(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2011-0006666
(43) 공개일자 2011년01월20일

(51) Int. Cl.
H04N 7/26 (2006.01)
(21) 출원번호 10-2010-7024022
(22) 출원일자(국제출원일자) 2009년03월27일
심사청구일자 없음
(85) 번역문제출일자 2010년10월26일
(86) 국제출원번호 PCT/US2009/038542
(87) 국제공개번호 WO 2009/120952
국제공개일자 2009년10월01일
(30) 우선권주장
08162031.2 2008년08월07일
유럽특허청(EPO)(EP)
61/072,316 2008년03월28일 미국(US)

(71) 출원인
톰슨 라이센싱
프랑스 92130 이씨레뮬리노 잔 다르크 뤼 1-5
(72) 발명자
뮐러, 스테판
독일 78048 빌링겐-슈베닝겐 프포제이머 스트라쎄 15/1
크라브트첸코, 알렉산더
독일 78056 빌링겐-슈베닝겐 데우텐버그링 146
(뒷면에 계속)
(74) 대리인
양영준, 백만기, 전경석

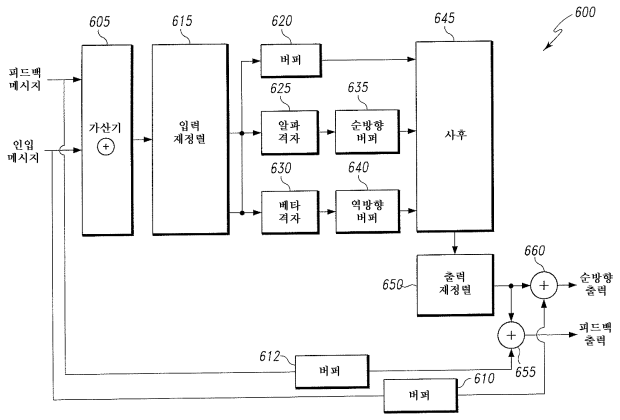
전체 청구항 수 : 총 32 항

(54) 신호 디코딩을 위한 장치 및 방법

(57) 요 약

본 실시예들의 태양에 따르면, 비트스트림을 디코딩하는 방법이 기술되며, 상기 방법은 바이트 코드 인코딩 프로세스를 사용하여 인코딩된 복조된 비트스트림을 수신하는 단계, 상기 복조된 비트스트림의 일부를 비트들의 부분 집합으로 배열하는 단계, 상기 비트들의 부분 집합을 재정렬하는 단계, 및 상기 바이트 코드 인코딩 프로세스 및 상기 비트들의 재정렬된 부분 집합의 속성에 기초하여 상기 비트들의 부분 집합을 디코딩하는 단계를 포함한다.

대 표 도 - 도6



(72) 발명자

클린크, 스타니슬라우

독일 피엘-78052 탄헤임 임 홀윈켈 7

로프레스토, 스콧

미국 60622 일리노이주 시카고 노쓰 옥클리 블러바드 에이피티. 넘버1 1329

특허청구의 범위

청구항 1

비트스트림을 디코딩하는 방법(1700)으로서,

복조된 비트스트림을 수신하는 단계(1710) - 상기 복조된 비트스트림은 바이트 코드(byte-code) 인코딩 프로세스를 사용하여 인코딩됨 - ;

상기 복조된 비트스트림의 일부를 비트들의 부분집합으로 배열하는 단계(1720);

상기 비트들의 부분 집합을 재정렬하는 단계(1720); 및

상기 바이트 코드 인코딩 프로세스 및 상기 비트들의 재정렬된 부분 집합의 속성에 기초하여 상기 비트들의 부분 집합을 디코딩하는 단계(1730)

를 포함하는 비트스트림 디코딩 방법(1700).

청구항 2

제1항에 있어서,

상기 속성은 패리티 검사 속성인 비트스트림 디코딩 방법(1700).

청구항 3

제1항에 있어서,

상기 디코딩 단계(1730)는 이진 저밀도 패리티 검사(binary low density parity check) 디코딩 프로세스를 사용하여 디코딩하는 단계를 포함하는 비트스트림 디코딩 방법(1700).

청구항 4

제3항에 있어서,

상기 이진 저밀도 패리티 검사 디코딩 프로세스는 원시 다항식(primitive polynomial)으로부터 패리티 검사 행렬을 생성하는 것에 기초하는 비트스트림 디코딩 방법(1700).

청구항 5

제1항에 있어서,

상기 디코딩 단계(1730)는 비 이진 저밀도 패리티 검사 디코딩 프로세스를 사용하여 디코딩하는 단계를 포함하는 비트스트림 디코딩 방법(1700).

청구항 6

제1항에 있어서,

상기 디코딩 단계(1730)는 비트 기반 격자(bit based trellis) 디코딩 프로세스를 사용하여 디코딩하는 단계를 포함하는 비트스트림 디코딩 방법(1700).

청구항 7

제1항에 있어서,

상기 디코딩 단계(1730)는 심볼 기반 격자 디코딩 프로세스를 사용하여 디코딩하는 단계를 포함하는 비트스트림 디코딩 방법(1700).

청구항 8

제1항에 있어서,

상기 바이트 코드 인코딩 프로세스는 유한체(Galois Field)(256) 공간에 기초하고,

상기 디코딩 단계(1730)는 유한체(2) 공간, 유한체(4) 공간 및 유한체(8) 공간 중 하나에 기초하는 비트스트림 디코딩 방법(1700).

청구항 9

비트스트림을 디코딩하는 방법(1800)으로서,

복조된 비트스트림을 수신하는 단계(1810);

상기 복조된 비트스트림의 일부를 비트들의 부분 집합으로 배열하는 단계(1820);

상기 비트들의 부분 집합에서 심볼을 식별하는 단계(1830); 및

상기 심볼의 이전의 디코딩된 상태 및 상기 비트들의 부분 집합 내의 적어도 두 비트 사이의 관계에 기초하여 상기 심볼의 현재 상태를 디코딩하는 단계(1850)

를 포함하는 비트스트림 디코딩 방법(1800).

청구항 10

제9항에 있어서,

상기 관계는 관계들의 집합인 비트스트림 디코딩 방법(1800).

청구항 11

제10항에 있어서,

상기 관계들의 집합은 패리티 관계들인 비트스트림 디코딩 방법(1800).

청구항 12

제9항에 있어서,

상기 비트들의 부분 집합에서 심볼을 식별하는 단계(1830)는 상기 비트들의 부분 집합에서 복수의 심볼을 식별하는 단계를 포함하는 비트스트림 디코딩 방법(1800).

청구항 13

제12항에 있어서,

상기 디코딩 단계에 앞서 상기 복수의 심볼을 재정렬하는 단계를 더 포함하는 비트스트림 디코딩 방법(1800).

청구항 14

제9항에 있어서,

상기 디코딩 단계(1850)는 콘볼루션(convolutional) 디코딩 프로세스를 포함하는 비트스트림 디코딩 방법(1800).

청구항 15

제9항에 있어서,

상기 비트스트림은 바이트 코드 인코딩 프로세스를 사용하여 인코딩되는 비트스트림 디코딩 방법(1800).

청구항 16

제15항에 있어서,

상기 바이트 코드 인코딩 프로세스는 유한체(256) 공간 내의 짧은 선형 블록 코드에 기초하는 비트스트림 디코딩 방법(1800).

청구항 17

복조된 비트스트림을 수신하기 위한 수단(1310) - 상기 복조된 비트스트림은 바이트 코드 인코딩 프로세스를 사용하여 인코딩됨 - ;

상기 복조된 비트스트림의 일부를 비트들의 부분 집합으로 그룹화하기 위한 수단(1330);

상기 비트들의 부분 집합을 재정렬하기 위한 수단(1330); 및

상기 바이트 코드 인코딩 프로세스 및 상기 비트들의 재정렬된 부분 집합의 속성에 기초하여 상기 비트들의 부분 집합을 디코딩하기 위한 수단(1360)

을 포함하는 장치(1300).

청구항 18

제17항에 있어서,

상기 속성은 패리티 검사 속성인 장치(1300).

청구항 19

제17항에 있어서,

상기 디코딩 수단(1360)은 이진 저밀도 패리티 검사 디코딩 프로세스를 사용하여 디코딩하기 위한 수단을 포함하는 장치(1300).

청구항 20

제19항에 있어서,

상기 이진 저밀도 패리티 검사 디코딩 프로세스는 원시 다항식으로부터 패리티 검사 행렬을 생성하는 것에 기초하는 장치(1300).

청구항 21

제17항에 있어서,

상기 디코딩 수단(1360)은 비 이진 저밀도 패리티 검사 디코딩 프로세스를 사용하여 디코딩하기 위한 수단을 포함하는 장치(1300).

청구항 22

제17항에 있어서,

상기 디코딩 수단(1360)은 비트 기반 격자 디코딩 프로세스를 사용하여 디코딩하기 위한 수단을 포함하는 장치(1300).

청구항 23

제17항에 있어서,

상기 디코딩 수단(1360)은 심볼 기반 격자 디코딩 프로세스를 사용하여 디코딩하기 위한 수단을 포함하는 장치(1300).

청구항 24

제17항에 있어서,

상기 바이트 코드 인코딩 프로세스는 유한체(256) 공간에 기초하고,

디코딩(1730)하는 것은 유한체(2) 공간, 유한체(4) 공간 및 유한체(8) 공간 중 하나에 기초하는 장치(1300).

청구항 25

비트스트림을 디코딩하기 위한 장치(600)로서,

복조된 비트스트림을 수신하기 위한 수단(605);

상기 복조된 비트스트림의 일부를 비트들의 부분 집합으로 배열하기 위한 수단(615);

상기 비트들의 부분 집합에서 심볼을 식별하기 위한 수단(615); 및

상기 심볼의 이전의 디코딩된 상태 및 상기 비트들의 부분 집합 내의 적어도 두 비트 사이의 관계에 기초하여 상기 심볼의 현재 상태를 디코딩하기 위한 수단(625, 630, 645)

을 포함하는 비트스트림 디코딩 장치(600).

청구항 26

제25항에 있어서,

상기 관계는 관계들의 집합인 비트스트림 디코딩 장치(600).

청구항 27

제26항에 있어서,

상기 관계들의 집합은 패리티 관계들인 비트스트림 디코딩 장치(600).

청구항 28

제25항에 있어서,

상기 비트들의 부분 집합에서 심볼을 식별하기 위한 수단(615)은 상기 비트들의 부분 집합에서 복수의 심볼을 식별하는 것을 포함하는 비트스트림 디코딩 장치(600).

청구항 29

제28항에 있어서,

상기 디코딩에 앞서 상기 복수의 심볼을 재정렬하기 위한 수단(615)을 더 포함하는 비트스트림 디코딩 장치(600).

청구항 30

제25항에 있어서,

상기 디코딩 수단(625,630,645)은 콘볼루션 디코딩 프로세스를 포함하는 비트스트림 디코딩 장치(600).

청구항 31

제25항에 있어서,

상기 비트스트림은 바이트 코드 인코딩 프로세스를 사용하여 인코딩되는 비트스트림 디코딩 장치(600).

청구항 32

제31항에 있어서,

상기 바이트 코드 인코딩 프로세스는 유한체(256) 공간 내의 짧은 선형 블록 코드에 기초하는 비트스트림 디코딩 장치(600).

명 세 서

기 술 분 야

[0001] 관련 출원들에 대한 상호 참조

[0002] 본 출원은 미국에서 2008년 3월 28일에 출원된 임시 특허 출원 제61/072,316호의 35 U.S.C.§119에 따른 이익

및 유럽 특허청에서 2008년 8월 7일에 출원된 유럽 특허 출원 EP08162031의 35 U.S.C.§119에 따른 이익을 청구한다.

[0003] 본 개시 내용은 일반적으로 디지털 신호 데이터 송신 시스템의 동작과 관련되고, 보다 구체적으로는 이동, 보행자 및 개인 장치들에 의해 사용되도록 의도되는 방송 텔레비전을 위한 데이터의 디코딩과 관련된다.

배 경 기 술

[0004] 이 절은 아래에 기술되는 본 발명의 다양한 태양과 관련될 수 있는 다양한 기술 태양을 독자에게 소개하고자 하는 것이다. 이러한 논의는 본 발명의 다양한 태양에 관한 보다 나은 이해를 촉진하기 위해 독자에게 배경 정보를 제공하는 데 도움이 될 것이라고 여겨진다. 따라서, 본 진술은 이에 비추어 읽혀져야 하며, 종래 기술임을 인정하는 것으로 읽혀지지 않아야 함을 이해해야 한다.

[0005] 전 세계의 텔레비전 방송 시스템들은 아날로그 및 비디오 신호들을 전달하는 것으로부터 현대의 디지털 통신 시스템들로 옮겨왔다. 예컨대, 미국에서 ATSC(Advanced Television Standards Committee)는 "ATSC 표준: 디지털 텔레비전 표준 A/53"(A53 표준)이라고 불리는 표준을 개발해 왔다. A53 표준은 디지털 텔레비전 방송을 위한 데이터가 어떻게 인코딩 및 디코딩되어야 하는지를 정의한다. 또한, 미국 FCC(Federal Communications Commission)는 텔레비전 방송을 위한 전자기 스펙트럼의 부분들을 배당하였다. 지상파(즉, 케이블이나 위성이 아님) 디지털 텔레비전 방송의 송신을 위해, FCC는 할당된 부분 내의 인접한 6 MHz 채널을 방송자(broadcaster)에게 할당한다. 6 MHz 채널 각각은 A53 표준의 인코딩 및 변조 형식에 기초하여 약 19 Mb/s의 채널 용량을 갖는다. 또한, FCC는 6 MHz 채널을 통한 지상파 디지털 텔레비전 송신은 A53 표준을 준수해야 함을 지시하였다.

[0006] A53 표준과 같은 디지털 방송 신호 송신 표준은 소스 데이터(예컨대 디지털 오디오 및 비디오 데이터)가 어떻게 처리되어 채널을 통해 송신되는 신호로 변조되는지를 정의한다. 상기 처리는 소스 데이터에 중복 정보(redundant information)를 추가하여, 송신되는 신호에 채널이 잡음 및 다중 경로 간섭을 추가하더라도 채널로부터 신호를 수신하는 수신자가 소스 데이터를 복구할 수 있도록 한다. 소스 데이터에 추가된 중복 정보는 소스 데이터가 송신되는 실효 데이터 속도를 감소시키지만, 송신되는 신호로부터 소스 데이터를 성공적으로 복구할 가능성을 증가시킨다.

[0007] A53 표준 개발 프로세스는 고해상도 텔레비전(High Definition Television; HDTV) 및 고정 수신에 초점이 맞추어졌다. 시스템은 이미 시장에 진입하기 시작한 대형 고해상도 텔레비전 화면들에 대한 비디오 비트 속도(bit rate)를 최대화하도록 설계되었다. ATSC A/53 표준, 또는 레거시(legacy) 인코딩 및 송신 표준 하에서 방송되는 송신은 이동 수신기들에 대한 어려움을 제시한다.

[0008] 이러한 사실을 인식하여, ATSC는 2007년에 방송자들이 이들의 디지털 방송 신호를 통해 이동 및 핸드헬드(handheld)장치들에게 텔레비전 컨텐트를 전달할 수 있도록 할 표준을 개발하기 위한 프로세스의 착수를 발표하였다. 레거시 송신 표준에 대한 변화는 추가적인 데이터 중복을 도입하기 위한 추가적인 인코딩 방식을 포함한다. 추가적인 인코딩은 이동, 핸드헬드 및 보행자 장치들 내의 발전된 수신기들을 사용하여 더 나은 성능을 보이면서도 레거시 A53 표준과 하위 호환성을 유지하도록 적응되었다. 제안된 변화는 또한 동일한 무선 주파수(Radio Frequency; RF) 채널에서 기존의 수신 장비에 대한 악영향 없이 기존의 ATSC 서비스가 동작할 수 있도록 한다.

발명의 내용

해결하려는 과제

[0009] 그러나, 현재의 A53 표준 방송 신호에 도입된 어떠한 새로운 인코딩 방식도 ATSC M/H 수신기의 디코딩 프로세스에 추가적인 복잡함을 가져온다. 예컨대, 레거시 동작을 위해 이미 존재하는 격자(trellis) 디코딩 또는 리드 솔로몬(Reed-Solomon) 디코딩 이외에 연접 블록 코딩(concatenated block coding)을 추가하는 것은, 블록 디코딩을 위해 종종 사용되는 반복 사후(iterative a-posteriori) 유형의 디코더들과 연관된 알려진 직렬화 입력 비트 처리에 기초한 수신기에서의 처리 시간을 현저히 증가시킬 수 있다. 디코딩 성능은 종종 최적의 또는 바람직한 디코딩된 출력을 달성하는 데 필요한 반복의 횟수와 직접 연관된다. 디코더에서의 반복 횟수 또는 효율은 디코더 아키텍처(architecture) 및 인입(incoming) 메시지 신호를 디코딩 아키텍처가 처리할 수 있는 속도의 결합에 의해 제한될 것이다. 또한, 디코더 처리 시간을 감소시키는 데 사용되는 큰 참조 테이블을 사용하는 것은 디코더 아키텍처의 물리적인 크기에 관해 비효율을 만들어낸다. 따라서, 개선된 디코딩 아키텍처를 개발함으로

써 디코딩 프로세스 및 디코더 효율을 개선하는 것이 바람직하다.

과제의 해결 수단

[0010] 본 실시예들의 태양에 따르면, 비트스트림(bitstream)을 디코딩하는 방법이 기술되며, 상기 방법은 바이트 코드(byte-code) 인코딩 프로세스를 사용하여 인코딩된 복조된 비트스트림을 수신하는 단계, 상기 복조된 비트스트림의 일부를 비트들의 부분 집합으로 배열하는 단계, 상기 비트들의 부분 집합을 재정렬하는 단계, 및 상기 바이트 코드 인코딩 프로세스 및 상기 비트들의 재정렬된 부분 집합의 속성에 기초하여 상기 비트들의 부분 집합을 디코딩하는 단계를 포함한다.

[0011] 본 실시예들의 다른 태양에 따르면, 비트스트림을 디코딩하는 방법이 기술되며, 상기 방법은 복조된 비트스트림을 수신하는 단계, 상기 복조된 비트스트림의 일부를 비트들의 부분 집합으로 배열하는 단계, 상기 비트들의 부분 집합에서 심볼(symbol)을 식별하는 단계, 및 상기 심볼의 이전의 디코딩된 상태 및 상기 비트들의 부분 집합 내의 적어도 두 비트 사이의 관계의 집합에 기초하여 상기 심볼의 현재 상태를 디코딩하는 단계를 포함한다.

[0012] 본 실시예들의 다른 태양에 따르면, 바이트 코드 인코딩 프로세스를 사용하여 인코딩된 복조된 비트스트림을 수신하기 위한 수단, 상기 복조된 비트스트림의 일부를 비트들의 부분 집합으로 그룹화하기 위한 수단, 상기 비트들의 부분 집합을 재정렬하기 위한 수단, 및 상기 바이트 코드 인코딩 프로세스 및 상기 비트들의 재정렬된 부분 집합의 속성에 기초하여 상기 비트들의 부분 집합을 디코딩하기 위한 수단을 포함하는 장치가 기술된다.

[0013] 본 실시예들의 다른 태양에 따르면, 복조된 비트스트림을 수신하기 위한 수단, 상기 복조된 비트스트림의 일부를 비트들의 부분 집합으로 배열하기 위한 수단, 상기 비트들의 부분 집합에서 심볼을 식별하기 위한 수단, 및 상기 심볼의 이전의 디코딩된 상태 및 상기 비트들의 부분 집합 내의 적어도 두 비트 사이의 관계의 집합에 기초하여 상기 심볼의 현재 상태를 디코딩하기 위한 수단을 포함하는 장치가 기술된다.

도면의 간단한 설명

[0014] 도 1은 본 개시 내용의 인코더에 관한 실시예의 블록도.

도 2는 본 개시 내용의 연접 바이트 코드 인코딩 회로에 관한 실시예의 블록도.

도 3은 본 개시 내용의 수신기에서 사용되는 디코더의 실시예의 블록도.

도 4는 본 개시 내용의 연접 바이트 코드 디코딩 회로에 관한 실시예의 블록도.

도 5는 본 개시 내용의 성분 바이트 코드 디코더에 관한 실시예의 블록도.

도 6은 본 개시 내용의 성분 바이트 코드 디코더의 다른 실시예의 블록도.

도 7은 본 개시 내용의 디코더에 대한 격자 트리(trellis tree)에 관한 실시예를 도시하는 도면.

도 8은 본 개시 내용의 디코더에 대한 격자 트리의 다른 실시예들 도시하는 도면.

도 9는 본 개시 내용의 디코더에 대한 격자 트리의 추가적인 실시예를 도시하는 도면.

도 10은 본 개시 내용의 디코더에 대한 격자 트리의 또 다른 실시예를 도시하는 도면.

도 11a 및 11b는 본 개시 내용의 디코더에 대한 격자 트리에 관한 실시예의 블록도.

도 12a 및 12b는 본 개시 내용의 디코더에 대한 사후 계산 블록에 관한 실시예의 블록도.

도 13은 본 개시 내용의 바이트 코드 디코더에 관한 실시예의 블록도.

도 14는 본 개시 내용의 바이트 코드 디코더의 다른 실시예의 블록도.

도 15는 본 개시 내용의 디코더에 대한 태너 그래프(tanner graph)에 관한 실시예를 도시하는 도면.

도 16은 본 개시 내용의 바이트 코드 디코더의 추가적인 실시예의 블록도.

도 17은 본 개시 내용의 디코딩 프로세스에 관한 실시예의 흐름도.

도 18은 본 개시 내용의 디코딩 프로세스의 다른 실시예의 흐름도.

발명을 실시하기 위한 구체적인 내용

[0015] 본 개시 내용의 특성들 및 장점들은 예로서 주어진 아래의 설명으로부터 보다 명확해질 수 있다.

[0016] 본 개시 내용의 하나 이상의 특정한 실시예가 아래에 기술될 것이다. 이러한 실시예들의 간결한 설명을 제공하기 위한 노력으로, 실제 구현예의 모든 특징들이 명세서에 기술되지는 않는다. 임의의 엔지니어링 또는 설계 프로젝트에서와 같이 임의의 이러한 실제 구현예의 개발에 있어서도, 구현예마다 달라질 수 있는 시스템 관련 및 사업 관련 제약들을 준수하는 것과 같은 개발자의 특정한 목표를 달성하기 위해 구현예에 특정된 다수의 결정이 이루어져야 함을 이해해야 한다. 더욱이, 그럼에도 불구하고 이러한 개발에 드는 노력은 본 개시 내용의 이점을 갖는 본 기술 분야의 당업자에게는 일상적인 설계, 조립 및 제조 업무일 것임을 이해해야 한다.

[0017] 아래의 설명은 텔레비전 방송 신호들과 관련되고, 특히 미국 내에서 사용하도록 정의되는 방송 신호들과 관련되는 시스템을 기술한다. 기술되는 실시예들은 이동, 핸드헬드, 또는 보행자 수신 장치들에서 사용될 수 있다. 사용되는 장치들의 예들은 셀룰러(cellular) 전화, 지능형 전화, PDA(Personal Digital Assistant), 랩톱 컴퓨터 및 휴대용 텔레비전을 포함하지만, 이에 한정되지 않는다. 다른 유형의 신호들을 송신 및 수신하는 데 활용되는 다른 시스템들은 유사한 구조들 및 프로세스들을 포함할 수 있다. 본 기술 분야의 당업자는 본 명세서에 기술된 회로들 및 프로세스들의 실시예들이 단지 잠재적인 실시예들 중 하나의 집합임을 이해할 것이다. A53 표준과 같은 방송 및 무선 표준들을 준수하는 신호들은 일반적으로 위성 링크, 동축 케이블, 또는 전화선 상의 통신을 포함하는, 공중파 외의 다른 방식으로 송신될 수 있음을 주목하는 것이 중요하다. 그러므로, 대안적인 실시예들에서, 시스템의 구성 요소들은 재배열 또는 생략될 수 있거나, 또는 추가적인 구성 요소들이 추가될 수 있다. 예컨대, 사소한 수정으로, 기술된 시스템은 해외에서 사용되는 서비스를 포함하는 다른 지상파 방송 서비스, 위성 비디오 및 오디오 서비스, 또는 전화 데이터 서비스에서 사용되도록 구성될 수 있다.

[0018] 아래에 기술된 실시예들은 주로 신호의 송신 및 수신과 관련된다. 소정의 제어 신호들 및 전력 공급 접속부들을 포함하지만 이에 한정되지 않는 본 실시예들의 소정의 태양들은 기술되거나 도면들에 도시되지 않았지만, 당업자에 의해 쉽게 확인될 수 있다. 본 실시예들은 마이크로프로세서 및 프로세서 코드 또는 커스텀(custom) 집적 회로들을 사용하는 것을 포함하여, 하드웨어, 소프트웨어, 또는 이 둘의 결합을 사용하여 구현될 수 있음에 주목해야 한다. 또한, 본 실시예들 중 다수는 반복 동작 및 본 실시예의 다양한 요소 사이의 접속을 수반함을 주목해야 한다. 본 명세서에 기술되는 반복 동작 실시예들에 연속하여 접속되거나, 이들을 대신하거나, 또는 그에 추가되는 반복되는 동일한 요소들을 이용하는 파이프라이닝(pipelining) 아키텍처들을 사용하는 대안적인 실시예들이 가능할 수 있다.

[0019] 이제 도 1을 살펴보면, 인코더(100)에 관한 실시예의 블록도가 도시된다. 인코더(100)는 특히 A53 송신 표준과 관련하여 견고(rugged)하고 강건(robust)한 데이터 스트림을 인코딩하고 송신하는 데 적합하다. 인코더(100)는 MPEG 운송 스트림 소스(102)를 포함한다. MPEG 운송 스트림 소스(102)는 몇몇 추가적인 블록을 포함하는 ATSC M/H 블록(110)에 접속된다. ATSC M/H 블록(110) 내에 포함된 블록들은 인입 데이터 스트림을 처리하고, 이동, 보행자 및 핸드헬드 장치들에 의한 수신 및 사용을 위해 적응된 견고한 데이터 스트림을 생성한다. 이러한 블록들은 아래에 더 기술될 것이다. ATSC M/H 블록(110)은 다중화기(mux)(130)에 접속된다. 다중화기(130)는 또한 레거시 ATSC A53 전용 인코딩에 사용하기 위한 운송 데이터 컨텐트를 수신한다. 다중화기(130)는 자신 안에 몇몇 추가적인 블록을 또한 포함하는 ATSC A53 레거시 블록(150)에 접속된다. ATSC A53 레거시 블록(150) 내의 블록들은 A53 신호 형식으로 기존의 방송 신호를 인코딩 및 송신하는 데 사용되는 블록들을 나타낼 수 있다. 이러한 블록들 또한 아래에 더 기술될 것이다. 제어기(170)는 다중화기(130) 및 MPEG 운송 스트림 소스(102)에 접속된다. 제어기(170)는 또한 인코더(100) 내의 다른 블록들에 접속될 수 있다.

[0020] ATSC M/H 블록(110) 내에서, 패킷 인터리버(packet interleaver)(112)는 패킷들 내에 배열된 데이터의 스트림을 수신한다. 각 패킷은 187 바이트를 포함하고, 패킷 식별에 사용되는 3 바이트 헤더를 포함한다. 패킷 인터리버(112)의 출력은 유한체(Galois Field; GF)(256) 직렬 연접 블록 코더(Serial Concatenated Block Coder; SCBC)(114)에 제공된다. GF(256) SCBC(114)의 출력은 패킷 디인터리버(packet deinterleaver)(116)에 접속된다. 패킷 디인터리버(116)의 출력은 운송 스트림 헤더 수정기(118)에 접속된다. 운송 스트림 헤더 수정기(118)의 출력은 사전 운송 패킷 삽입기(120)에 접속된다. 사전 운송 패킷 삽입기(120)는 다중화기(130)에 접속된다.

[0021] 패킷 인터리버(112)는 전형적으로 행들로 배열된 패킷들로서 수신된 데이터를 바이트들의 열들에 기초하여 패킷들의 행들로부터 코드워드(codeword)들로 재배열한다. 패킷 인터리버(112)는 고정된 개수의 연속 패킷으로부터 행 단위 순서로 바이트들을 취하여 이를 열 단위로 출력한다. 일 실시예에서, 패킷 인터리버(112)는 12행의 187 바이트 패킷을 읽어 들여 187행의 12 바이트 코드워드를 출력한다. 패킷 인터리빙의 결과, 제1 바이트들

전부가 함께 그룹화되고, 제2 바이트들 전부가 함께 그룹화되는 등이다. 패킷 인터리버(112) 내로 읽혀지는 패킷들의 개수는 소스 프레임을 구성하고, GF(256) SCBC(114)에서의 처리에 필요한 소스 코드워드들 또는 심볼들의 개수와 같다. 패킷 인터리버(112)의 규모는 포함된 메모리의 유형과 크기에 기초하여 바뀔 수 있음을 주목하는 것이 중요하다.

[0022] GF(256) SCBC(114)는 바이트 코드 인코더의 실시예이다. 특히, GF(256) SCBC(114)는 유한체(256) 공간 상에서 짧은 선형 블록 코드들을 사용하여 구현된다. 몇몇 가능한 성분 블록 코드들이 사용될 수 있다. 예컨대, 1/2 속도의 바이트 코드 인코더는 다음의 생성 행렬을 사용할 수 있다.

[0023] [수학식 1]

[0024] $$G = (1 \quad 2)$$

[0025] 2/3 속도의 바이트 코드 인코더는 다음의 생성 행렬을 사용할 수 있다.

[0026] [수학식 2]

[0027] $$G = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \end{pmatrix}$$

[0028] 생성 행렬은 단위 행렬 및 b개의 원소의 열(수학식 1 및 2의 마지막 열 내의 값들로서 표시됨)을 사용하여 형성된다. 다른 코드 속도들이 또한 사용될 수 있다.

[0029] 각 성분 코드에 대한 생성 행렬 내의 계수들은 변조 프로세스 및 전체 오류 정정 시스템 에 대한 블록 코드 인코딩의 관계에 기초하여 최적화되었음을 주목하는 것이 중요하다. 다른 계수들이 사용될 수 있다. 최적화는 특히 ATSC 8-VSB 변조에서 있어서 격자 코딩 및 비트 대 심볼 매핑을 고려하는데, 그 까닭은 이러한 태양들이 수신 및 복조 프로세스에서 만나게 되는 첫 태양들이기 때문이다.

[0030] GF(256) SCBC(114)는 단순 또는 연접 블록 코드 인코더 중 하나일 수 있다. 연접 블록 코드 인코더는 인터리빙 및 펑처링(puncturing)과 같은 다른 기능들을 포함할 수 있다. 연접 블록 코드 인코더의 실시예는 아래에 더 상세히 기술될 것이다. GF(256) SCBC(114)는 또한 복수의 인코딩 속도로 인코딩할 수 있고, 또한 도시되지 않은 속도 모드 제어기를 통해 속도 모드들을 전환할 수 있다. 바람직한 실시예에서, GF(256) SCBC(114)는 1/2 속도의 성분 코드, 12/26 속도의 연접 블록, 또는 24/208 속도의 연접 코드 중 하나를 사용하여 인입 데이터 스트림을 인코딩하도록 적응될 수 있다.

[0031] GF(256) SCBC(114)는 인터리버(112)로부터 출력된 열들을 따라 바이트들을 인코딩한다. 달리 말해, GF(256) SCBC(114)는 패킷 인터리버(112)에서의 처리를 통해 형성된 인터리버 행렬의 제2 차원을 따라 인코딩한다.

[0032] 패킷 디인터리버(116)는 GF(256) SCBC(114)에 의해 생성된 코드워드들의 인코딩된 스트림을 수신하고, 187 바이트 패킷의 재구성된 행들을 출력한다. 패킷 디인터리버(116)는 인코딩된 코드워드들을 열 단위 순서로 입력하고{각 열은 원래의 메시지 또는 시스템 바이트들 및 GF(256) SCBC(114)에서의 처리에 의해 추가된 중복 또는 비 시스템 바이트들을 포함함}, 바이트들을 행 단위 배열로 출력한다. 프로세스는 본질적으로 패킷 인터리버(112)에 대해 기술된 프로세스의 반대이다. 패킷 디인터리버(112)는 동일한 개수의 코드워드 열을 입력하는데, 이제 각 코드워드는 비 시스템 바이트들의 인코딩된 집합을 포함한다. 출력에서의 행들의 개수는 인코딩된 코드워드 길이에 대응한다. 예컨대, 12/26의 코드 속도로 26 행의 패킷들이 출력될 것이다.

[0033] MPEG 운송 스트림 헤더 수정기(118)는 시스템 및 비 시스템 패킷들의 그룹들을 포함하는 디인터리빙된 187 바이트의 패킷들을 수신하고, 각 패킷에 포함된 3 바이트의 식별자 헤더를 수정한다. 3 바이트는 패킷에 관한 정보를 나르는 데 사용되는 몇몇 다른 비트들 또는 비트들의 그룹들과 함께 프로그램 식별자(Program Identifier; PID)를 포함한다. ATSC M/H 인코딩된 패킷들을 정확하게 디코딩 및/또는 인식할 수 없지만 레거시 또는 A53 방송 신호를 수신할 수 있는 수신기들(예컨대 레거시 수신기들)의 가장 효율적인 동작을 유지하기 위해, ATSC M/H 패킷들 중 일부의 헤더들 내의 소정의 비트들이 수정될 수 있음을 주목하는 것이 중요하다.

[0034] 사전 추적 패킷 삽입기(a-priori tracking packet inserter)(120)는 미리 결정된 추적 패킷들을 견고한 데이터 스트림 내에 배치할 수 있다. 미리 결정된 패킷들은 이동, 보행자, 또는 핸드헬드 장치에서 사용되는 수신기와 같은, 견고한 데이터 스트림을 수신할 수 있는 수신기에 대해 완전히 또는 대부분 알려진 정보의 패킷들을 나타

낸다. 미리 결정된 패킷들은 신호 인코딩 및 송신 중 레거시 또는 기존 A53 인코딩 부분 동안에 생성된 격자 상태의 디코딩을 돕도록 수신기에서 사용된다. 미리 결정된 패킷들은 수신기의 등화기(equalizer) 부분에서의 수렴(convergence)을 도울 수 있다. 미리 결정된 패킷들은 레거시 수신기에서의 수신을 개선하고자 하는 것은 아니지만, 여전히 잠재적인 개선을 초래할 수 있음을 주목하는 것이 중요하다. 미리 결정된 추적 패킷들은 의사 난수 생성기 시퀀스(pseudo random number generator sequence)를 포함하는 훈련 시퀀스 프로세스들을 사용하는 다수의 방식으로 생성될 수 있다.

[0035] ATSC M/H 블록(110)에서 생성된 견고 또는 강인한 데이터 스트림은 다중화기(130)에 제공된다. 다중화기(130)는 또한 도시되지 않은 데이터 소스로부터 레거시 데이터 스트림을 수신한다. 일부 실시예들에서, 레거시 데이터 스트림은 MPEG 운송 스트림 소스(102)로부터 제공될 수 있음에 주목한다. 다중화기(130)는 강건한 인입 ATSC M/H 데이터 스트림 및 레거시 데이터 스트림의 부분들을 저장하기 위해 버퍼 메모리를 포함할 수 있다. 다중화기(130)는 견고한 ATSC M/H 데이터 스트림 및 레거시 데이터 스트림의 시간 다중화를 함께 생성하고, 다중화된 스트림을 ATSC A53 레거시 블록(150)에 제공한다.

[0036] 다중화기(130)는 제어기(170)에 의해 제어된다. 제어기(170)는 마이크로프로세서(microprocessor) 또는 마이크로컨트롤러(microcontroller)로서 구현되는 별개의 회로일 수 있다. 제어기(170)는 그 대신 MPEG 운송 스트림 소스(102)와 같은 다른 블록들 중 하나에 포함될 수 있다. 제어기(170)는 또한 전체 변조 및 송신 장치의 동작을 위해 사용되는 다른 제어기 내에 포함될 수 있다.

[0037] 레거시 ATSC 인코더(150)는 다중화된 데이터 스트림으로서 제공되는 견고한 패킷들과 레거시 패킷들을 레거시 A53 표준을 준수하여 동일하게 인코딩한다. 데이터 무작위화기(data randomizer)(152)는 패킷을 무작위화하고, 패킷을 리드 솔로몬 인코더(154)에 제공한다. 데이터의 무작위화는 전형적으로 데이터의 패킷들을 수신기에게도 알려진 난수 시퀀스로 곱함으로써 수행된다. 리드 솔로몬 인코더(154)는 20 패리티 바이트를 계산하고 무작위화된 데이터에 연접시켜 207 바이트를 갖는 R-S 패킷을 생성한다.

[0038] 종래의 인터리버(156)는 R-S 패킷을 인터리빙하여 데이터를 시간으로 더 무작위화한다. 격자 인코더(158)는 인터리빙된 패킷을 인코딩하여 828개의 3비트 심볼의 블록을 생성한다. A53 표준은 12개의 격자 인코더를 사용하는 것을 지정하는데, 각 격자 인코더는 인터리빙된 패킷에 존재하는 매 두 비트에 대해 3비트 심볼을 생성하는 2/3 속도의 격자 인코더이다. 그 결과, 격자 인코더(158)는 역다중화기, 12개의 병렬 2/3 속도 격자 인코더 및 다중화기를 포함한다. 콘볼루션 인터리버(convolutional interleaver)(156)로부터의 데이터는 역다중화되어 12개의 격자 인코더에 분배되고, 12개의 격자 인코더에 의해 생성된 심볼들은 심볼들의 스트림으로 다중화된다.

[0039] 동기 삽입기(sync inserter)(160)는 4개의 미리 정의된 세그먼트 동기 심볼을 각 828 심볼 블록의 시작에 삽입하여 832 심볼 세그먼트(segment)를 생성한다. 또한, 동기 삽입기(160)는 생성되는 매 312개의 세그먼트에 대해 832개의 심볼을 포함하는 필드 동기(field sync)를 삽입한다. 특히, 필드 동기 심볼들은 312개의 세그먼트에 선행한다.

[0040] 8-VSB 변조기(162)는 8-VSB(Vestigal Sideband) 변조를 사용하여 반송파 신호를 변조하기 위해 격자 인코더(158)에 의해 인코딩된 데이터, 세그먼트 동기 심볼들 및 필드 동기를 포함하는 다중화된 심볼들을 사용한다. 특히, 8-VSB 변조기(162)는 펄스 진폭 변조(Pulse Amplitude Modulated; PAM) 신호를 생성한다. PAM 신호의 진폭은 8개의 이산 레벨 중 하나에 있을 것이고, 각 이산 레벨은 특정한 3 비트 심볼에 대응한다. PAM 신호는 도시되지 않은 회로를 사용하여 디지털로부터 아날로그 신호 형식으로 변환되고, 정확한 신호 펄스 형상을 생성하도록 필터링되며, 무선 주파수로 상향 변환된다.

[0041] MPEG 운송 스트림 소스(102) 또는 레거시 컨텐트를 위한 소스와 같은 송신 소스에 의해 생성되는 데이터는 ISO/IEC(International Standards Organization/International Electrotechnical Commission) 13818-2 형식과 또한 동등한 MPEG(Motion Picture Entertainment Group) 2 형식을 사용하여 소스 인코딩된 비디오를 포함한다. 데이터는 또한 돌비(Dolby) AC-3(Arc Consistency algorithm #3)을 사용하여 소스 인코딩된 오디오 데이터를 포함한다. A53 표준은 또한 프로그램 안내 데이터와 같은 다른 프로그램 요소들을 위한 메타데이터를 사용할 수 있도록 하고, 이러한 프로그램 요소들은 다른 방법들을 사용하여 소스 인코딩될 수 있다. 또한, A53 표준은 표준 해상도 인터레이스 텔레비전(standard definition interlaced television) 품질 및 프로그레시브 스캔 와이드스크린 고해상도 텔레비전(progressive scan widescreen high definition television) 품질을 포함하지만 이에 한정되지 않는 다양한 비디오 품질 레벨 및 디스플레이 형식으로 비디오를 송신할 수 있도록 한다.

[0042] 이제 도 2를 살펴보면, 연접 바이트 코드 인코더(200)에 관한 실시예의 블록도가 도시된다. 연접 바이트 코드

인코더(200)는 도 1에 기술된 GF(256) SCBC 인코더(114) 대신에 사용될 수 있고, 12/16 코드 속도를 사용하여 데이터 스트림 내에 코드워드들을 인코딩할 수 있도록 한다. 연접 바이트 코드 인코더(200)는 패킷들 또는 코드워드들을 수신하고, 이들은 제1 2/3 속도 바이트 인코더(202)에 제공한다. 제1 2/3 속도 바이트 코드 인코더(202)의 출력이 인터리버(204)에 제공된다. 인터리버(204)의 출력은 제2 2/3 속도 바이트 코드 인코더(206)에 제공된다. 제2 2/3 속도 바이트 코드 인코더(206)의 출력은 바이트 평처 블록(208)에 제공된다. 평처 블록(208)의 출력은 데이터 패킷화기(210)에 제공된다. 데이터 패킷화기(210)의 출력은 추가 처리(예컨대 앞서 도 1에 기술된 바와 같은 레거시 송신 인코딩)를 위해 제공될 수 있다.

[0043] 제1 2/3 바이트 코드 인코더(202)는 12 바이트의 컨텐트 데이터 패킷들을 수신하고 그 12 바이트로부터 제1 바이트 코드 인코딩된 스트림을 생성한다. 12 바이트 중 매 두 개의 데이터 바이트 M_A 및 M_B에 대해, 제1 바이트 코드 인코딩된 스트림은 바이트 M_A 및 M_B의 사본(duplicate) 및 수학식 2의 생성 행렬을 사용하여 계산되는 바이트 M_{AB}를 포함한다. 일부 실시예들에서, 컨텐트 데이터 바이트 M_A 및 M_B는 데이터 생성기{예컨대 도 1의 데이터 생성기(102)}에 의해 생성된 하나의 컨텐트 데이터 패킷의 바이트들이다. 다른 실시예들에서, 제1 2/3 속도 바이트 코드 인코더(202)는 두 개의 상이한 컨텐트 데이터 패킷 A 및 B로부터 각각 컨텐트 데이터 바이트 M_A 및 M_B를 선택한다. 컨텐트 데이터의 매 12 바이트에 대해, 18 바이트가 제1 바이트 코드 인코딩된 출력 스트림의 일부로서 출력된다.

[0044] 제1 바이트 코드 인코더(202)로부터의 바이트 코드 인코딩된 스트림은 인터리버(204)에 의해 인터리빙되어 18개의 인터리빙된 바이트를 포함하는 인터리빙된 스트림을 생성한다. 인터리버(204)뿐만 아니라 아래에 기술되는 다른 인터리버들은 본 기술 분야에 알려진 인터리빙 방식들(예컨대 의사 무작위, 행-열, 코드 최적화 등) 중 임의의 것을 사용할 수 있다. 또한, 인터리버들은 전체 인터리버 데이터 길이를 저장하기 위한 저장 용량을 갖는 메모리를 포함할 수도 있다.

[0045] 인터리빙된 스트림은 제2 2/3 속도 바이트 코드 인코더(206)에 제공된다. 제2 2/3 속도 바이트 코드 인코더(206)는 인터리빙된 스트림 내의 18개의 인터리빙된 바이트의 그룹들을 인코딩하여 27 바이트의 그룹들을 포함하는 제2 바이트 코드 인코딩된 스트림을 생성한다. 앞서 기술된 바처럼, 인터리버에 의해 생성된 매 두 바이트 M_A 및 M_B에 대해, 제2 2/3 바이트 코딩된 스트림은 두 바이트 M_A 및 M_B의 사본 및 생성된 중복 바이트 M_{AB}를 갖는다. 바이트 M_A는 데이터 생성기{예컨대 도 1의 데이터 생성기(102)}에 의해 생성된 컨텐트 데이터의 바이트들 중 하나의 사본일 수 있거나, 또는 제1 바이트 코드 인코더(202)에 의해 중복 또는 비 시스템 바이트로서 전개된 바이트일 수 있음이 명백하다. 유사하게, 바이트 M_B는 컨텐트 데이터의 바이트의 사본이거나, 제1 바이트 코드 인코더(202)에 의해 중복 또는 비 시스템 바이트로서 전개된 바이트일 수 있다.

[0046] 바이트 평처 블록(208)은 제2 바이트 코드 인코딩된 스트림 내의 27 바이트의 그룹으로부터 1 바이트를 제거하여 26 바이트의 그룹들을 포함하는 평처링된 스트림을 생성한다. 바이트 평처링은 주어진 코딩 구조에 대해 제공되고 송신되는 바이트들의 개수를 감소시킴으로써 데이터 효율을 개선하는 데 사용된다. 그러나, 개선된 데이터 효율은 데이터 스트림으로부터의 하나 이상의 인코딩된 바이트의 결여로 인해 수신기 내의 디코딩 회로에서의 결과적인 성능 저하에 의해 상쇄된다. 바이트 평처링은 또한 송신 형식을 위해 편리한 인코딩된 데이터의 바이트들 또는 패킷들의 그룹 또는 블록을 생성하는 데 사용될 수 있다. 바이트들 또는 패킷들의 소정의 그룹들에 기초한 코딩 구조들은 종종 블록 코드들이라고 지칭된다.

[0047] 패킷화기(210)는 평처링된 스트림으로부터의 바이트들을 187 바이트의 이산 패킷들로 결합하고 그룹화한다. 바이트 코드 인코더(200)의 구성 요소들에 의해 생성된 견고한 데이터 스트림은 12/26 속도 데이터 스트림을 생성한다. 바이트 코드 인코더(200)는 또한 바이트 평처 블록(208)이 사용되지 않는 경우 12/27 속도 데이터 스트림을 생성할 수 있다.

[0048] 앞서 기술된 12/27 속도 및 12/26 속도의 견고한 데이터 스트림들 이외의 견고한 데이터 스트림들을 생성하기 위해 연접 바이트 코드 인코더(600)와 유사한 연접 바이트 코드 인코더들이 이용될 수 있다. 예컨대, 17/26 및 12/52 속도와 같은 코드 속도들을 갖는 데이터 스트림들은 성분 바이트 코드 인코더들, 인터리버들 및 평처 블록들의 결합들을 통해 생성될 수 있다. 유사하게, 인터리버들 또는 평처 블록들의 다른 유형들 또는 배열들이 상기 기술된 실시예들에서 사용된 것들을 대체할 수 있다.

[0049] 이제 도 3을 살펴보면, 수신기에서 사용되는 디코더(300)에 관한 실시예의 블록도가 도시된다. 디코더(300)는 공중의 전자기파와 같은 송신 매체 상에서의 신호의 송신에 의해 악영향을 받은 신호들을 수신 및 디코딩하기

위한 회로 및 처리를 포함한다. 디코더(300)는 견고한 데이터 스트림은 물론 레거시 데이터 스트림도 디코딩할 수 있다. 예컨대, 디코더(300)는 ATSC M/H 신호로서 송신되는 신호를 수신하고 디코딩할 수 있는 수신기 내에 포함될 수 있다.

[0050] 디코더(300)에서, 최초의 처리에 이어 인입 신호가 등화기(310)에 제공된다. 등화기(310)는 격자 디코더(320)에 접속되는데, 이는 두 개의 출력을 제공한다. 격자 디코더(320)로부터의 제1 출력은 피드백을 제공하고, 피드백 입력으로서 다시 등화기(310)에 접속된다. 격자 디코더(320)로부터의 제2 출력은 콘볼루션 디인터리버(330)에 접속된다. 콘볼루션 디인터리버(330)는 역무작위화기(de-randomizer)(340)에 접속된다. 역무작위화기(340)의 출력은 바이트 코드 디코더(350)에 접속되는데, 이는 또한 두 개의 출력을 제공한다. 바이트 코드 디코더(350)로부터의 제1 출력은 콘볼루션 인터리버(360) 및 무작위화기(370)를 통해 피드백 입력으로서 다시 격자 디코더(320)에 접속된다. 바이트 코드 디코더(350)로부터의 제2 출력은 리드 솔로몬 디코더(380)에 접속된다. 리드 솔로몬 디코더(380)의 출력은 데이터 디코더(390)에 접속된다. 견고한 스트림 제어기(395)는 바이트 코드 디코더(350) 및 리드 솔로몬 디코더(380)에 접속된다. 등화기(310), 격자 디코더(320), 콘볼루션 디인터리버(330), 역무작위화기(340), 리드 솔로몬 디코더(380) 및 데이터 디코더(390)는 ATSC A53 레거시 방송 신호들을 수신하는 데 사용되는 종래의 수신기 내의 블록들과 유사한 방식으로 접속되고 기능적으로 동작한다는 점을 주목하는 것이 중요하다.

[0051] 수신기(도시되지 않음)의 전단(front end) 처리(예컨대 안테나, 동조기, 복조기, A/D 변환기)로부터의 입력 신호가 등화기(310)에 제공된다. 등화기(310)는 수신된 신호를 복구하기 위한 시도로서 송신 채널 효과를 완전히 또는 부분적으로 제거하도록 수신된 신호를 처리한다. 다양한 제거 또는 등화 방법이 본 기술 분야의 당업자에게 잘 알려져 있고, 본 명세서에서 논의되지 않을 것이다. 등화기(310)는 FFE(Feed Forward Equalizer) 구획과 DFE(Decision-Feedback-Equalizer) 구획을 포함하는 처리 회로의 복수의 구획을 포함할 수 있다. 등화기(310)는 또한 동기 제거 및 타이밍과 사전 훈련 패킷 처리를 위한 회로들을 포함할 수 있다.

[0052] 등화된 신호는 격자 디코더(320)에 제공된다. 격자 디코더(320)는 등화기(310)의 DFE 구획에 제공되는 판정값들의 집합을 하나의 출력으로서 생성한다. 격자 디코더(320)는 등화기(310)의 DFE 구획에 또한 제공되는 중간 판정값들도 생성할 수 있다. DFE 구획은 격자 디코더(320)로부터의 중간 판정값들과 함께 판정값들을 사용하여 등화기(310) 내의 필터 탭(filter tap)들의 값들을 조정한다. 조정된 필터 탭 값들은 수신된 신호에 존재하는 간섭 및 신호 반사를 소거한다. 반복 프로세스는 등화기(310)가 격자 디코더(320)로부터의 피드백의 도움을 받아 시간에 따라 잠재적으로 변화하는 신호 송신 환경 조건을 동적으로 조정할 수 있도록 한다. 반복 프로세스는 인입 신호의 데이터 속도(예컨대 디지털 텔레비전 방송 신호에 대한 19 Mb/s)와 유사한 속도로 일어날 수 있음을 주목하는 것이 중요하다. 반복 프로세스는 또한 인입 데이터 속도보다 높은 속도로 일어날 수 있다.

[0053] 격자 디코더(320)는 또한 격자 디코딩된 데이터 스트림을 콘볼루션 디인터리버(330)에 제공한다. 콘볼루션 디인터리버(330)는 예컨대 콘볼루션 바이트 인터리버(156)에 의해 인코더에서 수행되는 인터리빙의 반대 방식으로 데이터 패킷들 내에서 조직되는 디인터리빙된 패킷들을 생성한다. 디인터리빙된 패킷들은 역무작위화기(340)에 의해 역무작위화된다. 역무작위화기(340)는 데이터 무작위화기(152)와 같은 인코더 내의 무작위화기에 의해 사용되는 복소수 시퀀스의 보수(complement)로 패킷들을 곱함으로써 인코더에서 추가된 무작위화 컨텐트를 제거한다. 역무작위화된 패킷들은 바이트 코드 디코더(350)에 제공된다. 바이트 코드 디코더(350)는 전형적으로 견고한 또는 강건한 데이터 스트림으로부터의 패킷들만을 처리한다. 그 결과, 견고한 데이터 스트림의 일부가 아닌 패킷들은 단지 바이트 코드 디코더(350)를 통해 리드 솔로몬 디코더(380)에 전달된다. 바이트 코드 디코더(350) 내의 나머지 블록들에 제공되는 격자 디코딩된 스트림은 데이터 스트림에 포함된 정보에 대한 확률 또는 신뢰도의 형태일 수 있음을 주목하는 것이 중요하다.

[0054] 바이트 코드 디코더(350) 및 격자 디코더(320)는 반복적인 방식(터보 디코더라 일컬어짐)으로 동작하여 견고한 데이터 스트림을 디코딩한다. 특히, 격자 디코더(320)는 디인터리빙 및 역무작위화 후에 제1 연판정 벡터(soft decision vector)를 견고한 데이터 스트림에 포함된 패킷들의 각 바이트에 대해 바이트 코드 디코더(350)에 제공한다. 전형적으로, 격자 디코더 (320)는 확률값들의 벡터로서 연판정을 생성한다. 일부 실시예들에서, 벡터 내의 각 확률값은 벡터와 연관된 바이트가 가질 수 있는 값과 연관된다. 다른 실시예들에서, 확률값들의 벡터는 시스템 패킷 내의 매 반 니블(nibble)(즉, 2 비트)에 대해 생성되는데, 그 까닭은 2/3 속도의 격자 디코더가 2 비트 심볼을 추정하기 때문이다. 일부 실시예들에서, 격자 디코더(320)는 바이트의 4 개의 반 니블과 연관된 네 개의 연판정을 결합하여 바이트가 가질 수 있는 확률값들의 벡터인 하나의 연판정을 생성한다. 이러한 실시예들에서, 바이트에 대응하는 연판정들이 바이트 코드 디코더(350)에 제공된다. 다른 실시예들에서, 바이트 코드 디코더는 시스템 패킷의 바이트에 관한 연판정을 4 개의 연판정 벡터로 분리하는데, 4개의 연판정 각각은 바

이트의 반 니블과 연관된다.

[0055] 바이트 코드 디코더(350)는 견고한 데이터 스트림의 패킷들을 포함하는 바이트들과 연관된 연판정 벡터를 사용하여 패킷들을 포함하는 바이트들의 제1 추정치를 생성한다. 바이트 코드 디코더(350)는 시스템 및 비 시스템 패킷들을 둘 다 사용하여 견고한 스트림을 포함하는 패킷들의 각 바이트에 대해 제2 연판정 벡터를 생성하고, 콘볼루션 인터리버(360)에 의한 재인터리빙과 무작위화기(370)에 의한 재무작위화 후에 제2 연판정 벡터를 격자 디코더(320)에 제공한다. 콘볼루션 인터리버(360) 및 무작위화기(370)는 도 1에 기술된 콘볼루션 바이트 인터리버(156) 및 데이터 무작위화기(152)와 유사한 방식으로 기능적으로 동작한다. 이후 격자 디코더(320)는 제2 연판정 벡터를 사용하여 제1 판정 벡터의 추가적인 반복을 생성하는데, 이는 바이트 디코더(350)에 제공된다. 격자 디코더(320) 및 바이트 코드 디코더(350)는 격자 디코더 및 바이트 코드 디코더에 의해 생성된 연판정 벡터가 수렴하거나 또는 미리 결정된 횟수의 반복이 수행될 때까지 이러한 방식으로 반복한다. 이후, 바이트 코드 디코더(350)는 시스템 패킷들의 각 바이트에 대해 연판정 벡터 내의 확률값들을 사용하여 시스템 패킷들의 각 바이트에 대한 경판정(hard decision)을 생성한다. 격자 디코더(320)는 MAP(Maximum A Posteriori) 디코더를 사용하여 구현될 수 있고, 바이트 또는 반 니블(심볼) 연판정들 중 하나에 대해 동작할 수 있다.

[0056] 터보 디코딩은 전형적으로 블록들 사이에서 판정 데이터를 전달하는 것과 관련된, 인입 데이터 속도들보다 높은 반복 속도들을 활용한다는 점을 주목하는 것이 중요하다. 가능한 반복의 횟수는 데이터 속도 및 반복 횟수의 비율로 한정된다. 그 결과, 실용적인 정도로 터보 디코더에서의 더 높은 반복 속도는 일반적으로 오류 정정 결과를 개선시킨다. 일 실시예에서, 인입 데이터 속도의 8배인 반복 속도가 사용될 수 있다.

[0057] 도 3에 기술된 것과 같은 소프트 입력 소프트 출력 바이트 코드 디코더는 벡터 디코딩 기능들을 포함할 수 있다. 벡터 디코딩은 시스템 및 비 시스템 바이트들을 포함하는 데이터의 바이트들을 그룹화하는 것을 수반한다. 예컨대, 1/2 속도로 바이트 코드 인코딩된 스트림에 대해, 1개의 시스템 바이트 및 1개의 비 시스템 바이트가 그룹화될 것이다. 두 바이트는 64,000개가 넘는 가능한 값을 갖는다. 벡터 디코더는 두 바이트의 가능한 값들 각각에 대한 확률을 결정 또는 추정하고, 확률 지도를 생성한다. 확률들의 일부 또는 전부의 가중치 및 가능한 코드워드에 대한 유클리드 거리(Euclidean distance)에 기초하여 연판정이 이루어진다. 유클리드 거리의 오차가 문턱값보다 작은 경우, 경판정이 이루어질 수 있다.

[0058] 바이트 코드 디코더(350)의 하드 입력(즉, 바이트 코드 인코딩된 정보)은 리드 솔로몬 디코더(380)에 제공된다. 리드 솔로몬 디코더(380)는 출력 데이터를 예컨대 207 바이트의 패킷들로 만든다. 리드 솔로몬 디코더(380)는 바이트 코드 디코더(350)에 의해 생성된 207 바이트의 시퀀스 각각을 하나 이상의 리드 솔로몬 코드워드로 간주하며, 코드워드들 또는 패킷들 내의 임의의 바이트가 송신 중의 오류로 인해 손상되었는지 여부를 결정한다. 이러한 결정은 코드워드들에 대한 신드롬(syndrome)들 또는 오류 패턴들의 집합을 계산하고 평가함으로써 종종 수행된다. 손상이 검출되면, 리드 솔로몬 디코더(380)는 패리티 바이트들 내에 인코딩된 정보를 사용하여 손상된 바이트들의 복구를 시도한다. 이후, 결과적인 오류 정정된 데이터 스트림은 데이터 디코더(390)에 제공되는데, 이는 송신되고 있는 컨텐트의 유형에 따라 데이터 스트림을 디코딩한다.

[0059] 데이터 디코더(390)는 디코딩된 패킷의 헤더 내의 PID와 같은 식별자를 사용하여 패킷 내에 운반되는 정보의 유형 및 이러한 정보를 어떻게 디코딩할지를 결정한다. 헤더 내의 PID는 데이터 스트림의 일부로서 주기적으로 송신되고 수신기에서 업데이트될 수 있는 PMT(Program Map Table) 내의 정보와 비교된다. 데이터 디코더(390)는 인식된 유형이 아닌 데이터 패킷들에 대한 PID를 갖는 어떠한 패킷도 무시한다.

[0060] 견고한 스트림 제어기(395)는 또한 오류 정정된 데이터 스트림을 수신한다. 견고한 스트림 제어기(395)는 마이크로프로세서 또는 마이크로컨트롤러로서 구현되는 별개의 회로일 수 있다. 견고한 스트림 제어기(395)는 그 대신 바이트 코드 디코더(350)와 같은 다른 블록들 중 하나에 포함될 수 있다. 견고한 스트림 제어기(395)는 또한 전체 수신 장치의 동작에 사용되는 제어기에 포함될 수 있다. 견고한 스트림 제어기(395)는 예컨대 견고한 데이터 스트림에 사용되는 프리앰블(preamble)의 형태로 사전 운송 패킷들이 존재하는지를 결정할 수 있다. 사전 운송 패킷들의 존재에 기초하여, 견고한 스트림 제어기(395)는 견고한 데이터 스트림 내의 제어 정보를 식별하고 디코딩한다.

[0061] 도 3에 기술된 것과 같은 디코더들은 앞서 기술된 바이트 코드 인코더들에 의해 인코딩(단순한 바이트 코드 인코더들 또는 연접 바이트 코드 인코더들에 의한 인코딩을 포함함)된 견고한 데이터 스트림을 디코딩할 수 있다. 도 3의 디코더는 단일 인코딩 단계만을 수반하는 단순한 또는 성분 바이트 코드 인코더에 의해 인코딩된 견고한 데이터 스트림의 디코딩을 기술한다. 연접 바이트 코드 디코딩은 디인터리빙, 디펑처링(de-puncturing) 및 재삽입과 같은 중간 처리에 부가하여 인입 코드워드들 또는 바이트들을 둘 이상의 디코딩 단계에서 디코딩하는 것

을 포함한다.

[0062] 이제 도 4를 살펴보면, 연접 바이트 코드 디코더(400)에 관한 실시예의 블록도가 도시된다. 연접 바이트 코드 디코더(400)는 도 3의 바이트 코드 인코더(350) 대신에 사용될 수 있다. 연접 바이트 코드 디코더(400)는 터보 디코더 구성에서 동작하도록 구성될 수 있다. 연접 바이트 코드 디코더(400)는 또한 반복 프로세스를 사용하는 터보 디코더로서 내적으로 동작하여 견고한 데이터 스트림 내의 연접 바이트 코드 인코딩된 패킷들을 디코딩한다. 연접 바이트 코드 디코더(400)는 12/26 속도의 바이트 코드 인코딩된 신호 스트림을 디코딩하여 원래 인코딩된 26 바이트로부터 12 바이트의 데이터를 생성하도록 적응된다.

[0063] 전형적으로 바이트 삽입 블록(402)에 제공되는 26 바이트의 연판정 값들을 나타내는 인입 메시지 데이터 스트림이 바이트 삽입 블록(402)에 제공된다. 바이트 삽입 블록(402)의 출력은 제1 2/3 속도 바이트 코드 디코더(404)에 접속된다. 제1 2/3 속도 바이트 코드 디코더(404)는 두 개의 출력을 제공한다. 제1 출력은 펑처 블록(406)에 접속되는데, 펑처 블록(406)의 출력은 인터리버를 통해 피드백 입력으로서 격자 디코더에 접속된다. 제1 2/3 속도 바이트 코드 디코더(404)의 제2 출력은 디인터리버(408)에 접속된다. 심볼 디인터리버(408)의 출력은 또한 두 개의 출력을 갖는 제2 2/3 속도 디코더(410)에 접속된다. 제1 출력은 인터리버(412)를 통해 피드백 입력으로서 제1 2/3 속도 바이트 코드 디코더(404)에 접속된다. 제2 출력은 리드 솔로몬 디코더와 같은 다른 처리 블록들에 접속된다.

[0064] 바이트 삽입 블록(402)에 대한 26 바이트 입력은 데이터의 시스템 바이트들 또는 시스템 패킷들에 관한 격자 디코더에 의해 생성된 제1 연판정 및 데이터의 비 시스템 바이트들 또는 비 시스템 패킷들에 관한 연판정을 포함한다. 데이터의 시스템 및 비 시스템 바이트들은 바이트 코드 인코딩된 패킷들로부터 유래할 수 있다. 2/3 속도 바이트 코드 디코더는 2 데이터 바이트를 디코딩하기 위해 3 바이트를 필요로 한다. 그러나, 원래의 연접 인코딩은 바람직하게는 비 시스템 바이트를 제거함으로써 코드워드를 27 바이트로부터 26 바이트로 감소시키도록 바이트를 제거하였다. 그 결과, 인코딩 프로세스에서 펑처링에 의해 제거되는 바이트를 대체하기 위해 하나의 바이트가 필요하다. 또한, 격자 디코더는 데이터 스트림 내의 펑처링된 바이트에 관한 어떠한 연판정도 생성하지 않는데, 그 까닭은 격자 디코더에 대한 입력 스트림이 그 바이트를 포함하지 않았기 때문이다. 그 결과, 펑처링된 바이트의 값이 동등한 확률을 가짐을 나타내는 연판정 값이 삽입된다. 바이트 삽입 블록(402)으로부터의 삽입된 연판정 값을 포함하는 제1 연판정이 제1 2/3 속도 바이트 코드 디코더(404)에 제공된다. 제1 2/3 속도 바이트 코드 디코더(404)는 시스템 및 비 시스템 패킷들의 바이트들의 디코딩에 기초하여 제2 연판정을 생성하도록 제1 연판정을 사용한다. 연판정의 생성은 예컨대 바이트들의 집합을 위의 수학식 2 및 3에 나타낸 것과 같은 바이트 코딩된 패킷을 전개시키는 데 사용된 원소 b_1 및 b_2의 값들의 역(inverse)으로 곱하는 것을 활용한다.

[0065] 제1 2/3 속도 바이트 코드 디코더로부터의 27 바이트 소프트 출력이 펑처 블록(506)에 제공된다. 27 바이트 소프트 출력은 제1 2/3 속도 바이트 코드 디코더에서의 디코딩 후의 시스템 및 비 시스템 바이트들 둘 다에 대한 연판정 값들의 업데이트된 집합을 나타낸다. 펑처 블록(506)은 바이트 형식을 격자 디코더에 의해 원래 처리된 26 바이트 형식으로 복귀시키기 위해 앞서 삽입된 연판정 바이트를 제거한다.

[0066] 시스템 바이트들만을 나타내는 제1 2/3 속도 바이트 코드 디코더로부터의 18 바이트 소프트 출력이 디인터리버(408)에 제공된다. 디인터리버(408)는 2/3 속도의 바이트 코드 인코딩 프로세스에서 수행된 인터리빙의 반대 방식으로 18 바이트의 데이터를 디인터리빙한다. 디인터리버(408)는 인코더에서의 인터리빙 맵을 정확히 반대로 한다. 디인터리빙된 바이트들은 제2 2/3 속도 바이트 코드 디코더(510)에 제공된다. 제2 2/3 속도 바이트 코드 디코더(410)는 디인터리빙된 연판정 시스템 바이트들을 사용하여, 앞서 기술된 것과 유사한 방식으로 연판정 바이트들의 두 개의 추가적인 출력을 생성한다. 18 바이트 소프트 출력이 인터리버(412)에 제공된다. 18 바이트 소프트 출력은 제1 2/3 속도 바이트 코드 디코더(404)에서의 디코딩으로부터의 시스템 및 비 시스템 바이트들 둘 다에 대한 연판정 값들의 업데이트된 집합을 포함하는 피드백 메시지를 나타낸다. 인터리버(412)는 디인터리빙된 바이트들을 제1 2/3 속도 바이트 코드 디코더에 의해 사용된 바이트 형식으로 되돌려 놓기 위해 디인터리빙된 바이트들을 재인터리빙한다. 인터리버(412)는 도 1의 인터리버(104)와 같은 인코더에서서 사용된 인터리버와 본질적으로 동일하고, 18 바이트의 재인터리빙된 집합을 제1 2/3 바이트 코드 디코더(404)에 제공한다. 18 바이트의 재인터리빙된 집합은 제1 2/3 속도 바이트 코드 디코더(404)에 의해 내려진 연판정들을 개선시키는 데 사용된다.

[0067] 제2 2/3 속도 바이트 코드 디코더(410)로부터의 12 바이트 출력은 시스템 바이트들을 12/26 속도로 바이트 코드 인코딩된 견고한 데이터 스트림에 대한 완전히 디코딩된 데이터 출력으로서 나타낸다. 바람직한 실시예에서,

제2 2/3 속도 바이트 코드 디코더(410)에 의해 생성된 12개의 시스템 출력 바이트에 대한 연판정이 확정적인 경우, 또는 정확한 데이터 값들로서 확정적인지에 관한 미리 결정된 문턱값 내에 있는 경우, 제2 2/3 속도 바이트 코드 디코더(410)는 연판정을 사용하여 12개의 출력 바이트에 관한 경판정을 생성하고, 12개의 바이트를 리드 솔로몬 디코더와 같은 추가 처리 블록들에 제공한다. 그러나, 제2 2/3 속도 바이트 코드 디코더에 의해 생성된 연판정이 확정적이지 않은 경우, 이전의 반복 중에 전개되고 피드백된 소프트 정보를 사용하여 위에서와 같이 추가적인 반복이 전개된다. 이러한 추가적인 소프트 정보는 각 소프트 디코더에게 자신의 후속 디코더에 의해 제공된다. 즉, 격자 디코더는 제1 2/3 속도 바이트 코드 디코더(404)로부터 펑처 블록(406)을 통해 제공되는 피드백 메시지 신호를 사용하고, 제1 2/3 속도 바이트 코드 디코더(404)는 제2 2/3 속도 바이트 코드 디코더(410)로부터 인터리버(412)를 통해 제공되는 피드백 메시지 신호를 사용한다. 제2 2/3 속도 바이트 코드 디코더(410)가 충분히 수렴할 때까지 또는 미리 결정된 횟수의 반복이 수행될 때까지 이러한 방식으로 반복이 계속된다. 터보 디코딩은 전형적으로 블록들 사이에서 판정 데이터를 전달하는 것과 관련된, 인입 데이터 속도들보다 높은 반복 속도들을 활용한다.

[0068] 이제 도 5를 살펴보면, 성분 바이트 코드 디코더(500)에 관한 실시예의 블록도가 도시된다. 성분 바이트 코드 디코더(500)는 도 3에 도시된 바이트 코드 디코더(350) 또는 도 4에 도시된 디코더(404)및 디코더(410)로서 사용될 수 있다. 바이트 코드 디코더(500)는 두 개의 입력 신호를 수신한다. 제1 입력 신호는 격자 디코딩 블록으로부터 주로 오는 인입 메시지 신호이다. 제2 입력은 후속하는 디코딩 단의 출력으로부터 피드백되는 피드백 메시지 신호이다. 각 입력은 가산기(510)에 접속된다. 가산기(510)의 출력은 코어 유닛(core unit)(520)에 제공된다. 코어 유닛(520) 내에서, 가산기(510)의 출력은 외부(extrinsic) 계산 블록(522)에 접속되고, 또한 가산기(524)의 하나의 입력에 접속된다. 외부 계산 블록(522)의 출력은 가산기(524)의 제2 입력에 접속된다. 가산기(524)의 출력은 코어 유닛(520)의 출력을 나타내고, 가산기(530) 및 가산기(540)의 하나의 입력으로서 제공된다. 원래의 입력 메시지는 또한 반전 신호로서 가산기(530)의 제2 입력에 제공된다. 피드백 메시지는 반전 신호로서 가산기(530)의 제2 입력에 제공된다. 가산기(530) 및 가산기(540)의 출력들은 성분 디코더(500)의 출력들을 나타내고, 도 4의 디코더(404) 및 디코더(410)에 대해 기술된 출력들과 유사하다.

[0069] 가산기(510)는 들어오는 코딩된 메시지 신호를 코딩되지 않은 피드백 메시지와 결합한다. 가산 기능은 전형적으로 특정한 비트들, 보다 구체적으로는 메시지 신호 내의 비트들 각각에 대한 특정한 비트 신뢰도에 기초하여 신호들을 결합하는 것을 수반한다. 코어 유닛(520)은 결합된 메시지 신호를 수신하고, 가산기(510)에 의해 공급되는 내부(intrinsic) 신뢰도 정보를 사용하여 인입 비트들에 대해 외부 계산 블록(522) 내의 외부 정보를 계산한다. 일 실시예에서, 각각의 인입 비트 신뢰도는 앞서 기술된 바와 같이 유클리드 거리 기법들 및 생성 행렬로부터의 역의 값들을 사용하여 처리된다. 외부 계산 유닛(522)에서 계산된 외부 정보는 가산기(510)로부터의 입력 내부 정보와 함께 가산기(524)에서 결합 또는 가산되어 사후 정보를 생성한다. 코어 유닛(520)에 의해 생성된 사후 정보는 원래의 신호 메시지, 피드백 메시지 및 계산된 외부 정보의 결합 또는 합을 나타낸다.

[0070] 가산기(540)는 입력으로부터 공급된 피드백 메시지 신호를 감산 프로세스를 통해 사후 정보와 결합한다. 바람직한 실시예에서, 피드백 메시지 신호는 가산기(540)에 진입하기 전에 반전된다. 결과적인 메시지 신호는 코어 유닛(520)에서 계산된 추가된 외부 정보와 함께 입력으로부터의 원래의 코딩된 메시지를 나타낸다. 정보 메시지 신호들은 전형적으로 메시지 내의 시스템 및 비 시스템 정보에 대한 확률 또는 신뢰도의 집합으로서 성분 바이트 디코더(500)로부터 및 성분 바이트 디코더(500)로 전달된다는 점을 주목하는 것이 중요하다. 바람직한 일 실시예에서, 외부 계산 블록(522)은 로그 계산을 수행하고, 가산기(540)의 출력에서의 결과적인 신뢰도들은 로그 우도비(log likelihood ratio; LLR)들로서 표현된다. 인입 메시지는 또한 로그 우도비들의 집합의 형태로 수신될 수 있다. 출력 메시지 신뢰도들은 메시지 내의 비트들에 기초하여 연접 바이트 코드 디코더 내의 후속 단에 제공되거나, 또는 그 대신 반복 바이트 코드 디코딩 프로세스의 최종 출력을 나타내고 후속하는 별개의 디코더에 제공된다.

[0071] 유사하게, 가산기(530)는 감산 프로세스를 통해 입력 메시지 신호를 사후 정보와 결합한다. 바람직한 실시예에서, 입력 메시지 신호는 가산기(530)에 진입하기 전에 반전된다. 결과적인 신호는 추가된 외부 정보와 함께 피드백 메시지 신호를 나타낸다. 위에서와 같이, 상기 정보는 확률 또는 신뢰도의 집합으로서 전달된다. 코딩된 메시지 신뢰도들의 집합은 피드백 신호로서 연접 디코더 내의 이전의 바이트 코드 디코더에 공급되거나, 또는 그 대신 피드백 신호로서 추가적인 반복 디코딩 개선을 위해 격자 디코더와 같은 이전의 디코딩 블록에 공급된다.

[0072] 성분 바이트 코드 디코더(500)는 비트 단위, 반 니블 단위, 또는 바이트 단위 방식으로 적용된 바이트 코드 인코딩에 기초하여 입력 메시지 신호를 반복적으로 처리할 수 있다. 앞서 기술된 바처럼, 반복 횟수는 비트값들

의 신뢰도 개선을 가능하게 한다. 반복 프로세스들의 개수는 하드웨어 크기 또는 하드웨어 속도에 의해 제한될 수 있다. 따라서, 보다 효율적인 아키텍처들 및 프로세스들이 바람직하다. 개선된 디코딩 아키텍처들은 앞서 기술된 직렬화된 직접 디코딩 프로세스 대신에 유한체 공간과 연관된 속성들을 활용함으로써 실현될 수 있다. 성분 바이트 코드들은 유한체 GF(256) 상에서 정의된 짧은 선형 블록 코드들임에 주목하는 것이 중요하다. 유한체 GF(256)는 GF(2)의 확장이고, 원시 다항식에 의해 더 정의될 수 있다. 하나의 바람직한 원시 다항식은 다음과 같다.

[0073] [수학식 3]

[0074] $p(x)=p_0+p_1x+..+p_8x^8=1+x^2+x^3+x^4+x^8 \quad \{p_i \in GF(2)\}$

[0075] 송신되는 코딩된 메시지는 다음과 같이 정의될 수 있다.

[0076] [수학식 4]

[0077] $c=mG$

[0078] 여기서 G는 코드 속도 1/2 및 2/3에 대해 수학식 1 및 2에 의해 각각 기술된 바와 같이 GF(256)에서 정의될 수 있다. 그러나, 생성 행렬은 코드 속도 1/2에 대한 원시 다항식에 기초하여 GF(2)에서 다음과 같이 재정의될 수 있다.

[0079] [수학식 5]

[0080] $G=[I \; S]$

[0081] 수학식 5에서, I는 8x8 크기의 단위 행렬이고, S는 다음과 같이 GF(2)에서 정의되는 부분 행렬이다.

[0082] [수학식 6]

[0083]
$$S=\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

[0084] 유사하게, 코드 속도 2/3에 대해, 생성 행렬은 원시 다항식에 기초하여 다음과 같이 GF(2)에서 정의될 수 있다.

[0085] [수학식 7]

[0086]
$$G=\begin{bmatrix} I & 0 & S \\ 0 & I & S \end{bmatrix}$$

[0087] 위에 정의된 행렬 등식들은 또한 GF(2)에서 패리티(parity) 검사 행렬을 생성하여 코딩된 메시지로부터 원래의 메시지를 복구하도록 할 수 있다. 패리티 검사 등식은 다음과 같이 쓸 수 있다.

[0088] [수학식 8]

[0089] $cH^T=0$

[0090] 1/2 코드 속도에 대한 패리티 검사 행렬은 다음과 같이 정의될 수 있다.

[0091] [수학식 9]

[0092] $H=[S^T \; I]$

[0093] 따라서, H는 다음과 같이 행렬 형태로 쓸 수 있다.

[0094] [수학식 10]

[0095]
$$H_{C=1/2}=\begin{bmatrix}0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0\\0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0\\1&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0\\1&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\1&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0\\1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\end{bmatrix}$$

[0096] 유사하게, 2/3 코드 속도에 대한 패리티 행렬은 다음과 같이 정의될 수 있다.

[0097] [수학식 11]

[0098] $H=[S^T\ S^T\ I]$

[0099] 따라서, 2/3 코드 속도에 대한 H는 다음과 같이 쓸 수 있다.

[0100] [수학식 12]

[0101]
$$H_{C=2/3}=\left[\begin{array}{cccccccc|cccccccc|cccccccc}0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0\\0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0\\1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0\\1&0&0&0&0&1&0&0&1&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\1&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0\\1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\end{array}\right]$$

[0102] 패리티 행렬은 행렬 등식을 통한 처리를 위해 비트들을 배열하거나 그룹화함으로써 직접적으로 수신기에서 인입 바이트 코드 코딩된 메시지에 적용될 수 있음을 주목하는 것이 중요하다. 또한, 패리티 행렬은 종종 패리티 검사 등식들로 일컬어지는 등식들의 집합으로 환원될 수 있는데, 여기서 각 등식은 인입 메시지의 소정의 바트들만을 수반한다. 상기 등식들은 메시지의 그룹 또는 부분 집합 내의 비트들에 대한 최종값들 또는 우도들을 결정하기 위해 연립 방정식들의 집합으로서 처리되고 풀릴 수 있다. 인입 비트들을 패리티 등식들로 또는 그 대신 패리티 행렬로 배열하는 것을 활용하는 디코더들, 그리고 특히 바이트 코드 디코더들은 아래에 기술되는 실시예들을 통해 보다 명확해질 것이다.

[0103] 이제 도 6을 살펴보면, 본 개시 내용의 태양들을 사용하는 성분 바이트 코드 디코더(600)에 관한 다른 실시예의 블록도가 도시된다. 성분 바이트 코드 디코더(600)는 도 4의 디코더(404) 및 디코더(410) 대신에 사용될 수 있다. 두 개의 입력 신호, 인입 메시지 신호 및 피드백 메시지 신호가 가산기(605)에 대한 입력들로서 제공된다. 가산기(605)의 출력은 입력 비트 재정렬 블록(615)에 접속된다. 입력 재정렬 블록(615)의 제1 출력은 지연 버퍼(620) 및 알파(alpha) 격자 블록(625)에 접속된다. 입력 재정렬 블록(615)의 제2 출력은 베타 격자 블록(630)에 접속된다. 알파 격자 블록(625)의 출력은 메모리 버퍼(635)에 접속된다. 베타 격자 블록(630)의 출력은 메모리 버퍼(640)에 접속된다. 지연 버퍼(620), 메모리 버퍼(635) 및 메모리 버퍼(640)의 출력들은 모두 사후 프로세서(645)에 접속된다. 사후 프로세서(645)의 출력은 출력 재정렬 블록(650)에 접속된다. 출력 재정렬 블록(650)의 출력은 가산기(655) 및 가산기(660)의 하나의 입력에 접속된다. 코딩된 메시지 신호가 또한 지연 버퍼(610)에 접속된다. 지연 버퍼(610)의 출력은 반전되어 가산기(655)의 다른 입력에 접속된다. 피드백 메시지 신호는 또한 지연 버퍼(612)에 접속된다. 지연 버퍼(612)의 출력은 반전되어 가산기(660)의 다른 입력에 접속된다. 가산기(655) 및 가산기(660)의 출력들은 도 5의 성분 바이트 코드 디코더(500)에 대해 기술된 것들과 유사한 성분 디코더(600)의 출력들을 나타낸다.

[0104] 가산기(605)는 두 개의 입력들, 입력 메시지 신호 및 피드백 메시지 신호를 수신하고, 신호들에 대한 신뢰도들을 결합 또는 추가한다. 입력 재정렬 블록(615)은 결합된 메시지 신호를 비트들의 부분 집합으로 그룹화하고, 신호 내의 비트들의 순차적인 순서를 변경하며, 재정렬된 입력 신호들을 알파 격자 블록(625) 및 베타 격자 블록(630)에 제공한다. 재정렬은 디코더의 효율을 개선하기 위해 수행될 수 있고, 아래에 더 상세히 기술될 것이다. 선택된 부분 집합의 크기는 사용되는 패리티 검사 프로세스의 함수일 수 있다. 바람직한 실시예에서, 부분 집합 크기는 16 비트이고, 벡터 b=[b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12} b_{13} b_{14} b_{15}]로서 식별된다. 입력 재정렬 블록(615)은 순방향으로(처음의 재정렬된 비트로부터 마지막의 재정렬된 비트로) 재정렬된 출력을 생성하는데, 이는 알파 격자 블록(625)에 제공되고 버퍼(620)에도 제공된다. 버퍼(620)는 내부 신호를 또한 나타내는 순방향 재정렬 출력의 타이밍 지연을 제공하여 격자 처리 후의 재결합을 가능하게 한다. 입력 재정렬 블록(615)은 또한 역순(마지막의 재정렬된 비트로부터 처음의 재정렬된 비트로) 신호를 생성하고 이 신호를 알파 격자 블록(630)에 제공한다.

[0105] 알파 격자 블록(625) 및 베타 격자 블록(630)은 격자 트리(trellis tree) 또는 격자 도표(trellis diagram)로 알려진 계산의 집합을 생성 처리하는 것에 기초하여 순방향 척도(forward metric) 및 역방향 척도(reverse metric) 값들을 계산한다. 격자 트리 또는 격자 도표의 특정한 구현예 및 알파 격자 블록(625) 및 베타 격자 블록(630)의 동작이 아래에서 더 상세히 기술될 것이다. 순방향 버퍼(635) 및 역방향 버퍼(640)는 알파 격자 블록(635) 및 베타 격자 블록(640)의 출력들을 수신한다. 순방향 버퍼(635) 및 역방향 버퍼(640)는 재결합을 위한 타이밍 지연을 제공한다. 또한, 역방향 버퍼(640)는 베타 격자 블록(640)의 출력의 비트 순서를 반대로 한다.

[0106] 알파 격자 블록(625) 및 베타 격자 블록(630)으로부터의 순방향 및 역방향 척도값들은 사후 블록(645) 내의 내부 정보와 결합된다. 사후 블록(645)은 메시지 스트림 내의 비트들의 부분 집합에 대한 사후 신뢰도들을 계산한다. 출력 재정렬 블록(650)은 상기 부분 집합을 이것의 원래 순서로 되돌리기 위해 상기 부분 집합에 대한 사후 신뢰도 정보를 재정렬한다. 가산기(655) 및 가산기(660)는 피드백 메시지 및 입력 메시지를 재정렬된 사후 메시지 신호와 결합하는데, 피드백 메시지와 입력 메시지 각각은 감산을 위해 반전된다. 버퍼(610) 및 버퍼(612)는 사후 신뢰도 정보 출력과의 올바른 결합을 가능하게 하기 위해 입력 메시지 및 피드백 메시지의 필요한 타이밍 지연을 제공한다. 가산기(660)의 순방향 출력 및 가산기(655)의 코딩된 피드백 출력은 순방향 출력값 및 피드백 출력값을 나타내고, 도 5의 성분 바이트 코드 디코더(500)에 대해 기술된 출력들과 기능적으로 유사하다는 점을 주목하는 것이 중요하다.

[0107] 성분 바이트 코드 디코더(600)는 로그 MAP 디코더를 사용하여 인입 메시지를 디코딩하는데, 이는 메시지 내의 비트들의 재정렬된 부분 집합의 격자 디코딩에 기초한다. 하나의 가능한 격자 알고리즘은 BCJR(Bahl-Cooke-Jelinek-Raviv) 알고리즘을 활용한다. 격자 매핑을 사용하여 로그 MAP 디코딩을 적용하기 위해, 인입 메시지는 앞서 기술된 패리티 행렬로부터 형성된 패리티 등식들의 집합에 기초하여 등가 비트 격자 구조로 변환된다. 1/2 속도 코드에 대해, 아래의 번호가 매겨진 등식들의 집합이 사용될 수 있다.

[0108] (1) $b_1+b_8=0$

[0109] (2) $b_2+b_9=0$

[0110] (3) $b_3+b_{10}=0$

[0111] (4) $b_0+b_4+b_{11}=0$

[0112] (5) $b_0+b_5+b_{12}=0$

[0113] (6) $b_0+b_6+b_{13}=0$

[0114] (7) $b_7+b_{14}=0$

[0115] (8) $b_0+b_{15}=0$

[0116] 비트 기반 격자 알고리즘은 형성된 격자 트리에서 왼쪽으로부터 오른쪽으로 보았을 때 증가하는 깊이를 갖는 임의의 특정한 순서로 비트들을 처리하는 것을 가능하게 한다. 비트가 주어진 깊이에서 선택되면, 그 비트를 수반하는 모든 패리티 검사 등식들이 활성화된다. 주어진 패리티 검사 등식 내의 모든 비트들이 처리된 경우, 그

패리티 검사 등식은 더 이상 활성화되지 않는다. 주어진 깊이에 있는 패리티 검사 등식들의 현재 상태가 연접되어 n 비트 상태를 형성하는데, 여기서 n은 활성 패리티 검사 등식의 현재 개수이다. 활성 패리티 검사 등식들 중 일부는 동일한 상태에 있을 수 있고, 따라서 이러한 등식들에 대해서는 하나의 비트만이 필요하다. 모든 깊이에 걸쳐 등식들의 최대 개수 n이 가능한 한 적게 되는 방식으로 비트들을 처리하는 비트 정렬을 사용하여 효율적인 디코딩이 가능하다. 바람직한 비트 격자 트리가 도 7에 도시된다.

[0117] 도 7에서, 아래쪽 x축은 입력 비트 재정렬 블록(615)에서 구현된 비트 순서를 나타낸다. 순방향 순서가 축 상의 왼쪽으로부터 오른쪽으로 도시되는 반면, 역방향 순서는 오른쪽으로부터 왼쪽으로 도시되어 있다. 위쪽의 x축은 각각의 비트 천이(transition) 또는 상태 변화에서 사용되는 등식의 개수를 도시한다. 실선은 다음 비트가 0의 값 또는 예상값을 갖는 경우 이전 비트와 다음 비트 사이의 천이를 도시한다. 파선은 다음 비트가 1의 값 또는 예상값을 갖는 경우 이전 비트와 다음 비트 사이의 천이를 도시한다.

[0118] 2/3 코드 속도에 대한 비트 격자 트리는 24 비트의 부분 집합 크기에 기초하여 아래의 번호가 매겨진 패리티 검사 등식들의 집합을 사용하여 유사한 방식으로 형성될 수 있다.

[0119] (1) $b_1+b_9+b_{16}=0$

[0120] (2) $b_2+b_{10}+b_{17}=0$

[0121] (3) $b_3+b_{11}+b_{18}=0$

[0122] (4) $b_0+b_4+b_8+b_{12}+b_{19}=0$

[0123] (5) $b_0+b_5+b_8+b_{13}+b_{20}=0$

[0124] (6) $b_0+b_6+b_8+b_{14}+b_{21}=0$

[0125] (7) $b_7+b_{15}+b_{22}=0$

[0126] (8) $b_0+b_8+b_{23}=0$

[0127] 2/3 코드 속도에 대한 바람직한 격자 트리가 도 8에 도시되어 있고, 도 7에 도시된 격자 트리에 대해 기술된 것과 유사한 방식으로 기술될 수 있다.

[0128] 앞서 기술된 디코딩 프로세스에 관하여 대안적인 패리티 검사 배열들이 사용될 수 있음에 주목하는 것이 중요하다. 예컨대, 비트들의 보다 작은 부분 집합이 사용될 수 있다. 그 결과, 비트들 중 둘 사이의 단일 관계를 사용하여 비트들의 보다 작은 부분 집합을 디코딩하는 것이 가능할 수 있다. 비트들의 보다 작은 부분 집합들 및 패리티 관계의 형태를 한 단일 관계는 반복적으로 디코딩될 수 있거나, 또는 비트들이 부분 집합에 추가 및 제거될 수 있고 상이한 패리티 등식이 단일 관계로서 후속 디코딩 단계에서 사용될 수 있다.

[0129] 성분 바이트 코드 디코더(600)는 또한 비트들의 다른 그룹을 허용할 수 있다. 예컨대, 16 또는 24 비트의 부분 집합을 형성하기 위한 그룹에 추가하여, 수신된 송신 또는 변조 심볼들에 대한 비트들의 관계에 기초하여 비트들을 그룹화함으로써 제2 그룹이 형성될 수 있다. 인입 메시지 내의 비트들과 연관된 추가적인 내부 정보를 포함시킴으로써, 추가 그룹은 디코딩 프로세스의 추가적인 개선을 제공한다. 성분 바이트 코드 디코더는 메시지의 2 비트를 포함하는 각 심볼에 대한 성상(constellation) 값들에 기초하여 메시지 정보를 수신하고 처리하여, 4개의 잠재적인 값을 더 표현한다. 하나의 심볼 내의 각 비트 성상 위치는 하나의 신뢰도 값에 의해 표현된다. 따라서, 2 비트로 이루어지는 하나의 심볼은 다음과 같이 식별되는 4개의 확률로서 표현될 수 있다.

[0130] $P(x_{symbol}="00")$, $P(x_{symbol}="01")$, $P(x_{symbol}="10")$, $P(x_{symbol}="11")$

[0131] 비트 대 심볼 관계 및 매핑과 후속 심볼 척도값 생성은 도 3의 격자 디코더(320)와 같은 이전의 처리단에서 수행될 수 있음을 주목하는 것이 중요하다. 이후 심볼 척도값들 및 심볼 매핑은 성분 바이트 코드 디코더(600)의 입력에 제공될 수 있다. 비트 대 심볼 매핑 및 척도값 생성은 또한 가산기 블록(605) 또는 입력 재정렬 블록(610), 또는 도시되지 않은 별개의 처리 블록에서와 같은 바이트 코드 디코더(600)에서의 입력 처리의 일부로서 수행될 수 있다. 또한, 심볼 확률은 앞서 기술된 바와 같은 메시지 내의 비트 확률에 대해 기술된 것과 동일한 방식으로 바이트 코드 디코더(600)로부터 유지, 처리 및 출력될 수 있다.

[0132] 대응하는 심볼 격자는 중간 천이를 고려함으로써 동일한 심볼의 비트들에서의 깊이를 단일 깊이로 붕괴

(collapse)시킴으로써 형성된다. 심볼 s_n의 2진 표현이 "$b_{2n}b_{2n+1}$"인 경우, 예컨대 심볼 s_0=3="11"은 상태 0으로부터 상태 1을 거쳐 상태 3으로 천이를 야기하고, 심볼 s_0=2="10"은 상태 0으로부터 상태 1을 거쳐 상태 2로 천이를 야기하고, 심볼 s_0=1="01"은 상태 0으로부터 상태 0을 거쳐 상태 1로 천이를 야기하고, 심볼 s_0=0="00"은 상태 0으로부터 상태 0을 거쳐 상태 0으로 천이를 야기한다. 이 패턴은 모든 가능한 상태에 대한 전체 심볼 격자가 형성될 때까지 계속된다.

[0133] 1/2 코드 속도 심볼 기반 격자 디코딩에 대한 바람직한 격자 트리가 도 9에 도시된다. 도 9에서 아래쪽 축은 S_0 내지 S_7로 표시된 재정렬된 심볼들 및 각 심볼과 연관된 대응 비트들을 도시한다. 앞서 기술된 바처럼, 순방향 순서가 왼쪽으로부터 오른쪽으로 도시되고 역방향 순서가 오른쪽으로부터 왼쪽으로 도시된다. 위쪽 축은 각 심볼 천이 또는 상태 변화에서 사용되는, 1/2 코드 속도 비트 기반 격자 트리에 대해 앞서 기술된 패리티 검사 등식의 개수를 도시한다. 실선은 다음 비트가 "00"의 값 또는 예상값을 갖는 경우 이전 비트와 다음 비트 사이의 천이를 도시한다. 파선은 다음 비트가 "01"의 값 또는 예상값을 갖는 경우 이전 비트와 다음 비트 사이의 천이를 도시한다. 점선은 다음 비트가 "10"의 값 또는 예상값을 갖는 경우 이전 비트와 다음 비트 사이의 천이를 도시한다. 마지막으로, 쇄선은 다음 비트가 "11"의 값 또는 예상값을 갖는 경우 이전 비트와 다음 비트 사이의 천이를 도시한다.

[0134] 유사하게, 2/3 코드 속도 심볼 기반 격자 디코딩에 대한 바람직한 격자 트리가 24 비트의 비트 그룹 부분 집합을 사용하여 도 10에 도시된다. 아래쪽 축은 S_0 내지 S_{12}로 표시된 재정렬된 심볼들 및 각 심볼과 연관된 대응 비트들을 도시한다. 위쪽 축은 각 심볼 천이 또는 상태 변화에서 사용되는, 2/3 코드 속도 비트 기반 격자 트리에 대해 앞서 기술된 패리티 검사 등식의 개수를 도시한다. 선 천이는 도 9와 동일하게 표시되어 있다.

[0135] 이제 도 11a 및 11b를 살펴보면, 본 개시 내용의 태양들을 사용하는 성분 바이트 디코더에서 사용되는 격자 블록(1100)에 관한 실시예의 블록도가 도시된다. 격자 블록(1100)은 도 6의 알파 격자 블록(625) 또는 베타 격자 블록(630) 중 하나를 구현하는 데 사용될 수 있다. 알파 격자 블록(625)은 아래에서 더 기술되는 베타 격자 블록(630)으로부터의 상이한 제어 시퀀스를 사용할 수 있다. 또한, 아래에 기술되는 바처럼, 각 블록은 격자 트리의 평가에서 상이한 척도 계산을 구현할 수 있다.

[0136] 격자 블록(1100)에서, 격자 블록(1100)에 제공되는 메시지 신호는 입력 선택 블록(1110)에 접속된다. 일련의 가산기(1125a 내지 1125p)는 입력 메시지의 선택된 부분들(즉 비트들 또는 심볼들) 및 피드백 선택 블록(1120)을 통해 선택되는 척도 계산 출력의 부분들(즉 비트들 또는 심볼들)을 결합한다. 척도 계산은 코어 프로세스 유닛들(1130a 내지 1130h 및 1135a 내지 1135d)에서 수행된다. 척도 계산은 알파 격자 블록(625) 및 베타 격자 블록(630)에 대해 구현된 바처럼 아래에서 더 기술될 것이다.

[0137] 코어 프로세스 유닛들(1130a 내지 1130h 및 1135a 내지 1135d)로부터의 계산된 척도들 각각은 출력 다중화기(1140)에 접속된다. 출력 다중화기(1140)로부터의 출력들은 D 버퍼들(1150a 내지 1150h)을 통해 버퍼링된다. D 버퍼들(1150a 내지 1150h)은 계산된 척도값들을 피드백 선택 블록(1120)에 제공한다.

[0138] 입력 선택 블록(1110), 피드백 선택 블록(1120) 및 출력 다중화기(1140)에 의해 수행되는 선택 및 다중화 동작들은 제어 블록(1170)에 의해 제어된다. 격자 블록(1100)은 또한 동기 클록 및 스트로브(strobe) 신호들, 사용중이 아닌 경우에 블록을 비활성화하기 위한 불능화 다중화기(1160), 및 제어 블록(1170)에 제공되는, 디코딩 코드 속도를 선택하기 위한 코드 속도 모드 입력 신호를 포함한다.

[0139] 알파 격자 블록(625)에 의해 수행되는 순방향 척도는 격자 트리의 좌측 상에서 시작하고, 우측으로 향하는 처리에 따라 진행한다. 알파 격자 블록(625)에 대한 격자 트리의 순방향 척도 계산은 다음과 같다.

[0140] [수학식 13]

[0141] $$\alpha_{n+1}^{k} = \log\left(e^{\alpha_n^i + L_n^j}\right)$$

[0142] 수학식 14에서, L_n^j는 n번째 심볼의 가지 j의 척도이다. 예컨대, "00"의 값을 갖는 심볼 n에 대한 척도 계산값은 L_n^0으로 표시된다. n번째 심볼의 상태 k의 척도는 α_n^k으로 표시된다. 도 9를 참조하면, 격자 트리의 순방향 척도는 다음과 같이 평가될 수 있다.

[0143] [수학식 14]

$$\alpha_0^k = 0 \forall k = 0,1..3$$
$$\alpha_1^0 = \log\left(e^{\alpha_0^0 + L_0^0}\right)$$
$$\alpha_1^1 = \log\left(e^{\alpha_0^0 + L_0^1}\right)$$
$$\alpha_1^2 = \log\left(e^{\alpha_0^0 + L_0^2}\right)$$
$$\alpha_1^3 = \log\left(e^{\alpha_0^0 + L_0^3}\right)$$
$$\alpha_2{}^0 = \log\left(e^{\alpha_1^0 + L_1^0} + e^{\alpha_1^1 + L_1^1} + e^{\alpha_1^2 + L_1^2} + e^{\alpha_1^3 + L_1^3}\right)$$
$$\alpha_2{}^1 = \log\left(e^{\alpha_1^0 + L_1^1} + e^{\alpha_1^1 + L_1^0} + e^{\alpha_1^2 + L_1^3} + e^{\alpha_1^3 + L_1^2}\right)$$
$$\alpha_2{}^2 = \log\left(e^{\alpha_1^0 + L_1^2} + e^{\alpha_1^1 + L_1^3} + e^{\alpha_1^2 + L_1^0} + e^{\alpha_1^3 + L_1^1}\right)$$
$$\alpha_2{}^3 = \log\left(e^{\alpha_1^0 + L_1^3} + e^{\alpha_1^1 + L_1^2} + e^{\alpha_1^2 + L_1^1} + e^{\alpha_1^3 + L_1^0}\right)$$
$$:$$
$$\alpha_{12}{}^0 = \log\left(e^{\alpha_{11}^0 + L_{11}^0} + e^{\alpha_{11}^1 + L_{11}^2} + e^{\alpha_{11}^2 + L_{11}^1} + e^{\alpha_{11}^3 + L_{11}^3}\right)$$

[0144]

[0145] 일반적으로, 코어 프로세스 유닛들에서 구현되는 코어 계산값은 다음과 같이 평가될 수 있다.

[0146] [수학식 15]

$$c = \log\left(e^a + e^b + e^c\right)$$

[0147]

[0148] 수학식 16에서, a, b 및 c는 수학식 15에 표시된 지수 계수들을 나타내고, 코어 프로세스 유닛들(1130a 내지 1130h 및 1135a 내지 1135d)에서 신호 표지들로서 도시된다. 계산 속도를 개선하기 위해, 코어 프로세스 유닛들(1130a 내지 1130h 및 1135a 내지 1135d)은 다음과 같이 반복적으로 계산값을 구현한다.

[0149] [수학식 16]

$$c = \log\left(e^{\log\left(e^a + e^b\right)} + e^c\right)$$

[0150]

[0151] 앞서 기술된 바처럼, 공통 아키텍처가 격자 블록(1100)에 의해 기술되는 바처럼 공유될 수 있더라도, 알파 격자 블록(625)은 베타 격자 블록(630)과 상이한 제어 시퀀스를 구현할 수 있다. 제어 시퀀스는 사용되는 격자 코드 및 격자 도표로부터 유도될 수 있다. 바람직한 실시예에서, 알파 격자 블록(625)의 일부로서 사용되는 입력 선택 블록(1110), 피드백 선택 블록(1120) 및 출력 다중화기(1140)에 대한 제어 시퀀스는 표 1에 도시된 바처럼 구현될 수 있다.

표 1

Mode	Clock	alpha_select[2:0]	alpha_alpha_select								fw_select[2:0]	alpha_fw_select								alpha_mux_select[2:]	alpha_mux
			alpha 0	alpha 1	alpha 2	alpha 3	4	5	6	7		alpha 0	alpha 1	alpha 2	alpha 3	4	5	6	7		alpha[7:0][..]
1 (C=2/3)	1	0									0	0		1		2		3			0:in[3:0][..]
	2	1	0 1 2 3	0 1 2 3	0 1 2 3	0 1 2 3					1	0 1 2 3	1 0 3 2	2 3 0 1	3 2 1 0						1:in[11:8][..]
	3	2	0 1	0 1	2 3	2 3					2	0 2	1 3	0 2	1 3						0:in[3:0][..]
	4	3	0 1	0 1	0 1	0 1	2 3	2 3	2 3	2 3	3	0 2	1 3	2 0	3 1	0 2	1 3	2 0	3 1		2:in[7:0][..]
	5	4	0 1 2 3	0 1 2 3	4 5 6 7	4 5 6 7					4	0 1 2 3	1 0 3 2	0 1 2 3	1 0 3 2						1:in[11:8][..]
	6	2	0 1	0 1	2 3	2 3					2	0 2	1 3	0 2	1 3						0:in[3:0][..]
	7	3	0 1	0 1	0 1	0 1	2 3	2 3	2 3	2 3	3	0 2	1 3	2 0	3 1	0 2	1 3	2 0	3 1		2:in[7:0][..]
	8	4	0 1 2 3	0 1 2 3	4 5 6 7	4 5 6 7					1	0 1 2 3	1 0 3 2	2 3 0 1	3 2 1 0						1:in[11:8][..]
	9	2	0 1	0 1	2 3	2 3					5	0 2	1 3	2 0	3 1						0:in[3:0][..]
	10	3	0 1	0 1	0 1	0 1	2 3	2 3	2 3	2 3	3	0 2	1 3	2 0	3 1	0 2	1 3	2 0	3 1		2:in[7:0][..]
	11	4	0 1 2 3	0 1 2 3	4 5 6 7	4 5 6 7					1	0 1 2 3	1 0 3 2	2 3 0 1	3 2 1 0						1:in[11:8][..]
	12	5	0 1 2 3								6	0 2 1 3									3:in[8][..]
0 (C=1/2)	1										0	0		1		2		3			0:in[3:0][..]
	2	2	0 1	0 1	2 3	2 3					2	0 2	1 3	0 2	1 3						0:in[3:0][..]
	3	2	0 1	0 1	2 3	2 3					2	0 2	1 3	0 2	1 3						0:in[3:0][..]
	4	2	0 1	0 1	2 3	2 3					2	0 2	1 3	0 2	1 3						0:in[3:0][..]
	5	2	0 1	0 1	2 3	2 3					5	0 2	1 3	2 0	3 1						0:in[3:0][..]
	6	2	0 1	0 1	2 3	2 3					5	0 2	1 3	2 0	3 1						0:in[3:0][..]
	7	2	0 1	0 1	2 3	2 3					5	0 2	1 3	2 0	3 1						0:in[3:0][..]
	8	5	0 1 2 3								6	0 2 1 3									3:in[8][..]

[0152]

[0153] 표 상의 빈 항목들 각각에 대해 하드웨어에 최대의 무부호 값을 공급할 수 있음을 주목하는 것이 중요하다. 또한, 알파 격자에서 사용되는 제어 시퀀스를 프로그래밍하기 위한 코드의 예시적인 실시예를 다음과 같이 나타낼 수 있다.

Mode = 1 (C=2/3); Clock 2;
Alpha 0
CPU[0] Input A = alpha 0 + fw_modlogi 0
CPU[0] Input B = alpha 1 + fw_modlogi 1
CPU[1] Input A = alpha 2 + fw_modlogi 2
CPU[1] Input B = alpha 3 + fw_modlogi 3
(CPU[8] Input A = CPU[0]
CPU[8] Input B = CPU[1])
Alpha 1
CPU[2] Input A = alpha 0 + fw_modlogi 1
CPU[2] Input B = alpha 1 + fw_modlogi 0
CPU[3] Input A = alpha 2 + fw_modlogi 3
CPU[3] Input B = alpha 3 + fw_modlogi 2
(CPU[9] Input A = CPU[2]
CPU[9] Input B = CPU[3])
Alpha 2
CPU[4] Input A = alpha 0 + fw_modlogi 2
CPU[4] Input B = alpha 1 + fw_modlogi 3
CPU[5] Input A = alpha 2 + fw_modlogi 0
CPU[5] Input B = alpha 3 + fw_modlogi 1
(CPU[10] Input A = CPU[4]
CPU[10] Input B = CPU[5])
Alpha 3
CPU[6] Input A = alpha 0 + fw_modlogi 3
CPU[6] Input B = alpha 1 + fw_modlogi 2
CPU[7] Input A = alpha 2 + fw_modlogi 1

[0154]

CPU[7] Input B = alpha 3 + fw_modlogi 0
(CPU[11] Input A = CPU[6]
CPU[11] Input B = CPU[7])

[0155] Alpha[7:0][..] = in[11:8][..] = CPU[11:8][..]

[0156] 알파 격자 블록(625)에 의해 생성되는 순방향 척도와는 대조적으로, 베타 격자 블록(630)에 의해 생성되는 역방향 척도는 격자 트리의 우측 상에서 시작하고, 좌측으로 향하는 처리에 따라 진행한다. 베타 격자 블록(630)에 대한 격자 트리의 역방향 척도 계산값은 다음과 같다.

[0157] [수학식 17]

[0158] $$\beta_n^k = \log\left(e^{\beta_{n+1}^i + L_n^j}\right)$$

[0159] 수학식 17에서, L_n^j 는 n번째 심볼의 가지 j의 척도이다. 앞서 기술된 바처럼, "00"의 값을 갖는 심볼 n에 대한 척도는 L_n^0 으로 표시된다. n번째 심볼의 상태 k의 척도는 β_n^k 으로 표시된다. 도 9를 다시 참조하면, 역방향 척도 계산값은 아래의 등식들을 사용하여 평가된다.

[0160] [수학식 18]

$$\beta_{12}^{k} = 0 \forall k = 0,1..3$$
$$\beta_{11}^{0} = \log\left(e^{\beta_{12}^{0}+L_{11}^{0}}\right)$$
$$\beta_{11}^{1} = \log\left(e^{\beta_{12}^{0}+L_{11}^{2}}\right)$$
$$\beta_{11}^{2} = \log\left(e^{\beta_{12}^{0}+L_{11}^{1}}\right)$$
$$\beta_{11}^{3} = \log\left(e^{\beta_{12}^{0}+L_{11}^{3}}\right)$$
$$\beta_{10}{}^{0} = \log\left(e^{\beta_{11}^{0}+L_{10}^{0}} + e^{\beta_{11}^{1}+L_{10}^{1}}\right)$$
$$\beta_{10}{}^{1} = \log\left(e^{\beta_{11}^{0}+L_{10}^{1}} + e^{\beta_{11}^{1}+L_{10}^{0}}\right)$$
$$\beta_{10}{}^{2} = \log\left(e^{\beta_{11}^{0}+L_{10}^{2}} + e^{\beta_{11}^{1}+L_{10}^{3}}\right)$$
$$\beta_{10}{}^{3} = \log\left(e^{\beta_{11}^{0}+L_{10}^{3}} + e^{\beta_{11}^{1}+L_{10}^{2}}\right)$$
$$\beta_{10}{}^{4} = \log\left(e^{\beta_{11}^{2}+L_{10}^{2}} + e^{\beta_{11}^{3}+L_{10}^{3}}\right)$$
$$\beta_{10}{}^{5} = \log\left(e^{\beta_{11}^{2}+L_{10}^{3}} + e^{\beta_{11}^{3}+L_{10}^{2}}\right)$$
$$\beta_{10}{}^{6} = \log\left(e^{\beta_{11}^{2}+L_{10}^{0}} + e^{\beta_{11}^{3}+L_{10}^{1}}\right)$$
$$\beta_{10}{}^{7} = \log\left(e^{\beta_{11}^{2}+L_{10}^{1}} + e^{\beta_{11}^{3}+L_{10}^{0}}\right)$$
$$\vdots$$
$$\beta_{0}{}^{0} = \log\left(e^{\alpha_{1}^{0}+L_{1}^{0}} + e^{\alpha_{1}^{1}+L_{1}^{1}} + e^{\alpha_{1}^{2}+L_{1}^{2}} + e^{\alpha_{1}^{3}+L_{1}^{3}}\right)$$

[0161]

[0162] 바람직한 실시예에서, 알파 격자 블록(625)의 일부로서 사용되는 입력 선택 블록(1110), 피드백 선택 블록(1120) 및 출력 다중화기(1140)에 대한 제어 시퀀스는 표 2에 도시된 바처럼 구현될 수 있다.

표 2

Mode	Clock	beta_select[2:0]	beta_beta_select								bw_select[2:0]	beta_bw_select								alpha_mux_select[2:]	beta_mux
			beta 0	beta 1	beta 2	beta 3	4	5	6	7		beta 0	beta 1	beta 2	beta 3	4	5	6	7		
1 (G=2/3)	1	0									0	0		2		1		3			0:in[3:0][..]
	2	1	0 1	0 1	0 1	0 1	2 3	2 3	2 3	2 3	1	0 1		1 0		2 3		3 2			1:in[7:0][..]
	3	2	0 1 2 3	0 1 2 3	4 5 6 7	4 5 6 7					2	0 1 2 3		2 3 0 1		0 1 2 3		2 3 0 1			2:in[11:8][..]
	4	3	0 1	0 1	2 3	2 3					3	0 1		2 3		2 3		0 1			0:in[3:0][..]
	5	1	0 1	0 1	0 1	0 1	2 3	2 3	2 3	2 3	1	0 1		1 0		2 3		3 2		2 3 3 2 0 1 1 0	1:in[7:0][..]
	6	2	0 1 2 3	0 1 2 3	4 5 6 7	4 5 6 7					2	0 1 2 3		2 3 0 1		0 1 2 3		2 3 0 1			2:in[11:8][..]
	7	3	0 1	0 1	2 3	2 3					4	0 1		2 3		0 1		2 3			0:in[3:0][..]
	8	1	0 1	0 1	0 1	0 1	2 3	2 3	2 3	2 3	5	0 1		1 0		2 3		3 2		0 1 1 0 2 3 3 2	1:in[7:0][..]
	9	2	0 1 2 3	0 1 2 3	4 5 6 7	4 5 6 7					2	0 1 2 3		2 3 0 1		0 1 2 3		2 3 0 1			2:in[11:8][..]
	10	3	0 1	0 1	2 3	2 3					4	0 1		2 3		0 1		2 3			0:in[3:0][..]
	11	4	0 1 2 3	0 1 2 3	0 1 2 3	0 1 2 3					6	0 1 2 3		1 0 3 2		2 3 0 1		3 2 1 0			2:in[11:8][..]
	12	5	0 1 2 3								7	0 1 2 3									3:in[8][..]
0 (G=1/2)	1										8	0		2		1		3			0:in[3:0][..]
	2	3	0 1	0 1	2 3	2 3					3	0 1		2 3		2 3		0 1			0:in[3:0][..]
	3	3	0 1	0 1	2 3	2 3					3	0 1		2 3		2 3		0 1			0:in[3:0][..]
	4	3	0 1	0 1	2 3	2 3					3	0 1		2 3		2 3		0 1			0:in[3:0][..]
	5	3	0 1	0 1	2 3	2 3					4	0 1		2 3		0 1		2 3			0:in[3:0][..]
	6	3	0 1	0 1	2 3	2 3					4	0 1		2 3		0 1		2 3			0:in[3:0][..]
	7	3	0 1	0 1	2 3	2 3					4	0 1		2 3		0 1		2 3			0:in[3:0][..]
	8	5	0 1 2 3								7	0 1 2 3									3:in[8][..]

[0163]

[0164] 이제 도 12a 및 12b를 살펴보면, 본 개시 내용의 태양들을 사용하는 성분 바이트 코드 디코더에서 사용되는 사후 블록(1200)에 관한 실시예의 블록도가 도시된다. 사후 블록(1200)에서, 순방향 척도, 역방향 척도 및 내부 척도를 나타내는 메시지 신호들은 순방향 선택 블록(1210), 역방향 선택 블록(1220) 및 내부 선택 블록(1230)에 접속된다. 일련의 가산기(1235a 내지 1235p 및 1240a 내지 1240p)는 순방향 척도 메시지, 역방향 척도 메시지 및 내부 메시지의 선택된 부분들(즉 비트들 또는 심볼들)을 결합한다. 새로운 척도 계산이 코어 프로세스 유닛들(1250a 내지 1250h 및 1255a 내지 1255d)에서 수행된다.

[0165] 코어 프로세스 유닛들(1250a 내지 1250h 및 1255a 내지 1255d)로부터의 계산된 척도들 각각이 후다중화기(post mux)(1260)에 접속된다. 후다중화기(1260)로부터의 출력들은 D 버퍼들(1270a 내지 1270d)을 통해 버퍼링된다. D 버퍼들(1270a 내지 1270d)은 계산된 척도값들을 사후 블록(1200)에 대한 출력들로서 제공한다.

[0166] 순방향 선택 블록(1210), 역방향 선택 블록(1220), 내부 선택 블록(1230) 및 후다중화기(1260)에 의해 수행되는 선택 및 다중화 동작들은 제어 블록(1280)에 의해 제어된다. 사후 블록(1200)은 또한 D 버퍼들(1270a 내지 1270d)을 통해 추가적으로 접속되는 동기 클록 및 스트로브 신호들, 및 제어 블록(1170)에 제공되는, 디코딩 코드 속도를 선택하기 위한 코드 속도 모드 입력 신호를 포함한다.

[0167] 사후 블록(1200)은 아래의 등식들을 사용하여 사후 척도를 계산한다.

[0168] [수학식 19]

$$\gamma_0^0 = \log\left(e^{\alpha_0^0+\beta_1^0+L_0^0}\right)$$
$$\gamma_0^1 = \log\left(e^{\alpha_0^0+\beta_1^1+L_0^1}\right)$$
$$\gamma_0^2 = \log\left(e^{\alpha_0^0+\beta_1^2+L_0^2}\right)$$
$$\gamma_0^3 = \log\left(e^{\alpha_0^0+\beta_1^3+L_0^3}\right)$$
$$:$$
$$\gamma_{11}^0 = \log\left(e^{\alpha_{11}^0+\beta_{12}^0+L_{41}^0}\right)$$
$$\gamma_{11}^1 = \log\left(e^{\alpha_{11}^2+\beta_{12}^0+L_{41}^1}\right)$$
$$\gamma_{11}^2 = \log\left(e^{\alpha_{11}^1+\beta_{12}^0+L_{41}^2}\right)$$
$$\gamma_{11}^3 = \log\left(e^{\alpha_{11}^3+\beta_{12}^0+L_{41}^3}\right)$$

[0169]

[0170] 바람직한 실시예에서, 사후 블록(1200)의 일부로서 사용되는 순방향 선택 블록(1210), 역방향 선택 블록(1220), 내부 선택 블록(1230) 및 후다중화기(1260)는 표 3에 도시된 바처럼 구현될 수 있다.

표 3

Mode	Clock	beta_select[2:0]	alpha_select post 0	alpha_select post 1	alpha_select post 2	alpha_select post 3	bw_select[2:0]	beta_select post 0	beta_select post 1	beta_select post 2	beta_select post 3	bw_select[2:0]	intrinsic_select post 0	intrinsic_select post 1	intrinsic_select post 2	intrinsic_select post 3	post_mux_select[2:]	post_mux post[3:0][]
1 (C=2/3)	1	0: 0	0	0	0	0	0: 0	0	1	2	3	0: 0	0	1	2	3	0:	in[3:0][.]
	2	1: 0 1 2 3	1 0 3 2	2 3 0 1	3 2 1 0		1: 0 1 2 3	0 1 2 3	0 1 2 3	0 1 2 3		1: 0 0 0 0	1 1 1 1	2 2 2 2	3 3 3 3	1:	in[7:4][.]
	3	2: 0 2	0 2	1 3	1 3		2: 0 2	1 3	0 2	1 3		2: 0 0	1 1	2 2	3 3	0:	in[3:0][.]
	4	3: 0 1 2 3	0 1 2 3	1 0 3 2	1 0 3 2		3: 0 2 4 6	1 3 5 7	0 2 4 6	1 3 5 7		1: 0 0 0 0	1 1 1 1	2 2 2 2	3 3 3 3	1:	in[7:4][.]
	5	4: 0 1 4 5	1 0 5 4	2 3 6 7	3 2 7 6		1: 0 1 2 3	0 1 2 3	0 1 2 3	0 1 2 3		1: 0 0 0 0	1 1 1 1	2 2 2 2	3 3 3 3	1:	in[7:4][.]
	6	2: 0 2	0 2	1 3	1 3		2: 0 2	1 3	0 2	1 3		2: 0 0	1 1	2 2	3 3	0:	in[3:0][.]
	7	3: 0 1 2 3	0 1 2 3	1 0 3 2	1 0 3 2		3: 0 2 4 6	1 3 5 7	0 2 4 6	1 3 5 7		1: 0 0 0 0	1 1 1 1	2 2 2 2	3 3 3 3	1:	in[7:4][.]
	8	5: 0 1 6 7	1 0 7 6	2 3 4 5	3 2 5 4		1: 0 1 2 3	0 1 2 3	0 1 2 3	0 1 2 3		1: 0 0 0 0	1 1 1 1	2 2 2 2	3 3 3 3	1:	in[7:4][.]
	9	6: 0 3	0 3	1 2	1 2		2: 0 2	1 3	0 2	1 3		2: 0 0	1 1	2 2	3 3	0:	in[3:0][.]
	10	3: 0 1 2 3	0 1 2 3	1 0 3 2	1 0 3 2		3: 0 2 4 6	1 3 5 7	0 2 4 6	1 3 5 7		1: 0 0 0 0	1 1 1 1	2 2 2 2	3 3 3 3	1:	in[7:4][.]
	11	5: 0 1 6 7	1 0 7 6	2 3 4 5	3 2 5 4		1: 0 1 2 3	0 1 2 3	0 1 2 3	0 1 2 3		1: 0 0 0 0	1 1 1 1	2 2 2 2	3 3 3 3	1:	in[7:4][.]
	12	7: 0	2	1	3		4: 0	0	0	0		3: 0	1	2	3	0:	in[3:0][.]
0 (C=1/2)	1	0: 0	0	0	0	0	0: 0	0	1	2	3	0: 0	0	1	2	3	0:	in[3:0][.]
	2	2: 0 2	0 2	1 3	1 3		2: 0 2	1 3	0 2	1 3		2: 0 0	1 1	2 2	3 3	0:	in[3:0][.]
	3	2: 0 2	0 2	1 3	1 3		2: 0 2	1 3	0 2	1 3		2: 0 0	1 1	2 2	3 3	0:	in[3:0][.]
	4	2: 0 2	0 2	1 3	1 3		2: 0 2	1 3	0 2	1 3		2: 0 0	1 1	2 2	3 3	0:	in[3:0][.]
	5	6: 0 3	0 3	1 2	1 2		2: 0 2	1 3	0 2	1 3		2: 0 0	1 1	2 2	3 3	0:	in[3:0][.]
	6	6: 0 3	0 3	1 2	1 2		2: 0 2	1 3	0 2	1 3		2: 0 0	1 1	2 2	3 3	0:	in[3:0][.]
	7	6: 0 3	0 3	1 2	1 2		2: 0 2	1 3	0 2	1 3		2: 0 0	1 1	2 2	3 3	0:	in[3:0][.]
	8	7: 0	2	1	3		4: 0	0	0	0		3: 0	1	2	3	0:	in[3:0][.]

[0171]

[0172] 이제 도 13을 살펴보면, 본 개시 내용의 태양들을 사용하는 바이트 코드 디코더(1300)의 실시예의 블록도가 도시된다. 바이트 코드 디코더(1300)는 앞서 기술된 바이트 코드 인코딩 프로세스에 대한 패리티 행렬 관계에 기초한 대안적인 디코딩 프로세스를 활용하기 위해 인입 정보의 그룹화 및 재정렬을 이용한다. 입력 코딩된 메시지가 입력 제어기 블록(1310)에 제공된다. 입력 제어기 블록의 출력은 신호 다중화기(1320)의 하나의 입력에 제공된다. 신호 다중화기(1320)의 출력은 사후 메모리(1330)에 접속된다. 사후 메모리(1330)의 출력은 가산기(1340)의 하나의 입력에 접속된다. 가산기의 출력은 지연 버퍼(1350) 및 외부 계산 프로세서(1360)에

접속된다. 외부 계산 프로세서(1360)의 출력은 외부 메모리(1370)에 접속된다. 외부 메모리(1370)의 출력은 반전되어 감산 기능을 구현하는 가산기(1340)의 제2 입력에 접속된다. 외부 계산 프로세서(1360) 및 지연 버퍼(1350)의 출력들은 가산기(1380)에 접속된다. 가산기(1380)의 출력은 입력 다중화기(1320)의 제2 입력에 제공된다. 사후 메모리(1330)의 출력은 또한 반복 디코딩 프로세스 후에 바이트 코드 디코더(1300)의 출력을 제공한다.

[0173] 입력 제어기(1310)는 인입 메시지를 수신하고, 인입 메시지의 일부를 비트들의 부분 집합으로 그룹화한다. 상기 부분 집합은 입력 메시지를 형성하고, 전형적으로 상기 부분 집합 내의 비트들 각각에 대한 LLR 값들과 같은 확률들로 이루어진다. 입력 제어기(1310)는 또한 다중화기(1320)에 대한 입력 메시지의 전달을 제어 또는 게이팅(gate)하고, 다중화기(1320)의 입력 스위칭을 제어할 수도 있다. 디코더(1300)의 반복 동작은 다중화기(1320)에 의해 제어된다. 다중화기(1320)는 최초의 디코딩을 위한 디코딩 프로세스를 통해 제1 반복시에 입력 메시지 스트림을 출력하거나, 또는 디코딩 프로세스를 통해 후속 반복들에 대해 가산기(1380)에 의해 공급되는 새로 계산된 피드백 메시지를 출력한다.

[0174] 다중화기(1320)로부터의 선택된 출력이 사후 메모리(1330)에 제공된다. 사후 메모리(1330)는 다중화기(1320)로부터 수신된 정보를 저장 및 배열한다. 바람직한 실시예에서, 사후 메모리(1330)는 디코딩 프로세스의 제1 반복 중에 비트 신뢰도들로서 수신된 인입 비트들의 부분 집합을 재정렬한다. 앞서 기술된 바처럼, 재정렬은 패리티 검사 행렬을 포함하는 디코딩 프로세스의 개선을 가능하게 한다. 사후 메모리(1330)는 재정렬된 정보 메시지를 비트들 또는 비트 신뢰도들의 부분 집합으로서 결합기(1340)에 제공한다.

[0175] 결합기(1340)는 재정렬된 정보 메시지를 외부 메모리(1370)에 의해 제공되는 외부 정보 메시지와 결합한다. 외부 메모리(1370)는 외부 계산 블록(1360)에서 계산된 외부 정보 메시지에 대한 외부 정보 값들을 저장한다. 바람직한 실시예에서, 결합기(1340)는 외부 정보 메시지 값들을 재정렬된 정보 메시지 값들로부터 빼고, 결합된 정보 메시지 값들을 외부 계산 블록(1360)에 출력한다.

[0176] 외부 계산 블록(1360)은 결합기(1340)로부터의 결합된 정보 메시지를 처리한다. 외부 계산 블록(1360)의 기능은 아래에서 더 기술될 것이다. 결합기(1340)로부터의 결합된 정보 메시지는 또한 버퍼(1350)에 제공된다. 버퍼(1350)는 결합된 정보 메시지를 지연시키는 데 사용되어 결합기(1380)에서 외부 계산 블록(1360)의 출력과의 올바른 재결합을 가능하게 한다. 결합기(1380)는 지연된 결합된 정보 메시지를 외부 계산 블록(1360)으로부터의 외부 정보 출력과 결합한다. 결합기(1380)는 사후 정보 메시지 값들을 포함하는 신호를 출력하고, 반복 디코딩 프로세스의 일부로서 사용하기 위해 상기 신호를 다중화기(1320)에 제공한다.

[0177] 성분 디코딩 프로세스 및 연접 디코딩 프로세스 둘 다는 하나의 패리티 검사 행렬을 사용하여 기술될 수 있고, 바이트 코드 디코더(1300)와 같은 단일 디코더를 사용하여 처리될 수 있음을 주목하는 것이 중요하다. 연접 디코딩 프로세스에서, 각 성분 디코딩 프로세스는 자기 자신의 패리티 검사 행렬에 의해 정의될 수 있다. 두 디코딩 프로세스, 그리고 따라서 두 패리티 행렬은 도 4에 도시된 바와 같은 인터리버 및 디인터리버를 통해 함께 접속된다. 패리티 검사 행렬들은 연접 디코딩 프로세스 내의 두 프로세스들에 대한 단일 패리티 검사 행렬을 형성하도록 결합될 수 있다. 인터리빙 단계는 결합된 패리티 검사 행렬들 내의 0이 아닌 원소들의 위치들을 정의한다.

[0178] 분리되어 접속된 성분 디코더들에 대한 패리티 검사 행렬들의 결합은 바이트 코드 디코더(1300)가 모든 가능한 바이트 코드 속도 구성을 디코딩할 수 있도록 한다. 인입 메시지의 부분에 대한 LLR 값들이 저장되고, 필요한 경우 사후 메모리에서 재정렬된다. 외부 정보는 사후 신뢰도들로부터 감산된다. 결과적인 내부 정보는 버퍼(1350) 및 외부 계산 블록(1360)에 공급된다. 계산된 외부 정보는 외부 메모리(1370)에 다시 저장된다. 또한, 계산된 외부 정보는 이전의 내부 정보에 추가되고, 사후 메모리(1330)에 다시 저장된다. 계산 및 프로세스는 각 반복 디코딩 단계에 대해 반복된다. 반복 디코딩은 정보 메시지에 대한 사후 신뢰도 값들이 신뢰도 문턱값을 초과하거나 할당된 디코딩 시간이 만료될 때까지 계속된다.

[0179] 이제 도 14를 살펴보면, 본 발명의 태양들을 사용하는 바이트 코드 디코더(1400)에 관한 다른 실시예의 블록도가 도시된다. 바이트 코드 디코더(1400)는 저밀도 패리티 검사(Low Density Parity Check; LDPC) 디코딩으로 알려진 특정한 패리티 검사 디코딩 프로세스를 구현하도록 구성될 수 있다.

[0180] 입력 메시지는 정보 메시지로부터의 비트들의 확률들을 나타내는 8개의 LLR 값의 시퀀스로서 치환기(permuter)(1410)의 입력에 제공된다. 치환기(1410)는 사전(즉, 제1 반복) 또는 사후 LLR 값들 중 하나를 순차적으로 수신한다. 치환기(1410)는 단순한 시프트 레지스터(shift register)이고, 가산기들(1420a 내지 1420h

및 1440a 내지 1440c)에 대한 인입 LLR 값들의 직접 접속을 가능하게 한다. 치환기(1410)는 또한 하나의 위치만큼 오른쪽으로 인입 값들이 순환 시프트되도록 할 수 있다. 치환기(1410)로부터의 8개의 출력 각각은 가산기들(1420a 내지 1420h)의 하나의 입력에 접속된다. 또한, 치환기(1410)로부터의 세 개의 출력(L3, L4 및 L5로 표시)은 각각 제2 집합의 가산기들(1440a 내지 1440c)의 하나의 출력에 제공된다.

[0181] 에지 선입 선출(edge FIFO) 버퍼(1470) 및 에지 FIFO 버퍼(1472)를 통해 전달되고 승산기(multiplier)(1430 및 1450)를 통해 부호 반전되는 계산된 외부 정보는 가산기들(1420a 내지 1420h 및 1440a 내지 1440c)의 제2 입력에 접속된다. 계산된 외부 정보는 인입 값들로부터 감산된다. 가산기들(1420a 내지 1420h)의 출력들에서의 결과적인 내부 정보는 검사 프로세스 업데이트 유닛(Check Process Update Unit)들(1465a 내지 1465h)에 제공되고 FIFO 버퍼들(1460a 내지 1460h)에도 제공된다. CPU들(1465a 내지 1465h)의 동작은 아래에서 더 상세히 기술될 것이다. CPU들(1465a 내지 1465h)로부터의 계산된 외부 정보는 가산기들(1420a 내지 1420h 및 1440a 내지 1440c)에 다시 제공되는 것 외에도 FIFO 버퍼들(1460a 내지 1460h)로부터의 버퍼링된 내부 정보와 결합된다. 결과적인 새로운 사후 정보는 비트들에 대한 LLR 값들로서 필요에 따라 결합기(1495)를 통해 이것의 원래의 위치로 다시 시프트되고, 새로운 사후 LLR 값들은 도 13에 도시된 사후 메모리(1330)와 같은 메모리에 다시 저장된다.

[0182] CPU들(1465d, 1465e 및 1465f)은 두 개의 외부 정보 값들을 계산할 수 있다. CPU들(1465d, 1465e 및 1465f)에 대한 제2 입력은 다중화기들(1445a 내지 1445c)을 통해 가산기들(1440a 내지 1440c)로부터 제2 입력을 수신한다. FIFO 버퍼들(1462a 내지 1462c)은 CPU들(1465d, 1465e 및 1465f)로부터의 새로 계산된 외부 값들의 하나의 집합을 결합하기 위해 이전에 계산된 외부 값들을 가산기들(1485a 내지 1485e)에 제공한다.

[0183] 바이트 코드 디코더(1400)에 의해 사용되는 디코딩 프로세스는 이진 LDPC 디코딩 프로세스로서 간주될 수 있다. 각 패리티 검사 등식은 저밀도 패리티 검사 코드로 해석될 수 있다. 바이트 코드 디코더(1400)는 8개의 병렬 처리가 가능하다. 그 결과, 바이트 코드 인코더(1400)는 8개의 등식을 갖는 하나의 등식 집합을 한 번에 처리할 수 있다.

[0184] CPU들(1465a 내지 1465h)은 비트 기반 LDPC 알고리즘을 사용하여 외부 정보를 계산한다. 바람직한 실시예에서, LDPC 알고리즘을 위한 등식은 다음과 같이 표현될 수 있다.

[0185] [수학식 20]

[0186]
$$R_{m,n} = 2\tanh^{-1}\left(\prod_{i\in C(m)\backslash n}\tanh\left(Q_{m,i}/2\right)\right) \overset{def}{\equiv} L\left(\sum_{i\in C(m)\backslash n}\oplus Q_{m,i}\right)$$

[0187] 위의 등식에서, Q_{m,i_j}는 CPU들(1465a 내지 1465h)에 대한 입력으로서 비트 i_j 및 패리티 검사 등식의 계산된 내부값 m을 나타낸다. R_{m,i_j}는 CPU들(1465a 내지 1465h)에 의해 계산되는 새로운 외부 값들을 나타낸다. $C(m)=\{n:H_{m,n}=1\}$은 검사 등식에 접속된 입력값들의 집합을 나타낸다. LDPC 디코딩 프로세스는 수학식 6으로부터의 단위 행렬 I 및 생성 행렬 S 둘 다에 기초하여 외부 값들을 계산한다. 단위 행렬의 디코딩 중에, 수정된 CPU 유닛의 제2 입력은 사용되지 않고, 다중화기들(1445a 내지 1445c)을 통해 최대의 양(positive)의 LLR 값을 공급함으로써 상기 입력이 단순히 불능화된다. S 행렬의 디코딩 중에, 내부 정보가 계산되어 제2 입력에 공급된다.

[0188] S 행렬의 제1열은 1의 값을 갖는 복수의 위치를 갖는다. 그 열의 대응하는 사후 값은 그 열의 각 등식의 모든 외부 정보의 합이다. 합산은 가산기(1485f)와 다중화기(1492)를 사용하여 달성된다. CPU(1465h)와 가산기(1480h)에 대한 새로운 사후 값을 계산하기 위해, 외부 값들이 CPU들(1465d, 1465e 및 1465f)에서 계산되고, 가산기(1480h)로부터의 사후 값이 가산기(1485)의 출력에서 결합된다.

[0189] 위에서 기술된 바이트 코드 디코더(1300)는 또한 비트들의 다른 그룹을 가능하게 할 수 있다. 예컨대, 이진 LDPC 디코딩 프로세스에서 사용되는 8 이상의 비트의 부분 집합을 형성하기 위한 그룹에 추가하여, LDPC 심볼들로 알려진 심볼들을 형성하도록 상기 부분 집합 내의 비트들을 더 그룹화함으로써 제2 그룹이 형성될 수 있다. 추가적인 그룹은 LDPC 디코딩 프로세스를 사용하는 경우 디코딩 시간을 감소시킴으로써 디코딩 프로세스의 추가적인 개선을 제공한다.

[0190] 비트들 대신에 LDPC 심볼들을 사용하여 외부 및 사후 확률값들을 계산하는 디코딩 알고리즘은 흔히 비 이진

LDPC 코드로 지칭되고, 유한체 공간의 속성들을 사용하여 정의될 수 있다. 바람직한 실시예에서, 원시 다항식이 다음과 같이 GF(8) 상에서 정의될 수 있다.

[0191] [수학식 21]

[0192] $p(x)=1+x+x^3$

[0193] 수신된 코딩된 메시지는 시스템 및 비 시스템 비트 정보를 포함하는 3개의 메시지 비트를 포함하는 심볼로서 표현될 수 있다.

[0194] 그 결과, 각각의 수신된 심볼은 8개의 확률로서 표현될 수 있다.

[0195] $P(x_{symbol}="000")$, $P(x_{symbol}="001")$, $P(x_{symbol}="010")$, $P(x_{symbol}="011")$, $P(x_{symbol}="100")$, $P(x_{symbol}="101")$, $P(x_{symbol}="110")$, $P(x_{symbol}="111")$

[0196] LDPC 심볼을 디코딩하기 위해, 패리티 검사 등식이 사용될 수 있다.

[0197] [수학식 22]

[0198] $cH^T=3c_2+2c_1+c_0=0 \quad \{c_i \in GF(8)\}$

[0199] 수학식 22에서, 각각의 심볼 c_i는 8개의 상이한 비트 성상 또는 확률을 나타낸다. 완전한 등식은 512(즉 8·8·8)개의 상이한 가능한 비트 성상을 포함할 것이다. 새로운 성상 확률들은 성상 확률들을 유입 정보로 콘볼루션(convolve)함으로써 계산된다. 예컨대, $c_0=0$에 대해 가능한 비트 성상은 다음과 같이 표현될 수 있다.

[0200] [수학식 23]

[0201]
$$r_{0,0}^0 = \sum_{x^{'}:x_0^{'}=0} \Pr ob\left[z_0 \mid x^{'}\right] \prod_{j \in N(0)\backslash 0} q_{0,j}^{x_j^{'}}$$
$$= q_{0,2}^0 q_{0,1}^0 + q_{0,2}^1 q_1^4 + q_{0,2}^2 q_{0,1}^3 + q_{0,2}^3 q_{0,1}^7 + q_{0,2}^5 q_{0,1}^6 + q_{0,2}^5 q_{0,1}^2 + q_{0,2}^6 q_{0,1}^5 + q_{0,2}^7 q_{0,1}^1$$

[0202] 수학식 23에서, q_j^x 값들은 가능한 성상 값들 각각에 대한 현재의 확률들을 나타내고, r_m^n은 가능한 성상 값들 각각에 대한 새로 계산된 확률들을 나타낸다. 각각의 r_{00}^i에 대해 $2^p=8$의 상이한 확률을 갖는다.

[0203] [수학식 24]

[0204]
$$r_{00,0}^1 = \sum_{x^{'}:x_0^{'}=1} \Pr ob\left[z_0 \mid x^{'}\right] \prod_{j \in N(0)\backslash 0} q_{0j}^{x_j^{'}}$$
$$= q_{0,2}^0 q_{0,1}^5 + q_{0,2}^1 q_{0,1}^1 + q_{0,2}^2 q_{0,1}^6 + q_{0,2}^3 q_{0,1}^2 + q_{0,2}^4 q_{0,1}^3 + q_{0,2}^5 q_{0,1}^7 + q_{0,2}^6 q_{0,1}^0 + q_{0,2}^7 q_{0,1}^4$$

[0205] [수학식 25]

[0206]
$$r_{0,0}^2 = q_{0,2}^0 q_{0,1}^1 + q_{0,2}^1 q_{0,1}^5 + q_{0,2}^2 q_{0,1}^2 + q_{0,2}^3 q_{0,1}^6 + q_{0,2}^4 q_{0,1}^7 + q_{0,2}^5 q_{0,1}^3 + q_{0,2}^6 q_{0,1}^4 + q_{0,2}^7 q_{0,1}^0$$
$$r_{0,0}^3 = q_{0,2}^0 q_{0,1}^4 + q_{0,2}^1 q_{0,1}^0 + q_{0,2}^2 q_{0,1}^7 + q_{0,2}^3 q_{0,1}^3 + q_{0,2}^4 q_{0,1}^2 + q_{0,2}^5 q_{0,1}^6 + q_{0,2}^6 q_{0,1}^1 + q_{0,2}^7 q_{0,1}^5$$
$$r_{0,0}^4 = q_{0,2}^0 q_{0,1}^2 + q_{0,2}^1 q_{0,1}^6 + q_{0,2}^2 q_{0,1}^1 + q_{0,2}^3 q_{0,1}^5 + q_{0,2}^4 q_{0,1}^4 + q_{0,2}^5 q_{0,1}^0 + q_{0,2}^6 q_{0,1}^7 + q_{0,2}^7 q_{0,1}^3$$
$$r_{0,0}^5 = q_{0,2}^0 q_{0,1}^7 + q_{0,2}^1 q_{0,1}^3 + q_{0,2}^2 q_{0,1}^4 + q_{0,2}^3 q_{0,1}^0 + q_{0,2}^4 q_{0,1}^1 + q_{0,2}^5 q_{0,1}^5 + q_{0,2}^6 q_{0,1}^2 + q_{0,2}^7 q_{0,1}^6$$
$$r_{0,0}^6 = q_{0,2}^0 q_{0,1}^3 + q_{0,2}^1 q_{0,1}^7 + q_{0,2}^2 q_{0,1}^0 + q_{0,2}^3 q_{0,1}^4 + q_{0,2}^4 q_{0,1}^5 + q_{0,2}^5 q_{0,1}^1 + q_{0,2}^6 q_{0,1}^6 + q_{0,2}^7 q_{0,1}^2$$
$$r_{0,0}^7 = q_{0,2}^0 q_{0,1}^6 + q_{0,2}^1 q_{0,1}^2 + q_{0,2}^2 q_{0,1}^5 + q_{0,2}^3 q_{0,1}^1 + q_{0,2}^4 q_{0,1}^0 + q_{0,2}^5 q_{0,1}^4 + q_{0,2}^6 q_{0,1}^3 + q_{0,2}^7 q_{0,1}^7$$

[0207] 도 15는 태너 그래프로서 수학식 25의 그래픽 표현을 도시한다. 태너 그래프들은 종종 비 이진 LDPC 디코딩에 수반되는 복소 수학적인 상호 작용들을 그래픽적으로 디스플레이하는 데 사용된다. 도 15는 값들의 임의의 재배열을 포함하는, 벡터 콘볼루션 함수를 통한 q_j^x 값들 각각에 대한 최초의 처리 및 벡터 콘볼루션 후의 결과적인 출력 r_m^n 값들을 도시한다.

[0208] 콘볼루션 함수, 특히 복수 벡터 콘볼루션 함수의 구현은 다수의 계산을 필요로 한다. 벡터들로서 표현되는 r^n_m 값들 및 q^x_j 값들에 변환 함수를 적용함으로써 단순화가 가능하다. 변환 함수를 적용하는 것은 변환된 r^n_m 값들 및 q^x_j 값들의 보다 간단한 벡터 승산을 가능하게 한다. 벡터 승산 후에, 역변환 함수가 결과적인 출력 벡터에 적용된다. 바람직한 실시예에서, 푸리에(Fourier) 변환 함수가 사용될 수 있고, 보다 간단한 고속 푸리에 변환(FFT)으로서 하드웨어 내에 더 구현될 수 있다. FFT는 변환 및 역변환이 승산 단계들 대신 가산 단계들을 사용하여 계산될 수 있도록 하는 효율적인 버터플라이(butterfly) 계산 기법을 종종 이용한다. 아다마르(Hadamard) 변환 및 이산 코사인(Discrete Cosine) 변환을 포함하지만 이에 한정되지 않는 다른 변환 함수들이 사용될 수 있음을 주목하는 것이 중요하다.

[0209] 이제 도 15를 살펴보면, 본 발명의 태양들을 사용하는 바이트 코드 디코더(1500)에 관한 추가적인 실시예의 블록도가 도시된다. 바이트 코드 디코더(1500)는 도 13의 디코더(1300) 및 도 14의 디코더(1400)에 대해 기술된 것과 유사한 방식으로 인입 정보 메시지 중 일부의 그룹화 및 재정렬과 연관된 속성과 바이트 코드 인코딩 프로세스와 연관된 패리티 검사 행렬을 활용한다.

[0210] 바이트 코드 인코더(1600)에서, 입력 메시지는 입력 제어기 블록(1610)에 제공된다. 입력 제어기 블록의 출력은 신호 다중화기(1620)의 하나의 입력에 제공된다. 신호 다중화기(1620)의 출력은 사후 메모리(1630)에 접속된다. 사후 메모리(1630)의 출력은 가산기(1635)의 하나의 입력에 접속된다. 가산기의 출력은 지연 버퍼(1640) 및 우도 계산 프로세서(1642)에 접속된다. 우도 계산 프로세서(1642)의 출력은 직렬 방식으로 치환기(1644), 고속 푸리에 변환 프로세서(FFT)(1646), 승산기(1648), 역 FFT(1650), 치환기(1652) 및 로그 우도 계산 프로세서(1654)에 접속된다. 로그 우도 계산 프로세서(1654)의 출력은 외부 메모리(1670) 및 가산기(1660)의 하나의 입력에 접속된다. 외부 메모리(1670)의 출력은 가산기(1635)의 제2 입력에 다시 접속된다. 버퍼(1640)의 출력은 가산기(1660)의 제2 입력에 접속된다. 가산기(1660)의 출력은 입력 다중화기(1620)의 제2 입력에 제공된다. 사후 메모리(1630)의 출력은 또한 반복 디코딩 프로세스 후의 바이트 코드 디코더(1600)의 출력의 역할을 한다. 입력 제어기 및 삽입기(1610), 다중화기(1620), 사후 메모리(1630), 외부 메모리(1670), 가산기(1635), 버퍼(1640) 및 가산기(1660)는 바이트 코드 디코더(1300)에 대해 기술된 동일한 블록들과 유사하게 접속되고 기능적으로 동일하며, 필요한 경우를 제외하고는 본 명세서에서 더 기술되지 않을 것임을 주목하는 것이 중요하다.

[0211] 우도 계산 프로세서(1642)는 가산기(1635)로부터 내부 정보 메시지를 수신하고, 내부 정보 내의 비트들에 대한 LLR 값들을 선형 또는 우도 확률들로 변환한다. 우도 계산 프로세서(1642)는 또한 비 이진 LDPC 디코딩 프로세스에 대한 추가적인 비트 대 심볼 그룹화에 기초하여 LDPC 심볼들에 대한 우도값들을 재계산할 수 있다. 치환기(1644)는 우도 계산 프로세서로부터 LDPC 심볼들에 대한 우도값들을 수신하고, 비 이진 LDPC 디코딩 프로세스에 필요한 LDPC 심볼들의 임의의 재정렬 또는 재배열을 제공한다. 치환기(1644)는 또한 어느 집합의 LDPC 심볼들이 FFT(1646)에 제공되는지를 제어한다.

[0212] FFT(1646), 승산기(1648) 및 역 FFT(1650)는 비 이진 LDPC 알고리즘과 관련하여 앞서 기술된 콘볼루션 기능을 수행한다. FFT(1646)는 LDPC 인입 시간 영역 기반 LDPC 심볼들을 등가의 주파수 영역 심볼들로 변환한다. 주파수 영역 LDPC 심볼들은 승산기(1648)에서 벡터 승산된다. 새로 계산된 주파수 영역 LDPC 심볼들은 역 FFT(1650)에서 시간 영역 LDPC 심볼들로 다시 변환된다. 새로 계산된 시간 영역 LDPC 심볼들은 치환기(1652)에 제공된다. 치환기(1652)는 치환기(1644)에 의해 수행된 임의의 재정렬 및 재배열을 반대로 한다.

[0213] 로그 우도 계산 프로세서(1654)는 LDPC 심볼들과 연관된 선형 우도 확률값들을 LLR 값들로 변환한다. 로그 우도 계산 프로세서(1654)는 또한 LDPC 심볼들 내의 비트들에 대한 LLR 값들을 재계산할 수 있다. 도 3의 격자 디코더(320) 및 리드 솔로몬 디코더(380)와 같은 디코더 내의 이전의 또는 후속하는 블록들 중 하나에 새로운 LLR 값들이 제공될 수 있도록 하기 위해, LDPC 심볼들이 아닌 비트들에 대한 LLR 값들로의 변환이 필요할 수 있다.

[0214] 이제 도 17을 살펴보면, 본 발명의 태양들을 사용하여 바이트 코드 인코딩된 신호를 디코딩하기 위한 프로세스(1700)에 관한 실시예에 대한 흐름도가 도시된다. 프로세스(1700)는 성분 바이트 코드 디코더(1300)와 관련하여 주로 기술될 것이다. 그러나, 프로세스(1700)는 성분 바이트 코드 디코더(600)와 같은 다른 바이트 코드 디코더들에 적용될 수 있고, 디코더(300)와 같은 복수의 디코더를 포함하는 더 큰 프로세스의 일부로서 더 사용될 수 있음을 주목해야 한다.

[0215] 프로세스(1700)는 시스템 및 비 시스템 바이트 코드 인코딩된 데이터의 패킷들 또는 바이트들을 포함하는 비트스트림으로서 바이트 코드 인코딩된 메시지를 수신하는 단계(1710)에서 시작한다. 수신된 메시지 비트스트림은 도 1의 GF(256) SCBC(114)에 대해 기술된 프로세스와 같은 선형 블록 인코딩 프로세스를 사용하여 신호의 인코딩 및 송신 중에 바이트 코드 인코딩된다. 메시지 비트스트림은 메시지 내의 각 비트에 대한 비트값들들 포함할 수 있거나, 또는 바람직하게는 비트값들에 대한 확률들 또는 신뢰도들, 또는 심볼들과 같은 비트들의 그룹들의 값들에 대한 확률들을 포함할 수 있다.

[0216] 다음으로, 단계(1720)에서, 인입 메시지가 부분 집합들로 배열된다. 인입 메시지 내의 비트들을 부분 집합들로 배열하는 것은 사용되는 디코딩 프로세스에 주로 기초하여 결정될 수 있다. 앞서 주목한 바처럼, 바이트 코드 디코딩은 전형적으로 비트 직렬 평가 프로세스를 수반한다. 디코딩 프로세스는 반복 평가에 의해 개선될 수 있다. 유한체들의 속성들에 더 기초하여 바이트 코드 인코딩 프로세스와 연관된 속성들을 활용함으로써 바이트 코드 인코딩 신호의 디코딩을 더 개선하는 것이 가능하다. 바람직한 실시예에서, 인입 메시지는 LDPC 패리티 검사 프로세스를 사용하여 패리티 검사 행렬의 일부로서 처리하기 위한 8 비트의 부분 집합들로 배열된다. 상기 배열은 패리티 검사 프로세스에 필요한 바에 따라 부분 집합 내의 피트들의 재정렬 또는 재배열을 더 포함할 수 있음을 주목하는 것이 중요하다.

[0217] 다음으로, 단계(1730)에서, 부분 집합들은 선택된 부분 집합의 속성들 및 바이트 코드 인코딩 프로세스의 속성들에 기초하여 선택된 디코딩 프로세스를 사용하여 디코딩된다. 바람직한 실시예에서, LDPC 패리티 검사 디코딩 프로세스는 원시 다항식 및 GF(2)에서의 패리티 행렬의 생성에 기초하여 선택된다. 디코딩 효율을 개선하기 위해, 생성 행렬 값들에 대한 역을 사용하여 GF(256) 공간에서 인입 메시지를 디코딩하는 대신 GF(2)에서의 LDPC 패리티 검사 디코딩 프로세스가 사용된다. 디코딩 단계(1730)는 또한 부분 집합 내의 메시지 비트들의 현재 상태에 대한 신뢰도들의 집합을 계산하는 단계를 포함할 수 있다. 확률들은 또한 메시지 내의 코딩된 중복 비트들에 대해 결정되고, 디코딩 단계(1730)에서 사용되는 다른 패리티 검사 프로세스 또는 이전의 격자 디코딩 프로세스와 같은 이전의 디코딩 프로세스에서 피드백 신호로서 사용되도록 이용 가능해진다.

[0218] 다음으로, 단계(1740)에서, 디코딩 프로세스가 메시지를 정확히 결정하고 디코딩하였는지 여부에 관한 결정이 내려진다. 상기 결정 단계는 계산된 확률들을 미리 결정된 문턱값들의 집합에 대해 비교하는 단계를 포함한다. 단계(1740)에서 디코딩 프로세스가 완료되지 않은 경우, 프로세스는 다른 디코딩 단계를 위해 단계(1730)로 복귀한다. 단계(1740)에서의 디코딩 프로세스가 완료되는 경우, 단계(1750)에서 최종값들이 디코딩 프로세스로부터 출력된다. 바람직한 실시예에서, 최종값들은 성분 바이트 코드 디코더(1300)의 출력으로서 제공된다. 최종값은 경판정, 또는 비트값들로서, 또는 연판정 확률값들로서 출력될 수 있다. 출력들은 다른 바이트 코드 디코더와 같은 추가 디코더, 또는 리드 솔로몬 디코더에 제공될 수 있다. 출력들은 또한 추가적인 격자 디코딩 개선을 위해 격자 디코더와 같은 이전의 디코딩 단에 다시 제공될 수 있다. 단계(1750)는 또한 출력 메시지 내의 비트들의 올바른 순서를 회복시키기 위해 부분 집합 내의 비트들의 재정렬 또는 재배열을 포함할 수 있다. 단계(1730 및 1740) 사이의 프로세스는 디코딩이 완료될 때까지, 또는 디코딩 단계의 완료를 위한 특정한 할당된 시간이 초과될 때까지 반복될 수 있다는 점에 주목하는 것이 중요하다.

[0219] 이제 도 18을 살펴보면, 본 발명의 태양들을 사용하여 바이트 코드 인코딩된 신호를 디코딩하기 위한 프로세스(1800)에 관한 다른 실시예의 흐름도가 도시된다. 프로세스(1800)는 도 6의 성분 바이트 코드 디코더(600)와 관련하여 주로 기술될 것이다. 그러나, 프로세스(1800)는 다른 바이트 코드 디코더들에 적용될 수 있고, 도 3의 디코더(300)와 같은 복수의 디코더를 포함하는 더 큰 프로세스의 일부로서 더 사용될 수 있음을 주목해야 한다.

[0220] 프로세스(1800)는 단계(1810)에서 인코딩된 메시지를 시스템 및 비 시스템 정보를 포함하는 비트스트림으로서 수신함으로써 시작한다. 메시지 비트스트림은 메시지 내의 각 비트에 대한 비트값들을 포함할 수 있거나, 또는 바람직하게는 비트값들에 대한 확률들 또는 신뢰도들, 또는 심볼들과 같은 비트들의 그룹들의 값들에 대한 확률들을 포함할 수 있다. 메시지는 또한 수신된 격자 변조된 심볼들 및 비트스트림 내의 비트들과 심볼들 사이의 관계에 관한 정보를 포함할 수 있다.

[0221] 다음으로, 단계(1820)에서, 인입 메시지 스트림이 부분 집합들로 배열된다. 앞서 주목한 바처럼, 바이트 코드 인코딩된 메시지들과 같은 블록 인코딩된 메시지들의 디코딩은 전형적으로 비트 직렬 평가 프로세스를 수반한다. 디코딩 프로세스는 종종 반복 평가를 통해 개선될 수 있다. 인코딩 프로세스와 연관된 속성들을 활용함으로써 바이트 코드 인코딩 신호의 디코딩을 더 개선하는 것이 가능하다. 바람직한 실시예에서, 인입 메시지는 16 비트의 부분 집합들로 배열되고, BCJR 알고리즘과 같은 반복 격자 트리 기반 알고리즘을 사용하여 디코

딩된다.

[0222] 다음으로, 단계(1830)에서, 식별된 심볼 관계에 기초하여 비트들의 부분 집합들을 추가적으로 그룹화하는 단계가 수행된다. 내부 송신 심볼 관계에 기초하여 비트들을 그룹화하는 단계는 디코딩 프로세스를 더 개선하는 데 사용될 수 있는 추가적인 내부 정보를 제공한다. 단계(1840)에서, 심볼들 및 기저 비트들이 격자 디코딩 프로세스를 최적화하기 위해 재정렬된다. 잠재적인 심볼 재정렬은 앞서 기술된 도 9 및 10에 도시된다.

[0223] 다음으로, 단계(1850)에서, 부분 집합들 내의 심볼들은 앞서 기술된 BCJR 알고리즘과 같은 심볼 기반 격자 디코딩 알고리즘을 사용하여, 그리고 선택된 부분 집합의 속성들 및 인코딩 프로세스의 속성들에 기초하여 선택된 패리티 등식들의 집합 중 하나 이상의 패리티 등식을 더 활용하여 디코딩된다. 바람직한 실시예에서, 패리티 검사 등식들의 집합은 GF(2)에서 패리티 행렬을 사용하여 생성된다. 앞서 기술된 바처럼, 디코딩 중의 시점에 보다 작은, 그리고 가능하게는 변화하는 비트들의 부분 집합 및 단일 패리티 등식을 사용하는 것이 또한 가능할 수 있다. 디코딩 단계(1850)는 또한 부분 집합 내의 메시지 비트들에 대한 신뢰도들의 집합을 계산하는 단계를 포함할 수 있다. 확률들은 또한 메시지 내의 코딩된 중복 비트들에 대해 결정되고, 이전의 디코딩 프로세스에서 피드백 신호로서 사용되도록 이용 가능해진다.

[0224] 다음으로, 단계(1860)에서, 디코딩 프로세스가 메시지를 정확히 결정하고 디코딩하였는지 여부에 관한 결정이 내려진다. 상기 결정 단계는 계산된 확률들을 미리 결정된 문턱값들의 집합에 대해 비교하는 단계를 포함한다. 단계(1860)에서 디코딩 프로세스가 완료되지 않은 경우, 프로세스는 다른 디코딩 단계를 위해 단계(1850)로 복귀한다. 단계(1860)에서의 디코딩 프로세스가 완료되는 경우, 단계(1870)에서 최종값들이 디코딩 프로세스로부터 출력된다. 바람직한 실시예에서, 최종값들은 성분 바이트 코드 디코더(600)의 출력으로서 제공된다. 최종값들은 경판정, 또는 비트값들로서, 또는 연판정 확률값들로서 출력될 수 있다. 출력들은 다른 바이트 코드 디코더와 같은 추가 디코더, 또는 리드 솔로몬 디코더에 제공될 수 있다. 출력들은 또한 디코딩 프로세스의 추가적인 개선을 가능하게 하도록 도 3의 격자 디코더(320)와 같은 이전의 디코딩 단에 다시 제공될 수 있다. 단계(1870)는 또한 출력 메시지 내의 비트들의 올바른 순서를 회복시키기 위해 부분 집합 내의 비트들의 재정렬 또는 재배열을 포함할 수 있다. 단계(1850 및 1860) 사이의 프로세스는 디코딩 프로세스가 완료될 때까지 반복될 수 있거나, 또는 특정한 할당된 디코딩 완료 시간에 기초하여 중단될 수 있음을 주목하는 것이 중요하다.

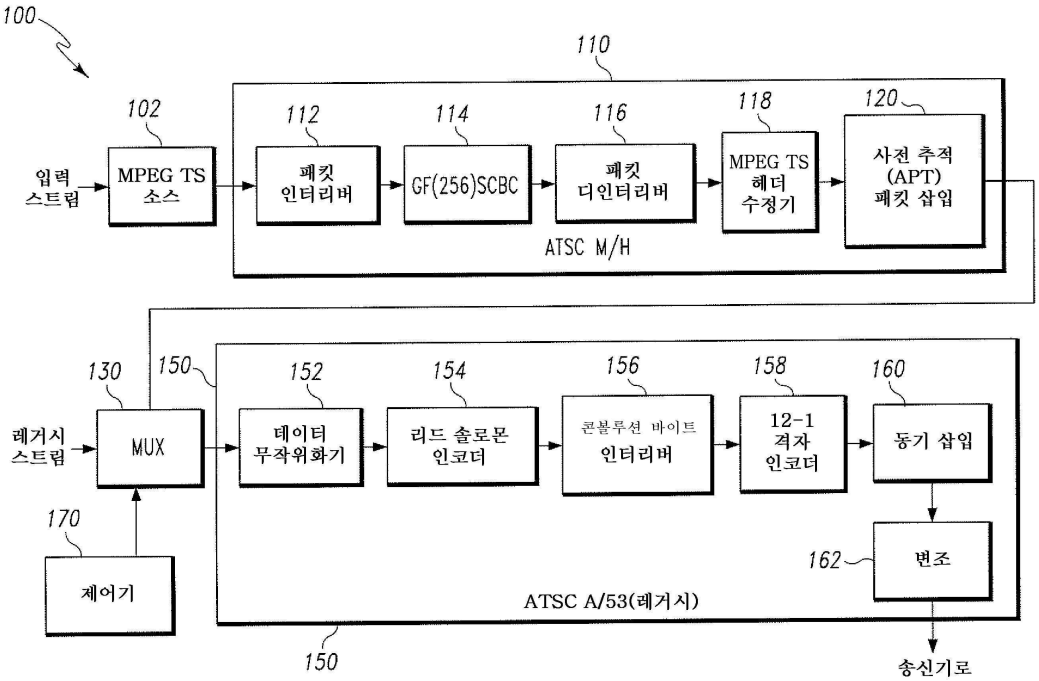
[0225] 프로세스(1800)가 심볼 기반 격자 디코딩 프로세스와 관련하여 주로 기술되었지만, 프로세스(1800)는 비트 기반 격자 디코딩 프로세스를 구현하는 데 사용될 수도 있다. 비트 기반 격자 디코딩 프로세스를 사용하여 프로세스(1800)를 구현하는 것은 예컨대 심볼 대 비트 관계를 식별하는 것과 관련된 단계(1830)를 생략할 수 있다.

[0226] 본 개시 내용은 1/2의 코드 속도 또는 2/3의 코드 속도 중 하나의 성분 코드 속도를 갖는 특정한 직렬 연접 블록 코드를 기술한다. 본 개시 내용은 디코더 효율 및 성능을 개선하기 위해 인접하지 않을 수 있는 인입 비트들의 그룹들을 사용하여 코드가 디코딩될 수 있는 성분 코드 구조의 특별한 속성을 이용한다. 본 개시 내용은 이진 LDPC 프로세스, 비 이진 LDPC 프로세스, 비트 격자 매핑 프로세스 및 변조 심볼 격자 매핑 프로세스를 포함하는 몇몇 가능한 비트 그룹화 방식을 기술한다.

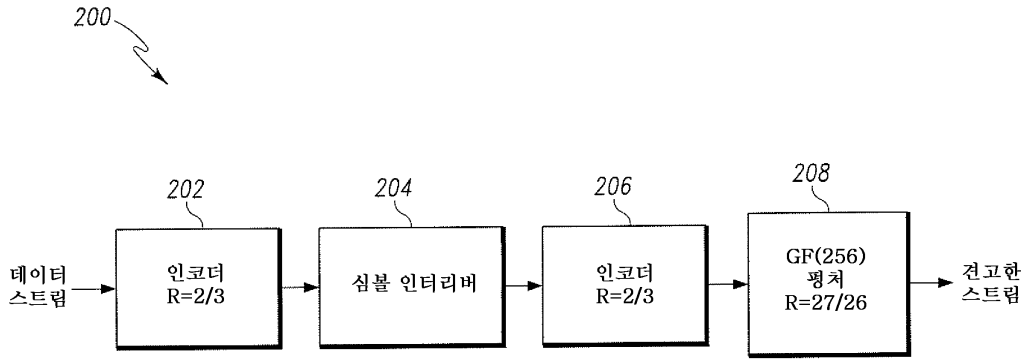
[0227] 본 실시예들에 다양한 수정이 가해질 수 있고 대안적인 형태를 취할 수 있지만, 특정한 실시예들은 예로서 도면들에 도시되었고 본 명세서에 상세히 기술되었다. 그러나, 본 개시 내용은 개시된 특정한 형태들로 한정되고자 하는 것이 아님을 이해해야 한다. 오히려, 본 개시 내용은 아래의 첨부된 청구항들에 의해 정의되는 바와 같은 본 개시 내용의 취지 및 범위 내에 속하는 모든 수정, 등가물 및 대안을 포괄하고자 하는 것이다.

도면

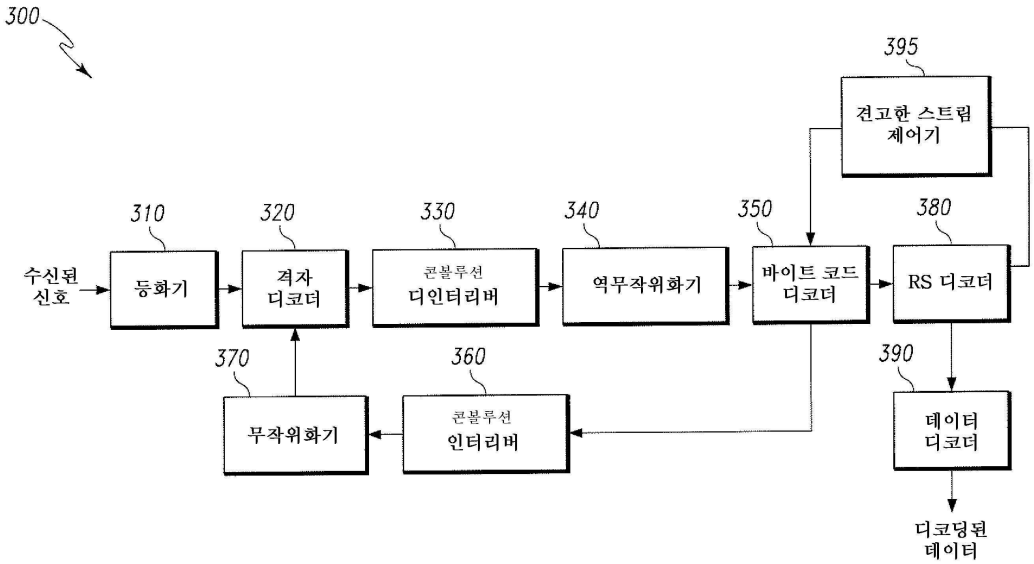
도면1


100
110
102
112
114
116
118
120
입력 스트림
MPEG TS 소스
패킷 인터리버
GF(256)SCBC
패킷 디인터리버
MPEG TS 헤더 수정기
사전 추적 (APT) 패킷 삽입
ATSC M/H
130
150
152
154
156
158
160
레거시 스트림
MUX
데이터 무작위화기
리드 솔로몬 인코더
콘볼루션 바이트 인터리버
12-1 격자 인코더
동기 삽입
170
162
제어기
변조
ATSC A/53(레거시)
150
송신기로

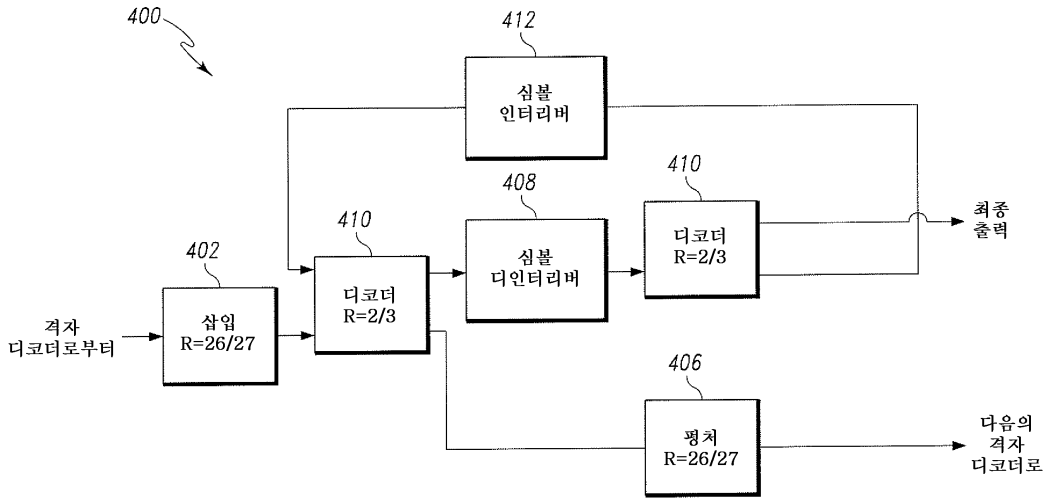
도면2


200
202
204
206
208
데이터 스트림
인코더 R=2/3
심볼 인터리버
인코더 R=2/3
GF(256) 펑처 R=27/26
견고한 스트림

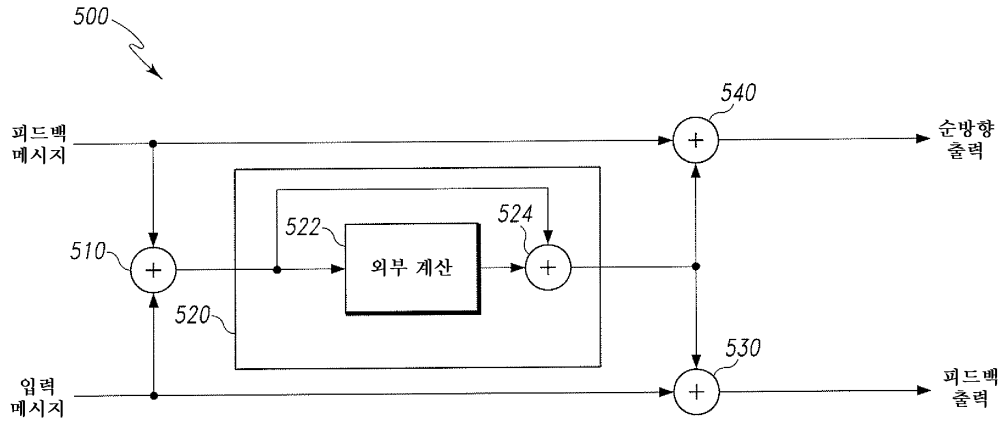
도면3


300
395
견고한 스트림 제어기
310
320
330
340
350
380
수신된 신호
등화기
격자 디코더
콘볼루션 디인터리버
역무작위화기
바이트 코드 디코더
RS 디코더
370
360
390
무작위화기
콘볼루션 인터리버
데이터 디코더
디코딩된 데이터

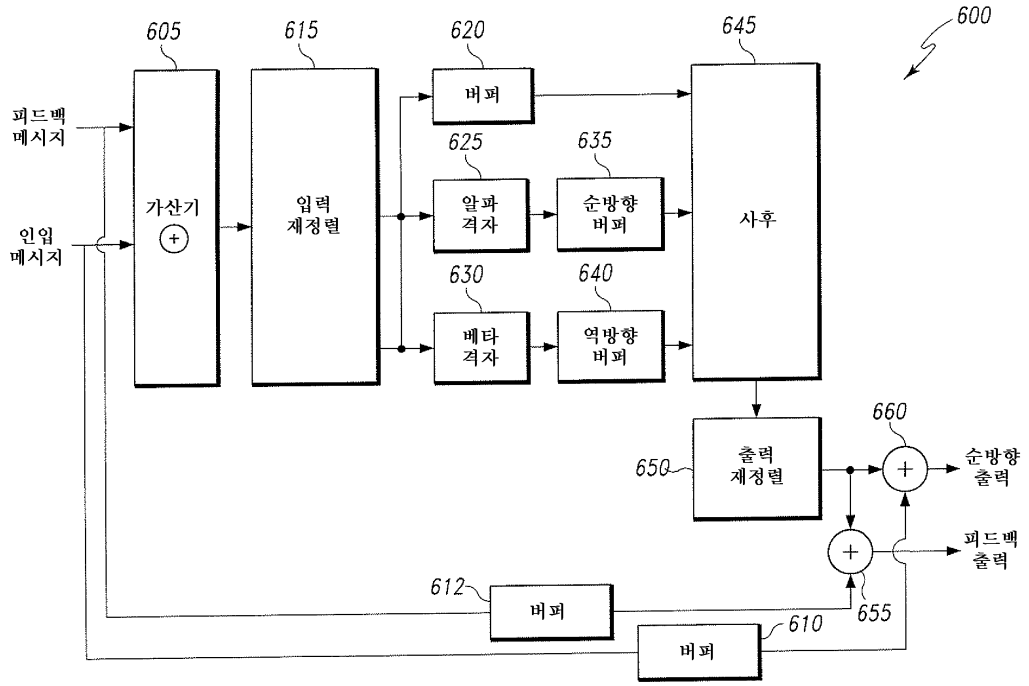
도면4


400
412
심볼 인터리버
410
408
410
402
최종 출력
디코더 R=2/3
심볼 디인터리버
디코더 R=2/3
격자 디코더로부터
삽입 R=26/27
406
펑처 R=26/27
다음의 격자 디코더로

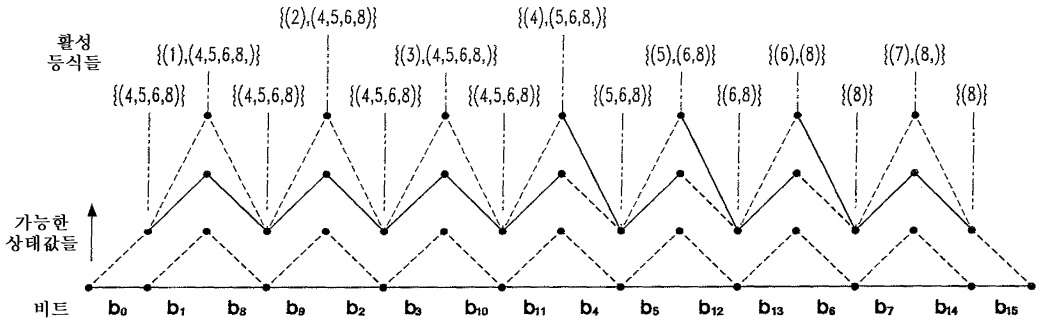
도면5


500
540
피드백 메시지
순방향 출력
522
524
510
외부 계산
520
530
입력 메시지
피드백 출력

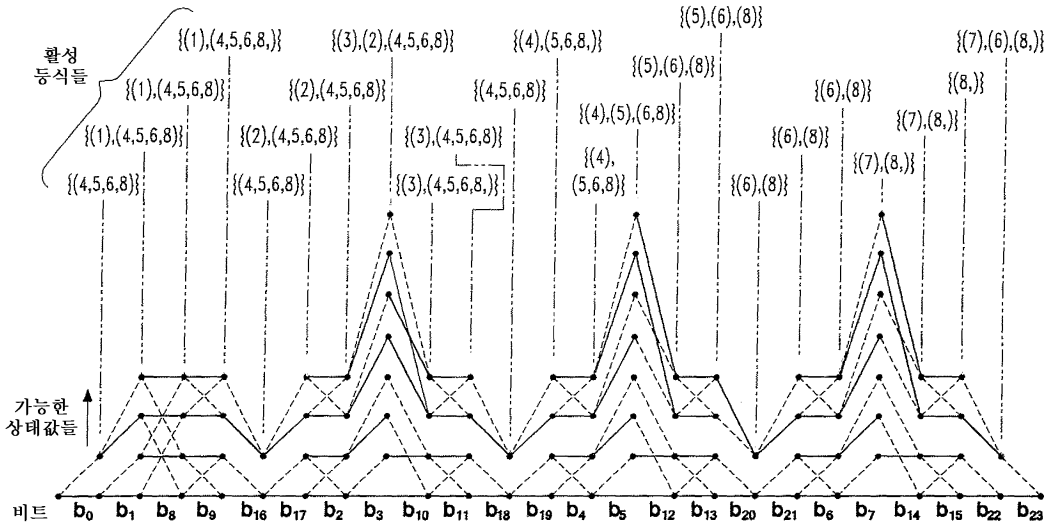
도면6


600
605
615
620
645
피드백 메시지
인입 메시지
가산기
입력 재정렬
버퍼
625
635
알파 격자
순방향 버퍼
사후
630
640
베타 격자
역방향 버퍼
660
650
출력 재정렬
순방향 출력
피드백 출력
612
655
610
버퍼
버퍼

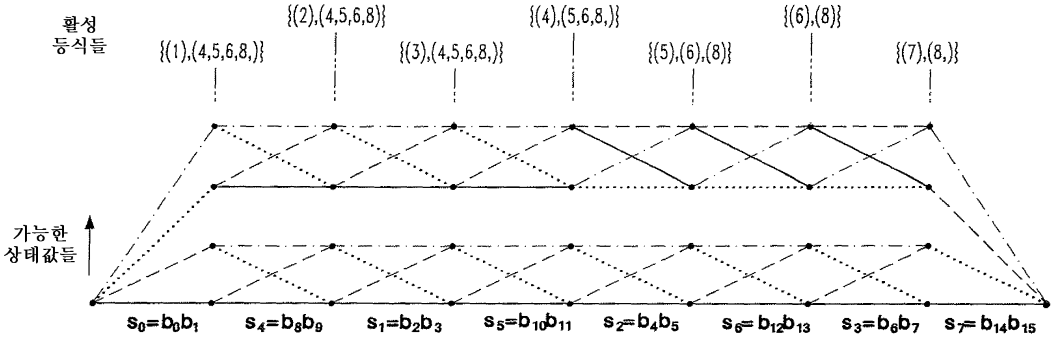
도면7


활성 등식들
{(1),(4,5,6,8,)}
{(2),(4,5,6,8)}
{(3),(4,5,6,8,)}
{(4),(5,6,8,)}
{(5),(6,8)}
{(6),(8)}
{(7),(8,)}
{(4,5,6,8)}
{(4,5,6,8)}
{(4,5,6,8)}
{(4,5,6,8)}
{(5,6,8)}
{(6,8)}
{(8)}
{(8)}
가능한 상태값들
비트
b0 b1 b8 b9 b2 b3 b10 b11 b4 b5 b12 b13 b6 b7 b14 b15

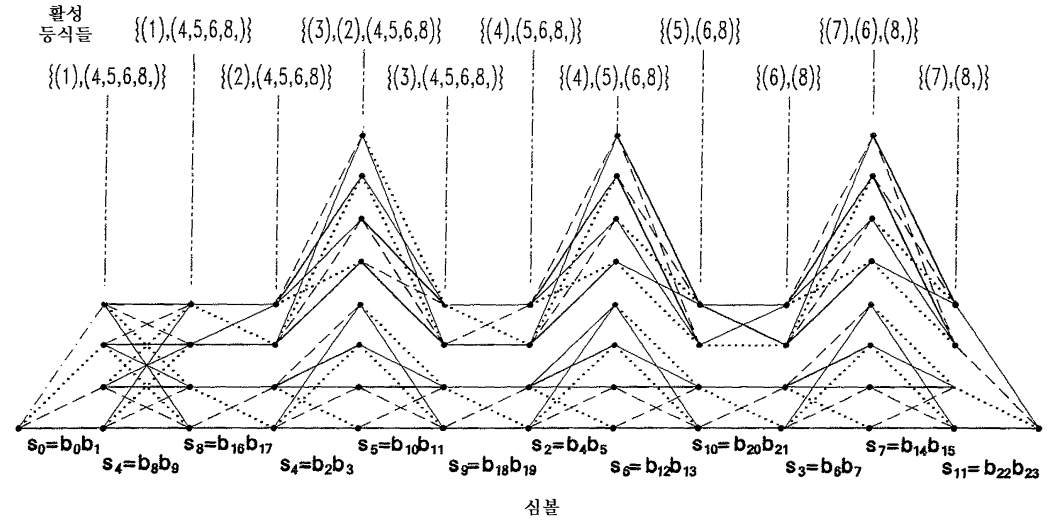
도면8


활성
등식들
{(1),(4,5,6,8,)}
{(3),(2),(4,5,6,8)}
{(4),(5,6,8,)}
{(5),(6),(8)}
{(7),(6),(8,)}
{(1),(4,5,6,8)}
{(2),(4,5,6,8)}
{(4,5,6,8)}
{(5),(6),(8)}
{(6),(8)}
{(8,)}
{(1),(4,5,6,8)}
{(2),(4,5,6,8)}
{(3),(4,5,6,8)}
{(4),(5),(6,8)}
{(6),(8)}
{(7),(8,)}
{(4,5,6,8)}
{(4,5,6,8)}
{(3),(4,5,6,8,)}
{(4),
(5,6,8)}
{(6),(8)}
{(7),(8,)}
가능한
상태값들
비트
b0 b1 b8 b9 b16 b17 b2 b3 b10 b11 b18 b19 b4 b5 b12 b13 b20 b21 b6 b7 b14 b15 b22 b23

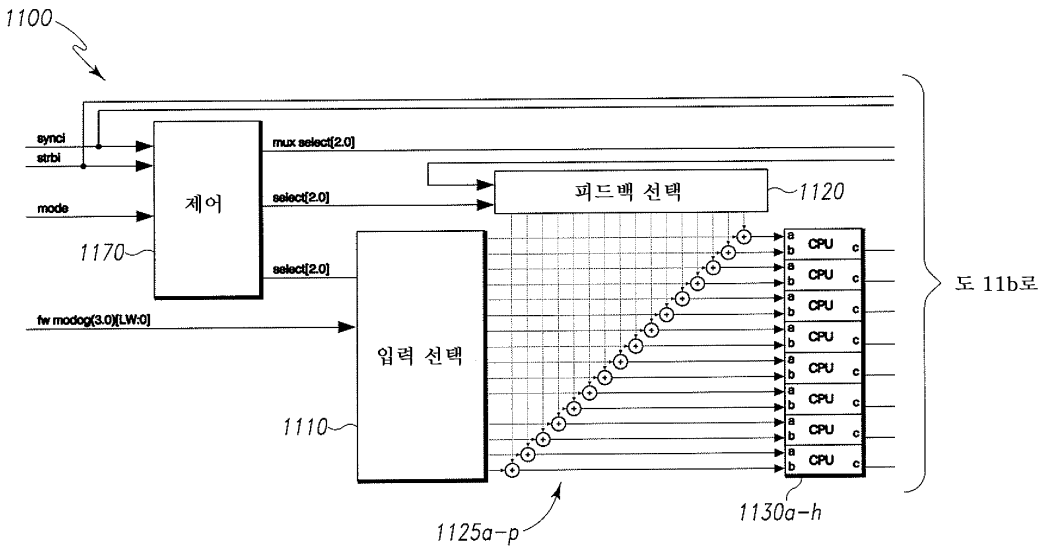
도면9


활성
등식들
{(2),(4,5,6,8)}
{(4),(5,6,8,)}
{(6),(8)}
{(1),(4,5,6,8,)}
{(3),(4,5,6,8,)}
{(5),(6),(8)}
{(7),(8,)}
가능한
상태값들
s0=b0b1 s4=b8b9 s1=b2b3 s5=b10b11 s2=b4b5 s6=b12b13 s3=b6b7 s7=b14b15

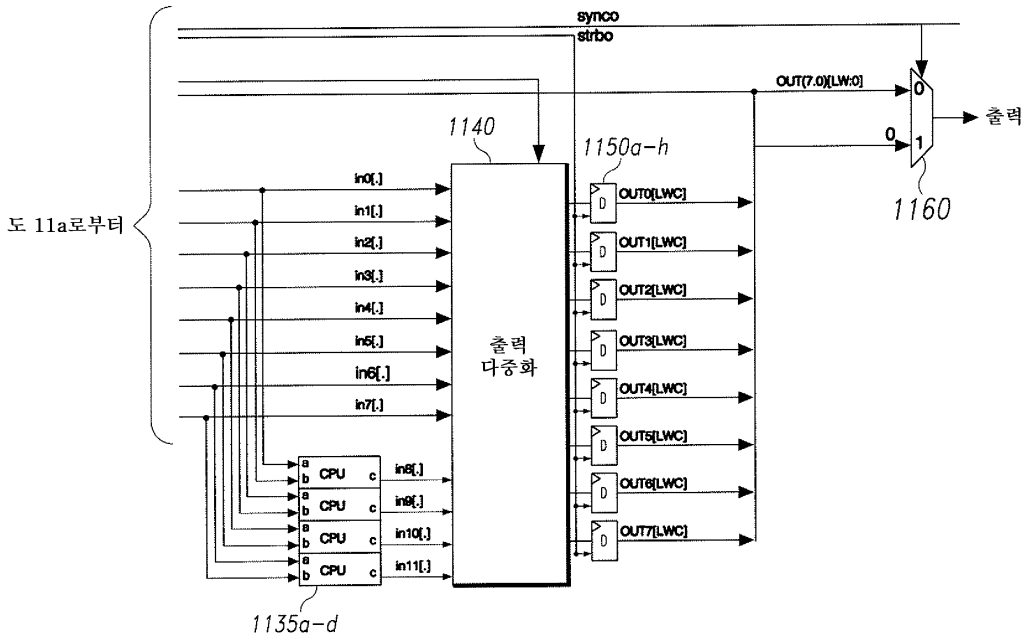
도면10


활성
등식들
{(1),(4,5,6,8,)}
{(3),(2),(4,5,6,8)}
{(4),(5,6,8,)}
{(5),(6,8)}
{(7),(6),(8,)}
{(1),(4,5,6,8,)}
{(2),(4,5,6,8)}
{(3),(4,5,6,8,)}
{(4),(5),(6,8)}
{(6),(8)}
{(7),(8,)}
s0=b0b1 s8=b16b17 s5=b10b11 s2=b4b5 s10=b20b21 s7=b14b15
s4=b8b9 s4=b2b3 s9=b18b19 s6=b12b13 s3=b6b7 s11=b22b23
심볼

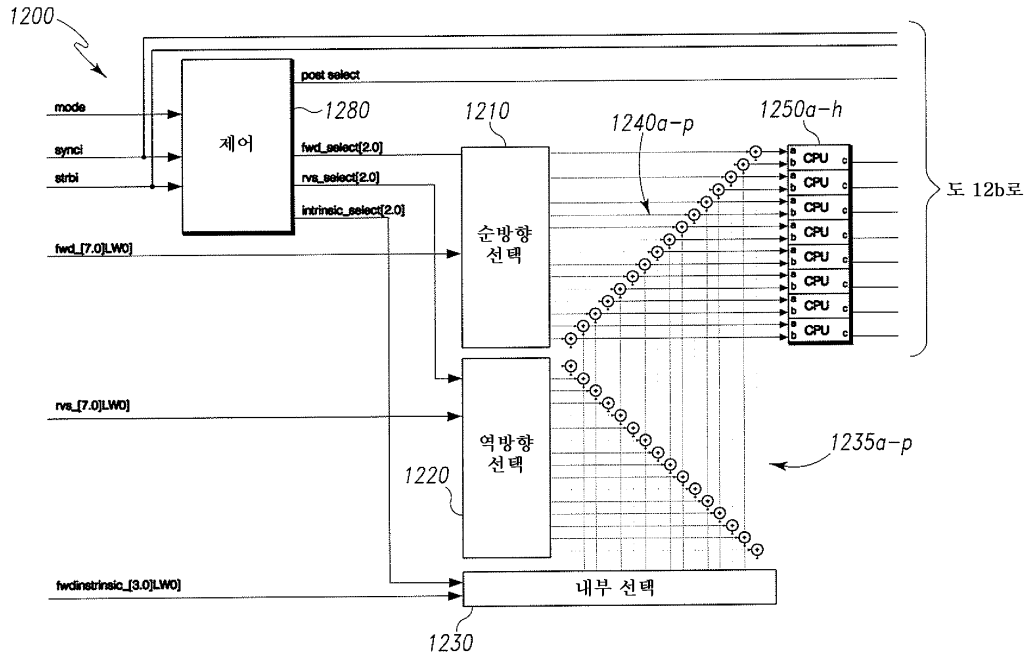
도면11a


1100
synci
strbi
mode
제어
1170
mux select[2.0]
select[2.0]
select[2.0]
fw modog(3.0)[LW:0]
피드백 선택
1120
입력 선택
1110
CPU
도 11b로
1125a-p
1130a-h

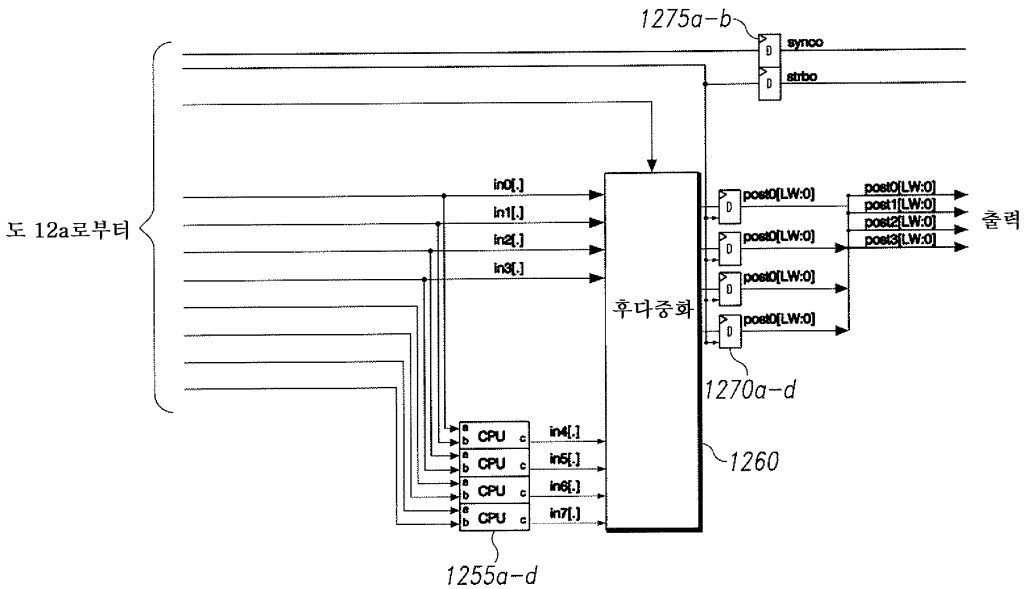
도면11b


synco
strbo
OUT(7.0)[LW:0]
출력
1140
1150a-h
1160
도 11a로부터
in0[.]
in1[.]
in2[.]
in3[.]
in4[.]
in5[.]
in6[.]
in7[.]
출력
다중화
OUT0[LWC]
OUT1[LWC]
OUT2[LWC]
OUT3[LWC]
OUT4[LWC]
OUT5[LWC]
OUT6[LWC]
OUT7[LWC]
CPU
in8[.]
in9[.]
in10[.]
in11[.]
1135a-d

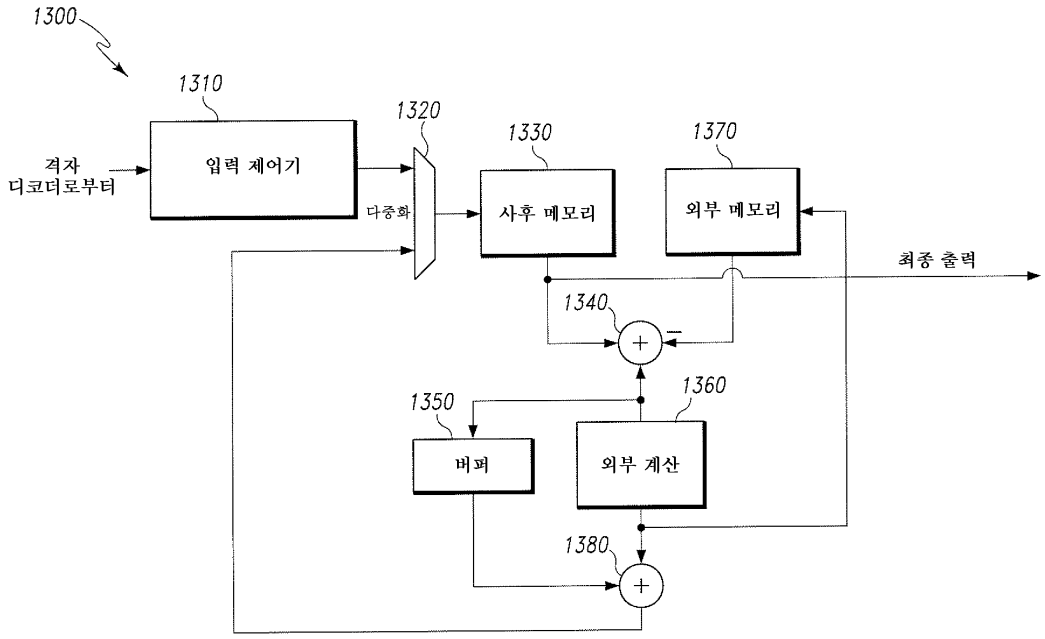
도면12a


1200
mode
synci
strbi
제어
post select
1280
fwd_select[2.0]
rvs_select[2.0]
intrinsic_select[2.0]
1210
1240a-p
1250a-h
CPU
도 12b로
fwd_[7.0]LW0]
순방향 선택
rvs_[7.0]LW0]
역방향 선택
1220
1235a-p
fwdinstrinsic_[3.0]LW0]
내부 선택
1230

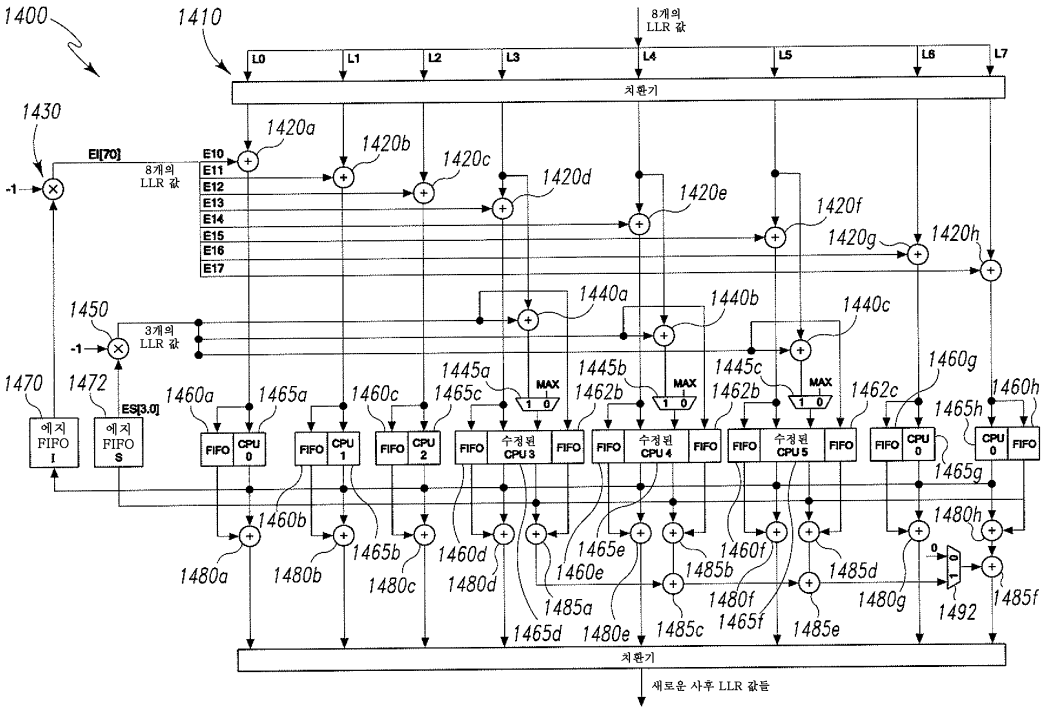
도면12b


1275a-b
synco
strbo
도 12a로부터
in0[.]
in1[.]
in2[.]
in3[.]
후다중화
post0[LW:0]
post1[LW:0]
post2[LW:0]
post3[LW:0]
출력
1270a-d
CPU
in4[.]
in5[.]
in6[.]
in7[.]
1260
1255a-d

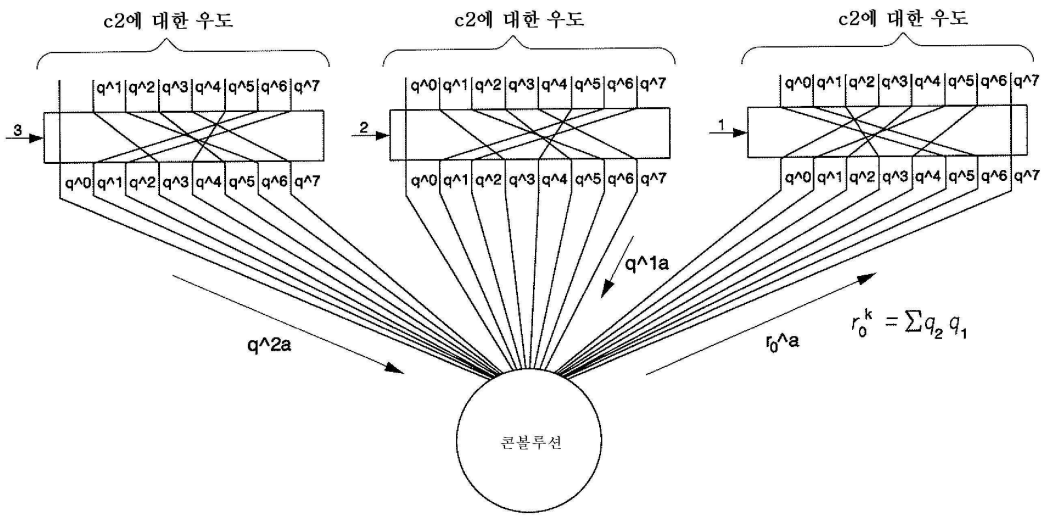
도면13


1300
1310
입력 제어기
격자
디코더로부터
1320
다중화
1330
사후 메모리
1370
외부 메모리
최종 출력
1340
1350
버퍼
1360
외부 계산
1380

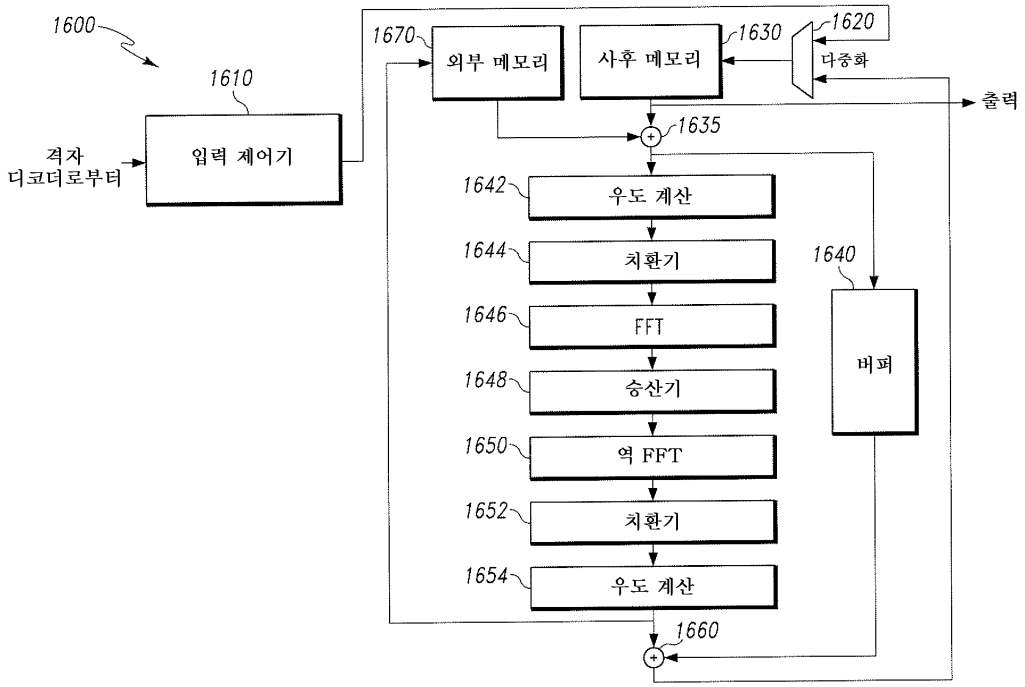
도면14


1400
1410
8개의 LLR 값
L0
L1
L2
L3
L4
L5
L6
L7
치환기
1430
EI[70]
8개의 LLR 값
E10
E11
E12
E13
E14
E15
E16
E17
1420a
1420b
1420c
1420d
1420e
1420f
1420g
1420h
1450
3개의 LLR 값
1440a
1440b
1440c
1470
1472
에지 FIFO I
에지 FIFO S
ES[3.0]
1460a
1465a
1460c
1445a
1465c
1445b
1462b
1445c
1462c
1460g
1460h
1465h
FIFO
CPU 0
CPU 1
CPU 2
수정된 CPU 3
수정된 CPU 4
수정된 CPU 5
MAX
1465g
1460b
1480h
1480a
1480b
1465b
1480c
1460d
1480d
1465e
1460e
1460f
1485b
1485d
1485a
1480f
1480g
1492
1485f
1465d
1480e
1485c
1465f
1485e
치환기
새로운 사후 LLR 값들

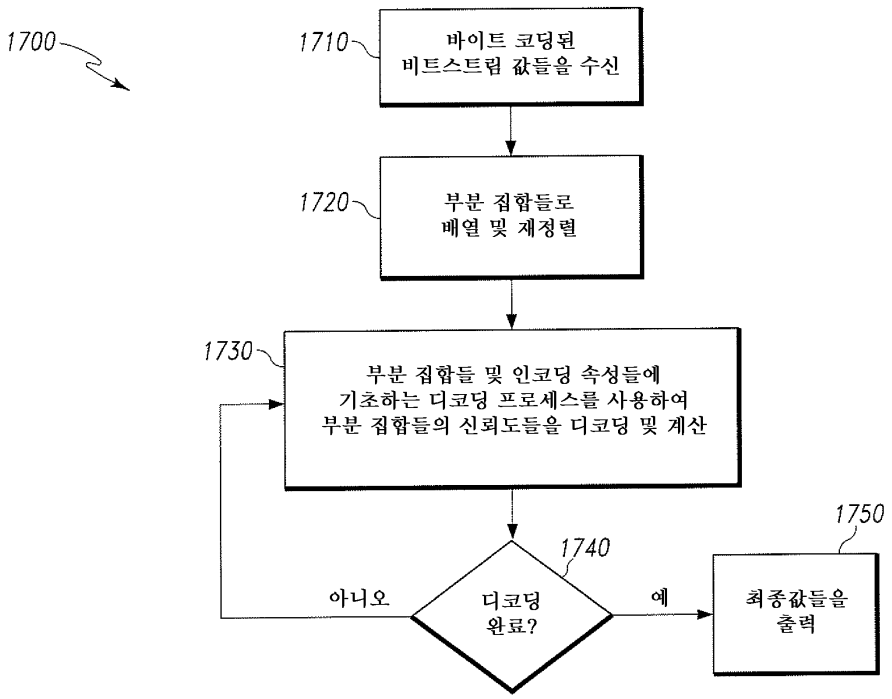
도면15


c2에 대한 우도
c2에 대한 우도
c2에 대한 우도
q^0 q^1 q^2 q^3 q^4 q^5 q^6 q^7
3
2
1
q^1a
q^2a
r_0^a
$r_0^k = \sum q_2 q_1$
콘볼루션

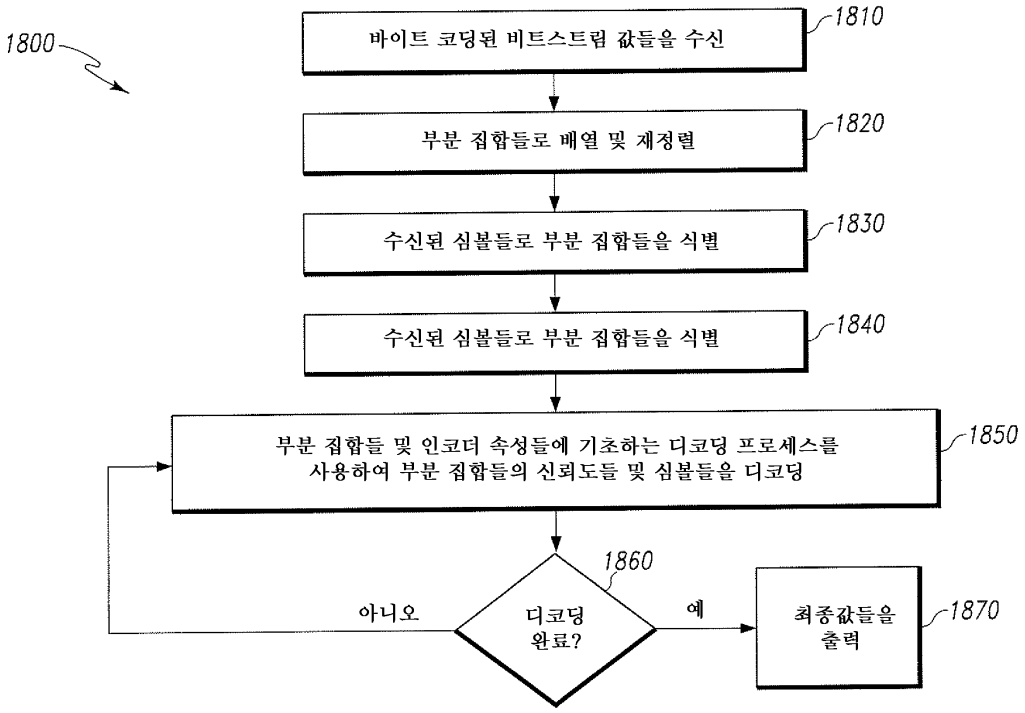
도면16


1600
1610
격자
디코더로부터
입력 제어기
1670
외부 메모리
사후 메모리
1630
1620
다중화
출력
1635
1642
우도 계산
1644
치환기
1646
FFT
1648
승산기
1650
역 FFT
1652
치환기
1654
우도 계산
1640
버퍼
1660

도면17


1700
1710
바이트 코딩된
비트스트림 값들을 수신
1720
부분 집합들로
배열 및 재정렬
1730
부분 집합들 및 인코딩 속성들에
기초하는 디코딩 프로세스를 사용하여
부분 집합들의 신뢰도들을 디코딩 및 계산
1740
디코딩
완료?
아니오
예
1750
최종값들을
출력

도면18


1800
바이트 코딩된 비트스트림 값들을 수신
1810
부분 집합들로 배열 및 재정렬
1820
수신된 심볼들로 부분 집합들을 식별
1830
수신된 심볼들로 부분 집합들을 식별
1840
부분 집합들 및 인코더 속성들에 기초하는 디코딩 프로세스를
사용하여 부분 집합들의 신뢰도들 및 심볼들을 디코딩
1850
1860
디코딩
완료?
아니오
예
최종값들을
출력
1870