(19)대한민국특허청(KR)
(12) 공개특허공보(A)

(51) 。Int. Cl.[7]
H03M 13/00

(11) 공개번호 10-2004-0075952
(43) 공개일자 2004년08월30일

(21) 출원번호 10-2004-7011352
(22) 출원일자 2004년07월22일
번역문 제출일자 2004년07월22일
(86) 국제출원번호 PCT/US2003/001913
(87) 국제공개번호 WO 2003/063363
(86) 국제출원출원일자 2003년01월22일
(87) 국제공개일자 2003년07월31일

(30) 우선권주장 10/055,114 2002년01월23일 미국(US)

(71) 출원인 톰슨 라이센싱 에스.에이.
프랑스 92648 블로뉴 세데 께 알퐁스 르 갈로 46

(72) 발명자 리트윈,2세,루이스,로버트
미국 08536 뉴저지주 플레인스보로 피.오. 박스 934

벨레즈,디디에르
미국 46032 인디애나주 카멜 난세몬드 드라이브 13970

(74) 대리인 주성민
백만기
전경석

심사청구 : 없음

(54) 인트라-디코더 컴포넌트 블록 메시징

요약

디코더 및 디코딩 방법이 개시되며, 디코더 및 디코딩 방법에 있어서, 신드롬은 신드롬 생성기(410)내 코드 워드로부터 계산되고, 에러 다항식은 에러 다항식 생성기(420)내 신드롬에 기초하여 생성되며, 에러 위치는 에러 위치 생성기(430)내 에러 다항식으로부터 결정되며, 에러 크기는 에러 크기 생성기(440)내 에러 다항식으로부터 계산되며, 그리고 코드 워드는 에러 위치와 에러 크기에 응답하는 에러 정정된 코드 워드 생성기(450)에 의해 정정된다. 인트라-디코더 블록 메시징 방식은, 하나 이상의 컴포넌트들(410,420,430,440,450)이 비활성 메시지들(640,650)을 생성하여 다음 코드 워드에 대응하는 데이터를 처리하는 능력의 신호를 전송하는 방식으로 개시된다. 이중 탐색 셀 블록(930,940) 구현은, Chien/Forney 블록(940)에 의해 수행된 에러 위치 및 크기의 계산들과는 별개로, 특정 코드 워드에 대응하는 에러들의 개수를 결정하기 위해 이용되는 것으로 개시된다. 강화된 Chien 탐색 셀 구조는, 부가적인 Galois 필드 덧셈기(1140)를 이용하여 코드 워드와 에러 벡터를 동기화하며, 그로 인하여, LIFO 레지스터(952)로 구현된 에러 정정 블록(950)에 대응하는 지연 및 비용을 감소시키는 방식으로 개시된다.

대표도

도 7

색인어

코드 워드, 신드롬, 에러 다항식, 에러 위치, 에러 크기

명세서

기술분야

관련 출원들에 대한 상호 참조

본 출원은 2002년 1월 23일자로 동시에 출원된 발명의 명칭 DUAL CHIEN SEARCH BLOCKS IN AN ERROR-CORRECTING DECODER의 미국 특허 출원 제10/055,076호(대리인 정리 번호 PU020003); 그리고 발명의 명칭 CHIEN SEARCH CELL FOR AN ERROR-CORRECTING DECODER의 미국 특허 출원 제10/055,470호(대리인 정리 번호 PU02001)와 관련된 것이며, 두 개의 출원 모두 본 명세서에 참조되어 있다.

본 발명은 디지털 통신 시스템 에러 검출 및 정정에 관한 것이다.

배경기술

현대식 디지털 통신 시스템의 중요한 기능 중 하나는 에러 제어 코딩이다. 에러 제어 코딩은 디지털 시스템의 에러들을 검출하고 정정하기 위한 기술들을 다루는 통신의 분야이다. 일반적으로, 에러 검출/정정 방식들은, 디지털 데이터의 전송 또는 저장 시 에러가 데이터에 도입되지 않음을 보장하거나, 또는 에러가 데이터에 도입되더라도, 도입된 에러는 정정됨을 보장하는 것이 바람직한 경우에 이용된다. 데이터 에러들을 검출 및/또는 정정하는 능력은 리던던시(redundancy)를 데이터에 부가함으로써 이루어진다. 전송 또는 저장된 데이터에 리던던트 비트들 을 포함시킴으로써, 본래의 코딩되지 않은 신호 또는 필드(field)보다 코딩된 신호 또는 필드가 많은 비트들로 구성된다.

에러 검출/정정을 위해 자주 이용되는 방식은 리드-솔로몬으로 불리는 코드이다. 리드-솔로몬 코드들은 비(非)이진 계통의 주기적인 선형 블록 코드들이다. 비(非)이진 코드들은 몇몇 비트들로 구성되는 심볼들로 동작한다. 리드-솔로몬 코드와 같은 비(非)이진 코드는 버스트(burst) 에러들의 정정에 적합한데, 이러한 코드들에 의한 정정이 심볼 레벨에서 이루어지기 때문이다. 리드-솔로몬 코드와 같은 계통 코드는, 메시지 심볼들을 변경되지 않은 폼(form)에 포함하는 코드 워드들을 생성한다. 인코더는, 리던던시, 또는 패리티(parity), 심볼들을 생성하기 위하여 가역성의 수학 함수를 메시지 심볼들에 적용한다. 코드 워드는 패리티 심볼들을 메시지 심볼들에 부가함으로써 형성된다. 리드-솔로몬 코드는 주기적인 코드로 여겨지는데, 임의의 유효 코드 워드의 주기적인 시프트(shift)가 다른 유효 코드 워드를 또한 제공하기 때문이다. 주기적인 코드들이 대중적인데, 그 코드들을 실행하기 위한 효율적이며 저렴한 디코딩 기술들이 존재하기 때문이다. 마지막으로, 리드-솔로몬 코드는 선형으로 여겨지는데, 어떤 두 개의 유효 코드 워드들을 합하면 다른 유효 코드 워드가 되기 때문이다.

전통적인 리드-솔로몬 디코더는 다음의 중요한 컴포넌트 블록들로 구성된다: (ⅰ) 신드롬 생성 블록, (ⅱ) 에러 다항식 블록, (iii) 에러 위치 블록, (iv) 에러 크기 블록, (ⅴ) 에러 정정 블록, 및 (vi) 지연 블록. 신드롬 생성 블록은 코드 워드를 수신하여 신드롬을 코드 워드로부터 생성하기 위해 이용된다. 신드롬은 에 러 다항식을 에러 다항식 블록에 생성하기 위해 이용된다. 에러 다항식은 에러 위치 및 에러 크기 블록들로 이동되는데, 그 블록들에서 코드 워드에 대한 에러 위치들과 크기들이 각각 결정된다. 에러 벡터는 에러 위치 및 크기로부터 생성된다. 수신된 코드 워드의 지연 버전(version)은 특정 코드 워드에 대응하는 에러 벡터를 이용하는 에러 정정 블록에 의해 정정된다. 지연 블록은 수신된 코드 워드를 저장하기 위한 메모리로 구성되는 반면, 디코딩 프로세스는 에러 벡터를 제공하기 위해 수행된다.

종래 기술의 디코더들은 이러한 블록들을 '전달 파이프라인'을 형성하기 위해 이용한다. 즉, 하나의 블록으로의 입력은 이전 블록의 출력에만 의존하며, 하나의 컴포넌트 블록으로부터 이전 또는 이후의 블록으로의 피드백은 존재하지 않는다. 그러므로, 한번에 하나의 코드 워드만이 처리될 수도 있으며, 처리될 다음 코드 워드는 이전 코드 워드의 프로세싱이 완료된 이후에 처리된다.

<발명의 개요>

종래 기술의 비효율성은, 코드 워드내 데이터 에러 검출을 수행하기 위한 본 발명의 방법 및 장치에서 극복된다. 데이터 에러 검출을 수행하는 디코더는, 신드롬을 코드 워드로부터 계산하기 위한 신드롬 생성기; 에러 다항식을 신드롬으로부터 생성하기 위한 에러 다항식 생성기, 에러 위치를 에러 다항식으로부터 결정하기 위한 에러 위치 생성기, 및 에러 크기를 에러 다항식으로부터 계산하기 위한 에러 크기 생성기를 포함한다. 디코더는, 인트라-디코더 비활성 메시지를 신드롬 생성기, 에러 다항식 생성기, 에러 위치 생성기, 및 에러 크기 생성기를 포함하는 그룹 으로부터 선택된 적어도 하나의 컴포넌트로부터 송신하도록 적응되는 것을 특징으로 한다.

도면의 간단한 설명

본 발명의 후술하는 설명들은 첨부한 도면들을 참조하여 이루어진다. 여기서:

도 1은 에러 정정 방식을 포함하는 디지털 데이터 전달 시스템의 블록도;

도 2는 종래의 에러 정정 방식 방법론를 설명하는 흐름도;

도 3은 다양한 에러 정정 방식들의 계층도;

도 4는 리드-솔로몬 디코더를 나타내는 블록도;

도 5는 본 발명의 원리에 따라 이용된 리드-솔로몬 디코더의 예시적인 실시예를 나타내는 블록도;

도 6은 본 발명의 원리에 따른 예시적인 인트라-디코더 핸드세이킹 프로토콜을 설명하는 블록도;

도 7은 본 발명의 원리에 따른 리드-솔로몬 디코더의 예시적인 실시예의 기능 블록들간의 핸드세이킹 프로토콜을 설명하는 블록도;

도 8은 본 발명의 원리에 따른 인트라-블록 핸드세이킹을 이용하는 디코더와 관련된 효율들을 입증하는 예시적인 리드-솔로몬 디코더들을 위한 타이밍도;

도 9는 본 발명의 원리에 따라 Chien 블록을 Chien/Forney 블록과 함께 이용하는 리드-솔로몬 디코더의 예시적인 실시예를 나타내는 블록도;

도 10은 리드-솔로몬 디코더에서 실행된 종래 기술의 Chien 탐색 셀을 나타 내는 블록도; 그리고

도 11은 본 발명의 원리에 따라 메모리 요구 조건들과 지연이 감소된 리드-솔로몬 디코더에서 실행된 개선된 Chien 탐색 셀을 나타내는 블록도.

실시예

도 1을 참조하여, 본 발명에서 구현된 원리에 따른 에러 검출/정정 방식을 포함하는 디지털 데이터 전달 시스템(100)의 블록도를 설명한다. 일반적으로, 에러 검출/정정 방식들은, 디지털 데이터의 전송 또는 저장 시 에러가 데이터에 도입되지 않음을 보장하거나, 또는 에러가 데이터에 도입되더라도, 도입된 에러는 정정됨을 보장하는 것이 바람직한 경우에 이용된다. 데이터 에러들을 검출 및/또는 정정하는 능력은 리던던시를 데이터에 부가함으로써 이루어진다. 전송 또는 저장된 데이터에 리던던트 비트들을 포함시킴으로써, 본래의 코딩되지 않은 신호 또는 필드보다 코딩된 신호 또는 필드가 많은 비트들로 구성된다. 이러한 부가적인 오버헤드의 허용에 대한 보상은 에러들을 검출하거나, 또는 검출하여 정정하는 능력이다. 에러 제어 코딩을 이용하여 얻은 성능 개선은 코딩 이득의 관점에서 자주 측정된다. 코딩되지 않은 통신 시스템은, 주어진 비트 에러율(BER)을 30dB의 신호대 잡음비(SNR)에서 획득한다고 가정하자. 3dB의 코딩 이득을 갖는 에러 제어 코딩 방식이 시스템에 부가된다면, 코딩된 시스템은 BER을 27dB의 훨씬 낮은 SNR에서 획득할 수 있을 것이다. 선택적으로, 시스템이 30dB의 SNR에서 동작한다면, 코딩된 시스템에 의해 획득된 BER은, 코딩되지 않은 시스템이 33dB의 SNR에서 획득했던 BER과 동일할 것이다. 코딩 이득의 능력은, 통신 시스템이 (ⅰ) 코딩없이 가능했 던 SNR보다 더 낮은 SNR에서 바람직한 BER을 유지하거나, 또는 (ⅱ) 코딩되지 않은 시스템이 주어진 SNR에서 얻을 수 있었던 BER보다 더 높은 BER을 획득한다는 점이다.

예로서, 인코더(110)의 기능은, 디지털 데이터를 데이터 소스로부터 수신하고 데이터를 채널을 경유하여 전송하거나 또는 데이터를 저장 매체(채널 또는 저장 디바이스(115)처럼 집합적으로 도시함)에 저장하거나, 그렇지 않다면, 데이터를 조작하거나 처리한다. 전송 또는 저장 프로세스 동안에 데이터가 잡음 또는 에러(125)에 도입될 수도 있으며, 그 결과 본래의 디지털 데이터로부터의 폼이 손상 또는 변경되는 경우가 가끔 발생한다. 디코더(120)는, 디지털 데이터의 소정의 부분이 손상되었는지 아니면 손상되지 않았는지를 검출하여 정정하거나, 또는 선택적으로 단순히 검출한다.

도 2를 참조하여, 전송기/채널/수신기 환경내에서 이용가능한 다양한 에러 검출/정정 프로세스들의 흐름도를 설명한다. 그러한 환경에서 설명되겠지만, 에러 검출/정정 프로세스들은 디지털 데이터(데이터 필드, 패킷, 스트림 등의 폼)가 처리되거나 조작되는 방송 전송, 디지털 데이터 저장, 또는 임의의 다른 프로세스에 동일하게 또한 적용된다는 것은, 본 기술 분야의 당업자에게는 명백하다. 예로서, 단순한 실례이지만 소모적이거나 배타적임을 의미하지는 않는,

다음의 기술들/디바이스들은 에러 검출/정정 방식들을 이용하여 성능, 무결성, 및 신뢰성을 개선시킨다: (ⅰ) 테이프, 콤팩트 디스크(CD), 디지털 비디오 디스크(DVD), 바코드 등을 포함하지만 그에 한정되지는 않는 다양한 저장 디바이스들, (ⅱ) 무선 또는 이동 통 신(휴대 전화, 양방향 송수신기, 마이크로파 링크 등을 포함), (iii) 위성 통신, (iv) 디지털 라디오, 디지털 텔레비전(DTV), 디지털 비디오 방송(DVB) 등, (ⅴ) 케이블, V.pcm, ADSL, xDSL 등을 포함하지만 그에 한정되지는 않는 모뎀들.

단계 210에 따르면, 초기에 링크를 설립하고 전송 채널 파라미터들과 교섭한 후, 전송 소스는 디지털 데이터를 전송에 적합한 폼으로 처리한다. 단계 215에 따르면, 전송 이전에 소스는 에러 코드를 생성한다; 에러 코드는 전송될 디지털 데이터 값의 적어도 일부분에 기초하며, 따라서 어느 정도의 데이터 리던던시를 제공한다. 단계 220에 따르면, 생성된 에러 코드는 디지털 데이터와 함께 부가, 부착, 다중화되거나, 그렇지 않으면, 포함되어 전송기로부터 수신기로 전송된다. 단계 225에 따르면, 디지털 데이터와 에러 코드는 수신기에 수신된다. 단계 230에 따르면, 초기 신호 프로세싱이 요구된다면, 수신기에서 실행된다. 단계 235에 따르면, 수신기는 에러 코드의 리던던트 비트들을 액세스하고, 그것들에 포함된 정보를 이용중인 에러 제어 코드 방식에 따라 처리한다. 단계 240에 따르면, 처리된 리던던트 비트들이, 수신된 디지털 데이터에 대해서 긍정적으로 검사하면, 데이터는 손상되지 않은 것으로 추정된다. 단계 245에 따르면, 디지털 데이터의 다른 신호 프로세싱(만약 있다면)은 수신기에서 다시 시작된다.

그러나, 처리된 리던던트 비트들이, 수신된 디지털 데이터가 손상(적어도 하나의 비트 에러를 포함함)됨을 의미한다면, 단계 250에 따르면, 데이터 에러들은 이용중인 에러 제어 방식내에서 정정가능한 것인지를 결정하기 위하여 평가된다. 즉, 몇몇 에러 제어 방식들은 에러 검출만 가능한 반면, 그러한 에러들의 정정을 위해 허용하는 리던던트 데이터의 품질과 형태를 포함하지 않는다. 다른 에러 제어 방식들은 에러 검출과 정정 능력 모두를 갖고 있다는 사실에도 불구하고, 에러 검출 능력들만을 이용할 수도 있다. 이는, 임의의 특정 데이터 신호, 메시지, 또는 패킷의 정확성이 중요하지 않고 다소 일관적이며, 데이터의 시간적 지연이 중요할 때에 자주 이용된 방식이다. 그러한 응용의 예는 음성, 오디오, 및 비디오 애플리케이션들을 위한 동기 스트림 데이터이다. 부가적으로, 에러 정정 방식이 에러들을 검출하여 정정하기 위해 이용될 때, 검출된 에러들의 개수 또는 버스트가 에러 정정 능력보다 더 크면(즉, 에러 코드 비트들에 의해 제공된 리던던트 정보를 초과), 데이터는 간단하게 복원될 수 없다. 단계 255에 따르면, 데이터가 정정 가능하면, 데이터 에러들은 정정되고, 다른 신호 프로세싱(만약 있다면)은 수신기에서 다시 시작된다. 그러나, 에러들이 정정 불가능하면, 단계 260에 따르면, 데이터의 정확성이 필수적인지를 결정하기 위한 평가가 수행된다. 데이터 정확성이, 순수한 데이터 시스템들처럼 필수적이면, 단계 265에 따르면, 단일 비트 에러가 중요하다는 가능성은 커지고, 재전송 요구는 전송 소스로 다시 송신된다. 그러나, 동기 특징의 데이터 메시지들(예를 들면, 음성, 오디오, 또는 비디오)의 경우처럼, 정정 불가능한 데이터의 정확성이 필수적이지 않으면, 단계 270에 따르면, 손상되고 정정 불가능한 데이터는 간단하게 폐기되며, 그 다음 순서의 데이터 메시지가 처리된다.

도 3을 참조하여, 다양한 에러 코드 분류들과 방식들을 위한 계층도를 설명한다. 에러 코드들(310)들은 두 개의 기본적인 분류들로 분리될 수 있다: (ⅰ) 자 동 재전송신 요구(ARQ) 또는 검출 코드들(315), 및 (ⅱ) 전송(forward) 에러 정정(FEC) 코드들(320). ARQ는 검출만 하는 코딩의 형태이며, 전송시의 에러들은 수신기에 의해 검출될 수 있지만, 정정될 수는 없다. 수신기는 임의의 수신 데이터를 요구해야 하며, 에러들이 검출된 수신 데이터는 재전송되어야 함을 요청해야 한다. 이러한 재전송들은 유용성있는 대역폭을 도용할 것이기 때문에, ARQ 코드들은 일반적으로 '순수(clean)' 전송 매체들(더 낮은 에러 확률을 가짐)을 위해 이용된다. 가장 공통적인 예들 중 하나는 단순한 패리티 검사(325)이며, 이는 RAM에서의 데이터 에러들을 검출하기 위해 자주 이용된다. 또다른 예는 주기적인 리던던시 검사(CRC)이며, 이는, 예를 들어 이더넷(Ethernet)을 통한 전송시의 에러들을 검출하기 위해 이용된다. 에러들이 검출되면, 메시지가 재전송될 것이다. 이더넷은 유선을 통해 우선적으로 전송되기 때문에, 에러들이 발생할 기회는 몇몇 다른 매체들보다 더 낮다. CRC 및 ARQ는 에러 정정 코드 방식들을 설명하는 두 개의 단순한 예이다; 그리고 다른 에러 정정 코드 방식들은 본 기술 분야의 당업자에게 공지되어 있다. 단순히 에러들을 검출하고 정정하지 않는 에러 코드들은, 에러 정정 코드들보다 상당히 적은 리던던시를 부가한다. 더욱이, 에러 검출 디코더는 에러 정정 디코더보다 복잡하지 않다. 에러 검출 코드 방식들을 이용하는 시스템들은, 데이터 재전송을 위해 초래된 오버헤드에 관해서 일반적으로 대역폭이 허용된다. 즉, 데이터 재전송은 전체 시스템 출력에 큰 영향을 미치지 않는다.

잡음 매체는, 에러를 주어진 전송에 상당히 도입하기 때문에, ARQ 방법들을 이용한다는 것은 데이터의 끊임없는 재전송을 의미하며, 시스템 출력을 용인할 수 없는 레벨들로 감소시킨다. 이러한 경우들에 있어서, 에러 정정 코드는 그 이름이 의미하는 것처럼, 수신단에서의 에러들의 검출뿐만 아니라 에러들의 정정을 훌륭하게 수행한다. 이는 데이터 재전송에 대한 요구를 감소시키며, 데이터 재전송은, 에러들의 개수가 이용된 에러 정정 방법에 의해 정정될 수 있는 개수보다 더 많을 때에만 요구된다. 에러 정정은 단방향 통신을 위해 또한 이용되며, 이때, 수신기가 송신기에게 재전송을 요청할 기회는 이용할 수 없다. 이러한 단방향 경로들을 설명하는 예들은 몇몇 위성 전송들과 자기 테이프 저장 매체들을 포함한다.

에러 정정 코드들은 두 개의 주요 하위 범주들로 분리될 수도 있다. 첫 번째는 블록 코드들(335)이고 두 번째는 콘볼루션 코드들(340)이다. 블록 코드들(335)은, 한정된 유한 길이의 메시지 블록들로 동작하는 에러 정정 코드들로서 자주 이용된다. 블록 코드들(335)로 명명된 이유는, 하위 범주가 메시지 심볼들의 블록을 처리하여 코드 워드 심볼들의 블록을 출력하는 인코더를 이용하기 때문이다. 블록 코드들은 일반적으로 두 개의 형태로 분류될 수 있다; 이진 코드(

345) 및 비(非)이진 코드(355). 이진 코드(345)의 예는 해밍 코드(350)이며, 문자 당 네 개의 정보 비트들과 세 개의 검사 비트들을 구비함을 특징으로 한다. 비(非)이진 코드의 예는 리드-솔로몬 코드(360)이다.

대조적으로, 콘볼루션 코드(340) 인코더들은 메시지 심볼들의 연속적인 스트림으로 동작하며, 동시에 연속적으로 인코딩된 출력 스트림을 생성한다. 콘볼루션 코드들로 명명된 이유는, 인코딩 프로세스가 메시지 심볼들과 인코더의 임펄스 응답의 콘볼루션처럼 보일 수 있기 때문이다. 콘볼루션 코드들(340)의 두 가지 예는 격자 코딩 변조(TCM)(365) 및 이진 콘볼루션 코딩(370)이다.

도 4는 예시적인 리드-솔로몬 디코더를 나타내는 블록도이다. 전술한 바와 같이, 리드-솔로몬 코드들은 비(非)이진 계통의 주기적인 선형 블록 코드들이다. 비(非)이진 코드들은 몇몇 비트들로 구성되는 심볼들로 동작한다. 비(非)이진 코드들에 대한 공통적인 심볼 크기는 8 비트, 또는 1 바이트이다. 리드-솔로몬 코드와 같은 비(非)이진 코드는 버스트 에러들의 정정에 적합한데, 이러한 코드들에 의한 정정이 심볼 레벨에서 이루어지기 때문이다. 디코딩 프로세스에서 심볼들로 동작함으로써, 이러한 코드들은 여덟 개 에러들의 버스트를 갖는 심볼을, 단일 비트 에러를 갖는 심볼을 정정할 수 있는 것처럼 쉽게 정정할 수 있다. 리드-솔로몬 코드와 같은 계통 코드는, 메시지 심볼들을 변경되지 않은 폼에 포함하는 코드 워드들을 생성한다. 인코더는, 리던던시, 또는 패리티, 심볼들을 생성하기 위하여 가역성의 수학 함수를 메시지 심볼들에 적용한다. 코드 워드는 패리티 심볼들을 메시지 심볼들에 부가함으로써 형성된다. 리드-솔로몬 코드는 주기적인 코드로 여겨지는데, 임의의 유효 코드 워드의 주기적인 시프트가 다른 유효 코드 워드를 또한 제공하기 때문이다. 주기적인 코드들이 대중적인데, 그 코드들을 실행하기 위한 효율적이며 저렴한 디코딩 기술들이 존재하기 때문이다. 마지막으로, 리드-솔로몬 코드는 선형으로 여겨지는데, 어떤 두 개의 유효 코드 워드들을 합하면 다른 유효 코드 워드가 되기 때문이다.

에러 제어 코드들의 이론은, 유한 필드들 또는 Galois 필드들로 알려진 수학적 구조를 이용한다. Galois 필드는 구성 요소들의 유한 개수를 포함하는 세트이 다. 이러한 세트에서 덧셈과 곱셈의 동작들이 정의되고, 그 동작들은 정상적인 연산으로부터 예상되는 것처럼 행동한다. 예를 들면, 덧셈 식별 구성 요소는 0이고, 곱셈 식별 구성 요소는 1이다. 리드-솔로몬 코드는, 코드의 주기적이며 선형적인 특징을 수행할 책임이 있는 Galois 필드 수학을 이용하여 실행되고, $q=p^m$ 순서의 Galois 필드들에서 동작하며, 이때, p는 소수인 양의 정수이며, m은 양의 정수이다. q 순서의 Galois 필드는 GF(q)에 의해 표시되며, q개의 별개의 구성 요소들을 포함한다.

주어진 리드-솔로몬 코드는 (n,k) 코드로 정의됨으로써 표시된다. 파라미터 n은 코드 워드내 심볼들의 개수에 관한 코드 워드 길이를 나타낸다. 파라미터 k는 코드 워드내 메시지 심볼들의 개수를 나타낸다. 부가된 패리티 심볼들의 개수는 n-k 이다. 코드의 에러 정정 능력은 t=(n-k)/2 이다. 코드는 T개의 에러들을 검출하여 정정할 수 있으며, 이때, T는 $0 \leq T \leq t$ 이다. 코드 워드는 메시지 심볼들에 기초하며, 리드-솔로몬 인코더를 이용하여 생성된다. 리드-솔로몬은 계통 코드이기 때문에, n개의 메시지 심볼들은 계통 코드처럼 전송되며, n-k 패리티 심볼들은 메시지 심볼들에 부가되어 코드 워드를 형성한다. 리던던시를 전송된 코드 워드에 부가하는 패리티 심볼들의 값들은 메시지 심볼들에 의존한다. 이러한 리던던시는 수신기의 디코더에 의해 이용되어 에러들을 검출하고 정정한다.

수신기의 디코더에서, 코드 워드들은 신드롬 생성 블록(410)에 입력으로서 수신된다. 디코더에 의해 수행된 첫 번째 단계는 신드롬 계산이며, 신드롬 생성 블록(410)(신드롬 생성 모듈, 또는 간단하게 신드롬 생성기로 또한 정의되기도 함)에 의해 수행된다. 신드롬은 n-k 심볼들로 구성되며, 그 값들은 수신된 코드 워드로부터 계산된다. 신드롬은 에러 벡터에만 의존하며, 전송된 코드 워드와는 무관하다. 즉, 각각의 에러 벡터는 유일한 신드롬 벡터를 갖지만, 서로 다른 수많은 수신 코드 워드들은, 에러 패턴이 동일하면 동일한 신드롬을 가질 것이다. 신드롬이 우선적으로 계산되는 이유는, 신드롬을 계산함으로써 에러 벡터에 대한 탐색 필드를 한정하기 때문이다. 신드롬을 우선적으로 인식함으로써, 적합한 에러 벡터들의 개수를 2^n 의 전체 가능한 에러 벡터들로부터 2^{n-k} 의 에러 벡터들로 그 범위를 한정할 것이다.

신드롬 생성 블록(410)이 신드롬을 계산하는 하나의 방법은, 수신 코드 워드를 Galois 필드 대수학을 이용하여 생성기 다항식으로 나누는 것이다. 이러한 나눗셈의 나머지는 신드롬 다항식 s(x)로 불린다. 실제 신드롬 벡터 S(x)는 α 부터 α^{n-k} 까지의 s(x)를 구함으로써 계산된다. 그러나, 이러한 방법은 하드웨어 관점에서 최적의 효과가 있는 것은 아니며, 하드웨어에서 자주 이용되는 다른 방법은 α부터 α^{n-k} 까지의 수신 코드 워드 R(x)를 직접 구하는 것이다. 신드롬 생성 블록(410)은, α부터 α^{n-k} 까지의 수신 코드 워드, 즉 $R(\alpha)$부터 $R(\alpha^{n-k})$를 구함으로써 신드롬 S를 계산한다. 리드-솔로몬 코드에 있어서, n-k=2t 이며, 따라서 계산하기 위한 2t 신드롬 값들이 존재한다: [S1 S2 S3... S(2t)]. 이러한 값들은 전통 적으로 병렬로 계산되고, 첫 번째 신드롬 생성기는 α에서의 수신 코드 워드를 구하여 S1을 형성하며, 그 다음의 신드롬 생성기는 α^2 에서의 수신 코드 워드를 구하여 S2, 등을 형성한다.

신드롬이 신드롬 생성 블록(410)에 의해 계산되면, 그 값은 에러 다항식 블록(420)으로 전달된다. 그곳에서, 신드롬은 에러 위치 다항식을 생성하기 위해 이용된다. 이러한 프로세스는 알려지지 않은 t에 대한 연립 방정식들을 해결하는 것을 포함한다. 몇몇 고속 알고리즘들이 이러한 계산들에 유용한데, 그러한 알고리즘들은 Berlekamp-Massey 알고리즘 또는 Euclid의 알고리즘을 포함한다. 이러한 알고리즘들은 리드-솔로몬 코드들의 특정 매트릭스 구조를 이용하며, 요구되는 계산 노력을 크게 감소시킨다.

에러 다항식 블록(420)은 에러 위치 다항식(결정된다면)을 에러 위치 블록(430)과 에러 크기 블록(440)으로 전달한다. 에러 위치 블록(430)은 에러 위치 다항식의 근(root)들이 에러 위치를 결정하는 것을 해결한다. 전통적으로, 이는 Chien 탐색 알고리즘, 또는 Chien 셀을 이용하여 이루어진다. 에러 위치 블록(430)에 의해 결정된 에러 위치들은 이전에 결정된 에러 위치 다항식에 따라 에러 크기 블록(440)으로 전달된다. 에러 크기 블록(440)은 알려지지 않는 t에 대한 연립 방정식들을 해결함으로써 에러 크기를 결정한다. 에러 크기 블록(440)을 실현하는데 이용된 고속이면서 널리 사용되는 알고리즘은 Forney 알고리즘이다.

계산된 에러 위치와 에러 크기는, 정정된 코드 워드에 오류가 발생되면, 정 정된 코드 워드를 복원하기 위해 에러 정정 블록(450)으로 전송된다. 흔히, 에러 위치와 에러 크기의 결합은 에러 벡터로 불린다. 에러 벡터는 코드 워드처럼 동일한 크기이며, 0이 아닌 값들을 에러들에 대응하는 위치들에 포함한다. 모든 다른 위치들은 0을 포함한다. 에러 정정 블록(450)으로의 또다른 입력은 지연 블록(460)의 출력이다. 지연 블록(460)은 수신 코드 워드를 입력으로 하여, 지연을 갖는 동일한 수신 코드 워드를 출력한다. 하나의 실시예에 있어서, 에러 정정 블록(450)은 LIFO(Last In, First Out) 블록(454)과 함께 Galois 필드 덧셈기(452)를 이용하여 구현된다. 수신 코드 워드내 에러들은 수신 코드 워드를 Galois 필드 덧셈기(452)를 이용하는 에러 벡터에 더함으로써 정정된다. LIFO 블록(454)은, 에러 벡터가 수신 코드 워드의 역순으로 생성되기 때문에 이용되며, 그러므로 LIFO 동작은, 양쪽 벡터들내 바이트들의 순서를 조화시키기 위하여 수신 코드 워드 또는 에러 벡터에 적용되어야 한다. 에러 정정 블록(450)의 출력은 본래의 코드 워드에 대한 디코더의 추정이다.

도 5는 리드-솔로몬 디코더의 예시적인 실시예를 나타내는 블록도이다. 입력은 신드롬 생성 블록(510)으로 전송되는 수신 코드 워드이다. 신드롬 S(x)가 신드롬 생성 블록(510)에 의해 계산되면, 그 값은 Euclid의 알고리즘 블록(520)으로 전달된다. Euclid의 알고리즘은, 에러 위치 다항식 Λ(x)와 에러 크기 다항식 Ω(x)를 생성하기 위하여 신드롬 S(x)를 처리한다. 즉, Key 방정식으로 불리는 다음의 수학식을 해결한다:

수학식 1

$$\Lambda(x)\,[1 + S(x)] = \Omega(x) \bmod x^{2t+1}$$

리드-솔로몬 디코딩에 이용된 알고리즘은, 두 개 다항식들의 최대 공약수(GCD)를 찾기 위한 Euclid의 알고리즘에 기초한다. Euclid의 알고리즘은 공지된 반복적인 다항식 분리 알고리즘이다.

에러 위치 다항식 Λ(x)가 계산되면, 그들의 근들을 찾기 위해 계산될 필요가 있다. Chien 탐색 알고리즘은 이러한 근들을 찾기 위해 이용된다. Chien 탐색은, 모든 가능한 입력 값들에 대한 다항식을 계산하는 맹목적인 강제적 알고리즘이며, 출력들이 0과 동일한 것인지를 결정한다. 에러가 위치 i에서 발생하면, 다음의 수학식은 0과 동일하다:

수학식 2

$$\sum_{j=0}^{t} \Lambda_j \alpha^{-ij} = 0, \quad \text{where } i = 0 \ldots (n-1)$$

Chien 탐색은, i 및 j의 모든 값들에 대한 수학식 2를 계산하며, 수학식이 0과 동일해지는 시점들의 개수를 산출한다. 0들의 위치는 에러 위치들이며, 0들의 개수는 에러가 있는 심볼들의 개수이다.

예시적인 실시예에 있어서, 하드웨어에서 구현되는 Chien 탐색의 (t+1) 단계들이 존재한다. 이러한 단계들의 각각은 (단계는 곱셈기, 멀티플렉서 및 레지스터로 구성됨) 상술한 Chien 탐색 수학식내 j에 대한 서로 다른 값을 나타낸다. 탐색은 n 클럭 주기들(각각의 클럭 주기는 상술한 수학식내 i의 서로 다른 값을 나타 냄) 동안에 작동하며, 덧셈기의 출력은 0과 동일한 것인지를 알기 위해 조사된다. 0과 동일하면, 0 검출 블록은 1을 출력할 것이며, 그렇지 않다면, 0을 출력할 것이다. Chien 탐색 블록의 출력은, '0' 또는 '1'의 값들을 갖는 n 비트들의 문자열이다. 각각의 '1'은 에러내 심볼의 위치를 나타낸다.

첫 번째 클러 주기 동안에, 멀티플렉서는 에러 위치 다항식 계수를 레지스터로 보낸다. 나머지 (n-1) 클럭 주기들 동안에, 곱셈기의 출력은 멀티플렉서를 경유하여 레지스터로 보내진다. 곱셈기들의 지수들은 음의 값들을 갖는다. 그러나, 이러한 값들은 모듈로(modulo) 연산자를 이용하여 사전에 계산될 수 있다. α^{-i} 의 지수는 (-i modulo n) = (-i modulo 255) 와 동일하다. 예를 들면, α^{-1} 은 α^{254} 와 동일하고 α^{-2} 는 α^{253} 와 동일하다.

Forney 알고리즘은 에러 값들 Y ¡를 계산하기 위하여 이용된다. 이러한 값들을 계산하기 위하여, Forney 알고리즘은 에러 위치 다항식 Λ(x) 및 에러 크기 다항식 Ω(x)를 이용한다. 에러 값들에 대한 수학식은 다음과 같다.

수학식 3

$$Y_i = \frac{\Omega(x)}{\Lambda'(x)} \quad \text{for } x = \alpha^{-i} \text{ where } \alpha^{-i} \text{ is a root of } \Lambda(x)$$

수학적 형식의 미분 Λ'(x)의 계산은 실질적으로 매우 단순하다. 예를 들어 Λ(x) = α ^{4}X ^{3} + α ^{3}X ^{2} + αX + α ^{2} 이라고 가정하자. 따라서, Λ'(x)는 다음과 같 다.

수학식 4

$$\begin{aligned}\Lambda'(x) &= 3\alpha^4X^2 + 2\alpha^3X + \alpha \\ &= (\alpha^4 + \alpha^4 + \alpha^4)\,X^2 + (\alpha^3 + \alpha^3)X + \alpha \\ &= \alpha^4 X^2 + \alpha\end{aligned}$$

미분은 X의 홀수 승들의 계수들을 획득하여, 그 계수들을 X의 그 다음으로 낮은 승(짝수일 것임)에 할당함으로써 형성된다.

Ω(x) 다항식은, Chien 탐색을 위해 이용된 것과 동일한 형태의 하드웨어를 이용하여 Λ'(x) 다항식과 함께 계산된다. Ω(x)를 계산하기 위하여, Ω $_{0}$ 계수는, Ω $_{1}$ 계수 × α $^{-1}$, Ω $_{2}$ 계수 × α $^{-2}$,,,Ω $_{t}$ 계수 × α $^{-t}$ 와 더해진다. 이러한 곱셈기들의 출력은 합해진다.

분자(numerator)는 역 곱셈을 이용하는 분모에 의해 곱해진다. 역 곱셈은 분모의 역을 찾는 룩업 테이블을 포함한다. 예를 들어, 분모가 α ^{3} 이면, 그 역은 α $^{-3}$ 이다. 이는 다음과 같이 표현될 수 있다.

수학식 5

$$\alpha^{-i} = \alpha^{(-i \bmod n)} = \alpha^{(-3 \bmod 255)} = \alpha^{252}$$

동일한 형태의 하드웨어가 Chien 탐색과 Forney 알고리즘을 위해 요구되기 때문에, 두 가지 기능들은, Chien/Forney 블록(530)으로 도시한 것처럼 동일한 블록에 포함될 수 있다. 이러한 구현에 있어서, 두 개의 덧셈기들은 Chien 탐색의 출력에 이용된다. 첫 번째 덧셈기는 짝수 단계들에 대한 값들을 합하고, 다른 덧셈기는 홀수 단계들에 대한 값들을 합한다. 최종 Chien 탐색 출력을 형성하기 위하여, 이러한 두 개 덧셈기들의 출력들은 합해지고, 0 검출 블록은 근들의 위치들을 검출한다. 홀수 단계들에 대한 덧셈기의 출력은 Forney 알고리즘에 또한 이용된다. 홀수 단계들의 합은 Forney 방정식의 분모를 나타낸다. 이와 같이 합해진 값은 반전되고, 에러 크기 다항식의 계산으로부터 형성된 분자 값에 의해 곱해진다. 출력은 0 검출 출력과 AND되는데, 에러 값들은 실질적인 에러 위치들에 대해서만 유효하기 때문이다(그렇지 않다면, 에러 값들은 0으로 세팅됨).

따라서, Chien/Forney 블록(530)은 에러 위치 다항식과 에러 크기 다항식을 이용하여 에러 벡터를 생성하며, 에러 벡터는 에러 정정 블록(540)으로 전송된다. 에러 벡터는 코드 워드처럼 동일한 크기이며, 0이 아닌 값들을 에러들에 대응하는 위치들에 포함한다. 모든 다른 위치들은 0을 포함한다. 에러 정정 블록(540)으로의 또다른 입력은 지연 블록(550)의 출력이다. 지연 블록(550)은 수신 코드 워드를 입력으로 하여, 지연을 갖는 동일한 수신 코드 워드를 출력한다. 실시예에 있어서, 에러 정정 블록(540)은 LIFO 블록(542)과 함께 Galois 필드 덧셈기(544)를 이용하여 구현된다. 수신 코드 워드내 에러들은 수신 코드 워드를 Galois 필드 덧셈기(544)를 이용하는 에러 벡터에 더함으로써 정정된다. LIFO 블록(542)은, 에러 벡터가 수신 코드 워드의 역순으로 생성되기 때문에 이용되며, 그러므로 LIFO 동작은, 양쪽 벡터들에 대한 바이트들의 순서를 동기화시키기 위하여 수신 코드 워드 또는 에러 벡터에 적용되어야 한다. 에러 정정 블록(540)의 출력은 본래의 코드 워드에 대한 디코더의 추정이다.

인트라-디코더 컴포넌트 블록 메시징

전술한 바와 같이, 리드-솔로몬 디코더는 적어도 네 개의 중요한 컴포넌트들, 또는 기능 블록들을 구비한다. 그 블록들은, 신드롬 생성기, 에러 다항식 블록, 에러 위치 블록, 및 에러 크기 블록이다. 디코더가 에러들을 검출하는 것과 함께 에러들을 정정한다면, 디코더는 에러 정정 블록을 또한 포함한다. 종래 기술의 디코더들에 있어서, 이러한 블록들은, 하나의 블록으로의 입력은 이전 블록의 출력에만 의존하기 때문에, '전달 파이프라인'을 형성한다. 즉, 하나의 블록으로부터 이전 블록으로의 피드백은 존재하지 않는다. 리드-솔로몬 디코더는 메모리 또는 버퍼 블록의 구현을 요구하여 수신 코드 워드를 저장하는 반면, 디코딩 프로세스는 특정 코드 워드(도 4 및 5의 지연 블록으로 표시함)에 대해서 수행된다. 에러 위치/크기 블록들은 디코딩 프로세스의 완료 시점에서 에러 벡터를 제공하며, 이러한 에러 벡터는, 디코더 출력을 형성하기 위하여 메모리에 저장된 수신 코드 워드와 함께 XOR된다. 전통적으로, 이용된 메모리의 크기는 하나의 코드 워드와 동일하며, 따라서 디코더는 한번에 단일 코드 워드만을 처리할 수 있다.

도 6은 본 발명의 원리에 따른 내부 디코더 핸드세이킹 프로토콜의 예시적인 실시예를 설명하는 블록도이다. 인트라-디코더 블록 A(610)과 인트라-디코더 블록 B(620)는, 블록들간의 피드백 또는 통신을 이용하는 리드-솔로몬 디코더 내 임의의 두 개 블록들(상술한 기능 블록들로부터 선택됨)을 나타낸다. 디코딩 '전달 파이프라인'은 데이터 채널(630)이며, 데이터 채널은 리드-솔로몬 디코더의 기능 블록 들간의 임의의 종래 기술의 단방향 채널을 나타낸다. 부가적으로 그리고 본 발명에 따르면, 피드백 채널들을 또한 도시한다. 피드백 채널들은, 하나의 기능 블록이 자신의 현재 또는 미래의 비활동을 상향 또는 하향 기능 블록으로 전달하는 수단으로 생각될 수도 있다. 예를 들어, 인트라-디코더 블록 A(610)는 특정 코드 워드에 대한 프로세싱 기능을 완료한다고 가정하자. 인트라-디코더 블록 A(610)는 비활동 메시지를 인트라-디코더 블록 B(620)로 개시하는데, 이는 특정 코드 워드에 대한 자신의 계산 함수의 결과를 인트라-디코더 블록 B(620)로 전송할 준비가 됨을 나타낸다. 따라서, 이러한 형태의 비활동 메시지는 '송신 준비(Ready-to-Send)'(RTS) 메시지(650)로 불린다. 역으로, 인트라-디코더 블록 B(620)는 특정 코드 워드에 대한 자신의 프로세싱 기능을 완료한다고 가정하자. 인트라-디코더 블록 B(620)는 비활동 메시지를 인트라-디코더 블록 A(610)로 개시하는데, 이는 수신할 준비가 됨을 나타낸다. 따라서, 이러한 형태의 비활동 메시지는 '수신 준비(Ready-to-Receive)'(RTR) 메시지(640)로 불린다.

본 발명의 이러한 실시예에 있어서, 기능 블록들간의 피드백 또는 핸드세이킹을 구비한 디코더 구현은, 이용자가, 디코더를 구현하기 위해 이용하고자 하는 메모리의 크기에 기초한 디코드의 성능을 만들 수 있는 지적재산권(IP) 코어(core)에 적합한 이용자 구성가능한 구조를 가능하게 한다는 장점이 있다.

도 7은 리드-솔로몬 디코더의 예시적인 실시예의 기능 블록들간의 헨드세이킹 프로토콜을 설명하는 블록도이다. 신드롬 생성 블록(410), 에러 다항식 블록(420), 에러 위치 블록(430), 에러 크기 블록(440), 에러 정정 블록(450)(Galois 필드 덧셈기(452) 및 LIFO 블록(454) 포함), 및 지연 블록(460)의 기능, 목적, 및 동작은 도 4를 참조하여 충분하게 설명되며, 그러므로 이하에서는 반복하지 않는다. 전술한 비활성 메시지들(또는 헨드세이킹 신호들)이 도 7에 새롭게 포함되어 나타낸다. 핸드세이킹 신호들(예를 들면, RTS 및 RTR 메시지들)은 데이터를 송신할 때 및/또는 데이터를 수신할 준비가 될 때를 나타내기 위해 프로세싱 블록들에 의해 이용된다. 그러므로, 신드롬 생성 블록(410)과 에러 다항식 블록(420)은 RTS 메시지를 RTS 채널(710)을 통해 교환하거나, RTR 메시지를 RTR 채널(712)을 통해 교환할 수 있다. 에러 다항식 블록(420)과 에러 위치 블록(430)은 RTS 메시지를 RTS 채널(718)을 통해 교환하거나, RTR 메시지를 RTR 채널(720)을 통해 교환할 수 있다. 에러 다항식 블록(420)과 에러 크기 블록(440)은 RTS 메시지를 RTS 채널(724)을 통해 교환하거나, RTR 메시지를 RTR 채널(726)을 통해 교환할 수 있다. 에러 크기 블록(440)과 에러 정정 블록(450)은 RTS 메시지를 RTS 채널(730)을 통해 교환하거나, RTR 메시지를 RTR 채널(760)을 통해 교환할 수 있다. 신드롬 생성 블록(410)은 RTR 메시지를 RTR 채널(740)을 통해 송신함으로써, 다음 수신 코드 워드를 요청하기 위해 또한 적응된다. 에러 정정 블록(450)은 RTS 메시지를 RTS 채널(760)을 통해 출력 프로세싱 단계로 송신하기 위해 또한 적응된다. 부가적으로, 본 발명의 실시예에 따르면, 지연을 적절하게 조절하여, 검사 및 정정되는 수신 코드 워드가 에러 벡터와 에러 정정 블록(450)에서 적당하게 동기화되기 위하여, 임의의 RTR 또는 RTS 메시지를 지연 블록(460)(메시지 채널 (750)을 통해)으로 송신할 필요가 있을 수도 있다.

본 발명으로부터 유래된 이로운 유익성은, 인트라-디코더 기능 블록들간의 핸드세이킹을 가능하게 함으로써, 기능 블록이 비활성일 때(즉, 기능 블록의 마지막 절차가 완료됨), 더 많은 데이터의 수신을 처리하도록 요청할 수 있다는 점이다. 유사하게, 기능 블록은 그 절차를 완료한 하향 기능 블록으로 신호를 보낼 수 있으며, 하향 기능 블록이 그 결과를 수신할 수 있으면 언제든지 결과를 전송할 수 있다. 예를 들어, 종래의 신드롬 생성 블록(410)은 클럭 주기들의 고정된(그리고 공지된) 개수를 이용하여 특정 길이의 수신 코드 워드에 대한 신드롬을 생성하지만, 에러 다항식 블록(420)은 실행 시간이 변할 수 있는 반복적인 프로세스이다. 더욱이, 종래의 에러 위치 블록(430)과 에러 크기 블록(440)은 클럭 주기들의 고정된 (공지된) 개수를 획득할 것이지만, 너무 많은 에러들이 코드 워드내에 존재하여 정정 불가능하다고 결정되면, 다른 프로세싱은 그 코드 워드에 관해서 중단될 수도 있으며, 에러 위치 블록(430)과 에러 크기 블록(440)은 RTR 메시지를 에러 다항식 블록(420)으로 적절한 RTR 채널들(720,726)을 통해 송신함으로써, 다음의 코드 워드에 관한 동작을 시작하도록 요청할 수 있다. 그러므로, 에러 위치 블록(430)과 에러 크기 블록(440)은 다양한 개수의 클럭 주기들을 또한 이용하여 완료할 수 있다.

도 7에 도시한 본 발명의 실시예는 본 발명의 사상과 범위를 벗어나지 않으면서 구현될 수도 있는 몇몇 실시예들 중 하나임을 주목해야 한다. 예를 들면, 디코더의 각 기능 블록들이 RTS 및 RTR 메시지들을 송신하기 위해 구비되는 것은 요구되지 않는다. 오히려, 설계 선택의 문제로서, 신드롬 생성 블록(410)과 에러 다 항식 블록(420)들 사이에서만 핸드세이킹을 가능하게 하는 것이 바람직할 수도 있다. 수많은 다른 변화들이 또한 가능하다. 더욱이, RTS 및 RTR 메시지들은 RTS 및 RTR 채널들을 통해 통신되는 것처럼 설명되지만, 그러한 설명은 단순한 논리적 설명이며, 메시징 채널들은 하나의 공통적인 물리 계층 메시징 채널을 통해 데이터 채널로부터 통합될 수도 있거나, 또는 선택적으로, 메시징 채널들은 동일한 물리 계층을 통해 데이터 채널 그 자체로서 통합될 수도 있다.

본 발명의 원리에 따르면, 이용자는 디코더의 성능을 메모리 할당 크기를 변경함으로써 구성할 수 있다. 그러므로, 하나의 코드 워드의 길이와 동일한 할당 메모리 크기를 갖는 실시예는 하드웨어 및 전력 효율적인 설계를 구비할 것이지만, 한번에 하나의 코드 워드만을 처리할 수 있을 것이다. 하나의 코드 워드 길이보다 더 긴 할당 메모리 크기를 갖는 실시예는 하나의 코드 워드를 처리할 수 있는 반면, 두 번째 코드 워드를 로드(load)하기 시작한다. 두 개 코드 워드들의 할당 메모리 크기를 갖는 실시예는 두 개 코드 워드들이 동시에 처리되게 할 수 있다. 물론, 통합된 메모리의 크기를 증가시키는 것은 대응하는 하드웨어 및 전력의 관점에서 더 많은 비용을 요구하지만, 그 장점은 디코딩 프로세스 속도에서의 증가 및 대기 시간(latency)에서의 감소인데, 수많은 개수의 코드 워드들이 주어진 시간내에 처리될 수 있기 때문이다. 핸드세이킹 신호들이 프로세스를 완전하게 자동화하여, 이용자는 디코더에 의해 이용된 메모리 크기만을 입력한다. 이러한 단일 파라미터를 제어함으로써, 이용자는 속도, 전력 및 크기(게이트 수)의 관점에서 디코더의 성능을 구성할 수 있다.

도 8은 본 발명의 원리에 따른 인트라 블록 핸드세이킹을 이용하는 디코더와 관련된 효율들을 입증하는 예시적인 리드-솔로몬 디코더들을 위한 타이밍도이다. 상부 타이밍도(810)는 인트라-블록 핸드세이킹이 없는 전통적인 디코더내 프로세싱을 위해 요구되는 시간을 나타낸다. 이러한 방식하에서, 코드 워드 1(CW1)의 프로세싱은 시간 t_0 에서 시작한다. CW1 신드롬은 시간 t_1 에서 생성되며, 그 결과는 에러 다항식 블록으로 전달된다. 에러 다항식 블록은, CW1이 시간 t_4 에서 완전하게 처리될 때까지, 프로세싱 CW1을 시간 t_2 등에서 완료한다. 그러므로, 코드 워드 2(CW2)의 프로세싱은 시간 t_4 에서 시작한다. 이러한 단일 함수적인 순차 프로세싱은, 시간 t_8 에서 CW2에 대한 에러 위치와 에러 크기 계산들이 완료될 때까지 CW2에 대해서 연속적이다. 전체적으로 (t_8 -t_0) 단위의 시간이 인트라-블록 핸드세이킹 없이 두 개의 코드 워드들을 처리하기 위해 필요함을 주목하자.

하부 타이밍도(820)는 인트라-블록 핸드세이킹이 있는 디코더내 프로세싱을 나타낸다. 세 개의 코드 워드들을 위해 충분한 크기의 메모리를 구현함으로써, 디코더는 한번에 세 개의 코드 워드들을 동시에 처리할 수 있다. 이러한 효율은, 각각의 블록이 이전 블록에 자신이 준비됨을 통지하여, 각각의 블록이 프로세싱을 종료하자마자 많은 데이터를 수신할 수 있기 때문에 획득된다. 이러한 예에 있어서, 코드 워드 3(CW3)의 프로세싱은 때때로 시간 t_6 와 t_7 사이에서 완료된다.

에러-정정 디코더내 이중 Chien 탐색 블록들

본 발명의 하나의 예시적인 실시예는 이중 Chien 탐색 블록들을 이용하여 에러 정정 프로세스를 구현한다. 본 발명의 이러한 실시예의 원리를 이용하는 디코더는 디코더 대기 시간과 디코더 저장/메모리 요구 조건들을 감소시킨다.

도 9는 본 발명의 원리에 따라 이중 Chien 탐색 블록들을 이용하는 리드-솔로몬 디코더의 예시적인 실시예를 나타내는 블록도이다. 입력은 수신된 코드 워드이며, 이는 신드롬 생성 블록(910)으로 전송된다. 신드롬 S(x)가 신드롬 생성 블록(910)에 의해 계산되면, 신드롬은 Euclid의 알고리즘 블록(920)으로 전송된다. Euclid의 알고리즘은, 에러 위치 다항식 $\Lambda(x)$ 및 에러 크기 다항식 $\Omega(x)$를 생성하기 위하여 신드롬을 처리하는데 이용된다.

본 발명의 실시예에 있어서, 새로운 Chien 블록(930)은 코드 워드내 에러들의 개수를 결정하기 위하여 통합된다. 에러 위치 다항식 $\Lambda(x)$이 계산되면, Chien 블록(930)은 Chien 탐색 알고리즘을 적용하여 근들을 평가한다. Chien 탐색은 모든 가능한 입력 값들에 대한 다항식을 평가하는 맹목적인 강제적 알고리즘이며, 어떤 출력들이 0과 동일한 것인지를 결정한다. 에러가 위치 i에서 발생하면, 다음의 수학식은 0과 동일하다:

수학식 6

$$\sum_{j=0}^{t} \Lambda_j \alpha^{-ij} = 0, \qquad \text{where } i = 0 \,..\, (n\text{-}1)$$

Chien 블록(930)은 i 및 j의 모든 값들에 대해 상술한 수학식을 계산하고, 수학식이 0과 동일해지는 시점의 개수를 산출한다. 결과 개수는 검출된 에러들의 개수이 다.

전술한 바와 같이, 에러 다항식은 에러 위치와 에러 크기 양쪽에 대해 계산되어야 한다. 동일한 형태의 하드웨어가 Chien 탐색과 Forney 알고리즘을 위해 요구되기 때문에, 두 가지 기능들은, Chien/Forney 블록(940)으로 도시한 것처럼 동일한 블록에 포함될 수 있다. 이러한 구현에 있어서, 두 개의 덧셈기들은 Chien 탐색의 출력에 이용된다. 첫 번째 덧셈기는 짝수 단계들에 대한 값들을 합하고, 다른 덧셈기는 홀수 단계들에 대한 값들을 합한다. 최종 Chien 탐색 출력을 형성하기 위하여, 이러한 두 개 덧셈기들의 출력들은 합해지고, 0 검출 블록은 근들의 위치들을 검출한다. 홀수 단계들에 대한 덧셈기의 출력은 Forney 알고리즘에 또한 이용된다. 홀수 단계들의 합은 Forney 방정식의 분모를 나타낸다. 이와 같이 합해진 값은 반전되고, 에러 크기 다항식의 계산으로부터 형성된 분자 값에 의해 곱해진다. 출력은 0 검출 출력과 AND되는데, 에러 값들은 실질적인 에러 위치들에 대해서만 유효하기 때문이다(그렇지 않다면, 에러 값들은 0으로 세팅됨).

따라서, Chien/Forney 블록(940)은 에러 위치 다항식과 에러 크기 다항식을 이용하여 에러 벡터를 생성하며, 에러 벡터는 에러 정정 블록(950)으로 전송된다. 에러 벡터는 코드 워드처럼 동일한 크기이며, 0이 아닌 값들을 에러들에 대응하는 위치들에 포함한다. 모든 다른 위치들은 0을 포함한다. 에러 정정 블록(950)으로의 또다른 입력은 지연 블록(960)의 출력이다. 지연 블록(960)은 수신 코드 워드를 입력으로 하여, 지연을 갖는 동일한 수신 코드 워드를 출력한다. 실시예에 있어서, 에러 정정 블록(950)은 LIFO 블록(952)과 함께 Galois 필드 덧셈기(954)를 이용하여 구현된다. 수신 코드 워드내 에러들은 수신 코드 워드를 Galois 필드 덧셈기(954)를 이용하는 에러 벡터에 더함으로써 정정된다. LIFO 블록(952)은, 에러 벡터가 수신 코드 워드의 역순으로 생성되기 때문에 이용되며, 그러므로 LIFO 동작은, 양쪽 벡터들내 바이트들의 순서를 조화시키기 위하여 수신 코드 워드 또는 에러 벡터에 적용되어야 한다. 에러 정정 블록(950)의 출력은 본래의 코드 워드에 대한 디코더의 추정이다.

리드-솔로몬 디코더는 t개의 에러들을 정정할 수 있으며, 이때, 포함된 패리티 비트들의 개수는 2t이다. t보다 많은 에러가 검출된다면, 수신 코드 워드를 변경되지 않고 정정되지 않은 디코더 출력으로 전송하는 것이 바람직한데, 코드 워드는, t보다 많은 에러가 검출되면 정정 불가능하기 때문이다. 종래 기술의 Chien/Forney 구현들에 있어서, 전체 코드 워드는, 에러들의 개수가 결정될 수 있기 전에 처리되어야 한다. 따라서, 동작은 N 클럭 주기들을 이용하는데, N은 코드 워드의 길이이다. Chien/Forney 계산이 종료되면, 에러들의 개수는 결정되고 t의 값과 비교된다. 에러들의 개수가 t보다 적거나 동일하다면, Chien/Forney 블록으로부터의 에러 값들은 수신 코드 워드와 함께 'eXclusive OR' (XOR) 동작하여 에러 정정을 수행하며, 따라서 최종 디코더 출력을 생성한다. 그러나, 에러들의 개수가 t보다 크면, 수신 코드 워드는 디코더 출력으로서 변경되지 않은채 전송된다. 따라서, 종래 기술의 방법은, Chien/Forney 블록에 의한 프로세싱이 종료될 때까지 전체 수신 코드 워드가 저장되는 것을 요구한다. 그러한 방식은 대기 시간 지연을 또한 도입하는데, 디코더 출력은 블록의 전체 N 클럭 주기들이 종료될 때까지 전송 될 수 없기 때문이다.

대조적으로, 본 발명의 실시예는 코드 워드들을 Chien 알고리즘에 두 번 위임한다; 에러들의 개수를 결정하기 위한 첫 번째 시간, 그리고 에러 위치를 결정한기 위한 두 번째 시간. 따라서, Chien 블록(930)이, 에러들의 개수가 한계값 t보다 더 크다는 것을 검출할 때, 수신 코드 워드는 디코더 출력으로서 전달되고, 메모리는 비어있게 된다. 반대로, 에러들의 개수가 한계값 t보다 적거나 동일할 때, 데이터는, 실질적인 에러 위치들과 크기들을 결정하기 위하여 Chien/Forney 블록으로 전달된다. 그러한 구현은, 도 6-8에 도시한 본 발명의 핸드세이킹 실시예와 함께 이용될 때 최적일 것이다. 그러한 결합으로 인하여 Chien/Forney 블록(940)은 동시에 에러들을 계산할 수 있는 반면, Chien 블록(930)은 다음의 코드 워드를 위한 에러들의 개수를 결정하고 있다. 따라서, Chien/Forney 블록(940)의 출력은 즉시 송신될 수 있으며, 대응하는 메모리는 비어있게 된다.

에러 정정 디코더를 위한 강화된 Chien 탐색 셀

전술한 바와 같이, Chien/Forney 블록은 두 개의 다항식; 에러 위치 및 에러 크기 다항식을 입력으로서 수신하고; 에러 벡터를 출력으로 생성한다. 에러 벡터는, 수신 코드 워드내 에러들의 디코더 추정을 나타내는 N 바이트들의 벡터이다. 에러 벡터는 에러들을 정정하기 위하여 수신 코드 워드와 XOR 동작되어, 본래 코드 워드의 디코더 추정을 형성한다. Chien/Forney 블록을 이용하여 에러 벡터를 결정하는 종래의 디코더들은 코드 워드 벡터에 대해 역순인 에러 벡터를 제공한다. 즉, 코드 워드와 에러 벡터는, 그 이상으로 프로세싱 되고 있는 에러 벡터(또는 반대 로, 코드 워드) 없이는 XOR 동작을 할 수 없다. 전통적으로, 이러한 프로세싱은 에러 벡터 또는 수신 코드 워드에 대해 LIFO(Last In, First Out) 동작의 형태를 갖는데, 그 목적은 입력의 순서를 반대로 하여, 코드 워드와 에러 벡터를 XOR 동작할 수 있게 하는 것이다. 유감스럽게도, 이는 N 클럭 주기의 지연을 도입하는데, N은 코드 워드내 바이트들의 개수이다. 부가적으로, 몇몇 종래의 구현들은 단일 코드 워드의 길이보다 더 큰 메모리 크기를 이용하며, 따라서 메모리는 두 개 이상의 개별적인 LIFO 클럭들로 분류되어야 한다. 이로 인하여, 어드레스 지정이 복잡해지고, 메모리 크기는 코드 워드 길이의 정수배로 또한 제한된다.

본 발명의 하나의 예시적인 실시예는 코드 워드에 대해 정확하게 동기화된 출력을 제공하는 새로운 Chien 탐색 셀이다. 즉, 에러 벡터 및 코드 워드 벡터는 XOR 동작되기 이전에 반전을 요구하지 않으며, 따라서 LIFO 블록 또는 다른 반전 수단에 대한 필요성을 제거한다. 그러므로, N 클럭 주기의 지연이 요구되지 않기 때문에, 더 짧은 대기 시간 주기를 얻는다. 더욱이, 본 발명의 실시예는 어드레스 지정 방식(종래 기술에 대해)을 단순화하고, 코드 워드 길이의 단순한 정수배와는 다른 메모리 크기들을 지원한다. 예를 들면, 코드 워드 길이의 두배 반인 메모리가 이용될 수도 있는

데, 이로 인하여, 다음 코드 워드의 일부분이 로드되는 동안에 디코더는 두 개의 코드 워드를 처리한다.

도 10은 종래 기술의 Chien 탐색 셀(1000)을 나타내는 블록도이며, Chien 탐색 및 Chien/Forney 블록들을 위한 블록을 기본적으로 만든다. Chien 탐색 블록은 에러 위치 다항식 Λ를 평가하여 근들을 찾기 위해 이용된다. 근들의 위치들은 수 신 코드내에서 에러가 있는 위치들과 대응한다. 도 10의 Chien 탐색 셀은 다음의 수학식을 구현하기 위해 이용된다:

수학식 7

$$x_i = \sum_{j=0}^{t} \Lambda_j \alpha^{-ij}, \quad \text{where } i = 0 .. (N-1)$$

수학식 7의 계산 결과는 에러들과 대응하는 코드 워드내 바이트 위치들을 위한 0 값이다. Chien 탐색 셀(1000)은 상술한 수학식의 단일 단계를 구현하기 위해 이용된 하드웨어적인 구조이다. 각각의 단계는 에러 위치 다항식 Λ의 단일 계수를 처리하며, 따라서 총 (t+1) 단계들이 존재한다(에러 위치 다항식은 최대 t의 등급을 구비할 수 있기 때문에, 등급 t의 다항식은 (t+1) 계수들을 구비함).

Chien 탐색 셀(1000)은 에러 위치 다항식 계수를 반복적인 방식으로 처리한다. 첫 번째 반복에서, 멀티플렉서(1020)는 처리되고 있는 코드 워드에 대응하는 적절한 다항식 계수(1010)을 수신하여 레지스터(1030)로 전달한다. 연속적인 클럭 주기들에서, 레지스터의 출력은 곱셈기(1040)로 먼저 전송되어 α^{-j} 만큼 곱해지며, 그 다음, 저장을 위해 멀티플렉서(1020)에 의해 레지스터로 다시 전달된다. 이러한 프로세스는 총 N 클럭 주기 동안에 수행된다. 다시 수학식 7을 참조하면, 인덱스 i는 반복 회수를 나타내고, 인덱스 j는 Chien 탐색 셀 단계를 나타낸다. 즉, j의 값은 0부터 t까지 증가하며, 따라서 하드웨어에 구현된 총 (t+1) Chien 탐색 셀 단계들이 존재한다.

설명에 도움이 되는 예로서, 셀은 두 번째 단계(j=1)를 나타낸다고 가정하 자. 따라서, 각각의 클럭 주기에서, 레지스터 출력은 α^{-1} 만큼 곱해질 것이며, 그 결과는 레지스터에 다시 저장된다. 이는 다음과 같은 시퀀스(sequence)를 제공한다:

$$\lambda + \lambda\alpha^{-1}X + (\lambda\alpha^{-1})\,\alpha^{-1}X^2 + ((\lambda\alpha^{-1})\,\alpha^{-1})\alpha^{-1}X^3 + \ldots$$

여기서, X^n 은 n 클럭 주기들에 의한 지연을 나타낸다:

항들을 결합함으로써, 다음과 같은 최종 시퀀스가 제공된다:

$$\lambda + \lambda\alpha^{-1}X + \lambda\alpha^{-2}X^2 + \lambda\alpha^{-3}X^3 + \ldots + \lambda\alpha^{-(N-1)}X^{N-1}$$

이러한 구현과 관련된 문제점은, 에러 위치들은 대응하는 코드 워드 바이트들의 역순으로 제공되며, 따라서 정정을 위해 코드 워드에 더해지기 전에 LIFO 블록에 의해 반전될 필요가 있다는 점이다. LIFO 블록은 다수의 레지스트를 포함하는 하드웨어적인 저장 소자이다. 모든 레지스트들이 채워지면, LIFO 블록은 출력들을 송신한다. 입력 시 마지막 소자는 출력 시 첫 번째 소자가 된다. LIFO 블록의 이용에는 두 가지 문제점들이 존재한다. 하나는, LIFO 블록은 큰 저장/메모리 소자이며, 따라서 IC의 게이트 총 수와 전력 소비가 증가한다는 점이다. 다른 하나는, LIFO 블록은 N 클럭 주기의 대기 시간을 도입한다는 점이다. 이러한 대기 시간은, LIFO 블록이 초기에 채워지는데 N 클럭 주기를 이용하며, 출력들은 마지막 소자가 LIFO에서 판독된 후에 생성될 수 있기 때문에 발생한다.

도 11은 개선된 Chien 탐색 셀(1100)을 나타내는 블록도이며, 본 발명의 원리에 따르면, 메모리 요구 조건들과 지연은 모두 감소된다. 본 발명에 따른 실시 예의 기본적인 원리는, 반대 순서(이하에서 '정규화 순서'로 표현함)로 생성된 계수들을 가짐에도 불구하고, Chien 탐색 블록이 표준 Chien 탐색에 의해 제공된 시퀀스와 동일한 시퀀스를 제공할 수 있다는 점이다. 본 실시예는, 에러 위치들을 코드 워드 정정을 위한 정규화 순서에서 생성하기 위해 이용된 다음의 두 가지 수학식(수학식 8 및 9)을 구현한다.

수학식 8

$$x_i = \sum_{j=0}^{t} \Lambda_j \alpha^{-j(N-1)} \quad \text{where } i = 0$$

수학식 9

$$x_i = \sum_{j=0}^{t} \Lambda_j \alpha^{j} \quad \text{where } i = 1 .. (N-1)$$

도 10에 도시한 종래의 Chien 탐색 셀(1000)은, '0'의 알파 지수로 시작하여 -j(N-1)의 값으로 감소될 때까지 매 클럭 주기 마다 감소되는 에러 위치 다항식 계수를 반복적인 방식으로 처리함을 상기하자. 도 11에 도시한 Chien 탐색 셀(1100)의 본 실시예에 대하여, 알파 지수의 값은 -j(N-1)의 값에서 시작하여 0이 될 때까지 매 클럭 주기 마다 증가된다.

그러므로, 첫 번째 클럭 주기 동안에, 다항식 계수(1110)는 선행 곱셈기(1150)에 의해 $\alpha^{-j(N-1)}$ 만큼 곱해지고, 멀티플렉서(1120)로 전송되며, 저장을 위해 레지스터(1130)로 전달된다. 연속적인 클럭 주기들에서, 레지스터(1130)의 출력은 곱셈기(1140)으로 전송되어 α^{j} 만큼 곱해진 다음, 멀티플렉서(120)에 의해 레지스 터로 다시 전달된다.

설명에 도움이 되는 예로서, 셀은 두 번째 단계(j=1)를 나타낸다고 가정하자. 이는 다음의 시퀀스를 제공한다:

$$\lambda\alpha^{-(N-1)}X + (\lambda\alpha^{-(N-1)})\alpha X^2 + ((\lambda\alpha^{-(N-1)})\alpha)\alpha X^3 + \ldots$$

항들을 결합하여 다음의 시퀀스들을 제공한다:

$$\lambda\alpha^{-(N-1)} + \lambda\alpha^{-(N-2)}X + \lambda\alpha^{-(N-3)}X^2 + \ldots + \lambda\alpha^{-1}X^{N-2} + \lambda X^{N-1}$$

이러한 시퀀스에서 생성된 계수들은 역순(지연 계수들의 역순에 의해 표시됨)이라는 점을 제외하고는, 도 10의 표준 Chien 탐색 셀(1000)에 의해 제공된 시퀀스와 동일하다는 점이 주목된다. 도 11의 Chien 탐색 셀(1100)은 별도의 Galois 필드 곱셈기(한 개 대신 두 개)를 이용하지만, 그 곱셈기들은 작고 저렴하게 하드웨어내에서 구현된다. 반전을 위해 더 이상 LIFO 블록을 요구하지 않고, LIFO를 채우기 위해 요구된 지연을 방지함으로써 발생된 장점과 유용성이, 별도의 Galois 필드 곱셈기를 각각의 Chien 탐색 셀을 위해 지원한다는 단점보다 더 중대하다.

Chien 탐색과 Forney 알고리즘을 단일 블록내에 결합하여 실행하는 것이 공통적인 설계인데, 이는 양쪽 알고리즘들이 유사한 기능들의 수행을 요구하기 때문이다. 도 11에 도시되어 설명된 본 발명에 따른 실시예의 원리는 Forney 블록 셀들에도 마찬가지로 동일하게 적용되는데, 이는 Forney 알고리즘이 매우 유사한 하드웨어를 이용하여 에러 크기 다항식을 처리하기 때문이다. 본 발명의 원리는 Forney 블록 셀들과 Chien/Forney 블록 셀들의 설계로 확장될 수도 있다는 것이 본 기술 분야의 당업자에게 알려져 있다.

수많은 현존하는 시스템들은 리드-솔로몬 코드들을 인코딩하고 디코딩하는 '상용' 집적 회로들을 이용한다. 이러한 IC들은 특정 수량의 프로그램 가능성(예를 들면, RS(255,k) 여기서 t = 1 부터 16 심볼들)을 지원하는 경향이 있다. 최근의 경향은 VHDL 또는 Verilog 설계들(논리 코어들 또는 지적재산권 코어들)에 관한 것이다. 이들은 표준 IC들에 비하여 수많은 장점들이 있다. 논리 코어는 다른 VHDL 또는 Verilog 컴포넌트들과 집적될 수 있으며, FPGA(Field Programmable Gate Array) 또는 ASIC(Application Specific Integrated Circuit)으로 종합될 수 있다 - 이는 소위 'System on Chip' 설계들을 가능하게 하여, 다중 모듈들이 단일 IC에 결합될 수 있다. 제품 양에 의존하기 때문에, 논리 코어들은 '표준' IC들보다 상당히 낮은 시스템 비용들을 자주 제공할 수 있다.

비록 본 발명은 하드웨어적인 구현으로 설명되지만, 본 발명의 원리는 그에 한정되어 해석되어서는 안된다. 최근까지, '실시간'에서의 소프트웨어적인 구현들은 가장 단순한 리드-솔로몬 코드들(즉, t의 작은 값들을 갖는 코드들)을 제외한 모두에 대해서 매우 많은 계산 능력을 요구하였다. 리드-솔로몬 코드들을 소프트웨어에 구현함에 있어서의 중대한 어려움은, 일반적인 목적의 프로세서들은 Galois 필드 산술 연산들을 지원하지 않는다는 점이다. 예를 들어, Galois 필드 곱셈을 소프트웨어에 구현하기 위해서는, 0을 위한 테스트, 두 개의 테이블 룩업, 모듈로 덧셈기 및 안티-로그 테이블 룩업을 요구한다. 그러나, 세심한 설계와 함께 프로세서 성능에 있어서의 증진은, 소프트웨어적인 구현들이 상대적으로 높은 데이터 전송율로 동작할 수 있음을 의미한다.

본 발명은 예시적인 실시예들 및/또는 구성들에 관하여 설명되지만, 본 발명은 본 명세서에서의 사상과 범위 내에서 더욱 수정될 수 있다. 그러므로, 본 출원은 본 발명의 일반적인 원리들을 이용하여 본 발명에 대한 임의의 변종, 이용, 적용을 포함하도록 의도된다. 또한, 본 출원은, 본 발명과 관련되고, 첨부된 특허청구범위의 범위에 포함되는 본 기술분야의 공지되거나 공용된 것으로서 본 명세서로부터 이탈한 각종 변형들을 포함하도록 의도된다.

(57) 청구의 범위

청구항 1.

적어도 네 개의 블록들 - 상기 블록들은 신드롬 생성 블록(410), 에러 다항식 블록(420), 에러 위치 블록(430), 및 에러 크기 블록(440)을 포함함 - 을 구비하는 주기적인 선형 블록 코드 에러 정정 디코더에서, 코드 워드의 데이터 에러 정정을 수행하기 위한 방법으로서,

신드롬을 상기 신드롬 생성 블록(410)내 상기 코드 워드로부터 계산하는 단계;

에러 다항식을 상기 에러 다항식 블록(420)내 상기 신드롬으로부터 생성하는 단계;

에러 위치를 상기 에러 위치 블록(430)내 상기 에러 다항식으로부터 결정하는 단계; 및

에러 크기를 상기 에러 크기 블록(440)내 상기 에러 다항식으로부터 계산하는 단계

를 포함하고,

상기 적어도 네 개의 블록들 중 적어도 하나의 블록은 비활성 메시지(640,650)를 상기 적어도 네 개의 블록들 중 다른 블록들로 전송하는 것을 특징으로 하는 데이터 에러 정정 방법.

청구항 2.

제1항에 있어서,

상기 코드 워드를 상기 에러 위치와 상기 에러 크기에 응답하여 정정하는 단계를 더 포함하는 데이터 에러 정정 방법.

청구항 3.

제1항에 있어서,

상기 비활성 메시지는 수신 준비 메시지(640)인 데이터 에러 정정 방법.

청구항 4.

제1항에 있어서,

상기 비활성 메시지는 송신 준비 메시지(650)인 데이터 에러 정정 방법.

청구항 5.

제1항에 있어서,

상기 에러 다항식 블록은 Euclid의 알고리즘 블록인 데이터 에러 정정 방법.

청구항 6.

제1항에 있어서,

상기 에러 다항식 블록은 Berlekamp-Massey 알고리즘 블록인 데이터 에러 정정 방법.

청구항 7.

제1항에 있어서,

상기 에러 위치 블록은 Chien 탐색 블록인 데이터 에러 정정 방법.

청구항 8.

제1항에 있어서,

상기 에러 크기 블록은 Forney 알고리즘 블록인 데이터 에러 정정 방법.

청구항 9.
제1항에 있어서,

상기 에러 위치 블록 및 상기 에러 크기 블록과 관련된 상기 동작들은 결합된 Chien/Forney 블록내에서 수행되는 데이터 에러 정정 방법.

청구항 10.
제1항에 있어서,

상기 주기적인 선형 블록 코드 에러 정정 디코더는 리드-솔로몬 디코더인 데이터 에러 정정 방법.

청구항 11.
코드 워드내 데이터 에러 검출을 수행하기 위한 디코더로서,

상기 코드 워드로부터 신드롬을 계산하기 위한 수단(410);

상기 신드롬으로부터 에러 다항식을 생성하기 위한 수단(420);

상기 에러 다항식으로부터 에러 위치를 결정하기 위한 수단(430); 및

상기 에러 다항식으로부터 에러 크기를 계산하기 위한 수단(440)

을 포함하고,

인트라-디코더 비활성 메시지(640,650)를 전송하는 수단은, 상기 신드롬을 계산하기 위한 수단(410), 상기 에러 다항식을 생성하기 위한 수단(420), 상기 에러 위치를 결정하기 위한 수단(430), 및 상기 에러 크기를 계산하기 위한 수단(440)을 포함하는 그룹으로부터 선택된 적어도 하나의 컴포넌트내에 통합되는 것을 특징으로 하는 디코더.

청구항 12.
제11항에 있어서,

입력들로서의 상기 코드 워드, 상기 에러 위치, 및 상기 에러 크기의 수신에 응답하여 상기 코드 워드를 정정하기 위한 수단(450)을 더 포함하는 디코더.

청구항 13.
제11항에 있어서,

상기 디코더는 리드-솔로몬 디코더인 디코더.

청구항 14.
제11항에 있어서,

상기 인트라-디코더 비활성 메시지는 수신 준비 메시지(640)인 디코더.

청구항 15.
제11항에 있어서,

상기 인트라-디코더 비활성 메시지는 송신 준비 메시지(650)인 디코더.

청구항 16.
제11항에 있어서,

에러 다항식(420)을 생성하기 위한 수단은 Euclid의 알고리즘 블록인 디코더.

청구항 17.
제11항에 있어서,

에러 위치(430) 블록을 생성하기 위한 수단은 Chien 탐색 블록인 디코더.

청구항 18.
제11항에 있어서,

에러 크기(440)를 계산하기 위한 수단은 Forney 알고리즘 블록인 디코더.

청구항 19.
코드 워드내 데이터 에러 검출을 수행하기 위한 디코더로서,

신드롬을 상기 코드 워드로부터 계산하기 위한 신드롬 생성기(410);

에러 다항식을 상기 신드롬으로부터 생성하기 위한 에러 다항식 생성기(420);

에러 위치를 상기 에러 다항식으로부터 결정하기 위한 에러 위치 생성기(430); 및

에러 크기를 상기 에러 다항식으로부터 계산하기 위한 에러 크기 생성기(440)

를 포함하고,

인트라-디코더 비활성 메시지(640,650)를 전송하는 수단은, 상기 신드롬 생성기(410), 상기 에러 다항식 생성기(420), 상기 에러 위치 생성기(430), 및 상기 에러 크기 생성기(440)을 포함하는 그룹으로부터 선택된 적어도 하나의 컴포넌트내에 통합되는 것을 특징으로 하는 디코더.

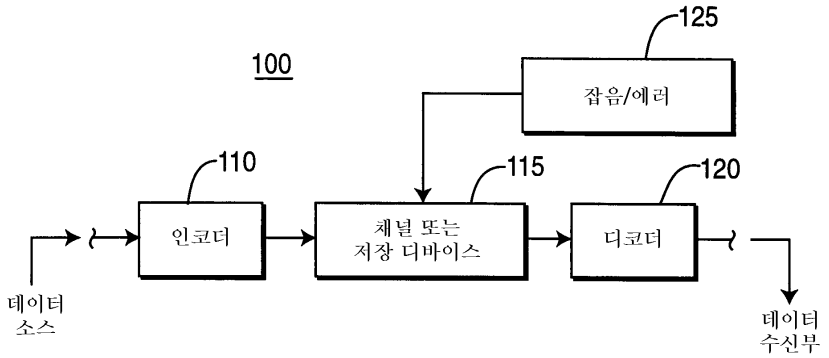
청구항 20.
제19항에 있어서,

입력들로서의 상기 코드 워드, 상기 에러 위치, 및 상기 에러 크기의 수신에 응답하여 상기 코드 워드를 정정하기 위한 에러 정정된 코드 워드 생성기(450)를 더 포함하는 디코더.

도면

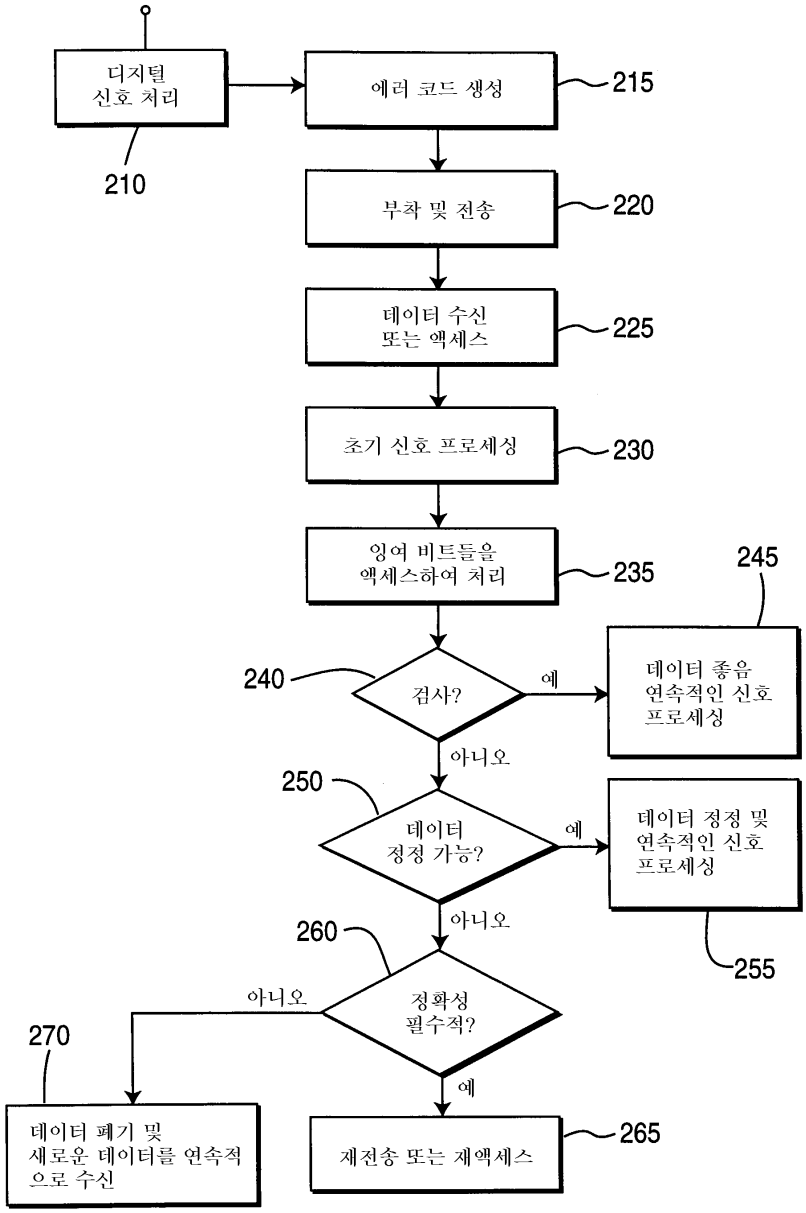
도면1

(종래 기술)



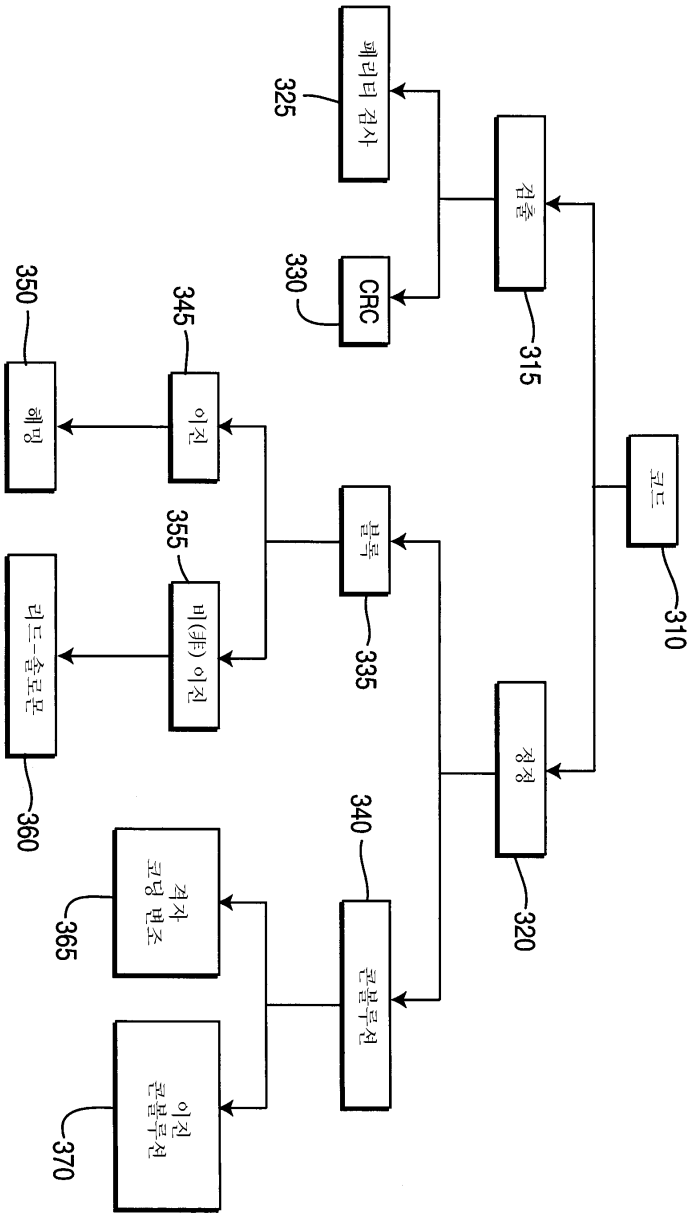
도면2

(종래 기술)


디지털
신호 처리
210
에러 코드 생성
215
부착 및 전송
220
데이터 수신
또는 액세스
225
초기 신호 프로세싱
230
잉여 비트들을
액세스하여 처리
235
245
240
검사?
예
데이터 좋음
연속적인 신호
프로세싱
아니오
250
데이터
정정 가능?
예
데이터 정정 및
연속적인 신호
프로세싱
255
아니오
260
정확성
필수적?
아니오
270
예
데이터 폐기 및
새로운 데이터를 연속적
으로 수신
재전송 또는 재액세스
265

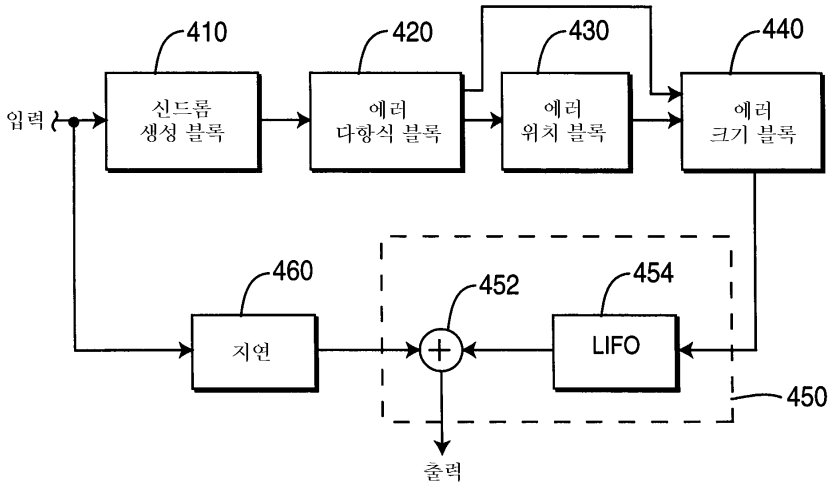
도면3

(종래 기술)

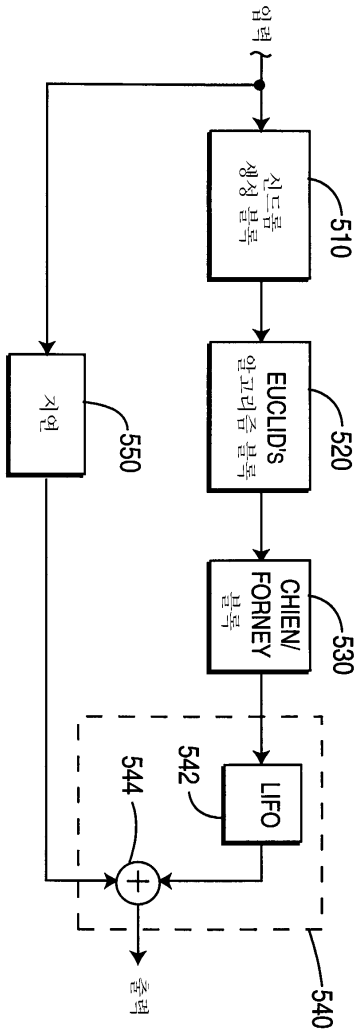


도면4

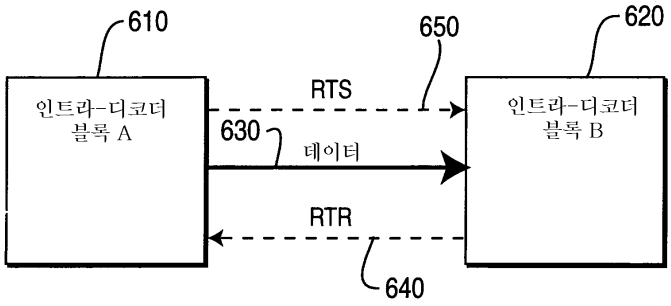
(종래 기술)



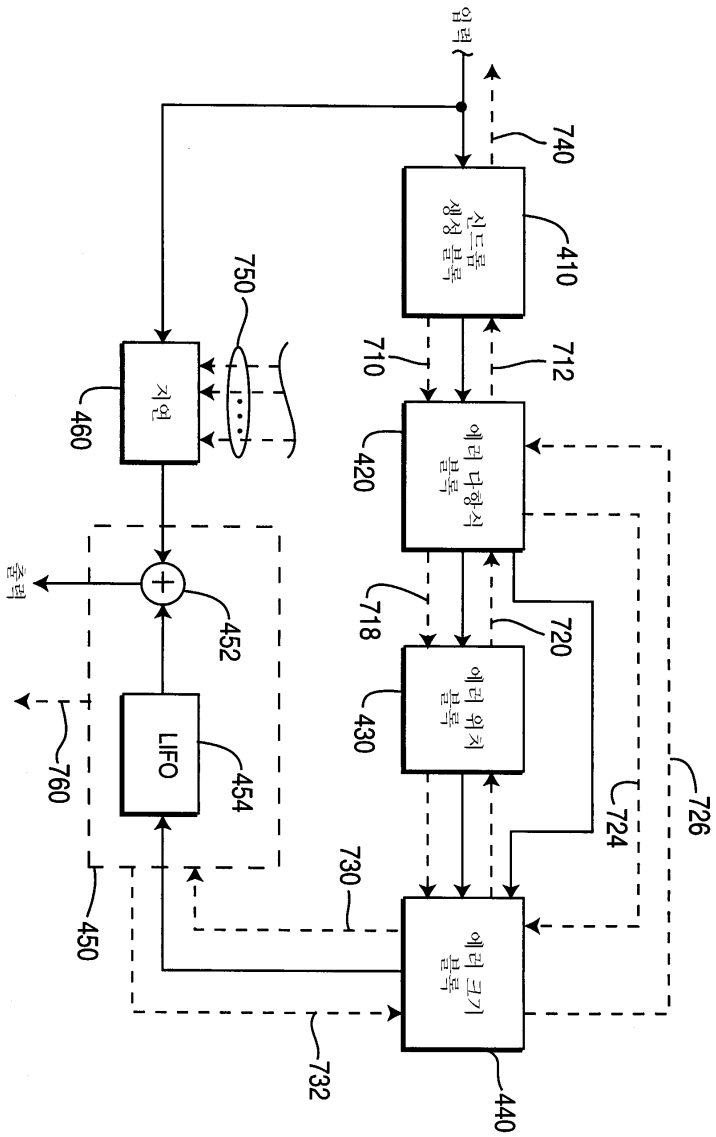
도면5


입력
신드롬 생성 블록
510
EUCLID's 알고리즘 블록
520
CHIEN/ FORNEY 블록
530
LIFO
542
544
540
지연
550
출력

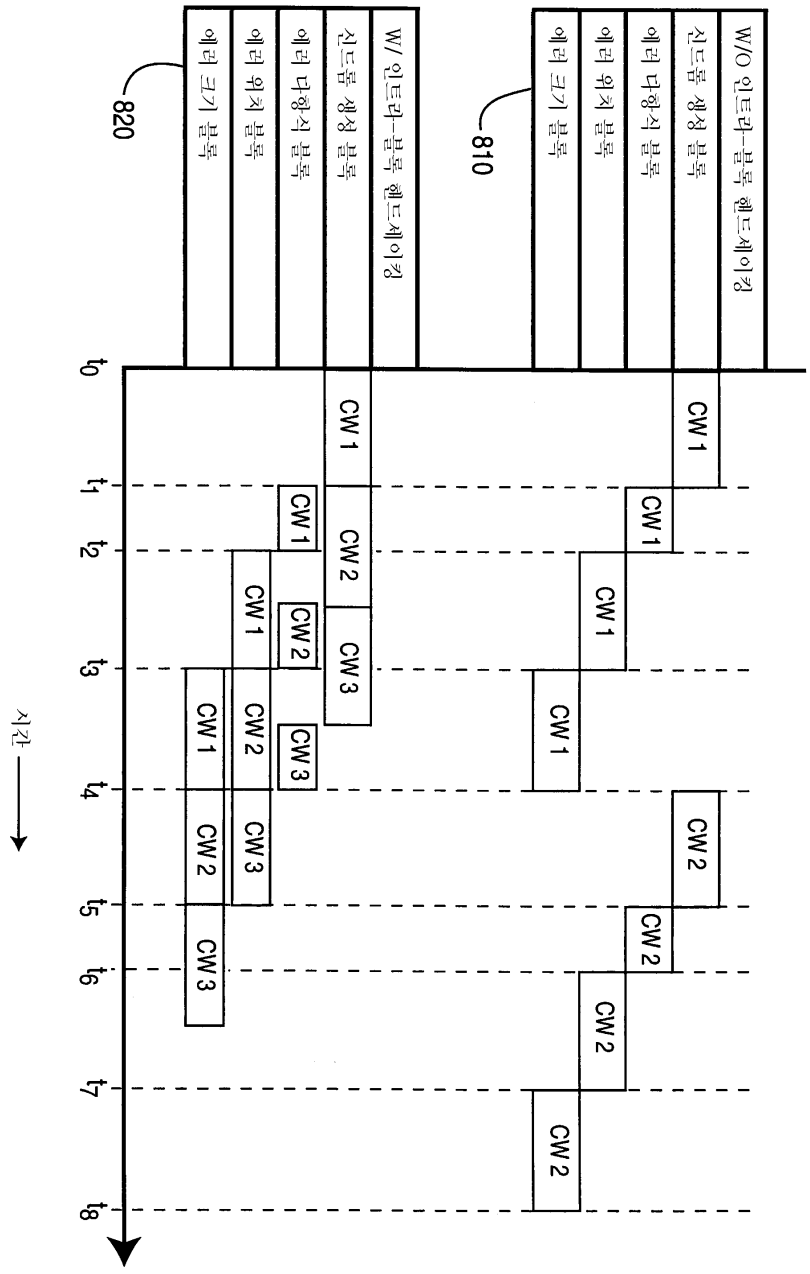
도면6


610
650
620
인트라-디코더 블록 A
RTS
630
데이터
RTR
640
인트라-디코더 블록 B

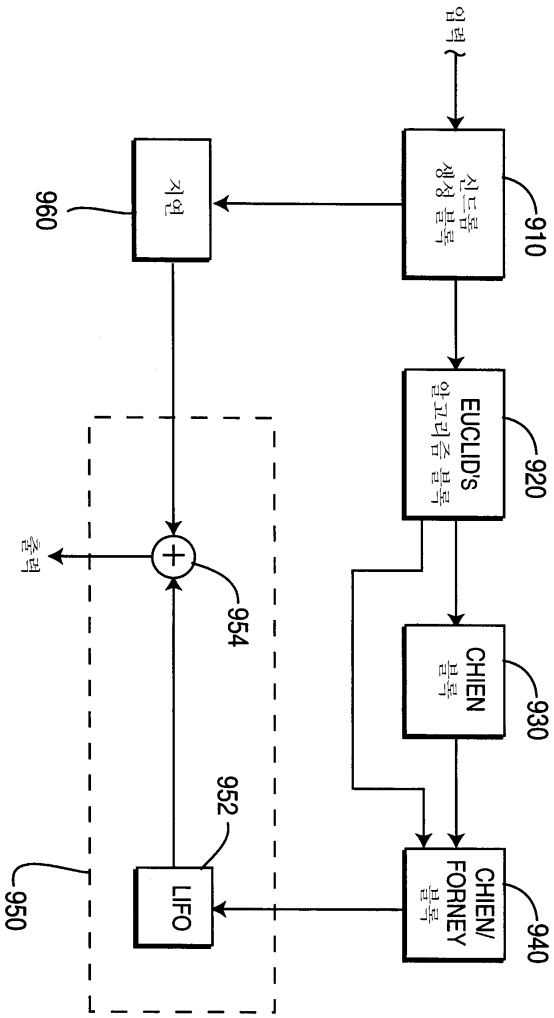
도면7


입력
740
신드롬 생성 블록
410
750
710
712
지연
460
에러 다항식 블록
420
출력
452
718
720
에러 위치 블록
430
760
LIFO
454
726
724
730
에러 크기 블록
450
732
440

도면8

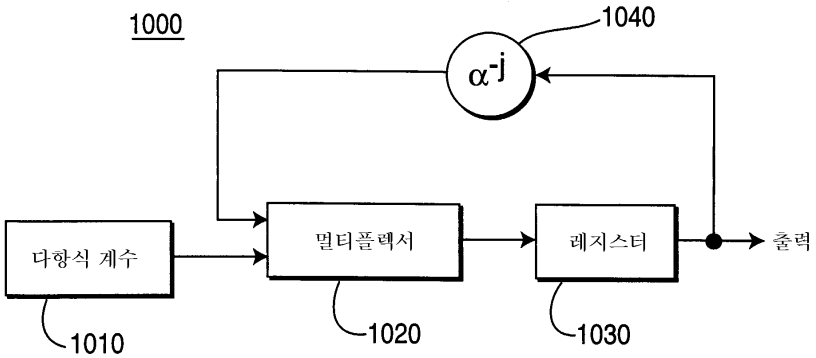

W/O 인트라-블록 헤드세이킹
신드롬 생성 블록
에러 다항식 블록
에러 위치 블록
에러 크기 블록
810
W/ 인트라-블록 헤드세이킹
신드롬 생성 블록
에러 다항식 블록
에러 위치 블록
에러 크기 블록
820
CW 1
CW 2
CW 3
t0
t1
t2
t3
t4
t5
t6
t7
t8
시간

도면9

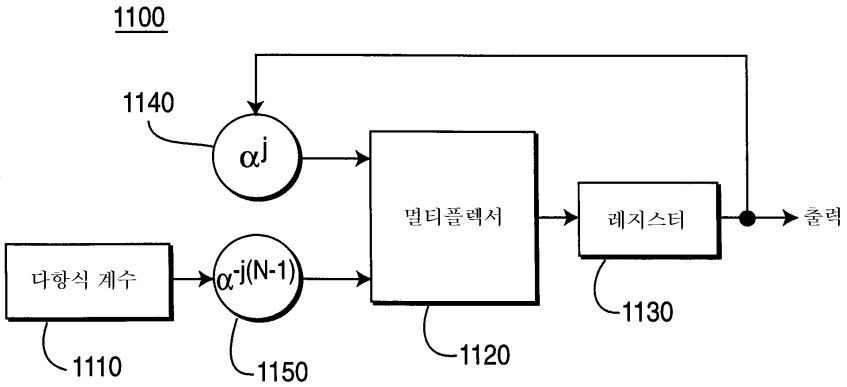


도면10

(종래 기술)



도면11


1100
1140
α^j
다항식 계수
$\alpha^{-j(N-1)}$
멀티플렉서
레지스터
출력
1110
1150
1120
1130