VHDL로 구현된 직렬승산 리드솔로몬 부호화기의 복잡도 분석

학생회원 백 승 훈*, 종신회원 송 익 호**, 배 진 수*

Complexity Analysis of a VHDL Implementation of the Bit-Serial Reed-Solomon Encoder

Seung hun Back* ***Student Member*** Iick ho Song**, Jin soo Bae* ***Regular Members***

요 약

리드솔로몬 부호화기를 구현하기 위해서 제안된 구조는 널리 알려진 대로 일반적인 구조와 직렬승산기를 쓰는 구조가 있다. 일반적 구조의 부호화기는 구조가 복잡한 대신 처리속도가 빠르고, 반면에 직렬승산기를 쓰는 부호화기는 구조는 단순하지만 처리속도는 그다지 빠르지 않은 것으로 알려져 있다. 이 논문에서는, 이 널리 알려진 사실이 VHDL로 구현할 때는 사실이 아닐 수도 있다는 것을 보인다. 이는, 직렬승산기에 필요한 쌍대기저 변환테이블을 구현하는 데에는 많은 게이트가 필요한 경우가 있기 때문인 것으로 해석된다. 한편 두 가지 구조를 써서 VHDL로 구현한 부호화의 처리속도는 모두 같다.

Key Words : Reed-Solomon code, VHDL, bit-serial, Berlekamp.

ABSTRACT

Reed-Solomon code is one of the most versatile channel codes. The encoder can be implemented with two famous structures: ordinary and bit-serial. The ordinary encoder is generally known to be complex and fast, while the bit-serial encoder is simple and not so fast. However, it may not be true for a longer codeword length at least in VHDL implementation. In this letter, it is shown that, when the encoder is implemented with VHDL, the number of logic gates of the bit-serial encoder might be larger than that of the ordinary encoder if the dual basis conversion table has to be used. It is also shown that the encoding speeds of the two VHDL implemented encoders are exactly same.

I. 서 론

리드솔로몬 (RS) 부호는 통신 및 저장시스템에 널리 사용된다[1]. RS 인코딩 동작 중에서 심벌 곱셈은 가장 복잡하고 시간이 많이 드는 과정이다[2]. 쌍대기저 변환을 응용한 곱셈 알고리즘을 채용한 직렬승산 RS 부호화기는 벌리캠프에 의해 제안되었다[3].

이 논문에서 RS 부호화기를 VHDL로 구현하여 성능과 복잡도 분석을 수행하기 위한 성능지표로서 각각 부호어 하나를 부호화하는데 필요한 클럭수와 VHDL로 부호화기를 구현하는데 필요한 논리 게이트 수를 사용한다. 일반적으로 알려져 있기로는, 직렬승산 부호화기는 일반적인 부호화기에 견주어볼 때, 구조가 간단하기 때문에 일반적인 부호화기보다

* 세종대학교 정보통신공학과 신호처리연구실 (baej@sejong.ac.kr),
** 한국과학기술원 전자전산학과 전기및전자공학전공 (isong@sejong.kaist.ac.kr)
논문번호 : KICS2005-02-071, 접수일자 : 2005년 2월 18일
※ 본 연구는 한국과학재단 특정기초연구 R01-2004-000-10019-0 지원으로 수행되었음.

더 적은 수의 논리게이트로 구현할 수 있는 반면, 데이터를 직렬로 처리하기 때문에 부호화 시간이 길다고 한다. 그러나 이 두 종류의 부호화기를 VHDL로 구현할 때 필요한 논리 게이트 수를 견주어 보면 어떤 경우에는 직렬승산 부호화기가 더 복잡할 수도 있다는 것을 밝힐 수 있다. 따라서 복잡도가 중요한 요소가 되는 응용 분야에서 RS 부호화기를 VHDL로 구현하고자 할 때 더욱 특별한 주의를 기울여야 한다.

Ⅱ. 직렬승산 부호화 알고리즘 : 쌍대기저 곱셈

일반적인 부호화기의 구성은 잘 알려져 있으므로 이 논문에서 따로 설명하지는 않는다 [1,4,6-8]. 대신 직렬승산 부호화기에 쓰이는 쌍대기저 곱셈 알고리즘을 간단히 소개한다. 갈루아체 GF(p^m)의 기저는 서로 선형 독립인 m개의 GF(p^m)의 원소들로 이루어진 집합을 일컫는다. 갈루아체 GF(p^m) 의 기저는 m 개의 선형독립인 원소들로 이루어져 있는데, 기저는 유일하게 존재하는 것은 아니어서 하나의 갈루아체는 복수개의 기저를 가질 수 있다. 또, 하나의 기저에 대해서 언제나 쌍대기저를 구할 수 있다. α가 GF(p^m)의 원시 다항식의 한 근일 때, $\{\alpha^0, \alpha^1, \cdots, \alpha^{m-1}\}$를 표준기저라 하고 다음 식을 만족하는 $\{\lambda_0, \lambda_1, \cdots \lambda_{m-1}\}$을 그 표준기저의 쌍대기저라한다.

$$\mathrm{Tr}(\alpha^j \lambda_k) = \begin{cases} 1, & j=k \\ 0, & j \neq k \end{cases} \quad (1)$$

여기서 대각합 Tr(·)은 다음 식과 같다.

$$\begin{aligned} \mathrm{Tr}(\beta) &= \beta + \beta^p + \cdots + \beta^{p^{m-1}} \\ &= \sum_{k=0}^{m-1} \beta^{p^k} \end{aligned} \quad (2)$$

GF(p^m)의 모든 원소는 표준기저의 선형조합으로도, 쌍대기저의 선형조합으로도 나타낼 수 있으며, 서로 변환도 가능하다. 쌍대기저 곱셈의 이점 가운데 하나는 덧셈기만을 이용해서 곱셈기를 구성할 수 있다는 것이다. 일반적으로 덧셈기보다 곱셈기가 훨씬 더 복잡하므로, 쌍대기저 곱셈을 이용하면, 상대적으로 간단하게 곱셈기를 구성할 수 있게 된다. 따라서 쌍대기저 곱셈에 바탕을 둔 직렬승산 RS 부호화기는 덧셈기만으로 구성할 수 있다 [3].

이 절에서는 쌍대기저 곱셈을 먼저 설명하고, 일반적인 부호화기가 주어졌을 때 그로부터 직렬승산 부호화기의 내부 계수들을 얻는 방법을 설명한다. GF(p^m)의 두 원소인 β와 γ의 곱을 σ라고 하자. β와 γ는 다음 식과 같이 각각 표준기저와 쌍대기저의 선형조합으로 나타낼 수 있다.

$$\beta = \sum_{i=0}^{m-1} b_i \alpha^i, \quad \gamma = \sum_{k=0}^{m-1} c_k \lambda_k \quad (3)$$

곱 σ는 쌍대기저 $\{\lambda_k\}$의 선형조합으로써 표현할 수 있으므로, 다음 식에서 선형조합의 계수 s_k만 구하면 σ를 얻을 수 있다.

$$\begin{aligned} \sigma = \beta \cdot \gamma &= \sum_{i=0}^{m-1} \sum_{k=0}^{m-1} b_i c_k \alpha^i \lambda_k \\ &= \sum_{k=0}^{m-1} s_k \lambda_k \end{aligned} \quad (4)$$

곱셈계수 s_k는 대각합 Tr(·)을 이용해서 얻을 수 있다 [6].

$$\begin{aligned} s_k &= \mathrm{Tr}(\sigma \cdot \alpha^k) = \mathrm{Tr}(\beta \cdot \gamma \cdot \alpha^k) \\ &= \mathrm{Tr}(\beta \cdot (\gamma \cdot \alpha^k)). \end{aligned} \quad (5)$$

이를 이용해서, $(n, k) = (31, 27)$인 일반적인 RS 부호화기로부터 RS 직렬승산 부호화기를 얻어보기로 하자. 같은 방법으로 어떤 (n, k)의 값을 갖는 직렬승산 RS 부호화기도 얻을 수 있다. 그림 1, 2는 각각 일반적인 부호화기와 직렬승산 부호화기의 블록도이다. 그림 1의 계수 g_i는 문헌에서 어렵지 않게 찾아볼 수 있다. (31,27)의 경우에는 $g_0 = \alpha^{10}$, $g_1 = \alpha^{29}$, $g_2 = \alpha^{19}$, $g_3 = \alpha^{24}$이다 [7,8]. 직렬승산 부호화기의 s_i는 다음과 같이 z_i의 선형조합으로써 얻을 수 있다.

$$s_0 = \mathrm{Tr}(Z \cdot g_0) = \mathrm{Tr}(Z \cdot \alpha^{10}) = z_0 + z_4 \quad (6)$$

$$s_1 = \mathrm{Tr}(Z \cdot g_1) = \mathrm{Tr}(Z \cdot \alpha^{29}) = z_0 + z_3 \quad (7)$$

$$s_2 = \mathrm{Tr}(Z \cdot g_2) = \mathrm{Tr}(Z \cdot \alpha^{19}) = z_1 + z_2 \quad (8)$$

$$\begin{aligned} s_3 &= \mathrm{Tr}(Z \cdot g_3) = \mathrm{Tr}(Z \cdot \alpha^{24}) \\ &= z_1 + z_2 + z_3 + z_4 \end{aligned} \quad (9)$$

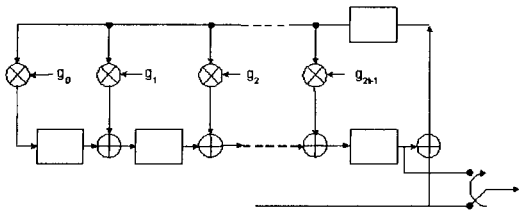

그림 1. (32,27) RS 일반적인 부호화기의 블록 다이어그램

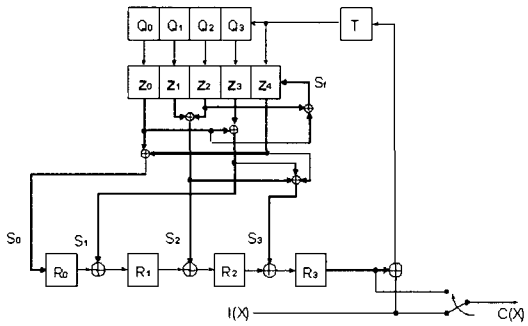

그림 2. (32,27) RS 직렬승산 부호화기의 블록 다이어그램

그림 2에서 Z는 이동 레지스터에 저장된 심벌 수열이다. 그림 1의 곱셈기가 비트 단위 곱셈기라면, 그림 2의 덧셈기는 심벌 단위 덧셈기이다. 곧, 직렬승산 부호화기는 비트 단위 곱셈기 대신 심벌 단위 덧셈기만을 요구하므로 구조가 매우 간단하다고 할 수 있다.

Ⅲ. 구현과 분석

이 절에서는 VHDL로 구현된 두 개의 부호화기들에 소요된 논리 게이트수와 하나의 부호어를 부호화하기 위해 필요한 클럭 수의 변화를 부호어 길이 n에 따라 비교한다. 부호어의 길이는 15, 31, 63, 55, 127, 255의 경우를 생각하고 공정한 비교를 위해 모든 선택된 부호들이 유사한 부호율 k/n 를 갖도록 메시지 심벌의 길이 k를 결정한다. 대체로 같은 부호율을 갖는 부호들은 비슷한 오류정정능력 갖는 것으로 추정된다. 즉, k/n는 일정하게 유지하면서 n을 변화시켜가며 논리 게이트수와 클럭수를 비교한다. 표 1은 선택된 부호 각각의 부호율, 원시 다항식 등을 보여준다. 여기서 t는 오류정정능력이다.

VHDL로 구현된 RS 부호화기의 블럭도는 그림 3에 나와있다. *makedata*와 *encoding* 두 개의 함수 블럭으로 구성되어 있다. *makedata* 블럭은 블럭으로 임의의 심벌 수열을 입력으로 제공하고 *encoding* 블럭은 그에 맞는 패리티 심벌 수열을 생성한다.

표 1. 선택된 부호들의 원시다항식과 부호율

m	(n, k)	t	원시다항식	k/n
4	(15,13)	1	X^4+X+1	0.867
5	(31,27)	2	X^5+X^2+1	0.871
6	(63,55)	4	X^6+X+1	0.873
7	(127,111)	8	X^7+X^3+1	0.874
8	(255,223)	16	$X^8+X^4+X^3+X^2+1$	0.875

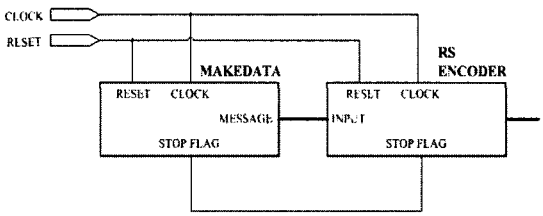

그림 3. (31,27) RS 부호화기의 VHDL 블록도

encoding 블럭은 직렬승산 부호화기의 경우에는 (심벌 레벨) 덧셈기로, 일반적인 부호화기의 경우에는 (비트 레벨) 곱셈기로 이루어져 있다. 주목할만한 것은 기저변환이 직렬승산 부호화기에서는 꼭 필요하므로 기저변환을 위해서는 미리 저장된 변환테이블이 항상 필요하다는 것이다. 직렬승산 부호화기에서 곱셈은 다음과 같이 수행된다. 표준기저로 표현된 입력 심벌 수열은 쌍대기저의 선형조합의 꼴로 변환된다. 이 때 심벌 레벨 덧셈기에서 곱셈이 수행된다. 마지막으로 출력은 쌍대기저 형태이므로 표준기저 형태로 변환해야만 한다. 변환된 출력은 이동 레지스터에 저장되고 이 과정을 반복하면 패리티 심벌 수열이 생성된다. 부호화기는 VHDL을 이용하여 알테라 FPGA에서 구현하였다.

그림 4는 변환테이블을 위한 논리게이트의 숫자가 n 이 커질수록 급속도로 증가하여 n 이 커질수록 직렬승산 부호화기가 일반적인 부호화기보다 더 많은 논리게이트를 필요로 하는 것을 보여준다.(여기서, 변환테이블을 사용하지 않은 부호화기의 논리게이트 수는 추정치임.) 주목할만한 점은 변환테이블을 제외한 직렬승산 부호화기의 논리게이트 수는 n 에 상관없이 일반적인 부호화기보다 훨씬 적다는 것이다. 다시 말해, VHDL로 구현한 직렬승산 부호화기는 일반적인 부호화기보다 더 많은 논리 게이트를 필요로 한다. 이것은 일견 직렬승산 부호화기가 덧셈기들로만 구성되어 더 단순할 것이라는 사실에 모순이 있는 것처럼 보인다. 하지만 이 모순의 원인은 n

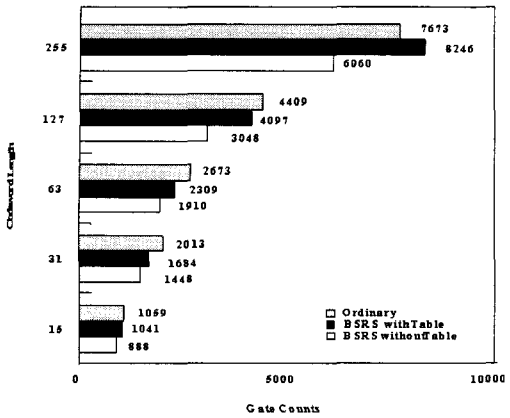

그림 4. 일반적인 RS 부호화기와 직렬승산 RS 부호화기의 게이트수 비교

이 증가할수록 변환테이블의 크기가 매우 급격히 증가하고, 결과적으로 변환테이블로 인한 논리게이트수의 증가가 간단한 알고리즘에 의한 게이트 수의 감소의 효과를 넘어서기 때문인 것이다. 쌍대기저 곱셈 알고리즘 자체가 단순한 알고리즘임을 부정하는 것은 아닌 것이다. 이는 단지 VHDL로 구현할 때에는 복잡도 문제가 변환테이블에 의해 생길수도 있다는 것을 의미할 뿐이다.

그림 5, 6을 보면 하나의 부호어를 부호화하기 위해 걸린 시간이 두 부호화기 모두 같다는 것을 알 수 있다. 표 2는 하나의 부호어를 부호화하기 위해 요구되는 클럭수가 n 이라는 것을 보여주는데, 이는 부호화기들이 리얼타임 부호화에 쓰일수 있다는 것을 의미한다.

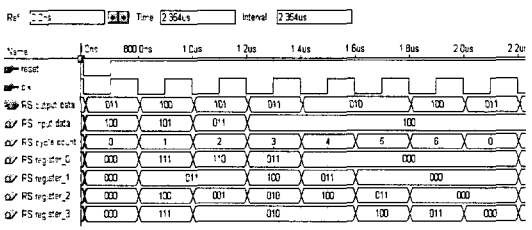

그림 5. 일반적인 (7,3) RS 부호화기의 구현결과

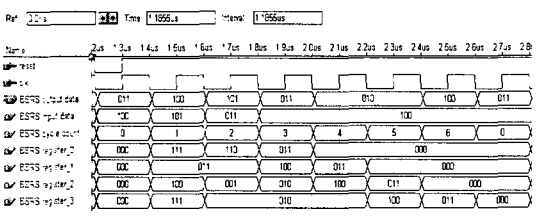

그림 6. 직렬승산 (7,3) RS 부호화기의 구현결과

표 2. 클럭 사이클 비교

	클럭사이클				
(n, k)	(15,13)	(31,27)	(63,55)	(127,111)	(255,223)
Ordinary	15	31	63	127	255
Bit-serial	15	31	63	127	255

Ⅳ. 맺음말

이 논문에서는 쌍대기저 변환 테이블이 VHDL로 구현된 RS 직렬승산 부호화기의 논리게이트 수에 미치는 영향을 정량적으로 분석하였다. 변환 테이블에 필요한 논리게이트 수는 부호어의 길이 n이 커질수록 매우 급격하게 증가하기 때문에 VHDL에서 구현되는 직렬승산 부호화기는 n이 매우 커질 때 일반적인 부호화기보다 훨씬 복잡해진다. 이는 직렬승산 부호화기가 간단하다고 알려져 있는 일반적인 사실과 일치하지 않는다. 쌍대기저 곱셈이 매우 간단한 알고리즘임에는 틀림이 없지만, 쌍대기저의 변환이 필수적이라는 사실에 주목할 필요가 있고, 이 쌍대기저 변환이 VHDL 구현에서는 상당히 많은 자원을 필요로 한다는 사실을 기억할 필요가 있다. 단, 변환 테이블을 참조할 필요가 없는 기저 변환을 사용한다면 직렬승산 부호화기가 부호어 길이 n에 상관없이 일반적인 부호화기보다 항상 더 적은 수의 논리게이트로 구현될 수 있다. 따라서, 변환 테이블 없는 기저 변환은 연구해 볼만하며 여지껏 연구결과가 발표되지않은 분야이다. 한편, 두 부호화기들의 부호화 속도는 같다는 것을 보였다.

참 고 문 헌

[1] I.S. Reed and G. Solomon, "Polynomial codes over certain finite field", J. Soc. Ind. Applied Math., Vol.8, pp. 300-304, 1960.

[2] I.S. Hus, T.K. Truong, L.J. Deutsch, and I.S. Reed, "A comparison of VLSI architecture of finite field multipliers using dual, normal, or standard bases", IEEE Trans. Comput., vol. 37, no. 6, pp. 735-739, Jun. 1988.

[3] E. Berlekamp, "Bit-serial Reed-Solomon encoders", IEEE Trans. Inf. Theory, vol. 28, no. 6, pp. 869-874, Nov. 1982.

[4] 이만영, BCH 부호와 Reed-Solomon 부호, 민

음사, 1990.
[5] R. Lidl, Finite Field, Addison Wesley, 1983.
[6] R.E. Blahut, Algebraic Codes for Data Transmission, Cambridge University Press, 2003.
[7] S. Bernard, Digital Communications, Prentice-Hall, 2001.
[8] S. Lin and D.J. Costello, Error Control Coding, 2nd Ed., Prentice-Hall, 2002.

백 승 훈(Seung hun Back) 학생회원

2001년 9월 세종대학교 입학
2001년 9월~현재 세종대학교 정보통신공학과 (2005년 8월 졸업예정)
<관심분야> 디지털신호처리, 채널코딩, 신호검파이론

송 익 호(Iick ho Song) 종신회원
1982년 2월 서울대학교 전자공학과 공학사(준최우등)
1984년 2월 서울대학교 전자공학과 공학석사
1985년 8월 펜실베니아대학교 전기공학과 공학석사
1987년 3월~1998년 2월 벨 통신연구소 연구원
1988년 3월~현재 한국과학기술원 전자전산학과 조교수, 부교수, 교수
1995년 1월~현재 한국통신학회 논문지 편집위원
1991년 11월, 1996년 11월 한국통신학회 학술상받음
1993년 11월 한국음향학회 우수연구상 받음
1998년 11월 한국통신학회 LG학술상 받음
1999년 11월 대한 전자공학회 해동논문상 받음
2000년 3월 젊은 과학자상 받음
2000년 11월 한국통신학회 모토롤라학술상 받음 대한전자공학회, 한국음향학회, 한국통신학회 종신회원; IEE 석학회원; IEEE 준석학회원
<관심분야> 통계학적 신호처리와 통신이론, 신호검파와 추정, 이동통신

배 진 수(Jin soo Bae) 종신회원
1990년 2월 경기과학고등학교 조기졸업 (우등)
1993년 2월 한국과학기술원 전기및전자공학과 공학사 (조기졸업, 최우등)
1995년 2월 한국과학기술원 전기및전자공학과 공학석사
1998년 2월 한국과학기술원 전기및전자공학과 공학박사
1997년 1월~1997년 12월 동경대학 객원연구원
1998년 1월~1998년 10월 엑센츄어 컨설턴트
1998년 11월~1999년 12월 일본모토로라 연구원
2000년 3월~현재: 세종대학교 정보통신공학과 전임강사/조교수, 한국통신학회 종신회원; IEEE 준석학회원
<관심분야> 신호처리이론, 신호검파이론