(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2023-0116051
(43) 공개일자 2023년08월03일

(51) 국제특허분류(Int. Cl.)
H03M 13/45 (2006.01) *G06F 11/10* (2006.01)
H03M 13/00 (2017.01) *H03M 13/13* (2006.01)
(52) CPC특허분류
H03M 13/45 (2013.01)
G06F 11/1012 (2013.01)
(21) 출원번호 10-2023-7023172
(22) 출원일자(국제) 2021년12월07일
심사청구일자 없음
(85) 번역문제출일자 2023년07월07일
(86) 국제출원번호 PCT/US2021/062201
(87) 국제공개번호 WO 2022/125545
국제공개일자 2022년06월16일
(30) 우선권주장
17/116,952 2020년12월09일 미국(US)

(71) 출원인
어드밴스드 마이크로 디바이시즈, 인코포레이티드
미국 캘리포니아 95054 산타 클라라 어거스틴 드라이브 2485
(72) 발명자
라 페트라 로스 브이.
미국 캘리포니아 95054 산타 클라라 어거스틴 드라이브 2485
(74) 대리인
박장원

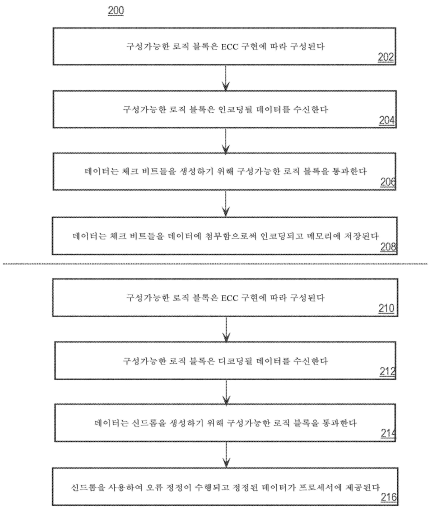
전체 청구항 수 : 총 20 항

(54) 발명의 명칭 **프로그래밍가능 오류 정정 코드 인코딩 및 디코딩 로직**

(57) 요 약

메모리 모듈은 특정 ECC 구현으로 구성가능한 로직 엘리먼트들을 포함한다. 본 명세서에서 사용되는 바와 같이, 용어 "ECC 구현"은, 오류 검출 및 후속 프로세싱, 예를 들어, 오류 검출의 결과를 사용하여 오류 정정을 수행하고 임의의 오류가 나중에 식별되고 정정될 수 있도록 데이터를 인코딩하는 것을 수행하기 위한 ECC 기능을 지칭한다. 접근법은 메모리 모듈 또는 컴퓨팅 디바이스가 호스트 사이에서 앞뒤로 발송될 요청들을 요구하지 않고 특정 ECC 구현에 구성될 수 있게 한다.

대 표 도 - 도2



(52) CPC특허분류
G06F 11/1076 (2013.01)
H03M 13/13 (2013.01)
H03M 13/616 (2013.01)
H03M 13/6508 (2013.01)

명 세 서

청구범위

청구항 1

방법으로서,

구성가능한 로직 엘리먼트들의 세트에서, 오류 정정 코드(ECC) 구현을 위해 상기 구성가능한 로직 엘리먼트들의 세트에서 마스킹할 비트들을 정의하는 구성 명령어들을 수신하는 단계;

상기 구성가능한 로직 엘리먼트들의 세트에서, 상기 ECC 구현에 따라 인코딩 또는 디코딩될 데이터를 수신하는 단계;

상기 구성 명령어들에 따라 마스킹된 데이터 또는 코드 워드 비트들을 갖는 상기 구성가능한 로직 엘리먼트들의 세트를 사용하여 상기 ECC 구현을 실행하는 단계를 포함하는, 방법.

청구항 2

제1항에 있어서,

상기 구성 명령어들은 상기 ECC 구현에 대응하는 G-행렬에 따라 상기 로직 엘리먼트들에서 마스킹할 비트들을 정의하는 것을 포함하고;

상기 ECC 구현을 실행하는 단계는 데이터를 인코딩하는 단계를 포함하고, 상기 데이터를 인코딩하는 단계는 특정 데이터 값에 대해,

상기 구성 명령어들에 따라 마스킹된 비트들을 갖는 엘리먼트들을 사용하여 상기 데이터 값에 대한 복수의 체크 비트 값들을 결정하는 단계; 및

상기 특정 데이터 값에 대한 코드 워드를 생성 및 저장하는 단계를 수행하는 것을 포함하고, 상기 코드 워드는 상기 복수의 체크 비트들 및 상기 데이터 값의 데이터를 포함하는, 방법.

청구항 3

제2항에 있어서, 상기 복수의 체크 비트 값들을 결정하는 단계는, 상기 구성 명령어들에 따라 마스킹된 비트들을 갖는 상기 로직 엘리먼트를 사용하여, 상기 복수의 체크 비트들을 생성하기 위해 상기 코드 워드에 상기 G-행렬을 곱하는 단계를 포함하는, 방법.

청구항 4

제1항에 있어서,

상기 구성 명령어들은 상기 ECC 구현에 대응하는 H-행렬에 따라 상기 로직 엘리먼트들에서 마스킹할 비트들을 정의하는 것을 포함하고;

상기 ECC 구현을 실행하는 단계는 코드워드를 디코딩하는 단계를 포함하고, 상기 코드워드를 디코딩하는 단계는,

상기 구성 명령어들에 따라 마스킹된 비트들을 갖는 상기 로직 엘리먼트들을 사용하여,상기 코드워드로부터 신드롬 열(column)을 계산하는 단계;

상기 신드롬 열과 매칭되는 상기 H-행렬의 특정 열을 식별하고, 이에 응답하여, 상기 H-행렬의 상기 특정 열에 대응하는 상기 코드워드의 값에 오류가 존재한다고 결정하는 단계를 포함하는, 방법.

청구항 5

제1항에 있어서,

상기 구성 명령어들은 상기 ECC 구현에 대응하는 H-행렬에 따라 상기 로직 엘리먼트들에서 마스킹할 비트들을

정의하는 것을 포함하고;

상기 ECC 구현을 실행하는 단계는 코드워드를 디코딩하는 단계를 포함하고, 상기 코드워드를 디코딩하는 단계는,

상기 H-행렬의 각각의 심볼 세트에 대해, 행들(rows)의 제1 세트에 대응하는 값들의 제1 세트 대 행들의 제2 세트에 대응하는 값들의 제2 세트의 비율들의 세트를 저장하는 단계;

상기 구성 명령어들에 따라 마스킹된 비트들을 갖는 상기 로직 엘리먼트들을 사용하여 신드롬을 계산하는 단계;

행들의 상기 제1 세트에 대응하는 상기 신드롬의 제1 부분이 특정 심볼 세트에 대한 비율과 행들의 상기 제2 세트에 대응하는 상기 신드롬의 제2 부분의 곱과 동일하다고 결정하고, 이에 응답하여, 상기 특정 심볼 세트에 대응하는 상기 데이터의 부분이 오류를 포함한다고 결정하는 단계를 포함하는, 방법.

청구항 6

제5항에 있어서, 상기 코드워드를 디코딩하는 단계는,

상기 신드롬의 상기 제1 부분의 몫(quotient)을 상기 특정 심볼에 대한 행들의 상기 제1 세트에 대응하는 값들의 상기 제1 세트로 계산함으로써 상기 오류의 크기를 결정하는 단계;

상기 오류의 크기 및 상기 오류를 포함하는 것으로 결정된 상기 코드워드의 부분을 사용하여 상기 데이터를 정정하는 단계를 더 포함하는, 방법.

청구항 7

제1항에 있어서, 상기 구성가능한 로직 엘리먼트들은 메모리 내에 코드워드를 저장하기 전에 프로세서에 의해 생성된 데이터의 인코딩 또는 상기 메모리 내에 저장된 상기 코드워드의 디코딩을 수행함으로써 메모리 모듈에 대한 상기 ECC 구현을 제공하도록 구성되는, 방법.

청구항 8

제1항에 있어서, 상기 구성가능한 로직 엘리먼트들은 네트워크를 통해 발송되거나 수신된 데이터에 대한 상기 ECC 구현을 제공하도록 구성되는, 방법.

청구항 9

장치로서,

구성가능한 로직 엘리먼트들의 세트를 포함하고;

상기 구성가능한 로직 엘리먼트들의 세트는 오류 정정 코드(ECC) 구현을 위해 상기 구성가능한 로직 엘리먼트들의 세트에서 마스킹할 비트들을 정의하는 구성 명령어들에 따라 구성되고;

상기 구성가능한 로직 엘리먼트들의 세트는 상기 구성 명령어들에 따라 상기 구성가능한 로직 엘리먼트들의 세트 내의 데이터 또는 코드워드 비트들을 마스킹함으로써 상기 ECC 구현을 실행하도록 구성되는, 장치.

청구항 10

제10항에 있어서,

상기 구성 명령어들은 상기 ECC 구현에 대응하는 G-행렬에 따라 상기 로직 엘리먼트들에서 마스킹할 비트들을 정의하는 것을 포함하고;

상기 ECC 구현을 실행하는 것은 데이터를 인코딩하는 것을 포함하고, 상기 데이터를 인코딩하는 것은 특정 데이터 값에 대해,

상기 구성 명령어들에 따라 마스킹된 비트들을 갖는 엘리먼트들을 사용하여 상기 데이터 값에 대한 복수의 체크 비트 값들을 결정하는 것; 및

상기 특정 데이터 값에 대한 코드 워드를 생성 및 저장하는 것을 수행하는 것을 포함하고, 상기 코드 워드는 상기 복수의 체크 비트들 및 상기 데이터 값의 데이터를 포함하는, 장치.

청구항 11

제10항에 있어서, 상기 복수의 체크 비트 값들을 결정하는 것은, 상기 구성 명령어들에 따라 마스킹된 비트들을 갖는 상기 로직 엘리먼트를 사용하여, 상기 복수의 체크 비트들을 생성하기 위해 상기 코드 워드에 상기 G-행렬을 곱하는 것을 포함하는, 장치.

청구항 12

제9항에 있어서,

상기 구성 명령어들은 상기 ECC 구현에 대응하는 H-행렬에 따라 상기 로직 엘리먼트들에서 마스킹할 비트들을 정의하는 것을 포함하고;

상기 ECC 구현을 실행하는 것은 코드워드를 디코딩하는 것을 포함하고, 상기 코드워드를 디코딩하는 것은,

상기 구성 명령어들에 따라 마스킹된 비트들을 갖는 상기 로직 엘리먼트들을 사용하여,상기 코드워드로부터 신드롬 열을 계산하는 것;

상기 신드롬 열과 매칭되는 상기 H-행렬의 특정 열을 식별하고, 이에 응답하여, 상기 H-행렬의 상기 특정 열에 대응하는 상기 코드워드의 값에 오류가 존재한다고 결정하는 것을 포함하는, 장치.

청구항 13

제9항에 있어서,

상기 구성 명령어들은 상기 ECC 구현에 대응하는 H-행렬에 따라 상기 로직 엘리먼트들에서 마스킹할 비트들을 정의하는 것을 포함하고;

상기 ECC 구현을 실행하는 것은 코드워드를 디코딩하는 것을 포함하고, 상기 데이터를 디코딩하는 것은,

상기 H-행렬의 각각의 심볼 세트에 대해, 행들의 제1 세트에 대응하는 값들의 제1 세트 대 행들의 제2 세트에 대응하는 값들의 제2 세트의 비율들의 세트를 저장하는 것;

상기 구성 명령어들에 따라 마스킹된 비트들을 갖는 상기 로직 엘리먼트들을 사용하여 신드롬을 계산하는 것;

행들의 상기 제1 세트에 대응하는 상기 신드롬의 제1 부분이 특정 심볼 세트에 대한 비율과 행들의 상기 제2 세트에 대응하는 상기 신드롬의 제2 부분의 곱과 동일하다고 결정하고, 이에 응답하여, 상기 특정 심볼 세트에 대응하는 상기 코드워드의 부분이 오류를 포함한다고 결정하는 것을 포함하는, 장치.

청구항 14

제13항에 있어서, 상기 코드워드를 디코딩하는 것은,

상기 신드롬의 상기 제1 부분의 몫을 상기 특정 심볼에 대한 행들의 상기 제1 세트에 대응하는 값들의 상기 제1 세트로 계산함으로써 상기 오류의 크기를 결정하는 것;

상기 오류의 크기 및 상기 오류를 포함하는 것으로 결정된 상기 코드워드의 부분을 사용하여 상기 데이터를 정정하는 것을 더 포함하는, 장치.

청구항 15

제9항에 있어서, 상기 구성가능한 로직 엘리먼트들은 메모리 내에 코드워드를 저장하기 전에 프로세서에 의해 생성된 데이터의 인코딩 또는 상기 메모리 내에 저장된 코드워드의 디코딩을 수행함으로써 메모리 모듈에 대한 상기 ECC 구현을 제공하도록 구성되는, 장치.

청구항 16

제9항에 있어서, 상기 구성가능한 로직 엘리먼트들은 네트워크를 통해 발송되거나 수신된 데이터에 대한 상기 ECC 구현을 제공하도록 구성되는, 장치.

청구항 17

메모리 모듈로서,

오류 정정 코드(ECC) 구현에 따라 인코딩된 데이터를 저장하는 메모리;

상기 메모리에 저장된 데이터에 대한 동작들을 수행하고, 상기 메모리에 추가 데이터를 기록하도록 구성된 프로세서;

오류 정정 코드(ECC) 구현을 위해 구성가능한 로직 엘리먼트들의 세트에서 마스킹할 비트들을 정의하는 구성 명령어들에 따라 구성된 상기 구성가능한 로직 엘리먼트들의 세트를 포함하고;

상기 구성가능한 로직 엘리먼트들의 세트는 상기 메모리에 대한 기록 전에 또는 상기 프로세서에 의한 판독 전에 상기 구성 명령어들에 따라 상기 구성가능한 로직 엘리먼트들의 세트 내의 비트들을 마스킹함으로써 상기 ECC 구현을 실행하도록 구성되는, 메모리 모듈.

청구항 18

제17항에 있어서,

상기 구성 명령어들은 상기 ECC 구현에 대응하는 G-행렬에 따라 상기 로직 엘리먼트들에서 마스킹할 비트들을 정의하는 것을 포함하고;

상기 ECC 구현을 실행하는 것은 데이터를 인코딩하는 것을 포함하고, 상기 데이터를 인코딩하는 것은 특정 데이터 값에 대해,

상기 구성 명령어들에 따라 마스킹된 비트들을 갖는 엘리먼트들을 사용하여 상기 데이터 값에 대한 복수의 체크 비트 값들을 결정하는 것; 및

상기 특정 데이터 값에 대한 코드 워드를 생성 및 저장하는 것을 수행하는 것을 포함하고, 상기 코드 워드는 상기 복수의 체크 비트들 및 상기 데이터 값의 데이터를 포함하는, 메모리 모듈.

청구항 19

제17항에 있어서,

상기 구성 명령어들은 상기 ECC 구현에 대응하는 H-행렬에 따라 상기 로직 엘리먼트들에서 마스킹할 비트들을 정의하는 것을 포함하고;

상기 ECC 구현을 실행하는 것은 코드워드를 디코딩하는 것을 포함하고, 상기 코드워드를 디코딩하는 것은,

상기 구성 명령어들에 따라 마스킹된 비트들을 갖는 상기 로직 엘리먼트들을 사용하여,상기 코드워드로부터 신드롬 열을 계산하는 것;

상기 신드롬 열과 매칭되는 상기 H-행렬의 특정 열을 식별하고, 이에 응답하여, 상기 H-행렬의 상기 특정 열에 대응하는 상기 코드워드의 값에 오류가 존재한다고 결정하는 것을 포함하는, 메모리 모듈.

청구항 20

제17항에 있어서,

상기 구성 명령어들은 상기 ECC 구현에 대응하는 H-행렬에 따라 상기 로직 엘리먼트들에서 마스킹할 비트들을 정의하는 것을 포함하고;

상기 ECC 구현을 실행하는 것은 코드워드를 디코딩하는 것을 포함하고, 상기 코드워드를 디코딩하는 것은,

상기 H-행렬의 각각의 심볼 세트에 대해, 행들의 제1 세트에 대응하는 값들의 제1 세트 대 행들의 제2 세트에 대응하는 값들의 제2 세트의 비율들의 세트를 저장하는 것;

상기 구성 명령어들에 따라 마스킹된 비트들을 갖는 상기 로직 엘리먼트들을 사용하여 신드롬을 계산하는 것;

행들의 상기 제1 세트에 대응하는 상기 신드롬의 제1 부분이 특정 심볼 세트에 대한 비율과 행들의 상기 제2 세트에 대응하는 상기 신드롬의 제2 부분의 곱과 동일하다고 결정하고, 이에 응답하여, 상기 특정 심볼 세트에 대응하는 상기 코드워드의 부분이 오류를 포함한다고 결정하는 것;

상기 신드롬의 상기 제1 부분의 몫을 상기 특정 심볼에 대한 행들의 상기 제1 세트에 대응하는 값들의 상기 제1

세트로 계산함으로써 상기 오류의 크기를 결정하는 것;

상기 오류의 크기 및 상기 오류를 포함하는 것으로 결정된 상기 코드워드의 부분을 사용하여 상기 코드워드를 정정하는 것을 포함하는, 메모리 모듈.

발명의 설명

기 술 분 야

배 경 기 술

[0001] 이 섹션에 설명되는 접근법은 추구할 수 있는 접근법이지만, 반드시 이전에 구상되거나 추구되었던 접근법은 아니다. 따라서, 달리 표시되지 않는 한, 이 섹션에 설명된 임의의 접근법이 단지 이 섹션에 포함되는 것에 의해 종래 기술로서 자격을 갖는다고 가정되어서는 안 된다. 또한, 이 섹션에 설명된 임의의 접근법들이 단지 이 섹션에 포함된 것들에 의해 잘 이해되고, 일상적이거나, 통상적이라고 가정되어서는 안 된다.

[0002] 오류 정정 코드(Error Correction Code; ECC) 메모리 모듈은 내부 데이터 손상(corruption)을 검출하고 정정하기 위해 ECC 엔진을 구현한다. 종래의 ECC 메모리 모듈들의 제한들 중 하나는, 특정 ECC 구현이 제조자에 의해 구성되고 제조자의 접근법에 독점적인 경향이 있다는 것이다. 따라서, 제조자가 특정 ECC 구현으로 호스트를 구성하는 경우, 데이터에 액세스하는 임의의 메모리 제어기는 동일한 ECC 구현으로 구성되어야 한다. 본 명세서에서 사용되는 바와 같이, 용어 "호스트"는 중앙 프로세싱 유닛(CPU), 그래픽 프로세싱 유닛(GPU), 메모리 제어기 등과 같은 임의의 유형의 프로세싱 엘리먼트를 지칭한다.예를 들어, 프로세서-인-메모리(PIM) 기반 메모리 모듈이 내부 오류 검출 및 정정을 위해 결과 비트들을 사용하고, 정정될 수 없는 손상된 데이터를 식별하기 위해, PIM 기반 메모리 모듈 및 호스트는 동일한 레벨의 오류 보호를 보장하고 결과 비트들의 일관성 없는 사용을 피하기 위해 동일한 ECC 구현을 사용해야 한다. 이것은 PIM 기반 메모리 모듈들이 다수의 호스트들과 함께 사용하도록 설계될 수 있지만, 예를 들어 시스템 온 칩(System-on-a-Chip; SoC)의 호스트 설계자 또는 회사가 PIM 기반 메모리 벤더가 독점적일 수 있는 그들이 사용하고 있는 ECC 구현을 알기를 원하지 않을 수 있기 때문에 문제가 된다. 유사하게, PIM 기반 메모리 모듈들을 사용하는 이점은 데이터가 PIM 기반 메모리 모듈로부터 외부 프로세서로 전송(transfer)될 필요가 없기 때문에 데이터가 프로세싱될 수 있는 속도이다. 따라서, 메모리 모듈에 저장된 데이터를 디코딩하기 위해 ECC를 구현하도록 구성될 수 있는 프로그래밍가능 메모리 모듈에 대한 필요성이 존재한다.

[0003] 추가적으로, 인코딩된 데이터를 수신하는 임의의 프로세스가 원래의 인코더에 다시 요청을 발송(send)하지 않고 데이터를 디코딩할 수 있는 것이 바람직할 수도 있다. 예를 들어, 네트워크를 통해 인코딩된 데이터를 수신하는 컴퓨팅 디바이스는 네트워크를 통해 발송 디바이스에 요청들을 발송하지 않고 데이터에 대한 오류 체크들을 수행할 수 있음으로써 개선될 수 있다. 따라서, 종래의 ECC 메모리 모듈 대신에 또는 그에 추가하여 ECC 구현을 구현하도록 구성될 수 있는 시스템에 대한 필요성이 있다.

발명의 내용

해결하려는 과제

과제의 해결 수단

발명의 효과

도면의 간단한 설명

[0004] 실시예들은 첨부된 도면의 도면들에 제한이 아닌 예로서 도시되고, 동일한 참조 번호는 유사한 엘리먼트들을 지

칭한다.

도 1a는 메모리 배열을 도시하는 블록도이다.

도 1b는 ECC 인코더 및/또는 디코더(160)의 구성가능한 로직 엘리먼트들의 예를 도시한다.

도 2는 ECC 인코더 또는 디코더를 프로그래밍하기 위한 접근법을 도시하는 흐름도이다.

도 3a는 ECC 구현을 구성하기 위해 H-행렬을 사용하는 예를 도시한다.

도 3b는 ECC 구현을 구성하기 위해 G-행렬을 사용하는 예를 도시한다.

도 3c는 특정 데이터 값을 인코딩하는 예를 도시한다.

도 3d는 저장된 코드 워드를 디코딩하는 예를 도시한다.

도 3e는 다중 비트 오류를 식별하기 위해 저장된 코드 워드를 디코딩하는 예를 도시한다.

발명을 실시하기 위한 구체적인 내용

[0005] 다음의 설명에서는, 설명의 목적을 위하여, 실시예의 완전한 이해를 제공하기 위하여, 다수의 특정 세부사항이 제시된다. 그러나, 본 기술분야의 당업자에게는 본 실시예들이 이러한 특정 세부사항들 없이 실시될 수 있다는 것이 명백할 것이다. 다른 경우에, 실시예를 불필요하게 모호하게 하는 것을 피하기 위해 공지의 구조 및 디바이스가 블록도 형태로 도시된다.

[0006] **I.개요**

[0007] **II.아키텍처**

[0008] **III.일반화된 인코더/디코더**

[0009] **IV.동작 개요**

[0010] **V.인코딩**

[0011] **VI.디코딩**

[0012] **I.개관**

[0013] 메모리 모듈은 특정 ECC 구현과 매칭하도록 프로그래밍될 수 있는 하나 이상의 범용 ECC 엔진들을 포함한다. 본 명세서에서 사용되는 바와 같이, 용어 "ECC 구현"은 오류 검출 및 후속 프로세싱, 예를 들어, 오류 검출의 결과를 사용하여 오류 정정을 수행하고 정정될 수 없는 손상된 데이터를 인코딩하는 등의 것을 수행하기 위한 ECC 기능을 지칭한다.

[0014] 범용 ECC 엔진을 제공함으로써, 메모리 모듈은 임의의 특정 ECC 구현에 대해 미리 프로그래밍되지 않고 ECC 구현을 구현하도록 구성될 수 있다. 이는 상이한 ECC 구현들을 지원하는 상이한 호스트 디바이스들과 페어링될 수 있는 PIM 기반 메모리 모듈들에서 특히 유용할 수 있다. 메모리 모듈은 호스트에 의해 사용되는 H-행렬과 같은 ECC 구현을 정의하는 입력을 수신하도록 구성된다. 메모리 모듈이 메모리 모듈에 저장될 데이터를 인코딩하거나 메모리 모듈에 저장된 데이터를 디코딩하라는 요청을 수신하면, 메모리 모듈은 메모리 모듈의 로직 엘리먼트들의 선택적 사용을 통해 ECC 구현을 실행한다.

[0015] **II.아키텍처**

[0016] 도 1a는 메모리 배열(100)을 도시하는 블록도이다. 메모리 배열(100)은 호스트(110) 및 메모리 모듈(120)을 포함한다. 호스트(110) 및 메모리 모듈(120)은 메모리 버스(130)를 통해 통신 가능하게 결합된다. 본 명세서에서 사용되는 바와 같이, 용어 "버스"는 와이어들, 전도체들, 및/또는 무선 통신 링크들과 같은 임의의 유형의 유선 또는 무선 결합을 지칭한다. 또한, 실시예들이 버스들의 맥락에서 본 명세서에서 설명되지만, 실시예들은 그 자체로 버스들로 제한되지 않으며, 직렬 링크들 및 용량성/유도성 결합과 같은 다른 형태들의 메모리 연결들에 적용가능하다. 실시예들은 도면들에 도시되고 설명의 목적들을 위해 단일 메모리 모듈(120)의 맥락에서 본 명세서에 설명되지만, 실시예들은 도 1의 M개의 메모리 모듈들과 같은 임의의 수의 메모리 모듈들을 갖는 메모리 배열들에 적용가능하다.

[0017] 호스트는 메모리 모듈(120)에 데이터를 기록하거나 메모리 모듈(120)로부터 데이터를 수신한다. 호스트는 오류

정정이 수행되지 않고 메모리로부터 직접 또는 오류 정정 코드(ECC) 인코더 및/또는 디코더(160)로부터 데이터를 획득하도록 구성될 수 있다. 메모리 배열(100)은 도면들에 도시되지 않고 설명을 위해 본 명세서에 설명된 다른 엘리먼트들을 포함할 수 있다.

[0018] 1 예를 들어 DRAM 메모리 모듈일 수 있는 메모리 모듈(120)은 N개의 PIM 유닛들(PIM 유닛 1 - PIM 유닛 N)을 갖는 메모리 배열을 포함한다. 각각의 PIM 유닛은 메모리(150), ECC 인코더 및/또는 디코더(160) 및 프로세서(170)를 포함한다. 메모리 모듈들은 도면들에 도시되지 않고 설명의 목적들을 위해 본 명세서에 설명되는 버퍼들 및 디코더들과 같은 다른 엘리먼트들을 포함할 수 있다. PIM 유닛들은 산술 연산을 수행하기 위한 프로세싱 로직 및 로컬 레지스터와 같은 저장소를 포함한다. 실시예들이 도면들에 도시되고 PIM 기반 메모리 모듈들의 맥락에서 본 명세서에 설명되지만, 실시예들은 이러한 맥락으로 제한되지 않으며 비-PIM 기반 메모리 모듈들에 적용가능하다.

[0019] 메모리 모듈(120)은 복수의 로직 엘리먼트들을 포함할 수 있다. 로직 엘리먼트들은 로직 동작들을 수행하도록 구성된 하드웨어 컴포넌트들을 포함한다. 예시적인 로직 엘리먼트들은 AND 게이트들, OR 게이트들, NOR 게이트들, NAND 게이트들, XOR 게이트들, 및 XNOR 게이트들을 포함한다. 로직 엘리먼트들은 전자 스위치들로서 작용하는 다이오드들 또는 트랜지스터들을 사용함으로써 하드웨어로 구현될 수 있다.

[0020] 도 1b는 ECC 인코더 및/또는 디코더(160)의 구성가능한 로직 엘리먼트들의 예를 도시한다. 본 명세서에서 사용되는 "구성가능한 로직 엘리먼트들" 또는 "구성가능한 로직 블록"은 특정 게이트들에서 특정 비트들을 설정함으로써 상이한 ECC 구현들로 구성될 수 있는 하드웨어 로직의 고정된 세트를 지칭한다. 도 1b의 예는 구성가능한 ECC 인코더 및/또는 디코더로서 사용될 수 있는 구성가능한 로직 엘리먼트들의 일 구현이다. 다른 옵션들은 다양한 레벨들에서 멀티플렉서들을 사용하는 방식들을 포함할 수 있다. 도 1b의 구현에서, 복수의 AND 게이트들은 어느 데이터 비트들이 XOR 게이트에 전달될 것인지를 결정한다. 도 1b의 AND 로직 엘리먼트들 각각은 구성 명령어들을 통해 설정되는 비트를 포함한다. 본 명세서에서 사용되는 바와 같이, 데이터 비트를 "마스킹"하는 것은, AND 게이트에 대해, 입력 값이 XOR 게이트로 전달되지 않도록 비트를 "0"으로 설정하는 것에 대응하는 한편, 마스킹되지 않은 데이터 비트들은 "1"로 설정된 비트들에 대응하므로, 입력 값은 AND 게이트를 통해 XOR 게이트로 전달된다. ECC 인코더 및/또는 디코더(160)는 본 명세서에서 추가로 설명되는 바와 같이 패리티 체크 행렬(H-행렬) 및/또는 생성기 행렬(G-행렬)에 따라 마스킹될 수 있다.

[0021] **III.일반화된 인코더/디코더**

[0022] 일 실시예에 따르면, 메모리 모듈(120)은 ECC 구현을 위한 일반화된 인코더 및/또는 디코더로 구성된다. 본 명세서에서 사용되는 바와 같이, 용어 "일반화된 인코더 및/또는 디코더"는 ECC 구현을 위한 데이터를 인코딩하고 및/또는 ECC 구현을 위한 데이터를 디코딩하는 프로그래밍가능 프로세싱 로직을 지칭한다. 일반화된 인코더 및/또는 디코더는, ECC 구현에 특정한 정보로 인코딩되는 한, 복수의 ECC 구현들 중 임의의 것으로 구현될 수 있도록 프로그래밍 가능하다. 인코더 및/또는 디코더를 구현하기 위한 엔진들은 특정 구현에 따라 변할 수 있는 여러 상이한 방식들로 구현될 수 있다. 예들은 CPU들, GPU들, 마이크로제어기들, 필드 프로그래밍가능 게이트 어레이들 (FPGA들), 주문형 집적 회로들 (ASIC들), 및 다른 유형들의 프로세싱 로직을 제한 없이 포함할 수도 있다. ECC 엔진들은 고대역폭 메모리-3(High Bandwidth Memory-3; HBM3) 메모리 모듈에서의 온-다이 ECC 엔진과 같은 메모리 모듈(120) 내부에 있는 임의의 ECC 능력들을 증강시킬 수 있다.

[0023] ECC 인코더 및/또는 디코더(160)는 메모리(150)와 프로세서들(170) 사이에서 구현될 수 있어서, 메모리(150)로부터 액세스되는 데이터는 프로세서들(170)에 의한 사용 전에 디코딩될 수 있고, 프로세서(170)에 의해 생성되는 데이터는 메모리(150)에 저장되기 전에 인코딩될 수 있다. 도 1a에서, 각각의 메모리 모듈은 PIM 유닛들에서 ECC 인코더 및/또는 디코더를 통해 구현된 로직 엘리먼트들에 대한 마스킹 명령어들을 포함한다. 다른 구현들에서, 마스킹 명령어들은 별도로 저장되고 로직 엘리먼트들을 마스킹하는데 사용될 수 있다. 추가적으로 또는 대안적으로, 복수의 PIM 유닛들은 마스킹 명령어들의 단일 세트에 대한 액세스를 공유할 수 있다. 다른 예로서, 다중 계층 메모리 모듈은 계층당 마스킹 명령어의 한 세트로 구성될 수 있다.

[0024] 본 명세서에서 "호스트"로도 지칭되는 호스트 프로세싱 엘리먼트는 특정 ECC 구현을 위해 마스킹할 복수의 로직 엘리먼트들을 식별하는 마스킹 명령어들을 제공함으로써 일반화된 인코더/디코더를 프로그래밍한다. 예를 들어, 호스트는 특정 ECC 구현에서 사용되는 H-행렬에 대응하는 데이터를 제공할 수 있다. 메모리 모듈은 H-행렬에 기초하여 활성화할 로직 엘리먼트들 및 활성화하지 않을 로직 엘리먼트들을 식별하거나 마스킹할 수 있다. 예를 들어, FPGA, EPROM 등은 특정 H-행렬 및 데이터를 인코딩 또는 디코딩하기 위해 H-행렬을 사용하는 방법에 대한 명령어들로 프로그래밍될 수 있다. 다른 예로서, 호스트는 특정 구현을 위해 H-행렬을 정의하는 데이터를 PIM

유닛 또는 마이크로제어기에 의한 실행을 위해 메모리 모듈에 제공할 수 있다. 마스킹 명령어들은 PIM 커맨드들을 프로세싱하는데 사용되는 PIM 유닛들의 프로세싱 로직에 의해, PIM 유닛들 내의 별도의 프로세싱 로직에 의해, 또는 PIM 유닛들 내의 마이크로제어기에 의해 프로세싱된다.

[0025] 본 명세서에 설명된 마스킹 명령어들 및 임의의 추가적인 ECC 구현 명령어들은 도 1에 도시된 바와 같이 메모리 모듈(120)의 메모리 어레이, 즉 뱅크들에 저장될 수 있다. 대안적으로, 명령어들은 PIM 유닛들에, 또는 별도의 디바이스에 저장될 수 있다. 마스킹 명령어들 및 임의의 추가적인 ECC 구현 명령어들은 임의의 유형 또는 형태의 명령어들일 수 있다. 예를 들어, 명령어들은 소스 코드, 예를 들어, RTL(Register Transfer Language) 명령어들, 파싱된 소스 코드, 컴파일된 코드, 예를 들어, 객체 코드, 또는 실행가능한 코드, 예를 들어, 이진 코드의 형태일 수 있다. 명령어들의 다른 예시적인 형태는 Verilog 및 VHDL과 같은 FPGA들과 함께 사용되는 하드웨어 설명 언어이다. 호스트는 저장되고 실행될 PIM 모듈에 제공되는 실행 가능 코드를 생성하기 위해 소스 코드를 컴파일할 수 있다. 대안적으로, 호스트는 소스 코드를 컴파일하여 저장되고 실행되는 실행 가능 코드를 생성하는 마이크로제어기 또는 PIM에 소스 코드를 제공할 수 있다. 따라서, 소스 코드로부터의 실행가능 코드의 생성은 호스트 또는 메모리 모듈에서 수행될 수 있다.

[0026] ECC 인코더 및/또는 디코더는 프로세서로부터 메모리에 기록된 모든 데이터를 인코딩하고 프로세서에 의해 메모리로부터 판독된 모든 데이터를 디코딩하도록 구성될 수 있다. 프로세서에 의해 메모리로부터 액세스되고 메모리에 저장된 데이터를 자동으로 인코딩 및 디코딩함으로써, 메모리 모듈은 메모리에 저장된 데이터가 디코딩을 야기하거나 인코딩된 데이터를 검출하기 위한 특수 커맨드를 요구하지 않고 성공적으로 디코딩될 수 있는 것을 보장함으로써 시스템의 성능을 향상시킨다.

[0027] 일 실시예에 따르면, 일반화된 ECC 인코더/디코더는 PIM 유닛 내의 프로세서와 메모리 사이의 ECC 프로세싱을 수행한다. 예를 들어, ECC 인코더/디코더는 PIM 유닛의 메모리로부터 데이터를 수신하고, 데이터에 대해 오류 체크를 수행하고, 그 결과를 PIM 유닛의 프로세서에 제공하도록 구성된 구성가능한 로직 블록일 수 있다. 반대로, ECC 인코더/디코더는 PIM 유닛의 프로세서로부터 새로운 데이터를 수신하고, 새로운 데이터를 체크 비트들로 인코딩하고, 인코딩된 데이터가 메모리에 저장되게 하도록 구성될 수 있다. 다른 구현에서, ECC 인코더/디코더는 네트워크를 통해 데이터를 수신하고 데이터를 이용하거나 데이터를 저장소로 발송하기 전에 데이터에 대해 오류 정정 및/또는 인코딩을 수행하도록 구성될 수 있다.

[0028] 실시예에 따르면, 일반화된 인코더/디코더는 상이한 ECC 구현들로 구현되도록 구성된다. 상이한 ECC 구현들은 호스트 머신에 의해 사용되는 독점적 인코더들/디코더들을 포함할 수 있다. 일반화된 인코더/디코더는 ECC 구현에 의해 사용되는 특정 H-행렬로 ECC를 프로그래밍함으로써 특정 ECC 구현에 대해 구성될 수 있다. 이는 호스트 머신에 의해 사용되는 ECC의 정확한 구현이 다를 때 일반화된 인코더/디코더가 상이한 시스템들에 적응될 수 있게 한다.

[0029] **IV.동작 개요**

[0030] 본 명세서에 설명된 일반화된 인코더/디코더는 ECC 구현을 위해 활성화할 로직 엘리먼트들을 식별하기 위해 구성 정보를 사용함으로써 하드웨어를 통해 구현될 수 있다. 다른 실시예들은 디지털 프로그래밍된 계산들을 통해 소프트웨어에서 유사한 로직 프로세스들을 구현할 수 있다.

[0031] 도 2는 ECC 인코더 및/또는 디코더를 구현하기 위한 접근법을 도시하는 흐름도이다. 단계들(202-208)은 저장 전에 데이터를 인코딩하기 위해 구성가능한 로직 엘리먼트들을 사용하기 위한 방법을 포함한다. 단계(202)에서, 구성가능한 로직 블록은 ECC 구현에 따라 구성된다. 예를 들어, 구성가능한 로직 블록은 복수의 XOR 게이트를 포함하는 XOR 트리를 포함할 수 있다. 구성가능한 로직 블록은 구성 명령어들의 세트에 따라 XOR 트리 내의 XOR 게이트들로 이어지는 복수의 AND 게이트들 내의 데이터 비트들을 마스킹하도록 구성될 수 있다. 명령어들은 마스킹할 AND 게이트들의 비트들을 직접 표시할 수 있다. 추가적으로 또는 대안적으로, 명령어들은 메모리 모듈이 AND 게이트들의 어느 비트들을 마스킹할지를 결정하기 위해 사용하는 이진 또는 심볼 행렬을 포함할 수 있다. ECC 구현과 매칭하기 위해 특정 비트들을 마스킹하기 위한 구현들이 본 명세서에서 더 설명된다.

[0032] 도 3a는 ECC 구현을 구성하기 위해 H-행렬을 사용하는 예를 도시한다. H-행렬(302)은 데이터로부터 코드 워드를 생성하는 선형 함수를 나타내는 호스트 디바이스로부터 수신된 H-행렬을 포함한다. 도 3a에 도시된 H-행렬은 $GF(2^3)$ 표현, α, 및 이진 표현을 포함한다. 본 명세서에 설명된 방법들은 이진 행렬들에 대해 설명되지만, 동일한 방법들이 임의의 H-행렬에 적용될 수 있다. 열 0-17은 데이터를 나타내는 반면 열 18-23은 체크 비트를 나타낸다. H-행렬을 사용하여, 시스템은 구성가능한 로직 엘리먼트들의 AND 게이트들에서 마스킹되지 않은 데이터

비트들 및 마스킹된 데이터 비트들을 포함하는 H-행렬 XOR 게이트 입력들(304)을 생성한다. H-행렬 XOR 게이트 입력들(304)의 형태는 명확한 예를 제공하기 위해 H-행렬과 동등한 형태로 도시된다. 실시예들에서, H-행렬 XOR 게이트 입력들(304)의 형태는 XOR 트리를 포함하며, 여기서 XOR 게이트들로 이어지는 AND 게이트들 내의 데이터 비트들은 구성 명령어들에 따라 마스킹된다. 일단 데이터 비트들의 마스킹을 통해 구성되면, H-행렬 XOR 게이트 입력들(304)은 인코딩된 데이터에 대해 오류 정정을 수행하는 데 이용될 수 있다.

[0033] 도 3b는 ECC 구현을 구성하기 위해 G-행렬을 사용하는 예를 도시한다. G-행렬(303)은 공지된 기술을 사용하여 H-행렬로부터 유도될 수 있고 및/또는 호스트 디바이스에 의해 제공될 수 있는 G-행렬을 포함한다. 도 3b에 도시된 G-행렬은 $GF(2^3)$ 표현, α, 및 이진 표현을 포함한다. 본 명세서에 설명된 방법들은 이진 행렬들에 대해 설명되지만, 동일한 방법들이 임의의 G-행렬에 적용될 수 있다. G-행렬을 사용하여, 시스템은 구성가능한 로직 엘리먼트들의 AND 게이트들에서 마스킹되지 않은 데이터 비트들 및 마스킹된 데이터 비트들을 포함하는 G-행렬 XOR 게이트 입력들(305)을 생성한다. G-행렬 XOR 게이트 입력들(305)의 형태는 명확한 예를 제공하기 위해 G-행렬과 동등한 형태로 도시된다. 실시예들에서, G-행렬 XOR 게이트 입력들(305)의 형태는 XOR 트리를 포함하며, 여기서 XOR 게이트들로 이어지는 AND 게이트들 내의 데이터 비트들은 구성 명령어들에 따라 마스킹된다. 일단 구성되면, G-행렬 XOR 게이트 입력들(305)은 인코딩될 데이터에 대한 체크 비트들을 생성하기 위해 이용될 수 있다.

[0034] 단계(204)에서, 구성가능한 블록은 인코딩될 데이터를 수신한다. 예를 들어, PIM 모듈은 호스트 프로세서에 대한 연결을 요구하지 않고 메모리에 대한 판독 및 기록을 허용하는 프로세서를 포함한다. 프로세서가 저장된 데이터로 계산을 완료할 때, 프로세서는 결과를 메모리에 저장한다. 결과를 저장하기 전에, 메모리 모듈은 ECC 인코더를 통해 데이터를 전달하여 ECC 인코더가 본 명세서에 설명된 방법들을 사용하여 결과를 인코딩하게 할 수 있다.

[0035] 실시예들이 데이터의 PIM 저장 및 검색에 대한 요청으로 설명되지만, 본 명세서에 설명된 시스템들 및 방법들은 별도의 디바이스에 의해 인코딩된 데이터를 저장하거나 수신하는 임의의 시스템에 적용될 수 있다. 예를 들어, 컴퓨팅 디바이스가 네트워크를 통해 인코딩된 정보를 수신했다면, 컴퓨팅 디바이스는 인코딩 디바이스에 발송될 추가적인 요청들을 요구하지 않고 검색 동안 데이터에 대해 오류 정정을 수행하기 위해 본 명세서에 설명된 방법들을 사용할 수 있다.

[0036] 일 실시예에서, 컴퓨팅 디바이스는 다수의 오류 정정 코드 구현으로 구성된다. 컴퓨팅 디바이스가 네트워크를 통해 인코딩된 데이터를 수신할 때, 컴퓨팅 디바이스는, 예컨대 데이터의 발송자에 기초하여 또는 데이터와 함께 발송된 추가 정보에 기초하여, 데이터에 적용되는 오류 정정 코드 구현을 식별할 수 있다. 컴퓨팅 디바이스는 식별된 오류 정정 코드 구현을 사용하여 데이터를 디코딩할 수 있다. 추가적으로, 데이터가 별도의 디바이스로 다시 발송되어야 하는 경우, 컴퓨팅 디바이스는 별도의 디바이스에 대응하는 오류 정정 코드 구현을 식별하고, 오류 정정 코드 구현을 사용하여 데이터를 인코딩할 수 있다.

[0037] 단계(206)에서, 데이터는 체크 비트들을 생성하기 위해 구성가능한 로직 블록을 통과한다. 예를 들어, 구성가능한 로직 엘리먼트들은 G-행렬에 따라 마스킹되고 마스킹되지 않은 비트들로 구성될 수 있다. 데이터는 구성가능한 로직 엘리먼트들을 통과할 수 있고, 마스킹되지 않은 AND 게이트들에 대응하는 데이터는 XOR 게이트들로 진행하여 체크 비트들을 생성한다.

[0038] 단계(208)에서, 데이터는 체크 비트들을 데이터에 첨부함으로써 인코딩되고 메모리에 저장된다. 구성가능한 로직 엘리먼트들을 사용하는 인코딩 스킴(scheme)들이 본 명세서에서 더 설명된다.

[0039] 단계들(210-216)은 사용 전에 데이터에 대한 오류 정정을 수행하기 위해 구성가능한 로직 엘리먼트들을 사용하기 위한 방법을 포함한다. 단계(210)에서, 구성가능한 로직 블록은 ECC 구현에 따라 구성된다. 예를 들어, 구성가능한 로직 블록은 복수의 XOR 게이트를 포함하는 XOR 트리를 포함할 수 있다. 구성가능한 로직 블록은 구성 명령어들의 세트에 따라 XOR 트리 내의 XOR 게이트들로 이어지는 복수의 AND 게이트들 내의 데이터 비트들을 마스킹하도록 구성될 수 있다. 명령어들은 마스킹할 AND 게이트들의 비트들을 직접 표시할 수 있다. 추가적으로 또는 대안적으로, 명령어들은 메모리 모듈이 AND 게이트들의 어느 비트들을 마스킹할지를 결정하기 위해 사용하는 이진 또는 심볼 행렬을 포함할 수 있다. ECC 구현과 매칭하기 위해 특정 비트들을 마스킹하기 위한 구현들이 본 명세서에서 더 설명된다.

[0040] 단계(212)에서, 구성가능한 로직 블록은 디코딩될 데이터를 수신한다. 예를 들어, PIM 모듈은 호스트 프로세서에 대한 연결을 요구하지 않고 메모리에 대한 판독 및 기록을 허용하는 프로세서를 포함한다. PIM의 메모리에

저장된 데이터가 호스트 프로세서에 의해 생성되었고 따라서 특정 ECC 구현으로 인코딩되는 경우, PIM의 프로세서는 데이터에 대해 하나 이상의 동작들을 수행하기 위해 메모리로부터 데이터를 요청할 수 있다. 실제적인 예로서, PIM은 호스트에 의해 기록된 데이터를 사용하여 시뮬레이션을 수행하고 그런 다음 시뮬레이션 결과를 메모리에 저장하도록 구성될 수 있다. 시뮬레이션을 수행하기 위해 데이터를 프로세서에 발송하기 전에, 메모리 모듈은 ECC 디코더가 본 명세서에 설명된 방법들을 사용하여 데이터를 디코딩하게 하기 위해 데이터를 ECC 디코더를 통해 전달할 수 있다.

[0041] 단계(214)에서, 데이터는 신드롬을 생성하기 위해 구성가능한 로직 블록을 통과한다. 예를 들어, 구성가능한 로직 엘리먼트들은 H-행렬에 따라 마스킹되고 마스킹되지 않은 비트들로 구성될 수 있다. 데이터는 구성가능한 로직 엘리먼트들을 통과할 수 있고, 마스킹되지 않은 AND 게이트들에 대응하는 데이터는 XOR 게이트들로 진행하여 신드롬을 생성한다.

[0042] 단계(216)에서, 신드롬을 사용하여 오류 정정이 수행되고 정정된 데이터가 프로세서에 제공된다. 예를 들어, 신드롬은 오류가 정정될 수 있도록 오류 위치 및 크기를 식별하는 데 사용될 수 있다. 구성가능한 로직 엘리먼트들을 사용하는 디코딩 스킴(scheme)들이 본 명세서에서 더 설명된다.

[0043] **V.인코딩**

[0044] 일 실시예에서, ECC 구현은 메모리에 저장될 새로운 데이터를 인코딩하는 것을 포함한다. 인코딩은 메모리에 데이터를 저장하기 위한 프로세서로부터의 요청에 응답하여 및/또는 다른 인코딩된 데이터와 함께 저장될 데이터를 네트워크를 통해 수신하는 것에 응답하여 수행될 수 있다.

[0045] 일 실시예에서, 데이터를 인코딩하는 것은 특정 ECC 구현에 따라 구성된 구성가능한 로직 엘리먼트들을 사용하여 특정 데이터 값에 대한 복수의 체크 비트들을 결정하는 것을 포함한다. 체크 비트들은 특정 데이터 값에 대한 코드 워드를 생성하기 위해 특정 데이터 값의 비트들에 첨부될 수 있다. 그런 다음, 코드 워드는 메모리에 저장될 수 있다. 일 실시예에서, ECC 인코더는 구성가능한 로직 엘리먼트들을 통해 데이터 값에 대응하는 비트들을 발송함으로써 복수의 체크 비트 값들을 결정하며, 이에 의해 행렬 곱셈을 생성한다. 예를 들어, ECC 인코더는 H-행렬에 따라 비트가 설정되는 복수의 AND 게이트를 포함할 수 있다. 데이터 값으로부터 데이터 비트들을 수신하는 AND 게이트들만이 XOR 게이트에 데이터 비트를 발송할 것이기 때문에, 데이터 값의 비트에 대응하지 않는 열들은 XOR 게이트에 대응하는 행에 대한 값을 계산하기 위해 XOR 게이트에 데이터 비트를 제공(contribute)하지 않을 것이다. 따라서, 특정 데이터 값이 0번째 비트 및 2번째 비트에서 "1"을 포함하면, ECC 인코더는 0번째 열 및 2번째 열의 AND 게이트들로부터 XOR 게이트로 비트들을 발송할 수 있다.

[0046] 도 3c는 특정 데이터 값을 인코딩하는 예를 도시한다. 도 3c의 예에서, 시스템은 데이터 값에 대한 체크 비트들을 계산하기 위해 G-행렬을 사용한다. 다른 실시예들에서, 시스템은 H-행렬을 사용하고, 데이터 값이 로직 엘리먼트들을 통해 곱해질 때 계산된 신드롬이 0이 되게 할 체크 비트 열들에 대한 값들을 계산할 수 있다.

[0047] 도 3c에서, 데이터 값(306)은 인코딩될 데이터 값의 이진 표현을 포함한다. 데이터 값(306)은 로직 엘리먼트들에서 인코딩된 G-행렬과 곱해진다. G-행렬 XOR 게이트 입력들(305)의 각각의 원은 마스킹되지 않은 비트를 갖는 AND 게이트에 대응한다. 따라서, 원을 갖지 않는 게이트를 통과하는 임의의 비트들은 XOR 게이트들로 전달되지 않을 것이다. 데이터 값(306)이 G-행렬 XOR 게이트 입력들(305)을 통과할 때, 데이터 값(306)의 각각의 데이터 비트(1)는 대응하는 열 내의 AND 게이트들을 통과하는 동안, 그에 의해 인코딩 XOR 게이트 입력들(308)을 생성한다. 인코딩 XOR 게이트 입력들(304)은 데이터 비트가 통과하는 각각의 게이트를 포함한다. 데이터 비트들이 1번째 및 3번째 열들에서 "1"로만 설정되기 때문에, G-행렬 XOR 게이트 입력들(305)의 1번째 및 3번째 열들만이 인코딩 XOR 게이트 입력들(308)에 도시된다.

[0048] 인코딩 XOR 게이트 입력들(308)의 나머지 비트들이 XOR 게이트들을 통과한 후에, 이들은 체크 비트들(310)을 생성한다. 예를 들어, 1번째 행은 AND 게이트를 통과한 2개의 비트들을 포함하고, 이에 의해 이들이 XOR 게이트를 통과할 때 0을 생성한다. 유사하게, 행 2 및 행 5는 2개의 비트들을 포함하고, 이에 의해 또한 이들이 그들 각각의 XOR 게이트들을 통과할 때 0을 생성한다. 행 3 및 행 4는 각각 XOR 게이트를 통과하는 단일 비트를 가져서, 1을 생성하고, 행 6은 XOR 게이트를 통과하는 비트들을 갖지 않는다. 따라서, 최종 체크 비트들(310)은 (0 0 0 1 1 0 0)을 포함한다. 체크 비트들(310)은 그런 다음 인코딩된 코드 워드(312)를 생성하기 위해 데이터 값(306)에 첨부된다.

[0049] 위의 설명은 체크 비트 값들을 생성하기 위해 사용되는 행렬 곱셈의 하드웨어 구현을 제공한다. 행렬 곱셈은 다음과 같이 계산될 수 있다:

[0050] $$\sum_{j=1}^{K} G_{ij}D_j = C_i \ for\ i = 1 \ldots R$$

[0051] 여기서, G_{ij}는 i번째 행 및 j번째 열에 대한 G-행렬의 값이고, D는 j번째 열에 대한 데이터 값이고, C_i는 i번째 행에 대한 체크 비트 값이고, K는 열의 총 개수이고, R은 G-행렬의 행의 개수이다.위의 행렬 곱셈은 심볼 공간 또는 이진 공간에서 수행되어 동일한 결과를 얻을 수 있다.

[0052] VI. 디코딩

[0053] 일 실시예에서, ECC 구현은 ECC 디코더에서 수신된 데이터를 디코딩하는 것을 포함한다. 디코딩은 메모리 모듈에 저장된 저장된 저장된 데이터에 액세스하기 위한 프로세서로부터의 요청에 응답하여 및/또는 네트워크를 통해 인코딩된 데이터를 수신하는 것에 응답하여 수행될 수 있다. 데이터를 디코딩하는 것은 저장된 코드 워드를 사용하여 코드 워드들의 하나 이상의 비트들에서의 오류를 식별하고 오류를 수정(fix)하기 위해 코드 워드의 비트들을 조정하는 것을 포함할 수 있다.

[0054] 일 실시예에서, ECC 디코더는 먼저 특정 ECC 구현에 따라 구성된 구성가능한 로직 엘리먼트들을 사용하여 신드롬 열을 계산한다. 그런 다음, 시스템은 신드롬과 매칭하는 열을 식별하기 위해 신드롬 열을 구성가능한 로직 엘리먼트에 대응하는 각각의 열과 비교한다. 시스템은 신드롬과 매칭되는 열이 오류를 포함한다고 결정한다. 이진 구현에서, 오류의 값은 항상 1이므로, 열이 식별되면, 비트는 수정될 수 있다.

[0055] 도 3d는 저장된 코드 워드를 디코딩하는 예를 도시한다. 도 3d에서, 코드 워드(314)는 오류를 포함하는 ECC 구현을 사용하여 인코딩된 데이터를 포함한다. 인코딩과 유사하게, 코드 워드(314)는 로직 엘리먼트들에서 인코딩된 H-행렬과 곱해진다. H-행렬 XOR 게이트 입력들(304)의 각각의 원은 마스킹되지 않은 비트를 갖는 AND 게이트에 대응한다. 코드 워드(314)가 인코딩 단계에서 생성된 체크 비트들을 포함하기 때문에, 체크 비트들은, 체크 비트들에 대응하는 열들을 포함하는 H-행렬을 사용하여 구성된 구성가능한 로직 엘리먼트들을 통해 추가적으로 공급된다. 따라서, 디코딩 XOR 게이트 입력들(316)은 결과적인 신드롬을 계산하기 위해 비트들이 통과하는 5개의 열들을 포함한다. 메모리 모듈은 신드롬(318)을 H-행렬 XOR 게이트 입력들(304)의 열들과 비교한다. 신드롬(318)이 2번째, 4번째 및 5번째 행들에서 1을 포함하기 때문에, 메모리 모듈은 2번째, 4번째 및 5번째 행들에서 열을 검색한다. 신드롬(318)은 H-행렬 XOR 게이트 입력들(304)의 5번째 열과 매칭한다. 따라서, 코드 워드(314)의 5번째 비트는 오류를 포함한다. 메모리 모듈은 비트를 오류 값만큼 변경함으로써 코드 워드를 정정할 수 있다. 따라서, 5번째 열의 1은 0으로 치환된다.

[0056] 위의 설명은 신드롬을 계산하기 위해 사용되는 행렬 곱셈의 하드웨어 구현을 제공한다. 행렬 곱셈은 다음과 같이 계산될 수 있다:

[0057] $$\sum_{j=1}^{KN} H_{ij}D_j = S_i \ for\ i = 1 \ldots$$

[0058] 여기서, H_{ij}는 i번째 행 및 j번째 열에 대한 H-행렬의 값이고, D는 코드 워드의 j번째 열에 대한 데이터 값이고, Si는 i번째 행에 대한 신드롬 값이고, N은 열의 총 개수이고, R은 H-행렬의 행의 개수이다.위의 행렬 곱셈은 심볼 공간 또는 이진 공간에서 수행되어 동일한 결과를 얻을 수 있다.

[0059] 도 3d의 실시예는 단일 비트 오류를 식별하고 수정하는 방법을 나타낸다. 따라서, 신드롬을 H-행렬의 열에 매칭시키는 것은 동일한 위치에서 신드롬과 동일한 값을 갖는 H-행렬의 열을 식별함으로써 수행될 수 있다. 계산된 신드롬이 단지 0 값들의 열이면, 메모리 모듈은 코드 워드에 오류가 존재하지 않는 것으로 결정할 수 있다. 계산된 신드롬이 H-행렬 내의 열들 중 임의의 열과 매칭되지 않는 열이면, 메모리 모듈은 코드 워드가 다중 비트 또는 다중 심볼 오류를 포함한다고 결정할 수 있다.

[0060] 다중 비트 오류들의 경우, 시스템은 심볼 표기법을 이용하여 동일한 심볼 내의 단일 비트 오류들 또는 다중 비트 오류들을 식별할 수 있다. 본 명세서에서 사용되는 바와 같이, 심볼은 3-비트와 같은 복수의 비트들의 세트를 지칭한다.예를 들어, 도 3a에서의 H-행렬의 제 1 표현은 1들을 포함하는 1번째 행 및 α의 거듭제곱들을 포함하는 2번째 행으로 표기된 심볼들의 2개의 행들을 포함하고, 그 각각의 값은 3-비트 심볼을 포함한다.심볼들에 대응하는 3×3 행렬들은 2개의 3-비트 심볼들의 곱셈의 표현을 포함할 수 있으며, 그 중 하나는 행렬로 인코딩되는 상수를 포함한다.예를 들어, 첫 번째 3개의 행들 및 첫 번째 3개의 열들은 D0와 1의 제1 곱셈에 대응하

는 반면, 두 번째 3개의 행들 및 첫 번째 3개의 열들은 D0와 α1의 제1 곱셈에 대응하고, 여기서 D0은 제1 데이터 심볼(3 비트들)이다.

[0061] 일 실시예에서, 디코딩은 H-행렬을 사용하여 수행되며, 여기서, 심볼 공간에서, 1번째 행에 대한 각각의 열은 1의 심볼 값을 포함한다. 예를 들어, 도 3a의 H-행렬에서, 첫 번째 3개의 행들에 대한 3개의 열들의 각각의 반복 세트는 대각선에 1들 및 다른 곳에서는 0들을 갖는 대각선 행렬을 포함한다. 각 열에 대해 행의 값이 1이면 신드롬은 오류를 포함한다. 따라서, 도 3d의 신드롬(318)의 첫 번째 3개의 값들, S0 = 010은 오류를 포함할 것이다. 오류는 신드롬이 심볼과 매칭할 때를 결정하기 위해 각각의 심볼 열에 적용될 수 있다. 각각의 심볼 열에 오류를 적용하는 것은 H-행렬의 심볼에 대응하는 AND 게이트들의 세트를 통해 오류를 전달하는 것 및 XOR 게이트를 통해 결과들을 계산하는 것을 포함할 수 있다.

[0062] 실제 예로서 도 3d를 사용하여, 010의 오류가 H-행렬 XOR 게이트 입력들(304)의 3개의 열들의 제1 세트와 곱해질 때, 2번째 열만이 AND 게이트들을 통과하는 데이터 비트를 수신한다. 2번째 열은 2번째 및 6번째 행에 설정 비트를 포함하기 때문에, XOR 게이트들은 (010001)의 값을 계산한다. 이것이 (010110)의 신드롬과 매칭하지 않기 때문에, 시스템은 다음 열들의 세트를 체크할 수 있다. 010의 오류가 H-행렬 XOR 게이트 입력들(304)의 열들의 제2 세트와 곱해질 때, 5번째 열만이 AND 게이트들을 통과하는 데이터 비트를 수신한다. 5번째 열이 2번째, 4번째, 및 5번째 행들에서 세트 비트를 포함하기 때문에, XOR 게이트들은 신드롬과 매칭하는 (010110)의 값을 계산한다. 따라서, 오류는 5번째 열에 있다.

[0063] 도 3e는 다중 비트 오류를 식별하기 위해 저장된 데이터를 디코딩하는 예를 도시한다. 오류가 110을 포함하면, 메모리 모듈은 첫번째 두 개의 열들을 유지하고 이를 오류와 비교했을 것이다. 예를 들어, 4번째 및 5번째 열들에 오류를 포함하도록 코드 워드를 변경하면(따라서 코드 워드의 첫 번째 6개의 값들은 101110임), 계산된 신드롬은 (110111)로 구성된다. 110의 오류가 제1 심볼, 즉, H-행렬 XOR 게이트 입력들(304)의 첫 번째 3개의 열들에 적용되면, 1번째 및 2번째 열들은 남아, (110011)의 값을 생성한다. 이 값이 신드롬과 매칭하지 않기 때문에, 메모리 모듈은 다음 심볼로 이동한다. 110을 제2 심볼, 즉 H-행렬 XOR 게이트 입력들(304)의 두 번째 3개의 열들에 적용하면, XOR 게이트들을 통해 조합될 때, 신드롬과 매칭하는 (110111)의 값을 생성하는 4번째 및 5번째 열들을 남긴다. 따라서, 시스템은 제2 심볼에 오류가 존재한다고 결정할 것이고, 제2 심볼에 오류 값(110)을 적용하는 것은 오류가 4번째 및 5번째 열들에 있다는 것을 표시한다.

[0064] 일반적인 시나리오에서, 시스템은 H-행렬의 n번째 행과 H-행렬의 복수의 다른 행들 사이의 비율을 사용하여 오류 위치를 계산하도록 구성될 수 있다. 이러한 비율들은 다음 방정식을 사용하여 ECC 구현을 위해 미리 계산되고 저장될 수 있다:

[0065] $$I_{ij} = \frac{H_{ij}}{H_{nj}}$$

[0066] 여기서 I_{ij}는 특정 행 및 특정 열에 대한 비율 값이고, H_{ij}는 특정 행 및 특정 열에서의 H-행렬의 값이고, H_{nj}는 n번째 행에서의 H-행렬의 값이다. 전술한 바와 같이, 모든 행에 대해 H_{nj}이 "1"이면, 비율은 특정 열 및 행에 대한 H-행렬의 값과 동일하다. H-행렬의 복수의 행들로부터 Hnj이 되도록 선택된 행은 행들 중 임의의 것일 수 있다.일 실시예에서, Hnj에 대한 선택된 행이 열에서 "0"을 포함하면, 시스템은 비율의 계산이 무한하지 않도록 그 열에 Hnj대해 상이한 행을 선택한다.

[0067] 오류의 위치를 계산하기 위해, 시스템은 각각의 열(j)을 테스트하여 열(j)의 모든 행에 대해 다음 방정식이 참인지를 결정할 수 있다:

[0068] $$S_i = I_{ij} * S_n$$

[0069] 여기서, S_i는 i번째 행에 대한 신드롬 값이고, S_{in}는 H_{nj}에 대해 사용된 것과 같은 동일한 행에 대한 신드롬 값이다. 따라서, 1번째 열에 대해, H_{nj}이 2번째 행이면, S_n이 n번째 행에 대한 신드롬 값이 될 것이다. 추가적으로, 2번째 열에 대해, H_{nj}이 6번째 행이면, S_n은 6번째 행에 대한 신드롬 값일 것이다. 특정 열에 대해, 각각의 행에 대한 신드롬 값과 비율의 곱이 그 행에 대한 신드롬 값과 동일한 경우, 시스템은 오류가 특정 열에 존재한다고 결정할 수 있다.

[0070] 시스템은 오류 크기를 계산하기 위해 신드롬의 값을 더 사용할 수 있다. 이 단계는 오류의 위치의 계산 전, 후,

또는 병렬로 수행될 수 있다. 오류 값은 다음과 같이 계산될 수 있다:

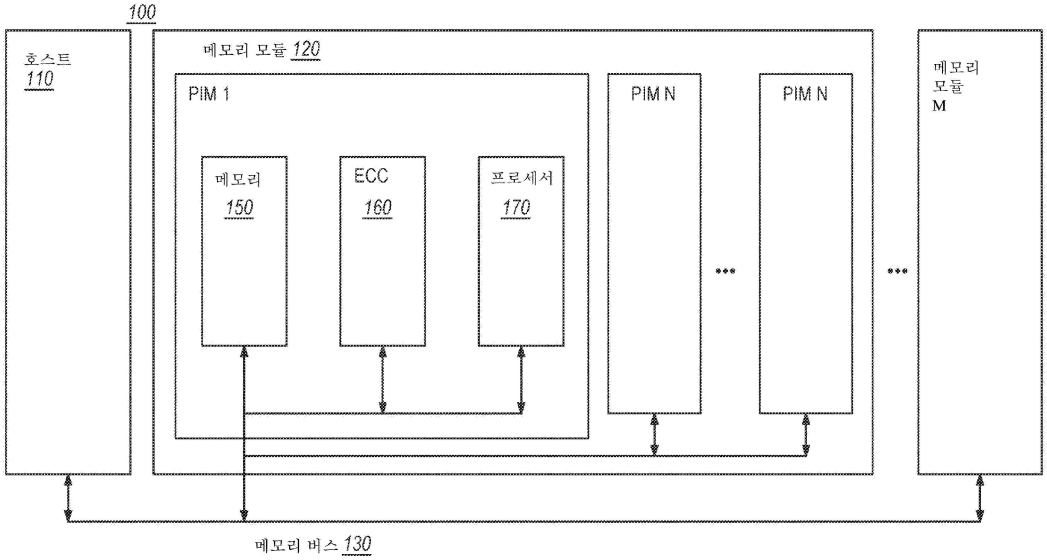
[0071] $$E_j = S_n * H_{nj}^{-1}$$

[0072] 여기서 E_j는 오류이다. S_n 및 H_{nj}^{-1}에 사용된 행은 오류 위치 계산에 사용된 것과 동일한 행일 수 있으며 및/또는 오류 계산 방정식에서 S_n 및 H_{nj}^{-1}에 대해 동일한 행이 사용되는 한 상이한 행일 수 있다. H_{nj}^{-1}의 값은 구현 전에 각 열에 대해 미리 계산될 수 있다. 이 값은 추가적으로 각각의 열에 대해 동일한 것을 사용할 수 있거나 열들 사이에서 행들을 변화시킬 수 있다. 시스템은 각각의 열에 대해 H_{nj}^{-1}에 대해 어떤 행이 사용되고 있는지 표시하는 데이터를 저장할 수 있다.

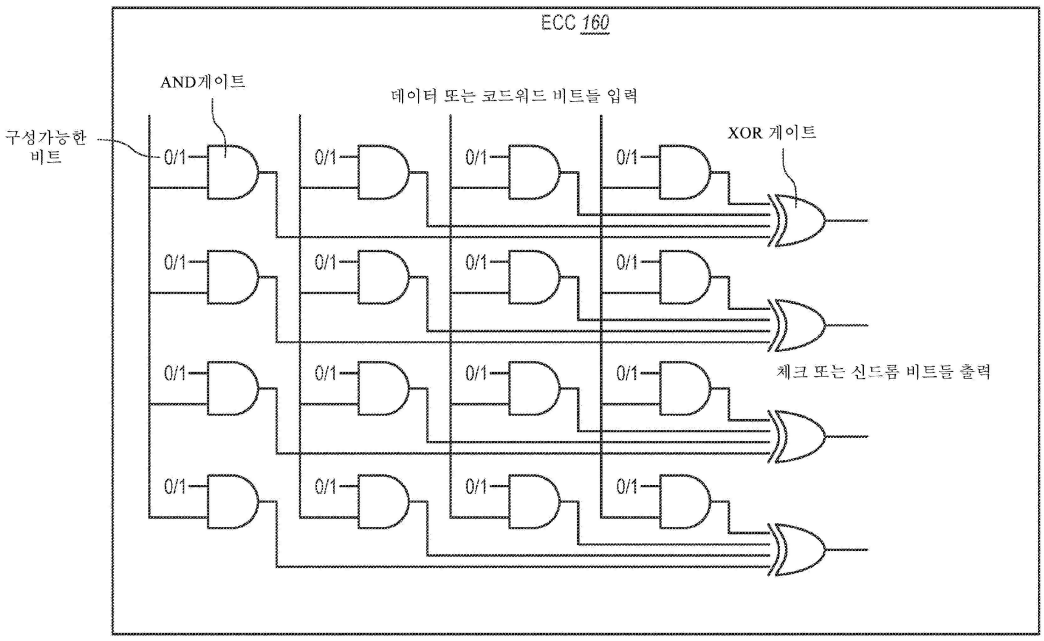
[0073] 오류 값 및 오류 위치가 알려지면, 시스템은 오류 위치에 오류 값을 적용함으로써 오류를 정정할 수 있다. 시스템이 오류의 위치를 식별하기 위해 위의 계산들을 수행하고 계산들 중 어느 것도 모든 행들에 대해 위의 방정식을 만족시키지 않는 경우, 시스템은 코드 워드가 다중 심볼 오류를 포함한다고 결정할 수 있다. 일반적으로, 위에서 설명한 방법들은 초기 ECC 코드가 검출할 수 있는 임의의 오류를 검출할 수 있고 임의의 단일 심볼 오류들을 정정할 수 있다. 본 명세서에 설명된 방법들은 임의의 선형 블록 코드에 대해 수행될 수 있다.

도면

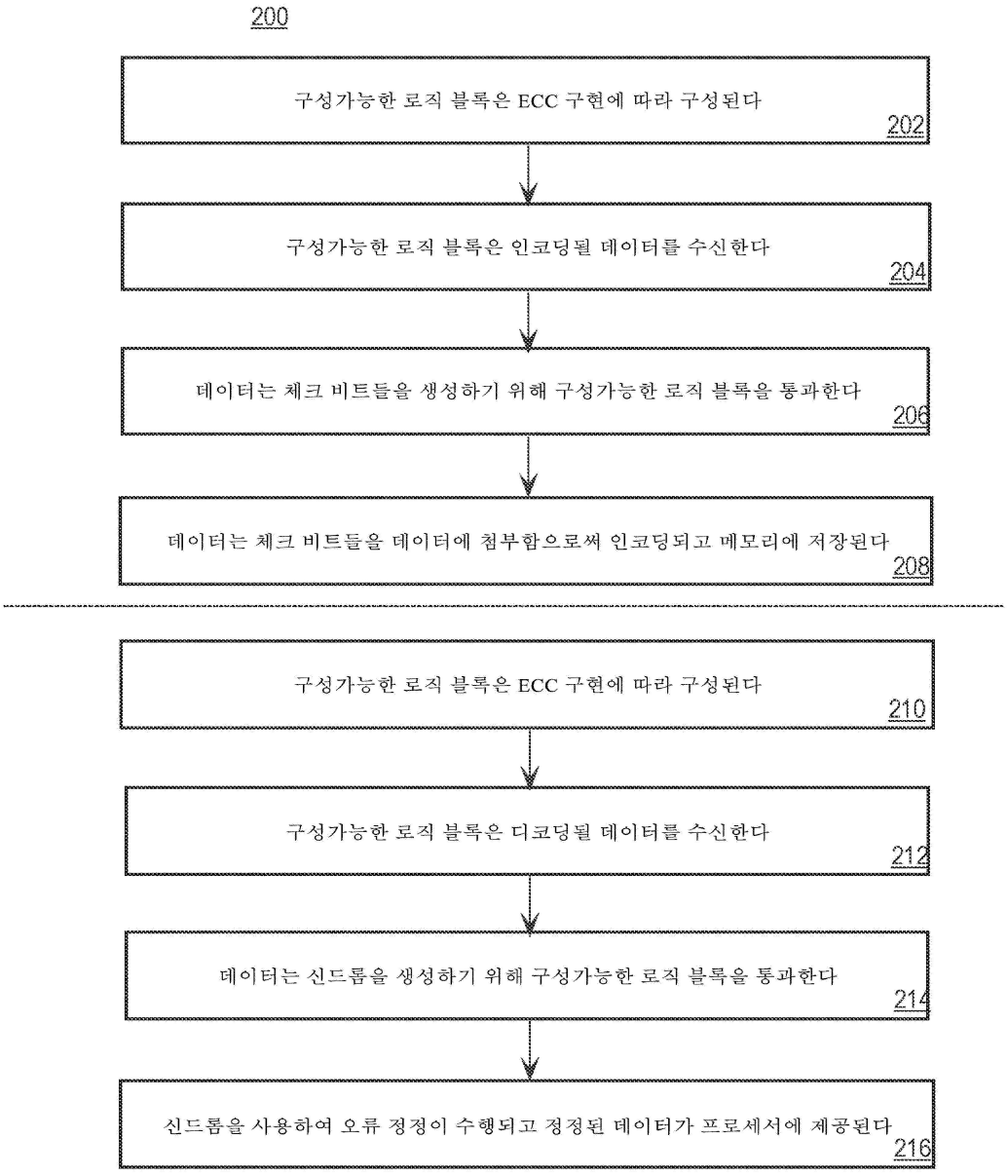
도면1a



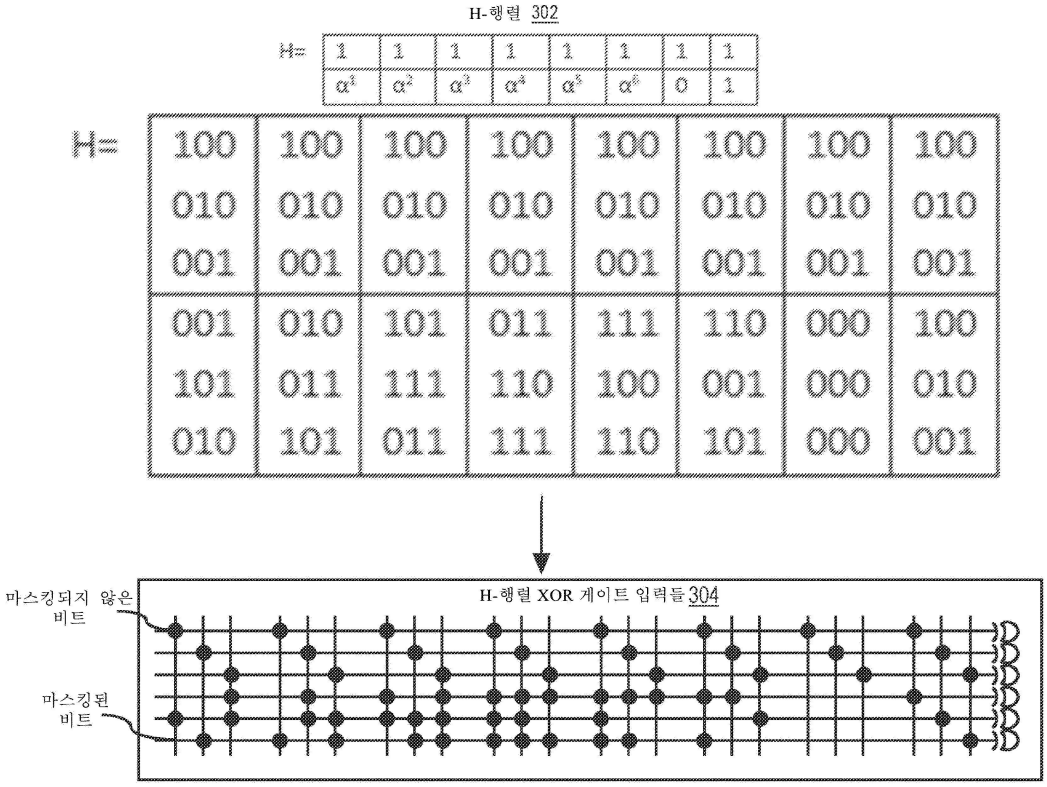
도면1b


ECC 160
AND게이트
데이터 또는 코드워드 비트들 입력
구성가능한 비트
XOR 게이트
0/1
체크 또는 신드롬 비트들 출력

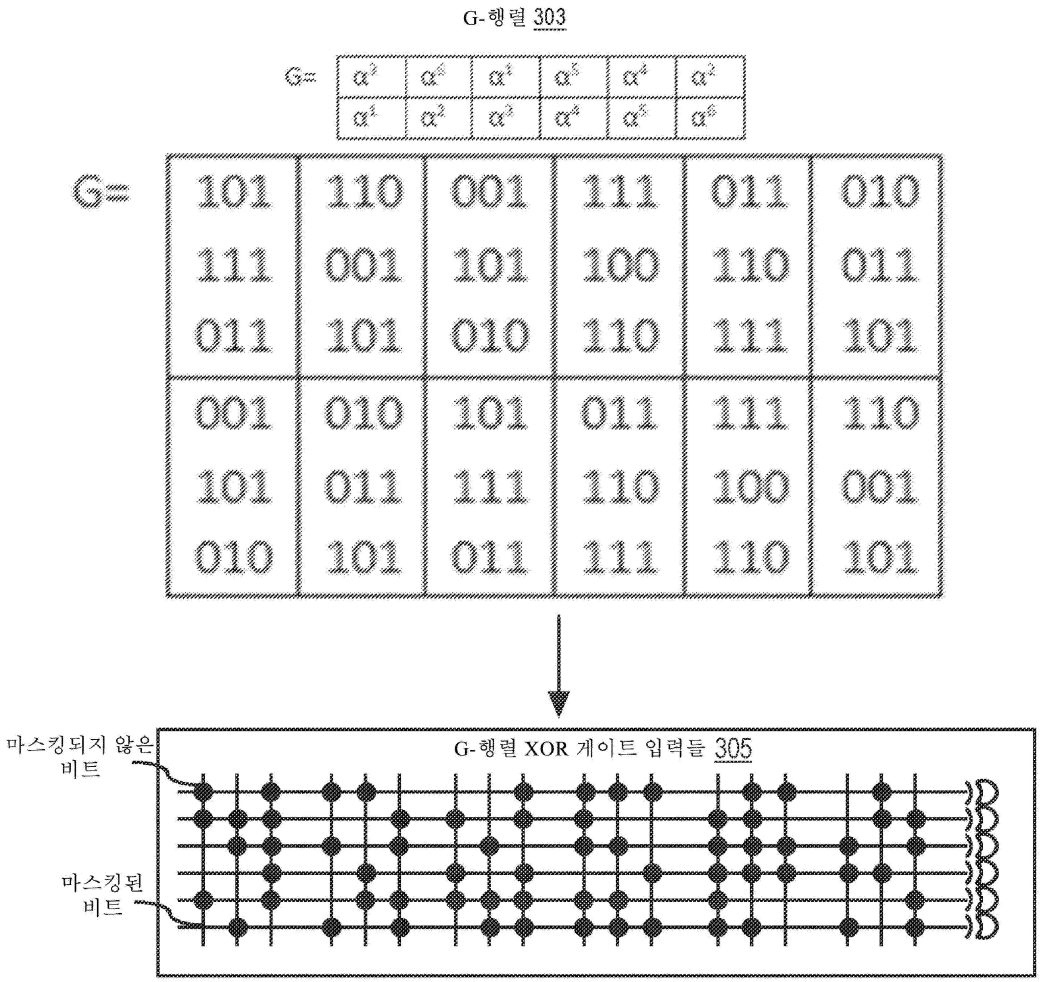
도면2


200
구성가능한 로직 블록은 ECC 구현에 따라 구성된다
202
구성가능한 로직 블록은 인코딩될 데이터를 수신한다
204
데이터는 체크 비트들을 생성하기 위해 구성가능한 로직 블록을 통과한다
206
데이터는 체크 비트들을 데이터에 첨부함으로써 인코딩되고 메모리에 저장된다
208
구성가능한 로직 블록은 ECC 구현에 따라 구성된다
210
구성가능한 로직 블록은 디코딩될 데이터를 수신한다
212
데이터는 신드롬을 생성하기 위해 구성가능한 로직 블록을 통과한다
214
신드롬을 사용하여 오류 정정이 수행되고 정정된 데이터가 프로세서에 제공된다
216

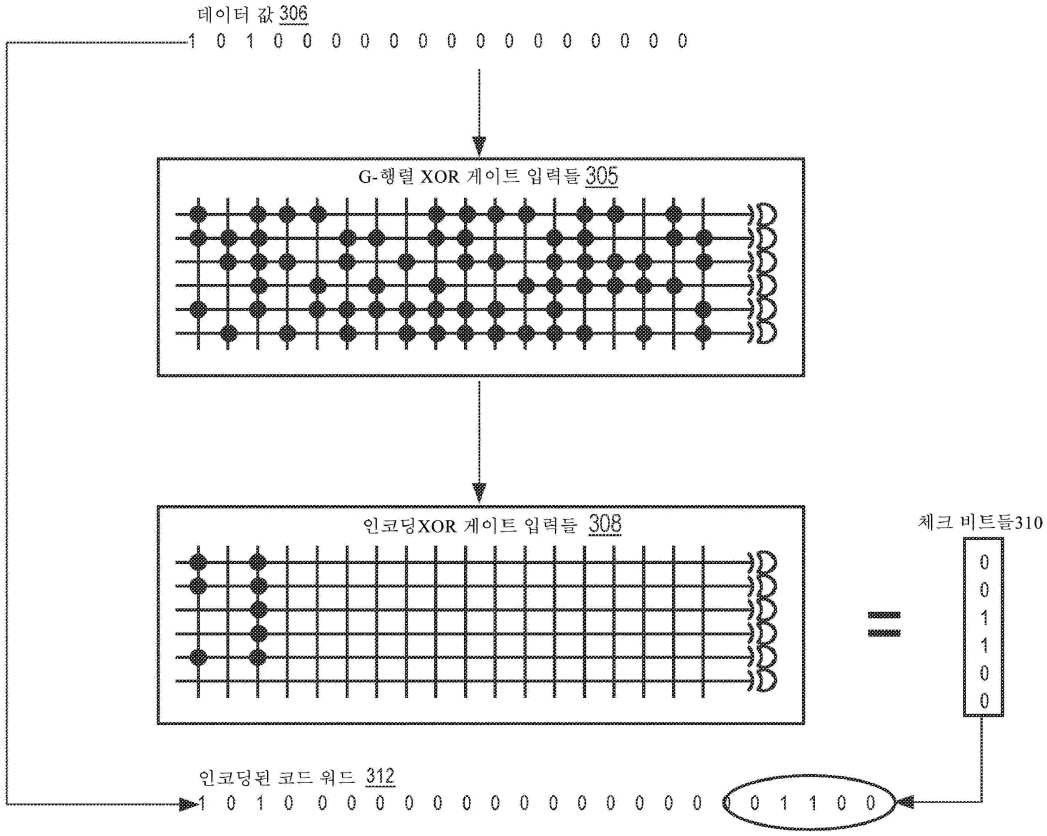
도면3a


H-행렬 302
H=
1 1 1 1 1 1 1 1
α⁵ α⁴ α³ α² α¹ α⁰ 0 1
H=
100 100 100 100 100 100 100 100
010 010 010 010 010 010 010 010
001 001 001 001 001 001 001 001
001 010 101 011 111 110 000 100
101 011 111 110 100 001 000 010
010 101 011 111 110 101 000 001
H-행렬 XOR 게이트 입력들 304
마스킹되지 않은 비트
마스킹된 비트

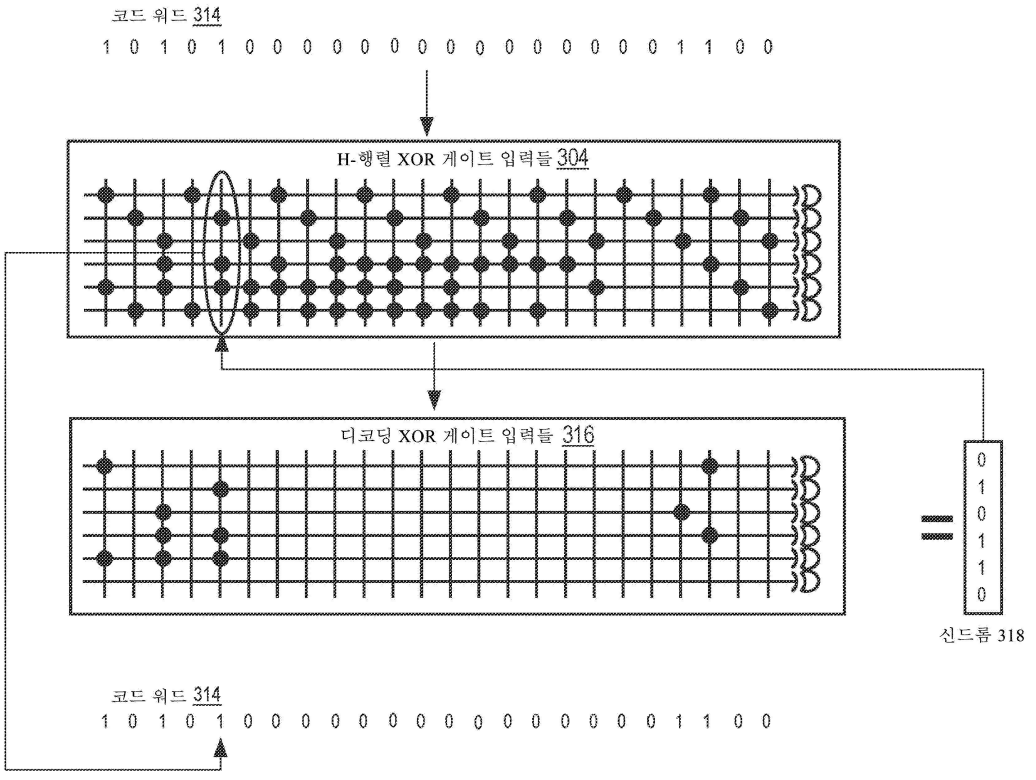
도면3b


G-행렬 303
G=
101 110 001 111 011 010
111 001 101 100 110 011
011 101 010 110 111 101
001 010 101 011 111 110
101 011 111 110 100 001
010 101 011 111 110 101
마스킹되지 않은 비트
마스킹된 비트
G-행렬 XOR 게이트 입력들 305

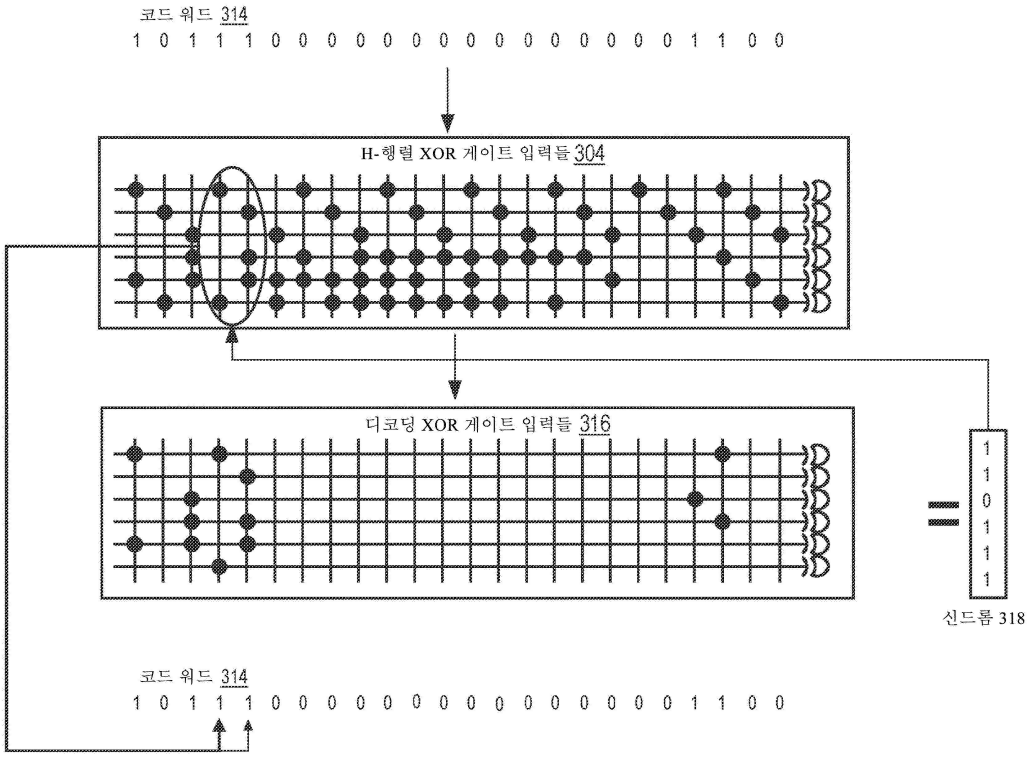
도면3c


데이터 값 306
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
G-행렬 XOR 게이트 입력들 305
인코딩XOR 게이트 입력들 308
체크 비트들310
0 0 1 1 0 0
인코딩된 코드 워드 312
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0

도면3d


코드 워드 314
1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
H-행렬 XOR 게이트 입력들 304
디코딩 XOR 게이트 입력들 316
0 1 0 1 1 0
신드롬 318
코드 워드 314
1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0

도면3e


코드 워드 314
1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
H-행렬 XOR 게이트 입력들 304
디코딩 XOR 게이트 입력들 316
1
1
0
1
1
1
신드롬 318
코드 워드 314
1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0