理學碩士 學位論文

셀룰라 오토마타에 기초한 오류 정정 부호

2002 年 8 月

釜 慶 大 學 校 大 學 院

應用數學科

高 貴 子

理學碩士 學位論文

셀룰라 오토마타에 기초한 오류 정정 부호

指導敎授 趙 聖 鎭

이 論文을 理學碩士 學位論文으로 提出함

2002 年 8 月

釜 慶 大 學 校 大 學 院

應用數學科

高 貴 子

高貴子의 理學碩士 學位論文을 忍准함

2002年 6 月 27日

主 審 理學博士 鄭 震 文 (印)

委 員 理學博士 沈 孝 燮 (印)

委 員 理學博士 趙 聖 鎭 (印)

< 차 례 >

Cellular Automata Based Error-Correcting Code

Gwi Ja Ko

Department of Applied Mathematics, Graduate School.
Pukyoung National University

Abstract

In this paper, a simple and efficient scheme for generating SEC-DED codes and decoding received codewords is presented by using the characteristic matrix of the CA.

And this reports a novel approach for designing byte error-correcting codes using cellular automata (CA). A simple scheme for generation and decoding of single-byte error-correcting and double-byte error-detecting codes, referred to as CA-SbEC-DbED, is presented. Extension of the scheme to locate/correct larger number of information byte errors has been also included. The encoding and decoding algorithms have been designed with the help of a linear operator that can be conveniently realized with a maximum length group CA. The regular, modular and cascadable structure of CA can be economically built with VLSI technology. Compared to the existing architecture of the Reed-Solomon decoder chip, CA-based implementation of the proposed decoding scheme provides a simple cost effective solution.

1. 서론

셀룰라 오토마타(Cellular Automata, 이하 CA)는 Von Neumann과 Ulam에 의해 스스로 조직화하고 재생산할 수 있는 모델로 소개되었다[10,12]. CA란 동역학계(dynamical system)를 해석하는 한 방법으로 공간과 시간을 이산적으로 다루고, 이산적인 공간을 셀룰라 공간(cellular space)의 기본 단위인 각 셀이 취할 수 있는 상태를 유한하게 처리하며, 각 셀들의 상태가 국소적인 상호작용에 의해서 동시에 갱신되는 시스템이다. CA는 그 후 Wolfram에 의하여 처음으로 암호학에 도입되었다[11]. 이후 Das 등에 의해서 행렬 대수학으로 분석이 이루어졌으며[7,8], Chaudhuri, Nandi 등이 많은 분야에 CA를 폭넓게 활용하였다[1]. 오류 검출 및 정정 부호는 메모리 시스템 설계, 디지털 데이터 통신 등에 널리 적용된다. 오류 검출 및 정정 부호는 원래의 정보에 검사비트 형태인 용장도(redundancy)를 추가함으로써 설계된다. 그래서 부호어의 Hamming 거리(distance)가 증가한다.

본 논문의 구성은 다음과 같다. 2장에서는 CA에 관한 전반적인 이론으로 CA Rule 구성방법과 여러 가지 CA의 종류에 대해 알아보고 1차원 CA의 전이행렬과 특성다항식을 구해본다. 3장에서는 CA에 기초한 비트 오류 정정 부호에 관한 이론으로 임의의 정보비트에 대한 SEC-DED 부호를 생성하고 그에 따른 올바른 복호 방법을 알아본다. 4장에서는 CA에 기초한 바이트 오류 정정 부호에 관한 이론으로 CA-SbEC-DbED Code의 생성 방법과 그에 따른 올바른 복호 방법을 알아보고 5장에서는 결론을 맺는다.

2. 1차원 Cellular Automata

2.1. 1차원 CA의 정의

CA란 유한개의 셀(cell)이 일정한 형태로 배열되어 있고, 셀의 상태가 자신을 포함한 셀의 이웃에 의존하여 전이되는 것이다.
3-이웃(3-neighborhood) CA의 다음상태전이 함수는 다음과 같이 나타난다.

$$q_i(t+1) = f[q_{i-1}(t), q_i(t), q_{i+1}(t)]$$

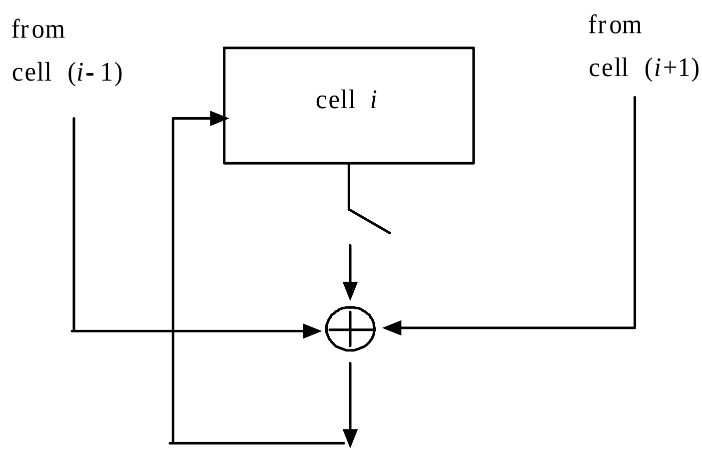
여기서 i 는 일차원으로 배열되어 있는 각 셀들의 위치이고 t는 시간 단계이며 $q_i(t)$는 시간 t에서 i번째 셀의 상태, $q_i(t+1)$는 시간 $t+1$에서 i번째 셀의 상태를 말한다.

2.2. CA Rule

3-이웃 다음상태전이 함수에서 f 는 결합논리를 가지는 국소 전이 함수이다. f 는 3개의 변수를 가지는 Boolean 함수이며 따라서 2^{2^3}(256)개의 상태전이 함수로 되어있고, 이것을 CA Rule이라 한다.

예를 들어 rule 90은 왼쪽과 오른쪽 셀에 영향을 받고, rule 150은 왼쪽과 자기 자신, 오른쪽 셀에 영향을 받아 다음상태가 만들어진다.

<그림 2.1>은 rule 90과 rule 150의 선형 셀 구조이다.



<그림 2.1> Rule 90과 Rule 150의 선형Cell 구조

3-이웃 CA에서 rule 90과 rule 150의 상태배열의 예를 살펴보면 다음과 같다.

$(q_{i-1}(t), q_i(t), q_{i+1}(t))$	(1,1,1)	(1,1,0)	(1,0,1)	(1,0,0)	(0,1,1)	(0,1,0)	(0,0,1)	(0,0,0)	rule
$q_i(t+1)$	0	1	0	1	1	0	1	0	rule 90
$q_i(t+1)$	1	0	0	1	0	1	1	0	rule 150

<**표**2.1>**은** CA Rule **이다**.

Linear Rule		Complemented Rule	
rule 60	$q_i(t+1) = q_{i-1}(t) \oplus q_i(t)$	rule 195	$q_i(t+1) = \overline{q_{i-1}(t) \oplus q_i(t)}$
rule 90	$q_i(t+1) = q_{i-1}(t) \oplus q_{i+1}(t)$	rule 165	$q_i(t+1) = \overline{q_{i-1}(t) \oplus q_{i+1}(t)}$
rule 102	$q_i(t+1) = q_i(t) \oplus q_{i+1}(t)$	rule 153	$q_i(t+1) = \overline{q_i(t) \oplus q_{i+1}(t)}$
rule 150	$q_i(t+1) = q_{i-1}(t) \oplus q_i(t) \oplus q_{i+1}(t)$	rule 105	$q_i(t+1) = \overline{q_{i-1}(t) \oplus q_i(t) \oplus q_{i+1}(t)}$
rule 170	$q_i(t+1) = q_{i+1}(t)$	rule 85	$q_i(t+1) = \overline{q_{i+1}(t)}$
rule 204	$q_i(t+1) = q_i(t)$	rule 51	$q_i(t+1) = \overline{q_i(t)}$
rule 240	$q_i(t+1) = q_{i-1}(t)$	rule 15	$q_i(t+1) = \overline{q_{i-1}(t)}$

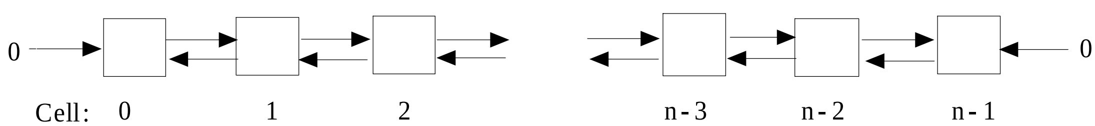
<**표**2.1> Additive CA Rules

2.3. CA의 종류

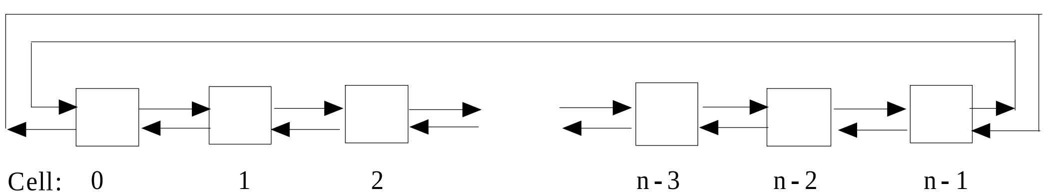
[정의 2.1] CA Rule**의 논리 종류에 따라 각 셀에 적용된 법칙이** XOR **논리로만 이루어진** CA**를 Linear CA 라하고 각 셀에 적용되는 법칙이** XNOR**과** XOR **논리로 이루어진** CA**를 Complemented CA라 한다**.

[정의 2.2] CA**의 셀이 몇 가지** Rule**에 의하여 전이되는가에 따라** CA**의 각 셀에 모두 같은 법칙이 적용된** CA**를 Uniform CA라하고** 2**가지 이상의 서로 다른 법칙이 적용된** CA**를 Hybrid CA라 한다**.

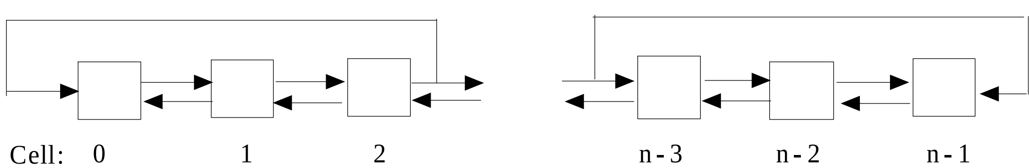
[정의 2.3] CA**에서 가장 왼쪽과 오른쪽의 셀은** 2**개의 이웃만을 가지므로 세 번째 이웃의 상태를 결정해 주어야 한다. 이것을** CA**의 경계조건이라 한다. 이** CA **경계조건에 따라 제일 왼쪽과 오른쪽의 셀들이** 0**상태에 연결되어 있는** CA**를 Null boundary CA(NBCA)라 하고 양끝의 셀들이 서로 연결되어 있는** CA**를 Periodic Boundary CA(PBCA)**, **가장 왼쪽**(**오른쪽**) **셀의 다음 상태가 그 자신과 그것의 오른쪽**(**왼쪽**) **이웃**, **두 번째 오른쪽**(**왼쪽**) **이웃 셀의 상태에 의존하는** CA**를 Intermediate Boundary CA(IBCA)라 한다**.



<**그림** 2.2> Null Boundary CA



<**그림** 2.3> Periodic Boundary CA



<**그림** 2.4> Intermediate Boundary CA

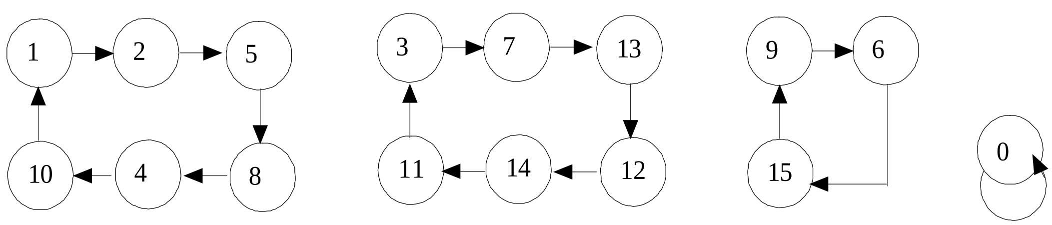
예를 들어 4-cell PBCA와 NBCA에 rule 90인 uniform CA를 이용해 이들의 전이행렬과 상태전이 그래프를 비교해보자.
다음은 각각 PBCA와 NBCA의 전이행렬이다.

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ \\ 0 & 10 & 5 & 15 & 10 & 0 & 15 & 5 & 5 & 15 & 0 & 10 & 15 & 5 & 10 & 0 \end{bmatrix}$$

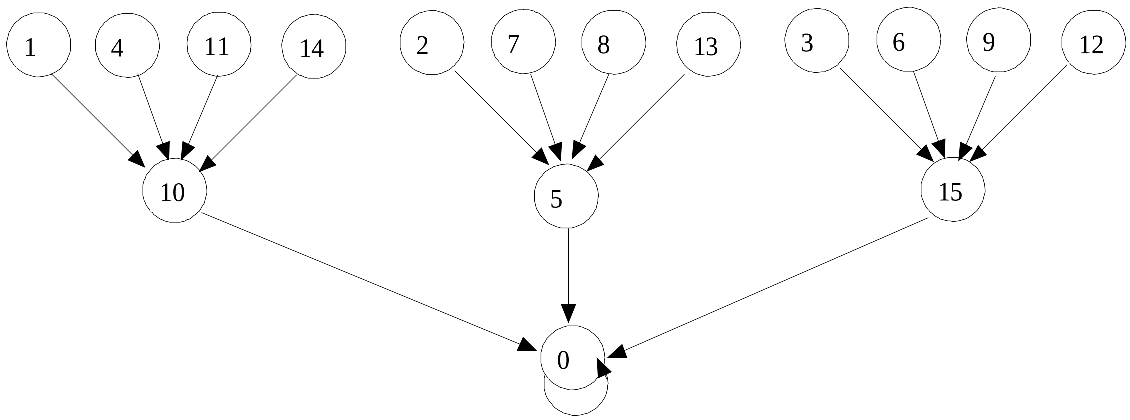
이런 CA는 모든 상태의 직전자 수가 0 또는 1개 보다 많기 때문에 순환하지 않음이 분명하고, 상태전이 그래프에서 초기상태는 어떤 직전자도 갖지 않음을 볼 수 있다.

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ \\ 0 & 2 & 5 & 7 & 10 & 8 & 15 & 13 & 4 & 6 & 1 & 3 & 14 & 12 & 11 & 9 \end{bmatrix}$$

이런 CA의 모든 상태는 한 개의 직전자를 갖고 순환상태에 놓여 있음을 알 수 있다.
다음은 이 두 전이행렬의 상태전이 그래프이다.



<그림 2.5> Rule 90인 Uniform CA를 가지고 만들어진 NBCA 상태전이 그래프



<그림 2.6> Rule 90인 Uniform CA를 가지고 만들어진 PBCA 상태전이 그래프

위의 두 그림으로부터 PBCA와 NBCA는 같은 rule을 가지고도 다른 상태전이 그래프가 그려짐을 알 수 있다.

[정의 2.4] CA Rule에 의해 변화되는 상태그래프의 형태에 따라 모든 셀들의 상태가 몇 개의 사이클을 이루며 반복되는 것을 **Group CA**라 하고 group CA가 아닌 CA를 **Nongroup CA**라 한다.

2.4. 1차원 CA의 전이행렬과 특성다항식

n 개의 셀을 가지는 1-차원 CA에서는 현재 상태를 다음 상태로 전이시키는 작용소(operator)를 $n \times n$ 행렬로 나타낼 수 있으며 이것을 전이행렬(transition matrix)이라 한다. 전이행렬 T 에서 i 번째 행은 i 번째 셀에 적용되는 rule이며 그 셀의 다음 상태가 현재 상태에 의존하면 1, 그렇지 않으면 0으로 쓴다.

$f_{(t)}(x)$가 시간 t에서 CA의 상태를 나타내면 시간 $t+1$에서의 상태는 다음 식으로 표현된다.

$$f_{(t+1)}(x) = T \times f_{(t)}(x)$$

$$f_{(t+p)}(x) = T^{p} \times f_{(t)}(x)$$

<예 2.1> 네 개의 셀을 가지는 1차원 NBCA의 rule이 [150, 150, 90, 150] 일 때 전이행렬과 다음 상태를 구해보자.

전이행렬 T는 CA rule 에 의해 다음과 같이 구해진다.

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

그리고 이 CA의 현재 상태가 $f_{(t)}(x) = [\,1\ 1\ 0\ 0\,]^{T}$ 이면 다음 상태는 전이행렬을 이용해서 다음과 같이 구할 수 있다.

$$f_{(t+1)}(x) = T \times f_{(t)}(x) = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

□

CA의 전이행렬이 T일 때, T의 특성다항식(Characteristic Polynomial) $C(x)$은 다음과 같다. 여기서 I는 n차 단위행렬이다.

$$C(x) = |T + xI|$$

<예 2.2> 위의 <예 2.1>에서 사용한 T 행렬에 대한 특성다항식을 구해보자.

$$|T + xI| = \begin{vmatrix} x+1 & 1 & 0 & 0 \\ 1 & x+1 & 1 & 0 \\ 0 & 1 & x & 1 \\ 0 & 0 & 1 & x+1 \end{vmatrix}$$

$$= (x+1)\begin{vmatrix} x+1 & 1 & 0 \\ 1 & x & 1 \\ 0 & 1 & x+1 \end{vmatrix} + 1\begin{vmatrix} 1 & 0 & 0 \\ 1 & x & 1 \\ 0 & 1 & x+1 \end{vmatrix} = x^4 + x^3 + 1$$

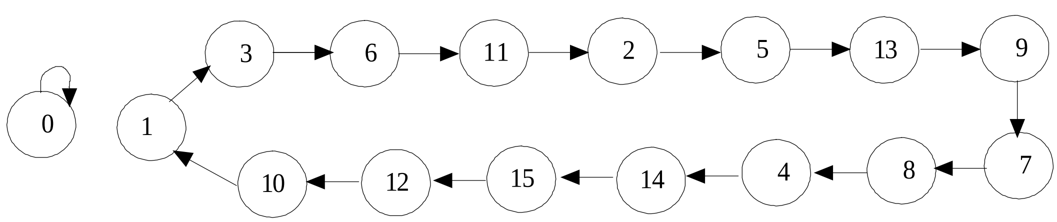
□

CA의 전이행렬 T가 $|T| = 1$이면 그 CA는 group CA이다. group CA는 최대길이를 갖는 CA와 최대길이를 갖지 않는 CA로 구별할 수 있다.

n 개의 셀로 이루어진 CA에서 모든 셀의 상태가 0인 경우를 제외한 $2^n - 1$개의 상태가 하나의 사이클 안에 있을 때 최대 길이를 가진다고 한다.

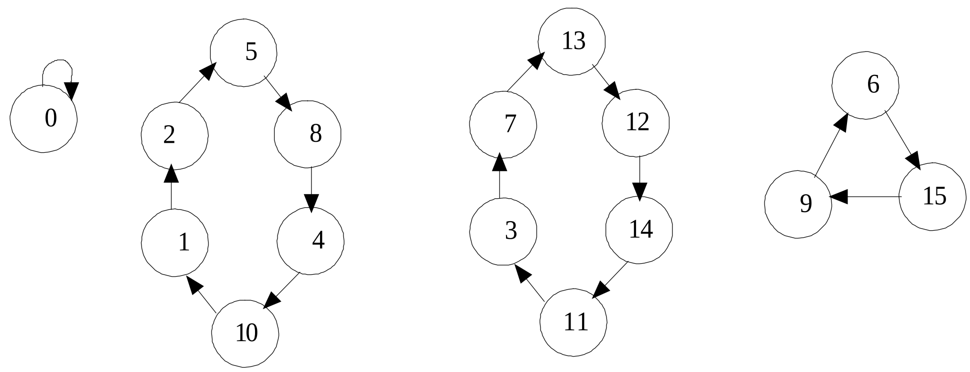
아래 그림은 rule [90, 150, 90, 150]인 4개의 cell로 구성된 최대길이를 가지는 CA이고 이것은 4장에서 다룰 바이트 오류에서 중요하게 사용될 것이다.

<그림 2.7>은 최대길이를 갖는 group CA이다.



<그림 2.7> 최대 길이를 갖는 Group CA

<그림 2.8>은 최대길이를 가지지 않는 선형 CA로써 rule [90, 90, 90, 90]을 갖는 uniform CA이다.

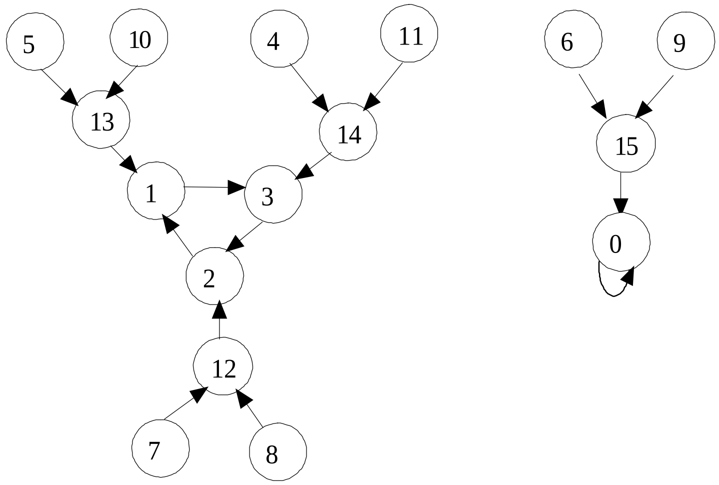


<그림 2.8> 최대 길이를 갖지 않는 Group CA

그림에서 알 수 있듯이 0을 제외한 다른 상태들이 몇 개의 서로 다른 사이클로 분리되어 있다. 이 사이클 길이의 최소공배수가 이 CA의 주기가 된다.

CA의 전이행렬 T가 $|T| = 0$ 이면 그 CA는 nongroup CA이다. nongroup CA는 전이행렬에 대한 역행렬이 존재하지 않으므로 임의의 상태에 대하여 이전 상태를 명확하게 알 수 없다.

아래 그림은 rule [102, 60, 90, 60]을 갖는 nongroup CA의 상태전이 그래프이다.



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

<**그림** 2.9> Nongroup CA

3. CA에 기초한 비트 오류 정정 부호

3.1. 오류 정정 부호의 기초

오류 정정 부호(Error Correcting Codes)**에서 부호 길이**(code length), **차원**(dimension), **최소 거리**(minimum distance)**는 가장 중요한 요소이다**. **고정된 유한 체**(finite field)**상에서 부호길이가 일정하다고 할 때**, **차원과 최소 거리가 클수록 좋은 부호라고 할 수 있다**.

q**개의 원소를 갖는 유한 체** $\mathbb{F}_q$**에서 각각의** $v_i(1\leq i\leq n)$**를 취하여** n**개의 성분을 갖는 벡터** $v=(v_1, v_2, \cdots, v_n)$**를 만들고**, **이러한 벡터들로 구성되는** n**차원 벡터공간** $V=\mathbb{F}_q{}^n$**을 생각한다**. q**진 선형** (n, k)**부호** C**는 벡터공간** $\mathbb{F}_q{}^n$**의** k**차원 부분공간이며**, C**의 원소들을 부호어**(codeword)**라고 부른다**. **여기서** n**은** C**의 부호길이**, k**는** C**의 차원이 된다**.

[정의 3.1] 체 $\mathbb{F}_q$ **위의 벡터공간** $\mathbb{F}_q{}^n$ **에서**, **두 벡터** $v=(x_1, \cdots, x_n)$, $w=(y_1, \cdots, y_n)$**의 성분 중에서 서로 다른 성분의 개수를** v, w **의 Hamming 거리 또는 간단히 거리(distance)라 하고 이것을** $d(v, w)$ **로 나타낸다**. **또**, **벡터** v **의 성분 중에서** 0 **이 아닌 성분의 개수를** v **의 Hamming 무게 또는 간단히 무게(weight)라 하고 이것을** $wt(v)$ **또는** $w(v)$ **로 나타낸다**.

$$d(v, w) = |\{i \mid x_i \neq y_i, 1\leq i\leq n\}|$$

$$wt(v) = w(v) = |\{i \mid x_i \neq 0, 1\leq i\leq n\}|$$

[정의 3.2] 체 $\mathbb{F}_q$ **위의 선형** (n, k) **부호** C **에 대하여**

$$d(C) = \min\{d(v, w) \mid v, w \in C,\ v \neq w\},$$
$$wt(C) = \min\{wt(v) \mid v \in C,\ v \neq \mathbf{0}\}$$

를 각각 C **의 최소 거리(minimum distance), 최소 무게(Hamming weight)라고 한다.**

또, $d(C) = d$ **일 때** C **를** $\mathbb{F}_q$ **위의 선형** $(\boldsymbol{n}, \boldsymbol{k}, \boldsymbol{d})$ **부호라고 한다.**

[정의 3.3] 체 $\mathbb{F}_q$ **위의** $m \times n$ **행렬** G **에 대하여**

$$C = (G \text{ 의 행공간})$$

일 때, G **를 부호** C **의 생성행렬(generator matrix)이라고 한다.**

특히, 체 $\mathbb{F}_q$ **위의** $k \times n$ **행렬** G **가**

$$G = [I_k \mid D] \qquad (I_k \text{ 는 } k \text{ 차의 단위행렬})$$

와 같은 꼴일 때, 분명히 $r(G) = k$ **이므로** G**를 생성행렬로 가지는 부호** C **는 체** $\mathbb{F}_q$ **위의 선형** (n, k) **부호이다. 이 경우에 행렬** G **를 부호** C **의 표준 생성행렬(standard generator matrix)이라고 한다.**

[정의 3.4] 체 $\mathbb{F}_q$ **위의** $m \times n$**행렬** H **에 대하여, 체** $\mathbb{F}_q$ **위의 부호** C **가**

$$H \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

의 해 공간 일 때, H **를** C **의 패리티 검사 행렬(parity check matrix)이라고 한다.**

여기서 $r(H) = r$ **이면** C **는** $\mathbb{F}_q$ **위의 선형** $(n, n-r)$ **부호이다.**

특히, 체 $\mathbb{F}_q$ **위의** $(n-k) \times n$ **행렬** H **가**

$$H = [A \mid I_{n-k}] \qquad (I_{n-k} \text{ 는 } n-k \text{ 차의 단위행렬})$$

와 같은 꼴일 때, $r(H) = n-k$ **이므로** H **를 패리티 검사 행렬로 가지**

는 부호 C 는 체 $\mathbb{F}_q$ 위의 선형 (n, k) 부호이다.

이 경우에 H 를 부호 C 의 표준 패리티 검사 행렬(standard parity check matrix)이라고 한다.

3.2. 부호화의 기본 원리

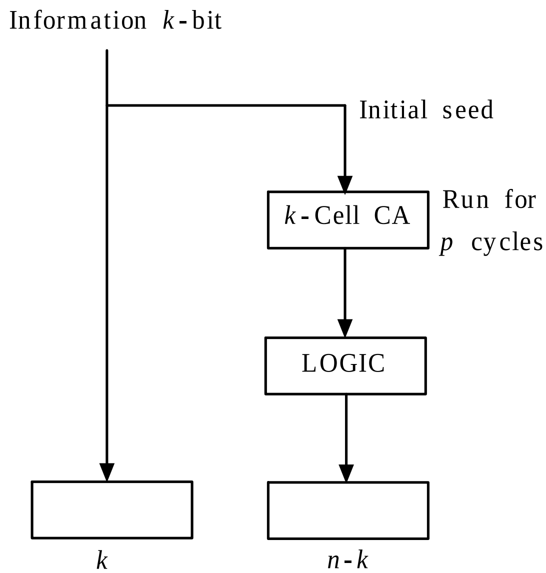
CA의 규칙적인 구조를 이용하여 부호계를 생성하는 기본 방법을 알아본다. 적당한 CA를 선택하면 최소 거리(minimum distance)가 $d(d \geq 4)$인 부호를 생성할 수 있다.

기본 개념은 정보비트를 seed로 적재하고 얼마간의 주기동안 CA를 자동적으로 실행시키는 것이다. CA의 마지막 상태로부터 검사비트는 생성된다. <그림 3.1>은 제시된 방법을 보여준다. (n, k, d) 부호에서 입력된 k 정보비트가 CA에 적재되고 p 주기동안 작동된다. n과 k 값에 의존하는 LOGIC 블록은 다음 두 함수 중 하나를 실행함으로써 CA의 k 비트 상태에서 $(n-k)$ 비트의 검사비트를 생성한다.

(a) 압축 (compressing) : $(n-k) < k$일 때, 검사비트의 부분집합을 선택.

(b) 확장(expanding) : $(n-k) > k$일 때, 몇 개의 비트를 XOR하여 부가된 검사비트를 생성.

$(n-k) = k$이면 LOGIC 블록은 검사비트인 CA상태로 바로 전송된다.



<그림 3.1> **검사비트의 생성**

T 를 CA의 특성행렬, I 를 k 비트 정보어 라고 하면, 검사어는 LOGIC 블록에 의해서 $T^{p}[I]$ 또는 $T^{p_1}[I], T^{p_2}[I], \cdots$로 생성된다.

부호어는 아래에 의해서 주어진다.

$C = I, T^{p}[I]$; 단일 멱의 p가 주어졌을 때

$C = I, [T^{p_1}[I], T^{p_2}[I], \cdots]$; 다중 멱의 p가 주어졌을 때

k 비트인 정보어가 $\{i_1, i_2, i_3, \cdots i_k\}$이고, $(n-k)$ 비트인 검사어가 $\{c_1, c_2, c_3, \cdots c_{n-k}\}$이면, 부호어는

$C = \{i_1, i_2, i_3, \cdots, i_k, c_1, c_2, c_3, \cdots, c_{n-k}\}$가 된다.

<**예 3.1**> (8, 4, 4) **부호를 생성하는 방법을 알아보자.**

정보비트 k**와 검사비트** $n-k$**가** 4**로 동일하다. 법칙** [90, 150, 150, 90] **에** 4-cell CA**를 택하고** 4**비트 검사비트를 생성하기 위해서** 4**비트 정보어를 적재하고** $p=2$**인 주기동안 실행한다.**

$I=\{i_1, i_2, i_3, i_4\}$**가** 4**비트 정보어이고,** T**가** CA**의 특성행렬이라면,** 4**비트 검사어** $CW=\{c_1, c_2, c_3, c_4\}$**는**

$$CW = T^2[I]$$

이다.

따라서, 8 **비트 부호어는**

$$C = I, CW = I, T^2[I]$$
$$= \{i_1, i_2, i_3, i_4, c_1, c_2, c_3, c_4\}$$

이 된다.

4 **비트 정보어가** (1010)**일 때, 부호어를 구해보자.**

법칙 [90, 150, 150, 90]**에 대한 특성행렬은** $T=\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$**이며,**

$T^2=\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$**이므로, 검사어는**

$$CW = T^2[I] = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$ **이 된다.**

따라서 부호어는 8 **비트** $[I, CW] = (10100101)$**가 된다.** □

[정리 3.1] 특성행렬 T를 가지는 k-cell CA는, 어떤 정수 p에 대해서 T^p이 다음 조건을 만족할 때, 거리가 d인 부호를 생성한다 :

모든 정수 i, $0 < i < d$ 에 대해서 T^p의 어떤 i개의 열의 비트별 합 (bitwise sum)은 적어도 $(d - i)$개의 1을 포함한다.

(증명) T^p이 정리의 조건을 만족한다고 하고 I_1과 I_2가 서로 다른 두 개의 정보어 라고 하자.

단, I_1과 I_2는 k 비트 단어들이다.

$C_1 = I_1, T^p[I_1]$과 $C_2 = I_2, T^p[I_2]$일 때 $d(C_1, C_2) \geq d$임을 보이고자 한다.

T가 $k \times k$ 행렬이므로 $T^p[I_1]$과 $T^p[I_2]$는 k 비트 단어이며, $C_1 = I_1, T^p[I_1]$과 $C_2 = I_2, T^p[I_2]$는 $2k$ 비트 부호어(codeword) 이다.

$d(I_1, I_2) = i$, 즉 I_1과 I_2가 i비트 위치만큼 다르다고 하자. 또 다른 i비트 위치를 $n_1, n_2, \cdots, n_i$라고 하자.

만약 $i \geq d$이면

$C_1 = I_1, T^p[I_1]$과

$C_2 = I_2, T^p[I_2]$는

I_1과 I_2에서만 i 비트 이상 차이가 나므로 $T^p[I_1]$과 $T^p[I_2]$가 똑 같다 하더라도 i 비트 이상 차이가 난다.

따라서 $d(C_1, C_2) \geq i \geq d$ 이다. 그러므로 $i < d$ 라고 가정하자.

행렬 T^p에서 $n_1, n_2, \cdots, n_i$ 번째 열 $(T^p)_{n_1}$, $(T^p)_{n_2}$, $\cdots$, $(T^p)_{n_i}$

의 비트별 합 $(T^p)_{n_1} + (T^p)_{n_2} + \cdots + (T^p)_{n_i}$은 가정에 의하여 적어도 $d-1$개의 1을 포함한다. 그 위치를 $r_1, r_2, \cdots, r_{t-i}, \cdots, r_j$라 하자.

$T^p[I_1]$과 $T^p[I_2]$의 r_1번째 위치에 있는 비트를 b_1과 b_2라 하면 각 $i = 1, 2$ 에 대하여

$$T^p[I_i] = \begin{bmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,k} \\ \vdots & \vdots & \cdots & \vdots \\ t_{r_1,1} & t_{r_1,2} & \cdots & t_{r_1,k} \\ \vdots & \vdots & \cdots & \vdots \\ t_{k,1} & t_{k,2} & \cdots & t_{k,k} \end{bmatrix} \begin{bmatrix} I_{i1} \\ I_{i2} \\ \vdots \\ I_{ik} \end{bmatrix}$$

$$= \begin{bmatrix} t_{1,1}I_{i1} & + & t_{1,2}I_{i2} & + & \cdots & + & t_{1,k}I_{ik} \\ \vdots & & \vdots & & \cdots & & \vdots \\ t_{r_1,1}I_{i1} & + & t_{r_1,2}I_{i2} & + & \cdots & + & t_{r_1,k}I_{ik} \\ \vdots & & \vdots & & \cdots & & \vdots \\ t_{k,1}I_{i1} & + & t_{k,2}I_{i2} & + & \cdots & + & t_{k,k}I_{ik} \end{bmatrix}$$

이므로

i가 1일 때

$$b_1 = t_{r_1,1}I_{11} + t_{r_1,2}I_{12} + \cdots + t_{r_1,k}I_{1k} \tag{3.1}$$

i가 2일 때

$$b_2 = t_{r_1,1}I_{21} + t_{r_1,2}I_{22} + \cdots + t_{r_1,k}I_{2k} \tag{3.2}$$

여기서, I_{ij}는 I_i의 j번째 비트이고, $t_{ij} = T^p[i,j]$이다.

(3.1)과 (3.2)로부터

$$b_1 + b_2 = t_{r_1,1}(I_{11} + I_{21}) + t_{r_1,2}(I_{12} + I_{22}) + \cdots + t_{r_1,k}(I_{1k} + I_{2k})$$

을 얻는다. 그런데

$$I_{1j} + I_{2j} = \begin{cases} 1, & \text{if } j = n_1, n_2, \cdots, n_i \\ 0, & otherwise. \end{cases}$$

이고,

$$(T^p)_{n_1} + (T^p)_{n_2} + \cdots + (T^p)_{n_i}$$

$$= \begin{bmatrix} t_{1,n_1} \\ t_{2,n_1} \\ \vdots \\ t_{r_1,n_1} \\ \vdots \\ t_{k,n_1} \end{bmatrix} + \begin{bmatrix} t_{1,n_2} \\ t_{2,n_2} \\ \vdots \\ t_{r_1,n_2} \\ \vdots \\ t_{k,n_2} \end{bmatrix} + \cdots + \begin{bmatrix} t_{1,n_i} \\ t_{2,n_i} \\ \vdots \\ t_{r_1,n_i} \\ \vdots \\ t_{k,n_i} \end{bmatrix}$$

$$= \begin{bmatrix} t_{1,n_1} + t_{1,n_2} + \cdots + t_{1,n_i} \\ t_{2,n_1} + t_{2,n_2} + \cdots + t_{2,n_i} \\ \vdots \\ t_{r_1,n_1} + t_{r_1,n_2} + \cdots + t_{r_1,n_i} \\ \vdots \\ t_{k,n_1} + t_{k,n_2} + \cdots + t_{k,n_i} \end{bmatrix}$$

에서 $r_1, r_2, \cdots, r_{t-i}, \cdots, r_j$ 번째 비트가 1이므로,

$b_1 + b_2 = t_{r_1,n_1} + t_{r_1,n_2} + t_{r_1,n_3} + \cdots + t_{r1,n_i} = 1$ 이다.

$T^p[I_1]$과 $T^p[I_2]$의 r_1번째 비트인 b_1과 b_2가 $b_1 + b_2 = 1$ 이므로 다르다는 것을 알 수 있다.

마찬가지로 $T^p[I_1]$**과** $T^p[I_2]$**의** $r_2, \cdots, r_{d_i}, \cdots, r_j$ **번째 비트가 모두 다르다는 것을 알 수 있다**.

그러므로 $d(T^p[I_1], T^p[I_2]) \geq d - i$ **이고** $d(I_1, I_2) = i$ **이므로** $d(C_1, C_2) \geq d$ **이다**. □

[정리 3.2][3] T^p**가 다음 세 조건을 만족하면** CA**는 주기** p**에서** $(2k, k, 4)$ **부호를 생성한다**.

(a) T^p**의 모든 열은 적어도 세 개의** 1**을 포함한다**.

(b) T^p**의 어떤 두 개의 열은 적어도 두 위치에서 다르다**.

(c) T^p**의 어떤 세 개 열의 비트별 벡터 합은 영 벡터**(zero vector)**가 아니다**.

[정리 3.3][3] k-cell CA**가** p**주기** $(2k, k, d)$ **부호를 생성한다면**, $p \geq d - 2$ **이다**.

3.3. 임의의 정보비트 수에 대한 SEC-DED 부호의 생성

이 절에서는 임의의 정보비트에 대한 결과를 일반화한다.

임의의 주어진 T_4에서 시작하여, $k>4$에 대해서 T_k는 다음 알고리즘을 이용하여 구성될 수 있다.

<알고리즘 3.1> T_k의 생성 ($k>4$)

제 1단계 : 기저(basis) 행렬로 T_4를 택한다.

$$T_4 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

제 2단계 : $k \leftarrow 4$

제 3단계 :

$$T_{k+1} = \begin{bmatrix} & & & & 0 \\ & & & & 0 \\ & \begin{bmatrix} T'_k \end{bmatrix} & & & \vdots \\ & & & & 0 \\ & & & & 1 \\ 0 \ 0 & \cdots & & 0 \ 1 & 0 \end{bmatrix}$$

여기서, $[T'_k]$는 $[T_k]$에서 (k,k)성분만 1로 다른 $k \times k$ 행렬.

제 4단계 : $k \leftarrow k+1$

제 5단계 : $k<s$ 이면 3단계로 간다.

제 6단계 : Stop

[정리 3.4][8] 주어진 어떤 k비트인 정보어 I에 대해서 $(I, T_K^2[I])$는 $(2k, k, 4)$부호를 생성한다.

<예 3.2> $T_4 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ **과 같을 때,** CA**의 상태전이 행렬** T_5 **를 구해보자.**

위의 <**알고리즘** 3.1>**에 의해** T_5 **을 구할 수 있다.**

$$T_5 = \begin{bmatrix} & [T_4'] & & & 0 \\ & & & & 0 \\ & & & & 0 \\ & & & & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

□

[정리 3.5][8] $(2k, k, 4)$**부호에서 검사비트** C_i**와** C_j**는** $C_i \oplus C_j$**에 의해서 다음 두 조건을 만족하는 주어진** T_k^2**으로 대체될 수 있다.**

(a) **행** i, j**는 같은 위치에서** 1**을 갖지 않는다.**

(b) **선형 벡터 합에 의해서 행** i, j**를 대체 함으로써** T_k^2**에서 얻은 축소된 행렬은** [**정리** 3.2]**의 조건을 만족한다.**

일반적으로 거리 $d \geq t + d_{et} + 1$ (t**와** d_{et}**는 정정되고 검출되는 오류의 최대 수를 각각 나타낸다**.)**인** (n, k, d)**부호에 대해서** LOGIC**블록은** [**정리** 3.4]**의 조건을 만족하는 논리회로를 실행한다.**

이것은 검사비트 수의 감소를 가져온다. 그 효과로 행렬 T_k^2**은** $(n - k)$ **검사비트를 생성하는 검사비트 생성 행렬** H'**로 수정된다.**

<예 3.3> $k = 8, d = 4$**일 때 부호를 생성해 보자**.

$k = 8$**이므로** 8-cell CA**가 요구된다**.

$$T_8 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad T_8^{\,2} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

다음 식에서 i_w**와** C_w**는** w**번째 정보비트와 검사비트를 나타낸다**.

$$\begin{aligned} C_7 &= i_7 \oplus i_6 \oplus i_5 \\ C_6 &= i_7 \oplus i_6 \oplus i_4 \\ C_5 &= i_7 \oplus i_5 \oplus i_3 \\ C_4 &= i_6 \oplus i_4 \oplus i_2 \\ C_3 &= i_5 \oplus i_3 \oplus i_1 \\ C_2 &= i_4 \oplus i_2 \oplus i_0 \\ C_1 &= i_3 \oplus i_1 \oplus i_0 \\ C_0 &= i_2 \oplus i_1 \oplus i_0 \end{aligned}$$

위의 등식의 선형 결합을 이용하면, [**정리** 3.1]**의 조건을 만족하는 등식의 수를** 7**로 줄이는 것이 가능하다**. C_{ij}**가** C_i**와** C_j**의 선형 결합에 의해서 얻어진 검사비트라 하면**, **축소된 등식은 다음과 같다**.

$$
\begin{aligned}
C_7 &= i_7 \oplus i_6 \oplus i_5 \\
C_6 &= i_7 \oplus i_6 \oplus i_4 \\
C_5 &= i_7 \oplus i_5 \oplus i_3 \\
C_{43} &= i_6 \oplus i_5 \oplus i_4 \oplus i_3 \oplus i_2 \oplus i_1 \\
C_2 &= i_4 \oplus i_2 \oplus i_0 \\
C_1 &= i_3 \oplus i_1 \oplus i_0 \\
C_0 &= i_2 \oplus i_1 \oplus i_0
\end{aligned}
$$

따라서 $(15, 8, 4)$**부호를 얻는다**. □

위 예에서 $(k \times k)$ T**행렬에서 시작하여** $(n - k) \times k$**행렬** H'**에 도달할 수 있다**.

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

행렬 H'**는 검사비트** CW**를 생성한다**. **즉** $CW = [H'][I]$**이다**. H'**는** [**정리** 3.5]**로** $T_8{}^2$**를 압축**(Compress)**함으로써 유도되며** <**예** 3.3>**에서 보였다**.

따라서 H'**는** LOGIC**블록에 의해서 실행되는** T^p**에서의 압축 함수의 출력이다**.

즉, $H' = Compress[T^p]$**이다**.

3.4. 최소 거리가 d인 부호의 생성

이제 임의의 값 d에 대해서 최소 거리가 d인 (n, k, d)부호를 생성하는 방법에 대해서 생각해 보자.

다음 알고리즘은 (n, k, d)부호를 생성하는 CA를 찾는 단계를 나타낸다.

<알고리즘 3.2> (n, k, d)부호의 생성

제 1단계 : k 비트 CA를 선택하고, seed로 k 정보비트를 적재시킨다.

제 2단계 : 주기가 p인 동안 CA를 작동한다. 여기서 $p = d - 2$

제 3단계 : T의 단일 멱 (T^p) 또는 T의 다중 멱 $(T^{p_1}, T^{p_2}, \cdots)$이 [정리 3.1]의 조건을 만족하는지 검사한다.

제 4단계 : [정리 3.1]의 조건을 만족하는 최소 거리가 d인 부호를 구하기 위해서 ;

(a) 확장인 경우 여분의 검사비트를 생성한다.

(b) 압축인 경우 검사비트의 수를 줄인다.

다음 예는 <알고리즘 3.2>에서 4단계 (a) 경우인 여분의 검사비트를 생성하는 것을 보여준다.

<예 3.4> 정보비트가 8일 때, 최소 거리가 5인 부호를 생성해 보자.

[정리 3.3]에 의해서 적어도 주기가 3인 CA가 요구된다. 즉, <알고리즘 3.2>의 2단계를 따른다. CA 법칙 < 90, 150, 90, 90, 90, 90, 150, 90 >은 조건을 만족한다. 특성행렬 T_8과 $T_8{}^3$은 아래와 같다.

$$T_8 = \begin{bmatrix} 0&1&0&0&0&0&0&0 \\ 1&1&1&0&0&0&0&0 \\ 0&1&0&1&0&0&0&0 \\ 0&0&1&0&1&0&0&0 \\ 0&0&0&1&0&1&0&0 \\ 0&0&0&0&1&0&1&0 \\ 0&0&0&0&0&1&1&1 \\ 0&0&0&0&0&0&1&0 \end{bmatrix} \qquad T_8{}^3 = \begin{bmatrix} 1&1&1&1&0&0&0&0 \\ 1&1&0&1&1&0&0&0 \\ 1&0&1&1&0&1&0&0 \\ 1&1&1&0&1&0&1&0 \\ 0&1&0&1&0&1&1&1 \\ 0&0&1&0&1&1&0&1 \\ 0&0&0&1&1&0&1&1 \\ 0&0&0&0&1&1&1&1 \end{bmatrix}$$

최소 거리가 5인 부호를 생성하기 위해서 여분의 검사비트를 추가해야한다. 즉, 행렬 $T_8{}^3$에 행을 덧붙여야 한다. 왜냐하면 $T_8{}^3$은 최소 거리가 4인 부호를 생성한다. 한 예로 정보비트 I가 (1 1 0 0 0 0 0 0) 이면, 부호어는 $(I,\ T_8{}^3[I])$ = (1 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0) 으로 거리가 4가 된다. 따라서, [정리 3.1]을 만족하도록 $T_8{}^3$에 행을 추가해야 한다. 여기서는 행 (1 0 0 0 0 0 0 1) 을 추가한다.

$$H' = \begin{bmatrix} 1&1&1&1&0&0&0&0 \\ 1&1&0&1&1&0&0&0 \\ 1&0&1&1&0&1&0&0 \\ 1&1&1&0&1&0&1&0 \\ 0&1&0&1&0&1&1&1 \\ 0&0&1&0&1&1&0&1 \\ 0&0&0&1&1&0&1&1 \\ 0&0&0&0&1&1&1&1 \\ 1&0&0&0&0&0&0&1 \end{bmatrix}_{9\times 8}$$

은 $(17, 8, 5)$부호를 생성한다.

□

여기서 CA 법칙을 < 150, 90, 90, 90, 90, 90, 90, 150 >을 이용해도 (17, 8, 5)부호를 생성 할 수 있지만 복호를 위해서는 역행렬이 존재해야만 한다. CA 법칙 < 150, 90, 90, 90, 90, 90, 90, 150 >으로 만든 행렬 $T_8{}^3$은 역행렬을 갖지 못하나 CA 법칙 < 90, 150, 90, 90, 90, 90, 150, 90 >으로 만든 행렬 $T_8{}^3$은 역행렬을 가진다.

이제 4단계 ⒝에 대해서 알아보자. 검사비트의 수를 줄이는 압축방법은 아래의 두 형태일 수 있다.

⒜ T의 단일 멱 (T^p)이 압축.

⒝ T의 다중 멱 (T^{p_1}, T^{p_2}, …)이 압축.

<예 3.5> 특성행렬의 다중 멱을 이용하여 정보비트가 8일 때, 최소거리가 5인 부호를 생성해 보자.

CA 법칙은 < 90, 150, 150, 90, 102, 90, 102, 90 >으로 택하고, 최소거리가 5인 부호를 생성하기 위해서 CA는 최소한 주기가 3만큼 실행해야 한다. 하지만, 검사어로 T의 다중 멱 ($T[I]$, $T^2[I]$)을 이용하면 최소 거리가 5인 부호를 구할 수 있다.

즉 8비트의 정보어는 16비트의 검사비트를 생성한다. 하지만, 압축방법을 쓰면 16비트는 9비트로 줄어든다.

특성행렬과 특성행렬의 제곱은 다음과 같다.

$$T_8 = \begin{bmatrix} 0&1&0&0&0&0&0&0 \\ 1&1&1&0&0&0&0&0 \\ 0&1&1&1&0&0&0&0 \\ 0&0&1&0&1&0&0&0 \\ 0&0&0&0&1&1&0&0 \\ 0&0&0&0&1&0&1&0 \\ 0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&1&0 \end{bmatrix} \qquad T_8^{\,2} = \begin{bmatrix} 1&1&0&1&0&0&0&0 \\ 1&0&0&0&1&0&0&0 \\ 0&0&0&1&0&1&0&0 \\ 1&0&1&1&1&1&1&0 \\ 0&0&0&0&1&0&0&1 \\ 0&0&0&0&0&1&1&1 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix}$$

행렬 T_8**에서** 1**행**$\oplus$5**행**, 2**행**$\oplus$8**행**, 3**행**$\oplus$7**행하고**, **행렬** $T_8^{\,2}$**에서** 1**행**$\oplus$5**행**$\oplus$7**행**, 2**행**$\oplus$3**행**, 4**행**$\oplus$8**행하면**,

$$T_8' = \begin{bmatrix} 0&1&0&0&1&1&0&0 \\ 1&1&1&0&0&0&1&0 \\ 0&1&1&1&0&0&1&1 \\ 0&0&1&0&1&0&0&0 \\ 0&0&0&0&1&0&1&0 \end{bmatrix} \qquad T_8^{\,2\,\prime} = \begin{bmatrix} 1&1&0&1&1&0&1&1 \\ 1&0&0&1&1&1&0&0 \\ 1&0&1&1&1&1&1&1 \\ 0&0&0&0&0&1&1&1 \end{bmatrix}$$

가 된다.

8**비트 정보어** I**에 대해서 검사비트는** 9**비트** $[\,T_8'[I],\ T_8^{\,2\,\prime}[I]\,]$**가 된다**. **따라서**, $(17, 8, 5)$ **부호를 생성한다**.

□

<**알고리즘** 3.2>**의** 4**단계** ⒜**에서 요구되는 검사비트를 찾는 것은 계산상으로 너무나 복잡하다**. **그래서**, 4**단계** ⒝**의 압축방법을 이용하는 것이 더 효율적이다**.

3.5. 복호의 기본 원리

오류가 있는 비트의 위치를 정정하기 위한 CAECC의 복호법에 대해서 알아본다.

정보비트와 검사비트가 정보원으로부터 목적지까지 전송되고 전송 중에 몇 개의 비트에 오류가 생긴다고 가정하자. 수신자는 수신된 정보어로부터 검사비트를 생성하고 수신된 검사비트와 XOR함으로써 오류 오증(Error Syndrome)을 계산한다. 오증 계산으로부터 오류가 있는 비트의 위치는 확인 될 수 있다.

다음 표기와 기본 결과는 복호법을 공식화하는데 이용된다.

I → 정보어(k비트) $i_0, i_1, \cdots, i_{k-1}$

CW → 검사어($n-k$비트) $CW_0, CW_1, \cdots CW_{n-k-1}$

I' → 수신된 정보어

CW' → 수신된 검사어

I_n → $n \times n$ 단위 행렬

T는 CA의 특성 행렬을 나타내고 seed로 I(정보어)가 적재된다. CA는 주기가 p인 동안 실행되며 CA의 상태는 CW(검사어)를 생성하기 위해서 압축 또는 확장된다. 즉 $[H'][I] = [CW]$.

$(2n, n, d)$부호에는 I와 CW의 비트의 수가 같으며, T^p는 H'와 동일하다. 하지만 CW의 비트 수가 I 보다 더 큰 부호를 생성하기 위해서는 T^p에 부가 행을 더한다. 반면에 CW의 비트 수가 I보다 적다면, T^p는 [정리 3.5]에 의해서 축소된다. 그래서 검사비트 생성 행렬 H'는 다음과 같이 나타낼 수 있다.

$$H' = \begin{cases} Compress[T^p] & \text{if } (n-k) < k \\ [T^p] & \text{if } (n-k) = k \\ \begin{bmatrix} T^p \\ H_0 \end{bmatrix} & \text{if } (n-k) > k \end{cases}$$

여기서, H_0는 확장된 검사비트에 대응되는 부분 행렬이다.

부호를 복호하는 첫 번째 단계는 $(n-k)$비트 오증 S를 계산하는 것이다. 오류가 i번째 검사비트에서 검출되었다면, 오증은 i번째 비트가 1인 벡터이다.

수신된 부호가 $w\ (=I', CW')$라면, 오증 $S(w)$는

$S(w) = [H][w]^T = [H]\begin{bmatrix} I' \\ CW' \end{bmatrix}$ 이다.

여기서, H는 H'와 단위행렬 I_{n-k}를 연결하여 형성된 $(n-k)\times n$ 패리티 검사 행렬(parity check matrix)이다.

즉, $H = [[H'][I_{n-k}]]$ 이다.

그래서,

$$S = [H]\begin{bmatrix} I' \\ CW' \end{bmatrix} = [[H'][I_{n-k}]]\begin{bmatrix} I' \\ CW' \end{bmatrix} = [H'][I'] \oplus [I_{n-k}][CW']$$

$$= [H'][I'] \oplus [CW']$$

로 표현될 수 있다.

S의 값이 0이 아니면, 수신어에 오류가 존재한다는 것을 나타낸다.

$I_e = \{e_1, e_2, \cdots e_k\}$과 $CW_e = \{e_{k+1}, e_{k+2}, \cdots e_n\}$를 수신어에서 정보비트와 검사비트에 대응되는 오류 벡터라고 하자. 그러면,

$I' = I \oplus I_e$

$CW' = CW \oplus CW_e$ 이다.

$$[H]\begin{bmatrix} I \\ CW \end{bmatrix} \oplus [H]\begin{bmatrix} I_e \\ CW_e \end{bmatrix} = [S]$$

여기서, $[H]\begin{bmatrix} I \\ CW \end{bmatrix} = 0$**이므로**,

$[H]\begin{bmatrix} I_e \\ CW_e \end{bmatrix} = [S]$**이 된다**. **즉**,

$$\begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,k} & 1 & 0 & \cdots & 0 \\ h_{2,1} & h_{2,2} & \cdots & h_{2,k} & 0 & 1 & \cdots & 0 \\ \vdots & & & & & & & \\ h_{n-k,1} & h_{n-k,2} & \cdots & h_{n-k,k} & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_k \\ e_{k+1} \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{n-k} \end{bmatrix}$$

에서 오증 비트는 다음 식에 의해서 주어진다.

$$s_1 = h_{1,1}e_1 \oplus h_{1,2}e_2 \oplus \cdots \oplus h_{1,k}e_k \oplus e_{k+1}$$

$$s_2 = h_{2,1}e_1 \oplus h_{2,2}e_2 \oplus \cdots \oplus h_{2,k}e_k \oplus e_{k+2}$$

$$\vdots$$

$$s_{n-k} = h_{n-k,1}e_1 \oplus h_{n-k,2}e_1 \oplus \cdots \oplus h_{n-k,k}e_k \oplus e_n$$

여기서 $h_{i,j} = [H]_{i,j}$.

$n > k$ **이므로**, **역사상**(reverse mapping)**은 유일하지 않을 수 있다**.

$$S => \begin{bmatrix} I_e \\ CW_e \end{bmatrix}$$

오류 검출과 정정에서 주된 목적은 주어진 오증 S**값에 대해서 오류 벡터** I_e**와** CW_e**를 결정하는 것이다**. **일단 오류벡터가 결정되면**, **오류가 있는 비트는 간단히** flip**될 수 있다**.

먼저 오증이 생성되면, 오증 식을 만족하는 특별한 오류벡터를 결정하기 위해서 연립방정식을 풀어야 한다.

미지수가 n개이고 $(n-k)$개의 식이 있으므로 유일한 해가 존재 할 수 없다.
복호법은 CA의 특성행렬의 역행렬의 특성을 이용한다.

수신어에서 n비트 오류형태를 E라고 하면, 오증 (S)와 다음 관계식으로 나타낼 수 있다.

$$[H][E] = [S] \tag{3.3}$$

여기서 H는 패리티검사행렬이다.

$$[E] = [H]^{-1}[S] \tag{3.4}$$

$[H]^{-1}$이 존재하려면 $[H]$는 $n \times n$정방행렬이어야 한다. 하지만, 여기서는 $[H]$가 정방행렬이 아니다. E에 대한 식을 풀기 위해서 다음 복호 방법을 제안한다.

3.4.1. 복호 방법

패리티 검사 행렬 $[H_{(n-k)\times n}]$를 정방행렬 $[T_{aug}]_{n\times n}$로 변환하기 위해서 k개의 추가 행을 $[H_{(n-k)\times n}]$에 덧붙인다. 이때 $[T_{aug}]_{n\times n}$가 group CA가 되도록 구성한다. 즉

$$det[T_{aug}] = 1$$

이다.

$$[T_{aug}] = \begin{bmatrix} [H]_{(n-k)\times n} \\ \cdots \\ [\text{추가행}]_{k\times n} \end{bmatrix}_{n\times n}$$

$$[T_{aug}][E] = \begin{bmatrix} [H] \\ \cdots \\ [\text{추가행}] \end{bmatrix}_{n \times n} \begin{bmatrix} I_e \\ \cdots \\ C_e \end{bmatrix}_{n \times 1} = \begin{bmatrix} S \\ \cdots \\ S_{aug} \end{bmatrix}_{n \times 1} \qquad (3.5)$$

T_{aug}가 결정되면, 가능한 오류벡터에 대해서 S_{aug}를 표로 작성할 수 있다. 표에서 S와 S_{aug}의 관계에 의해서 오류벡터를 구할 수 있다.

즉

$$[E] = [T_{aug}]^{-1} \begin{bmatrix} S \\ S_{aug} \end{bmatrix} \qquad (3.6)$$

<예 3.7> 4 비트 정보어에 대한 SEC-DED 부호의 복호 방법에 대해서 생각하자.

CA의 법칙은 < 90, 150, 150, 90 >이며, 검사비트 생성행렬은

$$H' = T^2 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

이다.

또한 패리티 검사행렬은

$$H = [[H'][I_4]] = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}_{4 \times 8}$$

이다.

이제 T_{aug}가 group CA가 되도록 H에 4개의 행을 덧붙인다.

가능한 T_{aug} 중 하나는 아래와 같다.

$$T_{aug} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}_{8 \times 8}$$

$[T_{aug}][E] = \begin{bmatrix} S \\ S_{aug} \end{bmatrix}$ 을 이용하여 다음 표를 작성할 수 있다.

가능한 오류벡터	오증 (Syndrome)	
	S	S_{aug}
00000000	0000	0000
10000000	1110	0001
01000000	1101	0011
00100000	1011	1011
00010000	0111	1000
00001000	1000	1011
00000100	0100	0000
00000010	0010	1100
00000001	0001	0011

4 비트 정보어가 $I = 1010$이라고 하자. 그러면, 검사어는

$CW = H'[I] = 0101$이다.

수신된 부호어가 $C' = 10000101$ ($I' = 1000$, $CW' = 0101$)이라면

먼저, 오증 S를 구한다.

$S = H'[I'] \oplus CW' = 1110 \oplus 0101 = 1011$

표에 의해서 오류벡터가 $E = 00100000$ 임을 알 수 있다. □

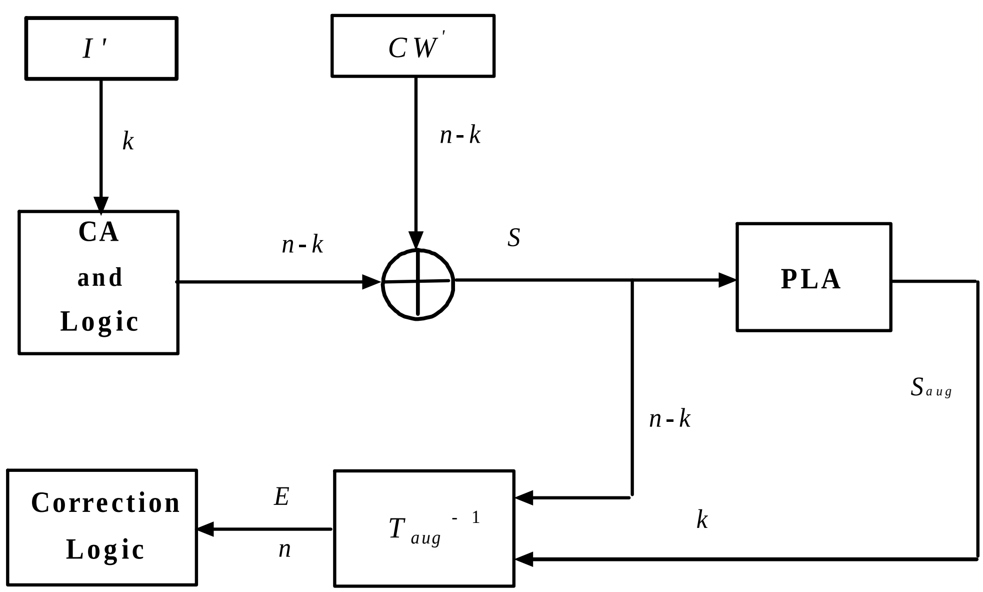
또한, 오류벡터는

$$[E] = [T_{aug}]^{-1} \begin{bmatrix} S \\ S_{aug} \end{bmatrix} \tag{3.7}$$

위의 식에 의해서 구해진다.

오증 S를 구하면 대응하는 S_{aug}는 표에 의해서 결정되지만, 복호 회로에서는 PLA(Programable Logic Array)소자를 이용하여 S에 대응하는 S_{aug}를 구한다.

<그림 3.3>는 복호 방법을 보여준다.



<그림 3.3 > 복호 방법

4. CA에 기초한 바이트 오류 정정 부호

4.1. CA-SbEC-DbED 부호의 생성

정보 블록의 모든 바이트는 각 바이트에 b비트를 포함한다고 하자(b-bit byte). 여기서 정보 바이트의 블록길이는 $N(\le 2^b-1)$으로 제한한다.

특성행렬이 T 이고 최대길이를 갖는 b셀 CA의 싸이클의 길이는 2^b-1이다.

검사 바이트는 CA를 N 회 동안 돌림으로써 생성되어지는데 이 때, 순차적으로 N 정보 바이트가 입력되어진다.

검사 바이트 C는 아래와 같이 주어진다.

$$C = T^{N-1}(B_0) \oplus T^{N-2}(B_1) \oplus \cdots \oplus T(B_{N-2}) \oplus T^0(B_{N-1}) \quad (4.1)$$

여기서 B_i 는 정보 바이트이다.

다중신호는 서로 다른 최대길이를 가지는 b셀 CA를 선택하여 계산된다.

CA-SbEC-DbED(CA based Single byte Error Correcting Double byte Error Detecting Code) 부호는 세 개의 검사 바이트를 가진다.

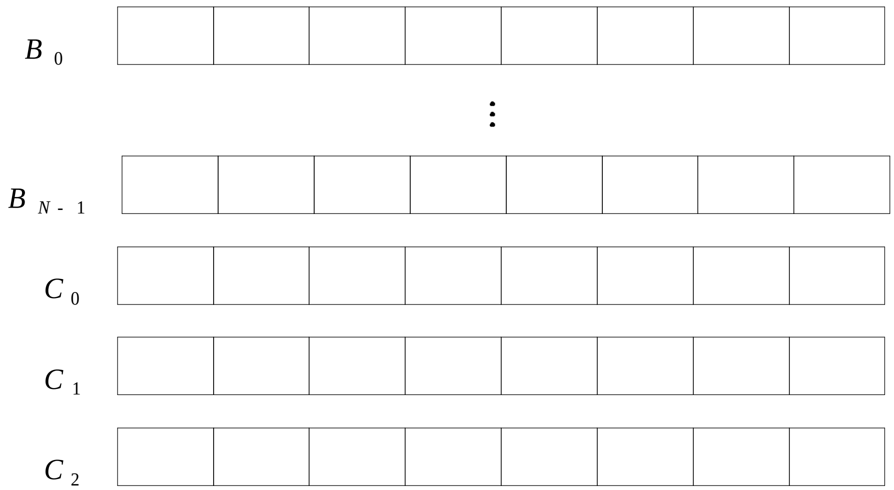
이것은 아래와 같이 생성되어진다.

$$C_0 = B_{N-1} \oplus B_{N-2} \oplus B_{N-3} \oplus \cdots \oplus B_0 \quad (4.2)$$

$$C_1 = B_{N-1} \oplus T[B_{N-2}] \oplus T^2[B_{N-3}] \oplus \cdots \oplus T^{N-1}[B_0] \quad (4.3)$$

$$C_2 = B_{N-1} \oplus T^2[B_{N-2}] \oplus T^4[B_{N-3}] \oplus \cdots \oplus T^{2(N-1)}[B_0] \quad (4.4)$$

정보 블록과 검사 바이트의 모양은 간단히 아래와 같다.



4.2. 복호 방법

복호의 주요 목적은 정보 바이트를 복원하는 것이다.
오류가 단지 검사 바이트에서만 발견되면 그것은 무시한다. 다음은 오류 오증과 오류 바이트의 정의이다.

[정의 4.1] 오류 오증 (error syndrome)

C_i를 발신한 정보 블록의 i번째 검사 바이트이고, C_i'은 수신된 정보 바이트로부터 계산된 i번째 검사 바이트라 하자. 그 때, i번째 검사 바이트에 오류 오증은 다음과 같이 정의된다.

$$S_i = C_i \oplus C_i' , \quad 0 \le i \le 2t \qquad (4.5)$$

[정의 4.2] 오류 바이트(error byte)

B_{N-1-j} 와 B'_{N-1-j} 는 각각 오류가 없는 j 번째 정보 바이트와, j 번째 수신된 정보 바이트이다. 왼쪽에서부터 세어서 B_j 번째는 오른쪽에서부터 세어서 B_{N-1-j} 번째와 같다.

이때, B_j 에 대한 오류 바이트는 다음과 같이 정의된다.

$$E_j = B_{N-1-j} \oplus B'_{N-1-j}, \quad j \geq 0 \tag{4.6}$$

모두 0인 오류 바이트는 대응하는 정보 바이트에 오류가 없다는 것을 의미하고, 0이 아닌 값은 오류에서 바이트 위치를 가리킨다.
오류 수정의 목적은 수신된 정보 바이트와 검사 바이트로부터 오류 벡터를 유도하는 것이다.

정확한 정보 바이트는 아래와 같이 얻어진다.

$$[B] = [B'] \oplus [E] \tag{4.7}$$

앞에서 서술했듯이 세 개의 검사 바이트는 SbEC-DbED 부호에서 필요하다.
바이트 블록을 받자마자 암호에서처럼 같은 논리를 사용함으로써 계산된다. 오류 오증 (S_0, S_1, S_2)은 수신된 검사 바이트를 가지고 새롭게 생성된 검사 바이트를 XOR 함으로써 계산한다.

오류가 왼쪽에서부터 i 번째와 j 번째에 오류가 발생했을 때 이것은 각각 E_i 와 E_j 에 대응한다고 가정하자.

수신된 쪽에서 생성된 검사 바이트는 아래와 같이 표현된다.

$$C_0' = B_{N-1} \oplus B_{N-2} \oplus B_{N-3} \oplus \cdots \oplus B_0 \oplus E_i \oplus E_j \quad (4.8)$$

$$C_1' = B_{N-1} \oplus T[B_{N-2}] \oplus T^2[B_{N-3}] \oplus \cdots \oplus T^{N-1}[B_0] \oplus T^i[E_i] \oplus T^j[E_j] \quad (4.9)$$

$$C_2' = B_{N-1} \oplus T^2[B_{N-2}] \oplus T^4[B_{N-3}] \oplus \cdots \oplus T^{2(N-1)}[B_0] \oplus T^{2i}[E_i] \oplus T^{2j}[E_j] \quad (4.10)$$

따라서, 오증은 아래와 같다.

$$S_0 = C_0 \oplus C_0' = E_i \oplus E_j \quad (4.11)$$

$$S_1 = C_1 \oplus C_1' = T^i[E_i] \oplus T^j[E_j] \quad (4.12)$$

$$S_2 = C_2 \oplus C_2' = T^{2i}[E_i] \oplus T^{2j}[E_j] \quad (4.13)$$

<예 4.1> 위에서 설명한 발신한 검사 바이트와 수신된 검사 바이트, 또한 오류 오증을 구해보자.

최대길이를 갖는 4셀 CA의 rule [90, 150, 90, 150] 일 때, 특성행렬 T는 다음과 같다.

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

정보 블록의 길이를 8로 하고 각각의 정보 바이트의 값을 다음과 같이 놓기로 하자.

$B_0 = 0001 \quad B_1 = 0010 \quad B_2 = 0011 \quad B_3 = 0100$

$B_4 = 0101 \quad B_5 = 0111 \quad B_6 = 1000 \quad B_7 = 1001$

여기서 정보 바이트 $B_3 = 0100$**와** $B_5 = 0111$ **을 수신자에게 보낼 때**, **수신자가 잘못된 오류 정보** $E_4 = 0001$**와** $E_2 = 0101$**을 받았다고 가정하자**.

그러면, **발신한 검사 바이트는 아래와 같이 구해진다**.

$$C_0 = B_7 \oplus B_6 \oplus B_5 \oplus B_4 \oplus B_3 \oplus B_2 \oplus B_1 \oplus B_0$$
$$= 1001 \oplus 1000 \oplus 0111 \oplus 0101 \oplus 0100 \oplus 0011 \oplus 0010 \oplus 0001$$
$$= 0111$$

$$C_1 = B_7 \oplus T[B_6] \oplus T^2[B_5] \oplus T^3[B_4] \oplus T^4[B_3] \oplus T^5[B_2] \oplus T^6[B_1] \oplus T^7[B_0]$$
$$= 1001 \oplus 0100 \oplus 0100 \oplus 0111 \oplus 1010 \oplus 1101 \oplus 0100 \oplus 1001$$
$$= 0100$$

$$C_2 = B_7 \oplus T^2[B_6] \oplus T^4[B_5] \oplus T^6[B_4] \oplus T^8[B_3] \oplus T^{10}[B_2] \oplus T^{12}[B_1] \oplus T^{14}[B_0]$$
$$= 1001 \oplus 1110 \oplus 1111 \oplus 1110 \oplus 1011 \oplus 1110 \oplus 0011 \oplus 1011$$
$$= 1010$$

따라서

$$C_0 = 0111 \qquad C_1 = 0100 \qquad C_2 = 1010$$

수신된 검사 바이트는 발신한 검사 바이트와 오류 바이트를 이용해 아래와 같이 구할 수 있다.

$$C_0' = C_0 \oplus E_4 \oplus E_2$$
$$= 0111 \oplus 0001 \oplus 0101 = 0011$$

$$C_1' = C_1 \oplus T^4[E_4] \oplus T^2[E_2]$$
$$= 0100 \oplus 0010 \oplus 1001 = 1111$$

$$C_2' = C_2 \oplus T^8[E_4] \oplus T^4[E_2]$$
$$= 1010 \oplus 0111 \oplus 1000 = 0101$$

따라서

$$C_0' = 0011 \quad C_1' = 1111 \quad C_2' = 1011$$

다음은 발신한 검사 바이트와 수신된 검사 바이트에 의해 얻어진 오류 오증이다.

$$S_0 = C_0 \oplus C_0' = 0111 \oplus 0011 = 0100$$
$$S_1 = C_1 \oplus C_1' = 0100 \oplus 1111 = 1011$$
$$S_2 = C_2 \oplus C_2' = 1010 \oplus 0101 = 1111$$

□

SbEC-DbED 부호에 대한 복호 알고리즘은 다음과 같다.

<알고리즘 4.1> SbEC-DbED 부호에 대한 복호

제 1단계 : 만약 모든 오증 바이트 S_0, S_1와 S_2가 모두 0이면 그 때 수신된 정보 블록에 오류가 없다.

제 2단계 : 오증 바이트 S_k $(k = 0, 1, 2)$ 중에 어떤 하나가 0이 아니고 다른 두 바이트가 0이라면 그 때 단지 검사 바이트 C_k 에 오류가 있다.

제 3단계 : 만약 오증 바이트에 하나 보다 더 많은 바이트가 0이 아니면 다음과 같은 식을 만족하는 i를 찾아라.

$T^i[S_0] = S_1$ and $T^{2i}[S_0] = S_2$

만약 이러한 i가 존재한다면 하나의 오류가 N-1-i 번째 바이트에 발생했고, 오류 바이트는 S_0이다.

만약 이러한 오류를 찾을 수 없다면 그것은 오류가 두 개 또는 그 이상이어서 수정할 수 없는 오류가 존재한다.

<예 4.2> 정보 바이트 $B_5 = 0111$ 가 잘못 수신되어 오류가 발생했을 때 발신한 검사 바이트, 수신된 검사 바이트, 오증을 구하고 오류의 위치를 찾아보자.

정보 바이트 $B_5 = 0111$ 가 잘못 수신되었을 때 수신된 정보 바이트

를 $B_5' = 0010$ 라고 하자.

그러면 발신한 검사 바이트는 <예 4.1>에서 사용한 정보 바이트로부터 아래와 같이 구해진다.

$$C_0 = 0111 \quad C_1 = 0100 \quad C_2 = 1010$$

수신된 정보 바이트로부터 다음과 같은 검사 바이트가 얻어진다.

$$C_0' = B_7' \oplus B_6' \oplus B_5' \oplus B_4' \oplus B_3' \oplus B_2' \oplus B_1' \oplus B_0'$$
$$= 1001 \oplus 1000 \oplus 0010 \oplus 0101 \oplus 0100 \oplus 0011 \oplus 0010 \oplus 0001 = 0010$$
$$C_1' = B_7' \oplus T[B_6'] \oplus T^2[B_5'] \oplus T^3[B_4'] \oplus T^4[B_3'] \oplus T^5[B_2'] \oplus T^6[B_1'] \oplus T^7[B_0']$$
$$= 1001 \oplus 0100 \oplus 1101 \oplus 0111 \oplus 1010 \oplus 1101 \oplus 0100 \oplus 1001 = 1101$$
$$C_2' = B_7' \oplus T^2[B_6'] \oplus T^4[B_5'] \oplus T^6[B_4'] \oplus T^8[B_3'] \oplus T^{10}[B_2'] \oplus T^{12}[B_1'] \oplus T^{14}[B_0']$$
$$= 1001 \oplus 1110 \oplus 0111 \oplus 1110 \oplus 1011 \oplus 1110 \oplus 0011 \oplus 1011 = 0010$$

다음은 발신한 검사 바이트와 수신된 검사 바이트에 의해 얻어진 오류 오증이다.

$$S_0 = C_0 \oplus C_0' = 0111 \oplus 0010 = 0101$$
$$S_1 = C_1 \oplus C_1' = 0100 \oplus 1101 = 1001$$
$$S_2 = C_2 \oplus C_2' = 1010 \oplus 0010 = 1000$$

□

여기서 $T^i[S_0] = S_1$ and $T^{2i}[S_0] = S_2$ 를 만족하는 i를 찾는다.

$T^i(0101) = 1001$ and $T^{2i}(0101) = 1000$을 만족하는 $i = 2$ 가 됨을 알 수 있다. 이것은 2장에서 최대길이를 갖는 4셀 group CA에서 i번째의 위치를 쉽게 찾을 수 있다.

따라서 $B_{N-1-i} = B_5$ 이므로 오른쪽에서 다섯 번째 정보 바이트에 오류가 발생했다는 것을 알 수 있다.

또한 정확한 정보 바이트도 아래와 같이 구할 수 있다.

$$B_5 = B_5' \oplus S_0$$
$$= 0010 \oplus 0101 = 0111$$

□

[정리 4.1.] <알고리즘 4.1>은 정확히 CA-SbEC-DbED 부호를 복호한다.

(증명) Case 1. 한 개의 바이트에 오류가 발생했을 때

오류가 검사 바이트 C_i에 발생했다면 그 때는 단하나 오증 바이트 S_i가 0이 아니고, 나머지 오증 바이트는 0이다. 이것은 오증 바이트 생성 방법으로부터 명백함을 알 수 있다.

만약 오류가 정보 부분에서 일어난다면, 그때

$$S_0 = E_i \tag{4.14}$$

$$S_1 = T^i[E_i] \tag{4.15}$$

$$S_2 = T^{2i}[E_i] \qquad (4.16)$$

따라서 이 알고리즘은 정보 바이트와 검사 바이트에 한 개 바이트 오류를 정확히 구별한다.

Case 2. 두 개의 바이트에 오류가 발생했을 때

두 개의 바이트에 오류가 발생했을 때는 다음과 같은 세 가지 경우로 나누어진다.
(a) **두 개 오류가 모두 검사 바이트에 있다. 이 경우에는 오류가 있는 검사 바이트에 대응하는 두 개의 오증 바이트가** 0**이 아니다.**

(b) **정보 부분에 한 개의 오류가 있고 나머지 하나는 검사 부분에 있다. 이 경우에는 두 개 또는 세 개 모두 오증 바이트가** 0**이 아니다.**

(c) **두 개 오류가 모두 정보 바이트에 있다. 이 경우에는 세 개의 오증 바이트가** 0 **이 아니다.**

여기서 (b) **와** (c) **의 경우는 한 개의 오류 경우에서처럼 같은 오증의 집합으로 대응하지 않는다는 것을 보여야 한다.**
오류가 N-1-i **번째와** N-1-j **번째에 있다고 가정하자. 이에 대응하는 오류 바이트는** E_i **와** E_j **라고 하자.**
N-1-i **번째와** N-1-j **번째의 오증의 합이** N-1-k **에 대응하는** E_k **라고 하자.**
그 때 오증은 아래와 같은 방정식을 만족시킨다.

$$S_0 = E_i \oplus E_j = E_k \qquad (4.17)$$

$$S_1 = T^i[E_i] \oplus T^j[E_j] = T^k[E_k] \tag{4.18}$$

$$S_2 = T^{2i}[E_i] \oplus T^{2j}[E_j] = T^{2k}[E_k] \tag{4.19}$$

(4.17)**와** (4.18) **의 식으로부터**

$$T^i[E_i] \oplus T^j[E_j] = T^k[E_i \oplus E_j] \tag{4.20}$$

또는,

$$(T^i \oplus T^k)[E_i] = (T^j \oplus T^k)[E_j] = E \tag{4.21}$$

(4.19)**로부터**

$$T^{2k}[E_k] = T^k(T^i[E_i] \oplus T^j[E_j])$$

$$= T^{2i}[E_i] \oplus T^j[E_j] \tag{4.22}$$

$$(T^{2i} \oplus T^{i+k})[E_i] = (T^{2j} \oplus T^{j+k})[E_j] \tag{4.23}$$

또는,

$$T^i(T^i \oplus T^k)[E_i] = T^j(T^j \oplus T^k)[E_j] \tag{4.24}$$

따라서 (4.21)**의 식으로부터**

$$T^i[E] = T^j[E] \tag{4.25}$$

위의 식은 단지 $i = j$ **일 때만 만족된다. 그러나 이것은 오류의 위치가 다르다는 것에 모순이 일어난다.**

따라서, 정보 부분에 대한 두 개의 오류에 대한 오류 오증은 한 개의 오류에 대한 것처럼 결코 같아 질 수 없다. □

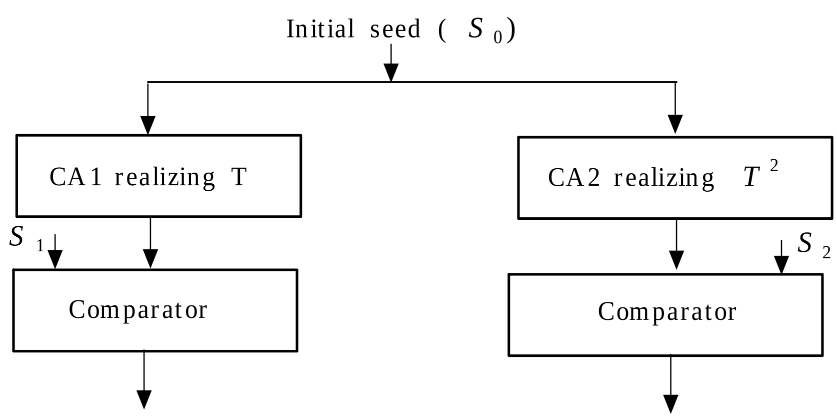
4.2.1 구현

복호 알고리즘의 제 3단계는 아래와 같이 구현한다.

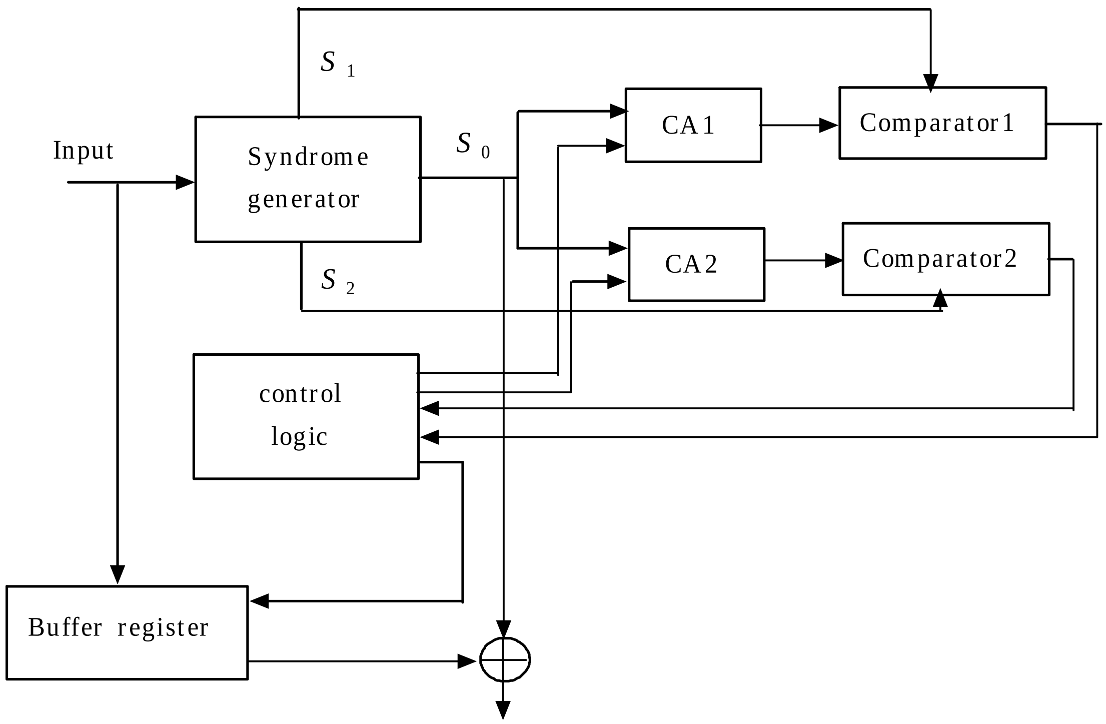
<알고리즘 4.2>

제 1단계 : CA에 오증 S_0를 로드하라.
제 2단계 : i=1 에서부터 N 까지 i 와 2 i 회 동안 돌려라. CA의 마지막 상태는 각각 R_1 과 R_2 에서처럼 구성된다. 만약 $S_1 = R_1$ 와 $S_2 = R_2$ 라면 그 때 오류의 위치는 N-1- i 번째이고 오류 벡터는 S_0 이다.
제 3단계 : 위와 같은 상태가 존재하지 않는다면 그 때는 수정할 수 없는 오류가 발생한다.

아래는 SbEC-DbED의 복호 방법이다



<그림4.1> SbEC-DbED의복호



<그림 4.2> 복호 방법 전체

4.3. CA-DbEL / DbEC 부호의 생성

이 절에서는 두 바이트 오류의 위치를 찾아내는 방법에 대해 소개할 것이다. 종종 메시지 전송 과정에서 앞으로 진행 과정동안 사용되지 않을 수 있는 어떤 바이트에서 오류가 발생할 수 있다. 이러한 경우 오류의 위치를 찾는 것은 오류의 수정보다 더 적절하다.
따라서 이러한 과정에서는 정정을 위해서 필요한 여분의 주기가 생략될 수도 있다.
여기서는 CA-DbEL / DbEC (CA based Double byte Error Locating

/ Double byte Error Correcting)의 생성 방법에 대해 보일 것이다. CA-DbEL / DbEC 부호는 다음 상황을 다룰 수 있도록 설계한다.

1. 만약 두 바이트 오류가 단지 정보 바이트 또는 검사 바이트에만 존재한다면 그 때 오류를 검출하고 수정할 수 있다.

2. 만약 정보 바이트에 한 개의 오류가 있고 검사 바이트에 다른 한 개의 오류가 있다면 최대 세 개의 정보 바이트 안에 오류가 위치한다.
만약 부호가 DbEL(Double-byte Error Locating)이라면 네 개의 검사 바이트가 아래와 같이 생성된다.

$$C_0 = B_{N-1} \oplus B_{N-2} \oplus B_{N-3} \oplus \cdots \oplus B_0 \tag{4.26}$$

$$C_1 = B_{N-1} \oplus T[B_{N-2}] \oplus T^2[B_{N-3}] \oplus \cdots \oplus T^{N-1}[B_0] \tag{4.27}$$

$$C_2 = B_{N-1} \oplus T^2[B_{N-2}] \oplus T^4[B_{N-3}] \oplus \cdots \oplus T^{2(N-1)}[B_0] \tag{4.28}$$

$$C_3 = B_{N-1} \oplus T^3[B_{N-2}] \oplus T^6[B_{N-3}] \oplus \cdots \oplus T^{3(N-1)}[B_0] \tag{4.29}$$

4.4 복호 방법

아래 알고리즘은 두 바이트 오류가 발생했을 때 오류의 위치와 올바른 오류패턴을 구하는 것이다.

<알고리즘 4.3> DbEL / DbEC 부호의 복호

제 1단계 : 만약 오증 바이트 S_0, S_1, S_2 와 S_3가 모두 0이면, 그 때 수신된 정보 블록에는 오류가 없다.

제 2단계 : 모든 오증 바이트가 0이 아니고 다음과 같은 i 와 j 가 존재하는지 확인한다.

$$T^{i}(S_{2}) \oplus S_{3} = T^{2j}[T^{i}(S_{0}) \oplus S_{1}]$$

위의 식에서 i 와 j를 찾을 수 없다면 아래의 식에서 확인하라.

$$T^{j}[T^{i}(S_{0}) \oplus S_{1}] = T^{i}(S_{1}) \oplus S_{2}$$

$$T^{j}[T^{2i}(S_{0}) \oplus S_{2}] = T^{2i}(S_{1}) \oplus S_{3}$$

위의 식으로부터 i 와 j 번째에 오류가 있다는 것을 확인할 수 있다.
만약 오류의 위치만 알고 싶다면 제 6단계로 가라.

제 3단계 : $T^{i+j} + T^{i+k} + T^{j+k} = 0$ 또는 $T^{i} + T^{j} + T^{k} = 0$ 를 만족하는 k를 찾으면 제 5단계로 가라.

제 4단계 : 다음 식을 사용해서 j 번째 바이트에 오류 패턴 E_{j} 를 계산해라.

$T^{x} = T^{i} \oplus T^{j}$ 을 만족하는 x 를 찾아 아래 식에 대입한다.

$$E_{j} = T^{-x}[T^{i}(S_{0}) \oplus S_{1}] \quad \text{또는}$$

$$E_{j} = T^{N-x}[T^{i}(S_{0}) \oplus S_{1}]$$

다음 식으로부터 오류 패턴 E_{i} 를 계산한

후 제 6단계로 가라.

$E_i = S_0 \oplus E_j$

제 5단계 : 두 바이트 오류가 i 와 j 번째 정보 바이트에 있거나, k 번째 정보 바이트에 하나가 있고 검사 바이트에 하나가 있다.

제 6단계 : Stop.

[정리 4.2] <알고리즘 4.3>은 정확히 DbEL / DbEC 부호를 복호한다.

(증명) 정보 바이트 두 개에 오류가 발생했다고 가정하자.
이것은 네 개의 오증 바이트가 0 이 되지 않을 수도 있다.
다른 두 쌍의 바이트에서 발생한 오류는 같은 오증으로 대응하지 않는 것을 보여야 한다.
N-1-i 번째 바이트의 오류를 E_i 라 하고, N-1-j 번째와 N-1-k 번째, N-1-r 번째 바이트의 오류를 각각 E_j, E_k, E_r 이라고 하자.
먼저 두 쌍 E_i, E_j 에 대한 오증은 다른 두 쌍 E_k, E_r 의 오증과 같다고 가정하자.

그러면 다음과 같은 오증 방정식을 구할 수 있다.

$$S_0 = C_0 \oplus C_0' = E_i \oplus E_j = E_k \oplus E_r \quad (4.33)$$

$$S_1 = C_1 \oplus C_1' = T^i[E_i] \oplus T^j[E_j] = T^k[E_k] \oplus T^r[E_r] \quad (4.34)$$

$$S_2 = C_2 \oplus C_2' = T^{2i}[E_i] \oplus T^{2j}[E_j] = T^{2k}[E_k] \oplus T^{2r}[E_r] \quad (4.35)$$

$$S_3 = C_3 \oplus C_3' = T^{3i}[E_i] \oplus T^{3j}[E_j] = T^{3k}[E_k] \oplus T^{3r}[E_r] \quad (4.36)$$

방정식 (4.33)**에** T^j **를 곱하고** (4.34)**에 더하면 아래와 같은 식을 얻는다.**

$$[T^i \oplus T^j][E_i] = [T^j \oplus T^k][E_k] \oplus [T^j \oplus T^r][E_r] \quad (4.37)$$

또는

$$E_1 = E_2 \oplus E_3 \quad (4.38)$$

방정식 (4.34)**에** T^j **를 곱하고** (4.35)**에 더하면 아래와 같은 식을 얻는다.**

$$T^i[T^i \oplus T^j][E_i]$$

$$= T^k[T^j \oplus T^k][E_k] \oplus T^r[T^j \oplus T^r][E_r] \quad (4.39)$$

$$T^i[E_1] = T^k[E_2] \oplus T^r[E_3] \quad (4.40)$$

방정식 (4.35)**에** T^j **를 곱하고** (4.36)**에 더하면 아래와 같은 식을 얻는다.**

$$T^{2i}[E_1] = T^{2k}[E_2] \oplus T^{2r}[E_3] \quad (4.41)$$

방정식 (4.38)**에** T^k **를 곱하고** (4.39)**에 더하면 아래와 같은 식을 얻는다.**

$$[T^i \oplus T^k][E_1] = [T^k \oplus T^r][E_3] = E_4 \quad (4.42)$$

방정식 (4.40)**에** T^k **를 곱하고** (4.41)**에 더하면 아래와 같은 식을 얻는다.**

$$T^i[T^i \oplus T^k][E_1] = T^r[T^k \oplus T^r][E_3] \quad (4.43)$$

또는

$$T^{i}[E_4] = T^{r}[E_4] \tag{4.44}$$

따라서

$$i = r \quad j = k$$

따라서 두 쌍의 다른 오류 바이트는 같은 오증 바이트가 될 수 없다.

다음은 오류 바이트는 유일하게 계산된다는 것을 보일 것이다.
N-1-i 번째와 N-1-j 번째 정보 바이트에 오류가 있다고 하자.
그때, 다음과 같은 오증 방정식이 구해진다.

$$S_0 = E_i \oplus E_j \tag{4.45}$$

$$S_1 = T^{i}[E_i] \oplus T^{j}[E_j] \tag{4.46}$$

$$S_2 = T^{2i}[E_i] \oplus T^{2j}[E_j] \tag{4.47}$$

$$S_3 = T^{3i}[E_i] \oplus T^{3j}[E_j] \tag{4.48}$$

위의 방정식으로부터 아래와 같은 식을 얻을 수 있다.

$$T^{i}(S_0) \oplus S_1 = [T^{i} \oplus T^{j}][E_j] \tag{4.49}$$

$$T^{2i}(S_0) \oplus S_2 = [T^{2i} \oplus T^{2j}][E_j] \tag{4.50}$$

또는

$$T^{i}(S_1) \oplus S_2 = T^{j}[T^{i} \oplus T^{j}][E_j] \tag{4.51}$$

$$= T^{j}[T^{i}(S_0) \oplus S_1] \tag{4.52}$$

$$T^{2i}(S_1) \oplus S_3 = T^{j}[T^{2i} \oplus T^{2j}][E_j] \tag{4.53}$$

$$= T^{j}[T^{2i}(S_0) \oplus S_2] \tag{4.54}$$

방정식 (4.49)로부터 오류 바이트 E_j 는 아래와 같이 구해진다.

$$T^{i}(S_0) \oplus S_1 = T^{x}[E_j] \tag{4.55}$$

또는

$$E_j = T^{-x}[T^{i}(S_0) \oplus S_1] \tag{4.56}$$

또는

$$E_j = T^{N-x}[T^{i}(S_0) \oplus S_1] \tag{4.57}$$

이제 E_j 의 값을 이용해 다른 하나의 오류 바이트 E_i 를 구할 수 있다.

$$E_i = S_0 \oplus E_j \tag{4.58}$$

<예 4.3> <예 4.1>에서의 정보 바이트 $B_3 = 0100$ 과 $B_5 = 0111$ 이 잘못 수신되어 오류가 발생했을 때 오류의 위치와 올바른 오류패턴을 구해보자.

정보 바이트 $B_3 = 0100$ 과 $B_5 = 0111$이 잘못 수신 되었을 때의 수신된 정보 바이트를 $B_3' = 0101$ 과 $B_5' = 0010$ 이라고 하자.

오류의 위치와 올바른 오류패턴은 <알고리즘 4.3>의 단계를 거치면서 얻어낼 수 있다.

<예 4.1>에서 발신한 정보 바이트를 이용하여 발신한 검사 바이트와 수신된 검사 바이트를 구하고 또한 그에 따른 오증 바이트도 구할 수 있다. 여기서는 네 개의 오증 바이트가 필요하다. 그것을 구하면 다음

과 같다.

$S_0 = 0100 \quad S_1 = 1011 \quad S_2 = 1111 \quad S_3 = 0001$

제 2단계 : $T^i(S_2) \oplus S_3 = T^{2j}[T^i(S_0) \oplus S_1]$ 을 만족하는 i 와 j 를 찾는다.

$$T^i(1111) \oplus 0001 = T^{2j}[T^i(0100) \oplus 1011]$$

최대길이를 갖는 4셀 group CA에서 $i = 4$ $j = 2$ 임을 찾을 수 있다. 따라서 세 번째와 다섯 번째 정보 바이트에서 오류가 났음을 알 수 있다.

제 3단계 : $T^i + T^j + T^k = 0$ 이 되는 k 를 구한다.

$$T^4 + T^2 + T^k = 0$$

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} + T^k = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

이므로

$$T^k = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$T^k = T^{10}$$

이므로

$$k = 10$$

여기서 우리는 정보 블록의 길이를 8로 했기 때문에 k 의 크기가 8을 넘으므로 k 값은 존재하지 않고 검사 바이트에 오류는 없다.
따라서 두 바이트 오류가 세 번째와 다섯 번째 정보 바이트에 있다는 것을 찾을 수 있다.
정보 바이트의 위치를 알았음으로 i 와 j 번째의 오류 패턴을 구하자.
제 4 단계 : $T^x = T^i \oplus T^j$ 을 만족하는 x 를 구한다.

$$T^x = T^4 \oplus T^2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} = T^{10}$$

따라서

$$x = 10$$

$E_j = T^{-x}[T^i(S_0) \oplus S_1]$ **식으로부터 오류 패턴** E_j **를 구한다.**

$$E_j = T^{-10}[T^4(0100) \oplus 1011]$$

$$= \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} [T^4(0100) \oplus 1011] = 0101$$

따라서

$$E_j = E_2 = 0101$$

다음 식으로부터 오류 패턴 E_i **를 구한다.**

$$E_i = S_0 \oplus E_j$$

$$= 0100 \oplus 0101 = 0001$$

따라서

$$E_i = E_4 = 0001$$

따라서 오류가 난 부분의 발신한 정보 바이트의 정확한 값을 구할 수 있다. □

4.4.1 구현

두개의 바이트 오류의 위치를 알기 위해서 제 2단계는 아래와 같이 구현되어진다.

1. CA1, CA2 그리고 CA3 각각 특성행렬 T, T 그리고 T^2 를 가지는 CA 라 하자.
   CA1과 CA2에 각각 오증 S_0 와 S_1을 로드하라.

2. $T^i(S_0) \oplus S_1$ 이 CA3 에 로드되고 j 주기 동안 작동되어진다.
   여기서 $0 \le j \le N-1$ 이다.
   만약 $T^{2j}[T^i(S_0) \oplus S_1] \oplus T^i(S_2) \oplus S_3 = 0$ 이면 그때 i 번째와 j 번째 정보 바이트에 오류가 있다. (그림 4.3)

3. 두 개의 바이트 오류를 정정하기 위해서 <알고리즘 4.3>의 제 4단계는 (그림 4.4)에서처럼 구현되어진다.

모두 0 이 아닌 어떤 b 비트 seed(X)는 $(T^i \oplus T^j)(X) = T^x(X)$ 를 계산하기 위해서 CA1 과 CA2에 보내진다.

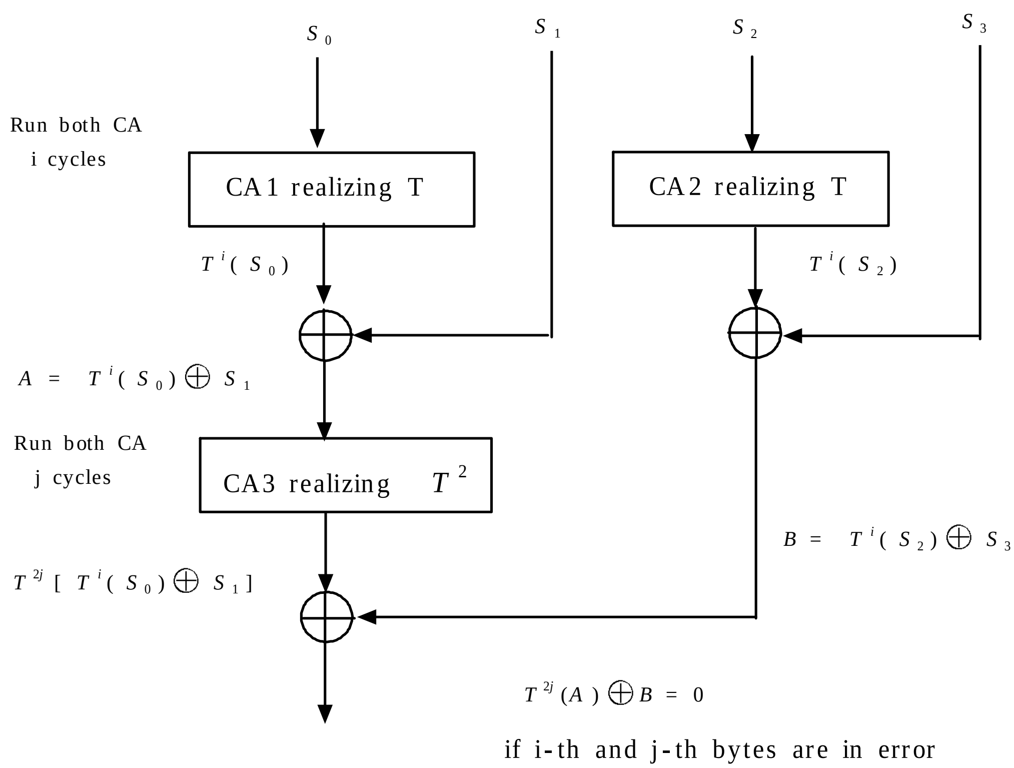
$T^N = I$ 이기 때문에

$$X = T^N(X)$$
$$= T^{N-x}[T^x(X)]$$ 이다.

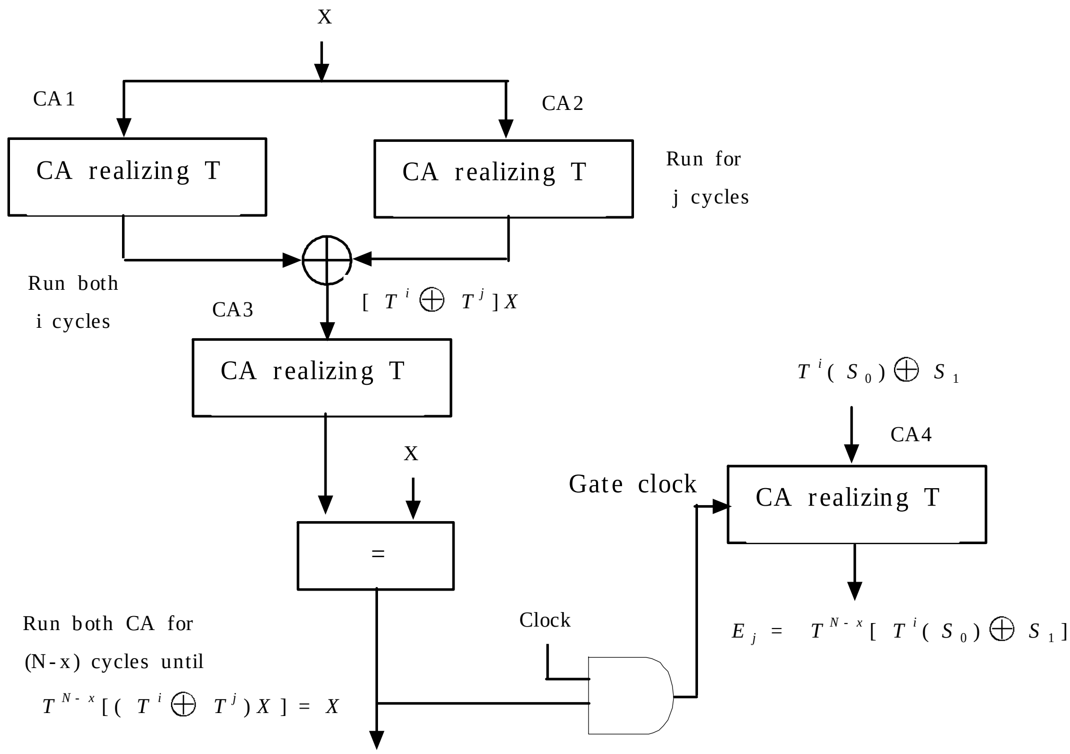
CA3 은 초기 seed로 $(T^i \oplus T^j)(X)$ 을 가지고 작동되어진다.
이런 CA 의 출력은 CA4 의 클릭을 조절한다.
그래서 CA4 는 주어진 $E_j = T^{N-x}[T^i(S_0) \oplus S_1]$ 에서 정확히 $N-x$회 동안 작동한다.

그리고 E_i 는 다음 식으로부터 구해진다.

$$E_i = S_0 \oplus E_j$$



<그림 4.3> DbEL 부호의 기본 셀 구조


X
CA1
CA2
CA realizing T
CA realizing T
Run for
j cycles
Run both
i cycles
CA3
[$T^i \oplus T^j$] X
CA realizing T
$T^i(S_0) \oplus S_1$
CA4
X
Gate clock
CA realizing T
=
Clock
$E_j = T^{N-x}[T^i(S_0) \oplus S_1]$
Run both CA for
(N-x) cycles until
$T^{N-x}[(T^i \oplus T^j)X] = X$

<그림4.4> 수정 방법

5. 결론

본 논문에서는 최근 활발한 연구가 진행중인 Cellular Automata(CA)와 오류 정정 부호를 소개하였다. CA의 정규적인 구조의 특성을 이용하여 비트(Bit)와 바이트(Byte) 오류 정정 부호를 효율적으로 설계하는 방법을 제시하였다.

CA에 기초한 비트 오류 정정 부호에서는 CA의 특성행렬을 이용하여 k-셀 CA블록을 p주기만큼 실행함으로써 검사비트를 생성하였다. 이때 원하는 부호를 생성하기 위한 압축/확장 방법과 k를 증가시켰을 때 부호를 생성하는 CA를 구성하는 방법도 알아보았다. 또한 특성행렬의 역행렬과 PLA(Programable Logic Array)소자를 이용하여 복호하는 방법도 제시하였다.

CA에 기초한 바이트 오류 정정 부호에서는 정보 바이트와 특성행렬을 이용하여 검사 바이트를 생성하였고, 오류 오증 알고리즘을 이용하여 오류의 위치를 찾고 오류를 정정할 수 있었다.

CA에 기초한 오류 정정 부호의 부호기와 복호기는 VLSI기술에 효율적으로 구현될 수 있다는 장점이 있다.

참 고 문 헌

[1] P.P. Chaudhuri, et. al., "*Additive cellular automata theory and applications*", Vol. 1, IEEE Computer Society Press, California, USA, 1997.

[2] D.R. Chowdhury, S. Basu, I.S. Gupta, P.P. Chaudhuri, "CA Based Byte Error-Correcting Code", IEEE Trans. Computers, Vol. 44, Oct. 1995.

[3] D.R. Chowdhury, S. Basu, I.S. Gupta, P.P. Chaudhuri, "Design of CAECC - Cellular Automata Based Error Correcting Code", IEEE Trans. Computers, Vol. 43, June 1994, pp. 759-764.

[4] S.J. Cho, U.S. Choi and H.D. Kim, Analysis of Complemented CA Derived from a Linear TPMACA(To Appear in Computers and Mathematics with Applications).

[5] S.J. Cho, U.S. Choi and H.D. Kim, Behavior of Complemented CA which the Complement Vector is Acyclic in a Linear TPMACA(Submitted).

[6] S.J. Cho, U.S. Choi and H.D. Kim, "*Linear nongroup one-dimensional cellular automata characterization on GF(2)*", J. Korea Multimedia Soc., Vol. 4, No. 1, 2001, pp. 91-95.

[7] A.K. Das and P.P. Chaudhuri, "*Efficient characterization of cellular automata*" , Proc. IEE(part E), Vol. 137, No. 1, 1990, pp. 81-87.

[8] A.K. Das and P.P. Chaudhuri, "Vector space theoretic analysis of additive cellular automata and its application for pseudo-exhaustive test pattern generation", *IEEE Trans. Compute.*, Vol. 42, pp. 340-352, 1993.

[9] T.R.N. Rao, E.Fujiwara, "Error-Control Coding for Computer System", Prentice Hall, Englewood Cliffs, N.J., 1989.

[10] J. Von Neumann, *Theory of self-reproducing automata*, University of Illinois Press Urbana, 1966.

[11] S. Wolfram, "*Cellular Automata and Complexity*", Addison-Welsly Publishing Company, 1994.

[12] S. Wolfram, "Statistical mechanics of cellular automata", *Rev. Modern Physics*, Vol. 55, No. 3, July 1983.

[13] **박승안** "**대수학과 암호학**", **경문사**, 2000.