저면적, 설정 가능 Reed-Solomon 코더의 설계

이 현 용, 박 인 철
KAIST 전기 및 전자과
전화 : 042-869-8061 / 핸드폰 : 011-9511-1587

Area-efficient VLSI Implementation of A Configurable Reed-Solomon Coder

Hyun-Yong Lee, In-Cheol Park
Dept. of Electrical Engineering, KAIST
E-mail : hylee@ics.kaist.ac.kr

Abstract

This paper present an area-efficient RS coder that can process all GF(256) codes. First, we selected algorithms to reduce the total area of the RS coder. In applications that utilize multiple RS codes, we can reduce the size of the hardware needed for supporting RS codes by making the RS hardware support multiple RS codes. The proposed RS coder can deal with multiple RS codes at the cost of a small additional area.

I. 서론

Reed-Solomon 코드는 VDSL, DVD, xDSL, W-CDMA, 위성 통신 등 가장 널리 이용되고 있는 에러 정정 코드이다. RS 코드는 primitive field generator polynomial, length, dimension, generator polynomial에 의해서 그 특성이 정해지게 된다.

RS 코드를 이용하는 어플리케이션 중에는 서로 다른 특성을 갖는 RS 코드들을 가지는 경우가 많이 있다. 예를 들어 DVD의 경우, row는 (182, 172), column은 (208, 192)의 RS 코드를 이용한다. 이 경우 time sharing으로 하나의 하드웨어로 row, column의 RS 코드를 모두 처리하면 전체 하드웨어 면적을 줄일 수 있다. [4]

따라서 본 논문에서는 RS의 n, k값을 입력 받아서 단일 하드웨어로 GF(256)상의 여러 RS 코드들을 지원할 수 있는 하드웨어를 설계하였다. 또한 최소의 하드웨어를 필요로 하는 알고리즘들을 선택하여 적은 면적으로 이를 구현하였다.

II. 본론

2.1 Reed-Solomon 코드

RS 코드는 기본적으로 linear block code이며, (n, k)로 표현되는 RS 코드의 block은 다음과 같은 구조를 가진다.

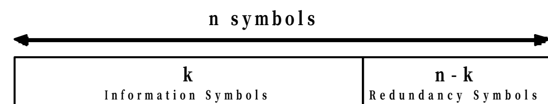

그림 1. RS 코드 block

여기서 n은 information 신호의 symbol 개수, k는 오류 정정에 이용되는 redundancy 신호의 symbol 개수이다. 또한 각각의 symbol들은 Galois Field (2^m)의 원소이다.

RS 코드를 생성하는 generator polynomial은 다음과 같이 정의된다.

$$g(x) = (x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{2t})$$

t는 RS code의 오류 정정 능력을 나타내며, $t = \left\lfloor \frac{n-k}{2} \right\rfloor$개의 symbol 오류를 정정할 수 있다. [1]

Shortened RS code는 기존의 $n=2^m-1$인 (n, k) RS 부호에서 information 신호의 일부 symbol을 삭제한 것이다. 따라서 (n-s, k-s) 형태의 값을 가진다. 이는 $n=2^m-1$인 경우에 information 신호의 상위 s개 symbol이 0으로 된 것으로 생각하여 처리한다.

2.2 적은 면적의 범용 Reed-Solomon 코더 설계

앞에서 언급한 바와 같이 본 논문의 하드웨어에서 지원하는 RS 코드는 m=8인 GF(256)로 한정하고 있다. RS 코드의 GF가 정해지면 multiplier의 구조가 고정되어 기존 구조에 큰 변화가 없이 구현이 가능하다. 만약 GF를 변경할 수 있도록 하려면 general multiplier가 필요하게 되며 이는 칩 전체 면적을 크게 증가시키게 된다. 그리고 대부분의 어플리케이션은 1byte씩 신호를 처리하는 RF(256)의 RS 코드를 사용하고 있다.

2.2.1 Encoder 설계

RS 코드 (c(x))는 cyclic, linear하므로 자신의 generator polynomial를 인수로 가져야 한다. 이를 이용하여 information 신호 (m(x))를 generator polynomial (g(x))로 나눈 나머지로 redundancy 신호 (p(x))를 얻는다. [2]

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

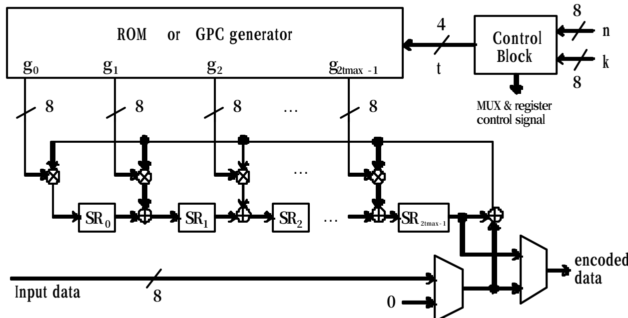

그림 2. 설계한 encoder 구조

Control block은 내부적으로 counter를 가지고 입력 신호의 횟수를 세면서 전체적인 동작을 조절한다. k번의 iteration으로 나머지를 계산하는 동안은 입력 신호를 그대로 출력으로 내보내고, 나머지 n-k번의 iteration에는 shift register에 저장되어 있는 나머지 값을 출력한다.

위의 연산을 위해서 각각의 t에 대한 generator polynomial이 필요하다. 이를 구하는 방법으로 generator polynomial의 정의로부터 2t번의 iteration 곱셈을 통해 계수를 얻는 방법을 생각해 볼 수 있다. 이 하드웨어는 다음과 같이 설계할 수 있다.

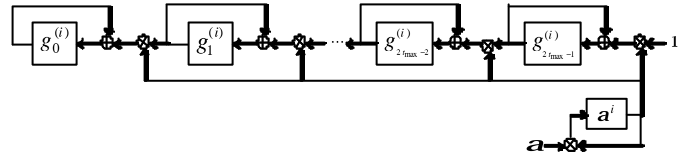

그림 3. Generator polynomial coefficient의 generator

t_{max}는 지원해야 하는 t값 중 가장 큰 값을 의미한다. 위의 하드웨어로 2t 번의 iteration을 통해 우측부터 (2t-1)개의 register에서 x^{2t}을 제외하고 높은 차수 순서로 계수를 얻을 수 있다. 즉, 주어진 t의 generator polynomial를 ROM의 출력인 $g_i(0 \le i \le 2t_{max})$으로 표현하면 다음과 같다.

$$g_t(x) = x^{2t} + \sum_{n=0}^{2t-1} g_{2t_{max}-2t+n}x^n$$

이 경우 모든 t값을 지원하는 장점을 가지지만, t값이 바뀔 때마다 2t cycle씩 기다려야 하는 단점이 있다.

generator polynomial의 계수를 얻는 다른 방법으로 ROM을 이용하는 방법이 있다. 어플리케이션에서 필요로 하는 t값이 고정되어 있으며 n, k값이 자주 바뀌는 경우에 지원해야 하는 t값들의 generator polynomial 계수를 미리 계산하여 ROM에 기록해 두는 것이다.

주어진 t의 generator polynomial를 ROM의 출력인 $g_i(0 \le i \le 2t_{max})$으로 표현하면 역시 다음과 같다.

$$g_t(x) = x^{2t} + \sum_{n=0}^{2t-1} g_{2t_{max}-2t+n}x^n$$

즉, g(x)의 계수에서 x^{2t}를 제외한 높은 차수 순서대로 ROM의 상위값을 채워나가야 한다. 따라서 우측부터 2t개의 cell만이 나눗셈 연산에 사용이 되며, 나머지 부분의 계수에는 0이 들어가므로 나눗셈 연산에 영향을 주지 않는다.

Shortened RS의 경우 앞에서 삭제된 s개의 information은 0으로 처리가 되므로 나머지 연산에 영향을 주지 않는다. 따라서 reset후에 바로 information 신호를 받아서 주어진 k만큼 iteration하면 된다.

2.2.2 Decoder 설계

RS decoder의 경우는 RS encoder와 달리 generator polynomial의 계수가 필요 없다.

아래와 같이 3단계의 pipeline 구조로 설계한다.

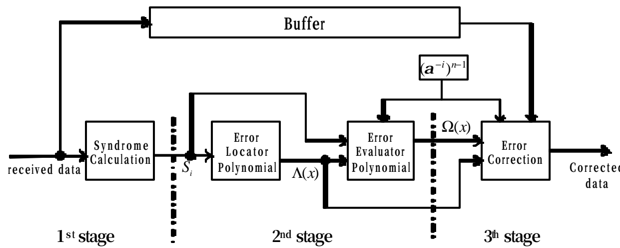

그림 4. 설계한 decoder 전체 구조

Pipeline 구조와 관계없는 것으로 buffer와 α^{256-n}을 계산하는 block이 있다.

Buffer의 역할은 error가 계산될 때까지 원래의 데이터를 저장해두는 것이다. 따라서 입력신호가 들어와서 error 계산되어 나오는 데까지 걸리는 latency가 적을수록 buffer의 크기를 줄일 수가 있다. Buffer는 다른 부분에 비해 전체 decoder에서 매우 큰 면적을 차지하고 있기 때문에 전체 면적을 줄이는데 매우 중요하다.

그 외 n값이 변할 때 사용되는 α^{256-n}값을 계산하는 block이 있다.

(1) 1st stage

(1.1) Syndrome 계산

n cycle동안 들어오는 수신 신호(r(x))에 $\alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{2t-1}$을 대입한 값을 계산한다. c(s)는 generator polynomial를 인수로 가지기 때문에 error polynomial의 $e(\alpha^0), e(\alpha^1), e(\alpha^2), \ldots, e(\alpha^{2t-1})$값을 얻을 수 있다.

$$r(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i)$$

다음의 식으로 iteration를 통해 syndrome를 구할 수 있다.

$$S_k = (\ldots((r_{n-1}\alpha^k + r_{n-2})\alpha^k + \ldots + r_1)\alpha^k + r_0)$$

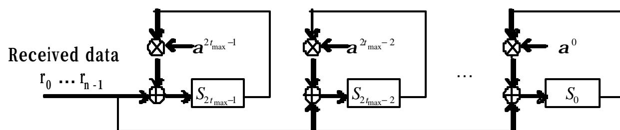

그림 5. Syndrome 계산 구조

위의 하드웨어에서 n cycle후에 각 cell들이 최종 syndrome를 갖게 된다. $t<t_{max}$의 경우 좌측 $2(t_{max}-t)$개의 cell은 동작할 필요가 없으나 면적을 줄이기 위해 특별히 제어를 하지 않았다.

Shorten RS의 경우 상위 s개는 0이므로 계산할 필요가 없으며 n번의 iteration 후에 바로 다음 단계로 진행하면 된다.

(2) 2nd stage

(2.1) Error Locator polynomial

Berlekamp-Messay (BM) 알고리즘이 Euclid 알고리즘에 비해서 하드웨어 구현에서 적은 면적을 차지하므로 이를 선택하였다. 또한 하드웨어 크기를 줄이기 위해서 inversion이 필요 없도록 수정된 inverse-free BM 알고리즘을 이용하였다. [3]

BM 알고리즘은 error 위치의 역수 값이 α의 차수가 되는 근을 가지는 error locator polynomial를 구하는 방법으로 Newton's Identities를 이용하여 위에서 계산한 2t개의 syndrome를 차례대로 하나씩 만족시키도록 수정해 가는 것이다. 따라서 2t번의 iteration이 필요하다.

앞에서 언급한 바와 같이 buffer의 면적이 전체 decoder 면적에 가장 영향이 크다. 따라서 serial BM과 parallel BM 중에서 적은 latency를 가지는 parallel BM 구조를 사용하여 buffer의 면적을 줄였다. Serial BM 구조는 BM 알고리즘 계산에 필요한 연산의 수는 줄어들지만 latency가 $2t^2$ cycle로 parallel의 2t cycle에 비해서 상당히 크기 때문에 상대적으로 buffer가 매우 커지게 된다.

위에서 선택한 parallel inverse-free BM 알고리즘의 식을 정리하면 다음과 같다.

$$\Delta_i = \sum_{j=0}^{i-1} \Lambda_j^{(i-1)} S_{i-j}$$

$$L_i = \delta(i - L_{i-1}) + (1-\delta)L_{i-1}$$

$$\begin{bmatrix} \Lambda^{(i)}(x) \\ B^{(i)}(x) \end{bmatrix} = \begin{bmatrix} \epsilon^{(i-1)} & -\Delta_i x \\ \delta & (1-\delta)x \end{bmatrix} \begin{bmatrix} \Lambda^{(i-1)}(x) \\ B^{(i-1)}(x) \end{bmatrix}$$

$$\epsilon^{(i)} = \delta\Delta_i + (1-\delta)\epsilon^{(i-1)}, \mathrm{for}\, i = 1, 2.., 2t$$

$$\begin{cases} \delta = 1 & , \mathrm{if}\, \Delta_i \neq 0, 2L_{i-1} \leq i-1 \\ \delta = 0 & , otherwise \end{cases}$$

초기 조건은 다음과 같다.

$$\Lambda^{(0)}(x) = 1,\ B^{(0)}(x) = 1,\ L_0 = 0,\ \epsilon^{(0)} = 1$$

위의 식으로부터 다음과 같은 하드웨어를 얻을 수 있다.

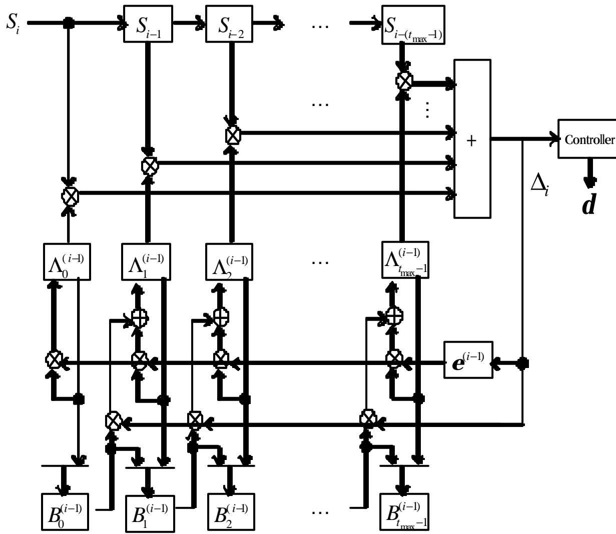

그림 6. BM 알고리즘 구조

그리고 2t cycle후에 다음 3번째 단계로 넘어가도록 하면 1st stage에서 계산한 여분의 $2(t_{max}-t)$개 syndrome을 무시할 수 있다.

(2.2) Error Evaluator polynomial

Error evaluator polynomial을 얻는 데는 두 가지 방법이 있다. 하나는 BM의 iteration 동안 error locator polynomial를 계산하면서 동시에 error evaluator polynomial을 같이 계산하는 방법이 있으며, 다른 방법으로는 error locator polynomial의 최종 결과를 이용하는 것이다. 전자가 후자보다 전체 latency가 t cycle 줄어들어서 buffer의 크기가 다소 줄어드는 장점이 있으나, 전자를 구현함으로써 생기는 하드웨어의 면적의 증가가 상대적으로 더 크다. 따라서 후자의 방식으로 error evaluator polynomial을 계산하는 방법을 선택하였다.

$$\Omega(x) = [1 + S(x)]\Lambda(x) \bmod x^{wt-1}$$

(3) 3rd stage

(3.1) Error Locator 계산 (Chien Search)

Chien search는 error locator polynomial인 $\Lambda(x)$에 가능한 경우수를 모두 대입하여 $\Lambda(x)=0$의 근을 구하는 방법이다. 즉, $\Lambda(\alpha^{-i})=0$의 경우 i번째 symbol에 error가 있음을 알 수 있다. $(0 \le i \le n-1)$

따라서 입력이 들어온 높은 차수 순서대로 error를 확인할 수 있도록 $\Lambda(x)$에 $\alpha^{-(n-1)}, \alpha^{-(n-2)}, \ldots, \alpha^{-0}$ 순서로 대입하여 0이 되는 가를 확인해야 한다.

이를 구현하는 방법으로 Chien Search가 있다.

$$\begin{aligned}\Lambda(\alpha^{-(n-1)}) &= \Lambda_{2t}(\alpha^{-(n-1)})^{2t} + \Lambda_{2t-1}(\alpha^{-(n-1)})^{2t-1} + \ldots + \Lambda_0(\alpha^{-(n-1)})^0 \\ &= \Lambda_{2t}(\alpha^{-2t})^{(n-1)} + \Lambda_{2t-1}(\alpha^{-(2t-1)})^{(n-1)} + \ldots + \Lambda_0(\alpha^{-0})^{(n-1)} \\ &= \Lambda_{2t}(\alpha^{2t})^{-n+1} + \Lambda_{2t-1}(\alpha^{2t-1})^{-n+1} + \ldots + \Lambda_0(\alpha^0)^{-n+1}\end{aligned}$$

$$\begin{aligned}\Lambda(\alpha^{-(n-2)}) &= \Lambda_{2t}(\alpha^{-(n-2)})^{2t} + \Lambda_{2t-1}(\alpha^{-(n-2)})^{2t-1} + \ldots + \Lambda_0(\alpha^{-(n-2)})^0 \\ &= \Lambda_{2t}(\alpha^{2t})^{-n+2} + \Lambda_{2t-1}(\alpha^{2t-1})^{-n+2} + \ldots + \Lambda_0(\alpha^0)^{-n+2}\end{aligned}$$

....

$$\begin{aligned}\Lambda(\alpha^{-0}) &= \Lambda_{2t}(\alpha^{-0})^{2t} + \Lambda_{2t-1}(\alpha^{-0})^{2t-1} + \ldots + \Lambda_0(\alpha^{-0})^0 \\ &= \Lambda_{2t}(\alpha^{2t})^{-0} + \Lambda_{2t-1}(\alpha^{2t-1})^{-0} + \ldots + \Lambda_0(\alpha^0)^{-0}\end{aligned}$$

즉, x의 각 i번째 계수별로 α^{2i}를 곱해 가면 된다.

여기서 n=255인 경우는 $\alpha^{-254}=\alpha$로 바로 시작이 가능하지만, shortened RS인 n<255의 경우에 첫 시작에서 $\alpha^{-(n-1)}$를 $\Lambda(x)$에 대입했을 때 각 차수의 값을 구해야 한다.

이를 구현하기 위해서 2가지의 방법을 생각할 수 있다. 먼저 shorten RS에서 삭제된 s번도 iteration에 포함하여 $\alpha^{-(n-1)}$값을 Chien search 내부에서 찾는 방법이다. 이 경우 Chien search만으로 간단히 구현이 가능해지는 장점이 있는 반면, shortened RS에서 n값이 작아진 경우에도 무조건 255번 iteration을 해야 하므로 전체 latency가 증가한다. 즉, 매번 $\alpha^{-(n-1)}$을 새로 계산해야 한다. 그리고 pipeline 구조이므로 1st stage인 syndrome도 역시 255 cycle을 소요해야 한다.

다음으로 본 논문에서 설계한 방식으로 처음으로 바뀐 n값이 들어올 때 미리 s번 iteration하여 $\alpha^{-(n-1)}$의 경우를 계산해두는 방식이다. 하드웨어 구조는 다음과 같다.

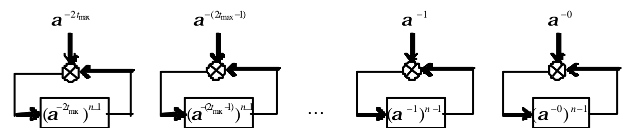

그림 7. $(\alpha^{-i})^{n-1}$ 구조

그리고 역시 어플리케이션의 n, k값이 고정되어 있는 경우는 미리 위의 값들을 모두 계산해서 ROM에 저장해두는 방법도 생각할 수 있다.

위의 방식을 적용할 수 있도록 수정된 Chien search의 구조는 다음과 같다.

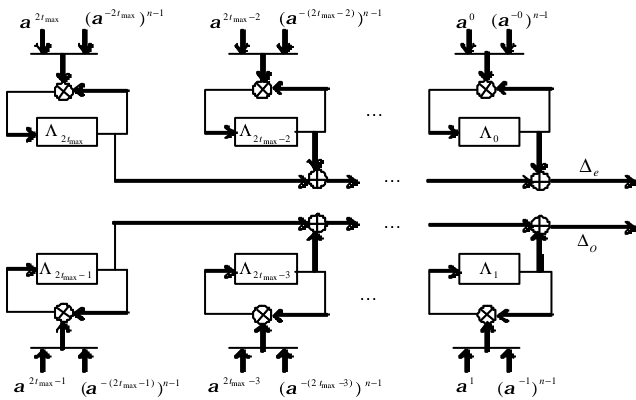

그림 8. Chien search 구조

Error evaluation의 Forney 알고리즘을 위해서 even, odd를 따로 계산한다. 그리고 위의 구조에서 모든 cell들이 맨 첫 cycle에만 α^{255-n}값을 선택하도록 control한다.

(3.2) Error Evaluation (Forney Algorithm)

Forney algorithm를 이용하면 error의 값을 예측할 수 있다.

$$e_l = \Omega(\alpha^{-l})\Delta_o^{-1}$$

하드웨어의 구조는 위의 error locator polynomial과 같은 방법으로 구할 수가 있다.

(3.3) Error Correction

Error correction은 원래의 입력 신호에 error값을 더해서 symbol의 error를 제거할 수 있다. Chien search로 error가 확인된 경우 원래의 input data에 Forney algorithm의 결과를 더하여 최종 corrected data를 얻을 수 있다.

따라서 위의 error evaluation와 error correction의 하드웨어를 구성하면 다음과 같다.

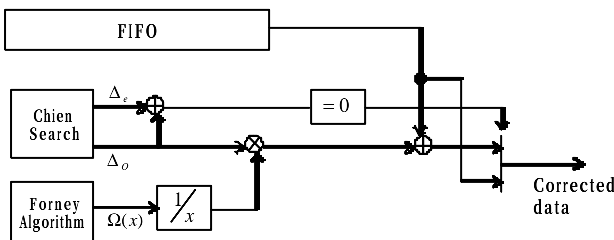

그림 9. Forney 알고리즘 & Error Correction 구조

Error evaluator polynomial의 inversion은 ROM을 이용하여 구한다.

이제 위에서 설계한 decoder의 전체 동작 타이밍을 정리하면 다음과 같다.

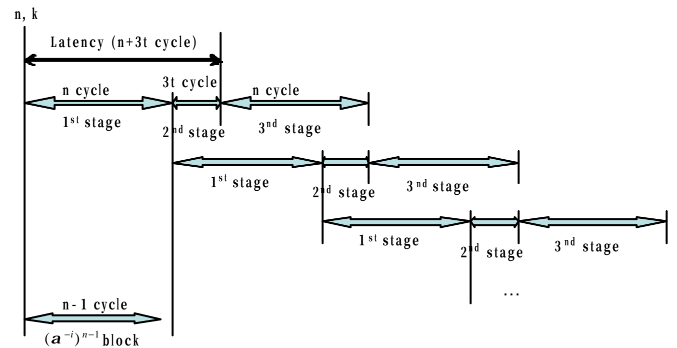

그림 10. RS decoder의 전체 동작 타이밍

control block은 n, k값을 받아서 위의 타이밍으로 각 block들이 동작할 수 있도록 control 신호를 보낸다.

2.3 동작 검증

설계한 RS 코더의 동작을 검증하기 위해서 Verilog로 RS 코더를 구현하고 아래와 같이 PLI를 이용하여 동작 검증 및 결과 분석을 위한 환경을 구성하였다.

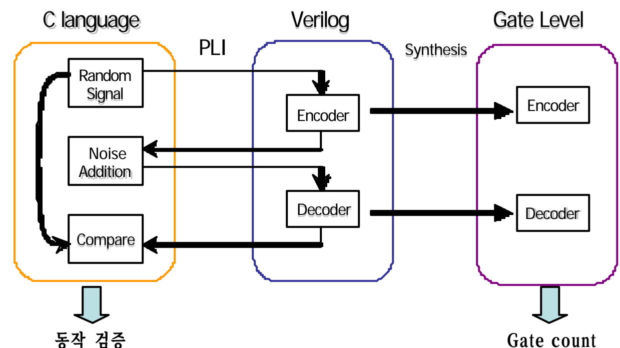

그림 11. RS 코더 검증 환경

그리고, DVD의 (182, 172), (208, 192) 두 가지 경우에 대해 encoder, decoder의 하드웨어의 동작을 검증하였다.

2.4 합성 결과

위의 DVD (182, 172), (208, 192)를 동시에 지원하는 하드웨어의 합성 결과이다. Synopsys 툴을 이용하여 하이닉스 0.35um 공정의 standard cell library를 이용하여 합성하였다.

2.4.1 Encoder 합성 결과

3가지 경우에 대해서 회로를 구현하여 gate count를 측정하였다. (1)번의 경우 (208, 192)로 고정된 경우를 구현한 것이다. 그리고 (2)번은 ROM으로 (182, 172), (208, 192) 두 가지를 지원하게 만든 것이며, 마지막 (3)번은 generator polynomial coefficient generator를 사용하여 $t_{\max}=8$로 $t \le 8$을 모두 지원하도록 만든 것이다.

	Fixed (1)	ROM (2)	GPC (3)
Gate-count	3,284	3,641	6,638

표 5. Encoder의 gate-count 결과

ROM을 이용한 경우에는 전체 gate count가 10%, generator polynomial coefficient generator를 이용한 경우에는 2배 정도로 gate count가 증가했음을 알 수 있다.

2.4.2 Decoder 합성 결과

FIFO에는 single-port SRAM을 이용하여 bank별로 나누어서 read, write가 동시에 가능하도록 하였다. SRAM의 전체 크기는 $n_{\max}+3t_{\max}$Bytes 이상이면 된다.

역시 다음 3개의 경우에 대해서 측정하였다. (1)번은 (208, 192)로 고정된 경우이고, (2)번은 ROM으로 (182, 172), (208, 192) 두 가지를 지원하는 경우, (3)번은 $n_{\max}=208$, $t_{\max}=8$로 $n \le 208$, $t \le 8$인 모든 경우를 지원하는 회로이다.

		Fixed (1)	ROM (2)	Multi (3)
Gate-count	Non-memory	11,597	12,741	14,923
	Memory	34,940	34,940	34,940
	Total	46,537	47,681	49,863

표 6. Decoder의 gate-count 결과

(2)의 경우는 전체적으로 3%, (3)의 경우 약 10%정도의 면적 증가가 있음을 알 수 있다.

그리고 단순히 pipeline의 타이밍만을 조절하므로 동작속도는 거의 변하지 않았다. 최소 clock time이 4.53ns로 약 200MHz까지 동작 가능하다. 따라서 data processing rate는 200Mbytes/s가 된다. DVD symbol rate가 최대 4Mbytes/s이므로 50배속 DVD까지 충분히 만족시킬 수 있다. [4]

V. 결론

본 논문에서는 RF(256)상의 여러 RS코드를 지원할 수 있는 RS 코드를 설계하고 DVD 어플리케이션의 경우에 동작을 직접 검증하였다.

RS encoder의 경우 generator polynomial의 계수를 정의로부터 직접 얻는 방법과 ROM을 이용하는 방법으로 구현하였다. RS decoder의 경우 고정된 회로에서 면적을 크게 늘리지 않으면서 전체 latency가 n+3t인 회로를 설계하였다.

Reference

[1] I. S. Reed, G. Solomon, "Polynomial Codes over Certain Finite Fields", The Society for Industrial and Applied Mathematics, 1960

[2] Stephen B. Wicker, Vijay K. Bhargava, "Reed-Solomon Codes and Their Applications", IEEE Press, 1994

[3] I. S. Reed, M. T. Shih, T. K. Truong, "VLSI Design of Inverse-free Berlekamp-Massey Algorithm", Proc. Inst. Elect. Eng., 1991

[4] Hsie-Chia Chang, C. Bernard Shung, Chen-Yi Lee, "A Reed-Solomon Product-Code Decoder Chip for DVD Applications", IEEE Journal of Solid-State Circuits, 2001

[5] E. Berlekamp, "Algebraic Coding Theory", McGrawHill, 1968