(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2009년02월03일
(11) 등록번호 10-0881843
(24) 등록일자 2009년01월28일

(51) Int. Cl.[9]
G11C 29/00 (2006.01)
(21) 출원번호 10-2002-0021258
(22) 출원일자 2002년04월18일
심사청구일자 2007년04월18일
(65) 공개번호 10-2002-0081673
(43) 공개일자 2002년10월30일
(30) 우선권주장
09/838,766 2001년04월19일 미국(US)
(56) 선행기술조사문헌
US5771240 A
US6253338 B1
JP2000310653 A
KR100439781 B1
전체 청구항 수 : 총 10 항

(73) 특허권자
베리지 (싱가포르) 피티이. 엘티디.
싱가포르 768923 이슌 애비뉴 7 넘버. 1
(72) 발명자
크레치알렌에스2세
미국콜로라도주80525포트콜린스레드베리코트1518
리크브레드디
미국콜로라도주80538
러브랜드파이어손드라이브8529
(뒷면에 계속)
(74) 대리인
김창세, 장성구

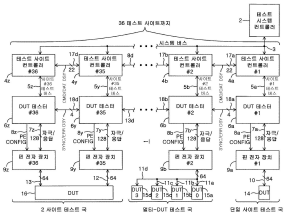
심사관 : 장호근

(54) 트리거 신호 생성 방법 및 전압 파형 표현 생성 방법

(57) 요 약

알고리즘적 테스트 프로그램(19)을 갖는 메모리 테스터에 대한 트리거 신호는 DUT(14)를 동작하기 위해 사용되는 기존의 하드웨어 양(97, 24)에 의해 표현되는 트리거 특수화의 발생을 검출한다(111). 이는 테스트 프로그램의 어느 부분이 실행되고 있는지에 따라 더 자격을 갖출 수 있는(95) 원래의 하드웨어(브레이크포인트) 트리거(96)를 형성한다. 자격있는 브레이크포인트 트리거는 스코프 모드를 트리거하고, 테스트 프로그램에 특정 경로를 강제하기 위해 선택된 값(147a - c)으로 에러 플래그(152, 153, 154)를 집어넣기 위해 사용될 수 있는 시스템 트리거 신호(115)가 되기 전에 0 또는 그 이상의 DUT 사이클 만큼 지연될 수 있다(114). 사용자는 테스트 프로그램의 일부가 아닌 프로세스와 상호작용하여, 하드웨어 레지스터 값이 레벨에서의 원래의 트리거 조건을 인식하는 마스크 및 비교 메카니즘(86)으로부터 트리거 특수화를 규정한다. 그 처리는 또한 컴파일러에게 테스트 프로그램의 어느 부분이 원래의 트리거 특수화를 가능하게 할 것인지(인스트럭션 워드(instruction words)내에 비트(93)를 설정함으로써 행하여짐)에 대해 알려준다. 전압 임계치(117, 118) 및 샘플 타이밍 오프셋(179)의 스위핑에 대해 안정된 파형을 제공하기 위해, 메모리 테스터는 트리거의 발생 후에 테스트 프로그램을 통해 초기 통과 동안 방출되는 전송 벡터의 타겟 시퀀스(target sequence)에 대한 어드레스(63)를 기록한다(130). 이러한 어드레스는 인스트럭션 그 자체로 교환되고, 그 후, 그 인스트럭션은 분기를 제거하기 위해 변경되어 그 인스트럭션의 근원인 메모리의 예비 부분내에 저장된다. 변경된 타겟 시퀀스가 저장되면, 원하는 정보가 전체 테스트 프로그램을 재시작하고 트리거로 다운되기 전과 같이 정확히 실행하게 함으로써 생성된다. 이제 트리거가 발생할 때 전송 벡터가 진행 중인 알고리즘보다는 기억된 타겟 시퀀스로부터 방출되며, 전압(166) 임계치 및 샘플 타이밍 오프셋(168)의 조합이 알맞게 전환된다(164, 169). 그 조합은 그 값의 획득 스위프를 따라가는 하나의 단계를 구성한다. 타겟 시퀀스의 수신 벡터는 그것이 도착하는 대로 저장된다(141, 142, 32b). 타겟 시퀀스 후에 테스트 프로그램은 정상 임계치 및 샘플 타이밍 오프셋으로 재시작된다. 결국 또다른 트리거가 존재하는데, 그 위에서 타겟 시퀀스가 다시 대체되고, 획득 스위프를 따라가는 단계에서의 다음 조합이 이루어진다. 이 프로세스는 전체 획득 스위프가 수행 완료될 때까지 계속된다. 저장된 데이터를 조사해보면 파형의 생성을 알 수 있다.

대 표 도



(72) 발명자

베일리랜디엘

미국콜로라도주80526
포트콜린스칩펜데일드라이브4616

프리즈맨존엠

미국콜로라도주80526포트콜린스그레이폭스로드4507

특허청구의 범위

청구항 1

테스터 내의 복수의 연산 장치(Arithmetic Logical Unit; ALU)(24) 하드웨어 레지스터의 내용(contents) - 상기 내용(27, 28)은 상기 테스터와 피시험 장치(Device Under Test; DUT) 사이의 관련 신호 트래픽을 나타냄 - 을 조작하는 알고리즘적 특성을 갖는 테스트 프로그램을 실행하는 테스터에서 트리거 신호(96, 113, 115, 122)를 생성하는 방법에 있어서,

(a) 상기 복수의 ALU 하드웨어 레지스터 각각에 대해, 상기 트리거 신호를 생성하는 데 기여할 비트 위치(bit positions)의 임의의 조합을 선택하는 단계(99, 100)와,

(b) 개별 트리거 값 레지스터(98)를 각각의 ALU 하드웨어 레지스터와 관련시키는 단계와,

(c) 상기 단계 (a)에서 선택된 적어도 하나의 비트 위치를 갖는 ALU 하드웨어 레지스터와 관련된 트리거 값 레지스터 각각에 대해, 상기 관련 ALU 하드웨어 레지스터가 생성될 트리거 신호에 대해 가질 비트 값에 대한 비트를 상기 선택된 비트 위치와 동일한 비트 위치로 로딩(loading)하는 단계(85)와,

(d) 트리거 값 레지스터의 각각에 대해, 대응하는 ALU 하드웨어 레지스터마다 상기 단계 (a)에서 선택된 것과 동일한 비트 위치의 조합을 선택하는 단계(99,101)와,

(e) 각각의 ALU 하드웨어 및 관련 트리거 값 레지스터 쌍(pair)(104,105,...,110)에 대해, ALU 하드웨어 비트의 선택된 조합과 관련 트리거 값 레지스터 내의 선택된 비트 조합이 동일한지를 비트별로 비교하는 단계(103)와,

(f) 상기 단계 (e)에서 동일하다고 판단되면, 트리거 신호(96)를 생성하는 단계(111)를 포함하는

트리거 신호 생성 방법.

청구항 2

제 1 항에 있어서,

(i) 상기 단계 (a)는 각각의 ALU 하드웨어 레지스터(97)와 서로 다른 마스크 레지스터(99)를 관련시키는 단계와, 각각의 마스크 레지스터로 하나의 값 - 상기 값은 상기 대응하는 ALU 하드웨어 레지스터 내에서 선택될 상기 비트 위치 내의 값임 - 을 로딩하는 단계와, 각각의 ALU 하드웨어 레지스터와 관련 마스크 레지스터 사이에서 비트별 AND 연산(a bit-wise AND'ing)(100) - 상기 비트별 AND 연산의 결과는 각각의 ALU 하드웨어 레지스터에 대한 비트 위치의 선택된 조합의 내용을 나타냄 - 을 수행하는 단계를 더 포함하고,

(ii) 상기 단계 (c)는 각각의 트리거 값 레지스터(98)와 서로 다른 마스크 레지스터(99)를 관련시키는 단계와, 각각의 마스크 레지스터로 하나의 값 - 상기 값은 상기 대응하는 트리거 값 레지스터 내에서 선택될 상기 비트 위치 내의 값임 - 을 로딩하는 단계와, 각각의 트리거 값 레지스터와 관련 마스크 레지스터 사이에서 비트별 AND 연산(101) - 상기 비트별 AND 연산의 결과는 각각의 트리거 값 레지스터에 대한 비트 위치의 선택된 조합의 내용을 나타냄 - 을 수행하는 단계를 더 포함하는

트리거 신호 생성 방법.

청구항 3

제 1 항에 있어서,

상기 테스트 프로그램에 대한 실행가능한 인스트럭션 워드 내에 트리거 인에이블 비트(a trigger enable bit)(93)를 포함하는 단계 - 상기 트리거 인에이블 비트(93)는 트리거 신호의 생성을 허용하도록 의도된 상기 테스트 프로그램의 일부에 대응하는 상기 실행가능한 인스트럭션 워드(92) 내에 포함됨 - 를 더 포함하고, 상기 단계 (f)는 상기 트리거 인에이블 비트가 상기 트리거 신호의 생성을 허용하기 위해 존재한다는 조건을 더 포함하는

트리거 신호 생성 방법.

청구항 4

제 1 항에 있어서,

(g) 상기 테스트 프로그램 내의 분기(branching)에 영향을 주는 에러 플래그(error flags)에 대해 선택된 대체 값(147a - c)을 규정하는 단계와,

(h) 상기 단계 (f)에서의 트리거 신호의 생성 후에, 상기 에러 플래그의 실제 값(152, 153, 154)을 대응하는 대체 값으로 대체하는 단계를 더 포함하는

트리거 신호 생성 방법.

청구항 5

제 1 항에 있어서,

(g) 상기 단계 (f)에서의 트리거 신호의 생성 후에, 상기 테스트 프로그램에 의해 실행되는 선택된 수의 연속적인 인스트럭션 워드인 타겟 시퀀스를 포착하는 단계(130, 71, 72)와,

(h) 상기 단계 (g) 후에, 상기 테스트 프로그램을 반복적으로 재시작하여 트리거 신호 발생시에 포착된 타겟 시퀀스를 이용하여 실행을 계속하도록 분기하는 단계(66)와,

(i) 상기 단계 (h)에 의해 포착된 타겟 시퀀스를 반복적으로 실행하는 동안, 상기 DUT의 선택된 채널에 대한 전압 임계치를 변화시키고, 샘플 타이밍 오프셋을 변화시키는 단계(76)와,

(j) 상기 단계 (i)에서 발생한 임계치 비교 데이터를 저장하는 단계(77, 80)와,

(k) 상기 단계 (j)에서 저장된 상기 임계치 비교 데이터로부터 상기 선택된 채널에 대한 전압 파형의 표현을 생성하고, 상기 트리거 신호 이후의 포착된 타겟 시퀀스의 실행에 대응하는 시구간을 나타내는 단계를 더 포함하는

트리거 신호 생성 방법.

청구항 6

제 1 항에 있어서,

상기 생성된 트리거 신호의 온셋(onset)을 선택된 양만큼 지연시키는 단계(114)를 더 포함하는

트리거 신호 생성 방법.

청구항 7

제 1 항에 있어서,

상기 테스터는 메모리 테스터이고, 상기 DUT(14)는 메모리인

트리거 신호 생성 방법.

청구항 8

테스터에 의해 동작되는 피시험 장치(Device Under Test; DUT)(14)의 선택된 채널에 대한 전압 파형의 표현을 생성하는 방법 - 상기 테스터는 테스트 프로그램에 의해 제공되는 자극에 대한 DUT의 응답에 기초하는 분기를 포함하는 알고리즘적 특성을 갖는 테스트 프로그램을 실행함 - 에 있어서,

(a) 상기 테스트 프로그램의 선택된 부분(93)의 실행을 포함하여, 상기 테스터의 하드웨어 내의 선택된 상태(96)의 동시 발생을 나타내는 트리거 신호(113)를 생성하는 단계와,

(b) 상기 단계 (a)에 의해 상기 트리거 신호를 최초로 생성한 후에, 상기 트리거 신호의 발생 이후에 상기 테스트 프로그램에 의해 실행되는 선택된 수의 연속적인 인스트럭션인 인스트럭션의 타겟 시퀀스(P/O 92)에 대응하는 어드레스의 타겟 시퀀스(P/O 63)를 포착하는 단계(65, 71, 130)와,

(c) 상기 단계 (b) 후에, 상기 포착된 어드레스의 타겟 시퀀스에 의해 식별되는 인스트럭션의 타겟 시퀀스를 인

출(fetching)하는 단계와,

(d) 상기 단계 (c) 후에, 상기 인출된 인스트럭션의 타겟 시퀀스를 변경하여 상기 시퀀스 내의 다음 인스트럭션을 가리키도록 임의의 분기를 조정하는 단계(72)와,

(e) 상기 단계 (d) 후에, 상기 변경된 인스트럭션의 타겟 시퀀스를 상기 테스트 프로그램이 실행되는 프로그램 메모리의 예비 부분 내에 저장하는 단계(P/O 72)와,

(f) 단계 (f6)가 선택된 채널에 대한 전압 파형 표현을 생성하기 위해 비교 결과들을 저장할 때까지 아래 단계들,즉

(f1) 상기 테스트 프로그램을 재시작하는 단계(73),

(f2) 상기 트리거 신호(122)의 후속 인스턴스(subsequent instance)를 생성하는 단계,

(f3) 상기 테스트 프로그램의 실행을 인터럽트하는 단계(P/O 76),

(f4) 선택된 채널에 대해 전압 비교 임계치 및 샘플 타이밍 오프셋의 선택된 값을 유효하게 하는 단계(P/O 76),

(f5) 변경되어 저장된 인스트럭션의 타겟 시퀀스의 시작에서 상기 테스트 프로그램의 실행을 재개하는 단계(P/O 77),

(f6) 상기 선택된 채널에 대한 비교 결과들을 저장하는 단계(P/O 77, 80)를

순서대로 반복하는 단계(82)와,

(g) 상기 저장된 비교 결과들로부터 상기 선택된 채널에 대한 전압 파형의 표현을 생성하는 단계를 포함하는

전압 파형 표현 생성 방법.

청구항 9

제 8 항에 있어서,

상기 단계 (a) 및 (f2)는 상기 트리거 신호의 온셋을 선택된 양만큼 지연시키는(114)

전압 파형 표현 생성 방법.

청구항 10

제 8 항에 있어서,

상기 테스터는 메모리 테스터이고, 상기 DUT(14)는 메모리인

전압 파형 표현 생성 방법.

명 세 서

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

<6> 본원은 2000년 9월 20일 출원되었으며 발명의 명칭이 ERROR CATCH RAM FOR MEMORY TESTER HAS SDRAM MEMORY SETS CONFIGURABLE FOR SIZE AND SPEED인 미국 특허 출원 S/N 09/665,892에 개시된 정보에 관한 것이며, 이는 후술하는 이유로 인해, 참조용으로 여기에 명백히 포함시켜 둔다. 본원의 청구 대상(subject matter)은 반도체 메모리를 테스트하기 위한 상당히 크고 복잡한 시스템의 동작의 일부에 관한 것이다. 설명하는 메모리 테스터는 그 자체내에 테스트를 수행하기 위한 전체 패러다임내의 구성요소로서 광범위한 메모리 서브시스템을 포함한다. 그 메모리 서브시스템의 특정의 능력이 향후 개시될 신규한 청구 대상의 일부에 대한 바람직한 근거가 된다는 점에서 여기에서의 관심의 대상이다. 제품에 있어서의 경제상의 이유 및 테스터내에 이용가능한 다량의 메모리를 포함하려고 하는 요구에 이끌려서, 심지어 랜덤하게 어드레스될 때에도 고속의 값 비싼 SRAM에 대한

대용물로서 값 싼 메모리(랜덤하게 액세스될 때 속도가 느린 DRAM)를 사용하는 방법이 개발되었다. 그 결과는, 다양한 다른 메모리 서브시스템 특징과 조합될 때, 가변 워드 폭(variable word width) 등을 구현하는 것뿐만 아니라, 그룹(Group)들 사이에서 멀티플렉싱(multiplexing)하고 뱅크(Bank)들 사이에서 인터리빙(interleaving)하는 것을 수반하는 매우 복잡한 일이다. 한편, 우리가 여기에 개시하고자 하는 특유의 특징은 복잡성이 상당히 감소된 SRAM만을 사용하는 시스템에서 구현될 수 있다. 그러나, 완성품을 성장가능한 상업적 기술이 아니라 공학적 호기심이 되도록 할 수 있는 상당한 경제적 불이익이 발생할 것이다. 우리는 본원에서 중간적 입장을 취했으며, 여기에서 우리는 SRAM이 확실하게 동작할 수 있을 지라도 전적으로 SRAM으로부터 시스템을 만들려고 하지 않았다. 물론 우리는 DRAM 기술을 포함하고 있지만, 그의 기본적인 원리에 대한 적당한 설명을 곁들였을 뿐, DRAM 기술의 내부 동작에 대한 복잡한 설명의 대부분을 자제하였다. 심지어 전문가가 아닌 독자 조차도 매우 흥미있는 자료(예컨대, 서로 다른 인터리빙과 어드레싱 방법 및 그들의 서로 다른 동작 모드로의 접속)가 생략되었다고 판단할 것이다. 그러나, 모든 독자는 심사숙고하여, 우리가 개시하고자 하는 기술 및 특징이 일정한 경우에 그 생략된 자료에 의해 조금은 영향을 받을 수도 있지만, 그 생략된 자료에 근본적으로 의존하지 않는다는 사실을 이해하게 될 것이다. 그래서, 주요 관심 대상의 주위에 있는 자질구레한 주변 문제들의 실례가 있다. 위에 포함된 기재는 DRAM 기술에 대한 풍부한 세부 사항을 제공한다. 본 기재가 메모리 서브시스템에 대한 해답을 주기보다는 오히려 더 많은 문제를 야기한다고 느끼는 사람들은 그러한 해답을 위해 포함된 기재를 볼 수 있다. 그 기재의 존재가 참조용으로 여기에 포함됨으로써 알려지게 된 것은 그러한 이유 때문이다.

<7> 이 두 기재의 내용을 조합하기 원하는 사람에 대한 경고가 있다고 한다. 그 포함된 기재는 관심의 대상이 되는 메모리 전체를 에러 캐치 RAM(Error Catch RAM; ECR)이라고 부르며, 그것은 메모리 세트(Momory Sets)로 분할된다고 하는 입장을 취한다. 이러한 관점은 그 포함된 기재내에서 별 어려움없이 작용하는데, 그 이유는 그 기재내에서 ECR은 그 다른 기능의 존재가 암시되어 있을 지라도 관심의 대상이 되는 거의 유일한 메모리 기능이기 때문이다. 그러나, 본원을 준비하는 동안 심사숙고하여, "내부 테스트 메모리(Interior Test Memory)"라는 용어를 사용하여 관심의 대상이 되는 메모리 전체를 설명하는 것이 더 편하다는 것을 알았으며, 그 내부 테스트 메모리는 4개의 개별 및 독립 메모리 세트로 차례로 구성되며, 그 메모리 세트내에는 (ECR을 포함하는) 다양한 기능적 메모리 메카니즘이 적합한 구성에 의해 규정될 수 있다. 그래서, 그 포함된 기재에는 메모리 세트가 ECR에 포함되는 것처럼 보일 것이나, 여기서는 전혀 다른 방식이다. 그럼에도 불구하고, 두 기재 모두 동일한 전체 시스템내에서 발견되는 청구 대상에 관한 것이다. 그 차이는 대체로 용어상의 문제이며, 두 기재 사이에 겉으로 보기에는 모순되는 것처럼 보이는 임의의 사항도, 그것이 고려되고 있는 메모리 서브시스템 동작의 하위 레벨의 세부 항목일 때, 없어진다. 그리고, 우리가 그 포함된 기재를 지적했다는 것이 그러한 세부 항목이 흥미있는 독자에게 유용할 수 있도록 하는 것이다.

<8> 본원의 청구 대상은 또한 2000년 9월 31일에 출원되었으며 발명의 명칭이 MEMORY TESTER WITH ENHANCED POST DECODE인 미국 특허 출원 S/N 09/702,631의 청구 대상에 관한 것이다. 그 기재에서 우리는 그 도 5의 주요 부분을 빌려서, 일부 수정하고 추가한 후 본원의 도 5로서 사용하였다. 공유 부분이 본원의 내용에서 발견될 수 있는 가능성을 막고 또한 간결하게 하기 위해, 우리는 MEMORY TESTER WITH ENHANCED POST DECODE를 참조용으로 여기에 포함시키기로 하였다.

<9> 전자 장치 및 그 응용 장치는 일상 생활에서 매우 일반적이게 되었다. 가정에서의 개인용 컴퓨터와 함께, 많은 개인들은 다양한 목적을 위한 하나 이상의 생산 도구를 가지고 있다. 대부분의 개인용 생산 전자 장치는 비휘발성 메모리의 일정 형태를 포함한다. 셀 폰은 전원이 꺼질 때 사용자 프로그램된 폰 번호 및 구성을 저장하고 보유하기 위해서 비휘발성 메모리를 이용한다. PCMCIA 카드는 그 카드가 컴퓨터내의 슬롯으로부터 제거될 때에도 정보를 저장하고 보유하기 위해서 비휘발성 메모리를 이용한다. 많은 다른 일반적인 전자 장치도 전원이 꺼진 어셈블리내의 비휘발성 메모리의 장기간 저장 능력으로부터 이득을 보고 있다.

<10> 전자 장비 제조자에게 판매하는 비휘발성 메모리 제조자는 그들이 생산하는 메모리의 적절한 동작을 실행하고 증명하기 위한 테스터를 필요로 한다. 일관되게 낮은 가격으로 제조되고 판매되는 비휘발성 메모리의 부피로 인해, 단일 부품을 테스트하는데 걸리는 시간을 최소화하는 것이 매우 중요하다. 최소한의 테스트 또는 테스트 없이 메모리 장치를 보다 값 비싼 어셈블리에 포함시키는 관행과 관련한 비용 절감으로 인해, 비휘발성 메모리의 구매자는 메모리 제조자로 하여금 높은 출하 수율을 제공하도록 요구한다. 따라서, 메모리 테스트 프로세스는 한번의 테스트 프로세스에서 규격에 부합하지 않는(non-conforming) 부품의 대부분, 바람직하게는 규격에 부

합하지 않는 모든 부품을 식별하기 위해 상당히 효율적이어야 한다.

<11> 비휘발성 메모리가 점점 더 커지고, 조밀해지고, 복잡해짐에 따라, 테스터는 메모리를 테스트하는데 걸리는 시간을 그다지 증가시키지 않으면서, 증가된 크기와 복잡성에 대처할 수 있어야 한다. 메모리 테스터는 종종 연속적으로 동작한다. 그래서, 테스트 시간은 최종 부품의 비용에 있어서의 주요한 요인으로 여겨진다. 메모리가 발전하고 개선됨에 따라, 테스터는 그 장치에 가해진 변화를 쉽게 수용할 수 있어야 한다. 비휘발성 메모리를 테스트함에 있어서의 특유한 또다른 문제는 그 메모리 셀에의 반복적인 기록이 부품의 전체 수명 성능(lifetime performance)을 악화시킬 수 있다는 것이다. 비휘발성 메모리 제조자는 메모리 장치내에 특수한 테스트 모드를 형성함으로써 많은 테스트 관련 문제에 대해 대응해 왔다. 이러한 테스트 모드는 메모리 구매자에 의해 사용되는 것이 아니라, 제조자에 의해 이용되어, 가능한 한 적은 시간에 가능한 한 효율적으로 메모리 전체 또는 메모리 중 상당 부분을 테스트할 수 있다. 비휘발성 메모리 중 일부는 또한 테스트 프로세스 동안 복구(repair)될 수 있다. 그러므로, 테스터는 복구할 필요, 복구할 위치, 필요한 복구의 유형을 식별할 수 있어야 하며, 그 후, 적합한 복구를 수행할 수 있어야 한다. 그러한 복구 프로세스는 메모리의 규격에 부합하지 않는 특정 부분을 검출하고 분리할 수 있는 테스터를 필요로 한다. 복구 기능뿐만 아니라 특수한 테스트 모드를 충분히 이용하기 위해서, 테스터가 장치로부터 예상되는 응답에 근거하여 조건적인 분기(conditional branching)를 지원하는 테스트 프로그램을 실행할 수 있는 것이 유익하다.

<12> 개념적으로 예상해보면, 메모리 테스트 프로세스는 알고리즘적 프로세스이다. 한 예로서, 전형적인 테스트는 0과 1을 메모리 셀에 기록하는 동안 메모리 어드레스를 순차적으로 증가시키거나 감소시키는 것을 포함한다. 통상적으로, 메모리 사이클 동안 기록되거나 판독되는 1과 0의 집합을 "벡터"라고 하며, "패턴"이라는 용어는 벡터의 시퀀스를 말한다. 통상적으로, 테스트는 체커보드(checkerboard), 워킹 1(walking 1's) 및 나비 패턴(butterfly pattern)과 같은 메모리 공간내에 기록 패턴을 포함한다. 테스트 개발자는 알고리즘적 구조의 도움으로 이러한 패턴을 만드는 프로그램을 보다 쉽고 효율적으로 생성할 수 있다. 알고리즘적으로 일관적인 테스트 패턴은 디버그하기 쉬울뿐만 아니라, 논리적 방법의 사용을 촉진하여, 예상 대로 수행하지 못하는 일부 패턴을 분리시킨다. 프로그래밍 루프(programming loop)내에서 반복되는 인스트럭션 및 커맨드를 사용하여 알고리즘적으로 생성된 테스트 패턴은 테스터 메모리내에서 보다 작은 공간을 차지한다. 따라서, 메모리 테스터내에 알고리즘적 테스트 패턴 생성 능력을 갖는 것이 바람직하다.

<13> 정확한 신호 에지 배치 및 검출 또한 비휘발성 메모리 테스터의 효율에 있어서 중요하다. 특정된 가장자리(margin)내에서는 부합하지 않으나, 중앙(median)에서는 일반적으로 부합하는 부품을 포착하기 위해서, 비휘발성 메모리 테스터는 시간상으로 다른 신호 에지에 대해 각 신호 에지를 정확히 배치할 수 있어야 한다. 또한, 어느 시점에서 신호 에지가 수신되는지 정확히 측정하는 것도 중요하다. 따라서, 비휘발성 메모리 테스터는 테스트 중인 장치(Device Under Test: DUT)(메모리)로부터의 자극과 응답의 타이밍 및 배치에 대한 충분한 유연성 및 제어를 가져야 한다.

<14> 메모리 테스터는 DUT에 인가된 "전송" 벡터(자극) 및 답변으로 기대되는 "수신" 벡터(응답)를 생성한다고 여겨진다. 이러한 벡터들을 생성하는 알고리즘적 로직은 메모리 테스터가 신호를 핀으로 보내며 핀으로부터 받는 맵핑 구성을 포함하고 있기 때문에, 어떻게 벡터내의 특정 비트가 DUT내의 특정 신호 패드에 도달하는지 또는 특정 신호 패드로부터 도달되는지에 대한 별 어려움 없이 일반적으로 그렇게 할 수 있다.

<15> 메모리 테스터는 테스트 프로세스를 촉진하는 데 사용되는 내부 테스트 메모리를 가진다. 이 내부 테스트 메모리는 몇가지 목적으로 사용될 수 있는데, 그 중에서 실시간으로 전송 벡터를 생성하는 것에 반해 시간에 앞서 전송 벡터를 저장하는 것과, 수신 벡터를 저장하는 것과, 다양한 에러 표시 및 테스트 동안 얻어지는 DUT 동작에 관한 다른 정보를 저장하는 것이 있다. (또한, SRAM을 사용하며 "내부 메모리"라는 용어의 범위내에 해당할 수 있는 메모리 테스터의 동작에 대한 내부의 관리 목적이 있다. 이러한 목적은 테스터의 내부 동작에 대해 비공개적이며 알고리즘적 레벨에서는 볼 수 없는 경향이 있고, 내부 제어 레지스터에 필적할 만한 것이다. 그 메모리는 "내부 제어 메모리"로 설명되며, 여기서 "내부 테스트 메모리"라는 용어가 의미하는 것으로부터 제외되는데, DUT의 자극 및 DUT로부터의 응답과 직접 관련된 비트 패턴을 저장하는데 사용되는 메모리를 설명하기 위해 이 용어를 사용한다.) 이 내부 테스트 메모리는 테스트가 수행되는 것만큼 적어도 빠르게 동작할 필요가 있다는 것은 이해하기 쉽다. 내부 테스트 메모리(또는 그 일부분)가 DUT에 인가되는 것과 동일한 어드레스(또는 그 파생물)에 의해 어드레스되는 것은 매우 일반적인 패러다임이다. 그렇다면 내부 테스트 메모리내의 그 어드레스된 위치에 저장되는 것은 DUT상에서 수행되는 테스트 동작 동안 그 어드레스에서의 DUT 동작을 나타내는 무언가(something)이다. 테스트 프로그램내에서의 알고리즘적 고려는 연속적인 전송 벡터와 연관된 어드레스의 시퀀스가 램덤할 수 있다는 것을 의미할 수 있다. 따라서, 내부 메모리는 고속 및 랜덤 어드레스성(random

addressability)이라는 이중의 특성을 가질 필요가 있다. SRAM은 빠르고, 제어하기 쉽고, 전체적인 랜덤 어드레싱을 용인할 수 있기 때문에 최근에 선호되고 있다. 정말로, 종래의 메모리 테스터는 SRAM을 그 내부 테스트 메모리로서 사용해왔다.

<16> 불행하게도, SRAM은 값이 매우 비싸며, 이것은 메모리 테스터와 함께 기능해야 하는 내부 테스트 메모리의 양을 제한한다. 그로 인해, 메모리 부족으로 인한 메모리 테스터 기능성(functionality)이 제한된다. DRAM은 상당히 덜 비싸지만, 랜덤 어드레싱을 용인할 수 없으며, 여전히 고속으로 수행할 수 없다.

<17> DRAM은 메모리 테스터내의 내부 테스트 메모리로서 SRAM을 대체할 수 있다. 후술하는 개요에서 간단히 설명되는 바와 같이, 내부 테스트 메모리로서 사용하기 위해 DRAM 동작의 속도를 증가시키는 문제는 그 속도를 증가시키는 대신에, 사용되는 DRAM의 양을 증가시킴으로써 해결될 수 있다. DRAM의 동일한 뱅크(Bank)의 수는 그룹(Group)으로서 취급된다. 그룹내의 메모리의 서로 다른 뱅크에 대해 신호를 인터리빙하는 것과 뱅크의 그룹들 사이에서 멀티플렉싱하는 것이 조합되면, 임의의 하나의 뱅크에 대한 메모리 트래픽이 뱅크에 의해 처리될 수 있는 속도로 감소된다. (독자의 편의를 위해서, 이 기술의 구조적인 면과 관련된 용어의 상당 부분이 후술하는 발명의 청구 대상의 설명에 유용하기 때문에, 우리는 이 기술의 매우 간결한 설명을 첨부한다.)

<18> 그룹에의 신호 트래픽에 대한 가요성 4 폴드 인터리빙 방식(flexible four-fold interleaving scheme)과 조합된 4개의 뱅크 각각의 3개의 그룹 사이의 3 방법 멀티플렉싱은 단지 3개의 메모리 버스만을 필요로 하면서, 동작 속도를 약 12배 증가시킨다. 멀티플렉서에 대한 다음 그룹(next Group)을 선택하기 위한 연속 전략(round robin strategy)은 간단하며, 각 그룹에 대한 인터리빙 메카니즘이 그의 가장 최근에 할당된 작업을 완성하는데 필요한 시간을 갖도록 한다. 그룹내의 모든 인터리빙된 액세스는 (그 그룹내의) 다음 뱅크 상에서 수행되며, 또한, 간단한 연속 선택(round robin selection)에 의해 선택된다. 이러한 구성에서, 12개의 뱅크의 각각은 전적으로 이용가능한 어드레스 공간의 복제된 인스턴스(duplicate instance)를 나타내고, 임의의 개개의 판독 사이클은 결국 12개의 뱅크 중 어느 하나를 액세스하게 될 수 있다. 테스트의 완료시, 임의의 어드레스 또는 관심의 대상이 되는 어드레스의 집합의 히스토리가 모든 12개의 뱅크에 걸쳐 퍼져있을 것이기 때문에, DUT의 테스트 동안 어떠한 결함(failures)이 발생했는지 알 수 있도록 모든 12개의 뱅크를 조사해야 한다는 것이 암시되어 있다. 따라서, 특정 채널은 12개의 비트(각 뱅크로부터의 하나의 비트이며, 그 뱅크에 대한 워드내의 비트 위치는 채널에 의해 결정됨)로 표현된다.

<19> 그러나, 결함 정보를 발견하기 위해 모든 12개의 뱅크를 (수동으로 소위) 개별적으로 조사해야 하는 것은 서투른 일일 것이다. 그래서, 어드레스에서의 판독 사이클 동안 모든 12개의 뱅크의 결과를 하나 또는 모든 12개의 뱅크에 저장될 수 있는 단일화된 결과로 자동적으로 "구성"(compose)(통합)하기 위한 실용 메카니즘이 제공되었다. 이는 구성된 데이터가 나중에 최고 속도로 판독되도록 한다. 한 실시예에서 최고 속도는 랜덤하게 어드레스된 메모리 처리에 대해 100MHZ 속도이다.

<20> 33MHZ가 충분히 빠르다면, 랜덤 액세스는 멀티플렉싱 없이 단지 인터리빙만으로 지원될 수 있으며, 그러한 경우, 구성 메카니즘 및 메모리 어드레싱 방식이 적절하게 조정된다. 어드레싱 방식은 메모리의 깊이를 랜덤한 100MHZ 동작에 대해서보다 3배 더 깊게 하는 여분의 그룹 선택 비트를 포함하도록 변화된다. 이러한 동작의 두 개 모드를 각각 R100 및 R33이라고 부른다. 또한 DRAM에 전송될 적절하게 작용하는 어드레스에 의존하는 단일 뱅크에의 100MHZ 동작의 L100 모드가 있다(열 어드레스(row address)의 절대적 최소치가 변화됨).

<21> 내부 테스트 메모리 조직의 최고 레벨에는, 4개의 메모리 세트가 있는데, 그 각각은 그 자신의 개별적이며 독립적인 어드레스 공간을 가지고 있고, 요구되는 메모리 처리를 수행한다. 두 개는 전술한 바와 같은 DRAM이고, 두 개는 SRAM이다. 각 메모리 세트는 그 자신의 컨트롤러를 가지고 있으며, 그 컨트롤러에 의해 메모리 처리가 영향을 받는다. 외부에서 볼 수 있는 동작적인 능력에 대해, 모든 4개의 메모리 세트는 본질적으로 동일하다. 그것들은 단지 메모리 공간의 크기 및 그것들이 어떻게 내부적으로 구현되는지에 있어서만 상이하다. SRAM 메모리 세트는 시작하기에 충분히 빠르기 때문에 멀티플렉싱 및 인터리빙을 이용하지 않는다. 그것들의 독립성에도 불구하고, 동일한 유형(SRAM 또는 DRAM)의 메모리 세트는 "스택"될 수 있는데, 그것은 처리된 하나의 큰 어드레스 공간을 말하는 것이다. 이것은 어드레스의 알고리즘적 생성과 메모리 처리를 실제로 어느 메모리 세트로 전송할 지에 대한 결정에 있어서, 메모리 세트 그 자체를 넘어선 제어 레벨에서 행하여진다. 그것은 메모리 세트 및 그 컨트롤러가 동작 모드 R100과 R33 사이에서와 같이 어드레스 공간을 3배로 하기 위해 그룹들을 스택할 수 있는 방법만큼이나 자동적이지 않다. 각각의 메모리 세트 컨트롤러에 있어서, 또다른 컨트롤러를 갖는 또다른 메모리 세트와 같은 것이 있다는 것은 근거가 없다.

<22> 따라서, 테스터의 내부 테스트 메모리는 4개의 메모리 세트, 그 중 두 개는 "내부" SRAM이고 그 중 두 개는 "외

부" DRAM으로 분할된다. 확실히 이러한 메모리 모두는 물리적으로 메모리 테스터의 내부에 있다. "내부" 및 "외부"라는 용어는 집적의 레벨과 관계가 깊다. SRAM은 테스터의 중앙 기능적 회로과 연관된 VLSI(Very Large Scale Integration)의 필수적 부품이고, DRAM은 VLSI 재료에 인접하여 탑재되며 개별적으로 패키징되는 부품이다. SRAM의 양은 상당히 적으며(즉,약 메모리 세트 당 1 메가비트), DRAM의 양은 충분하고 선택가능하다(즉, 메모리 세트 당 128 내지 1024 메가비트의 범위에 있음). SRAM 메모리 세트는 상시 존재하며, ROM(Read Only Memory)과 같이 DUT의 기대 내용(expected content)을 저장하는 것 등의 임의의 적합한 목적에 사용될 수 있다. DRAM 메모리 세트는 실제로 선택적이며, 다른 용도가 있기는 하지만, 통상적으로 복구를 하게 하는 후속 분석에 대한 트레이스(trace)를 만드는데 사용된다. 테스터는 SRAM과 DRAM 메모리 세트가 사용될 수 있는 서로 다른 목적에 대해, SRAM과 DRAM 메모리 세트 사이의 구별을 하지 않는다. 그러한 구별은 대부분 크기의 문제를 일으킨다. SRAM 메모리 세트는 작고, DRAM 메모리 세트는 잠재적으로 크다. 테스트 프로그램을 만드는 사람(또는 사람들)은 다양한 메모리 세트들을 어떻게 사용해야 할지에 대해 결정한다.

<23> 우리는 앞서 알고리즘적 능력이 메모리 테스터에 실행될 테스트 프로그램을 개발하고 유지함에 있어서 바람직하다고 언급했었다. 프로그램가능성이 루프내의 네스트(nest)된 루프, 테스트 결과에 관한 분기(branching)를 허용하는 복잡성 이외에, 소정의 고유한 특성을 갖는 비휘발성 메모리의 자동화 테스트뿐만 아니라, 일반적인 비휘발성 메모리의 자동화 테스트를 용이하게 하기 위해 다양한 특유의 동작적인 능력이 메모리 테스터내에 설계된다. 이 모든 것은 테스트 프로그램에 대한 개발 및 유지 활동에 부여된 대응하는 복잡성이 있을 수 있다는 것을 의미한다. 즉, 테스트 프로그램이 그 자체로 복잡한 메카니즘인 것이 일반적이다.

<24> 그렇다면, 테스트 프로그램을 개발하는 사람의 상황을 고려해보자. 최초의 버전이 기대한 대로 수행하지 못하는 것은 특이한 일이 아닐 것이다. 그 버전은 실행하지 않기로 한 것을 실행하고, 실행하기로 한 것을 실행하지 않는다. 코드를 연구하는 것은 별로 도움이 되지 않을 때가 있다(현재의 이해가 허용하는 만큼 올바른지를 생각하라). 그리고 진행 중인 것을 발견하기 위한 가장 적합한 도구가 전압 파형의 오실로그래프 트레이스이다. 추가적인 데이터의 소스는 발생 중인 것을 이해하는 프로세스를 촉진하기 위해 또는 이론 등을 증명하기 위해 필요하다. 불행하게도, 테스트 헤드에 단지 접근하여 당신이 마음에 드는 오실로스코프에 대해 하나 또는 두 개의 탐침(probe)을 부착하는 것은 실제적인 문제가 아니다. 탐침이 부착될 실제 물리적인 위치를 식별할 수 있다고 해도(이론적으로는 가능하지만, 실제적으로는 매우 어려움), 일반적으로 탐침을 부착할 충분한 공간도 없고, 탐침 및 그 케이블을 지지할 만큼 충분히 강한 물리적인 구조도 없다. 또한, 전기적 부하 또는 원하는 측정에 대한 다른 방해의 문제가 있다. 그리고, 그러한 모든 문제가 극복되었다고 해도, 선택된 데이터가 관심의 대상이 되는 문제와 관련되도록 어떻게 그 스코프를 트리거할 지에 관한 문제가 여전히 남아있다.

<25> 스코프 탐침을 접속하는 어려움에 대해, 우리가 좀 전에 얘기했던 것을 주장할 것이다. 스코프가 노출되는 데이터의 문제 및 무엇이 포착되었는가에 대해, 그 문제가 정확하게 이해될 수 있기 전에 이해해야 하는 배경이 매우 많다. 여기서 유용한 출발점은 에질런트 테크놀러지사(Agilent Technologies)로부터의 83000 VLSI 테스트 시리즈에 발견될 수 있는 것과 같은 "벡터" 머신이라는 용어의 메모리 테스터의 한 유형이다.

<26> 벡터 구동 테스터는 DUT에 차례로 인가할 자극 벡터의 명확한 리스트를 가지고 있다. 그것은 단지 자극 벡터를 인가하고 어떠한 수신 벡터가 발생했는지를 기록한다. 그것은 시스템의 매우 즉각적인 자극/즉각적인 응답의 유형이다. 즉, 이것을 집어넣고, 저것을 꺼내는 것이다. 벡터 머신 및 그 DUT의 본질은 리스트의 실행이 임의의 시간에 방해받을 수 있으며, 전송 벡터의 리스트내의 소정의 사전에 선택된 편리한 위치로부터 재시작된다는 것이다. 후술하는 알고리즘적으로 구동되는 패턴 지향 메모리 테스터와 유사하게, 벡터 테스터는 (수신 벡터의 생성을 위해) 소정의 클럭 신호에 관한 가변 샘플링 임계치 및 가변 샘플 오프셋을 가지고 있다. 존재하지 않는 것은 관련 채널의 아날로그 전압 파형을 디지털화할 아날로그-디지털 컨버터이다. 그럼에도 불구하고, 존재하는 자원은 선택된 벡터 중에 관심있는 채널에 관한 전압 파형을 재현하는 데 사용될 수 있다. 그것은 부담이 되는 일이나, 해야 한다. (어떠한 실용적인 대안도 없다). 여기에 방법이 있다.

<27> 정상적인(비스코프 모드) 동작 동안 임계치 및 샘플 타이밍 오프셋의 특정의 한 쌍이 선택되고, 데이터가 순차적으로 기록된다. 임계치는 VOH(Voltage Out High) 및 VOL(Voltage Out Low)이며, 약 10mV의 단계에서 조정가능하다. 그 채널의 샘플 타이밍 오프셋에 의해 특정된 시간에 채널 전압이 VOH를 초과할 경우, VOH 임계치에 도달되며, 채널 전압이 VOL보다 적을 경우 VOL 임계치에 도달된다. 이러한 조건을 표시하는 신호의 이름은 각각 YVOH 및 YVOL이다. 수신 벡터내의 연관된 채널에 대해 기록되는 것은 채널 전압이 샘플 타이밍 오프셋에 표시되는 순간에 각각의 임계치에 도달했는지 여부이다.

<28> 스코프 모드에 있어서, 기본적인 생각은 파형을 따라 알맞게 간격이 있는 시점에서, 그 시점에서 파형의 어느

한 측에 어떠한 임계치가 있는지를 찾는 것이다. 이를 위해서, 관심의 대상이 되는 이벤트를 포함하는 벡터의 시퀀스가 반복적으로 실행돼야 한다. 그러한 반복적인 실행 동안 VOH 전압 임계치는 반복의 인스턴스들에 따라 조금씩 변화된다. 그리고, 시퀀스가 계속적으로 반복됨에 따라 점점 더 많은 데이터가 저장된다. 이는 VOH 임계치에 대한 양 극단(extreme)이 사용될 때까지 계속될 수 있다. 시퀀스내의 각 벡터에 있어서, 기록된 수신 결과는 오실로그래픽 트레이스에 기여하도록 해석될 수 있다. (YVOH의 연속적인 인스턴스가 값이 [TRUE]로 동일하면, 소정의 인스턴스에서 YVOH의 새로운 값 [FALSE]로 변화되고, 그 후, 연속적인 값은 그 값이 이전에 동일했을 때 가졌던 값과 반대되는 값 [둘 다 FALSE]로 다시 동일하게 되고, 그 후, 반대로 전환된 상태로 남아있는다. 이는 값이 TRUE에서 FALSE로 변화되었을 때의 시점에서 채널에 대한 안정한 전압 값을 나타내며, 그 시점은 VOH 값의 경과에 사용되는 (불변하는) 샘플 타이밍 오프셋에 의해 식별된다. 안정한 전압 값은 YVOH가 값을 변화시킨 VOH의 그 연속적인 인스턴스들의 사이의 소정의 지점에 있다. 이 잘 이해되는 종래 기술에는 다른 경우가 있지만, 간결함을 위해 언급하지 않을 것이다.) 각 벡터에 있어서, 단지 그 샘플 타이밍 오프셋에 대해 전술한 기여가 있을 것이다.

<29> 하나의 극단에서부터 다른 극단으로 VOH를 단지 증가시키는 것은 단지 서로 알맞게 가깝고, 관련 YVOH가 반대 값을 갖는 한 쌍의 VOH 값을 발견할 수 있는 방법이 아니다라는 것이 이해될 것이다. 예컨대, 3V 부분에 대해 5V 범위에 걸쳐 스위핑하는 시점은 거의 없을 수 있다. 다음에, 출발점에 있어서, 약간의 노이즈 신호에 대해서도 전이가 발견되었을 때 혹은 더 많은 증가분이 그 전이가 정말로 그 근처에 있었다는 것을 확인한 후에 증가하는 것을 멈추는 것으로 충분할 것이다.(노이즈 전이를 통해 진행하는 샘플은 평균으로부터 이득을 줄 수 있다.) 결국, 예측을 포함하여 이원(binary) 검색 전략 혹은 다른 검색 전략을 사용하여, YVOH에서의 원하는 전이와 연관된 적절한 VOH 값을 발견하는데 필요한 반복의 수를 최소화하려고 할 수 있다. 이것은 사용자에게 관심의 대상이 되는데, 그 이유는 그러한 전략이 찾아진 파형을 생성하는 데 필요한 시간의 양을 상당히 줄일 수 있기 때문이다. 그렇다고 하더라도 그것은 전체 동작을 완성하기 위해서 여전히 몇분(minutes)을 필요로 할 것이다.

<30> 다음 단계는 다른 샘플 타이밍 오프셋으로 전체 동작을 반복하는 것이다. 결국 우리는 관심의 대상이 되는 벡터 동안 채널의 전압 동작 상태에 관한 완벽한 설명을 해낼 것이며, 임의의 디지털 오실로스코프에서와 같이, 그것을 전압 파형으로 표시할 수 있다. 수직(전압 축) 해상도(resolution)는 VOH 변화에 대한 눈금 크기일 것이며, 수평(시간 축) 해상도는 샘플 타이밍 오프셋 변화에 대한 눈금 크기(20 피코초(picoseconds)만큼 작을 수 있음)일 것이다. 이는 벡터 머신 유형인 테스터에 사용되었던 종래 기술이다. 우리는 우리의 알고리즘적으로 제어되는 패턴 머신에 그것을 사용하고 싶으나, 몇가지 심각한 문제가 있다.

<31> 벡터 머신은 단지 그 명시적인 리스트내의 벡터 중 하나를 트리거할 시점이라고 확인함으로써 그 트리거 기능을 쉽게 수행하는 반면에, 알고리즘적으로 제어되는 머신에서 "스코프를 트리거하는 것"(데이터를 저장하기 위해 특정 수신 벡터를 식별함)은 매우 중요하다는 것을 이해해야 한다. 디지털 환경에서는 일반적으로, 기울기를 갖는 간단한 아날로그 트리거 레벨은 거의 확실하게 쓸모가 없다. (동시) 신호 값의 소정의 조합을 검출하는 것(패러렐 워드)도, 그 자체로 종종 적합하지 않다. 그리고, 우리는 일반적으로 테스트 알고리즘을 수행하기 위해 실행되고 있는 코드로 접근해서, "여기 우리가 이 인스트럭션 워드를 실행할 때 트리거 하시오"라고 말할 수 없다. 그것이 작동할 때가 있을 수 있으나, 코드내에 그 실행이 결함과 특유하게 연관된 인스트럭션이 있을 것이 필요하기 때문이다. 알고리즘적 가능성은 종종 너무 복잡하여, "고장남(at fault)"이라는 인스트럭션이 루프내에 나타날 수 있으며, 그 인스트럭션은 에러가 발생하기 전에 수천 또는 수백만번 완전하게 작용한다.

<32> 우리는 또한 정확히 동일한 일련의 이벤트가 오실로그래픽 트레이스로서 파형이 재구성되기 위한 충분한 포착된 데이터가 있기 전에 여러번(대략 타이밍 오프셋에 대한 서로 다른 값의 수로 VOH의 서로 다른 값의 수를 생성할 수 있을 정도) 발생해야 한다는 것을 명심해야 한다. 이제 테스트 프로그램이 에러 플래그(에러 플래그는 DUT의 응답이 일정한 기대에 부합하지 않았다는 것을 의미함)에 관한 분기를 포함한다고 가정하자. 에러 플래그는 테스트 프로그램 흐름을 변경한다. 그러나 스코프 모드에 대해 반복하는 동안 에러 프래그는 그렇지 않을 때보다 더 빈번히 발생할 것이며, 트레이스에 기여할 모든 벡터에서 더 빈번히 발생할 것이다. 왜 그런 것일까? 임계치 및 샘플 타이밍 오프셋의 스위핑 때문이다. 그 스위핑은 DUT가 기능 장애에 있을 때에도 에러 플래그를 설정하고 프로그램 흐름을 변경하는 "잘못된 대답"을 생성할 것이다. 변경된 프로그램 흐름은 일정한 대응하는 방법으로 파형을 변경할 것이 거의 확실하다. 그러나, 파형 활동의 반복되는 동일한 인스턴스로부터 트레이스를 재구성하는 데 필요한 활동은 파형을 (프로그램 흐름을 변경하는 결과로서) 변경하고, 그 후, 안정된 파형은 존재하지 않을 것이며, 어떠한 이해가능한 트레이스도 재구성될 수 없다. 그러나 테스트 프로그램내의 에러 플래그에 관한 분기는 강력한 디바이스이며, 우리가 마지못해 포기하는 것이다. 즉, 테스트 프로그램이 스코프

모드로 사용가능한 일정한 제한된 방식으로 기록될 필요가 없을 것이다. 더욱이, 우리는 메모리 테스터의 사용자가 안정적으로 반복하기 위해 테스트 프로그램을 변경하지 않고 이전부터 존재하는 임의의 테스트 프로그램에 스코프 모드를 적용할 수 있기 바란다. 그리고 우리는 트리커 인스트럭션을 촉진하거나 포함하기 위해서 테스트 프로그램의 소스 코드 변경을 수행하기를 바라지 않는다. 대신에, 임계치 및 샘플 타이밍 오프셋을 스위핑하는 동안 얻어진 포착된 데이터로부터 트레이스를 재구성하는 것 이외에, 메모리 테스터가 우리 대신에 이러한 골치 아픈 문제를 감당해주기를 바란다.

<33> 아울러, 트리거 특수화 메카니즘(trigger specification mechanism)이 제공되면, 우리는 관련 에러 플래그의 실제 값에도 불구하고, 에러 플래그에 근거한 내부 테스트 프로그램 분기가 특정 경로를 따라가도록 하기 위한 결과 트리거를 사용할 수 있기를 바란다. 예컨대, 테스트 프로그램은 DUT내의 특정 유형의 결함에 대응하는 많은 분기를 포함할 수 있다. 프로그래머는 이러한 다양한 분기가 모두 액세스가능하고, 그것들이 의도되는 대로 수행한다는 것을 증명하고 싶을 것이다. 문제는 코드를 편집해야 하지 않고, 또한 테스트 프로그램의 원하는 실행을 하게 하는 올바른 기능 장애(malfunction)를 갖는 이미 알려진 특수한 DUT의 집합을 갖지 않고, 다양한 분기를 어떻게 발생시키는냐 하는 것이다.

발명이 이루고자 하는 기술적 과제

<34> 실질적인 알고리즘적 내용을 갖는 테스트 프로그램의 실행에 사용되는 알고리즘적 패턴 제너레이터를 갖는 메모리 테스터의 스코프 모드에 트리거 신호를 제공하는 문제에 대한 해결책은 알고리즘적 동작을 촉진하기 위해 제공되는 기존의 일정한 보조 레지스터의 내용을 포함시키는 것뿐만 아니라, 알고리즘적 패턴 제너레이터에 어드레스 및 데이터와 같은 DUT를 동작시키는 데 사용되는 기존의 하드웨어 양에 의해 표현되는 트리거 특수화의 발생을 검출하기 위한 하드웨어를 갖추는 것이다. 이는 테스트 프로그램의 어느 부분이 실행되는지에 따라서 더 자격이 부여된 원래의 하드웨어(raw hardware)(브레이크포인트) 트리거를 형성한다.(테스트 프로그램을 만드는 데 현재 사용되는 언어는 전술한 레지스터 중 하나가 아닌 임의의 메모리 위치내의 값으로서만 존재하는 추상적인 변수들의 생성을 허용하지 않는다. 따라서, 알고리즘적 패턴 제너레이터내의 특정 하드웨어 레지스터상에서 수행되는 하드웨어 마스킹 및 AND 연산은 트리거 생성을 위한 적합한 출발점이다.) 자격이 부여된 브레이크포인트 트리거는 스코프 모드를 트리거하고 (테스트 프로그램으로 특정 경로를 강제하기 위해) 에러 플래그를 선택된 값으로 하는 데 사용될 수 있는 시스템 트리거 신호로 되기 전에, 0 또는 그 이상의 DUT 사이클만큼 지연될 수 있다.

<35> 테스트 프로그램 그 자체는 전술한 바와 같이 허위 코드(pseudo code)와 유사한 언어로 그러나 제한된 일련의 변수를 가지고 기록된다. 컴파일러는 이 소스 코드상에서 동작하여, 알고리즘적 패턴 제너레이터내의 하드웨어의 잔여 부분내의 조건을 제어하고 그 조건에 응답하는 실행가능한 인스트럭션 워드를 생성한다. 스코프 모드 프로세스의 사용자 인터페이스의 태양은 개별 환경에서 실행하는 것이 바람직하며, 사용자는 그 프로세스와 상호작용하여 트리거 특수화를 규정한다. 스코프 모드 프로세스는 하드웨어 레지스터 값의 레벨에서 원래의 트리거 조건을 인식하는 마스크 및 비교 메카니즘을 설정한다. 그것은 또한 컴파일러에 테스트 프로그램의 어느 부분이 원래의 트리거 특수화를 가능하게 하는 것으로 간주되는지에 대해 알려준다. 이는 알고리즘적 프로그램 제너레이터에 대한 실행가능한 인스트럭션 워드에서 "트리거링 기능이 현재 가능하다"는 것을 의미하는 비트를 제공함으로써 컴파일된 코드에서 달성된다. 테스트 하드웨어 및 진정 로직 분석기 스타일 순차적 트리거와 밀접하게 관련되지 않는 추상적인 변수에 대한 값을 모니터링하는 것은 어떤 점에서 바람직하지만, 필요하지 않다.

<36> 전압 임계치 및 샘플 타이밍 오프셋의 스위핑에 대한 안정된 파형을 제공하는 문제에 대한 해결책은 메모리 테스터에 대해, 일정한 적당한 수, 즉, 128까지 트리거의 발생 후에 테스트 프로그램을 통해 초기 패스 동안 방출되는 전송 벡터의 시퀀스를 "기억하는 것"(즉, 기록하는 것)이다. 그 시퀀스를 "타겟 시퀀스"라고 부를 것이다. 타겟 시퀀스가 메모리내에 저장되면("기억되면"), 스코프 모드는 전체 테스트 프로그램을 재시작하고 (일정한 초기 조건이 설정될 필요가 있을 것임), 정확히 트리거로 다운되기 전과 같이 작동하게 함으로써, 원하는 정보를 추출할 수 있다. 그러나, 트리거가 발생할 때에는 전송 벡터가 소위 진행중인 알고리즘으로부터보다는 오히려 기억된 타겟 시퀀스로부터 방출되고, VOH와 샘플 타이밍 오프셋의 조합이 제자리로 전환된다. 그 조합은 그 값의 획득 스위프(acquisition sweep)를 따라가는 하나의 단계를 구성하며, 기억된 타겟 시퀀스의 끝에 도달할 때까지 유지된다.("획득 스위프"는 오실로그랙픽 표현의 "수평 스위프"와 혼동되지 않을 것이다.") 대응하는 수신 벡터의 시퀀스는 그들이 도착할 때 자동적으로 저장된다. 타겟 시퀀스의 끝에서 테스트 프로그램

은 다시 재시작된다.(그러나 정상적인 임계치 및 샘플 타이밍 오프셋을 가진다.) 당연한 과정으로 결국 또다른 트리거가 존재하는데, 그 위에서 기억된 타겟 시퀀스가 진행 중인 알고리즘 대신에 다시 사용되고, 타겟 시퀀스의 끝에 다시 도달될 때까지 획득 스위프를 따르는 단계에서의 다음번 조합이 마련된다. 이전과 같이, 획득 스위프에서의 그 단계와 관련된 수신 벡터가 자동적으로 저장된다. 이 프로세스는 전체 획득 스위프가 수행 완료될 때까지 계속되며, 그 때에 타겟 시퀀스 동안 관심의 대상이 되는 신호에 대한 파형이 자동적으로 저장된 수신 벡터의 대응하는 집합의 적합한 조사에 의해 생성될 수 있다.

<37> 전술한 기술은 메모리 테스터에 타겟 시퀀스를 기억하기 위한 FIFO(또는 다른 메모리 구조) 및 기억한 후 트리거가 발생할 때 기억된 타겟 시퀀스를 실행하도록 전환하는 것을 감독하는 제어 메카니즘을 갖춤으로써 촉진된다. 실행가능한 프로그램 언어는 프로그램을 통하여 표시되는 데 필요한 어드레스보다 훨씬 넓기 때문에(208 비트 넓이), 인스트럭션 워드 그 자체라기보다는 오히려 FIFO내에 저장되는 타겟 시퀀스내의 인스트럭션 워드에 대한 어드레스의 시퀀스이다. 타겟 시퀀스에 대한 어드레가 FIFO내에 저장된 후, 획득 스위프가 개시되기 전, 그 어드레스들은 인스트럭션 워드의 대응하는 실제 시퀀스를 인출(fetch)하여 저장하는 데 사용된다. 복제된 타겟 시퀀스내의 임의의 분기 구조물은 시퀀스내의 다음 워드를 지시하도록 변경된다.(그 워드는 여하튼 타겟 시퀀스의 초기 포착 동안 발생한 워드일 것이다.) 이 실행가능한 타겟 시퀀스는 컴파일된 테스트 프로그램인 인스트럭션 워드를 포함하는 프로그램 SRAM의 예비 부분내에 저장된다. 연속하는 시스템 트리거가 발생할 때, 테스트 프로그램 실행은 임시적으로 중지되어, 획득 스위프내의 다음 단계에 대한 VOH 및 샘플 타이밍 오프셋 값을 마련하고, 프로그램 SRAM을 어드레스하는 프로그램 카운터는 예비 부분의 개시로 설정된다. 그리하여, 획득 스위프내의 다음 단계가 설정되면, 테스트 프로그램 실행은 초기에 포착된 타겟 시퀀스를 반복하여 재개될 것이다. 획득 스위프내의 각 단계의 결과(수신 벡터)가 ECR(Error Catch RAM)내에 임시적으로 저장(즉 캐쉬(cache))된다. ECR내의 그러한 수신 벡터의 저장은 정상적인 동작이지만, 일반적으로 관심의 대상이 되는 소정의 에러상에서 조절된다. 여기서 우리는 에러에 상관없이 그것이 저장되기를 바라고, 이를 달성하기 위해 간단한 메카니즘이 제공된다.

<38> 더욱이, 전용 레지스터는 만족된 트리거 조건이 검출될 때, "정상"에서 "스위핑된 것"으로 임계치 및 샘플 타이밍을 급속하게 변화시키는 것을 지원한다. 이 기술은 타겟 시퀀스상의 획득 스위프의 각 단계 동안 자동적으로 저장된 ECR로부터 캐쉬된 수신 벡터를 내려놓는 데 사용되는 메모리의 전용 스태쉬(private stash)를 갖기 위해 테스트 프로그램의 실행을 감독하는 (및 스코프 모드를 관리하는) 운영체제를 정비함으로써 더욱 촉진된다. 스코프 모드가 관심의 대상이 되는 신호에 대한 파형을 축적하고 결국에는 생성하고 표시하기 위해 사용하는 것은 메모리의 스태쉬이다.

<39> 테스트 프로그램 검증의 문제는 실행하는 테스트 프로그램의 환경으로 선택된 에러 플래그에 대한 선택된 값을 채워넣기(jam) 위해 (아마도 동작의 스코프 모드가 없어서) 스코프 모드의 트리거를 사용함으로써 도움을 받는다. 이는 실제로는 발생하지 않았더라도, 연관된 에러의 유형의 출현을 강제한다. 확실히, 동일한 에러 플래그는 테스트 프로그램내의 서로 다른 위치에서 서로 다른 사항을 의미하기 쉽다. 프로그램의 특정 부분에 도달할 때까지 에러 플래그의 발생을 지연시키는 것은 트리거가 사용되는 방법이다. 트리거가 단지 어떤 에러 플래그를 "합성하느냐"하는 것은 구성가능하다. 이러한 방법으로 복잡한 프로그램의 다양한 에러 관련 분기가 사용중인 DUT가 완전히 양호하더라도, 프로그램을 통한 트리거를 "동반"함으로써 실행될 수 있다.

발명의 구성 및 작용

<40> 이제 도 1을 참조하면, 본 발명의 원리에 따라 구성된 비휘발성 메모리 테스트 시스템의 블록도 1이 도시되어 있다. 특히, 도시된 시스템은 64개가 넘는 테스트 포인트를 갖는 DUT를 테스트하기 위해 함께 접합될 테스트 자원의 집합 요소를 허용하는 재구성에 대한 제공으로, 한번에 36개의 개별 DUT까지 64개의 테스트 포인트로 동시에 테스트할 수 있다. 이러한 테스트 포인트는 아직 다이싱 및 패키징되지 않은 집적 회로 웨이퍼의 일부에 관한 소정의 위치일 수 있으며, 또는 패키징된 부품의 핀일 수 있다. "테스트 포인트"라는 용어는 신호가 인가될 수 있는(예컨대, 전원 공급, 클럭, 데이터 입력) 전기적 위치 또는 신호가 측정될 수 있는(예컨대, 데이터 출력) 전기적 위치를 말하는 것이다. 우리는 산업상 관례를 따라 테스트 포인트를 "채널"이라고 할 것이다. 전술한 "함께 접합될 테스트 자원의 집합"은 36개의 테스트 사이트만큼 많이 있는 것으로 이해될 수 있으며, 그 각각의 테스트 사이트는 테스트 사이트 컨트롤러(4), (64 채널) DUT 테스터(6) 및 DUT(14)에 실제적으로 접속하는 (64 채널) 핀 전자 장치의 집합(9)을 포함한다. DUT를 테스트하는 것이 64개 또는 그 이상 채널을 필요로 할 경우, 단일 테스트 사이트는 그 DUT 상에서 테스트를 수행하기에 충분하며, 예컨대, 테스트 사이트 #1(도 1에

도시되어 있음)이 "단일 사이트 테스트 국(Single Site Test Station)"을 형성하고 또한 동작한다고 할 수 있다. 한편, 전술한 재구성의 소정의 형태가 유효할 때, 2개(이상)의 테스트 사이트가 128개의 채널를 갖는 보다 큰 하나의 동일한 테스트 사이트로서 기능하기 위해 함께 접합된다. 따라서, 도 1에 도시된 예를 다시 참고하면, 테스트 사이트 #35 및 #36은 "2사이트 테스트 국"을 형성한다고 할 수 있다.

<41> 반대의 경우를 간략히 살펴보면, 단일 DUT를 테스트하기 위해 전체 테스트 사이트가 필요하거나, 단일 테스트 사이트가 단지 단일 DUT를 테스트할 수 있다고 가정해서는 안된다. 웨이퍼가 2, 3 또는 4개의(반드시 그렇지는 않지만 아마 인접한) 다이(die)를 가지고 있다면, 그 테스트 채널 요구의 합은 64개 채널 이하이다. 그러한 DUT(15a - d)는 단일 테스트 사이트(예컨대, 도 2에 도시된 테스트 사이트 #2)에 의해 동시에 테스트될 수 있다. 이것이 가능한 이유는 당연한 과정으로 기술될 소정의 하드웨어 특징에 의해 증가된 각 테스트 사이트의 범용 프로그램가능성(programmability)때문이다. 이론적으로는, 테스트 사이트에 의해 실행되는 테스트 프로그램은 테스트 사이트의 자원의 일부분이 DUT 중 하나를 테스트하기 위해 사용되고, 다른 부분이 다른 DUT를 테스트 하기 위해 사용되도록 기록될 수 있다. 결국, 처음 두개의 논리적 조합체인 제 3 DUT를 갖고 있다면, 우리는 단일 테스트 사이트로 제 3 DUT를 테스트할 수 있을 것이며, 그래서 소위 그 "구성요소 DUT"를 유사하게 테스트할 수 있을 것이라고 가정할 것이다. 물론 중요한 차이점은 "제 3" DUT에 대한 하나의 단일화된 대답과는 반대로, 두개의 "구성요소 DUT" 중 어느 것이 통과(pass)할 것인지 아니면 실패(fail)할 것인지를 개별적으로 추적하는 것이다. 즉, "제 3" DUT의 어느 부분이 실패했는지에 관한 문제가 있다. 또한, 결함있는 DUT로 구동 신호를 제거하거나 제한하는 것을 포함하여, 어느 DUT가 결함을 나타내는지에 근거하여 테스트 프로그램내에서 분기하고, 동시에 테스트 프로그램이 절망적으로 멀티 쓰레드화(multi-threaded) 되는 것을 방지하는 다른 문제들이 있다. 단일 테스트 사이트에서의 이러한 "멀티 DUT 테스트 국"의 소정의 특징은 상당히 간단하며, 다른 것은 복잡하다. 멀티 DUT 테스트는 2이상의 테스트 사이트를 함께 접합한다는 개념으로 혼동해서는 안된다.

<42> 테스트 사이트 재구성의 개념이 없다면, 테스트 사이트와 테스트 국의 사이에 어떠한 차이도 없을 것이며, 우리 그 용어 중의 하나는 사용하지 않을 것이다. 그러나, 사실은 테스트 국의 수가 테스트 사이트의 수와 동일할 필요가 없다는 것이 기꺼이 이해될 것이다. 종래에는, 테스트 사이트가 기본 멀티 DUT 테스트(전체 테스트 사이트를 소비할 만큼 충분히 복잡하지 않은 DUT)에 대한 보다 많은 테스트 국을 만들기 위해 때때로 분할되기 때문에, 그 수에 있어서 차이가 있을 수 있었다. 그러나, 이제 그 차이는 또한, 멀티 사이트 테스트 국(단일 테스트 사이트에 있어서는 너무 복잡한 DUT)을 형성하기 위해 함께 접합된 테스트 사이트 때문일 수 있다.

<43> 그러면 계속해서, 테스트 시스템 컨트롤러(2)는 시스템 버스(3)에 의해 그 이름 말미에 #1에서 #36가 붙은 36개의 테스트 사이트 컨트롤러(4a - 4z)에 접속된다.(아래 문자(subscript) a - z가 단지 1에서 36까지가 아닌 26까지인 것이 사실이다. 그러나, 이 사소한 속임수는 많은 참조 요소(reference character)상의 많은 아래 문자에 대해 바람직한 것처럼 보이며, 그것은 잠재적으로 매우 혼란스러울 수도 있다.) 테스트 시스템 컨트롤러(2)는 비휘발성 메모리를 테스트하는 작업과 관련된 적합한 테스트 시스템 컨트롤러 프로그램을 실행하는 컴퓨터(예컨대, NT를 실행하는 PC)이다. 테스트 시스템 제어 프로그램은 원하는 테스트를 달성하기 위한 일(labor)(및 복잡성)의 계층적 분할에서 추상(abstraction)의 최고 레벨을 나타낸다. 테스트 시스템 컨트롤러는 테스트 탐침 및 DUT를 필요한 대로 움직이는 로보틱스 시스템(도시되지 않음)을 감독하는 것뿐만 아니라, 어느 프로그램이 서로 다른 테스트 사이트에 의해 실행되고 있는지를 판정한다. 테스트 시스템 컨트롤러(2)는 소정의 테스트 사이트가 단일 사이트 테스트 국으로서 수행하기 위해 프로그램되고, 소정의 테스트 사이트는 멀티 DUT 테스트 국으로서 수행하기 위해 프로그램되는 반면에, 다른 테스트 사이트는 멀티 사이트 테스트 국을 형성하기 위해 함께 접합된다는 개념을 뒷받침하는 방식으로 기능할 수 있다. 명확하게 그러한 환경에서는, 테스트되는 서로 다른 부품들이 있으며, 서로 다른 테스트가 서로 다른 부품에 사용되는 것이 가장 바람직하다. 유사하게, 모든 단일 사이트 테스트 국은 동일한 스타일의 부품을 테스트할 필요가 없으며, 멀티 사이트 테스트 국에 대해서도 그러할 필요가 전혀 없다. 따라서, 테스트 시스템 컨트롤러(2)는 필요한 테스트 사이트 접합을 달성한 후 사용 중인 다양한 테스트 국에 대한 적합한 테스트 프로그램을 불러일으키기 위해 커맨드를 발행하도록 프로그램된다. 테스트 시스템 컨트롤러(2)는 또한 테스트로부터 얻어진 결과에 관한 정보를 수신하여, 결함있는 부품을 버리기 위한 적합한 동작을 취할 수 있으며, 제어에 사용될 수 있는 다양한 분석, 즉, 공장 환경에서의 생산 프로세스에 대한 로그(log)를 유지할 수 있다.

<44> 테스트 시스템 그 자체는 상당히 크고 복잡한 시스템이며, 로보틱스 서브시스템을 사용하여 핀 전자 장치(9)에 접속된 탐침 아래의 하나 이상의 차세대 다이를 순차적으로 위치시키는 스테이지 상에 웨이퍼를 올려놓는데, 그 핀 전자 장치 위에서 그 차세대 다이(웨이퍼는 아직 다이싱되지 않았음)가 테스트된다. 테스트 시스템은 또한 적합한 캐리어 상에 올려놓아진 패키징된 부품을 테스트하는 데 사용될 수 있다. 후술하는 바와 같이, 얼마나

많은 테스트 사이트가 그 테스트 국을 형성하는 데 사용되는지 또는 얼마나 많은 테스트 국이 테스트 사이트 상에 존재하는지에 상관없이, 사용 중인 각 테스트 국과 연관된 적어도 하나의 테스트 사이트 컨트롤러가 존재할 것이다. 테스트 사이트 컨트롤러는 비휘발성 메모리를 테스트하기 위한 이전의 제품(예컨대, 에질런트 V1300 또는 V3300)에 또한 사용되었던 VOS(VersaTest O/S)라고 불리는 독점적인 운영 체제를 실행하는 조합된 프로그램 및 데이터 메모리의 36MB에서 64MB를 갖는 인텔(Intel)사의 i960 프로세서일 수 있는 내장된 시스템이다. 지금 우리는 단일 사이트 테스트 국에 대한 상황만을 고려할 것이다. 명확한 예를 위해, 테스트 사이트 #1이 테스트 국 #1으로서 기능하고 있고, WHIZCO 부품 no.0013을 테스트하는 것이라고 가정하자. 그러한 테스트 조건은 100여개의 서로 다른 유형의 테스트(선택된 정보의 패턴을 단지 저장한 후 검색(retrieve)하는 대량의 도즈(dose)뿐만 아니라 전압 레벨, 펄스 폭, 에지 위치, 지연을 변화시키고 모니터링하는 것)를 포함하고, 각 유형의 테스트는 DUT에 대한 수백만의 개별 메모리 사이클을 포함한다. 최고 레벨에서, 테스트 시스템의 오퍼레이터는 테스트 시스템 컨트롤러(2)에게 테스트 국 #1을 사용하여 WHIZCO 0013의 테스트를 시작하라고 지시한다. 당연한 과정으로 테스트 시스템 컨트롤러(2)는 내장된 (컴퓨터) 시스템인 테스트 사이트 컨트롤러 #1(4a)에게 관련 테스트 프로그램, 즉, TEST_WHIZ_(13)을 실행하라고 명령한다. 그 프로그램이 테스트 사이트 컨트롤러 #1의 환경에서 이미 이용가능하다면, 그것은 간단하게 실행된다. 만일 그렇지 않다면, 테스트 시스템 컨트롤러(2)에 의해 공급된다.

<45> 이제, 이론상으로, 프로그램 TEST_WHIZ_(13)는 전적으로 자급식(self-contained)일 수 있다. 그러나, 만일 그렇다면, 그것은 거의 확실하게 대형일 것이며, 테스트 사이트 컨트롤러(4a)내의 내장된 시스템의 프로세스가 원하는 속도로, 혹은 하나의 DUT 메모리 사이클로부터 다음 DUT 메모리 사이클까지 균일한 속도로, 테스트를 생성하기에 충분히 빠르게 실행하는 것은 어려울 것이다. 따라서, 어드레스의 시퀀스 및 기록될 또는 판독 동작으로부터 기대되는 관련 데이터를 생성하는 낮은 레벨의 서브루틴 유형의 활동이 DUT 테스터(6)내에 위치한 프로그램가능 알고리즘적 메카니즘에 의해 요구된 대로 생성된다. 그러나, 그것은 테스트 사이트 컨트롤러(4)내에 내장된 시스템에 의해 실행되고 있는 프로그램과 동기하여 동작한다. 이를 소정의 낮은 레벨의 서브루틴 방식의 활동 및 DUT(14)의 하드웨어 환경에 인접한 메카니즘(DUT 테스터)으로 DUT 메모리 사이클을 개시하는 작업을 전하는 것으로 생각하라. 일반적으로 말해서, 그 후, 테스트 시스템 컨트롤러(2)가 테스트 사이트 컨트롤러에 테스트 프로그램을 갖출 때마다, 그것은 테스트 사이트 컨트롤러에 대한 프로그래밍에 의해 기술되거나 또는 요구되는 전체 활동을 달성하기 위해 필요한 적합한 낮은 레벨의 구현 루틴(implementation routine)(아마 테스트되고 있는 메모리에 특정됨)을 갖는 관련 DUT 테스터를 또한 공급한다. 낮은 레벨의 구현 루틴은 "패턴"이라는 용어로 사용되며, 그것은 (높은 레벨의 프로그래밍 언어에서의 기능 및 변수가 이름을 갖고 있는 것과 같이) 일반적으로 불리어진다.

<46> 각각의 테스트 사이트 컨트롤러 #n(4)는 사이트 테스트 버스 #n(5)에 의해 관련 DUT 테스터 #n(6)에 조합된다. 테스트 사이트 컨트롤러는 사이트 테스트 버스(5)를 사용하여 DUT 테스터의 동작을 제어하기도 하고, 테스트 결과에 대한 정보를 그로부터 수신하기도 한다. DUT 테스터(6)는 테스트 조건에 포함된 다양한 DUT 메모리 사이클을 고속으로 생성할 수 있으며, 판독 메모리 사이클의 결과가 기대한 바와 같은지 결정한다. 필수적으로, 그것은 판독 및 기록 DUT 메모리 사이클의 대응하는 유용한 시퀀스를 개시함(즉, 대응하는 패턴을 실행함)으로써 테스트 사이트 컨트롤러로부터 전송된 ("패턴"이라고 불리는) 커맨드 또는 동작 코드에 응답한다. 개념적으로, DUT 테스터(6)의 출력은 DUT에 인가될 자극 정보이며, 그것은 또한 그로부터 응답 정보를 받는다. 이러한 자극/응답 정보(7a)는 DUT 테스터(6a)와 핀 전자 장치 #1 어셈블리(9a) 사이에서 통과된다. 핀 전자 장치 어셈블리(9a)는 DUT(14)에 인가될 수 있는 64개까지의 탐침을 지지한다.

<47> 전술한 자극 정보는 DUT 테스터에 사용되는 로직 디바이스의 소정의 계열(family)의 전압 레벨에 따라 표현된 평행한 비트 패턴의 시퀀스(즉, "전송 벡터" 및 기대되는 "수신 벡터"의 시퀀스)이다. 자극/응답내의 비트 위치와 다이로 이어지는 탐침 사이에 구성가능한 맵핑(configurable mapping)이 존재하며, 이 맵핑은 DUT 테스터(6)에 의해 이해된다. 개별 비트는 그 타이밍 및 에지 배치에 대해 정확하다. 그러나, 맵핑 이외에 그 비트는 DUT에 인가될 수 있기 전에 전압 레벨 시프팅을 필요로 할 수도 있다. 유사하게, 자극에 이은 DUT에서 비롯되는 응답은 DUT 테스터로 피드백되는데 적합하다고 생각될 수 있기 전에 버퍼링 및 (반대) 레벨 시프팅을 필요로 할 수 있다. 이러한 레벨 시프팅 작업은 핀 전자 장치(9a)의 기능이다. WHIZCO 0013을 테스트하기 위해 필요한 핀 전자 장치 구성은 ACME 사의 부품을 테스트하기 위해서는 아마 작동하지 않을 것이며, 아마도 다른 WHIZ 사의 경우에도 마찬가지일 것이다. 그래서, 핀 전자 장치 어셈블리는 구성가능할 필요가 있으며, 그러한 구성가능성은 PE Config 라인(8a)의 기능이다.

<48> 상기한 사항은 단일 테스트 사이트가 DUT를 테스트하기 위해 어떻게 구성되는지에 대한 간략한 구조적 개요에

대해 결론을 내린다. 우리는 이제 동작할 많은 테스트 사이트가 있을 때 발생하는 문제에 대해 관심을 돌린다. 예비적으로, 우리는 다수의 테스트 사이트를 갖는 테스트 시스템을 구성하기 위한 바람직한 실시예를 설명할 것이다. 여러 면에서, 우리가 이제 설명할 소정의 정보는 소비자 선호도 및 비용 이득 분석에 관한 시장 조사에 근거하는 선택의 문제이다. 그렇다고 하더라도, 이들 중 하나를 만들기 위해서는 분명한 선택을 해야 하며, 일단 행해지면, 전체 시스템을 통틀어 가시적인 특정한 결과가 있다. 적어도 일반적인 방법으로, 테스트 시스템의 하드웨어 특성의 광범위한 개요를 설명하는 것이 유용하다고 느껴진다. 이러한 특성 중 일부가 확실하지 않다고 하더라도, 이에 대한 지식은 그럼에도 불구하고 본 발명을 설명하기 위해 사용되는 다양한 예들을 이해하는 데 도움을 줄 것이다.

<49> 그러면 우선, 4개의 상당히 큰 카드 케이지(card cage)를 생각해보자. 각 카드 케이지는 전원 공급 및 냉각수(팬은 세정실 환경에서 오염의 근원이 될 수 있으며, 충분히 부하가 걸린 시스템에 대한 수십 KW의 분산된 열을 제거하기 위한 에어컨보다는 차가운 물이 값이 싸다.)이외에, 모판(mother board), 전면(front plane), 후면(back plane)을 갖고 있다. 각 카드 케이지내에는 9개까지의 어셈블리가 배치될 수 있다. 각각의 어셈블리는 테스트 사이트 컨트롤러, DUT 테스트 및 핀 전자 장치를 포함한다. 우리는 테스트 사이트 컨트롤러가 어떻게 함께 접합되는지에 대한 일반적인 개요를 설명할 것이며, 그것은 데이지 체인(daisy chain)에 사용되는 소정의 버스를 포함할 것이다.

<50> "데이지 체인"이라는 용어에 대해 간단히 설명하는 것이 아마 적절할 것이다. 시스템 요소 A, B, C 및 D를 생각해보자. 그것들이 그 순서대로 함께 데이지 체인될 것이라고 가정해보자. A를 떠나 B로 가는 정보 또는 제어 경로가 존재한다고 할 수 있으며, 그 후 B는 B를 떠나 C로 가는 트래픽(traffic)을 선택적으로 통과할 수 있으며, 그 후 C는 D로 가는 트래픽을 선택적으로 통과할 수 있다고 할 수 있다. 이러한 동일한 종류의 구성은 다른 방향의 트래픽에 대해서도 존재할 수 있다. 데이지 체인은 종종 우선 방안(priority scheme)을 만들기 위해 사용된다. 우리는 데이지 체인을 사용하여 다양한 테스트 사이트 컨트롤러 간의 마스터/슬레이브(master/slave) 관계를 만들 것이다. 우리는 이러한 데이지 체인 스타일 통신 구성을 접미 명사 "BUS" 대신에 "DSY"로 표시할 것이다. 그리하여, 우리는 커맨드/데이터 BUS 대신에 커맨드/데이터 DSY라고 할 것이다. 이제, 정보가 "B에 들어가서 선택적으로 통과한다"는 개념은 통과되기 전에 트래픽이 개별적인 일련의 도체상에 복제된다는 것을 암시할 수 있다. 그것은 그러한 방식일 수 있으나, 성능상의 이유로 어드레스가능한 개체(entities)를 갖는 정규 버스와 더 유사하다. 프로그램가능한 어드레스 맵핑 구성 및 하방의 테스트 사이트 컨트롤러의 일부를 "슬립(sleep)으로" 하는 능력에 의해, 그 단일 버스는 복수의 데이지 체인으로서 논리적으로 나타나게(즉, 기능하게) 될 수 있다. 결국, 데이지 체인은 커맨드 및 제어 정보를 위한 고성능 통로(pathways)이며, 만일 그렇지 않다면, 우리는 마스터/슬레이브 조합(멀티 사이트 테스트 국)이 단일 테스트 사이트가 동작하는 것 만큼 빠르게 동작하는 것을 기대할 수 없다는 것이 이해될 것이다. 데이지 체인의 성능 덕분으로, 다양한 DSY는 그 각각의 카드 케이지를 떠나지 않는다. 이러한 결정의 효과는 어떤 (따라서 얼마나 많은) 테스트 사이트가 함께 접합될 수 있는지에 대해 소정의 제한을 둘 수 있다는 것이다. 이론상으로, 이러한 제한은 근본적으로 필요한 것은 아니며, 관련된 기술적인 실용성이 진정으로 부족한 것도 아니다. 그것은 단지 카드 케이지내에 이미 9개의 테스트 사이트가 존재하기 때문에, DSY를 확장하는 것은 비교적 적은 추가 이익에 대해 상당한 비용을 증가시킨다고 생각된다.

<51> 이제, 도 1의 논의에 대해 재개하면, 각각 9개의 테스트 사이트 컨트롤러를 갖는 4개의 카드 케이지를 수용할 수 있는 다수의 테스트 사이트 컨트롤러(4a - 4z)를 생각해보자. 그것들을 4a - 4f, 4g - 4m, 4n - 4t 및 4u - 4z로 표시하자.(앞서 설명한 바와 같이, 이들이 명목상으로는 단지 26개의 문자이라는 것을 전혀 신경쓸 필요가 없다. 독자는 거기 어딘가에 다른 10개의 문자가 더 있을 것이라고 생각하게 될 것이다.) CMD/DAT DSY(17a)(Command & Data Daisy Chain)는 하나의 카드 케이지내에 있는 테스트 사이트 컨트롤러(4a - 4f)를 상호접속하고, 서로 다른 CMD/DAT DSY(17b)는 또다른 카드 케이지내의 테스트 사이트 컨트롤러(4g - 4m)를 상호접속한다. 동일한 구성이 나머지 카드 케이지 및 각각 테스트 사이트 컨트롤러(4n - 4t 및 4u - 4z)에 존재한다. 우리는 일찍이 실제로 DSY를 형성하는 버스의 "말단(tail end)"이 카드 케이지를 떠나지 않고, 다른 카드 케이지에서 다음 세그먼트의 선두가 된다는 점에서, DSY가 카드 케이지를 떠나지 않는다고 하였다. 대신에, 테스트 시스템 컨트롤러(2)로부터의 시스템 버스(3)는 모든 테스트 사이트 컨트롤러로 가서, 각각 카드 케이지를 떠나지 않는 DSY 세그먼트의 선두에서 마스터가 될 수 있다.

<52> 우리가 이제까지 논의하고 있는 CMD/DAT DSY(17a - d)는 다수의 테스트 사이트 컨트롤러(4a - 4z) 사이에 존재한다. SYNC/ERR DSY(18a - 18d) 및 DUT 테스터(6a 6z)에 대한 유사한 구성이 존재한다. SYNC/ERR DSY(18a - 18d)에 의해 전달된 동기화 및 에러 정보는 DUT 테스터가 일치하여 기능하도록 허용한다. 이 두개의 데이지 체

인(17 및 18)은 약간 서로 다른 정보를 전달하지만, 각각 하나 이상의 테스트 사이트를 함께 테스트 국으로 접합하기 위한 동일한 일반적인 메카니즘의 일부로서 존재한다.

<53> 이제 우리는 도 2에 대해 논의해 보기로 하는데, 도 2는 도 1의 DUT 테스터(6)의 단순화된 상세 블록도이며, 그 중에는 36개씩이나 있을 수 있다. 그 하나의 예만을 설명하는 것으로도 현재 충분하다. 도 2를 살펴보면, 특히 "단순화된" 블록도에 대해 스터프(stuff)로 상당히 잘 수용되어 있음을 알 수 있을 것이다. DUT 테스터(6) 내에 있는 것 및 블록도에 표현되어 있는 것 중 일부는 기능적으로 매우 복잡하여 현재 시판 중인 형태로는 이용가능하지 않다. 여기서 두가지 점을 지적하는 것이 적합하다. 첫째, 도 2를 포함하는 주요 목적은 전체 비휘발성 메모리 테스트 시스템(1)내의 중요한 동작적 환경의 기본 특성을 설명하는 것이다. 도 3 및 그 이후의 도면과 연관되어 충분히 설명되는 본 발명은 후술하는 도 2의 설명에 기술된 메카니즘의 확장 또는 그 동기부여적 근거가 도 2에서 발견되는 새로운 메카니즘일 것이다. 어느 방식이든, 이들 중 어느 것이 독자 앞에 놓여질 것인지 정확히 알려지지 않도록 쓰여진다. 현재 목표는 다수의 바람직한 실시예의 많은 서로 다른 상세한 설명에 대한 단순화된 그러나 정보가 많은 출발점을 제공하는 것이며, 그래서 그들 중 각각이 (서로 다른 각 발명에 대한 모든 것을 개시하는 하나의 "대형" 명세서와는 반대로) 적당히 간결해질 수 있다. 두번째는, 일반적으로 도 2와 전체적으로 일치하는데, 확대된 또는 확장된 재료가 단순화된 버전과 정확히 "매치업(match-up)"되지 않는 정보를 포함할 수 있다. 이는 에러가 있었다거나, 혹은 상태가 치명적으로 일관성이 없다는 것을 의미하지 않는다. 그것은 정확한 축소 이미지가 되도록 무언가를 단순화하는 것이 때때로 어렵거나 불가능할 수 있기 때문에 발생한다. 그 상황은 오히려 지도와 유사하다. 콜로라도의 표준 사이즈의 콜로라도의 도로 지도에 의하면, I-70상에서 동쪽으로 갈 때, 당신은 덴버의 I-25상에서 북쪽으로 갈 수 있음을 알 수 있다. 그것은 죄회전처럼 보인다. 그리고 과거에는 실제 좌회전이었으나, 지금은 그렇지 않고, 그 교차로에 대한 상세한 지도에 의하면, 일련의 구성요소 회전 및 사이에 끼여있는 사거리가 있음을 알 수 있을 것이다. 그러나 어느 누구도 그 표준 사이즈 도로 지도가 잘못되었다고 말할 수 없으며, 추상적인 레벨에서는 올바른 것이다. 마찬가지로, 상당히 복잡하게 보이는 외관에도 불구하고, 도 2는 추상적인 중간 레벨에서 동작하는 단순화한 것이지만, 일부 외관상의 좌회전이 단지 죄회전만은 아닌 경우도 있다.

<54> 도 1에 도시된 바와 같이, DUT 테스터(6)로의 주 입력(major input)은 테스트 사이트 버스(5)의 한 인스턴스이며, 이는 관심의 대상이 되는 DUT 테스터(6)의 인스턴스와 관련된 테스트 사이트 컨트롤러(4)로부터 비롯된다. 테스트 사이트 버스(5)는 테스트 사이트 버스상의 트래픽을 링 버스(85) 또는 VT 버스(89)상의 트래픽으로 변환하는 멀티 버스 컨트롤러(88)에 조합된다. 반대로 링 버스 트래픽이 VT 버스 트래픽으로도 변환될 수 있다. 도 2의 거의 모든 부분은 대규모 집적 회로의 일부이며, 타이밍/포맷팅 및 비교 회로(52)(이하에 설명됨)는 간단하게 하기 위하여 하나의 개체로 도시되어 있지만, 실제로 그러한 IC 8개로 이루어져있다. 다수의 Ext.DRAM (그 일부는 또한 내부 테스트 메모리(87)의 일부분임(도 3 참조))을 제외하고는, 도 2의 스터프의 나머지 대부분은 APG(Automatic Pattern Generator)이라고 불리는 또다른 대규모 IC의 일부이다. 링 버스(85)는 DUT 테스터(6)의 APG 부분 내의 주요 요소를 구성하고, 동작 모드를 설정하는 등의 목적을 위한 범용 인터메카니즘 통신 경로이다. 또한 다수의 APG 요소 사이에 다수의 매우 넓고 빠른 전용 경로가 있다. VT 버스(89)는 DUT 테스터 그 자체 내에서 이용하는 IC 대 IC 버스이다.

<55> 링 버스(85)는 테스트 사이트 컨트롤러가 DUT 테스터(6)의 APG 부분과 통신하는 메카니즘이다. 링 버스(85)는 마이크로컨트롤러 시퀀서(19)(이는 특수 목적 마이크로프로세서에 비유될 수 있음)와 조합된다. 넥스트 어드레스 계산기(Next Address Calculator)(102)에 의해 생성된 어드레스를 이용하여, 마이크로컨트롤러 시퀀서는 프로그램 메모리에 저장된 프로그램으로부터 인스트럭션을 인출(fetch)하는데, 그 프로그램 메모리는 그 마이크로컨트롤러 시퀀서(19)의 내부의 프로그램 메모리(PGM SRAM(20)) 또는 그 외부의 프로그램 메모리(EXT. DRAM(21)) 중 하나일 수 있다. 이들 두 개의 메모리는 프로그램 카운터(또는 인스트럭션 인출 어드레스)로서 역할하는 본질적으로 논리 공통 어드레스(63)인 것에 의해 어드레스되는 것처럼 보이고, 둘 중 어느 하나가 수행될 프로그래밍의 소스가 될 수 있으나, (1) 임의의 시간 주기 동안에는 메모리 중 오직 하나만이 인스트럭션 인출 메모리 사이클을 수행하고, (2) 실제로 두 메모리는 전기적으로 서로 다른 신호에 의해 어드레스된다는 점에 유의해야 한다. SRAM은 고속이고 진정한 랜덤 액세스를 허용하지만, 마이크로시퀀스 컨트롤러(19)(대규모 APG IC의 일부임)내의 유용한 공간을 소모하기 때문에, 그 크기가 제한된다. 외부 DRAM은 많은 양의 조절 가능한 용량으로 제공될 수 있지만, 선형 실행(linear execution)을 포함하며 어떠한 분기도 포함하지 않는 순차적 청크(chunk)로 액세스될 때에만 고속이다. SRAM(20)에서의 프로그래밍은 종종 매우 알고리즘적인 것인 반면, 초기화 루틴과 랜덤 데이터나 비정규 데이터 등과 같이 알고리즘적 프로세스에 의해 그리 잘 생성되지 않는 재료에 대해서는 EXT. DRAM(21)이 가장 적합하다.

<56> 넥스트 어드레스 계산기(102)는, 다수의 PROGRAM CONTROL FLAGS(25), OTHER FLAGS(55), 그리고 명확성을 위하여 별도로 도시되며(DFE 0:3(103) 및 DPE 0:3(104)) 멀티 DUT 동작을 위하여 제공되는 소정의 다른 신호에서 조절된 무조건적 점프(unconditional jump) 인스트럭션, 또는 조건적 점프 또는 조건적 서브루틴 인스트럭션에 응답하여, 실행되고 있는 테스트 프로그램에서 분기를 구현할 수 있다.

<57> 마이크로 콘트롤러 시퀀서(19)에 의하여 인출되고 수행되는 인스트럭션 워드는 208비트로 상당히 길다. 인스트럭션 워드는 13개의 16비트 필드로 구성된다. 이들 필드는 대개 마이크로컨트롤러 시퀀서 외부의 적합한 메카니즘을 위하여 인출된 인스트럭션 정보를 나타낸다. 그러한 필드는 그 관련된 메카니즘에 전용된다. 하나의 ALU INSTRUCTION(22) 세트가 8개의 16비트 ALU 집합(24)에 제공되고, 이와 다른 ALU INSTRUCTION 세트가 DUT 테스터 도처에 분포된 다양한 기타 메카니즘에 분배된다. 후자의 경우가 "VARIOUS CONTROL VALUES & INSTRUCTIONS"(42) 라인 및 범례(legend)에 의하여 표현된다.

<58> 8개의 16 비트 ALU(24) 각각은 관련된 16비트 결과 레지스터(result register)(각각의 ALU는 또한 수개의 다른 레지스터도 포함함)의 주변에 만들어진 통상적인 산술적 인스트럭션 레퍼토리를 갖는다. 이들 결과 레지스터 중 3개의 레지스터와 이에 연관된 ALU는 DUT에 공급될 완결 어드레스(complete address)로 다양하게 조합되는 X, Y 및 Z 어드레스 성분(27)을 생성하기 위한 것이다. 8개의 ALU/레지스터(DH & DL) 중 둘 이상이 최대 유효부(most significant portion : DH)와 최소 유효부(least significant portion : DL)로 분할되는 32비트 데이터 패턴(28)의 알고리즘적 생성을 지원하기 위해 제공된다. 마지막 3개의 ALU/레지스터(A, B, C)는 카운터로 사용되고, 프로그램에 의하여 지정된 일정한 반복 횟수 또는 기타 다른 횟수 조건이 완료되는 경우의 프로그램 제어 및 분기를 지원하는 다양한 PROGRAM CONTROL FLAGS(25)의 생성에 기여한다. 이들 PROGRAM CONTROL FLAGS(25)는 마이크로 콘트롤러 시퀀서(19)로 되전송되어, 마이크로 프로그램형 실행 메카니즘의 경우와 유사한 방식으로 인스트럭션 인출 어드레스(넥스트 어드레스 계산기(102)에 의하여 생성됨)의 값에 영향을 미친다. 또한 프로그램 분기를 달성하는 데 이용될 수 있는 다수의 기타 플래그(55)가 존재한다. 이러한 플래그(55)는 인출된 인스트럭션 워드의 서로 다른 필드에 의해 제어되는 DUT 테스터(6)내의 다른 메카니즘 중 다수의 메카니즘으로부터 비롯된다. 하나의 특정 추가적 플래그는 개별 아이템, PD_ERR(90)로 특별히 표시되어 있다. 이는 PGM SRAM(20)으로 공급되며, 포스트 디코드 메카니즘(60)으로부터 비롯되어, 그 포스트 디코드 메카니즘(60)이 에러를 발견했음을 지시한다. 또다른 그러한 추가적 플래그는 VEC_FIFO_FULL(26)이다. 보다 덜 세부적인 다른 도면에서는, OTHER FLAGS(54)와 함께 일괄적으로 취급될 수 있다. 본 명세서에서는 마이크로컨트롤러 시퀀서(19) 동작의 한 특징을 설명하는데 도움을 주기 위해 이를 별도로 분리하고 있다.

<59> VEC_FIFO_FULL은 마이크로컨트롤러 시퀀서(19)에 의한 추가적 프로그램 수행을 (일시적으로) 중지시키는 기능을 수행한다. 마이크로컨트롤러 시퀀서(19)에 의해 인출되는 인스트럭션과 DUT에 인가될 테스트 벡터를 최종적으로 핸드 오프하는 메카니즘 사이에는 많은 단계의 파이프라인(pipeline)이 존재한다. 또한, DUT에 인가되도록 이동하는 동안 벡터에 수반하는 정보(baggage)의 일부는 궁극적인 벡터 애플리케이션의 속도 또는 각각의 벡터 지속 시간에 관한 정보이다. 따라서, DUT로의 벡터 애플리케이션의 속도가 일정할 필요는 없으며, 특히 어떤 그룹의 벡터는 생성하는 것보다 인가하는데 더 오래 걸릴 수 있다. 마이크로컨트롤러 시퀀서는 단지 자신의 최대 속도로 프로그래밍을 수행한다. 그러나 명백하게, 파이프라인이 거의 무제한으로 탄력적(elastic)이지 않도록, 평균적으로 "벡터 소모"의 속도는 "벡터 생성"의 속도와 동일해야 한다. 벡터 FIFO(45)는 후술할 어드레스 맵퍼(29)의 출력에 존재하여, 파이프라인에서 탄성 수용체(elastic capacity)로 기능한다. VEC_FIFO_FULL 신호는 파이프의 선두 말단(head end)에서 새로운 벡터의 생성을 일시 중지시킴으로써 그 파이프라인에서 제한된 수의 단계를 초과하는 것을 방지하는데 사용된다.

<60> 계속하여, (16비트의 3배인 48비트의) X, Y 및 Z 어드레스 성분(27)이 어드레스 맵퍼(29)에 인가되며, 그 어드레스 맵퍼(29)는 순서화된 48비트 어드레스 공간에서 거의 임의적으로 재구성된 어드레스 값으로 사전 선택하여 출력한다. 이것을 이해하기 위한 전제로서, 어드레스 맵퍼(29)는 48비트 어드레스 공간을 완전히 채운 메모리이고 각각의 어드레스에서 48비트 값을 보유한다고 먼저 가정하자(그런 메모리는 오늘날 대형 냉장고의 크기일 것이라는 것은 잠시 생각하지 않기로 한다). 이런 메모리가 주어지면, 임의의 인가된 어드레스를 대체 어드레스로 사용될 수 있는 또다른, 임의적으로 선택된, 48비트 값으로 매핑할 수 있는 참조 테이블(look-up table)을 구현할 수 있다. 이런 어드레스 매핑이 바람직한 이유는 X, Y 및 Z 어드레스 성분이 특정 DUT 내부 구조(대개는 하나의 큰 선형 디코더로 구현되지 않음)의 관점에서 유용한 의미를 갖기 때문이다. 행, 열 및 계층(layer), 블록(block) 또는 페이지(page)의 개념은 테스트 엔지니어에게 매우 유용할 수 있으며, 물리적으로 서로 가까운 위치에서 발생하는 결함은 그 X, Y 및 Z 어드레스내의 대응하는 근접성(closeness)을 포함할 수 있다. 테스트 결과에서의 이런 패턴은 무엇이 오류인지를 인식하는데 중요하며, 또한, 설계 레벨에서 또는 예

비 섹션의 동작으로 오류 섹션의 동작을 회피하도록 부품을 재프로그래밍하는 생성 레벨에서 그 오류를 정정하는데 중요하다. 이러한 개념에서 두 가지가 문제된다. 첫 번째는 DUT에 인가될 실제 비트 수(말하자면, 32비트, 혹은 16비트)로 48비트를 줄이는 것이다. 여기서 그 줄이는 방법을 간략히 언급할 것인데, 이는 주로 X로부터 얼마의 비트를 취하고, Y로부터 얼마의 비트를 취하고, Z로부터 그 나머지 비트를 취하는 문제이다. 그러나 전적으로 그런 것은 아니며, 또한 이는 부차적 문제인데, 그 이유는 소정의 어드레스가 회로 내에 놓여있고 그 회로는 회로의 다른 섹션에 대한 좌-우(left-for-right)(또는 좌-우 및 상-하) 대칭 이미지(mirror image)를 가질 수 있기 때문이다. 이는 그 회로 내의 물리적 순서대로 어떠한 순차적 어드레스 값이 주어지는지에 관한 한, 그 비트가 의미하는 바를 재구성하는 효과를 가져온다. 이러한 칩 레이아웃 특성은 수 차례 발생할 수 있고, 말하자면, Y에 대한 하나의 비트 그룹이 해석되는 방법이, 말하자면, Z비트의 수반되는 값에 의존하는 경우 이는 당연할 것이다. 어드레스 맵퍼(29)가 제공되어 원래의 X, Y 및 Z 어드레스가 "재패키징"되도록 하는데, 말하자면, 그러한 내부 구조 배열을 갖는 메모리를 테스트할 사람을 돕기 위하여 이러한 종류의 일을 반영한다. 실제 이루어지는 방법에 관하여, 어드레스 맵퍼(29)는 상당히 많은 수의 상호 접속형 멀티플렉서(interconnected multiplexer)로 구성된다. 이는, 설명을 위하여 앞서 임시로 가정하였던 완전히 채워진 메모리 디코드 방안(fully populated memory decode scheme)의 완전히 임의적인 참조 테이블 동작(completely arbitrary look-up table behavior)을 구현할 수 없다. 그러나, 어드레서 맵퍼(29)는 필요에 따라 X, Y 및 Z 어드레스 성분의 서브필드를 재구성할 수 있는데, 그 이유는 특히 이는 48비트를 실제 필요한 수로 감소시킬 이와 다른 메카니즘이 역시 존재하기 때문이다. 어드레스 맵퍼(29)는 또한 국부적 범위 내에서 제한된 임의적 맵핑을 수행하도록 하는 3개의 16비트(어드레스) 참조 테이블을 포함한다.

<61> 어드레스 맵퍼(29)의 매핑 어드레스 출력(30)은 다수의 버퍼 메모리 및/또는 태그 RAM(Tag RAM)(31A-B) 및 에러 캐치 RAM1/2(32A/B)에 어드레스로서 인가되는데, 이들은 각각 서로 다른 기능을 수행하지만, 그럼에도 불구하고 일괄하여 내부 테스트 메모리(87)를 구성하는 4개의 메모리 세트 내에서 선택 가능한 부분(partitions)으로서 구현될 수 있다. 맵핑 어드레스 출력(30)은 Addr.비트 선택 회로(37)에 하나의 입력으로서 인가되는데, 그 Addr.비트 선택 회로(37)의 멀티플렉싱 기능은 당연한 과정으로 설명될 것이다. 내부 테스트 메모리는 서로 다른 기능에 사용되는 다수의 RAM 기반형 메모리 구조의 많은 인스턴스들을 포함하도록 구성될 수 있다. 이는 서로 다른 메모리 세트의 소정 부분이 그 관련된 목적을 위해 사용됨을 나타냄으로써 달성된다. 도 2에 도시된 것은 그러한 장치 중 하나이고, 그 장치는 테스트가 진행됨에 따라 변화될 수 있으며, 이러한 메모리 세트 용도의 전체 동작은 매우 동적인 것으로 생각되어야 한다. 내부 테스트 메모리의 장착물(예컨대, 에러 캐치 RAM(32A-B)) 중 어떤 것도 영구적인 하드웨어 장치는 아니다. 4개의 메모리 세트만이 영구적이다. 그러나 그러한 메모리 세트 중 어떠한 부품이 주어진 시간에 에러 캐치 RAM이 되는지는(실제로 규정된 하나가 존재한다면) 어떤 구성이 확립되는지에 따라 달라진다.

<62> 버퍼 메모리(31A, 31B)를 고려해보자. 그 기능은 DUT에 인가될 수 있는 데이터 패턴(33) 및 어드레스(34)를 보유하는 것이다. 데이터 패턴(33)과 어드레스(34)는 이들 버퍼 메모리가 이중 "포트 메모리"는 아닐지라도, 그 관련된 버퍼 메모리로부터의 실제 개별 출력이다. 그러나, 두 개의 서로 다른 메모리 세트 부분으로 구성되는 것이 바람직하다. 계속하여, 저장된 데이터(Stored Data)(33)는 하나의 메모리 세트내에서 유지되고, 저장된 어드레스(Stroed Address)(34)는 다른 메모리 세트내에서 유지되는 것이 바람직하다. 또한, 버퍼 메모리에 기록하기 위한 명시적 메카니즘을 지금까지 설명하지 않았는데, 이를 달성할 수 있는 한가지 방법은 실행되고 있는 프로그램의 명령(behest)에 따라 테스트 사이트 컨트롤러(4)에 의해 개시되는 어드레스된 버스 동작에 의한 것이다. 도 2에서 거의 모든 부분에 대하여 "플로어보드 하부(under the floorboards)", 즉 링 버스(85)라고 불리는 "유틸리티 서비스(utility service)"(도면을 매우 복잡하게 하기 때문에 그 대부분을 도시하지 않음)가 존재한다. 메모리 세트에 정보를 기록하는 더 빠른 또다른 방법은 도 3과 관련하여 설명될 것이다.

<63> 에러 캐치 RAM(32A-B)은 버퍼 메모리에 인가되는 동일한 어드레스에 의해 어드레스되고, 에러에 관한 정보를 저장하거나 또는 검색하는데, 그 동작은 후술할 포스트 디코드 회로와 함께 수행된다. 버퍼 메모리(31A-B)로부터의 경로(33, 34)에 있어서와 같이, (에러 캐치 RAM1(32A)로부터의) 경로(62A-D)는, 링 버스(도시되지 않음)에 의해 분배되는 구성 정보에 따라서, (에러 캐치 RAM으로 동작하도록 구성된) 메모리 세트의 일부분으로부터의 멀티플렉스된 출력(MUX'ed output)인 것이 바람직하다.

<64> 데이터 MUX(35)는 ALU의 집합(24)내의 레지스터 DH 및 DL로부터의 데이터(28)뿐만 아니라 버퍼 메모리(31A)로부터의 저장된 데이터 출력(33)을 입력으로서 갖는다. 데이터 MUX(35)는, PGM SRAM(20)에 저장된 값(36)에 따라서, 이들 입력(28,32) 중 어떤 것을 그 출력(38)으로 제공할 것인지에 관한 초기 선택을 수행하는데, 후술되는 바와 같이 변형되지 않는다면, 이 출력(38)은 전송 벡터 맵퍼/시리얼라이저(Serializer)/수신 벡터 비교 데이터

회로(40)에 두 벡터 성분 중 하나로서 인가된다(다른 성분은 Addr.비트 선택 회로(37)의 출력(39)이다).

<65> 회로(40)는 세 가지 벡터 관련 기능, 즉 벡터 성분(38, 39)을 DUT에 인가(전송)될 전체 벡터의 순서화된 논리 표현으로 어셈블링하는 기능과, 전송 벡터 논리 표현의 순서화된 비트와 그 신호(즉, 그 벡터 내 비트)를 대신하여 DUT와 접촉할 핀 전자 장치의 실제 물리 채널 번호(즉, 소정의 탐침 팁) 사이에 임의적인 동적 대응(매핑)을 적용하는 기능과, 컴파일러와 협력하여 전체 논리 벡터를 DUT(이를 허용하는 DUT)에 개별적으로 또한 순서대로 인가될 부분(pieces)으로 분할하는 기능을 수행할 수 있다. 이러한 기능 중 어떠한 기능이 수행될 것인지는 SRAM(41)로부터의 제어 신호에 의해 결정되는데, SRAM(41)은 마이크로컨트롤러 시퀀서(19)에 의해 인출되는 208비트 인스트럭션에서의 한 필드에 따라 어드레스된다. 또한, DUT 디스에이블 로직(DUT Disable Logic)(90) 섹션이 회로(40)에 포함된다. 그 목적은 4개의 DUT 중 하나 이상의 DUT가 디스에이블될 것인지를 지시하는 다양한 조건(일부는 정적이고, 일부는 테스트 결과에 의존할 것이지만, 그 모두가 프로그램적으로 규정되어 있음)에 응답하는 것이다. 이러한 지시는 4개의 신호 DD 0:3 44b(DUT 0, DUT 1 등에 대한 DUT 디스에이블))에 의해 전달된다. 이는 테스트 사이트에 관한 멀티 DUT 테스트를 지지한다. 회로(40)의 출력은 DUT 디스에이블 신호(44b)와 함께 벡터 FIFO(45)에 인가되는 64비트에 달하는 벡터(44)인데, 벡터 FIFO(45)는 가득 채워지는 경우 신호 VEC_FIFO_FULL(26)(그 의미와 이용은 전술하였음)를 생성한다. 벡터 FIFO(45)의 최상위 벡터는 주기 제너레이터(Period Generator)(49)(간략히 설명될 것임)에서 발생하는 신호 VEC_FIFO_UNLOAD(47)의 수신시 제거된다. 그 제거된 벡터(46)는 핀 전자 장치(9)의 관련 인스턴스를 거쳐 DUT에 접속되는 타이밍/포맷팅 및 비교 회로(Timing/Formatting & Comparison circuit)(52)로 인가된다. 즉, 핀 전자 장치(9)의 (여러 테스트 사이트 중) 각각의 인스턴스는 관련된 타이밍/포맷팅 및 비교 회로(52)로부터 전송 & 수신 벡터(7)와 핀 전자 장치 구성 정보(8)를 수신한다.

<66> 타이밍/포매팅 및 비교 회로(52)는 VT 버스(89)에 조합되어 구성 및 제어 정보를 수신한다. 타이밍/포매팅 및 비교 회로(52)는 우리의 목적상 단일 개체로 취급되지만 실제로 8개의 집적 회로임을 상기하기 바란다.

<67> 타이밍/포매팅 및 비교 회로(52)는 마이크로컨트롤러 시퀀서(19)의 프로그램 SRAM(20)과 동일한 인스트럭션 어드레스(작은 원 내부의 "A")에 의해 어드레스되는 내부 SRAM(54)을 갖는다. (외부 DRAM(53)이 내부 SRAM(54)을 대신하여 사용될 수 있으나, 이는 증가된 카운터(incremented counter)(도시되지 않음)에 의해 국부적으로 어드레스됨). 내부 SRAM(54)(또는 외부 DRAM(53))은 관련 포맷을 갖는 구동 및 비교 사이클(Drive and Comparison cycle)의 생성을 지원한다. 구동 사이클은 RAM(53, 54) 중 하나에 의하여 공급되는 사전 선택형 포맷을 사용하여 DUT로 전송 벡터를 인가한다. 비교 사이클은 DUT에 의해 제공되는 벡터를 수신하여 이를 검사하고, 또한 사전 선택 RAM 공급형 포맷에 따라, 이전에 제공된 비교 데이터와 매칭하는지를 판정한다. 구동 사이클 및 비교 사이클 양자 모두 그 지속 시간(duration), 로드(load)가 인가되는지 여부 및 언제 인가되는지, 데이터가 언제 래치(latched)되는지 또는 스트로브(strobed)되는지, 신호가 0으로의 복귀(Return-To-Zero)인지 아닌지, 구동 신호를 그 상보 신호(complement)로 둘러싸는지 여부에 관하여 적절하게 조절 가능하다. (이들 선택 사항은 전술된 다양한 포맷을 갖는다)

<68> 타이밍/포맷팅 및 비교 회로(52)에 의해 수행되는 비교는 채널마다의 기초에 따라 논리 값이 부적당하기 때문에(기능적 에러) 그리고/또는 그 전기적 특성이 수용 가능한 한계를 벗어나기 때문에(파라미터적 에러) 채널이 고장났는지에 관한 정보를 포함한다. 또한, 다수의 DUT 테스트가 수행되는 경우 어떤 채널이 어떤 DUT에 관련되는지를 알 수 있다. 이로써 4개의 신호 DFE 0:3(DUT # 기능적 에러)(103) 및 4개의 신호 DPE 0:3(DUT # 파라미터적 에러)(104)가 생성된다.

<69> 타이밍/포매팅 및 비교 회로(52)에 의해 수행되는 비교는, 수신 벡터 역맵퍼/디시리얼라이저(deserializer)(57)에 인가되는 64비트 값(56)을 생성하는데, 그 수신 벡터 역맵퍼/디시리얼라이저의 기능은 회로(40)의 논리적 반전이라고 간주할 수 있다.(회로(57)의 동작은 SRAM(41)에 의한 회로(40)의 제어에 대응하는 SRAM(58)에 의해 제어된다). 이어서, 회로(57)의 출력(59)이 포스트 디코드 회로(60)로 인가되고, 또한 에러 캐치 RAM1(32A)으로 인가된다. 현재는, 포스트 디코드 회로(60)가 입력되는 에러 정보(59) 및 이전에 에러 캐치 RAM1(32A)에 저장된 에러 정보 양자 모두를 프로그램 기준(programmatic criteria)을 통해 검사하여, 압축된 그리고 보다 용이하게 해석가능한 에러 정보(이는 경로(61)를 통해 다른 에러 캐치 RAM2(32B)에 다시 저장될 것임)를 생성할 수 있다고 충분히 말할 수 있다. 일예로 특정 범위의 어드레스 내부에 에러가 몇 차례나 있었는지에 관한 합계(count)를 생성할 수 있는데, 이 정보는 대체 회로(substitute circuit)를 구동하여 온칩 수리(on-chip repair)를 시도할 시기를 결정하는데 유용할 수 있다.

<70> 이제 주기 제너레이터(49) 및 그 관련 타이밍 SRAM(51)을 설명하겠다. 주기 제너레이터(49)와 타이밍 SRAM(5

1)은 마이크로컨트롤러 시퀀서(19)에 의해 인출된 각 208비트 인스트럭션에 대해 타이밍/포매팅 및 비교 회로(52)의 관련 동작에 관한 지속 시간을 결정하는 8비트 신호 T_SEL(43)에 응답한다. T_SEL(43)은 인출된 인스트럭션내의 서로 다른 필드에 의해 표현되는 다수의 제어값 및 인스트럭션(Various Control Values & Instruction)(42)의 멤버이다. T_SEL(43)은 8비트 값으로서 256개의 서로 다른 경우를 표현하거나 또는 인코딩할 수 있다. 이에, 이들 각 "경우" 그 값은 타이밍 SRAM(51)에 저장된 28비트 값이고 T_SEL에 의해 어드레스된다. 어드레스되는 각각의 28비트 값(23)은 19.5 피코초(picosecond) 해상도(resolution)로 원하는 지속 시간을 지정한다. 액세스되는 28비트 지속 값(23)의 시퀀스는 그 시퀀스의 개별 멤버가 벡터 FIFO(45)에 저장되어 있는 의도된 대응 벡터의 검색과 동시에 검색 및 인가되도록 주기 FIFO(50)에 저장된다.

<71> FIFO(50)내의 최초 엔트리에서의 거친(coarse) 타이밍 값 필드는 5nsec의 해상도를 갖는 지속 시간 정보를 전달하고, 그로부터 다음 전송 벡터를 벡터 FIFO(45)에서 타이밍/포맷팅 및 비교 회로(52)로 전송하는 신호 VEC_FIFO_UNLOAD(47)를 생성한다. 부속 신호 TIMING REMAINDER(48)가 또한 회로(52)에 인가되는데, 이 회로(52)에서 19.5 피코초로의 최종 해상도가 달성된다.

<72> 상기한 사항은 메모리 테스터의 전체적인 본질에 관한 개괄적인 논의를 종결한다. 출발점으로서 상기 사항을 가지고, 이제 스코프 모드 및 그 관련 주제로 주의를 전환할 것이다.

<73> 이제 도 3을 참조하면, 도 3은 스코프 모드에서의 동작 동안 착수된 주요 활동의 흐름도 형태의 개관이다. 도 3의 흐름도(64) 두개의 주요 섹션(65, 66))을 가진다. 제 1 섹션(65)은 초기 스코프 모드 활동을 수행하고, 제 2 섹션(66)은 획득 스위프 활동을 수행한다. 섹션(65)에 의해 수행되는 초기 활동부터 살펴보기로 한다. 초기 활동의 목적은 트리거 특수화를 설정하고, 타겟 시퀀스를 획득하고, 그 반복적으로 실행가능한 버전을 생성하는 것이다.

<74> 위치(67)는 소정의 다른 활동으로부터 흐름도(64)의 밸런스로 들어가는 것을 나타내는데, 그 위의 단계(68)는 테스트 프로그램의 실행이 수반되는 트리거 특수화의 설정을 나타낸다. 현재의 바람직한 실시예에서, 트리거 특수화는 레지스터 X, Y, DH, DL, A, B, C의 내용상의 "AND" 조건인 ALU 성분을 포함한다. 이러한 레지스터의 임의의 서브세트(subset)(그 모두를 포함하거나, 하나도 포함하지 않음)는 트리거 특수화의 ALU 성분내에 포함될 수 있다. 각 레지스터에 대해, 그 비트의 임의의 서브세트(그 모두를 포함하거나, 하나도 포함하지 않음)가 지시될 수 있다. 모든 선택된 레지스터의 모든 지시된 비트가 지정될 때마다 트리거 특수화의 ALU 성분에 도달된다. 트리거 특수화는 또한 프로그램의 단지 소정의 부분 또는 그 전체 프로그램이 트리거가 만족되도록 할 수 있는 것을 지시하는 것이 가능하다는 점에서 프로그램 성분을 가질 수 있다. 트리거 특수화는 또한 적절한 트리거 특수화의 만족과 트리거된 활동을 개시하기 위한 트리거 신호의 방출을 분리할 지연(DUT 사이클내에서 표현되는 것이 바람직하지만, 절대적인 시간 간격으로서도 가능함)을 포함할 수도 있다.

<75> 트리거 특수화 활동이 사용자 인터페이스 레벨에서 나타날 수 있는 다양한 방법이 있다. 바람직한 방법은 운영체제(테스트 시스템 컨트롤러(2)상에서 실행됨)에 대해 스코프 모드 제어 프로세스를 실행할 윈도우를 제공하는 것이며, 여기에서 사용자는 어느 테스트 사이트 컨트롤러인지, 어느 DUT인지 및 파형 표시가 생성될 채널뿐만 아니라, 트리거 특수화의 프로그램과 ALU 성분을 지정할 것이다.

<76> 단계(69)에서는, 테스트 프로그램 활동이 트리거 특수화에 도달되도록 할 때까지(및 임의의 지정된 지연에 또한 도달했을 때까지) 대기 경로(70)가 발생하며, 그 때 단계(69)가 단계(71)로 전이된다. 이 시점에서 임계치 VOH는 방해받지 않은 상태로 남아 있으며, 프로그램은 128을 초과한 DUT 사이클에 대한 자연스러운 과정을 실행하게 된다. 이 128 DUT 사이클은 파형이 생성될 타겟 시퀀스이다. 타겟 시퀀스의 초기 실행 동안 PGM SRAM(20)로부터 인출된 인스트럭션 워드에 대한 다음 128 어드레스는 스코프 모드 FIFO(130)(도 4와 관련하여 설명될 것임)내로 로깅(logging)된다. 이 시점에서, 그 128 인스트럭션 워드가 무엇일 수 있는지에 관한 어떠한 제한도 없으며, 그것은 분기 인스트럭션을 포함할 수 있다.

<77> 타겟 시퀀스의 실행을 완료할 때, 테스트 프로그램의 실행은 중지되고, 스코프 모드에 대한 감독 환경은 흐름도(64)의 단계(72)에 도시된 활동을 수행한다. 그 활동은 링 버스(85)를 사용하여 스코프 모드 FIFO(130)내에 저장된 어드레스를 판독하고, PGM SRAM(20)으로부터 대응하는 인스트럭션 워드를 추출하기 위해 그 어드레스를 사용하며, 타겟 시퀀스내의 다음 워드를 지시하기 위해 그 인스트럭션 워드내의 임의의 분기 필드를 설정하고(이것은 의무이다. 파형을 불안정하게 하기 때문에 타겟 시퀀스가 계속해서 그 자신의 조건적인 분기를 하게 할 수 없다), 그 후, 그들을 PGM SRAM(20)의 예비 부분내에 순차적으로 저장하는 것이다.

<78> 테스트 프로그램의 후속 실행은 또한 트리거 조건을 만족할 것이며, 그 때에 테스트 프로그램은 일시적으로 중

지되고, VOH 및 샘플 타이밍 오프셋에 대한 값은 정상 값에서 획득 스위프의 다음 단계에서 사용되기 위한 값으로 변화되며, 타겟 시퀀스가 실행된다. 여기에 이것이 구현될 수 있는 두가지 방법이 있다. 첫째, 스코프 모드에 대한 감독 프로그램은 PGM_QUIT가 스코프 모드 동안 발생했다는 것을 검출할 수 있고, 프로그램 카운터를 예비 영역의 개시를 위한 어드레스로 이동시키는 넥스트 어드레스 계산기(102)(도 3 참조)내의 내부 프로그램 카운터 레지스터(도시되지 않음)로의 링 버스를 통한 변화에 따른 VOH 및 샘플 타이밍 오프셋 변화(스코프 모드 VOH 레지스터(166) 및 스코프 모드 샘플 타이밍 레지스터(168)에 기록함으로써 행하여짐. 양자 모두 도 5와 관련하여 설명될 것임)를 수행할 수 있다. 넥스트 어드레스 계산기(102)가 링 버스에 접속되는 것은 이러한 이유 때문이다. 이러한 활동이 달성될 수 있는 두번째 방법은 이와 유사하나, 넥스트 어드레스 계산기가 예비 영역의 시작 어드레스를 포함하기 위한 추가적 내부 레지스터(도시되지 않음)를 갖도록 함으로써 감독 스코프 모드 프로그램으로부터 부담의 일부를 덜어주는 것이며, 그것은 트리거된 인스턴스가 발생한 후 (획득 스위프 값을 적당한 값으로 하기에 적절한) 지연을 따라가도록 자동적으로 점프할 수 있다.

<79> 흐름도(64)의 획득 스위프 활동 부분(66)의 목적은 PGM SRAM(20)의 예비 부분내에 저장된 실행가능한 타겟 시퀀스를 반복적으로 수행하는 것이다. 단계(73)에서 테스트 프로그램은 (테스트 프로그램내의 알고리즘에 중요한 임의의 초기화가 발생하도록) 처음부터 재시작된다. VOH 및 샘플 타이밍 오프셋에 대한 정상 값으로 재시작되고, 예상하는 바로는 테스트 프로그램이 이전에 실행했던 것만큼 정확히 실행할 것이며, 결국 초기 실행에 있어서와 정확히 동일한 방식으로 트리거 특수화를 만족시킬 것이다.

<80> 테스트 프로그램이 실행되는 동안, 단계(74)는 루프 경로(75)를 통해 트리거 특수화의 만족을 기다린다. 도달할 때, 단계(76)는 테스트 프로그램을 중지시키고, 이어질 연속적인 획득 스위프 단계에 대한 첫번째(다음) 값으로 VOH 값을 설정한다.(여기에서 이 용어는 다수의 획득 스위프 단계가 획득 스위프를 만든다는 것이다.) 샘플 타이밍 오프셋에도 동일한 사항이 행하여진다. 그 다음에, 단계(77)는 테스트 프로그램이 계속 진행되도록 허용하고(타겟 시퀀스로 점프하게 됨), 수신 벡터가 ECR2(34b)내에 저장되도록 하는 메카니즘을 작동시킨다. 이제 타겟 시퀀스가 완성될 때까지 단계(78)에서 루프 경로(79)를 통해 기다린다. 도 4의 신호 PGM_QUIT(157)는 이를 지원한다. 마이크로컨트롤러 시퀀서(19)가 실행하는 것을 멈출 때마다 그 값은 TRUE이다. 이는 변경된 타겟 시퀀스의 129번째 인스트럭션 워드가 중지 인스트럭션이 되도록 함으로써 달성될 수 있다.

<81> 타겟 시퀀스의 실행이 완성될 때, 단계(80)는 ECR2(32b)의 내용을 스코프 모드의 감독 활동에 이용가능한 메모리로 이동시킨다(그것은 테스트 시스템 컨트롤러(2)상에서 실행하는 운영 체제하에 실행되는 프로그램인 것이 바람직함). 그 때에 스코프 모드 소프트웨어는 그에 따라 얻어진 부분적 결과를 형성하고 표시하는 작업을 시작할 수 있다.

<82> 다음 단계(81)는 획득 스위프의 끝에 도달했는지를 판정하는 것이다. 도달하지 않았다면, 경로(82)는 단계(73)를 고리 모양으로 만들어, 획득 스위프가 다른 단계로 계속될 수 있다. 흐름도(64)는 다음과 같은 면에서 단순화된다. 우리는 샘플 타이밍 오프셋의 전체 한벌의 각각 서로 다른 설정 값을 통해, VOH가 가장 빨리 스위핑된다는 개념을 다루는 외부 루프내의 내부 루프를 나타내지 않았다. 즉, VOH는 가장 빨리 변화하고, 그 변화의 시퀀스를 반복하며, 샘플 타이밍 오프셋은 가장 느리게 변화하고, 단지 하나의 변화의 시퀀스를 경험한다. 그리고 흐름도는 VOH를 변화시키기 위한 전략, 즉 철저한 선형 시도(end-to-end linear trials), 이원 검색 또는 예상 검색을 지정하지 않는다. 그리고 흐름도는 주어진 샘플 타이밍 오프셋에 대한 VOH의 변화를 종결하기 위한 전략, 즉 그 모든 것을 실행하거나, 옳은 대답이 발견되었다는 확신이 있을 때 정지하는 것을 지정하지 않는다.

<83> 획득 스위프가 종결된 후에 단계(81)로부터의 "예(YES)" 경로는 스코프 모드내의 감독 메카니즘으로 전이되는데, 이 감독 메카니즘 상에서 완성된 파형이 표시, 저장, 프린트 등이 될 수 있다.

<84> 이제 도 4를 참조하며, 도 4는 도 3에 도시된 활동이 어떻게 구현되는지에 관한 하드웨어 측면의 단순화된 블록도이며, 트리거 특수화의 만족에 근거하는 에러 플래그 재밍(jamming)에 필요한 하드웨어를 포함한다. 도 4는 도 2의 블록도내의 아이템과 관련된 3개의 부분(84, 86, 91)으로 분리가능하다. 즉, 부분(84)은 도 2의 마이크로시퀀서 컨트롤러(19)의 일부를 확대한 것이며, 부분(86)은 도 2의 8개의 16비트 ALU(24)의 일부를 확대한 것이고, 부분(91)은 도 2에 도시된 내부 테스트 메모리(87)내에서 발견되는 스터프를 확대한 것이다. 따라서, 도 4는 도 2(이 도면의 대부분은 현재 관심의 대상이 되는 특징과 직접 관련이 없게 됨)의 일부만을 확대한 도면이다.

<85> 부분(86)으로 시작할 것인데, 그것의 작업은 ALU_TRIGGER(96)이라고 불리는 신호를 생성하는 것이다. 부분(86) 및 ALU_TRIGGER는 트리거 특수화의 ALU 성분을 나타낸다. 우리는 X 레지스터에 대해 행해지는 것을 보여준다.

즉, 다른 7개의 레지스터(Y - C)에 대해 행해지는 것은 동일하다. 우리는 또한 이러한 레지스터에 대해 행해지는 것이 어떻게 신호 ALU_TRIGGER(96)를 생성하기 위해 조합되는지를 보여준다. 따라서, 시작되는 곳은 X 레지스터 스터프의 조사를 가진다.

<86> 링 버스(85)와 통신하는 16비트 X_ALU 레지스터(97)가 존재한다는 것을 유의해야 한다. 이는 실제 도 2의 X 어드레스 레지스터이며, 그의 비트는 48 어드레스 비트(27)의 3분의 1을 차지한다. 도 4에는 링 버스(85) 및 16개씩의 2입력 AND 게이트(100)에 접속되는 X_ALU 레지스터(97)가 도시되어 있다. X_ALU 레지스터(97)는 그 내용에 관한 산술 연산을 달성하기 위한 모든 종류의 다른 스터프에 접속된다는 것이 당연히 이해될 것이다. 그러나, 간략함을 위해서, 이 모두를 생략하였다. AND 게이트(100)의 출력이 16비트(도면을 상당히 도와주는 단순화)인 것도 이해될 것이다. AND 게이트(100)에 인가되는 다른 16비트는 16비트 X_마스크 레지스터(99)로부터 나온다. X_마스크 레지스터(99)내의 1 패턴은 X_ALU 레지스터(97)내의 어느 비트가 트리거 특수화내에서 관심의 대상이 되는지를 선택한다. 또한, 16비트 X_트리거 레지스터(98)가 존재한다. 그것은 트리거 특수화의 X 레지스터 성분을 포함하고, 그 목적으로 링 버스(85)에 접속된다. 또한, 그것은 그 입력의 다른 세트가 X_마스크 레지스터(99)의 세트인 또다른 16개씩의 2입력 AND 게이트(101)에 조합된다. AND 게이트(100)의 16개의 출력이 AND 게이트(101)으로부터의 16개의 출력과 동일(비트와 비트가 매칭됨)할 때마다, 트리거 특수화의 X 레지스터 성분이 만족된다. 이러한 환경의 결정은 =? 회로(103)를 위한 목적인데, 회로(103)은 입력으로서 AND 게이트(100, 101)의 두개의 16비트 출력을 수신하며, 그것은 출력으로서 동일 신호(104)를 생성한다.

<87> 간략함으로 위해 도시되지 않았지만, 관련 =? 회로(106) 및 그 출력(105)을 제외하고, 부분(86)의 Y 레지스터 회로는 동일하다. 실제로 6개가 넘는 동일 출력(각각의 남아있는 ALU 레지스터마다 하나씩)이 존재하는 것이 이해될 것이며, 그 중 하나만(110)이 도시되어 있다. 마스킹되지 않은 모든 ALU 레지스터 비트가 그 대응하는 _트리거 레지스터에 매치될 때마다 신호 ALU_트리거(96)가 생성된다. 따라서, 8개의 =? 회로로부터의 104...110의 집합은 AND 게이트(111)에 의해 함께 AND 연산되어 ALU_트리거(96)를 생성한다.

<88> 이제 마이크로시퀀서 부분(84)을 살펴보면, 신호 ALU_트리거(96)는 만족된 트리거 특수화를 생성하는 단지 하나의 성분이다. 또한 만족되어야 하는 프로그램 성분이 남아있다. 그 프로그램 성분은 신호 트리거_인에이블(93)에 의해 표현된다. 트리거_인에이블(93)이 존재한다(그 근원은 다음에 설명함)고 가정하면, 트리거_인에이블(93)과 ALU_트리거(96)는 AND 게이트(95)에 의해 조합되어, TRUE일 때, 기본 트리거 특수화에 도달하였지만, 실제 트리거가 발생되도록 허용되기 전에 지연이 만료될 필요가 있다는 상태를 나타내는 신호 ARM_시스템_트리거(113)를 생성한다.

<89> 이러한 생각으로 나아가면, 신호 ARM_시스템_트리거(113)가 그 지연의 양이 링 버스(85)에의 접속에 따라 지정되는 지연 회로(114)의 시작 입력에 인가되는 것을 유의해야 한다. 지연은 0 또는, 절대적인 시간 간격으로서 또는 DUT 사이클의 수로서 표현되는 소정의 다른 양일 수 있다. 그러한 회로는 통상적이라고 여겨지며, 특정의 세부사항은 생략하였다. 지연 회로(114)는 사전 설정 가능 카운터, 타이머, 전압 램프(ramp)/임계치 구성을 포함하는(그러나 이에 한정되지 않음) 다수의 적합한 유형의 지연 메카니즘 중 어느 하나일 수 있다. 어느 경우에도, 지연이 ARM_시스템_트리거(115)의 온셋(onset) 후 만료되었을 때, 신호 시스템_트리거(115)가 방출된다. 그것은 세트/리세트 플립 플롭(116)이 세트 입력에 조합된다. 시스템_트리거(115)는 플립 플롭(116)의 Q 출력이 TRUE가 되도록 한다. 그 Q 출력은 신호 TRIGGERED(122)이다. 그것은 타겟 시퀀스의 끝까지 유지될 것이며, 도 4에서의 다양한 다른 위치에서 사용된다. 획득 스위프내의 다음 단계의 시작에서 신호 TRIGGERED(122)는 신호 START(131)의 발생으로 인해 FALSE로 될 것인데, 신호 START(131)는 링 버스(85)에 조합된 원샷(132)으로부터 방출되며, 획득 스위프내의 다음 단계의 일부로서 테스트 프로그램의 재시작을 감독하는 소프트웨어에 의해 활성화된다.

<90> 이제 신호 TRIGGER_ENABLE(93)이 어떻게 시작되는지 살펴보기로 하자. 그것은 마이크로시퀀서 컨트롤러(19)에 의해 사용되는 208비트 크기의 인스트럭션 워드내의 비트라고 간단히 대답할 수 있다. 그것은 사용자가 스코프 모드에 대한 감독 메카니즘에 소스 코드내의 이러 이러한 라인은 어느 것의 머신 인스트럭션이 트리거링을 가능하게 할 것인지를 의미하는 것이라고 지정했을 때 설정되었다(그럴 가능성이 매우 높다). 또는 컴파일러가 프로그래머에 의해 명확히 놓여진 적합한 소스 코드 "여기에서 트리거 가능" 인스트럭션을 직면했을 때 설정되었다(그럴 가능성이 별로 없다). 첫번째 경우에서 스코프 모드에 대한 감독 메카니즘은 머신 인스트럭션 그 자체를 변경하고(이는 컴파일러 연루를 필요로 할 수도 있고, 안할 수도 있음), 두번째 경우에서, 사용자는 테스트 프로그램 그 자체의 소스 코드를 변경하여 재컴파일링하였다. TRIGGER_ENABLE 비트 세트를 갖는 단지 하나의 인스트럭션 워드가 존재할 수 있으며, 또는, 그들의 연속적인 줄(string)이 존재할 수 있으며, 또는 테스트 프

로그램이 실행될 때 비트가 오고 가는 중간 정도의 그 어떤 것이 존재할 수 있다.

<91> 계속해서, 208비트 인스트럭션 워드가 넥스트 어드레스 계산기(102)에 의해 생성된 어드레스(63)에 의해 어드레스되는 PGM SRAM으로부터 인출된다. 넥스트 어드레스 계산기(102)는 관심의 대상이 되는 수개의 추가적인 기능을 행한다. 첫째, 그것은 프로그램을 실행할 때마다 신호 BUSY(156)을 생성한다. 이는 마이크로 시퀀서 컨트롤러(19)에 의해 패턴의 실행을 제어함에 있어서 다수의 감독 메카니즘을 지원한다. 넥스트 어드레스 계산기는 또한 프로그램 실행을 중지할 상태에 직면하였다는 것을 의미하는 신호 PGM_QUIT(157)를 생성한다. 이러한 상태 중 하나는 신호 TRIGGERED(122)와 스코프_모드(138)의 AND이다. 이를 구현하기 위한 로직은 넥스트 어드레스 계산기(102)의 내부에 있으며, 명확하게 도시되어 있지 않다. 넥스트 어드레스 계산기(102)는 또한 신호 PGM_STEP(145)(원 내부의 B로 표시됨)를 생성하는데, 그 온셋에서의 의미는 새로운 유효 어드레스가 넥스트 어드레스 계산기(102)의 출력(63)에 존재한다는 것이다. PGM_STEP(145)의 온셋은 다른 메카니즘에게 패턴(테스트 프로그램)내의 다른 단계가 막 실행하려고 한다는 것을 알려주는 데 사용될 수 있다.

<92> 도 4의 이 부분을 자세히 살펴보면, 신호 PGM_QUIT(157) 및 TRIGGERED(122)가 링 버스에 의해 판독가능한 스테이터스 레지스터(Status Register)의 관련 비트(각각 158a 및 158b)에 조합된다. 이는 또한 정상 및 트리거된 동작을 감독하는 다수의 감독 소프트웨어 메카니즘을 지원한다는 것을 유의해야 한다.

<93> 이제 링 버스(85)에 접속된 스코프 모드 FIFO(130)(그 내용이 그 버스상에서 데이터 출력으로서 판독될 수 있음)의 존재를 살펴보자. 그것은 또한 넥스트 어드레스 계산기(102)(에 의해 생성된 어드레스)의 출력에 접속된 데이터 입력을 가진다. 이러한 접속은 타겟 시퀀스의 어드레스의 획득 및 그 이후의 PGM SRAM(20)(또한 링 버스(85)에 접속됨)의 예비 부분으로의 변경 및 저장을 위한 검색을 지지한다. 스코프 모드 FIFO는 신호가 LOAD 단자에 인가될 때 자동적으로 증가하는 내부 어드레싱 메카니즘을 가진다. 타겟 시퀀스를 저장하거나 판독하는 초기에 이 내부 어드레스가 다시 0으로 되도록 하기 위해서, RESET 단자는 그것에 조합된 START 신호(131)를 가진다. LOAD 단자는 AND 게이트(135)의 출력에 의해 구동되는데, AND 게이트(135)의 입력은 TRIGGERED(122)이며, FIFO(130)상의 FULL 단자로부터의 (인버터(134)에 의해) 반전된 신호이다. 이러한 구성에 의해, 타겟 시퀀스가 발생할 때, TRIGGERED가 TRUE로 되면, FIFO가 가득 찰 때까지 타겟 시퀀스를 연속적으로 저장할 수 있게 된다. 그것이 발생했다면, 스코프 모드에 대한 감독 소프트웨어는 FIFO의 내용을 판독할 수 있고, PGM_SRAM(20)으로부터 대응하는 인스트럭션 워드를 인출할 수 있으며, 그 인스트럭션 워드를 변경하여, 다음 인스트럭션으로 빠지도록 모든 분기를 조절할 수 있고 그 후, 변경된 인스트럭션을 PGM_SRAM(20)의 예비 부분내에 저장할 수 있다.

<94> 이제 링 버스(85)에 접속된 단일 비트 스코프 모드 레지스터(137)를 살펴보자. 그것은 감독 소프트웨어에 의해 설정되어 트리거 특수화가 유효한 상태에 있는 것을 지시한다. 스코프 모드 레지스터(137)의 출력은 신호 스코프_모드(138)이며, 그것은 두 개의 위치, 즉 에러 프래그 재밍(jamming) 메카니즘 및 내부 테스트 메모리(87)의 일부를 확대한 것인 부분(91)에 의해 사용된다. 동작의 스코프 모드, 즉 부분(91)을 지지하는데 필요한 도 4에 관한 마지막 사항을 살펴본 후까지 에러 플래그 재밍에 관한 논의는 뒤로 미룬다.

<95> 부분(91)의 목적은 타겟 시퀀스에 속하는 수신 벡터를 ECR2(32b)내에 저장하는 것이다. 그 수신 벡터는 한번에 획득 스위프내의 한 단계씩 거기에 캐쉬(cache)될 것이며, 파형의 구성에 대한 제어 스코프 모드 소프트웨어에 내려질 것이다. 저장되는 것은 비교 데이터(146)이다(그것은 도 5에 확대된 바와 같이, 타이밍/포맷팅 및 비교 회로(52)를 통해 핀 전자 장치(9)로 전달될 것이다.). 그러한 비교 데이터(146)가 ECR2의 데이터 입력에 제공되면, 그 순간에 우리가 관심있는 것은 어떻게 그것이 저장되게 하는가이다. 그것은 에러가 ECR, 즉, 신호 ECR_LOG_ENABLE(94)내에 저장되는지 판정하기 위한 정상적 메카니즘의 존재 혹은 부재에 따르는 방법으로 달성되어야 한다. 그것은 프로그래머가 테스트 프로그램의 일부에 대해 에러 데이터를 로깅하는 것을 원하는지 여부에 따라, 설정되거나 혹은 설정되지 않는 208비트 인스트럭션 워드내의 비트이다. 그렇다면, 우리가 해야 할 필요가 있는 것은 타겟 시퀀스가 실행되고 있을 때마다 ECR2(32b)에 대한 기록 인에이블 단자가 TRUE가 되도록 하는 것이다. 그것이 MUX(141)의 목적이며, MUX(141)는 스코프_모드가 FALSE일 경우(정상적인 동작), ECR_LOG_ENABLE로 ECR2의 기록 인에이블을 구동하고, 또는 스코프_모드가 TRUE일 경우, TRIGGERED로 구동한다. 다른 MUX(142)도 스코프_모드를 사용하여 어떤 어드레스가 ECR2에 인가되는지를 변화시킨다. 동작의 정상 모드에 있어서, 그것은 단지 어드레스 맵퍼(29)에 의해 공급되는 통상의 어드레스이다. 스코프 모드 동안 그것은 그 값이 신호 START(131)로 0으로 리셋되며, 신호 PGM_STEP(145)에 의해 증가되는 증가가능 레지스터(144)에 의해 공급되는 서로 다른 어드레스이다. 즉, 그것은 타겟 시퀀스의 실행과 동시에 증가한다. 이러한 구성은 타겟 시퀀스가 실행됨에 따라 비교 데이터(VOH 임계치에 도달되거나 혹은 도달되지 않음)를 저장한다. 프로그래

머가 테스트 프로그램에 대한 에러 데이터의 정상적인 로깅을 위해 ECR2를 사용하지 않을 것이 기대된다.

<96> 마지막으로, 링 버스(85)에 접속된 4비트 레지스터(147a - d)에 대해 살펴보자. 그것은 4개의 1비트 세그먼트를 포함하는데, 그 중 3개는 정상적인 에러 플래그 대신에 채워넣어진(jammed) 값을 나타낸다. 따라서, FERR_JAM(147a)은 FERR(154)를 대신할 수 있고, PERR_JAM(147b)은 PERR(153)를 대신할 수 있으며, ECR_FLG_JAM(147c)은 ECR_FLG(152)를 대신할 수 있다. 이러한 대체는 스코프_모드(138)이 TRUE이고(스코프 모드 레지스터(137)가 도달되고 있는 트리거 특수화의 결과로서 설정됨), 네번째 비트 TRIGGER_JAM_ENABLE(147d)이 또한 설정될 때 수행된다. AND 게이트(148)는 이러한 상태를 검출하고, 그 출력은 MUX(149, 150, 151)를 전환(switching)하기 위해 사용된다. 이러한 MUX를 전환하는 것은 무엇이 재밍(jamming)을 실행하는가에 달려있다. MUX의 조합된 출력(155)은 넥스트 어드레스 계산기(102)에 인가되고, 거기에서 그 출력은 자격 부여 입력으로서 사용되어 테스트 프로그램내의 분기에 영향을 준다.

<97> 이제 도 5를 살펴보면, 도 5에는 획득 스위프의 각 단계 동안 ECR2(32b)내에 저장될 실제 비교 데이터를 제공하는 낮은 레벨 회로(그 대부분은 채널마다의 기준상에 존재함)의 단순화된 블록도가 도시되어 있다. 이 도면의 왼편은 편입된 개선된 패스트 디코드를 갖는 메모리 테스터(MEMORY TESTER WITH ENHANCED PAST DECODE)내의 도 5의 서브세트이다. 따라서, 그에 관해 자세하게 설명하지는 않을 것이다. 간단한 요약만으로도 충분하다.

<98> DUT의 패드 또는 핀(109)를 살펴보자. 그것은 내부 SRAM(54)(회로(52)의 일부로서 도 2에도 도시되어 있음)로부터 포맷(140) 및 데이터(139)를 얻는 구동 포맷 제너레이터(107)에 의해 활성화된 레벨 시프터(108)로 구동될 수 있다. 핀(109)이 테스터에 의해 구동되고 있든지 아닌지간에 우리는 그 핀에서의 전압이 어떠한 동작이 수행되고 있을지라도 지정된 범위내에 있는지 알고 싶을 것이다. 이를 위해 핀(109)에 조합되고, 각각 CH VOH(171)(VOH의 채널마다의 인스턴스) 및 CH VOL(172)(VOL의 채널마다의 인스턴스)에 조합되는 두개의 비교기(107, 108)가 존재한다. 신호 CH VOH 및 CH VOL은 비교기가 비교하는 설정가능한 비교 전압이다. 비교기의 출력은 각각 수신 래치(119a, 119b)로 래치(latch)되고 그 후 그 출력은 각각 논리적 신호 YVOH 및 YVOL를 생성한다. 래칭은 표현된 샘플 타이밍 오프셋에 따라 스트로브 제너레이터(170)에 의해 판정된 DUT 사이클에서 한번에 실행되며, 이 모두에 대해 조금더 설명할 것이다.

<99> YVOH 및 YVOL은 포맷 신호(140), (기대되는) 데이터(139) 및 IVT 버스에 응답하는 F/P 에러 선택 레지스터(123)내의 정보에 따라 측정 입력이 올바른지를 평가하는 포맷된 수신기(124)에 인가된다. (그것은 VT 버스(89)의 칩간 확장(inter-chip extention)인데, 그 이유는 우리가 여기서 다루는 실리콘 조각-회로(52)-이 APG가 다루는 것과 상이하기 때문이다.) 어느 경우에도, 포맷된 수신기(하나의 퍼채널(per channel)이 존재함)는 기능적 에러 신호(126), 파라미터적 에러 신호(127) 및 비교 에러 신호(125)를 방출한다. 기능적 에러 및 파라미터적 에러는 각각의 래치(128, 129)에서 래치되고, 그 래치(128, 129)는 계속해서 더 처리될 수 있는 신호(176, 177)를 생성하며, 그 결과로서 도 2에서의 회로(52)로의 입력으로서 나타나는 신호 DFE0:3(103) 및 DPE0:3(104)가 얻어진다. 그것들이 의미하는 것은 본 기재의 범위를 벗어나는 것이다. 그것들은 멀티 DUT 동작이라고 불리는 것과 관련이 있다. 독자가 유의해야 하는 것은 비교 에러 신호(125)는 아마 ECR(Error Catch RAM)내의 "실제" 에러 데이터로서 통상적으로 저장되는 것이며, 또는 그것은 종류(데이터 분류 또는 포스트 디코드에서의 취급)에 대해 아마 제일 먼저 특징지어지고, 그 후 ECR내에 저장된 적합한 유형의 지시기라는 것이다.

<100> 어느 경우에도, 독자는 정상적인 동작에서 비교 에러(125)가 **포맷된 수신기(124)에 인가되는 포맷 신호(140)에 따라 생성될 때** 지나쳐버릴 필요가 있다는 것을 이해해야 한다. **그 정보는 (말하자면) "원래의" YVOH 및 YVOL과 동일할 수 없다.** 독자는 또한 그것이 획득 스위프내의 단계 동안 시스템의 잔여 부분으로 발송될 필요가 있는 정확히 "원래의" YVOH(단지 그것만)이다. 이러한 서로 다른 목적을 달성하는 것은 (채널마다(per channel)) MUX(173)의 기능이다. 그것은 퍼채널 비교 에러(125) 및 (채널마다) YVOH(175)를 둘다 수신하여, 신호 PIPED_TRIGGER(180)에 따라 단지 적합한 것을 전송한다. 즉, PIPED_TRIGGER가 FALSE일 때, 비교 에러 신호(125)가 선택되고, PIPED_TRIGGER가 TRUE일 때, 실제 (해석되지 않은) YVOH(175)가 선택된다.

<101> 여기서 신호 PIPED_TRIGGER(180)에 대해 살펴보아야 할 것이다. 그것은 ALU(24), 어드레스 맵퍼(29), 내부 테스트 메모리(87) 및/또는 데이터 MUX(35), ADDR. 비트 선택(37), 회로(40), 벡터 FIFO(45)를 통해 마이크로컨트롤러 시퀀서(19)로부터 최종적으로 타이밍/포맷팅 및 비교 회로(45)(도 5가 위치된 곳임)로 신호를 전달하는 파이프라인에 의해 유도된 지연고 동일한 양만큼 지연된다는 것을 제외하고, TRIGGERED(122)의 논리적 동등물이다. 이러한 파이프라인 및 그 지연으로 인해, 신호 TRIGGERED(122)(파이프라인의 선두에 있음)는 그와 관련된 전송 벡터가 실제로 DUT의 애플리케이션에 이용가능하기 전 적절한 시기에 발생한다. 도 5에는 PIPED_TRIGGER(180)에 의해 제어되는 3개의 MUX(164, 169, 173)가 존재한다. 파이프라인 지연의 효과를 소홀

히 하면, (즉, PIPED_TRIGGER 대신에 TRIGGERED를 사용함으로써) 그 MUX는 너무 빨리 변화될 것이며, 실제로(논리적인 의미로) 트리거의 발생에 선행하는 DUT 활동에 대한 동작의 스코프 모드를 불러일으킬 것이다. 그것은 여러 일을 상당히 망쳐놓을 것이다. 따라서, 신호 TRIGGERED(122)는 지연 메카니즘(181)에 의해 지연되어 PIPED_TRIGGER(180)이 된다. 지연 메카니즘(181)은 도시된 바와 같이 실제 개별 회로일 수 있고, 또는 파이프라인 그 자체내의 TRIGGERED에 대한 전용 경로일 수 있다.

<102> 이제 남은 것은 전압 임계치 VOL 및 VOH가 어떻게 생성되는지 및 샘플 타이밍 오프셋이 어떻게 달성되는지 설명하는 것이다. 그 마지막 두가지는 프로그램적으로 결정된 정상 값과 획득 스위프동안 여러 단계에 대해 요구되는 변화된 값 사이에서 빠르게 고정될 필요가 있다.

<103> 먼저 VOL을 처리하기로 하는데, 그 이유는 그것 자체는 관심의 대상이 되는 프로세스의 부분이 아니지만, 간단하며 유사한 출발점을 가지고 있기 때문이다. VOL은 링 버스(85) 상에서 전송되는 m개 비트의 디지털 값을 수신하기 위해 조합되는 VOL 레지스터(161)로부터 비롯된다. 그 m개 비트는 VOL DAC(162)에 인가되어, 비교기(118)에 인가되는 아날로그 임계치 전압 CH VOL(171)을 생성한다.

<104> CH VOL(172)의 생성은 그 VOH DAC(165)가 링 버스(85)에 의해 설정가능한 두개의 m비트 레지스터 사이에서 선택되는 MUX(164)의 m개 비트 출력에 의해 구동되는 것을 제외하고는 유사하다. 그러한 레지스터는 정상 모드 VOH 레지스터(163) 및 스코프 모드 레지스터(166)이다. MUX(164)는 MUX(173)의 경우와 같이, 신호 PIPED_TRIGGER(180)의 값에 따라 선택한다. 즉, PIPED_TRIGGER가 TRUE일 경우, 그것은 선택되는 스코프 모드 레지스터(166)의 m비트 값이고, 그렇지 않을 경우에는 레지스터(163)으로부터의 정상 값이다. 스코프 모드 레지스터(166)에서의 값은 감독 스코프 모드 소프트웨어에 의한 시기에 앞서 획득 스위프내의 단계 동안 VOH를 변화시키기 위한 임의의 원하는 전략과 일치하는 값으로 설정될 수 있다. 따라서, CH VOH 전압(172)의 값은 PIPED_TRIGGER(180)이 인가되면 그 정상 값으로부터 빠르게 변화될 수 있으며, PIPED_TRIGGER가 사라질 경우 마찬가지로 빠르게 정상 값으로 돌아온다.

<105> 마지막으로, 샘플 타이밍 오프셋의 정상 값과 획득 스위프에 요구되는 값 사이에서 유사하게 전환하는 문제를 다룰 필요가 있다. 그것은 링 버스(85)에 접속된 스코프 모드 샘플 타이밍 레지스터(168)의 출력과 정상 모드 '무언가(something)'(167, 후술함) 사이에서 선택하는 PIPED_TRIGGER(180)에 의해 제어되는 MUX(169)가 존재한다는 점에서 일반적으로 유사한 방식으로 행해질 것이다. 그 사이에 선택될 두가지는 각각 n개 비트의 크기를 가지며, 소정의 기준 신호에 관한 수신 래치(119a - b)의 스트로빙(strobing)에 대한 지연을 설명해주는데, 그 기준 신호를 시스템_DUT_사이클(178)이라고 부르며, 그것은 스트로브 제너레이터(170)에 인가된다. MUX(169)에 의해 선택된 지연 값(스코프 모드에 대한 레지스터(186)의 값, 그 밖에는 '무언가'(167)로부터의 값)이 또한 스트로브 제너레이터(170)에 인가된다. 스트로브 제너레이터(본질적으로 디지털하게 제어되는 지연 제너레이터)는 수신 래치(119a - b)에 인가되는 지연된 출력(179)를 방출한다. 그 출력은 현재의 목적상 MUX(169)로부터의 n개 비트 값의 양만큼 지연된 시스템_DUT_사이클(178)의 온셋으로 여겨질 수 있다. 그 n비트 값은 예컨대, 19.5피고초의 배수인 양으로 그 지연을 설명할 수 있다.

<106> 이제, 모호한 '무언가'(167)에 대해 살펴보기로 한다. 우선, 그것은 가장 간단한 실시예에서, 스코프 모드 샘플 타이밍 레지스터(168)와 같이, 링 버스(85)에 조합된 단지 n비트 레지스터일 수 있다. 그리고나서 그것을 정상 모드 샘플 타이밍 레지스터라고 부를 것이다. 이에 관한 문제는 그것이 작동하지 않는 것이 아니라(그것은 작동한다), 오히려 그것은 테스트 프로그램에게 하나의 샘플 타이밍 오프셋 자원을 제공하고, 그래서 프로그래머는 샘플 타이밍 오프셋이 변화될 때마다 많은 기타 작업을 해야 한다는 것이다. 정말 문제이다. 대신에 사전에 프로그램가능하고 테스트 프로그램이 실행되고 있을 때 그 사이에 쉽게 선택되는 선택가능한 '타임 세트'가 존재하는 것이 바람직하다. 이러한 타임 세트는 정상 모드 샘플 타이밍 참조 테이블(167)에 있으며, 즉, 그것들 중 16개가 있다. 그리고 그것들은 샘플 타이밍 오프셋 정보보다 더 많은 스터프를 보유할 수 있다. 간략함을 위해서 그러한 문제는 현재 목적과 관련이 없기 때문에 생략할 수 있다. 따라서, 어드레스가능한 16개의 요소 저장 메카니즘으로서 정상 모드 샘플 타이밍 참조 테이블(167)을 설명하는 것으로 충분하며, 각각의 그 어드레스가능한 위치는 소정의 번호 필드를 저장할 수 있으며, 그 중 하나는 n비트 값이며, 한번 어드레스된 상태로 그 출력에서의 다른 것들 사이에서 가장 최근에 어드레스된 위치에 현재 저장된 n비트 값을 제공한다. 설명한 바와 같이, 타임 세트의 개념은 통상적인 것으로 여겨지며, 요구되는 정상 모드 샘플 타이밍 참조 테이블(167)은 공지의 방법으로 표준 IC 라이브러리 성분으로부터 만들어질 수 있다.

발명의 효과

<107> 본 발명이 따르면, 실질적인 알고리즘적 내용을 갖는 테스트 프로그램의 실행에 사용되는 알고리즘적 패턴 제너레이터를 갖는 메모리 테스터의 스코프 모드에 트리거 신호를 효과적으로 제공할 수 있다.

도면의 간단한 설명

<1> 도 1은 본 발명에 따라 구성된 광범위하게 재구성가능하고 알고리즘적으로 구동되는 비휘발성 메모리 테스터의 블록도,

<2> 도 2는 도 1의 DUT 테스터의 단순화된 상세 블록도,

<3> 도 3은 스코프 모드 활동의 흐름도,

<4> 도 4는 트리거링 및 타겟 시퀀스의 포착 및 실행과 관련된 도 2의 블록도 부분의 보다 상세한 블록도,

<5> 도 5는 획득 스위프의 수행 동안 VOH 및 샘플 타이밍 오프셋의 값을 단계화하는 것과 관련된 도 2의 블록도 부분의 보다 상세한 블록도.

도면의 주요부분에 대한 부호의 설명

1 : 비휘발성 메모리 테스트 시스템

2 : 테스트 시스템 컨트롤러 3 : 시스템 버스

4a 내지 4z : 테스트 사이트 컨트롤러

5a 내지 5z : 사이트 테스트 버스 6a 내지 6z : DUT 테스터

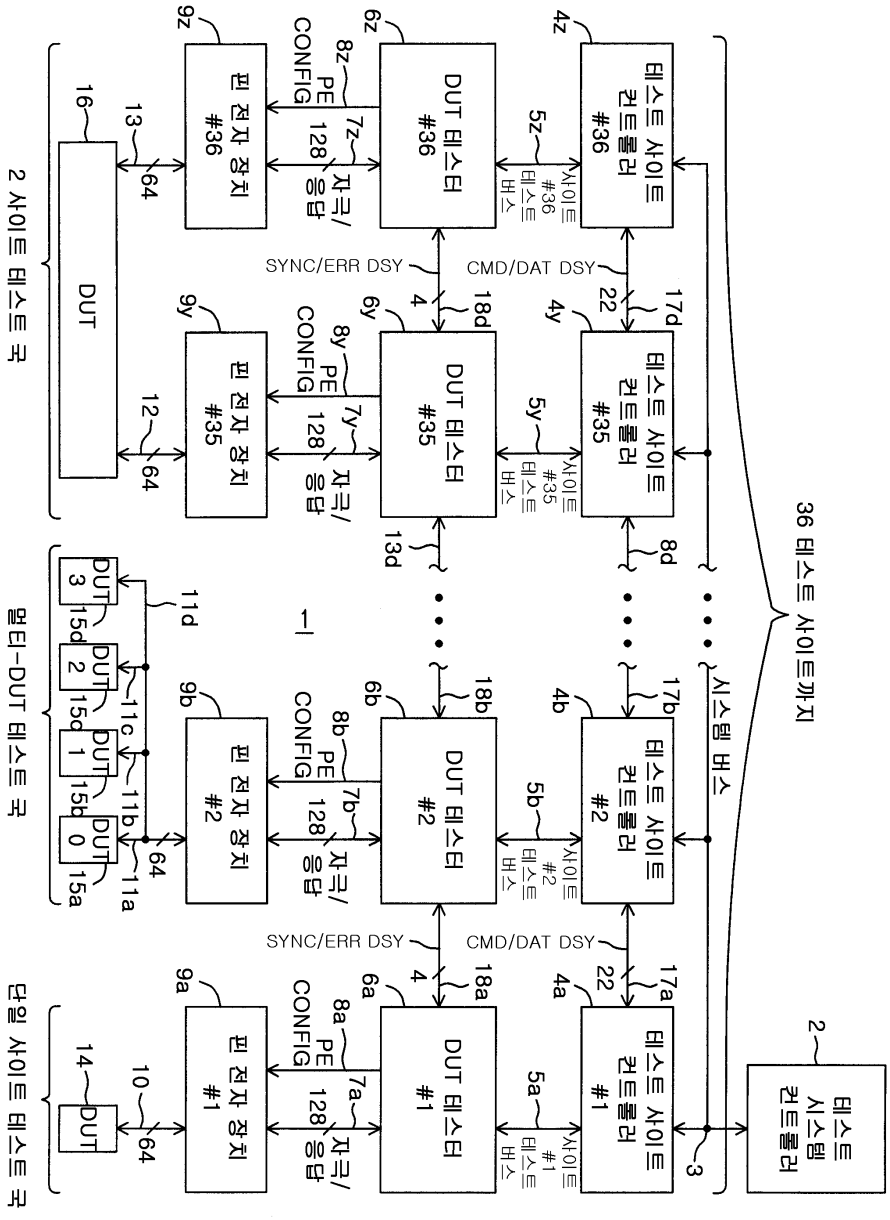
7a 내지 7z : 전송 & 수신 벡터

8a 내지 8z : 핀 전자 장치 구성 정보

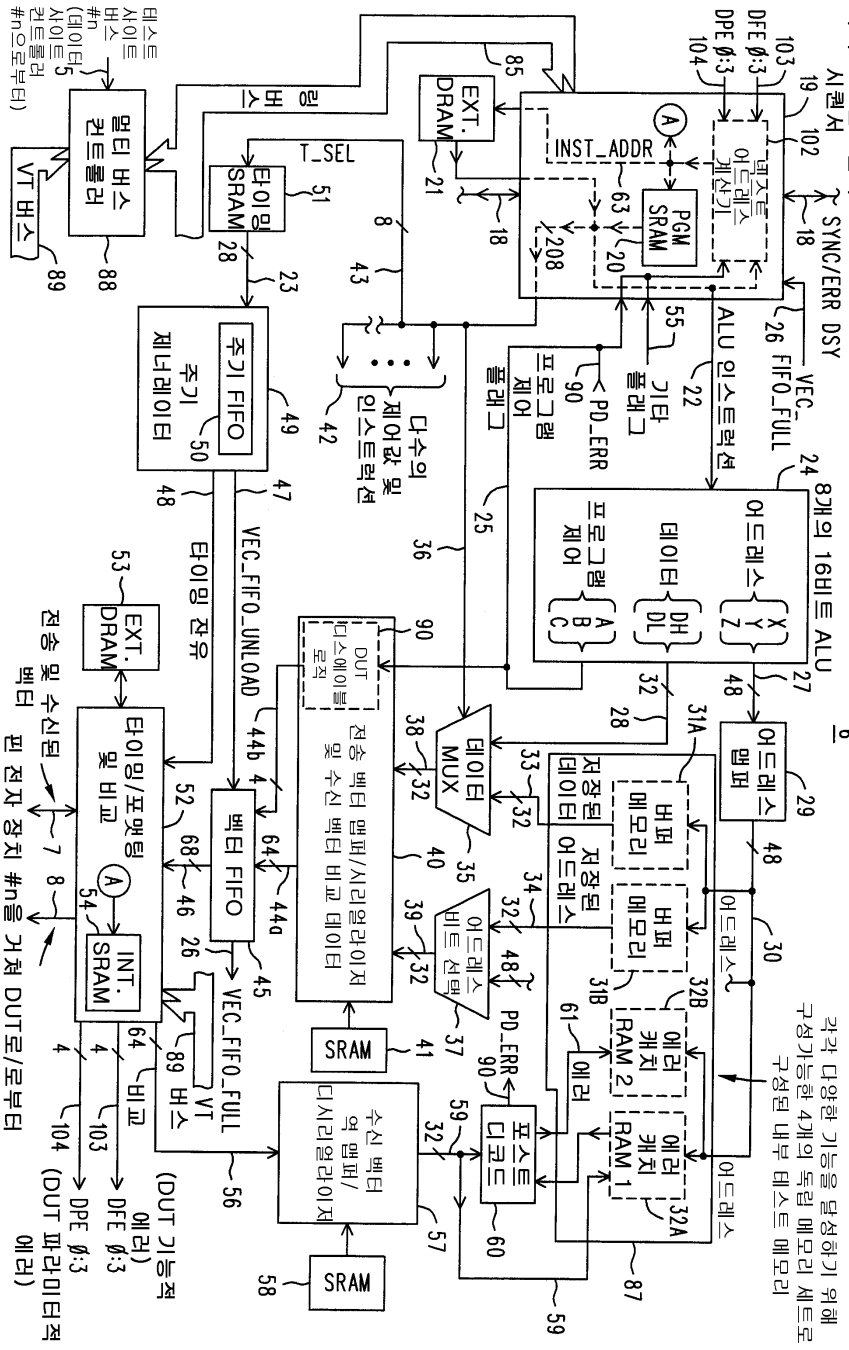
9a 내지 9z : 핀 전자 장치

도면

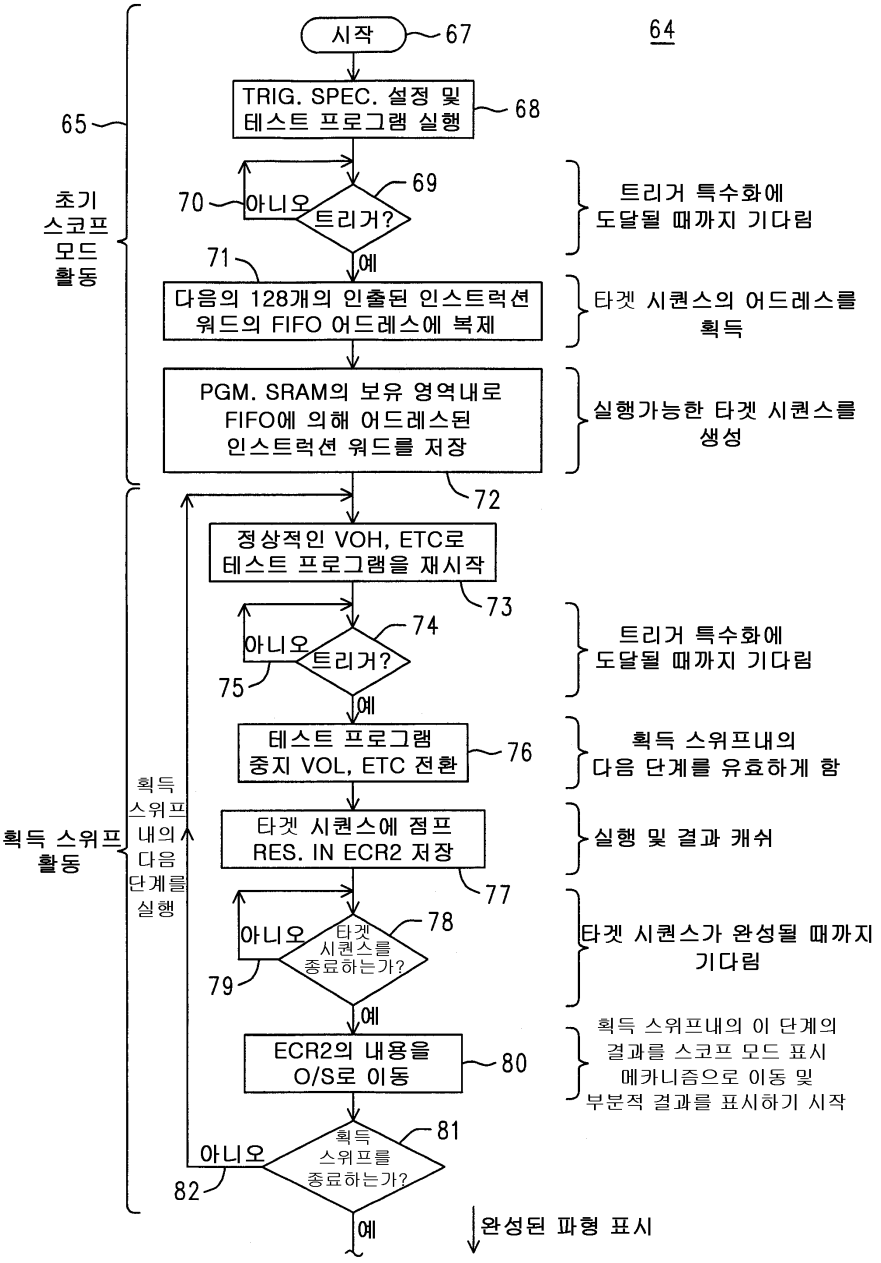
도면1


테스트 시스템 컨트롤러
2
3
36 테스트 사이트까지
시스템 버스
테스트 사이트 컨트롤러 #1
테스트 사이트 컨트롤러 #2
테스트 사이트 컨트롤러 #35
테스트 사이트 컨트롤러 #36
DUT 테스터 #1
DUT 테스터 #2
DUT 테스터 #35
DUT 테스터 #36
핀 전자 장치 #1
핀 전자 장치 #2
핀 전자 장치 #35
핀 전자 장치 #36
DUT
CMD/DAT DSY
SYNC/ERR DSY
PE CONFIG
자극/응답
단일 사이트 테스트 코
멀티-DUT 테스트 코
2 사이트 테스트 코

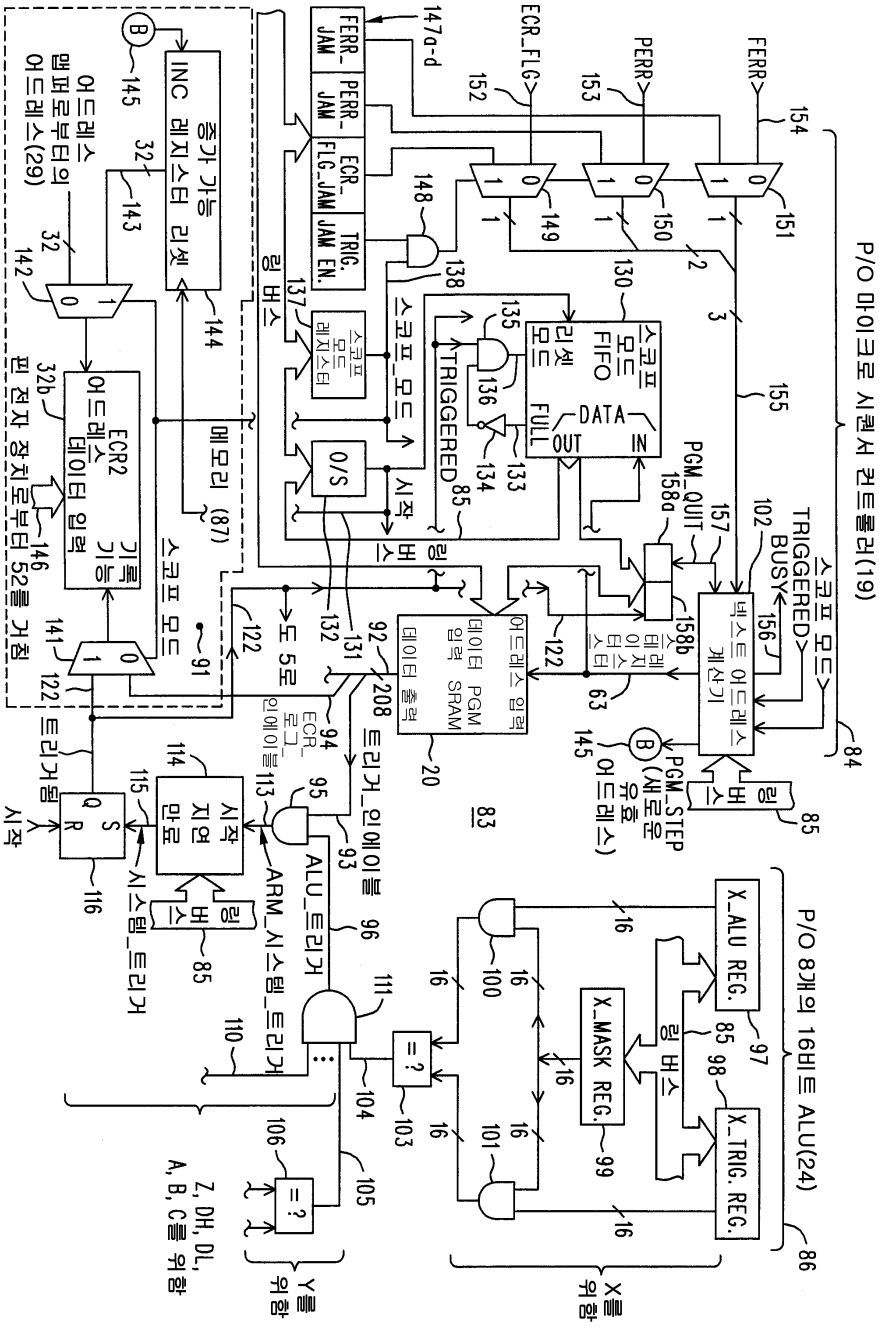
도면2


마이크로 컨트롤러 시퀀서
SYNC/ERR DSY
VEC_ FIFO_FULL
8개의 16비트 ALU
어드레스 맵퍼
넥스트 어드레스 계산기
INST_ADDR
PGM SRAM
ALU 인스트럭션
기타 플래그
PD_ERR
프로그램 제어 플래그
다수의 제어값 및 인스트럭션
EXT. DRAM
타이밍 SRAM
T_SEL
링 버스
멀티 버스 컨트롤러
VT 버스
테스트 사이트 버스 #n (데이터5 사이트 컨트롤러 #n으로부터)
주기 제너레이터
주기 FIFO
타이밍 잔유
VEC_FIFO_UNLOAD
데이터 MUX
DUT 디스에이블 로직
전송 벡터 맵퍼/시리얼라이저 및 수신 벡터 비교 데이터
SRAM
어드레스 비트 선택
PD_ERR
포스트 디코드
에러 캐치 RAM 2
에러 캐치 RAM 1
에러
버퍼 메모리
저장된 데이터
저장된 어드레스
수신 벡터 역 맵퍼/ 디시리얼라이저
타이밍/포맷팅 및 비교
INT. SRAM
벡터 FIFO
VEC_FIFO_FULL
VT 버스
비교
에러
DPE Ø:3
DFE Ø:3
(DUT 기능적 에러)
(DUT 파라미터적 에러)
전송 및 수신된 벡터 핀 전자 장치 #n을 거쳐 DUT로/로부터
각각 다양한 기능을 달성하기 위해 구성가능한 4개의 독립 메모리 세트로 구성된 내부 테스트 메모리
6

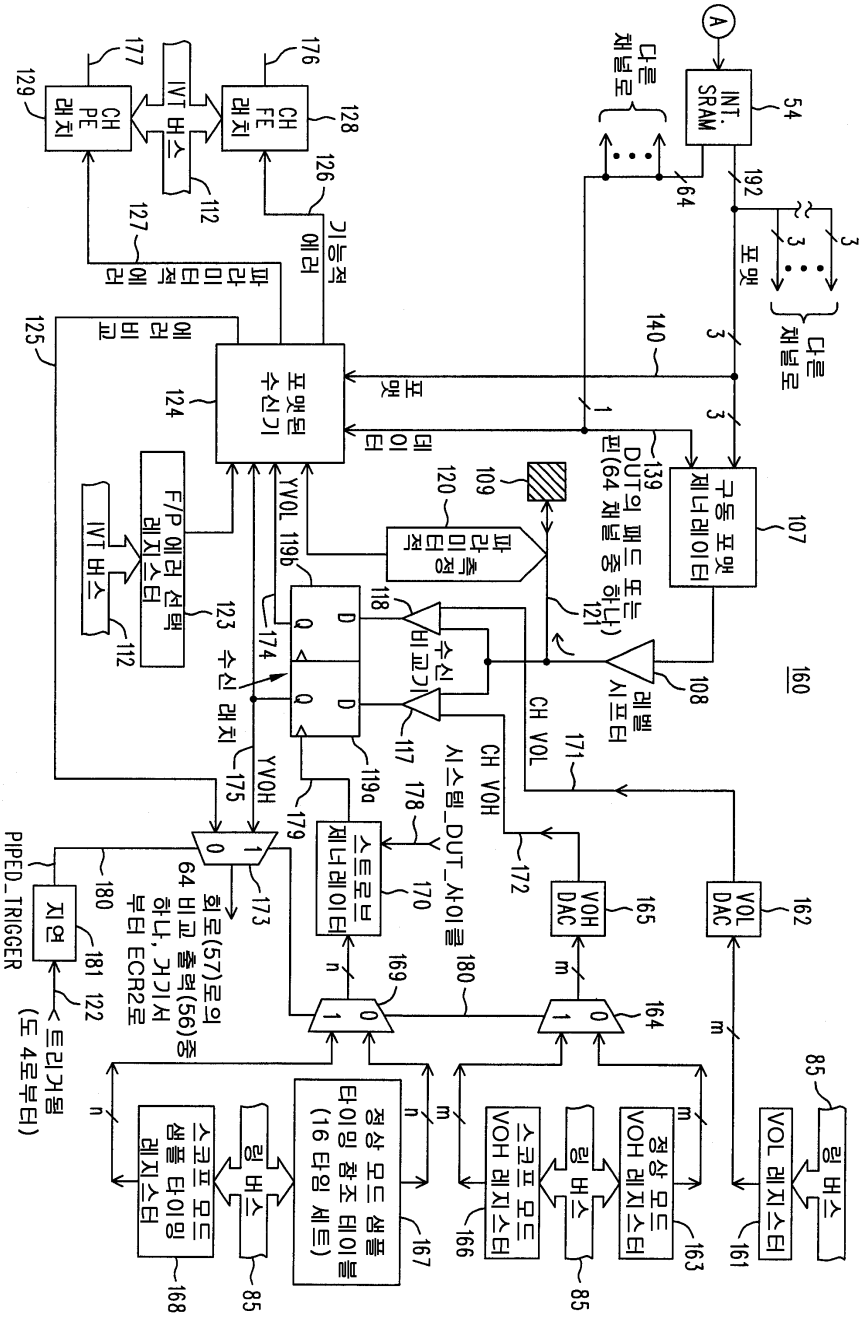
도면3


64
시작
67
TRIG. SPEC. 설정 및 테스트 프로그램 실행
68
65
초기 스코프 모드 활동
70
아니오
69
트리거?
예
71
다음의 128개의 인출된 인스트럭션 워드의 FIFO 어드레스에 복제
PGM. SRAM의 보유 영역내로 FIFO에 의해 어드레스된 인스트럭션 워드를 저장
72
트리거 특수화에 도달될 때까지 기다림
타겟 시퀀스의 어드레스를 획득
실행가능한 타겟 시퀀스를 생성
정상적인 VOH, ETC로 테스트 프로그램을 재시작
73
아니오
74
트리거?
75
예
테스트 프로그램 중지 VOL, ETC 전환
76
획득 스위프내의 다음 단계를 유효하게 함
획득 스위프 활동
획득 스위프내의 다음 단계를 실행
타겟 시퀀스에 점프 RES. IN ECR2 저장
77
실행 및 결과 캐쉬
아니오
78
타겟 시퀀스를 종료하는가?
79
예
타겟 시퀀스가 완성될 때까지 기다림
ECR2의 내용을 O/S로 이동
80
획득 스위프내의 이 단계의 결과를 스코프 모드 표시 메카니즘으로 이동 및 부분적 결과를 표시하기 시작
아니오
81
획득 스위프를 종료하는가?
82
예
완성된 파형 표시

도면4


P/O 마이크로 시퀀서 컨트롤러(19)
스코프 모드
TRIGGERED
BUSY
FERR
PERR
ECR_FLG
PGM_QUIT
PGM_STEP
(새로운 유효 어드레스)
넥스트 어드레스 계산기
레지스터
데이터 입력 PGM SRAM
어드레스 입력
데이터 출력
ECR 로그 인에이블
스코프 모드 FIFO
리셋 모드
DATA
FULL
OUT
IN
TRIGGERED
스코프_모드
시작
O/S
스코프 모드 레지스터
FERR_JAM
PERR_JAM
ECR_FLG_JAM
TRIG. JAM EN.
리 버스
메모리 (87)
스코프 모드
ECR2 어드레스 데이터 입력 기록 기능
핀 전자 장치로부터 52를 거침
어드레스 맵퍼로부터의 어드레스(29)
증가 가능 레지스터 리셋
INC
트리거됨
시작
시작 지연 만료
시스템_트리거
ARM_시스템_트리거
ALU_트리거
트리거_인에이블
도 5로
P/O 8개의 16비트 ALU(24)
X_ALU REG.
X_MASK REG.
X_TRIG. REG.
= ?
X를 위함
Y를 위함
Z, DH, DL, A, B, C를 위함

도면5


A
INT. SRAM
54
다른 채널로
64
192
3
3
3
다른 채널로
포맷
140
3
1
3
데이터
포맷
포맷된 수신기
124
에러 비교
125
파라미터적 에러
127
기능적 에러
126
CH FE 래치
176
128
IVT 버스
112
CH PE 래치
177
129
F/P 에러 선택 레지스터
123
IVT 버스
112
수신 래치
175
174
YVOL 119b
YVOH
179
119a
D
Q
D
Q
118
117
수신 비교기
파라미터적 미정
120
109
DUT의 패드 중 하나)
121
139
핀(64 채널
구동 포맷 제너레이터
107
레벨 시프터
108
160
CH VOL
171
CH VOH
172
시스템_DUT_사이클
180
스트로브 제너레이터
170
169
n
1
0
VOH DAC
165
m
1
0
164
VOL DAC
162
m
지연
180
181
122
PIPED_TRIGGER
(도 4로부터)
회로(57)로의 64 비교 출력(56)중 하나, 거기서 부터 ECR2로
0
1
173
n
스코프 모드 샘플 타이밍 레지스터
168
링 버스
85
n
정상 모드 샘플 타이밍 참조 테이블 (16 타임 세트)
167
m
스코프 모드 VOH 레지스터
166
링 버스
85
m
정상 모드 VOH 레지스터
163
m
VOL 레지스터
161
링 버스
85