논문 2010-47SD-11-8

연판정 Reed-Solomon 리스트 디코딩의 Factorization을 위한 효율적인 VLSI 구조

(Efficient VLSI Architecture for Factorization in Soft-Decision Reed-Solomon List Decoding)

이 성 만*, 박 태 근**

(Sungman Lee and Taeguen Park)

요 약

Reed-Solomon(RS) 코드는 강력한 에러 정정 능력으로 널리 사용된다. 최근 Sudan에 의해 Reed-Solomon 코드의 리스트 디코딩 알고리즘이 정립되었다. 리스트 디코더는 일반적인 디코더보다 더 큰 디코딩 반경을 가지며 하나 이상의 코드를 찾아낸다. 리스트 디코더는 복잡도와 latency가 매우 큰 Interpolation 과 Factorization 단계를 포함하므로 효율적인 하드웨어 설계가 필요하다. Factorization 은 latency가 매 단계마다 변하는 특성을 가져 복잡도가 높으며, 하드웨어 효율 저하의 문제가 발생한다. 본 논문에서는 하드웨어의 재사용을 높인 구조와 알고리즘의 효율적인 처리 스케쥴을 제안한다. 제안한 구조는 각 단계를 작은 단위의 R-MAC 유닛으로 나누어 매 단계마다 하드웨어를 재구성하여 처리함으로서 높은 하드웨어 효율과 효율적인 메모리 구조를 통해 복잡도가 낮은 순차처리를 적용하면서도 높은 처리량을 보이며, 여러 가지 어플리케이션에 적용가능하다. 제안한 구조는 동부 아남 0.18μm 표준 셀 라이브러리를 사용하여 합성한 결과 최대 동작 주파수는 330MHz이다.

Abstract

Reed-Solomon (RS) codes are the most widely used error correcting codes in digital communications and data storage. Recently, Sudan found algorithm of list decoder for RS codes. List decoder has larger decoding radius than conventional hard-decision decoding algorithms and return more than one candidate polynomial. But, the algorithm includes interpolation and factorization step that demand massive computations. In this paper, an efficient architecture and processing schedule are proposed. The architecture consists of R-MAC, memories, and control unit. The R-MAC computes both of RC and PU steps that are main part of the factorization algorithm. The proposed architecture can achieve higher hardware utilization efficiency (HUE) and throughput by using efficient processing schedule and memory architecture. Also, the architecture can be designed flexibly with scalability for various applications. We design and synthesize our architecture using Dongbu-Anam 0.18μm standard cell library and the maximum clock frequency is 330MHz.

Keywords: Reed-Solomon codes, soft-decision list decoder, factorization, VLSI architecture

Ⅰ. 서 론

Reed-Solomon(RS) 코드는 에러정정코드로써, 데이터 저장, 유, 무선 통신, 위성 통신 등 다양한 어플리케이션에 사용된다. 이러한 광범위한 사용은 RS 코드가 적은 수의 패리티만으로도 뛰어난 에러정정 능력을 보이며, Berlekamp-Massey 알고리즘, Euclid 알고리즘[1] 과 같은 효율적인 디코딩 알고리즘이 연구되어 왔기 때문이다. RS 코드는 k 개의 메시지 심볼에 $n-k$ 개의 패리티 심볼을 덧붙여 n 개의 코드워드 심볼을 만든다. RS 코드워드는 일반적으로 $GF(2^q)$의 원소인 q비트

* 학생회원, ** 정회원, 가톨릭대학교 정보통신전자공학부
(Department of Information, Communication, and Electronic Engineering, The Catholic University of Korea)

※ 이 논문은 2008년도 정부재원(교육인적자원부 학술연구조성사업비)으로 한국학술진흥재단의 지원을 받아 연구되었음(KRF-2008-313-D00743).”
※ 본 연구는 2010년도 가톨릭대학교 교비연구비의 지원으로 이루어졌음.
접수일자: 2010년5월17일, 수정완료일: 2010년11월1일

심볼로 이루어지고 하나의 심볼의 정정은 q비트의 정정의 효과를 보이므로 연집에러정정에 매우 적합하다.

전통적인 RS 디코더는 bounded minimum distance(BMD) 디코더로서, 실수 값을 갖는 채널을 통과한 코드워드를 수신 단에서 경판정을 통해 심볼로 변환하여 복호한다. BMD 디코더는 $t = \lfloor d_{\min}/2 \rfloor$의 디코딩반경을 갖는데, 여기서 $d_{\min} = (n-k+1)$이고, 코드워드 간의 최소거리를 뜻한다. 따라서 BMD 디코더는 이러한 디코딩 반경 내의 유일한 코드워드를 한 개 찾는다. 최근, Sudan[2]과 Guruswami and Sudan (GS)[3]은 다항시간(polynomial time)복잡도를 갖는 리스트 디코딩 알고리즘을 제안했다. 이 리스트 디코더는 $t' > \lfloor d_{\min}/2 \rfloor$의 디코딩 반경을 가지며, 하나 이상의 코드워드 리스트를 찾고 이중 가장 확률이 큰 코드워드를 고른다. GS 알고리즘은 $n-\sqrt{nk}$에 이르는 에러정정능력을 가지므로 코드율이 낮아질수록 더 높은 성능을 보인다[3]. RS 코드는 유한체 연산을 통한 대수적인 디코딩 과정을 가지므로 실수 값을 갖는 채널의 확률 정보를 활용한 연판정을 하기 어려운데, Koetter and Vardy(KV)가 채널의 확률을 정수로 변환하는 알고리즘을 제안함으로써 GS 리스트 디코더를 확장한 연판정 리스트 디코딩이 가능해졌다[4]. 연판정 디코더는 기존의 경판정 디코더에 비해 256-QAM 변조방식을 사용하여 Gaussian 채널에서 $0.25 \sim 1.5$dB, Rayleigh 채널에서 $2 \sim 9$dB의 코딩이득을 얻을 수 있었다[5~6].

RS 코드의 리스트 디코더는 Interpolation과 Factorization 과정을 거치는데, 모두 계산량이 매우 많고 latency가 크다. 따라서 이를 위한 효율적인 하드웨어의 구현이 필요하다. Interpolation은 리스트 디코더의 복잡도의 대부분을 차지하기 때문에 비교적 많은 연구가 이루어졌지만, Factorization은 계산량이 매우 많음에도 불구하고 많은 연구가 이루어지지 않았다. Factorization은 크게 근을 찾는 RC(Root Computation) 단계와 다항식을 갱신하는 PU(Polynomial Update) 단계로 이루어지는데 이중 전수조사(exhaustive search)를 이용한 RC 단계는 고정적으로 높은 latency를 가지므로 이를 해결하기 위한 연구가 주를 이루었다[9~11]. Zhang 등은 실험을 통해 Factorization 알고리즘의 반복문이 진행됨에 따라 구하고자 하는 근의 차수가 줄어드는 확률이 매우 작은 것을 발견하여 근의 차수 예측 방법(root order prediction)에 근거하여 직접적인 RC 방법을 제안하였고, 실험을 통해 첫 반복문에서 근의 차수가 다항식의 차수와 같은 확률이 높음을 발견하여 이방법을 더욱 보완하였다[9~10]. Ma 등은 다항식의 어파인 변환을 사용한 직접적인 RC 방법을 제안하였다[11]. 하지만 Zhang 등의 방법은 평균적인 성능은 향상시켰지만 예측의 실패확률이 존재하므로 일정한 성능을 보장하지 못하며, Ma 등의 방법은 다항식의 변환을 포함하여 하드웨어 복잡도가 높으며 하드웨어 비용이 매우 크다. 또한 고차다항식에 대한 일반해가 존재하지 않기 때문에 두 방법 모두 다항식의 차수를 낮게 제한하였으며, 근을 구하는 과정에 제곱, 제곱근의 계산을 포함하여 $GF(2^q)$ 원소의 표준기저와 정규기저 사이의 기저변환을 사용하기 때문에 추가 하드웨어가 필요하다. 앞서 제안된 구조는 병렬처리를 통해 Factorization의 높은 latency를 해결하였지만 이에 따라 동일한 하드웨어가 다항식의 차수만큼 사용되어 면적이 매우 크고 효율이 떨어진다. 따라서 우리는 새로운 스케쥴과 플래그를 통한 근의 발견 여부를 표시하는 방법을 통해 Factorization의 latency 문제를 극복하고 다항식의 차수에 따라 필요한 하드웨어가 다르다는 점을 이용하여 하드웨어를 재사용함으로서 성능을 향상시키고 면적을 효율적으로 줄인 구조를 제안한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서 리스트 디코더에 대한 간략한 소개와 Factorization 알고리즘, 선행연구가 이루어지고, Ⅲ장에서 새로운 스케쥴을 적용한 효율적인 Factorization 구조를 설명한다. Ⅳ장은 제안된 구조의 성능분석과 비교가 이루어지며, Ⅴ장에서는 결론을 맺고 본 논문을 마친다.

Ⅱ. RS 코드의 연판정 리스트 디코딩

$RS(n,k)$는 k 개의 데이터 심볼을 n 개의 코드워드 심볼로 부호화한다. 이때 각각의 심볼은 $GF(2^q)$의 원소이며, q 비트로 이루어진다. 부호화는 evaluation map이라는 방법으로 이루어지는데, 이는 일반적으로 사용되는 RS 코드와는 다른 원래의 정의에 따른다[7]. k 개의 메시지 심볼 $m_0, m_1, m_2, \ldots, m_{k-1}$을 다항식의 계수로 생각하면 다음과 같은 메시지 다항식이 생성된다.

$$f(x) = m_0 + m_1 x + m_2 x + \ldots + m_{k-1} x^{k-1} \qquad (1)$$

이 다항식에 n 개의 서로 다른 $GF(2^q)$의 원소를 대입하여 1 개의 코드워드를 얻는다. 일반적으로 $n=2^q-1$이며, 대입하는 원소는 0을 제외한 $\{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}$을 사용한다. 여기서 α는 원시원소이고, 대입을 통해 얻은 코드워드는 다음과 같다.

$$c = \{f(1), f(\alpha), f(\alpha^2), \ldots, f(\alpha^{n-1})\} \quad (2)$$

evaluation map방법으로 부호화된 코드워드의 복호는 필드의 2 차원 평면상에서 입력, 출력 값이 하나의 점으로 주어지고, 이 n 개의 점들을 지나는 $k-1$차 다항식을 구하는 Interpolation 문제로 귀결되며, 따라서 GS 알고리즘[3]에 의한 디코딩이 가능하다. 다음은 Interpolation 기반 디코딩 알고리즘에서 사용되는 몇 가지 정의이다.

Definition 1 : $Q(X, Y) = \sum_i^{\infty}\sum_j^{\infty} q_{i,j} X^i Y^j$을 $GF(2^q)$위에서 정의된 이변수 다항식이라 하고, w_x, w_y를 음이 아닌 정수라 하자. 그렇다면, $X^i Y^j$ 항의 (w_x, w_y)-weighted degree는 $i w_x + j w_y$이고, 다항식 $Q(X, Y) = \sum_i^{\infty}\sum_j^{\infty} q_{i,j} X^i Y^j$의 (w_x, w_y)-weighted degree는 0이 아닌 계수를 갖는 항중 가장 큰 값을 갖는 항의 (w_x, w_y)-weighted degree를 뜻한다.

Definition 2 : $P(X, Y)$ 가 $Q(X, Y)$를 각각 X, Y축으로 x, y만큼 평행 이동한 것이라 할 때, m보다 작은 차수의 모든 항의 계수가 모두 0이면, $Q(X, Y)$는 (x, y)를 multiplicity, m만큼 지나간다고 말한다. 이를 식으로 나타내면 다음과 같다.

$$\begin{aligned} P(X, Y) &= \sum_{\beta}^{r}\sum_{\alpha}^{w_t} p_{\alpha,\beta} X^{\alpha} Y^{\beta} \\ &= Q(X-x, Y-y) \\ &= \sum_{\beta}^{r}\sum_{\alpha}^{w_t} q_{\alpha,\beta} (X-x)^{\alpha} (Y-y)^{\beta} \end{aligned}$$

$$coef(Q(X-x, Y-y), X^{\alpha} Y^{\beta}) = p_{\alpha,\beta}$$

$$p_{\alpha,\beta} = 0, \ \forall \alpha+\beta < m \quad (3)$$

여기서 r은 Y의 최대차수이고, w_v 는 Y^v를 포함한 항 중에서 X의 최대차수를 나타낸다.

1. 리스트 디코더

그림 1은 RS 코드의 연판정 리스트 디코더의 블록도

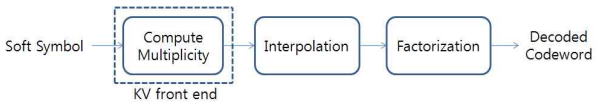


그림 1. RS 연판정 리스트 디코더 블록도
Fig. 1. RS soft decision list decoder block diagram.

이다. $GF(2^q)$상에서 정의된 (n, k)코드워드를 채널을 통해 수신하면 $2^q \times n$ 행렬(Π-행렬)로 표현할 수 있는데, Π-행렬의 각 원소 (x_i, y_j), $1 \le i \le n$, $1 \le j \le 2^q$는 y_j를 수신했을 때, x_i를 사용하여 코딩한 심볼을 전송한 확률을 의미한다. 따라서 Π-행렬은 채널정보를 담고 있다고 할 수 있는데, KV 알고리즘에 따라 Π-행렬을 정수 값을 갖는 M-행렬로 변환한다. 이 때 M-행렬의 각 원소는 Π-행렬에 비례하며, 이 중 0이 아닌 값을 해당 포인트의 multiplicity라고 한다. Interpolation 단계는 주어진 포인트가 해당하는 multiplicity 만큼 지나가는 이변수 다항식을 구하는 단계이며, KV 알고리즘에 의하면 일정 조건에서 이 다항식은 원래의 메시지 다항식을 인수로 포함한다[4]. Factorization은 Interpolation에서 구한 다항식중 하나를 골라 인수 분해하여 코드워드의 리스트를 구하는 과정이며, 마지막으로 이 코드워드를 Π-행렬과 곱하여 가장 확률이 높은 코드워드를 선택하여 디코딩 과정을 마친다. 리스트 디코더의 주요 단계를 요약하면 다음과 같다.

- Multiplicity calculation

  채널의 확률 정보를 사용하여 코드워드의 포인트마다 확률에 비례한 양의 정수인 multiplicity를 계산한다.
- Interpolation

  0이 아닌 multiplicity를 갖는 포인트 (x_i, y_j), $1 \le i \le n, 1 \le j \le 2^q$ 에 대해 multiplicity, $m_{i,j}$만큼 지나가며, 최소 $(1, k-1)$-weighted degree를 갖는 0이 아닌 이변수 다항식, $Q(X, Y)$를 구한다.
- Factorization

  Interpolation 단계에서 구한 이변수 다항식, $Q(X, Y)$를 인수 분해한다. 여기서 구한 다항식 인수는 $(Y-f(X))$형태를 가지며, $f(x)$는 후보 메시지가 된다. 따라서 $f(x)$의 차수는 k보다 작다.

2. Factorization

Interpolation 단계에서 구한 이변수 다항식을

Initialization : iteration level $i = 0$
Reconstruct($Q(X,Y)$, i){
find the largest nonnegative integer v, such that X^v divides $Q(X,Y)$
DIV_X : $\overline{Q}(X,Y) = Q(X,Y)/X^v$
RC : find all the roots of $\overline{Q}(0,Y)$ in $GF(2^q)$
for each root γ of $\overline{Q}(0,Y)$ do
$\Gamma[i] = \gamma$
if $i = k-1$
output $\Gamma[0],\Gamma[1],\Gamma[2],...,\Gamma[k-1]$
else
PU : $\widetilde{Q}(X,Y) = \overline{Q}(X,Y+\gamma)$
SHIFT : $\hat{Q}(X,Y) = \widetilde{Q}(X,XY)$
call Reconstruct($\hat{Q}(X,Y)$, $i+1$)
}

그림 2. Factorization 알고리즘
Fig. 2. Factorization algorithm.

$Q(X, Y)$라 가정한다. $Q(X, Y)$는 $GF(2^q)$위에서 정의되며 이를 수식으로 나타내면 다음과 같다.

$$Q(X, Y) = \sum_{t}^{r}\sum_{s}^{w_t} q_{s,t} X^s Y^t = (\sum_{\mu}^{w_r} q_{\mu,r} X^{\mu}) \prod_{\nu}^{r} (Y - f_{\nu}(X)) \quad (4)$$

$$f_{\nu}(X) = \sum_{\rho=0}^{k-1} f_{\rho}^{\nu} X^{\rho}, \ f_{\rho}^{\nu} \in GF(2^q)$$

식 (4)에서 $\sum_{\mu}^{w_r} q_{\mu,r} X^{\mu}$ 항은 인수분해에 영향을 주지 않으므로 본 논문에서는 편의상 $Q(X, Y)$를 식 (5)와 같이 정의한다.

$$Q(X, Y) = \prod_{\nu}^{r} (Y - f_{\nu}(X)) \quad (5)$$

그림 2는 $Q(X, Y)$를 인수분해하기 위한 Factorization 알고리즘이다[8]. 위 알고리즘은 차례로 상수항부터 $k-1$차수의 계수까지 k번의 반복문을 통해 하나의 인수다항식을 구하며 이를 r번 반복하여 r개의 후보다항식을 찾아낸다. 따라서 알고리즘은 최대 rk번의 반복문을 포함하며 이로 인해 latency가 매우 크다. 위 알고리즘의 출력인 길이 k의 r개 벡터는 각각 메시지를 복원하기 위한 후보 메시지가 되며, 이 후보 메시지 중 확률이 가장 큰 것을 고른다. 알고리즘은 다음과 같이 크게 4 개의 단계로 이루어진다.

1. DIV_X : $\overline{Q}(X, Y) = \dfrac{Q(X, Y)}{X^v}$
2. RC : find all root of $\overline{Q}(0, Y)$
3. PU : $\widetilde{Q}(X, Y) = \overline{Q}(X, Y+\gamma)$
4. SHIFT : $\hat{Q}(X, Y) = \widetilde{Q}(X, XY)$

이 중 DIV_X와 SHIFT는 계수가 저장된 메모리를 쉬프트 하거나 그의 주소를 연산함으로써 간단히 해결되지만 RC(Root Computation)와 PU(Polynomial Update)는 계산량이 많아 Factorization의 높은 latency의 요인이 된다. RC는 X 다항식의 상수항으로 이루어진 Y 다항식에 대한 r 개의 근을 구하는 과정이며 고차다항식의 근에 대한 일반해는 존재하지 않는다. 앞서 제안된 구조[11]는 일반식을 이용하여 직접 근을 구하기 위해 다항식의 차수를 제한하였다. 하지만 Y의 차수는 후보 다항식의 개수를 뜻하는 것으로 디코더의 에러정정능력을 결정한다. 따라서 높은 차수에도 적용 가능한 RC 방법이 필요한데, 대표적으로 전수조사방법(exhaustive search)이 있다. 전수조사방법은 단순하고 확실한 방법이지만 $GF(2^q)$의 모든 원소를 모두 대입해서 계산해야 하므로 $GF(2^q)$의 크기가 커질수록 latency가 커지는 문제가 있다. PU 단계는 다항식을 RC 단계에서 구한 근만큼 Y축으로 평행 이동하는 과정이며 일반적으로 다항식의 크기가 크므로 계산양이 매우 많다. PU 단계는 다음 RC 단계를 위해 다항식을 갱신하는 과정으로 알고리즘의 반복문을 수행함에 따라 다음 RC 단계에 영향을 주는 X 다항식의 차수는 점점 줄어들며 전체적인 Factorization의 복잡도가 증가된다.

3. Scheduling

이 장에서는 Factorization 알고리즘의 스케줄에 대해 다룬다. 스케줄은 크게 다항식 단위와 계수 단위의 스케줄로 나눌 수 있다. 그림 3은 Factorization의 인수다항식을 나타낸 것으로 가로축은 각각의 인수 다항식을, 세로축은 차수를 의미하며 상단의 상수항으로 시작하여 마지막 $k-1$차 항의 계수를 표시한다. 앞서 제안된 구조[9~11]은 다항식 단위의 스케줄로서 r 개의 다항식의 X 동차항 계수를 동시에 계산해 나가는 병렬처리방법을 사용하였다. 이 방법은 latency가 작지만 r 개의 다항식마다 레지스터와 프로세서가 필요하므로 하드웨어 비용이 크며, 중근을 갖는 경우 동일한 계산을 위해 여

러 개의 하드웨어가 같은 계산을 하게 되므로 하드웨어 효율이 떨어진다. 병렬처리의 하드웨어 비용문제를 해

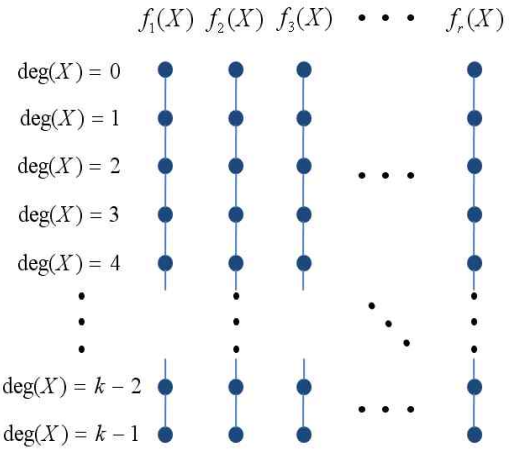
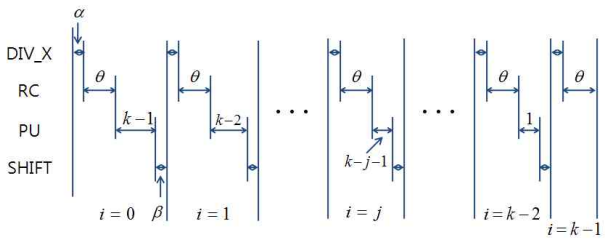
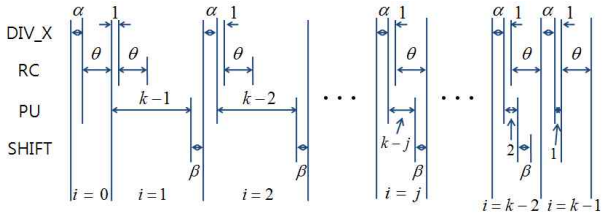


그림 3. 다항식의 근
Fig. 3. Root tree.



(a) sequential scheduling



(b) overlapped scheduling

그림 4. Factorization 알고리즘의 타이밍 다이어그램.
Fig. 4. Timing diagram of Factorization algorithm.

결하기 위해 순차처리를 사용할 수 있다. 순차처리는 두 가지가 있는데, 그림 3의 가로축 방향으로 진행하여 하나씩 아래로 내려가는 순서와 세로축 방향으로 진행하여 옆으로 진행하는 순서이다. 처음 방법은 여러 다항식의 동차항 계수를 순차적으로 구해나가는 것으로 각각의 다항식이 하나의 프로세서를 공유하기 때문에 하드웨어가 작고 효율이 높다. 하지만 갱신한 다항식을 저장하기 위한 레지스터가 다항식의 개수만큼 필요하다. 또 다른 순차처리방법은 하나의 인수다항식을 먼저 구하고 다음 다항식을 구하는 방법으로 위 방법과 같이 하나의 프로세서를 사용하며 한 번에 하나의 다항식을 구하므로 갱신한 다항식을 저장하기 위한 레지스터가 하나로 줄어들어 면적에서 매우 효율적이다. 하지만 위의 두 방법 모두 rk번의 반복문을 수행해야 하므로 latency는 증가한다.

그림 4는 계수 단위의 스케쥴을 나타내는 타이밍 다이어그램이다. θ, α, β는 각각 RC, DIV_X, SHIFT 단계의 latency를 나타낸 것으로 α, β는 RC, PU 단계에 비해 적은 클럭을 차지한다. 그림 4 (a)는 순차처리를 적용한 것으로 비교적 복잡도가 낮지만 각 단계마다 유휴 하드웨어가 있으므로 효율이 낮고 latency가 큰 단점이 있다. latency 문제는 내부 처리를 중첩함으로서 해결할 수 있는데, 그림 4 (b)는 중첩처리를 적용한 것을 나타낸다. RC 단계는 이전 PU 단계에서 처리한 계수 중 X에 대해 상수항만이 필요하므로 PU 의 첫 번째 출력 계수를 얻자마자 RC 단계를 시작할 수 있다. 이와 같은 RC와 PU의 중첩처리는 latency를 줄이고 하드웨어 효율을 높일 수 있지만 100%효율을 얻지 못하며 복잡도가 커진다.

Ⅲ. 제안한 구조

1. RC(Root Computation) Unit

RC 단계는 $\overline{Q}(0, Y)$의 근을 구하는 과정으로, 알고리즘의 반복문을 진행하면서 구하고자 하는 인수 다항식의 상수항부터 $k-1$차 항의 계수까지 차례로 구하게 된다. $Q(X, Y)$의 Y에 대한 최대 차수는 r로서 인수 다항식이 r개 존재한다는 것을 뜻하며 여기에는 중근이 포함된다. 따라서 r은 후보 메시지의 개수이며 이는 디코더의 성능을 결정하는 요소이다. 하지만 앞서 제안된 구조[9~11]는 직접적인 근을 구하는 여러 방법을 제시하지만, r을 낮은 값으로 제한함으로서 높은 r을 요하는 어플리케이션에는 사용하기 어렵다[11]. 본 논문에서는 차수에 상관없이 근을 구하는 전수조사 방법을 사용한다.

$$Q(0, Y) = (((q_{0,r} Y + q_{0,r-1}) Y + q_{0,r-2}) \ldots) Y + q_{0,0} \quad (6)$$

식 (6)은 honor's rule을 사용하여 다항식을 나타낸 것이다. 식에서 볼 수 있듯이 honor's rule은 곱셈과 덧셈의 반복으로 이루어지며 그 횟수는 다항식의 차수에 따라 결정된다. 하지만 이러한 전수조사를 이용한 방법은 latency가 매우 크다.

그림 5는 honor's rule을 사용한 전수조사 방법을 구현한 것이다. 그림과 같이 단위모듈로 나누면 임계경로

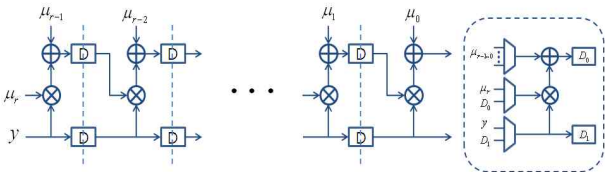


그림 5. RC 유닛과 R-MAC을 이용한 RC 유닛의 재구성
Fig. 5. RC unit and its reconfiguration using R-MAC.

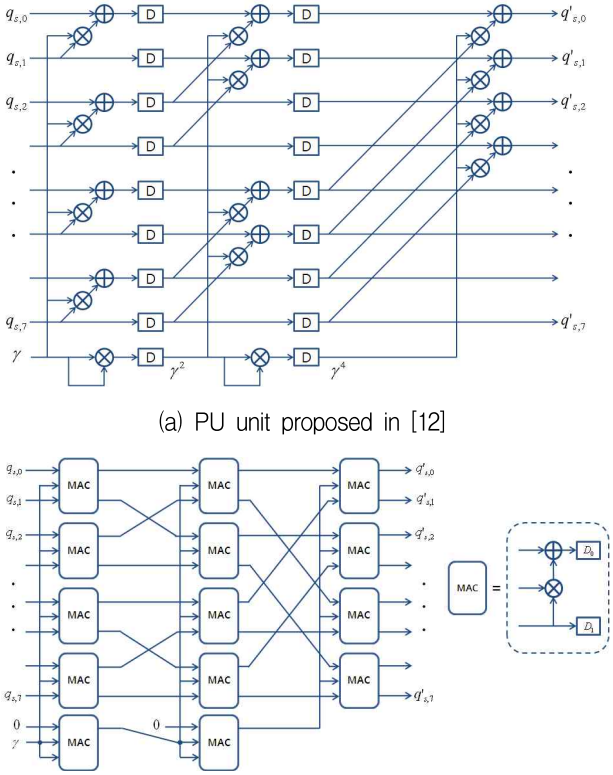
를 줄일 수 있으며 규칙적인 구조를 갖게 된다. 여기서 이 단위 유닛을 MAC(Multiply ACcumulate unit)이라 하며, 그림의 오른쪽과 같이 Mux를 추가하여 재구성이 가능한 것을 R-MAC(Reconfigurable-MAC)이라 한다. R-MAC은 세 개의 입력과 두 개의 출력을 갖는데, 출력에는 D_0, D_1 두 개의 레지스터를 사용하여 중간 값을 저장하도록 한다. 덧셈기의 입력은 곱셈기의 출력과 다항식의 계수들이며, 계수는 높은 차수에서부터 내림차순으로 선택되어 입력된다. 곱셈기는 왼쪽과 아래의 mux에서 입력을 받는데, 왼쪽의 mux는 다항식의 최고차항 계수와 D_0 레지스터를 선택하며, 아래의 mux는 근의 여부를 확인할 값, y와 D_1 레지스터를 선택한다. 이와 같이 구성한 경우, 하나의 값에 대해 근의 여부를 확인하기 위한 R-MAC의 latency는 다항식의 차수에 비례한다. 하지만 R-MAC은 하드웨어 비용이 작으므로 높은 병렬처리가 가능하다. 따라서 RC의 latency는 $\left\lceil \frac{2^q}{\sigma} \right\rceil \tau$가 된다. 여기서 q는 심볼의 비트수를 나타내며, τ는 해당 단계에서의 다항식의 차수를 의미하며, σ는 사용된 R-MAC의 개수를 의미한다.

2. PU(Polynomial Update) Unit

PU 단계는 다항식을 RC 단계에서 구한 근만큼 Y축으로 평행 이동하는 과정으로 식으로 나타내면 다음과 같다.

$$\widetilde{Q}(X, Y) = \overline{Q}(X, Y+\gamma) \tag{7}$$

그림 6 (a)는 [12]에서 제안된 PU 구조로서 $r=7$인 경우에 대해 그린 것이다. 입력은 동일한 X차수를 갖는 항의 계수이며 위에서부터 Y에 대해 0차, 1차, ... , r차이다. 위 구조는 r에 따라 확장가능하며 구조가 단순하고 면적이 작아 본 논문의 목적에 부합한다. 이 구조의 latency는 X의 최대 차수와 같으며, 일반적으로 매우 크기 때문에 병렬처리를 적용할 수 있다. 하지만



(a) PU unit proposed in [12]

(b) Reconfiguration of PU using MAC

그림 6. MAC을 이용한 PU 의 재구성
Fig. 6. Reconfiguration of PU using MAC.

앞서 스케줄에 대해 언급했듯이 RC와 PU단계의 중첩은 알고리즘 특성상 고르게 이루어지지 않으며, 이는 하드웨어 효율의 저하를 초래한다. 따라서 PU 유닛에 대해 높은 병렬처리를 적용할수록 하드웨어 효율이 낮아진다.

이러한 하드웨어 효율의 문제는 PU 구조를 MAC 단위로 분해함으로서 해결할 수 있다. 그림 6 (b)는 위 구조를 MAC을 사용하여 나타낸 것으로 입력에 맞게 배선을 다시 한 것이다. PU 단계에서 RC와 같이 R-MAC을 사용하는 방법은 latency가 매우 커지고 제어가 복잡해진다. 따라서 PU 구조는 MAC의 개수를 r에 따라 결정한다. 이러한 구조는 MAC에 mux를 추가하여 RC에서 사용된 R-MAC으로 재사용할 수 있기 때문에 매우 유용하다. 먼저, r에 따라 하나의 PU 에 사용되는 MAC의 개수를 구할 수 있으며, 높은 처리량이 요구될 경우 여러 개의 PU를 사용할 수 있으므로 단위 PU의 개수에 따라 전체 MAC의 개수가 결정된다. 각각의 MAC은 PU에서 입출력이 고정되어 있지만 mux를

추가하여 RC와 같이 입력을 추가하여 상황에 맞게 선택신호를 인가하면 R-MAC은 RC와 PU단계에서 모두

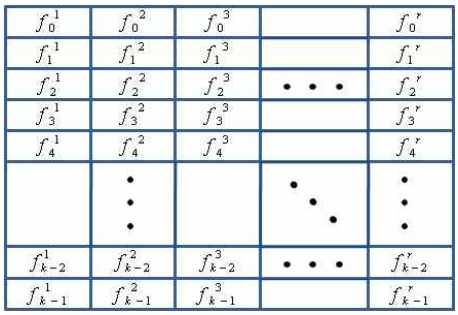


그림 7. 근의 플래그 레지스터
Fig. 7. Flag register for storing roots.

사용할 수 있게 된다. 이러한 방법은 하드웨어 효율을 높여주며 전체 구조가 단순해져 제어가 쉽다.

3. 근의 플래그 레지스터

$$\begin{aligned}\overline{Q}(0,\,Y) &= \overline{Q'}\,(0,\,Y)(Y+\gamma)^v \\ \overline{Q}(0,\,Y) &= \overline{Q}(0,\,Y+\gamma) = \overline{Q'}(0,\,Y+\gamma)\,Y^v\end{aligned} \tag{8}$$

그림 7은 근을 구했는지 여부를 표시하기 위한 플래그를 저장하는 레지스터를 나타낸 것이다. rk의 크기를 가지며 하나의 인수다항식 계수에 해당하는 플래그가 세로축을 따라 저장되고 가로축으로 r 개의 다항식이 해당된다. 외부에는 이와 같은 구조의 근을 저장하기 위한 메모리가 존재하며 이 메모리는 구한 근을 출력하기 위해 외부 인터페이스로 사용된다. Factorization 과정이 진행됨에 따라 메모리가 채워지며 외부에서 필요시 이 메모리에 접근하여 출력값을 읽을 수 있다. 하나의 인수다항식을 구하는 과정에서 i 번째 반복문의 RC 단계에서 구한 근은 메모리의 i 번째 행에 저장된다. 식 (8)과 같이 PU 단계를 거치고 나면 이 근의 차수를 알 수 있으므로 근의 차수만큼 같은 행의 메모리 셀에 근을 동일하게 저장하고 제어부에 위치한 플래그 레지스터에도 표시를 한다. 이와 같이 근을 구한 셀에 대해서 1 비트 플래그를 설정하면 다항식 단위의 스케쥴에 순차처리를 효율적으로 적용할 수 있다. 효율적인 순차처리에 대한 설명은 다음 장에서 다룬다.

4. 제안한 구조의 스케쥴

제안한 구조는 동일한 기본 구조를 가지고 단계마다 재구성하여 사용하므로 각 단계에서 모든 하드웨어가 집중되는 특성을 갖는다. 따라서 계수 단위의 스케쥴로서 순차처리를 사용하여 제어의 복잡도를 낮춘다. 그림 8은 제안한 구조의 계수 단위의 스케쥴을 나타낸 것이

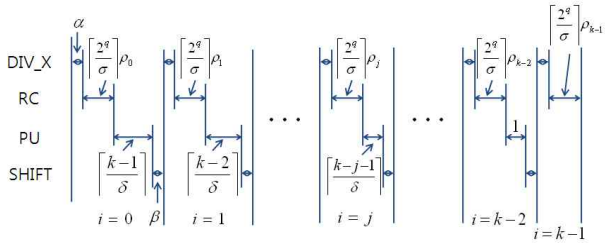


그림 8. 제안한 구조의 스케쥴
Fig. 8. proposed schedule.

다. 여기서 δ는 PU 유닛의 개수를 뜻하며 σ는 사용되는 전체 R-MAC의 개수를 의미하고 ρ_j는 j번째 반복문에서 근을 구하고자하는 다항식의 차수를 가리킨다. 제안한 구조는 구조가 단순하고 하드웨어 비용이 작기 때문에 다항식 단위의 스케쥴로 병렬처리를 적용할 수 있다. 하지만 우리는 면적이 중요한 어플리케이션을 위한 효율적인 순차처리 방법을 제안한다. 제안한 순서는 그림 3의 세로축 방향으로 하나의 인수다항식을 먼저 구하고 다음 인수다항식을 구하는 방법으로 프로세서와 중간 계산 값을 저장하기 위한 버퍼가 하나씩만 필요하므로 하드웨어 비용이 작지만 알고리즘을 rk번 반복해야 하므로 latency가 매우 크다. 하지만 앞 절에서 설명한 것과 같이 플래그 레지스터를 통해 이를 효율적으로 해결할 수 있다. PU 단계를 통해 근의 차수를 구하면 메모리 셀에 근을 차수만큼 동일하게 저장함과 동시에 플래그를 1로 설정하여 해당하는 셀의 근을 미리 구하여 두었음을 표시하고, 다음 인수다항식을 위한 반복문에서는 플래그의 설정 여부에 따라 RC 단계의 수행 여부를 결정한다. [9]는 근의 차수가 줄어드는 확률이 매우 작음을 실험을 통해 밝혔으며 [10]은 첫 반복문에서 다항식의 차수가 r과 매우 근접함을 발견하였으므로 플래그가 미리 설정되어 있을 확률은 매우 높다. 따라서 이 방법은 알고리즘의 반복문의 수를 효과적으로 줄여주며, 모든 근의 차수가 1인 경우 rk번의 반복문을 모두 수행해야 하지만 RC에서 하나의 근을 확인하기 위해 1 클록이 소요되므로 RC 단계의 전체 latency는 최소가 된다. 반대로 모든 근의 차수가 r인 경우, RC의 단위 latency는 r로서 최대가 되지만, 전체 알고리즘의 반복문은 줄어들어 전체적인 latency는 낮아진다. 모든 근의 차수가 r로 같지만 알고리즘 반복의 중간에 항의 계수가 달라질 경우 더 많은 반복문을 수행하여

latency가 커지지만 이러한 경우는 확률적으로 매우 드물며, 버퍼를 적절히 사용하여 해결할 수 있다.

5. 전체 구조

그림 9는 제안한 전체 Factorization 구조를 나타낸 것이다. R-MAC array는 R-MAC의 모음을 나타낸 것이며 컨트롤 신호에 따라 PU 유닛과 RC 유닛으로 재구성되어 사용된다. 제안된 구조는 두 개의 다항식을 위한 메모리가 사용되는데, 그림에서 아래의 SRAM은 처음 입력된 원본 다항식, $Q(X, Y)$을 저장하며, 위의 SRAM은 반복문을 수행하면서 갱신된 다항식을 저장한다. 제안된 구조는 R-MAC을 많이 사용할수록 성능이 향상되지만 병렬처리를 적용하면 한 번에 많은 양의 데이터를 읽고 써야하는 메모리의 입출력 대역폭의 문제가 발생한다. 이러한 대역폭의 문제는 SRAM을 여러 개의 뱅크로 나누어 사용함으로써 해결할 수 있다. Root storage는 Factorization이 진행되면서 계산되어 출력된 근을 저장하며, 외부 모듈과의 인터페이스를 위해 사용된다.

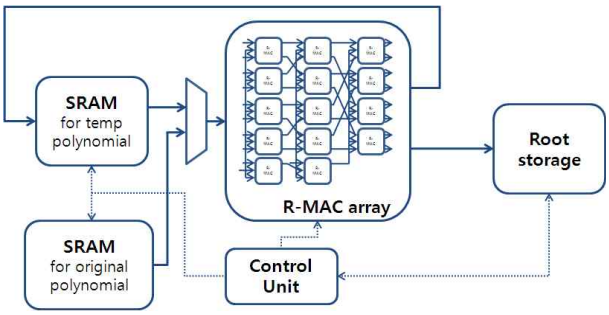


그림 9. 전체구조
Fig. 9. overall architecture.

Ⅳ. 설계 및 성능 분석

이 장에서는 제안한 구조의 하드웨어 비용과 성능을 살펴본다. 비교를 위해 [9]와 같이 $(n, k) = (255, 239)$, $r = 6$의 경우에 대해 분석하였다. 또한 re-encoding을 사용하면 Interpolation과 Factorization의 latency를 알고리즘 차원에서 효과적으로 줄일 수 있는데, Factorization은 k에서 2τ로 그 길이를 줄일 수 있다[13]. 여기서 τ는 오류 정정 능력으로 $\tau = 13$을 사용하여 반복문의 횟수는 $2\tau = 26$이 된다. 원시다항식은 $GF(2^8)$에서 $p(x) = 1 + x^2 + x^3 + x^4 + x^8$를 사용하였다.

표 1은 1, 2개의 PU를 사용했을 경우에 사용된 R-MAC의 개수와 RC의 latency를 계산한 것이다. $r = 6$이므로 1개의 PU는 11 개의 R-MAC이 사용된다. 다항식의 차수가 낮을수록 RC단계의 latency가 줄어드는 것을 볼 수 있다. 표 2는 계수 단위의 스케쥴에 대해서 순차처리와 중첩처리, 제안한 구조의 스케쥴의 latency를 비교한 것으로 이론값과 실제 값을 대입한 것을 나타내었다. $\alpha = \beta = 2$이며, RC는 전수조사방법을 가정하므로 $\theta = 256$이 된다. $k = 239$ 대신 re-encoding의 결과인 $2\tau = 26$을 적용하고, 각 스케쥴의 latency는 RC와 PU 모듈 1개씩을 사용함을 가정하였다. 제안한 구조의 경우 $\sigma = 11$이며 $\delta = 1$이다. ρ_i는 모든 근이 중근을 갖는 경우($\rho_i = 6$)와 개별 근을 갖는 경우($\rho_i = 1$)로 나누어 계산하였다. 마지막으로 세 경우 모두 다항식 단위의 스케쥴에 순차처리를 가정하므로

표 1. RC 성능 계산
Table 1. RC performance.

order of root	# of R-MAC		latency(clk)	
	PU:1	PU:2	PU : 1	PU : 2
6	11	22	$\lceil \frac{256}{11} \rceil \times 6 = 144$	$\lceil \frac{256}{22} \rceil \times 6 = 72$
5			$\lceil \frac{256}{11} \rceil \times 5 = 120$	$\lceil \frac{256}{22} \rceil \times 5 = 60$
4			$\lceil \frac{256}{11} \rceil \times 4 = 96$	$\lceil \frac{256}{22} \rceil \times 4 = 48$
3			$\lceil \frac{256}{11} \rceil \times 3 = 72$	$\lceil \frac{256}{22} \rceil \times 3 = 36$
2			$\lceil \frac{256}{11} \rceil \times 2 = 48$	$\lceil \frac{256}{22} \rceil \times 2 = 24$
1			$\lceil \frac{256}{11} \rceil = 24$	$\lceil \frac{256}{22} \rceil = 12$

표 2. process 방법에 따른 latency
Table 2. latency of each process.

process	latency	
sequencial	$\alpha k + \theta k + \frac{(k-1)k}{2} + \beta(k-1)$	
	7083 clk	
overlap	$\alpha k + \theta + \sum_{i=1}^{k-1} max(\theta+1, k-i)$	
	6733 clk	
proposed	$\alpha k + \beta(k-1) + \sum_{i=0}^{k-1} \lceil \frac{2^q}{\sigma} \rceil \rho_i + \sum_{i=0}^{k-2} \lceil \frac{k-i-1}{\delta} \rceil$	
	$\rho_i = 6$	4171 clk
	$\rho_i = 1$	6306 clk

순차처리와 중첩처리의 경우 이론값에 $r=6$을 곱해주어야 하며 제안한 구조의 스케쥴에서 $\rho_i=6$의 경우는 모든 근이 중근을 가지므로 하나의 인수다항식을 구하는 것으로 Factorization 과정을 마친다. 단순한 순차처리의 latency가 가장 크고, 중첩처리의 latency는 더 작은 것을 볼 수 있다. 제안한 스케쥴은 표 1에서 볼 수 있듯이 근의 차수가 가장 큰 경우에 RC의 latency가 가장 컸지만 여기서 구한 근은 중근이므로 다음 다항식의 처리에서 RC 단계를 생략할 수 있어 전체 알고리즘의 latency가 작아진다는 것을 확인할 수 있다.

하나의 인수다항식을 위해 소요되는 PU 단계의 클록은 $\sum_{i=1}^{2\tau-1} i = 325\,clock$이므로 RC 단계를 제외하면 $PU+2\tau(\alpha+\beta)-2 = 325+26\times4-2=427$ 이다. 이를 바탕으로 먼저 1 개의 PU 유닛을 사용한 경우의 latency를 계산하면 다음과 같다.

1) 모든 근의 차수가 1인 경우 :

$$RC(6)+r((2\tau-1)RC(1)+\alpha+\beta+PU)$$
$$=144+6*(25*24+4+427)=6330\,clk$$

2) 모든 근의 차수가 6인 경우 :

$$2\tau RC(6)+(\alpha+\beta+PU)$$
$$=26*144+4+427=4175\,clk$$

3) 모든 근의 차수가 6이고, 마지막 근의 차수가 1인 경우 :

$$(2\tau-1)RC(6)+r(RC(1)+\alpha+\beta+PU)$$
$$=25*144+6*(24+4+427)=6330\,clk$$

1개 PU 유닛을 사용한 경우, 약 6500 클록 이내의 latency를 보이며 이러한 구조는 성능에 비해 면적이 중요한 어플리케이션에 적합하다.

동일하게 2 개의 PU 유닛을 사용한 경우를 계산하면 PU 단계는 $\sum_{i=1}^{2\tau-1}\left\lceil \frac{i}{2} \right\rceil = 169\,clk$의 latency를 가지며, RC 단계를 제외하면 최대 $PU+2\tau(\alpha+\beta)-2$ $=169+26\times4-2=271\,clk$이다. 따라서 2 개의 PU 유닛을 사용한 경우의 latency는 다음과 같다.

1) 모든 근의 차수가 1인 경우 :

$$RC(6)+r((2\tau-1)RC(1)+\alpha+\beta+PU)$$
$$=72+6*(25*12+4+271)=3522\,clk$$

2) 모든 근의 차수가 6인 경우 :

$$2\tau RC(6)+(\alpha+\beta+PU)$$
$$=26*72+4+271=2147\,clk$$

3) 모든 근의 차수가 6이고, 마지막 근의 차수가 1인 경우 :

$$(2\tau-1)RC(6)+r(RC(1)+\alpha+\beta+PU)$$
$$=25*72+6*(12+4+271)=3522\,clk$$

2 개의 PU 유닛을 사용한 경우 4,000 클록 이내의 latency를 보이므로 성능이 중요하여 높은 처리량을 보이는 Interpolation 구조[14~15]를 사용한 어플리케이션에 적합하다. 이와 같이 제안한 구조는 확장성이 좋아 다양한 Interpolation 구조에 대해서 적용이 가능하다.

제안한 구조는 제어부와 함께 Verilog HDL로 모델링 되었으며, Synopsys Design Compiler를 이용하여 게이트 수준의 합성 결과를 얻었다. 동부 아남 0.18㎛ 표준 셀 라이브러리를 사용하여 합성한 결과, 최대 동작 주파수는 330㎒이며 약 20K게이트가 사용되었다. 앞서 제안된 구조는 알고리즘 레벨에서 게이트와 성능을 계산하여 임계 경로 등의 자세한 성능을 계산하기 어렵고 따라서 제안한 구조와 비교가 어렵다. [9~10]의 구조는 확률적으로 823clock의 성능을 보이지만 고정적이지 않으며, latency가 중요한 어플리케이션에서는 사용할 수 없다. [11]은 더 높은 성능을 보이지만 다항식변환을 위한 하드웨어가 필요하여 비용이 매우 크다. 두 구조 모두 RC 과정이 기저변환, 역과정을 포함하여 추가 하드웨어가 필요하다. 또한 낮은 차수의 다항식을 갖는 응용분야로 적용이 제한되며 일반적으로 더 높은 차수의 다항식에는 적용이 불가능하다. 반면 제안한 구조는 2 개의 PU 유닛만으로 약 $4{,}000\,clock$ 이내의 latency를 약속하며 더 높은 성능을 원할 경우 추가 비용이 매우 적고 높은 효율과 성능을 얻을 수 있다. 또한 특정 스케쥴에 제한되지 않고 앞서 사용된 병렬처리에도 적용이 가능하여 더 높은 성능을 얻을 수 있다. 면적이 중요한 어플리케이션이라면 제안한 스케쥴을 사용하여 프로세서와 레지스터를 줄일 수 있으며 제안한 플래그 방법은 순차처리의 latency 문제를 효율적으로 해결한다.

V. 결 론

본 논문은 연판정 RS 리스트 디코더의 Factorization

에 대해 하드웨어 비용이 적고 효율이 좋은 구조를 제안하였다. 제안한 구조는 1) 하드웨어 비용이 작고 구조가 단순하며 2) RC, PU 유닛에서 모두 사용가능한 동일한 기본 구조를 가지므로 재사용이 가능하여 하드웨어 효율이 높다. 또한 3) 성능에 따라 확장성이 좋고, 4) RC 단계에서 전수조사방법을 사용하여 다항식의 차수가 높은 어플리케이션에도 적용이 가능하다. 5) 제안한 스케쥴은 각 단계가 순차적으로 진행되므로 제어가 간단하며, 6) 플래그를 사용하여 근의 저장 여부를 표시하는 방법은 순차처리로 인한 latency를 효율적으로 줄여준다. PU 유닛을 2 개 사용한 경우에 대해 제안한 구조를 동부 아남 0.18㎛ 표준 셀 라이브러리를 사용하여 합성한 결과 약 20K의 면적으로 최대 330㎒로 동작이 가능하며, 성능이 중요한 어플리케이션에 적용이 가능하다.

감사의 글

저자들은 본 연구를 위하여 설계 환경을 제공하여준 IDEC에 감사드린다.

참 고 문 헌

[1] R. E. Blahut, *Theory and practice of Error Control Codes*, Addison-Wesley, Reading MA, 1983.

[2] M. Sudan, "Decoding of Reed-solomon codes beyond the error correction bound," *J. complexity*, vol. 12, pp. 180-193, 1997.

[3] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon and algebraic-geometric codes," *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 1755-1764, Sep. 1999.

[4] R. Koetter and A. Vardy, "Algebraic soft-ecision decoding of Reed-Solomon codes," *IEEE Trans. Inf. Theory*, vol. 49, no. 11, pp. 2809-2825, Nov. 2003.

[5] W. J. Gross, F. R. Kschischang, R. Koetter, and P. Gulak, "Simulation results for algebraic soft-decision decoding of Reed-Solomon codes," in Proc. 21st Biennial symp. Commun., pp. 356-360, 2002.

[6] W. J. Gross, F. R. Kschischang, R. Koetter, and P. Gulak, "Applications of algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Trans. Commun.*, vol. 54, no. 7, pp. 1224-1234, Jul. 2006.

[7] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *SIAM Journal of Applied Mathematics*, vol. 8, pp. 300-304, 1960.

[8] R. M. Roth and G. Ruckenstein, "Efficient decoding of Reed-Solomon codes beyond half the minimum distance," *IEEE Trans. Inform. Theory*, vol. 46, pp.246-258, 2000.

[9] X. Zhang and K. Parhi, "Fast factorization architecture in soft decision Reed-Solomon decoding," *IEEE Trans. VLSI. Syst.*, vol. 13, no. 4, pp. 413-426, April. 2005.

[10] X. Zhang, "Further exploring the strength of prediction in the factorization of soft-decision Reed-Solomon decoding," *IEEE Trans. VLSI. Syst.*, vol. 15, no. 7, pp. 811-820, July. 2007.

[11] J. Ma, A. Vardy, and Z. Wang, "Low-latency factorization arhitecture for algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Trans. VLSI. Syst.*, vol. 15, no. 11, pp. 1225-1238, Nov. 2007.

[12] A. Ahmed, R. Koetter, and N. Shanbhag, "VLSI architectures for soft-decision decoding of Reed-Solomon codes," in Proc. ICC, 2004, pp. 2584-2590.

[13] R. Koetter and A. Vardy, "A Complexity Reducing Transformation in Algebraic List Decoding of Reed-Solomon Codes," IEEE ITW2003, pp. 10-13, Paris, France, Mar. 2003.

[14] Z. Wang and J. Ma, "High-speed interpolation architecture for soft-decision decoding of Reed-Solomon codes," *IEEE Trans. VLSI systems*, vol. 14, no. 9, pp. 937-950, Sep. 2006.

[15] W. J. Gross, F. R. Kschischang, and P. Gulak, "Architecture and implementation of an interpolation processor for soft-decision Reed-solomon decoding," *IEEE Trans. VLSI systems*, vol. 15, no. 3, pp. 309-318, Mar. 2007.

저 자 소 개

이 성 만(정회원)

2008년 가톨릭대학교 정보통신전자공학과 졸업.
2010년 가톨릭대학교 정보통신전자공학과 석사 졸업.
2010년～현재 MTH 연구소 ASIC팀 연구원

<주관심분야 : VLSI 설계, 통신신호처리, SOC 설계>

박 태 근(정회원)-교신저자

1985년 연세대학교 전자공학과 졸업.
1988년 Syracuse Univ. Computer 공학석사 졸업.
1993년 Syracuse Univ. Computer 공학박사 졸업.
1991년～1993년 Coherent Research Inc. USA VLSI 엔지니어.
1994년～1998년 현대전자 System IC 연구소 책임연구원
1998년～현재 가톨릭대학교 정보통신전자공학부 교수

<주관심분야 : VLSI 설계, CAD, 컴퓨터 구조>