울산대학교 공학연구논문집 제30권 2호 pp. 491~504, 울산대학교.
Journal of Engineering Research Vol. 30, No. 2, pp. 491~504, 1999
University of Ulsan.

파이프라인 (63,51) DEC BCH부호/복호기의 설계 및 제작*

이선호 · 조상복
전기 전자 및 자동화 공학부 CAD & VLSI Lab.

<요 약>

Altera Maxplus II tool을 이용하여 이동통신 등에 사용되는 (63,51) DEC (Double Error Correction) BCH 부호/ 복호기(CODEC : enCOder / DECoder)를 설계하였다. FPGA(Field Programmable Gate Array) 칩을 이용하여 설계하는데 있어 가장 비효율적이고 어려운 문제 중의 하나가 칩 내부 셀을 이용하여 메모리를 구현하는 것이다. 따라서 보다 효율적인 내부 셀 이용을 위해 메모리 사용을 최소화 시키는 것이 평요하다. 부호 / 복호기를 설계함에 있어서 가장 복잡한 연산을 수행하는 부분인 에러 위치 다항식의 계수를 찾는 부분에 대해서 메모리 사용량과 면적을 줄이기 위해 알고리즘 계산 방식을 적용하였다. 적용된 알고리즘 계산 방법은 복잡한 다항식의 계산을 대신하여 메모리를 사용하게 되는데, 이때 Galois Field(GF)의 특성을 이용한 곱셈과 나눗셈을 활용하여 '사용되는 롬의 크기를 최소화 할 수 있도록 설계하였다. 또한 상대적으로 긴 복호화 시간을 부호화 시간과 동일하게 하기 위해 복호기를 파이프라인 구조로 설계하여 부호기와 복호기 사이 부호화 시간의 불균형을 해소하였다. 본 논문의 BCH 부호 / 복호기는 Altera FPGA tool인 Maxplus II 상에서 설계하였고, 16.6Mhz로 동작하는 것이 시뮬레이션으로 확인되었다. 설계된 회로는 Max 7000 칩 2개로 구현하여 정상 동작함을 확인하였다.

A Design and Fabrication of Pipelined (63.51) DEC BCH CODEC

Sun Ho Lee · Sang Bock Cho
School of Electrical Eng. and Automation, University of Ulsan

* 본 연구는 1998년도 울산대학교 학술연구비에 의하여 조성되었음

Abstract

This paper designs (63,51) DEC (Double Error Correction) BCH (Bose -Chaudhuri - Hocquenghem) CODEC (encoder / DECoder). When this system is designed with FPGA (Field Programmable Gate Array) chip, there are many restriction. First of all, it is not efficient to use memory in FPGA chip and it is very difficult to find a coefficient of error locator polynomial. So, we suggest new algorithmic computation method. The applied new algorithmic computation method can reduce the usage of memory cell and area. Also, this algorithmic computation method use look-up-table in place of complicated polynomial calculation. The capacity of look-up-table can be reduced by using characteristics of Galois Field. Total systems are designed with pipelined architecture. So, difference of coding cycle between encoding time and decoding time can be solved. These systems are simulated and fabricated with Altera FPGA tools.(MAX+PLUS II Ver. 7.0) We can see that this chip in implemented by FLEX 10k operates as 16.6Mhz.

I .서론

C. E. Shannon이 1948년에 발표한 channel coding에 관한 정리에 따르면 정보를 부호화하여 전송하면 정보전송률이 채널 용량을 넘지 않을 때 발생되는 에러를 적절한 수준까지 줄일 수 있다고 하였다(4). 에러 정정코드 중 BCH (Bose-Chaudhuri-Hocquenghem) 코드는 이 코드가 갖는 탁월한 에러 정정 능력과, 효율적인 복호 알고리즘 때문에 다양한 응용분야에 사용되고 있으며 앞으로도 이러한 에러 정정 코드의 사용은 늘어갈 전망이다[1][2].

일반적으로 BCH 코드의 복호화 과정은 신드롬을 계산하고, 이를 이용하여 에러 위치 다항식의 계수를 찾으며, 실제 에러 위치를 찾고 이를 수정하는 4개의 단계로 구성되어 있다. 신드롬의 계산은 LFSR(LinearFeedback Shift Register)을 이용한 나눗셈 알고리즘을 이용하여 계산하고[1], 에러 위치를 찾는 부분은 Chien이 제안한 알고리즘을 이용하면 대단히 효율적이며[4],에러 수정은 X-OR 게이트를 이용하여 간단히 구현할 수있다. 문제는 계산된 신드롬으로부터 에러 위치 다항식의 계수를 찾는 것이다. 이 계수를 계산하기 위해서는 GP(2^m)에서의 나눗셈, 곱셈 및 덧셈이 필요하다.

하지만 유한 영역(finite field)에서의 연산은 일반적인 대수적 연산과는 다르기 때문에, 이것을 하드웨어로 구현하는 것은 대단히 어려운 일이다. 이를 해결하기 위해 여러 가지 알고리즘이 발표되었다[1]-[5]. 이런 방법 중 적은 하드웨어로 비교적 고속으로 구동시킬 수 있는 대표적인 것이, 생성된 신드롬을 메모리 번지로 이용하여 원하는 계수를 롬에서 읽어내는 direct ROM generation 방법과 같은 Look-up table 방법이다[7]. 하지만 이를 이용할 경우 에러 정정 능력이 2개 이상이 되면 메모리의 크기가 대단히 커지게 되므로 실제로 구현하기가 불가능한 단점이 있다[2-5].

복호화 시간이라는 관점에서 보면, 부호화 과정에 비하여 복호화 과정이 상대적으로 복

잡한 관계로, 복호화 지연시간이 길어 부호화 시간만큼 다른 시간에 복호화를 수행할 수 없다. 따라서 복호기가 부호기와 동일한 지연시간을 갖도록 하기 위해서는 복호기의 클록 속도를 높여 주어야만 한다. 기존에 보고된 BCH 및 RS 복호기의 경우는 복호화 시간이 부호화 시간에 비해 최소 2배 이상이 되어야 하는 한계를 가지고 있다[7].

본 논문에서는 메모리의 사용을 효과적으로 줄일 수 있으며 덧셈기, 뺄셈기, 비교기 등의 자원을 최대한 활용할 수 있는 알고리즘 계산 방식을 적용하였다[7]. 적용한 알고리즘 계산 방식은 GF(2^m)의 원소들에 대한 곱셈과 나눗셈 및 그들이 혼합된 연산에서 활용할 수 있으며, 면적을 최소화할 수 있는 장점을 가지고 있다. 에러 위치 다항식의 계수를 찾는데 알고리즘 계산 방식을 적용한 (63,51) DEC BCH 부호 / 복호기를 설계하였다. 그리고 파이프라인 구조로 복호기를 설계함으로써 부호기와 복호기의 지연시간을 동일하여 그 결과 부호기 / 복호기의 부호화 시간의 불균형을 해소하였다. 즉 부호화(63 사이클)와 복호화(63 사이클)가 이루어지는 시간이 동일하므로 복호기의 클록 속도를 빠르게 하지 않고도 부호화만큼 빠르게 처리할 수 있는 장점을 갖는다.

II. BCH 코드

Cyclic 코드의 일종인 BCH 코드는 $n=2^m-1$ 의 크기의 코드 워드로 구성된다[1~5] systematic 특징을 갖는 (n,k) BCH 코드의 부호화는 생성 다항식(generator polynomial)을 이용하여 다음과 같이 쉽게 구해진다.

$$c(x)=x^r i(x) \bmod g(x)+x^r i(r) \qquad (1)$$

여기서 f'(f)는 k 비트의 정보(information)를 다항식의 형태로 변환한 것이다. 위의 식을 자세히 보면 k 비트의 정보가 코드 다항식의 상위 k 차수를 차지하도록 되어 있어 다음과 같은 형태의 systematic한 코드 워드가 생성_된다.

$$c(x) = p_1+p_2x+ \ \ldots \ +p_rx^{r-1}+i_1x^r+ \ \ldots \ +i_kx^{n-1} \qquad (2)$$

식 (2)에서 하위 r (n-k) 비트는 정보 다항식(information polynomial)을 생성 다항식으로 나누었을때 나머지이며, 에러 정정을 목적으로 하는 잉여 비트이다. ,

(n,k) BCH 코드의 복호화 과정은 부호화 과정에 비해 상대적으로 복잡하며 크게 아래와 같이 4개의 단계로 나누어 진다.

1 단계.: 신드롬을 계산한다.

2 단계 : 에러 위치 다항식의 계수를 계산한다.

3 단계 : 에러 위치 계수를 계산한다.

4 단계 : 에러를 정정한다.

$r = (r_0, r_1, \ \ldots \ r_{n-1})$을 전송받은 워드라고 할 때 신드롬 S는 $(S_l, S_{l+1}, \ \ldots \ S_{l+d-2})$의 형태

로 표현되며 여기서 $S_{l+i}=\sum_{j=0}^{v} r_j \alpha^{(l+i)j}$(i=0, 1, 2, ... d-2)이다. BCH 코드의 경우는 이들 중 홀수 번째 성분들만 필요하다. 즉, $S_1, S_3, \ldots$,들이 계산되어야 한다. 1 단계는 LFSR을 이용한 나눗셈 알고리즘으로, 4 단계는 X-OR게이트를 가지고 구현이 가능하고, 3 단계는 Chien이 제안한 [4][7] 직접 복호 알고리즘을 이용하면 효율적으로 구현이 가능하다.

Ⅲ. 알고리즘 계산 방법

알고리즘 계산 방식은 $GF(2^m)$의 다음과 같은 성질을 이용한 것이다.

- Every element in $GF(2^m)$ can be represented as a power of primitive element α
- The cyclicity of power representation

$$\alpha^n = \alpha^{n+k} = \alpha^{n+2k} = \alpha^{n+3k} = \ldots \qquad (3)$$

위의 특성을 이용하면 곱셈과 나눗셈을 덧셈과 뺄셈을 이용하여 수행할 수 있다.

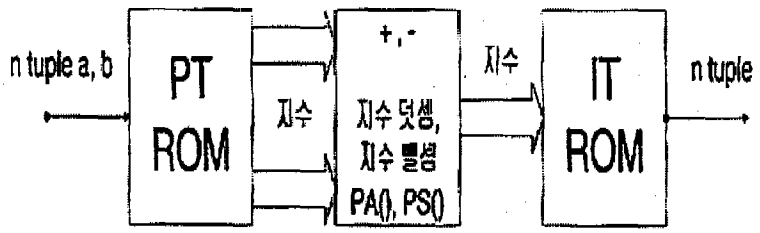

그림 1. 알고리즘 계산 방법

그림1에 알고리즘 계산 방식을 나타내었다. 우선 곱셈이나, 나눗셈이 필요한 2개의 원소들의 m tuple표현을 지수 표현으로 바꾸고, 이것을 각각 덧셈과 뺄셈을 수행한 후, 결과의 지수 표현을 다시 m tuple 표현으로 바꾸어 주면 원하는 계산 결과를 얻을 수 있다. 여기서 함수 PT( )는 m tuple 표현을 지수 표현으로의 변환해 주는 기능을 하고, 함수 IT( )는 지수 표현을 m tuple표현으로 바꾸어 주는 기능을 한다. 함수 PT( ) IT( )는 각각 $2^m \times m$ ROM을 이용하여 구현할 수 있다. 여기서 함수 PA( ) 와 PS( )는 각각 GF(2^m)에서의 지수 표현의 cyclicity를 이용한 지수 덧셈과 지수 뺄셈을 하는 것으로써 계산 결과가 유한 영역에서 벗어나게 되면 2^{m-1}을 일반 대수 연산과 동일하게 더하거나, 빼는 것이다. 표1에 GF(2^6)에서의 두 원소 "010101"과 "110101"에 대한 곱셈 (a)과 나눗셈 (b)의 예를 보였다.

앞에서 언급한 복호화 과정에서의 2 단계는 이러한 알고리즘 계산 방식을 이용해 수행할 수 있는데 우선 DEC의 경우를 예로 들어보면 다음과 같다. DEC의 경우 에러 위치 다항식은 아래와 같이 주어진다.

$$\sigma(x) = \sigma_2 x^2 + \sigma_1 x + 1 \qquad (4)$$

여기서 σ_1과 σ_2는 1 단계에서 구해진 신드롬들로 표현되는데

$$\sigma_1 = S_1$$

이고 $\sigma_2 = S_1^2 + \frac{S_3}{S_1}$로 표현된다. 그림2에 DEC (63,51) BCH코드의 경우를 direct ROM generation 방식과 알고리즘 계산 방식을 비교하여 나타내었다.

	함 수	조 건	결 과	비 고
1	PT("110101")		22	power representation of "110101"
2	PT("010101")		52	power representation of "010101"
3	addition		74 (22+52)	
		74≥63		
4	PS(84)		11(74-63)	where 63= 2^6-1
5	IT(11)		"1100011"	6 tuple representation of a×b

표 1. (a) Multiplication in GF(2^6) using algoritmic computation

	함 수	조 건	결 과	비 고
1	PT("110101")		22	power representation of "110101"
2	PT("010101")		52	power representation of "010101"
3		22<52		
	PA(22)	85(22+63)		
4	subtraction		33(85-52)	where 63= 2^6-1
5	IT(33)		"010010"	6 tuple representation of a×b

표 1. (b) Division in GF(2^6) using algoritmic computation

Direct ROM generation 방식에서는 S1과 S3을 address로 삼아 원하는 σ_2은 곧바로 구해낼 수 있으나 롬 크기가 $2^{2m} \times m$으로 너무 커지게 되어 실제적인 응용에서 바람직하지 못하다. 하지만 알고리즘 계산 방식을 이용하면 PT( )와 IT( )에 사용되는 2^m크기의 롬 2개로 가능하다. 따라서 롬의 크기만을 고려한다면 DEC의 경우 알고리즘 계산으로 얻을 수 있는 면적 이득은 2^{m-1}이 되고, (63, 51) DEC BCH 코드의 경우는 면적 이득이 32가 된다.

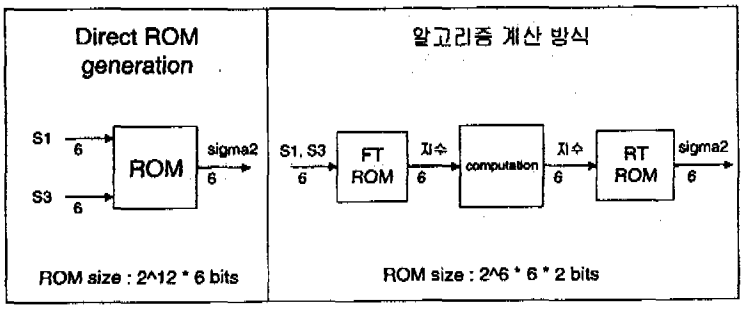

그림 2. Direct ROM generation 방식과 알고리즘 계산방식의 비교 : (63,51) DEC BCH code 의 경우

Ⅳ. (63,51) BCH 부호 / 복호기의 설계

신드롬 성분으로부터 일단 $\sigma(x)$가 결정되면 에러 위치 다항식은 다음과 같이 표현된다.

$$\sigma(x) = 1 + S_1 x + (S_1^2 + \frac{S_3}{S_1})x^2, \quad \text{여기서} S_1 \neq 0, S_3 \neq S_1^3 \qquad (5)$$

따라서 에러의 위치를 찾는 것은 $\sigma(\alpha^i) = 0$을 만족하는 α^i (for I=1,1,2....,63)을 찾는 것과 같다. 한 개의 에러가 발생한 경우를 고려해보면 2개의 신드롬 S_1과 S_3은 다음과 같다.

$$S_1 = \nu(\alpha) = \alpha^h = \beta_1 \qquad (6)$$

$$S_3 = \nu(\alpha^3) = (\alpha^h)^3 = \beta_1^3 \qquad (7)$$

결국 1개의 에러가 발생했을 경우는 $S_1^3 = S_3$과 같다. 따라서 1개의 에러가 있을 경우 $\sigma(x)$는 다음과 같다.

$$\sigma(x) = 1 + \beta_1 x = 1 + S_1 x \quad \text{여기서} S_1 \neq 0, \text{and} S_3 = S_1^3 \qquad (8)$$

만약 아무런 에러도 없었다면 복호기의 신드롬 발생 회로에서 신드롬이 0이기 때문에 이때는 에러 위치 다항식 $\sigma(x) = 1$이 된다. 이상을 요약해보면 다음과 같다.

$-\sigma(x) = 1$, $S_1 = S_3 = 0$ 오류가 없는 경우

$-\sigma(x) = 1 + S_1 x$, $S_1 \neq 0, S_3 = S_1^3$ 오류가 한개인 경우

$-\sigma(x) = 1 + S_1 x + (S_1^2 + \frac{S_3}{S_1})x^2$, $S_1 \neq 0, S_3 \neq S_1^3$ 오류가 두 개인 경우

위의 결과로부터, 에러 위치 다항식의 계수를 찾는데는 GF(2^6)에서의 곱셈과 나눗셈 및 modular 2 덧셈이 필요한 것을 알 수 있다.

1. (63,51) BCH 부호기의 설계

(63,51) BCH 부호기는 다음의 수식을 LFSR로써 구현할 수 있다.

$$c(x) = x^{12}i(x) \bmod g(x) + x^{12}i(x) \quad (9)$$

여기서 i(7)는 k 비트의 정보를 다항식의 형태로 나타낸 것이다. 그림4는 생성 다항식 (식 5)을 이용하여 설계한 (63,51) BCH 부호기의 블록 다이어그램이다. 처음 51 사이클 동안은 순차적으로 입력된 정보를 생성다항식으로 나누는 과정이 수행되고 각 레지스터에는 나머지가 채워진다. 이 때에는 sw1은 연결되어 입력된 정보와 12번째 쉬프트 레지스터의 출력의 X-OR 값을 뒤로 피드백시킨다. sw2는 처음 51 사이클 동안은 정보를 출력측으로 연결한다. 나머지는 12 비트의 패리티 비트는, 마지막 12 사이클 동안 51 비트의 정보 비트 마지막에 추가로 삽입되어 systematic 63 비트의 코드 워드를 생성하게 된다. 이때 sw1은 연결이 끊어지게 되어 피드백 루프가 끊어지게 지고, sw2는 쉬프트 레지스터의 출력을 출력측으로 연결하게 된다. 이러한 스위치의 제어 신호는 칩 내부에 컨트롤 블록에서 발생시키게 된다. 그림3은 (63,51) BCH 부호기의 전체 회로도이다.

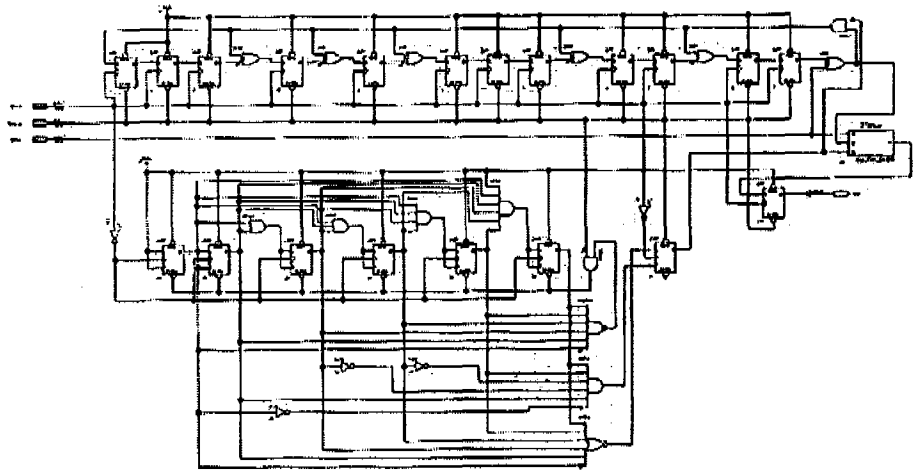

그림 3. (63,51) DEC BCH 부호기의 전체 회로도

2. (63,51) BCH 복호기의 설계

그림4는 (63,51) DEC 사양에 맞도록 설계한 파이프라인 BCH 복호기의 블록 다이어그램이다. 처음 63 사이클 동안 신드롬 S_1과 S_3가 생성되고 이를 이용하여 7사이클 (63 cycle 중 7 사이클만 소요. 나머지 사이클은 대기) 동안 σ_1, σ_2 계산이 진행된다. 순차적으로 계산을 수행하고 시스템 전체를 통제하기 위해 블록에 따라 제어 신호를 발생시키는 컨트롤러를 따로 설계하였다. 이 컨트롤러는 6 비트 동기형 카운터로 설계하였으며, 신드롬을 이용한 σ_1, σ_2 계산뿐 아니라 부호기에서 출력으로 정보 비트와 패리티 비트를 선택하는 스위

치를 제어하는 신호도 이 컨트롤러에서 발생시킨다 σ_2계산이 끝나면, 그 결과를 가지고 다음 63 사이클 동안 에러 위치 탐색 블록에서 에러의 위치를 찾아내어 에러를 정정하게 된다. 전송된 코드 워드는 63 비트 쉬프트 레지스터를 지나면서 결과적으로 126 사이클동안 지연되게 되고, 그후 63 사이클 동안 순차적으로 에러 위치탐색 블록의 출력과 X-OR를 취하게 되면서 오류를 정정하게 된다.

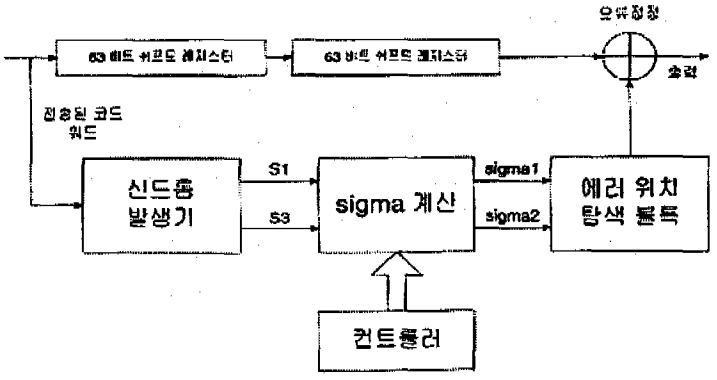

그림 4. 파이프 라인 (63,51) DEC BCH 복호기의 블록 다이어그램

2.1. 신드롬 발생기의 설계

다음은 신드롬 발생기를 설계하기 위해서 사용된 수식들이다. 생성 다항식을 인수분해하여 생기는 2개의 최소 다항식은 다음과 같다.

$$m_1(x) = x^6 + x + 1 \text{ minimal polynomial of } \alpha \tag{10}$$

$$m_3(x) = x^6 + x^4 + x^2 + x + 1 \text{ minimal polynomial of } \alpha^3 \tag{11}$$

$r(x)$을 전송된 정보의 다항식이라고 할 때 2개의 에러 정정을 위한 신드롬, S_1과 S_3은 다음과 같이 계산된다. 우선 $r(x)$을 최소 다항식 $m_1(x), m_3(x)$으로 나누었을 때의 나머지를 각각 $\gamma_1(x), \gamma_3(x)$ 라고 하면

$$\gamma_1(x) = r(x)/m_1(x) = \gamma_{10} + \gamma_{11}x + \gamma_{12}x^2 + \gamma_{13}x^3 + \gamma_{14}x^4 + \gamma_{15}x^5 \tag{12}$$

$$\gamma_3(x) = r(x)/m_3(x) = \gamma_{30} + \gamma_{31}x + \gamma_{32}x^2 + \gamma_{33}x^3 + \gamma_{34}x^4 + \gamma_{35}x^5 \tag{13}$$

구해진 $\gamma_1(x)$ 와 $\gamma_3(x)$로부터 신드롬 S_1과 S_3을 구하면

$$S_1 = \gamma_3(\alpha^3) = \gamma_{10} + \gamma_{11}\alpha + \gamma_{12}\alpha^2 + \gamma_{13}\alpha^3 + \gamma_{14}\alpha^4 + \gamma_{15}\alpha^5$$

$$S_3 = \gamma_3(\alpha^3) = \gamma_{30} + \gamma_{31}\alpha^3 + \gamma_{32}\alpha^6 + \gamma_{33}\alpha^9 + \gamma_{34}\alpha^{12} + \gamma_{35}\alpha^{15} \tag{14}$$

$$= (\gamma_{30} + \gamma_{32} + \gamma_{34}) + \gamma_{32}\alpha + \gamma_{34}\alpha^2 + (\gamma_{31} + \gamma_{33} + \gamma_{35})\alpha^3 + \gamma_{33}\alpha^4 + \gamma_{35}\alpha^5 \tag{15}$$

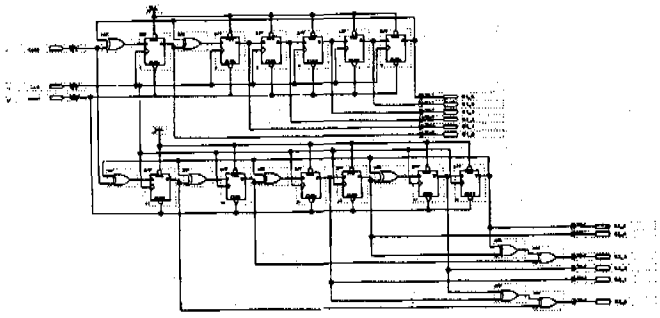

그림 5. 신드롬 발생기의 전체 회로도

이상의 유도된 수식으로부터 설계한 신드롬 생성기의 회로도를 그림5에 나타내었다. 63 사이클 동안 63 비트의 전송된 코드 워드로 부터 $\gamma_1(\alpha)$와 $\gamma_3(\alpha^3)$이 생성되고 이로부터 EXOR 게이트를 이용한 modular 2 합으로 신드롬을 생성한다.

2.2. σ_2계산 블록 설계

앞에서 언급한 것과 같이 에러 위치 다항식, $\sigma(x)=1+S_1x+[S_1^2+S_3/S_1]x^2$을 계산할 수 있도록 $\sigma_2[S_1^2+S_3/S_1]$을 찾는 것이 주된 문제인데, 이를 하드웨어로 구현하기 위해서는 GF(2^6) 에서의 제곱과 나눗셈 그리고 modular 2 덧셈이 필요한 것을 알 수 있다. PT( )와 IT ( ) 는 각각 다항식 형태를 유한 영역의 지수 형태로 변환하고, 반대로 유한 영역의 지수 형태를 다항식 형태로 변환을 수행하는 블록이다. 이 블록은 64×6 비트 롬 2개를 이용하여 구현하였다. 그림 6은 알고리즘 계산 회로에서의 계산 순서를 나타낸 것이고 그림7은 구현된 σ_2 계산 블록의 간략화 된 전체 순서도이다.

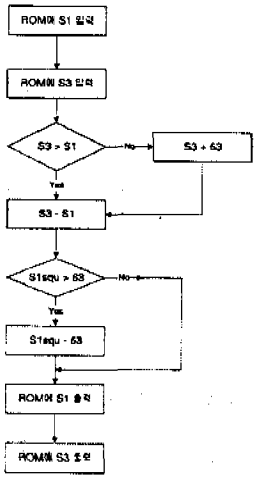

그림 6. σ_2계산 회로의 계산 순서

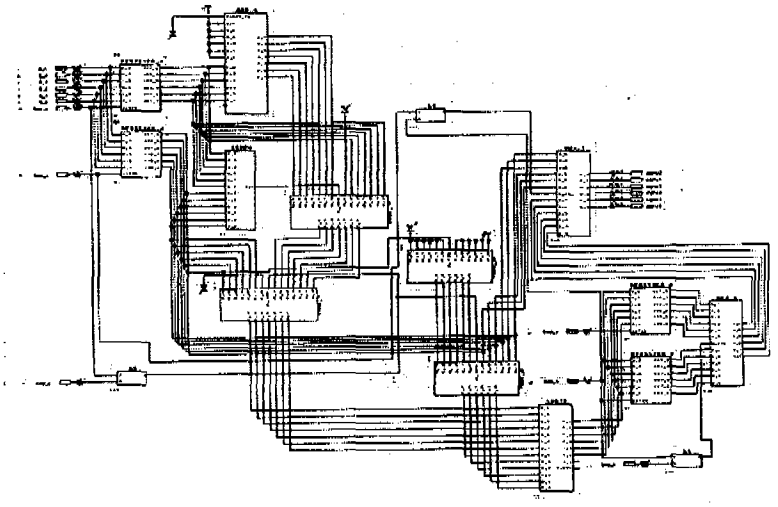

그림 7. σ_2계산 블록의 블록 다이어 그램

2.3. 에러 위치 탐색 블록의 설계

$\sigma(x)$를 구한 후, 그 다음의 중요한 복호화 과정은 실제로 에러가 발생한 위치(α^{n-j_l}, $0 \leq j_l \leq 62$)를찾는 것이다. 이 에러가 발생한 위치는 $\sigma(x)$의 근의 역수이다. Chien의 복호법에 의하면, 2개의 에러 정정이 가능한 경우에 2개의 α^1, α^3을 곱하기 위한 2개의 곱셈기가 필요하다. 초기치로 α_1, α_2을 2개의 곱샘기에 래치 시키고, 이 곱샘기가 r_1번 쉬프트 하면서 에러 위치를 찾게 된다. p번째 쉬프트 후의 t개의 쉬프트 레지스터는 $\sigma_1\alpha^p, \sigma_2\alpha^{2p}$을 갖는다. 따라서 $1+\sigma_1\alpha^p+\sigma_2\alpha^{2p}$가 0이면 α^{n-p}가 에러 위치 계수가 된다. 2개의 곱셈기를 구현하기 위해서 다음의 수식을 유도하였다.

$\sigma_1(\alpha)$을 다음과 같이 나타내면

$$\sigma_1(\alpha) = \sigma_{10} + \sigma_{11}\alpha + \sigma_{12}\alpha^2 + \sigma_{13}\alpha^3 + \sigma_{14}\alpha^4 + \sigma_{14}\alpha^4 + \sigma_{15}\alpha^5 \tag{16}$$

이로부터

$$\begin{aligned}\sigma_1(\alpha)\alpha &= \sigma_{10}\alpha + \sigma_{11}\alpha^2 + \sigma_{12}\alpha^3 + \sigma_{13}\alpha^4 + \sigma_{14}\alpha^5 + \sigma_{15}\alpha^6 \\ &= \sigma_{15} + (\sigma_{12}+\sigma_{15})\alpha + \sigma_{11}\alpha^2 + \sigma_{12}\alpha^3 + \sigma_{13}\alpha^4 + \sigma_{14}\alpha^5 \end{aligned} \tag{17}$$

마찬가지로 $\sigma_2(\alpha)$와 $\sigma_2(\alpha)\alpha^2$은 다음과 같이 표현된다.

$$\sigma_2(\alpha) = \sigma_{20} + \sigma_{21}\alpha + \sigma_{22}\alpha^2 + \sigma_{23}\alpha^3 + \sigma_{24}\alpha^4 + \sigma_{25}\alpha^5 \tag{18}$$

$$\begin{aligned}\sigma_2(\alpha)\alpha^2 &= \sigma_{20}\alpha^2 + \sigma_{21}\alpha^4 + \sigma_{22}\alpha^4 + \sigma_{23}\alpha^5 + \sigma_{25}\alpha^7 \\ &= \sigma_{24} + (\sigma_{24}+\sigma_{25})\alpha + (\sigma_{20}+\sigma_{25})\alpha^2 + \sigma_{21}\alpha^3 + \sigma_{22}\alpha^4 + \sigma_{23}\alpha^5 \end{aligned} \tag{19}$$

그림8은 에러 탐색 블록이며, 설계된 (63,51) DEC BCH 부호 / 복호기의 I/O 인터페이

스는 다양한 외부입력 클록에 따라 동기가 가능한 시리얼 인터페이스로 설계하였다. 부호기의 경우는 51 비트의 정보를 외부에서 가해지는 시리얼 클록으로 이를 부호기에 입력한다. 정보 비트는 클록의 상승하는 순간에 래치 되어 레지스터에 입력된 후 부호화 되어 63 비트의 코드워드로 변환시킨다. 이렇게 생성된 코드워드는 입력된 63개의 시리얼 클록에 동기 시켜 한 비트씩 내어주게 된다. 복호기의 경우는 63 비트의 전송된 코드 워드를 외부에서 가해지는 시리얼 클록으로 쉬프트 레지스터에 이를 복호화 시킨 후 63 비트의 코드 워드로 만들어 내며, 복호화가 끝나면 다시 외부에서 가해지는 63개의

시리얼 클록에 동기시켜 코드 워드를 한 비트씩 출력하게 된다. 그림9는 설계된 (63,51) DEC BCH 복호기의 전체 블록 다이어그램이다.

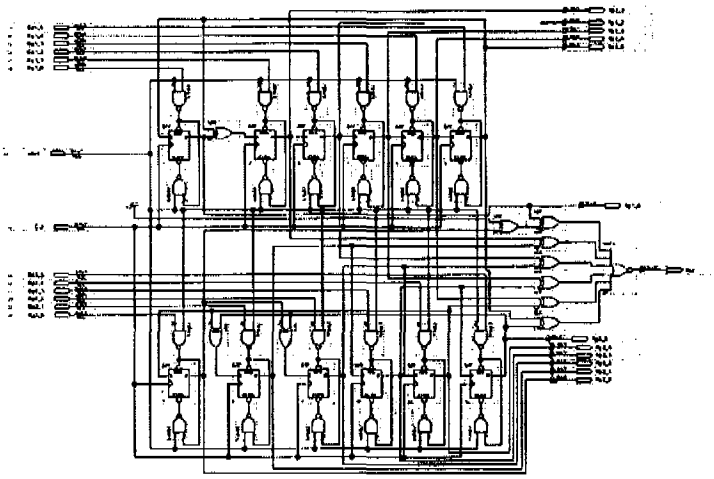

그림 8. 에러 탐색 블록의 상세 회로도

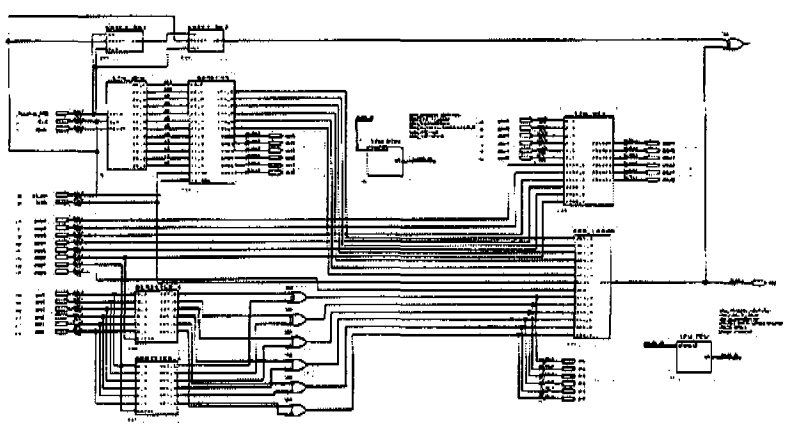

그림 9. 전체 시스템 블록 다이어그램

V. Simulation

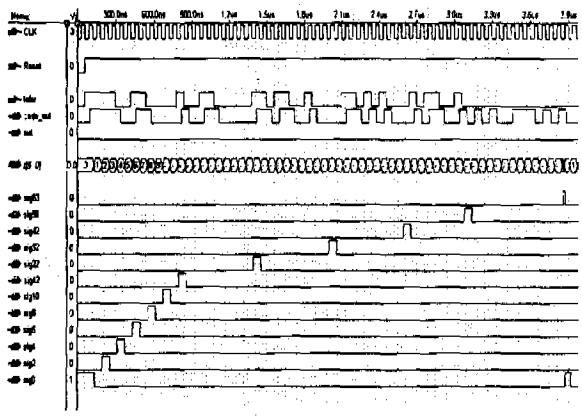

그림 10. 부호기 동작에 대한 검증 결과

설계된 (63, 51) DEC BCH 부호 / 복호기는 Altera 시뮬레이션 tool인 Maxplus2 ver8.1상에서 시뮬레이션 하였다. 시뮬레이션을 위해 Visual C++로 (63,51) DEC BCH 알고리즘을 구현하여 알고리즘의 이상유무를 확인하였고, 이 결과를 기준으로 시뮬레이션 출력과 비교하였다. 그림10은 111100110000100110000011011001000011010100010110010을 정보 비트(infor 단자)로 입력했을 때 부호기에서 출력(code_out 단자)되는 결과 값이다. 부호기의 출력 값은111001100001001100000110110010000110101000101100101010011101 01 이 출력됨을 확인할 수 있다. 클록 주파수는 16.6Mhz로 동작하고 있으며 코드 워드가 정상적으로 출력되고 있음을 알 수 있었다. 출력 아래쪽의 신호는 시스템의 제어를 위한 동기 카운터의 제어 신호 출력이다.

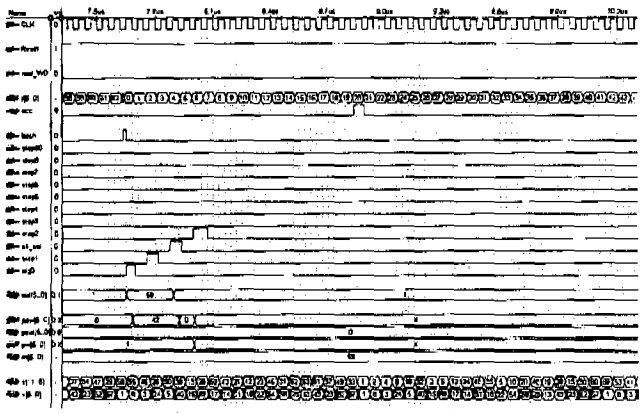

그림 11. 에러 위치 탐색 블록의 출력
(한 개의 에러가 발생한 경우)

그림11은 한 개의 에러를 전송 받았을 때 복호기의 X-OR 게이트로의 입력(ecc 단자)을 나타내고 있다. 전송된 정보에는 21번째 비트가 잘못 전송되었다. 이때 출력을 보면 정확히 21번째 클록(모드 62 카운터, t[5..0]의 출력이 '20'인 지점)에서 1이 출력되고 있음을 볼 수 있다. 그림12는 2개의 에러, 즉 7번째와 62번째 비트에 에러가 발생했을 때를 시뮬레이션한 경우를 보여주고 있다. 카운터 값이 '6'과 '61'을 나타낼 때 정확한 신호가 출력됨을 알 수 있다. 카운터 값이 36과 37사이를 넘어갈 때 순간적으로 펄스가 나타나지만 출력측에 적절한 버퍼를 삽입한다면 심각한 문제를 발생시킬 정도의 오류는 아니다.

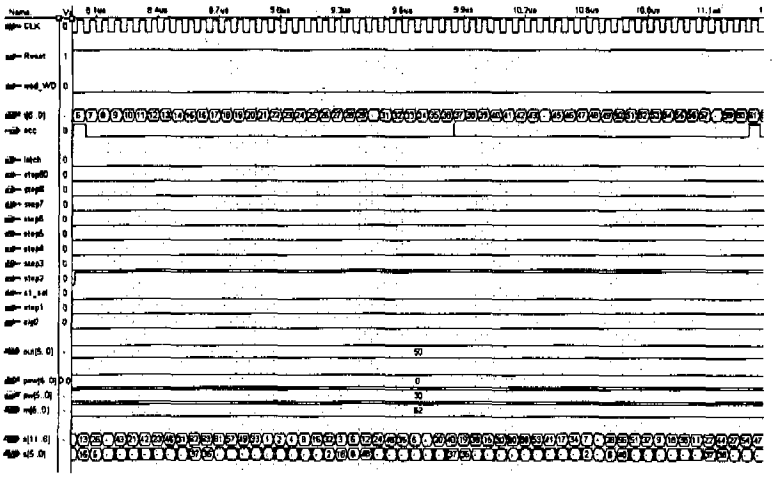

그림 12. 에러 위치 탐색 블록의 출력 (2개의 에러가 있을 경우)

VI. 결론

본 논문에서는 FPGA 칩 상에서 구현하기 까다로운 롬의 사용을 효과적으로 제한할 수 있고, 덧셈기, 뺄셈기, 비교기 등의 자원을 최대한 활용할 수 있는 GF(2^m)에서의 알고리즘 계산방식을 적용한 (63,51) BCH 부호 / 복호기를 설계하였다. 알고리즘 계산 방식을 에러 위치 다항식의 계수를 찾는데 적용함에 따라 BCH 복호기의 면적을 상당히 줄였고, 이를 FPGA 칩에서 구현하였다. 또한 상당량의 연산을 단번에 구현할 필요가 없으므로 임계경로가 길지 않아 동작 주파수를 높임으로서 고속 동작을 실현할 수 있었다. 복호기는 파이프라인 구조로 설계하여 부호화 시간과 복호화 시간을 동일하게 하였으며, 따라서 복호기의 클록 속도를 높히지 않고서도 부호기와 동일한 출력 지연 시간을 얻을 수 있었다 적용된 설계 구조와 알고리즘 계산 방식은 이중 에러 이상의 에러 정정 능력을 갖는 BCH 및 RS 코드의 복호기 설계에 효율적으로 적용이 가능하며, 복호기의 면적을 줄이는 데 기여할 수 있으리라 기대된다.

참고문헌

[1] Stephen 8. Wicker, Error Control system for Digital Communication and Storage. Prentice Hall, 1995

[2] Djimitri Wiggert, Error-Control Coding and Applications. Artech House, 1978

[3] Shu Lin, Daniel J. Costello, Jr., Error Control Coding. Prentice Hall, 1983

[4] Stephen G. Wilson, Digital Modulation and coding. Prentice Hall, 1996

[5] Peter Sweeney, Error Control Coding, Prentice Hall, 1991

[6] MAX+PLUS II getting started, ALTERA

[7] 권형준, "순차적 σ 계산 방식을 이용한 pipelined (63,51) DEC BCH 부호기기 및 복호기의 설계", '95 LG 반도체 설계 공모전 수상 논문집

[8] Hirokazu Okano and Hikeki Imai, "A Construction Method of High-Speed Decoders Using ROM's for Bose-chaudhuri-Hocquenghem and Reed-Solomon Codes," IEEE Trans. Comput., vol. C-36, No. 10, pp. 1165-1171, Oct. 1987

[9] C. C. Wang et al., "VLSI architecture for computing multiplications and inverses in GF(2^m)," IEEE Trans. Comput., vol C-34, no. 8, pp.391-403, Aug.1985.