(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2015-0008441
(43) 공개일자 2015년01월22일

(51) 국제특허분류(Int. Cl.)
G06F 11/36 (2006.01)
(21) 출원번호 10-2014-7033969
(22) 출원일자(국제) 2013년05월07일
심사청구일자 없음
(85) 번역문제출일자 2014년12월03일
(86) 국제출원번호 PCT/US2013/039934
(87) 국제공개번호 WO 2013/169766
국제공개일자 2013년11월14일
(30) 우선권주장
61/643,725 2012년05월07일 미국(US)

(71) 출원인
마이크로칩 테크놀로지 인코포레이티드
미국 85224-6199 아리조나 챈들러 웨스트 챈들러 블러바드 2355
(72) 발명자
밀크스, 저스틴
미국, 85286 애리조나, 챈들러, 이스트 우즈맨 플레이스 630
페르메, 토마스, 에드워드
미국, 85225 애리조나, 챈들러, 에스. 유칼립투스 플레이스 222
(뒷면에 계속)
(74) 대리인
특허법인세신

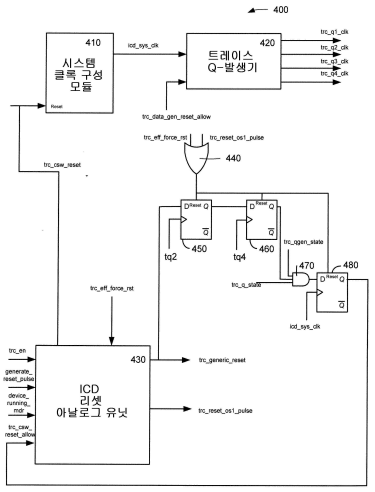
전체 청구항 수 : 총 20 항

(54) 발명의 명칭 **리셋 조건 트레이스 기능들을 갖춘 프로세서 디바이스**

(57) 요 약

디버그 기능들을 갖는 프로세서 디바이스는, 중앙 처리 유닛, 트레이스 모듈을 포함하는 디버그 회로부, 내부 클록 신호들을 제공하기 위한 시스템 클록 모듈, 및 디버그 모드 동안에 상기 시스템 클록 모듈이 리셋 신호를 수신하는 것을 방지하는 리셋 검출 유닛을 구비한다.

대 표 도 - 도4



(72) 발명자

발라수브라마니안, 순다르

미국, 85286 애리조나, 챈들러, 더블유. 블루버드 드라이브 1163

자바갈, 쿠샬라

미국, 85044 애리조나, 피닉스, 아파트 2050, 이. 레이 로드 3625

특허청구의 범위

청구항 1

디버그 기능들을 갖는 프로세서 디바이스로서,

중앙 처리 유닛;

트레이스 모듈을 포함하는 디버그 회로부;

내부 클록 신호들을 제공하기 위한 시스템 클록 모듈; 및

디버그 모드 동안에 상기 시스템 클록 모듈이 리셋 신호를 수신하는 것을 방지하는 리셋 검출 유닛을 포함하는, 프로세서 디바이스.

청구항 2

제 1 항에 있어서,

상기 트레이스 모듈은 리셋 신호가 검출된 후 트레이스 정보를 기록하도록 동작 가능하고, 그리고 상기 트레이스 정보를 기록한 후에 상기 시스템 클록 모듈을 리셋하는 신호를 생성하는, 프로세서 디바이스.

청구항 3

제 2 항에 있어서,

상기 트레이스 정보는 리셋 소스 정보를 포함하는, 프로세서 디바이스.

청구항 4

제 1 항에 있어서,

상기 트레이스 모듈은 실행된 명령어들에 관한 정보를 포함하는 트레이스 스트림을 생성하고, 상기 트레이스 스트림은 외부 인터페이스를 통해 출력되는, 프로세서 디바이스.

청구항 5

제 4 항에 있어서,

상기 트레이스 스트림은 패킷 기반인, 프로세서 디바이스.

청구항 6

제 5 항에 있어서,

상기 트레이스 패킷은 트리거 소스에 관한 정보를 포함하는, 프로세서 디바이스.

청구항 7

제 6 항에 있어서,

상기 정보는 조건부로 제공되며, 여기서 조건은 사용자에 의해 정의될 수 있는, 프로세서 디바이스.

청구항 8

제 1 항에 있어서,

상기 리셋 신호는 동기 리셋 신호 또는 비동기 리셋 신호일 수 있는, 프로세서 디바이스.

청구항 9

제 8 항에 있어서,

상기 동기 리셋 신호는 워치독 타이머(watchdog timer; WDT), 상기 중앙 처리 유닛에 의해 실행된 RESET

명령어, 스택 오버플로우/언더플로우 리셋에 의해 생성되는, 프로세서 디바이스.

청구항 10

제 8 항에 있어서,

상기 비동기 리셋 신호는 상기 프로세서 디바이스의 외부 핀을 통해 수신되는, 프로세서 디바이스.

청구항 11

프로세서 디바이스 내에서 실행된 코드를 디버깅하기 위한 방법으로서,

중앙 처리 유닛(CPU)에 의해 코드를 실행하는 단계;

리셋 결정시, 트레이스 모듈의 추가 동작을 가능하게 하기 위해 리셋 신호들을 시스템 클록 모듈을 제외한 마이크로컨트롤러의 내부 유닛들에 전송하는 단계; 및

상기 리셋의 수신 후에 트레이스 정보를 기록하는 단계를 포함하는, 실행된 코드 디버깅 방법.

청구항 12

제 11 항에 있어서,

상기 트레이스 정보가 기록된 후에 상기 시스템 클록 모듈을 리셋하는 단계를 더 포함하는, 실행된 코드 디버깅 방법.

청구항 13

제 12 항에 있어서,

상기 트레이스 정보는 리셋 소스 정보를 포함하는, 실행된 코드 디버깅 방법.

청구항 14

제 11 항에 있어서,

상기 트레이스 모듈은 실행된 명령어들에 관한 정보를 포함하는 트레이스 스트림을 생성하고, 상기 트레이스 스트림은 외부 인터페이스를 통해 출력되는, 실행된 코드 디버깅 방법.

청구항 15

제 14 항에 있어서,

상기 트레이스 스트림은 패킷 기반인, 실행된 코드 디버깅 방법.

청구항 16

제 15 항에 있어서,

상기 트레이스 패킷은 트리거 소스에 관한 정보를 포함하는, 실행된 코드 디버깅 방법.

청구항 17

제 16 항에 있어서,

상기 정보는 조건부로 제공되며, 여기서 조건은 사용자에 의해 정의될 수 있는, 실행된 코드 디버깅 방법.

청구항 18

제 11 항에 있어서,

상기 리셋 신호는 동기 리셋 신호 또는 비동기 리셋 신호일 수 있는, 실행된 코드 디버깅 방법.

청구항 19

제 18 항에 있어서,

상기 동기 리셋 신호는 워치독 타이머(WDT), 상기 중앙 처리 유닛에 의해 실행된 RESET 명령어, 스택 오버플로우/언더플로우 리셋에 의해 생성되는, 실행된 코드 디버깅 방법.

청구항 20

제 18 항에 있어서,

상기 비동기 리셋 신호는 상기 프로세서 디바이스의 외부 핀을 통해 수신되는, 실행된 코드 디버깅 방법.

명 세 서

기 술 분 야

[0001] 본 출원은, "PROCESSOR DEVICE WITH INSTRUCTION TRACE CAPABILITIES"라는 명칭으로 2012년 5월 7일 출원된 미국 가출원 번호 61/643,725 호의 이익을 주장하며, 상기 미국 가출원은 그 전체가 본 출원에 통합된다.

[0002] 본 개시는 프로세서 디바이스들에 관한 것으로, 특히 집적된 디버그 기능들을 구비한 마이크로컨트롤러 디바이스들에 관한 것이다.

배 경 기 술

[0003] 현대의 마이크로프로세서들 및 마이크로컨트롤러들은 소위 회로 내 디버거 또는 에뮬레이터 디바이스에 의해 실행중인 프로그램을 효율적으로 분석할 수 있게 하는 회로부를 포함한다. 이를 위해, 마이크로컨트롤러 또는 마이크로프로세서는 디버깅 기능들을 지원하는 내부의 회로부, 및 예를 들어 디버그 인터페이스로서 동작하기 위해 디바이스의 다수의 기능 핀들을 프로그래밍함으로써 활성화될 수 있는 특정 인터페이스를 제공한다. 보통 이러한 인터페이스는 실제 디바이스와 외부 디버거 또는 에뮬레이터 사이의 빠른 통신을 가능하게 하기 위해 고속 직렬 인터페이스로 구성될 수 있다. 따라서, 상기 디바이스 자체는 이 인터페이스를 활성화하지 않아 연관된 핀들이 다른 목적들에 사용될 수 있는 정상 동작 모드로 동작될 수 있고, 또한 외부 호스트 컴퓨터로부터 동작될 수 있고 그리고 이 외부 호스트 컴퓨터에 의해 동작될 수 있는 디버거들 또는 에뮬레이터들과 같은 각각의 외부 디바이스들과 데이터를 교환하기 위해 이 인터페이스를 이용하는 디버그 동작 모드로 동작될 수 있다. 상기 디버거들 또는 에뮬레이터들은 또한 프로그래머들로서 동작될 수 있으며, 여기서 프로그램은 동일한 디버그 인터페이스를 통해 타겟 디바이스 내로 전송된다. 따라서, 호스트 컴퓨터, 외부 디버거 또는 에뮬레이터는 저렴한 분석 및 디버깅 시스템을 형성한다.

[0004] 현대의 프로세서들 및 마이크로컨트롤러들은 각각의 디바이스 내부에 확장 세트의 디버그 기능들을 제공한다. 예를 들어, 다수의 브레이크포인트(breakpoint)들이 디바이스 내에 설정될 수 있어 디바이스가 실제로 실시간으로 동작할 수 있게 하는데, 이 실시간 동작은 고속 직렬 인터페이스만을 이용할 때에는 외부 디버거를 사용해서는 불가능할 것이고, 따라서 본드-아웃(bond-out) 칩들 및 값비싼 디버그 회로부를 필요로 할 것이다. 그러나, 이 내부 디버그 회로부들의 기능성이 물론 실리콘 점유 면적의 제한된 양이나 다른 이유들로 인해 어느 정도 제한된다. 예를 들어, 트레이스 백(trace back) 기능들은, 종종 외부의 회로 내 디버거들에 의해 지원되지 않으며, 또한 훨씬 더 복잡한 회로 내의 에뮬레이터들은 디버그 모드에 있을 때, 특히 디바이스가 리셋을 실행할 때, 트레이스 기능을 충분히 지원할 수 없다.

발명의 내용

해결하려는 과제

[0005] 따라서, 프로세서 또는 마이크로컨트롤러 디바이스 내의 향상된 회로 내 디버그 유닛이 필요하다. 예를 들어, 다양한 실시예들에 따른, 비동기 리셋 및 동기 리셋을 통한 명령어 트레이스가 가능하게 된다.

과제의 해결 수단

[0006] 일 실시예에 따르면, 디버그 기능들을 갖는 프로세서 디바이스는, 중앙 처리 유닛, 트레이스 모듈을 포함하는 디버그 회로부, 내부 클록 신호들을 제공하기 위한 시스템 클록 모듈, 및 디버그 모드 동안에 상기 시스템 클록 모듈이 리셋 신호를 수신하는 것을 방지하는 리셋 검출 유닛을 포함할 수 있다.

[0007] 추가 실시예에 따르면, 상기 트레이스 모듈은 리셋 신호가 검출된 후 트레이스 정보를 기록하도록 동작할 수 있

고, 그리고 상기 트레이스 정보를 기록한 후에 상기 시스템 클록 모듈을 리셋하는 신호를 생성한다. 추가 실시예에 따르면, 상기 트레이스 정보는 리셋 소스 정보를 포함할 수 있다. 추가 실시예에 따르면, 상기 트레이스 모듈은 실행된 명령어들에 관한 정보를 포함하는 트레이스 스트림을 생성할 수 있으며, 여기서 상기 트레이스 스트림은 외부 인터페이스를 통해 출력된다. 추가 실시예에 따르면, 상기 트레이스 스트림은 패킷 기반일 수 있다. 추가 실시예에 따르면, 상기 트레이스 패킷은 트리거 소스에 관한 정보를 포함할 수 있다. 추가 실시예에 따르면, 상기 정보는 조건부로 제공될 수 있으며, 여기서 조건은 사용자에 의해 정의될 수 있다. 추가 실시예에 따르면, 상기 리셋 신호는 동기 리셋 신호 또는 비동기 리셋 신호일 수 있다. 추가 실시예에 따르면, 상기 동기 리셋 신호는 워치독 타이머(watchdog timer; WDT), 상기 중앙 처리 유닛에 의해 실행된 RESET 명령어, 스택 오버플로우/언더플로우 리셋에 의해 생성될 수 있다. 추가 실시예에 따르면, 상기 비동기 리셋 신호는 상기 프로세서 디바이스의 외부 핀을 통해 수신될 수 있다.

[0008] 또 하나의 실시예에 따르면, 프로세서 디바이스 내에서 실행된 코드를 디버깅하기 위한 방법은: 중앙 처리 유닛(CPU)에 의해 코드를 실행하는 단계; 리셋 결정시, 트레이스 모듈의 추가 동작을 가능하게 하기 위해 리셋 신호들을 시스템 클록 모듈을 제외한 마이크로컨트롤러의 내부 유닛들에 전송하는 단계; 및 상기 리셋의 수신 후에 트레이스 정보를 기록하는 단계를 포함할 수 있다.

[0009] 상기 방법의 추가 실시예에 따르면, 상기 방법은 상기 트레이스 정보가 기록된 후에 상기 시스템 클록 모듈을 리셋하는 단계를 더 포함할 수 있다. 상기 방법의 추가 실시예에 따르면, 상기 트레이스 정보는 리셋 소스 정보를 포함할 수 있다. 상기 방법의 추가 실시예에 따르면, 상기 트레이스 모듈은 실행된 명령어들에 관한 정보를 포함하는 트레이스 스트림을 생성할 수 있으며, 여기서 상기 트레이스 스트림은 외부 인터페이스를 통해 출력된다. 상기 방법의 추가 실시예에 따르면, 상기 트레이스 스트림은 패킷 기반일 수 있다. 상기 방법의 추가 실시예에 따르면, 상기 트레이스 패킷은 트리거 소스에 관한 정보를 포함할 수 있다. 상기 방법의 추가 실시예에 따르면, 상기 정보는 조건부로 제공될 수 있으며, 여기서 조건은 사용자에 의해 정의될 수 있다. 상기 방법의 추가 실시예에 따르면, 상기 리셋 신호는 동기 리셋 신호 또는 비동기 리셋 신호일 수 있다. 상기 방법의 추가 실시예에 따르면, 상기 동기 리셋 신호는 워치독 타이머(WDT), 상기 중앙 처리 유닛에 의해 실행된 RESET 명령어, 스택 오버플로우/언더플로우 리셋에 의해 생성될 수 있다. 상기 방법의 추가 실시예에 따르면, 상기 비동기 리셋 신호는 상기 프로세서 디바이스의 외부 핀을 통해 수신될 수 있다.

발명의 효과

[0010] 본 발명에 따라, 프로세서 또는 마이크로컨트롤러 디바이스 내의 향상된 회로 내 디버그 유닛이 제공된다.

도면의 간단한 설명

[0011] 도 1은 다양한 실시예들에 따른 집적된 디버그 모듈의 블록도를 도시한다.

도 2는 도 1의 하드웨어 브레이크포인트 블록도를 보다 상세하게 도시한다.

도 3은 도 1의 이벤트 결합기 블록도를 보다 상세하게 도시한다.

도 4는 트레이스 클록을 처리하는 블록도를 도시한다.

도 5는 도 4에 사용된 바와 같은 예시적인 트레이스 Q-발생기를 도시한다.

도 6은 도 4에 사용된 바와 같은 예시적인 ICD 아날로그 리셋 유닛을 보다 상세하게 도시한다.

도 7은 도 1의 트레이스 블록도를 보다 상세하게 도시한다.

도 8은 전형적인 트레이스 신호 타이밍도를 도시한다.

도 9는 단순화된 블록도의 트레이스 서브(sub) 시스템을 도시한다.

도 10은 다양한 실시예들에 따른 내장된 디버그 유닛을 갖는 마이크로컨트롤러 및 외부의 회로 내 디버그(in circuit debug; ICD) 제어기를 이용하는 시스템의 블록도를 도시한다.

도 11은 시스템 클록(clkin) 및 관련 쿼드러처 신호들의 예시적인 타이밍도를 도시한다.

도 12는 다양한 실시예들에 따른 내장된 디버그 유닛을 갖는 마이크로컨트롤러 및 외부의 회로 내 디버그(ICD) 제어기를 이용하는 시스템의 블록도를 도시한다.

발명을 실시하기 위한 구체적인 내용

[0012] 전형적인 마이크로컨트롤러 디바이스는 몇몇 비동기식 및 동기식 시스템 리셋 소스들을 구비할 수 있다. 다양한 실시예들에 따르면, 마이크로컨트롤러 유닛(MCU) 시스템이 이러한 리셋 발생시 동작을 중지하더라도, 상기 디바이스의 디버그 유닛 내의 명령어 트레이스 모듈은 리셋 발생 및 소스를 기록할 것이다. 이를 위해, 다양한 실시예들은 외부의 비동기식 디바이스 리셋 및 내부의 동기식 디바이스 리셋을 통해 명령어 트레이스를 제공한다.

[0013] 다양한 실시예들에 따르면, 트레이스 시스템은 시스템 클록 구성 모듈로의 리셋을 차단하고, 리셋을 (비동기이면) 동기화하고, 그리고 리셋을 기록한다. 따라서, 상기 시스템에는 리셋에 대한 타이밍 및 감도(sensitivity)가 없다.

[0014] 도 1은 일 실시예에 따라 마이크로컨트롤러 내에 집적될 수 있는 회로 내 디버그 모듈의 블록도를 도시한다. 그러나, 위에 설명된 바와 같은 일반적인 개념은 다른 유형들의 온-칩 디버그 회로부에 구현될 수 있다. 상기 블록도는 예를 들어 마이크로컨트롤러에 구현될 수 있고 그리고 다음과 같은 5개의 기본 블록들로 이루어질 수 있는 모듈을 보여준다:

[0015] ● 브레이크포인트 비교 로직(135)

[0016] ● 스톱-워치(Stop-Watch) 사이클 카운터 로직(150)

[0017] ● 제어 및 상태 머신 로직(145)

[0018] ● 트레이스 로직(120)

[0019] ● 이벤트 결합기 로직(125)

[0020] CPU(110)는 내부 버스를 통해 하드웨어 브레이크포인트 유닛(135), 이벤트 검출 유닛(140), 제어 로직 유닛(145) 및 백그라운드 인터페이스(155)와 결합된다. 멀티플렉서(160)는 전용 디버그 클록 및 데이터 핀들(165)을 통해 디버그 유닛의 외부 디버거와의 결합을 가능하게 하는데 사용된다. 제어 로직(145)은 하나 이상의 외부 핀들과 결합될 수 있다. 도 1은 예시적인 TRGIN 핀(185)을 보여준다. 이러한 핀은 다른 기능이 전혀 없는 전용 핀일 수 있다. 그러나, 특히 로우(low)-핀 디바이스들에서, 이러한 핀은 프로그램 제어 하에 서로 다른 주변 디바이스들에 할당될 수 있는 다중-기능 핀일 수 있으며, 따라서 그의 프로그래밍된 할당에 따라 서로 다른 기능들을 수행할 수 있다. 예를 들어, 이러한 핀은 기능적으로 트레이스 트리거에 부가되도록 구성 레지스터에 의해 프로그래밍되어, 직렬 인터페이스 클록 핀, 디지털 I/O 핀, 아날로그 입력 핀 등으로서 동작할 수 있다. 도 1에 도시된 바와 같이, 제어 로직은 또한 핀(185)의 다중-기능 핀과 유사할 수 있는 트리거 출력 핀(190)을 제공한다. 트레이스 모듈(120)은 트레이스 스톨(stall) 입력 핀(115)과 트레이스 클록 및 데이터 출력 핀들(175)과 결합된다. 도 1은 또한 제어 로직(145)을 통해 구성 가능한 펄스 거부 필터들(170 및 180)을 도시한다. 이러한 필터들을 통한 신호 라우팅은 도 1에 도시되지 않았다. 몇몇 실시예들에 따르면, 브레이크포인트 디버깅은 "제로 스키드(zero skid)" 동작으로 불리는 명령어가 실행되기 전에 실행이 정지(halt)되도록 구현된다. 다른 디버거 실시예들에 따르면, 이것이 적용되지 않아서, 코드가 중지하거나 '스키드'하는 곳에서 프로세서가 정지되기 전에 명령어를 실행 가능하게 하는 문제들을 유발할 수 있다. 외부 이벤트들은 (정의상) 명령어 실행 스트림과 비동기적이다. 이를테면, 그들의 동작은 제로 스키드 개념들과 비교될 수 없다.

[0021] 내부 신호 debug_en = 1일 때, 상기 모듈은 활성화되고, 모든 "HALT" 이벤트들을 모니터링하고, 이벤트들을 생성하고, 데이터 캡처를 수행하는 등을 행한다. 상기 내부 신호 debug_en = 0이면, 모든 디버그 로직은 비활성화되고, 상기 모듈은 최소의 전력 모드를 소비하도록 구성된다.

[0022] 디버깅을 덜 침입적(invasive)으로 행하기 위해서는, 실시간으로 디바이스의 외부로 데이터를 스트리밍하는 수단을 구비하는 것이 유용할 수 있다. 트레이스 모듈(120)은 판독되거나 특정 어드레스에 기록되는 데이터를 획득하여 트레이스 포트 외부로 송신하는 방법을 지원한다. 이는 실시간 워치포인트(watchpoint)라고 말할 수 있다. 상기 디바이스의 정상 동작은 워치포인트를 사용할 때 인터럽트되지 않는다.

[0023] 데이터 캡처는 하드웨어 브레이크포인트를 생성하는데 사용되는 것과 동일한 하드웨어를 사용하여 수행될 수 있다. 홀트(halt) 생성과 동시에 데이터 캡처를 위한 브레이크포인트를 사용함으로써, 두 동작들 모두를 발생시킬 수 있을 것이다. 그러나, 데이터 캡처 및 데이터 매칭을 동시에 활성화하는 것은, 특히 하나보다 많은 브레이크포인트 카운트들에 대해서는, 예기치 않은 결과들을 발생시킬 수 있다. 데이터 캡처 및 데이터 비교가 동일한

물리 레지스터를 사용함에 따라, 비교 값은 모든 캡처에서 업데이트될 것이다.

[0024] 도 2는 하드웨어 브레이크포인트 유닛(135)을 보다 상세하게 도시한다. 하드웨어 브레이크포인트들은 프로그램 및 데이터 메모리 중 어느 하나의 어드레스들의 매칭시 정지시키도록(break on) 구성될 수 있다. 이를 위해, 각각의 비교기들(220 및 230) 및 디코딩 유닛들(240)이 도 2에 도시된 바와 같이 제공된다. 브레이크포인트가 데이터 메모리 상에서 동작하도록 선택될 때, 상기 브레이크포인트는 부가적으로 데이터 값 및 마스크로 자격을 얻을 수 있어, 특정 값들만이 브레이크포인트 이벤트를 생성할 수 있게 한다. 또한, 데이터 브레이크포인트들은 단지 판독이나 기록 사이클들을 정지시키도록 임의로 설정될 수 있다. 모든 경우들에서, 브레이크포인트들은, 브레이크포인트 이벤트가 생성되기 전에 특정 이벤트가 N번 발생해야 하도록, 카운터(210)를 구비한다. 이 카운터(210)는 몇몇 실시예들에 따라, 예를 들어 1부터 256번까지의 임의의 값이 설정될 수 있다.

[0025] 도 2의 블록도는 단일 브레이크포인트에 대해 도시되었다. 그러나, 구현된 브레이크포인트들의 수는 다양한 실시예들에 따라 가변적이고, 많은 브레이크포인트들이 존재할 수 있다. 도 2는 다양한 파라미터들이 브레이크포인트를 위한 트리거 요구사항들을 정의하도록 프로그래밍될 수 있게 하는 예시적인 실시예를 보여준다. 다른 실시예들에 따르면, 보다 많거나 보다 적은 이러한 파라미터들이 사용될 수 있다. 예를 들어, 브레이크포인트를 발생시키는데 필요한 브레이크포인트 발생 횟수는 카운터(210)의 BPxCNT 파라미터에 의해 설정될 수 있다. 각각의 브레이크포인트 모듈은 동일한 레지스터들을 가질 수 있다.

[0026] 브레이크포인트들은 이벤트 채널 정의들 사이에 나열되며, 사이클 카운터(150)를 개시하거나 중지하는데, 이벤트 결합기 스테이지(125)를 설정하거나 재설정(리셋)하는데, 트레이스 유닛(120)을 개시하거나 중지하는데, 또는 스택 스냅샷(stack snapshot)을 획득하는데 사용될 수 있다.

[0027] 일 실시예에 따르면, 브레이크포인트가 활성화되기 위해서는 제어 레지스터 ICDBFxCON의 비트 BPEN이 설정되어야 한다. 이 비트가 소거되면, 이 특정 브레이크포인트에 대한 모든 회로부는 비활성화되고, 어떠한 브레이크포인트 이벤트도 생성되지 않을 것이다. 브레이크포인트는 자격 조건들의 모든 N번째 발생시에 소정 동작을 단지 트리거하도록 구성될 수 있다. 예를 들어, 매 세 번째 발생마다 트리거하도록 브레이크포인트를 설정하기 위해, 카운터(210)는 BPxCNT = 2로 설정된다. 각각의 제어 레지스터들은 그 값을 재로딩(reloading)하고/하거나 현재 상태를 모니터링하기 위해 카운터(210)와 결합하여 사용될 수 있다.

[0028] 또한, 브레이크포인트들은 예를 들어 연관된 구성 레지스터의 각각의 비트들을 설정함으로써 실행 컨텍스트(메인 라인 코드, 인터럽트 핸들러, 또는 둘 다)를 기반으로 자격이 부여될 수 있다. 브레이크포인트는 이후 프로그램이 선택된 컨텍스트로부터 실행중일 때에만 발생할 수 있다.

[0029] 구성 레지스터의 각각의 비트들을 설정함으로써, 프로그램 카운터(PC 실행 어드레스)를 모니터링할 수 있게 하는 다른 또 하나의 브레이크포인트 파라미터가 사용될 수 있다. 프로그램 메모리 브레이크들은 제로 스키드이며, 동작이 실행되기 전에 발생한다. PC는 트리거 명령어의 어드레스를 표시할 것이다.

[0030] 각각의 제어 비트가 소거될 때, 예를 들어 BPAREN = '0'이면, 브레이크는 PC가 소정 어드레스와 동일할 때 트리거된다. BPAREN = '1'이면, 브레이크는 PC가 어드레스들의 소정 포함 범위에 들어갈 때 트리거된다.

[0031] 몇몇 실시예들에 따르면, 실행된 명령어들만이 브레이크포인트를 생성할 수 있다. PC가 실행되지 않는 어드레스에 있다면, 브레이크포인트는 트리거하지 않는다. 이는 다음을 포함한다:

[0032] ● 흐름 변경 명령어들(CALL, RETURN 등),

[0033] ● 스킵 명령어들(per BTFSS, BTFSC), 또는

[0034] ● PCL, FSR 또는 다른 두 사이클 명령어들 이후의 다음 페치.

[0035] 또 하나의 제어 비트 필드가 제어 레지스터에서 01, 10, 또는 11로 설정될 때, 브레이크포인트는 데이터 액세스들, 즉 어드레스 및 값 둘 다를 모니터링한다. 연관된 비트들의 세 가지 상태들은 판독 또는 기록 사이클들이 브레이크포인트를 결정하는데 사용되는지의 여부를 선택한다.

[0036] 데이터가 (적용되는 것처럼) 판독되거나 기록된 후, 데이터 브레이크포인트들은 명령어 실행의 끝에서 필연적으로 브레이크를 일으킨다. 모든 경우들에서, 명령어는 완료될 때까지 실행된다. 따라서, "브레이크"는 실제로 다음 명령어 실행시 발생하고, 그리고 PC는 트리거 명령어 다음에 오는 명령어를 표시할 것이다. 또한, 브레이크는 메모리 어드레스와 데이터 값의 자격요건(qualifier)들이 둘 다 충족되었을 때 트리거될 수 있다.

[0037] 사이클 카운터(150)는 사용자 코드가 프로파일될 수 있도록, 스톱워치 기능을 제공하는데 사용되는 카운터이다. 사이클 카운터는 각각의 제어 레지스터들에 의해 제어된다. 카운터(150)는 네 개의 8-비트 카운터/레지스터들로 이루어질 수 있다. 카운터(150)는 CPU의 매 Q-사이클의 끝에서 증분될 수 있으며; 멀티-사이클 명령어들(예컨대, GOTO)은 여러 번 카운트한다.

[0038] 다수의 기능들이 특정 이벤트들에 의해 제어될 수 있도록 하기 위해, 가능한 모드 소스들은 하나의 이벤트 버스로 결합될 수 있다. 이는 사이클 카운터(150), 트레이스 유닛(120), 및 이벤트 결합기 유닛(125)이 이들의 동작들을 선택하기 위해 동일한 설정값들(settings)을 사용할 수 있게 한다.

[0039] 도 3에 도시된 바와 같은 이벤트 결합기(300)는 다수의 이벤트 입력들(320)을 모니터링하고, 그리고 그 입력들의 결합들 및 시퀀스들에 기초하여 정지(halt) 또는 트리거 출력(190)을 생성할 수 있다. 이벤트 결합기(300)는 각각의 제어 비트가 설정될 때 활성화된다. 비활성화된 결합기들(300)은 출력 이벤트들을 생성하지 않는다. 이벤트 결합기들(300)은 이벤트 채널 정의들 사이에 나열되고, 그리고 사이클 카운터(150)를 개시하거나 중지하는데, 이벤트 결합기 스테이지(310)를 설정하거나 재설정하는데, 트레이스 유닛(120)을 개시하거나 중지하는데, 또는 스택 스냅샷을 획득하는데 사용될 수 있다. 이벤트 결합기 스테이지들(310)은 그 스테이지에 대한 각각의 제어 비트들이 연관된 제어 레지스터들 내에 설정될 때 독립적으로 활성화된다. 스테이지의 현재 출력은 연관된 상태 비트에 반영될 것이다. 스테이지들(310)은 도 3에 도시된 바와 같이 묵시적인 순서를 가지며, 다음과 같은 다수의 방식들로 결합될 수 있다:

[0040] ● 스테이지는 하나의 이벤트에 의해 개별적으로 활성화될 수 있다,

[0041] ● 스테이지는 다음의 낮은 스테이지가 활성인 동안 이벤트에 의해 활성화될 수 있다,

[0042] ● 스테이지는 하나의 이벤트에 의해 개별적으로 비활성화될 수 있다,

[0043] ● 스테이지는 하나의 이벤트에 의해 또는 다음의 낮은 스테이지가 비활성화될 때 비활성화될 수 있다.

[0044] 각각의 제어 비트를 설정함으로써, 결합된 이벤트(들)의 N+1번째 발생만이 출력 이벤트에 신호를 보낼 것이다. N은 0부터 255까지 설정될 수 있다. 결합된 트리거 조건들이 충족되면, 레지스터는 1씩 감분된다. 결합된 트리거 조건들이 충족되면, 이벤트 결합기 이벤트가 생성되고 그리고 카운터는 미리 설정된 값으로 재로딩(reloading)된다. 또한, 임의의 시간에 새로운 카운트 값이 각각의 제어 레지스터에 기록되고, 상기 카운터의 값은 재로딩된다. 예를 들어, 3번째 발생시에 트리거하도록 브레이크포인트를 설정하기 위해, 각각의 카운터 값은 2로 설정되어야 한다.

[0045] 게다가 몇몇 실시예들에 따르면, 핀 TRGIN(185)에 공급된 외부 신호는, 사용자 입력이 트레이스 스트림 내에 삽입될 트레이스 패킷들을 생성할 수 있게 하고, 정지들을 생성할 수 있게 하고 또한 임의로 트리거 TRGOUT 신호들을 생성할 수 있게 한다. "극성" = 0일 때(도 6), 트리거 입력은 액티브 하이(high)이고 상승 에지(edge)들은 이벤트들을 일으킨다. "극성" = 1일 때, 트리거 입력은 액티브 로우(low)이고 하강 에지들이 이벤트들을 일으킨다. 또 하나의 제어 비트는, 예를 들어 입력이 인식되기 위해 최소 시간 동안 액티브 상태에 있어야 함을 정의하기 위해, 필터를 제어하는데 사용될 수 있다. 이후 보다 짧은 펄스들이 무시된다.

[0046] TRGIN 이벤트는 이벤트 채널 정의들 사이에 리스트될 수 있고, 그리고 사이클 카운터(150)를 개시하거나 중지하는데, 이벤트 결합기 스테이지(310)를 설정하거나 재설정하는데, 트레이스 유닛(120)을 개시하거나 중지하는데, 또는 스택 스냅샷을 획득하는데 사용될 수 있다. 트리거 입력의 변화들은 트레이스가 활성화되면 트레이스 패킷을 생성할 것이다.

[0047] 브레이크포인트 같은 이벤트가 인에이블된 트리거에 의해 발생할 때, TRGOUT 핀(195)에 펄스가 생성된다. 기본 트리거 출력 신호 동작은 각각의 제어 비트들을 설정함으로써 구성된다. 이 제어 비트들은 예를 들어, 트리거 출력이 대략 트리거 이벤트의 지속기간 동안 어서트(assert)되는 것을 제어할 수 있다. 인접하거나 혹은 중첩하는 이벤트들은 신호를 어서트된 상태로 유지할 수 있다. 상기 제어 비트들은 또한 출력이 최소 시간 주기로 스트레칭(stretching)되는지의 여부를 제어할 수 있다. TRGOUT 원샷이 트리거되면, 시간 주기 내에 발생하는 더 많은 이벤트들이 무시될 것이다. 상기 원샷이 타임 아웃되고 TRGOUT가 제로(zero)로 귀환한 후, 그것은 또 하나의 이벤트에 의해 다시 트리거될 수 있다. 상기 원샷은 에지 트리거되고, 그리고 이벤트 신호가 지속되더라도 소정 시간 주기 이후 소거될 것이다.

[0048] 소프트웨어는 각각의 제어 비트를 설정함으로써 트리거 아웃을 일으킬 수 있다. 디바이스가 깨어 있으면, 비트

는 1 사이클 이후 하드웨어에 의해 소거된다. TRGOUT는 또한 각각의 제어 비트를 기록함으로써 소거될 수 있거나, 또는 디바이스가 깨어나면 자동으로 소거될 것이다.

[0049] 외부의 비동기식 리셋들은 전형적으로 전체 프로세서 디바이스에 영향을 미친다. 이러한 이벤트가 발생했음을 정확하게 기록하기 위해, 명령어 트레이스 모듈은 리셋의 타이밍과는 무관하게 리셋에 의한 영향을 받지 않도록 설계된다. 따라서, 다양한 실시예들은 온(on)-칩 디버그 기능들을 향상시키고, 이전에 종래의 마이크로컨트롤러 디바이스들에서는 이용할 수 없었던 특징들을 제공한다. 따라서, 단지 고가의 전용 회로 내 디버거들과 일반적으로 구별되는 첨단 디버깅 기능들은 "정상" 마이크로컨트롤러 또는 마이크로프로세서 디바이스들에 구현될 수 있다.

[0050] 도 4는 다양한 실시예들에 따른 고레벨 개념을 도시한다. 시스템 클록 구성 모듈(410)은 시스템 클록(icd_sys_clk)을 생성한다. 시스템 클록은 trc_q_clks 신호들, 즉 trc_q1_clk, trc_q2_clk, trc_q3_clk 및 trc_q4_clk의 4개의 위상들을 생성하는 트레이스 Q-발생기(420)에 의해 사용된다. 도 11은 명령어 실행에 관하여 시스템 클록(clkin) 및 관련 쿼드러처 신호들의 예시적인 타이밍도를 도시한다. 이 실시예에서는, clkin으로부터 유도된 4개의 클록들(q1, q2, q3, q4)이 하나의 명령어를 실행하는데 사용된다. 트레이스 클록들은 내부 클록들(q1, q2, q3, q4)로부터 유도될 수 있거나, 또는 이 클록들과 동일할 수 있다. 그러나, 다른 실시예들에 따르면, 예를 들어 단일 사이클 내의 명령어들을 실행시킬 수 있는 다른 중앙 처리 아키텍처가 구현될 수 있다.

[0051] 도 5는 트레이스 Q-발생기(420)의 개략도의 세부사항들을 도시한다. 트레이스 Q-발생기는 예를 들어 기존 방식으로 4개의 플립-플롭들(510, 520, 530 및 540)에 의해 형성된 4-비트 순환 시프트 레지스터일 수 있다. 리셋 소스들의 몇 가지 예들은 다음과 같다: 워치독 타이머(watchdog timer; WDT) 리셋, 중앙 처리 유닛에 의해 실행된 RESET 명령어, 스택 오버플로우/언더플로우 리셋 및 마스터 클리어(master clear; MCLR) 리셋. 그러나, 다른 리셋 소스들 및 신호가 적용될 수 있다.

[0052] 도 4를 다시 보면, 일반 리셋이 발생할 때, 리셋 아날로그 유닛(430)은 일반 리셋이 시스템 클록 구성 모듈(410)을 제외한 전체 칩에 어서트될 수 있게 한다. 이는 시스템 클록이 계속 실행될 수 있게 하고, 트레이스 Q-발생기(420) 및 따라서 또한 트레이스 데이터 발생기(120)는 계속 동작할 수 있다. 마이크로컨트롤러 중앙 처리 유닛(110)이 리셋으로 유지되는 동안, 트레이스 데이터 발생기는 리셋 관련 트레이스 정보를 기록하고, 이후 트레이스 Q-발생기에게 상기 기록이 수행되었음을 알린다.

[0053] 일반 리셋 조건은 또한, 예를 들어 플립-플롭들(450 및 460)에 의해 각각 trc_q2_clk 신호 및 trc_q4_clk 신호를 이용하여 동기화되고, 그리고 트레이스 Q-발생기(420)가 중지할 때, 리셋 아날로그 유닛(430)은 예를 들어 플립-플롭(480)에 의해 통지되어 시스템 클록 구성 모듈(430)이 리셋될 수 있게 한다.

[0054] 리셋 아날로그 유닛(430)의 예시적인 실시예는 도 6에 도시되어 있다. 리셋 아날로그 유닛(430)은 시스템 클록 구성 모듈(410)이 리셋될 수 있게 하는데, 다만 이렇게 하도록 통지를 받은 후에 가능하다. 상기 시스템 클록 구성 모듈(430)이 리셋됨과 동시에, 원-샷 플러스 trc_reset_osl_pulse가 생성되어 동기화기 레지스터 요소들(450, 460, 480)을 리셋하고, 이에 따라 전체 회로는 또 하나의 리셋 발생 검출을 위해 재장비될(re-armed) 수 있다. 도 6에 도시된 바와 같이, 로직은 여러 제어 신호들을 생성하도록 제공될 수 있다. 디바이스의 일반 설계에 따라, 다른 로직 회로들이 사용되고 적응될 수 있다.

[0055] 따라서, 이 다양한 실시예들은 레벨 민감성 리셋 및 에지 민감성 리셋을 둘 다 처리할 수 있는 기능을 가지며, 그리고 리셋이 발생하자마자 (시스템 클록 구성 모듈을 제외한) 전체 마이크로컨트롤러 시스템이 리셋될 수 있게 하고, 따라서 명령어 트레이스가 진행되는 경우에도 네이티브 동작에 더 가깝게 한다.

[0056] 다양한 실시예들은 최소 펄스 폭 지속기간이 시스템 클록 구성 모듈을 리셋하도록 보증됨을 보장하는데, 상기 모듈이 칩 리셋이 발생하자마자 리셋되는 것으로부터 방지되어 있다고 하더라도 그러하다.

[0057] 다음 절에서는, 디바이스와 외부 디버그 툴 사이의 트레이스 데이터 인터페이스를 상세히 설명한다. 도 7에 예를 들어 도시된 바와 같은 트레이스 서브시스템은, 디버그 툴에 의해 캡처되고 분석될 수 있는 명령어 실행 스트림의 실시간 기록을 제공한다. 트레이스 동작은 디버그 툴이 소스 코드 및 프로그램 메모리 콘텐츠에 액세스하였고 그리고 CPU 동작의 몇몇 양상들을 추론할 수 있다고 가정한다.

[0058] 명령어가 PCL에 (직접 또는 INDx를 통해) 기록할 때, 새로운 PC는 프로그램 메모리에 어디든지 있을 수 있다. 이 경우, 명령어 패킷은 소위 전체 프로그램 카운터(full program counter; FPC) 패킷으로 대체된다. 도 7에 도시된 바와 같은 FIFO는 코어 데이터 레이트를 (TRSTALL에 의해 지배되는) 디버그 툴 데이터 레이트와 매칭시키는데 사용될 수 있다. FIFO는 CPU 인코더로부터 채워지고, 도 7에 도시된 바와 같이, 데이터 인코더에 의해 비

워진다. FIFO는 최대 4096 명령어 패킷들을 보유하지만, 효과적인 운영 사이즈는 각각의 제어 비트들에 의해 선택될 수 있다. WATCH 이벤트 패킷들은 또한 FIFO 내에 배치되어서, 실제 인플라이트(in-flight) 명령어 패킷들의 수는 보통 적을 것이다. 토글링할 때, TRCLK 출력은 각각의 클록 에지에 따라 하나의 데이터 워드를 출력한다. 출력 클록 레이트는 항상 CPU 명령어 레이트에 링크되고, 그리고 소프트웨어가 SYSCLK를 변경시키거나 리셋이 SYSCLK를 변경시키면 변화할 것이다. 각각의 구성에 따르면, 출력 레이트는 항상 명령어 주기당 두 개의 트레이스 워드들일 수 있다. 상기 레이트는 명령어당 1 및 1/2로 감소할 수 있지만, 이로써 FIFO 오버플로우를 야기시킬 것이다. TRCLK 출력은, TRCPS 비트들의 설정에 따라, 데이터 변화들과 같은 위상이거나 다른 위상이도록 선택될 수 있다. FIFO 내 인코딩은 라인 인코딩과 상이할 수 있어서, 각각의 FIFO 위치는 TRDAT 인터페이스 내에 무려 3개나 되는 데이터 워드들을 나타낼 수 있다.

[0059] 트레이스 스톨(stall) 기능이 구현될 수 있으며, 여기서 구성 비트는 TRSTALL 입력이 효과를 나타내는지의 여부를 결정할 수 있다. 예를 들어, 제어 비트 TRXSE = 1이고 신호 TRSTALL = 1일 때, 트레이스 FIFO는 페이로드 경계(payload boundary)에서 비우기를 중지하고 클록을 중단할 것이다. TRSTALL(tr_stall_pin)이 '0'으로 귀환할 때, 클록킹은 재개될 것이고 FIFO는 데이터를 다시 비우기 시작할 것이다. 트레이스가 인에이블되고 FIFO가 비어있을 때, IDLE 또는 SLEEP 패킷들은 디바이스의 슬립(sleep) 상태에 따라 전송된다. 몇몇 실시예들에 따라 FIFO는 또한 강제로 비워질 수 있고, 트레이스는 비활성화될 수 있다. FIFO가 가득 차게 되면, 시스템 응답은 각각의 제어 설정에 의존할 수 있다. 여하튼, (스톨된(stalled) 또는 데이터를 포스팅(posting)하지 않는) 오버플로우 상태는, FIFO가 각각의 제어 레지스터에 의해 선택된 대로 25% 또는 75%로 가득 찰 때까지 지속될 것이다. 몇몇 실시예들에 따라, 트레이스 구현에 관한 다른 기능들이 부가될 수 있고, 그리고 개시된 바와 같은 몇몇 기능들은 구현되지 않을 수 있다. 전형적인 트레이스 신호 타이밍도가 도 8에 도시되어 있다.

[0060] 트레이스 페이로드 패킷들은, CPU 코어가 실행하는 명령어들 및 선택된 데이터 이벤트들을 인코딩하고, 또한 트레이스 스트림 동기화를 제공한다. 일 실시예에 따르면, 대부분의 패킷들은 1개 또는 2개의 7-비트 워드들로 이루어질 수 있고, 또는 FPC 패킷에 대해서는 3개의 워드들로 이루어질 수 있다. 일반적으로 말해서, 패킷들은 "워드 1" 및 임의의 "워드 2"로 이루어진다. 그러나, 또 다른 실시예들에 따라 다른 포맷들이 사용될 수 있다. 특정 실시예에 따르면, "워드 1"의 값은 패킷을 확인하고, "워드 2"가 존재하는지의 여부를 암시한다. (EX, EXD, 및 RESET과 같은) 명령어 실행과 동기하는 패킷들 및 비동기 "이벤트" 패킷들은 아래에 보다 상세히 개시되는 바와 같이 WATCH, RESET, 및 TRIGGER를 포함한다.

[0061] 동기화 패킷들은 실행 순서대로 방출된다. 이벤트 패킷들은 이벤트 시간 부근의 스트림에 나타나지만, 다수의 이벤트들이 동시에 일어나면, 몇몇 보고들은 지연될 것이다. 몇몇 경우들에서는, FPC가 동기화 보고이지만, 다른 때에는 FPC가 이벤트이다. 전송 층(transport layer; TR) 패킷들(RESYNC, IDLE 및 END)은 필요에 따라 삽입되어 인터페이스 데이터 스트림의 여러 상태들을 관리하고 확인한다. TR 패킷들(RESYNC 및 IDLE)은 명령어 트레이스를 분석할 때 버려질 수 있다.

[0062] RESYNC 패킷들은 각각의 제어 비트들에 의해 특정된 대로 주기적으로 삽입되고, 그래서 수신기는 RESYNC 패킷들이 올바르게 동기화되어 있는지 확인할 수 있다. 때때로 소정 시간 간격에 대략 대응하고, 그리고 어떠한 다른 FPC도 상기 간격 내에서 전송되지만 않았다면, FPC는 상기 스트림에 부가될 것이다. 이는 수신기가 올바르게 명령어 스트림을 추적하고 있다는 체크를 제공한다. 삽입된 FPC들은 항상, 다음에 오는 명령어의 어드레스를 표시한다. 패킷 내의 워드들의 수는 패킷 워드 1의 값에 의해 결정된다. RESYNC가 "워드 2"에 대해 틀린 값을 갖는 워드 1로서 수신되면, 스트림은 아웃 오브 싱크(out of sync)이고 에러는 플래그되어야 한다.

[0063] 표 1은 RESYNC가 다음에 오는, 제 2 워드가 0x7D인 2-워드 패킷을 포함하는 최악의 경우의 상황을 나타낸다. 수신기가 적당히 동기화되면, 수신된 워드 #3는 워드 1일 것이고, 워드 #4는 워드 2일 것이고, 패킷 #2로서 도시되는 완전한 RESYNC 쌍을 형성할 것이다. 패킷 #3은 워드 #5로 시작되고, 올바르게 해석될 것이다.

표 1

수신된 워드		패킷	
#	값	유형/데이터	#
1	0x71	EXD 0xFD	1
2	0x7D		
3	0x7D	RESYNC	2
4	0x7D		
5	0x74	EX	3

[0064]

[0065] 표 2는, 제 1 워드를 복제하고 수신기를 아웃 오브 싱크로 만드는 클록-바운스(clock-bounce)를 갖는 동일한 데이터를 보여준다. 워드 #1 및 워드 #2는 데이터=0x71를 갖는 EXD 패킷으로서 수신되고(하지만 이는 틀린 해석이다), 워드 #3 및 워드 #4는 RESYNC 쌍처럼 나타난다. 워드 #5는 새로운 RESYNC 패킷의 워드 1로서 취해지지만, 워드 #6은 0x7D가 아니고, 동기화되지 않은 상태로 나타난다. 워드 #6은 새로운 패킷을 개시한다.

표 2

수신된 워드		패킷	
#	값	유형/데이터	#
1	0x71	EXD 0xF1	1
2	0x71(1)		
3	0x7D	RESYNC	2
4	0x7D		
5	0x7D	Error	Note 2
6	0x74		3

Note 1: Receiving a duplicate word is typical of an impedance mismatch in the clock cable.

2: It is sufficient to say that the first non-0x7D that follows *any* 0x7D packet (word 1 = 0x7D) is always a word 1 (or FPC word #3). The receiver must immediately re-interpret word #6 as the first word in a new packet.

[0066]

[0067] 트레이싱이 개시될 때, 또는 디버그 실행을 위해 일시 중단된 후 트레이싱이 재개할 때, 전송된 제 1 패킷은 항상 FPC일 것이다. 전체 프로그램 카운터(FPC) 패킷은 스트림에 나타나는 다음 명령어의 절대 주소를 보고한다.

[0068] – 트레이스의 개시,

[0069] – 오버플로우 후의 재개, 및

[0070] – 디버그 후의 재개

[0071] 의 이 상황들에서의 FPC 보고는 단순히 다음 명령어의 주소를 표시한다.

[0072] FPC의 다른 모든 이용들은 명령어가 실행되었음을 표시하고, 몇몇 경우들에서는 그 명령어에 대해 보고되었어야 할 패킷을 대신한다. FPC가 분기(branch) 또는 프로그램 카운터 변경 명령어들의 실행을 나타낼 때, 보고된 값은 분기 타겟 주소이다. FPC 다음에 오는 명령어 패킷은 FPC가 가리키는 명령어의 실행을 나타낸다.

[0073] GOTO 및 CALL 명령어들은, 프로그램 카운터 PC[10:0]의 최하위 비트들이 (어셈블리 코드의) 디버그 환경에 알려져 있고, 새로운 PC의 상위 4 비트들만이 상부 일부분 프로그램 카운터(upper partial program counter; UPC) 패킷에 보고된다고 가정한다. 그러나, 다른 실시예들은 보다 많거나 보다 적은 정보를 보고할 수 있다. 보고된 값은 0x0F & (PCLATH >> 3)일 수 있고, 여기서 PCLATH는 프로그램 카운터의 상위 비트들을 래칭(latching)하는 구현 특정 레지스터를 나타낸다. 관련 브랜치(BRA)들의 목적지가 소스 코드에 알려져 있기 때문에, 명령어는 단순히 EX로서 보고된다. 매우 다양한 트레이스 페이로드들이 구현될 수 있다. 표 3는 서로 다른 페이로드 신호들의 예를 보여준다.

표 3

Mnemonic	Description	Group	Trace Encoding		Number of 7 bit words Sent	Sent when TRIEN = 0
			Word 1 (Word 3)	Word 2		
WATCH	Data trace watch point	Trace data	0 cccc d	ddd dddd	2	Yes
FPC	Full new PC, P = PC[14:0] Implies execution of the current instruction The 2 packets are sent contiguously.		100 pppp (PC[14:11])	ppp pppp (PC[10:4])	3	1 FPC within RESYNC interval
			101 pppp (PC[3:0])			
UPC	Upper Partial PC, P = PC[14:11] Implies execution of the current instruction		110 pppp (PC[14:11])		1	No
EXD	Execute instruction, D = data stored		111 000d	ddd dddd	2	No
EX	Execute instruction		111 0100		1	No
STALL	No instruction is executed (forced to NOP) PC is unchanged (Section 3.19.3.5)		111 0101		1	No
SKIP	No instruction is executed (forced to NOP) PC is incremented		111 0110		1	No
OVERFLOW	FIFO has overflowed; data was lost		111 0111	111 0111	2(3)	Yes
INT	Interrupt Vector Vectoring to interrupt vector N		111 1000	nnn nnnn Table 3-8	2	No
RESET	CPU is being reset PC is now equal to RSTVEC		111 1010	v00 nnnn Figure 3-4	2	Yes
ERROR	An internal error is noted; refer to hardware documentation for details.		111 1010	x1x eeee Figure 3-4	2	Yes
TRIGGER	TRGTR = 1 and Trigger input change		111 1011		1	Yes
SLEEP	TREN = 2'b1X, FIFO is empty, Sleeping	Transport	111 1100		1	Yes
RESYNC	Periodic resync. FPC will be sent with the same interval.		111 1101	111 1101	2(3)	Yes
IDLE	TREN = 2'b1X, FIFO is empty (not Sleeping, not TRSTALL)		111 1110		1	Yes
END	TREN = 2'b00		111 1111	111 1111	2 or 3(2,3)	Yes
Reserved			111 001x 111 1001		1	

Note 1: All fields are sent MSB first
2: The END packet will be sent at least twice, and possibly a third time so that TRCLK stops in the low state.
3: The receiver should handle OVERFLOW, RESYNC and END as 1-word packets; see the discussion in Appendix A.2.3.

Legend: c = Channel for watchpoint p = Program counter
d = Write Data

[0074]

[0075] 표 4는 실제 트레이스의 예를 보여준다.

표 4

사이클		명령어	패킷
	PC(0x)	연산코드	유형/데이터
1	123	MOVLW HIGH(2300)	EX
2	124	MOVWF PCLATH	EXD 8'h23
3	125	MOVLW #3	EX
4	125	CALL 200	
5			UPC-4 (1)
6	2200	BTFSS W,7	EX
7	2201	BRA $+4	
8			FPC
9	2205	NOP (Note 2)	EX
10	2206	BRW	
11			FPC 220A (1)
11	220A	RETLW #77	
12			FPC
13	126	NOP	EX
14	127	CALLW	
15			FPC 2377 (1)
16	2377	NOP	EX
17	2378	GOTO 500	
18			UPC-4 (1)
19	2500	NOP	EX
20	2501	RETURN	
21			FPC
22	128	NOP	EX

Note 1: If TRFPCB = 1, this instruction reports FPC.
2: The PC value for cycle 8 is not 15'h2205 because PCLATH = 8'h23 (from cycle 2); the UI should flag this error.

[0076]

[0077] 트리거 입력의 변화들은 각각의 제어 비트가 설정되어 있는 경우 트레이스 패킷을 생성할 것이다. 극성 비트 = 0이면, 이벤트는 상승 에지에서 트리거할 것이다. 극성 비트 = 1이면, 이벤트는 하강 에지에서 트리거할 것이다. 명령어가 프로그램 카운터 PCL에 직접적으로 또는 간접적으로 기록할 때, 새로운 PC는 프로그램 메모리에 어디든지 있을 수 있다. 이 경우, 명령어 패킷은 FPC 패킷으로 대체된다.

[0078] FIFO는 코어 데이터 레이트를 (TRSTALL에 의해 지배되는) 디버그 툴 데이터 레이트와 매칭시키는데 사용된다. FIFO는 CPU 인코더로부터 채워지고, 도 6에 도시된 바와 같이, 데이터 인코더에 의해 비워진다. FIFO는 최대 4096 명령어 패킷들을 보유하지만, 효과적인 운영 사이즈는 제어 레지스터 내의 각각의 비트들에 의해 선택된다. WATCH 이벤트 패킷들은 또한 FIFO 내에 배치되어서, 실제 인플라이트(in-flight) 명령어 패킷들의 수는 보통 적을 것이다.

[0079] 토글링할 때, TRCLK 출력은 각각의 클록 에지에 따라 하나의 데이터 워드를 출력한다. 출력 클록 레이트는 항상 CPU 명령어 레이트에 링크되고, 그리고 소프트웨어가 SYSCLK를 변경시키거나 리셋이 SYSCLK를 변경시키면 변화할 것이다. 연관된 제어 레지스터의 각각의 비트 필드가 설정될 때, 출력 레이트는 항상 명령어 주기당 두 개의 트레이스 워드들일 수 있다. 상기 레이트는 명령어당 1 및 1/2로 감소할 수 있지만, 이로써 몇몇 실시예들에 따르면 FIFO 오버플로우를 야기시킬 것이다. 트레이스 클록(TRCLK) 출력은, 각각의 제어 비트들의 설정에 따라,

데이터 변화들과 같은 위상이거나 다른 위상이도록 선택될 수 있다. FIFO 내 인코딩은 라인 인코딩과 상이해서, 각각의 FIFO 위치는 TRDAT 인터페이스 내에 무려 3개나 되는 데이터 워드들을 나타낼 수 있다.

[0080] 도 9는 트레이스 서브(sub) 시스템(700)의 단순화된 블록도를 도시한다. 서브시스템(700)은 코어 및 WATCH 이벤트 신호들로부터 명령어 코드들을 수신하고, TRDAT 신호들 상의 전달을 위해 이 데이터를 포맷한다. 시퀀스 제어기(710)는 명령어 및 WATCH 데이터를 FIFO에 로딩할 책임이 있다. 각각의 데이터 패킷은 단일 16-비트 워드로서 인코딩된다. q34동안, WATCH 신호들은 샘플링되고, 가장 높은 우선 순위의 신호가 인코딩되고, 로딩되고 그리고 리셋된다. 2 이상의 신호가 어서트되면, 가장 높은 우선순위의 신호만이 로딩되고, 나머지 다른 신호들은 이후의 q34 기회들을 기다려야 한다. 보고 우선순위가 브레이크포인트 넘버를 기반으로 하기 때문에, 이벤트들은 순서와 다르게 보고될 수 있다. q12 동안에는 이전 명령어로부터의 데이터가 인코딩되고 로딩된다(q3에서는 연산 코드가 인코딩되고, 버스 데이터는 안정적인 q3-q3이고, 그리고 모두는 상승 q1에서 유효하다). 일반적으로 말해서, 이것은 모든 명령어 주기에서 발생한다. 분기 및 호출 명령어들에 대해서는, 인터럽트 사이클들뿐만 아니라, 제 1 사이클 동안에는 아무것도 로딩되지 않으며, 패킷은 제 2 사이클 동안 인코딩되어(소위 "강제된 NOP"), UPC 및 FPC가 정확한 PC 값으로 방출될 수 있게 한다. 결과적으로, 두 WATCH 패킷들은 분기 동안 로딩될 수 있다. SKIP 및 STALL은 현재의 코어 동작을 기반으로 인코딩된다.

[0081] 시퀀스 제어기(710)는 명령어 사이클마다 두 번 로딩할 수 있고 스트림 관리자(730)는 명령어 사이클마다 두 번 언로딩(unloading)할 수 있어, 명령어 사이클마다 최대 4개의 메모리 사이클들을 필요로 할 수 있다. FIFO 제어기(720)는 시퀀스 제어기(710)에 의해 제공된 데이터를 관리한다. 데이터는 스트림 관리자(730)에 의해 요청될 때 동일한 순서로 전달된다. 스트림 관리자(730)는 16-비트 FIFO 워드들을 TRDAT 신호들 상에서 송신되는 데이터 워드들 내에 재포맷한다. 일부 패킷들(예컨대, EX)은 각각의 FIFO 워드에 대해 단일 TRDAT 워드를 생성하지만, 다른 패킷들(예컨대, FPC)은 그보다 많이 생성한다. 필요에 따라, 전송 관리 패킷들(RESYNC, IDLE 및 END)이 스트림 내에 삽입되고, 그리고 판독 동작은 TRSTALL 입력에 따라 일시 정지된다. 각각의 명령어 사이클 주기 동안에는, 2개의 TRDAT 워드들이 전송된다. TRCLK 신호는 시스템 리셋 동안 (사이클의 스트레치(stretch) 부분을) 일시 정지할 것이다(데이터는 손실되지 않을 것이다).

[0082] 도 10은 트레이스 수신기로서 동작하는 디버그 툴(820)과 결합된 다양한 실시예들에 따른 마이크로컨트롤러(810)를 갖는 시스템(800)을 도시한다. 디버그 툴은 예를 들어, 출원인에 의해 제조된 회로 에뮬레이터의 리얼(Real)-ICE일 수 있다. 수신기(820)는 트레이스 동기화를 수행하고, 모든 IDLE 및 전송 패킷들을 버리고, 남은 패킷들의 번들(bundle)들을 원격 호스트(830) - 예를 들어, 패킷 스트림 해석이 수행되는 퍼스널 컴퓨터 또는 워크 스테이션 - 로 송신한다.

[0083] ● 워드 1 분석 – 패킷들을 3-워드 폭의 데이터 버스로 변환한다.

[0084] ● 더블릿(doublet)들(RESYNC, OVERFLOW 등)을 검사하고, 그리고 (a) 더블릿들이 연속적인 워드들에 없을 때, 또는 (b) 미구현된 워드 1 코드 값이 나타날 때, 동기화 에러들을 플래그(flag)한다.

[0085] ● IDLE 및 다른 전송 패킷들을 버리고, 그리고 남은 값들을 FIFO에 스택(stack)한다.

[0086] ● 전체 패킷들을 원격 호스트에 송신한다.

[0087] TRSTALL을 어서트할 때, 수신기(820)는 재동기화 파이프라인에 인큐잉(en-queueing)될 수 있는 2개의 워드들을 더하여 최대 6개 더 많은 TRDAT 워드들(2개의 연속적인 FPC 패킷들의 상당물)을 받아들이도록 준비된다. TRSTALL을 해제할 때, 패킷 워드 1 정렬이 보장된다. IDLE 패킷들의 스트링으로부터, 대부분이 또는 때때로 모두가 버려져서 원격 호스트에 발송되지 않아 대역폭을 줄일 수 있다. 미구현된 연산 코드들 및 FPC 워드 3(7'h5x)은 또한 동기화 에러들로서 플래그되어야 하고 1-워드 패킷들로서 취급되어야 한다.

[0088] FPC의 제 3 워드는 단일-워드 패킷의 스타일(예컨대, 코드 7'h5x)로 인코딩된다. 동기화 여부에 상관없이, 예를 들어 7'h5x의 임의의 워드 다음에 오는 워드는 새로운 패킷의 워드 1이라고 가정될 수 있다. 워드 1을 추적하는 목적을 위해, OVERFLOW, RESYNC, SLEEP 및 END 패킷들은 1-워드 패킷들로서 취급되어야 한다. 7'h7D가 1-워드 패킷으로서 취급될 때, 이후 다음의 패킷은 또 하나의 7'h7D(한 쌍의 RESYNC 워드)이더라도 항상 워드 1의 값일 것이다. 워드 1-정렬 (의사(pseudo)) 데이터는 더블릿 분석에 전달되고, 더블릿 분석에서는 두 RESYNC가 스트림에 연속적으로 나타나지 않을 때 동기화 실패가 인식된다. 이와 마찬가지로, OVERFLOW(7'h77)는 수신기가 아웃 오브 싱크일 때 나타날 수 있으며, 제 2 워드가 매칭되지 않더라도 정확히 해석되어야 한다. 이는 또한 동기화 에러를 플래그할 수 있다. 유사한 추론이 SLEEP 및 END 패킷들에 적용될 수 있는데, 왜냐하면 수신기가 아웃 오

브 싱크이면, 단지 하나의 7'h7C 또는 7'h7F가 끝에 나타날 수 있고, 수신기가 제 2 값을 대기하는 동안 중단되는(hanging) 것이 부적합할 것이기 때문이다. 이와는 정반대로, 3개의 동일한 워드들이 나타날 수 있는데, 어느 것도 수신기를 혼동하지 말아야 한다.

[0089] 완벽한 분석을 위해, 호스트(830)는, 트레이스 데이터를 마이크로프로세서를 프로그래밍하는데 사용된 오리지널 소스 코드와 비교해야 한다. 대부분의 명령어들에 대해, 트레이스 데이터는 실행이 발생했음을 선언하지만, 동작 세부 사항들은 포함되지 않는다. 분기(branch)들이 EX 패킷들(TRFPCB = 0)만을 생성할 때, 분기 목적지는 단지 소스 코드를 검사함으로써 결정될 수 있다. 유사하게, 특정 실시예에 따르면, PCLATH에의 기록은 단지 부분적인 데이터만을 방출하고, 그리고 평가를 완료하기 위해 소스 지식을 필요로 한다. 동작에 의존하여, STALL 패킷들은 영향을 받는 명령어에 선행하거나 뒤따를 수 있다. STALL 패킷들은 사용자의 디스플레이 상에 강조 표기법을 써서, 비전형적인 동작이 일어났다는(예컨대, 파일 선택 레지스터(file select register; FSR)가 비휘발성 메모리에 기록함) 암시로 보여질 수 있다. WATCH 보고들은 많은 패킷들에 의해 트리거링 명령어를 지연시킬 수 있다. 밀집도가 높은 워치들은, 동일한 워치가 판독이 발생하기 전에 트리거하면 실제로 손실될 수 있다. 워치 포인트들이 데이터 어드레스만을 확인하기 때문에, 트레이스 분석은 데이터 액세스 포인터 값들이 (BSR 및 연산 코드의 지식을 필요로 하는) 직접-어드레싱 모드들로부터의 값들이든 (FSR들의 지식을 필요로 하는) 간접 모드들로부터의 값들이든 간에, 이 데이터 액세스 포인터 값들을 재구성할 수 있어야 한다. 워치 데이터는 항상 가장 최근 발생한 워치 이벤트로부터의 데이터이어야 한다. 몇몇 FPC 패킷들은 명령어들(예컨대, RETURN 명령어들)이 실행되었음을 암시하지만, 다른 경우들은 단순히 정보임을 주의한다.

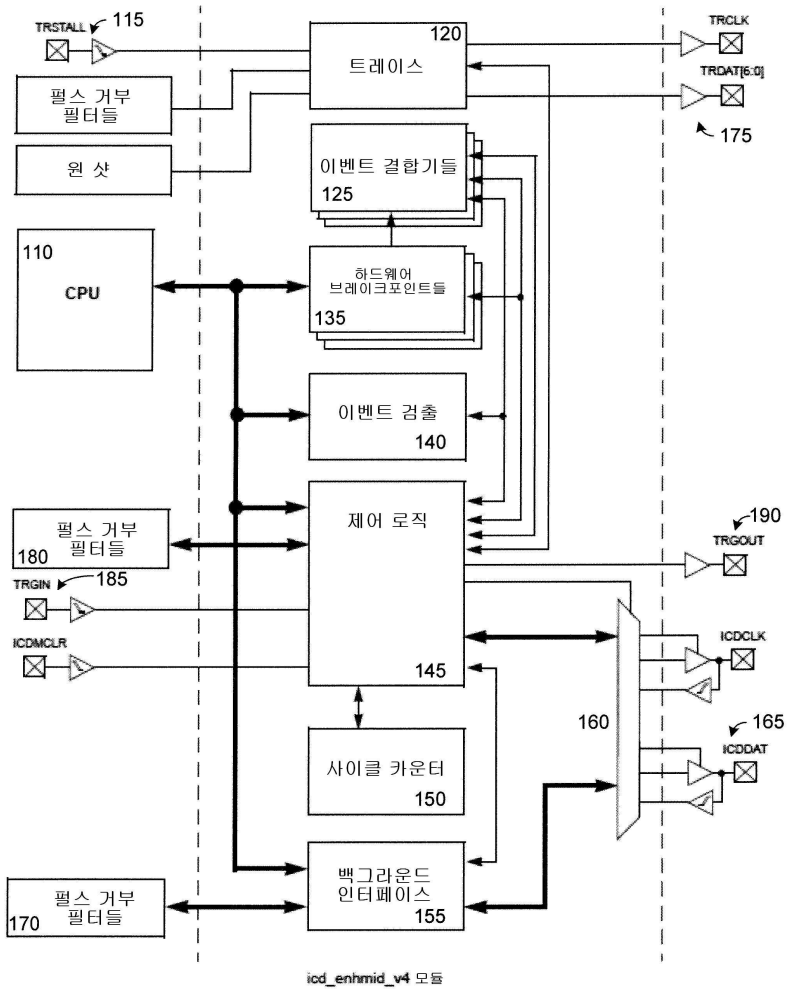
[0090] 도 12는 개발 프로그램을 실행하고 그리고 예를 들어 USB 인터페이스를 통해 외부 디버거/프로그래밍 유닛(520)과 연결되는 퍼스널 컴퓨터와 같은 호스트를 갖는 전형적인 디버깅/프로그래밍 시스템(500)을 도시한다. 외부 디버거 프로그래밍 유닛(520)은 디버거/프로그래머(520)의 내부에 생성된 전원 전압을 공급할 수 있는 전용 인터페이스를 제공한다. 그러나, 다른 실시예들은 전용 전원을 통해 공급 전압을 제공할 수 있거나, 또는 타겟 시스템은 자체 전원이 공급될 수 있다. 실제 디버그/프로그래밍 인터페이스는 동기식 직렬 인터페이스에 의해, 디버거/프로그래밍 유닛(520)에 의해 제공된 단방향 클록 신호(ICD_{CLK}) 및 양방향 데이터 라인(ICD_{Data})이 제공될 수 있다. 따라서, 최소 3개의 연결 라인들에서, ICD_{CLK}, ICD_{Data} 및 기준 포텐셜(GND)이 디버거/프로그래밍 유닛(520)을 타겟 시스템(510)과 결합시키는데 사용될 수 있으며, 최소한 상기 타겟 시스템(510)은 위에 개시한 바와 같은 다양한 실시예들에 따른 디버깅/프로그래밍 인터페이스를 갖는 마이크로컨트롤러일 수 있다.

[0091] 이러한 시스템은 사용자가 호스트 상에 실행되는 디버깅 프로그램을 프로그래밍할 수 있게 하여, 위에 설명한 바와 같은 조건들을 갖는 여러 브레이크포인트들을 설정하고, 그리고 프로그램이 실시간 실행되고 있는 동안에 부가 연결 라인들을 통해 트레이스 정보를 임의로 수신한다. 디버깅 소프트웨어가 소스 코드 내의 브레이크포인트들의 위치에 관련하여 여러 브레이크포인트들의 추적을 유지하는 동안, 디버거/프로그래머(520)는 브레이크포인트 정보를, 각각의 브레이크포인트들을 설정하고 그 연관된 레지스터들을 구성하는 타겟 디바이스에 전달한다. 또한, 트레이스 기능들의 설정 및 구성은 디버거/프로그래머(520)에 의해 타겟 디바이스(510)에 전달된다. 예를 들어, 메모리에 저장된 데이터 값의 매치에 의해 트리거되는 특정 브레이크포인트가 설정될 수 있다. 이후 사용자는 호스트 PC(530) 상에 실행되는 디버거 소프트웨어를 통해 타겟 디바이스(510)의 소프트웨어의 실행을 개시한다. 타겟 소프트웨어의 실행은 단지 브레이크포인트가 검출될 때 중지된다. 그러나, 트레이스 정보는 타겟 프로그램의 실행 동안 계속적으로 전송될 수 있다. 호스트 컴퓨터(530)는 이 트레이스 데이터를 평가할 수 있고, 그리고 트레이스 데이터를 텍스트 형성 및 그래픽 디스플레이의 사용 중 어느 하나의 경우에 사용할 수 있게 한다. 위에 개시한 바와 같이, 다양한 실시예들에 따라 트레이스 서브시스템이 여전히 클록킹되는 경우, 리셋 이벤트는 또한 완전히 추적될 수 있다.

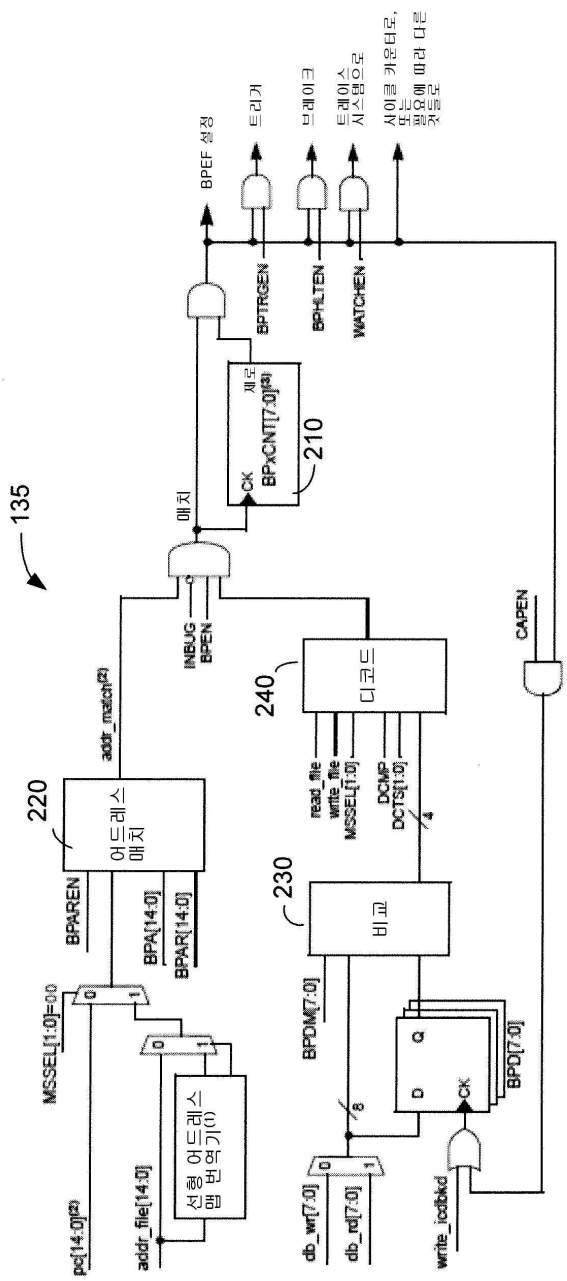
[0092] 본 실시예들은 위에 논의된 바와 같은 특징들을 구현함으로써 트레이스 스트림의 더 좋은 분석을 가능하게 한다. 그러므로, 다양한 실시예들은 온(on)-칩 디버그 기능들의 기술 상태를 향상시키고, 그리고 더 많은 다른 사용자들에게 첨단 디버깅 기능들을 가져다줄 것이다.

도면

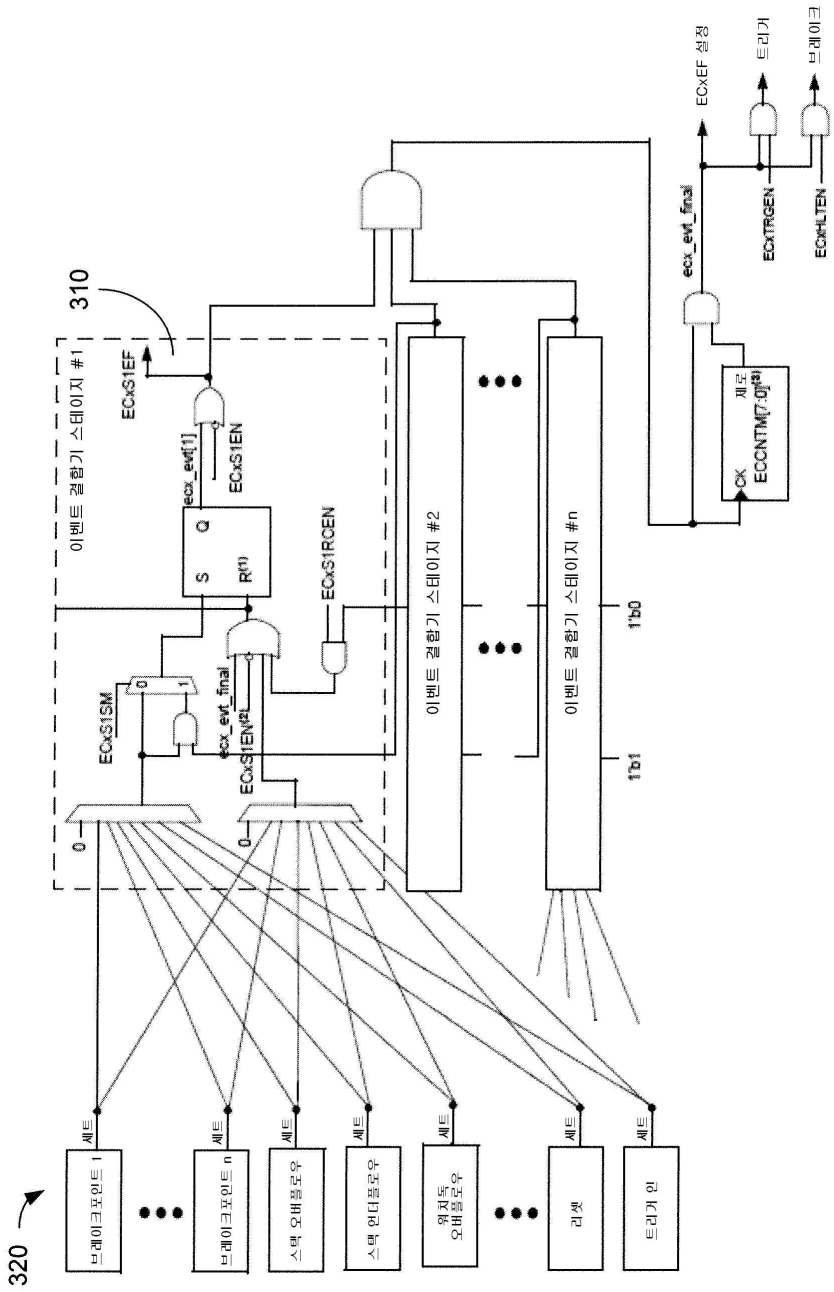
도면1


TRSTALL
115
120
트레이스
TRCLK
TRDAT[6:0]
175
펄스 거부
필터들
원 샷
이벤트 결합기들
125
110
CPU
하드웨어
브레이크포인트들
135
이벤트 검출
140
제어 로직
펄스 거부
필터들
180
190
TRGOUT
TRGIN
185
ICDMCLR
145
ICDCLK
160
165
ICDDAT
사이클 카운터
150
백그라운드
인터페이스
155
펄스 거부
필터들
170
icd_enhmid_v4 모듈

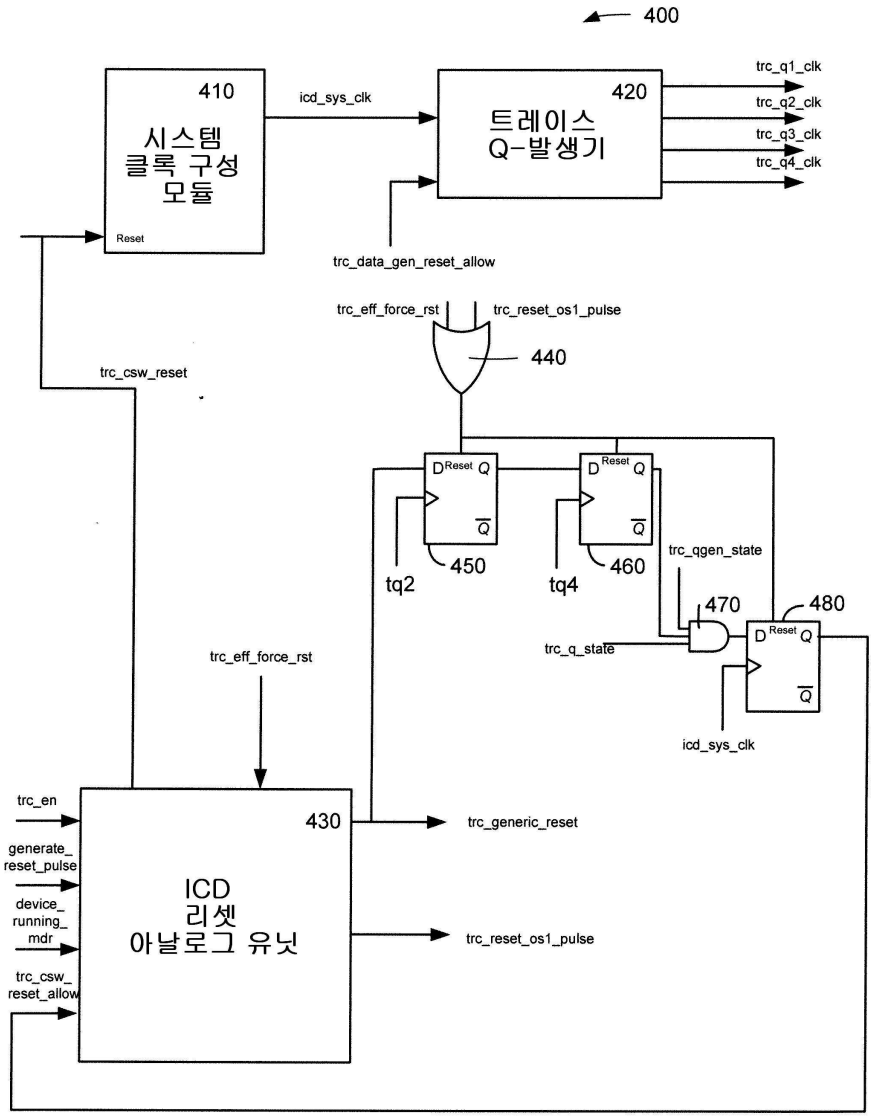
도면2


135
BPEF 설정
트리거
브레이크
트레이스 시스템으로
사이클 카운터로, 또는 플랫에 따라 다른 것들로
BPTRGEN
BPHLTEN
WATCHEN
CAPEN
매치
CK
BPxCNT[7:0]
제로
210
INBUG
BPEN
240
디코드
read_file
write_file
MSSEL[1:0]
DCMP
DCTS[1:0]
addr_match[2]
220
어드레스 매치
BPAREN
BPA[14:0]
BPAR[14:0]
230
비교
BPDM[7:0]
MSSEL[1:0]=0,0
선형 어드레스 램 번역기(i)
addr_file[14:0]
pc[14:0][2]
db_wr[7:0]
db_rd[7:0]
write_icdbkd
D
Q
CK
BPD[7:0]

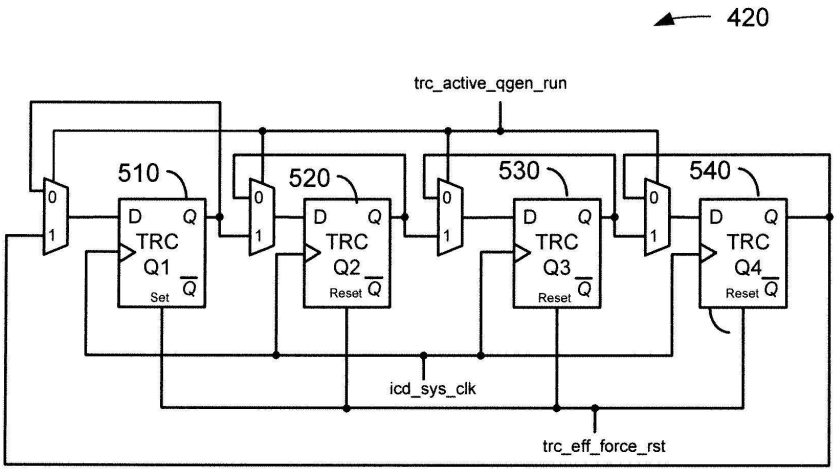
도면3


300
310
320
이벤트 결합기 스테이지 #1
ECxS1EF
ECxS1EN
ecx_evt[1]
ECxS1RCEN
ECxS1SM
ecx_evt_final
ECxS1EN[2:1]
이벤트 결합기 스테이지 #2
이벤트 결합기 스테이지 #n
150
151
ecx_evt_final
ECxEF 설정
트리거
브레이크
ECxTRGEN
ECxHLTEN
OK
ECCNTM[7:0]
제로
세트
브레이크포인트 1
브레이크포인트 n
스택 오버플로우
스택 언더플로우
워치독 오버플로우
리셋
트리거 인

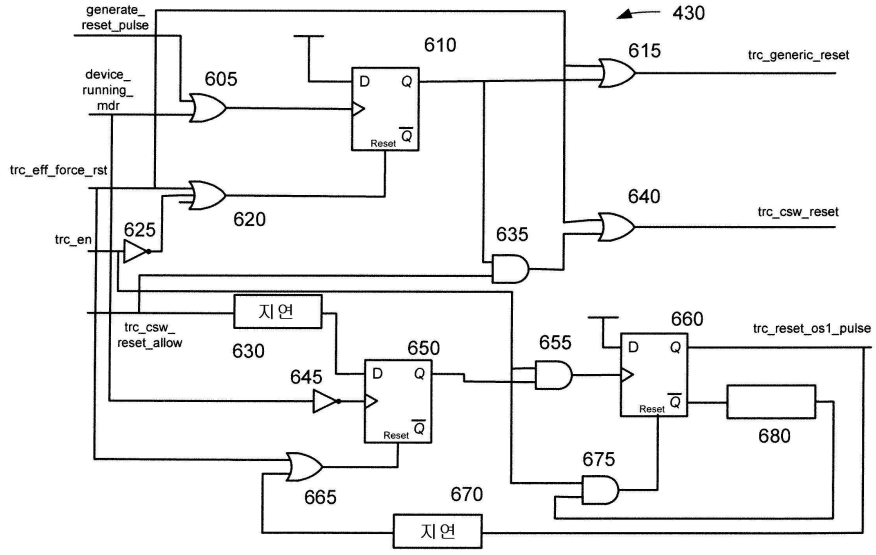
도면4


400
410
시스템
클록 구성
모듈
Reset
icd_sys_clk
420
트레이스
Q-발생기
trc_q1_clk
trc_q2_clk
trc_q3_clk
trc_q4_clk
trc_data_gen_reset_allow
trc_eff_force_rst
trc_reset_os1_pulse
440
trc_csw_reset
D Reset Q
Q
450
tq2
460
tq4
trc_qgen_state
470
480
trc_q_state
icd_sys_clk
trc_eff_force_rst
trc_en
generate_
reset_pulse
device_
running_
mdr
trc_csw_
reset_allow
430
ICD
리셋
아날로그 유닛
trc_generic_reset
trc_reset_os1_pulse

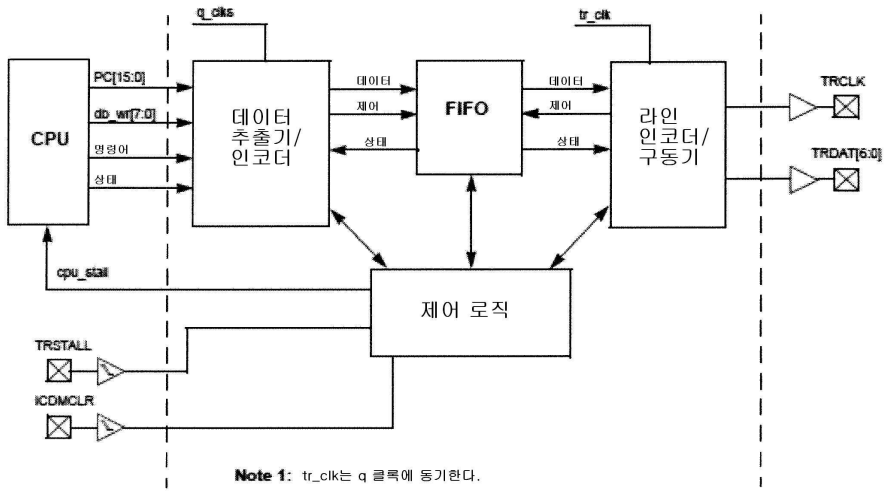
도면5


420
trc_active_qgen_run
510
520
530
540
TRC Q1
TRC Q2
TRC Q3
TRC Q4
Set
Reset
icd_sys_clk
trc_eff_force_rst

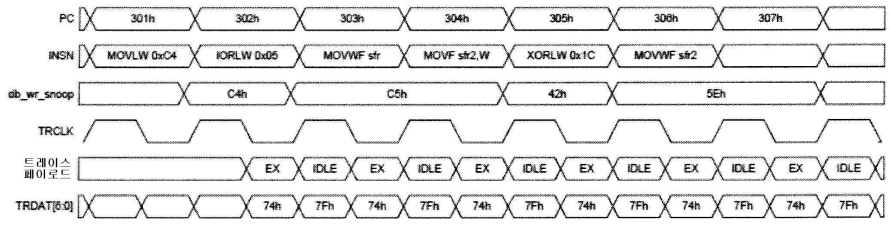
도면6


430
generate_ reset_pulse
device_ running_ mdr
trc_eff_force_rst
trc_en
trc_csw_ reset_allow
605
610
615
620
625
630
635
640
645
650
655
660
665
670
675
680
trc_generic_reset
trc_csw_reset
trc_reset_os1_pulse
지연
지연

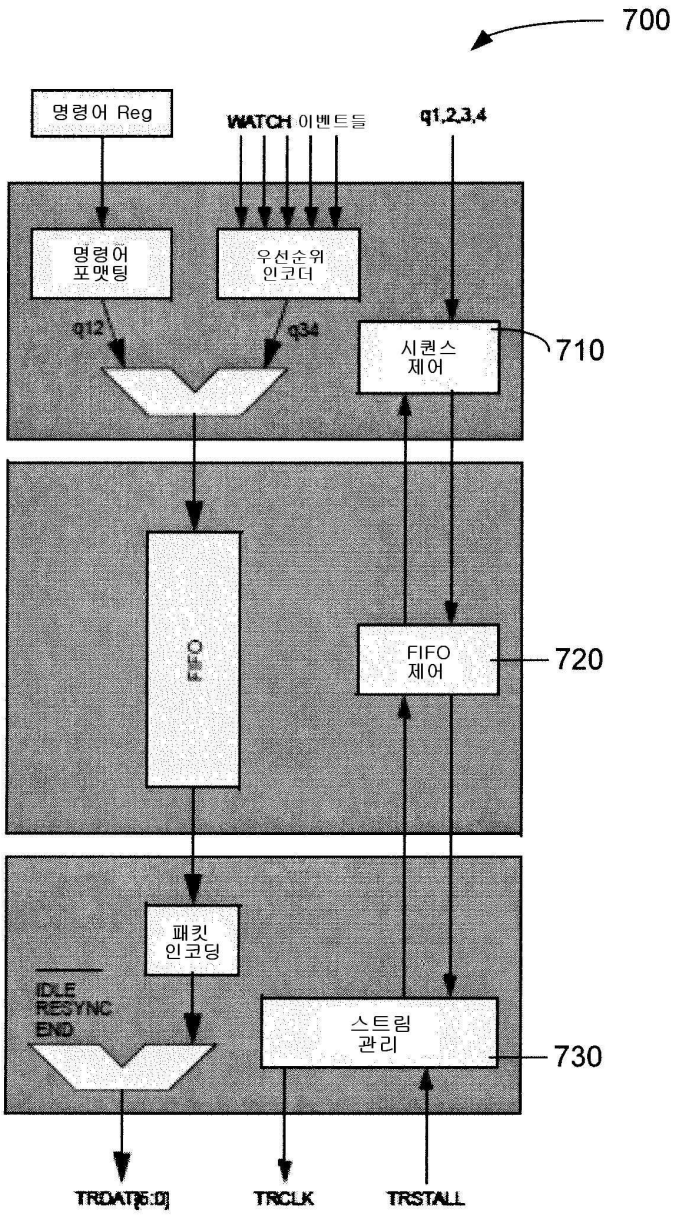
도면7


q_clks
tr_clk
CPU
PC[15:0]
db_wr[7:0]
명령어
상태
데이터 추출기/ 인코더
데이터
제어
상태
FIFO
데이터
제어
상태
라인 인코더/ 구동기
TRCLK
TRDAT[6:0]
cpu_stall
제어 로직
TRSTALL
ICDMCLR
Note 1: tr_clk는 q 클록에 동기한다.

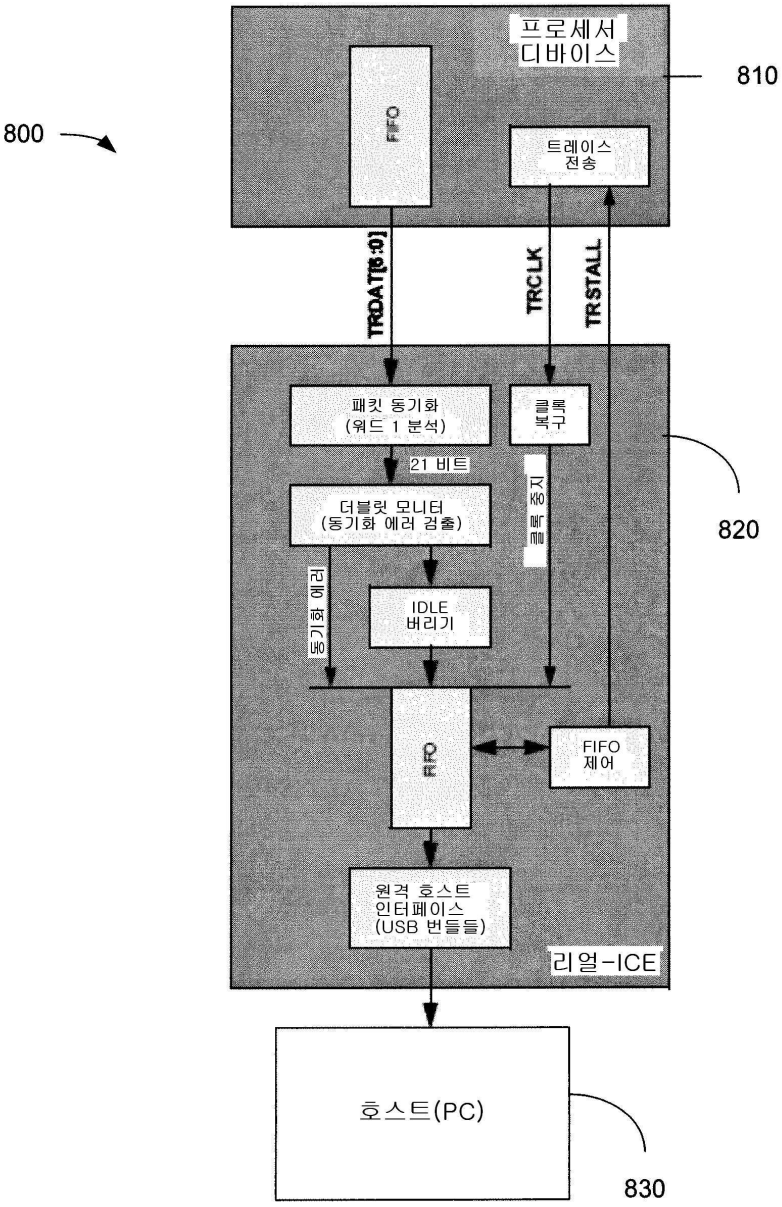
도면8


PC
301h
302h
303h
304h
305h
306h
307h
INSN
MOVLW 0xC4
IORLW 0x05
MOVWF sfr
MOVF sfr2,W
XORLW 0x1C
MOVWF sfr2
db_wr_snoop
C4h
C5h
42h
5Eh
TRCLK
트레이스 페이로드
EX
IDLE
TRDAT[6:0]
74h
7Fh

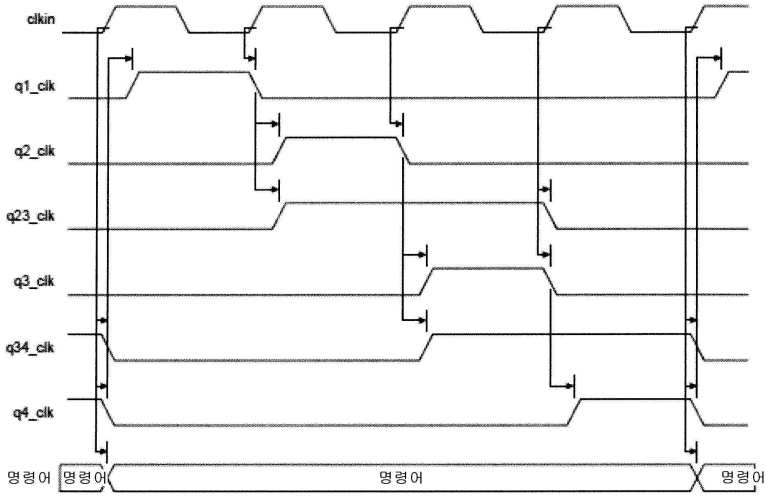
도면9


700
명령어 Reg
WATCH 이벤트들
q1,2,3,4
명령어 포맷팅
우선순위 인코더
q12
q34
시퀀스 제어
710
FIFO
FIFO 제어
720
패킷 인코딩
IDLE
RESYNC
END
스트림 관리
730
TRDAT[7:0]
TRCLK
TRSTALL

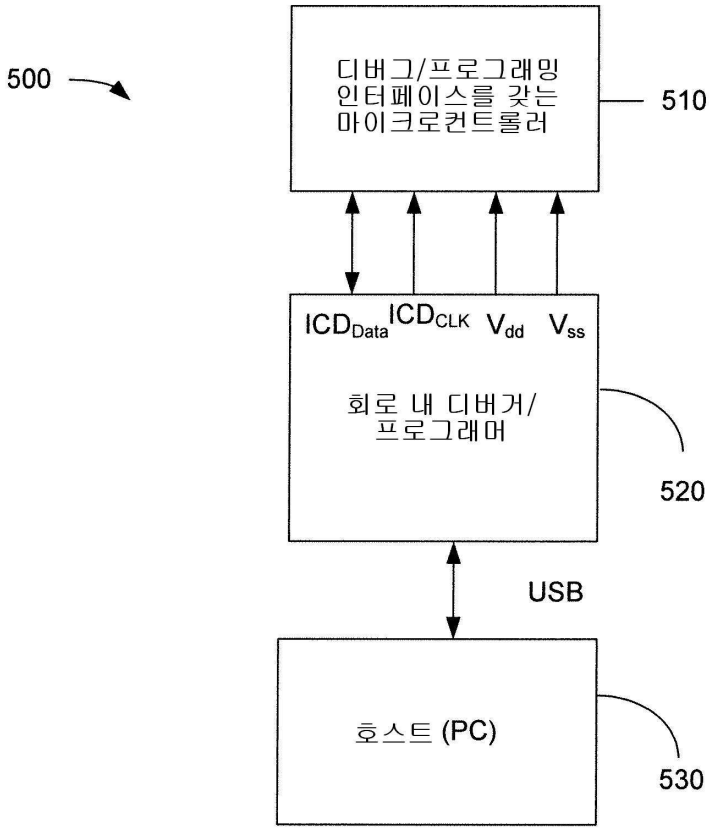
도면10


800
프로세서
디바이스
810
FIFO
트레이스
전송
TRDATA[8:0]
TRCLK
TRSTALL
패킷 동기화
(워드 1 분석)
클록
복구
21 비트
더블릿 모니터
(동기화 에러 검출)
클록 중지
동기화 에러
IDLE
버리기
820
FIFO
FIFO
제어
원격 호스트
인터페이스
(USB 번들들)
리얼-ICE
호스트(PC)
830

도면11


clkin
q1_clk
q2_clk
q23_clk
q3_clk
q34_clk
q4_clk
명령어
명령어
명령어
명령어

도면12


500
디버그/프로그래밍
인터페이스를 갖는
마이크로컨트롤러
510
ICDData ICDCLK Vdd Vss
회로 내 디버거/
프로그래머
520
USB
호스트 (PC)
530