뉴로모픽(Neuromorphic) 프로세서

5대 분야 Autonomous Driving • Function Processing • 기술분야명 지능형 반도체

Processing

담당 센터 SoC플랫폼 • 연구자 황태호

개념

인간 두뇌의 신경 동작 방식(SNN, Spiking Neural Network)을 모사하여 초저전력, 병렬/분산 컴퓨팅이 가능하도록 설계하는 반도체 SoC 기술

개발 내용

기술내용

SNN기반 지능 모방형 컴퓨팅 모델링으로 SNP(Spiking Neuron Processor) 및 스마트 디바이스 응용 SoC 개발
- 인간 생체의 신호전달체계를 아날로그 동작기반의 Analog Neuron Cell(ANC)로 모델링
- 저전력·소형화를 위한 디지털 방식의 Digital Neuron Logic H/W 설계로 프로세서(SNP) 및 스마트 디바이스용 응용 SoC 개발

지도 및 비지도 학습 알고리즘 개발 및 SNP용 통합 플랫폼
- FPGA 기반 SNP 학습알고리즘 H/W 및 S/W 검증
- 컴퓨터 비전 라이브러리 제작, IoT 응용 어플리케이션 및 DB 구축, 스마트 디바이스용 API 및 디바이스 시제품 등

차별성

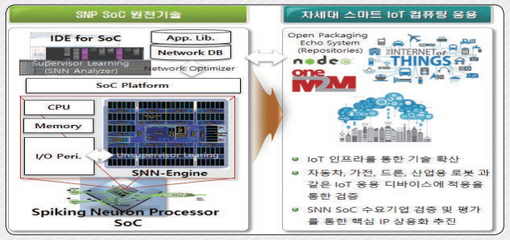
본 기술은 IBM社*와 달리 Cortex M0에서 M4F의 정도 플랫폼을 중심으로 모바일 스마트 디바이스에 적합한 경량화 SNN 기술

* IBM社가 개발한 SNN 기반의 딥러닝처리 기술은 서버기반의 SNN 기술로서 경량화 보다는 처리방식에 집중하여 대규모 데이터셋의 효율적인 학습을 장점으로 왓슨에 적용

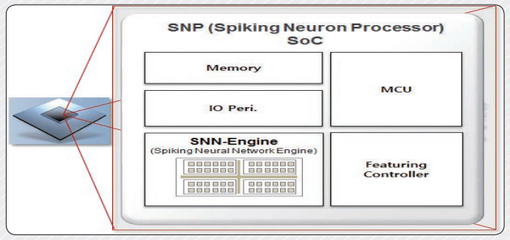
해외주요기관

IBM社(Watson/SNN), Intel社(curie/RBF)

개발 내용

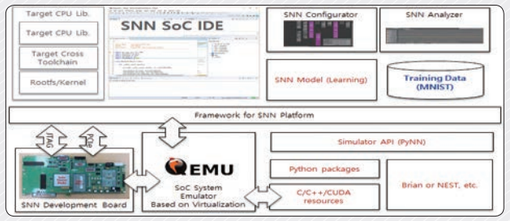


지능 모방형 컴퓨팅 기술 개발

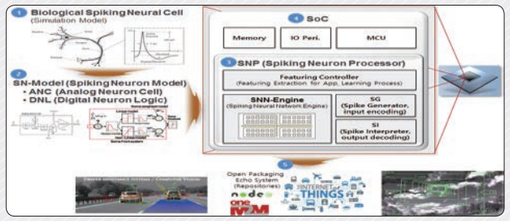


SNP SoC 구조의 개요

연구원 보유(개발) 핵심기술



SNN을 위한 통합 플랫폼



SNN을 위한 기본 엔진 기술

연구원 보유(개발) 핵심기술

KETI 핵심기술

능동적 딥러닝 처리를 위한 SNN 기반 1,024개의 Spiking Neural 급 기술(20nW/cell) 및 SoC 기반 플랫폼 기술 개발 중(~'21년)

* 셀당 20nW 급, 최대 1024개의 Spiking Neural 안정성 및 구동 특성 확보

8bit-, 32-bit 프로세서 설계 및 한국형 CPU 코어(MENSA) 설계 기술

* 인텔 8051호환의 자체 코어 개발과 RISC 방식 및 캐쉬 지원 구조 개량을 통한 저전력의 SoC IP, 자체 8/32-bit RISC 프로세서 코어(MENSA-core) 보유

차별성

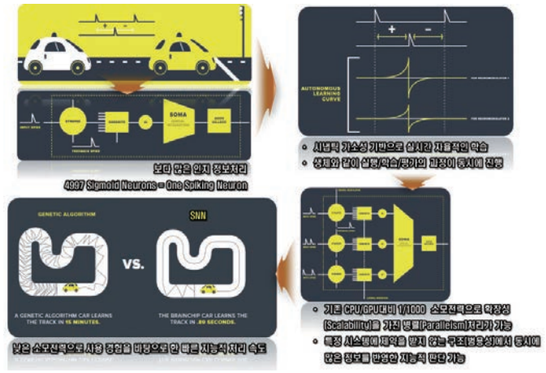
기존 선도 기업들의 SNN 관련기술보다 다수의 뉴런 모델(1,024개 수준)로 구성된 SoC를 개발 중으로 IoT분야에 성능 우위 및 가격 경쟁력 확보 가능

관련기술 보유 IP

그래픽 처리 장치의 동작을 위한 작업 할당 시스템 및 방법(국내/등록/2016)
객체 검출 장치, 차량용 객체 검출 장치 및 이들의 방법(국내/등록/2015)
CPU와 GPU 간의 협업 시스템 및 그 방법(PCT/출원/2014)
다수의 버스를 이용하는 논리연산 제어장치(PCT/출원/2013)

Business Model

- 자율주행, IoT 디바이스의 비전처리 분야
- 웨어러블 디바이스(슈즈, 밴드, 워치 등)의 동작관리 분야
- 수요 예상 기업
  - 의료기기, 소비재 가전제조사, 자동차부품 제조사



관련 연구 분야

Core Technology

뉴로모픽(Neuromorphic) 프로세서

융합/Soc플랫폼 황태호

Related Technology

고효율 반도체
Processing in Memory 설계 및 병렬처리 라이브러리
융합시스템/SoC플랫폼 김동순

ECU 플랫폼
멀티 도메인(V2X, IVN, 공조 등) 지원 가상 ECU 플랫폼
융합시스템/임베디드·SW 민수영

신호처리 SoC
다중 TOF신호 융합 및 프로세싱
정보통신미디어/지능형영상처리 황영배

(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2014년09월19일
(11) 등록번호 10-1442643
(24) 등록일자 2014년09월15일

(51) 국제특허분류(Int. Cl.)
G06F 13/14 (2006.01) *G06F 12/02* (2006.01)
(21) 출원번호 10-2013-0048061
(22) 출원일자 2013년04월30일
심사청구일자 2013년04월30일
(56) 선행기술조사문헌
JP2011175624 A*
KR1020040106472 A*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
전자부품연구원
경기도 성남시 분당구 새나리로 25 (야탑동)
(72) 발명자
황태호
서울 송파구 문정로 83, 128동 402호 (문정동, 문정래미안아파트)
김동순
경기 성남시 분당구 정자일로 248, 606동 3203호 (정자동, 파크뷰)
(74) 대리인
특허법인지명

전체 청구항 수 : 총 13 항 심사관 : 김세영

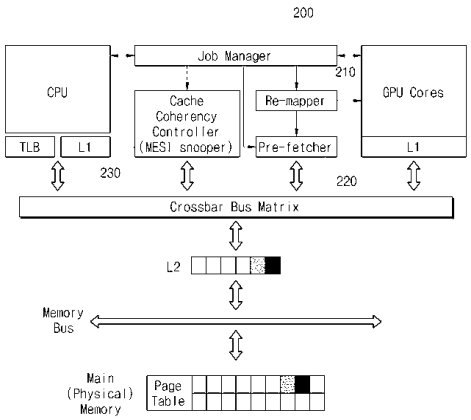
(54) 발명의 명칭 **CPU와 GPU 간의 협업 시스템 및 그 방법**

(57) 요 약

본 발명은 CPU와 GPU 간의 효율적인 협업 구조에 관한 것으로서, GPU를 제어하는 별도의 유닛을 통해 CPU의 로드를 경감시키고 GPU에 작업을 할당함에 있어서 직접적인 데이터의 복사가 없이 작업에 사용할 데이터의 주소 영역에 대한 정보만 제공되도록 함으로써, CPU와 GPU 간의 협업의 효율성을 높인 CPU와 GPU 간의 협업 시스템 및 그 방법을 제공한다.

또한 CPU와 GPU 간의 캐시일관성 유지를 위해 종래의 멀티 CPU 간의 캐시일관성 유지에 사용되는 프로토콜을 확장한 프로토콜을 제공하여 CPU와 GPU 간의 캐시 불일치 해소에 적합한 캐시일관성 유지 방법을 제공한다.

대 표 도 - 도2



이 발명을 지원한 국가연구개발사업
과제고유번호 10041664
부처명 지식경제부
연구관리전문기관 한국산업기술평가원
연구사업명 산업융합원천기술개발사업
연구과제명 멀티 Shader GPU 통합형 멀티 코어 퓨전 프로세서 원천 기술 개발
기 여 율 1/1
주관기관 전자부품연구원
연구기간 2012.06.01 ~ 2013.05.31

특허청구의 범위

청구항 1

CPU와 GPU 간의 협업 시스템에 있어서,

상기 CPU가 요청하는 작업을 전달받아 상기 GPU에 요청하며, 상기 GPU가 처리한 작업 결과를 상기 CPU로 전달하는 작업관리부; 및

상기 GPU의 주소 공간과 메인 메모리의 주소 공간의 매핑을 보조하는 주소매핑부를 포함하되,

상기 작업관리부는

상기 CPU가 요청하는 작업에 해당하는 코드 정보 및 상기 작업을 수행하기 위하여 필요한 데이터의 주소 정보를 상기 CPU로부터 전달받는 것

인 CPU와 GPU 간의 협업 시스템.

청구항 2

삭제

청구항 3

제1항에 있어서, 상기 작업관리부는

상기 GPU의 주소 공간과 상기 작업에 필요한 데이터의 주소 정보를 매핑한 테이블을 상기 주소매핑부에 로드하는 것

인 CPU와 GPU 간의 협업 시스템.

청구항 4

제1항에 있어서, 상기 작업관리부는

코프로세서 인터페이스와 동일한 인터페이스로 상기 CPU와 연결된 것

인 CPU와 GPU 간의 협업 시스템.

청구항 5

제1항에 있어서, 상기 작업관리부는

상기 CPU가 요청한 작업을 상기 GPU의 각 코어에 분배하여 요청하고, 상기 GPU의 각 코어의 작업 상태를 모니터링하는 것

인 CPU와 GPU 간의 협업 시스템.

청구항 6

제1항에 있어서,

상기 GPU가 처리 중인 데이터 다음에 처리되어야할 데이터를 상기 메인 메모리로부터 캐시 메모리로 가져오는 프리페처

를 더 포함하는 CPU와 GPU 간의 협업 시스템.

청구항 7

제6항에 있어서, 상기 프리페처는

상기 작업관리부로부터 작동신호를 입력받으면, 상기 GPU에 필요한 데이터를 상기 메인 메모리에서 상기 캐시 메모리로 가져오고 처리완료된 데이터를 상기 캐시 메모리에서 제거하는 것

인 CPU와 GPU 간의 협업 시스템.

청구항 8

제1항에 있어서,

상기 CPU의 캐시 메모리에 저장된 데이터와 상기 GPU의 캐시 메모리에 저장된 데이터를 일치시키는 캐시일관성제어부

를 더 포함하는 CPU와 GPU 간의 협업 시스템.

청구항 9

제8항에 있어서, 상기 작업관리부는

상기 CPU의 캐시 메모리에 저장된 데이터와 상기 GPU의 캐시 메모리에 저장된 데이터를 일치시킬 필요가 있는지 여부를 확인하고, 데이터 일치가 필요하면 상기 캐시일관성제어부를 작동시키는 것

인 CPU와 GPU 간의 협업 시스템.

청구항 10

CPU가 요청하는 작업을 전달받아 GPU로 요청하는 단계;

상기 GPU의 주소 공간을 메인 메모리의 주소 공간과 매핑하는 단계; 및

상기 GPU가 처리한 작업 결과를 상기 CPU로 전달하는 단계를 포함하되,

상기 CPU가 요청하는 작업을 전달받아 GPU로 요청하는 단계는

상기 CPU로부터 작업에 해당하는 코드 정보 및 작업에 필요한 데이터의 주소 정보를 전달받는 단계를 포함하는 것

인 CPU와 GPU 간의 협업 방법.

청구항 11

삭제

청구항 12

제10항에 있어서, 상기 CPU가 요청하는 작업을 전달받아 GPU로 요청하는 단계는

상기 전달받은 작업을 분배하여 상기 GPU의 각 코어에 요청하고, 상기 GPU의 각 코어의 작업 상태를 모니터링하는 단계를 포함하는 것

인 CPU와 GPU 간의 협업 방법.

청구항 13

제10항에 있어서, 상기 GPU의 주소 공간을 메인 메모리의 주소 공간과 매핑하는 단계는

상기 GPU의 주소 공간과 상기 작업에 필요한 데이터의 주소 정보를 매핑한 테이블을 생성하는 단계; 및

상기 테이블을 참조하여 상기 GPU가 주소를 변환하는 단계를 포함하는 것

인 CPU와 GPU 간의 협업 방법.

청구항 14

제10항에 있어서,

상기 GPU가 처리 중인 데이터 다음에 처리되어야할 데이터를 확인하는 단계; 및

상기 확인된 데이터를 상기 메인 메모리로부터 캐시 메모리로 가져오는 단계

를 더 포함하는 CPU와 GPU 간의 협업 방법.

청구항 15

제10항에 있어서,

상기 CPU의 데이터와 상기 GPU의 데이터를 일치시킬 필요가 있는 경우에 양 데이터를 일치시키기 위해 캐시일관성 제어 모듈을 작동시키는 단계

를 더 포함하는 CPU와 GPU 간의 협업 방법.

명 세 서

기 술 분 야

[0001] 본 발명은 CPU와 그래픽 프로세서(GPU) 간의 협업 시스템 및 그 방법에 관한 것으로서, 구체적으로는, CPU와 GPU 간의 효율적인 협업을 위한 메모리 구조와 관리 방법에 관한 것이다.

배 경 기 술

[0002] 최근 Samsung(社) Exynos, nVidia(社) Tegra, Texas Instrument(社) OMAP 등의 AP(Application Processor)에서 ARM cortex 계열의 멀티 CPU 및 nVidia 혹은 Imagination(社)의 SGX 계열의 멀티 GPU를 채택하여 one chip에 집적되는 추세이다.

[0003] 전통적으로 멀티 CPU의 경우 시스템 성능 향상을 위해 1차 혹은 2차 캐시를 공유하는 형태가 주류를 이루고 있다. 또한 각 CPU에 속한 캐시 간의 coherency를 위해 MESI(Modified, Exclusive, Shared, Invaild)와 같은 프로토콜이 채택되고 이를 위해 Snoop Control Unit(SCU)이 탑재되어 있다. 그리고 외부 메모리의 접근을 최소화하기 위해 write-back, write-once, write allocate 방식이 적용되었다.

[0004] Intel(社), AMD(社)가 주가 되어 PC 쪽에서 먼저 시작된 GP-GPU는 위에서 언급한 것과 같이 AP로 확장되어 one chip에 집적되고 있다. 공통적으로 하위 레벨의 캐시를 공유한다. 하지만 모바일 AP와 PC에서는 memory management의 방식에 큰 차이점이 존재한다.

[0005] 예컨대 AMD Fusion APU의 경우, CPU와 GPU 각각이 다른 page table을 가지고 동작을 한다. 이에 반해 ARM Mali

T604의 경우, Cortext A15과 같은 page table을 가지고 memory를 관리한다. 둘 중 어느 것이 더 좋은 것인지는 아직 검증이 되지 않고 있다.

[0006] 현재 CPU/GPU 집적 시스템에서 CPU는 Bridge(PC) 혹은 Bus(AP)를 통해 GPU를 제어하고 있다. 일반적으로 GPU는 주로 CPU를 통해 처리하여야할 작업들의 코드와 데이터를 메모리 인터페이스를 통해 위임받고, 이를 GPU 로컬 메모리에 복사한 후, GPU는 이를 처리하여 CPU의 메인 메모리에 결과를 다시 복사하는 구조이다. 이를 위해 현재 CPU-GPU 집적 시스템에서 운영체제의 소프트웨어 드라이버가 CPU에서 브릿지 혹은 버스 인터페이스를 통해 GPU를 제어하는 구조이며, 메모리 공유 및 캐시 컨트롤러는 이 제어 구조와는 별개로 작동한다.

[0007] 하지만 이로 인해 시스템 성능이 저하되기 때문에 CPU와 GPU 간의 직접적인 inter-processor communication이 필요하고 이를 위한 control unit이 별도로 추가되어야 한다. 또한 캐시 공유에 있어 CPU와 GPU가 별도의 page table을 가지는 것과 공통의 page table을 가지는 것 사이의 검증이 필요하다.

발명의 내용

해결하려는 과제

[0008] 본 발명은 전술한 문제점을 해결하기 위하여, 별도의 제어 모듈을 통해 GPU를 제어하여 CPU의 로드를 감소시킬 수 있는 CPU와 GPU 간의 협업 시스템 및 방법을 제공하는 것을 목적으로 한다.

[0009] 또한 멀티 프로세서 간의 캐시일관성 문제를 해소하는 종래의 프로토콜을 확장하여, CPU와 GPU 간의 캐시일관성 유지에 효율적인 캐시일관성 제어 모듈을 제공하는 것을 목적으로 한다.

과제의 해결 수단

[0010] 본 발명은 CPU와 GPU 간의 협업 시스템에 있어서, 상기 CPU가 요청하는 작업을 전달받아 상기 GPU에 요청하며, 상기 GPU가 처리한 작업 결과를 상기 CPU로 전달하는 작업관리부; 상기 GPU의 주소 공간과 메인 메모리의 주소 공간의 매핑을 보조하는 주소매핑부; 상기 GPU가 처리 중인 데이터 다음에 처리되어야할 데이터를 상기 메인 메모리로부터 캐시 메모리로 가져오는 프리페처; 및 상기 CPU의 캐시 메모리에 저장된 데이터와 상기 GPU의 캐시 메모리에 저장된 데이터를 일치시키는 캐시일관성제어부를 포함하는 CPU와 GPU 간의 협업 시스템을 제공한다.

[0011] 본 발명의 일면에 따르면, 상기 작업관리부는 상기 CPU가 요청하는 작업에 해당하는 코드 정보 및 상기 작업을 수행하기 위하여 필요한 데이터의 주소 정보를 상기 CPU로부터 전달받는 것인 CPU와 GPU 간의 협업 시스템을 제공한다.

[0012] 본 발명의 다른 일면에 따르면, 상기 작업관리부는 상기 GPU의 주소 공간과 상기 작업에 필요한 데이터의 주소 정보를 매핑한 테이블을 상기 주소매핑부에 로드하는 것인 CPU와 GPU 간의 협업 시스템을 제공한다.

[0013] 본 발명의 다른 일면에 따르면, 상기 작업관리부는 상기 CPU가 요청한 작업을 상기 GPU의 각 코어에 분배하여 요청하고 상기 GPU의 각 코어의 작업 상태를 모니터링하는 것인 CPU와 GPU 간의 협업 시스템을 제공한다.

[0014] 본 발명의 또 다른 일면에 따르면, 상기 프리페처는 상기 작업관리부로부터 작동신호를 입력받으면, 상기 GPU에 필요한 데이터를 상기 메인 메모리에서 상기 캐시 메모리로 가져오고 처리완료된 데이터를 상기 캐시 메모리에서 제거하는 것인 CPU와 GPU 간의 협업 시스템을 제공한다.

[0015] 본 발명의 또 다른 일면에 따르면, 상기 작업관리부는 상기 CPU의 캐시 메모리에 저장된 데이터와 상기 GPU의 캐시 메모리에 저장된 데이터를 일치시킬 필요가 있는지 여부를 확인하고, 데이터 일치가 필요하면 상기 캐시일관성제어부를 작동시키는 것인 CPU와 GPU 간의 협업 시스템을 제공한다.

[0016] 본 발명은 CPU가 요청하는 작업을 전달받아 GPU로 요청하는 단계; 상기 GPU의 주소 공간을 메인 메모리의 주소 공간과 매핑하는 단계; 상기 GPU가 처리한 작업 결과를 상기 CPU로 전달하는 단계; 상기 GPU가 처리 중인 데이터 다음에 처리되어야할 데이터를 확인하는 단계; 상기 확인된 데이터를 상기 메인 메모리로부터 캐시 메모리로 가져오는 단계; 및 상기 CPU의 데이터와 상기 GPU의 데이터를 일치시킬 필요가 있는 경우에 양 데이터를 일치시키기 위해 캐시일관성 제어 모듈을 작동시키는 단계를 포함하는 CPU와 GPU 간의 협업 방법으로 이용될 수 있다.

[0017] 본 발명의 일면에 따르면, 상기 CPU가 요청하는 작업을 전달받아 GPU로 요청하는 단계는, 상기 CPU로부터 작업

에 해당하는 코드 정보 및 작업에 필요한 데이터의 주소 정보를 전달받는 단계; 및 상기 전달받은 작업을 분배하여 상기 GPU의 각 코어에 요청하고, 상기 GPU의 각 코어의 작업 상태를 모니터링하는 단계를 포함하는 것인 CPU와 GPU 간의 협업 방법을 제공한다.

[0018] 본 발명의 다른 일면에 따르면, 상기 GPU의 주소 공간을 메인 메모리의 주소 공간과 매핑하는 단계는, 상기 GPU의 주소 공간과 상기 작업에 필요한 데이터의 주소 정보를 매핑한 테이블을 생성하는 단계; 및 상기 테이블을 참조하여 상기 GPU가 주소를 변환하는 단계를 포함하는 것인 CPU와 GPU 간의 협업 방법을 제공한다.

발명의 효과

[0019] 본 발명은 GPU의 작업을 관리하는 제어 모듈과 동기화되어 CPU가 GPU에 위임할 데이터 영역만을 공유하는 CPU와 GPU 간의 협업 시스템을 제공한다. 따라서 메모리 간의 복사없이 CPU가 사용하는 가상 주소 공간을 캐시에서 바로 접근할 수 있어 성능이 크게 향상된다.

[0020] 또한 캐시 레벨에서의 공유 구조에서 작업관리 모듈의 동작과 동기화되어 메인 메모리에서 캐시로의 프리페치를 효율적으로 제어할 수 있어, GPU의 직접적인 메인 메모리 접근을 최소화할 수 있는 장점을 가진다.

[0021] 그리고 CPU와 GPU의 캐시의 Coherency를 위한 제어는 작업에 따라 CPU가 작업관리 모듈을 통해 Enable/Disable 할 수 있는 구조이기 때문에 스누핑에 의한 성능 저하의 문제를 최적화할 수 있는 구조를 제공한다.

도면의 간단한 설명

[0022] 도 1은 종래의 CPU와 GPU 간의 협업 시스템의 구조를 나타낸 도면.

도 2는 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템의 구조를 나타낸 도면.

도 3은 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 작업관리부(Job Manager)의 구조를 나타낸 도면.

도 4는 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 주소매핑부(Re-mapper)의 구조를 나타낸 도면.

도 5는 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 프리페처(Pre-fetcher)의 구조를 나타낸 도면.

도 6 내지 도 10은 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 캐시일관성제어부(Cache Coherency Controller)의 구조와 이를 설명하기 위한 도면.

도 11은 본 발명의 일실시예에 따른 CPU와 GPU 간의 확장된 협업 시스템의 구조를 나타낸 도면.

발명을 실시하기 위한 구체적인 내용

[0023] 본 발명의 이점 및 특징, 그리고 그것들을 달성하는 방법은 첨부되는 도면과 함께 상세하게 후술 되어 있는 실시예들을 참조하면 명확해질 것이다. 그러나 본 발명은 이하에서 개시되는 실시예들에 한정되는 것이 아니라 서로 다른 다양한 형태로 구현될 것이며, 단지 본 실시예들은 본 발명의 개시가 완전하도록 하며, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에게 발명의 범주를 완전하게 알려주기 위해 제공되는 것이며, 본 발명은 청구항의 범주에 의해 정의된다.

[0024] 한편, 본 명세서에서 사용된 용어는 실시예들을 설명하기 위한 것이며 본 발명을 제한하고자 하는 것은 아니다. 본 명세서에서, 단수형은 문구에서 특별히 언급하지 않는 한 복수형도 포함한다. 명세서에서 사용되는 "포함한다(comprises)" 및/또는 "포함하는(comprising)"은 언급된 구성소자, 단계, 동작 및/또는 소자는 하나 이상의 다른 구성소자, 단계, 동작 및/또는 소자의 존재 또는 추가를 배제하지 않는다. 이하, 첨부된 도면을 참조하여 본 발명의 실시예를 상세히 설명하기로 한다.

[0025] 도 2는 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템의 구조를 나타낸 도면이다.

[0026] 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템은 도 1에 도시된 종래의 CPU와 GPU 간의 협업 시스템에

있어서, 작업관리부(Job Manager, 200), 주소매핑부(Re-mapper, 210), 프리페처(Pre-fetcher, 220) 및 캐시일관성제어부(Cache Coherency Controller, 230)를 포함한다.

[0027] 작업관리부(Job Manager : CPU/GPU Inter Processor Communication Controller, 200)는 CPU가 bus 혹은 bridge를 통해 GPU를 구동하지 않고 직접 구동할 수 있도록 서로 간의 인터페이스를 지정하고 통신하도록 한다.

[0028] 작업관리부(200)는 CPU의 co-processor 인터페이스로 CPU와 밀접하게 연결되어, CPU에서 발생한 요청사항을 다수개의 GPU 코어에 나누어 요청하고 처리 결과를 CPU로 알려주는 역할을 담당한다. 따라서 작업관리부(200)는 이에 필요한 정보들을 CPU로부터 주고받을 수 있는 인터페이스를 포함한다.

[0029] 주소매핑부(Re-mapper : Memory Management Unit for GPU, 210)는 GPU의 주소 공간을 CPU가 사용하는 메인 메모리의 주소 공간으로 매핑을 보조하는 기능을 한다.

[0030] 기존 GPU는 가상 주소메모리 공간을 사용하지 않고, 직접 물리 주소에 접근한다. 설사 GPU가 별도의 MMU를 통해 가상 주소를 사용하더라도 CPU에서 사용하는 주소영역과는 다르기 때문에 GPU가 바라보는 주소 공간을 CPU가 사용하는 메인 메모리의 페이지 테이블을 이용하여 주소 공간으로 매핑하는 기능이 필요한데, 이 기능을 주소매핑부(210)가 담당하는 것이다. GPU 측에서는 주소매핑부(210)를 통해 Unified Shared Memory에 접근한다.

[0031] 프리페처(Pre-fetcher, 220)는 주 메모리와 L2 캐시로부터의 데이터 블록 패턴을 발견하여 이를 참조를 위한 패턴으로 받고 이를 필요한 데이터를 pre-fetch한다.

[0032] 캐시일관성제어부(Cache Coherency Controller, 230)는 CPU와 GPU가 서로 cache를 공유할 수 있도록 제어하는 기능을 한다. 이는 CPU뿐만 아니라 GPU와의 coherency도 유지할 수 있도록 기존 SCU(Snoop Control Unit)를 확장하여 설계한다.

[0033] 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에 의한 협업 과정은 아래와 같이 진행된다.

[0034] CPU는 GPU 코어용으로 컴파일된 코드와 데이터, 그리고 GPU 코어별로 분할된 데이터의 주소 및 오프셋 정보들을 작업관리부(200)의 정해진 인터페이스에 전달한다. 작업관리부(200)는 주어진 메인 메모리의 데이터 주소 정보를 GPU 주소공간으로 remapping 하여 주소매핑부(210)에 로드한다.

[0035] 작업관리부(200)는 주어진 주소 정보를 바탕으로 프리페처(220)를 작동시켜 메인 메모리에서 L2 캐시로 데이터를 미리 가져오고, CPU에서 cache coherency의 제어가 필요할 경우 캐시일관성제어부(230)를 작동시킨다.

[0036] 작업관리부(200)는 GPU의 각 코어에 작업을 할당하며, 할당된 작업이 GPU에서 처리되는 동안 이어서 다음에 처리할 데이터를 프리페처(220)를 통해 L2로 가져오고, 이미 처리된 데이터가 있을 경우 메인 메모리에 해당 캐시 데이터를 Flush시킨다.

[0037] GPU는 위임받은 작업이 끝나면 작업관리부(200)에게 완료 신호를 보내며, 작업관리부(200)는 CPU로 작업이 완료되었음을 전달한다.

[0038] 도 3은 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 작업관리부의 구조를 나타낸 도면이다.

[0039] 기존 CPU가 GPU에게 작업을 위임하는 방식은 CPU가 제어의 요청을 시스템 버스를 통해 GPU의 Host Request Queue를 직접 관리하는 방식이다. 따라서 CPU는 GPU의 디바이스 드라이버 소프트웨어가 시스템 버스의 인터럽트 인터페이스를 통해 GPU의 동작을 지속적으로 관리하여야 하는 구조이다.

[0040] 반면에 본 발명은 이를 개선하기 위해 작업관리부의 별도의 하드웨어 장치를 통해 GPU가 작동하는 작업들의 관리를 위임하는 장치이다. 작업관리부를 통해 CPU는 GPU와 관련된 관리적인 로드가 크게 경감될 수 있다.

[0041] 작업관리부는 CPU의 co-processor 명령어와 같은 인터페이스로 연결되어 GPU가 실행해야할 작업 및 메모리 주소, 코어별 오프셋, 파라미터 등을 설정할 수 있는 레지스터들을 제공한다. 또한 GPU의 각 코어별 작업의 상태 및 동작을 모니터링할 수 있는 기능을 제공할 수 있다.

[0042] 작업관리부는 하나의 Host CPU의 인터페이스뿐만 아니라, 추가적인 인터페이스로 확장(최대 4개)이 가능하도록 설계되어 멀티 코어 프로세서와 다른 GPU 하드웨어와의 협업과 같은 이기종의 프로세서들과의 동작을 관리하는 역할을 수행할 수 있다.

[0043] 도 4는 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 주소매핑부의 구조를 나타낸 도면이다.

[0044] OpenCL, OpenGL의 모델은 CPU-GPU의 시스템이 non-unified memory 구조에서 동작을 가정하고 설계되었다. 즉,

물리적으로 분리된 메모리를 가지고 있기 때문에 CPU가 사용하는 가상 메모리 주소공간과 GPU가 사용하는 메모리 주소공간은 서로 다르게 사용하도록 발전해왔다. 그러나 최근 CPU-GPU의 구조는 SoC상에서 공유 메모리 기반의 구조로 개발되면서 CPU와 GPU는 Unified Shared Memory 상에서 주소체계 및 변환에 대한 필요가 발생하였다. 이 문제를 해결하기 위한 통상적인 방법은 GPU도 CPU와 같이 각각의 TLB를 통해 메인 메모리 상의 같은 페이지 테이블을 참조하여 동일한 가상 메모리 주소공간을 사용하도록 하는 방법이다.

[0045] 일반적으로 GPU는 CPU로부터 대용량의 데이터 처리를 위임받고, 이를 순차적으로 나누어 병렬 처리하여 결과를 되돌려주는 방식이다. 이러한 점을 고려하였을 때 Unified shared memory 접근을 위해 TLB를 통해 공통의 주소 매핑 테이블을 공유하는 구조는 문제점이 있다. GPU는 큰 범위의 데이터를 전달받게 되고, GPU를 구성하는 각 코어들은 각각의 해당 공간을 TLB를 통해 변환하게 된다.

[0046] 그러나 제한적인 TLB의 크기와 GPU의 분할 및 순차적인 처리 특성상 TLB에 존재하는 변환 정보의 재사용률이 낮은 점을 고려할 때, GPU가 처리해야할 데이터가 클 경우 메인 메모리의 페이지 테이블에 접근하는 횟수가 증가할 수밖에 없다. 또한 많은 GPU 코어들이 각각의 TLB를 가지고 메모리 버스에 접근할 경우 더욱 많은 트래픽이 발생할 뿐만 아니라 구현의 복잡도 역시 높아진다.

[0047] 이러한 문제점을 개선하기 위해 본 발명은 다음의 접근 방식으로 설계된다. CPU가 GPU에게 작업을 위임하기 전에 필요한 데이터의 범위와 위치가 정해져 있기 때문에 CPU에서 OpenCL/OpenGL API를 통한 드라이버는 GPU로 넘겨질 메모리를 가능한 연속된 페이지에 allocation하고, 해당 페이지의 물리적 주소를 연속된 GPU의 가상주소로 매핑하는 테이블을 주소매핑부에 로딩한다. 이때 데이터가 연속된 페이지에 위치하지 않고 페이지 단위로 fragmentation되었으면 이 페이지 정보를 GPU를 위한 연속된 가상 주소공간으로 remapping하여 주소 매핑 테이블에 반영한다.

[0048] 주소 매핑 테이블에는 GPU에 넘겨질 모든 데이터의 페이지주소 정보들이 포함되어 GPU는 주소변환을 위한 추가적인 메모리 접근 없이 주소매핑부에 로딩된 매핑 테이블의 정보를 참조하여 주소변환을 진행한다.

[0049] 주소매핑부의 주소변환은 GPU의 각 코어의 개수만큼 구현된 translator 장치에 의해 매핑 테이블을 참조하여 수행되고, 변환된 주소 정보로 cache controller를 통해 Unified Shared Memory로 접근한다.

[0050] 도 5는 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 프리페처의 구조를 나타낸 도면이다. GPU는 위임받은 작업을 분할하여 병렬적으로 그리고 순차적으로 처리하게 되고, 본 발명은 이를 보다 효율적으로 관리하기 위해 도 5와 같은 구조로 프리페처를 설계한다.

[0051] 작업관리부를 통해 GPU가 동작을 시작하는 것과 함께 프리페처는 L2의 캐시 영역을 GPU의 코어가 한 번의 작업에 필요한 공간의 2배를 예약하고 이를 두 개의 windows로 구분한다. 첫 번째 윈도우에는 현재 GPU의 작업에 필요한 데이터를 로딩하고, 두 번째 windows의 영역에는 다음에 이어서 처리할 작업을 위한 데이터를 로딩하기 위해 예약한다.

[0052] 이렇게 예약된 window 영역은 L2의 캐시 컨트롤러가 기존의 eviction rule을 적용하지 않으며 두 개의 windows는 GPU의 메모리 latency hiding을 위해 전용으로 사용된다.

[0053] 도 6은 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템에서 캐시일관성제어부의 구조를 나타낸 도면이다.

[0054] 캐시일관성제어부는 멀티코어 CPU와 GPU와의 L1 cache간의 coherency를 위한 프로토콜과 더불어 프로토콜에 따른 각 코어간의 memory-to-cache, cache-to-cache의 데이터 전송, 그리고 앞서 설명한 pre-fetchering을 위한 L2 캐시의 제어를 담당한다.

[0055] 캐시일관성제어부는 Single-Core CPU를 위한 구조와 이를 확장하는 구조의 두 가지로 설계된다. 첫 번째 Single-core CPU와 GPU간의 unified memory상에서의 공유를 위한 coherency 모델은 도 7에 도시된 바와 같다.

[0056] 도 7에서 이를 위한 상태 변환의 프로토콜은 도 8과 같다. 도 8의 프로토콜의 특징은 기본적으로 L1 캐시 간의 데이터 전송 기반으로 한다. 그리고 GPU에게 작업을 위임한 CPU가 GPU의 동작처리 과정 중에 해당 데이터를 다시 접근하는 경우가 낮기 때문에, Invalidation 기반의 GPU와의 coherency를 위해 snooping을 최소화한다. 즉 데이터의 ownership뿐만 아니라, 캐시된 데이터 자체도 복사되도록 하는 방식이다. 따라서 GPU와 공유된 데이터는 하나의 copy만 L1 캐시에 존재하도록 한다.

[0057] 그러나 멀티코어 CPU와 GPU를 위한 구조는 CPU간의 coherency의 프로토콜과 함께 동작하여야 하기 때문에 보다

복잡하다. 이를 위해 MOESI기반의 Dragon 프로토콜을 확장한다.

[0058] 도 9는 확장된 프로토콜에 필요한 상태들의 정의를 보여준다. RD의 상태가 추가되고 INV_REQ의 invalidation request가 추가된다. RD의 상태는 GPU가 데이터를 자신의 cache에 로딩 후, 데이터를 쓰기를 진행할 때의 상태를 나타낸다. 그리고 CPU간의 공유와 GPU와의 공유를 구분하기 위한 condition이 추가되는데 이것은 앞서 설명한 주소매핑부를 통해 제공된다. 주소매핑부는 자신의 테이블을 참조하여 접근하는 데이터의 경우 condition r을 true로 설정한다. 도 9에서 정의된 상태를 이용하여 설계된 coherency 프로토콜은 도 10과 같다.

[0059] 도 10에서 프로토콜은 기본적으로 앞서 설명한 Single-core CPU에서와 같이 GPU와의 공유되는 데이터는 기본적으로 invalidation을 기본으로 한다. 이것은 CPU가 위임한 작업을 위한 데이터에 대하여 CPU가 공유하여 쓰고자 할 때 update를 최소화하기 위해 기본적으로 GPU는 CPU의 공유된 캐시라인들을 invalidation하도록 한다.

[0060] 이러한 프로토콜을 포함하는 캐시일관성제어부의 개략적은 구조는 도 6에 도시된 바와 같고 캐시일관성제어부는 크게 세 가지 부분으로 구성된다.

[0061] 첫 번째는 앞서 설명한 프로토콜의 상태 변화를 조정하기 위한 comparator이다. comparator는 GPU와 CPU의 L1 cache controller로부터 주소와 line의 상태를 입력받아 이들의 상태를 관리한다.

[0062] 두 번째는 cache-to-cache 데이터 전송 unit이다. 이 unit은 comparator로부터 L1 cache간의 데이터 전송이 필요할 경우 이들 간의 데이터 전송을 담당한다.

[0063] 세 번째는 L2 cache controller이다. L2 controller는 통상적인 cache eviction rule을 적용하여 L2를 관리할 뿐만 아니라, 앞서 설명한 프리페처로부터 요청이 있을 경우 L2를 필요한 크기의 영역으로 partitioning하여 GPU의 프리페칭을 위해 필요한 메모리전송을 수행한다.

[0064] 도 11은 본 발명의 일실시예에 따른 CPU와 GPU 간의 협업 시스템이 확장된 시스템을 나타낸 도면으로, 도 11에 도시된 협업 시스템은 두 개의 CPU와 GPU가 메모리를 공유하는 구조이다.

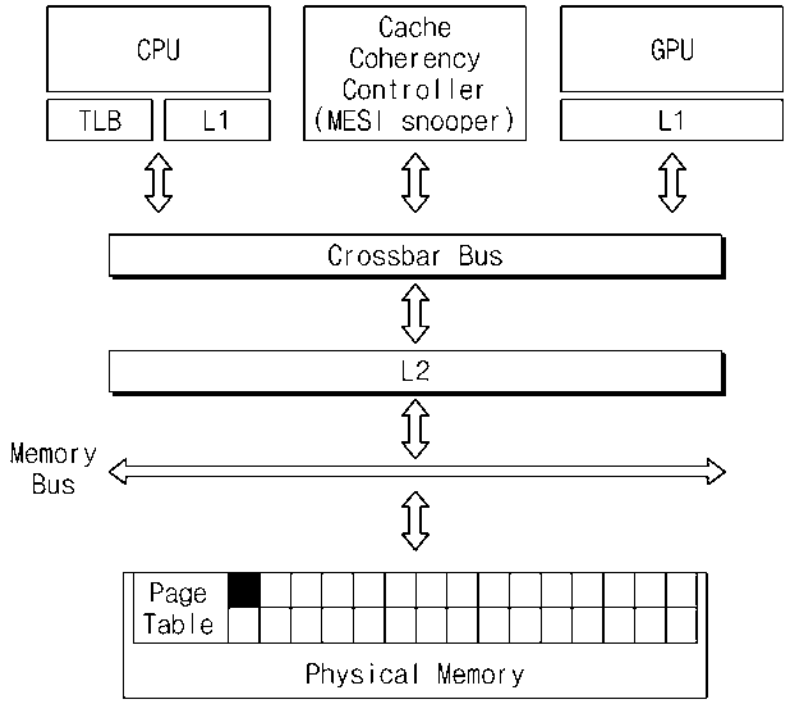
[0065] 전술한 CPU와 GPU 간의 협업 시스템의 구조는 L2 뿐만 아니라 L3 캐시를 통한 공유 구조로도 확장이 가능하며, 단일 CPU 뿐만 아니라 멀티 CPU와 GPU 간의 협업 구조로도 확장이 가능하다.

[0066] 멀티 CPU와 GPU는 L2 캐시는 각각 가지고 있으며, L3는 공유하는 구조이다. 작업관리부는 앞서 설명한 구조에서처럼 CPU와의 인터페이스를 통해 작동한다. 그러나 캐시일관성제어부는 CPU간의 메모리 공유를 위해 항상 동작하여야 한다.

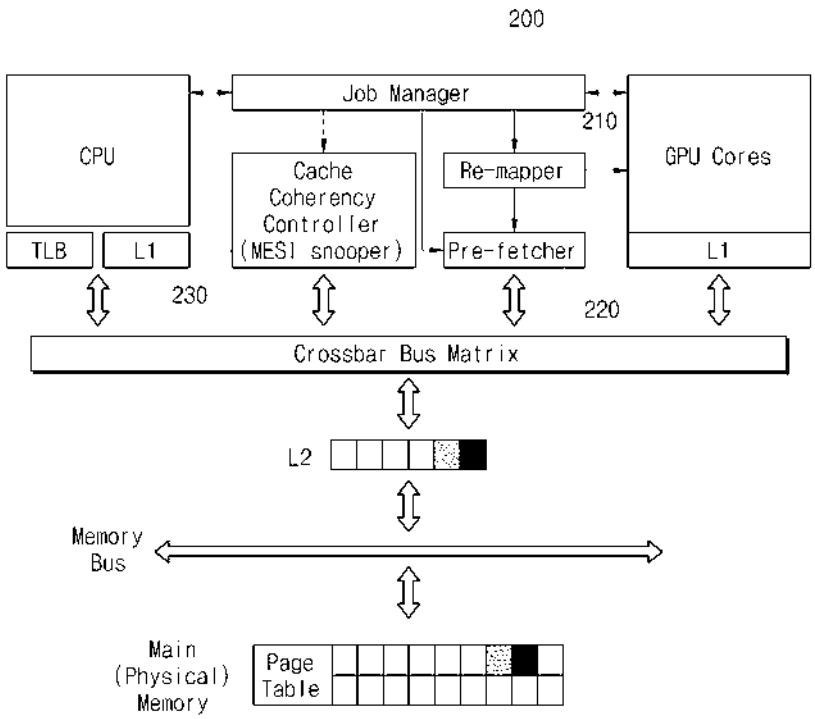
[0067] 이상의 설명은 본 발명의 기술적 사상을 예시적으로 설명한 것에 불과한 것으로서, 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자라면, 본 발명의 본질적 특성을 벗어나지 않는 범위에서 다양한 수정 및 변형이 가능하다. 따라서, 본 발명에 표현된 실시예들은 본 발명의 기술적 사상을 한정하는 것이 아니라, 설명하기 위한 것이고, 이러한 실시예에 의하여 본 발명의 권리범위가 한정되는 것은 아니다. 본 발명의 보호 범위는 아래의 특허청구범위에 의하여 해석되어야 하고, 그와 동등하거나, 균등한 범위 내에 있는 모든 기술적 사상은 본 발명의 권리범위에 포함되는 것으로 해석되어야 할 것이다.

도면

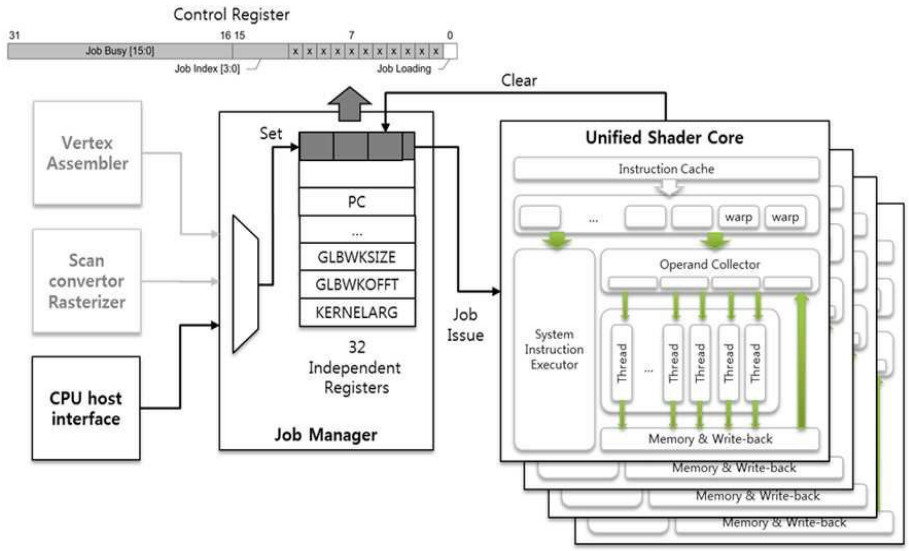
도면1


CPU
TLB
L1
Cache Coherency Controller (MESI snooper)
GPU
L1
Crossbar Bus
L2
Memory Bus
Page Table
Physical Memory

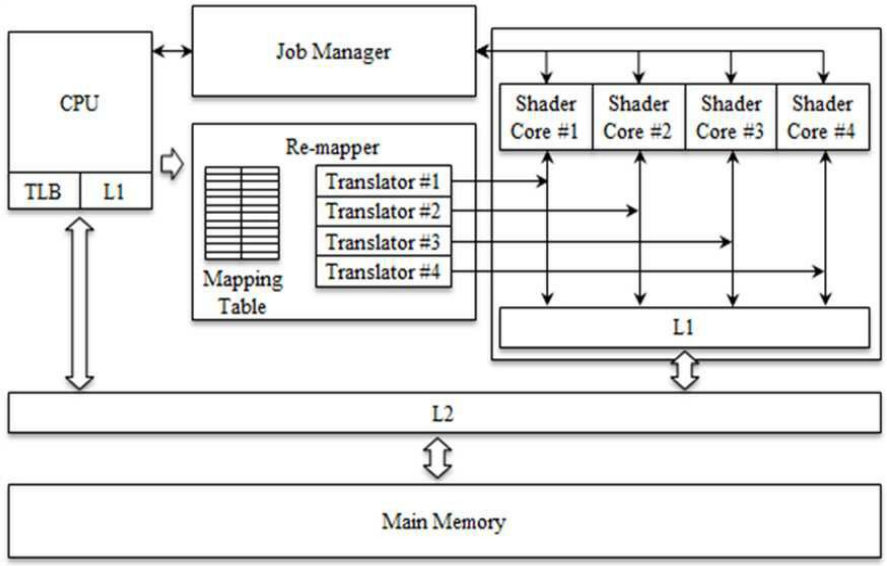
도면2


200
CPU
TLB
L1
Job Manager
210
Cache Coherency Controller (MESI snooper)
Re-mapper
Pre-fetcher
GPU Cores
L1
230
220
Crossbar Bus Matrix
L2
Memory Bus
Main (Physical) Memory
Page Table

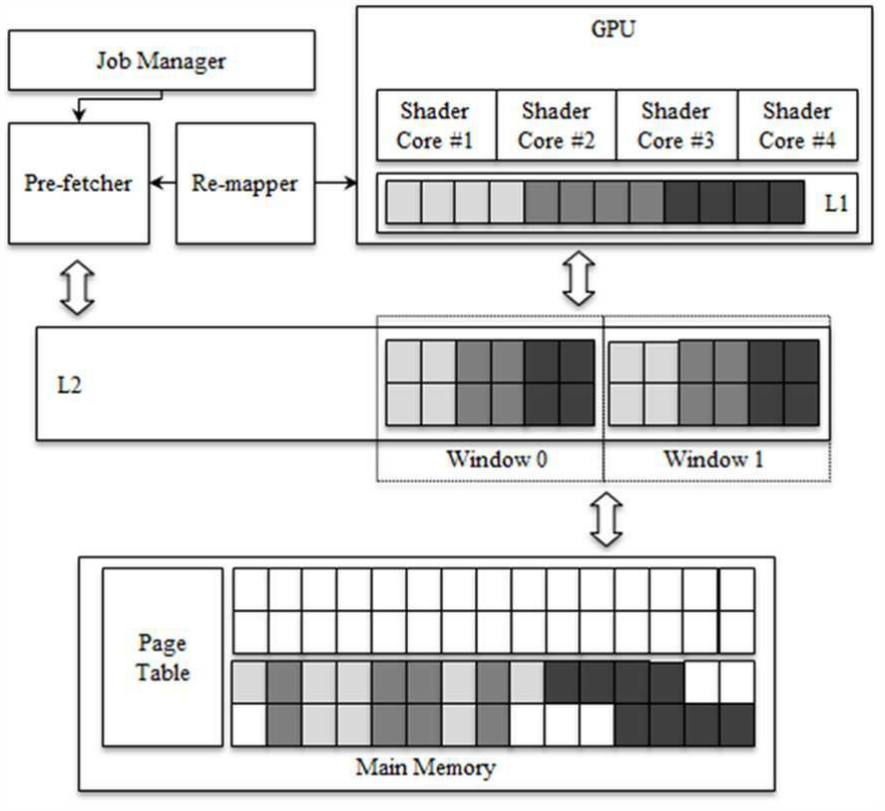
도면3


Control Register
31
16 15
7
0
Job Busy [15:0]
x x x x x x x x x x x x x x x x
Job Index [3:0]
Job Loading
Clear
Vertex Assembler
Scan convertor Rasterizer
CPU host interface
Set
PC
...
GLBWKSIZE
GLBWKOFFT
KERNELARG
32 Independent Registers
Job Manager
Job Issue
Unified Shader Core
Instruction Cache
warp
warp
Operand Collector
System Instruction Executor
Thread
Memory & Write-back
Memory & Write-back
Memory & Write-back
Memory & Write-back

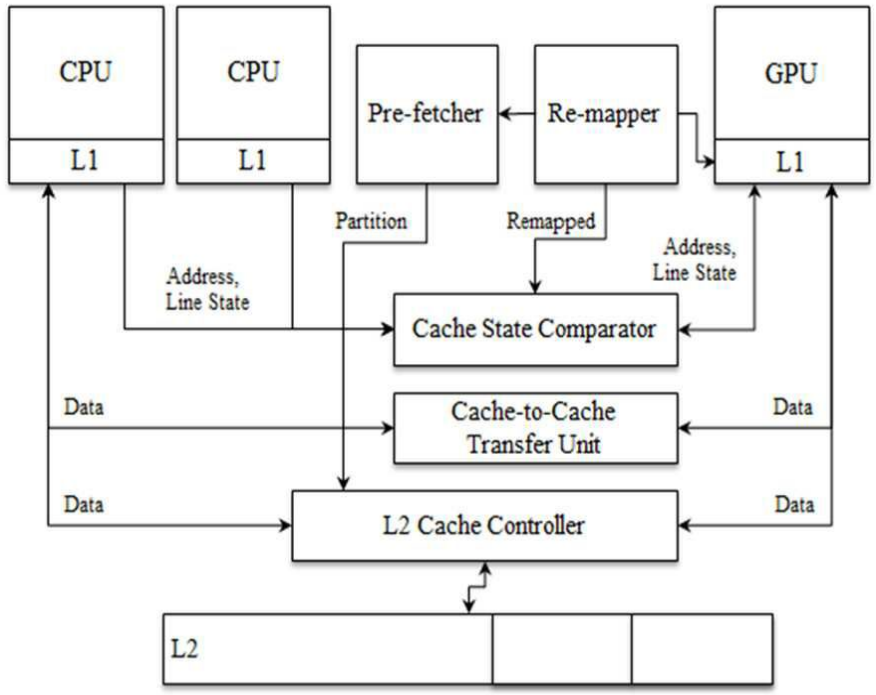
도면4


CPU
TLB
L1
Job Manager
Re-mapper
Translator #1
Translator #2
Translator #3
Translator #4
Mapping Table
Shader Core #1
Shader Core #2
Shader Core #3
Shader Core #4
L1
L2
Main Memory

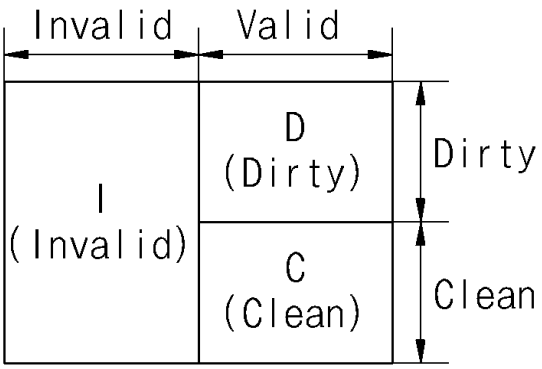
도면5


Job Manager
GPU
Shader Core #1
Shader Core #2
Shader Core #3
Shader Core #4
Pre-fetcher
Re-mapper
L1
L2
Window 0
Window 1
Page Table
Main Memory

도면6


CPU
CPU
Pre-fetcher
Re-mapper
GPU
L1
L1
L1
Partition
Remapped
Address, Line State
Address, Line State
Cache State Comparator
Data
Cache-to-Cache Transfer Unit
Data
Data
L2 Cache Controller
Data
L2

도면7



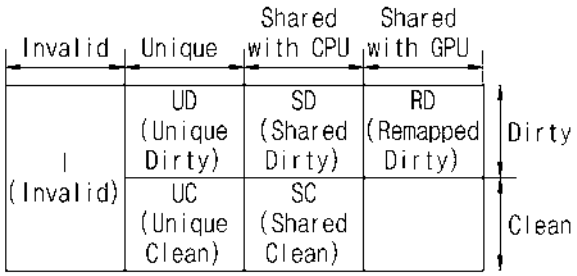
D: Dirty, must be written back
C: Clean
Invalid: Invalid

도면8

- LOAD : a read from processor
- STORE : a write from processor
- L_REQ : a load request to cache controller
- state() : change state to the next
- supply() : supply local cached data to others
- write_back() : write local cached data to next level cache

Current State	Request			
	LOAD	STORE	L_REQ	Eviction
I	L_REQ state(C)	L_REQ state(D)	-	-
C	state(C)	state(D)	supply() state(I)	state(I)
D	state(D)	state(D)	supply() state(I)	write_back() state(I)

도면9



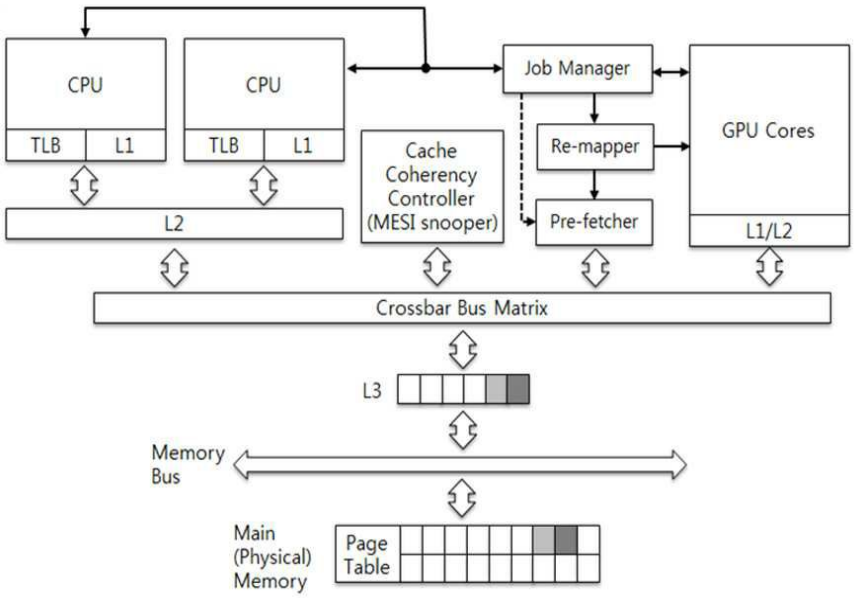
UD: Not shared, dirty, must be written back
SD: Shared with CPU, dirty, must be written back to memory
UC: Not shared, clean
SC: Shared with CPU, no need to write back, may be clean or dirty
RD: Shared with GPU, must be written back to memory
Invalid: Invalid

도면10

- LOAD : a read from processor
- STORE : a write from processor
- L_REQ : a load request to cache controller
- UP_REQ : an update request to cache controller
- INV_REQ : an invalidate request to cache controller
- s : condition, true if shared
- r : condition, true if remapped
- state() : change state to the next
- supply() : supply local cached data to others
- update() : update local copy from next level cache or others
- write_back() : write local cached data to next level cache

Current State	Request					
	LOAD	STORE	L_REQ	UP_REQ	INV_REQ	Eviction
I	L_REQ If ~s, state(UC) If s, state(SC)	L_REQ If ~s, state(UD) If s and ~r, UP_REQ and state(SD) If s and r, INV_REQ and state(RD)	-	-	-	-
SC	state(SC)	UP_REQ If ~s, state(UD) If s and ~r, UP_REQ and state(SD) If s and r, INV_REQ and state(RD)	state(SC)	update() state(SC)	state(I)	state(I)
UC	state(UC)	state(UD)	supply() state(SC)	-	-	state(I)
SD	state(SD)	UP_REQ If ~s, state(UD) If s and ~r, UP_REQ and state(SD) If s and r, INV_REQ and state(RD)	supply() state(SD)	update() state(SC)	write_back() state(I)	write_back() state(I)
UD	state(UD)	state(UD)	supply() state(SD)	-	-	write_back() state(I)
RD	state(RD)	UP_REQ state(RD)	supply() state(RD)	update() state(RD)	-	write_back() state(I)

도면11


CPU
CPU
TLB
L1
TLB
L1
L2
Job Manager
Re-mapper
Pre-fetcher
Cache
Coherency
Controller
(MESI snooper)
GPU Cores
L1/L2
Crossbar Bus Matrix
L3
Memory
Bus
Main
(Physical)
Memory
Page
Table

(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2015년07월20일
(11) 등록번호 10-1537559
(24) 등록일자 2015년07월13일

(51) 국제특허분류(Int. Cl.)
G06T 7/00 (2006.01) *G06K 9/46* (2006.01)
(21) 출원번호 10-2013-0166368
(22) 출원일자 2013년12월30일
심사청구일자 2013년12월30일
(65) 공개번호 10-2015-0077654
(43) 공개일자 2015년07월08일
(56) 선행기술조사문헌
JP평성09044681 A
KR1020030087960 A
방송공학회 논문 "객체 추적 카메라 제어를 위한 고속의 움직임 검출 및 추적 알고리즘"

(73) 특허권자
전자부품연구원
경기도 성남시 분당구 새나리로 25 (야탑동)
(72) 발명자
황태호
서울특별시 송파구 동남로 225 래미안파크팰리스 108동 601호
김동순
경기 성남시 분당구 정자일로 248, 606동 3203호 (정자동, 파크뷰)
권진산
서울특별시 송파구 동남로11길 4(가락동, 미륭아파트) 103동 305호
(74) 대리인
특허법인지명

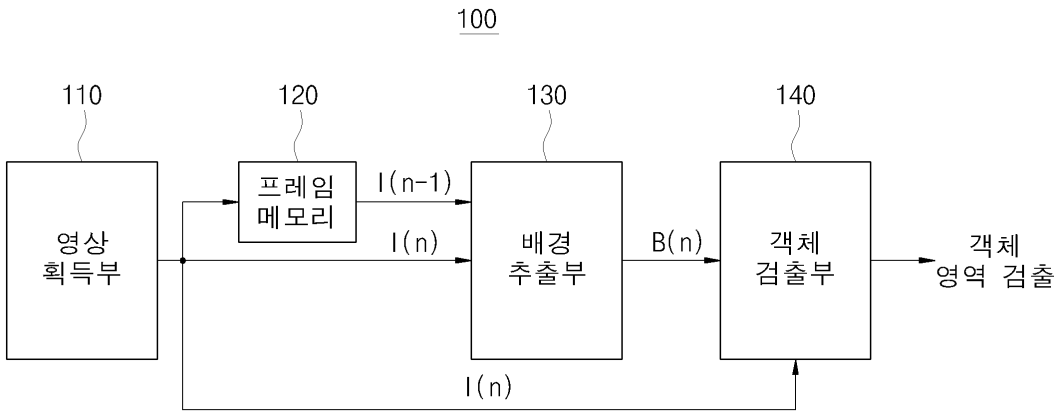
전체 청구항 수 : 총 7 항 심사관 : 신재철

(54) 발명의 명칭 **객체 검출 장치, 차량용 객체 검출 장치 및 이들의 방법**

(57) 요 약

객체 검출 방법이 개시된다. 이 방법은 옵티컬 플로우 방식을 이용하여 모든 화소들에 대한 이전 프레임과 현재 프레임 사이에서의 평균 이동량를 계산하는 과정과 계산된 평균 이동량만큼 상기 이전 프레임의 배경 프레임의 배경 이동량을 보상하여, 배경 이동량이 보상된 상기 배경 프레임을 상기 현재 프레임의 배경 프레임으로 구성하는 과정과, 상기 옵티컬 플로우 방식을 이용하여 상기 현재 프레임과 상기 구성된 배경 프레임 간의 각 화소들의 이동량을 계산하는 과정 및 기 설정된 임계치 이상의 이동량에 해당하는 각 화소들의 좌표값을 계산하여, 검출된 좌표값을 상기 현재 프레임 내의 객체 영역으로 검출하는 과정을 포함한다.

대 표 도 - 도1



이 발명을 지원한 국가연구개발사업
과제고유번호 10041125
부처명 지식경제부
연구관리전문기관 한국산업기술평가원
연구사업명 시스템반도체 상용화 기술개발 사업
연구과제명 SXGA급 자동차용 고화질 영상 처리기능 및 ECU통합 SoC 개발
기 여 율 1/1
주관기관 ㈜넥스트칩
연구기간 2011.12.01 ~ 2015.09.30

명 세 서

청구범위

청구항 1

옵티컬 플로우 방식을 이용하여 모든 화소들에 대한 이전 프레임과 현재 프레임 사이에서의 평균 이동량를 계산하는 과정;

계산된 평균 이동량만큼 상기 이전 프레임의 배경 프레임의 배경 이동량을 보상하여, 배경 이동량이 보상된 상기 배경 프레임을 상기 현재 프레임의 배경 프레임으로 구성하는 과정;

상기 옵티컬 플로우 방식을 이용하여 상기 현재 프레임과 상기 구성된 배경 프레임 간의 각 화소들의 이동량을 계산하는 과정; 및

기 설정된 임계치 이상의 이동량에 해당하는 각 화소들의 좌표값을 계산하여, 검출된 좌표값을 상기 현재 프레임 내의 객체 영역으로 검출하는 과정을 포함하되,

상기 평균 이동량을 계산하는 과정은,

이전 프레임과 현재 프레임 간의 각 화소들의 이동량를 계산하여, 상기 각 화소들의 이동량을 크기 순으로 배열하는 과정;

크기 순으로 배열된 이동량을 일정 단위의 구간으로 등분하고, 등분된 구간들 중 가장 많은 이동량이 분포한 구간을 선정하는 과정; 및

선정된 상기 구간 내에 분포한 이동량들의 평균치를 상기 평균 이동량으로 계산하는 과정;

을 포함하는 객체 검출 방법.

청구항 2

삭제

청구항 3

제1항에 있어서, 상기 배경 프레임으로 구성하는 과정은,

상기 현재 프레임의 배경 프레임 내의 각 화소들의 화소값을 추출하는 과정;

추출된 상기 화소값에 기 설정된 학습 계수를 연산한 보상된 화소값을 계산하는 과정; 및

상기 보상된 화소값을 상기 배경 이동량이 보상된 배경 프레임 내의 각 화소들의 화소값으로 대체하는 과정;

을 포함하는 객체 검출 방법.

청구항 4

제1항에 있어서, 상기 현재 프레임 내의 객체 영역으로 검출하는 과정은,

상기 보상된 배경 프레임과 현재 프레임 간이 차이에 해당하는 마스크 프레임을 생성하는 과정;

상기 검출된 객체 영역을 상기 마스크 프레임을 이용하여 필터링하는 과정; 및

필터링된 상기 객체 영역을 상기 현재 프레임 내의 최종 객체 영역으로 검출하는 과정;

을 포함하는 객체 검출 방법.

청구항 5

제1항에 있어서, 상기 기 설정된 임계치는,

상기 이전 프레임과 현재 프레임을 획득하는 카메라의 이동 속도값에 따라 결정되는 것인 객체 검출 방법.

청구항 6

이동 환경에서 주변 상황을 촬영하여 이전 프레임과 현재 프레임을 순차적으로 출력하는 영상 획득부;

옵티컬 플로우 방식을 이용하여 모든 화소들에 대해 이전 프레임과 현재 프레임 사이에서의 평균 이동량를 계산하고, 계산된 평균 이동량만큼 상기 이전 프레임의 배경 프레임의 배경 이동량을 보상하여, 배경 이동량이 보상된 상기 배경 프레임을 상기 현재 프레임의 배경 프레임으로 추출하는 배경 추출부; 및

상기 옵티컬 플로우 방식을 이용하여 상기 현재 프레임과 상기 보상된 배경 프레임 간의 각 화소들의 이동량을 계산하고, 기 설정된 임계치 이상의 이동량에 해당하는 각 화소들의 좌표값을 계산하여, 검출된 좌표값을 상기 현재 프레임 내의 객체 영역으로 검출하는 객체 검출부를 포함하되,

상기 배경 추출부는,

이전 프레임과 현재 프레임 간의 각 화소들의 이동량를 계산하여, 상기 각 화소들의 이동량을 크기 순으로 배열하고, 크기 순으로 배열된 이동량을 일정 단위의 구간으로 등분하고, 등분된 구간들 중 가장 많은 이동량이 분포한 구간을 선정하여, 선정된 상기 구간 내에 분포한 이동량들의 평균치를 상기 평균 이동량으로 계산함을 특징으로 하는 객체 추출 장치.

청구항 7

차량에 탑재된 차량용 객체 검출 장치에 있어서,

상기 차량 주행 중에 상기 차량의 주변 상황을 촬영하여 이전 프레임과 현재 프레임을 순차적으로 출력하는 영상 획득부;

옵티컬 플로우 방식을 이용하여 모든 화소들에 대해 이전 프레임과 현재 프레임 사이에서의 평균 이동량를 계산하고, 계산된 평균 이동량만큼 상기 이전 프레임의 배경 프레임의 배경 이동량을 보상하여, 배경 이동량이 보상된 상기 배경 프레임을 상기 현재 프레임의 배경 프레임으로 추출하는 배경 추출부; 및

상기 옵티컬 플로우 방식을 이용하여 상기 현재 프레임과 상기 보상된 배경 프레임 간의 각 화소들의 이동량을 계산하고, 상기 차량 내의 전자 제어 유닛으로부터 제공되는 차량의 주행 속도값에 따라 임계치를 결정하고, 결정된 상기 임계치 이상의 이동량에 해당하는 각 화소들의 좌표값을 계산하여, 검출된 좌표값을 상기 현재 프레임 내의 객체 영역으로 검출하는 객체 검출부를 포함하되,

상기 배경 추출부는,

이전 프레임과 현재 프레임 간의 각 화소들의 이동량를 계산하여, 상기 각 화소들의 이동량을 크기 순으로 배열하고, 크기 순으로 배열된 이동량을 일정 단위의 구간으로 등분하고, 등분된 구간들 중 가장 많은 이동량이 분포한 구간을 선정하여, 선정된 상기 구간 내에 분포한 이동량들의 평균치를 상기 평균 이동량으로 계산함을 특징으로 하는 객체 추출 장치.

청구항 8

차량에 탑재된 차량용 객체 검출 장치를 이용한 객체 검출 방법에 있어서,

상기 차량 주행 중에 상기 차량의 주변 상황을 촬영하여 이전 프레임과 현재 프레임을 순차적으로 출력하는 과정

옵티컬 플로우 방식을 이용하여 모든 화소들에 대해 이전 프레임과 현재 프레임 사이에서의 평균 이동량를 계산하고, 계산된 평균 이동량만큼 상기 이전 프레임의 배경 프레임의 배경 이동량을 보상하여, 배경 이동량이 보상

된 상기 배경 프레임을 상기 현재 프레임의 배경 프레임으로 추출하는 과정; 및

상기 옵티컬 플로우 방식을 이용하여 상기 현재 프레임과 상기 보상된 배경 프레임 간의 각 화소들의 이동량을 계산하고, 상기 차량 내의 전자 제어 유닛으로부터 제공되는 차량의 주행 속도값에 따라 임계치를 결정하고, 결정된 상기 임계치 이상의 이동량에 해당하는 각 화소들의 좌표값을 계산하여, 검출된 좌표값을 상기 현재 프레임 내의 객체 영역으로 검출하는 과정;을 포함하되,

상기 현재 프레임의 배경 프레임으로 추출하는 과정에서 상기 평균 이동량을 계산하는 과정은,

이전 프레임과 현재 프레임 간의 각 화소들의 이동량를 계산하여, 상기 각 화소들의 이동량을 크기 순으로 배열하는 과정;

크기 순으로 배열된 이동량을 일정 단위의 구간으로 등분하고, 등분된 구간들 중 가장 많은 이동량이 분포한 구간을 선정하는 과정; 및

선정된 상기 구간 내에 분포한 이동량들의 평균치를 상기 평균 이동량으로 계산하는 과정을 포함하는 차량에 탑재된 차량용 객체 검출 장치를 이용한 객체 검출 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 객체 검출 장치, 차량용 객체 검출 장치 및 이들의 방법에 관한 것으로서, 보다 상세하게는 카메라와 같은 영상 획득 수단이 이동하는 환경에서 획득된 영상 내의 배경으로부터 객체를 검출하는 객체 검출 장치 및 이의 방법에 관한 것이다.

배 경 기 술

[0002] 일반적으로 영상 정보만을 이용하여 움직이는 물체 또는 사람과 같은 객체를 검출하는 방법에는 옵티컬 플로우(optical flow) 기술과 배경 분리(background subtraction) 기술이 대표적이다.

[0003] 옵티컬 플로우 기술은 두 영상 프레임 사이에서 각 화소들이 움직인 방향과 거리 벡터를 계산하여 비슷한 형질을 가지는 벡터들을 군집하는 방법으로 객체를 검출하는 방식이다.

[0004] 배경 분리 방법은 영상 프레임을 순차적으로 입력받아서, 프레임 단위로 각 영상 프레임의 화소값 정보를 누적하여 배경 프레임을 만들고 움직이는 물체 부분을 현재 프레임과 배경 프레임의 차이가 큰 부분으로 정의하여, 이러한 차이에 기초해 객체를 검출하는 방식이다.

[0005] 그러나, 상기와 같은 종래의 기술들은 모두 한 장소에 고정 설치된 카메라, 예를 들어 CCTV에서 수집된 영상에 대해서만 움직이는 객체를 오류 없이 찾아낼 수 있으며, 카메라가 움직임으로써 배경도 함께 움직이는 일반적인 영상에 대해서는 벡터들을 군집하지 못하거나 화소값의 차이로 인해 배경 프레임을 누적하지 못하고 객체를 정확히 찾을 수 없는 문제점이 있다.

발명의 내용

해결하려는 과제

[0006] 따라서, 본 발명의 목적은 카메라와 같은 영상 획득 수단이 이동하는 환경에서 획득된 영상에서 객체를 오류 없이 검출하는 객체 검출 장치, 차량용 객체 검출 장치 및 이의 방법을 제공하는 데 있다.

과제의 해결 수단

[0007] 상기와 같은 목적을 달성하기 위한 본 발명의 일면에 따른 객체 검출 방법은, 옵티컬 플로우 방식을 이용하여 이전 프레임과 현재 프레임 간의 각 화소들의 평균 이동량를 계산하는 과정과, 계산된 평균 이동량만큼 상기 이전 프레임의 배경 프레임의 배경 이동량을 보상하여, 배경 이동량이 보상된 상기 배경 프레임을 상기 현재 프레

임의 배경 프레임으로 구성하는 과정과, 상기 옵티컬 플로우 방식을 이용하여 상기 현재 프레임과 상기 구성된 배경 프레임 간의 각 화소들의 이동량을 계산하는 과정 및 기 설정된 임계치 이상의 이동량에 해당하는 각 화소들의 좌표값을 계산하여, 검출된 좌표값을 상기 현재 프레임 내의 객체 영역으로 검출하는 과정을 포함한다.

[0008] 본 발명의 다른 일면에 따른 객체 검출 장치는, 이동 환경에서 주변 상황을 촬영하여 이전 프레임과 현재 프레임을 순차적으로 출력하는 영상 획득부와, 옵티컬 플로우 방식을 이용하여 이전 프레임과 현재 프레임 간의 각 화소들의 평균 이동량를 계산하고, 계산된 평균 이동량만큼 상기 이전 프레임의 배경 프레임의 배경 이동량을 보상하여, 배경 이동량이 보상된 상기 배경 프레임을 상기 현재 프레임의 배경 프레임으로 추출하는 배경 추출부 및 상기 옵티컬 플로우 방식을 이용하여 상기 현재 프레임과 상기 보상된 배경 프레임 간의 각 화소들의 이동량을 계산하고, 기 설정된 임계치 이상의 이동량에 해당하는 각 화소들의 좌표값을 계산하여, 검출된 좌표값을 상기 현재 프레임 내의 객체 영역으로 검출하는 객체 검출부를 포함한다.

[0009] 본 발명의 또 다른 일면에 따른 차량용 객체 검출 장치에 있어서, 상기 차량 주행 중에 상기 차량의 주변 상황을 촬영하여 이전 프레임과 현재 프레임을 순차적으로 출력하는 영상 획득부와, 옵티컬 플로우 방식을 이용하여 이전 프레임과 현재 프레임 간의 각 화소들의 평균 이동량를 계산하고, 계산된 평균 이동량만큼 상기 이전 프레임의 배경 프레임의 배경 이동량을 보상하여, 배경 이동량이 보상된 상기 배경 프레임을 상기 현재 프레임의 배경 프레임으로 추출하는 배경 추출부 및 상기 옵티컬 플로우 방식을 이용하여 상기 현재 프레임과 상기 보상된 배경 프레임 간의 각 화소들의 이동량을 계산하고, 상기 차량 내의 전자 제어 유닛으로부터 제공되는 차량의 주행 속도값에 따라 임계치를 결정하고, 결정된 상기 임계치 이상의 이동량에 해당하는 각 화소들의 좌표값을 계산하여, 검출된 좌표값을 상기 현재 프레임 내의 객체 영역으로 검출하는 객체 검출부를 포함한다.

[0010] 본 발명의 또 다른 일면에 따른 차량용 객체 검출 장치를 이용한 객체 검출 방법에 있어서, 상기 차량 주행 중에 상기 차량의 주변 상황을 촬영하여 이전 프레임과 현재 프레임을 순차적으로 출력하는 과정과, 옵티컬 플로우 방식을 이용하여 이전 프레임과 현재 프레임 간의 각 화소들의 평균 이동량를 계산하고, 계산된 평균 이동량만큼 상기 이전 프레임의 배경 프레임의 배경 이동량을 보상하여, 배경 이동량이 보상된 상기 배경 프레임을 상기 현재 프레임의 배경 프레임으로 추출하는 과정 및 상기 옵티컬 플로우 방식을 이용하여 상기 현재 프레임과 상기 보상된 배경 프레임 간의 각 화소들의 이동량을 계산하고, 상기 차량 내의 전자 제어 유닛으로부터 제공되는 차량의 주행 속도값에 따라 임계치를 결정하고, 결정된 상기 임계치 이상의 이동량에 해당하는 각 화소들의 좌표값을 계산하여, 검출된 좌표값을 상기 현재 프레임 내의 객체 영역으로 검출하는 과정을 포함한다.

발명의 효과

[0011] 본 발명에 의하면, 이동하는 카메라에서 획득한 영상에서 배경의 이동량을 계산하고, 계산된 이동량만큼 영상을 보정하여 카메라의 이동에 따른 배경 오차를 보상하여 배경을 추출함으로써, 카메라가 이동하는 환경에서 배경의 움직임이 그대로 반영한 상태로 객체를 추출함에 따른 객체 검출 오차를 줄일 수 있다. 또한 옵티컬 플로우와 배경 분리 방법을 결합함으로써, 움직이는 객체 검출의 정확도를 효율적으로 높일 수 있다.

도면의 간단한 설명

[0012] 도 1은 본 발명의 일 실시 예에 따른 객체 검출 장치의 내부 구성을 개략적으로 보여주는 블록도이다.

도 2는 도 1에 도시된 배경 추출부의 구성을 보여주는 블록도이다.

도 3은 도 1에 도시된 객체 검출부의 구성을 보여주는 블록도이다.

도 4는 도 1에 도시된 객체 검출부의 다른 실시 예를 보여주는 블록도이다.

도 5는 도 1에 도시된 객체 검출부의 또 다른 실시 예를 보여주는 블록도이다.

도 6은 본 발명의 일 실시 예에 따른 객체 검출 방법을 보여주는 순서도이다.

발명을 실시하기 위한 구체적인 내용

[0013] 본 발명은 움직이는 카메라에서 촬영한 영상에서 배경의 이동량을 계산하고,

[0014] 이동량만큼 영상을 움직여 카메라의 움직임을 보상한 배경을 추출함으로써, 배경의 움직임이 그대로 반영되는

종래 기술과는 달리, 움직임이 보상된 배경과 입력 영상을 이용하여 움직이는 객체를 검출하는 방법을 제공한다. 또한 옵티컬 플로우와 배경 분리 방법을 결합하여 움직이는 객체 검출의 정확도를 높이는 방법을 제공한다. 이렇게 함으로써, 카메라의 시점이 이동 중에도 배경의 움직임을 보상하면서 움직이는 객체를 검출 할 수 있으므로, 팬 틸트 기능이 있는 CCTV나 자동차의 전후방카메라 등과 같이 카메라가 이동하는 환경에서 취득한 영상에서 움직이는 객체를 찾아 사용자에게 경고하는 등의 후속 처리를 위한 다양한 기술분야에서 활용될 수 있다.

[0015] 이하, 첨부된 도면을 참조하여 본 발명의 일 실시 예에 대해 상세히 설명하기로 한다. 본 발명의 이점 및 특징, 그리고 그것들을 달성하는 방법은 첨부되는 도면과 함께 상세하게 후술되어 있는 실시 예들을 참조하면 명확해질 것이다. 그러나 본 발명은 이하에서 개시되는 실시 예들에 한정되는 것이 아니라 서로 다른 다양한 형태로 구현될 것이며, 단지 본 실시 예들은 본 발명의 개시가 완전하도록 하며, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에게 발명의 범주를 용이하게 이해할 수 있도록 제공되는 것이며, 본 발명은 청구항의 기재에 의해 정의된다. 한편, 본 명세서에서 사용된 용어는 실시 예들을 설명하기 위한 것이며 본 발명을 제한하고자 하는 것은 아니다. 본 명세서에서, 단수형은 문구에서 특별히 언급하지 않는 한 복수형도 포함한다. 명세서에서 사용되는 "포함한다(comprises)" 또는 "포함하는(comprising)"은 언급된 구성요소, 단계, 동작 및/또는 소자 이외의 하나 이상의 다른 구성요소, 단계, 동작 및/또는 소자의 존재 또는 추가를 배제하지 않는다.

[0016] 도 1은 본 발명의 일 실시 예에 따른 객체 검출 장치의 구성을 개략적으로 보여주는 블록도이다.

[0017] 본 발명의 일 실시 예에 따른 객체 검출 장치는(100)는 옵티컬 플로우 방식을 이용하여 카메라가 움직이는 환경에서 프레임간에서 발생하는 배경 이동량을 보상하고, 옵티컬 플로우 방식을 이용하여 현재 프레임과 보상된 배경 이동량에 따라 구성된 배경 프레임 간의 이동량에 기초해 영상 내의 객체를 검출한다.

[0018] 이를 위해, 본 발명의 일 실시 예에 따른 객체 검출 장치는(100)는 도 1에 도시된 바와 같이, 영상 획득부(110), 프레임 메모리(120), 배경 추출부(130) 및 객체 검출부(140)를 포함한다.

[0019] 영상 획득부(110)는 주변 상황을 촬영하여 이전 영상 프레임(I(n-1): 이하, 이전 프레임이라 한다.)과 현재 영상 프레임(I(n): 이하, 현재 프레임)의 영상을 순차적으로 출력하는 구성으로서, 팬 틸트 기능을 갖는 카메라 또는 차량에 장착되는 전후방 카메라일 수 있으며, 이에 특별히 한정되지 않고, 이동 환경에서 주변 상황을 촬영할 수 있는 모든 종류의 촬영 수단을 포함한다.

[0020] 프레임 메모리(120)는 상기 영상 획득부(110)로부터 현재 프레임(I(n))을 입력받아서 일시적으로 저장하는 일종의 버퍼로서, 상기 현재 프레임(I(n))의 입력에 따라 이전에 저장된 이전 프레임(I(n-1))을 출력한다.

[0021] 배경 추출부(130)는 상기 영상 획득부(110)로부터의 현재 프레임(I(n))과 프레임 메모리(120)로부터의 이전 프레임(I(n-1))을 입력받고, 이전 프레임(I(n-1))과 현재 프레임(I(n)) 간의 옵티컬 플로우를 이용하여 상기 이전 프레임(I(n-1))에 포함된 이전의 배경 영상 프레임(B(n-1): 이하, 이전 배경 프레임이라 한다.)을 보상하고, 보상된 이전의 배경 프레임 (B(n-1)')을 현재의 프레임의 배경 프레임(B(n))으로서 추출(분리 또는 구성)한다. 이에 대한 구체적인 설명은 아래의 도 2를 참조하여 상세히 설명하기로 한다.

[0022] 객체 검출부(140)는 배경 추출부(130)로부터의 보상된 배경 프레임(B(n-1)' 또는 B(n))과 상기 현재 프레임(I(n)) 간의 옵티컬 플로우를 이용하여 상기 현재 프레임(I(n)) 내의 객체 영역을 검출한다. 이에 대한 구체적인 설명은 아래의 도 3 내지 도 5를 참조하여 상세히 설명하기로 한다.

[0023] 먼저, 도 1에 도시된 배경 추출부(130)의 일 실시 예에 대해 상세히 설명한다.

[0024] 도 2는 도 1에 도시된 배경 추출부의 구성을 보여주는 블록도이다.

[0025] 도 2를 참조하면, 배경 추출부(130)은 제1 옵티컬 플로우(OF) 계산부(132), 배경 OF 기준값 생성부(134), 배경 보상부(136) 및 프레임 메모리(138)를 포함한다.

[0026] 제1 OF 계산부(132)는 모든 화소들에 대해 이전 프레임(I(n-1))과 현재 프레임(I(n)) 사이에서의 이동량(옵티컬 플로우값 또는 이동 벡터값)을 계산하고, 계산된 결과를 현재 프레임(I(n))에 대한 제1 OF 프레임(OF1(n))으로서 출력한다. 이때, 출력되는 제1 OF 프레임(OF1(n))는 배경 이동량과 객체 이동량을 모두 포함한 형태로 출력된다. 일례로, 이동량의 계산은 T.Brox에 의해 제안된 "High accuracy optical flow estimation based on a theory for warping" 방법에 따라 계산될 수 있다.

[0027] 배경 OF 기준값 생성부(134)는 제1 OF 프레임(OF1(n))에 포함된 배경 이동량과 객체 이동량 중에서 배경 이동량을 분리(추출 또는 검출)하여 이를 이전 프레임(I(n-1))의 배경 프레임(B(n-1))을 보상하기 위한 기준값(REF)으로 생성한다. 이를 위해, 배경 OF 기준값 생성부(134)는 제1 OF 프레임(OF1(n)) 내의 모든 화소들의 이동량을 크기 순으로 배열하고, 크기 순으로 배열된 이동량을 일정 단위의 구간으로 등분하는 과정을 수행한다. 이후, 등분된 구간들 중 가장 많은 이동량이 분포한 구간을 선정하고, 선정된 상기 구간 내에 분포한 이동량들의 평균치(또는 평균 이동량)를 상기 배경 OF 기준값(RFE)으로 생성한다. 예컨대, 제1 OF 프레임(OF1(n)) 내의 최대 이동량(최대 옵티컬 플로우값 또는 최대 이동 벡터값)과 최소 이동량(최소 옵티컬 플로우값 또는 최소 이동 벡터값) 사이 구간을 100 등분하여 모든 화소들에 대한 이동량을 등분된 구간들로 분류한 후, 가장 많은 개수의 이동량들을 포함하고 있는 구간을 선정하고, 선정된 구간의 중앙값 또는 평균값을 이전 프레임(I(n-1))의 배경 프레임(B(n-1))을 보상하기 위한 기준값(RFE)으로 사용한다. 이를 히스토그램 상에서 살펴보면, 히스토그램 상 최대빈도 구간의 중앙값이다.

[0028] 배경 보상부(136)는 배경 OF 기준값 생성부(134)로부터의 상기 배경 OF 기준값(RFE)을 이용하여 뒷단의 프레임 메모리(138)로부터의 이전 프레임(I(n-1))의 배경 프레임(B(n-1))을 보상하는 구성으로서, 상기 배경 OF 기준값(RFE)인 평균 이동량만큼 상기 이전 프레임(I(n-1))의 배경 프레임(B(n-1)) 내에 존재하는 각 화소들을 이동량을 보상한다. 만일, 프레임 메모리(138)에 이전에 보상된 배경 프레임이 없는 경우, 즉, 최초 프레임 자체를 이전에 구성한(생성한 또는 분리한) 배경 프레임으로 간주한다. 이렇게 보상된 배경 프레임(B(n-1)')을 구성한 배경 보상부(136)는 영상 획득부(110)로부터의 현재 프레임(I(n))을 더 입력 받아서, 상기 보상된 배경 프레임(B(n-1)')의 화소들을 상기 입력 받은 현재 프레임(I(n))의 대응하는 화소들의 화소값으로 대체(맵핑 또는 보상)한다. 이렇게 함으로써, 배경 이동량이 보상된 최종 배경 프레임을 분리 추출한다. 이때, 상기 보상된 배경 프레임(B(n-1)')의 화소들을 현재 프레임(I(n))의 대응하는 화소들의 화소값에 그대로 대체(맵핑 또는 보상)하지 않고, 현재 프레임(I(n))의 대응하는 화소들의 화소값을 사전에 설정한 학습 계수(예컨대, 0.1)만큼 곱한 화소값으로 보상하고, 상기 보상된 배경 프레임(B(n-1)')의 화소들을 상기 학습 계수에 의해 보상된 화소값으로 대체(맵핑 또는 보상)할 수도 있다. 즉, B(n)의 화소값은, 이동시킨 B(n-1) 프레임을 기준으로 특정 위치에 있는 화소값에, I(n)의 해당 위치의 화소값과 이동된 B(n-1)프레임에서의 화소값과의 차이와 학습계수를 곱한 결과값을 더한 값과 같다. 이를 수학식으로 표현하면, 아래의 수학식 1과 같다.

수학식 1

[0029] $$B(n)(x,\ y) = B(n-1)'+(B(n-1)'(x, y)\ -\ I(n)(x, y))\times k$$

[0030] 여기서, k는 학습 계수이다.

[0031] 이하, 상기 배경 추출부(130)에 의해 분리 추출된 배경 프레임을 이용하여 객체 영역을 추출하는 객체 추출부(140)에 대해 상세히 설명하기로 한다.

[0032] 도 3은 도 1에 도시된 객체 검출부의 구성을 보여주는 블록도이다.

[0033] 도 3을 참조하면, 객체 검출부(140)는 제2 OF 계산부(142) 및 객체 영역 분리부(144)를 포함한다.

[0034] 제2 OF 계산부(142)는 상기 배경 추출부(130)로부터의 배경 이동량이 보상된 배경 프레임(B(n))과 상기 영상 획득부(110)로부터의 현재 프레임(I(n))을 입력 받아서, 옵티컬 플로우를 이용하여 모든 화소들에 대해 상기 배경 프레임(B(n))과 현재 프레임(I(n)) 간의 이동량을 계산하여, 상기 현재 프레임(I(n))에 대한 제2 OP 프레임(OF2(n))을 출력한다. 여기서, 제2 OP 프레임(OF2(n))은 앞에서 설명한 제1 OF 계산부(132)로부터 출력되는 제1 OP 프레임(OF2(n))와는 달리 배경 이동량이 보상된 배경 프레임(B(n))을 현재 프레임(I(n))의 배경 프레임으로 간주할 수 있으므로, 현재 프레임(I(n))의 객체를 구성하는 화소들의 이동량이 두드러지게 나타난 형태로 출력될 것이다.

[0035] 객체 영역 분리부(144)는 설계자에 의해 설정된 임계값(TH)을 이용하여 제2 OP 프레임(OF2(n)) 내의 객체 영역을 분리 추출한다. 예컨대, 상기 임계값 이상에 해당하는 각 화소들의 좌표값을 계산하여, 검출된 좌표값을 상기 현재 프레임 내의 객체 영역으로 검출할 수 있다. 이때, 배경 분리 마스크를 이용하여 객체 검출의 정확도를 높일 수 있다. 이에 대한 설명은 아래의 도 4를 참조하여 설명하기로 한다.

[0036] 도 4는 도 1에 도시된 객체 검출부의 다른 실시 예를 보여주는 블록도이다.

[0037] 도 4에 도시된 다른 실시 예에 따른 객체 검출부(140)는 도 3의 실시 예에 마스크 생성부(146)와 필터부(148)를 추가한 점을 제외하면, 도 3의 실시 예와 동일한 동작 및 기능을 수행한다.

[0038] 마스크 생성부(146)는 객체 검출의 정확도를 높이기 위해, 배경 분리 마스크를 생성하는 구성으로서, 현재 프레임(I(n))과 배경 이동량이 보상된 배경 프레임(B(n))을 입력받아서, 이들 프레임들의 차이에 해당하는 마스크 프레임(47)을 생성하고, 이를 상기 배경 분리 마스크로 출력한다.

[0039] 필터부(148)는 상기 마스트 생성부(146)로부터의 마스크 프레임(47)을 이용하여 객체 영역 분리부(144)로부터 출력되는 객체 영역에 포함된 노이즈를 필터링 한다. 이렇게 함으로써, 객체 검출의 정확도를 높일 수 있다.

[0040] 한편, 객체 영역 분리부(144)에 적용되는 임계값(TH)을 카메라의 이동 속도를 고려하여 다양하게 설정함으로써, 객체 검출의 정확도를 높일 수도 있다. 이에 대해서는 도 5를 참조하여 설명하기로 한다.

[0041] 도 5는 도 1에 도시된 객체 검출부의 또 다른 실시 예를 보여주는 블록도이다.

[0042] 도 5에서는 본 발명의 객체 검출 장치(100)가 차량에 탑재된 경우, 차량의 주행 속도 즉, 차량에 장착된 카메라의 이동 속도에 따라 임계치(TH)가 조정되는 예를 설명한 것이다. 본 발명의 객체 검출 장치(100)가 차량에 적용되는 경우, 본 발명의 또 다른 실시 예에 따른 객체 검출부(140)는 CAN 통신 또는 LIN 통신과 같은 차량 내의 네트워크 통신(40)을 통해 차량 내의 전자 제어 유닛(200)으로부터 차량의 주행 속도를 제공받을 수 있다.

[0043] 상기 전자 제어 유닛(200)으로부터 차량이 주행 속도를 제공받는 객체 검출부(140)는 도 3에 도시된 구성들에 추가로 임계값 생성부(149)를 더 포함할 수 있다. 이러한 임계값 생성부(149)는 설계자에 의해 설정된 기준 속도(예컨대, 10 Km/h)와 전자 제어 유닛(200)로부터 수신된 차량의 주행 속도를 비교하고, 차량의 주행 속도가 상기 기준속도보다 낮으며, 제1 임계치(TH1)를 상기 객체 영역 분리부(144)로 출력하고, 반대로 차량의 주행 속도가 상기 기준속도보다 높으면, 상기 제1 임계치(TH1)보다 큰 제2 임계치(TH2)를 상기 객체 영역 분리부(144)으로 출력한다. 그러면, 객체 영역 분리부(144)는 현재의 차량 주행 속도 즉, 카메라의 이동 속도에 따라 임계치를 조절하여 제2 OF 프레임(OF2(n)) 내이 객체 영역을 적응적으로 검출할 수 있다. 차량의 속도가 높은 경우, 즉, 카메라의 이동속도가 높은 환경에서 획득된 프레임들의 경우, 그 인접한 프레임들 간의 옵티컬 플로우의 이동량 편차는 크므로, 이런 경우에, 낮은 임계치를 적용하게 되면, 정확한 객체 검출이 어렵다. 반대로 카메라의 이동속도가 낮은 환경에서 획득된 프레임들의 경우, 인접한 프레임들 간의 옵티컬 플로우의 이동량 편차는 작으므로, 이런 경우에 높은 임계치를 적용하게 되면, 마찬가지로 정확한 객체 검출이 어렵다. 그러나, 본 발명에서는 위와 같이, 카메라의 이동속도가 높은 경우에는 높은 임계치를 적용하고, 반대의 경우에는 낮은 임계치를 적용함으로써, 객체 검출의 정확도를 높일 수 있다.

[0044] 본 실시 예에서는, 객체 검출의 정확도를 높이기 위한 방안으로, 도 4와 도 5의 2가지 실시 예를 제시하고 있지만, 이들 실시 예들의 결합을 통해 객체 검출의 정확도를 더욱 높일 수 도 있다. 이러한 결합된 실시 예는 당업자라면 도 4와 도 5의 설명을 통해 충분히 구현할 수 있는 것이므로, 이에 대한 구체적인 설명은 생략하기로 한다.

[0045] 도 6은 본 발명의 일 실시 예에 따른 객체 검출 방법을 보여주는 순서도로서, 도 6에 도시된 각 단계의 수행 주체는 도 1 내지 도 5의 설명을 통해 명확히 이해될 수 있으므로, 아래의 설명에서는 각 단계의 수행 주체를 언급하지 않는다.

[0046] 도 6을 참조하면, 먼저, S610에서, 옵티컬 플로우 방식을 이용하여 모든 화소들에 대한 이전 프레임과 현재 프레임 사이에서의 평균 이동량을 계산하는 과정이 수행된다. 평균 이동량을 계산하는 과정은 이전 프레임과 현재 프레임 간의 각 화소들의 이동량를 계산하여, 상기 각 화소들의 이동량을 크기 순으로 배열하는 과정과, 크기 순으로 배열된 이동량을 일정 단위의 구간으로 등분하고, 등분된 구간들 중 가장 많은 이동량이 분포한 구간을 선정하는 과정 및 선정된 상기 구간 내에 분포한 이동량들의 평균치를 상기 평균 이동량으로 계산하는 과정을 포함한다.

[0047] 이어, S620에서, 계산된 평균 이동량만큼 상기 이전 프레임의 배경 프레임의 배경 이동량을 보상하여, 배경 이동량이 보상된 상기 배경 프레임을 상기 현재 프레임의 배경 프레임으로 구성하는 과정이 수행된다. 이 과정은 상기 현재 프레임의 배경 프레임 내의 각 화소들의 화소값을 추출하는 과정과, 추출된 상기 화소값과, 상기 화소값의 위치에 해당하는 상기 보상된 배경 프레임의 화소값의 차이에 기 설정된 학습 계수를 연산하는 과정 및 상기 연산 결과에 따른 화소값을 상기 보상된 배경 프레임 내의 각 화소들의 화소값으로 대체하는 과정을 포함

한다.

[0048] 이어, S630에서, 상기 화소값으로 대체된 배경 프레임을 현재 프레임 내의 배경 프레임의 구성하는 과정이 수행된다.

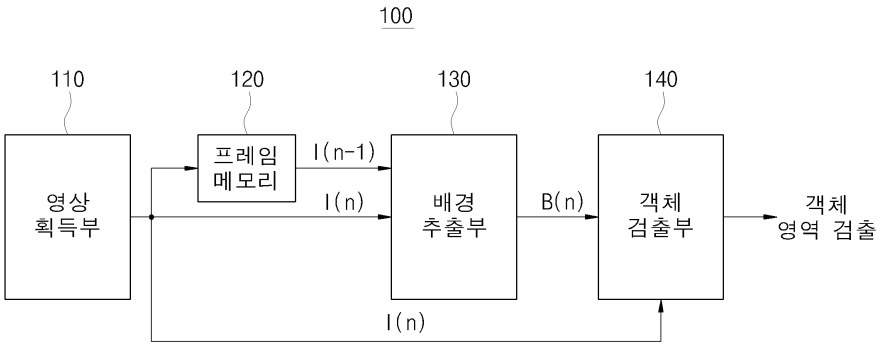
[0049] 이어, S640에서, 상기 옵티컬 플로우 방식을 이용하여 상기 현재 프레임과 상기 구성된 배경 프레임 간의 각 화소들의 이동량을 계산하는 과정이 수행된 후, S650에서, 기 설정된 임계치 이상의 이동량에 해당하는 각 화소들의 좌표값을 계산하여, 검출된 좌표값을 상기 현재 프레임 내의 객체 영역으로 검출하는 과정이 수행된다. 이때, 객체 영역의 검출 정확도를 높이기 위해, 상기 보상된 배경 프레임과 현재 프레임 간이 차이에 해당하는 마스크 프레임을 생성하는 과정과, 상기 검출된 객체 영역을 상기 마스크 프레임을 이용하여 필터링하는 과정 및 필터링된 상기 객체 영역을 상기 현재 프레임 내의 최종 객체 영역으로 검출하는 과정이 더 수행될 수 있다. 선택적으로, 카메라의 이동 속도값에 따라 상기 임계치를 적응적으로 조절하는 과정이 더 수행될 수 있다. 예컨대, 차량에 탑재된 카메라의 경우, 차량의 주행 속도가 상기 기준속도보다 낮으며, 제1 임계치(TH1) 이상의 이동량에 해당하는 각 화소들의 좌표값을 객체 영역으로 검출하고, 차량의 주행 속도가 상기 기준속도보다 높으면, 상기 제1 임계치(TH1)보다 큰 제2 임계치(TH2) 이상이 이동량에 해당하는 각 화소들의 좌표값을 객체 영역으로 검출할 수 있다.

[0050] 이상 설명한 바와 같이, 본 발명은 옵티컬 플로우를 이용한 배경 분리 방법을 제공함으로써, 영상에서 두 개의 연속적인 프레임을 입력 받아 옵티컬 플로우를 통해 영상 화소 전체의 이동량(또는, 이동 벡터)를 계산하고, 계산 결과 동일한 이동량의 개수를 기준으로 예컨대, 7할 이상을 차지하는 이동량을 이전 프레임에 역으로 적용하여 배경의 움직임을 보상한다. 그러면, 보상된 이전 프레임과 현재 프레임을 이용하여 통해 배경 프레임을 분리한다. 분리된 배경 프레임과 현재 프레임을 다시 옵티컬 플로우를 통해 이동하는 객체들에 대한 군집화된 이동량(벡터)를 얻는다. 군집의 개수에 따라 선형적으로 이동체를 구분하는 이동량의 기준값(벡터 경계값)을 자동 결정하여 이동량이 가장 큰 소수의 군집들만 결정한다. 결정된 군집들에 대해 배경 분리 마스크와 같은 이미지 노이즈 저감 필터들을 적용하고, 배경 프레임과 현재 프레임을 배경 분리 방법을 통해 얻은 배경 분리 마스크를 결정된 군집에 적용하여 정확도를 높인다. 마스크 프레임까지 통과한 군집에 한해 군집들의 이동량(벡터) 위치를 원본 영상의 좌표계로 변환하여 출력함으로써, 최종 객체 영역을 검출한다.

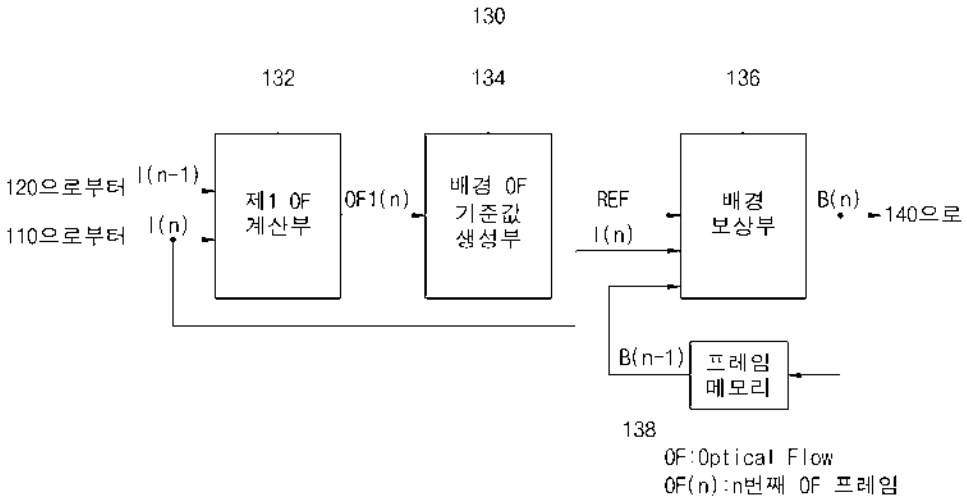
[0051] 이제까지 본 발명의 실시 예들을 중심으로 살펴보았다. 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자는 본 발명이 본 발명의 본질적인 특성에서 벗어나지 않는 범위에서 변형된 형태로 구현될 수 있음을 이해할 수 있을 것이다. 그러므로 개시된 실시 예들은 한정적인 관점이 아니라 설명적인 관점에서 고려되어야 한다. 본 발명의 범위는 전술한 설명이 아니라 특허청구범위에 나타나 있으며, 그와 동등한 범위 내에 있는 모든 차이점은 본 발명에 포함된 것으로 해석되어야 할 것이다.

도면

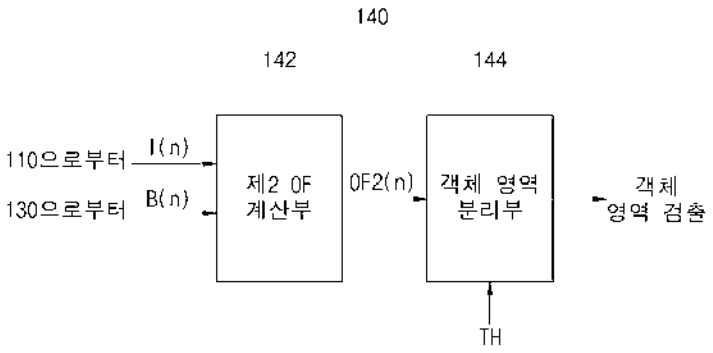
도면1



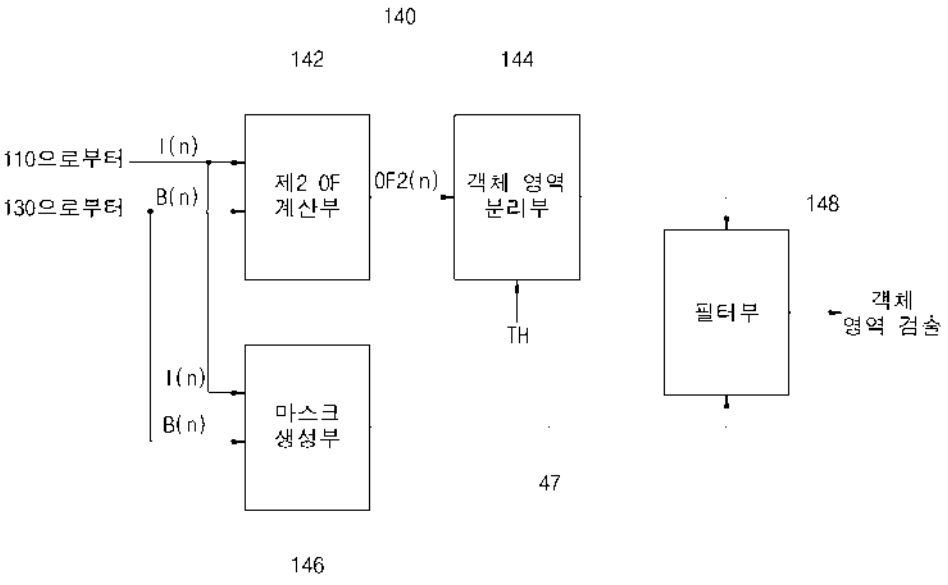
도면2


130
132
134
136
120으로부터
110으로부터
I(n-1)
I(n)
제1 OF
계산부
OF1(n)
배경 OF
기준값
생성부
REF
I(n)
배경
보상부
B(n)
140으로
B(n-1)
프레임
메모리
138
OF:Optical Flow
OF(n):n번째 OF 프레임

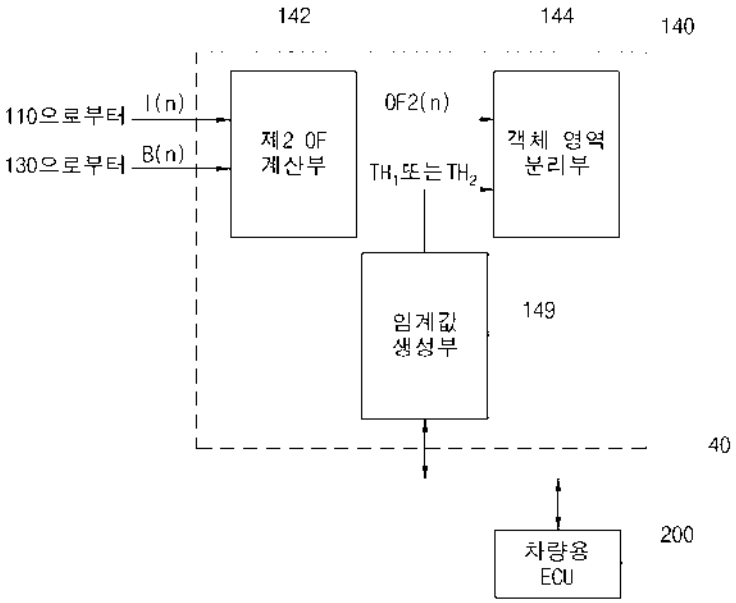
도면3


140
142
144
110으로부터
130으로부터
I(n)
B(n)
제2 OF
계산부
OF2(n)
객체 영역
분리부
객체
영역 검출
TH

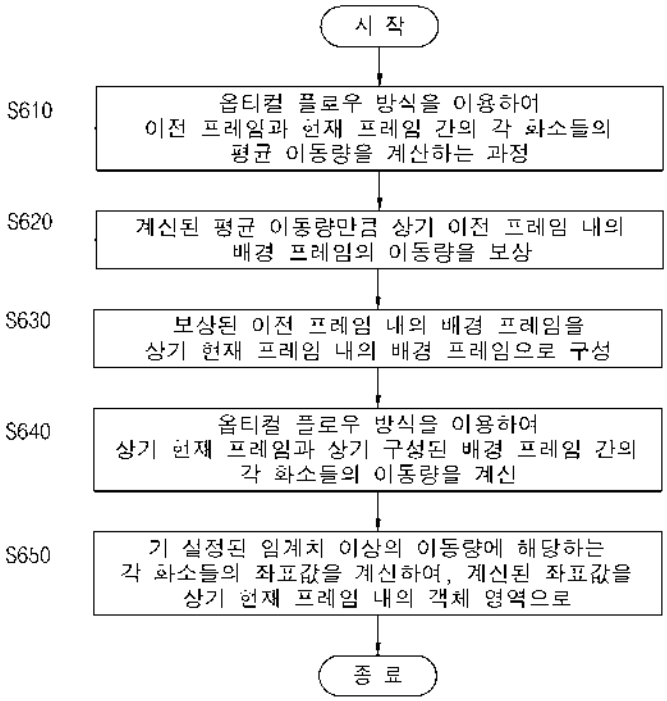
도면4


140
142
144
110으로부터
130으로부터
I(n)
B(n)
제2 OF
계산부
OF2(n)
객체 영역
분리부
148
필터부
객체
영역 검출
TH
I(n)
B(n)
마스크
생성부
47
146

도면5


142
144
140
110으로부터 I(n)
130으로부터 B(n)
제2 OF 계산부
OF2(n)
TH1또는TH2
객체 영역 분리부
임계값 생성부
149
40
200
차량용 ECU

도면6


시 작
S610
옵티컬 플로우 방식을 이용하여 이전 프레임과 현재 프레임 간의 각 화소들의 평균 이동량을 계산하는 과정
S620
계산된 평균 이동량만큼 상기 이전 프레임 내의 배경 프레임의 이동량을 보상
S630
보상된 이전 프레임 내의 배경 프레임을 상기 현재 프레임 내의 배경 프레임으로 구성
S640
옵티컬 플로우 방식을 이용하여 상기 현재 프레임과 상기 구성된 배경 프레임 간의 각 화소들의 이동량을 계산
S650
기 설정된 임계치 이상의 이동량에 해당하는 각 화소들의 좌표값을 계산하여, 계산된 좌표값을 상기 현재 프레임 내의 객체 영역으로
종 료

(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2016년03월15일
(11) 등록번호 10-1603711
(24) 등록일자 2016년03월09일

(51) 국제특허분류(Int. Cl.)
G06F 9/50 (2006.01) *G06F 13/14* (2006.01)
(21) 출원번호 10-2014-0049973
(22) 출원일자 2014년04월25일
심사청구일자 2014년04월25일
(65) 공개번호 10-2015-0123519
(43) 공개일자 2015년11월04일
(56) 선행기술조사문헌
KR1020120009919 A*
KR1020140006351 A*
KR101471303 B1
KR1020120031759 A
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
전자부품연구원
경기도 성남시 분당구 새나리로 25 (야탑동)
(72) 발명자
황태호
서울특별시 송파구 동남로 225 래미안파크팰리스 108동 601호
김동순
경기도 성남시 분당구 정자일로 248 파크뷰 606동 3203호
김선욱
경기도 남양주시 도농로 34 부영아파트 214동 2003호
(74) 대리인
특허법인지명

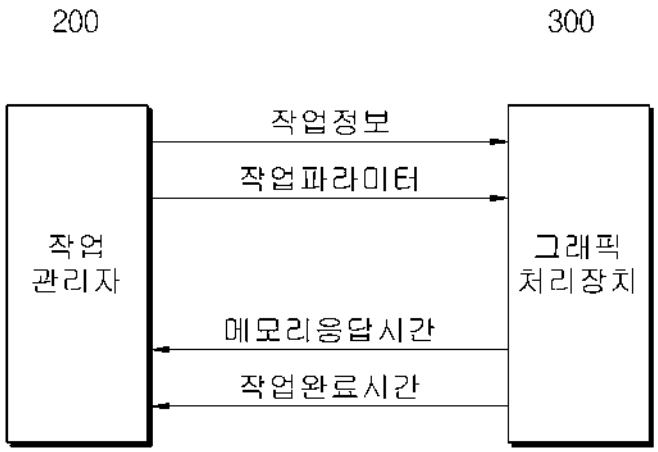
전체 청구항 수 : 총 8 항 심사관 : 유진태

(54) 발명의 명칭 **그래픽 처리 장치의 동작을 위한 작업 할당 시스템 및 방법**

(57) 요 약

본 발명은 그래픽 처리 장치(GPU)에 작업을 할당하고 그래픽 처리 장치가 할당받은 작업을 처리함에 있어서, 그래픽 처리 장치로부터 수신한 메모리 응답 시간에 기초하여 그래픽 처리 장치의 최적화된 코어 수를 조절하는 그래픽 처리 장치의 동작을 위한 작업 할당 시스템 및 방법을 제공한다. 본 발명에 따르면, 그래픽 처리 장치에서 최적화된 수의 코어가 작동하도록 함으로써 그래픽 처리 장치의 작업 처리 속도는 유지하면서 메모리 병목 현상으로 인한 작업 처리 지연을 감소시킬 수 있도록 한다.

대 표 도 - 도2



이 발명을 지원한 국가연구개발사업
과제고유번호 10041664
부처명 산업통산자원부
연구관리전문기관 산업기술평가관리원
연구사업명 산업기술원천기술개발사업
연구과제명 멀티 Shader GPU 통합형 멀티 코어 퓨전 프로세서 원천 기술 개발
기 여 율 1/1
주관기관 전자부품연구원
연구기간 2013.06.01 ~ 2014.05.31

명 세 서

청구범위

청구항 1

복수의 코어를 포함하며 중앙처리장치가 요청한 작업을 처리하는 그래픽처리장치; 및

상기 중앙처리장치가 요청한 작업을 상기 그래픽처리장치에 포함된 코어에 할당하고, 상기 그래픽처리장치로부터 일정 시간 간격으로 메모리 응답 시간 정보를 수신하며, 수신한 메모리 응답 시간 정보에 기초하여 상기 그래픽처리장치의 목표 코어 수를 지정하는 작업관리자를 포함하되,

상기 작업관리자는 상기 지정된 그래픽처리장치의 목표 코어 수가 상기 그래픽처리장치의 작동 코어 수보다 크면 상기 목표 코어 수와 상기 작동 코어 수의 차이에 해당하는 수의 작업이 할당되지 않은 코어에 작업을 할당하는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 시스템.

청구항 2

제1항에 있어서, 상기 작업관리자는

상기 수신한 메모리 응답 시간 정보의 개수가 기설정된 개수 이상이 되면 수신한 메모리 응답 시간의 평균을 산출하고, 산출된 평균에 기초하여 상기 그래픽처리장치의 목표 코어 수를 지정하는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 시스템.

청구항 3

제2항에 있어서, 상기 작업관리자는

상기 산출된 평균이 제1 임계값보다 크면 상기 그래픽처리장치의 목표 코어 수를 감소시키고, 상기 산출된 평균이 제2 임계값보다 작으면 상기 그래픽처리장치의 목표 코어 수를 증가시키는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 시스템.

청구항 4

삭제

청구항 5

제1항에 있어서, 상기 작업관리자는

상기 지정된 그래픽처리장치의 목표 코어 수가 상기 그래픽처리장치의 작동 코어 수보다 작으면 작동 중인 코어 중 상기 목표 코어 수와 상기 작동 코어 수의 차이에 해당하는 수의 코어를 작업 할당에서 제외하는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 시스템.

청구항 6

그래픽처리장치의 동작을 위한 작업관리자의 작업 할당 방법에 있어서,

복수의 코어를 포함하는 그래픽처리장치에 작업을 할당하는 단계;

상기 그래픽처리장치로부터 일정 시간 간격으로 메모리 응답 시간에 대한 정보를 수신하는 단계;

상기 수신한 메모리 응답 시간에 대한 정보에 기초하여 상기 그래픽처리장치의 목표 코어 수를 지정하는 단계; 및

상기 지정된 목표 코어 수에 해당하는 수의 코어에 작업이 할당되도록 제어하는 단계

를 포함하되,

상기 지정된 목표 코어 수에 해당하는 수의 코어에 작업이 할당되도록 제어하는 단계는,

상기 지정된 목표 코어 수가 작동 중인 코어 수보다 크면 작업이 할당되지 않은 코어에 작업을 할당하는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 방법.

청구항 7

제6항에 있어서, 상기 수신한 메모리 응답 시간에 대한 정보에 기초하여 상기 그래픽처리장치의 목표 코어 수를 지정하는 단계는

상기 수신한 메모리 응답 시간의 평균을 산출하는 단계; 및

상기 평균을 기설정된 임계값과 비교하고, 비교 결과에 기초하여 목표 코어 수를 지정하는 단계를 포함하는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 방법.

청구항 8

삭제

청구항 9

제6항에 있어서, 상기 지정된 목표 코어 수에 해당하는 수의 코어에 작업이 할당되도록 제어하는 단계는

상기 지정된 목표 코어 수가 작동 중인 코어 수보다 작으면 작업이 할당된 코어 중 할당된 작업이 적은 순서대로 코어를 작업 할당에서 제외하는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 방법.

청구항 10

제6항에 있어서, 상기 지정된 목표 코어 수에 해당하는 수의 코어에 작업이 할당되도록 제어하는 단계는

상기 지정된 목표 코어 수가 작동 중인 코어 수와 동일하면 작업이 가장 적게 할당된 코어에 작업을 할당하거나 작업 완료 신호를 전송한 코어에 작업을 할당하는 것

인 그래픽 처리 장치의 동작을 위한 작업 할당 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 그래픽 처리 장치(GPU, Graphics Processing Unit)의 작동을 제어하는 시스템 및 방법에 관한 것으로서, 구체적으로는, 중앙 처리 장치가 요청한 작업을 그래픽 처리 장치에 효율적으로 할당하는 그래픽 처리 장치의 동작을 위한 작업 할당 시스템 및 그 방법에 관한 것이다.

배 경 기 술

[0002] 그래픽 처리 장치(GPU, Graphics Processing Unit)는 컴퓨터에서 그래픽 연산 처리를 전담하는 반도체 코어 칩 또는 장치를 의미하는 것으로서, 현재 가장 널리 쓰이고 있는 NVIDIA 사의 Fermi GPU 구조에서는 GPU 자원을 최대한 활용하기 위해 모든 GPU 코어에 가능한 한 많은 작업을 할당하고 있다. 즉, 종래의 GPU 구조에서는 모든 코어에 최대 개수의 작업을 할당함으로써 GPU의 TLP(Thread Level Parallelism)을 최대로 끌어내고자 한다.

[0003] 종래의 GPU 구조에서의 이러한 작업 할당 정책은 모든 코어를 최대한 활용하는 것으로서, 이는 메모리 접근이 빈번하지 않은 GPU 프로그램을 수행할 때는 좋은 효과를 보여주나 메모리 접근이 빈번한 프로그램을 수행할 때는 메모리의 병목 현상으로 인하여 좋지 못한 성능을 보인다.

[0004] 즉, 메모리 접근 작업이 많은 GPU 프로그램을 수행할 경우 모든 코어에서 많은 수의 메모리 요청이 동시에 일어나므로 메모리에 병목 현상이 일어나게 되고, 메모리 병목현상으로 인해 GPU 코어는 요청한 데이터가 도착할 때까지 프로그램을 진행할 수 없으며, 이러한 지연(stall) 현상은 GPU 코어의 개수가 늘어날수록 증가한다. 이는 결국 개별 GPU 코어 성능을 크게 떨어뜨리고 개별 GPU 코어를 효율적으로 사용하지 못하게 하므로 전체적인 전력 소모량이 늘어나게 하는 문제점이 존재한다.

발명의 내용

해결하려는 과제

[0005] 본 발명은 전술한 문제점을 해결하기 위하여, 메모리 접근이 빈번한 프로그램에서 메모리 응답 시간(Memory Latency)을 기준으로 최적화된 GPU 코어 개수를 계산하여 활용함으로써 메모리 병목 현상을 줄이고 개별 GPU 코어의 성능을 높이는 그래픽 처리 장치의 동작을 위한 작업 할당 시스템 및 방법을 제공하는 것을 목적으로 한다.

과제의 해결 수단

[0006] 본 발명은 복수의 코어를 포함하며 중앙처리장치가 요청한 작업을 처리하는 그래픽처리장치; 및 상기 중앙처리장치가 요청한 작업을 상기 그래픽처리장치에 포함된 코어에 할당하고, 상기 그래픽처리장치로부터 일정 시간 간격으로 메모리 응답 시간 정보를 수신하며, 수신한 메모리 응답 시간 정보에 기초하여 상기 그래픽처리장치의 목표 코어 수를 지정하는 작업관리자를 포함하는 그래픽 처리 장치의 동작을 위한 작업 할당 시스템을 제공한다.

[0007] 상기 작업관리자는 상기 수신한 메모리 응답 시간 정보의 개수가 기설정된 개수 이상이 되면 수신한 메모리 응답 시간의 평균을 산출하고, 산출된 평균에 기초하여 상기 그래픽처리장치의 목표 코어 수를 지정한다.

[0008] 상기 작업관리자는 상기 산출된 평균이 제1 임계값보다 크면 상기 그래픽처리장치의 목표 코어 수를 감소시키고, 상기 산출된 평균이 제2 임계값보다 작으면 상기 그래픽처리장치의 목표 코어 수를 증가시킨다.

[0009] 상기 작업관리자는 상기 지정된 그래픽처리장치의 목표 코어 수가 상기 그래픽처리장치의 작동 코어 수보다 크면 상기 목표 코어 수와 상기 작동 코어 수의 차이에 해당하는 수의 작업이 할당되지 않은 코어에 작업을 할당한다.

[0010] 상기 작업관리자는 상기 지정된 그래픽처리장치의 목표 코어 수가 상기 그래픽처리장치의 작동 코어 수보다 작으면 작동 중인 코어 중 상기 목표 코어 수와 상기 작동 코어 수의 차이에 해당하는 수의 코어를 작업 할당에서 제외한다.

[0011] 본 발명의 다른 일면에 따르면, 복수의 코어를 포함하는 그래픽처리장치에 작업을 할당하는 단계; 상기 그래픽처리장치로부터 일정 시간 간격으로 메모리 응답 시간에 대한 정보를 수신하는 단계; 상기 수신한 메모리 응답 시간에 대한 정보에 기초하여 상기 그래픽처리장치의 목표 코어 수를 지정하는 단계; 및 상기 지정된 목표 코어 수에 해당하는 수의 코어에 작업이 할당되도록 제어하는 단계를 포함하는 그래픽 처리 장치의 동작을 위한 작업 할당 방법을 제공한다.

발명의 효과

[0012] 본 발명은 최적화된 개수의 GPU 코어만을 활용하여 메모리 병목 현상을 줄이고 개별 GPU 코어의 성능을 향상시킬 수 있도록 한다. 따라서 향상된 개별 GPU 코어 성능으로 더 적은 수의 GPU 코어를 사용하면서도 종래기술과 같은 수준을 실행 속도를 유지하도록 하고 필요없는 GPU 코어는 사용하지 않음으로써 종래기술에 비해 전력 소모량을 감소시킬 수 있도록 한다.

도면의 간단한 설명

[0013] 도 1은 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 시스템의 전체적인 구성을 나타낸 도면.

도 2는 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 시스템의 작업관리자와 그래픽 처리 장치의 인터페이스를 나타낸 도면.

도 3과 도 4는 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 방법의 과정을 나타낸 도면.

발명을 실시하기 위한 구체적인 내용

[0014] 본 발명의 이점 및 특징, 그리고 그것들을 달성하는 방법은 첨부되는 도면과 함께 상세하게 후술 되어 있는 실시예들을 참조하면 명확해질 것이다. 그러나 본 발명은 이하에서 개시되는 실시예들에 한정되는 것이 아니라 서로 다른 다양한 형태로 구현될 것이며, 단지 본 실시예들은 본 발명의 개시가 완전하도록 하며, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에게 발명의 범주를 완전하게 알려주기 위해 제공되는 것이며, 본 발명은 청구항의 기재에 의해 정의된다.

[0015] 한편, 본 명세서에서 사용된 용어는 실시예들을 설명하기 위한 것이며 본 발명을 제한하고자 하는 것은 아니다. 본 명세서에서, 단수형은 문구에서 특별히 언급하지 않는 한 복수형도 포함한다. 명세서에서 사용되는 "포함한다(comprises)" 및/또는 "포함하는(comprising)"은 언급된 구성소자, 단계, 동작 및/또는 소자에 하나 이상의 다른 구성소자, 단계, 동작 및/또는 소자의 존재 또는 추가함을 배제하지 않는다. 이하, 첨부된 도면을 참조하여 본 발명의 실시예를 상세히 설명하기로 한다.

[0016] 도 1은 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 시스템의 전체적인 구성을 나타낸 것이다.

[0017] 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 시스템은 도 1에 도시된 바와 같이 중앙 처리 장치(100), 그래픽 처리 장치(300) 및 DRAM(400)을 포함하며, 중앙 처리 장치(100)와 그래픽 처리 장치(300)의 사이에 작업 실행을 전반적으로 제어하는 작업관리자(200)를 포함한다.

[0018] 작업관리자(200)는 중앙 처리 장치(100)로부터 작업 처리를 위임받고 위임받은 작업을 그래픽 처리 장치(300)의 코어에 할당하며 그래픽 처리 장치(300)의 구동을 제어한다. 작업관리자(200)는 그래픽 처리 장치(300)의 연산 자원을 관리하는 기능을 탑재하고 있으며, 이를 이용하여 본 발명이 제안하는 그래픽 처리 장치(300)의 코어 개수 조절을 수행한다.

[0019] 도 2는 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 시스템의 작업관리자(200)와 그래픽 처리 장치(300) 간의 인터페이스를 나타낸 것이다.

[0020] 작업관리자(200)는 중앙 처리 장치(100)로부터 위임받은 작업정보를 그래픽 처리 장치(300)에 전달하고, 그래픽 처리 장치(300)의 각 코어에 작업을 할당한다. 그리고 할당한 작업의 파라미터를 그래픽 처리 장치(300)로 전달한다.

[0021] 그래픽 처리 장치(300)는 작업관리자(200)로부터 할당받은 작업을 처리하고, 할당받은 작업의 처리가 완료되면 작업관리자(200)에게 작업 완료 신호를 전송한다. 또한, 그래픽 처리 장치(300)는 작업관리자(200)로부터 할당받은 작업을 수행하면서 일정 주기마다 메모리 응답 시간(Memory Latency)에 대한 정보를 작업관리자(200)에게 전달한다.

[0022] 작업관리자(200)는 그래픽 처리 장치(300)의 각 코어에서 전달받은 메모리 응답 시간을 일정 개수만큼 저장하고 저장된 메모리 응답 시간의 평균(AML, Average Memory Latency)을 계산한다. 작업관리자(200)는 계산된 평균 메모리 응답 시간(AML)에 기초하여 그래픽 처리 장치(300)의 작동 코어 개수를 조절하여 그래픽 처리 장치(300)의 코어가 최적화된 개수만큼 작동하도록 하며, 작업관리자(200)가 그래픽 처리 장치(300)의 작동 코어 개수를 조절하는 과정은 도 3과 도 4를 통해 구체적으로 설명한다.

[0023] 도 3과 도 4는 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 방법의 과정을 나타낸 것으로서, 도 3은 작업관리자가 최적화된 목표 코어 수를 지정하는 과정을 나타낸 것이고 도 4는 최적화된 목표 코어 수와 현재 작동 코어 수에 따라 그래픽 처리 장치에 작업 할당을 제어하는 과정을 나타낸 것이다.

[0024] 도 3에 도시된 바와 같이, 본 발명의 일실시예에 따른 그래픽 처리 장치의 동작을 위한 작업 할당 시스템의 작업관리자는 중앙 처리 장치로 위임받은 작업을 그래픽 처리 장치의 각 코어에 할당하고, 일정 주기마다 그래픽 처리 장치로부터 각 코어의 메모리 응답 시간을 전달받는다.

[0025] 작업관리자는 그래픽 처리 장치의 각 코어에서 전달받은 메모리 응답 시간을 가지고 전체 그래픽 처리 장치에서의 평균 메모리 응답 시간(AML)을 계산한다(S300). 이때 작업관리자는 그래픽 처리 장치로부터 전달받은 메모리 응답 시간의 개수가 일정 개수 이상이 되면 평균 메모리 응답 시간(AML)을 계산할 수도 있다.

[0026] 작업관리자는 계산된 평균 메모리 응답 시간(AML)을 미리 지정된 응답 시간의 임계값과 비교하고 비교 결과에 따라 목표 코어 수를 조정하여 최적화된 개수의 코어가 작동할 수 있도록 한다.

[0027] 구체적으로, 작업관리자는 평균 메모리 응답 시간(AML)을 응답 시간의 최대 임계값인 제1 임계값과 비교하고(S320), 평균 메모리 응답 시간(AML)이 제1 임계값보다 크면 목표 코어 수를 감소시킨다(S340).

[0028] 그리고 평균 메모리 응답 시간(AML)이 제1 임계값보다 크지 않으면 평균 메모리 응답 시간(AML)을 응답 시간의 최소 임계값인 제2 임계값과 비교하고(S360), 평균 메모리 응답 시간(AML)이 제2 임계값보다 작으면 목표 코어 수를 증가시킨다(S380).

[0029] 평균 메모리 응답 시간(AML)이 제1임계값보다 크지 않고 제2 임계값보다 작지 않으면 목표 코어 수는 현재 목표 코어 수로 유지한다.

[0030] 즉, 본 발명은 평균 메모리 응답 시간(AML)을 미리 지정된 응답 시간의 임계값과 비교하고 비교 결과에 따라 목표 코어 수를 조정함으로써 그래픽 처리 장치의 코어가 최적화된 개수만큼 작동할 수 있도록 하며, 목표 코어 수 결정을 위한 제1 임계값과 제2 임계값은 실험적으로 구해질 수 있고 사용자에 의하여 임의로 설정될 수도 있다.

[0031] 도 4는 최적화된 목표 코어 수에 따라 그래픽 처리 장치의 코어에 작업을 할당하는 과정을 나타낸 것이다.

[0032] 작업관리자는 목표 코어 수가 지정되면 지정된 목표 코어 수를 현재 작동 중인 그래픽 처리 장치의 코어 수와 비교한다(S400).

[0033] 목표 코어 수가 현재 작동 중인 코어 수보다 크면(S410), 작업이 할당되지 않은 코어에 작업을 할당하여(S420) 작동 중인 코어 수가 최적화된 목표 코어 수와 동일하게 하거나 목표 코어 수에 근접하도록 한다.

[0034] 그리고 목표 코어 수가 현재 작동하는 코어 수보다 작으면(S430), 현재 작동하고 있는 코어 중 작업이 가장 적게 할당된 코어를 선택하고 선택된 코어를 작업 할당에서 제외한다(S440). 따라서 작업이 가장 적게 할당된 코어에는 추가적인 작업 할당을 하지 않고 해당 코어가 현재 할당된 작업만을 완료하고 작동을 중지하도록 한다.

[0035] 이때 목표 코어 수와 작동 중인 코어 수의 차이에 해당하는 수만큼의 코어를 작업이 적게 할당된 순서대로 선택하고 선택된 코어들을 작업 할당에서 제외할 수도 있다.

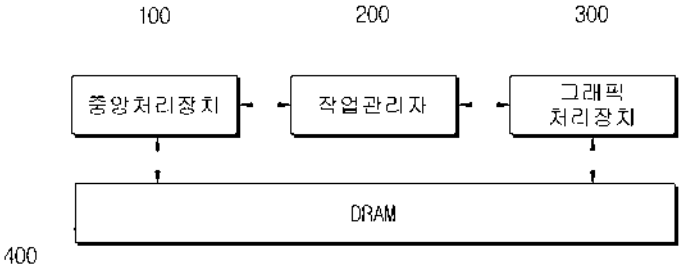
[0036] 현재 작동하고 있는 코어 수와 목표 코어 수가 동일하면 현재 작동 중인 코어 중에서 가장 적은 작업을 할당받은 코어를 지정한다(S450). 그리고 지정된 코어에 작업 할당이 바로 가능하다면(S460) 작업을 바로 할당한다. 지정된 코어에 이미 너무 많은 작업이 할당되어 있어 추가적인 작업 할당이 가능하지 않으면(460) 그래픽 처리 장치의 코어로부터 작업 완료 신호를 수신할 때까지 대기하고(S480), 작업 완료 신호를 수신하면 작업을 할당한다.

[0037] 이상의 설명은 본 발명의 기술적 사상을 예시적으로 설명한 것에 불과한 것으로서, 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자라면, 본 발명의 본질적 특성을 벗어나지 않는 범위에서 다양한 수정 및 변형이 가

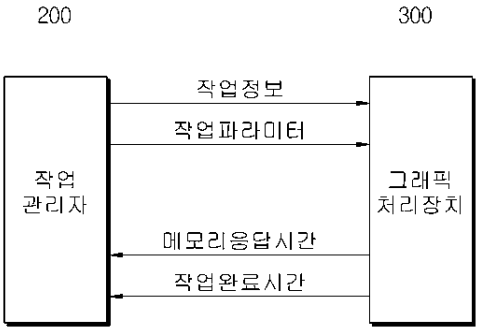
능하다. 따라서, 본 발명에 표현된 실시예들은 본 발명의 기술적 사상을 한정하는 것이 아니라, 설명하기 위한 것이고, 이러한 실시예에 의하여 본 발명의 권리범위가 한정되는 것은 아니다. 본 발명의 보호 범위는 아래의 특허청구범위에 의하여 해석되어야 하고, 그와 동등하거나, 균등한 범위 내에 있는 모든 기술적 사상은 본 발명의 권리범위에 포함되는 것으로 해석되어야 할 것이다.

도면

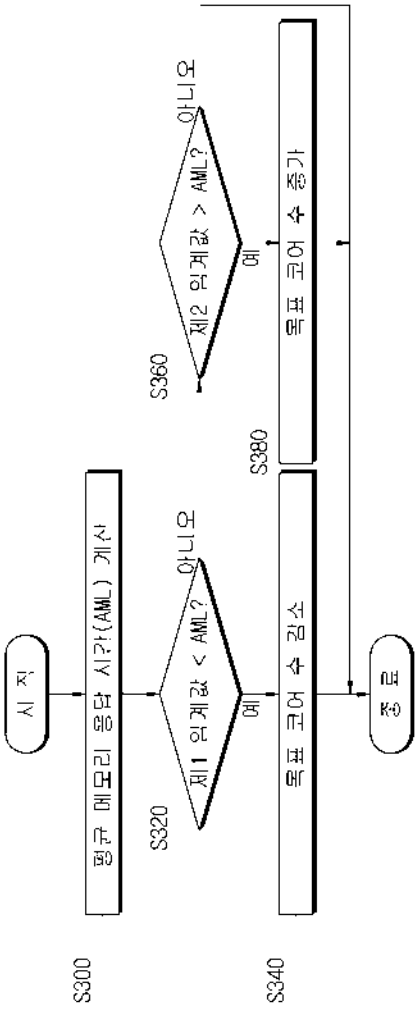
도면1



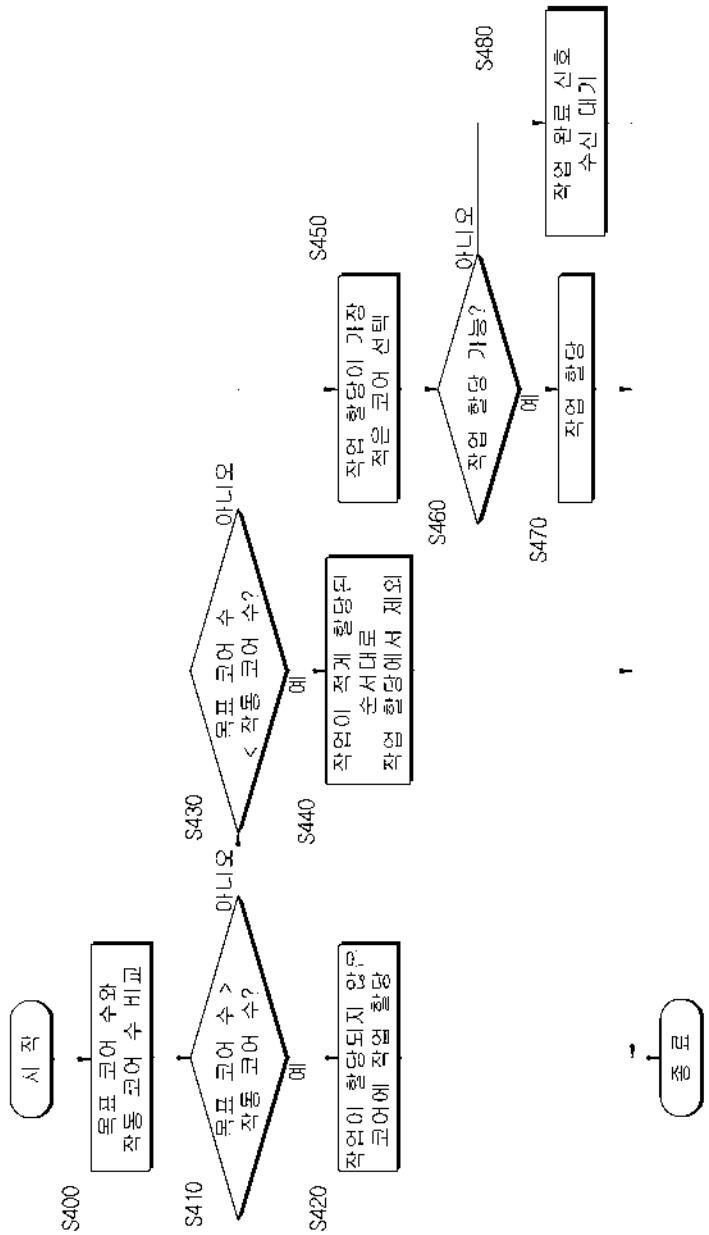
도면2



도면3


시 작
평균 메모리 응답 시간(AML) 계산
S300
제1 임계값 < AML?
S320
아니오
예
목표 코어 수 감소
S340
제2 임계값 > AML?
S360
아니오
예
목표 코어 수 증가
S380
종 료

도면4


시 작
목표 코어 수와
작동 코어 수 비교
S400
목표 코어 수 >
작동 코어 수?
S410
예
작업이 할당되지 않은
코어에 작업 할당
S420
아니오
목표 코어 수
< 작동 코어 수?
S430
예
작업이 적게 할당된
순서대로
작업 할당에서 제외
S440
아니오
작업 할당이 가장
적은 코어 선택
S450
작업 할당 가능?
S460
예
작업 할당
S470
아니오
작업 완료 신호
수신 대기
S480
종 료

(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2013년06월17일
(11) 등록번호 10-1275640
(24) 등록일자 2013년06월11일

(51) 국제특허분류(Int. Cl.)
G06F 13/14 (2006.01) *G06F 13/38* (2006.01)
(21) 출원번호 10-2011-0131343
(22) 출원일자 2011년12월08일
심사청구일자 2011년12월08일
(56) 선행기술조사문헌
JP2010049508 A*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
유니슨 주식회사
경상남도 사천시 사남면 해안산업로 477
전자부품연구원
경기도 성남시 분당구 새나리로 25 (야탑동)
(72) 발명자
함경선
경기도 용인시 수지구 죽전2동 벽산3단지아파트 301동 708호
김동순
경기도 수원시 장안구 정자1동 파크뷰 606동 3203호
(뒷면에 계속)
(74) 대리인
남충우, 노철호

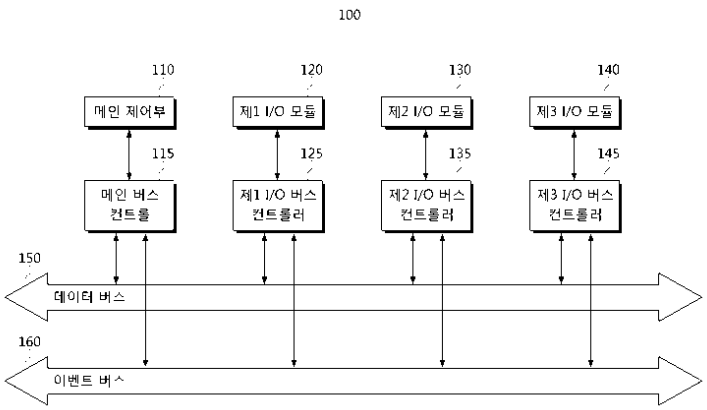
전체 청구항 수 : 총 7 항 심사관 : 이수철

(54) 발명의 명칭 **다수의 버스를 이용하는 논리연산 제어장치**

(57) 요 약

논리연산 제어장치가 제공된다. 본 논리연산 제어장치는 메인 제어부 및 적어도 하나의 I/O 모듈의 통신 경로가 되고 메인 제어부에 의해 통신이 제어되는 제1 버스 및 메인 제어부 및 적어도 하나의 I/O 모듈의 통신 경로가 되고 적어도 하나의 I/O 모듈에 의해 제어되는 제2 버스를 포함하여, 논리연산 제어장치에 특화된 고성능의 버스 기술을 제공할 수 있게 된다.

대 표 도 - 도1



(72) 발명자

황태호

서울특별시 송파구 가락2동 래미안파크팰리스아파트 108동 803호

정혜동

서울특별시 강남구 도곡동 527 도곡렉슬아파트 104동 1605호

김영환

경기도 용인시 수지구 풍덕천동 170-1 현대아파트 102동 103호

이 발명을 지원한 국가연구개발사업

과제고유번호	2010T100200257
부처명	지식경제부
연구사업명	신재생에너지기술개발사업
연구과제명	풍력발전시스템용 제어기술 및 기기 개발
주관기관	(주)유니슨
연구기간	2010.06.01 ~ 2014.02.28

특허청구의 범위

청구항 1

논리 연산을 수행하는 메인 제어부;

데이터를 입출력하는 적어도 하나의 I/O 모듈;

상기 메인 제어부 및 상기 적어도 하나의 I/O 모듈의 통신 경로가 되고, 상기 메인 제어부에 의해 통신이 제어되는 제1 버스; 및

상기 메인 제어부 및 상기 적어도 하나의 I/O 모듈의 통신 경로가 되고, 상기 적어도 하나의 I/O 모듈에 의해 제어되는 제2 버스;를 포함하고,

상기 적어도 하나의 I/O 모듈은,

우선순위 정보가 포함된 토큰(Token)을 이용하여 상기 제2 버스를 통한 상기 메인 제어부와의 통신 여부를 결정하는 것을 특징으로 하는 논리연산 제어장치.

청구항 2

제1항에 있어서,

상기 적어도 하나의 I/O 모듈 각각을 상기 제1 버스 및 상기 제2 버스와 연결하고, 상기 적어도 하나의 I/O 모듈 각각과 상기 제1 버스 간의 통신 및 상기 적어도 하나의 I/O 모듈 각각과 상기 제2 버스 간의 통신을 제어하는 적어도 하나의 I/O 버스 컨트롤러;를 더 포함하는 것을 특징으로 하는 논리연산 제어장치.

청구항 3

제2항에 있어서,

상기 I/O 버스 컨트롤러 각각은,

데이터를 상기 I/O 모듈의 프로토콜에 맞게 변형하여 송수신하는 입출력 인터페이스부;

상기 제1버스 또는 상기 제2버스를 통해 송수신되는 데이터를 인코딩 및 디코딩하는 버스 제어부;

상기 I/O 모듈에 이벤트가 발생된 경우, 입출력 이벤트를 생성하는 이벤트 생성부; 및

상기 버스 제어부 및 상기 이벤트 생성부에서 송수신되는 데이터를 버스 프로토콜에 맞게 변형하여 송수신하는 버스 인터페이스부;를 포함하는 것을 특징으로 하는 논리연산 제어장치.

청구항 4

제3항에 있어서,

상기 이벤트 생성부는,

상기 제2 버스를 통해 상기 생성된 입출력 이벤트를 상기 버스 제어부로 전송하는 것을 특징으로 하는 논리연산 제어장치.

청구항 5

제1항에 있어서,

상기 메인 제어부를 상기 제1 버스 및 상기 제2 버스와 연결하고, 상기 메인 제어부와 상기 제1 버스 간의 통신

및 상기 메인 제어부와 상기 제2 버스 간의 통신을 제어하는 메인 버스 컨트롤러;를 더 포함하는 것을 특징으로 하는 논리연산 제어장치.

청구항 6

제5항에 있어서,

상기 메인 버스 컨트롤러는,

데이터를 상기 메인 제어부의 프로토콜에 맞게 변형하여 송수신하는 메인 인터페이스부;

상기 제1 버스 또는 상기 제2 버스를 통해 송수신되는 데이터를 인코딩 및 디코딩하는 버스 제어부;

상기 I/O 버스 컨트롤러로부터 수신된 입출력 이벤트를 우선순위에 따라 정렬하여 상기 메인 제어부로 전송하는 이벤트 핸들러; 및

상기 버스 제어부 및 상기 이벤트 핸들러에서 송수신되는 데이터를 버스 프로토콜에 맞게 변형하여 송수신하는 버스 인터페이스부;를 포함하는 것을 특징으로 하는 논리연산 제어장치.

청구항 7

제6항에 있어서,

상기 이벤트 핸들러는,

상기 제2 버스를 통해 상기 I/O 버스 제어부로부터 상기 입출력 이벤트를 수신하는 것을 특징으로 하는 논리연산 제어장치.

청구항 8

삭제

명 세 서

기 술 분 야

[0001] 본 발명은 논리연산 제어장치에 관한 것으로, 더욱 상세하게는, 다수의 버스를 이용하는 논리연산 제어장치에 관한 것이다.

배 경 기 술

[0002] 논리연산제어장치의 구성을 위한 입출력시스템버스는 일반적으로 백플레인(Backplane) 버스 구조가 사용된다. 백플레인 버스의 구현을 위하여 검증된 시스템 버스 표준인 VME64, CPCL, VXI 등을 사용할 수 있다. 하지만, 표준버스는 다양한 시스템의 호환성을 위하여 복잡한 구조를 가지고 있으며, 논리연산 제어장치라는 특정 분야에 최적화 되어있지 않다.

[0003] 논리연사 제어장치의 시스템 버스를 구현하기 위하여 VME64와 같은 표준을 사용하는 경우도 있지만, 앞에서 언급한 문제점 때문에 산업계에서는 대부분은 논리연산 제어장치에 특화된 버스를 직접 설계하여 사용하는 것이 보편적이다. 업체에서 직접 설계된 버스구조는 외부에 공개하지 않음으로, 각 업체의 버스 성능을 객관적으로 평가하기 힘들뿐만 아니라, 새로운 논리연산 제어장치를 개발할 때, 기술을 참조하거나, 사용하는데 어려움이 많다.

[0004] 독자적인 논리연산제어장치를 개발하고자 할 경우에는 직접 버스구조를 설계하고 동작에 대한 안정성을 검증해야 하기 때문에, 개발기간이나 신뢰성 측면에서 어려움이 발생한다.

[0005] 이에 따라, 논리연산 제어장치에 특화된 고성능의 버스 기술을 개발하기 위한 방안의 모색이 요청된다.

발명의 내용

해결하려는 과제

[0006] 본 발명은 상기와 같은 문제점을 해결하기 위하여 안출된 것으로서, 본 발명의 목적은, 메인 제어부 및 적어도 하나의 I/O 모듈의 통신 경로가 되고 메인 제어부에 의해 통신이 제어되는 제1 버스 및 메인 제어부 및 적어도 하나의 I/O 모듈의 통신 경로가 되고 적어도 하나의 I/O 모듈에 의해 제어되는 제2 버스를 포함하는 논리연산 제어장치를 제공함에 있다.

과제의 해결 수단

[0007] 상기 목적을 달성하기 위한 본 발명의 일 실시예에 따른, 논리연산 제어장치는, 논리 연산을 수행하는 메인 제어부; 데이터를 입출력하는 적어도 하나의 I/O 모듈; 상기 메인 제어부 및 상기 적어도 하나의 I/O 모듈의 통신 경로가 되고, 상기 메인 제어부에 의해 통신이 제어되는 제1 버스; 상기 메인 제어부 및 상기 적어도 하나의 I/O 모듈의 통신 경로가 되고, 상기 적어도 하나의 I/O 모듈에 의해 제어되는 제2 버스;를 포함한다 .

[0008] 그리고, 상기 적어도 하나의 I/O 모듈 각각을 상기 제1 버스 및 상기 제2 버스와 연결하고, 상기 적어도 하나의 I/O 모듈 각각과 상기 제1 버스 간의 통신 및 상기 적어도 하나의 I/O 모듈 각각과 상기 제2 버스 간의 통신을 제어하는 적어도 하나의 I/O 버스 컨트롤러;를 더 포함할 수도 있다.

[0009] 또한, 상기 I/O 버스 컨트롤러 각각은, 데이터를 상기 I/O 모듈의 프로토콜에 맞게 변형하여 송수신하는 입출력 인터페이스부; 상기 제1버스 또는 상기 제2버스를 통해 송수신되는 데이터를 인코딩 및 디코딩하는 버스 제어부; 상기 I/O 모듈에 이벤트가 발생된 경우, 입출력 이벤트를 생성하는 이벤트 생성부; 및 상기 버스 제어부 및 상기 이벤트 생성부에서 송수신되는 데이터를 버스 프로토콜에 맞게 변형하여 송수신하는 버스 인터페이스부;를 포함한다.

[0010] 그리고, 상기 이벤트 생성부는, 상기 제2 버스를 통해 상기 생성된 입출력 이벤트를 상기 메인 버스 제어부로 전송할 수도 있다.

[0011] 또한, 상기 메인 제어부를 상기 제1 버스 및 상기 제2 버스와 연결하고, 상기 메인 제어부와 상기 제1 버스 간의 통신 및 상기 메인 제어부와 상기 제2 버스 간의 통신을 제어하는 메인 버스 컨트롤러;를 더 포함할 수도 있다.

[0012] 그리고, 상기 메인 버스 컨트롤러는, 데이터를 상기 메인 제어부의 프로토콜에 맞게 변형하여 송수신하는 메인 인터페이스부; 상기 제1 버스 또는 상기 제2 버스를 통해 송수신되는 데이터를 인코딩 및 디코딩하는 버스 제어부; 상기 I/O 버스 컨트롤러로부터 수신된 입출력 이벤트를 우선순위에 따라 정렬하여 상기 메인 제어부로 전송하는 이벤트 핸들러; 및 상기 버스 제어부 및 상기 입출력 이벤트 핸들러에서 송수신되는 데이터를 버스 프로토콜에 맞게 변형하여 송수신하는 버스 인터페이스부;를 포함할 수도 있다.

[0013] 또한, 상기 입출력 이벤트 핸들러는, 상기 제2 버스를 통해 상기 I/O 버스 제어부로부터 상기 입출력 이벤트를 수신할 수도 있다.

[0014] 그리고, 상기 적어도 하나의 I/O 모듈는, 우선순위 정보가 포함된 토큰(Token)을 이용하여 상기 제2 버스를 통한 상기 메인 제어부와의 통신 여부를 결정할 수도 있다.

발명의 효과

[0015] 본 발명의 다양한 실시예에 따르면, 메인 제어부 및 적어도 하나의 I/O 모듈의 통신 경로가 되고 메인 제어부에 의해 통신이 제어되는 제1 버스 및 메인 제어부 및 적어도 하나의 I/O 모듈의 통신 경로가 되고 적어도 하나의 I/O 모듈에 의해 제어되는 제2 버스를 포함하는 논리연산 제어장치를 제공할 수 있게 되어, 논리연산 제어장치에 특화된 고성능의 버스 기술을 제공할 수 있게 된다.

도면의 간단한 설명

[0016] 도 1은 본 발명의 일 실시예에 따른, 논리연산 제어장치의 구성을 도시한 블럭도,

도 2는 본 발명의 일 실시예에 따른, 메인 버스 컨트롤러 및 I/O 버스 컨트롤러의 구성을 도시한 도면,

도 3은 본 발명의 일 실시예에 따른, 이벤트 버스의 통신 방식을 도시한 도면이다.

발명을 실시하기 위한 구체적인 내용

[0017] 이하에서는 도면을 참조하여 본 발명을 보다 상세하게 설명한다.

[0018] 도 1은 본 발명의 일 실시예에 따른, 논리연산 제어장치(100)의 구성을 도시한 블럭도이다. 도 1에 도시된 바와 같이, 논리연산 제어장치(100)는 메인 제어부(110), 메인 버스 컨트롤러(115), 제1 I/O 모듈(120), 제1 I/O 버스 컨트롤러(125), 제2 I/O 모듈(130), 제2 I/O 버스 컨트롤러(135), 제3 I/O 모듈(140), 제3 I/O 버스 컨트롤러(145), 데이터 버스(150) 및 이벤트 버스(160)를 포함한다.

[0019] 메인 제어부(110)는 논리 연산을 수행한다. 또한, 제어부(110)는 논리연산 제어장치(100)의 전반적인 동작을 제어하며, 입력된 이벤트를 우선순위별로 처리한다.

[0020] 메인 버스 컨트롤러(115)는 메인 제어부(110)를 데이터 버스(150) 및 이벤트 버스(160)와 연결하고, 메인 제어부(110)와 데이터 버스(150) 간의 통신 및 메인 제어부(110)와 이벤트 버스(160) 간의 통신을 제어한다. 구체적으로, 메인 버스 컨트롤러(115)는 메인 제어부(110)가 데이터 버스(150)를 통해 I/O 모듈들(120, 130, 140)로 명령 패킷 및 데이터 요청 패킷을 전송하도록 제어한다. 그리고, 메인 버스 컨트롤러(115)는 메인 제어부(110)가 이벤트 버스(160)를 통해 I/O 모듈들(120, 130, 140)로부터 입출력 이벤트를 수신하도록 제어한다.

[0021] 제1 I/O 모듈(120), 제2 I/O 모듈(130), 및 제3 I/O 모듈(140)는 각각 연결된 외부 장치들과 데이터를 입출력한다.

[0022] 제1 I/O 버스 컨트롤러(125), 제2 I/O 버스 컨트롤러(135) 및 제3 I/O 버스 컨트롤러(145)는 I/O 모듈 각각(120,130,140)을 데이터 버스(150) 및 이벤트 버스(160)와 연결하고, I/O 모듈 각각(120,130,140)과 데이터 버스(150) 간의 통신 및 I/O 모듈 각각(120,130,140)과 이벤트 버스(160) 간의 통신을 제어한다. 구체적으로, 제1 I/O 버스 컨트롤러(125), 제2 I/O 버스 컨트롤러(135) 및 제3 I/O 버스 컨트롤러(145)는 I/O 모듈 각각(120,130,140)이 데이터 버스(150)를 통해 메인 제어부(110)로부터 명령 패킷 및 데이터 요청 패킷을 수신하도록 제어한다. 그리고, 제1 I/O 버스 컨트롤러(125), 제2 I/O 버스 컨트롤러(135) 및 제3 I/O 버스 컨트롤러(145)는 이벤트 버스(160)를 통해 I/O 모듈들(120, 130, 140)이 메인 제어부(110)로 입출력 이벤트를 전송하도록 제어한다.

[0023] 데이터 버스(150)는 메인 제어부(110) 및 I/O 모듈 각각(120,130,140)의 통신 경로가 되고, 메인 제어부(110)에 의해 통신이 제어된다. 데이터 버스(150)는 메인 제어부(110) 및 I/O 모듈 각각(120,130,140)이 연결된다.

[0024] 구체적으로, 데이터 버스(150)는 통신 방식이 동기식 방식이며, 마스터/슬레이브 구조로 되어있다. 이때, 논리연산 제어장치(100)의 메인 제어부(110)가 네트워크의 마스터가 되며, 그 외의 I/O 모듈들(120,130,140)이 슬레이브가 된다. 동기신호는 마스터에 의하여 생성되며, 동기신호를 제외한 다른 버스 컨트롤 신호는 존재하지 않는다. 데이터 버스(150)는 메인 제어부(110) 및 I/O 모듈들(120,130,140)을 포함하는 모든 모듈이 24비트 병렬 버스를 공유하여 사용하지만, 마스터만이 통신을 시작하기 때문에 특별한 충돌방지 메커니즘은 필요하지 않다. 슬레이브는 항상 마스터로부터 송신된 패킷을 대기한다. 마스터인 메인 제어부(110)로부터 전송되는 패킷은 크게 두 가지가 있는데, 명령 패킷과 데이터 요청 패킷이다. 명령 패킷은 마스터가 슬레이브에게 컨트롤 명령을 보내는 것으로 리셋명령, 채널 설정명령, 입출력 채널의 신호 출력 등이 있다. 데이터 요청 패킷은 마스터가 슬레이브의 현재 레지스터 상태 및 입출력 채널의 수신신호를 읽기위한 데이터 요청 패킷이다. 데이터 버스(150)에서는 어떠한 상황에서도 슬레이브가 능동적으로 데이터 버스를 점령할 수 없으며, 필요에 따라 마스터는 슬레이브에서 전송하고자 하는 데이터가 있는지 주기적으로 폴링(Polling)한다. 이와 같이, 데이터 버스(150)는 메인 제어부(110)에 의해 제어되는 것을 확인할 수 있다.

[0025] 이벤트 버스(160)는 메인 제어부(110) 및 I/O 모듈들(120,130,140)의 통신 경로가 되고, I/O 모듈들(120,130,140)에 의해 제어된다. 이벤트 버스(160)는 메인 제어부(110) 및 I/O 모듈 각각(120,130,140)이 연결

된다. 이벤트 버스(160)는 발생된 입출력 이벤트의 우선순위에 따라 I/O 모듈들(120,130,140) 중 버스를 점유하는 I/O 모듈가 결정된다. 따라서, 이벤트 버스(160)는 일정 주기마다 I/O 모듈들(120,130,140) 중 어느 하나가 점유하게 된다.

[0026] 구체적으로, 이벤트 버스(160)는 데이터 버스(150)와 마찬가지로 동기식 24비트 병렬버스이며 마스터/슬레이브 구조로 되어있다. 이때, 논리연산 제어장치(100)의 메인 제어부(110)가 네트워크의 마스터가 되며, 그 외의 I/O 모듈들(120,130,140)이 슬레이브가 된다. 동기신호는 마스터에 의하여 발생된다. 이벤트 버스(160)는 슬레이브가 능동적으로 버스를 점유할 수 있다. 즉, 이벤트 버스(160)는 슬레이브인 I/O 모듈들(120,130,140)이 점유할 수 있다. 다수의 슬레이브인 I/O 모듈들(120,130,140)이 동시에 이벤트 버스(160)에 연결되어 있기 때문에, 슬레이브가 동시에 버스를 점유하게 되면 데이터의 충돌이 발생할 수 있다. 이러한 문제점을 해결하기 위하여 이벤트 버스(160)의 통신에는 토큰(Token)방식이 이용된다. 토큰은 입출력 이벤트의 우선순위 정보가 포함되어 있다. 그리고, 슬레이브인 I/O 모듈들(120,130,140)은 우선순위 정보가 포함된 토큰(Token)을 이용하여 이벤트 버스(160)를 통한 메인 제어부(110)와의 통신 여부를 결정하게 된다. 토큰은 데이지 체인 버스를 통하여 전달된다. 여기에서, 데이지 체인 버스는 토큰을 전달하기 위한 이벤트 버스(160) 내의 논리적 버스이다. 데이지 체인 버스는 메인 제어부(110) → 제1 I/O 모듈(120) → 제2 I/O 모듈(130) → 제3 I/O 모듈(140) →메인 제어부(110)의 순서로 순환 형태로 토큰을 전송하는 경로이다. 슬레이브는 이벤트 버스(160)를 사용하기 위해서 데이지 체인을 통하여 들어오는 토큰을 항상 모니터링 하여야 한다.

[0027] 데이지 체인은 이벤트 버스(160)를 통해 슬레이브가 이벤트 버스를 점령하기 위한 토큰이 전달되는 경로를 나타낸다. 데이지 체인은 총 4개의 버스라인을 가지고 있으며, 동기신호를 전달하기 위한 Sync 라인과 현재 버스가 사용 중임을 나타내는 Busy 시그널 라인, 직렬 데이터를 보내기 위한 데이터 라인, 마지막으로 마스터와 직접 연결되는 또 다른 데이터 라인이 있다. 토큰의 생성 및 이용방식은 추후 상세히 설명한다.

[0028] 도 2는 본 발명의 일 실시예에 따른, 메인 버스 컨트롤러(115) 및 제1 I/O 버스 컨트롤러(125)의 구성을 도시한 도면이다. 제2 I/O 버스 컨트롤러(135) 및 제3 I/O 버스 컨트롤러(145)는 제1 I/O 버스 컨트롤러(125)와 구성이 동일하므로, 이하에서는 제1 I/O 버스 컨트롤러(125)의 구성에 대해서만 설명한다.

[0029] 도 2에 도시된 바와 같이, 메인 버스 컨트롤러(115)는 메인 인터페이스부(210), 버스 제어부(220), 이벤트 핸들러(230), 및 버스 인터페이스부(240)를 포함한다.

[0030] 메인 인터페이스부(210)는 데이터를 메인 제어부(110)의 프로토콜에 맞게 변형하여 송수신한다. 즉, 메인 인터페이스부(210)는 버스 제어부(220)에서 송수신 되는 데이터를 메인 제어부(110)의 프로토콜에 맞게 변형한 후에, 변형된 데이터를 메인 제어부(110)로 송수신하게 된다.

[0031] 버스 제어부(220)는 데이터 버스(150) 또는 이벤트 버스(160)를 통해 송수신되는 데이터를 인코딩 및 디코딩한다. 구체적으로, 버스 제어부(220)는 메인 제어부(110)에 입력하기 위한 데이터 패킷을 패킷 디코딩하고, 데이터 버스(150) 또는 이벤트 버스(160)로 출력하고자 하는 데이터를 패킷 인코딩하여 출력하게 된다.

[0032] 이벤트 핸들러(230)는 이벤트 버스(160)를 통해 I/O 버스 컨트롤러(125)로부터 수신된 입출력 이벤트를 우선순위에 따라 정렬하여 메인 제어부(110)로 전송한다. 구체적으로, 이벤트 핸들러(230)는 이벤트 버스(160)를 통해 I/O 버스 컨트롤러(125)로부터 입출력 이벤트를 수신한다. 그리고, 이벤트 핸들러(230)는 수신된 입출력 이벤트를 기설정된 입출력 이벤트별 우선순위에 따라 정렬하여 메인 제어부(110)로 전송한다.

[0033] 버스 인터페이스부(240)는 버스 제어부(220) 및 입출력 이벤트 핸들러(230)에서 송수신되는 데이터를 버스 프로토콜에 맞게 변형하여 송수신한다. 즉, 버스 인터페이스부(240)는 버스 제어부(220) 및 입출력 이벤트 핸들러(230)에서 송수신되는 데이터를 데이터 버스(150) 및 이벤트 버스(160)의 프로토콜에 맞게 변형한 후에, 변형된 데이터를 데이터 버스(150) 및 이벤트 버스(160)로 송수신하게 된다.

[0034] 이와 같은 구조의 메인 버스 컨트롤러(115)는 메인 제어부(110)가 이벤트 버스(160)를 통해 I/O 모듈들(120, 130, 140)로부터 입출력 이벤트를 수신하도록 제어할 수 있게 된다.

[0035] 또한, 도 2에 도시된 바와 같이, 제1 I/O 버스 컨트롤러(125)는 입출력 인터페이스부(250), 버스 제어부(260), 이벤트 생성부(270), 및 버스 인터페이스부(280)를 포함한다.

[0036] 입출력 인터페이스부(250)는 데이터를 제1 I/O 모듈(120)의 프로토콜에 맞게 변형하여 송수신한다. 즉, 입출력 인터페이스부(250)는 버스 제어부(260)에서 송수신 되는 데이터를 제1 I/O 모듈(120)의 프로토콜에 맞게 변형한 후에, 변형된 데이터를 제1 I/O 모듈(120)로 송수신하게 된다.

[0037] 버스 제어부(260)는 데이터 버스(150) 또는 이벤트 버스(160)를 통해 송수신되는 데이터를 인코딩 및 디코딩한다. 구체적으로, 버스 제어부(260)는 제1 I/O 모듈(120)에 입력하기 위한 데이터 패킷을 패킷 디코딩하고, 데이터 버스(150) 또는 이벤트 버스(160)로 출력하고자 하는 데이터를 패킷 인코딩하여 출력하게 된다.

[0038] 이벤트 생성부(270)는 제1 I/O 모듈(120)에 이벤트가 발생된 경우, 입출력 이벤트를 생성하게 된다. 입출력 이벤트는 제1 I/O 모듈(120)의 내부 또는 입력 신호 등에서 발생되는 이벤트를 나타내며, 예를 들어 입력 에러 이벤트 또는 고장 이벤트 등이 될 수 있다. 또한, 이벤트 생성부(270)는 생성된 입출력 이벤트의 우선순위가 다른 이벤트들에 비하여 최상위인 경우에 입출력 이벤트를 이벤트 버스(160)를 통해 메인 버스 컨트롤러(115)로 전송하게 된다.

[0039] 버스 인터페이스부(280)는 버스 제어부(260) 및 이벤트 생성부(270)에서 송수신되는 데이터를 버스 프로토콜에 맞게 변형하여 송수신한다. 즉, 버스 인터페이스부(280)는 버스 제어부(260) 및 이벤트 생성부(270)에서 송수신되는 데이터를 데이터 버스(150) 및 이벤트 버스(160)의 프로토콜에 맞게 변형한 후에, 변형된 데이터를 데이터 버스(150) 및 이벤트 버스(160)로 송수신하게 된다.

[0040] 이와 같은 구조의 제1 I/O 버스 컨트롤러(125)는 제1 I/O 모듈(120)에서 발생된 입출력 이벤트가 우선순위에 따라 이벤트 버스(160)를 통해 메인 제어부(110)로 전송되도록 제어할 수 있게 된다.

[0041] 이하에서는, 도 3을 참고하여, 이벤트 버스(160)를 통해 입출력 이벤트가 전송되는 과정에 대해 설명한다. 도 3은 본 발명의 일 실시예에 따른, 이벤트 버스(160)의 통신 방식을 도시한 도면이다.

[0042] 도 3에서 점선은 이벤트 버스(160)를 나타내고, 실선 화살표는 데이지 체인(300)을 나타낸다. 토큰(310, 320)은 이벤트 버스(160)를 통해 데이지 체인의 순서에 따라 전송된다.

[0043] 최초에 논리연산 제어장치(100)가 구동되면, 도 3에 도시된 바와 같이, 마스터인 메인 제어부(110)는 스타트 패킷(Start packet) 형태의 토큰을 생성하고, 메인 버스 컨트롤러(115)는 데이지 체인(300)을 따라 가장 가까운 슬레이브로 스타트 패킷을 포함하는 토큰을 전송한다. 스타트 패킷은 8비트 스타트 비트와 모듈 아이디를 포함한다.

[0044] 제일 가까이 연결된 슬레이브(S1)인 제1 I/O 모듈(120)의 제1 I/O 버스 컨트롤러(125)는 토큰을 수신한 후 메인 버스 컨트롤러(115)로 ACK신호를 보낸다.

[0045] 제1 I/O 버스 컨트롤러(125)는 수신한 토큰의 스타트 패킷의 뒷부분에 자신의 패킷인 슬레이브 패킷(S1)을 덧붙인다. 슬레이브 패킷(S1)은 스타트를 나타내는 8비트, 모듈 아이디, 모듈과 연결된 외부 장치를 나타내는 정보(FUNC), 그리고 마지막으로 보낼 입출력 이벤트 데이터가 있음을 우선순위별로 나타내는 8비트 비트마스크(Bit mask)를 포함한다.

[0046] 비트마스크는 이벤트 버스(160)를 통해 전송되는 패킷을 8개의 우선순위로 구분하여 표시한다. 각 I/O 버스 컨트롤러(125,135,145)에서 토큰 패킷에 자신의 정보를 추가하는 내용 중에 비트 마스크는 자신이 보낼 패킷이 존재함을 나타낸다. 비트 마스크의 최상위비트는 제일 높은 우선순위 패킷을 의미하며 마지막 비트는 제일 낮은 우선순위의 패킷을 의미한다. 여기에서, 패킷은 입출력 이벤트가 될 수도 있으며, 이외에 다른 패킷이 될 수도 있다. 예를 들어, 비트 마스크의 값이 '01000100'이면, 전송할 패킷이 우선순위가 2인 패킷과 우선순위가 6인 패킷으로 2개가 있다는 것을 나타낸다. 이와 같이, 비트 마스크는 우선순위별 전송 대기중인 패킷이 존재함을 나타낸다.

[0047] 이후에, 제1 I/O 버스 컨트롤러(125)는 토큰에 슬레이브 패킷(S1)을 덧붙인 후에, 데이지 체인을 따라 제2 I/O 버스 컨트롤러(135)로 토큰을 전송한다. 그 후에, 제2 I/O 버스 컨트롤러(135)와 제3 I/O 버스 컨트롤러(145)도 순차적으로 자신의 슬레이브 패킷(S2, S3)을 토큰에 덧붙이게 된다. 제3 I/O 버스 컨트롤러(145)는 ACK신호를 수신할 수 없기 때문에, 자신이 마지막임을 알 수 있게 되며, 완성된 토큰(320)을 다시 이벤트 버스(160)의 데이지 체인을 따라 메인 버스 컨트롤러로 전달한다.

[0048] 이 후에, 토큰(320)은 데이지 체인을 따라 계속적으로 '메인 버스 컨트롤러(115) → 제1 I/O 버스 컨트롤러(125) → 제2 I/O 버스 컨트롤러(135) → 제3 I/O 버스 컨트롤러(145) → 메인 버스 컨트롤러(115)'의 순서로 순환 전달된다. 그리고, 각 I/O 버스 컨트롤러(125, 135, 145)는 메인 제어부(110)로 전송하고자 하는 입출력 이벤트가 발생된 경우, 토큰(320)의 비트 마스크에 해당 우선순위영역의 비트마스크 값을 "1"로 기록한다. 그리고, 각 I/O 버스 컨트롤러(125, 135, 145)는 토큰(320)을 수신하게 되면, 비트마스크를 서로 비교하여 다른 모듈이 자신의 입출력 이벤트보다 우선순위가 높은지 여부를 판단한다. 만약, 자신의 입출력 이벤트의 우선순위가

가장 높은 경우, 해당 I/O 버스 컨트롤러는 이벤트 버스(160)를 점유하여 해당 입출력 이벤트를 이벤트 버스(160)를 통해 메인 버스 컨트롤부(115)로 전송한다. 그리고, 해당 I/O 버스 컨트롤러는 해당 비트 마스크를 '0'으로 클리어하고, 토큰(320)을 데이지 체인을 따라 다음 I/O 버스 컨트롤러로 전송한다.

[0049] 반면, 자신의 입출력 이벤트의 우선순위가 가장 높은 것이 아닌 경우, 해당 I/O 버스 컨트롤러는 데이지 체인을 따라 다음 I/O 버스 컨트롤러로 토큰(320)을 전송한다.

[0050] 예를 들어, 제1 I/O 버스 컨트롤러(125)의 비트 마스크는 '00100000'이고, 제2 I/O 버스 컨트롤러(135)의 비트 마스크는 '10000010'이고, 제3 I/O 버스 컨트롤러(115)의 비트 마스크는 '00000001'인 경우를 가정한다.

[0051] 그러면, 최초 데이지 체인을 순환할 때, 제2 I/O 버스 컨트롤러(135)의 비트 마스크가 우선순위가 가장 높으므로, 제2 I/O 버스 컨트롤러(135)가 우선순위 1에 해당되는 입출력 이벤트를 메인 버스 컨트롤러(115)로 전송한다.

[0052] 1번의 데이지 체인 순환이 종료되면, 제1 I/O 버스 컨트롤러(125)의 비트 마스크는 '00100000'이고, 제2 I/O 버스 컨트롤러(135)의 비트 마스크는 '00000010'이고, 제3 I/O 버스 컨트롤러(115)의 비트 마스크는 '00000001'가 된다. 2번째 데이지 체인 순환시에는 제1 I/O 버스 컨트롤러(125)가 먼저 우선순위가 3인 입출력 이벤트를 메인 버스 컨트롤러(115)로 전송하고, 그다음 제2 I/O 버스 컨트롤러(135)가 우선순위가 7인 입출력 이벤트를 메인 버스 컨트롤러(115)로 전송하며, 마지막으로 제3 I/O 버스 컨트롤러(145)가 우선순위가 8인 입출력 이벤트를 메인 버스 컨트롤러(115)로 전송하게 된다.

[0053] 따라서, 상기의 가정의 경우, 2번은 데이지 체인 순환을 통해 모든 입출력 이벤트의 전송이 완료되게 된다.

[0054] 이와 같이, 토큰(320)의 비트 마스크에 표시된 우선순위를 기반으로 하여, 이벤트 버스(160)는 통신이 제어되게 된다. 또한, 토큰(320)은 I/O 버스 컨트롤러들을 하나씩 거쳐감으로써, 다수의 I/O 버스 컨트롤러들이 충돌없이 이벤트 버스(160)를 이용하여 통신할 수 있게 된다.

[0055] 본 실시예에서는, I/O 모듈가 3개인 것으로 설명하였으나 이는 설명의 편의를 위한 것이며, I/O 모듈의 개수는 적어도 1개 이상이 될 수 있음은 물론이다.

[0056] 또한, 이상에서는 본 발명의 바람직한 실시예에 대하여 도시하고 설명하였지만, 본 발명은 상술한 특정의 실시예에 한정되지 아니하며, 청구범위에서 청구하는 본 발명의 요지를 벗어남이 없이 당해 발명이 속하는 기술분야에서 통상의 지식을 가진자에 의해 다양한 변형실시가 가능한 것은 물론이고, 이러한 변형실시들은 본 발명의 기술적 사상이나 전망으로부터 개별적으로 이해되어져서는 안될 것이다.

부호의 설명

[0057] 100 : 논리연산 제어장치

110 : 메인 제어부

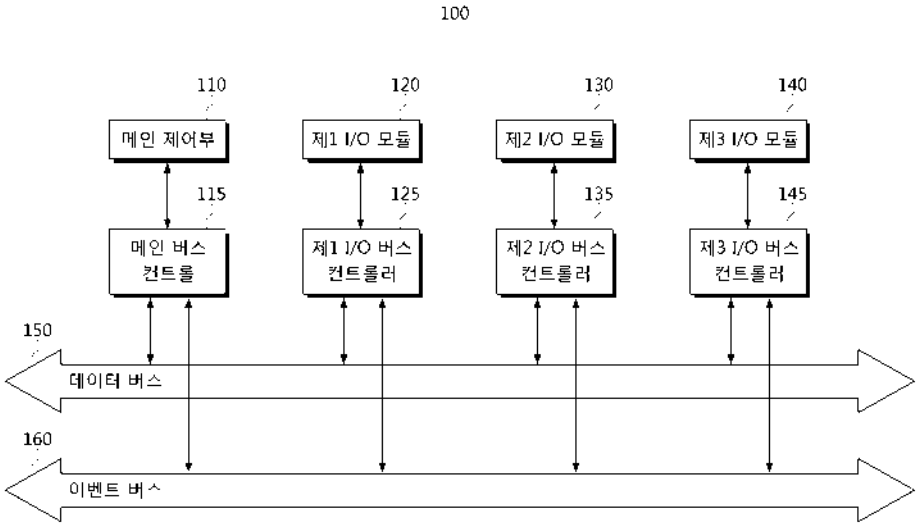
120 : 제1 I/O 모듈

130 : 제2 I/O 모듈

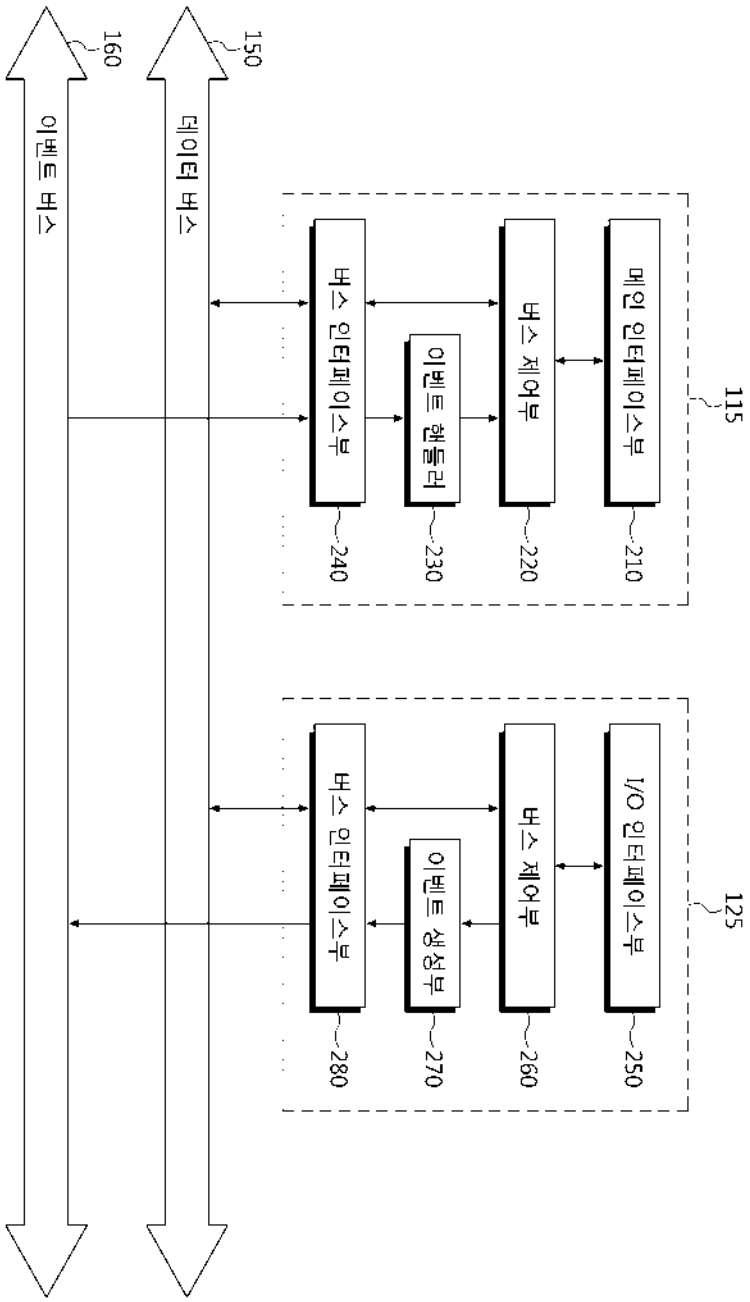
140 : 제3 I/O 모듈

도면

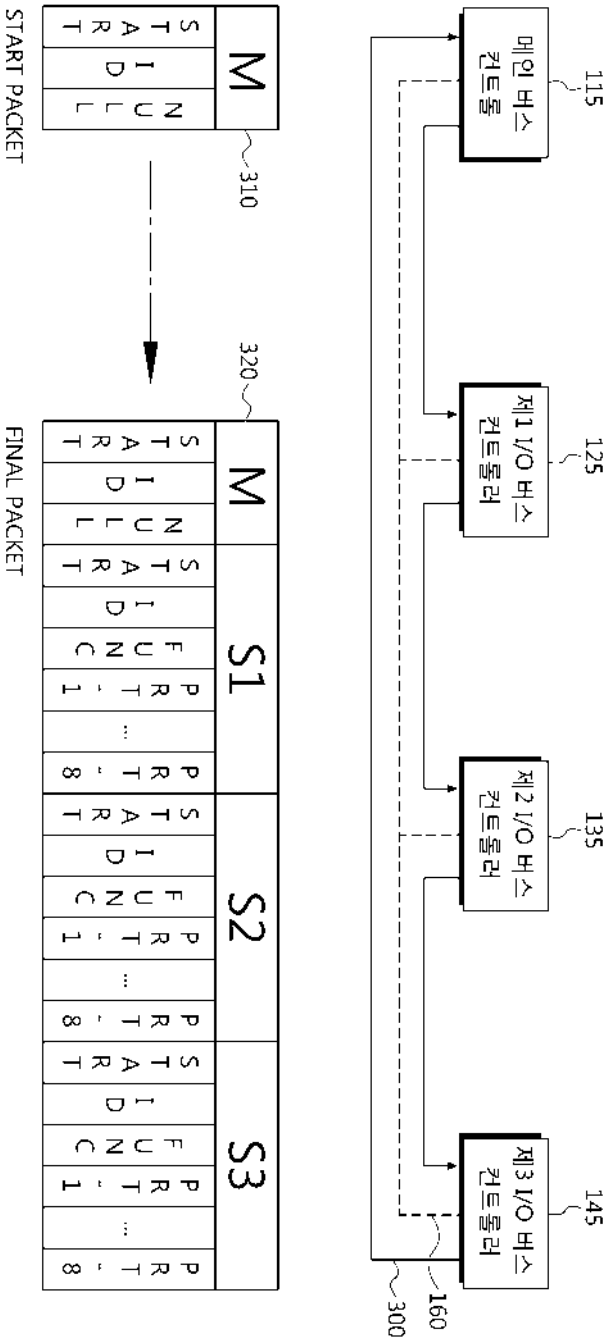
도면1


100
110
메인 제어부
120
제1 I/O 모듈
130
제2 I/O 모듈
140
제3 I/O 모듈
115
메인 버스 컨트롤
125
제1 I/O 버스 컨트롤러
135
제2 I/O 버스 컨트롤러
145
제3 I/O 버스 컨트롤러
150
데이터 버스
160
이벤트 버스

도면2


160
150
이벤트 버스
데이터 버스
115
메인 인터페이스부
버스 제어부
이벤트 핸들러
버스 인터페이스부
210
220
230
240
125
I/O 인터페이스부
버스 제어부
이벤트 생성부
버스 인터페이스부
250
260
270
280

도면3


START PACKET
FINAL PACKET
M
S1
S2
S3
310
320
115
125
135
145
160
300
메인 버스 컨트롤
제1 I/O 버스 컨트롤러
제2 I/O 버스 컨트롤러
제3 I/O 버스 컨트롤러