Self - Exercising k-order Comparators Based on Built-In Current Sensing

X. Kavousianos & D. Nikolos

Department of Computer Engineering and Informatics
University of Patras, 26500, Patras, Greece &
Computer Technology Institute
Kolokotroni 3, Patra, Greece

e-mail : kabousia@ceid.upatras.gr, nikolosd@cti.gr

Abstract

K-order comparator was defined recently in [16] as the combinational circuit that compares two operands and decides if these differ in less than k bits. Also in [16] a systematic method to design Self-Exercising Self-Testing Comparators was proposed. The applications of the proposed in [16] k-order comparators are restricted by the fact that when the operands under comparison are not identical, the k-order comparator exhibits static power consumption. In this paper we present two new designs for the k-order comparator that are suitable for a wide range of applications, as error detecting and correcting codes [2-7], fault tolerant cache memories [9] and broadcast networks.

Keywords : Self-Checking Circuits, Self-Testing, Self-Exercising Checker, Built In Self-Test, Built In Current Sensing

I. Introduction

Self Checking Circuits (SCC) [10] are widely used in applications with high reliability requirements, due to their ability to detect errors on line during the normal system operation. The type of errors covered include those caused by permanent, transient as well as intermittent faults. A SCC consists of a functional circuit, the output words of which belong to a certain code, and a checker that monitors the output of the functional circuit and indicates whether a code word or a non-code word has appeared. To achieve the totally self-checking goal (i.e.,

the first erroneous output of a functional block is signalled by the checker) [8] the checker was proposed to be Totally Self-Checking (TSC) [11] or Strongly Code Disjoint (SCD) [12]. However the achievement of the totally self-checking goal in practice depends on the actual input vectors that the checker receives during the operation of the functional unit, which usually differs from application to application.

In [1] it was shown that Self-Exercising (SE) Self-Testing or Strongly Code Disjoint checkers are more close to achieve the Totally Self-Checking goal than TSC and SCD checkers. Besides a SE checker has the advantage that can be designed to be self-testing or strongly code disjoint for a more realistic fault model than TSC and SCD checkers. SE self-testing checkers were defined in [1] as follows:

Definition. The self-exercising checker is self-testing with respect to a fault set F if for each fault f in F, either the checker receives during normal operation a code input that produces a noncode output, or a noncode output is produced to primary outputs (Z_0, Z_1) (figure 1) due to the test phase.

In [18] we presented a systematic method for designing a SE self-testing k-order comparator. When the input vectors are not identical the k-order comparator design proposed in [18] has high static power consumption. As we will show in the following there are applications where the compared operands are rarely not identical, thus the k-order comparator rarely exhibits static power consumption. But we will show also that in other applications this happens very often.

The final step in the error detection and correction procedure of the k-EC/d-ED/AUED, k-EC/AUED, k-EC/d-UED and k-EC/d-ED/m-ED codes [2-7] consists of a comparison exercising whether the received word and the corrected one differ in more than k bit positions. We can easily see that a (k+1)-order comparator is suitable for the implementation of this step. During the normal, fault free, operation the received word and the corrected one differ in t bits, $1 \leq t \leq k$, when a correctable error has occurred in the received word and in more than k bits when an only detectable error has occurred. The probability a correctable error to have occurred in the received word is much smaller than the probability the received word to be error free, while the probability an only detectable error to have occurred is even smaller. The k-order comparator proposed in [16] has static power consumption when the compared words are not identical. Then from the above we conclude that when the k+1-order comparator proposed in [16] is used for the implementation of the above codes rarely has static power consumption.

In the case that the k+1-order comparator is used for the implementation of a k-EC/d-ED code in the cache tag memory [9] the two operands (the search tag and the accessed tag) that are compared may differ in t positions. When t = 0, then the search tag and the accessed tag are identical, and the static power consumption of the (k+1)-

order comparator is quite low. When $0 < t \leq k$, an error has occurred in the accessed or the search tag. The probability an error to have occurred in the accessed or search tag is very small thus the (k+1)-order comparator rarely has static power consumption. In the case that $k < t$ the search tag and the accessed tag correspond to different blocks of main memory. In this case we have to distinct direct mapped caches and f-way set associative caches. In direct mapped caches just one (k+1)-order comparator is used. Since direct mapped caches with cache sizes greater than 8 Kbytes have miss ratio m, $m \leq 6.6\%$ [15, p.421] we conclude that only for the m% of the comparisons we will have static power consumption in the (k+1)-order comparator. In f-way set associative caches f (k+1)-order comparators are used. The usual value of f is 2 or 4. Then for each search f-1 (k+1)-order comparators consume static power, also another (k+1)-order comparator in the m% (in this case $m < 5.4\%$) [15, p.421] of the cases consume static power. Therefore, for f-way set associative caches the static power consumption of the (k+1)-order comparators is significant.

The implementation of the k-EC/d-ED code using a (k+1)-order comparator is also suitable in broadcast networks where in order to cope with errors occurring during the transfer of the packets the destination address is encoded in a k-EC/d-ED code. The classical implementation of the code implies that each host includes a k-EC/d-ED error decoder where the possible errors in the destination address are corrected and then the corrected destination address is compared with the host address. On the contrary using the (k+1)-order comparator the encoded destination address is compared in each host with the host address encoded in the k-EC/d-ED code. If the compared addresses differ in less than or equal to k bit positions a match is signalled. It is obvious that the two operands under comparison rarely differ in less than or equal to k bit positions, while very often are identical or differ in more than k bit positions.

From the above analysis we conclude that in all mentioned applications the two operands under comparison rarely differ in t bit positions, with $0<t\leq k$. On the contrary there are applications that the two operands which are compared may very often differ in more than k bit positions.

Taking into account the above and the fact that power consumption is emerging as a major design constrain in several digital applications two new designs for the k-order comparator are proposed in this paper. The first design of the k-order comparator exhibits static power consumption only in the case that the operands differ in less than k bit positions. This case as we have already shown in all applications is rare. The other design of the k-order comparator never exhibits static power consumption.

II . New Designs.

Figure 2 presents the k-order comparator proposed in [16]. In [16] also a systematic method to design module D was given which will be also used in the new designs A and B of Figures 3 and 4.

A. Design A.

The circuit of Figure 3 has two phases of operation, the precharge and the evaluation phase, which are determined by the clock input "clk". In the precharge phase (clk=Low) t_1 conducts while t_7 stops conducting, so line "com" is charged (High). In the same time line "feed" is Low so t_1 remains conductive. In the evaluation phase (clk=High) t_7 conducts. Then if less than k input lines X_i $i \in \{1..n\}$ are High, line "feed" remains Low and so t_1 remains conductive. When k or more input lines X_i $i \in \{1..n\}$ are High then line "feed" becomes High and since t_6 is not conductive in this phase, t_1 will stop conducting. Subsequently, line "com" will drop to 0 volts (discharge) and the output of the circuit (OUT) will become Low.

From the above analysis it is obvious that the only case that there is a conductive path from Vdd to Gnd is during the evaluation phase when t inputs X_i $i \in \{1..n\}$ with $1 \leq t < k$ are High. For the minimization of the total power consumption, the lines $X_1, X_2, \ldots, X_n$ are recommended to be stable during the evaluation phase, and to change during the precharge phase, because then t_7 is not conductive.

B. Design B

The circuit of Figure 4 has low static power consumption, even when the weight of the input vector $X_1, X_2, \ldots, X_n$ has weight less than k. Figure 4 has also two phases of operation, the precharge and the evaluation phase. The precharge phase of the circuit is similar the precharge phase of the circuit in Figure 3. The main difference of these circuits lies on the evaluation phase. In Figure 5 we see the signal of the line "trig" in relation with the clock. We observe that on the rising edge of the clock signal, line trig goes to Low for a small time period. Consequently, line "res" goes to High and the transistor t_7 is conducting during this small time period. If k or more of the transistors $q_1, q_2, \ldots, q_n$ are conductive, then line "feed" goes to High and t_1 stops conducting, so line "OUT" goes to Low and after the returning of line "trig" to High, t_7 will keep to conduct because "OUT" is Low. If less than k of the transistors $q_1, q_2, \ldots, q_n$ are conductive then during the event of line "trig", line "feed" will remain Low and line "OUT" High, so when trig return to High t_7 will stop to conduct. In other words, the circuit of Figure 4 triggers an event on the rising edge of the clock, during which both t_1 and t_7 transistors are conductive

and the circuit evaluates its final state. At the end of this event, the output line "OUT" determines if the transistor t_7 remains conductive or not.

It is obvious that the duration T_{event} of the event on line "trig" must be sufficient for the circuit to come to its steady state. During T_{event} both transistors t_1 and t_7 are on if less than k input lines $X_1 \ldots X_n$ are High, so if T_{event} is close to the half of the period of clock then the gain in power compared with the circuit of Figure 4 is insignificant. In other words, this circuit is useful only when the half period of the clock is much larger than T_{event}, or equivalently the circuit does not operate near to its maximum frequency.

III Experimental Results

For our experiments we used the following tools :

Alliance Cad System 3.0 (Graphic Layout Editor V1.10, Netlist extractor V1.10).

Cazm: circuit analyser using macromodeling.

Sigview: X11 tool for displaying Analog and Digital Simulation Data.

The technology that we used for our experiments is the SCN08H with minimum feature size 1.0 micron. Some typical values for this technology are V_{tn}=0,7522 volts V_{tp}=-0,8433 volts, $KP_n=1.207 \cdot 10^{-4}$ and $KP_p=3.434 \cdot 10^{-5}$. The design of module D was based on the following equation [16]

$$(k-1)\cdot\frac{KP_n}{KP_p}\cdot\frac{2\cdot(V_{dd}-V_{tn})\cdot V_{IHMIN}-V_{IHMIN}^2}{(V_{dd}+V_{tp})^2}\le\frac{W}{L}\le k\cdot\frac{KP_n}{KP_p}\cdot\frac{2\cdot(V_{dd}-V_{tn})\cdot V_{ILMAX}-V_{ILMAX}^2}{(V_{dd}+V_{tp})^2}$$

Table 1 presents the static power dissipation of the 2 order comparator designs of the figures 2, 3 (Design A) and 4 (Design B) respectively, for various values of W/L. We can see that for the circuit of figure 2 when one or more transistors are conductive we have static power consumption, while the circuit of figure 3 consumes static power only when we have one conductive transistor. For the circuit of figure 4 we never have static power consumption.

IV. Test Vector Generator

In this section we prove that k-order comparator has the self testing property, and we give the input vectors that are sufficient to detect all the faults from the fault classes FC={stuck-at 1, stuck-at 0, transistor stuck-on, transistor stuck-open}. We also give the generator that produces these input vectors.

Test vector generator (Figure 1) consists of two n-bit shift registers A and B (Figure 6) where n is the length of each input vector. In each shift register the output of the last cell drives the input of the first cell through an inverter. Shift register B is initialized with the all-zero state while the shift register A is initialized with the state (1....10.....0) where the count of ones and zeroes is respectively equal to k and n-k and the least significant bit is at the left.

When the XOR gates receive as inputs the vectors produced by the shift registers A and B, they generate a sequence of vectors with Hamming weight k and k-1 alternatively. Therefore, during the test phase when D is fault free its output will be in turn equal to zero and one. Taking into account the above and Figure 1 we conclude that the module CNCI can be realised by a flip-flop. This flip-flop changes its state at every clock pulse and generates the sequence 0101. The period of the clock input of the flip-flop should be half the period of the clock input of the shift registers A and B. Then during the test phase and for fault free operation the outputs Z_o, Z_1 will be double-rail encoded. We can easily see that the self-exercising k-order comparator of length n is tested by a test set consisting of 4n vectors.

In the subsections A and B we prove that the test vectors generated by the generator of Figure 6 are sufficient to detect all faults, belonging to FC, of the designs of Figures 3 and 4 respectively and we give the corresponding self exercising checker structures, similar to that of Figure 1, for the power optimized designs of Figures 3 and 4.

A. Testing of Design A

During logic testing the decision whether the circuit is faulty or not is based on the logic value of the output of the circuit. Specifically we apply a vector at the inputs of the circuit and we check the output. If the output has the value that we expect, then the circuit is error free or else it is faulty. Logic testing can cover a large set of faults for a circuit, but still there are some faults that can not be detected with this method. Such faults do not change the expected logic value of the output, but raise the power consumption of the circuit. Such faults are detected with I_{ddq} testing [17].

I_{ddq} testing can detect a fault, when this fault causes an unusually high current from Vdd to Gnd when the circuit is in the steady state. In this paper we use the Built-In Current Sensor (BICS) proposed in [18].

In the case of Figure 3, I_{ddq} testing can be used to detect faults that cause an unusual high current from Vdd to Gnd, but also certain faults that prevent the high current from Vdd to Gnd, due to a specific property of the circuit :

Lets assume that w of the transistors $q_1, q_2, \ldots, q_n$ are conductive. When $w \geq k$ the path from Vdd to Gnd is cut because t1 is not conductive, so the steady state current is quite low. When $1 \leq w < k$ there is a conductive path from Vdd to Gnd because both t_1 and t_7 transistors are conductive, so the steady state current is high.

The above property, leads us to the conclusion that BICS can be used to detect faults that cause high current leakage from Vdd to Gnd when $w \geq k$, or faults that prevent the flow of high current from Vdd to Gnd when $1 \leq w < k$. So if we assume that BICS output is High (5volts) when the current of the circuit is above the threshold, and Low (0 volts) when the current is below the threshold, then for the fault free operation of the circuit BICS gives :

1. BICS OUT=Low a. during precharge phase for $0 \leq w \leq n$ and
   b. during evaluation phase for $w \geq k$
2. BICS OUT=High during evaluation phase for $1 \leq w < k$.

We have to note here that in this case the I_{ddq} testing is used in a new, different way than the classical one [17]. According to the classical I_{ddq} testing, high current implies the existence of a fault in the circuit under test.

Experiments have shown that for the cases 1a and 1b, the current flowing from Vdd to Gnd is in the order of 10μA while for the case 2 the current is in the order of 3mA, so for the BICS we can use a threshold of 1mA.

Taking under consideration the above, we modify the structure of Self Exercising checker of Figure 1 in order to add the BICS sensor to the circuit and so we get the structure of Figure 7. In Figure 7 we have added a combinational circuit that gives another pair of 2-rail outputs (Y_0, Y_1) based on the output of BICS sensor. This circuit is self testing for the 4n vectors generated by the test vector generator. The truth table of the added combinational circuit is shown in Table 2.

In Table 3 we see the fault set F of the circuit of Figure 3. For each fault we give two detection methods, one with "logic testing" and the other with I_{ddq} testing. For the method of "logic testing" and for each fault we give the weight of the necessary input vector, the phase of the clock where the fault is detected and the output pair (Y_0, Y_1) of the circuit (which is not a two-rail code word). For the "I_{ddq} testing" method we give the weight of the input vector, the phase of the clock and the output of the sensor BICS. In some cases we give two vectors that should be applied sequentially in two successive clock periods.

We must notice here that the input vectors must be applied at the beginning of the precharge phase and preserved until the end of the evaluation phase.

B. Testing of Design B

Following the same analysis with B we give Table 4 which contains the same information with Table 3 for the circuit of Figure 4. Based on this table we modify the structure of SE k-order comparator of Figure 1, in order to add the BICS sensor in our design. The new modified circuit is shown in Figure 8.

References

[1] M. Nicolaidis, "Self-Exercising Checkers for Unified Built-in Self-Test (UBIST)", IEEE Trans. on CAD, Vol. 8, No 3, March 1989.

[2] B. Bose and D.K. Pradhan, "Optimal unidirectional error detecting correcting codes", IEEE Trans. Comput., Vol.C-31, pp.564-568, June 1982.

[3] D.J. Lin and B. Bose, «Theory and Design of t-Error Correcting and d (d>t)-Unidirectional Error Detecting (t-EC/d-UED) Codes», IEEE Trans. Comput., April 1988, pp. 433-439.

[4] T.R.N. Rao, E. Fujiwara, "Error-Control coding for computer systems." Prentice-Hall International.

[5] M. Blaum and H.V. Tilborg, "On t-Error Correcting/All Unidirectional Error Detecting Codes", IEEE Trans. Comp. Nov. 1989, pp. 1493-1501.

[6] D. Nikolos, "Theory and Design of t-Error Correcting/d-Error Detecting (t<d) and All Unidirectional Error Detecting Codes." IEEE Trans. Comp., Feb. 1991, pp.132-142.

[7] D. Nikolos, and A. Krokos, "Theory and Design of t-Error Correcting, k-Error Detecting and d-Unidirectional Error Detecting Codes with d>k>t.", IEEE Trans. on Comput., April 1992, pp. 411-419.

[8] Jien-Chung Lo and Eiji Fujiwara, "Probability to Achieve TSC Goal", IEEE Trans. on Comput., April 1996.

[9] H.T. Vergos and D. Nikolos, "Efficient Fault Tolerant Cache Memory Design", Micropr. and Microprogr., The Euromicro Journal, 41 (1995) pp. 153-169.

[10] W. C. Carter and P. F. Schneider, "Design of dynamically checked computers", in Proc. 4th Cong. IFIP, Edinburgh, Scotland, vol. 2, pp. 878-883, Aug. 5-10, 1968.

[11] D. A. Anderson, "Design of self-checking networks using coding techniques", Coord. Sci. Lab., Univ. Illinois, Urbana, IL, Tech. Rep. R-527, 1971

[12] M. Nicolaidis and B. Courtois, "Strongly Code Disjoint Checkers", IEEE Trans. Comput., June 1988.

[13] Neil H. E. Weste, Kamran Eshraghian, "Principles of CMOS VLSI Design, A systems Perspective" , 2nd ed., Addison Wesley.

[14] V.G. Oklobdzija and P.G. Konijanic, "On testability of CMOS-domino logic, " in Proc. 14th Int. Symp. Fault-Tolerant Comput., June 1984.

[15] J.L. Hennessy & D.A. Patterson, "Computer Architecture a Quantitive Approach", Morgan Kaufmann Publishers Inc.

[16] X. Kavousianos, D. Nikolos "Self-Exercising Self Testing k-order Comparators" 15th IEEE VLSI Test Symposium, California April 27 - April 30, 1997

[17] Jerry M. Soden, Charles F. Hawkins, Ravi M. Gulati and Weiwei Mao " I_{DDQ} Testing: A Review " Journal Of Electronic Testing: Theory and Applications 3, 291-303 (1992).

[18] Tung-Li Shen, James C. Daly and Jien-Chung Lo, 'A 2-ns Detecting Time, 2-μm CMOS Built-in Current Sensing Circuit ' IEEE Journal of Solid-State Circuits, Vol.28, No 1, January 1993

Table 1 Power Consumption (μWatts)

W/L	Figure 2			Figure 3			Figure 4		
	Conductive nmos			Conductive nmos			Conductive nmos		
	0	1	2	0	1	2	0	1	2
4	45	3119	4098	46	3196	47	50	58	48
4.5	45	2840	4479	46	2844	47	50	58	48
5	45	2524	6122	46	2491	47	50	57	48
5.5	45	2511	7432	46	2844	47	50	57	48
Static Consumption of 16 bit 2-order comparator designs									

Table 2

TEST	CLK	BICS	CNSI	Y0	Y1
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	1	1
1	0	0	0	0	1
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	1	1	1
1	1	0	0	0	1
1	1	0	1	0	0
1	1	1	0	1	1
1	1	1	1	1	0

Table 3

	Logic Testing			Iddq Testing		
Fault	Weight	Clock	(Z_0,Z_1)	Weight	Clock	(Y_0,Y_1)
clk/1	(k,k-1)	1	(1,0) (0,0)	k-1	1	(0,0)
clk/0	k	1	(1,1)	k-1	1	(0,0)
cp1/1	(k,k-1)	1	(1,0) (0,0)	k-1	1	(0,0)
cp1/0	k	1	(1,1)	k	1	(1,1)
cp2/1	k-1	1	(0,0)	k	1	(0,0)
cp2/0	k	1	(1,1)	k-1	1	(0,0)
cp~/0	(k,k-1)	1	(1,0) (0,0)	(k,k-1)	1	(0,1) (0,0)
cp~/1	k	1	(1,1)	k	1	(1,1)
lp/1	k-1	1	(0,0)	k-1	1	(0,0)
lp/0				k	1	(1,1)
lfeed/1	k-1	1	(0,0)	k-1	1	(0,0)
lfeed/0	k	1	(1,1)	k	1	(1,1)
lcom/1	k	1	(1,1)	k-1	1	(0,0)
lcom/0				k	0	(0,0)
out/1	k	1	(1,1)			
out/0	k-1	1	(0,0)			
t1-on				k	1	(1,1)
t1-open	k-1	1	(0,0)	k-1	1	(0,0)
t2-on						
t2-open	k	1	(1,1)	k	1	(1,1)
t3-on	k	1	(1,1)	k	1	(1,1)
t3-open						
t4-on				k	1	(1,1)
t4-open	(k,k-1)	1	(1,0) (0,0)			
t5-on	k-1	1	(0,0)			
t5-open	k	1	(1,1)			
t6-on	k	1	(1,1)	k	1	(1,1)
t6-open	(k,k-1)	1	(1,0) (0,0)	k-1	1	(0,0)
t7-on				k	0	(0,0)
t7-open	k	1	(1,1)	k-1	1	(0,0)
t8-on				k	1	(1,1)
t8-open	(k,k-1)	1	(1,0) (0,0)	(k,k-1)	1	(0,1) (0,0)
t9-on	(k,k-1)	1	(1,0) (0,0)	(k,k-1)	1	(0,1) (0,0)
t9-open	k	1	(1,1)	k	1	(1,1)
q_i-on	k-1	1	(0,0)	k-1	1	(0,0)
q_i-open	k	1	(1,1)	k	1	(1,1)
$x_i/1$	k-1	1	(0,0)	k-1	1	(0,0)
$x_i/0$	k	1	(1,1)	k	1	(1,1)

Table 4

	Logic Testing			Iddq Testing		
Fault	Weight	Clock	Out	Weight	Clock	BIC out
clk/1	(k-1,k)	1	(0,1) (1,1)			
clk/0	k	1	(1,1)			
cp~/0	(k-1,k)	1	(0,1) (1,1)			
cp~/1	k	1	(1,1)	k or k-1	1	(1,1)
clk_1/1	(k-1,k)	1	(0,1) (1,1)			
clk_1/0	k	1	(1,1)	k or k-1	1	(1,1)
clk_2/1	k	1	(1,1)	k or k-1	0	(1,1)
clk_2/0	k	1	(1,1)			
$cp\sim_1$/1	k	1	(1,1)			
$cp\sim_1$/0	(k,k-1)	1	(1,0) (0,0)			
$cp\sim_2$/1				k or k-1	1	(1,1)
$cp\sim_2$/0	k	1	(1,1)			
trig/1	k	1	(1,1)			
trig/0				k or k-1	1	(1,1)
res/1				k or k-1	1	(1,1)
res/0	k	1	(1,1)			
lcom/1	k	1	(1,1)			
lcom/0				k or k-1	1	(1,1)
lfeed/1	k-1	1	(0,0)	k or k-1	0	(1,1)
lfeed/0	k	1	(1,1)			
lp/1	k-1	1	(0,0)	k or k-1	0	(1,1)
lp/0				k	1	(1,1)
out/1	k	1	(1,1)	k	1	
out/0	k-1	1	(0,0)	k or k-1	0	(1,1)
x_i/1	k-1	1	(0,0)			
x_i/0	k	1	(1,1)			
t1-on				k	1	(1,1)
t1-open	k-1	1	(0,0)			
t2-on		1		k or k-1	1	(1,1)
t2-open	k	1	(1,1)			
t3-on	k	1	(1,1)	k	1	(1,1)
t3-open						
t4-on				k or k-1	1	(1,1)
t4-open	(k,k-1)	1	(1,0) (0,0)			
t5-on	k-1	1	(0,0)	k-1	1	(1,1)
t5-open						
t6-on	k	1	(1,1)			
t6-open	(k,k-1)	1	(1,0) (0,0)			
t7-on				k or k-1	0	(1,1)
t7-open	k	1	(1,1)			
q_i-on	k-1	1	(0,0)			
qi-open	k	1	(1,1)			

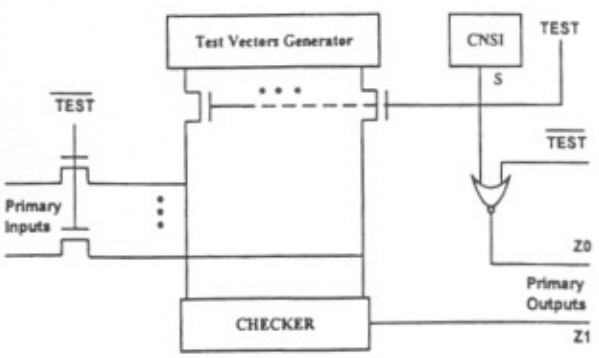

Figure 1. Structure of a Self-Exercising checker.

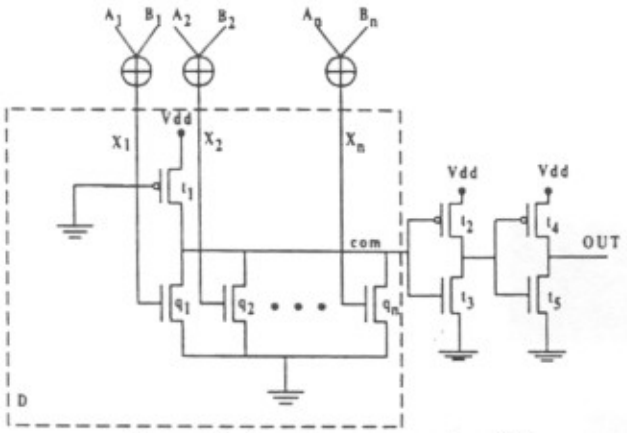

Figure 2. k-order comparator [16]

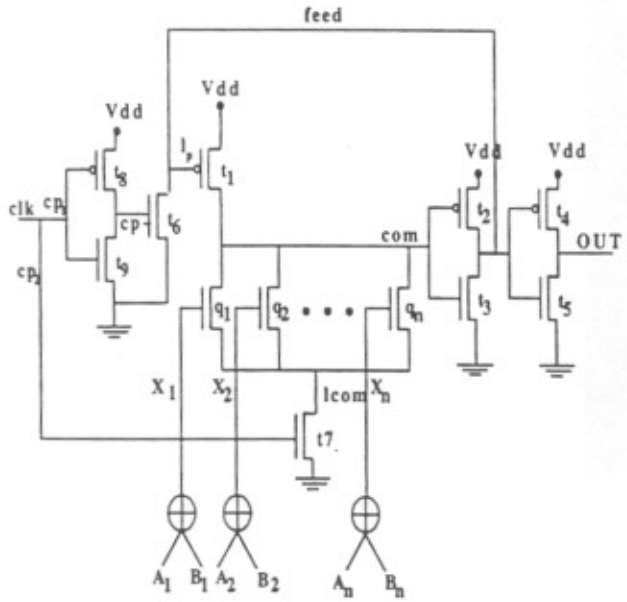

Figure 3 Power optimized k-order comparator.

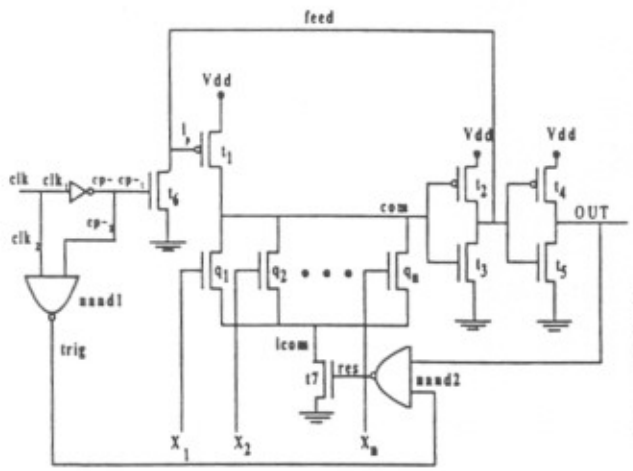

Figure 4 Power optimized k-order comparator

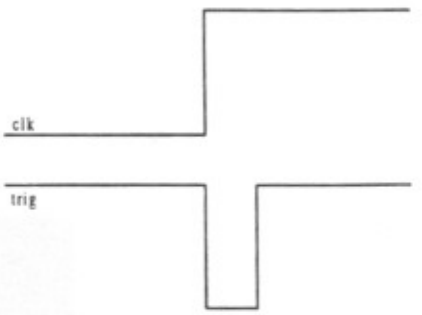

Figure 5 Signal of line "trig"

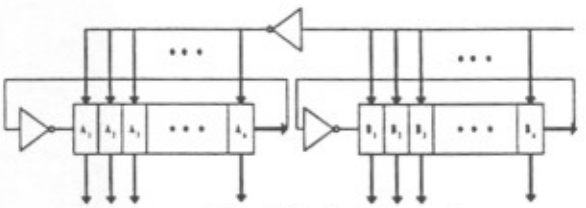

Figure 6 Test Vector Generator

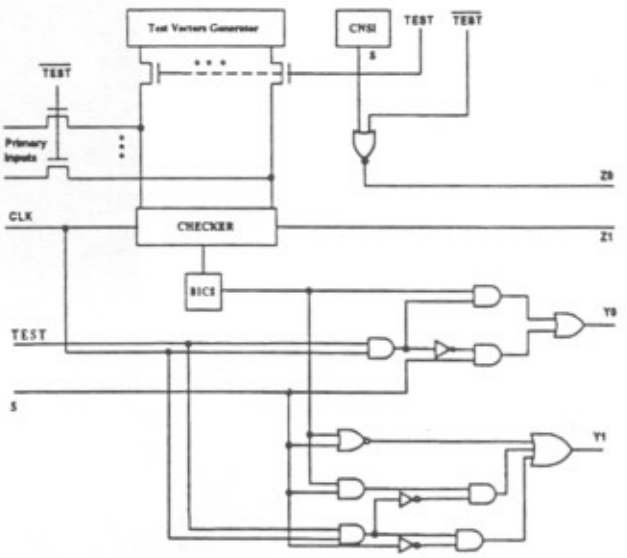

Figure 7

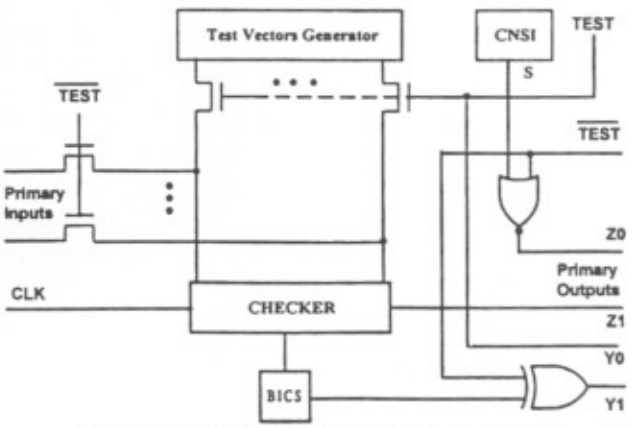

Figure 8 SE-checker for circuit of Fig. 4