논문 2006-43SD-4-6

DMB 휴대용 단말기를 위한 Reed-Solomon 복호기의 설계

(Hardware design of Reed-solomon decoder for DMB mobile terminals)

류 태 규*, 정 용 진**

(Tae-Gyu Ryu and Yong-Jin Jeong)

요 약

본 논문에서는 DMB(Digital Multimedia Broadcasting) 단말기에서 사용하기 위하여 유클리드(Euclid) 알고리즘 기반의 RS(255,239,t=8) 복호기를 설계하였다. DMB는 휴대 단말기 상에 방송서비스 제공이 목적이므로 사용된 RS 복호기는 면적이 작아야 하며 실시간처리를 위해 복호 지연시간이 짧아야 한다. 두 조건을 만족시키기 위해 에러의 위치 및 크기를 찾는 방법으로 유클리드 알고리즘을 수정하여 사용하였다. 유클리드 알고리즘 상에서 유한체 나눗셈 연산을 위해 사용하는 Inverse ROM을 17 클럭을 소모하는 나눗셈기로 대체하여 면적을 줄였으며, 유한체 나눗셈기로 인한 지연 시간을 줄이기 위해 차수 연산 없이 유클리드 알고리즘의 동작 제어가 가능한 수정된 유클리드 알고리즘을 제안하였다. 제안한 유클리드 알고리즘은 기본 유클리드 알고리즘에 비해 비슷한 지연시간 조건 하에서 면적을 25% 정도 줄일 수 있었다. 삼성 STD130 0.18㎛ 표준 셀 라이브러리를 이용하여 Synopsys 상에서 합성한 결과 유클리드 블록은 30,228개의 게이트수를 가지며 288 클럭을 소모하였으며, 전체 RS 복호기의 크기는 약 45,000 게이트였다.

Abstract

In this paper, we developed a hardware architecture of Reed-Solomon RS(255,239) decoder for the DMB mobile terminals. The DMB provides multimedia broadcasting service to mobile terminals, hence it should have small dimension for low power and short decoding delay for real-time processing. We modified Euclid algorithm to apply it to the key equation solving which is the most complicated part of the RS decoding. We also designed a small finite field divider to avoid the use of large Inverse-ROM table, and it consumed 17 clocks. After synthesis with Synopsis on Samsung STD130 0.18㎛ Standard Cell library, the Euclid block had 30,228 gates and consumed 288 clocks, which gave the 25% reduced area compared to other existing designs. The size of the entire RS decoder was about 45,000 gates.

Keywords : Reed-solomon, Euclid, DMB

I. 서 론

DMB는 이동 중인 휴대단말기에 고화질 고음질의 멀티미디어 방송을 끊김 없이 제공 할 수 있는 서비스이다. 그러므로 이동 중에 생기는 채널특성의 변화에 따라 높아지는 에러 확률 문제에 대처할 수 있어야 한다. DMB는 전송도중 발생한 에러의 정정을 위하여 에러 정정 부호인 RS(Reed-Solomon) 부호를 사용하고 있다. RS 부호는 BCH(Bose - Chadhuri - Hocquenghem) 부호의 한 범주에 속하며 유한체 GF(q) 상의 심벌 단위로 부호화 및 복호화 되므로 통신로 상에서 발생하는 산발오류(random error)와 연집오류(burst error)를 모두 정정할 수 있다. 휴대단말기에 방송서비스를 제공하기 위해 작은 면적을 가지면서도 DMB가 요구하는 성능을 만족 시킬 수 있는 RS 복호기가 필요하다. DMB가 요구하는 RS 복호기는 최대 1.8 Mbps의 고정된 데이터 전송율으로 입력되는 데이터의 처리가 가능하여야

* 준회원, 코아로직
(CoreLogic)
** 정회원, 광운대학교 전자통신공학과
(Dept. of Electronics and Communication Engineering Kwangwoon University)
※ 본 연구는 광운대학교 2005 교내학술연구 및 IDEC/SIPAC의 지원으로 이루어졌습니다.
접수일자: 2005년11월9일, 수정완료일: 2006년4월3일

하며[1] 실시간 처리를 위해 에러정정 지연시간이 짧아야 한다. 데이터 전송율 1.8 Mbps은 하드웨어로 구현된 RS 복호기가 충분히 제공할 수 있는 비교적 낮은 데이터 전송율이므로 휴대용 단말기의 전력특성을 위해 낮은 동작 주파수를 사용할수록 좋다. 이를 위해 데이터 처리 동안에 사용된 클럭의 수를 줄여야 한다. 정리해 보면 DMB에서 사용하기 위한 RS 복호기는 면적을 줄이고 입력된 데이터가 처리될 때까지 사용된 클럭의 수가 적어야 한다.

RS부호는 복호과정에서 에러위치 다항식(Error location polynomial) $\sigma(x)$와 에러값 다항식(Error value polynomial) $\omega(x)$를 구하여 에러를 정정하게 된다.[3] 두 다항식을 구하는 식을 Key Equation 이라 하며 대표적인 풀이방법으로 BM(Berlekamp-Massey)알고리즘과 유클리드 알고리즘이 있다. BM 알고리즘의 경우 회로가 복잡하고 많은 계산량 때문에 복호시간이 길어지고 크기가 커지는 문제점을 가지고 있다. 반면에 유클리드 알고리즘의 경우 비교적 규칙적인 연산과정을 가지고 있다. 그러나 GF(q)상의 나눗셈 연산을 기본으로 하는 유클리드 알고리즘은 제수의 역수를 구하기 위한 Inverse ROM 으로 인해 최장지연시간이 길어져 데이터 전송율이 떨어지고 면적이 커지는 문제점을 가지고 있다.[5]

최근에는 주로 나눗셈을 사용하지 않는 수정된 유클리드 알고리즘이 쓰이고 있다. 수정된 유클리드 알고리즘은 회로가 비교적 간단하고 나눗셈 연산이 없으므로 최장지연시간이 짧아 더욱 높은 동작주파수를 사용 할 수 있어 높은 데이터 전송율을 가질 수 있다. 기본 블록을 시스톨릭 어레이 구조로 배열하여 파이프라인 처리를 통해 데이터 전송율을 더욱 높일 수도 있다. 하지만 어레이 구조로 다수의 기본블록을 사용하므로 면적이 커지며 기본 유클리드 알고리즘에 비해 많은 클럭을 소모하는 단점을 가지고 있다.[7]

위와는 반대로 한 개의 기본 블록만을 반복적으로 사용하는 재귀(Recursive)구조가 있다. 시스톨릭 어레이 방식과 달리 한 개의 기본 블록만을 사용하므로 작은 면적을 가질 수 있고 역시 최장지연시간이 짧아 높은 동작주파수를 사용할 수 있어 높은 데이터 전송율을 가진다. 하지만 기본 블록 하나만을 반복적으로 사용하므로 시스톨릭 구조보다도 더 많은 클럭을 소모하는 단점을 가지고 있다.[8]

본 논문은 많은 클럭을 소모하는 나눗셈을 사용하지 않는 수정된 유클리드 알고리즘 대신 적은 클럭을 소모하는 유클리드 알고리즘을 사용하였다. 대신 유클리드 알고리즘의 단점을 보완하여 면적을 줄이면서도 연산과정에서 불필요하게 낭비되는 클럭을 제거하여 데이터 처리 지연시간을 줄일 수 있도록 수정하였다.

본 논문의 Ⅱ장에서는 DMB 표준에서 원하는 RS 부호의 성능을 보이고 RS 부호에 대하여 간단히 설명한다. Ⅲ장에서는 RS 복호기의 구조를 보이고 서브 블록들의 구조와 기능 동작에 대하여 설명하고 Ⅳ장에서는 Key Equation을 풀기 위하여 사용된 수정된 유클리드 알고리즘을 설명한다. Ⅴ장에서는 구현된 유클리드 알고리즘 블록을 보이고 타 논문과 비교 분석 한 후 Ⅵ장에서 결론을 맺는다.

Ⅱ. RS부호 지상파 DMB 표준

RS 부호는 RS(n,k)로 표시되며, 이를 그림으로 표현하면 다음과 같다.

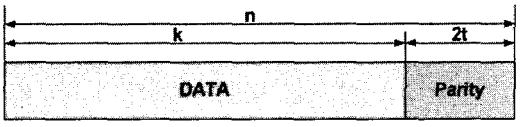

그림 1. Reed-Solomon 부호의 구조
Fig. 1. Structure of the Reed-Solomon code.

그림에서 n은 부호화된 패킷의 전체 심벌의 개수를 의미하며 k는 실제로 전송되어야 할 데이터를, 그리고 t는 복호화 단에서 정정할 수 있는 심벌의 개수를 의미한다. $2t$만큼의 패리티가 데이터 뒤에 붙어 최대 t개의 심벌에 발생한 에러를 정정할 수 있다. 표준 RS 부호에서는 한 심벌의 크기를 m_bit라 할 때 $n = 2^m - 1$이다.[2]

DMB에서 요구하는 RS부호의 규격은 우선 심벌의 크기는 8bit이며 RS(255, 239, t=8)에서 유도되는 단축된 RS(204, 188, t=8)이다. n이 255인 RS 부호화기의 입력 데이터의 앞단 51바이트를 0으로 가정하여 부호화 한 후 204 Byte 앞단의 0을 무시하면 n이 204인 RS 부호화된 전송패킷을 얻을 수 있다. 본 논문에서는 RS(255, 239, t=8)의 복호기 설계에 대해 기술하며, 이때 코드발생다항식 $g(x)$와 필드발생다항식 $p(x)$는 다음과 같다. 여기서 α는 원시원(element)이다.

$$g(x) = (x+\alpha^0)(x+\alpha^1)(x+\alpha^2)\cdot\cdot\cdot(x+\alpha^{15})$$
$$p(x) = x^8 + x^4 + x^3 + x^2 + 1 \quad (1)$$

전송되는 데이터의 전송율는 고정되어 있으며 서비스 가능한 데이터의 최대 데이터 전송율은 1.8 Mbps이다.[1]

1. Reed-Solomon 부호화

데이터 다항식을 $D(x)=d_0+d_1x+\cdots+d_{k-1}x^{k-1}$, 패리티 다항식을 $P(x)=p_0+p_1+\cdots+p_{2t-1}x^{2t-1}$라 하면 RS부호 코드워드의 다항식 $C(x)$는 다음과 같이 표현할 수 있다.

$$C(x)=x^{2t}D(x)+P(x)=\sum_{i=0}^{n-1}c_i x^i \tag{2}$$

RS 부호의 생성다항식 $g(x)$는 α^{m_0}, α^{m_0+1}, $\cdots$, $\alpha^{m_0+(2t-1)}$을 근으로 가지는 $GF(2^m)$상의 최소다항식의 곱이므로 다음과 같다.

$$g(x)=\prod_{i=0}^{2t-1}(x+\alpha^{m_0+i})=\sum_{i=0}^{2t}g_i x^i \tag{3}$$

m_0는 임의의 정수, α는 $GF(2^m)$의 원시원이며 g_i는 $GF(2^m)$상의 원소이다. RS 부호의 부호화는 $x^{n-k}D(x)$를 $g(x)$로 나누어 나머지 $P(x)$를 구하고 이를 $D(x)$의 뒤에 붙여 완성한 $C(x)$를 전송하게 된다.

2. Reed-Solomon 복호화

$C(x)$ 전송 코드워드다항식을 전송하면 수신측에서는 전송도중 발생한 에러가 더해져 수신 코드워드다항식 $I(x)$를 수신한다.

$$I(x)=C(x)+E(x)$$

이때 $E(x)$는 전송 도중 더해진 에러다항식으로 다음과 같이 표현한다.

$$E(x)=Y_1x^{i_1}+Y_2x^{i_2}+\cdots+Y_tx^t$$

이때, 에러의 값은 Y_1, Y_2, $\cdots$, Y_t이고 에러의 위치는 $X_1=\alpha^{i_1}$, $X_2=\alpha^{i_2}$, $\cdots$, $X_t=\alpha^{i_t}$이며 에러를 복구하기 위해서 에러의 위치, 에러의 값을 알아야만 한다. 이를 에러위치 다항식 $\sigma(x)$, 에러값 다항식 $\omega(x)$라고 하며 다음과 같이 표현한다.[2]

$$\sigma(x)=\prod_{j=1}^{t}(1-X_j x) \tag{4}$$
$$=1+\sigma_1x+\sigma_1x^2+\cdots+\sigma_t x^t$$

$$\omega(x)=\sum_{i=1}^{t}Y_iX_i\prod_{j=1,j\neq i}^{t}(1-X_jx) \tag{5}$$
$$=\omega_0+\omega_1x+\omega_2x^2+\cdots+\omega_{t-1}x^{t-1}$$

RS부호를 복호화 하기 위해서는 우선 신드롬(Syndrome)다항식 $S(x)$를 구하여야 한다. $S(x)$는 $I(x)$를 $g(x)$로 나눈 나머지로 전송된 패킷에 에러가 있는지 여부를 확인 할 수 있다. 자세한 내용은 3장에서 설명하였다. 구해진 $S(x)$를 바탕으로 $\sigma(x)$, $\omega(x)$을 구하며 $S(x)$와 $\sigma(x)$, $\omega(x)$의 관계를 Key Equation이라 하며 다음과 같다.[2][3]

$$\sigma(x)S(x)=\omega(x)\bmod x^{2t} \tag{6}$$

Key Equation을 풀면 에러위치 다항식과 에러값 다항식을 얻을 수 있다. 에러 위치 다항식의 근이 에러의 위치이며 구한 근을 에러값 다항식에 대입하면 해당 위치의 에러값을 구할 수 있다. 구한 에러값을 전송된 $I(x)$에 에러 위치에 따라 더하면 최종적으로 에러가 정정된 $C(x)$를 구할 수 있다.

III. Reed-Solomon 복호기의 구조

1. 신드롬계산 블록 (Syndrome Calculation Block)

신드롬계산 블록에서는 전송된 데이터 $I(x)$를 $g(x)$로 다시 나누며 이를 풀이하면 다음과 같다.

$$\frac{I(x)}{g(x)}=\frac{D(x)x^{n-k}+P(x)+E(x)}{g(x)}=\frac{E(x)}{g(x)}$$

결과는 에러다항식 $E(x)$를 $g(x)$로 나누는 것과 같다. 전송 도중 에러가 더해지지 않아 $E(x)$가 0이 된다면, $I(x)$를 $g(x)$로 나누어 남은 나머지가 0이 됨을 의미한다. 즉 나머지가 0이 되면 전송도중 에러가 발생치 않았음을 의미하며 0이 아닌 값을 가지게 되면 전송도중 에러가 발생했음을 의미한다. 이 때 남은 나머지를 신드롬 다항식 $S(x)$라고 한다.

$$S(x)=I(x)\bmod g(x) \tag{7}$$

하지만 코드발생 다항식 $g(x)$의 특징에 의하여 다음

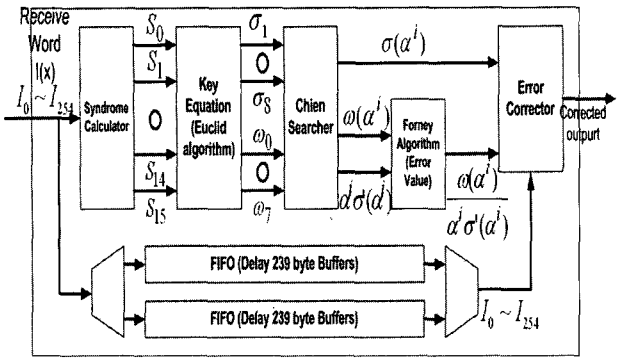

그림 2. 유클리드 알고리즘을 이용한 RS 복호기 전체 구조

Fig. 2. The RS decoder structure using Euclid algorithm.

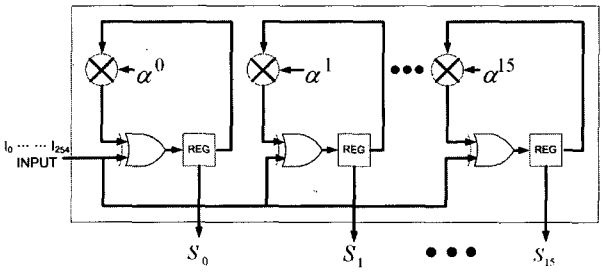

그림 3. 신드롬 계산 블록

Fig. 3. The syndrome calculation block.

과 같이 표현가능하다.[2]

$$S(x)=\sum_{i=0}^{2t-1} S_i x^i, \quad \text{with } S_i=\sum_{j=0}^{n-1} r_j \alpha^{ij} \tag{8}$$

이를 하드웨어로 구현하면 그림 3 과 같이 나타낼 수 있다. 구해진 $S(x)$는 유클리드 알고리즘 블록에 입력되어 Key equation 풀이 과정을 거치게 되며 만약 $S(x)$가 0이었다면 Key equation 풀이과정을 생략하고 FIFO에 저장되어 있던 전송받은 데이터를 출력한다.

2. 유클리드 알고리즘 블록(Key Equation Block)

유클리드 알고리즘은 기본적으로 두 다항식 사이의 최소공약수를 구하는 알고리즘이다. 하지만 그 형태가 Key equation과 일치하므로 유클리드 알고리즘에 Key equation을 대입하여 Key equation을 풀이할 수 있다.[2]

기본적인 유클리드 알고리즘 블록의 구조는 곱셈기 불록과 나눗셈기 블록으로 구성되어 있다. 나눗셈기 블록은 다음과 같은 연산을 반복한다.[2]

$$\begin{aligned} &R_0(x)=x^{2t}, \quad R_1(x)=S(x) \\ &R_{i-1}(x)=R_i(x)\,Q_i(x)+R_{i+1}(x), \\ &i=1,2,\cdots,k \end{aligned} \tag{9}$$

식 (9)는 $\deg[R_{k+1}(x)]<t$ 의 조건이 만족될 때까지 지속적으로 다항식 나누기 다항식의 결과 몫 Q_i 를 구하기 위한 나누기 연산을 반복하게 된다. 여기서 R 은 연산을 하기 위한 저장 레지스터를 의미하며 k는 연산의 끝나는 순간을 의미한다. 나머지인 R_{i+1}의 차수가 최대로 정정 가능한 심벌의 개수 t보다 작을 때 나누기 블록의 연산을 끝마치게 되며 이 때 i의 값이 k 이다. 구해진 Q_i는 곱셈기 블록으로 보내진다.

곱셈기 블록에서는 Q_i를 받아 다음과 같은 연산을 반복한다.

$$\begin{aligned} &B_{-1}(x)=0, \quad B_0(x)=1 \\ &B_i(x)=B_{i-1}(x)Q_i(x)+B_{i-2}(x), \\ &i=1,2,\cdots,k \end{aligned} \tag{10}$$

위의 다항식 곱셈 연산은 나눗셈기 블록에서 연산이 끝날 때까지 반복되며 모든 연산이 끝나면 최종적으로 $\sigma(x)$와 $\omega(x)$를 구하여야 하며 관련 식은 다음과 같다.[2]

$$\omega(x)=R_k(x)/B_k(0)$$

$$\sigma(x)=B_k(x)/B_k(0)$$

위의 내용은 기본적인 유클리드 알고리즘을 이용한 Key equation 풀이방법을 서술한 것이다. 면적을 최소화 하며 지연시간을 줄이기 위한 수정된 유클리드 알고리즘은 Ⅳ장에서 설명한다.

3. 친 서치 블록(Chien search Block)

친 서치(Chien search) 블록에서는 유클리드 알고리즘 블록에서 구해진 에러위치 다항식과 에러값 다항식을 이용하여 에러의 위치를 구하고 해당 위치의 에러값을 구하게 된다. 이때 에러위치 다항식의 근이 에러의 위치를 나타내지만 에러위치 다항식은 8차 다항식으로 하드웨어적으로 근을 구하기 힘들다. 그러므로 $GF(2^8)$ 상에서 가능한 모든 심벌 $\alpha^1 \sim \alpha^{255}$을 에러위치 다항식에 대입하여 에러위치 다항식의 근을 구한다.

$\sigma(x)=\sigma_0+\sigma_1 x+\sigma_2 x^2+\cdots+\sigma_8 x^8$ 에러위치 다항식은 다음과 같이 표현 할 수 있으며 에러값 다항식은 $\omega(x)=\omega_0+\omega_1 x+\omega_2 x^2+\cdots+\omega_7 x^7$으로 표현할 수 있다. 이를 짝수 차수와 홀수 차수로 나누어 하드웨어적으로 표현한 것이 그림 4의 (b)와 (c)이다. 이를 통

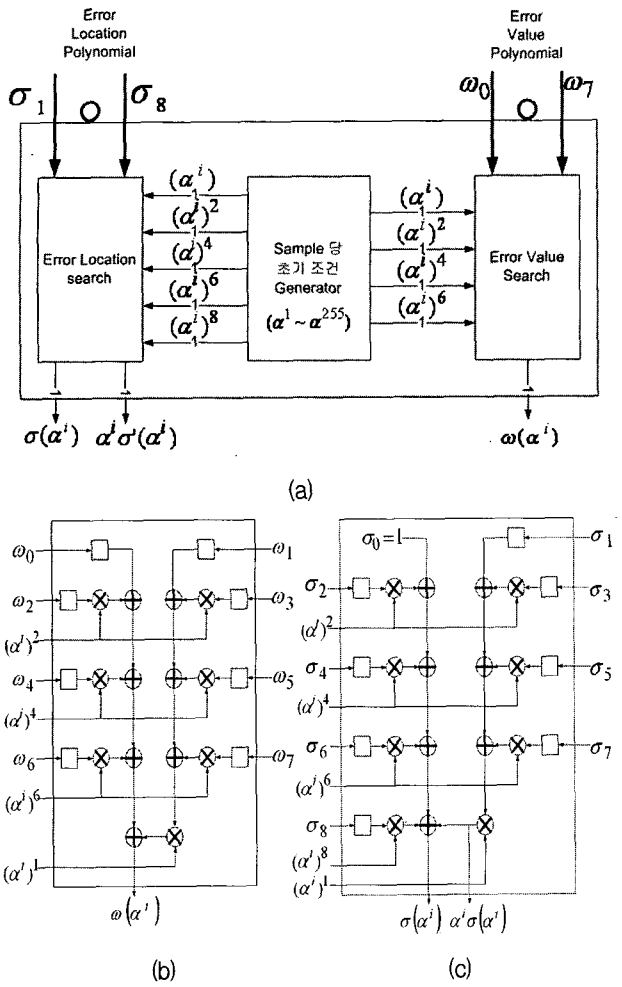

그림 4. (a) 친 서치 블록
(b) 에러값 탐색 블록
(c) 에러위치 탐색 블록

Fig. 4. (a) The Chien search block.
(b) Error value search block.
(c) Error location Search block.

해 최장지연시간을 절반으로 줄일 수 있다.[7] 그림 4에서 $x\sigma'(x)$는 에러위치 다항식을 미분한 결과로 이를 식으로 나타내면 다음과 같다.

$$\sigma'(x) = \sigma_1 + \sigma_3 x^2 + \cdots + \sigma_{t-1} x^{t-2}$$

$$x\sigma'(x) = \sigma_1 + \sigma_3 x^3 + \cdots + \sigma_{t-1} x^{t-1}$$

즉, $x\sigma'(x)$ 결과는 $\sigma(x)$ 결과의 홀수 차수부분만을 출력하면 된다. $\sigma(\alpha^i)$ 결과가 0이 되면 해당위치의 데이터 심벌에 에러가 있음을 의미한다.

4. 포니(Forney) 알고리즘 과 에러 정정 블록

포니 알고리즘 블록은 유한체 나눗셈기로 이루어져 있으며 $\omega(\alpha^i)$,$\alpha^i\sigma'(\alpha^i)$두 친 서치의 결과 데이터를 이

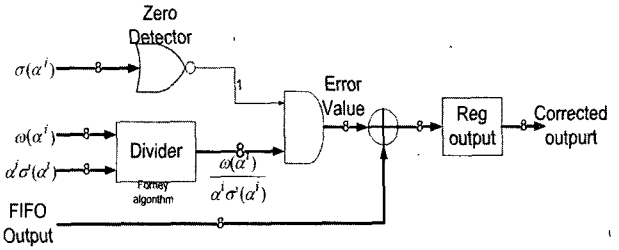

그림 5. 포니 알고리즘 & 에러 정정 블록

Fig. 5. Forney algorithm & Error correction block.

용하여 실질적인 에러의 값을 계산하게 된다. 에러위치 다항식의 결과 $\sigma(\alpha^i)$가 0이 나오면 전송된 데이터의 해당 위치 심벌에 에러가 있음을 의미하므로 전송받아 FIFO에 저장되어 있던 데이터에 포니 알고리즘 블록으로부터 나온 에러값을 더하면 에러를 정정한 데이터를 출력하게 된다.

5. FIFO(First in First out)

에러의 정정을 목적으로 전송받은 데이터 $I(x)$를 FIFO가 저장하고 있다. 하지만 RS 복호기가 패리티 데이터를 제외한 실질적 데이터만을 출력하므로 255 Byte를 저장하는 대신 패리티를 제외한 239 Byte만을 저장하면 된다. 또한 DMB 서비스는 실시간 방송서비스이므로 데이터가 일정한 전송율을 가지고 지속적으로 들어온다. 그러므로 들어온 하나의 데이터 패킷의 에러를 정정하는 동안 연이어 들어오는 다음 데이터 패킷을 저장하고 있어야만 하므로 239 Byte의 FIFO 두 개를 두어 연속적인 데이터 에러정정이 가능하도록 하였다.

Ⅳ. 유클리드 알고리즘의 개선

앞선 Ⅲ장 2절에서는 기본적인 유클리드 알고리즘의 동작을 설명하였다. 하지만 기본적인 유클리드 알고리즘을 하드웨어로 구현하는 것에는 세 가지의 문제점이 있다.

첫째는 연산중인 다항식들의 차수를 지속적으로 계산하여야 한다는 것이다. 유클리드 알고리즘은 다항식과 다항식 사이의 나눗셈과 곱셈을 기본연산으로 한다. 나눗셈 연산의 경우 제수 다항식의 최고 차수 항 계수의 역수를 구하여 제수 다항식에 곱한 후 피제수와의 차수 차이만큼 제수의 차수를 올려 더하는 방식으로 이루어진다. 하지만 연산이 반복되며 제수와 피제수 다항식의 차수는 데이터에 따라 랜덤하게 변화 하여 지속적으로 각 다항식들의 차수를 계산하여야 한다. 차수 계

산에 순수한 연산 동작보다 많은 클럭이 사용되는 문제가 있다.

둘째는 다항식 사이의 나눗셈 연산이 끝난 후 유클리드 알고리즘의 동작을 끝마칠 순간을 알기 위해서 나머지 다항식의 차수가 t보다 작은지를 확인하여야 한다. 연산이 끝난 후에도 레지스터 검색을 통해 나머지 다항식의 차수를 알아야 연산을 반복할 것인지 아니면 모든 연산을 끝마치고 최종 결과를 출력할 것인지 알 수 있다. 이 역시 클럭을 낭비하여 Key equation을 풀이하는데 걸리는 시간을 늘어나 데이터처리 지연시간이 늘어나는 결과를 초래 한다.

셋째는 유클리드 풀이과정에서 $GF(q)$상의 나눗셈 연산을 위해 Inverse ROM이 사용된 다는 점이다. 나눗셈 연산의 경우 제수의 역수를 구하여 피제수에 곱하는 형식으로 나눗셈 연산을 수행한다. 역수를 구하기 위한 특별한 방법이 존재치 않으므로 모든 수에 대한 역수를 저장하고 있는 Inverse ROM이 필요하다. 심벌의 크기가 8 Bit 이므로 255 Byte의 크기를 가지는 Inverse ROM이 필요하다. 이는 최장지연시간을 늘이고 유클리드 알고리즘 블록의 면적을 증가시키는 요인이 된다.

세 가지의 문제를 해결하기 위하여 다음과 같이 유클리드 알고리즘에 수정을 가하였다. 첫 번째로 Inverse ROM 대신 17clk이 소요되는 유한체 나눗셈기를 만들어 사용하였다. 본 나눗셈기는 유클리드 알고리즘을 이용하여 제수의 역수를 구하지 않고도 $GF(q)$상에서 나눗셈 연산을 할 수 있다.[4] 역수를 구하기 위한 Inverse ROM을 없어 유클리드 알고리즘 블록의 면적이 줄어드나 나눗셈 연산을 위하여 17clk이 소모되어 사용되는 클럭이 수가 늘어나는 문제를 가진다. 하지만 뒤에서 제안한 차수연산이 없는 유클리드 알고리즘을 사용하므로 차수연산에 낭비되던 클럭을 줄여 기존 유클리드 알고리즘과 동일한 클럭을 소모한다.

차수연산이 없는 수정된 유클리드 알고리즘을 아래에 나타내었다. 앞서 설명한 바와 같이 유클리드 알고리즘 블록은 신드롬 다항식$S(x)$를 입력으로 받아 Key equation을 풀이한다. $S(x)$는 16 Byte의 크기를 가지고 레지스터 R에 아래와 같이 저장된다. 출력 위치는 고정되어 있어 에러값 다항식을 구하기 위한 데이터는 R[15]~R[8]에서 출력되며 에러위치 다항식을 구하기 위한 데이터는 T[8]~T[1]에서 출력된다.

```
H = 17'h10000, G = 17'h00000,  C = 16'h8000
U = V = {0,0,0,0,0,0,0,0,0} (9 Byte)
T = {0,0,0,0,0,0,0,0,1} (9 Byte)
D = {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0} (17 Byte)
R = {0, S(x)} (17 Byte)

While(1) begin
  G_m ↔ H_m
  R = R << 8, G = G >> 1, H = H << 1
  swap = (!G_m-1 | !mode) & R_m
  mode = R_m | (mode & !G_m-1)
  if (swap == 1) begin
      G[15 : 0] ↔ H[15 : 0]
      D ↔ R
  end
  if( rm == 1) begin
     Q = division(D[16], R[16]);
     for(i=0;i < 16;i++) begin
         D_temp[i] =mul(Q,D[i]);
         R[i] = R[i] ^ D_tmp[i];
     end
  end
  if (mode == 1 && rm == 1) begin
      for(i=0;i<10;i++) begin
          T_temp[i] = mul(Q, T[i]);
           U[i] = U[i] ^ T_temp[i];
      end
  end
  if (H_m == 1) begin
      for(i=0;i<9;i++)  U[i] = U[i] ^ V[i];
  end
  if (H_m == 1) begin
      for(i=0;i<9;i++) begin
          V[i] = T[i];
          T[i] = U[i];
          U[i] = 0;
      end
  end
  if ( mode == 1 ) U = U << 8;
  if ((mode == 1 & swap == 1)
      | (mode == 0 & swap == 0) ) C = C >> 1;
  if (C_state == 1 && mode==1 && swap==1) break;
end
```

$$\omega(x) = \left\{ \frac{R[15]}{T[0]}, \frac{R[14]}{T[0]}, \cdots, \frac{R[8]}{T[0]} \right\} \text{ (8 Byte)}$$

$$\sigma(x) = \left\{ \frac{T[8]}{T[0]}, \frac{T[7]}{T[0]}, \cdots, \frac{T[1]}{T[0]} \right\} \text{ (8 Byte)}$$

수정된 유클리드 알고리즘 동작이 끝난 후 에러위치 다항식과 에러값 다항식은 다음과 같이 정의된다. 에러위치 다항식은 8차 다항식으로 9 Byte의 출력을 가져야 하나 최하위 차수의 계수 데이터는 어떠한 경우에서도 1이므로 출력하지 않는다.

수정된 유클리드 알고리즘을 이용해 구현된 유클리드 알고리즘 블록을 그림 6에서 보여주고 있다. 유클리

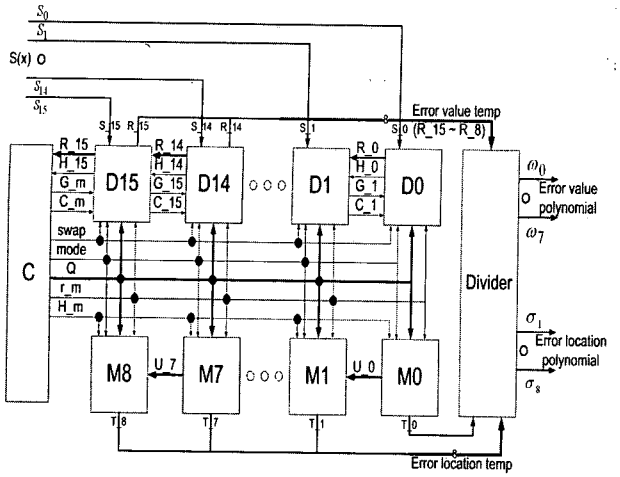

그림 6. 유클리드 알고리즘 블록
Fig. 6. The Euclid algorithm block.

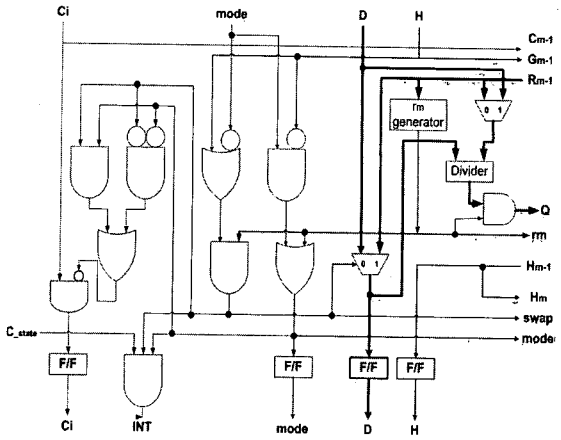

그림 7. 유클리드 알고리즘 블럭의 C_node
Fig. 7. C_node function of the Euclid algorithm block.

드알고리즘 블록은 C_node, D_node, M_node의 조합으로 이루어져 있고 C_node은 C로 D_node은 D15～D0로 M_node은 M8～M0로 표현하였다. 신드롬 다항식 $S(x)$는 D_node의 R 레지스터에 입력된다. 그림 상에서 왼쪽으로 데이터가 쉬프트 되며 Key Equation이 풀이된다. 연산이 끝나고 D15~D8의 R 레지스터에서 에러값 다항식의 임시 데이타가 출력되고 M8~M0의 T 레지스터에서 에러위치 다항식의 임시 데이타가 출력된다. 임시데이터는 나눗셈기로 입력되어 에러값 다항식과 에러위치 다항식을 출력한다.

다항식들 간의 나눗셈 연산중 다항식의 차수를 구하기 위하여 데이터 레지스터를 검색하는 문제는 최고차수의 위치를 그림 6상의 C_node로 고정시켜 해결 하였다. 그림 상에서 데이터를 왼쪽으로 쉬프트 시키고 C_node내에는 그림 7에서 볼 수 있듯이 나눗셈기가 있어 데이터가 C_node 도달하면 제수 다항식과 피제수 다항식의 최고차항의 계수를 나누어 몫 Q를 구한 후 이를 D_node에 보내 다항식 나누기 다항식의 연산을 수행한다. 하지만 연산 위치가 고정되어 있으므로 제수 다항식과 피제수 다항식간의 차수 차이를 알 수 없으므로 지금 이루어지고 있는 연산의 상황을 정확히 알 수 없게 만든다. 이를 해결하기 위하여 레지스터 H, G가 추가 되었다.

레지스터 H는 17bit로 그림 8에서 볼 수 있듯이 D_node에 1비트씩 16bit가 위치하고 최상위 비트 H_m만이 C_node 위치하고 H_m은 1로 초기화 되어 있다. G 역시 17bit로 최상위 비트 G_m을 포함하여 0으로 초기화 되어있다. 동작이 시작되면 H_m과 G의 최상위 비트를 교환한 후 H는 왼쪽으로 1 bit 쉬프트 되고 G는 오른쪽으로 1bit 쉬프트 된다. 이를 간단히 설명하자면 유클리드 알고리즘은 제수 다항식과 피제수 다항식의 차수 차이만큼 나누기 연산을 반복하여야 다항식 간의 나누기 연산이 완성된다. H와 G의 변화는 제수와 피제수상의 차수 차이의 변화를 나타낸다. H의 최상위 비트 H_m이 다시 1이 되면 이는 다항식 나누기 다항식의 연산이 끝났음을 의미한다. 나누기 연산을 통해 구해진 몫 Q는 M_node에도 전해져 및 에서 설명할 mode신호에 따라 다항식간의 곱하기 연산을 수행하며 그림 8에 나타내었다.

레지스터 H, G, R_m을 이용하여 mode, swap이란 신호를 만들어 사용 하는데 여기서 R_m은 R의 최상위 1 byte R[16]의 데이터 유무를 알려주며 데이터가 있다면 1이 되고 반대의 경우 0이 된다. mode가 1일 경우 다항식 나누기 다항식의 중간과정에서 나눗셈과 곱셈 등의 연산이 이루어져야함을 의미한다. swap이 1일 경우 D와 R의 데이터를 바꾸고 최상위 비트를 제외한 H와 G의 데이터를 바꾸게 된다. 신호 swap이 필요한 이유는 나눗셈 모듈에서 구해지는 Q를 위해 면적이 커지는 것을 막기 위해서 이다. 다항식과 다항식의 나누기이므로 Q역시 다항식의 형태를 가지고 있어 다수의 나눗셈 연산이 이루어져야 Q를 구할 수 있다. 이 때 제수 다항식이 중간과정에서 연산의 끝까지 보존 되어야 한다. 제수 다항식의 보존을 위하여 새로운 레지스터를 두어 면적을 낭비하는 대신 swap 신호를 이용하여 제수와 피제수가 저장되는 D, R 두 레지스터의 데이터를 서로 바꾸어 제수 다항식을 연산의 끝까지 보존할 수 있다.

다항식 나누기 다항식의 연산이 끝난 후 나머지 다항식의 차수를 계산하여 유클리드 알고리즘의 동작을 제

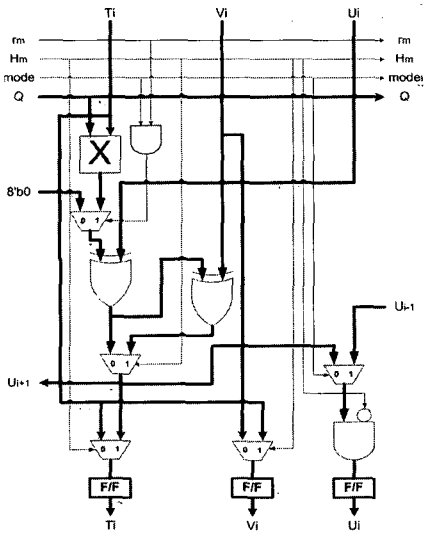

그림 9. 유클리드 알고리즘 블럭의 M_node
Fig. 9. M_node function of the Euclid algorithm bock.

어하는 문제는 C 레지스터를 두어 해결하였다. C의 최상위 비트는 1로 초기화 되어 있으며 mode와 swap 신호에 따라 왼쪽으로 쉬프트 된다. C 레지스터의 데이터가 왼쪽으로 한번 쉬프트 되는 것은 나머지의 차수가 연산과정에서 1만큼 떨어짐을 나타낸다. 그러므로 C 레지스터를 통해 정확한 타이밍에 연산을 끝낼 수 있다. 연산을 끝내는 동작을 보면 C_state의 값이 1이고 mode와 swap 모두 1이면 동작을 끝내도록 되어 있다. 여기서 C_state의 의미는 C 레지스터의 최하위 비트부터 여덟 번째 비트 사이에 1이 있는지 여부를 나타낸다. 1이 있다면 C_state 은 1이 된다. 이는 최종 나머지의 차수가 t보다 작음을 의미한다. 여기서 mode swap 모두가 1이 되어야 하는 이유는 출력 위치가 고정되어 있기 때문 이다. C_state이 1 인데 mode와 swap이 1이 아니라는 것은 데이터가 고정된 출력 위치에 도달하지 못했음을 의미한다. 그러므로 데이터가 쉬프트 되어 고정위치로 올 때까지 기다려야 한다. 이는 표 1에 잘 표현되어 있다.

표 1에서 C는 C레지스터의 몇 번째 비트가 1로 셋 되어 있는지를 의미하며 나머지의 차수를 의미한다. R은 동작중인 알고리즘 내에서 R 레지스터의 어떤 위치부터 데이터가 시작되는지를 보여준다. 예제들은 유클리드 알고리즘의 동작이 끝나는 여러 예제들을 보여주고 있다. 예제 1번의 경우 C_state이 1이 되는 순간에 mode와 swap이 1이 되어 출력 위치에 데이터가 도달되어 있음을 의미한다. 하지만 예제 2, 3, 4는 C_state이 1이 되었음에도 데이터가 출력 위치까지 도달하지 못하여 mode와 swap이 1이 될 때까지 기다려 줌을 보여준

표 1. 수정된 유클리드 알고리즘 동작
Table 1. Operation example of the modified Euclid algorithm.

Example 1				Example 2				Example 3				Example 4			
Mode	Swap	C	R 데이터 위치	Mode	Swap	C	R 데이터 위치	Mode	Swap	C	R 데이터 위치	Mode	Swap	C	R 데이터 위치
1	1	14	15	1	1	14	15	1	1	14	15	1	1	14	15
1	0		15	1	0		15	1	0		15	1	0		15
1	1	12	15	1	1	12	15	1	1	12	15	1	1	12	15
1	0		15	1	0		15	1	0		15	1	0		15
0	0		14	0	0		14	0	0		14	0	0		14
1	1	11	15	1	1	11	15	1	1	11	15	1	1	11	15
1	0		15	1	0		15	1	0		15	1	0		15
1	0		15	1	0		15	1	0		15	1	0		15
1	1	10	15	1	1	10	15	1	1	10	15	1	1	10	15
1	0		15	1	0		15	1	0		15	1	0		15
1	1	9	15	1	1	9	15	1	1	9	15	1	1	9	15
1	0		15	1	0		15	1	0		15	1	0		15
1	1	8	15	1	1	7	15	1	1	6	15	1	1	5	15
1	0		15	1	0		14	1	0		13	1	0		12
1	1	7	15	0	0		15	0	0		14	0	0		13
1	0		15	1	1	END		0	0		15	0	0		14
1	1	END						1	1	END		0	0		15
• 각 예제는 연산을 끝마치는 여러 경우를 보여주고 있다.												1	1	END	

다. 결국 R이 15가 된 이후에 유클리드 알고리즘의 동작을 끝마치게 된다.

다음과 같이 연산과정 중 그리고 연산이 끝나기 위한 차수 계산 과정을 모두 상태 레지스터에 저장하는 방법을 사용하여 클럭의 낭비를 줄일 수 있다. 나눗셈기의 사용으로 순수하게 연산을 위하여 필요한 클럭의 수는 늘어났지만 전체적으로 Key equation을 풀이하는데 필요한 클럭의 수는 기존 유클리드 알고리즘과 동일하다.

V. 결과 및 비교 분석

설계된 RS 복호기의 동작을 그림 10에서 보여 주고 있다. 전송된 첫 번째 데이터 패킷을 받는 동안 신드롬 계산 블록에서는 해당 패킷에 대한 신드롬 계산이 이루어진다. 첫 번째 데이터 패킷을 모두 전송 받으면 신드롬 계산 블록은 생성된 첫 번째 패킷에 대한 신드롬 데이터를 유클리드 알고리즘 블록에 넘기고 연속적으로 들어오는 두 번째 데이터 패킷을 처리한다. 두 번째 데이터 패킷이 전송되는 동안 유클리드 알고리즘 블록과 친서치 에러 정정 블록에서는 첫 번째 패킷에 대한 에러정정이 이루어져 입력된 255 byte 데이터 중 에러정정을 위해 붙은 패리티 16 byte가 제외된 239 byte의 에러가 정정된 데이터를 출력한다. 이는 본 논문의 RS 복호기가 DMB 서비스를 위해 연속적으로 들어오는 데이터를 처리할 수 있음을 보여준다.

하나의 패킷이 입력되었을 때 출력이 나오기 까지 생

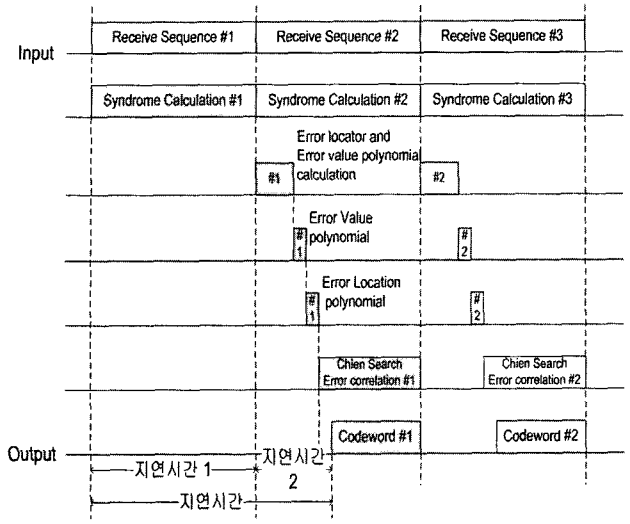

그림 10. RS 복호기 동작 타이밍도
Fig. 10. Action timing chart of the RS decoder.

표 2. RS 복호기 주요 블록들의 크기 비교
Table 2. The comparison between principal sub block of RS decoder.

Block	Syndrome Calculator	Chien Search	Euclid Algorithm
크기(gates)	6,757	8,430	30,228

기는 지연시간은 그림 10에서 보는바와 같다. 지연시간 1의 경우 고정된 데이터 전송율에 따라 하나의 패킷이 전송될 동안에 걸린 시간이므로 줄일 수 없으나 지연시간 2의 경우 유클리드 알고리즘이 동작하는 동안 걸린 시간이므로 유클리드 알고리즘에 걸리는 클럭의 수를 줄여 줄일 수 있다. 즉 유클리드 알고리즘 블록에 사용된 클럭의 수를 줄여 RS 복호기를 통과하며 생기는 데이터 처리 지연시간을 줄일 수 있다.

설계된 RS 복호기 주요 블록들의 크기를 표2에서 비교하고 있다. 이 결과는 삼성 STD130 0.18㎛ 라이브러리를 바탕으로 Synopsys 툴을 사용하여 얻은 결과이다. 유클리드 블록의 크기가 다른 블록에 비하여 월등히 큰 면적을 차지하고 있음을 확인 할 수 있다. 그러므로 유클리드 알고리즘 블록의 크기를 줄이는 것이 RS 복호기의 크기를 줄일 수 있는 가장 효율적인 방법임을 알 수 있다.

다른 유클리드 알고리즘 블록과 제안한 유클리드 알고리즘 블록을 표3에서 비교하였다. DMB를 위한 RS 복호기를 목표로 하였으므로 알맞은 비교 대상을 찾지 못하였다. 그래서 최근 논문 중 유클리드 알고리즘 또는 나눗셈을 사용하지 않는 수정된 유클리드 알고리즘

표 3. 유클리드 알고리즘 블록의 성능 비교
Table 3. The comparison performance of Euclid algorithm block.

디자인		제안한 구조	[8]	[7]	[5]
목적		DMB 휴대용 단말	높은 속도 낮은 복잡성	높은 속도	면적 최소화
구조	나눗셈	Divider	무	무	Inverse ROM
	Control	H, G, C 레지스터	차수 계산	차수 계산	차수 계산
# of clk		288	522	355	287
total # of gates		30,228	17,000	117,500	44,700
최대 데이터 전송율		1.2 Gbps @ 180 Mhz	5 Gbps @ 625 Mhz	5 Gbps @ 625 Mhz	2.4 Gbps @ 300 Mhz
1.8 Mbps 구현을 위한 최소 동작주파수		4.5 Mhz	7.8 Mhz	5.4 Mhz	4.5 Mhz

을 사용하였으며 RS(255,239), t=8의 같은 성능을 가진 RS 복호기의 설계 논문을 비교대상으로 선택하였다. 면적을 총 게이트 수로 표시하였고 지연시간의 비교를 위해 Key Equation 풀이에 사용된 클럭의 수를 표시하였다. 가능한 최대 데이터 전송율을 표시하였으며 동일한 동작 주파수상에서의 데이터 전송율과 DMB최대 데이터 전송율인 1.8Mbps를 만족시킬 수 있는 최소 동작주파수를 표기 하였다.

최대 데이터 전송율은 최장지연시간을 토대로 사용가능한 최대 동작 주파수를 사용할 때의 데이터 전송율을 의미한다. 본 논문에서 제안한 유클리드 블록이 가장 낮은 최대 데이터 전송율을 가지나 DMB는 최대 1.8 Mbps의 낮은 데이터 전송율을 필요로 히여 그 이상의 높은 데이터 전송율은 의미가 없다. 오히려 데이터 전송율을 높이기 위해서는 동작 주파수를 높여야 하므로 전력소비가 많아져 휴대용 단말에는 어울리지 않는다.

그림 6을 보면 255개의 심벌로 이루어진 하나의 패킷이 입력되는 동안 전 패킷에 대한 Key Equation 풀이와 에러가 정정된 239개 심벌의 출력이 끝난다. 출력에서는 에러의 정정을 위해 붙은 패리티 16개의 심벌이 제외된다. 즉 실시간 처리를 위해서는 16개의 심벌이 입력되는 동안 전 입력 데이터의 유클리드 알고리즘 동작과 친서치 블록에서 출력 데이터의 첫 번째 심벌을 위한 에러 유무, 에러값을 구하는 동작이 이루어져야 한다. 다음과 같은 동작이 가능한 동작 주파수를 계산해 보면 제안한 유클리드 알고리즘의 경우 약 4.5 MHz

이상의 동작 주파수만을 가지면 DMB가 원하는 최대 1.8Mbps의 데이터 전송율을 만족시킬 수 있다. 이는 다른 비교대상보다 낮은 수치이다. 더욱 낮은 동작 주파수로 DMB 성능을 만족 시킬 수 있으므로 좋은 전력 특성을 가진다 할 수 있다.

유클리드 알고리즘을 수정 없이 사용한 경우[5] 나눗셈을 위하여 Inverse ROM을 사용하며 제어방법으로는 차수연산을 사용하고 있다. 면적은 44,700 게이트로 제안한 구조에 비해 1.47배 크며 사용된 클럭의 수는 제안한 구조와 차이를 가지지 않았다. 면적이 더 작으므로 DMB에 사용하기에 더 적합하다 할 수 있다.

나눗셈을 사용하지 않는 수정된 유클리드 알고리즘을 사용한 경우 나눗셈 연산이 없지만 제어를 위하여 차수연산을 사용하고 있다. 시스톨릭 어레이 구조로 배열한 경우[7] 면적은 제안한 구조에 비해 3.88배 큰 면적을 가져 가장 큰 면적을 보이며 사용된 클럭의 수는 1.23배 더 소모한다. 면적과 지연시간 모두 제안한 유클리드 알고리즘이 DMB에서 사용하기에 적합하다.

재귀(Recursive)구조로 배열한 경우는[8] 제안한 구조보다 0.56배의 작은 면적을 가지나 1.82배 더 많은 클럭을 소모한다. 면적에서는 [8]이 제안한 유클리드 알고리즘보다 우수하나 사용된 클럭의 수는 제안한 유클리드 알고리즘이 [8]의 50%정도만을 사용한다. 하지만 DMB는 면적과 사용된 클럭의 수를 모두 줄여야 하므로 단순히 면적만을 줄인 경우보다는 제안한 유클리드 알고리즘이 DMB에 더욱 적합하다.

DMB에서는 앞서 설명한대로 비교적 낮은 주파수를 사용하므로 사용된 클럭의 수가 많을수록 복호 지연시간은 늘어나게 된다. 또한 유클리드 알고리즘 블록에서 많은 클럭을 소모하면 동일한 크기의 FIFO 사용 시 더욱 높은 동작주파수를 가져야만 데이터의 실시간 처리가 가능하다. 그러므로 비교적 작은 면적과 적은 클럭을 사용하는 제안한 유클리드 알고리즘이 DMB에 더욱 적합하다.

VI. 결 론

본 논문에서는 DMB 서비스의 방송을 수신하는 휴대 단말기에서 전송도중 생긴 에러의 정정을 위한 RS 복호기의 면적을 줄이면서도 지연시간을 줄일 수 있는 수정된 유클리드 알고리즘을 제안하였다. 면적을 줄이기 위하여 나눗셈연산에 필요한 Inverse ROM을 17clk이 소요되는 유한체 나눗셈기로 대체 하였다. 나눗셈기를 사용하여 클럭의 수가 늘어나지만 차수 연산이 필요 없도록 유클리드 알고리즘을 수정하여 이를 보완 하였다. 유클리드 알고리즘 블록은 30,228개의 게이트를 사용하여 기본 유클리드 알고리즘 블록에 비해 30%의 면적감소를 가질 수 있었고 최근 주로 사용되는 나눗셈연산을 사용치 않는 수정된 유클리드 알고리즘에 비교하면 재귀구조를 가지는 경우 1.77배 큰 면적을 가지는 단점을 가지나 사용된 클럭의 수는 50%가량 줄일 수 있다.

DMB에서 사용하기 위해서는 면적뿐 만 아니라 실시간 처리를 위하여 지연시간 역시 짧아야 하므로 사용된 클럭의 수가 작아야 한다. 사용된 클럭의 수가 적으면 복호 지연시간을 줄일 수 있고 사용할 수 있는 최소 동작주파수 역시 낮출 수 있다. 그러므로 제안한 수정된 유클리드 알고리즘은 비교적 작은 면적과 적은 클럭을 사용하므로 DMB 휴대 단말에서의 사용에 더욱 적절하다. 또한 최대 동작주파수 180 Mhz로 1.2 Gbps의 최대 데이터 전송율을 가지므로 다른 통신 분야 및 다양한 응용분야에서 활용 가능하다.

참 고 문 헌

[1] 한국정보통신기술협회 , "초단파 디지털라디오방송(지상파 DMB) 비디오 송수신 정합표준," Doc. TTAS.KO_07.0026., 2004년 8월.

[2] 이만영, BCH 부호와 Reed-Solomon 부호, 민음사, 1990.

[3] S. Lin, Error Control Coding: fundamentals and Applications. Englewood Cliffs, NJ:Prentice-Hall, 1983.

[4] Eui-Seok Kim and Young-Jin Jeong, "A New Finite Field Division Algorithm.", Submitted to International Journal of Electronics, May, 2005.

[5] Hanho Lee, "An Area-efficient Euclidean Algorithm for Reed-Solomon Decoder", IEEE Computer Society Annual Symposium on VlSI, 2003.

[6] Dilip V. Sarwate, Naresh R. Shanbang, "High-Speed Architectures for Reed-Solomon Decoder.", IEEE Transaction on VLSI Systems, VOL. 9, NO. 5, October 2001.

[7] Hanho Lee, "High-Speed VLSI Architecture for Parallel Reed-Solomon Decoder.", IEEE Transaction on VLSI Systems, VOL. 11, NO. 2, April 2003.

[8] Hanho Lee, "A High-Speed, Low-Complexity Reed-Solomon Decoder for Optical Communi-

cation." , IEEE Transaction on Circuits and Systems : Accepted for future publication Volume PP, Issue 99, 2005.

저 자 소 개

류 태 규(준회원)
2004년 2월 광운대학교 전자통신 공학과 학사 졸업.
2006년 2월 광운대학교 전자통신 공학과 석사 졸업.
2006년 3월 코아로직 연구원
<주관심분야 : 무선통신, SoC 설계>

정 용 진(정회원)
1983년 서울대학교 제어계측 공학과 학사 졸업.
1983년 3월~1989년 8월 한국전자통신연구원
1995년 2월 미국 UMASS 전산공학과 박사
1995년 4월~1999년 2월 삼성 전자 반도체 수석 연구원
1999년 3월 광운대학교 전자공학부 부교수
<주관심분야 : 무선통신, 정보보호, 영상처리 및 인식, SoC 설계>