苏州大学

硕士学位论文

嵌入式实时操作系统MQX的内核分析及应用研究

姓名：程玉娟

申请学位级别：硕士

专业：计算机应用技术

指导教师：王宜怀

201105

嵌入式实时操作系统 MQX 的内核分析及应用研究

中文摘要

2009 年，飞思卡尔半导体公司在中国市场上推出了应用于工业控制、汽车电子及消费电子的嵌入式实时操作系统 MQX。MQX 最早是 Precise Software Technologies 公司 1989 年开发，2009 年飞思卡尔收购后，开放源代码。MQX 采用微内核结构，具有标准的 API 接口、模块化架构、TCP/IP 协议栈和 USB 协议栈等，因此无论在实时性、裁减性还是移植性上都具有卓越的性能。鉴于国内对 MQX 的研究应用尚未开始，受飞思卡尔半导体公司委托，本课题主要从以下几个方面对 MQX 进行分析研究：

（1）研究 MQX 的微内核结构、内核的管理功能，并从 MQX 的启动执行、任务调度、设备驱动程序等方面，深入剖析 MQX 的运行实现机制。按照底层软件构件的思想，提出 MQX 工程框架组织结构，并重新编写设备驱动程序。

（2）对 MQX 的 FIFO 任务调度策略和 RR 任务调度策略进行分析研究，并把 FIFO 调度策略与 RR 调度策略结合起来，在 MQX 中设计并实现多级反馈队列调度算法，不仅使高优先级的任务得到响应又能使短任务迅速完成。

（3）把 MQX 移植到自动折弯系统的微控制器 MCF52223 上，详细阐述了移植步骤、移植过程中要修改和配置的文件，设计 LCD 触摸屏界面，完成了 MQX 上的自动折弯系统的软件平台设计。

采用 MQX 的自动折弯设备运行良好，有效的控制了折弯的角度、精确度等，并具有较高的实时响应能力。本课题完成了对 MQX 的分析、研究，对其在嵌入式控制领域的应用具有启发指导作用。

关键词：MQX、内核、任务调度、设备驱动程序、移植

作　　者：程玉娟

指导老师：王宜怀

Kernel Analysis and Application Research of Embedded Real-time Operating System MQX

Abstract

Embedded real-time operating system MQX that applies industrial control, automotive electronics and consumer electronics is launched by Freescale semiconductor in the chinese market in 2009. MQX was first developed by Precise Software Technologies Corporation in 1989. MQX opens source after Freescale's acquisition in 2009. MQX adopts micro-kernel structure, with standard API interface, module architecture, TCP/IP protocol stack, USB protocol stack and so on. So it has excellent performance in terms of real-time, transplant and reduction. In view of the domestic research and application of MQX not yet begin, commissioned by Freescale semiconductor, the paper analyzes and researches MQX mainly from the following aspects:

(1) Researches MQX micro-kernel structure and kernel management function, and deeply analyzes MQX running implementation mechanism from MQX's startup, task scheduling, device driver and so on. Puts forward MQX engineering framework structure by the idea of underlying software component, and rewrites device driver.

(2) Analyzes and researches FIFO scheduling policy and RR scheduling policy of MQX, combines FIFO scheduling policy with RR scheduling policy, designs and implements multi-level feedback queue scheduling algorithm in the MQX, which not only responds the task with high priority but also makes the short task done quickly.

(3) Transplants MQX to the automatic bending system microcontroller MCF52223, elaborates transplant procedure, files to modify and configure in the transplant process, designs LCD touch screen interface and completes the software platform design of automatic bending system on the MQX.

Automatic bending equipment based on MQX runs well, effectively controls angle and accuracy of bending, and possesses high real-time response capability. The paper completes analysis and research of MQX, which has heuristic instructional function for its study and application in the field of embedded control.

Key words: MQX, Kernel, Task Scheduling, Device Driver, Transplant

Written by ChengYujuan

Supervised by WangYihuai

第一章 绪 论

MQX是飞思卡尔半导体公司2009在国内推出的一款RTOS，其具有源码开放、可裁剪性强、占用ROM空间少等特点，具有巨大的市场前景和应用价值。但由于其在国内推出时间短，应用研究资料很少，仅有少量开源代码，但不规范，可重用性和可移植性也没有保障。受飞思卡尔公司委托，本文对MQX进行全面深入的分析和研究，分析其内核结构、应用实现机制、工程框架组成，并以自动折弯系统作为应用实例介绍其具体移植过程，从而达到其在国内推广和应用的目的。

作为全文的引导，本章首先介绍RTOS的发展历程、功能和基本概念，并对当前主流RTOS进行比较分析，得出其优缺点及应用领域；接着对MQX进行简介，阐述其版本升级过程和卓越的特性优势；然后阐述本课题的课题背景和研究依托；最后给出本文所做的主要研究工作和课题意义。

1.1 嵌入式实时操作系统

嵌入式系统的飞速发展，使其广泛应用到工业、农业、军事、通信、运输、金融、医疗、气象等众多领域[1]。随着嵌入式的应用领域不断扩大，嵌入式系统硬件规模越来越复杂，功能越来越强，对硬件实时性、稳定性要求更高。因此，嵌入式系统的软件规模也不断扩大，为了对这些软件进行有效的管理，逐步形成了嵌入式实时操作系统（Embedded Real-Time Operating System，ERTOS或RTOS）。

1.1.1 RTOS的发展历程

嵌入式实时操作系统从20世纪70年代产生，到今天已经有40年的历史。在这40年中，RTOS迅速发展，种类数目也急剧增长。到目前为止，有数十家RTOS公司，200多种RTOS[2]。RTOS的发展历程如下：

（1）20世纪70年代后期，专用嵌入式系统的操作系统才开始出现。当时的大多数RTOS是用汇编语言编写的，仅能用于特定的微处理器。若要更换处理器，必须要重新编写RTOS。直到C语言出现后，RTOS才使用一种高效、稳定和可移植的方式来编写[2]。

（2）1981年，美国Ready System公司开发了世界上第一个商业嵌入式实时内核（VRTX32）[3]。20世纪80年代的RTOS只有内核，以销售二进制代码为主。当时

的 RTOS 除 VRTX 外，还有 IPI 公司的 MTOS 和 80 年代末 ISI 公司的 PSOS。嵌入式产品只支持一些 16 位的微处理器（如 68k、8086），主要用于军事和电信设备[4]。

（3）20 世纪 90 年代初期，开始出现现代操作系统的设计思想，如微内核设计技术和模块化设计思想，并渗入到 RTOS 领域，产生了大量新一代的实时内核。同时，各家公司都力求摆脱完全依赖第三方工具的制约，通过自己收购、授权或使用免费工具链的方式，组成一套完整的开发环境[4]。

（4）20 世纪 90 年代中期，随着互联网逐渐深入人们的日常生活，要求 RTOS 具有网络和图形界面的功能。为了方便使用大量现存的软件代码，RTOS 都支持标准的应用程序编程接口 API（Application Programming Interface，API），如可移植操作系统接口 POSIX（Portable Operating System Interface of Unix，POSIX）、Win32 等。同时，RTOS 的开发环境也与 UNIX、Windows 等趋于一致。

1.1.2 RTOS的功能

RTOS 是一种实时的、可裁剪的、可固化的、可移植的、支持嵌入式系统应用的操作系统软件，通常包括与硬件相关的底层驱动程序、系统内核、设备驱动接口、通信协议、图形界面等[5]。RTOS 在系统实时高效性、硬件的相关依赖性、软件固态化以及应用的专用性等方面具有较为突出的特点[6]。

RTOS 为每个任务建立一个可执行的环境，可以很方便地在任务之间传递消息，在中断服务程序（Interrupt Service Routine，ISR）和任务之间传递事件，区分任务执行的优先级，并协调多个任务对同一个 I/O 设备的调用。RTOS 应具有如下的功能：

（1）包含多任务，任务管理。

（2）任务间的通信与同步。

（3）存储器优化管理。

（4）时钟管理服务。

（5）中断管理服务。

1.1.3 RTOS的相关基本概念

1. 任务

任务是 RTOS 的一个调度单位。在软件设计时，根据应用划分出的独立的、相互作用的程序集合，每个程序集合执行时就称为任务。

2. 调度

调度是内核的主要职责之一，决定该轮到哪个任务运行了。

3. 核心态和用户态

核心态执行全部指令（即操作系统指令），用户态执行非特权指令（即应用程序指令）。特权指令：只能由操作系统内核使用，不允许用户直接使用。非特权指令：所有程序均可直接使用。

4. 硬实时与软实时

硬实时要求在规定的时间内必须完成操作，是在设计操作系统时保证的；软实时则没有那么严格，只要按照任务的优先级，尽可能快地完成操作即可[9]。

5. 任务优先级

在一个多任务系统中，每个任务都有一个优先级。在任务优先级驱动的系统中：（1）正在运行的任务总是具有最高优先级的任务；（2）在任何给定的时间，总是把微处理器分配给最高优先级的就绪任务。

6. 中断

中断是一种硬件机制，用于通知 CPU 发生了一个异步事件。CPU 一旦识别出一个中断，保存任务上下文后，跳到该中断服务程序 ISR 执行，处理完这个中断后，返回到所有就绪态任务中具有最高优先级的任务执行。

7. 上下文切换

当多任务内核决定运行不同任务时，当前任务的上下文（即 CPU 寄存器）保存在它的堆栈中，新任务的上下文从它的存储空间中恢复，然后重新执行新任务的代码，这个过程称为任务切换或上下文切换。

8. 内核是否可抢占

任务一旦占用 CPU，是否允许被其他任务打断而让出 CPU。若允许，称任务是可抢占的；否则，直到任务自身放弃 CPU 时，系统才可以调度，称任务为不可抢占。

9. 任务间通信

在异步环境下的一组并发进程因直接制约而互相发送消息而进行互相合作、互相等待，使得各进程按一定的速度执行的过程称为进程间的同步和通信，主要方式包含：

信号量、事件、互斥、消息队列等。

10. 优先级逆转

当一个任务等待比它优先级低的任务释放资源而被阻塞时，产生优先级逆转，即优先级低的任务阻塞了优先级高的任务。

1.1.4 当今流行的RTOS比较

一般来说，嵌入式系统的应用范围可以粗略分为两大类：一类是电子系统的智能化（如工业控制、现代农业、家用电器、汽车电子、测控系统等）；另一类是计算机应用的延伸（如 MP4、手机、通信、网络、计算机外围设备等）[7]。虽然当前的 RTOS 种类繁多，但其应用范围主要在上述两个方向延伸。电子系统智能化方向的主流 RTOS 有：VxWorks、μC/OS-II、μClinux（micro-control Linux）、QNX、eCos（Embedded Configurable Operating System）等[8]-[13]；计算机应用的延伸方向的主流 RTOS 有：Windows CE、Android、Symbian 等[14]-[16]。表 1-1 对当前主流 RTOS 进行分类比较。

表 1-1 当前主流 RTOS 优缺点比较

名称	内核来源	优点	缺点	应用领域
VxWorks	美国 WindRiver 公司	由 400 多个相对独立、短小精悍的目标模块组成，用户可适当裁剪模块以配置系统，有较高实时性、安全性和可靠性。	需要付费，昂贵的价格让开发者望而却步。	应用在通信、军事、航空、航天等高尖技术及实时性要求极高的领域。
μC/OS-II	美国人 Jean Labrosse 于 1992 年完成	结构小巧、抢占式的多任务实时内核。能够管理 64 个任务，具有执行效率高、占用空间小、实时性能优良和可扩展性强等特点。	文件系统差，需自己编写移植和驱动程序。	适合对系统要求不苛刻，且 RAM 和 ROM 有限的小型嵌入式系统设备，以及入门学习。
μClinux	Linux 的一个嵌入式版本	具有良好的稳定性和移植性、强大的网络功能、出色的文件系统支持、标准丰富的应用程序编程接口 API，以及 TCP/IP 网络协议等。	没有 MMU，多任务实现需要一定技巧。	适合开发对事件要求不高的小容量、低成本的各类产品，及与网络应用密切相关的嵌入式设备。
QNX	加拿大 QNX 公司的产品	实时、稳定、可靠、建立在微内核和完全地址空间保护基础之上的实时操作系统，具有模块化程度高、剪裁自如、易于扩展的特点。	需要付费。	主要应用在汽车、军工、自动化、医疗设备和高端网络系统等领域。
eCos	由 GNU 开源开发工具支持	配置灵活、采用模块化设计，核心部分由相同的组件构成，每个组件可提供大量的配置选项。	内核的可选择性和可配置性存在缺点。	面向深度嵌入式应用，适合用于商业级或工业级对成本敏感的应用。
WinCE	微软针对小型设备设计的 OS	高度模块化，可裁剪，开发平台使用 Visual Studio，简单易学。	实时性稍差。	用在 PDA、手机、显示仪表等界面要求较高或者要求快速开发的场合。
Android	Google 推出的基于 Linux 的开源手机操作系统	基于 Linux 平台的开源操作系统，功能强大，该平台由操作系统、中间件、用户界面和应用软件组成。	实时性稍差。	智能电话、手机等。
Symbian	移动通讯设备商研发的手机操作系统	采用微内核、非抢占式实现，在智能移动终端上拥有强大的应用程序以及通信能力。	多媒体方面差，系统兼容性差。	智能移动终端。

1.2 MQX 简介

消息队列执行（Message Queue eXecutive，MQX）是 Precise Software Technologies 公司 1989 年开发的一款嵌入式实时操作系统。2000 年 3 月，MQX 被 ARC 公司收购，并在新的处理器体系中[17]（主要包含：Freescale 的 ColdFire 系列、IBM®/Freescale 的 PowerPC、ARM、ARC 和 i.MX 等[18]）继续开发。2009 年，飞思卡尔半导体公司出售 ColdFire MCU 时，附送 MQX，并在官方网站上提供开源源代码，使其成为开源 RTOS。MQX 的应用主要是向电子系统的智能化方向延伸。

1.2.1 MQX的版本升级

从 2009 年，MQX 推出第一个版本 RTOS 3.0.1 后，其版本不断进行升级和更新，功能不断加强，目前推出的最新版本号是 3.6.2。下面详细阐述其版本在升级过程中做的改进和添加的功能[19]-[21]，如表 1-2 所示。

表 1-2 MQX RTOS 的版本升级

版本号	推出时间	添加功能
3.0.1	2009,1	（1）为系统分配的存储器添加了内存块"类型"信息。CodeWarrior 中的任务调试插件（Task-Aware Debugging，TAD）能给出内核或系统组件分配的存储器块的详细信息。 （2）为板级支持包（Board Support Package，BSP）添加了如下驱动：微小文件系统（Trivial File System，TFS），I2C（Inter Integrated Circuit）驱动，FlexCAN 驱动，实时时钟（Real-Time Clock，RTC）驱动，串口驱动，以太网驱动，通用 I/O（General Purpose I/O，GPIO）驱动，PC 卡驱动，PC 闪存驱动。 （3）设置实时 TCP/IP 通信套组（Real-Time TCP/IP Communication Suite，RTCS），MQX 文件系统（MQX File System，MFS）和通用串行总线（Universal Serial BUS，USB）对具有有限 RAM 资源的设备是可以优化的。 （4）RTCS 中的 HTTP 服务器能够由一个单一的任务担任多个会话。 （5）添加新的 USB 主机 HUB 类。HUB 操作对用户应用程序是完全透明的。 （6）把单独的底层设备驱动、主机开发套件库和 USB 类库工程结合成为 USB 主机开发套件（Host Development Kit，HDK）。
3.1.0	2009,4	（1）存储器类型信息添加轻量级内存结构。 （2）改变简单网络管理协议 SNMPv2（Simple Network Management Protocol，SNMP）代码为能够基于 ROM 管理信息库（Management Information Base，MIB）的结构。 （3）添加了模数转换（Analog-to-Digital Converter，ADC）和串行外设接口（Serial Peripheral Interface，SPI）驱动。
3.2.1	2009,6	（1）重写以太网驱动，使其能够适用于 ColdFire V1 和 V2 处理器。 （2）移植 MQX Flash 驱动程序（称为 FlashX）用来支持内部 Flash 设备。
3.3.0	2009,8	（1）重写以太网驱动的设备相关部分，用来支持 ColdFire V1-V4 的快速以太网控制器（Fast Ethernet Controller，FEC）模块、多重介质访问控制层（Medium Access Control，MAC）设备。 （2）Shell 实例使用 MFS 库，并支持所有的文件传送输协议（File Transfer Protocol，FTP）命令。 （3）USB 主机更新：代码重构，更新大容量存储器件（MASS Storage Device，MSD）和通信器件类（Communication Device Class，CDC）。

3.4.0	2009,10	（1）实现对 USB 增强型主机控制器接口（Enhanced Host Controller Interface，EHCI）的支持。 （2）支持个人医疗器件类（Personal Hospital Device Class，PHDC）和处理器间通信（Inter-processor communication，IPC）。 （3）重写 SPI 驱动，并支持 SPI、QSPI 和 DSPI，并添加基于 SPI 的 SD 卡驱动程序。
3.5.0	2010,2	（1）添加定时基准测试应用程序，用来测量 MQX 内核的关键定时参数。 （2）SPI 的示例应用程序的测试从 EEPROM 扩展到一般 SPI 存储器（EEPROM，闪存或串行的 MRAM）。 （3）ENET MAC 接口结构扩展到支持通用和设备特定的控制命令（即媒体控制命令）。
3.5.1	2010,4	（1）代码大小的基准测试程序扩大到覆盖的 ColdFire V1-V4 平台。 （2）添加新驱动：嵌入式高容量 SD 存储卡（Embeded Secure Digital High Capacity，ESDHC）驱动，数模转换（Digital-Analog Converter，DAC）驱动，NAND Flash 驱动。
3.6.0	2010,6	（1）支持 CodeWarrior 10。 （2）支持嵌入式图形用户界面 eGUI（embedded Graphical user interface，eGUI）。 （3）添加电阻式触摸屏驱动，使用 UDP 实现处理器间通信 IPC。
3.6.2	2010,11	（1）为 Freescale Kinetis ARM® Cortex M4 PSP 做准备。 （2）使 ARM 的 IAR 嵌入式工作台支持飞思卡尔 MQX Kinetis 端口。 （3）增强 RTC 驱动程序的 API，修改 I/O 驱动程序代码，添加带有硬件时间戳的新 MAC-NET 以太网驱动程序来支持 Kinetis 平台。

1.2.2 MQX的特性

不像通用操作系统设计的桌面系统，MQX 是专为嵌入式系统的速度和规模效益而设计的。MQX 是面向应用的、专用特制的嵌入式实时操作系统。它除具有处理多任务、文件、设备驱动等基本的操作系统功能之外，还具有如下特性[22]：

（1）开放源码，成本低，软件资源丰富。

MQX 的内核源码可在 Freescale 官方网站免费下载，有专业人员提供技术支持。开放源码可以使用户不用从头做起，节省时间，降低费用。同时，在 MQX 安装目录下，提供了大量的应用实例，软件资源丰富，加快项目开发速度。

（2）采用微内核结构，系统体系结构具有可伸缩性、可裁减性[23]。

MQX 采用微内核结构，使用最小内核处理集，系统开销小，运行效率高，可以根据需要添加可定制组件，比较容易适用于各种嵌入式系统应用。

（3）高实时响应，内核实时效率高。

MQX 采用基于优先级的、抢占式调度策略。带有最优化上下文切换和中断处理，用于实现快速、高效的预测响应时间，具有高实时性。

（4）具有直接应用编程接口 API、高度模块化架构、MS-DOS 文件系统（MS-DOS File System，MFS），能够很好地满足各种不同应用需求。

（5）快速网络功能[24]。

使用 RTCS 协议支持统一的 MAC 访问层接口，提供 TCP/IP 协议栈、FTP、Telnet、DHCP、SNMP、DNS、HTTP 等服务。

（6）高速 USB 功能[25]。

USB 包含 USB 主机和 USB 设备。支持以下类别：个人医疗器件类应用 PHDC、人机交互设备 HID、大容量存储器件 MSD、通信器件类 CDC、HUB 等。

1.3 课题背景及研究依托

1.3.1 课题背景

随着信息化、智能化、网络化的发展，嵌入式系统技术获得了广阔的发展空间，在很大程度上改变了人们的生活、工作和娱乐方式，而且这些改变还在加速。嵌入式系统广泛的适应能力和多样性使其广泛地应用于生产、生活的各个领域。目前，各种新型的嵌入式系统设备的应用领域和数量已经远远超过了通用计算机，随着嵌入式系统设备的广泛应用，对嵌入式系统的功能、实时性、可靠性和稳定性的要求越来越高，因此，嵌入式实时操作系统应运而生。

到目前为止，RTOS 已经历了 40 年的发展历程，涌现出一批著名的 RTOS，如 VxWorks、μC/OS-II、μClinux、Windows CE、Android 等。如表 1-1 所示，这些操作系统要么用于商业领域，价格昂贵；要么为非开源代码，不利于开发人员进行学习、修改和升级；要么为开源代码，但兼容性差，需要开发人员编写模块驱动程序。因此，对于一般的工业控制来说，这类操作系统就失去了竞争力，选用一种功能能够满足实际需求、价格低廉的嵌入式操作系统成为人们的共识。

1.3.2 研究依托

鉴于国内 RTOS 存在种类繁多、架构复杂、应用方向不明确、入门难度大、成本高等局限性的现状，飞思卡尔半导体公司委托苏州大学飞思卡尔 MCU&DSP 实验室，以降低 RTOS 学习入门难度，加快产品研发速度，节省产品成本，提供高灵活性、高连通性、高成本效益以及低功耗的解决方案为目标，联合在国内推广适合中国市场的 RTOS MQX，课题以此为蓝本，受飞思卡尔半导体公司委托对 MQX 进行研究。

1.4 主要研究内容及意义

在国外，MQX 广泛应用于工业控制、消费类电子和汽车电子等方面；但国内对其研究、应用较少。MQX 采用微内核结构，具有完全开放的源代码，良好的可裁减

性和可移植性，标准丰富的 API 接口和模块化架构，以及 TCP/IP 协议栈等[22]。无论是在实时性、稳定性还是在移植性上，MQX 都具有卓越的性能。

1.4.1 研究内容

本课题主要从以下几个方面对 MQX 进行研究：

（1）剖析 MQX 的内核，对 MQX 微内核结构的组成进行研究。从 RTOS 的内核功能要求，对 MQX 的任务管理、时间管理、中断管理、存储管理和任务同步与通信等进行分析、研究。

（2）研究 MQX 的实现机制。首先，对 MQX 的启动执行过程进行跟踪分析、研究。其次，分析 MQX 的内核和任务调度方法和策略，详细阐述了如何设置 MQX 的内核和任务调度，并在 MQX 中引入了多级反馈队列调度以解决 FIFO 任务调度策略和 RR 任务调度策略的缺点。最后，归纳总结如何编写 MQX 的设备驱动程序，以及如何把设备驱动程序添加到 MQX 中。

（3）分析研究 MQX 安装目录的组成，以及各目录的作用。基于底层软件构件思想，对 MQX 的工程结构进行安排和调整，并以串口控制小灯作为实例，详细阐述了底层软件构件的编写方法和编写过程中的注意点。

（4）实现了 MQX 到自动折弯系统上的移植，并详细阐述了移植过程的实现。在自动折弯系统中，完成 LCD 触摸屏的界面设计，并对自动折弯系统中的任务进行划分和设计，添加中断设备驱动程序。

1.4.2 课题意义

由于目前硬件的飞速发展，系统速度越来越快，RTOS 在嵌入式系统中的作用越来越重要。目前的国内主流 RTOS，在不同方面或多或少存在一定的缺点，针对这一状况，受飞思卡尔半导体公司的委托对 MQX 进行研究，本课题对 MQX 的内核进行剖析，分析研究 MQX 的启动执行过程，驱动程序添加，以及如何进行移植等。通过对 MQX 的分析研究，方便初学者完成对 MQX 的入门学习，掌握 MQX 的实现机制，加快项目研发速度，节省研发时间。同时，也有利于学习者以 MQX 为蓝本，快速掌握其他 RTOS。从某种程度上讲，本课题对 MQX 在国内的推广及应用具有深远意义。

1.5 论文章节安排

本文共分为六章，各章内容安排如下：

第一章为绪论部分。首先介绍 RTOS 的发展历程、功能和基本概念，提出了 RTOS 的划分方法，根据该方法对当前主流 RTOS 进行比较分析；接着对 MQX 进行简介，阐述其版本升级过程和相对于其他 RTOS 的卓越性能；然后阐述本课题的课题背景和研究依托；最后给出本文所做的主要研究工作和课题意义。

第二章为 MQX 内核分析部分。首先深入分析研究了 MQX 的微内核结构的组成，从核心组件和可选组件两方面，论述了 MQX 各组件的功能；接着讲解了 MQX 的任务模板列表、任务状态转换、任务切换过程和任务管理的相关函数，从而说明 MQX 的任务管理功能；然后阐述 MQX 的内核的时间管理功能、存储管理功能、中断管理功能；最后从信号量、事件、互斥、消息和任务队列等方面讲解了 MQX 的任务同步和通信机制的实现。

第三章为MQX的实现部分。首先介绍了MQX的启动执行过程，详细阐述了系统上电启动后，对微控制器及其内部功能模块进行初始化的过程，分析了Bootloader的加载和启动执行过程，直到系统从自启动任务开始运行；接着对MQX的内核调度策略和任务调度类型进行讲解、分析，为了改善MQX任务调度策略的缺点，在MQX中引入多级反馈队列调度策略；最后分析研究了MQX的设备驱动程序的设计与实现。

第四章为 MQX 的工程框架部分。首先阐述了 MQX 的开发环境 CodeWarrior 版本类型，对 CW 开发环境的安装过程进行讲解，分析安装过程中的注意点；接着论述 MQX 的安装框架，详细说明其安装目录下的各目录的作用和内容；为了提高软件的可重用性、可移植性对 MQX 工程的逻辑组织结构和物理组织结构进行重新归类、设计，按照底层软件构件的思想对 MQX 的底层硬件驱动程序进行重新编写，并以串口控制小灯程序为例说明底层软件构件的编写方法。

第五章为应用实例部分。首先设计并实现在MQX上的自动折弯系统的软件平台；接着把 MQX 移植到自动折弯系统的微控制器 MCF52223 上，详细阐述了移植步骤、移植过程中要修改和配置的文件，论述移植过程中的关键技术；然后设计系统中的 LCD 触摸屏界面；最后提出自动折弯系统中任务划分、设计的依据，完成自动折弯系统的任务和中断程序设计。

第六章为总结和展望部分，对本文进行了总结，并进一步提出了一些尚待研究和实现的部分。

第二章 MQX 内核分析

MQX 是一个公开源码、可移植性强的多任务 RTOS，以其良好的持续发展能力、可定制的微内核结构以及友好的用户开发环境，必将使其迅速在 RTOS 领域占据一席之地。

多任务系统中，内核负责管理各个任务，为每个任务分配 CPU 时间，并且负责任务之间的通信，它是整个操作系统的基础。本章从 MQX 的微内核体系结构、任务管理的实现、时间管理、中断管理、存储管理、任务同步与通信等方面对 MQX 操作系统的内核特性和功能作了全面的阐述。

2.1 MQX 系统结构

MQX 采用微内核结构设计，包含核心组件和可选组件，用户可根据自己的需求对内核进行定制，选取所需组件，以满足非常小的内存需求。

2.1.1 微内核结构分析

微内核是一个小型操作系统的核心，它为模块化、构件化扩展提供了基础。微内核结构分为内核级服务和用户级服务[26]。微内核仅将操作系统最基本的功能放在核心态的内核中，其它的服务和应用程序在微内核之上进行构造，运行在用户态。微内核结构采用水平分层结构，内核中仅仅保留操作系统最基本的功能，其它服务和应用程序都在微内核基础之上构造，并在用户模式下执行。微内核外部的操作系统部件借助于微内核的消息传递机制实现交互[27]。微内核结构如图 2-1 所示。采用组件化的设计思路，将内核划分为非组件部分和组件部分。这样设计有两个好处：其一是增强了内核的可扩展性和灵活性；其二是给内核各个模块在开发和测试的过程中带来方便，从而在局部和整体上更好地保证了系统的可靠和稳定。

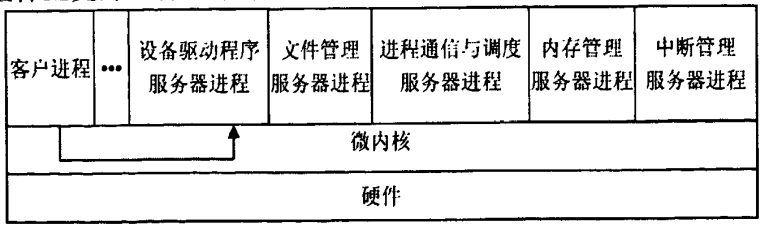


图2-1 微内核结构框图

微内核的优点是易于扩展和移植。它具有强大的可扩展性，修改内核原有功能和

增加新的服务时不需重新建构内核，因此实现时比较容易。另外微内核结构还可以方便地去除某些现有服务，以产生一个更小、更有效的应用。微内核结构的操作系统具有根据实际需要选取各种可选功能服务的特性，对于强调可剪裁特性的嵌入式应用来说具有很大的吸引力；微内核结构易于移植，它提供了对多种处理器代码的良好封装。移植到某一特定平台上时，只需要修改内核中很少量的代码即可。

微内核的主要缺点是性能问题。由于消息的发送和接收都是通过系统调用进行的，而且内核中还需要对消息进行解释，这要比普通的系统调用代价大很多。但是，这个问题可以通过两种办法解决：一种是有选择地把一些对于系统性能影响很大的子系统集成到微内核中，这样可以减少用户模式到内核模式切换的次数以及地址空间进程切换的次数；另一种方法是通过正确的设计使得微内核变小，一个非常小的微内核可以消除性能损失并提高可靠性与灵活性。

2.1.2 MQX的内核

MQX系统结构采用微内核结构设计，由核心组件（必选）和可选组件构成。对于核心组件，只有MQX或调用应用程序的函数包含在映像中。为满足要求，应用程序可通过加入可选组件来扩展和配置核心组件。图2-2为MQX的系统组成结构[22]，位于中心的是核心组件，是必选的；位于边缘的是可选组件。MQX的层次结构图，参见附录A。

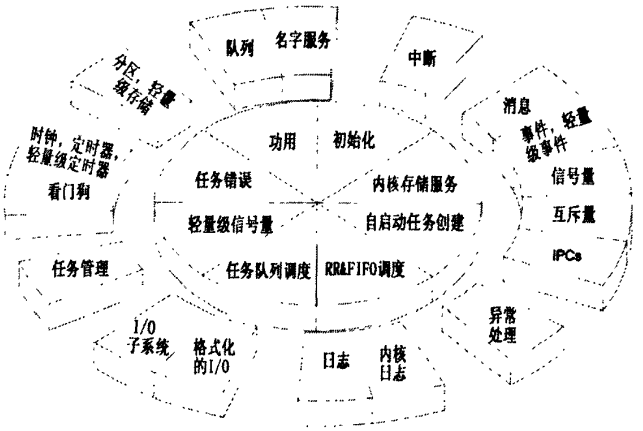


图2-2 MQX体系组成结构

图2-2显示了MQX的微内核的结构，下面详细说明核心组件和可选组件所包含的内容，以及它们在MQX系统中的作用，如表2-1所示。

表 2-1 核心组件和可选组件

类型	组件	内容及作用
核心	初始化	初始化硬件和创建自启动任务。
	任务管理	动态任务管理。创建、管理和终止任务，动态改变任务属性。
	调度	轮询（RR）调度和先进先出（FIFO）调度。
	任务同步和通信	轻量级信号量。对于实现共享资源同步存取的任务而言，这是一种低开销方式，需要很少的存储空间，运行速度快。
	存储管理	可变大小的存储块，用于分配和释放可变大小的内存块。
	检测工具	栈的运用。动态检查中断栈和任务栈，判断是否分配了足够的栈空间。
	错误处理	任务错误代码，异常处理，实时测试。
	队列操纵	实现队列元素双向链表结构，用于初始化队列、插入、移除和读取队列元素。
可选	任务管理	显式使用任务队列，使用任务队列明确地调度任务或创建相对复杂的同步机制。
	任务同步和通信	信号量、事件、互斥、消息、任务队列。
	处理器间通信	一个应用程序能够在多个处理器上同时运行，每个处理器的 MQX 可执行映像通过消息进行通信和协作，这些信息通过内存或处理器间的通信链传输。
	检测工具	内核日志：用于记录 MQX 活动状态，可在指定的位置或者 MQX 选择的位置创建或配置内核日志。 日志：保存和恢复应用程序。 轻量级日志：使用固定大小的记录，比普通的应用程序日志速度快。
	定时	轻量级定时器：为周期性调用应用函数提供低开销机制。 定时器：提供周期执行应用程序的功能。 看门狗：帮助用户检测任务级别的崩溃和死锁情况。
	存储管理	固定大小的存储块（区块）、存储器管理单元（Memory Management Unit，MMU）、高速缓存和虚拟存储控制、轻量级存储管理。
	名字组件	提供一个名称库，与动态定义的标量（如队列 ID）一一对应。
	嵌入式调试	EDS 服务器。它与运行在 EDS 客户端的主机通信，读写 MQX 信息，从主机获取各种配置信息。
可选（BSP）	定时	时间组件：可在 BSP 级允许或禁止它。时间有两种：消逝时间和绝对时间。
	中断处理	调度中断服务程序对中断进行处理。
	I/O 驱动	I/O 子系统：即 I/O 设备驱动程序。 格式化的 I/O：是 I/O 子系统的应用程序接口 API。

2.2 任务管理

任务（Task）是 RTOS 中最重要的操作对象，是用户的一个具体应用程序，是系统的基本组成元素，系统运行时的独立单元。在某个时刻，只有一个任务处于激活状态，被 CPU 执行。在设计一个较为复杂的应用程序时，通常把一个大任务分解成多个小任务，每个小任务都是整个应用程序的一个部分，具有一个任务 ID、一个优先级、一个任务控制块 TCB 和任务堆栈，以及一个任务实现函数[28]。

从应用程序设计的角度看，RTOS 的任务就是一个线程，就是解决用户问题的 C 语言函数和与之相关联的一些数据结构而构成的一个小实体[29]。在具体设计时，任务是一个无限循环体，运行时根据具体情况在不同任务状态之间来回切换。

2.2.1 任务模板列表

MQX通过任务模板列表（由任务模板结构TASK_TEMPLATE_STRUCT组成的一个列表）定义一组初始化模板（必须包含一个自启动的任务，即任务属性为MQX_AUTO_START_TASK），基于该模板任务可以在处理器上创建。初始化时，MQX查找任务模板的自启动任务并执行，当应用程序运行时，它能按任务模板（由任务模板定义或应用程序动态定义）生成其它任务。下面给出任务模板结构定义：

```
typedef struct task_template_struct
{
    _mqx_uint   TASK_TEMPLATE_INDEX;          //模板索引
    void (_CODE_PTR_ TASK_ADDRESS)(uint_32);  //任务函数
    _mem_size TASK_STACKSIZE;                 //任务栈大小
    _mqx_uint TASK_PRIORITY;                  //任务优先级
    char _PTR_ TASK_NAME;                     //任务函数名称
    _mqx_uint TASK_ATTRIBUTES;                //任务属性
    uint_32 CREATION_PARAMETER;               //任务参数
    _mqx_uint DEFAULT_TIME_SLICE;             //默认时间片
};
```

从同一任务模板列表生成的多个任务可以并存，而且每个任务是一个独特的实例。MQX保存了每个实例的上下文环境，包括程序计数器、寄存器和堆栈。每个任务有一个唯一的32位任务编号ID，MQX和其它任务通过该编号来区分不同的任务。

在MQX中，优先级为0的任务，具有最高的优先权，该任务运行时屏蔽所有中断。优先级为6以下的任务，意味着其能够屏蔽确定等级的中断，所以用户任务的优先级为7或以上。

任务属性用于决定任务的调度方式，一个任务的任务属性可以是几种属性的任意组合。主要包含以下几种属性：

（1）自启动——当MQX开始运行时，它生成任务的一个实例。

（2）数字信号处理（Digital Signal Processing，DSP）——MQX保存DSP协处理器寄存器作为任务上下文的一部分。

（3）浮点——MQX保存浮点寄存器作为任务上下文的一部分。

（4）时间片——MQX对任务使用轮询调度（默认是FIFO调度）。

2.2.2 MQX的任务状态

一般的OS具有三种任务状态：就绪态、运行态和阻塞态[30]。MQX中的任务共有四种状态，分别为：阻塞态、激活态、就绪态和终止态。在任一时刻，任务一旦创

建后所处的状态一定是四种任务状态之一，任务状态转换如图 2-3 所示[31]。

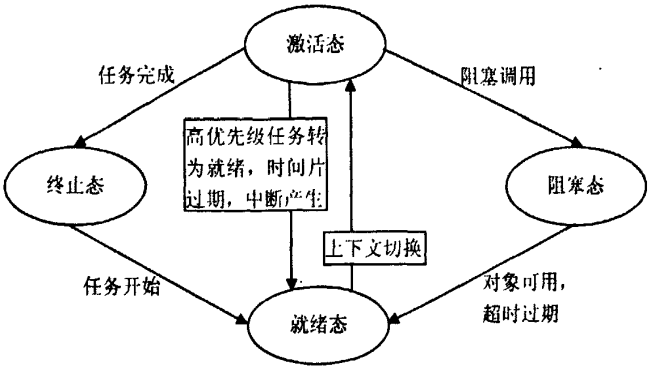


图2-3 任务状态转换图

当任务处于不同的状态时，其活动程度不同。下面按照活动程度由低到高的顺序，对 MQX 的任务状态进行介绍：

（1）终止态（Terminated）：任务已完成、或任务被破坏，不再需要使用 CPU。

（2）阻塞态（Blocked）：任务由于某种原因等待一个事件的发生，如等待其它任务放弃共享资源，自身延时一段时间等都会将任务变为此种状态，直到事件发生或是等待的时间超过用户指定的时间为止[30]。当条件具备时，任务就变为就绪态。

（3）就绪态（Ready）：任务已经就绪，但没有运行，因为不是最高优先级的就绪任务。处于这种状态的任务可以被调度成为激活态。

（4）激活态（Active）：任务已经就绪，并正在运行。在单 CPU 中，在某一时刻仅有且只有一个任务处于激活态，而且就绪态中优先级最高的任务才能进入激活态。

2.2.3 MQX任务切换

在 OS 中，上下文切换增加了应用程序开销。CPU 拥有的寄存器越多，开销越大。上下文切换的执行时间由 CPU 保存和恢复的寄存器数目和数据决定。MQX 执行上下文切换时，保存程序计数器、寄存器和堆栈。

MQX 任务切换执行过程如下：

（1）当任务调用阻塞函数、时间片到，或更高优先级的任务处于就绪态，或产生中断时，处在激活态的任务停止运行。

（2）保存该任务的上下文。

（3）将该任务放在就绪队列或等待队列末尾。

（4）使就绪队列中优先级最高的任务进入激活态。

2.2.4 MQX任务相关函数

任务管理是RTOS的核心，MQX的任务操作函数很多，包括建立任务、删除任务、任务的挂起和恢复、任务调度等功能。下面简要介绍主要任务操作函数[22][32]。

（1）创建任务

在多道程序环境中，只有任务才能在系统中运行。因此，为使程序能运行，就必须为它创建任务。创建任务时，必须将任务的起始地址与其他参数一起传给任务建立函数，将任务提交给内核进行管理[33]。

MQX中的任何任务均可以通过调用_task_create()或者_task_create_blocked()创建子任务，并传递处理器编号、任务模板索引和任务创建参数。_task_create()函数将子任务挂入任务就绪队列。_task_create_blocked()函数创建一个被阻塞的任务。该任务将保持非就绪态直到另一个任务调用_task_ready()函数。

（2）获取任务IDs

任务能够使用_task_get_id()函数直接获取其任务ID。任务能够使用_task_get_creator()函数直接获取其创建者的任务ID。

此外，任务ID还可以根据创建该任务的任务模板中的任务名称来确定，通过_task_get_id_from_name()函数实现，返回的是在任务模板列表中第一次匹配该任务名称的任务ID。

（3）获取修改任务优先级

_task_get_priority()函数用于获取任务优先级，_task_set_priority()用于设置或修改任务优先级。

（4）获取和设置调度方式

_sched_get_policy()函数用于获取任务的调度方式，_sched_set_policy()函数用于设置或修改任务调度方式。

（5）任务抢占

处于激活态的任务可以被其他任务抢占。当高优先级的任务就绪，并被激活，这样将抢占原激活任务的处理器。当中断处理器使一个高优先级的任务就绪，或是一个激活任务使另一个高优先级任务就绪时，也会发生任务抢占。

当任务处于运行状态时，如果调用了_task_stop_preemption()，那么该任务的处理器将不能被抢占。换句话说，即使在此时由某中断使得一个或几个更高优先级任务就绪，处于运行状态的这个任务的处理器也不会被抢占，直到此任务调用

_task_start_preemption()或自身调用操作系统函数将自己挂起或使高优先级任务就绪。

（6）管理任务错误

每一个任务都有与任务上下文相关的出错代码。一些 MQX 函数检测到错误时会及时更新任务出错代码。可以通过如下函数获取任务错误代码：_task_get_error()。

任务能通过调用_task_set_error()函数设置其任务错误代码为一个值或者为 MQX_OK。然而，只有当前任务的错误代码是 MQX_OK 时，MQX 才会更新任务的错误代码。

（7）重启任务

应用程序能够通过调用_task_restart()函数重新启动任务，该函数从该任务函数的起始位置重启任务，保留了同样的任务描述符、任务 ID 和任务栈。

（8）终止任务

任务能够终止自身或者其它任何已知任务 ID 的任务。当一个任务被终止时，其子任务并不终止。当一个任务终止时，MQX 释放该任务的所 MQX 资源。应用程序能够通过调用_task_destroy()或_task_abort()函数立刻终止一个任务。

2.3 时间管理

时间管理在内核中占有非常重要的地位。内核必须管理系统的运行时间以及当前的日期和时间。

2.3.1 MQX时间相关概念

MQX 中的时间可以用秒、毫秒、时钟滴哒（tick）来表示。

分辨率定义了 MQX 更新时间的频率以及时钟嘀答发生的频率。分辨率一般为每秒 200 个时钟嘀哒或 5 毫秒。MQX 启动时通过装载周期定时器 ISR 来设置硬件的时间分辨率。

MQX 提供两类时间：消逝时间和绝对时间。消逝时间为 MQX 在处理器上开始运行所经历的时间。当消息使用基于共同时间参考的时间戳时，为了使不同处理器的任务可以交换消息，定时器组件提供绝对时间。

2.3.2 MQX时间相关函数

_time_get_resolution()：获取时间的分辨率。

_time_set()：设置、修改系统的绝对时间。

_time_get()：获取系统的绝对时间。

_time_delay()：延迟时间，任务可以调用该函数将自己挂起。

2.3.3 MQX中的定时器

MQX中的定时器实现如下功能：

（1）在指定的时间产生通知功能。当MQX创建定时器组件时，它启动定时器任务来维护定时器和应用程序定义的通知信息。当定时器溢出时，定时器任务可调用适当的通知函数。

（2）在预定的时间周期结束后进行通信。

MQX中的定时器包含两种类型：一次性定时器和周期性定时器。一次性定时器：只能超时一次。周期性定时器：可以根据指定的时间间隔重复超时，当定时器超时后，MQX将定时器复位。

2.4 中断管理

中断本质上是一种特殊的电信号，由硬件设备发向处理器[34]；同时，中断也是一种异步机制，ISR不需要内核的调度就可以执行。尽管RTOS具有多任务管理的功能，但中断作为一种不占用CPU时间而通过硬件自发产生的、实时性更强的事件触发机制，在嵌入式系统应用中扮演着十分重要的角色。系统通过中断机制了解外部世界，并对外部事件作出响应，RTOS对中断的响应性能也是其重要的性能指标之一[35]。

RTOS系统响应中断的过程是：系统接收到中断请求后，如果这时CPU处于中断允许状态，系统就会中止正在运行的当前任务，保存部分或全部寄存器的值，而按照中断向量的指向转而去运行ISR；当ISR运行结束后，系统将会根据情况返回到被终止的任务继续执行，或者转向另一个具有更高优先级的就绪任务。关中断和开中断可以让微处理器在处理本次中断时不响应其他外部中断[36]。

2.4.1 中断管理的功能

在嵌入式实时应用中，要求ISR和其它任务之间协同工作，以快速、合理地响应外部事件，并完成后续的处理工作。内核对中断提供的管理功能使得这种协同机制能够得以实现。一般来说，内核的中断管理具有如下功能：

（1）安装指定中断的服务程序，使得一个硬件中断和一个中断服务程序相关联。当中断发生时，系统保存和恢复中断现场，并且转到相应的 ISR 中执行。ISR 负责处理中断，清除中断标记以使中断能够再次发生，以及进行设备操作。ISR 运行时可以使用当前被中断任务的堆栈，也可以使用专门的中断堆栈。实时内核一般提供专门的堆栈来处理中断，防止可能的任务堆栈溢出。

（2）将内核内部支持的各级中断映射到目标处理器的各级中断上。

（3）为某些设备提供缺省的中断处理程序。

（4）在 ISR 中可以使用内核提供的 API，以便完成与任务之间的通信。在 ISR 中使用 API 具有一定的条件限制，内核必须提供某种机制来避免在 ISR 中由于这些调用可能造成的意想不到的后果。

（5）提供使能和屏蔽中断的系统调用，使应用能够根据需要在运行时关闭和打开中断。

2.4.2 MQX的中断行为

MQX 的 ISR 标识一个事件的发生。MQX 中 ISR 最常用的行为是[37]：

（1）传递信号量。

（2）发送消息。

（3）设置事件。

（4）清除错误条件。

2.4.3 MQX中断管理

MQX 中断管理包含用户 ISR 和内核 ISR。

ISR 不是任务，而是能快速响应硬件中断和异常的小而且快速的程序。当 MQX 调用 ISR 时，它将传递一个由应用程序定义的参数，通过应用程序安装 ISR。

MQX 的内核 ISR 用汇编语言编写，用于管理所有的用户中断，它会在其它任何 ISR 运行之前运行，并完成如下任务：

（1）保护当前任务的现场；

（2）切换到中断堆栈；

（3）调用合适的用户 ISR；

（4）当用户 ISR 返回后，恢复具有最高优先级的就绪任务的现场。

当 MQX 启动后，会为所有可能的中断装载默认的 ISR。当用户 ISR 返回到内核

ISR后，如果此时用户ISR有一个处于就绪状态的、优先级更高的中断任务，内核ISR就会进行任务调度。也就是说，前一个激活任务的现场将被存储起来，而更高优先级的任务则成为当前的激活任务。图2-4为MQX中断处理框图。

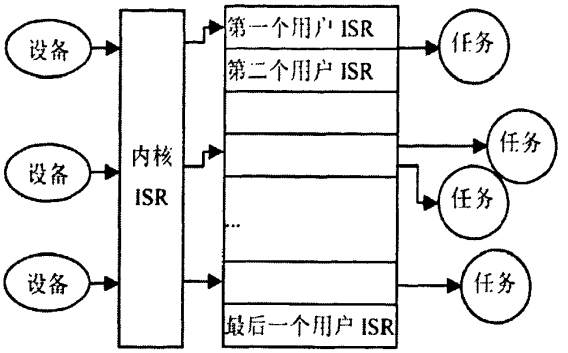


图2-4 MQX中断处理框图

MQX的中断处理过程如下：

（1）MQX启动后，首先将初始化ISR表，此表中包含了每个中断的入口（包含指向ISR的指针），每个入口的ISR都调用_int_default_isr()函数来产生默认的ISR。

（2）装载应用程序定义的ISR。当中断产生时，MQX将调用_int_install_isr()函数，使用应用程序定义的、面向特定中断的ISR代替默认ISR。_int_install_isr()必须包含：（1）中断向量号；（2）指向中断的指针；（3）当中断产生时，指向作为第一个参数传递给ISR的数据。在中断初始化设备之前，应用程序必须完成这个替换。

（3）装载内核ISR。通过调用_int_install_kernel_isr()函数，内核ISR将被用户ISR替换。安装新的内核ISR过程中，需要注意以下几点：（1）进入时保存所有的寄存器，退出时恢复它们。（2）不能调用任何MQX函数。（3）通过应用程序执行机制传递信息到其他任务。

2.5 存储管理

在RTOS中可以用malloc()和free()两个函数动态地分配内存和释放内存。但是，在RTOS中多次调用这两个函数会造成很多内存碎片。本来嵌入式系统的内存资源非常有限，由于这些碎片的大量存在，使得程序到后来连非常小的内存也分配不到。

因此，在MQX中，操作系统内存分为可变大小内存块和固定大小内存块。内存块可以是私有内存块（属于分配它的任务所有）或者系统内存块（不属于任何任务）。当任务结束时，MQX返回该任务的私有内存块给内存。MQX采用这样的内存管理算法，既可以满足任务内存需求，又解决了内存碎片问题。

为了更好的节省内存资源，以供代码和数据量较小的的任务使用，MQX 还利用可变大小块管理轻量级内存。

使用区块组件，可以管理固定大小内存块的分区，区块的大小是当任务创建分区的时候指定的。在默认的内存池中有可增长的动态区块，而在默认内存池之外又有不可增长的静态区块。

MQX 函数允许初始化、启用、禁用 MMU，或者为其增加一个存储区域，并通过 MMU 页表管理 MMU。虚拟存储器组件允许应用程序控制 MMU 页表，以便于管理虚拟内存，使它映射到相应的物理地址。应用程序能够使用虚拟内存组件为一个任务生成虚拟现场，它为该任务提供私有的内存空间，而且仅当该任务活动时才可见。图 2-5 为虚拟地址、MMU 页表、物理页以及物理地址之间的关系图。

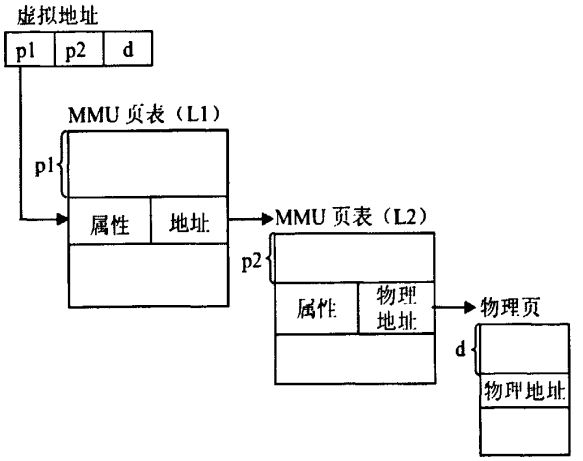


图2-5 MQX虚拟存储框图

2.6 任务同步与通信

任务同步是使并发执行的诸进程之间能有效地共享资源和相互合作[30]。任务通信，是指任务之间的信息交换，其所交换的信息量，少者是一个状态或数值，多者则是成千上万个字节。在 RTOS 中，任务之间一般是依靠信号量机制、消息队列、事件等进行同步和通信的。

任务间通信机制是多任务间相互同步和通信，以协调各自活动的主要手段。MQX 提供的任务间同步手段有事件、信号量和互斥。同时为了节省存储空间，提高运行速度，MQX 还提供了轻量级事件、轻量级信号量以实现任务之间的同步。MQX 提供的任务通信机制有事件、消息、信号量和任务队列。

2.6.1 事件

事件能用来同步多个任务，或实现任务与ISR之间的同步，它通过改变位状态的形式来传送信息。当其它任务或ISR出现了一个预先定义的事件就会产生一个事件标志。事件机制提供了复杂同步的功能，它具有的以下五个特点：（1）不提供数据传输功能；（2）事件之间是相互独立的；（3）任务可以同时等待多个事件；（4）对事件的等待方式有“与”、“或”两种逻辑操作；（5）事件没有队列，即在事件标志未被接收处理之前，多次向任务发送同一事件，只相当于发送一次。

MQX提供了事件和轻量级事件。事件组件包括事件组，它是事件位的集合。任务可以等待事件组中的事件位。如果事件位没有被置位，则任务将阻塞。任何其它任务或者ISR均可以置位事件位。当事件位被置位，MQX将从等待的任务中选择满足条件的任务进入就绪队列。如果事件组含有自动清除的事件位，只要该事件位被置位，MQX将立即清除事件位，并将该任务置入就绪队列。

MQX对事件的操作包含：生成事件组件、创建事件组、打开与事件组的连接、等待事件位、设置事件位、清除事件位、关闭与事件组的连接和撤销事件组等。

2.6.2 信号量

信号量是用于多任务内核中任务之间、任务与ISR之间的同步，以及任务间的互斥。其具有以下三方面功能：（1）控制共享资源的使用权，使其满足互斥条件；（2）标志某事件的发生；（3）同步任务之间、任务与中断之间的行为[38]。可以将信号量看作一把钥匙，与之相关的任务要运行下去，必须先得到这把钥匙。如果信号量已被别的用户占用，该任务只得被挂起，直到信号量被当前使用者释放[39]。

内核一般可以提供3种类型的信号量用于解决不同的问题，这三种类型的信号量为：解决互斥问题的互斥信号量；解决同步问题的二值信号量；解决资源计数问题的计数型信号量。内核到底将信号量给当前等待任务中的哪个任务，要看内核是如何调度的[9]。

MQX提供了信号量和轻量级信号量。在MQX中，任务要创建信号量，同时要指定：信号量的初始值，优先级队列，优先级继承等。

MQX对信号量的处理过程如下：任务等待信号量，如果信号量为0，则MQX阻塞该任务；否则，MQX降低信号量，并给该任务一信号量，该任务继续运行。如果带有该信号量的任务结束运行时，则它会传递信号量并保持就绪状态。如果任务正

在等待信号量，MQX 将该任务置入就绪队列，同时增加信号量。

当任务使用信号量获取共享资源时，会发生优先级倒置的现象。MQX 中的信号量使用优先级继承和优先级保护防止优先级倒置。优先级继承是指拥有互斥量的任务被提升到与下一个等待该互斥的最高优先级任务相同的优先级[40]。优先级保护是指获得互斥量的任务将其优先级提升到一个事先规定好的值[41]。

MQX 对信号量的操作包含：创建信号量组件、生成信号量、打开与信号量的连接、等待信号量、传递信号量、关闭与信号量的连接和撤销信号量等。

2.6.3 互斥

任务通过使用互斥保证某一时刻仅有一个任务访问共享数据。为了访问共享数据，任务可对互斥量加锁，如果该互斥量已经被加锁则等待。当任务完成对共享数据的访问，则解锁该互斥量。互斥通过优先级继承和优先级保护防止优先级倒置。

MQX 对互斥的操作包含：创建互斥组件、创建与初始化互斥、锁定互斥、解锁互斥和撤销互斥等。

2.6.4 消息

消息队列机制提供任务之间、任务与 ISR 之间的通信功能。消息队列一般有两种实现方案：（1）队列中存放消息指针，这样就可以发送多个字节的消息；（2）队列中存放字节变量。

一个消息就是一个可变长度的缓冲，在这个缓冲中存放信息以完成通信[42]。消息的长度及其存放的内容是由用户定义的，可以是：数据、指针、空（为空时消息的发送和接收是用来进行同步的，不进行数据传输）。

MQX 的任务之间可以通过交换消息实现相互通信。任务发送消息到消息队列，并从消息队列接收消息。

MQX 的任务从消息池分配消息。MQX 的消息池分为两种：系统消息池和私有消息池。系统消息池不是任何任务的资源，任何任务都可以从中分配消息。任何任务可以根据消息池 ID 从私有消息池分配消息。

MQX 的任务使用消息队列交换消息。消息队列可以是私有的也可以是系统的。当任务根据特定的值打开消息队列时，MQX 返回一个唯一的应用程序消息队列 ID，之后任务将据此访问消息队列。任务可以发送消息到任何私有消息队列，但仅有打开私有消息队列的任务才可以接收消息。系统消息队列不属于任何一个任务，而且任务

在接收消息时不会被阻塞，因此，ISR能够使用系统消息队列。

MQX对消息的操作包含：创建消息组件、创建消息池、分配与释放消息、发送消息、使用消息队列接收消息等。

2.7 本章小结

本章主要从以下几个方面对MQX的内核及MQX操作系统的功能进行剖析和研究：

（1）深入分析研究了微内核结构的组成和优缺点。从核心组件和可选组件两方面，详细分析了MQX的微内核结构组成和各组件的功能。

（2）讲解了MQX的任务模板列表、任务状态转换、任务切换过程和任务管理的相关函数，从而说明了MQX的任务管理功能的实现。

（3）详细论述了MQX的内核的时间管理功能、存储管理功能、中断管理功能。

（4）从信号量、事件、互斥、消息和任务队列等方面讲解了MQX的任务同步和通信机制的实现。

本章从MQX微内核组成、任务管理、时间管理、中断管理、存储管理和任务同步与通信等方面剖析了MQX的内核管理功能，对充分了解和使用MQX的内核具有指导意义，从而为后续章节的MQX实现机制做准备。

第三章 MQX 实现机制研究

为了快速学习并使用某个RTOS，就需要掌握其内部实现机制，这样在学习和使用过程中，RTOS就会发挥意想不到的作用。本章首先介绍了MQX的启动执行过程，详细阐述了系统上电启动后对微控制器及其内部功能模块进行初始化的过程，分析了Bootloader加载和启动执行过程，直到系统从自启动任务开始运行。其次，对MQX的内核调度策略和任务调度类型进行讲解、分析，为了改善MQX调度策略的缺点，在MQX中引入多级反馈队列调度策略。最后，分析研究了MQX的设备驱动程序的设计与实现。

3.1 MQX 的启动执行过程

MQX 启动执行时，首先完成 Bootloader 的下载和执行，从而完成硬件初始化。然后跳转到 main()函数，调用_mqx()函数。最后读取任务模板列表，执行自启动任务，使 MQX 运行，如图 3-1 所示。MQX 系统启动过程一共需要包含三部分的文件：CodeWarrior 库文件，板级支持包 BSP 文件，MQX 系统相关文件。

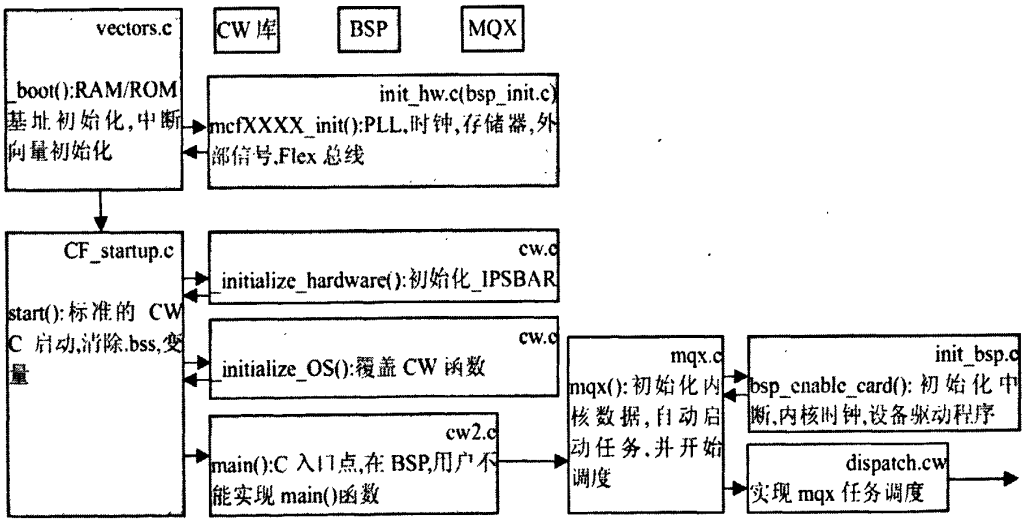


图3-1 MQX启动执行过程

下面对上图中的文件进行详细说明：

（1）ColdWarrior 相关文件

CF_startup.c：ColdWarrior 启动代码文件，包含_start()函数。启动代码将设置堆栈指针为_SP_INIT，设置数据相对基址地址，初始化.bss/.sbss 部分为零，调用静态 C

语言初始化函数（主要是 initialize_hardware()和 initialize_OS()），调用 main()函数。

（2）BSP 相关文件[37]

vectors.c：中断向量表文件，主要包含_boot()函数和中断向量表。

init_hw.c/bsp_init.c：初始化硬件文件，与硬件电路板相关。该文件包含 Flash 引导代码来初始化芯片和 SDRAM 存储空间，禁用看门狗定时器和初始化 PLL、Flex 总线。

cw.c：该文件包含对 Metrowerks 编译器的运行时间的支持，CodeWarrior 相关初始化，以及 CW 启动函数重载。

cw2.c：该文件包含 MQX C 入口代码，main()函数。

init_bsp.c：该文件包含初始化板所需的源函数，如：BSP 定时器，看门狗和 I/O 设备驱动程序初始化。

（3）MQX 相关文件

mqx.c：该文件包含了 MQX 主要源函数_mqx()。

dispatch.cw：该汇编文件包含任务调度函数。

3.1.1 Bootloader的分析

Bootloader 是在操作系统内核运行之前运行的一段小程序。它可以初始化硬件设备、建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态。然后加载操作系统映像到内存，跳转到操作系统代码执行。

Bootloader 用于连接硬件开发板和 MQX 系统，它是依赖于硬件实现的，不同的体系结构需要不同的 Bootloader，同时，它还依赖于具体的嵌入式板级设备的配置。也就是说，对于两块不同的嵌入式板而言，即使它们基于相同的 CPU 构建，运行在其中一块电路板上的 Bootloader，未必能够运行在另一块电路开发板上[43]。

3.1.2 Bootloader执行过程

系统在加电过后，芯片内的硬件机制会产生加电复位中断，这时系统到异常向量表中查找复位向量地址，并转向这个地址继续执行。异常向量表在 vectors.c 文件中，在该表开始的前两行分别是堆栈初始化指针和复位中断向量地址：

```
(vector_entry)__BOOT_STACK_ADDRESS,    //启动栈地址
__boot,                                //(0x004)初始化 PC
```

__BOOT_STACK_ADDRESS 在 intflash.lcf 链接文件中定义，它给出了系统在复

位过程中需要的启动栈地址，__boot 是汇编函数，描述了复位后程序执行过程。__boot 初始化指令计数器（Program Counter，PC），然后程序将从函数__boot 所代表的地址处开始执行。

接下来，执行_boot()函数，最终跳转到 start()函数。_boot()函数的主要作用是设置存储映射的基址参数，以安排 SRAM、Flash 和 IPSBAR 在 4G 的虚拟寻址空间中的位置。具体包括以下过程：定位堆栈指针；备份芯片信息；初始化 IPSBAR；初始化 FLASHBAR，定位内部 Flash，并使其生效；初始化 RAMBAR1，定位 SRAM 并使其生效；系统模块初始化并跳转到 start()函数。汇编指令如下：

```
jsr    mcf5222_init     //初始化硬件板
jmp    _start           //转到操作系统启动函数
```

程序在开始执行 jsr mcf5222_init 之前，会把当前指令的下条指令地址压入堆栈，在返回时，该地址将从堆栈中取出并放到 PC 中，那么就可以执行下条指令 jmp _start 了，这是一条跳转指令，跳转到开发环境中的 CF_startup.c 文件中执行 start()函数。该函数设置堆栈指针、清除存储空间、清除 CPU 寄存器和其他 ColdFire 常规事务，内部调用 initialize_hardware()和 initialize_OS()函数，并最终跳转到 main()函数。

3.1.3 MQX执行过程

Bootloader 下载执行完毕，从 main()函数开始执行，调用_mqx()运行，将控制权传递给 MQX 内核，MQX 系统开始运行。

main()函数在 cw2.c 中，其函数内容为 MQX 执行时，MQX 初始化结构定义的任务模板列表地址指向任务模板列表。通过任务模板列表（由任务模板结构 TASK_TEMPLATE_STRUCT 组成的一组列表）定义了一组初始化模板（必须包含一个自启动的任务，即任务属性为 MQX_AUTO_START_TASK），基于此，可以在处理器上生成、调用其他任务。初始化时，MQX 为每个任务生成一个实例。当应用程序运行时，它能按任务模板生成其它任务。main()函数代码如下：

```
int main(void)
{
    extern MQX_INITIALIZATION_STRUCT MQX_init_struct;
    _mqx(&MQX_init_struct );          //开始执行 MQX
    return 0;
}
```

mqx()函数完成以下功能：初始化内核数据，初始化内存管理器，分配和初始化中断堆栈，开启定时器，设置默认的时间片值，分配和初始化系统任务描述符 TD，

分配和初始化准备运行队列，创建内核信号量，初始化 BSP，创建空闲任务，创建自启动任务，调用调度程序。mqx()执行完毕，完成操作系统的启动工作，并创建被应用程序称为自启动的任务。初始化结构定义了应用程序和目标硬件的参数，通过调用_mqx()在初始化时传递给 MQX。其结构定义如下[22]：

```
typedef struct mqx_initialization_struct
{
    _mqx_uint PROCESSOR_NUMBER;                          //处理器号
    pointer START_OF_KERNEL_MEMORY;                      //内核使用的 RAM 首地址
    pointer END_OF_KERNEL_MEMORY;                        //内核使用的 RAM 末地址
    _mqx_uint INTERRUPT_STACK_SIZE;                      //中断栈大小
    TASK_TEMPLATE_STRUCT_PTR TASK_TEMPLATE_LIST;         //任务模板列表地址
    _mqx_uint MQX_HARDWARE_INTERRUPT_LEVEL_MAX;          //最大硬件中断优先级
    _mqx_uint MAX_MSGPOOLS;                              //最大消息缓冲区
    _mqx_uint MAX_MSGQS;                                 //最大消息队列数
    char _PTR_ IO_CHANNEL;                               //标识那个设备被用为默认 I/O
    char _PTR_ IO_OPEN_MODE;                             //默认 I/O 的打开标志
    _mqx_uint RESERVED[2];                               //保留
}MQX_INITIALIZATION_STRUCT, _PTR_ MQX_INITIALIZATION_STRUCT_PTR;
```

3.2 MQX 的调度策略

任务调度主要是协调任务对计算机系统资源的争夺使用。对系统资源非常匮乏的嵌入式系统来说，任务调度策略尤为重要，它直接影响到系统的实时性能。MQX 系统提供了三种任务调度策略：基于优先级抢占机制的先进先出（First In First Out，FIFO）调度（系统默认），轮询（Round Robin，RR）调度，显式调度（使用任务队列）。可以分别把每个任务和处理器设置成 FIFO 或者 RR 调度方式。对一个任务而言，可以使用 MQX 系统默认调度策略对任务进行调度；同时，也可以通过在任务模板列表中指定任务属性来改变任务调度策略。

3.2.1 内核调度策略

MQX 任务调度遵从 POSIX.4 标准（实时扩展）并且支持如下策略：

1. FIFO调度

MQX 的 FIFO 调度是基于优先级抢占机制的调度方式。FIFO 是默认的调度策略，定义在<MQX_install>\mqx\source\kernel\mqxiinit.h 中，代码如下：

```
#if MQX_HAS_TIME_SLICE
kernel_data->SCHED_POLICY = MQX_SCHED_FIFO;//设置创建任务的默认的调度策略
#endif
```

使用 FIFO 调度策略，接下来运行的任务是拥有最高优先级且等待时间最长的那个任务，如图 3-2 所示。当发生以下任意一种情况时，活动任务停止运行：

（1）由于调用了 MQX 阻塞功能函数，活动任务主动放弃处理器。

（2）产生了一个比活动任务优先级高的中断。

（3）更高优先级的任务已经处于就绪状态。

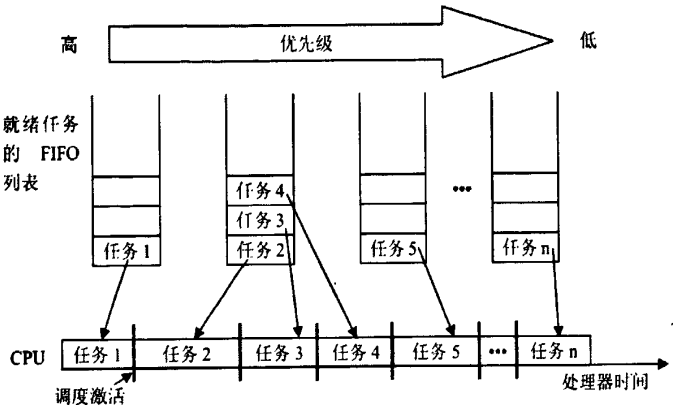


图3-2 基于优先级的FIFO调度

2. RR调度

RR 调度即时间片轮询调度方式，其将调度等待时间最长且未消耗自身时间片的任务。RR 调度的特点在于每个轮询任务有最长时间限制（时间片），在此时间内该任务可以被激活。

只有在任务模板结构中已设定 MQX_TIME_SLICE_TASK 属性的任务才使用 RR 调度。也就是说，首先，要在<MQX_install>\lib\<board>.cw\mqx\mqx_cnfg.h 中定义：

```
#define MQX_HAS_TIME_SLICE 1
```

其次，在任务模板中设置任务的属性为 MQX_TIME_SLICE_TASK。

任务的时间片是由任务模板结构中的 DEFAULT_TIME_SLICE 的值来确定的。但如果该值为 0，任务的时间片即为默认的处理器时间片。开始时，处理器默认的时间片是定时器所设定的时间间隔的十倍。比方说大部分的 BSP 定时器时间间隔是 5ms，那么默认时间片通常就是 50ms。可通过调用 _sched_get_rr_interval() 和 _sched_get_rr_interval_ticks()（系统时钟）获取时间片；调用_sched_set_rr_interval() 和_sched_set_rr_interval_ticks()函数设置时间片。当一个活动的轮询任务的时间片用完时，MQX 将会保存该任务的现场参数，执行切换操作，并检查就绪队列选择优先级最高的任务进入激活状态。MQX 将已过期的任务放到任务就绪队列的末尾，这样就可以开始控制就绪队列中的下一个任务。如果在就绪队列中没有其他的任务，则到

期的任务将继续运行。

在RR调度策略中，相同优先级的任务将以一种时间均等的方式共享处理器时间。如果多个具有相同优先级的任务进入就绪状态，则就绪队列中的最前面的任务将被激活，即每个就绪队列都是按照FIFO顺序排列的。如图3-3所示。

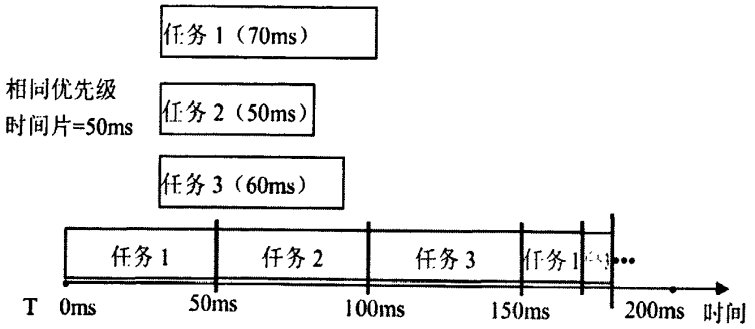


图3-3 具有相同优先级的轮询调度

3. 显式调度

使用任务队列显式地调度任务或创建相对复杂的同步机制。创建一个特殊的队列用于存放任务，直到显式地重新调度策略（包含先进先出调度和基于优先级调度），才把该任务放回就绪队列。

```
#define MQX_TASK_QUEUE_FIFO              (0x00)
#define MQX_TASK_QUEUE_BY_PRIORITY       (0x08)
```

显示调度主要使用如下函数：

（1）_taskq_create()：创建队列。

（2）_taskq_suspend()：通过将任务放到任务队列来挂起处于激活态的任务。

（3）_taskq_resume()：对队列中的第一个任务或所有任务重新启动（即放入就绪队列中）。

3.2.2 任务调度类型

在MQX系统中，任务调度类型是由任务属性决定的，定义在<MQX_install>\lib\<board>.cw\mqx\mqx.h中。

（1）任务属性为MQX_AUTO_START_TASK的任务是自启动任务，该任务在系统初始化时自动创建。

```
#define MQX_AUTO_START_TASK              (0x01)
```

（2）任务属性为MQX_FLOATING_POINT_TASK的任务是浮点任务，该任务

在上下文切换中将保存浮点协处理器寄存器。如果浮点寄存器从标准的寄存器中分离出来，则在任务切换过程中，它们的上下文被单独管理。只有当一个新的浮点任务被调度运行时，该寄存器才被保存或重新恢复。

```
#define MQX_FLOATING_POINT_TASK          (0x02)
```

（3）任务属性为 MQX_TIME_SLICE_TASK 的任务是时间片任务，该任务使用 RR 调度。

```
#define MQX_TIME_SLICE_TASK              (0x04)
```

（4）任务属性为 MQX_DSP_TASK 的任务是 DSP 任务，该任务在上下文切换中将保存 DSP 协处理器寄存器。如果 DSP 寄存器从标准的寄存器中分离出来，则在任务切换过程中，它们的上下文被单独管理。只有当一个新的 DSP 任务被调度运行时，该寄存器才被保存或重新恢复。

```
#define MQX_DSP_TASK                     (0x08)
```

一个任务的任务属性，可以指定是以上四种任务属性的任意组合。

3.3 任务调度策略的改进

对于嵌入式实时操作系统，除了要满足特定应用功能需求外，更重要的是要满足应用提出的实时性要求，而组成一个应用的众多实时任务对于实时性的要求是各不相同的，此外实时任务之间可能还会有一些复杂的关联和同步关系，这就为系统实时性的保证带来了很大的困难[44]。

实时调度问题就是在多任务并发的系统中，对并发运行的任务集合在系统处理机上的执行进行合理的安排，从而确保各个实时任务的时间约束都能得到满足。调度的实质是如何对系统的各个任务合理地分配运行所需的资源。调度算法将决定系统如何进行资源分配，它是一种服务于系统目标的策略[45]。

3.3.1 任务调度改进原因

MQX 包含三种任务调度策略：FIFO 调度，RR 调度和显示调度。

MQX 基于抢占机制的 FIFO 调度策略是从具有最高优先级的就绪队列中，选择最先进入该队列的进程，为之分配处理机，使之运行。该进程一直运行到完成或发生某事件而阻塞后，才放弃处理机。该调度策略的优点是：（1）算法简单、容易实现、系统开销小；（2）可预测性强。但该调度策略的缺点是：（1）比较有利于长作业，而不利于短作业；（2）有利于 CPU 繁忙的作业，而不利于 I/O 繁忙的作业。

MQX的RR调度策略是将系统中所有的就绪进程按照先来先服务的原则，排成一个队列。每次调度时将CPU分派给队首进程，让其执行一个时间片（其长度由用户设置）。当一个时间片结束时，发生时钟中断。调度程序据此暂停当前进程的执行，将其送到就绪队列的末尾，并通过上下文切换执行当前的队首进程。在执行过程中，如果发生阻塞，即使当前时间片未用完，执行进程也会让出CPU。该调度策略是一种公平的调度算法，就绪队列中的所有任务都能被公平的执行。但使用该调度策略必须要考虑以下问题：时间片长度的设置。时间片过长，该调度策略退化为FIFO，任务可能在一个时间片内都执行完，使得对短任务的交互响应变差；时间片过短，用户的一次请求需要多个时间片才能处理完，任务切换次数增加，系统中会有大量时间浪费到切换任务上，系统开销大，降低了CPU效率。

因此，使用RR调度策略设置时间片长度时需要考虑以下问题：

（1）系统对响应时间的要求。T（响应时间）=N（进程数目）*q（时间片）。

（2）就绪队列中进程的数目。就绪进程的数目越多，时间片长度应越小。

（3）系统的处理能力。应当使用户进程尽量在一个时间片内能处理完。

基于上述对MQX的FIFO调度策略和RR调度策略的描述，为了使高优先级的任务得到响应又能使短任务迅速完成，把FIFO调度策略与RR调度策略结合起来，设计并实现了多级反馈队列调度算法。

3.3.2 多级反馈队列调度

多级反馈队列根据任务执行情况的反馈信息而对任务队列进行组织并调度任务。在MQX中，多级反馈队列的设计如下[30][46]：

（1）设置N个就绪队列（$Q_1,Q_2....Q_N$），各个队列对应于处理机一个优先级。一般来说，第一个队列的优先级最高，第二个队列次之，其余各队列的优先权逐渐降低，即 $Priority(Q_1)>Priority(Q_2)>...>Priority(Q_N)$。对于各个就绪队列中的任务执行时间片的大小也各不相同，即各个队列的时间片是随着优先级的增加而减少的。在优先权愈高的队列中，为每个任务所规定执行的时间片就愈短。

（2）在每个就绪队列中，遵循时间片轮转法。当一个新任务进入内存后，首先将它放入 Q_1 的末尾，按FIFO原则排队等待调度。当轮到该任务执行时，如它能在该时间片内完成，便可撤离系统；如果它在一个时间片结束时尚未完成，调度程序便将该任务转入 Q_2 的末尾，再同样地按FIFO原则等待调度执行；如果它在 Q_2 中运行一个时间片后仍未完成，再依次将它放入 Q_3，.....，如此下去，

当一个长任务从 Q_1 依次降到 Q_n 后，在 Q_n 中便采取按时间片轮转的方式运行。

（3）仅当 Q_1 空闲时，调度程序才调度 Q_2 中的任务运行；仅当 Q_1~Q_{i-1} 均空闲时，才会调度 Q_i 中的任务运行。如果处理机正执行 Q_i 中的某任务时，又有新任务进入优先权较高的队列（Q_1~Q_{i-1} 中的任何一个队列），则此时新任务将抢占正在运行任务的处理机，即由调度程序把正在运行的任务放回到 Q_i 的末尾，然后把处理机分配给新到的高优先权任务。图 3-4 为多级反馈队列调度框图。

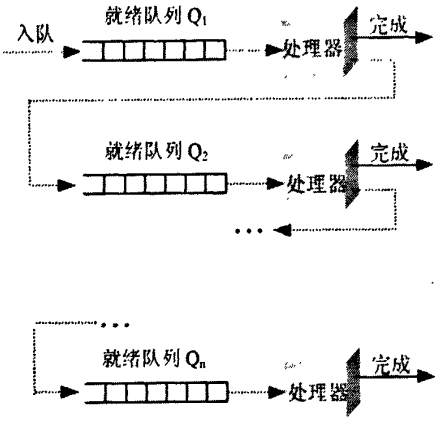


图3-4 多级反馈队列调度框图

假设系统中有 3 个反馈队列 Q_1, Q_2, Q_3，时间片分别为 2，4，8，且 Priority(Q_1)> Priority(Q_2) > Priority(Q_3)。现在有 3 个作业 J_1，J_2，J_3，分别在时间 0，1，3 时刻到达。而它们所需要的 CPU 时间分别是 3，2，1 个时间片。多级反馈队列的调度过程如图 3-5 所示。

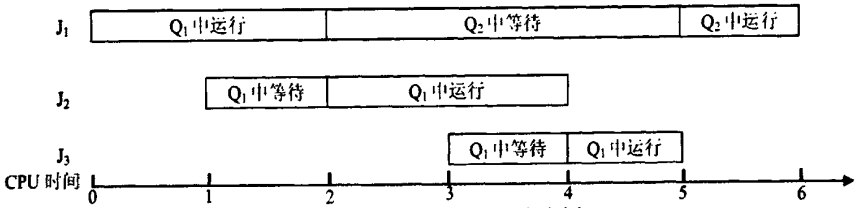


图3-5 多级反馈队列调度实例

3.3.3 任务调度改进实现

在 MQX 任务调度中每个任务都有一个任务描述符（Task Descriptor，TD）结构，包含了任务状态的各种信息，是对任务控制的唯一有效的手段。设置任务当前的优先级的函数是_task_set_priority()。真正执行任务调度的函数是_sched_internal()，_sched_setscheduler()和_sched_set_policy()。要在MQX中实现多级反馈队列任务调度，必须修改的部分有 kernel 的部分代码、任务描述符结构、_sched_internal()，

_sched_setscheduler()，以及_task_set_priority()函数。需要修改的主要文件有：

<MQX_install>\mqx\source\psp\coldfire\中的：

dispatch.cw：调度程序；

sc_irdyq.c：初始化调度器，设置内核优先级调度队列。

<MQX_install>\mqx\source\kernel 中的：

sc_ipq.c：根据优先级把任务描述符插入任务描述符队列；

sc_obsel.c，sc_spol.c：设置任务调度策略；

sc_srr.c：设置任务时间片。

MQX 中多级反馈队列任务调度的具体实现要点如下：

1）在 kernel 中的初始化代码中必须创建确定数量的任务就绪队列，数量可以根据处理器的能力大小确定。本课题中暂定为 6（0~5）级，前 5 级队列为 FIFO 型，最后一个队列是时间片轮询型。MQX 提供 5ms 的调度粒度，因此，时间片基数可定为 5ms，各级的时间片大小按 5*2n 计算（n 为进程队列序号，0≤n≤5）。除此之外，还应创建一个等待队列。

2）在 TD_STRUCT 结构中，需加入的成员变量有任务当前所在的就绪队列号 Qnumber 和任务在当前队列中的等待时间 WaitTime。

3）对_sched_internal()函数按如下原则修改：

（1）先检查第 6 级队列，如果有进程的 WaitTime 大于等待的极限时间 maxWaitTime 时，修改相应变量的值，然后将该进程重新移到第 1 级队列的末尾以避免饥饿现象。

（2）检查第 1 级队列，如果其中有就绪任务，则按照队列中排列的顺序进行 FIFO 调度，直到此队列空为止。对规定时间片内未运行完的进程，通过_task_set_priority()使优先级降低，放入下一级队列。接着按此方法依次检查第 2 级到第 5 级队列。对第 6 级队列，按时间片轮询运行，未运行完的进程仍排入第 6 级队列的末尾。在每次调度后都要做上述（1）步骤。

（3）如果有运行任务因为 I/O、任务间通信等事件被阻塞后，将运行任务放入等待队列，当任务重新就绪时，修改相应变量值，然后将其放入高一级的队列的末尾。

（4）当一个比当前运行任务优先级高的就绪队列中有任务到来时，当前任务就被抢占，而运行新到来的任务。修改被抢占任务的剩余时间片值（其值为当前所在队列的时间片常量减去已运行的时间片数）等相关参数，将其放到原就绪队列的末尾。

3.3.4 测试和对比

修改内核后，在 ColdWarrior 7.2 中直接对内核进行编译、调试和下载。本课题测试的软件环境为 3.6.2 版本的 MQX 内核；硬件环境为 MCF52223。在任务模板列表中仿真生成 80 个不同的任务，分别在 RR 调度和多级反馈队列任务调度中调度任务。如图 3-6 为平均周转时间比较结果。图 3-7 为平均等待时间比较结果。

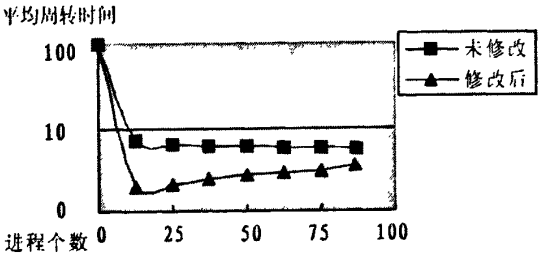


图3-6 平均周转时间对比

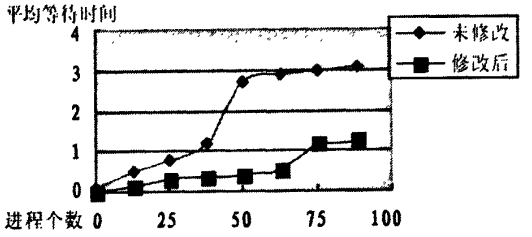


图3-7 平均等待时间对比

通过对比，修改后的 MQX 系统具有比原来更短的平均周转时间和平均响应时间，从而说明多级反馈队列调度策略确实要优于 MQX 原来的基时间片轮询的动态优先级调度策略。

3.4 MQX 设备驱动程序的设计

设备驱动程序从字面上可以理解为一类程序，这类程序的目的一般是驱动内部和外围的硬件正常工作，因此设备驱动程序都是针对特定的硬件来编写的。设备驱动程序主要完成如下功能：

（1）对设备进行初始化和释放设备；

（2）把数据从内核传送到硬件，或从硬件中读取数据；

（3）读取应用程序传送给设备文件的数据和回送应用程序请求的数据；

（4）检查处理设备出现的错误。

3.4.1 MQX下的驱动程序分析

在MQX系统中，设备分为三种类型：

（1）字符设备：以字节为单位逐个进行I/O操作；字符设备中的缓存可有可无；不支持随机访问。

（2）块设备：存取是通过buffer、cache来进行；可进行随机访问。

（3）网络设备。

MQX设备驱动程序是操作系统内核和硬件设备之间的接口，它是为特定的硬件而提供给用户程序的一组标准化接口，它隐藏了设备工作的细节[47]。

MQX系统中的设备驱动程序是运行在内核态的，是和内核连接在一起的程序。如果运行在用户态的应用程序想控制硬件设备，必须通过设备驱动程序来控制。MQX的设备驱动程序框架，如图3-8所示。I/O子系统采用统一的方式与I/O设备驱动程序进行通信，从而隐藏了特定的设备驱动程序函数。

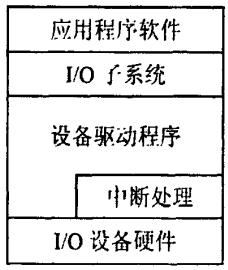


图3-8 MQX的驱动程序框架

编写驱动程序，要知道驱动程序的入口地点，在MQX中设备驱动程序的入口地点为FILE_STRUCT结构体，FILE_STRUCT结构体可以看成为函数指针的集合，在MQX中通过API把设备看成文件，对设备文件的操作也就对应成普通的文件操作，所以只需要实现驱动的入口就可以了，在FILE_STRUCT结构体中提供了许多的入口点，分别指向不同的设备驱动程序[48]，如图3-9所示。

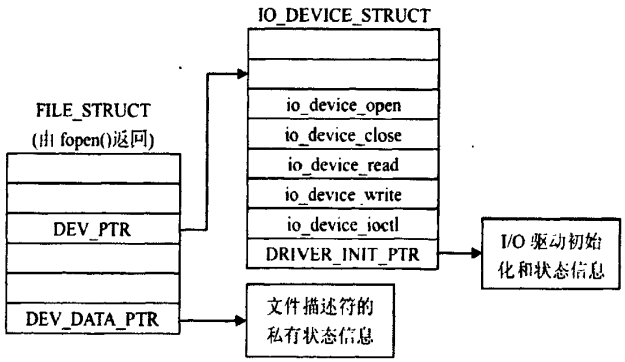


图3-9 文件句柄与I/O驱动程序的关系

MQX抽象了对硬件设备的处理，所有的硬件设备都可以作为普通文件来看待。可以使用与操作文件相同的系统调用来完成打开(open)、关闭(close)、读(read)、写(write)和I/O控制(ioctl)操作。对用户来说，设备文件与普通文件并没有区别。MQX访问设备就如同访问文件一样，设备驱动程序负责实现这些行为。I/O子系统的API函数采

用 POSIX 标准 I/O[37][49]。MQX 系统的 API 调度过程如图 3-10 所示。

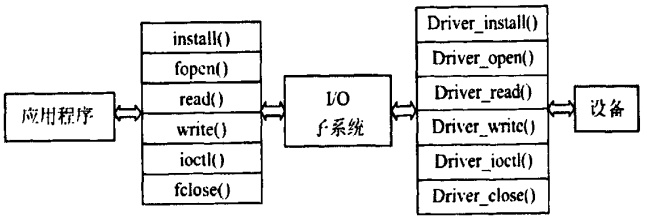


图3-10 MQX系统API的调度过程

MQX 对设备处理方式包含两种：轮询处理和中断处理。

（1）轮询处理通过内核定期对设备的状态进行查询，获取设备状态，对设备进行处理。其缺点是：消耗大量的内核资源；如果设备驱动被连接进内核，将会带来灾难性的后果。

（2）中断处理。设备驱动向内核注册其使用的中断，内核负责把硬件产生的中断传递给相应设备驱动，设备驱动对设备进行处理。

3.4.2 使用MQX设备驱动程序

在 MQX 中，设备驱动程序是可选部分。在 BSP 启动代码中，可以允许或禁止设备驱动程序。为了节约内存或 RAM，在<MQX_install>\config\<board>\user_config.h 文件中大部分的设备驱动程序默认都是禁止的。通过宏定义实现：

```
#define BSPCFG_ENABLE_TTYB        0
```

若要使用一个设备驱动程序，首先要将宏定义改为 1，然后重新编译 BSP 工程，最后通过 fopen 打开该设备驱动程序，才能在应用程序中使用该设备的操作函数。MQX 带有的主要设备驱动程序，如表 3-1 所示。

表 3-1 MQX 设备驱动程序

I/O 设备	设备模块	配置宏 (user_config.h)
ttya:	UART0（轮询）	BSPCFG_ENABLE_TTYA
ttyb:	UART1（轮询）	BSPCFG_ENABLE_TTYB
ttyc:	UART2（轮询）	BSPCFG_ENABLE_TTYC
ittya:	UART0（中断）	BSPCFG_ENABLE_ITTYA
ittyb:	UART1（中断）	BSPCFG_ENABLE_ITTYB
ittyc:	UART2（中断）	BSPCFG_ENABLE_ITTYC
i2c0:	I2C0（轮询）	BSPCFG_ENABLE_I2C0
i2c1:	I2C1（轮询）	BSPCFG_ENABLE_I2C1
spi0:	SPI0（轮询）	BSPCFG_ENABLE_SPI0
spi1:	SPI1（轮询）	BSPCFG_ENABLE_SPI1
gpio:	GPIO	BSPCFG_ENABLE_GPIODEV
adc:	ADC	BSPCFG_ENABLE_ADC
adcn:	其他 ADC 模块	BSPCFG_ENABLE_ADCn
flashx:	CFM	BSPCFG_ENABLE_FLASHX

3.4.3 设备驱动程序设计

I/O 驱动程序动态地安装到软件包中，为硬件提供一个直接的接口。MQX 设备驱动程序设计分为两个步骤：（1）编写相应的设备驱动程序文件，包含头文件和源文件；（2）把驱动程序文件添加到 BSP 工程，并重新编译 BSP 工程[50]。

1. 编写设备驱动程序文件

在 MQX 系统中，每个设备驱动程序包含 device.h 和 device.c 两个文件。

1）设备驱动程序头文件

device.h 实现设备驱动程序的安装，具体安装代码如下：

```
extern "C"
{
    extern _mqx_uint _io_device_install(char_ptr);
}
```

2）设备驱动程序实现文件

device.c 文件包含设备驱动程序私有函数的原型和实现。MQX 的所有设备驱动都必须包含以下函数：

（1）_io_device_install(char_ptr)：安装设备驱动程序，该函数调用_io_dev_install()向 MQX 注册设备，告诉内核有一个设备的驱动程序要安装到内核。参数标识符是一个输入的字符串，其标识 fopen 打开的设备。在入口函数编写完毕之后必须要进行对文件结构体的初始化工作，文件结构体是函数指针的集合，将结构体中函数指针初始化为对应入口函数即可，代码如下所示：

```
_mqx_uint _io_device_install(char_ptr   identifier)
{
    _mqx_uint result;
    result = _int_io_dev_install
    (
        identifier,
        _io_device_open,
        _io_device_close,
        _io_device_read,
        _io_device_write,
        _io_device_ioctl,
        NULL
    );
    return result;
}
```

（2）_io_device_open(FILE_PTR, char_ptr, char_ptr)：打开设备，驱动程序用来为

以后的操作完成初始化准备工作。当应用程序调用 fopen 时，通过 I/O 子系统映射为 _io_device_open，如前图 3-10 所示。它为后来的读、写、控制操作准备驱动程序。_io_device_open 函数完成的工作：初始化设备；检查设备相关错误；识别设备号等。

（3）_io_device_close(FILE_PTR)：关闭设备，用来关闭驱动程序和释放所有的资源。当应用程序调用 fclose 时，通过 I/O 子系统映射为_io_device_close，如前图 3-10 所示。_io_device_close 函数完成的工作：释放 xxx_open 分配的内存和资源。

（4）_io_device_read(FILE_PTR, char_ptr, _mqx_int)：从打开的设备中读取数据。当应用程序调用 read 时，通过 I/O 子系统映射为_io_device_read，如前图 3-10 所示。_io_device_read 函数的任务是将数据从设备复制到用户空间，完成内核空间和用户任务空间的数据传输。

（5）_io_device_write(FILE_PTR, char_ptr, _mqx_int)：向打开的设备写入数据。当应用程序调用 write 时，通过 I/O 子系统映射为_io_device_write，如前图 3-10 所示。_io_device_write 函数的任务是将数据从用户空间复制到设备，完成内核空间和用户任务空间的数据传输。

（6）_io_device_ioctl(FILE_PTR, _mqx_uint, pointer)：I/O 操作扩展，可根据设备情况来决定支持何种特殊的操作模式。_io_device_ioctl 是与具体设备相关的，它为驱动程序执行“命令”提供了一个与设备相关的入口点。当应用程序调用 ioctl 时，通过 I/O 子系统映射为_io_device_ioctl，如前图 3-10 所示。它允许应用程序访问被驱动硬件的特殊功能，如配置设备以及进入或退出操作模式。

3）保存设备驱动程序文件

device.h 和 device.c 编写完毕后，在<MQX_install>\mqx\source\io\目录中新建一个子目录并保存为 device，并把 device.h 和 device.c 文件复制到该目录下。

2. 添加设备驱动程序到BSP

MQX 系统包含两种工程：处理器支持包（Platform Support Package，PSP）工程和板级支持包（Board Support Package，BSP）工程。PSP 工程只包含平台相关（即处理器本身）的代码，不包含任何与电路控制板相关的代码。BSP 工程包含与电路控制板相关的代码。因此，新驱动程序要添加到 MQX 系统，必须重新编译 BSP 工程。在 Codewarrior 中打开 BSP 工程，其目录为：<MQX_install>\mqx\build\codewarrior\bsp_<board>.mcp。

（1）拖拽整个 device 目录到 Codewarrior 工程中的 IO Drivers 目录下。

（2）在弹出的窗口中单击“OK”，从而添加文件到所用的目录。

（3）拷贝驱动程序的头文件 device.h 到输出目录，即：<MQX_install>\lib\<board>.cw\mqx，编译、生成工程。

（4）应用程序需要使用设备驱动程序，只需要添加头文件，如：#include<device.h>。先调用_io_device_install 后，再使用 fopen，fclose，write，read 和 ioctl 函数。

3.5 本章小结

本章详细阐述了 MQX 的实现与使用，总体来说，本章讲述的内容有如下几点：

（1）MQX 的启动执行过程，分析了 MQX 从系统上电到启动执行的整个过程。系统上电启动后，Bootloader 的下载、执行过程；Bootloader 运行完毕后，MQX 跳转到 main()函数，在 main()内部调用 mqx()，从而创建任务模板列表实例，查找自动启动任务，直到操作系统开始运行。

（2）MQX 的任务调度。详细讲述了 MQX 的三种内核调度策略，阐述了如何通过改变任务的任务属性，对任务调度方法进行设置。同时分析了 MQX 的 FIFO 调度策略和 RR 调度策略的优缺点，为了弥补两种调度策略的缺点，提高 MQX 对任务的实时处理能力，在 MQX 中引入了多级反馈队列调度，并对修改前后任务调度的性能进行测试、比较，从比较结果看，MQX 的任务调度的平均周转时间和平均等待时间确实得到了提高。

（3）MQX 设备驱动程序的设计。首先，讲解了 MQX 的设备类型、设备驱动程序的作用。其次，研究了设备驱动程序的实现，即通过系统调用来实现到设备驱动程序的映射。最后，按步骤详细说明了 MQX 中设备驱动程序的设计实现过程。

本章从 RTOS 使用过程中的关键技术方面，对 MQX 进行了详细的分析和讲解，从而为后续章节的应用实例做准备。

第四章 MQX 的工程框架

基于前面的对 MQX 的功能介绍，本章将详细阐述 MQX 的工程框架。主要包含如下内容：（1）从 MQX 的开发环境开始，对开发环境的版本和安装过程进行讲解。（2）对 MQX 安装目录下的各目录作用和内容进行说明。（3）为了深入理解 MQX 工程组织，对 MQX 的工程组织和框架进行重新归类、设计。（4）为了提高软件的可重用性、可移植性，按照底层软件构件的思想对 MQX 的底层硬件驱动程序进行重新编写，并以串口控制小灯程序为例说明底层软件构件的编写过程。

4.1 MQX 的开发环境

嵌入式软件开发有别于桌面软件开发的一个显著的特点是它一般需要一个交叉编译和调试环境，即编辑和编译软件在通常的 PC 机上进行，而编译好的软件需要通过写入工具下载到目标机上执行。MQX 使用 CodeWarrior 作为编译和调试环境。

4.1.1 CodeWarrior环境简介

CodeWarrior 开发环境（简称 CW 环境）是飞思卡尔公司研发的面向 Freescale MCU 与 DSP 嵌入式应用开发的商业软件工具，其功能强大，是飞思卡尔向用户推荐的产品之一[51]。CodeWarrior 是一个多主机、多语言、多目标的 GUI 集成开发环境（Integrated Development Environment，IDE），在该环境下可编制并调试 MCU 的汇编语言、C 语言和 C++语言程序。

CodeWarrior 分为 3 个版本：特别版（Special Edition）、标准版和专业版。其中特别版是免费的，用于教学目的，对生成的代码量有一定限制，C 语言代码不得超过 12KB，对工程包含的文件数目也限制在 30 个以内。标准版和专业版没有这种限制。3 个版本的区别在于用户所获取的授权文件（license）不同，特别版的授权文件随安装软件附带，不需要特殊申请，标准版和专业版的授权文件需要付费。CodeWarrior 的版本号有：CodeWarrior for S08 V6.2、CodeWarrior for HC12 V4.6、CodeWarrior for ColdFire V6.3、CodeWarrior for ColdFire 7.1、CodeWarrior for Microprocessor 9.2、CodeWarrior for MCU V10 等。

CW 环境包括以下几个功能模块：编辑器、源码浏览器、搜索引擎、构造系统、调试器、工程管理器。编辑器、编译器、连接器和调试器对应开发过程的四个主要阶

段，其它模块用以支持代码浏览和构造控制，工程管理器控制整个过程。该集成环境是一个多线程应用，能在内存中保存状态信息、符号表和对象代码，从而提高操作速度；能跟踪源码变化，进行自动编译和连接[7]。CW 环境能够通过用户对模块设置的属性，自动生成代码，减少了很多底层繁琐冗余的工作，方便用户更容易的进行开发。

MQX 使用开发环境是 IAR，CodeWarrior for ColdFire V6.3 及其以上版本。

4.1.2 CW环境安装

CW 环境安装没有什么特别之处，在 Windows 操作系统上，只要按照安装向导单击鼠标就可以自动完成。

需要说明的是，安装完毕以后要上网注册以申请使用许可（license key）。无论是下载的软件还是申请到的免费光盘，安装后都要通过因特网注册，以申请使用许可。申请后会通过 E-MAIL 得到一个 License.dat 文件。对于免费的特别版本，安装好后用 License.dat 覆盖安装目录下的 License.dat。

打开带有 MQX 库的工程文件，进行编译、链接时，开发平台会自动绑定到相应的 MQX 库。

4.2 MQX 的安装框架

从飞思卡尔网站免费下载的 MQX 系统，安装后，在安装目录下会创建如下子目录，如图 4-1 所示：

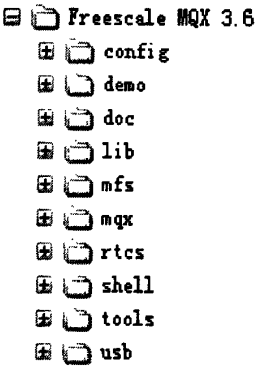


图4-1 MQX的安装目录

下面对各目录包含的文件进行说明：

（1）config：该目录下包含配置文件。该目录的一个子目录为 Common 目录，包括所有板子 BSP 共用的配置文件。其他的每个子目录都对应一个板子，包含针对每个 BSP 的个性化配置。

（2）demo：几个综合性的 demo 工程实例。

（3）doc：MQX 的英文文档，包括 MQX 用户手册、MFS 参考手册、RTCS 参考手册、USB 参考手册等。

（4）lib：包含每个板子 BSP 的最终输出结果、库文件、头文件等。

（5）mfs：MFS 文件系统源码和实例。该目录包含三个子目录：Build、Examples 和 Source。

（6）mqx：包含 MQX 操作系统代码、BSP 相关代码和驱动代码。该目录包含

三个子目录：Build、Examples 和 Source。

Build：包含不同编译器（CodeWarrior，IAR 等）的子目录，每个子目录下则是各个板子 BSP 和 PSP 的工程文件。此外，还包括一个 bat 子目录，里面是各个板子 BSP、PSP 的链接后批处理脚本。

Examples：样例程序，演示 MQX 主要功能，比如消息邮箱、中断服务等，可以用该目录下的程序测试验证移植的 MQX BSP 是否功能正确。

Source：包含 MQX 内核的所有代码，所有驱动程序代码，以及 BSP 和 PSP 相关代码。

（7）rtcs：TCP/IP 协议栈代码和实例。

（8）shell：shell 代码。

（9）tools：一些辅助的工具。

（10）usb：USB 协议栈代码，包括设备协议栈和主机协议栈两部分，每部分又有几个实例。

4.3 MQX 工程文件组织

嵌入式系统工程往往包含很多文件，如：程序文件、头文件、与编译调试相关的信息文件、工程说明文件以及工程目标代码文件等。工程文件的合理组织对一个嵌入式系统工程尤为重要，它不但会提高项目的开发效率，同时也降低项目的维护难度。

下面将从逻辑结构和物理结构两个层次剖析带 MQX 的工程文件的组织结构。逻辑结构是相对于工程而言的，而物理结构即说明其在存储磁盘介质上的实际分布情况。规范设计的工程组织结构对程序的可复用、可移植，减少重复编码，增强程序稳定性十分有益。下面分别从逻辑与物理两个层次概述 MQX 工程文件的组织情况。

4.3.1 工程文件的逻辑组织结构

嵌入式软件构件（Embedded Software Component，ESC）是实现一定嵌入式系统功能的一组封装的、规范的、可重用的、具有嵌入特性的软件单元，是组织嵌入式系统的功能单位[52]。根据嵌入式底层软件构件的思想，以 MCF52223 的小灯程序为例，介绍基于 CW 7.2 环境的 MQX 工程文件的逻辑组织结构设计，如图 4-2 所示。

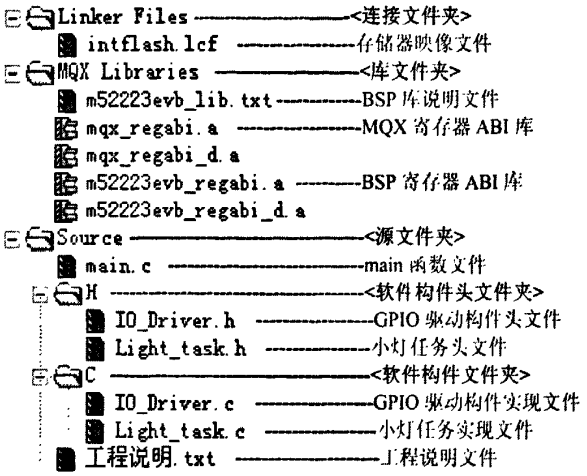


图4-2 MQX工程相关源文件的树型结构

图 4-2 给出了该工程相关源文件的树型结构，可分为:

（1）连接文件目录。该目录包含连接文件 inflash.lcf，用于告诉编译器，代码是如何存放在具体的地址空间，充分了解该文件的格式，有助于全面理解 MQX 的运作。

（2）库文件目录。BSP 库说明文件，用于说明如何在 MQX 中添加库和设备驱动程序。MQX 寄存器应用程序二进制接口（Application Binary Interface，ABI）库，用于支持 MQX 的寄存器参数传递，实现调试配置。BSP 寄存器 ABI 库，用于支持 BSP 的寄存器参数传递，实现调试配置[21]。

（3）源文件目录。该目录的内容是需要用户自己编写的。main 函数文件，包含任务 ID 定义，任务函数声明，以及任务模板列表实例。软件构件头文件夹，包含与硬件设备相关的底层软件驱动程序和任务的头文件。软件构件文件夹，包含与硬件设备相关的底层软件驱动程序的实现程序和任务的实现程序。工程说明文件，用于说明当前工程实现的功能和相关电路连接。

4.3.2 工程文件的物理组织结构

为了方便程序的移植和复用，不仅要从逻辑层面上合理安排工程，也要从存储目录对 MQX 的框架进行合理安排，下面以小灯程序的文件架构加以说明。

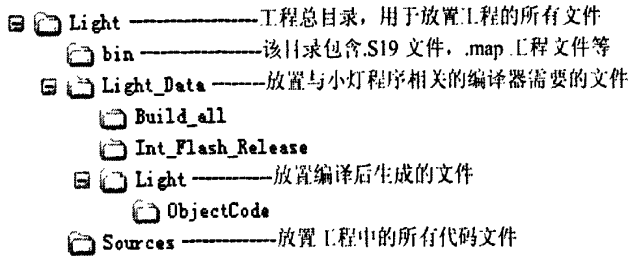


图4-3 MQX工程相关源文件的物理结构

如图 4-3 所示为 MQX 的工程文档结构图，在工程根目录下，除以上目录外，还包含一个 mcp 文件，这是工程的总文件，通过打开它来打开整个工程。

<bin>目录下的.S19 文件为工程最后的生成文件，可以通过写入工具写入到 MCU 芯片里运行。为了方便使用一般文本编辑器阅读，MQX 使用飞思卡尔公司定义的文本方式（.S19）文件存储机器码，优点是可直接使用通用的文件编辑器打开查看，缺点是下载到芯片之前，必须转为二进制机器码。而 bin 目录下的 map 文件为代码在 MCU 存储器中的分配提供了证据，map 文件表明了源代码被编译链接后的机器码，到底被下载到 MCU 内存储器中的什么地方。

<Progect_Data/Light/ObjectCode>目录下存放了工程各.c 文件经过编译后产生的目标代码文件（.o 文件）。这些文件经过"链接"后将生成可以下载到芯片中执行的机器码文件（.S19 文件）。

<Sources>目录下放置了所有程序头文件和源程序文件，但是文件组织没有 CodeWarrior 中那样的逻辑组织。

4.4 编写 MQX 工程

4.4.1 底层构件的实现方法和编程思想

每个底层软件驱动，由头文件和源程序文件两部分组成。

头文件中的内容主要有：包含下层构件头文件的#include 语句、用以描述构件属性的宏定义语句以及对外接口函数原型说明。在头文件中使用函数原型，建立代码模块和外部接口的规范，便于他人使用，是很有帮助的。使用这些函数的用户，不需要查找源代码去了解参数的具体类型，直接查看函数原型即可。

源程序文件中存放构件的内部函数和外部函数的定义，即函数的实现代码，以完成函数所要实现的功能。

在对底层构件进行设计时，最关键的工作是要对构件的共性和个性进行分析，抽

取出构件的属性和对外接口函数[53]。尽量做到：当一个底层构件应用到不同系统中时，仅需修改构件的头文件，对于构件的源程序文件则不必修改或改动很小。

在编写构件时，一般应注意以下几方面的内容：

（1）构件的头文件和源程序文件的主文件名一致，且为构件名。

（2）属性和操作的命名统一以构件名开头。这样做的好处是：当使用底层构件组装软件系统时，避免构件之间出现同名现象。

（3）对 MCU 内的模块寄存器名和端口名进行重定义，在其它的代码里面都将使用宏名对模块寄存器和端口进行操作。这样，当底层驱动程序移植到其它 MCU 时，只要修改重定义语句就可以了。

（4）内部函数与外部函数要设计合理，函数参数个数及类型要考虑全面。内部函数仅提供给同一构件中的其它内部函数或外部函数调用，作用域仅限于定义该函数的文件。外部函数是对外接口函数，供上层应用程序调用。在定义外部函数时，应该对函数名、函数功能、入口参数、函数返回值、使用说明、函数适用范围等进行详细描述，以增强程序的可读性。上层应用程序不能直接对构件的属性进行读取或设置，必须借助于该构件提供的接口操作函数来实现。

（5）应用程序在使用底层构件时，尽量避免通过全局变量来传递参数，所有的数据传递都要通过函数的形式参数来接收。这样做不但使得接口简洁，更加避免了全局变量可能引发的安全隐患。

4.4.2 如何在CW环境下新建MQX工程

新建工程有两种方法，一种是使用工程模板，另一种是使用已存在的工程复制一份继续进行新的工程编程。

第一种方法的操作步骤如下：

选择 File->New Project，弹出新建对话框，选择相应的 MCU，点击“下一步”，选中“C”的选项，如果程序中有汇编代码则应该选中“Relocatable assembly”，在右侧 Project name 中输入工程名，在 Location 中选择工程所在目录。单击确定即可。

第二种方法是使用已存的工程来建立另一个工程。当在已有工程的基础上，做另一个项目时，比如在 Light 工程的基础上编写 LCD 程序，需要进行如下设置：

（1）更改工程目录名为 LCD；

（2）更改 Light.mcp 为 LCD.mcp；

（3）删除 Light_Data 目录；

（4）将 bin 目录的所有内容删掉。

4.4.3 控制小灯闪烁工程设计与实现

本节以 MCF52223 串口接收的数字来控制发光二极管闪烁次数为例，详细阐述如何依照底层软件构件化思想，创建带有 MQX 的工程。在本节的工程实例中，灯的正端引脚接 MCF52223 的普通 I/O 口，负端引脚过电阻接地。当在 I/O 引脚上输出高或低电平时，指示灯就会亮或暗。串口使用 MCF52223 的串口 0，实现 MCF52223 与 PC 机的通信。同时，工程中定义了两个信号量，用来实现串口任务与小灯任务的通信。该工程一共包含三个任务 main_task 任务，led_task 任务，uart0_task 任务。

1. GPIO构件

GPIO 引脚可以被定义成输入、输出两种情况。若是输入，程序需要获得引脚的状态（逻辑 1 或 0）。若是输出，程序可以设置引脚状态（逻辑 1 或 0）。MCU 的 GPIO 引脚分为许多端口（Port），每个口有若干引脚。为了实现对所有 GPIO 引脚统一编程，设计了 GPIO 构件（由 IO_Driver.h、IO_Driver.c 两个文件组成）。这样，要使用 GPIO 构件，只需要将这两个文件加入到所建工程中即可，且只需看头文件中的相关函数说明，由于篇幅所限，本文只在源文件中添加了函数说明。

控制指示灯的亮或暗，通过调用 GPIO 构件完成。设有一盏灯，名称为叫 Light_Run0，它们所接在的 MCU 的 GPIO_PORT_TC 端口第 0 个引脚。这样它们具体接在 MCU 的哪个端口，哪个引脚，只要在 IO_Driver.h 中给出具体宏定义就可以。

1）GPIO构件的头文件IO_Driver.h

```
//-------------------------------------------------------------------------------*
//文件名:IO_Driver.h(GPIO 驱动函数头文件)                                        *
//-------------------------------------------------------------------------------*
#include<mqx.h>                                    //包含操作系统头文件
#include<bsp.h>                                    //包含硬件头文件
static FILE_PTR output_port=NULL;                  //定义文件结构指针
#define Light_PORT     GPIO_PORT_TC                //定义相应的端口，使用 GPIO 的 TC 口
#define Light_Run0 0                               //增加不同引脚，Run0 灯接在相应口的 0 引脚
//函数声明
boolean IO_Init(uint_32 port,uint_32 name,uint_32 state); //IO_Init()函数声明
void IO_Control(uint_32 port,uint_32 name,uint_32 state); //IO_Control()函数声明
```

2）GPIO构件的程序文件IO_Driver.c

IO_Driver.c 相当于硬件与操作系统之间的中间层，定义了如下内容：

```
//-------------------------------------------------------------------------*
//文件名:IO_Driver.c(GPIO 驱动函数文件)                                    *
//-------------------------------------------------------------------------*
#include<IO_Driver.h>
//-------------------------------------------------------------------------*
//函数名:IO_Init                                                           *
//功   能:初始化指示灯状态                                                *
//参   数:port:端口名                                                      *
//        name:指定端口引脚                                                *
//        state:初始状态,1=高电平(暗),0=低电平(亮)                         *
//返   回:引脚设置为 IO 是否成功                                          *
//-------------------------------------------------------------------------*
boolean IO_Init(uint_32 port,uint_32 name,uint_32 state)
{
    uint_32 output_set[] =                                  //填写数组内容
    {
        GPIO_PIN_STATUS_0,
        GPIO_LIST_END
    };
    output_set[0]=port|name|output_set[0];
    output_port=fopen("gpio:write",(char_ptr)&output_set);  //打开 GPIO 设备
    if(output_port)                                         //初始化 GPIO 引脚状态
    {
        ioctl(output_port,(state)?
        GPIO_IOCTL_WRITE_LOG1:GPIO_IOCTL_WRITE_LOG0,(pointer)&output_set);
    }
    return(output_port!=NULL);
}
//-------------------------------------------------------------------------*
//函数名:IO_Control                                                        *
//功   能:控制灯的亮和暗                                                  *
//参   数:port:端口名                                                      *
//        name:指定端口引脚                                                *
//        state:初始状态,1=高电平(暗),0=低电平(亮)                         *
//返   回:无                                                              *
//-------------------------------------------------------------------------*
void IO_Control(uint_32 port,uint_32 name,uint_32 state)
{
    uint_32 light[]=
    {
        GPIO_PIN_STATUS_0,
        GPIO_LIST_END
    };
    light[0]=port|name|light[0];
    if(output_port)                                         //设置 GPIO 引脚状态
    {
    ioctl(output_port,(state)?GPIO_IOCTL_WRITE_LOG1:GPIO_IOCTL_WRITE_LOG0,(pointer)&light);
    }
}
```

2. Light任务

小灯任务包含两个文件 Light_task.h、Light_task.c，Light_task.h 文件添加了头文

件和引用了相应轻量级信号量 UART_SEM、LIGHT_SEM，并定义了小灯任务声明函数；Light_task.c 文件实现了小灯任务，其功能是根据数据值，控制小灯的闪烁次数。

1）Light任务的头文件Light_task.h

```
//-------------------------------------------------------------------------------*
//文件名:Light_task.h (小灯任务头文件)                                            *
//-------------------------------------------------------------------------------*
#include <mqx.h>
#include <bsp.h>
#include <fio.h>
#include <IO_Driver.h>
extern uint_8 times;                                          //引用全局变量
extern LWSEM_STRUCT    UART_SEM;
extern LWSEM_STRUCT    LIGHT_SEM;
extern boolean IO_Init(uint_32 port,uint_32 name,uint_32 state);    //引用外部函数
extern void IO_Control(uint_32 port,uint_32 name,uint_32 state);
void led_task(uint_32);                                       //小灯任务函数声明
```

2）Light任务的程序文件Light_task.c

```
//-------------------------------------------------------------------------------*
//文件名:Light_task.c(小灯任务文件)                                               *
//-------------------------------------------------------------------------------*
#include<Light_task.h>
//-------------------------------------------------------------------------------*
//函数名:led_task                                                                 *
//功   能:根据传来的次数，控制小灯的闪烁的次数                                    *
//参   数:initial_data                                                            *
//返   回:无                                                                      *
//-------------------------------------------------------------------------------*
void led_task(uint_32 initial_data)
{
    uint_8 i;
    _lwsem_wait(&LIGHT_SEM);                          //等待小灯信号量
    for(i=0;i<times-'0';i++)
    {
        Light_Control(GPIO_PORT_TC,Light_Run0,1);     //控制小灯暗
        _time_delay(1000);
        Light_Control(GPIO_PORT_TC,Light_Run0,0);     //控制小灯亮
        _time_delay(1000);
    }
    _lwsem_post(&UART_SEM);                           //释放串口信号量
}
```

3. UART任务

串口任务包含两个文件 Uart_task.h、Uart_task.c，Uart_task.h 文件添加了头文件和引用了相应轻量级信号量 UART_SEM、LIGHT_SEM，并定义了串口任务声明函数；Uart_task.c 文件实现了串口任务，其功能是把接收的 PC 机数据值，返送给 PC 机，

并创建小灯任务。

1）UART任务的头文件Uart_task.h

```
#include<mqx.h>
#include<bsp.h>
#include<IO_Driver.h>                    //小灯驱动头文件
#define LED_TASK 6                       //小灯任务索引值
extern uint_8 times;                     //引用全局变量
extern LWSEM_STRUCT    UART_SEM;
extern LWSEM_STRUCT    LIGHT_SEM;
void uart0_task(uint_32 initial_data);   //串口任务函数声明
```

2）UART任务的程序文件Uart_task.c

```
#include<Uart_task.h>
//-------------------------------------------------------------------------------------------*
//函数名:uart0_task                                                                          *
//功   能:打开串口,把接收的数据发送出去,并根据接收的数据值,控制小灯的闪烁次数              *
//参   数:initial_data                                                                       *
//返   回:无                                                                                 *
//-------------------------------------------------------------------------------------------*
void uart0_task(uint_32 initial_data)
{
    uint_32 c;
    uint_32 flags=0;
    _task_id task_id;
    uint_32 result;
    FILE_PTR serial_fd = fopen("ttya:",0);                    //以轮询的方式打开串口设备0
    IO_Init (GPIO_PORT_TC,Light_Run0,0);                      //初始化小灯为亮
    ioctl(serial_fd, IO_IOCTL_SERIAL_SET_FLAGS, &flags);      //设置串口标志
    result = _lwsem_create(&UART_SEM, 1);                     //创建信号量
    result = _lwsem_create(&LIGHT_SEM, 1);
    if (result!=MQX_OK)
    {
        printf("\nCreating sem failed: 0x%X",result);
        _mqx_exit(0);
    }
    while(TRUE)
    {
        if(fstatus(serial_fd))                                //把接收到的数据发送出去
        {
            _lwsem_wait(&UART_SEM);                           //等待串口信号量
            c=fgetc(serial_fd);                               //接收字符
            times=c;
            if(c==IO_ERROR)break;
            putchar((char)c);                                 //发送字符
            _lwsem_post(&LIGHT_SEM);                          //释放小灯信号量
            task_id=_task_create(0,LED1_TASK,0);              //创建小灯任务
            if(task_id==MQX_NULL_TASK_ID)
            {
                printf("\nCould not create LED_TASK\n");
            }
            else
```

```
            {
                printf("\nLED_TASK created\n");
            }
        }
    }
    fclose(serial_fd);
}
```

4. 测试工程主程序

在主程序中包含任务模板列表，该列表中包含三个任务 main_task 任务，led_task 任务，uart0_task 任务。main_task 是一个自启动任务，该任务创建 uart0_task 任务，uart0_task 任务接收 PC 机串口传来的数据，led_task 任务根据串口数据值，控制小灯任务的闪烁次数。

```
//-------------------------------------------------------------------------------------*
//工 程 名:Light                                                                        *
//硬件连接:MCU 串口 0 与 PC 机串口相连                                                  *
//程序描述:用 I/O 口控制小灯闪烁,串口收发数据                                           *
//日    的:使用 MQX 编写小灯程序和串口程序                                             *
//说    明:提供 Freescale MCU 的编程框架，供教学入门使用                               *
//----------苏州大学飞思卡尔嵌入式系统实验室 2011 年-----------------------------------*
#include <IO_Driver.h>                                          //小灯驱动函数
#define MAIN_TASK 5                                             //任务 ID,定义任务
#define LED_TASK 6
#define UART_TASK 7
uint_8 times;                                                   //定义全局变量
LWSEM_STRUCT    UART_SEM;                                       //定义信号量
LWSEM_STRUCT    LIGHT_SEM;
//任务函数声明
extern void main_task(uint_32);
extern void led_task(uint_32);
extern void uart0_task(uint_32);
//编写任务列表
TASK_TEMPLATE_STRUCT  MQX_template_list[] =
{
    //索引号,    任务函数,任务栈,优先级,名称,    任务属性,              参数,时间片
    {MAIN_TASK,main_task,1500,  9,  "main",MQX_AUTO_START_TASK,0,   0},//主函数任务
    {LED_TASK,led_task,    1500,  10,  "led1",  0,                  0,   0},//小灯任务
    {UART_TASK,uart0_task,1500,  11,  "uart",  0,                  0,   0},//串口任务
    {0,            0,          0,      0,      0,      0,          0,   0} //任务模板结束
};
//-------------------------------------------------------------------------------------*
//函数名:main_task                                                                      *
//功    能:调用串口任务                                                               *
//参    数: initial_data                                                               *
//返    回:无                                                                         *
//-------------------------------------------------------------------------------------*
void main_task(uint_32 initial_data)
{
    int i=0;
    _task_id task_id;
    task_id= _task_create(0,UART_TASK,0);                       //创建串口任务
```

```
    if(task_id==MQX_NULL_TASK_ID)
    {
        printf("\n Could not create UART_TASK\n");
    }
    else
    {
        printf("\n UART_TASK created \n");
    }
}
```

基于底层软件构件思想，对以 MCF52223 为微控制器的 MQX 的其他底层硬件驱动程序进行进行重新编写，主要包含 UART 驱动、SPI 驱动、IIC 驱动、AD 驱动，Flash 驱动等，提高了底层软件的可重用性、可移植性。

4.5 本章小结

本章从 MQX 的工程框架对 MQX 进行剖析。主要包含如下内容：

（1）阐述了 MQX 使用的 CodeWarrior 开发环境的版本类型，对 CW 开发环境的安装过程进行讲解，分析安装过程中的注意点，并论述 MQX 的 CW 开发环境版本号。

（2）对 MQX 的安装框架进行论述，详细说明其安装目录下的各目录的作用和内容，为了解 MQX 的功能提供依据。

（3）为了更深入的理解 MQX 工程组织，对 MQX 工程的逻辑组织结构和物理组织结构进行重新归类、设计，方便底层软件的移植和复用。

（4）为了提高软件的可重用性、可移植性，按照底层软件构件的思想对 MQX 的底层硬件驱动程序进行重新编写，并以串口控制小灯程序为例说明底层软件构件的编写过程。

第五章 MQX 应用实例—自动折弯系统

MQX 在国内市场上市比较晚，应用产品和实例更是少之又少，在前几章阐述了 RTOS MQX 的内核、实现机制和工程的组织结构等基础上，本章将把 MQX 应用到自动折弯系统中，论述该系统的需求分析，详细阐述在 MQX 下的软件实现部分，并给出具体的移植过程，分析系统的主要硬件 LCD 触摸屏，对触摸屏界面进行设计，提出任务划分和设计的依据、完成任务设计和中断程序设计。

5.1 自动折弯系统的需求分析与软件设计

自动折弯系统是笔者和实验室的研究生共同完成的一个项目，该系统包括硬件主控板和软件开发平台。硬件主控板的设计与实现由实验室的其他研究生负责完成，选用的主控芯片为飞思卡尔 ColdFire V2 内核的 MCF52223[54]-[56]。笔者负责软件开发平台的设计与实现，在 RTOS MQX 平台上实现对整个自动折弯系统的控制。依据嵌入式底层软件构件思想，对自动折弯系统软件开发平台进行设计和编写。

5.1.1 需求分析

自动折弯系统工作流程为：高端软件根据要折字的字形制作好折弯数据后，以一定格式存入优盘，MCU 读取优盘中的折弯数据，结合传感器反馈信号和 LCD 触摸屏设置的控制信号对折弯过程进行控制。MCU 在整个系统中的工作是接收 LCD 触摸屏的触摸事件和传感器反馈信息，然后打出控制信号，控制步进/伺服电机、直流电机、电磁阀的下一步操作，从而实现对材料（不锈钢、铝或铁）的自动折弯功能（包含三种操作：送料、开槽、折弯）。自动折弯系统开发平台需要具有以下的功能：

（1）主控板上具有 1 个电源接口，4 路直流电机控制接口，6 路步进/伺服电机控制接口，1 个液晶显示接口，1 组无线发送接收接口，4 路指示灯接口，1 路蜂鸣器接口，14 路电磁阀接口，14 路传感器接口，1 个 USB 接口。

（2）主控板具有与接近传感器、红外传感器、LCD 和 USB 通信的接口。

（3）当系统出现出现故障时，能够发出报警信号。

5.1.2 软件设计

依据嵌入式底层软件构件思想，对自动折弯系统软件开发平台进行设计。主控板的硬件模块主要包含：MCF52223 最小系统、电源构件、直流电机构件、步进/伺服电机构件、电磁阀构件、传感器构件、LCD 接口构件和 USB 构件等，如图 5-1 所示。根据主控板的硬件模块，编写相应的底层软件驱动。自动折弯系统的硬件电路板参见附录 B。

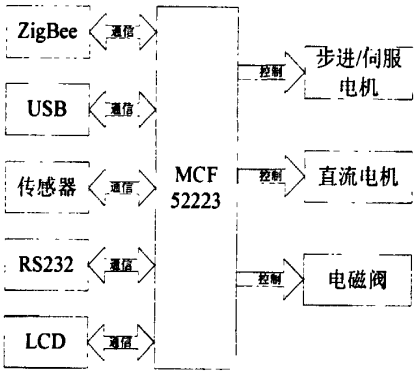


图5-1 自动折弯系统结构图

5.2 MQX 的移植过程

本节把 MQX 移植到自动折弯系统中，详细阐述以 ColdFire V2 系列的 MCF52223 作为硬件平台，MQX 具体移植步骤。如第四章的图 4-1 所示的 MQX 的安装目录中，mqx 目录是移植的重点，需要在 build 目录为自己的板子分别创建 BSP 和 PSP 的工程文件[57]。

1. 选择基准BSP

MCF52223 和 MCF52235 同属于 ColdFire V2 系列微控制器，并且 MQX 提供了 MCF52235 的实例，因此选择 MCF52235 作为基准 BSP。

2. 复制选择的BSP和PSP工程文件，配置文件，源码

在 <MQX_install>\mqx\build\cwcf72 目录下，复制 bsp_m52235evb.mcp 和 psp_m52235evb.mcp，分别命名为 bsp_MCF52223.mcp 和 psp_MCF52223.mcp。

在 <MQX_install>\mqx\build\bat 目录下，复制 bsp_m52235evb.bat 和 psp_m52235evb.bat，分别命名为 bsp_MCF52223.bat 和 psp_MCF52223.bat。

在<MQX_install>\mqx\source\bsp 目录下创建一个新子目录，命名为 MCF52223，

拷贝<MQX_install>\mqx\source\bsp\m52235evb 目录下所有文件到新目录 MCF52223。

在<MQX_install>\lib 目录下创建新目录，命名为 MCF52223.cw，并在<MQX_install>\lib\MCF52223.cw 目录下创建子目录 mqx。

3. 修改

修改 bsp_ MCF52223.bat 和 psp_ MCF52223.bat 文件，将其中所有的 m52235evb 字符串替换为 MCF52223。

将新创建的<MQX_install>\mqx\source\bsp\MCF52223\目录下的 m52235evb.h 文件改名为 MCF52223.h，打开这个文件，将里面所有的 m52235evb 字符串替换为 MCF52223。

将新创建的<MQX_install>\mqx\source\bsp\MCF52223\cw\dbg 目录下的 m52235evb.cfg 和 m52235evb.mem 文件改名为 MCF52223.cfg 和 MCF52223.mem。修改 MCF52223.mem 文件，将里面的 Flash 空间修改为 MCF52223 的 Flash 空间。

修改<MQX_install>\mqx\source\bsp\MCF52223\cw\intflash.lcf 文件，根据 MCF52223 的存储空间分配，修改 ROM 和 Flash 的存储空间范围。

将<MQX_install>\mqx\source\bsp\MCF52223\目录下所有文件，包括.c，.h 以及其他类型的文件中出现的 m52235evb 字符串全部替换为 MCF52223。

4. 修改PSP工程设置

用 codewarrior for Microcontroller V7.2 打开<MQX_install>\mqx\build\cwcf72\psp_MCF52223.mcp。

将 m52235evb user config 重命名为 MCF52223 user config。删除 user_config.h 文件，加入<MQX_install>\config\MCF52223\user_config.h。

点击 Debug RegABI setting 按钮，在弹出的对话框中：

（1）选择 Target Setting，点击 choose，选择 output Directory 为<MQX_install>\lib\MCF52223.cw\mqx。

（2）选择 Access Paths，将{Project}...\...\config\m52235evb 删除，点击 add，加入<MQX_install>\config\MCF52223。

（3）选择 BatchRunner P...，点击 choose，加入<MQX_install>\mqx\build\bat\psp_MCF52223.bat

5. 修改BSP工程设置

用 Codewarrior for Microcontroller V7.2 打开<MQX_install>\mqx\build\cwcf72\

bsp_MCF52223.mcp。

将m52235evb user config重命名为MCF52223 user config。删除user_config.h文件，加入<MQX_install>\config\MCF52223\user_config.h。

将m52235evb BSP Files改名为MCF52223 BSP Files，然后删除下面的所有文件，右键add files...，重新加入<MQX_install>\mqx\source\bsp\<MQX_install>目录下以及其子目录下的所有.h和.c文件。

点击Debug RegABI setting按钮，在弹出的对话框中：

（1）选择Target Setting，Access Paths和BatchRunner P...，把所有的文件的连接都设置为MCF52223。与修改PSP工程设置类似，在此不再累述。

（2）选择Coldfire Target，将File Name里的m52235evb_regabi_d.a换成MCF52223_regabi_d.a。

6. 其他修改

若硬件初始化不同，如芯片时钟不同，则要重新编写硬件初始化程序。在<MQX_install>\mqx\source\bsp\MCF52223\bsp_init.c文件中进行修改。若MQX没有提供与硬件相应的设备驱动程序，要用户自己编写，并进行添加。

5.3 LCD触摸屏界面设计

为了实现用户与MCU的交互，自动折弯系统提供了LCD触摸屏，方面用户对整个自动折弯过程进行实时的调整和控制。

5.3.1 LCD触摸屏简介

本系统采用的LCD触摸屏型号为DMT80600T080，该模组包含TFT液晶屏和四线电阻式触摸屏，使用MAX232与MCU通信，外设简单，具有较高的性价比[58]。

LCD触摸屏采用帧格式与MCU通信，数据在帧中采用十六进制表示，帧头固定为0xAA，帧尾固定为0xCC，0x33，0xC3，0x3C，在帧头帧尾之间的是帧内容，其长度由传递的指令类型决定[59]。帧格式如表5-1所示。

表5-1 帧格式

指令	帧头	帧内容		帧尾
含义	0xAA	指令(1字节)	数据	0xCC 0x33 0xC3 0x3C

从LCD触摸屏读取的帧命令包含三种类型：(1)读取的帧命令长度为18字节时，表示LCD触摸屏与MCU在进行握手操作。(2) 读取的帧命令长度为8字节时，表

示触摸屏有触发事件。(3) 读取的帧命令长度为 9 字节时，表示要设置 LCD 触摸屏的工作模式。写入 LCD 触摸屏的帧命令长度，根据操作指令类型不同，包含很多种，在此不再一一列举。

5.3.2 界面设计与实现

自动折弯系统包含 4 个主要界面：自动模式、手动模式、设备信息、系统设置。

自动折弯系统有两种工作模式：自动模式和手动模式。自动模式工作时，先在 PC 的高端软件机上制作折弯数据，以一定格式存入优盘，底端通过读取优盘数据进行各种折弯控制。用户通过 LCD 触摸屏选择要加工的形状，之后折弯机自动进行加工，实时显示加工进度，并提供暂停、继续、停止、复位等操作，当材料长度不够时，提示用户拼接材料。手动模式工作时，用户可以通过 LCD 触摸屏手动控制机械部件工作，即对送料、退料、折弯、折断、开槽等操作一步步的手动进行控制。

设备信息界面对自动折弯系统中硬件设备进行检测，并显示检测结果。

由于加工材料质地、长度、步进电机步进精度等的不同，需要用户输入不同的参数，从而根据这些参数进行更精确的控制，参数的输入操作由系统设置界面完成。

1. 自动模式

自动模式界面包含三个选项卡：导入任务、任务模拟、任务设置，如图 5-2 所示。导入任务用于从优盘把要加工的任务导入 MCU 的 Flash；任务模拟用于模拟材料的加工过程，显示加工进度；任务设置用于设置加工的参数，如材料的长度、厚度、加工次数等。

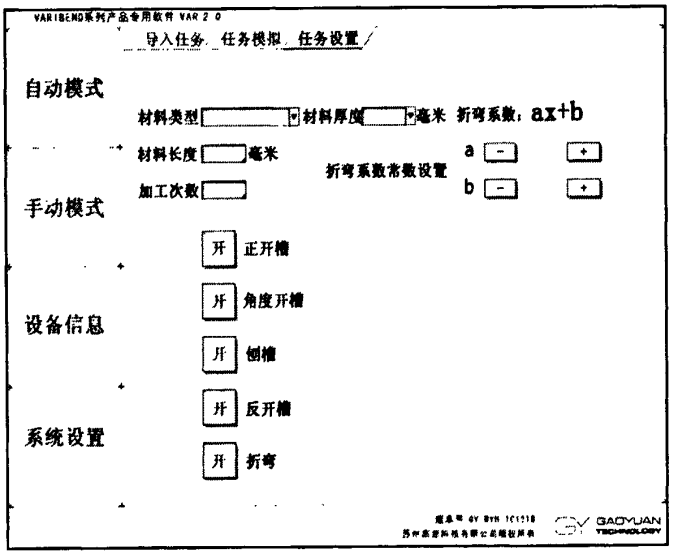


图5-2 自动模式界面

2. 手动模式

手动模式包含三个选项卡：送料机构、开槽机构、折弯机构，如图 5-3 所示。送料机构完成对送料、退料操作的手动控制。开槽机构完成对开槽动作、角度设置和刨槽动作的手动控制。折弯机构完成折弯的手动控制。

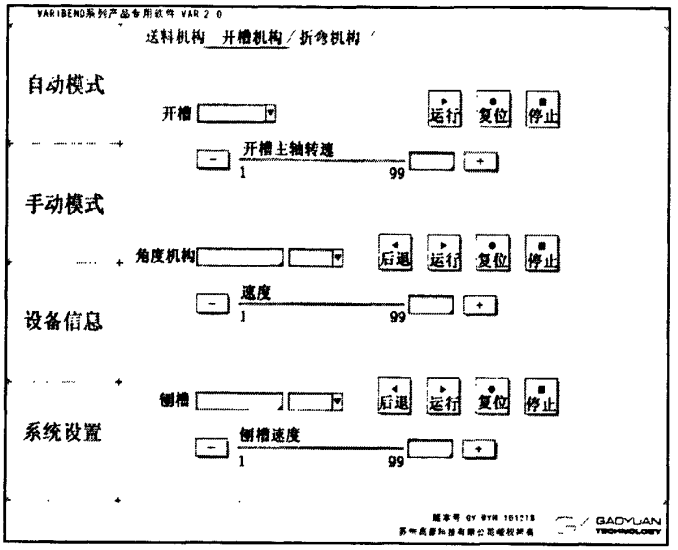


图5-3 手动模式界面

3. 设备信息

设备信息用于检测硬件电路板上各模块的工作状态，主要对传感器、电磁阀、指示灯和直流电机进行检测，如图 5-4 所示。指示灯包含运行指示灯和故障指示灯。传感器自检方法有接近传感器接近测试、红外传感器遮挡测试。

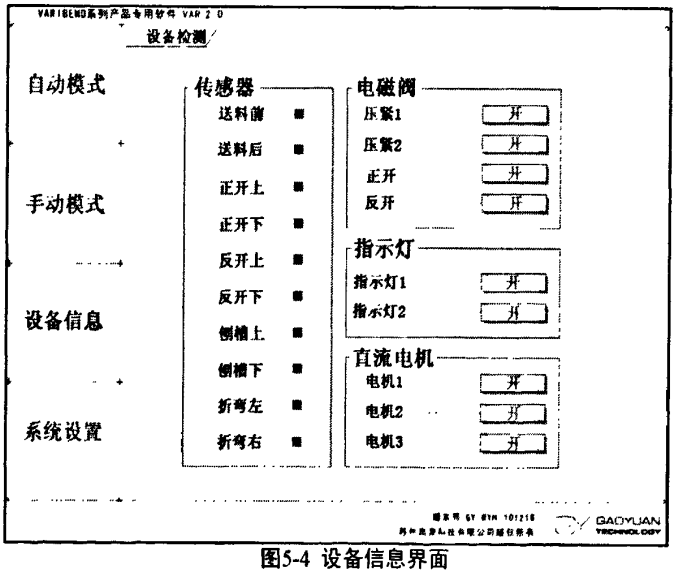


图5-4 设备信息界面

4. 系统设置

系统设置包含两个选项卡：设备设置和屏幕设置，如图 5-5 所示。设备设置主要包含：拼接距离，折弯最小间距，最大折弯角度，折弯回转角度等。屏幕设置是对 LCD 屏进行设置，主要包含：LCD 时间、日期，按键声音，背光强度和背光时间等。

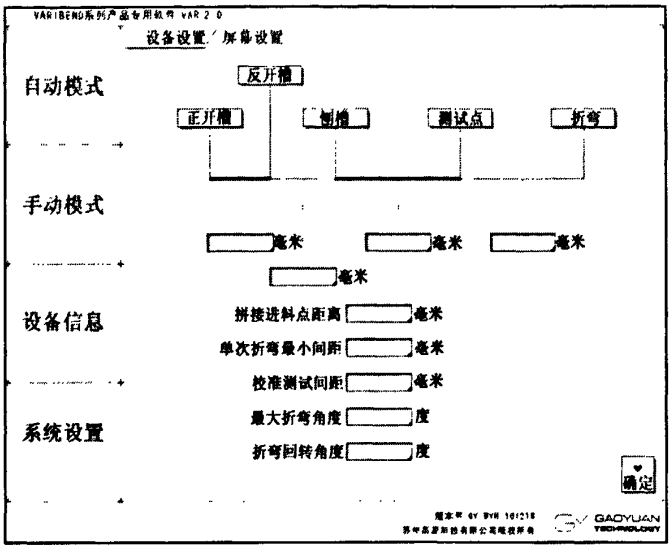


图5-5 系统设置界面

5.4 任务及中断程序的设计

5.4.1 任务设计

在抢占式 RTOS 中将一个软件系统划分为并行任务时，首先要分析数据流图中数据的变换，对数据流图进行分解，确定哪些变换可以并行，哪些变换本质上必须顺序执行。把数据流图中的每个处理映射为软件结构中一个适当的模块，并使用设计度量和启发式规则对分解的软件结构进行进一步精化，从而使一个变换成为一个任务，或者几个变换组成一个任务[28]。

决定系统中任务划分的最主要的因素是系统中所实现功能间的异步关系，即任务与任务间是如何相互触发和协调的，这可以通过任务间的通信来解决。在设计实际的应用系统时，根据系统的实际需求分析，从几个不同侧面考虑构造任务：I/O 功能、内部功能、任务内聚、任务优先级。对每个侧面，都有一些原则用于决定一个变换是单独组成一个任务还是和其它的变换共同组成任务[60]。

自动折弯系统的软件结构主要划分为以下任务：蜂鸣器任务、串口通信任务、LCD 管理任务（接收 LCD 触摸屏的命令任务、发送命令到 LCD 触摸屏任务）、电磁阀任

务、传感器任务、直流电机任务、步进电机任务、USB 通信任务和 Flash 任务，如图 5-6 所示。任务之间采用信号量进行通信和同步，使用全局变量进行数据交换。

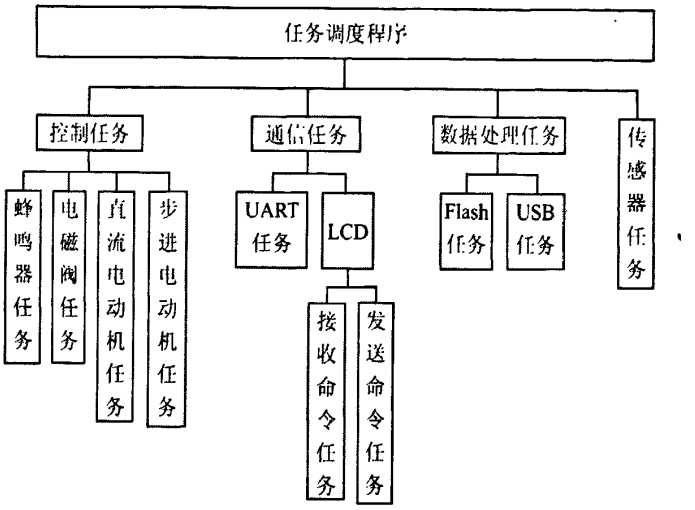


图5-6 系统任务框图

蜂鸣器任务主要用于发出提示信息。系统上电，对传感器、电磁阀、指示灯、直流电机等进行自检，若没有通过自检，蜂鸣器就会发出声音，进行提示。

电磁阀任务用于控制电磁阀的工作和停止。通过电磁阀的打开和关闭，对正开槽上、下气控制，反开槽上、下气控制，开槽压紧控制，折弯左、右轴控制，以及换刀控制等。

直流电机任务通过设置 PWM 的周期和占空比，控制直流电机的转速、从而实现材料正、反开槽时进行切削加工动作。

步进电机任务要控制两个 I/O 口，一个 I/O 口用 PIT 定时模块控制脉冲间隔从而控制步进电机转动，一个 I/O 口用于控制方向。通过调整电机的转速，用于控制送料、开槽和折弯操作。

通信任务包含串口（UART）通信任务和 LCD 通信任务。UART 任务主要是在软件程序调试过程中使用，可以模拟 LCD 触摸屏的帧格式调试 LCD，也可以把传感器接收的信号发送到 PC 机等。

LCD 任务包含主要包含两个任务：接收命令任务和发送命令任务。接收命令任务是中断服务程序，在所有通信任务中优先级最高，主要用于接收 LCD 发送来的帧命令，根据帧命令的长度，判断要执行的操作类型，从而使 MCU 做出相应的操作。系统启动时，它会挂起自己，当接收到数据帧后才转入运行状态。如图 5-7 所示为接收 LCD 命令任务执行流程图。

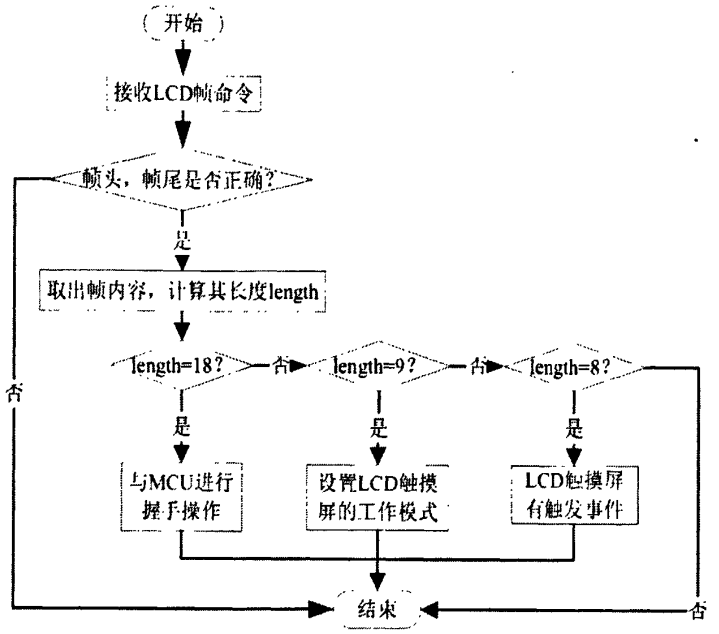


图5-7 接收LCD命令任务执行流程图

发送命令任务实现把 MCU 命令封装成帧，放到发送缓冲区，发送到 LCD 触摸屏，任务执行流程图如 5-8 所示。

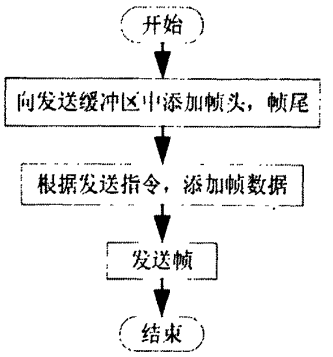


图5-8 发送命令到LCD任务流程图

USB 通信任务把可加工的文件从优盘导入 MCU 的 Flash。可加工的文件通过高端软件处理，以一定的格式保存的优盘中。

Flash 任务实现对 MCU 的 Flash 的读写操作。

传感器任务用于获得传感器引脚状态，并把引脚状态传送给 MCU，MCU 根据接收的信号对下一步操作进行控制。在系统中，主要用于感应正、反开槽的位置，以及折弯的位置。

5.4.2 中断程序设计

外部I/O设备的速度通常与处理器的速度完全不同，要比处理器慢很多。若某些设备的许多工作通常是在与处理器完全不同的的时间周期内完成的，这就会发生让处理器等待外部事件的情况，这样会明显降低处理器的效率，所以必须要有一种方法可以让外部I/O设备在产生某个事件时通知处理器，这种方法称为中断。一个中断仅仅是一个信号，当硬件需要获得处理器对它的关注时，就可以发送这个信号。

中断服务程序的功能就是将有关中断接收的信息反馈给设备，并根据正在服务的中断的不同含义对数据进行相应的读或写。中断服务程序的特殊之处在于中断服务程序是在中断时间内运行的，因此它的行为会受到限制。中断服务程序的限制包含：（1）不能向用户空间发送或者接收数据，因为它不是在任何任务的上下文中执行；（2）不能做任何可能发生休眠的情况；（3）不能调用任务调度函数。

MQX处理中断的方式在很大程度上与它在用户空间处理信号时一样，通常一个驱动程序只需要为它自己设备的中断注册一个处理程序，并且在中断到达时进行正确的处理。

自动折弯系统中设置了一个中断处理程序，该程序用于接收LCD触摸屏的命令。LCD触摸屏事件是用户触发事件，用于控制自动折弯系统的操作。LCD触摸屏驱动程序的成员函数为：io_LCD_int_install()，io_LCD_int_open()，io_LCD_int_close()，io_LCD_int_read()，io_LCD_int_write()，io_LCD_int_ioctl()，io_LCD_int_isr()。LCD ISR的功能是接收LCD触摸屏发送来的帧，判断帧长度，根据帧长度，转向不同的任务进行处理，ISR需要声明的驱动程序的结构和全局变量如下：

```
typedef struct io_ lcd _init_struct                    //定义 LCD 端口初始化时的参数
{
      _mqx_uint          QUEUE_SIZE;    //队列大小，用于缓冲输入/输出数据
      uint_32            DEVICE;        //初始化设备
      uint_32            VECTOR;        //中断驱动时的中断向量
      _int_level         LEVEL;         //中断驱动时的中断等级
}LCD_INT_STRUCT;
typedef struct io_ lcd _ info_struct                   //定义实时运行时的状态信息
{
      LCD_INT_STRUCT    INIT;           //当前的初始化值
      pointer OLD_ISR_DATA;             //先前的中断处理和数据
      void (_CODE_PTR_ OLD_ISR)(pointer);
   } LCD_INFO_STRUCT;
```

5.5 本章小结

本章主要阐述了 MQX 的应用实例——自动折弯系统的软件设计，主要从以下几个方面进行阐述：

（1）对自动折弯系统的需求进行分析，从而得到自动折弯系统的软件平台的组成。根据基于硬件构件的嵌入式底层软件构件思想，对 MQX 的软件平台进行划分和设计，编写软件平台的组成模块。

（2）把 MQX 移植到自动折弯系统的微控制器 MCF52223 上，详细阐述了移植步骤、移植过程中需要修改和配置的文件，论述移植过程中的关键技术。

（3）分析了系统的主要硬件 LCD 触摸屏的帧结构，对触摸屏界面进行设计，论述了主要界面的组成部分和功能，主要界面包含：手动模式界面、自动模式界面、设备信息界面和系统设置界面。

（4）提出 RTOS 中断的任务划分和设计依据。对自动折弯系统中的任务进行划分和设计，详细阐述了各任务的作用和实现。分析了 MQX 中的中断程序的作用，完成中断程序设计。

第六章 总结和展望

6.1 总结

随着计算机技术与芯片技术的飞速发展和广泛应用，RTOS 逐渐走向成熟，并有向行业标准化、网络化方向发展趋势。目前，RTOS 种类繁多，应用领域比较混杂，难于学习，而且各类 RTOS 或多或少存在一些缺点，缺乏一款通用的 RTOS。MQX 具有完全开放源代码，较高的可裁减性与可移植性，占用 ROM 空间少，采用 API 和模块化架构，同时还具有良好的持续发展能力、可定制的微内核结构、友好的用户开发环境，这些优点必将使其迅速在 RTOS 领域占据一席之地。它以其良好的可靠性和卓越的实时性被广泛地应用在嵌入式项目的研发中。本文结合相关资料文献和实际条件，以 RTOS MQX 为基础，完成了以下内容：

（1）提出了 RTOS 的划分方法，根据该方法对当前主流 RTOS 进行比较分析，得出其优缺点，从而引出对 MQX 分析研究的必要性，阐述本课题的课题背景和研究依托，归纳总结本课题主要研究工作和课题意义。

（2）深入剖析 MQX 的微内核结构的组成，详细阐述了 MQX 内核的任务管理、时间管理、存储管理、中断管理、任务同步和通信的相关内容，对理解 MQX 的内核提供参考和学习资料。

（3）阐述了MQX的启动执行过程，分析了Bootloader的加载和启动执行过程，直到MQX从自启动任务开始运行；论述了MQX任务调度策略的缺点，为克服该缺点，在MQX中引入多级反馈队列调度策略；详细说明了MQX的设备驱动程序的设计与实现过程。

（4）论述了 MQX 使用的开发环境 CodeWarrior，并详细讲解 CW 开发环境的安装过程；详细说明 MQX 的安装框架中的各目录的作用和内容；为了提高软件的可重用性、可移植性，对 MQX 工程的逻辑组织结构和物理组织结构进行重新归类、设计，按照底层软件构件的思想对 MQX 的底层硬件驱动程序进行重新编写。

（5）把 MQX 移植到自动折弯系统的微控制器 MCF52223 上，详细阐述了移植步骤、移植过程中要修改和配置的文件，论述移植过程中的关键技术，对系统中的 LCD 触摸屏界面进行设计，并提出 RTOS 中断的任务划分和设计依据，完成自动折弯系统的任务和中断程序设计。

6.2 展望

本文研究并分析了 RTOS MQX 的内核结构、实现过程，实现了 MQX 的成功移植，并应用到自动折弯系统中，使得 MQX 在嵌入式系统中的应用得到了一定程度的推广。为了达到更好的可用性和更高的效率，还需要围绕以下几个方面做进一步的研究：

（1）虽然 MQX 系统提供了 TCP/IP 协议栈，但实现过程本文并未研究，可以进一步研究 MQX 的以太网协议栈，使 MQX 真正成为网络上独立的、功能丰富的、具有使用价值的客户端。

（2）虽然 MQX 内核高效，但其外围功能还有待完善，以后还要添加 MQX 图形接口等功能。

（3）为适应不断发展的需求，以后很可能需要考虑把无线通信的功能添加 MQX 系统中。例如可以移植 Zigbee 协议栈到 MQX 系统中，并在此协议栈上编写应用程序，实现无线通信。

参考文献

[1] Wolf W，Madsen J. Embedded systems education for the future[J]. Proceedings of the IEEE，2000,88(1):23-30.

[2] 钟锡昌. 嵌入式操作系统在中国的发展现状与前景[J]. 信息技术与标准化，2002,(6):6-9.

[3] 白英彩. 嵌入式多任务实时操作系统核心 VRTX32[J]. 电脑开发与应用，1992,5(4):7-14.

[4] 何小庆. 嵌入式实时操作系统的现状和未来[J]. 单片机与嵌入式系统应用，2001,(3):12-17.

[5] 刘波，马连川，张建明. 嵌入式实时操作系统选用的初步分析[J]. 北方交通大学学报，2000,24(5):105-108.

[6] Mohamed Tag Elsir，Patrick Sebastian，Yap Vooi Voon. A RTOS for Educational Purposes[CA]. International Conference on Intelligent and Advanced Systems (ICIAS)，2010:1-4.

[7] 王宜怀，张书奎，王林，吴谨. 嵌入式技术基础与实践（第 2 版）[M]. 北京：清华大学出版社，2011.

[8] Paul Parkinson，Franco Gasperoni. High-Integrity Systems Development for Integrated Modular Avionics Using VxWorks and GNAT[C]. Blieberger and Strohmeier，2002:163-178.

[9] Jean J.Labrosse 著，邵贝贝译. 嵌入式实时操作系统 μC/OS-II(第 2 版)[M]. 北京：北京航空航天大学出版社，2003:88-91.

[10] Rick Lehrbaum . Using Linux in Embeded and Real time Systems[J/OL]. http://www.linuxdevices.com.

[11] P Mantegazza，Bianchi E，Dozio L，Papacharalambous S. RTAI: Real-Time Application Interface[J]. Linux Journal Magazine，2000,4:105-108.

[12] Cheng Fei，Miao Ke-jian，Wang Rui-min. Analysis and real-time performance tests of QNX and VxWorks[J] . Computer Engineering and Design，2008,29(18):4734-4739.

[13] Anthony J Massa. 嵌入式可配置实时操作系统 eCos 软件开发[M]. 北京：北京航空航天大学出版社，2006.

[14] Dong Yibing，Wang Lang. Design of Embedded Cropland Data Acquisition and Processing System Based On Xscale and WinCE[J]. International Forum on Information Technology and Applications，2009,31(13):22-119.

[15] Sarah Allen，Vidal Graupera，Lee Lundrigan. Pro Smartphone Cross-Platform Development iPhone，BlackBerry，Windows Mobile，and Android Development and Distribution[M]. Paul Manning，2010:35-50.

[16] Thomas Badura，Michael Becher. Testing the Symbian OS Platform Security Architecture[CA]. Advanced Information Networking and Applications，2009:838-844.

[17] Anon. ARC international precise/MQX[J]. Penton Publishing Co，2002,50(12):90-94.

[18] MQX RTOS[DB/OL]. http://en.wikipedia.org/wiki/MQX.

[19] Freescale MQX™ RTOS 3.0.1 Release Notes[DB/OL]. http://www.freescale.com.

[20] Freescale MQX™ RTOS 3.4.0 Release Notes[DB/OL]. http://www.freescale.com.

[21] Freescale MQX™ RTOS 3.6.2 Release Notes[DB/OL]. http://www.freescale.com.

[22] Freescale MQX™ Real-Time Operating System User's Guide [DB/OL]. http://www.freescale.com.

[23] Anon. Precise/MQX RTOS Gets User-configurable[J]. Cahner Publishing Co，2001,46(12):20-25.

[24] Freescale MQX™ RTCS™ User's Guide[DB/OL]. http://www.freescale.com.

[25] PHDC 支持的 USB 堆栈[DB/OL]. http://www.freescale.com.

[26] Gien M. Micro-kernel Architecture Key to Modern Operating System Design[EB/OL]. http://www.chorus.com/documentation/ref.html.

[27] 赵艺伟，张丽芬，陈朔鹰. 一个实时操作系统的设计及实现[J]. 北京理工大学学报，2001,21(1):113-115.

[28] 谈发明. 基于中低端单片机的抢占式 RTOS 精简设计[D]. 南京. 南京理工大学，2008.

[29] 李江等. 嵌入式操作系统设计中的若干问题[J]. 微型机与应用，

2000,19(8):13-15.

[30] 汤子瀛，哲凤屏，汤小丹著. 计算机操作系统[M]. 西安：西安电子科技大学出版社，2006:37-38.

[31] MQX RTOS Hands-On[DB/OL]. http://www.freescale.com.

[32] Freescale MQX™ RTOS Reference Manual[DB/OL]. http://www.freescale.com.

[33] Ren Peng，Xiang Zheng. A Multitask Scheduling Algorithm For Vxworks: Design and Task Simulation[CA]. International Conference on Artificial Intelligence and Computational Intelligence，2009,3:353-357.

[34] 刘兴亮. Linux 2.6 中断系统与调度算法的实时性分析与研究[D]. 辽宁. 辽宁工程技术大学，2008.

[35] 曹祥根. 基于 RAM 的 μC/OS-II 应用研究[D]. 四川. 四川大学，2005.

[36] 禹农，孙祥斌. 嵌入式实时操作系统核心的设计与实现[J]. 计算机工程与科学，2002,24(4):38-40.

[37] Hands-On Workshop: Freescale MQX Drivers and BSP's[DB/OL]. http://www.freescale.com，2010.

[38] Yan Li，Ping-Ping Gu，Xian-Shan Wang. The Implementation of Semaphore Management In Hardware Real-time Operating System[J] . Information Technology Journal，2011,10(1):63-158.

[39] 赵成. 基于 RTOS 的嵌入式温湿度控制系统的设计与应用[D]. 济南. 山东大学，2007.

[40] Tao Wang，Daxin Liu. Avoiding Unbounded Priority Inversion by Integrating Task Inheritance and Classification into Preemption Threshold Scheduling[CA]. ISSCAA，2006: 593-596.

[41] Jaehong Shim，Kyunghee Choi，Gihyun Jung，Seungkyu Park，HyeonSik Shin，Dongyoon Kim. Priority inversion handling in microkernel-based Real-Time Mike[CA]. Real-Time Computing Systems and Applications，1996:238-245.

[42] Shu Hong-xia，Wang Ji-hong. Design and Implementation of Message Passing for Distributed Real-time Operating System[J]. Computer Engineering and Design，2008,29(9):2239-2245.

[43] Xiaofeng Wan，Hailin Hu，Xiaojun Zhou. Analysis and Design of Bootloader on

Embedded Electric Power Steering System[CA]. ISECS International Colloquium on Computing, Communication, Control, and Management(CCCM), 2008,2:276-280.

[44] 宋刚. 嵌入式实时操作系统任务调度算法研究与改进[D]. 沈阳. 沈阳航空工业学院，2009.

[45] Hessel F, Marcon C, Santos T. High Level RTOS Scheduler Modeling for A Fast Design Validation[CA]. IEEE Computer Society Annual Symposium on VLSI, 2007:6-10.

[46] 黄斌. 多级反馈队列调度策略在Linux中的应用和实现[J]. 计算机工程，2004，30(20):81-83.

[47] Alessandro Rubini . Linux Device Drivers[M] . Sebastopol , CA O'Reilly&Associates, Inc, 1998.

[48] Freescale MQX™ I/O Drivers Users Guide[DB/OL]. http://www.freescale.com.

[49] Ji Chan Maeng, Jong-Hyuk Kim, Minsoo Ryu. An RTOS API Translator for Model-driven Embedded Software Development[CA]. 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2006:363-367.

[50] How to Develop I/O Drivers for MQX[DB/OL]. http://www.freescale.com.

[51] CodeWarrior Development Studio for ColdFire Architectures V7.2[DB/OL]. http://www.freescale.com.

[52] 王宜怀，陈建明，蒋银珍著. 基于32位ColdFire构建嵌入式系统[M]. 北京：电子工业出版社，2009.

[53] 荐红梅. 基于硬件构件的嵌入式底层软件开发方法研究及其应用[D]. 苏州. 苏州大学计算机科学与技术学院，2007.

[54] 申忠如，陶慧斌，曹建安. ColdFire 嵌入式系统设计[M]. 西安：西安电子科技大学出版社，2006.

[55] MCF52223 Data Sheet Rev 2[DB/OL]. http://www.freescale.com.

[56] MCF52223 ColdFire® Integrated Microcontroller Reference Manual Rev3[DB/OL]. http://www.freescale.com.

[57] MQX BSP 移植指南[DB/OL]. http://www.freescaleic.org.

[58] DMT80600T080 智能显示终端数据手册[DB/OL]. http://www.dwin.com.cn.

[59] DMT80600T080 指令集列表[DB/OL]. http://www.dwin.com.cn.

[60] 王春铭，刘振华，郭云飞. 实时操作系统中应用软件设计的任务划分[J]. 计算机工程，2000,26(7):190-192.

公开发表的学术论文及参与的主要科研项目

[1] 程玉娟，王宜怀，姚健东．基于二代身份证的 RFID 门禁考勤系统．计算机应用与软件，2011,28(3):44-46.

[2] 姚健东，王宜怀，程玉娟．图形化嵌入式开发平台的通用性设计．计算机应用与软件．（已录用）.

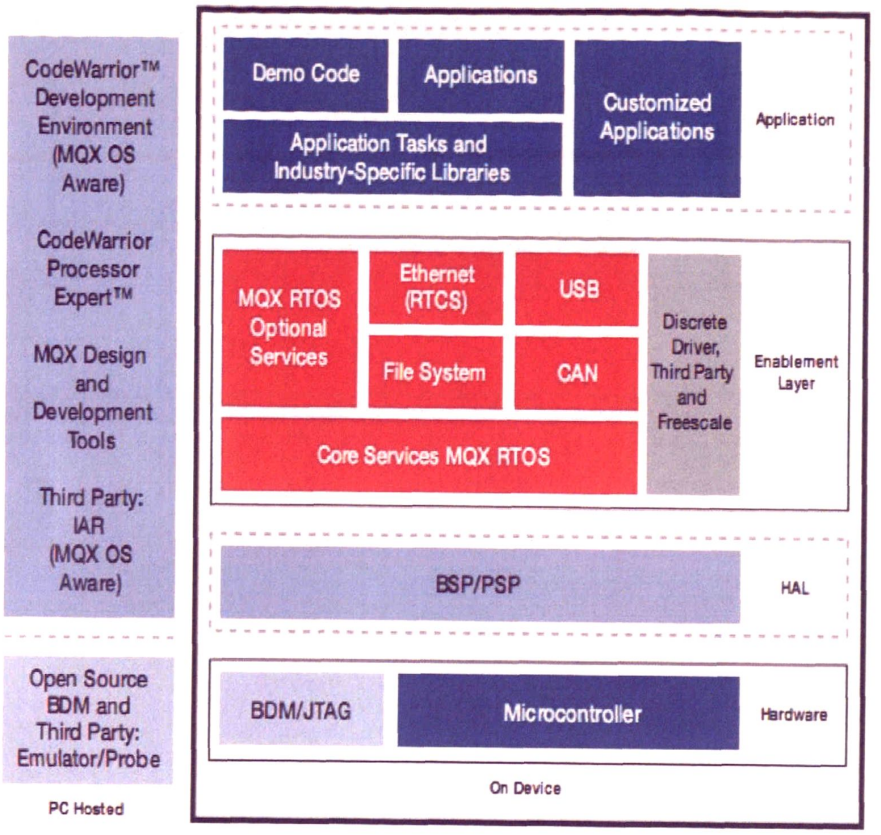
[3] 常赛，程玉娟．一种高实时性的工业以太网的构建方法．军民两用技术与产品．2011 第 2 期.

[4] 参与开发了“基于二代身份证会议签到系统”项目，该项目的软件已申请软件著作权，软著登字第 0171463 号.

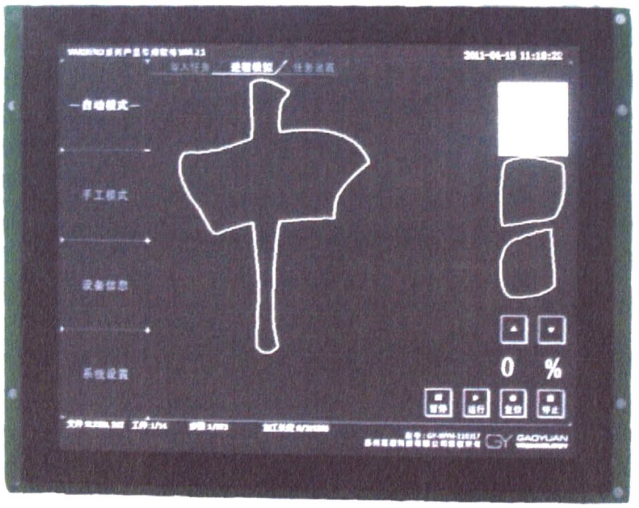
[5] 参与了《嵌入式技术基础与实践（第 2 版）》（王宜怀，张书奎，王林，吴谨著）一书的编写工作，该书已与 2011 年出版.

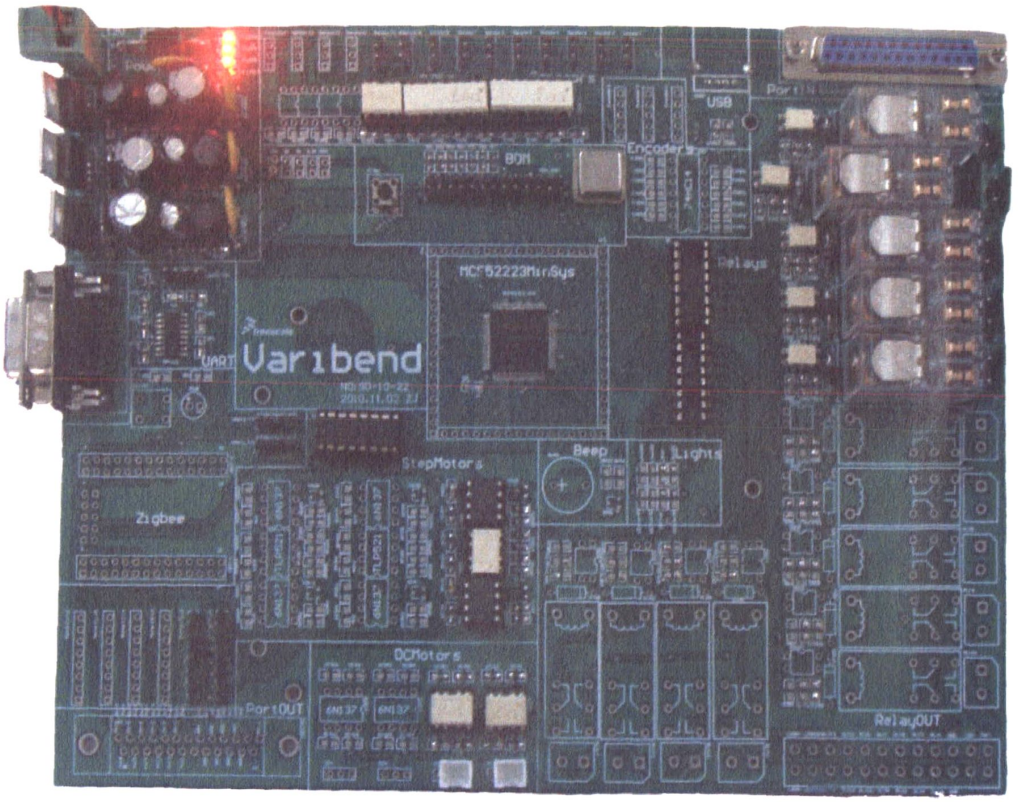
[6] 参与了 2009 全国“电脑鼠走迷宫”竞赛，并获得长三角赛区三等奖.

附录 A MQX 层次结构图



附录 B 自动折弯系统的硬件





致 谢

时光如水，生命如歌。转眼间，三年的研究生生活即将接近尾声，在此我首先要感谢我的导师王宜怀教授。在我攻读硕士学位期间，王老师不仅为我提供了良好的学习条件和实践锻炼机会，还非常注重对我学习方法的培养。在他的严格要求和悉心指导下，我逐步掌握了如何规范地进行嵌入式系统的开发，并且具备了一定的分析问题、解决问题的能力。王老师平时严谨的治学态度、执着的科研精神和朴素的生活作风，更是让即将走上工作岗位的我体会并懂得了做人的道理。

真诚感谢实验室的兄弟姐妹们，感谢你们营造的浓厚学习氛围。感谢赵伟、杜承虎、顾霞萍等同学在本文研究过程中给予的无私帮助。感谢周杰、李翠霞、冯上栋等同学在论文修改期间提供的大力支持。

特别感谢我的家人，你们的爱是我前进的动力。

最后，向审阅本文的专家、教授致敬！