嵌入式多任务 GUI 的通用解决方案

■电子科技大学　丘允阳　孙世新

摘　要　针对嵌入式多任务 GUI 系统需要非常强的灵活性、可移植性和可伸缩性的特点，提出一种通用解决方案；采用层次化、模块化和面向对象的设计思想，给出了 GUI 的体系结构，并对多任务 GUI 设计中的多任务调度策略及管理、消息驱动机制、桌面及窗口管理和对象树等关键技术进行了研究。该解决方案的原型已经成功应用于我国自主产权的实时操作系统 DeltaOS 中。

关键词　嵌入式系统　嵌入式 GUI　图形用户界面　多任务

嵌入式 GUI（Graphic User Interface）系统就是在嵌入式系统中为特定的硬件设备或环境而设计的图形用户界面系统。调查显示，越来越多具有灵活性、高效性和可移植性的嵌入式 GUI 系统被广泛应用于办公自动化、消费电子、通信设备、智能仪器等许多领域；而且随着硬件技术的发展，要求 GUI 实现的功能越来越丰富，GUI 系统也变得比以往更加复杂、多样。

大多数的嵌入式 GUI 系统都只能简单地支持单任务。单任务 GUI 的效率较低，无法满足未来 GUI 的发展需求，故多任务 GUI 是嵌入式 GUI 的发展方向。

目前，在嵌入式应用领域比较成功的嵌入式多任务 GUI 系统主要有：MiniGUI、MicroWindows 和 Qt/Embedded。它们主要是针对嵌入式 Linux 而设计的，通过 PThred 库来实现对多线程的支持；而 PThred 本身就比较复杂，很难将这些 GUI 系统移植到平台接口不符合 POSIX 标准的目标平台上。所以，上述几种 GUI 系统有一个共同的缺点，就是过分地依赖于某种特定的平台，导致可移植性差。

为了有效地兼容嵌入式领域的各种环境，下面提出一种通用、有效并且可移植性好的嵌入式 GUI 体系结构，并对多任务 GUI 设计中的关键技术进行研究。

1　体系结构

针对 GUI 需要非常强的灵活性、可移植性和可伸缩性的特点，在其体系结构的设计中，采用层次化、模块化和面向对象的设计思想。层次式的体系结构在许多软件系统中被采用，被公认为是一种合理的结构，但最重要的是如何来划分这些层次，使系统的结构最合理、最清晰。

在设计中采用如下划分策略：力求层次之间相对独立，对任一层的改动保证它对上层的接口不变，上层不受下层变化的影响。在这样的层次结构中，最底层和最高层都可能根据具体需要发生改变，因此应该为这两层提供充分的变动空间，而中间的层次则应是独立不变的。

GUI 在嵌入式应用环境中，硬件环境、操作系统和用户应用程序之间的层次如图 1 所示。

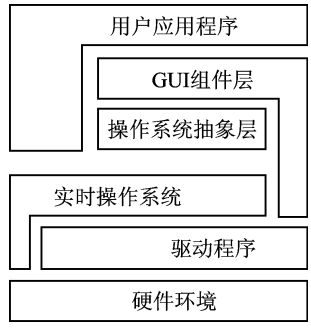

图 1　层次体系

图 1 中，通过驱动程序，GUI 组件部分与硬件隔离；通过操作系统抽象层，使核心与具体操作系统隔离。这种层次体系结构使得 GUI 具有良好的平台无关性，在不同的操作系统和硬件平台之间的移植非常方便。

按上述设计思想，GUI 层次模型划分如图 2 所示。图中，GUI 被划分为 3 层，每一层又根据具体功能的不同被划分为几个模块。

1.1　输入输出层

这一层的功能是将系统中设备和操作系统平台的具体细节屏蔽起来。该设备层被定义在 BSP 中，向上提供 GUI 对设备的显示特性操作。该层分为设备逻辑和硬件抽象两个子层。设备逻辑子层使用同一类设备的概念来描述 GUI 支持的外部设备及对该设备的逻辑操作，向上层提供了统一的设备操作接口；而硬件抽象子层则利用实际的设备控制器操作，根据硬件在不同平台上的驱动来实现硬件抽象子层中定义的接口。

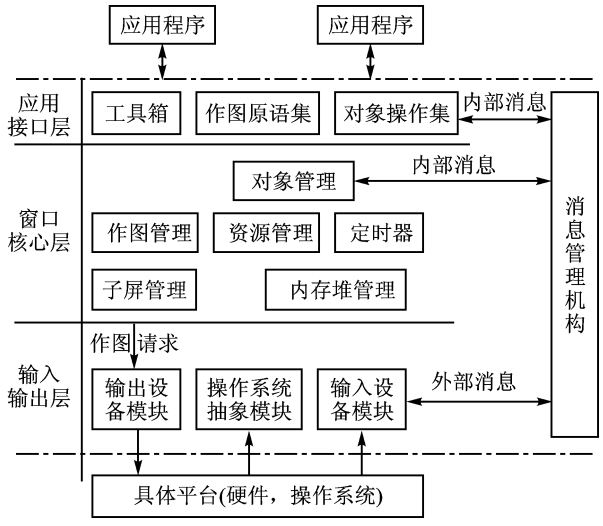

图 2 GUI 层次模型划分

1.2 窗口核心层

窗口核心层实现 GUI 的关键功能，根据功能可划分为消息管理、缓冲池管理、作图管理、定时器、资源管理、对象管理、子屏管理和内存堆管理等几部分。

由于 GUI 采取消息驱动的通信方式，因此消息管理构成 GUI 的灵魂，通过它把系统的各个部分联系起来。在应用程序的运行过程中，消息承载了系统各部分间的交换信息。

内存堆管理：目的是避免在系统运行过程中动态分配和释放内存时引起存储碎片。两个比较频繁的动态内存分配操作是消息空间的申请/释放和屏幕对象剪切域的刷新。

作图管理：完成画点、画线、画圆等作图操作。为了提高 GUI 的移植性，这层主要完成与硬件无关的作图过程。对具有特殊显示功能的应用平台来说，此层也可以大量调用硬件提供的功能函数(经输出硬件无关层包装后的接口)，从而实现特殊的作图效果。GUI 在结构上提供了这种灵活性。作图管理层向上以作图原语集的形式为应用提供调用接口。

定时器：根据系统时钟来为应用提供计数信息。

资源管理：主要对字体、图片和调色板等进行管理，需要实现资源的存储以及为应用提供适当的接口两大功能。

对象管理：采取合理的机制来组织在屏幕上显示的对象。GUI 把所有能在屏幕上显示的 GUI 元素称为“对象”，并通过对象树、Z 序和剪切域等机制来实现对象的管理。窗口核心层还为应用提供了添加、删除、隐藏对象等操作接口函数。

1.3 应用接口层

应用接口层封装了 GUI 为用户提供的一切接口，应用程序看到的 GUI 就是此层提供的所有接口函数，包括工具箱、作图原语集和对象操作集等 3 部分。

工具箱：GUI 为用户提供的控件集。这部分的大小是可根据应用的需要调整的，从而也在较大程度上影响了 GUI 库的大小。常用的控件有按钮、滚动条、窗口和编辑框等。

作图原语集：作图管理层提供的绘图函数接口。工具箱也是在它的基础上实现的。

对象操作集：主要实现 GUI 对象的添加和删除等操作功能。

2 多任务 GUI 关键技术分析

本文中提到的“任务”，在同一个地址空间执行并且可以无约束地直接访问所有的共享资源。下面对多任务 GUI 设计中的关键技术进行分析。

2.1 多任务调度策略及管理

多任务系统需要一个合理的任务调度策略来管理所有的任务。启动 GUI 后，会默认产生一个系统任务、一个事件任务和一个定时器任务；而应用任务的产生则要根据具体的用户需求。

(1) 系统任务

桌面对象运行的任务称为“系统任务”。系统任务是整个图形用户系统运转的核心。它不断地从系统的主消息队列中取消息，并按消息的目的和用途将其派发到对应的目的任务；同时负责所有应用任务的管理和维护以及桌面管理等事务。一个系统中仅有一个系统任务。

(2) 事件任务

事件任务负责收集外部事件，并将事件解释为与之对应的 GUI 消息，并放入系统的主消息队列中。用户的输入正是从这里传入 GUI 核心进行处理。一般情况下，一个系统中仅有一个事件任务。

(3) 定时器任务

定时器任务是通过操作系统的系统调用产生 GUI 定时器的。

(4) 应用任务

除系统任务之外，窗口所运行的其他任务称为“应用任务”。应用任务是用户程序运行的基本单位。应用逻辑都在此任务中运行，拥有自己的消息队列，接收来自 GUI 核心的消息，按一定规则独立进行消息循环。应用任务通过消息与系统任务交互，受系统任务的管理；通过应用接口层使用系统的软硬件资源。应用任务的上限数目仅受平台的资源数目限制。

在嵌入式 GUI 中，系统任务被赋予最高的优先权，其他任务可使用低于系统任务优先级的不同优先级；相同优

先级应采用时间片轮转方式运行。总之，如果有必要，则嵌入式GUI系统会同时采取时间片轮转和优先级抢占的任务调度策略，如图3所示。

在没有消息到达或者在等待某个事件时，任务需要将自己挂起。一旦有消息进入消息队列，任务将会被唤醒以处理消息。这样，有限的CPU资源就能够被充分地利用。

此外，系统任务维护一个用于跟踪及管理应用任务的列表。每一个应用任务对应一个包含其属性信息的任务信息块。任务属性包括消息队列指针、任务句柄和任务入口等，对指定任务进行了详细的描述。一个信息块的创建与销毁必须与应用任务的创建与销毁保持同步。系统任务必须维护该列表，以保证这些信息块的正确性。多任务管理机制如图4所示。

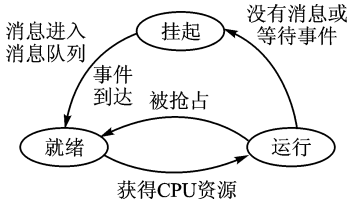

图3 任务调度策略

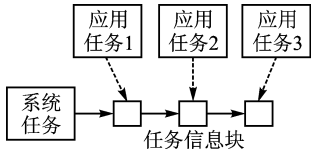

图4 多任务管理机制

对用户来说，只需要简单地指定任务的入口和优先级(如果有必要的话)，其他所有的工作都由系统任务自动完成。采用这种调度和管理方法，能使系统变得更加易用和高效。

2.2 消息驱动机制

消息机制的提出，最初是为了解决早期程序设计中基于硬件中断的事件处理问题。中断事件的发生是不可预期的、突发性的，因此当有多个应用等待并处理中断事件时，就会出现问题。消息机制可以很好地解决事件驱动的多应用设计问题，并且可以形成一种处理多个系统之间、系统内部件和部件之间关系的简洁而且可靠的方法。

在只支持单任务的GUI系统中只有一个串行化的消息队列，消息严格地按照顺序处理，导致响应慢、效率低，故有必要采用并行化的消息队列。当一个队列中的消息忙于处理一个冗长工作时，输入焦点能够切换到另外一个队列。

系统维护一个系统消息队列和多个指定任务信息队列，每个指定任务信息队列对应一个应用任务，如图5所示。事件任务将输入转化成消息并将其放入系统消息队列中，系统任务检索到输入消息后首先会对该消息进行检查，然后将消息邮寄给目标应用任务或直接对消息进行处理。每个应用任务从它的消息队列中移出消息，并将其发送到适当的窗口程序中进行处理。一个应用任务可以将消息邮寄到它自己的消息队列，也可以邮寄到其他应用任务的消息队列中。

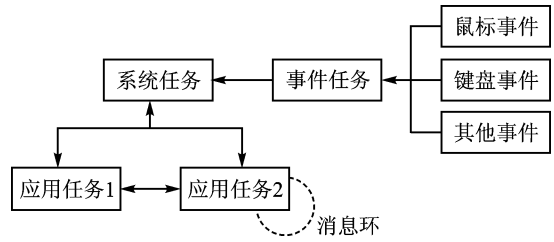

图5 消息驱动机制

此外，为了达到不同的目的，系统提供了同步和异步两种基本类型的消息。

2.3 桌面及窗口管理

屏幕中的窗口通常是相互覆盖，而且它们的相对位置也在不断地变动。这些窗口有可能分别属于不同的任务，但共享一个相同的屏幕。所以，如何方便并有效地计算和维护窗口便显得很重要。

首先介绍2个概念：全局剪切域和窗口剪切域。它们都与应用任务相关，前者指出哪些区域占据屏幕，后者给出在相同的应用任务中所有对象的剪切关系。

系统任务除了要维护它自己的剪切域外，还要负责管理所有应用任务的全局剪切域，如图6所示。一旦窗口位置发生变化，系统任务必须更新信息并且通知应用任务作出相应的变动。另一方面，在计算窗口剪切域时，应用任务只须关心它本身而不受其他任务的影响，好像整个屏幕只有它一个任务在运行。至于最后实际的剪切结果，只需将全局剪切和窗口剪切结果进行“与”运算。

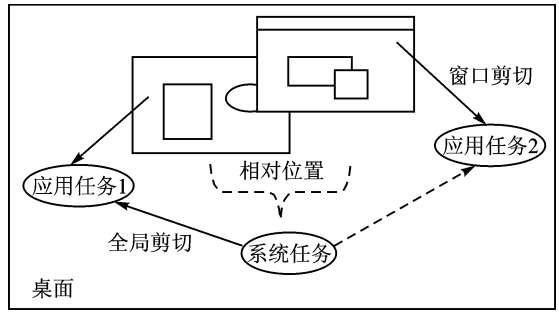

图6 剪切管理

2.4 Z序和对象树

Z序实际定义了屏幕中对象集(可显窗口)之间的层次关系。GUI用户可以通过选择要被置于前端的窗口来改变Z序。很多GUI系统将Z序当作一个带有指定运算集的显式列表来实现。然而，在嵌入式GUI系统中不用这种方式，而是通过对象树来说明这些GUI对象之间的层次关系和Z序。理论上，每一个GUI对象都有父亲、孩子和兄弟。因此，所有显示在屏幕上的对象便构成

24 位 EEPROM 具体写程序如下：

```
void SPI_WrieSignal(Uint32 address, Uint32 *dataadd,
        Uint32 datalength) {
  Uint32 k =0;                    //计数器
  Uint16 EPROM_status = 0; //状态寄存器
  int i=0;                        //计数器
  SPI_WriteEN();                  //写EEPROM 使能WREN
  EPROM_status = SPI_ReadStatusReg();
      //读 EEPROM 状态寄存器 RDSR
  while(!(EPROM_status &0x2)){};
      //判断 EEPROM 状态寄存器的低二位是不是为 1
  hMcbsp = MCBSP_open(MCBSP_PORT0, MCBSP_
          OPEN_RESET); //打开 MCBSP0
  SPI_wrdatainit(hMcbsp);
                  //初始化 MCBSP0 为时钟停止模式
  address = address &0xFFFFFF;
                  //获取要写的 EEPROM 的地址
  while(!MCBSP_xrdy(hMcbsp)){};
                  //如果 EEPROM 已经准备好
  GPIO_RSET(IODATA,0x00);
                  //复位 GPIO
  MCBSP_write32(hMcbsp,(SPI_WRITE + address));
                  //写入 6 位命令和 24 位地址
  SPIWR_Delay(); //写时延
  for(i=0;i<datalength;i++) {
      while(! MCBSP_xrdy(hMcbsp)){};
                  //如果 EEPROM 已经准备好
      MCBSP_write32(hMcbsp, *dataadd);
                  //写入 32 位数据
      dataadd++;                  //dataadd 自加 2
      SPIWR_Delay();              //读写时延
  }
  GPIO_RSET(IODATA,0x10);         //复位 GPIO
  for(k = 0; k<0x10000;k++)       //时延
  {}
  MCBSP_close(hMcbsp);            //关闭 MCBSP0
}
```

3 结 论

本设计方案大大节省了设计空间，降低了功耗，经过实际安装与调试完全可行。说明了基于高密度 SPI EEPROM 的 DSP 应用系统独立运行的过程。与同类的 SPI EEPROM 相比，具有容量大的特点；与 Flash 引导相比，具有读写、擦除简单的特点。硬件电路具有良好的可扩展性。以此电路为基础，可以适用于 C5509A 的小型化、低功耗和便携式应用的开发。

参考文献

[1] Using the TMS320VC5503/VC5507/VC5509/VC5509A Bootloader. Application Report SPRA375E, October 2004.

[2] TMS320C55X Assembly Language Tools User's Guide. Literature Number: SPRU280G March 2003.

[3] 刘益成. TMS320C54x DSP 应用程序设计与开发[M]. 北京:北京航空航天大学出版社, 2002.

[4] TMS320VC5501/5502/5503/5507/5509/5510 DSP Multichannel Buffered Serial Port (McBSP) Reference Guide, Literature Number: SPRU592E April 2005.

[5] SA25C020 Advanced Information. 20, July, 2003.

(收稿日期: 2006 08 11)

一棵以桌面为根节点的倒置树。对树进行“后根遍历”就能够容易地得到 Z 序。

图 7 描述了对象树的建立过程。对象树的采用极大地简化了桌面管理，能够在不增加额外工作的情况下方便地组合对象和实现 Z 序管理。

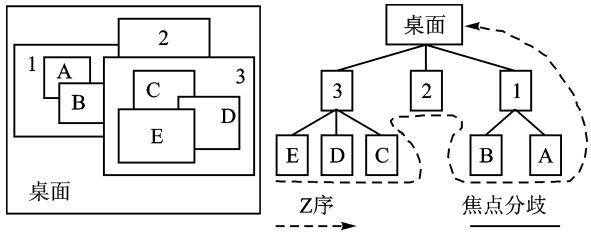

图 7 Z 序和对象树

3 小 结

将来的 GUI 系统将越来越复杂，要求 GUI 系统实现的功能也越来越丰富，这就需要一个更加开放且伸缩性好的体系结构。本文提出的嵌入式 GUI 体系结构具有很强的灵活性，且可移植性好，能够很好地应用于嵌入式领域的各种环境。

参考文献

[1] 罗蕾. 嵌入式实时操作系统及应用开发[M]. 北京:北京航空航天大学出版社, 2005.

[2] Charles Petzold. Windows 程序设计[M]. 第 5 版. 北京:北京大学出版社, 1999.

[3] Don Batory, Sean O Malley. The design and implementation of hierarchical software systems with reusable components [J]. ACM TOSEM archive Volume1, ACM Press, 1992.

[4] Luo Qi, Luo Lei. A Universal Solution of an Embedded Multitasking GUI System[C]. ICESS'05 archive Volume 00, IEEE Computer Society, 2005.

[5] Richard N Taylor, et al. Chiron-1: a software architecture for user interface development, maintenance, and run-time support[J]. ACM TOCHI archive Volume2, Issue 2, ACM Press, 1995.

(收稿日期: 2006 09 11)