



(19) 대한민국특허청(KR)

(12) 등록특허공보(B1)

(45) 공고일자 2025년03월12일

(11) 등록번호 10-2779980

(24) 등록일자 2025년03월06일

(51) 국제특허분류(Int. Cl.)

G06F 30/3308 (2020.01) G06F 11/10 (2006.01)

G06F 30/327 (2020.01) G06F 30/331 (2020.01)

(52) CPC특허분류

G06F 30/3308 (2020.01)

G06F 11/102 (2013.01)

(21) 출원번호 10-2023-0159265

(22) 출원일자 2023년11월16일

심사청구일자 2023년11월16일

(56) 선행기술조사문헌

A. Mahmudi 외 1인, "Designing VHDL to Simulate the Error Correction of Hamming Code", JOURNAL OF SCIENCE AND APPLIED ENGINEERING (JSAE), 2018.10.31., pp. 68-74.
1부.*

(뒷면에 계속)

전체 청구항 수 : 총 8 항

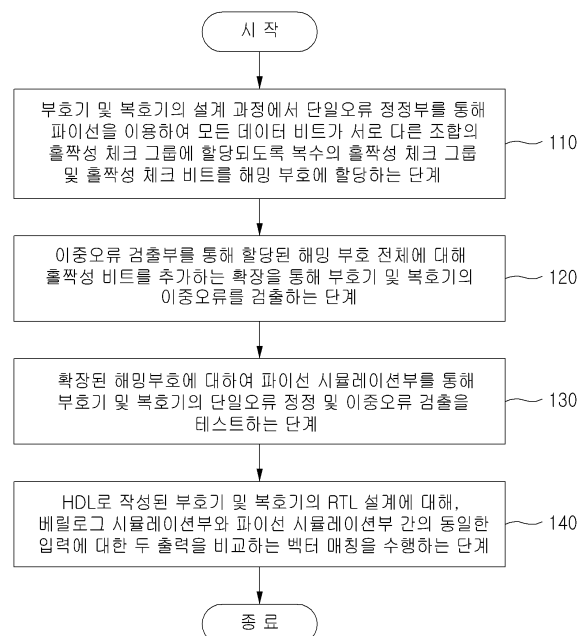
심사관 : 김경완

(54) 발명의 명칭 오류정정부호 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 프레임워크

(57) 요약

오류정정부호 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 방법 및 시스템이 제시된다. 본 발명에서 제안하는 오류정정부호 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 방법은 부호기 및 복호기의 설계 과정에서 단일오류 정정부부를 통해 파이썬을 이용하여 모든 데이터 비트가 서로 다른 조합의 출력성 체크 그룹에 할당되도록 복수의 출력성 체크 그룹 및 출력성 체크 비트를 해밍 부호에 할당하는 단계

(뒷면에 계속)

대표도 - 도1

그룹(parity-check group)에 할당되도록 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트(parity-check bit)를 해밍 부호에 할당하는 단계, 이중오류 검출부를 통해 상기 할당된 해밍 부호 전체에 대해 홀짝성 비트(parity bit)를 추가하는 확장을 통해 부호기 및 복호기의 이중오류를 검출하는 단계, 상기 확장된 해밍부호에 대하여 파이션 시뮬레이션부를 통해 부호기 및 복호기의 단일오류 정정 및 이중오류 검출을 테스트하는 단계 및 HDL(Hardware Description Language)로 작성된 상기 부호기 및 복호기의 RTL(Register-transfer level) 설계에 대해, 베릴로그 시뮬레이션부와 상기 파이션 시뮬레이션부 간의 동일한 입력에 대한 두 출력을 비교하는 벡터 매칭(vector matching)을 수행하는 단계를 포함한다.

(52) CPC특허분류

G06F 11/1048 (2013.01)

G06F 30/327 (2020.01)

G06F 30/331 (2020.01)

(56) 선행기술조사문헌

맹예지 외 1인, "Python을 활용한 RTL 설계 검증 자동화", 2022 한국정보기술학회 추계 종합학술대회 논문집, 2022.12.31., pp. 629-631. 1부.*

KR1019950033822 A

US20040177292 A1

KR101942030 B1

KR1020220106651 A

*는 심사관에 의하여 인용된 문헌

이 발명을 지원한 국가연구개발사업

과제고유번호	1711194127
과제번호	2021M3F3A2A01037531
부처명	과학기술정보통신부
과제관리(전문)기관명	한국연구재단
연구사업명	신소자원천기술개발
연구과제명	[Ezbaro] Dynamic logic 기반 고성능 STMG 로직 회로 설계
기 여 율	3/10
과제수행기관명	인하대학교
연구기간	2023.01.01 ~ 2023.12.31

이 발명을 지원한 국가연구개발사업

과제고유번호	1415186528
과제번호	20009972
부처명	산업통상자원부
과제관리(전문)기관명	한국산업기술기획평가원
연구사업명	차세대지능형반도체기술개발
연구과제명	고효율 초저전력 경량 엣지 디바이스용 소자회로 및 SoC 개발(4차년도)
기 여 율	3/10
과제수행기관명	인하대학교
연구기간	2023.01.01 ~ 2023.12.31

이 발명을 지원한 국가연구개발사업

과제고유번호	1711193592
과제번호	2021-0-02052-003
부처명	과학기술정보통신부
과제관리(전문)기관명	정보통신기획평가원
연구사업명	대학ITRC센터육성지원사업(교육운영수익)
연구과제명	[Ezbaro][정부] 스마트 모빌리티를 위한 인공지능 시스템반도체 핵심 기술 개발 및
인력 양성(3차년도)	
기 여 율	4/10
과제수행기관명	인하대학교
연구기간	2023.01.01 ~ 2023.12.31

공지예외적용 : 있음

명세서

청구범위

청구항 1

부호기 및 복호기의 설계 과정에서 단일오류 정정부를 통해 파이썬(Python)을 이용하여 모든 데이터 비트가 서로 다른 조합의 홀짝성 체크 그룹(parity-check group)에 할당되도록 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트(parity-check bit)를 해밍 부호에 할당하는 단계;

이중오류 검출부를 통해 상기 할당된 해밍 부호 전체에 대해 홀짝성 비트(parity bit)를 추가하는 확장을 통해 부호기 및 복호기의 이중오류를 검출하는 단계;

상기 확장된 해밍부호에 대하여 파이썬 시뮬레이션부를 통해 부호기 및 복호기의 단일오류 정정 및 이중오류 검출을 테스트하는 단계; 및

HDL(Hardware Description Language)로 작성된 상기 부호기 및 복호기의 RTL(Register-transfer level) 설계에 대해, 베릴로그 시뮬레이션부와 상기 파이썬 시뮬레이션부 간의 동일한 입력에 대한 두 출력을 비교하는 벡터 매칭(vector matching)을 수행하는 단계

를 포함하는 부호기 및 복호기의 설계 자동 검증 방법.

청구항 2

제1항에 있어서,

상기 부호기 및 복호기의 설계 과정에서 단일오류 정정부를 통해 파이썬을 이용하여 모든 데이터 비트가 서로 다른 조합의 홀짝성 체크 그룹에 할당되도록 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트를 해밍 부호에 할당하는 단계는,

상기 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트를 해밍 부호에 할당하기 위해 임의로 결정된 데이터와 오류 위치를 txt 파일로 출력하고, 상기 txt 파일로 출력된 데이터와 오류 위치를 복수의 테스트 셋(test set)으로써 사용하여, 오류정정부호 부호화 및 복호화 모듈을 통해 데이터 부호화, 오류 정정 및 복호화를 수행하며, 상기 오류 위치의 부호 비트를 반전시켜 오류를 발생시키는

부호기 및 복호기의 설계 자동 검증 방법.

청구항 3

제2항에 있어서,

상기 확장된 해밍부호에 대하여 파이썬 시뮬레이션부를 통해 부호기 및 복호기의 단일오류 정정 및 이중오류 검출을 테스트하는 단계는,

파이썬 시뮬레이션 후, 상기 테스트 셋 각각에 대해 상기 결정된 오류와 검출된 오류의 개수를 비교하고, 단일 오류 또는 오류가 없는 경우에는 복호화된 데이터가 상기 결정된 데이터와 같은지 비교함으로써 부호화 및 복호화 모듈의 단일오류 정정 및 이중오류 검출을 테스트하는

부호기 및 복호기의 설계 자동 검증 방법.

청구항 4

제1항에 있어서,

상기 HDL로 작성된 상기 부호기 및 복호기의 RTL 설계에 대해, 베릴로그 시뮬레이션부와 상기 파이썬 시뮬레이션부 간의 동일한 입력에 대한 두 출력을 비교하는 벡터 매칭을 수행하는 단계는,

상기 파이썬 시뮬레이션부를 통해 파이썬 시뮬레이션을 수행한 후 베릴로그 HDL 컴파일러(Verilog HDL Compiler)를 실행하여 컴파일하고, 베릴로그 시뮬레이션을 수행하며,

상기 베릴로그 시뮬레이션부로부터 획득한 출력값을 파이선 시뮬레이션부로부터 획득한 출력값과 비교하는 벡터 매칭을 수행함으로써 파이선으로 구현된 부호화 및 복호화 모듈과 RTL 설계 간 동일성을 검증하는

부호기 및 복호기의 설계 자동 검증 방법.

청구항 5

부호기 및 복호기의 설계 과정에서 파이선(Python)을 이용하여 모든 데이터 비트가 서로 다른 조합의 홀짝성 체크 그룹(parity-check group)에 할당되도록 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트(parity-check bit)를 해밍 부호에 할당하는 단일오류 정정부;

상기 할당된 해밍 부호 전체에 대해 홀짝성 비트(parity bit)를 추가하는 확장을 통해 부호기 및 복호기의 이중 오류를 검출하는 이중오류 검출부;

상기 확장된 해밍부호에 대하여 부호기 및 복호기의 단일오류 정정 및 이중오류 검출을 테스트하는 파이선 시뮬레이션부; 및

HDL(Hardware Description Language)로 작성된 상기 부호기 및 복호기의 RTL(Register-transfer level) 설계에 대해, 상기 파이선 시뮬레이션부와 동일한 입력에 대한 두 출력을 비교하는 벡터 매칭(vector matching)을 수행하는 베릴로그 시뮬레이션부

를 포함하는 부호기 및 복호기의 설계 자동 검증 시스템.

청구항 6

제5항에 있어서,

상기 단일오류 정정부는,

상기 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트를 해밍 부호에 할당하기 위해 임의로 결정된 데이터와 오류 위치를 txt 파일로 출력하고, 상기 txt 파일로 출력된 데이터와 오류 위치를 복수의 테스트 셋(test set)으로써 사용하여, 오류정정부호 부호화 및 복호화 모듈을 통해 데이터 부호화, 오류 정정 및 복호화를 수행하며, 상기 오류 위치의 부호 비트를 반전시켜 오류를 발생시키는

부호기 및 복호기의 설계 자동 검증 시스템.

청구항 7

제6항에 있어서,

상기 파이선 시뮬레이션부는,

파이선 시뮬레이션 후, 상기 테스트 셋 각각에 대해 상기 결정된 오류와 검출된 오류의 개수를 비교하고, 단일 오류 또는 오류가 없는 경우에는 복호화된 데이터가 상기 결정된 데이터와 같은지 비교함으로써 부호화 및 복호화 모듈의 단일오류 정정 및 이중오류 검출을 테스트하는

부호기 및 복호기의 설계 자동 검증 시스템.

청구항 8

제5항에 있어서,

상기 베릴로그 시뮬레이션부는,

상기 파이선 시뮬레이션부를 통해 파이선 시뮬레이션을 수행한 후 베릴로그 HDL 컴파일러(Verilog HDL Compiler)를 실행하여 컴파일하고, 베릴로그 시뮬레이션을 수행하며,

상기 베릴로그 시뮬레이션부로부터 획득한 출력값을 파이선 시뮬레이션부로부터 획득한 출력값과 비교하는 벡터 매칭을 수행함으로써 파이선으로 구현된 부호화 및 복호화 모듈과 RTL 설계 간 동일성을 검증하는

부호기 및 복호기의 설계 자동 검증 시스템.

발명의 설명

기술 분야

[0001] 본 발명은 오류정정부호 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 방법 및 시스템에 관한 것이다.

배경 기술

[0002] DRAM(Dynamic Random Access Memory) 셀 밀도의 증가로, 셀 간섭으로 인한 DRAM의 신뢰성이 감소하게 되었고, 이를 해결하기 위해 JEDEC(Joint Electron Device Engineering Council)에서는 DDR5 SDRAM(컴퓨팅 인터페이스 개발에서 DDR5 SDRAM(Double Data Rate Fifth-Generation Synchronous Dynamic Random-Access Memory) 규격부터는 중단간 오류를 검출하기 위한 CRC(Cyclic Redundancy Check) 외에도, On-die ECC(Error Correction Code)를 도입하여, DRAM 셀 밀도 증가에 따른 오류를 검출 및 정정하여 신뢰성을 높였다. 하지만 적용된 On-die ECC는 단일오류정정만을 보장하므로, 에일리어싱(aliasing)을 줄여 다중오류 검출률을 높이거나, 다중오류 검출이 가능한 오류 정정 부호를 최소 비용으로 DRAM 셀에 적용하는 연구들이 진행 중이다.

[0003] 부호화율을 높이기 위해 큰 크기의 부호를 사용하는 부호기 및 복호기를 설계하지만, 긴 길이의 부호의 유효성을 단순 계산으로 검사하기는 매우 번거롭다.

[0004] 따라서, 파이썬(Python)으로 구현한 오류정정부호의 부호화 및 복호화 프로그램과 베릴로그(Verilog)로 작성된 부호기 및 복호기 RTL(Register-Transfer Level) 설계를 검증하는 방안을 필요로 한다.

선행기술문헌

비특허문헌

[0005] (비특허문헌 0001) [1] Micron Technology, DDR5 SDRAM Product Core Data Sheet, Oct 2022.
(비특허문헌 0002) [2] R. W. Hamming, "Error detecting and error correcting codes," in The Bell System Technical Journal, vol. 29, no. 2, pp. 147-160, April 1950.

발명의 내용

해결하려는 과제

[0006] 본 발명이 이루고자 하는 기술적 과제는 파이썬(Python)으로 구현한 오류정정부호의 부호화 및 복호화 프로그램과 베릴로그(Verilog)로 작성된 부호기 및 복호기 RTL(Register-transfer level) 설계에 대해, 동일한 입력에 대한 두 출력을 비교하기 위한 벡터 매칭(vector matching)을 수행하는 파이썬 스크립트(Python Script)를 제안한다.

과제의 해결 수단

[0007] 일 측면에 있어서, 본 발명에서 제안하는 오류정정부호 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 방법은 부호기 및 복호기의 설계 과정에서 단일오류 정정부를 통해 파이썬(Python)을 이용하여 모든 데이터 비트가 서로 다른 조합의 홀짝성 체크 그룹(parity-check group)에 할당되도록 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트(parity-check bit)를 해밍 부호에 할당하는 단계, 이중오류 검출부를 통해 상기 할당된 해밍 부호 전체에 대해 홀짝성 비트(parity bit)를 추가하는 확장을 통해 부호기 및 복호기의 이중오류를 검출하는 단계, 상기 확장된 해밍부호에 대하여 파이썬 시뮬레이션부를 통해 부호기 및 복호기의 단일오류 정정 및 이중오류 검출을 테스트하는 단계 및 HDL(Hardware Description Language)로 작성된 상기 부호기 및 복호기의 RTL(Register-transfer level) 설계에 대해, 베릴로그 시뮬레이션부와 상기 파이썬 시뮬레이션부 간의 동일한 입력에 대한 두 출력을 비교하는 벡터 매칭(vector matching)을 수행하는 단계를 포함한다.

[0008] 상기 부호기 및 복호기의 설계 과정에서 단일오류 정정부를 통해 파이썬을 이용하여 모든 데이터 비트가 서로 다른 조합의 홀짝성 체크 그룹에 할당되도록 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트를 해밍 부호에 할당하는 단계는 상기 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트를 해밍 부호에 할당하기 위해 임의로 결정된 데이터와 오류 위치를 txt 파일로 출력하고, 상기 txt 파일로 출력된 데이터와 오류 위치를 복수의 테스트 셋

(test set)으로써 사용하여, 오류정정부호 부호화 및 복호화 모듈을 통해 데이터 부호화, 오류 정정 및 복호화를 수행하며, 상기 오류 위치의 부호 비트를 반전시켜 오류를 발생시킨다.

[0009] 상기 확장된 해밍부호에 대하여 파이선 시뮬레이션부를 통해 부호기 및 복호기의 단일오류 정정 및 이중오류 검출을 테스트하는 단계는 파이선 시뮬레이션 후, 상기 테스트 셋 각각에 대해 상기 결정된 오류와 검출된 오류의 개수를 비교하고, 단일오류 또는 오류가 없는 경우에는 복호화된 데이터가 상기 결정된 데이터와 같은지 비교함으로써 부호화 및 복호화 모듈의 단일오류 정정 및 이중오류 검출을 테스트한다.

[0010] 상기 HDL로 작성된 상기 부호기 및 복호기의 RTL 설계에 대해, 베릴로그 시뮬레이션부와 상기 파이선 시뮬레이션부 간의 동일한 입력에 대한 두 출력을 비교하는 벡터 매칭을 수행하는 단계는 상기 파이선 시뮬레이션부를 통해 파이선 시뮬레이션을 수행한 후 베릴로그 HDL 컴파일러(Verilog HDL Compiler)를 실행하여 컴파일하고, 베릴로그 시뮬레이션을 수행하며, 상기 베릴로그 시뮬레이션부로부터 획득한 출력값을 파이선 시뮬레이션부로부터 획득한 출력값과 비교하는 벡터 매칭을 수행함으로써 파이선으로 구현된 부호화 및 복호화 모듈과 RTL 설계 간 동일성을 검증한다.

[0011] 또 다른 일 측면에 있어서, 본 발명에서 제안하는 오류정정부호 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 시스템은 부호기 및 복호기의 설계 과정에서 파이선(Python)을 이용하여 모든 데이터 비트가 서로 다른 조합의 홀짝성 체크 그룹(parity-check group)에 할당되도록 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트(parity-check bit)를 해밍 부호에 할당하는 단일오류 정정부, 상기 할당된 해밍 부호 전체에 대해 홀짝성 비트(parity bit)를 추가하는 확장을 통해 부호기 및 복호기의 이중오류를 검출하는 이중오류 검출부, 상기 확장된 해밍부호에 대하여 부호기 및 복호기의 단일오류 정정 및 이중오류 검출을 테스트하는 파이선 시뮬레이션부 및 HDL(Hardware Description Language)로 작성된 상기 부호기 및 복호기의 RTL(Register-transfer level) 설계에 대해, 상기 파이선 시뮬레이션부와 동일한 입력에 대한 두 출력을 비교하는 벡터 매칭(vector matching)을 수행하는 베릴로그 시뮬레이션부를 포함한다.

발명의 효과

[0012] 본 발명의 실시예들에 따르면 파이선(Python)으로 구현한 오류정정부호의 부호화 및 복호화 프로그램과 베릴로그(Verilog)로 작성된 부호기 및 복호기 RTL(Register-transfer level) 설계에 대해, 파이선 스크립트(Python Script)를 통해 동일한 입력에 대한 두 출력을 비교하기 위한 벡터 매칭(vector matching)을 수행한다. 본 발명의 실시예에 따르면, 단일오류정정-이중오류검출(Single Error Correction-Double Error Detection; SECDED)이 가능한 확장된 해밍부호를 구현하여 사용할 수 있다.

도면의 간단한 설명

[0013] 도 1은 본 발명의 일 실시예에 따른 오류정정부호 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 방법을 설명하기 위한 흐름도이다.

도 2는 본 발명의 일 실시예에 따른 오류정정부호 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 시스템의 구성을 나타내는 도면이다.

도 3은 본 발명의 일 실시예에 따른 확장된 해밍부호의 파이선 시뮬레이션의 순서도이다.

도 4는 본 발명의 일 실시예에 따른 확장된 해밍부호 부호기 및 복호기의 구조를 설명하기 위한 도면이다.

도 5는 본 발명의 일 실시예에 따른 파이선을 이용한 벡터 매칭 순서도이다.

도 6은 본 발명의 일 실시예에 따른 파이선 시뮬레이션 및 벡터 매칭 실행 결과를 나타내는 도면이다.

도 7은 본 발명의 일 실시예에 따른 의도한 오류로 벡터 매칭에 실패한 경우의 예시도이다.

발명을 실시하기 위한 구체적인 내용

[0014] 본 발명은 Python을 사용하여 큰 크기의 오류정정부호 부호기 및 복호기의 RTL(Register-transfer level) 설계를 검증하는 방법을 제시한다. 같은 입력을 인가할 때, Python으로 작성된 오류정정부호의 부호화 및 복호화 프로그램과 베릴로그 테스트벤치(Verilog testbench)의 출력을 비교하여 RTL 설계 검증을 수행한다. 또한 입력 벡터와 출력 벡터를 비교함으로써 오류 정정 능력도 시험한다. 본 발명에서는 단일오류정정-이중오류검출이 가능한 확장된 해밍부호를 예시로 사용한다. 이하, 본 발명의 실시 예를 첨부된 도면을 참조하여 상세하게

설명한다.

- [0016] 도 1은 본 발명의 일 실시예에 따른 오류정정부호 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 방법을 설명하기 위한 흐름도이다.
- [0017] 제안하는 오류정정부호 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 방법은 부호기 및 복호기의 설계 과정에서 단일오류 정정부를 통해 파이썬(Python)을 이용하여 모든 데이터 비트가 서로 다른 조합의 홀짝성 체크 그룹(parity-check group)에 할당되도록 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트(parity-check bit)를 해밍 부호에 할당하는 단계(110), 이중오류 검출부를 통해 상기 할당된 해밍 부호 전체에 대해 홀짝성 비트(parity bit)를 추가하는 확장을 통해 부호기 및 복호기의 이중오류를 검출하는 단계(120), 상기 확장된 해밍부호에 대하여 파이썬 시뮬레이션부를 통해 부호기 및 복호기의 단일오류 정정 및 이중오류 검출을 테스트하는 단계(130) 및 HDL(Hardware Description Language)로 작성된 상기 부호기 및 복호기의 RTL(Register-transfer level) 설계에 대해, 베릴로그 시뮬레이션부와 상기 파이썬 시뮬레이션부 간의 동일한 입력에 대한 두 출력을 비교하는 벡터 매칭(vector matching)을 수행하는 단계(140)를 포함한다.
- [0018] 단계(110)에서, 부호기 및 복호기의 설계 과정에서 단일오류 정정부를 통해 파이썬(Python)을 이용하여 모든 데이터 비트가 서로 다른 조합의 홀짝성 체크 그룹(parity-check group)에 할당되도록 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트(parity-check bit)를 해밍 부호에 할당한다.
- [0019] 본 발명의 실시예에 따르면, 상기 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트를 해밍 부호에 할당하기 위해 임의로 결정된 데이터와 오류 위치를 txt 파일로 출력하고, 상기 txt 파일로 출력된 데이터와 오류 위치를 복수의 테스트 셋(test set)으로써 사용하여, 오류정정부호 부호화 및 복호화 모듈을 통해 데이터 부호화, 오류 정정 및 복호화를 수행하며, 상기 오류 위치의 부호 비트를 반전시켜 오류를 발생시킨다.
- [0020] 단계(120)에서, 이중오류 검출부를 통해 상기 할당된 해밍 부호 전체에 대해 홀짝성 비트(parity bit)를 추가하는 확장을 통해 부호기 및 복호기의 이중오류를 검출한다.
- [0021] 단계(130)에서, 상기 확장된 해밍부호에 대하여 파이썬 시뮬레이션부를 통해 부호기 및 복호기의 단일오류 정정 및 이중오류 검출을 테스트한다.
- [0022] 본 발명의 실시예에 따르면, 파이썬 시뮬레이션 후 상기 테스트 셋 각각에 대해 상기 결정된 오류와 검출된 오류의 개수를 비교하고, 단일오류 또는 오류가 없는 경우에는 복호화된 데이터가 상기 결정된 데이터와 같은지 비교함으로써 부호화 및 복호화 모듈의 단일오류 정정 및 이중오류 검출을 테스트한다.
- [0023] 단계(140)에서, 상기 HDL로 작성된 상기 부호기 및 복호기의 RTL 설계에 대해, 베릴로그 시뮬레이션부와 상기 파이썬 시뮬레이션부 간의 동일한 입력에 대한 두 출력을 비교하는 벡터 매칭을 수행한다.
- [0024] 본 발명의 실시예에 따르면, 상기 파이썬 시뮬레이션부를 통해 파이썬 시뮬레이션을 수행한 후 베릴로그 HDL 컴파일러(Verilog HDL Compiler)를 실행하여 컴파일하고, 베릴로그 시뮬레이션을 수행한다. 상기 베릴로그 시뮬레이션부로부터 획득한 출력값을 파이썬 시뮬레이션부로부터 획득한 출력값과 비교하는 벡터 매칭을 수행함으로써 파이썬으로 구현된 부호화 및 복호화 모듈과 RTL 설계 간 동일성을 검증한다.
- [0026] 도 2는 본 발명의 일 실시예에 따른 오류정정부호 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 시스템의 구성을 나타내는 도면이다.
- [0027] 본 실시예에 따른 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 시스템(200)은 프로세서(210), 버스(220), 네트워크 인터페이스(230), 메모리(240) 및 데이터베이스(250)를 포함할 수 있다. 메모리(240)는 운영체제(241) 및 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 루틴(242)을 포함할 수 있다. 프로세서(210)는 단일 오류 정정부(211), 이중 오류 검출부(212), 파이썬 시뮬레이션부(213) 및 베릴로그 시뮬레이션부(214)를 포함할 수 있다. 다른 실시예들에서 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 시스템(200)은 도 2의 구성요소들보다 더 많은 구성요소들을 포함할 수도 있다. 그러나, 대부분의 종래기술적 구성요소들을 명확하게 도시할 필요성은 없다. 예를 들어, 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 시스템(200)은 디스플레이나 트랜시버(transceiver)와 같은 다른 구성요소들을 포함할 수도 있다.
- [0028] 메모리(240)는 컴퓨터에서 판독 가능한 기록 매체로서, RAM(random access memory), ROM(read only memory) 및 디스크 드라이브와 같은 비소멸성 대용량 기록장치(permanent mass storage device)를 포함할 수 있다. 또한, 메모리(240)에는 운영체제(241)와 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 루틴(242)을 위한 프로그램 코드가 저장될 수 있다. 이러한 소프트웨어 구성요소들은 드라이브 메커니즘(drive mechanism, 미도시)

을 이용하여 메모리(240)와는 별도의 컴퓨터에서 판독 가능한 기록 매체로부터 로딩될 수 있다. 이러한 별도의 컴퓨터에서 판독 가능한 기록 매체는 플로피 드라이브, 디스크, 테이프, DVD/CD-ROM 드라이브, 메모리 카드 등의 컴퓨터에서 판독 가능한 기록 매체(미도시)를 포함할 수 있다. 다른 실시예에서 소프트웨어 구성요소들은 컴퓨터에서 판독 가능한 기록 매체가 아닌 네트워크 인터페이스(230)를 통해 메모리(240)에 로딩될 수도 있다.

- [0029] 버스(220)는 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 시스템(200)의 구성요소들간의 통신 및 데이터 전송을 가능하게 할 수 있다. 버스(220)는 고속 시리얼 버스(high-speed serial bus), 병렬 버스(parallel bus), SAN(Storage Area Network) 및/또는 다른 적절한 통신 기술을 이용하여 구성될 수 있다.
- [0030] 네트워크 인터페이스(230)는 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 시스템(200)을 컴퓨터 네트워크에 연결하기 위한 컴퓨터 하드웨어 구성요소일 수 있다. 네트워크 인터페이스(230)는 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 시스템(200)을 무선 또는 유선 커넥션을 통해 컴퓨터 네트워크에 연결시킬 수 있다.
- [0031] 데이터베이스(250)는 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증을 위해 필요한 모든 정보를 저장 및 유지하는 역할을 할 수 있다. 도 2에서는 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 시스템(200)의 내부에 데이터베이스(250)를 구축하여 포함하는 것으로 도시하고 있으나, 이에 한정되는 것은 아니며 시스템 구현 방식이나 환경 등에 따라 생략될 수 있고 혹은 전체 또는 일부의 데이터베이스가 별개의 다른 시스템 상에 구축된 외부 데이터베이스로서 존재하는 것 또한 가능하다.
- [0032] 프로세서(210)는 기본적인 산술, 로직 및 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 시스템(200)의 입출력 연산을 수행함으로써, 컴퓨터 프로그램의 명령을 처리하도록 구성될 수 있다. 명령은 메모리(240) 또는 네트워크 인터페이스(230)에 의해, 그리고 버스(220)를 통해 프로세서(210)로 제공될 수 있다. 프로세서(210)는 단일 오류 정정부(211), 이중 오류 검출부(212), 파이션 시뮬레이션부(213) 및 베릴로그 시뮬레이션부(214)를 위한 프로그램 코드를 실행하도록 구성될 수 있다. 이러한 프로그램 코드는 메모리(240)와 같은 기록 장치에 저장될 수 있다.
- [0033] 단일 오류 정정부(211), 이중 오류 검출부(212), 파이션 시뮬레이션부(213) 및 베릴로그 시뮬레이션부(214)는 도 1의 단계들(110~140)을 수행하기 위해 구성될 수 있다.
- [0034] 부호기 및 복호기의 하드웨어 가속기 설계 및 자동 검증 시스템(200)은 단일 오류 정정부(211), 이중 오류 검출부(212), 파이션 시뮬레이션부(213) 및 베릴로그 시뮬레이션부(214)를 포함할 수 있다.
- [0035] 본 발명의 실시예에 따른 단일 오류 정정부(211)는 부호기 및 복호기의 설계 과정에서 파이션(Python)을 이용하여 모든 데이터 비트가 서로 다른 조합의 홀짝성 체크 그룹(parity-check group)에 할당되도록 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트(parity-check bit)를 해밍 부호에 할당한다.
- [0036] 본 발명의 실시예에 따른 단일 오류 정정부(211)는 상기 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트를 해밍 부호에 할당하기 위해 임의로 결정된 데이터와 오류 위치를 txt 파일로 출력하고, 상기 txt 파일로 출력된 데이터와 오류 위치를 복수의 테스트 셋(test set)으로써 사용하여, 오류정정부 부호화 및 복호화 모듈을 통해 데이터 부호화, 오류 정정 및 복호화를 수행하며, 상기 오류 위치의 부호 비트를 반전시켜 오류를 발생시킨다.
- [0037] 본 발명의 실시예에 따른 이중 오류 검출부(212)는 상기 할당된 해밍 부호 전체에 대해 홀짝성 비트(parity bit)를 추가하는 확장을 통해 부호기 및 복호기의 이중오류를 검출한다.
- [0038] 예를 들어, 각 데이터 비트(data bits)에 대해 XOR(배타적논리합) 연산을 수행함으로써 데이터 내 1의 홀짝성(parity)를 계산할 수 있다. 여기서 1bit의 홀짝성 비트(parity bit)를 추가하고, 홀짝성 결과에 따라 0 또는 1을 추가해 짝수개의 1을 가지도록, 짝수 패리티(even parity)를 만족하도록 구성하면 홀수개의 오류를 검출할 수 있다(단일오류검출). 하지만 오류의 위치를 특정할 수 없으므로 오류를 정정할 수는 없다.
- [0039] 본 발명의 실시예에 따르면, R. W. Hamming은 모든 데이터 비트들이 서로 다른 조합의 홀짝성 체크 그룹(parity-check group)에 할당되도록 복수의 홀짝성 체크 그룹 및 홀짝성 체크 비트(parity-check bit)를 부호에 적절히 할당하는 방법을 제안하였다. 본 발명의 실시예에 따르면, 오류 위치에 따라 각 그룹의 홀짝성 조합인 오류 패턴(syndrome)이 달라지므로 단일오류의 위치를 특정할 수 있으며, 해당 bit를 반전시켜 오류를 정정할 수 있다(단일오류정정).
- [0040] 본 발명의 실시예에 따르면, 이러한 해밍 부호 전체에 대한 홀짝성 비트를 추가하는 확장을 통해 이중오류를 검

출할 수 있다(단일오류정정-이중오류검출).

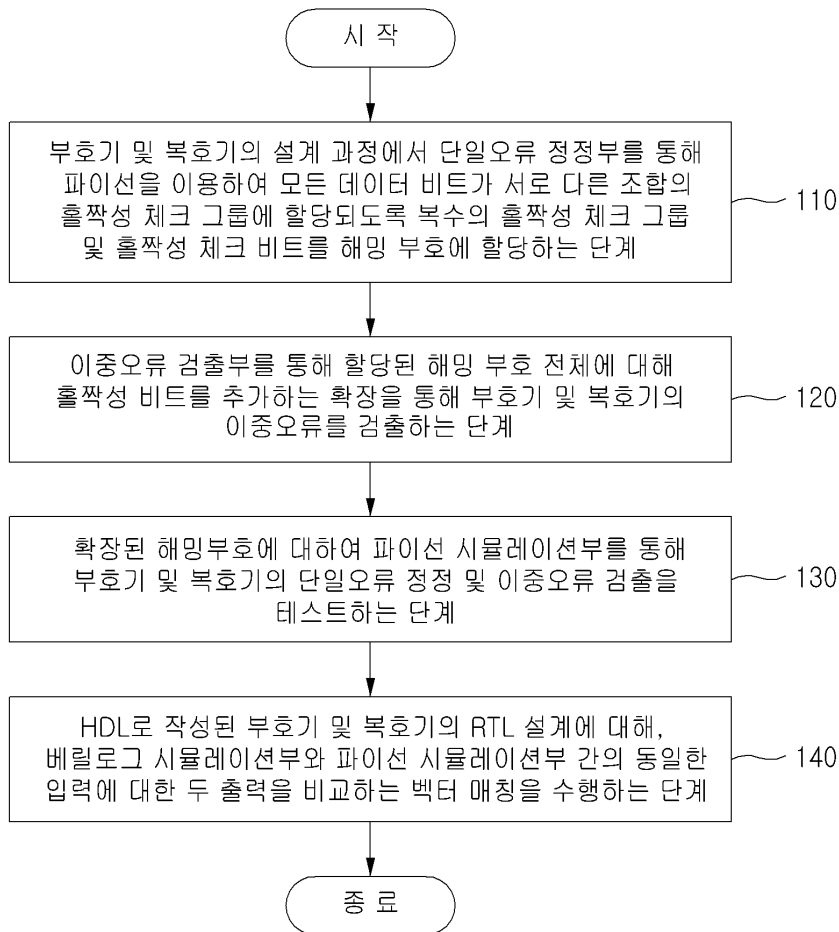
- [0041] 본 발명의 실시예에 따른 파이선 시뮬레이션부(213)는 상기 확장된 해밍부호에 대하여 부호기 및 복호기의 단일 오류 정정 및 이중오류 검출을 테스트한다.
- [0042] 본 발명의 실시예에 따른 파이선 시뮬레이션부(213)는 파이선 시뮬레이션 후 상기 테스트 셋 각각에 대해 상기 결정된 오류와 검출된 오류의 개수를 비교하고, 단일오류 또는 오류가 없는 경우에는 복호화된 데이터가 상기 결정된 데이터와 같은지 비교함으로써 부호화 및 복호화 모듈의 단일오류 정정 및 이중오류 검출을 테스트한다.
- [0043] 본 발명의 실시예에 따른 베릴로그 시뮬레이션부(214)는 상기 HDL로 작성된 상기 부호기 및 복호기의 RTL 설계에 대해, 상기 파이선 시뮬레이션부와와의 동일한 입력에 대한 두 출력을 비교하는 벡터 매칭을 수행한다.
- [0044] 본 발명의 실시예에 따른 베릴로그 시뮬레이션부(214)는 상기 파이선 시뮬레이션부(213) 통해 파이선 시뮬레이션을 수행한 후 베릴로그 HDL 컴파일러(Verilog HDL Compiler)를 실행하여 컴파일하고, 베릴로그 시뮬레이션을 수행한다. 상기 베릴로그 시뮬레이션부(214)로부터 획득한 출력값을 파이선 시뮬레이션부(213)로부터 획득한 출력값과 비교하는 벡터 매칭을 수행함으로써 파이선으로 구현된 부호화 및 복호화 모듈과 RTL 설계 간 동일성을 검증한다.
- [0046] 도 3은 본 발명의 일 실시예에 따른 확장된 해밍부호의 파이선 시뮬레이션의 순서도이다.
- [0047] 본 발명의 실시예에 따른 확장된 해밍부호 부호화 및 복호화 알고리즘을 파이선으로 구현하여 예시로 사용하였다. 도 3은 확장된 해밍부호의 파이선 시뮬레이션(Python Simulation)을 위한 순서도이다. 먼저 "testset_gen.py"(310)를 통해 임의(무작위)로 결정된 데이터(321)와 오류 위치(322)를 txt 파일로 출력하고, 복수의 test set으로써 사용한다. "test.py"(330)의 "test_py" 함수에서는 오류정정부호 부호화 및 복호화 모듈 "SECCED.py"를 이용해 데이터를 부호화(331) 및 오류 정정(332), 복호화(333)를 수행한다. 이때, 앞서 결정한 오류 위치(322)의 부호 bit를 반전(332)시켜 오류를 발생시킨다. 시뮬레이션 후, 각 test set에 대해 지정한 오류(322)와 검출된 오류(350)의 개수를 비교하고, 단일오류 또는 오류가 없는 경우에는 복호화한 데이터(340)가 입력된 데이터(321)와 같은지 비교함으로써 부호/복호화 모듈의 단일오류정정-이중오류검출 능력을 시험할 수 있다.
- [0049] 도 4는 본 발명의 일 실시예에 따른 확장된 해밍부호 부호기 및 복호기의 구조를 설명하기 위한 도면이다.
- [0050] 본 발명의 실시예에 따른 확장된 해밍부호 부호/복호화 알고리즘에 따라 XOR 게이트를 적절히 배치하여 홀짝성(Parity)을 계산하고, 홀짝성 체크 그룹(parity-check group)에 따라 적절한 오류 패턴-오류 위치 해독기를 사용하여 도 4와 같이 확장된 해밍부호 부호기 및 복호기를 베릴로그 등의 HDL로 RTL 설계가 가능하다.
- [0051] 베릴로그로 기술된 RTL 설계를 검증하기 위해 테스트벤치(testbench)를 작성하고 적절한 입력을 인가하였을 때 출력 파형을 확인할 수 있다. 본 발명에서는 베릴로그 HDL의 파일 입출력 함수를 사용해, 앞서 파이선 시뮬레이션에서 사용한 test set을 입력으로 인가했을 때 얻는 출력값을 txt 파일로 출력하는 테스트벤치를 사용하였다.
- [0053] 도 5는 본 발명의 일 실시예에 따른 파이선을 이용한 벡터 매칭 순서도이다.
- [0054] 도 5는 파이선 시뮬레이션과 베릴로그 시뮬레이션(Verilog Simulation) 간의 벡터 매칭(Vector Matching)을 수행하는 과정의 순서도이다. 도 3의 파이선 시뮬레이션(510)을 수행하고, "test.py"(520)의 "test_v" 함수에서 시스템 명령(system command)으로 운영체제에 설치된 베릴로그 HDL 컴파일러(Verilog HDL Compiler)인 'iverilog'를 실행(530)하여 컴파일하고 iverilog runtime engine인 'vvp'를 통해 베릴로그 시뮬레이션을 수행한다(540).
- [0055] 베릴로그 시뮬레이션(Verilog Simulation)으로부터 얻은 출력값을 파이선 시뮬레이션(Python Simulation)으로부터 얻은 출력값과 비교하는 벡터 매칭(Vector Matching)(550)을 수행함으로써 파이선(Python)으로 구현된 부호/복호화 모듈과 RTL 설계 간 동일성을 검증할 수 있다.
- [0057] 도 6은 본 발명의 일 실시예에 따른 파이선 시뮬레이션 및 벡터 매칭 실행 결과를 나타내는 도면이다.
- [0058] 본 발명에서는 파이선으로 구현된 부호/복호화 결과와 베릴로그 시뮬레이션 간의 벡터 매칭을 수행하여 복잡한 부호화 및 복호화의 검증 없이 RTL 설계와 파이선 모듈 및 알고리즘의 동일성을 검증하는 방법을 제안하였다.
- [0059] 도 6은 앞서 설명한 "testset_gen"으로 test set을 생성하고, "test.py"로 파이선과 베릴로그 시뮬레이션의 결과에 대해 벡터 시뮬레이션을 수행한 결과이다. 파이선 스크립트 실행 시 부호 및 데이터의 크기를 인자로 입력

받아 다양한 크기의 부호에 대해서도 검증할 수 있다.

- [0061] 도 7은 본 발명의 일 실시예에 따른 의도한 오류로 벡터 매칭에 실패한 경우의 예시도이다.
- [0062] 도 7을 참조하면, 의도적으로 추가적인 오류를 발생시킨 경우의 결과로, 파이션과 베릴로그 시뮬레이션 결과 및 그 차이를 도시하고 있다. 파이션 모듈과 다른 부호/복호화를 수행하는 RTL 설계에 대해서는 도 7과 같이 일치하지 않는 부분을 출력하여 실패한 경우를 쉽게 확인할 수 있도록 하였다.
- [0063] 본 발명의 실시예에 따른 파이션 스크립트는 실행 결과를 txt 파일로도 출력하므로, 'diff'나 기타 txt 파일 간의 차이를 비교할 수 있는 프로그램을 활용하여 각각의 test set에 대해 분석도 가능하다.
- [0064] 또한, 새로운 오류정정부호의 부호/복호화를 구현할 경우, 오류정정부호의 부호/복호화 알고리즘이 구현된 파이션 모듈 및 RTL 설계만 교체하면 본 발명의 실시예에 따른 RTL 설계 검증 파이션 스크립트를 사용하여 빠르게 검증할 수 있다. 따라서, 새로운 오류정정부호를 위한 RTL 설계 과정에서 소요되는 개발 시간 및 노력을 단축할 수 있다.
- [0066] 이상에서 설명된 장치는 하드웨어 구성요소, 소프트웨어 구성요소, 및/또는 하드웨어 구성요소 및 소프트웨어 구성요소의 조합으로 구현될 수 있다. 예를 들어, 실시예들에서 설명된 장치 및 구성요소는, 예를 들어, 프로세서, 콘트롤러, ALU(arithmetic logic unit), 디지털 신호 프로세서(digital signal processor), 마이크로컴퓨터, FPGA(field programmable gate array), PLU(programmable logic unit), 마이크로프로세서, 또는 명령(instruction)을 실행하고 응답할 수 있는 다른 어떠한 장치와 같이, 하나 이상의 범용 컴퓨터 또는 특수 목적 컴퓨터를 이용하여 구현될 수 있다. 처리 장치는 운영 체제(OS) 및 상기 운영 체제 상에서 수행되는 하나 이상의 소프트웨어 애플리케이션을 수행할 수 있다. 또한, 처리 장치는 소프트웨어의 실행에 응답하여, 데이터를 접근, 저장, 조작, 처리 및 생성할 수도 있다. 이해의 편의를 위하여, 처리 장치는 하나가 사용되는 것으로 설명된 경우도 있지만, 해당 기술분야에서 통상의 지식을 가진 자는, 처리 장치가 복수 개의 처리 요소(processing element) 및/또는 복수 유형의 처리 요소를 포함할 수 있음을 알 수 있다. 예를 들어, 처리 장치는 복수 개의 프로세서 또는 하나의 프로세서 및 하나의 콘트롤러를 포함할 수 있다. 또한, 병렬 프로세서(parallel processor)와 같은, 다른 처리 구성(processing configuration)도 가능하다.
- [0067] 소프트웨어는 컴퓨터 프로그램(computer program), 코드(code), 명령(instruction), 또는 이들 중 하나 이상의 조합을 포함할 수 있으며, 원하는 대로 동작하도록 처리 장치를 구성하거나 독립적으로 또는 결합적으로(collectively) 처리 장치를 명령할 수 있다. 소프트웨어 및/또는 데이터는, 처리 장치에 의하여 해석되거나 처리 장치에 명령 또는 데이터를 제공하기 위하여, 어떤 유형의 기계, 구성요소(component), 물리적 장치, 가상장치(virtual equipment), 컴퓨터 저장 매체 또는 장치에 구체화(embodiment)될 수 있다. 소프트웨어는 네트워크로 연결된 컴퓨터 시스템 상에 분산되어서, 분산된 방법으로 저장되거나 실행될 수도 있다. 소프트웨어 및 데이터는 하나 이상의 컴퓨터 판독 가능 기록 매체에 저장될 수 있다.
- [0068] 실시예에 따른 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 상기 매체에 기록되는 프로그램 명령은 실시예를 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다. 컴퓨터 판독 가능 기록 매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(magnetic media), CD-ROM, DVD와 같은 광기록 매체(optical media), 플롭티컬 디스크(floptical disk)와 같은 자기-광 매체(magneto-optical media), 및 롬(ROM), 램(RAM), 플래시 메모리 등과 같은 프로그램 명령을 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다. 프로그램 명령의 예에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함한다.
- [0069] 이상과 같이 실시예들이 비록 한정된 실시예와 도면에 의해 설명되었으나, 해당 기술분야에서 통상의 지식을 가진 자라면 상기의 기재로부터 다양한 수정 및 변형이 가능하다. 예를 들어, 설명된 기술들이 설명된 방법과 다른 순서로 수행되거나, 및/또는 설명된 시스템, 구조, 장치, 회로 등의 구성요소들이 설명된 방법과 다른 형태로 결합 또는 조합되거나, 다른 구성요소 또는 균등물에 의하여 대치되거나 치환되더라도 적절한 결과가 달성될 수 있다.
- [0070] 그러므로, 다른 구현들, 다른 실시예들 및 특허청구범위와 균등한 것들도 후술하는 특허청구범위의 범위에 속한다.

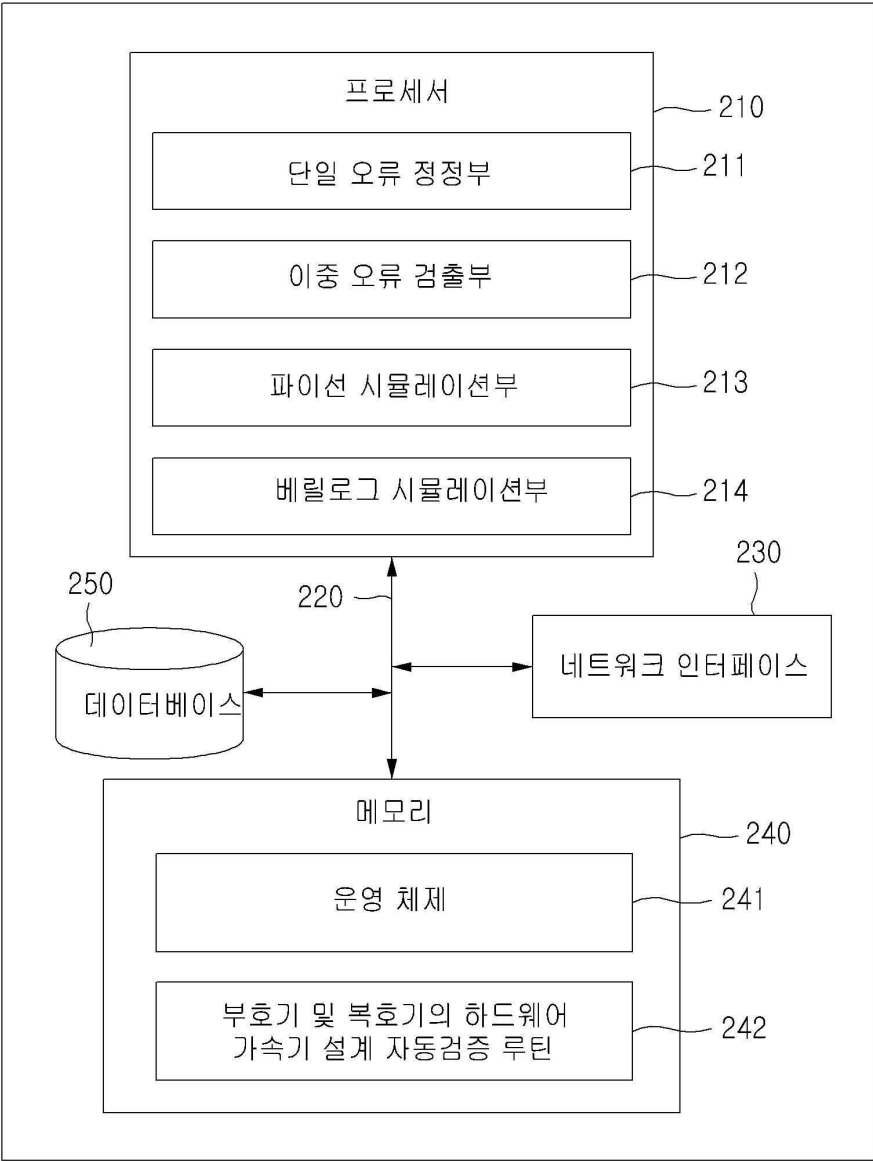
도면

도면1

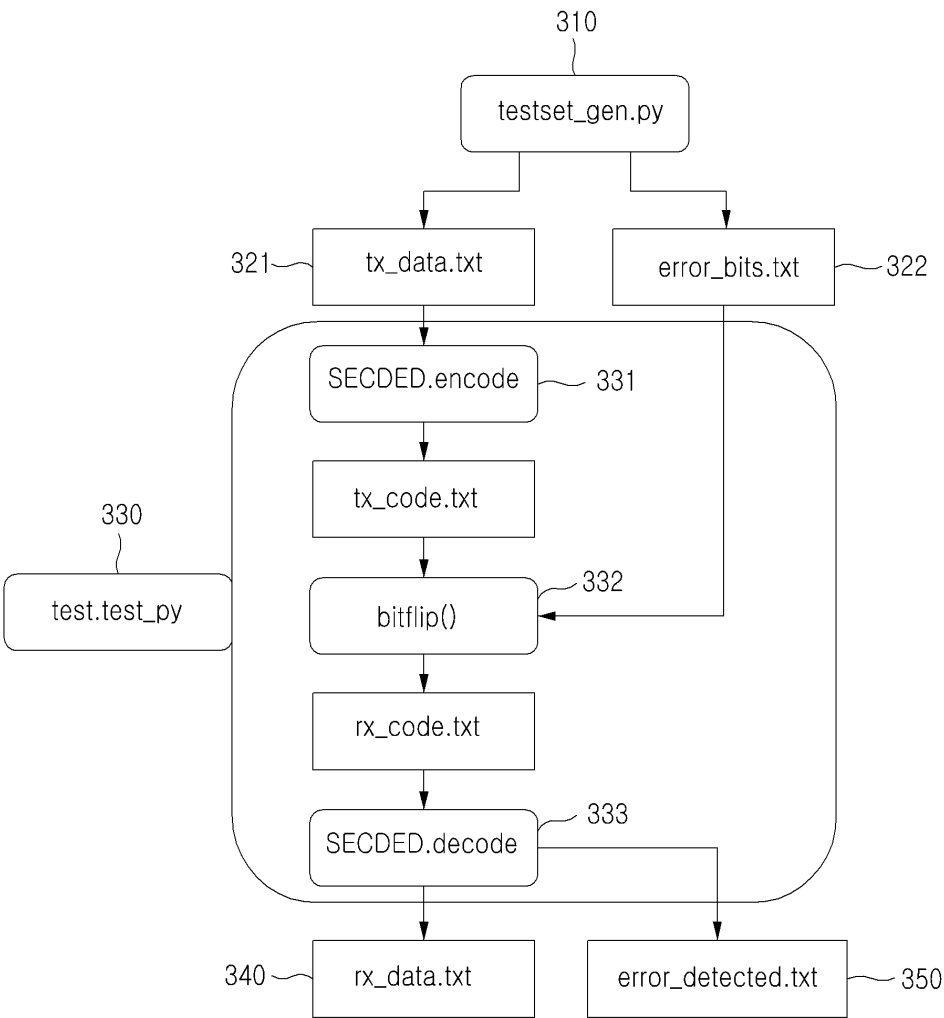


도면2

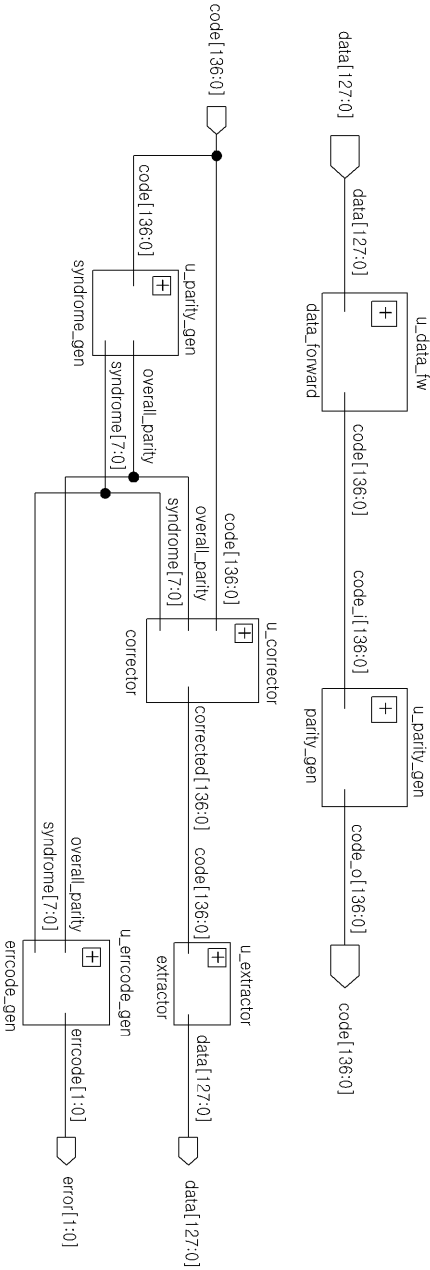
200



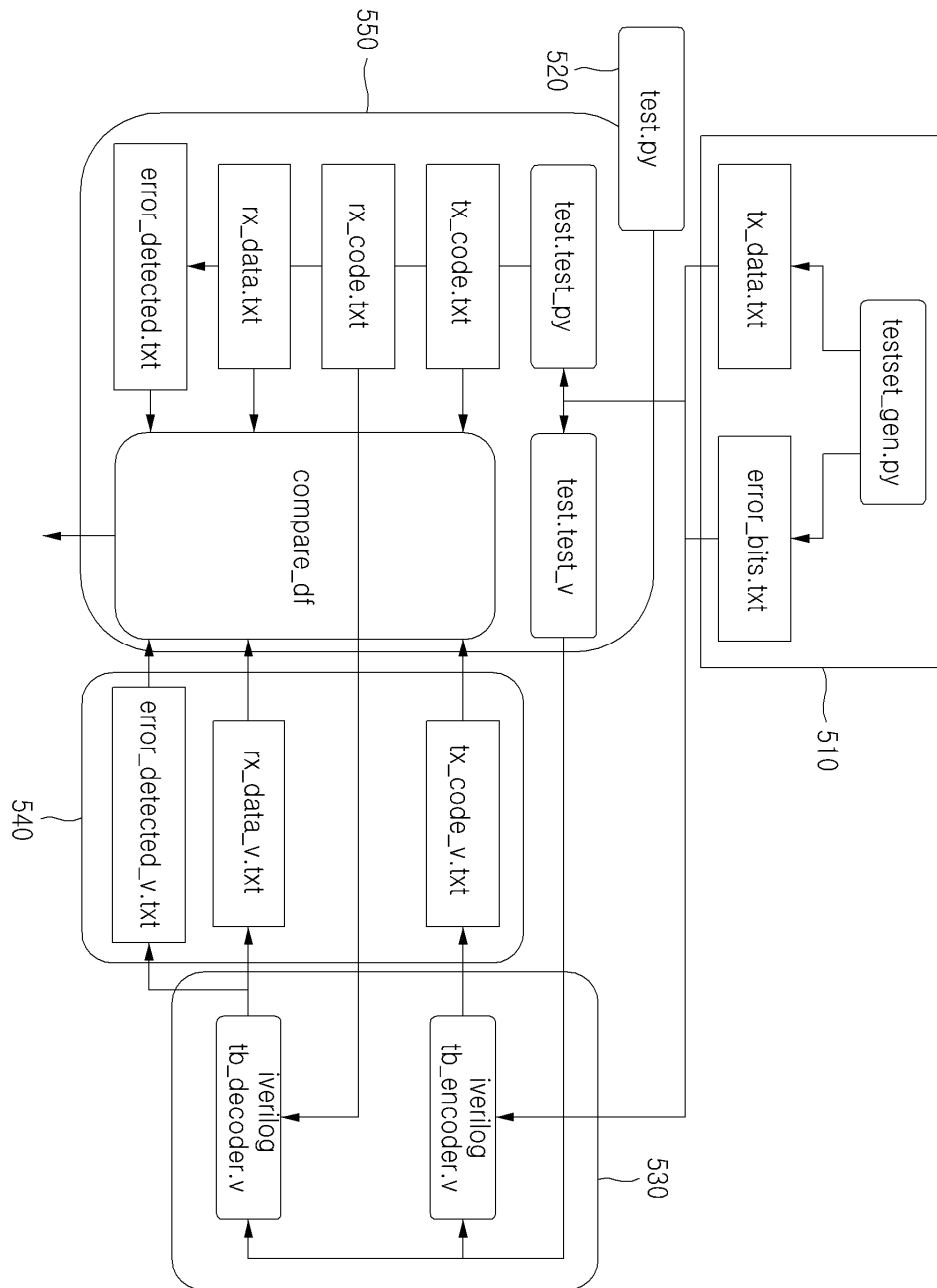
도면3



도면4



도면5



도면6

```
(base) [kig01@circuits_tmp02 SECDDED]$ python testset_gen.py -n 137 -k 128 -sets 256
(base) [kig01@circuits_tmp02 SECDDED]$ python test.py -v -f
No error found in python implementation!
Verilog simulation result is perfectly matched with Python Simulation!
```

도면7

```
*****
tx_code mismatch on testset255!
python : [1 0 0 1 0 1 1 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1
0 1 1 1 1 0 1 0 0 0 1 0 0 0 0 1 1 0 1 1 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1
1 1 1 1 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0
1 0 0 1 1 0 1 0 1 1 1 0 0 1 0 1 0 1 0 0 0 0 1 1 0 1]
diff : [1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
verilog : [0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1
0 0 0 1 1 0 1 0 0 0 1 0 0 0 0 1 1 0 1 1 1 1 0 0 0 0 0 1 1 1 0 1 0 1 0 1
1 1 1 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0
0 0 0 1 1 0 1 0 1 1 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0]
```

Verilog Implementation validation failed!