(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2022년12월30일
(11) 등록번호 10-2483739
(24) 등록일자 2022년12월28일

(51) 국제특허분류(Int. Cl.)
*G11C 29/44* (2006.01) *G06F 11/07* (2006.01)
*G11C 15/04* (2006.01) *G11C 29/40* (2006.01)
*G11C 7/10* (2021.01)
(52) CPC특허분류
*G11C 29/44* (2013.01)
*G06F 11/0793* (2013.01)
(21) 출원번호 10-2021-0091570
(22) 출원일자 2021년07월13일
심사청구일자 2021년07월13일
(56) 선행기술조사문헌
KR1020140000454 A
*(뒷면에 계속)*
전체 청구항 수 : 총 15 항

(73) 특허권자
**연세대학교 산학협력단**
서울특별시 서대문구 연세로 50 (신촌동, 연세대학교)
(72) 발명자
**강성호**
서울특별시 마포구 양화로 45, 103동 1904호(서교동, 메세나폴리스)
**이하영**
서울특별시 서대문구 성산로16길 7-11, 202호(연희동)
(74) 대리인
**특허법인우인**

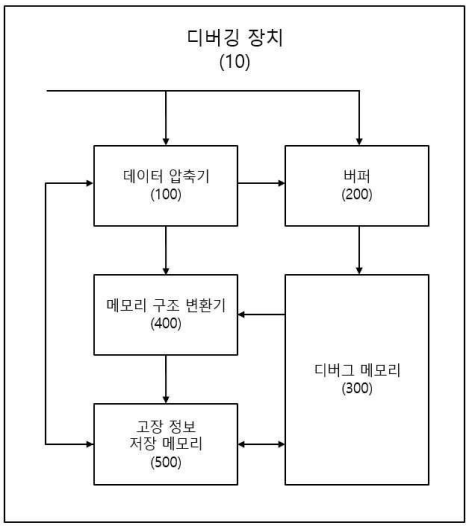
심사관 : 신우열

(54) 발명의 명칭 **BIRA CAM 구조를 재활용한 DRAM 기반 포스트 실리콘 디버깅 방법 및 장치**

## (57) 요 약

본 실시예들은 포스트 실리콘 디버깅 과정에서 온-칩 오류 감지를 위해 BISR(Built-In Self-Repair) 모듈의 일부 구성을 최적 데이터의 저장소 및 오류 검출의 비교기로 활용하고, 상하위 MISR(Multiple-Input Signature Register)을 적용하여 오류 의심 주기를 정확하게 검출하고, 오류 의심 디버그 데이터를 선별적으로 캡처한 후 디버그 메모리에 저장하는 방식을 통해 버퍼 크기, DRAM 사용량, 디버그 시간을 최소화할 수 있는 디버깅 장치 및 방법을 제공한다.

*대 표 도* - 도3



(52) CPC특허분류
*G11C 15/04* (2013.01)
*G11C 29/40* (2013.01)
*G11C 7/1039* (2013.01)

(56) 선행기술조사문헌
KR1020190062879 A
KR102135470 B1
US20090172483 A1
US20210174892 A1

이 발명을 지원한 국가연구개발사업
과제고유번호 1711131125
과제번호 2019R1A2C3011079
부처명 과학기술정보통신부
과제관리(전문)기관명 한국연구재단
연구사업명 중견연구자지원사업
연구과제명 인-메모리 컴퓨팅의 로버스트니스 향상을 위한 반도체 설계 기술
기 여 율 1/1
과제수행기관명 연세대학교 산학협력단
연구기간 2021.03.01 ~ 2022.02.28
공지예외적용 : 있음

# 명 세 서

## 청구범위

### 청구항 1

디버그 데이터를 수신하여 압축하는 데이터 압축기;

상기 디버그 데이터를 수신하여 저장하는 버퍼;

상기 버퍼로부터 상기 디버그 데이터를 수신하여 저장하고, 상기 디버그 데이터가 에러에 의하여 발생된 것인지 여부를 판단하기 위하여, 시뮬레이션을 통해 미리 산출된 데이터인 골든 시그니처(Golden Signature)를 저장하는 디버그 메모리; 및

상기 압축된 디버그 데이터와 상기 골든 시그니처를 비교하는 고장 정보 저장 메모리를 포함하는, 디버깅 장치.

### 청구항 2

제1항에 있어서,

메모리 수리 과정 및 포스트 실리콘 디버깅 과정을 구분하고, 상기 포스트 실리콘 디버깅 과정에서 상기 고장 정보 저장 메모리의 내부 구조를 변경하는 메모리 구조 변환기를 포함하는 것을 특징으로 하는 디버깅 장치.

### 청구항 3

제2항에 있어서,

상기 고장 정보 저장 메모리는 상기 메모리 수리 과정에서 사용되는 BIRA(Built-In Redundancy Analysis) 모듈의 CAM(Content Addressable Memory)을 적용하는 것을 특징으로 하는 디버깅 장치.

### 청구항 4

제1항에 있어서,

상기 고장 정보 저장 메모리는 상기 디버그 메모리로부터 상기 골든 시그니처를 수신하여 저장하고, 상기 데이터 압축기로부터 상기 압축된 디버그 데이터를 수신하여 상기 골든 시그니처와 비교하는 것을 특징으로 하는 디버깅 장치.

### 청구항 5

제1항에 있어서,

상기 고장 정보 저장 메모리는 상기 디버그 메모리의 상기 골든 시그니처를 디버그 간격에 따라 파이프라인을 수행하는 것을 특징으로 하는 디버깅 장치.

### 청구항 6

제1항에 있어서,

상기 데이터 압축기는 MISR(Multiple Input Signature Register)로 구현되어, 디버그 세션의 디버그 간격에 따른 디버그 데이터를 압축하여 상위 데이터(Parent Signature, PS)를 생성하고, 상기 디버그 세션에 따른 디버그 데이터를 압축하여 하위 데이터(Child Signature, CS)를 생성하는 것을 특징으로 하는 디버깅 장치.

### 청구항 7

제1항에 있어서,

상기 골든 시그니처는 이종의 데이터를 가지며,

디버그 세션의 디버그 간격의 오류에 관한 상위 골든 시그니처(Golden Parent Signature, GPS) 및 오류 주기에 관한 하위 골든 시그니처(Golden Child Signature, GCS)를 포함하는 것을 특징으로 하는 디버깅 장치.

**청구항 8**

제7항에 있어서,

상기 고장 정보 저장 메모리는 상기 디버그 세션의 디버그 간격의 오류를 감지하여 상위 태그(Parent Tag, PT)를 출력하고, 상기 오류 주기를 감지하여 하위 태그(Child Tag, CT)를 출력하는 것을 특징으로 하는 디버깅 장치.

**청구항 9**

제1항에 있어서,

상기 버퍼는,

상기 디버그 데이터를 캡처하는 트레이스 버퍼;

상기 디버그 데이터 중에서 오류가 있는 디버그 데이터를 선택하는 선택 캡처 모듈; 및

상기 오류가 있는 디버그 데이터를 캡처하는 쉐도우 버퍼를 포함하는 것을 특징으로 하는 디버깅 장치.

**청구항 10**

제9항에 있어서,

상기 선택 캡처 모듈은 상기 고장 정보 저장 메모리가 출력한 이종의 태그를 수신하여 캡처 활성 신호를 출력하고 상기 캡처 활성 신호를 쉐도우 버퍼로 전송하는 것을 특징으로 하는 디버깅 장치.

**청구항 11**

제9항에 있어서,

상기 트레이스 버퍼는 상기 디버그 데이터에 오류가 없으면 다음 디버그 데이터를 저장하고,

상기 쉐도우 버퍼는 상기 디버그 데이터에 오류가 없으면 바이패스하는 것을 특징으로 하는 디버깅 장치.

**청구항 12**

제9항에 있어서,

상기 디버그 메모리는 상기 쉐도우 버퍼에 캡처된 상기 오류가 있는 디버그 데이터를 저장하는 것을 특징으로 하는 디버깅 장치.

**청구항 13**

디버깅 장치에 의하여 수행되는 디버깅 방법에 있어서,

메모리 수리 과정에서 고장 정보 저장 메모리를 사용하는 단계;

디버그 메모리를 사용하는 포스트 실리콘 디버깅 과정에서 상기 고장 정보 저장 메모리의 내부 구조를 변경하는 단계; 및

상기 포스트 실리콘 디버깅 과정에서 상기 내부 구조가 변경된 고장 정보 저장 메모리를 다시 사용하는 단계를 포함하며,

상기 고장 정보 저장 메모리는 CAM(Content Addressable Memory)을 포함하고, 상기 고장 정보 저장 메모리의 내부 구조를 변경하는 것은, 메모리 복구 프로세스 동안 고장 저장을 위해 여러 부분으로 구성된 상기 CAM의 내부 구조를 변경하는 것을 특징으로 하는, 디버깅 방법.

**청구항 14**

제13항에 있어서,

상기 고장 정보 저장 메모리를 다시 사용하는 단계는,

상기 디버그 메모리의 데이터를 상기 내부 구조가 변경된 고장 정보 저장 메모리에 저장하고 상기 내부 구조가

변경된 고장 정보 저장 메모리를 통해 데이터 압축기의 데이터와 비교하는 것을 특징으로 하는 디버깅 방법.

**청구항 15**

제13항에 있어서,

상기 고장 정보 저장 메모리를 다시 사용하는 단계는,

상기 디버그 메모리의 데이터를 디버그 간격에 따라 상기 고장 정보 저장 메모리로 파이프라인하여 상기 고장 정보 저장 메모리를 파이프라인 버퍼로 사용하는 것을 특징으로 하는 디버깅 방법.

## 발명의 설명

### 기 술 분 야

[0001] 본 발명이 속하는 기술 분야는 BISR(Built-In Self-Repair)을 재사용하여 실리콘 디버그를 위한 온-칩 오류 검출을 수행하는 회로에 관한 것이다.

### 배 경 기 술

[0002] 이 부분에 기술된 내용은 단순히 본 실시예에 대한 배경 정보를 제공할 뿐 종래기술을 구성하는 것은 아니다.

[0003] 공정의 발달로 인하여 회로가 복잡해짐에 따라 이를 검증하는 과정 또한 복잡해지고 있으며, 제조 공정 전 설계 과정에서 발견하지 못하는 에러의 양이 점점 늘어나고 있다.

[0004] 이러한 에러들을 검출하기 위한 포스트 실리콘 과정이 필요하다. 포스트 실리콘 디버깅의 가장 큰 제약 사항은 칩 레벨에서 디버그를 진행하기 때문에 너무나도 많은 양의 데이터들을 실시간으로 관찰해야 한다는 점이다.

[0005] 이를 해결하기 위해 DRAM 기반 디버그 방법이 제안되었으나 기존의 DRAM을 사용하는 디버그 방법은 실제 디버깅 중인 회로의 속도와 DRAM의 읽기/쓰기 속도의 차이 때문에 회로 내 버퍼 등 추가적인 디버그 설계(Design for Debug, DfD) 구조를 요구하며 이는 결국 많은 디버그 비용을 요구하게 된다.

### 선행기술문헌

#### 특허문헌

[0006] (특허문헌 0001) 한국등록특허공보 제10-1958540호 (2019.03.08.)

## 발명의 내용

### 해결하려는 과제

[0007] 본 발명의 실시예들은 메모리 수리 과정에서 사용되는 BIRA(built-In Redundancy Analysis) 모듈 내 CAM(Content Addressable Memory) 구조를 재활용하여 포스트 실리콘 디버그 과정을 진행함으로써 디버그 시간, 버퍼 사이즈, DRAM 사용량을 최소화하는데 발명의 주된 목적이 있다.

[0008] 본 발명의 명시되지 않은 또 다른 목적들은 하기의 상세한 설명 및 그 효과로부터 용이하게 추론할 수 있는 범위 내에서 추가적으로 고려될 수 있다.

### 과제의 해결 수단

[0009] 본 실시예의 일 측면에 의하면, 디버그 데이터를 수신하여 압축하는 데이터 압축기; 상기 디버그 데이터를 수신하여 저장하는 버퍼; 상기 버퍼로부터 상기 디버그 데이터를 수신하여 저장하고, 압축된 최적 데이터를 저장하는 디버그 메모리; 및 상기 압축된 디버그 데이터와 상기 압축된 최적 데이터를 비교하는 고장 정보 저장 메모리를 포함하는 디버깅 장치를 제공한다.

[0010] 상기 디버깅 장치는 메모리 수리 과정 및 포스트 실리콘 디버깅 과정을 구분하고, 상기 포스트 실리콘 디버깅 과정에서 상기 고장 정보 저장 메모리의 내부 구조를 변경하는 메모리 구조 변환기를 포함할 수 있다.

[0011] 상기 고장 정보 저장 메모리는 상기 메모리 수리 과정에서 사용되는 BIRA(Built-In Redundancy Analysis) 모듈의 CAM(Content Addressable Memory)을 적용할 수 있다.

[0012] 상기 고장 정보 저장 메모리는 상기 디버그 메모리로부터 상기 압축된 최적 데이터를 수신하여 저장하고, 상기 데이터 압축기로부터 상기 압축된 디버그 데이터를 수신하여 상기 압축된 최적 데이터와 비교할 수 있다.

[0013] 상기 고장 정보 저장 메모리는 상기 디버그 메모리의 상기 압축된 최적 데이터를 디버그 간격에 따라 파이프라인을 수행할 수 있다.

[0014] 상기 데이터 압축기는 MISR(Multiple Input Signature Register)로 구현되어, 디버그 세션의 디버그 간격에 따른 디버그 데이터를 압축하여 상위 데이터(Parent Signature, PS)를 생성하고, 상기 디버그 세션에 따른 디버그 데이터를 압축하여 하위 데이터(Child Signature, CS)를 생성할 수 있다.

[0015] 상기 최적 데이터는 이종의 데이터를 갖고, 상기 고장 정보 저장 메모리는 상기 디버그 세션의 디버그 간격의 오류에 관한 상위 최적 데이터(Golden Parent Signature, GPS) 및 오류 주기에 관한 하위 최적 데이터(Golden Child Signature, GCS)를 저장할 수 있다.

[0016] 상기 고장 정보 저장 메모리는 상기 디버그 세션의 디버그 간격의 오류를 감지하여 상위 태그(Parent Tag, PT)를 출력하고, 상기 오류 주기를 감지하여 하위 태그(Child Tag, CT)를 출력할 수 있다.

[0017] 상기 버퍼는, 상기 디버그 데이터를 캡처하는 트레이스 버퍼; 상기 디버그 데이터 중에서 오류가 있는 디버그 데이터를 선택하는 선택 캡처 모듈; 및 상기 오류가 있는 디버그 데이터를 캡처하는 쉐도우 버퍼를 포함할 수 있다.

[0018] 상기 선택 캡처 모듈은 상기 고장 정보 저장 메모리가 출력한 이종의 태그를 수신하여 캡처 활성 신호를 출력하고 상기 캡처 활성 신호를 쉐도우 버퍼로 전송할 수 있다.

[0019] 상기 트레이스 버퍼는 상기 디버그 데이터에 오류가 없으면 다음 디버그 데이터를 저장하고, 상기 쉐도우 버퍼는 상기 디버그 데이터에 오류가 없으면 바이패스할 수 있다.

[0020] 상기 디버그 메모리는 상기 쉐도우 버퍼에 캡처된 상기 오류가 있는 디버그 데이터를 저장할 수 있다.

[0021] 본 실시예의 다른 측면에 의하면, 디버깅 방법에 있어서, 메모리 수리 과정에서 고장 정보 저장 메모리를 사용하는 단계; 디버그 메모리를 사용하는 포스트 실리콘 디버깅 과정에서 상기 고장 정보 저장 메모리의 내부 구조를 변경하는 단계; 및 상기 포스트 실리콘 디버깅 과정에서 상기 내부 구조가 변경된 고장 정보 저장 메모리를 다시 사용하는 단계를 포함하는 디버깅 방법을 제공한다.

[0022] 상기 고장 정보 저장 메모리를 다시 사용하는 단계는, 상기 디버그 메모리의 데이터를 상기 내부 구조가 변경된 고장 정보 저장 메모리에 저장하고 상기 내부 구조가 변경된 고장 정보 저장 메모리를 통해 데이터 압축기의 데이터와 비교할 수 있다.

[0023] 상기 고장 정보 저장 메모리를 다시 사용하는 단계는, 상기 디버그 메모리의 데이터를 디버그 간격에 따라 상기 고장 정보 저장 메모리로 파이프라인하여 상기 고장 정보 저장 메모리를 파이프라인 버퍼로 사용할 수 있다.

## 발명의 효과

[0024] 이상에서 설명한 바와 같이 본 발명의 실시예들에 의하면, 온-칩 오류 감지를 위해 BISR(Built-In Self-Repair) 모듈의 일부 구성을 최적 데이터의 저장소 및 오류 검출의 비교기로 활용하고, 상하위 MISR(Multiple-Input Signature Register)을 적용하여 오류 의심 주기를 정확하게 검출하고, 오류 의심 디버그 데이터를 선별적으로 캡처한 후 디버그 메모리에 저장하는 방식을 통해 버퍼 크기, DRAM 사용량, 디버그 시간을 최소화할 수 있는 효과가 있다.

[0025] 여기에서 명시적으로 언급되지 않은 효과라 하더라도, 본 발명의 기술적 특징에 의해 기대되는 이하의 명세서에서 기재된 효과 및 그 잠정적인 효과는 본 발명의 명세서에 기재된 것과 같이 취급된다.

## 도면의 간단한 설명

[0026] 도 1은 3차원 집적회로의 BISR을 예시한 도면이다.

도 2는 본 발명의 일 실시예에 따른 디버깅 방법을 예시한 흐름도이다.

도 3은 본 발명의 다른 실시예에 따른 디버깅 장치를 예시한 블록도이다.

도 4는 본 발명의 다른 실시예에 따른 디버깅 장치의 동작을 신호 흐름을 예시한 도면이다.

도 5 내지 도 8은 본 발명의 다른 실시예에 따른 디버깅 장치의 오류 검출 동작을 예시한 도면이다.

도 9 및 도 10은 본 발명의 다른 실시예에 따른 디버깅 장치의 선택 캡처 모듈 및 동작을 예시한 도면이다.

도 11은 본 발명의 다른 실시예에 따른 디버깅 장치의 고장 정보 저장 메모리의 파이프라인 동작을 예시한 도면이다.

도 12는 본 발명의 다른 실시예에 따른 디버깅 장치의 메모리 구조 변환기를 예시한 도면이다.

도 13은 본 발명의 다른 실시예에 따른 디버깅 장치의 고장 정보 저장 메모리의 구조를 예시한 도면이다.

## *발명을 실시하기 위한 구체적인 내용*

[0027] 이하, 본 발명을 설명함에 있어서 관련된 공지기능에 대하여 이 분야의 기술자에게 자명한 사항으로서 본 발명의 요지를 불필요하게 흐릴 수 있다고 판단되는 경우에는 그 상세한 설명을 생략하고, 본 발명의 일부 실시예들을 예시적인 도면을 통해 상세하게 설명한다.

[0028] 도 1은 3차원 집적회로의 BISR을 예시한 도면이다.

[0029] 도 1에 도시된 예비 자원을 이용한 메모리 수리(Built-In Self-Repair, BISR) 모듈은 수율 향상과 테스트 비용 절감을 위해 사용된다. BIST(Built-In Self-Test) 모듈은 테스트 대상 회로를 테스트하기 위한 패턴을 자동적으로 생성하여 테스트 대상 회로에 자동적으로 입력시킨 후, 테스트 패턴과 테스트 후 결과 값이 사용자가 원하는 결과 값과 비교하여 회로의 고장을 판단한다. BIRA(Built-In Redundancy Analysis) 모듈은 BIST 모듈이 메모리의 고장 부분을 찾아낸 후, 고장 정보를 이용하여 여분의 메모리에 어떻게 재배치할 것인지 판단한다. BISR는 재배치 정보를 이용하여 사용자가 고장난 주소로 데이터를 읽거나 쓰기를 할 때 재배치 정보를 이용하여 여분의 메모리로 대체함으로써, 사용자는 고장이 없는 메모리를 사용하는 것과 같이 메모리를 사용할 수 있다.

[0030] 메모리는 단층형 메모리 또는 적층형 메모리 장치로 구현될 수 있다. 적층형 메모리 장치는 하나 이상의 결합된 메모리 다이 층들, 메모리 패키지들, 또는 다른 메모리 요소들을 포함하는 3차원 컴퓨터 메모리를 의미한다. 적층형 메모리 장치는 복수의 메모리 층 및 시스템 층을 포함하며, 기판 위에 구현될 수 있다.

[0031] 적층형 메모리 장치는 수직 적층형 또는 수평(예컨대, 사이드-바이-사이드) 적층형이거나, 서로 결합되어 있는 메모리 요소들을 포함할 수 있다. 복수의 메모리 층은 DRAM(Dynamic Random Access Memory)으로 구현될 수 있으나, 이에 한정되는 것은 아니다. 적층형 DRAM 표준(Wide IO)의 출현으로 DRAM 웨이퍼는 메모리 스택을 가진 동일 패키지에 시스템 온 칩(SoC) 웨이퍼와 같은 시스템 요소와 함께 적층될 수 있다. 적층형 메모리 장치에서는 DRAM의 제조 회사에 따라 장치마다 메모리 층들이 변할 수 있다.

[0032] 복수의 메모리 층은 실리콘 관통전극을 통하여 상호 연결될 수 있다. 적층형 메모리는 실리콘 관통전극(Through Silicon Via, TSV) 제조 기법들을 이용하며, 메모리 스택을 통한 신호 경로들을 제공하기 위해 실리콘 다이들을 통하여 비아들이 생성된다. 실리콘 관통전극으로 연결할 때, 최상부(또는 최외곽) 메모리 다이 층은 제외될 수 있다.

[0033] 시스템 층은 중앙 처리 유닛(Central Processing Unit, CPU), 메모리 컨트롤러, 및 기타 관련 시스템 요소들과 같은 요소들을 포함할 수 있다. 시스템 층은 시스템 온 칩(SoC)을 포함할 수 있다. 로직 칩은 애플리케이션 프로세서 또는 그래픽 처리 장치(Graphics Processing Unit, GPU)일 수 있다.

[0034] 메모리 수리는 포괄적이고 메모리의 제조 회사에 의해 시행되는 임의의 메모리 포맷에 적용될 수 있다. 일부 실시예들에서 메모리 장치는 CAM 등과 같은 별개의 메모리에 저장하거나 메모리의 결함 있는 부분들의 주소들을 메모리의 양호한 부분들로 변환함으로써 결함 있는 요소들에 대한 여분(Redundancy)을 제공한다. 일부 실시예들에서 메모리의 결함 있는 메모리의 여분의 행들, 열들, 또는 블록들이 소정의 종래의 DRAM들에서 구현될 수 있다.

[0035] 여분 메모리는 기본 메모리에서 오류가 발생한 셀을 대체한다. 여분 메모리는 셀 단위 또는 라인 단위로 오류가 발생한 하나 이상의 셀을 대체할 수 있다. 여분 메모리는 복수의 메모리 층의 행 또는 열과 연결된 라인을 교체하는 방식으로 수리할 수 있다. 여분 메모리는 기본 메모리의 행 또는 열과 연결된 라인을 교체하는 방식으로 수리할 수 있다. 여분 메모리는 물리적인 대체를 이용할 수 있고, 메모리 층에 해당하는 행렬처럼 동작하는 논

리적인 대체를 수행할 수도 있다.

[0036] 3D-IC의 복잡성이 증가함에 따라 구성 요소를 완전히 검증하고 또는 검증하는 것과 프리 실리콘 검증 및 제조 테스트 중에 감지되지 않는 오류 수를 줄이는 것은 쉽지 않으므로, 포스트 실리콘 디버그는 회로 구현에서 중요한 단계에 해당한다.

[0037] 포스트 실리콘 디버그의 주요 목적은 첫번째 실리콘에서 논리적, 타이밍 및 전기적 오류와 같은 오류를 감지하여 실리콘 재 스핀으로 인한 비용 증가를 방지한다.

[0038] 회로의 가능한 최대 내부 상태를 관찰하기 위해 스캔 기반 실리콘 디버깅 방법이 도입되었으나 스캔 덤프를 수행하려면 회로 작업을 일시 중지해야 한다. 오류는 수천 클록 사이클 동안 회로 상태에 나타날 수 있기 때문에 이러한 런-스톱 디버깅 방법에서 이러한 오류를 감지하는 것은 곤란하다.

[0039] 이를 극복하기 위해서 실시간 트레이스 버퍼 기반 실리콘 디버그 방식은 작업을 중단하지 않고 실시간 디버그 데이터를 관찰한다. 트리거 포인트, 실시간 디버그 데이터 등을 관리하기 위해 추가 온칩 버퍼와 임베디드 로직 분석기가 필요하다. 그러나 주요 과제는 트레이스 버퍼 크기가 제한되어 있고 DfD 하드웨어 오버 헤드가 발생하기 때문에 포스트 실리콘 디버그의 목적을 달성하는데 제한적이다. 트레이스 버퍼 기반 실리콘 디버그 방식은 긴 디버깅 시간이 필요하다.

[0040] 외부 DRAM을 사용하는 디버깅 방식은 MISR을 사용하여 오류 디버그 간격을 감지하고 트레이스 및 쉐도우 버퍼를 통해 오류가 의심되는 디버그 데이터 덤프를 DRAM에 저장한다. 여러 개의 동일 코어에 대한 DRAM 기반 실리콘 디버그 방법은 다른 코어의 오류 데이터에 대해 코어의 오류없는 간격 데이터를 골든 데이터(최적 데이터)로 적용할 수 있다는 사실을 활용한다. 이러한 DRAM 기반 디버깅 방법은 트레이스 버퍼 기반 방법의 한계를 극복한다.

[0041] 그러나 DRAM과 DfD 간의 통신으로 인해 버퍼 및 DRAM 사용과 같은 상당한 디버그 리소스가 여전히 필요하다.

[0042] 실리콘 디버깅은 시뮬레이션을 통해 미리 산출된 골든 데이터를 사용하여 공간(오류 논리) 및 시간(디버그 발생한 정확한 클록 주기) 정보 측면에서 근본 원인을 최대한 빠르게 감지하는 것을 목적으로 한다.

[0043] 본 실시예에 따른 디버깅 방법 및 장치는 DRAM 기반 DfD을 기반으로 디버그 리소스 비용을 줄이기 위해 메모리 수리에서 사용된 BISR을 재사용한다. 상하위 MISR을 사용하는 DRAM 기반 온칩 오류 감지 방식으로 미리 계산된 골든 데이터가 재사용된 BISR에 저장되므로 오류가 있는 디버그 데이터를 보다 정확하게 감지한다. 선택적 디버그 데이터 캡처 및 저장 방법을 적용한다. BISR의 일부 구성을 버퍼로 사용하여 발생하는 DfD와 DRAM 간의 통신 문제를 극복하고, 버퍼 크기, DRAM 사용량 및 총 디버그 시간을 감소시킨다.

[0044] 도 2는 본 발명의 일 실시예에 따른 디버깅 방법을 예시한 흐름도이다.

[0045] 디버깅 방법은 디버깅 장치에 의해 수행될 수 있다.

[0046] 디버깅 방법은 메모리 수리 과정에서 고장 정보 저장 메모리를 사용하는 단계(S11), 디버그 메모리를 사용하는 포스트 실리콘 디버깅 과정에서 고장 정보 저장 메모리의 내부 구조를 변경하는 단계(S12), 및 포스트 실리콘 디버깅 과정에서 내부 구조가 변경된 고장 정보 저장 메모리를 다시 사용하는 단계(S13)를 포함한다.

[0047] 고장 정보 저장 메모리를 다시 사용하는 단계(S13)는, 디버그 메모리의 데이터를 내부 구조가 변경된 고장 정보 저장 메모리에 저장하고, 내부 구조가 변경된 고장 정보 저장 메모리를 통해 데이터 압축기의 데이터와 비교할 수 있다.

[0048] 고장 정보 저장 메모리를 다시 사용하는 단계(S13)는, 디버그 메모리의 데이터를 디버그 간격에 따라 상기 고장 정보 저장 메모리로 파이프라인하여 고장 정보 저장 메모리를 파이프라인 버퍼로 사용할 수 있다.

[0049] 도 3은 본 발명의 다른 실시예에 따른 디버깅 장치를 예시한 블록도이고, 도 4는 본 발명의 다른 실시예에 따른 디버깅 장치의 동작을 신호 흐름을 예시한 도면이다.

[0050] 디버깅 장치(10)는 데이터 압축기(100), 버퍼(200), 디버그 메모리(300), 메모리 구조 변환기(400), 고장 정보 저장 메모리(500)를 포함한다.

[0051] 데이터 압축기(100) 및 버퍼(200)에 디버그 데이터가 입력된다. 데이터 압축기(100)는 버퍼(200), 메모리 구조 변환기(400), 고장 정보 저장 메모리(500)에 연결된다. 버퍼(200)는 데이터 압축기(100), 디버그 메모리(300)에 연결된다. 디버그 메모리(300)는 버퍼(200), 메모리 구조 변환기(400), 고장 정보 저장 메모리(500)에

연결된다. 메모리 구조 변환기(400)는 데이터 압축기(100), 디버그 메모리(300), 고장 정보 저장 메모리(500)에 연결된다. 고장 정보 저장 메모리(500)는 데이터 압축기(100), 메모리 구조 변환기(400), 디버그 메모리(300)에 연결된다.

[0052] 데이터 압축기(100)는 디버그 데이터를 수신하여 압축한다. 데이터 압축기(100)는 MISR(Multiple Input Signature Register)로 구현되어, 디버그 세션의 디버그 간격에 따른 디버그 데이터를 압축하여 상위 데이터(Parent Signature, PS)를 생성하고, 디버그 세션에 따른 디버그 데이터를 압축하여 하위 데이터(Child Signature, CS)를 생성할 수 있다.

[0053] 버퍼(200)는 디버그 데이터를 수신하여 저장한다. 버퍼(200)는 디버그 데이터를 캡처하는 트레이스 버퍼, 디버그 데이터 중에서 오류가 있는 디버그 데이터를 선택하는 선택 캡처 모듈, 및 오류가 있는 디버그 데이터를 캡처하는 쉐도우 버퍼를 포함할 수 있다.

[0054] 선택 캡처 모듈은 고장 정보 저장 메모리가 출력한 이종의 태그를 수신하여 캡처 활성 신호를 출력하고 캡처 활성 신호를 쉐도우 버퍼로 전송할 수 있다. 트레이스 버퍼는 디버그 데이터에 오류가 없으면 다음 디버그 데이터를 저장하고, 쉐도우 버퍼는 디버그 데이터에 오류가 없으면 바이패스할 수 있다.

[0055] 디버그 메모리(300)는 버퍼로부터 디버그 데이터를 수신하여 저장하고, 압축된 최적 데이터를 저장한다. 디버그 메모리(300)는 쉐도우 버퍼에 캡처된 오류가 있는 디버그 데이터를 저장할 수 있다.

[0056] 메모리 구조 변환기(400)는 메모리 수리 과정 및 포스트 실리콘 디버깅 과정을 구분하고, 포스트 실리콘 디버깅 과정에서 고장 정보 저장 메모리의 내부 구조를 변경한다.

[0057] 고장 정보 저장 메모리(500)는 압축된 디버그 데이터와 압축된 최적 데이터를 비교한다. 고장 정보 저장 메모리(500)는 메모리 수리 과정에서 사용되는 BIRA(Built-In Redundancy Analysis) 모듈의 CAM(Content Addressable Memory)을 적용할 수 있다. 고장 정보 저장 메모리(500)는 디버그 메모리로부터 압축된 최적 데이터를 수신하여 저장하고, 데이터 압축기로부터 압축된 디버그 데이터를 수신하여 압축된 최적 데이터와 비교할 수 있다. 고장 정보 저장 메모리(500)는 디버그 메모리의 압축된 최적 데이터를 디버그 간격에 따라 파이프라인을 수행할 수 있다.

[0058] 최적 데이터는 이종의 데이터를 갖고, 고장 정보 저장 메모리(500)는 디버그 세션의 디버그 간격의 오류에 관한 상위 최적 데이터(Golden Parent Signature, GPS) 및 오류 주기에 관한 하위 최적 데이터(Golden Child Signature, GCS)를 저장할 수 있다.

[0059] 고장 정보 저장 메모리(500)는 디버그 세션의 디버그 간격의 오류를 감지하여 상위 태그(Parent Tag, PT)를 출력하고, 오류 주기를 감지하여 하위 태그(Child Tag, CT)를 출력할 수 있다.

[0060] 디버깅을 진행하기 위해서는 JTAG(Joint Test Action Group)과 같은 디버깅 포트를 이용하여 디버깅을 위한 설정을 해야 한다. 디버깅 동안의 데이터를 압축하기 위해 MISR(Multiple Input Signature Register)이 사용된다. 디버깅을 진행하면서 해당 디버그 데이터가 실제 에러가 있는지 검출하기 위해 디버그 전에 워크스테이션 단계에서 행동 시뮬레이션(behavioral simulation) 등을 통해 골든 값을 먼저 계산한다. 디버깅이 시작된 후 해당 골든 값들을 압축한 골든 시그니처 데이터를 DRAM에 저장하여 해당 구간이 에러인 경우 버퍼에 저장된 디버그 데이터를 DRAM에 저장하고 그렇지 않은 경우 다음 디버그 데이터를 버퍼에 저장하는 방법으로 디버그를 진행함으로써 에러인 구간을 선별적으로 검출해 낼 수 있다.

[0061] 도 5 내지 도 8은 본 발명의 다른 실시예에 따른 디버깅 장치의 오류 검출 동작을 예시한 도면이다.

[0062] DI(Debug Interval)와 DS(Debug Session)는 디버그 간격과 디버그 세션을 각각 나타낸다. 먼저 부모(Parent) MISR 시그니처(Signature)는 에러가 존재하는 영역을 감지하고, 동시에 자식(Child) MISR 시그니처(Signature)를 사용하여 각 세션에서 잘못된 디버그 주기를 감지하기 위해 디버그 데이터가 주기적으로 압축된다. 이러한 시그니처들은 CAM에 저장된 골든 시그니처들과 비교되며, 디버그 데이터가 트레이스 버퍼에 캡처된 후 잘못된 디버그 데이터가 쉐도우 버퍼에 선택적으로 캡처되고 DRAM에 저장된다. 이 프로세스는 버퍼 크기 및 DRAM 사용량과 같은 디버그 리소스 사용을 줄일 수 있다. DRAM 사용과 밀접한 관련이 있는 총 디버그 시간 또한 이를 이용해 줄일 수 있다.

[0063] 디버그 실험 전에 골든 서명은 DRAM에서 전송 된 후 CAM에 저장된다. 골든 시그니처 주기를 스케줄링하기 위해 CAM 크기를 계산하여 트레이스 버퍼 깊이(M), 사이클의 총 디버그 세션 수(DS), 사이클의 총 디버그 간격 수

(DI)를 결정한다.

[0064] 기존 DRAM 기반 방식에서는 DRAM과 CUD(Circuit Under Debug) 간의 속도로 인해 DI가 메모리 액세스 지연 시간보다 크게 설정된다. DS와 M이 DI와 유사하게 결정된다.

[0065] 본 실시예는 기존 방식과 달리 DRAM과 DfD 사이의 통신을 고려하여 DRAM과 통신시 CAM을 버퍼로 사용하므로 DI를 줄일 수 있다. M도 줄일 수 있다.

[0066] 본 실시예는 두 가지 유형의 골든 시그니처를 생성한다. 제1 유형의 골든 시그니처의 경우에 골든 디버그 데이터를 순차적으로 압축하여 디버그 세션의 해당 디버그 간격이 잘못되었는지 여부를 감지한다. 이러한 골든 시그니처를 GPS(Golden Parent Signature)라고 한다. 제2 유형의 골든 시그니처의 경우에 골든 디버그 데이터는 디버그 세션 중에 특정 오류주기를 감지하기 위해 주기적으로 압축된다. 이러한 골든 시그니처를 GCS(Golden Child Signature)라고 한다. 두 가지 골든 시그니처 GPS 및 GCS는 디버그 테스트 중에 DRAM 및 CAM에 저장되어 온칩 오류 감지를 수행한다.

[0067] 디버그 프로세스가 시작된 후 생성 된 디버그 데이터는 DI 사이클 동안 트레이스 버퍼에서 추적된다. 디버그 데이터가 추적 버퍼에 캡처되는 동안 각 DI 사이클의 디버그 간격 데이터는 MISR에 의해 압축되어 오류 여부를 감지한다.

[0068] 디버그 간격을 감지하는 MISR을 상위(부모) MISR이라고 상위(부모) MISR에서 생성한 시그니처를 상위(부모) 시그니처(PS)라고 한다. 각 DI 사이클에서 PS는 사이클에서 CAM을 사용하여 해당 GPS와 비교되며 비교 결과는 비트로 획득된다. 이러한 비트를 상위(부모) 태그 비트(PT)라고 한다.

[0069] PT가 1이면 트레이스 버퍼의 디버그 데이터가 쉐도우 버퍼에 의해 캡처된다. 그렇지 않은 경우 바이패스되고 다음 디버그 데이터가 트레이스 버퍼에서 덮어쓴다.

[0070] 오류 주기를 감지하기 위해 디버그 데이터는 PS 생성 중에 다른 MISR에 의해 주기적으로 압축된다. 이러한 오류 주기를 감지하기 위한 MISR을 하위(자식) MISR이라고하고 하위(자식) MISR에 의해 생성된 시그니처를 하위(자식) 시그니처(CS)라고 한다. 이러한 CS는 DS의 마지막 DI 동안 완전히 생성된다.

[0071] 사이클의 순서에 따라 CAM을 사용하여 해당 GCS와 비교되며 비교 결과는 비트로 얻어진다. 이러한 비트를 하위(자식) 태그 비트 (CT)라고 한다.

[0072] 도 6 내지 도 8을 참조하면, 각 디버그 사례에 대한 온칩 오류 감지 방법의 예를 보여준다. DS(Debug session in cycles)는 12, DI(Debug interval in cycles)와 M(Trace buffer depth)은 4, I(Number of debug intervals for DS cycles)은 3, SE(Number of segments for a debug interval)는 4인 간단한 디버그 사례로 가정하여 설명한다. 이 경우 7 번째 사이클이 오류 사이클이라고 가정한다.

[0073] 디버그 프로세스가 시작된 후 디버그 데이터가 트레이스 버퍼에 캡처된다. 동시에 상위 MISR을 사용하여 압축된다. $PS_1$이 생성된 후 CAM에서 $GPS_1$과 비교되고 $PT_1$이 생성된다. 디버그 간격은 오류가 없으므로 $PT_1$은 0이다. 첫 번째 DI(1-4 번째 사이클)의 디버그 데이터가 바이패스된다. 디버그 데이터는 도 6과 같이 CS를 생성하기 위해 자식 MISR에 의해 주기적으로 압축된다. $PT_1$이 생성된 후 후속 디버그 데이터는 트레이스 버퍼에서 덮어 쓰여지고 유사하게 압축된다.

[0074] $PT_2$는 해당 디버그 간격이 오류가 의심되기 때문에 1이다. 결과적으로 도 7과 같이 두 번째 DI 사이클(5-8 사이클)의 디버그 데이터가 쉐도우 버퍼에 캡처된다.

[0075] 최종 DI(사이클 9-12)에서는 도 8과 같이 CS와 $PS_3$가 생성된다. 9 번째 사이클에서는 $CS_1$이 생성되고 CAM을 사용하여 $GCS_1$과 비교된다. $CT_1$이 생성된다. $CT_1$은 첫 번째, 다섯 번째, 아홉 번째 사이클이 오류 사이클이 아니기 때문에 0이다. 이러한 방식으로 PT와 CT는 12주기에서 완전히 생성됩니다.

[0076] 7 번째 사이클이 에러 사이클인 것을 감지할 수 있다. 후속 디버그 데이터가 칩에서 유사하게 감지된다.

[0077] 도 9 및 도 10은 본 발명의 다른 실시예에 따른 디버깅 장치의 선택 캡처 모듈 및 동작을 예시한 도면이다.

[0078] 도 9에서는 DRAM 사용량을 줄이기 위한 방법의 일종으로 에러가 포함된 데이터만 선별하는 하드웨어의 구조를 보여준다. CE는 쉐도우 버퍼에 대한 캡처 활성화 신호이며, CEi 및 CEij 각각은 PTi 및 CTi를 사용하여 생성된다. 또한 CEi 및 CEij는 쉐도우 버퍼 및 DRAM에서 캡처된 디버그 데이터를 선택하는 데 사용된다.

[0079] 도 10에서는 DRAM 사용량을 줄이기 위한 방법의 일종으로 에러가 포함된 데이터만 선별하는 프로세스의 예시를 보여준다. $CE_2$는 1인 반면 $CE_1$과 $CE_3$은 임피던스를 나타낸다. 즉, 두 번째 DI의 디버그 데이터만 쉐도우 버퍼에 캡처되는 반면 다른 사이클의 디버그 데이터는 바이패스된다. 디버그 데이터가 쉐도우 버퍼에 캡처된 후 DRAM 컨트롤러 및 $CE_{ij}$ 신호를 사용하여 에서 데이터가 DRAM에 캡처된다. 이 경우 $CE_{23}$은 1이고 나머지($CE_{21}$, $CE_{22}$, $CE_{24}$)는 0이므로 7 번째 사이클의 디버그 데이터만 DRAM에 저장된다.

[0080] 디버그 실험이 끝나면 캡처된 디버그 데이터가 외부 워크 스테이션으로 언로드되고 온칩 오류 감지 결과로 디버그 후 분석이 수행됩니다. 디버그 프로세스가 완료되면 $T_{CUD}$(Time for silicon debug for the CUD) 및 $DU_{CUD}$(DRAM usage for the CUD)가 계산되어 알고리즘이 끝날때 반환된다. 그런 다음 후속 디버그 계획이 유사하게 수행된다.

[0081] 도 11은 본 발명의 다른 실시예에 따른 디버깅 장치의 고장 정보 저장 메모리의 파이프라인 동작을 예시한 도면이다.

[0082] DRAM의 골든 시그니처를 BISR의 CAM으로 파이프라인을 수행할 수 있다. CAM은 일반적으로 메모리 복구 프로세스 동안 효율적인 메모리 오류 저장 프로세스를 위해 여러 부분으로 구성된다.

[0083] 도 11에서 I는 3이고 SE는 4로 가정한다. GPS는 각 디버그 간격이 시작될 때 상위(부모) CAM이라고 할 수 있는 첫 번째 부분으로 파이프라인된다. CS가 마지막 디버그 간격의 GCS와 비교되기 때문에 GCS는 중간 디버그 간격 동안 하위(자식) CAM이라고 할 수 있는 두 번째 부분으로 파이프라인된다.

[0084] 파이프라인을 사용하면 메모리 액세스 대기 시간을 유지하면서 DI를 줄일 수 있다. 결과적으로 DI가 감소하기 때문에 M을 줄일 수 있다.

[0085] DI 과정에서 생성된 디버그 데이터는 트레이스 버퍼에 캡처되고, 잘못된 간격 디버그 데이터는 PT를 사용하여 쉐도우 버퍼에 선택적으로 캡처된다. 디버그 데이터는 주기적으로 압축되어 하위 MISR을 통해 각 DI의 오류 주기를 감지한다.

[0086] DS의 최종 DI 동안 CT가 생성되고 오류 데이터를 보다 정확하게 감지할 수 있다.

[0087] 도 12는 본 발명의 다른 실시예에 따른 디버깅 장치의 메모리 구조 변환기를 예시한 도면이다. BIRA 내 CAM을 디버깅에 재활용하기 위해 컨트롤러의 구조를 보여준다.

[0088] CAM을 디버깅에서 활용하기 위하여 상태를 변경하기 위해 모드 레지스터가 존재한다. 메모리 수리 프로세스는 칩 유효성 검사에서 실리콘 디버그 프로세스 이전에 수행되기 때문에 모드 레지스터는 CAM의 현재 상태가 처음에는 메모리 수리를 위한 상태가 되도록 구성한다. 메모리 수리 프로세스 동안 CAM 및 메모리 고장 분석기는 기존 BIRA 컨트롤러에 의해 제어되며, 수리 프로세스 후 모드 레지스터는 디버그 단계에서 CAM의 상태를 변경한다.

[0089] MISR은 디버깅 도중 온 칩 오류 감지를 위한 실시간 시그니처를 생성하는 데 필요하다. 디버그 테스트 중에 온 칩 오류 감지를 수행하려면 구성 단계 중에 FPGA 보드에서 동작 시뮬레이션 또는 에뮬레이션을 수행하여 골든 MISR 시그니처를 생성해야 한다.

[0090] 도 13은 본 발명의 다른 실시예에 따른 디버깅 장치의 고장 정보 저장 메모리의 구조를 예시한 도면이다.

[0091] 여러 가지 BIRA들 중에서도 대표라고 할 수 있는 BRANCH의 CAM을 예시로 하며, 메모리 수리 단계에서는 메모리 수리를 위해 CAM 공간을 이용하다가 디버깅 단계에서 CAM 공간을 변경시켜 디버깅에 적합하게 변형한다.

[0092] 기존 CAM 자체가 복수 개의 부분으로 구분되어 있어서 복수 개의 부분으로 구분된 것을 디버깅에서 그대로 차용하여 디버깅 과정에서 CAM을 파이프라인 버퍼로 이용할 수 있다. 특정 시점부터 BIRA는 대부분이 CAM 회로를 사용하고 있고, 도 13에서 예시를 든 CAM 구조와 유사한 구조 및 특징을 모두 가지고 있기 때문에 대부분의 BIRA에 대해 본 발명의 적용이 가능하다.

[0093] BISR 및 DfD 작업은 JTAG와 같은 외부 포트를 통해 구성된다. 각 프로세스를 처리하려면 도 12에서 설명된 추가 컨트롤러가 필요하다. 현재 상태를 확인하기 위해 모드 레지스터가 구현된다.

[0094] 메모리 복구 프로세스는 일반적으로 칩 유효성 검사에서 실리콘 디버그 프로세스 이전에 수행되기 때문에 모드 레지스터는 현재 상태가 처음에는 복구 단계가 되도록 구성된다. 메모리 복구 프로세스 동안 CAM 및 리던던시

분석기는 기존 BISR 컨트롤러에 의해 제어된다. 복구 프로세스 후 모드 레지스터는 디버그 단계에서 현재 상태를 변경한다. 실리콘 디버그 중에 MISR은 온칩 오류 감지를 위한 실시간 시그니처를 생성하는 데 필요하다.

[0095] 디버그 테스트 중에 온칩 오류 감지를 수행하려면 구성 단계 중에 FPGA 보드에서 동작 시뮬레이션 또는 에뮬레이션을 수행하여 골든 MISR 서명을 생성해야 한다.

[0096] 각 모드에서 CAM의 주소 크기는 카운터 레지스터를 사용하여 계산된다. 주소 생성기는 CAM 데이터의 정확한 크기를 조정하는 데 사용된다. 이러한 전체 작업은 FSM(Finite State Machine)에 의해 제어된다. DRAM의 골든 시그니처는 디버그 프로세스 동안 CAM에 저장되어야 하므로 DRAM 컨트롤러와 FSM 간의 통신에 DfD 컨트롤러가 사용된다.

[0097] FSM은 디버그 주기를 계산하고 CAM과 DRAM 간의 데이터 전송을 조절한다. 골든 시그니처에 CAM을 적절하게 재사용하려면 CAM의 크기에 따라 골든 시그니처를 생성하는 것이 중요하다.

[0098] BRANCH의 메모리 오류 저장 로직은 CAM으로 구성되며 Rs 스페어 행과 Cs 스페어 열이 있는 J X K 메모리 블록에 사용된다.

[0099] 부모 CAM의 크기는 $(3+\log_2(JK))(Rs+Cs)$이고 자식 CAM의 크기는 $(2+\log_2\{(\max(J,K))(Rs+Cs)\})(Rs(Cs-1)+Cs(Rs-1))$. 실시예의 경우 골든 부모(자식) 시그니처의 필수 크기는 $1+L+\log_2 I(SE)$이다.

*수학식 1*

[0100]
$$P_{max} = \frac{(3+log_2(JK))(R_s + C_s)}{1 + L + log_2 I}$$

*수학식 2*

[0101]
$$C_{max} = \frac{(2+log_2\{(\max(J, K))(Rs+Cs)\})(R_s(C_s-1)+C_s(R_s-1))}{1 + L + log_2 SE}$$

[0102] p와 c는 Pmax와 Cmax가 디버그 세션 동안 골든 시그니처의 최대 수를 나타내는 것으로 계산되며 각각 I 및 SE보다 커야 한다.

[0103] 모든 메모리 셀에서 메모리 오류가 발생할 수 있으므로 모든 오류 메모리 주소는 메모리 복구를 위해 CAM에 저장되어야 하고, CAM은 대면적 오버헤드이지만 필요하다. 메모리 복구를 위한 CAM의 크기는 메모리 복구를 위한 Rs 및 Cs에 비례하며, 메모리의 밀도와 용량이 크게 증가하기 때문에 Rs 및 Cs는 일반적으로 2보다 크다.

[0104] 제안된 실리콘 디버그를 위한 CAM의 크기는 L(Number of observed signals), I(Number of debug intervals for DS cycles) 및 SE(Number of segments for a debug interval)에 의해 결정될 수 있다. L은 메모리 복구를 위한 CAM의 전체 크기에 비해 작은 값이며 공간 압축기(예, XOR 트리)를 사용하여 줄일 수 있다. 또한 I와 SE가 2 이상으로 설정된 경우에만 제안된 방법을 적용 할 수 있다. 그러나 메모리 복구 용 CAM의 크기가 매우 크기 때문에 I와 SE가 상당히 큰 값으로 설정되어 있어도 Pmax 및 Cmax도 매우 클 수 있다. 따라서 메모리 복구를 위한 CAM의 크기는 제안된 방법을 재사용하기에 충분하므로 제안된 방법을 적용하는 것은 문제가 되지 않는다. BRANCH는 대표적인 BIRA 방법으로 예시일 뿐이며, BISR을 재사용하는 제안된 방법은 모든 BISR에 유사하게 적용될 수 있다.

[0105] 골든 시그니처는 온칩 오류 감지를 수행하는 데 필요하며 디버그 데이터는 추적 버퍼에 캡처된다. 온칩 오류 감지의 결과는 비트로 획득된다. 여기서 1은 오류를 나타내고 0은 오류가 없음을 나타낸다. 디버그 테스트 중에 잘못된 데이터는 쉐도우 버퍼에 선택적으로 캡처되고 태그 비트를 사용하여 DRAM에 저장된다. 그런 다음 후속

디버그 데이터가 추적 버퍼에 캡처되고 후속 골든 시그니처가 DRAM에서 CAM으로 전송되어 CAM에 저장된다. 디버그 프로세스가 끝난 후 CUD(Circuit Under Debug)가 제품 릴리스 자격을 충족하면 유효성 검사 프로세스가 종료된다.

[0106] 디버깅 장치에 포함된 복수의 구성요소들은 상호 결합되어 적어도 하나의 모듈로 구현될 수 있다. 구성요소들은 장치 내부의 소프트웨어적인 모듈 또는 하드웨어적인 모듈을 연결하는 통신 경로에 연결되어 상호 간에 유기적으로 동작한다. 이러한 구성요소들은 하나 이상의 통신 버스 또는 신호선을 이용하여 통신한다.

[0107] 디버깅 장치는 하드웨어, 펌웨어, 소프트웨어 또는 이들의 조합에 의해 로직회로 내에서 구현될 수 있고, 범용 또는 특정 목적 컴퓨터를 이용하여 구현될 수도 있다. 장치는 고정배선형(Hardwired) 기기, 필드 프로그램 가능한 게이트 어레이(Field Programmable Gate Array, FPGA), 주문형 반도체(Application Specific Integrated Circuit, ASIC) 등을 이용하여 구현될 수 있다. 또한, 장치는 하나 이상의 프로세서 및 컨트롤러를 포함한 시스템온칩(System on Chip, SoC)으로 구현될 수 있다.

[0108] 디버깅 장치는 하드웨어적 요소가 마련된 컴퓨팅 디바이스에 소프트웨어, 하드웨어, 또는 이들의 조합하는 형태로 탑재될 수 있다. 컴퓨팅 디바이스는 각종 기기 또는 유무선 통신망과 통신을 수행하기 위한 통신 모뎀 등의 통신장치, 프로그램을 실행하기 위한 데이터를 저장하는 메모리, 프로그램을 실행하여 연산 및 명령하기 위한 마이크로프로세서 등을 전부 또는 일부 포함한 다양한 장치를 의미할 수 있다.

[0109] 도 2에서는 각각의 과정을 순차적으로 실행하는 것으로 기재하고 있으나 이는 예시적으로 설명한 것에 불과하고, 이 분야의 기술자라면 본 발명의 실시예의 본질적인 특성에서 벗어나지 않는 범위에서 도 2에 기재된 순서를 변경하여 실행하거나 또는 하나 이상의 과정을 병렬적으로 실행하거나 다른 과정을 추가하는 것으로 다양하게 수정 및 변형하여 적용 가능할 것이다.

[0110] 본 실시예들에 따른 동작은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능한 매체에 기록될 수 있다. 컴퓨터 판독 가능한 매체는 실행을 위해 프로세서에 명령어를 제공하는데 참여한 임의의 매체를 나타낸다. 컴퓨터 판독 가능한 매체는 프로그램 명령, 데이터 파일, 데이터 구조 또는 이들의 조합을 포함할 수 있다. 예를 들면, 자기 매체, 광기록 매체, 메모리 등이 있을 수 있다. 컴퓨터 프로그램은 네트워크로 연결된 컴퓨터 시스템 상에 분산되어 분산 방식으로 컴퓨터가 읽을 수 있는 코드가 저장되고 실행될 수도 있다. 본 실시예를 구현하기 위한 기능적인(Functional) 프로그램, 코드, 및 코드 세그먼트들은 본 실시예가 속하는 기술분야의 프로그래머들에 의해 용이하게 추론될 수 있을 것이다.

[0111] 본 실시예들은 본 실시예의 기술 사상을 설명하기 위한 것이고, 이러한 실시예에 의하여 본 실시예의 기술 사상의 범위가 한정되는 것은 아니다. 본 실시예의 보호 범위는 아래의 청구범위에 의하여 해석되어야 하며, 그와 동등한 범위 내에 있는 모든 기술 사상은 본 실시예의 권리범위에 포함되는 것으로 해석되어야 할 것이다.

## *부호의 설명*

[0112] 10: 디버깅 장치

100: 데이터 압축기

200: 버퍼

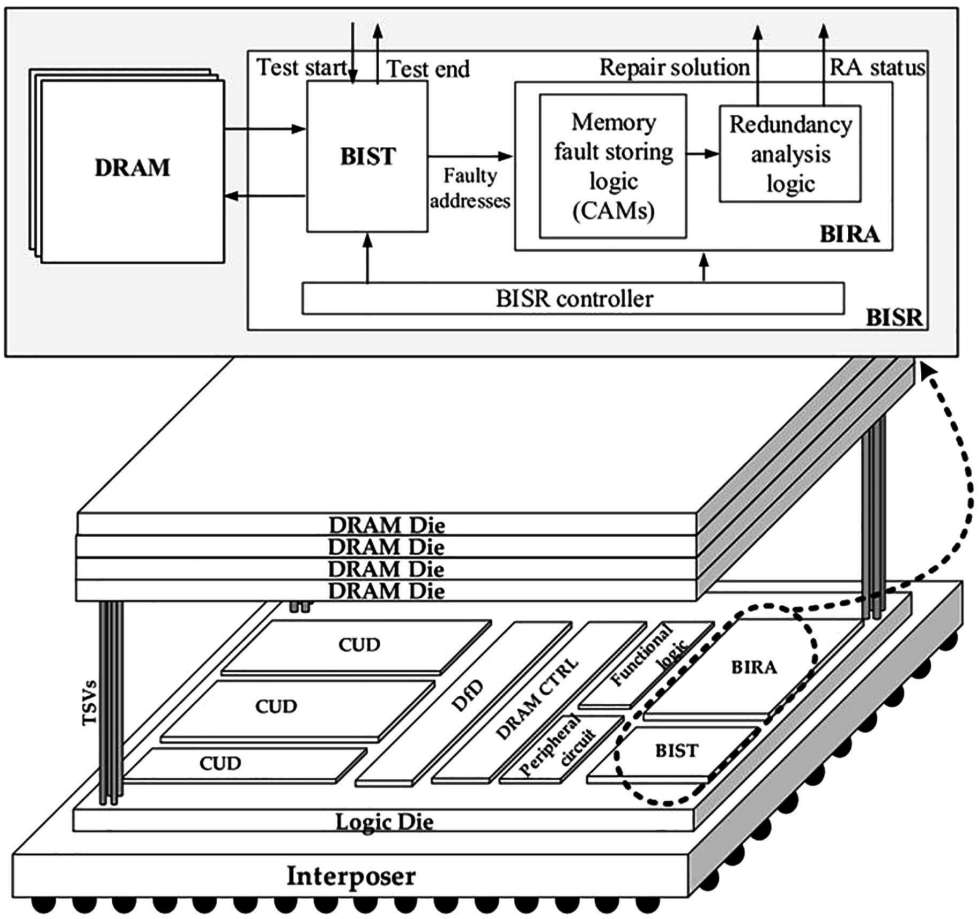
300: 디버그 메모리

400: 메모리 구조 변환기

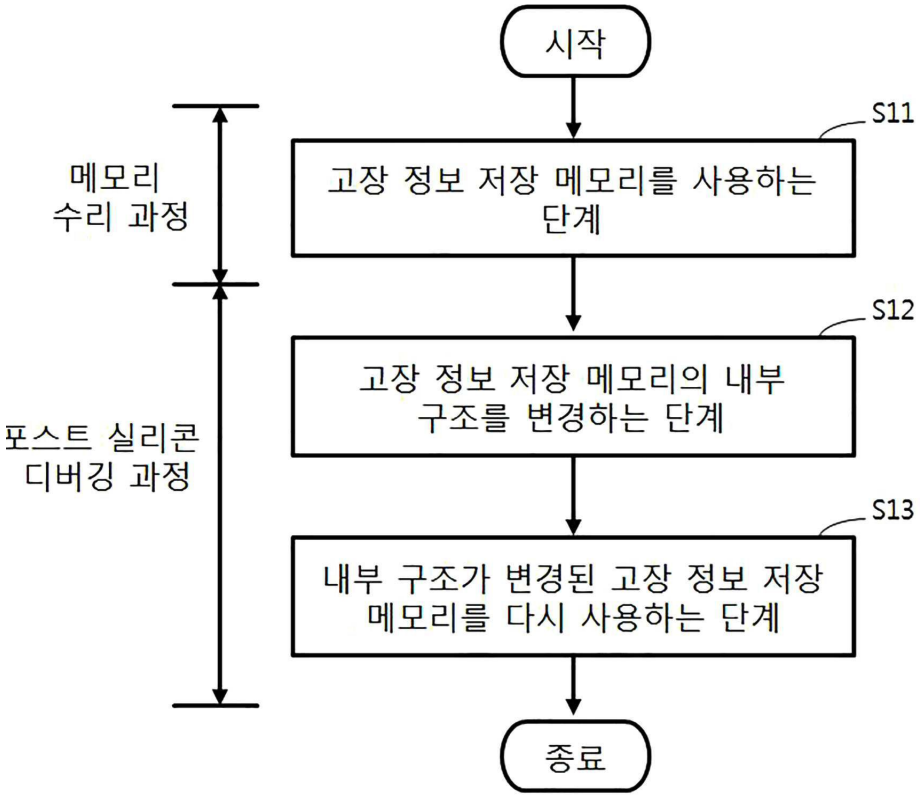
500: 고장 정보 저장 메모리

도면

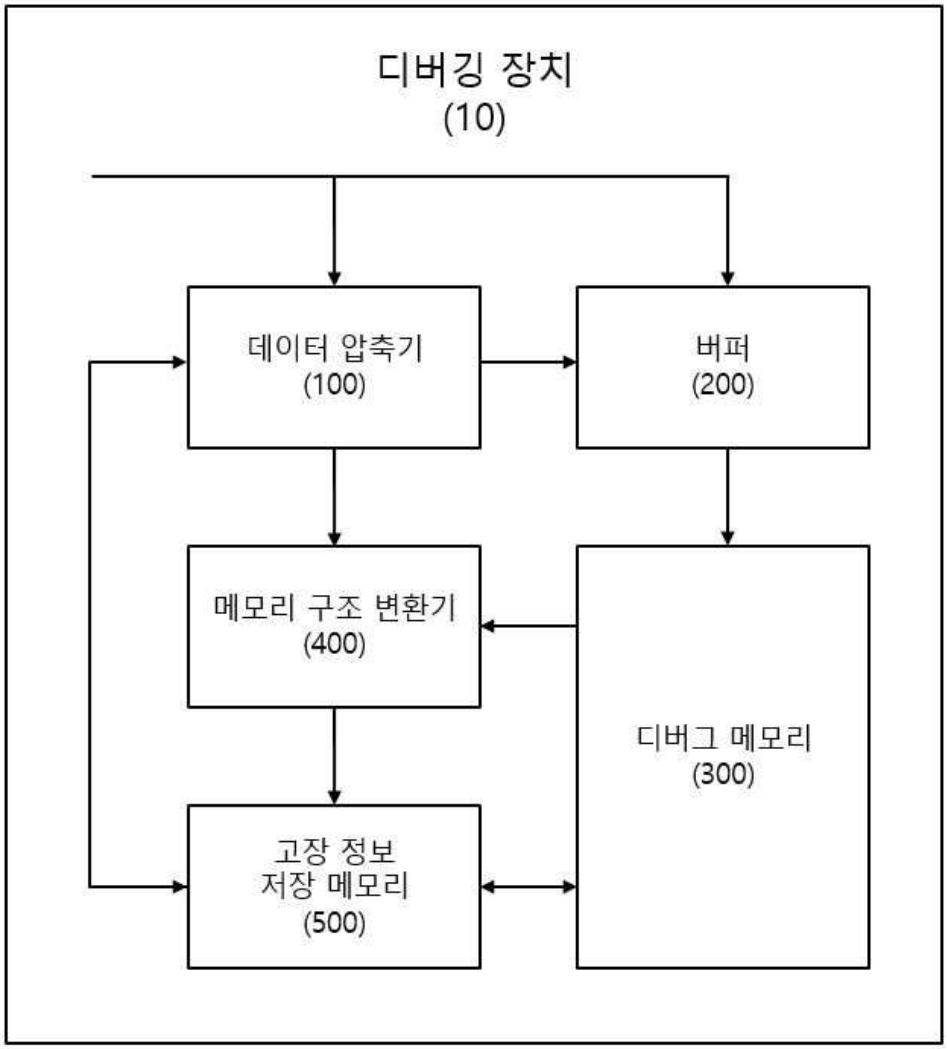
도면1


Test start
Test end
Repair solution
RA status
DRAM
BIST
Faulty addresses
Memory fault storing logic (CAMs)
Redundancy analysis logic
BIRA
BISR controller
BISR
DRAM Die
DRAM Die
DRAM Die
DRAM Die
CUD
CUD
CUD
TSVs
DfD
DRAM CTRL
Peripheral circuit
Functional logic
BIRA
BIST
Logic Die
Interposer

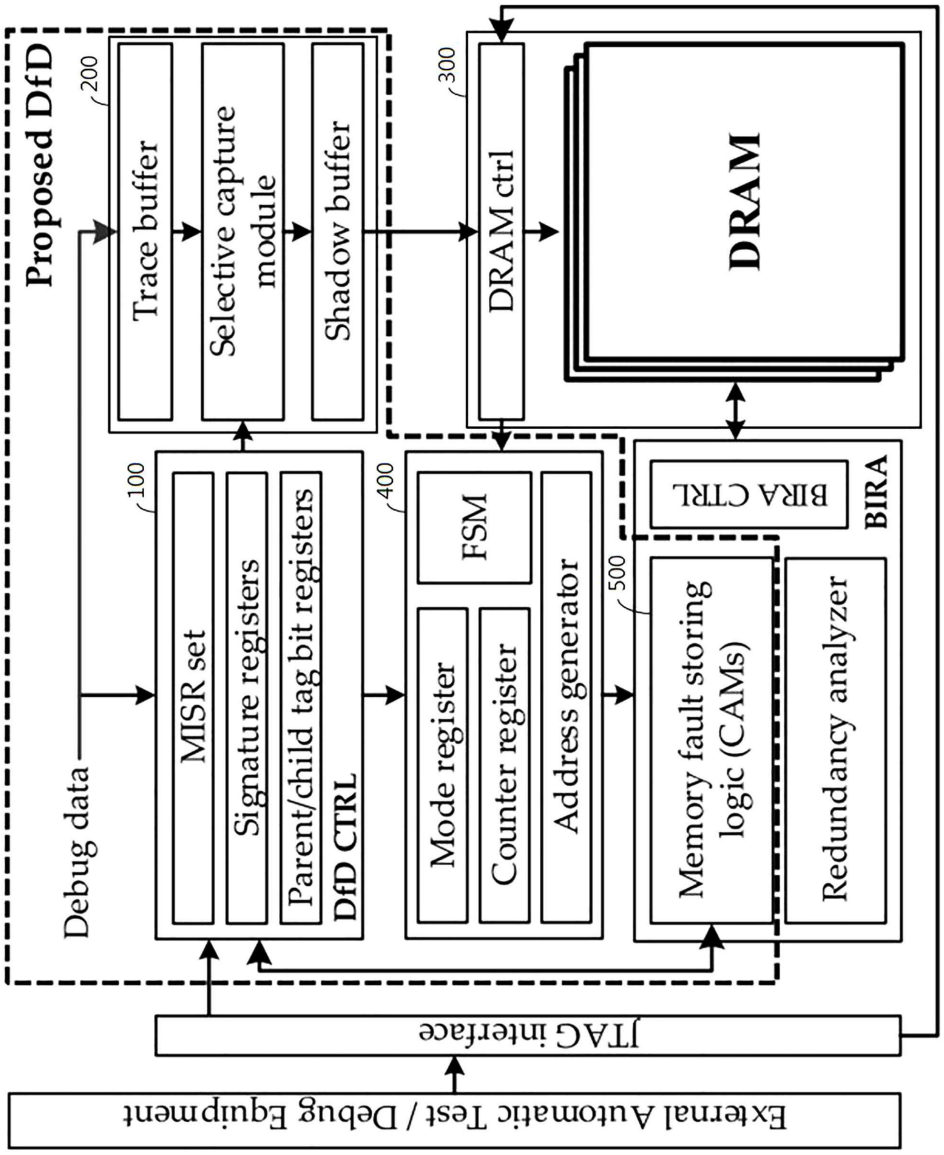
도면2


시작
S11
고장 정보 저장 메모리를 사용하는 단계
S12
고장 정보 저장 메모리의 내부 구조를 변경하는 단계
S13
내부 구조가 변경된 고장 정보 저장 메모리를 다시 사용하는 단계
종료
메모리 수리 과정
포스트 실리콘 디버깅 과정

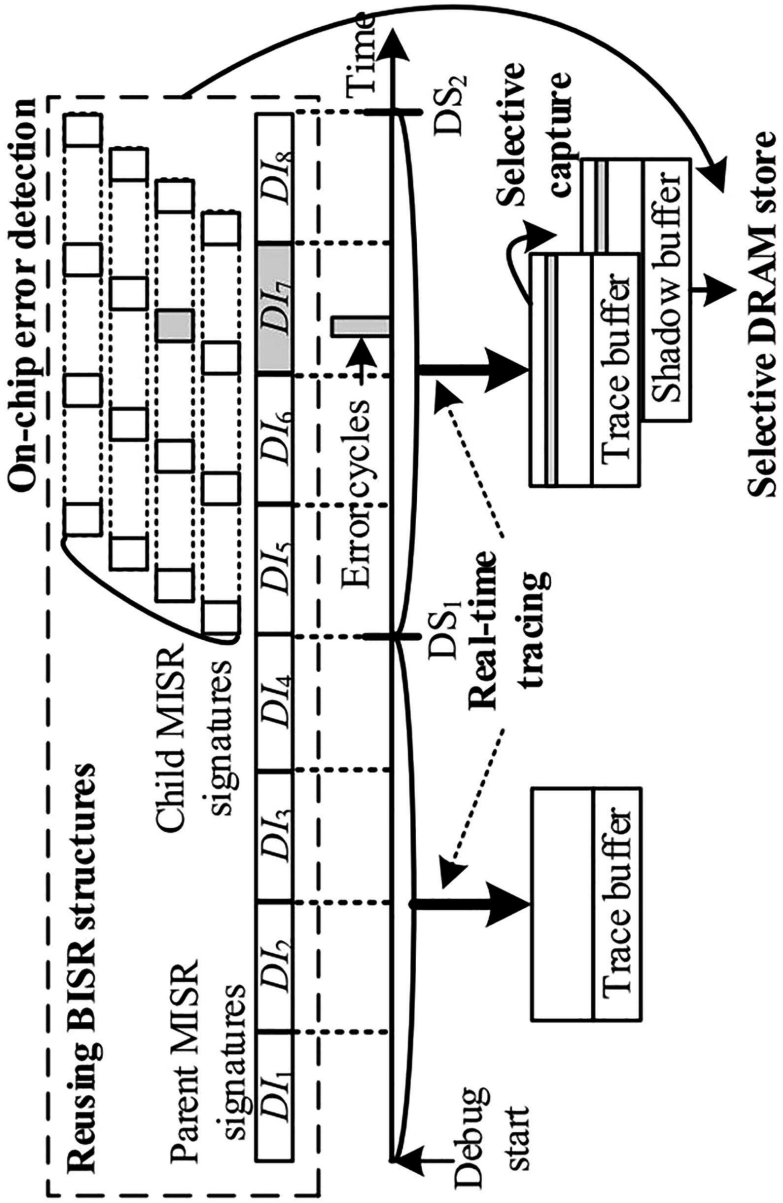
도면3


디버깅 장치
(10)
데이터 압축기
(100)
버퍼
(200)
메모리 구조 변환기
(400)
디버그 메모리
(300)
고장 정보
저장 메모리
(500)

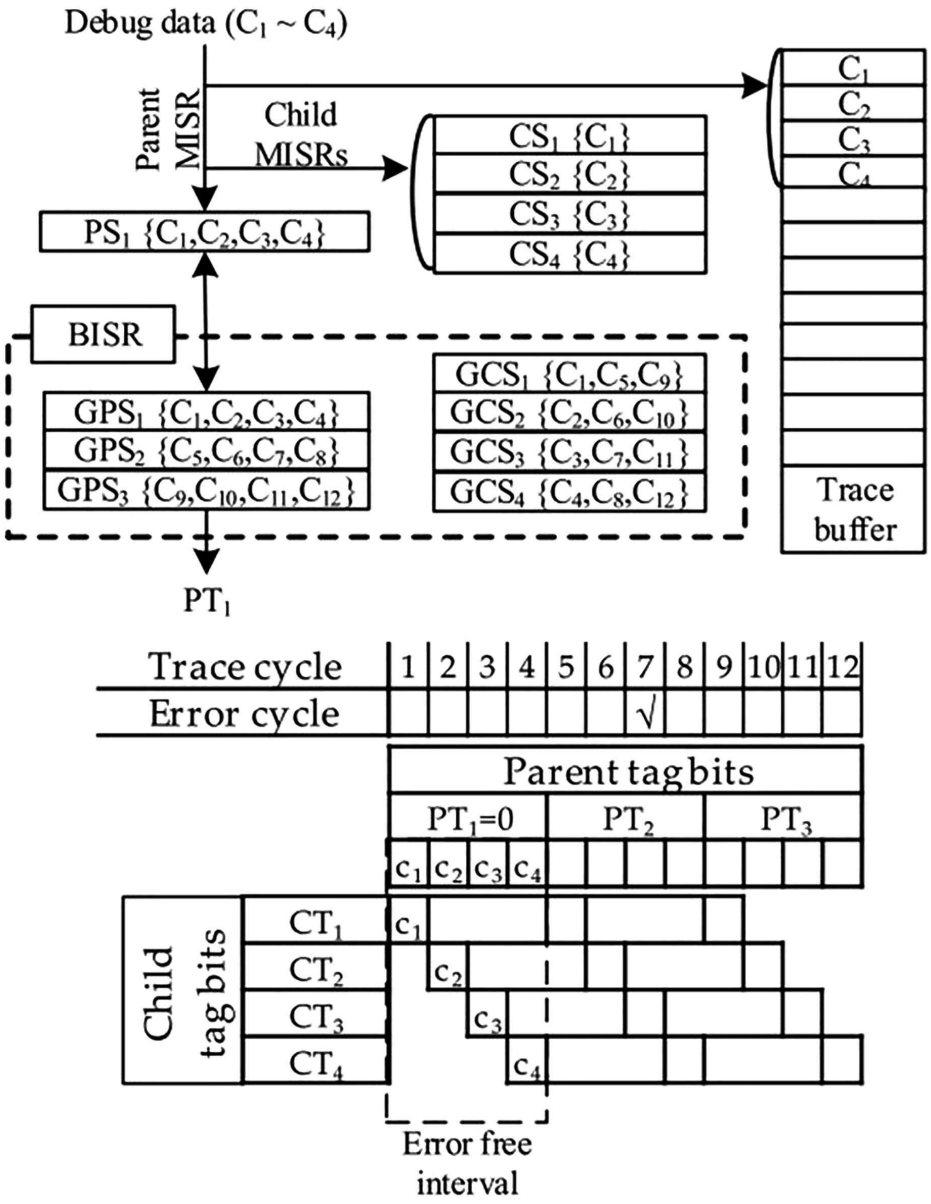
도면4


External Automatic Test / Debug Equipment
JTAG interface
Proposed DfD
Debug data
100
MISR set
Signature registers
Parent/child tag bit registers
DfD CTRL
200
Trace buffer
Selective capture module
Shadow buffer
400
FSM
Mode register
Counter register
Address generator
500
BIRA CTRL
Memory fault storing logic (CAMs)
Redundancy analyzer
BIRA
300
DRAM ctrl
DRAM

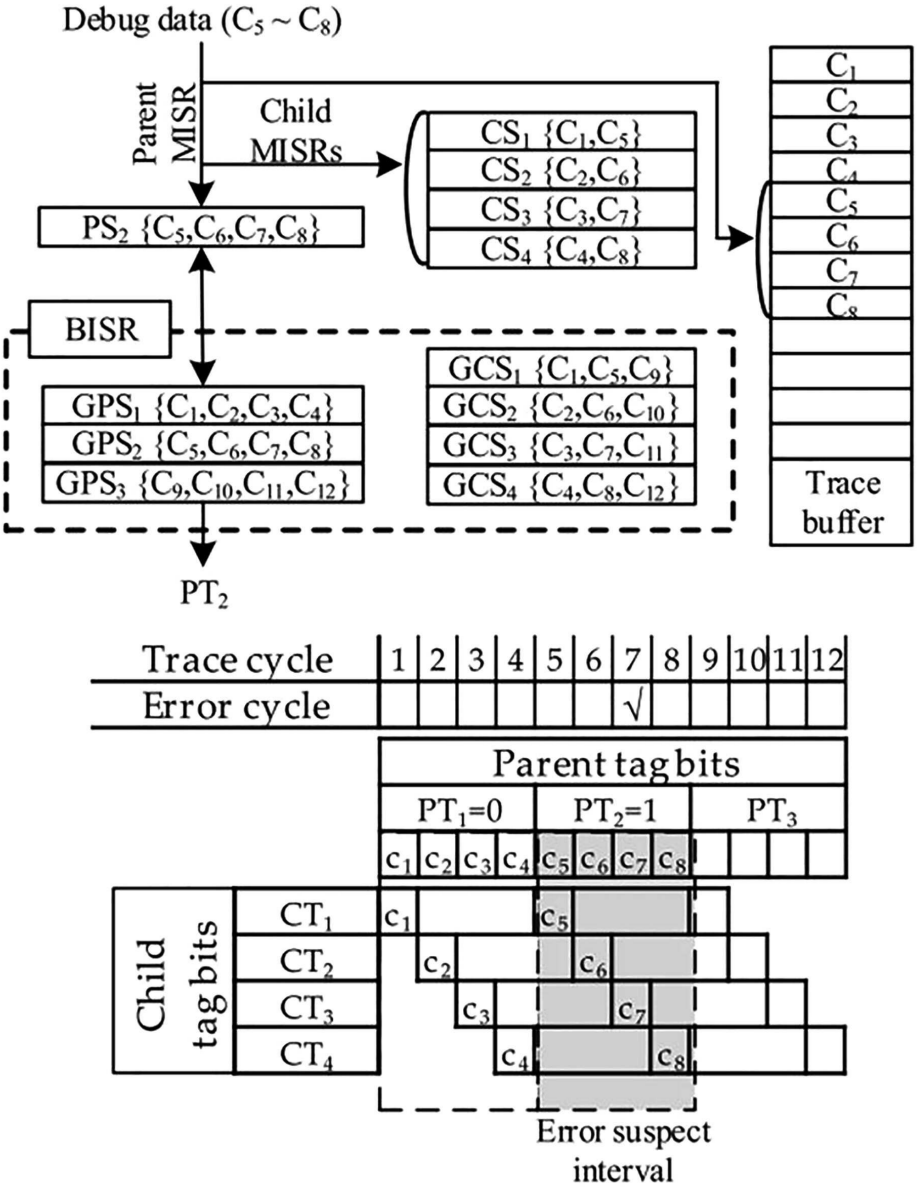
도면5


On-chip error detection
Reusing BISR structures
Parent MISR signatures
Child MISR signatures
DI1
DI2
DI3
DI4
DI5
DI6
DI7
DI8
Time
Error cycles
DS1
Real-time tracing
DS2
Selective capture
Trace buffer
Shadow buffer
Selective DRAM store
Trace buffer
Debug start

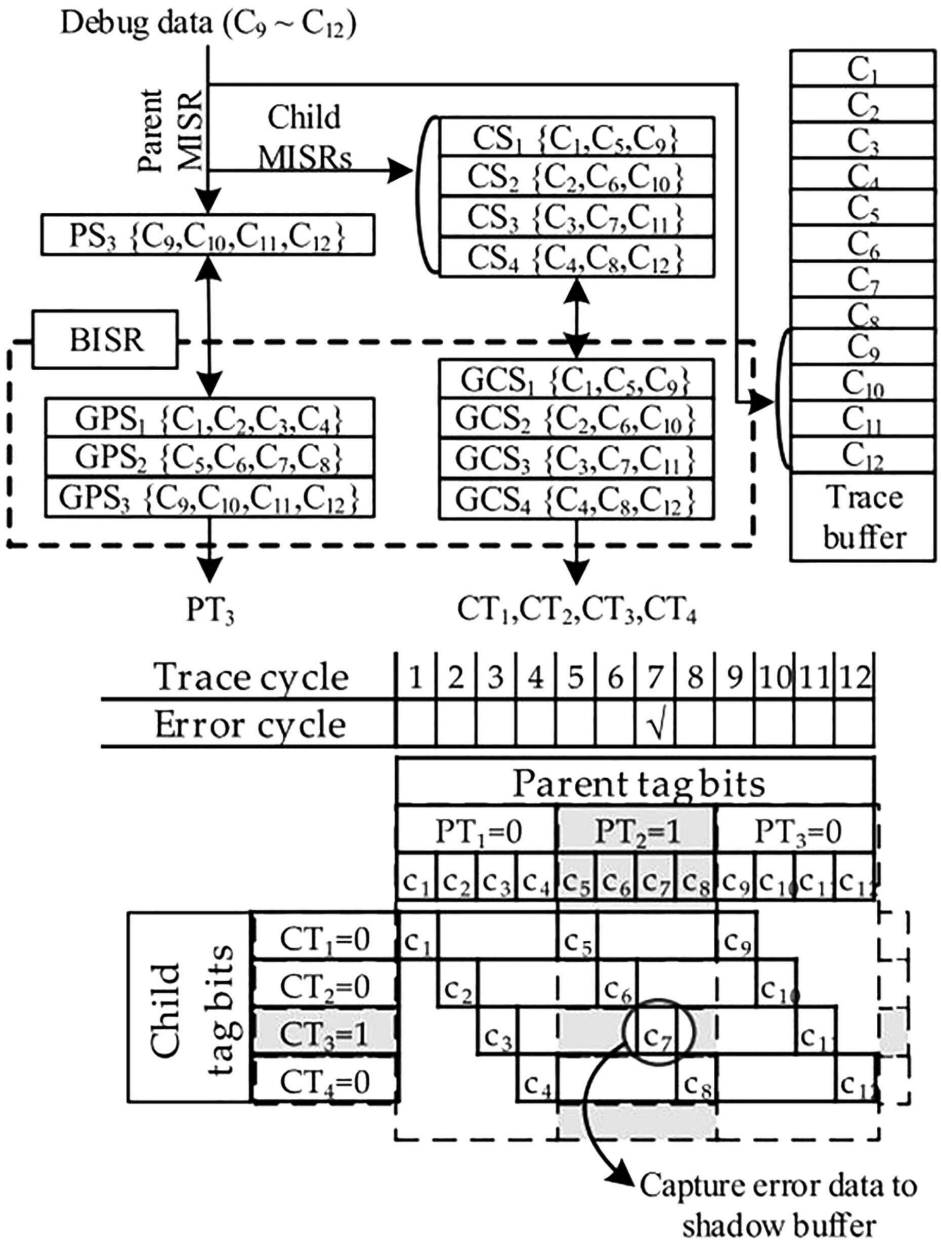
도면6


Debug data ($C_1 \sim C_4$)
Parent MISR
Child MISRs
$CS_1$ {$C_1$}
$CS_2$ {$C_2$}
$CS_3$ {$C_3$}
$CS_4$ {$C_4$}
$C_1$
$C_2$
$C_3$
$C_4$
$PS_1$ {$C_1$,$C_2$,$C_3$,$C_4$}
BISR
$GPS_1$ {$C_1$,$C_2$,$C_3$,$C_4$}
$GPS_2$ {$C_5$,$C_6$,$C_7$,$C_8$}
$GPS_3$ {$C_9$,$C_{10}$,$C_{11}$,$C_{12}$}
$GCS_1$ {$C_1$,$C_5$,$C_9$}
$GCS_2$ {$C_2$,$C_6$,$C_{10}$}
$GCS_3$ {$C_3$,$C_7$,$C_{11}$}
$GCS_4$ {$C_4$,$C_8$,$C_{12}$}
Trace buffer
$PT_1$
Trace cycle
1 2 3 4 5 6 7 8 9 10 11 12
Error cycle
√
Parent tag bits
$PT_1$=0
$PT_2$
$PT_3$
$c_1$ $c_2$ $c_3$ $c_4$
Child tag bits
$CT_1$
$CT_2$
$CT_3$
$CT_4$
Error free interval

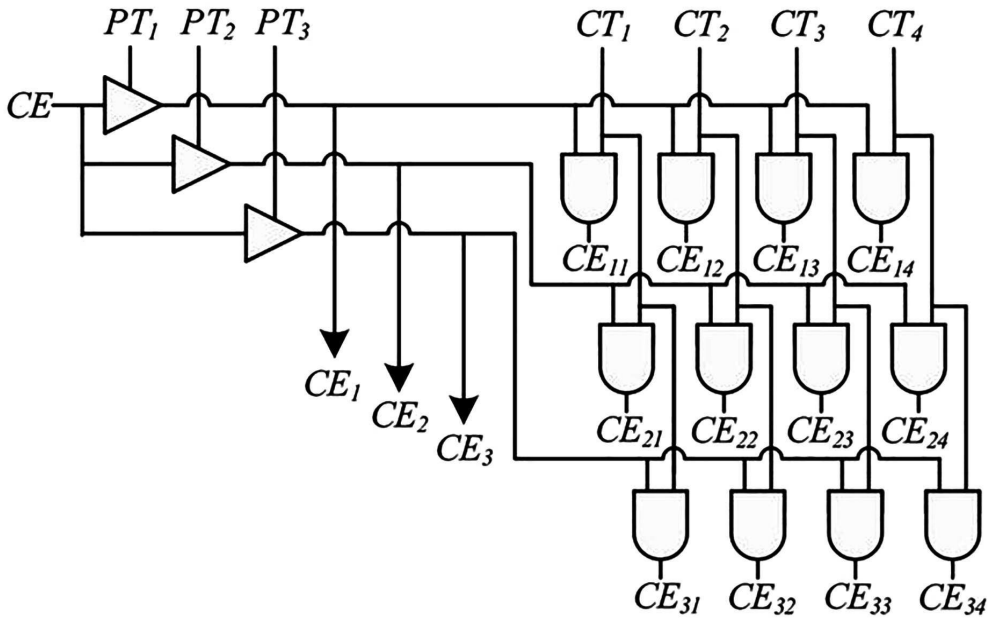
도면7


Debug data ($C_5 \sim C_8$)
Parent MISR
Child MISRs
$CS_1$ {$C_1$,$C_5$}
$CS_2$ {$C_2$,$C_6$}
$CS_3$ {$C_3$,$C_7$}
$CS_4$ {$C_4$,$C_8$}
$PS_2$ {$C_5$,$C_6$,$C_7$,$C_8$}
BISR
$GPS_1$ {$C_1$,$C_2$,$C_3$,$C_4$}
$GPS_2$ {$C_5$,$C_6$,$C_7$,$C_8$}
$GPS_3$ {$C_9$,$C_{10}$,$C_{11}$,$C_{12}$}
$GCS_1$ {$C_1$,$C_5$,$C_9$}
$GCS_2$ {$C_2$,$C_6$,$C_{10}$}
$GCS_3$ {$C_3$,$C_7$,$C_{11}$}
$GCS_4$ {$C_4$,$C_8$,$C_{12}$}
$C_1$ $C_2$ $C_3$ $C_4$ $C_5$ $C_6$ $C_7$ $C_8$
Trace buffer
$PT_2$
Trace cycle 1 2 3 4 5 6 7 8 9 10 11 12
Error cycle √
Parent tag bits
$PT_1$=0
$PT_2$=1
$PT_3$
$c_1$ $c_2$ $c_3$ $c_4$ $c_5$ $c_6$ $c_7$ $c_8$
Child tag bits
$CT_1$
$CT_2$
$CT_3$
$CT_4$
Error suspect interval

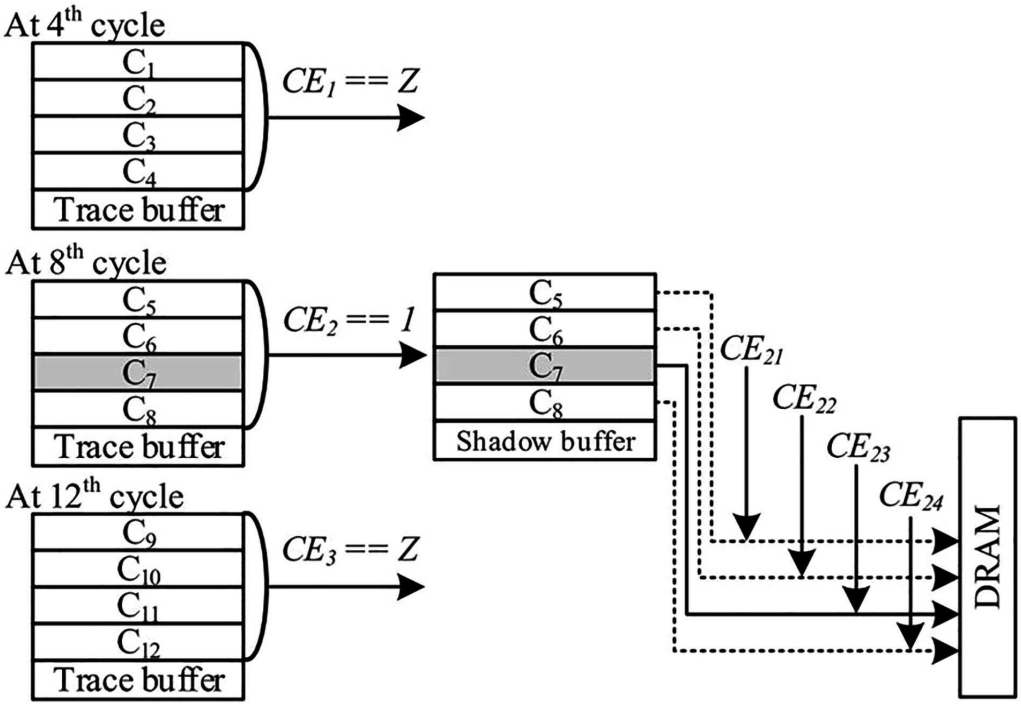
도면8


Debug data ($C_9 \sim C_{12}$)
Parent MISR
Child MISRs
$CS_1$ {$C_1$,$C_5$,$C_9$}
$CS_2$ {$C_2$,$C_6$,$C_{10}$}
$CS_3$ {$C_3$,$C_7$,$C_{11}$}
$CS_4$ {$C_4$,$C_8$,$C_{12}$}
$PS_3$ {$C_9$,$C_{10}$,$C_{11}$,$C_{12}$}
BISR
$GPS_1$ {$C_1$,$C_2$,$C_3$,$C_4$}
$GPS_2$ {$C_5$,$C_6$,$C_7$,$C_8$}
$GPS_3$ {$C_9$,$C_{10}$,$C_{11}$,$C_{12}$}
$GCS_1$ {$C_1$,$C_5$,$C_9$}
$GCS_2$ {$C_2$,$C_6$,$C_{10}$}
$GCS_3$ {$C_3$,$C_7$,$C_{11}$}
$GCS_4$ {$C_4$,$C_8$,$C_{12}$}
$C_1$ $C_2$ $C_3$ $C_4$ $C_5$ $C_6$ $C_7$ $C_8$ $C_9$ $C_{10}$ $C_{11}$ $C_{12}$
Trace buffer
$PT_3$
$CT_1$,$CT_2$,$CT_3$,$CT_4$
Trace cycle 1 2 3 4 5 6 7 8 9 10 11 12
Error cycle √
Parent tag bits
$PT_1$=0 $PT_2$=1 $PT_3$=0
Child tag bits
$CT_1$=0 $CT_2$=0 $CT_3$=1 $CT_4$=0
Capture error data to shadow buffer

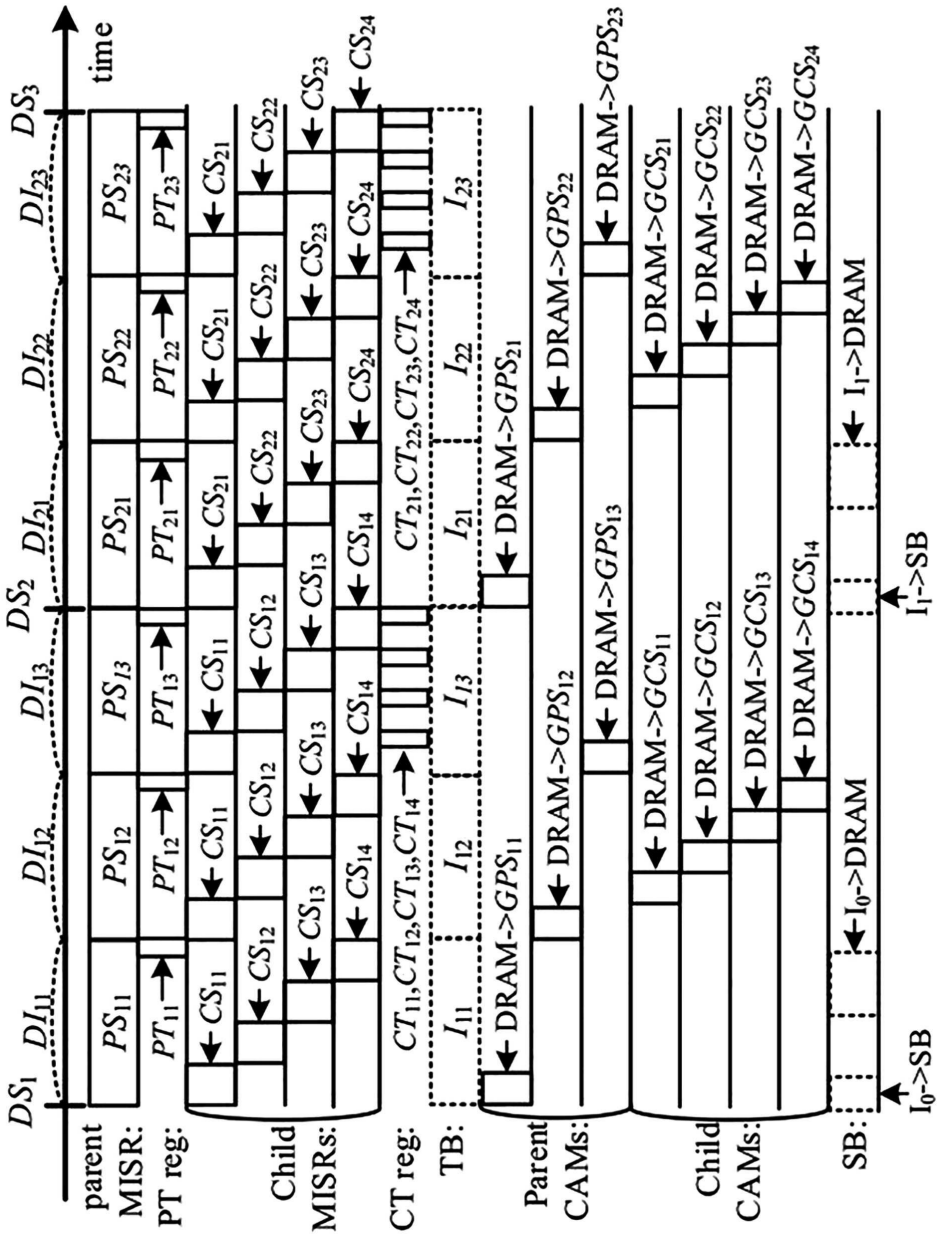
도면9


PT1 PT2 PT3
CT1 CT2 CT3 CT4
CE
CE11 CE12 CE13 CE14
CE1 CE2 CE3
CE21 CE22 CE23 CE24
CE31 CE32 CE33 CE34

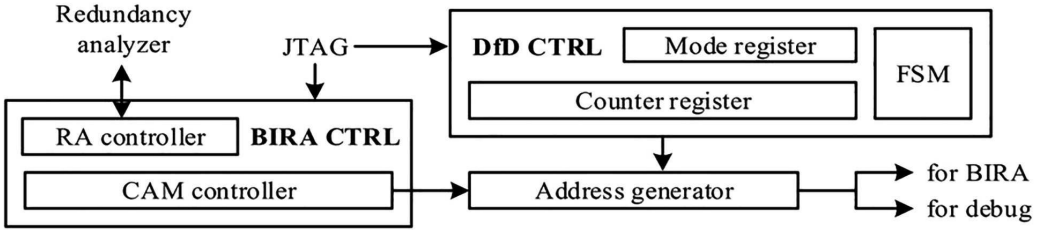
도면10


At 4th cycle
C1
C2
C3
C4
Trace buffer
CE1 == Z
At 8th cycle
C5
C6
C7
C8
Trace buffer
CE2 == 1
C5
C6
C7
C8
Shadow buffer
CE21
CE22
CE23
CE24
DRAM
At 12th cycle
C9
C10
C11
C12
Trace buffer
CE3 == Z

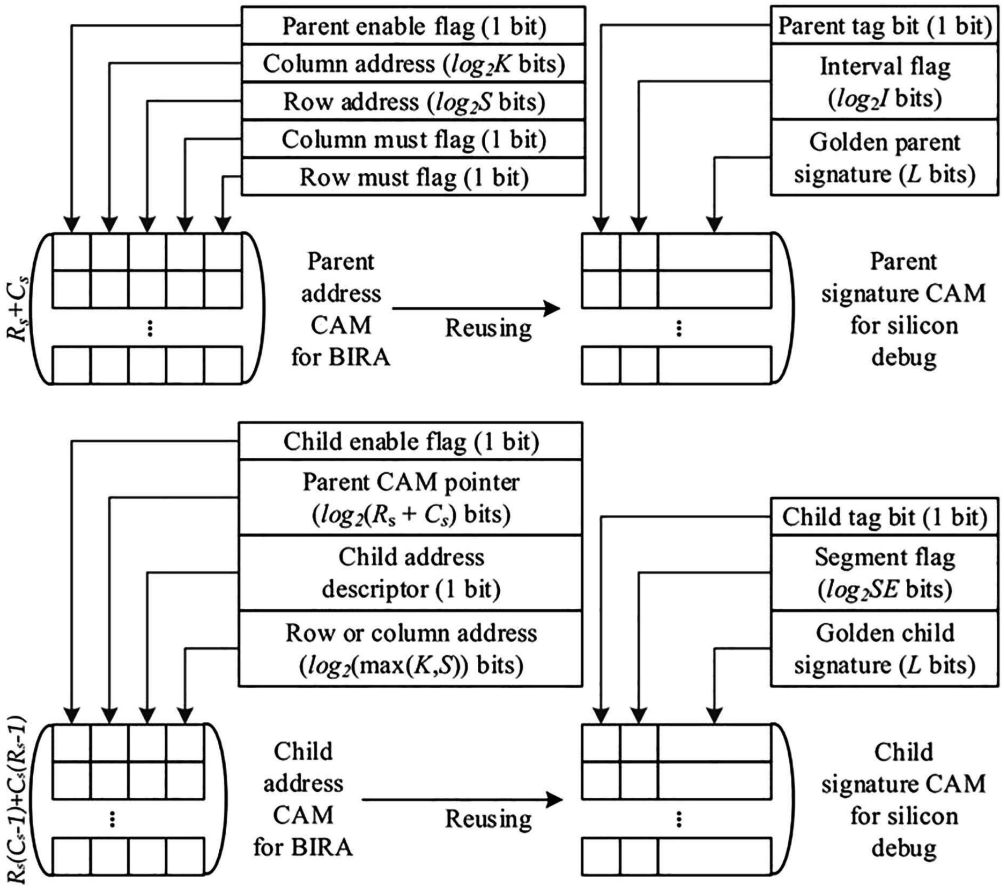
도면11


time
parent MISR:
PT reg:
Child MISRs:
CT reg:
TB:
Parent CAMs:
Child CAMs:
SB:
DRAM->GPS11
DRAM->GPS12
DRAM->GPS21
DRAM->GPS22
DRAM->GPS13
DRAM->GPS23
DRAM->GCS11
DRAM->GCS21
DRAM->GCS12
DRAM->GCS22
DRAM->GCS13
DRAM->GCS23
DRAM->GCS14
DRAM->GCS24
I0->DRAM
I1->DRAM
I0->SB
I1->SB

도면12


Redundancy analyzer
JTAG
DfD CTRL
Mode register
Counter register
FSM
RA controller
BIRA CTRL
CAM controller
Address generator
for BIRA
for debug

도면13


Parent enable flag (1 bit)
Column address ($log_2K$ bits)
Row address ($log_2S$ bits)
Column must flag (1 bit)
Row must flag (1 bit)
Parent tag bit (1 bit)
Interval flag ($log_2I$ bits)
Golden parent signature ($L$ bits)
$R_s+C_s$
Parent address CAM for BIRA
Reusing
Parent signature CAM for silicon debug
Child enable flag (1 bit)
Parent CAM pointer ($log_2(R_s + C_s)$ bits)
Child address descriptor (1 bit)
Row or column address ($log_2(\max(K,S))$ bits)
Child tag bit (1 bit)
Segment flag ($log_2SE$ bits)
Golden child signature ($L$ bits)
$R_d(C_s-1)+C_s(R_s-1)$
Child address CAM for BIRA
Reusing
Child signature CAM for silicon debug