논문 2008-45SD-2-14

광통신 시스템을 위한 40Gb/s Forward Error Correction 구조 설계

(40Gb/s Foward Error Correction Architecture for Optical Communication System)

이 승 범*, 이 한 호**

(Seung-Beom Lee and Hanho Lee)

요 약

본 논문은 40Gb/s급 광통신 시스템에서 사용되는 고속 리드-솔로몬 (RS) 복호기의 하드웨어 면적을 줄인 새로운 구조를 소개하고 RS 복호기 기반의 고속 FEC구조를 제안한다. 특히 높은 데이터처리율과 적은 하드웨어 복잡도를 가지고 있는 차수 연산 블록이 제거된 pDCME 알고리즘 구조를 소개한다. 제안된 16채널 RS FEC구조는 8개의 신드롬 계산 블록이 1개의 KES 블록을 공유하는 8 채널 RS FEC구조 2개로 구성되어 있다. 따라서 4개의 신드롬 계산 블록에 1개의 KES블록을 공유하는 기존의 16채널 3-병렬 FEC 구조와 비교하여 하드웨어 복잡도를 약 30%정도 줄일 수 있다. 제안된 FEC 구조는 1.8V의 공급전압과 0.18-㎛ CMOS 기술을 사용하여 구현하였고 총 250K개의 게이트수와 5.1Gbit/s의 데이터 처리율을 가지고 400MHz의 클럭 주파수에서 동작함을 보여준다. 제안된 면적 효율적인 FEC 구조는 초고속 광통신뿐만 아니라 무선통신을 위한 차세대 FEC 구조 등에 바로 적용될 수 있을 것이다.

Abstract

This paper introduces a high-speed Reed-Solomon (RS) decoder, which reduces the hardware complexity, and presents an RS decoder based FEC architecture which is used for 40Gb/s optical communication systems. We introduce new pipelined degree computationless modified Euclidean (pDCME) algorithm architecture, which has high throughput and low hardware complexity. The proposed 16 channel RS FEC architecture has two 8 channel RS FEC architectures, which has 8 syndrome computation block and shared single KES block. It can reduce the hardware complexity about 30% compared to the conventional 16 channel 3-parallel FEC architecture, which is 4 syndrome computation block and shared single KES block. The proposed RS FEC architecture has been designed and implemented with the 0.18-㎛ CMOS technology in a supply voltage of 1.8 V. The result show that total number of gate is 250K and it has a data processing rate of 5.1Gb/s at a clock frequency of 400MHz. The proposed area-efficient architecture can be readily applied to the next generation FEC devices for high-speed optical communications as well as wireless communications.

Keywords: Error correction coding, area-efficient, Reed-Solomon decoder, fiber optic, communications.

I. 서 론

리드-솔로몬 (Reed-Solomon (RS)) 부호는 마그네틱, 광 저장매체, 유선 및 위성 통신 등 다양한 응용분야에 널리 쓰이는 Forward Error Correction (FEC) 기술이다. 8 바이트 오류정정(error correction) RS(255,239) 코드는 해저 광섬유 시스템을 위해 국제통신연합(ITU)에 의해 채택되었다[1]. 현재 가장 일반적으로 사용되는 RS 복호기(decoder) 구조는 오류를 감지하고 정정하는 세 개의 주요한 부분으로 구성되어있다. 첫 번째 부분은 Syndrome Computation(SC) 블록이다. SC블록에서는 신드롬 다항식 (syndrome polynomial) $S(x)$를 발생시

* 학생회원, ** 정회원, 인하대학교 정보통신공학부
(School of Information and Communication Engineering, Inha University)

※ 본 연구는 2007년도 정부(과학기술부)의 재원으로 한국과학재단의 지원(No. R01-2006-000-10596-0)을 받아 수행된 연구임

접수일자: 2007년12월5일, 수정완료일: 2008년1월28일

키고, 수신된 코드워드(code word)의 오류패턴을 표현한다. 다항식 $S(x)$는 RS 복호기의 두 번째 부분인 Key-Equation Solver(KES)블록에서 사용되어진다. KES블록에서는 키 등식 (Key equation) $S(x)\cdot\sigma(x)=\omega(x) \mod x^{2t}$을 해결하기 위해 유클리드(Euclidean) 알고리즘 (EA), 수정 유클리드(modified Euclidean) 알고리즘 (MEA), 또는 Berlecamp-Massay 알고리즘 (BMA) 등이 오류위치 다항식(error-locator polynomial) $\sigma(x)$와 오류 추정 다항식(error-evaluator polynomial) $w(x)$을 구하기 위하여 사용될 수 있다[2]. $\sigma(x)$와 $\omega(x)$의 두 다항식은 Chien Search 블록과 Forney 알고리즘 블록에서 오류의 위치와 오류 값을 구하기 위하여 사용된다. 복호기가 오류를 감지하고 정정하는 과정을 실행하는 동안 FIFO 메모리가 버퍼(buffer)로 사용된다. FIFO 메모리의 깊이(depth)는 복호기의 총 지연성(latency)과 관련이 있고 FIFO 메모리의 출력과 Forney 알고리즘 블록의 출력이 XOR 연산을 거쳐 오류 정정된 코드워드가 출력된다.

광통신 네트워크 시스템 구축을 위한 초고속 데이터 전송 기술은 높은 데이터율을 얻기 위한 요구와 맞물려 초고속 FEC 구조의 구현을 필요로 하게 되었다. Dense Wavelength Division Multiplexing (DWDM)의 출현과 함께 광전송 시스템은 지난 십여년간 급속도로 발전되어 왔으며 8바이트 오류정정능력에 기인한 RS(255,239) 코드가 고속(10-Gb/s 이상) 광전송 시스템에 일반적으로 사용되고 있다[3~5]. 그러나 광전송 시스템이 급속도로 발전함에 따라 40-Gb/s 이상의 고속 데이터 전송율을 필요로 하게 되었고, 이에 따라 하드웨어 복잡도와 전력소모가 매우 큰 현존하는 대부분의 RS 복호기는 시스템 레벨 통합의 어려움을 가져왔다.

본 논문에서는 차수의 직접적인 연산을 제거하고 다항식의 차수 편차를 이용하여 제어 신호를 생성하는 pDCME 알고리즘 블록과 16 채널 RS FEC 구조를 제안한다. 16 채널 RS FEC 구조는 2개의 8 채널 RS 복호기 구조로 구성되어 있고 8 채널 RS 복호기는 8개의 신드롬 계산 블록이 한 개의 KES블록을 공유하고 다시 KES 블록의 출력은 8개의 오류 정정 블록과 연결되는 형태이다.

Ⅱ. 표준화된 FEC 코드

리드-솔로몬 부호는 m이 2보다 큰 양의 정수일 때, m비트 열로 만든 심벌로 이루어진 비2진 순환 부호(non-binary cyclic code)이다. m비트 심벌에 대한 RS (n,k) 부호는 다음 조건을 만족시키는 모든 n과 k 값에 존재한다[10].

$$0<k<n<2^m+2 \tag{1}$$

여기서 k는 부호화될 데이터 심벌의 수이고, n은 부호화된 블록에 있는 부호 심벌의 수이다. 대부분의 기존 RS(n,k)부호는 다음과 같다.

$$(n,k)=(2^m-1,2^m-1-2t) \tag{2}$$

여기서 t는 부호의 심벌 오류 정정 능력이고, $n-k=2t$는 패리티 심벌의 수이다. RS 부호는 동일한 부호기 입력 및 출력 길이를 가진 어느 선형부호보다 최소거리 $d_{\min}$이 최대가 되는 부호이다. 비2진 부호에서 두 부호어 사이의 거리는 (해밍 거리와 마찬가지로) 부호어 열에서 서로 다른 심벌의 수로 정의한다. RS부호의 최소 거리는 다음과 같이 주어지고 부호는 어느 t개나 그 이하의 오류를 정정할 수 있는데, t는 다음과 같이 표현할 수 있다.

$$d_{\min}=n-k+1$$
$$t=\lfloor\frac{d_{\min}-1}{2}\rfloor=\lfloor\frac{n-k-1}{2}\rfloor \tag{3}$$

여기서 $\lfloor x\rfloor$는 x보다 크지 않은 가장 큰 정수를 의미한다.

8 바이트 오류정정 (error correction) RS(255,239) 부호는 해저 광섬유 시스템을 위해 국제통신연합(ITU)에 의해 채택되었다[1]. 여기서 RS 부호는 10^{-12}의 BER에서 대략 5.5 dB 네트 코딩게인(net coding gain)을 제공한다. 이는 10^{-4}의 입력 BER에 대해 10^{-15}의 출력 BER를 가지는 것을 의미한다. 특히 RS 부호는 연집 오류(burst errors)를 수정할 수 있으며, 하나의 RS(255, 239) 부호는 최대 64비트의 오류를 정정할 수 있다. 이것은 16개의 채널이 인터리빙되는 RS 부호에서는 더 긴 길이의 연집 오류를 정정할 수 있다는 것

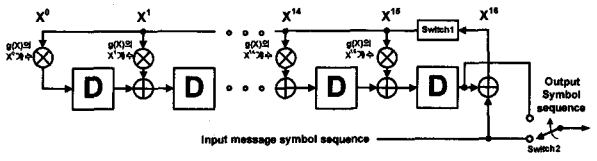


그림 1. RS 부호기.
Fig. 1. RS encoder.

을 보여준다. 16개의 채널에서 각각의 채널이 코드워드를 생성하고 각 코드워드의 심벌이 순차적으로 한 개씩 전송되면 16 채널 RS 부호는 최대 1024비트의 오류를 정정할 수 있게 된다. ITU-T G.975에서 이와 같이 1024비트의 오류를 정정할 수 있는 16 채널 RS 부호를 권장하고 있다.

Ⅲ. RS 부호기

수식 (4)는 파라미터 n, k, t와 임의의 양의 정수 $m>2$으로 RS 부호를 나타낸 가장 전형적인 형태이다. 여기에 반복하면 다음과 같다[10].

$$RS(n,k)=(2^m-1,2^m-1-2t) \tag{4}$$

여기서 $n-k=2t$는 패리티 심벌의 수이고, t는 부호의 심벌 오류 정정 능력이다. RS 부호의 생성 다항식은 다음의 형태를 취한다. ITU-T G.975에서는 $g(x)$의 값을 α^0부터 α^{15}으로 권장하고 있고 다음과 같은 형태가 된다[1].

$$g(x)=g_0+g_1x+\cdots+g_{2t-1}x^{2t-1}+g_{2t}x^{2t} \tag{5}$$

생성 다항식의 차수는 패리티 심벌의 수와 같다. 생성 다항식의 차수가 $2t$이므로, 다항식의 근은 정확히 $2t$개의 연속적인 α의 거듭제곱이어야 한다[10].

$$g(x)=(x-\alpha^0)(x-\alpha^1)\cdots(x-\alpha^{15}) \tag{6}$$

RS 부호는 순환 부호이므로 조직적인(systematic) 형태의 부호화는 메시지 다항식 $m(x)$에 x^{n-k}를 곱하고 패리티 다항식 $p(x)$를 더하는 형태가 된다. $p(x)$는 $m(x)\cdot x^{n-k}$를 $g(x)$로 나눈 나머지이다.

$$x^{n-k}m(x)=q(x)g(x)+p(x) \tag{7}$$

$$p(x)=x^{n-k}m(x) \bmod g(x) \tag{8}$$

부호화된 다항식 $u(x)$는 다음과 같다.

$$u(x)=x^{n-k}m(x)+p(x) \tag{9}$$

수식(7)의 $x^{n-k}m(x)$를 수식(9)에 대입하면 부호화된 다항식 $u(x)$는 $q(x)g(x)$가 되고 이는 부호화된 다항식에 $g(x)$의 근을 대입하면 0이 됨을 뜻한다.

그림 1에서 왼쪽에서 오른쪽까지의 곱하기 항들은 수식 (5)에 있는 다항식의 (낮은 차수에서 높은 차수로의) 계수들에 해당한다. 그림 1의 RS 부호기는 부호어를 조직적인 형태로 만들기 위해 다음의 절차를 따른다. 처음 k 클럭 주기 동안 메시지 심벌이 $(n-k)$단 이동 레지스터로 들어가도록 스위치 1은 닫고 메시지 심벌이 동시에 바로 출력 레지스터로 가도록 처음 k 클럭 주기 동안 스위치 2는 아래 위치에 둔다. k번째 메시지 심벌이 출력 레지스터로 이동한 후, 스위치 1은 열고, 스위치 2의 위치는 위로 바꾼다. 나머지 $n-k$ 클럭 주기 동안 이동 레지스터에 있던 패리티 심벌을 출력 레지스터로 내보낸다. 메시지가 부호화되어 출력으로 완전히 나오기까지 n 클럭의 시간이 걸리고 출력은 수식 (9)와 같이 된다.

Ⅳ. RS 복호기

부호화를 통해 얻은 부호어 다항식은 전송하는 동안 잡음에 의해 오염되어 심벌의 오류로 수신되었다고 가정하면 오류 패턴은 다음과 같이 다항식으로 나타낼 수 있다.

$$e(x)=\sum_{i=0}^{n-1}e_ix^i \tag{10}$$

그러면 오염되어 수신된 부호어 다항식 $r(x)$는 다음과 같이 송신 부호어 다항식과 오류 패턴 다항식의 합으로 나타낼 수 있다.

$$r(x)=u(x)+e(x) \tag{11}$$

수신된 부호어 다항식은 신드롬 계산 블록, KES 블록, Chien Search 및 오류 정정 블록을 거쳐 오류 정정된 코드워드를 출력으로 내보낸다.

1. Syndrome Computation (SC) 블록

신드롬 계산은 수신된 다항식 $r(x)$가 오류에 오염되었는지 아닌지를 판단하고 오류에 의해 오염되었을 때 오류 패턴을 찾을 수 있는 값이 된다. 신드롬 계산은 수신된 다항식 $r(x)$에 $g(x)$의 모든 근을 대입하는 형태로 이루어진다. $r(x)$가 합법적인 부호어라면 $r(x)$는 $u(x)$가 되고 $u(x)$는 $g(x)$로 나누어지므로 신드롬 계산의 결과는 0이 된다. 신드롬의 계산 결과가 0이 아니면 수신된 다항식이 오류로부터 오염되었음을 나타낸다. 신드롬 S는 $n-k$개의 심벌 S_i, $(i=0,\cdots,n-k-1)$로 구성되었고 신드롬 심벌의 계산은 다음과 같이 표현할 수 있다.

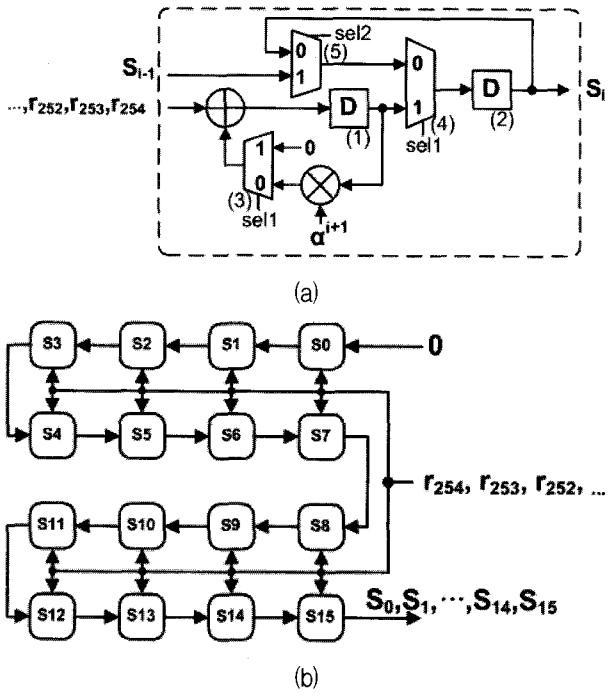


그림 2. (a) 신드롬 셀,
(b) 신드롬 다항식을 구하는 신드롬 계산 블록
Fig. 2. (a) Syndrome cell,
(b) Syndrome Computation block.

$$S_i = r(x)|_{x=\alpha^i} = r(\alpha^i) \quad i = 0, \cdots, n-k-1 \quad (12)$$

신드롬 계산 블록은 신드롬 값 S_i를 구하고 이를 다항식 $S(x)$의 형태로 출력으로 내보내는 역할을 한다. 수신된 코드워드 $r(x)$의 계수 값들이 1 클럭 사이클마다 하나씩 입력되므로 수식 (12)를 수식 (13)의 반복적인 형태로 변형하고 이를 이용하여 신드롬 계산 블록을 구현할 수 있다.

$$r(\alpha^i) = ((\cdots (r_{254}\alpha^i + r_{253})\alpha^i \cdots + r_1)\alpha^i + r_0 \quad (13)$$

16개의 신드롬 값은 수신된 다항식 $r(x)$에 대입되는 입력 α^i의 차수만 다르기 때문에 신드롬 값을 구하는 그림 2(a)의 신드롬 셀 블록과 같은 구조가 된다. 먼저 수신된 다항식 $r(x)$의 최고차항의 계수 r_{254}는 레지스터 (1)과 레지스터 (2)의 값에 관계없이 레지스터에 저장이 되어야 하므로 멀티플렉서 (3)의 셀렉트(select) 신호 $sel1$은 1이 되고 레지스터 (1)에는 r_{254}가 저장이 된다. 이후 r_{253}부터 r_0까지의 계수 값은 클럭 사이클마다 차례로 신드롬 블록의 입력이 되고 레지스터 (1)에 저장된 값에 α^i를 곱한 것과 입력 계수를 더하고 다시 레지스터 (1)에 저장하는 반복적인 연산이 수행된다. 이 때 $sel1$ 신호는 0이 되어야 누적된 값이 레지스터 (1)에 저장이 된다. 255클럭 동안 연산을 수행하면 16개의 신드롬 셀의 레지스터 (1)에는 S_i가 저장이 되고 멀티플렉서 (4)의 셀렉트 신호 $sel1$에 의해서 레지스터 (2)로 보내진다. 블록의 첫 데이터가 입력으로 들어오는 시기와 255클럭 동안의 연산으로 구해진 S_i가 레지스터 (2)로 전달되는 시기가 같기 때문에 멀티플렉서 (3)과 멀티플렉서 (4)는 같은 셀렉트 신호를 사용한다. 신드롬 계산 블록은 신드롬 다항식의 최고차항을 계수부터 순차적으로 출력으로 내보내기 때문에 16개의 신드롬 셀은 그림 2(b)와 같이 연결된다. 신드롬 셀의 레지스터 (2)는 쉬프트 레지스터의 역할을 하며 멀티플렉서 (5)의 셀렉트 신호 $sel2$에 의해서 쉬프트를 할지 현재 값을 유지할지를 결정한다. $sel2$ 신호가 1이 되면 신드롬 다항식이 신드롬 계산 블록의 출력으로 나오게 된다.

2. Chien Search 및 오류 정정 블록

수정 유클리드 알고리즘을 이용하여 오류 위치 다항식을 구하였고 오류 위치다항식은 정의에 따라 오류위치의 역수를 근으로 가진다. 이 근을 알면 오류 위치도 알게 된다. 그러므로 $\sigma(x)$의 유한체(Galois Field) 모든 원소들을 $\sigma(x)$에 대입함으로써 근을 구하고 이를 이용하여 오류의 위치를 알 수 있다. RS 부호의 첫 심벌은 x^{254} 위치에 해당하고 이 위치에서 오류가 발생했는지 확인하기 위해 α^{254}의 역수 α^1을 $\sigma(x)$의 근으로 대입하고 α^2, α^3, $\cdots$, α^{255}순으로 검사한다.

오류의 값은 오류 추정 다항식과 오류 위치 다항식의 미분 다항식으로 얻을 수 있다. 오류 위치 다항식 $\sigma(x)$의 미분 다항식 $\sigma'(x)$는 다음과 같이 표현할 수 있다. 이 때 $X_l = \alpha^{i_l}$, $Y_l = e^{i_l}$이고 각각 오류의 위치와 오류의 값을 나타낸다.

$$\sigma'(x) = -\sum_{i=1}^{v} X_i \prod_{l \neq i} (1 - X_l x) \quad (14)$$

$\sigma'(x)$와 $\omega(x)$를 이용하여 오류 값 Y_l을 구할 수 있다.

$$\begin{aligned} \sigma'(X_l^{-1}) &= X_l \prod_{l \neq i} (1 - X_l x) \\ \omega(X_l^{-1}) &= Y_l \prod_{l \neq i} (1 - X_l x) \\ Y_l &= \frac{\omega(X_l^{-1})}{\sigma'(X_l^{-1}) X_l^{-1}} \end{aligned} \quad (15)$$

수신된 다항식의 계수 값이 r_{254}부터 입력이 된 것과

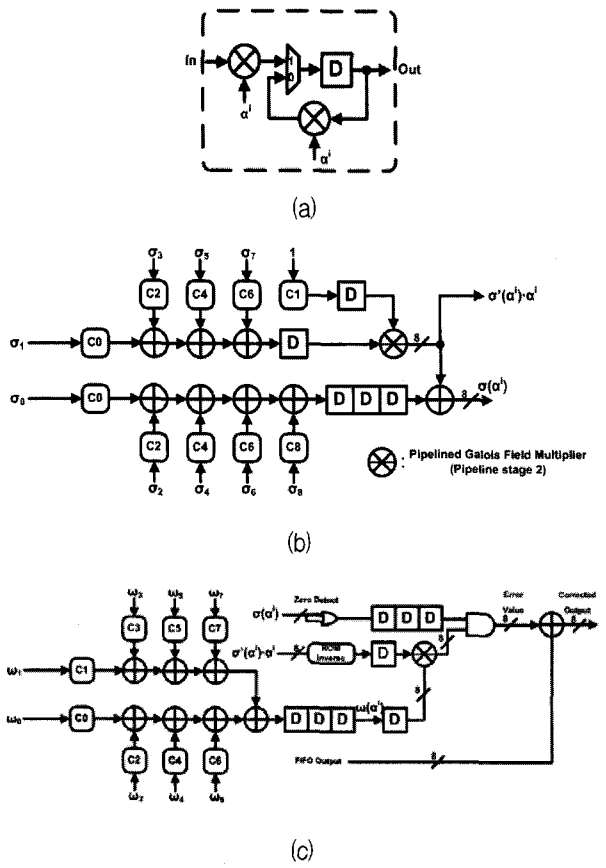


그림 3. (a) 기본 셀, (b) Chien Search 블록,
(c) Forney 알고리즘 및 오류 정정 블록.

Fig. 3. (a) Basic cell, (b) Chien search block,
(c) Forney algorithm & error correction block.

마찬가지로 오류가 정정된 후의 오류 정정된 다항식 $c(x)$도 높은 차수의 계수부터 출력되어야 한다. r_{254}에 해당하는 위치는 a^{254}로 표현되었고 오류 위치 다항식은 오류 위치의 역수를 근으로 가지기 때문에 r_{254}에 오류가 발생했는지 여부를 확인하기 위해 오류 위치 다항식의 근으로 α^1을 대입하여야 한다. 그러므로 오류 위치 다항식에는 α^1부터 $\alpha^{255}(=\alpha^0)$까지 차례로 대입된다. 오류 위치에 해당하는 오류 값을 구하기 위해 수식 (15)에서 보는 바와 같이 오류 추정 다항식 $\omega(x)$와 오류 위치 다항식의 미분 다항식 $\sigma'(x)$에도 α^1부터 $\alpha^{255}(=\alpha^0)$을 차례로 대입하여야 한다. 오류 위치 다항식 $\sigma(x)$와 오류 위치 다항식의 미분 다항식 $\sigma'(x)$는 각각 다음과 같이 표현될 수 있다.

$$\begin{aligned}\sigma(x) &= \sigma_t x^t + \sigma_{t-1}x^{t-1} + \cdots + \sigma_1 x^1 + \sigma_0 \\ \sigma'(x) &= t\sigma_t x^{t-1} + (t-1)\sigma_{t-1}x^{t-2} + \cdots + \sigma_1\end{aligned} \quad (16)$$

$GF(2^8)$에서의 연산에서 $\alpha+\alpha=2\alpha=0$이므로 오류 위치 다항식의 짝수차수의 항은 미분 후 모두 0이 된다. 그러므로 오류 위치 다항식의 미분 다항식 $\sigma'(x)$를 다시 표현하면 다음과 같다.

$$\sigma'(x) = \sigma_{t-1}x^{t-2} + \sigma_{t-3}x^{t-4} + \cdots + \sigma_1 \quad (17)$$

수식 (17)를 이용하여 $\sigma(x)$를 다시 표현하면 다음과 같게 되고 $x\sigma'(x)$는 오류 값을 구할 때도 사용된다.

$$\begin{aligned}\sigma(x) &= \sigma_t x^t + \sigma_{t-2}x^{t-2} + \cdots + \sigma_0 \\ &\quad + (\sigma_{t-1}x^{t-2} + \sigma_{t-3}x^{t-4} + \cdots + \sigma_1)x \\ &= \sigma_t x^t + \sigma_{t-2}x^{t-2} + \cdots + \sigma_0 + x\sigma'(x)\end{aligned} \quad (18)$$

오류 위치 다항식의 근은 α^1부터 α^{255}까지 클럭 사이클마다 α^1배 커지는 형태로 대입된다. $\sigma_t x^t$항을 보면 근이 α^1일 때 $\sigma_t(\alpha^1)^t$가 되고 근이 α^2일 때 $\sigma_t(\alpha^2)^t$가 된다. 즉, 매 클럭 사이클마다 $\sigma_t(\alpha^1)^t$에 α^t가 곱해지는 구조가 된다. 그림 3(a)의 기본 셀은 오류 위치 다항식의 각 항을 구하는 블록이고 각 항의 차수에 따라 곱해지는 α^i 값만 달라진다. 그림 3(b)는 오류 위치의 값을 구하는 Chien Search 블록을 보여주며, 기본 셀을 차수 별로 적용하여 수식 (18)을 구한다.

Forney 알고리즘 블록은 오류 추정 다항식 $\omega(x)$를 $x\sigma'(x)$로 나누는 연산을 수행하는데 $x\sigma'(x)$는 Chien Search 블록으로부터 입력 받고 $\omega(x)$를 구하는 블록은 오류 위치 다항식과 마찬가지로 기본 셀을 이용하여 구현할 수 있다. Forney 알고리즘 블록은 나누기 연산이 포함되어 있지만 $x\sigma'(x)$의 역수를 구하면 나누기를 곱하기로 구현할 수 있으므로 그림 3(c)와 같이 역수를 구하는 ROM과 곱셈기를 이용하였다. 오류 위치 다항식에서 해당 위치의 오류가 발생했는지 아닌지 여부를 그 값이 0인지 아닌지로 판단하고 0이면 추정한 오류 값과 FIFO를 거쳐서 나온 오류가 포함된 심벌을 XOR 연산을 통해서 오류를 정정하고 최종 출력을 얻을 수 있다.

3. Key Equation Solver (KES) 블록

가. 기존의 Modified Euclidean (ME) 알고리즘 블록

수정 유클리드 알고리즘은 최고차항을 소거하여 반복적으로 차수를 줄이는 방식으로 오류 위치 및 오류 값 다항식을 구하기 위해 수식 (19)과 같이 초기 값을 정한다. $S(x)$는 앞서 언급한 신드롬 다항식이다.

$$R_0(x) = x^{2t},\ Q_0(x) = S(x),\ L_0(x) = 0,\ U_0(x) = 1 \quad (19)$$

$$R_i(x)=[\sigma_{i-1}b_{i-1}R_{i-1}(x)-\overline{\sigma_{i-1}}a_{i-1}Q_{i-1}(x)] \\ -x^{|l_{i-1}|}[\sigma_{i-1}a_{i-1}Q_{i-1}(x)-\overline{\sigma_{i-1}}b_{i-1}R_{i-1}(x)] \quad (20)$$

$$Q_i(x)=\sigma_{i-1}Q_{i-1}(x)-\overline{\sigma_{i-1}}R_{i-1}(x) \quad (21)$$

$$L_i(x)=[\sigma_{i-1}b_{i-1}L_{i-1}(x)-\overline{\sigma_{i-1}}a_{i-1}U_{i-1}(x)] \\ -x^{|l_{i-1}|}[\sigma_{i-1}a_{i-1}U_{i-1}(x)-\overline{\sigma_{i-1}}b_{i-1}L_{i-1}(x)] \quad (22)$$

$$U_i(x)=\sigma_{i-1}U_{i-1}(x)-\overline{\sigma_{i-1}}L_{i-1}(x) \quad (23)$$

수식(20)~(23)은 반복적으로 i번 수행될 $R_i(x)$, $Q_i(x)$, $L_i(x)$, $U_i(x)$에 대해 나타낸 것이다. a_i와 b_i는 각각 $R_i(x)$와 $Q_i(x)$의 최고차항의 계수이고 다항식 $R_i(x)$, $Q_i(x)$, $L_i(x)$, $U_i(x)$의 값은 다항식 $R_{i-1}(x)$, $Q_{i-1}(x)$의 차수에 따라 값이 결정된다. $\deg(R_i(x))$와 $\deg(Q_i(x))$는 각각 다항식 $R_i(x)$, $Q_i(x)$의 차수를 나타낸다. $\deg(R_i(x))<\deg(L_i(x))$가 되면 반복 연산을 멈추게 되고 그 때의 $R_i(x)$와 $L_i(x)$가 각각 오류 추정 다항식과 오류 위치 다항식이 된다.

$$l_{i-1}=\deg(R_{i-1}(x))-\deg(Q_{i-1}(x)) \quad (24)$$

$$\sigma_{i-1}=\begin{cases}1, & \text{if } l_{i-1}\geq 0\\ 0, & \text{if } l_{i-1}<0\end{cases} \quad (25)$$

수정 유클리드 알고리즘은 위의 수식 (20)~(23)까지 반복적으로 연산을 하는 것이고 기존의 방식은 수식 (24)의 두 다항식 $R_{i-1}(x)$와 $Q_{i-1}(x)$의 차수를 계산하고 l_{i-1}을 구하여 수식 (20)~(23)이 바르게 연산될 수 있도록 제어 신호를 만들어 주었다. 제어 신호는 반복 연산의 종료 조건인 $\deg(R_i(x))<\deg(L_i(x))$에 따른 정지($stop$) 신호도 포함하고 있다. 즉, 기존의 ME 알고리즘 블록은 차수 계산에 따라 제어 신호를 생성하고 제어 신호에 따라 연산을 수행하는 것이다[5~6].

나. 제안된 Pipelined Degree Computationless Modified Euclidean(pDCME) 알고리즘 블록

수식 (20)은 σ_{i-1}의 값이 정해지면 수식 (26) 또는 (27)의 형태가 된다.

$$R_{i-1}(x)=b_{i-1}R_{i-1}(x)-x^{|l_{i-1}|}a_{i-1}Q_{i-1}(x) \quad (26)$$

$$R_{i-1}(x)=a_{i-1}Q_{i-1}(x)-x^{|l_{i-1}|}b_{i-1}R_{i-1}(x) \quad (27)$$

$R_{i-1}(x)$와 $Q_{i-1}(x)$의 값을 서로 바꾸면 a_{i-1}과 b_{i-1}

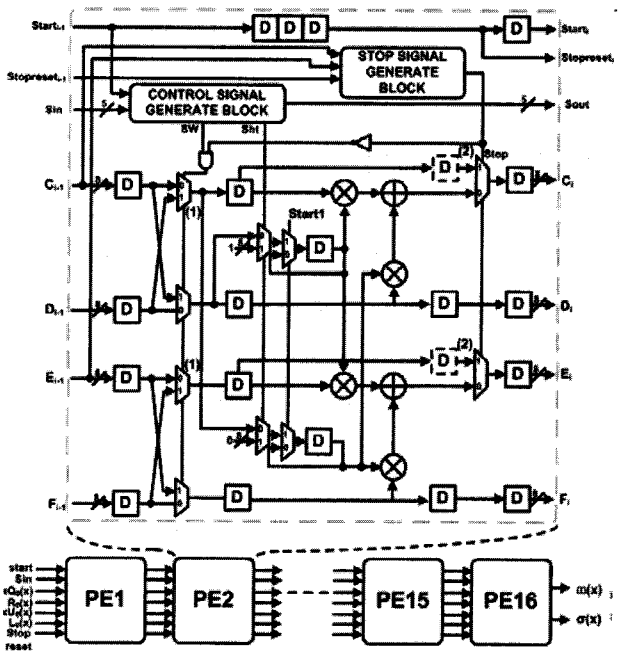


그림 4. Pipelined Degree Computationless Euclidean (pDCME) 알고리즘 블록.

Fig. 4. Pipelined Degree Computationless Euclidean (pDCME) algorithm block.

의 값이 함께 바뀌기 때문에 수식 (27)은 다시 수식 (26)의 형태로 표현할 수 있다[9]. 같은 방법으로 수식 (21)~(23)을 바꿀 수 있다. 그러므로 ME 알고리즘의 연산은 $R_{i-1}(x)$와 $Q_{i-1}(x)$의 값을 서로 바꾸는 swap 연산, $x^{|l_{i-1}|}$를 위한 지연 연산, $R_{i-1}(x)$의 최고차항의 계수를 소거하는 다항식 연산으로 구분할 수 있다.

그림 4는 ME 알고리즘의 연산을 수행하는 Processing Element(PE)를 나타낸다. PE 블록의 'sw', 'sht' 신호는 ME 알고리즘 연산을 가능하게 하고 '$stop$' 신호에 의해 연산이 종료 된다.

기존의 ME 알고리즘은 실제 다항식의 차수를 구하여 'sw', 'sht', '$stop$' 제어 신호를 생성하였지만 pDCME 알고리즘 블록은 $R_{i-1}(x)$와 $Q_{i-1}(x)$의 차수를 계산하고 비교하는 연산 없이 ME 알고리즘의 연산을 수행할 수 있도록 구현한 것이다. DCME 알고리즘은 초기 입력이 정해져 있고 입력 다항식을 알 때 연산 결과의 범위를 어느 정도 유추할 수 있다는 점을 이용하여 수식 (24)의 직접적인 차수 비교 연산 대신에 입력 패턴과 그 결과에 따른 상대적인 차수 가감을 통해 차수 비교를 할 수 있다. 즉, 차수의 비교 연산 없이 앞서 언급한 세 개의 제어 신호를 생성할 수 있다. 각 PE의 입력은 다음의 네 가지 패턴을 벗어나지 않으며 입력 패턴의 특징을 이용하여 제어 신호를 생성한다.

$$R_0(x) = x^{2t} \quad \text{for all cases}$$

$$Q_0(x) = \sum_{i=0}^{2t-1} S_i x^i \quad S_i \neq 0 \quad (0 \le i \le 2t-1) \tag{28}$$

$$Q_0(x) = \sum_{i=0}^{2t-1} S_i x^i \begin{cases} S_i = 0 & (2t-1-k \le i \le 2t-1,\ k \ge 0) \\ S_i \neq 0 & otherwise \end{cases} \tag{29}$$

$$Q_0(x) = \sum_{i=0}^{2t-1} S_i x^i \quad \begin{cases} S_i = 0 & (2t-1-m \le i < 2t-1,\ m \ge 1) \\ S_i \neq 0 & otherwise \end{cases} \tag{30}$$

$$Q_0(x) = \sum_{i=0}^{2t-1} S_i x^i \quad \begin{cases} S_i = 0 & (2t-1-l \le i < 2t-1,\ l \ge 0) \\ S_i = 0 & (2t-1-l-n \le i \le 2t-1,\ n \ge 0) \\ S_i \neq 0 & otherwise \end{cases} \tag{31}$$

$R_0(x)$의 차수는 항상 $2t$이고 $Q_0(x)$의 차수는 $2t-1$이하이므로 두 입력 다항식의 차수를 같게 하는 지연 연산을 줄이기 위해 $Q_0(x)$에 x를 곱한 다항식이 $R_0(x)$와 함께 항상 PE1의 입력이 된다. $Q_0(x)$의 차수가 $2t-1$이하이므로 $xQ_0(x)$의 차수는 $2t$이하가 된다. $xQ_0(x)$의 차수가 $2t$미만일 때 다항식 연산을 하기 전에 지연 연산을 수행해야 하므로 PE1의 입력단 C에는 $xQ_0(x)$가 들어가고 입력단 D에는 $R_0(x)$가 들어간다. PE1의 스타트(Start) 신호와 동기하여 들어오는 C와 D 입력의 값이 0인지 아닌지를 관찰하면 초기 입력 다항식과 관계하여 두 입력 다항식 $R_0(x)$와 $xQ_0(x)$의 차수가 같은지 아닌지 여부를 알 수 있다. 수식 (28)의 패턴은 $R_0(x)$는 x^{2t}이고 $Q_0(x)$는 $S(x)$이며 S_{2t-1}은 0이 아니라고 가정하고 있다. 이 경우 PE1의 스타트 신호와 동기하여 들어온 두 입력 C와 D는 0이 아니므로 $R_0(x)$와 $xQ_0(x)$의 차수를 $16(2t=16)$이라고 판단할 수 있고 PE1은 그림 5(a)와 같이 스왑 연산과 다항식 연산이 수행된다. PE1의 D 출력은 한 클럭만큼 데이터를 지연시키므로 PE1의 D 출력단($Q_1(x)$)의 다항식의 차수는 15가 되고 PE1이 다항식 연산을 수행하였으므로 C 출력단($R_1(x)$)의 다항식의 차수는 15가 된다. PE2에서 $R_1(x)$와 $Q_1(x)$의 차수가 15로 같으므로 스타트 신호와 동기하여 들어온 C와D의 입력 값이 둘 다 0이 아니다. PE2는 두 입력 다항식이 같은 차수이기 때문에 다항식 연산을 수행한다. PE2의 D 출력단($Q_2(x)$)은 1 클럭만큼 지연되지만 이 경우 $Q_2(x)$의 차수가 감소되지 않고 C 출력단($R_2(x)$)에 x를 곱하는 효과를 가진다. 따라서 PE2의 C와 D 출력 다항식의 차수는 15가 된다. 입력 패턴에 따라 D 출력단의 지연은 C 출력단에 x를 곱하거나 D 출력단을 x로 나누는 역할을 한다. PE3의 두 입력 다항식의 차수는 15로 같지만 $R_2(x)$에 x가 곱해진 것을 알고 있기 때문에 실제 $R_2(x)$의 차수가 $Q_2(x)$보다 작다. 그러므로 스왑연산이 PE3의 다항식 연산 전에 이루어진다.

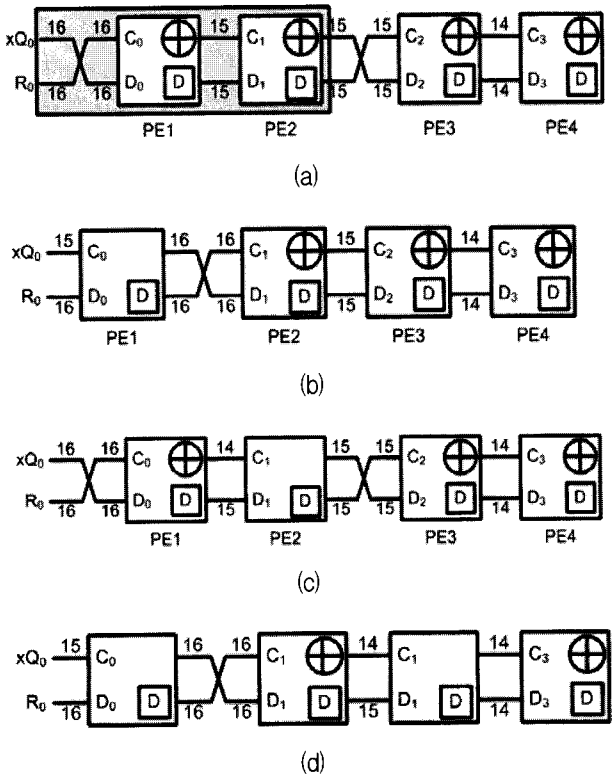


그림 5. (a) 수식(9), (b) 수식 (10), (c) 수식 (11), (d) 수식 (12)에 대한 Processing Element (PE)의 연산

Fig. 5. Processing Element (PE) operation for (a) equation (9), (b) equation (10), (c) equation (11), (d) equation (12).

수식 (29)의 경우에 스타트 신호와 동기하여 들어오는 C와 D의 입력 값 중 C의 값이 0이므로 $xQ_0(x)$의 차수가 15이하임을 알게 된다. 그러므로 PE1은 지연 연산을 수행하여 다항식 연산이 가능한 형태로 만들어야 한다. 그림 5(b)와 같이 PE1은 지연 연산을 수행하기위해 D 출력단을 한 클럭만큼 지연시키고 이것은 $xQ_0(x)$에 x를 곱하는 것과 같다. PE1의 C 출력단의 다항식의 차수는 16이하이고 PE2의 스타트 신호와 동기하여 C와 D의 입력 다항식이 모두 0이 아니라면 C 입력 다항식의 차수가 16이 되고 다항식 연산을 위해 스왑 연산을 먼저 수행한다.

수식 (30)의 경우에는 PE1은 수식 (28)와 같이 동작하지만 C 출력 다항식의 차수가 14이하가 되기 때문에

PE2가 그림 5(b) PE1과 같은 연산을 수행하여 그림 5(c)와 같이 된다.

수식 (31)에서 l과 n의 값이 1인 경우에 PE의 연산은 그림 5(d)에 나타나 있다. $xQ_0(x)$의 차수가 16이 아니므로 PE1은 수식 (29)의 PE1과 같은 연산을 수행하지만 PE2의 출력 다항식의 차수가 같지 않으므로 PE3은 지연 연산을 수행하여 두 다항식의 차수를 같게 한다. 위의 네 가지 패턴에 따른 PE의 연산은 스타트 신호와 동기하여 들어오는 C와 D의 입력 값에 의존하여 선택적으로 수행하였다. 그림 6(a)는 C와 D의 입력 값이 동시에 들어오는지 아닌지를 나타내는 SI (Simultaneously Input) 신호를 입력으로 하고 세 가지의 PE 연산을 결정하는 두 개의 제어 신호 Sw(Swap)과 Sht(Shift)를 출력으로 하는 Finite State Machine (FSM)을 나타낸다. S0 상태(State)는 초기 상태이며 S16 상태는 $S(x)$의 최고차항의 계수부터 연속적으로 8개의 계수가 0일 때 도달하는 상태이므로 채널에서 오류가 발생하지 않았음을 나타내는 상태이다. 오류가 t개 발생한 경우 $2t$개의 PE 블록을 거친 후에 오류 위치 다항식과 오류 값 다항식을 얻을 수 있다. 하지만 발생한 오류의 개수가 $v(<t)$개이면 $2v$개의 PE 블록의 연산 후에 오류 위치 다항식과 오류 추정 다항식을 구하게 되고 남은 PE 블록은 ME 알고리즘 연산을 중단하고 오류 위치 다항식과 오류 추정 다항식을 PE16의 출력으로 나오도록 쉬프트 연산을 수행한다. 앞서 언급한 바와 같이 ME 알고리즘의 연산 수행 중에 의 $R_i(x)$ 차수가 t 이하이거나 $L_i(x)$의 차수보다 작을 때

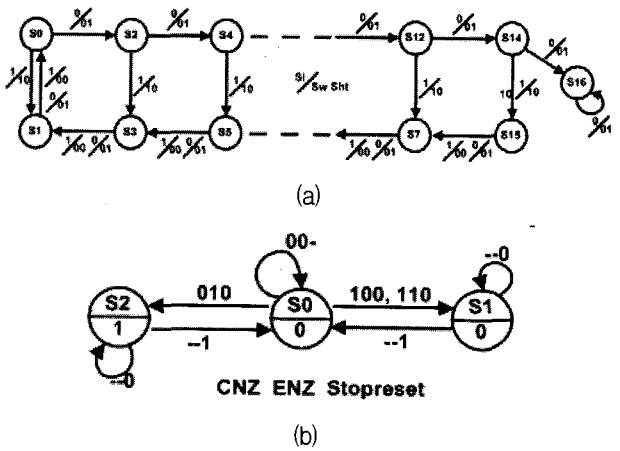


그림 6. (a) 제어 신호 생성 블록에 대한, (b) Stop 신호 생성 블록에 대한 Finite state machine (FSM).

Fig. 6. Finite state machine (FSM) for (a) control signal generate block and (b) stop-signal generate block.

연산의 종료 조건은 만족시킨다. 즉, PE의 C보다 E의 입력이 먼저 0이 아닌 값이 들어오는지 여부를 관찰하여 종료 조건을 만족하는지 아닌지를 *stop* 신호를 통해 알 수 있다. 그림 6(b)는 *stop* 신호에 대한 FSM이며 *stopreset* 신호에 의해 상태가 초기화된다. 초기의 상태는 S0 상태이고 ME 알고리즘의 연산 종료 조건이 만족되면 S2 상태가 된다. 종료 조건이 만족되면 그림 4의 *stop* 신호가 1이 되고 PE의 입력은 각각 4 클럭만큼의 지연 후 PE의 출력으로 나온다. 하지만 종료 조건이 PE16 이전에 만족된 경우 C와 E 의 입력은 2개의 PE 동안 8 클럭이 아닌 7 클럭만큼 지연되어야 PE16에서의 결과 값이 Chien Search 블록과 Forney 알고리즘 블록으로 전달되기 때문에, 그림 4에서 짝수 PE의 레지스터(2)는 제거된다.

V. 정정 불가능한 블록의 감지

RS(255,239) 복호기는 8개의 심벌 오류를 정정할 수 있다. 즉, 9개 이상의 오류가 발생한 경우 복호기는 코드워드의 오류를 정정할 수 없을 뿐만 아니라 정정 과정에서 오류가 추가되기도 한다. 그러므로 오류 정정이 불가능할 경우 이를 알려주는 신호가 필요하다. 블록의 오류 정정 가능 여부를 확인하기 위해 두 가지 방법을 이용할 수 있다. 하나는 신드롬 계산 블록을 복호기의 출력단에 추가하여 신드롬 값이 0인지 아닌지를 확인하는 것이다. 다른 하나는 KES블록의 출력인 오류 위치 다항식을 관찰하는 것이다. 오류가 9개 이상일 때와 8개 일 때의 KES 블록의 출력으로 나오는 오류 위치 다항식의 차수는 둘 다 8이므로 실제 오류가 8개가 발생했는지 아닌지를 확인하기 위해 실제 근을 대입한다. 오류가 8개인 경우 그 근이 8개가 나오고 그렇지 않은 경우 오류 위치 다항식의 근이 8개 미만이거나 초과하여 나오게 된다. 그러므로 오류 위치 다항식에 근을 대입함으로써 정정 가능한 근의 개수와 정정 가능 여부를 알 수 있게 된다. 전자의 방법은 신드롬 계산 블록을 추가하여 야 하므로 면적이 증가하는 단점이 있고 정정 가능한 블록일 때 정정한 오류의 수를 나타내는 블록을 추가하여야 한다. 후자의 방법은 오류 위치 다항식의 차수를 확인하고 오류 위치 다항식의 근의 개수를 카운트(count)하면 되므로 후자의 방법이 더욱 효율적이다.

VI. 40Gb/s RS FEC 구조

RS 복호기는 크게 신드롬 계산 블록, KES 블록, 오류 정정 블록으로 구성되어 있다. KES 블록은 16개의 신드롬 값을 순차적으로 받아서 처리하는 구조로 Systolic Array 형태로 구성되어 있다. 즉, 16개의 신드롬 계산 값이 순차적으로 PE1의 입력이 되고 이 값이 PE2, PE3로 거쳐 가는 구조이다. 이 구조의 장점은 출력이 다시 입력이 되는 궤한 구조가 아니기 때문에 중간에 플립플롭을 추가하여 쉽게 클럭 주파수를 높일 수 있다. 그러나 KES 블록을 구성하는 16개의 PE 중 동시에 들어오는 입력을 처리하는 PE는 4개 정도이고 나머지는 아무런 동작을 하지 않게 된다. 수신된 다항식 $R(x)$의 계수 값은 차례로 신드롬 계산 블록의 입력이 되고 RS(255,239)에서의 신드롬 계산 블록은 255 클럭 동안 255개의 계수 값을 연산하여 출력으로 16개의 신드롬 다항식의 값을 차례로 내보낸다. 즉, KES 블록의 PE들은 16 클럭의 연산을 위해 255 클럭을 기다리는 형태가 된다. 그러나 본 논문에서 제안한 파이프라인 형태의 KES 블록은 서로 다른 입력들을 처리하는 것이 가능하고 하나의 PE 블록에서 서로 다른 입력이 2 클럭이상의 간격을 가지면 데이터의 충돌 없이 연산을 수

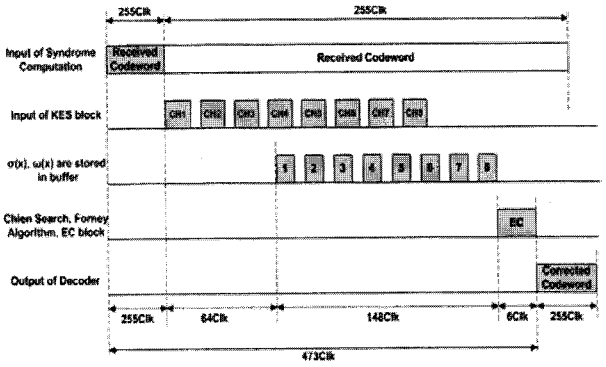


그림 8. 8채널 RS FEC구조의 타이밍도
Fig. 8. Processing timing chart of 8-Ch. RS FEC architecture.

행할 수 있다. 즉, 제안한 KES 블록은 18 클럭 사이클마다 새로운 신드롬 계산 블록의 입력을 처리할 수 있다. 그림 9는 각 채널의 신드롬 값이 KES의 입력으로 20클럭 사이클 간격으로 들어가는 것을 보여준다. 또한 하나의 KES 블록이 여러 개의 신드롬 계산 블록의 출력을 처리하기 때문에 PE가 연산을 수행하기 위해 입력을 기다리는 대기 시간을 줄일 수 있다. KES 블록은 18 클럭 사이클에 하나의 신드롬 계산 블록을 처리하므로 신드롬 계산 블록이 새 출력 값을 생성하기까지 최

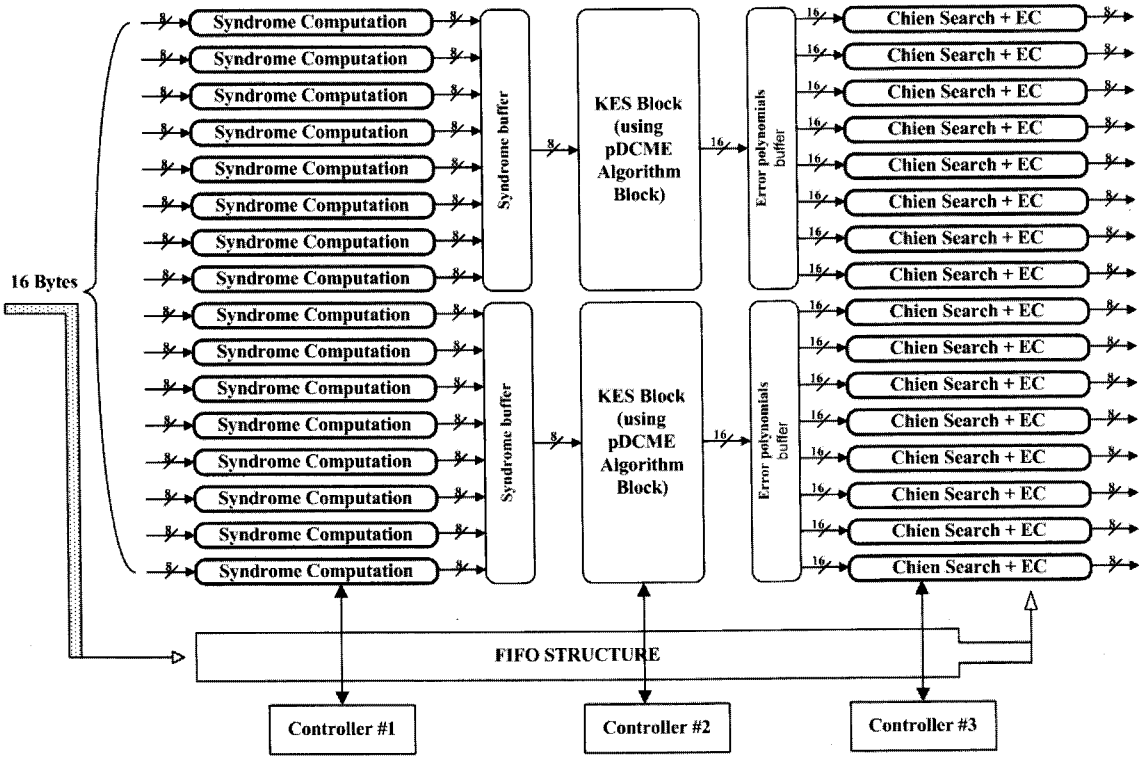


그림 7. 16 채널 RS 복호기 기반의 FEC 구조.
Fig. 7. 16-channel RS decoder based FEC architecture.

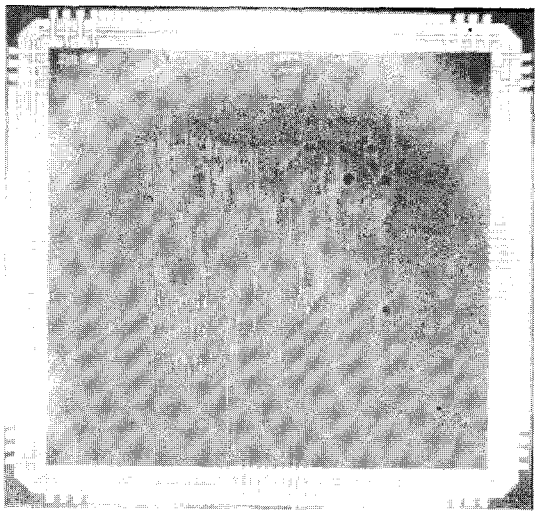


그림 9. 8채널 RS 기반 FEC칩의 다이 포토그래프
Fig. 9. Die photograph of the fabricated 8-Ch. RS decoder based FEC chip.

대 14개 신드롬 계산 블록의 출력을 처리할 수 있다. 그림 7은 한 개의 KES 블록이 8개의 신드롬 계산 블록의 출력을 처리하는 형태를 가진 8채널 RS 복호기 두 개를 병렬로 구성한 16 채널 RS 복호기 기반 FEC 구조를 보여준다. RS 복호기에서 KES 블록이 차지하는 면적이 약 80%이고 약 10% 지연성(latency)을 가지므로 그림 7과 같이 8개의 신드롬 계산 블록이 한 개의 KES 블록을 공유함으로써, 각각의 채널이 KES블록을 가질 때와 비교하여 총 지연성을 약 20% 늘어나는데 반해 약 73%정도 하드웨어 복잡도가 줄어든다.

Ⅶ. 결과 및 비교

본 논문에서 제안된 40Gb/s 급 16채널 RS 복호기 기반 FEC구조는 Verilog-HDL로 설계하였고 Mentor ModelSim 시뮬레이터로 검증하였다. Verilog-HDL로 설계한 RS 복호기의 결과는 C 언어로 설계한 모델과 정확히 일치하였다. 이런 검증 단계 후에는 SYNOPSYS Design Compiler(DC)를 이용하여 적절한 time 및 area constraint와 1.8*V*의 공급전압과 0.18-㎛ CMOS 기술을 이용하여 합성(synthesize)하고 구현하였다. 그림 9는 8채널 RS 복호기 기반의 FEC칩을 구현한 한 다이 포토그래프를 보여주며 칩 코어 사이즈는 1.1×1.1 mm^{2}이다. 표 1은 몇 종류의 RS 복호기의 게이트 수, 클록 속도, 지연성, 데이터처리율 등을 비교한 결과들을 보여주고 있다. RS 복호기에서 FIFO 메모리를 제외하고 하드웨어 복잡도를 비교한 결과 본 논문에서 제안한 RS 복호기는 이전의 다 채널 복호기보다 게이트 수가 줄어들었음을 알 수 있다. 본 논문에서 제안된 RS 복호기는 400*MHz*에서 동작하고 0.83 ㎲의 지연성, 5*Gb/s*의 처리속도를 갖는다. 그러므로 본 논문에서 제안한 RS 복호기가 지금까지 발표된 기존의 복호기보다 하드웨어복잡도와 데이터처리율을 고려한 효율성면에서 우수하다 것을 알 수 있다.

표 1. 16 채널 RS 복호기 기반 FEC의 구현 결과
Table 1. Implementation results of the 16-Ch. RS decoder based FEC architecture.

Design	Proposed	Three -Parallel [8]	Serial [5]
Syndrome	*48,800*	*40,000*	*60,000*
KES	*100,400*	*84,000*	*280,000*
Chien+EC	*100,700*	*240,000*	*132,000*
Total # of Gates	***249,900***	***364,000***	***472,000***
Clock Rate *(MHz)*	*400*	*112*	*625*
Latency (Clocks)	*473*	*168*	*355*
Throughput *(Gb/s)*	*51*	*43*	*80*
Technology	*0.18 ㎛ CMOS, 1.8V*	*0.16 ㎛ CMOS, 1.5V*	*0.13 ㎛ CMOS,1.2V*

Ⅷ. 결 론

본 논문은 40Gb/s 광통신시스템을 위한 16 채널 RS 복호기 기반의 FEC 구조를 제안하였다. Degree computation 블록을 제거한 pDCME 알고리즘 블록은 기존의 KES 블록에 비해 약 10%정도 면적을 줄였다. 8개의 채널이 한 개의 KES 블록을 공유하여 각각의 채널이 KES블록을 가지는 구조와 비교할 때 약 73%정도 하드웨어 복잡도를 줄였다. 제안된 RS 복호기 기반의 FEC는 ITU-T의 권고사항을 따르고 있으며 높은 면적 효율성과 높은 데이터 처리율을 가지고 있다. 그러므로 초고속 광통신장비를 위한 차세대 40Gb/s FEC등에 바로 적용될 수 있다.

참 고 문 헌

[1] "Forward Error Correction for Submarine Systems," Telecommunication Standardization Section, International Telecom. Union, ITU-T Recommendation G.975, Oct. 2000.

[2] S. B. Wicker, "Error Control Systems for Digital Communication and Storage," *Prentice Hall,*

1995.

[3] H. M. Shao, T. K. Truong, L. J. Deutsch, J. H. Yuen and I. S. Reed, "A VLSI Design of Pipeline Reed-Solomon Decoder," *IEEE Trans. on Computers*, Vol. C-34, No.5, pp.393-403, May 1985.

[4] W. Wilhelm, "A New Scalable VLSI Architecture for Reed-Solomon Decoders" *IEEE Jour. of Solid-state Circuits*, Vol34, No.3, Mar. 1999.

[5] H. Lee, "High-Speed VLSI Architecture for Parallel Reed-Solomon Decoder," *IEEE Trans. on VLSI Systems*, Vol. 11, No. 2, pp. 288-294, April 2003.

[6] H. Lee, "An Area-Efficient Euclidean Algorithm Block for Reed-Solomon Decoder," *IEEE computer society Annual Symposium on VLSI*, pp. 209-210, Feb. 2003.

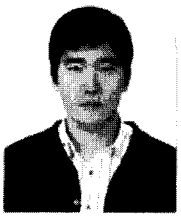
[7] D. V. Sarwate and N. R. Shanbhag, "High-Speed Architecture for Reed-Solomon Decoders," *IEEE Trans. on VLSI Systems*, Vol 9, No.5, pp.641-655, Oct. 2001.

[8] L. Song, M-L. Yu and M. S. Shaffer, "10 and 40-Gb/s Forward Error Correction Devices for Optical Communications," *IEEE Journal of Solid-State Circuits*, Vol. 37, No. 11, pp. 1565-1573, Nov. 2002.

[9] S. Lee, H. Lee, J. Shin, J.-S. Ko, "A High-Speed Pipelined Degree-Computationless Modified Euclidean Algorithm Architecture for Reed-Solomon Decoders," *2007 IEEE International Symposium on Circuits and Systems (ISCAS 2007)*, pp. 901-904, May 27. 2007.

[10] Bernard Sklar, 디지털 통신공학: 기본과 응용, 박상규 외 역, *교보문고*, pp. 516-544, 2003.

저 자 소 개



이 승 범(학생회원)
2006년 인하대학교 전자공학과 학사 졸업
2006년~현재 인하대학교 정보통신공학부 석사재학
<주관심분야 : 통신용 VLSI설계, SoC설계>

이 한 호(정회원)
1993년 충북대학교 전자공학과 학사 졸업.
1996년 Univ. of Minnesota 전기컴퓨터공학 석사
2000년 Univ. of Minnesota 전기컴퓨터공학 박사
2002년 Member of Technical Staff, Lucent Technologies, USA.
2004년 Assistant Prof. Dept. of Electrical and Computer Engineering, Univ of Connecticut, USA
2004년~현재 인하대학교 정보통신공학부 부교수
<주관심분야 : 통신용 VLSI설계, SoC설계>