第 35 卷 第 6 期 计 算 机 工 程 2009 年 3 月
Vol.35 No.6 Computer Engineering March 2009
·开发研究与设计技术· 文章编号：1000—3428(2009)06—0269—03 文献标识码：A 中图分类号：TP391

基于 Blackfin 处理器的嵌入式 GUI 优化与实现

卢 刚
(浙江财经学院信息学院，杭州 310018)

摘 要：针对 Blackfin 处理器强大的多媒体性能，研究图形用户界面(GUI)的相关优化技术，完成一个基于 uClinux 的嵌入式 GUI 系统。该系统具有体积小、高性能等特点，适用于需要高分辨率显示的电子产品。将该嵌入式 GUI 与 Microwindows 进行比较，相关的性能测试结果证明，该嵌入式 GUI 的实现为高性能嵌入式多媒体产品奠定了基础。

关键词：Blackfin 处理器；图形用户界面；DMA 优化

Optimization and Implementation of Embedded GUI Based on Blackfin Processor

LU Gang
(College of Information, Zhejiang University of Finance and Economics, Hangzhou 310018)

【Abstract】 Aiming at the strong capability of Blackfin processors on multimedia, this paper researches the techniques related to Graphic User Interface(GUI). It implements an embedded GUI system based on uClinux, which achieves smaller size and higher performance. It is suitable for electronic products requiring display of high definition. It compares the performance between the GUI with Microwindows. Results show that the development of embedded applications, especially multimedia products with high performance can benefit greatly from the implementation of the GUI.

【Key words】 Blackfin processor; Graphic User Interface(GUI); DMA optimization

1 概述

目前许多用户、操作系统供应商和开发工具公司都开始使用 Blackfin 处理器。支持 Blackfin 处理器的操作系统主要有 uClinux 和 uC/OS 等，基于这些操作系统的嵌入式 GUI 还不多，主要有 MiniGUI, Microwindows 等移植性较好的 GUI，这些 GUI 适用于分辨率较小的显示设备。

随着液晶显示技术在各种嵌入式系统中的普及，人们对终端显示设备提出了更高显示分辨率需求。目前在 Blackfin 处理器平台上使用较多的嵌入式 GUI 产品是 Microwindows。该系统提供了相对完善的图形功能和一些高级的特性[1]。但其图形引擎存在许多问题，可归纳如下：

(1)无任何硬件加速能力；

(2)图形引擎中存在许多低效算法；

(3)2003 年后该项目的开发开始陷于停滞状态。

本文提出一个基于 Blackfin 处理器的专用嵌入式 GUI，针对 Blackfin 处理器和高分辨率液晶显示设备的特点进行了一系列优化。

2 嵌入式 GUI 优化技术

2.1 总体框架

嵌入式 GUI 的总体框架设计如图 1 所示，其中，硬件平台为支撑本 GUI 系统的嵌入式硬件平台。GUI 系统通过操作系统提供的设备文件来读写 framebuffer 设备。可通过设备驱动程序优化、Blackfin 处理器的二维 DMA 功能、单周期多数据流汇编指令、合理使用片上一级 SRAM 等技术，对硬件抽象层以及窗体模块进行优化，提高 GUI 的运行性能。

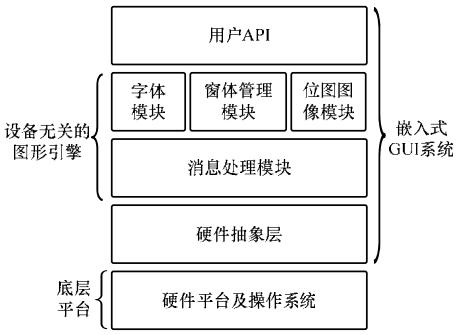

图 1 GUI 系统结构

2.2 设备驱动优化

由于 GUI 需要与外设交互，如果这些设备性能不足、响应较慢都将直接影像 GUI 的操作效果，因此在 GUI 的优化中，应先优化设备驱动。

对于液晶屏设备，如果 CPU 输出的 RGB 信号可以直接用于液晶屏显示，则液晶屏的性能取决于液晶屏、CPU 频率、内存与 PPI 总线速度等硬件设备本身，驱动程序的优化空间较小。

对于触摸屏设备驱动程序，需要根据实际环境来调整响

基金项目：浙江省科技厅重大科技专项和优先主题基金资助项目(2007C13050)

作者简介：卢 刚(1974－)，男，副教授、硕士，主研方向：网络安全，计算机视觉，计算机图形图像处理

收稿日期：2009-01-07 **E-mail：**hz_lugang@163.com

应速度和精度的平衡。另外，触摸屏和 GUI 之间通过设备文件的方式交互，通过 2 个 input_event 结构体分别传递触摸屏的 *X*, *Y* 坐标。由于液晶屏的坐标信息是 12 位的，而 input_event 结构体 1 次可以传递 32 位坐标信息，因此一个 input_event 结构可容纳 2 个坐标信息。在传递坐标时，可将 *X* 坐标值左移 16 位加上 *Y* 坐标值后一次传递，如图 2 所示。

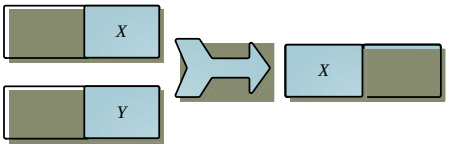

图 2　一次传递 *X*, *Y* 坐标

GUI 在接收到坐标值后，再通过移位操作分别取出 *X*, *Y* 坐标。这样对于每一对 *X*, *Y* 坐标，可以减少 1 次读取坐标的时间。

2.3　片上存储器优化

Blackfin 处理器的片上一级存储器与 CPU 工作频率相同，延迟很小或几乎没有延迟。有效使用一级存储器可大大优化应用程序的运行效率。一级存储器分为一级 Cache 和一级 SRAM，两者在编译内核时可以配置。将单个函数甚至整个应用程序的代码段、数据段放入一级 SRAM 中可以提高执行速度[2]。

例如，对于 GUI 硬件抽象层的像素绘制函数，其代码量不大，可以将其放入一级 SRAM 中执行。步骤如下：

(1)修改编译选项，将应用程序编译为 FDPIC 格式(一种 elf 可执行文件格式)。

(2)在函数申明时，添加“ __attribute__ ((l1_text)) ”修饰符，告诉编译器该函数需要放置在一级 SRAM 中执行。

(3)对于函数中用到的全局变量，在申明时增加“ __attribute__ ((l1_data)); ”选项。只有全局变量、文件内的变量和函数局部静态变量可以放在一级数据 SRAM 中。

(4)如果函数需要动态申请一级 SRAM 空间，需要包含头文件：bfin_sram.h，然后使用 *sram_alloc* 和 *sram_free* 函数动态申请和释放 SRAM 空间。

(5)修改函数所在文件的编译选项，增加“ -fno-jump-tables ”选项。

另外，如果程序把堆栈空间也放置在 SRAM 中，可以使用一级存储器中的 4Kb 暂存器[3]，须使用 Blackfin 提供的 bfin-uclinux-flthdr 命令来编译程序。

2.4　DMA 优化技术

GUI 通过硬件抽象层直接与操作系统交互，对硬件抽象层的优化将直接影像到整个 GUI 各方面性能。可以使用 DMA 优化硬件抽象层的直线、矩形绘制、区域填充等函数。

2.4.1　直线及矩形绘制的 DMA 优化

现有的 GUI 系统在绘制水平或垂直直线时，通常使用简单的循环赋值方法。由于 Blackfin 处理器提供了内存二维 DMA 功能，具体细节可以参照硬件参考手册[4]。在绘制水平或垂直直线时，使用内存二维 DMA 来拷贝像素能大大优化直线绘制速度。水平或垂直直线可看成是高度或宽度为 1 的矩形区域，因此，可用矩形绘制的函数来完成。而且与直线相比，矩形区域是较大的内存块，更能体现 DMA 对内存访问的速度优势。

用二维内存 DMA 绘制矩形区域的方法如图 3 所示。其中，内存源地址存放区域填充颜色的 RGB 值；内存目标地址存放需要绘制的矩形区域。假如像素格式为 24 位色，即 3 个字节表示 1 个像素，则对于内存源，可设置 *Y_MODIFY*=-3，这样就可以实现 RGB 颜色的复制。

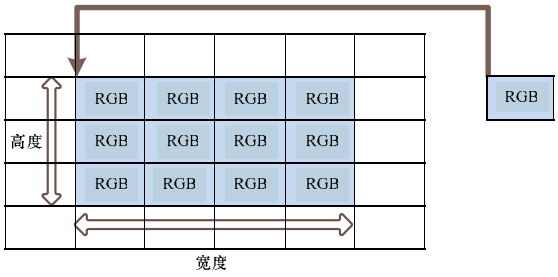

图 3　二维内存 DMA 绘制矩形区域

对于内存目的地址，*Y_MODIFY*=(屏幕宽度-区域宽度)×3。因此，用 DMA 绘制矩形区域的步骤如下：

(1)计算填充色的 RGB 值。

(2)设置内存 DMA 源控制器，起始地址为填充色的内存地址，*X_COUNT*=3, *X_MODIFY*=1, *Y_MODIFY*=-3, *Y_COUNT*=*width*×*height*×3。

(3)计算矩形左上角像素地址：*addr=fb_base* ＋ (*Y*×*fb_width*＋*X*)×3。

(4)设置内存 DMA 目标控制器：*X_COUNT=width*×3, *Y_COUNT=height*。

(5)设置 DMA 中断处理函数，当 DMA 完成时会触发内核中断，调用这个函数。须在中断函数中设置一个全局变量，用以标记 DMA 的执行情况。

(6)开启内存 DMA 源控制器和目标控制器。

对于区域拷贝，同样只需修改内存源的参数配置，配置好内存源宽度、高度和地址，即可实现区域拷贝功能。

2.4.2　扫描线算法的 DMA 优化

本嵌入式 GUI 系统使用扫描线算法来实现多边形区域填充。扫描线算法是按扫描线顺序，计算扫描线与多边形的相交区间，再用填充色填满这些区间的像素，完成填充工作。

Blackfin 系列处理器提供了强大的 DMA 能力，在填充扫描线区间时，可使用 DMA 来完成。由于在执行 DMA 的时候无须 CPU 干预，因此只需设置好 DMA 寄存器并启动 DMA 后，即可进行下一条扫描线的计算，不必等待填充完成。可以实现数据操作和 CPU 运算的并行处理，提高效率。

用 DMA 优化扫描算法，如图 4 所示。对于第 1 条扫描线，当计算好需要填充的区间为 *AB* 区间后，设置并开启 DMA 控制器，其起始地址为 *A* 像素点的内存地址，长度为 *AB* 点距离。开始计算下一条扫描线与多边形的交点，在计算的同时，DMA 已经开始进行填充，当第 2 条扫描线计算好交点为 *CD* 点后，等待 *AB* 区间填充完成，然后开始进行 *CD* 区间的填充，并计算下一条扫描线。

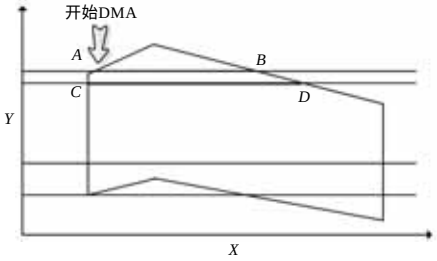

图 4　扫描线算法的 DMA 优化

其中，在进行 *CD* 区间填充之前是否需要等待，取决于 DMA 填充 *AB* 区间的时间和计算第 2 条扫描线交点的时间。

2.5 汇编优化技术

可以通过汇编重写关键代码来提高显示性能。Blackfin 处理器在每个时钟周期内除了能够完成多个 ALU/MAC 操作外，在同一时钟周期内还可以完成 1 次存取数据的操作，并更新地址[5]。指令如下：

```
R2.H=(A1+=R1.H*R0.H)
R2.L=(A0+=R1.L*R0.L)||R0=[I0--]||[I1++]=R2
```

该指令的执行框架如图 5 所示。

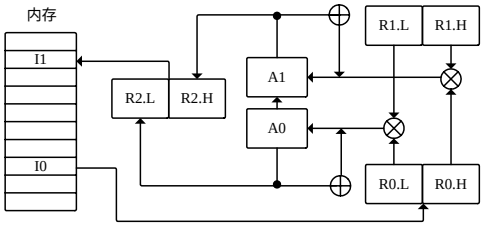

图 5　单周期多数据流指令

这是一条单周期多数据流指令，它将 R1.H*R0.H 的乘积累加到 A1，存储在 R2.H，并将 R0.L*R1.L 的乘积累加到 A0，存储在 R2.L，然后将 R2 寄存器的值存储到内存 I1 的位置，并增加 I1 指针，同时读取内存 I0 位置的值到 R0，并将 I0 指针自减。

使用这样的汇编指令，可以在一个周期内完成多条运算和存取指令，可以大大增加程序的执行效率。

另外，可以利用硬件循环缓冲区和零开销硬件循环来优化循环结构[6]。Blackfin 汇编使用 LSETUP(loop, label)指令设置硬件循环开始和结束标号。这样，硬件循环缓冲区和循环计数器无须在每次循环末尾加入跳转指令，减少了循环开销。

3　实验结果

下文在相同硬件平台上用相同或相似的软件操作，将本嵌入式 GUI 与 Microwindows 的各项基本性能进行比较。测试平台：Blackfin BF531 处理器，主频 405 MHz，总线频率 101 MHz，内存 64 MB，显示设备：8 英寸数字液晶屏。

3.1 直线绘制

为测试 GUI 的直线绘制性能，本文分别测试了绘制一定数量水平线、垂直线和普通直线的绘制时间，并与 Microwindows 软件进行对比。

测试方法如下：液晶屏的分辨率为 800×600，对于水平直线，连续绘制 256 组，每组 600 条，从上到下填充屏幕。对于垂直直线，连续绘制 256 组每组 800 条，从左到右填充屏幕。对于普通直线，连续绘制了 100 000 条随机直线，然后计算其消耗的时间。利用 Blackfin 汇编指令可以获得 CPU 当前的 Cycle 数，2 次 Cycle 数相减即可得到 2 次之间消耗的时钟周期数，可以根据 CPU 时钟频率进一步得到消耗的时间。测试结果如图 6 所示。

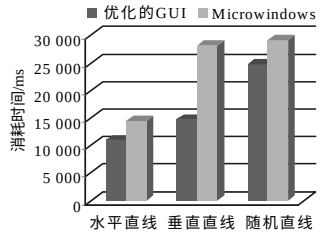

图 6　GUI 直线绘制时间比较

测试分析：对于水平直线，由于未优化的 Microwindows 使用循环的方法操作显存，而水平直线在显存中的内存区域是连续的，CPU 的数据预取可以提高绘制效率。而对于垂直直线，其像素点在内存中的分布不连续，因此，Microwindows 的绘制时间明显变长。对于本文优化过的 GUI，不论水平还是垂直直线，都可使用二维 DMA 一次完成，时间变化不明显，而且都要优于 Microwindows。

对于随机直线本 GUI 在算法上与 Microwindows 相同，只是优化了像素绘制函数，其效果不如前面的明显。

3.2 多边形填充

为了测试 GUI 的多边形填充性能，本文通过绘制最简单的三角形来测试 GUI，并与 Microwindows 进行对比。

测试方法如下：随机生成 100 到 40 000 个三角形，并用随机生成的颜色填充三角形，计算其平均每个三角形的填充时间。同样利用汇编指令获得 CPU 的 Cycle 数来计算消耗的时间。测试结果如图 7 所示。

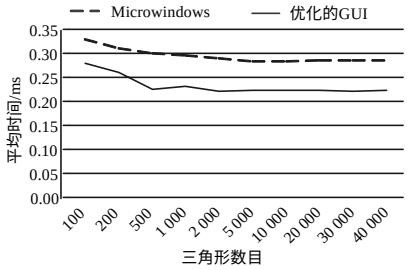

图 7　GUI 填充三角形平均时间比较

测试分析：在三角形数目较少的情况下，优化的 GUI 和 Microwindows 的差别不明显，当三角形数目达到 500 个以上的时候，可以明显看出优化的 GUI 比 Microwindows 的三角形填充时间减少。

由于本 GUI 使用 DMA 优化了多边形的扫描线填充算法，其填充和计算交点可以并行执行，因此性能提高较大。

3.3 多窗体绘制

为了测试本 GUI 的多窗体绘制性能，本文通过在程序中绘制大量窗体来测试其性能。

测试方法：通过生成 1 000 个大小和位置均随机的窗体，并用随机生成的颜色绘制窗体。每绘制一个窗体，都将其设置为激活窗体，并置于所有窗体的顶层显示。然后测试其总共绘制时间，并与 Microwindows 进行比较，测试结果见图 8。

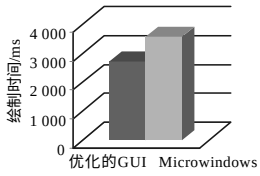

图 8　GUI 窗体绘制时间比较

测试分析：在绘制大量窗体时(1 000 个随机窗体)，Microwindows 需要大约 3 573 ms，而本嵌入式 GUI 使用了优化的窗体绘制函数以及区域填充函数，只需 2 715 ms，少于 Microwindows。这是因为本 GUI 中对窗体的绘制函数使用了硬件抽象层优化的直线和区域填充算法，在多窗体情况下能够较快绘制出窗体。

经过以上几项测试可以看出，本嵌入式 GUI 在各项基本绘制性能上均要优于 Microwindows。

(下转第 274 页)