



(19) 대한민국특허청(KR)

(12) 등록특허공보(B1)

(45) 공고일자 2013년12월03일

(11) 등록번호 10-1336516

(24) 등록일자 2013년11월27일

(51) 국제특허분류(Int. Cl.)
G06F 21/00 (2006.01) G06F 17/50 (2006.01)

(21) 출원번호 10-2008-7013174

(22) 출원일자(국제) 2006년10월26일

심사청구일자 2011년10월26일

(85) 번역문제출일자 2008년05월30일

(65) 공개번호 10-2008-0072028

(43) 공개일자 2008년08월05일

(86) 국제출원번호 PCT/US2006/060261

(87) 국제공개번호 WO 2007/102887

국제공개일자 2007년09월13일

(30) 우선권주장

11/290,368 2005년11월30일 미국(US)

(56) 선행기술조사문헌

US7168041 B1

US20050229128 A1

전체 청구항 수 : 총 2 항

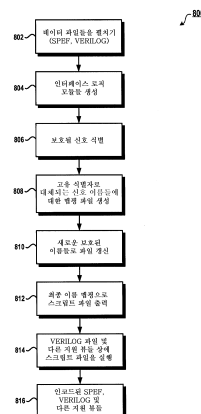
심사관 : 문남두

(54) 발명의 명칭 EDA 툴 설계 뷰들에서 정보를 보호하기 위한 방법 및프로그램 제품

(57) 요약

특정 EDA 툴 설계 뷰들 내의 모듈 이름들 및 구조들과 같은 HDL 인터페이스 로직 모델들 내의 민감한 회로 설계 정보(806)는, 설계 뷰들을 회로 설계의 시뮬레이션 모델의 일부로서 전달하기 전에, 선택된 인스턴스 및 네트 이름들을 비관련된 고유의 식별자들로 대체함으로써(808) 제거되어, 그 결과 그 정보의 권한 밖의 이용을 회피한다. IC 설계의 상이한 설계 뷰들 내의 신호 이름들을 인코딩하는 방법은 복수의 설계 데이터베이스들에 포함된 이름들의 리스트를 제공하는 단계(806), 이름들의 리스트 내의 각각의 이름을 보호된 이름으로 변경하는 단계(808), 및 각각의 설계 뷰 데이터베이스 내의 각각의 변경되는 이름을 연관된 보호된 이름으로 대체하는 단계(810, 812, 814, 816)를 포함한다.

대표도 - 도8



특허청구의 범위

청구항 1

설계 정보를 보호하기 위한 방법으로서,

IC 설계 내의 하나 이상의 로직 경로를 식별하는 네트 이름(net name)들 및 인스턴스 이름(instance name)들을 찾기 위해 상기 IC 설계의 제1 뷰(view)를 스캔하는 단계;

상기 스캔하는 단계에서 발견된 하나 이상의 이름들 각각을, 연관된 고유의 식별자에 맵핑하는 단계 - 상기 연관된 고유의 식별자는 그와 연관된 이름과는 다름 - ;

상기 제1 뷰에서, 맵핑되는 이름의 하나 이상의 출현들(occurrences)을 그와 연관된 고유의 식별자로 대체하여 상기 IC 설계의 제1 보호된 뷰를 생성함으로써, 상기 IC 설계 내의 하나 이상의 로직 경로가 상기 제1 보호된 뷰에 드러나지 않도록 하는 단계; 및

상기 IC 설계의 제2 또는 그 이상 번째의 뷰(a second or more view)에서, 맵핑되는 이름의 하나 이상의 출현들을 그와 연관된 고유의 식별자로 대체하여 상기 IC 설계의 제2 보호된 뷰를 생성함으로써, 상기 IC 설계 내의 하나 이상의 로직 경로가 상기 제2 보호된 뷰에 드러나지 않도록 하는 단계

를 포함하는 설계 정보 보호 방법.

청구항 2

상이한 정보를 포함하는 복수의 데이터베이스들 내의 신호 이름들을 인코딩하는 방법으로서,

복수의 데이터베이스들에 포함된 이름들의 리스트를 제공하는 단계 - 상기 복수의 데이터베이스들 각각의 데이터베이스는 IC 설계에 관련된 상이한 정보를 포함함 - ;

상기 이름들의 리스트에 있는, 상기 IC 설계 내의 하나 이상의 로직 경로를 식별하는 네트 이름 및 인스턴스 이름 각각을 보호된 이름으로 변경함으로써, 상기 IC 설계 내의 하나 이상의 로직 경로가 드러나지 않도록 하는 단계; 및

상기 복수의 데이터베이스들 각각의 데이터베이스 내의 각각의 변경된 이름을 연관된 보호된 이름으로 대체함으로써, 상기 IC 설계 내의 하나 이상의 로직 경로가 상기 복수의 데이터베이스들 중 어느 데이터베이스에도 드러나지 않도록 하는 단계

를 포함하는, 신호 이름들을 인코딩하는 방법.

청구항 3

삭제

청구항 4

삭제

청구항 5

삭제

청구항 6

삭제

청구항 7

삭제

청구항 8

삭제

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

명세서

기술분야

[0001] 본 발명은 일반적으로 디지털 회로를 위한 컴퓨터 지원 설계(computer aided design: CAD) 분야에 관한 것으로, 구체적으로는 전자 설계 자동화(electronic design automation: EDA) 모델들에 포함된 설계 정보의 보호에 관한 것이다.

배경기술

[0002] 다층 초미세 상보성 금속산화막 반도체(multi-layer deep sub-micron complimentary metal-oxide semiconductor(CMOS)) 기술과 같은 실리콘 기술의 현대의 진보는 더욱 크고 더욱 복잡한 설계들이 단일 칩 상에 형성되도록 허용하고, 이에 따라 전체 "SOC(System-on-a-chip)" 설계들이 동일한 실리콘 기판 상에 배치되는 것을 가능하게 해주는 원인이 된다. 이 복잡한 설계들은 예를 들면, 하나 또는 그 이상의 프로세서 컴포넌트들,

디지털 신호 처리(digital signal processing: DSP) 컴포넌트, 메모리, 몇몇의 통신 인터페이스들, 및 그래픽 지원 컴포넌트를 일체화할 수 있다.

[0003] SOC 설계의 도래는 전자 시스템 산업이 기존의 회로 블럭들(circuit blocks) 혹은 "코어들(cores)"의 이용에 기초한 새로운 설계 패러다임으로 움직이게 한다. 이 패러다임 하에서, 시스템들은 하나 이상의 기존 회로 블럭들을 동일한 실리콘 기관 상에 집적화함으로써 조립된다. IC 설계자들이 모든 필요한 하드웨어를 각각 새로운 IC 설계로써 재설계하는 것보다는 이미 개발된 컴포넌트들을 일체화하는 것이 보다 더 효율적이고 실용적이다. 이러한 움직임으로부터, 상이한 칩 설계 공급업체들(vendors)이 기존의 회로 블럭들을 SOC를 개발하는 시스템-레벨 통합자 또는 제작자에게 공급하는 것이 자연스럽게 뒤따랐고, 종종 이 코어들에는 그러한 공급업체들 또는 IC 통합자가 아닌 그룹들에 의해 소유된 지적재산권(intellectual property: IP)이 따른다.

[0004] 시장의 요구들은 전자 시스템 제작자들이 보다 빠르고 효율적으로 시스템을 개발하도록 강요를 계속함에 따라, 전자 시스템 통합자들 또는 제작자들은 SOC의 시뮬레이션 및 설계 통합을 허용받기 위해 코어의 설계 모델들을 받는 것이 유용하다. IC 설계 시뮬레이션 모델은, IC 설계자가, 하나의 컴포넌트가 설계자의 요구를 달성하기 위해 다른 컴포넌트와 작동할 것인지 여부를 알 수 있게 해준다. 설계와 방법의 복잡도가 증가함에 따라, 유용성 및 통합을 허용하기 위해 코어 설계에 대한 더 많은 상세 사항과 가시성이 요구된다. 안타깝게도, 더 정확하고 상세한 모델이 SOC 통합자에게 전달될수록, 그 설계들의 도난과 권한 밖의 조작과 같은 오용의 위험이 더 증가한다.

[0005] 코어 설계들과 모델들은 저장 장치 상에 데이터 또는 코드로서 물리적으로 존재하기 때문에, 이들을 복사하고, 위조하고, 재설계하는 것이 비교적 쉽다. 또한, 코어들의 존재에 의해 초래되는 증가된 효율성은 이 아이템들의 권한 밖의 사용, 재사용, 전달 또는 판매를 위한 동기를 제공한다. 그 결과, 설계 및 방법의 복잡성은 SOC 설계자에 의한 코어의 이용가능성을 허용하기 위해 상당한 양의 정보를 전달할 것을 요구하고 있는 사이에, SOC 설계에 참가하는 제3자들, 공급업체들, 소비자들에게 IP가 전달되는 것을 방지하기 위한 노력이 필요하다는 것이 더 중요해지고 있다.

[0006] 시뮬레이션 모델이 설계의 동작 및 기능에 대한 유익한 통찰력을 제공하기 때문에, 설계에 있어서 IP의 소유자들은 전형적으로 협력자들과의 사이에서 주의깊게 초안을 잡은 법적 협약이 없이는 시뮬레이션 모델을 공개하지 않는다. 그러한 협약을 협상하는 것은 모든 참가자들에게 시간을 들게 하고 돈을 들게 한다. SOC 설계자가 설계 또는 아이디어를 시험하기 원하면, 그러한 협약을 획득하는 것은, 빠르고, 효율적이고, 비용-효율적인 설계 개발에 상당한 손해이다. 또한, 그들만으로도, 그러한 법적 수단은 제한적으로 유용한데, 왜냐하면 독점 코어들의 불법적인 복사, 위조, 전달 또는 재사용의 검출이 어렵기 때문이다. 코어가 특별한 사용자에게 의해 불법적으로 획득되었는지, 또는 회로 설계 내의 무슨 기밀 정보가 그의 전자적인 형태로부터 획득되었는지를 결정하는 것은 매우 힘들 수 있다. 그러므로, 코어들의 공급자들은 그들이 코어들의 설계, 개발 및 획득에 소비한 자원들의 이익을 빼앗기지 않기 위해 그들의 설계들을 보호하는 효과적인 방법이 필요하다.

발명의 상세한 설명

[0017] 본 발명의 예시적인 실시예들에 대한 다음의 설명에서, 본 발명이 실시될 수 있는 특정한 예시적인 실시예들은 당업자들이 본 발명을 실시하기에 충분히 상세히 기술되어 있으며, 다른 실시예들도 활용될 수 있고, 본 발명의 사상 또는 범위를 벗어나지 않고서 로직의, 아키텍처의, 프로그램의, 기계적인, 전기적인, 그리고 기타 다른 변경들이 만들어질 수 있다는 것을 이해할 것이다. 그러므로, 다음의 상세한 설명은 제한적인 의미로 해석되지 말아야 하고, 본 발명의 범위는 첨부된 청구항들에 의해서만 정의된다. 특히, 하드웨어 기술 언어(hardware description language: HDL) 표현이나 다른 심볼의 표현을 이용하여 IC 설계나 또는 IC 설계의 시뮬레이션 모델을 나타내는 임의의 방식은 본 발명의 범위 내로 고려되어야 한다는 것은 당업자들에게 자명할 것이다.

[0018] 본 명세서에서 설명된 하나 또는 그 이상의 실시예들에서, 회로 설계의 시뮬레이션 모델의 일부로서 전달되는 기밀 회로 설계 정보를 보호하고, 그 결과 그 정보의 허가되지 않은 사용을 회피하기 위한 방법 및 프로그램 제품이 제공되는데, 기밀 정보란 특정 EDA 툴 설계 뷰들 내의 모듈 이름과 구조 등을 포함한다. 이러한 방식에서, 본 발명의 방법 및 시스템은, SOC 설계자가 코어 설계의 시뮬레이션 모델을 이용하여, 코어의 동작 및 시뮬레이션 모델의 컴포넌트들이 모델 내에 포함된 기밀 정보에 대한 접근이 허용되지 않은 채 SOC 설계자의 요구를 만족시키는지를 결정할 수 있게 한다. 이것은, 그렇지 않다면, 시뮬레이션 모델에 구체화될 수 있는 기밀의 코어 설계 정보를 보호하기 위해 SOC 설계자(혹은 회사)와 IP 소유자 사이에 있을 기밀보장 협약에 대한 의존을, 부분적으로, 완화시킨다.

- [0019] 가상 컴포넌트 블록들은 특별한 칩 설계를 위해 요구되는 추상화의 레벨에 따라, 다수의 단계들을 수반할 수 있는 설계 프로세스의 일부로서 전형적으로 생성된다. 칩 설계자들은 종종 설계 프로세스에 있어서 돕기 위해, 그리고 시제품화(prototyping)나 생산 전에 칩 설계의 시뮬레이션을 허용하기 위해 전자 설계 자동화(EDA) 소프트웨어 툴들을 종종 이용한다. EDA 소프트웨어 툴들을 이용한 칩 설계는 일반적으로 반복(iterative) 프로세스를 수반하는데, 이에 의해 칩 설계가 점진적으로 완성된다. 전형적으로, 칩 설계자는 필요한 회로 설계의 부분들을 디스플레이하도록 높은 품질의 그래픽 성능을 일반적으로 갖는 컴퓨터 워크스테이션에 정보를 입력함으로써 회로를 조립한다. 하향식(top-down) 설계 방법인, 예를 들면 베릴로그(Verilog) 또는 VHDL 등과 같은 하드웨어 기술 언어들(HDLs)을 이용하여 통상적으로 채용될 수 있는데, 이에 의해 설계자는 회로의 기능 컴포넌트들을 계층적으로 정의하고, 그런 다음 각각의 컴포넌트를 더 작고 더 작은 컴포넌트들로 분해함으로써 집적 회로(integrated circuit)를 생성한다.
- [0020] 집적 회로의 다양한 컴포넌트들은 그들의 기능적 동작들과, 관련된 입력 및 출력에 의해 처음에 정의된다. 이 설계 상태들 동안, 설계자는 고려해야 할 계층 정보를 이용하여 회로를 일반적으로 구조화하고, 설계에 실질적인 규칙성을 전형적으로 제공한다. 디지털 회로의 설계는 컴퓨터 시스템의 메모리 내에서 상이한 방법들로 나타낼 수 있거나 특정될 수 있다. 디지털 회로 표현의 각각의 타입은 뷰(view)라고 불린다. 상이한 뷰들은 회로의 구조에 관한 상이한 양 및 타입의 정보를 포함한다. 회로의 구조에 관해 더 많은 정보를 포함하는 뷰들은, 구성(construct)하고 조작(manipulate)하기 위해 컴퓨터의 메모리를 더 많이 필요로 하고 컴퓨터의 시간을 더 많이 요구한다.
- [0021] HDL 또는 다른 하이 레벨 기술(description)로부터, 실제의 로직 셀 구현은, 회로의 기능적인 기술을 특정 회로 구현으로 변환하는 로직 합성(logic synthesis)에 의해 전형적으로 결정된다. 그런 다음 로직 셀들은 "배치"되고(즉, 회로 레이아웃 내에 특정 좌표 위치들이 주어짐) "라우트"된다(즉, 설계자의 회로 정의에 따라 배선되거나 함께 연결됨). 배치 및 라우팅 루틴들은 로직 합성 프로세스에 의해 생성되었던 펼쳐진 네트리스트(flattened netlist)를 그들의 입력으로서 일반적으로 받아들인다. 이 펼쳐진(flattened) 또는 계층적인(hierarchical) 네트리스트는 타겟 표준 셀 라이브러리로부터 특정 로직 셀 인스턴스들을 식별하고, 특정한 셀-대-셀 접속을 기술한다.
- [0022] IC 칩은 반도체 기관의 표면 상에 형성된 셀들 및 셀들 사이의 접속부들을 포함한다. 셀은 기능을 수행하기 위한 그룹화된 트랜지스터들, 캐패시터들, 및 다른 기본 회로 소자들과 같은 하나 또는 그 이상의 회로 소자들의 그룹이다. IC는 많은 수의 셀들을 포함할 수 있고, 셀들 사이의 복잡한 접속부들을 필요로 한다. IC의 셀들 각각은 하나 또는 그 이상의 핀들을 가질 수 있고, 핀들 각각은, 차례로, 배선들에 의해 IC의 하나 또는 그 이상의 다른 핀들에 접속될 수 있다. 핀을 IC에 접속하는 배선들이 또한 칩의 표면 상에 형성된다.
- [0023] 네트(net)는 두개 혹은 그 이상의 상호접속 핀들의 세트이고, 이에 의해 핀들을 갖는 로직 회로들을 연결한다. 전형적인 칩은 다양한 조합들로 연결되어야 하는 수천, 수만, 수십만의 핀들을 갖기 때문에, 칩은 또한 수천, 수만, 수십만의 "네트들(nets)" 혹은 핀들의 세트들의 연결들의 정의들을 포함한다. 칩에 대한 네트들의 개수는 전형적으로 그 칩 상의 셀들의 개수의 수준과 동일한 수준이다. 통상적으로, 네트들의 대다수는 연결될 두개의 핀들만을 포함하지만, 많은 네트들이 세개 또는 그 이상의 핀들을 포함한다. 어떤 네트들은 연결될 수백 또는 수천의 핀들을 포함할 수 있다.
- [0024] IC 모델은 HDL에 전형적으로 표현된 상호접속된 회로 셀들 및 네트들의 세트이고, 가끔 본 명세서에서 모듈, 코어, 지적 재산(IP), IP 블록(block), 블록(block), 또는 컴포넌트로서 일컬어진다. 코어는, 전형적으로 HDL로 특정된, 하드웨어로 구현가능한 전자 설계이고, 또한 전형적으로 탑-레벨 네트리스트(top-level netlist)로 표현된다. "네트리스트"는 접속된 핀들의 이름들을 포함하는 네트들의 리스트("네트 이름들(Net Names)")이거나, 또는 셀들의 핀들에 접속하는 네트들의 이름들을 포함하는 셀들의 리스트("인스턴스 이름들(Instance Names)")이다. 탑-레벨 RTL은 로우 레벨의 추상화에서 게이트-레벨 네트리스트를 생성하기 위해 합성된다. 설계의 합성 후에, 게이트-레벨 네트리스트가 검증되고, 회로의 레이아웃이 결정되고, IC가 제조된다.
- [0025] 오늘날, 대부분의 디지털 집적 회로들(IC's)의 설계는 HDL 방법론에 기초한 매우 구조화된 프로세스이다. HDL 방법론은 명세 문서(specification document)에 의해 특정되는 IC를 이용하여 시작한다. 그런 다음, IC 명세가 레지스터 전송 언어(Register Transfer Language: RTL)로서 일컬어지는 설계 추상화의 레벨에서 하이-레벨 HDL 코드로 기록된다. RTL 코드는 회로들을 레지스터들, 불리언 방정식들, 및 제어 로직, 예를 들면 복잡한 이벤트 시퀀스들뿐만 아니라 "if-then-else" 문장들의 모임(collection)으로서 기술한다. RTL 코드는 전형적으로 RTL 레벨의 추상화에서 Verilog-HDL("Verilog") 또는 VHDL과 같은 HDL 언어를 이용하여 전형적으로 구현되는데, IC

설계는, 데이터가 회로의 입력, 출력 및 클럭되는 레지스터들 사이를 흐를 때, 데이터 상에 수행되는 연산들을 기술함으로써 규정된다. RTL 레벨에서, 설계자들은 설계 계층화, 및 파티셔닝(partitioning), 클럭킹 스킴(clocking scheme), 리셋 스킴(reset scheme), 및 레지스터들의 위치들과 같은 모든 주요한 설계 결정을 해야만 한다. 모든 그러한 결정들은 RTL 코드에 포함되고 반영된다. RTL 코드는 설계 툴들로부터 독립적일 뿐만 아니라 일반적으로 기술 독립적이다.

[0026] RTL 코드에 의해 표현되는 IC 설계는, 그런 다음 게이트-레벨 기술 또는 넷리스트를 생성하기 위해 합성된다. 합성(synthesis)은 RTL 코드에 의해 표현된 설계의 아키텍처 및 기능의 기술(description)을 로직-레벨 및 게이트-레벨 기술들과 같은 설계의 로우-레벨 표현으로 변환하기 위해 수행되는 단계이다. IC 설계 명세와 RTL 코드는 일반적으로 기술 독립적이다. 즉, 명세와 RTL 코드는, 일반적으로, 설계를 구현하기 위해 이용될 정확한 게이트들이나 로직 디바이스들을 규정하지 않는다. IC 설계의 게이트-레벨 기술은 기술 의존적이다. 이는 왜냐하면, 합성 프로세스 동안, 합성 툴은 기술 독립적 RTL 코드를 기술-의존적 게이트-레벨 넷리스트로 맵핑하기 위해 주어진 기술 라이브러리를 이용할 수 있기 때문이다. 그러나, 넷리스트는 범용적이거나 또는 기술 특정적일 수 있다. 범용적인 넷리스트는 셀들의 기술-특정적인 라이브러리와 연관되어 있지 않은 RTL 코드로부터 생성된 넷리스트이다. 기술-특정적 넷리스트 또는 맵 넷리스트(map netlist)는 IC 설계가 셀들의 특별한 기술-특정적 라이브러리에 맵핑된 후 생성되는 넷리스트이다.

[0027] 본 명세서에서 설명되는 본 발명의 실시예들은 광범위하고 다양한 가상의 컴포넌트 블록들에 응용가능하며, 다양한 가상의 컴포넌트 블록들(virtual component blocks)은 시스템 칩 설계에 사용되는 시스템 레벨 매크로(macros), 메가셀(megacells), 또는 임베디드 소프트웨어 "코어들(cores)"과 관련되든지 어떻든지 상관없다. 가상의 컴포넌트 블록들은, 일반적으로, 설계 스테이지에서 추상화의 레벨에 한해서는 그들의 "견고성(firmness)"에 따라 카테고리가 정해질 수 있다. 가상 컴포넌트 블록들은, 예를 들면, 새개의 견고성 카테고리, 즉 "소프트(soft)", "펌(firm)", 및 "하드(hard)" 중 하나로 분류될 수 있다. 견고성의 카테고리는 기능적인 명세로부터 특별한 제조 프로세스까지의 범위에서, 개발 스테이지를 특성화한다.

[0028] "소프트" 가상 컴포넌트 블록들은 전형적으로 합성가능한 하드웨어 기술 언어(예를 들면, VHDL 또는 Verilog 등과 같은)의 형태로 SOC 통합자(SOC integrator)에게 전달된다. 소프트 가상 컴포넌트들은 매우 플렉서블(flexible)하고, 범용적일 수 있고 구성가능(configurable)할 수 있다. "펌" 가상 컴포넌트 블록들은 범용의 기술에 대해 최적화된 주로 구조적-타입의 모델들이다. 펌 가상 컴포넌트 블록의 상세 수준은, 예를 들면, RTL 서브-블록들의 영역 배치가 지정되는 회로 설계로부터, 상대적으로 배치된 데이터 경로들을 통해, 완전히 배치된 넷리스트를 갖는 회로 설계까지의 범위에 걸칠 수 있다. 펌 가상 컴포넌트 블록들은 일반적으로 라우팅을 포함하지 않으며 타겟 기술들에 걸쳐 포터블(portable)하다. "하드" 가상 컴포넌트들은 특정 기술에 맵핑되고, 일반적으로 타겟 기술에서의 파워(power), 크기 및 성능에 대해 최적화된다. 하드 가상 컴포넌트들의 예들은 주문형 물리적 레이아웃(custom physical layout), 또는 주어진 기술에 대해 완전히 배치되고 라우트되고 최적화된 넷리스트를 포함할 수 있다. 하드 가상 컴포넌트 블록들은 프로세스 또는 공급업체(vendor) 특정적일 수 있으며, 보통 GDSII와 같은 표준 설계 언어로 표현된다. 일반적으로 사용자는, 하드 가상 컴포넌트 블록을 효과적으로 이용하기 위해, GDSII 파일과 함께 하이 레벨 거동적 모델(high level behavioral model)과 완전한 물리적/타이밍 모델들에 대한 액세스가 주어진다.

[0029] SOC의 설계 및 분석의 태스크들을 단순화하기 위해, 복잡한 설계는 블록들, 즉 칩 내의 사용자-정의 또는 시스템-정의 계층으로 분할될 수 있다. 이 시점에서, 블록들은 인터페이스 모델들로 대체될 수 있다. 유익하게, 그러한 모델들은 예를 들면, 블록들에 대한 인터페이스 정보를 블록의 SOC 통합자와 교환하는 컴팩트한 수단을 제공할 수 있다. 블록이나 코어를 위해 완전한 게이트-레벨 넷리스트 대신 인터페이스 모델들의 사용은, 특히 게이트 개수가 5백만 게이트를 초과하는 복잡한 설계들을 위해, 칩-레벨 분석 및 시뮬레이션과 연관되는 성능 및 용량(capacity)을 향상시키기 위한 열쇠이다. 또한, 그러한 인터페이스 로직 모델들은, 칩 설계자가 인터페이스 로직 모델(interface logic model: ILM)과 함께 전달될 필요가 없는 내부 게이트 레벨 설계를 보존하고 보호하도록 허용한다.

[0030] 자동 모델 생성(automatic model generation)은 ILM들의 이용을 포함할 수 있다. ILM들은 넷리스트, 제약들(constraints), 및 설계상의 내부 레지스터-대-레지스터 경로들과 연관된 역향 주석(back-annotation)을 무시하며, 이에 의해 정확성을 희생하지 않고서 상당한 실행 시간(runtime) 및 성능 향상을 제공한다. 또한, ILM은 원래 넷리스트의 인터페이스 로직만을 포함한다. 예를 들어, 블록 상의 레지스터-대-레지스터 경로들에만 포함된 로직은 ILM 내에 배치되지 않는다. ILM은, 입력 포트에서 시작하여 에지-트리거드 레지스터(edge-triggered register)(즉, 순차적(sequential) 로직)나 출력 포트 중 어느 하나에서 끝나는 타이밍 경로에 포함

된 셀들(즉, 로직과 연관 네트들(본 명세서에서 인터페이스 네트들로 일컬음)의 조합)을 포함한다. 투명한 래치가 그러한 타이밍 경로 내에서 만나지게 되면, 그것은 에지-트리거드 레지스터 또는 출력 포트가 만나질 때까지 조합의 로직 및 경로 추적이 래치를 통해 계속되는 것으로 취급된다.

- [0031] ILM은 설계의 인터페이스 타이밍의 시뮬레이션 모델 표현으로서 이용될 수 있다. ILM 모델은 설계 Verilog 네트리스트 및 적당한 기생 정보(parasitic information)로부터 생성될 수 있다. 컴포넌트의 ILM 모델은 어떻게 그 컴포넌트가 타이밍에 대해 동작하고 다른 컴포넌트들과 상호작용하는지에 대한 상세한 이해와 탐구를 가능하게 한다.
- [0032] 이제 도면들을 참조하면, 특히 도 1을 참조하면, 본 발명의 일 실시예의 블록도가 도시된다. IC 설계 정보(Verilog 설계)(102)는 Verilog로 구현된다. IC 설계 기생 정보(SPEF)(104)는 IC 설계 정보(102)와 연관되고, 표준 기생 교환 포맷(Standard Parasitic Exchange Format: SPEF)이다. 또한, Verilog 설계(102)와 SPEF(104)는 코어 또는 설계 블록에 대한 설계 정보의 중요한 세트를 제공한다. 표준 EDA 툴들을 이용하여, SPEF(104)는 ILM(106)으로 변환될 수 있고, Verilog 설계(102)는 Verilog ILM(108)으로 변환될 수 있다. ILM을 생성할 때, EDA 툴은 플랫 게이트-레벨 네트리스트를 포맷하는데, 이것은 회로 시뮬레이션, 타이밍 검증 및 회로 배치 및 라우팅을 위해 EDA 툴들에 의해 때때로 이용된다. SPEF ILM(106)과 Verilog ILM(108)은 회로 설계의 모델 뷰들을 나타낸다. 이해하게 되는 바와 같이, 각 모델 뷰는 모듈의 동작 및 기능에 대한 통찰력을 제공할 수 있는 기밀의 시뮬레이션 정보를 포함한다.
- [0033] 도 2는 플랫 게이트-레벨 SPEF의 예이다. 네트리스트(200)는 이름 맵(202), 포트 기생 정보(204), 및 네트 연결과 기생 정보(206)를 포함한다.
- [0034] 도 3은 Verilog 뷰에서 모듈 또는 코어 설계의 예를 도시한다. 플랫 게이트-레벨 네트리스트(300)는 각각의 "인스턴스 이름", 각각의 "네트 이름", 및 인스턴스 이름들과 네트 이름들 사이의 계층적 연관을 정의한다. 인터페이스들(302)은 모듈의 탑-레벨의 입력들 및 출력들에 접속하는 네트들을 식별한다. 네트 이름들(304)은 모듈의 탑-레벨의 입력 및 출력에 접속하지 않는 네트의 인스턴스 각각을 식별한다. 인스턴스 이름(306)은 뷰 내에서 인터페이스 셀의 각 인스턴스를 식별한다. 도 4는 인스턴스 이름(402), 복수의 입력들(404, 406)(또는 그 이상) 및 출력(408)에 의해 식별되는 인스턴스 이름들(306)의 포맷을 나타낸다.
- [0035] 도 1을 다시 참조하면, 설계 보호 시스템(110)은 설계자에게 컴포넌트들, 코어 또는 모델이 설계 목적을 달성하는지의 여부에 관한 이해를 제공하면서도, 시뮬레이션 모델, 코어 및/또는 컴포넌트들의 독점 부분들(proprietary portions)의 공개를 차단한다. 바람직한 실시예에 따르면, 설계 보호 시스템(110)은 IC 모델들 내에 포함된 인스턴스 이름들, 네트 이름들 및, 가능하다면, 연관된 계층 정보를 재정의함으로써, 기밀 정보를 보호한다. 설계 보호 시스템(110)은 일관적인 방식으로 모든 설계 뷰들에 걸쳐 특정 데이터를 재정의한다. 설계의 제1 뷰는, 뷰 내의 특정 인스턴스 이름들과 네트 이름들을 재정의하기 위해 암호를 이용하여 인코딩된다. 플랫 네트리스트 내의 인스턴스 이름, 네트 이름 또는 계층적 연관이 완전한 로직 경로들을 식별할 때, 이들은 그 인스턴스 또는 네트에 대한 단일의 고유 식별자로 대체된다. 그런 다음, 뷰들에 걸쳐 일관성을 유지하기 위해 관련 상보적인 뷰들이 정확히 동일한 방식으로 인코딩된다.
- [0036] 바람직한 실시예에서, 설계 보호 시스템(110) 내의 스크립트(script)는 민감하거나 기밀의 정보의 일부 또는 모두가 제거된 보호된 SPEF ILM(112) 및 보호된 Verilog ILM(114)을 생성하지만, 제3자 측의 사용자들이 모델들과 완전한 상호작용을 하도록 여전히 허용한다. 보호된 SPEF ILM(112) 및 보호된 Verilog ILM(114)은 표준 EDA 툴들과 완전히 호환적인 포맷으로 코어의 시뮬레이션이 이루어지게 한다.
- [0037] 도 5는 본 발명의 바람직한 실시예에 따른, 뷰들(102, 104) 또는 ILM들(106, 108) 각각에서 발견되는 인스턴스 이름들, 네트 이름들 및 계층적 연관들을 인코딩하기 위해 설계 보호 시스템(110) 내의 스크립트에 의해 수행되는 대체 스킴의 맵핑을 도시한다. 설명의 목적을 위해, 설계 보호 시스템(110)은 ILM들(106, 108)(도해상 예시적인 뷰들(200, 300)로부터 유도된 것으로 고려됨)을 보호하는 것으로 기술되지만, 설계 보호 시스템(110)은 예를 들면 뷰들(102, 104)을 포함한 대부분의 어떠한 뷰들도 보호하기 위해 작동할 수 있다. 맵핑 키(500)의 각 라인은 SPEF(104) 내의 인스턴스 이름(처음 두개의 "!" 사이로서 정의됨)을 지정하는 식별자(502)와, 식별자(502)를 대체하는 인코딩된 식별자(504)(두번째와 세번째 "!" 사이로 정의됨)를 갖는다. 예를 들어, 라인(506) 내의 인스턴스 이름(502)(즉, 이름 맵(202)의 라인 "*12"에서 발견되는 "and_gate_3")은 라인(506) 내에 도시된 고유 식별자(504)(즉, "JWW12JWW")로 대체된다. 이 예에서, 인코딩된 식별자(504)는, 인스턴스 이름이 식별되고 미리 선택된 문자들이 그에 덧붙여지거나 부가되어 있는 SPEF ILM(106)(SPEF(104)로부터 변환 후) 내의 라인을 식별함으로써 생성되었다. 당업자들은 네트리스트 내의 각각의 인스턴스 이름, 네트 이름 및 계층적 연관

에 고유의 그리고 비공유의 식별자가 주어지는 한, 어떠한 이름짓기 또는 인코딩 스킴도 이용될 수 있다는 것을 이해할 것이다. 맵핑 키(500)는 또한 인코딩된 인스턴스 이름 및 인코딩되지 않은 인스턴스 이름 둘다를 이용하여 네트 이름의 인코딩을 제공한다. 예를 들어, 도 5의 라인(508)에 도시된 바와 같이, 고유의 식별자 "JWW17JWW"는 이전에 재명명된 인스턴스 "JWW12JWW"와 셀 "nand_gate"를 연결하는 네트를 식별하기 위해 생성된다.

[0038] 바람직한 실시예에서, 설계 보호 시스템(110)은 인코딩된 인스턴스 이름들과 네트 이름들을 식별하기 위해 이름 맵(SPEF(200) 내의 이름 맵(202)으로부터 유도됨)이 있는지 SPEF ILM(106)을 스캔하는 스크립트에 의해 구현된다. 스캔으로부터, 설계 보호 시스템(110)은 네트리스트 내에 포함된 이름들 또는 다른 기밀 정보와 관련되지 않은 명명법을 이용하여, 각 식별된 이름을 유일하게 식별하기 위한 인코딩 맵(500)을 생성한다. 그런 다음, 설계 보호 시스템(110)은, 선택된 민감하거나 또는 기밀의 정보가 네트리스트로부터 제거되었지만 완전히 기능적인 ILM 설계 뷰로서, 보호된 SPEF ILM(112)을 제공한다.

[0039] 도 6은 본 발명의 바람직한 실시예에 따른, 보호된 SPEF의 예를 도시한다. 이 예에서, 도 2의 예의 라인 "*12"-"*17"의 이름 맵(202)은 기밀 정보인 것으로 결정되었고, 보호된 SPEF(600)를 생성하기 위해 맵핑 키(500)를 이용하여 고유의 코드들(602)로 인코딩되었다.

[0040] 그런 다음, 설계 보호 시스템(110)은 동일한 맵핑 키(500)를 이용하여 코어 설계의 모든 다른 연관된 설계 뷰들을 보호하기 위해 진행한다. 바람직한 실시예에서, 설계 보호 시스템(110)은 맵핑 키(500)를 이용하여 Verilog ILM(108)으로부터 보호된 Verilog ILM(114)을 생성한다.

[0041] 맵핑 키(500)를 Verilog(300)에 적용함으로써 생성된 보호된 Verilog 파일의, HDL 네트리스트(700)로서 구현된, 예가 도 7에 도시된다. Verilog(300) 내의 네트들(304)은, 보호된 네트 이름들(702)로 도시된, 고유하게 명명된 연결들(connections)로 인코딩되어 있다. 마찬가지로, 인스턴스 이름들(306)이 맵핑 키(500)에 따라 보호된 인스턴스 이름들(704)로 인코딩되어 있다. 예를 들어, 인스턴스 이름들(306) 중 첫번째 "NAND3x1"은 비연관된 인스턴스 이름 "JWW17JWW"와 고유의 식별된 출력 "JWW30JWW"을 이용하여 제1 보호된 인스턴스 이름(704의)으로 인코딩되어 있다.

[0042] 도 8을 참조하면, 본 발명의 바람직한 실시예에 따른, EDA 툴의 설계 뷰들에서 기밀 정보를 보호하기 위한 프로세스의 순서도가 도시된다. 프로세스(800)는 단계(802)에서 IC 설계를 기술하는 HDL 코드가 계층적 구조를 제거하도록 펼쳐지는 것로부터 시작한다. 예를 들어, 시뮬레이션 목적으로 제3자 측에 전달될 파일들을 보호할 때, SPEF 및 Verilog HDL 파일들이 펼쳐진다. 그런 다음 프로세스는 단계(804)로 진행하여, ILM 모델들이 생성된다. 그런 다음 프로세스는 단계(806)로 진행하여 보호될 신호 이름들이 식별된다. 여기서 프로세스는 SPEF 파일에서 유지하거나 또는 재명명될 신호 이름들의 개수와 위치를 결정하기 위해 SPEF 파일을 파스(parse)할 것이다. 예를 들어, SPEF 파일은, 모듈 포트들이 그들의 원래 형태로 보존되도록 하기 위해, 모듈 포트들의 개수 및 위치를 결정하기 위해 파스될 것이다.

[0043] 그 후, 프로세스는 단계(808)로 진행하여, 선택된 신호 이름들을, 원래 신호 이름들과 관련되지는 않으나 각 신호 이름이 다른 것들로부터 식별되도록 유일하게 식별하는 고유의 식별자들로 맵핑함으로써, 신호 이름들을 보호된 신호 이름들과 연관시키는 맵핑 파일이 생성된다. 바람직한 실시예에서 포트 이름들 내에 포함된 신호 이름들은 맵핑되거나 대체되지 않는다. 그런 다음 프로세스는 단계(810)로 진행하여, 원래 신호 이름들의 인스턴스들이 있는지 파일을 스캔하고 그들을 그들의 맵핑된 보호된 이름들로 대체함으로써, SPEF 파일이 보호된 신호 이름들로 인코딩된다. 그 후, 프로세스는 단계(812)로 진행하여, 신호 이름들을 그들의 고유한 보호된 신호 이름들로 맵핑하기 위한 코드를 포함하는 스크립트 파일이 생성된다. 그 후, 프로세스는 단계(814)로 진행하여, 스크립트 파일이, 첫째 파일과 동일한 신호 이름들을 포함하는 Verilog 설계 파일들, 및 IC의 다른 지원 설계 뷰들 상에 실행된다. 실행 시에, 스크립트는 대상의 설계 뷰 파일에 단계들(806, 808, 810)을 수행하여, 선택된 신호 이름들을 그들의 고유한, 맵핑된, 보호된 신호 이름들로 대체함으로써, 선택된 신호 이름들의 각 인스턴스를 보호한다. 프로세스는 각 설계 뷰 사이에서 일관성을 유지하며 설계 뷰들의 제3자 측 수취인들에 의한 EDA 툴들의 완전한 유용성을 제공하면서, 모듈 이름들과 구조들을 숨기는 보호된 포맷으로 인코딩된 SPEF, Verilog 및 다른 지원되는 설계 뷰들을 출력함으로써 단계(816)를 종료한다.

[0044] 이해하게 되는 바와 같이, 본 발명의 바람직한 실시예는, 다중 IC 설계 뷰들에 있어서, 다중 뷰들에 걸쳐 표준 EDA 툴들에 의한 활용을 허용하기 위한 필수적인 정보의 손실이 없이, 보호된 설계 뷰들을 제공하기 위한 효율적인 방식으로 기밀 정보가 필터링되도록 해준다. 보호된 설계 뷰들의 인코딩이 맵핑 키를 이용하여 수행되기 때문에, 원래의 설계자는 보호된 파일 설계 뷰의 사용자로부터의 코멘트들이나 기술적인 문제들을 용이하게 해

결할 수 있다. 이것은 기술적인 설계에 관한 문제들 및 의문들에 대하여 보호된 설계 뷰들과 상호작용하는 소비자들을 지원하기 위해 중요하다.

[0045] 이해하게 되는 바와 같이, 본 발명의 바람직한 실시예들에서 프로세스들은 소프트웨어, 펌웨어 또는 하드웨어를 프로그래밍하는 컴퓨터의 임의의 조합을 이용하여 구현될 수 있다. 본 발명을 소프트웨어로 실시하기 위한 예비 단계로서, 바람직한 실시예에 따른 컴퓨터 프로그래밍 코드(소프트웨어든지 펌웨어든지 간에)는 고정식(하드) 드라이브, 디스켓, 광 디스크, 자기 테이프, 반도체 메모리, 예를 들면 ROM이나, PROM들 등등의 하나 또는 그 이상의 머신 판독가능 매체들에 전형적으로 저장될 것이며, 이에 의해 본 발명에 따른 제품이 만들어진다. 컴퓨터 프로그래밍 코드를 포함하는 제품은 저장 장치로부터 직접 코드를 실행하거나, 또는 저장 장치로부터 하드 디스크, RAM 등등과 같은 다른 저장 장치에 코드를 복사하거나, 또는 원격 실행을 위해 코드를 전송함으로써 이용된다. 본 발명의 방법 형태는 본 발명에 따른 코드를 포함하는 하나 또는 그 이상의 머신 판독가능 저장 장치들을, 거기에 포함된 코드들을 실행하기 위한 적당한 표준 컴퓨터 하드웨어와 결합함으로써 실시될 수 있다. 본 발명을 실시하기 위한 장치는 본 발명에 따라 코딩된 컴퓨터 프로그램(들)을 갖거나 그에 대한 네트워크 액세스를 갖는 하나 또는 그 이상의 컴퓨터와 저장 시스템들일 수 있다.

[0046] 본 발명은 바람직한 실시예들을 참조하여 구체적으로 도시되고 설명되었지만, 당업자들은, 본 발명의 사상과 범위를 벗어나지 않고서, 형태와 상세한 사항들에 다양한 변경들이 이루어질 수 있다는 것을 알 것이다. 설명된 실시예들에 대한 임의의 변형, 변경, 부가, 및 개량들이 가능하며, 이들은 다음의 청구항들에 열거된 본 발명의 범위 내에 들어올 수 있다.

도면의 간단한 설명

[0007] 본 발명은 유사한 번호들이 동일하거나 유사한 엘리먼트들을 나타내는 첨부 도면들을 참조하여 다음의 설명의 바람직한 실시예에서 설명된다.

[0008] 도 1은 본 발명의 일 실시예의 블록도이다.

[0009] 도 2는 플랫 게이트-레벨(flat gate-level) SPEF의 예이다.

[0010] 도 3은 베릴로그(Verilog) 뷰 내의 모듈이나 코어 설계의 예를 도시한다.

[0011] 도 4는 인스턴스 이름들(Instance Names)의 예시적인 포맷을 도시한다.

[0012] 도 5는 본 발명의 바람직한 실시예에 따른 설계 보호 시스템에 의해 수행되는 대체 스킵의 맵핑을 도시한다.

[0013] 도 6은 본 발명의 바람직한 실시예에 따른, 보호된 SPEF의 예를 도시한다.

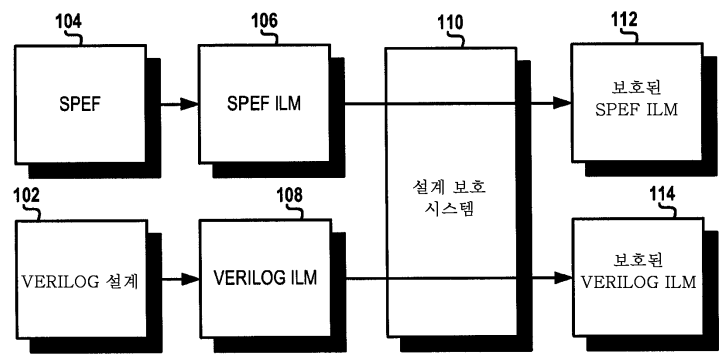
[0014] 도 7은 본 발명의 바람직한 실시예에 따른, 보호된 베릴로그(Verilog)를 도시한다.

[0015] 도 8은 본 발명의 바람직한 실시예에 따른, EDA 툴의 설계 뷰들에서 기밀 정보를 보호하기 위한 프로세스의 순서도를 도시한다.

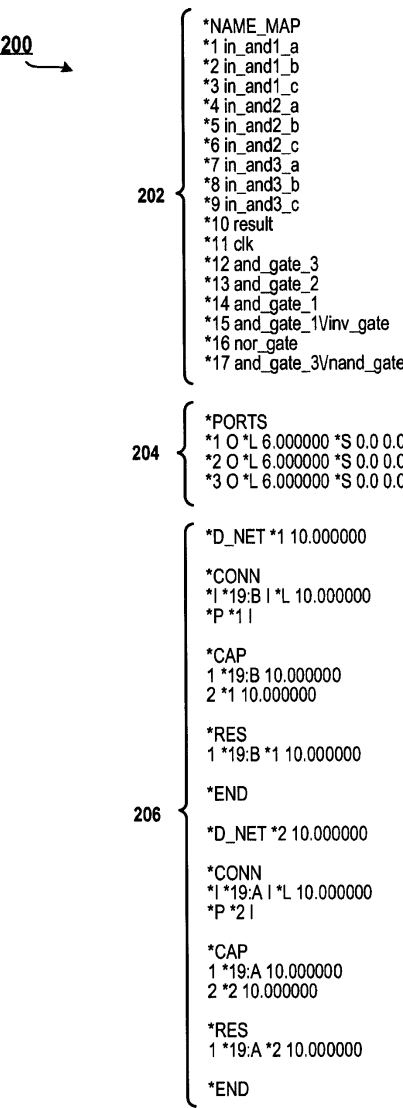
[0016] 본 발명의 예시적인 실시예들에 대한 다음의 상세한 설명에서, 본 발명이 실시될 수 있는 특정한 예시적인 실시예들을 도해하는 방식으로 도시되고, 본 명세서의 일부분을 형성하는 첨부 도면들을 참조한다.

도면

도면1



도면2



도면3

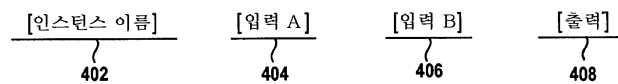
```

302 { module top_example ( in_and1_a, in_and1_b, in_and1_c, in_and2_a, in_and2_b,
                                in_and2_c, in_and3_a, in_and3_b, in_and3_c, result, clk );
    input in_and1_a, in_and1_b, in_and1_c, in_and2_a,
        in_and2_b, in_and2_c, in_and3_a, in_and3_b, in_and3_c, clk;
    output result;

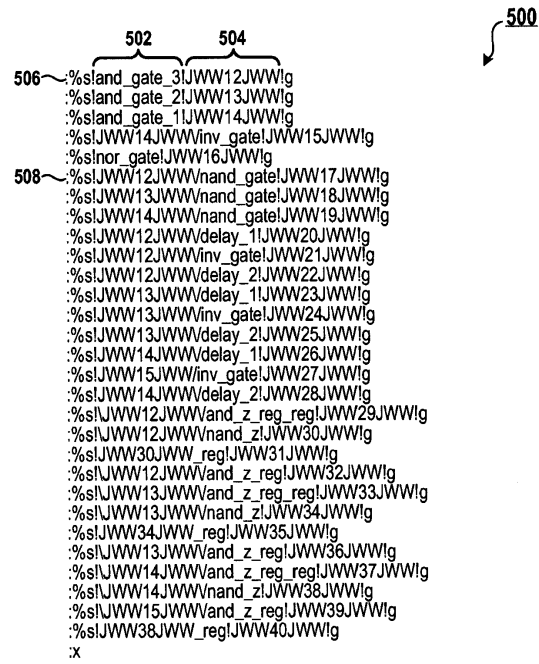
304 { wire \and_gate_3/and_z_reg_reg ;
        wire \and_gate_3/nand_z ;
        wire \and_gate_3/nand_z_reg ;
        wire \and_gate_3/and_z_reg ;
        wire \and_gate_2/and_z_reg_reg ;
        wire \and_gate_2/nand_z ;
        wire \and_gate_2/nand_z_reg ;
        wire \and_gate_2/and_z_reg ;
        wire \and_gate_1/and_z_reg_reg ;
        wire \and_gate_1/nand_z ;
        wire \and_gate_1/inv_gate/and_z_reg ;
        wire \and_gate_1/nand_z_reg ;

        NAND3x1 \and_gate_3/nand_gate ( .A(in_and3_b), .B(in_and3_a), .C(in_and3_c), Z(\and_gate_3/
            nand_z ));
        NAND3x2 \and_gate_2/nand_gate ( .A(in_and2_b), .B(in_and2_a), .C(in_and2_c), Z(\and_gate_2/
            nand_z ));
        NAND3x1 \and_gate_1/nand_gate ( .A(in_and1_b), .B(in_and1_a), .C(in_and1_c), Z(\and_gate_1/
            nand_z ));
        DFFx1 \and_gate_3/delay_1 ( .D(\and_gate_3/nand_z ), .CK(clk), .Q(\and_gate_3/nand_z_reg ));
        INVx8 \and_gate_3/inv_gate ( .A(\and_gate_3/nand_z_reg ), .Z(\and_gate_3/and_z_reg ));
        DFFx1 \and_gate_3/delay_2 ( .D(\and_gate_3/and_z_reg ), .CK(clk), .Q(\and_gate_3/
            and_z_reg_reg ));
306 { DFFx1 \and_gate_2/delay_1 ( .D(\and_gate_2/nand_z ), .CK(clk), .Q(\and_gate_2/nand_z_reg ));
        INVx1 \and_gate_2/inv_gate ( .A(\and_gate_2/nand_z_reg ), .Z(\and_gate_2/and_z_reg ));
        DFFx1 \and_gate_2/delay_2 ( .D(\and_gate_2/and_z_reg ), .CK(clk), .Q(\and_gate_2/
            and_z_reg_reg ));
        DFFx1 \and_gate_1/delay_1 ( .D(\and_gate_1/nand_z ), .CK(clk), .Q(\and_gate_1/nand_z_reg ));
        INVx1 \and_gate_1/inv_gate/inv_gate ( .A(\and_gate_1/nand_z_reg ), .Z(\and_gate_1/inv_gate
            and_z_reg ));
        DFFx1 \and_gate_1/delay_2 ( .D(\and_gate_1/inv_gate/and_z_reg ), .CK(clk),
            .Q(\and_gate_1/and_z_reg_reg ));
        NOR3x1 nor_gate ( .A(\and_gate_2/and_z_reg_reg ), .B(\and_gate_3/and_z_reg_reg ), .C(\
            and_gate_1/and_z_reg_reg ), .Z(result) );
    endmodule
    
```

도면4



도면5



도면6

*NAME_MAP
 *1 in_and1_a
 *2 in_and1_b
 *3 in_and1_c
 *4 in_and2_a
 *5 in_and2_b
 *6 in_and2_c
 *7 in_and3_a
 *8 in_and3_b
 *9 in_and3_c
 *10 result
 *11 clk
 {
 *12 JWW12JWW
 *13 JWW13JWW
 *14 JWW14JWW
 *15 JWW15JWW
 *16 JWW16JWW
 *17 JWW17JWW
 }
 *PORTS
 *1 O *L 6.000000 *S 0.0 0.0
 *2 O *L 6.000000 *S 0.0 0.0
 *3 O *L 6.000000 *S 0.0 0.0

 *D_NET *1 10.000000
 *CONN
 *I *19:B I *L 10.000000
 *P *1 I
 *CAP
 1 *19:B 10.000000
 2 *1 10.000000
 *RES
 1 *19:B *1 10.000000
 *END
 *D_NET *2 10.000000
 *CONN
 *I *19:A I *L 10.000000
 *P *2 I
 *CAP
 1 *19:A 10.000000
 2 *2 10.000000
 *RES
 1 *19:A *2 10.000000
 *END

600

602

도면7

```

module top_example ( in_and1_a, in_and1_b, in_and1_c, in_and2_a, in_and2_b,
                     in_and2_c, in_and3_a, in_and3_b, in_and3_c, result, clk );
input in_and1_a, in_and1_b, in_and1_c, in_and2_a, in_and2_b, in_and2_c,
    in_and3_a, in_and3_b, in_and3_c, clk;
output result;

702 { wire JWW29JWW;
    wire JWW30JWW;
    wire JWW31JWW;
    wire JWW32JWW;
    wire JWW33JWW;
    wire JWW34JWW;
    wire JWW35JWW;
    wire JWW36JWW;
    wire JWW37JWW;
    wire JWW38JWW;
    wire JWW39JWW;
    wire JWW40JWW;

704 { NAND3x1 JWW17JWW ( .A(in_and3_b), .B(in_and3_a), .C(in_and3_c), .Z(JWW30JWW));
    NAND3x2 JWW18JWW ( .A(in_and2_b), .B(in_and2_a), .C(in_and2_c), .Z(JWW34JWW));
    NAND3x1 JWW19JWW ( .A(in_and1_b), .B(in_and1_a), .C(in_and1_c), .Z(JWW38JWW));
    DFFx1 JWW20JWW ( .D(JWW30JWW), .CK(clk), .Q(JWW31JWW));
    INVx8 JWW21JWW ( .A(JWW31JWW), .Z(JWW32JWW));
    DFFx1 JWW22JWW ( .D(JWW32JWW), .CK(clk), .Q(JWW29JWW));
    DFFx1 JWW23JWW ( .D(JWW34JWW), .CK(clk), .Q(JWW35JWW));
    INVx1 JWW24JWW ( .A(JWW35JWW), .Z(JWW36JWW));
    DFFx1 JWW25JWW ( .D(JWW36JWW), .CK(clk), .Q(JWW33JWW));
    DFFx1 JWW26JWW ( .D(JWW38JWW), .CK(clk), .Q(JWW40JWW));
    INVx1 JWW27JWW ( .A(JWW40JWW), .Z(JWW39JWW));
    DFFx1 JWW28JWW ( .D(JWW39JWW), .CK(clk), .Q(JWW37JWW));
    NOR3x1 JWW16JWW ( .A(JWW33JWW), .B(JWW29JWW), .C(JWW37JWW), .Z(result));
endmodule

```

도면8

