



## USB接口芯片D12 开发的经验

前段时间接手了课题组的一个开发 USB 接口的项目。当时几乎是白手起家的状况，除了老板给我 PDIUSB12 的一些英文的 PDF 文档和一些源代码之外，就几乎没有其他任何资料。经过几个月的摸索，可以说已经基本上对 USB 开发的框架有了比较深入的了解，而且可以初步应用到实际的设备中。其中可以说走了不少弯路。现在想开发 USB 的网友越来越多，我也不妨把这几个月以来自己开发中的经历告诉大家，也许少走些弯路。也许我的一些想法也是很很不成熟的，让内行见笑了。

我个人觉得，要想搞 USB 的话起码得有以下几方面的知识：计算机硬件技术基础，单片机原理以及 Windows 程序设计。当然你的设备不一定非得用 Phil 的 PDIUSB12 接口芯片，还有很多可以供选择的接口芯片。大侠们可以举例，呵呵。不过既然导师给我 D12 芯片的资料，所以我一开始也不得不选择它，现在看来 D12 芯片还是比较好用的，不过最大的缺点就是市场上太难买到了。我的这片也是老板通过朋友才搞到的，所以到现在我还一直舍不得用，唉.....

刚开始的时候大概看了一下 USB2.0 的英文规范，不过实在是太痛苦了。不过现在网上到处可以下 USB1.1 的中文规范，而且不少“开发指南”的书就是 USB1.1 规范的翻译，可以拿来看看，了解一下 USB 到底是什么东西，有些什么特点。作为应用，我觉得了解一下就可以了，至于里面那些电气特性和 Hub 的规范，先不看也罢。还有其中一些细节的东西，也不用看得太仔细，有了点实践经验之后再回头看看也许效果会更好。不过最好还是了解一点 USB 设备的“配置”，“接口”，“端点”，“管道”的一些概念。因为以后的软件方面的开发和这些概念的是密不可分的。由于我现在弄的还是比较简单的单功能设备，一般来说都是一个配置，一个接口，多个端点或管道的设备。所以我对“配置”和“接口”的概念还不是特别理解，但我觉得对于入门来说，理解 EndPoint，Pipe 这两个概念还是很有必要的。起码你得知道 Usb 支持哪几种类型的端点，管道，它们都有什么特点。而你的设备的端点又是什么类型的。

照我的理解，USB 其实就是一种外设与计算机通讯的协议。而且现在的大部分的机器都支持 USB 设备，具体协议是怎么电气化的实现，我们可以根本不用知道。不管是 PC 端还是外设，都可以用专用的芯片来实现 USB 传输的电气特性和其中主要的协议。PDIUSB12 就是一种这样的芯片，当然这种芯片提供的仅仅是接口的功能，而另外有一些系列的芯片如 Intel 系列的本身就是一个微处理器。相比之下，D12 芯片需要用单片机（通常就是 51 系列）和它配合工作才能起作用，这种情况下你甚至可以把它当作 8255A，8279 那样的芯片，单片机和它们的通讯方式都是类似，只不过作为 USB 的接口芯片，D12 的控制要复杂的多。当然，这时你最好得对 C51 语言有一定的了解，毕竟用汇编来实现 8051 的 USB 传输实在是不敢想象。另外，要知道你的 51 单片机还要做很多其他的事情的，Usb 只是实现与 P



C 通讯的手段而已。像我接手的项目就是做个高速 AD 采集板，这才是主要方面。你不可能做一个除了 51 单片机和 PDIUSB12 就什么都有的设备吧？^\_^Phil 的 PDF 上提供了一些 D12 芯片应用的例子的资料，其中包括它的原理图。看看原理图再看看它的功能实现的一些源代码是比较有帮助的。虽然它的例子代码一都比较长，但是针对 D12 芯片的特点，程序的大部分都不需要做太多的改动，就可以适应于不同的功能的外设。

从例子程序中理解设备的工作原理是比较清晰的 就我看到的程序来说 那是一个跟 MFC 程序差不多长的一段 C51 代码的程序，但是它编译之后出来的代码的大小还是可以接受的。至少用 At89C52 的内部 8Kflash 还是绰绰有余的。

如果大家用的也是 D12 芯片，我们不妨一起来分析一下它所附带的一些代码程序。几个主要的文件是不可少的：

hal.c 文件，定义实现 C51 与外部存储器/IO 端口的读写操作的函数。直接对外的硬件操作。

USB12.c 文件，用于实现向 PDIUSB12 发送特定的命令字的函数，这些定义都是为了方便以后对芯片控制时的调用方便。比如对于 8255A 芯片，我们就可以定义一个 SetMode 函数，这样我们在以后的程序就可以很方便的调用这个函数来设置 8255A 于特定的工作状态。

MAINLOOP 文件和 ISR 文件，可以理解这两个文件就是整个设备工作的主程序的所在，PDIUSB12 与 51 单片机之间是工作在一种中断的方式状态下，D12 通过中断向 51 发送各种请求。Mainloop 文件里面是整个程序的 Main 函数的描述，其中 Main 函数作为程序设备 Reset 时的程序入口，调用了一些初始化设备的函数，比如各种寄存器如中断寄存器，定时器，计数器等，初始化 D12 芯片并完成连接等工作，然后程序进入循环等待阶段，等待着中断的发生。而 Isr 文件定义的就是与中断处理相关的一些函数。因为 D12 从硬件连线上来说只有一个中断请求管脚，我们可以把它与 51 的 Int0 脚或 Int1 脚相连，并为之定义中断服务函数，但是 D12 的中断是有好几种类型的，我们可以在中断发生之后调用读 D12 的中断寄存器的命令来判断是什么类型的中断，比如是端点 1 的写入中断，就可以调用相应的子函数处理，并在处理结束前清除中断寄存器中的标志以等待下一次中断。ISR 程序和 Mainloop 程序之间可以通过一些全局变量来进行通讯。

另外还有一个奇怪的文件叫做 Chap9.c，刚开始搞了半天不知道它的作用是什么，后来回去仔细看了看 1.1 规范的第九章，原来是规范中规定必须实现的一些与 PC 通讯时的标准的请求响应函数。比如说 SetAddress，SetConfiguration 等函数，另外还有一个文件叫 Vdor.c，翻译成卖主自定义的请求，不过这里买主这个概念定义得比较晦涩。但是简单点说就是，这



个设备是你设计的，卖主就是你。呵呵整个程序的框架就是这样。而照我自己理解，系统的整个工作原理应该是这样：当系统连接到 PC 的 Usb 端口上时，PC 的 Usb 控制器采用默认的地址 0 和你这个新接入系统的设备进行通讯（大部分工作应该都是 USB 的总线驱动干的）。我想设备干的第一件事情应该是 Reset 吧（也可能不是），然后 Pc 向 D12 的端点 0（默认的控制端点，无需进行特殊的配置）发送一些标准请求，也就是前面提到的 Chap9 里定义的一些标准的设备请求。比如查询设备的状态，设备的描述符，设置设备的状态，设置设备的地址等。这些请求应该是由系统的总线驱动发送到 D12，然后 51 响应 D12 的中断，判断检测到是端点的输入中断，然后再根据请求的一些参数来调用 Chap9 里定义的函数来实现这些主机的请求，比如设置地址。因为地址 0 是 Usb 设备默认的地址，所有新接入的 Usb 设备都通过这个地址和主机进行通讯，所以你不能占用这个地址太久，必须迅速的给你分配一个合适的地址，以后主机就通过这个地址和你进行通讯，地址 0 就让出来给新的设备。这时 51 检测到 D12 的中断之后调用 SetAddress 函数为设备设置地址，地址的参数当然是从主机传过来的了。完成初始化系统的配置之后，主机就可以和设备提供的各个端点进行通讯了，也可以实现一些“卖主”定义的特殊的请求。比如，让 51 外接一个小发光二极管，可以定义几个自己的请求，比如点亮小灯 LedOn，熄灭 LedOff 和闪动小灯 LedFlash，这些函数实现起来应该很简单，就可以把它们加入你的 Vendor Request 函数集中，以后可以在驱动中进行调用。虽然简单，但是当你看到通过你的 Windows 中的一个小程序可以控制设备的小灯一亮一灭闪闪发光的，是不是很有成就感啊？呵呵，虽然只是你的一小步，却是人类的一大步呀，hiahia。

BTW：C51 的编译环境推荐使用 Keil C51 uVision 6.20 版的完全解密版。没有解密的版本用起来有很多的不便。而且会有写 bug，当然这个版本据说也是有一些 bug，但是至少现在觉得还是比较好用的。

说了那么多，其实设备端的 Firmware 的设计相比于整个“工程”来说只是很小的一部分，真正的困难的地方是开发 Windows 端的驱动。在我自己的经历中，研究驱动的开发这段过程几乎占据了 80% 的时间，可以说现在也在研究，以后还得继续研究。有点庆幸自己大一的时候 C 语言学得还可以，现在的整个工程几乎没有那个环节和 c 语言没有关系的。从设备端的 Firmware 开发，设备驱动的开发和驱动程序的编写，没有一样离得开 C 语言的。当然，开发驱动所用到的 C 语言环境有和以前所接触到的 C 语言的环境有很大的不一样，也许语法结构是一样的。但是以前很多标准的有用的 C 库函数就不能随便乱用了。

刚开始的时候，包括我和我的老板都有那样的想法，Windows 本身已经有了 Usb 总线的驱动程序，所以在硬件设计完毕之后只要再设计 Pc 端的应用程序就可以和设备进行通讯了。就连 Phil 的资料也是直接就给了几个 CreateFile，ReadFile 和 WriteFile，说这样就可以和设备进行通讯了。其实不然，总线驱动固然可以和你的设备直接通讯，但是不同的 Usb 设备还是有自己不同的特点的，这些特点总线驱动不可能都服务到的，总线驱动只是完成一些最



基本的工作,根据设备的特点来控制对设备的读写等操作还是必须得由设备的客户驱动来完成,不过客户驱动也只是管理对总线驱动的调用就可以了。对于 WDM 驱动开发,书籍和网上的资料也不少,我也不可能完全的按照我自己的理解都说出来。不过我还是可以把我学习过程中的一些感受跟大家说说,刚开始看的书是我们组里的一本《Windows WDM 设备驱动开发指南》,Chris Cant 编的,那本书看得我好费劲呀,感觉他写得太笼统了,如果对 Windows 的内核环境以前从来没有过什么了解的话很多概念理解起来比较困难。尤其是 USB 开发的那章。后来换了一本 Microsoft Press 的《WDM 设备驱动开发》Walter oney 写的那本,那本书不知道现在有没有翻译好的发行。我在 DriverDevelop 下载了一本站长翻译的电子版的。感觉这本比较适合入门,讲得比较详细,像设备接口,中断请求级(IRQL)以及内核同步问题,Chris Cant 那本书对这些概念问题几乎避而不谈,而这些概念我都是看了 Walter oney 的书之后才比较理解,尽管看电子版的书比较痛苦,而且也付出了视力下降的代价。但是 Walter oney 的书中的例子不太好理解,因为他自己也做一个 Generic 的封装,移植性不是很好,而且书中的大部分程序都需要合适的外设才能调试。而 Chris Cant 的例子很值得好好研究(个人觉得),首先他的程序设计完全是利用 DDK,除了借助 VC 的 IDE 开发环境之外,而且 DebugPrint 的调试方式更有利于理解驱动程序的原理。设想一下,你可以打开 DebugPrint 的 Monitor,然后把你的设备插入 PC 的 Usb 插槽,你就可以很清楚的看见整个驱动的工作过程,Pnp 管理器的工作过程。

刚开始接手这个工作的时候,有个师兄吓唬我说:写驱动啊?很恐怖的,调试的时候必须用两台电脑拿电缆连接起来才行。幸亏我没被吓倒,不过写驱动的确是很头痛的事情,调试驱动则是更恐怖的事情。你会有机会经常看见 win2000 的难得一见的死亡蓝屏,呵呵。不过幸运的是,写 Usb 驱动相比于其他驱动程序来说要简单一下,因为我们所需要的 Usb 设备的客户驱动几乎不用直接和设备的硬件打交道,很多事情都可以交给总线驱动来完成,就像前面所说的一些标准的请求。我们所要做的事情就是响应应用程序的 IO 请求,稍加处理后将请求发送给总线驱动,我们这里所要做的请求不是很复杂,比如管理读写的数据量,管理读写对应的管道等。还有 Io 控制请求,比如我们可以像设备发送一个 Vendor Request,让小灯一灭一闪。这些请求都是通过 Usb 的总线驱动来完成,具体怎么完成我们就不用管太多了。一般来说,只要你循规蹈矩的,不要做什么“越轨”的事情,死亡蓝屏很少出现的,顶多是输入输出的数据不太对罢了。建议使用 Win2000 作为开发平台,个人感觉 Win2000 的 Pnp 管理似乎更加完善。开发的时候可以省去很多麻烦,至于和 98 兼容的问题可以以后在慢慢考虑,应该不是主要问题。

Visual Studio 和 2000DDK 是必不可少的开发工具,另外 Platform SDK 也应该装上,里面有很多有用的工具可以用的上的,比如 Guiden,我就经常得用到。另外作为调试,DriverStudio 也是必须的。

前段时间邮购了一套 DriverDevelop 开发的 USB 开发板,感觉还是很好使的。虽然暂时





只有 LedOn, LedOff 和 LedFlash 几个功能,但是用于测试你的程序还是比较方便的。而且我觉得有了这么一个硬件环境来实践你的程序的确是相当难得的,我就是因为没有一个好的硬件环境使得我的工作搁浅了很长的一段时间。驱动开发网上有一篇文章是“10 分钟开发一个 Usb 驱动”,完全照着那个驱动开发一个 DriverWorks 的驱动程序就可以对这个设备进行控制了。整个工作是相当方便的,至于应用程序的编写,相比之下就是小 case 了。呵呵。另外用 DriverWorks 你还可以很方便的测试 Usb 的其他几种类型的传输比如 Bulk, interrupt 和 Iso,但是为了加深对 WDM 驱动的理解,我最后还是用 DDK 重写了它的驱动。虽然费了点事,但最后还是获得了成功。就 Chris Cant 的那本书的章节而言,我觉得只要理解了前 9 章的内容(包括驱动的框架,驱动的初始化,IRP 分发例程,PNP 管理)和后面关于中断控制,Usb 传输的内容之后就可以设计一个基本的 Usb 客户驱动了。其他方面以后可以慢慢加深,也不是说没有必要。

关于驱动程序的调试,正如前面说的 DebugPrint 是很不错的工具,另外,DriverStudio 中的 SoftIce 还是必不可少的,毕竟可以在单机上进行源代码级的程序调试还是很爽的事情。不过 SoftIce 的界面不是很友好哦,有时候它莫名其妙的弹出来会给你一种恐怖感。另外你就不能一边调驱动一边听音乐了。呵呵。

对于 SoftIce 不是很熟,目前也就会加载程序,设断点而已。程序的代码加载的过程不是很复杂,用 SoftIce symbol Loader 打开你的驱动(\*.sys 文件),然后生成 SoftIce 需要的\*.nms 符号文件,打开\*.nms 文件并载入模块,就可以在 SoftIce 中调试你的驱动了,比如你可以在 Read 的分发例程中设一个断点,调用你的应用程序读你的设备,你就会看见 SoftIce 的窗口弹了出来,停在你设的端点上了。是不是很爽?不过我以前犯过一个比较愚蠢的错误,就是用 Free 版本的模块去调试,结果断点永远不起作用。呵呵,记得得用 Checked 版本的驱动。

关于驱动的安装和 Inf 文件的编写,可以参考那两本书,说得还是比较详细的。

我所做的工作目前也就是这个地步,虽说已经迈出了“人类历史的一大步”,但是后面还是很多“步”要走,所以,继续吧。

在这里还得谢谢许多网友在这段时间给我的帮助,让我能迅速的得到提高。希望我的文章也能对初接触 USB 的朋友有所帮助。也希望大侠能提供一些更好的建议。