Pulse Mode VLSI Asynchronous Circuits

Q. Chen and G. Maki
NASA Space Engineering Research Center
for VLSI System Design
College of Enginering
University of Idaho
Moscow, Idaho 83843

Abstract - **A new basic VLSI circuit element is presented that can be used to realize pulse mode asynchronous sequential circuits. A synthesis procedure is developed along with an unconventional state assignment procedure. Level input asynchronous sequential circuits can be realized by converting a regular flow table into a differential mode flow table, thereby allowing the new synthesis technique to be general. The new circuits tolerate 1-1 crossovers. This circuit also provides a means for state sequence detection and real time fault detection.**

1 Introduction

Many asynchronous sequential circuits can be modeled as a pulse mode circuit since the inputs are presented in the form of pulses [1]. Level input sequential circuits can be modeled as a pulse mode circuit by detecting input state changes [2]. This work presents a basic circuit that can used to realize state variables that are effective in the realization of pulse mode circuits.

Sequential circuits are normally defined in terms of flow tables, such as shown in Table 1. The inputs are shown across the top and the states along the side. The states are encoded with internal state variables y_i. Next state variables Y_i identify the next state that the circuit will assume.

This paper presents a VLSI circuit element that allows for efficient realizations of pulse mode asynchronous sequential circuits. The network consists of pass transistor next state forming logic with a unique buffer.

The paper describes the following:

- Synthesis procedures for pulse mode asynchronous sequential circuits.
- State assignment procedure for differential mode asynchronous sequential circuits.
- Tolerance of 1-1 input crossover situations. (This circuit is designed to tolerate 0-0 input crossover situations also.)
- State sequence detection.
- Real time fault detection.

2 Pulse Mode Circuits

The next state equations can be expressed as follows[3]:

$$Y_i = f_{i1}I_1 + f_{i2}I_2 \cdots + f_{in}I_n \tag{1}$$

where Y_i is next state variable, I_p is the input state and f_{ip} is a sum-of-products expression of state variables. It has been shown that the next state equations can be expressed as a pass logic expression[3]:

$$Y_i = I_1(f_{i1}) + I_2(f_{i2}) \cdots + I_n(f_{in}) \tag{2}$$

where $I_p(f_{ip})$ means input I_p passes function f_{ip}.

The basic circuit to implement pulse mode circuits is shown in Fig. 1. Each state variable is realized with this circuit. If there are m state variables, then there would be m such circuits except that there is only one NOR gate.

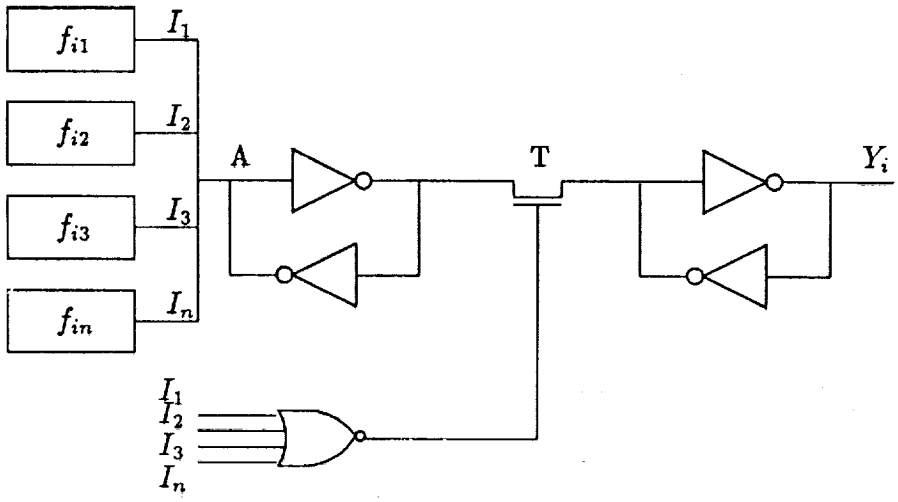

Figure 1: Next State Circuit Module

In pulse mode operation, all I_p could be 0. When all input states I_p are 0, the pass networks f_{ip} are disabled and hence are tristated from the inverter input of the first stage. The feedback inverter in the first stage is provided to sustain the value at point A of the first stage. However, the feedback inverter consists of weak devices that can be overdriven by the $I_p(f_{ip})$ networks. The same kind of inverter is placed in the second stage of the circuit after transistor T.

For pulse mode operation, assume one and only one input state I_p is 1 at a time or all I_p are 0. In other words, only one input pulse is present at a time. When $I_p = 1$, pass network f_{ip} presents the proper next state value to $\hat{Y}_i$ as specified in Eq. 2 for Y_i to the input of the inverter at point A. The feedback inverter is composed of weak pullup and pulldown transistors such that they can be overdriven by the value passed by $I_p(f_{ip})$. Therefore the correct next state value as defined by Eq. 2 is present at point A in Fig. 1

and $\hat{Y}_i$ contains the complement of Y_i. When all $I_p = 0$, transistor T is enabled and $\hat{Y}_i$ is passed to Y_i and the circuit assumes the proper next state.

To summerize, when one $I_p = 1$, $\hat{Y}_i$ assumes the complement of the proper next state value of Y_i as defined by Eq. 2. When all $I_p = 0$, $\hat{Y}_i$ is passed to the second stage of the inverter and Y_i assumes the value defined by Eq. 2. The new present state feeds back to the f_{ip} networks to generate the new next state values to the first stage, dependent on which $I_p = 1$. An interesting observation can be made which is common to all asynchronous sequential circuits, but perhaps is more easily seen here. When all $I_p = 0$, the present state, as determined by present state variables y_i, feed back to the f_{ip} logic. All possible next states are generated and appear at the input of the pass transistors controlled by I_p. The circuit has "calculated", as determined by Eq. 2, all possible next states that the circuit could enter and is prepared to assume any and every next state as defined by the f_{ip} terms. The exact next state is specified by the I_p state that becomes 1.

The state assignment problem for asynchronous sequential circuits is always a significant problem. Pulse mode flow tables are in every way asynchronous in operation. Therefore, the designer must be concerned about state assignment issues. Assume the present state of the circuit is S_i and state S_j is the next state when input I_j becomes 1. When all inputs are 0 prior to $I_j = 1$, the state variables y_i define the circuit to be in state S_i. When $I_j = 1$, since transistor T is disabled, the next state variables Y_i do not change. $\hat{Y}_i$ changes to assume values associated with S_j as defined by Eq. 2 when $I_j = 1$. However, Y_i remains unchanged as long as $I_j = 1$. Y_i does not change to the value of S_j until I_j returns to 0, at which time $\hat{Y}_i$ cannot change. Therefore, each state transition occurs in two stages:

$$\begin{aligned} \hat{Y}_i &= \overline{\sum I_p(f_{ip})} && \text{when } I_p = 1 \\ Y_i &= \hat{Y}_i && \text{when all } I_p = 0 \end{aligned}$$

A critical race can exist in an asynchronous sequential circuit only when the state variables y_i being fed back can affect Y_i without a change in input. Since the inputs must change before present state variable y_i can affect next state variable Y_j, no critical race can occur. The following theorem has been established.

Theorem 1 *Asynchronous sequential circuits implemented with the basic circuit shown in Fig. 1 are void of critical races.*

If the circuit cannot experience a critical race, then the Single Transition Time (STT) state assignment procedures need not be followed, specifically the Tracey conditions[5] need not be met. Moreover, since the STT conditions need not be met, any state assignment is satisfactory as long as each state has a unique code.

The design procedure can be stated as follows:

Procedure 1 Step 1 *Create an appropriate flow table.*

Step 2 *Provide a state assignment where each state has a unique code.*

			XC				Z
y_1	y_2		00	01	11	10	
0	0	A	-	A	-	B	0
0	1	B	-	C	-	-	0
1	1	C	-	A	-	D	1
1	0	D	-	C	-	-	1

Table 1: Example Flow Table

Step 3 *Form the state table.*

Step 4 *Find the next state equations in the following form:*

$$Y_i = \sum I_p(f_{ip})$$

where each input passes an f_{ip} expression of state variables.

Example 1 *Realize a circuit which has two pulse inputs X and C and a level output Z. C represents a clock that produces pulses at a regular interval. Z must be 1 between pulses C_i and C_{i+1} only if an X pulse occurred between clock pulses C_{i-1} and C_i.*

The reduced flow table with the state assignment is shown in Table 1. The design equations for this flow table are:

$$Y_1 = X(y_1) + C(\overline{y_1}(y_2) + y_1(\overline{y_2}))$$
$$Y_2 = X(\overline{y_1}) + C(\overline{y_1}(y_2) + y_1(\overline{y_2}))$$

2.1 Design By Inspection

The synchronous state assignment procedure allows for a great deal of flexibility. The one-hot-code is well known as a state assignment that allows one derive the design equations by inspection. A one-hot-code encodes an n-row flow table with n-state variables where state S_i is encoded with $y_i = 1$ and all other $y_j = 0$, $j \neq i$. A predecessor state of state S_i is a state the circuit is in prior to an input change that forces the circuit into S_i.

If S_j is a predecessor state to S_i under input I_p, the partial next state equation is

$$Y_i = I_p(y_j).$$

If $S_{j1}, S_{j2}, \cdots, S_{jk}$ are predecessor states to S_i, then the partial next state equation is

$$Y_i = I_p(y_{j1} + y_{j2} + \cdots + y_{jk}).$$

In general, the f_{ip} terms become simple sum-of-products where each product is an uncomplemented state variable.

Design Procedure 1 can be employed by simply changing Step 2 to implementing a one-hot-code. The equations can be formed by the well known inspection method. Simplier f_{ip} terms result. The disadvantage is that more state variables are generally needed. The design equations for Table 1 are

$$\begin{aligned} Y1 &= C(y_1 + y_3) \\ Y2 &= X(y_1) \\ Y3 &= C(y_2 + y_4) \\ Y4 &= X(y_3) \end{aligned}$$

Liu[6] proposed a design technique for iterative logic array synchronous sequential circuits that have the unique property where each state has predecessor states only in one input. The next state equations have the form

$$Y_i = I_p(f_{ip})$$

where each f_{ip} is a sum-of-products with each product term consisting of a single complemented or uncomplemented state variable. This technique reduces the amount of logic further for each next state variable in that only one input I_p pass gate is needed. The potential disadvantage is that more state variables can be needed.

3 Tolerance to 1-1 Input Overlap

In the previous section there were no constraints on the width of each input pulse. (The minimum width must be long enough to pass the signal to the output of the input inverters at the first state). It was assumed that only one I_p would be 1. This condition can be relaxed. For simplicity, suppose two inputs I_p and I_q are both 1. Moreover, suppose the circuit should transition from S_i to S_p or S_q under I_p or I_q respectively. When both I_p and I_q are 1,

$$\hat{Y}_i = \overline{I_p(f_{ip}) + i_q(f_iq)}$$

As long as I_p and I_q are 1, there can be conflicting signals at the input to the inverter of the first stage of Fig. 1. Since, at least one input = 1, transistor T is not enabled and y_i does not change and the conflict does not affect the present state. The circuit remains in state S_i and will remain in S_i until both I_p and $I_q = 0$. If $I_p(I_q)$ remains 1 longer than $I_q(I_p)$, then $f_{ip}(f_{iq})$ will be passed to specify $\hat{Y}_i$ and only when both inputs are 0 will the circuit transition to $S_p(S_q)$. Therefore, the circuit action is determined by the input that remains 1 last.

Theorem 2 *If more than one input state is 1, then the next state of the circuit is determined by the input that remains 1 last.*

Proof: If more than one input = 1, then $\hat{Y}_i$ is determined by the equation

$$\hat{Y}_i = \overline{\sum_{j=\ell}^{k} I_j(f_{ij})}$$

where I_j are those inputs that are 1. Since y_i changes only when all I_j are 0, the circuit does not transition until all $I_j = 0$. Suppose I_p is the last input that is 1. Then the equation for $\hat{Y}_i$ becomes

$$\hat{Y}_i = \overline{I_p(f_{ip})}.$$

When I_p transitions to 0, then Y_i assumes the state determined by $\hat{Y}_i$ which was specified by I_p.
QED.

From Theorem 2, it is clear that the order in which inputs transition from $1 \rightarrow 0$ is important. Transitions from $0 \rightarrow 1$ are unimportant. Therefore, if more than one input state is 1, it is unimportant which order the inputs transition $0 \rightarrow 1$. The next state is specified by the last input that transitions $1 \rightarrow 0$. For example, suppose there are four input states for a circuit. If the inputs transition as shown in Fig. 2, the circuit will assume the state specified by I_3 when all the inputs are 0.

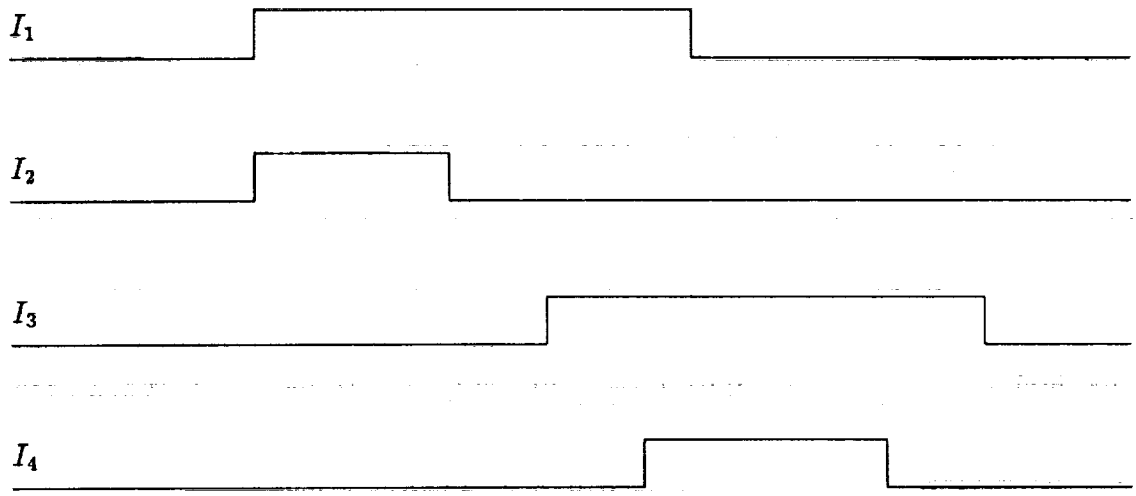

Figure 2: Input Waveform Example

4 Level Input Circuits

The previous discussion focused on pulse mode circuits. Several researchers have introduced the notion of transition sensitive asynchronous sequential circuit design [2,7]. Bredeson [2] converted a level input flow table to a transition sensitive (TS) flow table. A TS flow table shows the table entries that result from a change in inputs. The essential feature in a TS design is that inputs are represented as pulses which are created whenever the input state transitions from $0 \rightarrow 1$. Consider the level input flow table of Table 2. The TS representation of this flow table is shown in Table 3. Once the flow table is in the TS form, the design procedure in Section 2 applies.

Bredeson introduced another notion in the design of TS circuits. If one begins with a primitive row flow table, then the input state variables can become the state variables. Additional state variables are needed only to produce unique codes for the states and this is accomplished by partitioning stable states in each column of the flow table. In Table 3,

				X_1	X_2		
y_1	y_2	y_3		00	01	11	10
0	0	0	A	A	B	-	C
0	1	0	B	A	B	F	G
1	0	0	C	D	H	E	C
0	0	1	D	D	B	E	C
1	1	0	E	D	B	E	G
1	1	1	F	A	H	F	C
1	0	1	G	A	H	E	G
0	1	1	H	D	H	F	C

Table 2: Level Input Flow Table

				X_1	X_2		
y_1	y_2	y_3		00	01	11	10
0	0	0	A	C	-	-	-
0	1	0	B	-	F	-	-
1	0	0	C	-	-	-	D
0	0	1	D	C	-	-	-
1	1	0	E	-	-	B	-
1	1	1	F	-	-	H	-
1	0	1	G	-	-	-	A
0	1	1	H	-	F	-	-

Table 3: Transition Sensitive Flow Table

y_1 and y_2 are assigned to x_1 and x_2 respectively. State variable y_3 is assigned to partition the stable states in each column. For example y_3 partitions states A and D in the first column. Therefore only one state variable is needed to implement the flow table rather than the expected three.

5 State Sequence Detection

It might be desirable to be able to detect the potential transition between a pair of states that might be associated with a critical event. Suppose state S_k can be entered only from state S_i under fault free conditions. If state S_k is entered from state S_n, $i \neq n$, then an error has occurred. In some cases, such a transition should not be allowed.

The circuit presented here is capable of providing information necessary to detect the occurrence of a transition between a pair of states prior to the actual transition. If one knows that an undesirable transition is about to occur, it is possible to prevent the transition and avoid an unwanted event.

State information is present at two points in the circuit of Fig. 1. The present state is available at the output of the second stage Y_i. When the next input state is 1, the next state information is specified by $\hat{Y}_i$. To detect a sequence between a pair of states, then the state information at Y_i and $\hat{Y}_i$ can be decoded.

If it was desired to permit a transition to state S_k only from state S_i, then S_i can be decoded from Y_i and S_k can be decoded from $\hat{Y}_i$. If the next state as specified by $\hat{Y}_i$ is S_k and the present state is not S_i as specified by Y_i, then an error condition can be signaled. This is depicted in Fig. 3. To prevent the circuit from assuming state S_k under the error condition, the error signal can be fed into the NOR gate which drives transistor T in Fig. 1. The error signal would prevent the circuit transition to state S_k. Moreover, since transistor T is not enabled when the error condition is detected, the circuit will not transition to S_k and remain in S_i. It might be desirable to stop all processing when the error condition is detected. If so, the error signal can be used to disable all further input state changes and the circuit would remain in the current state without any further state transitions. $\hat{Y}_i$ will specify the incorrect state S_j, $S_j \neq S_i$. If one desired to know the value of S_j, $\hat{Y}_i$ could be examined to reveal the error state to help with diagnostics.

6 Fault Detection

Classical fault detection of sequential faults includes using an error detection code on the state assignment [8]. If hardware is not shared, a single error detection code is sufficient. Since the design approach used here does not share logic, except for the NOR gate which drives the T transistors, a single error detection code can be employed and is used in the work presented here. It is assumed that the NOR gate is hard core for this discussion. Moreover, it is assumed that only one device can fail at a time and that the circuit will assume all total circuit states before a second fault can occur. In this discussion, all faults

that can cause a false next state value are detectable; this includes stuck-at, stuck-open and stuck-on faults.

The circuit presented thus far has some interesting fault detection capabilities. Most other fault detection mechanisms for sequential circuits detect the presence of a fault after the circuit has assumed a faulty state. This circuit is able to detect the presence of a fault in most of the circuit before the circuit actually enters the fault state.

In this discussion, it is assume that a simple parity code is used for fault detection. Under the single fault assumptions above, only one extra state variable needs to be added. Let the states of the flow table be encoded with an even parity state assignment. Whenever odd parity is assumed by the state variables, a fault condition is detectable. Let all odd parity states (fault states) be assigned to have a next state value that is also odd parity. Therefore, whenever an odd parity state is assumed, the next state is also an odd parity state.

The circuit for fault detection is shown in Fig. 4. The fault detector simply detects the presence of odd parity on the state assignment; f is assigned to equal 1 when an odd parity state is present. The fault detector monitors the parity of $\hat{Y}_i$. If a fault occurs to any of the circuitry that produces $\hat{Y}_i$, f will detect its presence. With a fault, f = 1, and since f feeds into the NOR gate, the T transistor is not enabled and the fault state cannot be assumed by Y_i. In this case, the circuit does not enter the fault state. Moreover, if the input states can be disabled, the circuit will remain in the current state.

Signal f will be driven towards a 1 value as the circuit transitions between unstable and stable states. Signal f then would prevent the T transistor from being enabled, but this actually helps the circuit not enter an improper state. Signal f can be used therefore to produce a self synchronizing signal, but this is a subject beyond the scope of this paper.

If a fault occurs in the second stage after the T transistor, then an odd parity state will be entered. The next state value as specified by the f_{ip} terms will be odd parity also since it is assumed that only one fault is present. If the f_{ip} terms generate odd parity, then $\hat{Y}_i$ will also have odd parity and then f = 1 with the fault being detected.

A fault in a T transistor will have the same impact as a fault in the second stage. If Y_i assumes the correct value in spite of a faulty T, no error is detected and the circuit operates as designed. Only when Y_i assumes an incorrect value will an odd parity state be entered and hence detected.

7 Summary

A fundamental logic circuit has been presented that will allow for efficient implementation of pulse mode asynchronous sequential circuits. Level input flow tables can be transformed into transition sensitive flow tables which can be directly implemented with the circuit presented here. The resulting circuits are tolerant of 1-1 crossover conditions. The final next state of the circuit is determined by the last input that is 1 whenever more than one input state is 1.

The unique characteristic of state sequence detection can be achieved with this circuit.

It is very easy to detect the present and next state in the circuitry and to prevent next state transitions to occur. In addition to state sequence detection, real time fault detection can be achieved where a fault state can be detected prior to a transition to a fault state. This fault detection capability covers a wide range of fault conditions and possible faults in the circuit.

References

[1] K. Cameron, S. Whitaker and J. Canaris, "ACE: Automatic Centroid Extractor for Real Time Target Tracking", NASA Symposium on VLSI Design, pp. 8.2.1-8.2.8, Nov, 1991.

[2] J. Bredeson and P. Hulina, "Synthesis of Multiple-Input Change Asynchronous Circuits Using Transition-Sensitive Flip-Flops", IEEE Trans. Comput., vol. C-32, no. 5, pp. 37-44, May 1973.

[3] S. K. Gopalakrishnan and G. K. Maki, "VLSI Asynchronous Sequential Circuit Design", ICCD, Sept, 1990, pp 238-242.

[4] S. Whitaker and G. Maki, "Pass-Transistor Asynchronous Sequential Circuits", IEEE JSSC, Vol.24, No.1, Feb. 1989, pp. 71-78

[5] J. Tracey, "Internal State Assignment for Asynchronous Sequential Machines", IEEE Transactions on Electronic Computers, Vol. EC-15, Aug. 1966, pp. 551-560.

[6] M. Liu, K. Liu, G. Maki and S. Whitaker, "Automated ILA Design for Synchronous Sequential Circuits", NASA Symposium on VLSI Design, Vol 3, October 1991.

[7] J. Smith and C. Roth, "Analysis and Synthesis of Asynchronous Networks Using Edge Sensitive Flip-Flops", IEEE Trans on Computers, vol. C-20, pp. 847-855, Aug 1971.

[8] John Meyer, "Fault Tolerant Sequential Machines", IEEE Transactions on Computers, vol. C-20, October 1971. sequential circuits. IEEE TC around 1970.

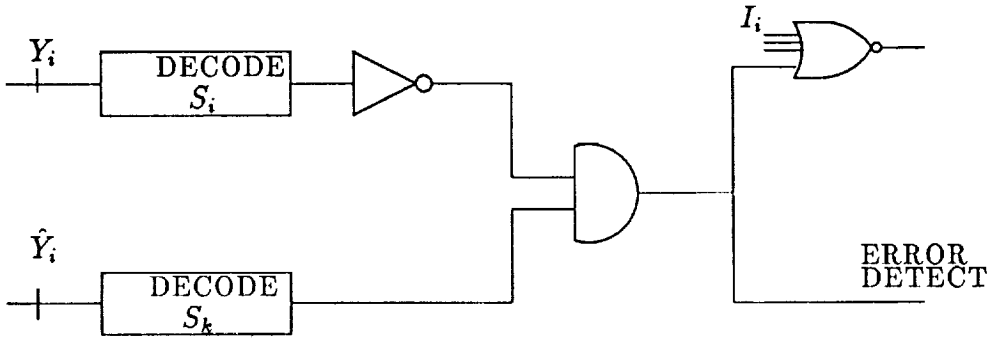

Figure 3: State Sequence Detection Logic

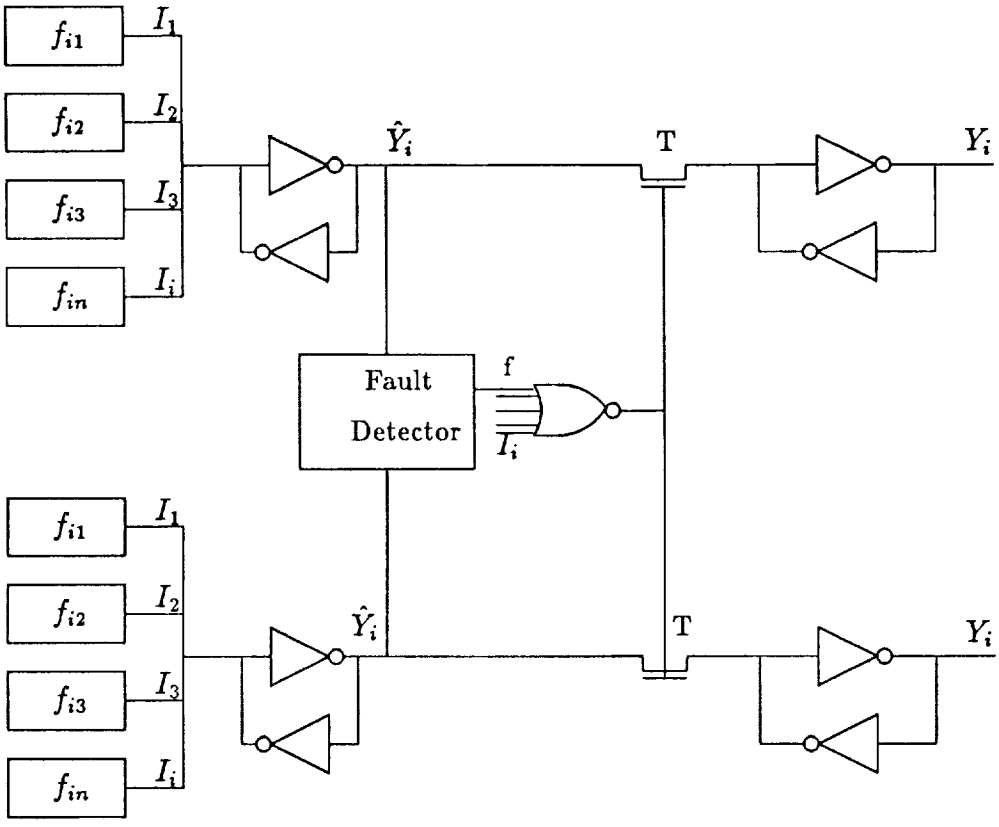

Figure 4: Fault Detection Logic