論 文

마이크로프로그래밍 방식을 이용한 CDP용 Reed-Solomon 부호의 복호기 설계

正會員 金 泰 鎔* 正會員 金 在 均*

Design of a Reed-Solomon Code Decoder for Compact Disc Player using Microprogramming Method

Tae Yong Kim*, Jae Kyoon Kim* *Regular Members*

要 約

본 논문에서 마이크로프로그램 제어방식을 이용하여 CDP(Compact Disc Player)에서 사용되는 RS 부호(Reed-Solomon code)의 복호기를 설계하였다. 사용한 복호방법은 Newton 항등식들로 부터 얻어진 연립방정식들을 이용하여 오류위치다항식의 계수들을 구하고, C2(외부호)복호에서의 소실데이타 갯수를 확인한다. 또한 C2복호에서 소실데이타 값들을 C1(내부호)복호 결과와 신드롬들을 이용하여 구한다. 이와 같은 복호방법을 이용하여 4개의 소실정정까지 할 수 있도록 해서 오류정정능력을 높였다. 설계한 복호기는 오류정정에 필요한 GF(2^8)상에서 연산을 수행할 수 있는 복호연산기와 프로그램 ROM을 가지고 있는 복호제어기 및 마이크로명령어(microinstruction)들로 구성된다. 마이크로명령어들을 이용하여 RS 부호의 복호 알고리즘을 프로그램할 수 있으며, 성능향상이나 다른 용도에 사용하기 위해서는 프로그램 ROM만 바꾸면 가능하므로 간편하다. 본 논문에서 설계한 복호기는 Verilog HDL의 Logic Level Modeling을 이용하여 구현했으며, 설계된 복호기에서 각 마이크로명령어들은 14비트(=1 word)이고, 프로그램 ROM의 크기는 360 word이다. 또한 C1과 C2를 모두 복호하는데 걸리는 최대시간은 424 clock-cycle이다.

ABSTRACT

In this paper, an implementation of RS(Reed-Solomon) code decoder for CDP(Compact Disc Player) using microprogramming method is presented. In this decoding strategy, the equations composed of Newton's identities are used for computing the coefficients of the error locator polynomial and for checking the number of erasures in C2(outer code). Also, in C2 decoding the values of erasures are computed from syndromes and the results of C1(inner code) decoding. We pulled up

*韓國科學技術院 電氣·電子工學科
Dept. of Electrical and Electronics Eng., KAIST
論文番號 : 93-151

the error correctability by correcting 4 erasures or less. The decoder contains an arithmetic logic unit over GF(2^8) for error correcting and a decoding controller with programming ROM, and also microinstructions. Microinstructions are used for an implementation of a decoding algorithm for RS code. As a result, it can be easily modified for upgrade or other applications by changing the programming ROM only. The decoder is implemented by the Logic Level Modeling of Verilog HDL. In the decoder, each microinstruction has 14-bits(=1 word), and the size of the programming ROM is 360 words. The number of the maximum clock-cycle for decoding both C1 and C2 is 424.

I. 서 론

CD(Compact Disc), DAT(Digital Audio Tape) 등의 디지탈 오디오들은 오류데이타의 검출 및 정정을 위하여 GF(2^8)상의 단축형 RS부호(Reed-Solomon code)를 사용한다. GF(2^8)상의 원소들은 모두 256개이기 때문에 8비트로 모든 원소들의 표현이 가능하다. CD에서는 각각 4개의 검사심볼이 부가되는 내부호(inner code) C1(32,28), 외부호(outer code) C2(28,24)를 사용하고[1], DAT에서는 4개의 검사심볼이 부가되는 내부호 C1(32,28)과 6개의 검사심볼이 부가되는 외부호 C(32,28)를 사용한다[2]. CD, DAT 모두 내부호와 외부호를 2중으로 배열함과 동시에 순차적인 데이타를 서로 인터리빙(interleaving)시키기 때문에 산발오류(random error)와 연집오류(burst error)를 아주 효과적으로 정정할 수 있다.

2중 부호화된 RS 부호(CIRC, Cross-Interleaved Reed-Solomon Code)의 복호는 그림 1과 같이 C1, C2복호를 반복해서 수행한다. 따라서 C2복호에서는 C1복호의 결과(C1 flag)를 이용하여 소실정정(erasure correction, 오류위치를 알고 있는 오류데이타의 정정)을 행할 수가 있다. CDP(Compact Disc Player)의 경우 C1복호에서는 최대로 2개까지의 오류데이타(error data, 오류가 발생된 위치와 오류의 값을 모두 모르는 데이타)를 정정할 수 있고, C2복호에서는 C1 flag를 이용하여 최대로 4개까지의 소실데이타(erasure data, 오류가 발생된 위치는 알고 있으나 오류의 값을 모르는 데이타)를 정정할 수 있다. 이런 복호방법에는 Super Strategy[3]와 T.Arai 등이 제안한 방법[4]등이 있다. Super Strategy는 C1복호에서 2개의 오류데이타를 정정하고, C2복호에서는 C1 flag를 확인하는 방법으로 2개까지의 오류데이타를 정정한다. T.Arai 등이 제안한 방법에서는 C1복호에서 2개의 오류를 정정하고, C2복호에서는 최대 1개의 오류와 2개의 소실데이타를 동시에 정정한다. 이상의 두가지 방법들은 모두 C2복호에서 C1 flag를 이용하여 오류정정능력을 높이고 있지만, C2복호에서 3개 혹은 4개의 소실정정을 행하지 않고 있다.

RS 부호의 오류정정을 위해서는 오류위치다항식의 계수들을 신드롬(syndrome)들로 부터 구해야만 하는데, 이를 위한 방법들로 여러가지가 제안되어 있다. Berlekamp, Massey의 반복 계산에 의한 방법[5,6]을 비롯하여 Welch, Scholtz의 Continued fraction에 의한 방법[7], Blahut의 변환기술에 의한 방법[8]등이 있는데, 이 방법들은 여러가지의 변수와 함께 반복적인 계산으로 많은 계산이 요구되어지기 때문에 실제구현에 적합치 않다. Gorenstein-Zierler의 계산방법[9]은 Newton 항등식(Newton's identities)들에 의해서 만들어지는 행렬을 계산하는 것으로 오류갯수가 적을 경우 비교적 간단하다.

이상의 복호를 수행하기 위한 복호기중에서 Berlekamp, Massey등이 제안한 FSR(Feedback Shift Register)에 의한 방법[5,6]은 하드웨어가 복잡하고, 많은 계산량 때문에 복호시간이 너무 길어진다. 또한 이만영, 김창규가 제안한 복호기[10]는 복호속도는 매우 빠르지만 ROM으로 구현된 9개의 승산기와 2개의 역원계산기 때문에 하드웨어가 대단히 커져서 실제구현에 문제가 있다. 이상의 두가지 방법들은 실제 오

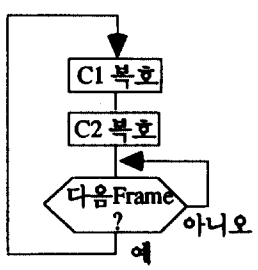


그림 1. 2중으로 중첩된 RS 부호의 복호.
Fig. 1. Decoding for crossinterleaved Reed-Solomon code (CIRC).

류정정에서 일어나는 오판확률(misdetection probability)등에 대한 고려를 하지 않기 때문에 정정된 데이타의 오류률(symbol error rate)이 높을 것으로 생각된다.

T.Arai등이 제안한 복호기[4]는 프로그램에 의한 방법으로 설계되어 있는데, 이 방법은 성능개선이 용이하고 체계적인 구조를 갖고 있다. 참고문헌(4)에서 보면 GF(2^8)상의 곱셈과 나눗셈을 지수들의 덧셈과 뺄셈으로 대신하기 위해서 각각 256×8 비트 크기의 log-ROM(GF(2^8)상의 원소 α^i, i=0, 1, 2, ..., 254들을 지수값, i로 바꾸는 ROM) 및 antilog-ROM(i를 GF(2^8)상의 원소 α^i, i=0, 1, 2, ..., 254로 바꿔주는 ROM)과 함께 8비트 가산기를 사용했는데, 이들은 모두 대단히 큰 하드웨어들이다. 또한 데이타를 저장하기 위한 메모리와 복호된 데이타가 오류인지/아닌지의 상태를 표시하는 flag의 저장을 위한 메모리를 분리했기 때문에 어드레스 구조가 복잡할 것으로 생각된다. 이러한 구조를 갖는 하드웨어에서 subroutine을 사용할 수 있도록 한다면 프로그램 ROM의 크기를 많이 줄일 수 있을 것이며, flexibility 측면에서도 많은 장점이 있을 것이다.

본 논문에서는 오류위치다항식(error locator polynomial)의 계수들을 Newton항등식들로 구성된 방정식들을 풀어서 구하며, 소실정정에서도 C1 flag가 있는 소실데이타의 갯수를 Newton항등식을 이용하여 확인함으로서 정확한 오류정정이 되도록 했다. 또한 C2복호에서 4개의 소실정정까지 할 수 있도록해서 오류정정능력을 높였고, 오류위치에 해당되는 오류값들은 신드롬들의 연립방정식을 풀어서 구했다. 아울러 마이크로프로그래밍 제어방법을 이용하여 CDP에서 사용할 수 있는 RS 부호의 복호기를 설계했는데, 전체적으로 복호연산기와 이를 제어하기 위한 복호제어기로 구성되어 있다. 이상의 복호기는 Verilog HDL의 Logic Level Modeling[11]을 이용하여 구현했다.

마이크로프로그래밍 제어방법을 이용한 설계방법은 기본적인 기능블럭들만으로 하드웨어를 구성한 다음, 소프트웨어로 원하는 알고리즘을 구현하기 때문에 성능개선이 대단히 용이하다. 또한 하드웨어가 간단하면서도 체계적인 구조를 갖기 때문에 IC화에 적합하며, 시스템을 시험할 경우에도 조직적이고도 체계적인 검증이 가능해서 문제를 쉽게 찾을 수 있다. 따라서 본 논문에서 설계한 복호기는 프로그램의 변경만으로 DAT, CD-ROM(Compact Disc ROM) 등에도 적용시킬 수 있다.

Ⅱ장에서는 복호원리와 본 논문에서 제안한 복호방법에 관하여 논하고, Ⅲ장에서는 설계한 복호연산기에 대해서 설명하며, Ⅳ장에서는 마이크로프로그램에 의한 복호제어장치의 설계에 대해서 설명한다. 마지막으로 Ⅴ에서 기존 복호기들과의 비교, 검토와 함께 결론을 맺고자 한다.

Ⅱ. RS부호의 복호방법

1. 오류정정 방법

GF(2^m)은 모두 2^m개의 원소를 갖는 유한체이며, 원소들은 {0, 1, α, α^2, ..., α^{2^m-2}}이다. GF(2^m)상의 RS부호는 (2^m-1)개 심볼들이 한개의 블록을 구성하며, 단축형 RS부호[12]는 n($<2^m-1$)개의 심볼들로 한개의 블록이 구성된다. GF(2^8)상의 단축형 RS부호를 사용하는 CD의 경우, C1부호의 n은 32이고, C2부호의 n은 28이다[1]. n개의 심볼로 구성된 부호블럭을 다항식으로 표시하면 다음과 같고,

$$c(x)=c_{n-1}x^{n-1}+c_{n-2}x^{n-2}+\ldots+c_1x+c_0 \qquad (1)$$

여기서 c_i, $0\le i\le n-1$는 GF(2^8)상의 어떤 원소이다. 부호다항식 c(x)는 생성다항식 g(x)를 이용하여 k개의 심볼을 갖는 정보다항식 d(x)에 (n-k)개의 검사심볼을 부가한 것이며, 이것을 (n,k) RS부호라 한다. 이와 같이 만들어진 c(x)는 g(x)를 인수로 가지며 다음의 관계를 갖는다.

$$c(x)=M(x)g(x) \qquad (2)$$

그림2와 같이 부호다항식 c(x)를 전송했을 때, 오

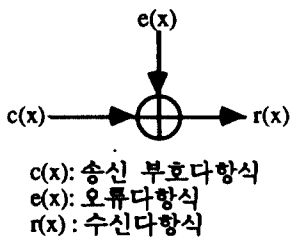


그림 2. 송신부호와 오류의 관계.
Fig. 2. Code transmission and error.

류다항식 e(x)가 발생하면 수신다항식 r(x)는 다음과 같다.

$$r(x) = c(x) + e(x) \tag{3}$$

(n,k) RS부호가 t개의 오류까지 정정가능하기 위해서는 부호간의 최소거리가 $d_{min} \geq 2t+1$ 이어야 한다. 만일 v개의 오류가 발생했다면 e(x)는 다음과 같고,

$$e(x) = e_{j1} x^{j1} + e_{j2} x^{j2} + \ldots + e_{jv} x^{jv}, \quad 0 \leq jv \leq n-1, \quad 1 \leq v \leq t \tag{4}$$

생성다항식 g(x)를 다음과 같이 두면

$$g(x) = \prod_{i=0}^{n-k-1} (x + \alpha^i) \tag{5}$$

식 (2)로부터 $c(\alpha^i) = 0$, $0 \leq i \leq n-k-1$이기 때문에 신드롬 s_i는 다음과 같다.

$$s_i = r(\alpha^i), \quad 0 \leq i \leq n-k-1 \tag{6}$$

$$s_i = e(\alpha^i) = e_{j1}(\alpha^{j1})^i + e_{j2}(\alpha^{j2})^i + \ldots + e_{jv}(\alpha^{jv})^i, \quad 0 \leq i \leq n-k-1 \tag{7}$$

오류위치 α^{jv}를 구하기 위하여 오류위치다항식 $\sigma(x)$를 다음과 같이 정의하면

$$\sigma(x) = x^v + \sigma_1 x^{v-1} + \sigma_2 x^{v-2} + \ldots + \sigma_{v-1} x + \sigma_v \tag{8}$$

$$= (x+\alpha^{j1})(x+\alpha^{j2})\ldots(x+\alpha^{jv}) \tag{9}$$

이때 $\sigma(x)$의 계수들은 다음과 같다.

$$\sigma_1 = \alpha^{j1} + \alpha^{j2} + \ldots + \alpha^{jv} \tag{10}$$

$$\sigma_2 = \alpha^{j1}\alpha^{j2} + \alpha^{j2}\alpha^{j3} + \ldots + \alpha^{jv-1}\alpha^{jv} \tag{11}$$

$$\ldots\ldots\ldots\ldots$$

$$\sigma_v = \alpha^{j1}\alpha^{j2}\ldots\alpha^{jv} \tag{12}$$

신드롬 s_i, $0 \leq i \leq n-k-1$와 σ_j, $1 \leq j \leq v$의 관계는 Newton 항등식에 의해서 다음과 같은 식으로 나타낼 수 있다.

$$s_i + \sigma_1 s_{i-1} + \ldots + \sigma_{v-1} s_{i-v-1} + \sigma_v s_{i-v} = 0, \quad v \leq i \leq n-k-1 \tag{13}$$

식 (13)에 의해서 주어지는 v개의 Newton 항등식들로 부터 σ_j, $1 \leq j \leq v$을 계산할 수 있고, 이와 같이 얻어진 $\sigma(x)$를 0으로 하는 해가 바로 오류위치 α^{j1}, α^{j2}, ..., α^{jv}들이다.

CD에서 사용하는 생성다항식 g(x)는 다음과 같고[1],

$$g(x) = \prod_{i=0}^{3} (x + \alpha^i) \tag{14}$$

오류데이타가 없으면 신드롬 s_i, $0 \leq i \leq 3$들은 모두 0이고, 1개의 오류가 α^{j1}, $0 \leq j1 \leq n-1$에서 발생하면 $s_i/s_{i-1} = \alpha^{j1}$, $0 \leq i \leq 3$가 된다. 2개의 오류가 α^{j1}, α^{j2}, $0 \leq j1, j2 \leq n-1$에서 발생했다고 가정했을 때 신드롬은 다음과 같다.

$$s_i = e_{j1}(\alpha^{j1})^i + e_{j2}(\alpha^{j2})^i, \quad 0 \leq i \leq 3 \tag{15}$$

이 때 오류위치 다항식은 다음과 같은 식으로 나타낼 수 있다.

$$\sigma(x) = (x+\alpha^{j1})(x+\alpha^{j2}) \tag{16}$$

$$= x^2 + \sigma_1 x + \sigma_2 \tag{17}$$

식 (15)의 신드롬과 오류위치다항식의 계수 σ_1, σ_2는 식 (13)의 Newton 항등식으로 부터 다음과 같이 되고,

$$s_2 + s_1\sigma_1 + s_0\sigma_2 = 0 \tag{18}$$

$$s_3 + s_2\sigma_1 + s_1\sigma_2 = 0 \tag{19}$$

따라서 σ_1, σ_2는 다음과 같다.

$$\sigma_1 = \frac{s_0 s_3 + s_1 s_2}{s_0 s_2 + s_1^2} \tag{20}$$

$$\sigma_2 = \frac{s_1 s_3 + s_2^2}{s_0 s_2 + s_1^2} \tag{21}$$

이와 같이 얻어진 $\sigma(x)$에 $x = \alpha^i$, $0 \leq i \leq n-1$를 대입하면 $\sigma(\alpha^{j1})$, $\sigma(\alpha^{j2})$이 각각 0이 되며, 따라서 α^{j1}, α^{j2}

가 오류위치가 된다.

오류위치를 알면 오류값들은 신드롬들의 연립방정식을 이용하여 쉽게 계산할 수 있으며, 이것을 소실정정이라 한다. 예를 들면 3개의 오류일치 α^{j1}, α^{j2}, α^{j3}를 알 때, s_0, s_1, s_2에 의한 연립방정식을 풀면 e_{j1}, e_{j2}, e_{j3}는 다음과 같다.

$$e_{jk}=\frac{s_2+s_1A_k+s_0B_k}{\alpha^{2jk}+\alpha^{jk}A_k+B_k},\quad k=1,2,3 \tag{22}$$

$$A_1=\alpha^{j2}+\alpha^{j3},\quad A_2=\alpha^{j3}+\alpha^{j1},\quad A_3=\alpha^{j1}+\alpha^{j2} \tag{23}$$

$$B_1=\alpha^{j2}+\alpha^{j3},\quad B_2=\alpha^{j3}+\alpha^{j1},\quad B_3=\alpha^{j1}\alpha^{j2} \tag{24}$$

또한 소실정정에서 소실데이타가 C1 flag의 수와 일치하는지를 검사하는데 Newton항등식이 이용될 수 있다. 예를 들면 C1 flag가 α^{j1}, α^{j2}, α^{j3}에 있다면, 오류위치다항식의 계수 σ_1, σ_2, σ_3는 다음과 같고,

$$\sigma_1=\alpha^{j1}+\alpha^{j2}+\alpha^{j3} \tag{25}$$

$$\sigma_2=\alpha^{j1}\alpha^{j2}+\alpha^{j2}\alpha^{j3}+\alpha^{j3}\alpha^{j1} \tag{26}$$

$$\sigma_3=\alpha^{j1}\alpha^{j2}\alpha^{j3} \tag{27}$$

α^{j1}, α^{j2}, α^{j3}를 제외한 다른 위치에 오류가 없다면 식 (13)의 Newton항등식으로 부터 다음이 성립된다.

$$s_3+s_2\sigma_1+s_1\sigma_2+s_0\sigma_3=0 \tag{28}$$

따라서 식 (28)이 만족되면 3개 이하의 오류가 α^{j1}, α^{j2}, α^{j3}에 존재한다.

2. 전체적인 복호과정

2중으로 중첩된 RS부호는 C1복호와 C2복호를 순차적으로 행한다. 그런데 복호방법에 있어서 오류정정 능력 못지않게 중요한 것이 오류검출 능력이다. 정정 불가능한 오류들이 검출되지 않고 정상적인 데이타로 처리된다면 오디오에서는 shot잡음 등을 일으킨다. 본 논문에서는 C1에서 2개의 오류까지 정정하고 C2에서 4개의 소실정정까지 가능하도록 했다.

가. C1복호

C1부호의 부호간 최소거리가 5이기 때문에 C1복호에서 오류정정이 잘못 되어지는 경우는 다음과 같다.

(1)3개의 오류를 2개의 오류로 오판하거나,

(2)4개의 오류를 1개의 오류로 오판하거나,

(3)5개의 오류를 오류가 없는 것으로 오판

(1),(2),(3)의 경우를 가정해서 식을 풀어보면, (1)에 대한 확률은 255^{-2}, (2)에 대한 확률은 255^{-3}, (3)에 대한 확률은 255^{-4}이다. 32개의 심볼로 구성된 C1부호에서 3개, 4개, 5개의 오류가 발생할 확률과 (1),(2),(3)을 결합하면 C1부호에서의 오류정정이 잘못되어질 확률은 표1과 같은 값(Ps는 심볼오류률, symbol error rate)을 갖는다. 표1로 부터 3개 이상의 오류가 2개의 오류로 판정될 확률이 대단히 높다는 것을 알 수 있으며, 이에 대한 대비는 필연적이다. 이것을 고려한 C1복호방법이 그림3이며, C1 flag는 표1과 같이 생성된다.

표 1. C1복호에서 오판확률.
Table 1. Misdetection Probability for C1 decoding.

오류정정 갯수	C1 flag	오 판 확 률
오류 없음	0	7.3×10^{-6} Ps^5
1개 오류	0	9.4×10^{-3} Ps^4
2개 오류	1	4.8 Ps^3
3개 오류	1	465 Ps^3

*Ps : Symbol Error Rate

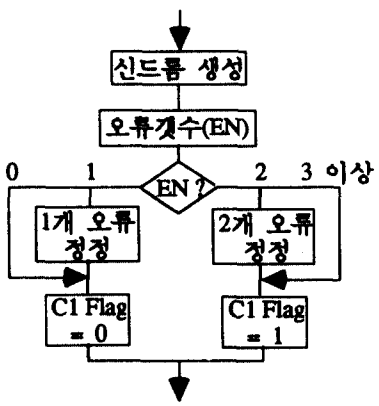


그림 3. C1 복호 방법.
Fig. 3. C1 decoding flowchart.

그림3의 C1복호방법은 먼저 C1부호 32개의 심볼을 읽어서 신드롬을 생성시키고, 신드롬을 이용하여 오류갯수를 판정한다. 오류갯수에 따라 오류정정을 행한 다음 정정 결과에 따라 C1 flag를 생성시킨다. 오류가 1개 이하인 경우에는 오류정정을 행한 후 C1

flag를 0으로하고, 2개인 경우에는 오류정정을 행한 후 오류가 3개 이상이어서 정정이 불가능한 경우와 마찬가지로 C1 flag를 1로 한다. 이와같은 flag생성 방법은 표1에 주어진 오판확률에 근거한 것으로 C1 flag가 1인 데이타의 갯수 및 위치 등이 C2복호에서 활용되어진다.

나. C2복호

C2복호의 입력심볼들은 이미 C1복호를 거쳤기 때문에 모든 심볼들이 각각의 C1 flag를 가지고 있으며, 이를 위한 C2복호 방법은 그림4와 같다. C2부호 28개 심볼을 읽어서 신드롬을 생성시키고, 28개 심볼에 대한 C1 flag를 읽어서 flag가 1인 것을 계산한다. 신드롬을 이용하여 오류갯수를 판정해서 오류가 1개 이하이면 오류정정을 행하고 C2 flag를 0으로 한다. 2개 이상의 오류로 판정될 경우에는 C1 flag의 수에 따라 처리를 달리한다. C1 flag의 수가 5개 이상이면 정정불능으로 간주해서 C2 flag에 C1 flag를 그대로 복사하고, 1개 이하이면 정정불능의 오류로 간주해서 C2 flag를 1로 한다. C1 flag가 1인 것이 2개 이상 4개 이하일 경우에는 C1 flag가 1인 위치의 데이타를 오류로 간주해서 소실정정을 행하고, 그 결과를 검사해서 올바르면 C2 flag를 0으로 하고, 그렇지 않으면 정정불능의 오류로 생각해서 C2 flag를 1로 한다.

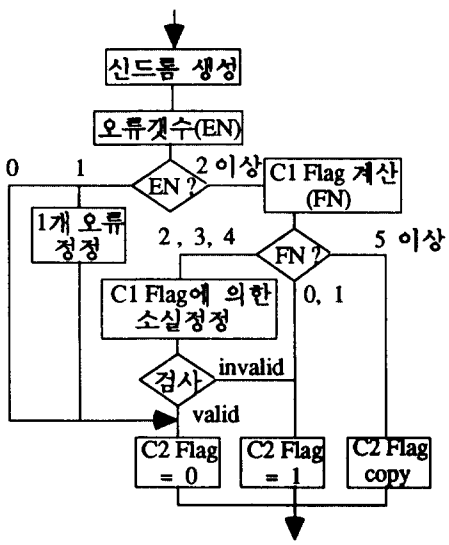


그림 4. C2 복호 방법.
Fig. 4. C2 decoding flowchart.

C2복호에서 3개의 소실정정이 일어나는 경우는 28개의 C2 데이타 중에서 C1에서 3개 이상의 오류로 판정된 데이타가 3개 이상일 경우로써, 이때의 확률은 $P_3 = {}_{27}C_2({}_{31}C_2)^3 Ps^9$이며, 약 $3.529\times10^{10} Ps^9$가 된다. 마찬가지로 4개의 소실정정이 일어날 확률, P_4는 $1.368\times10^{14} Ps^{12}$정도이다. 그림5는 C2복호에서 어떤 심볼이 3개의 소실정정 혹은 4개의 소실정정에 의해서 정정되어질 확률 P_3과 P_4의 Ps에 대한 그래프인데, 심볼오류율, Ps가 10^{-3}일때 P_3과 P_4는 각각 3.529×10^{-17}, 1.368×10^{-22}정도이다. 따라서 C2복호에서 C1 flag를 이용하여 3개와 4개의 소실정정을 행함으로써 오류정정능력이 향상됨을 알 수 있다.

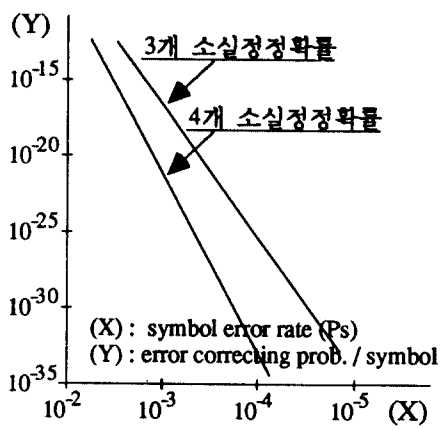


그림 5. 소실정정에 의한 오류정정확률.
Fig. 5. Error correcting probability by erasure correction.

Ⅲ. 복호연산기(ECU_ALU) 설계

일반적으로 2중으로 중첩된 RS부호의 복호기는 대단히 복잡한 구조를 갖는데 비해 본 논문에서는 최대한 간단한 하드웨어 구조를 갖도록 하고 마이크로 프로그래밍으로 여러가지 복잡한 기능을 구현할 수 있도록 설계했다. 간단한 하드웨어 구조를 가지면 제어에 필요한 명령어의 수가 적어서 프로그램을 짜는 것이 용이해질 뿐만 아니라, 프로그램을 이해하기가 훨씬 쉽기 때문에 성능개선을 위한 프로그램의 변경도 쉬워진다. 반면에 프로그램이 약간 길어지는 단점이 있기는 하지만 복잡한 하드웨어로 구성하는 것에 비교할 때 간단한 하드웨어가 갖는 잇점이 훨씬 크기 때문에 이와같은 방법을 선택했다.

복호연산기의 구조는 그림6과 같다. 앞에서 설명한 바와 같이 2중으로 중첩된 RS부호를 복호하기 위한 최소한의 기능블럭들은 신드롬 생성기, $GF(2^8)$상의 연산을 위한 $GF(2^8)$ALU, flag처리를 위한 flag처리기, 복호 중간에 발생하는 데이타를 저장하기 위한 임시 메모리와 데이타 메모리(DMEM)의 메모리의 데이타를 읽거나 쓸 때 필요한 어드레스를 발생시키기 위한 어드레스 발생기 등이 필요하다. DMEM은 CD로부터 복조된 데이타를 저장하거나 정정된 데이타와 flag 등을 저장하며, zero 검출기는 조건판단을 위해서 필요하다. 다음에서 복호연산기를 구성하는 각 블럭들의 기능 및 구조에 대해서 구체적으로 설명한다.

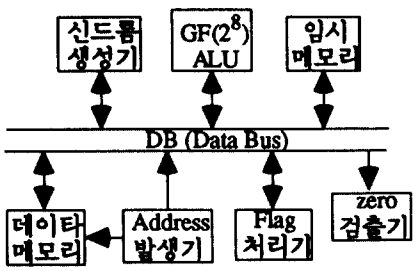


그림 6. 본 논문에서 설계한 복호 연산장치의 구조.
Fig. 6. ECU_ALU block diagram proposed by this paper.

1. 신드롬 생성기

신드롬 생성기는 그림7과 같으며, DB(Data Bus)로 부터 수신데이타를 받아서 신드롬 s_i, $0 \le i \le 3$를 생성한 다음 신드롬 레지스터, SR(SiR, $0 \le i \le 3$)에 저장하며, SR의 내용을 데이타 버스로 내보낼 수 있다. 신드롬생성기는 DMEM+SR*AP $\longrightarrow$ SR (AP는 α^i, $0 \le i \le 3$)의 기능을 갖기 때문에 $\sigma(x)$의 근을 찾는데 이용할 수 있다. 예를 들면 식(13)의 근을 구할 경우 S2R의 초기값은 1, S1R의 초기값은 σ_1, S0R의 초기값은 σ_2로 하고 SR*AP $\longrightarrow$ SR을 j번 반복 수행했을 때 S0R+S1R+S2R=0이 되면 α^j, $0 \le j \le n-1$ (C1일 때 n=32, C2일 때 n=28)가 $\sigma(x)$의 근으로 오류위치가 된다.

2. $GF(2^8)$ ALU

$GF(2^8)$ ALU 블럭은 $GF(2^8)$의 원소들끼리의 사칙연산을 수행하는 블럭으로 그림8과 같다. 나눗셈을

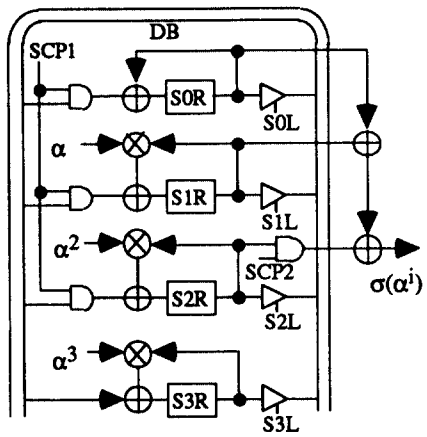


그림 7. 신드롬 생성기.
Fig. 7. Syndrome generator.

위하여 α^i를 α^{-i}, $0 \le i \le 254$로 변환할 수 있는 역원 ROM(nROM)이 있고, 곱셈기의 입력 레지스터 MAR, MBR, 덧셈기의 입력 레지스터 ABR 등이 있다. 또한 곱셈기의 출력은 덧셈기의 입력이 되고, 덧셈기의 출력은 accumulator를 두지 않고 바로 DB로 출력되게 하였다. 그림8의 MCON(Multiplier CONtrol)과 ACON(Adder CONtrol)은 각각 승산기와 가산기를 제어하는 신호로써, MCON은 MBR를 1로 만드는데 사용되고, ACON은 ABR를 0으로 만드는데 사용되어진다. 따라서 MAR*MBR+0, MAR*1+ABR, MAR*MBR+ABR 등의 기능을 수행할 수 있으며, nROM을 통해서 MAR로 들어가는 것을 nMAR로 나타내고, 이때는 나눗셈이 된다.

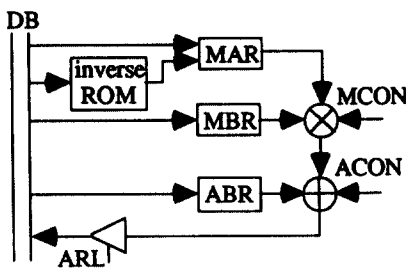


그림 8. $GF(2^8)$ ALU 블럭.
Fig. 8. $GF(2^8)$ ALU block.

3. Flag 처리기

Flag에 사용되어질 0 혹은 1을 DB로 내보낼 수 있는 기능과 C1 flag를 읽어서 C2 flag에 쓸 수 있는 flag 복사기능을 수행할 수 있다. FLAG(0), FLAG(1)은 각각 flag처리기가 상수 0, 1을 발생시킴을 뜻한다. C2복호를 수행할 때 오류정정에서 구해진 오류위치 α^{j1}, α^{j2}의 C1 flag를 읽어서 저장할 수 있는 레지스터 C1FR1, C1FR2(C1FR)가 있고 이 신호는 CON_SEL (CONdition SELection) 블럭으로 입력된다.

4. 어드레스 발생기

어드레스 발생기는 그림9와 같다. C1, C2복호시 신드롬 생성을 위한 데이타를 읽기나, C2복호에서 C1 flag를 읽거나, C2 flag를 쓰거나, C1 flag를 C2 flag로 복사할 때는 5비트 counter(ACNT)를 이용해서 어드레스를 발생시키고 deinterleaving ROM(DIR)을 통하여 DMEM의 번지를 지정한다. 오류위치 레지스터에는 $\sigma(x)$의 근을 저장하기 위한 i1R, i2R(iR)이 있고, C1 flag가 1인 C2데이타의 위치를 저장하기 위한 ER1~ER4(ER)가 있으며, 이들은 shift register 형태로 연결된다. 오류위치 레지스터에 있는 내용들은 α^i의 i에 해당되므로 i를 α^i로 변환하는 ROM (iROM)을 통해서 DB로 나가는데, 이것은 GF(2^8) ALU에서의 모든 연산이 α^i를 근거로 하기 때문이다. 또한 오류위치 레지스터들의 내용은 오류정정을 위하여 DMEM의 내용을 읽거나 쓰기 위한 어드레스로 사용되어진다.

$\sigma(x)$의 근 α^{j1}의 j1을 구하는 방법은 신드롬 생성기에서 설명한 바와 같이 신드롬 레지스터, S0R, S1R, S2R을 $\sigma(x)$의 계수들로 초기화시키고, SR*AP → SR과 ACNT를 동기시켜 동작시킨 후, S0R+S1R+S2R=0이 되면 그 때의 ACNT의 값이 j1이 된다.

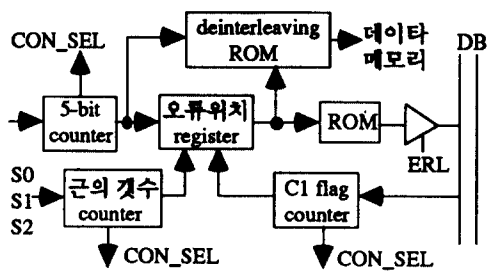


그림 9. Address 발생기 블럭.
Fig. 9. Address generator block.

C1복호에서 2개의 오류정정인 경우 ACNT가 0~31까지 count했을 때 2개의 오류가 정확하다면 $\sigma(x)$의 근의 갯수 counter(RTCNT)는 2가 되고, i1R, i2R에 각각 j1, j2가 저장되게 된다.

C2복호에서 C1 flag가 1인 위치를 ER에 저장하는 방법은 ACNT를 0~27까지 count하면서 이것을 DMEM의 어드레스로 사용해서 C1 flag를 읽은 다음 C1 flag counter(C1FCNT)로 가져온다. C1 flag가 1이면 C1FCNT를 1만큼 증가시키고, 그 때의 ACNT의 값을 ER로 shift시킨다. 이 과정이 끝난 후 C1 flag가 1인 것이 3개이면 C1FCNT는 3이고, ER2, ER3, ER4에는 각각 j1, j2, j3가 들어있게 된다.

5. 임시 메모리와 ZERO 검출기

임시 메모리는 14개의 일반 레지스터(GPR)들과 1개의 임시 레지스터(TMPR)로 구성되어 있으며, DB로 부터 데이타를 받아서 저장하거나 저장된 데이타를 DB로 내보낼 수 있다. 14개 GPR의 이름은 각각 GPR(S0), GPR(S1), GPR(S2), GPR(S3), GPR(TS1b1), GPR(TS2b2), GPR(TS3b3), GPR(NS1a1), GPR(NS2a2), GPR(NS3an), GPR(Q1ei1), GPR(Q2ei2), GPR(ei3), GPR(ei4)이다.

ZERO검출기(ZCNT)는 zero감지기와 counter로 구성되어 있으며, DB로 부터 받은 데이타가 0이면 counter를 1 만큼 증가시킨다. Counter의 출력은 CON_SEL로 간다.

Ⅳ. 복호제어기 및 마이크로명령어 (microinstruction) 설계

1. 복호제어기 구조

제어기 구조로는 초기의 wilkes model[13]을 비롯하여 여러가지가 제안되었으며[14,15], 본 논문에서는 그림10과 같은 구조로 설계하였다. 복호연산기와 마찬가지로 복호제어기도 최소한의 기능블럭들만 가지도록 했는데, 복호 프로그램을 저장할 수 있는 CS_ROM (Control Storage ROM), CS_ROM의 내용대로 제어신호를 발생시키기 위한 DEC(DECoder)가 있고, CS_ROM의 현재 어드레스를 가지고 있는 CSAR (CS_ROM Address Register), 현재 어드레스를 증가시키기 위한 +1블럭, subroutine call에서 return할 어드레스를 저장하거나, 반복 수행이 필요한 loop의 첫번째 어드레스를 저장하기 위한 STACK 등으로 구성되어 있다. CON_SEL 블럭은 현재 수행된 결

과들로 부터 주어진 조건에 따라 다음 프로그램으로의 진행을 제어하기 위한 것이다.

STACK에는 call명령으로 branch가 일어날 때는 return address(CSAR+1)이 들어가고, loop를 만나면 그 loop의 첫번째 어드레스(CSAR의 현재 내용)이 들어간다. CSAR에는 현재 어드레스가 들어 있으며, CSAR로 들어갈 다음 어드레스가 결정되는 것으로는 4가지 경우가 있다. 보통은 CSAR의 현재 내용에 1만큼 증가한 것이 다음 어드레스가 되고, 현재 명령의 반복(repeat)일 경우에는 CSAR에 있는 현재 어드레스가 다음 어드레스로 되며, return일 경우와 loop의 마지막 명령에서 조건에 의하여 loop를 반복수행할 경우에는 STACK의 내용이 다음 어드레스로 된다.

MFR(Monitor Flag Register)에는 C1복호인가 C2복호인가를 나타내는 C2R과 5개의 레지스터, MFR4~MFR0를 두고 있는데, 이 레지스터들은 명령에 의해서 값을 설정할 수 있다. 따라서 프로그램의 특정한 부분에서 이 레지스터의 값을 설정하면 프로그램의 진행상황을 모니터할 수 있으며, 프로그램을 debugging하거나, 하드웨어 test 등을 위해서 사용되어질 수 있다. 또한 실제 응용 시스템에서는 입력데이타의 오류상황과 정정상황등을 추적할 수 있는 장치로 활용 가능하다.

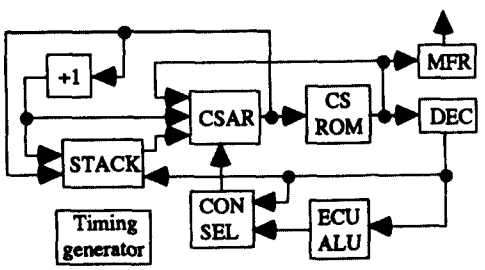


그림 10. 본 논문에서 설계한 복호 제어장치.
Fig. 10. Decoding controller for ECU_ALU proposed by this paper.

2. 마이크로명령어 설계

마이크로명령어는 한 clock cycle내에서 수행되어질 수 있는 마이크로동작(microoperation)들의 표현으로 vertical 형태와 horizontal 형태로 분류된다[16]. Vertical 형태는 한개의 마이크로명령어가 제어할 수 있는 기능블럭의 수가 적어서 serial processing에 가깝고, horizontal 형태는 한개의 마이크로명령어가 많은 기능블럭들을 제어할 수 있기 때문에 병렬처리의 형태를 갖는다. 따라서 컴퓨터를 설계할 경우에는 vertical 형태의 마이크로명령어는 12~24 비트정도의 크기를 가지며, horizontal형태는 60 비트 혹은 그 이상의 크기를 가진다. 마이크로명령어를 encoding하는 방법으로는 direct encoding, indirect encoding, bit steering 등의 방법이 있으며[16], 프로그램의 길이를 optimization하거나, 마이크로명령어의 비트 수를 줄이는 방법에도 여러가지가 있다[17]. 본 논문의 복호기는 컴퓨터의 하드웨어에 비해 규모는 작지만 이상과 같은 방법들을 적용해서 설계할 수 있다.

본 논문에서는 처리속도를 빠르게하기 위해서 병렬처리 형식을 갖는 horizontal microinstruction을 주로 사용했으며, 일부는 vertical microinstruction의 형식을 취했다. 아울러 마이크로명령어를 encoding하는 방법으로는 비트수를 최소로 하기 위하여 indirect와 bit steering방식을 적당히 혼용했다. 마이크로명령어의 비트수가 많아지면 DEC는 간단해지지만 CS_ROM이 커져서 낭비가 많아지고, 비트수를 극단적으로 적게 하면 DEC가 지나치게 커지고 복잡해지기 때문에 이런점을 잘 고려해서 설계해야 한다.

본 논문의 하드웨어는 컴퓨터 하드웨어에 비해서 기능블럭들의 수가 적기 때문에 병렬처리의 형식을 취하더라도 마이크로명령어의 비트수가 지나치게 커지지는 않는다. 이와같이 설계된 본 논문의 마이크로명령어는 14비트의 크기를 가지며, address generation, data transfer & ALU function, branch & call, set MFR, clear register 등 5개로 분류하고, 전체적으로는 7개의 group으로 구분했으며, 다음에서 이들에 대하여 상세히 설명하고자 한다.

가. Address Generation Group

복호기가 DMEM으로 부터 데이타를 읽거나 쓰는 명령은 그림11, 12와 같다. 그림11의 Group1은 ACNT에 관련된 것이고, 그림12의 Group2는 iR과 ER에 관련된 것이다. 그림11와 그림12에서 AM은 어드레스의 종류를 구분하는 것이며, OP는 어드레스 발생과 동시에 처리해야 할 데이타에 대한 마이크로동작이다. 신드롬을 생성시킬 때 혹은 복호 후 flag를 읽거나 쓸 때와 $\sigma(x)$의 근을 계산할 때 등은 ACNT를 이용하여 어드레스를 발생시키며, 이때 OP가 필요하다. 예를 들면 신드롬을 생성시킬 경우 데이타를 읽어옴과 동시에 DMEM+SP*AP ⟶ SR의 동작이 신드

Group 1 : ACNT

0	0	0	1	AM(3)	OP (3)	con.(4)

```
AM(3)
001 ACNT-->DIR; (rd,c2,data)-->DIR;
010 ACNT-->DIR; (rd,c1,flag)-->DIR;
011 ACNT-->DIR; (rd,c1,data)-->DIR;
100 ACNT-->DIR; (wr,c2,flag)-->DIR;
101 ACNT-->DIR; (wr,c2,data)-->DIR;
110 ACNT-->DIR; (wr,c1,flag)-->DIR;
111 ACNT-->DIR; (wr,c1,data)-->DIR;
OP(3)
001 0+SR*AP-->SR;
010 DMEM+SR*AP-->SR;
011 DMEM-->C1FCNT;
100 DMEM-->C1FR;
101 C1FR-->DMEM;
110 FLAG(00)-->DMEM;
111 FLAG(03)-->DMEM;
con.(4)
0001 ACNT++;
0010 ACNT++; uncon. ACNT-->iR;
0011 ACNT++; uncon. CSAR-->STACK;
0100 ACNT++; if(ACNT/=23) goto(STACK);
0101 ACNT++; uncon. ACNT-->iR;
     if(C1F=1) {iR-->ER;C1FCNT++;};
     for(ACNT/=3127) repeat;
0110 if(S1+S0) {ACNT-->iR;RTCNT++;};
     for(ACNT/=3127) repeat;
0111 if(S2+S1+S0) {ACNT-->iR;RTCNT++;};
     for(ACNT/=3127) repeat;
1000 for(ACNT/=3127) repeat;
```

그림 11. Group1 microinsruction.
Fig. 11. Group1 microinsruction.

Group 2 : iR, ER

0	1	1	1	1	AM(3)	i1R	i2R	ER1	ER2	ER3	ER4

```
AM(3)
001 (rd,c2,data)-->DIR;  i1R = 1 i1R-->DIR;
010 (rd,c1,flag)-->DIR;  i2R = 1 i2R-->DIR;
011 (rd,c1,data)-->DIR;  ER1 = 1 ER1-->DIR;
100 (wr,c2,flag)-->DIR;  ER2 = 1 ER2-->DIR;
101 (wr,c2,data)-->DIR;  ER3 = 1 ER3-->DIR;
110 (wr,c1,flag)-->DIR;  ER4 = 1 ER4-->DIR;
111 (wr,c1,data)-->DIR;
```

그림 12. Group2 microinsruction.
Fig. 12. Group2 microinsruction.

롬 생성기에서 이루어져야 한다. Con.은 OP와는 달리 어드레스 발생과 동시에 조건에 따라서 처리해야 할 동작들을 위한 것이다. 특히 repeat는 CS_ROM의 크기를 줄일 위하여 같은 명령의 반복수행을 위하여 사용되었다. Group2의 명령어에는 특별한 OP나 con.이 필요하지 않으며, 따라서 Group1과 2의 마이크로명령어를 이용한 실제 프로그램은 그림13과 같이 할 수 있다. 그림13에서 for 다음의 {}내에 있는 모든 동작은 한 clock cycle 동안에 완성되도록 했고, 어드레스가 발생된 다음 3 clock cycle후에 원하는 데이타가 DB에 나타나도록 설계했다. 또한 if(ACNT=3127)는 C1복호일 때는 ACNT=31, C2복호일 때는 ACNT=27이 되는 조건을 의미한다.

```
*C1 복호시 신드롬 생성.*
for(ACNT/=3127) {
  {ACNT,(rd,c1,data)}-->DIR;ACNT++;
   DMEM+SR*AP-->SR;}
*C1복호, 2개 오류에서 σ(x)의 근 계산*
for(ACNT/=3127) {
   0+SR*AP-->SR;
  if(S2+S1+S0=0) {ACNT-->iR;RTCNT++;};
   ACNT++;}
*C2 복호시 C1 flag를 읽음*
for(ACNT/=3127) {
  {ACNT,(rd,c1,flag)}-->DIR;
   ACNT-->iR;ACNT++;
   DMEM-->C1FCNT;
   if(C1 flag=1) {iR-->ER;C1FCNT++;};}
*C1 복호시 오류정정*
{i1R,(rd,c1,data)}-->DIR;
{i1R,(wr,c1,data)}-->DIR;
GPR(Q1ei1)-->MAR
DMEM-->ABR
MAR*1+ABR-->DMEM
*C2 복호시 소실정정*
{ER4,(rd,c2,data)}-->DIR;
{ER4,(wr,c2,data)}-->DIR;
GPR(ei4)-->MAR
DMEM-->ABR
MAR*1+ABR-->DMEM
```

그림 13. 프로그램 예제.
Fig. 13. Example program.

나. Data Transfer & ALU Function Group

그림14의 Group3는 GPR로 부터 다른 레지스터로 데이타를 전달하는 것이고, 그림15의 Group4는 ALU의 결과 혹은 SR, ER, iR 등의 데이타를 GPR과 ALU 레지스터, TMPR 등으로 보내는 것이다. 처리속도를 빠르게 하고 CS_ROM의 크기를 줄이기 위하여 S(source)로 부터 데이타를 보낼 수 있는 D(destination)를 두개로 했다. Group4의 ZCE는 S를 D1,

Group 3 : S(GPR) --> D2, D1

0	0	0	0	S(GPR)(4)	D2(3)	D1(3)

S(GPR)(4)	D2(3)	D1(3)
0010 GPR(S0)	000 none	000 none
0011 GPR(S1)	001 MAR	001 MAR
0100 GPR(S2)	010 nMAR	010 nMAR
0101 GPR(S3)	010 nMAR	010 nMAR
0110 GPR(TS1b1)	011 MBR	011 MBR
0111 GPR(TS2b2)	100 ABR	100 ABR
1000 GPR(TS3b3)	101 TMPR	101 SOR
1001 GPR(NS1a1)	110 ZCNT	110 S1R
1010 GPR(NS2a2)		111 ZCNT
1011 GPR(NS3an)		
1100 GPR(Q1ei1)		
1101 GPR(Q2ei2)		
1110 GPR(ei3)		
1111 GPR(ei4)		

그림 14. Group3 microinsruction.
Fig. 14. Group3 microinsruction.

Group 4 : S --> D1(GPR), D2, ZCNT

0	0	ZCE	D1(GPR)(4)	D2(3)	S(4)

D2(3)	S(4)
001 MAR	0001 MAR*MBR+0
010 nMAR	0010 MAR*1+ABR
011 MBR	0011 MAR*MBR+ABR
100 ABR	0100 SOR
101 TMPR	0101 S1R
110 DMEM	0110 S2R
	0111 S3R
* D1은 Group3	1000 ER1
의 S와 같음.	1001 ER2
	1010 ER3
* ZCE=1이면	1011 ER4
S-->ZCNT	1100 i1R
	1101 i2R
	1110 TMPR
	1111 FLAG(01)

그림 15. Group4 microinsruction.
Fig. 15. Group4 microinsruction.

D2와 더불어 ZCNT에도 전달할 수 있기 위함이다.

다. Branch & Call Group

Branch를 위한 조건으로는 C2R, ZCNT, RTCNT, C1FCNT등이 있으며, subroutine의 사용을 위해서는 call과 return이 있어야 하는데, 이들을 위한 마이크로명령어는 그림16과 같다. Branch address에 9비트가 할당되어 있기 때문에 프로그램의 최대길이는 512를 넘지 못한다.

Group 5 : Branch & Call

con.(5)	branch address (9)

con.(5)	con.(5)
011 01 RTCNT/=1	11 001 C1FCNT<=1
10 RTCNT/=2	010 C1FCNT=2
	011 C1FCNT=3
10 001 ZCNT=0	100 C1FCNT=4
010 ZCNT/=0	101 C1FCNT>=5
011 ZCNT>=2	
100 ZCNT>=3	
101 ZCNT=3	11110 call
110 ZCNT=4	11111 return
111 C2R=1	

그림 16. Branch와 Call microinstruction.
Fig. 16. Branch & Call microinstruction.

라. Set MFR Group과 Clear Register Group

Group6은 MFR에 있는 레지스터들의 값을 설정하기 위한 것이고, Group7은 SR, ACNT, ZCNT, C1FCNT, RTCNT 등을 clear시키기 위한 microinsruction으로 그림17과 같다.

Group 6 : setting MFR

0	0	0	0	0	0	0	0	1	MFR(6)

MFR(6) =
(C2R,MFR4,MFR3,MFR2,MFR1,MFR0)

Group 7 : clear register

0	0	0	0	0	0	0	0	0	CLR(6)

CLR(6)l = 000000 NOP(No OPeration)
1xxxxx clear SR
x1xxxx clear ACNT
xx1xxx clear ZCNT
xxx1xx clear RTCNT
xxxx1x clear C1FCNT
xxxxx1 clear (reserved)

그림 17. MFR값 설정과 clear register.
Fig. 17. Microinstruction for setting MFR and reset register.

V. 검토 및 결론

복호연산기 및 복호제어기는 ROM과 5종류의 D-F/F

(D-type flip/flop), 4종류의 multiplexer 및 기본 gate들을 이용하여 설계했으며, Verilog HDL의 Gate Level Modeling[11]을 이용하여 구현하였다. 이때 복호기에 사용한 기본 clock으로는 프레임신호(7.35kHz)의 588배인 4.3218MHz를 사용했다. Ⅳ장에서 설계한 마이크로명령어로 Ⅱ장의 CD복호 방법을 프로그램한 결과, 프로그램 ROM의 크기는 360 word(1 word =14 bit)가 되었다. 프로그램은 C1, C2복호와 아울러 subroutine 1, subroutine 2를 가지고 있는데, subroutine 1은 C2복호시 4개의 소실정정에서 오류값을 구하는데 사용되고, subroutine 2는 C2복호시 3개의 소실정정에서 오류값을 구하는데 쓰여진다. C1, C2복호를 모두 끝내는데 가장 많은 시간이 소요되는 경우는 C1복호에서 2개의 오류를 정정하고, C2복호에서 4개의 소실정정을 행할 때이며, 본 프로그램에서는 424 clock cycle이 소요된다. 이상에서 설계된 복호기는 현재 CDP에 사용하기 위해서 IC 설계중에 있다.

Super Strategy[3]와 T.Arai등이 제안한 복호방법[4]등은 C2복호에서 C1 flag를 확인하는 방법으로 2개까지의 오류데이타를 정정하는 반면, 본 논문의 복호기는 C2복호에서 3개 뿐만 아니라, 4개까지의 소실정정을 할 수 있도록 해서 오류정정능력을 향상시켰다. 심볼오류율, Ps가 10^{-3}일때 3개와 4개의 소실정정에 의해 향상되는 오류정정확률, P_3, P_4는 각각 3.529×10^{-17}, 1.368×10^{-22}정도이다.

본 논문에서 사용한 복호방법은 Gorenstein-Zierler의 계산방법[9]과 비슷한 것으로, Newton항등식등에 의해서 만들어지는 연립방정식을 이용하여 오류위치 다항식의 계수들을 구하고, 소실데이타의 갯수를 확인하며, 신드롬들의 연립방정식을 풀어서 오류값을 구하는 것이다. 이 복호방법은 본 논문에서 설계된 복호기처럼 중앙연산장치를 가지고 있는 구조에 적합하다.

Berlekamp, Massey등의 FSR에 의한 방법(5,6)은 하드웨어가 복잡하고, 이만영, 김창규[10]등이 제안한 복호기는 하드웨어가 너무 커지는 반면, 본 논문에서 설계한 RS 복호기는 간단하면서도 체계적인 구조를 갖기 때문에 성능개선을 위한 복호방법 혹은 하드웨어의 변경이 용이하다.

T.Arai등이 제안한 복호기[4]와는 다르게 본 논문에서 설계한 복호연산기는 GF(2^8)상의 역원 ROM과 승산기 및 가산기 각각 1개씩으로 구성했는데, 이 경우 여러가지 연산기능을 할 수 있으면서도 제어가 간단하다. 또한 call과 repeat명령을 사용해서 프로그램 ROM의 낭비를 줄일 수 있도록 했다.

본 논문에서와 같이 마이크로프로그래밍 기법을 이용한 설계방법은 이와 유사한 시스템의 설계에도 이용될 수 있을 것이다.

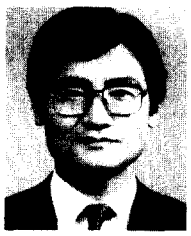
참 고 문 헌

1. N.V.Philips, Sony Corp, System Description of Compact Disc, Confidential Information, September 1982.
2. DAT Conference, Recomended design standard 〈R-DAT〉, April 1986.
3. M.Yashiro, H.Imai, "Decoding Properties of the Super Strategy for CIRC," IECE Tr. J66A, No. 3, pp.284-285, March 1983.
4. T.Arai et al., "High Capability Error Correction LSI for CD Player and CD-ROM," IEEE Tr. on Consumer Electronics, vol.CE-30, No.3, pp.353-359, August 1984.
5. E.R.Berlekamp, Algebraic Coding Theory, McGraw Hill, New York, 1968.
6. J.L.Massey, "Shift Register Synthsis and BCH Coding," IEEE Trans. Inf. Theory, IT-15, pp. 122-127, 1969.
7. L.R.Welch, R.A.Scholtz, "Continued Fractions and Berlekamp's Algorithm," IEEE Trans. Inf. Theory, IT-25, pp.19-27, 1979.
8. R.E.Blahut, "Transform Techniques for Error-control Code," IBM J.Res.Dev., 23, pp.299-315, 1979.
9. D.C.Gorenstein and N.Zieler, "AClass of Error-correcting Codes in pm Symbols," J. Soc. Indust. Appl. Math., 9, pp.207-214, 1961.
10. 이만영, 김창규, "2중 오류정정 Reed-Solomon부호의 부호기 및 복호기 정치화에 관한 연구," 전자공학회논문지, 제26권 제2호, pp.10-17, 1989년 2월.
11. Verilog-XL Reference Mannual, Cadence Design System, Inc., 1991.
12. M.Y.Rhee, Error Correcting Coding Theory, McGraw Hill, 1989.
13. M.V.Wilkes, "The Best Way to Design an Automatic Calculating Machine," Report of the Manchester University Computer Inaugural Con-

ference, Electrical Engineering Department of Manchester University, Manchester, England, pp.16-18, July 1951, reprinted in Earl E.Swartzlander, Jr., ed., Computer Design Develpment-Principal Papers, Hayden Book Co., Rochelle Park, NJ, 1976, pp.266-270.

14. C.A.Papachristou, "Hardware Microcontrol Schemes using PLAs," 14th Annual Workshop on Microprogramming, MICRO-14, pp.3-16, 1982.
15. V.Milutinovic, Tutorial : Microprogramming and Firmware Engineering, IEEE Computer Society Press, 1989.
16. T.G.Rouscher, P.M.Adams, "Microprogramming : A Tutorial and Survey of Recent Developments," IEEE Tr. on Computers, vol.C-29, No.1, pp.2-20, Jannuary 1980.
17. T.Agerwala, "Microprogram Optimization : A Survey," IEEE Tr. on Computers, vol.C-25, No. 10, pp.962-973, October 1976.

金 泰 鎔(Taeyong Kim) 正會員
1955년 7월 18일생
1983년 2월 : 경북대학교 전자공학과 졸업
1986년 2월 : 한국과학기술원 전기 및 전자공학과 공학석사
1992년 3월 ~ 현재 : 한국과학기술원 전기 및 전자공학과 박사과정
1983년 1월 ~ 현재 : 삼성전자 반도체 부천연구소 선임연구원
※주관심분야 : 영상신호처리, 디지탈 신호처리, 부호이론, 통신시스템, VLSI구조



金 在 均(Jae Kyoon Kim)종신회원
1938년 9월 17일생
1961년 12월 : 항공대학 졸업
1967년 2월 : 서울대학교 전자공학과 석사
1971년 8월 : 남 캘리포니아대학 전자공학과 박사
1973년 4월 ~ 현재 : 한국과학기술원 교수
1993년 1월 ~ 현재 : 한국통신학회 회장