Reliable VLSI Sequential Controllers

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

Abstract – **A VLSI architecture for synchronous sequential controllers is presented that has attractive qualities for producing reliable circuits. In these circuits, one hardware implementation can realize any flow table with a maximum of 2^n internal states and m inputs. Also all design equations are identical. A real time fault detection means is presented along with a strategy for verifying the correctness of the checking hardware. This self check feature can be employed with no increase in hardware. The architecture can be modified to achieve fail safe designs. With no increase in hardware, an adaptable circuit can be realized that allows replacement of faulty transitions with fault free transitions.**

1 Introduction

This paper presents a VLSI architecture for controllers that provides real time fault detection and reliability enhancements. The reliability advances are based on a new VLSI architecture for synchronous sequential circuits. This controller design supports important features such as real time fault detection, fail safeness and fault tolerance. The class of faults that is covered by this design are stuck-at, stuck-open and stuck-on. Controllers on two full custom VLSI data compression chips for NASA have been implemented using this architecture [1].

Most digital systems include a controller. This can be either a general machine such as a microprocessor, or a dedicated, custom designed sequential state machine. Dedicated controllers can be implemented as programmable PLA based structures, or as a random logic designs. The realization of state machines based on random logic often results in the most compact and highest performance circuits, but the logic is a function of the state assignment, flip flop type and flow table (actual sequence). Controllers can also be implemented in PLA structures, reducing the layout effort, but are less area efficient and have reduced system performance. PLA based controllers can be reconfigured to some extent but the reconfigurability is limited by the number of minterms available in the PLA.

An architecture that retains the traditional strengths of dedicated state machines, but offers the programmability of a microcontroller was presented in [2]. This architecture produces controllers whose logic is invariant with respect to the actual sequence desired. State machines designed using this method approach the performance and size of random logic based state machines and have a programmability superior to a PLA based design.

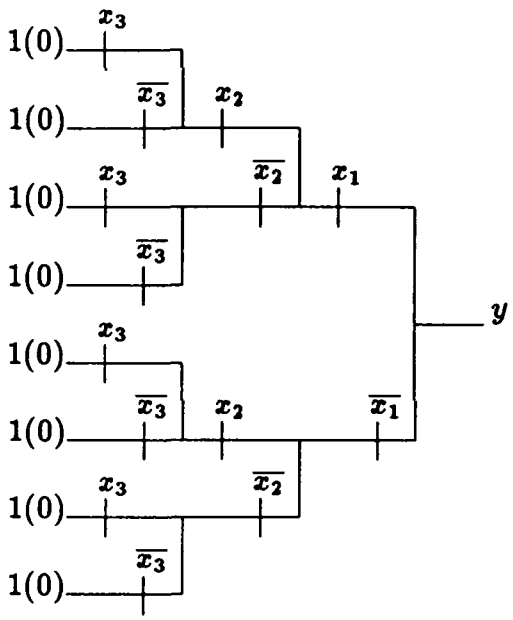

Figure 1: General three-variable BTS network.

2 Binary Tree Structured Logic

Pass transistor logic can have significant advantages in speed and density when compared with gate logic [3,4]. A pass transistor network realized in Binary Tree Structure (BTS) form often requires fewer transistors and displays attractive fault detection characteristics [3]. In general, a BTS circuit is characterized by having a maximum of $2^n - 1$ nodes, each node having exactly two branches. One branch is controlled by variable x_i and the other by $\overline{x_i}$. The maximum number of transistors in a BTS network is $2^{n+1} - 2$. A general BTS network contains the maximum number of transistors and represents a complete decoding of an input space and hence only constants are input to the network. A general BTS network is employed here to formulate the next state equations for sequential circuits. Figure 1 shows a general BTS network which implements all three-variable functions, any of which can be realized by simply changing the pass variable constants, 1(0), at the input to the appropriate branch.

The delay through a series string of pass transistors is proportional to the square of the number pass transistors. This limits the size of a sequential circuit using BTS structures to about 5 state variables. However, this is not a significant limitation. Large state controllers can almost always be partitioned into small distributed state machines. Two full custom developments [5,6] illustrate this partitioning. About 100 bits of state control excluding counters and pipe delays were required on the 200,000 transistor Reed Solomon decoder for the NASA Hubble Space Telescope [6]. The state control was partitioned such that no state machine required more than 5 state variables. About 225 bits of state control excluding counters and pipe delays were required on the 210,000 transistor Auto Centroid

	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

Table 1: Example flow table.

Calculator chip designed for Lawrence Livermoore Laboratory [5]. Again the state control was partitioned such that only one individual state machine required more than 5 state variables.

3 State Machine Design

3.1 Architecture

The logic that forms each next state equation, Y_i, consists of the following elements: a storage device (normally a flip-flop), next state excitation circuitry which generates the next state values to the flip-flop, and input logic. Present state information is fed back by state variables y_i to the excitation logic. The excitation logic is a combinational logic function of the input and state variable information. In general, the information needed to generate all possible next state values for the circuit is resident within the excitation logic. The current input and state variables select the specific next state value.

In order for the circuit to implement arbitrary state transitions, the next state circuitry must assume a unique form. First, the hardware for each next state variable must be identical. Second, specific next state information must not be hardwired into the logic that forms the next state equations. Rather, specific next state values must be presented from an external source. The architecture presented here yields a circuit that can realize any flow table up to a maximum of m input states and 2^n internal states without a change in hardware.

Conceptionally, the new architecture operates as follows: For each predecessor state of S_i, there exists a pass transistor path in the excitation network that presents the next state value, S_i, to the flip-flops. Predecessor states for state S_i under an input I_p are all states which have S_i as a next state entry. In Table 1, the predecessor state for C under I_1 is A. Whenever the circuit is in a predecessor state of S_i, the next clock pulse will effect a transition to S_i. The pass transistor network consists of a single pass implicant that covers each predecessor state such that when any state is entered, a unique pass transistor path is enabled that passes the proper next state value to the flip-flops.

All equations are identical when they are realized with general BTS networks that completely decode all the internal states. That is, if there are n state variables, then the BTS network must decode all 2^n states. The value for the next state entries for each

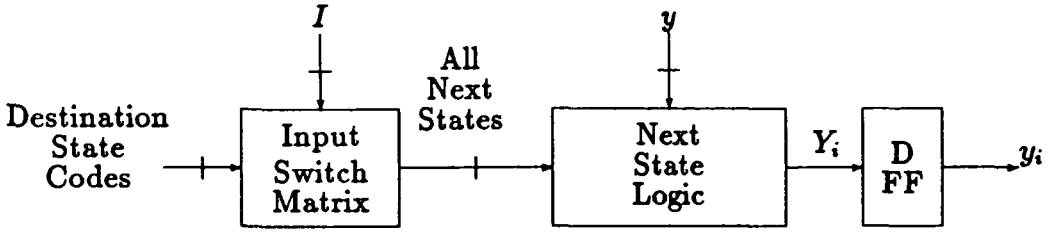

Figure 2: General block diagram.

predecessor state for S_i is the code for S_i and the constants for this code are input to the BTS network. A general block diagram is shown in Figure 2, where the next state logic is a general BTS network.

3.2 Operation

The following illustrates specifically how this architecture works. Let Table 2 depict an example for a general 3 state variable, 3 input state machine. I_1, I_2 and I_3 are the inputs, $S_0 \ldots S_7$ are the present states, and $N_{S_0 I_1}, N_{S_0 I_2} \ldots N_{S_7 I_3}$ are the next states. This can be generalized so that $N_{S_i I_j}$ are the next states for S_i under input I_j. $N_{S_i I_j}$ has been abbreviated as N_{ij}. The set of N_{ij} also comprise the destination state codes. Let the state assignment be $S_0 = 000$, $S_1 = 001$, $S_2 = 010$, $\ldots$, $S_7 = 111$.

The next state logic is a general BTS circuit with paths that decode each state. The input switch matrix is a pass transistor matrix, that passes the destination state codes to the next state pass network as shown in Figure 3. The circuit realization of this network operates in the following manner: All of the destination state codes N_{ij} are presented to the input switch matrix. For each input state I_i, all of the destination states in I_i are presented to the next state logic. The present state variables, y, select one and only one next state entry which is passed to the flip-flips. If the machine is in state S_1 and input I_2 is asserted, then N_{12_i} would be passed to the input of the flip-flop for next state variable Y_i. The current input state selects the set of potential next states that the circuit can assume (selects the input column) and the present state variables select the exact next state (row in the flow table) that the circuit will assume at the next clock pulse.

3.3 Design Example

Consider the state assignment and next state entries shown in Table 3. The circuit in Figure 3 shows the logic for implementing each state variable y_i. Each state is covered by a path through the BTS network that forms the next state logic. The logic of Figure 3 is replicated three times and the inputs are driven by the destination state information which is taken from Table 3. Figure 4 shows the programming of the input switch matrix for next state variable Y_3. Except for the constant values driving the input switch matrix, the design equations and pass transistor realizations for each state variable are identical.

	I_1	I_2	I_3
S_0	N_{01}	N_{02}	N_{03}
S_1	N_{11}	N_{12}	N_{13}
S_2	N_{21}	N_{22}	N_{23}
S_3	N_{31}	N_{32}	N_{33}
S_4	N_{41}	N_{42}	N_{43}
S_5	N_{51}	N_{52}	N_{53}
S_6	N_{61}	N_{62}	N_{63}
S_7	N_{71}	N_{72}	N_{73}

Table 2: General eight-state three-input flow table.

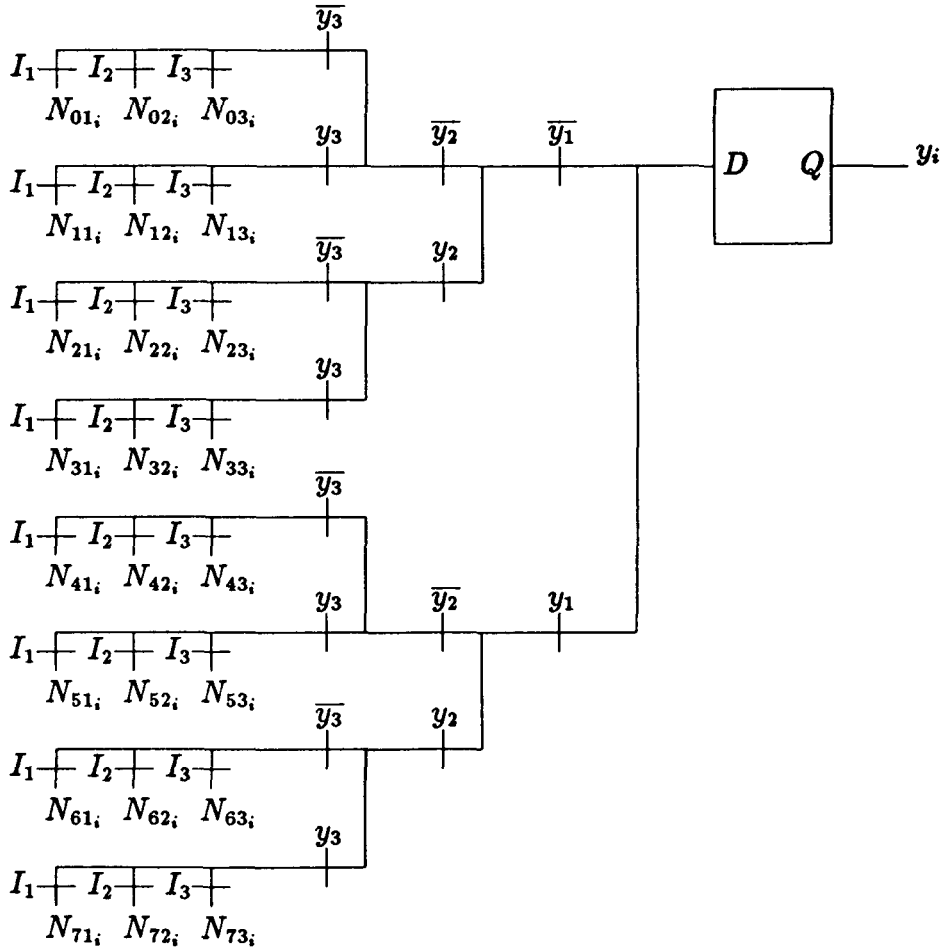

Figure 3: General eight-state three-input next state equation circuit.

y_1	y_2	y_3		I_1	I_2	I_3
0	0	0	A	010	001	000
0	0	1	B	011	010	001
0	1	0	C	100	011	010
0	1	1	D	101	100	011
1	0	0	E	000	101	100
1	0	1	F	001	000	101

Table 3: Example flow table with next state entries.

Since only six of the eight available states are utilized, the paths decoding S_6 and S_7 can have arbitrary next state constants. Here they are set to 0.

It is no longer necessary to derive the pass logic configuration for each next state equation. The next state information is only used as the input pattern to the input switch matrix. Since the next state information is stored in the input switch matrix, only the programming of the destination codes needs be changed to implement a different flow table.

3.4 Safe Operation

In some circuit designs it is possible to enter states that are not specified in the original flow table and it is then impossible to return to an original specified state. If this occurs, the circuit is termed unsafe [7]. The above architecture can be guaranteed to operate safely by simply defining the next state for all unspecified states as any of the originally specified states. Since the BTS network generates the next states for all possible states, there is no increase in the hardware necessary to produce a safe design.

4 Reliable Design

4.1 Real Time Fault Detection

In the following discussion, specified internal states denote those states that are specified in the original flow table and fault states denote those that do not appear in the original flow table. For example, in Table 3 state 001 is a specified state and 110 is a fault state. A circuit never enters a fault state under fault free conditions.

A key feature of the design presented in the previous section is that each state variable utilizes independent logic. Because of this feature, a failure in any component in the BTS network affects at most only one state variable. The occurrence of a fault may force the circuit into a state that is at most a distance one from the intended state.

Meyer has shown that a minimum distance-two state assignment will provide real time fault detection if the fault detector can detect the presence of a fault state within one clock period [8]. With a minimum distance-two state assignment, a single fault forces the circuit into a fault state. When this happens, the fault detector must simply detect the presence of fault states to detect the presence of a fault. For example, suppose the state assignment

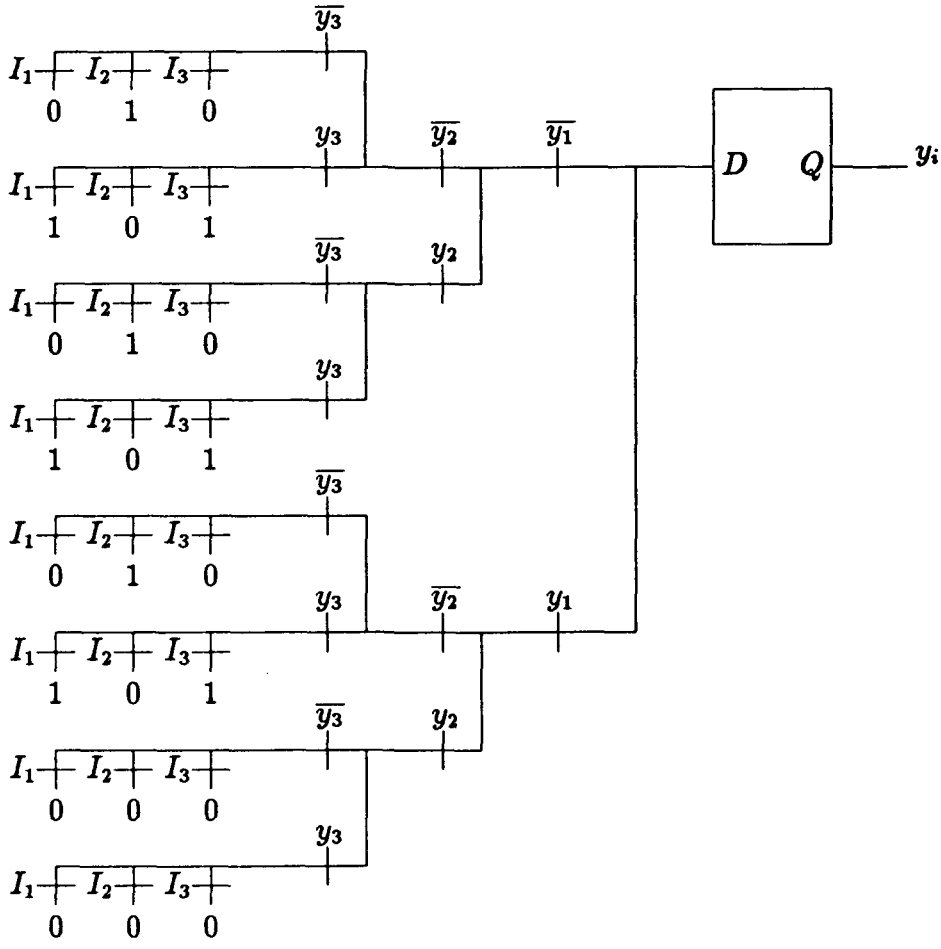

Figure 4: Programming of the input switch matrix for next state equation Y_3.

encodes all specified states to possess even parity and encodes the fault states to have odd parity. The fault detector would simply detect the presence of odd parity over the state variables. The hardware for this fault detector could be realized by adding another BTS network that is identical to any state variable circuit. However, a smaller more efficient fault detector would be a simple exclusive-or gate. A typical VLSI trade-off has to be made at this point. If design time is important, then replicating the BTS network is best; if minimum area is critical, then implementing the exclusive-or gate is best.

Open-circuit faults often create problems for fault detection because the charge stored on the node can mask failures. For example, if a transistor experiences an open circuit, the input to the D flip-flop will float when the path with the faulted transistor is enabled. If the previous input value to a flip-flop is 1(0) and the next value ought to be 1(0), then the state variable will not change when an open circuit is present. This assumes that the inputs change at a rate faster than the time it takes to discharge the input node to the flip-flop. In the case where there is no change in state, the circuit remains in a proper state and the fault is not detected. The circuit will malfunction whenever the previous input value to a flip-flop is 1(0) and the next value ought to be 0(1). In this situation, the input should transition but the presence of an open circuit does not allow a transition. Whenever the faulted path ought to force a transition in a flip-flop and does not, the presence of an open circuit will result in an unchanged state variable and the circuit entering a fault state which is detectable. In general, the fault detector detects only those faults that cause the circuit to assume a fault state. Open-circuit faults which cause a circuit to remain in a proper state are not detectable.

Only one extra state variable is needed to translate a minimum variable state assignment into a minimum distance-two state assignment. As stated above, a BTS circuit identical to a state variable can be used as the fault detector. Therefore, two additional BTS circuits are required to provide real time fault detection. If n BTS circuits are needed to implement a non-fault detecting circuit, then $n + 2$ BTS circuits are needed for real time fault detection. The depth of the each BTS circuit increases by one upon adding an additional state variable. Since each additional BTS circuit is identical, the extra VLSI layout and checking efforts are small.

Checking the checker is an important issue. To generate confidence in the checking hardware, it is important to have a mechanism that can achieve a self check. Detection of faults in the checking circuit itself is difficult because the states that the fault detector decodes are states that the circuit never enters under fault free conditions. One method to check for detector faults is to force the circuit into a fault state during an off-line test. A complete test would require that the circuit cycle through all fault states. Since the next state entries for the fault states are unspecified, it is a simple matter to specify the next state entries such that the circuit will cycle through the fault states.

Cycling through the fault states in the checking circuit is illustrated with the example shown in Table 4. In this example, let the next state entry of all fault states be specified such that the circuit cycles through all the fault states when a fault state is entered. Shown in Figure 5 are the next state entries which implement this condition. The specified states are noted on the K-map and their next state entries are left blank. The fault detector

y_1	y_2	y_3	y_4		
0	1	0	1	1	2
1	1	1	1	2	3
1	1	0	0	3	4
1	0	1	0	4	5
1	0	0	1	5	6
0	1	1	0	6	7
0	0	1	1	7	1

Table 4: Fault example

y_3y_4 \ y_1y_2	00	01	11	10
00		1 0111	0 3	1 1011
01	1 0010	0 1	1 1110	0 5
11	0 7	1 1101	0 2	1 0001
10	1 0100	0 6	1 1000	0 4

Figure 5: Cycle operation

output is noted in the top half of each cell with 1 denoting a fault state. The fault states show the next state entry in the bottom half of each cell. Notice that the states cycle among themselves: 0001 → 0010 → 0100 → 0111 → 1101 → 1110 → 1000 → 1011→ 0001.

A BTS circuit can easily be altered to enter the above cycle. A single change in any one of the constants feeding an input to the BTS network, like the one shown in Figure 3, will cause the circuit to enter the fault states. The input that effects the change can be an external signal to the controller. The important features of this fault checker are that it can be tested, there is no increase in hardware and the cycle test can be invoked externally.

4.2 Fail Safe Operation

A fail safe circuit is designed such that a fault can never produce an "unsafe" output. Whenever a fault occurs in a sequential circuit, the circuit must be forced into a well defined set of fault states with safe outputs. A circuit must not be allowed to assume random states, because unsafe outputs could be generated [9].

In an n-variable minimum distance-two state assignment, there can be no more than 2^{n-1} specified states; at least half of the total states are fault states. Under worst case conditions, a single fault will force the circuit into a fault state. For fail safe operation, the next state entries for the fault states must be specified in such a manner that they will prevent the circuit from re-entering any specified state. The circuit architecture defined earlier will allow the next state entries for fault states to be specified in any desired manner without a hardware penalty. A judicious choice of next state entries can provide the desired fail safe qualities.

Let S_0 be the state where all state variables are 0. A fail safe design must consist of the following elements:

- Minimum distance-two state assignment
- S_0 and all states a distance one from S_0 are specified as fault states
- All fault states are programmed with a next state entry of S_0
- Outputs are programmed to be safe in all fault states

Since there is no sharing of hardware, the occurrence of a single fault can immediately affect no more than one state variable. Fail safe operation is guaranteed for the following set of fault conditions. Let the circuit be in state S_i with the next state entry S_j under input I_p. The combination of S_i and I_p selects a unique pass transistor path that presents the next state value for S_j to the inputs of the flip-flops. Assume that a fault is propagated to cause faulty operation in a next state variable.

Stuck-at-fault within the BTS network A fault can occur in one of two locations:

1. A stuck-at-fault is present within the BTS network, including the primary input lines. In this case, the next state will be fault state $S_j \oplus$ e, where e is an n-tuple of weight 1. Since the next state for all fault states is S_0, the circuit transition is $S_j \rightarrow S_f \rightarrow S_0$. Since the next state for S_0 is S_0, the circuit is safe.

2. A fault occurs at the output of the BTS network for state variable Y_k. In this case, fault state $S_0 \oplus$ e is assumed, where e is an n-tuple that is all 0 except bit position k = 1. Since the next state entry for this state is the fault state S_0, the circuit will not transition further, and is therefore safe.

Consider the example in Figure 4 and the circuit shown in Figure 6. The next state entry for the fault states are denoted with an "f" and are coded 0 for S_0. Let the current state be 6 (0110); the next state is 7 with code 0011. The pass transistor path corresponding to 0110 is enabled for each state variable, passing the code for state 7 to the output of the BTS network. If there is a stuck-at-1 fault along this path, then the possible next states are 1011, 0111 or 0011 depending on which next state variable is affected. Notice that a stuck-at-1 fault for either Y_3 or Y_4 is masked since a 1 is suppose to be passed. If states 1011 or 0111 are entered, then the next state will be 0000, unless the outputs of the BTS network for Y_1 or Y_2 are stuck-at-1, in which case the circuit will assume state 1000 or 0100. Since present state and next state are the same in all three cases, the circuit remains stable. A similar situation occurs for a stuck-at-0 fault except state S_0 is entered and $e = 0$.

Transistor stuck-on If a given transistor is stuck-on, then two paths are enabled at some node. Along one path is the code for the specified state, and along the other is the code for S_0. If the two signals agree, the fault is masked. If they differ, then a conflict is present and the exact logic level is determined by the electrical characteristics of the network. If the path for code S_0 dominates and propagates a 0 to the output, then the circuit will assume a fault state. At this point one of two things can happen:

1. The fault state can enable an entirely different set of transistors causing the stuck-on fault to be isolated and guarantees that the circuit enters fault state S_0.

2. The fault state does not isolate the stuck-on fault. S_0 will still be entered because only 0's are being passed when the circuit is in the fault state.

Continuing with the above example, let the circuit be in state 6. If the transistor controlled by y_4 in path 0111 is stuck-on, then there is a conflict between 0 (coming from f) and 1 (coming from 7) at the first node. If the 1 dominates the 0, the circuit will operate fault free. If the 0 dominates the 1, then the next state will be 0001 or 0010 if Y_3 or Y_4 are affected. Either of these states will force the circuit to S_0 and the operation is then fail safe.

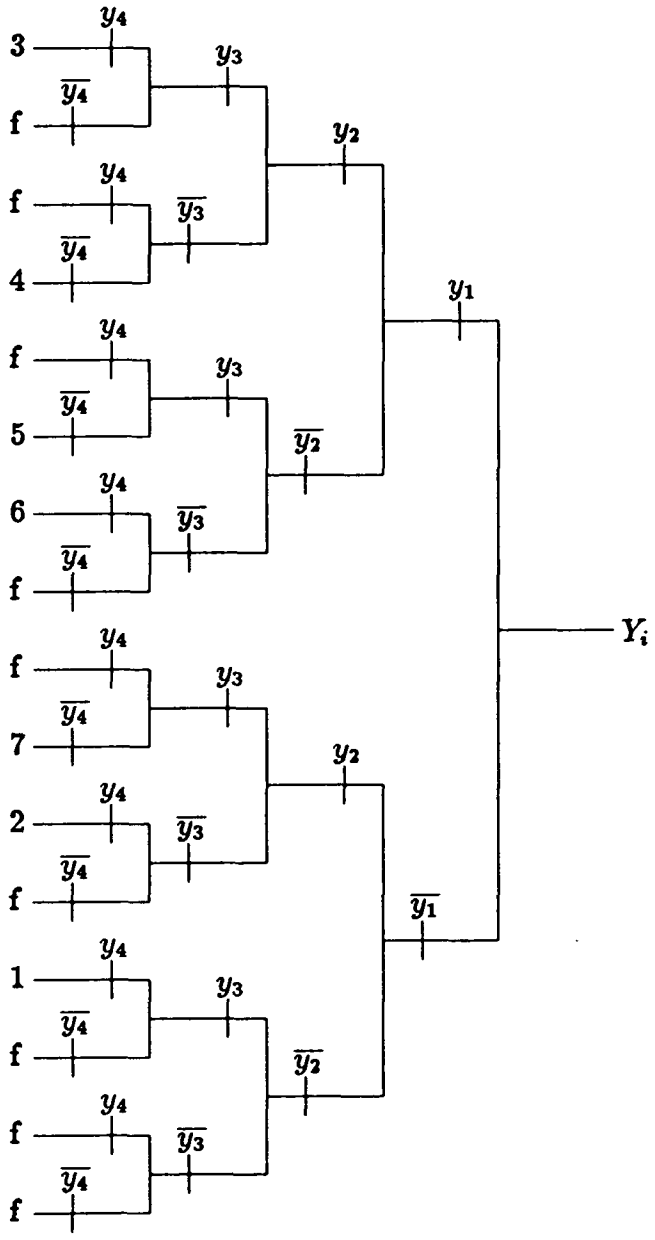

Figure 6: BTS network for fail safe operation.

State variable stuck-at-1 If the output of flip-flop y_i assumes a stuck-at-1 value, then an entire set of transistors controlled by y_i or its complement are turned on. Let the circuit be in state S_i where $y_k = 0$ and the output of the flip-flop for Y_k becomes stuck-at-1. The BTS circuit will respond as if the circuit is in fault state $S_i \oplus e$, where e is all 0 except for bit $k = 1$. The next state entry for this state is 0 for all state variables which will force the circuit to $S_0 \oplus e$. This state is another fault state with a next state entry of S_0. Therefore, the circuit is stable and safe.

This circuit cannot be guaranteed to operate in a fail safe manner for stuck-open or for state variable stuck-at-0 faults. These faults can disable the pass transistor path that drives the flip-flops and can cause a tristate input to the flip-flops. A tristate input to the flip-flops may or may not force the circuit to a fault state. The circuit can remain in the same valid state until the charge at the BTS output node leaks off. Many clock cycles could occur and since the circuit is in a valid state, the fault detector and fail safe circuitry would not respond to the failure.

4.3 Fault Tolerance

A designer can achieve fault tolerance using either of the following procedures:

1. Place an error-correcting code on the state assignment as proposed by Meyer [8]. For single fault tolerance, all fault states adjacent to a specified state are encoded with the same next state entry as the specified state. If a single fault occurs, a fault state adjacent to the specified state is assumed. However, since the states adjacent to each specified state have the same next state entry, the circuit will transition to the proper next state, or at least to a state within a distance 1 of the specified state.

2. Use $n + 1$ safe circuits to achieve an n fault tolerant circuit [10]. Either a simple AND or a simple OR gate could be used to produce the fault tolerant output.

4.4 Adaptive Operation

One of the attractive features of the architecture proposed here is its adaptive nature. The constants, which are input to the BTS network, can either be derived from connections to V_{dd} and V_{ss} lines or they can come from a register or other storage cell. Destination constants (codes) coming from a register can be changed. With each change of the destination constants an entirely different set of transitions can be implemented.

If it would be possible to identify transitions that are affected by faults, then changing the flow table could be accomplished to avoid the faulty transition. For instance, if a given transition from state S_i produced a malfunction and it was determined that some transistor along the path decoding S_i had failed, then that particular pass transistor path could be a avoided. A spare non-fault state S_s could assume the role of S_i and every transition to S_i could then transition to state S_s. This change can be effected with a new set of constants that define S_s as the next state for all predecessor states of S_i. Moreover,

all next state entries for S_i can be mapped to S_*. Therefore, state S_i is no longer entered and the particular pass transistor path that contained the fault is not used again.

5 Conclusion

A new architecture is presented that has attractive features for producing reliable sequential controllers. A design procedure that is applicable to VLSI is given for realizing the new architecture. Without any increase in hardware, a circuit can be designed to be safe. Through the addition of only one more state variable and its associated BTS network, a circuit can be realized that has real time fault detection and fail safe capabilities. The new architecture also allows for the implementation of adaptable circuits that are capable of initiating alternative transition sequences to replace those that are faulty. The adaptable nature of the circuit is achieved through no increase in the fail safe design hardware.

References

[1] J. Venbrux, and N. Liu, "VLSI Chip-set for Data Compression Using the Rice Algorithm," NASA SERC Symposium on VLSI Design, Moscow, Idaho, pp.41-51, Jan. 1990.

[2] S. Whitaker, S. Manjunath and G. Maki, "Sequence Invariant State Machines", submitted to the IEEE Journal of Solid State Circuits.

[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, pp. 139-144, Oct. 1984.

[4] D. Radhakrishnan, S. Whitaker and G. Maki, "Formal Design Procedures for Pass-Transistor Switching Circuits," IEEE Journal of Solid State Circuits, pp. 531-536, Apr. 1985.

[5] K. Cameron, "ACE: Automatic Centriod Extractor for Real Time Target Tracking", accepted for publication at IEEE Northcon, Seattle, Washington, Oct., 1990.

[6] S. Whitaker, K. Cameron, P. Owsley and G. Maki, "Custom CMOS Reed Solomon Coder for the Hubble Space Telescope", accepted for publication in the IEEE Military Communications Conference.

[7] R. Wickersham and G. Maki, "Safe Asynchronous Sequential Circuits," IEEETC Vol. C-23, pp. 494-500, May 1974.

[8] J. F. Meyer, "Fault Tolerant Sequential Machines," IEEETC, Vol C-20, Oct. 1971.

[9] G. Maki and D. Sawin, "Fail Safe Asynchronous Sequential Machines," IEEETC Vol. C-24, pp 675-677, June 1975.

[10] D. Sawin and G. Maki, "Fault Tolerant Asynchronous Sequential Circuits," IEEETC, Vol. C-23, pp. 651-657, July 1974.

This research was supported in part by NASA under the NASA Space Engineering Research Center grant NAGW-1406 and by the Idaho State Board of Education under grants 88-038 and 89-041.