논문 2007-44SD-9-9

SoC 플랫폼 기반 모바일용 3차원 그래픽 Hardwired T&L Accelerator 구현

(Implementation of a 3D Graphics Hardwired T&L Accelerator based on a SoC Platform for a Mobile System)

이 광 엽*, 구 용 서**

(Kwang-Yeob Lee and Yong-Seo Koo)

요 약

본 논문에서는 휴대 정보기기 시스템에서 더욱 향상된 실시간 3D 그래픽 가속 능력을 갖는 SoC(System on a Chip) 구현을 위해 효과적인 T&L (Transform & Lighting) Processor 구조를 연구하였다. T&L 과정에 필요한 IP들을 설계하였으며, 이를 바탕으로 SoC Platform 기반으로 검증하였다. 설계된 T&L Processor는 24 bits 부동소수점 형식과 16 bits 고정소수점 형식을 적절하게 혼용하고 계산식의 병렬성을 최대한 활용하여 Transform 과정 연산과 Lighting 과정 연산의 지연시간을 균일하게 배분하여 Transform 과정만 처리할 때와 Lighting과 혼용으로 처리할 때 연산 속도의 차이가 없이 동작이 가능하다. 설계된 T&L Processor는 SoC 플랫폼을 이용하여 성능 측정 실험 및 검증을 하였고, Xilinx-Virtex4 FPGA에서 80 MHz의 동작 주파수를 확인하였고 초당 20M개의 정점(Vertex) 처리 성능을 확인하였다.

Abstract

In this paper, we proposed an effective T&L(Transform & Lighting) Processor architecture for a real time 3D graphics acceleration SoC(System on a Chip) in a mobile system. We designed Floating point arithmetic IPs for a T&L processor. And we verified IPs using a SoC Platform. Designed T&L Processor consists of 24 bit floating point data format and 16 bit fixed point data format, and supports the pipeline keeping the balance between Transform process and Lighting process using a parallel computation of 3D graphics. The delay of pipeline processing only Transform operation is almost same as the delay processing both Transform operation and Lighting operation. Designed T&L Processor is implemented and verified using a SoC Platform. The T&L Processor operates at 80MHz frequency in Xilinx-Virtex4 FPGA. The processing speed is measured at the rate of 20M Vertexes/sec.

Keywords: 3D Graphics accelerator, T&L Processor, SoC Platform

I. 서 론

2D 그래픽의 세계에서는 면의 세계에 국한된 표현만이 가능했지만 3D 그래픽의 세계는 면으로 이루어진 세계를 표현하기 때문에 그 표현범위가 대폭 넓어지게 되었다. 그 세계는 현실감이 존재하는 세계이다. 3D 그래픽 응용 분야로는 가장 흔하게 적용되는 3D 페인팅 분야의 렌더링과 CAD분야를 비롯하여 3D 애니메이션, 가상현실, 만화영화, 게임, 영화, 입체영상 등 다양한 분야에서 쓰여 지고 있다. 실시간 3D 그래픽 기술의 등장으로 그 활용 분야가 대폭 넓어지게 되었다.[1~3]

특히 게임 분야에서 사용자들의 현실감에 대한 꾸준한 요구로 게임기의 성능이 일반적인 PC의 성능을 넘어서고 있다. 이러한 3D 그래픽의 발전에 힘입어 3D 그래픽 가속기의 필요성이 점점 더 요구되고 있는 추세

* 평생회원, 서경대학교 컴퓨터공학과
(Dept. of Computer Engineering Seokyeong Univ.)
** 평생회원, 서경대학교 전자공학과-교신저자
(Dept. of Electronic Engineering Seokyeong Univ.)
※ 본 논문은 서울특별시 나노혁신 클러스터 사업의 지원과 한국전자통신연구원 SoC 산업진흥센터의 지원으로 제작되었습니다.
접수일자: 2007년6월7일, 수정완료일: 2007년8월16일

이다. 3D 그래픽을 표현하기 위해서는 많은 데이터를 연산해야 하며 연산 과정 또한 복잡하다. 초기에는 CPU가 이 모든 연산 및 처리를 담당하여 3D 영상을 표현하였으나 보다 정교하고 현실감 있는 3D 영상을 표현하기 위한 데이터의 양이 기하급수적으로 증가함에 따라 CPU만으로 이를 처리하기에는 큰 부담이 되었다.[3] 따라서 그래픽 처리를 위한 가속 하드웨어를 별도로 설계 하거나, 고속의 범용 Processor를 이에 맞게 확장한 방법이 사용 되고 있다.

화면상의 한 장면이 실제와 가깝게 보이도록 모델링을 하기 위해서는 한 장면에 약 백만 개의 그래픽 프리미티브들로 이루어져야 한다. 이 장면을 실시간으로 처리하기 위해서는 특수하게 제작된 그래픽 가속기가 필요로 하게 되는데 이런 장치들은 상당히 고가이므로 요구되는 그래픽의 속도와 화면의 질에 따라 가속기의 기능을 가감한 형태의 가속기들이 사용되기도 한다.

일반적으로 3D 그래픽의 처리는 크게 T&L 처리 과정과 Rasterization 과정으로 나눌 수 있다. T&L 처리의 이전 단계는 일반적으로 호스트에 의해 처리되며 그 이후의 단계를 그래픽 가속기가 담당한다. T&L 처리 과정은 모델링한 데이터 형식이 실수이기 때문에 부동소수점 연산을 하며, Rasterization 과정은 화면의 해상도와 관련되어 연산을 수행하므로 주로 정수 연산을 한다. 일반적인 산술 연산은 부동소수점 연산과 정수연산으로 구분한다. 부동소수점 연산이 정수 연산보다 두 배 이상의 계산량을 필요로 하기 때문에 부동소수점 연산을 빠르게 처리할 수 있는 연산기의 설계가 필수적이다. 또한 일반적으로 그래픽 데이터들은 병렬성이 굉장히 높기 때문에 다수의 연산기들을 이용해서 그래픽 가속기를 설계한다.

본 논문은 휴대용 정보기기 시스템에서 3D 가속을 제공하기 위한 효과적인 T&L Processor 구조를 설계하고, 이를 SoC 플랫폼 기반에서 검증함으로써 복잡도가 높은 SoC의 설계 및 검증 방법을 개선하였다. 검증 시스템은 ARM과 AMBA가 내장된 SoC Base 1.0 기반의 SoC Platform을 사용하였다.

II. 3D 그래픽 처리 과정

3차원 그래픽 처리는 크게 T&L(Transform & Lighting) 처리와 Rasterization 처리로 나눌 수 있다. T&L 처리는 정점 변환, 색 계산, 좌표계 변환으로 구성되며, Rasterization은 좌표계의 이미지에 실제적인 색깔 값을 결정하고, 화면에 나타낼 영상 데이터를 처리하여 프레임 버퍼(Frame Buffer)에 데이터를 전송한다.

1. 3D 그래픽 렌더링의 기본 이론

3D 그래픽 처리는 기본적으로 그림 1과 같이 Application 단계, T&L 처리 단계와 Rasterization 단계, 프레임 버퍼를 거쳐 LCD로 출력해주는 단계를 순차적으로 수행하는 4 가지 단계로 이루어진다.[4]

Application 단계는 모델 제네레이션(Generation)으로 생성된 오브젝트의 데이터베이스를 3D 그래픽 렌더링 파이프라인에서 처리할 수 있는 데이터로 전달하는 과정이며, 일반적으로 호스트 CPU로 처리한다.

T&L 처리 단계는 앞 단계에서 생성된 데이터를 기하학적으로 변환시키는 과정으로 사용자의 시점에 따라 모델 데이터의 좌표계 변환이 이루어지고, 광원의 모델과 빛의 세기, 모델링 물체의 색과 속성에 의해 그래픽 프리미티브의 한 정점에 대한 색을 계산한다. View Volume에 따른 실제화면에 나타나는 프리미티브가 결정되며, 이 들을 화면에 나타내기 위해 2D 좌표계로 변환된다.

Rasterization 단계는 T&L 처리에 의해 변환된 프리미티브(점, 선, 삼각형 등)들을 프레임 버퍼에 픽셀 데이터 값으로 변환시키는 과정이며 프리미티브들의 모서리 기울기를 좌표 값과 색의 정보에 대해 계산한다. 프리미티브의 한 정점에 대한 색 값과 모서리의 기울기 정보를 이용한 보간법(Interpolation) 연산으로 프리미티브 내부의 R, G, B, Z 값을 계산하여 서브 픽셀의 값을 생성한 후 최종적으로 프레임 버퍼에 갱신한다.

3D 그래픽 렌더링 단계를 거치는 과정에서 주요한 병목지점은 T&L 처리의 부동소수점 연산 부분과 Rasterization 과정에서 픽셀의 값을 생성해 내는 부분, 프레임 버퍼의 갱신에 충분한 대역폭(Bandwidth) 부분이다. 특히 T&L 처리에서는 좌표 변화, 투영, 화면 매

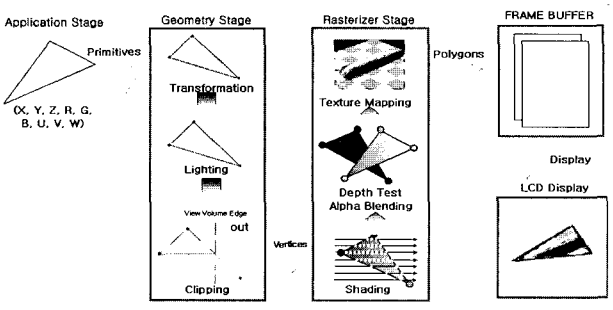

그림 1. 3D 그래픽 렌더링 단계
Fig. 1. 3D graphic rendering stage.

핑과 정점의 색 값을 계산하는 단계에서 아주 많은 부동 소수점 연산을 요구한다.

2. 3D 그래픽 T&L 처리 과정

T&L 단계에서 사용되는 입력 데이터는 부동소수점 형식이며 T&L 처리를 수행하기 위해서는 기본적인 사칙 연산 외에 더 복잡한 연산들을 요구한다. T&L 처리는 호스트에 의해 처리된 모델 데이터의 정보들의 입력 받음으로서 시작한다. 모델 데이터는 한 좌표의 정보, 각 좌표의 수직 방향을 나타내는 노말 벡터(Normal Vector), 물질과 광원의 Ambient, Diffuse, Specular 색 속성 정보와 각 단계에서 요구되는 파라미터들이 포함한다. T&L 처리는 그림 2와 같이 모델 데이터에 대한 공간상의 위치 변환을 처리하는 Transform 과정과 모델 데이터를 구성하는 각 정점의 빛에 대한 색 계산을 하는 Lighting 과정으로 구분한다.[4~5]

Transform 처리 과정에서는 동차 좌표(Homogeneous Coordinates)를 사용하여 모델의 정점 좌표를 x, y, z, w로 나타낸다. 이는 모델의 정점 좌표 변환 시 크기 변환, 회전 변환과 이동 변환을 동시에 한번의 4 * 4 변환 행렬 곱셈으로 처리할 수 있으며, 이러한 특성은 3D 그래픽에서 매우 유용하다. 동차 좌표는 모델 정점 좌표에 w를 1로 설정하고, 4 * 4 행렬 변환 행렬의 마지막 대각 성분에 1을 넣어 이용한다.[5]

Transform 처리 과정은 모델의 좌표를 시점과 이동 정보로 변환하는 Model/View Transform 과정, 3차원 View Volume으로 투영하는 Projection 과정, 모델의 정점 좌표 x, y, z를 w로 나누는 Divide by W 과정과 2D 화면 좌표로 모델 좌표를 변환하는 Viewport Transform 과정으로 구성된다.

Lighting 처리 과정은 3D 그래픽 모델의 각 정점에 빛 요소를 적용하여 해당 정점의 색을 구하는 과정으로

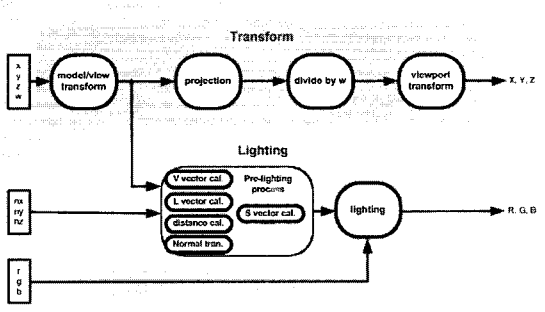

그림 2. 3D 그래픽 T&L 처리 과정
Fig. 2. 3D graphic T&L process.

모델의 정점에 대한 수직 방향을 나타내는 노말 벡터(Normal Vector), 광원에 대한 반사율을 계산하기 위한 방향 벡터인 라이팅 벡터(Lighting Vector), 뷰 벡터(View Vector)와 Ambient, Diffuse, Specular 효과를 내기 위한 상수 값들을 이용하여 모델의 정점의 색 R,G,B를 계산하는 과정으로 구성된다.[6~7]

Ⅲ. T&L Processor

3D 그래픽 Processor는 3D 그래픽의 고속처리를 위해, 부동소수점 연산이 집중 되어 있는 T&L 연산의 가속화를 목적으로 한다. 눈에 띄게 발전하는 화려함과 웅장한 3D 그래픽을 처리하기 위해서는 그만큼의 처리해야하는 연산양이 많아진다. 상기 목적을 달성하기 위해서는 T&L 연산과정을 잘 파악하여 구조를 보다 효율적으로 설계하여야 한다. 이 장에서는 Processor를 효율적으로 설계하기 위해 T&L Processor의 구조를 연구하고 이를 기반으로 T&L Processor를 설계하였다.

1. T&L Processor 구조 연구

T&L 과정에 포함되는 Transform 과정과 Lighting 과정이 모두 연산되어야 1개의 정점을 처리하기 때문에 이 2가지 과정을 병렬로 처리하는 것이 효율적이며, 병렬로 처리되는 두과정이 가능한 동시에 마무리 되어서 한 쪽이 스톨(Stall)되는 것을 지양해야 한다.

제안하는 Processor의 구조는 Hardwired Decoder Architecture이다. Hardwired 구조의 특징은 명령어 없이 컨트롤되는 구조이다. 명령어 중심의 Processor로 설계했을 경우보다 유연성이 떨어지지만 명령어 구조에서 빈번하게 일어나는 Load/Store 과정이 없고 각 명령어마다 존재하는 ID단과 WB단이 존재하지 않고, 연산 중에 생기는 저장 레지스터가 줄어든다. 상대적으로 단순한 로직만을 실행할 수밖에 없지만 속도가 빠르고 메모리 용량을 줄일 수 있다는 장점이 있고 T&L 처리를 병렬로 수행할 수 있는 장점을 갖고 있다. T&L 처리 과정에서 Transform 부분이나 Lighting 부분이 똑같은 패턴의 연산을 계속 사용한다는 것을 이용하여 Hardwired 구조를 채택하였다.

Transform 과정에서는 3D 그래픽 모델의 좌표를 변환하는 과정으로 4×4 행렬과 4×1 행렬의 곱셈 연산이 주를 이룬다. 이 연산 과정은 곱셈과 덧셈 연산으로 이루어지며, 이 외에 나눗셈 연산이 필요하다. 곱셈과 덧셈 연산은 Transform 과정의 모든 과정에서 사용하며,

나눗셈 연산은 원근 투영 변환 후 모델의 좌표 동차화 과정에서 사용한다. Transform 과정에서 Divide by W 계산에 따른 지연시간을 최소화하기 위해 x, y, z, w 값들 중에서 w 값을 가장 먼저 계산하는 방식으로 설계하여 파이프라인의 스톨없이 동작하도록 하였다. Transform 과정에서 사용한 데이터 형식은 부동소수점 형식으로 IEEE 754 부동소수점 단정도 표준의 32 bit를 수정하여 24 bit의 길이로 만들었다.[8~11] 이는 모바일 기반의 3D 그래픽 가속기를 설계에 있어서 가능한 적은 면적과 작은 전력으로 구현하기 위해 채택하였다.

Lighting 처리 과정은 3D 그래픽 모델의 각 정점에 빛 요소를 적용하여 해당 정점의 색을 구하는 과정이다. Lighting 처리 전 과정에서 주로 곱셈과 덧셈 연산으로 이루어지며, 광원과 모델의 정점간의 거리 계산을 위해 나눗셈 연산을 사용하며, Lighting 효과 중 하나인 Specular 계산을 위해 멱승 연산을 사용한다.[12~13]

Lighting 처리 과정은 Transform 처리 과정보다 더 많은 연산량을 요구한다. Lighting 처리 과정을 Transform 처리 과정과 비슷한 속도로 연산을 맞춰주기는 것이 필요하다. 이를 위해 Lighting 처리 과정에서는 데이터 형식에 변화를 주어 16 bit 고정소수점 형식을 사용하고, 최대한 연산기를 병렬로 배치하여 설계하였다. 고정소수점 형식에 따른 연산기는 지수에 대한 변화를 고려하지 않아도 되기 때문에 같은 덧셈이나 곱셈 연산을 하더라도 2~3배 빠르게 처리할 수 있다.

2. 제안하는 T&L Processor의 구조

가. T&L Processor를 위한 연산기

일반적인 3D 그래픽 T&L 처리 과정은 많은 부동소수점 연산을 요구한다. IEEE 754 단정도 부동소수점을 기반으로 하며, 처리 과정에 필요한 연산의 종류는 사칙 연산 외에 더 복잡한 연산을 요구한다. 모바일 Platform 기반의 3D 그래픽 가속기를 설계하기 위해서는 범용 컴퓨터 이상의 규모에서 사용하던 연산기 형식을 사용하는 것은 전력소모, 시스템의 소형화 그리고 가격 경쟁에서 큰 어려움이 있다. 가능한 적은 면적에 소전력으로 구현할 수 있는 연산기를 요구한다. 그래서 T&L 처리 과정에 적합한 데이터 형식을 정리하고 각 과정에 필요한 연산기를 설계하였다.

나. T&L Processor에 적용된 데이터 형식

T&L Processor에서 사용한 데이터 형식은 부동소수점 형식과 고정소수점 형식으로 나뉜다. 부동소수점 형식을 기반으로 하되 Lighting 처리 과정의 성능을 높이기 위하여 Lighting Unit에서는 주로 고정소수점 형식을 사용하였다. 부동소수점 형식은 IEEE 754 부동소수점 표준이 가지는 형식을 수정하여 24 bit의 길이로 만들었다. 그림 3은 제안하는 부동소수점 형식이다.[14]

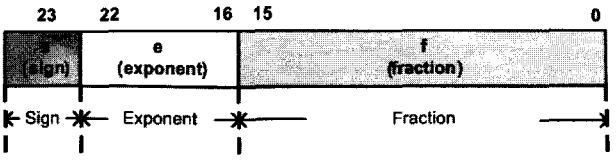

그림 3. 제안하는 부동소수점 형식

Fig. 3. Proposed 24-bit Floating Point Data Format.

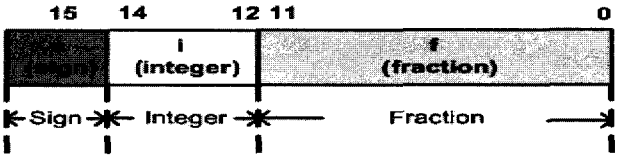

그림 4. 제안하는 고정소수점 형식

Fig. 4. Proposed 16-bit fixed Point Data Format.

그림 4는 제안하는 고정소수점 형식이다. 제안하는 고정소수점 형식는 총 16 bit로 이루어져 있다.

다. 제안하는 부동소수점 연산기

부동소수점 형식을 기반으로 하는 연산기는 덧셈기, 곱셈기, Transform 과정에서 사용되는 역수기, Lighting 과정에서 고정소수점 형식과 혼용으로 사용하는 멱승기가 있다.

그림 5의 제안하는 부동소수점 덧셈/뺄셈기는 특수값

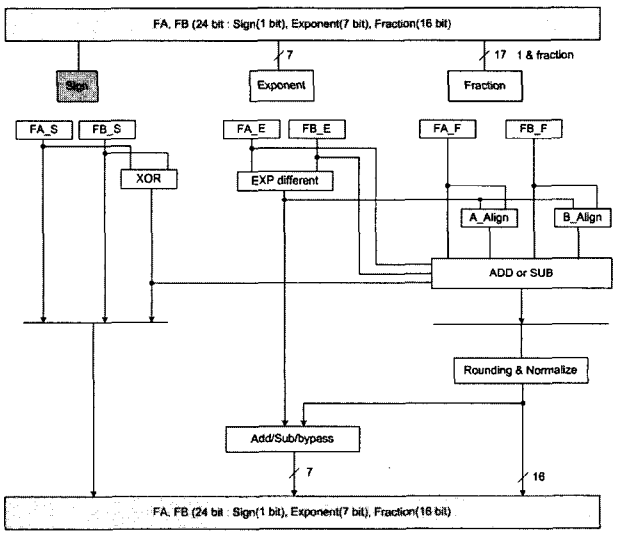

그림 5. 제안하는 부동소수점 덧셈/뺄셈기

Fig. 5. Proposed 24-bit Floating Point Adder.

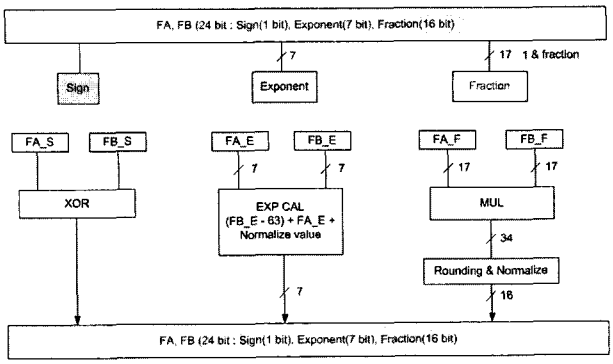

그림 6. 제안하는 부동소수점 곱셈기

Fig. 6. Proposed 24-bit Floating Point Multiplier.

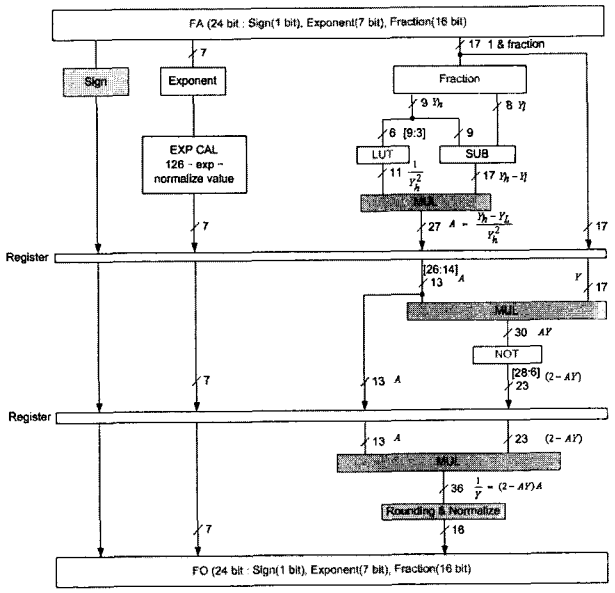

그림 7. 제안하는 부동소수점 역수기

Fig. 7. Proposed 24-bit Floating Point Reciprocal.

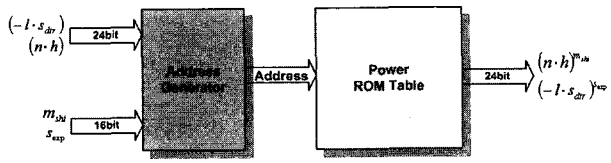

그림 8. 제안하는 부동소수점 멱승기

Fig. 8. Proposed 24-bit Floating Point Power.

표현을 줄이고 Rounding 유닛을 간결화해 단 사이클로 구성하였다.

그림 6의 제안하는 곱셈기는 덧셈/뺄셈기와 마찬가지로 특수값 표현을 줄이고 Rounding 유닛을 간결화해 단 사이클로 구성하였다.

그림 7의 제안하는 나눗셈 연산기는 3D 그래픽 처리 특성상 나머지 값이 필요하지 않고, 적은 수행 단계의 파이프라인 수행이 가능하며, 역수 계산에 유리한 NR(Newton-Raphson) 방식으로 나눗셈(역수) 연산기를 설계하였다.

멱승 연산은 음영 처리과정 중 정반사 성분 와 집중 조명광을 연산하기 위해 사용된다. 본 논문에서는 그림 8과 같이 속도 면에서 가장 좋은 롬 테이블 방식을 사용했다.

라. Transform Unit

Transform 과정은 모델의 좌표를 시점과 이동 정보로 변환하는 Model/View Transform 과정, 3차원 View Volume으로 투영하는 Projection 과정, 모델의 정점 좌표 x, y, z를 w로 나누는 Divide by W 과정과 2D 화면 좌표로 모델 좌표를 변환하는 Viewport Transform 과정으로 구성된다.

Transform 처리 과정은 동차 좌표(Homogeneous Coordinates)를 사용하여 모델의 정점 좌표를 x, y, z, w로 나타낸다. 이는 모델의 정점 좌표 변환 시 크기 변환, 회전 변환과 이동 변환을 동시에 한번의 4 * 4 변환 행렬(Matrix) 곱셈으로 처리할 수 있으며, 이러한 특성은 3차원 그래픽에서 매우 유용하다. 동차 좌표는 모델 정점 좌표에 w를 1로 설정하고, 4 * 4 행렬 변환 행렬의 마지막 대각 성분에 1을 넣어 이용한다. 다음 그림 9는 Transform 내부의 블록 다이어그램이다.

Transform 과정에선 4*4 Matrix와 4*1 Vector 간의 곱셈 연산을 하여 각각의 x, y, z, w 값을 구하고 x, y, z에 1/w값을 곱하는 과정이 필요하다. 1/w 값을 반영

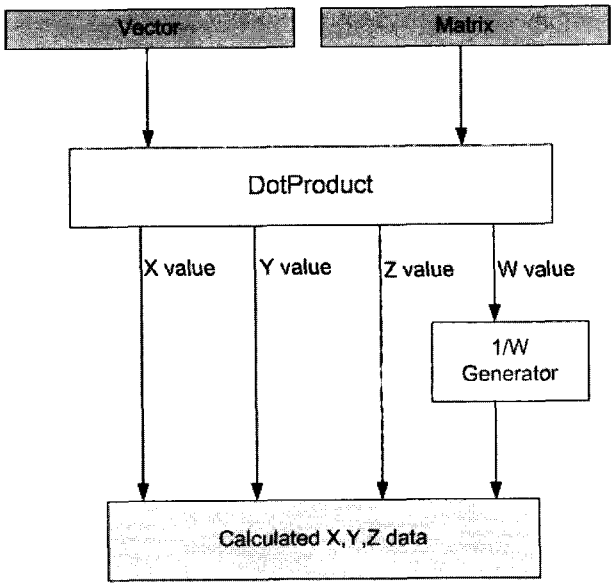

그림 9. Transform Unit의 블록 다이어그램

Fig. 9. Block diagram of Transform Unit.

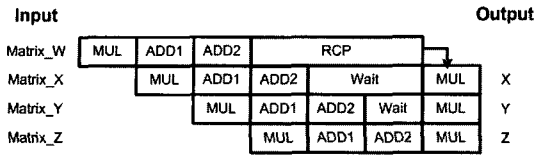

그림 10. Transform Unit 파이프라인
Fig. 10. Transform Unit Pipeline.

하기 전에 x, y, z, w 값을 계산할 때는 최대한 빠르게 계산하기 위해 연산기를 병렬로 배열하여 구성하였기 때문에 부동소수점 가/감산기는 4개, 부동소수점 승산기는 3개가 필요하다. x, y, z, w 값을 구하고 x, y, z에 1/w 값을 곱하는 과정이 있기 때문에 x, y, z 값이 계산되어지더라도 1/w 값이 계산될 때까지 기다리는 현상이 발생한다. 그래서 제안하는 Transform Unit에서는 w값을 제일 먼저 계산하고 x, y, z를 계산하여 전체 연산과정의 길이를 최대한 줄였다. 아래 그림 10은 제안하는 Transform Unit의 파이프라인 구조이다.

마. Lighting Unit

Lighting 과정은 노말 벡터와 물질과 광원의 Ambient, Diffuse, Specular 색 속성 정보와 각 단계에서 요구되는 파라미터들을 입력받아 이를 처리하는 과정이다. Lighting 과정의 조명 처리식은 경우에 따라 효과를 주는 방식이 다르기 때문에 유동적으로 많이 사용되나 본 논문의 T&L Processor에서는 OpenGL에서 실제 연산하고 있는 조명 처리식을 사용하였다. 다음 그림 11은 Lighting 과정 내부의 블록 다이어그램이다.

Lighting Unit에서 처리하는 과정을 크게 4가지로 분류한다. Ambient 효과를 계산하는 부분, Diffuce 효과를 계산하는 부분, Specular 효과를 계산하는 부분, 광원과 정점과의 거리를 계산해주는 부분으로 구분한다.

Ambient 효과는 모델 자체에서 가지는 최소한의 색상을 계산한다. 거리에 따른 감쇠 효과 계산과 벡터(Vector)에 따른 요소가 없기 때문에 광원과 모델의 Ambient 효과 값만으로 간단히 계산한다.

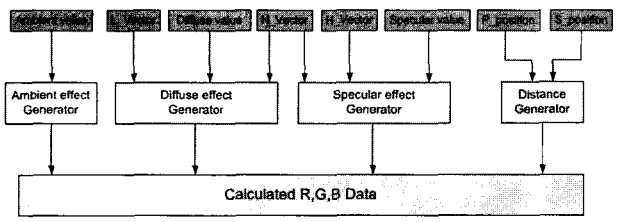

그림 11 Lighting Unit의 블록 다이어그램
Fig. 11. Block diagram of Lighting Unit.

Diffuse 효과는 특정 방향으로부터 비춰지지만 빛이 표면에 고르게 반사되는 색상을 계산한다. Diffuse 효과는 방향 성분을 포함하기 때문에 모델의 표면과 광원과의 각도, 광원과의 거리에 따른 감쇠 요소에 따라 달라진다. Diffuse 효과에 사용되는 벡터는 라이트 벡터와 노말 벡터가 쓰인다. 라이트 벡터와 노말 벡터간의 내적 연산을 통해 계산된 값을 모델과 광원의 Diffuse 효과 값과 연산하여 광원의 거리에 따른 감쇠효과를 계산하는 것으로 Diffuse 효과를 계산하게 된다.

Specular 효과는 하이라이트를 생성함으로서 표면이 반짝 거리도록 보이게 하는 효과로서 Diffuse 효과처럼 방향성을 갖고 있지만 특정 방향으로 정확히 반사되는 효과로서 Diffuse 효과보다 긴 연산 시간을 갖는다. 노말 벡터와 하프 벡터간의 내적 연산을 통한 값과 Specular 성분의 필요 입력 값인 Shininess 값을 이용하여 멱승 연산을 하게 된 후 모델과 광원의 Specular 효과 값을 연산 한 후 광원의 거리에 따른 감쇠효과를 계산하는 것으로 Specular 효과를 계산하게 된다.

Diffuse와 Specular 효과에서 감쇠 효과를 계산해주는 Distance Generator는 광원의 위치와 모델의 정점의 위치의 절대 길이를 계산한다. 이 절대 길이가 길면 길수록 감쇠 효과가 커지기 때문에 절대 길이를 계산한 후에 역수를 구하여 감쇠 효과 값을 계산한다.

Transform 과정은 정점 1개를 처리하는데 7 cycle이 소요되고 연속적으로 정점 계산을 하게 되면 산술적으로 4 cycle마다 정점 1개를 처리한다. 그렇지만 Transform 과정에 필요한 Matrix와 Vector의 값의 크기가 크기 때문에 실제적으로 데이터를 로드하는데 걸리는 시간까지 고려하더라도 5~6 Cycle이면 처리한다. 위에서 제안하였듯이 Transform 과정에 비해 Lighting 과정의 경우에는 처리해야할 연산이 엄청나게 많기 때문에 Transform 과정에 사용된 부동소수점 형식을 사용하게 될 경우 몇 배가 더 많은 Cycle의 수가 요구된다. 따라서 다른 방식으로 데이터 형식을 구성해야할 필요성이 있다. 적용한 데이터 형식은 부동소수점 형식을 사용하지 않고 대부분의 Lighting 연산에서는 고정소수점 형식을 사용하였다. 고정소수점 형식을 적용할 수 있었던 이유는 Lighting에 사용되는 변수 대부분이 0~1로 구성되어 있음을 이용하였고, 일반적으로 사용되는 RGB 포맷이 32bits와 16 bits가 많이 쓰이는데 32 bits의 경우라도 각각 RGB의 값의 최대 bit 수가 8 bits이기 때문에 16 bits의 정밀도에서 소수점 이하 비트들에 할당된 변수가 12 bits 이기 때문에 충분히 표현할

수 있는 범위라 선택하였다. 연산계산의 지연을 최대한 줄이기 위한 또 다른 방법으로는 연산의 병렬성을 최대한 활용해야 한다. Lighting 과정은 Ambient, Diffuse, Specular 계산을 한 후에 이 3가지 결과를 조합하여 RGB 값을 결정하는데 목적이 있다. 최대한 3가지 과정을 병렬로 처리해야 연산의 지연시간을 줄일 수 있다.

Lighting 과정에 계산되는 RGB 값은 광원이 1 개일 경우와 2개 이상일 때로 나눌 수 있다. 1 개의 광원을 사용하면 Ambient, Diffuse, Specular 속성을 더해 구해진 RGB 값을 그대로 반영하면 되지만 2개 이상일 경우는 각각 계산된 RGB 값을 더해주는 과정이 필요하게 된다. 따라서 Lighting 과정에 필요한 파이프라인 연산들을 정리하면 다음과 같다.

1. 각각 Diffuse, Ambient, Specular 속성의 RGB 값들을 계산하는 동시에 광원과 정점의 거리를 계산한다.

2. Diffuse와 Specular의 경우 광원과 정점과의 거리에 따른 감쇠효과가 있기 때문에 Diffuse의 RGB 값과 Specular의 RGB 값을 더한 RGB 값과 1번 과정에서 구해진 광원과 정점의 거리 값을 곱한다.

3. 2번 과정을 거쳐 나온 RGB 값과 Ambient의 RGB 값을 더한다. 여기서 계산된 RGB 값은 한 개의 광원에서 계산된 RGB 값이 된다.

4. 1개의 광원만 사용될 경우 3번에서 계산된 RGB값을 그대로 출력으로 보내면 되지만 2개 이상의 광원이 필요할 경우 임시 레지스터를 활용한다. 광원이 3개 들어온다고 가정할 경우 첫 번째 광원의 RGB 값은 그대로 임시 레지스터로 저장하고 다음 파이프라인에서 계산되어 나온 2번째 RGB 값이 나오면 임시 레지스터에 저장되어 있는 RGB값과 더하여 다시 임시 레지스터로 저장한다. 3번째 광원의 RGB 값이 들어오면 임시 레지스터에 저장되어 있는 RGB값을 계산하여 최종 RGB 값을 출력한다. 이 방식을 이용하게 되면 광원 1개 추가시마다 Cycle 수가 1개만 증가하게 되어 효율적인 파이프라인 구조가 된다.

위의 과정을 거쳐 계산되어지는 Lighting 처리 과정 중 Specular 과정의 파이프라인을 그림 12에 정리하였다. Specular 과정의 파이프라인 과정을 보인 이유는 빛 속성 중에 가장 긴 파이프라인을 갖고 있기 때문이다. DOT1과 DOT2 과정 그리고 POW로 처리된 1~3 Stage에서 Specular의 기본 값이 계산되고 그리고 ADD1이라고 표현된 과정에서 Specular 색 속성 정보에 따른 값이 계산된다. 이 과정과 병렬적으로 Diffuse 색 속성 정보에 따른 값과 계산되어진다. ADD2에서는 거리에 따른 감쇠효과의 RGB 계산과 같은 파이프라인 과정을 거친 Ambient RGB값이 더해지게 된다. 고로 한 개의 광원을 처리할 때 5 cycle이 소요됨을 알 수 있다. 그리고 ADD2에서는 임시 레지스터에 저장된 RGB 값과 더하는 과정이 있을 수도 있다. 고로 광원이 2개 이상 존재할 때는 기본 5 Cycle에 광원 1개당 추가로 1 Cycle씩 추가된다.

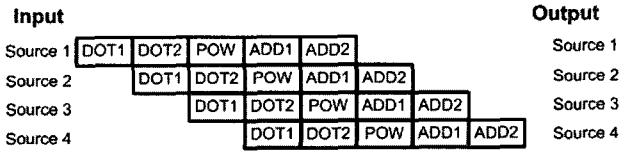

그림 12. Lighting Unit의 Specular 과정 파이프라인
Fig. 12. Specular stage pipeline of Lighting Unit.

바. Transform Unit과 Lighting Unit의 비교

Transform과 Lighting 과정의 파이프라인에 따라 정리하여 그림 13에 표현하였다. 그림 13에서 Transform 과정은 한 개의 Output을 얻기 위해서는 4 사이클에 걸쳐 Input을 입력 받아야 한다. 그리고 Output은 최대 4 사이클에 한 개씩 얻어낼 수 있다. Lighting 과정은 Enable 신호를 받는 사이클의 수만큼의 Output을 얻게 되고 소스1과 소스2처럼 중간 사이클에 Enable 신호를 내려주는 것으로 Output 값이 유효하지 않다. Lighting 과정에 적용되는 데이터 형식을 적용한 결과 Transform 과정과 Lighting 과정에 걸리는 시간이 거의 비슷하다. Lighting 과정에서 광원의 개수가 늘어남에 따라 걸리는 시간이 다르긴 하지만 실제 DirectX나 OpenGL에서 자주 사용하는 광원의 개수로 판단할 때 대부분의 경우 광원은 5개 이내고 평균 2~3개의 광원을 사용하므로 정점 1개당 6~7 cycle이 소요된다.

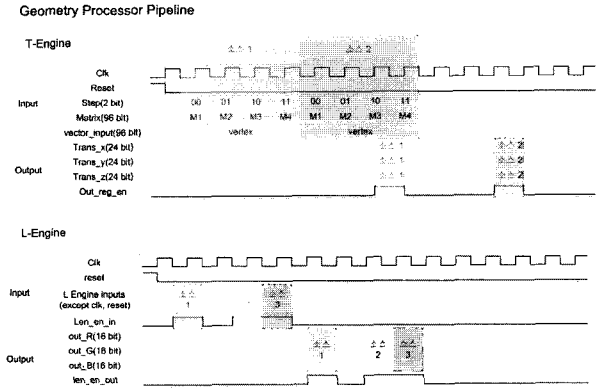

그림 13. T&L 과정의 파이프라인 비교
Fig. 13. compare with T&L pipeline.

Ⅳ. 검증 및 성능 분석

전체 3D 그래픽 처리 과정은 T&L 과정을 처리하고 Rasterization 과정을 거친 후 프레임 버퍼(Frame Buffer)에 데이터 저장을 한 후 LCD로 출력을 하는 구조로 이루어져 있다. T&L Processor를 검증하려면 Modelsim등을 통한 시뮬레이션 검증도 있겠지만 정확한 검증을 하려면 LCD를 이용한 검증환경이 필요하다. 그러기 위해선 전체 3D 그래픽 처리 과정을 전부 설계해야 한다. 이 장에서는 3D 그래픽 처리 과정에 필요한 S/W의 소개, 검증 시스템의 구조 및 실제 칩으로 구현하는 MPW 과정에 대해 소개한다.

1. SoC Platform 설계

일반 PC에서 사용되는 GPU는 기본적인 그래픽의 처리 연산을 국제 API 표준인 DirectX나 OpenGL을 사용한다. 이 둘은 서로 비슷한 성격의 API가 정의되어 있다. DirectX나 OpenGL에서 GPU로 제공하기 위한 기본적인 연산을 한 후에 GPU로 보내어지고 GPU의 연산이 끝나게 되면 연산된 값들을 반영하여 모니터로 반영되어 3D 그래픽을 출력한다.

T&L Processor를 검증하기 위한 시스템에서 사용하기 위한 S/W의 필수 조건으로 DirectX나 OpenGL이 하던 역할을 하고 일반 GPU의 요소중 하나인 Rasterization의 역할을 할 수 있어야 한다. 다음 그림 14는 SoC Platform에 사용된 S/W의 구조이다.

검증에 사용된 S/W를 크게 Vector Generator, Matrix Generator, Controller, Memory, Rasterization으로 구분한다.

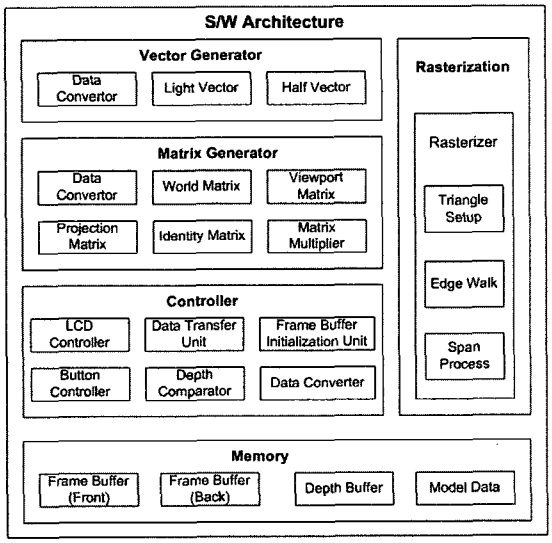

그림 14. SoC Platform에 사용된 S/W의 구조
Fig. 14. S/W architecture in SoC Platform.

Matrix Generator는 Transform Unit에서 사용할 4*4 Matrix를 생성하는데 사용한다. Transform Unit에서 사용되는 Matrix들은 총 World Matrix, Viewport Matrix, Projection Matrix들이 필요하고 4*4 Matrix끼리 곱셈을 하는 Matrix Multiplier가 필요하게 된다. 이때 단위 행렬(Identity Matrix)를 생성해주는 단위 행렬 생성기와 Transform Unit에서 사용되는 24 bits 부동소수점 데이터 타입으로 변환해주기 위한 변환기가 필요하다. 일반 C언어와 임베디드 리눅스에서 사용되는 부동소수점의 데이터 포맷은 32 bits 데이터 포맷으로 구성되어 있기 때문에 이를 24 bits 포맷으로 변환해주기 위함이다. Data Convertor를 제외하고 Matrix Generator에서 사용되어지는 유닛들은 전부 DirectX에서 제공되는 함수들이고 이를 DirectX가 없는 Software 환경에서 사용하기 위하여 함수 레벨로 만들었다.

Vector Generator는 주로 Lighting Unit에서 사용할 Vector들을 생성한다. Matrix Generator에서와 마찬가지로 Lighting Unit의 데이터 포맷에 맞는 Data Converter가 필요하고 Light Vector는 DirectX에서 지원되는 함수와 동일한 형식을 갖는다. Half Vector는 DirectX에서도 Light Vector와 View Vector만 생성해주면 자동적으로 생성해주는 구조이기 때문에 직접적으로 DirectX에서 지원되는 함수가 없이 Light Vector와 View Vector를 활용하여 만들었다.

메모리(Memory)는 프레임 버퍼(Frame Buffer)와 뎁스 버퍼(Depth Buffer), Model에 대한 정보를 담을 메모리를 필요로 한다. 프레임 버퍼는 이중 프레임버퍼를 사용하도록 구성하였다. 단일 프레임 버퍼를 사용하게 되면 사용자가 보는 화면에서 Rasterization 연산을 하는 과정이 전부 눈으로 확인하게 되어 영상이 부자연스럽다. 이중 프레임 버퍼를 사용하게 되면 Rasterization 과정을 처리하는 중인 메모리와 실제 화면에 뿌려지고 있는 메모리가 분리되어 Rasterization 연산이 끝난 메모리만을 영상으로 출력하기 때문에 자연스러운 영상을 볼 수 있다. 뎁스 버퍼는 z값에 따라 가장 가까운 곳에 있는 정보만을 프레임 버퍼로 적용하는 메모리이다. 뎁스 버퍼와 프레임 버퍼는 GPU의 SDRAM이나 DDRRAM 등으로 구현되어 있다. Model에 대한 메모리는 DirectX에서 사용되는 FVF(Flexible Vertex Format)의 역할을 하게 된다. FVF에 포함된 데이터로는 Model의 각 정점에 대한 좌표, 색, 노말 벡터에 대한

정보가 포함된다. Model 데이터가 들어 있는 메모리에서 T&L 과정의 연산 전 기본 정보를 읽어와 T&L 과정의 연산 후에 연산된 결과들을 이 메모리로 저장해주게 된다. FVF는 연산 전의 내용을 담고 있는 메모리와 연산후에 저장할 메모리 2개가 필요하다.

Controller는 LCD Controller, Button Controller, Frame Buffer initialization Unit, Depth Comparator, Data Convertor, Data Transfer Unit으로 구성한다. DirectX에서는 함수레벨로 존재하지 않는 유닛들로서 특별한 코딩없이 자동으로 연산하는 부분이지만 DirectX가 없는 검증 시스템에서 사용하기 위해 필요한 부분이다. LCD Controller는 프레임 버퍼에 저장된 영상 데이터들을 LCD로 뿌려주는 역할을 하는 제어 역할을 한다. Button Controller는 검증 시스템에서 버튼에 따라 모델이나 광원들에 대한 제어를 목적으로 설계하였다. 프레임 버퍼 초기화 유닛(Frame Buffer Initialization Unit)은 Rasterization 과정 전에 새로운 영상을 추출하기 위해 배경 이미지를 뿌려주는 과정과 뎁스 버퍼를 최하위 값으로 세팅하는 과정을 동시에 수행하여 새로운 모델에 대한 영상을 생성할 준비를 하는 단계이다. 뎁스 비교기(Depth Comparator)는 뎁스 버퍼에 저장된 데이터와 Rasterization 연산을 끝낸 정점 데이터의 z값을 비교하여 프레임 버퍼로 저장의 여부를 결정하는 유닛이다. 깊이 비교기를 통해 프레임 버퍼로 남아 있는 데이터가 z값으로 보았을 때의 가장 가까운 곳에 위치한 정점 정보만 남는다. Controller에 있는 Data Converter는 부동소수점 데이터 형식을 정수 데이터 형식으로 변환하는 용도로 사용한다. 일반 C언어에서 사용하지 않는 특수한 데이터 형식을 사용하기 때문에 이로 변환할 때 데이터의 비트 단위로 접근하기 위해 정수 데이터 형식으로 변환하여 사용한다.

Controller에 있는 Data Transfer Unit은 AMBA로 데이터를 보내거나 데이터를 받을 경우에 사용하는 유닛이다. AMBA로 데이터를 보내거나 받기 위해서 약속된 주소를 사용해야 하는데 이는 포인터를 활용한다. Rasterization 과정은 T&L Processor에서 계산되어진 정점 정보를 계산하는 입력받아 프레임 버퍼로 보내주기 위한 영상을 계산하는 단계이다. T&L Processor에서 변환된 정점정보를 삼각형으로 묶어내는 트라이앵글 셋업단계를 거쳐 삼각형의 세 정점을 정의하고, 형성된 삼각형을 에지 워크, 스팬 프로세스 과정을 거쳐는 프레임 버퍼로 전달한다.

2. Platform 기반 설계 및 검증

설계한 T&L Processor를 검증하기 위해 사용된 SoC 플랫폼의 AMBA Matrix는 그림 15와 같다. 전체 SoC Platform의 스펙은 CPU는 ARM 9 core tile로 구성되어 있고 리눅스가 포팅되어 있어서 소프트웨어 레벨로서 검증환경 제작이 가능하다. 이 리눅스로 제작된 소프트웨어와 CPU를 연결해주는 버스가 AMBA로서 설계되어 FPGA에 내장되어 있다. AMBA의 Dummy에 설계한 T&L Processor를 연결하여 ARM 과 T&L Processor간의 DATA 교환을 가능하게 하였다.

그림 16은 전체 시스템을 간단히 표현하였다. AMBA에서 검증에 필요한 구성요소들을 연결하여, T&L Processor에서 계산되어 나온 데이터들을 Software에서 받아 이 데이터를 Software Rasterizer에서 LCD로 3D 모델을 출력하는 전체 검증 시스템을 구축하였다.

모델 데이터는 DirectX 9.0c에서 제공되는 모델 파일의 좌표와 색상정보들을 텍스트로 추출하여 이를 리눅스 환경의 소프트웨어에서 읽어 들여 사용하였다.

기본적으로 연산에 필요한 Matrix와 Lighting 계산에 필요한 입력 파라미터들을 리눅스 환경의 소프트웨어에서 생성해 내어 모델 데이터에서 추출한 정보 데이터와 함께 AMBA를 통해 T&L Processor로서 전달된다. T&L 연산 후에 계산된 DATA들은 AMBA를 통해 리눅스 환경의 소프트웨어로 전달받도록 구성하였다.

T&L 처리를 통해 계산된 DATA들이 올바르게 계산

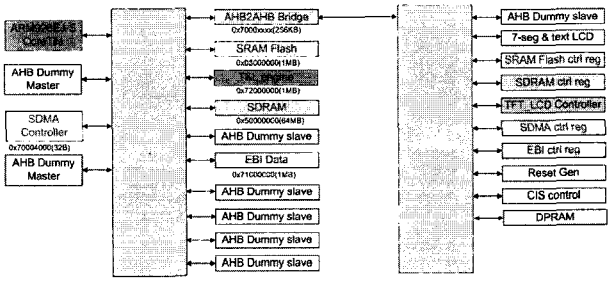

그림 15. SoC 플랫폼의 AMBA Matrix
Fig. 15. AMBA Matrix of SoC Platform.

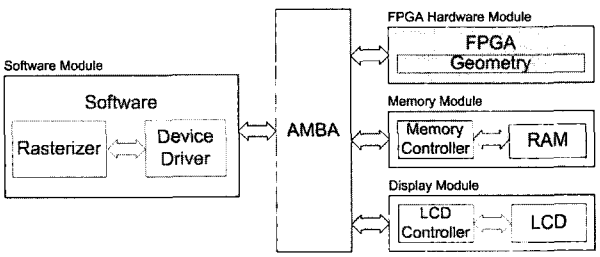

그림 16. 검증 시스템 인터페이스
Fig. 16. Verification System Interface.

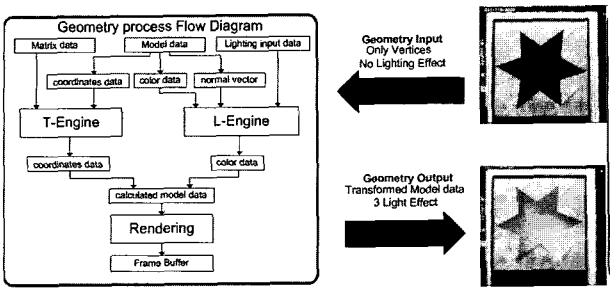

그림 17. SoC 플랫폼 기반 그래픽 처리의 Flow Diagram
Fig. 17. Flow Diagram of 3D graphics base on SoC Platform.

되어 졌는지 확인을 하려면 LCD에 출력을 하여 시각적으로 자연스러운지의 여부를 판단하여야 한다. 이 때 LCD로 출력하기 위한 연산을 하는 과정을 Rasterization이라고 하는데 실제적인 GPU 환경에서는 하드웨어로서 구성되어 있지만 T&L Processor만을 검증하기 위해 Rasterization 처리는 순수 소프트웨어 연산으로서 구성하였다.

Rasterization 처리를 통해 연산된 데이터들은 LCD Controller로 보내지게 되어 이때 LCD에 뿌려진 영상을 통해 Transform 과정과 Lighting 과정의 처리가 올바르게 동작하는지를 검증할 수 있다.

그림 17은 그래픽 처리 과정에 대한 흐름도이다. 입력으로 아무런 효과 없는 영상 정보가 입력으로 사용되어 Transform 과정과 Lighting 과정을 지나 Rasterization 과정을 거쳐 Frame Buffer에 저장되는 과정을 거치면 빛 처리가 연산된 영상이 표현된다.

3. 데모화면

검증환경에 사용한 엔젤 모델은 1197개의 정점으로 이루어져 있고 별 모델은 42개의 정점으로 이루어져 있다. 리눅스 환경의 소프트웨어에서 메트릭스를 생성하여 Transform을 하면서 동시에 빨간색, 초록색, 파란색의 색을 가진 광원을 각각 생성하여 Ambient, Specular, Diffuse 속성을 지닌 광원 3개를 정상적으로 Lighting 처리가 반영되는 것을 확인하였다.

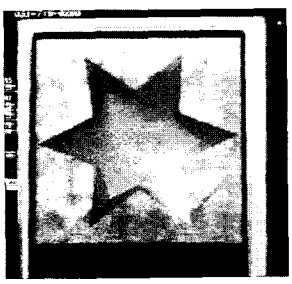

그림 18. 데모 영상(1)
Fig. 18. Demo Image(1).

그림 19. 데모 영상(2)
Fig. 19. Demo Image(2).

그림 18은 별 모델에 RGB 광원 3개를 연산할 때의 캡쳐 화면이다. R 광원은 왼쪽 아래 G 광원은 오른쪽 아래 B 광원은 위쪽에 위치하여 색상을 표현하였다. 다음 그림 19는 엔젤 모델에 RGB 광원 3개를 연산할 때의 캡쳐 화면이다. 광원의 위치를 별 모델을 사용하여 표현함으로써 광원의 위치를 쉽게 알 수 있다.

4. 성능 분석

제안하는 T&L Processor는 Magnachips 0.25 공정에서 합성하였다. Clock 주기는 100MHz로 설정하고, Operating Condition은 Worst_case_industrial로 설정하였다. 아래 표 1은 T&L Processor에 사용된 부동소수점 연산기들과 T&L Processor의 전체 구조에 대한 총 Gate의 크기를 나타낸다.

제안하는 T&L Processor는 Combinational area는 2237012 , Noncombinational area는 167083 을 기록하였고 Total Cell Area는 2405203 을 기록하였고 이 Total Cell Area 크기를 Gate Count로 표현하기 위해 17.28을 나누어 139190 Gate Count의 크기를 얻어냈다.

제안하는 T&L Processor의 동작 주파수를 측정하기 위해 FPGA를 사용하여 속도의 성능을 측정하여 본 결과 FPGA에서는 80MHz의 속도로 동작함을 확인하였다. 다음 표 2는 측정되어 나온 주파수를 이용해 계산한 성능 표이다.

표 1. T&L Processor의 합성 결과
Table 1. Synthesis result of T&L Processor.

Units	Gate Count
Adder/Subtractor Unit	3016
Multiplier Unit	3311.9
Reciprocal Unit	10836.7
T&L Processor	139190

표 2. T&L Processor의 성능
Table 2. Performance of T&L Processor.

Transform & Light		Performance
FPGA	Transform Only	20M vertices/sec @80MHz
	Transform & Lighting	20M vertices/sec @80MHz

제안하는 T&L Processor는 Transform 처리 과정과 Lighting 처리 과정을 병렬로 수행하도록 설계하였기 때문에 Transform 과정만이나 Lighting 과정과 동시에 수행하더라도 같은 속도가 소요된다. Transform Unit의 Throughput이 4 cycle이고 Lighting Unit의 광원 1개일 경우의 Throughput이 1 cycle이기 때문에 Transform 과정만을 처리할 때에 클락 주파수 80MHz 값의 나누기 4를 하게 되어 20M vertices/sec의 값을 얻어내었다. 이는 Lighting Unit 연산을 하여도 Throughput이 4 cycle인 것은 변함없기 때문에 같은 수치로 나타나게 된다. 20M vertices/sec의 속도는 30 FPS(Frames Per Second)로 동작하는 LCD에 적용할 경우 1 Frame당 666K의 정점 처리를 의미한다.

5. 타 T&L Processor의 성능 비교

제안하는 T&L Processor와 다른 그래픽 Processor의 T&L 연산을 비교하였다. 비교 대상으로는 PC용 GPU로 사용되었던 nVidia의 "Geforce2" 계열의 GPU이다. 표 3은 다른 GPU들과 성능을 비교한 표이다.[15~16]

표의 측정된 수치는 메모리의 대역폭은 고려하지 않았고, 최고 성능을 나타낼 때의 수치이기 때문에 절대적인 성능지표를 나타내지 않는다. "Geforce 2" 제품들의 경우도 메모리 대역폭의 한계로서 실제 그래픽 성능이 제약을 받게 되었다고 발표했으므로 제안하는 T&L Processor도 성능의 제약을 받지 않는다고는 말할 수 없다. "Geforce 2" 제품들과의 절대적인 비교는 불가능하지만 예전에 발매된 그래픽 Processor들과 비슷한 성능 수치를 갖는다.

표 3. T&L Processor의 비교
Table 3. Comparison of T&L Processor.

Processor	T&L Performance
GeForce256 DDR	15M vertices/sec
GeForce2 MX	20M vertices/sec
GeForce2 GTS	25M vertices/sec
GeForce2 Untra	31M vertices/sec
Programmable T&L	1M vertices/sec
Vertex Shader	12.5M vertices/sec
Proposed T&L	20M vertices/sec

Ⅳ. 결 론

본 논문에서는 3차원 그래픽을 실시간으로 가속하기 위한 T&L 처리 과정에 적합한 부동소수점 연산기를 설계하였다. T&L Processor를 설계하기 위해 이에 적합한 부동소수점 연산기를 연구하고, 이를 기반으로 T&L Processor를 설계한 다음 SoC 플랫폼과 MPW 제작을 통해 그 성능을 검증하였다.

본 논문에서 제안한 T&L Processor는 24 bits Floating Point Data Format과 16 bits Fixed Point Data Format을 적절하게 혼용하고 계산식의 병렬성을 최대한 활용하여 Transform 과정과 Lighting 과정의 균형을 맞췄고, Transform 과정만 처리할 때와 Lighting과 혼용으로 처리할 때 연산 속도가 균일하게 함으로서 T&L Processor의 효율을 최대화 하도록 하였다.

설계한 T&L Processor는 Verilog HDL을 이용하여 설계하고, Mentor사의 ModelSim을 이용하여 회로기능을 확인 후, SoC Base 1.0을 사용하는 SoC Platform을 이용하여 검증하였다. 검증을 위하여 Rasterizer를 S/W로 설계하여 ARM Processor를 처리하고 설계해 T&L Processor의 AMBA 기반의 버스로 연동하였다. FPGA로 구성된 SoC Platform에서 80MHz의 동작 속도를 보였고 135K 게이트 면적을 갖고 있음을 확인하였다.

Acknowledgement

본 논문의 실험 및 결과는 IDEC의 장비를 활용하여 제작되었습니다.

참 고 문 헌

[1] David H. Eberly, "3D Game Engine Design," Morgan Kaufmann, May, 2001.
[2] L. Garber, "The wild world of 3D graphics chips," IEEE Computer, vol. 33, no. 9, pp. 12-16, Sep. 2000.
[3] Udo Flohr, "3-D for Everyone," Byte, pp.76-88, Oct. 1996.
[4] C. B. Harell, F. Fouladi, "Graphics Rendering Architecture for a High Performance Desktop Workstation," Proceeding of SIGGRAPH '93, pp. 93-99, 1993.
[5] N. Trevett, "GLINT Gamma: A 3D Geometry and Lighting Processor for the PC," Proceeding

Notebook for HOT Chips IX, pp. 235-246, 1997.
[6] Jun-Hee Lee "Exploiting Parallelism of 3D Graphics Geometry using VLIW Geometry Processor," KAIST, Master Thesis, 1999.
[7] Cheol-Ho Jeong, "Design of an Effective Control and Execution Method for Geometry Engines and Rasterizers within Embedded 3D Graphics Accelerators," Yonsei Univ., PhD Thesis, 2003.
[8] Asger Munk Nielsen, David W.Matula, C.N.Lyu, and Guy Even, "An IEEE compliant floating-point adder that conforms with the pipelined Packet-Forwarding Paradigm," IEEE Transactions on Computers, vol.49, no.1, January 2000.
[9] J. C. Jeong, W. C. Park, W. Jeong, T. D. Han, M. K. Lee "A Cost-Effective Pipelined Divider with a Small Lookup Table", IEEE Transactions on Computers, 2004 , pp489-495
[10] P. Hung, H. Fahmy, O. Mencer and M.J. Flynn, "Fast division algorithm with a small lookup table," Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems and Computers, Vol. 2, pp. 1465 - 1468, May 1999.
[11] Won-Suk Kim, "The Implementation of Geometry Accelerator Simulator for 3D Graphic Accelerator Hardware Design," Yonsei Univ., Master Thesis, 2003.
[12] Behrooz Parhami, "Computer Arithmetic : Algorithms and Hardware Design," Oxford University Press, pp. 128-211, 2000.
[13] Israel Koren, "Computer Arithmetic Algorithms," John Wiley & Sons. Inc, pp. 35-40, 1978.
[14] IEEE Std 754-1985, "IEEE standard for binary floating-point arithmetic," IEEE, 1985.
[15] nVidia, "Technical briefs: An in-depth look at Geforce3 features," nVidia Corporation, http://www.nvidia.com/Products/GeForce3.nsf/technical.html.
[16] nVidia, "Transform, lighting and rasterization system embodied on a single semiconductor platform," nVidia Patent, Dec. 1999.

저 자 소 개

이 광 엽(평생회원)
1985년 8월 서강대학교 전자공학과 학사.
1987년 8월 연세대학교 전자공학과 석사.
1994년 2월 연세대학교 전자공학과 박사.
1989년~1995년 현대전자 선임연구원.
1995년~현재 서경대학교 컴퓨터공학과 부교수.
<주관심분야 : 마이크로 프로세서, 암호프로세서, Embedded System, 3D Graphics System>

구 용 서(평생회원)-교신저자
1981년 서강대학교 전자공학과 학사.
1983년 서강대학교 전자공학과 석사.
1993년 서강대학교 전자공학과 박사.
1983년~1993년 한국전자통신연구원 선임연구원.
1993년~현재 서경대학교 전자공학과 부교수.
<주관심분야 : Smart Power IC 설계, 나노 ESD 보호회로 설계>