FINITE STATE MACHINE SYNTHESIS WITH CONCURRENT ERROR DETECTION

Chaohuang Zeng, Nirmal Saxena and Edward J. McCluskey

Center for Reliable Computing
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305

ABSTRACT

A new synthesis technique for designing finite state machines with on-line parity checking is presented. The output logic and the next-state logic of the finite state machines are checked independently. By checking parity on the present state instead of the next state, this technique allows detection of errors in bistable elements (that were hitherto not detected by many previous techniques) while requiring no changes in the original machine specifications. This paper also examines design choices with respect to parity prediction circuits. Two such examined choices are the multi-parity-group and the single-parity-group techniques. A new state encoding technique based on the JEDI program is developed for the synthesis of the next-state logic with an additional parity output. Synthesis results produced by our proposed procedure for the MCNC'89 FSM benchmark circuits show on average a 25% reduction in literal counts compared to previous techniques.

1. INTRODUCTION

Concurrent error detection (CED) is very important in highly dependable computing systems. As technology advances to deep sub-micron levels, circuits are more susceptible to errors. CED techniques can be used to detect permanent and transient errors in these circuits. In this paper, we present a technique for synthesizing a totally self-checking finite state machine (FSM) with significantly lower area overhead than that of previous techniques. This technique can be easily extended to synthesize a general sequential circuit with concurrent error detection.

A general structure of concurrent error detection is shown in Fig. 1. The output of a circuit has a certain property that is monitored by a checker. If an error causes a violation of the property, the checker gives an error indication. An easy approach is to duplicate the original circuit and compare the outputs. However, duplication requires a significant amount of area overhead and does not provide protection against common mode failures [Mitra 99]. Several approaches have been taken in the past to design self-checking combinational and sequential circuits with lower area overhead than duplication.

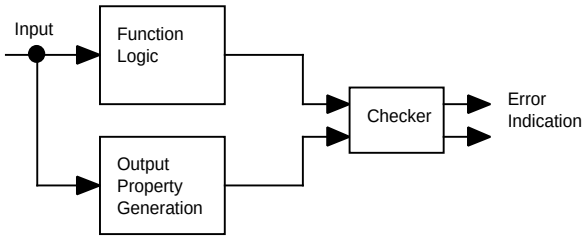

Figure 1. CED Circuit Structure

Concurrent error detection synthesis techniques that do not require structural constraints during logic optimization are reported in [Leveugle 90] [Robinson 92] [ElGuibaly 95] [Parekhji 95] [Bolchini 95a]. These techniques provide error detection for the next-state logic of an FSM by encoding the states with a special property. In [Parekhji 95], the next-state logic of the original machine is synthesized as usual after state assignment, and a monitoring machine is used to monitor the states of the original machine. The effectiveness of this technique is studied with respect to the delay fault model. However, for the stuck-at fault model, the large number of incompatibilities between the original machine and the monitoring machine could result in high area overhead [Parekhji 95]. Leveugle, *et al*. [Leveugle 90] use the control flow checking technique [Namjoo 82] to detect illegal state transitions of an FSM based on path signatures. When the FSM is running, the next state is compressed by a multiple-input signature register (MISR). When the machine reaches a certain state that is designated as a checkpoint, the concurrently calculated signature is compared with the reference signature. This technique has uncertain error detection latency and low fault coverage [Rochet 95]. In the worst case (when the state transition graph is fully connected), the fault cov-

erage is zero because every path is a legal path. A method similar to [Leveugle 90] with lower error detection latency was proposed in [Robinson 92]; however, the fault coverage is still low [Rochet 95].

There are design techniques for self-checking circuits that impose structure constraints during logic optimization and thereby limit the scope of fault propagation [Sagomonyan 74][Aksenova 75][Khodadad-Mostashiry 79] [Dwahan 88][Jha 93][De 94][Bolchini 95b][Touba 97][Das 98]. In [Jha 93] and [Bolchini 95b], the circuit is synthesized in such a way that all inverters are pushed to the primary inputs and the final circuit is inverter-free except at the primary inputs. The output is encoded with unidirectional error detection codes, such as the Berger code and the *m*-out-of-*n* code, and monitored by totally self-checking (TSC) checkers. All internal unidirectional stuck-at faults except those inside the flip-flops can cause only unidirectional errors at the output, which are always detected by TSC checkers. However, these techniques cannot guarantee to detect errors at the outputs of the flip-flops (the present state) caused by the unidirectional stuck-at faults inside the flip-flops because the logic implementation inside the flip-flops is not *inverter-free*.

De, *et al.* [De 94] proposed two techniques for synthesizing combinational multilevel circuits. One technique is based on the Berger code. The other technique, based on the parity code, partitions the combinational circuit into different logic blocks such that the logic sharing inside each block is maximized, but no logic sharing is allowed between logic blocks except at the primary input. Each block has its own parity output and the whole circuit could have several parity outputs. This multi-parity concept was proposed by Sogomonyan [Sogomonyan 74].

Touba and McCluskey [Touba 97] also proposed a technique for synthesizing the combinational multilevel circuits. This technique partitions the combinational circuits into multiple groups by adaptively choosing the optimal parity codes. For each group of the outputs, an additional parity output is generated and added to this group. Logic sharing is disallowed inside each group but allowed for different groups. Any single internal fault can cause only single error at the outputs of the same group. The error will always be detected because it changes the parity of the group.

Dwanhan and De Vries [Dwahan 88] proposed a technique to design a self-checking FSM. This technique assumes that the machine is completely specified (defined in Sec. 2) and requires the number of states equal to 2^n (n is the number of state bits). This stringent requirement makes this technique impractical for a large FSM. For example, the circuit *sand* from the MCNC89 benchmark circuits has 11 input bits and 9 state bits. If a fully specified machine is required, it needs 2^{20} specifications, while the original machine has only 104 specifications. The technique used in [Aksenova 75] has the same constraint because it only checks the primary output.

This paper describes an automatic synthesis procedure to design a self-checking finite state machine based on parity codes. Instead of using only one parity output [Dwahan 88], we use two (or more) parity outputs to monitor the output logic and the next-state logic independently to avoid the stringent conditions required by Dwahan's technique. By checking the present state instead of the next state, this new scheme can detect errors in the bistable elements in addition to the output errors. The main contributions of this paper are:

1. A novel totally self-checking FSM structure is presented. This structure does not require any change in the original machine specification. It also detects errors in the bistable elements.
2. This paper examines various design choices with respect to parity prediction circuits. In particular, two such choices are the multiple-parity-group technique and the single-parity-group technique (defined in Sec. 2).
3. A new state encoding technique based on the JEDI program [Lin 89] is used to synthesize the next-state logic with an additional parity output.
4. Synthesis results on the MCNC'89 FSM benchmark circuits are presented.

The paper is organized as follows. Section 2 introduces some terminology and concepts. Section 3 describes the structure of our self-checking finite state machine. Section 4 presents three different methods of synthesizing the parity output of the next-state logic and describes a new state encoding technique. In Sec. 5, the results generated by our synthesis procedure are presented and compared with previous techniques. A summary and conclusions are presented in Sec. 6.

2. PRELIMINARIES

A *parity group* consists of a subset of the circuit outputs and an additional output that equals the parity of these outputs. If two outputs (including the parity output) belong to the same parity group, no logic sharing is allowed between them. If the circuit is synthesized such that all the circuit outputs belong to the same parity group, we define it as the *single-parity-group* technique. If the outputs are partitioned into several different parity groups, we define it as the *multiple-parity-group* technique.

A finite state machine can be represented by its state table or its state diagram [McCluskey 86]. Suppose an FSM has n primary inputs, there could be 2^n state transitions from each state. If all these transitions and the outputs corresponding to the transitions are specified for each state, the machine is called a *completely specified ma-*

chine. If at least one transition or one output corresponding to some transition is not defined, the machine is called an *incompletely specified machine*.

In the following, we define a totally self-checking sequential circuit because it is very important for a dependable computing system.

DEFINITION 1: A sequential circuit is *fault secure* if, for every fault from a prescribed fault set, the circuit never produces an incorrect code space output for normal inputs [Wakerly 78].

DEFINITION 2: A sequential circuit is *self-testing* if, for every fault from a prescribed fault set, the circuit produces a non-code space output for at least one input received in normal operation [Wakerly 78].

DEFINITION 3: A sequential circuit is *totally self-checking* (TSC) if it is fault secure and self-testing.

In this paper, sequential circuits are synthesized to be TSC and then checked by TSC checkers [Khakbaz 84] [Hughes 84].

3. TOTALLY SELF-CHECKING FINITE STATE MACHINE

The general form of an FSM is shown in Fig. 2. The machine consists of output logic that produces the primary output, next-state logic that produces the next state, and bistable elements (such as flip-flops or latches). In this section, we explain the single-parity-group technique and the multiple-parity-group technique in more detail. Then we present our technique to design a TSC FSM.

3.1 Single-parity-group technique

This technique has been used to synthesize combinational circuits [De 94][Khodadad-Mostashiry 79] and sequential circuits [Dwahan 88][Devadas 89]. Since all of the outputs belong to the same parity group, each output is synthesized independently and thus there is no logic sharing between any pair of the outputs. Consequently, faults inside each output block (Fig. 3) can cause at most one output error. Figure 3 illustrates the logic implementation of the primary output and the parity output. A TSC parity checker [Khakbaz 84] is used to detect output error concurrently.

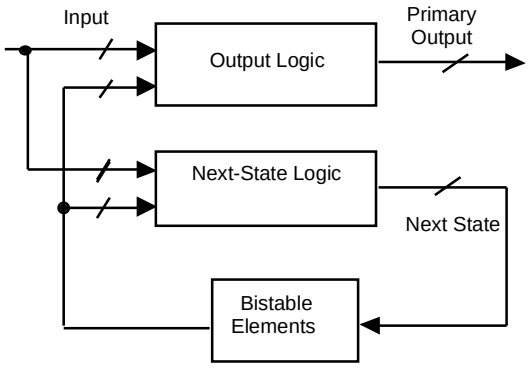

Figure 2. Diagram of a Finite State Machine

3.2 Multiple-parity-group technique

The single-parity-group technique allows no logic sharing among different outputs, resulting in large area overhead for some circuits. The multiple-parity-group technique was used [De 94][Touba 97] to allow logic sharing among some outputs. It partitions the primary outputs into different parity groups. Logic sharing is allowed only among outputs that are in different parity groups. The outputs within the same parity group are checked by a TSC parity checker. The outputs of the parity checkers are checked by a TSC two-rail checker [Hughes 84], which gives the final error indication (Fig. 4). This technique was used to synthesize combinational circuits.

There is an area trade-off in determining the number of parity groups. While more parity groups allow more logic sharing, they also increase the number of parity outputs. Finding an optimum number of parity groups is difficult due to the heuristic nature of the logic optimization algorithms. Nur Touba [Touba 97] developed a heuristic algorithm to find a partitioning of the primary outputs into several parity groups, where logic sharing between different parity groups are allowed. In this paper, we extend Touba's technique to synthesize an FSM and call it the multiple-parity-group technique.

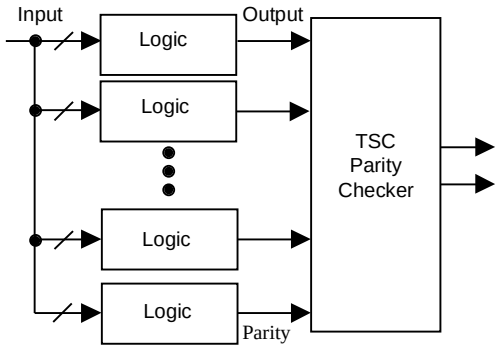

Figure 3. Single-parity-group Technique for Combinational Circuit

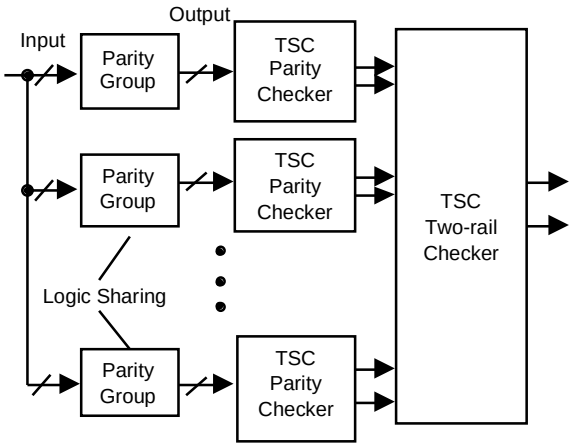

Figure 4. Multiple-parity-group Technique for Combinational Circuit

3.3 Totally Self-checking FSM

The new totally self-checking structure of an FSM is shown in Fig. 5. The output logic and the next-state logic are checked independently by TSC checkers (Fig. 5a). The outputs of the TSC checkers are checked by a two-rail TSC checker that gives the final error indication (Fig. 5b). The output logic of the FSM can be synthesized using the single-parity-group technique or the multiple-parity-group technique, depending on the circuit size. If it is the single-parity-group technique, the corresponding checker (TSC 1) is a TSC parity checker. If it is the multi-parity-group technique, the checker is the same as that shown in Fig. 4, consisting of TSC parity checkers followed by a two-rail checker. For the next-state logic, we always use the single-parity-group technique because the next-state logic usually does not have much logic sharing. It has much less area overhead than the multiple-parity-group technique based on our experimental results described in Sec. 5. The corresponding checker is a TSC parity checker, which is placed after the outputs of flip-flops to check the parity of the present state. It should be noted that one additional flip-flop is needed to hold the value of the parity output of the next-state logic. However, the parity bit is not used as an input of the FSM.

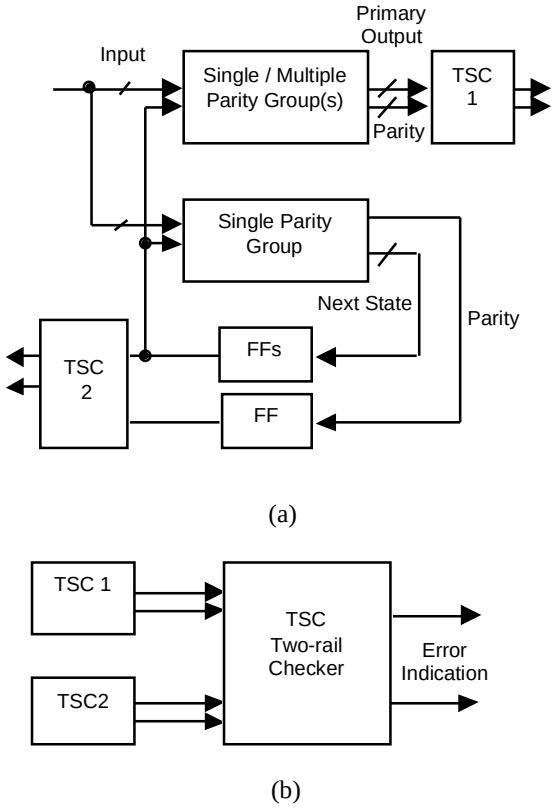

Figure 5. Self-checking Finite State Machine

In this scheme, only one additional flip-flop is required because the next-state logic has only one parity bit. In contrast, duplication would require as many additional flip-flops as the original circuit and a bigger checker circuit. Because the output logic and the next-state logic belong to different parity groups, logic sharing between them is allowed to further reduce the area overhead.

THEOREM: An FSM is totally self-checking with respect to single stuck-at faults if it is synthesized as described above (Fig. 5) and is irredundant.

PROOF: (1) the circuit is fault-secure. Suppose there is a stuck-at fault inside the output logic or the next-state logic. This fault could cause errors in several outputs. However, These erroneous outputs must belong to distinct parity groups because there is no logic sharing inside the same parity group. Therefore, if the error occurs in some or several outputs, it must change the parity of the groups to which these outputs belongs. If the parity groups are in the output logic, the error can be detected by TSC checker 1 within the same clock cycle. If the parity group is in the

next-state logic, the logic values of the next-state outputs and the corresponding parity output will be held by the flip-flops. In the next clock cycle, the error will be detected by TSC checker 2. Consider a stuck-at fault in one of the flip-flops. If the fault causes an error in the output of the flip-flop, the parity of the present state is changed and the error is then detected by TSC checker 2. If at least one of the TSC checkers in Fig. 5a detects the errors, the two-rail checker gives the final error indication (Fig. 5b).

(2) The circuit is self-testing. For any single stuck-at fault, there exists a pair of the primary inputs and the present state to activate the fault and produce an error in the output because the circuit is irredundant. One of the fundamental assumptions of the TSC circuits [Smith 78] is that faults occur one at a time and between any two faults a sufficient time elapses so that all the specified inputs are applied to the circuit. Based on this assumption, the fault can always manifest itself as output error(s), which are detected by the TSC checkers shown in Fig. 5.

The following is a simple mod-3 counter used to illustrate the new synthesis technique. The state table of the counter is shown in Table 1. The state is encoded with the binary vector in the parenthesis (Table 1). For example, S0 is encoded with 00. The circuit synthesized by our procedure is shown in Fig. 6. This circuit has two parity groups, one for the next-state logic and another for the output logic. The parity for each group is odd. There is logic sharing (the gate Y1 in Fig. 6) between the two parity groups but no sharing in the same parity group.

Many previous techniques [Jha 93][Bolchini 95b] check the next state instead of the present state. Consequently, these techniques cannot guarantee to detect errors in the flip-flops because the incorrect present state could result in an incorrect codeword output. We illustrate this problem using the above example.

Table 1. State Table of the Counter

Input	Present state	Next state Y1 Y2	Output Z1 Z2
-	S0 (00)	S1	00
-	S1 (01)	S2	01
-	S2 (11)	S0	10

Suppose the present state is S0 (encoded with 00) and the output of the flip-flop (FF2) is stuck at 0. The next state will be S1 (encoded with 01). After one clock cycle, the present state will unfortunately still stay at S0 due to the stuck-at fault. Therefore, the machine always stays at state S0. If the checker is used to monitor the next state, the error can not be detected. Dwahan's technique [Dwahan 88] circumvents this problem by requiring the FSM to be a completely specified machine. This solution becomes impractical when the machine becomes large because the actual machine specifications are much smaller than the complete specifications.

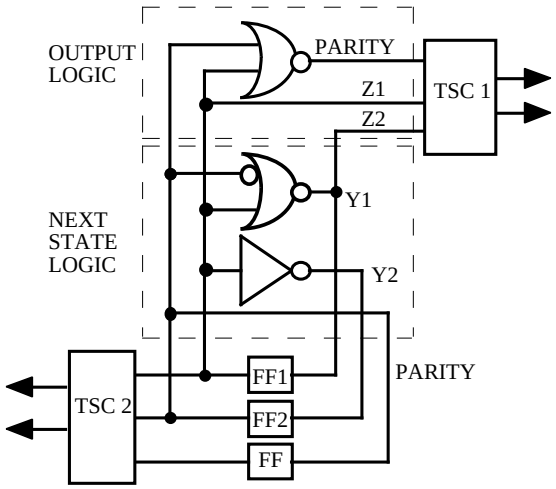

Figure 6. Self-checking Counter Circuit

To overcome this problem, our technique checks the present state directly so that the error propagation from the present state to the output is not necessary. The machine can be synthesized without any change to the original specifications.

4. PARITY OUTPUT GENERATION FOR THE NEXT STATE

The three basic steps in synthesizing an FSM are: (1) minimizing the number of machine states; (2) encoding the state symbols with binary vectors; and (3) optimizing the logic implementation. In an FSM, the next-state logic differs from the output logic in that the states are represented by symbols. Therefore, for the next-state logic, the parity output can be added at three different stages in the above synthesis flow:

1. During state assignment (at step 2). At this stage, each state is first encoded with binary vector of even (odd) parity and then one bit in the binary vector is designated as a parity output. This method tries to find such an encoding that will optimize not only the next state but also the parity output.
2. After state assignment but before logic optimization (between step 2 and step 3). At this stage, states are already encoded with binary vectors. Thus, the parity output can be calculated straightforwardly by taking bit-wise exclusive-or operation on the binary vector.

3. During logic optimization (at step 3). The next-state logic is optimized first. Then the parity output is inserted and optimized.

In the third option, the parity output is optimized after the next-state logic. Therefore, the parity output takes less advantage of the input and output *don't care* sets than the next state during the process of the logic optimization. Our results also indicate that the third option has a higher area overhead than the first two.

In the rest of this section, we discuss a new state encoding technique that extends previous work [Lin 89]. The objective of this encoding technique is to reduce the area overhead of the self-checking FSM.

We first describe briefly the normal state encoding technique implemented in the JEDI program [Lin 89]. State assignment can be performed for either the present state (present-state encoding) or the next state (next-state encoding). In JEDI program, there are two cost functions associated with them respectively. The JEDI program assigns binary vectors to the state symbols based on minimization of either one of the cost functions, depending on the user's choice. The default option is to minimize the cost function for the next-state encoding.

The new state encoding is performed as follows.

First, the states are encoded with odd-parity binary vectors by modifying the JEDI program. In the example shown in the previous section, states S0, S1, and S2 are encoded with 001, 010, and 111, respectively. We modify the program to encode the states with the binary vectors of odd parity such that the cost function for the next-state encoding is minimized. This step is to optimize both the next state and the parity output because they are indistinguishable at this point.

Second, one bit in the binary vector is designated as the parity bit. Every bit in the binary vector can potentially be a parity bit. Because the parity bit of the present state is not an input to the FSM, we want to choose such a bit in the binary vector that the area of the FSM is minimal. Heuristically, we choose the parity bit in such a way that, after removing it from the binary vector, the cost function for the present-state encoding is minimal, which means that this bit contributes less logic sharing than the other bits. For example, bit 3 is designated as the parity bit in the above example.

Third, the designated parity bit is removed from the binary vector and the resulting binary vector is assigned to the state. For example, after the third bit is removed, states S0, S1, and S2 are then encoded with 00, 01, and 11, respectively. The new state assignment after the bit removal is valid because each state symbol is still encoded with a unique binary vector. This step is necessary because the present state excludes the parity bit in order to reduce the area overhead (Fig. 5).

5. EXPERIMENTAL RESULTS

The self-checking FSM synthesis technique proposed in this paper has been implemented by modifying SIS 1.2 [Sentovich 92]. The state assignment program (JEDI) is modified to add the capability of the special state encoding described in Sec. 4. The logic optimization technique that has structure constraints on the circuit (such as no logic sharing among a parity group) [Touba 97] is extended to sequential circuits. The MCNC89 FSM benchmark circuits are synthesized by our procedure to be totally self-checking circuits.

Table 2 compares the literal counts of two techniques: the single-parity-group technique and the multiple-parity-group technique. This comparison is for both the output logic and the next-state logic (the parity is added during the state assignment as described in Sec. 4). The literal counts include those of the TSC checkers. For the output logic, the single-parity-group technique is generally better for circuits with a small number of primary outputs and primary inputs (<10). However, the multiple-parity-group technique is generally better for circuits with a large number of primary outputs (>15), such as *ex1* and *scf*. Therefore, in our synthesis procedure for a finite state machine, we select the multi-parity-group technique or the single-parity technique depending on whether the number of the primary outputs is larger than 15 or not. If the synthesis time is not a critical factor, we can synthesize the machine with these two different techniques and then choose the best result. For the next-state logic, the single-parity-group technique has about 15% percent lower literal counts than the multiple-parity-group technique. The single-parity-group technique also has fewer flip-flops than the multiple-parity-group technique. Therefore, we use the single-parity-group technique to synthesize the next-state logic of FSM.

Table 3 compares the literal counts of the next-state logic generated by three different methods described in Section 4. The literal counts of the TSC checker are not included because they are the same for these three methods. The third column shows the results when the parity output is added during the logic optimization. The fourth column shows the results when the parity output is added after the state assignment but before logic optimization. The last column shows the results when the parity is added during the state assignment by the new state encoding technique. The last two columns have generally better results than the third column because the parity output is generated before logic optimization, taking greater advantage of the input and output *don't care* sets. The new state encoding technique gives better results in most of the benchmark circuits. However, the normal state encoding

technique occasionally has better results because of the heuristic nature of state assignment and logic optimization.

Table 2. Two Different Techniques, Literal Counts Including TSC checkers

Benchmark Circuits	Inputs	Outputs	Output Logic of the FSM (*l.c.*)			Next-state Logic of the FSM (*l.c.*)		
			Original	Multiple	Single	Original	Multiple	Single
dk14	3	5	72	161	141	53	114	106
dk16	2	3	54	102	102	195	434	324
ex1	9	19	155	429	521	133	283	253
ex2	2	2	15	38	19	59	154	114
ex4	6	9	41	108	100	37	80	68
ex6	5	8	74	182	191	48	87	87
kirkman	12	6	146	350	619	49	117	116
opus	5	6	51	141	102	42	103	93
planet	7	19	310	772	744	242	561	397
pma	8	8	89	188	160	143	334	275
scf	27	56	292	812	1003	515	831	754
sse	7	7	58	145	150	87	160	139
styr	9	10	252	560	528	313	640	482
tma	7	6	93	234	201	108	258	222

Table 4 shows the results for four different implementations: duplicate-and-compare, [Das 98], [Jha 93], and the proposed technique. "NA" in the table means the data is not available. It should be noted that the techniques [Jha 93] target the unidirectional fault model. The proposed technique assumes the single stuck-at model. The comparison to [Jha 93] is valid only when the single stuck-at fault model is adequate for a given circuit. In fact, the proposed technique can also detect multiple faults inside any single output block shown in Fig. 3. Moreover, compared to previous techniques our technique has on average 25% lower literal counts. In addition, our approach has fewer flip-flops than the other three techniques.

Table 3. Different Parity-bit Generation Stages (Literal Counts, Not Including the TSC Checkers)

circuits	Next-state logic of FSM			
	Original	During logic optimization	After state assignment	During state Assign-ment
dk14	53	102	**93**	97
dk16	195	312	**270**	295
ex1	133	241	184	**180**
ex2	59	106	127	**101**
ex4	37	60	64	**56**
ex6	48	83	111	**73**
ex7	13	26	25	**21**
kirkman	49	104	110	**74**
mark1	52	72	72	**59**
opus	42	85	83	**61**
planet	242	381	412	**368**
pma	143	263	267	**236**
s27	24	61	**47**	50
sand	325	475	**450**	476
scf	515	734	**696**	708
sse	87	131	121	**115**
styr	313	**470**	505	478
sbk	170	**365**	435	376
tma	108	210	168	**147**

7. SUMMARY AND CONCLUSIONS

This paper presents a novel self-checking FSM design and synthesis technique. This technique can detect any internal single stuck-at fault with at most one clock cycle latency. By checking the present state instead of the next state, this new scheme can detect any single error in the bistable elements without changing the machine specifications.

Specific to the next-state logic, the parity output can be added at three different stages in the synthesis flow: 1) during logic optimization, 2) after state assignment but before logic optimization, and 3) during state assignment. We present the results for these three different cases. The last case with the new special state encoding generally produces the better synthesis results.

The single-parity-group technique is compared with the multiple-parity-group technique. For the next-state logic, the single-parity-group technique is much better

than the multiple-parity-group technique because there is not much logic sharing in the next-state logic. For the output logic, the single-parity-group technique usually has better results for circuits with a small number of outputs, while the multiple-parity-group technique is in general better for circuits with a large number of outputs. Our synthesis results show that our self-checking FSM has significantly lower literal counts than the previous techniques for most of the MCNC89 benchmark circuits.

Table 4. Proposed Technique versus Previous Techniques (Literal Counts)

Circuits	Original	Duplication	[Das 98]	[Jha 93]	Proposed Technique
cse	183	402	NA	446	**377**
dk14	101	226	281	316	**214**
dk15	69	158	196	230	**137**
dk16	239	502	559	445	**417**
dk17	57	130	NA	NA	**114**
dk27	24	60	NA	NA	**53**
dk512	56	132	NA	NA	**106**
ex1	221	**530**	NA	777	682
ex2	74	164	NA	NA	**133**
ex3	25	62	NA	NA	**51**
ex4	70	184	NA	310	**159**
ex5	12	32	NA	NA	**28**
ex6	78	**192**	NA	298	248
ex7	22	52	NA	NA	**42**
kirkman	186	**408**	NA	588	466
mark1	80	232	NA	407	**190**
mc	23	66	NA	135	**57**
opus	77	186	NA	227	**166**
planet	213	1098	1109	1016	**1010**
pma	237	470	NA	NA	**434**
s27	36	80	NA	NA	**74**
sand	476	1000	NA	818	**989**
scf	763	1770	NA	1876	**1185**
sse	120	286	NA	344	**283**
styr	468	988	1249	**789**	1066

NA: Not Available

ACKNOWLEDGEMENT

The authors would like to thank Professor Nur Touba for his valuable comments and Dr. Santiago Fernandez-Gomez for his help in implementation. This work was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. DABT63-97-c-0024.

REFERENCES

[Aksenova 75] Aksenova, G.P. and Sogomonyan, E.S., "Design of Self-checking Built-in Check Circuits for Automata with Memory", *Automation and Remote Control*, Vol. 36, pp. 1169-77, July 1975.

[Bolchini 95a] Bolchini, C. et al. "A State Encoding for Self-checking Finite State Machines" *Proceedings of VLSI95*, pp711-16, Aug. 1995.

[Bolchini 95b] Bolchini, C. and Sciuto, D., "An Output/state Encoding for Self-checking Finite State Machine," *Proceedings of International Symposium on Circuits and Systems*, Vol.3, pp. 2136-9, May 1995.

[Das 98] Das, D. and Touba, N.A., "Synthesis of Circuits with Low-Cost Concurrent Error Detection Based on Bose-lin Codes", *IEEE VLSI Test Symposium*, pp. 309-315, Apr. 98.

[Davadas 89] Devadas, S., Ma, H.-K.T., and Newton, A.R., "A Synthesis and Optimization Procedure for Fully and Easily Testable Sequential Machines", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.8, no.10, pp. 1100-7, Oct. 1989.

[De 94] De, K., Natarajan C., Nair D, and Banerjee, P., "RSYN: A System for Automated Synthesis of Reliable Multilevel Circuits," *IEEE Trans. Very Large Scale Integration*, Vol. 2, pp. 186-195, Jun. 1994.

[Dhawan 88] Dhawan, S. and De Vries, R.C., "Design of Self-checking Sequential Machines," *IEEE Trans. Computers*, Vol. 37, pp. 1280-1284, Oct. 1988.

[Hughes 84] Hughes, J.L.A., E.J.McCluskey, and D.J. Lu, "Design of Totally Self-checking Comparators with an Arbitrary Number of Inputs", *IEEE Trans. Computers*, pp 546-50, Jun.1984.

[Jha 93] Jha, N.K. and Wang, S.J., "Design and Synthesis of Self-checking VLSI Circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, pp. 878-887, June 1993.

[Khakbaz 82] Khakbaz, J., "Self-testing Embedded Parity Trees", *International Symposium on Fault-Tolerant Computing*, pp109-16, June 1982.

[Khodadad-Mostashiry 79] Khodadad-Mostashiry, B., "Parity Prediction in Combinational Circuits", *International Symposium on Fault-Tolerant Computing*, pp 185-8, Jun. 1979.

[Leveugle 90] Leveugle, R. and Saucier, G., "Optimized Synthesis of Concurrently Checked Controller," *IEEE Trans. Computers*, Vol. 39, pp. 419-425, Apr.1990.

[Lin 89] Lin, B. and Newton, A.R., "Synthesis of Multiple Level Logic from Symbolic High-level Description Language," *Proceedings of the International Conference on VLSI*, pp. 414-417, Aug. 1989.

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

[Mitra 99] Mitra, S., Saxena, N. and McCluskey, E.J., “A Design Diversity Metric and Reliability Analysis for Redundant Systems”, *International Test Conference*, to appear in Sept. 1999.

[Namjoo 82] Namjoo, M., "Techniques for Concurrent Testing of VLSI Processor Operation" *International Test Conference*, pp461-8, Nov. 1982.

[Parekhji 95] Parekhji, R.A., Venkatesh, G. and Sherlekar, S.D., "Concurrent Error Detection using Monitoring Machines," *IEEE design & Test of Computers*, Vol. 12, pp.24-32, Fall 1995.

[Robinson 92] Robinson, S.H. and Shen, J.P., "Direct Methods for Synthesis of Self-monitoring State Machines," *International Symposium on Fault-Tolerant Computing*, pp306-315, Jul. 1992.

[Rochet 95] Rochet, R., Leveugle, R. and Saucier, G., "Efficiency Comparison of Signature Monitoring Schemes for FSMs," *proceedings of VLSI95*, pp705-10, Aug. 1995.

[Sentovich 92] Ellen M. Sentovich, et. al.,”SIS: A System for Sequential Circuits Synthesis,” *Technical Report No. UCB/ERL M92/41, Department of EECS, U.C. Berkeley*, May 1992.

[Smith 78] Smith, J.E. and Metze, G, “Strongly Fault Secure Logic Network,” IEEE Trans. Computers, Vol. C-27, No. 6, pp. 491-499, Jun. 1978.

[Sogomonyan 74] Sogomonyan, E.S., "Design of Built-in Self-checking Monitoring Circuits for Combinational Devices", *Automation and Remote Control*, Vol. 35, pp. 280-9, Feb. 1974.

[Touba 97] Touba, N.A. and McCluskey, E.J., “Logic Synthesis of Multilevel Circuits with Concurrent Error Detection”, *IEEE Trans. Computer-Aided Design of Integrated Circuits and System*, Vol. 16, pp. 783-789, Jul. 1997.

[Wakerly 78] Wakerly, J., “Error Detecting Codes, Self-checking Circuits and Applications”, North Holland, 1978.