N94-18344

Fault Tolerant Sequential Circuits Using Sequence Invariant State Machines

M. Alahmad and S. Whitaker
NASA Space Engineering Research Center
for VLSI System Design
University of Idaho
Moscow, Idaho 83843

Abstract - **The idea of introducing redundancy to improve the reliability of digital systems originates from papers published in the 1950s. Since then, redundancy has been recognized as a realistic means for constructing reliable systems. This paper will introduce a method using redundancy to reconfigure the Sequence Invariant State Machine (SISM) to achieve fault tolerance. This new architecture is most useful in space applications, where recovery rather than replacement of faulty modules is the only means of maintenance.**

1 Introduction

Fault tolerance is essential feature for digital systems where reliability, availability and safety are of vital importance. Such systems include aerospace missions, where a recovery procedure must be employed as means of maintenance, rather than replacement procedures which would be impossible during such missions.

Most digital systems can be divided into two functional blocks: the controller and the data path. The controller is a sequential circuit that performs certain tasks based on external and internal information. A programmable hardware architecture has been developed that enables a controller's hardware to be designed without a knowledge of the exact sequence of the input data to be incorporated [1]. This programmable architecture is called a Sequence Invariant State Machine (SISM).

This paper will introduce a method to achieve fault tolerance in the SISM design using dynamic redundancy. With this method, faulty controllers can recover and resume operation. Two different architectures are proposed and analyzed in terms of transistor count, size and fault detection. One architecture is clearly superior to the other.

2 SISM Overview

With the SISM realization, any flow table can be implemented without a change in the hardware configuration. That is given the number of states m and the number of inputs n, a hardware circuit is easily derived, that can implement any sequence of states.

	I_1	I_2	I_3
A	C, 1	B, 1	A, 0
B	D, 0	C, 1	B, 0
C	E, 0	D, 0	C, 0
D	F, 1	E, 1	D, 1
E	A, 0	F, 0	E, 1
F	B, 0	A, 1	F, 1

Table 1: General 6-states, 3-input flow table.

y_1	y_2	y_3			I_1			I_2			I_3	
0	0	0	A	0	1	0, 1	0	0	1, 1	0	0	0, 0
0	0	1	B	0	1	1, 0	0	1	0, 1	0	0	1, 0
0	1	0	C	1	0	0, 0	0	1	1, 0	0	1	0, 0
0	1	1	D	1	0	1, 1	1	0	0, 1	0	1	1, 1
1	0	0	E	0	0	0, 0	1	0	1, 0	1	0	0, 1
1	0	1	F	0	0	1, 0	0	0	0, 1	1	0	1, 1
1	1	0	G	0	0	0, 0	0	0	0, 0	0	0	0, 0
1	1	1	H	0	0	0, 0	0	0	0, 0	0	0	0, 0

Table 2: State Assignment and Redundant States for Table 1.

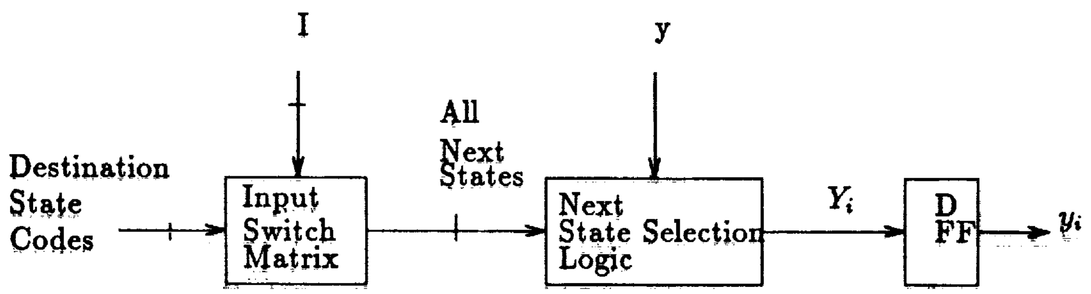

Figure 1: General SISM Architecture.

Table 1 shows a general 6 states, 3-input flow table. The state assignment for this table is shown in Table 2. Figure 1 shows the SISM architecture for one of the next state variables in Table 2. There are two identical architectures for the remaining two variables. Only the destination state codes are different. The Figure consist of the following components.

- The destination state codes which are derived from the next state entries in the state assignment table by inspection. For example, the destination state codes for state B and state variable y_i are the next state bits Y_i associated with state B. Therefore, the destination state codes for state B are $(000, 110, 101)$ under input states $(I_1; I_2; I_3)$ and variables $(y_1; y_2; y_3)$respectively.
- The input switch matrix which is combinational logic that produces all the possible next state entries for each current input state.
- The next state logic which consists of an independent path for each of the present states in the state assignment flow table.
- The storage element, a D-FF, that preserves the present state.

The current input state selects the set of potential next states that the circuit can assume (input column in the flow table). The present state variables select the exact next state (row in the flow table) that the circuit will assume at the next clock pulse.

3 SISM Implementation

Two pass transistor networks which make the SISM fault tolerant will next be discussed and compared in terms of space and the number of transistors. The input switch matrix is shown in both structures as a logic block, since it is identical in both designs.

3.1 FCS Design

A Fully Coded Structure (FCS) [4] network is defined as a fully specified pass network circuit. A knowledge about the number of next state variables is sufficient to achieve this design. Thus, the FCS is a design by inspection. Using Table 2 as a reference, three state variables are required to implement this table. Therefore, eight unique states can be represented. Each state will have an independent branch with all the variables as control terms. Those branches are all connected to the output pass function. Only one branch is activated by any combination of control variables at a given time, since each branch is encoded uniquely. The output pass function is the logical OR or the summation of all states. Figure 2 shows the complete FCS structure for the next state variable $Y1$ in Table 2. The other two variables have identical structure, but different destination state codes.

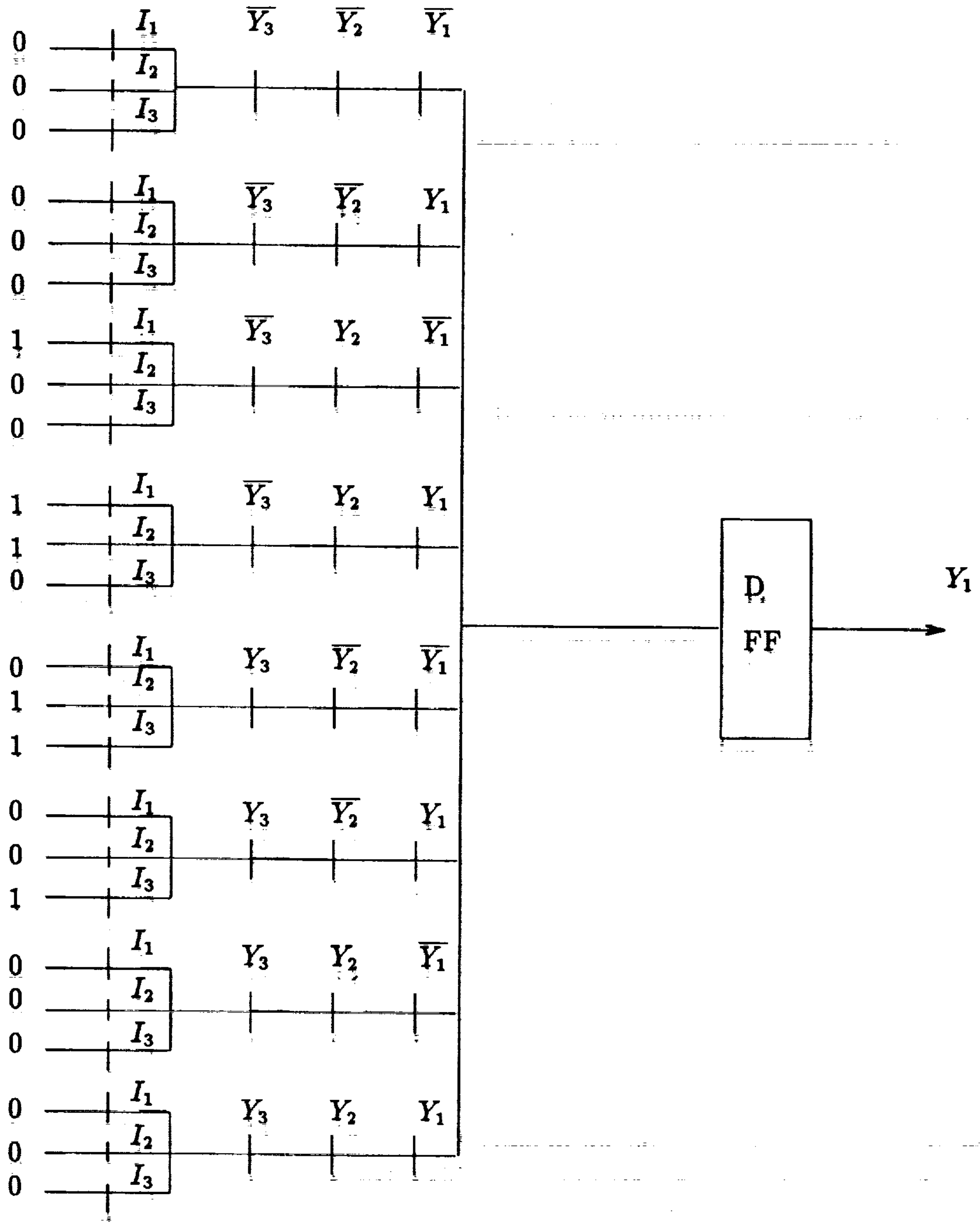
0 0 0
I_1 I_2 I_3 $\overline{Y_3}$ $\overline{Y_2}$ $\overline{Y_1}$
0 0 0
I_1 I_2 I_3 $\overline{Y_3}$ $\overline{Y_2}$ Y_1
1 0 0
I_1 I_2 I_3 $\overline{Y_3}$ Y_2 $\overline{Y_1}$
1 1 0
I_1 I_2 I_3 $\overline{Y_3}$ Y_2 Y_1
0 1 1
I_1 I_2 I_3 Y_3 $\overline{Y_2}$ $\overline{Y_1}$
0 0 1
I_1 I_2 I_3 Y_3 $\overline{Y_2}$ Y_1
0 0 0
I_1 I_2 I_3 Y_3 Y_2 $\overline{Y_1}$
0 0 0
I_1 I_2 I_3 Y_3 Y_2 Y_1
D FF
Y_1

Figure 2: Structure of the next state variable Y_1 using the FCS structure.

3.2 BTS Design

A Binary Tree Structure (BTS) [3] network is defined as a pass network in which exactly two branches join at every node and the control term of one branch is the complement of the control term of the other branch. Generally, each control term is a single control variable and the number of nodes exceeds one. A BTS network is constructed by partitioning each next state variable in a specific manner until all the variables have been partitioned. Therefore, a BTS network is also designed by inspection.

Consider the flow table shown in Table 1. Three variables are needed to implement this flow table. The procedure is general and can be applied to any state machine. Firstly, start with the output node and partition the variable Y_3 into two branches. One of the branches will have Y_3 as the control variable and the other branch will have $\overline{Y_3}$ as the control variable. Secondly, for each node at the end of each of the newly constructed branchs, construct two more branches for the control variable Y_2 and its complement. Thirdly, for each node at the end of the new branch, construct two branches for the variable Y_1 and its complement. With this step the design structure is completed. Figure 3 shows the complete BTS structure for the next state variable Y_1. The other two next state variables are identical in structure and only the destination state codes are different.

3.3 Comparison

The BTS and FCS structures both use pass transistor networks. The number of transistors in the BTS structure is less than the number of transistors in the FCS structure, since the BTS structure is partitioned around each control variable. In terms of space and size, the BTS would appear to require less space. However, using the SISM compiler developed by Buehler [2] to design the BTS structure, the space required for each design is basically the same. The extra space available in the BTS structure is difficult to utilize. Using the SISM compiler, a custom drawn SISM layout for one of the variables in Table 2, using both structures, is shown in Figures 4 and 5.

3.4 Destination State Codes Implementation

The destination state codes are all the inputs that must be fed to either the BTS or the FCS structure in order to implement a state table. The inputs can be driven in several ways. They could be directly connected to VDD/VSS or they could be driven by the output of a shift register. The input array could also be constructed as a programmable memory such as EPROM.

In order to achieve programmability in the SISM structure, the data must not be hard-wired. If data were implemented using VDD and VSS connections then, the programmable nature of this design is limited to single mask programmability. Using a shift register will achieve the programmability objectives. The shift register will, however, increase the size of the circuit. If the EPROM is implemented on the IC, the size of the controller will also increased but since an EPROM cell is considerably smaller than a D-FF, the size impact

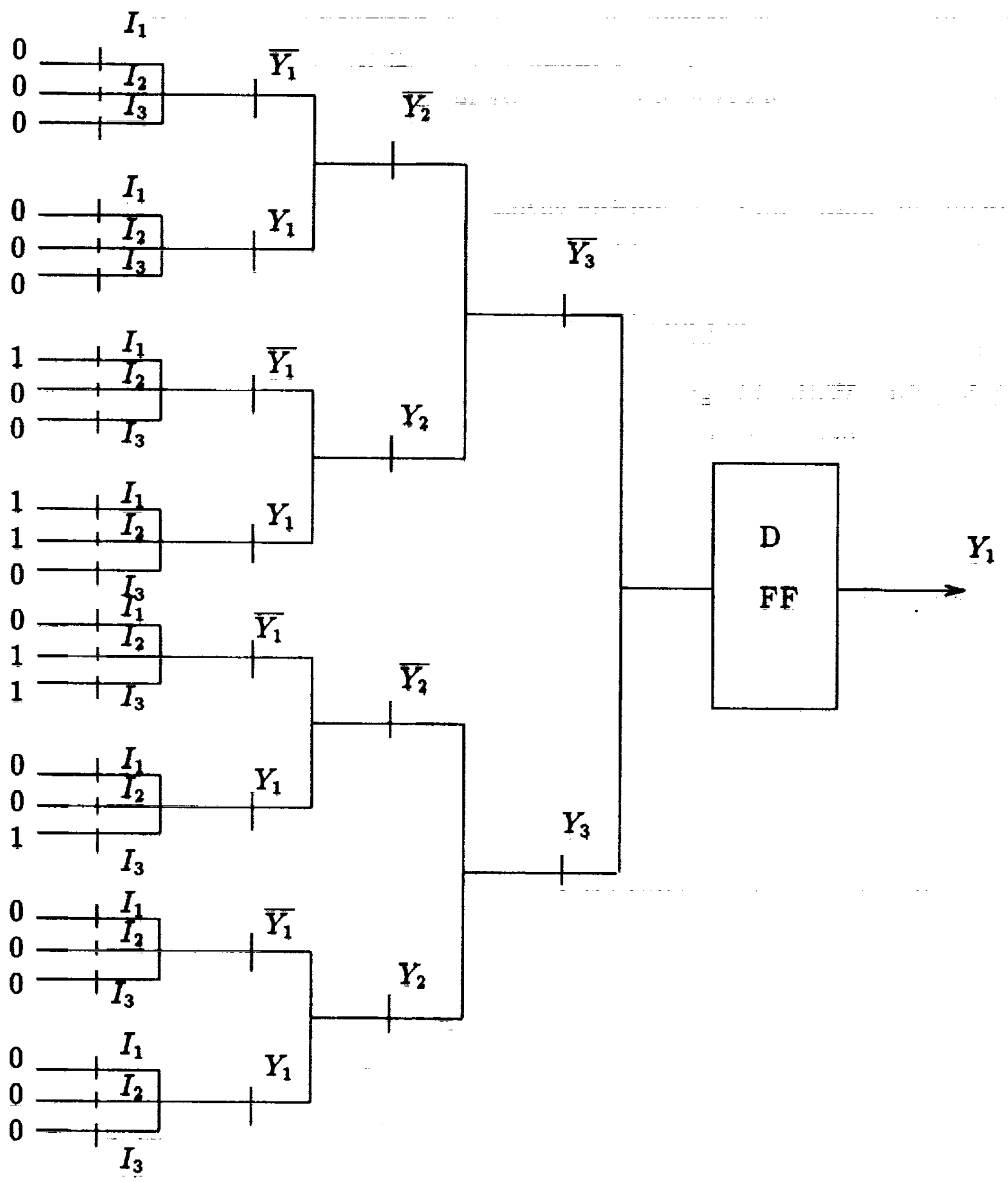

Figure 3: Complete structure for the variable Y_1 using the BTS architecture.

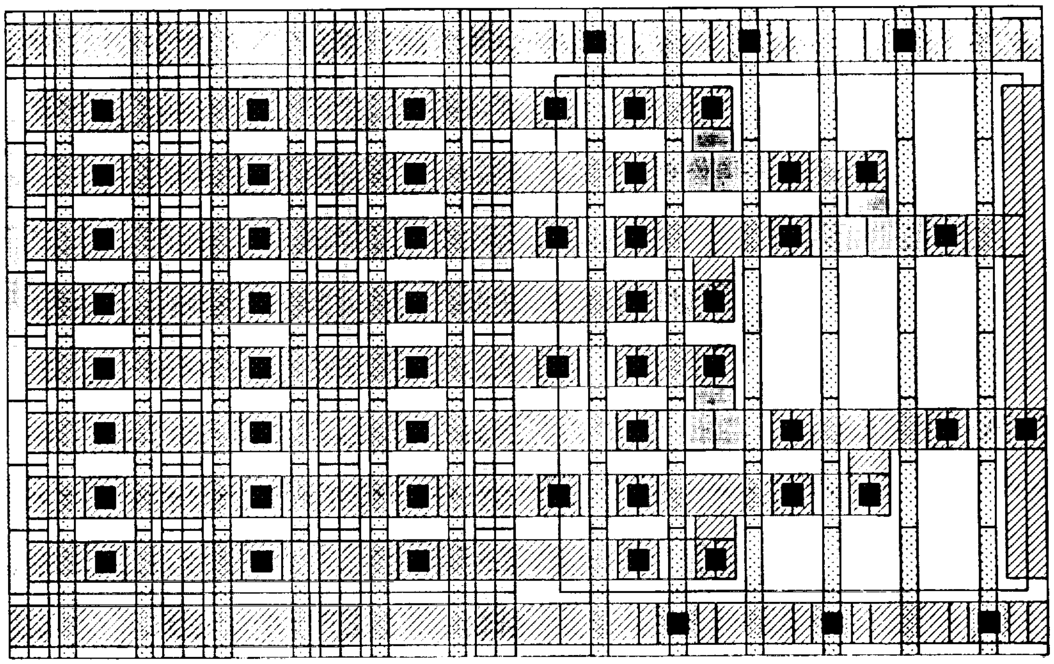

Figure 4: SISM layout using the BTS structure.

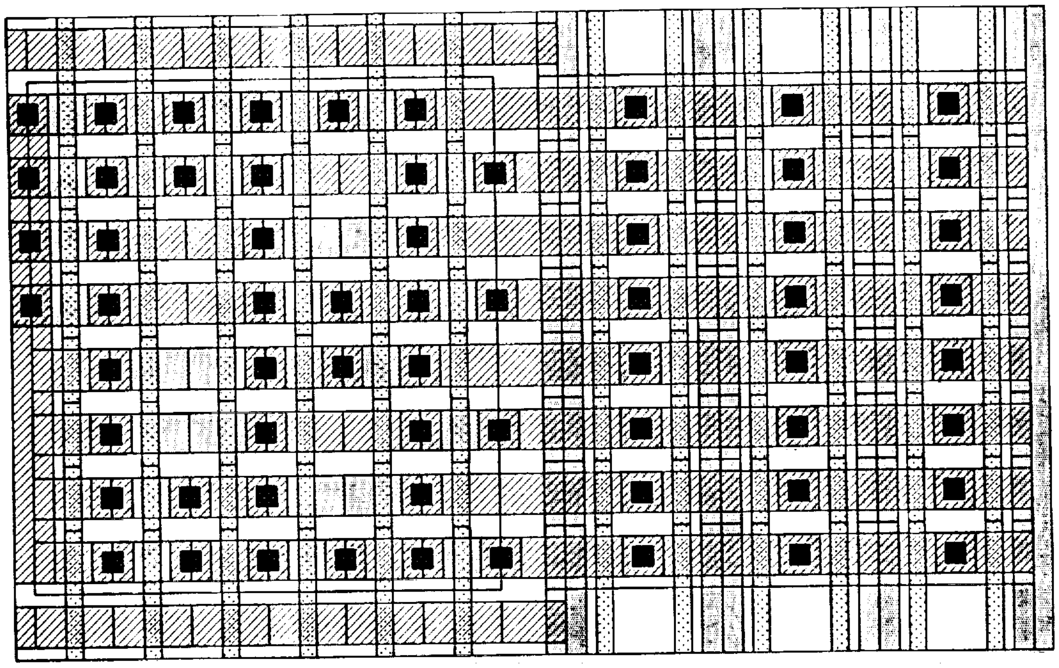

Figure 5: SISM layout using the FCS structure.

	I_1	I_2	I_3
A	C	B	A
B	D	C	B
C	E	D	C
D	F	E	D
E	A	F	E
F	B	A	F
G	A	A	A
H	A	A	A

Table 3: Fully specified flow table.

is much less than that for the shift register approach.

4 Achieving Fault Tolerance

In incorporating fault tolerance in any digital system, two approaches can be considered. The first approach is called static redundancy, also known as fault masking, which uses extra components such that the effect of a faulty component is masked instantaneously. The second approach is called dynamic redundancy, which has extra components but only one component operates at a time. If a fault is detected in the operating module, it is switched out and replaced by a spare. This dynamic redundancy requires consecutive actions of fault detection and fault recovery [5].

The idea of dynamic redundancy to achieve fault tolerance can be applied to the SISM structure. Hence, the operating module refers to all the paths (states) in the next state selection logic that construct the state machine. And the spare parts refer to the unutilized logic (redundant states) in the architecture. Therefore, if a fault has been detected in a given state (i.e. the path that identifies that state), a spare path is switched to replace the current path and correct operation is resumed.

Most state machines do not utilize all available states. Therefore, some of those states can be thought of as spare states and are redundant. To optimize the versatility and robustness of a controller, the redundant states can be used to replace any state which exhibits a malfunction. By applying a method for reconfigurability, the redundant states can be used to improve the reliability and to enhance the performance of an IC.

With reference to Table 1, there are six states, therefore three variables are needed to implement this flow table. With three variables, a maximum of eight states are available. Six of these states are used and two states are redundant. However, the next state entries for each of the two redundant states have been assigned the initial value (which is a safe output in all cases) as shown in Table 3, with the assumption that state A is the initial state. If state B tested faulty, then one of the redundant states, such as state G, could be used to replace state B to achieve correct operation.

Both the BTS and the FCS will have extra logic, and the reconfigurability method can be applied to use the extra logic. However, the location of a fault in the BTS can limit the use of the redundant logic and therefore decrease fault tolerance. That is, if a fault affects any of the transistors controlling Y_1 or its complement in Figure 3, then the method is valid and redundant logic can be used to replace that faulty branch. However, if a fault affects Y_3 or its complement then, there is not enough redundant logic to replace the entire faulty section. Therefore, the redundant logic has limited capabilities in the BTS structure. An identical structure can be added, but in doing so static redundancy can be achieved easily, at the cost of increasing the structure size by a factor of two.

The FCS structure, possesses a good structure. If any s-a-fault or s-op faults occur at the input or in the structure, then only one path (state) is effected. However, if a stuck-on faults occur in the structure, then two paths (states) will be affected at most. For example if a stuck at fault affects state B, then only the path that represents state B is affected and can be replaced. However, if a stuck-on affects state B, then two paths will be enabled at the same time. Therefore, the redundant logic can be used to replace this malfunction state. Hence the FCS structure is more applicable if dynamic redundancy is to be used.

Furthermore, the redundant logic in the FCS structure does not mask any of the faults that could occur in the structure. The reason being that the redundant logic does not replicate any of the existing states. Therefore, a fault in the structure or even in the redundant logic itself is testable.

5 Design Procedure

If any path in the FCS architecture becomes faulty due to the input being stuck at 1 or stuck at 0, a stuck open or shorted pass transistor, or any other malfunction, then the entire path is no longer correct and therefore must be replaced or recovered. To achieve fault tolerance, three methods must be used. They are error detection, fault location, followed by replacement and recovery. The primary concern is with the replacement and recovery technique. Once the designer has concluded that an error has occurred in a part of the IC, fault detection and location techniques are then applied to detect and locate the faulty part. If the faulty part is in the controller section of the circuit, then it must be determined where the fault has occurred, and the kind of fault that occurred.

Referring to Table 3, assume that the fault diagnosis has shown that state B is a faulty state. This corresponds to the path $(\overline{Y_3}; \overline{Y_2}; Y_1)$ in Figure 2, then the following steps are applied.

STEP1

Examine the flow table at hand and determine which of the redundant states will be used to replace state B. Since this flow table has two redundant states, State G is chosen. State H could have just as validly been chosen, but for simplicity the next state in order was chosen. Hence state G, $(Y_3; Y_2; \overline{Y_1})$ is chosen to replace state B.

STEP2

Modify the flow table to reflect the new changes. That is scan the flow table and replace each next state entry of B with the new state G. Therefore, every where in the next state

	I_1	I_2	I_3
A	C	G	A
B	D	C	G
C	E	D	C
D	F	E	D
E	A	F	E
F	G	A	F
G	A	A	A
H	A	A	A

Table 4: Second step in the replacement procedure.

	I_1	I_2	I_3
A	C	G	A
B	D	C	G
C	E	D	C
D	F	E	D
E	A	F	E
F	G	A	F
G	D	C	G
H	A	A	A

Table 5: Third step in the replacement procedure.

entry of the state table, replace B with a G. Table 4 reflects this replacement process.

STEP3

Fill the next state entry of state G with the same next state entry as that of state B. That is the next state entries for G will be the same next state entries for B providing that step2 was completed. Table 5 shows the result of this step.

STEP4

The next state entries for state B are modified in such a way that masks the kind of permanent fault in the hardware.

1. If a stuck at fault, s-op or s-on faults occur at the input of the destination state codes or in the input switch matrix or a s-a-1 or s-a-0 fault on the destination codes, then disabling the B state is sufficient.

2. If a s-op is occurred in any of the variables, then the path is already disabled.

3. If a s-on fault occurs in any of the variables, then the destination state codes to the faulty path must be identical to those of the new path the fault assumes. That is, if the variable $\overline{Y_3}$ in state B is stuck on, then this state becomes $(1;\overline{Y_2};\ Y_1)$ which is the same as state F. Therefore, the next state entries of state B must be the same as that of state F. Hence, when state F is enabled, state B is also enabled. To achieve

	I_1	I_2	I_3
A	C	G	A
B	G	A	F
C	E	D	C
D	F	E	D
E	A	F	E
F	G	A	F
G	D	C	G
H	A	A	A

Table 6: modified flow table.

y_1	y_2	y_3		I_1			I_2			I_3		
0	0	0	A	0	1	0	1	1	0	0	0	0
0	0	1	B	1	1	0	0	0	0	1	0	1
0	1	0	C	1	0	0	0	1	1	0	1	0
0	1	1	D	1	0	1	1	0	0	0	1	1
1	0	0	E	0	0	0	1	0	1	1	0	0
1	0	1	F	1	1	0	0	0	0	1	0	1
1	1	0	G	0	1	1	0	1	0	1	1	0
1	1	1	H	0	0	0	0	0	0	0	0	0

Table 7: Modified flow table.

correct operation, both states must have the same next state entries. As a result, the fault is masked. Table 6 shows the resulting flow table.

STEP5

The new state assignment is then reflected in the modified flow table. Table 7 shows the state assignment and the next state entries assignment.

STEP6

The destination state codes derived from the modified flow table determine the new data entries for the shift register.

With the completion of Step6, the operation of the circuit can be resumed with the same expected results.

Two final points are worth discussing. Firstly, if the state machine does utilize all of its states then an additional state variable must be added to allow this procedure to be employed. In order to demonstrate the procedure, the flow table shown in Table 8 is considered. As can be seen there are no extra states. Therefore, a new state variable is added and then the state assignment is revisited during the initial design to achieve redundancy. The next state equations and the hardware implementation will reflect this modification. The modified flow table is shown in Table 9.

Secondly, this method can be extended to achieve fault tolerance in the remaining parts of the circuit. This would be achieved by determining the faulty part and reconfiguring

Y_1	Y_2		I_1	I_2
0	0	A	D	B
0	1	B	C	C
1	0	C	A	D
1	1	D	B	A

Table 8: General 4-states, 2-input flow table.

Y_1	Y_2	Y_3		I_1	I_2
0	0	0	A	D	B
0	0	1	B	C	C
0	1	0	C	A	D
0	1	1	D	B	A
1	0	0	E	A	A
1	0	1	F	A	A
1	1	0	G	A	A
1	1	1	H	A	A

Table 9: Modified flow table.

the state machine in such a way as not to enable the faulty part, and to activate another part to replace it.

References

[1] S. Whitaker, S. Manjunath and G. Maki, "Sequence Invariant State Machines", *IEEE Journal of Solid State Circuits*, Vol. SC-26, Aug. 1991, pp .

[2] David M. Buehler, "Sequence Invariant State Machine Compiler", Master Thesis, Dept. of Elect Engr., University of Idaho, Moscow, Idaho, Dec. 1990.

[3] G. Peterson and G. Maki, "Binary Tree Structured Logic Circuits: Design and Fault Detection", *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers*, Port Chester, NY, Oct., 1984, pp. 671-676.

[4] D. Radhakrishnan and G. Maki, *Digital Systems Design*, EE 440 Lecture Notes, University of Idaho, Fall 1989.

[5] Parag K. Lala, *Fault Tolerant & Fault Testable Hardware Design*, Prentice-Hall International, Inc., London 1985.