(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2016년07월15일
(11) 등록번호 10-1640295
(24) 등록일자 2016년07월11일

(51) 국제특허분류(Int. Cl.)
G06F 9/45 (2006.01)
(21) 출원번호 10-2013-7022309
(22) 출원일자(국제) 2012년01월24일
심사청구일자 2015년06월11일
(85) 번역문제출일자 2013년08월23일
(65) 공개번호 10-2014-0006913
(43) 공개일자 2014년01월16일
(86) 국제출원번호 PCT/US2012/022435
(87) 국제공개번호 WO 2012/103143
국제공개일자 2012년08월02일
(30) 우선권주장
61/436,013 2011년01월25일 미국(US)
(56) 선행기술조사문헌
US20020091512 A1
WO2008073824 A1
US20040133869 A1
US20070296458 A1

(73) 특허권자
마이크론 테크놀로지, 인크.
미국, 아이다호, 보이세, 사우스 페더럴 웨이 8000
(72) 발명자
글렌데닝, 폴
미국 94062 캘리포니아주 우드사이드 블레이크우드 웨이 323
수, 준주안
미국 95131 캘리포니아주 산 호세 쿠아드로스 레인 2008
(74) 대리인
양영준, 백만기

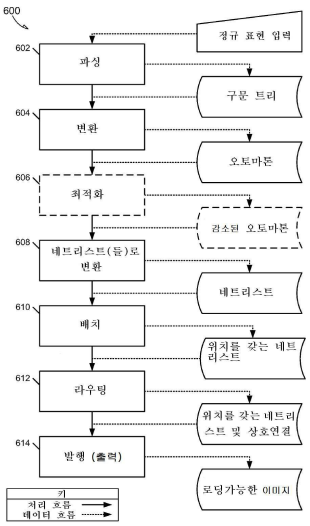
전체 청구항 수 : 총 29 항 심사관 : 유진태

(54) 발명의 명칭 **정규 표현을 컴파일하기 위한 방법 및 장치**

(57) 요 약

컴파일러를 위한 장치, 시스템 및 방법이 개시된다. 하나의 이러한 컴파일러는 소스 코드를 상태와 이 상태들 사이에 전이를 포함하는 오토마톤으로 변환하며, 여기서 상기 오토마톤의 상태는 특수 목적 하드웨어 요소에 대응하는 특수 목적 상태를 포함한다. 컴파일러는 오토마톤을 네트리스트로 변환하고, 타깃 디바이스를 구성하기 위한 기계 코드를 제공하기 위해 이 네트리스트를 배치하고 라우팅한다.

대 표 도 - 도6



명 세 서

청구범위

청구항 1

컴퓨터로서,

명령을 저장한 메모리; 및

상기 메모리에 통신가능하게 연결된 프로세서를 포함하되,

상기 명령은 상기 프로세서에 의해 실행될 때 상기 프로세서로 하여금,

소스 코드를 상태 및 상기 상태들 사이의 전이를 포함하는 오토마톤(automaton)으로 변환하는 동작으로서, 상기 오토마톤의 상태는 특수 목적 하드웨어 요소에 대응하는 특수 목적 상태를 포함하는 것인, 상기 변환하는 동작;

상기 오토마톤을 네트리스트(netlist)로 변환하는 동작; 및

타깃 디바이스를 구성하기 위한 기계 코드를 제공하기 위해 상기 네트리스트를 배치하고 라우팅하는 동작을 수행하게 하는 것인 컴퓨터.

청구항 2

제1항에 있어서, 상기 소스 코드는 양화를 포함하고,

상기 프로세서로 하여금 소스 코드를 변환하게 하는 동작은 상기 양화가 특수 목적 하드웨어 요소로 맵핑될 조건을 충족할 때 상기 양화를 상기 특수 목적 상태를 포함하는 복수의 상태로 변환하는 동작을 포함하는 것인 컴퓨터.

청구항 3

제2항에 있어서, 상기 프로세서로 하여금 소스 코드를 변환하게 하는 동작은 상기 양화가 특수 목적 하드웨어 요소로 맵핑될 조건을 충족하지 않을 때 상기 양화를 복수의 일반 목적 상태로 언롤링하는 동작을 포함하는 것인 컴퓨터.

청구항 4

제3항에 있어서, 언롤링하는 동작은 상기 오토마톤의 진입 차수(in-degree)를 제어하도록 상기 양화를 언롤링하는 동작을 포함하는 것인 컴퓨터.

청구항 5

제1항에 있어서, 상기 프로세서로 하여금 상기 오토마톤의 최적화를 수행하게 하는 명령을 더 포함하되, 상기 최적화하는 동작은 상기 오토마톤의 특정 상태의 예측된 진입 차수가 상기 타깃 디바이스의 제약을 초과할 때 상기 오토마톤의 특정 상태를 다수의 상태로 분할하는 동작을 포함하는 것인 컴퓨터.

청구항 6

제5항에 있어서, 상기 특정 상태를 분할하는 동작은 상기 다수의 상태 각각의 진입 차수가 제약을 충족하도록 상기 특정 상태의 구동 상태를 상기 다수의 상태로 분배하는 것을 포함하는 것인 컴퓨터.

청구항 7

제1항에 있어서, 상기 프로세서로 하여금 상기 오토마톤을 네트리스트로 변환하게 하는 동작은 상기 상태를 상기 네트리스트의 인스턴스(instance)로 맵핑하는 동작을 포함하며, 상기 맵핑하는 동작은 상기 특수 목적 상태를 특수 목적 요소에 대응하는 특수 목적 인스턴스로 맵핑하는 동작을 포함하는 것인 컴퓨터.

청구항 8

제7항에 있어서, 상기 프로세서로 하여금 상기 오토마톤을 네트리스트로 변환하게 하는 동작은 상기 타깃 디바이스의 물리적 설계에 기초하여 상태를 함께 그룹화하는 동작을 포함하는 것인 컴퓨터.

청구항 9

제7항에 있어서, 상기 인스턴스는 상태 기계 요소(state machine element: SME) 하드웨어 요소에 대응하는 상태 기계 요소(SME) 인스턴스와, SME 그룹을 포함하는 하드웨어 요소에 대응하는 SME 그룹 인스턴스를 포함하고, 그룹화하는 것은 상태를 SME 그룹 인스턴스로 그룹화하는 것을 포함하는 것인 컴퓨터.

청구항 10

컴퓨터로 구현되는 방법으로서,

하나 이상의 프로세서를 사용하여, 소스 코드를 구문 트리(syntax tree)로 파싱(parsing)하는 단계;

상기 하나 이상의 프로세서를 사용하여, 상기 구문 트리를 오토마톤으로 변환하는 단계로서, 상기 오토마톤은 복수의 상태와 상기 복수의 상태들 사이의 전이를 가지는 거동 모델을 한정하며, 상기 오토마톤의 구조는 타깃 하드웨어 디바이스에 의해 지시되는 것인, 상기 오토마톤으로 변환하는 단계;

상기 하나 이상의 프로세서를 사용하여, 상기 오토마톤을 네트리스트로 변환하는 단계로서, 상기 네트리스트는 복수의 인스턴스를 포함하며, 각 인스턴스는 타깃 하드웨어 디바이스의 하드웨어 요소에 대응하는 것인, 상기 네트리스트로 변환하는 단계;

상기 하나 이상의 프로세서를 사용하여, 상기 인스턴스 각각을 배치하는 단계;

상기 하나 이상의 프로세서를 사용하여, 상기 네트리스트의 함수로서 상기 하드웨어 요소들 사이에 연결을 라우팅하는 단계; 및

배치하는 단계와 라우팅하는 단계에 기초하여 상기 타깃 하드웨어 디바이스를 프로그래밍하는데 사용되는 프로그래밍 데이터를 생성하는 단계를 포함하되,

상기 배치하는 단계는 상기 네트리스트의 각 인스턴스를 상기 타깃 하드웨어 디바이스의 하드웨어 요소에 할당하는 단계를 포함하는 것인 컴퓨터로 구현되는 방법.

청구항 11

제10항에 있어서, 상기 구문 트리를 오토마톤으로 변환하는 단계는 상기 소스 코드의 양화를 상기 타깃 하드웨어 디바이스의 카운터 요소에 대응하는 특수 목적 상태를 포함하는 복수의 상태로 변환하는 단계를 포함하는 것인 컴퓨터로 구현되는 방법.

청구항 12

제10항에 있어서, SME에 대응하는 복수의 일반 목적 상태는 GOT(Group of Two) 하드웨어 요소의 출력 제한에 기초하여 GOT 인스턴스를 형성하도록 함께 그룹화된 것인 컴퓨터로 구현되는 방법.

청구항 13

제10항에 있어서, 상기 구문 트리를 오토마톤으로 변환하는 단계는 상기 오토마톤의 진입 차수를 제한하는 단계를 포함하되, 상기 진입 차수를 제한하는 단계는 상기 오토마톤의 상태로의 전이 개수를 제한하는 것을 포함하는 것인 컴퓨터로 구현되는 방법.

청구항 14

제13항에 있어서, 상기 진입 차수를 제한하는 단계는 특정 상태를 다수의 상태로 분할하는 단계와, 상기 다수의 상태 각각의 진입 차수가 제약을 충족하도록 상기 특정 상태의 구동 상태를 상기 다수의 상태로 분배하는 단계를 포함하는 것인 컴퓨터로 구현되는 방법.

청구항 15

제13항에 있어서, 상기 진입 차수를 제한하는 단계는 양화를 복수의 언롤링된 상태로 언롤링하는 단계와, 상기 언롤링된 상태 중 어느 것에 대해 진입 차수의 개수를 제한하는 단계를 포함하는 것인 컴퓨터로 구현되는 방법.

청구항 16

프로그래밍가능한 디바이스로서,

하나 이상의 입력과 하나 이상의 출력을 포함하는 복수의 프로그래밍가능한 요소;

외부 회로에 대해 상기 복수의 프로그래밍가능한 요소의 일부를 인터페이싱하기 위한 입력 블록과 출력 블록;

상기 복수의 프로그래밍가능한 요소와 상기 입력 블록과 상기 출력 블록을 통신가능하게 연결하는 복수의 프로그래밍가능한 스위치; 및

상기 복수의 프로그래밍가능한 요소와 상기 복수의 프로그래밍가능한 스위치를 구성할 수 있는 프로그래밍 데이터를 저장하도록 구성된 복수의 레지스터를 포함하되,

하나 이상의 프로그래밍가능한 스위치를 설정하는 것은 상기 복수의 프로그래밍가능한 요소와 상기 복수의 프로그래밍가능한 스위치 중 임의의 2개 이상 사이에 신호 라우팅을 선택적으로 제어하고,

상기 프로그래밍 데이터는,

하나 이상의 프로세서를 사용하여, 소스 코드를 구문 트리로 파싱하는 동작;

상기 하나 이상의 프로세서를 사용하여, 상기 구문 트리를 오토마톤으로 변환하는 동작으로서, 상기 오토마톤은 복수의 상태와 상기 복수의 상태들 사이의 전이를 가지는 거동 모델을 한정하고, 상기 오토마톤의 구조는 타깃 디바이스에 의해 지시되는 것인, 상기 오토마톤으로 변환하는 동작;

상기 하나 이상의 프로세서를 사용하여, 상기 오토마톤을 네트리스트로 변환하는 동작으로서, 상기 네트리스트는 복수의 인스턴스를 포함하며, 각 인스턴스는 상기 타깃 디바이스의 하드웨어 요소에 대응하는 것인, 상기 네트리스트로 변환하는 동작;

상기 하나 이상의 프로세서를 사용하여, 각 인스턴스를 배치하는 동작;

상기 하나 이상의 프로세서를 사용하여, 상기 네트리스트의 함수로서 상기 하드웨어 요소들 사이에 연결을 라우팅하는 동작; 및

배치하는 동작과 라우팅하는 동작에 기초하여 상기 타깃 디바이스를 프로그래밍하는데 사용되는 프로그래밍 데이터를 생성하는 동작

에 의해 생성되며,

배치하는 동작은 상기 네트리스트의 각 인스턴스를 상기 타깃 디바이스의 하드웨어 요소에 할당하는 동작을 포함하는 것인 프로그래밍가능한 디바이스.

청구항 17

일시적 전파 신호가 아닌 컴퓨터 판독가능 매체로서,

상기 컴퓨터 판독가능 매체는 명령을 포함하고,

상기 명령은, 하나 이상의 프로세서에 의해 구현될 때, 이하의 동작들:

소스 코드를 구문 트리로 파싱하는 동작;

상기 구문 트리를 오토마톤으로 변환하는 동작으로서, 상기 오토마톤은 복수의 상태와 상기 복수의 상태들 사이의 전이를 가지는 거동 모델을 한정하고, 상기 오토마톤의 구조는 타깃 디바이스에 의해 지시되는 것인, 상기 오토마톤으로 변환하는 동작;

상기 오토마톤을 네트리스트로 변환하는 동작으로서, 상기 네트리스트는 상기 타깃 디바이스와 연관된 복수의 하드웨어 요소를 포함하고, 상기 네트리스트는 상기 하드웨어 요소들 사이에 연결을 한정하는 것인, 상기 네트리스트로 변환하는 동작;

상기 하드웨어 요소들 각각을 배치하는 동작;

상기 네트리스트의 함수로서 상기 하드웨어 요소들 사이에 연결을 라우팅하는 동작; 및

배치하는 동작과 라우팅하는 동작을 반영하기 위해 상기 타깃 디바이스를 프로그래밍하는데 사용되는 프로그래

밍 데이터를 생성하는 동작을 수행하게 하며,

배치하는 동작은 상기 네트리스트의 각 하드웨어 요소를 상기 타깃 디바이스 내 위치에 할당하는 것을 포함하는 것인 컴퓨터 판독가능한 매체.

청구항 18

제17항에 있어서, 상기 구문 트리를 오토마톤으로 변환하는 동작은,

상기 소스 코드의 양화가 상기 타깃 디바이스의 카운터로 맵핑될 조건을 충족하는지 여부를 결정하는 동작;

상기 양화가 상기 조건을 충족할 때, 상기 양화를 카운터 상태를 포함하는 복수의 상태로 변환하는 동작; 및

상기 양화가 상기 조건을 충족하지 않을 때, 상기 양화를 언롤링하는 것에 의해 상기 양화를 복수의 SME 상태로 변환하는 동작을 포함하는 것인 컴퓨터 판독가능한 매체.

청구항 19

제18항에 있어서, 상기 양화가 상기 조건을 충족하는지 여부를 결정하는 동작은 상기 양화가 처리되고 있는 동안 상기 양화를 위한 구동 표현(drive expression)이 매치될 수 있는지 여부를 결정하는 동작을 포함하는 것인 컴퓨터 판독가능한 매체.

청구항 20

제19항에 있어서, 상기 양화가 상기 조건을 충족하는지 여부를 결정하는 동작은 상기 양화의 반복된 표현이 상기 양화의 다른 반복된 표현의 접두사인지 여부를 결정하는 동작을 포함하는 것인 컴퓨터 판독가능한 매체.

청구항 21

제20항에 있어서, 상기 양화를 카운터 상태를 포함하는 복수의 상태로 변환하는 동작은 상기 복수의 상태를 상기 양화와 상기 카운터 상태의 반복된 표현을 포함하는 루프로서 구현하는 동작을 포함하고, 상기 카운터 상태는 상기 반복된 표현이 매치되는 횟수를 카운트하도록 구성되고, 상기 카운터 상태는 상기 반복된 표현이 상기 양화에 의해 지정된 횟수만큼 매치될 때 다운스트림 상태를 활성화하는 것인 컴퓨터 판독가능한 매체.

청구항 22

제20항에 있어서, 언롤링하는 동작은 상기 타깃 디바이스의 진입 차수의 제약에 기초하여 상기 오토마톤의 진입 차수를 제어하도록 상기 양화를 언롤링하는 동작을 포함하는 것인 컴퓨터 판독가능한 매체.

청구항 23

제17항의 동작들 모두를 순차적으로 수행함으로써 생성된 프로그래밍 데이터를 사용하여 프로그래밍된 프로그래밍가능한 디바이스.

청구항 24

컴퓨터로 구현되는 방법으로서,

하나 이상의 프로세서를 사용하여, 소스 코드를 구문 트리로 파싱하는 단계;

상기 하나 이상의 프로세서를 사용하여, 상기 구문 트리를 오토마톤으로 변환하는 단계로서, 상기 변환하는 단계는

타깃 하드웨어 디바이스에 기초하여 상기 오토마톤의 구조를 제한하는 단계를 포함하고, 상기 타깃 하드웨어 디바이스는 GOT로 쌍으로 형성된 상태 기계 요소를 포함하는 것인, 상기 오토마톤으로 변환하는 단계;

상기 하나 이상의 프로세서를 사용하여, 상기 오토마톤을 네트리스트로 변환하는 단계로서, 상기 네트리스트는 상기 타깃 하드웨어 디바이스와 연관된 복수의 하드웨어 요소를 포함하고, 상기 네트리스트는 상기 하드웨어 요소들 사이에 연결을 한정하는 것인, 상기 네트리스트로 변환하는 단계;

상기 하나 이상의 프로세서를 사용하여, 각 하드웨어 요소를 배치하는 단계;

상기 하나 이상의 프로세서를 사용하여, 상기 네트리스트의 함수로서 상기 하드웨어 요소들 사이에 연결을 라우

팅하는 단계; 및

배치하는 단계 및 라우팅하는 단계를 반영하기 위해 상기 타깃 하드웨어 디바이스를 프로그래밍하는데 사용되는 복수의 비트를 생성하는 단계를 포함하되,

배치하는 단계는 상기 네트리스트의 각 하드웨어 요소를 상기 타깃 하드웨어 디바이스 내 위치에 할당하는 단계를 포함하는 것인 컴퓨터로 구현되는 방법.

청구항 25

제24항에 있어서, 구문 트리를 오토마톤으로 변환하는 단계는 공통 출력을 공유하는 GOT에 기초하여 오토마톤의 구조를 제한하는 단계를 포함하는 것인 컴퓨터로 구현되는 방법.

청구항 26

제24항에 있어서, 오토마톤의 구조를 제한하는 단계는 상기 타깃 하드웨어 디바이스의 카운터 요소에 기초하여 상기 오토마톤의 구조를 제한하는 단계를 포함하는 것인 컴퓨터로 구현되는 방법.

청구항 27

제24항에 있어서, 상기 복수의 비트를 생성하는 단계에 응답하여, 복수의 비트를 발행하는 단계를 더 포함하는 컴퓨터로 구현되는 방법.

청구항 28

제24항에 있어서, 상기 구문 트리를 오토마톤으로 변환하는 단계에 응답하여, 상기 오토마톤내의 복수의 상태를 감소시키기 위해 상기 오토마톤을 최적화하는 단계 - 상기 오토마톤내의 복수의 상태는 특수 목적 하드웨어 요소에 대응하는 특수 목적 상태를 포함함 - 를 더 포함하는, 컴퓨터로 구현되는 방법.

청구항 29

제24항의 방법에 의해 생성된 프로그래밍 데이터를 사용하여 프로그래밍된 프로그래밍가능한 디바이스.

발명의 설명

기 술 분 야

[0001] **우선권의 주장**

[0002] 본 특허 출원은 미국 가특허 출원 제61/436,013호(출원일: 2011년 1월 25일, 발명의 명칭: "METHOD AND APPARATUS FOR COMPILING REGULAR EXPRESSIONS")의 우선권의 이익을 주장하며, 이 기초 출원은 그의 전체 내용이 본 명세서에 참조로 병합된다.

배 경 기 술

[0003] 유한 상태 기계(finite state machine)(FSM)(이는 또한 유한 상태 오토마톤(automaton), 오토마톤, 또는 단순히 상태 기계라고도 지칭됨)는 상태(state), 상태와 동작 사이의 전이(transition)를 나타낸다. 유한 상태 기계는 병렬 기계(parallel machine)를 위한 디지털 논리, 컴퓨터 프로그램, 또는 이미지를 설계하는데 사용될 수 있다. 유한 상태 기계는 유한 개의 상태, 이들 상태들 사이의 전이, 및 출력으로 구성된 거동(behavior) 모델이다. 유한 상태 기계는 그래프로 표시될 수 있는데, 여기서 이 그래프의 정점(vertices)이 유한 상태 기계의 상태에 대응하고 그래프의 에지(edge)가 유한 상태 기계에 대한 하나 이상의 입력으로 인해 발생하는 상태들 사이의 전이에 대응한다. 유한 상태 기계는 확률 전이(probabilistic transition), 퍼지 상태(fuzzy state), 또는 다른 특이성(oddity)을 더 구비할 수 있다. 유한 상태 기계는 유한 내부 메모리, 입력 특징 및 선택적 출력 특징을 구비한다. 출력을 구비하는 유한 상태 기계는 유한 상태 트랜스듀서(transducer)라고 언급될 수 있다.

[0004] 유한 상태 기계의 응용은 전자 설계 오토마톤, 통신 프로토콜 설계, 생물 및 인공 지능 조사, 및 자연 언어의 문법을 기술하는 언어학을 포함한다.

도면의 간단한 설명

[0005] 도 1은 본 발명의 여러 실시형태에 따른 병렬 기계의 일례를 도시한 도면;

도 2는 본 발명의 여러 실시형태에 따라 유한 상태 기계 엔진으로 구현된 도 1의 병렬 기계의 일례를 도시한 도면;

도 3은 본 발명의 여러 실시형태에 따라 도 2의 유한 상태 기계 엔진의 블록의 일례를 도시한 도면;

도 4는 본 발명의 여러 실시형태에 따라 도 3의 블록의 행(row)의 일례를 도시한 도면;

도 5는 본 발명의 여러 실시형태에 따라 도 4의 2개의 행의 그룹의 일례를 도시한 도면;

도 6은 본 발명의 여러 실시형태에 따라 도 1의 병렬 기계를 프로그래밍하도록 구성된 이미지로 소스 코드(source code)를 변환하는 컴파일러를 위한 방법의 일례를 도시한 도면;

도 7은 본 발명의 여러 실시형태에 따라 구문 트리(syntax tree)를 오토마톤으로 변환하는 예시적인 방법에 포함될 수 있는 여러 동작을 도시한 도면;

도 8은 본 발명의 여러 실시형태에 따라 구문 트리를 오토마톤으로 변환하는 방법을 도시한 도면;

도 9는 본 발명의 여러 실시형태에 따라 특수 목적 카운터 상태를 가지는 예시적인 오토마톤을 도시한 도면;

도 10은 본 발명의 여러 실시형태에 따라 특수 목적 카운터 상태를 가지는 다른 예시적인 오토마톤을 도시한 도면;

도 11a 및 도 11b는 본 발명의 여러 실시형태에 따른 예시적인 오토마톤을 도시한 도면;

도 12는 본 발명의 여러 실시형태에 따라 무 접두사 조건(no prefix condition)를 충족하지 않는 양화(quantification)를 가지는 regex에 대한 예시적인 오토마톤을 도시한 도면;

도 13a 내지 도 13c는 본 발명의 여러 실시형태에 따라 언롤링된(unrolled) 양화를 위한 예시적인 오토마톤을 도시한 도면;

도 14a 내지 도 14b는 본 발명의 여러 실시형태에 따라 오토마톤의 진입 차수(in-degree)를 제한하는 일부로서 상태 분할을 도시한 도면;

도 15a 및 도 15b는 본 발명의 여러 실시형태에 따른 예시적인 네트리스트(netlist)를 도시한 도면;

도 16은 본 발명의 여러 실시형태에 따라 폰 노이만 아키텍처(Von Nuemann architecture)를 가지는 컴퓨터의 일례를 도시한 도면.

발명을 실시하기 위한 구체적인 내용

[0006] 이하 상세한 설명과 도면은 이 기술 분야에 통상의 지식을 가진 자로 하여금 본 발명을 실시할 수 있게 하는 특정 실시형태를 충분히 예시한다. 다른 실시형태는 구조적, 논리적, 전기적 공정, 및 다른 변화를 포함할 수 있다. 일부 실시형태의 부분과 특징은 다른 실시형태의 것에 포함되거나 이를 대체할 수 있다. 청구범위에 제시된 실시형태는 이 청구범위의 모든 이용가능한 균등물을 포함한다.

[0007] 본 문서는 특히 소스 코드(예를 들어, 정규 표현(regular expression))를 병렬 기계를 구성하는(예를 들어, 프로그래밍하는) 기계 코드(이미지)로 변환하는 컴파일러를 기술한다. 컴파일러에 의해 생성된 이미지(출력 파일)는 특정 기능을 수행하도록 병렬 기계를 프로그래밍할 수 있다. 특정 예에서, 병렬 기계는 유한 상태 기계(FSM) 엔진, 전계 프로그래밍가능한 게이트 어레이(FPGA) 및 이들의 변형을 포함할 수 있다.

[0008] 도 1은 타깃 디바이스의 일례(예를 들어, 병렬 기계(100))를 도시한다. 병렬 기계(100)는 입력 데이터를 수신하고 이 입력 데이터에 기초하여 출력을 제공할 수 있다. 병렬 기계(100)는 입력 데이터를 수신하는 데이터 입력 포트(110)와, 출력을 다른 디바이스에 제공하는 출력 포트(114)를 포함할 수 있다. 데이터 입력 포트(110)는 병렬 기계(100)에 입력될 데이터를 위한 인터페이스를 제공한다.

[0009] 병렬 기계(100)는 일반 목적 요소(general purpose element)(102)와 특수 목적 요소(special purpose element)(112)를 포함하는 복수의 프로그래밍 가능한 요소를 포함한다. 일반 목적 요소(102)는 하나 이상의 입력(104)과 하나 이상의 출력(106)을 포함할 수 있다. 일반 목적 요소(102)는 복수의 상태 중 하나로 프로그래밍될 수 있다. 일반 목적 요소(102)의 상태는 일반 목적 요소(102)의 어떤 출력(들)이 주어진 입력(들)에 기초하여 제공할 수 있는지를 결정한다. 즉, 일반 목적 요소(102)의 상태는 프로그래밍가능한 요소가 주어진 입력에

반응(예를 들어, 응답)할 수 있는 방식을 결정한다. 데이터 입력 포트(110)에의 데이터 입력은 복수의 일반 목적 요소(102)에 제공되어 일반 목적 요소(102)로 하여금 이에 동작을 취하도록 할 수 있다. 일반 목적 요소(102)의 예로는 예를 들어, 다른 프로그래밍가능한 요소들 중에서도 아래에서 상세히 설명된 상태 기계 요소(state machine element: SME), 카운터 및/또는 구성가능한 논리 블록을 포함할 수 있다. 일례에서, SME는 주어진 입력이 데이터 입력 포트(110)에 수신될 때 특정 출력(예를 들어, 하이(high) 또는 "1" 신호)을 제공하도록 프로그래밍(예를 들어, 설정)될 수 있다. 주어진 입력과는 다른 입력이 데이터 입력 포트(110)에 수신될 때 SME는 상이한 출력(예를 들어, 로우(low) 또는 "0" 신호)을 제공할 수 있다. 일례에서, 구성가능한 논리 블록은 데이터 입력 포트(110)에 수신된 입력에 기초하여 부울리안 논리 함수(Boolean logic function)(예를 들어, AND, OR, NOR 등)를 수행하도록 설정될 수 있다. 카운터의 일례가 본 명세서에서 차후 기술된다. 특수 목적 요소(112)는 메모리(예를 들어, RAM), 논리 게이트, 카운터, 룩업 테이블, 전계 프로그래밍가능한 게이트 어레이(FPGA), 및 다른 하드웨어 요소를 포함할 수 있다. 특수 목적 요소(112)는 일반 목적 요소(102)와 반응하며 특수 목적 기능을 수행할 수 있다.

[0010] 병렬 기계(100)는 병렬 기계(100)에 프로그램(예를 들어, 이미지)을 로딩하기 위해 프로그래밍 인터페이스(111)를 더 포함할 수 있다. 이미지는 일반 목적 요소(102)의 상태를 프로그래밍(예를 들어, 설정)할 수 있다. 즉, 이미지는 주어진 입력에 특정 방식으로 반응하도록 일반 목적 요소(102)를 구성할 수 있다. 예를 들어, 일반 목적 요소(102)는 데이터 입력 포트(110)에 문자 'a'가 수신될 때 하이 신호를 출력하도록 설정될 수 있다. 일부 예에서, 병렬 기계(100)는 일반 목적 요소(102)의 동작 타이밍을 제어하는 클록 신호를 사용할 수 있다. 일부 실시형태에서, 데이터 입력 포트(110)에 수신된 데이터는 시간에 따라 수신된 고정된 데이터 세트 또는 한 번에 모두, 또는 시간에 따라 수신된 데이터의 스트림을 포함할 수 있다. 데이터는 병렬 기계(100)에 연결된 데이터베이스, 센서, 네트워크 등과 같은 임의의 소스로부터 수신되거나 이 소스에 의해 생성될 수 있다.

[0011] 병렬 기계(100)는 병렬 기계(100)의 상이한 요소(예를 들어, 일반 목적 요소(102), 데이터 입력 포트(110), 출력 포트(114), 프로그래밍 인터페이스(111) 및 특수 목적 요소(112))를 서로 선택적으로 연결하는 복수의 프로그래밍가능한 스위치(108)를 더 포함한다. 따라서, 병렬 기계(100)는 요소들 중에서 형성된 프로그래밍가능한 매트릭스를 포함한다. 일례에서, 프로그래밍가능한 스위치(108)는 일반 목적 요소(102)의 입력(104), 데이터 입력 포트(110), 프로그래밍 인터페이스(111), 또는 특수 목적 요소(112)가 하나 이상의 프로그래밍가능한 스위치(108)를 통해 일반 목적 요소(102)의 출력(106), 출력 포트(114), 프로그래밍 인터페이스(111), 또는 특수 목적 요소(112)에 연결될 수 있도록 2개 이상의 요소를 서로 선택적으로 연결할 수 있다. 따라서, 요소들 사이에 신호를 라우팅(routing)하는 것은 프로그래밍가능한 스위치(108)를 설정하는 것에 의해 제어될 수 있다. 도 1은 주어진 요소와 프로그래밍가능한 스위치(108) 사이에 특정 개수의 전도체(예를 들어, 와이어)를 도시하지만, 다른 예에서, 상이한 개수의 전도체가 사용될 수 있는 것으로 이해된다. 또한, 도 1은 프로그래밍가능한 스위치(108)에 개별적으로 연결된 각 일반 목적 요소(102)를 도시하지만, 다른 예에서, 다수의 일반 목적 요소(102)는 프로그래밍가능한 스위치(108)에 그룹(예를 들어, 도 2에 도시된 바와 같이 블록(202))으로 연결될 수 있다. 일례에서, 데이터 입력 포트(110), 데이터 출력 포트(114) 및/또는 프로그래밍 인터페이스(111)는 레지스터로 구현되되 레지스터(register)에의 기록이 각 요소로부터 또는 각 요소로 데이터를 제공하도록 구현될 수 있다.

[0012] 일례에서, 단일 병렬 기계(100)는 물리적 디바이스 위에 구현되지만, 다른 예에서, 2개 이상의 병렬 기계(100)가 단일 물리적 디바이스(예를 들어, 물리적 칩) 위에 구현될 수 있다. 일례에서, 다수의 병렬 기계(100) 각각은 이산 데이터 입력 포트(110), 이산 출력 포트(114), 이산 프로그래밍 인터페이스(111) 및 일반 목적 요소(102)의 이산 세트를 포함할 수 있다. 나아가, 일반 목적 요소(102)의 각 세트는 대응하는 입력 데이터 포트(110)에서의 데이터와 반응할 수 있다(예를 들어, 하이 또는 로우 신호를 출력할 수 있다). 예를 들어, 제1 병렬 기계(100)에 대응하는 일반 목적 요소(102)의 제1 세트는 제1 병렬 기계(100)에 대응하는 제1 데이터 입력 포트(110)에서의 데이터와 반응할 수 있다. 제2 병렬 기계(100)에 대응하는 일반 목적 요소(102)의 제2 세트는 제2 병렬 기계(100)에 대응하는 제2 데이터 입력 포트(110)와 반응할 수 있다. 따라서, 각 병렬 기계(100)는 일반 목적 요소(102)의 세트를 포함하고, 일반 목적 요소(102)의 상이한 세트는 상이한 입력 데이터와 반응할 수 있다. 유사하게, 각 병렬 기계(100)와, 일반 목적 요소(102)의 각 대응하는 세트는 이산 출력을 제공할 수 있다. 일부 예에서, 제1 병렬 기계(100)로부터 출력 포트(114)는 제2 병렬 기계(100)의 입력 포트(110)에 연결되되, 제2 병렬 기계(100)에 대한 입력 데이터는 제1 병렬 기계(100)로부터 오는 출력 데이터를 포함할 수 있도록 연결될 수 있다.

[0013] 일례에서, 병렬 기계(100)에 로딩될 이미지는, 일반 목적 요소(102)의 상태를 설정하고 프로그래밍가능한 스위치(108)를 프로그래밍하고 병렬 기계(100) 내 특수 목적 요소(112)를 구성하기 위한 복수의 정보 비트(bit)를

포함한다. 일례에서, 이미지는 병렬 기계(100)에 로딩되어 병렬 기계(100)를 특정 입력에 기초하여 원하는 출력을 제공하도록 프로그래밍할 수 있다. 출력 포트(114)는 입력 포트(110)에 수신된 데이터에 대한 일반 목적 요소(102)의 반응에 기초하여 병렬 기계(100)로부터 출력을 제공할 수 있다. 출력 포트(114)로부터 출력은 주어진 패턴의 매치(match)를 나타내는 단일 비트, 복수의 패턴과 매치 및 비-매치를 나타내는 복수의 비트를 포함하는 워드(word), 및 주어진 순간에 모든 또는 특정 일반 목적 요소(102)의 상태에 대응하는 출력 벡터(output vector)를 포함할 수 있다.

[0014] 병렬 기계(100)에 대한 예시적인 사용은 패턴 인식(예를 들어, 음성 인식, 이미지 인식 등) 신호 처리, 이미징, 컴퓨터 비전, 암호화 등을 포함한다. 특정 예에서, 병렬 기계(100)는 유한 상태 기계(finite state machine: FSM) 엔진, 전계 프로그래밍가능한 게이트 어레이(field programmable gate array: FPGA) 및 이들의 변형을 포함할 수 있다. 나아가, 병렬 기계(100)는 컴퓨터, 페이저(pager), 셀폰(cellular phone), 퍼스널 오거나이저(personal organizer), 휴대용 오디오 플레이어, 네트워크 디바이스(예를 들어, 라우터, 방화벽, 스위치, 또는 이들의 임의의 조합), 제어 회로, 카메라 등과 같은 더 큰 디바이스의 성분(component)일 수 있다.

[0015] 도 2 내지 도 5는 유한 상태 기계(FSM) 엔진(200)으로 구현된 다른 병렬 기계를 도시한다. 일례에서, FSM 엔진(200)은 유한 상태 기계의 하드웨어 구현을 포함한다. 따라서, FSM 엔진(200)은 FSM의 복수의 상태에 대응하는 복수의 선택적으로 연결가능한 하드웨어 요소(예를 들어, 프로그래밍가능한 요소)를 구현한다. FSM의 상태와 유사하게, 하드웨어 요소는 입력 스트림을 분석하고 입력 스트림에 기초하여 다운스트림 하드웨어 요소를 작동시킬 수 있다.

[0016] FSM 엔진(200)은 일반 목적 요소와 특수 목적 요소를 포함하는 복수의 프로그래밍가능한 요소를 포함한다. 일반 목적 요소는 많은 상이한 기능을 구현하도록 프로그래밍될 수 있다. 일반 목적 요소는 행(206)(도 3 및 도 4에 도시) 및 블록(202)(도 2 및 도 3에 도시)으로 계층적으로(hierarchically) 구성된 SME(204, 205)(도 5에 도시)를 포함한다. 계층적으로 구성된 SME(204, 205) 사이에 신호를 라우팅하기 위해 프로그래밍가능한 스위치의 계층이 사용되는데, 이들 스위치에는 블록간(inter-block) 스위치(203)(도 2 및 도 3에 도시), 블록 내부(intra-block) 스위치(208)(도 3 및 도 4에 도시) 및 행 내부(intra-row) 스위치(212)(도 4)를 포함한다. SME(204, 205)는 FSM 엔진(200)으로 구현된 FSM의 상태에 대응할 수 있다. SME(204, 205)는 후술되는 바와 같이 프로그래밍가능한 스위치를 사용하여 서로 연결될 수 있다. 따라서, FSM은, 상태의 함수에 대응하도록 SME(204, 205)를 프로그래밍하고 FSM의 상태들 사이의 전이에 대응하도록 SME(204, 205)를 서로 선택적으로 연결하는 것에 의해 FSM 엔진(200) 상에 구현될 수 있다.

[0017] 도 2는 예시적인 FSM 엔진(200)의 전체 뷰를 도시한다. FSM 엔진(200)은 프로그래밍가능한 블록간 스위치(203)에 선택적으로 서로 연결될 수 있는 복수의 블록(202)을 포함한다. 추가적으로, 블록(202)은 신호(예를 들어, 데이터)를 수신하고 데이터를 블록(202)에 제공하기 위해 입력 블록(209)(예를 들어, 데이터 입력 포트)에 선택적으로 연결될 수 있다. 블록(202)은 블록(202)으로부터 신호를 외부 디바이스(예를 들어, 다른 FSM 엔진(200))로 제공하기 위해 출력 블록(213)(예를 들어, 출력 포트)에 선택적으로 더 연결될 수 있다. FSM 엔진(200)은 FSM 엔진(200)에 프로그램(예를 들어, 이미지)을 로딩하기 위해 프로그래밍 인터페이스(211)를 더 포함할 수 있다. 이미지는 SME(204, 205)의 상태를 프로그램(예를 들어, 설정)할 수 있다. 즉, 이미지는 입력 블록(209)에서 주어진 입력에 특정 방식으로 반응하도록 SME(204, 205)를 구성할 수 있다. 예를 들어, SME(204)는 입력 블록(209)에 문자 'a'가 수신될 때 하이 신호를 출력하도록 설정될 수 있다.

[0018] 일례에서, 입력 블록(209), 출력 블록(213) 및/또는 프로그래밍 인터페이스(211)는 레지스터로 구현되되 레지스터에의 기록이 각 요소로부터 또는 각 요소로 데이터를 제공하도록 구현될 수 있다. 따라서, 프로그래밍 인터페이스(211)에 대응하는 레지스터에 저장된 이미지로부터 비트는 SME(204, 205)에 로딩될 수 있다. 도 2는 블록(202), 입력 블록(209), 출력 블록(213) 및 블록간 스위치(203) 사이에 특정 개수의 전도체(예를 들어, 와이어, 트레이스)를 도시하지만, 다른 예에서, 더 적은 수 또는 더 많은 수의 전도체가 사용될 수 있는 것으로 이해된다.

[0019] 도 3은 블록(202)의 일례를 도시한다. 블록(202)은 프로그래밍가능한 블록 내부 스위치(208)와 서로 선택적으로 연결될 수 있는 복수의 행(206)을 포함할 수 있다. 추가적으로, 행(206)은 블록간 스위치(203)로 다른 블록(202) 내 다른 행(206)에 선택적으로 연결될 수 있다. 일례에서, 버퍼(201)는 블록간 스위치(203)로/로부터 신호 타이밍을 제어하기 위해 포함된다. 행(206)은 본 명세서에서 GOT(groups of two)(210)라고 지칭되는 요소의 쌍으로 구성된 복수의 SME(204, 205)를 포함한다. 일례에서, 블록(202)은 16개의 행(206)을 포함한다.

[0020] 도 4는 행(206)의 일례를 도시한다. GOT(210)는 프로그래밍가능한 행 내부 스위치(212)에 의해 행(206) 내 다른

GOT(210) 및 임의의 다른 요소(224)에 선택적으로 연결될 수 있다. GOT(210)는 블록 내부 스위치(208)로 다른 행(206) 내 다른 GOT(210)에 더 연결되거나 또는 블록간 스위치(203)로 다른 블록(202) 내 다른 GOT(210)에 더 연결될 수 있다. 일례에서, GOT(210)는 제1 및 제2 입력(214, 216)과 출력(218)을 구비한다. 제1 입력(214)은 GOT(210)의 제1 SME(204)에 연결되고, 제2 입력(214)은 GOT(210)의 제2 SME(204)에 연결된다.

[0021] 일례에서, 행(206)은 제1 및 제2 복수의 행 상호연결 전도체(220, 222)를 포함한다. 일례에서, GOT(210)의 입력(214, 216)은 하나 이상의 행 상호연결 전도체(220, 222)에 연결될 수 있고, 출력(218)은 하나의 행 상호연결 전도체(220, 222)에 연결될 수 있다. 일례에서, 제1 복수의 행 상호연결 전도체(220)는 행(206) 내 각 GOT(210)의 각 SME(204)에 연결될 수 있다. 제2 복수의 행 상호연결 전도체(222)는 행(206) 내 각 GOT(210)의 하나의 SME(204)에 연결될 수 있으나, GOT(210)의 다른 SME(204)에는 연결될 수 없다. 일례에서, 제2 복수의 행 상호연결 전도체(222)의 제1 절반(half)은 행(206) 내 SME(204)(각 GOT(210)로부터 하나의 SME(204))의 제1 절반에 연결될 수 있고, 제2 복수의 행 상호연결 전도체(222)의 제2 절반은 행(206) 내 SME(204)(각 GOT(210)로부터 다른 SME(204))의 제2 절반에 연결될 수 있다. 제2 복수의 행 상호연결 전도체(222)와 SME(204, 205) 사이에 제한된 연결은 본 명세서에서 "패리티(parity)"라고 지칭된다.

[0022] 일례에서, 행(206)은 카운터, 프로그래밍가능한 부울리안 논리 요소, 전계 프로그래밍가능한 게이트 어레이(FPGA), 응용 특정 집적 회로(application specific integrated circuit: ASIC), 프로그래밍가능한 프로세서(예를 들어, 마이크로프로세서) 및 다른 요소와 같은 특수 목적 요소(224)를 더 포함할 수 있다. 추가적으로, 일례로서, 특수 목적 요소(224)는 상이한 행(206)에서는 상이하다. 예를 들어, 블록(202) 내 행 중 4개의 행(206)은 특수 목적 요소(224)로 부울리안 논리를 포함할 수 있고, 블록(202) 내 다른 8개의 행(206)은 특수 목적 요소(224)로 카운터를 포함할 수 있다.

[0023] 일례에서, 특수 목적 요소(224)는 카운터(본 명세서에서 카운터(224)라고도 지칭됨)를 포함한다. 일례에서, 카운터(224)는 12비트 프로그래밍가능한 다운 카운터(down counter)를 포함한다. 12비트 프로그래밍가능한 카운터(224)는 카운트 입력, 리셋 입력 및 제로 카운트 출력을 구비한다. 카운트 입력은 선언될 때 일(one)만큼 카운터(224)의 값을 감분(decrement)한다. 리셋 입력은 선언될 때 카운터(224)로 하여금 연관된 레지스터로부터 초기 값을 로딩하게 한다. 12비트 카운터(224)에서, 최대 12비트 수는 초기 값으로 로딩될 수 있다. 카운터(224)의 값이 제로(0)로 감분되면, 제로 카운트 출력이 선언된다. 카운터(224)는 또한 적어도 2개의 모드, 즉 펄스(pulse) 모드와 홀드(hold) 모드를 구비한다. 카운터(224)가 펄스 모드로 설정되면, 제로 카운트 출력은 카운터(224)가 제로로 감분될 때 제1 클록 사이클 동안 선언되고, 이후 클록 사이클에서 제로 카운트 출력은 카운트 입력이 선언되는 경우에도 더 이상 선언되지 않는다. 이 상태는 카운터(224)가 선언되는 리셋 입력에 의하여 리셋될 때까지 지속한다. 카운터(224)가 홀드 모드로 설정될 때 제로 카운트 출력은 카운터(224)가 제로로 감분될 때 제1 클록 사이클 동안 선언되고, 카운터(224)가 선언되는 리셋 입력에 의해 리셋될 때까지 카운트 입력이 선언될 때 선언이 유지된다.

[0024] 도 5는 GOT(210)의 일례를 도시한다. GOT(210)는, 입력(214, 216)을 가지고, OR 게이트(230) 및 3대1 다중화기(multiplexer)(242)에 연결된 출력(226, 228)을 가지는 제1 SME(204) 및 제2 SME(205)를 포함한다. 3대1 다중화기(242)는 GOT(210)의 출력(218)을 제1 SME(204), 제2 SME(205), 또는 OR 게이트(230)에 연결하도록 설정될 수 있다. OR 게이트(230)는 두 출력(226, 228)을 서로 연결하여 GOT(210)의 공통 출력(218)을 형성하는데 사용될 수 있다. 일례에서, 제1 및 제2 SME(204, 205)는 전술된 바와 같이 패리티를 나타내며, 여기서 제1 SME(204)의 입력(214)은 행 상호연결 전도체(222)의 일부에 연결될 수 있고, 제2 SME(205)의 입력(216)은 다른 행 상호연결 전도체(222)에 연결될 수 있다. 일례에서, GOT(210) 내 2개의 SME(204, 205)는 어느 하나 또는 둘 모두의 스위치(240)를 설정하는 것에 의해 자체적으로 캐스케이드연결(cascaded)되고 및/또는 루프백(looped back)될 수 있다. SME(204, 205)는 SME(204, 205)의 출력(226, 228)을 다른 SME(204, 205)의 입력(214, 216)에 연결하는 것에 의해 캐스케이드연결될 수 있다. SME(204, 205)는 출력(226, 228)을 자기 자신의 입력(214, 216)에 연결하는 것에 의해 자체적으로 루프백될 수 있다. 따라서, 제1 SME(204)의 출력(226)은 제1 SME(204)의 입력(214) 및 제2 SME(205)의 입력(216) 중 어느 것에도 연결되지 않거나 이들 중 어느 하나 또는 둘 모두에 연결될 수 있다.

[0025] 일례에서, 상태 기계 요소(204, 205)는 검출 라인(234)과 병렬 연결된 동적 랜덤 액세스 메모리(DRAM)에 종종 사용된 것과 같은 복수의 메모리 셀(232)을 포함한다. 하나의 이러한 메모리 셀(232)은 하이 또는 로우 값(예를 들어, 1 또는 0)에 대응하는 것과 같은 데이터 상태로 설정될 수 있는 메모리 셀을 포함한다. 메모리 셀(232)의 출력은 검출 라인(234)에 연결되고, 메모리 셀(232)의 입력은 데이터 스트림 라인(236) 상의 데이터에 기초하여 신호를 수신한다. 일례에서, 데이터 스트림 라인(236)의 입력은 메모리 셀(232) 중 하나를 선택하도록 디코딩된

다. 선택된 메모리 셀(232)은 저장된 데이터 상태를 검출 라인(234) 상에 출력으로 제공한다. 예를 들어, 데이터 입력 포트(209)에서 수신된 데이터는 디코더(미도시)에 제공될 수 있고, 디코더는 데이터 스트림 라인(236) 중 하나를 선택할 수 있다. 일례에서, 디코더는 ACSII 문자를 256개의 비트 중 하나로 변환할 수 있다.

[0026] 그리하여, 메모리 셀(232)이 하이 값으로 설정되고 데이터 스트림 라인(236) 상의 데이터가 메모리 셀(232)에 대응할 때 메모리 셀(232)은 검출 라인(234)에 하이 신호를 출력한다. 데이터 스트림 라인(236) 상의 데이터가 메모리 셀(232)에 대응하고 메모리 셀(232)이 로우 값으로 설정될 때 메모리 셀(232)은 검출 라인(234)에 로우 신호를 출력한다. 메모리 셀(232)로부터 검출 라인(234) 상으로의 출력은 검출 회로(238)에 의해 센싱된다. 일례에서, 입력 라인(214, 216) 상의 신호는 각 검출 회로(238)를 활성(active) 상태 또는 비활성(inactive) 상태로 설정한다. 비활성 상태로 설정될 때 검출 회로(238)는 각 검출 라인(234) 상의 신호에 상관없이 각 출력(226, 228)에 로우 신호를 출력한다. 활성 상태로 설정될 때, 검출 회로(238)는 각 SME(204, 205)의 메모리 셀(234) 중 하나로부터 하이 신호가 검출될 때 각 출력 라인(226, 228)에 하이 신호를 출력한다. 활성 상태에 있을 때, 검출 회로(238)는 각 SME(204, 205)의 메모리 셀(234) 전부로부터 신호가 로우일 때 각 출력 라인(226, 228) 상에 로우 신호를 출력한다.

[0027] 일례에서, SME(204, 205)는 256개의 메모리 셀(232)을 포함하며, 각 메모리 셀(232)은 상이한 데이터 스트림 라인(236)에 연결된다. 따라서, SME(204, 205)는 데이터 스트림 라인(236) 중 선택된 하나 이상의 것이 하이 신호를 가질 때 하이 신호를 출력하도록 프로그래밍될 수 있다. 예를 들어, SME(204)는 제1 메모리 셀(232)(예를 들어, 비트 0)을 하이로 설정하게 하고 모든 다른 메모리 셀(232)(예를 들어, 비트 1 내지 255)을 로우로 설정하게 할 수 있다. 각 검출 회로(238)가 활성 상태에 있을 때, SME(204)는 비트 0에 대응하는 데이터 스트림 라인(236)이 하이 신호를 가질 때 출력(226)에 하이 신호를 출력한다. 다른 예에서, SME(204)는 적절한 메모리 셀(232)을 하이 값으로 설정하는 것에 의해 다수의 데이터 스트림 라인(236) 중 하나가 하이 신호를 가질 때 하이 신호를 출력하도록 설정될 수 있다.

[0028] 일례에서, 메모리 셀(232)은 연관된 레지스터로부터 비트를 판독하는 것에 의해 하이 또는 로우 값으로 설정될 수 있다. 따라서, SME(204)는 컴파일러에 의하여 생성된 이미지를 레지스터에 저장하고 레지스터 내 비트를 연관된 메모리 셀(232)에 로딩하는 것에 의해 프로그래밍될 수 있다. 일례에서, 컴파일러에 의해 생성된 이미지는 하이 및 로우(예를 들어, 1 및 0) 비트의 바이너리 이미지(binary image)를 포함한다. 이미지는 SME(204, 205)를 캐스케이드연결하는 것에 의해 FSM으로 동작하도록 FSM 엔진(200)을 프로그래밍할 수 있다. 예를 들어, 제1 SME(204)는 검출 회로(238)를 활성 상태로 설정하는 것에 의해 활성 상태로 설정될 수 있다. 제1 SME(204)는 비트 0에 대응하는 데이터 스트림 라인(236)이 하이 신호를 가질 때 하이 신호를 출력하도록 설정될 수 있다. 제2 SME(205)는 초기에 비활성 상태로 설정될 수 있으나, 활성일 때 비트 1에 대응하는 데이터 스트림 라인(236)이 하이 신호를 가질 때 하이 신호를 출력하도록 설정될 수 있다. 제1 SME(204) 및 제2 SME(205)는 제1 SME(204)의 출력(226)을 제2 SME(205)의 입력(216)에 연결하도록 설정하는 것에 의해 캐스케이드연결될 수 있다. 따라서, 하이 신호가 비트 0에 대응하는 데이터 스트림 라인(236)에서 센싱될 때, 제1 SME(204)는 출력(226)에 하이 신호를 출력하고 제2 SME(205)의 검출 라인(238)을 활성 상태로 설정한다. 하이 신호가 비트 1에 대응하는 데이터 스트림 라인(236)에서 센싱되면, 제2 SME(205)는 다른 SME(205)를 활성화시키거나 FSM 엔진(200)으로부터 출력하기 위해 출력(228)에 하이 신호를 출력한다.

[0029] 도 6은 병렬 기계를 프로그래밍하도록 구성된 이미지로 소스 코드를 변환하는 컴파일러를 위한 방법(600)의 일례를 도시한다. 방법(600)은 소스 코드를 구문 트리(syntax tree)로 파싱(parse)하는 단계(블록 602), 상기 구문 트리를 오토마톤으로 변환하는 단계(블록 604), 상기 오토마톤을 최적화하는 단계(블록 606), 상기 오토마톤을 네트리스트로 변환하는 단계(블록 608), 상기 네트리스트를 하드웨어에 배치하는 단계(블록 610), 상기 네트리스트를 라우팅하는 단계(블록 612) 및 결과 이미지를 발행(publish)하는 단계(블록 614)를 포함한다.

[0030] 일례에서, 컴파일러는 소프트웨어 개발자로 하여금 FSM 엔진(600)에 FSM을 구현하기 위한 이미지를 생성하게 하는 애플리케이션 프로그래밍 인터페이스(application programming interface: API)를 포함한다. 컴파일러는 FSM 엔진(600)을 프로그래밍하도록 구성된 이미지로 소스 코드의 정규 표현(regular expression)의 입력 세트를 변환하는 방법을 제공한다. 컴파일러는 폰 노이만 아키텍처(Von Nuemann architecture)를 가지는 컴퓨터를 위한 명령에 의해 구현될 수 있다. 이들 명령은 컴퓨터 상의 프로세서로 하여금 컴파일러의 기능을 구현하게 할 수 있다. 예를 들어, 명령은 프로세서에 의해 실행될 때 프로세서로 하여금 프로세서에 액세스가능한 소스 코드에 대해 블록(602, 604, 606, 608, 610, 612 및 614)에 기술된 동작을 수행하게 할 수 있다. 폰 노이만 아키텍처를 가지는 예시적인 컴퓨터는 도 16에 도시되고 후술된다.

[0031] 일례에서, 소스 코드는 심볼(symbol) 그룹 내 심볼 패턴을 식별하기 위한 검색 스트링(search string)을 기술한다. 이 검색 스트링을 기술하기 위하여 소스 코드는 복수의 정규 표현(regex)을 포함할 수 있다. regex는 심볼 검색 패턴을 기술하는 스트링일 수 있다. regex는 프로그래밍 언어, 텍스트 편집기, 네트워크 보안 등과 같은 여러 컴퓨터 영역에서 널리 사용된다. 일례에서, 컴파일러에 의해 지원되는 정규 표현은 구조화되지 않은 데이터(unstructured data)의 검색을 위한 검색 기준을 포함한다. 구조화되지 않은 데이터는 자유 형태이고 데이터 내 워드에 적용된 인덱싱(indexing)이 없는 데이터를 포함할 수 있다. 워드는 데이터 내에, 프린트가능한 및 프린트가능하지 않은, 임의의 바이트 조합을 포함할 수 있다. 일례에서, 컴파일러는 Perl(예를 들어, Perl 호환가능한 정규 표현(PCRE)), PHP, 자바 및 .NET 언어를 포함하는 regex를 구현하기 위한 다수의 상이한 소스 코드 언어를 지원할 수 있다.

[0032] 다시 도 6을 참조하면, 블록(602)에서, 컴파일러는 소스 코드를 파싱하여 상관 연결된 오퍼레이터(relationally connected operator)들의 배열을 형성할 수 있고, 여기서 상이한 유형의 오퍼레이터는 소스 코드에 의해 구현된 상이한 기능(예를 들어, 소스 코드에서 regex에 의해 구현된 상이한 기능)에 대응한다. 소스 코드를 파싱하면 소스 코드의 일반 표현(generic representation)을 생성할 수 있다. 일례에서, 일반 표현은 구문 트리로서 알려진 트리 그래프의 형태로 소스 코드에서 regex의 인코딩된 표현을 포함한다. 본 명세서에서 설명된 예는 구문 트리(또한 "추상 구문 트리(abstract syntax tree)"라고도 알려진 것)로서의 배열을 말하지만, 다른 예에서는, 구체적 구문 트리(concrete syntax tree) 또는 다른 배열이 사용될 수 있다.

[0033] 전술된 바와 같이 컴파일러는 다수의 소스 코드 언어를 지원할 수 있으므로, 파싱은 언어에 상관없이 소스 코드를 비 언어 특정 표현, 예를 들어, 구문 트리로 변환한다. 따라서, 컴파일러에 의한 추가적인 처리(블록 604, 606, 608, 610)는 소스 코드의 언어에 상관없이 공통 입력 구조로부터 작업할 수 있다.

[0034] 전술된 바와 같이, 구문 트리는 상관 연결된 복수의 오퍼레이터를 포함한다. 구문 트리는 다수의 상이한 유형의 오퍼레이터를 포함할 수 있다. 즉, 상이한 오퍼레이터는 소스 코드에서 regex에 의해 구현된 상이한 기능에 대응할 수 있다.

[0035] 블록(604)에서, 구문 트리는 오토마톤으로 변환된다. 일례에서, 오토마톤은 FSM의 소프트웨어 모델을 포함하고, 이에 따라 결정론적(deterministic) 또는 비결정론적인 것으로 분류될 수 있다. 결정론적 오토마톤은 주어진 시간에 단일 실행 경로를 가지는 반면, 비 결정론적 오토마톤은 다수의 동시 실행 경로를 구비한다. 오토마톤은 복수의 상태를 포함한다. 오토마톤은 노드에 의해 표현될 수 있는 복수의 상태를 포함한다. 구문 트리를 오토마톤으로 변환하기 위하여, 구문 트리에서 오퍼레이터 및 오퍼레이터들 사이의 관계는 오토마톤의 상태들 사이의 (방향성 에지(directed edge)에 의해 표현된) 전이를 가지는 (노드에 의해 표현된) 상태로 변환된다. 일례에서, 오토마톤은 FSM 엔진(200)의 하드웨어에 부분적으로 기초하여 변환될 수 있다.

[0036] 일례에서, 오토마톤의 입력 심볼은 알파벳 심볼, 부호 0 내지 9 및 다른 프린트가능한 문자를 포함한다. 일례에서, 입력 심볼은 바이트 값 0 내지 255를 포함하여 이 바이트 값에 의해 표현된다. 일례에서, 오토마톤은 그래프의 노드(node)들이 상태의 세트에 대응하는, 방향성 그래프(directed graph)로 표현될 수 있다. 일례에서, 오토마톤에 의해 허용된(예를 들어, 매칭된) 데이터는 오토마톤으로 순차적으로 입력될 때 최종 상태에 도달하는 모든 가능한 문자 데이터의 세트이다. 오토마톤에 의해 허용된 데이터에서 각 심볼은 시작 상태로부터 하나 이상의 최종 상태로 경로를 추적한다.

[0037] 일례에서, 오토마톤은 일반 목적 상태 및 특수 목적 상태를 포함한다. 일반 목적 상태 및 특수 목적 상태는 컴파일러가 기계 코드를 생성하고 있는 타깃 디바이스에 의하여 지원되는 일반 목적 요소 및 특수 목적 요소에 대응한다. 상이한 유형의 타깃 디바이스는 상이한 유형의 일반 목적 요소 및 하나 이상의 상이한 유형의 특수 목적 요소를 지원할 수 있다. 일반 목적 요소는 일반적으로 넓은 범위의 기능을 구현하는데 사용될 수 있는 반면, 특수 목적 요소는 일반적으로 더 좁은 범위의 기능을 구현하는데 사용될 수 있다. 그러나, 일례에서, 특수 목적 요소는 예를 들어 좁은 범위의 기능에서 더 우수한 효율을 달성할 수 있다. 따라서, 특수 목적 요소는 예를 들어 타깃 디바이스에서 특정 기능을 구현하는데 필요한 기계 자원 또는 기계 사이클을 감소시키는데 사용될 수 있다. 일부 예에서, 타깃 디바이스는 단지 특수 목적 요소를 지원하며, 여기서 다수의 상이한 유형의 특수 목적 요소들이 지원된다.

[0038] 컴파일러가 FSM 엔진(200)의 기계 코드를 생성하고 있는 예에서, 일반 목적 상태는 SME(204, 205)에 대응할 수 있고, 이에 따라 일반 목적 상태는 본 명세서에서 "SME 상태"라고 지칭된다. 나아가, 컴파일러가 FSM 엔진(600)의 기계 코드를 생성하고 있을 때 특수 목적 상태는 카운터(224)에 대응할 수 있고, 이에 본 명세서에서 "카운터 상태"라고 지칭된다. 일례에서, 오토마톤에서 SME 상태는, SME로 맵핑되지 않는 오토마톤의 시작 상태를

제외하고는, FSM 엔진(200)에서 SME(예를 들어, SME(204, 205))로 1대1 맵핑을 한다. 카운터(224)는 카운터 상태로 1대1 맵핑을 할 수도 있고 또는 하지 않을 수도 있다.

[0039] 일례에서, 입력 심볼 범위 외 특수 전이 심볼이 오토마톤에서 사용될 수 있다. 이들 특수 전이 심볼은 예를 들어, 특수 목적 요소(224)의 사용을 가능하게 하는데 사용될 수 있다. 나아가, 특수 전이 심볼은 입력 심볼과는 다른 것에서 발생하는 전이를 제공하는데 사용될 수 있다. 예를 들어, 특수 전이 심볼은 제2 상태와 제3 상태 모두가 인에이블될 때 제1 상태가 인에이블되는 (예를 들어, 전이되는) 것을 나타낼 수 있다. 따라서, 제1 상태는 제2 상태와 제3 상태 모두가 활성화될 때 활성화되며, 제1 상태로의 전이는 입력 심볼에 직접 종속하지 않는다. 특히, 제2 상태와 제3 상태 모두가 인에이블될 때 제1 상태가 인에이블되는 것을 나타내는 특수 전이 심볼은 예를 들어 특수 목적 요소(224)로서 부울리안 논리에 의하여 수행되는 부울리안 AND 함수를 나타내는데 사용될 수 있다. 일례에서, 특수 전이 심볼은 카운터 상태가 제로에 도달하여 다운스트림 상태로 전이하는 것을 나타내는데 사용될 수 있다.

[0040] 일례에서, 구문 트리로부터 생성된 오토마톤은 균일한 오토마톤(homogeneous automaton)이다. 균일한 오토마톤은 일반 오토마톤 정의에 제한을 둔다. 이 제한은 상태에 들어가는 모든 전이들이 동일한 입력 심볼(들)에서 발생하여야 하는 것을 요구한다. 균일한 오토마톤은 다음 조건을 만족한다: 임의의 2개의 상태, q_1 및 q_2에 대해, $r \in \delta(q_1) \cap \delta(q_2)$라면, $S_1 = \{a \mid a \in \sum, r \in \delta(q_1, a)\}$, $S_2 = \{a \mid a \in \sum, r \in \delta(q_2, a)\}$라고 표시한다. S_1는 q_1이 r로 전이하게 하는 심볼 세트이고, S_2는 q_2가 r로 전이하게 하는 심볼 세트이다. 여기서 $S_1=S_2$이고, 여기서 상태(q_1) 및 상태(q_2) 모두가 상태(r)로 전이한다면, 균일한 제한은 전이가 동일한 심볼(들)에 발생하여야 한다는 것이다.

[0041] 구문 트리의 오토마톤으로의 변환에 대한 추가적인 상세는 도 7을 참조하여 후술된다.

[0042] 블록(606)에서, 오토마톤이 구성된 후에, 오토마톤은 특히 그 복잡성과 사이즈를 감소시키도록 최적화된다. 오토마톤은 중복 상태(redundant state)들을 결합하는 것에 의해 최적화될 수 있다.

[0043] 블록(608)에서, 오토마톤은 네트리스트로 변환된다. 오토마톤을 네트리스트로 변환하는 것은 오토마톤의 상태를 FSM 엔진(200)의 하드웨어 요소(예를 들어, SME(204, 205), GOT(210), 특수 목적 요소(224))의 인스턴스(instance)로 맵핑하며, 인스턴스들 사이의 연결을 결정한다. 일례에서, 네트리스트는 복수의 인스턴스를 포함하며, 각 인스턴스는 FSM 엔진(200)의 하드웨어 요소에 대응한다(예를 들어, 이 하드웨어 요소를 나타낸다). 각 인스턴스는 다른 인스턴스에 연결하기 위한 하나 이상의 연결점(이는 "포트(port)"라고도 지칭됨)을 구비할 수 있다. 네트리스트는 인스턴스에 대응하는 하드웨어 요소를 연결하는 전도체에 대응하는(예를 들어, 전도체를 나타내는) 인스턴스의 포트들 사이에 복수의 연결을 더 포함한다. 일례에서, 네트리스트는 상이한 유형의 하드웨어 요소에 대응하는 상이한 유형의 인스턴스를 포함한다. 예를 들어, 네트리스트는 일반 목적 하드웨어 요소에 대응하는 일반 목적 인스턴스와, 특수 목적 하드웨어 요소에 대응하는 특수 목적 인스턴스를 포함할 수 있다. 일례로서, 일반 목적 상태는 일반 목적 인스턴스로 변환될 수 있고, 특수 목적 상태는 특수 목적 인스턴스로 변환될 수 있다. 일례에서, 일반 목적 인스턴스는 SME(204, 205)의 SME 인스턴스와, SME 그룹을 포함하는 하드웨어 요소에 대한 SME 그룹 인스턴스를 포함할 수 있다. 일례에서, SME 그룹 인스턴스는 GOT(210)에 대응하는 GOT 인스턴스를 포함하지만, 다른 예에서, SME 그룹 인스턴스는 3개 이상의 SME의 그룹을 포함하는 하드웨어 요소에 대응할 수 있다. 특수 목적 인스턴스는 카운터(224)를 위한 카운터 인스턴스 및 논리 요소(224)를 위한 논리 인스턴스를 포함할 수 있다. GOT(210)는 2개의 SME(204, 205)를 포함하므로, GOT 인스턴스는 2개의 SME 인스턴스를 포함한다.

[0044] 네트리스트를 생성하기 위해, 오토마톤의 상태는, 시작 상태가 대응하는 인스턴스를 가지지 않는 것을 제외하고는, 네트리스트에서 인스턴스로 변환된다. SME 상태는 GOT 인스턴스로 변환되고, 카운터 상태는 카운터 인스턴스로 변환된다. 추가적으로, 제1 인스턴스로부터 제2 인스턴스로 대응하는 연결은 제1 인스턴스에 대응하는 상태로부터 제2 인스턴스에 대응하는 상태로의 전이에 대해 생성된다. FSM 엔진(200)에서 SME(204, 205)가 GOT(210)라고 지칭된 쌍으로 그룹화되므로, 컴파일러는 SME 상태를 GOT 인스턴스의 쌍으로 그룹화될 수 있다. GOT(210)의 물리적 설계로 인해, 모든 SME 인스턴스가 GOT(210)를 형성하기 위해 서로 쌍으로 형성되는 것은 아니다. 따라서, 컴파일러는 SME 상태 중 어느 것이 GOT(210)로 서로 맵핑될 수 있는 것인지를 결정하고 이 결정에 기초하여 SME 상태를 GOT 인스턴스로 쌍으로 형성한다. 오토마톤을 네트리스트로 변환하는 예시적인 방법에 대한 추가적인 상세는 아래 도 15a 및 도 15b를 참조하여 후술된다.

[0045] 블록(610)에서, 일단 네트리스트가 생성되면, 네트리스트는 네트리스트의 각 하드웨어 요소 인스턴스에 대해 타

깃 디바이스(예를 들어, SME(204, 205), 다른 요소(224))의 특정 하드웨어 요소를 선택하도록 배치된다. 본 발명의 일 실시형태에 따르면, 배치하는 것은 하드웨어 요소에 대한 일반 입력 및 출력의 제약에 기초하여 각 특정 하드웨어 요소를 선택한다.

[0046] 배치하는 것은 어려운 문제일 수 있고 학습법(heuristics)을 사용하여 일반적으로 해결된다. 이것은 힘 방향(force directed) 기술, 파티셔닝(partitioning) 기술, 시뮬레이션된 어닐링(simulated annealing), 또는 전술된 기술의 조합과 같은 방법을 사용하여 수행될 수 있다.

[0047] 일례에서, 2개의 방법이 큰 결합 최적화(combinatorial optimization) 문제를 해결하는데 사용될 수 있고; 이들은 시뮬레이션된 어닐링 및 다중 레벨 하이퍼 그래프 파티셔닝(multi-level hyper-graph partitioning)이다. 이들 방법 사이에 트레이드 오프(trade-off)는 정밀도 대 속도이다. 시뮬레이션된 어닐링은 초고품질 배치를 생성할 수 있으나 중앙 처리 유닛(CPU) 시간에 비해 극히 값비싸다. 대조적으로, 하이퍼 그래프 파티셔닝은 한 자리수 더 빠를 수 있으나 덜 최적인 배치를 생성하는 경향이 있다. 일례에서, 시뮬레이션된 어닐링은 타깃 하드웨어 디바이스의 필요를 충족하는 고품질의 배치를 보장하는데 사용될 수 있다. 다른 예에서, 하이퍼 그래프 파티셔닝이 제1 단계로 사용되고 나서 하이퍼 그래프 파티셔닝 단계에 의해 생성된 배치를 정교하게 하기 위해 시뮬레이션 어닐링 동작을 사용할 수 있다. 일부 예에서 시뮬레이션된 어닐링 및 다중 레벨 하이퍼 그래프 파티셔닝의 조합을 사용하여 각 학습법의 강도를 집중시킬 수 있다.

[0048] 블록(612)에서, 배치된 네트리스트는 네트리스트에 의해 기술된 연결을 달성하기 위해 선택된 하드웨어 요소를 서로 연결하기 위하여 프로그래밍 가능한 스위치(예를 들어, 블록간 스위치(203), 블록 내부 스위치(208) 및 행 내부 스위치(212))에 대한 설정을 결정하도록 라우팅된다. 일례에서, 프로그래밍가능한 스위치에 대한 설정은 프로그래밍가능한 스위치에 대한 설정과, 선택된 하드웨어 요소를 연결하는데 사용될 수 있는 FSM 엔진(200)의 특정 전도체를 결정하는 것에 의해 결정된다. 라우팅은 블록(610)에서 배치하는 것보다 하드웨어 요소들 사이에 보다 특정한 연결 제한을 고려할 수 있다. 따라서, 라우팅은 FSM 엔진(200)에 전도체의 실제 제한이 주어지면 적절한 연결을 만들기 위하여 전체적인 배치에 의하여 결정된 하드웨어 요소들 중 일부의 위치를 조절할 수 있다.

[0049] 네트리스트가 배치되고 라우팅되면, 배치되고 라우팅된 네트리스트는 FSM 엔진(200)의 프로그래밍을 위한 복수의 비트로 변환될 수 있다. 복수의 비트는 본 명세서에서 이미지라 지칭된다.

[0050] 일부 예에서, 블록(608)에서 오토마톤을 네트리스트로 변환하기 전에, 오토마톤은 다수의 더 작은 오토마톤으로 분할되고, 각 더 작은 오토마톤은 블록(608)에서 네트리스트로 개별적으로 변환된다. 블록(610)에서 배치의 복잡성은 인스턴스의 개수가 증가함에 따라 증가하므로, 오토마톤을 복수의 더 작은 오토마톤으로 분할하고, 더 작은 오토마톤을 개별적인 네트리스트로 변환하는 것은 블록(610)과 블록(612)에서 배치하고 라우팅하기 위해 더 작은 네트리스트를 제공할 수 있다. 따라서, 더 작은 네트리스트를 배치하는 것은 허용가능한 구성을 결정하는데 필요한 시간을 감소시키게 할 수 있다. 일례에서, 오토마톤은 그래프 이론을 사용하여 다수의 더 작은 오토마톤으로 분할된다. 각 더 작은 오토마톤은 네트리스트로 개별적으로 변환될 수 있고(블록 608), 할당된 영역 내에 배치될 수 있다(블록 610). 따라서, 이용가능한 영역은 상이한 네트리스트로 분할되고 할당될 수 있으며 각 네트리스트는 할당된 부분 내에 개별적으로 배치된다. 전술된 바와 같이, 이용가능한 영역의 일부 부분은 미할당된 채 유지될 수 있어서 다른 네트리스트를 배치하는데 이용가능하다. 일례에서, 더 작은 오토마톤으로부터 형성된 각 네트리스트는 전체 처리 시간을 감소시키기 위해 병렬로 결정된 구성을 가질 수 있다.

[0051] 블록(614)에서 이미지는 컴파일러에 의해 발행된다. 이미지는 FSM 엔진(200)의 특정 하드웨어 요소 및/또는 프로그래밍 가능한 스위치를 프로그래밍하기 위한 복수의 비트를 포함한다. 이미지가 복수의 비트(예를 들어, 0 및 1)를 포함하는 실시형태에서, 이미지는 바이너리 이미지라 지칭될 수 있다. 이 비트는 FSM 엔진(200)에 로딩되어 SME(204, 205)의 상태, 특정 목적 요소(224) 및 프로그래밍 가능한 스위치를 프로그래밍하여 프로그래밍된 FSM 엔진(200)이 소스 코드에 의해 기술된 기능을 가지는 FSM을 구현할 수 있게 한다. 배치(블록 610) 및 라우팅(블록 612)은 FSM 엔진(200) 내 특정 위치에 있는 특정 하드웨어 요소를 오토마톤 내 특정 상태로 맵핑할 수 있다. 따라서, 이미지 내 비트는 원하는 기능(들)을 구현하기 위해 특정 하드웨어 요소 및/또는 프로그래밍가능한 스위치를 프로그래밍할 수 있다. 일례에서, 이미지는 기계 코드를 컴퓨터 판독가능한 매체에 저장하는 것에 의해 발행될 수 있다. 다른 예에서, 이미지는 디스플레이 디바이스 상에 이미지를 디스플레이하는 것에 의해 발행될 수 있다. 또 다른 예에서, 이미지는 FSM 엔진(200)에 이미지를 로딩하기 위해 프로그래밍 디바이스와 같은 다른 디바이스에 이미지를 송신하는 것에 의해 발행될 수 있다. 또 다른 예에서, 이미지는 병렬 기계(예를 들어, FSM 엔진(200))에 이미지를 로딩하는 것에 의해 발행될 수 있다.

[0052] 일례에서, 이미지는 이미지로부터 SME(204, 205) 및 다른 하드웨어 요소(224)로 비트 값을 직접 로딩하는 것에 의해 또는 이미지를 하나 이상의 레지스터에 로딩한 다음 이 레지스터로부터 비트 값을 SME(204, 205) 및 다른 하드웨어 요소(224)에 기록하는 것에 의해 FSM 엔진(200)에 로딩될 수 있다. 일례에서, FSM 엔진(200)의 하드웨어 요소(예를 들어, SME(204, 205), 다른 요소(224), 프로그래밍가능한 스위치(203, 208, 212))는 컴퓨터(예를 들어, 컴퓨터에 연결되거나 이와 일체화된 프로그래밍 디바이스)가 이미지를 하나 이상의 메모리 어드레스에 기록하는 것에 의해 이미지를 FSM 엔진(200)에 로딩할 수 있도록 메모리 맵핑된다.

[0053] 도 7은 구문 트리를 오토마톤으로 변환하는 컴파일러를 위한 예시적인 방법(604)에 포함될 수 있는 여러 추가적인 동작을 도시한다. 방법(604)은 심볼(710)을 삽입하는 동작, 구문 트리를 타깃 디바이스(720) 내에서 동작하도록 처리하는 동작, 구문 트리 요소(730)를 분류하는 동작, 비 결정론적 오퍼레이터(740)를 대체하는 동작을 포함할 수 있다. 도 7에 도시된 방법(604)은 하나 이상의 구문 트리를 하나 이상의 오토마톤으로 변환하는 동안 수행될 수 있는 여러 동작을 도시한다. 도 7에 도시된 동작 순서는 단지 예시적인 것이며, 이 동작은 여러 순서로 수행될 수 있다. 추가적으로, 특정 예에서, 상이한 조합의 동작이 사용될 수 있다.

[0054] 일례에서, 방법(700)은 구문 트리를 오토마톤으로 변환하는 동안 특수 전이 심볼을 오토마톤으로 삽입한다. 하나의 이러한 예에서 특수 전이 심볼은 오토마톤 상태로 1:1 맵핑되지 않는 오퍼레이터에 대응한다. 전술된 바와 같이 특수 전이 심볼은 특히 부울리안 연산, 카운터 및 데이터 종료(End-of-data) 기능을 위해 예비될 수 있다.

[0055] (720)에서 방법(700)은 타깃 하드웨어 디바이스의 제약을 감안하여 구문 트리를 조절하는 동작을 포함할 수 있다. 일례에서 타깃 하드웨어 디바이스(예를 들어, FSM 엔진(200))의 제약은 오토마톤의 구조에 제한을 부과할 수 있다. 이러한 제약이 제한을 부과하는 상황에서 컴파일러는 하드웨어 제약에 순응하도록 오토마톤 내에 생성된 상태 및/또는 전이를 조절하는 동작을 변환 단계에 포함할 수 있다.

[0056] (730)에서, 방법(700)은 속성 세트를 사용하여 각 구문 트리를 분류하는 동작을 포함할 수 있다. 일례에서, 동작은 글루시코브(Glushkov)의 방법과 같은 표준 기술 중 하나를 사용하여 분류될 수 있다.

[0057] (740)에서, 방법(700)은 등가 결정론적 오퍼레이터로 구문 트리의 비 결정론적 오퍼레이터를 대체하는 동작을 포함할 수 있다. 일례에서, 루프와 같은 비 결정론적 오퍼레이터의 특정 유형은 표준 프로그래밍가능한 요소와 함께 카운터를 사용하여 구현될 수 있다. 일례에서, 비 결정론적 오퍼레이터가 카운터와 같은 특수 목적 하드웨어 요소로 구현하기에 적절치 않다면, 비 결정론적 오퍼레이터는 언롤링될 수 있다. 오퍼레이터를 언롤링하는 것은 비 결정론적 오퍼레이터에 대응하는 모든 가능한 상태 조합을 직렬화하는 것에 의해 달성될 수 있다.

[0058] 오토마톤으로 변환될 때 특정 정규 표현은 다수의 상태를 야기할 수 있다. 다수의 상태는 구현을 위해 다수의 일반 목적 요소(102)를 사용할 수 있다. 상태의 수를 감소시켜 사용되는 일반 목적 요소(102)의 수를 감소시키기 위해 특수 목적 하드웨어 요소(112)는 특정 정규 표현을 구현하는데 사용될 수 있다. 예를 들어, 표준 일반 목적 요소(102)로 변환될 때 다수의 상태를 요구할 수 있는 하나의 정규 표현은 양화 표현(quantification expression)이다. 양화 표현은 다수회 하나 이상의 표현을 반복하는 루프 구조에 대응한다. 양화 표현은 언롤링되고 직렬인 다수의 일반 목적 상태로 구현된다. 그러나, 일례에서, 카운터와 같은 특수 목적 하드웨어 요소(예를 들어, 다른 요소(112))는 양화 표현을 구현하는데 사용되는 상태의 수를 감소시키기 위해 양화 표현에 있는 반복된 표현을 이용하는데 사용될 수 있다.

[0059] 양화는 이 기술 분야에 잘 알려져 있는 것이고, 반복된 패턴을 기술하는데 사용된다. 일례로서, "A(B){n1, n2}C"는 일반적인 정규 표현이고, 여기서 A, B 및 C는 서브 표현(sub-expression)이고,"(B){n1, n2}"는 양화를 포함한다. 본 명세서에 설명된 바와 같이, 대문자(upper-case letter)는 정규 표현이나 정규 표현의 일부(예를 들어, 서브 표현)를 나타내는데 사용된다. 이중 인용 마크는 혼동을 회피하기 위하여 정규 표현이나 서브 표현 주위에 추가될 수 있다. 따라서, 표현을 기술하는 대문자는 다수의 입력 심볼을 위한 검색 스트링에 대응할 수 있다. 예를 들어, "A" 표현은 입력 스트링 'abbc'에 대응할 수 있다.

[0060] 나아가, 표현 및 서브 표현 이라는 용어는 본 명세서에서 관계 설명(예를 들어, 서브 표현은 표현의 일부임)을 위한 용도로만 사용되는 것이고, 표현 및 서브 표현 이라는 용어는 임의의 특정 길이, 구문, 또는 문자의 수로 제한되는 것은 아닌 것으로 이해된다. 특히, 소스 코드는 다수의 문자(메타-문자 및 검색 문자를 포함함)를 포함할 수 있고, 이들 문자의 전체 세트 또는 임의의 개별 부분은 "표현"으로 고려될 수 있다. 예를 들어, 이하 각각, 즉 "a(bb|d?){5, 20}c", "(b){0, 10}", "(b|d)" 및 "b"은 표현으로 고려될 수 있다.

[0061] 양화는 regex에서 "(B){n1, n2}"로 표현되고, 여기서 B는 서브 표현이고 n1 및 n2는 이전의 서브 표현이 발생할 수 있도록 허용되는 횟수를 지정하는 정수이다. B는 본 명세서에서 반복된 서브 표현으로 지칭되는데 그 이유는

B는 n1 및 n2에 의해 지정된 횟수만큼 반복된 서브 표현이기 때문이다. 양화 "(B){n1, n2}"와 매칭하기 위하여, 반복된 서브 표현(B)은 n1 내지 n2 횟수에 매칭되어야 한다. 예를 들어, regex "(B){5, 7}"은 서브 표현(B)이 5, 6 또는 7회에 매칭될 것을 요구한다. regex "A(B){n1, n2}C"에서, 서브 표현(A)은 본 명세서에서 구동 표현으로 지칭되는데 그 이유는 서브 표현(A)이 매칭될 때 양화로 전이하기 때문이다. 추가적으로, 양화를 위한 카운트를 반복하고 증분시키는 것을 계속하기 위하여 양화의 반복된 서브 표현(들)은 연속적으로 매칭되어야 한다. 즉, 반복된 서브 표현이 양화의 주어진 루프 동안 매칭되지 않으면, 양화는 종료한다. 일례에서, 심볼 "?"은 양화에 더 대응하며, 여기서 "?" 이전의 심볼은 일(one)회 또는 제로(zero)회 식별될 수 있다.

[0062] 타깃 디바이스가 FSM 엔진(200)이라면, 방법(800)은 특정 양화를 식별하고 이 양화를 FSM 엔진(200) 상의 카운터(224)로 맵핑할 수 있다. 카운터(224)로 특정 양화를 구현하는 것은 상태 기계 요소(204, 205)로 양화를 구현하는 것에 비해 효율적일 수 있다. 따라서, FSM 엔진(200)을 위한 오토마톤과 결과 이미지는 간략화될 수 있다. 예를 들어, 양화를 구현하는 구문 트리의 일부는 구현하는데 다수의 SME(204, 205)를 요구할 수 있다. 그러나, 일례에서, 이들 양화 중 일부는 SME(204, 205)에 의해 요구되는 것보다 더 적은 상태로 카운터(224)를 사용하여 구현될 수 있다.

[0063] 블록(802)에서 컴파일러는 FSM 엔진(200)에서 카운터(224)로 구현을 가능하게 하기 위해 양화에 대응하는 구문 트리의 일부를 식별한다. 구문 트리의 일부가 양화에 대응하지 않으면, 방법(800)은 블록(803)으로 진행하고 여기서 일부는 SME(204, 205)로 구현하기 위해 일반 목적 상태로 변환된다. 구문 트리의 일부가 양화에 대응하면 양화는 식별된 부분이 카운터(224)로 구현될 수 있는지 여부를 결정하기 위해 더 분석된다.

[0064] 양화가 카운터(224)로 구현될 수 있을지 여부를 결정하기 전에, ℒ(B)로 쓴 패턴 'B'의 언어(즉, 'B'가 매칭하는 모든 스트링)가 빈 스트링을 포함한다고 하면, "B{n1, n2}"의 양화는 "B'{0, n2}"로 재기록되고, 여기서 B'는 B의 비어 있지 않은 스트링 버전이고, ℒ(B') = ℒ(B) - Φ 이다. 예를 들어, "(bc|){10, 20}"는 "(bc){0, 20}"로 재기록될 수 있는데 그 이유는 이들 regex는 정확히 동일한 데이터를 수용하기 때문이다. 이후, 주어진 양화 B{n1, n2}에 대해, 양화는 가능하게는 카운터로 구현될 수 있고(방법은 블록 804로 진행됨) 또는 대안적으로 SME를 가지되 카운터 없이 이하의 조건에 따라서 구현될 수 있다(방법은 블록 808로 진행된다):

[0065] 1) (n1 = 0, n2 = -1)이면, 양화는 SME(204, 205)를 가지되 카운터(224) 없이 언롤링된다(블록 808). 여기서 카운터(224)는 필요치 않다.

[0066] 2) (n1 = 1, n2 = -1)이면, 양화는 SME(204, 205)를 가지되 카운터(224) 없이 언롤링된다(블록 808). 여기서 카운터(224)는 필요치 않다.

[0067] 3) (n1 > 1, n2 = -1)이면, 양화는 2개의 regex B{n1-1} 및 B+로 분할되는데, 그 이유는 B{n, -1}가 B{n1-1}B+이기 때문이다. 양화 B{n1-1}는 가능하게는 카운터로 구현될 수 있는(블록 804) 반면, B+ 는 SME(204, 205)를 가지되 카운터(224) 없이 구현된다(블록 808). B+에 대해 카운터(224)는 필요치 않다.

[0068] 4) (n1 = 0, n2 > 0)이면, 양화는 (B{1, n2})? 로 수정되는데, 그 이유는 (B{1, n2})?가 B{0, n2}이기 때문이다. 비 널 가능한(non-nullable) B{1, n2}는 가능하게는 카운터(224)로 구현될 수 있다(블록 804).

[0069] 5) (n1 > 0, n2 > 0)이면, 양화는 가능하게는 카운터(224)로 B{n1, n2}로 구현될 수 있다(블록 804).

[0070] 요약해서, 수정 없이 카운터(224)로 구현될 수 있는 양화(블록 804)는 B{n1, n2}로 재기록될 수 있고, 여기서 B는 널 가능하지 않고 n1 > 0, n2 > 0 및 n1 ≤ n2이다.

[0071] 블록(804)에서, 컴파일러가 카운터(224)로 구현될 수 있는 양화를 일단 식별하면, 컴파일러는 식별된 부분에 따응하는 구문 트리의 부분이 결정론적인지 여부를 결정한다. 식별된 부분이 결정론적이면, 식별된 부분은 하나 이상의 카운터(224)로 구현될 수 있고, 방법(800)은 블록(806, 807)으로 진행하고, 여기서 식별된 부분은 하나 이상의 SME 상태와 함께 하나 이상의 카운터 상태로 변환된다. 식별된 부분이 비 결정론적이면, 식별된 부분은 카운터(224)를 사용하여 구현되지 않고, 방법(800)은 블록(808)으로 진행하고, 여기서 식별된 부분은 도 13에 대하여 후술되는 하나 이상의 SME 상태를 사용하여 언롤링된다.

[0072] 일반적으로, 블록(806)과 블록(808, 810)은 양화를 오토마톤으로 변환하는 2가지 방법에 대응한다. 블록(806)에서 양화는 루프로 양화를 구현하기 위해 하나 이상의 SME 상태와 함께 하나 이상의 카운터 상태를 사용하여 변환된다. 블록(808, 810)에서, 양화는 SME 상태를 사용하되 카운터 상태 없이 양화를 "언롤링"하는 것에 의해 변환된다. 언롤링하는 것은 비양화 구문으로 양화를 재기록하는 것을 포함한다. 예를 들어, regex "(b|c){1, 2}"는 "(b|c)(b|c)?"으로 언롤링될 수 있다. 언롤링의 장점은 (1) 결과적인 오토마톤이 방향성 비순환 그래프

(directed acyclic graph)(DAG)이고, 분석하고 구현하기에 용이할 수 있다는 것과 (2) 결과적인 오토마톤이 특수 목적 요소 대신에 일반 목적 요소, 특히 상태 기계 요소로 구현될 수 있다는 것을 포함한다. 그러나, 언롤링된 양화를 구현하는데 사용되는 일반 목적 상태의 수, 그리하여 상태 기계 요소의 수는 n1 과 n2에 선형이다. 따라서, 상태의 수는 n1 또는 n2가 큰 수인 경우 클 수 있다. 특히 실제 생활 자원이 제한되어 있어 일부 예에서 이 언롤링 기술은 제한된 카테고리의 양화에만 사용된다. 양화를 언롤링하는 것에 관한 보다 상세한 사항은 블록(808, 810) 및 도 13a 내지 도 13c에 대하여 후술된다.

[0073] 그러나, 타깃 디바이스는 카운터(224)와 같은 카운팅 기능을 구현하도록 설계된 특수 목적 요소를 구비할 때 언롤링은 특정 경우에 회피될 수 있다. 이 방법의 장점은 반복된 표현의 더 작은 개수의 사본이 오토마톤에 요구되고 이 사본의 개수는 n1과 n2에 독립적이라는 것이다. 그리하여, 상당한 자원이 절감될 수 있다. 예를 들어, 하나 이상의 카운터(224)는 반복된 표현(들)과 하나 이상의 카운터(224)로 루프를 생성하는 것에 의해 양화를 구현하는데 사용될 수 있다. 반복된 표현(들)이 매칭될 때마다, 카운터(224)는 증분(또는 감분)될 수 있다. 반복된 표현(들)은 다른 매치를 검색하도록 재활성화될 수 있다. 카운터(224)가 양화에 의해 언급된 횟수와 동일하게 증분(또는 감분)되면, 카운터(224)는 양화 이후 상태(들)를 활성화시킬 수 있다. 따라서, 양화는 반복된 표현(들)을 구현하는데 사용되는 SME가 재사용되므로 더 적은 개수의 SME(204, 205)로 구현될 수 있다. 그러나, 전체 오토마톤(예를 들어, 전체 구문 트리에 대응하는)의 병렬화, 즉 동시에 활성화될 수 있는 다수의 상태로 인해, 카운터(224)는 일부 예에서 전체 오토마톤의 결정론적 부분에 대응하는 양화에만 사용될 수 있다.

[0074] 도 9는 양화를 구현하는데 특수 목적 카운터 상태(902)를 사용하여 오토마톤(900)으로 변환된 regex의 일례를 도시한다. 오토마톤(900)은 양화의 두 카운팅 값(예를 들어, n1, n2)이 동일한 regex "A(B){n1, n1}C" 에 대응한다. 두 카운팅 값이 동일하므로, 단일 카운터(224)가 양화를 구현하는데 사용된다. 도 9에 도시된 바와 같이, 양화(900)는 그래프의 노드가 상태 세트에 대응하는 방향성 그래프로 표현될 수 있다.

[0075] regex "A(B){n1, n1}C"는 수 개의 SME 상태(904, 906, 910, 908)와 카운터 상태(902)로 변환된다. SME 상태(904, 906, 908, 910)는 서브 표현("A", "B" 및 "C")에 대응한다. SME 상태(904, 906, 910, 908)는 SME(204, 205)로 구현될 수 있는 반면, 카운터 상태(902)는 카운터(224)로 구현될 수 있다. 오토마톤(910)이 FSM엔진(200) 상에 구현될 때 카운터 상태(902)에 대응하는 카운터(224)는 초기에 값(n1)으로 로딩되고, 카운터(224)의 값이 제로에 도달할 때 제로 카운트 출력을 선언하도록 설정된다. n1이 n2일 때, 카운터(224)는 정지 0 및 펄스 출력 모드로 설정될 수 있고 이는 카운터(224)가 그 값이 일단 제로에 도달하면 출력을 선언하고, 카운터(224)가 리셋될 때까지 카운터(224)는 제로에 있고 어떤 신호도 발행시키지 않는다는 것을 의미한다.

[0076] 오토마톤(900)은 상태(904)에서 시작하여, 서브 표현 "A"과 매칭할 때 상태(906)로 전이한다. 상태(906)에 있는 동안 서브 표현 "B"이 매칭될 때마다 카운터 상태(902)의 입력 포트는 활성화되고 카운터 상태(902)는 일(one)만큼 증분한다. 추가적으로, 서브 표현 "B"이 매칭될 때마다 상태(906)는 자체적으로 활성화하고 상태(910)를 활성화시킨다. 카운터 상태(902)가 제로에 도달하면, 출력이 활성화되고 오토마톤(900)은 서브 표현 "C"을 검색한다. 이하 사이클에서 2개의 시나리오가 발생한다, 즉 제1 시나리오는 "~B"가 매칭될 때 발생한다. "~B"가 매칭되면, 카운터 상태(902)는 리셋되고 그 값은 n1으로 다시 설정된다. 따라서, 그 다음으로 서브 표현 "A"이 매칭될 때 공정은 상태(904)로부터 시작한다. 제2 시나리오에서 상태(906)의 자체 루프는 여전히 활성이고 카운터(902)의 입력 포트는 서브 표현 "B"의 매치시에 계속 트리거된다. 카운터 상태(902)가 펄스 모드에서 구성되므로, 카운터 상태(902)는 출력을 다시 활성화하지 않으나 상태(906)의 자체 루프가 활성으로 유지된다.

[0077] 서브 표현 "B"의 부정 버전(negated version)은 본 명세서에서 "~B"로도 지칭된다. 일례에서, 서브 표현 "B"의 부정 버전은 카운터 상태(902)의 리셋 포트를 활성화시키는데 사용된다. 이것은 "B"가 양화"(B){n1, n1}"의 반복된 표현이어서" B가 아닌 어떤 것(예를 들어, "B"의 부정 버전)이 (상태(906)가 일단 활성화되면) 입력에 수신될 때 양화는 종료하고 이에 따라 카운터가 리셋되기 때문이다. 따라서, 상태(910)가 활성화되면, 카운터 상태(902)는 리셋되고 양화는 서브 표현 "B"의 부정 버전이 매칭될 때 매칭되지 않는다. 일례에서, 반복된 표현(들)은 표준 오토마톤 이론을 사용하여 부정된다.

[0078] 단일 카운터 상태(224)가 n1이 n2일 때 양화를 구현하도록 예시되고 기술되었지만, 다수의 카운터(224)가 단일 카운터(224)에 의해 지원되는 것보다 더 큰 수를 고려하기 위하여 캐스케이드 연결될 수 있는 것으로 인식된다.

[0079] 도 10은 양화로 regex를 구현하기 위해 복수의 특수 목적 카운터 상태(1002, 1004)를 사용하여 오토마톤(1000)으로 변환된 regex의 다른 예를 도시한다. 오토마톤(1000)은 n1이 n2 미만인 regex "A(B){n1, n2}C"에 대응한다. 양화 "(B){n1, n2}"에서 n1이 n2 미만이므로 2개의 카운터 상태(1002, 1004)가 사용된다. 카운터 상태(1002, 1004)는 정지 0 및 홀드 모드로 구성되며, 이는 카운터 상태(1002, 1004)가 제로에 도달할 때 카운터 상

태(1002, 1004)가 출력을 활성화하고, 카운터 상태(1002, 1004)가 리셋되기 전에 카운터 상태(1002, 1004)는 제로에 있고 입력 포트가 활성화될 때마다 출력을 활성화하는 것을 의미한다. 이 예에서, 카운터 상태(1002)로부터 카운터 상태(1004)까지의 레이턴시(latency)는 2개의 사이클이 소요된다.

[0080] 카운터 상태(1002)는 초기에 n1으로 설정되고, 카운터 상태(1004)는 초기에 n2로 설정된다. 서브 표현 "A"이 매칭될 때 오토마톤은 상태(1006)로부터 상태(1008)로 전이한다. 상태(1008)가 활성화되면, 카운터 상태(1002)와 카운터 상태(1004)의 입력 포트는 서브 표현 "B"이 매칭될 때마다 활성화된다. 따라서, 카운터 상태(1002)와 카운터 상태(1004)는 일만큼 증분된다. 카운터 상태(1002)가 제로에 도달하면, 출력은 활성화되고, 오토마톤(1000)은 서브 표현 "C"의 매치를 검색하며 상태(1010)를 활성화한다. 서브 표현 "B"이 n1회 매치되면, 카운터 상태(1004)의 값은 n2-n1이다. 차후에 서브 표현 "B"이 매치될 때마다, 카운터 상태(1002)의 입력 포트는 활성화되고, 카운터 상태(1002)의 값은 제로에 있고 출력은 여전히 활성화된다. 한편, 카운터 상태(1004)는 계속 감분된다. 서브 표현 "B"이 n2회 매치되면, 카운터 상태(1004)가 제로에 도달하고 그 출력은 활성화되어 카운터 값(1002)의 리셋 포트를 구동한다. 카운터 상태(1004)에서 카운터 상태(1002)로의 레이턴시가 2개의 사이클이므로, 카운터 상태(1002)는 출력을 상태(1010)로 계속 활성화시킨다. 그 다음 사이클에서 카운터 상태(1002)는 카운터 상태(1004)의 출력으로부터 리셋되고, 출력은 카운터 상태(1002)로부터 선언되지 않는다. 이하 사이클에서 2개의 시나리오가 발생한다. 제1 시나리오에서 "~B"가 매치된다. 카운터 상태(1002)와 카운터 상태(1004) 모두는 상태(1012)에 의해 리셋되고 그 값은 각각 n1과 n2로 설정된다. 따라서, 그 다음으로 상태(1006)가 활성이고 그 다음으로 서브 표현 "A"이 매치될 때, 상태(1008)는 활성화되고 카운터 상태(1002, 1004)는 다시 감분된다. 제2 시나리오에서 상태(1008)의 자체 루프는 활성화되어 유지되고 두 카운터 상태(1002, 1004)의 입력 포트가 활성화된다. 카운터 상태(1004)가 출력을 계속 활성화하므로, 카운터 상태(1002)는 계속해서 리셋되고, 상태(1008)의 자체 루프가 활성화되는 한, 출력을 활성화하지 않는다.

[0081] 나아가, 상태(1008)가 활성화되는 동안 서브 표현 "B"의 매치는 상태(1012)를 활성화시킨다. 일단 상태(1012)가 활성화되고 "~B"가 매치되면, 카운터 상태(1002, 1004)가 리셋되고 양화는 매치되지 않는다. "B"는 양화 "(B){n1, n2}"의 반복된 표현이므로 서브 표현 "B"의 부정 버전이 사용된다. 따라서, 상태(1008)에서 표현 'B'은 n1 내지 n2 횟수에서 반복적으로 매칭될 수 있다. 단일 카운터가 하한 임계값(예를 들어, n1)과 상한 임계값(예를 들어, n2)을 각각 구현하도록 예시되고 기술되었지만, 이 기술 분야에 통상의 지식을 가진 자라면 단일 카운터에 의해 지원되는 것보다 더 많은 수를 카운트하기 위하여 다수의 카운터를 캐스케이드 연결할 수 있다는 것을 인식할 수 있을 것이다.

[0082] 카운터 상태를 사용하여 양화를 변환하기 전에, 컴파일러는 블록(804)에서 양화에 대응하는 오토마톤이 결정론적인지 여부를 결정한다. 일례에서, 오토마톤은 표현이 후술되는 무 접두사(no-prefix) 및 무 재진입(no re-entrance) 조건을 모두 충족하면 결정론적이다. 즉, 양화를 카운터(224)로 맵핑하기 위하여 양화는 후술되는 바와 같이 무 접두사 및 무 재진입 조건을 충족하여야 한다.

[0083] 도 10의 오토마톤(1000)을 참조하면, 무 재진입 조건이 상태(1006)로부터 상태(1008)로의 에지는 활성화될 수 없고 카운터 상태(1002)는 활성화될 것(예를 들어, 카운터 상태(1002)가 카운팅될 것)을 요구한다. 즉, 양화가 이미 처리되고 있는 동안 양화를 위한 구동 표현이 매치될 수 있는지 여부가 결정된다. 구동 표현을 매치하는 것은 양화 직전의 상태가 양화에 대응하는 상태로 전이하는 것을 의미한다. 따라서, 양화는 카운터 상태가 반복된 표현을 여전히 처리하고 있는 동안 "재진입"할 수 있다. FSM 엔진(200)의 이 예에서 카운터(224)는 임의의 주어진 시간에 단일 루프만을 구현할 수 있으므로, 루프가 이미 처리되고 있는 동안 양화로 전이하는 것은 카운터(224)로 하여금 주어진 루프 동안 부정확하게 카운트하게 할 수 있다.

[0084] 도 11a 및 도 11b는 무 재진입 조건을 더 설명하는데 사용될 수 있는 오토마톤(1100, 1114)을 도시한다. 도 11a는 양화에 대응하는 오토마톤이 결정론적인지 여부를 결정하기 위해 컴파일러가 분석할 수 있는 구문 트리로 양화에 대응하는 예시적인 오토마톤(1100)을 도시한다.

[0085] 오토마톤(1100)은 정규 표현 abb?(b|c){1, 2}" 에 대응하고 시작 상태(1102)와 최종 상태(1112, 1104)를 포함한다. 최종 상태는 이중 원으로 도 11a에 식별된다. 시작 상태(1102)는 초기에 활성화되어 입력 심볼 'a'에서 상태(1106)로 전이한다. 상태(1106)는 입력 심볼 'b'에서 두 상태(1108)와 상태(1110)로 전이한다. 상태(1108)는 입력 심볼 'b'에서 상태(1110)로 전이하고, 상태(1110)는 입력 심볼 'b' 또는 'c'에서 상태(1112)로 전이한다. 또한, 오토마톤(1100)은 입력 심볼 'b' 또는 'c'에서 상태(1112)로부터 상태(1104)로 전이한다.

[0086] 오토마톤(1100)은 무 재진입 조건에 순응하는지 체크하기 위해 regex "abb?(b|c){1, 2}"에 대한 오토마톤을 포함한다. 오토마톤(1114)은 오토마톤(1100)의 regex "abb?(b|c){1, 2}"로부터 유도된 regex SS "abb?(b|

c){2}"의 오토마톤을 포함한다. SS(M, N)는 M, N으로부터 유도된 regex로 한정된다. 구동 단계는, 1) M과 N을 연쇄 연결하고, 그 결과는 "MN"이라고 표시되고, 2) "MN"에 대해 오토마톤을 구성하고, 이는 A(MN)이라고 표시되고, 3) A(MN)을 다음과 같이 수정하는데, 즉 a) A(MN)의 시작 상태를 모든 다른 상태를 구동하게 하고, b) "N"에 대응하는 모든 상태를 최종 상태로 만들고, 마지막으로, 4) 수정된 오토마톤에 대한 regex를 SS(M, N)로 표시하는 것을 포함한다. SS(M, N)의 허용된 데이터는 "MN"의 임의의 상태로부터 시작하고 N의 임의의 상태에서 종료하는 서브 스트링으로 구성된다.

[0087] 무 재진입 조건은 이하와 같이 한정될 수 있다. 양화 "AB{n1, n2}C"로 정규 표현이 주어지면, 무 재진입 조건은 $£(SS(A, B\{n1, n2\}) \cap £(A) = \emptyset$ 을 요구한다. 다시 말해, 서브 표현 "A"이 매치되고 카운터 상태(1002)가 카운트를 시작하면, 무 재진입 조건을 충족하기 위하여 상태(1006)로부터 상태(1008)로의 에지는 "B{n1, n2}"이 수행(매치 또는 실패)될 때까지 다시 활성화되지 않는다. 예를 들어, "abb"$\in £$("abb?") $\cap £$(SS("abb?", "(b|c){2}"), 및 이에 따라 "abb?(b|c){1, 2}"은 카운터(224)로 올바르게 구현되지 않는다.

[0088] 이제 도 12를 참조하면, 무 접두사 조건이 오토마톤(1200)을 참조하여 설명된다. 무 접두사 조건은 £(B)의 임의의 스트링이 £(B)의 다른 스트링의 접두사이지 않아서 B가 카운터(들)로 하여금 하나를 초과하여 카운팅하지 않게 하는 것을 보장하는 것을 말한다. 다시 말해, 양화는 양화의 제1 반복된 서브 표현이 양화의 제2 반복된 서브 표현의 접두사일 때 카운터(224)로 구현되지 (그리하여 이로 변환되지) 않는다. 형식적 명제(statement)는 모든 l_i에 대해, $l_j \in £(B)$, $l_i \neq l_j$이고, $\{l_i.^*\} \cap \{l_j.^*\} = \emptyset$ 을 요구한다.

[0089] 예를 들어, regex "a(b|bc){3}"은 무 접두사 조건을 충족하지 않는다. 따라서, regex "a(b|bc){3}"은 카운터 상태를 사용하여 변환되지 않을 수 있어서, 카운터(224)로 구현되지 않을 수 있다. 대신, regex "a(b|bc){3}"은 임의의 카운터 상태 없이 일반 목적 상태로 변환될 수 있다.

[0090] regex "a(b|bc){3}"이 카운터(224)로 구현된다면, 입력 'abbc'은 거짓으로 매치될 수 있다. 예를 들어, 오토마톤(1200)은 카운터 상태(1212)를 사용하여 regex "a(b|bc){3}"의 가정적 변환의 결과이다. 후술되는 바와 같이, 이 변환 결과는 카운터 상태(1212)의 부정확한 성능을 초래한다. 상태(1202)는 초기에 활성화되고 입력 "a"에서 상태(1202)는 상태(1204)를 활성화시킨다. 상태(1204)가 활성화되면, 입력 "b"에서 상태(1204)는 상태(1206, 1208)를 활성화시키고 상태(1204) 그 자체를 재활성화시킨다. 또한, 입력 "b"에서 상태(1204)는 카운터(1212)의 입력 포트를 활성화시키고 여기서 카운터 상태(1212)의 초기 값은 3이고 이후 2로 감소된다. 상태(1204, 1206, 1208)가 활성화되면, 카운터 상태(1212)의 입력 포트는 다른 입력 "b"에서 다시 상태(1204)에 의하여 활성화되고, 카운터 상태(1212)의 값은 1로 감소된다. 이 점에서 상태(1204, 1206, 1208)가 활성화된다. 이후, 입력 값 "c"이 카운터 상태(1212)의 입력 포트로 하여금 상태(1208)에 의해 활성화되어 카운터(1212)의 값을 0으로 감소시키게 한다. 카운터(1212)의 값이 제로인 경우 출력은 활성화되고 상태(1214)는 활성화되어 매치를 나타낸다. 그러나, 이 매치는 시퀀스 "abbc"가 regex"a(b|bc){3}"을 충족하지 않을 때 입력 "abbc"가 매치를 유발한 것이므로 거짓 긍정이다. 따라서, regex "a(b|bc){3}"은 무 접두사 조건을 충족하지 않고, 카운터 상태를 사용하여 변환되지 않으며 카운터(224)로 구현되지 않는다.

[0091] 양화가 블록(804)에서 무 접두사 조건과 무 재진입 조건을 충족하면, 양화는 블록(806)에서 특수 목적 카운터 상태를 사용하여 변환된다. 양화는 상기 도 10 및 도 11에 대하여 설명된 바와 같이 변환된다. 그러나, 양화가 무 접두사 또는 무 재진입 조건을 충족하지 않으면, 양화는 블록(808, 810)에서 양화를 언롤링하고 일반 목적 상태와 무 카운트 상태(224)로 변환하는 것에 의해 변환된다. 양화는 이에 따라 SME(204, 205)로 구현되고 카운터(224)로 구현되지 않는다.

[0092] 블록(808)을 다시 참조하면, 단일 개수의 루프로 매칭될 수 있는 양화는 직렬로 링크된 복수의 반복된 서브 표현을 가지는 오토마톤을 형성하도록 언롤링된다. 단일 개수의 루프를 가지는 양화는 n1이 n2인 양화에 대응한다. 예를 들어 양화 "B{n1}"는 B의 n1개의 사본을 가지는 "BB...B"로 언롤링될 수 있다.

[0093] 블록(810)에서, 다수 개의 루프로 매칭될 수 있는 양화는, n1이 n2와 같지 않을 때, 그리고 n1이 1이고 n2가 1보다 클 때 언롤링된다. n1이 1보다 클 때, 양화는 n1-1개의 루프로 매칭될 수 있는 제1 양화와, 1 내지 n2-n1+1개의 루프로 매칭될 수 있는 제2 양화로 분할된다. 예를 들어, 양화 B{n1, n2}, 여기서 n1 > 1, n2 > 1 및 n1 < n2는 다수 개의 루프로, 특히 n1 내지 n2개의 루프로 매칭될 수 있다. 이 양화 B{n1, n2}는 이하 양화 B{n1 - 1}B{1, n2 - n1 + 1}로 분할될 수 있다. 제1 양화는 n1-1개와 같은 다수의 루프로 매칭될 수 있는 반복된 서브 표현(B)이다. 이 제1 양화는 1 내지 n2-n1+1개의 다수의 루프에 의하여 매칭될 수 있는 반복된 서브 표

현을 구비하는 제2 양화와 연쇄적으로 연결된다. 제1 양화(B{n1-1})는 (1302)에서 상태로 언롤링된다.

[0094] 제2 양화 B{1, n2-n1 + 1}는 결과적인 오토마톤의 진입 차수(in-degree) 및/또는 진출 차수(out-degree)에 기초하여 언롤링될 수 있다. 양화를 언롤링하는 것은 큰 진입 차수 또는 큰 진출 차수를 가지는 상태를 생성할 수 있다. 일례에서, 진입 차수는 오토마톤의 상태로 전이하는 개수에 대응하고, 진출 차수는 오토마톤의 상태로부터 전이하는 개수에 대응한다. 따라서, 제2 양화는 제2 양화를 오토마톤으로 변환할 때 상태로 들어가게 전이하는 것(진입 차수) 또는 상태로부터 나오게 전이하는 것(진출 차수)을 제어하도록 언롤링될 수 있다. 예를 들어, 양화는 각 언롤링된 상태의 진입 차수를 임계값 수 미만으로 제한하도록 언롤링될 수 있다. 진입 차수를 제한하는 것은 예를 들어 타깃 디바이스 내 요소의 측면 및/또는 제한을 고려하도록 수행될 수 있다. 나아가, 언롤링 동안 진입 차수를 제한하는 것은 컴파일러에 대한 후속 처리를 감소시킬 수 있다.

[0095] 일례로서, 양화 B{1, n2-n1+1}를 언롤링할 때, 오토마톤은 진입 차수와 진출 차수 사이에 트레이드오프(trade-off)로 생성된다. 따라서, 진입 차수를 감소시키는 것은 진출 차수를 증가시킬 수 있고 진출 차수를 감소시키는 것은 진입 차수를 증가시킬 수 있다. 일례에서, 양화 B{1, n2-n1+1}의 루프 구조를 언롤링하기 위하여 오토마톤으로 하여금 k개의 연쇄 연결된 B의 임의의 스트링을 허용하도록 언롤링된 상태로 또는 언롤링된 상태로부터 다수의 전이가 이루어지며, 여기서 1<=k<=n2-n1+1이다. 언롤링된 상태로 또는 언롤링된 상태로부터 전이가 이루어지는지 여부를 제어하는 것은 오토마톤에 대한 진입 차수/진출 차수를 제어하는데 사용될 수 있다.

[0096] 방법(800)이 단일 양화에 대응하는 것으로 기술되어 있으나, 방법(800)은 구문 트리 내에 복수의 양화에 대해 반복될 수 있고, 결과적인 개별 오토마톤은 더 큰 오토마톤으로 링크될 수 있다.

[0097] 도 13a는 표현(AB{1, 4})이 진입 차수를 최소화하도록 언롤링된 오토마톤(1300)의 일례를 도시한다. 진입 차수를 최소화하기 위해 언롤링된 양화로부터 초래되는 오토마톤은 본 명세서에서는 또한 분산 패턴(scatter pattern)이라고도 지칭된다. 표현(AB{1, 4})의 언롤링된 버전인 분산 패턴은 표현 A((((B?) B)? B)? B)에 직접 대응하고, 이 표현에 오토마톤(1300)이 대응한다. 오토마톤(1300)은 양화 B{1, 4}를 위한 구동 상태(1302)와, 양화의 제1 상태(1304)와, 양화의 마지막 상태(1308)를 포함하는 복수의 언롤링된 상태(1304 내지 1308)를 포함한다. 일례에서, 표현(A, B) 각각은 미도시된 더 작은 오토마톤을 위한 다수의 상태에 대응할 수 있다. 오토마톤(1300)의 진입 차수를 최소화하기 위하여 양화를 위한 전이는 제1 상태(1304)로부터 다른 언롤링된 상태(1305 내지 1308)로 진출 전이(out-transition)로 할당된다. 따라서, 제1 상태(1304)는 큰 진출 차수(4개의 진출 전이)를 구비하며, 모든 양화 상태(1304 내지 1308)는 작은 진입 차수(1 또는 2개의 진입 전이(in-transition))를 구비한다.

[0098] 도 13b는 표현(AB{1, 4}이 진출 차수를 최소화하기 위해 언롤링된 오토마톤(1310)의 일례를 도시한다. 진출 차수를 최소화하기 위해 언롤링된 양화로부터 초래되는 오토마톤은 본 명세서에서는 또한 병합 패턴(merge pattern)이라고도 지칭된다. 표현(AB{1, 4})의 언롤링된 버전의 병합 패턴은 언롤링된 표현(AB(B(B(B)?)?)?)에 직접 대응한다. 오토마톤(1308)은 상태(1302, 1304 내지 1308)들 사이에 상이한 전이를 가지는 오토마톤(1300)과 동일한 상태(1302, 1304 내지 1308)를 포함한다. 오토마톤(1310)의 진출 차수를 최소화하기 위하여 양화를 위한 전이는 양화의 마지막 상태(1308)로 가는 입력 전이로 할당된다. 입력 전이는 언롤링된 상태(1304 내지 1307) 각각으로부터 온다. 따라서, 양화 상태(1304 내지 1308) 전부는 작은 진출 차수(1 또는 2개의 진출 전이)를 구비하지만, 양화의 마지막 상태(1308)는 큰 진입 차수(4개의 진입 전이)를 구비한다.

[0099] 일례에서, 양화를 구비하는 표현은 진출 차수 또는 진입 차수 중 하나를 임계값 미만으로 제한하도록 언롤링된다. 일례에서, 진입 차수를 임계값으로 제한하도록 표현(AB{1, n1})을 언롤링하기 위하여 임계값까지 양화(B{1, n1})를 위한 다수의 전이는 양화(B{1, n1})의 마지막 상태로 가는 진입 전이로 할당될 수 있고, 다른 전이는 양화(B{1, n1})의 제1 상태를 위한 진출 전이로 할당될 수 있다. 역으로, 진출 차수를 임계값으로 제한하기 위하여 표현(AB{1, n1})을 언롤링하기 위하여 임계값까지 양화(B{1, n1})를 위한 다수의 전이는 양화를 위한 제1 상태에 가는 진출 전이로 할당될 수 있고, 다른 전이는 양화(B{1, n1})의 마지막 상태를 위한 진출 전이로 할당될 수 있다.

[0100] 도 13c는 임의의 상태를 위한 진입 전이를 3 이하로 제한하기 위해 표현(AB{1, 4})이 언롤링되는 오토마톤(1312)의 다른 예를 도시한다. 오토마톤(1312)은 상태(1302, 1304 내지 1308)들 사이에 상이한 전이를 가지는 오토마톤(1300, 1308)과 동일한 상태(1302, 1304 내지 1308)를 포함한다. 일례에서, 오토마톤(1312)의 진입 차수를 3개 이하의 진입 전이로 제한하기 위하여 양화를 위한 전이는 3의 제한값에 도달할 때까지 양화(B{1, 4})의 마지막 상태(1308)로 가는 진입 전이로 초기에 할당되고, 다른 전이는 양화 상태(1304 내지 1308)로부터 진출 전이로 할당된다. 따라서, 양화의 마지막 상태(1308)와, 다른 양화 상태(1304 내지 1307)는 3의 제한값 이하

의 진입 차수를 가지고, 제1 상태(1304)는 3의 진출 차수를 구비한다.

[0101] 다른 예에서, 표현의 진입 차수와 진출 차수는 서로 특정 비(ratio)(예를 들어, 1대1, 2대1)로 설정될 수 있다. 또 다른 예에서, 표현의 진입 차수와 진출 차수는 임계값이 진입 전이 또는 진출 전이에 도달할 때까지 서로 특정 비로 설정될 수 있고, 다른 비도 사용될 수 있으며, 또는 모든 전이는 진입 전이 또는 진출 전이로 각각 할당될 수 있다.

[0102] 네트리스트는 연결 인스턴스이며, 여기서 인스턴스는 하드웨어 요소에 대응하고, 방향성 에지는 하드웨어 요소를 연결하는 네트(net)이다. 상태가 다른 상태를 구동하면, 2개의 상태가 할당된 SME들 사이에 전기적 연결이 있을 수 있다. 대부분의 물리적 디바이스는 하드웨어 성분들 사이에 연결의 개수에 일부 제한을 가지고 있다. 오토마톤으로부터 물리적 디바이스로 가능한 맵핑을 얻기 위하여 오토마톤은 모든 상태의 진입 차수가 하드웨어 연결 제한을 충족하도록 변환되어야 한다.

[0103] 전술된 바와 같이, 컴파일러는 SME 중 어느 것이 만약 있다면 FSM 엔진(200)의 제한에 기초하여 서로 그룹화 될 수 있는지를 결정한다. 따라서, GOT(210)에서 컴파일러는 SME 중 어느 것이 GOT(210)에서 SME(204, 205)에 대한 출력 제한에 기초하여 서로 쌍으로 형성될 수 있는지를 결정한다.

[0104] 하나의 이러한 실시형태에서, FSM 엔진(200)은 모든 하드웨어 성분에 대해 제한된 연결을 가진다. 오토마톤으로부터 FSM 엔진(200)으로 가능한 맵핑을 얻기 위하여, 오토마톤은 모든 상태의 진입 차수가 연결 제한을 충족하도록 변환되어야 한다.

[0105] 그리하여, 일례에서 오토마톤의 진입 차수는 타깃 디바이스의 하드웨어 제약에 기초하여 제한될 수 있다. 일례에서 2개 레벨의 구조가 FSM 엔진(200)에 사용되어 SME(204, 205)의 구동 입력을 제한할 수 있다. 제일 먼저, GOT(210)에 의해 제공된 OR 게이트(230)를 레버리지(leverage)할 수 있고, 이는 SME 상태로부터 출력을 단일 출력으로 쌍으로 형성하는 것에 의해 최대 50%만큼 진입 차수를 감소시킬 수 있다. 다른 성분은 복소 논리 함수를 제공하도록 구성될 수 있는 부울리안이다. 이 예에서는, 단지 단일 OR 성분으로만 이를 고려한다. 이하 알고리즘에서, 보수적인 방식으로 GOT(210)의 사용을 예측한다. 이 알고리즘에서 예측된 진입 차수가 FSM 엔진(200)의 제약을 초과하면, 일부 상태는 다수의 상태로 분할되거나, 또는 부울리안 OR이 필요에 따라 진입 차수를 감소시키도록 삽입된다.

[0106] 상태 분할의 기본적인 아이디어는 상태를 다수의 상태로 분할하고, 각 분할된 상태의 진입 차수가 제약을 충족하도록 미리 분할된 상태의 구동 입력을 분할된 상태에 분배하는 것이다. 오토마톤이 방향성 비순환 그래프(DAG)라면, 오토마톤의 간단한 폭 제1 횡단이 이 문제를 해결할 수 있다. 그러나, 루프(예를 들어, 양화)가 존재할 때, 분할은 상태의 개수를 지수적으로 증가시키거나 또는 가능한 해법이 가능하지 않는 상황을 야기할 수 있다. 부울리안 논리는 증가된 상태를 감소시키는 것을 도와주어서 이 상황을 완화시킬 수 있다. 일 실시형태에서, 부울리안 OR 또는 그 등가물이 루프 조건을 처리하는데 사용된다.

[0107] 상태를 분할하는 일례는 도 14a 및 도 14b에 도시된다. 도 9a에 도시된 예에서, 8개의 상태(1430)는 하나의 상태(1432)를 공급하며, 이는 다시 2개의 상태(1434)를 공급한다. 도 9b에 도시된 바와 같이, 전술된 바와 같이 분할된 상태는 2개의 새로운 상태(1436)(C2, C3)의 추가를 초래한다. 그러나, 이제 3개의 상태는 2개의 상태(1434)를 공급한다.

[0108] 도 5를 다시 참조하면, GOT(210)는 SME(204, 205)에 출력 제한을 구비한다. 특히, GOT(210)는 2개의 SME(204, 205)에 의해 공유된 단일 출력(218)을 구비한다. 따라서, GOT(210)에서 각 SME(204, 205)는 출력(218)을 독립적으로 구동할 수 없다. 이 출력 제한은 SME 상태 중 어느 것이 GOT 인스턴스로 서로 쌍으로 형성할 수 있는 것인지를 제한한다. 특히 외부 SME 상태(예를 들어, GOT 인스턴스의 외부 SME에 대응하는 SME 상태)의 상이한 세트를 구동(예를 들어, 전이, 활성화)하는 2개의 SME 상태는 GOT 인스턴스에서 서로 쌍으로 형성될 수 없다. 그러나, 이러한 제한은, GOT(210)가 스위치(240)를 통해 이 기능을 내부에서 제공할 수 있으므로, 2개의 SME 상태가 서로 또는 자체 루프를 구동하는지 여부를 제한하지 않는다. FSM 엔진(200)은 SME(204, 205)에 대응하는 특정 물리적 설계를 가지는 것으로 기술되었으나, 다른 예에서, SME(204, 205)는 다른 물리적 설계를 구비할 수 있다. 예를 들어, SME(204, 205)는 SME(204, 205)의 3개 이상의 세트로 서로 그룹화될 수 있다. 추가적으로, 일부 예에서, SME(204, 205)으로부터 출력(226, 228)에 제한이 있거나 없이 SME(204, 205)의 입력(214, 216)에 제한이 있을 수 있다.

[0109] 그러나, 어느 경우이든, 컴파일러는 SME 상태 중 어느 것이 FSM 엔진(200)의 물리적 설계에 기초하여 서로 그룹화될 수 있는 것인지를 결정한다. 따라서, GOT 인스턴스에서, 컴파일러는 SME 상태 중 어느 것이 SME(204, 20

5)에 대한 출력 제한에 기초하여 GOT(210)로 서로 쌍으로 형성될 수 있는지를 결정한다. 일례에서, GOT(210)의 물리적 설계에 기초하여 2개의 SME 상태가 GOT(210)를 형성하도록 서로 쌍으로 형성될 수 있는 5개의 상황이 존재한다.

[0110] 제1 및 제2 SME 상태가 GOT(210)로 서로 쌍으로 형성될 수 있는 제1 상황은 제1 또는 제2 SME 상태 중 어느 것도 최종 상태가 아닌 경우 및 제1 및 제2 SME 상태 중 하나가 제1 또는 제2 SME 상태와는 다른 임의의 상태를 구동하지 않을 때 발생한다. 일례로서, 제1 상태는 제1 상태가 제2 상태로 전이할 때 제2 상태를 구동하는 것으로 고려된다. 이 제1 상황이 발생하면, 제1 및 제2 SME 상태 중 최대 하나가 외부 상태(들)를 구동한다. 따라서, 제1 및 제2 SME 상태는 GOT(210)의 출력 제한에 의해 영향을 받지 않고 서로 쌍으로 형성될 수 있다. 그러나, SME(204, 205)를 서로 내부적으로 연결할 수 있는 GOT(210)의 능력으로 인해, 제1 및 제2 SME 상태는 서로 구동하고 자체 루프 연결되어 자체적으로 구동하는 것이 허용된다. 오토마톤 용어에서 제1 SME 상태(상태 q1에 대응함) 및 제2 SME 상태(상태 q2에 대응함)는 q1과 q2 중 어느 것도 최종 상태에 있지 않고 δ(q1) - {q1, q2}가 비어 있을 때 또는 δ(q2) - {q1, q2}가 비어있을 때 서로 쌍으로 형성될 수 있다.

[0111] 제1 및 제2 SME 상태가 GOT(210)로 서로 쌍으로 형성될 때의 제2 상황은 제1 또는 제2 SME 상태 중 어느 것도 오토마톤에서 최종 상태에 있지 않을 때 및 제1 및 제2 SME 상태 모두가 동일한 외부 상태를 구동할 때 발생한다. 본 명세서에 사용된 바와 같이 외부 상태는, 예를 들어, GOT 인스턴스에서 제1 및 제2 SME 상태가 서로 또는 자체 루프 연결되어 구동하는지에 상관없이 GOT 인스턴스의 외부 상태에 대응한다. 여기서 다시, GOT(210)의 출력 제한은, 제1 및 제2 SME 상태가 동일한 외부 상태를 구동하므로, 제1 및 제2 SME 상태에 영향을 미치지 않는다. 또한, SME(204, 205)를 서로 내부적으로 연결할 수 있는 GOT(210)의 능력으로 인해, 동일한 상태를 구동하는데 있어서 제한은 제1 및 제2 상태가 서로 또는 자체 루프 연결되어 구동하는지 여부를 포함하지 않는다. 오토마톤 용어를 사용하면, 제1 SME 상태(상태 q1에 대응하는) 및 제2 SME 상태(상태 q2에 대응하는)는, q1과 q2 중 어느 것도 최종 상태에 있지 않고 δ(q1) - {q1, q2} = δ(q2) - {q1, q2}일 때 서로 쌍으로 형성될 수 있다.

[0112] 제1 및 제2 SME 상태가 GOT(210)로 서로 쌍으로 형성될 수 있는 제3 및 제4 상황은, 제1 및 제2 SME 상태 중 어느 하나가 최종 상태에 있고 제1 및 제2 SME 상태 중 다른 하나가 임의의 외부 상태를 구동하지 않을 때 발생한다. 즉, 제1 SME 상태(상태 q1에 대응하는) 및 제2 SME 상태(상태 q2에 대응하는)는, q1이 최종 상태에 있고 δ(q2) - {q1, q2}가 비어 있을 때, 또는 q2가 최종 상태에 대응하고, δ(q1) - {q1, q2}가 비어있을 때 서로 쌍으로 형성될 수 있다. 최종 상태는 regex에 매치의 표시(indication)를 출력하므로, 최종 상태에 대응하는 SME 상태는 매치를 나타내기 위하여 GOT(210)의 출력(218)의 독립적인 사용을 가져야 한다. 따라서, GOT(210)에서 다른 SME 상태는 출력(218)을 사용하는 것이 허용되지 않는다.

[0113] 제1 및 제2 SME 상태가 GOT(210)로 서로 쌍으로 형성될 수 있을 때의 제5 상황은, 제1 및 제2 SME 상태 모두가 오토마톤에서 최종 상태에 대응하고 제1 및 제2 SME 상태 모두가 동일한 외부 상태를 구동할 때 발생한다. 오토마톤 용어를 사용하면 제1 상태(상태 q1에 대응하는) 및 제2 SME 상태(상태 q2에 대응하는)는, q1 및 q2 모두가 최종 상태이고 δ(q1) - {q1, q2} = δ(q2) - {q1, q2}일 때 서로 쌍으로 형성될 수 있다.

[0114] 일단 컴파일러가 하나 이상의 SME 상태가 서로 쌍으로 형성될 수 있는지 여부를 결정하면, 컴파일러는 SME 상태를 GOT 인스턴스로 쌍을 형성한다. 일례에서, 컴파일러는 GOT 인스턴스를 형성하도록 쌍으로 형성될 수 있도록 결정된 순서로 SME 상태를 GOT 인스턴스로 쌍으로 형성한다. 즉, 2개의 특정 SME 상태가 서로 쌍으로 형성될 수 있는 것으로 결정되면, 이들 2개의 SME 상태는 GOT 인스턴스로 쌍으로 형성될 수 있다. 2개의 SME 상태가 GOT 인스턴스를 형성하도록 쌍으로 형성되면, 쌍을 이루는 SME 상태는 다른 SME 상태와 쌍을 형성하는데 이용될 수 없다. 이 공정은 쌍을 형성할 SME 상태가 더 이상 남아있지 않을 때까지 지속될 수 있다.

[0115] 일례에서, 컴파일러는 SME 중 어느 것이 GOT 인스턴스로 서로 쌍으로 형성될 것인지를 결정하는데 그래프 이론을 사용한다. 특정 SME만이 서로 쌍을 이룰 수 있으므로, 일부 SME 쌍은 다른 SME가 자기 자신의 GOT 인스턴스로 구현되게 하고 GOT 인스턴스에서 다른 SME 위치는 미사용되어 폐기되게 할 수 있다. 그래프 이론은 네트리스트의 GOT 인스턴스의 미사용된 SME 인스턴스의 수를 감소시키는 것에 의해 GOT(210)로 SME 사용을 최적화하는데(예를 들어, 미사용된 SME의 개수를 감소시키는데) 사용될 수 있다. 그래프 이론을 사용하면, 컴파일러는 전술된 FSM 엔진(200)의 물리적 설계에 따라 SME 상태들 사이에 모든 가능한 쌍을 제일 먼저 결정한다. 컴파일러는 이후 그래프의 정점(vertices)이 SME 상태에 대응하고 그래프의 에지(edge)가 SME 상태의 가능한 쌍에 대응하는 그래프를 생성한다. 즉, 2개의 SME 상태가 GOT 인스턴스로 서로 쌍으로 형성될 수 있는 것으로 결정되면, 2개의 대응하는 정점은 에지에 연결된다. 따라서, 그래프는 SME 상태의 모든 가능한 쌍을 포함한다.

[0116] 컴파일러는 SME 상태 중 어느 것이 GOT(210)로 서로 쌍으로 형성될 것인지를 식별하기 위해 그래프의 정점을 매칭시키는 것을 찾아낼 수 있다. 즉, 컴파일러는 그래프의 매칭 정점들 사이에 2개의 에지들이 공통 정점을 공유하지 않도록 에지(및 그리하여 정점의 쌍)를 식별한다. 일례에서, 컴파일러는 그래프의 최대 매칭을 찾아낼 수 있다. 다른 예에서, 컴파일러는 그래프의 최대 매칭을 찾아낼 수 있다. 최대 매칭은 최대로 가능한 수의 에지를 포함하는 매칭이다. 많은 최대 매칭이 있을 수 있다. 일반적인 그래프에서 최대 매칭을 찾는 문제는 다항식 시간(polynomial time)으로 해결될 수 있다.

[0117] 모든 매칭 정점이 (예를 들어, 최대 매칭으로) 식별되면, 매칭 정점에 대응하는 SME 상태의 각 쌍은 GOT 인스턴스로 맵핑된다. 매칭되지 않는 정점에 대응하는 SME 상태는 자기 자신의 GOT 인스턴스로 맵핑된다. 즉, 매칭되지 않는 정점에 대응하는 SME 상태는 GOT 인스턴스에서 SME 위치의 하나로 맵핑되고, GOT 인스턴스에서 다른 SME 위치는 미사용된다. 따라서, 네트리스트(N)와 및 매칭 정점(M)의 대응하는 세트가 주어지면, 사용된 N의 GOT 인스턴스의 수는 $|Q| - 1 - |M|$과 같으며, 여기서 Q는 오토마톤의 상태 세트이고, "-1"은 이 예에서 오토마톤의 시작 상태가 SME 상태에 대응하지 않기 때문이다.

[0118] 일례에서, 네트리스트(N)는 GOT 인스턴스의 최소 수를 사용하는 G의 최대 매칭 M으로부터 구성된다. 이것은 다음에 의해 입증될 수 있다, 즉 GOT 인스턴스의 더 적은 수를 사용하는 다른 네트리스트(N')가 존재하면 대응하는 매칭을 M'로 표시한다. N'의 GOT 인스턴스의 수는 $|Q| - 1 - |M'|$이므로, $|M| < |M'|$가 된다. 이것은 M이 최대 매칭이라는 사실과 충돌한다. 그리하여 네트리스트(N)는 GOT 인스턴스의 최소 수를 사용한다.

[0119] SME 상태가 GOT 인스턴스로 쌍으로 형성되면, GOT 인스턴스, 카운터 인스턴스 및 논리 인스턴스는 오토마톤의 상태들 사이의 전이에 따라 연결된다. 각 GOT(210)는 단일 출력을 가지므로, 네트리스트에서 각 GOT 인스턴스는 다른 인스턴스에 연결하기 위해 단일 출력 포트를 구비한다. 따라서, 제1 GOT 인스턴스에서 SME 상태가 제2 GOT 인스턴스에서 SME 상태를 구동하면, 제1 GOT 인스턴스의 출력 포트는 제2 GOT 인스턴스의 입력에 연결된다.

[0120] 도 15a 및 도 15b는 균일한 오토마톤으로부터 생성된 예시적인 네트리스트(1500, 1502)를 도시한다. 네트리스트(1500, 1502)는 SME 인스턴스(1506, 1508, 1510, 1512, 1514)를 포함한다. 네트리스트(1500)는 비-최적의 네트리스트의 일례이다. 네트리스트(1500)는 미사용된 3개의 SME 인스턴스(1518)를 남겨둔 채 4개의 GOT 인스턴스(1516)를 사용한다. 그러나, 네트리스트(1502)는 최대 매칭을 식별하기 위해 그래프 이론을 사용하여 생성된 최적의 네트리스트의 일례이다. 네트리스트(1502)는 3개의 GOT 인스턴스(1516)를 사용하며 단일 미사용된 SME 인스턴스(1518)를 구비한다. 네트리스트(1502)에서 인스턴스(1510)는 (예를 들어, 스위치(240)를 통해) GOT 인스턴스의 내부 연결을 통해 인스턴스(1512)에 연결될 수 있다.

[0121] 본 명세서에 설명된 예시적인 방법은 적어도 부분적으로 기계 또는 컴퓨터로 구현될 수 있다. 일부 예는 상기 예에서 설명된 방법을 수행하도록 전자 디바이스를 구성하도록 동작가능한 명령으로 인코딩된 컴퓨터 판독가능한 매체 또는 기계 판독가능한 매체를 포함할 수 있다. 이러한 방법의 구현은 마이크로코드, 어셈블리 언어 코드, 고차 레벨 언어 코드 등과 같은 코드를 포함할 수 있다. 이 코드는 여러 방법을 수행하는 컴퓨터 판독가능한 명령을 포함할 수 있다. 코드는 컴퓨터 프로그램 제품의 일부를 형성할 수 있다. 나아가, 코드는 실행 동안 또는 다른 시간에 하나 이상의 휘발성 또는 비휘발성 컴퓨터 판독가능한 매체에 유형적으로 저장될 수 있다. 이들 컴퓨터 판독가능한 매체는 하드 디스크, 이동식 자기 디스크, 이동식 광 디스크(예를 들어, 콤팩트 디스크 및 디지털 비디오 디스크), 자기 카세트, 메모리 카드 또는 스틱, 랜덤 액세스 메모리(RAM), 판독 전용 메모리(ROM) 등을 포함할 수 있으나 이들로 제한되지 않는다.

[0122] 도 16은 일반적으로 폰 노이만 아키텍처를 가지는 컴퓨터(1600)의 일례를 도시한다. 본 발명의 내용을 판독하고 이해하면 이 기술 분야에 통상의 지식을 가진 자라면 소프트웨어 프로그램이 소프트웨어 프로그램에 한정된 기능을 실행하도록 컴퓨터 기반 시스템에서 컴퓨터 판독가능한 매체로부터 론칭(launched)될 수 있는 방식을 이해될 수 있을 것이다. 이 기술 분야에 통상의 지식을 가진 자라면 여러 프로그래밍 언어들이 본 명세서에 개시된 방법을 구현하고 수행하도록 설계된 하나 이상의 소프트웨어 프로그램을 생성하는데 사용될 수 있다는 것을 이해할 수 있을 것이다. 이 프로그램은 자바(Java), C++ 또는 하나 이상의 다른 언어와 같은 객체 지향 언어(object-oriented language)를 사용하여 객체 지향 포맷으로 구조화될 수 있다. 대안적으로, 프로그램은 어셈블리, C 등과 같은 절차 언어(procedural language)를 사용하여 절차 지향 포맷으로 구조화될 수 있다. 소프트웨어 성분은 원격 절차 호출 등을 포함하는 애플리케이션 프로그램 인터페이스 또는 프로세스간 통신 기술과 같은 이 기술 분야에 통상의 지식을 가진 자에게 잘 알려진 다수의 메커니즘 중 임의의 것을 사용하여 통신할 수 있다. 여러 실시형태의 개시 내용은 임의의 특정 프로그래밍 언어 또는 환경으로 제한되지 않는다.

[0123] 따라서, 다른 실시형태들이 실현될 수 있다. 예를 들어, 컴퓨터, 메모리 시스템, 자기 또는 광 디스크, 일부 다

른 저장 디바이스, 또는 임의의 유형의 전자 디바이스 또는 시스템과 같은 제조 물품은 하나 이상의 프로세서(1602)에 의해 실행될 때 상기 방법에 대하여 전술된 동작 중 어느 것을 수행할 수 있게 하는 명령(1624)(예를 들어, 컴퓨터 프로그램 명령)을 저장한 메모리(예를 들어, 이동식 저장 매체 및 전기, 광, 또는 전자기 전도체를 구비하는 임의의 메모리)와 같은 컴퓨터 판독가능한 매체(1622)에 연결된 하나 이상의 프로세서(1602)를 포함할 수 있다.

[0124] 컴퓨터(1600)는 직접 및/또는 버스(1608)를 사용하여 다수의 성분에 연결된 프로세서(1602)를 가지는 컴퓨터 시스템의 형태를 취할 수 있다. 이 성분은 메인 메모리(1604), 정적 또는 비휘발성 메모리(1606) 및 대용량 저장 매체(1616)를 포함할 수 있다. 프로세서(1602)에 연결된 다른 성분은 비디오 디스플레이와 같은 출력 디바이스(1610), 키보드와 같은 입력 디바이스(1612) 및 마우스와 같은 커서 제어 디바이스(1614)를 포함할 수 있다. 프로세서(1602) 및 다른 성분을 네트워크(1626)에 연결하는 네트워크 인터페이스 디바이스(1620)는 버스(1608)에 더 연결될 수 있다. 명령(1624)은 다수의 잘 알려진 전송 프로토콜(예를 들어, HTTP) 중 어느 하나를 사용하여 네트워크 인터페이스 디바이스(1620)를 통해 네트워크(1626)를 통해 더 송수신될 수 있다. 버스(1608)에 연결된 이들 요소 중 어느 것은 구현될 특정 실시형태에 따라 없을 수도 있거나, 하나만 존재할 수 있거나, 또는 복수개 존재할 수도 있다.

[0125] 일례에서, 프로세서(1602), 메모리(1604, 1606), 또는 저장 디바이스(1616) 중 하나 이상은 실행될 때 컴퓨터(1600)로 하여금 본 명세서에 설명된 방법 중 임의의 하나 이상을 수행할 수 있게 하는 명령(1624)을 각각 포함할 수 있다. 대안적인 실시형태에서, 컴퓨터(1600)는 독립형 디바이스로서 동작하거나 또는 다른 디바이스에 연결(예를 들어, 네트워킹 연결)될 수 있다. 네트워킹 환경에서, 컴퓨터(1600)는 서버-클라이언트 네트워크 환경에서 서버 또는 클라이언트 디바이스의 능력으로 동작하거나 또는 피어 투 피어(또는 분산) 네트워크 환경에서 피어 디바이스로서 동작할 수 있다. 컴퓨터(1600)는 퍼스널 컴퓨터(PC), 태블릿 PC, 셋탑 박스(set-top box: STB), PDA(Personal Digital Assistant), 셀폰(cellular telephone), 웹 기기(web appliance), 네트워크 라우터, 스위치 또는 브리지, 또는 이 디바이스에 의해 취해질 동작을 특정하는 명령 세트(순차 또는 그 밖의 것)를 실행할 수 있는 임의의 디바이스를 포함할 수 있다. 나아가, 단일 컴퓨터(1600)만이 도시되어 있으나, "컴퓨터"라는 용어는 본 명세서에 설명된 방법 중 임의의 하나 이상을 수행하는 명령 세트(또는 다수의 세트)를 개별적으로 또는 공동으로 실행하는 임의의 디바이스 집합을 포함하는 것으로 또한 해석되어야 한다.

[0126] 컴퓨터(1600)는 하나 이상의 통신 프로토콜(예를 들어, USB(universal serial bus), IEEE 1394 등)을 사용하여 주변 디바이스와 통신하는 출력 제어기(1628)를 더 포함할 수 있다. 출력 제어기(1628)는 예를 들어, 컴퓨터(1600)에 통신가능하게 연결된 프로그래밍 디바이스(1630)에 이미지를 제공할 수 있다. 프로그래밍 디바이스(1630)는 병렬 기계(예를 들어, 병렬 기계(100), FSM 엔진(200))를 프로그래밍하도록 구성될 수 있다. 다른 예에서, 프로그래밍 디바이스(1630)는 컴퓨터(1600)와 통합되고 버스(1608)에 연결되거나 또는 네트워크 인터페이스 디바이스(1620) 또는 다른 디바이스를 통해 컴퓨터(1600)와 통신할 수 있다.

[0127] 컴퓨터 판독가능한 매체(1624)가 단일 매체로 도시되어 있으나, "컴퓨터 판독가능한 매체"라는 용어는 하나 이상의 명령(1624) 세트를 저장하는 단일 매체 또는 다수의 매체(예를 들어, 중앙집중된 또는 분산된 데이터베이스, 또는 연관된 캐시 및 서버, 및 또는 여러 유형의 저장 매체, 예를 들어, 프로세서(1602) 레지스터, 메모리(1604, 1606) 및 저장 디바이스(1616))를 포함하는 것으로 해석되어야 한다. "컴퓨터 판독가능한 매체"라는 용어는 컴퓨터에 의해 실행하기 위한 명령 세트를 저장, 인코딩 또는 운반할 수 있고, 컴퓨터로 하여금 본 발명의 방법 중 임의의 하나 이상의 것을 수행하게 하거나 또는 이 명령 세트에 의하여 사용되거나 이와 연관된 데이터 구조를 저장, 인코딩 또는 운반할 수 있는 임의의 매체를 포함하는 것으로 해석되어야 한다. 따라서 "컴퓨터 판독가능한 매체"라는 용어는 솔리드 스테이트 메모리(solid-state memory), 광 매체 및 자기 매체와 같은 유형적인 매체를 포함하나 이로 제한되지 않는 것으로 해석되어야 한다.

[0128] 본 명세서의 특성과 개요를 독자들에게 확인할 수 있게 하는 요약을 요구하는 37 C.F.R. 1.72(b)항에 따라 요약이 제공된다. 이 요약은 청구의 범위 또는 의미를 제한하거나 해석하는데 사용되어서는 안 되는 것으로 이해된다. 이하 청구범위는 상세한 설명에 포함되며, 각 청구항은 별개의 실시형태로서 각자 존재한다.

[0129] 예시적인 실시형태

[0130] 실시형태 1은 명령을 저장한 메모리를 구비하는 컴퓨터를 포함한다. 상기 컴퓨터는 또한 상기 메모리에 통신가능하게 연결된 프로세서를 더 포함하고, 여기서 상기 명령은 프로세서에 의해 실행될 때 프로세서로 하여금, 소스 코드를 상태 및 상기 상태들 사이에 전이를 포함하는 오토마톤으로 변환하도록 하며, 상기 오토마톤에서 상태는 특수 목적 하드웨어 요소에 대응하는 특수 목적 상태를 포함한다. 상기 명령은 또한 상기 프로세서로 하여

금 상기 오토마톤을 네트리스트로 변환하게 하고, 상기 네트리스트를 배치하고 라우팅하여 타깃 디바이스를 구성하기 위한 기계 코드를 제공하게 한다.

[0131] 실시형태 2는 하나 이상의 프로세서를 사용하여, 소스 코드를 구문 트리로 파싱하는 단계를 포함하는 컴퓨터로 구현되는 방법을 포함한다. 본 방법은 하나 이상의 프로세서를 사용하여, 상기 구문 트리를 오토마톤으로 변환하는 단계를 더 포함하며, 여기서 상기 오토마톤은 복수의 상태와 상기 복수의 상태들 사이의 전이를 가지는 거동 모델을 한정하며, 상기 오토마톤 구조는 타깃 하드웨어 디바이스에 의해 지시된다. 본 방법은 하나 이상의 프로세서를 사용하여, 상기 오토마톤을 네트리스트로 변환하는 단계를 더 포함하며, 상기 네트리스트는 복수의 인스턴스를 포함하며, 각 인스턴스는 타깃 디바이스의 하드웨어 요소에 대응한다. 본 방법은 하나 이상의 프로세서를 사용하여, 각 인스턴스를 배치하는 단계를 더 포함하며, 상기 배치하는 단계는 상기 네트리스트의 각 인스턴스를 상기 타깃 디바이스의 하드웨어 요소에 할당하는 단계를 포함한다. 본 방법은 하나 이상의 프로세서를 사용하여, 네트리스트의 함수로서 하드웨어 요소들 사이에 연결을 라우팅하는 단계와, 배치하는 단계와 라우팅하는 단계에 기초하여 타깃 디바이스를 프로그래밍하는데 사용되는 프로그래밍 데이터를 생성하는 단계를 더 포함한다.

[0132] 실시형태 3은 하나 이상의 입력과 하나 이상의 출력을 구비하는 복수의 프로그래밍가능한 요소를 포함하는 프로그래밍가능한 디바이스를 포함한다. 프로그래밍가능한 디바이스는 외부 회로에 대해 복수의 프로그래밍가능한 요소의 일부를 인터페이싱하기 위한 입력 블록과 출력 블록을 더 포함한다. 프로그래밍가능한 디바이스는 복수의 프로그래밍 가능한 요소와 입력 블록과 출력 블록을 통신가능하게 연결하는 복수의 프로그래밍가능한 스위치를 더 포함하며, 여기서 하나 이상의 프로그래밍가능한 스위치를 설정하는 것은 상기 복수의 프로그래밍가능한 요소와 상기 복수의 프로그래밍 가능한 스위치 중 임의의 2개 이상 사이의 신호 라우팅을 선택적으로 제어한다. 프로그래밍가능한 디바이스는 복수의 프로그래밍가능한 요소와 복수의 프로그래밍가능한 스위치를 구성할 수 있는 프로그래밍가능한 데이터를 저장하도록 구성된 복수의 레지스터를 더 포함하며, 여기서 상기 프로그래밍가능한 데이터는 하나 이상의 프로세서를 사용하여, 소스 코드를 구문 트리로 파싱하는 것; 상기 하나 이상의 프로세서를 사용하여, 구문 트리를 오토마톤으로 변환하는 것으로서, 상기 오토마톤은 복수의 상태와 상기 복수의 상태들 사이에 전이를 가지는 거동 모델을 한정하고, 상기 오토마톤 구조는 타깃 하드웨어 디바이스에 의해 지시되는 것인, 변환하는 것; 상기 하나 이상의 프로세서를 사용하여, 상기 오토마톤을 네트리스트로 변환하는 것으로서, 여기서 상기 네트리스트는 복수의 인스턴스를 포함하고, 각 인스턴스는 타깃 디바이스의 하드웨어 요소에 대응하는 것인, 변환하는 것; 상기 하나 이상의 프로세서를 사용하여, 각 인스턴스를 배치하는 것으로서, 상기 배치하는 것은 네트리스트의 각 인스턴스를 타깃 디바이스의 하드웨어 요소에 할당하는 것을 포함하는 것인, 배치하는 것; 상기 하나 이상의 프로세서를 사용하여, 상기 네트리스트의 함수로서 상기 하드웨어 요소들 사이에 연결을 라우팅하는 것; 및 상기 배치하는 것과 라우팅하는 것에 기초하여 상기 타깃 디바이스를 프로그래밍하는데 사용된 프로그래밍 데이터를 생성하는 것에 의해 생성된다.

[0133] 실시형태 4는, 명령을 포함하는 컴퓨터 판독가능한 매체로서, 상기 명령이 하나 이상의 프로세서에 의해 실행될 때 이하 동작, 즉 소스 코드를 구문 트리로 파싱하는 동작; 상기 구문 트리를 오토마톤으로 변환하는 동작으로서, 상기 오토마톤은 복수의 상태와 상기 복수의 상태들 사이에 전이를 가지는 거동 모델을 한정하고, 상기 오토마톤 구조는 타깃 하드웨어 디바이스에 의해 지시되는 것인, 변환하는 동작; 상기 오토마톤을 네트리스트로 변환하는 동작으로서, 상기 네트리스트는 타깃 디바이스와 연관된 복수의 하드웨어 요소를 포함하고, 상기 네트리스트는 하드웨어 요소들 사이에 연결을 한정하는 것인, 변환하는 동작; 각 하드웨어 요소를 배치하는 동작으로서, 상기 배치하는 동작은 네트리스트의 각 하드웨어 요소를 타깃 디바이스 내의 위치로 할당하는 동작을 포함하는 것인, 배치하는 동작; 네트리스트의 함수로서 상기 하드웨어 요소들 사이에 연결을 라우팅하는 동작; 및 상기 배치하는 동작과 라우팅하는 동작을 반영하기 위해 타깃 디바이스를 프로그래밍하는데 사용된 프로그래밍 데이터를 생성하는 동작을 수행하게 한다.

[0134] 실시형태 5는, 하나 이상의 프로세서를 사용하여, 소스 코드를 구문 트리로 파싱하는 단계; 상기 하나 이상의 프로세서를 사용하여, 상기 변환하는 단계를 사용하여 상기 구문 트리를 오토마톤으로 변환하는 단계를 포함하고, 상기 변환하는 단계는 타깃 디바이스에 기초하여 오토마톤 구조를 제한하는 것을 포함하고, 상기 타깃 하드웨어 디바이스는 GOT(group of two)로 쌍으로 형성된 상태 기계 요소를 포함하는 것인 컴퓨터로 구현되는 방법을 포함한다. 본 방법은, 상기 하나 이상의 프로세서를 사용하여, 오토마톤을 네트리스트로 변환하는 단계로서, 상기 네트리스트는 타깃 디바이스와 연관된 복수의 하드웨어 요소를 포함하고, 상기 네트리스트는 상기 하드웨어 요소들 사이에 연결을 한정하는 것인, 변환하는 단계; 상기 하나 이상의 프로세서를 사용하여, 각 하드웨어 요소를 배치하는 단계로서, 상기 배치하는 단계는 네트리스트의 각 하드웨어 요소를 타깃 디바이스 내 위치로

할당하는 단계를 포함하는 것인, 배치하는 단계; 상기 하나 이상의 프로세서를 사용하여, 상기 네트리스트의 함수로서 상기 하드웨어 요소들 사이에 연결을 라우팅하는 단계; 및 상기 배치하는 단계와 라우팅하는 단계를 반영하기 위해 상기 타깃 디바이스를 프로그래밍하는데 사용되는 복수의 비트를 생성하는 단계를 더 포함한다.

[0135] 실시형태 6에서, 실시형태 1 내지 5 중 어느 하나의 주제는, 소스 코드를 변환하는 것이 양화가 특수 목적 하드웨어 요소로 맵핑되는 조건을 충족할 때 양화를 특수 목적 하드웨어 상태를 포함하는 복수의 상태로 변환하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0136] 실시형태 7에서, 실시형태 1 내지 6 중 어느 하나의 주제는, 소스 코드를 변환하는 것이 양화가 특수 목적 하드웨어 요소로 맵핑되는 조건을 충족하지 않을 때 양화를 복수의 일반 목적 상태로 언롤링하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0137] 실시형태 8에서, 실시형태 1 내지 7 중 어느 하나의 주제는, 언롤링하는 것이 오토마톤의 진입 차수를 제어하기 위해 양화를 언롤링하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0138] 실시형태 9에서, 실시형태 1 내지 8 중 어느 하나의 주제는, 오토마톤을 최적화하는 것을 선택적으로 포함하며, 여기서 상기 최적화하는 것은 특정 상태의 예측된 진입 차수가 타깃 디바이스의 제약을 초과할 때 오토마톤의 특정 상태를 다수의 상태로 분할하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0139] 실시형태 10에서, 실시형태 1 내지 9 중 어느 하나의 주제는, 특정 상태를 분할하는 것이 특정 상태의 구동 상태를 다수의 상태로 분배하되 다수의 상태 각각의 진입 차수가 제약을 충족하도록 분배하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0140] 실시형태 11에서, 실시형태 1 내지 10 중 어느 하나의 주제는, 오토마톤을 네트리스트로 변환하는 것이 상태를 네트리스트의 인스턴스로 맵핑하는 것을 포함하고, 맵핑은 특수 목적 상태를 특수 목적 요소에 대응하는 특수 목적 인스턴스로 맵핑하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0141] 실시형태 12에서, 실시형태 1 내지 11 중 어느 하나의 주제는, 오토마톤을 네트리스트로 변환하는 것이 타깃 디바이스의 물리적 설계에 기초하여 상태를 서로 그룹화하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0142] 실시형태 13에서, 실시형태 1 내지 12 중 어느 하나의 주제는, 인스턴스가 SME(state machine element) 하드웨어 요소에 대응하는 상태 기계 요소(SME) 인스턴스와, SME 그룹을 포함하는 하드웨어 요소에 대응하는 SME 그룹 인스턴스를 포함하고, 상기 그룹화하는 것은 상태를 SME 그룹 인스턴스로 그룹화하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0143] 실시형태 14에서, 실시형태 1 내지 13 중 어느 하나의 주제는, 구문 트리를 오토마톤으로 변환하는 것이 소스 코드의 양화를 타깃 디바이스의 카운터 요소에 대응하는 특수 목적 상태를 포함하는 복수의 상태로 변환하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0144] 실시형태 15에서 실시형태 1 내지 14 중 어느 하나의 주제는, SME에 대응하는 복수의 일반 목적 상태가 GOT 하드웨어 요소의 출력 제한에 기초하여 GOT 인스턴스를 형성하도록 서로 그룹화되는 것을 선택적으로 포함할 수 있다.

[0145] 실시형태 16에서, 실시형태 1 내지 15 중 어느 하나의 주제는 오토마톤의 진입 차수를 제한하는 것을 선택적으로 포함하며, 여기서 상기 진입 차수를 제한하는 것은 오토마톤의 상태로 전이하는 개수를 제한하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0146] 실시형태 17에서, 실시형태 1 내지 16 중 어느 하나의 주제는, 진입 차수를 제한하는 것이 특정 상태를 다수의 상태로 분할하고 상기 특정 상태의 구동 상태를 다수의 상태로 분배하되 상기 다수의 상태 각각의 진입 차수가 제약을 충족하도록 분배하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0147] 실시형태 18에서, 실시형태 1 내지 17 중 어느 하나의 주제는, 진입 차수를 제한하는 것이 양화를 복수의 언롤링된 상태로 언롤링하는 것과 상기 언롤링된 상태 중 어느 것에 대해 진입 전이의 개수를 제한하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0148] 실시형태 19에서, 실시형태 1 내지 18 중 어느 하나의 주제는, 변환하는 것이 소스 코드의 양화가 타깃 디바이스의 카운터로 맵핑될 조건을 충족하는지 여부를 결정하는 것과; 상기 양화가 상기 조건을 충족할 때, 상기 양화를 카운터 상태를 포함하는 복수의 상태로 변환하는 것; 및 상기 양화가 상기 조건을 충족하지 않을 때 상기 양화를 언롤링하는 것에 의해 상기 양화를 복수의 SME 상태로 변환하는 것을 포함하는 것을 선택적으로 포함할

수 있다.

[0149] 실시형태 20에서, 실시형태 1 내지 19 중 어느 하나의 주제는, 양화가 조건을 충족하는지 여부를 결정하는 것이 양화를 위한 구동 표현이 양화가 처리되고 있는 동안 매치될 수 있는지 여부를 결정하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0150] 실시형태 21에서, 실시형태 1 내지 20 중 어느 하나의 주제는, 양화가 상기 조건을 충족하는지 여부를 결정하는 것이 상기 양화의 반복된 표현이 상기 양화의 다른 반복된 표현의 접두사인지 여부를 결정하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0151] 실시형태 22에서, 실시형태 1 내지 21 중 어느 하나의 주제는, 상기 양화를 카운터 상태를 포함하는 복수의 상태로 변환하는 것이 상기 복수의 상태를 양화와 카운터 상태의 반복된 표현을 포함하는 루프로 구현하는 것을 포함하고, 상기 카운터 상태는 반복된 표현이 매치되는 횟수를 카운트하도록 구성되며, 상기 카운터 상태는 반복된 표현이 양화에 의해 지정된 횟수에서 매치될 때 다운스트림 상태를 활성화시키는 것을 선택적으로 포함할 수 있다.

[0152] 실시형태 23에서, 실시형태 1 내지 22 중 어느 하나의 주제는, 언롤링하는 것이 타깃 디바이스의 진입 차수 제약에 기초하여 오토마톤의 진입 차수를 제어하도록 양화를 언롤링하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0153] 실시형태 24에서, 실시형태 1 내지 23 중 어느 하나의 주제는, 변환하는 것이 공통 출력을 공유하는 GOT에 기초하여 오토마톤 구조를 제한하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

[0154] 실시형태 25에서, 실시형태 1 내지 24 중 어느 하나의 주제는, 제한하는 것이 타깃 디바이스의 카운터 요소에 기초하여 오토마톤 구조를 제한하는 것을 포함하는 것을 선택적으로 포함할 수 있다.

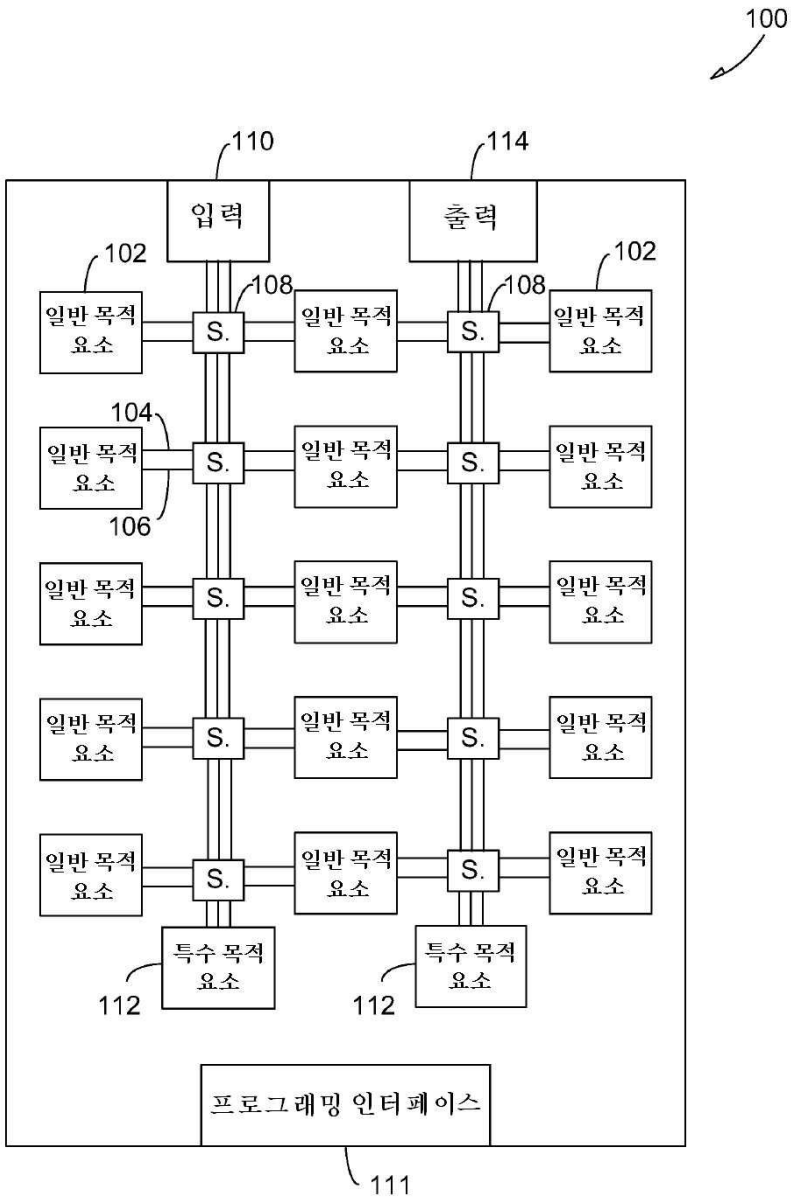
[0155] 실시형태 26에서, 실시형태 1 내지 25 중 어느 하나의 주제는 복수의 비트를 발행하는 것을 선택적으로 포함할 수 있다.

[0156] 실시형태 27에서, 실시형태 1 내지 26 중 어느 하나의 주제는 상기 복수의 상태를 감소시키기 위해 상기 오토마톤을 최적화하는 것을 선택적으로 포함할 수 있다.

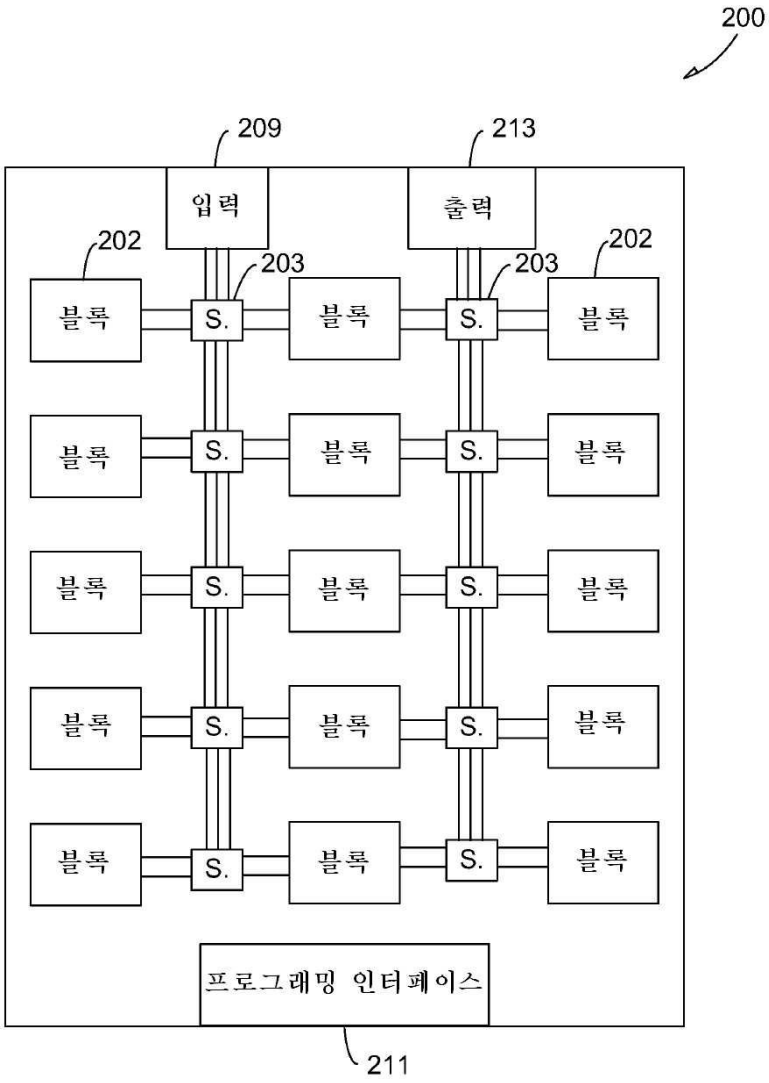
[0157] 실시형태 28에서, 실시형태 1 내지 27 중 어느 것의 주제를 사용하여 생산된 이미지에 의하여 프로그래밍된 병렬 기계를 포함한다.

도면

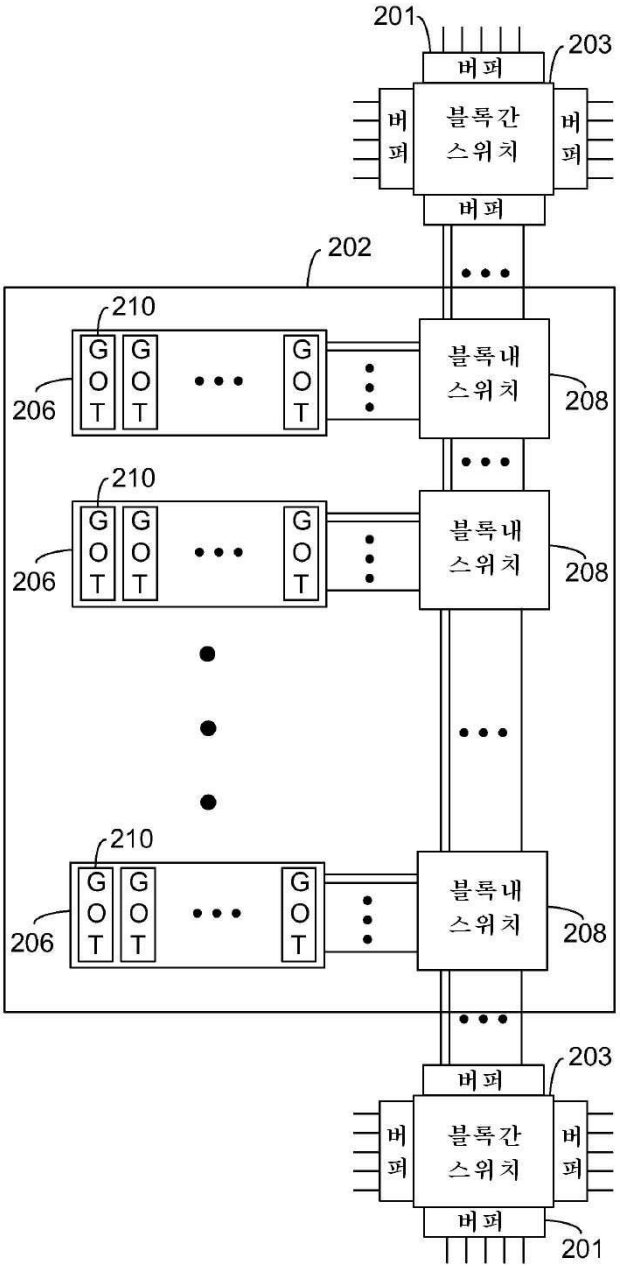
도면1


100
110
114
입력
출력
102
102
108
108
104
106
일반 목적 요소
S.
특수 목적 요소
112
112
프로그래밍 인터페이스
111

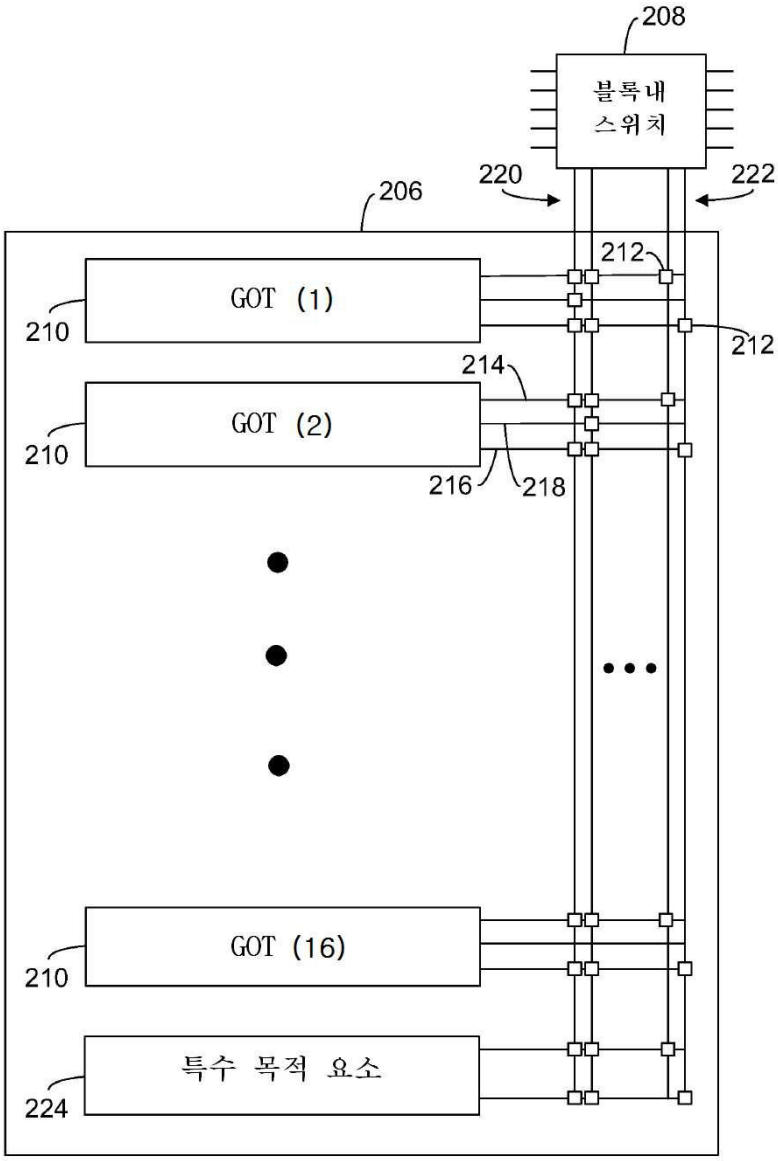
도면2



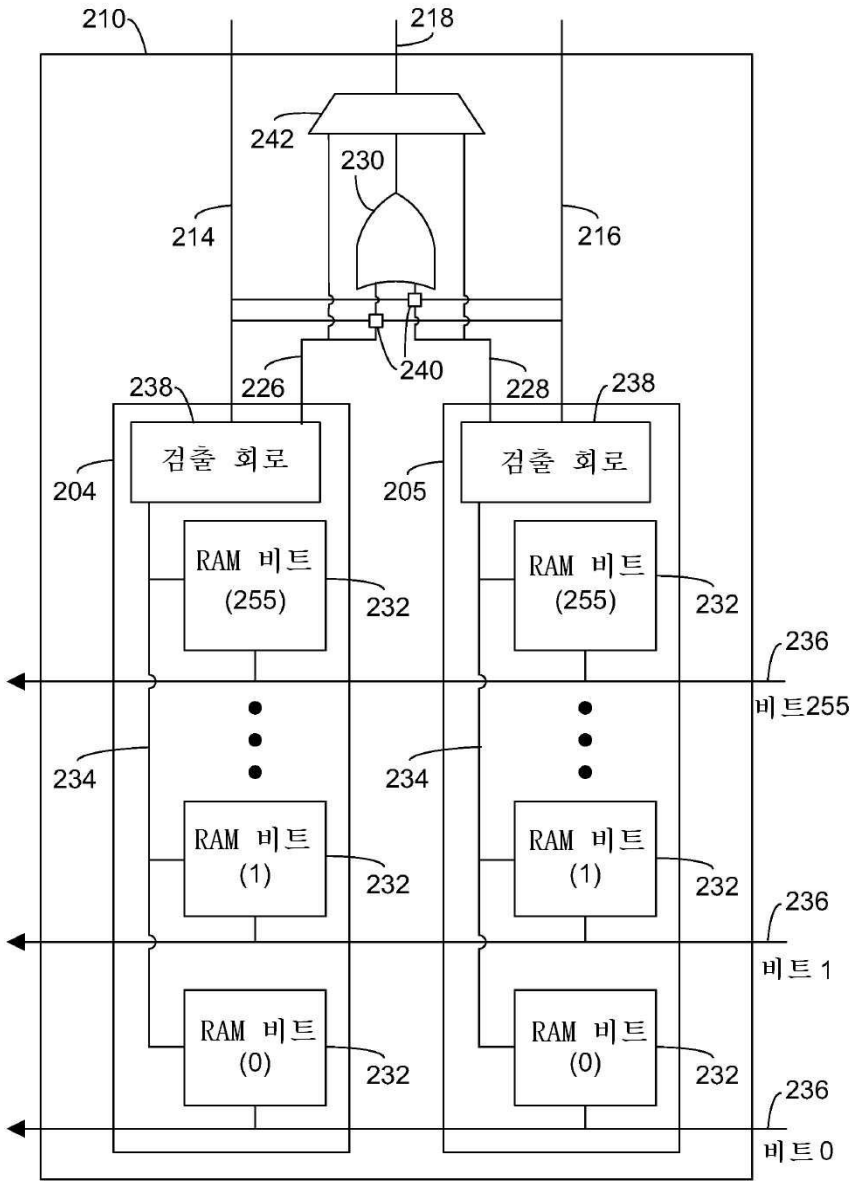
도면3


201
203
버퍼
버퍼
블록간
스위치
버퍼
버퍼
202
210
GOT
206
블록내
스위치
208
210
GOT
206
블록내
스위치
208
210
GOT
206
블록내
스위치
208
203
버퍼
버퍼
블록간
스위치
버퍼
버퍼
201

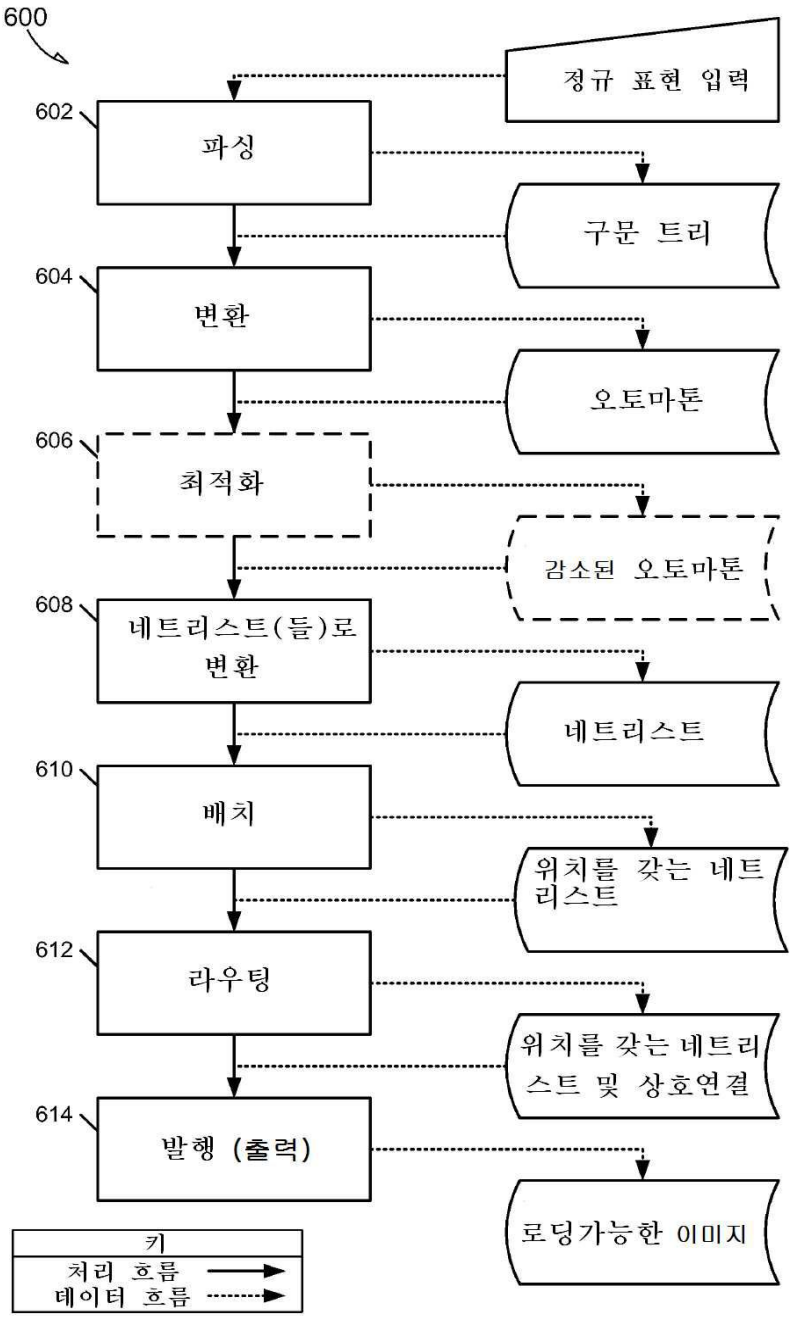
도면4


208
블록내
스위치
220
222
206
212
GOT (1)
210
212
214
GOT (2)
210
216
218
GOT (16)
210
특수 목적 요소
224

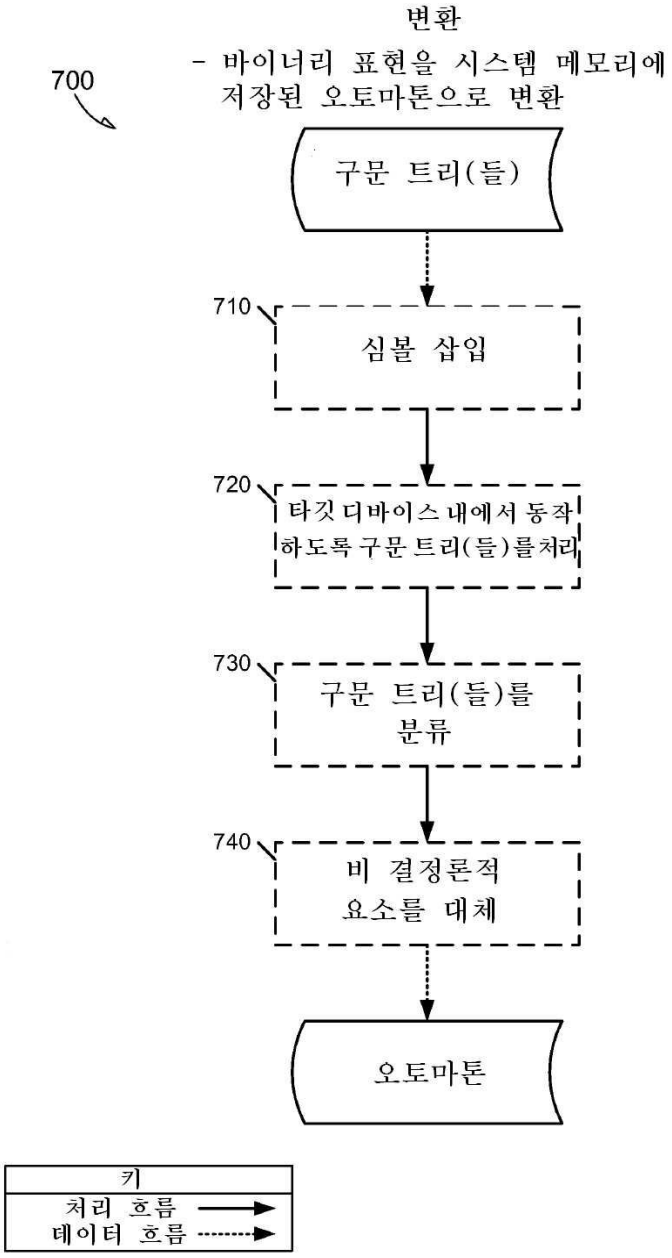
도면5


210
218
242
230
214
216
240
238
226
228
238
검출 회로
검출 회로
204
205
RAM 비트 (255)
232
RAM 비트 (255)
232
236
비트255
234
234
RAM 비트 (1)
232
RAM 비트 (1)
232
236
비트 1
RAM 비트 (0)
232
RAM 비트 (0)
232
236
비트 0

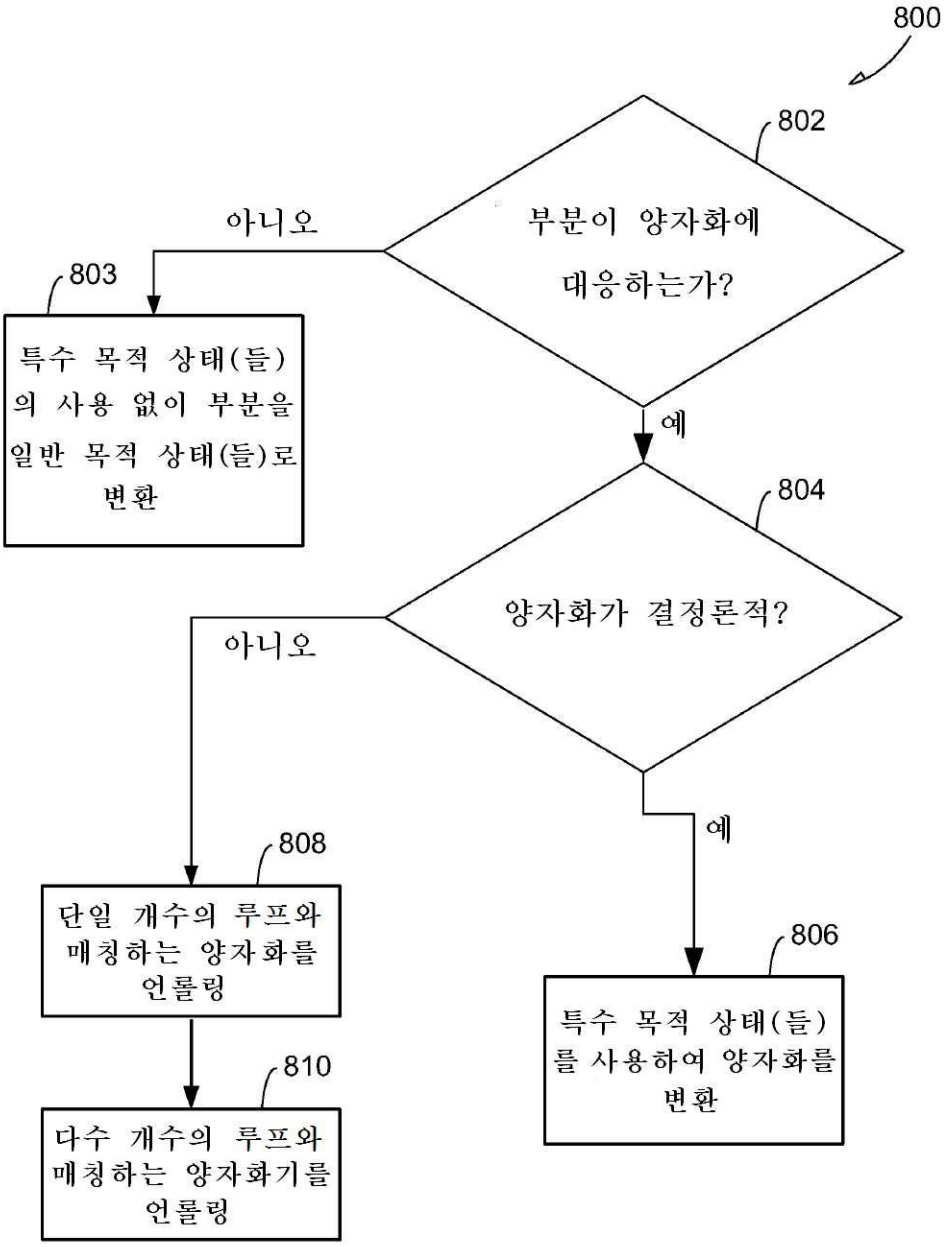
도면6


600
정규 표현 입력
602
파싱
구문 트리
604
변환
오토마톤
606
최적화
감소된 오토마톤
608
네트리스트(들)로 변환
네트리스트
610
배치
위치를 갖는 네트리스트
612
라우팅
위치를 갖는 네트리스트 및 상호연결
614
발행 (출력)
로딩가능한 이미지
키
처리 흐름
데이터 흐름

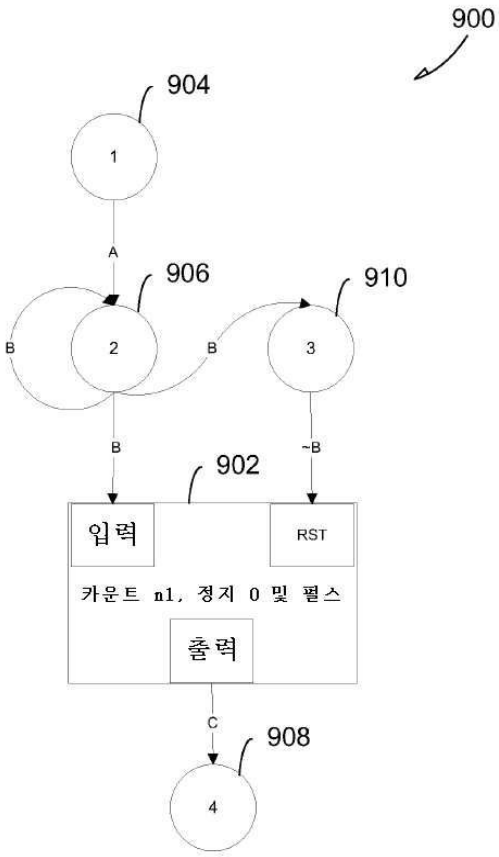
도면7


변환
- 바이너리 표현을 시스템 메모리에 저장된 오토마톤으로 변환
700
구문 트리(들)
710
심볼 삽입
720
타깃 디바이스 내에서 동작하도록 구문 트리(들)를 처리
730
구문 트리(들)를 분류
740
비 결정론적 요소를 대체
오토마톤
키
처리 흐름
데이터 흐름

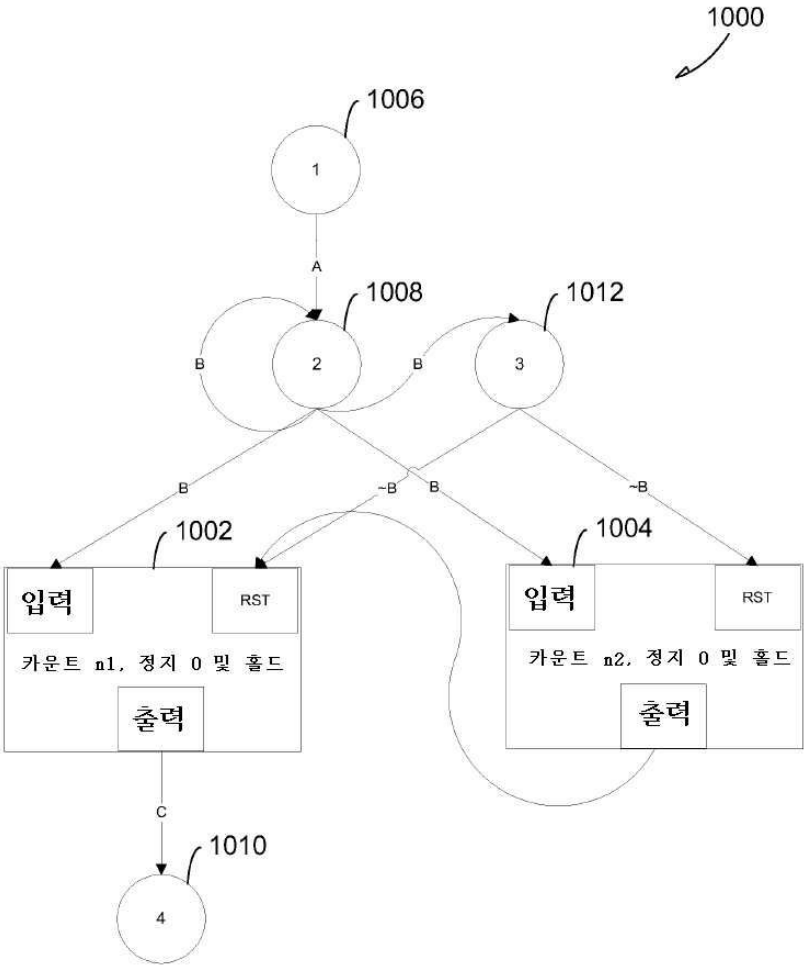
도면8


800
802
부분이 양자화에 대응하는가?
아니오
803
특수 목적 상태(들)의 사용 없이 부분을 일반 목적 상태(들)로 변환
예
804
양자화가 결정론적?
아니오
808
단일 개수의 루프와 매칭하는 양자화를 언롤링
810
다수 개수의 루프와 매칭하는 양자화기를 언롤링
예
806
특수 목적 상태(들)를 사용하여 양자화를 변환

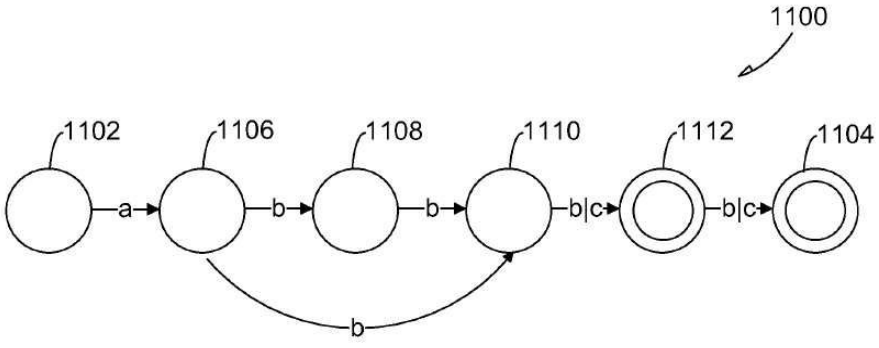
도면9



도면10

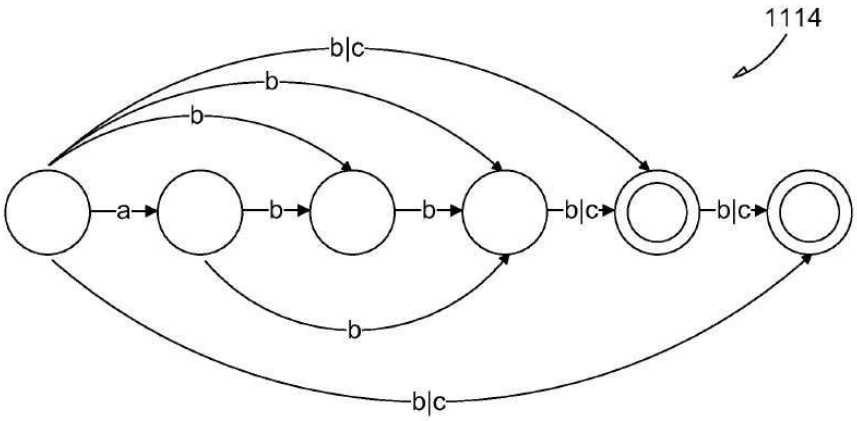


도면11a



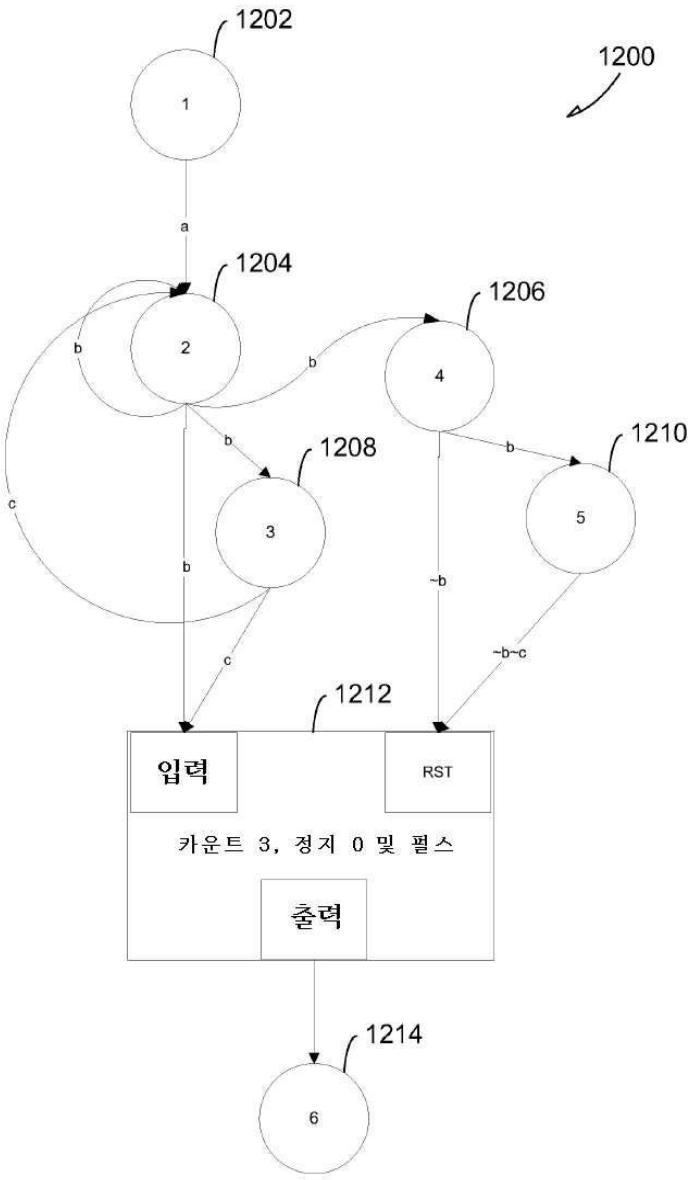
abb?(b|c){1, 2}

도면11b

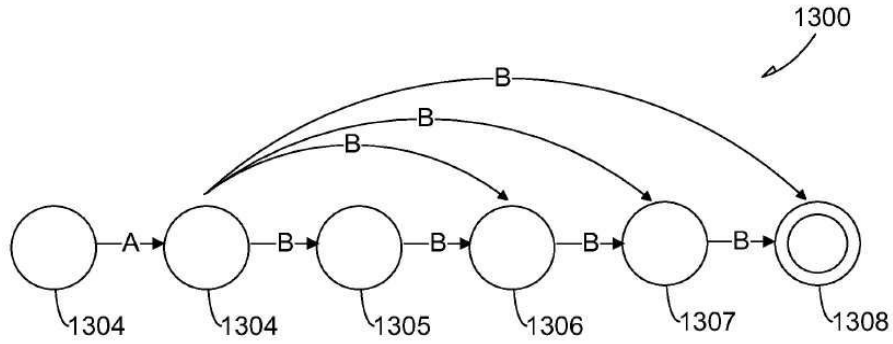


SS(abb?(b|c){1, 2})

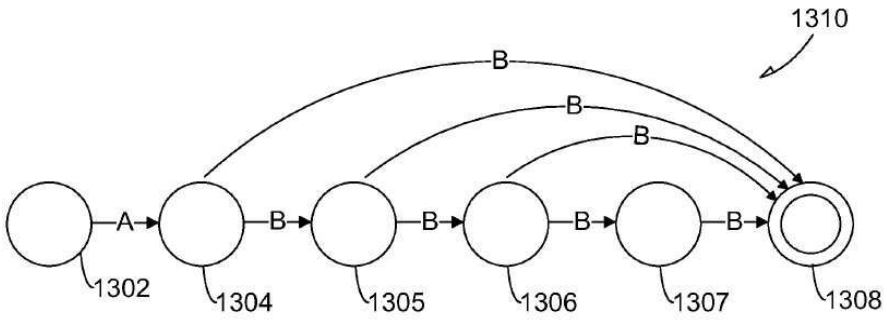
도면12



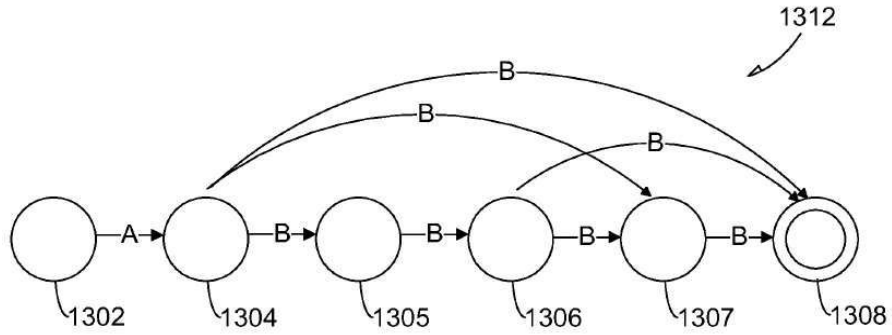
도면13a



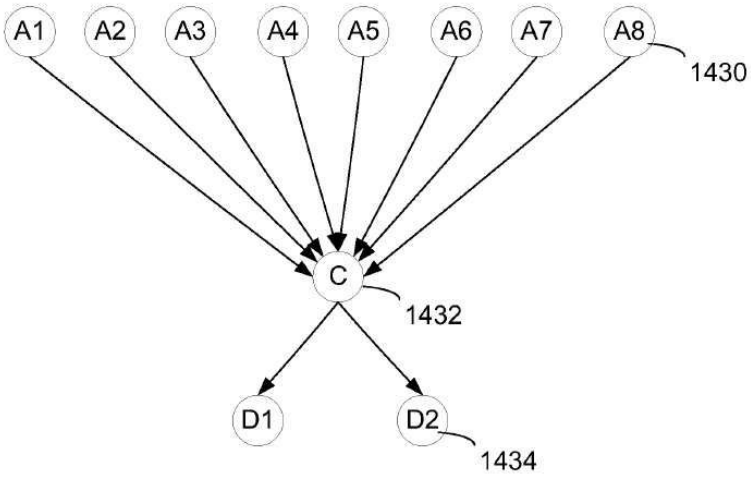
도면13b



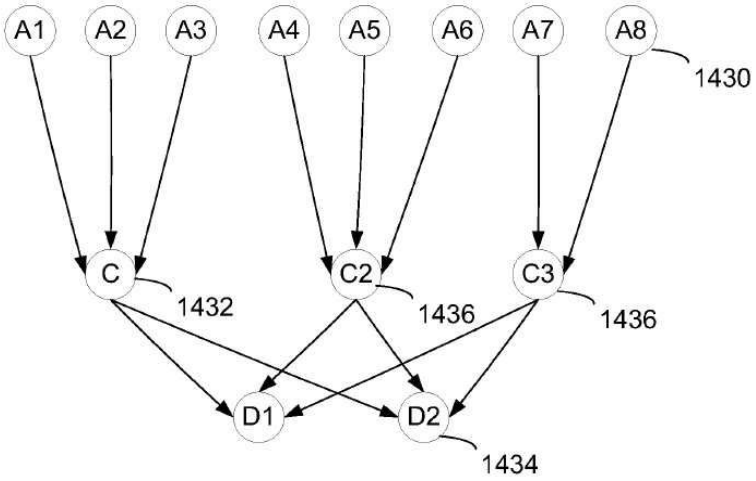
도면13c



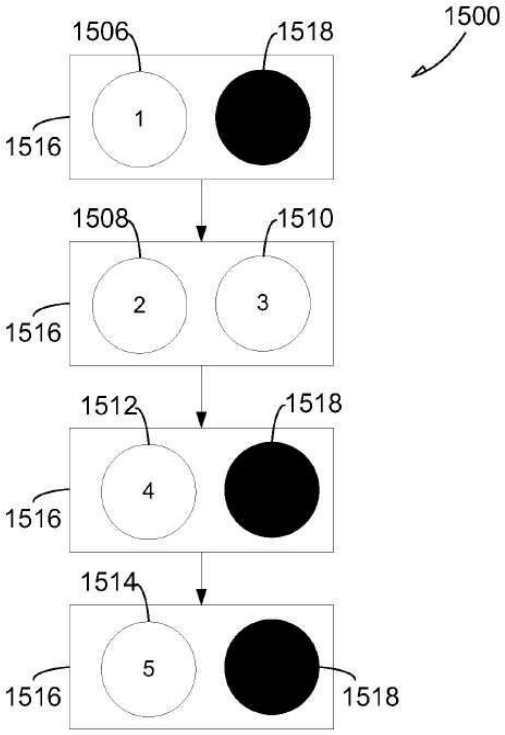
도면14a



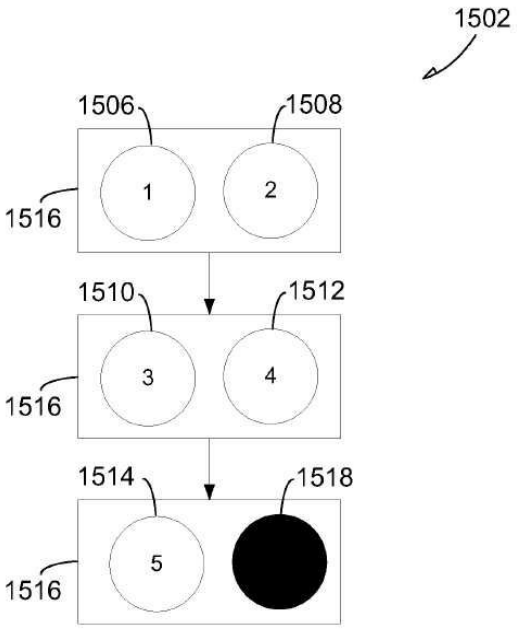
도면14b


A1
A2
A3
A4
A5
A6
A7
A8
1430
C
1432
C2
1436
C3
1436
D1
D2
1434

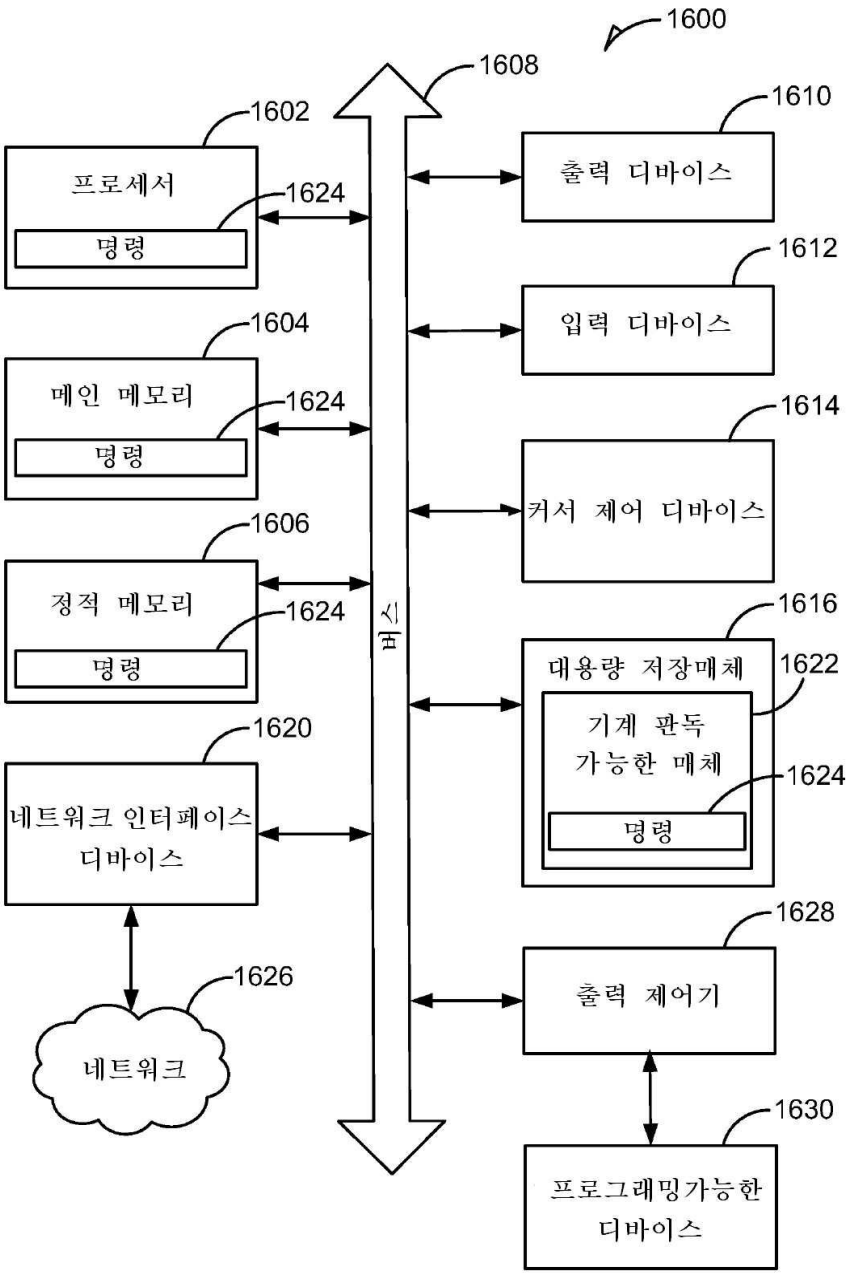
도면15a


1500
1506
1518
1
1516
1508
1510
2
3
1516
1512
1518
4
1516
1514
5
1516
1518

도면15b



도면16


1600
1608
1602
프로세서
명령
1624
1604
메인 메모리
명령
1624
1606
정적 메모리
명령
1624
1620
네트워크 인터페이스 디바이스
1626
네트워크
버스
1610
출력 디바이스
1612
입력 디바이스
1614
커서 제어 디바이스
1616
대용량 저장매체
1622
기계 판독 가능한 매체
명령
1624
1628
출력 제어기
1630
프로그래밍가능한 디바이스