(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2011년02월09일
(11) 등록번호 10-1012625
(24) 등록일자 2011년01월27일

(51) Int. Cl.
G06T 1/00 (2006.01)
(21) 출원번호 10-2008-7031222
(22) 출원일자(국제출원일자) 2007년05월25일
심사청구일자 2008년12월23일
(85) 번역문제출일자 2008년12월23일
(65) 공개번호 10-2009-0021286
(43) 공개일자 2009년03월02일
(86) 국제출원번호 PCT/US2007/069803
(87) 국제공개번호 WO 2007/140338
국제공개일자 2007년12월06일
(30) 우선권주장
11/441,696 2006년05월25일 미국(US)
(56) 선행기술조사문헌
WO2002015000 A2
전체 청구항 수 : 총 18 항

(73) 특허권자
퀄컴 인코포레이티드
미국 92121-1714 캘리포니아주 샌 디에고 모어하우스 드라이브 5775
(72) 발명자
부르드 알렉세이 브이
미국 92131 캘리포니아주 샌디에고 카미니토 아가디르 10310
두 윈
미국 92129 캘리포니아주 샌디에고 케이티디드 써클 12341
(뒷면에 계속)
(74) 대리인
특허법인코리아나

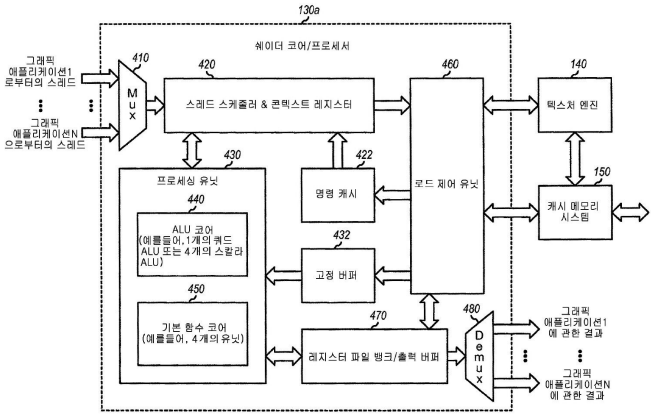
심사관 : 권성호

(54) 산술 유닛 및 기본 함수 유닛을 갖는 그래픽 프로세서

(57) 요 약

효율적으로 산술 연산을 수행하고 기본 함수를 계산할 수 있는 그래픽 프로세서가 설명되어 있다. 그래픽 프로세서는 산술 연산들을 수행할 수 있는 적어도 하나의 ALU(arithmetic logic unit) 및 기본 함수들을 계산할 수 있는 적어도 하나의 기본 함수 유닛을 갖는다. ALU(들) 및 기본 함수 유닛(들)은 이들이 병렬적으로 동작하여 스루풋을 개선할 수 있도록, 배열될 수도 있다. 그래픽 프로세서는 또한 ALU(들)보다 더 적은 기본 함수 유닛들, 예를 들어, 4개의 ALU(들)와 1개의 기본 함수 유닛을 포함할 수도 있다. 4개의 ALU는, (1) 1개의 픽셀에 관한 속성의 4개의 컴포넌트 또는 (2) 4개의 픽셀에 관한 속성의 1개의 컴포넌트에 대해 산술 연산을 수행할 수도 있다. 1개의 기본 함수 유닛은 한번에 1개의 픽셀의 1개의 컴포넌트에 대해 동작할 수도 있다. 1개의 기본 함수 유닛의 사용은 여전히 우수한 성능을 제공하면서, 비용을 감소시킬 수도 있다.

대 표 도



(72) 발명자

위 춘

미국 92131 캘리포니아주 샌디에고 싸이프레스 우즈 드라이브 11496

자오 궈팡

미국 92130 캘리포니아주 샌디에고 헌터스 글렌 드라이브 10680

특허청구의 범위

청구항 1

출력 버퍼;

산술 로직 유닛 (ALU) 명령들에 기초하여 산술 연산들을 수행하도록 동작하는 적어도 하나의 ALU; 및

기본 함수 유닛 명령들에 기초하여 기본 함수들을 계산하도록 동작하는 적어도 하나의 기본 함수 유닛을 포함하며,

상기 적어도 하나의 기본 함수 유닛 및 상기 적어도 하나의 ALU는 상기 출력 버퍼에 독립적으로 커플링되고 상이한 명령들에 대해 병렬적으로 동작하도록 구성되며,

상기 ALU 명령들은, 상기 기본 함수 유닛 명령들에 의존하는 상기 ALU 명령들이 그 의존하고 있는 기본 함수 유닛 명령들에 후속함을 보장하는 동기화 비트들을 포함하는, 그래픽 프로세싱을 수행하기 위한 장치.

청구항 2

제 1 항에 있어서,

4개의 ALU를 포함하는, 그래픽 프로세싱을 수행하기 위한 장치.

청구항 3

제 2 항에 있어서,

상기 4개의 ALU는 일 픽셀에 관한 속성의 최대 4개의 컴포넌트에 대해 산술 연산을 수행하도록 동작가능한, 그래픽 프로세싱을 수행하기 위한 장치.

청구항 4

제 2 항에 있어서,

상기 4개의 ALU는 최대 4개의 픽셀에 관한 속성의 일 컴포넌트에 대해 산술 연산을 수행하도록 동작가능한, 그래픽 프로세싱을 수행하기 위한 장치.

청구항 5

제 1 항에 있어서,

2 이상의 ALU 및 ALU보다 더 적은 기본 함수 유닛을 포함하는, 그래픽 프로세싱을 수행하기 위한 장치.

청구항 6

제 1 항에 있어서,

1개의 기본 함수 유닛을 포함하는, 그래픽 프로세싱을 수행하기 위한 장치.

청구항 7

삭제

청구항 8

삭제

청구항 9

삭제

청구항 10

제 1 항에 있어서,

상기 적어도 하나의 ALU 및 상기 적어도 하나의 기본 함수 유닛은 상이한 레이턴시를 갖는, 그래픽 프로세싱을 수행하기 위한 장치.

청구항 11

제 1 항에 있어서,

상기 적어도 하나의 ALU와 메모리 시스템 사이 및 상기 적어도 하나의 기본 함수 유닛과 상기 메모리 시스템 사이의 데이터 교환을 제어하도록 동작하는 로드 제어 유닛을 더 포함하는, 그래픽 프로세싱을 수행하기 위한 장치.

청구항 12

제 11 항에 있어서,

상기 적어도 하나의 기본 함수 유닛은 상기 로드 제어 유닛에 커플링되어 있는, 그래픽 프로세싱을 수행하기 위한 장치.

청구항 13

제 12 항에 있어서,

상기 적어도 하나의 기본 함수 유닛은 상기 로드 제어 유닛과 병렬적으로 동작가능한, 그래픽 프로세싱을 수행하기 위한 장치.

청구항 14

제 12 항에 있어서,

상기 적어도 하나의 기본 함수 유닛에 대한 요청 및 상기 로드 제어 유닛에 대한 로드 요청들은 버스를 공유하며,

상기 적어도 하나의 기본 함수 유닛 및 상기 로드 제어 유닛은 상이한 스레드를 병렬적으로 실행하도록 동작가능한, 그래픽 프로세싱을 수행하기 위한 장치.

청구항 15

제 1 항에 있어서,

적어도 하나의 그래픽 애플리케이션으로부터 스레드를 수신하도록 동작하며, 상기 적어도 하나의 ALU 및 상기 적어도 하나의 기본 함수 유닛에 의해 상기 스레드의 실행을 스케줄링하도록 동작하는 스케줄러를 더 포함하는, 그래픽 프로세싱을 수행하기 위한 장치.

청구항 16

제 15 항에 있어서,

상기 적어도 하나의 기본 함수 유닛은 상기 스케줄러에 커플링되어 있는, 그래픽 프로세싱을 수행하기 위한 장치.

청구항 17

삭제

청구항 18

출력 버퍼;

산술 로직 유닛 (ALU) 명령들에 기초하여 산술 연산들을 수행하도록 동작하는 적어도 하나의 ALU; 및

기본 함수 유닛 명령들에 기초하여 기본 함수들을 계산하도록 동작하는 적어도 하나의 기본 함수 유닛을 포함하

며,

상기 적어도 하나의 기본 함수 유닛 및 상기 적어도 하나의 ALU는 상기 출력 버퍼에 독립적으로 커플링되고 상이한 명령들에 대해 병렬적으로 동작하도록 구성되며,

상기 ALU 명령들은, 상기 기본 함수 유닛 명령들에 의존하는 상기 ALU 명령들이 그 의존하고 있는 기본 함수 유닛 명령들에 후속함을 보장하는 동기화 비트들을 포함하는, 그래픽 프로세싱을 수행하기 위한 집적 회로.

청구항 19

제 18 항에 있어서,

4개의 ALU 및 4개보다 적은 기본 함수 유닛을 포함하는, 그래픽 프로세싱을 수행하기 위한 집적 회로.

청구항 20

출력 버퍼, 산술 로직 유닛 (ALU) 명령들에 기초하여 산술 연산들을 수행하도록 동작하는 적어도 하나의 ALU, 및 기본 함수 유닛 명령들에 기초하여 기본 함수들을 계산하도록 동작하는 적어도 하나의 기본 함수 유닛을 포함하는 그래픽 프로세서로서, 상기 적어도 하나의 기본 함수 유닛 및 상기 적어도 하나의 ALU는 상기 출력 버퍼에 독립적으로 커플링되고 상이한 명령들에 대해 병렬적으로 동작하도록 구성되며, 상기 ALU 명령들은, 상기 기본 함수 유닛 명령들에 의존하는 상기 ALU 명령들이 그 의존하고 있는 기본 함수 유닛 명령들에 후속함을 보장하는 동기화 비트들을 포함하는, 상기 그래픽 프로세서; 및

상기 그래픽 프로세서를 위한 데이터를 저장하도록 동작하는 메모리 시스템을 포함하는, 그래픽 프로세싱을 수행하기 위한 무선 디바이스.

청구항 21

제 20 항에 있어서,

상기 그래픽 프로세서는 4개의 ALU 및 4개보다 적은 기본 함수 유닛을 포함하는, 그래픽 프로세싱을 수행하기 위한 무선 디바이스.

청구항 22

삭제

청구항 23

삭제

청구항 24

삭제

청구항 25

삭제

청구항 26

제 18 항에 있어서,

2 이상의 ALU 및 ALU보다 더 적은 기본 함수 유닛을 포함하는, 그래픽 프로세싱을 수행하기 위한 집적 회로.

명 세 서

[0001] **배경**

[0002] **Ⅰ. 기술분야**

[0003] 본 발명은 일반적으로 회로에 관한 것으로서, 더 구체적으로는 그래픽 프로세서에 관한 것이다.

[0004] **Ⅱ. 배경기술**

[0005] 그래픽 프로세서는 비디오 게임, 그래픽, CAD(computer-aided design), 시뮬레이션 및 시각화 툴, 이미징 등과 같은 다양한 애플리케이션을 위해 2-D(2-dimensional) 이미지 및 3-D(3-dimensional) 이미지를 렌더링(render)하는데 널리 사용된다. 3-D 이미지는 면들에 의해 모델링되며, 각각의 면은 다각형들(통상적으로 삼각형들)에 의해 근사화될 수도 있다. 3-D 이미지를 표현하는데 사용되는 삼각형들의 개수는 이미지의 원하는 해상도뿐 아니라 면의 복잡성에 따라 다르며, 예를 들어, 수백만 개 정도로 상당히 클 수도 있다. 각각의 삼각형은 3개의 꼭지점에 의해 정의되며, 각각의 꼭지점은 공간 좌표, 컬러 값, 텍스처 좌표와 같은 다양한 속성들과 연관되어 있다. 각각의 속성은 최대 4개의 컴포넌트를 가질 수도 있다.

[0006] 그래픽 프로세서는 이미지를 렌더링하기 위해 다양한 그래픽 연산들을 수행할 수도 있다. 그래픽 연산들은 래스터화(rasterization), 스텐실 및 깊이 테스트(stencil and depth tests), 텍스처 맵핑(texture mapping), 쉐이딩(shading) 등을 포함할 수도 있다. 이미지는 다수의 삼각형으로 구성되며, 각각의 삼각형은 화소(픽셀)로 구성된다. 그래픽 프로세서는 삼각형 내의 각각의 픽셀의 컴포넌트들의 값을 결정함으로써 각각의 삼각형을 렌더링한다.

[0007] 그래픽 프로세서는, 쉐이딩과 같은 특정 그래픽 연산을 수행하기 위해 쉐이더 코어(shader core)를 채택할 수도 있다. 쉐이딩은 라이팅(lighting), 쉐도잉(shadowing) 등을 포함하는 매우 복잡한 그래픽 연산이다. 쉐이더 코어는 사인, 코사인, 역수, 로그, 지수, 제곱근, 및 역제곱근과 같은 초월 기본 함수(transcendental elementary function)들을 계산하는데 필요할 수도 있다. 이들 기본 함수는 다항식 표현에 의해 근사화될 수도 있으며, 이는 산술 로직 유닛 (ALU) 에 의해 실행되는 상대적으로 간단한 명령들에 의해 값을 구할 수도 있다. 그러나, 쉐이더 성능은, ALU를 사용하는 이러한 방법으로 기본 함수들을 계산하기에는 매우 곤란하다.

[0008] **요약**

[0009] 여기서는, 효율적으로 산술 연산들을 수행하고, 기본 함수들을 계산할 수 있는 그래픽 프로세서를 설명한다. "연산"이라는 용어와 "함수"라는 용어는 종종 상호 호환적으로 사용된다. 그래픽 프로세서는 쉐이더 코어 및 가능한 다른 유닛들을 포함한다. 쉐이더 코어는 산술 연산들을 수행할 수 있는 적어도 하나의 ALU 및 기본 함수들을 계산할 수 있는 적어도 하나의 기본 함수 유닛을 갖는다. 일부 실시형태에서, ALU(들) 및 기본 함수 유닛(들)은 이들이 동일하거나 상이한 스레드에 관한 명령에 대해 병렬적으로 동작하여 스루풋을 개선할 수 있도록, 배열되거나 상호 연결되어 있다. 예를 들어, ALU(들)는 1개의 스레드에 관해 1개의 명령을 수행할 수도 있으며, 기본 함수 유닛(들)은 다른 스레드에 관해 다른 명령을 동시에 실행할 수도 있다. 이들 스레드는 동일하거나 상이한 그래픽 애플리케이션들을 위한 것일 수도 있다.

[0010] 다른 실시형태에서, 쉐이더 코어는 ALU(들)보다 더 적은 기본 함수 유닛들, 예를 들어, 4개의 ALU(들)와 1개의 기본 함수 유닛을 갖는다. 4개의 ALU는, (1) 1개의 픽셀에 관한 속성의 최대 4개의 컴포넌트 또는 (2) 최대 4개의 픽셀에 관한 속성의 1개의 컴포넌트에 대해 산술 연산을 수행할 수도 있다. 1개의 기본 함수 유닛은 한번에 1개의 픽셀의 1개의 컴포넌트에 대해 동작할 수도 있다. 1개의 기본 함수 유닛의 사용은, (기본 함수는 산술 연산보다 더 낮은 평균 사용율을 갖기 때문에) 여전히 우수한 성능을 제공하면서, (기본 함수 유닛이 ALU보다 더 복잡하고 비용이 들기 때문에) 비용을 감소시킬 수도 있다.

[0011] 이하, 본 발명의 다양한 양태 및 실시형태들을 더 상세하게 설명한다.

[0012] **도면의 간단한 설명**

[0013] 본 발명의 특징 및 특성은, 동일한 도면부호가 도면 전체에 대응적으로 식별되는 도면들과 결합하여 이해될 때, 이하 개시된 발명의 상세한 설명으로부터 더 명확해질 것이다.

[0014] 도 1은 그래픽 애플리케이션들을 지원하는 그래픽 프로세서를 도시한다.

[0015] 도 2는 픽셀의 속성 및 컴포넌트들을 나타낸다.

[0016] 도 3a는 4개의 스칼라 ALU에 의한 픽셀-병렬적 프로세싱을 도시한다.

[0017] 도 3b는 1개의 쿼드 ALU에 의한 컴포넌트-병렬적 프로세싱을 도시한다.

[0018] 도 4는 4-유닛 ALU 코어 및 4-유닛 EF(elementary function)를 갖는 쉐이더 코어를 도시한다.

[0019] 도 5는 병렬적인 ALU 코어 및 EF 코어를 갖는 쉐이더 코어를 도시한다.

[0020] 도 6은 4-유닛 ALU 코어 및 1-유닛 EF 코어를 갖는 쉐이더 코어를 도시한다.

[0021] 도 7은 그래픽 프로세서를 갖는 무선 디바이스의 블록도를 도시한다.

[0022] **발명의 상세한 설명**

[0023] 여기에서 "예시적인"이라는 단어는 "예, 예시, 또는 예증으로서 기능하는"이라는 의미로 사용된다. 여기에서 "예시적인"이라고 설명한 임의의 실시형태 또는 설계는 반드시 다른 실시형태들 또는 설계들에 대해 바람직하거나 유리한 것으로서 해석되는 것은 아니다.

[0024] 도 1은 N개의 그래픽 애플리케이션들/프로그램들(110a 내지 110n)을 지원하는 그래픽 시스템(100)의 블록도를 도시한 것이며, 여기서, 일반적으로, N ≥ 1이다. 그래픽 시스템(100)은 자립형 시스템일 수도 있고, 컴퓨터 시스템, 무선 통신 디바이스 등과 같은 더 큰 시스템의 일부일 수도 있다. 그래픽 애플리케이션들(110a 내지 110n)은 비디오 게임, 그래픽 등을 위한 것일 수도 있으며, 동시에 작동할 수도 있다. 각각의 그래픽 애플리케이션(110)은 원하는 결과를 달성하기 위해 스레드들을 생성할 수도 있다. 스레드(또는 스레드의 실행)는 하나 이상의 명령들의 시퀀스에 의해 수행될 수도 있는 특정 태스크를 가리킨다. 스레드들은 그래픽 애플리케이션으로 하여금 상이한 유닛들에 의해 동시에 수행되는 다수의 태스크를 갖게 하며, 또한, 상이한 그래픽 애플리케이션들로 하여금 리소스를 공유하게 한다.

[0025] 그래픽 프로세서(120)는 그래픽 애플리케이션들(110a 내지 110n)로부터 스레드들을 수신하고, 이들 스레드가 가리키는 태스크들을 수행한다. 도 1에 도시한 실시형태에서, 그래픽 프로세서(120)는 쉐이더 코어/프로세서(130), 텍스처 엔진(140), 및 캐시 메모리 시스템(150)을 포함한다. 일반적으로, 코어는 집적 회로 내의 프로세싱 유닛을 지칭한다. "코어", "엔진", "프로세서", 및 "프로세싱 유닛"이라는 용어들은 종종 상호 호환적으로 사용된다. 쉐이더 코어(130)는 쉐이딩과 같은 특정 그래픽 연산을 수행할 수도 있으며, 초월 기본 함수들을 계산할 수도 있다. 텍스처 엔진(140)은 텍스처 맵핑과 같은 다른 그래픽 연산들을 수행할 수도 있다. 캐시 메모리 시스템(150)은 하나 이상의 캐시를 포함하며, 이러한 캐시들은 쉐이더 코어(130) 및 텍스처 엔진(140)을 위한 데이터 및 명령들을 저장할 수 있는 고속 메모리이다.

[0026] 그래픽 프로세서(120)는 다른 프로세싱 유닛, 제어 유닛, 엔진, 및 메모리들을 포함할 수도 있다. 예를 들어, 그래픽 프로세서(120)는 삼각형 설정, 래스터화, 스텐실 및 깊이 테스트, 속성 설정, 및 픽셀 보간 등을 수행하는 하나 이상의 추가적인 엔진을 포함할 수도 있다. 여기에서 설명하는 다양한 그래픽 연산들은 당업계에 공지되어 있다. 추가적인 엔진(들)은 그래픽 애플리케이션들(110)과 쉐이더 코어(130) 사이에 커플링될 수도 있으며, 쉐이더 코어(130)에 커플링될 수도 있다. 그래픽 프로세서(120)는 OpenGL(Open Graphics Library), Direct3D 등과 같은 소프트웨어 인터페이스를 구현할 수도 있다. OpenGL은 공개적으로 입수가능한 2004년 10월 22일자 버전 2.0의 "The OpenGL® Graphics System: A Specification"이라는 명칭의 문서에 설명되어 있다.

[0027] 메인 메모리(160)는 그래픽 프로세서(120)로부터 멀리 떨어져서 위치하는(예를 들어, 오프-칩인) 용량이 큰 저속 메모리이다. 메인 메모리(160)는 캐시 메모리 시스템(150) 내의 캐시들로 로딩될 수도 있는 데이터 및 명령들을 저장한다.

[0028] 도 2는 픽셀의 속성 및 컴포넌트들을 나타낸다. 전술한 바와 같이, 2-D 이미지 또는 3-D 이미지는 다수의 삼각형들로 구성될 수도 있으며, 각각의 삼각형은 픽셀들로 구성될 수도 있다. 각각의 픽셀은 공간 좌표, 컬러 값, 텍스처 좌표 등과 같은 다양한 속성들을 가질 수도 있다. 각각의 속성은 최대 4개의 컴포넌트를 가질 수도 있다. 예를 들어, 공간 좌표는 수평 및 수직 좌표(x 및 y), 및 깊이(z)에 관한 3개의 컴포넌트로 주어지거나, x, y, z, 및 w에 관한 4개의 컴포넌트로 주어질 수도 있으며, 여기서, w는 동차 좌표(homogeneous coordinate)에 관한 4번째 항목이다. 동차 좌표는 변환(translation), 스케일링, 회전 등과 같은 특정 그래픽 연산들에 유용하다. 통상적으로, 컬러 값은 적색(r), 녹색(g), 청색(b)으로 주어진다. 통상적으로, 텍스처 좌표는 수평 및 수직 좌표(u 및 v)로 주어진다. 또한, 픽셀은 다른 속성들과 연관되어 있을 수도 있다.

[0029] 다수의 경우에, 렌더링될 이미지 내의 픽셀들의 그룹들에 대해 연산하는 것이 바람직하다. 그룹 사이즈는 하드웨어 요구사항, 성능 등과 같은 다양한 팩터에 기초하여 선택될 수도 있다. 2×2의 그룹 사이즈는 다양한 팩터들간의 우수한 트레이드오프를 제공할 수도 있다. 2×2 그리드 내의 4개의 픽셀에 대한 프로세싱은 몇몇 방법으로 수행될 수도 있다.

[0030] 도 3a는 4개의 동일한 스칼라 ALU들(ALU1 내지 ALU4)에 의한 4개의 픽셀들(픽셀 1 내지 픽셀 4) 각각에 대한 픽셀-병렬적 프로세싱을 도시한다. 이러한 예에서, 연산되는 속성의 4개의 컴포넌트는 $A_{p,1}$, $A_{p,2}$, $A_{p,3}$ 및 $A_{p,4}$라고 표시되며, 여기서, 픽셀 1 내지 픽셀 4에 관해 p는 픽셀 인덱스이고, $p \in \{1, 2, 3, 4\}$이다. 이들 컴포넌트는 공간 좌표, 컬러 값, 텍스처 좌표 등을 위한 것일 수도 있다. 4개의 컴포넌트에 적용되는 4개의 피연산자들은 $p \in \{1, 2, 3, 4\}$인 경우, $B_{p,1}$, $B_{p,2}$, $B_{p,3}$ 및 $B_{p,4}$라고 표시되며, 상수일 수도 있다. 이러한 예에서, ALU는 MAC(multiply and accumulate) 연산을 수행한다. 따라서, 각각의 픽셀의 4개의 컴포넌트는 4개의 피연산자와 곱해지며, 4개의 중간 결과가 누산되어 그 픽셀에 관한 최종 값을 생성한다.

[0031] 도 3a에서의 픽셀-병렬적 프로세싱의 경우, 각각의 스칼라 ALU는 1개의 픽셀의 4개의 컴포넌트에 대해 동작하며, 4개의 ALU는 4개의 픽셀에 대해 동시에 동작한다. ALU1는, 제1 클럭 주기 T_1에서 컴포넌트 $A_{1,1}$과 $B_{1,1}$을 곱하고, 그 다음, 제2 클럭 주기 T_2에서 컴포넌트 $A_{1,2}$과 $B_{1,2}$를 곱하여 이전 결과와 함께 이 결과를 누산하며, 그 다음, 제3 클럭 주기 T_3에서 컴포넌트 $A_{1,3}$과 $B_{1,3}$을 곱하여 이전 결과와 함께 이 결과를 누산하고, 그 다음, 제4 클럭 주기 T_4에서 컴포넌트 $A_{1,4}$와 $B_{1,4}$를 곱하여 이전 결과와 함께 이 결과를 누산한다. ALU2 내지 ALU4는 각각 픽셀 2 내지 픽셀 4의 컴포넌트에 대해 유사하게 동작한다.

[0032] 도 3b는 1개의 쿼드 ALU에 의한 4개의 픽셀에 대한 컴포넌트-병렬적 프로세싱을 도시하며, 쿼드 ALU는 또한 벡터-기반 ALU라고 지칭될 수도 있다. 컴포넌트-병렬적 프로세싱의 경우, 쿼드 ALU는 1개의 픽셀의 4개의 모든 컴포넌트에 대해 한번에 동작한다. 따라서, 쿼드 ALU는 컴포넌트 $A_{p,1}$, $A_{p,2}$, $A_{p,3}$ 및 $A_{p,4}$와 피연산자 $B_{p,1}$, $B_{p,2}$, $B_{p,3}$ 및 $B_{p,4}$를 각각 곱하며, 제1 클럭 주기 T_1에서, 4개의 중간 결과를 누산하여 제1 픽셀에 관한 최종 결과를 획득한다. 쿼드 ALU는, 클럭 주기 T_2, T_3, T_4에서 각각 제2 픽셀, 제3 픽셀, 제4 픽셀의 컴포넌트들에 대해 유사하게 동작한다.

[0033] 도 3a 및 도 3b는 최대 4개의 픽셀에 관한 속성의 최대 4개의 컴포넌트에 대한 쿼드 프로세싱을 수행하는 2가지 방식을 도시한다. 산술 연산에 관한 쿼드 프로세싱은 1개의 쿼드 ALU 또는 4개의 스칼라 ALU에 의해 수행될 수도 있다. 다음의 설명에서, ALU는 다른 표시가 없는 경우 스칼라 ALU라고 가정한다. 쿼드 프로세싱은 실질적으로 성능을 개선할 수도 있다. 따라서, 쿼드 프로세싱을 수행하는 능력을 갖는 쉐이더 코어(130)가 설계될 수도 있다.

[0034] 도 4는 4-유닛 ALU 코어(440) 및 4-유닛 기본 함수 코어(450)를 갖는 쉐이더 코어/프로세서(130a)의 일 실시형태의 블록도를 도시한다. 쉐이더 코어(130a)는 도 1에서의 쉐이더 코어(130)로 사용될 수도 있다.

[0035] 쉐이더 코어(130a) 내에, Mux(멀티플렉서; 410)는 그래픽 애플리케이션들(110a 내지 110n)로부터 스레드를 수신하며, 이들 스레드를 스레드 스케줄러 및 콘텍스트 레지스터(420)에게 제공한다. 스레드 스케줄러(420)는 스레드 실행을 스케줄링하고 관리하기 위해 다양한 함수들을 수행한다. 스레드 스케줄러(420)는 새로운 스레드들을 받아들여서, 각각의 받아들여진 스레드에 관한 레지스터 맵 테이블을 생성하고 그 스레드들에 리소스를 할당할지 여부를 결정한다. 레지스터 맵 테이블은 논리적 레지스터 어드레스와 물리적 레지스터 파일 어드레스 사이의 맵핑을 가리킨다. 각각의 스레드에 대해, 스레드 스케줄러(420)는 스레드에 관해 요구되는 리소스들이 준비되었는지 여부를 결정하여, 스레드에 관한 임의의 리소스(예를 들어, 명령, 레지스터 파일, 또는 텍스처 판독)가 준비되지 않는 경우, 스레드를 슬립 큐(sleep queue)로 밀어넣고, 모든 리소스들이 준비된 경우, 스레드를 슬립 큐로부터 액티브 큐로 이동시킨다. 스레드 스케줄러(420)는 스레드에 관한 리소스들을 동기화하기 위해 로드 제어 유닛(460)과 인터페이스한다.

[0036] 스레드 스케줄러(420)는 또한, 스레드들의 실행을 관리한다. 스레드 스케줄러(420)는 명령 캐시(422)로부터 각각의 스레드에 관한 명령(들)을 인출(fetch)하고, 필요하다면 각각의 명령을 디코딩하며, 스레드에 관한 플로우 제어를 수행한다. 스레드 스케줄러(420)는 실행을 위한 액티브 스레드를 선택하고, 선택된 스레드들 사이의 판독/기록 포트 충돌에 관해 체크하며, 충돌이 없는 경우, 1개의 스레드에 관한 명령(들)을 프로세싱 코어(430)로 전송하고, 다른 스레드들에 관한 명령(들)을 로드 제어 유닛(460)으로 전송한다. 스레드 스케줄러(420)는, 각각의 스레드에 관한 프로그램/명령 카운터를 보유하며, 명령들이 실행되거나 프로그램 플로우가 변경될 때, 이 카운터를 업데이트한다. 스레드 스케줄러(420)는 또한, 누락된 명령에 대한 인출 요청을 이슈하며, 완료된 스레드들을 제거한다.

[0037] 명령 캐시(422)는 스레드들을 위한 명령들을 저장한다. 이들 명령은 각각의 스레드를 위해 수행될 특정 연

산들을 가리킨다. 각각의 연산은 산술 연산, 기본 함수, 메모리 액세스 연산 등일 수도 있다. 명령 캐시(422)는 캐시 메모리 시스템(150) 및/또는, 필요한 경우, 로드 제어 유닛(460)을 통해 메인 메모리(160)로부터의 명령들에 의해 로딩될 수도 있다.

[0038] 도 4에 도시한 실시형태에서, 프로세싱 코어(430)는 ALU 코어(440) 및 기본 함수 코어(450)를 포함한다. ALU 코어(440)는 덧셈, 뺄셈, 곱셈, 곱셈 후 누산, 절대값, 부정, 비교, 포화 등과 같은 산술 연산들을 수행한다. ALU 코어(440)는 또한, AND, OR, XOR 등과 같은 로직 연산을 수행할 수도 있다. ALU 코어(440)는 또한, 예를 들어 정수로부터 부동 소수점 수로의 변환, 및 그 반대의 포맷 변환을 수행할 수도 있다. 도 4에 도시한 실시형태에서, ALU 코어(440)는 1개의 쿼드 ALU 또는 4개의 스칼라 ALU일 수도 있다. 도 3a에 도시한 바와 같이, ALU 코어(440)는 최대 4개의 픽셀들에 관한 속성의 1개의 컴포넌트에 대해 픽셀-병렬적 프로세싱을 수행할 수도 있다. 다른 방법으로, 도 3b에 도시한 바와 같이, ALU 코어(440)는 1개의 픽셀에 관한 속성의 최대 4개의 컴포넌트에 대해 컴포넌트-병렬적 프로세싱을 수행할 수도 있다.

[0039] 도 4에 도시한 실시형태에서, 기본 함수 코어(450)는 최대 4개의 픽셀에 관한 속성의 1개의 컴포넌트(픽셀-병렬적) 또는 1개의 픽셀에 관한 속성의 4개의 컴포넌트(컴포넌트-병렬적)에 관한 기본 함수를 계산할 수 있는 4개의 기본 함수 유닛으로 구성된다. 기본 함수 코어(450)는 쉐이더 명령들에서 널리 사용되는 사인, 코사인, 역수, 로그, 지수, 제곱근, 및 역제곱근 등과 같은 초월 기본 함수를 계산할 수도 있다. 기본 함수 코어(450)는, 간단한 명령들을 사용하여, 기본 함수들의 다항식 근사화를 수행하는데 요구되는 시간보다 훨씬 더 짧은 시간 내에 기본 함수들을 계산함으로써 쉐이더 성능을 개선할 수도 있다.

[0040] 로드 제어 유닛(460)은 쉐이더 코어(130a) 내의 다양한 유닛에 관한 데이터 및 명령들의 플로우를 제어한다. 로드 제어 유닛(460)은 캐시 메모리 시스템(150)과 인터페이스하며, 캐시 메모리 시스템(150)으로부터의 데이터 및 명령들을 명령 캐시(422), 고정 버퍼(432), 레지스터 파일 뱅크/출력 버퍼(470)에 로드한다. 로드 제어 유닛(460)은 또한, 출력 버퍼(470) 내의 데이터를 캐시 메모리 시스템(150)으로 저장한다. 로드 제어 유닛(460)은 또한, 명령들을 텍스처 엔진(140)에게 제공한다.

[0041] 고정 버퍼(432)는 ALU 코어(440)에 의해 사용되는 상수 값을 저장한다. 출력 버퍼(470)는 스레드에 관한 ALU 코어(440) 및 기본 함수 코어(450)로부터의 최종 결과뿐 아니라 임시 결과들을 저장한다. Demux(디멀티플렉서; 480)는 출력 버퍼(470)로부터 스레드의 실행에 관한 최종 결과를 수신하며 이들 결과를 그래픽 애플리케이션들에게 제공한다.

[0042] 도 4에 도시한 실시형태에서, 프로세싱 코어(430)는 ALU 코어(440) 및 기본 함수 코어(450) 모두를 포함한다. 이 실시형태는 ALU 코어(440) 및 기본 함수 코어(450)로 하여금 쉐이더 코어(130a) 내의 다른 유닛들(예를 들어, 스레드 스케줄러(420) 및 출력 버퍼(470))에 코어들(440 및 450)을 커플링하는 버스들을 공유하게 한다.

[0043] 일반적으로, 기본 함수 유닛은 ALU보다 더 복잡하다. 비용-효율적인 구현의 경우에도, 통상적으로, 기본 함수 유닛은 ALU보다 더 큰 회로 영역을 차지하며, 이에 따라, ALU보다 더 비싸다. 모든 쉐이더 명령에 대해 높은 쉐이더 스루풋을 달성하기 위해, 기본 함수 유닛의 개수는 ALU의 개수와 일치하도록 선택될 수도 있으며, 도 4에 도시한 실시형태에서는 4개이다. 그러나, 연구들은, 기본 함수들이 널리 사용되는 경우에도 기본 함수들의 평균 사용율은 ALU 연산들의 평균 사용율보다 상당히 낮다는 것을 보여주고 있다. 더 낮은 평균 사용율은, 산술 연산보다 기본 함수들에 의해 연산되는 컴포넌트들이 더 적을 뿐아니라 산술 연산보다 기본 함수들이 덜 호출되는 것으로부터 기인한다. 예를 들어, 통상적으로 기본 함수들은 산술 연산들보다 덜 호출되며, 이에 따라, 더 적은 기본 함수 유닛들에 의해 적절히 지원될 수도 있다. 또한, 4개의 ALU를 갖는 것으로부터 이익을 얻는 경우로서 속성의 4개의 모든 컴포넌트들에 대해 덧셈 또는 곱셈을 수행하는 것은 일반적일 수도 있지만, 4개의 모든 컴포넌트들에 대해 기본 함수를 수행하는 것은 덜 일반적이다. 따라서, 기본 함수들이 컴포넌트들의 서브세트(예를 들어, 1 또는 2개의 컴포넌트들)에 대해서만 수행되는 다수의 경우에, 더 적은 기본 함수 유닛들은 우수한 성능을 제공할 수도 있다. 더 적은 기본 함수 유닛들을 구현하는 것은 여전히 우수한 성능을 제공하면서 비용을 감소시킬 수도 있다.

[0044] 도 5는 4-유닛 ALU 코어(540) 및 L-유닛 기본 함수 코어(550)를 갖는 쉐이더 코어(130b)의 일 실시형태의 블록도를 도시하며, 여기서, $1 \le L < 4$이다. 쉐이더 코어(130b)는 또한, 도 1에서의 쉐이더 코어(130)로 사용될 수도 있다. 쉐이더 코어(130b)는 도 4에서의 유닛들(410, 420, 422, 432, 440, 450, 460, 470, 및 480)과 각각 유사한 방법으로 동작하는 멀티플렉서(510), 스레드 스케줄러 및 콘텍스트 레지스터(520), 명령 캐시(522), 고정 버퍼(532), ALU 코어(540), 기본 함수 코어(550), 로드 제어 유닛(560), 레지스터 파일 뱅크/출력 버퍼(570), 디멀티플렉서(580)를 포함한다.

[0045] ALU 코어(540)는 1개의 쿼드 ALU이거나 4개의 스칼라 ALU일 수도 있다. ALU 코어(540)는 일 세트의 버스를 통해 스레드 스케줄러(520), 고정 버퍼(532), 및 출력 버퍼(570)와 커플링되어 있다. 기본 함수 코어(550)는 1개의 픽셀의 L개의 컴포넌트 또는 L개의 픽셀의 1개의 컴포넌트에 관해 기본 함수를 계산할 수 있는 1개, 2개, 또는 3개의(L개의) 기본 함수 유닛들로 구성될 수도 있다. 기본 함수 코어(550)는 다른 세트의 버스를 통해 스레드 스케줄러(520), 고정 버퍼(532), 및 출력 버퍼(570)에 커플링되어 있다. 도 5에 도시한 실시형태에서, ALU 코어(540) 및 기본 함수 코어(550)는 서로 별도로 구현되며, 별도의 버스들을 통해 쉐이더 코어(130b) 내의 다른 유닛들에 커플링된다. ALU 코어(540) 및 기본 함수 코어(550)는 상이한 명령들에 대해 병렬적으로 동작할 수도 있다. 이들 명령은 동일하거나 상이한 그래픽 애플리케이션들을 위한 것일 수도 있다.

[0046] 도 5에 도시한 실시형태에서, 기본 함수 유닛의 개수는, ALU의 개수보다 적으며, 비용과 성능 간의 트레이드오프에 기초하여 선택될 수도 있다. 다수의 경우에, 기본 함수 코어(550)는 기본 함수의 더 낮은 사용율로 인해 ALU 코어(540)와 보조를 맞출 수 있을 것이다. 스레드 스케줄러(520)는 (4 대신) L개의 기본 함수 유닛들이 이용가능하다는 지식에 의해 기본 함수 연산들을 적절히 스케줄링한다.

[0047] 도 6은 4-유닛 ALU 코어(640) 및 1-유닛 기본 함수 코어(650)를 갖는 쉐이더 코어/프로세서(130c)의 일 실시형태의 블록도를 도시한다. 쉐이더 코어(130c)는 또한, 도 1에서의 쉐이더 코어(130)로 사용될 수도 있다. 쉐이더 코어(130c)는 도 4에서의 유닛들(410, 420, 422, 432, 440, 450, 460, 470, 및 480)과 각각 유사한 방법으로 동작하는 멀티플렉서(610), 스레드 스케줄러 및 콘텍스트 레지스터(620), 명령 캐시(622), 고정 버퍼(632), ALU 코어(640), 기본 함수 코어(650), 로드 제어 유닛(660), 레지스터 파일 뱅크/출력 버퍼(670), 디멀티플렉서(680)를 포함한다.

[0048] ALU 코어(640)는 1개의 쿼드 ALU이거나 4개의 스칼라 ALU일 수도 있다. ALU 코어(640)는 일 세트의 버스를 통해 스레드 스케줄러(620), 고정 버퍼(632), 및 출력 버퍼(670)와 커플링되어 있다. 기본 함수 코어(650)는 한번에 1개의 픽셀의 1개의 컴포넌트에 관해 기본 함수를 계산할 수 있는 1개의 기본 함수 유닛으로 구성될 수도 있다. 도 6에 도시한 실시형태에서, 기본 함수 코어(650)는 로드 제어 유닛(660) 및 출력 버퍼(670)에 커플링되어 있다. 이러한 실시형태는 별도의 ALU 코어(640) 및 기본 함수 코어(650)를 지원하는 버스들의 개수를 감소시킨다. 이 실시형태는 또한, 레지스터 파일 판독/기록 포트, 명령 디코딩 등과 같은 더 효율적인 리소스 공유와 같은 다른 이익들을 제공할 수도 있다.

[0049] 쉐이더 코어(130c) 내의 기본 함수 코어(650)의 배치뿐 아니라 설계가 주어지면, 기본 함수들의 명령들(또는 EF 명령들)이 적절한 방법으로 생성될 수도 있다. (도 4에 도시한 바와 같이) EF 유닛의 개수가 ALU 유닛의 개수와 동일한 경우와 EF 유닛이 ALU 유닛과 동일한 파이프라인 레이턴시를 갖는 경우에, EF 명령들은 예측가능한 파이프라인 지연을 갖는 ALU 명령들로 취급될 수도 있다. 그러나, ALU 코어(640) 및 기본 함수 코어(650)의 불규칙한 구현은 불규칙한 스루풋을 초래한다. 따라서, 일 실시형태에서, 쉐이더 코어(130c)는 기본 함수 코어(650)를 로드 리소스로서 취급하며, 예를 들어, 텍스처 로드 또는 메모리 로드와 동일한 동기화에 의해, 그리고, 이들과 유사한 방법으로 EF 명령들을 프로세싱한다. 예를 들어, 쉐이더 컴파일러는 ALU 명령들 대신에 텍스처 로드에 관한 명령들로서 EF 명령들을 컴파일할 수도 있으며, 이는 도 4 및 도 5에 도시한 실시형태에서의 경우일 수도 있다.

[0050] 쉐이더 컴파일러는 명령들 내에 sync(동기화) 비트를 적절하게 포함할 수도 있다. sync 비트는, sync 비트를 포함하는 현재의 명령이 하나 이상의 이전 명령들과의 데이터 의존성을 갖는다는 것을 가리킬 수도 있으며, 이는 예측불가능한 지연 또는 레이턴시를 가질 수도 있다. 예측불가능한 레이턴시는 일부 소스들로 인한 것일 수도 있다. 첫째, 텍스처 로드 또는 메모리 로드의 예측불가능한 레이턴시는 캐시 적중/누락, 메모리 액세스 능력, 메모리 액세스 시퀀스 등과 같은 예측불가능한 실행 조건들로부터 발생할 수도 있다. 둘째, 예측불가능한 레이턴시는 ALU 코어 및 기본 함수 코어의 불규칙한 구현들에 의해 발생할 수도 있다. 쉐이더 컴파일러는 이전 EF 명령들과 데이터 의존성을 갖는 명령들 내에 sync 비트를 삽입할 수도 있으며, 이는 예측불가능한 지연을 가질 수도 있다. 이들 sync 비트는 그 명령들이 이들의 의존적인 EF 명령들에 후속하고, 이에 따라, 적당한 데이터에 대해 동작한다는 것을 보장한다.

[0051] 도 6에 도시한 실시형태에서, 스레드 스케줄러(620)는, 데이터 로드 요청과 버스를 공유할 수도 있는 기본 함수 요청을 생성할 수도 있다. 이러한 공유 버스는 스레드 스케줄러(620)로부터 로드 제어 유닛(660)으로의 버스를 포함할 수도 있다. 그러나, 기본 함수 코어(650)는 로드 제어 유닛(660) 내의 로드 명령들과 병렬적으로 실행될 수도 있다. 다른 실시형태에서, 기본 함수 코어(650)는, 예를 들어, 도 5에 도시한 바와 같은 전

용 버스를 통해 스레드 스케줄러(620)로 직접 커플링될 수도 있다. 이 실시형태에서, 기본 함수 요청 및 데이터 로드 요청은 별도의 버스들을 사용할 수도 있다. 양 실시형태에서, 스레드 스케줄러(620), ALU 코어(640), 기본 함수 코어(650), 및 로드 제어 유닛(660)은 성능 개선을 위해 상이한 스레드에 대해 병렬적으로 동작할 수도 있다.

[0052] 도 4 내지 도 6은 쉐이더 코어들(130a, 130b, 및 130c)의 특정 실시형태들을 도시한다. 쉐이더 코어들(130a, 130b, 및 130c)의 다른 변형들 또한 가능하다. 예를 들어, 도 4에서의 기본 함수 코어(450)는 4 보다 더 적은 개수의 기본 함수 유닛들을 포함할 수도 있다. 다른 실시형태로서, 도 6에서의 기본 함수 코어(650)는 2개 이상의 기본 함수 유닛, 예를 들어, 2개의 기본 함수 유닛을 포함할 수도 있다.

[0053] 일반적으로, 쉐이더 코어는 임의의 개수의 프로세싱 유닛, 제어 유닛, 메모리 유닛을 포함할 수도 있으며, 이들은 임의의 방법으로 배열될 수도 있다. 이들 유닛은 또한, 다른 명칭으로 지칭될 수도 있다. 예를 들어, 로드 제어 유닛은 또한, I/O(입력/출력) 인터페이스 유닛으로 지칭될 수도 있다. 일부 실시형태에서, 성능을 약간 디그레이드하면서 비용을 감소시키기 위해, 쉐이더 코어는 ALU보다 더 적은 기본 함수 유닛들을 포함할 수도 있다. 다른 실시형태에서, 쉐이더 코어는 동일하거나 상이한 그래픽 애플리케이션을 위한 상이한 명령들에 대해 병렬적으로 동작할 수 있는 별도의 ALU 코어 및 기본 함수 코어를 포함할 수도 있다. ALU 및 기본 함수 유닛은 당업계에 공지된 다양한 설계에 의해 구현될 수도 있다. 쉐이더 코어는 또한, 동기화 인터페이스 및/또는 비동기화 인터페이스를 통해 외부 유닛들과 인터페이스할 수도 있다.

[0054] 여기에서 설명한 그래픽 프로세서 및 쉐이더 코어들은 무선 통신, 컴퓨팅, 네트워킹, 개인 전자장비 등에 사용될 수도 있다. 이하, 무선 통신을 위한 그래픽 프로세서의 예시적인 사용을 설명한다.

[0055] 도 7은 무선 통신 시스템에서의 무선 디바이스(700)의 일 실시형태의 블록도이다. 무선 디바이스(700)는 셀룰러 전화기, 단말기, 핸드셋, PDA(personal digital assistant) 또는 기타 다른 디바이스일 수도 있다. 무선 통신 시스템은 CDMA(Code Division Multiple Access) 시스템, GSM(Global System for Mobile Communications) 시스템, 또는 기타 다른 시스템일 수도 있다.

[0056] 무선 디바이스(700)는 수신 경로 및 송신 경로를 통해 양-방향 통신을 제공할 수 있다. 수신 경로에서, 기지국들에 의해 송신된 신호들은 안테나(712)에 의해 수신되어, 수신기(RCVR; 714)에게 제공된다. 수신기(714)는 수신 신호를 컨디셔닝하고 디지털화하여, 더 프로세싱하기 위한 디지털 섹션(720)에게 샘플들을 제공한다. 송신 경로에서, 송신기(TMTR; 716)는 디지털 섹션(720)으로부터 송신된 데이터를 수신하고, 그 데이터를 프로세싱하고 컨디셔닝하여, 변조 신호를 생성하며, 이는 안테나(712)를 통해 기지국으로 송신된다.

[0057] 디지털 섹션(720)은 예를 들어, 모뎀 프로세서(722), 비디오 프로세서(724), 애플리케이션 프로세서(726), 디스플레이 프로세서(728), 제어기/프로세서(730), 그래픽 프로세서(740), 및 EBI(external bus interface; 760)와 같은 다양한 프로세싱 유닛 및 인터페이스 유닛들을 포함한다. 모뎀 프로세서(722)는 데이터 송신 및 수신을 위해 프로세싱(예를 들어, 인코딩, 변조, 복조, 및 디코딩)을 수행한다. 비디오 프로세서(724)는 캠코더, 비디오 재생, 및 비디오 회의와 같은 비디오 애플리케이션을 위한 비디오 콘텐츠(예를 들어, 스틸 이미지, 동영상, 및 이동 텍스트)에 대해 프로세싱을 수행한다. 애플리케이션 프로세서(726)는 멀티-웨이 호출, 웹 브라우징, 미디어 플레이어, 및 사용자 인터페이스와 같은 다양한 애플리케이션들을 위한 프로세싱을 수행한다. 디스플레이 프로세서(728)는 디스플레이 유닛(780)에서 비디오, 그래픽, 및 텍스트들의 디스플레이를 용이하게 하는 프로세싱을 수행한다. 제어기/프로세서(730)는 디지털 섹션(720) 내의 다양한 프로세싱 유닛 및 인터페이스 유닛들의 동작을 지시할 수도 있다.

[0058] 그래픽 프로세서(740)는 그래픽 애플리케이션을 위한 프로세싱을 수행하며, 전술한 바와 같이 구현될 수도 있다. 예를 들어, 그래픽 프로세서(740)는 도 1에서의 쉐이더 코어/프로세서(130) 및 텍스처 엔진(140)을 포함할 수도 있다. 캐시 메모리 시스템(750)은 그래픽 프로세서(740)를 위한 데이터 및/또는 명령들을 저장한다. 캐시 메모리 시스템(750)은 (1) 그래픽 프로세서(740) 내의 상이한 엔진들에 할당될 수도 있는 구성가능한 캐시 및/또는 (2) 특정 엔진들에 할당된 전용 캐시들로 구현될 수도 있다. EBI(760)는 디지털 섹션(720; 예를 들어, 캐시)과 메인 메모리(770) 사이의 데이터 전송을 용이하게 한다.

[0059] 디지털 섹션(720)은 하나 이상의 DSP(digital signal processor), 마이크로-프로세서, RISC(reduced instruction set computer) 등으로 구현될 수도 있다. 디지털 섹션(720)은 또한, 하나 이상의 ASIC(application specific integrated circuit) 또는 기타 다른 타입의 IC(integrated circuit) 상에 제작될 수도 있다.

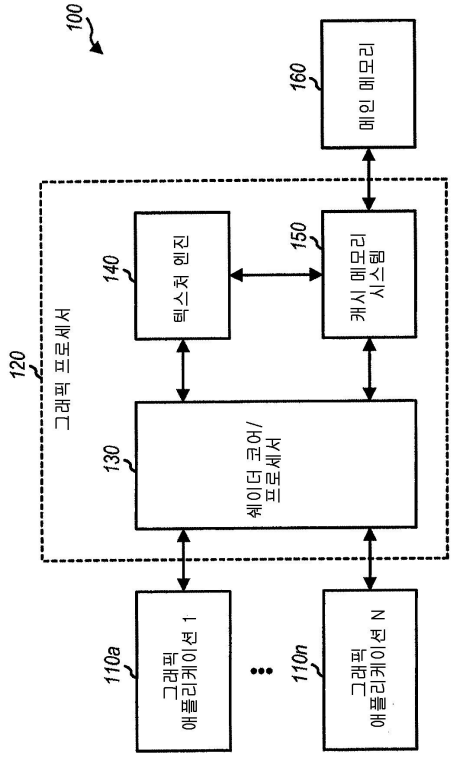
[0060] 여기에서 설명한 그래픽 프로세서 및 쉐이더 코어/프로세서는 다양한 하드웨어 유닛들로 구현될 수도 있다. 예를 들어, 그래픽 시스템들 및 쉐이더 코어/프로세서들은 ASIC, DSP(digital signal processor), DSPD(digital signal processing device), PLD(programmable logic device), FPGA(field programmable gate array), 프로세서, 제어기, 마이크로-제어기, 마이크로프로세서, 및 다른 전자장비 유닛들로 구현될 수도 있다.

[0061] 그래픽 프로세서들의 특정 부분은 펌웨어 및/또는 소프트웨어로 구현될 수도 있다. 예를 들어, 스레드 스케줄러 및/또는 로드 제어 유닛은 펌웨어 및/또는 소프트웨어 모듈들(예를 들어, 절차, 함수 등)로 구현될 수도 있다. 펌웨어 코드 및/또는 소프트웨어 코드는 메모리(예를 들어, 도 7에서의 메모리(750 또는 770)) 내에 저장될 수도 있으며, 프로세서(예를 들어, 프로세서(730))에 의해 실행될 수도 있다. 메모리는 프로세서 내에 구현되거나, 프로세서 외부에 구현될 수도 있다.

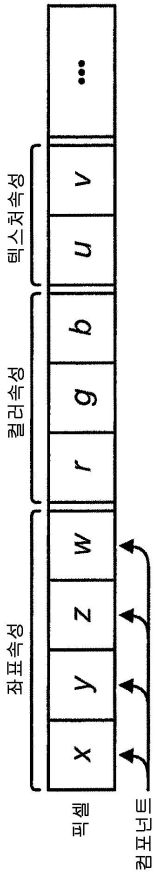
[0062] 개시된 실시형태의 상기 설명은 당업자로 하여금 본 발명을 실시하고 사용할 수 있게 하기 위해 제공된다. 이들 실시형태에 대한 다양한 변형은 당업자에게 용이하게 명백할 것이며, 본 발명의 사상 및 범위를 벗어나지 않으면, 여기에 정의된 특징적인 원리들은 다른 실시형태들에 적용될 수도 있다. 따라서, 본 발명을, 여기에 도시한 실시형태들로 제한하려는 것이 아니라, 여기에 설명한 원리 및 신규한 특징들에 부합하는 최광의 범위에 일치시키려는 것이다.

도면

도면1



도면2



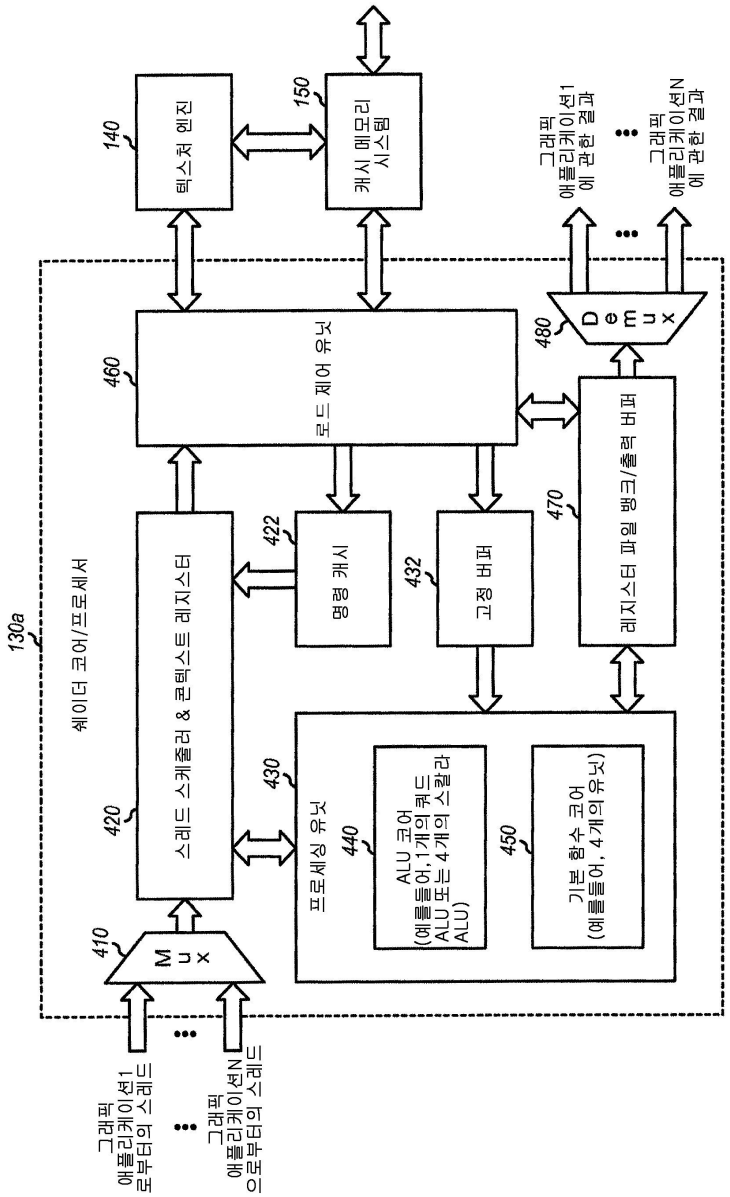
도면3a

클럭 주기	스칼라 ALU1 (픽셀 1)	스칼라 ALU2 (픽셀 2)	스칼라 ALU3 (픽셀 3)	스칼라 ALU4 (픽셀 4)
T_1	$A_{1,1} \cdot B_{1,1}$	$A_{2,1} \cdot B_{2,1}$	$A_{3,1} \cdot B_{3,1}$	$A_{4,1} \cdot B_{4,1}$
T_2	$+ A_{1,2} \cdot B_{1,2}$	$+ A_{2,2} \cdot B_{2,2}$	$+ A_{3,2} \cdot B_{3,2}$	$+ A_{4,2} \cdot B_{4,2}$
T_3	$+ A_{1,3} \cdot B_{1,3}$	$+ A_{2,3} \cdot B_{2,3}$	$+ A_{3,3} \cdot B_{3,3}$	$+ A_{4,3} \cdot B_{4,3}$
T_4	$+ A_{1,4} \cdot B_{1,4}$	$+ A_{2,4} \cdot B_{2,4}$	$+ A_{3,4} \cdot B_{3,4}$	$+ A_{4,4} \cdot B_{4,4}$

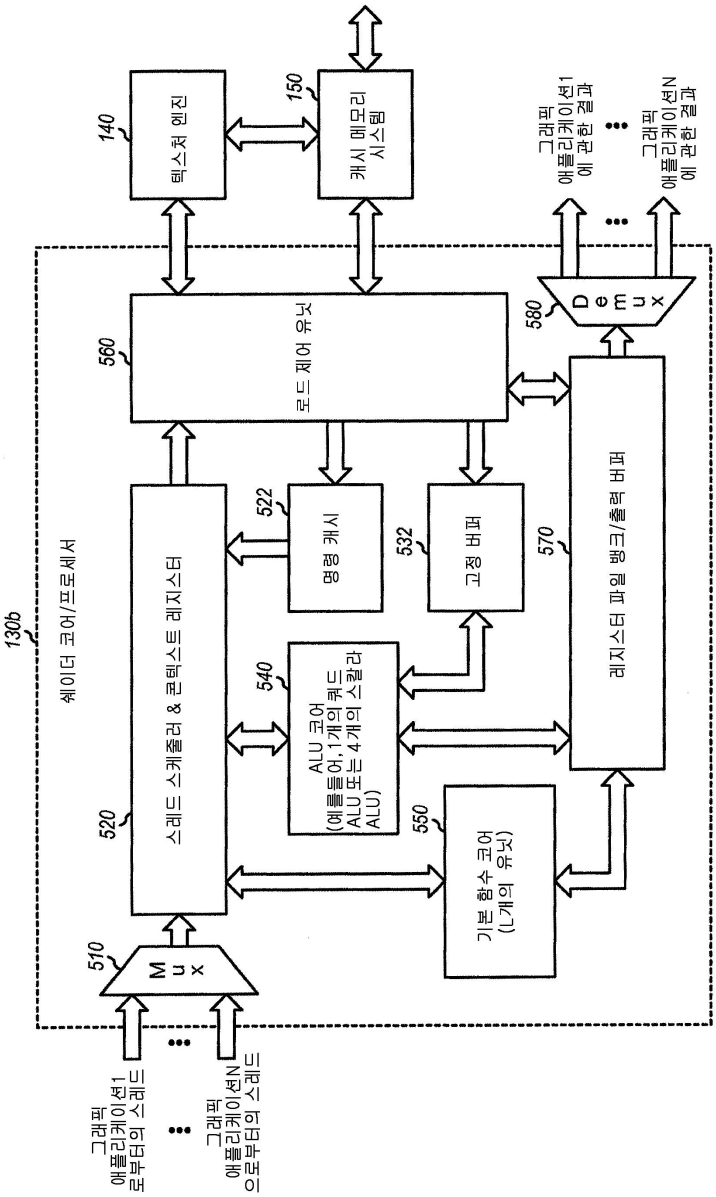
도면3b

클럭 주기	쿼드 ALU (픽셀들 1-4)
T_1	$A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{1,2} + A_{1,3} \cdot B_{1,3} + A_{1,4} \cdot B_{1,4}$
T_2	$A_{2,1} \cdot B_{2,1} + A_{2,2} \cdot B_{2,2} + A_{2,3} \cdot B_{2,3} + A_{2,4} \cdot B_{2,4}$
T_3	$A_{3,1} \cdot B_{3,1} + A_{3,2} \cdot B_{3,2} + A_{3,3} \cdot B_{3,3} + A_{3,4} \cdot B_{3,4}$
T_4	$A_{4,1} \cdot B_{4,1} + A_{4,2} \cdot B_{4,2} + A_{4,3} \cdot B_{4,3} + A_{4,4} \cdot B_{4,4}$

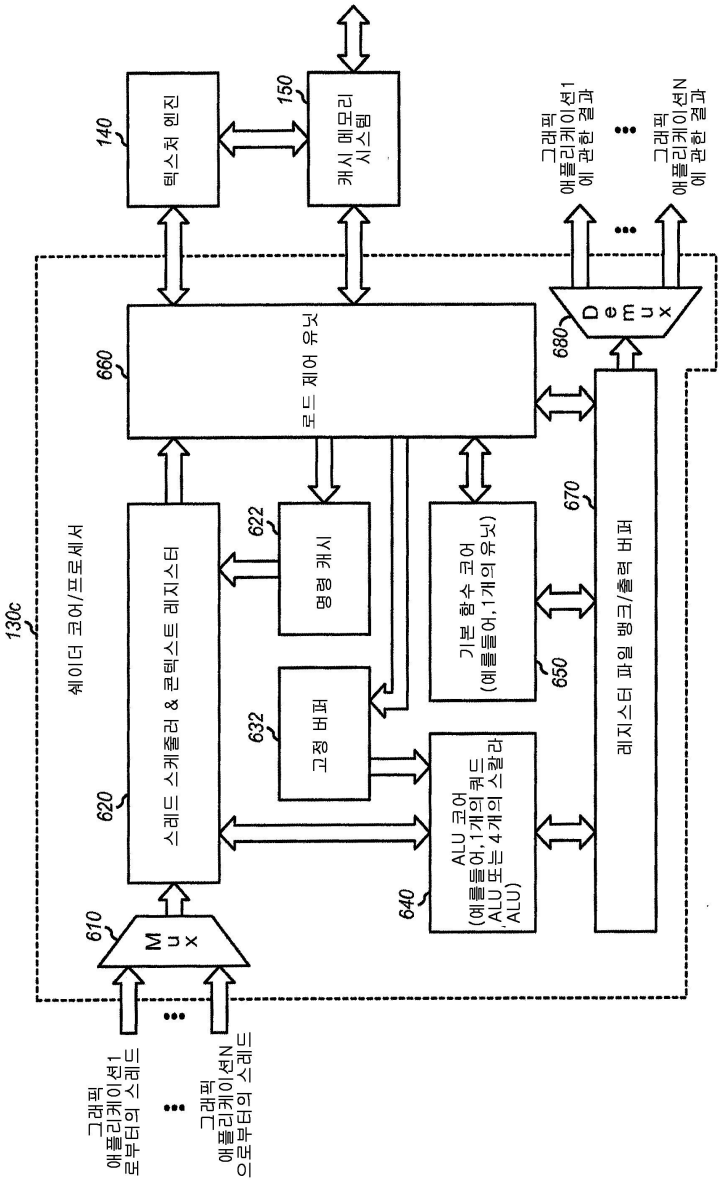
도면4



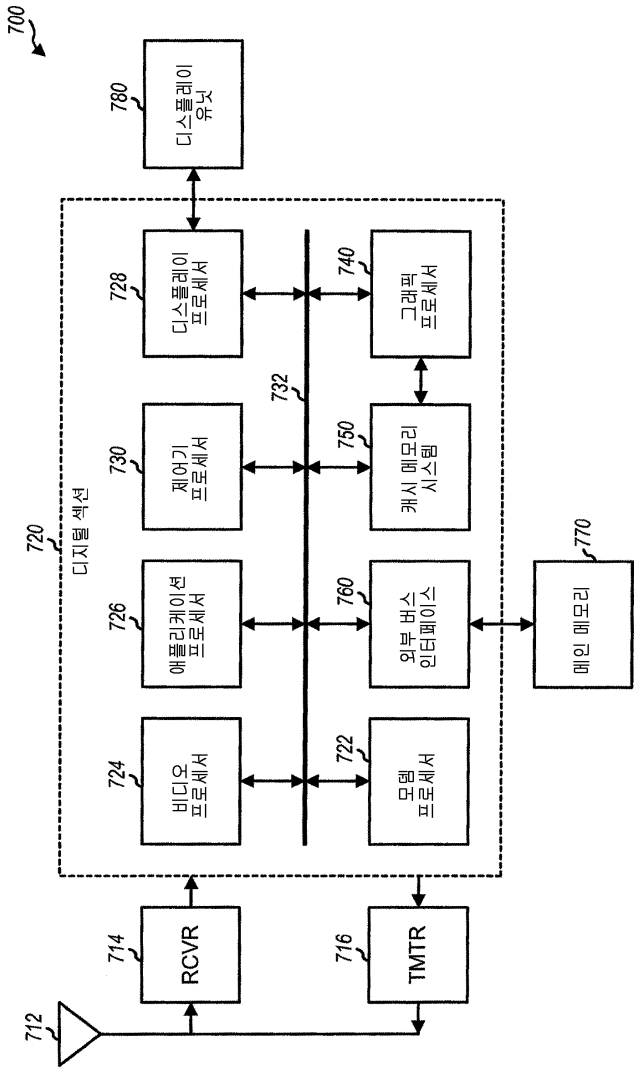
도면5


텍스쳐 엔진
140
캐시 메모리 시스템
150
그래픽 애플리케이션1에 관한 결과
그래픽 애플리케이션N에 관한 결과
Demux
580
로드 제어 유닛
560
쉐이더 코어/프로세서
130b
스레드 스케줄러 & 콘텍스트 레지스터
520
명령 캐시
522
고정 버퍼
532
레지스터 파일 뱅크/출력 버퍼
570
ALU 코어
(예를들어, 1개의 쿼드 ALU 또는 4개의 스칼라 ALU)
540
기본 함수 코어
(L개의 유닛)
550
Mux
510
그래픽 애플리케이션1로부터의 스레드
그래픽 애플리케이션N으로부터의 스레드

도면6


130c
쉐이더 코어/프로세서
텍스처 엔진
140
캐시 메모리 시스템
150
그래픽 애플리케이션1 에 관한 절과
그래픽 애플리케이션N 에 관한 절과
Demux
680
로드 제어 유닛
660
스레드 스케줄러 & 콘텍스트 레지스터
620
명령 캐시
622
고정 버퍼
632
기본 함수 코어 (예를들어,1개의 유닛)
650
ALU 코어 (예를들어,1개의 쿼드 ALU 또는 4개의 스칼라 ALU)
640
레지스터 파일 뱅크/출력 버퍼
670
Mux
610
그래픽 애플리케이션1 로부터의 스레드
그래픽 애플리케이션N 으로부터의 스레드

도면7


700
780
디스플레이 유닛
720
디지털 섹션
728
디스플레이 프로세서
740
그래픽 프로세서
732
730
제어기 프로세서
750
캐시 메모리 시스템
726
애플리케이션 프로세서
760
외부 버스 인터페이스
770
메인 메모리
724
비디오 프로세서
722
모뎀 프로세서
714
RCVR
716
TMTR
712