摘 要

嵌入式图形用户界面是嵌入式设备与使用者之间的对话接口，由于它具有良好的人机交互性能，因此在嵌入式软件系统中得到广泛的应用。随着嵌入式设备在人们日常生活中的使用越来越广泛，人们对高性能嵌入式图形用户界面的要求也越来越迫切。在此背景下，对嵌入式系统的关键技术——嵌入式 GUI 的研究有着非常重要的意义。

在变电站的日常运行和维护的过程中，需要将大量的遥信、遥测以及保护自动装置的信息通过显示器显示出来供维护人员观测。传统的变电站自动装置的人机界面已经远远不能适应当前的需求，因此开发研制出一套便于操作和使用的嵌入式图形用户界面对变电站自动化的水平的提高有非常大的帮助。

本文首先分析了嵌入式操作系统及其 GUI 技术的发展现状，并对几种嵌入式 Linux 下的 GUI 系统进行了优缺点的比较。通过阅读和分析 Qt/Embedded 的源代码，对 Qt/Embedded 的系统架构进行了深入的剖析和研究，从不同的角度分析了它的运行机制，为后面的移植及软件编写奠定了基础。然后通过对变电站通信控制器 GUI 系统项目需求的分析，给出了系统软硬件开发的总体设计方案。最后，在选定的硬件平台上实现了 BootLoader 的移植、嵌入式 Linux 的移植及其根文件系统的建立、Qt/Embedded 的移植以及变电站通信控制器图形用户界面软件的设计与实现等工作。

实践证明，该 GUI 系统可以较好的满足变电站自动化对人机界面的需求。

关键词：嵌入式系统　图形用户界面　变电站自动化　嵌入式 Linux　Qt/Embedded

ABSTRACT

Embedded graphic user interface (GUI) is the dialogic interface between embedded devices and users. Because of its' good interactive performance, embedded GUI has been widely used in embedded software systems. As the wider use of embedded devices in our daily lives, the requirement for higher performance of embedded user interface is more urgent. So, researches on embedded GUI, which is the critical technology of embedded system, are very important.

In the process of daily operating and maintaining of substation, a large number of information of tele-indication, tele-measurement, and self-protection devices are acquired and displayed in screens to be observed by the maintainers. The traditional interface in substation automation devices can not meet the current demand far. As a consequence, whether there is an embedded GUI in the communication controller, and whether the embedded GUI is easy to operate and control or not, have critical influences on the automation level of a substation.

In the first part of this thesis, the developments on embedded operating systems, especially on the embedded GUI technology, were summarized. And several kinds of embedded GUI systems in Linux were also compared in this part. Then, by reading and analysing the codes of Qt/Embedded, the system architecture of the Qt/Embedded was analyzed in detail. And the operation mechanism of this system was also studied from several aspects. The above parts build up the foundation of porting and software development for the project in this thesis. Following that, after a comprehensive consideration of the project requirements of the communication controller's GUI system for a substation, the design scheme that Qt/Embedded was adopted as the main development tool is established. Finally, the main parts of this project, including porting BootLoader to the specific hardware platform, porting embedded Linux system and establishing its root filesystem, porting Qt/Embedded, and designing the GUI software of communication controller for a substation, were introduced and described in detail.

In practice, the application of the GUI system confirmed that it meet the requirements of substation automation well.

KEYWORDS: Embedded System; GUI; Substation automation; Embedded Linux; Qt/Embedded

第1章 绪论

1.1 变电站综合自动化

变电站综合自动化系统是利用先进的计算机技术、现代电子技术、通信技术和信息处理技术等实现对变电站二次设备(包括继电保护、控制、测量、信号、故障录波、自动装置及远动装置等)的功能进行重新组合、优化设计，对变电站全部设备的运行情况执行监视、测量、控制和协调的一种综合性的自动化系统。通过变电站综合自动化系统内各设备间相互交换信息，实现数据共享，完成变电站运行监视和控制任务。变电站综合自动化替代了变电站常规二次设备，简化了变电站二次接线。变电站综合自动化是提高变电站安全稳定运行水平、降低运行维护成本、提高经济效益、向用户提供高质量电能的一项重要技术措施[1]。

变电站作为整个电网中的一个节点，担负着电能传输、分配的监测、控制和管理的任务。变电站继电保护、监控自动化系统是保证上述任务完成的基础。在电网统一指挥和协调下，电网各节点(如变电站、发电厂)具体实施和保障电网的安全、稳定、可靠运行。变电站自动化系统是变电站的核心系统，对变电站及电网的安全运行是至关重要的。因此，要求变电站综合自动化系统运行高效、实时、可靠，对变电站内设备进行统一监测、管理、协调和控制。同时，又必须与电网系统进行实时、有效的信息交换、共享，优化电网操作，提高电网安全稳定运行水平，提高经济效益，并为电网自动化的进一步发展留下空间。

发展变电站综合自动化的必要性还体现在以下几个方面：一是随着电网规模不断扩大，新增大量的发电厂和变电站，使得电网结构日趋复杂，这样要求各级电网调度值班人员掌握、管理、控制的信息也大量增长，电网故障处理和恢复却要求更为迅速和准确；二是现代工业技术的发展，特别是电子技术的发展，计算机技术的普遍应用，对电网可靠供电提出了更高的要求；三是市场经济的发展，使得整个社会对环保要求更高，这样也对电网的建设、运行和管理提出许多的要求，如要求电力企业参与市场竞争，降低成本，提高经济效益；要求发电厂、变电站减少占地面积。要解决上述问题，显然仅依靠各级电网调度运行值班人员是难以解决的。现代控制技术的发展，计算机技术、通信技术和电力电子技术的进步与发展，电网自动化系统的应用，为上述问题提供了解决方案。这些技术的综

合应用造就了变电站综合自动化系统的产生与发展[2]。

1.2 图形用户界面(GUI, Graphics User Interface)

计算机用户界面是指计算机与其使用者之间的对话接口，是计算机系统的重要组成部分。计算机的发展史不仅是计算机本身处理速度和存储容量飞速提高的历史，而且是计算用户界面不断改进的历史。人机交互界面作为一个独立的、重要的研究领域已倍受各界人士的关注。从计算机技术的发展过程来看，人机界面技术引导了相关软硬件技术的发展，是新一代计算机系统取得成功的保证。

计算机发展的初期，主要用于科学计算等任务，当时用户不关心界面方面的细节。随着计算机的发展和普及，人们对人机界面的要求越来越高，对它的研究也受到了人们的高度重视。研究表明，人机交互的内容、形式将影响最终用户使用计算机和计算机技术的推广，甚至影响人们的工作和生活。国外对大量软件系统的统计分析结果表明，人机界面的开发工作量极大，占软件系统开发总工作量的 40%-60%左右，而且不同用户对界面的要求不同，使得它成为计算机软件研制中最困难的部分之一。美国 21 世纪信息技术计划的基础研究内容(软件、人机交互、网络、高性能计算)中就包含了人机交互技术的研究。与此同时，围绕人机建模已形成了计算机产业又一新的竞争领域。美国人机建模研究在信息技术中被列为与软件和计算机并列的六项国家关键技术之一，并被认为“对于计算机工业有着突出的重要性，对其它工业也是很重要的”。美国国防关键技术计划不仅把人机交互界面列为软件技术发展的重要内容之一，而且还专门增加了与软件技术并列的“人机界面”这一内容。日本 FPIEND21 计划(Future Personalized Information Enviroment Development)的目标就是开发 21 世纪的计算机界面。我国 973、S-863、十五计划均将人机界面技术列为主要内容。

GUI 是“Graphics User Interface”首字母的简写，一般译为“图形用户界面(或者图形用户接口)”。它是一种人与计算机接口的技术，该技术除了使用字符外，主要使用图形、图标、图象和控件等界面与用户进行交互和接口。计算机用户通过使用输入设备(如鼠标、键盘、触摸屏等)操作和使用图象、图标和控件等与计算机进行交互，计算机将结果显示在输出设备上供用户观察。GUI 是在命令行方式上，人机接口的一次巨大飞跃。

窗口系统(Windows)最早出现在名为 SAGE 的实时图象显示系统中。上世纪

60 年代，人们在研究 SRI 人工智能系统时，第一次使用鼠标操作窗口系统；70 年代早期，Xerox 的 PARC 团队在移植 SRI 时，将 GUI 定义为 WIMP 标准，即 Windows、Icons、Menus 和 Pointers/Pull-down Menus，基本上形成了现代 GUI 系统的雏形。随后 GUI 被广泛应用，得到了大力发展，第一个商业化的 GUI 系统 Macintosh，以及随后的 Windows 系统，极大的推动了计算机的推广和普及。

图形用户界面系统在嵌入式系统上的发展，与在桌面系统的发展类似，基本上是一个从无到有、从字符界面到使用图形图象交互的过程。早期的工控系统基本没有用户界面，或者仅仅靠简单的文字信息和 LED 显示与用户进行交互。随着嵌入式技术的发展，近年来消费电子、通信、汽车、工业和军事等领域广泛采用嵌入式系统。在信息家电、PDA，Smart Phone 等众多受欢迎的终端产品中，已经可以看到相对成熟的 GUI 系统。完善的图形用户界面(GUI)不仅可以表示丰富的内容，而且具有多种表达方式，已经成为现代终端系统和嵌入式系统的重要组成部分，也是当今主流的人机界面。

同时，终端系统已经开始广泛采用 32 位处理器芯片，配置触摸屏、键盘等多种输入设备和 LCD 等图象显示设备，这为 GUI 在嵌入式系统上应用提供了基础硬件平台。因此，嵌入式 GUI 系统的发展不但成为可能，而且也是应用发展的必然要求。

图形用户界面的广泛流行是当今计算机技术的重大成就之一，它极大地方便了非专业用户的使用，人们不再需要死记硬背大量的命令，而可以通过窗口、菜单方便地进行操作。图形用户界面的主要特征是：

- WIMP，其中

W(Windows)指窗口，是用户或系统的一个工作区域，一个屏幕上可以有多个窗口。

I(Icon) 指图标，形象化的图形标志，易于人们隐喻和理解。

M(Menu)指菜单，可供用户选择的功能提示。

P(Pointing Devices)指鼠标器等，便于用户直接对屏幕对象进行操作。

- 用户模型

GUI 采用了不少 Desktop 桌面办公的隐喻，使应用者共享一个直观的界面框架。由于人们熟悉办公桌的情况，因而对计算机显示的图形符号的含义容易理解，

例如：文件夹、收件箱、画笔、工作薄、钥匙及时钟等。

- 直接操作

过去的界面不仅需要记忆大量命令，而且需要制定操作对象的位置，如行号、空格数、X 及 Y 的坐标等。采用 GUI 后，用户可以直接对屏幕上的对象进行操作，如拖动、删除、插入以至放大和旋转等。用户执行操作后，屏幕能立即给出反馈信息或结果，因而被称为“所见即所得(What You See Is What You Get)”。并且用视、点(鼠标)代替了记、击(键盘)，给用户带来了方便。

1.3 课题的目的与意义

变电站综合自动化系统要将大量的遥控、遥信、遥测以及保护自动装置信息通过显示器显示出来，因此人机界面是否友好，成为变电站运行人员关心的问题，也是自动化专业及综合自动化厂家需要解决的问题。传统的变电站自动化装置一般只有中英文数据显示，交互信息少，调试运行界面简陋，往往导致使用上的困难和操作上的失误。随着变电站自动化水平日益提高，要求电力自动装置调试操作更加方便，人机界面更加友好智能，交互信息更加丰富。传统的人机界面显得相对落后，不能满足要求。因此，研制和开发出一种既能显示包括主接线图、三遥数据、设备定值、历史事件、通信报文等丰富的系统数据和信息，同时又操作方便、界面友好的人机界面成为了当前的迫切需求。

在今天的变电站自动化系统中，电力系统自动装置所使用的微处理器速度越来越快，性能越来越高，从频率几兆、十几兆的 8 位单片机，走向频率上百兆的 16 位、32 位数字信号处理器和嵌入式微控制器；存储系统容量越来越大，读写越来越方便；而它们的价格也越来越便宜。这使得系统能够承担起采用嵌入式 GUI 所造成的额外 CPU 和存储系统消耗。另一方面，人机接口设备也在不断的升级，就液晶显示器 LCD 而言，已逐渐向大分辨率、灰度和彩屏发展，有些场合已经使用了触摸屏。这些为采用嵌入式 GUI 设计更加友好和更加丰富的人机界面提供了硬件基础。

采用嵌入式 GUI，可以利用它所提供的强大的显示和绘图功能、丰富的图形元素(窗口、菜单、控件等)以及方便的应用编程接口，开发出高质量的人机界面，满足产品的需求。本论文的主要目的是设计出一套能够满足变电站通信控制器实际需求的嵌入式 GUI 系统，从而满足当前变电站综合自动化发展的迫切需求。

嵌入式 GUI 技术在变电站综合自动化中的应用，能够极大节省开发和维护成本，极大丰富人机交互信息，更有利于变电站现代化的进程，并能够大大地提高电力系统的安全运行水平，对国内变电站综合自动化的发展具有重大的意义。

1.4 论文的内容及组织结构

综合电力行业智能化发展的趋势，根据变电站自动化的市场需求，利用嵌入式 GUI 技术实现变电站自动化系统中的人机交互势在必行。本文就是在这种形势下，针对变电站自动化系统中对人机界面的信息丰富性、操作便捷性、安全性、可靠性和工作的高效性的需求，提出了一种基于嵌入式 GUI 技术的人机界面的实现方法。本文所讨论的主要内容包括：基于嵌入式 GUI 的人机界面的系统设计及其在硬件平台上的具体实现。

论文的组织结构如下。

第一章主要结合变电站综合自动化技术的发展趋势，分析了设计和开发基于嵌入式 GUI 技术的人机界面的必要性和重大意义。

第二章介绍了嵌入式系统、嵌入式操作系统及其 GUI 技术的概况，并对几种常见的嵌入式 GUI 系统进行了分析和比较。

第三章在仔细阅读了 Qt/Embedded 源代码的基础上，对其系统架构进行了深入的剖析和研究，为下一步的移植及软件编写奠定了基础。

第四章提出了系统的总体设计方案。

第五章详细介绍了系统平台的搭建及软件设计的具体实现过程，对系统的调试技术做了简单的介绍，最后介绍了对系统进行测试的情况。

第六章对课题进行了简单的总结并对进一步的工作进行了展望。

第2章 嵌入式系统及其 GUI 技术概述

2.1 嵌入式系统概况

2.1.1 嵌入式系统的定义

嵌入式系统的一般定义是："以应用为中心、以计算机技术为基础、软件硬件可裁剪、适应应用系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统"。广义上讲，凡是带有微处理器的专用软硬件系统都可以称为嵌入式系统。所以也有人简单的说："嵌入式系统是指操作系统和功能软件集成于计算机硬件系统之中。"狭义上讲，人们更加强调那些使用嵌入式微处理器构成独立系统，具有自己的操作系统并且具有某些特定功能的系统[3]。

在现在日益信息化的社会中，计算机与网络已经渗透到我们日常生活的每一个方面，而嵌入式系统，正是这个渗透过程的主要推动力量。与我们生活息息相关的家用电器、汽车电子、我们随身携带的手机、MP3、手表、PDA、数码相机、数码录像机，这一切都与嵌入式系统密切相关；而在工业领域，使用嵌入式设备控制的生产流水线、数字机床、智能工具也正在其中扮演着极其重要的角色。

与通用计算机不同，嵌入式系统是针对具体应用的专用系统，一般具有成本敏感性，它的硬件和软件必须高效地设计，好的嵌入式系统是完成目标功能的最小系统。嵌入式系统一般要求高的可靠性，例如在高温、高压、电磁干扰严重的工业环境就对嵌入式系统有很高的要求。嵌入式处理器的功耗、体积、处理能力在具体应用中也有很高的要求，这在消费类电子产品方面表现的非常明显。嵌入式处理器要针对用户的具体需求，对芯片配置进行裁减和添加，才能达到理想的效果。嵌入式系统软件与嵌入式应用软件也与通用计算机有所不同。一般嵌入式软件要求高质量的代码与高可靠性。另外，许多嵌入式应用系统要求系统软件具有实时处理能力，在多任务嵌入式系统中，对重要性各不相同的任务进行统筹兼顾的合理调度是保证每个任务及时执行的关键。

2.1.2 嵌入式系统的历史沿革

嵌入式系统的概念是在 1970 年左右出现的。随着新技术的不断出现，嵌入

式系统也不断的向前发展，进入 90 年代后，以计算机和软件为核心的数字化技术取得了迅猛发展，掀起了一场数字化技术革命。多媒体技术与 Internet 的应用迅速普及，消费电子、计算机、通信(3C)一体化趋势日趋明显，嵌入式技术再度成为一个研究热点。综观嵌入式技术的发展，大致经历了以下 4 个阶段[4]。

第一个阶段是以单芯片为核心的可编程控制器形式的系统，同时具有监测、伺服、指示等设备与其相配合，典型的应用如数控机床。这种系统大部分应用于一些专业性极强的工业控制系统中，并且一般没有操作系统的支持。它使用汇编语言编程对系统进行直接控制，运行结束后清除内存。这一阶段系统的主要特点是：系统结构和功能都相对单一，处理效率较低，存储容量较小，几乎没有用户接口。由于这种嵌入式系统使用简便、价格很低，因此以前在国内工业控制领域应用较为普遍，但是它已经远远不能适应高效的、需要大容量存储介质的现代化工业控制和新兴的信息家电等领域的需求。

第二个阶段是以嵌入式 CPU 为基础、以简单操作系统为核心的嵌入式系统。这一阶段系统的主要特点是：CPU 种类繁多，通用性比较弱；系统开销小，效率高；操作系统具有一定的兼容性和扩展性；应用软件比较专业，但用户界面不够友好；系统主要用来控制系统负载以及监控应用程序的运行。

第三个阶段是以嵌入式操作系统(EOS)为标志的嵌入式系统。比较典型的嵌入式操作系统包括：VxWorks、Linux、pSOS、OS-9、WinCE 等。这一阶段系统的主要特点是：嵌入式操作系统能运行于各种不同类型的微处理器上，兼容性好；操作系统内核精小、效率高，并且具有高度的模块化和可扩展性；具备文件和目录管理、设备支持、多任务、网络支持、图形窗口以及用户界面等功能；具有大量的应用程序接口(API)，开发应用程序相对简单；同时，其嵌入式应用软件较为丰富。

第四个阶段是以基于 Internet 为标志的嵌入式系统，这是一个正在迅速发展的阶段。目前大多数嵌入式系统还孤立于 Internet 之外，但随着 Internet 的发展以及 Internet 技术与信息家电、工业控制技术等的结合日益密切，嵌入式设备与 Internet 的结合将代表着嵌入式技术的未来发展方向。嵌入式系统的技术特点

2.1.3 嵌入式系统的技术特点

嵌入式系统一般指非 PC 系统，它包括硬件和软件两部分，它是集硬件、软

件于一体的可独立工作的“器件”。其中硬件部分包括微处理器(MPU)或者微控制器器(MCU)以及相关支撑硬件如：存储器(ROM、RAM、FLASH 等)、显示设备(LED、LCD、触摸屏等)、通讯设备(网络、蓝牙、红外等)及外部 I/O 接口、图形控制器等。软件部分包括操作系统软件(OS)和实现特定功能的应用程序。应用程序负责控制系统的运作和行为；而具有实时性和多任务操作特性的操作系统则负责控制应用程序与硬件之间的交互作用。

其中作为硬件主体的微处理器或微控制器具有如下的特点：

- 对实时多任务有很强的支持能力，能完成多任务并且有较短的中断响应时间，从而使内部的代码和实时内核的执行时间减到最少。
- 具有功能很强的存储区保护功能。这是由于嵌入式系统的软件结构已模块化，而为了避免在软件模块之间出现错误的交叉作用，需要设计强大的存储区保护功能，同时这也有利于软件问题的诊断。
- 可扩展的处理器结构，这样就能够最迅速地扩展出满足应用的最高性能的嵌入式微处理器。
- 功耗极低，尤其是对于那些用于便携式的无线及移动的计算和通信设备中靠电池供电的嵌入式系统更是如此，比如需要功耗只有 mW 甚至 μW 级。
- 嵌入式系统的软件包括与硬件相关的底层软件、操作系统、图形界面、通讯协议、数据库系统和应用软件等。其中应用软件是实现嵌入式系统功能的关键。

对嵌入式系统应用软件的要求也与通用计算机有所不同，有如下几个特点。

- 软件要求固化存储。为了提高执行速度和系统可靠性，嵌入式系统中的软件一般都需要固化在存储芯片或单片机中，而不是存储在磁盘等存储介质中。
- 软件代码具有高质量和高可靠性。尽管半导体技术的发展，使处理器速度不断提高，芯片上的存储容量不断增加，但在大多数应用中，存储空间仍然是宝贵的，同时还存在实时性的要求。为此，要求程序编写和编译工具的质量要高，以尽可能的减少程序二进制代码的长度，提高程序的执行速度。
- 许多应用要求操作系统(OS)具有实时处理能力。在多任务嵌入式系统中，对重要性各不相同的任务进行统筹兼顾的合理调度是保证每个任务及时执行的关键。单纯通过提高处理器速度无法达到要求并且效率低下。因此，这种任务调度只能由嵌入式操作系统来完成，也就要求操作系统具有实时处理能力。

- 需要提供良好的用户界面。对操作系统而言，这里的界面也可以称作编程接口(API), 提供完善方便的编程接口，可以大大简化用户应用程序的开发，同时可以将用户与系统硬件隔离，使所开发的程序具有良好的可移植性。而对于应用程序，良好的用户界面是用户与系统进行交互的必要手段，美观、使用方便是对这部分的基本要求。

总而言之，与通用型计算机系统相比，嵌入式系统具有功耗低、可靠性高；功能强大、性能价格比高；实时性强，支持多任务；占用空间小，效率高；面向特定应用，可根据需要灵活定制等特点，可以嵌入到现有任何信息家电和工业控制系统中。

2.2 嵌入式 Linux

2.2.1 嵌入式操作系统(EOS)概述

嵌入式操作系统是当今嵌入式系统的核心部分。嵌入式操作系统一般可以分为两类。一类是面向控制、通信等领域的实时操作系统；另一类是面向消费电子产品的非实时操作系统。

下面对一些在国内广泛使用的嵌入式操作系统进行简单的介绍[5]。

- VxWorks

VxWorks 是目前嵌入式系统领域中使用最为广泛、市场占有率最高的嵌入式实时操作系统，以其良好的可靠性和卓越的实时性被广泛的应用在军事、航空、航天、电力、通信等高精尖技术及实时性要求极高的领域中。VxWorks 支持多种处理器，如 x86，i960，Sun Sparc，Motorola MC68xxx，MIPS，POWER PC 等等。它使用的是和 UNIX 不兼容的环境，大多数的 VxWorks API 是专有的。采用 GNU 的编译和调试器编写程序。

- QNX

QNX 是一个实时的，可扩充的操作系统。QNX 提供了一个很小的微内核以及一些可选的配合进程。其内核仅提供 4 种服务：进程调度、进程间通信、底层网络通信和中断处理，每个进程在独立的地址空间内运行。所有的其它 OS 服务，都作为协作的用户进程来实现，因此 QNX 内核非常小巧(QNX4.x 大约为 12Kb)而且运行速度极快。

- WinCE

WinCE 是由微软公司推出的嵌入式实时操作系统，全称是 Microsoft Windows CE。它是从整体上为有限资源的平台设计的多任务、多线程、多优先级的操作系统。其模块化的设计允许它对于从掌上电脑到专用的工业控制器的用户电子设备进行定制。WinCE 的基本内核需要至少 200KB。它最大的缺点是实时性不好，是一个软实时操作系统，只能用于对实时性要求不高的场合；同时由于价格太高，使使用它开发的产品在成本上处于劣势。

- Linux

Linux 是一种类似于 Unix 的操作系统，是一个完全免费的操作系统。它的内核代码是全部从头编写的，只是由于它符合 POSIX 1003.1 标准，并且 Unix 中所有的命令它都有，同 Unix 十分相似，所以人们也称它为 Unix 的“克隆”。 自 1991 年诞生至今，Linux 在很多方面已经赶上甚至超过了很多商用的 UNIX 系统。它充分利用了 x86CPU 的任务切换机制，实现了真正的多任务、多用户环境。

嵌入式 Linux 在 Linux 操作系统的基础上，针对嵌入式系统的特点进行了一些裁剪，从而使其可以在资源有限的嵌入式系统上运行。由于 Linux 本身的诸多优势，在嵌入式系统这个 IT 产业的新的关键领域，嵌入式 Linux 逐渐成为了嵌入式操作系统的热点。

2.2.2 嵌入式 Linux 的优势和特色

Linux 从 1991 年问世到现在，短短的十几年时间已经发展成为功能强大、设计完善的操作系统之一。在新兴的嵌入式操作系统领域，Linux 也获得了飞速发展。嵌入式 Linux 的开发和研究是操作系统领域中的一个热点，目前已经开发成功的嵌入式系统中，大约有一半使用的是 Linux。Linux 之所以能在嵌入式系统市场上取得如此辉煌的成果，与其自身的优良特性是分不开的[5]。嵌入式 Linux 有如下一些显著的优点。

- 广泛的硬件支持。Linux 能够支持 x86、ARM、MIPS、ALPHA、PowerPC 等多种体系结构，目前已经成功移植到数十种硬件平台，几乎能够运行在所有流行的 CPU 上。
- 内核高效稳定。Linux 内核的高效和稳定已经在各个领域内得到了证实，Linux 的内核设计非常精巧，分成进程调度、内存管理、进程间通信、虚拟文件

系统和网络接口五大部分，其独特的模块机制可以根据用户的需要，实时地将某些模块插入到内核或从内核中移走。这些特性使得 Linux 系统内核可以裁剪得非常小巧，尤其适合于嵌入式系统的需要。

- 开放源代码，软件丰富。Linux 是开放源代码的自由操作系统，它为用户提供了最大限度的自由度，由于嵌入式系统千差万别，往往需要针对具体的应用进行修改和优化，因而获得源代码是至关重要的。Linux 的软件资源十分丰富，每一种通用程序在 Linux 上几乎都可以找到，并且数量还在不断增加。在 Linux 上开发嵌入式应用软件一般不用从头做起，而是可以选择一个类似的自由软件做为原型，在其上进行二次开发。
- 优秀的开发工具。开发嵌入式系统的关键是需要有一套完善的开发和调试工具。传统的嵌入式开发调试工具是在线仿真器(In-Circuit Emulator，ICE)，它通过取代目标板的微处理器，给目标程序提供一个完整的仿真环境，从而使开发者能够非常清楚地了解到程序在目标板上的工作状态，便于监视和调试程序。但是在线仿真器的价格非常昂贵，而且只适合做非常底层的调试，如果使用的是嵌入式 Linux，一旦软硬件能够支持正常的串口功能时，即使不用在线仿真器也可以很好地进行开发和调试工作，从而可以节省开发费用。嵌入式 Linux 为开发者提供了一套完整的工具链(Tool Chain)，它利用 GNU 的 gcc 做编译器，用 gdb、kgdb、xgdb 做调试工具，能够很方便地实现从操作系统到应用软件各个级别的调试。
- 完善的网络通信和文件管理机制。Linux 从诞生之日起就与 Internet 密不可分，它支持所有标准的 Internet 网络协议，并且很容易移植到嵌入式系统当中。此外，Linux 还支持 ext2、fat16、fat32、romfs 、yaffs 等文件系统，这些都为开发嵌入式系统应用打下了良好的基础。

2.3 嵌入式 GUI 技术

2.3.1 人机界面概述

计算机人机界面是指计算机与其使用者之间的对话接口，是计算机系统的重要组成部分。计算机的发展史不仅是计算机本身处理速度和存储容量飞速提高的历史，同时也是计算机人机界面不断改进的历史。早期的计算机是通过面板上的指示灯来显示二进制数据和指令，人们则通过面板上的开关、扳键及穿孔纸带输

入各种数据和命令。50 年代中、后期，由于采用了作业控制语言(JCL)及控制台打字机等，使计算机可以批处理多个计算任务，从而代替了原来笨拙的手工扳键方式，提高了计算机的使用效率。1963 年，美国麻省理工学院在 709/7090 计算机上成功地开发出第一个分时系统 CTSS，该系统连接了多个分时终端，并最早使用了文本编辑程序。从此，命令行形式的多用户分时终端成为 70 年代乃至 80 年代用户界面的主流。80 年代初，由美国 Xerox 公司 Alto 计算机首先使用的 Smalltalk－80 程序设计开发环境，以及后来的 Lisa、Macintosh 等计算机，将用户界面推向图形用户界面的新阶段。随之而来的用户界面管理系统和智能界面的研究均推动了用户界面的发展。用户界面已经从过去的人去适应笨拙的计算机，发展到今天的计算机不断地适应人的需求。用户界面的重要性在于它极大地影响了最终用户的使用，促进了计算机的推广应用，甚至影响了人们的工作和生活[6]。由于开发用户界面的工作量极大，加上不同用户对界面的要求也不尽相同，因此，用户界面已成为计算机软件研制中最困难的部分之一。当前，Internet 的发展异常迅猛，虚拟现实、可视化及多媒体技术等对用户界面提出了更高的要求。

2.3.2 图形用户界面(GUI)的结构模型及特征描述

图形用户界面(GUI)的广泛流行是当今计算机技术的重大成就之一，它极大地方便了非专业用户的使用，人们不再需要死记硬背大量的命令，而可以通过窗口、菜单方便地进行操作。

一个图形用户界面系统通常由三个基本层次组成。它们是显示模型，窗口模型和用户模型。图 2-1 给出了图形用户界面系统的层次结构。

图 2-1 中的最底层是嵌入式硬件平台。硬件平台的上面是操作系统。操作系统之上是图形用户界面的显示模型，它决定了图形在屏幕上的基本显示方式。不同的图形用户界面系统所采用的显示模型各不相同。例如大多数在 UNIX 上运行的图形用户界面系统都采用 X 窗口作显示模型；MS Windows 则采用 Microsoft 公司自己设计的图形设备接口(GDI)作显示模型。显示模型之上是图形用户界面系统的窗口模型。窗口模型确定窗口如何在屏幕上显示，如何改变大小，如何移动，及窗口的层次关系等。因为 X 窗口不但规定了如何显示基本图形对象，也规定了如何显示窗口，所以它不但可以充当图形用户界面的显示模型，也可以充当它的窗口模型。窗口模型之上是用户模型，图形用户界面的用户模型又称为图

形用户界面的视感。它包括两个部分：一是构造用户界面的工具；二是对于如何在屏幕上组织各种图形对象，以及这些对象之间如何进行交互的说明。图形用户界面系统的应用程序接口由其显示模型、窗口模型和用户模型的应用程序接口共同组成。

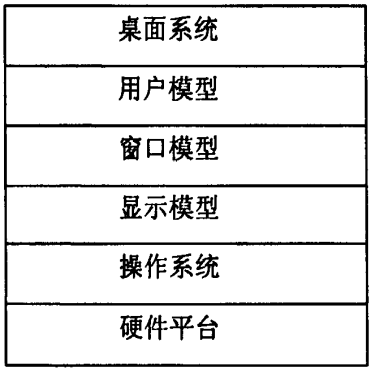

图 2-1 图形用户界面系统的层次结构

Fig. 2-1 structure of GUI system

分层体系结构便于抽象、层次结构清晰、层之间的接口功能定义明确，而且在层之间可以很方便的插入新的层，很容易增强系统的功能。采用分层体系，每一层只需要关心与之相邻的上下两层之间的功能定义和接口，容易设计和实现，也便于对单层的测试，每一层的可靠性可以大大提高。而层之间的接口定义相对简单、方便，大大方便了系统集成和测试，增强整个系统的可靠性和稳定性。采用分层体系结构设计，在不修改层之间的接口定义的情况下，可以很容易的替换和改写其中的一层或者多层，可以方便的增强和改写系统的功能。

GUI 的基本特性主要有以下几点。

- 直接操作

GUI 采用的是位映像图形显示技术，用户对应用程序的控制主要通过操纵显示在屏幕上的图形对象来完成，这些图形对象(如窗口、菜单、按钮等)都是在软件的控制下由位映像图形(即点阵图形)来实现的。

- 用户控制

应用程序的运行不再由编程安排耗时的过程来驱动，而是由用户通过 GUI 引入的输入设备来移动光标或点击图形对象，实现对应用程序的直接操纵，这是一种消息——事件驱动方式，它体现了人在控制应用程序运行中的中心地位。

- 界面定制

GUI 允许用户根据需要对应用程序的界面进行剪裁和定制，如移动、缩小或放大窗口、设置颜色等。

- 界面一致

GUI 系统作为一个完整的运行环境，应该提供一个不依赖于具体问题的界面设计标准，使得在不同环境下运行的各种应用程序的界面风格及与用户交互的方式都具有良好的一致性。

2.3.3 GUI 在嵌入式系统中的地位及嵌入式系统对 GUI 的要求

从用户的观点来看，GUI 是系统中最至关重要的一个方面：用户通过 GUI 与系统进行交互，所以 GUI 应该易于使用并且非常可靠，而且它还需要有内存意识，可以在内存受限的、微型嵌入式设备上运行。从二次开发者的角度看，GUI 是一个友好的开发环境，开发者无需经过艰苦的学习就能适应开发过程，这样才能使得基于此平台的应用很快地丰富起来。对于二次开发商而言，也才有兴趣使用此产品为终端产品制造商提供解决方案。另外，必须清楚的是，嵌入式系统往往是一种定制设备，它们对 GUI 的需求也各不相同。因此，GUI 也必须是可定制的。从系统的体系结构来看,GUI 系统属于应用层的软件系统，但通常而言，GUI 有别于一个简单的图形库，一个 GUI 系统通常会有自己的应用开发模式，从这个意义上讲，GUI 应该属于中间件的范畴[8]。GUI 在整个系统中所处的位置如图 2-2 所示。

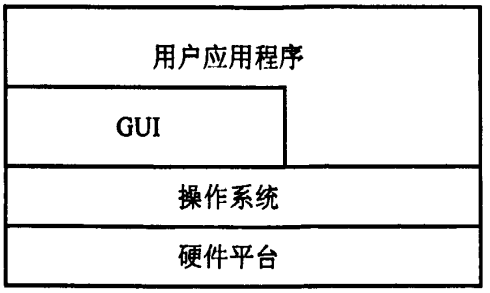

图 2.2 GUI 在系统中所处的位置

Fig. 2-2 situation of GUI in system

由于在过去 10 几年中，桌面操作模式取得了巨大成功，于是许多开发者在嵌入式设计中开始使用类似于桌面的操作系统和图形用户接口。在嵌入式环境下，GUI 系统的整体构架跟我们熟悉的个人电脑相去不远，例如绘图函数库、字

体、事件处理等也都是嵌入式 GUI 系统所要面临的。但是嵌入式系统本身由于体积小、资源少的特点，所以在整体设计上必须更为严谨，必须考虑更多的条件。为了满足嵌入式系统的不同要求，嵌入式系统对 GUI 的一般要求为：体积小巧、高性能、高可靠性、可定制、界面友好以及能提供完整的 API 支持。

2.3.4 嵌入式 GUI 的主要技术

嵌入式 GUI 涉及的主要技术有消息机制和事件驱动、屏幕管理技术、资源管理技术和设备管理。其中消息机制和事件驱动负责 GUI 与操作系统、输入设备以及其它系统等进行信息交换，也用于处理 GUI 系统内部的各种事件，它是整个 GUI 的动力系统，驱动着 GUI 系统的不停运行，是 GUI 系统的真正核心。屏幕管理技术是 GUI 的关键技术，它关系到 GUI 中的'G' (Graphical 表现方式)、'U' (User 与用户交互)和'I'(Interface 是与用户交互的接口界面)，它负责 GUI 系统的 Icon、Menu 等各种组件和元素的外观、组织方式和表现形式。用户可以通过操作和控制 GUI 的组件和元素，使用 GUI 系统。GUI 系统的资源多种多样，如字库、Icons，对内存等资源需求较大，因此需要进行有效的管理和统一调配。

- 消息机制和事件驱动

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。因为中断事件的发生是不可预期的、“突发”性的，当有多个应用等待并处理中断事件时，就会出现问题。消息机制可以很好的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办法。目前的大多数操作系统都采用了消息机制，基本上形成了一种应用设计的结构风格。

系统为不同的消息建立不同的消息过滤器。硬件产生的消息被送往消息过滤器，经过过滤处理后的消息被送往消息队列中，等待处理程序处理。消息处理机制具有以下特性。

1. 消息反映的是系统硬件事件的发生，因此与事件有相同的“突发”性，满足了对中断事件的处理需要。

2. 消息到达应用之前需要进行过滤和排队等待处理，这样为扩展对消息的处理特性和集中管理消息提供了环境，但是同时也降低了消息响应的实时性。这一点对 GUI 这种对实时性要求不是特别高的系统而言是完全可以满足的。

3. 应用对于消息的响应不是在中断服务程序中直接进行的，这就为灵活设计甚至动态调整事件和应用响应自己的关系创造了条件，也为抽象和分层提供了条件和可能。

消息机制包括消息的产生、传递(发送和接收)、管理、分派和处理等过程。消息作为一种系统之间、部件之间以及系统与部件之间的信息和数据联系，它并不一定由硬件事件产生，完全可以由应用程序、其它系统和其它部件产生，并将它们发到其它应用或者系统去，以激活和控制系统的行为。

消息在传递过程中需要得到过滤器的处理，然后进入消息队列。一般来说，消息过滤器都是安装在应用端的，应用开发者可以根据自己的要求修改或者重写过滤器，从而改变应用对事件(消息)的响应。消息的传递可以采用点对点方式(Point-to-Point)和广播方式(Broadcasting)。

消息机制本身是一种异步机制，消息发生后一般不会得到立即响应，因此需要特定的机构对消息进行管理。队列的先进先出特性比较适合对消息进行管理，因此我们采用消息队列进行消息管理，主要工作包括包括消息入队、消息缓存、消息分发等。消息管理和消息分发是消息机制的核心。

采用消息机制后，部件之间形成完全松散的耦合关系，增加了系统部件连接和集成的灵活性，这是通过消息和响应的非直接、隐式调用实现的。消息机制和隐式调用的优点是对软件复用具有强有力的支持，这也正是面向对象编程思想的初衷。选用消息机制、面向对象技术作为嵌入式 GUI 系统的设计，正好也适合了嵌入式系统本身的特点。嵌入式 GUI 本身只需要维持一个消息处理机，其它的部件可以配置、裁剪，也可以大大增加系统的可移植性。采用消息机制，任何部件只要定义了消息和响应过程之间的连接关系，都可以不加限制地加入到系统中和谐地工作，因此用户关心的只是产生消息和处理消息。

- 屏幕管理技术

随着嵌入式系统的发展和应用的要求，嵌入式 GUI 也由单窗口系统向多窗口系统发展。单窗口系统和多窗口系统的最大的差别在于对窗口单元的管理问题，单窗口系统几乎不需要窗口管理系统，只是在需要进行窗口切换时将另一个窗口显示出来。但是多窗口系统要求能够显示丰富的信息，而且可以进行方便、快速的切换，它在很大程度上依赖于窗口管理系统，又称为屏幕管理。任何显示

设备的显示区域总是有限的，而且在一个区域同时只能显示有限的信息。这对多窗口系统是远远不够的，这就对窗口系统本身的设计和实现提出了较高的要求，需要强大的屏幕管理功能。

多窗口系统中，我们定义每个可见的、具有独立功能的矩形区域为一个窗口单元。多窗口系统由若干窗口单元组成，屏幕管理就是对窗口单元和窗口单元之间的相互关系进行管理和处理。这些窗口单元之间的相互关系主要有：窗口单元之间的相互位置关系，窗口单元之间的显示顺序关系，以及当窗口单元的位置信息和显示顺序发生变化时窗口单元之间的变化关系等。

窗口单元间的关系可以分为两种:平面关系和立体关系。窗口单元间的平面关系是指任何时候平行于显示屏幕的切面上窗口单元之间的状态和关系，它们是平面的、静止的，这种关系可以通过坐标系统进行表示。但是多个窗口位于同一个屏幕时，窗口单元之间将会相互覆盖，这种关系是立体的，只有位于立体上面的窗口单元才会被显示出来。这种关系称为窗口单元 Z 序。

- 资源管理技术

一个完善的窗口系统除了窗口单元之外，还需要大量的资源支持，如显示字库、图标 Icon、位图资源 Bitmap、输入法和码表，以及其它一些相关资源。这些资源是凌乱的，每种资源之间没有太大的共性，对他们的管理将会决定窗口系统使用的方便性和完善性。因此资源管理对 GUI 系统也是非常重要的。

2.3.5 嵌入式 Linux 下 GUI 的实现方法

尽管 GUI 在嵌入式系统中有着广泛的应用，但嵌入式 GUI 的实现方法各有不同。

- 极少数大型厂商有能力自己开发满足自身功能需要的 GUI 系统。
- 某些厂商没有将 GUI 作为一个软件层从应用程序中剥离，GUI 的支持逻辑由应用程序自己负责。
- 还有些厂商采用某些比较成熟的 GUI 系统。

在上述手段中，第一种方法往往比较费时费力，同时可移植性较差。而第二种方法是一种临时解决方案，利用这种手段编写的程序，无法将显示逻辑和数据处理逻辑划分开来，从而导致程序结构不好，不便于调试，并导致大量的代码重复。第三种方法将嵌入式 GUI 以应用编程接口库的形式出现，使其能很快地被

具有编程经验的人员所接受，同时还可以根据需求作适当的剪裁进而转变成专用嵌入式GUI，也可根据需求对原有的应用编程接口库进行扩充。GUI是一种类似于操作系统的基础软件，这种软件系统应该遵循一定的标准，并且应该是开放源码的自由软件，从而可以让开发者集中精力开发自己的应用程序。目前看来，在Linux上进行嵌入式开发有如下几种GUI系统可供选择：MicroWindows、OpenGUI、MiniGUI、Qt/Embedded。

一个能够移植到多种硬件平台上的嵌入式GUI系统，应该至少抽象出两类设备[10]：基于图形显示设备(如VGA卡)的图形抽象层GAL(Graphic Abstract Layer)和基于输入设备(如鼠标，键盘，触摸屏等)的输入抽象层IAL(Input Abstract Layer)。GAL层完成系统对具体的显示硬件设备的操作，极大程度上隐蔽各种不同硬件的技术实现细节，为程序开发人员提供统一的图形编程接口。IAL层则需要实现对于各类不同输入设备的控制操作，提供统一的调用接口。GAL层与IAL层的设计概念，可以极大程度的提高嵌入式GUI的可移植性。

2.4 几种常用的嵌入式GUI系统

2.4.1 MicroWindows

MicroWindows是嵌入式系统中广为使用的一种图形用户接口，这个项目的早期目标是在嵌入式Linux平台上提供和普通个人电脑上类似的图形用户界面。该项目源于NanoGUI，现在二者已经结合在一起发布。

MicroWindows提供了现代图形窗口系统的一些特性。它能直接在显示硬件上操作，不需要其他图形系统的支持。在Linux操作系统上，MicroWindows可以充分利用Linux提供的Framebuffer机制进行图形显示，也可以使用著名的SVGALib库进行图形显示。Framebuffer技术允许图形用户程序将图形控制器的显存作为Framebuffer，映射到用户进程的内存空间里面。这样用户就可以像使用存储设备一样使用显示设备了。当用户需要绘制图形时，只要将显示信息写入Framebuffer，图形控制器就能将要显示的信息显示在显示器上面。使用这项技术后，图形用户程序不需借助X Windows系统，就可以在不知道底层显示设备硬件特性的情况下，将图形显示出来。MicroWindows的一个主要目标就是运行在嵌入式Linux上。

MicroWindows 可移植性非常好，基本上用 C 语言实现，只有某些关键代码被用汇编重写以提高速度。作为个人电脑的 X Windows 替代品，MicroWindows 提供了和 X Windows 类似的功能，但是占用的内存要少得多。根据用户的配置不同，MicroWindows 占用的内存资源在 60KB-100KB 之间。

MicroWindows 是一个基于典型的客户/服务器体系结构的 GUI 系统。它使用了分层设计的方法，允许改变不同的层来适应不同的应用。最底层提供了键盘、鼠标或触摸屏的驱动程序，使程序能访问实际的硬件设备和其它用户定制设备。中间层是一个可移植的图形引擎，提供了绘制线条、区域填充、剪切以及使用颜色模型等方法。最高层分别提供兼容于 X Windows 和 Windows CE(win32 子集)的 API，可以方便地实现应用程序的移植[11]。

但是作为一个窗口系统，该项目提供的窗口处理功能还需要进一步完善，比如控件或构件的实现还很不完备，键盘和鼠标等的驱动还很不完善。另外，MicroWindows 的图形引擎也存在一些问题。首先，它没有任何硬件加速能力。其次，它的图形引擎中存在许多低效算法，同时未经任何优化。再次，它的代码质量较差。由于该项目缺少一个强有力的核心代码维护人员，因此代码质量参差不齐，这必将影响系统整体的稳定性。

2.4.2 OpenGUI

OpenGUI 在 Linux 系统上存在已经很长时间了。它最初的名字叫 FastGL，能支持多种操作系统平台，比如 MS-DOS、QNX 和 Linux 等，不过目前只支持 x86 硬件平台。OpenGUI 也分为三层。最底层是由汇编语言编写的快速图形引擎；中间层提供了图形绘制 API，包括线条、矩形、圆弧等，并且兼容于 Borland 的 BGI API。第三层用 C++编写，提供了完整的 GUI 对象集。

OpenGUI 提供了 2 维绘图原语、消息驱动的 API、BMP 文件格式的支持。OpenGUI 功能强大，使用方便。我们甚至可以实现 Borland BGI 风格的应用程序，或者是 Qt 风格的窗口。OpenGUI 支持鼠标和键盘事件，在 Linux 上，使用 Framebuffer 或者 SVGALib 实现绘图。在颜色模型方面，OpenGUI 已经支持 8，15，16 和 32 位模型，这基本上能够满足应用的要求[12]。

由于它的内核是基于汇编语言实现的并利用 MMX 指令进行了优化，OpenGUI 运行速度非常快。由于历史悠久，OpenGUI 非常稳定。但是，由于其

内核使用汇编语言实现，因此可移植性较差，目前的发展也基本停滞。

2.4.3 MiniGUI

由北京飞漫软件技术有限公司开发的MiniGUI，是面向实时嵌入式系统的轻量级图形用户界面支持系统。自1999年初遵循GPL条款1发布第一个版本以来，MiniGUI 已广泛应用于手持信息终端、机顶盒、工业控制系统及工业仪表、便携式多媒体播放机、查询终端等产品和领域。

在 MiniGUI 九年的发展过程中，有许多值得一提的技术创新点。正是由于这些技术上的创新，才使MiniGUI更加适合实时嵌入式系统，而且也使MiniGUI的灵活性非常好，可以应用在包括手持设备、机顶盒、游戏终端等在内的各种高端或低端的嵌入式系统当中。这些技术创新包括[13]。

- 图形和输入抽象层。图形和输入抽象层对顶层API基本没有影响，但大大方便了MiniGUI自身以及应用程序的移植、调试等工作。
- 多字体和多字符集支持。MiniGUI的字符集支持不同于通过UNICODE内码实现的传统多字符集支持，这种实现占用资源少，更加适合于嵌入式系统。
- 针对不同操作系统特点的运行模式。为了适合不同的操作系统环境，我们可将MiniGUI配置成MiniGUI-Threads、MiniGUI-Processes及MiniGUI-Standalone三种运行模式。

MiniGUI 是一个根据嵌入式系统应用特点量身定做的完整的图形支持系统[14]。作为操作系统和应用程序之间的中间件，MiniGUI将底层操作系统及硬件平台差别隐藏了起来，并对上层应用程序提供了一致的功能特性，这些功能特性主要包括。

- 跨操作系统支持，具体包括普通嵌入式 Linux/μClinux、VxWorks、eCos、μC/OS-II、pSOS、ThreadX、Nucleus、OSE。
- 内建资源支持。我们可以将MiniGUI所使用的资源，诸如位图、图标和字体等编译到函数库中，该特性可提高 MiniGUI 的初始化速度，并且非常适合eCos/μCOS-II/ThreadX等无文件系统支持的实时嵌入式操作系统。
- 提供丰富的常用控件类。
- 对话框和消息框支持。
- 界面皮肤支持。用户可通过皮肤支持获得华丽的图形界面。

- 支持低端显示设备(比如单色 LCD)和高端显示设备(8 位色及以上显示设备)。
- 提供有增强 GDI 函数，包括光栅操作、复杂区域处理、椭圆、圆弧、多边形以及区域填充等函数。
- 各种流行图像文件的支持，包括 Windows BMP、GIF、JPEG、PNG 等。
- 输入法支持，用于提供各种可能的输入形式。内建有适合 PC 平台的汉字(GB2312)输入法支持，包括内码、全拼、智能拼音、五笔及自然码等。
- 针对嵌入式系统的特殊支持，包括一般性的 I/O 流操作、字节序相关函数等。

总之，将现代窗口和图形技术带入到嵌入式设备的 MiniGUI，是一个非常适合于实时嵌入式设备的图形用户界面支持系统，它具有支持多种嵌入式操作系统、优秀的可移植性、可伸缩的系统架构、易于扩展、功能丰富、可灵活剪裁的突出优点。但同时，MiniGUI 也存在界面开发可视化程度不高以及在多窗口的支持上有一定的不足的缺点。

2.4.4 Qt/Embedded

Qt 是著名的 GUI 库开发商 TrollTech 发布的一个跨平台的 C++图形界面应用程序框架。它实际上是一个类库，里面包括了大量的可重用类，其中既有按钮、窗口等这些可见类，也有定时器这样的不可见类和一些抽象类。在代码设计上，Qt 巧妙地利用了 C++独有的机制，如继承、多态、模板等，具体实现非常灵活。Qt 完全面向对象，拥有良好的扩展性与稳定性，并支持模块化编程。它广泛应用于 Linux 系统上，人们所熟知的 Linux 的 KDE 桌面环境就是以 Qt 为基础。利用 Qt 编程，可以充分利用其高度面向对象和模块化的特征，从繁琐的 X 编程中解脱出来，专注于程序本身的内容，使 Linux 下窗口程序设计成为一件非常轻松的事情。Qt 中的各种窗口可以根据参数以几种不同的风格显示，例如 MS Windows 风格、KDE 风格等。几乎所有的控件都可以在 Qt 中找到相应的类[15]。

Qt/Embedded 在体系上采用客户/服务器结构，任何一个 Qt/Embedded 程序都可以作为系统中唯一的一个 GUI Server 存在。

关于对象间通信的问题，Qt 采取了一种被称作“Signal-Slot”的方式，这是 Qt 的重要特征之一。在 MS-Windows 中，程序通过消息队列和消息循环的方式进行消息的传递与事件的触发，而 Qt 的“Signal-Slot”机制采取了这样的方式：一个类可以定义多个 Signal 和多个 Slot，Signal 就好像是“事件”，而 Slot 则是

响应事件的“方法”，并且和一般的成员函数没有太大的区别。当需要实现它们之间的通信时，就将某个类的Slot与另一个类的Signal“连接”起来，从而实现“事件驱动”。信号与槽是一种强有力的对象间通信机制，它完全可以取代原始的回调和消息映射机制；信号与槽是迅速的、类型安全的、健壮的，是完全面向对象并用C++来实现的一种机制。

Qt具有如下一些突出的优点。

- 优良的跨平台特性，支持多种操作系统。
- 面向对象。Qt的良好封装机制使得Qt的模块化程度非常高，可重用性较好，对于用户开发来说是非常方便的。
- 具有丰富的API，包括多达250个以上的C++类。
- 支持2D/3D图形渲染，支持OpenGL和XML。
- 拥有大量的开发文档。

Qt/Embedded是Qt的嵌入式版本，它针对嵌入式系统对Qt做了重要修改，使其图形引擎基于Linux下的FrameBufer而不是X11，它自己管理图形区域而不是由X11 Server来管理。因为Qt是KDE等项目使用的GUI支持库，所以有许多基于Qt的X Window程序可以非常方便地移植到Qt/Embedded版本上。因此，自从Qt/Embedded以GPL条款形式发布以来，就有大量的嵌入式Linux开发者转到了Qt/Embedded系统上。

第3章 Qt体系结构及运行机制的研究

在使用Qt进行开发之前，我们有必要深入了解Qt的内部体系结构和运行机制，这是我们将其移植到硬件平台和利用其 API 进行应用程序开发的基础。本章中我们通过对Qt源代码的阅读和研究，从Qt的系统架构、窗口系统、事件处理、信号与槽机制以及其线程机制等方面对其进行深入的剖析。

3.1 Qt系统架构的分析

Qt是一个跨平台的框架，它使用本地样式的API严格遵循每个支持平台中的用户界面的指导方针。Qt 绘制了所有控件，且编程人员可以通过重新实现虚函数的方式来扩展或自定义所有控件。Qt的窗体可精确模拟所支持平台的观感，使用这一功能，开发人员还可以生成自己的自定义样式，为其应用程序提供具有鲜明特色的外观。

Qt 在它所支持的不同平台中使用底层 API。这与传统的“分层”跨平台工具套件不同，传统工具套件是指在单个平台工具套件中使用的简单封装(例如，在Windows中使用MFC；在X11中使用Motif)。通常，分层工具套件速度较慢，其原因在于：库函数的每次调用都会产生许多要经过不同 API 层的附加调用。分层工具套件往往会受到基本工具套件的功能和行为的限制，导致应用程序中出现隐性错误。

Qt非常专业地支持各种平台，并且可以充分利用各种平台的优点：Microsoft Windows、X11、Mac OS X 和 Embedded Linux。使用单个源代码树，Qt应用程序可以编译成每个目标平台的可执行程序。尽管Qt是一个跨平台的框架，但与许多平台特定的工具套件相比，Qt更易于学习，效率更高。

Qt的功能是建立在它所支持平台的底层API之上的。这使Qt灵活、高效，使Qt应用程序可与单平台的应用程序配套。其架构概览如图3-1所示。

Qt 定义了多个模块，每个模块包含相对独立的库文件并实现各自相应的功能。其主要模块有：

- QtCore，Qt的基本模块，定义了其他模块使用的Qt核心的非GUI类，所有其他模块都依赖于该模块。

<table>
<tr><td colspan="4">用户应用程序</td></tr>
<tr><td colspan="4">Qt API</td></tr>
<tr><td>Qt/Windows</td><td>Qt/X11</td><td>Qt/Macintosh</td><td>Qt/Embedded</td></tr>
<tr><td>GDI</td><td>Xlib</td><td>Carbon</td><td rowspan="2">Embedded Linux</td></tr>
<tr><td>Windows</td><td>Unix/Linux</td><td>Mac OS X</td></tr>
</table>

图 3-1 Qt 架构概览

Fig. 3-1 architecture of Qt

- QtGui。定义了图形用户界面类，包括常用的窗体、文本框、菜单、按钮等。
- QtNetwork。定义了 Qt 的网络编程类。
- QtOpenGL。定义了 Qt 的 OpenGL 支持类。
- QtSql。定义了访问数据库的类。
- QtXml。定义了处理 XML 语言的类。
- 其他辅助模块。例如 QtDesigner、QtAssistant 等。

3.2 Qt/Embedded 窗口系统的分析

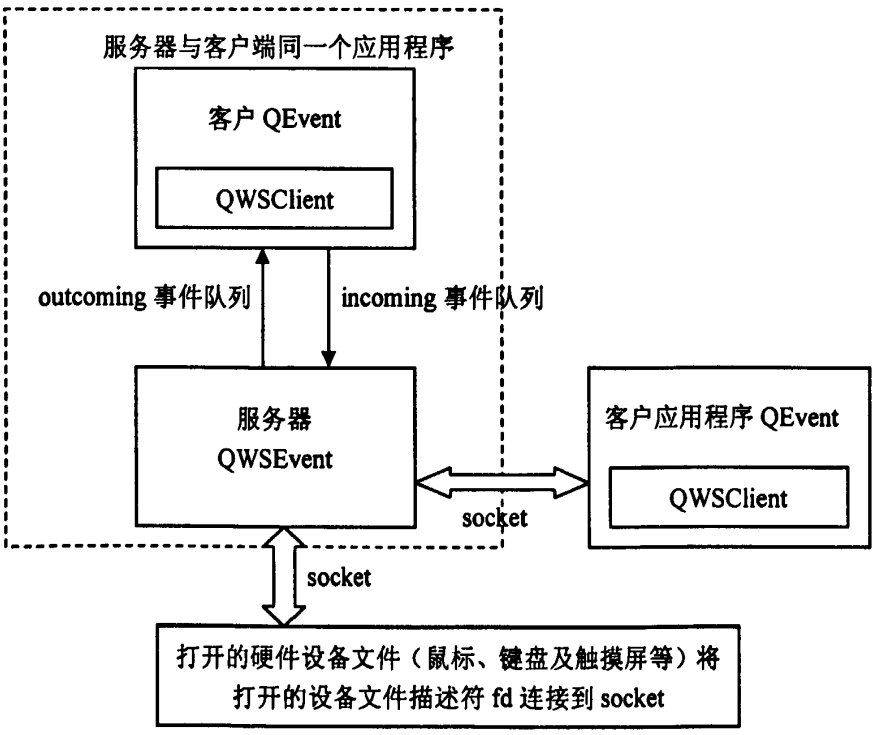

图 3-2 Qt/Embedded 窗口系统结构

Fig. 3-2 structure of window system in Qt/Embedded

Qt/Embedded的窗口系统采用了一种客户/服务器的体系结构，如图3-2所示。Qt/Embedded 的客户/服务器模型被封装在 Qt/Embedded 核心中，通过QApplication 建立模型的各种关系，用 Qt/Embedded 类库完成与模型的各种通信及操作。一个典型的 Qt/Embedded 窗口系统一般包括：一个服务器进程、一个或多个客户进程。服务器负责为客户进程和其本身分配显示区域、生成鼠标和键盘事件，它通常包含那些启动客户进程的用户界面。而客户进程则通过与服务器进行通信来申请显示区域，接收鼠标和键盘事件。客户进程可以直接访问所分配的显示区域，以便为用户提供 GUI 服务。服务器进程和客户进程通过共享内存的方式来传递所分配显示区域上的信息。

3.2.1 服务器端

服务器维护着一组区域，对所有的窗口进行全局管理。当窗体被创建、移动、改变大小和破坏时，通过这组区域来响应每个客户的申请。该区域存放在共享内存中，在执行绘图操作时，客户进程可以从中读取信息。服务器连接着一些系统设备，如鼠标、键盘和触摸屏，服务器负责将这些设备所产生的事件发送到适当的客户进程。键盘事件由服务器进程来维护，它是一个设备独立的事件，通常使用 Unicode 码和固定的键编码(keycodes)。在有些嵌入式设备中，由于没有物理键盘，服务器通过提供手写识别或虚拟键盘功能，用户可以利用触笔进行交互。服务器进程之所以能够实现诸如虚拟键盘这样的用户界面元素，是因为服务器进程本身是一个主客户进程(master client)，因此它能够像客户进程一样地使用各种 GUI 功能。任何一个 Qt/Embedded 应用程序都可以作为主进程运行，只要它在启动时设置一个标志即可，但是只允许有一个服务器进程存在。

3.2.2 客户端

Qt/Embedded 为我们提供的 API 与标准的 Qt API 是一致的。当我们在 Qt/Embedded 下使用其 API 画线时，Qt/Embedded 库能直接访问显存，完成画线工作。而当我们在 Qt/X11 下画线时，Qt/X11 库向 X 服务器发送一个消息，由 X 服务器去做实际的画线工作。同样，当我们在 Qt/Windows 环境下画线时，Qt/Windows 库调用 Win32 操作系统来完成画线任务。在这几种平台的实例中，应用程序使用相同的代码，即调用了标准 Qt API 中相同的画线函数。

Qt/Embedded的客户端在一些情况下需要与服务器进程建立连接。比如，在客户进程启动时；发生了会影响到全局后果的操作而与服务器通信时，例如由于窗口覆盖而导致显示区域的变化，即改变了其他客户端正在使用的区域；使用了全局剪贴板或拖放操作时；从用户那里接收到鼠标和键盘事件时等等。对于多进程来说，客户端将一些需要服务器处理的窗体管理及鼠标键盘命令数据通过socket以事件形式发送给服务器。同时，客户端还通过socket接收来自服务器的事件，分发给客户端相关的窗口进行处理。对于单个进程来说，因为在同一个进程空间当中，所以事件只需要通过一个队列结构即可传递。

值得注意的是：当执行绘图操作时客户端不必与服务器进行交互。因为服务器为客户端分配了一块能自由读写的显示区域，客户端用它来显示各种用户界面控件、视频剪辑及图像。客户端绘图函数建立在一个容易支持硬件加速操作的体系结构上，一些基本的绘图函数能够直接在Linux的framebuffer上进行操作。

客户端负责处理所有的绘画操作，包括文本显示和字体处理。另外，它还处理那些定制的窗口装饰(如标题条等)。

3.3 信号与槽机制的研究

信号与槽机制是Qt的核心机制。信号与槽是一种高级接口，应用于对象之间的通信，它是Qt的核心特性，也是Qt区别于其它工具包的重要地方。信号与槽是Qt自行定义的一种通信机制，它独立于标准的C/C++语言，因此要正确的处理信号和槽，必须借助一个称为元对象编译器(moc)的Qt工具，该工具是一个C++预处理程序，它为高层次的事件处理自动生成所需要的附加代码。

3.3.1 信号与槽的基本原理

在GUI用户界面中，当用户操作一个窗口部件时，需要其他窗口部件的响应或者能够激活其他的操作。在程序开发中，经常使用回调机制来实现。所谓回调，就是事先将一个回调函数指针传递给某个处理过程，当这个处理过程得到执行时，回调预先定义好的回调函数以实现激活其他处理过程的目的。

不同于回调函数机制，Qt提供了信号与槽机制。信号是一个特定的标识，一个槽就是一个函数，与一般函数不同，槽函数既能够和信号关联，也能象普通函数一样直接被调用。当某个事件出现时，通过发送信号，可以将与之关联的槽

函数激活。在程序中，使用 QObject::connect()函数来将某个信号与某个槽进行关联。

一个类想要支持信号和槽，必须从 QObject 或其子类继承。

信号与槽的关联关系可以有以下几种模式：一个信号和一个槽关联；一个信号和多个槽关联；多个信号和一个槽关联。

一个信号和多个槽关联的情况下，当发出该信号时，与该信号关联的各个槽以任意的先后顺序立即执行(即槽函数的执行顺序是随机的，与槽关联的顺序没有关系)。信号与槽机制是完全独立于 GUI 事件循环的。

此外，信号还可以和信号进行关联。当两个信号关联时，第一个信号的发出将会激活 Qt 发送第二个信号。信号与槽机制的关联关系如图 3-3 所示。

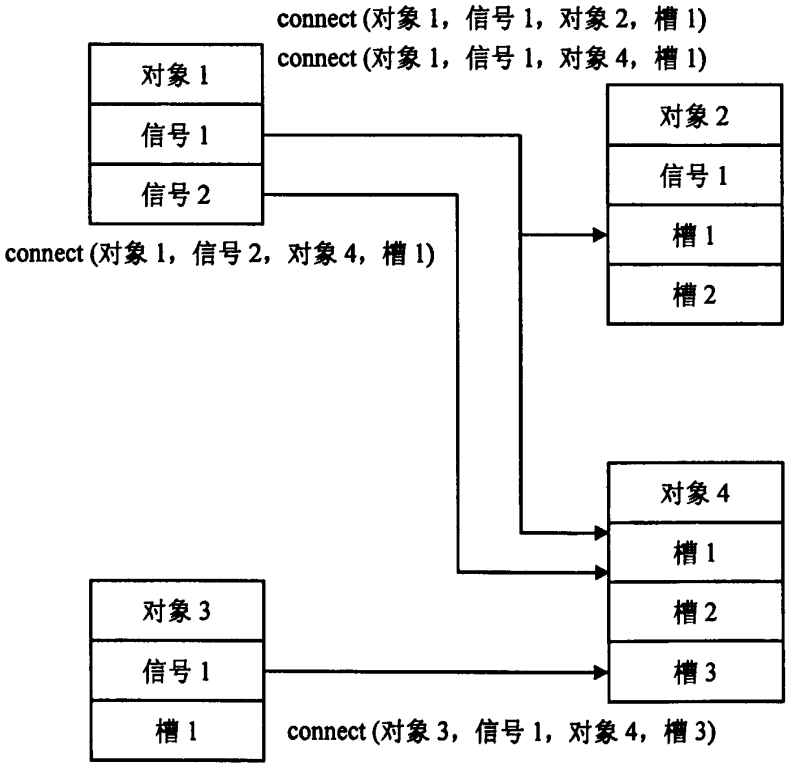

图 3-3 信号与槽的关联

Fig. 3-3 connection of signals and slots

3.3.2 信号与槽机制的优点

Qt 的信号与槽机制区别于回调函数，具有以下一些优点。

- 类型安全的

需要关联的信号和槽的签名必须是等同的，即信号的参数类型和参数个数同接受该信号的槽的参数类型和参数个数相同；不过，一个槽的参数个数可以少于信号的参数个数，但缺少的参数必须是信号参数的最后一个或几个参数。如果信号和槽的签名不符，编译器就会报错。

- 松散耦合的

Qt 信号与槽机制减弱了 Qt 对象的耦合度。激发信号的 Qt 对象无须知道是哪个对象的哪个槽需要接收它发出的信号，而只需在适当的时候发送信号，它不需要知道也不必关心它的信号有没有被接收到，更不需要知道是哪个对象的哪个槽接收到了信号。同样，对象的槽也不知道是哪些信号关联到了自己。而一旦关联信号和槽，Qt 就保证了适合的槽得到调用。即使关联的对象在运行时被删除，应用程序也不会出现崩溃。

3.3.3 信号与槽机制的效率

信号与槽机制增强了对象间通信的灵活性，然而这也损失了一些性能。同回调函数相比较，信号与槽机制是有些慢的。造成这一现象的原因主要有：需要定位接收信号的对象；需要安全的遍历所有的关联；需要编组/解组所传递的参数；多线程的时候，信号可能需要排队等待。

然而，与创建堆对象的 new 操作以及删除堆对象的 delete 操作相比较，信号和槽的代价只相当于它们很少的一部分。信号与槽机制导致的这点性能损耗，对实时应用程序来说是可以忽略的。同信号与槽所提供的灵活性和简便性相比，这点性能的损失也是值得的。

3.4 Qt 的事件处理

Qt 的事件由窗口系统或 Qt 自身产生，用以响应各种行为或情况。如当按下或释放键盘时，会产生键盘事件；当某个窗口第一次显示时，会产生一个 paintEvent 事件用来告知窗口绘制自身，从而使得该窗口可见。大多数事件是作为用户行为的响应而产生的，但也有例外，比如定时器、网络数据之类的事件则是由系统自身产生的。

3.4.1 Qt 的事件机制

Qt 中定义的事件是一个从 QEvent 类继承而来的对象，它表示应用程序内部或者外部发生了某些应用程序自身必须知道的事情。任何从 QObject 类派生的对象均可以通过 QObject::event()方法接收事件。事件产生时，Qt 会创建一个合适的 QEvent 对象或其子对象，然后通过调用 QObject 类的 event()函数将这个事件对象传给特定的 QObject 对象或其子对象。event()函数自身并不处理事件，而是根据事件类型调用相应的事件处理器，其返回值告知这个事件是否被接收并进行了处理。

在 Qt 内部，Qt 通过函数 QApplication::exec()启动主事件循环不停抓取事件队列中的本地窗口事件，然后将它们转换成对应的 QEvent 对象，并最终将这些 QEvent 对象发送到目的 QObject 对象来完成上述一系列事件的处理动作。

事件和信号是两个不同的概念，事件比信号更底层。通常，在使用已有的窗口部件时，信号是十分有用的，而在自定义新的窗口部件时，事件则更为重要。

事件的来源多种多样，最常见的有来自窗口系统的鼠标和键盘事件，以及来自系统的定时器事件，还有一些事件则来自应用程序。

3.4.2 Qt 的事件处理方法

Qt 提供了以下 5 种处理事件的方式。

- 重新实现特定的事件处理器。

  重新实现一些 Qt 已事先定义的事件处理器是进行事件处理的最简单方式。

- 重新实现 QObject::event()函数。

  通过重新实现 event()函数，可以在事件到达特定的事件处理器之前截获并处理它们。这种方法可以用来覆盖已定义事件的默认处理方式，也可以用来处理 Qt 中尚未定义特定事件处理器的事件。

- 在 QObject 中注册事件过滤器。

  如果对象使用 installEventFilter()函数注册了事件过滤器，目标对象中的所有事件将首先发给这个监视对象的 eventFilter()函数。

- 在 QApplication 中注册事件过滤器。

  如果一个事件过滤器被注册到程序中唯一的 QApplication 对象，应用程序中

所有对象里的每一个事件都会在他们被送达其他事件过滤器之前，首先到达这个eventFilter()函数。

继承QApplication并重新实现notify()函数。

Qt调用QApplication来发送一个事件。重新实现notify()函数是在事件过滤器得到所有的事件之前获得他们的唯一方法。但事件过滤器的使用更为方便，因为可以同时有多个事件过滤器，而notify()函数只有一个。

包括鼠标和键盘事件在内的很多事件都可以被传递。如果事件到达其目标对象之前没有被截获处理，或者没有被它的目标对象处理，那么此时目标对象的父对象将变成新的目标对象，整个事件处理过程将重复进行，直到该事件被处理或者到达最顶层对象。

3.4.3 Qt的事件处理器

通常事件传递后将调用相应的虚函数，即事件处理器，这个虚函数将作为本事件的处理器作出适当响应。

如果用户不想重新实现这些事件处理器，可以调用Qt中其父类已有的实现；如果用户希望替换原有基类的功能，则必须重新实现全部相关功能。但是，前一种方式的适应性不强，无法满足实际的编程需求。后一种方法又往往比较繁琐复杂，因此最好的方法是对原有基类的事件处理函数进行扩充，即在已有的事件处理器虚函数中补充相关功能的实现，不需要改动的地方则可以直接调用基类中默认的实现。

3.5 Qt的线程机制

在具有图形用户界面的Qt应用程序中，主线程由GUI线程充当，该线程是Qt中唯一可以执行GUI相关操作的线程。但是，它可以同时拥有一个或者多个非GUI线程作为工作线程来处理其他耗时操作。这样处理后，即使在负载很重的情况下，应用程序也可以保证图形用户界面的响应。

3.5.1 可重入与线程安全

在Qt文档中，可重入和线程安全均用于描述能用在多线程环境下的函数。如果一个函数能同时被多个线程调用，并且为每一个调用者提供一份单独的数

据，那么称这个函数是可重入的。如果一个函数能同时被多个线程调用，并且所有的调用者引用同一份数据，访问数据时串行处理，那么称这个函数是线程安全的。扩展来讲，如果一个类中的所有函数在这个类的各个实例中可以被多个线程同时调用，那么这个类是可重入的；同样，如果类的一个实例可以同时被多个线程操作，那么这个类是线程安全的。

对于 Qt 类，其是否是可重入的和是否是线程安全的可作如下概括：QObject 类是可重入的；大多数非 GUI 类也是可重入的，从而使得这些类可以同时用于多个线程。但是 GUI 类均是不可重入的，他们只能用于主线程。Qt 中线程安全的类包括 QThread、QMutex、QMutexLocker、QReadWriteLocker、QReadLocker、QWriteLocker、QSemaphore、QWaitConditon 和 QThreadStorage。此外，函数 QApplication::postEvent()、QApplication::removePostedEvent()和 QApplication::wakeup()也是线程安全的。但是 QObject 及其子类以及整个事件分发机制都不是线程安全的。

3.5.2 线程与事件循环

GUI 线程是唯一允许创建 QApplication 对象并且调用它的 exec()函数的线程。在调用了 exec()函数后，GUI 线程要么等待一个事件，要么处理一个事件。每一个线程都可以有自己的事件循环，如图 3-4 所示。起始线程通过 QApplication::exec()函数启动事件循环，其他非 GUI 线程通过 QThread::exec()启动各自的事件循环。

线程中的事件循环机制使得在线程中使用某些需要事件机制的非 GUI 类成为可能，例如 QTimer 类、QTcpSocket 类和 QProcess 类等；它同样使处于不同线程内的对象可以进行信号和槽的连接。

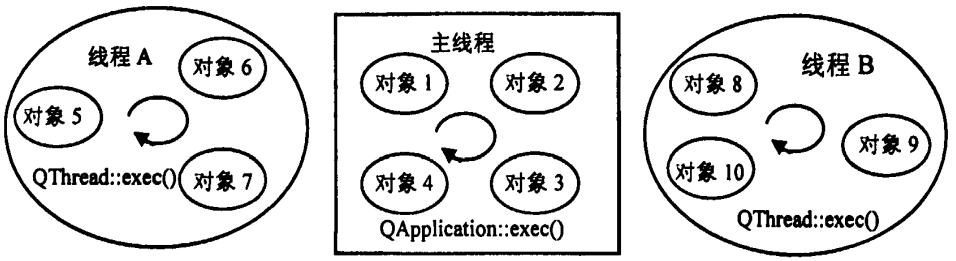

图 3-4 Qt 线程事件模型

Fig. 3-4 thread model in Qt

Qt 使用了一种称为隐式数据共享的技术来优化其值类，最典型的是 QString 和 QImage 类。Qt 使用的隐式共享技术的类可以和其他类一样在线程间拷贝，这些类完全是可重入的。实际上，Qt 使用了原子引用计数操作，这项技术速度非常快，它根据不同的平台由汇编语言实现。

3.5.3 线程和信号与槽机制

QThread 继承 QObject，它可以发出信号表示当前线程开始执行或退出，并且还提供了其他一些槽函数。十分重要的一点是 QObject 对象可以用于多线程，其发出的信号能够激活其他线程的槽函数，并且还可以向属于其他线程的对象发送事件。这些都是线程事件循环机制的功劳。信号与槽机制可以支持跨线程的连接，这大大方便了 GUI 线程和其他工作线程间的通信。目前 Qt 支持以下三种信号和槽的连接方式。

- 直接连接方式

这种连接方式下信号发出后相应的槽函数将立即被调用。这个槽函数在发出信号的线程中执行，而不一定必须在接收对象所属的线程中。

- 排队连接方式

这种连接方式下信号发出后需等到接收对象所属的线程的事件循环取得控制权时才调用响应的槽函数。这个槽函数在信号接收对象所属的同一个线程中进行。

- 自动连接方式

这种连接方式是 Qt 的默认连接方式，如果发出信号的对象与接收这个信号的对象同属一个线程，那么工作方式与直接连接方式相同，否则与排队连接方式相同。

所选择的连接类型可以通过参数传递给 connect()函数。需要注意的是，当信号的发送者和接收者分属不同的线程且接收者线程存在自身事件循环时，使用直接连接方式是不安全的。

在实际应用中，试图在非主线程中调用 GUI 类是不可能的，通常的解决方法是将那些耗时的操作放在独立的工作线程中，当这些工作线程得出结果后再将结果放到主线程中显示。跨线程的信号与槽机制打开了工作线程和 GUI 线程通信的方便之门。

第4章 变电站GUI系统总体设计

4.1 系统需求分析

4.1.1 变电站GUI系统的作用及功能要求

通信控制器是变电站自动化系统的信息中心，它通过不同的通信介质和通信规约，对变电站内各种设备的信息进行采集处理，形成标准的信息并通过数据通道传送到集控中心和电网自动化系统。变电站综合自动化系统中，信息的采集和传送是根据优先级进行划分的，这要求通信控制器具有实时、分时的特性。由于电力系统对系统可靠性的较高要求，系统一般采用双机冗余热备份，以保证其正常运行。

通信控制器的 GUI 系统即人机界面是进行人机交互的唯一手段，用户通过它可以完成查看系统信息、进行系统配置以及对远方设备进行控制等工作。一般说来，它的设计应该包含以下一些主要内容。

- 主接线图。主接线图以图形的方式直观的表示出变电站内部各主要元件(主变压器、断路器、电容、电抗、线路等)之间的电气连接关系，使用户可以了解整个变电站的拓扑结构概况。一般要求可以通过主接线图显示各断路器的分/合情况及关键元件的当前状态。
- 遥测、遥信量表。以表格的形式显示系统当前的各遥信、遥测量值，以供用户查看。
- 通信报文。可供用户查看系统各通信接口的报文信息。
- 事件记录。可供用户查看系统的报警信息和历史事件记录等，用于辅助变电站值班维护人员进行故障分析及判断。
- 系统调试功能。值班维护人员通过系统调试界面可以对远端设备进行遥控调试及系统复位等工作。
- 系统配置功能。通过系统配置界面可以完成串口参数配置、网络参数配置、CAN 网参数配置以及设置系统其他相关参数的工作。
- VQC(电压无功控制)功能。某些系统具有 VQC 功能，可以通过人机界面完成查看系统的九域图信息、查看当前状态信息、查看及修改定值设置、查看系统历史事件记录、进行系统调试等工作。

4.1.2 变电站 GUI 系统的实现要求

总的说来变电站 GUI 系统的实现有如下几个基本要求。

- 功能完善，系统精简

出于成本的考虑，变电站 GUI 系统所采用的硬件平台需要对其使用的 CPU、FLASH、RAM 等有一定的限制，这就需要 GUI 软件的实现尽可能少的占用系统的资源。同时，由于 GUI 系统要实现大量的图形、表格以及文字的显示和刷新，又需要 GUI 软件能够以较快的速度实现完善的图形、表格以及文字处理等功能。

- 具有较好的可移植性

由于变电站设备更新换代的要求，需要变电站 GUI 系统具有良好的可移植性，以便于日后对设备的升级。

- 菜单结构清晰，便于操作

应用软件的菜单是用户进行各种操作的最终对象，应使各功能菜单分类清晰，适应电力系统用户的使用习惯，便于查找和操作。

4.2 系统的总体结构

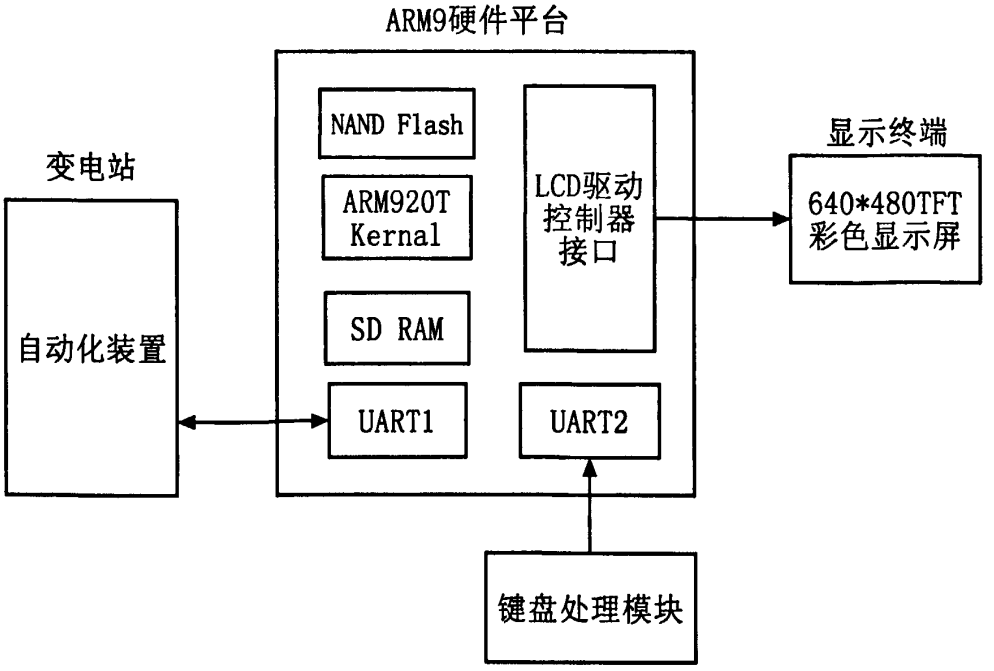

图 4-1 系统示意图

Fig. 4-1 diagrammatic sketch of system

由于变电站设备对于系统的实时性和多任务控制的较高要求，为避免 GUI 系统与设备主处理程序之间的相互影响和干扰，本方案采用一套独立的硬件平台配合 GUI 软件实现，通过串口与主处理程序进行信息交互，从而最大程度的保证设备的正常工作。系统从设备的主机侧获取数据，并依据用户的键盘操作将相应的数据显示到液晶显示器上。

整个系统包括硬件平台和软件系统两大部分，软件系统在硬件平台的支撑下实现系统的功能要求。该系统以嵌入式 Linux 为操作系统，采用 32 位的高性能嵌入式处理器 ARM9 系列的 S3C2410，使用 Qt/Embedded 作为图形接口，系统的总体示意图如图 4-1 所示。

4.3 系统的硬件设计

硬件平台采用的处理器为韩国三星公司的 ARM 芯片 S3C2410，外部扩展 64M 的 RAM 和 64M 的 FLASH 存储器，3 个串行接口，1 个 10M 的网口，以及其他外围接口电路。显示屏采用 640×480 的彩色液晶模块，由 S3C2410 本身所提供的 LCD 控制器进行控制[17]。如图 4-2 所示。

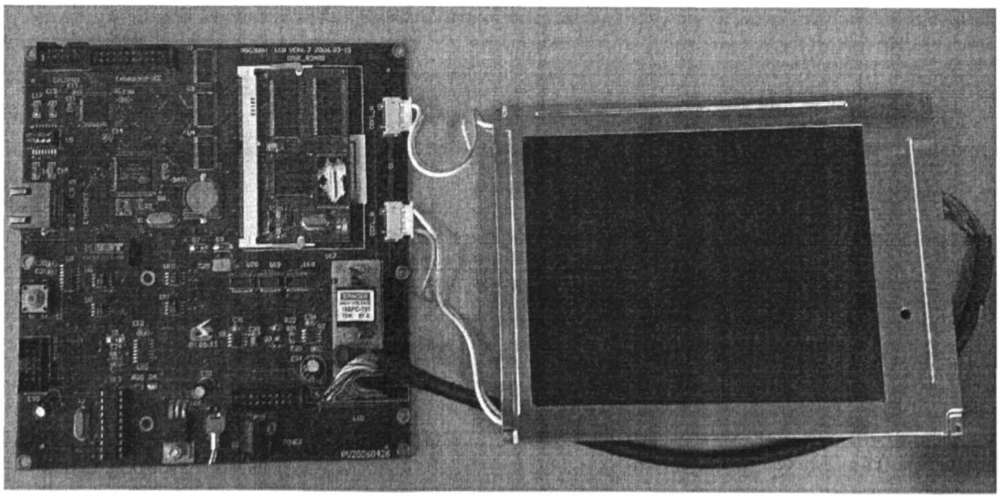

图 4-2 硬件实物图

Fig. 4-2 hardware platform

4.3.1 S3C2410 嵌入式处理器

S3C2410 是一款基于 ARM920T 核的 32 位嵌入式处理器，主要面向高性价比及低功耗的应用，它的工作频率是 203MHz。ARM920T 核由 ARM920TDMI、

内存管理单元(MMU)和高速缓存三部分组成。其中，内存管理单元可以管理虚拟内存，高速缓存由独立的 16K 高速 cache 组成。

S3C2410 的资源包括：16KB 数据 Cache，16KB 指令 Cache，MMU，外部存储器控制器；一个 LCD 控制器(支持黑白、灰度、Color STN、TFT 屏)以及触摸屏接口;NAND FLASH 控制器,SD/MMC 接口,4 个 DMA 通道;3 通道 UART，1 个多主 I2C 总线控制器，1 个 IIS 总线控制器；4 通道 PWM 定时器及一个内部定时器；117 个通用 I/O 口；24 个外部中断源；8 通道 10 位 ADC；实时时钟及看门狗定时器等。

4.3.2 S3C2410 外围硬件配置

- NAND FLASH 存储器

系统采用一片三星公司的 K9F1208 芯片作为 FLASH 存储器,其容量为 64M。FLASH 存储器的主要作用是存放系统的软件代码,包括 BootLoader、Linux 内核、文件系统、应用程序以及用户配置文件和图形文件等。

- SDRAM 存储器

系统采用两片 32M 的 SDRAM 组成 64M 的 SDRAM 存储器作为系统的内存使用。其主要作用是在系统启动后将运行代码拷贝到内存中，供程序的运行。

- 以太网控制芯片

外扩一片以太网控制芯片 CS8900，通过双绞线提供 10M 的以太网口。它的主要作用是进行程序的烧写以及系统的调试等。

- RS232 接口芯片

外扩 3 个 RS232 接口芯片。其一用于与键盘处理模块的通信，另外两个用于与通信控制器主机的数据通信。

4.4 软件总体结构设计

软件系统由以下几个部分组成。

- BootLoader。对开放源码的 U-boot 进行移植，主要作用是对硬件进行必要的初始化设置，为 Linux 的运行做好准备。
- 嵌入式 Linux 操作系统。对 Linux 内核进行在硬件平台上的移植，主要作用是对硬件进行配置，加载各外围设备的驱动程序，为应用程序的正常运行提供良

好的支持。

- 嵌入式 GUI。将 Qt/Embedded 移植到嵌入式 Linux 环境中，为应用程序的运行提供接口。
- 应用程序。本系统中即为通信控制器的人机界面，其主要作用是提供给用户使用，使用户可以查看或者修改系统参数、各状态量及进行各种控制操作等。

软件系统各组成部分之间的关系如图 4-3 所示。

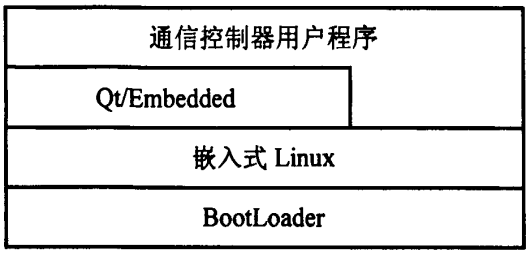

图 4-3 软件系统组成

Fig. 4-3 composition of software system

根据前面对嵌入式 Linux 下几种 GUI 系统的分析和其各自的特点的比较，结合本项目的具体需求，最终采用了 Qt/Embedded 完成 GUI 系统的开发，这主要是由于以下几个原因。首先，Qt/Embedded 是一个开放源码的项目，我们可以从网站上得到 Qt/Embedded 的全部源代码，这对于我们深入理解其工作原理，从而更好的使用它是很有帮助的。其次，Qt/Embedded 是一个非常流行的嵌入式 GUI 系统，得到了很多开发者的认同，因此可以方便的找到大量的参考资料，可以大大加快开发进度。最后，Qt/Embedded 是由 Qt 裁剪而来的，它与 Qt 之间保持统一的 API，桌面系统中的许多应用都可以方便的移植到 Qt/Embedded 中。

4.5 GUI 应用软件的设计

4.5.1 用户界面设计概述

在人和机器的交互过程中，有一个人和机器通信的接口，即我们所说的界面。从心理学意义来分，界面可分为感觉和情感两个层次。用户界面设计是屏幕产品的重要组成部分。界面设计是一个复杂的有不同学科参与的工程，包括认知心理学、设计学、语言学等学科都在里面扮演着重要的角色。用户界面设计有三大原则：置界面于用户的控制之下；减少用户的记忆负担；保持界面的一致性。

用户界面设计在工作流程上分为结构设计、交互设计、视觉设计三个部分。

- 结构设计

结构设计是界面设计的骨架。通过对用户研究和任务分析，制定出产品的整体架构。结构设计是用户界面设计的核心，确定用户界面的主用功能。

- 交互设计

交互设计的目的是使产品让用户能简单使用。任何产品功能的实现都是通过人和机器的交互来完成的。常用的交互设备有鼠标、键盘、手写笔、触摸屏、面板、遥控器等。

- 视觉设计

在结构设计的基础上，参照目标群体的心理模型和任务达成进行视觉设计。包括各种窗口、组件、文本、图形等。视觉设计要达到用户愉悦使用的目的。一般说来，好的用户界面设计需要遵循以下几个原则：

1. 简洁明了。对于用户来说界面相当重要的是简洁明了，用户马上就能够知道他们需要实现的功能。复杂的功能模块不应该错综复杂的罗列在界面上，以免混淆或分散用户对于主要功能实现的注力；

2. 使用方便。用户可以按照他们的理解自由支配操作方式，线程必须是灵活可变的，而不能规定用户只能这样不能那样，否则界面方案就失去了用户的理解和支持；

3. 尊重用户习惯。如同语言有习惯用语一样，界面也应与现实的习惯相贴近，与真实世界的知识相吻合；

4. 图标表示直观。设计元素或控件应当直观，符合用户的直觉，用户自然就能与软件交互；

5. 操作结果可预期和撤销。界面的设计能够让用户知道操作后的可预期结果以及可自由回溯撤消先前操作。设计者应当理解用户的任务、目标和模式，如此用户也不用担心他们的操作会带来不可逆转的后果；

6. 进度提示。让用户知道操作正在进行中或正在取得阶段性的成果，辅以进度条、消息状态栏的方式，这样用户能够随时掌控并作出正确的判断，不至于取消快要完成的计划或屡屡重试已在进行的进程；

7. 容错机制。允许用户有非常规操作，设置友善的出错提示，以避免系统

崩溃并纠正用户不良习惯；

8. 个性方案。好的界面设计应当提供给用户个性化的 GUI 方案，以符合不同的视觉品味、操作风格，界面自定义、操作自定义属于界面方案中的较高级别；

9. 亲和力强。设计效果需要在功能基础上，实现视觉上的突破，对于用户有亲和力、有吸引力，从而能够愉快地进行交互，并且印象深刻，同样功能的软件胜出的必是界面亲和力强的。

4.5.2 GUI 软件的功能及模块划分

根据 GUI 应用软件所需要提供的功能，系统可以划分为 3 个基本模块，分别为界面显示模块(即 GUI 模块)、数据通信模块和键盘处理模块。界面显示模块负责将得到的数据或者图形信息以文本、表格、图形等形式显示在屏幕上，同时负责处理用户的输入，对用户的操作作出响应。数据通信子系统负责 GUI 系统与通信控制器主机侧的数据通信，一方面可以接收需要显示的数据信息，另一方面也可以把用户更改的数据或者控制指令传送到主机侧。键盘处理模块负责接收串口发送过来的键盘编码，并将其转换为 Qt 的标准编码，之后通过 socket 传送给界面显示模块，从而相应用户的操作。由于数据通信模块需要与通信控制器主机侧交换大量的数据，为避免当其负载较高时对 GUI 模块造成影响而出现假死、不响应用户操作等意外情况的出现，我们将界面显示模块和数据通信模块各由一个单独的线程来实现。系统的模块结构图如图 4-4 所示。

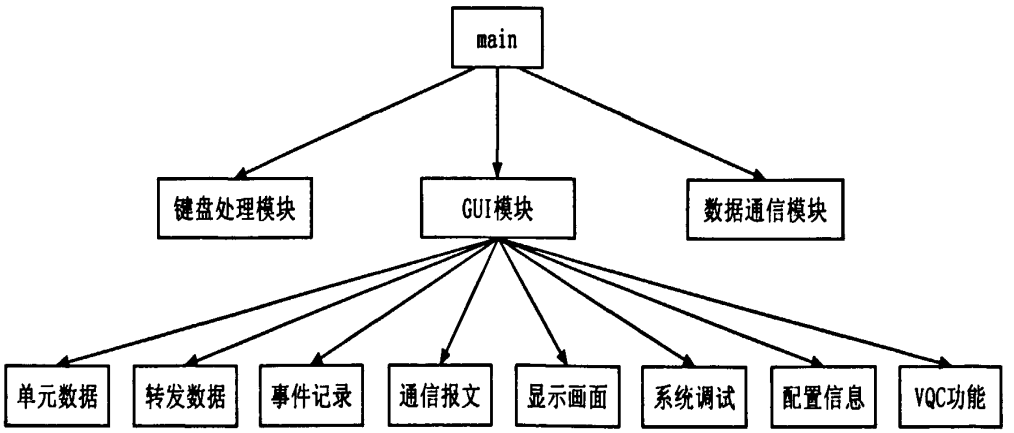

图 4-4 软件的模块划分

Fig. 4-4 partition of software

GUI 模块可以划分为 8 个子模块，每个子模块由完成同一类型的功能的一个

或多个最终功能模块组成，下面对其进行具体的划分和说明。

1. 单元数据子模块

单元数据子模块用于显示某一单元内的节点的数据信息，它包括以下最终功能模块：单元遥信模块、单元遥测模块、单元遥脉模块、单元 COS 模块以及单元 SOE 模块。

2. 转发数据子模块

转发数据子模块用于显示转发的节点数据信息，它包括转发遥信模块、转发遥测模块和转发遥脉模块。

3. 事件记录子模块

事件记录子模块用于显示系统的历史事件记录，它包括事件记录模块、遥控记录模块和自检信息模块。

4. 通信报文子模块

通信报文子模块用于显示系统各通信端口的报文信息。

5. 显示画面子模块

显示画面子模块用于显示系统的一些可以用图形来显示的信息，包括主接线图模块和工况图模块。

6. 系统调试子模块

系统调试子模块用于用户对远方装置进行控制，它包括设置遥信模块、设置遥测模块、设置电度模块、遥控操作模块、时间设置模块及总控复位操作模块。

7. 配置信息子模块

配置信息子模块用于完成用户对系统的配置，它包括系统配置模块、介质配置模块和节点配置模块。

8. VQC 功能(电压无功控制)子模块

VQC 功能子模块用于要求有电压无功控制功能的系统中，通过它可以查看系统的 VQC 功能的相关数据信息，它包括九域图模块以及定值修改模块等。

将系统的功能划分为各个小的模块就方面我们后面对其具体的功能进行设计了。

4.5.3 GUI 软件数据流分析

系统的数据来源主要有：通信控制器主机、用户键盘操作、系统的配置文件

及主接线图文件。系统通过数据通信线程与通信控制器主机进行通信，获取各节点的实时数据信息或通信控制器系统中存放的其他信息，获取的信息通过共享内存与 GUI 线程共享数据；用户通过键盘的操作进行界面的切换，执行定值修改、遥控操作等向 GUI 线程传递数据；GUI 线程在必要时还会读取配置文件以及主接线图文件获取相应的数据。

系统的数据流如图 4-5 所示。

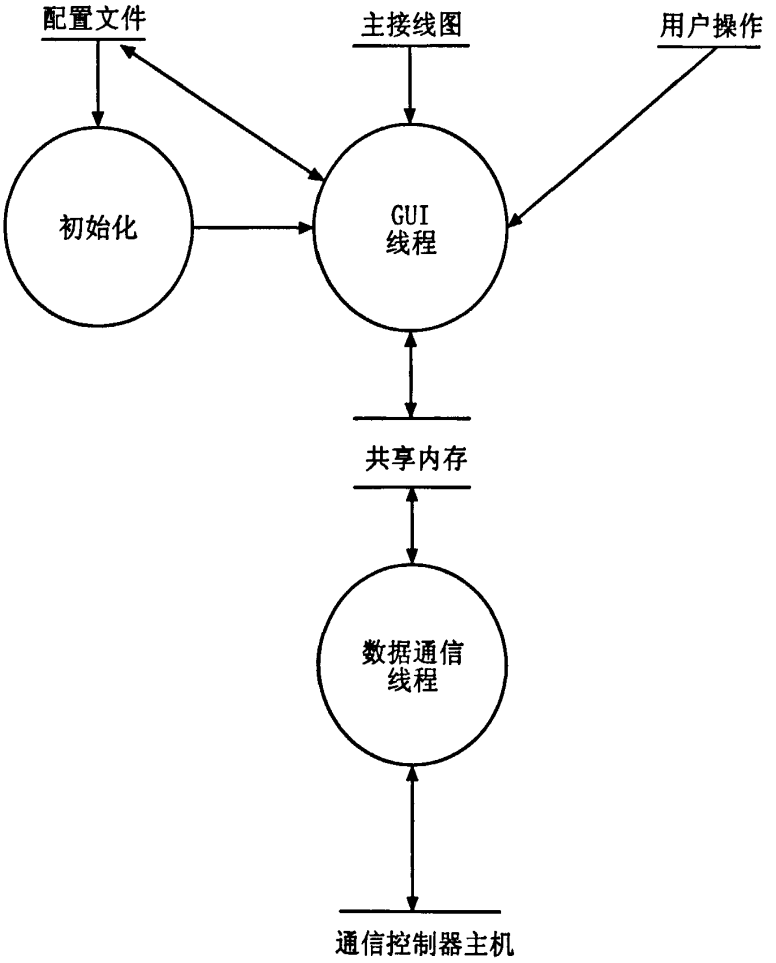

图 4-5 GUI 系统的数据流

Fig. 4-5 data stream of GUI system

4.5.4 通信协议

GUI 应用程序与通信控制器主机间的通信通过 RS232 口进行，所使用的通信协议定义如下。

表 4.1 传输帧格式

Table4.1 Transmission frame format

字节	报文内容	备注
1	同步头 EB	同步头，用来解决网络传输的连包、分包、断包问题
2	同步头 90	
3	同步头 EB	
4	同步头 90	
5	同步头 EB	
6	同步头 90	
7	Length	Length=Type+Data
8	Type	见后
9	Data1	数据
.....	.....	
7+Length	SUML	SUM= Length+Type+Data
8+Length	SUMH	

每一帧报文包含 8 字节的帧信息头，长度 L 字节的信息内容和 2 字节的帧尾。报文字节在线路上传输按字节 1、字节 2、字节 3 ..... SUML、SUMH 的顺序依次在线路上出现，即低位先发。考虑传输效率，规定了信息内容 Length＜256，大于 256 字节的自动分包发送。同步字符 EBH 90H EBH 90H EBH 90H 定义了数据帧中的起始点，用来解决数据流发送过程中产生分包、断包时重新定位数据帧。

系统共定义了 17 种数据传输类型，下面对其一一进行说明。

1. 0x01 系统遥测

Send：

EB 90 EB 90 EB 90 Length(=3) 0x01 YcCount(=12) PageNum(0-29) SumL SumH

Receive：

EB 90 EB 90 EB 90 Length(=27) 0x01 YcCount(=12) PageNum(0-29) Data SumL SumH

2. 0x02 单元遥测

Send:

EB 90 EB 90 EB 90 Length(=5) 0x02 Unit Media YcCount(=12) PageNum(0-15) SumL SumH

Receive:

EB 90 EB 90 EB 90 Length(=29) 0x02 Unit Media YcCount(=12) PageNum(0-15) Data SumL SumH

3. 0x03 系统遥信

Send:

EB 90 EB 90 EB 90 Length(=3) 0x03 YxCount(=24) PageNum(0-39) SumL SumH

Receive:

EB 90 EB 90 EB 90 Length(=6) 0x03 YxCount(=24) PageNum(0-39) Data SumL SumH

4. 0x04 单元遥信

Send:

EB 90 EB 90 EB 90 Length(=5) 0x04 Unit Media YxCount(=24) PageNum(0-3) SumL SumH

Receive:

EB 90 EB 90 EB 90 Length(=8) 0x04 Unit Media YxCount(=24) PageNum(0-3) Data SumL SumH

5. 0x05 系统电度

Send:

EB 90 EB 90 EB 90 Length(=3) 0x05 YmCount(=12) PageNum(0-29) SumL SumH

Receive:

EB 90 EB 90 EB 90 Length(=51) 0x05 YmCount(=12) PageNum(0-29) Data SumL SumH

6. 0x06 单元电度

Send:

EB 90 EB 90 EB 90 Length(=5) 0x06 Unit Media PageNum(0-15)

YmCount(=12) SumL SumH

Receive：

EB 90 EB 90 EB 90 Length(=53) 0x06 Unit Media PageNum(0-15) YmCount(=12) Data SumL SumH

7. 0x07 主机地址

Send:

EB 90 EB 90 EB 90 Length(=1) 0x07 SumL SumH

Receive:

EB 90 EB 90 EB 90 Length(=9) 0x07 IP1.1 IP1.2 IP1.3 IP1.4 IP2.1 IP2.2 IP2.3 IP2.4 SumL SumH

8. 0x08 装置状态

Send:

EB 90 EB 90 EB 90 Length(=3) 0x08 CanCount(=24) PageNum(0-3) SumL SumH

Receive：

EB 90 EB 90 EB 90 Length(=6) 0x08 CanCount(=24) PageNum(0-3) Data SumL SumH

9. 0x09 系统参数

Send:

EB 90 EB 90 EB 90 Length(=3) 0x09 SnumL SnumH SumL SumH

Receive:

EB 90 EB 90 EB 90 Length(=10) 0x09 SnumL SnumH Addr Media YacCount YdcCount YxCount YkCount YmCount SumL SumH

10. 0x0a 串口参数

Send:

EB 90 EB 90 EB 90 Length(=2) 0x0a ComNum SumL SumH

Receive:

EB 90 EB 90 EB 90 Length(=8) 0x0a ComNum BaudrateL BaudrateH DataBit CheckBit StopBit Mode(0: 异步方式，1: 同步方式) SumL SumH

11. 0x0b 遥控转发序号

Send:

EB 90 EB 90 EB 90 Length(=3) 0x0b YknumL YknumH SumL SumH

Receive:

EB 90 EB 90 EB 90 Length(=6) 0x0b YknumL YknumH Addr Media Point SumL SumH

12. 0x0c 遥控选择

Send:

EB 90 EB 90 EB 90 Length(=5) 0x0c Addr Media Point Action SumL SumH

Receive:

EB 90 EB 90 EB 90 Length(=5) 0x0c Addr Media Point Action SumL SumH

13. 0x0d 遥控执行

Send:

EB 90 EB 90 EB 90 Length(=5) 0x0d Addr Media Point Action SumL SumH

Receive:

EB 90 EB 90 EB 90 Length(=5) 0x0d Addr Media Point Action SumL SumH

14. 0x0e 遥控取消

Send:

EB 90 EB 90 EB 90 Length(=5) 0x0e Addr Media Point Action SumL SumH

Receive:

EB 90 EB 90 EB 90 Length(=5) 0x0e Addr Media Point Action SumL SumH

15. 0x0f 遥测调试

Send:

EB 90 EB 90 EB 90 Length(=4) 0x0f Addr Media Point SumL SumH

Receive:

EB 90 EB 90 EB 90 Length(=38) 0x0f Addr Meida Point Data SumL SumH

16. 0x10 通信数据

Send:

EB 90 EB 90 EB 90 Length(=2) 0x10 ComNum SumL SumH

Receive:

EB 90 EB 90 EB 90 Length(=58) 0x10 ComNum Data SumL SumH

17. 0x11 Soe 事件数据

Send:

EB 90 EB 90 EB 90 Length(=14) 0x11 Addr Media Point Action SoeMsL SoeMsH SoeSec SoeMin SoeHour SoeDay SoeMon SoeYearL SoeYearH SumL SumH

第5章 变电站GUI软件系统的实现

5.1 BootLoader移植的实现

在嵌入式系统中，通常没有像PC机中BIOS那样的固件程序，因此整个系统的启动加载任务完全由BootLoader来完成。在一个基于ARM的嵌入式系统中，系统上电或复位时通常都从地址0x00000000处开始执行，而这个地址通常就存放系统的BootLoader。通过这段小程序可以初始化硬件设备，建立内存空间的映射图，如初始化CPU、堆栈、初始化存储器系统等，为嵌入式操作系统内核准备好正确的软硬件环境。S3C2410支持从Nandflash存储器启动，它具备一个内部SRAM缓冲区，称为Steppingstone。在系统启动时，Nandflash的前4K字节将会被拷贝到Steppingstone执行。

U-boot是一个开放源代码、支持多种处理器和操作系统并且稳定、灵活、可靠的BootLoader，同时它还提供了对网络的支持。鉴于以上这些优点，选用U-boot作为系统的BootLoader。U-boot 1.2.0版本的源代码可以从网上下载到，里面已经加入了对S3C2410处理器的支持。

由于U-boot代码是遵循GNU协议的，而GNU编译器通常都支持编译脚本。因此U-boot目录中还存放着脚本文件，如Makefile、mkconfig等。脚本文件的编写规则和Linux是相同的，可以参考有关资料对其进行编辑修改。

U-boot移植的主要工作是针对特定的平台对其源代码进行修改及进行交叉编译。由于这个版本的U-boot已经加入了对S3C2410芯片的支持，因此只需要针对我们的硬件平台进行少量的修改就可以了。本文中主机的开发平台使用RedHat Linux 9.0操作系统，交叉编译工具使用arm-linux-gcc 2.95.3。具体的移植步骤如下。

首先建立交叉编译环境。在装有RedHat Linux 9.0操作系统的主机上建立工作目录，以root身份登录系统，并把下载到的arm-linux-gcc 2.95.3程序包解压到此工作目录。解压后生成一个名为2.95.3的目录，交叉编译工具就在这个目录中的/bin目录下。增加这个路径到环境变量，并修改U-boot代码根目录下的Makefile文件，配置交叉编译环境编译器的路径。

然后修改U-boot代码根目录下的/board/smdk2410子目录中的config.mk文

件，定义 start.s 文件开始执行的地址。修改 U-boot 目录下的/include/configs 子目录中的 smdk2410.h 文件。根据硬件平台的实际情况分别定义网卡地址、RAM 的地址映射及其大小以及 Flash 存储器的控制地址和块大小。其他文件如 flash.c 和 serial.c 等不必加以改动。

最后，对修改后的源代码进行交叉编译，生成 U-boot 映像文件。执行如下命令。

将执行 make 命令后生成的 U-boot.bin 文件通过 JTAG 口使用 Flash 烧写软件写入 NAND Flash，重新上电，就可以通过串口看到 U-boot 的启动信息了。

5.2 嵌入式 Linux 在目标平台上移植的实现

5.2.1 Linux 2.6 内核简介

作为各种 Linux 发行版的共同核心，Linux 内核多年来一直不间断地引进新技术进行革新，逐步提高自身的各种性能例如可伸缩性、可用性和技术支持等。众多的改进都是围绕增加对其它架构、处理器、总线、端口和外设的支持而进行的。从 Linux 2.2 内核开始，因为每个新内核的诞生都要经过严格的检验和测试，所以其开发周期大约都保持在两年左右。2003 年 12 月 18 日，Linux 2.6.0 内核稳定版本正式推出。内核是一个操作系统最核心的部分，它完成操作系统最基本的功能，包括进程调度、磁盘管理、设备管理、网络管理等，其实现算法或方式的优劣将直接影响到整个操作系统的性能，甚至影响在该操作系统上开发的应用程序的性能。Linux 2.6 内核中针对嵌入式开发所做的改进有以下几个方面。

- 抢占式内存管理

在 Linux 2.6 内核中，代码被设置了抢占点，这就意味着调度程序会中止现在正在运行的进程而来执行优先级更高的进程。在系统调用过程中，Linux 2.6 会定时地检查抢占点，以避免不合理的延迟发生。而在检查过程中，调度进程很可能就会中止当前的进程来让另外一个进程运行。

有执行时间限制的软件和虚拟内存请求页面调度是不兼容的，因为这种方法处理页面慢的缺点会破坏程序的响应能力。而 Linux 2.6 内核可以被编译成没有虚拟内存的系统来消除这个问题，这就要求软件设计人员必须考虑要有足够的实内存来运行应用程序。

- 共享内存的改进

针对一个有多处理器的嵌入式设备，Linux 2.6 内核为多进程提供了一种不同的途径，即 NUMA (Non Uniform Memory Access)。这种方法中，内存和处理器是相互连接的，但对于每一个处理器，某些内存是“关闭”的，而某些内存则“更远”。这意味着当内存竞争出现时，“更近”的处理器对就近的内存有较高的使用权。Linux 2.6 内核提供了一套功能来定义内存和处理器之间的拓扑关系，调度程序可以利用这些信息来为任务分配本地内存。这样将减少内存竞争造成的瓶颈，提高吞吐量。

- 集成 μCLinux

μCLinux 是 Linux 应用在微控制器平台的一个项目，是一种针对不带 MMU(内存管理单元)的 ARM 微处理器的嵌入式操作系统。μCLinux 完全符合 GNU/GPL 公约，完全开放源代码，它的很多特性都和 Linux 相同，最典型的特征是无 MMU。Linux 2.6 内核扩展多嵌入式平台支持的一个主要途径就是把 μCLinux 的大部分并入主流内核功能中。目前许多嵌入式处理器如 ARM 系列等，很多都是无 MMU 的。μCLinux 在嵌入式系统中的应用非常广泛。因此，Linux 2.6 内核对无 MMU 体系结构的支持，即将 Linux 和 μCLinux 合并到统一的新内核中，无疑为 Linux 在嵌入式领域的广泛应用加重了砝码。

- POSIX 线程、信号和计时器

在 Linux 2.6 内核中，POSIX 标准得到了很大的改进。与 POSIX 线程一起，Linux 2.6 内核把 POSIX 信号和 POSIX 高精度计时器作为了主流内核的一个组成部分，新的 POSIX 信号不能被丢失，并且可以携带信息作为参数。此外，POSIX 信号也可以从一个 POSIX 线程传送至另外一个线程，而不是像 Unix 信号一样，只能从一个进程至另外一个进程。

嵌入式系统通常要求硬件能够在固定的时间安排下来运行任务。POSIX 计时器可以轻松地让任何一个任务都可以周期性地得到预定安排的时间。计时器的时钟可以达到很高的精度，从而可以让软件工程师更加精确地控制任务的调度。

5.2.2 Linux 2.6 内核结构分析

本系统中我们选用 Linux 2.6.18 版本进行移植，下面对其代码的结构进行分析。Linux 内核主要由 5 个子系统组成：进程调度、内存管理、虚拟文件系统、

网络接口、进程间通信。整个代码的分布如图 5-1 所示。

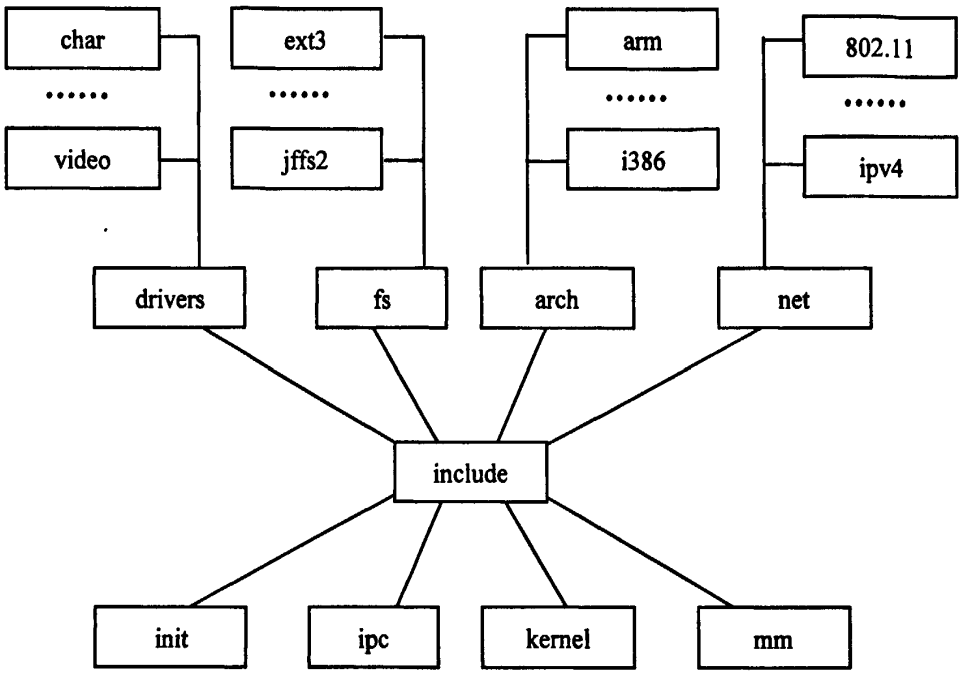

图 5-1 Linux 2.6 内核代码结构

Fig. 5-1 structure of Linux 2.6 kernel

/arch 子目录包含了所有硬件结构特定的内核代码，如 arm、i386 等。Linux 系统能支持如此多的平台的一个原因就是因为其内核把源程序代码清晰的划分为与体系结构相关的部分和与体系结构无关的部分。arch 目录下即为与体系结构相关的代码。其中每一个子目录都代表一种硬件平台，例如我们要使用的 ARM 平台。因此，arch 目录下的文件是移植过程中我们需要特别关注的。对于任何平台，都必须包含如下几个子目录。

/boot 子目录存放启动内核所使用的部分或全部平台特有代码。

/kernel 子目录存放支持体系结构所特有的特征(比如信号处理)的实现。

/lib 子目录存放高速的体系结构特有的通用函数(比如 strlen 和 memcpy)的实现。

/mm 子目录存放体系结构特有的内存管理程序的实现。

/math-emu 子目录模拟 FPU 的代码。对于 ARM 处理器来说，此目录用 mach-xxx 代替。

/drivers 子目录包含了内核中所有的设备驱动程序，如网卡、Flash 等。

/fs 子目录包含了所有文件系统的代码，如 ext3、jffs2 等。

/include 子目录包含了建立内核代码时所需的大部分头文件，例如与平台无关的头文件在 include/linux 子目录下。不同平台需要的头文件会有所不同，因此该目录跟 arch 目录一样，按平台的不同划分了多个子目录。

/init 子目录包含了内核的初始化代码，内核从此处开始工作。

/ipc 子目录包含了进程间通信的代码。

/kernel 子目录包含了主内核代码。

/mm 子目录包含了内存管理代码。

5.2.3 嵌入式 Linux 的文件系统

文件系统是操作系统的重要组成部分，用于控制对数据、文件及设备的存取。它提供对文件和目录的分层组织形式、数据缓冲以及对文件存取权限的控制。Linux 的一个重要的特点就是它支持多种不同的物理文件系统，这使得 Linux 系统的应用非常灵活。

常用的嵌入式 Linux 的文件系统有 CRAMFS、JFFS/JFFS2、YAFFS 等。其中 CRAMFS 文件系统具有只读和可压缩的特点，因此使用 CRAMFS 可以很好的保护系统文件不被改写和破坏，满足系统安全性的要求；而 YAFFS 文件系统是可读写的，它的速度更快，因此使用 YAFFS 可以使系统更加灵活，提供断电可靠性，从而满足不同的需求。在本系统中将采用 CRAMFS 作为根文件系统，YAFFS 作为用户数据存储分区，这样既能保证操作系统本身数据的安全性，又能满足变电站通信控制器 GUI 系统对数据存储的灵活性的要求。

5.2.4 Linux 2.6.18 内核在目标平台上移植的具体实现

Linux 2.6.18 内核已经实现了对 S3C2410 芯片的支持，因此需要改动的源代码并不是很多。这里主要是加入对 YAFFS 文件系统的支持以及对 NAND Flash 芯片的支持。具体移植步骤如下。

1. 交叉编译环境的建立。这一步与建立 U-boot 交叉编译环境的过程类似，所不同的是要使用的交叉编译器版本为 arm-linux-gcc-3.4.1。

2. Linux 2.6.18 内核及 YAFFS 文件系统源代码的准备。Linux 2.6.18 内核及 YAFFS 文件系统的源代码包可以从网上下载得到，之后我们需要将其解压到工

作目录，为下面的工作做好准备。

3. 修改内核目录树根目录下的的 Makefile，指明平台和交叉编译器。

4. 建立内核对 YAFFS 文件系统的支持。

首先在内核/fs 目录中建立/yaffs 子目录，并将解压出来的 YAFFS 文件系统源代码拷贝到该目录下。

然后修改内核目录中/fs/Kconfig 文件，添加 YAFFS 配置选项。

source “fs/yaffs/Kconfig”

修改内核目录中/fs/makefile 文件，添加 YAFFS 选项。

obj-$(CONFIG_YAFFS_FS) += yaffs/

在内核目录/fs/yaffs 目录下建立 Kconfig 和 Makefile 文件，可以参考其他文件系统目录下的同名文件进行修改。

其中 Makefile 文件内容为：

```
obj-$(CONFIG_YAFFS_FS) += yaffs.o
yaffs-y := yaffs_ecc.o yaffs_fs.o yaffs_guts.o
yaffs-y += yaffs_packedtags2.o
yaffs-y += yaffs_tagscompat.o yaffs_tagsvalidity.o
yaffs-y += yaffs_mtdif.o yaffs_mtdif2.o
```

5. 建立内核对 NAND Flash 的支持并添加 Flash 分区信息。

内核目录中 arch/arm/mach-s3c2410/devs.c 文件是对 S3C2410 平台的设备定义文件，其中的核心数据结构为 platform_device。这个结构在内核/include/linux/platform_device.h 中进行了定义，其结构如下。

```
Struct platform_device {
const char * name;
u32    id;
struct device   dev;
u32     num_resources;
struct resource   * resource;
};
```

首先添加头文件。

```
#include <linux/mtd/partitions.h>
```

```
#include <linux/mtd/nand.h>
#include <asm/arch/nand.h>
```

然后加入 NAND Flash 分区信息。

```
static struct mtd_partition partition_info[] = {
    {
        name:       "boot",
        size:        0x20000,
        offset:      0x0,
     },
     {
        name:       "kernel",
        size:        0xe0000,
        offset:      0x20000,
     },
    {
        name:       "root",
        size:        0x1400000,
        offset:      0x100000,
     },
    {
        name:       "yaffs",
        size:        0x2b00000,
        offset:      0x1500000,
    }
};
```

以上代码将 Flash 存储器分为 4 个区，分别为 boot 区、kernel 区、root 区和 yaffs 区。其中 boot 区用于存放 BootLoader(即 U-boot)及其参数，kernel 区存放 Linux 内核映像，root 区存放根文件系统的映像，yaffs 区为用户数据区，可用于存放用户应用程序及各种配置文件、数据文件等。

之后将上述分区信息加入。

```
struct s3c2410_nand_set nandset =
{
    /* 分区数 */
    nr_partitions: 4,
    /* 分区信息 */
    partitions: partition_info,
};
```

并添加 NAND Flash 控制器初始化代码。

```
struct s3c2410_platform_nand myplatform=
{
    tacls:0,
    twrph0:20,
    twrph1:0,
    sets: &nandset,
    nr_sets: 1,
};
```

此结构中的参数需要根据 S3C2410 的数据手册进行设定。

最后加入设备信息。

```
struct platform_device s3c_device_nand =
{
    .name = "s3c2410nand",
    .id = -1,
    .dev = {
            .platform_data = &myplatform
           }
    .num_resources = ARRAY_SIZE (s3c_nand_resource),
    .resource = s3c_nand_resource, /* Nand Flash Controller Registers*/
};
```

6. 添加内核启动时初始化 NAND Flash 的代码。

修改内核中/arch/arm/mach-s3c2410/mach-smdk2410.c 文件。

```
static struct platform_device *smdk2410_devices[] __initdata = {
    &s3c_device_usb,
    &s3c_device_lcd,
    &s3c_device_wdt,
    &s3c_device_i2c,
    &s3c_device_iis,
    /* 添加如下语句即可 */
    &s3c_device_nand,
    };
```

7. 禁止 NAND Flash ECC 校验。

由于我们的内核是使用 U-boot 烧写到 Flash 上，U-boot 是用软件 ECC 算法产生校验码，而内核中的 ECC 校验码是由 S3C2410 中 NAND Flash 控制器产生的，为避免二者发生冲突，我们选择禁止内核 ECC 校验。

8. 配置并编译内核。

每一个选项前面都有一个"[]"或者"<>",其含义如下。

"[]"表示该选项有两种选择方式：

[*] 直接编译进内核；

[] 不进行编译。

"<>"表示该选项有三种选择方式：

<*> 直接编译进内核；

<M> 编译成模块形式，而不编译进内核；

<> 不编译。

对内核进行配置需要根据具体情况进行量体裁衣，配置的选项非常多(有几百个)，这里只对其中需要特别注意的部分项目进行说明。

System Type 菜单项，需选择 S3C2410。

Device Drivers 菜单项。

该菜单选项及其子菜单选项提供所有设备驱动程序的配置选择。

进入 Memory Technology Devices(MTD)子菜单，配置如下。

<*> Memory Technology Devices (MTD) support

[*] MTD partitioning support

NAND Flash Device Drivers-->

<*> NAND Device Support

<*> NAND Flash support for S3C2410 SoC

进入 Network Device support 子菜单，配置如下。

[*] Network Device support

Ethernet (10M or 100M bit)-->

[*] Ethernet (10M or 100M bit)

<*> CS8900 support

File Systems 菜单项。

该菜单选项提供所有文件系统的配置选择。

进入 Miscellaneous filesystems 子菜单，配置如下。

<*> Compressed ROM file system support (cramfs)

<*> YAFFS file system support

内核配置完成后，选择 exit 并对所作配置进行保存。

最后编译生成内核映像文件 zImage。

5.2.5 制作 Linux 根文件系统

根文件系统(Root Filesystem)是存放运行、维护系统所必需的各种工具软件、库文件、脚本、配置文件和其他特殊文件的地方，也可以安装各种软件包。Linux 内核必须与根文件系统协同工作才能提供给用户使用。一般来说，Linux 的根文件系统包含如下一些目录和文件。

- /bin 目录。存放常用的二进制程序，如 ls、rm 等。
- /dev 目录。存放设备文件，如串口、存储器等。
- /etc 目录。存放系统配置文件，如 group、profile 等。
- /home 目录。用户的根目录。
- /root 目录。超级用户的根目录。
- /sbin 目录。存放超级用户运行的二进制文件。

- /mnt 目录。为其他文件系统提供安装点。
- /lib 目录。存放库文件。

制作根文件系统首先需要建立目标板空根目录文件夹及根目录下的各子文件夹。然后，编译 BusyBox，基本按照默认选项进行配置就可以。配置好之后，保存退出，并对其编译和安装到刚才建立的根文件系统目录下。最后，在根文件系统/etc/init.d 目录下，建立一个启动脚本文件，命名为 rcS，并添加如下内容。

```
# Net Setting
/sbin/ifconfig lo 127.0.0.1 up
/sbin/ifconfig eth0 192.168.0.7 netmask 255.255.255.0
/sbin/route add default gw 192.168.0.1
#! /bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin:
#mount
mount -t yaffs /dev/mtdblock/3 /yaffs
```

保存退出后，在根文件系统目录上一级路径执行 mkcramfs 命令即可得到根文件系统映像 root.cramfs。

5.3 Qt/Embedded 在目标平台上移植的实现

首先从网上下载得到 Qt/Embedded 的源代码包 qt-embedded-free-3.3.1.tar.gz，将其解压。

```
#cd /usr/local/arm
#tar zxvf qt-embedded-free-3.3.1.tar.gz
#cd qt-embedded-free-3.3.1
```

然后设置编译 Qt/Embedded 的环境变量。

```
#export QTDIR=/usr/local/arm/qt-embedded-free-3.3.1/
#export QTEDIR=$QTDIR
#export PATH=$QTDIR/bin:$PATH
#export D_LIBRARY_PATH=/usr/local/arm/qt-embedded-free-3.3.1/lib:
$LD_LIBRARY_PATH
```

之后使用 configure 命令生成 Makefile 文件。

最后运行 make 命令进行编译。

这样在 qt 目录/lib 下生成四个库文件，分别是 libqte.so、libqte.so.3、libqte.so.3.3、libqte.so.3.3.1。将这 4 个库文件拷贝到目标板根文件系统的/usr/qt/lib 目录下，并修改/usr/etc/profile 文件。

PATH=/bin:/sbin:/usr/bin:/usr/sbin: /usr/qt/bin

LD_LIBRARY_PATH=/lib: /usr/lib: /usr/qt/lib

最后，重新使用 mkcramfs 命令生成新的根文件系统映像，Qt/Embedded 在目标板上的移植就完成了。

5.4 用户界面程序的实现

5.4.1 主程序流程

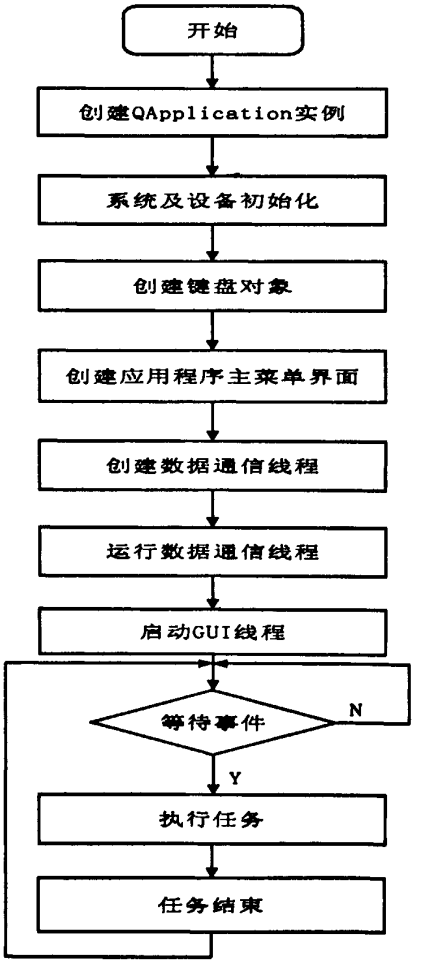

图 5-2 主程序流程图

Fig. 5-2 flow chart of main

main()程序是 Qt 的主程序，是应用程序的入口点。main()程序的作用是在把控制交给 Qt 库之前执行一些初始化，然后 Qt 库通过事件来向程序告知用户的行为。其流程如图 5-2 所示。

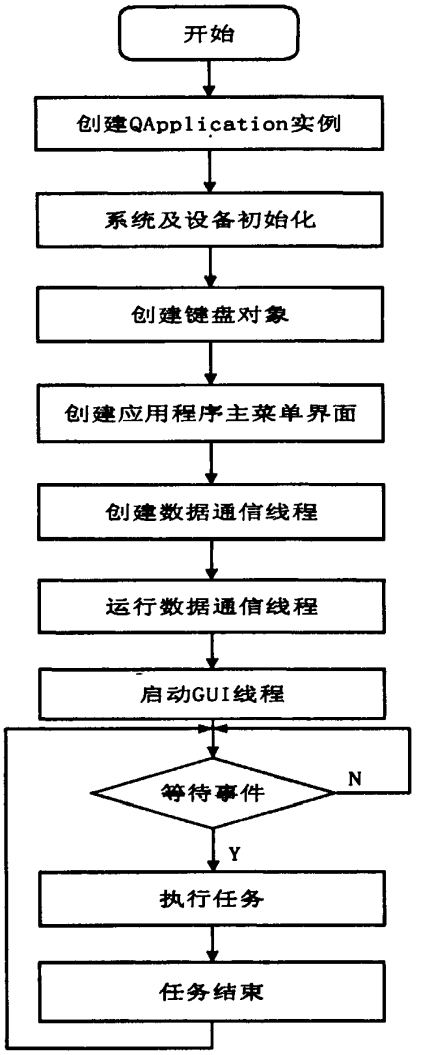

图 5-2 主程序流程图

Fig. 5-2 flow chart of main

5.4.2 键盘处理程序

系统共设置了 9 个按键，分别为“上”、“下”、“左”、“右”、“加”、“减”、“确定”、“取消”、“切换”键，按键识别和编码由一片单独的单片机 89C2051 完成，通过 89C2051 的串口中断方式与系统的串口 1 通信将键盘编码传递给 GUI 程序。GUI 程序的键盘处理由 ComKeyboardHandle 类来实现，在 ComKeyboardHandle 类的构造函数中建立串口数据与 QSocketNotifier 对象的关

联。

串口 1 数据的变化就会触发调用 ComKeyboardHandle 的 readKeyboardData() 函数，readKeyboardData()函数的作用是将接收到的串口数据转换成标准的 Qt 键盘码，其映射关系如表 5-1 所示。

表 5-1 串口数据与标准 Qt 键盘码映射表

Table 5-1 mapping table from serial data to Qt keycode

串口数据	标准 Qt 键盘码
0x11	Qt::Key_Tab
0x12	Qt::Key_Up
0x13	Qt::Key_Escape
0x14	Qt::Key_Left
0x15	Qt::Key_Enter
0x16	Qt::Key_Right
0x17	Qt::Key_Minus
0x18	Qt::Key_Down
0x19	Qt::Key_Plus

5.4.3 数据通信线程

由于变电站 GUI 系统与其主机侧有大量的数据需要交换，我们将其放在一个独立的线程里运行。当处于接收状态时，数据通信线程将通过串口取得的数据放在共享数据区供 GUI 线程使用；当处于发送状态时，数据通信线程从共享数据区取得需要发送的数据，对其按规定的格式编码后通过串口传送给主机侧。其程序流程如图 5-3 所示。

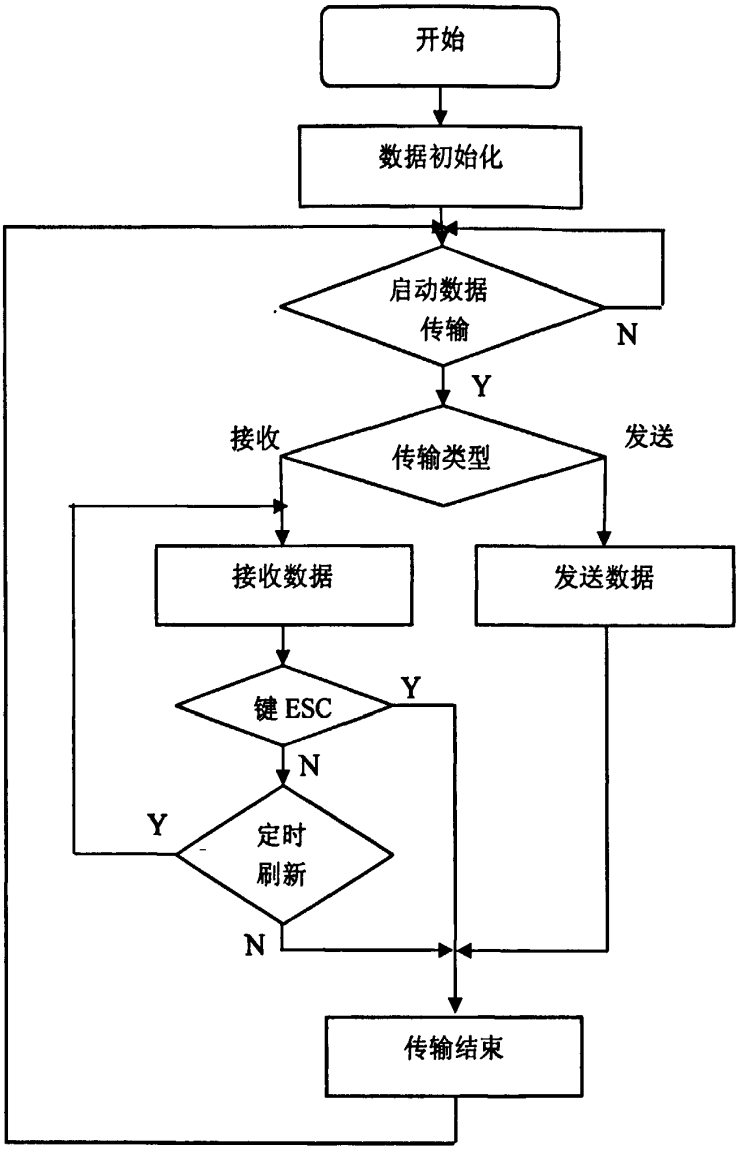

图 5-3 数据通信程序流程图

Fig. 5-3 flow chart of data communication program

5.4.4 主接线图界面的实现

本系统中设计实现了 pic_show 类来实现主接线图的显示，其工作机制如下所述。首先，系统对相关数据进行初始化，然后读取存放在/yaffs 目录下的配置文件，根据配置文件将各部件(例如母线、主变压器、断路器等)根据其位置坐标、放大比例、旋转角度等信息显示到屏幕上。然后启动数据传输，从主机侧获取各元件的当前状态信息，将这些数据放在共享数据区。最后，启动定时器，定时读取共享数据区中最新的元件状态信息，根据这些状态信息刷新显示各元件。这样

就实现了断路器的分/合以及其他元件当前状态的准实时显示。其流程如图 5-4 所示。

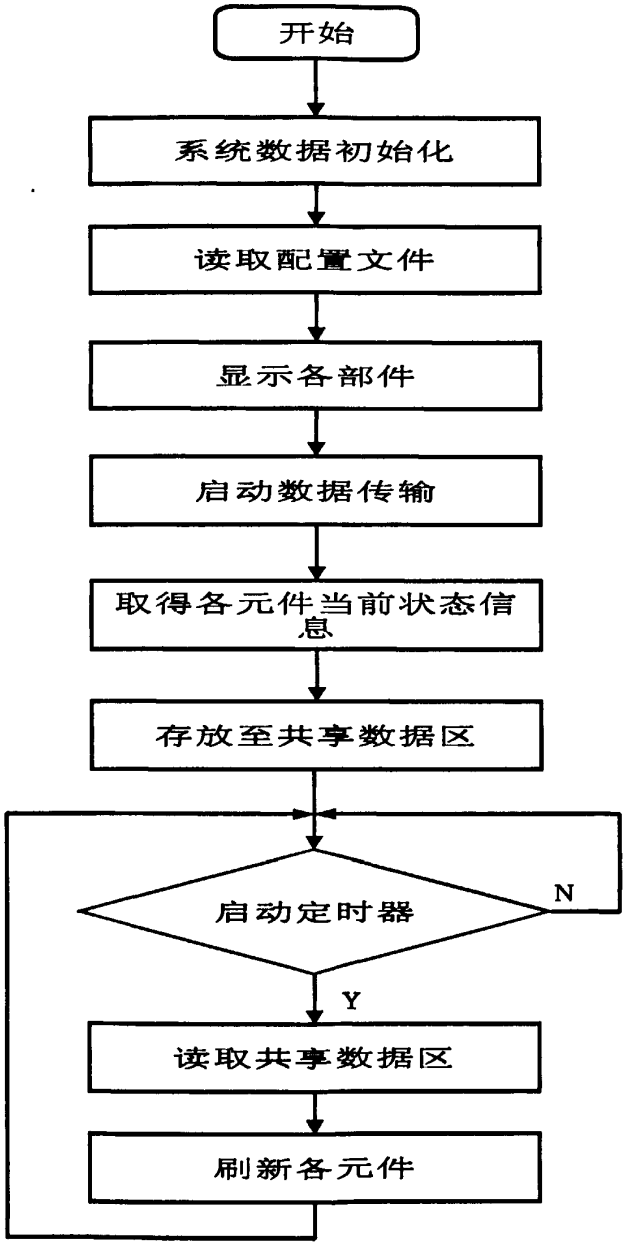

图 5-4 主接线图实现流程图

Fig. 5-4 flow chart of system wiring program

5.4.5 数值修改功能的实现

在变电站通信控制器 GUI 系统中有时会用到少量的数值输入、修改操作，比如定值的修改等。由于系统没有提供数字键，无法直接进行数字的输入操作，

因此我们对 QLineEdit 类进行了扩展，生成 MyLineEdit 类来实现使用方向键及“+”、“—”键完成数值的输入和修改功能。功能的实现主要通过重新实现 QLineEdit 类的虚函数 keyPressEvent()来完成，其流程如图 5-5 所示。

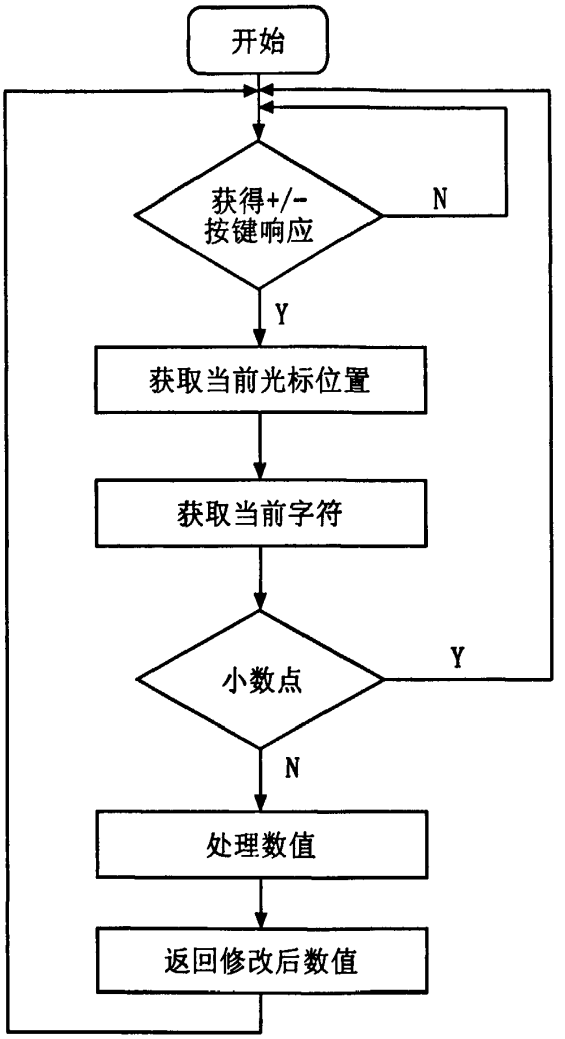

图 5-5 数值修改功能流程图

Fig. 5-5 flow chart of changing value fuction

5.4.6 九域图界面的实现

在具有电压无功控制(VQC)功能的系统中需要实时显示当前状态的九域图信息。在 VQC 理论中，根据电压的上下限和无功的上下限将平面坐标系分割成 9 个区域，系统需要根据当前工作状态处于哪一个区域来决定采取什么措施来保证电网的正常运行，值班维护人员也可以通过查看系统的九域图信息直观的了解系统的当前工作状态，从而及时作出反应。在本 GUI 软件系统中设计了 show_map

类来实现九域图的显示功能。在这个类中，界面上的线和点以及坐标等基本图形的绘制均使用 Qt 的底层绘图函数来实现，其工作机理如下所述。

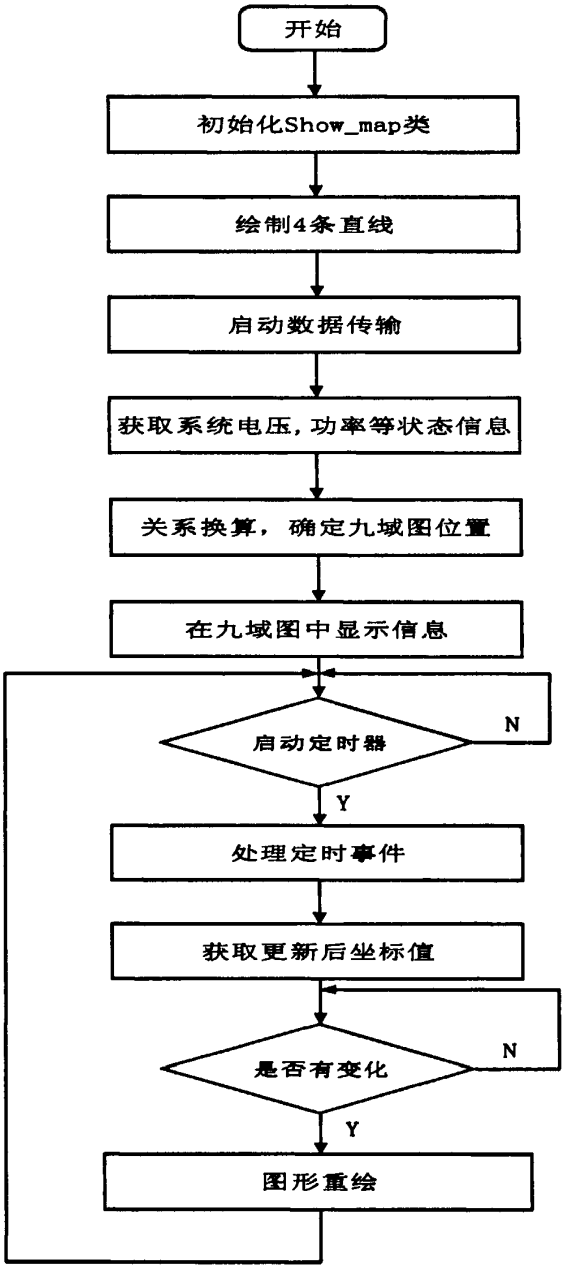

图 5-6 九域图实现流程图

Fig. 5-6 flow chart of nine-area map

首先，对 show_map 类的内部数据包括点的坐标、线的位置等进行初始化，并使用 Qt 的 drawLine()函数在屏幕的固定位置画出 4 条直线，从而将整个屏幕划分为 9 个区域。之后，启动数据传输，从通信控制器的处理程序获取电压上下限、无功上下限、系统当前状态坐标等信息，之后根据其相互之间的关系进行换

算，从而确定其在九域图中所处的位置，并使用 Qt 的 drawEllipse()函数将其显示在对应的区域中。最后，启动定时器触发定时事件，在定时事件处理函数中根据获取的更新后的坐标值对整个图形进行重绘，从而实现系统九域图信息的准实时显示。其实现流程如图 5-6 所示。

值得注意的是，与用户平常习惯的纵坐标自下而上递增的坐标系统不同，Qt 中所采用的坐标系统定义为屏幕的左上角为(0,0)坐标，其纵坐标是自上而下递增的。因此，我们在进行坐标换算的时候必须将其考虑在内，从而正确的实现坐标的换算。

5.4.7 其他用户界面

其他的用户界面基本可以使用 Qt 已有的 GUI 类例如 QLable、QTable、QPushButton 等来编写，之后在需要获取数据时调用相应的数据传输函数，并添加数据处理代码以完成相应的功能。

5.5 软件的调试

在系统的开发过程中，软件的调试是一项很重要的工作，它在整个开发过程中占据了很大的比例。在本系统中包括对 BootLoader 移植的调试、Linux 内核移植的调试以及 Qt 应用程序的调试三个部分。

对于ARM系统的软件调试，一般采用宿主机－目标平台的交叉调试的方法。传统上有通过仿真器进行跟踪调试和纯软件调试两种办法。在本系统中，由于对 BootLoader 以及 Linux 内核的修改不是特别多，所以采用了纯软件调试的方法。这种方法的主要手段就是在其执行代码中某些关键位置添加 printf 语句，然后通过 Windows 的超级终端与目标平台进行通信，通过查看超级终端的打印输出信息判断问题，从而达到代码调试的目的。同样，对于 Qt 应用程序的调试，使用这种方法也完全可以达到代码调试的目的，所不同的是对于 Qt 应用程序的调试是使用 Linux 下的 ztelnet 与开发板进行通信从而查看其打印输出信息。

调试系统的硬件主要由一台计算机主机、目标平台、一条串口连接线、一根交叉网线以及 JTAG 连接线组成。在软件方面主机上需要安装 Windows 及 Linux 双系统(也可以采用在 Windows 中安装虚拟机，并在虚拟机上安装 Linux 操作系统的方案)。在 Windows 系统中需安装 Flash 烧写软件，并对超级终端进行配置。

5.6 系统测试

为了验证在实际情况下系统能否正常运行，以及其功能是否能够达到需求，我们需要对系统进行测试。

5.6.1 测试系统的构建

- 烧写 U-boot 到目标板并设置启动参数。

首先通过并口电缆及接口板连接 PC 机的并口和目标板的 JTAG 口，使用 Jflash 软件将编译好的 U-boot.bin 文件烧写到目标板的 NAND Flash 中。然后通过串口电缆建立目标板串口和 PC 机串口的连接，重新启动目标板，通过 PC 机上的超级终端可以看到系统的启动信息。最后，U-boot 成功启动后需要对其参数进行设置。

设置目标板的 IP 地址：

setenv ipaddr 10.144.85.3

设置主机 IP 地址：

setenv serverip 10.144.85.201

设置 Linux 内核启动参数并保存。

setenv linux_arg noinitrd root=/dev/mtdblock/2 init=/linuxrc

console=ttySAC0

setenv bootcmd nandr c 1d0000 30008000\;bootm

saveenv

- 烧写 Linux 内核及文件系统

由于 U-boot 提供了对网络的支持，因此使用 tftp 下载内核及文件系统，可以达到较高的传输速度。为使 GUI 应用程序可以在 Linux 系统启动后自动运行，需要修改根文件系统/etc/init.d 目录下的 rcS 文件，在文件中加入如下命令行。

./yaffs/ccGUI –qws

保存 rcS 文件后重新将根文件系统打包即可。

在主机建立 tftp 服务器，然后在 U-boot 环境下使用如下命令将内核映像和根文件系统映像烧写到 NAND Flash 中。

tftp 30000000 zImage

nandw c 1d0000 30000000

tftp 30000000 root.cramfs

nandw 80 1e00000 30000000

- 下载配置文件及 ccGUI 应用程序

重新启动目标板，等待 Linux 系统成功启动后，使用 ftp 工具把所有的系统配置文件及 ccGUI 应用程序下载到 Linux 系统的/yaffs 目录下。至此，测试系统已经成功建立起来了。

5.6.2 功能测试

进行功能测试的目的是验证系统能否实现所有设计的功能。

分别对系统的各个界面进行功能测试，结果均能实现预定的功能，说明系统功能达到设计要求。

5.6.3 紧张度测试

紧张度测试是使系统在高度紧张的情况下运行，即在很短的时间内使数据量达到峰值。

由于事件记录界面工作时所创送的数据量相对较多，我们选用该界面模块进行紧张度的测试。在通信控制器主机侧编写测试程序，使其每 200ms 发送一条事件记录，该 GUI 系统仍能及时相应用户的操作，并且事件记录的显示无停滞现象。

5.6.4 可靠性测试

在变电站自动化中，对系统的可靠性要求相对较高，因此我们进行了可靠性的测试。将该系统在变电站环境中实际运行 30 天，在这段时间内，系统工作一直正常，并无故障发生，说明软件和硬件工作稳定。

下面是几个典型界面的截图。

图 6-1 系统主菜单界面

Fig. 6-1 interface of main menu

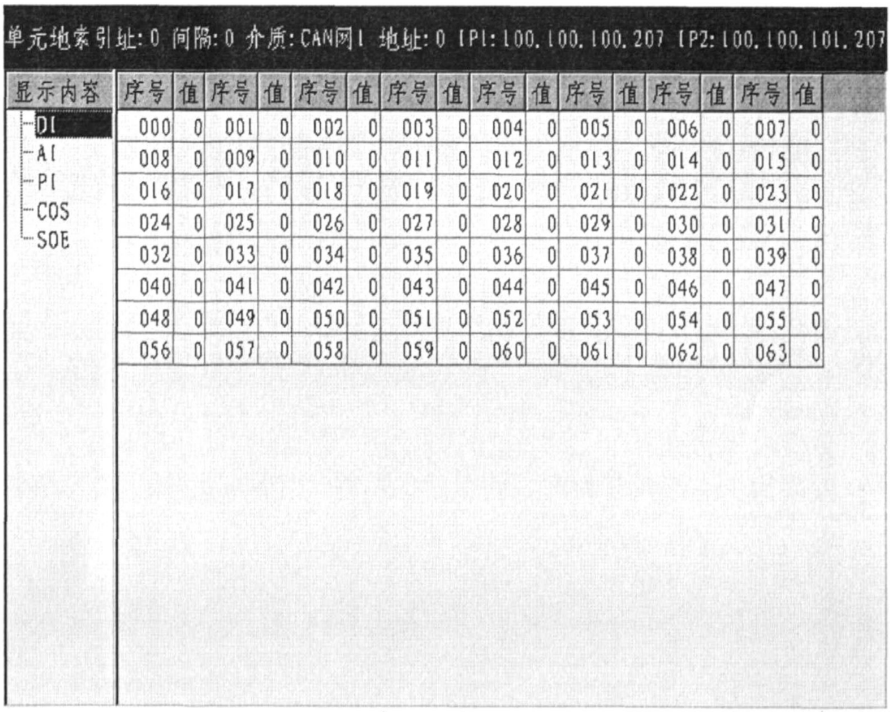

图 6-2 遥信数据界面

Fig. 6-2 interface of tele-indacation values

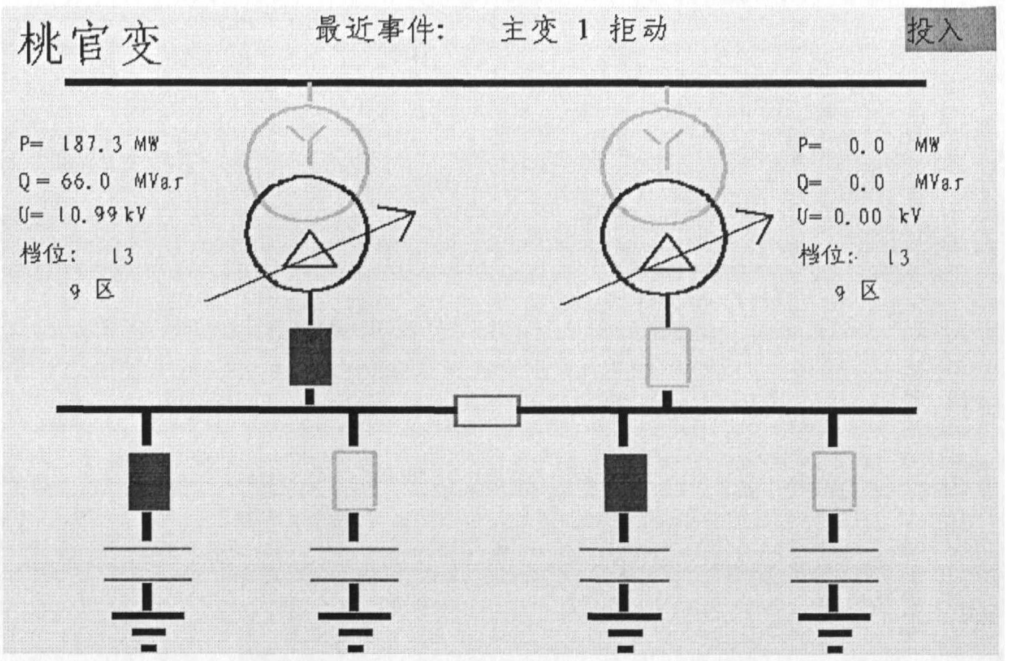

图 6-3 主接线图界面

Fig. 6-3 interface of primary electric wiring of system

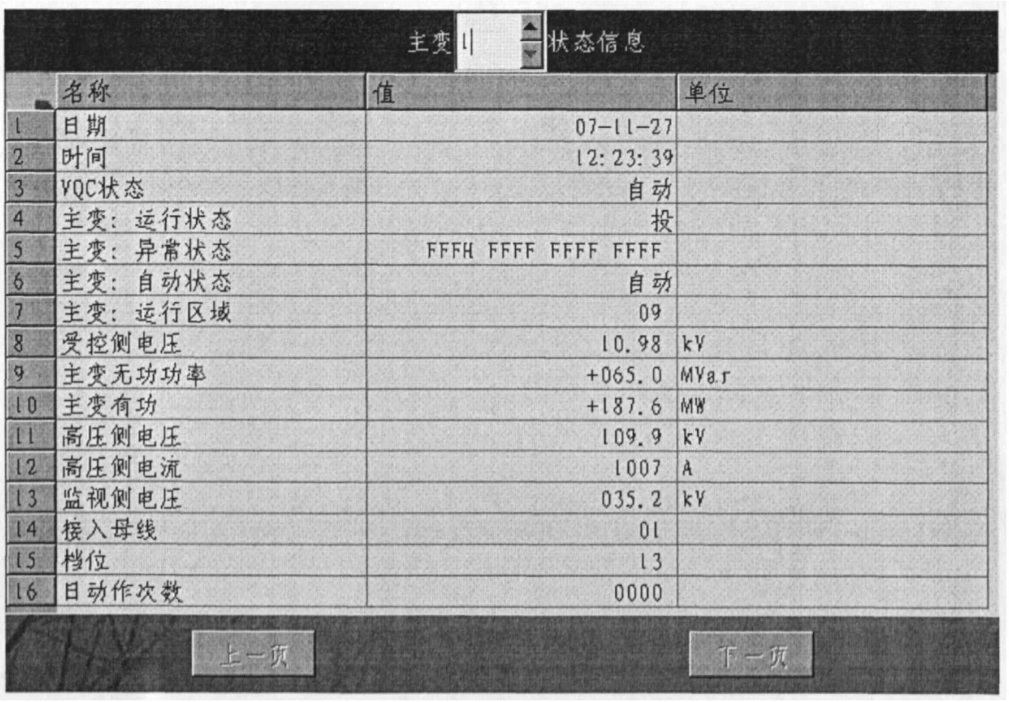

图 6-4 状态信息界面

Fig. 6-4 interface of status information

目前，该系统已经在某变电站的自动化系统中得到了应用，运行状况良好。

第6章 总结与展望

随着嵌入式系统在工业控制领域越来越广泛的应用，用户对嵌入式图形用户界面(GUI)的要求也越来越高。嵌入式 GUI 系统是嵌入式设备进行人机交互的唯一手段，它的性能和稳定性对其使用有着重大的影响。

本文通过对嵌入式 GUI 系统研究现状及其发展趋势的分析，结合变电站自动化系统对嵌入式 GUI 的需求，提出了一种使用 Qt/Embedded 进行人机界面开发的方案。该方案采用以 ARM9 处理器为核心的嵌入式系统硬件平台，在此硬件平台上实现了 BootLoader 的移植、嵌入式 Linux 操作系统的移植以及 Qt/Embedded 的移植，最后使用 Qt/Embedded 作为开发工具完成了变电站通信控制器 GUI 应用软件的开发。经实际运行测试证明，该 GUI 系统运行稳定、性能良好，完全可以满足变电站通信控制器对 GUI 系统的要求，达到了预期目标。

变电站通信控制器 GUI 系统是一个庞大的体系，功能比较复杂。由于本人时间及精力的限制，仅实现了其基本功能，后面还有很多进一步的工作要做。具体内容包括以下几个。

1. 由于嵌入式系统的资源限制以及变电站通信控制器对其 GUI 系统的稳定性、可靠性和操作灵活性的要求，需要对 Linux 操作系统内核进行一定的裁减以及对应用程序代码进行优化。

2. 随着技术的发展，触摸屏由于其操作的方便性在嵌入式 GUI 系统中得到了越来越广泛的应用。我们需要进行触摸屏的支持以及使用触摸屏完成输入等方面的研究。

参考文献

[1] 黄益庄．变电站综合自动化系统技术[M]．北京：中国电力出版社，2000.

[2] 孟祥忠，王博．电力系统自动化[M]．北京：中国林业出版社，2006.

[3] 陈翌，田捷，王金刚．嵌入式软件技术开发[M]．北京：国防工业出版社，2003.

[4] 邹思轶．嵌入式 Linux 设计与应用[M]．北京：清华大学出版社，2002.

[5] 孙天泽，袁文菊，张海峰．嵌入式设计及 Linux 驱动开发指南[M]．北京：电子工业出版社，2005，9.

[6] 罗仕鉴，朱上上，孙守迁．人机界面设计[M]．北京：机械工业出版社，2002.

[7] 林甲威，孙未未，陈金海． 嵌入式 Linux 上的 GUI 应用的开发技术[J]． 微型电脑应用，2004，20(7).

[8] 徐广毅，张晓林，崔迎炜等．嵌入式 Linux 系统中 GUI 系统的研究与移植[J]．单片机与嵌入式系统应用，2004(12)：14～17.

[9] 魏永明．实时嵌入式 Linux 系统上 GUI 的发展与展望[J]．微电脑世界，2000(49).

[10] 张利敏，丁坚勇．嵌入式技术及其在电力系统中的应用[J]．继电器，2002，30(3)：43～47.

[11] Microwindows．Microwindows Paper[EB/OL]．http://www.microwindows.org.

[12] 苏东．主流 ARM 嵌入式系统设计技术与实例精解[M]．北京：电子工业出版社，2007.

[13] MiniGUI．MiniGUI Tech White Paper[EB/OL]．http://www.minigui.org.

[14] 章晓燕，马琪．嵌入式系统的 GUI——MiniGUI[J]．计算机与现代化，2005，1：10～12.

[15] Qt 中文网．Qt3 White Paper[EB/OL]．http://www.qtcn.org.

[16] 路广，张伯明，孙宏斌．嵌入式实时 Linux 及其在电网自动化系统中的应用[J]．电力系统自动化，2002，26(7)：62～65，69.

[17] 周维，陈默．基于 S3C2410 的 ARM 开发平台[J]．电子技术，2004(7).

[18] RubiniA．LINUX 设备驱动程序[M]．北京：中国电力出版社，2004.

[19] 孙海彬，傅谦，徐良贤．Linux 内核模块的实现机制[J]．微型机与应用，2000，

11(3)：32～36.

[20] 毛德操，胡希明著. LINUX 内核源代码情景分析[M]. 浙江：浙江大学出版社，2001.

[21] Lee C T, Rong Z W, Lin J M. Linux kernel customization for embedded systems by using call graph approach[C]. Design Automation Conference, 2001, 47(3): 33～37.

[22] 胡骏.Linux 环境下基于 QT 的 WLAN 管理信息系统[J].计算机工程，2004，30(3)：114～116.

[23] Arthur Griffith. KDE2 /Qt 编程宝典[M]. 北京：电子工业出版社，2002.

[24] Karim Yaghmour. Building Embedded Linux Systems[M]. O'Reilly，2003.

[25] Neil Matthew，Richard Stones. Beginning Linux Programming[M]. Wrox Press Ltd，2002.

[26] Alessandro Rubini. Linux Device Drivers 1st Edition[M]，O'Reilly，2004.

[27] Trolltech . Qt Reference Documentation(Open Source Edition) [EB/OL]. http://doc.trolltech.com / 4.1 /-index.html，2006.

[28] Free Standards Group. Linux Standard Base Specification[EB/OL]. http://www.linuxbase.org/spec.

[29] 张娟，张雪兰. 基于嵌入式 Linux 的 GUI 应用程序的实现[J]. 计算机应用，2003，23(4)：11～13.

[30] David J. Kruglinski. VC++技术内幕(第四版) [M]. 北京：清华大学出版社，2001.

[31] James D. Foley，Andries van Dam，Steven K. Feiner，et al(美). 计算机图形学原理及实践——C 语言描述[M]. 北京：机械工业出版社，2004，3.

[32] Michael Barr. C/C++嵌入式系统编程[M]. 北京：中国电力出版社，2001.

[33] Bruce Eckel. C++编程思想[M]. 北京：机械工业出版社，2000.

[34] 怀石工作室. LINUX 上的 C 编程[M]. 北京：机械工业出版社，2003.

[35] 谢春，陶烨，瞿坦等. 基于嵌入式 Linux 系统的多进程图形用户界面 GUI 系统研究[J]. 工业控制计算机，2003(05)：28～29.

[36] 贺鹏. 图形用户界面系统的层次结构模型[J]. 计算机应用研究，1997(02)：

23～25.

[37] 钱华锋，雷航. 面向对象的嵌入式 GUI 研究和模式应用[J]. 计算机应用，2004(04)，10～15.

[38] Trolltech. Qt Cross-platform C++ GUI Application Framework Technical Overview[EB/OL]. http://www.trolltech.com.

[39] Trolltech. Signals and Slots in Qt[EB/OL]. http://doc.trolltech.com/3.1/signalsandslots.html.

[40] Bart Broekman，Edwin Notenboom. 嵌入式软件测试[M]. 北京：电子工业出版社，2004.

攻读学位期间成果

1. 滕春涛. 基于 LPC2214 的μCLinux 内核的移植. 南京工业大学第九届研究生科技论坛，2007，4.

2. 滕春涛，黄冰，马新平. 嵌入式 Linux 文件系统的研究与应用. 微计算机信息(已录用).

3. 滕春涛，黄冰. 基于 CAN 总线的分布式建筑塔钟控制系统设计. 建筑电气(已录用).

致 谢

感谢我的导师黄冰老师对本论文及课题研究的精心指导和热情帮助，我研究生在读期间的每一点成绩都离不开黄老师的支持和鼓励。感谢黄老师给予我的谆谆教诲并为我提供了优良的科研条件，使我有机会参与科研项目并因此而得到不断的提高。在我硕士学业即将完成之际，谨向黄老师表达我深深的敬意和最诚挚的谢意。

我要感谢国电南瑞科技股份有限公司研发中心的马新平总工，他在我实习期间，从工作上和生活上都给予了我非常多的帮助，在论文的选题和撰写中给予了我很多指导，使我受益终生。同时感谢吴海、田小峰、白钟等几位工程师对我的帮助和指导。

感谢周计文、沈峰亭、沙峰、陆源、杨志飞等同学在学习中和生活上对我的帮助和鼓励。

感谢其他所有关心我、帮助我的师长和朋友们。

再次衷心的感谢并祝福你们！