

中国科学技术大学

硕士学位论文

几何变换系统的硬件研究与FPGA实现

姓名：孙大成

申请学位级别：硕士

专业：电路与系统

指导教师：郭立

20070501

摘 要

近年来, 计算机图形学应用越来越广泛, 尤其是三维(3D)绘图。3D 绘图使用 3D 模型和各种影像处理产生具有三维空间真实感的影像, 应用于虚拟真实情况以及多媒体的产品上, 且多半是使用低成本的实时 3D 计算机绘图技术为基础。在初期, 所有关于图形的计算都是由图形处理单元 (GPU, Graphic Processing Unit) 来完成了, 但是随着 3D 图形学的发展, 计算量越来越大, 若所有计算工作仍然都由 GPU 来达成, 那么将会使得其应用效能下降, 无法满足今日这些实时的应用程序的需求。所以, 需要利用其它技术来辅助加快处理速度, 以提升整体效能。

通常加快算法处理速度的方法, 是使用更快速的处理器、利用并行性和专门的硬件结构等, 随着数字电路技术的发展, 尤其是在 FPGA 方面的快速进步, 让我们可以尝试另外一条途径。

3D 绘图流程大致分为: 几何转换子系统 (Geometry Subsystem) 和着色子系统 (Render Subsystem), 而本论文主要定位于几何转换子系统的研究, 尝试进行该部分的硬件描述语言实现, 并希望借次推广到整个 GPU 单元的实现, 本文主要特色表现在:

- 1、针对几何转换子系统, 提出一种硬件实现方案, 该方案能对基本的几何变换如: 平移、缩放、旋转和投影进行操作。首先构造出总体变换矩阵, 随后进行矩阵乘法运算, 再进行投影变换, 最后输出变换座标。

- 2、设计实现了单精度浮点数加法器和乘法器。图形学变换中需要进行大量的浮点数运算, 最基本的运算是浮点数加法器和乘法器。这两个模块的实现是实现整个几何转换系统所必须的。

- 3、提出一种脉动阵列结构, 用于两个矩阵的乘法运算。找到一种快捷的方法来实现矩阵相乘, 将能大大提高系统的效率。

关键词: 几何变换 图形变换 矩阵乘法 脉动阵列

Abstract

In recent years, computer graphics has been widely used, particularly in three-dimensional (3D) graphics. 3D graphics using 3D models and various image processing produce realistic three-dimensional images used in the virtual reality and multimedia products, and most of them use low-cost real-time 3D computer graphics technology as the foundation. In early days, all the graphics computation is calculated by the graphics processing unit (GPU) to complete, but with the development of 3D graphics, increasingly large amount of computation, if all calculations are still working to achieve by GPU, then the application efficiency will be declined, unable to meet today's real-time application needs. Therefore, we must use other technology to speed up the process and to improve overall effectiveness.

The most common methods of accelerating algorithm processing speed is to use a faster processor, using parallelism and specialized hardware. With the development of digital circuit technology, particularly in the FPGA rapid progress, we find another way.

3D drawing process can be divided into: the Geometry Subsystem and the Render Subsystem. This paper mainly focuses on the study of Geometry Subsystem, and try to use hardware description language for its realization. And I hope to extend this realization to the total GPU unit. The main characteristics of this paper are as follows:

1、 This paper presents a hardware implementation program for Geometry Subsystem. The program can operate the basic geometry transformation such as: translation 、 scaling、 rotation、 projection.

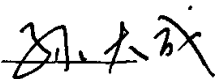
2、 The paper presents a design and implementation for single-precision floating point adder and multiplier. Graphics transform needs a large number of floating point calculations. The basic operation is floating point adder and multiplier. This two modules' realization is the requirement of the entire geometry subsystem.

3、 The paper presents a systolic array for matrix multiplication. Finding an efficient way to achieve matrix multiplication, will greatly enhance the efficiency of the system.

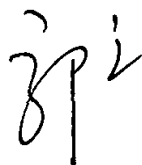
Key words: Geometric transformations Graphics transform Matrix multiplication
Systolic Array

原创性声明

本人郑重声明:本学位论文研究所有投稿论文都是本人在导师指导下,独立进行研究工作所取得的成果.文中除研究明确注明引用或参考的内容外,其它研究成果不包含如何其他人享有著作权的内容.对本论文所涉及的研究工作做出贡献的其他个人和集体,也均已在文中以明确方式示出.

签 名: 

日 期: 2007.5.22



第1章 绪 论

1.1 计算机图形学发展概述^[1]

计算机图形学^{[2][3][4]}是近30年来发展迅速、应用广泛的新兴学科,是计算机科学最活跃的分支之一。如何在计算机中表示图形、以及利用计算机进行图形的计算、处理和显示的相关原理与算法,构成了其主要研究内容。

1950年,第一台图形显示器作为美国麻省理工学院(MIT)旋风I号(Whirlwind I)计算机的附件诞生了。该显示器用一个类似于示波器的阴极射线管(CRT)来显示一些简单的图形。1958年美国Calcomp公司由联机的数字记录仪发展成滚筒式绘图仪,GerBer公司把数控机床发展成为平板式绘图仪。在整个50年代,只有电子管计算机,用机器语言编程,主要应用于科学计算,为这些计算机配置的图形设备仅具有输出功能。计算机图形学处于准备和酝酿时期,并称之为:“被动式”图形学。到50年代末期,MIT的林肯实验室在“旋风”计算机上开发SAGE空中防御体系,第一次使用了具有指挥和控制功能的CRT显示器,操作者可以用笔在屏幕上指出被确定的目标。与此同时,类似的技术在设计和生产过程中也陆续得到了应用,它预示着交互式计算机图形学的诞生。

1962年,MIT林肯实验室的Ivan E.Sutherland发表了一篇题为“Sketchpad:一个人机交互通信的图形系统”的博士论文,他在论文中首次使用了计算机图形学“Computer Graphics”这个术语,证明了交互计算机图形学是一个可行的、有用的研究领域,从而确定了计算机图形学作为一个崭新的科学分支的独立地位。他在论文中所提出的一些基本概念和技术,如交互技术、分层存储符号的数据结构等至今还在广为应用。1964年MIT的教授Steven A.Coons提出了被后人称为超限插值的新思想,通过插值四条任意的边界曲线来构造曲面。同在60年代早期,法国雷诺汽车公司的工程师Pierre Bézier发展了一套被后人称为Bézier曲线、曲面的理论,成功地用于几何外形设计,并开发了用于汽车外形设计的UNISURF系统。Coons方法和Bézier方法是CAGD最早的

开创性工作。值得一提的是,计算机图形学的最高奖是以 Coons 的名字命名的,而获得第一届(1983)和第二届(1985) Steven A. Coons 奖的,恰好是 Ivan E. Sutherland 和 Pierre Bézier,这也算是计算机图形学的一段佳话。

70 年代是计算机图形学发展过程中一个重要的历史时期。由于光栅显示器的产生,在 60 年代就已萌芽的光栅图形学算法,迅速发展起来,区域填充、裁剪、消隐等基本图形概念、及其相应算法纷纷诞生,图形学进入了第一个兴盛的时期,并开始出现实用的 CAD 图形系统。又因为通用、与设备无关的图形软件的发展,图形软件功能的标准化问题被提了出来。1974 年,美国国家标准化局(ANSI)在 ACM SIGGRAPH 的一个与“与机器无关的图形技术”的工作会议上,提出了制定有关标准的基本规则。此后 ACM 专门成立了一个图形标准化委员会,开始制定有关标准。该委员会于 1977、1979 年先后制定和修改了“核心图形系统”(Core Graphics System)。ISO 随后又发布了计算机图形接口 CGI(Computer Graphics Interface)、计算机图形元文件标准 CGM(Computer Graphics Metafile)、计算机图形核心系统 GKS(Graphics Kernel system)、面向程序员的层次交互图形标准 PHIGS(Programmer's Hierarchical Interactive Graphics Standard)等。这些标准的制定,为计算机图形学的推广、应用、资源信息共享,起到了重要作用。

70 年代,计算机图形学另外两个重要进展是真实感图形学和实体造型技术的产生。1970 年 Bouknight 提出了第一个光反射模型,1971 年 Gourand 提出“漫反射模型+插值”的思想,被称为 Gourand 明暗处理。1975 年 Phong 提出了著名的简单光照模型. Phong 模型。这些可以算是真实感图形学最早的开创性工作。另外,从 1973 年开始,相继出现了英国剑桥大学 CAD 小组的 Build 系统、美国罗彻斯特大学的 PADL. 1 系统等实体造型系统。

1980 年 Whitted 提出了一个光透视模型. Whitted 模型,并第一次给出光线跟踪算法的范例,实现 Whitted 模型;1984 年,美国 Cornell 大学和日本广岛大学的学者分别将热辐射工程中的辐射度方法引入到计算机图形学中,用辐射

度方法成功地模拟了理想漫反射表面间的多重漫反射效果；光线跟踪算法和辐射度算法的提出，标志着真实感图形的显示算法已逐渐成熟。从 80 年代中期以来，超大规模集成电路的发展，为图形学的飞速发展奠定了物质基础。计算机的运算能力的提高，图形处理速度的加快，使得图形学的各个研究方向得到充分发展，图形学已广泛应用于动画、科学计算可视化、CAD/CAM、影视娱乐等各个领域。

计算机图形学之所以能在它短短的 30 多年历史中获得飞速发展，其根本原因是图形为传递信息的最主要媒体之一。人们要利用计算机进行工作，必须有任何计算机之间传递信息的手段——人机界面。人机界面从早期的读卡机及控制板上的开关、指示灯发展到键盘和字符中断，再发展到基于键盘、鼠标、光笔等输入设备和光栅显示器的图形用户界面，而最终必然过渡到带给用户身临其境感觉的三维用户界面——虚拟环境（虚拟现实）。人机界面的发展过程正好对应着计算机技术从初级到高级的发展过程。计算机图形学来源于生活、科学、工程技术、艺术、音乐、舞蹈、电影制作等，反过来，它又大大促进了这些领域的发展。

1.2 本文使用工具介绍

1.2.1 软件

VHDL 语言^{[5][6]}

VHDL 的英文全名是 Very High Speed Integrated Circuit Hardware Description Language, 诞生于 1982 年。1987 年底, VHDL 被 IEEE 和美国国防部确认为标准硬件描述语言。自 IEEE 公布了 VHDL 的标准版本, IEEE.1076 (简称 87 版) 之后, 各 EDA 公司相继推出了自己的 VHDL 设计环境, 或宣布自己的设计工具可以和 VHDL 接口。此后 VHDL 在电子设计领域得到了广泛的接受, 并逐步取代了原有的非标准的硬件描述语言。1993 年, IEEE 对 VHDL 进行了修改, 从更高的抽象层次和系统描述能力上扩展 VHDL 的内容, 公布了新版本的 VHDL, 即 IEEE 标准的 1076—1993 版本 (简称 93 标准)。现在, VHDL 和 Verilog 作为

IEEE 的工业标准硬件描述语言，又得到众多 EDA 公司的支持，在电子工程领域，已成为事实上的通用硬件描述语言。

应用 VHDL 进行工程设计的优点是多方面的：

(1) 与其他的硬件描述语言相比，VHDL 具有更强的行为描述能力，从而决定了它成为系统设计领域最佳的硬件描述语言。强大的行为描述能力是避开具体的器件结构，从逻辑上描述和设计大规模电子系统的重要保证。

(2) VHDL 丰富的仿真语句和库函数，使得在如何大系统的设计早期就能查验设计系统的功能可行性，随时可对设计进行仿真模拟。

(3) VHDL 语句的行为描述能力和程序结构决定了它具有支持大规模设计的分解和已有设计的再利用功能。负荷市场需求的大规模系统高效，告诉的完成必须有多人甚至多个代发组共同并行工作才能实现。

(4) 对于 VHDL 完成的一个确定的设计，可以利用 EDA 工具进行逻辑综合和优化，并自动的把 VHDL 描述设计转变成门级网表。

(5) VHDL 对设计的描述具有相对独立性，设计者可以不懂硬件的结构，也不必管理最终设计实现的目标器件是什么。而进行独立的设计。

正是因为 VHDL 的这些优点，本文模块的实现，均采用 VHDL 语言进行描述。并且采用 Mentor Graphics 公司的 Modelsim 软件进行模拟仿真。

Modelsim 仿真工具

Modelsim 是 Model Technology (Mentor Graphics 的子公司) 的 HDL 硬件描述语言仿真软件，可以实现 VHDL, Verilog 以及 VHDL/Verilog 混合设计的仿真，是 HDL 仿真业界中应用最为广泛的产品。

ModelSim 是业界最优秀的 HDL 语言仿真器它提供最友好的调试环境，是唯一的单内核支持 VHDL 和 Verilog 混合仿真的仿真器。是作 FPGA/ASIC 设计的 RTL 级和门级电路仿真的首选，它采用直接优化的编译技术、Tcl/Tk 技术、和单一内核仿真技术，编译仿真速度快，编译的代码与平台无关，便于保护 IP 核，个性化的图形界面和用户接口，为用户加快调错提供强有力的手段。全面支持 VHDL 和 Verilog 语言的 IEEE 标准，支持 C/C++ 功能调用和调试。

ModelSim具有强大的交互DEBUG功能、衰减测试支持和高内存利用效率功能,为仿真设计提供了广阔的范围。除此之外,Modelsim还能够与C语言一起对HDL设计文件实现协同仿真。同时,相对于大多数的HDL仿真软件来说,Modelsim在仿真速度以及稳定性上也有明显优势。这些特点使Modelsim越来越受到EDA设计者、尤其是FPGA设计者的青睐。

Matlab 工具

MATLAB 是矩阵实验室 (Matrix Laboratory) 之意。除具备卓越的数值计算能力外,它还提供了专业水平的符号计算,文字处理,可视化建模仿真和实时控制等功能。

MATLAB 的基本数据单位是矩阵,它的指令表达式与数学、工程中常用的形式十分相似,故用 MATLAB 来解算问题要比用 C, FORTRAN 等语言完相同的事情简捷得多。

简单介绍一下 MATLAB 的主要特点:

1) 语言简洁紧凑,使用方便灵活,库函数极其丰富。MATLAB 程序书写形式自由,利用起丰富的库函数避开繁杂的子程序编程任务,压缩了一切不必要的编程工作。由于库函数都由本领域的专家编写,用户不必担心函数的可靠性。可以说,用 MATLAB 进行科技开发是站在专家的肩膀上。

2) 运算符丰富。由于 MATLAB 是用 C 语言编写的, MATLAB 提供了和 C 语言几乎一样多的运算符,灵活使用 MATLAB 的运算符将使程序变得极为简短。

3) MATLAB 既具有结构化的控制语句(如 for 循环, while 循环, break 语句和 if 语句),又有面向对象编程的特性。

4) 程序限制不严格,程序设计自由度大。例如,在 MATLAB 里,用户无需对矩阵预定义就可使用。

5) 程序的可移植性很好,基本上不做修改就可以在各种型号的计算机和操作系统上运行。

6) MATLAB 的图形功能强大。在 FORTRAN 和 C 语言里,绘图都很不容易,但在 MATLAB 里,数据的可视化非常简单。MATLAB 还具有较强的编辑图形界面的能

力。

7) MATLAB 的缺点是,它和其他高级程序相比,程序的执行速度较慢。由于 MATLAB 的程序不用编译等预处理,也不生成可执行文件,程序为解释执行,所以速度较慢。

8) 功能强大的工具箱是 MATLAB 的另一特色。MATLAB 包含两个部分:核心部分和各种可选的工具箱。核心部分中有数百个核心内部函数。其工具箱又分为两类:功能性工具箱和学科性工具箱。这些工具箱都是由该领域内学术水平很高的专家编写的,所以用户无需编写自己学科范围内的基础程序,而直接进行高,精,尖的研究。

9) 源程序的开放性。开放性也许是 MATLAB 最受人们欢迎的特点。除内部函数以外,所有 MATLAB 的核心文件和工具箱文件都是可读可改的源文件,用户可通过对源文件的修改以及加入自己的文件构成新的工具箱。

1.2.2 硬件

FPGA^{[7][8]}

FPGA 是英文 Field Programmable Gate Array 的缩写,即现场可编程门阵列,它是在 PAL、GAL、EPLD 等可编程器件的基础上进一步发展的产物。它是作为专用集成电路(ASIC)领域中的一种半定制电路而出现的,既解决了定制电路的不足,又克服了原有可编程器件门电路数有限的缺点。FPGA 采用了逻辑单元阵列 LCA (Logic Cell Array) 这样一个新概念,内部包括可配置逻辑模块 CLB (Configurable Logic Block)、输出输入模块 IOB (Input Output Block) 和内部连线 (Interconnect) 三个部分。

FPGA 的基本特点主要有:

- 1) 采用 FPGA 设计 ASIC 电路,用户不需要投片生产,就能得到合用的芯片。
- 2) FPGA 可做其它全定制或半定制 ASIC 电路的中试样片。
- 3) FPGA 内部有丰富的触发器和 I/O 引脚。
- 4) FPGA 是 ASIC 电路中设计周期最短、开发费用最低、风险最小的器件之

一。

5)FPGA 采用高速 CHMOS 工艺,功耗低,可以与 CMOS、TTL 电平兼容。可以说, FPGA 芯片是小批量系统提高系统集成度、可靠性的最佳选择之一。

FPGA 的设计开发流程^[9]主要包括四个步骤:设计输入 (Design Entry)、仿真 (Simulation)、综合 (Synthesis) 及布局布线 (Place & Route)。

1. 设计输入 (Design Entry)

Summit 公司的 VisualHDL、Mentor 公司的 Renoir、Aldec 公司的 ActiveHDL。均支持图文混合的层次化设计。三者都提供 PC 版本, VisualHDL 还有工作站版本。

图形输入包括状态图、真值表、流程图、方框图等。其中流程图输入方法是 Renoir 独有的。文本输入包括 VHDL 和 Verilog, 上述工具都而且同时支持两种语言。

Renoir 支持 HDL2GRAPH, 即从 VHDL/Verilog 语言模块转换到图形。这一特性有助于分析已有 HDL 的语言结构。

ActiveHDL 提供 HDL 语法高亮显示、自动产生文本结构、自动格式化文本等非常有益的文本编辑浏览特性。Renoir 和 VisualHDL 甚至不提供最基本的 HDL 语法高亮显示。

2. 仿真 (Simulation)

仿真包括功能仿真和时序仿真。其中, 功能仿真在布局布线之前; 时序仿真在布局布线之后。仿真工具有 Mentor 公司的 Modelsim 和 Aldec 公司的 ActiveHDL, 二者同时支持 VHDL 和 Verilog 的仿真。Cadence 公司也提供仿真工具, 似乎对 Verilog 的支持更强, 没有评估过。Modelsim 同时提供 PC 和工作站版本, ActiveHDL 只有 PC 版本。

其中 Modelsim 是工业界应用最广的仿真工具, 已经成为事实上的标准。界面简洁, 仿真速度快, 功能强大而稳定。

ActiveHDL 提供图示化仿真激励输入, 而且有 testbench 的自动生成模板, 这些特性都是独有的。而且语言的在线帮助系统非常好。

3. 综合 (Synthesis)

综合工具实现从 HDL 语言到 FPGA 或 ASIC 网表的生成。目前有 Synopsys 公司的 FPGA Compiler II、Mentor 公司的 Exemplar 和 Synplify 公司的 Synplicity。三者都有 PC 和工作站版本。

其中 FPGA Compiler II 是应用最广的，只支持 FPGA 的综合。Synopsys 公司另外有 ASIC 的综合工具。

Exemplar 同时支持 FPGA 和 ASIC。

Synplicity 界面简洁，据说综合速度比其他二者更快。

4. 布局布线 (Place & Route)

布局布线采用 FPGA 厂商提供的工具。Xilinx 有 Foundation Series 和 Alliance Series 两个系列，分别支持几十门级以下和以上的 FPGA。Altera 的两个系列是 MaxPlusII 和 Quartus。

这一整个设计流程都会有各类 EDA 软件的参与。通常是以某一主要 EDA 软件(由器件厂商提供)为核心，再辅以若干个第三方软件来进行的。Xilinx 公司的 FPGA 开发流程如图：

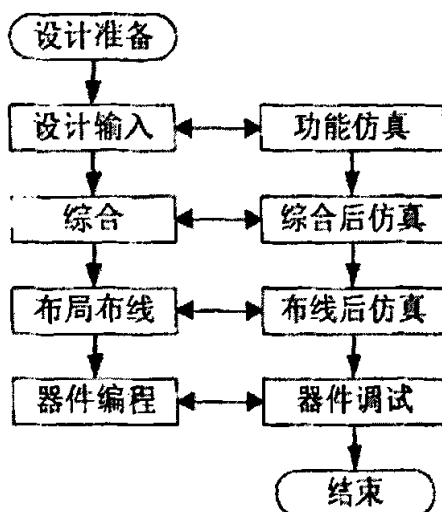


图 1.1 Xilinx FPGA 开发流程

1.3 课题背景及意义

1.3.1 真实感图形绘制^{[10][11]}

图形绘制技术是计算机图形学的一个重要领域，它研究的是如何将几何定义的模型变成可视的图像。真实感图形绘制技术是图形学绘制技术发展的结果，它的目标是根据几何场景的造型、材质和光源分布，将其转变成跟真实场景在视觉效果上非常相似的图像，可以与实拍照相相媲美。在近三十年的时间里，随着硬件技术和计算机图形学的高速发展，真实感图形绘制技术取得了举世瞩目的成就，并且已广泛应用于CAD / CAM、计算机动画、虚拟现实、科学计算可视化等众多领域，为社会的发展与进步做出了重大贡献。

几十年来，CAD/CAM 技术的飞速发展，对许多行业的设计和生产过程产生了巨大影响，使其自动化程度达到了前所未有的高度。在机械设计方面，传统的产品设计通常需制作实物模型来检查设计效果，尤其是那些对外形美感要求较高的产品，一般要先按图纸制作样品，再根据样品反映的问题进行多次修改，耗费大量人力、物力和时间。若采用真实感图形显示技术，则可方便地在屏幕上显示产品在各种视角下的逼真图形，并且可直接在计算机上对设计参数进行交互修改和绘制，直到满意为止。既可大大减少反复生产样品的时间，缩短设计周期，也节约了生产样品的资金和人力；在建筑方面，建筑师往往要花费很大的精力绘制效果图，如果采用真实感图形绘制技术，不但可以把这方面的工作交给计算机处理，还可以结合虚拟现实技术进行建筑物体的虚拟漫游，这样，建筑师和用户在建筑物还处于设计阶段就可以非常直观地了解建成后的外观和内部空间的布置及采光效果等，做到胸有成竹。

计算机动画是真实感图形绘制技术的又一重大应用领域。在电影和电视广告中，要达到某些特定的效果，往往需要进行许多次的试验和拍摄。那些危险场面、破坏场面和需要动物配合的场面不但令电影工作者们非常为难，而且提高了电影的制作费用，还可能对人员造成伤害。另外，有些事物和景象更是现实世界无法找到的，比如科幻片或者灾难片，无法实景拍摄。通过计算机绘制出高度真实感的虚拟场景，这些困难都可以方便地解决。可以说，真实感图形

绘制技术给影视工作者和艺术家提供了一个尽情发挥想象力的舞台。

虚拟现实是当前的一個热门研究课题，它在飞行训练、战斗模拟、计算机游戏等许多领域具有十分广阔的应用前景。真实感图形绘制技术是基于图形的虚拟现实的核心技术之一，直接关系到人机交互速度和图像输出质量。

科学计算可视化是当前的另一热门课题，它已在医学、地球科学、天文物理、化学、机械工程等许多方面取得了巨大成就，将大批工程技术人员和医务人员从大量繁琐的数据中解放出来。真实感图形是科学计算可视化的输出手段之一，绘制质量将直接关系到工程技术人员对原始数据的理解和医务人员对病人病情和病因的判断，具有突出的现实意义。

真实感图形绘制技术的应用领域非常广阔，它的影响正迅速地向工业生产和社会生活的各个领域渗透。可以预料，在未来社会里它必将发挥更为重要的作用。

1.3.2 研究意义

在计算机图形学中，引人注目的研究方向之一是图形的真实感问题。长期以来，图形真实感问题一直是计算机图形学研究的一个主要课题。这是由于计算机技术的飞速发展，其应用越来越广泛，某些应用领域(计算机艺术造型、计算机制作、三维游戏、计算机娱乐和广告动画设计等)已把高度真实感的图形作为其发展的目标，要求我们能逼真地在计算机屏幕上再现真实世界。

真实感图形绘制技术的应用发展的速度是如此之快，随之而来的是绘图计算量的扩大，使得一般的 GPU 单元在处理速度上有些吃紧。正因为如此，借助辅助硬件资源，来提高图形处理单元(GPU)处理速度的需求变得越来越普遍。

计算机图形学除了个人计算机及工作站上大量应用外，随着电子技术的发展，诸如 PDA、汽车导航系统、移动电话等领域都得到了大量的应用，对于这类产品，开发出低功耗，低成本的专用图形加速器有着广阔的应用前景，同时也是科技创新，走拥有自主知识产权的必由之路。

1.4 本文的主要研究内容和结构安排

本文在认真研究了3D图形学原理的基础上, 尝试采用Top. Down的方法对GPU几何变换部分进行FPGA设计实现, 作为GPU的辅助功能模块, 减轻GPU的运算负担, 以达到加速的目的。主要研究内容有: 3D图形学技术的发展状况; 3D绘图管线介绍; 浮点数运算单元的实现; 矩阵构造单元的实现, 矩阵乘法单元的实现等。

文章包括五个章节, 其结构安排如下: 第一章为绪论部分, 主要介绍 3D 图形学技术的发展状况, 以及本文所用到的软件和硬件; 第二章介绍 3D 绘图硬件设计目的, 3D 绘图管线, 座标变换和 3D 图形硬件; 第三章介绍了 IEEE 浮点数表示标准, 并分析了浮点数加法器和乘法器的实现; 第四章给出了几何变换部分硬件实现的方案, 并对其中的关键部分: 矩阵构造单元和脉动阵列结构的矩阵乘法进行了详细分析; 第五章为总结与展望。

第 2 章 3D 图形硬件相关研究

2.1 引言

近二十年来, 电脑绘图已成为人机介面中, 最重要的资料显示方法, 并广泛的运用在各种应用之中。例如计算机辅助设计 (CAD), 医疗方面的影像处理, 以及电脑游戏等等。尤其是 3D 电脑绘图, 到了 90 年代初期, 很快的从高深的技术领域, 扩展到非技术领域上的运用。而多媒体 (multimedia) 以及虚拟现实 (virtual reality) 产品现在越来越普及。不但是人机介面上的重大突破, 更在娱乐有用中扮演着重要的角色。而这些的应用, 多半是以低成本的即时 3D 电脑绘图技术为基础。2. D 电脑绘图是一种常用以将资料和内容表现出来的普遍技术, 特别是在互动应用。另一方面, 3D 绘图是电脑绘图中一股越来越大的分支, 它使用 3D 模型和各种影像处理来产生具有三维空间真实感的影像。使用者一直要求 3D 绘图系统有更好的效率和更低的价格, 而半导体技术的进步使它成为可能。VLSI 技术使个人电脑能够做到工作站级的绘图处理器的效果。VLSI 技术的长足进步之下, 因为量产所带来的价格优势, 使得 3D 电脑绘图的加速硬件越来越普及, 甚至成为电脑中人机介面的标准配置。

2.2 3D 绘图硬件的设计目标^[12]

即时着色 (real.time Rendering)

我们可以把需要快速反应时间的 3D 应用分成以下三大类:

1. 即时应用 (Real.time Application): 每秒必需输出数十张画面, 通常为 24 到 30 张。
2. 交谈应用 (Interactive Application): 一张画面的产生时间需少于一或数秒。
3. 批次应用 (Batch Application): 一张画面的产生时间大于数十秒。

由于多媒体和虚拟现实的盛行, 电脑绘图也被要求要拥有即时着色的能力。然而巨大的计算负荷, 资料的存取及记忆体频宽的问题, 使得大多数的 CPU 难以负荷 3D 绘图的要求。因此为了速度的要求, 使用特定的硬件来解决使最自然

的方式。

影像品质 (Image Quality)

3D 绘图被广泛地使用在许多领域, 上至 CAD, 下至电脑游戏都可以见到它的应用。随着电脑效能的增进, 对影像的品质要求也越来越高。故可以预见的是每个框架(frame)的复杂度会越来越高, 同时使用一些新的演算法, 如 texture mapping, 也可以有效地增加影像的真实度。综合以上所述, 计算复杂度, 资料储存, 和资料的负荷也会随之增加。

系统整合 (System Integration)

现在的消费性电子产品中, 多媒体扮演了越来越重要的角色。一个多媒体系统整合了文字, 绘图, 影像, 和声音。为了使这么多种的资料同步, 多媒体系统必须在短时间内处理相当大量的运算操作。因此需要不同的处理器, 来加强一个多媒体系统中, 不同部分的处理能力。3D 处理器使其中的一员, 它需要和其它的处理器协同工作。在这样的考虑之下, 许多设计使用了最常见的 frame buffer 系统来加快 3D 绘图的速度。虽然有记忆体频宽的瓶颈但比起重新设计一个与目前架构不同的新架构, 沿用旧架构可以很简单地将 3D 绘图处理器整合进现有 PC 的多媒体显示系统中。另一方面, 为了达成相容性和系统整合, 设计师不只要考虑硬件的设计, 同时也要顾忌软件的开发。一些已经成熟的软件 API 可以用来当做设计硬件时的参考, 因为这些 API 正是使这些 3D 绘图处理器得以推广的动力。

2.3 标准 3D 绘图着色管线

为了加快 3D 绘图的速度, 标准 3D 绘图着色管线^[12]使用一种最常见被使用到的方式。接下来介绍目前最常见的 3D 绘图管线。

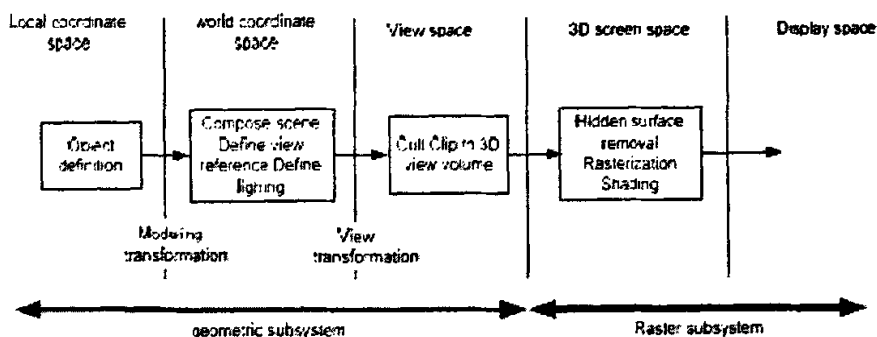


图 2.1 3D 绘图管线

在图 2.1 中, 可以看到整体着色的管线根据不同的座标系统而分割成数个部分。物体定义 (Object definition) 里的物体使一些 3D 模型的描述定义, 所使用的座标系统参考其本身的参考点, 故称之为本地座标 (Local Coordinate Space)。要合成一张 3D 影像时, 各个不同的物体从资料库中被读取, 并转换至一个统一的世界座标 (World Coordinate)。其次, 就像真实世界中的照相机一般, 我们需要定义观测站 (View Point) 的位置。由于绘图系统硬件解析度的限制, 必须将连续的座标转换至含有 X 及 Y 座标, 硬件深度值 (常称为 Z 座标) 的 3D 荧幕空间。在做完隐藏面的消除 (hidden surface removal) 以及将物体以像素的方式描绘出来后, frame buffer 会保留结果的图像并且将其输出至荧幕上。通常将处理这类标准管线的各单元分成两大部分:

1. 几何转换子系统 (Geometry Subsystem)
2. 着色子系统 (Raster Subsystem)

其两个系统与整个系统上的阶层关系表示于图 3.2。

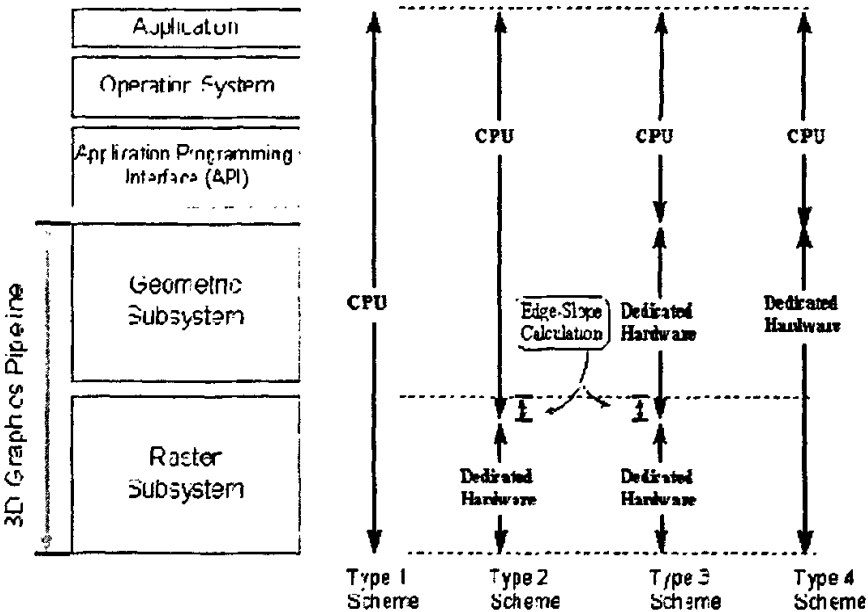


图 2.2 3D 绘图管线硬件实现上的分类

目前已有许多种方式来实现这条标准的 3D 绘图管线, 如图 3.2 所示, Type1 完全使用软件, Type2 和 Type3 利用特定的硬件 (Dedicated Hardware) 加速部分子系统之功能。第二种中只将着色子系统以硬件来实现, 第三种中更跨及几何转换子系统。Type4 则是完全以特定硬件来实现 3D 绘图管线, 又称完全硬件实现 (Total hardware solution), 因为不需考虑到 3D 绘图管线内部介面来与他种系统相容, 在演算法上的选择与架构设计更具弹性, 也是目前最普遍使用的。

在标准管线中, 大部分的座标都是以浮点数的形式来表示, 因此浮点运算的最大需求是几个转化的部分。每个 primitive 的顶点都需要做 modeling 和 view 的 transformation, 而这些转换是 4x4 的矩阵运算, 所以“乘加”是座标转换中最主要的一种运算动作。接下来, clipping units 将部分在观测空间中的 primitives 做适当的切割, lighting stage 利用如 Phone illumination model 对 primitives 做处理。在这儿所需要的运算有加, 乘, 向量内积等。通常 lighting stage 是整个系统加速时会遇到的最大障碍。

2.3.1 3D 几何管线

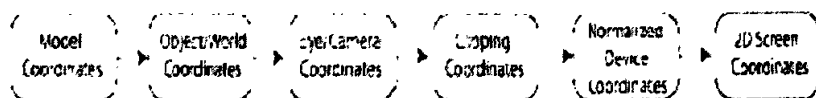


图 2.3 3D 几何管线

由图可以发现几何转换子系统一共有两个坐标变换，这是几何转换子系统最重要的运算所在，多数为 4×4 的矩阵运算。在本地坐标，世界坐标以及观察坐标中，其坐标值均为三维且连续的。为了保持坐标转换的精确度，大多使用浮点数来表示坐标值。在这部分最重要的工作就是坐标的变换，因此提供这些坐标转换运算功能的单元通常被称之为几何转换子系统。而在 3D 绘图处理中，这部分的加速硬件通常被称之为“几何引擎”(Geometric Engine)。另外，在前级有 primitive 的选取动作，可以决定 primitive 是否在可视空间中，减少不必要的运算。而那些仅有部分在可视范围内的 primitive，则会在 clipping 中被削切成符合观测空间的形状。由 View space 转成 Screen space 时由一个 Perspective divide 的动作，将观测空间坐标除以一个常数 w ，并且利用 3D Screen Mapping 将坐标转换至 x, y 荧幕坐标 (Screen Coordinate)。以备接下来的着色子系统使用。当使用 Gouraud Shading 时，lighting module 仅存在于几何转换子系统中。

2.3.2 3D 坐标变换介绍

在计算机图形学中最通常的表示物体的方法是多边形网孔模型。一个物体由一系列多边形连接而成，而每个多边形又由一系列顶点 (x, y, z) 连接而成。如下图所示：

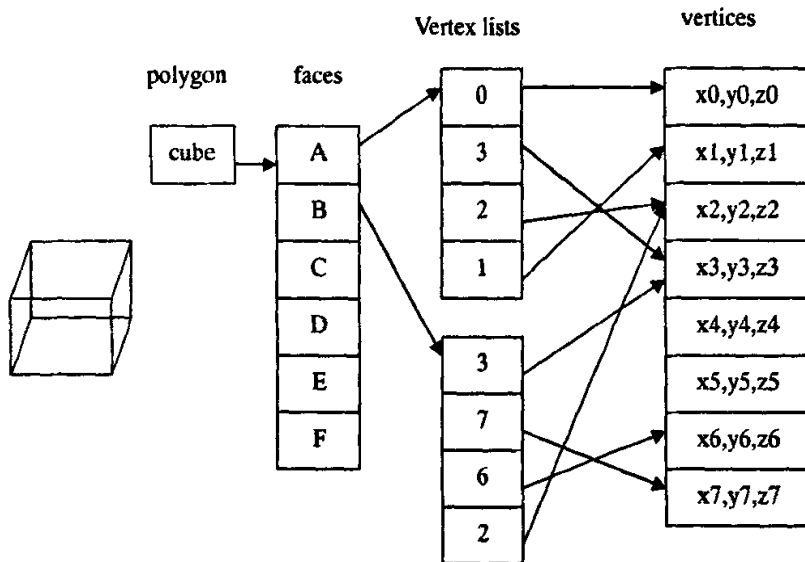


图 2.4 物体模型的数据结构

3D图形变换^{[2][3][4]}包括旋转, 缩放, 裁减和平移几个部分. 一系列图形变换可以合成为一个变换式. 一个物体的3维顶点集合可以通过线形变换为另外一组点的集合. 在图形学中通常用矩阵来表示各种变换. 利用矩阵表示时, 若顶点 V 要进行平移, 缩放, 旋转变换, 可以用如下表示:

$$V' = D + V, V' = S \times V, V' = R \times V$$

其中 D 为平移矩阵, S 和 R 分别为缩放和旋转矩阵.

如果要将所有变换都能在硬件结构上实现, 对于所有变换运用规范化的表示是必须的. 由于在笛卡儿坐标系下不能将平移变换和其他变换结合起来, 因而引入了奇次坐标系. 笛卡儿坐标系和奇次坐标系之间的转换也很容易. 在奇次坐标系中, 顶点数据 $V(x, y, z)$ 表示为 $V(X, Y, Z, w)$ (w 不等于0), 两坐标之间的关系为:

$$x = X/w, y = Y/w, z = Z/w$$

在计算机图形学中一般 w 取为1, 则顶点的矩阵表示为 $[x, y, z, 1]^T$

用奇次坐标表示的平移变换这时可以表示为矩阵相乘的形式:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = T \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

因此, 在奇次坐标系下任何变换的矩阵表示为:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} A & D & G & J \\ B & E & H & K \\ C & F & I & L \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

该通用矩阵可以划分为如下4个模块:

$$\left[\begin{array}{c|c} \text{Scaling \& rotation} & \text{Translation} \\ \hline \text{Part of the homogeneous} & \\ \text{representation} & 1 \end{array} \right]$$

缩放矩阵:

$$V' = S \times V$$

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

旋转矩阵:

对物体的旋转必须选定旋转轴, 虽然旋转轴的位置可以任意, 不过都可以转化为平行坐标轴的形式, 绕X, Y, Z坐标轴的旋转变换矩阵为:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y = \begin{bmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

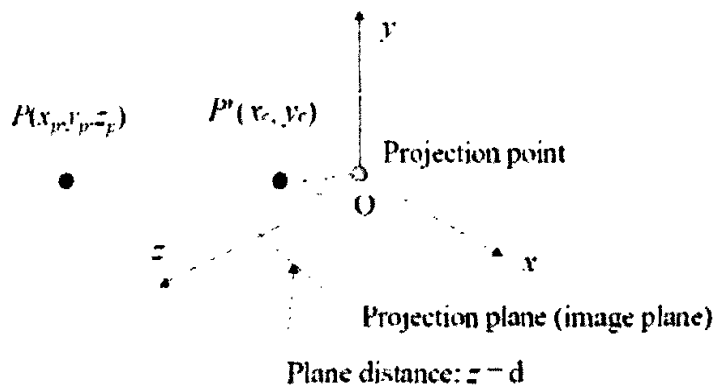
投影矩阵:

目的是将视体变换为一个单位立方体。这个立方体的顶点是 (.1, .1, .1) 和 (1, 1, 1)。单位立方体也称为规范视体。投影主要有两种方式: 正投影(平行投影)和透视投影。

正投影的可视体通常是一个矩形盒子，正投影可以把这个视体变换为单位立方体。正投影的主要特点是平行线在投影变换以后依然平行，这种变换是平移和缩放的组合。

透视投影比正投影复杂，这种投影方式下，物体距离相机位置越远，投影之后就变得更小。平行线在投影之后可以相交。透视投影变换与人类感觉物体大小的过程非常相似。从几何说，视体也称为锥体，即一个以矩形为底面的被截金字塔，同样也可以将这个锥平截投体变换为单位立方体。

正投影和透视投影都可以通过4X4的矩阵来实现，在任何一种变换以后，都可以认为模型位于归一化处理之后的设备坐标系中。投影变换后产生的图象的z坐标系将不存在，投影以后模型从三维变成二维，这里以透视投影为例：



该阶段也相当于一个矩阵变换过程：

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \times \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} x_p & 0 & 0 & 0 \\ 0 & y_p & 0 & 0 \\ 0 & 0 & z_p & 0 \\ 0 & 0 & z_p/d & 0 \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \\ z_p \\ z_p/d \end{bmatrix}$$

为了使w为1，进行一下除法即可得到投影变换后的坐标 $(\frac{d}{z_p}x_p, \frac{d}{z_p}y_p, d)$ 。

2.4 3D 图形硬件

2.4.1 图形处理单元

在大多数现代计算机中，3D图形管线的复杂计算是通过专用图形卡来实现的，而在过去这些计算完全都是在CPU上完成的。随着3D图形学的发展，计算量

越来越大, 耗费太多的CPU时隙, 会使得其应用效能下降, 为此, 现代的计算机都引入图形卡与通常CPU协同工作, 专门处理3D图形运算中大量的矩阵和向量运算。这些具有独自微处理器的图形卡称为图形处理单元 (GPU)。

GPU 处理矩阵和向量运算的速度如此的快, 以至于人们开始将 GPU 用于其他非图形学的用途, 如生成立体图象。与此同时, 人们也开始为老的 GPU 采用 ASIC 的制造技术来制造寻求替代品。考虑到可重配置的自由性, 人们更青睐于用 FPGA 来实现 GPU。

2.4.2 FPGA 作为替代和辅助

计算机图形算法的计算量是庞大的, 正是这个原因使得人们寻求各种手段来加速算法。传统的加速方法是使用更快速的处理器、利用并行性和专门的硬件结构等, 随着数字电路技术的发展, 尤其是在 FPGA 方面的快速进步, 我们可以尝试另外一条途径。

FPGA 是完全可重配置的硬件器件, 编程简单并且价格便宜。它的多样性的配置特点, 使得它能用于如何硬件的模拟, 甚至于复杂的 GPU 单元。这使得任何人都能在任何时候尝试实现自己的新想法和算法。

FPGA 具有的一些优点, 使得它可能作为 GPU 的一种替代, 但是仍然存在一些障碍, 这样一个障碍就是速度。目前来说 FPGA 的速度还是较 ASIC 慢, 这是一个很大的缺点, 尤其是在图形处理中速度是个必须的要素。另外一个 ASIC 优于 FPGA 的是它的低的功率消耗, 这在移动器件中尤为重要。FPGA 的另外一个劣势的可重配置的面积需求, 需要使用宝贵的存储空间。尽管如此, FPGA 在速度和规模上的改善速度均高于 GPU 和 ASIC, 所以按照摩尔定律, FPGA 肯定会在短期内得到改善。

正是由于这个原因, 虽然FPGA可能永远不能完全代替专用的图形处理单元, 但是我觉得它可以很好地与GPU和CPU结合, 提供更加可靠的并行性运算服务, 或者节省更多的时隙。FPGA可以提供的其他优点就是能很好地修正BUG, 使得功能趋向正确。

第3章 浮点运算单元

3.1 引言

图形学变换中需要进行大量的浮点数运算，最基本的运算是浮点数加法器和乘法器。本章介绍了单精度浮点数加法器和乘法器的设计。章节组织如下：3.2节对底层模块进行了介绍，3.3节介绍了浮点数加法器的设计实现，3.4节为浮点数乘法器的设计与实现，3.5节对本章进行了总结。

3.2 IEEE单精度浮点数

在二进制数据系统中，非负整数是用‘1’和‘0’来表示的。整数A表示成

$$A = \sum_{i=0}^{n-1} 2^i a_i。$$

该数据表示法的最大不足是不能表示负数。Sign-Magnitude表示法和二进制补码表示法既能表示正数，也能表示负数。前者与二进制数据系统类似，只不过在数的最左边添加一个额外的符号位，来表示正或负。二进制补码表示法也很类似。这样一个整数A可表示为：

$$A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i。$$

这个表示法简化了大多数基本的算术运算。为了在这些格式中包括分数表示，引入固定位和根植点的概念，这称为定点数表示系统。这个转换的限制就是非常大和非常小的数都不能被表示。此外数的表示范围也被限制在了一定的比特位。

浮点数在科学表示法中用于表示实数，格式一般包括3个部分：符号位，尾数和指数。最大有效位MSB用于表示数的符号，‘1’表示负数，‘0’表示正数。紧接着MSB位的是分数的尾数部分，指数表示数的有效值。这样，一个实数就可以表示为：

$$\pm 1 \text{ mantissa} \times 2^{\pm \text{exponent}}。$$

IEEE754标准^[3]是一种被广泛使用的浮点数表示格式，定义了单精度32位和双精度64位两种表示。单精度格式有1位符号位，8位尾数和23位指数。双精度具有1个符号位，11个尾数位和52个指数位。

单精度格式		
Sign(1 bit)	Exponent(8 bits)	Mantissa(23 bits)
双精度格式		
Sign(1 bit)	Exponent(11 bits)	Mantissa(52 bits)

表3. 1是IEEE754标准的一些参数：

表3. 1 IEEE754标准参数		
参数	单精度	双精度
Word Width	32	64
Exponent width	8	11
Exponent Bias	127	1023
Mantissa width	23	52
Number of values	1.98×2^{31}	1.99×2^{63}
Range(base 10)	10^{-38} to 10^{38}	10^{-308} to 10^{308}

信号与图像处理方面的应用需要浮点数表示，因为这些领域需要使用很大和很小的数，且需要很高的精确度。正因为其充分大的数的表示范围和精度，浮点数表示很受欢迎。

3. 3 参数化的底层模块

浮点数算术运算需要频繁的调用基本的功能模块，如顶点加法器，定点乘法器和其他一些逻辑功能模块。底层硬件模块也都进行了参数化，为高层模块反复调用提供方便。本节对这些模块的结构，功能用途做个简单介绍。

(1) 定点数加法和减法器

加法和取整操作需要一个定点数加法器。Parameterized_adder是一个提供该功能的专用模块。定点数减法器是另外一个使用频繁的操作，Parameterized_subtractor模块提供该功能。这些模块的输入和输出位宽都进行了参数化处理。

(2) 定点乘法器

该模块在浮点数乘法操作中用用到, 模块的输出位宽是输入信号的两倍, 输入信号的位宽被参数化。

(3) 变量移位器

好几种浮点数算术运算需要用到变量移位器。

Parameterized_variable_shifter用来完成此功能, 输入变量位宽, 移动方向和需要移动的位数都进行了参数化。

(4) 延时模块

该模块用于将输入信号延时几个期望的时钟周期, 这在流水线电路中对信号进行同步所必须的。参数化信号为: 输入, 输出和时延值。

(5) 优先级编码器 (priority encoder)

在规格化的步骤中, 需要判定输出结果的具有 ‘1’ 的MSB。

Parameterized_priority_encoder接收一个输入, 然后产生一个无符号的定点数输出。输入和输出信号的位宽可进行参数化。

(6) 多路选择器 (multiplexer)

Parameterized_mux模块具有两个参数化的输入信号和一位的控制信号, 控制信号选择输出, 实现2选1的功能。

(7) 比较器 (comparator)

Parameterized_comparator模块的功能是用于比较两个输入 (A和B), 指示是 $A>B$, $A<B$ 还是 $A=B$, 输入位宽可以进行参数化。

(8) 去归一化 (denormalization)

浮点数的规格化表示时, 一位 ‘1’ 表示数的整数部分, 但是在浮点数的IEEE表示时, 为了增加存储效率, 这一位被忽略了。然而当执行算术运算时, 这一位需要重新引入。Denorm模块的功能就是将这一位 ‘1’ 插入到数中。由于这仅仅是一个简单的不需要后续逻辑的操作, 所以其延时为零。指数和尾数可以进行参数化。

(9) 取整与归一化 (rounding and normalization)

Round_norm模块用于将前级数学运算的输出转化为具有标准位宽和规格的IEEE浮点数表示。

3.4 浮点数加法器

浮点数加法时一个复杂的算数运算，需要进行指数对齐和尾数改变。操作数的尾数部分，必须要在加法和减法操作之前使之一致。首先，将操作数的尾数进行比较，小的尾数必须依据大的尾数进行调整，小的数的指数也要相应的进行移动。然后进行定点数的加法和减法操作，结果值最后再调整到期望的格式。

浮点数加法运算的基本步骤为：

- 1. 指数对齐操作
- 2. 尾数的加减
- 3. 修正和归一化

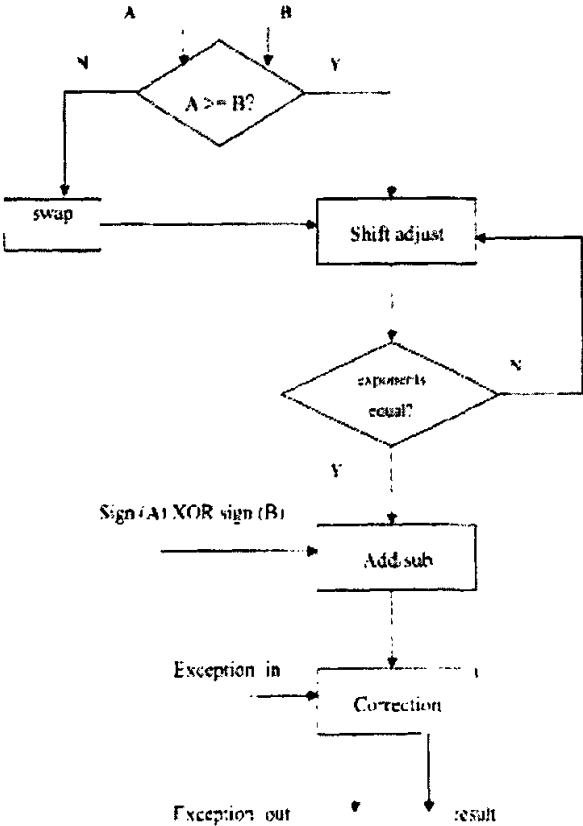


图3.1 单精度浮点数加法流程图

Fp_add模块实现对两个浮点数的加法, 分为以下模块:

Swap模块

输入A和B首先通过parameterized_comparator进行比较，然后通过parameterized_mux进行交换，这就使得大数一直在‘1’端口。

Shift_adjust模块

较小数的指数会依据尾数的差别相应地右移多少位。
parameterized_subtractor模块用于生成尾数的差别值，差别值再输入到parameterized_variable_shifter模块执行移位操作。在移位操作中，小的操作数会引入一个保护位防止信息流失，这样的结果就是Fp_add模块会多出额外的一位。

Addition/subtractor模块

两个输入操作数的符号位输入到一个XOR门，以决定是执行加法操作还是减法操作。XOR的输出结果作为parameterized_mux模块的控制信号。该模块选择参数化的加法/减法器来对输入指数进行加减。

Corrcetion模块

如果输入端发生异常，那么就会传送到输出端。当两个数具有相同的数量级但符号位相反时，结果就为零。如果在指数的加法过程中发生了溢出，输出指数就需要向右移动一位，尾数也要增加1。输入值送到加法器之前需要先经过denorm模块进行反规格化，因为需要在数前面加上一个‘1’。加法器的输出使用round_norm模块进行取整规格化，以保持和输入有相同的格式。下图为浮点数加法的前端和后端处理模块：

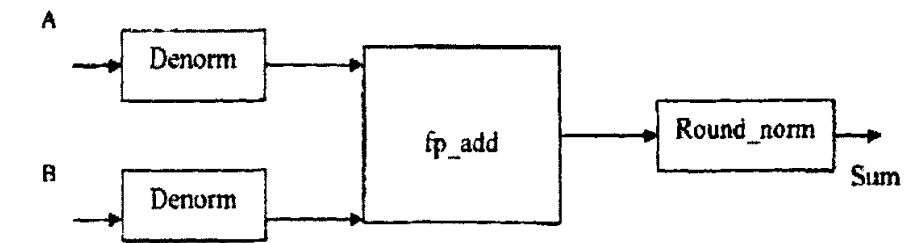


图3.2 单精度浮点数加法器模块图

3.4.1 模拟与仿真

采用Modelsim对两个浮点数进行模拟的结果如下。表3.2是输入和输出数的十进制和IEEE标准的表示方法，同样也给出了32位单精度浮点数的十六进制表示法，以便于观察。时序图见图3.3。

表3.2 加法器输入和输出数的十进制和IEEE标准的表示

	IEEE. 754 32.bit	十进制	十六进制
A	0 10000101 100011111111111111111111	99.99999	42C7FFFF
B	0 01101110 01001111100010110101100	0.00001	3727C5AC
A+B	0 10000101 100100000000000000000000	100.00000	42C80000

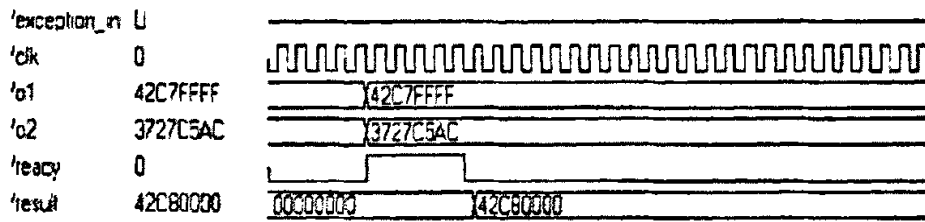


图3.3 加法器时序图

从上面的时序图可知浮点数加法器需要6个时钟周期，具体见表3.3。

表3.3 加法器时延

运算模块	延时（时钟周期数）
去归一化	0
浮点加	4
取整归一化	2
合计	6

3.5 浮点数乘法器

相比较于加法，浮点数的乘法要简单的多。如果s1, m1, e1为第一个数的符号位，尾数和指数，s2, m2, e2为第二个数的符号位，尾数和指数，那么给定

的这两个浮点数的乘积为：

$$(-1)^{s1 \oplus s2} \times (m1 \times m2) \times 2^{(e1 + e2)}$$

乘法运算的算法流程包括以下四个步骤：

1. 对两个操作数的符号位进行XOR（异或）操作，得到乘积的符号位
2. 乘积的尾数由两个操作数的尾数进行定点乘法得到
3. 乘积的指数由两个操作数的指数相加得到。指数部分需要进行偏移操作以增加浮点数的值域，这样两个指数的和就包括了两次偏移，那么结果就要进行以此减法操作。

4. 对尾数进行规格化

由于前三步彼此独立，所以可以并行执行。

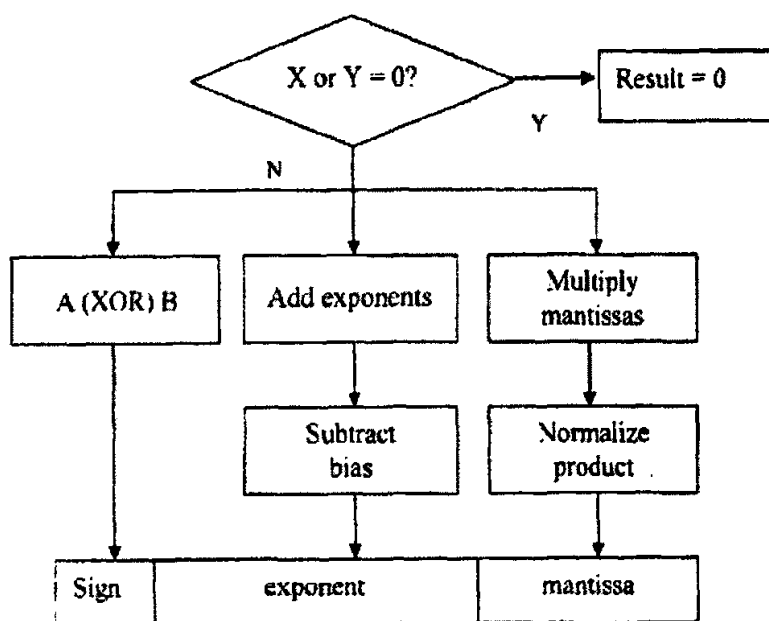


图3.4 单精度浮点数乘法流程图

Fp_mul乘法模块对两个浮点数进行如下操作：操作数的指数通过parameterized_adder模块进行相加，再通过parameterized_subtractor模块从和中减去多的偏移。parameterized_multiplier模块对尾数进行操作。由于采用的是定点乘法器，那么得到的尾数积的具有输入操作数的两倍的位宽。如果

输入数中的任何一个为零或者输入ready信号为零，parameterized_mux就会将积清零。输入到乘法模块的操作数首先要通过denorm模块进行处理，乘法器的输出还要经过round_norm模块进行处理再输出。

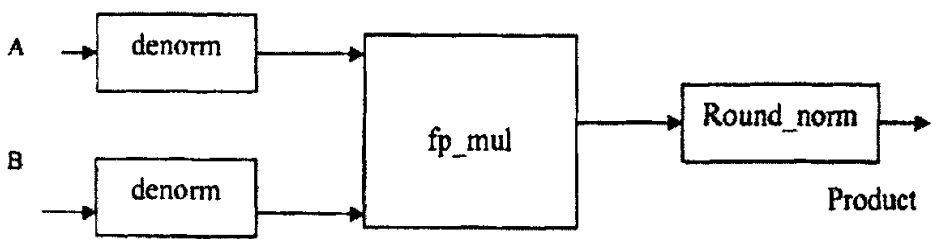


图3.5 单精度浮点数乘法器模块图

3.5.1 模拟与仿真

下面对两个浮点数的乘法进行模拟，表3.4为输入和输出数的十进制和IEEE标准表示。时序图见图3.6。

表3.4 乘法器输入和输出数的十进制和IEEE标准的表示

	IEEE.754 32.bit	十进制	十六进制
A	0 10000101 100011111111111111111111	99.99999	42C7FFFF
B	0 01101110 01001111100010110101100	0.00001	3727C5AC
A*B	0 10000101 100100000000000000000000	0.0009999999	3A83126E

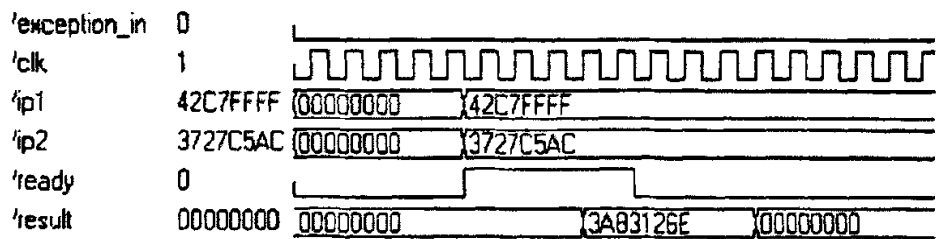


图3.6 乘法器时序图

浮点数乘法器需要耗费5个时钟周期，表3.5为详细信息。

表3.5 乘法器时延

运算模块	延时（时钟周期数）
去归一化	0
浮点乘	3
取整归一化	2
合计	5

3.6 本章小结

本章是对单精度浮点数算术运算的实现。首先对IEEE754标准进行了介绍，随后对基本的底层模块进行了说明，在此基础上分别讨论了单精度浮点数加法器和乘法器的设计与实现，并利用Modelsim进行了功能仿真。

第 4 章 几何变换系统的设计

4.1 引言

随着计算机绘图规模的需要,借助辅助硬件资源,来提高图形处理单元(GPU)处理速度的需求越来越普遍。几何变换是图形处理单元(GPU)算法实现中的重要一环,其运算速度直接关系到 GPU 的运算效率。结合计算机绘图及 FPGA 的特点,本章采用 Top. Down 的方法对 GPU 几何变换单元部分进行了 FPGA 设计实现。所有结构模块均实现了 RTL 级建模,并对其中较复杂的矩阵构造单元和矩阵乘法器模块给出了详细的描述。最后借助电子设计自动化工具(EDA)对整个模块进行了验证综合,结果表明符合设计需求,该方案能很好地完成几何变换功能,资源耗费低,最高工作频率可达 120MHz。

本章组织结构:第一部分引言,第二部分为实现方案部分,第三部分为验证综合结果,结论在第四部分给出。

4.2 几何变换硬件架构设计^[14]

接下来讨论 3D 图学中的几何处理系统的设计,将一座标做几何转换,计算方式为矩阵一向量相乘,会得到一新座标:

$$\begin{pmatrix} X_tmp \\ Y_tmp \\ Z_tmp \\ W_tmp \end{pmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

其中

$$\begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \text{ 称为转换矩阵。}$$

一般的几何转换处理器共有三种基本功能:

1. 缩放 (scaling): 对一物体上的顶点做 scaling, 就是将这物体放大或缩小, 其转换矩阵如下:

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

其中 S_x 代表 x 轴方向的放大或缩小, S_y 代表 y 轴方向的放大或缩小, S_z 代表 z 轴方向的放大或缩小。

2. 平移 (translation): 其转换矩阵如下:

$$\begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

dx, dy, dz 分别为 x 轴, y 轴与 z 轴方向的位移量。

3. 旋转 (rotation): 在三维坐标上可分为以 x 轴, y 轴或 z 轴为中心的旋转, 转换矩阵分别为:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} R_y = \begin{bmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

在3D管线的几何处理单元中, 基本的处理元素为三角形的顶点 (vertex), 器座标表示法为 $v = [x \ y \ z \ w]^T$, 分别对这些顶点做如上所述的一些转换得

到心的座标:

$v' = \tau(v)$ 每个几何变换 τ 都可表示成矩阵 M_τ , 所以

$$v' = M_\tau \cdot v$$

Rotation, translation, scaling, 一般来说, 我们会对一个顶点做一个以上的几何变换, 例如:

$$v_1 = M_{T0} \cdot v_0, v_2 = M_{T1} \cdot v_1$$

或者可以先将两个矩阵相乘:

$$v_2 = (M_{T1} \cdot M_{T0}) \cdot v_0$$

会得到相同的结果。

假设对 v 做一连串的转变 $\tau_1, \tau_2, \dots, \tau_k$,

得到新座标 $v' = M_{Tk} \dots M_{T2} \cdot M_{T1} v$

先将 $A = M_{Tk} \dots M_{T2} \cdot M_{T1}$ 算出, 最后才计算矩阵一向量乘法得到

v' 。

针对几何变换实时性的要求, 本文提出如下的FPGA实现方案, 总体框图流程如图4.1所示:

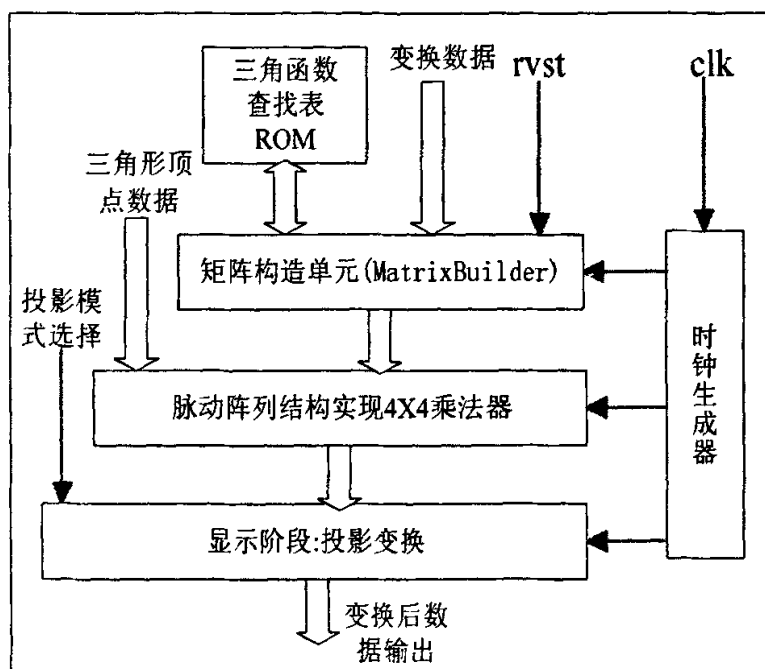


图4.1 总体实现方案流程

工作原理:

已知：输入端给定变换数据：平移量 tx , ty , tz ；缩放量及参考点 sx , sy , sz ；
旋转轴位置及旋转角度

分以下三步进行实现：

- (1) 依据模式选择信号 $rvst$ 和变换数据，得到总的变换矩阵
- (2) 将变换矩阵与变换前三角形顶点坐标向量相乘
- (3) 判断投影方式，输出对应的结果

下面分节对各个功能部分进行介绍。

4.3 矩阵构造单元

我们知道，对一个顶点可能同时要来进行好几种变换操作，如果一步一步的进行，就需要好积此矩阵乘法运算，非常耗费时间，但是如果先将几种变换矩阵合成一个变换矩阵，然后再相乘，就值需要进行一次矩阵乘法运算，很显然这种方法要好的多。

对矩阵构造模块的实现，我们可以选用一个RVST信号（RVST即Rotation Version Scale Translate的缩写）来决定组合的变换矩阵到底是什么样子的。代码书写中，我使用case语句来判断输出矩阵的每个元素是多少。

例如，如果RVST为0011，这表示没有旋转变换操作，只有缩放和平移变换，那么 Sx , Sy , Sz 和 Tx , Ty , Tz 就会被发送到矩阵构造单元的输出响应位置。旋转模式RV 用2位bit数表示，表示4种模式：00表示无旋转，01表示绕X轴旋转，10 表示绕Y轴旋转，11表示绕Z轴旋转。选择任何一种模式，输出端就有相应的矩阵输出。模块端口图如图4.2所示：

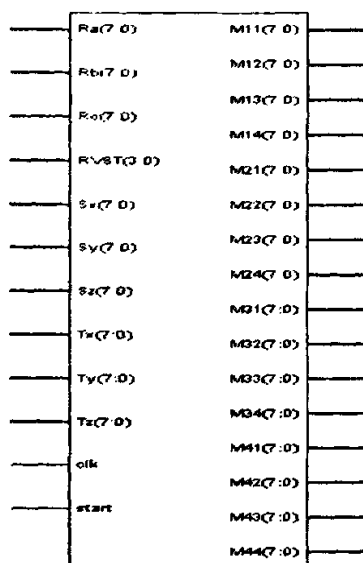


图4.2 矩阵构造单元端口图

如果同时具有旋转和平移（如RVST=1X01，X为任意值），输出矩阵就是在相应的旋转矩阵上，将偏移量加到最后一列上。

一共有四种形式的变换矩阵，如图4.3所示：

$$\begin{pmatrix} S_x & 0 & 0 & T_x \\ 0 & \cos \alpha * S_y & -\sin \alpha & T_y \\ 0 & \sin \alpha & \cos \alpha * S_x & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos \alpha * S_x & 0 & \sin \alpha & T_x \\ 0 & S_y & 0 & T_y \\ -\sin \alpha & 0 & \cos \alpha * S_x & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

绕 X 轴旋转+缩放+平移

绕 Y 轴旋转+缩放+平移

$$\begin{pmatrix} \cos \alpha * S_x & -\sin \alpha & 0 & T_x \\ \sin \alpha & \cos \alpha * S_y & 0 & T_y \\ 0 & 0 & S_z & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} S_x & 0 & 0 & T_x \\ 0 & S_y & 0 & T_y \\ 0 & 0 & S_z & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

绕 Z 轴旋转+缩放+平移

基本矩阵

图4.3 四种输出矩阵组合样式

VHDL语句编程时，通过如下语句实现：

```
case RVST(3 downto 2) is
```

```
when "00"    => 矩阵1
when "01 "   => 矩阵2
when "10 "   => 矩阵3
when "11 "   => 矩阵4
when others => 矩阵5
end case
```

从上图可知,当我们要将旋转和缩放合而为一的时候(如 $RVST=IXIX$),问题就出现了,最终变换矩阵的一些元素会公用相同的位置。为了解决该问题,需要引入三个乘法器,用于计算 $\cos\theta$ 和 S_x , S_y , S_z 的乘积,最后将乘积放入矩阵的正确位置。

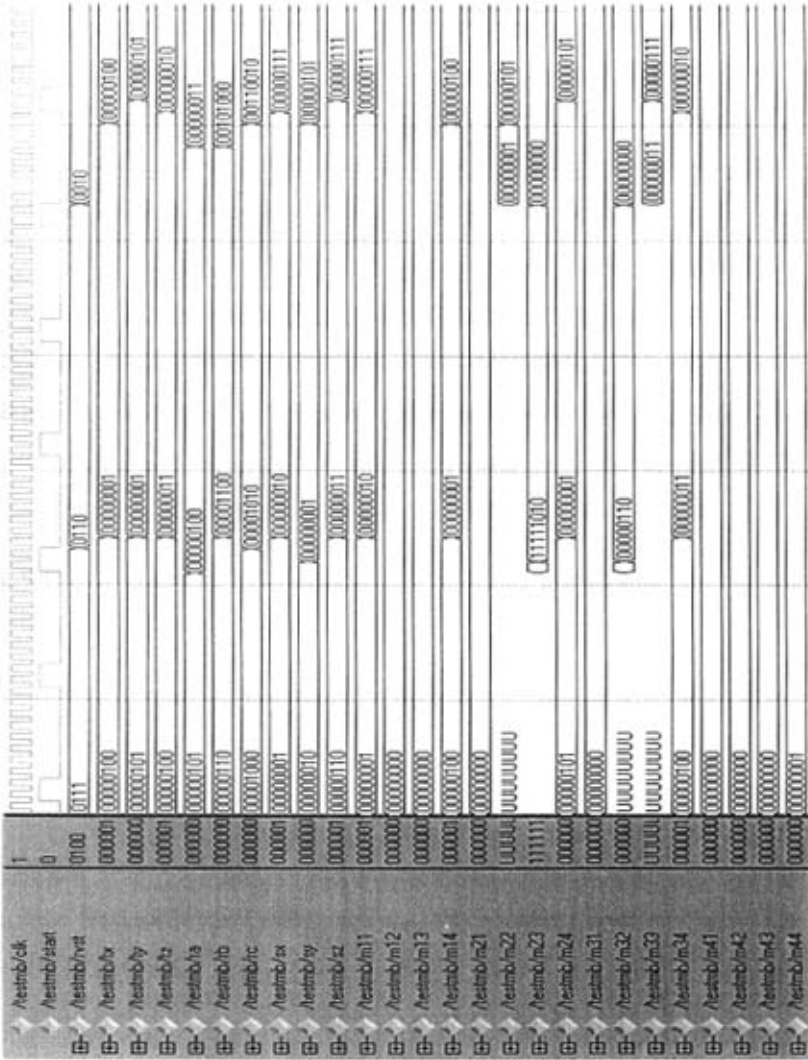


图4.4 矩阵构造模块的RTL时序仿真图

4.4 三角函数查找表

运算中，需要用到三角函数值为了使得设计简单，需要一个计算三角函数 $\cos \theta$ 和 $\sin \theta$ 的模块。鉴于查找表方法的简单性，我们建立2个查找表来存储算好的 $\cos \theta$ 和 $\sin \theta$ 的值。

鉴于三角函数的周期性特点，我们可以只存储一个周期内的值，另外考虑到在一个周期内还可以化简，我们只需要存储1/4周期内的值。

为了试验方便，这里将1/4周期内的值分成128份。整个查找表结构如图4.5所示：

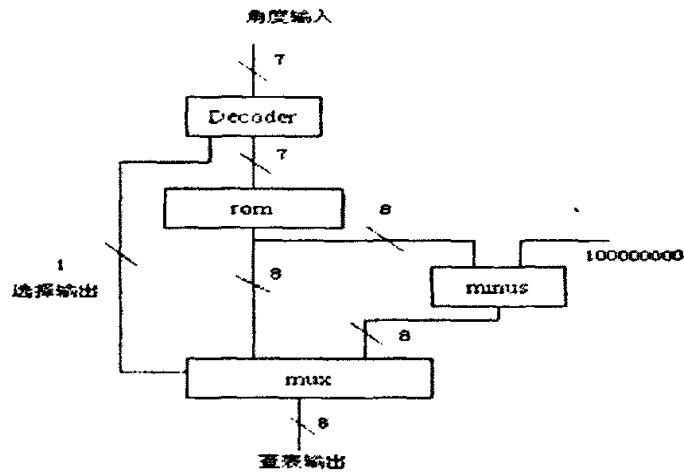


图4.5 查找表结构图

Decoder为地址映射单元，用于将超出1/4周期的角度变换到1/4周期内。

简单的VHDL语句描述如下：

```
entity Angle_dec is
    port(angle_in:in std_logic_vector(7 downto 0);
          dec_out:out std_logic_vector(6 downto 0);
          sign_out:out std_logic
    );
end Angle_dec;
architecture Behavioral of Angle_dec is
begin
    process(angle_in)
        variable temp:std_logic_vector(7 downto 0);
```

```
begin
    if angle_in<="01111111" then dec_out<=angle_in(6 downto
0);
                                sign_out<='0';
        elsif angle_in>"01111111" then
            temp:=angle_in."01111111";
            dec_out<="11111111".temp(6 downto 0);
            sign_out<='1';
            else dec_out<="00000000"; sign_out<='0';
        end if;
end process;
end Behavioral;
rom存储了1/4周期的函数值，Rom数据由Matlab程序产生。
```

minus用于产生负数的补码形式。

Mux多路选择其单元根据Decoder产生的符号判断位选择输出。

Cos查找表元件例化语句如下：

```
u1: Angle_dec port map(angle, signal1, sig0);
u2: rom_cos port map(signal1, signal2);
u3: mux_out port map(signal2, sig0, dataout);
```

sin查找表元件例化语句如下：

```
u1: Angle_dec port map(angle, signal1, sig0);
u2: rom_sin port map(signal1, signal2);
u3: mux_out port map(signal2, sig0, dataout);
```

图4.6是cosθ和sinθ模块的RTL时序仿真图：

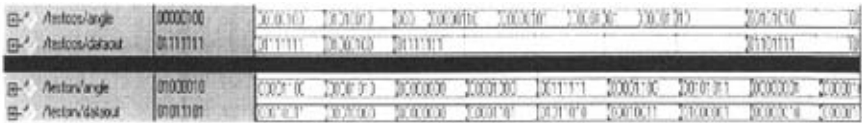


图4.6 cosθ和sinθ模块的RTL时序仿真图

4.5 矩阵乘法器设计^{[15][16][17]}

矩阵相乘在 3D 变换中是被频繁用到的一种计算,但在矩阵相乘过程中用到了大量的矩阵运算,而运算单元对于乘法的效率是比较低的,远低于加法运算,所以,如果我们能找到一种快捷的方法来实现矩阵相乘,将能大大提高系统的效率。

4.5.1 矩阵乘法的算法分析

任意两个矩阵的乘积定义如下:

$$c_{ik} = a_{ij}b_{jk}.$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

$$c_{np} = a_{n1}b_{1p} + a_{n2}b_{2p} + \dots a_{nm}b_{mp}$$

因而,为了矩阵乘法能够被执行,矩阵的维数必须满足

$$(n \times m)(m \times p) = (n \times p).$$

以 4x4 的矩阵乘为例,完成结果的输出需要进行 64 次乘法运算,48 次加法运算。运行效率可想而知。

为提高矩阵乘法的运算速度,可采用并行处理结构及其对应的并行算法。脉动阵列简单重复的 PE 构成和 PE 局部连接性质使它特别适合于具有局部相关关系的线性递推运算的实现,矩阵乘法就是其中一种。在执行矩阵相乘时,数据间相关关系具有局部性、一致性,映射到阵列结构上时,仅需局部、规则的 PE 间的通讯,特别适合于用脉动阵列实现。

4.5.2 脉动阵列结构

4.5.2.1 脉动阵列介绍

(1) 脉动阵列定义^[18]:

从1978年开始,美国卡内基—梅隆大学(CMU)的H. T. Kung等人在研究算法计

算与专用芯片的体系结构的关系时, 将计算任务按照通信与计算的比重不同分为受计算限制(computation bound)的和受输入、输出限制(I/O bound)的两大类。脉动阵列主要针对受计算限制的问题而提出的。

脉动阵列由一组简单的、重复的PE构成, 每个PE能够执行固定的、简单的操作。每个PE只与相邻的PE有规则地连接。除了极少数边界上的PE外, 所有内部的PE的内部构造都是一样的。进入、输出阵列的数据必须在边界上的PE进行输入、输出。脉动阵列中通过有规则的、局部的PE间相互连接得到, 这种特性使之特别适合用VLSI技术来实现。

关于脉动阵列的定义有很多, S. Y. Kung给出的定义为:

脉动阵列使一个具有如下特性的计算机网络:

- (1) 同步性: 通过网络的数据是有节奏性地被计算(由一个全局时钟定时)和传递。
- (2) 模块化与规则化: 阵列是由带有均匀互连拓扑的模块化处理单元组成, 计算网络可以无限扩展。
- (3) 空间局部性与时间局部性: 阵列内部的变量以局部通信的形式互连接成某种结构, 即空间局部性。信号从一个节点传送到下一个节点中, 至少具有一个单位时间的延迟。
- (4) 加速比: 阵列具有线性速率的流水能力, 即处理器速率达到 $O(M)$ 的加速比, 其中 M 是处理器单元个数。

(2) 脉动阵列基本结构:

脉动阵列的原理^[19]如图4.7(b)所示。数据从存储器中有规则地压入阵列, 经过许多连成流水线的PE, 沿途得到一系列连续的处理。

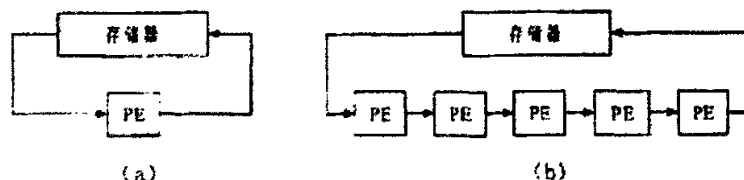


图4.7 (a)传统的计算机结构 (b)脉动阵列结构

脉动阵列的有效性可以通过以下的例子有效地说明:

图4.7中, 每个PE以20ns的时钟周期工作, 在传统的存储器处理机结构图4.7(a)中, 系统最高的运算性能为50万次/秒。在同样的时钟效率下, 脉

动阵列的4.7(b)可以接近于250万秒。性能的提高主要是增加了流水线处理的结果。一旦数据从存储器中取出并由边界上的PE进入阵列,它就按流水线方向沿着阵列从一个PE传送到相邻的PE,并在所经的PE得到有效、充分的应用。

脉动阵列实际上是某些算法的硬件直接实现。它把算法中蕴含的操作并行性用具有同样逻辑功能的PE通过简单、规则的通信局部互连起来的阵列来实现。这样的阵列除了连接的几何形状有所变化外,在阵列内,无论数据输入数据流或者结果数据流的速度与方向都有所变化。这是传统的流水线结构所没有的。在功能上,一个这样的阵列相当于软件中含有循环语句的过程。由于算法被阵列固化了,其中不含有软件成份,整个阵列作为一个功能单元。脉动阵列可以解决一大类基本的计算问题,其中包括绝大部分的矩阵运算、数字信号与图像处理的相关及其许多数值运算。PE的内部操作与计算可根据算法的不同而异。

(3) 脉动阵列的适用性:

脉动阵列实质上是一种线性时间阵列。数据在阵列内相邻的PE之间流动,时时、处处都用相同的时间单位,即它的数据流动的频率是稳定的。因此,任何算法只要能够在代数空间找到一个线性表示,就至少存在一种脉动阵列结构来实现它。

由于在脉动阵列中,数据从存储器中有规则地压入阵列,经过许多连成流水线的PE,沿途得到一系列连续的处理,所以它的数据运动是局部的,即在这个PE中的运算导致的数据改变不会影响后续的PE运算。所以对于数据运动具有全局性的算法,很难找到一种合适的脉动阵列结构。

以傅里叶变换为例,离散傅立叶变换(DFT)的表示是线性的,它的数据运动具有局部性,所以很容易找到脉动阵列的结构来实现。但是在快速傅立叶变换中(FFT),它的数据运动具有全局性,则找不到一种脉动阵列结构来实现。相关研究表明,用脉动阵列实现的DEF的运算速度接近FFT的运算速度,这从另一个方面表明了脉动结构的有效性。而且绝大部分的运算无法找到一种类似于FFT一样性能得到巨大提高的快速算法,在VLSI计算中,脉动阵列结构发挥了巨大

的作用。

脉动阵列免除了形成数据流所需的控制开销。阵列内PE间的局部连接方式,使得阵列中的负载均匀、连接极短,最大限度地减少了系统内部的通信延时,提高了PE的利用率,使整个阵列的系统性能得到充分的发挥。因此,这种结构得到了普遍的重视。国外设计完成的脉动阵列处理机有:美国CMU的可编程脉动阵列机Warp, Saxpy公司设计的Matrix. 1, ES L公司设计的脉动阵列机。其中,CMU的协印由多个高性能的处理机构成,整个阵列机的平均处理能力达到1亿次/秒,它可应用于数字图像处理与识别、数字信号处理、人工智能等方面。

(4) 脉动阵列的设计:

脉动阵列的设计主要是如何设计一个阵列机构用于完全表达算法内部所固有的并行性和流水性。相关图用图形的形式反映了数据的流水性和相关性,并允许对其结构作一定修改。所以脉动阵列的设计的主要方法为如何通过投影和变换把DG转化到脉动阵列结构。

脉动阵列的结构可以通过相关图直接导出,许多文献论述了从相关图导出脉动阵列的映射方法^{[20][21][22]}。对于比较复杂的算法,常用的方法是通过相关图得到信号流图(SFG),再从信号流图映射到脉动阵列结构。

(5) DG到SFG映射:

为了得到局部递归算法的有效阵列结构,一种直接的设计方法是在DG的每一个节点安排一个合理的PE。但这样会使PE的利用率比较低,因为每一个PE只在计算时间的少部分期间处于激活状态。为了提高PE使用率,需要通过变换将DG的节点映射到较少量的PE上,一个比较好的方法是转化为信号流图。

信号流图(SFG)是由处理节点、通信边和延迟环组成的。一个SFG描述包括了功能描述和结构描述,功能描述定义了节点内的行为,结构描述定义了节点间的互联方式。与DG相比,SFG具有如下特征:

(1) SFG比DG简洁;

(2) SFG更接近于实际的硬件设计,它同时也决定了将要获得的阵列类

型:

(3) 当DG中不存在环时, SFG可能含有环, 但每个环上至少具有一个单位D延迟。

从DG映射到SFG主要解决两个基本问题是:

(1) 如何分配各个处理器的运算使得处理器之间的数据通信最少;

(2) 如何安排多个运算到一个处理器使整个计算时间最少。

因此, DG到阵列处理器的映射主要分成两步:处理器映射和任务调度。处理器映射主要完成沿着投影矢量 d 的所有DG节点分配到一个共同的PE上, 投影的结果得到一个SFG表示。调度分配是用来指定所有PE上各运算的顺序, 一个调度函数代表了从DG中的N维下标空间到一维调度空间的一个映射。为了导出一个规则的脉动阵列, 一般用线性投影进行处理器的分配, 使用线性调度进行调度的分配。

对于给定一个DG和一个投影矢量 d , 在SFG投影中最可能使用的调度有:

(1) 缺省调度:超平面正交于投影方向 d , 或超平面的法线。平行于投影方向 d ;

(2) 循环调度:调度矢量 s 平行于DG下标空间的一个轴, 通常是使用相应于循环次数的那个轴;

(3) 脉动调度:是一个用于脉动阵列的调度。在最后的SFG的每一条边上至少应存在一个单位的延时;

(4) 优化调度:该调度有数据相关性、处理器可用性以及设计准则所确定。

调度并不是任意的, 因为在一个算法的计算中, 内在的存在一个由DG指定的次序, 比如, 从节点 x 到节点 Y 存在一条有向路径, 则由节点 Y 表示的计算必须在节点 x 所表示的计算完成以后才能执行。

(6) SFG到阵列结构的映射:

SFG在结构上已经非常接近脉冲阵列了, 大多数SFG的主要缺陷是他们不是以时间局部化的形式给出。即需要重定时序使得SFG变换为一个等效的时间局部化形式。

所以对于脉动阵列和信号流图的关系可以写成:

脉动阵列=SFG阵列+流水重定时序

重定时序是要将SFG变换为一个等效的时间局部化形式。重定时序的方法在计算机网络中得到了广泛地研究,其中使用简单、证明比较直接的是割集重定时序准则。

割集重定序方法的目标是要将一个SFG转换为一个时间局部的等效形式,从而使其各模块化分区之间的所有边上至少有一个延迟单元。

SFG的割集是将SFG分割成两部分边的最小集合,割集重定时序方法是基于以下两个准则:

(1) 时间标度:所有延迟 D 都可以用 $D \rightarrow \vartheta D'$ 来进行标度, ϑ 是一个正整数,称为SFG的流水周期。相应的,输入输出速率也必须由 ϑ 因子进行标度。

(2) 延迟转移:给定SFG的任意割集,它将SFG分成两个部分,根据割集

可以把所有边的指向分为向内的边和向外的边两种。延迟准则允许在所有向内的边上加上 k 个延迟时间,在所有向外的边上加上 $k(D')$ 的超前时间单元。

可以证明所有的可计算的SFG都可以根据割集重定时序准则而变换为时间局部形式,即空间局部的规则SFG阵列总可以被脉动化。

利用下列主要基于割集重定时序准则的脉动化方法,可以容易地将一个规则的SFG阵列脉动化为一个脉动阵列:

(1) 基本运算模块的选择:基本模块的粒度越细,脉动阵列的速度效应越高;

(2) 运用重定时序准则:为了维持SFG的模块化结构,应对整个网络均匀地进行割集重定时序;

(3) 延迟和运算模块地结合:为了将一个SFG转化为脉动形式,只需要在每个运算模块中引入一个适当的延迟,然后该延迟就可以与模块运算相结合形成一个基本的脉动单元。

(7) 矩阵乘法脉动阵列结构:

对于简单矩阵乘法SFG运用准则2,所得的脉动化SFG的每一条边上都具有一个延迟,因此它是一个局部化的网络。对来自B的每一个列和来自A的每一个行的输入,必须在它们到达阵列之前采用一定数量的延迟对其进行调整。可以看

出: B 的第一列不需要调整, 第二列需要加一个延迟, 第三列加两个延迟, 如此等等, B 的矩阵将被倾斜输入: 对 A 也做类似的处理, 脉动阵列结构如图 4.8 所示。

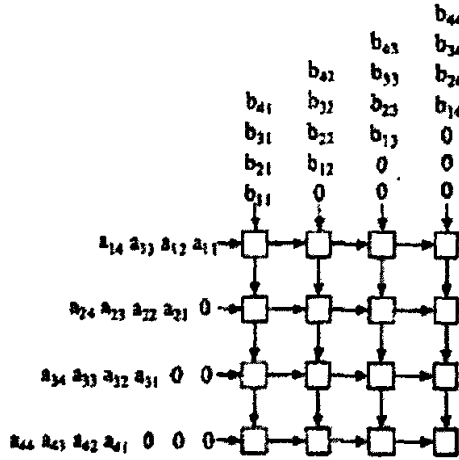


图 4.8 常用矩阵乘法脉动阵列结构

(8) 矩阵乘法脉动阵列性能分析^[19]:

矩阵运算脉动阵列性能也从加速比、效率和速效积三方面来衡量。

(1) 加速比

对于 n 阶方阵乘法运算, 当采用串行算法在单处理器运行时, 需要的计算时间是 n^3 次加法和 n^3 次乘法。当采用 n^2 个 PE 以图所示脉动阵列完成矩阵乘法, 需要时间为: $2n+1$ 次乘法和加法。由于相对于乘法来说, 加法一般运算速度远远高于乘法, 所以忽略加法时间, 以乘法来估计加速比。加速比的意义在于表示并行算法的有效性。

$$S_p(n) = T_1(n) / T_p(n) = \frac{n^3}{2n-1} \approx 0.5n^2$$

它的速度提高了 $0.5n^2$ 倍。对于大型矩阵来说, 这个加速比还是很可观的。假设对于图象处理中的 $n=10$ 的矩阵, 通过 100 个阵列处理器实现脉动阵列对其进行计算和通过一个处理器进行计算相比, 速度提高了 50 倍。

(2) 效率

$$E_p(n) = S_p(n) / P(n) \approx 0.5$$

这表示用 2 个 PE 才取得某种速度的提高, 可以得出每个处理器的处理能力发挥的程度为 0.5。

4.5.2.2 脉动阵列结构矩阵乘法器的实现

我们并不直接采用上小节介绍的脉动阵列结构, 而是做了点调整。文献[23]也做了介绍。

在矩阵相乘的二维阵列中, 矩阵 A、B、C 元素的流动方式决定着阵列的连

接方式与处理器的个数和计算时间。如果将某一矩阵的元素存入各 PE 中，而另两个矩阵的元素在阵列中沿不同方向移动，则在阵列中就有 2 个不同的数据流动方向，每一 PE 就有 2 对输入/输出口，这就构成了正方形表示的内积器，这样的阵列称为正方形阵列，简称方阵。本文提出如图 4.9 结构的二维方阵结构的矩阵乘法器，图中每个小方块代表一个内积器 PE。

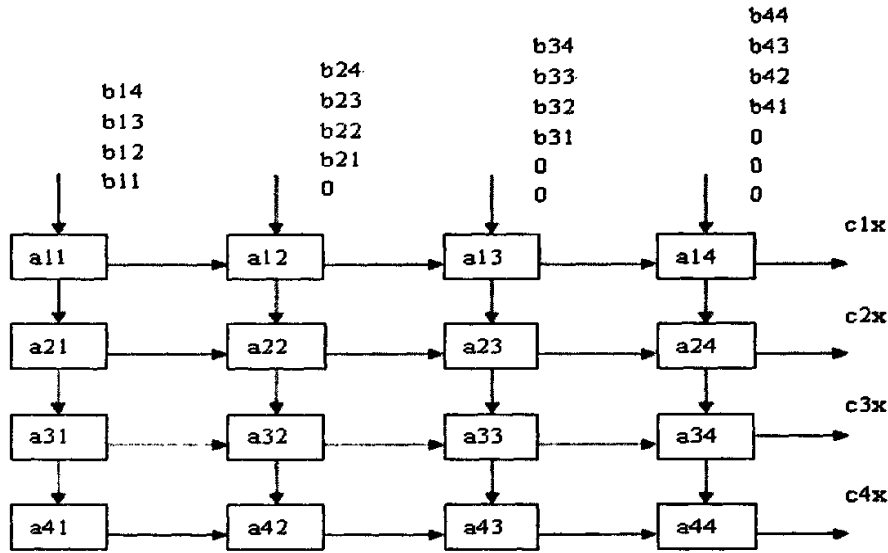


图 4.9 本文的脉动阵列结构矩阵乘法器结构

1). PE 的端口如图4.10所示,clk 和clk1 是不同频率时钟。PE 内部有三个寄存器RA、RB 和RC。每个寄存器有两个接头,分别用于输入和输出。在clk 的上升沿和load.en的有效信号作用下,将矩阵A 的元素从loaddatain 装入到PE 的RA 中存放;并在clk的下降沿,将存放在RB 中的矩阵B 的元素与A 的元素进行有符号数相乘,并结合从mult.advance 口进来的RC 完成操作 $RC \leftarrow RC + RA \times RB$ 。然后将RB 中的输入值和RC 中的新值分别送到输出线matrixdataout 和q 上(这些输出线与其它处理单元的输入线相连)。

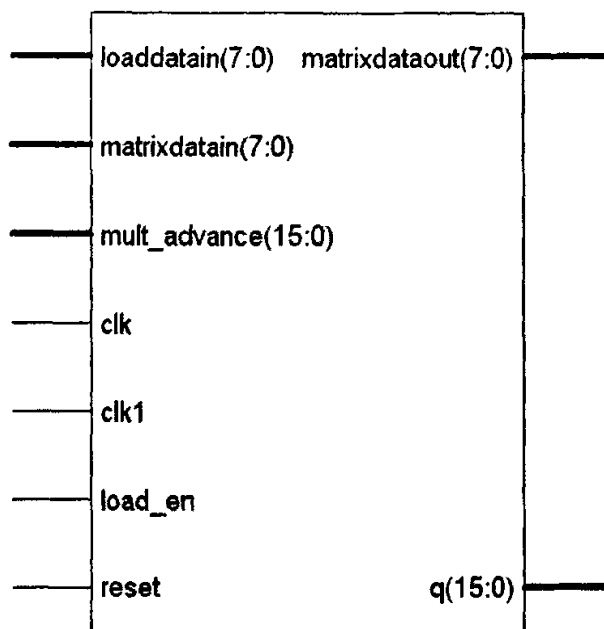


图4.10 PE输入/输出端口

图4.11给出了PE单元的RTL时序仿真图:

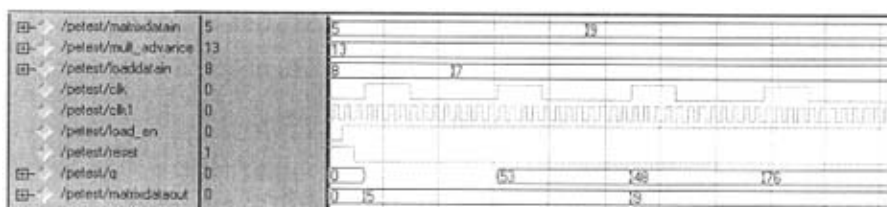


图4.11 PE单元的RTL时序仿真图

2). PE间的连接结构

从横向连接看, 将PE的q口连到下一个PE的mult. advance口; 从纵向连接看, 将PE的matrixdataout口连到下一个PE的matrixdatain口。内积器的clk、clk1、reset和load.en口为全局信号。

在clk的上升沿和load.en的作用下, 将矩阵A的元素同时加载到此16个PE处理器中。下一个clk开始输入矩阵B元素, 并在clk下降沿作用下, 进行内积逐步运算。因为矩阵运算是元素乘积的累加和, 所以B矩阵第i行元素的输入要比第i.1行元素延迟一个clk。

通过元件例化语句将16个PE单元进行连接:

```

u1:PE port map(b1x, "0000000000000000", load11, clock, clock1, load, reset, q1, d1);
u2:PE port map(b2x, q1, load12, clock, clock1, load, reset, q2, d2);
u3:PE port map(b3x, q2, load13, clock, clock1, load, reset, q3, d3);
u4:PE port map(b4x, q3, load14, clock, clock1, load, reset, c1x, d4);
u5:PE port map(d1, "0000000000000000", load21, clock, clock1, load, reset, q4, d5);
u6:PE port map(d2, q4, load22, clock, clock1, load, reset, q5, d6);
u7:PE port map(d3, q5, load23, clock, clock1, load, reset, q6, d7);
u8:PE port map(d4, q6, load24, clock, clock1, load, reset, c2x, d8);
u9: PE port map(d5, "0000000000000000", load31, clock, clock1, load, reset, q7, d9);
u10:PE port map(d6, q7, load32, clock, clock1, load, reset, q8, d10);
u11:PE port map(d7, q8, load33, clock, clock1, load, reset, q9, d11);
u12:PE port map(d8, q9, load34, clock, clock1, load, reset, c3x, d12);
u13:PE port map(d9, "0000000000000000", load41, clock, clock1, load, reset, q10);
u14:PE port map(d10, q10, load42, clock, clock1, load, reset, q11);
u15:PE port map(d11, q11, load43, clock, clock1, load, reset, q12);
u16:PE port map(d12, q12, load44, clock, clock1, load, reset, c4x);

```

图4. 12是采用二维脉动结构实现矩阵乘法的仿真结果, 其中, b1x~b4x是矩阵B第1行元素到第4行元素的输入口; load_en是矩阵A 元素的加载信号; q1~q4是积矩阵元素第1行到第4行的输出。

图4. 12中描述的是如下矩阵相乘的结果:

$$AB = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 5 & 1 \\ 3 & 6 & 9 & 0 \\ 5 & 5 & 6 & 6 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 3 & 0 & 0 & 1 \\ 4 & 0 & 0 & 0 \end{bmatrix}$$

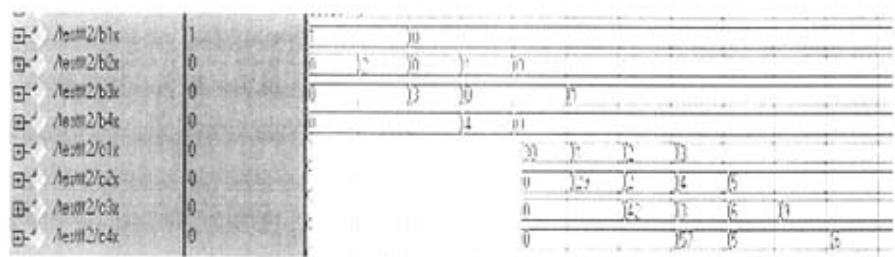


图4.12 脉动阵列乘法器仿真结果

由仿真结果可知，只要经过第一次运算循环(4个clk)后,每个clk 周期都可输出一个积矩阵元素，实现了流水线意义上的矩阵乘法。如果我们将N设为矩阵的维数，那么整个运算能在 $(2N-1)$ 个周期内完成，并且需要 N^2 个PE单元。

4.6 综合分析

本设计中所有模块，包括矩阵构造单元、三角函数查找表、矩阵乘法器和投影模块均使用VHDL语言完成了RTL级的编码。实验运用Xilinx ISE 7.1i软件在Xilinx FPGA上综合并仿真，综合工具为XST，仿真工具为使用Mentor公司的Modelsim6.0对各个功能模块进行了RTL功能仿真。图4.3和图4.10给出了两个关键模块矩阵构造单元和矩阵乘法器的RTL时序仿真波形。

同时，我们用Xilinx公司的ISE7.0对整个模块进行了综合，选用器件为：v3200efg1156.6，所耗资源如下表：

表4.1 耗费资源列表

Number of Slices	438 out of 32448	1%
Number of Slice Flip Flops	770 out of 64896	1%
Number of 4 input LUTs	655 out of 64896	1%
Number of bonded IOBs	176 out of 808	21%
Number of GCLKs	4 out of 4	100%

从表4.1可以看出本设计资源消耗并不大，而且在很多方面可以进行优化设计。

4.7 本章小结

本章主要讨论几何变换部分的实现问题,首先依据实时性几何变换的特别,提出了一种整体解决方案,随后对方案进行解剖,详细分析各个组成模块的实现。

主要介绍了矩阵构造模块,三角函数查找表和脉动阵列结构的矩阵乘法器的实现,其中穿插讲解了脉动阵列的工作原理及如何产生需要的脉动阵列。

第5章 总结与展望

本文探讨了计算机图形学中几何转换阶段的 FPGA 实现技术,该部分是用 FPGA 实现 GPU 整体功能的尝试,希望借此推广到整个 GPU 的实现。本文首先概述了 3D 图形学技术的发展状况、3D 绘图硬件的设计目标,并对 3D 绘图管线进行了较为详细的描述,接着介绍了 IEEE 浮点数表示方法,对几何变换中要用到的单精度浮点数加法器和乘法器的实现方法进行了说明。最后提出一种几何变换实现的解决方案,对各个功能模块的实现进行了详细分析。

本文主要工作总结如下:

1、提出一种几何转换子系统的硬件实现方案。将整个实现系统按功能划分成几个主要模块,随后重点介绍了几个主要模块的实现。该方案能对基本的几何变换如:平移、缩放、旋转和投影进行操作。

2、设计实现了单精度浮点数加法器和乘法器。图形学变换中需要进行大量的浮点数运算,最基本的运算是浮点数加法器和乘法器。这两个模块的实现是实现整个几何转换系统所必须的。仿真结果表明功能正确,满足基本的运算需要。

3、在论述矩阵运算并行算法的基础上,提出了基于二维正方形心动阵列结构的矩阵乘法器,并研究了二维方阵结构的矩阵乘法器的FPGA硬件实现方法。实验结果表明采用二维正方形心动阵列实现的矩阵乘法器,具有高度并行性和流水线性特点,能在保证运算速度的同时,节约硬件资源。

由于本研究还处于起步阶段,所以存在着很多不足,具体表现在:

(1) 脉动阵列矩阵乘法器的研究实现还很简单,没有进行优化和进一步的研究。

(2) 查找表结构过于简单,需要进行优化,在提高速度、精度的同时将ROM表的数据量减少。

(3) 只进行了简单的功能仿真,还未到具体的FPGA板级调试。

FPGA实现GPU功能,是一个比较新但具有较大前景的研究方向,它的研究成果对于辅助GPU,实现图形学复杂算法的实时运算有着重要的意义。

参考文献

- [1] http://www.lnnu.edu.cn/xdjy/jx/tuxing/Chapter1/CG_Txt_1_006.htm
- [2] James.D. Foley. Computer Graphics: Principle and Practice , Pearson Education
- [3] 计算机图形学的算法基础 (原书第2版), David F. Rogers著, 石教英, 彭群生等译, 北京, 机械工业出版社, 2005, 8
- [4] 计算机图形学基础教程, 孙家广, 胡事明编著, 北京, 清华大学出版社, 2005
- [5] VHDL实用教程, 潘松, 王国栋编著, 成都, 电子科技大学出版社, 2001, 7
- [6] VHDL数字电路设计教程, 佩德罗尼 (Pedroni, V. A.) 著, 乔庐峰 等译, 电子工业出版社, 2005, 9
- [7] 数字信号处理的FPGA实现 (第2版), 迈耶. 贝斯 (Meyeer. Baese, U.) 著, 刘凌 译, 清华大学出版社, 2006, 6
- [8] <http://www.fpga.com.cn/>
- [9] 简弘伦, 张凯锋. 《精通Verilog HDL: IC设计核心技术实例详解》, 电子工业出版社, 2005
- [10] 实时计算机图形学 (第2版), Tomas Akenine.Möller, Eric Haines著, 曹建涛译, 北京大学出版社, 北京, 2004, 7;
- [11] Computer Graphics Principles and Practice Second Edition in C, James D.等著, 机械工业出版社, 北京, 2002, 6;
- [12] 浮点运算CORDIC之实现与其再3D图学之应用, 王博立, 国立中山大学硕士论文
- [13] IEEE standards Board and ANSI. IEEE Standard for Binary Floating-Point Arithmetic, 1985. IEEE Std 754.1985.
- [14] George M. Papadourakis , George N. Bebis. The Design of a Systolic Architecture to Implement Graphic Transformations. 1991 IEEE
- [15] A. Amira, F. Bensasali. AN FPGA BASED PARAMETERISABLE SYSTEM FOR MATRIX PRODUCT IMPLEMENTATION, 2002 IEEE
- [16] Jong. Chuang Tsay and Pen. Yuang Chang . Design of Efficient Regular Arrays for Matrix Multiplication by Two-Step Regularization , IEEE Transactions on Parallel and Distributed Systems Vol. 6, No. 2, Feb 1995.
- [17] Sesidhar. Pvrakhya. Real-time matrix multiplication in FPGA. 2005
- [18] S. Y. Kung. VLSI Array Processors, Prentice Hall, 1988, Englewood Cliffs, New Jersey.

-
- [19] 陈国良 陈峻编著 VLSI计算理论与并行算法. 中国科学技术大学出版社, 1991. 合肥.
- [20] D. I. Moldovan. On the design of algorithm for VLSI systolic arrays. Processing of the IEEE, Vo 1. 71, January 1983.
- [21] P. R. Cappello etc. VLSI signal Processing. IEEE PRESS, N. Y, 1984.
- [22] S. k. Rao, Regular Iterative Algorithms and Their Implementations On Processor Array. PHD thesis, Stanford University, Stanford, California, 1985.
- [23] 吴淑泉, 王前, 谢运祥. 适于消谐模型求解的矩阵乘法器设计与实现. 华南理工大学学报 (自然科学版), 2003年8月, Vol. 31 No. 8
- [24] F. Bensaali, A. Amira, I. S. Uzun, A. Ahmedsaid. AN FPGA IMPLEMENTATION OF 3D AFFINE TRANSFORMATIONS, 2003 IEEE
- [25] Keshab K. Parhi, VLSI Digital Signal Processing Systems Design and Implementation. Wiley, 1999
- [26] Daniel Mc Keon, Synthesizable VHDL Model of 3D Graphics Transformations, Final Year Project 2005
- [27] F. Bensaali, A. Amira, I. S. Uzun, A. Ahmedsaid, AN FPGA IMPLEMENTATION OF 3D AFFINE TRANSFORMATIONS. IEEE, 2003
- [28] Niklas Knutsson, An FPGA based 3D Graphics System, 2004 Master's thesis
- [29] Johnson, K. T. . Hurson, A. R. . Shirazi, B. : General Purpose Systolic Arrays. IEEE Computer, November 1993, pp. 20. 31.
- [30] Chris H. Dick, FPGA BASED SYSTOLIC ARRAY ARCHITECTURES FOR COMPUTING THE DISCRETE FOURIER TRANSFORM. IEEE, 1996
- [31] Pavel Zencik, Hardware Acceleration of Graphics and Imaging Algorithms Using FPGA
- [32] VLSI数字信号处理系统设计与实现, 陈弘毅, 白国强, 吴行军等译, 机械工业出版社, 北京, 2004

附录A 攻读硕士学位期间发表的论文及参加科研情况

1. 孙大成，郭立，“几何变换单元的设计及其FPGA实现”，《电子测量技术》，已录用
2. 参与一款芯片的设计工作

附录B 相关VHDL程序

B. 1 FLOATING POINT ADDER

--IEEE Libraries--

Library IEEE;

Use IEEE.std_logic_1164.all;

Use IEEE.std_logic_arith.all;

Use IEEE.std_logic_unsigned.all;

Use IEEE.std_logic_signed.all;

Library work;

Use work.float_pkg.all;

Entity fpaddnormal is

Generic

```
(
    Exp_bits:integer:=0;
    Man_bit:integer:=0
);
```

Port

```
(
    O1: in std_logic_vector(man_bits+exp_bits downto 0);
    O2: in std_logic_vector(man_bits+exp_bits downto 0);
    Ready:in std_logic;
    Exception_in:in std_logic;
    Clk:in std_logic;
    Result:out std_logic:= ' 0' ;
    Done:out std_logic:= ' 0'
);
```

End fpaddnormal;

Architecture fpaddnormal_arch of fpaddnormal is

Component fp_add is

```

Generic(
    Exp_bits:integer:=0;
    Man_bit:integer:=0
);

Port(
    OP1:std_logic_vector(man_bits+exp_bits downto 0);
    OP2:std_logic_vector(man_bits+exp_bits downto 0);
    Ready:in std_logic;
    Exception_in:in std_logic;
    Clk:in std_logic;
    Result:out std_logic_vector (man_bits+exp_bits+1 downto
0) :=(others=>' 0' );
    Exception_out:out std_logic:=' 0' ;
    Done:out std_logic:=' 0'
);

End component;

Component rnd_norm is
Generic
    (
        Exp_bits:integer:=0;
        Man_bits_in:integer:=0;
        Man_bits_out:integer:=0
    );
Port
    (
        In1:in std_logic_vector((exp_bits+man_bits_in) downto 0);
        Ready:in std_logic;
        Clk:in std_logic;
        Round:in std_logic;

```

```

Exception_in:in std_logic;

Out1:out std_logic_vector((exp_bits+man_bits_in) downto 0):=
(others=>'0');

Done:out std_logic:='0';

Exception_out:out std_logic:='0'

);

End componenet;

Component denorm is
Generic(
    Exp_bits:integer:=0;
    Man_bits:integer:=0
);
Port(
    In1:in std_logic_vector(exp_bits+man_bits downto 0);
    Ready:in std_logic;
    Exception_in:in std_logic;
    Out1:out std_logic_vector(exp_bits+man_bits+1 downto 0):=
(others=>'0');
    Done:out std_logic:='0';
    Exception_out:out std_logic:='0'
);

End component;

Signal done1,done2,dong3:std_logic;
Signal result1:std_logic_vector(exp_bits+man_bits+2 downto 0);
Signal OP1,OP2:std_logic_vector(exp_bits+man_bits+1 downto 0);
Signal e:std_logic;
Signal ein1:std_logic;
Signal ein2:std_logic;

```

```

Begin
..FLOATING.POINT.ADDITION.MODULE
Addition:fp_add
Generic map(
    Exp_bits=>exp_bits,
    Man_bits=>man_bits+1
)
Port map ( OP1,OP2,READY,EXCEPTION_IN,CLK,RESULT1,E,DONE3);

..ROUNDING.AND.NORMALIZING.OUTPUT
Output_round:rnd_norm
Generic map(
    Exp_bits=>exp_bits,
    Man_bits_in=>man_bits+2,
    Man_bits_out=>man_bits
)
Port map( result1,ready,clk,'1',e' result,done,exception_out);
..2.inputs.denormalizing
Input1_norm:dnorm
Generic map(
    Exp_bits=>exp_bits,
    Man_bits=>man_bits
)
Port map(OP1,READY,EXCEPTION_IN,OP1,DONE1,ein1);
Input2_norm:dnorm
Generic map(
    Exp_bits=>exp_bits,
    Man_bits=>man_bits
)

```

```
Port map(O2, READY, EXCEPTION_IN, OP2, DONE2, ein2);
```

```
End fpaddnormal_arch;
```

B.2 FLOATING POINT MULTIPLIER

```
..IEEE Libraries
```

```
Library IEEE;
```

```
Use ieee.std_logic_1164.all;
```

```
Use ieee.std_logic_arith.all;
```

```
Use ieee.std_logic_unsigned.all;
```

```
Use ieee.std_logic_signed.all;
```

```
Entity norma is
```

```
Generic(
```

```
    Exp_bits:integer:=0;
```

```
    Man_bits:integer:=0
```

```
);
```

```
Port(
```

```
    IP1:in std_logic_vector(exp_bits+man_bits downto 0);
```

```
    IP2:in std_logic_vector(exp_bits+man_bits downto 0);
```

```
    READY:in std_logic;
```

```
    EXCEPTION_IN:std_logic;
```

```
    CLK: std_logic;
```

```
    RESULT:out std_logic_vector(exp_bits+man_bits downto  
0):=(others=>' 0' );
```

```
    EXCEPTION_OUT:out std_logic:=' 0' ;
```

```
    DONE:out std_logic:=' 0'
```

```
);
```

```
End entity;
```


Architecture normaa of norma is

Component fp_mul is

```
Generic(
    Exp_bits:integer:=0;
    Man_bits:integer:=0
);

Port(
    OP1:in std_logic_vector(exp_bits+man_bits downto 0);
    OP2:in std_logic_vector(exp_bits+man_bits downto 0);
    READY:in std_logic;
    EXCEPTION_IN: in std_logic;
    CLK: in std_logic;
    RESULT: out std_logic_vector(exp_bits+(2*man_bits) downto 0);+
        (others=>' 0' );
    EXCEPTION_OUT:out std_logic:=' 0' ;
    DONE :out std_logic:=' 0'
);
```

End component;

Component rnd_norm is

```
Generic(
    Exp_bits: integer:=0;
    Man_bits_in: integer:=0;
    Man_bits_out: integer:=0
);

Port(
    IN1:in std_logic_vector((exp_bits_man_bits_in) downto 0);
    READY: in std_logic;
    CLK: in std_logic;
    ROUND: in std_logic;
```

```

    EXCEPTION_IN: in std_logic;

    OUT1: out std_logic_vector((exp_bits+man_bits_out) downto
0);:=(others=>' 0' );

    DONE:out std_logic:=' 0' ;

    EXCEPTION_OUT:out std_logic:=' 0'

);

End component;

Signal mulout :std_logic_vector(exp_bits+(2*(man_bits+1)) downto 0);

Signal man :std_logic_vector((man_bits*2)+1 downto 0);

Signal input1: std_logic_vector(exp_bits+man_bits+1 downto 0);

Signal input2: std_logic_vector(exp_bits+man_bits+1 downto 0);

Signal done1:std_logic;

Signal e:std_logic;

Begin

..de.normalizing inputs

Input1(exp_bits+man_bits+1 downto man_bits+1)<=IP1(exp_bits+man_bits downto
man_bits);

Input2(exp_bits+man_bits+1 downto man_bits+1)<=IP2(exp_bits+man_bits downto
man_bits);

Input1(man_bits.1 downto 0)<=IP1(man_bits.1 downto 0);

Input2(man_bits.1 downto 0)<=IP2(man_bits.1 downto 0);

Input1(man_bits)<=' 1' ;

Input2(man_bits)<=' 1' ;

..floating.point multiply

Multiply:fp_mul

Generic map(

    Exp_bits=>exp_bits,

    Man_bits=>man_bits+1

)

```

```

Port map(input1, input2, READY, EXCEPTION_IN, CLK, multout, donel);
..output rounding and normalizing
Output_round:rnd_norm
Generic map(
    Exp_bits=>exp_bits,
    Man_bits_in=>(man_bits+1)*2,
    Man_bits_out=>man_bits
)
Port map(multout, donel, CLK, ' 1' , e' RESULT, DONE, EXCEPTION_OUT);
End normaa;

```

B.3 PROCESSING ELEMENT

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PE is
    port( matrixdatain: in std_logic_vector(7 downto 0);
          mult_advance: in std_logic_vector(15 downto 0);
          loaddatain: in std_logic_vector(7 downto 0);
          clk: in std_logic;
          clk1: in std_logic;
          load_en: in std_logic;
          reset: in std_logic;
          q: out std_logic_vector(15 downto 0);
          matrixdataout: out std_logic_vector(7 downto 0));
end PE;

```

```

architecture Behavioral of PE is
component multi8x8
    port( clk:in std_logic;
          start:in std_logic;
          a:in std_logic_vector(7 downto 0);
          b:in std_logic_vector(7 downto 0);
          dout:out std_logic_vector(15 downto 0));
end component;
component adder16b
    port (cin: in std_logic;
          a:in std_logic_vector(15 downto 0);
          b:in std_logic_vector(15 downto 0);
          s:out std_logic_vector(15 downto 0);
          cout: out std_logic);
end component;
component reg8
    port( d: in std_logic_vector(8 downto 1);
          clk: in std_logic;
          load_en: in std_logic;
          clr : in std_logic;
          q: out std_logic_vector(8 downto 1)
          );
end component;
component reg16
    port( d: in std_logic_vector(16 downto 1);
          clk: in std_logic;
          load_en: in std_logic;
          clr: in std_logic;
          q: out std_logic_vector(16 downto 1)
    );
end component;

```

```

    );

end component;

    signal q1,q2: std_logic_vector(7 downto 0);

    signal q3,q4: std_logic_vector(15 downto 0);

begin

    u1:reg8 port map(loaddatain,clk,load_en,reset,q1);

    u2:reg8 port map(matrixdatain,clk,'1',reset,q2);

    u3:multi8x8 port map(clk1,clk,q1,q2,q3);

    u4:adder16b port map(cin=>'0', a=>q3,b=>mult_advance,s=>q4);

    u5:reg16 port map(q4,clk,'1',reset,q);

    matrixdataout<=q2;

end Behavioral;

```

B.4 SYSTOLIC MATRIX MULTIPLIER

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity matrix2d is

    port(clock,clock1,load,reset:in std_logic;

        load11,load12,load13,load14:in std_logic_vector(7 downto 0);

        load21,load22,load23,load24:in std_logic_vector(7 downto 0);

        load31,load32,load33,load34:in std_logic_vector(7 downto 0);

        load41,load42,load43,load44:in std_logic_vector(7 downto 0);

        b1x,b2x,b3x,b4x:in std_logic_vector(7 downto 0);

        c1x,c2x,c3x,c4x:out std_logic_vector(15 downto 0));

end matrix2d;

```

architecture Behavioral of matrix2d is

component PE

```

    port( matrixdatain: in std_logic_vector(7 downto 0);
          mult_advance: in std_logic_vector(15 downto 0);
          loaddatain: in std_logic_vector(7 downto 0);
          clk: in std_logic;      ..start
          clk1: in std_logic;
          load_en: in std_logic;
          reset: in std_logic;
          q: out std_logic_vector(15 downto 0);
          matrixdataout: out std_logic_vector(7 downto 0));

```

end component;

```

signal q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12: std_logic_vector(15 downto 0);

```

```

signal d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12: std_logic_vector(7 downto 0);

```

begin

```

u1:PE port map(b1x, "0000000000000000", load11, clock, clock1, load, reset, q1, d1);
u2:PE port map(b2x, q1, load12, clock, clock1, load, reset, q2, d2);
u3:PE port map(b3x, q2, load13, clock, clock1, load, reset, q3, d3);
u4:PE port map(b4x, q3, load14, clock, clock1, load, reset, clx, d4);
u5:PE port map(d1, "0000000000000000", load21, clock, clock1, load, reset, q4, d5);
u6:PE port map(d2, q4, load22, clock, clock1, load, reset, q5, d6);
u7:PE port map(d3, q5, load23, clock, clock1, load, reset, q6, d7);
u8:PE port map(d4, q6, load24, clock, clock1, load, reset, c2x, d8);
u9: PE port map(d5, "0000000000000000", load31, clock, clock1, load, reset, q7, d9);
u10:PE port map(d6, q7, load32, clock, clock1, load, reset, q8, d10);
u11:PE port map(d7, q8, load33, clock, clock1, load, reset, q9, d11);
u12:PE port map(d8, q9, load34, clock, clock1, load, reset, c3x, d12);
u13:PE port map(d9, "0000000000000000", load41, clock, clock1, load, reset, q10);

```

```
u14:PE port map(d10, q10, load42, clock, clock1, load, reset, q11):
```

```
u15:PE port map(d11, q11, load43, clock, clock1, load, reset, q12):
```

```
u16:PE port map(d12, q12, load44, clock, clock1, load, reset, c4x):
```

```
end Behavioral;
```

致 谢

首先我要衷心感谢我的导师郭立教授。您不仅学识渊博,而且为人正直,治学严谨,平易近人。感谢您三年来给我的悉心教导与关怀,从最初对科研的懵懂无知,到最终硕士论文的完成,都离不开郭老师的耐心指导和严格要求。可能我这三年里的一事无成,让您失望了,谢谢您的宽容。还要感谢集成电路与系统实验室的黄鲁、白雪飞等老师以及班主任朱领娣老师,感谢你们三年来对我的关心与指导。

感谢实验室的杨毅、史鸿声和已经毕业的刘璐、郑军等同学,你们的钻研精神和学习工作态度永远是我学习的榜样,同你们的讨论和交流,扩展了我的知识面,感谢徐华结、刘冬志、周珍艮、魏文龙等同学,感谢大家在我的三年的学习中所给予的热情帮助。

最后向一直关心支持我的家人、朋友和老师们的表示由衷的感谢,是你们的关心与鼓励,我才能顺利地完成这篇论文。

孙大成

2007年5月

作者：[孙大成](#)
学位授予单位：[中国科学技术大学](#)

本文读者也读过(4条)

1. [季健](#) [面向移动设备的像素渲染器设计](#)[学位论文]2009
2. [吴思](#) [基于FPGA的三维图形几何管线的算法及其实现技术研究](#)[学位论文]2008
3. [周珍艮](#) [纹理映射算法研究与FPGA实现](#)[学位论文]2007
4. [刘冬志](#) [真实感图形绘制中明暗效果的FPGA实现](#)[学位论文]2007

引用本文格式：[孙大成](#) [几何变换系统的硬件研究与FPGA实现](#)[学位论文]硕士 2007