(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2014년11월07일
(11) 등록번호 10-1459135
(24) 등록일자 2014년10월31일

(51) 국제특허분류(Int. Cl.)
H03M 13/11 (2006.01)
(21) 출원번호 10-2007-0076597
(22) 출원일자 2007년07월30일
심사청구일자 2012년07월25일
(65) 공개번호 10-2008-0011631
(43) 공개일자 2008년02월05일
(30) 우선권주장
11/496,121 2006년07월31일 미국(US)
(56) 선행기술조사문헌
KR1020060007362 A*
JP2005223440 A
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
에이저 시스템즈 엘엘시
미합중국 펜실베니아 18109 알렌타운 노스이스트 아메리칸 파크웨이 1110
(72) 발명자
하라트슈 에리크 에프
미국 펜실베니아주 18017 베들레헴 바바리 스트리트 5105
라트나야크 루완
미국 메사추세츠주 02138 캠브릿지 옥스포드 스트리트 33
(74) 대리인
제일특허법인, 김원준

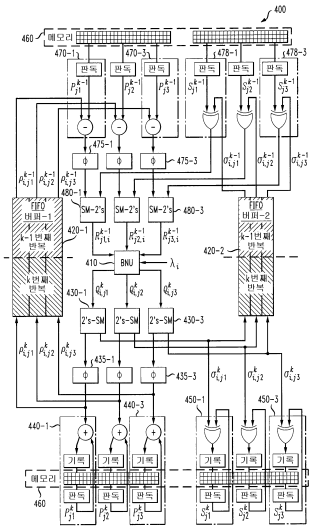
전체 청구항 수 : 총 12 항 심사관 : 성경아

(54) 발명의 명칭 **디코딩 방법 및 디코더**

(57) 요 약

상호연결된 비트 노드 및 체크 노드를 갖는 이원 그래프를 사용하여 기술될 수 있는 코드를 디코딩하는 방법 및 장치가 제공된다. 체크 노드(j)에서 비트 노드(i)로의 체크 노드-비트 노드 메시지의 크기가, 체크 노드(j)에 연결된 다수의 비트 노드에 대한 비트-체크 노드 메시지의 변환된 크기의 합에서 비트 노드(i) 및 체크 노드(j)에 대한 비트-체크 노드 메시지의 변환된 크기를 뺀 값에 기초하여 계산된다. 체크 노드(j)에서 비트 노드(i)로의 체크-비트 노드 메시지의 부호는 또한 체크 노드(j)에 연결된 다수의 비트 노드 사이의 비트-체크 노드 메시지의 부호의 곱(S_j)에, 비트 노드(i)와 체크 노드(j)에 대한 비트-체크 노드 메시지의 부호를 곱함으로써 계산될 수 있다. 또한, 상호연결된 비트 노드 및 체크 노드를 갖는 이원 그래프를 사용하여 기술될 수 있는 코드를 디코딩하는 디코더 아키텍처가 개시된다. 개시된 디코더는 소프트 출력 검출기와 연결될 수 있다.

대 표 도 - 도4



특허청구의 범위

청구항 1

상호연결된 비트 노드와 체크 노드를 갖는 이원 그래프(bipartite graph)를 이용하여 기술될 수 있는 코드를 디코딩하는 방법으로서,

체크 노드(j)에서 비트 노드(i)로의 체크-비트(check-bit) 노드 메시지의 크기를 계산하는 단계-상기 크기는 상기 체크 노드(j)에 연결된 복수의 비트 노드에 대한 비트-체크 노드 메시지의 변환된 크기의 합에서 비트 노드(i) 및 체크 노드(j)에 대한 비트-체크 노드 메시지의 변환된 크기를 뺀 값에 기초하고, 상기 변환된 크기의 각각은 비트-체크 노드 메시지의 대응하는 크기에 적용된 비선형 함수의 결과를 포함함-와,

상기 체크 노드(j)에 연결된 복수의 비트 노드 사이의 비트-체크 노드 메시지의 부호의 곱(product)(S_j)에, 비트 노드(i) 및 체크 노드(j)에 대한 비트-체크 노드 메시지의 부호를 곱함으로써, 체크 노드(j)에서 비트 노드(i)로의 상기 체크-비트 노드 메시지의 부호를 계산하는 단계를 포함하되,

하나의 비트 노드 업데이트 유닛에 연결된 복수의 병렬 체크 노드 업데이트 유닛이 상기 체크-비트 노드 메시지를 계산하는

디코딩 방법.

청구항 2

제 1 항에 있어서,

상기 이원 그래프는 패리티 체크 매트릭스에 대한 그래픽 표현인

디코딩 방법.

청구항 3

삭제

청구항 4

상호연결된 비트 노드와 체크 노드를 갖는 이원 그래프를 이용하여 기술될 수 있는 코드를 디코딩하는 디코더로서,

하나의 비트 노드 업데이트 유닛과,

상기 비트 노드 업데이트 유닛에 연결되어, 비트-체크 노드 메시지의 대응하는 크기에 적용된 비선형 함수의 결과를 각각 포함하는 하나 이상의 변환된 크기에 기초하여 복수의 체크-비트 노드 메시지를 계산하고, 상기 체크 노드(j)에 연결된 복수의 비트 노드에 대한 사전 정보 값의 부호의 곱(S_j)에, 비트 노드(i)에 대한 사전 정보 값의 부호를 곱함으로써 체크 노드(j)에서 비트 노드(i)로의 상기 체크-비트 노드 메시지의 부호를 계산하는, 복수의 병렬 체크 노드 업데이트 유닛을 포함하는

디코더.

청구항 5

제 4 항에 있어서,

상기 체크 노드 업데이트 유닛 중 하나는, 복수의 비트-체크 노드 메지시 중 하나 이상의 변환된 크기의 합에서 단 하나의 비트-체크 노드 메시지의 변환된 크기를 뺀 값과, 복수의 비트 노드에 대한 사전(a-priori) 정보 값에서 단 하나의 비트 노드의 사전 정보 값의 변환된 크기를 뺀 값에 기초하여 상기 체크-비트 노드 메시지의 크기를 계산하는

디코더.

청구항 6

제 4 항에 있어서,

상기 체크 노드 업데이트 유닛 중 하나는 메모리에 연결된 가산기 및 기록/판독 회로를 포함하여 복수의 비트-체크 노드 메시지의 변환된 크기의 합 또는 사전 정보 값을 계산하는

디코더.

청구항 7

상호연결된 비트 노드와 체크 노드를 갖는 이원 그래프를 이용하여 기술될 수 있는 코드를 디코딩하는 방법으로서,

체크 노드(j)에서 비트 노드(i)로의 체크-비트 노드 메시지의 크기를 계산하는 단계 - 상기 크기는 상기 체크 노드(j)에 연결된 복수의 비트 노드에 대한 사전 정보 값의 변환된 크기의 합에서 비트 노드(i)에 대한 사전 정보 값의 변환된 크기를 뺀 값에 기초하고, 상기 변환된 크기는 비트-체크 노드 메시지의 대응하는 크기에 적용된 비선형 함수의 결과를 포함하고, 하나의 비트 노드 업데이트 유닛에 연결된 복수의 병렬 체크 노드 업데이트 유닛이 상기 체크-비트 노드 메시지를 계산함 - 와,

상기 체크 노드(j)에 연결된 복수의 비트 노드에 대한 사전 정보 값의 부호의 곱(S_j)에, 비트 노드(i)에 대한 사전 정보 값의 부호를 곱함으로써 체크 노드(j)에서 비트 노드(i)로의 상기 체크-비트 노드 메시지의 부호를 계산하는 단계를 포함하는

디코딩 방법.

청구항 8

제 7 항에 있어서,

상기 사전 정보 값은 소프트 출력 검출기에 의해 생성되는

디코딩 방법.

청구항 9

제 7 항에 있어서,

상기 계산 단계는 소프트 출력 검출기와 연결된 LDPC 디코더에 의해 수행되는

디코딩 방법.

청구항 10

삭제

청구항 11

제 1 항에 있어서,

비트 노드(i)와 체크 노드(j) 사이의 비트-체크 노드 메시지($Q_{i,j}$)의 상기 변환된 크기($\rho_{i,j}$)는 $\rho_{i,j} = f(|Q_{i,j}|)$로서 계산되고, 여기서 "||"는 크기에 대한 표기인

디코딩 방법.

청구항 12

제 1 항에 있어서,

상기 비선형 함수는 -logtanh(x/2)와 $\log\frac{e^x+1}{e^x-1}$ 중 적어도 하나를 포함하는

디코딩 방법.

청구항 13

제 4 항에 있어서,

비트 노드(i)와 체크 노드(j) 사이의 비트-체크 노드 메시지($Q_{i,j}$)의 상기 변환된 크기($\rho_{i,j}$)는 $\rho_{i,j} = f(|Q_{i,j}|)$로서 계산되고, 여기서 "||"는 크기에 대한 표기인

디코더.

청구항 14

제 4 항에 있어서,

상기 비선형 함수는 -logtanh(x/2)와 $\log\frac{e^x+1}{e^x-1}$ 중 적어도 하나를 포함하는

디코더.

명 세 서

발명의 상세한 설명

기 술 분 야

[0001] 본 발명은 일반적으로 코딩 및 디코딩 기술에 관한 것으로서, 보다 구체적으로는 인코딩된 신호를 디코딩할 때 사용하는 디코더 알고리즘 및 아키텍처에 관한 것이다.

배 경 기 술

[0002] 코딩은 디지털 정보의 전송 시에 비트 및 패킷 에러 확률을 감소시키는 데 폭넓게 사용된다. 많은 애플리케이션에서는, 그 목적을 위해 컨볼루션 코드가 사용된다. 컨볼루션 코드는 인코딩될 입력 비트에 의해 결정된 상태 천이를 갖는 상태 머신에 의해 정의된다. 컨볼루션 코드 또는 컨볼루션 코드에 기초한 코드에 대한 가장 보편적인 두 가지 디코딩 알고리즘은 비터비 알고리즘(Viterbi algorithm)과 MAP(maximum a-posterior probability) 알고리즘이다.

[0003] 보다 큰 에러 보정 능력을 요구하는 애플리케이션에 대해서는 터보 코드 또는 LDPC 코드가 흔히 사용된다. 터보 코드와 같은 연결 코드(concatenated codes)의 디코딩은 다음 코드 트렐리스를 디코딩하기 위한 입력으로서 하나의 코드 트렐리스를 디코딩한 결과를 요구하며, 후속 코드 트렐리스에 대해서도 마찬가지다. 터보 코드에 대한 디코딩 알고리즘의 반복적 특성은 구현을 상당히 어렵게 한다.

[0004] LDPC 코드는 전도유망한 차세대 에러 보정 코드로서, 코딩 이득면에서 터보 코드를 능가할 수 있다. LDPC 코드의 병렬 또는 직렬 디코딩에 대하여 다수 개의 기술이 제안 및 제시되고 있다. A.J.Blanksby와 C.J.Howland의 "A 690-mW 1-Gb/s 1024-b, Pate-1/2 Low-Density Parity-Check Decoder", IEEE J. Solid-State Circuits, Vol.37, 404-412 (2002년 5월) 및 Blanksby 등의 미국 특허 번호 제6,539,367호에는, 예를 들어 저밀도 패리티 체크(LDPC) 코드의 블록 동시 디코딩이 개시되어 있다.

[0005] LDPC 디코더의 직렬 구현은 몇몇 하드웨어를 공유할 수 있고, 그에 따라 병렬 디코딩 방법에 비해 보다 적은 수의 계산 소자 및 보다 덜 복잡한 라우팅 메커니즘을 요구한다. 그러나, 직렬 방법은 전형적으로 이원 그래프(a bipartite graph)의 가장자리 상의 모든 메시지가 저장될 필요가 있기 때문에 추가 메모리를 요구하며, 처리량은 직렬 구현이 다수의 클록 사이클을 취하여 코드 블록을 디코딩하므로 제한된다. 일반적으로, 이원 그래프는 LDPC 코드에 의해 사용되는 패리티 체크 매트릭스의 그래픽 표현이다. 전형적인 메모리 요건은 코드 내의 에지의 수의 2배에 비례하며, 사이클의 수는 전형적으로 코드 내의 비트 노드의 수와 체크 노드의 수의 합에 비례한다.

[0006] 예를 들어, Zining Wu와 Gregory Burd는 LDPC 디코더가 채널 검출기와 연결된 통신 시스템용 직렬 아키텍처를 제안했다. 개시된 아키텍처는 통상적인 직렬 구현에 비해 메모리 요건을 감소시킨다. Z.Wu와 G.Burd의

"Equation Based LDPC Decoder for Intersymbol Interference Channels", IEEE Proc. of Acoustics, Speech, and Signal Processing 2005 (ICASSP'05), Vol. 5, 757-60(2005년 3월 18-23일)을 참조하라. D.E.Hocevar도 통상적인 직렬 구현에 비해 메모리 요건을 감소시키는 독립형 LDPC 디코더에 대한 직렬 아키텍처를 제안한다. D.E.Hocevar의 "LDPC Code Construction With Flexible hardware Implementation", IEEE Int'l Conf. on Comm. (ICC), 2708-2712 알라스카 앵커리지(2003년 5월)를 참조하라.

[0007] 독립형 및 연결형 LDPC 디코더 양측 모두에는 병렬 구현과 직렬 구현 모두에 비해 메모리 요건 또는 1회 반복시의 사이클 수(또는 양측 모두)에서의 추가 개선 및 비트 에러 레이트 성능에서의 추가 개선을 나타내는 직렬 LDPC 디코딩 아키텍처에 대한 필요성이 여전히 존재한다.

발명의 내용

과제 해결수단

[0008] 일반적으로, 상호연결된 비트 노드 및 체크 노드를 갖는 이원 그래프를 사용하여 기술될 수 있는 LDPC 코드와 같은 코드를 디코딩하는 방법 및 장치가 제공된다. 이원 그래프는 예를 들어 패리티 체크 매트릭스의 그래픽 표현일 수 있다. 본 발명의 일 측면에 따르면, 체크 노드(j)에서 비트 노드(i)로의 체크-비트 노드 메시지(check-to-bit node message)의 크기가 계산되고, 이 크기는 체크 노드(j)에 연결된 다수의 비트 노드에 대한 비트-체크 노드 메시지의 변환된 크기의 합에서 비트 노드(i) 및 체크 노드(j)에 대한 비트-체크 노드 메시지의 변환된 크기를 뺀 값에 기초한다. 체크 노드(j)에서 비트 노드(i)로의 체크-비트 노드 메시지의 부호는 또한 체크 노드(j)에 연결된 다수의 비트 노드 간의 비트-체크 노드 메시지의 부호의 곱(S_j)에, 비트 노드(i)와 체크 노드(j)에 대한 비트-체크 노드 메시지의 부호를 곱함으로써 계산될 수 있다.

[0009] 본 발명의 연관된 실시예에서, 체크 노드(j)에서 비트 노드(i)로의 체크-비트 노드 메시지의 크기가 계산되는데, 이 크기는 체크 노드(j)에 연결된 다수의 비트 노드에 대한 사전 정보 값(a-priori information value)의 변화된 크기의 합에서 비트 노드(i)에 대한 사전 정보 값의 변환된 크기를 뺀 값에 기초한다. 체크 노드(j)에서 비트 노드(i)로의 체크-비트 노드 메시지의 부호는 또한 체크 노드(j)에 연결된 다수의 비트 노드에 대한 사전 정보 값의 부호의 곱(S_j)에, 비트 노드(i)에 대한 사전 정보 값의 부호를 곱함으로써 계산될 수 있다. 이 계산은 소프트 출력 검출기와 연결된 LDPC 디코더에 의해 수행될 수 있다. 사전 정보 값은 소프트 출력 검출기에 의해 생성되거나 또는 소프트 출력 검출기에 의해 제공되는 소프트 출력에 기초하여 계산된다.

[0010] 본 발명의 또 다른 측면에 따르면, 상호연결된 비트 노드 및 체크 노드를 갖는 이원 그래프를 사용하여 기술될 수 있는 코드를 디코딩하는 디코더 아키텍처가 개시된다. 개시된 디코더는 독립형 모드로 구현될 수도 있고 또는 소프트 출력 검출기와 연관될 수 있다. 개시된 디코더는 비트 노드 업데이트 유닛 및 이 비트 노드 업데이트 유닛에 연결된 다수의 병렬 체크 노드 업데이트 유닛을 포함하여 다수의 체크-비트 노드 메시지를 계산한다.

[0011] 독립형 실시예에서, 체크 노드 업데이트 유닛은 다수의 비트-체크 노드 메시지의 변환 크기의 합에서 단 하나의 비트-체크 노드 메시지의 변환 크기를 뺀 값에 기초하여 체크-비트 노드 메시지의 크기를 계산한다. 연관된 실시예에서, 체크 노드 업데이트 유닛은 다수의 비트 노드에 대한 사전 정보 값의 변환된 크기의 합에서 단 하나의 비트 노드의 사전 정보 값의 변환 크기를 뺀 값에 기초하여 체크-비트 노드 메시지의 크기를 계산한다.

[0012] 메모리에 연결된 다수의 회로 유닛이 개시된다. 회로 유닛은 가산기, 감산기 또는 XOR 게이트와 같은 디지털 로직 소자, 및 판독 회로 또는 판독/기록 회로로 구성된다. 개시된 회로 유닛은 독립형 실시예에서는 다수의 비트-체크 노드 메시지의 변환된 크기 및 부호에 대해, 또한 연관된 실시예에서는 사전 정보 값의 변환된 크기 및 부호에 대해 다양한 연산을 수행한다.

[0013] 본 발명에 대한 보다 완전한 이해, 및 본 발명의 또 다른 특징과 장점은 후속하는 상세한 설명 및 도면을 참조하여 습득될 것이다.

효 과

[0014] 본 발명은 하드웨어 공유 및 직렬 합-곱 아키텍처(a hardware-sharing and serial sum-product architecture)를 사용하는 LDPC 디코딩을 위한 방법 및 장치를 제공함으로써, 계산 소자의 수가 감소하고 또한 라우팅 메커니즘이 간단해지는 효과를 제공한다.

발명의 실시를 위한 구체적인 내용

[0015] 본 발명은 하드웨어 공유 및 직렬 합-곱 아키텍처를 사용하는 LDPC 디코딩을 위한 방법 및 장치를 제공한다. 개시한 LDPC 디코더는 통상적인 구현에 비해 보다 작은 메모리와 보다 적은 클록 사이클을 갖는 합-곱 디코딩 알고리즘을 수행한다.

[0016] 저밀도 패리티 체크 코드(Low-Density parity Check Codes)

[0017] LDPC 코드 및 LDPC 디코딩에 관한 다음의 배경 논의는 본 명세서에서 참조로서 인용된 A.J.Blanksby와 C.J.Howland의 "A 690-mW 1-Gb/s 1024-b, Pate-1/2 Low-Density Parity-Check Decoder", IEEE J. Solid-State Circuits, Vol.37, 404-412 (2002년 5월)에서의 논의에 기초한다. 보다 상세한 논의에 대해서, 독자는 Blanksby와 Howlan의 전체 논문을 참조하라.

[0018] LDPC 코드의 매트릭스 표현(Matrix Representation of LDPC Codes)

[0019] LDPC 코드는 선형 블록 코드이다. 모든 코드워드 세트 $x \in C_x$는 패리티 체크 매트릭스 H의 영 공간(null space)에 걸쳐 있다.

[0020] $$Hx^T = 0, \qquad \forall x \in C_x \qquad (1)$$

[0021] LDPC 코드에 대한 패리티 체크 매트릭스는 희박 이진 매트릭스(a sparse binary matrix)이다.

[0022] 도 1은 LDPC 매트릭스 H의 일반적인 구조(100)를 예시한다. 도 1에 도시한 바와 같이, 패리티 체크 매트릭스 H의 각 행(row)은 패리티 체크에 상응하고, 세트 원소 h_{ji}는 데이터 비트 i가 패리티 체크 j에 참여함을 나타낸다. n 비트의 블록에는 m개의 잉여 패리티 비트가 존재한다. 코드 레이트는 다음과 같이 주어진다.

[0023] $$r=(n-m)/n \qquad (2)$$

[0024] 패리티 체크 매트릭스 H의 세트 행 및 열 원소는 희망 행 및 열 가중치 프로파일을 충족하도록 선택되는데, 여기서 행 및 열 가중치는 주어진 행 및 열 내의 세트 원소의 개수로서 각각 정의된다. 규칙적인 LDPC 코드에서, 모든 행은 균일한 가중치를 가지며, 모든 열도 마찬가지이다. 행 및 열이 균일한 가중치를 가지지 않는 경우, LDPC 코드는 불규칙한 것으로 간주된다.

[0025] LDPC 코드의 그래프 표현(Graph Representation of LDPC Codes)

[0026] LDPC 코드는 또한 이원 그래프를 이용하여 표현될 수 있는데, 이 경우 노드들의 한 세트는 패리티 체크 제약을 표현하고, 다른 세트는 데이터 비트를 표현한다. 도 2는 LDPC 코드의 예시적인 이원 그래프 표현(200)이다. 패리티 체크 매트릭스는, H 내의 엔트리 h_{ji}가 설정되는 경우, 즉 영이 아닌 경우, H 내의 열(i)에 대응하는 비트 노드(i)가 H 내의 행(j)에 대응하는 체크 노드(j)에 접속되는 그래프의 접속 매트릭스(incident maxrix)이다.

[0027] LDPC 코드를 디코딩하는 데 사용되는 알고리즘은 합-곱 알고리즘으로 알려져 있다. 이 알고리즘에 의한 양호한 디코딩 성능을 위해서, LDPC 코드의 그래프 표현에서의 사이클 길이는 가능한 한 긴 것이 중요하다. 도 2의 예시적인 표현에는, 길이 4의 예시적인 짧은 사이클이 예시되어 있다. 도 2에 예시된 길이-4 사이클과 같은 짧은 사이클은 합-곱 알고리즘의 성능을 저하시킨다.

[0028] 합-곱 알고리즘(The sum-Product Algorithm)

[0029] 합-곱 알고리즘은 LDPC 코드를 디코딩하는 반복 알고리즘이다. 합-곱 알고리즘은 또한 메시지 통과 알고리즘 또는 신뢰 전파(belief propagation)로서 알려져 있다. 합-곱 알고리즘에 대한 보다 상세한 논의를 위해서는, 예를 들어, 본 명세서에서 참조로서 각각 인용된 A.J.Blanksby와 C.J.Howland의 "A 690-mW 1-Gb/s 1024-b,

Pate-1/2 Low-Density Parity-Check Decoder", IEEE J. Solid-State Circuits, Vol.37, 404-412 (2002년 5월) 및 D.E.Hocevar의 "LDPC Code Construction With Flexible hardware Implementation", IEEE Int'l Conf. on Comm. (ICC), 2708-2712 알라스카 앵커리지(2003년 5월)를 참조하라.

[0030] 비트 노드(i)로부터 체크 노드(j)로의 메시지는 다음과 같이 주어진다.

[0031] $$Q_{i,j} = \sum_{l \in B_i, l \neq j} R_{l,i} + \lambda_i \qquad (3)$$

[0032] 본 명세서에서 사용된 표기법은 명세서의 말미에 있는 표에 정의되어 있다. 체크 노드(j)로부터 비트 노드(i)로의 메시지는 다음과 같이 주어진다.

$$R_{j,i} = s_{j,i} \cdot \phi \left(\sum_{l \in C_j, l \neq i} \phi \left(|Q_{l,j}| \right) \right) \qquad (4)$$

여기서,

$$s_{j,i} = \prod_{l \in C_j, l \neq i} sign(Q_{l,j}); \text{ 및}$$

[0033] $$\phi(x) = -\log \tanh(x/2) = \log \frac{e^x + 1}{e^x - 1}$$

[0034] 비트(i)에 대해 귀납적 LLR(log-likelihood ratio)이라고도 호칭되는 귀납적 정보 값은 다음과 같이 주어진다.

[0035] $$\Lambda_i = \sum_{l \in B_i} R_{l,i} + \lambda_i$$

[0036] LDPC 디코더(LDPC Decoder)

[0037] LDPC 코드를 디코딩하는 합-곱 알고리즘을 구현하는 경우 현저한 어려움은 메시지의 통과를 관리하는 것이다. 체크 및 비트 노드 모두의 기능은 비교적 간단하기 때문에, 그들의 제각기의 구현은 단지 적은 수의 게이트만을 포함한다. 중요한 쟁점은 기능 노드 간에 메시지를 전달하는데 필요한 대역폭의 구현이다.

[0038] 하드웨어 공유 디코더 아키텍처

[0039] 도 3은 예시적인 하드웨어-공유 LDPC 디코더 아키텍처(300)에 대한 블록도이다. 도 3에 도시되어 있는 바와 같이, 일반화된 LDPC 디코더 아키텍처(300)는 제각각 체크 또는 비트 노드 기능을 구현하는 다수의 기능 유닛(310,320)과, 메시지를 저장하고 그래프 연결성을 구현하는 메모리 패브릭(350)을 포함한다. 제어 로직(330)은 메모리 패브릭(350)의 구성을 제어한다. 하드웨어 공유 LDPC 디코더 아키텍처(300)의 구현에 대한 상세한 설명에 대해, 예를 들어, E. Y대 등의 "VLSI Architectures for Iterative Decoders in Magnetic Recording Channels," IEEE Trans, On Magnetics, Vol.37, No. 2, 748-755(March 2001)을 참조한다.

[0040] 이러한 하드웨어 공유 아키텍처는 디코더의 영역을 감소시킨다는 것을 알게되었다.

[0041] 변형된 LDPC 패리티 체크 방정식

[0042] 본 발명은 상기 LDPC 패리티 체크 방정식의 구성요소가 메모리 및 클록 싸이클 요건을 개선하도록 재구성될 수 있음을 인지한다. 방정식(4)에 도시된 체크 노드 계산은 다음과 같이 몇몇 구성요소로 분리될 수 있다.

[0043] $$\rho_{i,j} = \phi(|Q_{i,j}|) \qquad (5)$$

[0044] 여기서, $\rho_{i,j}$는 비트 노드(i)와 체크 노드(j) 간의 비트-체크 노드 메시지($Q_{i,j}$)의 변환된 크기이고, "||"는 크기에 대한 표기이다.

[0045] $$\sigma_{i,j} = \text{sign}(Q_{i,j}) \qquad (6)$$

[0046] 따라서, $\sigma_{i,j}$는 비트 노드(i)와 체크 노드(j) 간의 비트-체크 노드 메시지($Q_{i,j}$)의 부호이다.

[0047] $$P_j = \sum_{l \in C_j} \rho_{l,j} \quad (7)$$

[0048] P_j는 체크 노드(j)에 연결된 모든 비트 노드에 대한 비트-체크 노드 메시지의 변환된 크기의 합으로서 체크 노드(j)에 대해 계산된다.

[0049] $$S_j = \prod_{l \in C_j} \sigma_{l,j} \quad (8)$$

[0050] 따라서, 주어진 체크 노드(j)에 대한 S_j는 체크 노드(j)에 연결된 모든 비트 노드에 대한 비트-체크 노드 메시지의 부호 곱이다.

[0051] 체크 노드(j)에서 비트 노드(i)로의 메시지에 대한 크기 및 부호는 다음과 같이 주어진다.

[0052] $$\left|R_{j,i}\right| = \phi(P_j - \rho_{i,j}) \quad (9)$$

[0053] $$\text{sign}(R_{j,i}) = S_j \cdot \sigma_{i,j} \quad (10)$$

[0054] 따라서, 방정식(4)에 따라 체크-비트 노드 메시지를 계산하는 종래의 방식은 계산으로부터 현재의 비트 노드(i)를 배제하지만($l \in C_j, l \neq i$), 본 발명은 체크 노드(j)에 연결된 모든 비트 노드(l)에 대한 비트-체크 노드 메시지의 변환된 크기($\rho_{l,j}$)의 합으로서 중간 값(P_j)을 계산하고, 그런 다음 P_j로부터 $\rho_{i,j}$를 감산하여 체크 노드(j)에서 비트 노드(i)로의 메시지($R_{j,i}$)의 크기를 계산한다.

[0055] 예시적인 실시예에서, 비트 노드 계산은 2의 보수로 수행되며, 체크 노드 계산은 부호-크기(SM)로 수행된다.

[0056] 직렬 합-곱 아키텍처를 갖는 LDPC 디코더

[0057] 도 4는 본 발명의 특징을 포함하는 LDPC 디코더(400)의 블록도이다. 도 4에 도시되어 있는 LDPC 디코더(400)에 대한 예시적인 실시예는 d_c가 3이고, B_i가 {j1, j2, j3}인 예를 나타낸다. 소자(430,435,440,450,470,475,478,480)는 병렬 체크 노드 업데이트 유닛을 포함하는 체크 노드 업데이트 블록을 형성한다. 각각의 시간 싸이클에서, 하나의 비트 노드에 관련된 계산이 수행된다. 따라서, 비트 노드(1 내지 n)에 대한 완전한 계산을 수행하기 위해서는 n 싸이클이 필요하다. 비트 노드(1)는 첫 번째 시간 싸이클 동안 계산되는 것으로 가정한다. $(n \cdot (k-1)+i)$번째 시간 싸이클에서, 비트 노드 업데이트 유닛(410)은 k번째 반복에서 i번째 비트 노드에 대응하는 d_c 메시지($Q_{i,j}^k$,$j \in B_i$)를 생성한다. 이들 d_c 메시지는 스테이지(430)에서 2의 보수로부터 부호-크기 숫자 표현으로 변환된다. 크기 부분은 함수(ϕ)를 수행하는 병렬 변환 유닛(435-1 내지 435-3)으로 전송되고, 이들 유닛(435)으로부터의 출력은 방정식(5)에 의해 정의되는 $\rho_{i,j}^k, j \in B_i$이다.

[0058] 이들 $\rho_{i,j}^k, j \in B_i$는 가산기 및 판독/기록 회로로 구성된 d_c 변환된 크기 업데이트 유닛(440)(체크 노드(j)에 연결된 모든 비트 노드에 대한 합(P_j^k)을 계산하고, 그런 다음 P_j^k는 이후의 다음 반복에서 사용됨)에 공급된다. 병렬 변환된 크기 업데이트 유닛(440)은 메모리(460) 내의 m개의 메모리 소자로부터 임의의 d_c 소자를 액세스할 수 있으며, 각 메모리 소자는 q비트를 저장하고, q는 예시적인 실시예에서 P_j^k를 나타내는데 사용되는 비트의 개수이다. 이들 병렬 변환된 크기 업데이트 유닛(440)은 반복의 끝(즉, $(n \cdot k)$번째 싸이클)에서 방정식(7)에 정의된 $P_j^k, j \in 1 \ldots m$ 가 m개의 메모리 소자에 저장되도록 관련 메모리 소자를 업데이트한다. 다시 말해, 이들 메모리 소자는 관련 ρ 값의 누적 합(running sum)을 유지한다.

[0059] k번째 반복에 대한 메모리 소자(j) 내의 중간 값이 i=1...n에 대해 $P_j^k(i)$에 의해 주어지는 경우, $(n \cdot (k-1)+i)$번째 시간의 경우에서의 누적 합은 다음과 같이 주어진다.

$$P_j^k(i) = \begin{cases} P_j^k(i-1) + \rho_{i,j}^k & j \in B_i \text{ 인 경우} \\ P_j^k(i-1) & \text{그렇지 않은 경우} \end{cases} \quad (11)$$

[0060]

[0061] (n · k)번째 시간의 경우에서와 같이 반복의 끝에서는

[0062] $$P_j^k = P_j^k(n) \quad (12)$$

[0063] 이다.

[0064] 방정식(6)에 의해 정의되는 $\sigma_{i,j}^k,\ j \in B_i$ 인 비트-체크 노드 메시지($Q_{i,j}^k,\ j \in B_i$)의 부호는 아래와 같이 처리된다. 상술한 절차와 유사하게, $\sigma_{i,j}^k,\ j \in B_i$ 은 XOR 게이트 및 판독/기록 회로로 구성되는 또 다른 세트의 d_c 부호 업데이트 유닛(450)에 공급된다. 이들 병렬 부호 업데이트 유닛은, 반복의 끝에서 방정식(8)에서 정의된 $S_j^k,\ j \in 1..m$ 이 m개의 메모리 소자에 저장되도록 메모리(460) 내의 관련 메모리 소자를 업데이트하며, 각각의 메모리 소자는 1비트를 저장한다. 다시 말해, 이들 메모리 소자는 부호-비트 σ 값의 누적 곱을 유지한다. 이 곱은 XOR 게이트에 의해 얻어진다. 이전 단락에서 설명한 바와 같이, k번째 반복에 대한 메모리 소자(j)의 중간 값이 i=1,...n에 대해 $S_j^k(i)$ 에 의해 주어지는 경우, 누적 곱은 다음과 같이 주어진다.

$$S_j^k(i) = \begin{cases} S_j^k(i-1) \cdot \sigma_{i,j}^k & j \in B_i \text{ 인 경우} \\ S_j^k(i-1) & \text{그렇지 않은 경우} \end{cases} \quad (13)$$

[0065]

[0066] 예를 들어 (n · k)번째 시간의 경우와 같은 반복의 끝에서는

[0067] $$S_j^k = S_j^k(n) \quad (14)$$

[0068] 이다.

[0069] 각 시간 싸이클에서 계산된 $\rho_{i,j}^k,\ j \in B_i$ 은 또한 선입선출(FIFO) 버퍼(420-1)(도 4의 버퍼-1)에 전송된다. FIFO(420-1)는 d_c · q 열(각 싸이클에서 ρ 값의 d_c 수가 생성되고 각 ρ은 예시적인 실시예에서 q비트를 사용하여 표현된다) 및 n 행으로 구성된다. 시간 싸이클(n · (k-1)+i)에서, $\rho_{i,j}^k, j \in B_i$ 의 세트는 FIFO(420-1)의 후단에 공급되고 이전 반복으로부터의 $\rho_{i,j}^{k-1}, j \in B_i$ 세트는 버퍼(420-1)의 전단으로부터 판독된다. (n · k)번째 시간 싸이클에서, 모든 $P_j^k,\ j = 1...m$ 는 메모리(460)에서 이용가능하고 모든 $\rho_{i,j}^k,\ i = 1...n,\ j \in B_i$ 은 제 1 FIFO 버퍼(420-1)에서 이용가능하다. FIFO(420-1)의 최상단 행은 $\rho_{1,j}^k, j \in B_1$ 을 저장하고 그 행 이후에는 $\rho_{2,j}^k, j \in B_2$ 를 저장하며 버퍼(420-1)의 마지막 행은 $\rho_{n,j}^k, j \in B_n$ 으로 구성된다.

[0070] 각 시간 싸이클에서 계산된 $\sigma_{i,j}^k, j \in B_i$ 은 또 다른 FIFO 버퍼(420-2)(도 4의 버퍼-2)에 공급된다. 제 2 FIFO 버퍼(420-2)는 d_c 개수의 1비트 열(각 $\sigma_{i,j}$는 1비트를 사용하여 표현되기 때문)과 n개의 행으로 구성된다. 시간 싸이클(n · (k-1)+i)에서, $\sigma_{i,j}^k,\ j \in B_i$ 의 세트는 FIFO(420-2)의 후단에 공급되고 이전 반복으로부터의 $\sigma_{i,j}^{k-1},\ j \in B_i$ 의 세트는 버퍼(420-2)의 전단으로부터 판독된다.

[0071] 이제, 메모리(460)에 저장된 $P_j^{k-1}, j = 1...m$ 및 이전 반복(k-1)으로부터 제 1 FIFO 버퍼(420-1)에 저장된 $\rho_{i,j}^{k-1}, j \in B_i$ 로부터 체크-비트 노드 메시지($R_{j,i}^{k-1}$)의 크기를 계산하는 절차가 기술된다. 필요한 $P_j^{k-1}, j \in B_i$ 는 메모리(460)로부터 판독되고 $\rho_{i,j}^{k-1}, j \in B_i$ 은 제 1 FIFO 버퍼(420-1)로부터 판독된다. 그런 다음, 각각의 $j \in B_i$마다 병렬의 변환 크기 감산 유닛(470-1 내지 470-3)을 통해 차이 $\left(P_j^{k-1} - \rho_{i,j}^{k-1}\right)$ 가 계산되고 대응하는 결과는 함수 ϕ을 수행하는 병렬의 변환 유닛(475-1 내지 475-3)에 전달된다. 이들 변환 유닛(475)의 출력은 (k-1)번째 반복에서

d_c 체크 노드에서 i번째 비트 노드로의 대응 메시지의 크기이다. 즉, 방정식(9)에 따라 $|R_{j,i}^{k-1}|, j \in B_i$이다.

[0072] (k-1)번째 반복에서 체크-비트 노드 메시지의 부호$(\text{sign}(R_{j,i}^{k-1}))$는 유사한 방식으로 계산된다. 필요한 $S_j^{k-1}, j \in B_i$는 메모리(460)로부터 판독되고 $\sigma_{i,j}^{k-1}, j \in B_i$은 제 2 FIFO 버퍼(420-2)로부터 판독된다. 곱($S_j^{k-1} \cdot \sigma_{i,j}^{k-1}$)은 각각의 $j \in B_i$마다 병렬의 부호 처리 유닛(478-1 내지 478-3)(각각은 예시적인 실시예에서 XOR 게이트를 사용함)을 사용하여 계산된다. 이들 곱은 (k-1)번째 반복에서 d_c 체크 노드에서 i번째 비트 노드로의 대응 메시지의 부호 비트이다. 즉, 방정식(10)에 따라 $\text{sign}(R_{j,i}^{k-1}), j \in B_i$ 이다.

[0073] 체크-비트 노드 메시지의 부호 및 크기는 d_c 부호-크기를 거쳐 2의 보수 변환 유닛(480)에 전달된다. 이들 유닛(480)으로부터의 결과는 $R_{j,i}^{k-1}, j \in B_i$ 이며, 이는 이어서 비트 노드 업데이트 유닛(410)에 대한 입력이다. 비트 노드 업데이트 유닛(410)은

$$Q_{i,j}^{k} = \sum_{l \in B_i, l \neq j} R_{l,i}^{k-1} + \lambda_i \quad (15)$$

에 따라(방정식(3)도 참조) k번째 반복에 대해 비트-체크 노드 메시지($Q_{i,j}^{k}$)를 계산한다.

[0074] 메모리 요건 및 처리량

[0075] 예시적인 제 1 FIFO 버퍼(420-1)는 $n \cdot d_c \cdot q$ 비트의 크기를 가지며, 제 2 예시적인 제 2 FIFO 버퍼(420-2)는 $n \cdot d_c$ 비트의 크기를 갖는다.

[0076] 1...m,인 P_j,j에 대해 필요한 메모리의 양은 $2 \cdot q \cdot m$ 비트이며, 2의 배수는 반복(k 및 k-1)에 속하는 값에 대한 것이다.

[0077] 1...m,인 S_j,j에 대해 필요한 메모리의 양은 $2 \cdot m$ 비트이며, 2의 배수는 반복(k 및 k-1)에 속하는 값에 대한 것이다.

[0078] 전체 메모리 요건은 $(2 \cdot m+n \cdot d_c) \cdot (q+1)$ 비트이다.

[0079] 각 시간 싸이클에서, 개시된 방법은 d_c 체크-비트 노드 메시지와 d_c 비트-체크 노드 메시지를 계산한다. 따라서, 일 비트 노드 업데이트 유닛(410)을 통한 반복마다 길이 n 블록의 데이터를 처리하는데 n 싸이클이 사용된다.

[0080] 표준 LDPC 디코딩 아키텍처와 비교할 때, 개시된 아키텍처는 $(2 \cdot m+n \cdot d_c) \cdot (q+1)$ 비트의 메모리 공간만을 필요로 하며 반복마다 n 싸이클만을 필요로 한다. 전형적인 직렬 합-곱 아키텍처는 $2 \cdot n \cdot d_c \cdot (q+1)$ 비트의 메모리 공간과 반복 당 m+n 싸이클을 필요로 한다.

[0081] 직렬 합-곱 아키텍처를 갖는 연결 LDPC 디코더

[0082] LDPC 코드는 심볼간 간섭(ISI) 및 잡음에 의해 손상되는 채널을 위해 사용되어 비트 에러 레이트를 개선할 수 있다. 도 5는 LDPC 인코더(510), ISI 채널(520)(심볼간 간섭 및 잡음을 야기함), 소프트-입력 소프트-출력(SISO) 검출기(615) 및 LDPC 디코더(600)를 포함하는 전형적인 통신 시스템(500)을 도시한다. 도 5에서, LDPC 디코더(600)는 SISO 검출기(615)와 연결되며, 이는 도 6과 연계하여 이하에서 더 설명된다. SISO 검출기(615)는 LDPC 디코더(600)로부터 사전 정보 값으로 사용되는 채널 출력 및 부대(extrinsic) 정보 값을 취한다. SISO 검출기(615)는 다음 반복을 위해 LDPC 디코더(600)에 의해 사전 정보 값(λ_i)으로서 사용되는 부대 정보 값을 제공한다. LDPC 디코더로부터의 부대 정보 값은 SISO 검출기를 위한 사전 정보 값으로서 사용되고 SISO 검출기로부터의 부대 정보는 LDPC 디코더를 위한 사전 정보 값으로서 사용된다. SISO 검출기는 SIS 채널을 고려하여 예를 들어 당업계에 공지된 MAP 알고리즘 또는 소프트 출력 비터비 알고리즘(SOVA)을 사용하여 부대 정보를 계산하는 반면, LDPC 디코더는 LDPC 코드를 고려하여 부대 정보 값을 계산한다. 시스템의 반복은 일단 SISO 검출

기(615) 및 LDPC 디코더(600)에 의한 데이터 처리로 구성된다.

[0083] 도 6은 개시된 연결 SISO 검출기 및 LDPC 디코더 아키텍처를 보다 상세히 도시한다. SISO 검출기(615)는 LDPC 디코더(600)로부터 채널 출력 및 부대 정보 값을 취하고 다음 반복을 위해 LDPC 디코더(600)에 의해 사전 정보 값으로서 사용되는 부대 정보 값을 제공한다. 본 발명은 LDPC 디코더(600)를 통과할 때마다, 비트 노드(i)에서 그의 체크 노드로의 d_c 메시지, 즉 $\hat{Q}_{i,j}, \forall j \in B_i$ 은 비트 노드(i)의 사전 정보 값 또는 사전 LLR λ_i와 동일하다. 즉,

[0084] $$\hat{Q}_{i,j} = \lambda_i, \quad \forall j \in B_i \tag{16}$$

[0085] 체크 노드 계산은 상술한 유사한 방식으로 분할된다.

[0086] $$\hat{\rho}_i = \phi(|\lambda_i|) \tag{17}$$

[0087] $$\hat{\sigma}_i = \text{sign}(\lambda_i) \tag{18}$$

[0088] $$\hat{P}_j = \sum_{l \in C_j} \hat{\rho}_l \tag{19}$$

[0089] $$\hat{S}_j = \prod_{l \in C_j} \hat{\sigma}_l \tag{20}$$

[0090] 체크 노드(j)에서 비트 노드(i)로의 메시지의 부호 및 크기는 다음과 같이 주어진다.

[0091] $$\left|\hat{R}_{j,i}\right| = \phi\left(\hat{P}_j - \hat{\rho}_i\right) \tag{21}$$

[0092] $$\text{sign}\left(\hat{R}_{j,i}\right) = \hat{S}_j \cdot \sigma_i \tag{22}$$

[0093] 이들 $\hat{R}_{j,i}, j \in B_i$ 을 사용하면, SISO 검출기(615)로 전달되는 LDPC 디코더로부터의 부대 정보 값이 부대 정보 계산 유닛(610)에 의해 계산될 수 있다. 즉,

[0094] $$\Lambda_{ext,i} = \sum_{l \in B_i} R_{l,i} \tag{23}$$

[0095] 이 부대 정보 값은 사전 정보 값으로서 SISO 검출기(615)에 의해 사용된다.

[0096] 도 6은 연결형 시스템에 대해 제안된 아키텍처(600)에 대한 블록도이며, d_c는 3이고, B_i는 {j1,j2,j3}이다. SISO 검출기(615)로부터 주어진 부대 정보 값과 동일한, 비트 노드(i)에 속하는 각 사전 LLR λ_i는 부호-크기에 대한 2의 보수 변환 유닛(630)에 전달된다. 크기는 함수(ϕ)를 수행하는 변환 유닛(635)에 전송되고 유닛(635)으로부터의 출력은 방정식(17)에 의해 정의된 $\hat{\rho}_i^k$ 이다. k번째 반복 및 i번째 비트 노드에 대한 변환된 크기($\hat{\rho}_i^k$)는 메모리 내의 $\hat{P}_j^k, \forall j \in B_i$ 을 업데이트하는 d_c 변환된 크기 업데이트 유닛(640)에 공급된다. 유사하게, 부호($\hat{\sigma}_i^k$)는 메모리 내의 $\hat{S}_j^k, \forall j \in B_i$ 을 업데이트하는 부호 업데이트 유닛(650)의 또 다른 세트에 전달된다. 또한, $\hat{\rho}_i^k$ 및 $\hat{\sigma}_i^k$ 은 제 1 FIFO 버퍼(620-1) 및 제 2 버퍼(620-2)에 각각 공급된다.

[0097] (k-1)번째 반복에 대한 체크-비트 노드 메시지, 즉 $\hat{R}_{j,i}^{k-1}, j \in B_i$ 는 도 4와 관련하여 상술한 절차와 유사한 방식으로 계산된다. 그러나, 제 1 FIFO 버퍼(620-1)로부터 하나의 값만이 사용되고($\hat{\rho}_i^{k-1}$이 사용되어 크기를 얻음) 제 2 버퍼(620-2)로부터 하나의 값이 사용된다($\hat{\sigma}_i^{k-1}$ 이 사용되어 부호를 얻음). 따라서, FIFO 버퍼(620)에 대한 메모리 요건은 도 4와 관련하여 상술한 독립형 아키텍처보다 적은 d_c 배이다.

[0098] 메모리 요건 및 처리량

[0099] 예시적인 제 1 FIFO 버퍼(620-1)는 $n \cdot q$ 비트의 크기를 가지며, 제 2 예시적인 제 2 FIFO 버퍼(620-2)는 n 비

트의 크기를 갖는다.

[0100] 1...m,인 $P_{j,j}$에 대해 필요한 메모리의 양은 $2 \cdot q \cdot m$ 비트이다.

[0101] 1...m,인 $S_{j,j}$에 대해 필요한 메모리의 양은 $2 \cdot m$ 비트이다.

[0102] 전체 메모리 요건은 $(2 \cdot m+n \cdot d_c) \cdot (q+1)$ 비트이다.

[0103] 각 시간 싸이클에서, 개시된 방법은 d_c 체크-비트 노드 메시지와 부대 정보 값을 계산한다. 따라서, 반복마다 길이 n 블록의 데이터를 처리하는데 n 싸이클이 사용된다.

[0104] 개시된 아키텍쳐는 방정식(17-23)에 정의된 합-곱 알고리즘을 수행한다. 개시된 방법은 Wu 및 Burd에 의해 제안된 보다 덜 최적인 방법보다 나은 에러 레이트 성능을 갖는데, 최소 LLR 값만이 합-곱 알고리즘에서 고려된다. 또한 Wu 및 Burd에 의해 제안된 방법은 연결 LDPC 디코더 시스템에만 적용된다. 본 발명은 $(2 \cdot m+n) \cdot (q+1)$ 메모리 공간을 필요로 하며 반복 당 n 싸이클만이 사용된다.

[0105] 표기

[0106] 후속하는 표기는 본 명세서에서 사용된다. 즉

[0107] i는 비트 노드의 인덱스,

[0108] j는 체크 노드의 인덱스,

[0109] k는 반복의 인덱스,

[0110] $Q_{i,j}$는 비트 노드(i)에서 체크 노드(j)로의 메시지,

[0111] $R_{j,i}$는 체크 노드(j)에서 비트 노드(i)로의 메시지,

[0112] λ_i는 비트(i)에 속하는 사전 정보 값 또는 사전 로그-확률 비율(LLR),

[0113] Λ_i는 귀납적 정보 값 또는 비트(i)에 속하는 귀납적 LLR,

[0114] $\Lambda_{ext,i}$는 부대 정보 값 또는 비트(i)에 속하는 부대 LLR,

[0115] B_i는 비트 노드(i)에 연결된 체크 노드 세트,

[0116] C_j는 체크 노드(j)에 연결된 비트 노드 세트,

[0117] n은 비트 노드의 개수,

[0118] m은 패리티 체크 노드의 개수,

[0119] d_r은 패리티 체크 매트릭스의 행 가중치,

[0120] d_c는 패리티 체크 매트릭스의 열 가중치,

[0121] 비트 노드는 1...n,

[0122] 체크 노드는 1...m이다.

[0123] 본 발명의 예시적인 실시예는 디지털 로직 유닛과 관련하여 기술되었지만, 당업자에게 분명한 것은, 소프트웨어 프로그램 내에서, 회로 소자 또는 상태 머신에 의한 하드웨어에서, 또는 소프트웨어와 하드웨어의 조합 내에서 프로세싱 단계로서 다양한 기능이 디지털 도메인에서 구현될 수 있다는 것이다. 이러한 소프트웨어는 예를 들어 디지털 신호 프로세서, 마이크로-제어기, 또는 범용 컴퓨터 내에 이용될 수 있다.

[0124] 따라서, 본 발명의 기능은 이들 방법을 실행하는 방법 및 장치의 형태로 구현될 수 있다. 본 발명의 하나 이상의 측면은 예를 들어 저장 매체에 저장되거나, 머신내로 로딩 및/또는 머신에 의해 실행되거나, 또는 소정의 전송 매체를 통해 전송되는 프로그램 코드의 형태로 구현될 수 있는데, 이 프로그램 코드가 컴퓨터와 같은 머신내

로 로딩 및 머신에 의해 실행되는 경우, 머신은 본 발명을 실행하는 장치가 된다. 범용 프로세서 상에서 구현되는 경우, 프로그램 코드 세그먼트는 프로세서와 결합하여 특정 로직 회로와 유사하게 동작하는 장치를 제공한다.

[0125] 당업계에 알려져 있는 바와 같이, 본 명세서에서 기술한 방법 및 장치는 컴퓨터 판독가능 코드 수단이 포함된 컴퓨터 판독가능 매체를 포함하는 제조 물품으로서 분배될 수 있다. 컴퓨터 판독가능 프로그램 코드 수단은 컴퓨터 시스템과 연계하여 동작가능하며 본 명세서에서 기술된 방법을 수행하는 단계 전부 또는 일부를 수행하거나 또는 본 명세서에서 기술된 장치를 생성한다. 컴퓨터 판독가능 매체는 기록가능 매체(예를 들어, 플로피 디스크, 하드 드라이브, 컴팩트 디스크, 메모리 카드, 반도체 디바이스, 칩, 주문형 집적 회로(ASIC))일 수 있으며, 또는 전송 매체(예를 들어, 광섬유, 월드-와이드 웹, 케이블, 또는 시분할 다중 접속, 코드 분할 다중 접속을 사용하는 무선 채널 또는 그 밖의 무선 주파수 채널을 포함하는 네트워크)일 수 있다. 컴퓨터 시스템에 사용되기에 적절한 정보를 저장할 수 있는 임의의 공지된 또는 개발된 매체가 사용될 수 있다. 컴퓨터 판독가능 코드 수단은 컴퓨터로 하여금 자기 매체 상의 자기 변화 또는 컴팩트 디스크의 표면 상의 높이 변화와 같은 인스트럭션 및 데이터를 판독할 수 있게 해주는 임의의 메카니즘이다.

[0126] 메모리 및 버퍼는 분배될 수도 있고 국부적일 수도 있으며 프로세서는 분배될 수도 있고 단 하나일 수도 있다. 메모리 및 버퍼는 전기, 자기 또는 광학 메모리로서, 또는 이들 또는 다른 유형의 저장 장치의 임의의 조합으로서 구현될 수 있다. 또한, "메모리", "버퍼" 및 "FIFO 버퍼"라는 용어는 매체로부터 판독 또는 매체에 기록될 수 있는 임의의 정보를 포함하도록 충분히 넓게 해석되어야 한다. 이러한 정의에 따르면, 네트워크 상의 정보는 여전히 메모리 내에 존재하는데, 그 이유는 관련 프로세서가 네트워크로부터 정보를 검색할 수 있기 때문이다.

[0127] 본 명세서에서 기술하고 도시한 실시예 및 변형은 단지 본 발명의 원리에 대한 예시일 뿐이며, 본 발명의 범주 및 사상을 벗어나지 않고서 당업자에 의해 다양한 수정이 구현될 수 있음을 이해해야 한다.

도면의 간단한 설명

[0128] 도 1은 LDPC 매트릭스 H의 일반적 구조를 예시한 도면,

[0129] 도 2는 LDPC 코드의 예시적인 이원 그래프 표현도,

[0130] 도 3은 예시적인 하드웨어 공유 LDPC 디코더 아키테처의 블록도,

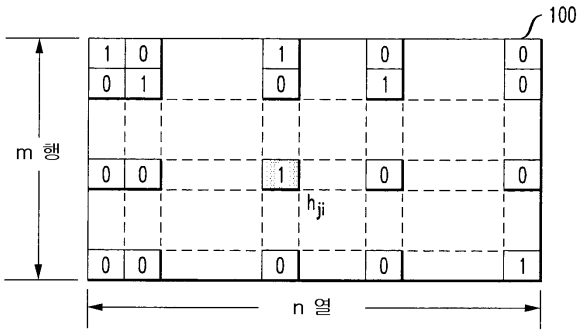
[0131] 도 4는 본 발명의 특징을 포함하는 LDPC 디코더의 블록도,

[0132] 도 5는 LDPC 디코더가 SISO(soft input soft output) 검출기와 연결된 전형적인 통신 시스템의 블록도,

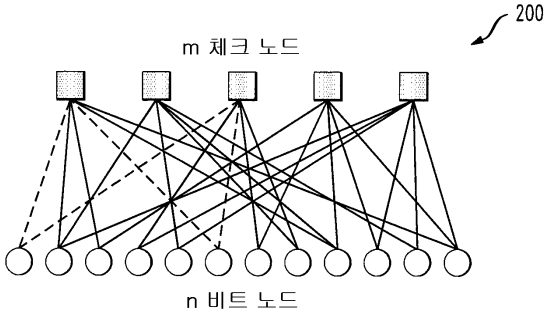
[0133] 도 6은 SISO 검출기와 연결된 LDPC 디코더이 블록도이다.

도면

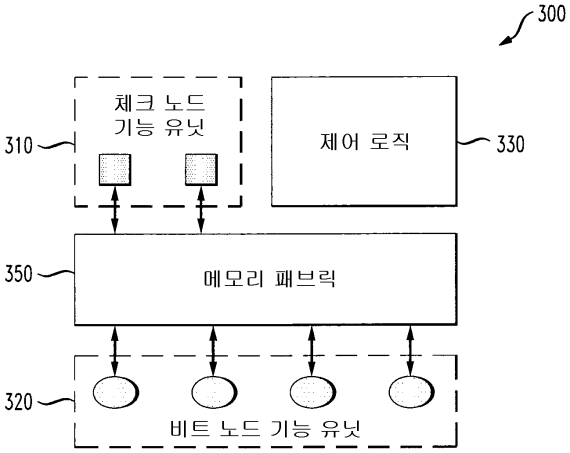
도면1



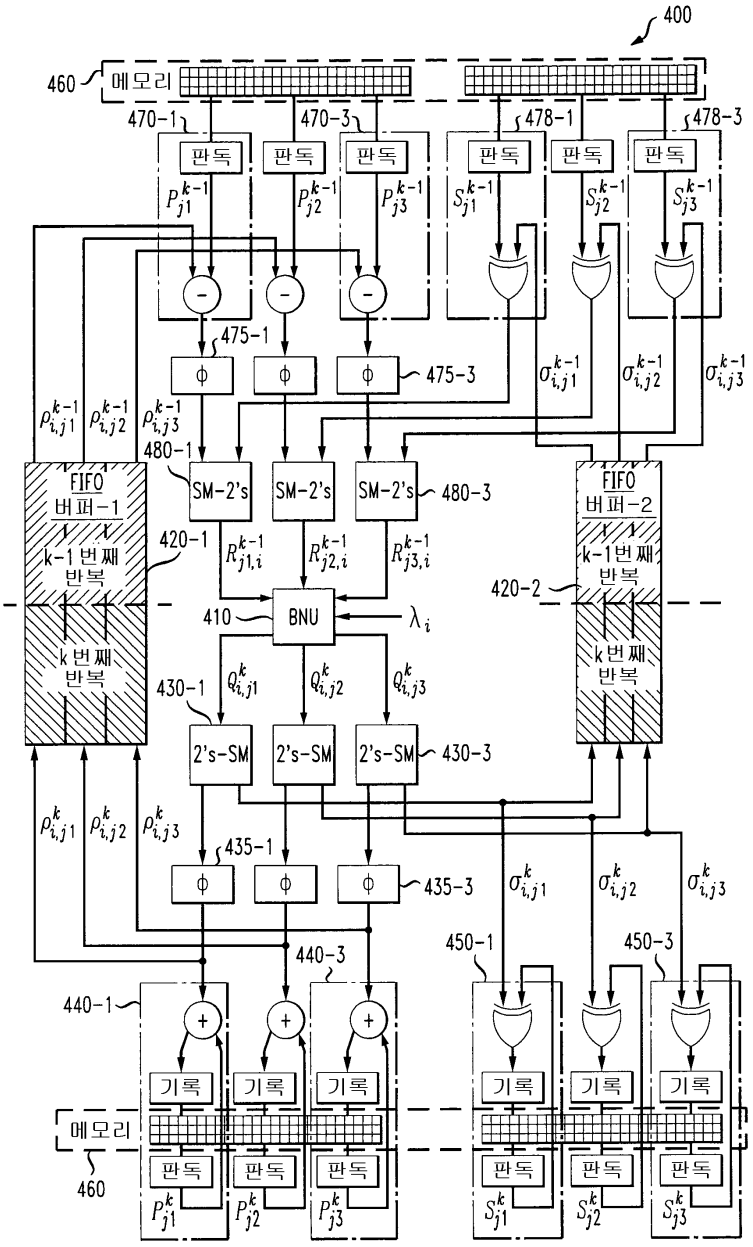
도면2


200
m 체크 노드
n 비트 노드

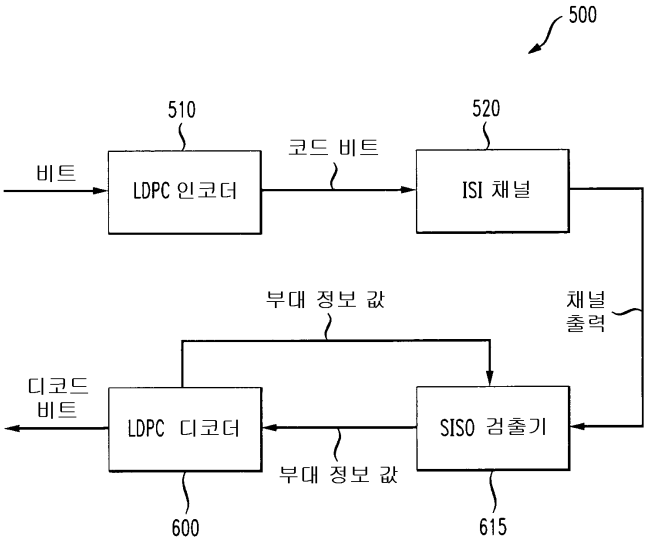
도면3


300
체크 노드
기능 유닛
310
제어 로직
330
350
메모리 패브릭
320
비트 노드 기능 유닛

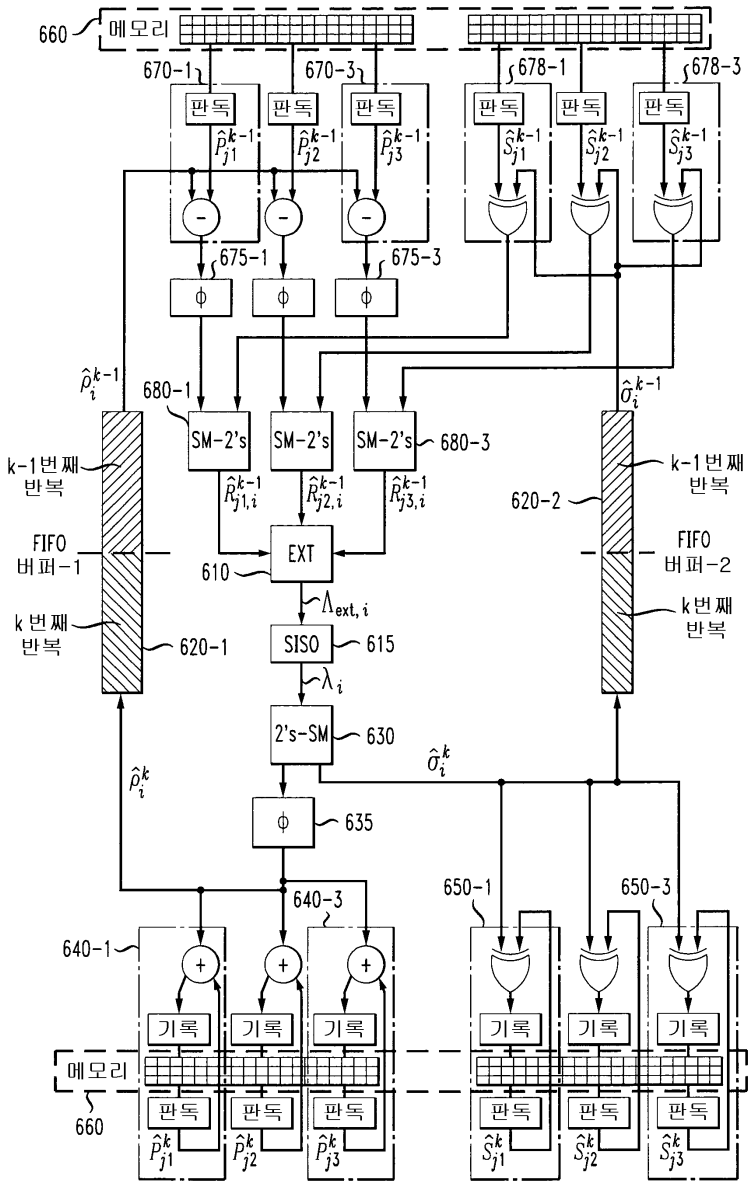
도면4


400
460
메모리
470-1
470-3
478-1
478-3
판독
475-1
475-3
480-1
480-3
SM-2's
FIFO
버퍼-1
k-1 번째
반복
k 번째
반복
420-1
420-2
410
BNU
430-1
430-3
2's-SM
435-1
435-3
440-1
440-3
450-1
450-3
기록
메모리
460

도면5


500
510
520
비트
LDPC 인코더
코드 비트
ISI 채널
부대 정보 값
재널
출력
디코드
비트
LDPC 디코더
SISO 검출기
부대 정보 값
600
615

도면6


660
메모리
670-1
670-3
678-1
678-3
판독
675-1
675-3
680-1
680-3
SM-2's
k-1번째 반복
620-2
FIFO 버퍼-1
FIFO 버퍼-2
610
EXT
k번째 반복
620-1
SISO
615
2's-SM
630
635
640-1
640-3
650-1
650-3
기록