(19)대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) 。Int. Cl.[7]
G06F 9/455

(45) 공고일자 2005년03월16일
(11) 등록번호 10-0476785
(24) 등록일자 2005년03월07일

(21) 출원번호	10-2001-7015516	(65) 공개번호	10-2002-0022061
(22) 출원일자	2001년12월03일	(43) 공개일자	2002년03월23일
번역문 제출일자	2001년12월03일		
(86) 국제출원번호	PCT/US2000/014257	(87) 국제공개번호	WO 2001/06421
국제출원일자	2000년05월24일	국제공개일자	2001년01월25일

(81) 지정국

국내특허 : 아르메니아, 오스트리아, 오스트레일리아, 바르바도스, 불가리아, 브라질, 벨라루스, 캐나다, 스위스, 중국, 체코, 독일, 덴마크, 에스토니아, 스페인, 핀란드, 영국, 그루지야, 헝가리, 이스라엘, 아이슬랜드, 일본, 케냐, 키르키즈스탐, 북한, 대한민국, 카자흐스탄, 스리랑카, 리베이라, 리투아니아, 룩셈부르크, 라트비아, 몰도바, 마다가스카르, 몽고, 말라위, 멕시코, 노르웨이, 뉴질랜드, 폴란드, 포르투칼, 루마니아, 러시아, 수단, 스웨덴, 싱가포르, 슬로베니아, 슬로바키아, 타지키스탄, 투르크맨, 트리니아드토바고, 우크라이나, 우간다, 우즈베키스탄, 베트남,

AP ARIPO특허 : 케냐, 레소토, 말라위, 수단, 스와질랜드, 우간다, 시에라리온, 가나, 감비아, 짐바브웨,

EA 유라시아특허 : 아르메니아, 아제르바이잔, 벨라루스, 키르키즈스탐, 카자흐스탄, 몰도바, 러시아, 타지키스탄, 투르크맨,

EP 유럽특허 : 오스트리아, 벨기에, 스위스, 사이프러스, 독일, 덴마크, 스페인, 핀란드, 프랑스, 영국, 그리스, 아일랜드, 이탈리아, 룩셈부르크, 모나코, 네덜란드, 포르투칼, 스웨덴,

OA OAPI특허 : 부르키나파소, 베닌, 중앙아프리카, 콩고, 코트디브와르, 카메룬, 가봉, 기니, 말리, 모리타니, 니제르, 세네갈, 차드, 토고, 기니 비사우, 적도 기니,

(30) 우선권주장 09/354,955 1999년07월16일 미국(US)

(73) 특허권자
아트멜 코포레이숀
미국 캘리포니아주 95131 샌호제 오챠드 파크웨이 2325

(72) 발명자
메이슨,마틴토마스
미국,캘리포니아95128,산호세,체리스톤드라이브2249

맥커넬,데이빗앤드류
미국,캘리포니아94502,알라메다,피지레인3208

다사리,아지스쿠마르벤카타
미국,캘리포니아95051,산타클라라,버킹엄드라이브100,아파트먼트 236

(74) 대리인
강명구

심사관 : 강갑연

(54) 필드 프로그래머블 시스템 레벨 소자 구현용 소프트웨어 툴

명세서

기술분야

발명은 필드 프로그래머블 시스템 레벨 집적 회로(FPSLIC)에 관한 것이다. 특히 발명은, FPSLIC 소자의 하드웨어 시뮬레이션 및 소프트웨어 시뮬레이션을 함께 확인하기 위한 시스템 및 방법에 관한 것이다.

배경기술

시스템 레벨 통합(SLI)은 전자 장치 구현을 위해 선호되는 방법이 되어가고 있다. 시스템 레벨 집적 회로(IC)에 모든 시스템 기능을 통합시키는 것은 성능을 향상시키고, 전력 소모를 줄이며, 단위생산 비용을 절감하고, 크기를 소형화 시킨다. 이들은 전화통신, 멀티미디어, 네트워크 응용 등에 중요한 잇점을 보인다.

"시스템"은 세 개의 주된 블록, 즉, 프로세서, 메모리, 로직으로 이루어진다. 일반적으로, 프로세서는 제어 흐름 로직에 사용되고, 메모리는 프로그램 및 데이터 저장에 사용되며, 로직은 데이터경로 로직에 사용된다. 진실한 시스템-레벨 해법은 이 세가지 요소를 모두 포함하여야 한다.

ASIC 해법

지금까지, 시스템 레벨 통합은 셀 기반, 또는 마스킹된 전용 집적 회로(ASIC)로 구현되었다. 왜냐하면, 이들이 시스템 레벨 설계를 좌우할 충분한 밀도를 가진 유일한 해법들이기 때문이다. 불행하게도, ASIC는 비-순환 공정(NRE) 비용이 높고, 리드 시간이 길며, 최소 주문양이 상당하다. 그 결과, 시스템 레벨 ASIC 구현은 상대적으로 긴 제품 수명 주기를 가지는 최소 부피의 설계에만 접근가능하다. 시스템 레벨 ASIC 소자의 최소 부피 요구사항은 매년 설계당 $500K를 넘는다. 제품 수명 주기가 짧고 부피가 작거나 중간 정도이며 시장 압력에 대한 시간이 짧고 급속하게 전개되는 표준을 따른 설계는 여유있는 개발 주기를 제공할 수 없고, ASIC 해법과 관련된 위험 및 높은 NRE 차지(charge)를 제공할 수 없다.

ASIC 해법에 대해 부피/비용 균형이 충족될 때조차, 오류 교정이나 개선을 위한 설계상의 변화는 개발자에게 거의 무용한 부분의 다량의 목록과, 또다른 길이의 ASIC 설계 주기를 남긴다. 이는 무선통신, 네트워킹, 그리고 멀티미디어에서와 같이 급속하게 전개되는 설계에서 특히 의심스럽다. 이들 및 이와 다른 설계의 경우에, 프로그래밍가능한 해법이 선호된다. 왜냐하면, 이 설계는 개발 중에서와 사용 환경에서 모두 뜻대로 변경할 수 있기 때문이다. ASIC 해법은 이 설계들에 대한 옵션이 아니다.

이렇게 급속하게 전개되는 설계의 상당수는 프로그래밍가능한 로직, 구분된 표준 제품(마이크로제어기, 메모리), 전용 표준 제품(ASSP)의 조합으로 구현된다. 그 예로, T1 인터페이스, ATM, 10/100 PHY, 비디오/오디오 코덱 등이 있다. 이 접근법이 설계의 급속한 전개에 유연성을 제공하지만, 단일 시스템 레벨 집적 회로의 성능, 출력, 공간, 신뢰성 등의 장점을 제공하지는 못한다. 단일-칩의, 프로그래밍가능한 해법은 선호되는 대안이다.

프로그래밍가능한 시스템 레벨 통합에 대한 다중접근

필드 프로그래머블 게이트 어레이(Field Programmable Gate Array)(FPGA)와 그외 다른 IC의 판매자는 프로그래머블 시스템 레벨 통합을 제공하기 위한 다방면의 접근법을 개발하였다. 이들은 "퓨어 플레이(pure play)" 고밀도 FPGA와, FPGA 및 고정 로직 기능을 조합한 하이브리드 소자를 포함한다.

오늘날 프로그래머블 SLI를 얻는 가장 폭넓게 증진된 수단은 Xilinx의 Virtex와 Altera의 APEX FPGA 제품군으로서, 수백만 게이트를 자랑한다. 이 게이트 숫자가 과장된 것일 수 있음을 산업 분석가들은 알고 있지만, 이 소자들은 시스템 레벨 통합의 설계를 지원하기에 충분히 크다. 그렇지 않을 경우 이 설계들은 마스킹식, 또는 셀 기반의 ASIC로 만들어질 것이다. FPGA는 밀도와 가격 측면에서 마스킹-ASIC와 이제 경쟁한다. 고밀도 FPGA는 시스템 레벨 통합에 대한 프로그래머블 단일 칩 해법으로 제시되고 있다. 대형 FPGA의 프로그래머빌러티가 매우 매력적이지만, 단점도 또한 가지고 있다.

깊은 서브-미크론 처리 기술이 저/중 밀도의 FPGA 가격을 절감시켜서 여러 경우에 FPGA 가격이 ASIC 가격과 동등하게 하지만, 높은 게이트 카운트 소자들은 매우 고가이다. 가령, Xilinx의 백만 게이트 Vertex XCV1000 소자는 개당 $4,298.00에 팔리고 있다. 이 소자들의 높은 가격은 소량의 초고가 제품의 제품 가동 및 ASIC 프로토타이핑에 이용되는 것을 제한한다. 비교가능한 밀도의 마스킹된 ASIC가 $50이라고 생각할 때, 이 대형 FPGA는 대부분의 부피 설계/응용에서 벗어난다.

FPGA 소자들이 ASIC 개발 사이클을 반으로 자를 수 있지만, 대형 FPGA의 복잡도는 주목할만한 설계와, 시스템 레벨 설계에 대한 개발 과정을 지시한다. 오늘날 "시간에 따른 시장(time-to-market)"은 제품의 성공 및 실패간의 차이이다. 수백만 게이트의 FPGA 로직을 설계하는 것은 상당한 시간을 소요한다. 설계 주기 단축을 위해 지적 재산권(IP) 코어가 자주 사용된다. 그러나, 소프트웨어 IP를 공급하는 판매자를 설계 내로 통합하는 것은 그 자체로 귀찮은 일이고 시간을 잡아먹는 일이다.

시뮬레이션은 대형 FPGA가 지닌 또다른 문제점이다. HDL 시뮬레이션은 대형 설계의 시뮬레이션에서 엄청나게 느리며, 특히 복합 소프트 IP 코어를 이용하는 설계의 시뮬레이션에서 특히나 느리다. 백만 게이트 FPGA 설계를 시뮬레이션하는 것은 매우 오래 걸리기 때문에, 많은 설계자들이 바람직한 것보다는 덜 완전하게 시뮬레이션을 실행한다. 그 결과, 이 설계들은 디버그 주기를 확장시키는 발견되지 않은 버그를 가지기 쉽고, 제품 소개를 지연시킨다.

이 문제는 대형 FPGA 설계에 마이크로제어기 소프트 IP 코어가 사용될 때 더욱 커진다. 기존 MCU 설계법은 대형 FPGA를 기반으로 한 설계 흐름에서 가용하지 않다. 일반적으로, 마이크로제어기 설계자는 마이크로코드 디버그 과정에 사용될 코드 개발 및 디버그 툴을 가진다. 이 툴은 대형 FPGA에 사용되는 소프트 IP 코어에서 가용하지 않은 경우가 자주 있어서, 코드 개발 및 디버그 과정이 의심스럽다. 더욱이, 프로세서 코어에 대해 가용한 코드 개발 및 디버그 툴의 결여 때문에, 이 설계의 마이크로제어기 부분의 통합 및 디버그는 매우 어렵다. 시스템-레벨 FPGA 설계에서 적절한 분석을 위해 유사한 독립변수들이 쉽게 만들어질 수 있다.

시스템 레벨 FPGA 설계시 복잡도에 대한 한가지 해법은 "드롭-인(drop-in)" 소프트 지적 재산권 코어를 이용하는 것이다. 메모리, 로직, 그리고 제한된 수의 프로세서 IP 코어는 제 3 자인 판매자로부터 구매할 수 있고, 대형 FPGA 내로 떨구어질 수 있다(drop-in). 그러나, 소프트 IP 코어는 고가이고, 설계시 통합하기 어려우며, 비효율적인 실리콘인 경향이 있다. 서로 다른 3자 판매자로부터 IP 코어를 통합하는 어려움은 제품 개발 주기를 크게 늘릴 수 있다. 가령, 프랑스 Dolphin에 의해 공급되는 Xilinx Virtex의 8051 코어는 면허에만 $10,000이 지출되며, XCV1000의 16.4%나 1010CLB를 이용한다. 각 XCV1000에 대한 $4,298의 가격 목록에서, 8051 코어의 실리콘 가격은 코어 자체의 가격을 제외하여 $704이다.

"Flip(플립)-8051 코어"는 하부 성능 옵션을 포함하는 마이크로프로세서와, 타이머 및 시리얼 인터페이스와 같은 주변 장치를 포함하는 마이크로제어기 구조의 제품군의 심장을 형성한다. Virtex FPGA에 대한 EDIF 포맷의 경우 $10,000에서부터 가격이 시작된다. 다른 설계 파일 포맷도 가능하다. VHDL 소스-코드 버전이 추가 비용에서 테스트 벤치로 가용하다.

여러 코어간에 인터페이스를 구축하고 타이밍 문제를 교정함과 관련하여 추가적인 설계 문제점이 존재한다. 전형적인 설계 프로젝트의 첫 번째 반은 통합 및 검사 단계에 소모되고, 이는 실제로 설계 주기의 전면 단부로부터 오류 누적을 교정하는 수련시간이 된다. 이 문제들은 다중-사이트, 다중-엔지니어 개발 티임이 대형 FPGA-기반 시스템에서 일할 때 확대된다. 오류는 다시 설계 및 분할 단계까지 줄곧 다다르는 경우가 잦고, 이때 하드웨어/소프트웨어 인터페이스의 모호성이 삽입되고, 그후 하드웨어/소프트웨어 구현 단계 중 증폭된다. 소프트웨어 고정이 최종 제품에 절충된 성능을 유발할 수 있더라도, 이 오류의 치료는 ASIC의 긴 리드 시간이나 고가로 인해, 또는 대형 FPGA의 턴(turn)으로 인해 소프트웨어로 강제된다.

FPGA가 데이터-경로 기능을 구현하기 위한 효율적 수단을 나타내지만, 제어 로직이 CPLD나 마이크로제어기 구조에 더 적합하다. 제어 로직의 FPGA 구현은 실리콘 효율적이지 못하다. 이는 Altera가 차세대 대형 FPGA에 구현하는 구조적 변화에 의해 나타난다. 이 대형 소자 내 로직의 CPLD 블록 삽입은 현재의 균일한 대형 FPGA 해법에서 제어 흐름 로직을 구현하는 데 약하다는 것을 명백하게 보여준다. CPLD 구조의 삽입은 Altera APEX 구조의 제어 흐름을 돕지만, CPLD 해법은 제어 흐름 구현과 결정 과정에서 마이크로제어기보다 덜 효율적이다.

가령, 메일 소팅 시스템은 고속 카메라를 이용하여 각 메일에서 시각적 데이터를 획득하여야 하고, 데이터를 아날로그에서 디지털로 변환하여야 하며, 어드레스 위치 및 방향을 식별하도록 사전처리를 하여야하고, 어드레스 해독을 위해 어드레스 화소를 처리하여야하며, 소팅 머신에 의해 판독될 수 있는 기계 판독 코드를 발생시켜야 한다. 이미지에서 어드레스를 식별하는 것은 엄청난 양의 데이터 처리를 요구한다. 왜냐하면 각각의 화소가 이를 둘러싸는 모든 화소에 비교되어야 하기 때문이다. 다수의 이미지가 서로 관련없을 가능성이 높기 때문에, 전체 이미지를 프로세서로 처리하는 것은 비효율적이고 느리다. FPGA 기반의 DSP 알고리즘은 스트리트 어드레스의 일부가 되지 않는 이미지의 부분을 필터링하여 제거시키는 데 사용된다. 이 과정은 대부분의 이미지를 제거하여, 처리되는 문제를 관리가능한 크기로 절단한다. FPGA는 이러한 종류의 작동에 이상적이며, DSP 프로세서보다 훨씬 빠르게 이러한 작동을 실행할 수 있다.

다른 한편으로, 프로세서는 FPGA보다 훨씬 효율적으로 이미지 추출, 디-스쿠잉(de-skewing), 회전, 데이터 해역에 대한 알고리즘을 실행한다. FPGA는 다수의 복합 결정이 연관되어있기 때문에 이 작업에 부적절하다. 제어 흐름 작업을 실행하기에는 이는 너무 느리다. FPGA 코어 셀을 이용하여 앞서 제시한 작업을 할 수 있는 프로세서를 구현하는 것은 매우 어렵고 실리콘 비효율적이며, 따라서 매우 고가이다. 모든 시스템이 데이터경로와 제어 흐름 로직을 내장하기 때문에, 범용 FPGA의 시스템 레벨 통합은 효율적일 수 없다. 간단히 말해서 범용 FPGA는 해법이 아니다.

전력 소모는 세가지 성분, 즉, 정적 전력 소모, 동적 전력 소모, 입력/출력이나 시스템 전력 소모를 가진다. 정적 전력 소모는 소자 내 트랜지스터 수의 함수이고 FPGA의 크기와 함께 증가한다. 그러나, 조심스런 설계로, 대형 FPGA도 매우 낮은 정적 전력 소모를 가질 수 있다. 전력 소모의 제 2 소스인 동적 전력 소모는 I/O 구조에서 소모된다. 출력이 한 로직 상태에서 다른 로직 상태로 변경될 때마다 상당한 출력이 소모된다. 인쇄 회로 보드(PCB) 상의 축전적 로딩이 이 전력 소모에 대한 이유이다. 통합을 통해 시스템 부품의 수를 감소시키는 것은 이러한 측면의 시스템 전력 소모를 크게 감소시킨다. 대부분의 대형 FPGA가 고대역폭 마이크로제어기 버스에 연결하여야하기 때문에, 이 인터페이스를 통해 상당한 출력이 소모된다.

전력 소모의 세 번째 성분은 동적 전력 소모이다. 설계 구현 및 내부 클록 분포 트리에 대한 다수의 코어 셀 조합은 전력 소모에 크게 기여한다. 따라서, SLI 설계에 사용되어야하는 대형 FPGA는 이에 비례하여 보다 큰 출력을 이끌어낸다. 대형 FPGA에서 프로그래머블 SLI는 상당량의 전력을 소모하기 쉽다.

앞서의 문제점 때문에, FPGA의 밀도를 단지 증가시키는 것은 시스템 레벨 통합으로 프로그래머빌러티(programmability)를 얻는 가장 실용적 해법이 아니다.

최근에, 한단의 패블리스 IC 스타트-업(fabless IC start-up)이 하드-와이어드 마이크로프로세서 코어로 프로그래머블 로직의 블록을 통합하는 하이브리드 소자를 개발하였다. 이 소자의 삽입은 시스템 구조의 SLI 수요를 해결하는 제품에 대한 필요성을 나타낸다.

이 소자들은 범용 SLI 카테고리로 분류하기에는 너무나 특화되어 있다. 가령, Triscend의 8052 EV5 마이크로제어기가 5,000-30,000 게이트를 가진 온-칩 FPGA로 프로그래밍되는 주변장치를 가진 단일-칩 MCU에 대한 대체물로 위치한다. 이 소자들은 기존 MCU 판매자가 파생물 생성을 고려하는 것을 필요로하는 100,000 유닛을 주문할 필요없이 구현된 시스템 설계자들에게 기존 MCU 파생물을 제공한다.

또다른 실리콘 밸리 시동은 "재설정가능한 네트워크 프로세서"라고 알려진 Chameleon Systems이다. 제품 세부사항이 모두 공개되지는 않았으나, 모든 지적사항은 전용 무선통신/네트워킹 장치를 처리하는 특화된 제품을 가리킨다.

Chameleon은 데이터 네트워킹 및 무선통신 시장의 수렴에 우세한 스위치 패브릭과 물리적 인터페이스 사이에서 복합 통신 처리 요구사항에 초점을 맞출 것이다.

특화된 FPGA 기반 소자, 즉 Lucent Technologies와 Quicklogic은 PCI와 같은 특화 기능을 포함하는 FPGA를 발표하였다. 그러나, 어떤 회사도 프로세서, 메모리, 데이터경로 로직을 갖춘 진정한 시스템 레벨 제품을 소개하지 못하고 있다.

어떤 앞서의 해법도 시스템 레벨 설계법을 지원하는 설계 툴을 가진 진정한 프로그래머블 시스템 레벨 통합, 즉 프로그래머블 로직, 마이크로제어기, 메모리를 제공하지 못한다.

발명의 상세한 설명

AVR, ARM, 8051과같은 마이크로프로세서, RAM, ROM, EEPROM과 같은 구현된 메모리 블록, SPI, USB, PCI, I^2C와 같은 인터페이스 회로, AT40K와 같은 FPSLIC 소자 등을 포함하는 복합 블록을 내장한 FPSLIC 소자를 프로그래머블 로직 사용자가 손쉽게 설계할 수 있도록 하는 것이 발명의 일반적 목적이다.

발명의 또다른 목적은 설계 및 방법 흐름 관리자로 작용하는 소프트웨어 툴이다.

발명의 추가적인 목적은 설계 관리자에서 선택된 소자를 바탕으로 여러 CAE 개발 툴 변화를 통한 설계 흐름을 가지는 소프트웨어 툴이다.

본 발명에 따라, 필드-프로그래머블-게이트 어레이(FPGA)의 하드웨어 시뮬레이션 프로그램드 로직과 필드-프로그래머블-게이트 어레이(FPGA)의 소프트웨어 시뮬레이션 프로그램 코드를 동시확인하는 방법이 제공된다.

FPSLIC 소자는 필드 프로그래머블 게이트 어레이(FPGA) 코어, 마이크로제어기, 메모리를 포함한다. FPGA 코어는 가령, SRAM 기반의 FPGA 8-면 로직 셀 구조일 수 있는 다수의 게이트를 포함한다. 상기 로직 셀 구조는 버스 리소스에 충격을 가하지 않으면서 복합 기능을 실행한다.

다수의 게이트는 외부 NVM, 배치 메모리, 또는 마이크로제어기에 의해 프로그래밍가능하다. 가령, 전용 프로그램은 프로그램에 대한 FPSLIC 소자 FPGA 프로그래머블 로직을 설정하기 위해 마이크로제어기의 프로그램을 필요로 한다. 여러 프로그램에서, FPGA의 로직은 전원을 켤 때 구동된다.

응용프로그램용 소프트웨어는 FPSLIC 메모리에 저장될 수 있다. 메모리는 동적 할당 프로그램 메모리, 고정 데이터 메모리, 그리고 메모리 제어기를 포함한다. 동적 할당 프로그램 메모리는 한 예로, 프로그램 명령 저장용 20 나노초 SRAM의 32킬로바이트(32K X 8) 블록일 수 있다. 명령 저장용으로 32킬로바이트 전부가 필요하지 않을 경우, 메모리는 추가 데이터 메모리 저장공간 제공을 위해 블록으로 설계 개발 중 파티션을 나눌 수 있다. 부가적으로, 고정 데이터 메모리는 동적 할당 프로그램 메모리로부터 파티션을 추가함으로서 증가될 수 있다.

하드웨어 승산 가속기는 복합 디지털 신호 프로세서(DSP) 연산을 마이크로제어기가 신속하고 효율적으로 실행하게 한다.

발명의 추가적인 목적 및 장점은 부분적으로 아래에 이어지는 설명에서 설명될 것이고, 부분적으로는 아래의 설명으로부터 명백할 것이다.

도면의 간단한 설명

도 1A와 B는 FPSLIC 소자의 블록도표.

도 2는 하드웨어 및 소프트웨어 흐름, 시뮬레이션의 동시확인 도면.

도 3은 동시 확인을 포함한 하드웨어 (FPGA) 흐름 및 시뮬레이션(마이크로제어기)과 소프트웨어 흐름 및 시뮬레이션의 순서도.

실시예

필드 프로그래머블 게이트 어레이(FPGA)에 대한 시간에 따라 증가하는 수요는 여러 구분된 '점' 컴퓨터 보조 작업(CAE:Computer Aided Engineering) 툴을 관리하고 밀집식으로 통합하는 소프트웨어 툴을 필요하게 한다. CAE 툴은 FPSLIC 소자에 대한 코드를 개발하기 위해 설계자에 의해 광범위하게 사용될 수 있다.

본 발명은 칩상의 시스템을 생성하는 실리콘 효율적 수단을 제공하는 전용 기능을 갖춘 시스템 레벨 집적 회로(IC)의 군을 개발하기 위해 프로그래머빌러티를 시스템 레벨 통합과 조합한다. 필드 프로그래머블 시스템 레벨 IC(FPSLIC)는 예를 들어, 데이터 경로 로직용으로 ATMEL AT40K FPGA, 제어 로직용으로 RISC-기반 AVR 마이크로제어기, 하드웨어 승산기, MCU 주변 장치, 그리고 36킬로바이트 SRAM를 포함한다. FPSLIC 구조는 네트워킹, 무선통신, 멀티미디어, 오디오, 휴대용, 산업 제어 분야의 구현에 이상적이다.

도 1A와 1B에 도시되는 바와 같이, 필드 프로그래머블 시스템 레벨 집적 회로(FPSLIC 소자)(20)는 필드 프로그래머블 게이트 어레이(FPGA) 코어(21), 마이크로제어기(22), 메모리를 포함한다. FPGA 코어는 SRAM 기반 AT40K FPGA 8-면 로직 셀 구조(75)로 도시되는 다수의 게이트를 포함한다. 로직 셀 구조(75)는 버스 리소스에 영향을 미치지 않으면서 복합 기능을 실행한다. 구분된 자유-RAM 10 나노초 이중 포트 SRAM 블록(74)이 각 4X4 셀 섹터의 코너에 위치한다. 이 SRAM 블록들을 어레이 전체에 위치시키는 것은 필요한 곳에 메모리를 놓게하고, 고성능 선입선출(FIFO) 설계를 지원한다.

다수의 게이트는 외부 NVM, 배치 메모리, 또는 마이크로제어기(22)에 의해 프로그래밍가능하다. 가령, 전용 프로그램은 프로그램용 FPSLIC 소자 FPGA 프로그래머블 로직(21)을 설정하기 위해 마이크로제어기(22) 내에 프로그램을 필요로한다. 여러 응용 프로그램에서, FPGA의 로직은 전원을 켤 때 로딩된다. 응용 프로그램용 소프트웨어는 마이크로제어기(22)의 메모리에 저장될 수 있다. 연산 시에, 마이크로제어기(22)는 응용프로그램에 대한 FPSLIC 프로그래머블 로직(21)의 게이트로 설정된다.

메모리는 동적 할당 프로그램 메모리(23), 고정 데이터 메모리(24), 메모리 제어기(25)를 포함한다. 동적 할당 프로그램 메모리(23)는 예를 들어, 프로그램 명령 저장을 위한 20 나노초 SRAM의 32킬로바이트(32K X 8) 블록일 수 있다. 32 킬로바이트 전부가 필요하지 않을 경우, 프로그램 메모리는 추가 데이터 메모리 저장 공간 제공을 위해 예를 들어 8개의 4킬로바이트 블록으로 설계 개발 중 파티션을 나눌 수 있다. 추가적으로, 고정-데이터 메모리(24)는 동적 할당 프로그램 메모리(23)로부터 파티션(81)을 추가함으로서 증가될 수 있다.

하드웨어-승산 가속기(29)는 복합 디지털 신호 프로세서(DSP) 연산을 마이크로제어기(22)가 보다 신속하고 용이하게 실행할 수 있게 한다.

FPSLIC 해법은 독자적인 고성능 FPGA 구조와 여러 마이크로제어기 코어 기술로부터 생긴다. FPSLIC는 상기 기능을 조합하여 실리콘 효율적이고 비용 효과적인 시스템 레벨 장치를 만들도록 셀-기반 및 마스킹 ASIC와 전용 표준 제품(ASSP) 개발, 기술, 의견을 이용한다.

AVR 기반 FPSLIC 제품은 Atmel의 AT40K 구현 FPGA 기술을 이용한다. AT40K FPGA 코어는 프로그래밍가능한 로직으로서, 완전한 PCI-순응식인 SRAM 기반의 FPGA이며, 10 나노초 프로그래머블 동기/비동기식 이중포트/단일포트 SRAM, 68 외부 광역 클럭 및 2AVR 광역 클럭, 캐시 로직 능력, 그리고 10,000-40,000 이용 게이트를 갖추고 있다.

마이크로제어기는 단일 클럭 사이클에서 명령을 실행하여 MHz 당 한개의 MIPS에 접근하는 생산성을 달성한다. 시스템 구조가 전력 소모 대 처리 속도를 최적화하도록 생산성이 얻어진다. AVR 마이크로제어기 코어는 풍부한 명령 세트를 32 범용 워킹 레지스터와 조합하는 향상된 RISC 구조를 바탕으로 한다. 모든 32 레지스터는 산술 로직 유닛(ALU)에 직접 연결되어, 두 개의 독립 레지스터가 한 클럭 사이클에서 실행되는 단일 명령으로 접근되게 한다. 최종 구조는 코드 효율적으로서, 기존 CISC 마이크로제어기보다 10배까지 빠른 생산성을 얻는다.

메모리는 SRAM으로부터 실행된다. FPGA 배치 메모리와 마이크로제어기(22) 명령 코드 SRAM은 Atmel의 인-시스템 프로그래머블 AT17 시리즈 EEPROM 배치 메모리를 이용하여 시스템 전력-온 시에 자동으로 로딩될 수 있다. 세 개의 메인 시스템 빌딩 블록을 단일 프로그래머블 소자 상에서 조합함으로서, 고성능 시스템 레벨 제품이 생성되고, 이는 범용 SLI 소자로 사용될만큼 충분히 유연하고 충분히 가격합리적이다.

FPSLIC EDA 툴인 시스템 디자이너는 진실로 통합된 시스템 설계 방법과 설계 툴을 제공한다. FPSLIC는 표준으로 공동확인 툴을 포함하기 위한 유일한 프로그래머블 솔루션이다. 공동확인은 가상 프로토타입의 생성을 용이하게 하고, 이는 시스템 개발 중에 맞닥뜨리는 문제들을 개발 주기 내에서 보다 쉽게 해결하고, 결과적으로 개발 주기를 줄인다. 공동 확인은 신속한 "what if" 거래가 실행되게하며, 보다 양호한 시스템 효율을 촉진한다.

FPSLIC 실리콘과 시스템 디자이너 소프트웨어는 시간에 따른 시작을 가속시키기 위한 완전한 SLI 솔루션으로 개발되고 있다.

FPSLIC를 내장한 설계

한 개의 프로그래머블 솔루션에 시스템 레벨 제품을 생성하기 위해 필요한 모든 기능을 통합함으로서 FPSLIC 소자는 포괄적이며 집적된 솔루션을 사용자에게 제시한다. FPSLIC 소자는 전형적인 시스템 레벨 구조를 모방한다. 이는 이미 구현된 로직과 마이크로제어기 메모리 사이에 공통 인터페이스를 가진다. 그래서 유연성이나 성능의 절충없이 시스템 설계의 부가가치 부분에 설계자가 집중하는 것이 가능하다.

하드웨어(로직)/소프트웨어(프로그램코드) 공동확인을 추가한 기존 표준 설계 툴은 버그로부터 자유로운 신뢰성있는 마이크로제어기와 FPGA 개발 소프트웨어가 소프트웨어 기반 시스템 레벨 시뮬레이션으로 조합된다는 것을 의

미한다. 그 결과는 생산성을 향상시키는 개발 툴(시스템 디자이너)을 쉽게 이용하는 것으로서, 동시적인 소프트웨어와 하드웨어 설계를 가능하게하고, 설계 개발을 크게 가속시키며, 제품의 주기를 단축시킨다.

단일 SLI 소자 상에서 메모리(SRAM), 로직(FPGA), 마이크로제어기(22)의 조합은 SLI 설계의 데이터 경로(로직)와 제어 흐름(AVR 마이크로제어기) 측면의 효율적 구축을 제공한다. 설계를 균일한 FPGA 솔루션으로 슈-호닝(shoe-horning)하는 대신에, FPSLIC 소자는 시스템의 모든 면을 효율적으로 구축한다. 실리콘 효율로 인해, 다이 크기가 감소하고, 개발 시간이 빨라지며, 고성능 설계가 가능하고 전력 소모도 작아진다. FPSLIC 소자가 가지는 효과는 매우 대단하여, 소자 비용이 대형 FPGA 솔루션과 경쟁하는 것보다 백배 이상 작다. 실리콘 효율과 관련된 일상적인 절충으로 인해 유연성의 결여가 나타난다. 그러나, FPGA를 갖춘 고성능 RISC 마이크로제어기와 동적으로 할당된 SRAM 메모리의 이용은 효율과 유연성을 동시에 제공한다.

FPSLIC 소자에서, 시스템 레벨 통합에 필요한 지적 재산(IP)은 소자의 내재적인 부분(inherent part)이다. 추가적인 IP 블록이 설계의 FPGA(로직) 부분에 추가될 수 있다. 예를 들어, 매개변수로 나타낼 수 있는 마크로 발생기의 라이브러리로부터 추가될 수 있다. 대형 FPGA와 달리, 소프트 IP 프로세서를 통합할 때 시뮬레이션, 위치설정, 타이밍 상의 도전은 FPSLIC 부분으로부터 대부분 제거되었다. 이는 시간에 따른 시장을 가속시키고, SLI 설계의 부가가치 측면에 설계자가 집중하는 것을 가능하게 한다. 로직 기반의 소프트 IP 코어는 설계의 로직 부분에 필요할 경우 FPSLIC 소자에 사용될 수 있다.

FPSLIC 소자의 다중 특징은 로직 자원의 보다 효율적 이용을 포함하여 FPGA나 구분된 해법에 비해 전력 소모를 크게 줄인다. 제어 로직의 구현을 위해 "하드(hard)" 마이크로제어기 코어를 포함함으로서, FPSLIC는 로직 자원을 끌어당기는 전력을 절약한다. 대형 FPGA의 "소프트(soft)" 마이크로제어기 코어는 보다 큰 전력을 소모하는 것보다 더 많은 로직 게이트를 필요로한다.

FPSLIC 소자가 모든 필요한 시스템 블록을 통합하기 때문에, FPSLIC는 소자간 PCB 연결에 연계된 축전적 로딩과 I/O 포트를 제거하여, 시스템 전력을 크게 절약한다.

FPSLIC 소자는 FPGA 클러킹 트리 구조를 포함하고, 이 구조는 작은 세그먼트로 분할되어 있어서, FPSLIC 소자가 레지스터로 클럭 라인을 필요한대로 구동하기만 한다. 클럭 분할(clock partitioning)은 전형적 설계의 FPSLIC 소자의 FPGA 부분에서 동적 전력 소모의 50% 이상을 절약할 수 있다.

FPSLIC의 마이크로컴퓨터(22)는 "버스트-모드" 처리에 사용될 수 있다. 버스트 모드 처리는 AVR이 매우 짧은 시간 주기동안 처리과정을 실행할 수 있게하고, 대부분의 시간에 대해 전력-오프 모드로 놓이게 하여 전력을 절약한다.

FPSLIC 소자의 마이크로제어기와 FPGA의 조합은 FPGA 코어의 부분 재배치를 가능하게 한다. 따라서, 여러 용도로 작용하도록 재배치되는 단일 설계는 실리콘과 전력을 모두 절약한다.

이 인-시스템 재배치 능력은 3세대 이동 통신을 위해 개발중인 "소프트 무선"과 같이 다중 표준을 구현할 수 있어야 하는 설계에 유용하다. 단일 FPSLIC는 어떤 위치나 환경에서도 작동하도록 다중 이동 전화(기지-대역) 표준을 내장할 수 있다. 광대역 코드 분할 다중 접속(W-CDMA)이 표준인 위치에서, W-CDMA 설계는 FPSLIC 소자로 로딩될 수 있다. 전화 사용자가 GSM이 표준인 위치(가령, 유럽)로 이동할 때, FPSLIC 소자는 비행 중에 GSM 표준을 가진 채로 재설정될 수 있다. 이는 여러 다른 시스템 레벨 이동 전화 표준에 단일 FPSLIC 소자가 사용되게 한다.

Atmel의 시스템 디자이너 EDA 툴은 증분 설계 변화, 방대한 라이브러리 제어, 비트스트림 유틸리티를 지원함으로서 재설정가능한 연산을 지원한다.

<u>설계 툴</u>

ASIC나 ASSP 설계를 고안하고 시뮬레이션하며 오류제거하는 것은 기껏해봐야 기운이 꺽이는 일이다. 대형 FPGA를 설계하는 것은 마찬가지로 성가시다. 설계자가 확신을 가지고 마이크로제어기의 소프트-IP 코어 내로 들어갈 때조차 설계자 자신은 일부 중대한 설계 툴 도전에 직면한다. 위치, 경로 시간, 설계 복잡도, 다른 IP 벤더부터의 IP간 상호작용, 설계 성능 등의 문제가 대형 FPGA의 설계자가 접하는 거대한 도전 중 일부이다.

SLI 구현에 관련된 추가적인 도전사항을 수용하도록 변경되지 못한 대형 FPGA 설계 툴과 달리, FPSLIC 설계 툴은 툴의 프로그래머블 로직과 마이크로제어기 영역 사이에 씸리스(seamless) 개발 환경을 보장하는 FPSLIC 구조로 동시에 개발된다.

<u>시스템 디자이너 공동 확인 EDA 툴</u>

FPGA는 Verilog나 VHDL과 같은 하드웨어 기술 언어(HDL)를 이용하여 설계되는 것이 일반적이며, 그후 HDL 시뮬레이터를 이용하여 시뮬레이팅된다. 마이크로제어기 설계는 C-언어나 어셈블리로 행하여지며, 소프트웨어 디버거나 ICE(인-서킷 에뮬레이터)를 이용하여 오류제거된다. Atmel이 직면한 도전은 이 두 해법을 하나의 환경으로 통합시키는 것이었다. 즉, 손쉬운 제품 개발 및 시간에 따른 시장의 가속화를 이룰 뿐 아니라, 설계 과정 초기에 설계의 하드웨어 측면과 소프트웨어 측면간 광범위한 "WHAT IF" 분석을 설계자가 할 수 있게 하는, 그러한 환경으로 통합시키는 것이었다.

Atmel은 시스템 디자이너 동시확인 EDA 툴을 발전시킴으로서 소프트웨어/하드웨어 공동설계의 문제점을 해결하였다. 하드웨어/소프트웨어 공동 확인의 필요성은 기존 설계 주기에 내재하는 생산성과 시간-이익의 장애물로부터 성장한다. 전형적 설계 프로젝트의 반까지가 통합 및 테스트 단계에 소요된다. 실제로 이는 설계 주기의 전면 끝으

로부터 오류 누적을 교정하는 기간이다. 이 오류들은 명세 및 분할 단계까지 줄곧 다다르게 되고, 여기서 하드웨어/소프트웨어 인터페이스의 모호성이 삽입되고 하드웨어/소프트웨어 구현 단계 중에 증폭된다. 이 오류들의 치료는 긴 리드 시간과 ASIC 턴의 높은 비용으로 인해 소프트웨어로 강제되는 경우가 자주 있다. 소프트웨어 고정이 최종 제품의 기능이나 성능을 절충시킬 수 있다.

시스템 디자이너는 Atmel의 FPGA 설계 툴과 3자 하드웨어(베리로그/VHDL) 시뮬레이터를 그 AVR 마이크로제어기 명령 시뮬레이터와 디버깅 툴과 통합한다(씸리스(seamless)). 추가적으로, 동시확인 프레임웍은 하드웨어와 소프트웨어 실행을 완전히 동기화하고, 소스 및 어셈블리 레벨 소프트웨어 디버깅을 완전히 동기화한다. 툴은 완전한 가시도(visibility)의 AVR 메모리와 레지스터를 제공하고, 완전한 하드웨어 설계 가시도를 제공한다. 시스템 디자이너는 완전한 신뢰성의 완전한 하드웨어 및 소프트웨어 설계를 설계자가 할 수 있게 한다. 소프트웨어/하드웨어 절충은 최적의 구현에 도달할때까지 만들어지고 검사될 수 있다. 설계 사이클은 90%만큼 잘려나갈 수 있다. 소프트웨어 개발 툴을 로직 시뮬레이션과 조합함에 추가하여, 동시 확인 환경은 구분된 해법에 앞서 고성능 동시확인 기간을 제시한다. 동시 확인 환경은 소프트웨어 및 하드웨어 개발을 병렬의 활동으로 가능하게 하며, 임계 경로로부터 소프트웨어를 제거하고, 통합 오류로부터 발생하는 하드웨어 프로토타입 반복의 위험을 감소시킨다.

시스템 디자이너의 FPGA 설계 소프트웨어는 Atmel의 IDS 툴을 바탕으로 한다. 이는 Macro Generators, HDL플래너, 푸시버튼 자동 위치 및 루트, 플로어 플래닝, 타이밍 구동 설계, 정적 및 대화형 타이밍 분석, 비트스트림 유틸리티, 증분 설계 변화 능력, 구조 매핑, 백 애노테이션, 대화형 배치 에디터, 라이브러리 매니저를 포함한다.

시스템 디자이너는 FPSLIC FPGA 어레이에 대한 하드/소프트 지적 재산(IP) 코어의 설계를 용이하게 하는 푸시버튼 매크로 발생기가 동반된다. 상기 하드/소프트 지적 재산 코어는 완전히 매개변수화될 수 있다. 매크로 발생기는 전력 소모 영역과, 매크로의 피치를 계산한다. 모든 Atmel 매크로는 AT40K 구조에 대해 최적화된다. 어떤 복잡도의 매개변수가능 IP 코어를 생성하기 위해 사용될 수 있는 시스템 디자이너에 50개 이상의 매크로 발생기가 가용하다. 매크로 발생기는 가산기, FIFO, 카운터, 비교기, 디코더, 플립플롭, 래치, RAM, CRC, 정수 분할기, 선형 피드백 시프트 레지스터, 승산기, 먹스, 니게이션, 시프터, ROM, 감산기, 삼상 버스 제어를 포함한다. 이 기능들은 워드-폭, 전력, 또는 영역에 대해 매개변수화될 수 있고, 흥정이 쉽게 이루어질 수 있다.

시스템 디자이너의 매크로 발생기는 풀다운 메뉴로부터 호출되어, 원하는 기능 구현을 위해선 설계자가 가리키고 누르기만 하면 된다. 매크로 발생기가 호출되면, 대화 박스는 매크로로 적절한 매개변수를 설계자가 구체화하게 한다.

디자이너는 FIR 필터, IIR 필터, 컨볼버(convolver), 그리고 Atmel의 FPGA IP 라이브러리 성장으로부터 자유롭게 가용한 그외 다른 기능과 같이 복합 IP 코어의 세부사항을 Atmel의 WWW 사이트로부터 다운로드할 수도 있다.

Atmel의 HDL플래너 툴은 문장구성상 정확한 베리로그나 VHDL 코드를 자동적으로 발생시킴으로서 FPGA VHDL이나 베리로그 기술의 개발을 자동화한다. HDL플래너는 시스템 디자이너의 매크로 발생기 툴과 함께 개발된 매크로로부터 HDL 기술을 발생시킬 수 있다. HDL 플래너를 이용하여 행해진 어떤 설계도 완전한 소자이고 독립적인 기술이며, 어떤 ASIC나 FPGA에서 구현을 위해 산업 표준 합성 툴을 이용하여 합성될 수 있다. HDL 플래너는 Atmel FPGA에 대해 최적화된 성분을 자동적으로 실증한다. 실증된 성분은 사용자에게 완전히 투명하다. AT40K FPGA 로직에 대해 이들이 구조적으로 최적화되었으나, 실증된 성분은 HDL 설계의 기술적 독립성에 전형 영향을 미치지 않는다.

<u>FPSLIC 펌웨어 설계와 디버깅:</u>

Atmel의 AVR 소프트웨어 설계 환경은 시스템 디자이너 EDA 툴의 통합부분이다. 이는 내장형 명령 세트 시뮬레이터를 이용하여 AVR 프로그램의 개발, 실행, 디버깅을 가능하게 한다.

AVR 스튜디오는 현재 실행중인 코드를 표시하는 포인터와 프로그램 코드를 "소스" 윈도에 제공한다. AVR 스튜디오는 설계의 디버깅을 돕는 여러 화면을 가진다. 이 화면들은 C 프로그램의 순간 변수를 포함한 정해진 심볼의 값을 디스플레이하는 윈도, AVR 레지스터의 모든 32개 내역을 디스플레이하는 윈도, AVR 스튜디오에 의해 지급되는 메시지를 보고하는 윈도, 모든 AVR 메모리 자원의 내역을 보고 수정하게 하는 윈도, 다음 명령이 실행될 어드레스를 보여주는 윈도, 최종 리셋 이후 경과된 클럭 사이클의 수를 가리키는 윈도, AVR 타이머/카운터에 관한 정보를 디스플레이하는 윈도, AVR 포트의 각각에 세 개의 I/O 레지스터를 보여주는 윈도, 온-칩 주변장치(UART, SPI)의 상태를 보여주는 윈도를 포함한다.

FPSLIC 동시 확인 루틴은 HDL 시뮬레이터와 AVR 명령 세트 시뮬레이션이 동시에 대화형으로 구동되게 한다. 하드웨어와 소프트웨어가 함께 설계되고 디버깅되기 때문에, 첫 번에 작동하는 설계를 얻을 가능성이 크게 증가된다. 전체 시스템 개발 시간은 10-90%까지 단축될 수 있다. FPSLIC 시스템 디자이너 소프트웨어는 시간에 따른 시장을 가속시킬만한 완전함과 능력에서 독보적이다.

FPSLIC EDA 환경은 하드웨어 및 소프트웨어의 동시 확인을 포함하여 방법을 설계하기 위한 시스템 접근을 바탕으로 한다.

도 2는 하드웨어 흐름 및 시뮬레이션(31)과 소프트웨어 흐름 및 시뮬레이션(32)의 동시확인(30)을 폭넓게 보여준다. 하드웨어 흐름 및 시뮬레이션은 FPSLIC 소자(20)의 하드웨어 구현이다. 소프트웨어 흐름 및 시뮬레이션은 FPSLIC 소자(20)나 프로세서, 컴퓨터의 소프트웨어 구현이다. 그러나, 소프트웨어 구현(32)은 FPSLIC 소자 상에 FPSLIC 소자(20)의 순수한 소프트웨어 구현이다.

소프트웨어 흐름 및 시뮬레이션(32)을 하드웨어 흐름 및 시뮬레이션(31)과 동시 확인(30)하는 것은 본 발명의 핵심이다. 소프트웨어 흐름 및 시뮬레이션(32)은 즉시 가용한 소프트웨어이다. 동시확인(30)은 한 통일된 환경에서 FPSLIC 설계의 소프트웨어적 측면과 FPSLIC 설계의 하드웨어적 측면을 시뮬레이팅하는 능력을 지원한다. 설계 중 FPSLIC 소자(20)의 일부가 변화할 경우, 소프트웨어 흐름 및 시뮬레이션(32)에서 수정이 가해질 수 있다.

도 3에 도시되는 장치의 한 예에서, 필드-프로그래머블 게이트 어레이의 하드웨어 시뮬레이션과 필드 프로그래머블 게이트 어레이의 소프트웨어 시뮬레이션의 동시 확인 방법이 제공된다. 이 방법은 FPSLIC 소자를 하드웨어적으로 시뮬레이팅하는 단계들을 포함한다.

특히, FPSLIC 소자를 구동하기 위한 FPSLIC 소프트웨어가 FPSLIC 소자에 입력된다(41). FPSLIC 소프트웨어는 소자가 사용될 응용프로그램의 시뮬레이션이다. FPSLIC 소자 소프트웨어는 시뮬레이션으로 구동된다(42). 시뮬레이션에 무제가 있다면(43), FPSLIC 소자 소프트웨어가 수정되거나 변경되고 재입력되며(41), 시뮬레이션이 다시 구동된다(42).

FPSLIC 소자 소프트웨어에 아무 문제가 없을 경우, FPSLIC 소자 소프트웨어로부터 합성(44)과 합성으로부터의 네트리스트(45)가 생성된다. 네트리스트(45)로부터, 위치 및 루트(46)가 생성되고, 최종 시뮬레이션 사후배치(47)가 생성된다. FPSLIC 소자에서, 시뮬레이션 사후배치(47)의 내역이 한 문제점(48)에 대해 확인되며 문제가 있을 경우, 위치와 루트(46)에 변경이 가해질 수 있고, 또는 입력(41)에서의 FPSLIC 소자에 변경이 가해질 수 있다.

시뮬레이션 사후배치(47)에 아무 문제(48)가 없을 경우, FPSLIC 소자를 실리콘(60)으로 구현하기 위해 비트 스트림이 발생될 수 있다.

소프트웨어에서, 이 방법은 프로세서나 컴퓨터를 이용하여 명령 세트 시뮬레이터로 FPSLIC 소자를 시뮬레이팅하는 단계를 포함하고, 명령 세트 소프트웨어로부터 레지스터 내역을 출력하는 단계를 또한 포함한다. 특히, 하이-레벨 코드(51)는 FPSLIC 소자의 시뮬레이팅을 위해 기록된다. 하이-레벨 코드는 한예로 C-언어일 수 있다. 하이-레벨 코드(51)는 컴파일러(52)에 의해 어셈블리 코드로 컴파일된다. 대안으로, 어셈블리 코드가 FPSLIC 소자를 위해 기록될 수 있고, 이 처리과정은 어셈블리 코드(53)와 함께 시작될 수 있다.

명령 세트 시뮬레이터(54)는 FPSLIC 소자의 시뮬레이션으로 어셈블리 코드를 가동한다. 명령 세트 시뮬레이터는 레지스터(61), 주변장치(62), UART 포트, 그리고 소프트웨어의 그외 달리 바람직한 체크포인트에 대한 내역을 출력할 수 있다. 출력 및 명령 세트 시뮬레이터(54)는 문제점(55)에 대해 확인된다. 문제점이 있을 경우, 하이-레벨 코드나 어셈블리 코드가 수정될 수 있고, 명령-세트 시뮬레이터(54)가 다시 구동된다.

문제점(55)이 없을 경우, 어셈블리 코드는 객체 코드(56)로 출력되고, 프로그램 코드(57)로 출력된다. 프로그램 코드(57)는 실리콘(60)에서 FPSLIC 소자를 제작하는 데 사용된다.

이 방법은 시뮬레이션 사후배치로부터의 내역을 레지스터 내역과 확인하는 단계를 추가로 포함한다. 상기 확인은 레지스터 카운터(61), 주변장치(62), UART 포트(63), 시뮬레이터 배치(47)의 검사 내역(50)으로 설명된다. 동시 확인은 실리콘 내의 FPSLIC 구현을 위한 과정의 속도를 향상시킨다. 왜냐하면, 소프트웨어 흐름 및 시뮬레이션과 하드웨어 흐름 및 시뮬레이션이 대화형으로 확인되기 때문이다.

(57) 청구의 범위

청구항 1.

필드 프로그래머블 시스템 레벨 집적 회로(FPSLIC)를 설계하기 위한, 컴퓨터에 의해 실행되는 소프트웨어 시뮬레이션 방법으로서, 상기 필드 프로그래머블 시스템 레벨 집적 회로는 비트 스트림에 의해 구성되는 필드 프로그래머블 게이트 어레이(FPGA) 소자 부분과, 프로그램 코드를 실행하기 위한 마이크로제어기 부분을 포함하며, 상기 소프트웨어 시뮬레이션 방법은,

- 동시확인 소프트웨어를 이용하여 상기 FPSLIC의 사전-배치(pre-layout)를 시뮬레이팅하고,

- FPSLIC의 동시확인을 위한 구현의 시뮬레이션 사후-배치 모델을, 명령-세트 시뮬레이터 소프트웨어를 이용한 FPSLIC의 로직 구현으로부터 발생시키며,

- 상기 명령-세트 시뮬레이터 소프트웨어로 프로그램 코드를 시뮬레이팅하고,

- FPSLIC의 동시확인을 위해 상기 명령-세트 시뮬레이터 소프트웨어로부터 레지스터 내역을 출력하며,

- FPSLIC의 FPGA 소자 부분에 대한 비트 스트림을 생성하고, 그리고

- FPSLIC의 마이크로제어기 부분에 대한 프로그램 코드를 생성하는

단계를 포함하는 것을 특징으로 하는 컴퓨터에 의해 실행되는 소프트웨어 시뮬레이션 방법.

청구항 2.

제 1 항에 있어서, 상기 방법은,

- 상기 비트 스트림과 프로그램 코드로부터, 마이크로제어기에 대한 프로그램 코드와 FPGA 소자에 대한 데이터를 가진 조합된 비트 스트림을 저장 파일에 생성하는

단계를 추가로 포함하는 것을 특징으로 하는 컴퓨터에 의해 실행되는 소프트웨어 시뮬레이션 방법.

청구항 3.

제 1 항에 있어서, 상기 방법은,

- 명령-세트 시뮬레이터 소프트웨어로부터 주변장치 내역을 출력하고, 그리고

- 시뮬레이션 사후-배치로부터의 내역을 주변장치 내역과 확인하는

단계를 추가로 포함하는 것을 특징으로 하는 컴퓨터에 의해 실행되는 소프트웨어 시뮬레이션 방법.

청구항 4.

제 1 항에 있어서, 상기 방법은,

- 명령-세트 시뮬레이터 소프트웨어로부터 UART 내역을 출력하고,

- 시뮬레이션 사후-배치로부터의 내역을 UART 내역과 확인하는

단계를 포함하는 것을 특징으로 하는 컴퓨터에 의해 실행되는 소프트웨어 시뮬레이션 방법.

청구항 5.

필드 프로그래머블 시스템 레벨 집적 회로(FPSLIC)를 설계하기 위한, 컴퓨터에 의해 실행되는 소프트웨어 시뮬레이션 시스템으로서, 상기 FPSLIC는 비트 스트림에 의해 구성되는 필드 프로그래머블 게이트 어레이(FPGA) 부분과, 프로그램 코드를 실행하기 위한 마이크로제어기 부분을 포함하며,

상기 소프트웨어 시뮬레이션 시스템은 소프트웨어 시뮬레이터, 명령-세트 시뮬레이터 소프트웨어, 그리고 FPGA 소프트웨어를 포함하고,

상기 소프트웨어 시뮬레이터는 동시확인 소프트웨어를 이용하여 사전-배치 FPSLIC를 시뮬레이팅하며,

상기 명령-세트 시뮬레이터 소프트웨어는 프로그램 코드를 시뮬레이팅하여 FPSLIC의 동시확인을 위해 레지스터 내역을 출력하고, 그리고

상기 FPGA 소프트웨어는 FPSLIC의 FPGA 소자 부분에 대한 비트 스트림을 생성하고 FPSLIC의 마이크로제어기 부분에 대한 프로그램 코드를 생성하는 것을 특징으로 하는 컴퓨터에 의해 실행되는 소프트웨어 시뮬레이션 시스템.

청구항 6.

제 5 항에 있어서,

상기 FPGA 소프트웨어는, 상기 비트 스트림 및 프로그램 코드로부터, 마이크로제어기에 대한 프로그램 코드와 FPGA 소자에 대한 데이터를 가진 조합된 비트 스트림을 저장 파일에 생성하는 것을 특징으로 하는 컴퓨터에 의해 실행되는 소프트웨어 시뮬레이션 시스템.

청구항 7.

제 5 항에 있어서,

상기 명령-세트 시뮬레이터 소프트웨어는 주변장치 내역을 출력하고, 그리고

상기 동시확인 소프트웨어는 시뮬레이션 사후-배치로부터의 내역을 주변장치 내역과 확인하는 것을 특징으로 하는 컴퓨터에 의해 실행되는 소프트웨어 시뮬레이션 시스템.

청구항 8.

제 5 항에 있어서,

상기 명령-세트 시뮬레이터 소프트웨어는 UART 내역을 출력하고,

상기 동시확인 소프트웨어는 시뮬레이션 사후-배치로부터의 내역을 UART 내역과 확인하는 것을 특징으로 하는 컴퓨터에 의해 실행되는 소프트웨어 시뮬레이션 시스템.

청구항 9.
삭제

청구항 10.
삭제

청구항 11.
삭제

청구항 12.
삭제

청구항 13.
삭제

청구항 14.
삭제

청구항 15.
삭제

청구항 16.
삭제

청구항 17.
삭제

청구항 18.
삭제

청구항 19.
삭제

청구항 20.
삭제

청구항 21.
삭제

청구항 22.
삭제

청구항 23.
삭제

청구항 24.
삭제

청구항 25.
삭제

청구항 26.
삭제

청구항 27.
삭제

청구항 28.
삭제

청구항 29.
삭제

청구항 30.
삭제

청구항 31.
삭제

청구항 32.
삭제

청구항 33.
삭제

청구항 34.
삭제

청구항 35.
삭제

청구항 36.
삭제

요약

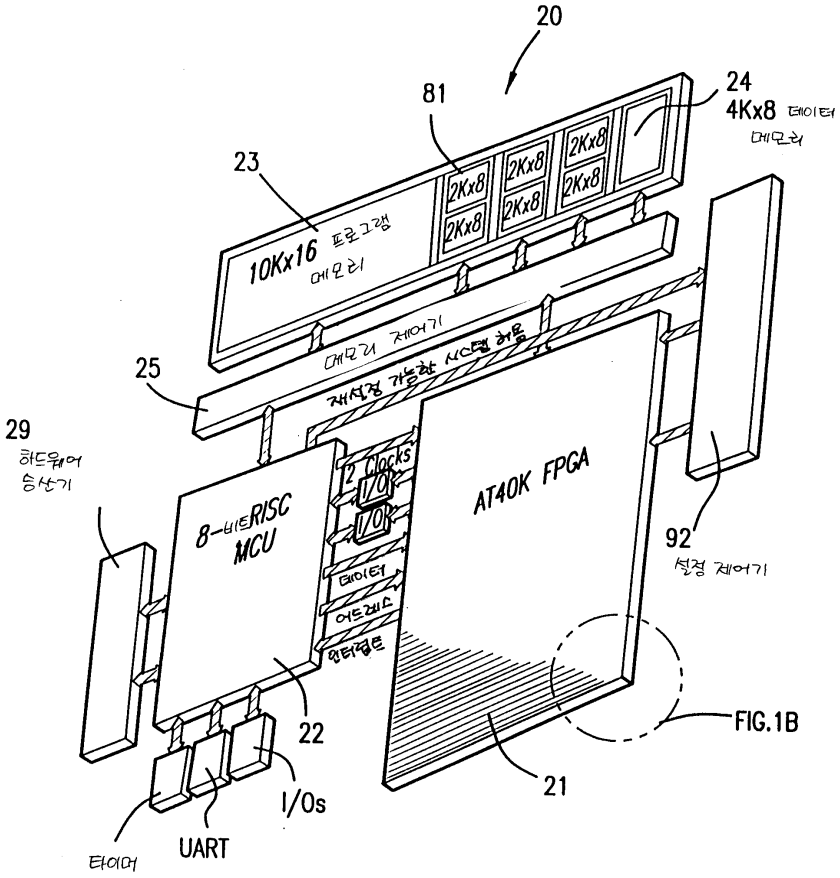
필드 프로그래머블 시스템 레벨 집적 회로(FPSLIC)의 하드웨어 시뮬레이션과 FPSLIC의 소프트웨어 시뮬레이션을 동시확인하는 방법 및 시스템이 공개된다. FPSLIC는 하드웨어로 시뮬레이팅되고(42), FPSLIC 소자의 시뮬레이터-포트 배치(47)가 생성된다. 소프트웨어에서, 상기 방법은 FPSLIC 소자를 명령-세트 시뮬레이터(54)로 시뮬레이팅하고, 명령-세트 소프트웨어로부터 레지스터 내역(61)을 출력한다. 시뮬레이터-포트 배치(47)로부터의 내역이 레지스터 내역(61)과 확인된다. 추가적으로, 상기 방법은 명령-세트 시뮬레이터(54)로부터의 주변 장치 내역(62)을 출력하고 시뮬레이터-포트 배치(47)로부터의 내역을 주변 장치 내역(62)과 확인하는 단계를 또한 포함한다. UART 내역(63)이 명령-세트 시뮬레이터(54)로부터 출력될 수 있으며, 시뮬레이터-포트 배치(47)로부터의 내역을 UART 내역(63)과 확인할 수 있다.

대표도

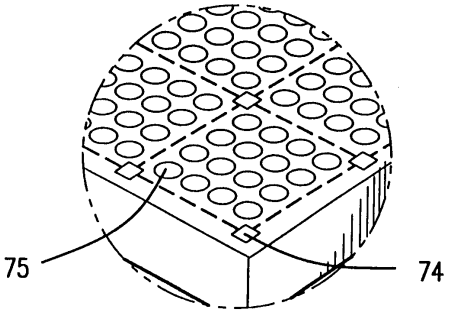
도 3

도면

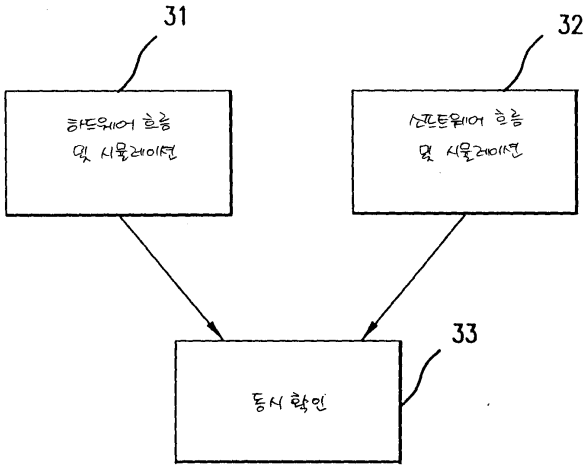
도면1a


20
81
24
4Kx8 데이터 메모리
23
10Kx16 프로그램 메모리
2Kx8
메모리 제어기
재설정 감독관 서드펠 허용
25
29
하드웨어 승산기
12 Clocks
I/O
I/O
데이터
어드레스
인터럽트
8-비트RISC MCU
AT40K FPGA
92
설정 제어기
22
I/Os
21
FIG.1B
타이머
UART

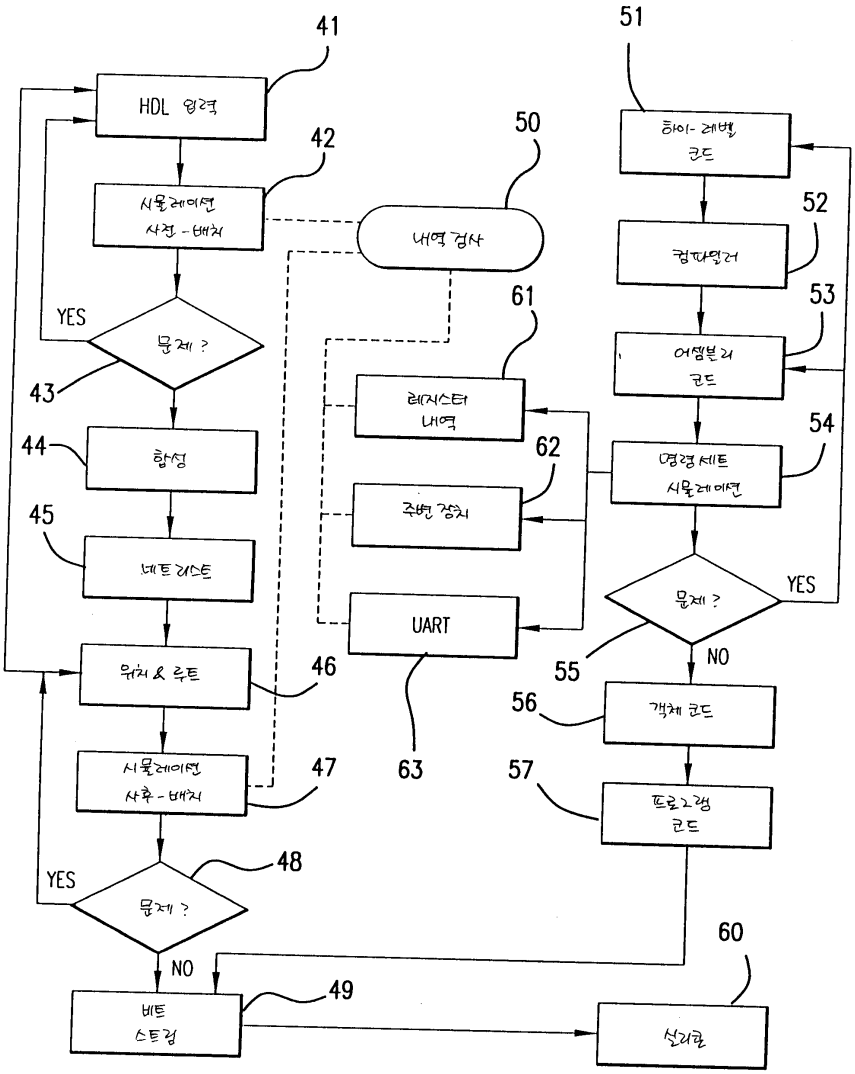
도면1b


75
74

도면2


31
32
하드웨어 흐름
및 시뮬레이션
소프트웨어 흐름
및 시뮬레이션
33
동시 확인

도면3


41
HDL 입력
42
시뮬레이션
사전 - 배치
50
내역 검사
YES
문제 ?
43
44
합성
45
네트 리스트
46
위치 & 루트
47
시뮬레이션
사후 - 배치
48
YES
문제 ?
NO
49
비트
스트림
61
레지스터
내역
62
주변 장치
UART
63
51
하이 - 레벨
코드
52
컴파일러
53
어셈블리
코드
54
명령 세트
시뮬레이션
YES
문제 ?
55
NO
56
객체 코드
57
프로그램
코드
60
선리킹