論文93-30B-11-3

Berlekamp 알고리듬을 이용한 Reed-Solomon 복호기의 VLSI 구조에 관한 연구

(A Study on a VLSI Architecture for Reed-Solomon Decoder Based on the Berlekamp Algorithm)

金 勇 煥*, 鄭 榮 謨*, 李 商 郁*

(Yong Hwan Kim, Young Mo Chung and Sang Uk Lee)

要 約

본 논문에서는 Berlekamp 알고리듬에 바탕을 둔 Reed-Solomon(RS) 복호기의 VLSI구조를 제안 하였다. 제안된 복호기는 에러와 erasure의 정정 능력을 갖추고 있다. 본 논문에서는 RS 복호기의 VLSI 구현시 필요한 chip 면적을 줄이기 위하여 Berlekamp 알고리듬을 재구성 하였다. 제안한 알고리듬은 순환적인 구조 (recursive structure)로 이루어져서 errata 위치 다항식을 계산하는 cell의 수를 줄일 수 있게 되었다. 또한 제안한 방식에서는 1심볼 클럭 동안에 한 번의 곱셈만을 수행하기 때문에 복호화 속도를 향상시킬 수 있다. 그리고 복호기의 전체적인 구조는 실시간 구현에 적합하도록 병렬 구조와 pipeline 구조로 구성 하였다. 성능 평가 결과 본 논문에서 제안한 VLSI 구조는 Euclid 알고리듬에 바탕을 둔 순환적인 구조[6] 에 비해 VLSI 구현시 면적면에서 유리하다는 결론을 얻었다.

Abstract

In this paper, a VLSI architecture for Reed-Solomon (RS) decoder based on the Berlekamp algorithm is proposed. The proposed decoder provides both erasure and error correcting capability. In order to reduce the chip area, we reformulate the Berlekamp algorithm. The proposed algorithm possesses a recursive structure so that the number of cells for computing the errata locator polynomial can be reduced. Moreover, in our approach, only one finite field multiplication per clock cycle is required for implementation, provided an improvement in the decoding speed, and the overall architecture features parallel and pipelined structure, making a real time decoding possible. From the performance evaluation, it is concluded that the proposed VLSI architecture is more efficient in terms of VLSI implementation than the recursive architecture based on the Euclid algorithm. [6]

I. 서 론

최근 고품위 텔레비젼 (HDTV)이나 VTR (video tape recorder)등에서 영상 신호의 전송과 저장을 디지탈 방식으로 구현하는 연구가 활발히 수행되고 있다. 그런데 이러한 기기들의 디지탈 정보를 전송하거나 저장하는 과정에서 잡음 그리고 먼지 혹은 지문

*正會員, 서울大學校 制御計測工學科
(Dept. of control and Instrumentation Eng., Seoul Nat's Univ.)
接受日字 : 1992年 7月 8日

등에 의해 필연적으로 에러가 발생하게된다. 그러므로 전송하거나 저장하고자 하는 정보의 신뢰도를 높이기 위해서는 발생하는 에러의 검출및 정정을 위한 에러 정정 부호 (error correction code)의 도입이 필수적이다. 디지탈 통신 시스템에서의 에러는 주로 산발적으로 발생하는 랜덤 (random) 에러와 집중적으로 발생하는 버스트 (burst) 에러의 두 가지 종류가 있다. 이와 같은 에러를 검출하고 정정하기 위해서 많이 사용되고 있는 에러 정정 부호가 Reed-Solomon (RS) 부호이다.[1,3] RS 부호의 부호기는 쉬프트 레지스터를 이용해서 간단하게 하드웨어로 구현할 수 있다. 그러나 RS 복호기의 경우는 부호기와 같이 간단하게 구현되지 않고 방대한 하드웨어를 필요로 한다. RS 부호의 복호화 과정에서 에러 위치에 대한 정보를 포함하고 있는 에러 위치 다항식 (error locator polynomial)을 계산하는 과정이 가장 복잡하고 많은 시간을 필요로 한다. 에러 위치 다항식을 구하기 위해서는 복잡한 연립 방정식들을 풀어야 하기 때문에 이 식들의 해를 구하기 위한 반복적인 알고리듬 (iterative algorithm) 들이 많이 연구되었다.[1,2] 지금까지 알려진 알고리듬들 중에서 많이 사용되고 있는 알고리듬으로는 Berlekamp 알고리듬과 Euclid 알고리듬을 들 수 있다. 최근까지 위의 알고리듬들을 이용하여 RS 복호기를 VLSI로 구현하는 연구들이 많이 이루어져왔다.[4-9] 예를 들어 Liu[4]는 Berlekamp 알고리듬에 기초한 RS 복호기의 VLSI 구조를 제안하였다. 여기에서 Liu는 자신이 제안한 RS 복호기를 VLSI로 구현하기 위해서는 같은 복호기를 SSI나 MSI를 사용해서 구현했을 경우에 비해 약 1/10 정도의 chip 만이 필요하다는 것을 보였다. Shao[5] 역시 Euclid 알고리듬에 기초한 RS 복호기의 VLSI 구조를 제안하였다. 나중에 Shao는 기존의 Euclid 알고리듬을 개선하여 RS 복호기를 VLSI로 구현할 때 필요한 면적을 줄일 수 있는 새로운 구조를 제안하였다.[6] Shao[6]의 새로운 VLSI 구조에 적용한 기본 개념은 여러개의 cell을 동시에 사용하는 대신에 한 개의 cell을 순환적으로 사용한다는 것이다. 그렇지만 이 경우 역시 실질적으로 하드웨어에 적용하게 되면 여전히 많은 면적을 필요로 한다. 그러므로 RS 부호를 VLSI로 구현하고자 할 때 chip 면적을 줄일 수 있는 VLSI 구조에 관한 연구는 필수적이다. Euclid 알고리듬에서 각각의 반복 시에 변경되어져야할 계수들의 수는 주어진 RS 부호가 정정할 수 있는 에러의 갯수 t와 같다. 반면에 Berlekamp 알고리듬에서는 매 반복시 변경되어야할 계수의 수가 차례로 증가하여 알고리듬의 마지막 반복 과정에 이르렀을 때 비로소 t와 같아진다. 그러므로 Shao[6]가 적용한 것과 같은 순환적인 기법 (recursive technique) 을 Berlekamp 알고리듬에 적용한다면 하드웨어 구현시 Euclid 알고리듬에서 필요로 하는 클록 수보다 더 적은 클록으로 구현할 수 있을 것이다. 이것은 결국 하드웨어 구현시 면적의 복잡도가 감소한다는 것을 의미한다.

본 논문에서는 이와 같은 개념을 바탕으로 기존의 Berlekamp 알고리듬을 수정하여 RS 복호기 구성시 chip 면적을 줄일 수 있도록 하였고 수정된 알고리듬을 이용하여 RS 복호기의 새로운 VLSI 구조를 제안하였다. 제안한 VLSI 구조는 순환적 형태 즉, 각각의 계수들이 순차적으로 계산되어 지는 형태로 이루어져 에러 위치 다항식을 계산하기 위한 cell의 수를 감소시킬 수 있게 되었다. 또한 본 논문에서는 기존의 Berlekamp 알고리듬을 개선하여 1개의 심볼 클록 내에 단 한 번의 곱셈만이 수행되도록 함으로써 복호화 속도를 향상시킬 수 있음을 보였다.

Ⅱ. Berlekamp 알고리듬

본 절에서는 설명의 편의를 위하여 기존의 Berlekamp 알고리듬을 간략하게 기술한다. (N, I) RS 부호에서 채널을 통과하여 수신된 심볼들의 다항식 $R(x)$를 다음과 같이 정의한다.

$$R(x)=\sum_{k=0}^{N-1} r_k x^k = r_{N-1}x^{N-1} + \cdots + r_1 x + r_0 \qquad (1)$$

(N, I) RS 부호에서 최소거리 d는 $N-I+1$로 주어진다. 만약 $R(x)$에 $E(\leq (d-1)/2)$개의 에러가 존재한다고 가정하면 신드롬 (syndrome)들은 다음과 같이 정의된다.

$$S_j=\sum_{k=0}^{N-1} r_k \cdot \alpha^{(j+1)k}, \qquad 0 \leq j \leq N-I-1 \qquad (2)$$

그러면 신드롬 다항식 $S(x)$는 다음과 같이 주어진다.

$$S(x)=\sum_{i=0}^{N-I-1} S_i \cdot x^i \qquad (3)$$

한편 수신된 심볼들의 다항식 $R(x)$의 $i_1, i_2, \cdots, i_E$ 번째의 위치에 에러가 발생하였다고 가정하면 에러 위치 다항식 $\Psi(x)$는 다음과 같이 정의된다.

$$\Psi(x)=\prod_{k=1}^{E}\left(1-x\alpha^{i_k}\right)=\Psi_E x^E + \Psi_{E-1}x^{E-1} + \cdots + \Psi_0 \qquad (4)$$

미지의 $\Psi(x)$를 신드롬으로 부터 계산하는 문제는 $S(x)$

를 합성할 수 있는 최소 길이의 LFSR (linear feedback shift register)을 구하는 것과 동일한 과정임이 잘 알려져 있다.[2] LFSR에서 피드백 연결을 계수로 하는 다항식을 연결 다항식 (connection polynomial)이라 한다. n 번째 반복에서 L_n 차수를 가지는 연결 다항식을 $\Psi_n(x)$라 하면 n 번째의 discrepancy d_n은 다음과 같이 정의된다.

$$d_n = \sum_{k=0}^{L_n} \Psi_{n,k} \cdot S_{n-k} \tag{5}$$

여기에서 $\Psi_{n,k}$는 $\Psi_n(x)$의 k 차 항의 계수를 나타낸다. 한편 $\Psi_n(x)$를 $\Psi_{n+1}(x)$로 수정하기 위해 사용되는 정정 다항식 (correction polynomial)을 $D_n(x)$라고 정의하면 (n+1)번째 연결 다항식은 다음 식에서와 같이 계산된다.

$$\Psi_{n+1}(x) = \Psi_n(x) - d_n \cdot D_n(x) \tag{6}$$

그리고 $\Psi_{n+1}(x)$의 차수 L_{n+1}은 다음 식과 같다.

$$L_{n+1} = \begin{cases} L_n, & d_n = 0, \\ \max(L_n,\ n+1-L_n), & d_n \neq 0. \end{cases} \tag{7}$$

그러면 다음 반복에서의 정정 다항식은 다음 식에서 구할 수 있다.

$$D_{n+1}(x) = \begin{cases} xD_n(x), & L_{n+1} = L_n, \\ x\Psi_n(x)/d_n, & L_{n+1} > L_n. \end{cases} \tag{8}$$

이와 같은 알고리듬의 초기 조건들은 각각 n=0, L_0=0, $\Psi_0(x)$=1, 그리고 $D_0(x)$=x이다. 알고리듬의 각 과정들은 n = 0,1,2,…,N-I-1에 대해 반복해서 수행되며 마지막 반복이 수행된 후의 $\Psi_n(x)$가 구하고자 하는 에러 위치 다항식 $\Psi(x)$가 된다.

Ⅲ. 알고리듬의 분석

Liu가 제안한 VLSI 구조[4,10]에서의 문제점 중의 한 가지는 비효율적인 cell의 사용이다. Liu의 구조에서 cell의 사용 상태를 그림 1에 제시하였다. 그림 1에서 $C_1, C_2, \cdots, C_{d-1}$은 에러 위치 다항식의 계수들을 계산하는 ($N$-$I$)개의 cell들을 나타낸다. 그림 1(a)의 구조에서 (N-I)개의 계수들은 하나의 심볼 클록 동안에 동시에 계산되어진다. 그러나 Berlekamp 알고리듬에서는 (N-I)개의 계수들이 모두 동시에 계산되지 않는 특성이 있다. 즉, 식 (7)에서 고찰한 Berlekamp

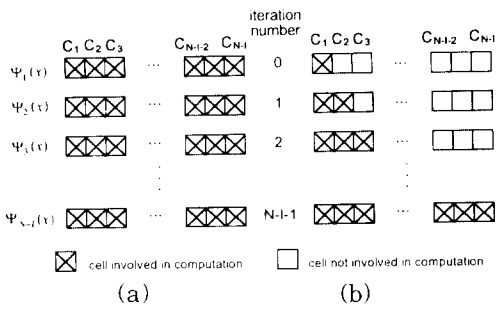


그림 1. 에러 위치 다항식 계수의 수정 과정
(a) LFSR 구조
(b) Berlekamp 알고리듬

Fig. 1. Updating procedure for the coefficients of the error locator polynomial.
(a) LFSR structure,
(b) Berlekamp algorithm.

알고리듬의 수행 과정 중에서 에러 위치 다항식의 차수는 n에관한 증가 함수이고 또한 0에서부터 차례로 증가해 간다. 실제 Berlekamp 알고리듬에서 계수가 변경되어 가는 과정을 그림 1(b)에 제시하였다. 그림 1(b)에 제시된 바와 같이 실제로 계수를 변경하는 cell들의 수는 반복이 계속됨에 따라 증가함을 알 수 있다. 따라서 본 절에서는 모든 계수들을 순차적으로 계산하는 방법을 제안한다.

에러 위치 다항식의 계수들을 순차적으로 계산하기 위해서는 각각의 반복시 계수들을 변경하기 위한 여러개의 sub-step들이 필요하다. 먼저 1 에서 $L_n(n)$ 까지의 값을 가지는 sub-step의 지표 i를 정의한다. 여기서 $L_n(n)$은 n번째 반복시에 수행되어야할 sub-step의 갯수이다.

그리고 n번째 정정 다항식 $D_n(x)$의 k차 항의 계수를 $D_{n,k}$로 나타낸다. 그러면 $D_{n+1}(x)$는 $\Psi_{n+1}(x)$의 차수 L_{n+1}에 따라 $xD_n(x)$나 $x\Psi_n(x)$로부터 얻을 수 있다. 그러므로 하나의 계수 $D_{n+1,i}$는 이보다 한 차수 낮은 $D_{n,i-1}$이나 $\Psi_{n,i-1}$로부터 얻어진다. 반면에 식 (6)에서 알 수 있듯이 $\Psi_{n+1}(x)$의 차수는 $\Psi_n(x)$나 $D_n(x)$의 차수와 동일하다. 그러므로 $\Psi_{n+1,i}$의 값은 $\Psi_{n,i}$ 와 $D_{n,i}$ 값으로부터 결정이 된다. 한편 n번째 LFSR이 (n+1)번째의 신드롬들을 정확히 합성한 경우 차수 L_{n+1}은 L_n과 동일하게 되며, 이 경우 $D_{n+1,i}$ 값은 $D_{n,i-1}$로부터 변함이 없다. 그러나 이 조건을 만족하지 않은 경우에 $D_{n+1,i}$는 $\Psi_{n,i-1}$과 d_n^{-1}으로 부터 수정된다. 그리고 마지막으로 (n+1) 번째의 discrepancy d_{n+1}은 이미 계산되어 있는 $\Psi_{n+1,i}$로부터 얻을 수 있다. 이 과정을

정리하면 다음과 같다.

FOR n = 1 *TO* ($N - I$) *DO*:
BEGIN
$d_t = 0$
FOR $i = L_r(n)$ *TO* 1 *DO*:
BEGIN
IF ($L_{n+1} = L_n$)
THEN $D_{n+1,i} = D_{n,i-1}$.
ELSE $D_{n+1,i} = \Psi_{n,i-1}\ (d_n)^{-1}$.
$\Psi_{n+1,i} = \Psi_{n,i} + d_n D_{n,i}$.
$d_t = d_t + \Psi_{n+1,i} S_{n+1-i}$.
END
$d_{n+1} = d_t$
END

위의 과정을 기존의 Berlekamp 알고리듬과 비교했을 때의 차이점은 n번째 반복을 수행할 때 기존의 알고리듬이 n번째 discrepancy를 계산하는 반면에 제안한 순환적인 과정에서는 sub-step 내에서 계산이 완료되어 있는 $\Psi_{n+1,i}$의 값들을 이용해서 (n+1)번째 discrepancy를 계산한다는 것이다.

이제 알고리듬의 반복 과정에서 계수들의 변경을 순차적으로 수행하기 위하여 필요한 sub-step의 갯수를 결정하는 방법에 대하여 제안한다. 그림 1(b)에 제시된 바와 같이 반복이 계속됨에따라 각 반복에서 수행되는 sub-step의 갯수도 증가한다. $L_r(n+1)$을 (n+1)번째 반복에서 수행되어져야할 sub-step의 갯수라 하면 $L_r(n+1)$은 $\Psi_{n+1}(x)$와 $D_{n+1}(x)$ 가운데서 높은 차수에 따라 결정되는데, 이 값은 다시 L_n, L_{n+1} 그리고 $L_r(n)$에 의해서 구하여진다. 먼저 L_{n+1}이 L_n과 같을 경우 $D_{n+1}(x)$의 차수는 $D_n(x)$의 차수보다 한 차수 커지게 된다. 그러므로 이 경우 $L_r(n+1)$은 $L_r(n)+1$로 증가 된다. 다음 L_{n+1} 값이 변하는 경우는 $\Psi_{n+1}(x)$의 차수에 변화가 있다는 것을 의미하는데, 이 경우 $L_r(n+1)$의 값은 $L_{n+1}+3$ 이 된다. 이 과정 중 3이 더해지는 이유는 다음과 같다. 먼저 L_{n+1}은 $\Psi_{n+1,1}$부터 $\Psi_{n+1,L_{n+1}}$ 까지 계수들의 갯수이므로 $\Psi_{n+1,0}$를 처리하기 위해 필요한 1개의 sub-step이 있어야 한다. 그리고 $D_{n+1}(x)$의 차수가 $\Psi_{n+1}(x)$의 차수보다 한 차수 더 높으므로 이 과정에서 하나의 sub-step이 더 필요하다. 마지막으로 d_t를 계산하기 위한 $\Psi_{n,L_r(n)-1}$의 값을 저장하는 레지스터가 필요하므로 이 레지스터의 내용을 쉬프트시키기 위한 하나의 sub-step이 더 있어야 한다. 여기에 사용되는 레지스터에 관해서는 다음 4.1절에서 자세히 논한다. 한편 L_{n+2}는 n번째 반복에서 계산되어진다. d_{n+1}의 값이 n번째 반복에서 이미 계산되어 있으므로 L_{n+2}의 값은 (7)에 보인 바와 같이 L_{n+1}과 d_{n+1}로부터 얻을 수 있다. 이 과정을 정리하면 다음과 같다.

IF ($L_{n+1} = L_n$)
THEN $L_r(n+1) = L_r(n) + 1$.
ELSE $L_r(n+1) = L_{n+1} + 3$.
IF ($d_{n+1} = 0$)
THEN $L_{n+2} = L_{n+1}$.
ELSE $L_{n+2} = max(\ L_{n+1},\ n + 2 - L_{n+1})$.

이와 같이 알고리듬을 변경할 경우 Liu [4] 의 구조에서와 같이 동시에 모든 계수들을 계산하는 알고리듬에 비해 복호화하는데 소요되는 클록의 수가 더 많이 필요하게 되는 문제점이 발생한다. 즉, 한 부호어를 복호화할 때 소요되는 클럭의 수가 한 부호어의 심볼 클럭 수를 초과하게 된다. 그렇지만 이 문제점은 여러개의 cell을 multiplexer와 demultiplexer로 연결함으로써 해결할 수 있다. 이 것에 관해서는 다음 4.2절에서 자세히 설명한다.

Ⅳ. RS 복호기의 VLSI 구조

본 절에서는 먼저 에러 위치 다항식을 계산하기 위한 새로운 VLSI 구조를 제안한다. 두번째로 심볼 클록 수의 증가 문제를 해결하기 위해 여러개의 cell을 사용하는 구조에 대해서 논한다. 그리고 이와 같은 VLSI 구조들을 이용한 RS 복호기의 전체적인 VLSI 구조를 제시한다. 마지막으로 본 논문에서 제안한 구조의 성능을 평가하기 위하여 곱셈기와 레지스터 구성시 필요한 면적을 Shao의 구조 [6] 에서 필요로 하는 면적과 비교한다.

1. 에러 위치 다항식을 계산하기 위한 VLSI 구조

그림 2는 에러 (혹은 errata) 위치 다항식을 계산하는 VLSI 구조이다. 그림 2의 VLSI 구조는 기본적으로 계수들을 변경하는 cell과 레지스터 블록(register block) 으로 구성되어 있다. 그림 2에서 cell의 입력과 출력은 각각의 레지스터 블록에 연결되어 계수들의 개별적인 저장및 호출이 가능하다. 레지스터 블록의 내부 구조는 그림 3에 제시 하였다. 그림 3에서 Ψ_k 와 D_k는 각각 연결 다항식의 계수값들을 저장하는 레지스터블록과 정정 다항식의 계수값들을 저장하는 레지스터 블록의 (k+1)번째 레지스터를 나타낸다. 이제 에러 위치 다항식을 계산하는 과정을

각 단계에 따라 상세히 논한다.

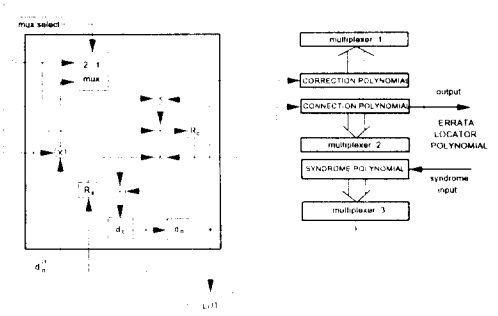


그림 2. Berlekamp 알고리듬을 위한 제안한 VLSI구조

Fig. 2. Proposed VLSI architecture for the Berlekamp algorithm.

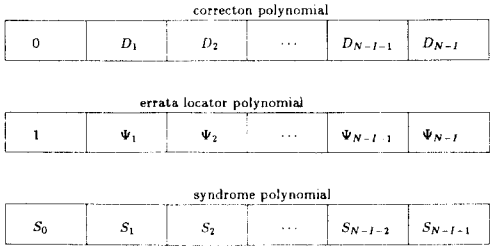


그림 3. 각 레지스터 블록들의 구성

Fig. 3. Structure for the register blocks.

초기화

RS 복호기가 erasure의 정정 능력을 가지려면 알고리듬 수행시 연결 다항식과 정정 다항식의 초기 조건으로 erasure 위치 다항식이 인가되어야한다. Erasure 위치 다항식 $\Gamma(x)$는 다음과 같이 정의된다.

$$\Gamma(x)=\prod(x-\alpha^{-i}), \qquad \forall\alpha^{-i}\in\Gamma. \tag{9}$$

여기에서 $\Gamma=\{\alpha^{-i_i} \mid i_i,\ i=1,2,\cdots,N$은 erasure} 이다. 입력되는 부호어에서 erasure의 위치를 알 수 있기 때문에 부호어의 입력이 끝난 직후 erasure 위치 다항식을 얻을 수 있다.

신드롬 포인터 (syndrome pointer)

신드롬 계산부로 부터 얻어진 신드롬들은 먼저 신드롬 다항식 레지스터에 저장된다. 그 후 신드롬들은 그림 2의 multiplexer 3을 통하여 임시 저장 레지스터인 R_s에 저장되어 d_n의 계산에 사용된다. 그런데 각 반복에서 사용되어지는 신드롬들의 수는 각 반복에 따라 다르다. 그러므로 포인터 변수인 $Sind$를 사용하여 각 반복마다 사용되어지는 신드롬의 위치를 표시한다. 그림 2에서 multiplexer 3은 신드롬 포인터 $Sind$에 따라 적절한 신드롬을 선택해서 R_s에 저장한다. 한편 d_{n+1}을 계산하기 위하여 $\Psi_{n+1,i}$값을 저장하는 레지스터가 필요하다. 그림 2에서 multiplexer 2에 의해 선택되어진 $\Psi_n(x)$의 계수들은 Berlekamp 알고리듬에 의해 적절하게 변경된 후 높은 차수의 계수부터 낮은 차수의 계수까지 차례로 R_c에 입력된다. 그러므로 임시 레지스터 R_s에는 낮은 차수의 신드롬으로부터 높은 차수의 신드롬까지 차례로 저장되어야 한다. 신드롬 포인터의 자세한 변경과정은 그림 4의 흐름도에 제시하였다.

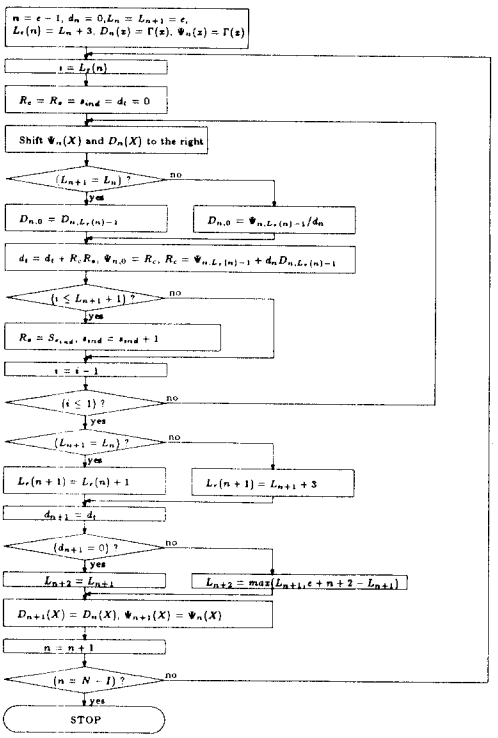


그림 4. 에러 위치 다항식을 계산하는 흐름도

Fig. 4. Flowchart for computing the error locator polynomial.

정정 다항식과 연결 다항식의 순차적 변경

R_c, R_s 그리고 Sind의 값들이 적절히 초기화되면 정정 다항식과 연결 다항식의 계수들의 순차적인 변경과정이 수행된다. 먼저 그림 2의 각 레지스터 블록에

서는 각 다항식의 계수들이 낮은 차수로부터 높은 차수로 한 단계씩 쉬프트되고 cell의 출력들은 각 다항식의 가장 낮은 차수의 계수로 입력된다. 동시에 다항식 레지스터 블록으로부터 쉬프트된 $D_{n,Lr(n)-1}$이 계수 변경 과정에 이용되기 위하여 cell에 입력된다. 만약 L_{n+1}이 L_n과 같을 경우 $D_{n,Lr(n)-1}$은 변경없이 다음 반복에 사용된다. 그렇지 않을 경우 $D_{n,Lr(n)-1}$은 신드롬들을 올바르게 합성할 수 있도록 변경되어야만 한다. 변경된 $D_{n,Lr(n)-1}$은 다시 가장 낮은 차수의 레지스터인 $D_{n,0}$에 입력된다. 그리고 d_t의 출력은 accumulator에 계속 입력되어 discrepancy를 형성하게 되고 R_c에 저장되어 있던 값은 $\Psi_{n,0}$로 쉬프트된다. R_c에는 새로 변경된 연결 다항식의 계수가 입력된다. 이상에서 고찰한 3가지 과정들은 동시에 수행되어진다.

각각의 계수들의 변경 과정이 완료되면 accumulator에 저장되어 있는 d_t가 d_{n+1}이 되고 연결 다항식의 계수들을 변경하는 데 사용되는 $(d_{n+1})^{-1}$의 값은 look-up-table (LUT)을 통해서 얻을 수 있다. 또한 $\Psi_{n+1}(x)$ 와 $D_{n+1}(x)$의 계수들은 해당되는 레지스터들인 $\Psi_0, \Psi_1, \cdots, \Psi_{Lr(n)}$ 과 $D_0, D_1, \cdots, D_{Lr(n)}$에 각각 저장된다.

2. 여러개의 cell을 사용하는 구조

에러 위치 다항식의 계수들을 계산하는 데 소요되는 클럭수는 erasure의 수 e와 관계가 있고 이의 최대치는 $e = 0$ 일 때 발생한다. 초기화 과정들에 소요되는 클럭수를 포함한 소요 시간의 최대치를 t_m이라 하면 t_m은 다음 식과 같이 주어진다.

$$t_m = \sum_{n=-1}^{N-I-1} L_r(n) \quad symbol\ clocks \tag{10}$$

그러나 위의 식에서 $L_r(n)$은 입력되는 부호어에 따라 변화되는 값이기 때문에 $L_r(n)$을 정확히 계산하기는 어렵다. 그러나 t_m의 상한값은 다음과 같이 계산할 수 있다.

$$t_m = 3+4+5+\cdots+(N-I+3)\ symbol\ clocks. \tag{11.a}$$

그리고 이 식은 다음과 같이 간략화 된다.

$$t_m = \frac{(N-I+1)(N-I+6)}{2} \quad symbol\ clocks \tag{11.b}$$

위의 식에서 알 수 있듯이 t_m의 값은 N보다 크다. 이 것은 위의 알고리듬을 실시간으로 구현하기 위해서는 여러개의 cell을 사용하여야함을 의미한다. 에러 위치 다항식의 모든 계수들이 N 심볼 클럭내에 계산되어져야 함으로 실시간 구현을 위해서는 $\lfloor t_m/N \rfloor$개의 cell이 필요함을 알 수 있다. 여기에서 $\lfloor x \rfloor$는 x 보다 큰 최소의 정수를 의미한다. 이와 같이 여러개의 cell을 사용하는 구조를 그림 5에 제시하였다. 그림 5에서 신드롬 다항식과 erasure 위치 다항식의 계수들은 multiplexer를 통하여 서로 다른 cell들에 연결된다. 그리고 매 t_m마다 계산되어지는 에러 위치 다항식의 계수들은 demultiplexer를 통하여 N 심볼 클럭마다 출력된다. 이와같이 multiplexing 기법과 demultiplexing 기법을 사용함으로써 pipeline 구조의 구현이 가능하다.

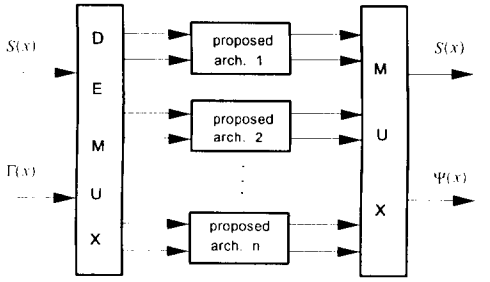


그림 5. 여러 cell 구조의 블록선도
Fig. 5. Blockdiagram for multiple cell structure.

3. 전체적인 VLSI RS 복호기

RS 복호기의 전체적인 블럭 다이어그램을 그림 6에 제시하였다. 이 구조는 일반적으로 알려진 전체적인 RS 복호기의 구조[4]와 비교하여 erasure 위치 다항식을 발생시키는 블록과 errata 평가 다항식을 계산하는 블록이 포함된 것이 다르다. 본 절에서는 이 두 블록에 대한 동작 과정을 제시한다.

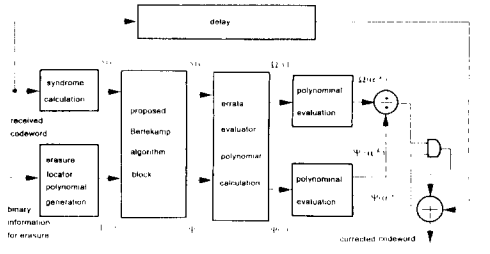


그림 6. RS 복호기의 전체 블록선도
Fig. 6. Overall blockdiagram for RS decoder.

Erasure 위치 다항식을 생성하는 VLSI 구조

Erasure 위치 다항식은 (9)식과 같이 계산된다. 따라서 e개의 erasure가 $j_1, j_2, \cdots, j_e$ 위치에 발생되었다고 가정하면 erasure 위치 다항식 $\Gamma(x)$는 다음과 같이 주어진다.

$$\Gamma(x)=\prod_{k=1}^{e}(1-x\alpha^{j_k}). \tag{12}$$

위의 식에서 $\Gamma(x)$는 $(1-x\alpha^{j_k})$의 연속적인 곱셈으로 계산된다. 따라서 e = 2 인 경우, 다음의 식을 얻을 수 있다.

$$(1-x\alpha^{j_1})(1-x\alpha^{j_2})=(1-x\alpha^{j_1})-x\alpha^{j_2}(1-x\alpha^{j_1}) \tag{13}$$

식 (13)을 수행하려면 두 단계가 필요하다. 먼저 $(1-x\alpha^{j_1})$ 과 α^{j_2} 의 곱셈을 수행하고 한 차수 높은 쪽으로 쉬프트시킨다. 그리고 위의 식과 같이 두 항의 덧셈을 수행한다. e = 3일 경우에는 위의 과정을 한 번 더 반복하면 된다. 이러한 과정을 수행하는 VLSI 구조를 그림 7에 제시 하였다. 그림 7에서 상수항의 계수를 나타내는 맨 왼쪽 cell에서 입력 '1'은 최초 클록에서만 인가되고 그후에는 '0'이 입력된다.

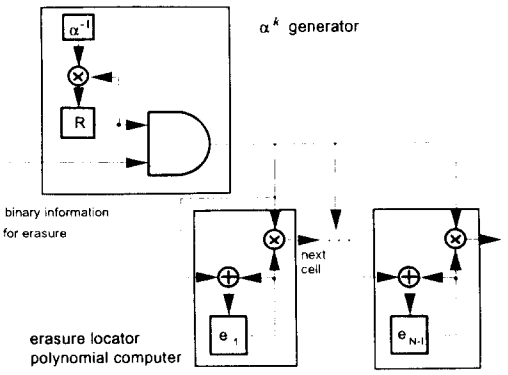


그림 7. Erasure 위치 다항식을 생성하는 구조
Fig. 7. Architecture for generating erasure locator polynomial.

Errata 평가 다항식을 생성하는 VLSI 구조

Erasure가 있을 경우 errata 평가 다항식 $\Omega(x)$는 다음 식에서 계산되어 진다. [2]

$$S(x)\times\Psi(x)\equiv\Omega(x)\bmod x^{N-I}. \tag{14}$$

이제 이 식을 수행할 수 있는 VLSI 구조를 제안한다. 만약 수신된 부호어에 E개의 에러와 e개의 erasure가 존재한다고 가정하면 Ψ_i, $i = 2E + e + 1, 2E + e + 2, \cdots, N - I$ 는 모두 0의 값을 갖는다. 그러면 위 식에서 errata 위치 다항식 $\Psi(x)$는 다음과 같이 주어진다.

$$\Psi(x)=\Psi_{N-I}x^{N-I}+\Psi_{N-I-1}x^{N-I-1}+\cdots+\Psi_0 \tag{15}$$

그러면 errata 평가 다항식 $\Omega(x)$는 (3) 식과 (15) 식을 이용하여 다음과 같이 나타낼 수 있다.

$$\Omega(x)=S_0\Psi_0+(S_0\Psi_1+S_1\Psi_0)x+\cdots+\sum_{i=0}^{N-I-1}S_i\Psi_{N-I-i}x^{N-I-1}. \tag{16}$$

그림 8. Errata 평가 다항식을 계산하는 과정
Fig. 8. Procedure for computing errata evaluator polynomial.

Errata 평가 다항식을 생성하는 자세한 과정을 그림 8에 제시하였다. 그리고 그림 9에는 errata 평가 다항식을 계산하는 구조를 제시하였다. 그림 9에서 (N-I)개의 cell들은 초기값으로 각각 S_0로부터 S_{N-I-1}의 값들을 저장하고있다. 초기에 Ψ_0가 입력되면 Ψ_0와 각 신드롬들과의 곱셈이 수행되고 그 결과들은 각 cell내에 있는 Ω_i, $i = 1, 2, \cdots, N-I-1$ 들에 저장된다. 이 과정의 수행이 끝나면 신드롬 레지스터 S_i에 저장되어 있던 신드롬 값들이 S_{i+1}로 쉬프트되고 S_0 레지스터에는 0이 입력된다. 새로 쉬프트된 신드롬 값들은 Ψ_1과 곱해져서 다시 각 cell내에 있는 Ω_i들에 더해진다. 이 과정은 각 Ω_i들에 errata 평가 다항식의 계수들이 저장될 때까지 수행된다.

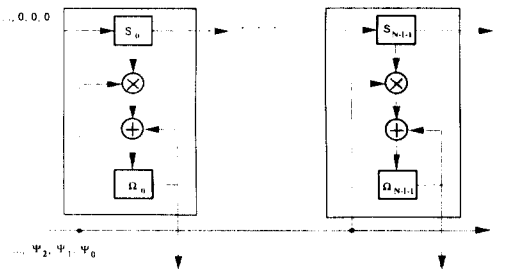


그림 9. Errata 평가 다항식을 위한 VLSI구조
Fig. 9. VLSI architecture for errata evaluator polynomial.

4. Shao의 VLSI 구조와의 비교

본 절에서는 제안한 RS 복호기의 VLSI 구조와 Shao가 제안한 VLSI 구조[6]의 성능을 비교한다. 일반적으로 신호처리 알고리듬을 VLSI로 구현할 경우 면적은 수율 (yield ratio), 설계 시간 등에 영향을 미치는 가장 중요한 요소이다. 또한 구조의 성능을 평가할 때 modularity와 자료 접근 방법등도 중요한 고려의 대상이다. 그런데 후자의 항목들을 비교의 대상으로 삼는 것은 각각의 구조에서 사용되어진 기술과 가정의 차이 때문에 일반적으로 쉽지 않다. 따라서 본 절에서는 성능의 평가를 소요되는 면적으로써 행한다. 그러나 전체 chip 면적중 배선이 차지하는 면적은 직접적으로 비교하기가 매우 어려우므로 본 절에서는 VLSI 구현시 필요한 곱셈기와 레지스터가 차지하는 면적만을 비교한다.

면적의 비교를 위하여 (255,223) RS 부호를 복호화 하는 경우를 예로 든다. 동등한 조건에서의 비교를 위하여 [6] 에서 정의된 것과 동일하게 위하여 상수×변수를 수행하는 cell의 면적을 단위 면적이라 하고 A_u로 나타낸다. 단위 면적을 이와 같이 정의할 경우 변수×변수 연산을 수행하는 곱셈기로 구성되어진 cell의 면적은 대략 $2A_u$[6]가 된다. 표 1에 본 논문에서 제안한 VLSI 구조와 Shao가 제안한 VLSI 구조[6]에서 각각 필요로하는 면적에 대한 비교를 제시하였다. 먼저 errata 위치 다항식을 계산하는 블럭에 필요한 면적에 대해 고찰하면 Shao의 구조[6]에서 변경된 Euclid 알고리듬을 처리하는 cell은 변수 변수를 수행하는 4개의 곱셈기와 4×32 피드백 지연 레지스터로 구성되는 데 각각 $8A_u$와 $32A_u$의 면적을 필요로한다. (N,I) RS 복호기에서 완전한 throughput rate 를 유지하려면 $\lfloor (N-I)^2/N \rfloor$개의 cell이 필요하다. 따라서 (255,223) RS 복호기의 경우 총 5개의 cell이 필요하다. 그러므로 변경된 Euclid 알고리듬을 사용하는 구조에는 총 $200A_u$의 면적이 필요하다. 본 논문에서 제안한 구조에서는 변경된 Berlekamp 알고리듬을 처리하는 cell은 변수×변수를 수행하는 3개의 곱셈기와 3×32 피드백 지연 레지스터로 구성되는 데 각각 $6A_u$와 $24A_u$의 면적이 필요하다. 그리고 변경된 Berlekamp 알고리듬을 수행하기 위하여 이와같은 cell이 총 $\lfloor t_m/N \rfloor$개가 소요된다. 그러므로 (255,223) RS 복호기의 경우 총 3개의 cell만으로도 완전한 throughput rate 를 유지할 수 있다. 결국 Berlekamp 알고리듬의 수행을 위해 필요한 면적은 $90A_u$가된다.

한편 두 구조에서 공통적으로 사용되지 않는 블록들을 비교하면 제안하는 구조에서는 errata 평가 다항식이 필요한 반면, Shao[6]의 구조에서는 변경된 신드롬들을 계산하기 위한 블록이 필요함을 알 수 있다. 그러나 이 두 구조에서 필요한 면적은 표 I로 부터 동일함을 알 수 있다.

표 1. Chip 면적의 비교(단위: A_u)
Table 1. Chip area comparison(Unit:A_u).

blocks	*Shao's*	*proposed*
syndrome computation	32	32
polynomial expansion for erasure locator polynomial	64	64
polynomial expansion for modified syndrome	64	
α^k generation	8	1
proposed architecture for Berlekamp algorithm		90
errata evaluator polynomial		64
modified Euclidean algorithm	200	
polynomial evaluation for errata locator polynomial	32	32
polynomial evaluation for errata evaluator polynomial	32	32
total area	432	315

본 논문에서는 단지 곱셈기와 레지스터가 차지하는 면적만을 비교했지만 본 논문에서 제안한 VLSI 구조에서 에러 위치 다항식을 구하기 위한 cell의 면적이 Shao의 구조[6]에서 cell의 면적보다 작고 전체적으로 적은 수의 cell들이 필요하기 때문에 본 논문에서 제안한 VLSI 구조가 Shao의 것에 비해 면적의 복잡도 측면에서 유리하다고 판단된다.

Ⅴ. 결 론

본 논문에서는 Berlekamp 알고리듬에 바탕을 둔 RS 복호기의 새로운 VLSI 구조를 제안하였다. 본 논문에서 제안한 VLSI 구조는 에러및 erasure의 정정 능력을 갖추고 있다. 본 논문에서는 VLSI 구현시 필요한 chip의 면적을 줄이기 위해 기존의 Berlekamp 알고리듬을 개선하였다. 제안된 알고리듬은 에러 위치 다항식을 계산하는 cell의 수를 줄일 수 있도록 순환적인 구조로 이루어져 있다. 또한 본 논문에서 제안한 VLSI 구조는 한 심볼 클럭내에 한 번의 곱셈만을 수행 하도록 이루어져 있다. 그리고 각각의 복호화 module들의 구조는 VLSI 구현에 맞도록 간단한 처리기들로 이루어져 있어서 실시간 구현이 가능하게 되었다. 제안한 VLSI 구조의 면적에 대한 성능 분석을 위해 기존의 VLSI 구조중에서 순환적인 기법을 사용한 Shao[6]의 VLSI 구조와 비교하였다. 그 결과 본 논문에서 제안한 알고리듬을 VLSI 구조로 구현할 경우 Shao의 알고리듬을 사용했을 때에 비해 VLSI 구현시 필요한 면적을 감소시킬 수 있었다. 결론적으로 본 논문에서 제안한 RS 복호기의 VLSI 구

조는 많은 데이타를 처리하는 디지탈 시스템이나 빠른 복호화 속도가 요구되는 시스템에 사용될 수 있을 것으로 판단된다.

본 논문에서 제안한 VLSI 구조의 성능 분석은 Shao[6] 구조와의 동등한 환경에서의 비교를 위하여 곱셈기와 레지스터가 차지하는 면적들이 고려되었고 각각의 레지스터와 곱셈기 사이의 연결선(interconnection) 이 차지하는 면적은 고려되지 않았다. 그리고 이러한 연결선이 차지하는 면적은 실제로 layout 을 행하지 않고 추정하는 것이 매우 어렵다. 따라서 제안한 구조의 보다 정확한 성능 분석을 위하여 제안한 구조의 실제 layout이 앞으로의 연구 과제로서 필요하다.

參 考 文 獻

[1] S. Lin and D. J. Costello, Jr., *Error Control Coding*. Prentice-Hall : Englewood Cliffs, 1983.

[2] G. C. Clark, Jr. and J. B. Cain, *Error-Correction Coding for Digital Communications*. Plenum Press : New York, 1981.

[3] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*. 2nd Ed., MIT Press : Cambridge, 1972.

[4] K. Y. Liu, "Architecture for VLSI design of Reed-Solomon Decoders," *IEEE Trans. Comput.*, vol. C-33, pp. 178-189, Feb. 1984.

[5] H. M. Shao and *et. al.*, "A VLSI design of a pipeline Reed-Solomon Decoder," *IEEE Trans. Comput.*, vol. C-34, pp. 393-403, May 1985.

[6] H. M. Shao and I. S. Reed, "On the VLSI design of a pipeline Reed-Solomon Decoder using systolic arrays," *IEEE Trans. Comput.*, vol. C-37, pp. 1273-1280, Oct. 1988.

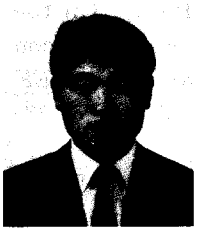
[7] Y. R. Shayan, T. Le-Ngoc, and V. K. Bhargaba, "A versatile time-domain Reed-Solomon Decoder," *IEEE J. Select. Areas in Commun.*, vol. 8, pp. 1535-1542, Oct. 1990.

[8] B. Arambepola and S. Choomchuay, "VLSI array architecture for Reed-Solomon Decoding," in *Proc. Int. Symp. on Circuits and syst.*, pp. 2963-2966, June 1991.

[9] S. R. Whitaker, J. A. Canaris, and K. B. Cameron, "Reed-Solomon VLSI codec for advanced television," *IEEE Trans. Circuit Syst. for Video Technol.*, vol. 1, pp. 230-236, June 1991.

[10] J. L. Massey, "Shift-Register synthesis and BCH decoding," *IEEE Trans. Inf. Theory.*, vol. IT-15, pp. 122-127, Jan. 1969.

著 者 紹 介



金 勇 煥(正會員)

1966年 3月 12日生. 1989年 2月 서울대학교 제어계측공학과 졸업 (학사). 1991年 2月 서울대학교 제어계측공학과 졸업 (석사). 1991年 8月 ~ 1992年 2月 군복무(특수전문요원). 1992年 8月 ~ 현재 미국 University of Texas, Austin 박사과정 재학중. 주관심 분야는 디지탈 신호처리, VLSI 구현, BISDN 및 ATM 등임.

李 商 郁(正會員) 第 22券 第 1號 參照

현재 서울대학교 제어계측공학과 교수

鄭 榮 謨(正會員)

1964年 2月 15日生. 1986年 2月 서울대학교 제어계측공학과 졸업 (학사). 1988年 2月 서울대학교 대학원 제어계측공학과 졸업 (석사). 1993年 2月 서울대학교 대학원 제어계측공학과 졸업 (박사). 1988年 8月 ~ 1989年 2月 군복무 (특수전문요원). 1992年 4月 ~ 1993年 10月 서울대학교 제어계측공학과 유급조교. 1993年 10月 ~ 현재 서울대학교 자동화시스템공동연구소 연구원. 1993年 3月 ~ 현재 숭실대학교 정보통신공학과 강사. 주관심분야는 이동통신, 디지탈 신호처리 및 VLSI 구현 등임.