Fault Diagnosis for RAMs using Walsh Spectrum

Atsumu Iseno[1], Yukihiro Iguchi[1] and Tsutomu Sasao[2,3]
[1]Department of Computer Science, Meiji University
[2]Department of Computer Science and Electronics, Kyushu Institute of Technology
[3]Center for Microelectronic Systems, Kyushu Institute of Technology

Abstract

In this paper, we show a method to locate a single stuck-at fault of a random access memory (RAM). From the fail-bitmaps of the RAM, we obtain their Walsh spectrum. For single stuck-at fault, we show that the fault can be identified and located by using only the 0-th and 1-st coefficients of the spectrum. We also show a circuit to compute these coefficients. The computation time is $O(2^n)$, where n is the number of bits in the address of the RAM. The computation time is much shorter than one that use logic minimization method.

1 Introduction

Random access memories (RAMs) are extensively used in various electronic systems. RAMs are used not only in storage-units for computer systems, but also in logic devices such as FPGAs. Many RAMs are integrated into one chip SoC (System on Chip), so enhancing the test method for RAMs is very important.

Fail-bitmaps contain important information for locating the faults. Fail-bitmaps are two-dimensional arrays showing the locations of the faulty cells of a RAM. Each map has the same size as the RAM under test. The map has '0's for the cells whose responses are the same as the expected values, and '1's otherwise.

The patterns in fail-bitmaps depend on the locations and types of faults in RAMs. We use this property to diagnose RAMs. A straightforward method is to store the fail-bitmaps for each fault. However, such method is inefficient. We consider a fail-bitmap as a logic function, and represent it by a sum-of-products expression (SOP). Different faults corresponds to different SOPs. Unfortunately, the computation time for converting fail-bitmap into an SOP is sometimes very long. So, we have to use an alternative method.

In this paper, we show a diagnosis method of RAMs using the Walsh spectrum. From the fail-bitmaps of the RAM, we obtain their Walsh spectrum. For a single stuck-at fault, we show that the fault can be identified and located by using only the 0-th and 1-st coefficients of the spectrum. We also show a method to compute these coefficients. The computation time of the proposed method is shorter than the method using an SOP minimizer. The rest of the paper is organized as follows: Section 2 shows the structure and a test method of RAMs. Section 3 surveys the Walsh transform and Walsh spectrum, and proposes a diagnosis method using the 0-th and 1-st coefficients of the Walsh spectrum. Section 4 shows a circuit to compute the 0-th and 1-st coefficients of the Walsh spectrum. Section 5 shows an experimental result, and Section 6 concludes the paper.

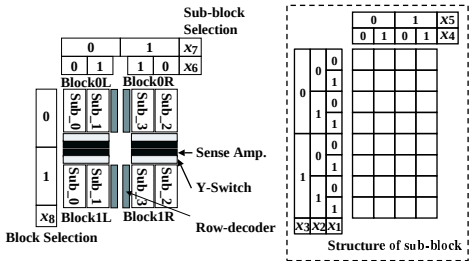

Figure 2.1: Structure of memory.

2 Structure and Test of RAMs

In this section, we show the structure of RAMs, their fault model, and their test patterns.

2.1 Structure of RAMs and their fault model

Figure 2.1 shows a RAM, which has a two dimensional structure of memory cells. A RAM consisting of single big array is slow and dissipates much power, so RAMs are typically built using a hierarchical structure. Cells are grouped to form a sub-block, sub-blocks are grouped to form a **block**, blocks are grouped to form a RAM. A sub-block consists of horizontal **word-lines** and vertical **bit-lines**, and each **memory cell** is located at an intersection of two lines.

In this paper, we consider the fault model in Table 2.1, and use a memory model in Figure 2.2 for the RAM shown in Figure 2.1. This memory has four blocks; a block has two sub-blocks; a **row decoder** selects a word in the block; a sub-block has four bit-lines; a **Y-switch** selects one out of four bit lines; a **sense amplifier(SA)** is connected to a Y-switch; and a **block·sub-block selection multiplexer** selects one from four blocks, or eight sub-blocks. We consider eight types of single stuck-at faults shown in Table 2.1.

2.2 Test Method for RAM

We apply test patterns to RAMs to detect faults. Various classes of test patterns for RAMs are known: N, $N^{1.5}$, and N^2[3]. Each denotes the length of the test, where $N = 2^n$ is the size of a RAM, and n is the number of bits in the address. Testing time for N is the shortest, and for N^2 is the longest. Normally, test patterns with the length N are used to test large-scale

Table 2.1: Type of faults and fault model.

Type of faults	Fault model
Cell	SA-0(1) on a cell.
Row Decoder	SA-0(1) on a input of Row decoder.
Bit Line	SA-0(1) on a bit-line
Sense Amplifier	SA-0(1) on an output of sense amplifier.
I/O	SA-0(1) on I/O data line.
Block·Sub-block Selection MUX	SA-0(1) on a selection input for block·sub-block selection multiplexer.
Y-Switch	SA-0(1) on a selection input of Y-switch.
Address Line	SA-0(1) on an address line.

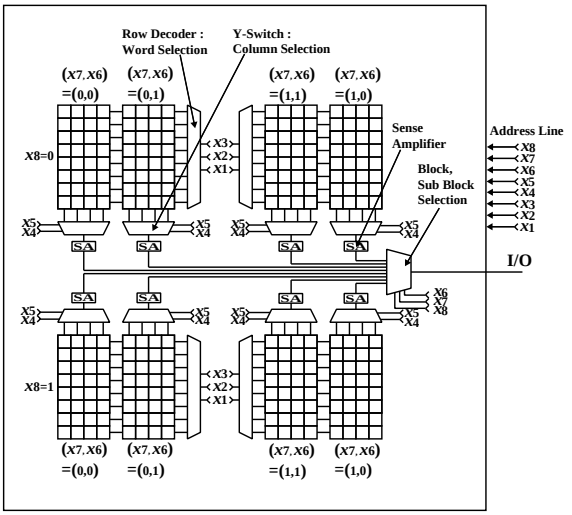

Figure 2.2: Memory model.

RAMs. The **march** pattern is the typical test pattern with the length N. Various march patterns exists. In this paper, we use the shortest march pattern, called **MATS+**. When we test a RAM using MATS+, we write values, read values, and test cells as follows:

1. **Step M0**: Write 0 (0W: Zero Write) to all the cells.
2. **Step M1**: Read values from cells from the address 0 to N in the ascending order. When the output is equal to the expected value, we produce 0 which shows that the value of the output is consistent to the expected value. When the output is different from the expected value, then we produce 1 which shows the value is incorrect. After reading a value from the cell, we write 1 to the cell. We will do this to all the cells.
3. **Step M2**: We read a value from each cell in the descending order. In this case, the expected value of the cell is one. After reading a value from the cell, we write 0 to the cells. We will do this to all the cells.

We obtain **fail-bitmap 1** in Step M1, and **fail-bitmap 2** in Step M2. With these fail-bitmaps, we can identify

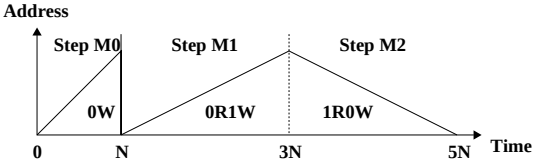

Figure 2.3: Test pattern: MATS+

and locate all types of single stuck-at faults shown in Table 2.1.

Example 2.1 *Figure 2.4(a)–(d) show examples of fail-bitmaps for the RAM shown in Figure 2.2.*

- *Figure 2.4(a) shows the fail-bitmaps for the RAM where one cell has a stuck-at 0 fault in the sub-block* **Sub_2** *in* **Block0R**. *In this case, we read expected value 0 from the faulty cell in Step M1. Thus, the fail-bitmap 1 has all 0's, while the fail-bitmap 2 has one 1.*
- *Figure 2.4(b) and (c) show fail-bitmaps 1 and 2 for the RAM where the input x_1 of the row decoder in* **Block0L** *has a stuck-at 0 fault. In this case, we can only access one half of the cells in* **Sub_0** *and* **Sub_1** *of* **Block0L**. *For example, we read the expected value 0 from the cell in the address 0 in Step M1, and then write one to the same cell. When we read the value from the address 1, the stuck-at 0 fault makes the row decoder to read the value from the cell in the address 0 again. At this time, the cell has the value 1. The value 1 that indicates this inconsistency is written in fail-bitmap 1. It appears that all the cells with the odd number of addresses are faulty. When the address are accessed in descending order in Step M2, it appears that all the cells in the even number of addresses are faulty.*
- *Figure 2.4(d) shows the fail-bitmap for the RAM where the bit-line 1 has a stuck-at 0 fault in* **Sub_2** *of* **Block1R**. *In this case, fail-bitmap 1 has all 0's.* ∎

By analyzing the fault model and the memory model, we have the followings:

1. A single fault often makes the multiple 1's in fail-bitmaps.
2. Different faults produce different fail-bitmaps.
3. In many cases, fail-bitmaps 1 and 2 are different.
4. By using fail-bitmaps, we can identify and locate the fault of the RAM.

From these, we have

Theorem 2.1 *Given the memory model in Figure 2.2 and the fault model shown in Table 2.1, a single stuck-at fault can be identified and located using two fail-bitmaps produced by MATS+.*

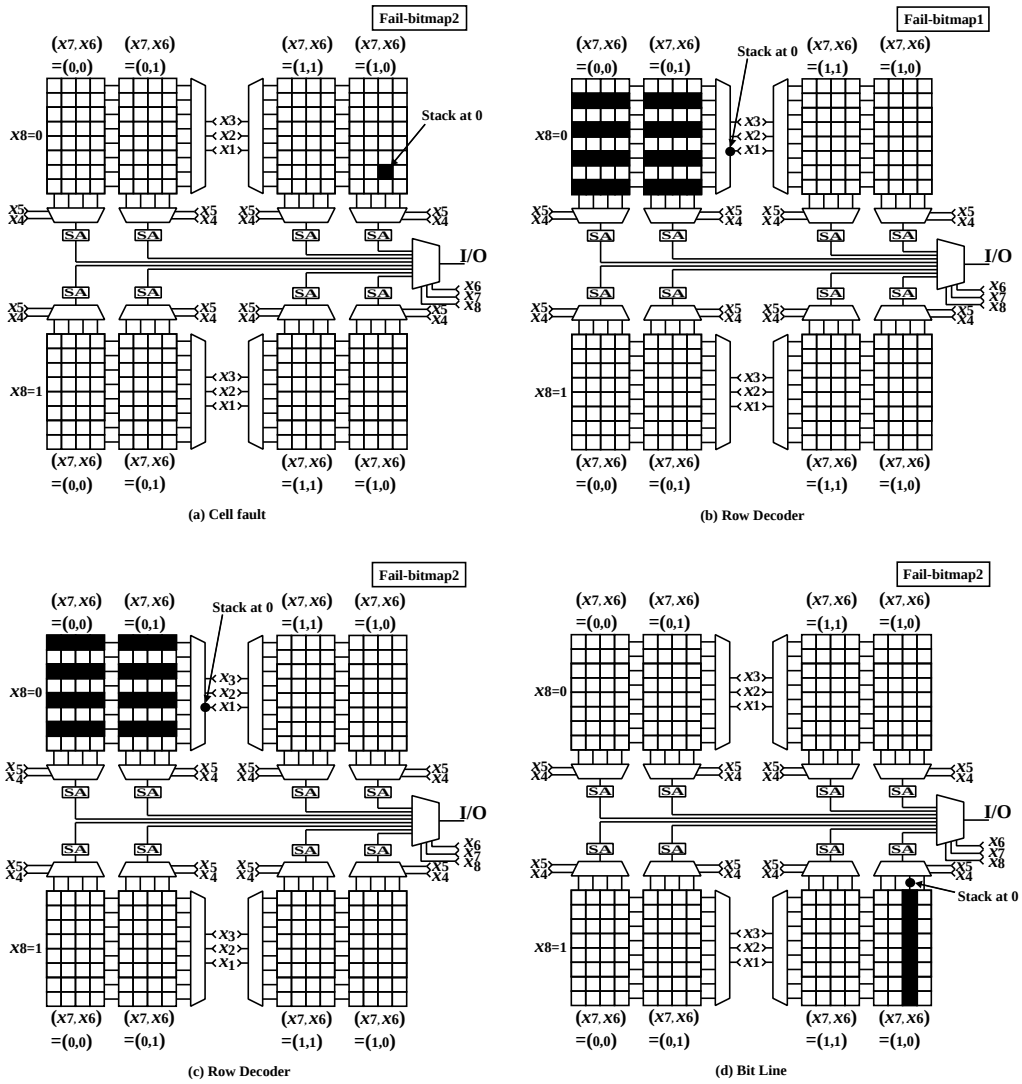

Figure 2.4: Examples of fail-bitmaps

3 Fault Diagnosis using Walsh Spectrum

In Section 2, we showed that different faults produce different fail-bitmaps. In this section, we propose a method to identify and locate a single stuck-at fault by using the Walsh spectrum. In Subsection 3.1, we survey the Walsh transform. In Subsection 3.2, we show a property of fail-bitmaps for a single stuck-at fault. In Subsection 3.3, we propose a fault diagnosis method for RAMs by using the Walsh spectrum.

3.1 Walsh transform and Walsh spectrum[7, 12, 14]

Let $\mathbf{W}(n)$ be the Walsh transform matrix defined by

$$\mathbf{W}(n) = \bigotimes_{i=1}^{n} \mathbf{W}(1), \quad \mathbf{W}(1) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

where $\bigotimes$ denotes the Kronecker product. The matrix $\mathbf{W}(1)$ is called the basic Walsh transform matrix. In a symbolic representation, we have

$$\mathbf{W}(1) = \begin{bmatrix} 1 & 1 - 2x_i \end{bmatrix}.$$

In this representation, $x_i = 0$ denotes the first row of the matrix, while $x_i = 1$ denotes the second row of the matrix.

Definition 3.1 *For a function f given by the truth-vector* $\mathbf{F}$, *the* **Walsh spectrum** $\mathbf{W}_f = [w_0, \ldots, w_{2^n-1}]^t$ *is*

$$\mathbf{W}_f = 2^{-n}\mathbf{W}(n)\mathbf{F}.$$

The Walsh transform matrix is self-inverse up to the constant 2^n. Therefore, the inverse Walsh trans-

form is defined by

$$f = \mathbf{X}_w \mathbf{W}_f, \quad \mathbf{X}_w = \bigotimes_{i=1}^{n} \begin{bmatrix} 1 & 1-2x_i. \end{bmatrix}$$

Example 3.1 *Consider the function* $f(x_1, x_2) = \bar{x}_1 \vee x_2$. *The truth vector is* $\mathbf{F} = [1,0,1,1]^t$. *The Walsh spectrum is*

$$\begin{aligned} \mathbf{W}_f &= 2^{-n}\mathbf{W}(n)\mathbf{F} \\ &= 2^{-2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \\ &= 2^{-2}\begin{bmatrix} 3 \\ 1 \\ -1 \\ 1 \end{bmatrix}. \end{aligned}$$

$\mathbf{X}_w = [1, 1-2x_1, 1-2x_2, (1-2x_1)(1-2x_2)]$
In the above representation, by assigning $x_2 = 0$ *and* $x_1 = 0$, *we have the first row of the matrix:*$[1,1,1,1]$.
By assigning $x_2 = 0$ *and* $x_1 = 1$, *we have the second row of the matrix:*$[1,-1,1,-1]$.
By assigning $x_2 = 1$ *and* $x_1 = 0$, *we have the third row of the matrix:*$[1,1,-1,-1]$.
By assigning $x_2 = 1$ *and* $x_1 = 1$, *we have the last row of the matrix:*$[1,-1,-1,1]$.
f *is represented by*

$$\begin{aligned} f &= \mathbf{X}_w \mathbf{W}_f \\ &= \frac{1}{4}[1, 1-2x_1, 1-2x_2, (1-2x_1)(1-2x_2)]\begin{bmatrix} 3 \\ 1 \\ -1 \\ 1 \end{bmatrix} \\ &= \frac{1}{4}[3 + (1-2x_1) - (1-2x_2) + (1-2x_1)(1-2x_2)] \end{aligned}$$

∎

The last expression in the above example is the **Walsh expression**. We can obtain the Walsh spectrum $\mathbf{W}_f$ from the Walsh transformation of the truth vector $\mathbf{F}$. On the other hand, we have the original logic function f from the inverse Walsh transformation of the Walsh spectrum.

3.2 Properties of fail-bitmaps for a single stuck-at fault

As shown in Section 2, a single stuck-at fault can be identified and located by the number of 1's and/or the location of 1's in the fail-bitmaps.

To identify and locate the faults of RAMs, we use fail-bitmaps. Most methods diagnose RAMs by classifying the fail patterns or shapes of fail-bitmaps[10, 15, 16, 9]. Since fail-bitmaps are too large to store, most methods compress the fail-bitmap data. Vollrath et al. compressed them using graphical method with simple hardware, and they compressed a 64M-bit fail-bitmap into 2k-bit allowing classification of 13 fail pattern[15]. Iseno et al. compressed them by using hardware-compressor[9]. Chen et al. proposed a "stripped-down" compressor[1], which overcome the loss of data. It compresses by stripping row data and column data of the fail-bitmap, and performing OR operation to each segment. Their diagnosing system decompresses the compressed data, and can handle the decompressed fail-bitmaps with X's, where X denotes 0 or 1[2]. The compression ratio is the width of a fail-vector divided by two, e.g., when the word width is 32, the compression ratio is 16.

In this paper, we consider automatic diagnosis for single stuck-at faults by using fail-bitmaps. The simplest method is counting the **weight of the map function**, the number of 1's in the fail-bitmaps. For example, when a single stuck-at fault occurs in a cell, the weight of either fail-bitmap 1 or 2 becomes 1. However, this method does not distinguish different faults that produce fail-bitmaps with the same weights. Let the **map function 1(2)** represent the fail-bitmap 1(2). The map functions for a single stuck-at fault can be represented by a product term or a constant function. For example, map functions for Figure 2.4(a)-(d) are represented by product terms or constants, as shown in Table 3.1. Where the map function 1 is shown in the upper part, and the map function 2 is shown in the lower part. Consider the example in Figure 2.4(b). The input x_1 of the row decoder has a stuck-at 0 fault, and map function 1 is expressed by the product term $\bar{x}_8\bar{x}_7x_1$. Map functions corresponding to other faults are also represented by product terms or constants, as shown in Table 3.1. The product term and the weight of the map function 1(2) are shown in the upper(lower) row. For example, when the input x_1 of the row decoder has a stuck-at 0 fault, the weight of the map function 1(2) is 32, and the product term is $\bar{x}_8\bar{x}_7x_1(\bar{x}_8\bar{x}_7\bar{x}_1)$. Note that stuck-at faults in the inputs of block·sub-block selection MUX are missing in Table 3.1, but they appear in Table 2.1. This is because faults in the address lines produce the same error as the faults in the inputs of the block.

As shown in Table 3.1, weights of the map function 1 and 2 are distinct for different faults. Thus, we can locate the faulty unit. In this particular example, different faults produce different map functions with distinct weights. However in a different RAM, different faults may produce fail-bitmaps with the same weights. In this case, we can locate the faults by checking literals of the product. In this way, we can identify and locate the faults. By inspecting all possible single stuck-at faults, we have the next theorem.

Theorem 3.1 *With the fault model in Table 2.1, each map function for single stuck-at fault of the RAM shown in Figure 2.2 can be represented by a constant or a product term. We can identify and locate the fault by using these expressions.*

3.3 Diagnosis of Single Stuck-at Fault by Walsh Spectrum

In this section, we show a method to diagnose a single stuck-at fault using the Walsh spectrum. We also show that only the 0-th and 1-st coefficients of the spectrum are necessary to locate and identify single stuck-at fault.

Example 3.2 *Table 3.2 shows five map functions,* $\bar{x}_3\bar{x}_2x_1$, x_3x_2, $x_3\bar{x}_2$, $x_3\bar{x}_1$, *and* x_3. *Table 3.3 shows their Walsh spectrum* $\mathbf{W}(\bar{x}_3\bar{x}_2x_1)$, $\mathbf{W}(x_3x_2)$, $\mathbf{W}(x_3\bar{x}_2)$, $\mathbf{W}(x_3\bar{x}_1)$, *and* $\mathbf{W}(x_3)$. ∎

Table 3.1: Relation of stuck-at faults and map functions.

Type of faults	SA-0		SA-1	
	Map function	Weight	Map function	weight
Cell	0	0	$\bar{x}_8x_7\bar{x}_6x_5\bar{x}_4x_3x_2\bar{x}_1$	1
	$\bar{x}_8x_7\bar{x}_6x_5\bar{x}_4x_3x_2\bar{x}_1$	1	0	0
Row Decoder	$\bar{x}_8x_7\bar{x}_1$	32	$\bar{x}_8x_7\bar{x}_1$	32
	$\bar{x}_8x_7x_1$	32	$\bar{x}_8x_7x_1$	32
Bit Line	0	0	$x_8x_7\bar{x}_6x_5\bar{x}_4$	8
	$x_8x_7\bar{x}_6x_5\bar{x}_4$	8	0	0
Sense Amplifier	0	0	$\bar{x}_8x_7\bar{x}_6$	32
	$\bar{x}_8x_7\bar{x}_6$	32	0	0
I/O	0	0	1	256
	1	256	0	0
Y-Switch	$\bar{x}_8x_7\bar{x}_6x_4$	16	$\bar{x}_8x_7\bar{x}_6x_4$	16
	$\bar{x}_8x_7\bar{x}_6\bar{x}_4$	16	$\bar{x}_8x_7\bar{x}_6\bar{x}_4$	16
Address Line	x_4	128	x_4	128
	$\bar{x}_4$	128	$\bar{x}_4$	128

Table 3.2: Example of map functions.

x_3	x_2	x_1	$\bar{x}_3\bar{x}_2x_1$	x_3x_2	$x_3\bar{x}_2$	$x_3\bar{x}_1$	x_3
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	1	1	1
0	1	1	0	0	0	0	0
1	0	1	0	0	1	0	1
1	1	0	0	1	0	1	1
1	1	1	0	1	0	0	1

Table 3.3: Example of Walsh spectrum.

S	$\mathbf{W}$ $(\bar{x}_3\bar{x}_2x_1)$	$\mathbf{W}$ (x_3x_2)	$\mathbf{W}$ $(x_3\bar{x}_2)$	$\mathbf{W}$ $(x_3\bar{x}_1)$	$\mathbf{W}$ (x_3)
s_ϕ	1	2	2	2	4
s_1	-1	0	0	2	0
s_2	1	-2	2	0	0
s_3	1	-2	-2	-2	-4
s_{12}	-1	0	0	0	0
s_{13}	-1	0	0	-2	0
s_{23}	1	2	-2	0	0
s_{123}	-1	0	0	0	0

Definition 3.2 *Let j-th component of the Walsh spectrum be* $\mathbf{w}_j$. *Let the binary representation of* j *be* $(k_n, k_{n-1}, \ldots, k_1)$, *and let* $R \subseteq \{1, 2, \ldots, n\}$ *be a set of integer* l *such that* $k_l = 1$. *In this case,* $s_R = \mathbf{w}_j$. s_ϕ *shows* **the 0-th coefficient**, s_i *shows* **the 1-st coefficient**, s_{ij} *shows* **the 2-nd coefficient**, *and* s_{ijk} *shows* **the 3-rd coefficient**.

In Table 3.3, s_ϕ represents the weight for the map function. $s_i (i = 1, \ldots, n)$ represents the correlation between f and x_i. s_{ij} $(i, j = 1, \ldots, n,\ i \neq j)$ represents the correlation between f and $x_i \oplus x_j$. s_{ijk} $(i, j, k = 1, \ldots, n,\ i \neq j \neq k)$ represents the correlation between f and $x_i \oplus x_j \oplus x_k$.

Example 3.3 *1. Construct the original logical function from the Walsh spectrum* $\mathbf{W}(\bar{x}_3\bar{x}_2x_1)$ *shown in Table 3.3:*
$2^{-3}[1-(1-2x_1)+(1-2x_2)+(1-2x_3)-(1-2x_2)(1-2x_1)-(1-2x_3)(1-2x_1)+(1-2x_3)(1-2x_2)-(1-2x_3)(1-2x_2)(1-2x_1)] = x_1 - x_1x_2 - x_1x_3 + x_1x_2x_3$.

2. Construct the original logical function from the Walsh spectrum $\mathbf{W}(x_3x_2)$ *shown in Table 3.3:*
$2^{-3}[2-2(1-2x_2)+2(1-2x_3)-2(1-2x_3)(1-2x_2)] = x_2x_3$. ∎

Lemma 3.1 *The absolute value of a coefficient of the Walsh spectrum for the* n*-variable logic function* $f(x_1, x_2, \ldots, x_n) = x_1x_2\cdots x_{n-t}$ *is either* 2^t *or* 0. *Non-zero coefficients correspond to the entries for* $x_{n-t+1} = x_{n-t} = \cdots = x_n = 0$.

(Proof) The Walsh function for $x_1x_2\cdots x_n$ is $\bigwedge_{j=1}^{n}(1-2x_j)$. The Walsh function for $x_1x_2\cdots x_{n-1}\bar{x}_n$ is $\bigwedge_{j=1}^{n-1}(1-2x_j)$. Thus, the Walsh function for $x_1x_2\cdots x_{n-1}$ is $\bigwedge_{j=1}^{n}(1-2x_j) + \bigwedge_{j=1}^{n-1}(1-2x_j) = 2\bar{x}_n\bigwedge_{j=1}^{n-1}(1-2x_j)$. In this way, we can show that the Walsh function for $x_1x_2\cdots x_{n-t}$ is $2^t\bar{x}_n\bar{x}_{n-1}\cdots\bar{x}_{n-t+1}\bigwedge_{j=1}^{n-t}(1-2x_j)$. This expression shows that the absolute values of the coefficients for $x_n = x_{n-1} = \cdots = x_{n-t+1} = 0$ are either 2^t or 0. (Q.E.D.)

Example 3.4 *Consider the Walsh spectrum for the logic function* x_2x_3. *In this case,* $n = 3$ *and* $t = 1$ *in Lemma 3.1. We see that the only coefficients corresponding to* $x_1 = 0$ *take non-zero values, and absolute values of them are 2. We can verify this from the column of* $\mathbf{W}(x_3x_2)$ *in Table 3.3.*

Consider the Walsh spectrum for the logic function x_3. *In this case,* $n = 3$ *and* $t = 2$ *in Lemma 3.1. We see that only coefficients corresponding to*

$x_1 = x_2 = 0$ *take non-zero values, and the absolute values of them are* $2^2 = 4$. *We can verify this from the column of* $\mathbf{W}(x_3)$ *in Table 3.3.* ∎

Lemma 3.2 *[7, pp. 89–97]. Let* α *be a product term. Let* β *be the product term in which some literals in* α *are permutated and/or negated. Let* $\mathbf{W}_\alpha$ *and* $\mathbf{W}_\beta$ *be the Walsh spectra for the functions represented by the product terms* α *and* β*, respectively. Then* $\mathbf{W}_\beta$ *is obtained by permuting coefficients and/or changing the sign of coefficients of* $\mathbf{W}_\alpha$.

Definition 3.3 *Let* $|f|$ *denote the weight of function* f*, i.e., the number of binary vectors* $\vec{a}$ *such that* $f(\vec{a}) = 1$.

Example 3.5 $n = 3$, $|x_2x_3| = 2$, $|x_1 \vee x_2 \vee x_3| = 7$.

Lemma 3.3 *Let* W_α *be the Walsh spectrum of a function* f. *Let* s_i *be the 1-st order coefficient of* W_α*, then* $s_i = |f| - 2v_i$*, where* $v_i = |x_i f|$.

(Proof) s_i is the coefficient for $(1 - 2x_i)$ in the Walsh expression representing a function f. Thus, we have

$$\begin{aligned} s_i &= |f \cdot (1 - 2x_i)| = |f| - 2|f \cdot x_i| \\ &= |f| - 2v_i. \end{aligned}$$

(Q.E.D.)

Lemma 3.4 *Consider the Walsh spectrum of a product term* $p = \hat{x}_1\hat{x}_2 \cdots \hat{x}_{n-t}, where 0 \le t \le n$. *If* $s_i = 0$*, then* p *has no literal of* x_i*, if* $s_i > 0$*, then* p *has a literal* $\bar{x}_i$*, and if* $s_i < 0$*, then* p *has a literal* x_i*, where* $\hat{x}_j$ *represents a literal* x_j *or its complement.*

(Proof) For $f = \hat{x}_1\hat{x}_2 \cdots \hat{x}_{n-t}$, where $i > n - t$, we have

$$\begin{aligned} s_i &= |f| - 2|x_i f| \\ &= |\hat{x}_1\hat{x}_2 \cdots \hat{x}_{n-t}| - 2|\hat{x}_1\hat{x}_2 \cdots \hat{x}_{n-t}\hat{x}_i| \\ &= |\hat{x}_1\hat{x}_2 \cdots \hat{x}_{n-t}\bar{x}_i| + |\hat{x}_1\hat{x}_2 \cdots \hat{x}_{n-t}x_i| \\ &\qquad -2|\hat{x}_1\hat{x}_2 \cdots \hat{x}_{n-t}\bar{x}_i| \\ &= 0. \end{aligned}$$

For $f = \hat{x}_1\hat{x}_2 \cdots \hat{x}_t$, where $i \le n - t$, if $x_i = \hat{x}_i$ then, we have

$$\begin{aligned} s_i &= |f| - 2|x_i f| \\ &= |\hat{x}_1\hat{x}_2 \cdots \hat{x}_{n-t}| - 2|\hat{x}_1\hat{x}_2 \cdots \hat{x}_{n-t}| \\ &= -|\hat{x}_1\hat{x}_2 \cdots \hat{x}_{n-t}| = -2^t < 0. \end{aligned}$$

If $\bar{x}_i = \hat{x}_i$ then, we have

$$\begin{aligned} s_i &= |f| - 2|x_i f| \\ &= |\hat{x}_1\hat{x}_2 \cdots \hat{x}_{n-t}| - 0 = 2^t > 0. \end{aligned}$$

(Q.E.D.)

By Theorem 3.1 and Lemma 3.4, we have.

Theorem 3.2 *By using the 0-th and 1-st coefficients* $(s_\phi; s_1, s_2, \ldots, s_n)$*, we can identify and locate a single stuck-at fault of the RAM shown in Figure 2.2 using the fault model in Table 2.1.*

In the fault diagnosis using the Walsh spectrum, we only use the 0-th and 1-st coefficients $(s_\phi; s_1, s_2, \ldots, s_n)$. The number of the 0-th and 1-st coefficients are 1 and n, respectively. Since we use two map functions, we need $2n + 2$ coefficients to diagnose the fault.

We construct a fault dictionary represented by the Walsh spectrum. Different fail-bitmaps correspond to different faults. Theorem 3.1 shows that the values of all non-zero coefficients of the Walsh spectrum are the same. For example, if a bit-line has a stuck-at 0 fault, then coefficients s_ϕ, s_4, s_5, s_6, s_7, and s_8 take values 8, and others take value 0. In such case, we write the coefficient as $(s_\phi; s_1, s_2, \ldots, s_8) = (8; 0, 0, 0, 8, 8, 8, 8, 8)$, or $8(1; 0, 0, 0, 1, 1, 1, 1, 1)$ as shown in Table 3.4.

Example 3.6 *Consider three functions* x_3x_2, $x_3\bar{x}_2$, *and* $x_3\bar{x}_1$ *shown in Table 3.2 and 3.4.* $x_3\bar{x}_2$ *is derived from the product* x_3x_2 *where the literal* x_2 *is negated.* $x_3\bar{x}_1$ *is derived from the product* $x_3\bar{x}_2$ *where the literal* x_2 *and* x_1 *are replaced.*

1. *Non-zero coefficients of* $\mathbf{W}(x_3x_2)$ *are* s_ϕ, s_2, s_3, s_{23}. *Since* $s_2 < 0$ *and* $s_3 < 0$, *we can see that the product has two literals* x_2 *and* x_3.
2. *Non-zero coefficients of* $\mathbf{W}(x_3\bar{x}_2)$ *are* s_ϕ, s_2, s_3, s_{23}. *Since* $s_2 > 0$ *and* $s_3 < 0$, *we can see that the product has two literals* $\bar{x}_2$ *and* x_3.
3. *Non-zero coefficients of* $\mathbf{W}(x_3\bar{x}_1)$ *are* s_ϕ, s_1, s_3, s_{13}. *Since* $s_1 > 0$ *and* $s_3 < 0$, *we see that the product has two literals* $\bar{x}_1$ *and* x_3. ∎

Thus, we have the following diagnosis method.

Algorithm 3.1 *(Generation a Diagnosis Program)*

1. **Generate the Fault Dictionary**
   Derive fail-bitmaps from the RAM model by inspection. Then, compute the Walsh spectrum, and construct the fault dictionary, as shown in Table 3.4. Note that only the 0-th and 1-st coefficients are necessary.
2. **Construct a Multi-Terminal Multi-valued Decision Tree**
   (a) *Decide the condition of branching in each node by the absolute values of the coefficient,* $|s_{i,j}|(i = \phi, 1, \ldots, n, j = 1, 2)$, *where* $s_{i,j}$ *denotes the coefficient* s_i *of spectra* j.
   (b) *According to the faults in the fault dictionary, construct the tree, and enter the type of faults in the terminal nodes. In this case, the number of terminal nodes is 14, and the number of edges of each non-terminal nodes is larger than three. Thus, the tree obtained by this method is an MTMDT(multi-terminal multi-valued decision tree).*
3. **Generate a Diagnosing Program**
   Reduce the MTMDT, and generate the branching program from the reduced MTMDT.

Table 3.4: Example of fault dictionary by using Walsh spectrum.

Type of faults	SA-0	SA-1
	Spectrum 1 Spectrum 2	Spectrum 1 Spectrum 2
Cell	1(0; 0, 0, 0, 0, 0, 0, 0, 0) 1(1; 1, 1, 1, 1, 1, 1, 1, 1)	1(1; 1, 1, 1, 1, 1, 1, 1, 1) 1(0; 0, 0, 0, 0, 0, 0, 0, 0)
Row Decoder[1]	32(1; 1, 0, 0, 0, 0, 0, 1, 1) 32(1; 1, 0, 0, 0, 0, 0, 1, 1)	32(1; 1, 0, 0, 0, 0, 0, 1, 1) 32(1; 1, 0, 0, 0, 0, 0, 1, 1)
Row Decoder[2]	32(1; 0, 1, 0, 0, 0, 0, 1, 1) 32(1; 0, 1, 0, 0, 0, 0, 1, 1)	32(1; 0, 1, 0, 0, 0, 0, 1, 1) 32(1; 0, 1, 0, 0, 0, 0, 1, 1)
Row Decoder[3]	32(1; 0, 0, 1, 0, 0, 0, 1, 1) 32(1; 0, 0, 1, 0, 0, 0, 1, 1)	32(1; 0, 0, 1, 0, 0, 0, 1, 1) 32(1; 0, 0, 1, 0, 0, 0, 1, 1)
Bit Line	1(0; 0, 0, 0, 0, 0, 0, 0, 0) 8(1; 0, 0, 0, 1, 1, 1, 1, 1)	8(1; 0, 0, 0, 1, 1, 1, 1, 1) 1(0; 0, 0, 0, 0, 0, 0, 0, 0)
Sense Amplifier	1(0; 0, 0, 0, 0, 0, 0, 0, 0) 32(1; 0, 0, 0, 0, 0, 1, 1, 1)	32(1; 0, 0, 0, 0, 0, 1, 1, 1) 1(0; 0, 0, 0, 0, 0, 0, 0, 0)
I/O	1(0; 0, 0, 0, 0, 0, 0, 0, 0) 256(1; 0, 0, 0, 0, 0, 0, 0, 0)	256(1; 0, 0, 0, 0, 0, 0, 0, 0) 1(0; 0, 0, 0, 0, 0, 0, 0, 0)
Y-Switch[1]	16(1; 0, 0, 0, 1, 0, 1, 1, 1) 16(1; 0, 0, 0, 1, 0, 1, 1, 1)	16(1; 0, 0, 0, 1, 0, 1, 1, 1) 16(1; 0, 0, 0, 1, 0, 1, 1, 1)
Y-Switch[2]	16(1; 0, 0, 0, 0, 1, 1, 1, 1) 16(1; 0, 0, 0, 0, 1, 1, 1, 1)	16(1; 0, 0, 0, 0, 1, 1, 1, 1) 16(1; 0, 0, 0, 0, 1, 1, 1, 1)
Address Line[1]	128(1; 1, 0, 0, 0, 0, 0, 0, 0) 128(1; 1, 0, 0, 0, 0, 0, 0, 0)	128(1; 1, 0, 0, 0, 0, 0, 0, 0) 128(1; 1, 0, 0, 0, 0, 0, 0, 0)
Address Line[2]	128(1; 0, 1, 0, 0, 0, 0, 0, 0) 128(1; 0, 1, 0, 0, 0, 0, 0, 0)	128(1; 0, 1, 0, 0, 0, 0, 0, 0) 128(1; 0, 1, 0, 0, 0, 0, 0, 0)
⋮	⋮	⋮
Address Line[8]	128(1; 0, 0, 0, 0, 0, 0, 0, 1) 128(1; 0, 0, 0, 0, 0, 0, 0, 1)	128(1; 0, 0, 0, 0, 0, 0, 0, 1) 128(1; 0, 0, 0, 0, 0, 0, 0, 1)

Algorithm 3.2 *(Diagnosis of Faults of a RAM)*

1. *Obtain map functions 1 and 2 by applying MATS+ to the RAM.*
2. *Compute the 0-th and 1-st coefficients of the Walsh spectrum 1 and 2.*
3. *By using absolute values of coefficients, run the diagnosing program, and identify the type of the fault.*
4. *Locate the fault by checking the signs of the coefficients.*

Example 3.7 *Figure 3.1 shows the MTMDT for the RAM shown in Figure 2.2. In this example, by using only the 0-th coefficients $s_{\phi,1}$ and $s_{\phi,2}$, we can identify all types of faults. However, in general, we also need the 1-st coefficients to identify faults.*

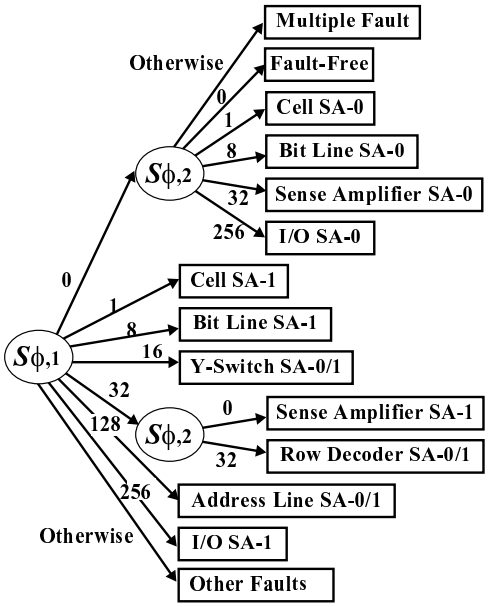

Figure 3.1: A multi-terminal multi-valued decision tree for diagnosis program.

4 Circuit to Compute Walsh Spectrum

In this section, we present a circuit to compute the 0-th and 1-st coefficients of the Walsh spectrum.

The 0-th coefficient s_ϕ is equal to the number of 1's in the map function f. The 1-st coefficient s_i is computed from $|f|$ and $|f \cdot x_i|$ by using Lemma 3.3. We can compute the coefficients of the Walsh spectrum by a circuit shown in Figure 4.1. We use $n+1$ registers to compute and store the 0-th and 1-st coefficients. First, set all the registers to zero. Next, compute the fail-bitmap function f. For every address, compute the EXOR of the value in the memory and the expected value to make f. If $f = 1$ then increment Reg_ϕ. Also, make AND with x_i and f and increment Reg_i if $f \cdot x_i = 1$. Finally, compute the value of s_i from Reg_ϕ and Reg_i by using Lemma 3.1.

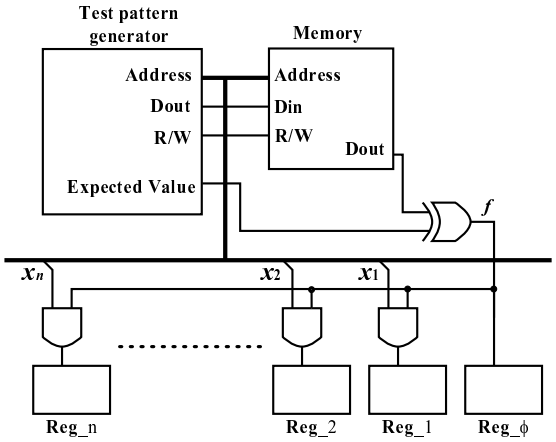

Figure 4.1: Circuit for computing Walsh spectrum

Do this computation for spectra 1 and 2. Since Reg_0 shows the value of $|f|$ and Reg_i shows the value of $|f\ x_i|$, we can compute the value of s_i from the values of Reg_0 and Reg_i. The values of Reg_0 and Reg_i can be obtained by just after applying the MATS+ patterns. Note that the length of MATS+ is $O(2^n)$. Also, the calculation of s_i can be done in $O(n)$ steps. Also the diagnosis by using s_ϕ and s_i can be done in $O(n)$ steps. Thus, the diagnosis of a RAM with single stuck-at fault can be done in $O(2^n)$ step, where is n is the number of bits in the address.

5 Experimental Results

To evaluate the performance, we applied proposed method to a $1M \times 4$-bit SRAM. We developed a fault simulator for a RAM, and measured the time needed diagnose various single stuck-at fault.

We used ESPRESSO MV to minimize the SOPs of map functions. We used a computer utilizing PentiumⅢ(1.13GHz),384MByte RAM, and WindowsXP. Minimization time for the SOP of the map function for single stuck-at fault of a cell, an address line, and I/O, were $28[msec]$, $21[sec]$, and $23[sec]$, respectively. For some cases, ESPRESSO MV took too long time to be practical. We also designed a circuit to compute the 0-th and 1-st coefficients of the Walsh spectrum by using Xilinx SpartanII-XC2S200-5fg456. The simulation result shows that it works with the clock of 101.9MHz. From this result, we can see that the circuit requires computation time $2^{n+1} \times 1/(101.9 \times 10^9)$[sec] for any types of faults. We can evaluate the 0-th and 1-st coefficients of the Walsh spectrum in $213[\mu sec]$ for $1M \times 4$-bits SRAM. Thus, by using proposed circuit, we can diagnose faults much faster than using a logic minimizer.

6 Conclusions

In this paper, we considered a method to identify and locate a single stuck-st the fault of a RAM from the fail-bitmaps generated by MATS+. We consider fail-bitmaps as logic functions. Different faults produce different map functions. We also showed that the map function for single stuck-at fault can be represented by a product term or a constant. In the fault dictionary, logical expressions are the straightforward method to represent the map functions. To convert the truth table of the map function into the product term, an SOP minimizer is necessary. Especially, when the RAM has multiple-faults, SOPs for the map functions tend to be very complex. In such cases, the logic minimization may take excessive computation time. Such a property is undesirable for the memory tester in the production line. To overcome this problem, we propose the new diagnosis method by using only the 0-th and 1-st coefficients of the Walsh spectrum of the fail-bitmaps. We also presented a simple circuit to compute them. The computation time is $O(2^n)$, where n is the number of bits in the address. It is much faster than the method using a logic minimizer.

Acknowledgments: A part of this research is supported by the Aid for Scientific Research of the Japan Society for the Promotion of science(JSPS), and grant of the Takeda Foundation. The authors thank Dr. H. Kawamoto of SOC Design Center of Kitakyusyu City, and Prof. K. Nakamura of Kyushu Institute of Technology for their guidance of memory testing methods. Dr. A. Mishchenko's comments improved the presentation.

References

[1] J. T. Chen, J. Rajski, J. Khare, O. Kebichi, and W. Maly, "Enabling embedded memory diagnosis via test response compression," in *Proc. 19th IEEE VLSI Test Symposium,* pp. 292-298, 2001.

[2] J. T. Chen, J. Khare, K. Walker, S. Shaikh, J. Rajski, and W. Maly, "Test response compression and bitmap encoding for embedded memories in manufacturing process monitoring," in *Proc. IEEE Int. Test Conf.*, pp. 258-267, 2001.

[3] A. J. van de Goor and C. A. Verrujit, "An overview of deterministic functional RAM chip testing," *ACM Computing Surveys*, Vol.22, No.1, March 1990.

[4] A. J.van de Goor, "Testing memories embedded memory cores: fault models, DFT, BIST, BISR, and industrial results," in *Proc. Asian Test Symp.*, Tutorial 1, November, 2001.

[5] A. J. van de Goor, *Testing Semiconductor Memories —Theory and Practice—*, ComTex Publishing (Gouda), 1998.

[6] K. D. Heidtman, "Arithmetic spectrum applied to fault detection for combinational networks," *IEEE Trans. on Comput.* vol. C-40, No. 3, pp.320–324, 1991.

[7] S. L. Hurst, D. M. Miller, J. C. Muzio, *Spectral Techniques in Digital Logic*, Academic Press Inc. (London) LTD. 1985.

[8] M. G. Karpovsky (ed.), *Spectral Techniques and Fault Detection*, Academic Press, Boston, 1985.

[9] A. Iseno, and Y. Iguchi, "A method for storing fail bit maps in burn-in memory testers" in *Proc. Int. Workshop on Electronic Design, Test, and Applications*, pp. 142-145, Christchurch, Jan. 2002.

[10] W. Malzfeldt, W. Mohr, K. Kodalle, "Fast automatic failbit analysis for DRAMs, " in *Proc. IEEE Int. Test Conf.*, pp. 431-438, 1989.

[11] R. Rudell, and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Trans. on Computer Aided-Design*, vol.6, pp.727-750, Sep 1987.

[12] R. S. Stankovic, T. Sasao, and C. Moraga, "Spectral transform decision diagrams," in T. Sasao and M. Fujita (ed.), *Representations of Discrete Functions*, Kluwer Academic Publishers 1996.

[13] A. K. Susskind, "Testing by verifying Walsh coefficients," in *The 11th IEEE Int. Symp. on Fault-Tolerant Computing Symposium*, (FTCS-11), Portland (USA), pp. 206-208, June 1981.

[14] M. A. Thornton, R. Drechsler and D. M. Miller, *Spectral Techniques in VLSI CAD*, Kluwer Academic Publishers, 2001.

[15] J. Vollrath, U. Lederer, and T. Hladschik, "Compressed bit fail maps for memory fail pattern classification," in *IEEE European Test Workshop*, pp. 125-132, Cascais, May 2000.

[16] J. Vollrath, and R. Rooney, "Pseudo fail bit map generation for RAMs during component test and burn-in in a manufacturing environment," in *IEEE Int. Test Conf.*, pp. 768-775, 2001.

[17] J. L. Walsh, "A closed set of normal orthogonal functions, " *American Journal of Mathematics*, 1923.