論文 86-23-3-2

Bit Code割當에 依한 GF(2^m)上의 多値論理函數 構成 理論

(A Construction Theory of Multiple-Valued Logic Functions on GF(2^m) by Bit Code Assignment)

金 興 壽*, 朴 春 明*

(Heung Soo Kim and Chun Myoung Park)

要 約

本 論文에서는 多値論理函數를 構成하기 위하여 Galois體上에서 2値化가 容易한 素數가 2인 경우의 모든 元素들을 bit code로 割當하는 알고리즘을 提示하고, 이 알고리즘으로 구한 GF(2^m)上 元素들간의 bit code 演算(加算, 乘算)을 멀티플렉서(multiplexer)를 사용하여 이행한 후 이를 토대로 加算器와 乘算器를 構成하였다.

또한 一般的인 多値論理函數 構成에는 狀態遷移圖를 構成하여 狀態를 最小化시킨후, 單一 및 多入出力에 대한 回路實現을 VLSI設計에 널리 사용되고 있는 PLA로 實現하였다.

Abstract

This paper presents a method of constructing multiple-valued logic functions based on Galois field. The proposed algorithm assigns all elements in GF(2^m) to bit codes that are easily converted binary. We have constructed an adder and a multiplier using a multiplexer after bit code operation (addition, multiplication) that is performed among elements on GF(2^m) obtained from the algorithm. In constructing a generalized multiple-valued logic functions, states are first minimized with a state-transition diagram, and then the circuits using PLA widely used in VLSI design for single and multiple input-output are realized.

I. 序 論

多値論理를 2値論理化하여 現在 사용되고 있는 디지탈시스템 및 스위칭理論에 適用하는 方法은 지금까지 많은 論文을 통해 發表 되어 왔다.[1-3] 특히 P가 素數이고 m이 陽의 整數인 GF(P^m) 上에서 m의 擴張에 따른 多値를 2値化하는 경우는 GF(2^m)∈Z_2={0, 1}이므로 素數P가 2인 경우에 2値化가 容易하다. 이때 GF(2^m) 上에는 2^m개의 元素가 存在하고 이들 元素들을 2値化하는데는 m개의 bit code가 必要하다.[4-8]

Error-correcting codes를 비롯하여 digital signal processing 및 image processing 등의 分野에서도 GF(2^m) 上의 元素들을 2値化함으로써 解釋을 容易하게 展開 시킬 수 가 있다.[6][8-11] [15-17]

이와같은 GF(2^m) 上의 모든 元素들을 2値로 변환하는 方法으로써, Boonsieng Benjauthrit와 I.S.Reed[9]는 GF(2^2)上의 既約多項式으로 부터 係數를 구하여 元素를 2値로 割當하였고, K. S. Menger[5] 등은 GF(2^3)上의 元素들을 2値化 하였는데 이는 $(2^3)!$ = 40320개의 割當종류중 한가지이며, Iwaro Takahashi[7] 등은 GF(2^4)上의 既約多項式으로 부터 얻은 係數를 各各 2値로 割當하여 元素를 구하였다.

한편 GF(P^m)上의 加算은 modP 合算[4,11-12]이므로

*正會員, 仁荷大學校 電子工學科
(Dept. of Elec. Eng., Inha Univ.)

接受日字 : 1985年 10月 29日

GF(2^m)上의 加算은 mod2 合算인 exclusive OR 演算으로 가능하다.[5-6,9,11,13] 이를 토대로 Dhiraj K. Pradhan과 Arvind M. Patel[11] 등은 GF(2^2)上의 加算器를 構成하였다. 또한 I. S. Reed[6] 등은 自乘(squaring) 方法에 의해 GF(2^4)上의 乘算을 행하고자 sequential 형태, parallel형태 및 pipeline형태로 各各 Massey-Omura 乘算器를 構成 하였다.

本 論文의 叙述過程은 다음과 같다.

第Ⅱ章에서는 素數가 2인 Galois體의 既約多項式으로 부터 係數를 bit code로 割當하는 알고리즘을 밝히고, 第Ⅲ章에서는 bit code로 割當된 元素들 간의 演算(加算, 乘算)을 멀티플렉서를 사용하여 加算器와 乘算器를 構成하였다. 第Ⅳ章에서는 Ⅲ章에서 다룬 內容과는 별도로 函數構成時 狀態遷移圖를 이용한 函數最小化方法 알고리즘을 提示하고, 이들의 回路構成은 VLSI設計에 널리 사용하는 PLA로 實現하였다. 第Ⅴ章에서는 Ⅱ章과 Ⅳ章의 내용을 他論文의 例를 引用하여 適用하고, 引用한 論文의 結果와 本 論文의 結果를 比較 檢討 하였다. 第Ⅵ章의 結論 에서는 本 論文에서 提示한 加算器와 乘算器 및 函數最小化方法의 특징을 요약하였고 앞으로의 展望을 記述하였다.

Ⅱ. GF(2^m)上의 모든 元素들을 Bit Code로 割當하는 알고리즘

이 章에서 引用된 GF(P^m)上에서 成立하는 數學的 性質은 이미 發表된 여러 論文에서 證明없이 도입하여 사용한다. 그중 다음 式(1)을 因數分解 하여 m次 既約多項式을 구하고 이 既約多項式을 0으로 하는 한 根을 α로 할 때 式(2)와 같은 多項式을 얻을 수 있다.

$$X^{P^m} - X = 0 \quad [18-19] \qquad (1)$$

여기서 P : 素數, m : 陽의 整數

$$F(\alpha) = \sum_{i=0}^{m-1} a_i \alpha^i$$

$$= a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{m-1}\alpha^{m-1} \quad [18-19] \qquad (2)$$

여기서 α는 P를 法으로 한 整數體 Z_P의 元素를 係數로 하는 m次 既約多項式의 根이고 $a_i \in Z_P$ (i=0, 1, 2, ⋯, P^m-1) 이다.

여기에서는 P=2인 경우이므로 式(2)는 모든 係數가 0 또는 1로 표시된다. 즉, 2値의 값을 나타내는 bit code로써 표시 할 수 있다. 따라서 P=2인경우는 式(2)로 부터 係數를 bit code로 割當함으로써 GF(2^m)上의 모든 元素로 표시 할 수 있다. 이때 GF(2^m)上의 모든 元素를 bit code로 割當하는데 必要한 bit code의 갯수는 m개이고, 最高次數係數인 a_{m-1}을 MSB(most significant bit)로 最低次數係數인 a_0를 LSB(least significant bit)로 표시한다.

또한 係數 a_i가 1인 갯수를 level이라하면 다음과 같은 level과 元素갯수가 生成된다.

0 -level의 元素갯수 : ${}_mC_0 = 1$[개]
m -level의 元素갯수 : ${}_mC_m = 1$[개]
1 -level의 元素갯수 : ${}_mC_1$[개]
2 -level의 元素갯 : ${}_mC_2$[개]
⋮
(m-2)-level의 元素갯수 : ${}_mC_{m-2} = {}_mC_2$[개]
(m-1)-level의 元素갯수 : ${}_mC_{m-1} = {}_mC_1$[개]

위에서와 같이 GF(2^m)上의 元素를 bit code로 표시하면 (m+1)개의 level이 生成된다.

以上의 內容을 綜合하면 GF(2^m)上의 元素들을 bit code로 割當하는 알고리즘을 다음과 같이 세울 수 있다.

〔알고리즘〕

元素를 e_i (i=0, 1, 2, ⋯, 2^m-1)로 표시 한다.

단계 1 : $a_{m-1}a_{m-2}\cdots a_1a_0$의 순서로 모든 係數를 나열하고 係數 a_{m-1}을 MSB로 係數 a_0를 LSB로 위치시킨다.

단계 2 : 元素 e_0는 모든 係數가 0일때도 割當한다. (0 -level)

단계 3 : 元素 e_{2^m-1}은 모든 係數가 1일 때로 割當한다. (m -level)

단계 4 : 1 -level의 ${}_mC_1$개의 元素들은 LSB로 부터 bit code를 1로하여 채워나가고 순서대로 元素 e_i를 割當한다.

단계 5 : 2 -level의 ${}_mC_2$개의 元素들은 MSB로 부터 bit code를 1로하여 채워나가고 bit code 들끼리 組合하여 차례로 元素 e_i를 割當한다.

단계 6 : (m-1) -level까지 各各의 元素에 대해 단계 5와 같은 方法으로 차례로 元素 e_i를 割當한다.

例) GF(2^3)上의 모든 元素들을 앞에서 提示한 알고리즘에 의해 bit code로 割當하면 表 1과 같다.

Ⅲ. 加算器 및 乘算器 構成

1. Bit Code에 依한 元素들간의 演算

GF(2^m)上의 모든 元素들을 bit code로 割當한 元素들간의 bit code 演算(加算, 乘算)은 다음과 같은 一般的인 형태로 표시 할 수 있다.

1) 加算

GF(2^m)上에서 彼加算元素를 $e_i(a_{m-1}a_{m-2}\cdots a_1a_0)$, 加算元素를 $e_j(b_{m-1}b_{m-2}\cdots b_1b_0)$라 하고 두元素 e_i와 e_j의 加算後의 元素를 e_A라 하면 다음 式(3)으로 표시된

표 1. GF(2³)의 원소를 bit code 할당
Table 1. An assignment of elments on GF(2³) to bit codes.

STEP	element	coefficient	LEVEL	$F(\alpha)=a_2\alpha^2+a_1\alpha+a_0$
1		$a_2\ a_1\ a_0$		
2	e_0	0 0 0	0	0
4	e_1	0 0 1	1	1
	e_2	0 1 0		α
	e_3	1 0 0		α^2
5	e_4	1 1 0	2	$\alpha^2+\alpha$
	e_5	1 0 1		α^2+1
	e_6	0 1 1		$\alpha^3=\alpha+1$
3	e_7	1 1 1	3	$\alpha^2+\alpha+1$

where, $\alpha^3=\alpha+1$, $-1\equiv 1$

다.

$$e_A=e_i\oplus e_j=(a_k\oplus b_k)=(z_k) \quad (3)$$

여기서 $a_k, b_k, z_k\in Z_2=\{0,1\}$ $(k=m-1, m-2, \cdots, 1, 0)$
$i, j=0, 1, \cdots, 2^m-1$
$\oplus$: mod 2合

2) 乘算

GF(2^m)上에서 彼乘算元素를 $e_i(a_{m-1}a_{m-2}\cdots a_1a_0)$, 乘算元素를 $e_j(b_{m-1}b_{m-2}\cdots b_1b_0)$라 하고 두元素 e_i와 e_j의 乘算後의 元素를 e_M이라 하면 다음 式(4)로 표시된다.

$$e_M=e_i\cdot e_j=(a_k\cdot b_k)=(S_l z_k) \quad (4)$$

여기서 S_l : sign bit
$a_k, b_k, z_k, S_l\in Z_2=\{0,1\}$ $(k=m-1, m-2, \cdots, 1, 0 : l=k-1)$
$i, j=0, 1, \cdots, 2^m-1$

여기에서 乘算後 元素 e_M은 式(4)의 sign bit S_l의 組合에 따라 bit code z_k의 값을 그대로 유지하거나 補數를 취하면 된다. 따라서 S_l이 다음 3節에서의 乘算器의 制御入力으로 사용된다.

2. 加算器 構成

彼加算元素 e_i와 加算元素 e_j의 bit code 加算에서 加算後 元素 e_A의 bit code z_k는 다음과 같다.

彼加算元素 e_i의 bit code a_k가 (1) 0일때는 加算元素 e_j의 bit code b_k값이 그대로 z_k가 되고, (2) 1일때는 b_k를 補數로 취한값이 z_k가 된다. 따라서 彼加算元素 e_j의 bit code a_k가 制御入力으로 사용된다.

以上을 式으로 표시하면 다음 式(5)와 같고

$$e_i(a_k)\oplus e_j(b_k)=e_A(z_k) \quad (5)$$

여기서 $z_k=\begin{cases} b_k & \text{iff } a_k=0 \\ b'k & \text{iff } a_k=1 \end{cases}$

이 式을 멀티플렉서(MUX)를 사용한 加算器의 一般的인 block線圖로 나타내면 그림 1과 그림 2와 같다.

例) GF(2³)上에서 成立하는 加法인 다음 表2를 MUX를 사용한 加算器로 構成하면 다음과 같다.

먼저 式(5)를 이용하여 加算後 元素 e_A를 制御入力 bit code인 彼加算元素의 b_k로 표시하면 表 3과 같은 加算MUX表를 얻을 수 있으며 이를 MUX를 사용한 回路로 構成하면 그림 3과 같다.

3. 乘算器 構成

彼乘算元素 e_i와 乘算元素 e_j의 bit code 乘算에서

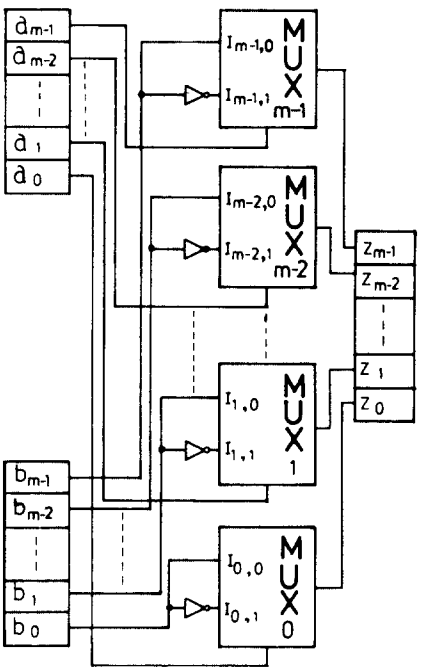

그림 1. GF(2^m)의 MUX를 사용한 일반적인 가산기 형태
Fig. 1. A generalized Adder type using MUX on GF(2^m).

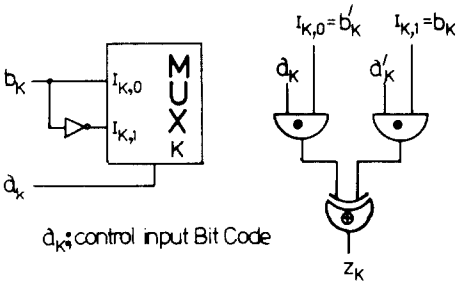

그림 2. MUX 내부회로
Fig. 2. Internal circuits of MUX.

표 2. GF(2³)의 가법표
Table 2. Addition table on GF(2³).

+	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7
e_0	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7
e_1	e_1	e_0	e_6	e_5	e_7	e_3	e_2	e_4
e_2	e_2	e_6	e_0	e_4	e_3	e_7	e_1	e_5
e_3	e_3	e_5	e_4	e_0	e_2	e_1	e_7	e_6
e_4	e_4	e_7	e_3	e_2	e_0	e_6	e_5	e_1
e_5	e_5	e_3	e_7	e_1	e_6	e_0	e_4	e_2
e_6	e_6	e_2	e_1	e_7	e_5	e_4	e_0	e_3
e_7	e_7	e_4	e_5	e_6	e_1	e_2	e_3	e_0

표 3. GF(2³)의 가산 MUX표
Table 3. Addition MUX table on GF(2³).

$e_i(b_2b_1b_0)$	$e_j(b_2b_1b_0)$	$e_A(z_2z_1z_0)$
$e_0(0\,0\,0)$	$e_j(\ b_n\ )$	$e_A(b_2b_1b_0)$
$e_1(0\,0\,1)$	$e_j(\ b_n\ )$	$e_A(b_2b_1b_0')$
$e_2(0\,1\,0)$	$e_j(\ b_n\ )$	$e_A(b_2b_1'b_0)$
$e_3(1\,0\,0)$	$e_j(\ b_n\ )$	$e_A(b_2'b_1b_0)$
$e_4(1\,1\,0)$	$e_j(\ b_n\ )$	$e_A(b_2'b_1'b_0)$
$e_5(1\,0\,1)$	$e_j(\ b_n\ )$	$e_A(b_2'b_1b_0')$
$e_6(0\,1\,1)$	$e_j(\ b_n\ )$	$e_A(b_2b_1'b_0')$
$e_7(1\,1\,1)$	$e_j(\ b_n\ )$	$e_A(b_2'b_1'b_0')$

where, $j=0,1,2,\cdots\cdots,7$
$n=0,1,2$

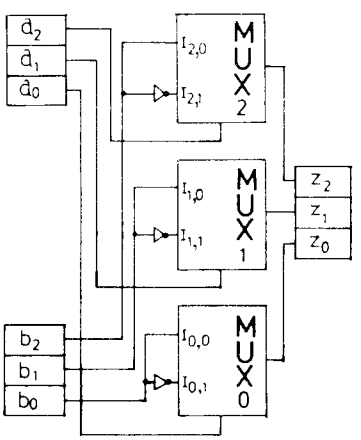

그림 3. GF(2³)의 MUX를 사용한 가산기
Fig. 3. Adder on GF(2³) using MUX.

는 sign bit S_i이 生成되는데 이 S_i의 組合이 乘算器의 制御入力으로 사용된다.

따라서 式(4)를 토대로 GF(2ᵐ)上의 一般的인 乘算器를 構成하면 그림 4와 같다.

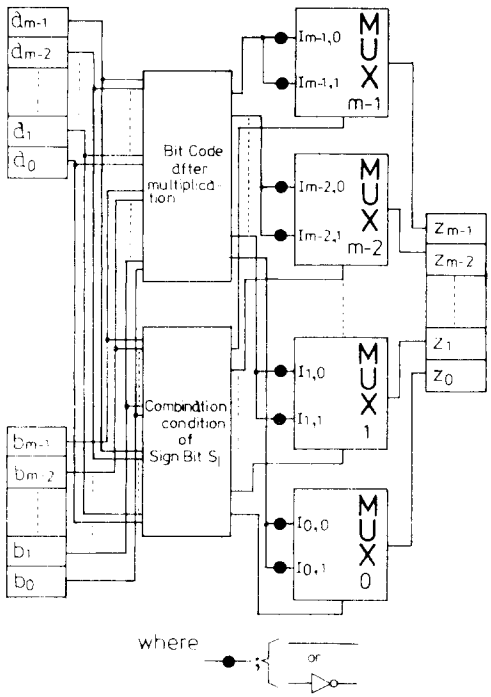

그림 4. GF(2ᵐ)의 MUX를 사용한 일반적인 승산기 형태
Fig. 4. A generalized Multiplier type using MUX on GF(2ᵐ).

例) GF(2³)인 경우의 乘算器를 式(4)와 그림4를 사용하여 構成하면 다음과 같다.

먼저 式(4)를 展開하면 다음 式(6)이 되고

$$e_M = e_i \cdot e_j = (a_2a_1a_0)\cdot(b_2b_1b_0) = (S_1S_0z_2z_1z_0) \quad (6)$$

이기서
$$\begin{cases} S_1 = a_2 \cdot b_2 \\ S_0 = a_2 \cdot b_1 \oplus a_1 \cdot b_2 \\ z_2 = a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \\ z_1 = a_1 \cdot b_0 \oplus a_0 \cdot b_1 \\ z_0 = a_0 \cdot b_0 \end{cases}$$

이고 最終 乘算後의 元素 e_M의 bit code z_k(K=0, 1, 2)는 다음과 같다.

$$z_2 = \begin{cases} z_2 & \text{iff} \quad S_1 = 0 \\ z_2' & \text{iff} \quad S_1 = 1 \end{cases}$$

$$z_1 = \begin{cases} z_1 & \text{iff} \quad S_T = 0 \\ z_1' & \text{iff} \quad S_T = 1 \end{cases}$$

여기서 $S_T = S_1 \oplus S_0$

$$z_0 = \begin{cases} z_0 & \text{iff} \quad S_0 = 0 \\ z_0' & \text{iff} \quad S_0 = 1 \end{cases}$$

위 內容을 乘算MUX表로 나타내면 表 4와 같고 MUX를 사용하여 乘算器를 構成하면 그림 5와 같다.

표 4. GF(2³)의 승산 MUX표
Table 4. Multiplication MUX table on GF(2³).

Sign Bit S_1 S_0	Bit Code conversion for Sign Bit condition: Z_2	control input Bit Code	Z_1	control input Bit Code	Z_0	control input Bit Code
0 0	Z_2	S_1	Z_1	$S_T = S_1 \oplus S_0$	Z_0	S_0
0 1	Z_2		Z_1'		Z_0'	
1 0	Z_2'		Z_1'		Z_0	
1 1	Z_2'		Z_1		Z_0'	

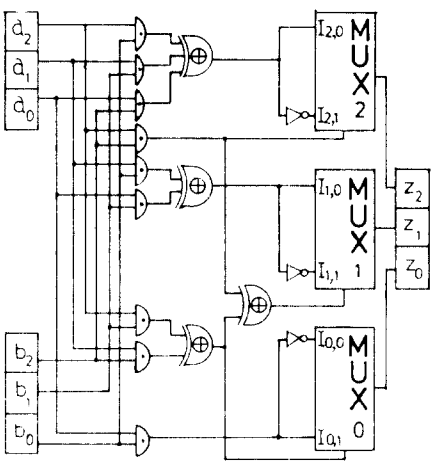

그림 5. GF(2³)의 MUX를 사용한 승산기
Fig. 5. Multiplier on GF(2³) using MUX.

Ⅳ. 函數構成理論 및 回路實現

1. 數學的 表記 및 演算子

이 章에서 사용하는 數學的 表記 및 演算子는 다음과 같다.

1) $\boxed{K_i}$: K_i bit code에서 don't care 성분을 뜻함.
여기서 K : 變數名 (a, b, c, …)
i : i번째 bit code

2) $-\boxed{K_i}-$: 狀態遷移圖에서 사용되며 연결자(connecter)라 명칭하고 인접한 元素들을 연결시켜 준다.

3) $\oplus$: mod2合

4) • : AND

5) $e_k^{\dot{R}}$: 元素표시

여기서 ① K : 變數名 (a, b, c, …)

② i : i번째 元素

③ 元素들 사이는 "·"로 연결한다.

例) $e_1^a = a_1' a_0$ (GF(2²)인 경우) $e_2^b = b_2' b_1 b_0'$ (GF(2³)인 경우)

$e_4^a \cdot e_4^b = a_3 a_2' a_1' a_0 \cdot b_3 b_2' b_1' b_0'$ (GF(2⁴)의 2變數인 경우)

6) $e_k^{\dot{m}\ \dot{m}-1\ \dot{m}-2 \cdots \dot{1}\ \dot{0}}$: bit code로 표시된 元素

여기서 ① K : 變數名 (a, b, c, …)

② m-1 m-2…10 : bit code 번호

③ $\dot{R}$ ($\dot{R}$ = m-1 m-2…1 0)는 다음과 같다.

$$\dot{R} = \begin{cases} R & : \text{만일 bit code값이 1일때} \\ R' & : \text{만일 bit code 값이 0일때} \end{cases}$$

④ 만일 任意의 bit code가 don't care이면 "–"로 해당 bit code를 표시한다.

例) $e_a^{2\,1\,0} = a_2 a_1' a_0$ $\quad e_b^{3\,0} = b_3 b_0'$

$e_c^{3\,-\,1} = c_3 c_1'$ $\quad e_a^{2\,1\,0} \cdot e_b^{3\,0} \cdot e_c^{3\,1} = a_2 a_1' a_0 b_3 b_0' c_3' c_1$

2. 狀態遷移圖 構成 알고리즘

GF(2^m) 上의 모든 元素를 bit code로 割當한 것은 2値化한 것과 같으므로 現在의 論理理論에서의 函數最小化方法을 약간 수정없이 適用할 수 있는 이점을 있지만, 2値에서의 여러 函數最小化方法과 같이 重複하여 函數를 最小化 할 수는 없다. 그 理由는 GF(2^m) 上에서는 mod2算法을 하여야 하므로 重複을 할 수 없기 때문이다.[14]

이러한 點을 고려하여 本 論文에서는 새로운 형태의 狀態遷移圖를 이용한 方法에 의한 函數最小化方法을 提示 한다.

Ⅱ章에서의 bit code로 割當한 GF(2^m) 上의 元素들 중에서, 初期狀態를 e_0(0-level)로 最終狀態를 e_{2^m-1}(m-level)로 하면 元素들간의 狀態遷移圖를 構成하는 알고리즘을 다음과 같이 세울 수 있다.

〔알고리즘〕

단계 1 : 初期狀態인 e_0에서 1개의 bit code가 다른 값을 가진 元素를 앞節의 연결자를 사용하여 e_0와 연결한다. (1-level)

단계 2 : 1-level에 속하는 元素들로 부터 各各 1개의 bit code가 다른 값을 가진 元素를 연결자로 연결한다. (2-level)

단계 3 : 단계 2와 같은 方法으로 (m-1)-level까지의 元素들을 연결자로 연결한다.

단계 4 : (m-1)-level의 元素들과 最終狀態인 e_{2^m-1}을 연결자로 연결한다.

위 알고리즘을 綜合하여 GF(2^m) 上의 一般的인 狀態遷移圖를 構成하면 그림 6과 같고 이로부터 GF(2^2),

GF(2³), GF(2⁴)上의 狀態遷移圖를 구하면 各各 그림 7, 그림 8, 그림 9와 같다.

여기서 $\boxed{a_k}$ 는 上位 level들의 연결자를 제외한 연결자 중의 1개의 연결자 이다.

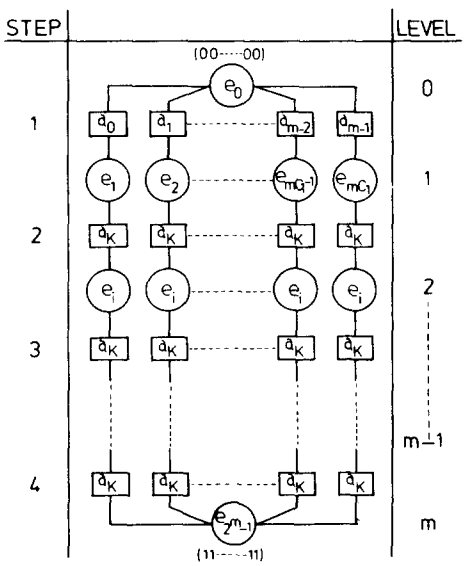

그림 6. GF(2ᵐ)上의 일반적인 상태천이도
Fig. 6. A generalized state-transition diagram on GF(2ᵐ).

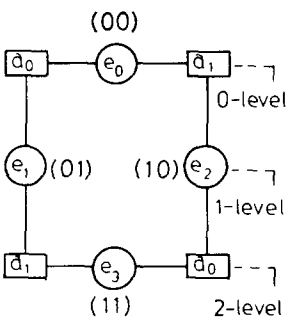

그림 7. GF(2²)의 상태천이도
Fig. 7. State-transition diagram on GF(2²).

3. 函數最小化方法 및 回路實現 알고리즘

1節의 數學的 表記와 演算子, 2節의 狀態遷移圖를 이용하여 最小化된 函數를 구하고 이 函數式으로 부터 PLA로 回路를 實現하는 알고리즘은 다음과 같다.

단계 1 : 任意의 眞理表에서 入力元素와 出力元素를 各各 Ⅱ章에서 割當한 bit code로 변환한다.

단계 2 : 단계 1에서 구한 表의 出力을 bit code 별로 各各 分割하고 出力이 1인 入力元素를 2節의 狀態遷移圖에 "(e_i^K)" 로 표시한다.

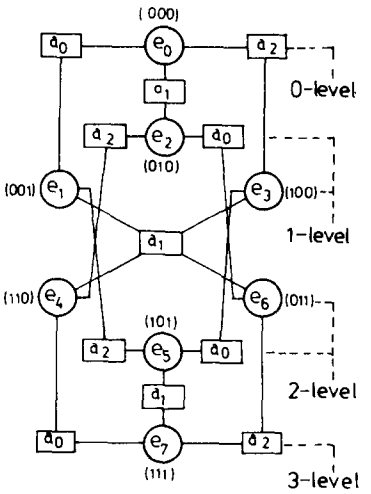

그림 8. GF(2³)의 상태천이도
Fig. 8. State-transition diagram on GF(2³).

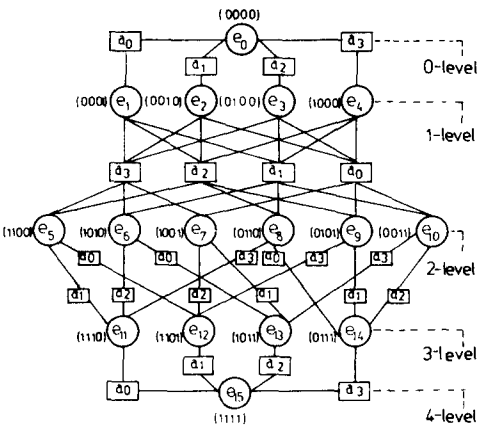

그림 9. GF(2⁴)의 상태천이도
Fig. 9. State-transition diagram on GF(2⁴).

단계 3 : 단계 2에서 구한 狀態遷移圖로 부터 다음과 같은 과정을 거쳐 函數를 最小化한다.

단계3-1 : 狀態遷移圖에서 연결자 (—$\boxed{K_i}$—)로 연결된 元素를 찾아 K_i별로 갯수를 파악한다.

단계3-2 : K_i중 가장 갯수가 많은 K_i를 基準으로 묶는다. 이때 연결자로 연결된 元素들은 "X"로 지운다.

단계3-3 : 만일 서로 연결되지 않고 독립적으로 나타나는 元素는 "√"로 표시한다.

단계3-4 : 모든 元素가 "X"와 "√"로 표시 되었으면 1節의 數學的 表記와 演算子들을 이

용하여 函數를 最小化 한다.

단계3-5 : 동일한 연결자로 연결된 元素 쌍(pair)이 다시 任意의 동일한 연결자로써 연결되어 閉루프를 形成하면, 이는 don't care 2개가 生成됨을 의미한다. 이때의 閉루프를 1次元 閉루프라 명칭한다.

단계3-6 : 다시 단계3-5의 閉루프가 任意의 동일한 연결자로써 연결되어 새로운 閉루프가 形成되면 don't care 3개가 生成됨을 의미한다. 이때의 閉루프를 2次元 閉루프라 명칭한다.

단계 4 : 단계 3에서 구한 函數式에 mod2算法의 性質을 適用하여 최종의 最小化函數式을 구한다.

단계 5 : 단계 4에서 구한 最小化 된 函數式으로 부터 다음과 같은 과정을 거쳐 PLA回路를 設計한다.

단계5-1 : 最小化函數式에서 各 項의 bit code 갯수별로 PLA表를 作成한다. 이때 最大 bit code 갯수는 入力元素의 bit code 갯수와 동일하다.

단계5-2 : 任意의 項이 分割된 最小化 函數式에 포함되어 있다면 PLA表의 해당 函數에 "√" 표시 한다.

단계5-3 : 단계5-2에서 구한 PLA表로 부터 AND-exclusive OR array의 PLA回路를 實現한다.

V. 適用例

이 章에서는 Ⅱ章과 Ⅳ章의 內容이 어떻게 適用되는지를 他論文에서 다룬 例에 適用한후, 結果를 比較檢討하였다.

例 1)[11] GF(2²)上 2變數 入力, 單一 出力인 경우

단계 1 : 眞理表 5를 Ⅱ章에서 提示한 bit code 割當 알고리즘에 의해 元素들을 bit code로 割當하면 表 6과 같다.

진 리 표 5. GF(2²)의 2변수 입력, 단일출력
Truth table 5. Two-variable input, single output on GF(2²).

e_j \ e_i	e_0	e_1	e_2	e_3	
e_0	e_0	e_0	e_1	e_1	F_Z
e_1	e_1	e_0	e_3	e_2	
e_2	e_3	e_1	e_3	e_1	
e_3	e_2	e_1	e_1	e_2	

표 6. 진리표 5를 bit code로 할당
Table 6. An assignment of elements in Truth table5 to bit codes.

$e_i(a_1\,a_0)$ / $e_j(b_1\,b_0)$	$e_0(00)$	$e_1(01)$	$e_2(10)$	$e_3(11)$	
$e_0(00)$	$e_0(00)$	$e_0(00)$	$e_1(01)$	$e_1(01)$	$F_Z(Z_1Z_0)$
$e_1(01)$	$e_1(01)$	$e_0(00)$	$e_3(11)$	$e_2(10)$	
$e_2(10)$	$e_3(11)$	$e_1(01)$	$e_3(11)$	$e_1(01)$	
$e_3(11)$	$e_2(10)$	$e_1(01)$	$e_1(01)$	$e_2(10)$	

위 表 6에 Ⅳ章의 函數最小化方法을 適用하고 結果를 PLA回路로 構成하면 다음과 같다.

단계 2 ~ 단계 4

1) $F(Z_1)$에 대해서

$$F(Z_1) = (e_0^a(e_2^b \oplus e_3^b),\ e_2^a(e_1^b \oplus e_2^b)),\ e_3^a(e_1^b \oplus e_3^b)$$
$$= (e_0^a\,\boxed{a_1}\,e_2^a)\,e_2^b \oplus e_0^a(e_3^b) \oplus e_2^a(e_1^b) \oplus e_3^a(e_1^b \oplus e_3^b)$$
$$= e_a^{-0'} \cdot e_b^{10'} \oplus e_a^{1'0'} \cdot e_b^{10} \oplus e_a^{10'} \cdot e_b^{1'0} \oplus e_a^{10} \cdot e_b^{-0} \qquad (7)$$

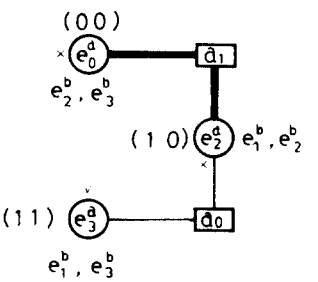

그림10. 표 6에 대한 부분함수 $F(Z_1)$
Fig. 10. Partitioned function $F(Z_1)$ for the Table 6.

2) $F(Z_0)$에 대해서

$$F(Z_0) = ((e_0^a(e_1^b \oplus e_2^b),\ e_2^a(e_0^b \oplus e_1^b \oplus e_2^b \oplus e_3^b)),$$
$$(e_1^a(e_2^b \oplus e_3^b),\ e_3^a(e_0^b \oplus e_2^b))$$
$$= (e_0^a\,\boxed{a_1}\,e_2^a)\,(e_1^b \oplus e_2^b) \oplus e_2^a(e_0^b \oplus e_3^b)$$
$$\oplus (e_1^a\,\boxed{a_1}\,e_3^a)\,e_2^b \oplus e_1^a(e_3^b) \oplus e_3^a(e_0^b)$$
$$\therefore F(Z_0) = e_a^{-0'} \cdot e_b^{1'0} \oplus e_a^{-0'} \cdot e_b^{10'} \oplus e_a^{10'} \cdot e_b^{10} \oplus e_a^{-0} \cdot e_b^{1'0'}$$
$$\oplus e_a^{1'0} \cdot e_b^{10} \oplus e_a^{1-} \cdot e_b^{1'0'} \qquad (8)$$

따라서 최종 구하는 最小化 된 函數式은 다음과 같다.

$$F_Z = F(Z_1Z_0)$$
$$= ((e_a^{-0'} \cdot e_b^{10'} \oplus e_a^{1'0'} \cdot e_b^{10} \oplus e_a^{10'} \cdot e_b^{1'0} \oplus e_a^{10} \cdot e_b^{-0})$$
$$(e_a^{-0'} \cdot e_b^{1'0} \oplus e_a^{-0'} \cdot e_b^{10'} \oplus e_a^{10'} \cdot e_b^{10} \oplus e_a^{-0} \cdot e_b^{10'}$$
$$\oplus e_a^{1'0} \cdot e_b^{10} \oplus e_a^{1-} \cdot e_b^{1'0'})) \qquad (9)$$

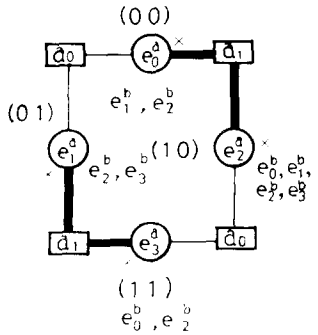

그림11. 표6에 대한 부분함수 $F(Z_0)$
Fig. 11. Partitioned function $F(Z_0)$ for the table 6.

단계 5 : 式(9)로 부터 PLA表를 作成하면 表7과 같고 이를 PLA回路로 實現하면 그림12와 같다. 위 結果를 引用한 論文의 結果와 比較하면 다음 表8과 같다.

표 7. PLA 표
Table 7. PLA table.

Number of Bit Code	product term	input Bit Code $a_1\ a_0\ b_1\ b_0$	output F_z $F(z_1)$	$F(z_0)$
4	$e_a^{0'} e_b^{10}$	0 0 1 1	√	
	$e_a^{10'} e_b^{10}$	1 0 0 1	√	
	$e_a^{10'} e_b^{10}$	1 0 1 1		√
	$e_a^{10'} e_b^{10}$	0 1 1 1		√
3	$e_a^{-0'} e_b^{10}$	− 0 1 0	√	√
	$e_a^{10} e_b^{-0}$	1 1 − 1	√	
	$e_a^{-0'} e_b^{10'}$	− 0 0 1		√
	$e_a^{-0} e_b^{10'}$	− 1 1 0		√
	$e_a^{1-} e_b^{10'}$	1 − 0 0		√

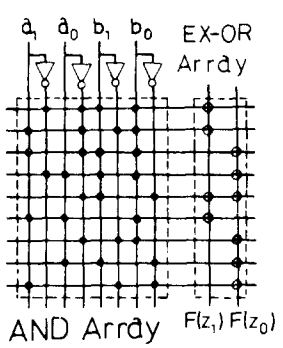

그림12. 함수 F_z의 PLA 논리회로 실현
Fig. 12. PLA logic circuits realization of the function F_z.

표 8. 비교표
Table 8. Comparison table.

	Dhiraj K. Pradhan and Arvind M. Patel	This paper
AND gate	28	31
EX-OR gate	21	10
Inverter		4
Total	49	45

例2)[5] $GF(2^3)$上 單一變數 入出力인 경우
단계 1 : 眞理表9를 例1의 方法과 같이 하면 表10을 얻는다.

진 리 표9. $GF(2^3)$의 단일변수 입출력
Truth table 9. Single-variable input-output on GF (2^3).

e_i	F_z
e_0	e_5
e_1	e_0
e_2	e_6
e_3	e_5
e_4	e_2
e_5	e_0
e_6	e_2
e_7	e_6

표 10. 진리표9를 bit code로 할당
Table 10. An assignment of elements in truth table 9 to bit codes.

$e_i(a_2 a_1 a_0)$	$F_z(z_2 z_1 z_0)$
$e_0(0\ 0\ 0)$	$e_5(1\ 0\ 1)$
$e_1(0\ 0\ 1)$	$e_0(0\ 0\ 0)$
$e_2(0\ 1\ 0)$	$e_6(0\ 1\ 1)$
$e_3(1\ 0\ 0)$	$e_5(1\ 0\ 1)$
$e_4(1\ 1\ 0)$	$e_2(0\ 1\ 0)$
$e_5(1\ 0\ 1)$	$e_0(0\ 0\ 0)$
$e_6(0\ 1\ 1)$	$e_2(0\ 1\ 0)$
$e_7(1\ 1\ 1)$	$e_6(0\ 1\ 1)$

위 表10에 例1의 단계 2부터 단계 5까지의 方法과 같이 하면 다음과 같다.

단계 2～단계 4

1) $F(Z_2)$에 대해서

$$F(Z_2) = (e_0^a, e_3^a) = e_0^a \widetilde{a_2} e_3^a = e_a^{1'0'} \quad (10)$$

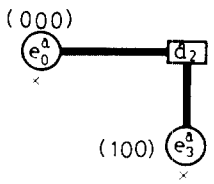

그림13. 표10에 대한 부분함수 $F(Z_2)$
Fig. 13. Partitioned function $F(Z_2)$ for the table 10.

2) $F(Z_1)$에 대해서

$$F(Z_1) = ((e_2^a, e_4^a), (e_6^a, e_7^a))$$
$$= (e_2^a \boxed{a_2} e_4^a) \boxed{a_0} (e_6^a \boxed{a_2} e_7^a)$$
$$= e_a^{-1} \qquad (11)$$

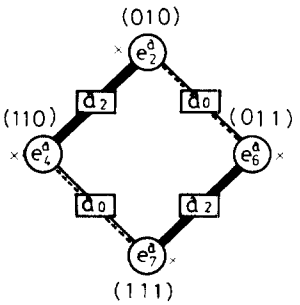

그림14. 표10에 대한 부분함수 $F(Z_1)$
Fig. 14. Partitioned function $F(Z_1)$ for the table 10.

그림14는 1次元 閉루프가 生成됨을 보여준다.

3) $F(Z_0)$에 대해서

$$F(Z_0) = (e_0^a, e_3^a), e_2^a, e_7^a$$
$$= (e_0^a \boxed{a_2} e_3^a) \oplus e_2^a \oplus e_7^a$$
$$= e_a^{-1'0'} \oplus e_a^{2'1'0'} \oplus e_a^{210} \qquad (12)$$

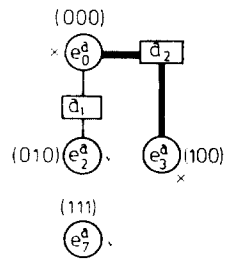

그림15. 표10에 대한 부분함수 $F(Z_0)$
Fig. 15. Partitioned function $F(Z_0)$ for the table10.

따라서 $F_Z = F(Z_2 Z_1 Z_0)$

$$= ((e_a^{-1'0'}) (e_a^{-1-}) (e_a^{-1'0'} \oplus e_a^{2'1'0'} \oplus e_a^{210})) \qquad (13)$$

단계 5 : 式(13)으로 부터 PLA表와 PLA回路를 實現하면 다음 表11과 그림16과 같다.

위 結果와 引用한 論文의 結果를 比較하면 다음 表12와 같다.

표 11. PLA 표
Table11. PLA table.

Number of Bit Code	product term	input Bit Code a_2 a_1 a_0	output F_Z $F(Z_2)$	$F(Z_1)$	$F(Z_0)$
3	$e_a^{2'1'0'}$	0 1 0			√
	e_a^{210}	1 1 1			√
2	$e_a^{-1'0'}$	− 0 0	√		√
1	e_a^{-1-}	− 1 −		√	

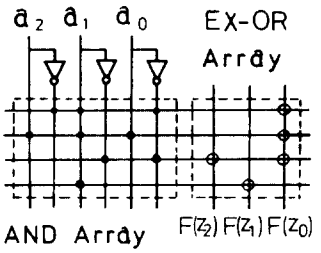

그림16. 함수 F_Z의 PLA 논리회로 실현
Fig. 16. PLA logic circuits realization of the function F_Z.

표 12. 비교표
Table12. Comparison table.

	Karl. S.Menger	This paper
Input	18	3
Inverter		3
AND gate	5	9
EX-OR gate	4	5
Total	27	20

例) 3[7] GF(2⁴)上 單一變數 入出力인 경우

단계 1 : 眞理表13을 例1의 方法과 같이 하면 表14를 얻는다.

表14에 例1의 단계 2 부터 단계 5 까지의 方法을 適用하면 다음과 같다.

단계 2 ~ 단계 4

1) $F(Z_3)$에 대해서

진 리 표13. GF(2⁴)의 단일변수 입출력
Truth table13. Single-variable input-output on GF (2⁴).

e_i	F_Z	e_i	F_Z
e_0	e_0	e_8	e_{11}
e_1	e_1	e_9	e_8
e_2	e_1	e_{10}	e_8
e_3	e_1	e_{11}	e_{11}
e_4	e_1	e_{12}	e_{11}
e_5	e_8	e_{13}	e_0
e_6	e_8	e_{14}	e_{11}
e_7	e_8	e_{15}	e_8

표 14. 진리표13을 bit code로 할당
Table14. An assignment of elements in truth table 13. to bit codes.

$e_i(a_3\,a_2\,a_1\,a_0)$	$F_Z(z_3\,z_2\,z_1\,z_0)$	$e_i(a_3\,a_2\,a_1\,a_0)$	$F_Z(z_3\,z_2\,z_1\,z_0)$
$e_0(0\,0\,0\,0)$	$e_0(0\,0\,0\,0)$	$e_8(0\,1\,1\,0)$	$e_{11}(1\,1\,1\,0)$
$e_1(0\,0\,0\,1)$	$e_1(0\,0\,0\,1)$	$e_9(0\,1\,0\,1)$	$e_8(0\,1\,1\,0)$
$e_2(0\,0\,1\,0)$	$e_1(0\,0\,0\,1)$	$e_{10}(0\,0\,1\,1)$	$e_8(0\,1\,1\,0)$
$e_3(0\,1\,0\,0)$	$e_1(0\,0\,0\,1)$	$e_{11}(1\,1\,1\,0)$	$e_{11}(1\,1\,1\,0)$
$e_4(1\,0\,0\,0)$	$e_1(0\,0\,0\,1)$	$e_{12}(1\,1\,0\,1)$	$e_{11}(1\,1\,1\,0)$
$e_5(1\,1\,0\,0)$	$e_8(0\,1\,1\,0)$	$e_{13}(1\,0\,1\,1)$	$e_0(0\,0\,0\,0)$
$e_6(1\,0\,1\,0)$	$e_8(0\,1\,1\,0)$	$e_{14}(0\,1\,1\,1)$	$e_{11}(1\,1\,1\,0)$
$e_7(1\,0\,0\,1)$	$e_8(0\,1\,1\,0)$	$e_{15}(1\,1\,1\,1)$	$e_8(0\,1\,1\,0)$

$$F(Z_3) = (e_8^a, e_{11}^a), e_{12}^a, e_{14}^a$$
$$= (e_8^a \boxed{a_3} e_{11}^a) \oplus e_{12}^a \oplus e_{14}^a$$
$$= e_a^{\overline{2\,1\,0}} \oplus e_a^{3\,2\,1\,\overline{0}} \oplus e_a^{\overline{3}\,2\,1\,0} \quad (14)$$

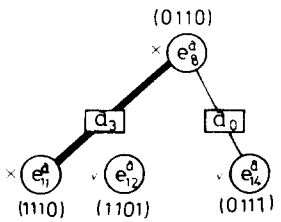

그림17. 표14에 대한 부분함수 $F(Z_3)$
Fig. 17. Partitioned function $F(Z_3)$ for the table 14.

2) $F(Z_2)$에 대해서

$$F(Z_2) = ((e_5^a, e_{11}^a), (e_{12}^a, e_{15}^a)), (e_9^a, e_{14}^a), e_8^a, e_7^a, e_6^a, e_{10}^a$$
$$= (e_5^a \boxed{a_1} e_{11}^a) \boxed{a_0} (e_{12}^a \boxed{a_1} e_{15}^a) \oplus (e_9^a \boxed{a_1} e_{14}^a) \oplus e_6^a \oplus e_7^a \oplus e_8^a \oplus e_{10}^a$$
$$\therefore F(Z_2) = e_a^{32--} \oplus e_a^{\overline{3}\,2-0} \oplus e_a^{32\,\overline{1}\,0} \oplus e_a^{32\,\overline{1}\,\overline{0}} \oplus e_a^{\overline{3}\,2\,\overline{1}\,0} \oplus e_a^{\overline{3}\,\overline{2}\,1\,0} \quad (15)$$

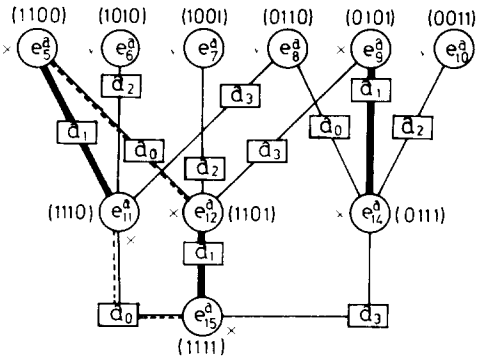

그림18. 표14에 대한 부분함수 $F(Z_2)$
Fig. 18. Partitioned function $F(Z_2)$ for the table 14.

3) $F(Z_1)$에 대해서

$F(Z_1)$에 대한 狀態遷移圖는 그림18과 같다. 따라서 部分函數 $F(Z_1)$은 $F(Z_2)$와 같다.

$$F(Z_1) = F(Z_2) = e_a^{32--} \oplus e_a^{\overline{3}\,\overline{2}\,\overline{1}\,0} \oplus e_a^{32\,\overline{1}\,\overline{0}} \oplus e_a^{32\,\overline{1}\,\overline{0}} \oplus e_a^{\overline{3}\,\overline{2}\,1\,\overline{0}} \oplus e_a^{\overline{3}\,\overline{2}\,\overline{1}\,0} \quad (16)$$

4) $F(Z_0)$에 대해서

이 경우는 各各의 狀態가 연결자로 연결되지 않았으므로 더 이상 狀態를 最小化할 수 없다.

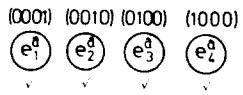

그림19. 표14에 대한 부분함수 $F(Z_0)$
Fig. 19. Partitioned function $F(Z_0)$ for the table 14.

$$F(Z_0) = e_1^a, e_2^a, e_3^a, e_4^a$$
$$= e^{\overline{3}\,\overline{2}\,\overline{1}\,\overline{0}} \oplus e_a^{\overline{3}\,\overline{2}\,\overline{1}\,0} \oplus e_a^{\overline{3}\,\overline{2}\,1\,\overline{0}} \oplus e_a^{32\,\overline{1}\,\overline{0}} \quad (17)$$

따라서 최종 구하는 最小化된 函數式은 다음 式(18)과 같다.

$$F_Z = F(Z_3 Z_2 Z_1 Z_0) \quad (18)$$

단계 5 : 式(18)로 부터 PLA表와 PLA回路를 實現하면 다음 表15와 그림20과 같다.

例4)[20] GF(2²)上 2變數 入力, 2出力 인 경우.

단계 1 : 眞理表16을 bit code로 割當하면 表17과같다.

단계 2 ~ 단계 4

1) F_{Z1}

표 15. PLA표
Table15. PLA table.

Number of Bit Code	product term	input Bit Code $a_3\ a_2\ a_1\ a_0$	output F_z F_{Z_3}	F_{Z_2}	F_{Z_1}	F_{Z_0}
4	e_0^{3210}	1 1 0 1	\			
	e_0^{3210}	0 - - 1	\			
	e_0^{3210}	1 0 1 0		\	\	
	e_0^{3210}	1 0 0 1		\	\	
	e_0^{3210}	0 - 1 0		\	\	
	e_0^{3210}	0 0 1 1		\	\	
	e_0^{3210}	0 0 0 -				\
	e_0^{3210}	0 0 - 0				\
	e_0^{3210}	0 1 0 0				\
	e_0^{3210}	1 0 0 0				\
3	e_3^{210}	- 1 1 0	\			
	e_0^{321}	0 1 - 1		\	\	
2	e_0^{32}	1 1 - -		\	\	

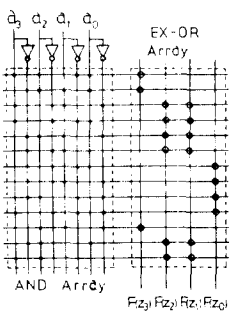

그림20. 함수 F_z의 PLA논리회로 실현
Fig. 20. PLA logic circuits realization of the function F_z.

진 리 표16. GF(2²)의 2변수 입력, 2출력
Truth table16. 2-Variable input, 2 output on GF(2²).

e_{i1}	e_{j1}	e_{i2}	e_{j2}	F_{Z1}	F_{Z2}
e_0	e_0	e_0	e_0	e_0	e_0
e_0	e_1	e_0	e_1	e_0	e_0
e_0	e_2	e_0	e_2	e_1	e_0
e_0	e_3	e_0	e_3	e_1	e_0
e_1	e_0	e_1	e_0	e_0	e_0
e_1	e_1	e_1	e_1	e_1	e_1
e_1	e_2	e_1	e_2	e_2	e_2
e_1	e_3	e_1	e_3	e_3	e_3
e_2	e_0	e_2	e_0	e_1	e_0
e_2	e_1	e_2	e_1	e_3	e_2
e_2	e_2	e_2	e_2	e_1	e_0
e_2	e_3	e_2	e_3	e_3	e_2
e_3	e_0	e_3	e_0	e_1	e_0
e_3	e_1	e_3	e_1	e_2	e_3
e_3	e_2	e_3	e_2	e_2	e_2
e_3	e_3	e_3	e_3	e_1	e_1

표 17. 진리표16을 bit code로 할당
Table17. An assignment of elements in truth table16 to bit codes.

$e_{i1}(a_1\ a_0)$	$e_{j1}(b_1\ b_0)$	$e_{i2}(c_1\ c_0)$	$e_{j2}(d_1\ d_0)$	$F_{Z1}(z_{11}\ z_{10})$	$F_{Z2}(z_{21}z_{20})$
$e_0(0\ 0)$	$e_0(0\ 0)$	$e_0(0\ 0)$	$e_0(0\ 0)$	$e_0(0\ 0)$	$e_0(0\ 0)$
$e_0(0\ 0)$	$e_1(0\ 1)$	$e_0(0\ 0)$	$e_1(0\ 1)$	$e_0(0\ 0)$	$e_0(0\ 0)$
$e_0(0\ 0)$	$e_2(1\ 0)$	$e_0(0\ 0)$	$e_2(1\ 0)$	$e_1(0\ 1)$	$e_0(0\ 0)$
$e_0(0\ 0)$	$e_3(1\ 1)$	$e_0(0\ 0)$	$e_3(1\ 1)$	$e_1(0\ 1)$	$e_0(0\ 0)$
$e_1(0\ 1)$	$e_0(0\ 0)$	$e_1(0\ 1)$	$e_0(0\ 0)$	$e_0(0\ 0)$	$e_0(0\ 0)$
$e_1(0\ 1)$	$e_1(0\ 1)$	$e_1(0\ 1)$	$e_1(0\ 1)$	$e_1(0\ 1)$	$e_1(0\ 1)$
$e_1(0\ 1)$	$e_2(1\ 0)$	$e_1(0\ 1)$	$e_2(1\ 0)$	$e_2(1\ 0)$	$e_2(1\ 0)$
$e_1(0\ 1)$	$e_3(1\ 1)$	$e_1(0\ 1)$	$e_3(1\ 1)$	$e_3(1\ 1)$	$e_3(1\ 1)$
$e_2(1\ 0)$	$e_0(0\ 0)$	$e_2(1\ 0)$	$e_0(0\ 0)$	$e_1(0\ 1)$	$e_0(0\ 0)$
$e_2(1\ 0)$	$e_1(0\ 1)$	$e_2(1\ 0)$	$e_1(0\ 1)$	$e_3(1\ 1)$	$e_2(1\ 0)$
$e_2(1\ 0)$	$e_2(1\ 0)$	$e_2(1\ 0)$	$e_2(1\ 0)$	$e_1(0\ 1)$	$e_0(0\ 0)$
$e_2(1\ 0)$	$e_3(1\ 1)$	$e_2(1\ 0)$	$e_3(1\ 1)$	$e_3(1\ 1)$	$e_2(1\ 0)$
$e_3(1\ 1)$	$e_0(0\ 0)$	$e_3(1\ 1)$	$e_0(0\ 0)$	$e_1(0\ 1)$	$e_0(0\ 0)$
$e_3(1\ 1)$	$e_1(0\ 1)$	$e_3(1\ 1)$	$e_1(0\ 1)$	$e_2(1\ 0)$	$e_3(1\ 1)$
$e_3(1\ 1)$	$e_2(1\ 0)$	$e_3(1\ 1)$	$e_2(1\ 0)$	$e_2(1\ 0)$	$e_2(1\ 0)$
$e_3(1\ 1)$	$e_3(1\ 1)$	$e_3(1\ 1)$	$e_3(1\ 1)$	$e_1(0\ 1)$	$e_1(0\ 1)$

(1-1) $F(Z_{11})$에 대해서

$$F(Z_{11}) = (e_1^a(e_2^b \oplus e_3^b) \oplus e_3^a(e_1^b \oplus e_2^b)),\ e_2^a(e_1^b \oplus e_3^b)$$
$$= (e_1^a \boxed{a_1} e_3^a)e_2^b \oplus e_1^a e_3^b \oplus e_3^a e_1^b \oplus e_2^a(e_1^b \boxed{b_1} e_3^b)$$
$$= e_a^{-0}e_b^{10} \oplus e_a^{1'0}e_b^{10} \oplus e_a^{10}e_b^{1'0} \oplus e_a^{10'}e_b^{-0} \quad (19)$$

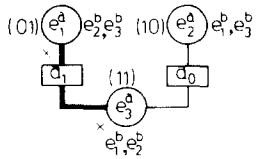

그림21. 표17에 대한 부분함수 $F(Z_{11})$
Fig. 21. Partitioned function $F(Z_{11})$ for the table17.

(1-2) $F(Z_{10})$에 대해서

$$F(Z_{10}) = (e_0^a(e_2^b \oplus e_3^b) \oplus e_2^a(e_0^b \oplus e_1^b \oplus e_2^b \oplus e_3^b)),$$
$$(e_1^a(e_1^b \oplus e_3^b) \oplus e_3^a(e_0^b \oplus e_3^b))$$
$$= (e_0^a \boxed{a_1} e_2^a)(e_2^b \boxed{b_0} e_3^b) \oplus e_2^a(e_0^b \boxed{b_0} e_1^b)$$
$$\oplus (e_1^a \boxed{a_1} e_3^a)e_3^b \oplus e_1^a e_1^b \oplus e_3^a e_0^b$$
$$= e_a^{-0'}e_b^{1-} \oplus e_a^{10'}e_b^{1'-} \oplus e_a^{-0}e_b^{10} \oplus e_a^{1'0}e_b^{1'0}$$
$$\oplus e_a^{10}e_a^{1'0'} \quad (20)$$

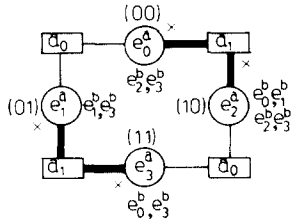

그림22. 표17에 대한 부분함수 $F(Z_{10})$
Fig. 22. Partitioned function $F(Z_{10})$ for the table17.

2) F_{Z2}

(2-1) $F(Z_{21})$에 대해서

$$F(Z_{21}) = (e_1^c(e_2^d \oplus e_3^d) \oplus e_3^c(e_1^d \oplus e_2^d)),\ e_3^c(e_1^d \oplus e_3^d)$$
$$= (e_1^c \boxed{c_1} e_3^c)\, e_2^d \oplus e_1^c e_3^d \oplus e_3^c e_1^d \oplus e_2^c(e_1^d \boxed{d_1} e_3^d)$$
$$= e_c^{-0} e_d^{10'} \oplus e_c^{1'0} e_d^{10} \oplus e_c^{10} e_d^{1'0} \oplus e_c^{10'} e_d^{-0} \qquad (21)$$

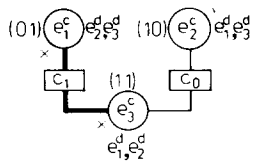

그림23. 표17에 대한 부분함수 $F(Z_{21})$
Fig. 23. Partitioned function $F(Z_{21})$ for the table 17.

(2-2) $F(Z_{20})$에 대해서

$$F(Z_{20}) = = e_1^c(e_1^d \oplus e_3^d),\ e_3^c(e_1^d \oplus e_3^d)$$
$$= (e_1^c \boxed{c_1} e_3^c)\ (e_1^d \boxed{d_1} e_3^d)$$
$$= e_c^{-0} e_d^{-0} \qquad (22)$$

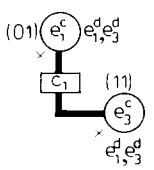

그림24. 표17에 대한 부분함수 $F(Z_{20})$
Fig. 24. Partitioned function $F(Z_{20})$ for the table17.

따라서 최종 구하는 最小化된 函數式은 다음 式(23)과 같다.

$$F_Z = (F_{Z1}\ F_{Z2})$$
$$= ((F(Z_{11})\ F(Z_{10}))\ \ (F(Z_{21})\ F(Z_{20}))) \qquad (23)$$

단계 5 : 式(23)으로 부터 PLA表와 PLA回路를 實現하면 다음 表18과 그림25와 같다.

本 論文에서 提示한 "函數 構成 方法 및 回路實現" 結果를 引用한 論文의 內容과 比較 檢討하면 다음과 같다.

例 1 의 Dhiraj K. Pradhan과 Arvind M. Patel[11]은 주어진 眞理表의 入力들을 分割하여 各各 새로운 入力으로 割當한후 行列(matrix)을 사용하여 函數를 구하고, 函數의 積과 合에 대해서는 乘算器와 加算器를 各各 構成하여 回路를 實現 하였는데 이는 많은 計算이 요구되며 函數에서 積과 合이 많을수록 게이트 갯수가 늘어나는 단점이 있다.

例 2 의 Karl S. Menger[5]는 GF(2^m)上의 元素들

표 18. PLA표
Table18. PLA table.

Number of Bit Code	product term	input Bit Code $a_1\ a_0\ b_1\ b_0\ c_1\ c_0\ d_1\ d_0$	output F_Z: F_{Z1} $F(z_{11})$	F_{Z1} $F(z_{10})$	F_{Z2} $F(z_{21})$	F_{Z2} $F(z_{20})$
4	$e_a^{10} e_b^{10}$	0 1 1 1	√			
	$e_a^{10} e_b^{'0}$	1 1 0 1	√			
	$e_a^{'0} e_b^{10}$	0 1 0 1		√		
	$e_a^{10} e_b^{'0}$	1 1 0 0		√		
	$e_c^{'0} e_d^{10}$	0 1 1 1			√	
	$e_c^{10} e_d^{'0}$	1 1 0 1			√	
3	$e_a^{-0} e_b^{10}$	− 1 1 0	√			
	$e_a^{10} e_b^{-0}$	1 0 − 1	√			
	$e_a^{0} e_b^{'-}$	1 0 0 −		√		
	$e_a^{-0} e_b^{10}$	− 1 1 1		√		
	$e_c^{-0} e_d^{10'}$	− 1 1 0			√	
	$e_c^{10'} e_d^{-0}$	1 0 − 1			√	
2	$e_a^{0'} e_b^{1-}$	− 0 1 −		√		
	$e_c^{0} e_d^{-0}$	− 1 − 1				√

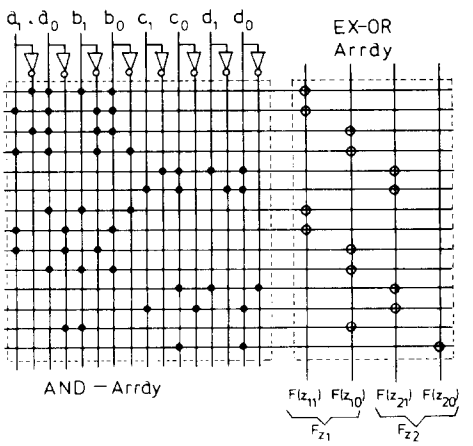

그림25. 함수F_Z의 PLA논리회로 실현
Fig. 25. PLA logic circuits realization of the function F_Z.

을 2値로 割當한후 加算모듈(PLUS module)과 乘算모듈(times module)을 사용하여 函數 構成후 이를 토대로 回路를 實現하는 方法인데 이는 表12와 같이 많은 入力이 요구 된다.

例 3 의 Iwaro Takahashi[7]는 既約多項式의 次數와 根을 各各 구하여 多項式의 係數 a_i를 결정하는 函數 構成方法인데 이는 복잡한 計算과정을 요구 한다.

例 4 에서는 有限體上에서 成立 하는 多項式에 대응되도록 Taylor급수를 展開시켜 固有行列을 산출한후

Galois스위칭 函數를 構成하였는데, 이는 行列을 계산하는데 많은 時間을 요구하며 函數의 回路實現이 구체적인 素子를 사용하여 實現되지 않았다.

위에서 設明된 기존의 函數構成方法 및 回路實現에 比하여 本 論文에서 提示한 方法은 狀態遷移圖에 圖示하여 函數를 구하므로 복잡한 計算과정을 줄일 수 있으며 PLA回路 實現은 I²L을 비롯한 여러가지의 多値論理素子를 사용하지 않고도 기존의 素子로써 多値論理回路를 實現할 수 있으며 VLSI化 할 수있는 利點이 있다. 그러나 本 論文에서 提示한 函數最小化方法은 m≤4인 경우에는 유리하나 m이 이보다 擴張될 경우에는 有效하지 못하므로 이에따른 앞으로의 硏究가 必要하다.

Ⅵ. 結 論

本 論文에서는 多値論理函數를 構成하기 위하여 有限體 GF(Pᵐ)上에서 2値化가 容易한 素數P가 2인경우의 모든 元素들을 bit code로 割當하는 알고리즘을 提示하였는데, 이는 sequential machine에서 狀態를 어떻게 效率的으로 割當하는가와 개념이 유사하다고 생각된다.

이들 bit code로 割當한 元素들간의 bit code 演算(加算, 乘算)은 MUX(multiplexer)를 사용하여 이행한 후 이를 토대로 加算器와 乘算器를 構成하였다.

本 論文에서 構成한 加算器와 乘算器의 특징을 要約하면 다음과 같다.

1) 加算器 : 彼加算元素의 bit code가 0 또는 1이냐에 따라 加算元素의 bit code가 그대로 유지 되거나 補數를 취한값이 加算後 元素의 bit code가 된다. 따라서 彼加算元素의 bit code를 加算器의 MUX 制御入力으로 사용한다.

2) 乘算器 : 彼乘算元素의 bit code와 乘算元素의 bit code 乘算時 生成되는 sign bit S_i들의 組合이 0 또는 1이냐에 따라 乘算後 bit code가 그대로 유지되거나 補數를 취한 값이 乘算後 元素의 bit code가 된다. 따라서 sign bit S_i들의 組合이 乘算器의 MUX' 制御入力으로 사용된다.

또한 本 論文에서 提示한 狀態遷移圖를 이용한 函數最小化方法의 특징은 다음으로 要約된다.

1) 狀態遷移圖에 出力이 1인 入力元素를 표시하여 最小化하므로 mod2 sum-of-product 형태를 쉽게 파악할 수 있다.

2) 元素들간의 don't care bit code 위치를 쉽게 파악할 수 있다.

3) Quine-McClusky方法은 don't care 갯수가 늘어감에 따라 cube를 계속 生成해야 하지만 狀態遷移圖를 이용한 方法은 한번에 1개 以上의 don't care를 처리 할 수 있다.

4) 重複하여 函數를 最小化할 수 없으므로 어떤 元素가 이미 한번 사용되어졌는가를 쉽게 파악할 수 있다.

5) 이 狀態遷移圖를 이용한 方法은 多入出力인 경우에도 擴張이 容易하다.

한편 回路 構成은 VLSI設計에 널리 사용하는 PLA로 實現하였다.

以上의 內容을 綜合해보면, GF(2ᵐ)上의 모든 元素들을 2値化하여 現在 사용하고 있는 디지탈시스템 및 스위칭理論에 그대로 適用시킬 수가 있으므로 기존의 컴퓨터를 GF(2ᵐ)上의 m의 擴張에 따른 多値컴퓨터로 適用할 수 있다고 展望 된다.

參 考 文 獻

[1] E.J. McClusky, *A Discussion of Multiple-Valued Logic Circuits.* The 12th Int. Symp. on Multiple-valued logic, pp. 200-205, Paris, France, 25-27, May. 1982.

[2] Stanley L. Hurst, "Multiple-valued logic-its status and its future," *IEEE Trans. Compt.*, vol. C-33, pp. 1160-1179, Dec. 1984.

[3] Tustomu Sasao, "An algorithm to derive the complement of a binary function with multiple-valued inputs," *IEEE Trans. Compt.*, vol. C-34, pp. 131-140, Feb. 1985.

[4] Vinod K. Malhotra and Robert D. Fisher, "A double error-correction scheme for peripheral system," *IEEE Trans. Compt.*, vol. C-25, pp. 105-115, Feb. 1976.

[5] Karl S. Menger, "A transform for logic networks," *IEEE Trans. Compt.*, vol. C-18, pp. 241-250, Mar. 1969.

[6] Charles C. Wang, T.K. Trung, Howard M. Shao, Leslie J. Deutsch, Jim K. Omura and Irving S. Reed, "VLSI architecture for computing multiplications and inverses in GF(2^m)," *IEEE Trans. Compt.*, vol. C-34, pp. 709-717, Aug. 1985.

[7] Iwaro Takahashi, "Switching functions constructed by Galois extension field," *Inform. Contr.*, vol. 48, pp. 95-108, Jan. 1981.

[8] Richard E. Blahut, "Algebraic field, signal processing, and error control," *Proceeding of the IEEE*, vol. 73, no. 5, May. 1985.

[9] Boonsieng Benjathrit and Irving S. Reed,

"Galois switching functions and their applications," *IEEE Trans. Compt.*, vol. C-25, pp. 78-86, Jan. 1976.

[10] In-Shek Hsu, Irving S. Reed, T.K. Trung, Ke Wang, Chiunn-Shyong Yeh and Leslie J. Deutsch, "The VLSI implementation of a Reed-Solomon encoder using Berlekamp's Bit-Serial multiplier algorithm," *IEEE Trans. Compt.*, vol. C-33, Oct. 1984.

[11] Dhiraj K. Pradhan and Arvind M. Patel, "Reed-Muller like canonic forms for multivalued functions," *IEEE Trans. Compt.*, pp. 206-210, Feb. 1975.

[12] Stephen S. Yau and Jackson Chung, "On the design of modulo arithmetic units based on cyclic groups," *IEEE Trans. Compt.*, vol. C-25, pp. 1057-1067, Nov. 1976.

[13] B.A. Laws and C.K. Rushforth, "A cellular-array multiplier GF(2^m)," *IEEE Trans. Compt.*, Short notes, pp. 1573-1578, Dec. 1971.

[14] X. Wu, X. Chen and S.L. Hurst, "Mapping of Reed-Muller coefficients and minimisation of exclusive OR-switching functions," *IEEE Proc.*, vol. 129, pp. 15-20, Jan. 1982.

[15] Richard E. Blahut, *Theory and practice of error control codes*, Addison-Wesley Publishing Com., Inc., 1983.

[16] Shu Lin, *An introduction to error-correcting codes*, Prentice-Hall Inc., Englewood Cliffs, N.J., 1970.

[17] Ian F. Blake, *Algebraic coding theory: History and development*, Dowden, Hutchinson & Ross, Inc. 1973.

[18] Garrett Birkhoff and Thomas C. Bartee, *Modern applied algebra*, McGraw-Hill, Inc., N.Y., 1970.

[19] John B. Fraleigh, *A first course in abstract algebra*, Addison-Wesley Publishing Comp., California, 1982.

[20] 李大淵, 固有行列을 利用한 Galois스윗칭 函數의 構成理論, 仁荷大學校 大學院 碩士學位請求論文, 1983.