



(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2024년06월25일
(11) 등록번호 10-2678313
(24) 등록일자 2024년06월20일

(51) 국제특허분류(Int. Cl.)
G06F 9/22 (2018.01) G06F 9/28 (2017.01)
G06F 9/30 (2018.01) G06F 9/50 (2018.01)
(52) CPC특허분류
G06F 9/226 (2013.01)
G06F 9/28 (2013.01)
(21) 출원번호 10-2022-0023420
(22) 출원일자 2022년02월23일
심사청구일자 2022년02월23일
(65) 공개번호 10-2023-0126364
(43) 공개일자 2023년08월30일
(56) 선행기술조사문헌
KR1020170016378 A

(73) 특허권자
삼성전자주식회사
경기도 수원시 영통구 삼성로 129 (매탄동)
연세대학교 산학협력단
서울특별시 서대문구 연세로 50 (신촌동, 연세대학교)
(72) 발명자
김한준
서울특별시 서대문구 연세로 50, 연세대학교(신촌동)
김영석
서울특별시 서대문구 연세로 50, 연세대학교(신촌동)
(뒷면에 계속)
(74) 대리인
리앤목특허법인

전체 청구항 수 : 총 18 항

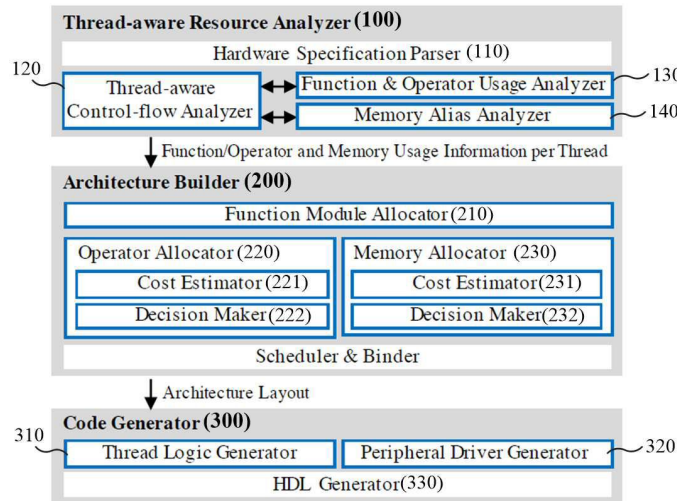
심사관 : 지정훈

(54) 발명의 명칭 하드웨어 최적화를 위한 하이레벨 합성 방법 및 장치

(57) 요약

본 발명은 멀티 스레드 프로그램을 인가받아 다수의 스레드를 인지하고, 인지된 각 스레드가 이용하는 리소스를 분석하는 단계, 분석 결과에 따라 둘 이상의 스레드가 이용하는 리소스를 판별하고, 판별된 리소스를 둘 이상의 스레드 각각의 내부에 포함하여 구성하는 아키텍처와, 스레드와 독립적으로 리소스를 구성하고 둘 이상의 스레드가 공유하도록 연결하는 아키텍처 중 더 작은 크기의 하드웨어로 구현될 수 있는 아키텍처를 선택하여 아키텍처 레이아웃을 구성하는 단계 및 아키텍처 레이아웃을 멀티 스레드 프로그램을 하드웨어 기술 언어로 변환하는 단계를 포함하여 구현되는 하드웨어의 크기를 최적화하고, 성능을 향상시킬 수 있는 하이레벨 합성 방법 및 장치를 제공한다.

대표도 - 도4



(52) CPC특허분류

G06F 9/3005 (2023.08)

G06F 9/30123 (2013.01)

G06F 9/5027 (2013.01)

(72) 발명자

김창수

서울특별시 서대문구 연세로 50, 연세대학교(신촌동)

정신녕

서울특별시 서대문구 연세로 50, 연세대학교(신촌동)

조성준

서울특별시 서대문구 연세로 50, 연세대학교(신촌동)

이용우

서울특별시 서대문구 연세로 50, 연세대학교(신촌동)

송진호

서울특별시 서대문구 연세로 50, 연세대학교(신촌동)

공지예외적용 : 있음

명세서

청구범위

청구항 1

프로세서 및 저장 매체를 포함하는 하이레벨 합성 장치에 의해 수행되는 하이레벨 합성 방법에 있어서,
멀티 스레드 프로그램을 인가받아 다수의 스레드를 인지하고, 인지된 각 스레드가 이용하는 리소스를 분석하는 단계;

분석 결과에 따라 둘 이상의 스레드가 이용하는 리소스를 판별하고, 판별된 리소스를 상기 둘 이상의 스레드 각각의 내부에 포함하여 구성하는 아키텍처와, 스레드와 독립적으로 리소스를 구성하고 상기 둘 이상의 스레드가 공유하도록 연결하는 아키텍처 중 더 작은 크기의 하드웨어로 구현될 수 있는 아키텍처를 선택하여 아키텍처 레이아웃을 구성하는 단계; 및

상기 아키텍처 레이아웃을 상기 멀티 스레드 프로그램을 하드웨어 기술 언어(Hardware Description Language)로 변환하는 단계를 포함하는 하이레벨 합성 방법.

청구항 2

제1항에 있어서, 상기 아키텍처를 선택하는 단계는

상기 리소스에 포함되는 각 함수 모듈을 이용하는 스레드의 수를 판별하고,

함수 모듈을 이용하는 스레드가 다수이면, 각 스레드에 함수 모듈이 포함되는 아키텍처에 따라 함수 모듈이 반복적으로 구현되어 차지하는 크기를 추정하며,

공유하여 이용하는 아키텍처에 따라 스레드와 독립된 하나의 함수 모듈과 이를 공유하여 이용하는 다수의 스레드를 연결하는 아키텍처가 차지하는 크기를 추정하고,

아키텍처에 따라 추정된 크기를 비교하여, 작은 크기를 갖는 아키텍처를 선택하는 하이레벨 합성 방법.

청구항 3

제2항에 있어서, 상기 아키텍처를 선택하는 단계는

함수 모듈을 이용하는 스레드가 하나이면, 함수 모듈을 이용하는 스레드에 포함시켜 상기 아키텍처 레이아웃을 구성하는 하이레벨 합성 방법.

청구항 4

제3항에 있어서, 상기 하이레벨 합성 방법은

상기 아키텍처를 선택하는 단계 이후, 상기 리소스 중 연산자를 구현하기 위한 연산자 할당 방식을 선택하는 단계를 더 포함하고,

상기 연산자 할당 방식을 선택하는 단계는

각 스레드 내에 포함된 각 함수 모듈 내에 할당하는 함수 모듈 개별 할당 방식과

상기 연산자를 각 스레드 내에 단일하게 포함하여 상기 스레드 내의 함수 모듈이 연산자를 공유하여 이용하는 스레드 내 공유 할당 방식, 및

상기 스레드와 독립적으로 구현된 연산자를 다수의 스레드의 함수 모듈이 공유하는 스레드간 공유 할당 방식 각각에 따라 구현되는 하드웨어 크기를 추정하고,

추정된 크기가 가장 작은 할당 방식을 선택하여 상기 아키텍처 레이아웃에 포함하는 하이레벨 합성 방법.

청구항 5

제4항에 있어서, 상기 함수 모듈 개별 할당 방식은

하드웨어로 구현되는 연산자의 크기와 상기 멀티 스레드 프로그램에서 해당 연산자를 이용하는 모든 함수 모듈의 개수의 곱으로 구현되는 하드웨어 크기가 추정되는 하이레벨 합성 방법.

청구항 6

제4항에 있어서, 상기 스레드 내 공유 할당 방식은

상기 연산자의 크기와 해당 연산자를 이용하는 함수 모듈이 포함된 스레드의 개수의 곱과,

각 스레드에서 함수 모듈과 연산자를 연결하는 논리합 게이트의 크기의 합으로 구현되는 하드웨어 크기가 추정되는 하이레벨 합성 방법.

청구항 7

제4항에 있어서, 상기 스레드간 공유 할당 방식은

상기 연산자의 크기와 스레드의 개수에 대응하는 개수의 논리합 게이트의 크기와 다수의 스레드와 상기 연산자를 연결하는 아비터 및 상기 연산자에 대한 경합을 제어하기 위해 각 스레드 내의 함수 모듈 각각에 대응하여 구비되는 경합 관리자의 크기의 합으로 하드웨어 크기가 추정되는 하이레벨 합성 방법.

청구항 8

제1항에 있어서, 상기 리소스를 분석하는 단계는

멀티 스레드 프로그램의 다수의 스레드를 인지하고,

함수 모듈, 연산자 및 메모리 블록을 포함하는 리소스 중 인지된 각 스레드에서 이용되는 리소스를 분석하며,

다수의 스레드 또는 함수 모듈이 이용하는 리소스 중 둘 이상의 스레드 또는 함수가 이용하여 경합이 발생할 수 있는 리소스를 판별하는 하이레벨 합성 방법.

청구항 9

제8항에 있어서, 상기 리소스를 분석하는 단계는

다수의 리소스 중 상기 다수의 스레드 또는 함수 모듈 각각에 의해 이용되는 리소스를 확인하여 리소스 테이블을 생성하는 하이레벨 합성 방법.

청구항 10

프로세서 및 상기 프로세서에 의해 실행되는 하이레벨 합성(High-Level Synthesis: HLS) 프로그램이 저장된 저장 매체를 포함하는 하이레벨 합성 장치에 있어서,

상기 프로세서는 멀티 스레드 프로그램을 인가받아 다수의 스레드를 인지하고, 인지된 각 스레드가 이용하는 리소스를 분석하며,

분석 결과에 따라 둘 이상의 스레드가 이용하는 리소스를 판별하고, 판별된 리소스를 상기 둘 이상의 스레드 각각의 내부에 포함하여 구성하는 아키텍처와, 스레드와 독립적으로 리소스를 구성하고 상기 둘 이상의 스레드가 공유하도록 연결하는 아키텍처 중 더 작은 크기의 하드웨어로 구현될 수 있는 아키텍처를 선택하여 아키텍처 레이아웃을 구성하며,

상기 아키텍처 레이아웃을 상기 멀티 스레드 프로그램을 하드웨어 기술 언어(Hardware Description Language)로 변환하는 하이레벨 합성 장치.

청구항 11

제10항에 있어서, 상기 프로세서는

상기 분석 결과에 따라 상기 리소스에 포함되는 각 함수 모듈을 이용하는 스레드의 수를 판별하고,

함수 모듈을 이용하는 스레드가 다수이면, 각 스레드에 함수 모듈이 포함되는 아키텍처에 따라 함수 모듈이 반

복적으로 구현되어 차지하는 크기를 추정하며,

공유하여 이용하는 아키텍처에 따라 스레드와 독립된 하나의 함수 모듈과 이를 공유하여 이용하는 다수의 스레드를 연결하는 아비터가 차지하는 크기를 추정하고,

아키텍처에 따라 추정된 크기를 비교하여, 작은 크기를 갖는 아키텍처를 선택하는 하이레벨 합성 장치.

청구항 12

제11항에 있어서, 상기 프로세서는

함수 모듈을 이용하는 스레드가 하나이면, 함수 모듈을 이용하는 스레드에 포함시켜 상기 아키텍처 레이아웃을 구성하는 하이레벨 합성 장치.

청구항 13

제12항에 있어서, 상기 프로세서는

상기 아키텍처가 선택되면, 상기 리소스 중 연산자를 구현하기 위한 연산자 할당 방식을 선택하며,

상기 프로세서는

각 스레드 내에 포함된 각 함수 모듈 내에 할당하는 함수 모듈 개별 할당 방식과

상기 연산자를 각 스레드 내에 단일하게 포함하여 상기 스레드 내의 함수 모듈이 연산자를 공유하여 이용하는 스레드 내 공유 할당 방식, 및

상기 스레드와 독립적으로 구현된 연산자를 다수의 스레드의 함수 모듈이 공유하는 스레드간 공유 할당 방식 각각에 따라 구현되는 하드웨어 크기를 추정하고,

추정된 크기가 가장 작은 할당 방식을 선택하여 상기 아키텍처 레이아웃에 포함하는 하이레벨 합성 장치.

청구항 14

제13항에 있어서, 상기 프로세서는

상기 함수 모듈 개별 할당 방식에 따른 하드웨어 크기를 하드웨어로 구현되는 연산자의 크기와 상기 멀티 스레드 프로그램에서 해당 연산자를 이용하는 모든 함수 모듈의 개수의 곱으로 추정하는 하이레벨 합성 장치.

청구항 15

제13항에 있어서, 상기 프로세서는

상기 스레드 내 공유 할당 방식에 따른 하드웨어 크기를

상기 연산자의 크기와 해당 연산자를 이용하는 함수 모듈이 포함된 스레드의 개수의 곱과,

각 스레드에서 함수 모듈과 연산자를 연결하는 논리합 게이트의 크기의 합으로 추정하는 하이레벨 합성 장치.

청구항 16

제13항에 있어서, 상기 프로세서는

상기 스레드간 공유 할당 방식에 따른 하드웨어 크기를

상기 연산자의 크기와 스레드의 개수에 대응하는 개수의 논리합 게이트의 크기와 다수의 스레드와 상기 연산자를 연결하는 아비터 및 상기 연산자에 대한 경합을 제어하기 위해 각 스레드 내의 함수 모듈 각각에 대응하여 구비되는 경합 관리자의 크기의 합으로 추정하는 하이레벨 합성 장치.

청구항 17

제10항에 있어서, 상기 프로세서는

멀티 스레드 프로그램이 인가되면, 상기 멀티 스레드 프로그램의 다수의 스레드를 인지하고,

함수 모듈, 연산자 및 메모리 블록을 포함하는 리소스 중 인지된 각 스레드에서 이용되는 리소스를 분석하며,

다수의 스레드 또는 함수 모듈이 이용하는 리소스 중 둘 이상의 스레드 또는 함수가 이용하여 경합이 발생할 수 있는 리소스를 판별하는 하이레벨 합성 장치.

청구항 18

제17항에 있어서, 상기 프로세서는

다수의 리소스 중 상기 다수의 스레드 또는 함수 모듈 각각에 의해 이용되는 리소스를 확인하여 리소스 테이블을 생성하고, 상기 리소스 테이블을 상기 저장 매체에 저장하는 하이레벨 합성 장치.

발명의 설명

기술 분야

[0001] 본 발명은 하이레벨 합성 방법 및 장치에 관한 것으로, 쓰레드 인지에 기반하여 하드웨어 최적화를 수행할 수 있는 하이레벨 합성 방법 및 장치에 관한 것이다.

배경 기술

[0002] 사물 인터넷(Internet of Things: 이하 IoT)의 적용 분야가 확장됨에 따라 다양한 종류의 임베딩 장치가 개발되고 있다. 최근 임베딩 장치는 네트워크 통신, 센서의 센싱 데이터의 처리 및 액추에이터(actuator) 구동 등과 같은 다양한 동작을 함께 수행할 것이 요구되고 있다. 따라서 임베딩 장치는 서비스 품질(이하 QoS)를 만족하면서 다양한 주변 장치의 이벤트에 대응하여 요구되는 동작을 함께 수행할 수 있어야 한다. 이에 현재 임베딩 장치는 센서나 액추에이터와 같은 주변 기기에 대한 이벤트를 각각 실행하는 다수의 스레드(thread)를 병렬로 처리할 수 있도록 하드웨어 레벨에서 설계되어야 한다.

[0003] 라즈베리 파이(Raspberry Pi)나 아두이노(Arduino)와 같은 범용 하드웨어 랫폼은 임베딩 장치를 구현하는데 매우 편리한 솔루션을 제공할 수 있다. 그러나 범용 하드웨어 플랫폼은 구현된 임베딩 장치에 요구되는 QoS에 따른 전력 효율성이나 성능을 제공하지 못하는 경우가 대부분이다. 즉 임베딩 장치의 하드웨어가 최적화될 수 없다는 한계가 있다.

[0004] 이에 ASIC(Application Specific Integrated Circuit)이나 FPGA(Field Programmable Gate Array)와 같은 맞춤형 하드웨어 플랫폼(custom hardware platform)은 전력 효율성이나 성능에 대한 제약 조건을 충족하는 임베딩 장치를 구현할 수 있도록 하지만, 각 임베딩 장치의 요구 사항에 따라 하드웨어 레벨에서 서로 다르게 설계되어야 하므로, 설계 시간 및 비용이 크게 증가하는 문제가 있다.

[0005] 이에 맞춤형 하드웨어 플랫폼의 설계 비용을 저감시키기 위한 방법으로 하이 레벨 합성(High-Level Synthesis: 이하 HLS) 기법이 제안되어 이용되고 있다. HLS는 C 또는 C++ 등과 같은 하이 레벨 언어(High-Level Language: 이하 HLL)로 작성된 HLL 프로그램을 하드웨어로 변환이 용이한 Verilog와 같은 로우 레벨의 하드웨어 기술 언어(Hardware Description Language)로 변환하여 HDL 프로그램을 획득하는 기법을 나타낸다. HLS는 하이 레벨의 HLL 프로그램을 인가받아, HLL로 작성된 서비스의 알고리즘을 해석하고, 해석된 알고리즘에 따라 서비스를 수행하는 하드웨어를 생성하기 위한 로우 레벨의 HDL 프로그램으로 변환하는 컴파일러로서 기능할 수 있다.

[0006] 따라서 HLS를 사용하는 경우, 개발자는 인간이 용이하게 인식할 수 있는 HLL을 이용하여 임베딩 장치의 하드웨어를 설계할 수 있으므로, 설계 효율성을 크게 높일 수 있다. 그러나 기존의 HLS는 설계되어 구현되는 하드웨어의 크기를 고려하지 않고 고정된 토폴로지를 사용하여 HDL 프로그램을 생성하고, 스레드 레벨에서의 병렬 처리를 지원하지 않는다. 이로 인해 여전히 하드웨어 최적화가 수행되지 않아, 범용 하드웨어 플랫폼을 이용하는 경우에 비해 효율성이 크게 개선되지 않는다는 한계가 있다.

선행기술문헌

특허문헌

[0007] (특허문헌 0001) 한국 공개 특허 제10-2020-0139525호 (2020.12.14 공개)

발명의 내용

해결하려는 과제

- [0008] 본 발명의 목적은 구현되는 하드웨어의 크기를 최적화하여 제조 비용을 저감시킬 수 있는 하이레벨 합성 방법 및 장치를 제공하는데 있다.
- [0009] 본 발명의 다른 목적은 성능을 향상시키면서 전력 소모를 저감할 수 있는 하이레벨 합성 방법 및 장치를 제공하는데 있다.

과제의 해결 수단

- [0010] 상기 목적을 달성하기 위한 본 발명의 일 실시예에 따른 하이레벨 합성 방법은 멀티 스레드 프로그램을 인가받아 다수의 스레드를 인지하고, 인지된 각 스레드가 이용하는 리소스를 분석하는 단계; 분석 결과에 따라 둘 이상의 스레드가 이용하는 리소스를 판별하고, 판별된 리소스를 상기 둘 이상의 스레드 각각의 내부에 포함하여 구성하는 아키텍처와, 스레드와 독립적으로 리소스를 구성하고 상기 둘 이상의 스레드가 공유하도록 연결하는 아키텍처 중 더 작은 크기의 하드웨어로 구현될 수 있는 아키텍처를 선택하여 아키텍처 레이아웃을 구성하는 단계; 및 상기 아키텍처 레이아웃을 상기 멀티 스레드 프로그램을 하드웨어 기술 언어(Hardware Description Language)로 변환하는 단계를 포함한다.
- [0011] 상기 아키텍처를 선택하는 단계는 상기 리소스에 포함되는 각 함수 모듈을 이용하는 스레드의 수를 판별하고, 함수 모듈을 이용하는 스레드가 다수이면, 각 스레드에 함수 모듈이 포함되는 아키텍처에 따라 함수 모듈이 반복적으로 구현되어 차지하는 크기를 추정하며, 공유하여 이용하는 아키텍처에 따라 스레드와 독립된 하나의 함수 모듈과 이를 공유하여 이용하는 다수의 스레드를 연결하는 아비터가 차지하는 크기를 추정하고, 아키텍처에 따라 추정된 크기를 비교하여, 작은 크기를 갖는 아키텍처를 선택할 수 있다.
- [0012] 상기 아키텍처를 선택하는 단계는 함수 모듈을 이용하는 스레드가 하나이면, 함수 모듈을 이용하는 스레드에 포함시켜 상기 아키텍처 레이아웃을 구성할 수 있다.
- [0013] 상기 하이레벨 합성 방법은 상기 아키텍처를 선택하는 단계 이후, 상기 리소스 중 연산자를 구현하기 위한 연산자 할당 방식을 선택하는 단계를 더 포함하고, 상기 연산자 할당 방식을 선택하는 단계는 각 스레드 내에 포함된 각 함수 모듈 내에 할당하는 함수 모듈 개별 할당 방식과 상기 연산자를 각 스레드 내에 단일하게 포함하여 상기 스레드 내의 함수 모듈이 연산자를 공유하여 이용하는 스레드 내 공유 할당 방식, 및 상기 스레드와 독립적으로 구현된 연산자를 다수의 스레드의 함수 모듈이 공유하는 스레드간 공유 할당 방식 각각에 따라 구현되는 하드웨어 크기를 추정하고, 추정된 크기가 가장 작은 할당 방식을 선택하여 상기 아키텍처 레이아웃에 포함할 수 있다.
- [0014] 상기 함수 모듈 개별 할당 방식은 하드웨어로 구현되는 연산자의 크기와 상기 멀티 스레드 프로그램에서 해당 연산자를 이용하는 모든 함수 모듈의 개수의 곱으로 구현되는 하드웨어 크기가 추정될 수 있다.
- [0015] 상기 스레드 내 공유 할당 방식은 상기 연산자의 크기와 해당 연산자를 이용하는 함수 모듈이 포함된 스레드의 개수의 곱과, 각 스레드에서 함수 모듈과 연산자를 연결하는 논리합 게이트의 크기의 합으로 구현되는 하드웨어 크기가 추정될 수 있다.
- [0016] 상기 스레드간 공유 할당 방식은 상기 연산자의 크기와 스레드의 개수에 대응하는 개수의 논리합 게이트의 크기와 다수의 스레드와 상기 연산자를 연결하는 아비터 및 상기 연산자에 대한 경합을 제어하기 위해 각 스레드 내의 함수 모듈 각각에 대응하여 구비되는 경합 관리자의 크기의 합으로 하드웨어 크기가 추정될 수 있다.
- [0017] 상기 리소스를 분석하는 단계는 멀티 스레드 프로그램의 다수의 스레드를 인지하고, 함수 모듈, 연산자 및 메모리 블록을 포함하는 리소스 중 인지된 각 스레드에서 이용되는 리소스를 분석하며, 다수의 스레드 또는 함수 모듈이 이용하는 리소스 중 둘 이상의 스레드 또는 함수가 이용하여 경합이 발생할 수 있는 리소스를 판별할 수 있다.
- [0018] 상기 리소스를 분석하는 단계는 다수의 리소스 중 상기 다수의 스레드 또는 함수 모듈 각각에 의해 이용되는 리소스를 확인하여 리소스 테이블을 생성할 수 있다.
- [0019] 상기 목적을 달성하기 위한 본 발명의 다른 실시예에 따른 하이레벨 합성 컴파일 장치는 프로세서 및 상기 프로세서에 의해 실행되는 하이레벨 합성(High-Level Synthesis: HLS) 프로그램이 저장된 저장 매체를 포함하고, 상기 프로세서는 멀티 스레드 프로그램을 인가받아 다수의 스레드를 인지하고, 인지된 각 스레드가 이용하는 리소

스를 분석하며, 분석 결과에 따라 둘 이상의 스레드가 이용하는 리소스를 판별하고, 판별된 리소스를 상기 둘 이상의 스레드 각각의 내부에 포함하여 구성하는 아키텍처와, 스레드와 독립적으로 리소스를 구성하고 상기 둘 이상의 스레드가 공유하도록 연결하는 아키텍처 중 더 작은 크기의 하드웨어로 구현될 수 있는 아키텍처를 선택하여 아키텍처 레이아웃을 구성하며, 상기 아키텍처 레이아웃을 상기 멀티 스레드 프로그램을 하드웨어 기술 언어로 변환한다.

발명의 효과

[0020] 따라서, 본 발명의 실시예에 따른 하이레벨 합성 방법 및 장치는 멀티 스레드 프로그래밍 기법으로 프로그래밍된 하이레벨 프로그램을 해석하여, 각 스레드가 요구하는 리소스 사이의 관계를 분석하고, 분석된 리소스 관계에 따라 각 스레드가 리소스를 독립적 또는 공통으로 이용하도록 하드웨어를 구성하므로, 성능을 향상시키고, 전력 소모를 저감할 수 있을 뿐만 아니라, 구현되는 하드웨어의 크기를 최적화하여 제조 비용을 저감시킬 수 있다.

도면의 간단한 설명

[0021] 도 1은 임베딩 장치의 동작에 따른 구성의 일 예를 나타낸다.

도 2는 본 발명의 일 실시예에 따른 하이레벨 합성 장치의 개략적 구성을 나타낸다.

도 3은 본 실시예에 따른 하이레벨 합성 장치에서 수행되는 동작에 따른 구성의 일 예를 나타낸다.

도 4는 도 3의 하이레벨 합성 장치의 상세 구성의 일 예를 나타낸다.

도 5는 도 4의 함수 할당기가 스레드에 함수를 할당하는 방식을 설명하기 위한 도면이다.

도 6은 도 4의 연산자 할당기가 연산자를 할당하는 방식을 설명하기 위한 도면이다.

도 7은 도 6의 연산자 할당 방식에 따라 구현되는 하드웨어 크기를 비교한 결과를 나타낸다.

도 8은 본 발명의 일 실시예에 따른 하이레벨 합성 방법을 나타낸다.

발명을 실시하기 위한 구체적인 내용

[0022] 본 발명과 본 발명의 동작상의 이점 및 본 발명의 실시예에 의하여 달성되는 목적을 충분히 이해하기 위해서는 본 발명의 바람직한 실시예를 예시하는 첨부 도면 및 첨부 도면에 기재된 내용을 참조하여야만 한다.

[0023] 이하, 첨부한 도면을 참조하여 본 발명의 바람직한 실시예를 설명함으로써, 본 발명을 상세히 설명한다. 그러나, 본 발명은 여러 가지 상이한 형태로 구현될 수 있으며, 설명하는 실시예에 한정되는 것이 아니다. 그리고, 본 발명을 명확하게 설명하기 위하여 설명과 관계없는 부분은 생략되며, 도면의 동일한 참조부호는 동일한 부재를 나타낸다.

[0024] 명세서 전체에서, 어떤 부분이 어떤 구성요소를 "포함"한다고 할 때, 이는 특별히 반대되는 기재가 없는 한 다른 구성요소를 제외하는 것이 아니라, 다른 구성요소를 더 포함할 수 있는 것을 의미한다. 또한, 명세서에 기재된 "...부", "...기", "모듈", "블록" 등의 용어는 적어도 하나의 기능이나 동작을 처리하는 단위를 의미하며, 이는 하드웨어나 소프트웨어 또는 하드웨어 및 소프트웨어의 결합으로 구현될 수 있다.

[0025] 도 1은 임베딩 장치의 동작에 따른 구성의 일 예를 나타낸다.

[0026] 도 1에서는 임베딩 장치(10)의 일 예로서 웨더보드가 수행하는 동작 따른 구성을 도시하였다. 여기서는 임베딩 장치(10)의 각 동작이 스레드(thread)에 기반하여 수행되는 것으로 가정하며, 이에 임베딩 장치(10)의 구성을 스레드 단위로 구분하여 도시하였다. 비록 도 1에서는 임베딩 장치(10)의 구성을 동작에 기반하여 소프트웨어적인 구분 단위인 스레드별로 구분하여 도시하였으나, 실제 임베딩 장치(10)는 다수의 스레드가 실행되는 하드웨어로 구현된다. 이때 하드웨어는 개별 스레드의 동작을 수행하기 위해 각 스레드에 따라 구분되어 개별적으로 구현되거나, 둘 이상의 스레드를 함께 실행할 수 있도록 구현될 수도 있다. 그리고 하드웨어 구성은 이하에서 설명하는 하이레벨 합성 장치에 의해 결정될 수 있다.

[0027] 도 1을 참조하면, 웨더보드로 이용되는 임베딩 장치(10)는 네트워크 스레드(11), 온도 스레드(12) 및 UV 스레드(13)를 포함할 수 있다. 네트워크 스레드(11)는 포함된 네트워크 드라이버 API를 이용하여 네트워크 모듈(14)을 구동함으로써, 클라우드 서비스를 요청하는 다수의 서비스 장치(Service 1 ~ Service N)와 통신을 수행한다.

네트워크 스레드(11)는 서비스 장치(Service 1 ~ Service N)로부터 데이터 요청(Request Data)을 인가받고, 인가된 데이터 요청(Request Data)에 따라 온도 스레드(12) 및 UV 스레드(13)로 전송 요청 함수(sendRequest)를 호출하여 온도 스레드(12) 및 UV 스레드(13)로 온도 데이터와 UV 데이터를 요청할 수 있다. 그리고 네트워크 스레드(11)는 온도 스레드(12) 및 UV 스레드(13)에서 전송된 온도 또는 UV 데이터를 서비스 장치(Service 1 ~ Service N)로 전송할 수 있다.

- [0028] 온도 스레드(12)는 네트워크 스레드(11)가 전송 요청 함수(sendRequest)를 호출하여 데이터 요청이 인가되면, 온도 드라이버 API를 이용하여 온도 센서(15)를 제어하고, 온도 데이터 획득 함수(getTemp Data)를 이용하여 온도 센서(15)로부터 온도 데이터를 획득한다. 그리고 로우 패스 필터 함수(lowpass Filter)를 이용하여 획득된 온도 데이터를 필터링하고, 화씨 변환 함수(toFahrenheit)를 이용하여 필터링된 온도 데이터를 화씨 데이터로 변환할 수 있다. 변환된 화씨 데이터는 네트워크 스레드(11)로 전달된다.
- [0029] 유사하게 UV 스레드(13)는 네트워크 스레드(11)로부터 데이터 요청이 인가되면, UV 드라이버 API를 이용하여 UV 센서(16)를 제어하고, UV 데이터 획득 함수(getUV Data)를 이용하여, UV 센서(16)로부터 UV 데이터를 획득한다. 그리고 로우 패스 필터 함수(lowpass Filter)를 이용하여 획득된 UV 데이터를 필터링하여, 필터링된 UV 데이터를 네트워크 스레드(11)로 전달할 수 있다.
- [0030] 여기서 네트워크 모듈(14), 온도 센서(15) 및 UV 센서(16)는 임베딩 장치(10)의 주변 기기(Peripheral device)라 할 수 있다.
- [0031] 그리고 각 스레드(11 ~ 13)는 버퍼(Buffer) 또는 큐(Queue)에 데이터 요청(Request Data)와 전송 요청(sendRequest) 및 데이터를 임시 저장하여 전달할 수 있다. 여기서 버퍼와 큐는 하드웨어 메모리 모듈에서 일부 메모리 영역을 할당받아 구현될 수 있다.
- [0032] 즉 임베딩 장치(10)는 주변 기기를 구동하여 요구되는 각종 동작을 수행할 수 있어야 하며, 이때 임베딩 장치(10)가 수행하는 동작은 멀티 스레딩 프로그래밍 기법에 따라 하이레벨 언어로 프로그래밍되어, 발생하는 이벤트에 따라 단독 또는 병렬로 수행되는 다수의 스레드로 표현될 수 있다.
- [0033] 도 2는 본 발명의 일 실시예에 따른 하이레벨 합성 장치의 개략적 구성을 나타낸다.
- [0034] 본 실시예에 따른 하이레벨 합성 장치(20)는 적어도 하나의 프로세서(21), 컴퓨터 판독 가능한 데이터가 저장되는 저장매체(22) 및 통신 버스(24)를 포함한다.
- [0035] 프로세서(21)는 도 2의 장치가 하이레벨 합성 장치(20)로 동작하도록 제어할 수 있다. 예컨대, 프로세서(21)는 저장매체(22)에 저장된 컴퓨터 판독 가능한 하나 이상의 프로그램들(23)을 실행할 수 있다. 하나 이상의 프로그램들(23)은 하나 이상의 컴퓨터 실행 가능 명령어를 포함할 수 있으며, 컴퓨터 실행 가능 명령어는 프로세서(21)에 의해 실행되는 경우 하이레벨 합성 장치(20)로 하여금 예시적인 실시예에 따른 동작들을 수행하도록 구성될 수 있다.
- [0036] 예로서 프로세서(21)는 저장매체(22)에 저장된 HLS 프로그램을 실행하여, HLL 프로그램을 HDL 프로그램으로 변환할 수 있다.
- [0037] 저장매체(22)는 컴퓨터 실행 가능 명령어 내지 프로그램 코드, 프로그램 데이터 및/또는 다른 적합한 형태의 정보를 저장하도록 구성된다. 여기서는 저장매체(22)에 HDL 프로그램으로 변환되어야 하는 HLL 프로그램이 저장될 수 있으며, HLL 프로그램을 HDL 프로그램으로 변환하기 위한 HLS 프로그램이 저장될 수 있다.
- [0038] 저장매체(22)에 저장된 프로그램(23)은 프로세서(21)에 의해 실행 가능한 명령어의 집합을 포함한다. 일 실시예에서, 저장매체(22)는 휘발성 메모리, 비휘발성 메모리 또는 이들의 적절한 조합, 하나 이상의 자기 디스크 저장 디바이스들, 광학 디스크 저장 디바이스들, 플래시 메모리 디바이스들, 그 밖에 하이레벨 합성 장치(20)에 의해 액세스되고 원하는 정보를 저장할 수 있는 다른 형태의 저장매체, 또는 이들의 적합한 조합일 수 있다.
- [0039] 통신 버스(24)는 프로세서(21) 및 저장매체(22)를 포함하여 하이레벨 합성 장치(20)의 다른 다양한 컴포넌트들을 상호 연결한다.
- [0040] 하이레벨 합성 장치(20)는 또한 하나 이상의 입출력 장치를 위한 인터페이스를 제공하는 하나 이상의 인터페이스 모듈(25) 및 하나 이상의 통신 모듈(26)을 포함할 수 있다. 인터페이스 모듈(25) 및 통신 모듈(26)은 통신 버스(24)에 연결된다. 인터페이스 모듈(25)과 통신 모듈(26)은 각각 프로세서(21)의 제어에 따라 각종 주변 기기를 제어하거나 데이터를 전송하고, 주변 기기에서 인가되는 데이터를 프로세서(21)로 전달할 수 있다.

- [0041] 하이레벨 합성 장치(20)는 하드웨어, 펌웨어, 소프트웨어 또는 이들의 조합에 의해 로직회로 내에서 구현될 수 있고, 범용 또는 특정 목적 컴퓨터를 이용하여 구현될 수도 있다. 장치는 고정배선형(Hardwired) 기기, 필드 프로그래밍 가능한 게이트 어레이(Field Programmable Gate Array, FPGA), 주문형 반도체(Application Specific Integrated Circuit, ASIC) 등을 이용하여 구현될 수 있다. 또한, 장치는 하나 이상의 프로세서 및 컨트롤러를 포함한 시스템온칩(System on Chip, SoC)으로 구현될 수 있다.
- [0042] 하이레벨 합성 장치(20)는 하드웨어적 요소가 마련된 컴퓨팅 디바이스 또는 서버에 소프트웨어, 하드웨어, 또는 이들의 조합하는 형태로 탑재될 수 있다. 컴퓨팅 디바이스 또는 서버는 각종 기기 또는 유무선 통신망과 통신을 수행하기 위한 통신 모듈 등의 통신장치, 프로그램을 실행하기 위한 데이터를 저장하는 메모리, 프로그램을 실행하여 연산 및 명령하기 위한 마이크로프로세서 등을 전부 또는 일부 포함한 다양한 장치를 의미할 수 있다.
- [0043] 도 3은 본 실시예에 따른 하이레벨 합성 장치에서 수행되는 동작에 따른 구성의 일 예를 나타내고, 도 4는 도 3의 하이레벨 합성 장치의 상세 구성의 일 예를 나타낸다. 그리고 도 5는 도 4의 함수 할당기가 스레드에 함수를 할당하는 방식을 설명하기 위한 도면이고, 도 6은 도 4의 연산자 할당기가 연산자를 할당하는 방식을 설명하기 위한 도면이며, 도 7은 도 6의 연산자 할당 방식에 따라 구현되는 하드웨어 크기를 비교한 결과를 나타낸다.
- [0044] 도 3 및 도 4의 각 구성은 하나 이상의 프로세서(21)에 의해 실행되는 저장매체(22)에 저장된 컴퓨터 판독 가능한 프로그램(23)의 적어도 하나의 기능이나 동작을 구분한 것으로 볼 수 있으며, 따라서 하이레벨 합성 장치에서 수행되는 동작 단계로 볼 수도 있다.
- [0045] 도 3을 참조하면, 하이레벨 합성 장치(20)는 리소스 분석기(100), 아키텍처 빌더(200) 및 코드 생성기(300)를 포함할 수 있다.
- [0046] 리소스 분석기(100)는 하이레벨 언어로 작성된 멀티 스레드 프로그램을 인가받고, 인가된 멀티 스레드 프로그램의 스레드를 인지 분석하여, 각 스레드가 요구하는 리소스를 확인한다.
- [0047] 여기서 멀티 스레드 프로그램에는 소프트웨어 스레드를 구현되는 임베딩 장치(10)에 이식하여 관리할 수 있도록 이식 가능한 스레드 함수를 호출하기 위한 POSIX(Portable Operating System Interface + X) 스레드 API가 포함될 수 있다.
- [0048] 또한 리소스 분석기(100)는 구현 대상이 되는 임베딩 장치(10)의 주변 기기 등에 대한 하드웨어 환경 정보를 함께 인가받을 수 있다. 즉 도 1에 도시된 바와 같이, 임베딩 장치(10)가 제어하는 통신 모듈(14)이나 온도 센서(15) 및 UV 센서(16)와 하드웨어로 구현되는 메모리의 용량 등 대한 정보를 함께 인가받을 수 있다. 그리고 멀티 스레드 프로그램에는 하드웨어 환경 정보에 따라 주변 기기를 조작하기 위한 주변 기기 드라이버 API가 호출되어 포함될 수 있다. 또한 하드웨어 환경 정보에는 회로가 구현되는 환경 정보가 포함될 수 있다. 예로서 임베딩 장치(10)가 커스텀 반도체 장치로 제조되는 경우, 해당 반도체 장치의 선폭 등과 같은 제조 조건이 하드웨어 환경 정보에 포함될 수 있다.
- [0049] 리소스 분석기(100)는 하드웨어 환경 정보에 따라 공유될 수 있는 리소스에 대한 멀티 스레드 프로그램의 스레드 내 및 스레드 간 경합 발생 가능 여부를 분석한다. 이때 리소스 분석기(100)는 경합 발생 가능성을 판별하기 위해, 리소스 테이블을 생성할 수 있다. 여기서 리소스는 연산자(operator), 함수 모듈(Function module), 메모리 블록 및 주변 기기 등의 하드웨어 리소스 의미한다. 이 중 연산자와 함수 모듈은 하이레벨 합성 장치(20)에 의해 설계되어 임베딩 장치(10)에 하드웨어로 구현되는 리소스이고, 메모리 블록은 임베딩 장치(10)에 포함되는 메모리 공간을 나타낸다.
- [0050] 리소스 분석기(100)는 도 4에 도시된 바와 같이, 하드웨어 파서(110), 제어 흐름 분석기(120), 함수 및 연산자 이용 분석기(130) 및 메모리 앨리어스 분석기(140)를 포함할 수 있다.
- [0051] 하드웨어 파서(110)는 멀티 스레드 프로그램을 분석하여, 멀티 스레드 프로그램의 각 스레드를 인지하고, 인지된 스레드의 리소스 이용 여부에 따라 파싱한다.
- [0052] 제어 흐름 분석기(120)는 멀티 스레드 프로그램에서 스레드를 인지하고, 리소스 이용 여부에 따라 파싱된 각 스레드가 리소스를 이용하는 제어 흐름을 분석하여 다수의 스레드가 동일한 리소스를 이용하고자 하여 경합이 발생할 수 있는지를 확인한다.
- [0053] 제어 흐름 분석기(120)는 다수의 스레드 각각의 함수 또는 연산자 호출과 이에 따라 전달되어야 하는 데이터 경로 및 데이터 경로 상에서 실제 이용되어야 하는 필수 연산자 등을 분석하고, 데이터 경로에 따라 전달되는 데이터를 저장하기 위해 요구되는 메모리 용량 및 용도에 따른 앨리어스 포인터 등을 확인한다. 즉 다수의 스레

드와 이용하고자 하는 리소스 사이의 관계를 분석한다.

[0054] 함수 및 연산자 이용 분석기(130) 및 메모리 앨리어스 분석기(140)는 제어 흐름 분석기(120)가 정확하게 리소스 제어 흐름을 분석할 수 있도록 보조하기 위해, 함수 및 연산자와 메모리 리소스에 대한 이용을 분석한다. 함수 및 연산자 이용 분석기(130)는 멀티 스레드 프로그램의 다수의 스레드 각각에서 호출될 수 있는 함수와 연산자 리소스를 분석한다. 메모리 앨리어스 분석기(140)는 다수의 스레드 각각이 임베딩 장치(20)에 하드웨어로 구현되는 메모리 리소스를 이용하는 앨리어스 포인터로서, 도 1에 도시된 바와 같이 리드 버퍼, 온도 큐 또는 UV 큐 등으로 구분하여 분석한다.

[0055] 리소스 분석기(100)의 제어 흐름 분석기(120)는 스레드 인지 리소스 분석의 결과로 표 1과 같은 리소스 테이블을 생성할 수 있으며, 생성된 리소스 테이블은 도 2에 도시된 저장매체(22)에 저장될 수 있다.

표 1

User	Function				Operator		Memory		
	getUV Data	getTemp Data	send Request	lowPass Filter	Add	Mul	readBuffer	tempReqQ	UVReqQ
sendRequest			-					✓	✓
lowPassFilter				-	✓	✓			
Network Thread			✓				✓		
Temp Thread		✓		✓	✓	✓		✓	
UV Thread	✓			✓	✓	✓			✓

[0056]

[0057] 표 1은 일 예로 리소스 분석기(100)가 도 1에 도시된 임베딩 장치(10)를 구현하기 위해 작성된 멀티 스레드 프로그램의 리소스를 분석한 결과를 나타낸다. 표 1에서 사용자(User)는 함수 또는 연산자를 호출하는 스레드이거나, 데이터를 호출하는 함수이다. 구체적으로 도 1의 임베딩 장치(10)의 구성에 따라 전송 요청 함수(sendRequest), 로우 패스 필터(lowPassFilter), 네트워크 스레드(11), 온도 스레드(12) 및 UV 스레드(13)가 사용자로 포함될 수 있다.

[0058] 그리고 리소스에는 함수, 연산자 및 메모리가 포함된다. 구체적으로 함수에 UV 데이터 획득 함수(getUV Data), 온도 데이터 획득 함수(getTemp Data), 로우패스 필터(lowPassFilter)가 포함되고, 연산자에 가산 연산자(Add)와 곱셈 연산자(Mul)가 포함되며, 메모리에는 리드 버퍼(readBuffer), 온도 요청큐(tempReqQ), UV 요청큐(UV ReqQ)가 포함될 수 있다.

[0059] 도 1에 도시된 바와 같이, 함수는 스레드에 의해 호출되어 이용되는 리소스이면서 동시에 메모리를 이용하는 사용자로도 사용된다.

[0060] 도 1을 참조하여 표 1을 살펴보면, 사용자로서 전송 요청 함수(sendRequest)는 메모리의 온도 요청큐(tempReqQ), UV 요청큐(UV ReqQ)만을 이용하고, 로우 패스 필터(lowPassFilter)는 가산 연산자(Add)와 곱셈 연산자(Mul)를 이용하며, 네트워크 스레드(11)는 전송 요청 함수(sendRequest)와 리드 버퍼(readBuffer)를 이용한다. 그리고 온도 스레드(12)는 온도 데이터 획득 함수(getTemp Data)와 로우 패스 필터(lowPassFilter), 가산 연산자(Add)와 곱셈 연산자(Mul) 및 온도 요청큐(tempReqQ)를 이용하고, UV 스레드(13)는 UV 데이터 획득 함수(getUV Data)와 로우 패스 필터(lowPassFilter), 가산 연산자(Add)와 곱셈 연산자(Mul) 및 UV 요청큐(UV ReqQ)를 이용한다.

[0061] 표 1에서 동일한 열 방향으로 체크 표시 여부를 확인하면, 동일한 리소스를 이용하는 사용자들을 확인할 수 있으며, 이는 해당 리소스에 대해 사용자인 함수 또는 스레드가 경합할 수 있음을 나타낸다.

[0062] 여기서는 설명의 편의를 위하여 리소스 분석기(100)가 멀티 스레드 프로그램을 직접 인가받는 것으로 설명하였으나, 일반적으로 리소스 분석기(100)는 LLVM 컴파일러와 같은 프론트엔드 컴파일러(Front-end Compiler)가 먼저 중간 표현 코드로 번역한 멀티 스레드 프로그램을 인가받아 분석한다. 따라서 도시하지 않았으나, 하이레벨 합성 장치(20)는 멀티 스레드 프로그램의 스레드 프로그램을 인가받아 중간 표현 코드로 번역하여 리소스 분석기(100)로 전달하는 프론트엔드 컴파일러(미도시)를 더 포함할 수 있다.

[0063] 아키텍처 빌더(200)는 리소스 분석기(100)에서 리소스 이용 분석 결과에 따라 멀티 스레드 프로그램에 따른 동작을 수행하는 아키텍처 레이아웃을 설계한다. 아키텍처 빌더(200)는 함수 및 데이터 호출, 데이터 경로 및 리소스의 하드웨어 사양에 따라, 함수 및 연산자를 인스턴스화하고, 이에 대한 배치 및 연결 방법을 결정한다. 또한 아키텍처 빌더(200)는 메모리 리소스를 할당할 수 있으며 리소스를 제어하기 위한 컨트롤러를 생성할 수 있다.

- [0064] 이때 아키텍처 빌더(200)는 리소스 테이블을 이용하여 하드웨어로 구현되는 리소스의 수를 조절하여 아키텍처 레이아웃을 최적화함으로써, 리소스 경합으로 인해 발생하는 지연을 저감시켜 성능을 향상시키고, 전력 소모를 저감시킬 수 있다. 뿐만 아니라, 구현되는 하드웨어의 크기를 최적화할 수 있다.
- [0065] 아키텍처 빌더(200)는 함수 모듈 할당기(210), 연산자 할당기(220) 및 메모리 할당기(230)를 포함할 수 있다.
- [0066] 함수 모듈 할당기(210)는 멀티 스레드 프로그램의 다수의 스레드가 호출하는 함수 리소스를 확인하고, 확인된 함수 리소스에 따라 해당 함수 모듈의 아키텍처를 결정한다. 그리고 결정된 함수 모듈 아키텍처에 따른 연결 관계를 결정한다.
- [0067] 구체적으로 함수 모듈 할당기(210)는 멀티 스레드 프로그램의 다수의 스레드 각각에 함수 모듈이 개별적으로 포함되도록 아키텍처를 결정하거나, 함수 모듈이 스레드와 독립적으로 구성되어 다수의 스레드가 하나의 함수 모듈을 호출하는 방식으로 공유하여 이용하도록 아키텍처를 결정할 수 있다. 특히 본 실시예에서 함수 모듈 할당기(210)는 임베딩 장치(20)의 하드웨어의 크기를 고려하여, 다수의 함수 모듈이 다수의 스레드에 포함되거나, 독립적으로 구현되어 공유되는 두가지 아키텍처가 혼합된 방식을 이용한다.
- [0068] 예로서 도 4에서는 도 1에 도시된 임베딩 장치(10)의 3개의 스레드(11 ~ 13) 각각이 이용하는 함수 모듈에 대한 아키텍처를 나타낸다. 도 4에서 (a)는 모든 스레드의 내부에 자신이 이용하는 함수 모듈이 포함되는 아키텍처를 나타내고, (b)는 모든 함수가 스레드와 독립적으로 별도로 구성되어 다수의 스레드가 각 함수 모듈을 호출하여 접근하는 형태로 이용하는 아키텍처를 나타낸다.
- [0069] (a)의 경우, 각 스레드에 사용하는 함수 모듈이 직접 구현되므로, 경합이 발생하지 않아 성능이 향상될 수 있으나, 동일한 함수 모듈이 여러 번 반복하여 구현되어야 하므로, 하드웨어로 구현되는 임베딩 장치(20)의 크기가 증가되며 전력 효율이 나쁘다는 문제가 있다.
- [0070] 그에 반해 (b)의 경우, 모든 함수 모듈이 1회로 독립적으로 구현되며, 이에 함수를 이용하고자 하는 다수의 스레드는 해당 함수 모듈을 호출할 수 있도록 연결되어야 한다. 이 경우 다수의 스레드가 동일한 함수 모듈을 호출하여 접근할 수 있도록 데이터 경로가 연결되어야 한다. 즉 함수 모듈은 다수의 스레드가 공동으로 이용할 수 있도록, 이용하고자 하는 다수의 스레드와 연결될 수 있어야 한다. 또한 다수의 스레드가 동일한 함수 모듈을 이용하고자 하는 경우, 경합 또는 충돌이 발생할 수 있다. 따라서 (b)와 같은 아키텍처에서는 함수 모듈을 이용하고자 하는 다수의 스레드의 경합을 조절하기 위한 아비터(Arbiter)(Arb)가 더 구현되어야 한다. 아비터(Arb)는 특정 함수 모듈을 이용하고자 하는 스레드가 다수개인 경우, 해당 함수 모듈에 대응하여 구비된다. 따라서 하나의 스레드만이 단독으로 특정 함수 모듈을 이용하는 경우에 해당 함수 모듈에 대응하는 아비터(Arb)는 구비될 필요가 없다. 다만 대응하는 함수 모듈에 연결되어야 하는 스레드의 개수가 증가할 수록 아비터(Arb)의 구조는 복잡해지게 되며, 함수 모듈을 이용하고자 스레드들이 경합하므로, 지연이 발생하여 성능이 저하될 수 있다. 이로 인해, (a)와 같이 개별 스레드에 각 함수 모듈을 직접 포함하는 경우보다, 하나의 함수 모듈을 아비터(Arb)와 함께 이용하는 경우에 실제 하드웨어로 구현되는 임베딩 장치(20)의 크기가 더 커지는 경우도 발생하게 된다. 즉 성능뿐만아니라 크기 측면에서도 효율성도 낮아지는 결과를 초래할 수 있다.
- [0071] 이에 본 실시예의 함수 모듈 할당기(210)는 하드웨어 구현의 크기 효율성 측면에서 구현된 크기가 최소화되도록 아키텍처를 결정한다. 함수 모듈 할당기(210)는 도 4의 (c)에 도시된 바와 같이, 각 함수 모듈이 스레드 내에 포함되거나 스레드와 독립적으로 구분하여 별도로 구현되도록 아키텍처를 결정할 수 있다. 이때 각 함수 모듈의 스레드 내 포함 기준은 다수의 스레드에 의한 공동 이용 여부와 함께 구현 크기이다.
- [0072] 함수 모듈 할당기(210)는 우선 다수의 스레드가 이용하는 함수 모듈 중 공동으로 이용되는 함수 모듈을 판별한다. 만일 특정 함수 모듈을 하나의 스레드만이 이용하는 경우, 해당 함수 모듈은 다른 스레드와 공유될 필요가 없다. 따라서 함수 모듈은 이용하고자 하는 스레드 내에 포함될 수 있다. 이에 도 4의 (c)에서는 단독으로 이용하는 함수 모듈(sendRequest, getUVData, getTempData, toFahrenheit)가 개별 스레드에 포함되었음을 알 수 있다.
- [0073] 그러나 특정 함수 모듈을 다수의 스레드가 이용하고자 하는 경우, 함수 모듈 할당기(210)는 다수의 스레드 각각의 내부에 함수 모듈을 반복하여 구현하는 경우와 독립적인 하나의 함수 모듈과 이에 대응하는 아비터(Arb)를 구현하는 경우 각각의 크기를 예측하고 예측된 크기가 작은 경우를 선택하여, 해당 함수에 대한 아키텍처를 결정한다.
- [0074] 일반적으로 HLS 프로그램은 하드웨어 환경 정보가 획득되면, 각 함수 모듈 또는 연산자 모듈 등이 구현되는 경

우의 크기를 추정할 수 있으며, 아키텍처에 따라 연결 구성이 가변되는 아비터(Arb)의 크기 또한 추정할 수 있다. 이에 하이레벨 합성 장치(20)의 함수 모듈 할당기(210)는 특정 함수 모듈이 각 스레드에 반복하여 구현되는 경우와 아비터(Arb)와 함께 독립적으로 1회 구현되는 경우 사이의 크기를 미리 예측할 수 있다.

[0075] 예로서 도 4의 (c)에서 로우패스 필터 함수(lowPassFilter)는 다른 함수들에 비해 상대적으로 큰 크기를 갖는 함수이며, 온도 스레드(12)와 UV 스레드(13)의 2개의 스레드가 이용하는 함수이다. 이에 함수 모듈 할당기(210)는 임베딩 장치(10)의 크기 효율성을 위해, 로우패스 필터 함수(lowPassFilter)를 온도 스레드(12)와 UV 스레드(13) 각각에 중복되어 구현하지 않고, 아비터(Arb)와 함께 독립적으로 하나만 구현되도록 아키텍처를 결정할 수 있다.

[0076] 연산자 할당기(220)는 리소스 테이블에 기초하여, 각 연산자 리소스에 대한 사용자인 함수 또는 스레드의 이용 여부를 확인하고, 확인된 연산자 리소스에 대한 이용 여부에 따라 구현되는 하드웨어의 크기에 따라 연산자의 아키텍처를 결정한다. 표 1에 나타난 바와 같이, 가산 연산자(Add)와 곱셈 연산자(Mul) 등의 연산자 리소스는 각 스레드에서 이용할 수도 있으나, 로우 패스 필터 함수(lowPassFilter)와 같이 함수 모듈이 이용할 수도 있다. 따라서 연산자 할당기(220)는 도 5에 도시된 바와 같이, 함수 모듈 개별 할당, 또는 스레드 내 공유 할당 및 스레드 간 공유 할당 방식 중에서 할당 방식을 결정한다.

[0077] 도 5의 (a)는 함수 모듈 개별 할당 방식으로 각 스레드의 개별 함수 모듈((Fn 1 ~ Fn F₁), ..., (Fn 1 ~ Fn F_N))이 이용하는 연산자를 모두 개별 함수 모듈 내에 포함시키는 할당 방식을 나타내고, (b)는 스레드 내 공유 할당 방식으로 각 스레드 내에서 이용되는 각 연산자가 하나씩만 포함되도록 하여, 해당 스레드 내의 함수 모듈((Fn 1 ~ Fn F₁), ..., (Fn 1 ~ Fn F_N)) 중 동일한 연산자를 이용하는 함수 모듈이 연산자를 공유하도록 할당하는 방식을 나타낸다. 이때, 공유 연산자를 이용하는 다수의 함수 모듈은 동일 스레드 내에 포함되어 제어 흐름 관점에서 동시에 공유 연산자를 호출하지 않는다. 따라서 다수의 함수 모듈은 논리합 게이트를 이용하여 공유 연산자를 사용할 수 있다.

[0078] 그리고 (c)는 스레드 간 공유 할당 방식으로 각 연산자가 스레드와 독립적인 하드웨어로 하나로 구현되고, 다수의 스레드가 구현된 연산자를 공유하여 사용하도록 하는 할당 방식이다. (c)의 경우, 각각 독립적인 제어 흐름을 갖는 다수의 스레드가 연산자를 공유하므로, 다수의 스레드의 접근을 조절하기 위한 아비터(Arb)가 더 구비되어야 하며, 각 스레드의 함수 모듈((Fn 1 ~ Fn F₁), ..., (Fn 1 ~ Fn F_N)) 또한 다른 스레드의 함수 모듈과의 경합을 방지하기 위해, 경합 관리자(contention manager: CM)이 더 구현되어야 한다. 3가지 할당 방식 각각에 따라 연산자를 하드웨어로 구현하는 경우에 하드웨어의 크기는 각각 수학적 1 내지 3으로 계산될 수 있다.

수학식 1

$$OP \times \sum_1^N F_i$$

수학식 2

$$OP \times N + \sum_1^N OR_{F_i}$$

수학식 3

$$OP + \sum_1^N OR_{F_i} + Arb_N + CM \times \sum_1^N F_i$$

[0082] 여기서 OP는 연산자의 예측 크기, OR은 논리합 게이트의 크기, Arb는 아비터의 크기, CM은 경합 관리자의 크기, F_i는 N개의 스레드 전체에서 연산자를 이용하는 함수의 수를 나타낸다.

- [0083] 즉 하드웨어로 구현하는 경우에 차지하는 크기는 연산자 할당 방식에 따라 수학적 1 내지 수학적 3에 따라 계산된다. 그리고 각 할당 방식에 따른 하드웨어 크기는 연산자 자체의 구현 복잡도에 따른 크기와 스레드 내에서 연산자를 이용하는 함수의 개수($\sum_1^N F_i$)에 따라 서로 상이하게 나타날 수 있다.
- [0084] 예로서 도 7의 (a)에서와 같이 곱셈 연산자(Mul)를 이용하는 함수의 개수($\sum_1^N F_i$)가 4개 또는 8개와 같이 작은 경우, 도 6의 (a)와 같이 다수의 스레드의 다수의 함수 모듈이 하나의 곱셈 연산자(Mul)를 공유하여 이용하는 것이 더 적은 크기로 구현될 수 있어 효율적인 것을 알 수 있다.
- [0085] 그에 반해, 도 7의 (b)와 같이 가산 연산자(Add)는 곱셈 연산자(Mul)에 비해 간단한 구조로 구현될 수 있고, 가산 연산자(Add)를 이용하는 다수의 스레드의 함수 모듈이 24개 또는 32개로 많은 경우, 가산 연산자(Add)는 각 함수 모듈 내에 구현되는 것이 효율적인 것을 알 수 있다.
- [0086] 즉 연산자 할당기(220)는 도 6의 (a) 내지 (c)에 따른 연산자 할당 방식에 따라 구현되는 하드웨어 크기를 수학적 1 내지 수학적 3으로 계산되고, 계산된 하드웨어 크기가 가장 작은 연산자 할당 방식을 선택한다.
- [0087] 메모리 엘리먼트 할당기(220)는 각 스레드가 이용하는 메모리 객체, 즉 저장되어야 하는 데이터의 형식, 크기 및 개수 등에 따라 임베딩 장치(20)의 메모리 모듈 내에서 구분된 메모리 블록들을 엘리먼트 포인트로 지시하여 할당한다.
- [0088] 상기한 각 함수 모듈에 대해 결정된 아키텍처와 연산자 할당 방식은 임베딩 장치(20)를 구현하기 위한 아키텍처 레이아웃으로서 코드 생성기(300)로 전달된다.
- [0089] 코드 생성기(300)는 아키텍처 빌더(200)에 의해 결정된 멀티 스레드 프로그램에 대한 아키텍처 레이아웃이 인가되면, 아키텍처 레이아웃을 하드웨어로 구현 가능하도록 표현한 HDL로 변환하여 출력한다.
- [0090] 코드 생성기(300)는 기존의 하이레벨 합성 장치(20)와 동일한 방식으로 아키텍처 레이아웃을 HDL로 변환하여 출력할 수 있으며, 이에 여기서는 변환 방법에 대해 상세하게 설명하지 않는다. 다만 기존의 하이레벨 합성 장치에서는 코드 생성기가 HDL 생성기만을 구비하는데 반해, 본 실시예의 하이레벨 합성 장치(20)에서는 코드 생성기(300)가 HDL 생성기(330)와 함께 스레드 로직 생성기(310)와 주변 기기 드라이버 생성기(320)를 더 포함할 수 있다.
- [0091] 여기서 스레드 로직 생성기(310)는 멀티 스레드 프로그램의 다수의 스레드 각각에 따른 아키텍처 레이아웃을 구분하여 각 리소스에 대한 하드웨어 로직을 생성하고, 생성된 하드웨어 로직을 데이터 경로에 따라 연결할 수 있다. 스레드 로직 생성기(310)는 다수의 스레드를 관리하기 위한 스레드 관리자를 추가로 생성할 수 있다.
- [0092] 그리고 주변 기기 드라이버 생성기(320)는 멀티 스레드 프로그램에 포함된 주변 기기 드라이버 API를 확인하고, 확인된 주변 기기 드라이버 API에 따라 미리 설정된 하드웨어 함수 로직을 획득한다. 주변 기기 드라이버 API의 경우, 이를 구현하기 위한 하드웨어 함수 로직이 미리 제공될 수 있다. 이에 주변 기기 드라이버 생성기(320)는 주변 기기 드라이버 API에 따른 하드웨어 함수 로직을 획득함으로써, HDL 생성기(330)가 처리해야 하는 작업을 경감시킬 수 있다.
- [0093] HDL 생성기(330)는 스레드 로직 생성기(310)와 주변 기기 드라이버 생성기(320)에서 생성된 하드웨어 로직을 인가받고, 스레드와 함수에 대한 하드웨어 로직을 결합하고, HDL로 변환한다.
- [0094] 결과적으로 본 실시예에 따른 하이레벨 합성 장치(20)는 하이레벨 언어로 작성된 멀티 스레드 프로그램을 인가받고, 인가된 멀티 스레드 프로그램에서 다수의 스레드가 이용하는 다수의 함수 모듈 각각이 각 스레드의 내부에 구현되는 아키텍처와 스레드의 외부에 독립적으로 구현되어 공유되는 아키텍처를 구분하여, 구현될 하드웨어 크기를 미리 예측한다. 그리고 더 적은 하드웨어 크기로 구현되는 아키텍처를 선택하여 결정한다.
- [0095] 또한 다수의 스레드에 포함된 함수 모듈들이 이용하는 연산자가 각 함수 모듈 내부에 구현되도록 할당하는 함수 모듈 개별 할당 방식과 동일 스레드 내의 함수들이 동일한 연산자를 공유하여 이용하도록 개별 스레드 내에 연산자를 하나씩 할당하는 스레드 내 공유 할당 방식 및 스레드와 별도로 독립적으로 연산자를 구현하여 다수의 스레드의 함수가 모두 공유하여 이용하는 스레드 간 공유 할당 방식 중 가장 작은 하드웨어 크기로 구현되는 할당 방식을 선택한다.
- [0096] 그리고 결정된 아키텍처와 선택된 할당 방식에 따라 구성되는 아키텍처 레이아웃을 HDL로 변환함으로써, 크기가 최소화된 임베딩 장치(20)가 하드웨어로 구현될 수 있도록 한다. 이때 아키텍처 결정 과정에서, 각 스레드가

독립적으로 또는 공통으로 함수 모듈을 이용하는지 여부에 따라 각 함수가 스레드 내에 포함되거나 독립적으로 구현되므로, 불필요한 데이터 경로 형성을 억제하고 아비터(Arb)의 사용을 줄여 성능이 개선되고 전력 소모가 저감될 수 있다.

[0097] 도 8은 본 발명의 일 실시예에 따른 하이레벨 합성 방법을 나타낸다.

[0098] 도 2 내지 도 7을 참조하면, 하이레벨 합성 방법은 우선 하드웨어 임베딩 장치로 구현되어야 하는 멀티 스레드 프로그램을 획득한다(S10). 여기서 멀티 스레드 프로그램에는 스레드 단위 동작을 수행할 수 있도록 하는 스레드 API와 주변 기기 구동을 위한 주변 기기 드라이버 API가 함께 포함될 수 있으며, 하드웨어 환경 정보가 함께 획득될 수 있다.

[0099] 멀티 스레드 프로그램이 획득되면, 획득된 멀티 스레드 프로그램에서 다수의 스레드 각각을 인지하고, 인지된 스레드 각각이 이용하는 리소스와 스레드의 제어 흐름을 분석하여 동일 리소스에 대한 경합 발생 가능성을 확인한다(S20).

[0100] 이때 분석 결과에 따라 각 스레드와 스레드에 포함된 함수 모듈의 리소스이용 여부를 정리하여 표 1과 같은 리소스 테이블을 생성할 수 있다.

[0101] 그리고 리소스 분석 결과에 따라 다수의 스레드 중 적어도 하나의 스레드에서 이용되는 함수 모듈 각각에 대한 아키텍처를 결정한다(S30). 이때, 함수 모듈 각각에 대한 아키텍처는 특정 함수 모듈을 이용하는 스레드가 하나인 경우, 해당 함수 모듈이 스레드 내에 포함되도록 결정된다. 그러나 특정 함수 모듈을 이용하는 스레드가 다수개인 경우에는 해당 함수 모듈이 다수의 스레드 각각에 포함되는 경우와 스레드와 별도로 하나의 함수 모듈과 이를 스레드들과 연결하기 위한 아비터(Arb)가 형성되는 경우 사이에 예측된 하드웨어 구현 크기 작은 경우를 아키텍처로 선택한다.

[0102] 한편, 함수 모듈에 대한 아키텍처가 결정되면, 다수의 함수 모듈이 이용하는 각 연산자를 할당한다(S40). 여기서 다수의 연산자 각각은 함수 모듈 개별 할당, 또는 스레드 내 공유 할당 및 스레드 간 공유 할당 방식 중에서 하드웨어로 구현 시에 작은 크기로 구현되도록 하나의 방식이 선택되어 할당될 수 있다. 함수 모듈 개별 할당 방식은 각 스레드의 개별 함수 모듈 각각이 연산자를 포함하는 할당 방식이고, 스레드 내 공유 할당 방식은 각 스레드 내에서 이용되는 각 연산자가 하나씩만 포함되도록 하여, 해당 스레드 내의 함수 모듈이 연산자를 공유하여 이용하도록 하는 할당 방식이며, 스레드 간 공유 할당 방식은 각 연산자가 스레드와 독립적인 하드웨어로 하나로 구현되고, 다수의 스레드가 구현된 연산자를 공유하여 사용하도록 하는 할당 방식이다.

[0103] 그리고 각 스레드가 이용하는 데이터의 형식, 크기 및 개수 등에 따라 하드웨어 환경 정보로 획득된 크기의 메모리에서 구분된 메모리 블록들을 할당한다(S50).

[0104] 이후 각 함수 모듈에 대해 결정된 아키텍처와 연산자 할당 방식에 따라 아키텍처 레이아웃이 결정되면, 결정된 아키텍처 레이아웃을 지정된 방식에 따라 하드웨어로 구현이 용이한 HDL로 변환한다(S60).

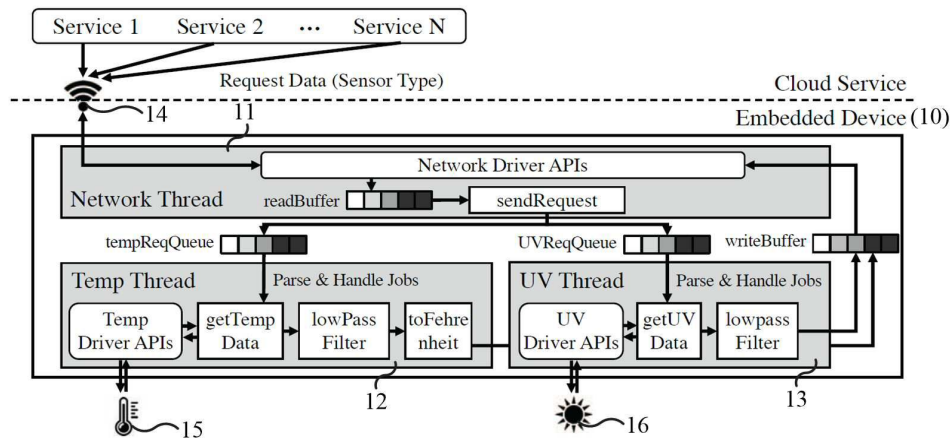
[0105] 본 발명에 따른 방법은 컴퓨터에서 실행시키기 위한 매체에 저장된 컴퓨터 프로그램으로 구현될 수 있다. 여기서 컴퓨터 판독가능 매체는 컴퓨터에 의해 액세스될 수 있는 임의의 가용 매체일 수 있고, 또한 컴퓨터 저장 매체를 모두 포함할 수 있다. 컴퓨터 저장 매체는 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 또는 기타 데이터와 같은 정보의 저장을 위한 임의의 방법 또는 기술로 구현된 휘발성 및 비휘발성, 분리형 및 비분리형 매체를 모두 포함하며, ROM(판독 전용 메모리), RAM(랜덤 액세스 메모리), CD(컴팩트 디스크)-ROM, DVD(디지털 비디오 디스크)-ROM, 자기 테이프, 플로피 디스크, 광데이터 저장장치 등을 포함할 수 있다.

[0106] 본 발명은 도면에 도시된 실시예를 참고로 설명되었으나 이는 예시적인 것에 불과하며, 본 기술 분야의 통상의 지식을 가진 자라면 이로부터 다양한 변형 및 균등한 타 실시예가 가능하다는 점을 이해할 것이다.

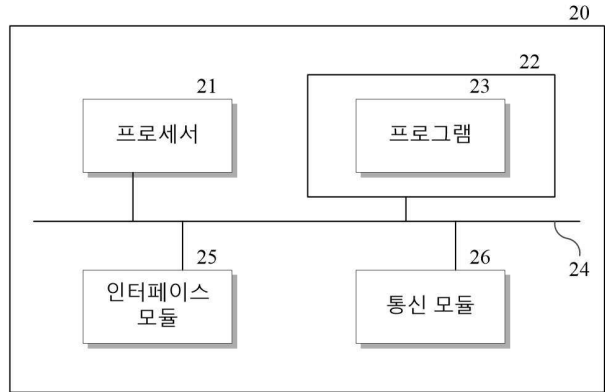
[0107] 따라서, 본 발명의 진정한 기술적 보호 범위는 첨부된 청구범위의 기술적 사상에 의해 정해져야 할 것이다.

도면

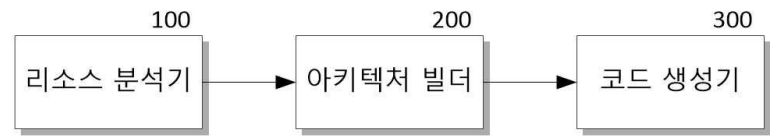
도면1



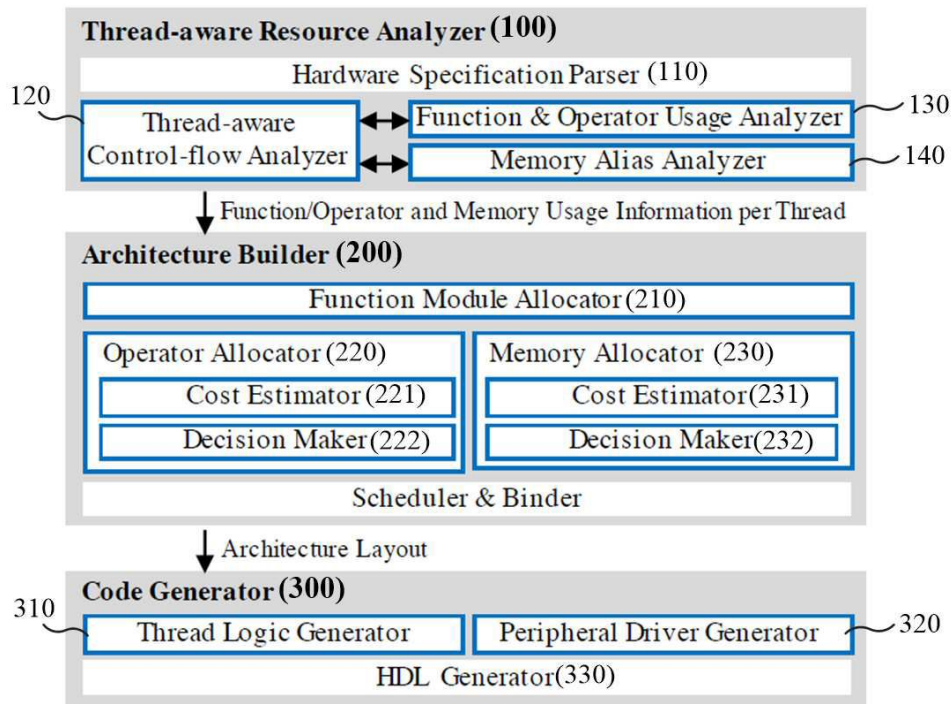
도면2



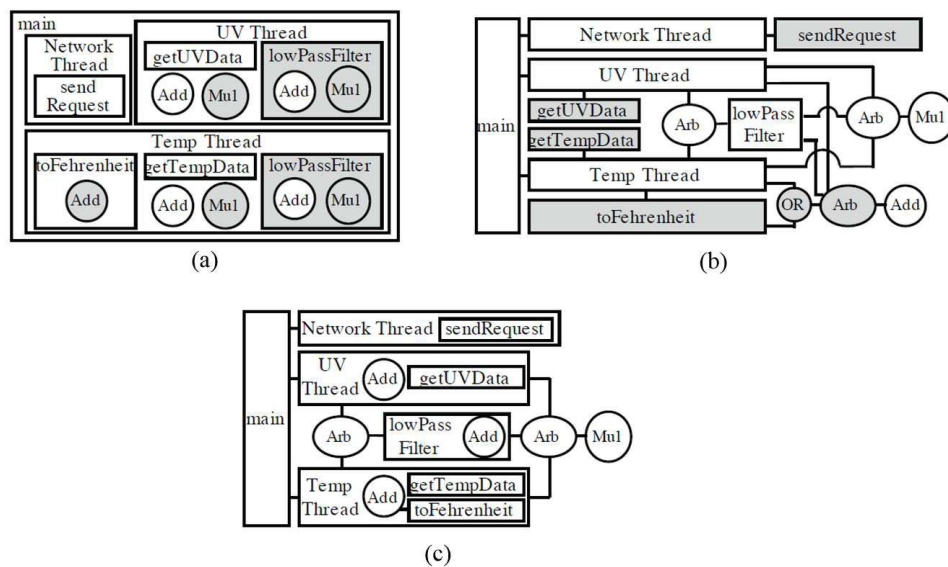
도면3



도면4

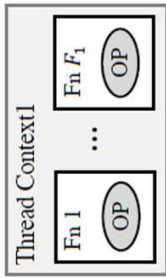


도면5



도면6

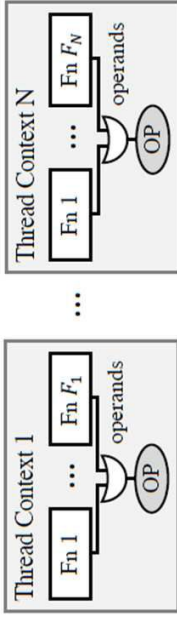
Policy 1: Private



$$\text{Cost: } OP \times \sum_1^N F_i$$

(a)

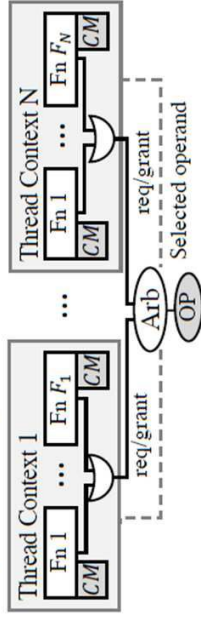
Policy 2: Sharing within thread



$$\text{Cost: } OP \times N + \sum_1^N OR_{F_i}$$

(b)

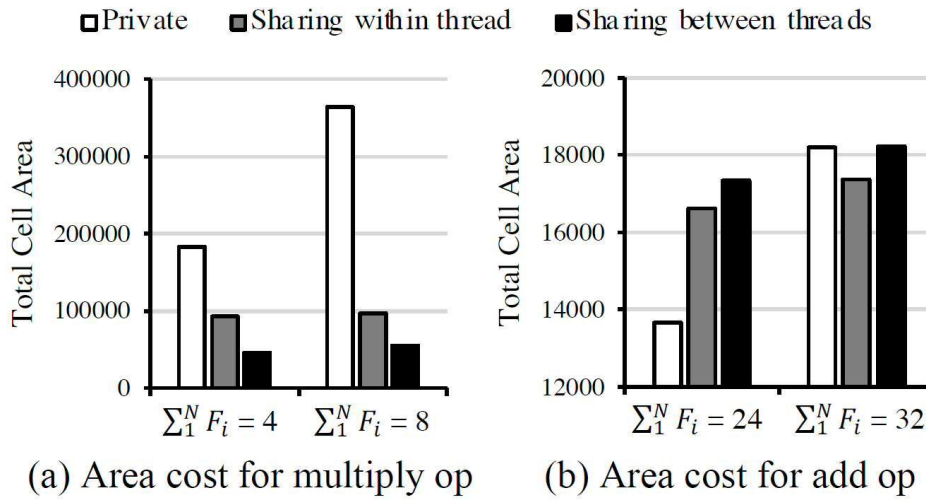
Policy 3: Sharing between threads



$$\text{Cost: } OP + \sum_1^N OR_{F_i} + Arb_N + CM \times \sum_1^N F_i$$

(c)

도면7



도면8

