Testing Priority Address Encoder Faults of Content Addressable Memories

Jin-Fu Li
Advanced Reliable Systems (ARES) Laboratory
Department of Electrical Engineering
National Central University
Jungli, Taiwan, R.O.C.

Abstract

Content addressable memory (CAM) is one key component in many digital systems. Although the CAM cell usually is implemented with a RAM cell and a comparison logic, the CAM testing is more difficult than the RAM testing. Also, the CAM testing is very different from the RAM testing. Most stuck-at faults (SAFs) in the RAM peripheral circuitry can be mapped to the RAM cell faults. This cannot be analogous to the testing of the priority encoder of CAMs. This paper presents a test algorithm for testing SAFs of the priority encoder in a CAM. The test algorithm only requires $3N$-2 Write operations and N+2 Compare operations to cover 100% stuck-at faults of the CMOS priority encoder of an $N \times B$-bit CAM. Compared with typical tests for CAM cell array faults, the fault coverage of SAFs in the priority encoder is increased from 90.2% or 60.5% to 100% for a CAM with 64 words.

Keywords: *Content addressable memories, comparison faults, priority address encoder faults.*

1 Introduction

Content addressable memories (CAMs) are widely used in digital systems, such as networking, cryptography, compression, and so on. A CAM cell consists of a storage cell (e.g., SRAM cell or DRAM cell) and a comparison logic, such that a CAM can execute the function of parallel search (compare). Thus, functional faults of a CAM cell array usually consists of RAM faults and Comparison faults [1–3]. Testing Comparison faults of a CAM is more difficult than testing RAM faults of a CAM, since the fault effects of Comparison faults must be observed by the comparison results (the output of Hit Signal Generator and/or Priority Encoder). This results that the CAM testing is more difficult than the RAM testing.

Many CAM test schemes have been reported before (see, e.g., [1–17]). A test methodology for detecting stuck-at faults in the memory management unit was reported in [4]. The BIST scheme presented in [5] is used to detect stuck-at faults, adjacent cell coupling faults, and the neighborhood pattern sensitive faults (NPSFs) of dynamic CAMs. A design-for-testability (DFT) circuitry also is added to the Hit signal generator to determine whether there is a hit for all the even match lines, and separately for all the odd match lines. In [6], a BIST circuit to test a reconfigurable CAM was proposed. The test circuit only covers approximately 75% stuck-open faults and comparison faults in the array. The BIST circuit [7] is designed to test a cache memory (including the CAM and RAM blocks) and incorporates a serial interface into the cache such that it can execute a complete SMARCH algorithm [18] on the CAM Read/Write port. Another BIST architecture with a parallel test approach was reported in [9]. It reduces the testing time by modifying the address decoder such that the parallel test approach can be used to detect coupling faults and NPSFs. In [11], a specific test algorithm is used to test a CAM whose priority encoder returns the lowest matched address. A fault effect can be masked if the fault occurs on a word whose address is higher than the matched address. A match compare circuit is added to each match line output such that the array can be fully tested by the BIST circuit.

In [8], a functional fault model for CAMs was derived by investigating the functional failures in the storage cell and the comparison logic. In [10], an approach for modeling and testing memories and its application to CAMs was presented. March-like test algorithms presented in [1] are used to detect the CAM-specific comparison faults. Assume that the comparison result is observed only by the Hit output, the test algorithms can cover 100% comparison faults. But the fault coverage of conventional RAM faults is low. In [12], test algorithms for CAMs which can execute Read and Compare operations concurrently were proposed. Thus, the target CAM has the basic cells with individual bit lines and comparison lines. The authors also assume that the comparison result is observed by the priority encoder. In [13], a test methodology for testing the delay faults of CAMs was presented. The test algorithms and fault location algorithms reported in [2, 3] can detect 100% typical RAM faults and CAM-specific comparison faults. Here the CAM with Read and Compare operations are assumed. Furthermore, a diagnosis scheme was used to distinguish different types of RAM faults and comparison faults [15].

So far, the mentioned previous works all focus on the testing of binary CAMs. Recently, several works discuss the testing of ternary CAMs. Different from binary CAMs, a ternary CAM cell can store two binary values to represent one of three logic states: 0, 1, and don't care. In [14], a test algorithm was proposed to detect the search path failures of ternary CAMs based on transistor-level faults. In [16], the author presents a test algorithm for comparison faults of ternary CAMs based on comparison faults of binary CAMs. In [17], comparison faults based on short and open defects was defined. A test algorithm for testing these comparison faults also was developed.

The previous works described above all develop the test algorithms for the CAM cell array faults, i.e., comparison faults and RAM faults. However, the CAM testing is very different from the RAM testing. In addition to the Comparison faults that must be covered, testing CAM peripheral circuitries also cannot be analogous to testing RAM peripheral circuitries. In RAMs, most of stuck-at faults (SAFs) in peripheral circuitries can be mapped to the cell array faults. But this is not true for CAMs, SAFs in some peripheral circuitries cannot be mapped to the cell array faults. That is, it is not enough if only the test algorithms for cell array faults are applied to test the CAMs.

This paper presents a test algorithm for testing SAFs of the priority encoder in a CAM. The test algorithm only requires $3N$-2 Write operations and N+2 Compare operations to cover 100% SAFs of the CMOS priority address encoder of an $N \times B$-bit CAM. Compared with the conventional tests for CAM cell array faults, the fault coverage of SAFs in the prefix computation logic of the priority encoder is increased from 90.2% or 60.5% to 100% for a CAM with 64 words.

The rest of this paper is organized as follows. Section 2 overviews the architecture of a typical CAM and tests for CAM cell array faults. Section 3 describes the proposed test algorithm for the Priority Encoder. Section 4 explains the testing of Hit Signal Generator. Section 5 summarizes the simulation results of fault coverage. Finally, Sec. 6 concludes this paper.

2 Preliminary

2.1 Typical CAM Architecture

Figure 1 shows a typical CAM architecture. The Address Decoder and Data I/O are similar to those in a RAM. When the CAM executes a Compare operation, the compared data (comparand) is prefetched into the Comparand Register. The Mask Register stores a binary pattern determining whether the corresponding bits in a word are to be masked from further Write and Compare operations or not. Each of valid bits (VBi) indicates whether the match signal of the corresponding word is valid or invalid. The Hit Signal Generator evaluates the valid match signals, and generates a hit output (Hit=1) if there is at least one valid match. The Priority Encoder exports the highest priority matched address (either the lowest matched address or the highest matched address).

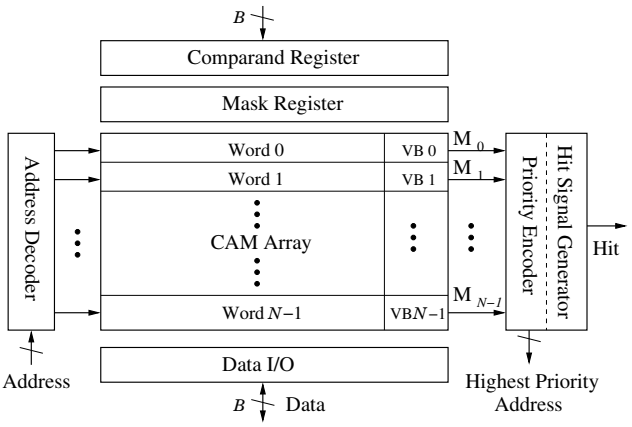

Figure 1: A typical CAM architecture.

Figure 2 shows a B-bit CAM word with NOR-type match line. Each cell is composed of an SRAM cell and a comparison logic (formed by transistors T3, T4, and T5) [19], such that the CAM can perform Compare operation simultaneously. Bit line and search line of the CAM cell share the same line (BL_i/SL_i). Also, $\overline{BL_i}/\overline{SL_i}$ denotes the complement of BL_i/SL_i. A CAM usually has the following basic operations: Write, Read, Compare, Erase, and Masked Compare. The Read and Write operations are the same as those of a RAM. The Masked Compare operation compares an input pattern with all words in the CAM simultaneously, with one or more bits blocked (not compared) by setting the corresponding bits of the mask pattern. The Masked Write operation writes an input pattern to a specified word, with one or more bits blocked (not written) by setting the corresponding bits of the mask pattern. The Write operations also set the valid bit of the corresponding word so that it is in the valid state. The word-line pass transistors (T1 and T2) are turned off when the CAM executes the Compare and Masked Compare operations. The match lines (M_i) are precharged to V_{dd} before the Compare operation, by resetting the Precharge signal. The input pattern is then compared with the all the CAM words simultaneously. If the pattern stored in any word is the same as the input pattern, the corresponding match signal will be high (1) since all the T5 transistors of the word are turned off. Also, the Hit signal is 1. The Erase operation resets the corresponding Valid Bit Flip-Flop of a specified word.

2.2 Tests for CAM Cell Array Faults

CAM cell array major consists of two types of faults: RAM faults and Comparison faults [1]. Testing RAM faults of a CAM is similar to that of a RAM. Typical RAM faults, such as stuck-at faults and coupling faults, can be detected by March tests which consists of Read and Write test operations [2,3]. However, testing Comparison faults of a CAM

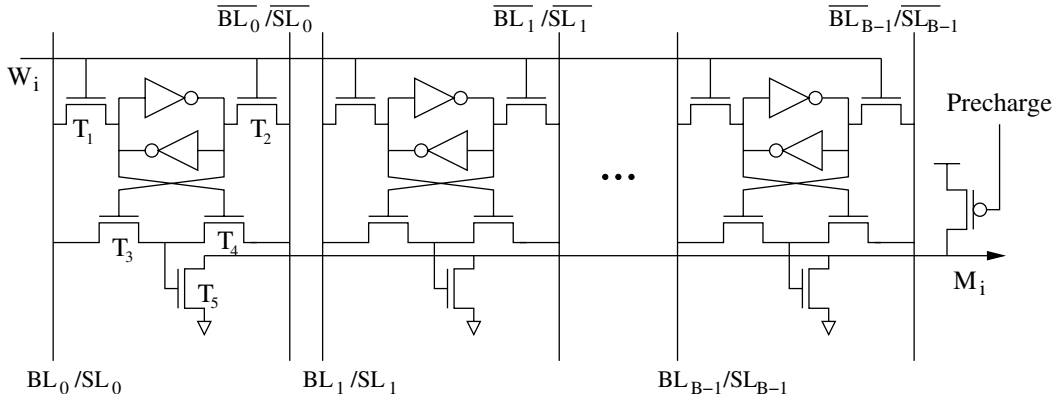

Figure 2: An NOR-type CAM word with B-bit cells.

needs a test algorithm (test) which consists of Write and Compare test operations. The Compare operation is used to observe the fault effect of Comparison faults through the Priority Encoder and/or Hit Signal Generator.

When the test algorithms reported in the previous works [1–3, 12, 13] for Comparison faults are applied to the CAM under test, the expected responses observed by the match signals (M_0,M_1,. . .,$M_{N\text{-}1}$) can be classified into four types of response patterns shown in Table 1. As the table shows, the second row denotes that N Compare operations are performed and N corresponding expected responses are called Type-1 response patterns. The last column denotes the fault-free output of comparison results. In this case, the comparison results are observed by the Hit Signal Generator. The third row also shows N expected responses with respect to N Compare operations, but the output of comparison results are observed by the Priority Encoder with exporting the highest matched address. On the contrary, if the lowest matched address is exported, the corresponding expected match signals are shown in the fourth row. The last row shows an all-0 expected response when a Compare operation is executed and the comparison result is propagated to the Hit output.

3 Testing Priority Encoder Faults of CAMs

3.1 Priority Encoder

When a CAM performs a Compare operation, multiple matches may occur. To identify only one matched word, a priority encoder usually is used to block the matched words with lower priorities and export the address of the match word with the highest priority. The highest priority may be the highest address or the lowest address. Without loss of generality, we assume that the least significant bit of the input corresponds to the highest priority, i.e., the lowest matched address, in this paper. Figure 3 shows the architecture of a typical priority encoder. A priority encoder consists of a prefix computation logic (PCL) and an encoder. Assume that N=2^n. The 2^n-to-n encoder only has an assertive logic value, either 0 or 1, on one of 2^n input lines and causes the corresponding binary code to appear at the output, Address. Thus, the encoder can easily be tested with functional patterns, i.e., walking-1 patterns. Therefore, we first discuss the testing of PCL.

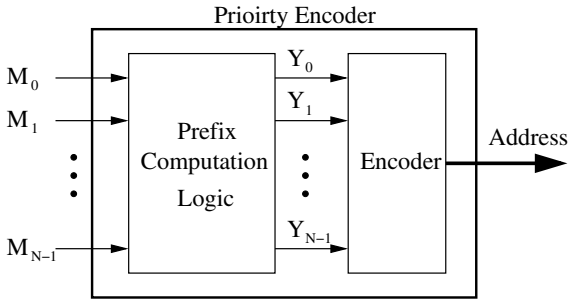

Figure 3: A typical priority encoder architecture.

Let inputs and outputs of the PCL be $\{M_0, M_1, \ldots, M_{N\text{-}1}\}$ and $\{Y_0, Y_1, \ldots, Y_{N\text{-}1}\}$, respectively. Then the boolean function of the PCL can be expressed as [20]:

$$\begin{aligned} Y_0 &= M_0 \\ Y_1 &= M_1 \cdot \overline{M}_0 \\ Y_2 &= M_2 \cdot \overline{M}_1 \cdot \overline{M}_0 \\ &\vdots \\ Y_{N-1} &= M_{N-1} \cdot \overline{M}_{N-2} \cdot \ldots \cdot \overline{M}_0 \end{aligned}$$

Let A_i=$\Pi_{j=0}^{j=i-1}\overline{M}_j$, then Y_0=M_0 and Y_i=$A_i \cdot M_i$ for $1 \leq i \leq N$-1 and $i \in$N. Thus, the PCL is usually realized by a two-stage logic circuit including a group logic and a output logic. The output logic is implemented by N-1 2-input ANDs which generate N-1 outputs Y_i for $1 \leq i \leq N$-1. Also, the inputs of each 2-input AND are A_i and M_i. The group logic realizes the function of A_i for $1 \leq i \leq N$-2. Different group networks (e.g., ripple, lookahead, increment, tree, etc. [20]) can be used to build the group logic. For example, Fig. 4 shows a PCL implemented with ripple group logic which is comprised of the AND gates in shade.

Table 1: Types of expected responses observed by the match signals when Compare operations are executed in an $N \times B$-bit CAM.

# Compare Operations	Expect responses (M_0,M_1,. . .,M_{N-1})	Type	Result output
N	(1,0,0,. . .,0,0,0) (0,1,0,. . .,0,0,0) (0,0,1,. . .,0,0,0) ⋮ (0,0,0,. . .,1,0,0) (0,0,0,. . .,0,1,0) (0,0,0,. . .,0,0,1)	Type-1	Hit
N	(1,0,0,. . .,0,0,0) (1,1,0,. . .,0,0,0) (1,1,1,. . .,0,0,0) ⋮ (1,1,1,. . .,1,0,0) (1,1,1,. . .,1,1,0) (1,1,1,. . .,1,1,1)	Type-2	Highest Matched Address
N	(0,0,0,. . .,0,0,1) (0,0,0,. . .,0,1,1) (0,0,0,. . .,1,1,1) ⋮ (0,0,1,. . .,1,1,1) (0,1,1,. . .,1,1,1) (1,1,1,. . .,1,1,1)	Type-3	Lowest Matched Address
1	(0,0,0,. . .,0,0,0)	Type-4	Miss

3.2 Testing Stuck-At Faults of PCL

According to the description above, we divide the testing of PCL into two parts: testing of group logic and testing of output logic. In this paper, we consider the PCL testing based on single stuck-at fault (SAF). The testing of output logic is first discussed. The output logic consists of N-1 2-input AND gates. It is well known that the test patterns for SAFs of a 2-input AND are $\{11,01,10\}$. We denote the ith AND gate of the output logic as AND_O$_i$ which output is Y_i and inputs are A_i and M_i. Table 2 lists the test patterns for detecting SAFs of the output logic. If the all-1 test pattern is applied to the input of the PCL, each AND_O can receive the test pattern$\{01\}$. The AND_O$_1$ can receive the test $\{11\}$ if the input of the PCL receives the test pattern (01XX. . .XX), where X denotes don't care. Since $A_2 \sim A_{N\text{-}1}$ are 0s when M_1=1, 0 or 1 can be applied to $M_2 \sim M_{N\text{-}1}$. In the same way, we see that every AND gate of the output logic can receive the test $\{11\}$ when the test patterns shown in the third row of Table 2 are applied. If the all-0 test pattern is applied to the PCL, all AND_O gates receive the test 10, since the values of A_i for $1 \leq i \leq N$-1 all are 1s.

Subsequently, we discuss the testing of group logic. The boolean function of the group logic is $A_i = \Pi_{j=0}^{j=i-1} \overline{M}_j$ for $1 \leq i \leq N$-1, i.e., $A_1 = \overline{M}_0$, $A_2 = \overline{M}_0 \cdot \overline{M}_1$, . . ., $A_{N\text{-}1} = \overline{M}_0 \cdot \overline{M}_1 \cdot \ldots \cdot \overline{M}_{N\text{-}2}$. Thus, the group logic consists of N-1 inputs and N-1 outputs. In this paper we consider the PCL testing based on ripple group logic. As Fig. 4 shows, SAFs at the outputs A_i of the ripple group logic are detected by the test patterns shown in Table 2 since A_i also are the inputs of the output logic. Therefore, only SAFs at the inputs of the group logic need to be covered. That is, we can consider the testing of the ripple group logic as the testing of an AND gate with N-1 inverted inputs ($\overline{M}_0 \overline{M}_1 \ldots \overline{M}_{N-2}$) and a output $A_{N\text{-}1}$. Because $A_{N\text{-}1}$ is not the output of the PCL, the fault effect propagated through the $A_{N\text{-}1}$ must be propagated to the output $Y_{N\text{-}1}$. Since $Y_{N\text{-}1} = A_{N\text{-}1} \cdot M_{N\text{-}1}$, the input $M_{N\text{-}1}$ must have the value of logic 1 such that the fault propagation path is sensitized. Table 3 summarizes the test patterns for detecting SAFs at the inputs of the group logic. As the table shows, if the test pattern $\{000\ldots001\}$ is applied, inputs of the N-1-input AND gate of the group logic (AND_G) receive all-1 data. Then stuck-at-0 faults at the inputs of the AND_G gate can be detected. When the

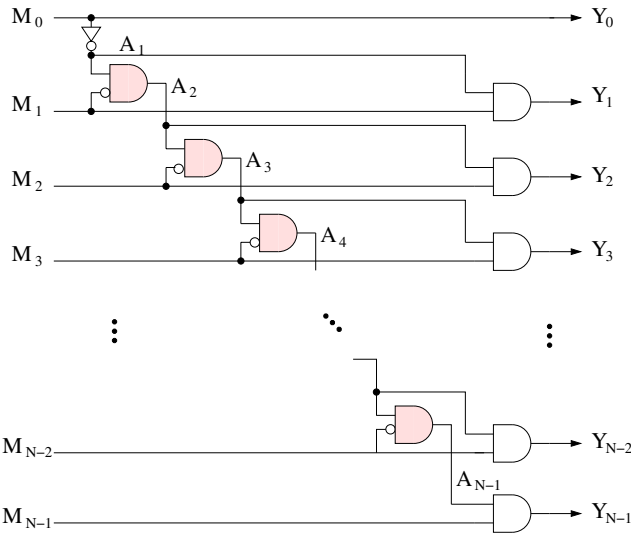

Figure 4: A CMOS PCL with ripple group logic.

Table 2: Test patterns for detecting SAFs of the output logic.

Test pattern ($M_0 M_1 \ldots M_{N-1}$)	Test data ($A_i M_i$) received by AND_O$_i$
(1111. . .1111)	01 received by all AND_Os
(01XX. . .XXXX)	11 received by AND_O$_1$
(001X. . .XXXX)	11 received by AND_O$_2$
(0001. . .XXXX)	11 received by AND_O$_3$
⋮	⋮
(0000. . .1XXX)	11 received by AND_O$_{N-4}$
(0000. . .01XX)	11 received by AND_O$_{N-3}$
(0000. . .001X)	11 received by AND_O$_{N-2}$
(0000. . .0001)	11 received by AND_O$_{N-1}$
(0000. . .0000)	10 received by all AND_Os

test patterns in the second row of Table 3 are applied to the inputs of the PCL, the stuck-at-1 faults at the inputs of the AND_G gate can be detected.

Table 3: Test patterns for detecting SAFs of group logic.

Test pattern $(M_0M_1 \ldots M_{N-1})$	Test data received by the AND_G
(0000...0001)	111...111 received by AND_G
(1000...0001)	011...111 received by AND_G
(0100...0001)	101...111 received by AND_G
(0010...0001)	110...111 received by AND_G
⋮	⋮
(0000...1001)	111...011 received by AND_G
(0000...0101)	111...101 received by AND_G
(0000...0011)	111...110 received by AND_G

For example, if the test pattern shown in the second row of Table 3 is applied to the inputs of the ripple group logic with two-input AND gates as shown in Fig. 4, each two-input AND gate of the group logic receives the test data $\{11\}$. That is, the fault-free output of $A_{N\text{-}1}$. Since $M_{N\text{-}1}$=1, a stuck-at-0 fault existing at any one of the AND gates changes the value of the output $Y_{N\text{-}1}$ from 1 to 0. If the first test pattern shown in the last row of Table 3 is applied, the AND gate with the output A_2 receives the test data $\{01\}$ and the fault-free value of A_2 is 0. Therefore, the AND gates with the outputs A_3, A_4, ..., $A_{N\text{-}1}$ also receive the test data $\{01\}$. If the second (third, ..., last) test pattern shown in the same row is applied, the AND gate with the output A_2 (A_3, ..., $A_{N\text{-}1}$) receives the test data $\{10\}$. Consequently, all the AND gates of the ripple group logic can receive the test data $\{11,10,01\}$. That is, all SAFs of the ripple group logic are covered.

In a similar way, tests for detecting SAFs of the PCLs with the other types of group logics can be developed. For example, Fig. 5 shows an 8-bit PCL with lookahead group logic [20]. As the figure shows, the 8-bit inputs of the PCL is partitioned into two parts for prefix computation. Apparently, all SAFs of the output logic also can be covered by the test patterns shown in Table 2. The testing of lookahead group logic also is similar to that of ripple group logic. We can consider the testing of the lookahead group logic as the testing of two individual group logics: one 5-bit group logic and one 8-bit group logic. Both the group logics can be tested with the test patterns shown in Table 3 by replacing the N with 4 and 7, respectively. Compared with the ripple group logic, the number of test patterns for the lookahead group logic is increased. In the sequel of this paper, we discuss the testing of CAM priority encoder with the ripple group logic. The testing of CAM priority encoder with the other possible group logic can be developed in a similar way.

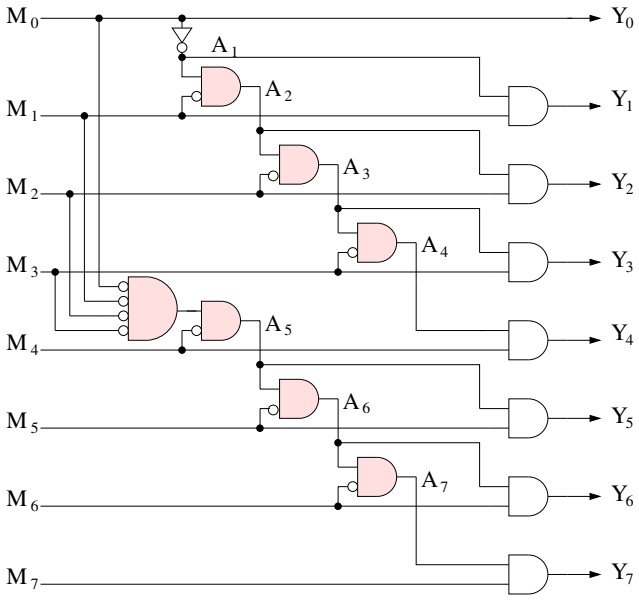

Figure 5: A CMOS PCL with lookahead group logic.

3.3 Test for CAMs with Priority Encoder Faults

According to Tables 2 and 3, we summarize the test patterns for detecting SAFs of the PCL with ripple group logic in Table 4. Thus, we need a test algorithm which can generate the match signal responses as shown in Table 4 on the match lines of a CAM under test. Let the size of the CAM is $N \times B$, where N is the number of words of the CAM and B is the number of bits of a word. Also, assume that the priority encoder exports the lowest matched address when a Compare operation is executed.

Table 4: Test patterns for detecting SAFs of PCL with ripple group logic.

Test pattern $(M_0M_1 \ldots M_{N-1})$
(1111...1111)
(0000...0000)
(0000...0001)
(1000...0001)
(0100...0001)
(0010...0001)
⋮
(0000...1001)
(0000...0101)
(0000...0011)

The proposed test algorithm (T_{PE}) for detecting SAFs of priority encoder faults is depicted in Algorithm 1. The test procedure of T_{PE} can be divided into four steps. Step 1 initializes the CAM under test into the desired state. After Step 1, state of bit B-1 of all words is all-0. Then a Mask Compare operation which compares 0 with bit B-1 of all words of the CAM is executed in Step 2. All words

issue matched signals to the priority encoder and the output matched address is 0, since the state of bit B-1 of all words are all-0. If the output matched address is not 0, the priority encoder is faulty. Similarly, Step 3 executes the operation the same as Step 2 by replacing the compared data with 1. Therefore, comparison results of all words are mismatch, i.e., the inputs of the priority encoder all are 0s. This causes the priority encoder to output an invalid address. If the address output is an valid address, the priority encoder is faulty. Step 4 compares $2^{B\text{-}1}$-1 with all words. Because the data of word i for $0 \le i \le N$-2 are all 0s and the data of word N-1 is $2^{B\text{-}1}$-1, only the content of word N-1 is the same as the compared data. Thus, the priority encoder receives the input pattern the same as the third pattern shown in the second row of Table 4. Then check if this Compare operation results that the priority encoder exports the expected address N-1. Step 5 consists of three test operations, one Compare and two Write operations, in which the Write operations are executed in ascending address sequence. One Write operation writes the data $2^{B\text{-}1}$-1 into the word i and one Compare operation compares $2^{B\text{-}1}$-1 with all words. If the matched address is not i, the priority encoder is faulty. The other Write operation writes all-0 data into the same word. These three operations are repeatedly performed at each word until word N-2. When the test operations of Step 5 are completed, the corresponding N-1 match signal response patterns are the same as those shown from the the fourth pattern to the last pattern in the second row of Table 4. Consequently, the priority encoder of a CAM can receive the test patterns shown in Table 4 when T_{PE} is applied to the CAM.

Algorithm 1 Test for Priority Encoder Faults (T_{PE})

(1) FOR i=0 to N-1 DO{
  IF(i=N-1){
    Write on word i with the binary value of 2^{B-1}-1;}
  ELSE{
    Write on word i with the binary value of 0.}}
(2) Compare 0 with the bit B-1 of all words and the other bits of all words are masked. Check if the matched address is 0.
(3) Compare 1 with the bit B-1 of all words and the other bits of all words are masked. Check if the matched address is an invalid address.
(4) Compare $2^{B\text{-}1}$-1 with all words and check if the matched address is N-1.
(5) FOR i=0 to N-2 DO{
  Write on word i with the binary value of $2^{B\text{-}1}$-1.
  Compare $2^{B\text{-}1}$-1 with all words and check if the matched address is i.
  Write on word i with the binary value of 0.}}

For example, consider a 4×8-bit CAM under test. When T_{PE} is applied to the CAM, Step 1 initializes word 0, word 1, and word 2 to all-0 state and writes 127 (01111111) into word 3. Thus, the state of the CAM is shown in Fig. 6(a) when Step 1 is completed. Step 2 compares 0 with bit 7 of all words, i.e., bit 0 to bit 6 are masked. As Fig. 6(b) shows, the Compare operation causes that

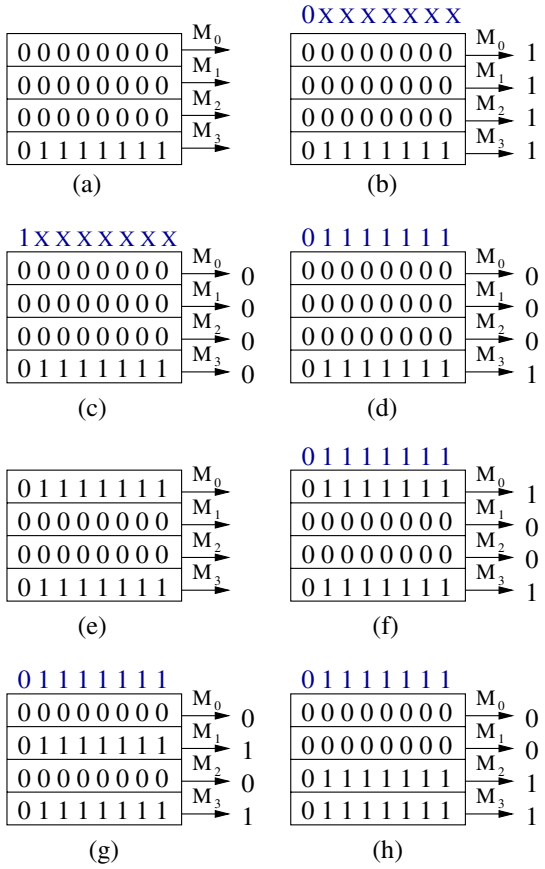

Figure 6: Fault-free states of the 4×8-bit CAM under test when T_{PE} is executed.

$\{M_0M_1M_2M_3\}=\{1111\}$. Similarly, Step 3 compares 1 with bit 7 as shown in Fig. 6(c) and the corresponding match signal response is $\{M_0M_1M_2M_3\}=\{0000\}$. Step 4 compares 127 with all words of the CAM as shown in Fig. 6(d) and the corresponding match signal response is $\{M_0M_1M_2M_3\}=\{0001\}$. Finally, Step 5 first performs a Write-127 operation on word 0, and the state of the fault-free CAM is shown in Fig. 6(e). Then a Compare operation compares 127 with all words as shown in Fig. 6(f). This causes that the match signal response is $\{M_0M_1M_2M_3\}=\{1001\}$. Subsequently, a Write operation writes all-0 data into word 0. Then the second word (word 1) is addressed and the data 127 is written into the addressed word. Again, the Compare operation compares 127 with all words as shown in Fig. 6(g). This causes that the match signal response is $\{M_0M_1M_2M_3\}=\{0101\}$. Finally, the last operation of Step 5 writes all-0 data into the word 1. The same operations are repeatedly performed on word 2 and the corresponding match signal responses $\{0011\}$ as shown in Fig. 6(h). Therefore, we see that the match signal responses shown in Table 4 all can be generated when the T_{PE} is executed on the CAM. That is, we conclude that T_{PE} can cover 100% SAFs of the ripple PCL of the priority encoder.

When T_{PE} is executed, the encoder of the priority encoder also can receive its all possible functional patterns.

Table 5 lists the corresponding output patterns of PCL when the match signal patterns shown in Table 4 appear at inputs of the PCL. As Table 5 shows, all-0 and walking-1 patterns are included, which are all the possible functional patterns of the encoder. Therefore, T_{PE} can execute the functional testing for the encoder. That is, it also can detect the SAFs of the encoder of the priority encoder.

Table 5: Corresponding output patterns of PCL when the patterns shown in Table 4 are applied to the PCL.

Output patterns of PCL $(Y_0Y_1 \ldots Y_{N-1})$
(1000...0000)
(0000...0000)
(0000...0001)
(1000...0000)
(0100...0000)
(0010...0000)
⋮
(0000...1000)
(0000...0100)
(0000...0010)

According to Algorithm 1, we see that Step 1 needs N Write operation; Step 2, Step 3, and Step 4 need 3 Compare operations; and Step 5 needs 2(N-1) Write operations and N-1 Compare operations. Thus, T_{PE} only requires $3N$-2 Write operations and N+2 Compare operations to cover the SAFs of the priority encoder for an $N \times B$-bit CAM.

4 Hit Signal Generator Testing

The Hit Signal Generator evaluates the match signals with bit-wise OR operation. Thus, the output Hit signal can be expressed as Hit=$M_0|M_1|\ldots|M_{N-1}$, where $|$ represents the OR operation. Therefore, the Hit Signal Generator can be regarded as an N-input OR gate with a output—Hit. $N+1$ test patterns are needed for detecting SAFs of the N-input OR gate. The test patterns are $\{000\ldots00, 100\ldots00, 010\ldots00, 001\ldots00, \ldots, 000\ldots10, 000\ldots01\}$. As Table 1 shows, these test patterns can be covered by Type-1 and Type-4 patterns. Therefore, typical tests for CAM cell array faults also can fully cover the SAFs of the Hit Signal Generator. But, if the tests for CAM cell array faults cause that the match signal patterns belong to Type-2/Type-3 and Type-4, most of the stuck-at-0 faults of the Hit Signal Generator cannot be detected. The reason can easily be shown by observing the Type-2/Type-3 expected responses on match signals.

5 Fault Coverage Analysis

In this section, we analyze the fault coverage of SAFs. We use the Verifault of Verilog-XL Simulator to simulate the fault coverage of SAFs of the PCL with ripple group logic. Table 6 summarizes fault coverages of SAFs when different tests are applied to the CAMs with N=8, 16, 32, and 64. The second column shows the fault coverages of SAFs in the ripple PCL when a test is used to test the CAM and causes that the match signal patterns are the same as Type-1 and Type-4 patterns shown in Table 2. For example, the tests reported in [1–3] belong to this kind of test, called Test A. Similarly, the third column summarizes the fault coverages of SAFs in the ripple PCL when a test is applied to the CAM and causes that the match signal patterns are the same as Type-3 and Type-4 patterns shown in Table 2. For example, the tests presented in [12] are this kind of test, called Test B. As the table shows, conventional tests used for CAM cell array faults cannot fully cover the SAFs in the PCL of the priority encoder. Note that the fault coverages in the second and third column are slightly decreased with N. However, the proposed test T_{PE} achieves 100% fault coverage of SAFs regardless of the N.

Table 6: Comparison of fault coverage.

	Type-1, Type-4	Type-3, Type-4	T_{PE}
$N = 8$	64.3%	91.4%	100%
$N = 16$	62%	90.7%	100%
$N = 32$	61%	90.3%	100%
$N = 64$	60.5%	90.2%	100%

Finally, we compare the SAF fault coverage of priority encoder and hit signal generator when different types of tests are applied. Table 7 summarizes the comparison results. As the table shows, the Test A (causing the CAM has Type-1 and Type-4 match signal responses) has lower SAF fault coverage for the priority encoder, since it only can provide about 60% SAF fault coverage for the PCL of the priority encoder. However, it can cover 100% SAFs of hit signal generator as described in Sec. 4. The Test B (causing the CAM has Type-2/Type-3 and Type-4 match signal responses) achieves medium fault coverage for the priority encoder. But it cannot detect most of stuck-at-0 faults in the hit signal generator as described in Sec. 4. According to Table 7, we see that the combination of Test A and T_{PE} can achieve the best fault coverage for both the priority encoder and the hit signal generator.

Table 7: Comparison of test algorithms.

	Priority Encoder Fault Coverage	Hit Signal Generator Fault Coverage
Test A	Low	High
Test B	Medium	Low
Test A+T_{PE}	High	High
Test B+T_{PE}	High	Low

6 Conclusions

In this paper we have presented a test algorithm for testing SAFs of the priority encoder of a CAM. For an $N \times B$-bit CAM, the proposed test algorithm (T_{PE}) only requires

$3N$-2 Write operations and N+2 Compare operation to cover 100% SAFs in the ripple prefix computation logic of the priority encoder. It also provides the all possible functional patterns for the encoder of the priority encoder, such that the encoder is tested functionally as well. The T_PE also can be extended to test the SAFs of the carry lookahead prefix computation logic of the priority encoder. Fault coverage simulation results show that the SAF coverage of the prefix computation logic is increased from 90.2% or 60.5% to 100% for a CAM with 64 words. If T_{PE} is combined with the other tests for CAM cell array faults, high fault coverage of the priority encoder can be achieved.

Acknowledgment

This work was supported in part by the National Science Council, R.O.C., under Contract NSC 93-2215-E-008-017 and the Caiser of University System of Taiwan (UST).

References

[1] K.-J. Lin and C.-W. Wu, "Testing content-addressable memories using functional fault models and March-like algorithms", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 5, pp. 577–588, May 2000.

[2] J.-F. Li, R.-S. Tzeng, and C.-W. Wu, "Testing and diagnosing embedded content addressable memories", in *Proc. IEEE VLSI Test Symp. (VTS)*, Monterey, California, Apr. 2002, pp. 389–394.

[3] J.-F. Li, R.-S. Tzeng, and C.-W. Wu, "Testing and diagnosis methodologies for embedded content addressable memories", *J. Electronic Testing: Theory and Applications*, vol. 19, no. 2, pp. 207–215, Apr. 2003.

[4] G. Giles and C. Hunter, "A methodology for testing content addressable memories", in *Proc. Int. Test Conf. (ITC)*, 1985, pp. 471–474.

[5] P. Mazumder, J. H. Patel, and W. K. Fuchs, "Methodologies for testing embedded content addressable memories", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 1, pp. 11–20, Jan. 1988.

[6] A. J. McAuley and C. J. Cotton, "A self-testing reconfigurable CAM", *IEEE Journal of Solid-State Circuits*, vol. 26, no. 3, pp. 257–261, Mar. 1991.

[7] S. Kornachuk, L. McNaughton, R. Gibbins, and B. Nadeau-Dostie, "A high speed embedded cache design with non-intrusive BIST", in *Proc. IEEE Int. Workshop on Memory Technology, Design and Testing (MTDT)*, San Jose, 1994, pp. 40–45.

[8] W. K. Al-Assadi, A. P. Jayasumana, and Y. K. Malaiya, "On fault modeling and testing of content-addressable memories", in *Proc. IEEE Int. Workshop on Memory Technology, Design and Testing (MTDT)*, 1994, pp. 78–81.

[9] Y. S. Kang, J. C. Lee, and S. Kang, "Parallel BIST architecture for CAMs", *Electronics Letters*, vol. 33, no. 1, pp. 30–31, Jan. 1997.

[10] P. R. Sidorowicz and J. A. Brzozowski, "An approach to modeling and testing memories and its application to CAMs", in *Proc. IEEE VLSI Test Symp. (VTS)*, Apr. 1998, pp. 411–416.

[11] T. Chadwick, T. Gordon, R. Nadkarni, and J. Rowland, "An ASIC-embedded content addressable memory with power-saving and design for test features", in *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, 2001, pp. 183–186.

[12] J. Zhao, S. Irrinki, M. Puri, and F. Lombardi, "Testing SRAM-based content addressable memories", *IEEE Trans. Computers*, vol. 49, no. 10, pp. 1054–1063, Oct. 2000.

[13] X. Du, S. M. Reddy, J. Rayhawk, and W.-T. Cheng, "Testing delay faults in embedded CAMs", in *IEEE Asian Test Symp. (ATS)*, 2003, pp. 378–383.

[14] D. Wright and M. Sachdev, "Transistor-level fault analysis and test algorithm development for ternary dynamic content addressable memories", in *Proc. Int. Test Conf. (ITC)*, Sep. 2003, pp. 39–47.

[15] J.-F. Li, "Diagnosing binary content addressable memories with comparison and RAM faults", *IEICE Trans. Information and Systems*, vol. E87-D, pp. 601–608, Mar. 2004.

[16] J.-F. Li, "Testing comparison faults of ternary CAMs based on comparison faults of binary CAMs", in *Proc. Asia and South Pacific Design Automation Conf. (ASP-DAC)*, Shanghai, Jan. 2005, pp. 65–70.

[17] J.-F. Li and C.-K. Lin, "Modeling and testing comparison faults for ternary content addressable memories", in *Proc. IEEE VLSI Test Symp. (VTS)*, Palm Springs, May 2005, pp. 60–65.

[18] B. Nadeau-Dostie, A. Silburt, and V. K. Agarwal, "A serial interfacing technique for external and built-in self-testing of embedded memories", *IEEE Design & Test of Computers*, vol. 7, no. 2, pp. 56–64, Apr. 1990.

[19] K. J. Schultz, "Content-addressable memory core cells: A survey", *Integration, the VLSI J.*, vol. 23, pp. 171–188, 1997.

[20] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, Addison-Wesley, Reading, Massachusetts, third edition, 2005.