文章编号:1671-4598(2014)05-1658-03　　中图分类号:TP391　　文献标识码:A

MQX－RTOS 底层设备驱动构件可复用性研究

朱仕浪[1,2]，王宜怀[2]，冯德旺[1]
(1. 福建农林大学 计算机与信息学院，福州　350002；2. 苏州大学 计算机科学与技术学院，江苏 苏州　21500)

摘要：针对 MQX 实时操作系统下如何有效复用已经成熟的底层设备驱动构件，提高操作系统下底层设备驱动软件的开发速度、效率和质量等问题；在详细分析和研究 MQX 实时操作系统 IO 设备驱动三层管理体系构架基础上，将底层设备驱动和操作系统剥离，提出了 MQX 四层设备驱动管理体系框架；通过设计底层设备驱动接口构件，成功地复用了已有的底层设备驱动构件，避免底层驱动软件重复开发，有效缩短嵌入式软件开发的周期，并通过实例证明了该体系框架的可行性。

关键词：软件复用；MQX；设备驱动；接口构件

MQX－RTOS Underlying Device Driver Component Reusability Study

Zhu Shilang[1,2]，Wang Yihuai[2]，Feng Dewang[1]
(1. College of Computer and Information，Fujian Agriculture and Forestry University，Fuzhou　350002，China；
2. School of Computer Science and Technology，Soochow University，Suzhou　215006，China)

Abstract：How to effectively reuse the underlying device driver that has matured components，in MQX real－time operating system，and improve the underlying operating system device driver software development speed，efficiency and quality issues etc. Detailed analysis and research MQX RTOS IO device driver three－tier management system architecture，the underlying device driver to be stripped from the operating system，and MQX four－tier device driver management system framework be proposed. The underlying device driver components have been reused successfully by designing the underlying device driver interface components，and avoid duplication of development underlying driver software，effectively shorten the development cycle of embedded software，and through examples prove the feasibility of the system framework.

Key words：software reuse；MQX；device driver；interface component

0　引言

软件的复用是软件产品成熟的标志，复用成熟高质量的软件产品可以降低软件开发的难度、减少重复劳动、降低开发成本、提高开发效率和软件质量以及缩短软件开发周期[1-3]。目前，软件复用成为软件工程的研究热点，是解决软件危机的重要手段[4]。软件构件技术是支持软件复用的核心技术，但如何将成熟的嵌入式软件构件复用到操作系统中，与操作系统拼接是一个值得探讨和研究的问题。MQX 是一款面向工业控制、消费电子、支持多任务的嵌入式实时操作系统，它提供了设备驱动管理的三层体系结构机制，便于用户通过标准的接口函数对设备进行访问[5]。但是这种体系结构开发的底层设备驱动程序与操作系统耦合紧密，不易于设备驱动程序的移植和复用，也难以复用已有的成熟底层设备驱动构件。文章针对这些问题，详细分析和研究了 MQX 设备管理机制，提出了设备驱动管理的四层体系结构机制，将底层设备驱动程序和操作系统剥离，使底层设备驱动与操作系统无关联，有效解决了底层驱动的复用和移植的问题。

1　MQX 设备管理机制

1.1　设备管理的三层体系结构

MQX 对设备的驱动和访问采用三层体系结构，分别为标准应用输入输出层、输入输出子系统层和底层设备驱动层，如图 1 所示[6]。

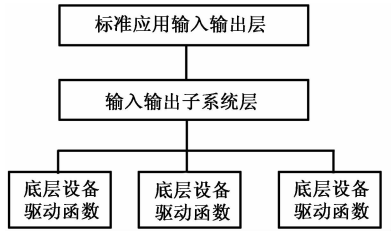

图 1　MQX 设备驱动三层管理结构

在标准应用输入输出层，MQX 为用户提供了标准的 ANSI（美国国家标准协会，American National Standards Institute）文件接口操作界面，包括 fopen、fclose、read、write、ioctl 等函数，用户任务可以使用这些标准的文件接口函数对外部设备进行访问操作。实际上，MQX 是将 ANSI 标准调用函数通过 # define 宏定义，将这些标准函数一一映射到对应的功能函数上，表 1 列出了部分标准文件接口函数的映射关系。对应的接口功能函数通过对输入输出子系统层进行访问，再由输入输出子系统层提供的设备属性信息和底层驱动构件，实现具体的功能。

表 1　ANSI 标准调用函数的宏定义

# define	fopen	io_fopen//打开文件操作
# define	fclose	io_fclose //关闭文件操作
# define	read	io_read //从设备缓冲区读取数据
# define	write	io_write //向设备缓冲区写数据
# define	ioctl	io_ioctl //向设备发送控制命令

收稿日期：2013-11-27；　**修回日期**：2014-01-18。
基金项目：国家自然科学基金资助项目(60871086)。
作者简介：朱仕浪(1970-)，男，福建沙县人，讲师，硕士，主要从事嵌入式与物联网技术方向的研究。

输入输出（I/O）子系统层是标准应用输入输出层和底层设备驱动层的枢纽，是 MQX 设备管理的关键层，负责对 IO 设备进行调配和管理。其主要的任务是将标准的 fopen、fclose、read、write、ioctl 以及设备的私有属性信息与底层设备驱动程序进行关联，以方便 MQX 标准层可以间接地对底层设备以文件的方式进行访问和调用。

为实现输入输出（I/O）子系统层的枢纽功能，MQX 采用设备驱动管理队列的组织形式；并提供了专门的系统调用 API 函数对设备驱动管理队列进行维护和管理，这些维护管理函数主要有：_ io _ dev _ install、_ io _ dev _ install _ ext、_ io _ dev _ uninstall、_ io _ get _ handle、_ io _ init、_ io _ set _ handle 等。有关这些 API 函数的使用将结合具体应用进行说明，下面重点分析设备驱动管理队列的组织和创建方式。

1.2　MQX 设备驱动管理队列

在 MQX 中对 I/O 设备驱动是通过队列的形式进行管理和组织的，它是输入输出子系统层的具体表现形式。各设备的私有属性信息和驱动函数注册到一个设备节点上，并将各设备节点链接成链表队列，为标准输入输出层的 fopen、fclose、read、write、ioctl 等函数提供服务。队列中的每一个节点由 IO _ DEVICE _ STRUCT 结构体表示，每一个设备被使用之前，必须先以此结构体创建一个节点，并加入到设备驱动管理队列。如图 2 所示[6-7]。

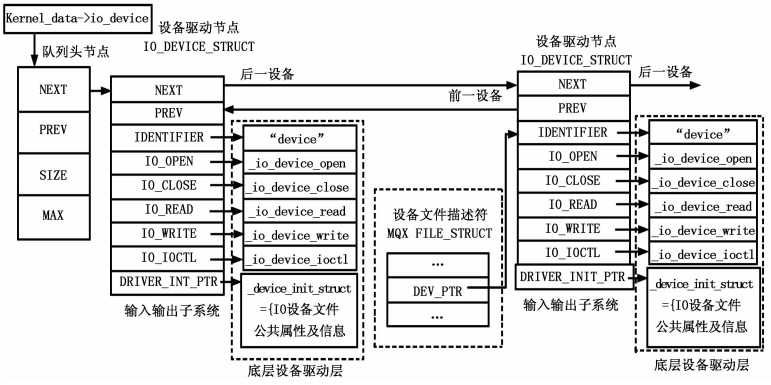

图 2　MQX I/O 设备驱动管理队列

从图 2 中可以看出，MQX 的 I/O 设备驱动管理队列由一个头节点和设备驱动成员节点组成，队头节点在 MQX 启动过程中初始化内核数据区时创建，位于内核数据区，由全局变量指针 Kernel _ data－＞io _ device 指向头节点，每次向队列添加设备或删除设备时从队头开始查找定位设备节点。在第一设备节点入队时需要初始化队列，使队头节点的 NEXT 指针指向自己。当有成员节点入队后，队头节点的 NEXT 指向第一个节点成员，而最后一个成员节点的 NEXT 也是指向自己的。利用这一特点，在队列查找时若 NEXT 指向自身，则说明已经到了最后一个节点。队列头节点的 SIZE 成员记录当前队列的节点数；MAX 成员设置队列允许的最大的节点数量，若 MAX 的值设置为 0，则队列的节点数不受限制。设备驱动成员节点由系统 API 函数 io _ dev _ install 注册设备驱动时创建，新创建的设备驱动节点加到队尾。

设备驱动节点的结构体成员 IDENTIFIER 是设备唯一标识符，可由用户在创建设备节点时设定，但必须是唯一的，不能与其他设备重名，此标识符代表了该设备，后续所有对设备的访问，都只针对此标识符进行。DRIVER _ INIT _ PTR 是指向设备初始化结构体常量指针，该结构体常量存储了设备公共参数信息，用于对设备进行访问时使用。另外，设备驱动节点的结构体成员 IO _ OPEN、IO _ READ 等是指向设备具体驱动函数的指针，例如当用户任务执行 fopen（设备标识符，参数）标准函数时，通过节点的关联，转向执行 IO _ OPEN 指针指定的具体驱动函数 _ io _ device _ open。

图 2 中的设备文件描述符，不属于设备驱动队列成员，它由 fopen（）函数打开一个设备时创建，当设备被关闭时设备文件描述符被销毁。其主要的作用通过该描述符结构体内的 DEV _ PTR 指针保存向由 fopen（）打开的设备节点。当 fopen（）函数返回时带回文件描述符的指针，该指针称为文件句柄，有了这一文件句柄，其他标准输入输出函数 fclose、read、write、ioctrl 等对设备访问，就不必要从设备驱动队列头节点一个个查找，可由文件句柄快速定位到打开的设备，这样对设备的访问就如同对一个文件的访问。

2　可复用 MQX 设备驱动四层体系结构设计

2.1　设备驱动四层体系结构模型

通过以上 MQX 设备驱动三层体系结构管理机制的分析，特别是通过设备驱动链表队列的操作，使用户对设备的访问就如同像对一个文件的访问，极大的方便用户任务对设备的操作。但这种机制下开发的设备驱动程序与 MQX 操作系统紧密结合，设备的公共参数信息的分离使用与设备驱动程序浑为一体，不便于向其他平台移植。更重要的是这种体系结构无法复用已有的经过验证成熟的设备驱动构件。例如，当一个嵌入式应用系统从无操作系统环境升级到有操作系统时，原有的设备驱动无法满足 MQX 专用调用接口的要求，仍需重新设计专门面向 MQX 系统平台的设备驱动，这将导致不必要的重复开发。

通过以上的分析，MQX 设备驱动程序无法复用，其主要根源是设备驱动程序与 MQX 操作系统紧密结合，设备的公共参数信息的提取与设备驱动程序浑为一体。因此，只要将设备驱动程序从操作系统中剥离，使底层设备驱动与操作系统无关，底层设备驱动按照嵌入式软件构件的思想进行设计，底层设备驱动构件有明确的对外入口参数和出口参数，应用程序使用底层设备驱动构件时，严格禁止通过全局变量来传递参数[8]。按照这样的规则，将底层设备驱动分为驱动接口构件层和底层设备驱动构件层，而标准应用输入输出层和输入输出子系统层保持不变，用户程序任务依旧可以调用 MQX 提供的标准函数对设备进行访问。这样设备驱动管理形成了四层体系结构，如图 3 所示。

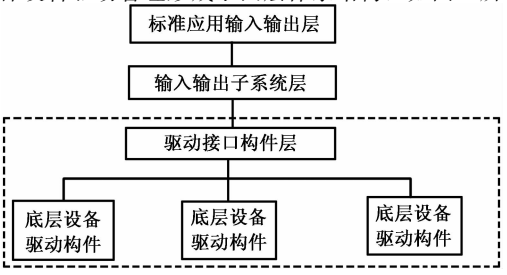

图 3　MQX 设备驱动四层管理结构

按照四层设备驱动管理体系，用户只需设计好驱动接口构件，就能复用已经成熟的底层驱动构件。同时，按照此体系开发的驱动程序也便于移植和复用，将减少重复的劳动以及有利于缩短嵌入式软件的开发周期和提高软件的开发质量。

2.2 设备驱动接口构件的设计

设备驱动接口构件承接上层传来的数据参数和命令，下达底层驱动构件，主要任务是在MQX设备驱动管理的框架下，从设备驱动节点中的设备属性数据结构体中，或从用户调用的函数传入的参数中，提取分解有效参数；并调用底层驱动构件函数，同时把这些参数传入到底层设备驱动构件中，实现接口的转接。由于设备驱动接口构件是从底层驱动程序中分离出来的，它满足操作系统对驱动函数的调用规则和对驱动节点中驱动属性信息的解析方法相同；因此，在实际的编码过程中，可以通过复制改写已有接口驱动构件的方式进行开发，以保证与操作系统完好对接。接口驱动构件具体设计过程见实例部分。

3 复用底层设备驱动构件的实例

本实例以苏大飞思卡尔嵌入式研究中心在无操作系统环境开发的串口底层驱动构件为基础，详细分析驱动接口构件的MQX底层设备驱动的复用方法。通过编写底层驱动接口构件函数，实现MQX系统与自定义的无操作系统的串口驱动构件拼接，提供一个MQX环境下复用成熟底层驱动构件的样例。

3.1 四层体系结构下设备驱动管理节点

在MQX下设备四层体系结构管理机制与三层体系结构的相同，某一设备在使用之前必须通过系统API函数io_dev_install注册到设备驱动管理队列中，图4给出了uart4在设备管理四层体系结构下的设备驱动节点，不同于三层体系结构之处是设备节点内的结构体成员指向的是驱动接口构件，而不是底层设备驱动程序，而驱动接口构件函数再调用底层驱动构件函数。

3.2 驱动接口构件设计

从图4中可以看出串口4驱动接口构件包含io_uart_open、io_uart_read等接口驱动函数和设备属性uart4_init_struct结构体。

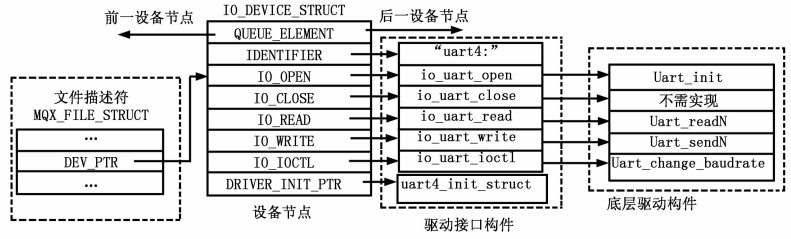

图4 UART设备驱动节点示意图

3.2.1 封装初始化数据结构体

根据UART设备的特性，将通道号、模块时钟频率、通信波特率等属性封装成UART驱动数据结构类型UART_INIT_STRUCT。加载设备驱动时，该类型的UART属性信息将被加载到MQX设备管理队列的UART设备节点上，其结构体定义如下：

```
typedef struct uart_ init_struct
{   uint_8 DEV_ID;// 设备号
    uint_32 BUS_CLK; // 模块工作频率
    uint_32 BAUD_RATE;//通信波特率
    uint_32 QUEUE_SIZE;//输入缓冲队列长度
    CHARQ_STRUCT_PTR CHARQ;//输入队列
} UART_ INIT_STRUCT, * UART_INIT_PTR;
```

3.2.2 编程接口驱动函数

实现接口驱动函数时，在函数内部解析出加载到设备节点上的属性信息，并调用底层驱动构件函数。以初始化UART设备驱动为例，设计实现_io_uart_open接口驱动构件，其代码如下：

```
//=================================================
// 函数名称:_io_uart_open
// 函数功能:初始化设备驱动函数,由 fopen 调用。
// 函数参数:fd_ptr — 设备文件句柄;
//open_name_ptr — 设备标识名,通过 fopen 传入;
// flag — 初始化属性,由用户通过 fopen 传入。
//=================================================
_mqx_int _io_uart_open(MQX_FILE_PTR fd_ptr, char_ptr open_name
_ptr, char_ptr flags);
{IO_DEVICE_STRUCT_PTR dev_ptr = fd_ptr->DEV_PTR;
UART_INIT_PTR uart_init_ptr = (UART_INIT_PTR)(dev_ptr->
DRIVER_INIT_PTR);
// 解析驱动属性信息
uint_8 dev_id = uart_init_ptr->DEV_ID;
uint_32 bus_clk = uart_init_ptr->BUS_CLK;
uint_32 baud_rate = uart_init_ptr->BAUD_RATE;
// 调用底层驱动构件
uart_init(dev_id, bus_clk, baud_rate);
return MQX_OK;
}
```

设备文件句柄fd_ptr由fopen创建并传入_io_uart_open函数。在_io_uart_open函数中，首先通过传入的设备文件句柄定位与之关联的MQX UART设备节点；然后获得对UART设备属性数据体的访问，从中解析出通道号（DEV_ID）、模块时钟频率（BUS_CLK）和通信波特率（BAUD_RATE）等属性信息，传入并调用底层的初始化函数uart_init，完成函数调用及参数传递。_io_uart_close、_io_uart_read、_io_uart_read、_io_uart_ioctl接口驱动函数的实现与此类似，解析出有效参数后传入并调用相应的底层驱动构件。

3 在任务中使用驱动服务

注册设备驱动后，就可以在任务中通过标准输入输出函数fopen、fclose、read、write进行访问。但首先要通过fopen打开设备，获取文件句柄，其余各函数才能通过此文件句柄进行操作。以下代码是应用write函数通过串口4发送字串测试任务。

```
void task_uart4 ( uint_32 initial_data)
{ MQX_FILE_PTR fd_ptr;
char uart_buf[] = "UART4 TEST";
fd_ptr = fopen("uart4:", NULL); //打开"uart4:"
if (fd_ptr ) //若打开成功,从串口4输出字串
{ write(fd_ptr, uart_buf, strlen(uart_buf)); }
fclose(fd_ptr); // 关闭"uart4:"设备
_task_block(); // 阻塞本任务
}
```

(下转第1663页)

使用 1 kHz、10 kHz、20 kHz、100 kHz 对不同的待测阻抗进行激励，最终得到表 2 的测量结果，从表中可以看出测量相对误差可以保证在 1%以内，说明设计的阻抗测量仪可靠性高，精度满足一般测量需求。

表 2　测量阻抗对比结果

通道	频率	*RC* 串联实际值		HP 4285A	本测量仪测	相对误
	(*f*/kHz)	*R*(kΩ)	*C*(pF)	测量值(kΩ)	量值(kΩ)	差(%)
1	1	5.1	4700	34.225	34.004	0.646
2	10	20	2200	21.257	21.058	0.936
3	20	51	470	53.729	53.289	0.819
4	100	100	680	100.028	99.072	0.956

在进行扫频测量时，通过 PC 配置 AD5933 扫描起始频率、扫描点数及频率参数，微处理器采集的数据通过 USB 总线实时传输至上位机，上位机将数据存储并通过阻抗－频率曲线显示出来。对 10 kΩ 纯电阻进行扫频测量（起始频率设置为 20 kHz，频率增量步长 200 Hz，扫描点数 200）绘制图 4，对由 25 kΩ 和 200 pF 电容串联组成的阻抗进行扫频测量（起始频率设置为 20 kHz，频率增量步长 500 Hz，扫描点数 80）绘制图 5。

通过图 4、5 可以看出扫频测量的结果都是围绕理论计算值上下波动的，测量相对误差小于 1%。通过上位机的频率－阻抗图可以非常直观的读出某一特定频率下的阻抗值。

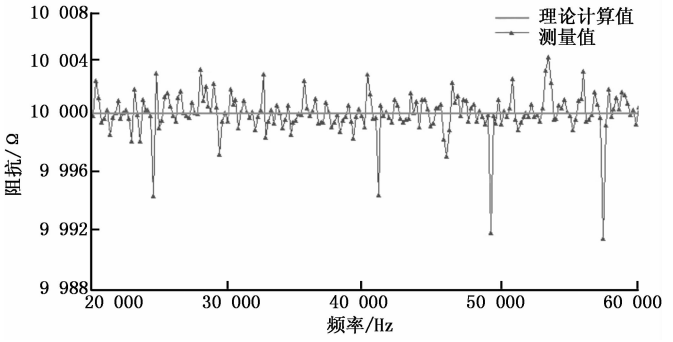

图 4　10 kΩ 纯电阻扫频测量结果

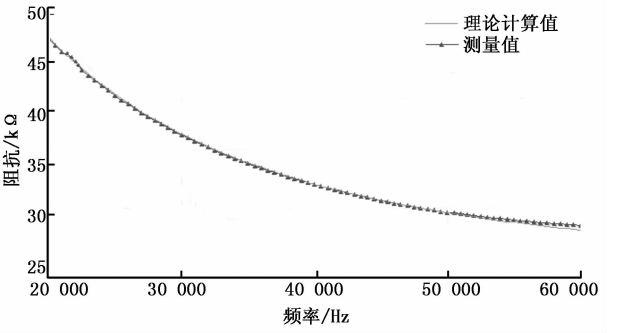

图 5　25 kΩ 电阻与 200 pF 电容串联扫频测量结果

4　结论

本文设计的基于 AD5933 的全自动多通道阻抗测量仪具有精度高，测量范围广，操作简单，可靠性高等优点，很好地克服了传统阻抗测量仪的一些缺点，给新型阻抗测量仪的设计提供了一个新的思路和较好的框架。

参考文献：

[1] 郭兴明，彭承林，唐　敏. 生物阻抗测量系统的研究 [J]. 电子技术应用，1995，(11)：13－15.

[2] 吴成雄，王　君，程　功. 基于集成芯片的细胞生理参数自动分析仪的硬件设计 [J]. 仪表技术与传感器，2009，(s)：281－231.

[3] Analog Device，Inc. AD5933 Data Sheet [Z]. 2008：1－44.

[4] 崔传金，郭志强，赵　楠，等. 用 AD5933 实现电导率测量的研究 [J]. 机电工程技术，2008，37 (4)：44－47.

[5] 王可宁，王小攀，张雄星，等，AD5933 在肉类阻抗谱测量系统中的应用 [J]. 西安工业大学学报，2012，32 (10)：806－810.

[6] 李　静，陈世利，靳世久. 基于 AD5933 的阻抗分析仪的设计和实现 [J]. 现代科学仪器，2009，(2)：28－30.

[7] 傅　元，吴　然，韩吉声. AD5933 测量水电导率电路设计中的若干问题 [J]. 仪表技术与传感器，2011，(7)：63－65.

[8] 温新华，颜小飞，安　东. 基于 AD5933 的便携式阻抗仪研制 [J]. 计算机测量与控制，2013，21 (4)：1090－1092.

(上接第 1660 页)

4　结束语

软件构件技术和软件复用技术是嵌入式软件发展的两大法宝，是解决软件危机的重要手段。文章针对 MQX 嵌入式实时操作系统难以复用现有成熟的底层驱动构件的问题，详细分析和研究了 MQX 底层设备驱动管理机制。研究结果表明其根源在于设备驱动程序与 MQX 操作系统的耦合过于紧密，设备的公共参数信息的提取与设备驱动程序浑为一体。针对此问题，提出了将设备驱动程序从操作系统中剥离的思想，使底层设备驱动与操作系统无关，设计了底层设备驱动管理的四层体系结构。并应用该体系结构成功地复用了已成熟的串口底层驱动构件，验证了该方法的可行性，为操作系统底层驱动构件的复用提供了思路。

参考文献：

[1] 郭　坚，叶志玲，陆　岚. 星载软件复用技术探讨 [J]. 计算机测量与控制，2007，15 (4)：541－543.

[2] 吕明琪，薛锦云，胡启敏. 基于软件体系结构的可复用构件模型 [J]. 计算机应用研究，2008，25 (1)：120－122.

[3] 马永杰，蒋兆远，张　燕. 基于功能构件的软件复用方法 [J]. 计算机应用与软件，2009，26 (8)：75－77.

[4] 杨芙清，梅　宏，李克勤. 软件复用与软件构件技术 [J]. 电子学报，1999，(2)：68－75.

[5] Freescale. MQX Real－Time Operating System User Guide [EB/OL]. http：//www.freescale.com/mqx，2013.

[6] Freescale. Freescale MQX I/O Drivers Users Guide [EB/OL]. http：//www.freescale.com/mqx，2013.

[7] Carlos N，Luis R. How to Develop I/O Drivers for MQX [EB/OL]. http：//www.freescale.com/mqx，2009.

[8] 王宜怀，朱仕浪，郭　芸. 嵌入式技术基础与实践（第 3 版）－ARM Cortex－M0＋ Kinetis L 系列微控制器 [M]. 北京：清华大学出版社，2013.