논문 2008-45SD-9-13

SIMD 프로그래머블 셰이더를 위한 멀티포트 레지스터 파일 설계 및 구현

(Multi-Port Register File Design and Implementation for the SIMD Programmable Shader)

윤 완 오*, 김 경 섭*, 정 진 하*, 최 상 방**

(Wan-oh Yoon, Kyeong-seob Kim, Jin-ha Cheong, and Sang-bang Choi)

요 약

3D 그래픽 알고리즘은 특성상 방대한 양의 스트림 데이터에 대하여 복잡한 연산을 수행하여야 한다. 이러한 알고리즘을 하드웨어에서 신속하게 수행할 수 있는 버텍스 셰이더와 픽셀 셰이더의 도입으로 그래픽 프로세서는 "소프트웨어 셰이더의 하드웨어화"라는 목표를 어느 정도 달성한 것처럼 보이지만, 여전히 Z-버퍼 기반이라는 특정 알고리즘의 틀에서 벗어나지 못하고 있다. 향후 그래픽 프로세서가 궁극적으로 추구하는 모델은 알고리즘에 독립적인 그리고 버텍스 셰이더와 픽셀 셰이더가 통합된 셰이더로 발전할 것이다. 본 논문에서는 프로그래머블 통합 셰이더 프로세서에서 고성능 3차원 컴퓨터 그래픽 영상을 지원하기 위한 멀티포트 레지스터 파일 모델을 설계하고 구현하였다. 설계한 멀티포트 레지스터 파일을 기능적 레벨에서 시뮬레이션을 하여 그 성능을 검증 하였으며, FPGA Virtex-4(xc4vlx200)에 직접 구현하여 하드웨어 리소스 사용율과 속도를 확인 하였다.

Abstract

Characteristically, 3D graphic algorithms have to perform complex calculations on massive amount of stream data. The vertex and pixel shaders have enabled efficient execution of graphic algorithms by hardware, and these graphic processors may seem to have achieved the aim of "hardwarization of software shaders." However, the hardware shaders have hitherto been evolving within the limits of Z-buffer based algorithms. We predict that the ultimate model for future graphic processors will be an algorithm-independent integrated shader which combines the functions of both vertex and pixel shaders. We design the register file model that supports 3-dimensional computer graphic on the programmable unified shader processor. we have verified the accurate calculated value using FPGA Virtex-4(xcvlx200) made by Xilinx for operating binary files made by the implementation progress based on synthesis results.

Keywords : Graphic processor, Register file, shader processor, HDL, FPGA

I. 서 론

최근 컴퓨터 산업의 눈부신 발전으로 인하여 3차원 그래픽 기술의 사용은 특정 분야에 국한된 것이 아니라 게임, 영상, 교육, 출판, 군사, 의료 등 모든 분야에서 걸쳐서 그 영역이 확대 되고 있다. 3차원 컴퓨터 그래픽은 멀티미디어 환경을 구축하기위한 가장 핵심적인 연구 분야일 뿐만 아니라 컴퓨터를 기반으로 하는 대부분의 응용분야에서는 3차원 그래픽 기술이 필수적으로 요구된다. 현실감 있는 3차원 컴퓨터 그래픽 영상을 지원하기 위해서는 방대한 양의 데이터와 복잡한 연산을 효율적으로 처리할 수 있는 고성능 그래픽 프로세서 기술

* 학생회원, ** 평생회원 인하대학교 전자공학과
(Dept. of Electronic Engineering, Inha University)
※ 본 연구는 지식경제부 및 정보통신연구진흥원의 IT 신성장동력핵심기술개발사업의 일환으로 수행하였음. [2006-S-044-02, 멀티코아 CPU 및 MPU 기반 크로스플랫폼 게임기술]
접수일자: 2008년3월20일, 수정완료일: 2008년8월26일

이 요구된다[1~7]. 그동안 연구된 그래픽 프로세서는 고정된 기능에 한해 지오메트리(geometry) 연산과 랜더링(rendering)을 수행할 수 있고, 리얼리티를 높이기 위한 대부분의 랜더링 작업은 소프트웨어 단의 지원을 받는 비교적 간단한 알고리즘만을 구현해 왔으나 최근에는 대부분의 랜더링 작업을 그래픽 프로세서에서 처리할 수 있도록 프로그래머블 셰이더를 구현하여 유연하고 정밀한 알고리즘을 하드웨어 단에서 구동하도록 연구되고 있다[8].

프로그래머블 셰이더가 구현된 고성능의 그래픽 프로세서는 원활한 데이터 처리를 위해 연산 유닛에서 연산된 결과를 효과적으로 저장하고 출력하는 과정이 프로세서의 성능향상에 많은 영향을 주고 있다[9~11]. 따라서 본 논문에서는 프로그래머블 통합 셰이더 프로세서에서 고성능 3차원 컴퓨터 그래픽 영상을 지원하기 위한 멀티포트 레지스터 파일을 설계하고 구현 하였다. 기능적 레벨 시뮬레이션을 통하여 설계된 레지스터 파일의 효율적인 동작을 검증하였으며 FPGA Vertex-4에 구현하여 최적화된 하드웨어 리소스의 사용율과 성능을 확인하였다.

본 논문은 다음과 같이 구성된다. Ⅱ장에서는 그래픽 프로세서의 특징 및 기존 연구에 대해서 설명하고, 프로그래머블 셰이더의 전체적인 구성과 특징에 대해서 살펴본다. Ⅲ장에서는 프로그래머블 통합 셰이더 프로세서의 구조를 제시하는 동시에 이 구조를 기반으로 레지스터 파일의 모델을 제안하고, Ⅳ장에서는 하나의 레지스터 파일에 대한 다중 레지스터로의 확장 및 그에 따른 내부 로직에 대한 설계를 다룬다. Ⅴ장에서는 HDL 합성 후 기능적 레벨 시뮬레이션을 통해 제안된 레지스터 파일 모델을 검증하였다. 마지막으로 Ⅵ장에서는 본 논문의 결론 및 향후 연구 과제를 제시하며 끝을 맺는다.

Ⅱ. 관련 연구

본 장에서는 그래픽 프로세서의 특징 및 기존 연구에 대해서 설명하고, 프로그래머블 셰이더의 전체적인 구성과 특징에 대해서 설명한다.

2.1. 그래픽 프로세서의 특징 및 기존 연구

일반적으로 3차원 그래픽은 연산 집약적인 특징을 갖는다. 하나의 영상은 지오메트리 연산과 렌더링 연산을 거쳐 완성된다. 지오메트리 연산이란 기하학적 형태를 나타내는 과정이고 렌더링은 조명이나 주변 사물에 의한 효과를 고려하여 색과 음영을 입히는 것이다. 특히 렌더링 과정은 다양한 효과를 반영하기위해 많은 연산을 요구하는 복잡한 알고리즘을 필요로 한다. 3차원 그래픽은 영상을 구성하기위해 많은 양의 데이터를 필요로 하는 특징을 갖는다. 그러므로 처리 속도를 빠르게 하기위해 각각의 데이터에 대해 병렬로 수행될 필요가 있다. 즉, 영상을 구성하는 각 픽셀은 {x, y, z, w}로 구성되는 위치정보와 {R, G, B, (투명도)}로 구성되는 색채 정보로 표현된다. 따라서 명령어는 각각의 데이터에 대해 4가지 정보를 처리해야 하므로 이를 위해 동시에 여러 개의 데이터를 처리할 수 있는 구조를 갖는 프로세서가 효율적이다. 또한, 그래픽 프로그램의 특성상 특정 사물의 움직임을 표현하는 과정이나, 개별 픽셀의 위치를 동일한 패턴으로 이동시키는 연산을 자주 요구하기 때문에 여러 픽셀 데이터에 대해서 동일한 연산을 취해야 하는 경우가 대부분이다. 따라서 이러한 3차원 그래픽 처리의 특성을 이용하여 성능을 극대화시키기 위해 많은 데이터에 대해 연속적으로 연산을 수행할 수 있는 스트림 구조에 대한 연구가 진행되어 왔다. 그림 1은 스트림 구조를 이용한 대표적인 프로세서인 스탠포드 대학의 이미진 프로세서(imagine processor)를 보여준다. 이미진 프로세서의 특징은 스트림 구조로 데이터를 처리하기 위하여 8개의 ALU(arithmetic logic unit) 클러스터를 이용하여 최대 8개의 픽셀 데이터를 동시에 처리할 수 있도록 하였다. 각 ALU 클러스터는 독립적으로 명령어를 수행할 수 있어, 마치 8개의 프로세서를 병렬로 연결한 것과 같이 동작함으로써 데이터를 효율적으로 처리한다. 그러나 동일한 명령어를 처리한다 하더라도 다른 데이터에 대해서는 또다시 동일한 명령어 셋을 패치하여 ALU가 수행되는 방식이므로 동일한 명령어를 반복해서 패치하는 과정이 요구된다[12].

그림 2는 최근 유럽에서 제안된 ATTILA의 셰이더 구조를 보여준다. ATTILA는 4개의 스레드를 하나로 묶어 병렬처리 하는 방식으로, 각 스레드는 4개의 데이터를 동시에 처리할 수 있는 SIMD(single instruction multiple data)구조로 되어있다. 이는 한 번의 명령어 패치로 4개의 스레드를 동시에 수행함으로써 효율을 높인다. 명령어 세트는 다중 명령어로 구성되는 VLIW(very long instruction word) 구조에 기반을 두어 동시에 여러 개의 스레드로 연산을 수행하는 것을 가능하게 하였다. 이는 명령어 세트의 수를 줄임으로써 전체적인 패치시간을 줄이고 있다[13].

이와 같은 기존의 3차원 그래픽 프로세서들의 특징은 SIMD, 슈퍼스칼라, VLIW, 벡터프로세서 등의 다양한 병렬처리 기술을 이용하였으나 고가의 칩을 다수 사용하여 칩 사용 면적이 크거나 아니면 적은 칩 사용 크기 대신에 그래픽 처리하는 부분을 소프트웨어적으로 설계하여 속도와 성능 면에서 부족하다는 단점이 있다. 또한 다양한 3차원 그래픽 알고리즘을 처리하기에는 부족하여 일부 자주 사용되는 알고리즘만을 처리한다.

그러나 사용자가 3차원 영상에 만족하는 것을 넘어서 실사수준의 영상을 요구하다보니 그래픽 프로세서의 설계 패러다임도 바뀌고 있다. 이에 최근에 개발되고 있는 그래픽 프로세서는 3차원 영상처리를 위해 필요한 모든 알고리즘을 수행할 수 있는 프로그래머블 셰이더로 구현되고 있다.

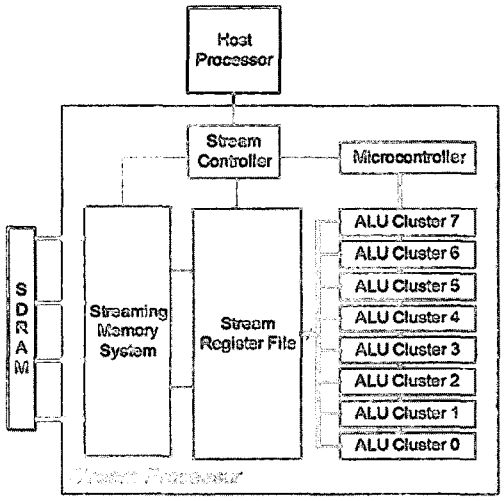

그림 1. 이미진 프로세서 구조
Fig. 1. Imagine processor architecture.

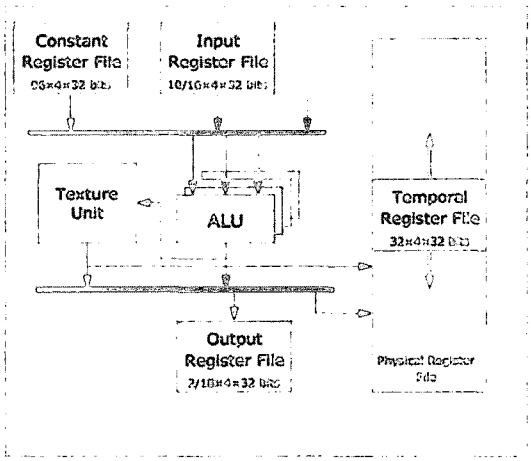

그림 2. ATTILA의 셰이더 구조
Fig. 2. The shader architecture of ATTILA.

2.2. 프로그래머블 셰이더의 구조

프로그래밍 가능한 셰이더에는 정점 셰이더와 픽셀 셰이더 두 종류가 있다. 두 종류의 셰이더는 목적이 완전히 다른 별개의 것이다. 각 셰이더의 차이는 표 1과 같으며 각 세이더 모델은 그림 3, 4와 같다. 정점 셰이더는 응용으로부터 받은 입력 레지스터의 정점 데이터 v#을 가공해서, 출력 레지스터 o#에 결과를 출력한다. 입력되는 데이터는 1개의 정점에 관한 위치 좌표나 법선 벡터 등의 정보이다. 출력되는 데이터도 똑같이 정점에 관한 좌표나 색이 된다. 단 정점의 개수를 늘리거나 줄일 수 없다.

픽셀 셰이더는 정점 셰이더에서 출력되는 정점 색이나 텍스처 좌표를 사용하여 화면에 출력되는 하나하나의 화소 색을 계산한다. 픽셀 셰이더에서는 최대 4개의

표 1. 정점 셰이더와 픽셀 셰이더 비교
Table 1. Compare vertex shader with pixel shader.

명칭	정점 셰이더	픽셀 셰이더
주목적	폴리곤 정점 좌표를 변환해서 화면에서의 위치 산출	스크린에 표시되는 픽셀 색을 결정
입력데이터	정점의 위치 좌표값	랜더링되는 픽셀의 색과 텍스처좌표
출력데이터	투명 좌표계에서 위치좌표	화면에 출력될 색
랜더링 공정	주사선 변환전	주사선 변환 후, 어떤 픽셀에 출력될지 결정된 후
분기 명령	O	X(삼항 연산자 : 정도는 가능)
랜더링 위치 변경	O	X
텍스처 읽기	△(변위 맵핑에서 제한적으로 가능)	O
포그 색 변경	O	X

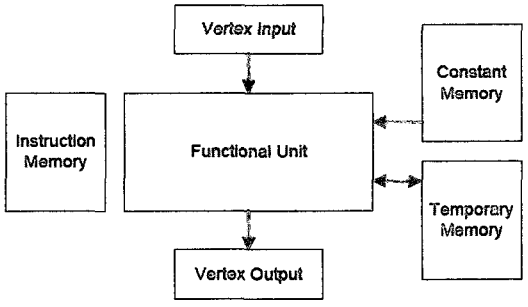

그림 3. 정점 셰이더 모델
Fig. 3. Vertex shader model.

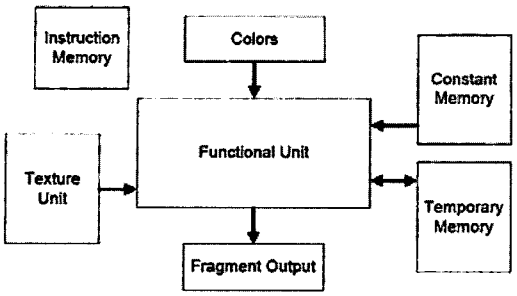

그림 4. 픽셀 셰이더의 모델
Fig. 4. Pixel shader model.

색과 1개의 깊이 값을 출력 할 수 있다. 즉, 4개의 프레임 버퍼 혹은 텍스처, 깊이 버퍼에 각각 다른 색을 동시에 출력할 수 있다. 실제로는 픽셀 셰이더 후에 반 투명 합성이나 포그 이펙트의 합성이 들어가므로 최종적인 색은 아닐 수 있지만 표시하려는 폴리곤 색은 픽셀 셰이더에 의해 거의 완벽하게 제어 할 수 있다[14~15].

III. SIMD 프로그래머블 셰이더 프로세서 구조 및 레지스터 파일 모델

본 장에서는 SIMD 프로그래머블 셰이더 프로세서의 구조를 제시하고, 이 구조를 기반으로 레지스터 파일의 구조를 제안한다.

3.1. SIMD 프로그래머블 셰이더 프로세서 구조

본 논문에서 제시하는 SIMD 프로그래머블 셰이더 프로세서는 기본적으로 자일링스 FPGA칩 중 하나인 Virtex-4에서의 구현을 목표로 설계되었다. 따라서 본 절에서는 자일링스 FPGA를 기본으로 SIMD 프로그래머블 셰이더 프로세서의 기본 아키텍쳐를 제시하기로 한다. 본 논문의 기반 아키텍처인 SIMD 프로그래머블 셰이더 프로세서는 성능향상을 위해 파이프라인과 병렬처리의 장점을 모은 것으로, 슈퍼스칼라 프로세서 구조와[16] 벡터 프로세서 구조를[17] 혼합한 것이다. 여기서 슈퍼스칼라 프로세서는 파이프라인을 사용하는 일반 프로세서처럼 여러 개의 명령어를 시간적으로 중첩시켜 실행하지만 동시에 독립적인 여러 개의 파이프라인 유닛을 사용하여 공간적으로 명령어를 수행하는 구조로 여러 개의 파이프라인에서 동시에 명령어들이 병렬로 수행하는 파이프라인 된 4개의 연산유닛과 8개의 이슈 유닛(issue unit)을 둠으로써 동시에 명령어들이 병렬로 수행되도록 하였다. 또한 대부분의 그래픽 프로시저가 개개의 픽셀에 대한 동일한 연산을 수행하는 경우가 대

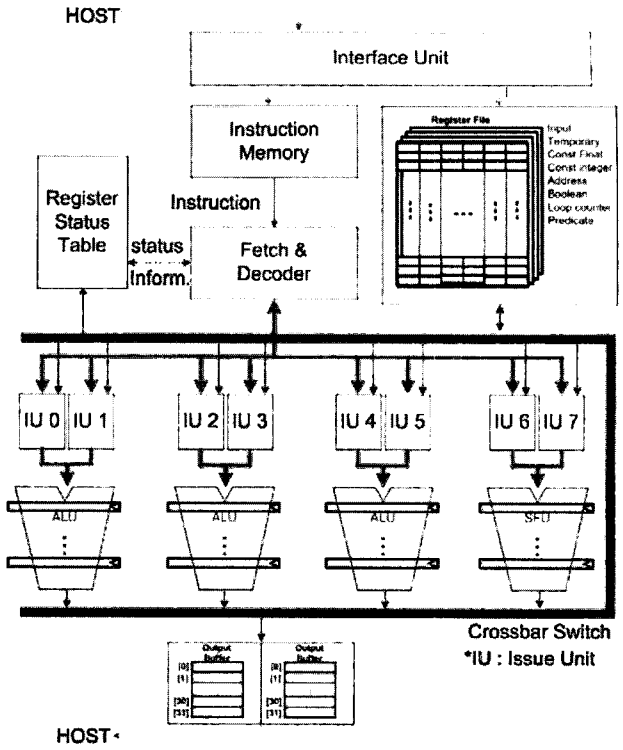

그림 5. SIMD 프로그래머블 셰이더 프로세서의 구조
Fig. 5. The architecture of SIMD programmable shader processor.

부분을 차지하기 때문에 많은 데이터가 연속적으로 하나의 명령어로 동일한 연산을 수행하는 벡터 프로세서 구조를 이용하여 32개의 데이터를 하나의 명령어로 동일한 연산을 수행하도록 32개의 데이터를 하나의 번들(bundle)로 묶어서 처리하였다. 이 두 구조를 혼합 사용한 SIMD 프로그래머블 셰이더 프로세서의 전체 구조는 그림 5와 같다.

상위 제어부는 크게 명령어 메모리, 명령어 패치/디코더, 레지스터 상태표로 나누어진다. 이 제어 부분은 실제 연산부 제어에 필요한 제어신호를 발생하는 부분이다. 하위 제어부는 SIMD 그래픽 프로세서에서 ALU 또는 SFU RAW 해저드를 검사하고 원활한 파이프라인을 동작하기 위해서 각각 2개의 이슈 유닛을 가지고 있다. 연산유닛은 ALU(arithmetic logic unit), SFU (special function unit), 레지스터 파일로 구분되며 입력 데이터는 IEEE-754 단정도 부동소수점 형식이다. ALU는 일반적인 산술연산을 수행하고, SFU는 ALU에서 실행하지 못하는 복잡한 연산을 수행하며, 레지스터 파일은 다중의 입출력이 있는 멀티포트 방식으로 데이터를 처리한다. ALU는 이슈 유닛으로부터 제어정보, 소스값, 그리고 목적지 값을 받아와서 abs, add, sub, crs, dp2, dp3, dp4, mad, lpr, min, max 등의 연산을 실행한다.

SFU는 ALU와 마찬가지로 이슈 유닛으로부터 제어정보, 소스값, 그리고 목적지 값을 받아와서 SQRT, RCP, DST, FRC, LOG, LIT, POW등의 연산을 실행한다. 레지스터 파일은 임시 레지스터, 입력 레지스터, 출력 레지스터 등으로 구성된다. 또한 이슈 유닛으로부터 연산에 필요한 레지스터 정보와 주소를 받아서 그에 해당하는 레지스터가 주소를 받아서 데이터를 이슈 유닛에 다시 보내주게 된다.

3.2 확장 가능한 레지스터 파일 모델

일반적인 레지스터 파일과는 다르게 SIMD 프로그래머블 셰이더 프로세서와 같은 구조의 레지스터 파일은 기본적으로 다중 수행 구조로 되어있기 때문에, 다중 읽기 및 쓰기 포트를 지녀야 하고 그에 따른 포트간의 선택 및 실제 데이터를 가져오는데 걸리는 액세스 타임의 증가를 초래한다. 실제적으로 이러한 레지스터 파일 구조가 다중 이슈 프로세서들에 있어서 설계상의 큰 어려움 중의 하나로 작용하고 있으며, 레지스터 파일의 다이나믹 포트할당 방식 등에 대한 연구를 필요로 한다. 그러한 이유로 레지스터 파일의 구현 시에 풀 커스텀 설계를 통하여 좀 더 빠른 액세스 타임을 지니도록 여러 가지 방법으로 설계가 되어지지만, 본 논문에서는 구현을 하고자하는 SIMD 프로그래머블 통합 셰이더 프로세서를 위한 레지스터 파일의 구조를 위해서 자일링스 FPGA에서 제공하는 IP(intellectual property)를 이용하여 레지스터 파일 모델을 제안 한다.

현재 자일링스 FPGA에서 제공하는 IP의 블록램은

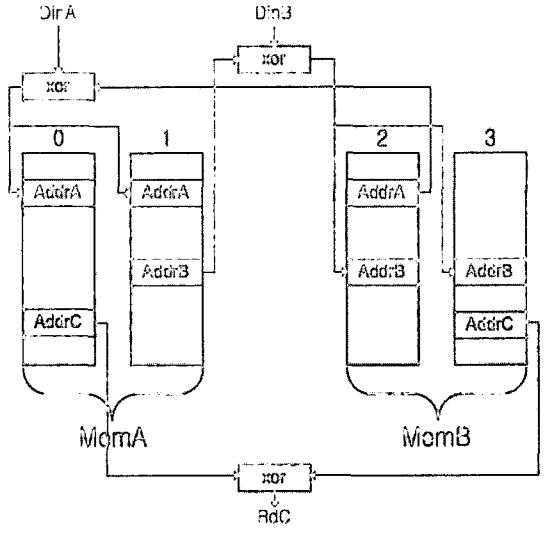

그림 6. 읽기포트 1개와 쓰기포트 2개를 포함하는 2W1R 레지스터 파일
Fig. 6. Register file with 1 read-port and 2 write-port.

포트 당 입출력 신호가 각각 한 개씩 존재하고 듀얼 포트를 지원한다. 그러나 한 개의 포트에서 동시에 입출력을 수행 할 수 없고 듀얼포트를 이용하더라도 입출력 신호를 각 포트에서 동시에 하나만 사용 할 수 있다. 이러한 구조는 동일한 레지스터를 동시에 여러개의 읽기와 쓰기를 필요로 하는 프로그래머블 통합 셰이더 프로세서의 구조를 위해 사용하기에는 무리가 있다. 따라서 단일 싸이클에 2개의 쓰기(2W)와 1개의 읽기(1R)가 동시에 가능한 기본적인 구조를 FPGA에서 제공하는 듀얼 포트 블록램을 사용해 하나의 블록으로 구성한 개선된 2W1R 블록램 구조를 그림 6과 같이 제안한다. 그림 6의 레지스터 파일에서의 2개의 쓰기와 1개의 읽기에 대한 동작은 다음과 같다.

```
if(WriteEnableMemA)
    MemA[AddrA] := MemB[AddrA] ⊕ DinA;
endif;

if(WriteEnableMemB)
    MemB[AddrB] := MemA[AddrB] ⊕ DinB;
endif;

if(ReadEnableMemAddC)
    Rdc = MemA[AddrC] ⊕ MemB[AddrC];
        = (MemB[AddrA] ⊕ DinA)
          ⊕ MemB[AddrA];
        = DinA ⊕ (MemB[AddrA]
          ⊕ MemB[AddrA]);
        = DinA;
endif;
```

2W1R 레지스터 파일 모델은 4개의 블록램을 사용한다.쓰기 포트와 읽기 포트를 확장 할 경우 필요한 블록램의 개수는 식 1과 같이 구할 수 있다.

$$BlockRamNum = 2*(WritePortNum) + (ReadPortNum-1)*WritePortNum \quad (1)$$

각 레지스터별로 독립적인 레지스터 파일이 존재하는 구조로 구성하기 위해 표 2와 같이 SIMD 프로그래머블 셰이더 프로세서를 위한 입출력 레지스터들을 정의한다. 표 2의 구성을 살펴보면 레지스터 종류는 입력과 출력으로 나누어지고, 레지스터는 해당하는 레지스터의 기호를 나타내며 기호 뒤의 '#' 표시는 레지스터가 여러개 존재한다는 의미이다.

레지스터명은 각각의 기호에 대한 실제 기능을 나타내고, Count는 번들을 곱해준 수 만큼의 크기를 갖는 레지스터를 갖는다. read/write는 해당하는 레지스터의 읽기 및 쓰기 속성이 무엇인지를 나타내고, read port#

표 2. 셰이더 프로세서를 위한 입출력 레지스터
Table 2. Input/output register for shader processor.

레지스터 종류	레지스터	레지스터명	Count (*Bundle)	Read/ Write	Read ports#
입력 레지스터	v#	Input Register	16(*B)	R	1
	r#	Temporary Register	32(*B)	R/W	3
	c0	Constant Float	1(*B)	R	1
	a0	Address Register	1(*B)	R/W	1
	b#	Boolean Register	16(*B)	R	1
	i0	Constant Integer	1(*B)	R	1
	aL	Loop Counter	1(*B)	R	1
	p0	Predicate Register	1(*B)	R/W	1
출력 레지스터	o#	Output Register	12(*B)	W	-

는 읽기 포트의 개수를 의미한다. 이러한 입출력 레지스터들을 정의함으로써 각 레지스터별로 독립적인 레지스터 파일을 구성할 수 있다.

IV. SIMD 프로그래머블 셰이더 프로세서를 위한 레지스터 파일의 설계

4.1 레지스터 파일의 설계

프로그래머블 통합 셰이더 프로세서를 위한 레지스터 파일의 전체적인 구조를 결정하기 위해 그래픽 프로그램에서 자주 사용되는 특정 프로시져에 대한 벤치마크 시뮬레이션을 수행 하였다. 그 결과 표 3과 같이 반복된 DP(dot product), MAD(multiply and add) 연산이 코드의 주를 이루고 있고 DP와 MAD 연산을 하기 위한 유닛이 많을수록 IPC(instruction per cycle)가 증가하는 것을 확인 하였다. 그러나 연산 유닛이 일정 수 이상 늘어날 경우 반복된 DP, MAD 연산이 코드의 주를 이루기 때문에 하드웨어 사용대비 IPC 증가량이 크게 오르지 않았다. 추가적으로, 데이터 의존성 문제로 인한 지연을 최소화하기 위해 연산유닛 앞에 두 개의 이슈 유닛을 두고 시뮬레이션을 수행 하였다. 이것 또한 마찬가지로, 이슈 유닛을 무한정 증가시키더라도 한정된 연산유닛으로 인해 명령어 처리 시간이 길어질 경우 이

표 3. 샘플 소스 코드를 이용한 벤치마크 시뮬레이션 결과
Table 3. The simulation result using sample source code.

메트릭	샘플 소스 코드	다중 포트 읽기 동작	
		IU:2, ALU:3, SFU:1	IU:2, ALU:2, SFU:1
IPC	HDRLighing / pixelmotionblur	4.188	2.386
	parallaxocclusion mapping / depthoffield	3.539	2.547
	textbump	4.320	2.600

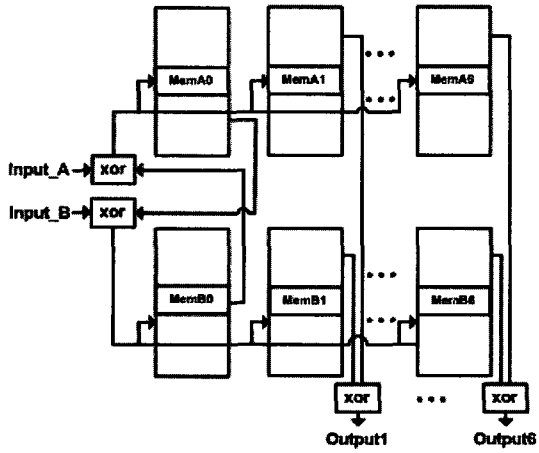

그림 7. 레지스터 파일 모델을 이용한 확장된 블록램
Fig. 7. The extended block RAM using register file model.

슈 유닛에서 대기해야 하므로 성능향상이 거의 없었다. 그러나 레지스터에서 동시에 데이터를 여러 개를 읽어오는 경우 성능향상이 있었고, 특정한 연산유닛에서 나온 결과가 레지스터에 쓰여 지기 전에 다른 연산유닛에서 곧바로 사용될 경우 성능향상이 있는 것을 확인 하였다.

따라서 벤치마크 시뮬레이션 결과를 토대로 프로그래머블 통합 셰이더 프로세서의 연산 유닛을 4개로 결정하여 설계 하였다. 표 2에서 가장 많은 포트를 필요로 하는 레지스터는 r# 레지스터로 3개의 읽기 포트와 1개의 쓰기 포트를 필요로 한다. 따라서 하나의 연산유닛이 레지스터에 접근 하기 위해서는 3개의 읽기 포트와 1개의 쓰기 포트가 필요하고 4개의 연산 유닛이 동시에 접근하기 위해서는 쓰기 포트 4개와 읽기 포트 12개가 필요하다. 이러한 포트의 수에 맞게 설계하는데 필요한 블록램의 개수는 식 1에 의해 52개의 블록램이 필요하다. vertex4의 전체 블록램의 개수는 336개이다. 따라서 r# 레지스터 하나를 구성하기 위한 블록램 사용율은 전체 블록램의 15.4%이다. 이것은 하드웨어를

너무 많이 사용하는 결과를 초래한다. 이러한 이유로 본 논문에서는 쓰기 포트 2(2W)개와 읽기 포트 6(6R)개로 구성되는 레지스터 파일과 시간 분할 매커니즘을 이용하여 4개의 연산 유닛이 동시에 레지스터 파일에 접근 하는 것을 가능 하도록 설계하였다. 그림 7은 2W6R의 포트를 가지도록 설계된 확장된 레지스터 파일의 모델이다. 2W6R의 포트를 가지는 레지스터 파일의 블록램 사용율은 14개로 전체 블록램의 4.17%을 사용하게 된다.

4.2 레지스터 파일의 시간 분할 매커니즘

2W6R로 결정된 레지스터 포트 사이즈를 기반으로 파이프라인 된 4개의 연산 유닛에서 레지스터를 동시에 참조하고, 각각의 명령어에 대해 네 가지의 픽셀정보를 포함한 다수의 오퍼랜드를 처리하는 구조인 SIMD 프로세서를 위해 동시 참조가 가능하도록 그림 8와 같이 시간 분할 매커니즘을 사용한다. 시간 분할 매커니즘을 사용하기 위해서는 레지스터 파일 동작을 위한 클럭신호와 전체 프로세서 동작을 위한 클럭신호가 있어야 하며 두 신호는 서로 동기화 되어 있어야 한다. 따라서 두 개의 클럭 신호를 발생시키고 클럭 동기화를 위한 제어 클럭 발생회로와 4개의 연산 유닛이 2W6R의 포트로 이루어진 레지스터 파일을 동시에 읽고 쓸수 있도록 입출력 버퍼회로를 구성하였다. 2W6R의 포트 사이즈를 가지는 레지스터 파일과 시간 분할 매커니즘을 적용한 레지스터 파일의 내부 동작을 살펴보면 쓰기 동작 시에는 그림 8의 a와 같이 4개의 연산 유닛으로부터 최대 4개의 쓰기 데이터를 받게 되면 2개는 해당 레지스터 파일 쪽으로 바이패스 되고 나머지 2개는 입력 버퍼쪽에 저장된다. 그리고 레지스터 클럭의 다음 클럭에 입력

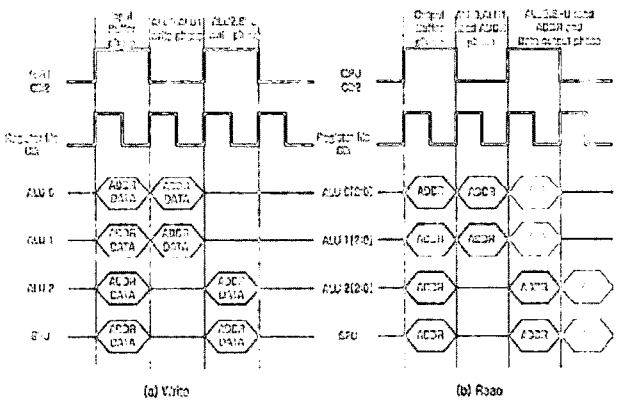

그림 8. 레지스터 파일에 대한 읽기 및 쓰기
Fig. 8. Timing diagram of reading and writing the register file.

버퍼에 있던 2개의 데이터가 다시 해당 레지스터 파일로 전달되어 연산 유닛으로부터 들어온 4개의 데이터에 대한 저장이 완료된다. 읽기 동작 시에는 그림 8의 b와 같이 연산 유닛으로부터 최대 12개의 레지스터 어드레스를 출력버퍼가 받게 된다. 출력버퍼는 12개의 어드레스 신호 중에서 6개의 신호는 바이패스하고 나머지 6개의 신호는 다음 클럭에 내보내게 된다. 이때, 레지스터 파일 내부의 클럭이 프로세서의 클럭보다 더 빠르게 동작한다. 이러한 읽기와 쓰기 과정이 동시에 수행되므로, 2W6R 포트를 가지는 레지스터 파일에서 4개의 쓰기와 12개의 읽기가 동시에 가능하도록 하였다.

4.3 레지스터 포트 인터페이스 로직의 설계

최대 4개까지의 연산 유닛들이 동시에 동작을 할 수 있고, 각각의 연산 유닛들을 위한 레지스터 파일의 읽기 및 쓰기 경로는 해당 명령어를 위한 레지스터 파일 쪽으로의 적절한 포트 선택을 필요로 한다. 본 논문의 레지스터 파일 선택은, 레지스터 파일을 연산 유닛별로 단일 싸이클에 3개의 읽기와 1개의 쓰기를 모델링 했으므로, 그러한 레지스터 파일에 맞는 입력 및 출력 버퍼를 설계하도록 한다. 따라서 4개의 연산 유닛들로부터의 레지스터 접근 포트를 각각 해당하는 레지스터 쪽으로 선택하여 주는 모듈을 작성한다. 설계된 레지스터 파일에 대한 전체 블록도는 그림 9와 같다.

그림 9에서와 같이 포트 선택기가 많은 수의 입출력 포트들을 다루고 내부적으로 많은 수의 멀티플렉서 등의 조합 논리회로가 존재하여야 하기 때문에 비교적 많은 로직 딜레이가 존재하게 된다. 이러한 부분이 레지스터 파일 액세스 시간과 같이 고려되어져서, 실제로 본 구조에서 시간 분할 매커니즘으로 동기화 시켜야 되는 이유가 된다.

4.4 레지스터 파일의 모듈별 구성

레지스터 파일의 전체 모듈 구성은 그림 10과 같으며 각 모듈의 동작은 표 4와 같다. 여기서 "Frequency Divider" 와 "Input/Output Buffer" 모듈은 시간 분할 매커니즘을 수행하기 위해 필요한 모듈들이다. "Frequency Divider" 모듈은 호스트로부터의 입력 클럭을 1/2로 만들어 출력하고, 그 출력값을 세이더 프로세서의 입력 클럭으로 사용하여 레지스터 파일이 시간 분할으로 동작할 수 있도록 클럭을 제공한다. 또한 입출력 버퍼에 대한 동작의 동기화를 위한 상태값을 함께 출력한다.

"Input/Output Buffer" 모듈은 클럭에 동기화 되면서 레지스터 파일이 다중 입출력을 수행 할 수 있도록 임시저장소의 역할을 한다. 입력 버퍼의 경우에는 동시에 2개씩 쓰는 레지스터 파일을 이용하여 4개의 데이터를 쓸 수 있게 하기 위해서 중간에서 버퍼의 역할을 하고, 출력 버퍼의 경우에는 동시에 6개씩 읽는 레지스터 파일을 이용하여 12개를 읽을 수 있게 하기 위한 중간 버퍼의 역할을 한다. 그리고 임시 레지스터 파일에 대한 다중 포트 레지스터 파일을 구현하기 위해서 "Temporary Register File" 모듈을 사용한다. 또한 요구하는 레지스터 타입을 보고 해당하는 레지스터 파일에 주소와 인에이블 신호들을 전달해 줌으로써 여러 레지스터 파일을 묶어서 한꺼번에 관리하는 "Register File Group" 모듈을 사용한다.

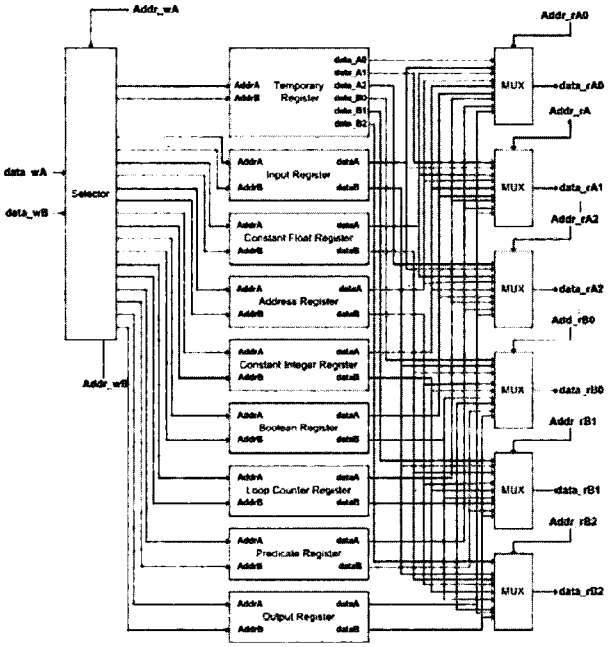

그림 9. 설계된 레지스터 파일에 대한 전체 블록도
Fig. 9. Block diagram of the register file.

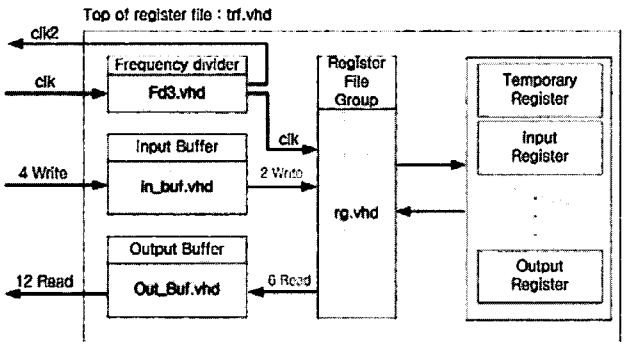

그림 10. 레지스터 파일 모듈
Fig. 10. Module of register file.

결과적으로, 연산 유닛의 입장에서 보면, 레지스터 파일에 대한 최상위 모듈인 "Top of Register File"의 모듈과 입출력 포트가 연결되면 단일 싸이클에 다중 입출력이 수행되는 것이다.

V. 로직 합성 및 설계 검증

본 논문에서 구현된 모든 구조는 HDL 언어를 사용하여 기술하였으며, 자일링스 ISE 9.2 프로그램을 사용하여 합성한 후, Model SIM 6.0 시뮬레이터를 이용한 기능적 레벨 시뮬레이션을 수행하였다.

5.1 기능적 레벨 시뮬레이션

레지스터 파일을 검증하기 위해서는 기본적으로 모든 레지스터 신호에 대한 읽기 및 쓰기 검증을 수행하여야 한다. 또한 다중 포트 구조를 지니고 있기 때문에, 동시에 여러 개의 포트를 통하여 레지스터 파일에 액세스 하였을 때 정상적인 읽기, 쓰기 동작을 보이는지를 같이 검증하도록 한다. 본 논문에서 보이는 레지스터 파일의 구조는 모두 4개의 연산 유닛과 포트를 공유하

표 4. 레지스터 파일 모듈 구성
Table 4. Hierarchy of the register file module.

모듈 이름	설 명
Top of Register File	이 모듈은 하나의 쓰기 값과 세 개의 읽기 값을 지원하는 레지스터 파일들의 탑 모듈이다.
Frequency Divider	이 모듈은 입력 클럭의 2배로 만들어 프로세서를 위한클럭을 출력하고 입출력 버퍼들의 동작의 동기화를 위한 상태값을 함께 출력한다.
Input Buffer	이 모듈은 레지스터 파일을 연산 유닛의 동작 클럭보다 2배 빠른 클럭으로 쓰면서 동시에 2개씩 쓰는 레지스터 파일을 이용하여 4개를 쓸 수 있게 하기 위해서 중간에서 버퍼의 역할을 한다.
Output Buffer	이 모듈은 레지스터 파일을 연산 유닛의 동작 클럭보다 2배 빠른 클럭으로 쓰면서 동시에 6개씩 읽는 레지스터 파일을 이용하여 12개를 읽을 수 있게 하기 위해서 중간에서 버퍼의 역할을 한다.
Register File Group	이 모듈은 여러 레지스터 파일을 묶어서 한꺼번에 관리하는 모듈이다. 요구하는 레지스터 타입을 보고 해당하는 레지스터 파일에 주소와 인에이블 신호들을 전달해 준다.
Register File	이 모듈은 Temporary Register를 제외한 레지스터들의 저장을 위한 레지스터 파일로 사용된다.
Temporary Register File	이 모듈은 Temporary Register의 저장을 위한 레지스터 파일로 사용된다.

면서 액세스 할 수 있는 구조로 되어있다. 이러한 동작이 가능하려면, 내부의 신호 선택기와 시간 분할 매커니즘이 중요하기 때문에 기본적인 레지스터 파일에 대한 동작의 검증을 수행하고, 기능 유닛들과의 검증도 같이하였다. 기본적으로 레지스터 파일에 접근을 하려면, 레지스터 타입과 어드레스번호, 데이터 및 이 신호 등을 조합하여 모든 신호에 대한 검증이 이루어지도록 한다. 그리고 벡터연산을 수행하기 위하여 번들 개수만큼의 데이터를 읽어오기 위한 액세스 신호가 같이 검증되도록 테스트하였다. 또한 레지스터 인터페이스 모듈의 내부 로직의 검증을 위해서 4개의 연산 유닛으로부터 각각의 포트들로의 경로가 제대로 연결이 되어 지는지를 검증하였다. 기본적으로 레지스터 파일 인터페이스 모듈에 대한 검증을 위해서 수행한 항목은 다음과 같다.

레지스터 파일의 모든 신호에 대한 기본적인 읽기, 쓰기
다중 포트로 여러 데이터의 동시 읽기, 쓰기 상황
4개의 기능 유닛과 레지스터 파일 간의 버퍼의 동작
프로그래머블 통합 셰이더 내에서 레지스터 파일의 실제 읽기, 쓰기 테스트

레지스터 파일을 연결하여, 실제 기능 유닛으로부터의 읽기 및 쓰기 검증은, 본 논문을 작성하기에 앞서 미리 설계하였던 프로그래머블 통합 셰이더 프로세서를 가지고 시뮬레이션하여 보았으며 그 결과 파이프라인된 4개의 연산 유닛에서 레지스터 파일을 동시에 참조해야 하는 프로세서에서 4개의 쓰기와 12개의 읽기가 가능함을 확인 할 수 있었다.

5.2 HDL 합성

자일링스 ISE 9.2를 사용하여 합성한 결과는 표 5와 같다. 본 논문에서는 표 5와 같이 합성한 결과를 토대로 실제 구현을 위해 Implementation 과정을 거쳐 생성된 비트파일을 logic tile for xc4vlx200에 다운로드하였다. 그리고 호스트 역할을 하는 versatile platform baseboard에 logic tile for xc4vlx200을 연결하여 실제 하드웨어에 타겟팅을 하여, 기능적 레벨 시뮬레이션 결과와 동일한 출력 결과를 확인하였다.

위 결과를 분석해보면, 로직의 동작속도가 레지스터 파일의 최상위 모듈인 경우에 206.695Mhz의 속도로 수행되는 것을 확인 할 수 있었다. 그리고 위의 슬라이스

표 5. 합성 결과
Table 5. Synthesize result.

세부모듈	Slices	동작속도(Mhz)
Top of Register File	6465	206.695
Frequency Divider	2	653.637
Input Buffer	555	685.895
Output Buffer	1927	685.895
Register File Group	5461	288.663
Register File	4274	301.568
Temporary Register File	1187	288.663

개수를 통하여 제안된 구조에 대하여 약 6465개의 슬라이스가 소모됨을 알 수 있어 실제 하드웨어 리소스를 알 수 있었다.

VI. 결론 및 향후 연구 과제

본 논문에서는 한정된 자원에서 고성능 3차원 컴퓨터 그래픽 영상을 지원하기 위한 방대한 양의 데이터와 복잡한 연산을 효율적으로 처리할 수 있는 멀티포트 레지스터 파일을 설계하였다. 본 논문에서 설계된 레지스터 파일은 한정된 자원을 가지고 있는 프로세서에서 많은 양의 데이터를 소수의 명령어를 사용하여 빠르게 처리하기 위해 단일 싸이클에 3개의 읽기신호와 1개의 쓰기신호가 동시에 가능하도록 자일링스 FPGA에서 기본적으로 제공하는 듀얼 포트 블록램을 사용해 하나의 블록으로 구성한 개선된 블록램 구조를 사용한다. 즉, 쓰기 동작 시에는, 연산 유닛으로부터 최대 4개의 데이터를 받게 되면 2개는 해당 레지스터 파일 쪽으로 바이패스 되고 나머지 2개는 입력 버퍼쪽에 저장된다. 그리고 다음 클럭에 두 개가 다시 해당 레지스터 파일로 전달되면 연산 유닛으로부터 들어온 4개의 데이터에 대한 저장이 완료된다. 읽기 동작 시에는, 이슈 유닛으로부터 최대 12개의 데이터를 받게 되면 데이터신호 6개가 해당 레지스터 파일로부터 출력 버퍼쪽으로 전달되고 그 다음 클럭에 6개가 전달된다. 이때, 레지스터 파일 내부의 클럭이 셰이더 프로세서의 클럭보다 더 빠르게 동작한다. 이러한 읽기와 쓰기 과정이 동시에 수행되므로, 셰이더 프로세서 클럭으로 단일 싸이클에 4개의 쓰기와 12개의 읽기가 가능하게 된다. 설계된 레지스터 파일은 자일링스 ISE 9.2를 사용하여 합성하였고, 합성한 결과

를 토대로 실제 구현을 위해 Implementation 과정을 거쳐 생성된 비트파일을 logic tile for xc4vlx200에 다운로드하였다. 그리고 호스트 역할을 하는 versatile platform baseboard에 logic tile for xc4vlx200을 연결하여 실제 하드웨어에 타겟팅을 하였다. 또한 설계된 로직의 동작속도가 레지스터 파일의 최상위 모듈인 경우에 206.695Mhz의 속도로 수행되는 것을 확인 할 수 있었다. 그리고 위의 슬라이스 개수를 통하여 제안된 구조에 대하여 약 6460개의 슬라이스가 소모됨을 알 수 있었다. 테스트는 Model SIM 6.0을 이용하여 기능적 레벨 시뮬레이션을 한 결과 모든 처리가 정확히 동작하는 것을 확인하였다. 마지막으로, 향후 연구과제로는 사용포트와 레지스터 파일의 크기가 증가하더라도 전체 시스템에 대한 자원 사용도와 동작속도의 감소가 최소화 되도록 레지스터 파일을 효율적으로 관리하는 알고리즘에 관한 연구가 필요하다.

참 고 문 헌

[1] T. Mitra, T. Chiueh, "Dynamic 3D Graphics Workload Characterization and the Architectural Implications", *In MICRO International Symposium on Microar-chitecture*, 1999.

[2] W. K. Jeong, *A SIMD-DSP/FPU for High-Performance Embedded Microprocessors*, Phd Thesis, Yonsei University, Dec, 2002.

[3] Cheol-Ho Jeong, "The Design of Geometry Processor for 3D Graphics," *Master Thesis*, Yonsei University, Dec, 1998.

[4] J. H. Sohn., R. Woo, H. J. Yoo, "A Programmable Vertex Shader with Fixed-point SIMD Data Path for Low Power Wireless" Applications. *Graphics Hardware*, 2004.

[5] D. Kim, K. Chung, C. Yu, C. Kim, I. Lee, J. Bae, Y. Kim, J. Park, N. Seong, "An SoC with 1.3Gtexels/s 3-D Graphics Full Pipeline for Consumer plications," *IEEE Journal of Solid State Circuits* Vol 40, No.1, Jan. 2006.

[6] B. Khailany, "The VLSI Implementation And Evaluation Of AREA-And Energy-efficient Streaming Media Processors", *PhD dissertation*, Stanford University, Jun. 2003.

[7] J. L. Hennessy and D. A. Patterson, Computer Architecture: *A Quantitative Approach, Third Edition*, Morgan Kaufmann, 2003.

[8] Foley, V. D., Feiner, Hughes, *Computer Graphics Principle and Practice*, Addison· & Wesley, 1996

[9] R. Balasubramonian, S. Dwarkadas, and D. H. Albonesi, "Reducing the complexity of the register file in dynamic superscalar processors", *Proceedings of 34th annual ACM/IEEE international symposium on Micro architecture*, pp. 237-248, 2001.

[10] J. L. Cruz, A. Gonzalez and M. Valero and N. P. Topham, "Multiple-banked register file architectures", *Proceedings of 27th annual international symposium on computer architecture*, pp. 316-325, 2000.

[11] S. Wallace and N. Bagherzadeh, "A Scalable Register File Architecture for Dynamically Scheduled Processors", *Proceedings of the Conference on Parallel Architectures and Compilation Techniques*, 20-23 Oct. pp. 179-184, 1996.

[12] B. Khailany, "The VLSI Implementation And Evaluation Of AREA-And Energy-efficient Streaming Media Processors," *PhD dissertation*, Stanford University, Jun. 2003.

[13] U. J. Kapasi, W. J. Dally, S. Rixner, J. D. Owens, and B. Khailany. The Imagine Stream Processor. In Proceedings of the 2002 IEEE International Conference on Computer Design: *VLSI in Computers and Processors* (ICCD'02).

[14] T. Imagire, DirectX9 Shader Programming Book, Tuttle-Mori, 2004.

[15] Microsoft, MSDN Asm Shader Reference Vertex/Pixel Shader 3.0

[16] M. Johnson, Superscalar Microprocessor Design, Prentice Hall, pp. 9-22, 1991.

저 자 소 개

윤 완 오(학생회원)
2000년 경기대학교 전자공학과 학사 졸업.
2002년 인하대학교 전자공학과 석사 졸업.
2002년~현재 인하대학교 전자공학과 박사 과정.
<주관심분야 : 분산 처리 시스템, 병렬프로그래밍, 컴퓨터 아키텍처>

김 경 섭(학생회원)
2002년 한남대학교 전자공학과 학사 졸업.
2000년~현재 인하대학교 전자공학과 석사 과정.
<주관심분야 : 컴퓨터 아키텍처, SoC & 임베디드 시스템 디자인>

정 진 하(학생회원)
1992년 인하대학교 전자공학과 학사 졸업.
1994년 인하대학교 전자공학과 석사 졸업.
1994년~1999년 한미 기술연구소(주)
2000년~현재 인하대학교 전자공학과 박사과정.
<주관심분야 : 컴퓨터구조, 병렬 및 분산 처리 시스템, 시스템 디자인, Fault-tolerant computing>

최 상 방(평생회원)
1981년 한양대학교 전자공학과 학사 졸업.
1981년~1986년 LG 정보통신(주)
1988년 University of Washington 석사 졸업.
1988년 University of Washington 박사 졸업.
1991년~현재 인하대학교 전자공학과 교수
<주관심분야 : 컴퓨터 구조, 컴퓨터 네트워크, 무선 통신, 병렬 및 분산 처리 시스템, antFault-tolerant computing>