(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2008년09월04일
(11) 등록번호 10-0856336
(24) 등록일자 2008년08월28일

(51) Int. Cl.
G06F 11/28 (2006.01)
(21) 출원번호 10-2001-0005819
(22) 출원일자 2001년02월07일
심사청구일자 2006년02월07일
(65) 공개번호 10-2001-0078358
(43) 공개일자 2001년08월20일
(30) 우선권주장
09/498,812 2000년02월07일 미국(US)
(56) 선행기술조사문헌
US 5978937 A
US 5511215 A

(73) 특허권자
프리스케일 세미컨덕터, 인크.
미합중국 텍사스 (우편번호 78735) 오스틴 윌리암 캐논 드라이브 웨스트 6501
(72) 발명자
모예르윌리암씨.
미국,텍사스78620,드리핑스프링스,피에르브랜치로드1005
피즈심몬스마이클디.
미국,텍사스78731,오스틴#723,노스캐피탈오브텍사스하이웨이7700
콜린스리차드지.
미국,텍사스78729,매치록코브오스틴8109
(74) 대리인
이병호, 장훈

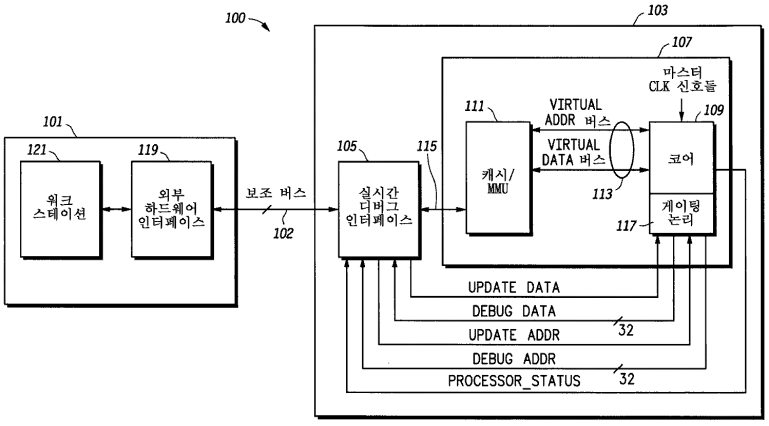
전체 청구항 수 : 총 4 항 심사관 : 정재우

(54) 실시간 프로세서 디버그 시스템

(57) 요 약

본 발명은 전력 소비를 줄이고 버스 로딩으로 인한 성능 영향(performance impact)을 최소화하기 위해 실시간 동작들 동안 코어 프로세서의 가상 버스 어드레스 및 데이터 신호들을 선택적으로 샘플링하는 실시간 프로세서 디버그 시스템에 관한 것이다. 매입형 프로세서용 게이팅 논리는 매입형 시스템 칩 또는 집적 회로(IC)의 디버그 인터페이스에 어드레스 및 데이터 신호들을 포함하는 코어 프로세서의 가상 버스 신호들을 제공한다. 게이팅 논리는 해당 디버그 어드레스 및 데이터 버스들의 스위칭을 제어하기 위해 디버그 인터페이스로부터 업데이트 데이터 및 업데이트 어드레스 신호들을 수신한다. 코어 프로세서는 디버그 인터페이스가 코어 프로세서에 의해 실행되는 가상 트랜잭션을 결정하도록 디버그 인터페이스에 프로세서 상태 신호들을 제공한다. 프로세서 상태 신호(processor status signal)들은 명령 페치(fetch) 또는 데이터 페치 사이클들 뿐만 아니라 흐름의 변화(COF:change of flow) 이벤트들을 나타낸다. 프로세서 상태 신호들은 또한 데이터 사이클이 판독인지 기록인지를 나타낸다. 디버그 인터페이스는 상당한 이벤트들을 확인하고 가상 버스를 샘플링함에 따라 업데이트 데이터 또는 업데이트 어드레스 신호들을 어서트(assert)하는 프로세서 상태 신호들을 모니터한다. 이러한 방법에서, 프로그램 트레이스(trace) 메시징 동안, 디버그 인터페이스는 COF 이벤트 동안의 디버그 어드레스 버스를 경유하여 가상 어드레스 버스를 샘플링한다. 또한, 디버그 인터페이스는 데이터 페치 사이클들을 검출하고, 가상 어드레스 버스를 샘플링하고, 하나 이상의 미리결정된 관련 어드레스들이나 어드레스 영역들에 대한 어드레스를 비교한다. 상당한 데이터 사이클의 어드레스가 미리규정된 영역 내에 있으면, 디버그 인터페이스는 가상 데이터 버스를 샘플링하기 위해 업데이트 데이터 신호를 어서트(assert)한다.

대표도



특허청구의 범위

청구항 1

집적 회로에 있어서,

코어 프로세서의 정규 동작(normal operation) 동안 동작 신호를 제공하기 위한 코어 프로세서;

상기 동작 신호를 수신하기 위한 입력, 디버그 업데이트 신호를 수신하기 위한 제 2 입력, 및 출력을 갖는 논리 회로; 및

상기 디버그 업데이트 신호를 제공하는 제 1 출력, 상기 논리 회로의 출력에 결합된 제 1 입력, 및 상기 집적 회로의 외부에 액세스 가능한 보조 버스에 결합된 제 2 출력을 갖는 실시간 디버그 회로를 포함하고,

상기 집적 회로는 디버그 모드를 갖고, 상기 실시간 디버그 회로는 상기 디버그 모드의 서브세트로서 선택가능한 모드를 갖는, 집적 회로.

청구항 2

실시간 디버그 시스템에 있어서,

가상 어드레스 버스를 포함하는 제 1 출력 및 코어 트랜잭션 상태(core transaction status) 출력 신호를 제공하는 제 2 출력을 갖는 코어 프로세서;

상기 가상 어드레스 버스에 결합된 제 1 입력, 디버그 어드레스 버스로서의 출력, 및 제 2 입력을 갖는 제 1 논리 회로;

상기 코어 트랜잭션 상태 출력 신호가 어드레스 흐름의 변화(an address change of flow:COF)를 지시할 때 어드레스 디버그 신호를 상기 논리 회로의 제 2 입력에 어서팅(asserting)하는 디버그 회로; 및

상기 가상 데이터 버스에 결합된 제 1 입력, 데이터 디버그 버스로서의 출력, 및 제 2 입력을 갖는 제 2 논리 회로를 포함하고,

상기 디버그 회로는 상기 코어 트랜잭션 상태 출력 신호가 데이터 액세스를 지시할 때 데이터 디버그 신호를 상기 제 2 논리 회로의 제 2 입력에 어서팅하는, 실시간 디버그 시스템.

청구항 3

집적 회로의 내부 사용을 위해 동작 신호들을 발생시키는 코어 프로세서 및 디버그 회로를 갖는 상기 집적 회로 내의 실시간 디버그 시스템에서의 방법에 있어서,

상기 집적 회로의 정규 동작 동안 디버그 모드로 들어가는 단계;

상기 디버그 모드의 서브세트인 선택가능한 모드로 들어가는 단계; 및

상기 선택가능한 모드 동안 디버그 신호를 어서팅하는 상기 디버그 회로에 응답하여 상기 디버그 회로에 상기 동작 신호들을 결합시키고, 상기 선택가능한 모드 동안 상기 디버그 신호를 디어서팅(de-asserting)하는 상기 디버그 회로에 응답하여 상기 디버그 회로로부터 상기 동작 신호들을 차단하는 단계를 포함하는, 방법.

청구항 4

실시간 디버그 시스템에 있어서,

정규 동작 동안 집적 회로의 내부에만 제공되는 출력을 갖는 코어 프로세서; 및 논리 회로로서, 상기 코어 프로세서의 상기 출력에 결합된 제 1 입력, 디버그 출력 및 상기 논리 회로가 디버그 신호의 디어서트(de-assert)에 응답하여 상기 디버그 출력의 논리 상태를 스위칭하는 것을 막기 위한 제 2 입력을 갖는 상기 논리 회로를 포함하는, 실시간 디버그 시스템.

명 세 서

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

<12> 발명의 분야

<13> 본 발명은 실시간 프로세서 디버그 시스템에 관한 것으로, 특히 전력 소비를 줄이고 버스 로딩(bus loading)으로 인한 성능 영향을 최소화하기 위해 실시간 동작 동안 코어(core) 프로세서의 가상 버스 어드레스 및 데이터 신호들을 선택적으로 샘플링하는 디버그 시스템에 관한 것이다.

<14> 발명의 배경

<15> 코어 프로세서, 온칩(on-chip) 메모리 및 외부 메모리 인터페이스 모듈을 포함하는 매입형 시스템들이 일반적이다. 전류 발생 시스템들은 또한 캐시 메모리들을 통합하고 있다. 시스템 레벨 코드 개발자(developer)들은 매입형 시스템을 위해 필요한 특정 기능들을 실행하는 매입형 시스템들용 소프트웨어나 응용 코드를 기록해야한다. 기능의 가산 또는 분할, 바람직하지 못한 특성들의 제거, "버그들(bugs)" 또는 소프트웨어 에러들의 제거를 포함하고 다양한 이유들로 인해 개발자들은 여러번 대부분의 소프트웨어를 교정해야 한다. 그래서, 이것은 적당한 동작을 확보하기 위해 소프트웨어를 디버그하는 응용 코드 개발(develop) 동안 필요하다. 매입형 시스템의 응용 코드는 또한 동작 동안 진행하거나, 계속적인 또는 주기적인 조정을 요할 수 있다. 몇몇 구성들에서, 임의의 매개변수들, 특정 값(characteristic value)들 또는 상수들의 표(table)들은 실시간 동작 동안 조정 또는 변환을 요할 수 있다. 본 명세서에 언급되는 바와 같이 디버그 프로세서는 동작 동안 실시간 측정 또는 조정 기능들뿐만 아니라 초기 응용 코드 개발(develop)을 포함한다.

<16> 캐시 메모리가 없는 시스템들을 포함하는 몇몇 종래의 시스템들은 코어 프로세서 사이클들을 디버깅 목적용으로 쉽게 이용가능하도록 코어 프로세서 버스를 외부에서 제공한다. 시스템 레벨 디버그는 코어 프로세서의 핀(pin) 또는 버스 비지빌리티(visibility)를 더 이상 이용할 수 없으므로 캐시 메모리를 갖춘 매입형 시스템 환경에서 더 힘들어 진다. 특히, 캐시 메모리는 종종 코어 프로세서 버스를 외부에서 이용할 수 없도록 외부적인 물리적 버스(physical bus) 및 매입형 코어 프로세서 사이에 위치된다. 예를 들어, "쇼 사이클(show cycle)"의 제공은 캐시 메모리들을 갖추지 않은 종래의 매입형 시스템들에서처럼 선택사항(option)이 아니다.

<17> 전통적인 디버그 방법은 배경 디버그 모드(background debug mode; BDM)로서 언급된다. BDM은 프로세서가 디버그 동작들을 실행하는 코드 개발자를 위해 순서대로 정지되는 스태틱(static) 디버그 프로세스이다. 코드 개발자는 특정 포인트(specific point)들에서 프로세서를 정지시키기 위해 브레이크 포인트들을 설정하거나, 개개의 명령들을 통해 프로세서의 진행을 모니터하고 문제들 및 버그들을 확인하는 단계(step)를 선별할 수 있다. BDM은 몇몇 응용들에 대해 적당하지만, 다이나믹(dynamic) 조정 또는 측정을 포함하는 실시간 동작 동안에는 충분치 못하다. 또한, 임의의 디버그 동작들은 응용 코드가 매입형 코어 프로세서를 멈추지 않고 시험 및 수정될 것을 요구한다. 예를 들어, 자동차 엔진을 제어하기 위한 매입형 프로세서 시스템은 일반적으로 엔진이 가동되는 동안 디버그되므로, 코어 프로세서를 정지시키는 것은 엔진을 멈추게 하므로 효과적인 선택사항이 아니다.

<18> 다이나믹 디버그 방법들은 상기 프로세서의 멈춤 없이 프로세서 동작을 모니터하기 위해 개발되고 있다. 캐시로부터 물리적 버스를 외부에서 이용가능한 경우에, 결국 매입형 프로세서 핀들의 과도한 수를 야기하므로 코어 프로세서 버스를 외부에서 제공하는 것이 실질적이거나 실현가능하지 않다. 프로세서 버스를 모니터하고 내부의 보조 버스 또는 포트를 경유하여 메시지들을 제공하는 디버그 인터페이스를 개발하는 것이 고려된다. 코어 프로세서 및 캐시 사이의 데이터 버스들 및 어드레스를 모니터링함으로써 다이나믹 디버그는 전력 소비 및 관입 처리(intrusive process) 될 수 있다. 캐시 및 코어 프로세서 사이의 버스는 종종 "가상(virtual)" 버스로서 언급된다. 가상 버스에 결합된 커다란 버퍼들은 버스를 로딩시켜 매입형 프로세서 시스템의 실행을 달성할 수 있다. 또한, 어드레스 및 데이터 버스 신호들을 모니터하고 외부의 보조 버스에 대응 메시지들을 제공하는 다이나믹 프로세서는 유용한 전력을 소비한다. 휴대용 통신 장치들은 보통 포켓용이고 배터리로 동작되며, 주로 매입형 프로세서 시스템들에 사용된다. 이러한 휴대용 통신 장치들은, 예를 들어 개인 디지털 보조기(PDA), 셀 전화기들, 호출기들, 글로벌 위치(global positioning; GPS) 모듈들 등을 포함한다. 이러한 휴대용 통신 장치들에 있어 배터리 수명을 보전하기 위해 전력 소비를 최소화하는 것은 필연적이다.

<19> 이것은 정상 시스템 동작 동안 다이나믹 측정을 실행하는 시스템들에서 전력 소비를 최소화하기 위해 또한 중요

하다. 자용차용 응용들이 이런 요구에 대한 중요한 예이다. 예를 들어, 임의의 자동차의 응용들에서, 실시간 측정(calibration)(상수) 튜닝은 응용 및 프로세서가 동작하는 것과 같이 기능하는 디버그 모듈을 요한다. 매입형 시스템의 전력 소비의 최소화가 강력히 요구된다.

<20> 매입형 프로세서 시스템 상에서 버스 로딩으로인한 전력 소비 및 성능 영향을 최소화하기 위한 방법으로 다이나믹 디버그를 실행하는 것이 요구된다.

<21> 바람직한 실시예의 상세한 설명이 도면과 관련하여 고려될 때 본 발명을 더 잘 이해할 수 있다.

발명이 이루고자 하는 기술적 과제

<22> 본 발명은 전력 소비를 줄이고 버스 로딩으로 인한 성능 영향(performance impact)을 최소화하기 위해 실시간 동작 동안 코어 프로세서의 가상 버스의 어드레스 및 데이터 신호들을 선택적으로 샘플링하는 실시간 프로세서 디버그 시스템에 관한 것이다.

발명의 구성 및 작용

<23> 넥서스(Nexus) 표준 또는 GEPDIS(Global Embedded Processor Debug Interface Standard)는 현재 매입형 시스템들의 다이나믹 디버그을 실행하고 다른 문제들 및 실시간 비관입(non-intrusive) 디버그 지원(support)과 관련된 이슈(issue)들을 해결하기 위해 개발 중에 있다. 그밖에 IEEE-ISTO 포럼 5001(Institute of Electrical and Electronics Engineers-Industry Standards and Technology Organization)로서 공지된 GEPDIS는 매입형 시스템 프로세서의 코어를 정지함이 없이 비관입 디버깅을 위한 방법을 표준화하기 위해 개발된 매입형 디버그 인터페이스 표준을 상술한다. 많은 디버그 성능들은 프로그램의 흐름(flow) 및 데이터 흐름에 가시성(visibility)을 제공함으로써 프로그램 실행을 모니터하기 위해 GEPDIS로 규정된다. 이 가시성은 전용 멀티 비트 또는 멀티 핀 시리얼 인터페이스 또는 보조 포트를 넘어서 외부 개발 시스템에 제공된 정보 메시지들의 순차로 구성된다. 이후에 프로그램 흐름 메시지들은 매입형 프로세서의 실질적인 명령 실행 순차를 재구성하기 위해 프로그램의 스태틱 영상과 조합된다. 데이터 흐름 메시지들은 미리 규정된 어드레스 영역들까지 프로세서 판독들 및 기록들을 추적한다.

<24> GEPDIS 다이나믹 디버그는 동기화 메시징 및 데이터 트레이스(trace) 메시징을 포함하는 프로그램 트레이스 메시징을 이용하여 실행된다. 프로그램 트레이스 메시징의 실행은 흐름의 변화(change of flow;COF) 이벤트들을 명령하는 상태 정보(status information)와 관련하여 코어 프로세서에 의해 실행된 명령 페치(fetch)들의 순차를 모니터하고, 직·간접적인 흐름의 변화 이벤트들을 포함할 것을 요구한다. 직접적인 COF 이벤트들은 프로그램 계수기 관련 분기(branch)들을 포함하고, 간접적인 COF 이벤트들은 레지스터 간접 분기들을 포함하고 벡터링(vectoring)은 제외된다. 프로그램 트레이스 동기 메시지들의 실행은 전송될 적당한 인근 명령 어드레스나 현재 실행되고 있는 명령 어드레스를 요구한다. 데이터 트레이스 메시징의 실행은 데이터 액세스 어드레스들을 모니터하고 관련 데이터를 조건부로 제공하는 것을 요한다. 데이터 트레이스 메시징은 하나 이상의 규정된 어드레스 영역들에서의 데이터 판독들 또는 판독된 메시징 및 하나 이상의 규정된 어드레스 영역들에서의 데이터 기록들이나 기록 메시징을 포함한다.

<25> GEPDIS는 실시간 다이나믹 및 비관입(non-intrusive) 디버그 시스템을 상술하기 위한 취지이다. GEPDIS는 가상 버스를 액세스하고 보조 버스 또는 포트를 경유하여 외부 시스템과 대응하는 매입형 디버그 인터페이스를 고려한다. 하지만, GEPDIS는 특히 버스 로딩을 어드레스하거나 전력을 소비하지 않는다. 디버그 인터페이스에 직접적으로 가상 버스 신호들을 제공하는 것은 매우 관입적(intrusive)이고 잘못된 동작을 일으킬 수 있다. 버퍼링은 디버그 인터페이스 영향을 감소시키기 위해 이용될 수 있는데, 여전히 실질적인 성능 영향은 가상 버스의 모든 신호들의 변환들이 디버그 신호들에 반영되기 때문이다. 디버그 버스 상의 전력 소비는 상당하고 심지어 임의의 실시예들에서 과도할 수 있다.

<26> 도 1은 매입형 시스템(103)의 보조 버스(102)에 결합된 외부적인 개발 시스템(101)을 포함하는 디버그 시스템(100)의 개략적인 블록도이다. 개발 시스템(101)은 매입형 시스템(103)의 매입형 프로세서(107)의 프로세서 코어(109)에 의해 실행되는 소프트웨어의 실시간 디버깅 동작들을 실행하기 위해 이용된다. 하지만, 실시간 측정 시스템(도시되지 않음)이 실시간 측정을 실행하기 위한 목적을 위해 보조 버스(102)에 또한 결합될 수 있다는 것이 이해되고, 본 발명의 원리들도 동일하다.

<27> 도시된 본 실시예에서, 매입형 시스템(103)은 집적 회로(IC) 또는 칩 상에 내장되고 디버그 시스템(100)과 관련된 것과 같은 보다 커다란 시스템의 기판 상에 적당한 소켓에 접속된다. 매입된 시스템(103)은 각각의 물리적

버스(115)를 경유해서 매입형 프로세서(107)에 결합된 실시간 디버그 인터페이스(105)를 포함한다. 매입형 프로세서(107)는 중앙처리 장치(CPU) 또는 가상 버스(113)를 경유해서 캐시/MMU(Memory Management Unit; 111)에 결합된 코어(109)를 포함한다. 가상 버스(113)는 코어(109)의 동작을 규정하는 복수의 동작 신호들을 포함한다. 캐시 및 MMU는 보통 개별적인 시스템들이지만, 설명의 편의를 위해 조합해서 도시된다. MMU는 선택적이고 매입형 프로세서(107)의 특정 실시예들 상에 제공되지 않을 수 있다. 본 발명은 많은 다른 변형예들 및 매입형 프로세서의 실시예들을 고려할 수 있고 이러한 변형예들에 의해 제한되지 않는다.

<28> 가상 버스(113)는 캐시/MMU(111)에 코어(109)에 의해 어서팅된 어드레스 신호들을 구비하는 VIRTUAL ADDR 및 코어(109) 및 캐시/MMU(111) 간의 쌍 방향 VIRTUAL DATA 버스를 포함한다. 용어 "가상"은 가상 버스(113)가 비지블(visible)이 아니거나 매입형 프로세서(107)의 외부에서 매입형 시스템(103)의 다른 구성 성분들에 제공되지 않음을 나타낸다. 물리적 버스(115)는 가상 버스(113)와 유사하고 유사한 어드레스 데이터 및 제어 신호들을 포함하지만, 동작 동안 두 개의 버스들간의 동작이 변한다. 캐시/MMU(111) 내부의 캐시는 물리적 버스(115)와 같은 유사한 방법으로 가상 버스(113)가 동작하는 경우에 디스에이블(disable)된다. 하지만, 캐시가 인에이블되면, 물리적 버스(115)는 다른 방법으로 동작하고 코어(109)의 동작을 잘 나타내지 못한다. 코어(109)의 동작을 모니터하거나 디버그하는 가상 버스(113)를 모니터하거나 그 밖의 코어(109)에 의해 실행되는 응용 소프트웨어의 측정 동작들을 실행하는 것이 요구된다.

<29> 가상 버스(113)는 코어(109)에 결합된 가상 버스 게이팅 논리(117)에 제공된다. 가상 버스 게이팅 논리(117)는 코어(109) 내부에 내장되거나 코어(109)로부터 외부적으로 설계될 수 있다. 가상 버스 게이팅 논리(117)는 UPDATE DATA 신호, UPDATE ADDR 신호, DEBUG DATA 버스 및 DEBUG ADDR 버스를 경유해서 실시간 디버그 인터페이스(105)에 결합된다. DEBUG ADDR 버스는 이하에서 더 상술된 속성 신호(attribute signal)들을 포함할 수 있다. UPDATE DATA 신호는 DEBUG DATA 버스 상의 신호들을 선택적으로 구동시키거나 인에이블하는 데이터 디버그 업데이트 신호이고, UPDATE ADDR 신호는 DEBUG ADDR 버스를 선택적으로 구동시키거나 인에이블하는 어드레스 디버그 업데이터 신호이다. 코어(109)에 의해 발생된 PROCESSOR STATUS 신호들은 아래에 상술한 바와 같이 코어(109)의 임의의 상태 이벤트(status event)들을 확인하기 위해 실시간 디버그 인터페이스(105)에 제공된다. PROCESSOR STATUS 신호들은 코어(109)의 트랜잭션 상태(transaction status)를 명령하는 적어도 하나의 코어 트랜잭션 상태 출력 신호를 포함한다.

<30> 보조 버스(102)는 매입형 시스템(103)의 실시간 디버그 인터페이스(105)에 의해 제어된 스케일러블(scalable) 출력 디버그 포트이다. 실시예에서, 보조 버스(102)는 개발 시스템(101)에 의해 코어(109)의 모니터링을 인에이블하는 GEPDIS 에 의해 규정된다. 보조 버스(102)는 외부 하드웨어 인터페이스(119)에 제공되고, 개발 시스템(101) 내부에 개발 워크 스테이션(121)에 디버그 신호들을 제공하기 위해 버퍼들 펌웨어(buffers firmware), 프로세싱 등을 포함한다. 워크 스테이션(121)은 임의의 펌웨어, 하드웨어 또는 소프트웨어를 내장하고, 응용 소프트웨어의 개발 또는 디버그 동작들을 제어하기 위한 개인용 컴퓨터 등과 같이 실행될 수 있다. 워크스테이션(121)에서 오퍼레이터는 다른 작동 중에서 가상 버스(113)의 어드레스 및 데이터 신호들을 모니터하기 위한 실시간 디버그 인터페이스(105)의 동작들을 제어하기 위해 디버그 소프트웨어와 인터페이스한다.

<31> 매입형 시스템(103)은 정규 동작들을 위한 정상 모드 및 디버그 모드를 포함하는 동작의 여러 모드들을 가질 수 있다. 실시간 디버그 인터페이스(105)는 디버그 모드를 촉진하고 디버그 모드의 서브세트들과 같은 여러 선택가능한 모드들을 가질 수 있다. 예를 들어, 실시간 디버그 인터페이스는 프로그램 트레이스 모드 및 데이터 트레이스 모드를 포함하며, 여기서 데이터 트레이스 모드는 또한 디버그 모드들을 판독하고 기록할 수 있다.

<32> 도 2는 본 발명의 실시예에 따라 가상 어드레스 버스 게이팅 논리(200)의 개략적인 단면도이다. 가상 어드레스 버스 게이팅 논리(200)는 DEBUG ADDR 버스에 가상 버스(113)의 VIRTUAL ADDR 버스를 결합시키기 위해 가상 버스 게이팅 논리(117) 내부에 제공된다. 상술한 실시예에서, VIRTUAL ADDR 버스는 VIRTUAL ADDR [31:0]으로 표시된 32개 각각의 어드레스 신호들을 포함한다. 하지만, 본 발명은 어드레스 신호들의 임의 특정 수에 제한되지 않는다. 각각의 VIRTUAL ADDR 버스 신호는 D 타입 래치들과 같은 래치들의 임의의 적당한 타입일 수 있는 일련의 래치(201)들의 해당 입력에 제공된다. 래치(201)들 각각의 Q 출력은 일련의 버퍼들(203)의 각 입력에 제동된다. 버퍼들(203)의 출력들은 각 신호들 DEBUG ADDR [31:0]을 포함하는 DEBUG ADDR 버스의 각 신호들을 어서트한다. 매입형 프로세서(107)는 2 입력 AND 게이트(205) 중 하나의 입력에 제공되는, CLK1인 내부 마스터 클록 신호(internal master clock signal)를 포함한다. CLK1 신호는 제어 및 타이밍 목적을 위한 코어(109)에 제공되는 적어도 하나의 마스터 클록 신호이다. AND 게이트(205)의 다른 입력은 UPDATE ADDR 신호를 수신한다. AND 게이트(205)의 출력은 래치들(201) 각각의 클록 입력에 제공되는 신호 DEBUG ADDR CLK를 어서트한다.

<33> 동작에서, UPDATE ADDR 신호는 VIRTUAL ADDR 버스의 각 신호들을 CLK1 신호의 어서팅에 동기화된 DEBUG ADDR 버스로 전송하는 인에이블 신호로서의 역할을 한다. 이러한 방법에서, UPDATE ADDR 신호가 어서팅될 때, CLK1 신호는 VIRTUAL ADDR [31:0] 신호들을 래치하는 래치들(201)을 DEBUG ADDR[31:0] 신호들의 각 신호들에 클록한다. UPDATE ADDR 신호들이 부정되거나 논리 제로일 때, 래치들(201)은 DEBUG ADDR[31:0] 신호가 안정한 상태로 남아있도록 스태틱(static) 상태이다. DEBUG ADDR 버스를 포함하는 매입형 프로세서(107) 버스들 일부의 변환들은 상당량의 전력을 소비한다는 것이 주지된다. 아래에 상술된 바와 같이, DEBUG ADDR 버스의 불필요한 변환들을 줄이거나 제거하기 위해 UPDATE ADDR 신호를 사용하면 상당한 전력의 절약을 달성할 수 있다.

<34> 도 3은 본 발명의 실시예에 따라 실행된 가상 데이터 버스 게이팅 논리(300)의 개략적인 도면이다. 가상 데이터 버스 게이팅 논리(300)는 DEBUG DATA 버스에 가상 버스(113)의 각 데이터 신호들을 선택적으로 전달하기 위해 가상 게이팅 논리(117)에 또한 내장된다. 도시된 실시예에서, 가상 버스(113)의 VIRTUAL DATA 버스는 32개의 신호들, 즉 VIRTUAL DATA [31:0]을 포함한다. 또한, DEBUG DATA 버스는 32개의 신호들, 즉 DEBUG DATA [31:0]을 포함한다. 하지만, 본 발명은 임의 특정 수의 데이터 신호들에 제한됨이 없다는 것이 이해된다. UPDATE DATA 신호는 복수의 2 입력 NAND 게이트들(301) 각각의 하나의 입력에 제공된다. NAND 게이트들(301) 각각의 다른 입력은 VIRTUAL DATA [31:0] 신호들 중 각각의 신호를 수신한다. 예를 들어, 제 1의 NAND 게이트(301)는 VIRTUAL DATA [0] 신호를 수신하고, 제 2의 NAND 게이트(301)는 VIRTUAL DATA [1] 신호 등을 수신한다. NAND 게이트들(301) 각각의 출력는 복수의 인버터들(303)의 중 각 인버터의 입력에 제공된다. 인버터들(303) 각각은 그들 각각의 출력들에 DEBUG DATA [31:0] 신호들의 각 신호들 중 각각의 신호를 어서트한다.

<35> 가상 데이터 버스 게이팅 논리(300)의 동작은 CLK1 신호에 동기화가 없다는 것을 제외하고는 VIRTUAL ADDRESS 버스 게이팅 논리(200)과 유사하다. 하지만, 가상 어드레스 버스 게이팅 논리(200)는 가상 데이터 게이팅 논리(300)처럼 유사한 방법으로 실행되며, 그 역도 동일하다. 상술한 특정 실시예에서, 가상 어드레스 버스 게이팅 논리(200)는 다소 사이클이 실재로 유효하고 적당히 래치되도록하는 어드레스를 지연시킨다. 또다른 이익은 DEBUG ADDR 버스의 타이밍 및 신호들의 중요성이 감소되고, 버퍼들(203)이 보다 작고 더 적은 전력 소비용 소자들을 이용하여 실행될 수 있다는 것이다. UPDATE DATA 신호가 로우(low)로 부정되면, DEBUG DATE 버스는 스태틱 상태로 되고 변화되지 않는다. UPDATE DATA 신호가 하이(high)로 어서트될 때, DEBUG DATA 버스는 논리적으로, 논리 게이트들(301, 303)을 통해 조금 지연된 후 가상 버스(113)의 VIRTUAL DATA 버스를 뒤따른다. 특히, UPDATE DATA 신호가 하이로 어서트될 때, DEBUG DATA[0] 신호는 virtual data[0] 신호를 뒤따르고, DEBUG DATA[1] 신호들은 가상 데이터[1] 신호 등을 뒤따른다.

<36> 가상 어드레스 버스 게이팅 논리(200) 및 가상 데이터 버스 게이팅 논리(300)는 각각 UPDATE ADDR 신호 및 UPDATE DATA 신호를 경유하여 대응하는 DEBUG ADDR 버스 및 DEBUG DATA 버스의 선택적인 제어 및 시동(activation)을 제공한다. 이 방법에서, UPDATE ADDR 버스 및 UPDATE DATA 신호들은 DEBUG ADDRESS 및 DEBUG DATA 버스들의 전력 소비를 제어하기 위해 이용될 수 있다. 차례로, 이는 디버그, 시험 또는 측정 동작 동안 매입형 시스템(103)의 전력 소비의 제어를 인에이블한다. 특정 배터리로 동작하는 호출기들 및 이동 전화기들 등과 같은 휴대용 장치들에서, 디버그 모드는 정규 동작 모드에서 디스인에이블된다. UPDATE ADDR 및 UPDATE DATA 신호들은 DEBUG ADDR 및 DEBUG DATA 버스들이 스위치되지 않도록 로우(low)로 강제된다. 이는 부가적인 디버그 모드 논리 또는 회로가 귀중한 배터리 전력을 소비하지 않도록하여 배터리의 수명에 영향을 미치지 않을 것이다.

<37> 실시간 디버그 인터페이스(105)는 DEBUG DATA 버스 및 DEBUG ADDR 버스의 디버그 신호들의 샘플링을 필요로 할 때 UPDATE DATA 및 UPDATE ADDR 신호들을 어서트한다. 또한, 다양한 속성 신호들은 코어(109)로부터 PROCESSOR STATUS 신호들로 도시된 실시간 디버그 인터페이스(105)로 제공된다. 개발 시스템(101)의 제어 하에서 실시간 디버그 인터페이스(105)는 실시간 디버그 인터페이스(105)에 의한 모니터링을 위해 DEBUG DATA 및 DEBUG ADDR 신호들을 구동시키거나 인에이블하는 UPDATE DATA 및 UPDADE ADDR 신호들을 어서트할 때를 결정한다. 특정 실시예들에서, 실시간 디버그 인터페이스(105)는 흐름(COF) 변환 이벤트들 동안 코어(109)에 의해 실행된 명령 페치들의 순차를 모니터하는 프로그램 트레이스를 구동시키거나 인에이블할 수 있다. PROCESSOR STATUS 신호들은 COF 상태를 지시하는 반면에 가상 버스(113)의 가상 어드레스 버스는 COF 어드레스를 어서트한다. PROCESSOR STATUS 신호들에 의해 지시된 바와 같이, COF 상태 또는 이벤트 동안, 실시간 디버그 인터페이스(105)는 DEBUG ADDR 버스를 구동하고 샘플링하기 위해 UPDATE ADDR 신호를 어서트한다. 동기화 메시지들은 또한 실시간 디버그 인터페이스(105)가 UPDATE ADDR 신호를 어서트하는 프로그램 트레이스 동작의 일부분일 수 있다.

<38> 데이터 트레이스 메시지들은 데이터 액세스 어드레스들을 모니터하는 단계 및 관련된 데이터를 상태에 따라 제공하는 단계를 포함한다. 데이터 트레이스는 하나 이상의 규정된 어드레스 영역들 중 임의의 영역에서 데이터

판독들 및 기록들을 위해 유용하다. 데이터 트레이스 메시징이 인에이블될 때, 실시간 디버그 인터페이스(105)는 판독 또는 기록 사이클들에 포함된 데이터 페치들을 위한 PROCESSOR STATUS 신호들을 모니터한다. PROCESSOR STATUS 신호들은 또한 판독 또는 기록 사이클이 일어나고 있는 지를 지시한다. 판독 또는 기록 메시징을 위해, 실시간 디버그 인터페이스(105)는 적당한 데이터 사이클 동안 해당 어드레스를 붙잡기 위해 데이터 페치동안 UPDATE ADDR 신호를 어서트한다. 예를 들어, 판독 메시징을 위해, 실시간 디버그 인터페이스(105)는 프로세서 상태 버스가 판독 사이클을 지시하면 UPDATE ADDR 신호를 어서트한다. 게다가, 실시간 디버그 인터페이스(105)는 프로세서 상태 버스가 기록 메시징을 위해 기록 사이클을 명령하면 UPDATE ADDR 신호를 어서트한다. 물론, 실시간 디버그 인터페이스(105)는 판독 및 기록 메시징이 모두 인에이블되면 판독 및 기록 신호들에 대한 UPDATE ADDR 신호를 어서트한다. 또한, 데이터 트레이스가 인에이블될 때, 실시간 디버그 인터페이스(105)는 판독 또는 기록 메시징을 위한 하나 이상의 규정된 어드레스 영역들 중 임의의 어드레스와 DEBUG ADDR 버스로부터 검출된 어드레스를 비교한다. 어드레스 정합이 판독 또는 기록 사이클 동안 발견될 때, 실시간 디버그 인터페이스(105)는 가상 버스 게이팅 논리(117)가 DEBUG DATA 버스 상의 대응하는 데이터를 어서트하도록하는 UPDATE DATA 신호를 어서트한다.

<39> 도 4는 본 발명의 실시예에 따른 프로그램 트레이스 수빅 어드레스 게이팅 동작을 도시하는 타이밍도이다. 코어(109) 마스터 클록 신호들인 CLK1 및 CLK2는 매입형 시스템(103)에 대한 타이밍 참조부호와 같이 도시된다. 타이밍도는 ADDRESS1, ADDRESS2, COF ADDR, ADDRESS4, 및 ADDRESS5로 표시된 5개의 어드레스들의 어서트를 설명하는 가상 버스(113)의 VIRTUAL ADDR를 도시한다. 타이밍도는 또한 DATA1, DATA2, DATA3, DATA4, 및 DATA5로 각각 도시된 5개의 해당 데이터 신호들을 나타내는 VIRTUAL DATA 버스를 도시한다. 또한 PROCESS STATUS 신호들은 명령 페치, COF 이벤트, 데이터 페치 및 다른 명령 페치의 순서를 나타내는 것을 도시한다. 또한 타이밍도는 UPDATE ADDR 신호, DEBUG ADDR CLK 신호 및 DEBUG ADDR 버스를 도시한다.

<40> 도 4의 예에서, 프로그램 트레이스는 데이터가 디스인에이블되는 동안 인에이블된다. 데이터 트레이스가 인에이블되므로, VIRTUAL DATA 버스 상의 데이터를 모니터할 필요가 없다. 그래서, 실시간 디버그 인터페이스(105)는 DEBUG DATA 버스가 VIRTUAL DATA 버스 상의 활동성(activity)에 상관없이 정적이고 비활성(inactive)되도록 로우(low)로 부정된 UPDATE DATA 신호를 유지한다. 이는 전력이 DEBUG DATA 버스를 스위칭하여 불필요하게 소비되지 않도록 전력 소비가 상당히 감소되는 즉각적인 이익을 제공한다.

<41> 실시간 디버그 인터페이스(105)는 VIRTUAL ADDR 버스 상의 모든 활동성을 모니터할 필요가 없다. VITUAL ADDR 버스에 어서트된 많은 어드레스들은 앞서 어서트된 어드레스의 증가된 버전(version)들이다. 이 방법에서, 일단 일련의 순차적인 어드레스들의 초기 어드레스가 알려지면, 남아있는 어드레스들도 알려지거나, 어드레스들이 DEBUG ADDR 버스에 전달될 필요성이 없다는 것을 예상할 수 있다. 이는 이원 순차가 알려지면, 순차적인 어드레스들이 필연적으로 증대되지 않는 전략들을 포함하는 임의의 어드레싱 전략에 대해 정확하다. 실시간 디버그 인터페이스(105)는 단지 조건적이거나 비조건적인 분기(branch) 및 루프 명령들이나 임의의 직·간접적인 COF 이벤트들에 응답하는 것과 같은 순차적인 순서를 따르지 않는 어드레스들을 확인하고 판독할 것을 요한다. PROCESSOR STATUS 신호들 상에 어서트된 COF 명령은 비순차적인 어드레스들을 확인한다.

<42> 도 4에 도시된 바와 같이, DEBUG ADDR 버스는 순자적인 어드레스들인 ADDRESS1 및 ADDRESS2가 VIRTUAL ADDR 버스 상에 어서트될 지라도 마지막의 COF 어드레스를 갖고 정적으로 유지된다. 특히, 실시간 디버그 인터페이스(105)가 PROCESSOR STATUS 신호들 상의 COF 명령를 검출하고, 이벤트 화살표(401)로 명령된 바와 같은 UPDATE ADDR 신호를 어서트한다. UPDATE ADDR 신호의 어서트는 도 2에 도시된 래치(201)들을 인에이블한다. UPDATE ADDR 신호가 어서트되는 동안 CLK1 신호의 다음 융기한 단부에서, DEBUG ADDR CLK 신호는 405에 지시된 바와 같이 어서트된다. DEBUG ADDR CLK 신호의 어서트는 래치들(201)을 인에이블하여, VIRTUAL ADDR 버스 신호들이 이벤트 화살표(407)에 의해 도시된 바와 같은 DEBUG ADDR 버스 신호들 상에 구동된다.

<43> 이 방법에서, DEBUG ADDR 버스는 단지 실시간 디버그 인터페이스(105)에 의해 어서트된 UPDATE ADDR 신호에 응답하여 마지막 COF 어드레스로부터 새로운 COF 어드레스로 스위치한다. PROCESSOR STATUS 신호들 상의 COF 명령은 이벤트 화살표(409)에 의해 명령된 바와 같은 CLK2 신호의 다음 융기한 단부 상에서 완성된다. 실시간 디버그 인터페이스(105)는 이벤트 화살표(411)에 의해 지시된 바와 같은 COF 명령의 완성에 응답하여 UPDATE ADDR 신호를 어서트하지 않는다. 응답에서, DEBUG ADDR CLK 신호는 이벤트 화살표(413)에 의해 도시된 바와 같이 부정된다. DEBUG ADDR CLK 신호가 이후에 부정된채로 유지되므로, DEBUG ADDR 버스는 새로운 COF 어드레스와 함께 변화되지 않고 정적으로 유지되어, 전력을 감소시킨다.

<44> 도 5는 본 발명의 실시예에 따른 프로그램, 데이터 트레이스 가상 어드레스 및 데이터 게이팅을 도시하는 타이

밍도이다. 이 예에서, 프로그램 트레이스 및 데이터 트레이스는 인에이블된다. 프로그램 트레이스가 인에이블되므로, 실시간 디버그 인터페이스(105)는 도 4를 참조로 이전에 상술된 바와 같이 COF 명령을 어서트 할 때, VIRTUAL ADDR 버스를 모니터한다. 데이터 트레이스가 인에이블되므로, 판독 및 기록 데이터 트레이스가 인에이블된다고 가정하면, 실시간 디버그 인터페이스(105)는 VIRTUAL DATA 버스 상에서 일어나는 상당한 데이터 이벤트들을 모니터한다. 상당한 데이터 이벤트들이 워크스테이션(121)에 의해 규정되고 모니터하는 동안 실시간 디버그 인터페이스(105)에 전달된다. 예를 들어, 특정 어드레스 또는 복수의 레지스터들 또는 특정 어드레스 영역 내부에서 메모리 장치의 임의의 다른 종류에서 특정 레지스터에 기록되거나 특정 레지스터로부터 판독된 임의의 데이터를 확인하는 것이 요구될 수 있다. 다수의 이러한 어드레스들 또는 어드레스 영역들은 규정될 수 있다.

<45> 실시간 디버그 인터페이스(105)는 데이터 트레이스가 인에이블되는 동안 임의의 DATA FETCH 명령들에 대해 PROCESSOR STATUS 신호들을 모니터한다. DATA FETCH 명령이 검출될 때, 실시간 디버그 인터페이스는 VIRTUAL ADDR 버스로부터 DEBUG ADDR 버스로 판독 또는 기록 어드레스를 붙잡기 위해 UPDATE ADDR 신호를 어서트한다. 이후에 실시간 디버그 인터페이스(105)는 모니터될 임의 및 모든 이용가능한 어드레스들 또는 어드레스 영역들과 DEBUG ADDR 버스로부터 검출된 어드레스를 비교한다. 어드레스는 모니터된 어드레스 또는 어드레스 영역과 같다면, 실시간 디버그 인터페이스(105)는 데이터 이벤트가 충분하다면 DEBUG DATA 버스에 VIRTUAL DATA 버스 상에서 어서트된 데이터를 전송하게 하는 UPDATE DATA 신호를 어서트한다. 판독 및 기록이 특정 어드레스 영역에 대해 모니터되면, 이후에 데이터 이벤트는 충분히 고려된다는 것이 주지된다. 그밖에, 실행되는 사이클의 종류가 모니터되는 사이클의 종류와 동일하기만 하면 데이터 이벤트는 충분하다.

<46> 실시간 디버그 인터페이스(105)는 또한 데이터 사이클이 판독 또는 기록될지 PROCESSOR STATUS 신호들로부터 결정한다. 단지 판독 데이터 트레이스가 인에이블되고 기록 사이클이 검출되거나 그 역이면, 실시간 디버그 인터페이스(105)는 데이터 이벤트가 충분하다고 여겨지지 않기 때문에, UPDATE ADDR 신호를 어서트하지 않을 수 있다. 판독 및 기록 데이터 트레이스가 인에이블된다 할지라도, 특정 어드레스 영역에 대한 판독 또는 기록 사이클 간의 구별이 필요하게 될 수 있다. 예를 들어, 하나의 어드레스 범위로부터 판독 사이클 및 제 2의 다른 어드레스 범위에 기록 사이클을 식별하는 것이 요구될 수 있다. 일 실시예에서, 실시간 지연 인터페이스(105) 내부의 비교 논리는 이용가능한 모니터된 어드레스 영역들을 결정하기 위해 사이클 종류를 이용한다. 물론, 판독 및 기록이 모든 규정된 어드레스 영역들에 대해 인에이블되면, 판독 및 기록 사이클 간의 구별이 불필요하게 될 수 있다.

<47> 도 5에 도시된 바와 같이, COF 명령은 PROCESSOR STATUS 신호들 상에 나타나고, 이 신호들은 실시간 디버그 인터페이스(105)에 의해 검출된다. 실시간 디버그 인터페이스(105)는 이벤트 화살표(501)에 의해 지시된 바와 같이 UPDATE ADDR 신호를 대응하게 어서트한다. UPDATE ADDR 신호가 어서트되는 동안, CLK1의 어서트들은 이벤트 화살표(503)에 의해 지시된 바와 같이 DEBUG ADDR CLK 신호의 수반하는 어서트들을 야기한다. CLK2 신호의 다음 어서트는 COF 명령으로부터 DATA FETCH 명령으로 PROCESSOR STATUS 신호들 상의 변환을 야기한다. PROCESSOR STATUS 신호들 상의 DATA FETCH 명령은 실시간 디버그 인터페이스(105)에 의해 검출되고, 실시간 디버그 인터페이스는 이 예에서 데이터 트레이스가 인에이블되므로 이벤트 화살표(507)에 의해 지시된 바와 같이 어서트된 UPDATE ADDR 신호를 유지한다. UPDATE ADDR 신호가 어서트된 채로 유지되므로, DEBUG ADDR CLK 신호의 제 2 어서트는 이벤트 화살표(511)에 의해 명령된 바와 같이 DEBUG ADDR 버스에 래치될 VIRTUAL ADDR 버스 상에 데이터 기록 어드레스가 나타나도록 한다. 데이터 트레이스가 인에이블되므로, DEBUG ADDR 버스 상에 어서트된 데이터 기록 어드레스(DATA WR ADDR)는 실시간 디버그 인터페이스(105)에 의해 검출되고, 실시간 디버그 인터페이스는 검출된 어드레스와 그 시간에 구동된 판독 또는 기록 메시징을 위한 임의의 특정 규정된 어드레스 영역들을 비교한다. 도 5에 도시된 예에서, 실시간 디버그 인터페이스(105)는 데이터 기록 어드레스와 액티브 어드레스 영역들을 비교하고 디버그 어드레스 비교 정합을 결정하고 이벤트 화살표(513)에 의해 지시된 바와 같은 UPDATE DATA 신호를 대응적으로 어서트한다. 실시간 디버그 인터페이스(105)에 의한 UPDATE DATA 신호의 어서트로 인해 VIRTUAL DATA 버스 상의 신호들이 결국 이벤트 화살표(515)에 의해 명령되는 바와 같은 DEBUG DATA 버스 상에 기록 데이터를 어서트되게 하는 논리 게이트들(301, 303)을 경유하여 DEBUG DATA 버스에 전달되도록 한다.

<48> 논리 게이트(205, 301), 래치들(201) 및 버퍼들(203, 303)은 가상 버스(113) 상에서 로딩을 감소시키는 이점을 제공하는 아이솔레이션(isolation)을 제공한다는 것이 이해된다. DEBUG ADDR 및 DEBUG DATA 버스들의 스위칭을 줄이기 위해 UPDATE ADDR 및 UPDATE DATA 신호들을 사용은 가상 버스(113)의 로딩을 감소시키고 또한 매입형 프로세서(107) 및 매입형 시스템(103)에 대한 상당한 전력을 절약하는 이점을 더 제공한다.

<49> 본 발명에 따른 시스템 및 방법이 바람직한 실시예와 관련하여 상술되었지만, 본 명세서의 특정 형태로 제한될 의도는 아니며, 이와는 반대로, 첨부된 청구항들에 의해 정의된 바와 같이 본 발명의 정신과 범위에 포함될 수

있는 이러한 대안들, 변형예들, 및 대응물들을 보호하려는 것이다.

발명의 효과

<50> 매입형 프로세서에 대한 게이팅 논리는 매입형 시스템 칩이 디버그 인터페이스 또는 집적 회로(IC)에 어드레스 및 데이터 신호들을 포함하는 코어 프로세서의 가상 버스 신호들을 제공한다. 게이팅 논리는 해당 디버그 어드레스 및 데이터 버스들의 스위칭을 제어하기 위해 디버그 인터페이스로부터 업데이트 및 업데이트 어드레스 신호들을 수신한다. 코어 프로세서는 디버그 인터페이스가 코어 프로세서에 의해 실행되는 특정 트랜잭션을 결정하도록 디버그 인터페이스에 프로세서 상태 신호들을 제공한다. 프로세서 상태 신호(processor status signal)들은 명령 페치 또는 데이터 페치 사이클들을 뿐만 아니라 흐름의 변화(COF) 이벤트들을 나타낸다. 프로세서 상태 신호들은 또한 데이터 사이클이 판독인지 기록인지를 나타낸다. 디버그 인터페이스는 상당한 이벤트들을 확인하고 가상 버스를 샘플링함에 따라 업데이트 데이터 또는 업데이트 어드레스 신호들을 어서트하는 프로세서 상태 신호들을 모니터한다. 이러한 방법에서, 프로그램 트레이스 메시징 동안, 디버그 인터페이스는 COF 이벤트 동안 디버그 어드레스 버스를 경유하여 가상 어드레스 버스를 샘플링한다. 또한, 디버그 인터페이스는 데이터 페치 사이클들을 검출하고, 가상 어드레스 버스를 샘플링하고, 하나 이상의 미리결정된 관련 어드레스들이나 어드레스 영역들에 대한 어드레스를 비교한다. 충분한 데이터 사이클의 어드레스가 미리규정된 영역 내에 있으면, 디버그 인터페이스는 가상 데이터 버스를 샘플링하기 위해 업데이트 데이터 신호를 어서트한다.

도면의 간단한 설명

<1> 도 1은 본 발명에 따른 실시간 프로세서 디버그 시스템을 설명하는 디버그 시스템의 개략적인 블록도.

<2> 도 2는 본 발명의 실시예에 따라 구현된 가상 어드레스 버스 게이팅(gating) 논리의 개략도.

<3> 도 3은 본 발명의 실시예에 따라 구현된 가상 데이터 버스 게이팅 논리의 개략도.

<4> 도 4는 본 발명의 실시예에 따른 실시간 디버그 인터페이스 시스템의 프로그램 트레이스(trace) 가상 어드레스 게이팅 동작을 도시하는 타이밍도.

<5> 도 5는 본 발명의 실시예에 따른 실시간 디버그 인터페이스 시스템의 데이터 게이팅 동작 및 데이터 트레이스(trace) 가상 어드레스 및 프로그램을 도시하는 타이밍도.

<6> * 도면의 주요부분에 대한 부호의 설명 *

<7> 100 : 디버그 인터페이스 101 : 외부 개발 시스템

<8> 102 : 보조 버스 109 : 프로세서 코어

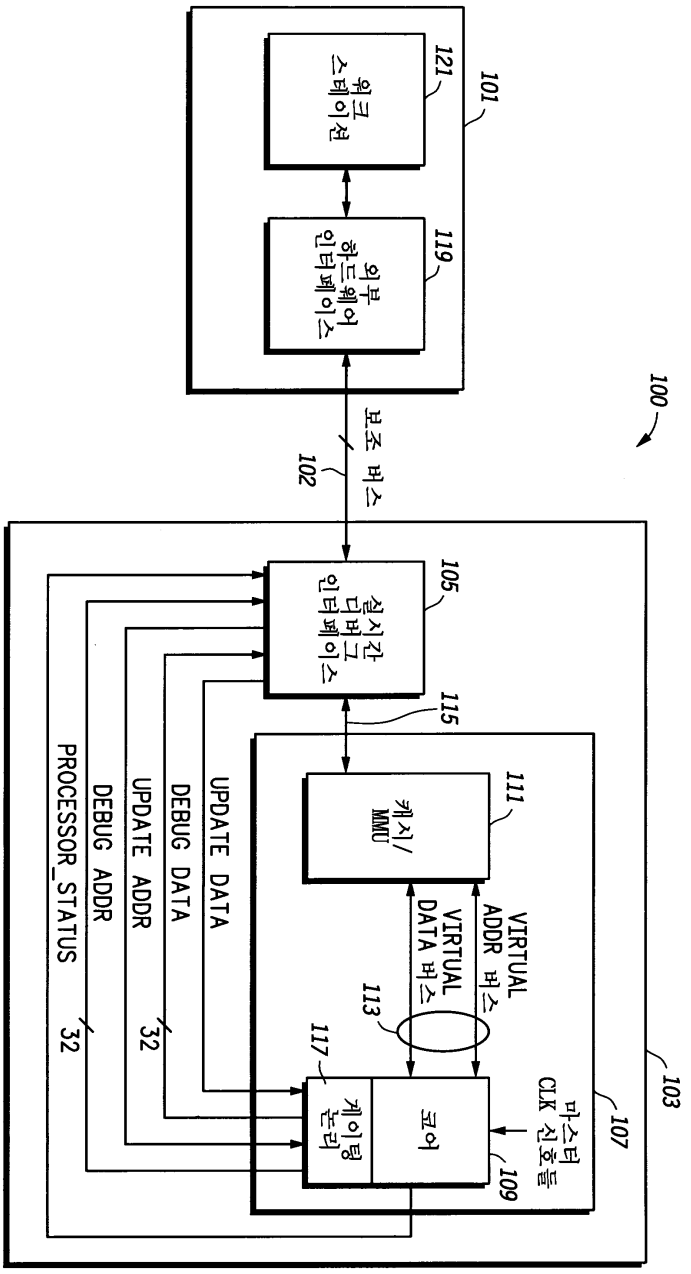
<9> 103 : 매입형 프로세서 113 : 가상 버스

<10> 117 : 가상 버스 게이팅 논리 121 : 개발 워크스테이션

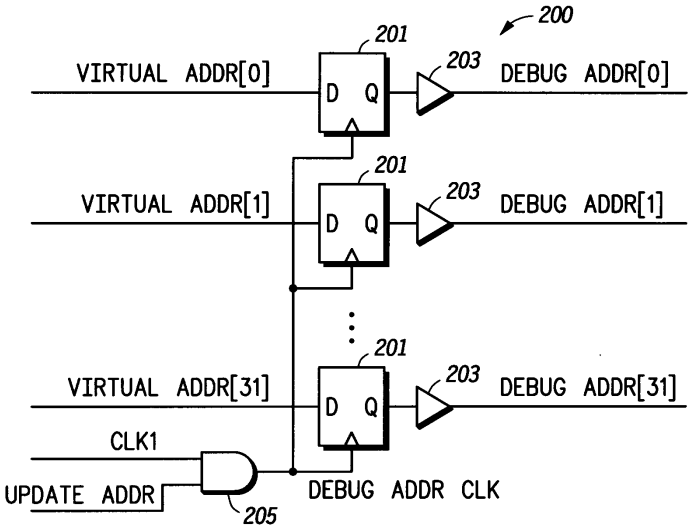
<11> 201 : 래치 203 : 버퍼

도면

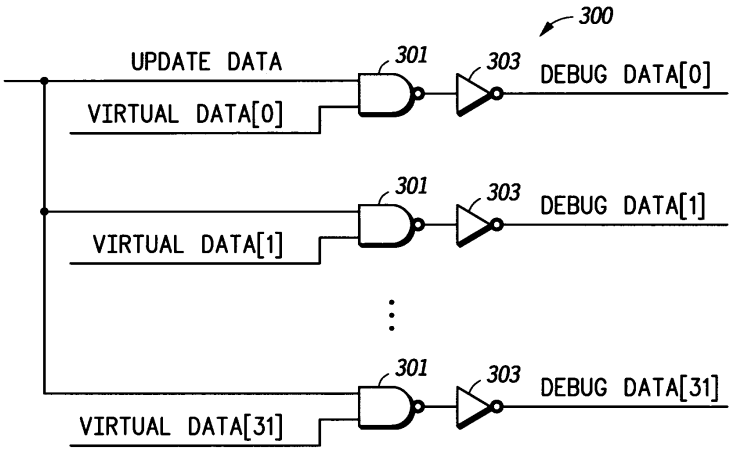
도면1


100
101
121
워크스테이션
119
외부 하드웨어 인터페이스
보조 버스
102
103
105
실시간 디버그 인터페이스
115
111
캐시/MMU
VIRTUAL ADDR 버스
VIRTUAL DATA 버스
113
117
게이팅 논리
코어
107
마스터 CLK 신호들
109
UPDATE DATA
DEBUG DATA
UPDATE ADDR
DEBUG ADDR
PROCESSOR_STATUS
32
32

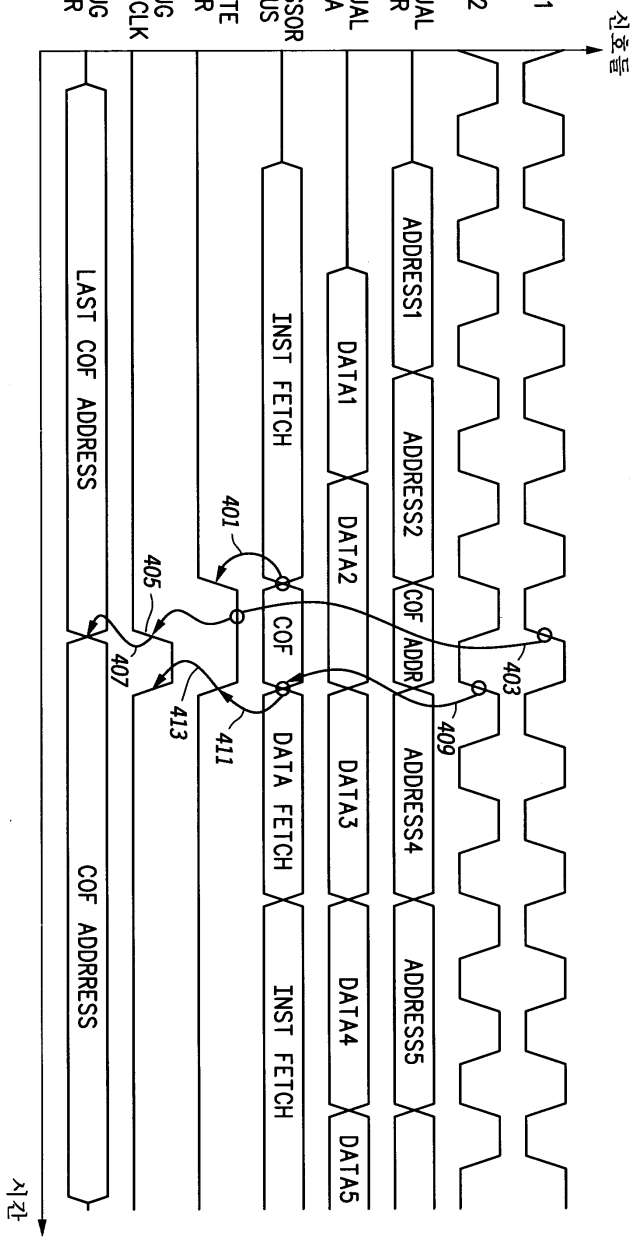
도면2


200
201
203
VIRTUAL ADDR[0]
D Q
DEBUG ADDR[0]
VIRTUAL ADDR[1]
DEBUG ADDR[1]
VIRTUAL ADDR[31]
DEBUG ADDR[31]
CLK1
UPDATE ADDR
205
DEBUG ADDR CLK

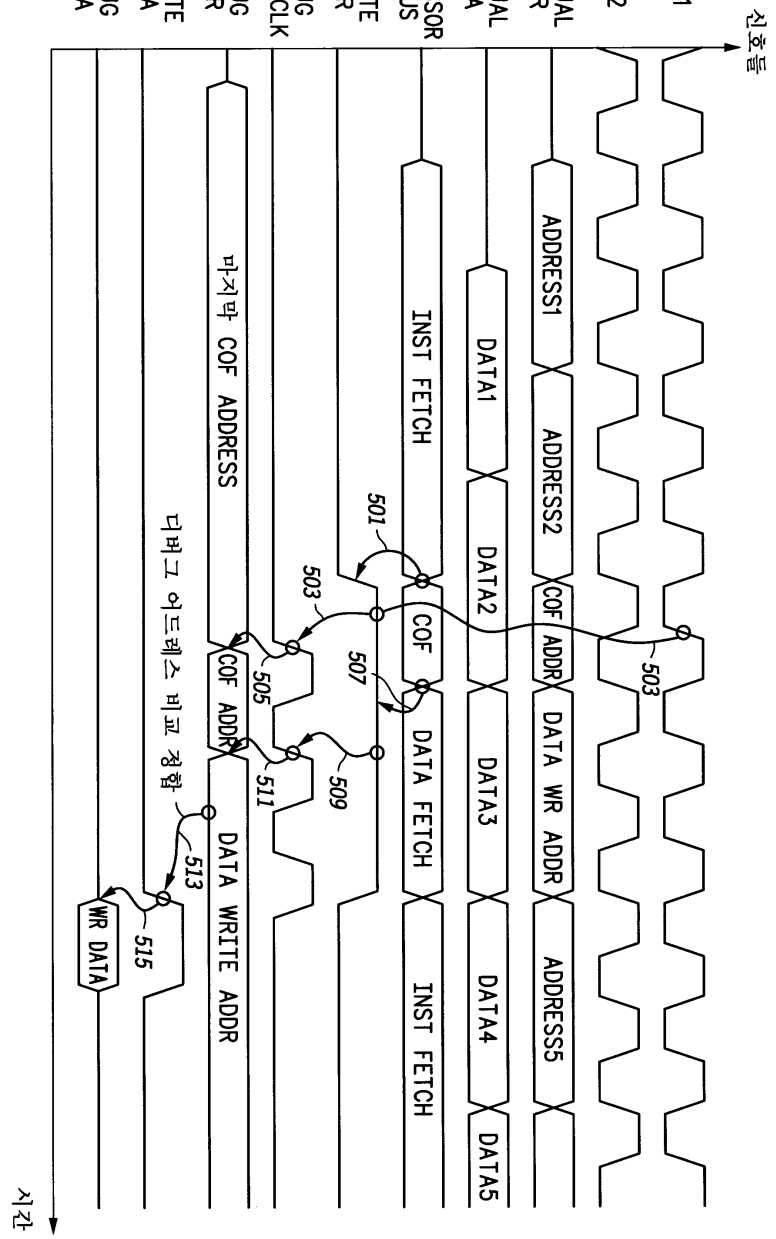
도면3


300
UPDATE DATA
301
303
DEBUG DATA[0]
VIRTUAL DATA[0]
DEBUG DATA[1]
VIRTUAL DATA[1]
DEBUG DATA[31]
VIRTUAL DATA[31]

도면4


신호들
CLK1
CLK2
VIRTUAL ADDR
VIRTUAL DATA
PROCESSOR STATUS
UPDATE ADDR
DEBUG ADDR CLK
DEBUG ADDR
ADDRESS1
ADDRESS2
COF ADDR
ADDRESS4
ADDRESS5
DATA1
DATA2
DATA3
DATA4
DATA5
INST FETCH
COF
DATA FETCH
INST FETCH
LAST COF ADDRESS
COF ADDRRESS
401
403
405
407
409
411
413
시간

도면5


신호들
CLK1
CLK2
VIRTUAL ADDR
VIRTUAL DATA
PROCESSOR STATUS
UPDATE ADDR
DEBUG ADDR CLK
DEBUG ADDR
UPDATE DATA
DEBUG DATA
ADDRESS1
ADDRESS2
COF ADDR
DATA WR ADDR
ADDRESS5
DATA1
DATA2
DATA3
DATA4
DATA5
INST FETCH
COF
DATA FETCH
INST FETCH
마지막 COF ADDRESS
COF ADDR
DATA WRITE ADDR
디버그 어드레스 비교 결합
WR DATA
501
503
505
507
509
511
513
515
시간