(19)대한민국특허청(KR)
(12) 공개특허공보(A)

(51) 。Int. Cl.
H03M 13/15 (2006.01)
H03M 13/00 (2006.01)

(11) 공개번호 10-2006-0125837
(43) 공개일자 2006년12월06일

(21) 출원번호 10-2006-7013905
(22) 출원일자 2006년07월11일
심사청구일자 없음
번역문 제출일자 2006년07월11일
(86) 국제출원번호 PCT/US2004/038272
국제출원일자 2004년11월16일
(87) 국제공개번호 WO 2005/062472
국제공개일자 2005년07월07일

(30) 우선권주장 10/735,423 2003년12월12일 미국(US)

(71) 출원인 아나로그 디바이시즈 인코포레이티드
미국 매사추세츠 노우드 쓰리 테크놀로지 웨이 (우:02062)

(72) 발명자 리버롤, 보리스
미국 01801 매사추세츠주 워번 아파트먼트 402 킴볼 코트 6
리, 윤, 영
미국 01810 매사추세츠주 앤도버 오리올레 드라이브 46

(74) 대리인 주성민
백만기
이중희

전체 청구항 수 : 총 24 항

(54) 데이터를 인코딩하기 위한 방법 및 장치

(57) 요약

본 발명은 데이터를 인코딩하기 위한 방법 및 장치를 제공한다. 본 방법은 (a) 코드 생성자 계수들을 곱한 제1 성분 값들의 제1 테이블, 및 상기 코드 생성자 계수들을 곱한 제2 성분 값들의 제2 테이블을 제공하는 단계, (b) 메시지 심볼 및 최상위의 나머지 심볼에 기초하여 갈루아 필드 요소(Galois field element)를 결정하는 단계, (c) 상기 갈루아 필드 요소를 제1 및 제2 성분으로 분리시키는 단계, (d) 상기 제1 성분 및 생성자 계수 인덱스를 이용하여 상기 제1 테이블 내의 하나 이상의 제1 테이블 값에 액세스하고, 상기 제2 성분 및 상기 생성자 계수 인덱스를 이용하여 상기 제2 테이블 내의 하나 이상의 제2 테이블 값에 액세스하는 단계, (e) 동일한 생성자 계수 인덱스들을 갖는 제1 및 제2 테이블 값과, 이전의 나머지 심볼에 기초하여 현재의 나머지 심볼을 결정하는 단계, (f) 상기 코드 생성자의 각각의 생성자 계수에 대해 단계 (e)를 수행하여 나머지 심볼들의 그룹을 제공하는 단계, 및 (g) 코드워드(codeword) 내의 각각의 메시지 심볼에 대해 단계 (b) 내지 (f)를 수행하여, 코드워드의 메시지 심볼들과 함께 전송될 검사 심볼들을 구성하는 최종 나머지 심볼들의 그룹을 제공하는 단계를 포함한다.

대표도

도 6

특허청구의 범위

청구항 1.

데이터를 인코딩하기 위한 방법으로서,

(a) 코드 생성자 계수들을 곱한 제1 성분 값들의 제1 테이블, 및 상기 코드 생성자 계수들을 곱한 제2 성분 값들의 제2 테이블을 제공하는 단계,

(b) 메시지 심볼 및 최상위의 나머지 심볼(high order remainder symbol)에 기초하여 갈루아 필드 요소(Galois field element)를 결정하는 단계,

(c) 상기 갈루아 필드 요소를 제1 및 제2 성분으로 분리시키는 단계,

(d) 상기 제1 성분 및 생성자 계수 인덱스를 이용하여 상기 제1 테이블 내의 하나 이상의 제1 테이블 값에 액세스하고, 상기 제2 성분 및 상기 생성자 계수 인덱스를 이용하여 상기 제2 테이블 내의 하나 이상의 제2 테이블 값에 액세스하는 단계,

(e) 동일한 생성자 계수 인덱스들을 갖는 제1 및 제2 테이블 값과, 이전의 나머지 심볼에 기초하여 현재의 나머지 심볼을 결정하는 단계,

(f) 상기 코드 생성자의 각각의 생성자 계수에 대해 상기 단계 (e)를 수행하여 나머지 심볼들의 그룹을 제공하는 단계, 및

(g) 코드워드(codeword) 내의 각각의 메시지 심볼에 대해 상기 단계 (b) 내지 (f)를 수행하여, 상기 코드워드의 메시지 심볼들과 함께 전송될 검사 심볼들을 구성하는 최종 나머지 심볼들의 그룹을 제공하는 단계

를 포함하는 데이터 인코딩 방법.

청구항 2.

제1항에 있어서,

상기 단계 (b)는 상기 메시지 심볼 및 상기 최상위의 나머지 심볼에 대한 XOR(exclusive OR) 연산을 포함하는 방법.

청구항 3.

제2항에 있어서,

상기 단계 (e)는 중간 값을 제공하는 상기 제1 및 제2 테이블 값에 대한 XOR 연산, 및 상기 현재의 나머지 심볼을 제공하는 상기 중간 값 및 상기 이전의 나머지 심볼에 대한 XOR 연산을 포함하는 방법.

청구항 4.

제1항에 있어서,

상기 제1 및 제2 성분은 각각 상기 갈루아 필드 요소의 높은 해프워드(halfword)와 낮은 해프워드를 포함하는 방법.

청구항 5.

제1항에 있어서,

상기 코드워드의 상기 메시지 심볼은 메시지 바이트들을 포함하는 방법.

청구항 6.

제5항에 있어서,

상기 단계 (g)는 상기 코드워드 내의 8개까지의 에러를 정정하기 위한 16개의 검사 심볼을 생성하는 방법.

청구항 7.

제1항에 있어서,

상기 단계 (d)는 제1 및 제2 테이블 값들의 둘 이상의 집합에 병렬로 액세스하는 단계를 포함하고, 상기 단계 (e)는 둘 이상의 현재의 나머지 심볼들을 병렬로 결정하는 단계를 포함하는 방법.

청구항 8.

제1항에 있어서,

상기 갈루아 필드 요소를 상기 제1 및 제2 성분들로 분리시키는 단계는 상기 갈루아 필드 요소를 제1 및 제2 마스크와 AND연산하는 단계를 포함하는 방법.

청구항 9.

제1항에 있어서,

상기 단계 (d)는 제1 및 제2 테이블 값들의 w개의 집합에 병렬로 액세스하는 단계를 포함하고, 상기 w는 상기 갈루아 필드 요소의 단위로 인코딩을 수행하는 프로세서의 워드 폭을 나타내고, 상기 단계 (e)는 w개의 현재의 나머지 심볼을 병렬로 결정하는 단계를 포함하는 방법.

청구항 10.

제1항에 있어서,

상기 단계 (a)는 연산 동안, 규정된 코드 매개변수에 대한 상기 제1 및 제2 테이블 내의 상기 테이블 값을 생성하는 단계를 포함하는 방법.

청구항 11.

제1항에 있어서,

상기 단계 (g)는 상기 생성자 계수의 개수와 동일한 개수의 검사 심볼을 제공하는 단계를 포함하는 방법.

청구항 12.

제1항에 있어서,

상기 단계 (g)는 상기 생성자 계수의 개수보다 적은 개수의 검사 심볼을 제공하는 단계를 포함하는 방법.

청구항 13.

데이터를 인코딩하기 위한 장치로서,

코드 생성자 계수들을 곱한 제1 성분 값들을 포함하는 제1 룩업 테이블,

상기 코드 생성자 계수들을 곱한 제2 성분 값들을 포함하는 제2 룩업 테이블, 및

메시지 심볼 및 최상위의 나머지 심볼에 기초하여 갈루아 필드 요소를 결정하기 위한 수단, 상기 갈루아 필드 요소를 제1 및 제2 성분으로 분리시키기 위한 수단, 상기 제1 성분 및 생성자 계수 인덱스를 이용하여 상기 제1 룩업 테이블 내의 하나 이상의 제1 테이블 값들에 액세스하기 위한 수단, 상기 제2 성분 및 상기 생성자 계수 인덱스를 이용하여 상기 제2 테이블 내의 하나 이상의 제2 테이블 값들에 액세스하기 위한 수단, 및 상기 코드 생성자의 각각의 생성자 계수에 대해, 동일한 생성자 계수 인덱스들을 갖는 제1 및 제2 테이블 값과, 이전의 나머지 심볼에 기초하여 현재의 나머지 심볼을 결정하여, 나머지 심볼들의 그룹을 제공하기 위한 수단을 포함하는 프로세서

를 포함하고,

코드워드의 각각의 메시지 심볼은 상기 코드워드의 상기 메시지 심볼들과 함께 전달될 검사 심볼들을 구성하는 최종 나머지 심볼들의 그룹을 제공하도록 처리되는 데이터 인코딩 장치.

청구항 14.

제13항에 있어서,

상기 갈루아 필드 요소를 결정하기 위한 수단은 상기 메시지 심볼 및 상기 최상위의 나머지 심볼에 대한 XOR 연산을 수행하기 위한 수단을 포함하는 장치.

청구항 15.

제13항에 있어서,

상기 현재의 나머지 심볼을 결정하기 위한 수단은 상기 제1 및 제2 테이블 값들에 대한 XOR 연산을 수행하여 중간 값을 제공하기 위한 수단, 및 상기 중간 값 및 상기 이전의 나머지 심볼에 대한 XOR 연산을 수행하여 상기 현재의 나머지 심볼을 제공하기 위한 수단을 포함하는 장치.

청구항 16.

제13항에 있어서,

상기 제1 및 제2 성분은 각각 상기 갈루아 필드 요소의 높은 해프워드와 낮은 해프워드를 포함하는 장치.

청구항 17.

제13항에 있어서,

상기 코드워드의 상기 메시지 심볼은 메시지 바이트들을 포함하는 장치.

청구항 18.

제13항에 있어서,

상기 프로세서는 상기 코드워드 내의 8개까지의 에러를 정정하기 위한 16개의 검사 심볼을 생성하도록 구성된 장치.

청구항 19.

제13항에 있어서,

상기 제1 및 제2 테이블 값들에 액세스하기 위한 수단은 제1 및 제2 테이블 값들의 둘 이상의 집합에 병렬로 액세스하기 위한 수단을 포함하고, 상기 현재의 나머지 심볼을 결정하기 위한 수단은 둘 이상의 현재의 나머지 심볼을 병렬로 결정하기 위한 수단을 포함하는 장치.

청구항 20.

제13항에 있어서,

상기 갈루아 필드 요소를 제1 및 제2 성분으로 분리시키기 위한 수단은 상기 갈루아 필드 요소를 제1 및 제2 마스크와 AND연산하기 위한 수단을 포함하는 장치.

청구항 21.

제13항에 있어서,

상기 제1 및 제2 테이블 값들에 액세스하기 위한 수단은 제1 및 제2 테이블 값들의 w개의 집합에 병렬로 액세스하기 위한 수단을 포함하고, 상기 w는 상기 갈루아 필드 요소의 단위로 상기 프로세서의 워드 폭을 나타내고, 상기 현재의 나머지 심볼을 결정하기 위한 수단은 w개의 현재의 나머지 심볼들을 병렬로 결정하기 위한 수단을 포함하는 장치.

청구항 22.

제13항에 있어서,

상기 프로세서는 연산 동안, 규정된 코드 매개변수에 대한 상기 제1 및 제2 룩업 테이블 내의 상기 테이블 값을 생성하기 위한 수단을 더 포함하는 장치.

청구항 23.

제13항에 있어서,

상기 코드워드의 각각의 메시지 심볼은 상기 생성자 계수의 개수와 동일한 개수의 검사 심볼을 제공하도록 처리되는 장치.

청구항 24.

제13항에 있어서,

상기 코드워드의 각각의 메시지 심볼은 상기 생성자 계수의 개수보다 더 적은 개수의 검사 심볼을 제공하도록 처리되는 장치.

명세서

기술분야

본 발명은 데이터 코딩에 관한 것으로서, 특히, 리드 솔로몬(Reed-Solomon) 코딩을 위한 방법 및 장치에 관한 것이다.

배경기술

에러 검출 및 정정을 위한 리드 솔로몬 코드(Reed-Solomon code)를 이용하여 정보를 전송 및 수신하기 위한 프로시저가 이하에 설명될 수 있다. 보편적으로 비트 스트림으로 나타나는 전송될 정보는 특정 크기 m의 작은 블럭들로 나뉘어 진다. 이러한 m 비트의 작은 블럭들은, 모든 가능한 2^m개의 m 비트 블럭들과, 2^m 차원의 갈루아 필드(Galois field, GF), 즉, GF(2^m)의 모든 2^m개의 요소들의 집합 사이에 일대일 맵핑이 있다는 점에서, 유한 필드(또는, 갈루아 필드)로서 알려진 수학적 구조의 요소로 식별될 수 있다. 갈루아 필드 자체는, 요소가 0과 1이고, 두개의 필드 연산, 즉, 덧셈과 곱셈이 덧셈은 m-튜플 차순의 다항식 계수들에 대한 XOR(exclusive OR) 연산으로서 수행되고, 곱셈은 필드 곱셈으로서 m차의 특정 "원시(primitive)" 이진 다항식을 법으로 하여 다항식 곱으로서 구현되도록 정의된 "기반(ground)" 이진 필드 GF(2)로부터의 계수를 갖는 차수<m인 모든 다항식들의 집합으로서 구성될 수 있는데, 앞의 곱셈에서 "원시" 이진 다항식은 약분될 수 없으며(더 낮은 차수의 이진 다항식들의 곱으로 더 인수분해될 수 없음), 또한, d<2^m-1에 대해 x^d-1 형태의 어떤 이진 다항식도 나누지 못한다.

가장 중요하고 흔한 경우들 중 하나는 m=8인 경우인데, 여기서, 그러한 블럭 각각은 바이트로서 나타나고, 임의의 종류의 컴퓨터에 의해 편리하게 핸들링될 수 있다. 따라서, 다음의 설명은, "바이트"가 m≠8인 2^m 차원의 갈루아 필드의 요소 또는 "심볼"일 수 있다는 조건의, 바이트가 GF(256)의 요소인 경우에 대한 것이다. GF(256) 구축을 허용하는 원시 다항식은 $P(x)=x^8+x^4+x^3+x^2+1$이다.

정보의 기본 단위(바이트 또는 심볼) 및 대응하는 갈루아 필드(GF)를 설정한 후에, 전송될 정보는 동일한 크기 k의 심볼들의 그룹들로 나뉘어지는데, k는 코드의 요구된 에러 정정 능력에 기초하여 구해진다. 그러한 그룹 각각은 "메시지"라고 불린다. 메시지는 또한, k-1차의 GF에 대한 다항식 M(x)로서 해석된다.

$$M(x)= m_0x^{k-1}+ m_1x^{k-2}+...+ m_{k-1}$$

여기서, 오름차순의 계수 m_i는 GF 요소로서 다루어진 원래의 비트 스트림으로부터 획득된 순서의 정보 심볼이다. t개의 에러를 정정할 수 있는 리드 솔로몬 코드를 형성하기 위해, 생성자 다항식으로서 알려진 2t차의 다항식 G(x)는 다음의 수학식에 따라 정의된다.

$$G(x)= (x-a^{2t-1}) (x-a^{2t-2})... (x-a^1) (x-a^0)= x^{2t}+ g_{2t-1} x^{2t-1}+...+ g_0$$

여기서 α는 두개의 중요한(그리고, 관련된) 속성을 갖는 GF 프로세싱의 소위 "원시(primitive)" 요소를 나타낸다. 우선, 그것은 방정식 p(α)=0을 만족시키고, 둘째로, 결과로서, GF의 모든 0이 아닌 요소는 원시 요소의 제곱으로서 표시되고 GF 곱셈의 연산 하에서 아벨(abelian)(또는, 가환) 그룹을 형성할 수 있다. 그것은 정확히, "안티 로그(anti-log)" 테이블을 이용하여 GF 요소의 곱셈을 수행하는 간편한 방법을 설정하는 "안티 로그" 테이블의 생성을 허용하는 속성이다.

x^{2t}로 곱해진 메시지 다항식 M(x)를 생성자 다항식 G(x)로 나눈 나머지 R(x)는 다음과 같다.

$$M(x)x^{2t} = Q(x)*G(x)+R(x)$$

나머지는 2t-1차의 다항식이며, 마지막으로, 이 다항식의 2t개의 계수는 조직적 형태로 "코드워드(codeword)"를 포함하는 k+ 2t개의 심볼들의 스트링을 형성하는 k개의 메시지 심볼들에 첨가되는 검사 심볼로서 사용된다. 코드워드의 길이는 GF 내의 0이 아닌 요소들의 개수, 즉, n-1을 초과해서는 안된다. 따라서, 실제로 통신 채널을 통해 전송된 메시지는 다항식으로서 나타날 수 있다.

$$M(x)*x^{2t}+R(x) = M(x)*x^{2t}+r_{2t-1}x^{2t-1}+\ldots+r_0$$

여기서, r_m은 검사 심볼, 즉, t개까지의 에러의 검출 및 정정을 허용하는 여분의 심볼이다.

따라서, 인코딩 태스크는 계수가 임의의 GF의 심볼인 다항식의 나누기를 수행하는 것으로 구성된다. GF에서의 계산을 다루는 통상의 소프트웨어 접근은 GF 심볼의 다항식 형태로부터 지수 형태로 및 역방향의 변환이 저장되어 있는 2개의 룩업 테이블을 이용하는 것에 기초한다. 이 방법은 (여기에서 주요한 GF(2^8)의 경우에 대해) 다음과 같이 요약될 수 있다.

1. 다항식 형태의 심볼들(GF의 요소들)은 바이트로서 나타난다.

$$b=(b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0)$$

2. 심볼들의 합은 비트의 XOR 연산, f+ g=f^g로서 구해진다.

3. 다항식 형태의 원시 요소 α는 α=(00000010)으로서 나타나는데, 이것은 다항식 표식에서 α(x)=x에 대응한다.

4. 각각의 0이 아닌 심볼 s는 특정 승수으로 제곱된 원시 요소 α이다 $s=\alpha^k$. 이진 다항식을 이용한 다항식 대수 처리를 이용하여, 개별적인 대수 테이블 log_table 및 지수 테이블 exp_table이 미리 계산되어, (0이 아닌) 요소들의 지수(승수) 및 다항식(바이트) 표시 간의 변환을 용이하게 하여,

$$\text{if } a^k=b, \text{ then } k=log_table[b],\ b=exp_table[k]$$

이 성립하도록 한다.

이들 테이블을 이용하여 GF 심볼들을 곱하기 위해, 피연산자 u와 v 및 결과 r이 바이트 형태라고 가정하고서 다음의 일련의 연산들이 수행되어야 한다.

$$GFMUL(u,v) = GFMUL(a^{log_table[u]}, a^{log_table[v]}) = a^{(log_table[u]\ +\ log_table[v])\%255}$$

$$r=GFMUL(u,v) = exp_table[(log_table\ [u] + log_table\ [v]\ )\ \%\ 255\ ]$$

여기서 %는 모듈로 연산을 표시한다(x%255는 x를 255로 나눈 나머지임).

이러한 접근에 기초한 GF에 대한 곱셈 결과의 계산 속도는, 테이블로의 복수의 액세스가 필요하고 덧셈에 의해 지수 테이블로의 엔트리의 주소가 계산되어야 한다는 사실에 의해 제한된다. 많은 경우에, 수정된 주소 레지스터를 통한 액세스는 2 내지 3의 대기 주기를 요구하여, 하나의 GF 곱셈의 총 시간은 심볼(바이트) 당 10 내지 12 클럭 주기일 수 있다. 이것은 리

드 솔로몬 코드의 소프트웨어 구현이 도전적인 문제가 되게 한다. 예를 들어, 188개의 데이터 바이트와 16개의 검사 바이트를 갖는 (204, 188, 16) 리드 솔로몬 코드의 일반적 경우에, 하나의 코드워드를 인코딩하기 위해서는 이하에 설명되는 바와 같이 188*16*(10 - 12)~=30 내지 36 K의 클럭이 필요하다. 따라서, 예를 들어, 4 K 심볼/초의 전송률을 이용한 ADSL 적용에 있어서, 인코더 연산만에 대한 MIPS의 총 소모는 120 내지 144 MIPS이며, 이것은 매우 높은 수치이다.

<본 발명의 요약>

본 발명의 일 실시예에 따르면, 데이터를 인코딩하기 위한 방법이 제공된다. 본 방법은 (a) 코드 생성자 계수들을 곱한 제1 성분 값들의 제1 테이블, 및 코드 생성자 계수들을 곱한 제2 성분 값들의 제2 테이블을 제공하는 단계, (b) 메시지 심볼 및 최상위의 나머지 심볼(high order remainder symbol)에 기초하여 갈루아 필드 요소를 결정하는 단계, (c) 갈루아 필드 요소를 제1 및 제2 성분으로 분리시키는 단계, (d) 제1 성분 및 생성자 계수 인덱스를 이용하여 제1 테이블 내의 하나 이상의 제1 테이블 값에 액세스하고, 제2 성분 및 생성자 계수 인덱스를 이용하여 제2 테이블 내의 하나 이상의 제2 테이블 값에 액세스하는 단계, (e) 동일한 생성자 계수 인덱스들을 갖는 제1 및 제2 테이블 값과, 이전의 나머지 심볼에 기초하여 현재의 나머지 심볼을 결정하는 단계, (f) 코드 생성자의 각각의 생성자 계수에 대해 단계 (e)를 수행하여 나머지 심볼들의 집합을 제공하는 단계, 및 (g) 코드워드 내의 각각의 메시지 심볼에 대해 단계 (b) 내지 (f)를 수행하여, 코드워드의 메시지 심볼들과 함께 전송될 검사 심볼들을 구성하는 최종 나머지 심볼들의 집합을 제공하는 단계를 포함한다.

본 발명의 제2 양태에 따르면, 데이터를 인코딩하기 위한 장치가 제공된다. 본 장치는 코드 생성자 계수들을 곱한 제1 성분 값들을 포함하는 제1 룩업 테이블, 코드 생성자 계수들을 곱한 제2 성분 값들을 포함하는 제2 룩업 테이블, 및 프로세서를 포함한다. 프로세서는 메시지 심볼 및 최상위의 나머지 심볼에 기초하여 갈루아 필드 요소를 결정하기 위한 수단, 갈루아 필드 요소를 제1 및 제2 성분으로 분리시키기 위한 수단, 제1 성분 및 생성자 계수 인덱스를 이용하여 제1 룩업 테이블 내의 하나 이상의 제1 테이블 값에 액세스하고, 제2 성분 및 생성자 계수 인덱스를 이용하여 제2 룩업 테이블 내의 하나 이상의 제2 테이블 값에 액세스하기 위한 수단, 및 코드 생성자의 각각의 생성자 계수에 대해, 동일한 생성자 계수 인덱스들을 갖는 제1 및 제2 테이블 값과, 이전의 나머지 심볼에 기초하여 현재의 나머지 심볼을 결정하여, 나머지 심볼들의 집합을 제공하기 위한 수단을 포함한다. 코드워드의 각각의 메시지 심볼은 코드워드의 메시지 심볼과 함께 전송될 검사 심볼을 구성하는 최종의 나머지 심볼들의 집합을 제공하도록 처리된다.

실시예

본 발명의 리드 솔로몬(RS) 코딩 프로시저에 따른 메시지를 인코딩하는 특정 태스크를 수행하는 더 효과적인 방법을 설명하기 위해, 계산에 대한 상세한 설명이 필요하다. 문제는 x^{2t} 로 곱해진 메시지 다항식 M(x)를 코드 생성자 다항식 G(x)로 나눈 나머지를 구하는 것으로서 공식화된다.

나머지는 메시지를 반복함으로써, 즉, 메시지의 길이를 늘리고, 메시지의 이미 수신된 바이트들을 왼쪽으로 시프트하고, 오른쪽으로부터 새로운 바이트를 집어넣음으로써 계산되어, 제1 단계에서 메시지 $M_0(x)$가 하나의 바이트 M_0만을 포함하도록 하고,

$$M_0(x) = m_0x^0=m_0$$

$M_1(x)$는 두개의 바이트 m_1 및 m_0을 포함하는 것으로 간주되는데, 제1 바이트 m_0은 한 위치 왼쪽으로 시프트된다. 이것은 다항식 표현으로 다음과 같이 표기될 수 있다.

$$M_1(x) = m_0x+m_1 = M_0(x)*x + m_1$$

생성자 다항식 G(x)의 "꼬리" g(x)를 다음과 같이 도입하는 것이 편리하며,

$$G(x)= x^{2t}+ g_{2t-1} x^{2t-1}+...+ g_0 = x^{2t}+ g(x)$$

이것은 속성

$$G(x)+g(x)= x^{2t}$$

을 갖는다.

이 속성을 이용하여, 제1 단계에서, 메시지의 제1 심볼만이 인코딩될 때(나머지 r_0이 구해짐), 다음의 관계가 유지된다.

$$M_0(x)\,x^{2t} = m_0x^{2t} = Q_0(x)G(x) + R_0(x),\ R_0(x) = r_{0,2t-1}x^{2t-1} + \ldots + r_{0,0}$$

$$m_0x^{2t} = m_0G + m_0g$$

따라서, 이하의 식을 이용하여 반복이 시작된다.

$$Q_0(x)=m_0,\ \ R_0=m_0g$$

제2 단계 이후에, 나머지는 다음과 같이 계산될 수 있다.

$$M_1(x)\,x^{2t} = (M_0(x)*x + m_1)\,x^{2t} = (x*M_0(x)*x^{2t} + m_1x^{2t}) = x*Q_0*G + (x*R_0(x) + m_1x^{2t}) =$$

$$= (Q_0*x)*G + (r_{0,2t-1}x^{2t} + m_1x^{2t}) + (r_{0,2t-2}*x^{2t-1} + \ldots + r_{0,0}*x) =$$

$$= (Q_0*x + r_{0,2t-1} + m_1)*G + (m_1 + r_{0,2t-1})*g(x) + (r_{0,2t-2}*x^{2t-1} + \ldots + r_{0,0}*x + 0*x^0 +),$$

따라서, 다음의 수학식이 성립된다.

$$R_1(x) = (m_1 + r_{0,2t-1})*g(x) + (r_{0,2t-2}*x^{2t-1} + \ldots + r_{0,0}*x + 0*x^0)$$

이러한 접근을 이용하여, 일반적으로, k+ 1 단계 이후의 나머지 $r_{k+1}(x)$는 이전 단계 k에서의 나머지 $r_k(x)$ 및 새로 들어오는 바이트 m_{k+1}로부터 정확히 동일한 방식으로 계산될 수 있다.

$$R_{k+1}(x) = (m_{k+1} + r_{k,2t-1})*g(x) + (r_{k,2t-2}*x^{2t-1} + \ldots + r_{k,0}*x + 0*x^0)$$

$$= ((m_{k+1} + r_{k,2t-1})*g_{2t-1} + r_{k,2t-2})*x^{2t-1} + \ldots + ((m_{k+1} + r_{k,2t-1})*g_1 + r_{k,0})*x + (m_{k+1} + r_{k,2t-1})*g_0$$

이 계산은 2t개의 심볼로 구성된 "시프트 레지스터" r의 일반적 표현을 이용하여 구현될 수 있다.

$$r = r_{2t-1}, r_{2t-2}, \ldots r_0$$

각각의 반복 시에, 2t개의 심볼은 왼쪽으로 시프트되면서 오른쪽으로부터 0 심볼이 시프트-인되고 가장 왼쪽의 심볼 r_{2t-1}은 레지스터로부터 시프트-아웃된다. 이 가장 왼쪽의 심볼은 들어오는 심볼로 추가되고(XOR됨) "피드백" 심볼을 형성한다.

$$f_{k+1} = m_{k+1} + r_{k,2t-1}$$

이러한 정의를 이용하여, 나머지 $R_k(x)$는 다음의 공식에 따라 반복된다.

$$R_{k+1}(x) = (f_{k+1}g_{2t-1} + r_{k,2t-2})x^{2t-1} + (f_{k+1}g_{2t-2} + r_{k,2t-3})x^{2t-2} + \ldots + f_{k+1}g_0$$

나머지의 반복을 나타내는 개략적 도면이 도 1에 도시된다. 도 1에서, 메시지 심볼(10)(M_s)은 최상위의 나머지(high order remainder)(46)(r_3)와 합산되어, 갈루아 필드 요소(14)(f)를 제공한다. 상술된 바와 같이, 갈루아 필드 덧셈은 XOR

연산으로서 수행된다. 갈루아 필드 요소(14)는 코드 생성자 계수들(20, 22, 24 및 26)(g_0, g_1, g_2 및 g_3)로 곱해져, 중간 값들(30, 32, 34 및 36)을 제공한다. 중간 값들은 이전의 나머지 심볼들과 합산되어 현재의 나머지 심볼들을 제공한다. 따라서, 곱 fg_0은 0과 합산되어 나머지 심볼(40)(r_0)을 제공하고, 곱 fg_1은 이전의 나머지 심볼(40)과 합산되어 현재의 나머지 심볼(42)(r_1)을 제공하고, 곱 fg_2는 이전의 나머지 심볼(42)과 합산되어 현재의 나머지 심볼(44)(r_2)을 제공하고, 곱 fg_3은 이전의 나머지 심볼(44)과 합산되어 현재의 나머지 심볼(46)(r_3)을 제공한다. 이 프로세스는 코드워드 내의 각각의 메시지 심볼(10)에 대해 반복된다. 코드워드 내의 모든 메시지 심볼이 처리된 후에, 나머지 심볼들(40, 42, 44 및 46)은 메시지 심볼들과 함께 전송되기 위한 검사 심볼들을 나타낸다. 따라서, 코드워드는 메시지 심볼들 및 검사 심볼들을 포함한다. 도 1의 예는 메시지 심볼들과 함께 전송되기 위한 4개의 검사 심볼을 생성하며, 코드워드 내의 2개의 에러를 정정할 수 있다. 더 흔한 예로, 코드 생성기는 16개의 코드 생성자 계수 g_i를 포함하고, 코드워드 내의 8개의 에러를 정정하기 위한 16개의 검사 심볼을 생성한다.

나머지 다항식의 계산에 대한 통상의 룩업 테이블 접근이 사용될 때, 계산의 흐름에서의 주요 병목현상은 GF의 곱셈의 결과의 반복 계산이다. 메시지의 모든 바이트에 있어서, 2t번의 곱셈이 필요하다. 따라서, n개의 정보 바이트 및 2t개의 검사 바이트로 하나의 코드워드를 인코딩하기 위해서는, 2nt번의 GF 곱셈이 수행되어야 한다.

GF(256) t=8, n=239 이하로부터의 심볼을 이용하여 8개의 에러를 정정하는 리드 솔로몬 코드의 중요한 경우에, 총 3824번의 GF 연산이 수행된다. 10 내지 12의 주기/연산을 이용하면, 이것은 40K 주기/코드워드까지에 이른다.

2002년 3월 19일자 Yang의 미국 특허 제6,360,348호에서, 룩업 테이블은 나머지 다항식의 1회의 반복에 포함된 모든 GF 연산이 동일한 제2 피연산자들의 집합, 즉, 코드 생성자 다항식 g(x)의 2t개의 계수에 대해 수행된다는 소견에 기초하여, 계산을 용이하게 하도록 구성된다. 따라서, GF의 모든 n개의 요소들을 코드 생성자 다항식 g(x)의 2t개의 계수들로 곱한 결과인 2t*n개의 값은 미리 계산되고 룩업 테이블에 저장되어, 테이블 내의 각각의 엔트리 F(f, i)의 컨텐츠가 다음과 같이 정의되도록 한다.

$$F(f,i) = f \otimes g_i,$$

여기서, f는 다항식 형태의 갈루아 필드 요소를 나타내고, g_i는 코드 생성자 다항식의 계수이고, "$\otimes$"는 갈루아 필드 곱셈 연산을 표시한다.

이러한 접근법이 사용되는 경우, 정정된 에러의 개수가 비교적 작고(8개의 에러라고 가정) 사용된 갈루아 필드가 GF(256)더라도, 테이블의 크기는 4Kbyte(16*256*1바이트, 16은 8개의 에러를 정정하기 위한 검사 바이트의 개수임)이다. 종종, RS 코드에 대한 몇몇 상이한 옵션들이 실행되어야 한다. 예를 들면, ADSL 적용에 몇몇 지연 경로를 사용하여, 하나의 지연 경로에 있는 코드워드는 16개의 검사 바이트를 갖고 다른 지연 경로에서는 12개의 검사 바이트를 갖는 경우, 메모리 크기는 7Kb가 될 것이다.

본 발명의 특징에 따르면, GF 연산 $f \otimes g_i$(f는 다항식 형태의 임의의 GF 요소를 나타내고, g_i는 코드 생성자 다항식의 계수이고, "$\otimes$"는 갈루아 필드 곱셈 연산을 나타냄)는 두개의 작은 테이블을 이용하는 두개의 단계에서 수행된다.

1) 임의의 GF 요소 f는 우선, 그것의 "낮은" 부분과 "높은" 부분의 합산으로서 나타난다. 예를 들어, GF(256)에서, 다음과 같이 표기할 수 있다.

$$f = H0 + 0L$$

여기서, H 및 L은 높은 마스크와 낮은 마스크를 이용한 두개의 AND 연산을 이용하여 f로부터 쉽게 획득된다.

$$H = f \,\&0xF0,\ L = f\& \ 0x0F$$

2) GFMUL 연산 $f \otimes g_i$는 갈루아 필드의 속성을 이용하여 다음과 같이 수행될 수 있다.

$$f \otimes g_i = (H + L) \otimes g_i = H \otimes g_i + L \otimes g_i$$

수학식 23 내지 25의 연산은 도 2에 나타나 있다. 도시된 바와 같이, 갈루아 필드 요소(14)는 수학식 24에 따라 높은 마스크와 낮은 마스크를 이용하여 높은 해프워드(50)(H0)와 낮은 해프워드(52)(HL)로 분리된다. 높은 해프워드와 낮은 해프워드는 각각, 각각의 코드 생성자 계수(20, 22, 24 및 26)로 곱해진다. 각각의 곱들은 합산되어 중간 값들을 제공한다. 중간 값들은 이전의 나머지 심볼들과 합산되어 현재의 나머지 심볼들을 제공한다. 도 2의 아키텍처는 룩업 테이블과 결합되어, 이로운 결과를 달성한다.

3) 수학식 25를 이용하여, 두개의 작은 룩업 테이블이 형성되고, 각각은 "높은" 해프워드와 "낮은" 해프워드의 모든 16가지 경우에 의한 생성자 다항식 계수들 g_i의 집합의 GFMUL(갈루아 필드 곱셈)의 연속적 결과를 유지한다.

$$TH(H,i) = (H0) \otimes g_i$$

$$TL(L,i) = (0L) \otimes g_i$$

4) 코드 생성자 다항식의 GF(256) 및 16개의 계수의 경우, 각각의 테이블의 크기는 256 바이트이고, 두개의 테이블의 총 크기는 종래의 기술보다 8배 더 작은 512 바이트이다. 이 때, 연산의 ~50%는 GFMUL 연산의 실행과 직접 연결되지 않기 때문에 총 실행 시간은 ~50%만이 더 길어진다.

하나의 단계(하나의 정보 심볼/바이트)에 의한 나머지의 반복과 관련된 연산은 다음의 순서로 실행될 수 있다.

$$f = m_{k+1} \wedge r_{k,2t-1}$$

$$f = H0 + 0L$$

$$\text{for } i=1:2t-1 \;\; s = TH(H,i) \wedge TL(L,i), \;\; r_i = s \wedge r_{i-1},\; r_0 = TH(H,0) \wedge TL(L,0)$$

각각의 반복은 룩업 테이블들로의 두번의 액세스, 및 임의의 마이크로프로세서에 쉽게 이용가능한 후속하는 XOR 연산을 요구한다.

룩업 테이블 TH 및 TL을 이용한 코드 생성은 도 3에 개략적으로 도시되어 있다. 상술된 바와 같이, 제1 룩업 테이블은 코드 생성자 계수들 각각으로 곱해진 높은 해프워드 H0의 값을 포함하고, 제2 룩업 테이블은 코드 생성자 계수들 각각으로 곱해진 낮은 해프워드 0L의 값을 포함한다. 높은 해프워드 값은 계수 인덱스와 함께 사용되어, 테이블 TH에 액세스하여 하나 이상의 높은 테이블 값(54)을 제공하고, 낮은 해프워드 값은 그 계수 인덱스와 함께 사용되어, 테이블 TL에 액세스하여 하나 이상의 낮은 테이블 값(56)을 제공한다. 동일한 계수 인덱스를 갖는 각각의 높은 테이블 값 및 낮은 테이블 값은 합산 연산(60)으로 합산되어 중간 값들을 제공하고, 중간 값들은 합산 연산(62)으로 각각의 이전의 나머지 심볼들과 합산되어 현재의 나머지 심볼들(40, 42, 44 및 46)을 제공한다. 이하에 설명되는 바와 같이, 룩업 테이블은 코드 생성을 수행하는 시스템의 아키텍처에 따라, 합산될 테이블 값들의 단일 집합, 또는 합산될 테이블 값들의 둘 이상의 집합을 제공할 수 있다.

본 발명의 일 실시예에 따른 룩업 테이블의 컨텐츠는 도 4a 및 도 4b에 도시되어 있다. 도 4a는 높은 해프워드 테이블(70)(TH)을 도시하고, 도 4b는 낮은 해프워드 테이블(72)(TL)을 도시한다. 도시된 바와 같이, 테이블(70)은 코드 생성자 계수들 각각으로 곱해진 높은 해프워드 H0의 각각의 가능한 값의 곱을 포함한다. 따라서, 테이블(70) 내의 각각의 엔트리는 g_iH의 값을 포함하는데, 여기서, i는 0 내지 2t-1까지의 범위에 있고, H는 0 내지 높은 해프워드 H의 최대 값까지의 범위에 있다. 마찬가지로, 테이블(72) 내의 각각의 엔트리는 giL의 값을 포함하는데, 여기서, i는 0 내지 2t-1까지의 범위에 있고, L은 0 내지 낮은 해프워드 L의 최대 값까지의 범위에 있다. 메시지 심볼이 바이트인 흔한 경우에 있어서, 높은 해프워드는 00 내지 F0(16진수 표시를 사용) 까지의 범위의 16개의 가능한 값들을 갖는다. 코드 생성자 계수의 개수는 요구된 검사 심볼의 개수에 의존하며, 8개의 검사 심볼의 경우에 있어서 16일 수 있다. 마찬가지로, 테이블(72)은 코드 생성자 계수들 각각으로 곱해진 낮은 해프워드 0L의 각각의 가능한 값의 곱을 포함한다. 메시지 심볼이 바이트이면, 낮은 해프워드 값은 0 내지 F까지의 범위에 있다. 따라서, 메시지 심볼이 바이트이고 코드 생성자가 16개의 계수를 갖는 경우에 있어서, 룩업 테이블(70 및 72) 각각은 256개의 엔트리를 갖는다. 도 4a 및 도 4b의 예에서, 테이블(70 및 72)은 각각의 로우가 하나의 해프워드 값에 대응하는 곱을 포함하도록 조직된다. 이하에 설명된 바와 같이, 이 조직은 일반적인 프로세서에서의 병렬 실행을 용이하게 한다.

본 발명은 8비트 갈루아 필드 요소가 높은 해프워드와 낮은 해프워드로 나뉘어지고 두개의 룩업 테이블이 이용되는 실시예에 관련하여 설명되었다. 다른 실시예에서, 각각의 갈루아 필드 요소는 9 이상의 비트를 가질 수 있다. 또한, 각각의 갈루아 필드 요소는 3개 이상의 성분로 분리될 수 있고, 3개 이상의 룩업 테이블이 사용될 수 있다. 16 비트 갈루아 필드 요소를 생각해 보자. 특허 제6,360,348호에 개시된 접근법은 2M 바이트의 룩업 테이블을 필요로 한다. 각각의 16비트 갈루아 필드 요소가 두개의 성분로 분리되는 실시예에서, 각각이 8K 바이트(16 계수 × 256 해프 GF 요소 값 × 2 엔트리 당 바이트)인 두개의 룩업 테이블이 필요하다. 각각의 16비트 갈루아 필드 요소가 4개의 성분로 분리되는 실시예에서, 각각이 512 바이트(16 계수 × 16 쿼터 GF 요소 값 × 2 엔트리 당 바이트)인 4개의 룩업 테이블이 필요하다.

데이터 액세스 워드 크기가 9비트 이상의 폭인 임의의 DSP(digital signal processor)에 있어서(모든 현대의 DSP의 경우임), 테이블 액세스는 단조롭게 증가하는 제2 인덱스를 이용하여 행해져서, 테이블은 순차적인 액세스를 갖는 단방향 어레이로서 사용될 수 있고, 시프트 레지스터 셀들은 엄격한 순차로 사용된다는 사실 때문에, 본 실시예의 추가적 이점이 단순히 달성될 수 있다. 이 사실은, 우선 한 주기에서 복수의 엔트리가 판독되고, 둘째로 동시에 복수의 갈루아 필드 심볼들(예를 들면, 32비트 데이터 워드 경로를 갖는 DSP 상에 구현될 경우에는 한번에 4개의 심볼들, 64비트 데이터 워드 경로를 갖는 DSP에 대해서는 8개의 심볼들 등)에 대한 XOR 연산도 수행될 수 있기 때문에 실행의 속도를 매우 증가시킨다. 예를 들어, 특정 제한을 갖는 32비트 데이터 경로를 갖는 Blackfin DSP 상에서의 한 특정 구현은 1.5 주기/검사 바이트/데이터 바이트(통상의 방법보다 5 내지 6배 빠르고, 심지어는 내장형 갈루아 필드 곱셈기를 갖는 DSP보다 2배 더 빠름)에 가까운 속도로 동작한다.

본 발명의 일 실시예에 따라 데이터를 인코딩하기 위한 디지털 신호 프로세서의 블럭도는 도 5에 도시되어 있다. 본 발명의 일 실시예에 따라, w개의 갈루아 필드 요소의 워드 폭을 갖는 디지털 신호 프로세서에서 데이터를 인코딩하기 위한 프로세스가 도 6에 도시되어 있다. 도 5의 실시예에서, 디지털 신호 프로세서의 코어 프로세서(100)는 메시지 심볼 m을 수신하고, 코드워드의 메시지 심볼과 함께 전송될 검사 심볼 r을 공급한다. 코어 프로세서(100)는 높은 해프워드 룩업 테이블(110)에 액세스하고 테이블 값들 g_iH, $g_{i+1}H$, $g_{i+2}H$ 및 $g_{i+3}H$를 수신한다. 코어 프로세서(100)는 또한, 낮은 해프워드 룩업 테이블(112)에 액세스하고 테이블 값 g_iL, $g_{i+1}L$, $g_{i+2}L$ 및 $g_{i+3}L$을 수신한다. 도 5의 실시예에서, 메시지 심볼 m은 바이트이고 코어 프로세서(100)는 32비트의 워드 폭을 갖는다. 따라서, 코어 프로세서(100)는 도 5에 도시되고 이하에 설명되는 룩업 테이블(110) 및 룩업 테이블(112) 각각으로부터 주기 당 4개의 테이블 값을 처리할 수 있다. 서로 다른 워드 폭들을 갖는 코어 프로세서는 주기 당 더 많거나 더 적은 테이블 값들을 처리할 수 있다.

도 6을 다시 참조하면, 단계(150)에서 다음의 메시지 심볼 m이 수신되고 처리된다. 단계(152)에서, 갈루아 필드 요소 f는 수학식 28에 나타난 바와 같이, 메시지 심볼 및 최상위의 나머지 심볼로부터 계산된다. 단계(154)에서, 갈루아 필드 요소는 수학식 23 및 24에 나타난 바와 같이, 제1 및 제2 성분 H0 및 0L로 분리된다. 이 실시예에서, 갈루아 필드 요소는 높은 해프워드와 낮은 해프워드로 분리된다. 그러나, 다른 실시예에서, 갈루아 필드 요소는 3개 이상의 성분들로 분리될 수 있다.

단계(156)에서, 높은 해프워드 H0 및 인덱스 i는 룩업 테이블(도 5의 110)로부터 w개의 테이블 값들에 액세스하는 데 사용된다. 인덱스 i는 생성자 계수 g_i의 인덱스이다. 코드워드는 16개의 검사 심볼을 갖고 코드 생성자 다항식은 16개의 계수를 갖는 예에서, 인덱스 i는 0 내지 15까지의 범위에 있다. 또한, 단계(156)에서, 낮은 해프워드 0L와 인덱스 i는 룩업 테이블(도 5의 112)로부터 w개의 테이블 값을 획득하는 데 사용된다. 상술된 바와 같이, 동시에 액세스된 테이블 값들의 개수는 갈루아 필드 요소의 단위로 표현된 코어 프로세서(100)의 워드 폭에 의존한다. 코어 프로세서(100)가 32비트 워드 폭을 갖고 메시지 심볼이 바이트인 일반적인 경우에 있어서, w=4인 테이블 값들은 룩업 테이블(110 및 112) 각각에서 동시에 액세스된다.

단계(160 및 162)에서, 나머지 심볼 r_i는 단계(156)에서 액세스된 테이블 값들의 w개의 집합들 각각에 대해 계산된다. 특히, 중간 값 s_i는 단계(160)에서 수학식 30에 도시된 바와 같이 테이블 값들의 w개의 집합들 각각에 대해 계산된다. 테이블 값들의 집합은 룩업 테이블(110)로부터의 테이블 값, 및 룩업 테이블(112)로부터의 테이블 값을 포함하며, 이 두개의 테이블 값들은 동일한 생성자 계수 인덱스를 갖는다. 그리고 나서, 현재의 나머지 심볼 r_i는 수학식 30에도 나타난 바와 같이, w개의 중간 값들 s_i 각각에 대해, 중간 값 s_i 및 이전의 나머지 심볼 r_{i-1}에 대한 XOR 연산에 의해 계산된다. 따라서, 단계(162)에서 w개의 나머지 심볼들이 계산된다.

단계(170)에서, 2t개의 나머지 심볼들이 계산되었는지에 대한 결정이 이루어지고, 이 때, 2t는 코드 생성자 다항식 내의 계수들의 개수를 나타낸다. 2t개의 나머지 심볼들이 계산되지 않았으면, 단계(172)에서 인덱스 i는 w만큼 증가한다. 그 후,

프로세스는 단계(156)로 돌아와, 테이블(110) 내의 w개의 새로운 테이블 값들, 및 테이블(112) 내의 w개의 새로운 테이블 값들에 액세스한다. 단계(156, 160, 162, 170 및 172)를 포함하는 루프는 2t개의 나머지 심볼들이 계산될 때까지 반복된다. w=4이고 2t=16인 예에 있어서, 루프를 4번 반복하는 것이 필요하다.

2t개의 나머지 심볼들이 계산되었으면, 단계(180)에서, 코드워드의 끝에 도달되었는지에 관한 결정이 이루어진다. 일반적인 코드워드는 188개의 메시지 심볼들을 포함할 수 있다. 코드워드의 끝에 도달되지 않았으면, 프로세스는 단계(150)로 되돌아가, 다음의 메시지 심볼 m을 처리한다. 코드워드의 끝에 도달되었으면, 단계(182)에서 2t개의 검사 심볼이 출력된다. 따라서, 예를 들어, 16개의 검사 심볼은 코드워드의 188개의 메시지 심볼과 함께 전송되기 위해 출력될 수 있다.

ADSL 모뎀과 같은 몇몇 응용에서, RS 코드의 매개변수들은 "작동 중에" 설정될 수 있고, 하나의 ADSL 채널 내에서 서로 다른 베어러 채널(bearer channel)들에 대해 서로 다를 수 있다. 이 경우에, 본 발명에 기초한 두개의 인코딩 방법이 실현될 수 있다.

제1 방법에 따르면, 가장 가능성 높은 개수의 검사 바이트들에 대한 테이블만이 생성되고, 더 낮은 에러 정정 능력을 갖는 코드가 사용되는 경우, 인코더는 다음과 같이 두개의 단계에서 동작한다.

우선, 더 높은 차수의 코드 생성자 다항식 G_{2t}는 더 낮은 차수의 코드 생성자 다항식 G_{2s}를 인수로서 포함하는 것이 관찰된다.

$$G_{2s}(x) = (x-a^{2s-1})\ldots(x-a^{0})$$

$$G_{2t}(x)=(x-a^{2t-1})\ldots(x-a^{0})=(x-a^{2t-1})\ldots(x-a^{2s})(x-a^{2s-1})\ldots(x-a^{0}) = h_{2d}(x)G_{2s}(x)$$

여기서,

$$h_{2d}(x)=(x-a^{2t-1})\ldots(x-a^{2s})$$

이고, d=t-s는 코드의 에러 정정 능력들 간의 차이이다.

따라서, n차의 메시지 다항식은 두개의 부분, 즉, 더 높은 n-2d개의 항들과 더 낮은 2d개의 항들로 분할된다.

$$M_n(x) = M_{n-2d}(x)x^{2d} + M_{2d}(x)$$

코드 생성자 다항식이 나뉘어져야 하는 시프트된 메시지 다항식은 다음과 같이 나타난다.

$$M_n(x)x^{2s} = M_{n-2d}(x)\,x^{2d}x^{2s} + M_{2d}(x)x^{2s} = M_{n-2d}(x)\,x^{2s+2d} + M_{2d}(x)x^{2s} = M_{n-2d}(x)\,x^{2t} + M_{2d}(x)x^{2s}.$$

이제, 더 높은 2t차의 코드 생성자 다항식에 관련된 메시지의 처음 n-2d개의 바이트들에 대한 나머지는 다음과 같이 획득된다.

$$M_{n-2d}(x)x^{2t} = Q(x)G_{2t}(x) + R(x)$$

여기서 첫번째 항은 다음과 같이 표기될 수 있다.

$$Q(x)G_{2t}(x) = Q(x)h_{2d}(x)G_{2s}(x) = Q_1(x)G_{2s}(x)$$

이 때,

$$Q_1(x) = Q(x)h_{2d}(x)$$

이다.

동일한 논리에 따라, 2t로 구성된 "긴" 나머지 R(x)는 두개의 부분, 즉, 더 높은 2d개의 항들 및 더 낮은 2s개의 항들로 분리된다.

$$R(x) = r_{2d}(x)x^{2s} + R_{2s-1}(x)$$

마지막으로, 인코딩은 다음의 수학식에 따라 수행될 수 있다.

$$M_n(x)x^{2s} = M_{n-2d}(x)\,x^{2t} + M_{2d}(x)x^{2s} = Q_1(x)G_{2s}(x) + M_{2d}(x)x^{2s} + r_{2d}(x)x^{2s} + R_{2s-1}(x)$$

이것은 비교적 짧은 메시지(단 2d 바이트)를 인코딩하는 것

$$(M_{2d}(x) + r_{2d}(x))x^{2s} = Q_2(x)G_{2s}(x) + R_2(x)$$

및, 이 연산의 결과와 "긴" 인코딩의 "짧아진" 결과를 더하는 것

$$M_n(x)x^{2s} = Q_1(x)G_{2s}(x) + Q_2(x)G_{2s}(x) + R_2(x) + R_{2s-1}(x) = (Q_1(x) + Q_2(x))G_{2s}(x) + (R_2(x) + R_{2s-1}(x))$$

과 동등하다.

결과로서, 총 일련의 인코딩은 다음과 같이 설명될 수 있다. 메시지 $M_{2d}(x)$의 마지막 2d개의 바이트들을 획득된 나머지 $r_{2d}(x)$의 처음의(가장 왼쪽의) 2d개의 바이트와 더하고(XOR됨), 이 2d개의 바이트들은 짧은 메시지로서 다루어지고, 통상의 방법을 이용하여 인코딩된다. 결과는 "긴" 나머지 R(x)의 남은 바이트들에 더해진다. 이것이 인코딩의 최종 결과이다.

이 방법은 2d개의 바이트들의 추가의 "재인코딩"에 기인한 오버헤드 D(t, d)를 갖는데, 이것은 인코딩된 바이트들의 개수, 및 결과 내의 바이트들의 개수에 비례한다.

$$D(t,d) = K*2d*2(t-d)$$

여기서, K는 통상의 인코딩에 필요한 주기의 수, 보통, 10 내지 12이다. 이 오버헤드는 d=t/2 $K*t^2$에 대한 최대값을 가지며, K=12, t=8인 768의 주기 레벨일 수 있는데, 이것은 인코딩의 주요 부분의 ~20%를 구성한다.

다른 옵션은 추가의 메모리를 필요로 할 수 있는 임의의 요구된 코드 매개변수에 대해 작동 중에 "올바른" 테이블을 생성하는 것이다. 그러나, 이 경우에서는, 넓은 데이터 경로를 갖는 DSP를 충분히 이용하기 위해 본 발명의 일 양태에 따라, 검사 바이트의 개수가 4의 정수배가 아닐 때, 테이블은 32*4k 크기로 생성되는데, 여기서 4k는 위에서부터 특정 개수의 검사 바이트에 가장 가까운 4의 정수배, 예를 들면, 14에 대해서는 16, 10에 대해서는 12 등이다. 남은 64바이트는 0으로 채워져서, 이들 값이 판독될 때, 그것들은 계산의 결과에 영향을 끼치지 않는다.

인코더에 사용된 동일한 테이블은 다음의 방식으로 신드롬(syndrome)을 계산하기 위해 디코더에서 사용될 수 있다. 체계적 형태의 수신된 코드워드는 우선 "재인코딩"된다. 인코더 메커니즘은 코드워드의 정보 바이트에 대해 실행되어, 실제로 수신된 검사 바이트들의 시퀀스의 "복사본"을 획득한다. 그 후, 이 재구성된 "복사본"은 실제로 수신된 검사 바이트를 가지고 논리 AND 연산자를 이용하여 비트마다 비교되어, 신드롬과 매우 관련된 다항식을 형성한다. 즉, GF의 원시 요소의 연속적인 곱에 대한 이 다항식의 값들은 정확히, 다음의 관계로 인한 신드롬의 성분들이다.

$$M(x) = Q(x)G(x) + R(x),\ S_k = M(a^k) = Q(a^k)G(a^k) + R(a^k) = R(a^k),\ k = 0,1,\ldots 2t-1$$

코드 생성자 다항식의 정의에 의해 $G(a^k)=0$이기 때문에,

$$G(x) = (x-a^{2t-1})\ldots(x-a^0)$$

이다.

따라서, n 바이트 길이의 원래의 코드워드보다 훨씬 더 짧은 2t 바이트들의 시퀀스에 대한 통상의 접근법을 이용하여 신드롬의 축적이 수행될 수 있다. 이것은 에러 위치지정자 다항식의 계수를 구하기 위해 berlekamp massey 또는 유클리드 알고리즘을 구현하는 데 필요한 연산들의 동일한 순서인 $(2t-1)^2$개의 고려할 만한 GF 곱셈들을 포함한다. GF 곱셈들은 피연산자들 중 하나의 고정된 집합에 대해 수행되기 때문에, 이 계산의 속도는 실제로, 일반 GF 곱셈보다 30 내지 50% 높아서, (255, 239, 16) RS 코드의 경우에, 많아야 255*6=1350 주기가 필요하며, 이것은 30% 오버헤드이다.

지금까지 특징적 개념 및 다수의 예시적인 실시예를 설명하였지만, 본 기술분야에 숙련된 자들은 본 발명이 여러 방식으로 구현될 수 있으며, 변경 및 개선이 쉽게 일어날 것임을 인지할 것이다. 따라서, 주어진 예는 제한적인 것으로 의도되지 않으며, 예를 통해서만 제공된다. 본 발명은 오직, 다음의 청구범위 및 그 동등물에 의해 요구되는 것으로서 제한된다.

도면의 간단한 설명

본 발명을 가장 잘 이해하기 위하여, 본 명세서에 참조로서 통합된 첨부 도면에 대한 참조가 이루어진다.

도 1은 리드 솔로몬 인코딩을 위한 종래의 방법의 개략적인 블럭도.

도 2는 코드 생성자 계수에 의한 곱셈 이전에 갈루아 필드 요소가 해프워드들로 분리되는 리드 솔로몬 인코딩을 위한 방법을 도시하는 개략적인 블럭도.

도 3은 본 발명의 일 실시예에 따른 리드 솔로몬 인코딩을 도시하는 개략적인 블럭도.

도 4a 및 도 4b는 본 발명의 일 실시예에 따른 리드 솔로몬 인코딩을 위한 룩업 테이블의 실시예를 도시.

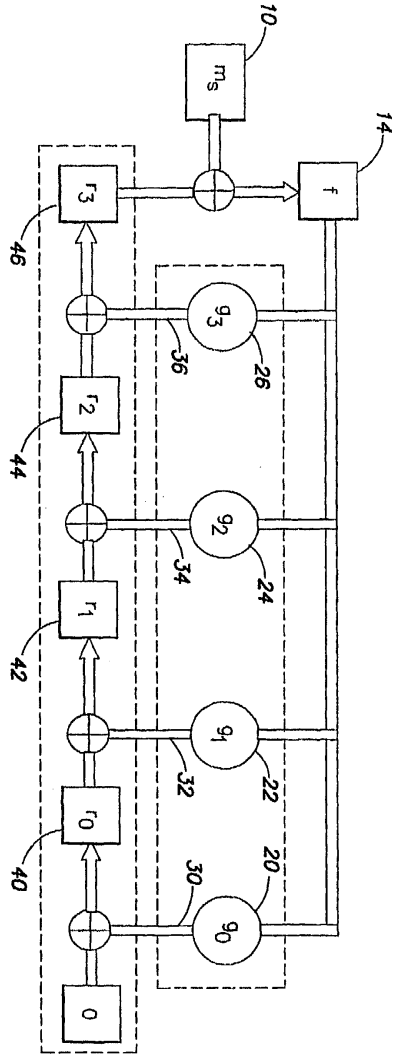
도 5는 본 발명의 일 실시예에 따른 디지털 신호 프로세서 내의 리드 솔로몬 인코딩을 도시하는 개략적인 블럭도.

도 6은 본 발명의 일 실시예에 따른 리드 솔로몬 인코딩을 도시하는 흐름도.

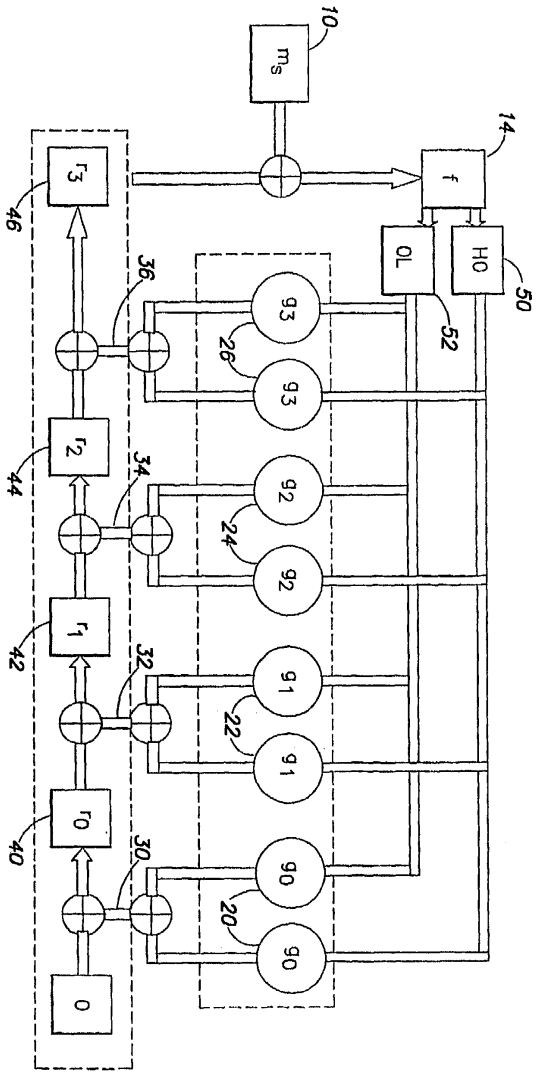
도면

도면1

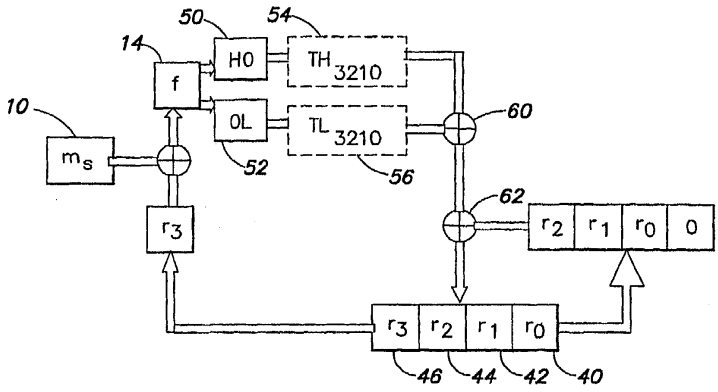
(종래 기술)



도면2


10
ms
14
f
OL
H0
50
52
r3
46
36
g3
26
g3
r2
44
34
g2
24
g2
r1
42
32
g1
22
g1
r0
40
30
g0
20
g0
0

도면3


50
54
14
H0
TH 3210
f
OL
TL 3210
10
ms
52
56
60
62
r2
r1
r0
0
r3
r2
r1
r0
46
44
42
40

도면4a

70

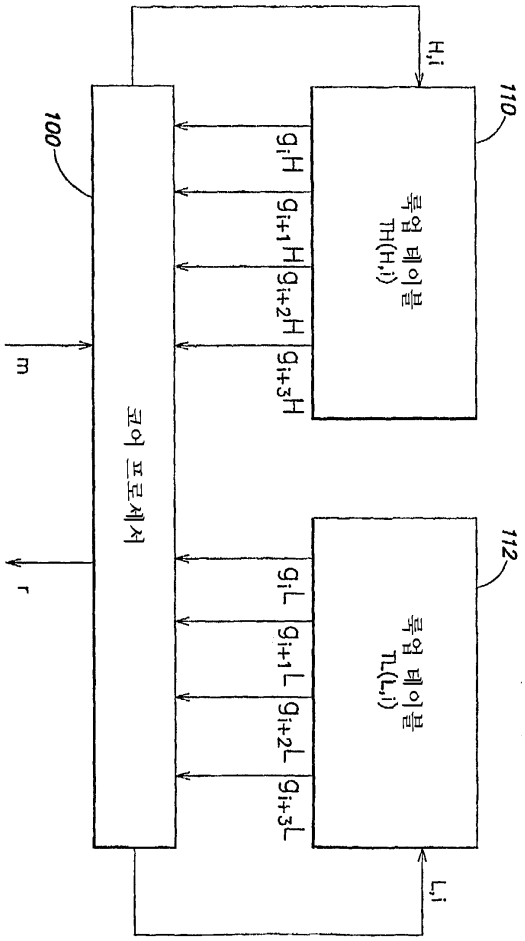
TabH	g_0H	g_1H	•••	$g_{2t-1}H$
00	0	0	0	0
10	g_0*10	g_1*10	•••	$g_{2t-1}*10$
•••	•••	•••	•••	•••
F0	g_0*F0	g_1*F0	•••	$g_{2t-1}*F0$

도면4b

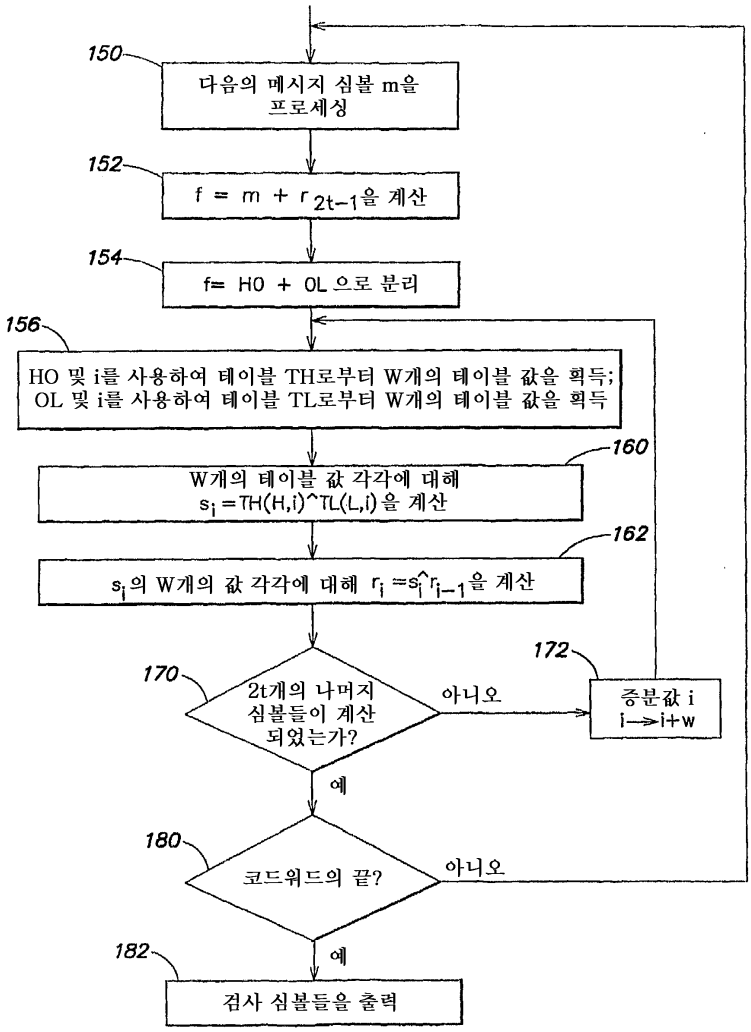
72

TabL	g_0L	g_1L	•••	$g_{2t-1}L$
0	0	0	0	0
1	g_0	g_1	•••	g_{2t-1}
•••	•••	•••	•••	•••
F	g_0F	g_1F	•••	$g_{2t-1}F$

도면5



도면6


150
다음의 메시지 심볼 m을 프로세싱
152
f = m + r2t-1을 계산
154
f= HO + OL 으로 분리
156
HO 및 i를 사용하여 테이블 TH로부터 W개의 테이블 값을 획득; OL 및 i를 사용하여 테이블 TL로부터 W개의 테이블 값을 획득
160
W개의 테이블 값 각각에 대해 si = TH(H,i)^TL(L,i)을 계산
162
si의 W개의 값 각각에 대해 ri = si^ri-1을 계산
170
2t개의 나머지 심볼들이 계산 되었는가?
아니오
172
증분값 i
i->i+w
예
180
코드워드의 끝?
아니오
예
182
검사 심볼들을 출력