文章编号:1671-4598(2020)03-0163-05 DOI:10.16526/j.cnki.11—4762/tp.2020.03.034 中图分类号:TP316;TP311.1 文献标识码:A

LINUX 系统下 GPIB 驱动优化设计与实现

赵 昕[1], 郭恩全[1,2], 李小杰[2]
(1. 西安精密机械研究所, 西安 710075; 2. 陕西海泰电子有限责任公司, 西安 710075)

摘要: 为方便开发基于 LINUX 操作系统的 GPIB 仪器, 近几年, Linux 内核也集成了专用在测试测量领域里的 GPIB 总线驱动, 但直接拿来利用发现, 在向仪器发命令的频率比较高时, 此驱动传输性能不是很理想, 偶尔还会出现驱动挂掉, 造成 LINUX 内核崩溃; 针对以上问题, 给出了在 LINUX 架构下 GPIB 驱动优化设计方案; 分析了 LINUX 字符设备驱动模型; 在中断服务程序底半部里, 引入了结合睡眠机制的非原子操作工作队列, 提高了驱动运行效率; 提出了利用 FIFO 半满而非传统的全满标志位作为数据传输判断标准, 提升了数据传输速率, 引入了读写操作互斥的信号量, 消除了由于读写竞态引起的驱动异常; 对优化后的 GPIB 驱动进行测试, 结果表明, 上述问题得到了一定的改善。

关键词: LINUX; GPIB; 优化; 半满位; 工作队列; 信号量

Design and Implementation of GPIB Driver Optimal in LINUX System

Zhao Xin[1], Guo Enquan[1,2], Li Xiaojie[2]
(1. Shanxi HitechElectronic limited liability Company, Xi'an 710075, China;
2. Xi'an Precision Machinery Research Institute, Xi'an 710075, China)

Abstract: In order to facilitate the development of GPIB instruments based on LINUX operating system, in recent years, the LINUX kernel has also integrated a GPIB bus driver specially used in the field of testing and measurement. However, after being directly used, it is found that the transmission performance of this driver is not very ideal when sending commands to instruments with relatively high frequency, occasionally there will be drive hanging off, to cause kernel crash. In view of the above problems, the GPIB-driven optimization design scheme under the LINUX architecture is given. The LINUX character device driver model is analyzed; In the bottom half of the interrupt service program, a non-atomic operating work queue combined is introduced to improve the driving operation efficiency; The use of FIFO half full instead of the traditional full flag bit as the data transmission standard is proposed to improve the data transfer rate; Introducing mutually exclusive signal for read and write operations, eliminating drive-running anomalies caused by read and write race; Tests on the optimized GPIB driver, result shows that the above problem have been improved to some extent.

Keywords: LINUX; GPIB; optimize; half full; work queue; semaphore

0 引言

随着嵌入式技术快速发展, 功能更加齐全, 性能更加强大的台式仪器已经面世, 例如 LXI 仪器, 其数据传输吞吐率, 基于时间触发, 组建远程分布式测控系统都具有明显优势, 但 GPIB 经历了几十年的发展, 总线协议已经相当成熟, 仍然受到传统用户的青睐, 因此众多国内外测试测量研发商开发高端的台式仪器时, 都会保留传统的 GPIB 接口。由于 LINUX 操作系统内核是开源的, 使用者可以根据自身的需求, 对其进行裁减配置, 形成高效的嵌入式操作系统, 使得 LINUX 成为当前最流行的嵌入式操作系统之一, 但不足之处是, 它不像 WINDOWS, 有微软专门的研发团队对其进行不断优化和升级, LINUX 并非以商业盈利为目的, 没有专门组织对其进行官方维护, 从而 LINUX 内核在集成各个芯片厂商的驱动时, 不少驱动在一些应用情况下, 传输性能还不够理想, 尤其像 GPIB 这种只是测试测量领域里的专用总线, 相比应用广泛的 USB、LAN, 更具典型性。

为了提升 LINUX 系统下 GPIB 传输性能, 文章对其驱动进行了优化。将 GPIB 中断服务程序分割成顶半部和底半部, 在底半部里开辟了配置成非原子操作的工作队列, 同时采用睡眠机制, 高效的实现了驱动运行效率; 在数据接收流程里, 采用 FIFO 半满位判断标准, 使得从 FIFO 中收数据到内核缓冲区和向 FIFO 里传数据可以并发进行, 从而提升了数据传输速率; 设备文件操作中的读写函数可能会同时操作临界资源, 应用信号量避免了竞态的发生, 增强了驱动可靠性。

1 LINUX 字符设备驱动模型分析

LINUX 字符设备驱动程序[5]一般包括 3 部分: 初始化、中断服务、设备文件操作。在驱动程序初始化时, 要向系统注册此驱动程序, 系统后续才能调用驱动里各设备文件操作接口。在 Linux 系统里, 是通过调用 register _ chrdev 向系统注册设备驱动程序, 初始化部分除了注册设备驱动程序, 一般还需要给驱动程序申请系统资源, 包括内存、时钟、I/O 端口等, 芯片的初始化也在这里进行, 另外还要

收稿日期:2019-12-15; **修回日期**:2020-01-17。

作者简介:赵 昕(1983-),男,陕西西安人,硕士,工程师,主要从事测控技术和嵌入式方向的研究。

设置清除，它是让在初始化里注册的资源，全部从内核里注销掉。对于设备经常会提出请求给 CPU，来执行设备需要完成的操作，这就需要有中断服务，驱动程序通过调用 request _ irq 函数来申请中断，其原型：int request _ irq (unsigned int irq，void (* handler) (int irq，void dev _ id，struct pt _ regs * regs)，unsigned long flags，const char * device，void * dev _ id)；参数 irq 表示所要申请的硬件中断号，handler 为向系统申请的中断服务程序，中断产生时由系统来调用，调用时所带参数 irq 为中断号，dev _ id 为申请时告诉系统的设备标识，regs 为中断响应时信息寄存器地址；flag 是申请时的选项，它决定中断处理程序的一些特性，其中最重要的是影响处理速率和资源开销的中断响应方式；device 为设备名，在/proc/interrupts 文件里被记录，中断服务具体的内容由设备需要完成的操作来决定。设备文件操作是指 file _ operations 结构中的成员，它们全部是函数指针，每个函数都完成一种设备操作，如读写操作，选择操作，控制操作等。

字符设备驱动程序的开发流程是：

1）查看原理图及芯片资料理解设备的工作原理；

2）定义主设备号、注册资源、注销设备号及资源；

3）设计芯片初始化流程；

4）设计中断服务；

5）定义 file _ operations 结构，设计所要实现的文件操作；

6）编写用户态测试程序，调试设备驱动。

调试时是通过 insmod 和 rmmod 命令实现驱动的加载和卸载，加载与卸载的原理如图 1 所示。

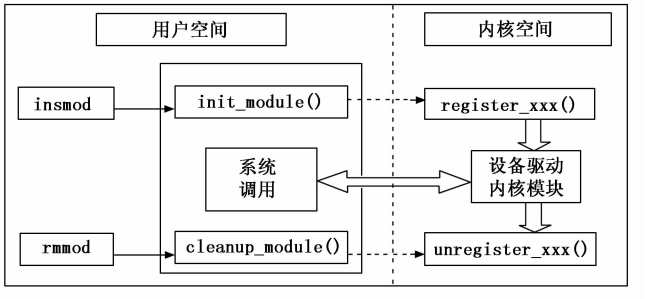

图 1 加载与卸载的原理

2 GPIB 驱动程序优化设计

2.1 TNT4882 原理

TNT4882 是美国 NI 公司的一款高速且听讲功能兼备的 GPIB（General purpose interface bus）接口专用芯片。它内部集成了 Turbo488（高速传输电路）及 NAT4882 (IEEE488.2 兼容电路)，能够兼容 ANSI IEEE Standard 488.1 和 ANSI IEEEStandard 488.2 规范，其内置还具有 16 个增强型 IEEE488 兼容收发器[6]。

TNT4882 内部寄存器[7]共 32 个，每个寄存器的控制字都是 8 位，地址通常是 TNT4882 的基地址加上各个寄存器所对应的偏移量而确定的。在 GPIB 驱动设计中，只有对寄存器进行正确设置，才能实现对 GPIB 的各种操作。工作模式可分为单芯片模式和 Turbo ＋ 9914 模式，工作模式的选择和转换，由寄存器的设置来决定。Turbo＋9914 相当于 TMS9914A 芯片的工作模式，目地是为了兼容此款芯片，但此时功能更加强大，单芯片工作模式是 NI 公司推荐的 TNT4882 工作模式，工作时的内部结构如图 2 所示。由于单芯片模式采用的是最简单又是最快速的结构，并且是推荐的 TNT4882 工作模式，因此，本驱动是在这种模式下进行设计。

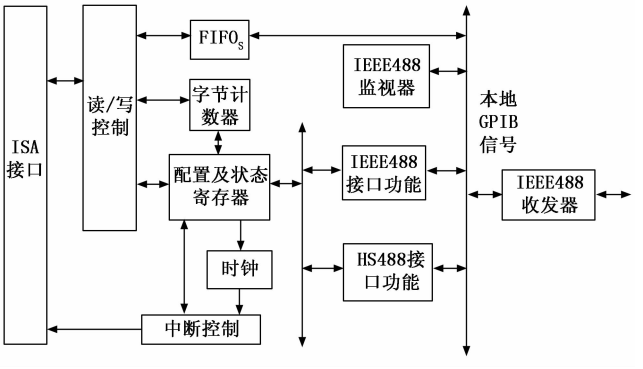

图 2 单芯片工作模式

2.2 初使化函数设计

在初使化函数里除了注册设备号以外，还要有以下方面要实现：寄存器硬地址映射到内存的虚拟地址；分别初始化读写互斥的信号量、中断服务程序底半部的工作队列、阻塞的等待队列；初始化芯片。

由于内核无法访问 TNT4882 寄存器硬地址，所以要通过映射到内存的虚拟地址，来访问 TNT4882 寄存器，映射的实现是通过调用内核函数 ioremap () 来实现，此函数第一个参数设置为 TNT4882 芯片的首地址，TNT4882 芯片的首地址是由硬件使用的片选信号线决定，函数的返回值便是首地址在内存的映射地址，由于芯片手册给出了每个寄存器的偏移量，这样 TNT4882 的首地址在内存的映射地址加上偏移量就是每个寄存器在内存的地址。

信号量的本质是一个整数值，它和一对函数联合使用，这一对函数通常称为 P 和 V，P 函数使信号量的值增加，V 函数使信号量的值减少。如果信号量的值大于零，则进程可以操作临界区的资源，反之，如果值小于零，则进程不能操作临界区的资源[8]。由于读写函数可能会操作同一资源，这种情况会产生竞态，严重时会导致系统内核的崩溃，所以需要通过信号量互斥，使用信号量之前要对其进行初使化，一般放在驱动初使化函数里，调用的内核函数为 init _ MUTEX ()；由于中断处理是独占 CPU 的，这样如果整个任务需要别的进程在中断后很短时间立刻执行，那么中断处理就不能耗时过长，所以经常把中断处理分为顶半部和底半部，顶半部只用来响应中断而底半部通过开辟个内核进程进行真正的处理，而内核进程是不独占 CPU 的，这

样别的进程也可同时执行，在底半部开辟进程常用的机制就是工作队列，而且工作队列还允许睡眠机制，使用工作队列时同样也需进行初使化，它通过调用 INIT _ WORK（）完成；阻塞是实现进程睡眠的机制，进程在运行时如果需要等待一个事件来到才能继续运行，这时就需要用阻塞来使进程睡眠，由于应用层的进程需要等待数据来到才能继续处理，所以在内核的读函数里需要添加阻塞，内核给驱动提供的最简便的阻塞，就是等待队列机制，使用前一样要进行初使化，调用的函数是 init _ waitqueue _ head（）。

TNT4882 芯片在上电运行前，要进行初使化，初使化的流程为：复位 TNT4882 芯片中的 Turbo 电路；将 TNT4882 设置成 Turbo＋7210 式；将 TNT4882 设置成单芯片模式；使 Local _ PowerOn 信号有效；设置 TNT4882 的 GPIB 主地址，屏蔽副地址以及设置 GPIB 握手参数；清除 Local _ PowerOn 信号后，开始 GPIB 操作。此过程全部是给寄存器赋值，这可通过内核 iowrite8（）函数进行。

2.3　中断服务优化设计

中断函数的流程如图 3 所示，ATN（Attention）代表注意命令，REN（Remote Enable）代表远控使能状态，LACS（Listener Active State）代表听者作用状态，TACS（Talker Active State）代表讲者作用状态。GPIB 主设备发来 ATN 命令，TNT4882 产生中断，如果 GPIB 接口处是远控使能和听者作用状态，那么 GPIB 接口就准备接收数据，如果 GPIB 接口处是远控使能和讲者作用状态，那么 GPIB 接口就可以发送数据。具体实现的方法是：利用 linux 字符设备驱动中断注册接口 request _ irq，将中断响应注册到内核里，中断号设置成硬件设计时占用中断向量表中的中断号，从参数 * device 得到的 GPIB 设备名，写入/proc/interrupts 文件里，同时将中断响应属性 flags 配置成中断底半部独立运行，用 iowrite8（）将 TNT4882 中断使能寄存器中有关的 ATN、REN、TAC、SLACS 全部使能，在中断服务程序中，用 ioread8（）读 TNT4882 中断响应信息寄存器的信息，出现与 LACS 相同的信息，调用收操作，出现与 TACS 一致的信息，调用发操作。

对于 TNT4882 芯片，收发操作首先要对收发数据状态初始化，然后循环收发数据，最后退出收发数据状态。对于具体流程，发送数据流程，如图 4 所示。由于 GPIB 是通过接口之间的握手协议来进行通信，所以接收和发送流程的初始化和退出要完全按照协议规定来设计，他们是一致的，这样在接收数据的具体流程中，只详细分析了数据接收过程，如图 5 所示。TNT4882 的 FIFO 是一个 16 字节深的缓冲区，TNT4882 不仅提供了它的全满标志位而且还提供了半满标志位，此流程中没有使用传统的全满位作为接收数据的判断位，而是采用了 FIFO 半满位进行判断，这样从 FIFO 中收数据到内核缓冲区和向 FIFO 里传数据可以同时并发进行，而如果用全满位进行判断，只有先收数据后，有了空间才能向 FIFO 里传数据，所以采用半满位可以提高数据接收速率。

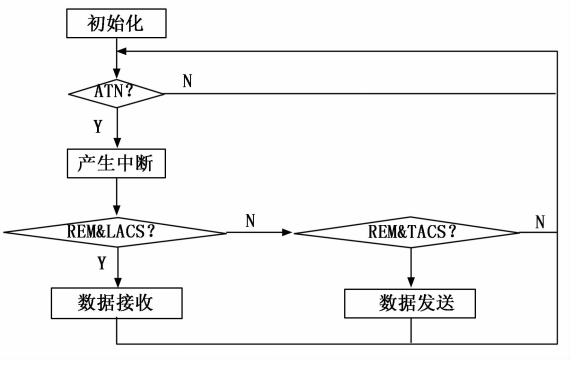

图 3　中断函数的流程

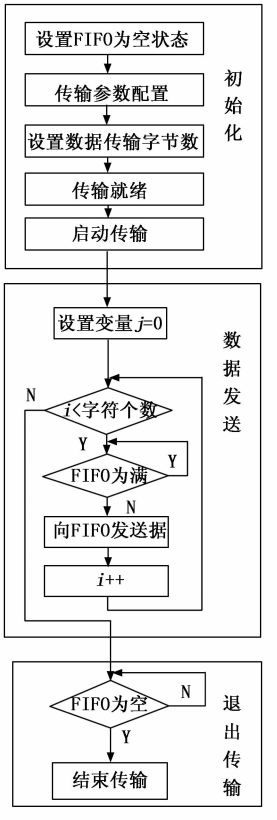

图 4　数据发送流程

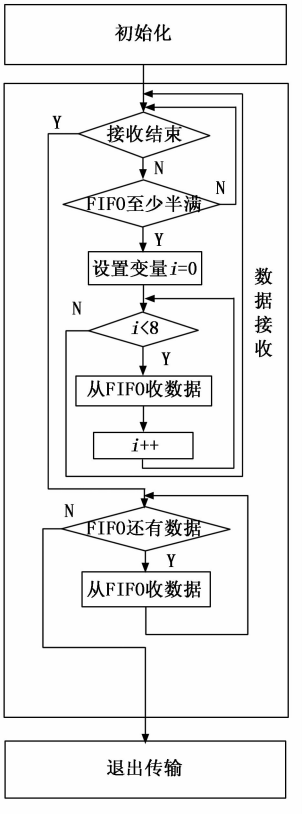

图 5　数据接收流程

在中断函数里，数据收发操作是通过调用自定义的两个函数：short Receive（void），short Send（void）来实现的。对于收是直接调用收函数，而对于发则是间接调用，通过调用工作队列来调用发函数，如果直接调用数据发送函数，那么根据中断服务程序运行是独占 CPU 的原理，导致用户态的应用程序只能等中断服务程序结束才能执行，这样数据还没有过来，就先执行数据发送，反复轮询，这显然是低效的，而调用工作队列，等于内核开辟了一个进程，便释放了 CPU，并配置成非原子操作模式[9]，这样内核态的工作队列和用户态的进程可同时运行，当在工作队列开始处再加上睡眠，数据到达内核后，唤醒工作队列，此时再调用发函数，数据进行发送，此时运行效率明显

提升。

数据收发操作具体实现的方法是：在数据发操作方面，用 creat _ workque 创建工作队列，将该创建的工作队列赋给 queue _ work 函数第一个参数，将 Send（void）函数指针赋给 queue _ work 函数第二个参数，从而将 Send（void）函数插入工作队列，并将 workqueue _ struct 类型的结构体变量中的属性成员配置成非原子操作，中断服务程序运行到 queue _ work 接口处，内核将自动开辟进程执行 Send（void），在开辟进程开始处，调入 wait _ event _ interruptible（）使 Send（void）睡眠，在文件操作写函数结束处，调入 wake _ up _ interruptible（）来唤醒该睡眠；对于 Send（void）本身，用 memset 函数将 TNT4882 的 FIFO 清空，并用 iowrite8（）在传输寄存器里配置传输参数和传输字节数，配置完成后，再用 ioread8（）读取传输就绪位，等待传输就绪，最后使能传输位，完成初始化；设置传输字符计数变量，读取 FIFO 状态寄存器，判断是否 FIFO 为满，不为满时往里面循环写数据，直到计数变量到达此次发送字符数，退出发送操作，注意发操作结束后，要利用的 destroy _ workque 注销工作队列。在数据收操作方面，初始化与发操作一致，不做赘述，用 ioread8（）读取中断使能寄存器的听状态位，如果是听状态，并且 FIFO 至少半满，设置接收计数变量，开始从 FIFO 中用循环取数据，直至取 FIFO 半满的数据，然后再判断中断使能寄存器的听状态位，如果仍是听状态，重复以上操作，如果听状态结束，FIFO 中仍有数据，用 ioread8（）将数据取回，退出收操作。

2.4 设备文件操作读写优化设计

GPIB 驱动里的读写函数即 read 和 write 分别实现用户态从内核收数据和用户态向内核发数据，实际上就是在 read 和 write 里分别调用内核的 copy _ to _ user 和 copy _ from _ user 来实现的。根据在初使化里的分析，读写要进行互斥，采用的方式是信号量。具体实现的思路是：设信号量为驱动全局变量并调用内核函数 DECLARE MUTEX（）使其初始值为 1，在读写函数里的开始处调用 down（）使信号量减 1，结束处调用 up（）使信号量加 1。这样如果写进程操作了临界资源，信号量的值变为零，读进程则无法操作临界资源，当写进程结束后，信号量的值恢复为 1，此时读进程可以操作临界资源，信号量的值又变为零，写进程则无法操作临界资源，当读进程结束后，信号量的值又恢复为 1，此时写进程又可再次操作临界资源，这样就实现了读写互斥，消除了竞态。

除此之外，由于内核的读函数是给应用程序调用的函数接口，为了实现应用程序在数据没有到来的时候能够被挂起，内核的读函数还要实现阻塞，具体实现的思路是：在读函数里的开始处可以通过调用 wait _ event _ interruptible（）来使读函数进入睡眠状态，wait _ event _ interruptible（）要放到 down（）前面，否则会产生死锁，在 GPIB 接口处听者作用状态结束时，通过 wake _ up _ interruptible（）来唤醒睡眠状态，这样就实现了读阻塞。

3 测试验证

将仪器与计算机通过 GPIB 电缆相连，将 mknod 建立设备结点命令和 insmod 向内核插入驱动命令的参数均设置为优化后的 GPIB 驱动模块，然后将以上命令编写成脚本，在 LINUX 终端下运行此脚本，GPIB 驱动便加载到 LINUX 内核，并运行测试程序，因为在 GPIB 驱动的 read 函数里用了阻塞机制，所以在数据没有到来之前，程序在阻塞状态。从图 6 可以看到 GPIB 设备已打开，驱动被成功加载到内核。

```
[root@192 /]# bash q
Using ./gpib.ko
gpib_irq=17
ioConfig=55AA
GPIB BASE_ADDR: C4994000
InterruptConfig : 00004040
InterruptConfig : 00004000
globalvar register success.
[root@192 /]# cd tmp
[root@192 tmp]# ./t
gpib open.
gpib read.
```

图 6 优化后的 GPIB 驱动被成功加载到内核

接着在计算机上运行 GPIB 管理器软件，*IDN?[10] 命令随软件启动自动发送，枚举当前在线仪器，图 7 展示了仪器的信息成功返回；最后再对 GPIB 驱动进行命令快速连续发送测试，如图 8 所示。

图 7 仪器信息成功返回

图 8 命令快速连续发送测试

从以上测试步骤可以看到，优化后的 GPIB 驱动加载到 LINUX 内核，系统运行正常，说明与系统兼容性良好，利用 GPIB 管理器软件对在线 GPIB 仪器进行枚举，设备被正常识别，说明驱动通信正常，以上两方面的测试，反映了优化后的 GPIB 驱动并没有影响驱动原有的正常运行和功能，最后再针对向仪器发命令的频率比较高时，GPIB 驱动传输性能不是很理想的问题，进行命令快速连续发送测试，所有命令返回值均立即并正确返回，GPIB 管理器软件并无警告或异常报错提示，这表明优化后的 GPIB 驱动，使此问题得到了有效的改善。

4　结束语

本文主要从中断服务程序和设备文件操作读写函数对 GPIB 驱动进行了优化设计。在中断服务程序底半部里，利用非原子操作工作队列，实现了内核态的进程和用户态的进程并行，并结合了睡眠机制，从而提高了驱动运行效率；受并行工作思想的启发，提出了 FIFO 半满位作为接收数据流程中的判断标准，实现了 FIFO 中收数据到内核缓冲区和向 FIFO 里传数据可以并发进行，对加快数据传输速率起到了较好的效果；在读写函数里，应用了信号量，避免了读写可能同时操作临界资源的隐患，对驱动的可靠运行进行了加固。在对驱动运行效率，传输速率，可靠性几方面的优化后，GPIB 驱动传输性能得到了一定的提升。

参考文献：

[1] 魏永明，沈　岳. LINUX 设备驱动程序设计 [M]. 北京：电子工业出版社，2016.

[2] 孙　琼. 嵌入式 Linux 应用程序开发详解 [M]. 北京：人民邮电出版社，2006.

[3] 胡华伟. 智能数据采集仪总体方案 [Z]. 西安：陕西海泰电子，2017.

[4] TNT4882TMProgrammer Reference Manualv [S]. National Instruments Corporation，1995.

[5] 李　桥. 嵌入式 LINUX 设备驱动程序的开发研究 [J]. 计算机与数字工程，2009，37 (2)：87－89.

[6] 罗光坤，张令弥，王　彤. 基于 GPIB 接口的仪器与计算机之间的通讯 [J]. 仪器仪表学报，2016，27 (6)：635.

[7] 李建华. 数据接口总线 GPIB 及其应用 [J]. 中国测试技术，2014，30 (6)：63－66.

[8] 舒克毅，胡荣强. 嵌入式 Linux 字符设备驱动程序设计 [J]. 仪表技术，2010 (2)：4－5.

[9] 杜　俊. 嵌入式 Linux 字符设备驱动程序设计研究 [J]. 甘肃科技，2015，31 (18)：28－31.

[10] 井　涛，郭永瑞. 一种实用的 SCPI 语法分析设计方法 [J]. 国外电子测量技术，2006，25 (2)：42－44.

(上接第 162 页)

同时，通过极值对比可以看到，RTX 驱动的闭环时间相对稳定，波动保持在 0.003 ms 之内；Windows 驱动的闭环时间相对不稳定，波动在 0.008 ms 之内。

本次实验表明，无论在响应时间方面，还是在稳定性方面，RTX 驱动的性能都处在领先地位，对于追求实时、稳定的环境而言，RTX 驱动无疑是首选。

由于系统设计出发点的不同，无论是在线程调度算法、线程优先级定义、定时器精度方面，Windows 系统均不是 RTX 对手。因此，Windows 驱动的落败也是在预料之中的。

这也表明，RTX 可以对一个单一的低成本的平台进行扩展，使其满足一个广泛的嵌入式应用程序的要求。之所以很多高校和研究所广泛使用 RTX，确实是有一定依据的。

5　结束语

本文介绍了 PCI－1751 接口卡在 RTX 实时系统下驱动程序的编写方法，出色的实现了板卡提供的 DI/O 功能、中断采集功能，可以满足绝大多数工业、生产、仿真的实时性要求。同时对于其他类型的接口卡，亦可借鉴本文中列举的方法和框架进行开发驱动。对于 PCI－1751 板卡的定时器/计数器等功能，由于篇幅所限未能介绍，感兴趣的读者可以参考本文的思路，自己探索尝试。

参考文献：

[1] 吴成富，成海朋，陈怀民. PCI 智能多路串口卡的设计与实现 [J]. 现代制造工程，2010，11，99－102.

[2] 李宏宇，李茂月，刘献礼. 基于 RTX 系统的 PCI 硬件设备驱动程序开发 [J]. 制造技术与机床，2019，679 (1)：185－189.

[3] 王少虎，刘亚斌. 基于 Win7＋RTX2012 的 LVDS 遥测采集卡驱动程序开发 [J]. 电子设计工程，2018，26 (9)：10－14.

[4] 段雨昕，赵　斌，周　敏. 外部硬时钟在半实物仿真中的应用 [J]. 现代防御技术，2017，45 (6)：185－190.

[5] 陈丽平. 在 RTX 环境下的 PCI 板卡驱动方法研究 [J]. 数字技术与应用，2016 (7)：94.

[6] 张沛露，张宇鹏. 基于 RTX 系统 PCI 总线驱动程序设计实例 [J]. 经贸实践，2015 (8)：344－345.

[7] 胡　浩. 基于 RTX 实时系统 ARINC429 总线通信驱动开发 [J]. 计算机测量与控制，2015，23 (1)：310－312.

[8] 赵常寿，孙明月，樊　蓉. RTX 实时系统下 PCI－5565 反射内存卡驱动程序设计 [J]. 电脑编程技巧与维护，2014 (11)：5－18.

[9] 李俊贤，李红宇. 基于 RTX 的实时通用测控软件设计与实现 [J]. 微电子学与计算机，2016，33 (10)：120－124.

[10] 金相男. 基于 Ardence RTX 的 1553B 驱动程序开发 [J]. 电子设计工程，2014，22 (2)：149－151.

[11] 高振伟，王保勇. 基于 windows 设备驱动程序的开发与优化 [J]. 科学与财富，2015 (4)：155－155.

[12] 金　鹏. Windows 平台下 PCI9054 的驱动程序的研究 [J]. 网友世界·云教育，2013 (22)：3－3.

[13] 龚　俊，张　京，王　璐. 基于 WDM 的 CPCI 多串口数据通信卡驱动程序设计 [J]. 软件导刊，2015，14 (8)：115－117.