

GDI 函数硬件加速器设计与实现

王 汐, 蒋 林, 张 敏

(西安邮电大学 研究生学院, 陕西 西安 710061)

摘 要: 设计实现了加速 Microsoft GDI 中 AlphaBlend、BitBlt、MaskBlt、StretchBlt、TransparentBlt 等函数的硬件结构; 对函数实现中缩放算法的数据相关性进行研究, 提出一种高效的缩放结构, 与 Marvell PXA300 相比性能有明显提升. 并且使用 FPGA 对本结构进行验证, 结果与 Microsoft GDI 一致, 在 SMIC 0.13 μm CMOS 工艺标准单元库下使用 Design Compiler 进行综合, 频率可达 203 MHz.

关键词: 2D 图像; GDI; 图像缩放; FPGA

中图分类号: TP393

文献标识码: A

文章编号: 1000-7180(2014)05-0044-05

Design and Implementation of GDI Accelerator

WANG Xi, JIANG Lin, ZHANG Min

(School of Gradient, Xi'an University of Posts and Telecommunications, Xi'an 710061, China)

Abstract: This paper designs and implements the accelerator circuit for Microsoft GDI AlphaBlend, BitBlt, MaskBlt, StretchBlt, TransparentBlt functions; Put forward a kind of efficient scale structure, improve the handling capacity of the images, after consider Data Dependence of Scaling algorithm. The FPGA output is in accordance with Microsoft GDI. Compared with Marvell PXA300 has obvious improvement in performance. In addition, the operating frequency can reach 203 MHz with SMIC 0.13 μm standard CMOS cell library using Design Compiler.

Key words: Two-Dimensional Image; GDI; Image Zoom; FPGA

1 引言

图形化界面已广泛应用于 PAD、GPS 导航、智能手机等消费电子产品. 目前大部分嵌入式操作系统及相关软件的 GUI(Graphical User Interface)都是基于 2D 技术实现的^[1], 为加快 2D 图形图像的处理速度, 提升用户的操作效率和体验指数, 在嵌入式系统中硬件加速常用的 2D 图像处理方法是很有必要的. 目前国内研究图形加速器中图像功能仅有矩形块拷贝, 无法完成现有消费电子产品中透明混合等常用操作.

本文将要介绍的 2D 图像加速器支持 Microsoft GDI(graphics display interface)中 AlphaBlend、BitBlt、MaskBlt、StretchBlt、TransparentBlt 和剪裁^[2]等常用功能, 可以满足用户对 GUI 的需求, 从而提

升用户体验.

2 2D 相关函数简介及算法选择与优化

图像缩放处理主要采用插值法, 经典的方法有最近邻插值、双线性插值、样条插值、基于边缘的插值、曲面拟合插值和弹性模型插值等^[3-5].

这些算法都存在各自缺点, 在算法复杂度较低时, 锯齿现象不可避免. 综合考虑硬件复杂度和图像缩放质量, 本结构将采用一种自适应 Catmull-Rom 样条算法^[5]实现 GDI 函数中的缩放功能. 算法简述如下:

本 Catmull-Rom 样条算法根据传统 Catmull-Rom 样条算法改进而来. 利用每相邻 4 个节点, 就可构造一段 Catmull-Rom 样条曲线, 且每 4 个节点构成的样条可以拼接成 C^1 连续的曲线. 这样就可利

收稿日期: 2013-07-12; **修回日期:** 2013-08-30

基金项目: 国家自然科学基金面上项目(61272120)

用 Catmull-Rom 样条函数对图像进行插值,得到变化自然的缩放图像. Catmull-Rom 样条函数对节点 (X_i, P_i) 处一阶导数有如下要求:

$$P'_i = \tau(P_{i+1} - P_{i-1}) \quad (1)$$

用于插值时, $\tau = 0.5$ 最佳. 我们定义一组步长为 1 的等距点列,任一相邻的 5 个节点依次为 P_{i-2} 、 P_{i-1} 、 P_i 、 P_{i+1} 和 P_{i+2} ,在 (X_{i-1}, P_{i-1}) 和 (X_i, P_i) 间构造一段 3 次样条曲线,设插值点坐标为 $X \in [X_i, X_{i+1}]$; 其对应的增量为 $t = X_{i+1} - X_i \in [0, 1]$,则该 3 次样条函数如下:

$$P(X) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 = \sum_{k=0}^3 c_k t^k \quad (2)$$

根据已知各端点值可知,式(2)应满足如下条件:

$$\begin{cases} P(0) = c_0 = P_{i-1} \\ P(1) = c_0 + c_1 + c_2 + c_3 = P_i \\ P'(0) = c_1 = \tau(P_i - P_{i-2}) \\ P'(1) = c_1 + 2c_2 + 3c_3 = \tau(P_{i+1} - P_{i-1}) \end{cases} \quad (3)$$

解式(3)可得本 Catmull-Rom 样条函数的系数为

$$\begin{cases} c_0 = P_{i-1} \\ c_1 = (-\tau)P_{i-2} + \tau P_i \\ c_2 = (2\tau)P_{i-2} + (\tau-3)P_{i-1} + (3-2\tau)P_i + (-\tau)P_{i+1} \\ c_3 = (-\tau)P_{i-2} + (2-\tau)P_{i-1} + (\tau-2)P_i + (\tau)P_{i+1} \end{cases} \quad (4)$$

根据式(1)、(2)、(4)得本文所用自适应 Catmull-Rom 样条函数插值矩阵如下:

$$P(t) = [1 \quad t \quad t^2 \quad t^3] \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau-3 & 3-2\tau & -\tau \\ -\tau & 2-\tau & \tau-2 & \tau \end{bmatrix} \quad (5)$$

$t \in [0, 1], \tau = 0.5$

一般,样条插值过程如图 1 所示.以放大 2 倍为例,对一幅图像进行插值时,首先要对它的行(列)方向上每两个像素间插入插值点(A, B, C 点),生成垂直分辨率为原图 2 倍的中间图像;再对中间图像列(行)方向上每两个像素间插入插值点(D, E 点),完成整幅图像的放大.其余倍数的放大与 2 倍放大相似,只是每两个像素间插值点数不同.对于图像的缩小也是先计算行(列)缩小点,再计算列(行)

缩小点,用一个缩小点代替多个实际像素点,以完成整幅图像的缩小.

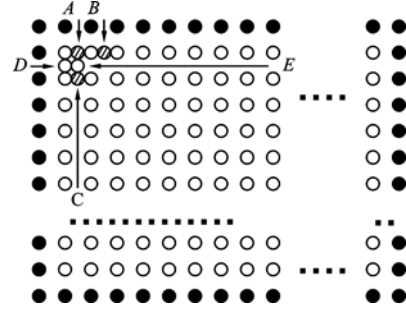


图 1 Catmull-Rom 样条插值过程图

由 Catmull-Rom 样条插值算法知,在独立计算行(列)插值时,生成点与相邻行(列)方向的 4 个像素有数据相关性,而在计算整幅图像时要先计算行(列)插值,再计算列(行)插值.这就使得后计算的列(行)插值点与先计算的行(列)插值点有数据相关性,如图 1 中 E 点.若要以这种方式进行插值计算要存储整幅中间图像,需要较多的存储资源,计算效率不高.根据数据相关性,本文提出了一种新的插值过程(详见 3.2 节伪代码),以减少存储的使用、提高效率.

3 2D 图像加速引擎硬件结构

本结构加速函数的主要可以由:确定图像处理范围;根据处理范围缩放图像;对图像进行逐像素处理,三个主要功能来实现.根据上述特点,本文提出如图 2 所示的硬件流水线结构.

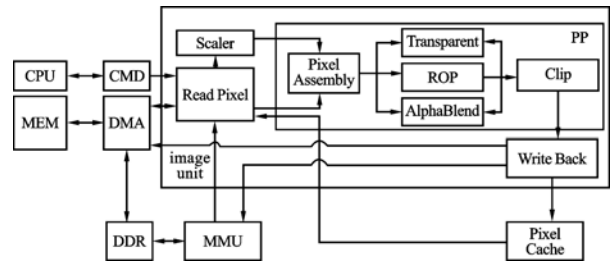


图 2 图像加速器的总体结构图

如图 2 所示,本结构由像素读取模块(Read Pixel)、缩放器(Scaler)、像素装配单元(Pixel Assembly)、像素处理模块(Pixel Processing, PP)、裁剪模块(Clip)、写回模块(Write Back)共 6 个模块组成.待处理的 4 种像素(源图像像素、目标图像像素、掩码图像像素和刷子像素)经过像素读取模块、缩放器(需缩放时像素数据才经过本级)和像素装配单元的预处理后,通过 128 bit 的数据通道发送给像素处理模块、裁剪模块做进一步处理,最终由写回模块将

结果写回 MMU 或像素 Cache, 进而完成图像的全部处理^[6].

3.1 像素读取模块(Read Pixel)

像素读取模块完成命令读取、命令下发、根据图像处理范围读取像素和下发像素的功能. 这个模块是流水线的第一级, 负责从命令处理器(CMD)读入配置命令, 根据读入的命令配置自身的工作方式, 并将其他流水线的配置命令向下级透传, 实现整个图像处理器的串行配置; 此外还负责启动 DMA(Direct Memory Access)从 MMU(Memory Management Unit)或从像素 Cache 读取待处理像素的任务. 本结构使用 128 bit 的 MMU 数据通道, 在读取像素时需要确定 128 bit 数据(32 bpp)中有效起始像素点的位置. 本结构采用一种可配置起始指针位置、输入为 128 位、输出为 32 位的非对称 FIFO 解决此问题.

3.2 缩放器(Scaler)

缩放器使用自适应 Catmull-Rom 样条算法, 实现了从缩小 4 倍到放大 8 倍的图像缩放功能, 综合考虑处理效率、电路面积及 2.1 节中提到的数据相关性, 提出了如图 3 的电路结构实现缩放功能. 图中最左边是一个深度为 4 的 32 位移位寄存器(Shift Register, SR), 它将待插值的 4 个行像素点以行扫描的顺序发送给 Catmull-Rom 样条计算单元(流水线结构)R, 计算单元 R 将行插值结果写入到一个由 4 个 BANK 组成的行像素缓存中, 每一个 BANK 都是一个深度 1 600 宽度 32 的双口 RAM. 一旦满足列缩放的条件, 多路选择器(MUX)将把经过行缩放的数据传给计算单元 C 完成列插值, 最终实现缩放图像以行扫描的顺序逐像素点输出.

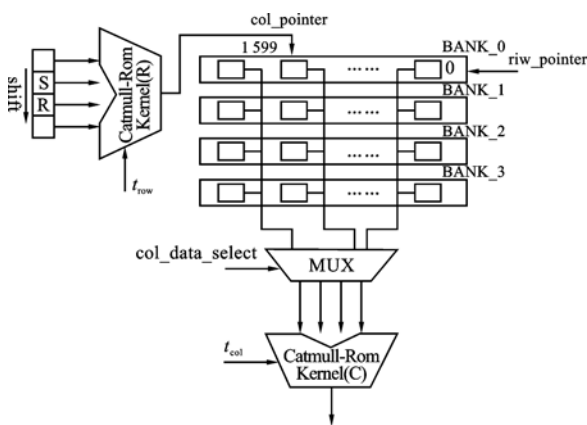


图 3 缩放器结构

在流水线全部填满后, 缩放器可以每时钟周期输出一个像素, 具体工作流程见下述伪代码; shift

(x) 表示 SR 移位输入 x 个像素点, Catmull_Rom(P, t) 表示 Catmull_Rom 样条计算, P 为待处理的 4 个像素点, t 为缩放因子; 所有变量的初始值均为 0:

```
calculate_a_row_pixel() {
    if (trow ≥ 1) { trow = trow - 1; shift(1); }
    * (col_pointer) = Catmull_Rom(shift_register, trow);
    trow = trow + row_factor;
    col_pointer++;
    if (一行缩放完毕) {
        row_pointer++; col_pointer = 0;
        trow = 0; shift(4); }
        else { col_pointer++; }
    }
    row_factor = src_width/dst_width;
    col_factor = src_height/dst_height;
    shift(3);
    while (1) { //填充流水线
        calculate_a_row_pixel();
        if (开始写入第四行像素点) { break; }
    }
    while (源图像缩放未完成) {
        if (4 个 BANK 中有剩余空间接收新的行缩放结果) {
            row_pointer == row_pointer % 4;
            calculate_a_row_pixel(); }
        if (4 个 BANK 中 col_ready 列数据都准备好) {
            col_data = {BANK_0(col_ready), BANK_1(col_read-
y),
            BANK_2(col_ready), BANK_3(col_ready)};
            output_pixel = Catmull_Rom(col_data, tcol);
            col_ready++;
            if (col_ready > dst_width) {
                col_ready = 0;
                tcol = tcol + col_factor; }
            if (有不再使用的行缩放像素点) {
                free_a_oldest_space(); }
            if (tcol ≥ 1) { tcol = 0; }
            return(output_pixel); }
    }
```

为实现上述处理流程, 缩放器使用 3 层状态机控制, 如图 4 所示.

行缩放控制状态机中 INIT 为初始状态, ens 为缩放使能信号. 启动缩放后状态机进入 GET_POINTS_1 状态根据上级的 emp 空满信号, 跳入 GET_POINTS_2 状态, 此状态下根据要读入的像素点数 nnu 将像素值读入移位寄存器 SR. 当已读入像素点数 cnt 与所需像素点数 nnu 相等时, 计算

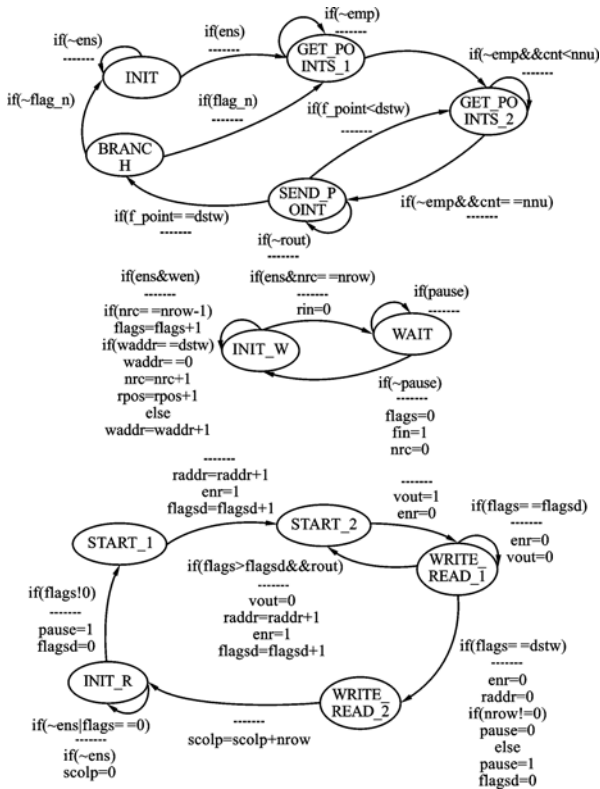


图4 行缩放控制状态机

(上:行缩放控制状态机;中:行插值计数状态机;
下:列计算控制状态机)

插值参数 t 并跳入 SEND_POINT 状态,将准备好的 4 个像素和参数 t 传给行计算单元,由计算单元将行插值结果写入存储中,若已插值点数 f_point 与目标图像宽度 $dstw$ 相同,则跳入 BRANCH 状态;若不同则跳回 GET_POINTS_2 状态,计算下次插值所需的步长值并决定要读入的像素点个数,准备计算本行剩余的插值点。在 BRANCH 状态根据行插值结束标志 $flag_n$ 决定跳入 GET_POINTS_1 状态继续计算新的插值行或返回 INIT 状态完成所有行插值等待处理下次缩放任务。

列插值控制状态机由行插值计数状态机和列计算控制状态机两层组成。行插值计数状态机负责记录行插值点的写入情况,在 INIT_W 状态下当缩放使能信号 ens 有效且写使能 wen 有效时递增行插值点写入地址 $waddr$,若写入地址与目标图像宽度 $dstw$ 相同,行计数器 nrc 自加;当行计数器等于所需行数 $nrow$ 时,列差值状态满足,跳入 WAIT 状态,等待列差值控制状态机计算完毕,即等待 $pause$ 信号为低,再返回 INIT_W 状态写入新的行插值点,其中 rin 信号为允许写入像素信号,高有效。

列计算控制状态机用于从已经完成行插值的具有 4 个 bank 的存储中一次读出 4 个像素点进行列

差值,实现光栅顺序的像素输出。在初始状态 INIT_R 时,若列插值条件满足 $flags! = 0$,跳入 START_1 状态,读出像素给列差值计算单元进行列差值,将结果输出到后续模块;当已读出像素点等于已存行插值点时,需在 WRITE_READ_1 状态等待新的行像素点继续计算列差值。当一行像素输出完后,进入 WRITE_READ_2 状态,同时计算新的列差值参数 t 决定是否读入新行,如果需读入新行 $pause$ 拉低,激活上层状态机,进行新行的插值。

3.3 像素处理模块(PP, Pixel Processing)

像素装配单元与透明选择流水线(Transparent)、光栅操作流水线(ROP, Raster Operation)、Alpha 混合(AlphaBlend)流水线和裁剪单元共同构成了像素处理模块,每 2 时钟可处理一个像素。源图像、目标图像、掩码图像和刷子图像的像素点,经像素装配单元组织后,通过 128 位数据通路传给 3 条并行流水线,完成 Transparent、ROP 或 AlphaBlend 的操作。经处理后像素传入裁剪模块,进行裁剪操作。

3.4 写回模块(Write Back)

写入模块负责将处理完的像素写回 MMU 或像素 Cache。由于本结构可以每 2 个时钟处理一个像素点,而写入模块与 MMU 和像素 Cache 通过握手机制写入像素,每 3~5 个时钟可完成一次写入操作。MMU 和 Pixel Cache 都支持 128 bit 的数据通路,即 4 个像素点(32bpp),只有使用缓冲技术,才可充分发挥出流水线各级的处理能力。本模块使用深度为 4 的乒乓 FIFO 进行写入缓冲,以最大化流水线效率,减少停顿。

4 实现与验证

本结构在 Synopsys VCS 下完成了功能仿真和验证;使用 Design Compiler 在 SMIC0.13 μm CMOS 工艺标准单元库下综合频率可达 203 MHz。选用 Xilinx Virtex XC6VLX550T 芯片进行了 FPGA 验证。

5 结束语

本文在研究 Windows GDI 常用函数 AlphaBlend、BitBlt、MaskBlt、TransparentBlt、StretchBlt 的基础上,根据函数的共性与个性及图像缩放的数据相关性,使用 Top-Down 设计方法,进行了电路设计,电路工作频率可达 200 MHz。经 FPGA 验证电路功能与 Windows GDI 函数一致,处

理性能与 Marvell PXA300 相比有明显提升.

参考文献:

- [1] 袁扬智, 韦明, 曾献军. 2D 图像硬件加速引擎的设计优化[J]. 微电子学与计算机, 2009, 26(4): 241-248.
- [2] Microsoft. GDI Functions (Windows CE 5.0) [EB/OL]. [2013-07-10]. <http://msdn.microsoft.com/zh-cn/library/ms927613.aspx>
- [3] Lehmann T M, Gonner C, Spitzer K. Addendum: B-spline interpolation in medical image processing[J]. Medical Imaging IEEE Transactions, 2001, 20(7): 660-665.
- [4] 康牧. 图像处理中几个关键算法的研究[D]. 西安: 西安电子科技大学, 2009.
- [5] 魏美荣, 蒋林. 一种自适应 Catmull-Rom 样条图像缩放算法的设计实现[J]. 微电子学与计算机, 2011, 28(12): 63-67.

- [6] 蒋永刚. 嵌入式多媒体系统中硬件加速技术的应用[D]. 上海: 上海交通大学, 2009.

作者简介:



王 汐 男, (1989—), 硕士研究生. 研究方向为专用集成电路设计与集成系统.

蒋 林 男, (1970—), 教授, 硕士生导师. 研究方向为专用集成电路设计.

张 敏 女, (1990—), 硕士研究生. 研究方向为网络与多媒体通信.

(上接第 43 页)

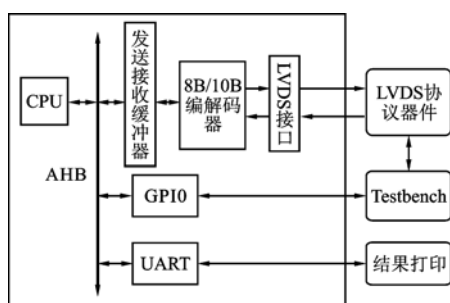


图 6 FPGA 系统验证实现

5 结束语

本文在分析 8B/10B 编码规则的基础上, 提出了一种编解码的方案. 采用 VHDL 语言完成了设计实现, 在 Modelsim 上进行了仿真, 并做了 FPGA 器件验证, 验证结果完全正确. 该方案层次分明、结构简单、编解码速度快, 满足在 LVDS 中应用的设计要求.

参考文献:

- [1] 陈锋. 基于 FPGA 的 8B/10B 编解码设计[J]. 电子设计工程, 2010, 18(5): 152-154.
- [2] 许军, 许西荣. PCI-Express 中 8B/10B 编解码器的设计与实现[J]. 微电子学与计算机, 2006, 23(3): 37-39.
- [3] Lattice Semiconductor Corp. 8b/10b Encoder/Decoder [EB/OL]. [2013-06-20] (2012-02-15). www.lattice-emi.com.
- [4] Widmer A X, Fransz P A. A DC balanced, partitioned block, 8B/10B transmission code[J]. IBM J. RES. DEVELOP, 1983, 27(5): 440-451.
- [5] 李宥谋. 8B/10B 编码器的设计及实现[J]. 电讯技术, 2005(6): 26-32.
- [6] 司朝良. LVDS 接口电路及设计[J]. 今日电子, 2003(1): 17-18.

作者简介:

杨 佩 女, (1985—), 硕士研究生. 研究方向为计算机应用.