基于 ARM Cortex-M4 的 MQX 中断机制分析与中断程序框架设计

石 晶 王宜怀 苏 勇 沈 忱
(苏州大学计算机科学与技术学院 苏州 215000)

摘 要 中断机制是决定 RTOS 实时性指标的核心机制。MQX 是一个由 Freescale 维护、源代码公开、支持多任务的抢占式的 RTOS,将会广泛用于 ARM Cortex-M 微处理器的应用中。MQX 的中断机制具有实时响应、动态管理的特点。以 ARM Cortex M4 Kinetis 微控制器为蓝本,深入分析了 MQX 的中断顶半部和底半部的运行机理,提出了 MQX 的中断实时特性的评估算法,明确了程序时间的可控性。在此基础上,根据嵌入式软件工程的基本原理,不拘泥于传统程序结构设计方法,提出了一种 MQX 下中断程序框架及编程要素分布的基本原则,从而较好地满足了程序可复用性及可移植性要求。

关键词 MQX,ARM Cortex-M4,Kinetis,中断机制,中断程序框架
中图法分类号 TP316 **文献标识码** A

Analysis of MQX Interrupt Mechanism and Design of Interrupt Program Frame Based on ARM Cortex-M4

SHI Jing WANG Yi-huai SU Yong SHEN Chen
(College of Computer Science and Technology, Soochow University, Suzhou 215000, China)

Abstract Interrupt mechanism is the core mechanism which decides the instantaneity of the RTOS. MQX is an open source, multitask support, preemptive RTOS which is maintained by Freescale. It will be widely used in the application of the ARM Cortex-M microprocessor. MQX interrupt mechanism has the characteristics of real-time response and dynamic management. The paper analysed the MQX interrupt operation mechanism of top half and bottom half, based on the ARM Cortex M4 Kinetis series microcontroller, and put forward the evaluation algorithm which is used to describe the instantaneity of the RTOS, making the controllability of program running time clear. On this basis, according to the embedded software engineering basic principles, not constrained by the traditional program structure design methods, this paper proposed a basic principle describing the interruption program structure under MQX and the distribution of programming elements, which satisfies the requirements of program reusability and portability.

Keywords MQX, ARM Cortex-M4, Kinetis, Interrupt mechanism, Interrupt program frame

1 引言

中断是嵌入式系统获取外界事件的基本手段,是嵌入式实时操作系统(real-time operating system, RTOS)的重要组成部分。中断机制设计的质量直接影响到 RTOS 运行效率,是任何 RTOS 实时性问题的研究基础和实现起点。RTOS 的中断机制研究一直是嵌入式操作系统应用研究的热点之一。MQX(Message Queue eXecutive)是飞思卡尔半导体公司 2009 年在国内推出的一款源代码开放、可裁剪性强、占用 ROM 空间少的 RTOS,具有巨大的市场前景和应用价值[1]。目前国内比较流行的几款嵌入式操作系统如 Ucos、VxWorks[2]、WinCE,各自都有对中断处理的方式,可以说是各有千秋。不同于 MQX 操作系统,这些操作系统在国内推广时间较长,因此受到国内学者深入的研究。较 Ucos 将中断服务例程(Interrupt Service Routine, ISR)地址直接置于中断向量表中的管理方式[3],MQX 使用静态中断向量表管理中断顶半部,动态中断向量表管理中断底半部,该中断管理机制具有响应稳定、灵活多变等特点。本文以 freescale K60 芯片为例说明了 MQX 中断检测、响应及处理的过程,分析了 MQX 中断的运行机理[4,5];同时针对 K60 硬件平台上 MQX 中断机制的性能评估,归纳出了评估算法公式,基于此公式可计算中断延迟,明确程序时间的可控性。

目前业界对嵌入式操作系统下的编程框架没有统一标准,导致嵌入式操作系统程序代码可移植性差,结构混乱,大大降低了程序开发效率,增加了后期维护难度,令软件成本居高不下。设计出好的框架能更好地展示出中断机制的优点,一个良好的框架结构更是一个规范工程的基本。为解决上述存在的问题,本文在充分分析 MQX 中断机制的基础上,融合

到稿日期:2012-08-29 返修日期:2012-12-02 本文受国家自然科学基金项目(61070169)资助。
石 晶(1987-),男,硕士生,主要研究方向为嵌入式系统、传感网、智能控制,E-mail:20104227042@suda.edu.cn;王宜怀(1962-),男,博士,教授,博士生导师,主要研究方向为嵌入式系统、传感网、智能控制;苏 勇(1986-),男,硕士生,主要研究方向为嵌入式系统、智能控制;沈 忱(1989-),男,硕士生,主要研究方向为嵌入式系统、智能控制。

软件工程中可靠性、可重用性、可维护性的基本思想，利用嵌入式构件的软件理论[6,7]，设计出了一种 MQX 中断程序框架，提出了中断框架设计的基本原则。传统的中断程序架构受限于当时的软件理论与实现技术，随着软件技术的发展，结构新颖、运行稳定的工程框架必然将取而代之。本文提出的中断程序框架以构件的方式组织文件，工程结构清晰，调试定位方便，后期维护容易。所以，本文中针对 MQX 中断机制的分析及中断程序框架的设计，对 MQX 操作系统的推广和工业领域的应用有着重要的作用。

2 MQX 中断机制原理

2.1 中断环境初始化

MQX 中断运行环境是实现中断机制的基础。系统上电时，MQX 将中断向量表置于 Flash 中，称不支持动态更新的该中断向量表为静态中断向量表。因 Flash 的读取速度较快，系统并没有将静态中断向量表复制到 RAM 中，而直接置于 Flash 中。相比 Ucos 等操作系统，MQX 并非直接将 ISR 地址存放于 RAM 地址内的中断向量表中，而把实际使用到的中断进行动态管理，这个实际使用到的中断向量表被称为动态中断向量表。以链表的形式管理动态中断向量表，链表中的元素为中断向量表链表项元素，表项元素内容包括 ISR 地址、中断向量号、ISR 运行参数地址及中断发生错误时调用的 ISR 地址。所有中断信息以链表队列的组织形式存于堆地址空间中。把连续 8 个外设模块向量号对应的中断信息项归为一组管理，如图 1 所示，n 的值为中断个数除以 8 所得。

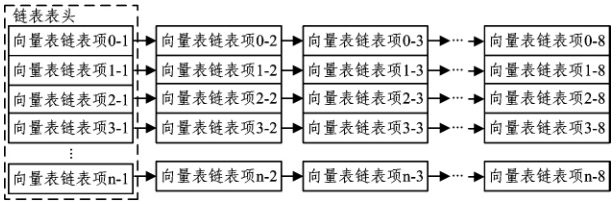

图 1 中断向量链表组织形式

静态中断向量表的 ISR 地址设置为默认 ISR 的地址。默认 ISR 为一段极为短小、快捷的汇编程序，在响应中断时，它完成保存处理器中断上下文、确定所要求的中断服务、调用中断服务程序、在中断服务程序返回时恢复中断现场等工作。而 NMI、fault 等系统异常服务例程地址也初始化为 MQX 提供的各自默认的异常例程地址。

设备中断会打断内核中任务的正常调度和运行，系统对更高吞吐率的追求势必要求中断服务程序尽可能地短小精悍。但是现实往往与愿望并不吻合。在大多数真实的系统中，中断到来时，要完成的工作往往并不会是短小的，它可能要进行较大量的耗时处理。不管中断服务程序执行期间系统中断是否被屏蔽，系统中断服务程序都不宜过长，过于庞大的中断服务例程会增加中断延迟时间，影响实时性，给系统带来风险。为了在尽可能缩短中断执行时间与中断处理需完成大量工作之间找到一个平衡点，MQX 将中断处理程序分解为两个半部：顶半部和底半部。

顶半部完成尽可能少且较紧急的任务，即简单地完成读取寄存器中中断状态、保存中断上下文、找到对应的底半部中断任务地址等工作后开放总中断，执行中断底半部任务，MQX 中默认 ISR 便是顶半部任务。顶半部任务执行内容少、速度快，这样便可服务更多的中断请求。而系统将用户自定义中断服务内容主体置于底半部任务中，推迟到相对安全的时间点执行。底半部相对顶半部来说并不是非常紧急，而是比较耗时的。顶半部中断任务不同于底半部，往往被设计成不可屏蔽中断，而底半部中断任务会开放总中断以允许中断优先级嵌套，这种方式的好处是在底半部工作期间，顶半部仍然可以继续为新中断服务。MQX 中的用户自定义外设 ISR 为底半部任务。MQX 的这种机制最大程度地兼顾了系统中断的实时性和稳定性。而 Ucos 操作系统并没有将中断分为顶半部和底半部，而是直接将中断服务例程整合为一个整体管理[3]。对于执行量较小的 ISR，这种管理方法更为便捷快速；但是对于任务里较重的 ISR，Ucos 的中断管理机制并没有 MQX 实时与稳定。MQX 顶半部和底半部中断机制示意图如图 2 所示。

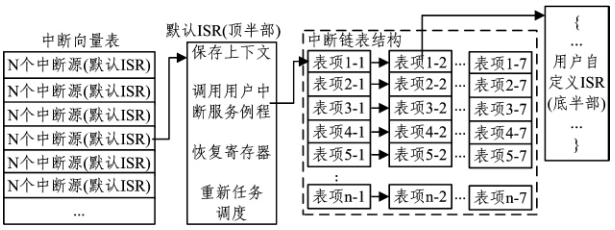

图 2 MQX 顶半部和底半部中断机制

MQX 中断初始化包括中断向量初始化、中断栈初始化及中断信息链表初始化[8,9]。过程归纳如下：(1) Flash 中申请静态中断向量地址空间，初始化 ISR 为默认 ISR。(2)初始化系统中断栈，为中断栈开辟足够的系统空间。(3)在堆中申请向量信息链表结构空间。图 3 为 MQX 中断初始化过程。

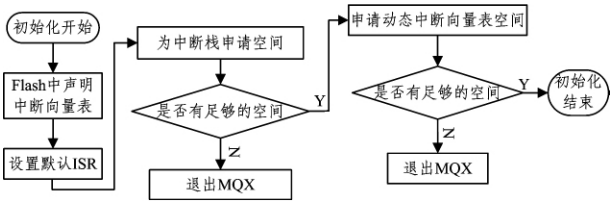

图 3 中断环境初始化流程图

在 MQX 初始化过程中若出现任何错误，MQX 调用 _mqx_exit函数退出 MQX，所谓退出 MQX 其实就是执行一段“死循环程序”。在外界用户看来会出现系统“死机”的状况。在系统初始化过程中，中断栈不宜申请太大，否则系统将无法满足这么大的内存要求而发生错误，退出 MQX；而应根据可能的最大中断嵌套深度，每嵌套一级中断，至少需保存 32 个字节(内核寄存器内容)于堆栈空间，同时配合估算中断服务例程可能运用的堆栈大小。事实上准确计算中断栈大小往往是不可能的，也容易过于保守而造成内存空间的浪费。一般情况是先在系统中设置足够大的空间给中断栈，在程序调试的过程中，时间长了就大概能估算中断栈的需求，最后再来设定中断栈的大小较为合适。这里 MQX 默认使用 1kB 大小的中断栈。

2.2 中断响应

使用 MQX 中断之前，用户必须要调用 MQX 提供的中断安装函数(_int_install_isr)，将中断对应的 ISR 即中断底半

部首地址、ISR 参数地址等信息打包成图 1 中所示的向量表链表项结构加入到动态中断向量表中。若未调用中断安装函数，则中断发生时 MQX 首先调用顶半部任务（默认 ISR）执行，通过中断向量号在动态中断向量表中查找具有相同中断向量号的向量表链表项；若未找到对应链表项，则 MQX 会直接阻塞产生该中断的任务，设置该任务的任务描述符（task descriptor）中“任务状态”字段为“未定义中断错误状态”。MQX 安装中断的过程如图 4 所示。中断向量号以参数的形式传给 _int_install_isr() 函数，错误返回是指该函数判断参数错误，返回一个“NULL”表示中断安装失败。

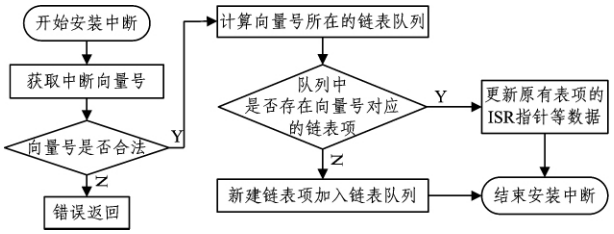

图 4　中断安装流程

当某一中断源需要 CPU 为其服务时，会向 CPU 发出中断请求信号。K60 微处理器检测到中断请求信号后自动完成如下动作。(1)保存被中断任务的上下文，即将 K60 内核寄存器 R0－R3、R12、LR、PC、xPSR 存到系统中断栈，在中断返回时自动弹出它们。由于 Cortex M4 架构使用双堆栈机制，即主堆栈（MSP）和进程堆栈（PSP）[10]，若响应中断，当前代码正在使用 PSP，则还要将 PSP 寄存器存于中断栈中；否则就将 MSP 寄存器存于中断栈中。一旦进入中断服务例程，将一直使用主堆栈。(2)更新指示当前中断向量号的 IPSR 寄存器内容。IPSR 为一个 32 位寄存器，该寄存器的低 8 位指示了当前中断的中断向量号。通过查找静态中断向量表，设置 PC 寄存器值为相应中断 ISR 地址，此时 PC 的值为默认 ISR 地址。(4)更新 LR 寄存器（链接寄存器）的值。LR 寄存器是一个 32 位寄存器，出入 ISR 的时候，LR 值将被重新赋值，在进入 ISR 时由系统计算并赋给 LR，并在中断返回时使用它。赋给 LR 的值称为“EXC_RE_TURN”，该值除了低 4 位外其它位全部为 1，并且最低 4 位有其具体的含义，如表 1 所列。

表 1　EXC_RE_TURN 值

数值	功能
[31:4]	EXC_RE_TURN 的标识：必须全为 1。
3	0＝返回后进入 Handler 模式，1＝返回后进入线程模式
2	0＝从主堆栈出栈后使用 MSP，1＝从进程堆栈出栈后使用 PSP
1	保留，必须为 0
0	0＝返回 ARM 状态，1＝返回 Thumb 状态，在 M4 中必须为 1

任务运行于线程模式，而中断运行于 Handler 模式。从表 1 得出合法的 EXC_RE_TURN 值只有 3 个即 0xFFFF_FFF1：中断结束后返回 Hander 模式；0xFFFF_FFF9：中断结束后返回线程模式，并使用主堆栈（SP＝MSP）；0xFFFF_FFFD：返回线程模式，并使用线程堆栈（SP＝MSP）。

2.3　中断处理

ARM 内核完成其中断响应动作后，开始调用中断顶半部的任务执行，即执行初始化设置的默认 ISR。默认 ISR 根据 IPSR 寄存器记录的中断向量号，查找动态向量表中具有相同中断向量号的向量表链表项，并取出其中自定义的 ISR 地址，跳转到该地址执行 ISR，即执行中断的底半部内容。该过程示意图如图 5 所示。

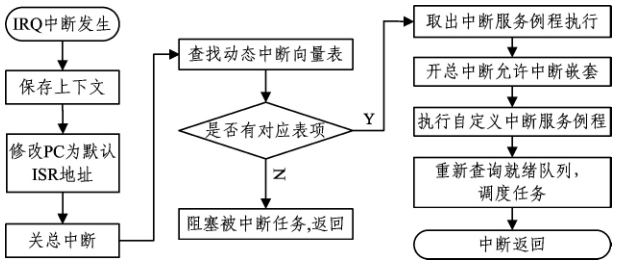

图 5　中断处理流程

执行完成后将重新进行一次任务调度，查看优先级最高的就绪队列中的第一个任务，即优先级最高、符合运行条件、将要激活的任务。若就绪队列中第一个任务仍是被中断的任务，则继续执行该任务；若不是，说明更高优先级的任务在中断服务执行期间满足运行条件，MQX 将调度该新任务。中断响应和处理的流程如图 5 所示。

3　MQX 中断性能分析

3.1　中断嵌套

Cortex M4 的中断控制器（Nested Vectored Interrupt Controller，NVIC）是管理中断的核心部件。通过 NVIC 控制器中的中断优先级寄存器（Interrupt Priority Register）可以配置中断的优先级[11]。高优先级中断可以抢占并挂起低优先级中断，同时将低优先级中断上下文保存于系统中断栈中，以便下次中断恢复时继续执行。当一个高优先级的中断执行完后，系统会在挂起的中断中选择一个优先级最高的中断继续执行，该过程全由 ARM Cortex M4 的 NVIC 自动完成。由于中断只使用中断栈，每嵌套一级中断，至少需保存 32 个字节（内核寄存器内容）于堆栈空间，加之 ISR 运行过程中也需利用中断栈空间，当系统出现较深的嵌套中断时，系统对中断栈空间的需求较高。所以中断环境初始化时，应视最深中断深度，合理分配中断栈空间大小。

3.2　中断延迟

中断的延迟是指从检测到某中断请求、再到执行其服务例程的第 1 条指令所经历的时间。一个中断的中断延迟时间取决于中断优先级，表现为最高优先级中断会最先得到执行，其它中断将会被延迟执行。中断延迟受以下 4 个与时间相关的变量影响：

C—内核屏蔽中断的最长时间；

D—中断源发出中断信号到硬件响应中断的时间；

L—内核默认服务例程查找动态中断向量的时间；

R—ISR 执行与中断现场还原的时间。

ARM Cortex-M4 中从一个中断源发出中断请求信号到硬件响应中断请求的延迟时间固定为 12 个周期，这 12 个周期内系统执行入栈和取址等一系列动作。若中断信号连续发生，ARM_Cortex_M4 则可优化中断与中断间的硬件响应间隔时间至 6 个周期。综合上述因素，中断延迟时间（interrupt latency time，ILT）如式(1)所示：

$$ILT=C+D+L+\sum_{n=0}^{N}(D+L+R) \tag{1}$$

式中，N 是在所关注的中断之后发生的较高优先级且非屏蔽中断的数量。

理想情况下，系统中一个优先级最高的非屏蔽中断发生时，系数 $C=0$，$N=0$，式(1)可简化为式(2)：

$$ILT=D+L \tag{2}$$

即理想情况下中断延迟时间等于硬件中断响应时间(固定为12个周期)加上默认中断服务例程 _int_kernel_isr 查找动态中断向量表的时间。而最坏的情况下中断延迟时间要看同一时间系统悬挂了几个高优先级的中断，优先级最低的中断必须等到比它优先级高的中断得到响应并执行完中断服务例程退出后才能得到服务，它的时间是不确定的，决定于式(1)中 N 的大小。从式(1)得出，较紧急的中断应当设定较高的优先级以使 N 较小，从而保证其实时性。

4 MQX 中断程序框架设计

嵌入式操作系统工程往往包含很多文件，如操作系统源代码文件、编译相关的信息文件、集成开发环境相关文件、工程说明文件以及目标代码文件等。正因如此，工程文件的合理组织对一个嵌入式操作系统工程尤为重要，既可提高项目的开发效率又可降低项目代码的结构复杂度，保障操作系统的实时性。

框架是一个能够被开发人员实例化的系统构架，规定了应用软件的体系结构，定义了模块和对象的分割，确定了各部分的主要职责、协作关系及控制流程。框架设计遵循软件工程的基本思想，应具有可移植、可重用、层次结构清晰、易于后期维护的特点。基于上述若干点核心设计要点，提出了遵循嵌入式构件化思想的中断程序框架，即对所有操作系统任务功能的共性和特性进行归纳，抽取出共性部分，将其封装成一组规范的、可复用的具有嵌入特性的软件单元置于特定的文件目录中。

MQX 程序由 PSP(processor Support Package)工程、BSP(Board Support Package)工程和 APP(Application)工程 3 大部分组成。APP 工程在 MQX 系统开发的过程中有特殊的地位。APP 工程执行仅调用 MQX 系统提供的接口，不直接操纵底层硬件，用户需求都在 APP 工程下实现。APP 工程利用 BSP 和 PSP 工程生成的库文件和功能完善的函数库作为系统库(Lib 库)调用。Lib 库再配合软件构件形成了程序框架的软件平台，APP 工程中设计的任务功能都以此平台为实现基础。程序框架层次模型如图 6 所示。

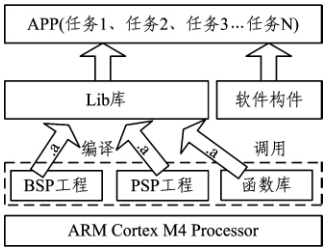

图 6 程序框架层次模型

本文在应用层设计的 APP 工程中断程序框架运用 MQX3.8 版本，其在 CodeWarrior10.2 集成开发环境下的工程结构如图 7 所示。

app_sdk60n512 : sdk60n512_Int_Flash_Debug	
app_sdk60n512_sdk60n512_Int_Flash_Debug_PnE	存放下载调试配置信
Binaries	生成的机器码映像文件
Includes	所有的引用路径
SeAnalysispointsManager.apconfig	
sdk60n512_Int_Flash_Debug	工程创建生成目录
Sources	自定义应用目录
app	应用文件目录
app_inc.h	应用任务总包含头文件
isr.c	ISR文件
isr.h	ISR文件头文件
Lib	引用库文件目录
swComponents	软件构件文件目录
Task	任务文件目录
task_listener.c	自定义任务1
task_start.c	自定义任务2
task_uart.c	自定义任务3
tasks.c	任务模板列表文件

图 7 应用工程在 CodeWarrior 10.2 下的工程树

代码都放于中断程序框架的 Sources 文件夹下。按照功能需求，取出共性与特性，将分类后的代码文件分别放于 Sources 文件夹下的 4 个子文件夹内，这样程序框架简单清晰、结构化组织明了，便于对程序开发的错误定位，大大提高了工程组织的规范性和合理性。

Sources 文件夹下的 4 个目录放置文件原则如下。

app：应用程序目录。用户编写应用程序执行逻辑的代码文件在该目录下。其中 includes.h 文件是应用任务总头文件，isr.c 中包含了中断函数的实现代码，isr.h 是 isr.c 文件的头文件，存放中断函数声明。

Lib：引用库文件目录。在操作系统使用中，可使用他人已开发好的驱动库文件，例如 BSP 工程和 PSP 工程生成的包含所有 MQX 操作系统调用接口的库文件。

swComponents：软件构件目录。软件构件是指具有移植性、不与底层硬件打交道、能够完成一系列相关功能的代码集合。在嵌入式开发过程中，使用一些稳定的、移植性良好的软件构件，例如某些通信协议设计代码、pid 算法和一些通用函数等，可以有效地丰富系统功能，提高系统的开发效率和稳定性。

Task：任务文件目录。操作系统应用程序主体由任务构成，对操作系统程序的设计本质便是对操作系统下任务的设计。该目录下存放用户所创建的任务文件。任务文件设计的基本共识是“一个任务一个文件”，这样任务需求结构清晰，易于理解。

除统一的目录组织结构外，中断程序框架还必须将底层调用接口重新封装成应用层统一调用接口。否则应用程序很难做到平台化，即程序跨平台时，需要重构程序内容，从而将诸多不稳定因素引入应用程序。统一应用层调用接口后，若应用程序的用户需求不变，而仅将应用程序转移到不同处理器平台或软件平台时，只需修改封装的底层调用接口便可实现应用程序移植。

结束语 本文通过对 MQX 中断机制的深入分析，阐述了 MQX 中断结构中顶半部和底半部的基本思想与实现方

(下转第 79 页)

法小。

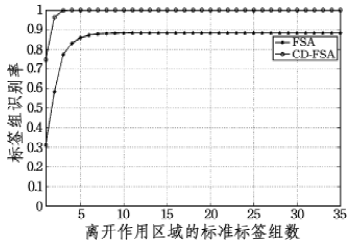

图 4　两种算法的标签组识别率比较

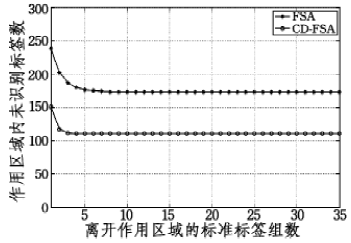

图 5　两种算法的未识别标签数比较

当然，由于引入 CDMA 思想，且标签返回的信号是经正交序列码字调制的信号，读写器端需要增加存储所有正交序列码字的存储器和解正交码字的模块，而标签端也需要增加存储各自正交序列码字的存储器。因此，基于 CD-FSA 算法的 RFID 系统对系统的硬件复杂度有一定的要求，即系统成本有所上升。

结束语　本文提出了一种基于码分多址思想的帧时隙 ALOHA 算法，减小了标签碰撞概率，提高了标签识别率，加快了标签的识别过程，然后针对现实应用中标签与读写器存在相对运动的情况建立数学模型并进行了分析。实验结果表明，无论标签是在静止还是运动状态，CD-FSA 算法的标签识别率都优于 FSA 算法。当然，新算法的性能是在牺牲一定的硬件复杂度的基础上获得的，对应用于特殊场合的 RFID 系统的防碰撞算法设计具有一定的参考价值。

参 考 文 献

[1] Finkenzeller K. RFID-Handbook Fundamentals and Applications in Contactless Smart Cards and Identification (Second Edition)[M]. New York: John Wiley & Sons Ltd, 2003

[2] 王平，胡爱群，裴文江. 一种基于码分复用机制的超高频 RFID 防碰撞方法[J]. 电子与信息学报，2007，29(11)：2637-2640

[3] Lee S-R, Joo S-D, Lee C-W. An Enhanced Dynamic Framed Slotted ALOHA Algorithm for RFID Tag Identification[C]// IEEE Proceedings of the Second Annual International Conference on Mobile and Ubiquitous Systems. Washington, DC: IEEE Computer Society, 2005: 166-172

[4] 翟永，徐进. 一种用于 RFID 系统的防碰撞算法[J]. 计算机工程，2009，39(5)：272-274

[5] Choi J H. Query tree-based reservation for efficient RFID tag anti-collision[J]. IEEE Communications Letters, 2007, 11(1): 85-87

[6] Yang Lei, Han Jin-song, Qi Yong. Revisting Tag Collision Problem in RFID Systems[C]// IEEE 39th International Conference on Parallel Processing. 2010: 178-187

[7] 贺洪江，丁晓叶，翟耀绪. 标签运动状态下的 RFID 系统反碰撞算法[J]. 计算机应用，2011，31(08)：2048-2051

[8] 梁彪，胡爱群，秦中元. 一种新的 RFID 防碰撞算法设计[J]. 电子与信息学报，2007，29(9)：2158-2160

[9] 刘拓晟，何怡刚，侯再平. 超高频射频识别的防碰撞算法设计[J]. 计算机工程与应用，2008，44(36)：87-89

[10] 李泽兰，何怡刚，刘拓晟. 基于广义地址码的 RFID 防碰撞算法[J]. 计算机测量与控制，2010，18(5)：1114-1117

(上接第 44 页)

法，总结了 MQX 顶半部具有屏蔽总中断、快速响应及快速退出特征；底半部具有一般任务或函数特征，在开放总中断背景下，允许中断嵌套等，从而为清晰理解 MQX 中断机理奠定了良好基础。文章研究了 MQX 中断机制的实时性能，归纳出中断延迟公式，根据该公式可准确计算出延迟时间、评价系统实时性能，从而为增强程序稳定性和时间可控性评估提供理论依据。文章根据软件工程原理融合了嵌入式构件思想，根据可复用与可移植的基本要求提出了一种利用程序分层设计、文件分类管理、统一函数调用接口的中断程序框架，为 MQX 的应用编程提供了一个结构清晰、层次分明、可复用与可移植性强的模板。本文工作对 MQX 的应用研究与程序开发具有重要的参考价值。

参 考 文 献

[1] Freescale. Freescale MQX Real-Time Operating System User's Guide[EB/OL]. http://www.freesacle.com, 2011-04

[2] 王运盛，王坚. VxWorks 实时操作系统中的中断处理机制[J]. 电讯技术，2007，8(2)：2-4

[3] Labrosse Jean J. 嵌入式实时操作系统 uC/OS-II(第 2 版)[M]. 邵贝贝，等译. 北京：北京航空航天大学出版社，2003：61-72

[4] 王宜怀，吴瑾，蒋银珍. 嵌入式系统原理与实践—ARM Cortex-M4 Kinetis 微控制器[M]. 北京：电子工业出版社：45-74

[5] Freescale. K60 Sub-Family Reference Manual[EB/OL]. http://www.freesacle.com, 2011

[6] 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，2008，4：36-44

[7] Richard H C, Janell A. Review of current embedded system hardware, OS, development systems and application domains for instructional design[C]// Conference Proceedings ASEE Annual Conference and Exposition. 2007: 16

[8] Freescale. Freescale MQX RTOS Reference Manual[EB/OL]. http://www.freesacle.com, 2011-04

[9] Freescale. MQX RTOS 3.8.1 Release Notes[EB/OL]. http://www.freesacle.com, 2012-06

[10] Yiu J. ARM Cortex-M3 权威指南[M]. 宋岩，译. 北京：北京航空航天大学出版社，2009，7：11-41，110-154

[11] ARM. Cortex-M4 Devices Generic User Guide [EB/OL]. http://www.arm.com