論文94-31A-8-13

반도체 메모리의 테스트를 위한 MTA(Memory TestAble code) 코드

MTA(Memory TestAble) Code for Testing in Semiconductor Memories

李 仲 鎬*, 趙 相 福*

(Joong Ho Lee and Sang Bock Cho)

要 約

본 논문에서는 ECC(Error Correcting Code) 기법에 기초하여 반도체 메모리의 기능고장을 검출하기에 적합한 새로운 코드인, MTA(Memory TestAble) 코드를 제안한다. 제안된 코드에 대한 성질을 분석하여 기존의 코드들과 비교하였으며, 이 코드의 부호화와 복호화 과정을 보인다. 오류 검출시 MTA 코드를 이용하면 기존의 해밍 코드나 Hsiao 코드에 비해 복호회로가 약 70% 정도 감소되며, 복호처리 속도가 2배 이상 증가된다. MTA 코드는 정보장과 검사장이 동일하게 설정되어 반도체 메모리의 병렬 테스트에 적용이 용이하다. 또한 MTA 코드는 반도체 메모리에서 대부분의 삼중고장까지의 고장을 검출할 수 있다.

Abstract

This paper proposes a memory testable code called MTA(Memory TestAble) code which is based on error correcting code technique for testing functional faults in semiconductor memories. The characteristics of this code are analyzed and compared with those of conventional codes. The developed decoding technique for this code can reduce the decoder circuits up to 70% and obtain two-times faster decoding speed than other codes such as Hamming code or Hsiao code. The MTA code is effectively applicable to parallel testing of semiconductor memories because it has the same information length and parity length. It can detect from single error functional faults to triple error in semiconductor memories.

Ⅰ. 서론

VLSI 기술의 발전으로 메모리 집적도가 급속하게 증가되어 매 2-4 년마다 4 배씩 증가하였고, 현재 국내 산업체에서는 64Mb DRAM의 개발에 성공하였다. 이와 같이 집적도가 증가함에 따라 테스트 경비와 테스트 시간이 문제점으로 나타나고 있으며 국내 산업체에서도 이에 대한 인식을 같이 하고 있다. 기존의 반도체 메모리 기능고장 테스트 방식으로 deterministic 테스트, random 테스트, concurrent 테스트 등이 있었으나 이들은 메모리 신뢰도와 테스트의 경제성을 모두 만족시키기란 매우 어려웠다.

* *正會員,蔚山大學校 電子工學科
(Dept. of Elec. Eng., Ulsan Univ.)
接受日字 : 1993年 12月 20日

ECC 기법에 의한 오류정정 부호는 디지탈 통신시스템이나 메모리장치에 높은 신뢰성을 유지할 수 있어서 널리 사용되고 있다. 특히 반도체 메모리의 테스트에 해밍 SEC-DED(Single Error Correct & Double Error Detect) 코드를 기초로 한 오류정정 부호가 많이 사용된다.[1] 그 이유는 반도체 메모리의 한 워드에 일반적으로 한 비트씩의 정보가 저장되기 때문이다. 그리고 해밍 코드를 개선시켜 복호회로를 간소화시킨 Hsiao 코드가 제안되었다.[2] 또한 기존의 비트형 코드를 더욱 확장하여 유한체 GF(2^m) 상의 바이트 단위 오류정정 방식인 Reed-Solomon 코드나 BCH 코드 등이 제안되었다.[3,4] ECC 기법에서 매우 중요한 복호화 과정은 부가회로의 면적을 가능한 감소시키고, 속도를 향상시키며 신뢰도를 유지할 수 있어야 한다. ECC 기법을 사용하여 반도체 메모리가 설계되는 경우, 부가회로 면적이 10-15% 정도가 되어 메모리 설계자에게는 적지않은 부담이 된다.[5,7] 또한 on-chip 상에서 송신되는 정보를 항상 검사하여야 하므로, 복호회로가 복잡할수록 복호의 처리속도가 떨어진다.

본 논문에서는 ECC 기법을 기초로 하여 반도체 메모리를 테스트할 수 있는 MTA 코드를 제안하고 그 성질을 분석하였다. 제안한 MTA 코드는 기존의 해밍 코드나 Hsiao 코드에 비해 복호회로가 간략화될 수 있는 장점을 가진다. 또한 제안한 MTA 코드를 이용하여 반도체 메모리의 기능고장 검출을 위한 테스트방식에의 적용가능성을 제시하였다. 예를 들어 부호간의 해밍거리를 4로 설정한 MTA 코드를 사용하면 반도체 메모리의 단일고장과 대부분의 이중고장, 및 삼중고장이 검출가능하다는 것을 보였다.

Ⅱ. MTA 코드의 구성

메모리는 한번에 한 개의 데이터 비트를 출력하도록 일반적으로 구성되어 있고, 나타나는 오류의 형태는 한 비트 단위로 나타난다. 따라서 메모리의 오류검출에 대해 단일 비트 오류검출과 수정이 가능한 SEC 코드가 가장 많이 사용되어 왔다. 기존의 해밍 SEC 코드는 짝수 패리티 검사방식으로 부호(codeword)를 구성한다. 예를 들어 부호장(codeword length)이 7, 정보장(information length)이 4인 (7,4) 코드에 대하여 생각해보자. 이때 검사장의 각 비트는 1, 2, 4 번째에 위치하고, 각 검사비트가 검사하는 부호 비트는 1 번째 검사비트가 3, 5, 7 번째 부호비트, 2 번째 검사비트가 3, 6, 7 번째 부호비트, 4 번째 검사비트가 5, 6, 7 번째 비트를 각각 검사하게 된다. Hsiao는 해밍 코드를 더욱 발전시켜 복호(decoding)회로를 줄일 수 있는 기법으로 Hsiao 코드를 제안하였다. 이러한 기법들은 on-chip 상태에서 메모리를 테스트하는데 적용하여 왔으며, 부가회로 면적(10-15%)을 줄이기 위해 많은 시도가 있었지만 별로 효과적이지 못하였다. 뿐만아니라 이러한 이유 때문에 기능고장을 검출하기 위한 메모리 테스트에는 적용되지 않았다.

1. MTA 코드의 구성

본 절에서는 ECC 기법을 이용하여 메모리 테스트를 효율적으로 수행할 수 있는 새로운 코드 생성방법에 대해 기술한다. ECC 방식에서는 부호의 오류를 검출하고 정정하기 위해 검사비트를 필요로 한다. 검사비트의 길이인 검사장이 해밍거리에 해당하는 길이보다 길어지면 오류검출 및 정정이 용이하나 너무 길어지면 on-chip 테스트 방식에서는 경제성이 떨어지므로, 검사장을 무한히 크게 할 수 없다. 따라서 검사장을 적정 길이로 유지하여야 한다. 예를 들어 단일비트 오류정정을 위해 최소 해밍거리는 3을 유지해야 하므로[5] 검사장은 이를 만족할 수 있도록 설정되어야 한다. 하지만 본 연구에서는 메모리 테스트에 적절하도록 검사장을 설정한다.

정보장이 m, 검사장이 k, 부호장이 n 인 H행렬을 다음과 같이 표현해 보자.

$$\mathbf{H}=\begin{bmatrix} p_{0,0} & p_{1,0} & \cdots & p_{j,0} & \cdots & p_{k-1,0} & 1 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ p_{0,1} & p_{1,1} & \cdots & p_{j,1} & \cdots & p_{k-1,1} & 0 & 1 & 0 & \cdots & 0 & \cdots & 0 \\ & & & \cdots & & & & & & & \cdots & & \\ p_{0,i} & p_{1,i} & \cdots & p_{j,i} & \cdots & p_{k-1,i} & 0 & 0 & 0 & \cdots & 1 & \cdots & 0 \\ & & & \cdots & & & & & & & \cdots & & \\ p_{0,m-1} & p_{1,m-1} & \cdots & p_{j,m-1} & \cdots & p_{k-1,m-1} & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

$$=[\mathbf{P}^T\mathbf{I}_k] \quad (2)$$

여기서 $\mathbf{P}^T$는 m×k로 구성된 행렬이고, $\mathbf{I}_k$는 k×k로 구성된 단위행렬이다. 이때 $\mathbf{P}^T$행렬에서 j 열(column)의 0의 갯수를 V_j이라 하고 i 행(row)의 0의 갯수를 H_i이라 할 때, MTA 코드를 다음과 같이 정의한다.

[정의] MTA(Memory TestAble) 코드의 H행렬은 다음의 각 조건들을 모두 만족하여야 한다.

① m = k

② $V_j = H_i = 1$

MTA (16,8) 코드의 H행렬에 대한 예를 그림 1에 나타내었다. 또한 (16,8) 코드의 H행렬 구성방식과 같이 구성할때의 (8,4) MTA 코드는 표 1에 나타난 바와 같다. 표 1에서 1 4는 데이터 비트의 위치를 나

타내고, 5~8은 검사 비트의 위치를 나타낸다. 또한 TP1 TP16은 각각의 코드들을 나타낸다.

$$H = \begin{bmatrix} 1&1&1&1&1&1&0&1&0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&0&1&0&1&0&0&0&0&0&0 \\ 1&1&1&1&1&0&1&1&0&0&1&0&0&0&0&0 \\ 1&1&1&1&0&1&1&1&0&0&0&1&0&0&0&0 \\ 1&1&1&0&1&1&1&1&0&0&0&0&1&0&0&0 \\ 1&1&0&1&1&1&1&1&0&0&0&0&0&1&0&0 \\ 1&0&1&1&1&1&1&1&0&0&0&0&0&0&1&0 \\ 0&1&1&1&1&1&1&1&0&0&0&0&0&0&0&1 \end{bmatrix}$$

그림 1. MTA (16,8) 코드의 H 행렬
Fig. 1. The H matrix of MTA (16,8) code.

표 1. MTA 코드에 의한 (8,4) 코드의 구성
Table 1. (8,4) code by MTA code.

	5	6	7	8	1	2	3	4
TP1	0	0	0	0	0	0	0	0
TP2	0	1	1	1	0	0	0	1
TP3	1	0	1	1	0	0	1	0
TP4	1	1	0	0	0	0	1	1
TP5	1	1	0	1	0	1	0	0
TP6	1	0	1	0	0	1	0	1
TP7	0	1	1	0	0	1	1	0
TP8	0	0	0	1	0	1	1	1
TP9	1	1	1	0	1	0	0	0
TP10	1	0	0	1	1	0	0	1
TP11	0	1	0	1	1	0	1	0
TP12	0	0	1	0	1	0	1	1
TP13	0	0	1	1	1	1	0	0
TP14	0	1	0	0	1	1	0	1
TP15	1	0	0	0	1	1	1	0
TP16	1	1	1	1	1	1	1	1

2. MTA 코드의 성질 분석 및 부호화, 복호화

본 절에서는 MTA 코드의 성질을 분석하였으며, 부호화 및 복호화 과정을 보인다.

1) MTA 코드의 성질

MTA 코드는 다음과 같은 성질이 있다.

정리 1) MTA 코드는 부호장이 항상 짝수로 구성된다.

증명) 정의의 ①항에 의해 m=k이므로 명백하다.

정리 2) MTA 코드의 P^T 행렬은 선형독립(linearly independent)이다.

증명) MTA 코드에서 P^T 행렬의 각 행벡터들이 서로 독립이 아니라면, P^T 행렬에서 최소한 2 개 이상의 행이 서로 동일하여 최소한 1개 이상의 열에서 0이 2개 이상 나타난다. 이는 MTA 코드의 정의에 위배되므로 MTA 코드가 되기 위해서는 P^T 행렬의 각 행벡터들이 서로 독립이어야 한다. 각 열 벡터들에 대해서도 마찬가지 방법으로 증명되므로 MTA 코드가 되기 위해서는 P^T 행렬의 각 열벡터들이 서로 독립이어야 한다. 따라서 MTA 코드의 P^T 행렬은 선형독립이다.

정리 3) MTA 코드에서 유한체 GF(2) 에대해 P^T 행렬의 행들로 구성된 집합과 열들로 구성된 집합은 서로 동일하다.

증명) (k×k) P^T 행렬의 행들로 구성된 집합은 0을 하나만 포함하는 서로 다른 k 개의 행벡터들로 구성되며, (k×k) P^T 행렬의 열들로 구성된 집합은 0을 하나만 포함하는 서로 다른 k 개의 열벡터들로 구성된다. 그리고 k 개의 원소들로 구성되는 벡터로서 0을 하나만 포함하는 서로 다른 벡터의 수는 k 개 이다. 따라서 (k×k) T 행렬의 행들로 구성된 집합과 열들로 구성된 집합은 서로 동일하다.

정리 4) MTA 코드는 systematic 코드이다.

증명) MTA 코드의 H행렬은 $[P^T\ I_k]$ 로 표시되므로 systematic 코드이다.

코드 생성행렬인 G행렬은 $G = [I_k\ P]$ 와 같이 구성된다. 이때 코드 생성과정은 다음과 같다. 코드 생성을 위한 데이터 벡터는 $\vec{d} = (d_0, d_1, \cdots, d_{k-1})$라 하고, 코드벡터는 $\vec{u}$ 라 하자.

$$\vec{u} = \vec{d} \cdot \mathbf{G} = (d_0, d_1, \ldots, d_{k-1})[\mathbf{I_k P}] = (d_0, d_1, \ldots, d_{k-1}, p_k, \ldots, p_{n-1})$$

위의 수식으로 부터 코드벡터에 데이터성분이 그대로 나타나므로 systematic 코드이다. 단 p'_k은 $\vec{d}$ 와 $p_{k,0}$번째 행벡터와의 행렬곱에 의한 결과이고, 마찬가지로 p'_{n-1}은 $\vec{d}$ 와 $p_{k-1,m-1}$번째 행벡터와의 행렬곱에 의한 원소를 나타낸다.

정리 5) MTA 코드를 구성할 수 있는 P^T 행렬의 수는 k! 개 이다.

증명) 정의의 ①항에 의해 P^T 행렬은 k×k의 정방행렬이고, ②항을 만족하기 위해 구성할 수 있는 행의 경우의 수는 식 (3)과 같다.

$$ {}_kC_{k-1} = {}_kP_{k-1}\left(\frac{1}{(k-1)}\right) = k \qquad (3)$$

또한 정리 2에 의해 각 행은 서로 독립이어야 하므로 P^T 행렬의 각 행을 중복하여 사용할 수 없다. 따라서 각 행들을 조합하여 P^T를 구성할 수 있는 경우의 수는 k! 이다.

예를 들어 k=4 일 경우 식 (3)으로 부터 행을 구성할 수 있는 경우의 수(=${}_4C_3$)는 4 이며, 이 행들은

1110, 1101, 1011, 0111 들이다. 따라서 P^T 행렬의 경우의 수는 4! 로서 24개 이다.

정리 6) MTA 코드의 (k×k) P^T 행렬에서 대칭행렬의 수를 R_k라 할때, 다음의 관계가 성립한다.

$R_k = R_{k-1} + (k-1)\times R_{k-2}$ (단 $k \geq 3$)

증명) 첫번째 행을 고정시켜 대칭행렬을 구성하기로 한다.

(1) 첫번째 행이 (011 … 1)인 경우의 대칭행렬 수, P_k :

P^T의 k×k 행렬이 대칭행렬일 경우 P^T의 첫번째 열은 $(011 \cdots 1)^T$가 된다. 따라서 P^T 행렬에서 첫번째 행과 첫번째 열이 제외되어 구성된 (k-1)×(k-1) 행렬은 MTA 코드의 성질을 만족하는 대칭행렬로서 R_{k-1} 개가 존재한다. 그러므로 P_k는 R_{k-1}이다.

(2) 첫번째 행이 (101 … 1)인 경우의 대칭행렬 수, Q_k :

P^T의 k×k 행렬이 대칭행렬일 경우 P^T의 첫번째 열은 $(101 \cdots 1)^T$가 되고, MTA 코드의 성질에 의하여 두번째 행은 (011 … 1)이고 두번째 열은 $(011 \cdots 1)^T$이다. 따라서 P^T행렬에서 첫째행, 둘째행, 첫째열, 둘째열 들이 제외되어 구성된 (k-2)×(k-2) 행렬은 MTA 코드의 성질을 만족하는 대칭행렬로서 R_{k-2} 개가 존재한다. 그러므로 Q_k는 R_{k-2}이다.

(3) 첫번째 행이 (1101 … 1), (11101 … 1), … (111 … 10)인 경우에도 (2)항과 마찬가지로 각각의 Q_k는 R_{k-2}가 된다.

(2)와 (3)항에서 발생된 대칭행렬의 수는 $(k-1)\times Q_k$ 이다. 따라서 P^T행렬에서 대칭행렬의 수 R_k는 P_k와 $(k-1)\times Q_k$의 합이므로 R_k는 $R_{k-1} + (k-1)\times R_{k-2}$이다.

정리 6을 만족하는 예로서 표 2에 k=4일 경우의 대칭행렬에 대해 나타내었다. 또한 k=5 일때 대칭행렬의 수는 26개, k=6 일때 76개, k=7 일때 232개 이다.

표 2. k=4일 경우의 대칭행렬 예

Table 2. The example of symmetric matrix for the case of k=4.

	대 칭 행 렬
k=4	1110 1110 1101 1101 1011 1011 0111 0111 0111 0111 1101 1011 1110 1011 0111 0111 1101 1110 1011 1011 1011 1101 0111 0111 1110 1101 1011 1101 1101 1110 0111 0111 1011 1110 1101 1110 1110 1011 1110 1101

정리 7) MTA 코드의(k×k) P^T행렬에서 임의의 P^T행렬을 P_s^T라하고 (P_s^T) • (P_s^T)를 R_s라 할때, GF(2) 상에서 다음과 같은 성질이 있다.

i) k가 홀수인 경우, R_s는 MTA 코드를 구성하는 P^T행렬이 된다. 특히 P_s^T가 대칭행렬인 경우 s의 대각의 원소들은 모두 0이다.

ii) k가 짝수인 경우, R_s의 모든 원소들을 보수화시켜 구성한 행렬은 MTA 코드를 구성 하는 P^T 행렬이 된다. 특히 P_s^T가 대칭행렬인 경우 R_s의 대각의 원소들은 모두 1(=단위 행렬)이다.

증명) i) P_s^T의 임의의 열벡터 또는 행벡터는 k(=홀수)개의 원소로서 구성되고 0을 하나만 포함하므로 1의 원소들은 짝수개가 되어 동일 벡터와의 곱은 GF(2) 상에서 0이 되고, 서로 다른 벡터의 곱은 1이 된다.

따라서 (P_s^T) • (P_s^T)의 행렬곱에서 P_s^T의 i 번째 행벡터에 대해 반드시 임의의 j 번째 열에 하나의 동일 벡터가 존재하며, j 번째 열을 제외한 나머지 열들에는 i 번째 행 벡터와 같지 않은 벡터가 (k-1)개 존재하므로 동일 벡터와의 곱인 항만 0이 되고, 나머지 항들은 모두 1이 된다. 이는 $V_F H_i = 1$ 을 만족한다. P_s^T의 나머지 행벡터에 대해서도 마찬가지 방법으로 성립되고, P_s^T의 행벡터들이 선형독립이므로 R_s는 MTA 코드를 구성하는 P^T행렬이 된다.

P_s^T가 대칭행렬인 경우에는 P_s^T의 i 번째 행벡터와 P_s^T의 i 번째 열벡터는 서로 동일하다. 따라서 (i,i) 위치의 성분은 모두 0가 된다.

ii) P_s^T의 임의의 열벡터 또는 행벡터는 k(=짝수)개의 원소로서 구성되고 0을 하나만 포함하므로 1의 원소들은 홀수개가 되어 동일 벡터와의 곱은 GF(2) 상에서 1이 되고, 서로 다른 벡터의 곱은 0이 된다. 이와 같은 사실은 i)과 상반되어 i)과 마찬가지 방법으로 증명된다.

정리 8) MTA 코드의 P^T 행렬에 의해 발생된 각 부호들은 다음과 같은 성질을 만족한다.

i) 각 부호의 검사비트에 해당하는 각 벡터들은 서로 다르다.

ii) 데이터 비트의 1의 수가 홀수개 일때, 검사비트에 동일한 홀수개의 0이 존재한다.

iii) 데이터 비트의 1의 수가 짝수개 일때, $\bar{0}$ 는 제외한 검사비트에 동일한 짝수개의 1이 존재한다.

증명) i) 데이터 벡터 $\bar{d}$와 행렬의 행렬 곱에 의해 생성된 벡터로 구성된 검사행렬을 D라 하자. 이때 D는 식 (4)와 같이 표현된다.

$$\mathbf{D} = \bar{d} \cdot \mathbf{P} \tag{4}$$

데이터 비트가 k일때 k개의 기저벡터가 존재하고, k

개의 기저벡터는 선형독립이므로 기저벡터의 선형조합에 의해 $\vec{0}$를 포함하여 2^k개의 서로 다른 데이터 벡터를 생성한다. 이때 k개의 기저벡터와 P의 행렬곱에 의해 생성된 행렬을 생각해 보면 식 (5)와 같다. P의 각 행벡터에서 첫번째 행을 P_0, 두번째 행을 P_1, …, k번째 행을 P_k라 하자.

$$\mathbf{D}=\begin{bmatrix}1&0&\cdots&0&0\\0&1&\cdots&0&0\\&&\cdots&&\\0&0&\cdots&1&0\\0&0&\cdots&0&1\end{bmatrix}\begin{bmatrix}p_{0,0}&p_{0,1}&\cdots&p_{0,k-2}&p_{0,k-1}\\p_{1,0}&p_{1,1}&\cdots&p_{1,k-2}&p_{1,k-1}\\&&\cdots&&\\p_{k-2,0}&p_{k-2,1}&\cdots&p_{k-2,k-2}&p_{k-2,k-1}\\p_{k-1,0}&p_{k-1,1}&\cdots&p_{k-1,k-2}&p_{k-1,k-1}\end{bmatrix}=\begin{bmatrix}\vec{D}_0\\\vec{D}_1\\\cdots\\\vec{D}_{k-2}\\\vec{D}_{k-1}\end{bmatrix}=\begin{bmatrix}\vec{P}_0\\\vec{P}_1\\\cdots\\\vec{P}_{k-2}\\\vec{P}_{k-1}\end{bmatrix} \quad (5)$$

식 (5)에서 생성된 행렬의 각 벡터들은 P행렬의 각 열벡터들이 정리 2에 의해 선형독립이므로 P행렬의 각 열벡터와 같다. 또한 임의의 데이터 벡터에 의해 생성된 행렬은 P행렬의 임의의 열벡터의 선형조합에 의해 생성된 벡터와 같다. 예로서 (00110…0)의 데이터 벡터와 P행렬의 곱에 의해 생성된 검사벡터는 P_2+P_3의 선형조합에 의한 벡터와 같다. 따라서 P행렬의 각 열의 선형조합에 의해 $\vec{0}$를 포함하여 2k개의 서로 다른 벡터를 발생한다.

ii) 식 (5)에서 $\vec{d}$의 임의의 벡터에서 1이 포함된 비트를 d_i라 하자. 이때의 데이터 벡터를 다음과 같이 표시할 수 있다.

$$\vec{d}_i=(0,\ldots,0,d_{i1},0,\ldots,d_{i2},\ldots,d_{is},0),\quad (d_{i1}=d_{i2}=d_{is}=1) \quad (6)$$

$\vec{d}_i$에 의해 발생된 검사벡터를 $\vec{D}_i$이라 하면 다음과 같이 표현된다.

$$\vec{D}_i=\vec{P}_{i1}+\vec{P}_{i2}+\vec{P}_{is} \quad (7)$$

식 (7)은 정리 8의 i)에 의한 결과이다. 이때 s가 홀수이면 홀수개의 $\vec{P}_i$가 존재하며, $\vec{P}_{i1}$, $\vec{P}_{i2}$, $\vec{P}_{is}$ 벡터들의 0이 포함된 행의 합은 0이 되어 홀수개 발생하고, 나머지는 모두 1이 홀수개 발생된다. 따라서 정리의 ii)를 만족한다.

iii) s가 짝수라면 정리 ii)와 같은 성질에 의해 짝수개의 $\vec{P}_i$가 존재하며, 0이 포함된 행의 합은 1이 되어 짝수개 발생하고, 나머지는 모두 0이 짝수개 발생된다.

정리 8의 ii)의 k=4일 경우에 대한 예로서 표 1의 경우 TP2, TP3, TP5, TP8, TP9, TP12, TP14, TP15 들이 그 예이다. 또한 iii)의 예로서 ii)에 대한 표 1의 예를 제외한 나머지 경우에 해당한다.

2)부호화

MTA 코드를 구성하는 H행렬은 식 (8)과 같이 표현된다고 하자.

$$\mathbf{H}=[\mathbf{P}^T\mathbf{I}_k] \quad (8)$$

여기서 $\mathbf{P}^T$ 행렬은 k×k의 정방행렬이고, $\mathbf{I}_k$는 k×k의 단위행렬이다.

식 (8)에서 MTA 코드를 생성하기 위한 생성행렬 G는 식 (9)와 같이 표현할 수 있다.

$$\mathbf{G}=[\mathbf{I}_k\mathbf{P}] \quad (9)$$

$$=\begin{bmatrix}1&0&0&\cdots&0&0&p_{0,0}&p_{0,1}&\cdots&p_{0,k-2}&p_{0,k-1}\\0&1&0&\cdots&0&0&p_{1,0}&p_{1,1}&\cdots&p_{1,k-2}&p_{1,k-1}\\0&0&1&\cdots&0&0&p_{2,0}&p_{2,1}&\cdots&p_{2,k-2}&p_{2,k-1}\\&&&\cdots&&&&&\cdots&&\\0&0&0&\cdots&1&0&p_{k-2,0}&p_{k-2,0}&\cdots&p_{k-2,k-2}&p_{k-2,k-1}\\0&0&0&\cdots&0&1&p_{k-1,0}&p_{k-1,1}&\cdots&p_{k-1,k-2}&p_{k-1,k-1}\end{bmatrix} \quad (10)$$

따라서 MTA 코드 C는 (11) 식과 같이 부호화 된다.

$$\mathbf{C}=\vec{d}\cdot\mathbf{G} \quad (11)$$

여기서 $\vec{d}$는 데이터 벡터로서 $(d_0, d_1, \cdots, d_{k-1})$이다.

3) 복호화

수신된 부호를 $\vec{v}=(v_0, v_1, \cdots, v_{n-1})$이라 하고 $\vec{v}$에 대한 오증(syndrome)을 $\mathbf{S}(\vec{v})=(S_0, S_1, \cdots S_k)$라 하면 아래와 같은 관계를 가진다.

$$\mathbf{S}(\vec{v})=\vec{v}\cdot\mathbf{H}^T \quad (12)$$

이때 $\mathbf{S}=\vec{0}$이면 $\vec{v}$는 정확한 부호이고, $\mathbf{S}\neq\vec{0}$이면 고장이 발생했음을 의미한다. 식 (12)에 의해 MTA 코드의 오증은 식을 구성하면 식 (13)과 같다.

$$\begin{aligned}
S_0&=v_0p_{0,0}\oplus v_1p_{0,1}\oplus\cdots\oplus v_{k-3}p_{0,k-3}\oplus v_{k-2}p_{0,k-2}\oplus v_k\\
S_1&=v_0p_{1,0}\oplus v_1p_{1,1}\oplus\cdots\oplus v_{k-3}p_{1,k-3}\oplus v_{k-2}p_{1,k-1}\oplus v_{k-1}\\
&\cdots\\
S_{k-2}&=v_0p_{k-2,0}\oplus v_1p_{k-2,2}\oplus\cdots\oplus v_{k-2}p_{k-2,k-2}\oplus v_{k-1}p_{n-2,k-1}\oplus v_{n-2}\\
S_{k-1}&=v_1p_{k-1,1}\oplus v_2p_{k-1,2}\oplus\cdots\oplus v_{k-2}p_{k-1,k-2}\oplus v_{k-1}p_{k-1,k-1}\oplus v_{n-1}
\end{aligned} \quad (13)$$

오증식은 오류를 검출하여 오류가 발생된 비트에 대한 위치를 찾아내고자 함이 목적이다. 따라서 하나의 데이터 비트를 여러개의 검사비트에서 중복 검사함으로써 최종적으로 오류가 발생된 위치를 파악할 수 있으며 오류정정이 가능하다. 그러나 오류검출 만을 목적으로 하면서 단일오류 이상의 오류를 검출하

고자 한다면 식 (13)과 같은 복잡한 오증식을 MTA 코드의 경우 간단화 할 수 있다. 이때 식 (13)의 각 오증식에서 하나의 검사비트에 대해 하나의 데이터 비트만 검사하면서 데이터 비트를 중복되게 검사하지 않으면 된다. 따라서 오류검출 만을 위한 오증식을 구성하도록 하자.

오류검출을 위한 오증식 구성시 검사비트에 대해 임의의 데이터 비트를 설정하여 비교하도록 구성할 수 없다. 왜냐하면 수신된 부호가 정상적인 부호 일 때 $\vec{S}=\vec{0}$ 이 되도록 구성할 수 없기 때문이다. 그 예로서 표 1의 (8,4) MTA 코드에서 TP2 코드를 살펴보자. 표 1의 각 코드에 대해 1, 2, 3, 4 번째 비트(데이터 비트)가 $v_0 \sim v_3$ 이고, 나머지 비트가 $v_4 \sim v_7$ (검사비트)이라 할때, 검사비트가 순서대로 한 비트씩 데이터 비트를 검사하도록 구성해보면 식 (14)와 같이 표현된다.

$$\begin{aligned} S_0 &= v_0 \cdot p_{0,0} \oplus v_4 \\ S_1 &= v_1 \cdot p_{1,1} \oplus v_5 \\ S_2 &= v_2 \cdot p_{2,2} \oplus v_6 \\ S_3 &= v_3 \cdot p_{3,3} \oplus v_7 \end{aligned} \tag{14}$$

식 (14)에 TP2의 각 비트의 값을 대입하면 S_0=0, S_1=1, S_2=1, S_3=0 이 되므로, 정상 부호이지만 오류가 있는 것으로 판정된다. 따라서 정상적인 부호일때 $\vec{S}=\vec{0}$을 만족하는 오증식 구성을 위해 다음과 같은 사항을 고려해 보자.

데이터 비트의 1의 수가 홀수일때 정리 8의 ii)에 의해 데이터 비트의 0과 검사비트의 1의 수가 같으며, 데이터 비트의 1의 수가 검사비트의 0의 수와 같으므로, 데이터 비트를 검사비트에 대해 보수대칭이 되게 구성할 수 있다. 이에 대한 경우를 식 (15)에 나타내었다. 식 (15)의 $\vec{d}_i$ 이나 $\vec{D}_i$ 의 벡터의 원소를 적절히 구성하여 두개의 벡터의 합이 모두 1이 되게 구성할 수 있다.

$$\begin{aligned} &\vec{d}_1 = (0,\ldots,0,d_{j1},0,\ldots,0,d_{j2},0,\ldots,0,d_{js},0,\ldots,0) \\ &\vec{D}_1 = (1,\ldots,1,D_{j1},1,\ldots,1,D_{j2},1,\ldots,1,D_{js},1,\ldots,1) \\ &(\text{단, } D_{j1}=\overline{d_{j1}},\ D_{j2}=\overline{d_{j2}},\ D_{js}=\overline{d_{is}}, \\ &\quad d_{j1}=d_{j2}=d_{is}=1,\ j1\neq i\ 1, j\ 2\neq i\ 2, js\neq is) \end{aligned} \tag{15}$$

표 1의 TP2에서 데이터 비트 v_0, v_1, v_2, v_3에 대해 검사비트는 각각 v_7, v_6, v_5, v_4가 보수대칭 비트가 되며, 이때 보수대칭 비트끼리 서로 비교하면 모든 원소 값이 1이 된다. 따라서 어느 한쪽(데이터 비트 혹은 검사비트)을 모두 보수값을 취해 비교하면 정상 부호일 경우 모든 원소값이 0을 만족한다. 이에 대한 MTA 코드의 일반적인 오증생성 과정을 식 (18)에 나타내었다. 이때 $\vec{C}_1$ 은 $\vec{d}_1$ 에 해당하는 부호이다.

$$\mathbf{P} = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,i1} & \cdots & p_{0,i2} & \cdots & p_{0,is} & \cdots & p_{0,k-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,i1} & \cdots & p_{1,i2} & \cdots & p_{1,is} & \cdots & p_{1,k-1} \\ & & & & \cdots & & & & & \\ p_{i1,0} & p_{i1,1} & \cdots & p_{i1,i1} & \cdots & p_{i1,i2} & \cdots & p_{i1,is} & \cdots & p_{2,k-1} \\ & & & & \cdots & & & & & \\ p_{i2,0} & p_{i2,1} & \cdots & p_{i2,i1} & \cdots & p_{i2,i2} & \cdots & p_{i2,is} & \cdots & p_{k-2,k-1} \\ & & & & \cdots & & & & & \\ p_{is,0} & p_{is,1} & \cdots & p_{is,i1} & \cdots & p_{is,i2} & \cdots & p_{is,is} & \cdots & p_{k-2,k-1} \\ & & & & \cdots & & & & & \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,i1} & \cdots & p_{k-1,i2} & \cdots & p_{k-1,is} & \cdots & p_{k-1,k-1} \end{bmatrix} \tag{16}$$

$$\vec{C}_1 = (\vec{d}_1, \vec{D}_1) \tag{17}$$

$$\vec{S} = \vec{C}_1 \cdot \begin{bmatrix} (\vec{P}_0)^T & (\vec{P}_1)^T & \cdots & (\vec{P}_{k-1})^T \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{18}$$

식 (16)에서 $p_{i1,i1}=p_{i2,1}=p_{is,0}=0$ 이라 하자. 부호 발생시 데이터 비트에서 원소 1의 위치에 해당하는 P행렬의 열에 0이 있으면, 발생된 검사비트의 P행렬의 열에 해당하는 원소는 0이 된다. 또한 i1, i2, is 위치 이외의 p_{ij}항에도 MTA 코드의 정의에 의해 임의의 값이 지정될것 이지만, 식 (15)의 데이터 벡터의 원소 1이외의 값은 $p_{i,j}$항에 상관 없이 0이 될것 이므로 고려하지 않기로 한다. 식 (18)에 의해 생성된 일반적인 오증식은 식 (19)에 나타내었다. 이때 S_0, S_1, S_{i1}의 $p_{i,j}$항 중 0이 하나 포함되어 있으므로 그때의 검사비트는 0이 된다.

$$\begin{aligned} S_0 &= d_{j1} \cdot p_{j1,0} \oplus d_{j2} \cdot p_{j2,0} \oplus d_{js} \cdot p_{js,0} \oplus 0 \\ S_1 &= d_{j1} \cdot p_{j1,1} \oplus d_{j2} \cdot p_{j2,1} \oplus d_{js} \cdot p_{js,1} \oplus 0 \\ &\cdots \\ S_{i1} &= d_{j1} \cdot p_{j1,i1} \oplus d_{j2} \cdot p_{j2,i1} \oplus d_{js} \cdot p_{is,i1} \oplus 0 \\ &\cdots \\ S_{i2} &= d_{j1} \cdot p_{j1,i2} \oplus d_{j2} \cdot p_{j2,i2} \oplus d_{js} \cdot p_{is,i2} \oplus 1 \\ &\cdots \\ S_{is} &= d_{j1} \cdot p_{j1,is} \oplus d_{j2} \cdot p_{j2,is} \oplus d_{js} \cdot p_{is,is} \oplus 1 \\ &\cdots \\ S_{k-1} &= d_{j1} \cdot p_{j1,k-1} \oplus d_{j2} \cdot p_{j2,k-1} \oplus d_{js} \cdot p_{is,k-1} \oplus 1 \end{aligned} \tag{19}$$

만약 js≥j2≥j1 이라면 S_0의 검사비트는 D_{j1}이고, S_1는 D_{j2}, S_{i1}는 D_{js}이다. 식 (19)로 부터 검사비트가 검사하는 데이터 비트를 한 비트만 설정할 경우, S_0, S_1, S_{i1}은 검사비트의 값이 0이므로 정상적인 부호에 대해 체크비트나 데이터 비트 중에서 한쪽에 보수를 취해서 정상부호일때 오증값이 모두 0인 오증식을 구할 수 있다. 데이터 비트의 항에 대해 보수를 취해 오증식을 구성한 예는 식 (20)과 같다.

$$\begin{aligned} \vec{S}_0 &= \overline{d_{j1} \cdot P_{j1,0}} \oplus D_{j1} \\ \vec{S}_1 &= \overline{d_{j2} \cdot P_{j2,1}} \oplus D_{j2} \\ \vec{S}_{i1} &= \overline{d_{j1} \cdot P_{j1,i1}} \oplus D_{js} \end{aligned} \tag{20}$$

식 (20)을 제외한 나머지 오증값은 위에서 검사한 데이터 비트를 제외한 나머지 비트를 검사하여야 하며, 이때 검사비트의 값은 1이므로 데이터 비트의 보수를 취해 $\vec{S}=\vec{0}$이 되는 오증식을 중복됨이 없이 구성 가능하다.

데이터 비트의 1의 수가 1개인 경우 정리 8에 의해 검사비트의 0의 수는 1개이며, 오증식 구성시 식 (20)처럼 데이터 비트의 1이 위치하는 비트는 반드시 검사비트의 0이 위치하는 비트와 검사되어야 한다. 따라서 1의 수가 1개인 k개의 데이터 벡터는 k개의 검사벡터에서 반드시 0과 비교되어야 한다. 각 검사벡터의 0의 위치는 P^T 행렬의 0의 위치와 일치하고 k개의 검사벡터에서 0의 위치는 서로 다르므로 k개의 1이 1개인 데이터 벡터를 모두 검사 가능하다. 또한 1이 k-1개인 데이터 비트에서는 검사비트의 0의 수가 k-1개 존재하며, 위와 동일한 방법으로 P^T 행렬의 0의 위치와 비교하면 검사 가능하다. 또한 1의 수가 홀수인 데이터 비트에 대해 똑같은 방식으로 적용하였을때, 1의 수가 홀수인 데이터 비트에서 1의 항에 대해 검사비트의 0의 항에 대해 검사 가능하며, 데이터 비트의 0의 항에 대해서는 검사비트의 1의 항에 대해 검사 가능하다. 따라서 P^T 행렬의 0의 위치에 대해 오증식을 구성한다. 예를 들어 표 1의 (8,4) MTA코드에 대한 오증식은 식 (21)과 같이 간략화할 수 있다.

$$\begin{aligned} S_0 &= \overline{v_3 \cdot p_{0,3}} \oplus v_4 \\ S_1 &= \overline{v_2 \cdot p_{1,2}} \oplus v_5 \\ S_2 &= \overline{v_1 \cdot p_{2,1}} \oplus v_6 \\ S_3 &= \overline{v_0 \cdot p_{3,0}} \oplus v_7 \end{aligned} \tag{21}$$

데이터 비트의 1의 수가 짝수인 경우 정리 8에 의해 검사비트에 1의 수가 동일한 짝수개 존재하며, 이런 경우에는 검사비트에 대한 데이터 비트를 보수대칭이 되게 구성하여 비교할 수 없다. 그 이유는 정리 8의 iii)에 의해 데이터 비트에서 1의 위치에 따라 P 행렬에서 열벡터의 0의 원소에 의해 검사비트에서 1이 생성되며 데이터 비트와 동일한 짝수개가 생성되기 때문이다. 따라서 P^T 행렬의 0의 항에 대해 식 (21)과 같은 방식으로 오증식을 구성하여 각 비트를 비교하면, 비교하고자하는 검사비트와 데이터 비트의 각 비트의 값은 서로 동일한 값이 되어 정상적인 부호일 경우 모두 $\vec{S}=\vec{1}$이 되며 오류가 있을 경우 $\vec{S}=\vec{0}$이 된다. 따라서 오증식 구성을 위해 P^T 행렬의 0의 항에 대해 오증식을 구성하도록 하며, MTA 코드의 오류검출만을 위한 오증식 구성을 위해 앞에서 나타낸 성질들을 만족하는 오증식을 다음과 같이 정의한다.

앞에서 구한 오증식의 해는 정리 2와 3에 의한 성질에 의해 MTA 코드에서 대해 모두 만족된다.(단, k=3인 경우는 일반적인 오증식을 따른다) 따라서 MTA 코드에 대한 일반적인 오증식은 식 (22)와 같이 P^T 행렬의 0의 원소에 대해 구성하도록 한다. 단 i(=0, 1, … , k)는 P^T 행렬에서 0의 위치이다.

$$\begin{aligned} S_0 &= \overline{v_{k-1} p_{0,i1}} \oplus v_{k+1} \\ S_1 &= \overline{v_{k-2} p_{1,i2}} \oplus v_{k+2} \\ &\cdots \\ S_{k-2} &= \overline{v_1 p_{k-2,i3}} \oplus v_{k+m-2} \\ S_{k-1} &= \overline{v_0 p_{k-1,i4}} \oplus v_{k+m-1} \end{aligned} \tag{22}$$

식 (22)와 같이 구성한 오증회로에서도 부호간의 해밍거리를 4로 유지하면 이중고장을 검출할 수 있으며, 삼중고장은 단일고장으로 검출된다. 그림 2와 그림 3에 MTA 코드에 의해 구성된 부호장이 (8,4)와 (16,8)인 코드에 대한 H행렬과 오증에 대한 관계를 나타내었으며, 확장된 부호에 대해 똑같은 방식으로 구성할 수 있다.

$$H = \begin{array}{c} \begin{matrix} v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \end{matrix} \\ \left| \begin{matrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{matrix} \right| \end{array} \qquad \begin{aligned} S_0 &= \overline{v_3} \oplus v_4 \\ S_1 &= \overline{v_2} \oplus v_5 \\ S_2 &= \overline{v_1} \oplus v_6 \\ S_3 &= \overline{v_0} \oplus v_7 \end{aligned}$$

그림 2. MTA (8,4) 코드의 H 행렬과 오증의 구성
Fig. 2. The H matrix and syndrome of MTA (8,4) code.

$$H = \left| \begin{matrix} 1&1&1&1&1&1&1&0&1&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&0&1&0&1&0&0&0&0&0&0 \\ 1&1&1&1&1&0&1&1&0&0&1&0&0&0&0&0 \\ 1&1&1&1&0&1&1&1&0&0&0&1&0&0&0&0 \\ 1&1&1&0&1&1&1&1&0&0&0&0&1&0&0&0 \\ 1&1&0&1&1&1&1&1&0&0&0&0&0&1&0&0 \\ 1&0&1&1&1&1&1&1&0&0&0&0&0&0&1&0 \\ 0&1&1&1&1&1&1&1&0&0&0&0&0&0&0&1 \end{matrix} \right| \qquad \begin{aligned} S_0 &= \overline{v_7} \oplus v_8 \\ S_1 &= \overline{v_6} \oplus v_9 \\ S_2 &= \overline{v_5} \oplus v_{10} \\ S_3 &= \overline{v_4} \oplus v_{11} \\ S_4 &= \overline{v_3} \oplus v_{12} \\ S_5 &= \overline{v_2} \oplus v_{13} \\ S_6 &= \overline{v_1} \oplus v_{14} \\ S_7 &= \overline{v_0} \oplus v_{15} \end{aligned}$$

그림 3. MTA (16,8) 코드의 행렬과 오증의 구성
Fig. 3. The matrix and syndrome of MTA (16,8) code.

그림 2와 그림 3에서 H행렬의 각 행 간의 최소 해밍거리는 4 로써, 이때 생성된 부호간의 해밍거리는 최소 4가 된다.[2] 또한 더욱 확장된 (32,16), (64,32), 등의 코드에 대해서도 똑같은 관계가 성립된다.

4) 오증생성 단계의 분석

본 절에서는 제안한 복호회로 구성 방식과 기존의

복호회로 구성 방식에 대해 비교 분석하였다. (8,4) 코드의 경우 그림 2와 같은 H행렬로 부터 기존의 방식과 같이 복호회로를 구성하면 그림 4와 같다. 또한 제안한 방식과 같이 복호회로를 구성하면 그림 5와 같다. 복호회로의 감소율을 복호회로의 패리티 비교기의 크기로 비교했을때 약 70%의 감소를 얻을 수 있다. 이는 패리티 비교기의 입력수에 대한 게이트의 비로 계산한 결과로써 기존의 방식은 패리티 비교기가 4입력 XOR 게이트로 구성되어야 하며, MTA 코드의 경우 2입력 XOR 게이트로 구성되기 때문이다. 따라서 복호화 과정에서 복호처리 시간을 2배 개선할 수 있다. 그림 4와 그림 5에서 오류정정 과정의 회로는 비교대상에서 생략하였다.

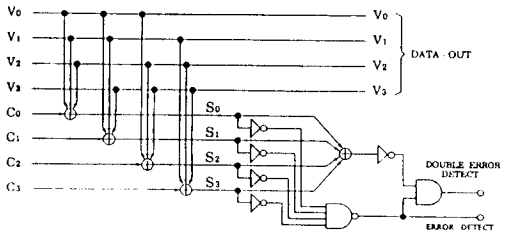

그림 4. 기존의 ECC 방식에 의한 복호회로

Fig. 4. Decoding circuit of conventional ECC technique.

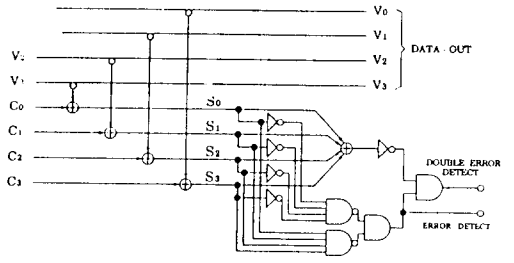

그림 5. 제안한 방식에 대한 복호회로

Fig. 5. Decoding circuit of proposed technique.

기존의 해밍 코드나 Hsiao 코드와 같은 코드들은 정리 8과 같은 성질이 없으므로 오류검출만을 위해서는 일반적으로 $\mathbf{P}^T$ 행렬의 0의 항에 대해 오증식을 구성할 수 없으며, 또한 다른 방법으로도 오증식을 간략화 할 수 없다. $\mathbf{P}^T$ 행렬의 0의 항에 대해 오증식을 구성할 수 있는 경우는 쌍대성(duality)이 성립되는 경우이며 이에 대한 예는 다음과 같다. 기존의 코드들 중에서 H행렬의 $\mathbf{P}^T$ 행렬에서 각 열의 0 과 1의 수가 같을 경우 오증식 구성시 쌍대성이 성립되어 0인 원소에 대해 오증식을 구성할 수 있으며,그렇지 않을 경우 쌍대성이 성립되지 않아서 0인 원소에 대해 오증식을 구성할 수 없다. 그렇지만 이러한 경우에는 복호회로를 줄일 수가 없다. 그 예로써 그림 6에 해밍 SEC-DED코드에 대한 (22,16) 코드의 H 행렬을, 그림 7에 Hsiao SEC-DED 코드의 H행렬을 나타내었다.

$$H = \begin{vmatrix} 1&1&0&1&1&0&1&0&1&0&1&1&0&1&0&1&1&0&0&0&0&0 \\ 1&0&1&1&0&1&1&0&0&1&1&0&1&1&0&0&0&1&0&0&0&0 \\ 0&1&1&1&0&0&0&1&1&1&1&0&0&0&1&1&0&0&1&0&0&0 \\ 0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0&0&0&0&1 \\ 1&1&1&0&1&1&0&1&0&0&1&1&1&0&1&0&0&0&0&0&0&1 \end{vmatrix}$$

그림 6. (22,16) 코드에 대한 해밍 SEC-DED 코드의 H 행렬

Fig. 6. H matrix of Hamming SEC-DED code for (22,16) code.

$$H = \begin{vmatrix} 1&1&1&1&1&1&0&0&0&0&1&0&0&0&1&0&1&0&0&0&0&0 \\ 1&1&1&0&0&0&1&1&1&1&0&0&1&0&0&0&0&1&0&0&0&0 \\ 1&0&0&0&1&0&1&1&1&0&0&0&0&1&1&1&0&0&1&0&0&0 \\ 0&0&0&0&0&1&1&0&0&1&1&1&0&1&1&1&0&0&0&1&0&0 \\ 0&1&0&1&0&0&0&1&0&1&1&1&1&1&0&0&0&0&0&0&1&0 \\ 0&0&1&1&1&1&1&0&0&1&0&0&1&1&0&0&1&0&0&0&0&1 \end{vmatrix}$$

그림 7. (22,16) 코드에 대한 Hsiao SEC-DED 코드의 H 행렬

Fig. 7. H matrix of Hsiao SEC-DED code for (22,16) code.

그림 8에 Hsiao SEC-DED 코드의 (22,16) 코드에 대한 오증을 나타내었으며, 그림 9에 쌍대성에 대한 오증식을 나타내었다. 그림 8과 그림 9를 비교해 보면 오증의 간략화 작업이 의미가 없다는 것을 쉽게 이해할 수 있을 것이다. 그림 8과 그림 9 이외에 다른 부호에 대해서도 별다른 복호회로상의 잇점을 발견할 수 없다.

$$\begin{aligned} S_0 &= V_0 \oplus V_1 \oplus V_2 \oplus V_3 \oplus V_4 \oplus V_5 \oplus V_{10} \oplus V_{14} \oplus V_{16} \\ S_1 &= V_0 \oplus V_1 \oplus V_2 \oplus V_6 \oplus V_7 \oplus V_8 \oplus V_9 \oplus V_{12} \oplus V_{17} \\ S_2 &= V_0 \oplus V_4 \oplus V_6 \oplus V_7 \oplus V_8 \oplus V_{13} \oplus V_{14} \oplus V_{15} \oplus V_{18} \\ S_3 &= V_5 \oplus V_6 \oplus V_9 \oplus V_{10} \oplus V_{11} \oplus V_{13} \oplus V_{14} \oplus V_{15} \oplus V_{19} \\ S_4 &= V_1 \oplus V_3 \oplus V_7 \oplus V_9 \oplus V_{10} \oplus V_{11} \oplus V_{12} \oplus V_{13} \oplus V_{20} \\ S_5 &= V_2 \oplus V_3 \oplus V_4 \oplus V_5 \oplus V_8 \oplus V_{11} \oplus V_{12} \oplus V_{15} \oplus V_{21} \end{aligned}$$

그림 8. (22,16) 코드의 Hsiao SEC-DED 코드에 대한 오증

Fig. 8. Syndrome of Hsiao SEC-DED code for (22,16) code.

$$S_0 = \overline{V_6} \oplus \overline{V_7} \oplus \overline{V_8} \oplus \overline{V_9} \oplus \overline{V_{11}} \oplus \overline{V_{12}} \oplus \overline{V_{13}} \oplus \overline{V_{15}} \oplus v_{16}$$
$$S_1 = \overline{V_3} \oplus \overline{V_4} \oplus \overline{V_5} \oplus \overline{V_{10}} \oplus \overline{V_{11}} \oplus \overline{V_{13}} \oplus \overline{V_{14}} \oplus \overline{V_{15}} \oplus v_{17}$$
$$S_2 = \overline{V_1} \oplus \overline{V_2} \oplus \overline{V_3} \oplus \overline{V_5} \oplus \overline{V_9} \oplus \overline{V_{10}} \oplus \overline{V_{11}} \oplus \overline{V_{12}} \oplus v_{18}$$
$$S_3 = \overline{V_0} \oplus \overline{V_1} \oplus \overline{V_2} \oplus \overline{V_3} \oplus \overline{V_4} \oplus \overline{V_7} \oplus \overline{V_8} \oplus \overline{V_{12}} \oplus v_{19}$$
$$S_4 = \overline{V_0} \oplus \overline{V_2} \oplus \overline{V_4} \oplus \overline{V_5} \oplus \overline{V_6} \oplus \overline{V_8} \oplus \overline{V_{14}} \oplus \overline{V_{15}} \oplus v_{20}$$
$$S_5 = \overline{V_0} \oplus \overline{V_1} \oplus \overline{V_6} \oplus \overline{V_7} \oplus \overline{V_9} \oplus \overline{V_{10}} \oplus \overline{V_{13}} \oplus \overline{V_{14}} \oplus v_{21}$$

그림 9. 재구성한 오증
Fig. 9. Reconstructed syndrome.

$$H = \begin{bmatrix} 1&1&0&1&1&0&1&0&1&0 & 1&0&0&0&0 \\ 1&0&1&1&0&1&1&0&0&1 & 0&1&0&0&0 \\ 0&1&1&1&0&0&0&1&1&1 & 0&0&1&0&0 \\ 0&0&0&0&1&1&1&1&1&1 & 0&0&0&1&0 \\ 1&1&1&0&1&1&0&1&0&0 & 0&0&0&0&1 \end{bmatrix}$$

(a)

$$H = \begin{bmatrix} 1&1&0&1&1&0&1&0&1&0&1 & 1&0&0&0&0 \\ 1&0&1&1&0&1&1&0&0&1&1 & 0&1&0&0&0 \\ 0&1&1&1&0&0&0&1&1&1&1 & 0&0&1&0&0 \\ 0&0&0&0&1&1&1&1&1&1&1 & 0&0&0&1&0 \\ 1&1&1&0&1&1&0&1&0&0&1 & 0&0&0&0&1 \end{bmatrix}$$

(b)

그림 10. (15,10) 와 (16,11) 코드에 대한 해밍 SEC-DED 코드의 H 행렬 (a) 해밍 (15,10) SEC-DED 코드의 H 행렬 (b) (16,11) 해밍 SEC-DED 코드의 H 행렬
Fig. 10. H matrix of Hamming SEC-DED code for (15,10) and (16,11) codes. (a) H matrix of Hamming (15,10) SEC-DED codeword (b) H matrix of Hamming (16,11) SEC-DED codeword.

몇가지 예를 더 살펴보면, 그림 10에 (16,11) 코드와 (15,10) 코드에 대한 해밍 SEC-DED 부호 예를 나타내었으며 이런경우는 쌍대성이 성립되지 않으므로 복호회로의 간략화 단계를 적용 시킬 수 없다. 결과적으로 기존의 해밍 코드나 Hsiao 코드 또는 기존의 ECC 코드기법에서는 오증을 간략화 할 수 없다.

Ⅲ. MTA 코드를 이용한 반도체 메모리 테스트에의 적용

반도체 메모리의 대용량화에 따라 초기의 비교적 간단한 고장모델인 고착고장, 천이고장과, 복잡한 고장모델들로서 결합고장, PSF를 취급하여 분류하고 있다. 또한 이러한 고장들이 서로 연결되어 연쇄고장이나 연결고장을 유발한다. 연쇄고장이나 연결고장과 같은 복합된 고장모델들은 결합고장이나 PSFs와 같은 형태에서 발생하는 것으로써 그 종류가 많고 매우 복잡하며, 이러한 고장 모두를 검출한다는 것은 불가능하다.

1. MTA 코드를 이용한 반도체 메모리 테스트예

본절에서는 반도체 메모리에서의 기능고장 모델들에 의한 고장을 ECC를 기초한 고장형태로 매핑하여 본 방식에서 제안하는 MTA 코드에 의한 메모리 테스트시 고장검출 가능성을 나타내고자 한다.

MTA 코드를 사용하여 적절한 테스트 알고리듬을 적용한다면 반도체 메모리의 기능고장을 효과적으로 검출할 수 있다. 먼저 다음과 같은 가정을 한다.

[가정 1] MTA 코드에 의해 발생된 부호는 테스트 패턴이라 한다.

[가정 2] 반도체 메모리가 물리적으로 분할되어 하나의 블럭이 $\sqrt{n} \times \sqrt{n}$ 구조를 가진다

[가정 3] MTA 코드에서 k가 2보다 큰 짝수이고 P^T 행렬이 대칭행렬일 경우 P^T 행렬의 각 행과 I_m 행렬의 각 행이 서로 보수대칭이 되게 구성가능한 P^T 행렬을 사용한다.

[가정 3] 에 의해 부호의 정보비트와 검사비트가 서로 대칭 혹은 보수대칭이다. 이때 정보비트의 1의 갯수가 짝수이면 대칭, 홀수이면 보수대칭이다. 왜냐하면 [가정 3] 에 의해 정보비트의 1의 수가 홀수일때 정보비트의 1의 원소와 행렬곱에서 P^T 행렬의 0과 정보비트의 1의 곱은 0 이므로 0이 포함된 열만 제외하고 GF(2)에 의해 부호는 모두 1이 되며 이는 검사비트를 구성하게 된다. 또한 정보비트와 I_m 행렬의 행렬곱에서 I_m 행렬의 1인 항만 제외하고 모두 0이 되어 자신의 정보비트가 그대로 나타나서 정보비트를 구성하게 된다. 따라서 이때에는 정보비트와 검사비트가 서로 보수대칭 관계를 가지게 된다. 정보비트의 1의 수가 짝수 일때 발생되는 부호는 정보비트의 1의 원소에 대해 위와 동일한 이유에 의해 P^T 행렬의 0인 열만 제외하고 모두 0이되는 검사비트를 구성하게 된다. 또한 I_m 행렬의 1인 열만 제외하고 모두 0이되는 정보비트를 구성하게 된다. 이때에는 정

보비트와 검사비트가 서로 대칭 관계를 가지게 된다. 따라서 위의 관계를 만족한다.

[가정 3] 에 의한 (8,4)부호의 예를 표 1에 나타내었다. 이때 1, 2, 3, 4 번은 정보비트의 위치이고, 5, 6, 7, 8 번은 검사비트의 위치이다.

위와 같은 가정하에서 테스트 패턴을 발생하여 RAM의 분할된 블럭에 기록하고, 판독하면 복호회로에 의해 고장을 간단히 검출할 수 있다. 한 예로서 고착고장을 검출하기 위해 MTA 코드의 (8,4) 코드를 테스트 패턴으로 사용하여 표 1의 TP1을 기록한 후 판독한다면 stuck-at-1 고장에 의한 삼중고장을 검출할 수 있다. 이를 확장하여 더욱 복잡한 기능고장을 검출하기 위해 테스트 알고리듬을 설정하면 간단히 테스트를 완료할 수 있다. 이러한 고장을 검출하는 회로는 테스트 패턴발생기, 복호회로, 그이외의 간단한 제어회로를 사용하여 테스트회로의 BIST설계를 효과적으로 완료할 수 있다.

Ⅳ. 결론

본 논문에서는 ECC 기법에 기초하여 MTA 코드를 제안하였으며 이에 대해 분석하였다. 이 코드는 반도체 메모리의 테스트를 효과적으로 수행하기 위한 방식으로써, 메모리 블럭의 물리적인 분할을 고려하여 정보장과 검사장을 같게 설정하였다. 또한 복호회로를 줄이기 위해 **H**행렬을 적절히 구성하였으며, 오류검출시 MTA 코드의 복호회로를 줄일 수 있도록 오증식을 구성하였다. 따라서 기존의 복호방식보다 복호회로를 약 70% 정도 감소할 수 있었다. 또한 복호처리 시간을 2배 이상 개선할 수 있으며, 시스템의 속도를 더욱 개선할 수 있다.

특히 메모리 기능고장 검출시에 MTA 코드는 검사비트를 저장하기 위한 부가면적을 필요로 하지 않기 때문에 메모리의 기능 테스트에 적용이 용이하다. 또한 제안한 MTA 코드는 반도체 메모리의 단일고장과 대부분의 이중고장 및 삼중고장을 검출한다.

RAM에서 발생 가능한 각종 기능고장의 검출시 MTA 코드를 사용하여 적절한 알고리듬을 적용한다면 효과적으로 검출 가능하다는 것을 일 예로 제시하였다. 또한 BIST 회로로 구현할 경우 기존의 ECC 기법에 의한 부가회로상의 부담을 해결할 수 있을 뿐만 아니라, 테스트회로를 간단히 구성할 수 있어서, 기존의 반도체 메모리 테스트 알고리듬에 비해 더욱 효과적으로 구현가능하다는 것을 보였다.

參 考 文 獻

[1] R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell System Tech. J* 29, pp. 147-160, April 1950.

[2] M. Y. Hsiao, "A Class of Optimal Minimum Odd-weight-column SEC-DED Codes," *IBM J. Res. Develop.* 14, pp. 395-401, July 1970.

[3] R. C. Bose, and D. K. Ray-Chaudhuri, "On A Class of Error Correcting Binary Group Codes," Information and Control 3, pp. 68-79, 1960.

[4] I. S. Reed, and G. Solomon, "Polynomial Codes over Certain Finite Fields," J. Soc. Ind. Appl. Math. 8, pp. 300-304, June 1960.

[5] T. R. N. Rao and E. Fujiwara, "Error-Control Coding for Computer Systems," Prentice-Hall, 1989.

[6] K.Furutani, et al., "A Built-In Hamming Code ECC Circuit for DRAM's," *IEEE J. Solid-State Circuits*, vol. 24, no.1, pp. 50-56, Feb. 1989.

[7] M.Asakura et al., "An Experimental 1-Mbit Cache DRAM with ECC," *IEEE J. Solid-State Circuits*, vol. 25, no. 1, Feb. 1990.

著 者 紹 介

李 仲 鎬(正會員)

1965年 1月 23日生. 1988年 2월 울산대학교 전자 및 전산기공학과 졸업(학사). 1990年 2월 울산대학교 대학원 전자 및 전산기공학과 졸업(석사). 1994年 8월 울산대학교 대학원 전자 및 전산기 공학과 박사학위 수여 예정. 주관심 분야는 Testable design, VLSI system 설계, ASIC Design 및 Test 등임.

趙 相 福(正會員)

1955年 6月 10日生. 1979年 2月 한양대학교 전자공학과 졸업(학사). 1981年 2月 한양대학교 대학원 전자공학과 졸업(석사). 1985年 2月 한양대학교 대학원 전자공학과 박사. 1981年 2月 ~ 1986年 2月 광운대학교, 한양대학교 강사. 1986年 2月 ~ 현재 울산대학교 전자 및 전산기공학과 부교수. 주관심 분야는 VLSI/ULSI 테스트 및 설계, 초고집적 RAM의 Test 및 Testable Design, ASIC Design, VLSI CAD tool 개발 등임.