논문번호 07-04-33

A design of a floating point unit with 3 stages for a 3D graphics shader engine

Kwang-Yeob Lee*
Department of Computer Engineering, Seokyeong University

Abstract

This paper presents a floating point unit(FPU) with 3 stages for a 3D graphics shader engine. It targeted to accelerate 3D graphics in portable device environments. In order to design a balanced architecture for a shader engine, we analyzed shader assembly instructions and estimated the performance of FPU with the method we propose. The proposed unit handles 4-dimensional data through separated two paths that are lead to general operation module and special function module. The proposed FPU is compiled as a form of the cascade FPU with 3 stages to efficiently handle a matrix operation with relatively low hardware overhead. Except some complex instructions that are executed using macro instructions, all instructions complete an operation in a single instruction cycle at 100MHz frequency. A special function module performs all operations in a single clock cycle using the Newton Raphson method with the look-up table.

Key-words : 3D Graphics, Shader, OpenGL ES2.0

I. INTRODUCTION

3D computer graphics is used in many different areas of entertainment, education, health service, etc. In recent years, even portable devices provide multimedia service based on real time 3D graphics due to architectural innovation and rapid growth of semiconductor technology. The early stage of 3D graphics, all operations of 3D graphics was processed with CPU. 3D graphics needs complex operation for processing a lot of data. There is an excessive burden on a CPU. In order to reduce a bottle neck that occurred by operation of graphics on the CPU, hardware engineer developed the GPU which is used to process only graphics operations. Hardware acceleration for 3D graphics using GPU allowed more various and realistic effects. Then two standard 3D programming interfaces emerged, such as OpenGL and Direct3D. Now programmers can access the functionality of GPU through that interface.

In modern GPUs, vertex shader and pixel shader are used to process geometry operation and pixel operation that is the most important part of the graphics pipeline. As a shader is used, a user is able to write a program that should be executed on the graphics hardware. It enables us to make a variety of graphics effects and handle data flexibly. These were impossible in fixed function pipeline [3].

Vertex shader changes vertex attributes such as position, normal, vertex color, and texture coordinates. Pixel shader enables pixel lighting, NPR(Non photo realistic rendering), and bump mapping by changing pixel color.

Figure 1. Example of Shader effect

A shader consists of registers and operating unit. A shader needs input register, output register, constant register, temporary register, and address register. Depending on the version of shader, several registers are added or subtracted. Operating unit is one of the most important parts for accelerating graphics. It is responsible for floating point vector operations. This paper describes a design of a floating point unit (FPU) with 3 stages for accelerating 3D graphics. Core precision is 24bit floating point referred IEEE 754. It should reduce area and power consumption. The proposed operating unit is compiled as a form of the cascade structure with 3 stages with a SIMD architecture. It enables efficiently handle vector processing of 3D graphics. 4-floating point data are operated simultaneously in FPU. Designed FPU completes all operations in a single instruction cycle at 100MHz operating frequency, except some complex instructions that are executed using macro instructions.

* Dept. of Computer Engineering, Seokyeong University
※ This work was supported by "Nano IP/SoC Innovative Promotion Group" and partially supported by "SoC Industry Promotion Center" . We use tools supported by IDEC for an FPGA implementation.
Manuscript received: Aug. 10, 2007 ; Accepted Nov. 06.2007

II. Shader Instruction Analysis

A. Functional change of shader

At first, shader model 1.0 had been defined. Next, shader model was defined up to version 1.4. Most recently, shader model 4.0 of DirectX 10 was emerged on the back of the shader model 2, 3 of DirectX 9. Figure 2 shows major changes of shader up to shader model 3.0 [1].

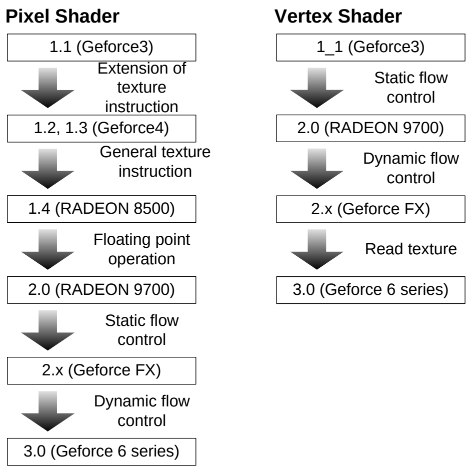

Figure 2. Functional changes of shader

In pixel shader model 2.0, an operating ability was improved in comparison with previous shader models.

B. Classification of Shader instructions

Shader assembly instructions are defined on Direct3D [1]. Direct3D is a standard of 3D programming interface. Table 1 shows Functional classification of shader assembly instructions.

Arithmetic, special function, compare, setting, move and modifier instructions are directly related to the FPU's operation. Arithmetic instruction performs a basic arithmetic operation and matrix operation. It consists of combination of multiplication and addition. Special function instructions perform special operations such as logarithm, exponential, reciprocal, and reciprocal square root. Special function unit performs full precision and partial precision instruction. Logarithm unit and exponential unit mostly perform an operation with color. In mobile environment, there is not enough difference that man can sense it. A logarithm unit and an exponential unit perform an operation with a Look-up table (LUT).

TABLE 1
Classification of Shader Instructions

TABLE 2
Instruction type and Execution stage

Each unit uses a 160 byte LUT. Reciprocal unit and reciprocal square root unit is operated with 736byte and 1Kbyte LUT each. Compare instruction generates the comparison result. Setting instruction modifies an input value. Modifier affects the result of the instruction before it is written into the destination register or copy the source register component to any temporary register component. Instructions that require complex operation are performed using macro instructions.

ALU related Instructions are classified into 3-groups according to the number of stages for the execution as shown in table 2. First group of Primitive type operations finish its operation in a single clock cycle. It needs 1-stage floating point unit. It is able to perform operations including addition, subtraction, multiplication, comparison, setting and move. Second group including MAD performs mixed addition and multiplication operation. Additional two adders are needed on 1-stage floating point unit to finish 2-stage instructions with a single instruction cycle. There is one more adder added on 2-stage floating point unit to complete 3-stage instructions in a single instruction cycle. 2, 3 stage instructions are able to perform operations using macro instructions without additional adders. However, the run-time of program is increased comparing to the case with 2 or 3 stage floating point unit

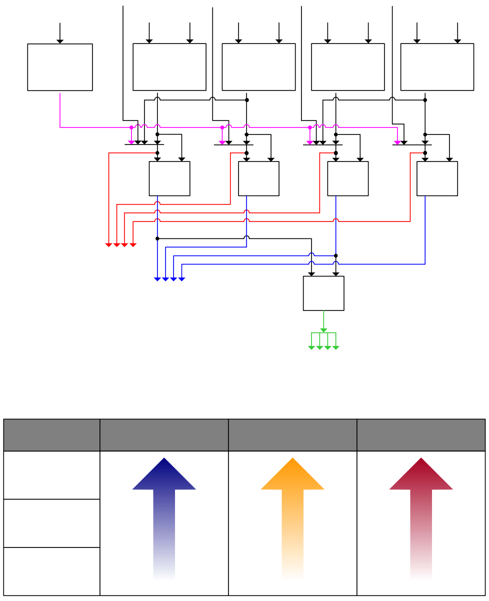

Figure 3. Execution stage and Trade-off

According to the number of floating point unit's execution stage, there are some trade-off between runtime, area, power consumption, and complexity of control. Performance of FPU with 1 stage is lowest. But its area is the small and control is relatively simple. FPU with 3 stages has an opposite characteristics. Hardware engineer have to design the hardware satisfying both balancing three factors and meeting the constraints simultaneously.

In order to decide the execution stage of FPU that should be able to execute shader program efficiently, we estimate the influence of using macro instructions for 2, 3 stage instructions. In this case, M type and DP type instruction is divided into 2 or 3 sub-instructions. It causes increasing runtime of the program. More memory space is required to keep the additional instructions. Also we compare a change of area according to the number of FPU's stages.

III. Pipeline Analysis

A. Equation

The following Equation is the approximation equations used in this study.

$$R_{SP} = R_{FI} + N_P + (N_M*4) + (N_{DP}*7) + (S_P*2) + (S_M*2) + (S_{DP}*2) \quad (1)$$

Rx is the runtime that is a required clock cycle to finish a given type of operation. SP is a shader program. FI is a first instruction of a shader program. Table 3 shows required clock cycle to complete an operation and an additional clock cycle by stall of a given type of instruction.

TABLE 3
Latency of Instruction

N_X is the number of type of instruction. Sx is the number of additional clock cycle by stall. It is defined according to the given type of instruction.

We compute the number of clock cycle to estimate performance of FPU. Bump mapping and volume fog effect program are used. Figure 4 shows the result of comparison about changes of runtime according to the ratio of M type and DP type instructions.

A designed adder for this study consists of about 3K gate equivalents. This number of gate is less than 10% of the

C.x C.y C.

A.s A.x B.x A.y B.y

x, +, x, +,

number of gate of FPU with 1 stages including special function unit (SF). The weight of adder's area as compared with entire shader engine should be less than 3%. In contrast, runtime increases rapidly according to decreased stage of FPU as shown in Figure 4.

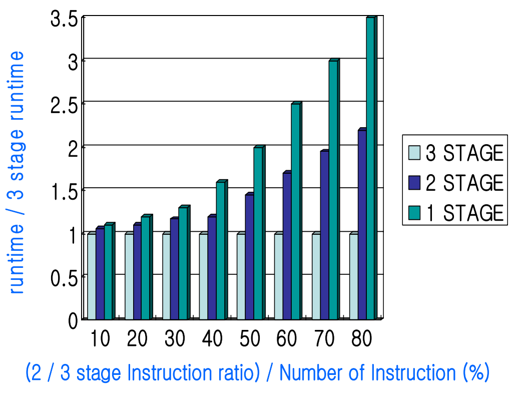

Figure 4. Execution stage and Trade-off

IV. Proposed FPU Architecture

In order to efficiently handle a matrix operation which is essential for the 3D graphics, it is necessary to optimize a dot product operation. There are 3 matrix instructions. DP3 computes the three component dot product of the source registers. DP4 computes the four component dot product of the source registers. DPH computes homogeneous dot product of the source registers. These instructions require three or four multiplication and two or three addition. Following equations describe each instruction's operation.

DP3

$$\begin{aligned}&dest.x = dest.y = dest.z = dest.w = \\ &(SourceA.x * SourceB.x) + (SourceA.y * SourceB.y) + \\ &(SourceA.z * SourceB.z)\end{aligned} \quad (1)$$

DP4

$$\begin{aligned}&dest.x = dest.y = dest.z = dest.w = \\ &(SourceA.x * SourceB.x) + (SourceA.y * SourceB.y) + \\ &(SourceA.z * SourceB.z) + (SourceA.z * SourceB.w)\end{aligned} \quad (2)$$

DPH

$$\begin{aligned}&dest.x = dest.y = dest.z = dest.w = \\ &(SourceA.x * SourceB.x) + (SourceA.y * SourceB.y) + \\ &(SourceA.z * SourceB.z) + SourceB.w)\end{aligned} \quad (3)$$

Proposed FPU with 3 stages provides an effective architecture for a matrix operation with relatively low hardware overhead.

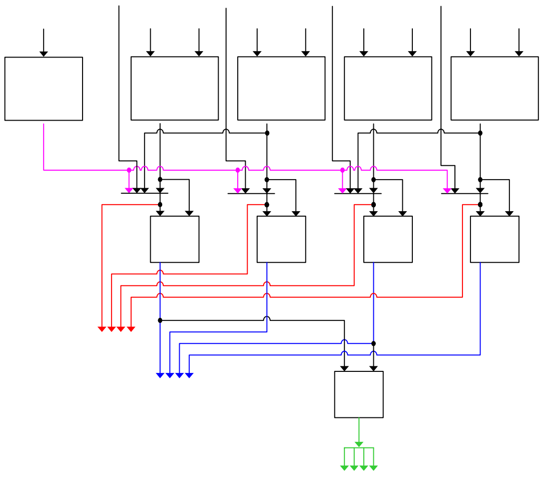

Figure 5. FPU Architecture

The process starts with the P type operation. Each component of source A and source B is entered into the first stage FPU. SF runs through the separated path with the unit for the general operation. It performs an operation using the Newton Rapson method. Designed SF performs both a full precision and a partial precision instruction. Because exponential unit and logarithm unit is mostly used to calculate relatively color, there is not enough visual difference that man can sense it. A logarithm unit and an exponential unit are designed with each 160 byte LUT. Reciprocal unit and reciprocal square root unit is designed with 736 byte and 1K byte LUT each.

Next, either an output of first stage FPU or a component of source C is selected as each component's input of second stage FPU. Four adders perform an operation for M type and DP type instructions. Finally, two outputs of second stage instruction lead to an adder then the result of DP type instruction is generated. The FPU is implemented as shown in Figure 5.

V. Implementation

This section provides the synthesized results of the proposed FPU. Since the designed chip is targeted for operating at 100MHz, the basic operating unit such as adder, multiplier, Exponential unit(SF_EXP), Logarithm unit(SF_LOG), Reciprocal unit(SF_Rcp), and Reciprocal square root unit(SF_RSQ) have to be synthesized to finish the given works within 10ns delay. The synthesized result for each component organizing the FPU is enumerated in the table 4.

TABLE 4
SYNTHESIZED RESULTS OF THE BASIC OPERATING UNITS

VI. VERIFICATION

We verified the FPU using a vertex shader engine. It is compatible with vertex shader model 1.1. The graphics performance of vertex shader using proposed FPU is 12.5M polygons/s at 100MHz operating frequency. Figure 6 and Figure 7 shows the result image. Figure 6 shows the sphere mapping image. Sphere mapping program uses DP type instructions and M type instructions. The used model consists of 21,458 vertices. Figure 7 shows the cartoon rendering image. Cartoon rendering image program consists of all type of instructions : DP4, DP3, MAX, ADD, MIN. The used model consists of 1197 vertices.

VII. CONCLUSION

We propose a design of a floating point unit with 3 stages for a 3D graphics shader engine, and present how to estimate the performance of FPU according to the number of stages. The proposed FPU is targeted for the shader in portable device environment which have limited power and small area. In the limited environment, system must meet tight cost, power consumption, and performance constraints [7]-[8]. It is difficult and important to balance these factors simultaneously. In order to design a balanced architecture for a shader engine, we define the group of shader instructions by the function and required stage of an operation, and propose a method that able to estimate the performance of FPU.

FPU is compiled as a form of the cascade FPU with 3 stages to efficiently handle a matrix operation, and it performs a SIMD floating point operations. FPU with 3 stages provide an effective architecture for matrix operations with relatively low hardware overhead. Special function instruction performs an operation using the Newton Raphson method with LUT. It completes an operation in a single clock cycle.

Proposed floating point unit consists of 9 adders, 4 multipliers, a logarithm unit, an exponential unit, a reciprocal unit, a reciprocal square root unit, and 4 setting and comparison logics. It uses 24bit floating point data with SIMD technology. Floating point unit is implemented with 0.25um CMOS technology and it takes about 60K gate count. FPU completes all operation of instructions in a single clock cycle at 100MHz operating frequency. The measured results clearly indicate the FPU is suitable for a shader engine of a portable device.

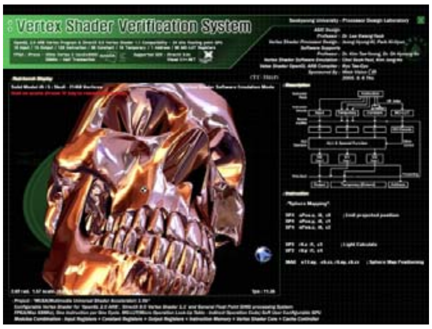

Figure 6. Result of sphere mapping program

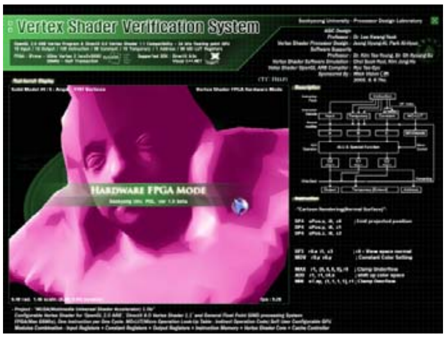

Figure 7. Result of cartoon rendering program

Unit	Cell Area	Gate Cou
Multiplier	57288.960938	3315.333
Adder	52116.480469	3016.000

REFERENCES

[1] Microsoft MSDN ASM Shader Reference (http://msdn.microsoft. com/archive/en-us/directx9_c_Aug_2005/directx/graphics/reference/ assemblyshaderlanguage/assemblyshaderlanguage.asp)

[2] Makoto Awaga(Fujitsu Limited), Tatsushi Ohtsuka, Shigeru Sasaki(Fujitsu Laboratories Ltd.), 3D Graphics Processor Chip Set. In December 1995 IEEE Micro. 37~45 page

[3] Erik Lindholm, Mark J Kilgard, Henry Moreton. A User-Programmable Vertex Engine. In ACM SIGGRAPH 2001, 12-17 August 2001.

[4] Masatoshi Kameyama, et al, " 3-D LSI core for mobile phones – Z3D, " Graphics Hardware, pp.60-67, 2003

[6] Israel Koren, " Computer Arithmetic Algorithms, " Prentice Hall, 1993

[7] Masatoshi Kameyama, Yoshiyuki Kato, Hitoshi Fuji-moto, Hiroyasu Negishi, Yukio Kodama, Yoshitsugu Inoue, Hiroyuki Kawai, " 3D graphics LSI core for mobile phone Z3D," Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, pp.60 - 67, 2003

[8] Kanako Yosida, Tadashi Sakamoto and Tomohiro Hase, "A 3D Graphics Library For 32-bit Microprocessors For Embedded Systems", IEEE Trans. On Consumer Electronis, Vol.44, pp.1107-1114, Aug.1998

Kwang Yeob Lee (Member)

K. Y. Lee studied electronics engineering at Sogang University and Yonsei University from 1979 to 1987. In 1994 he received the Ph.D from the Yonsei University. From 1989 to 1995, he was with Hyundai Electronics as a designer of System LSI. During that time, he was responsible for the design of microcontroller. In 1995, he joined the Department of Computer Engineering , Seokyeong University. His research interests include Embedded System, Mobile 3D Graphics Accelerator, SoC Design.