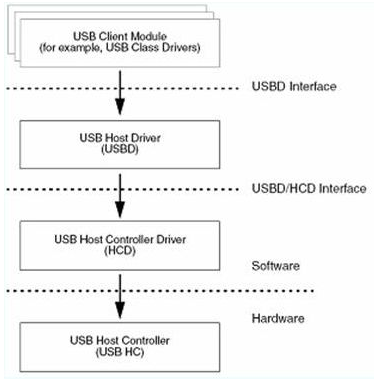
Vxworks中USB驱动栈-1

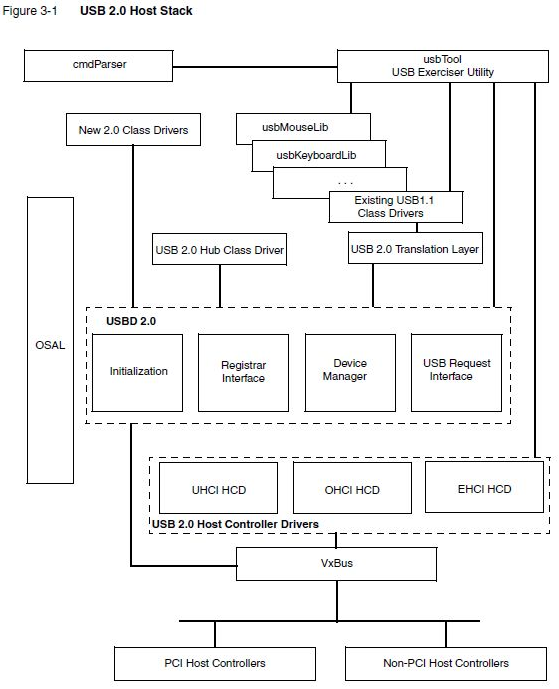
USB模块可以分为多端口主机（MPH）模块和双角色（DR）模块，它们都能够连接一个或者二个外部端口，这些模块和外部端口总称为USB接口.Mpc8379的USB模块采用的是DR，它的寄存器和**[数据结构](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/juana1/article/details/_blank)**均基于Intel的EHCI(Enhanced Host Controller Interface Specification for Universal Serial Bus)，DR模块可以充当USB总线上的主机、外设、以及支持便携式On-The-**[Go](http://lib.csdn.net/base/go" \o "Go知识库" \t "http://blog.csdn.net/juana1/article/details/_blank)**（OTG)）可协商主机/外设。DR模块有三个基本操作模式：主机(Host)、设备和OTG。可以把DR模块配置成使用UTMI、ULPI 或者FS/LS串行收发器中的任何一种PHY接口。由于引脚的限制，UTMI接口只能用在设备操作模式。值得注意的是，设备模式下的DR模块不支持LS操作。

风河USB为通用串行总线提供USB传输初始化(USB hosts)和允许vxWorks目标作为USB外设的支持，USB hosts(又叫USB host stack)和USB外设(又叫USB peripheral stack)都遵循USB 2.0规约。Host Stack使vxWorks可以使用USB设备，而Peripheral Stack允许Windows机器将vxWorks板当做一个USB设备。首先介绍Host，下图为USB Host驱动栈结构：



USB Host Stack驱动包括USB驱动(USBD)、主机控制驱动(HCD)、hub驱动和class驱动。风河为标准接口协议EHCI、UHCI、OHCI提供 HCD驱动，另外，还为USB控制器提供了root hub diver，为各种USB外设提供class driver合集。其中，USBD是硬件独立的，它提供了驱动栈上层(包括USB class drivers)与USB总线的通讯通道，还负责电源管理、USB带宽管理、动态挂载/释放USB设备等功能。

下图为vxWorks中所体现的USB Host Stack组成图：



这里从上往下来理解，usbTool是风河提供的USB**[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库" \t "http://blog.csdn.net/juana1/article/details/_blank)**工具包，从使用usbTool的过程可知，USB在初始化时的步骤如下：

1. 先执行usbInit初始化USB2的Host Stack，源码如下：

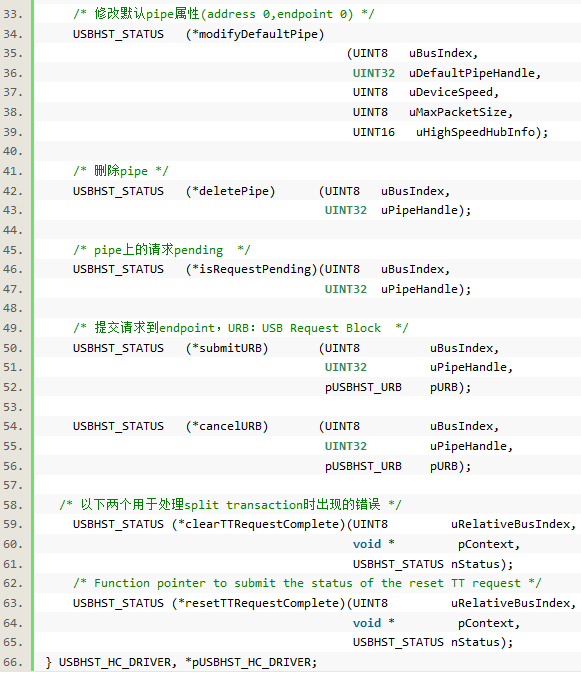


2、然后执行usbdInitialize初始化USBD，该函数在系统调用其他USBD函数前必须执行一次以上，有一个公用计数器guUSBDInited ，Initialize函数时会加1，ShutDown时会减1，大于0时表示初识化完成，注意：由于usbdInitialize可以嵌套执行，所以在执行计数器操作时，必须要遵循互斥访问规则。它用于准备访问URBs所需的USBD和传送单元。oss为O/S-independent services，用于保护互斥信号量。Host Stack的USBD2.0和Hub class模块的源码可以参考这两个目录：

installDir**/target/src/hwif/usb**和installDir**/target/src/hwif/busCtlr/usb/hub**

1. 再初始化EHCI、OHCI、UHCI控制 器，在usbTool中，这个过程叫做Attach，用于初始化hcd并注册到vxBus。先后调用usbxhcdInit和vxbUsbxhciRegister两个函数，前者的使用过程实际上就是调用usbHstHCDRegister，将HCD注册到USBD中，这里涉及 到一个结构体USBHST\_HC\_DRIVER，它包含了HCD的函数指针，在HCD初始化时会将它传给USBD，后者利用这些指针和HCD进行通信。该结构体定义如下：





后者则分析意义不大，全部都是调用vxbDevRegister注册到vxBus上。

至于这些驱动，风河已经编译好了，见目录installDir**/target/src/hwif/busCtrl/usb/hcd**，三种接口都分开放置了，尽管一次只能使用一种接口，但风河允许同时添加多个接口驱动，从而加强了产品的兼容性。到这里需要注意，添加usbTool就不能添加其他任何Init函数，否则会编译出错。

若不是有usbTool，将组件完全包含后，系统若要使用USBD2.0接口，需要在vxWorks启动时经历4步初始化过程：

1、根据选择的组件将USB host控制器注册到vxBus。该过程是vxBus发现控制器设备并执行特定的vxBus初始化操作，调用函数为vxbUsbControllerRegister，

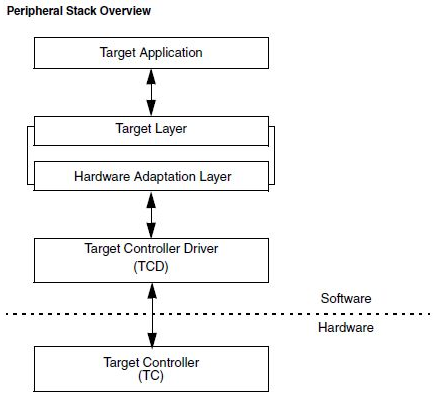
2、执行USBD入口函数usbdInit，

3、执行usbHubInit初始化hub class Drivers；

4、执行usbHcdInit将特定的HCD注册到USBD。

    上图中还有个OSAL，该组件用于为vxWorks关于host Stack的**[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "http://blog.csdn.net/juana1/article/details/_blank)**服务提供一个抽象、简化的视图。它包含了进程管理、互斥量、内存调度以及系统时间等，在usbd中，就是ossLib库。需要注意，在编写USB驱动时，需要有大部分精力是放在互斥访问和内存调度上的。

介绍完了Host，再来看下Peripheral驱动栈，下图为Peripheral驱动栈的结构图：



风河USB Peripheral驱动栈中，位于底层的是目标控制器TC，它是Peripheral栈中用于连接USB的硬件部分。对于每种类型的TC，都会有对应的TCD，风河提供了Freescale Dual Role、NetChip NET2280、PDIUSBD12和PhilipsIsp1582四种TCD，它们的功能主要包括：

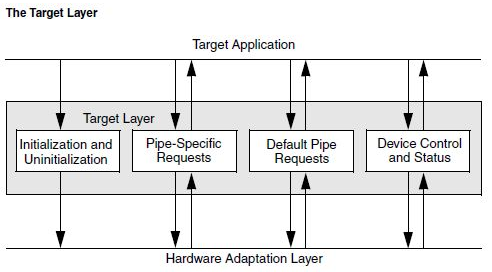
1、实现任何硬件相关的功能；

2、实现寄存器访问，USB Peripheral栈的其它层都不允许实现寄存器访问；

3、为与栈中上层通讯提供entry point。

在这之上就又是HAL(Hardware Adaptation Layer)，该层为驱动栈中的更上层提供了硬件独立的访问方式，使得整个驱动栈更容易移植到新的TC硬件上。target layer的功能与之类似，也是一个抽象的中介物。在运行时，目标应用程序会命令目标层Attach一个TCD，之后目标层就负责TCD与目标应用程序间 的请求及回应，它可以同时处理多个TCD与应用程序的通讯。所以，这部分的重点就分布在目标层和TCD上。

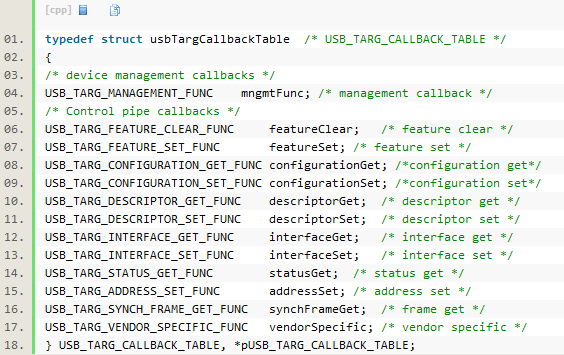
先看目标层，下图显示了目标层是如何串联应用层和HAL的，并描述了目标层的内部组成：



要通过该层实现通讯，初始化代码和应用程序需要经过以下几步：

1、初始化目标层：和USB Host栈类似，这里有一个初始化代码usbTargInitialize，主要功能也是初始化OS库、创建句柄和互斥访问量，同样的嵌套式调用，所以需要至少调用一次。

2、实现必须的 callback函数：目标层有一个callback表，列举了应用程序中所有的功能入口函数。一但TCD与之attach成功，目标层会通过这些入口执行异步回调。所以在第3步开始前，应用程序会根据表中的入口函数将对应的功能函数指针与之对应。回调表的原型定义在usbTargLib.h中，定义如下：



3、Attach一个TCD：在目标程序能从host收发指令之前，初始化代码还必须将自己和TCD attach起来，使用函数usbTargTcdAttach，该函数原型如下：

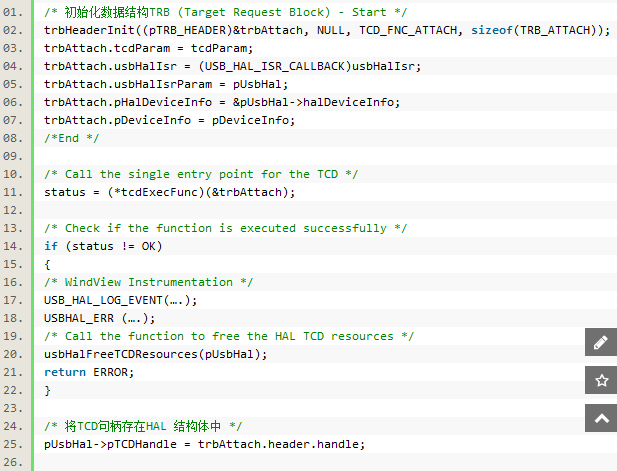
usbTargTcdAttach (USB\_TCD\_EXEC\_FUNC tcdExecFunc, pVOID tcdParam,

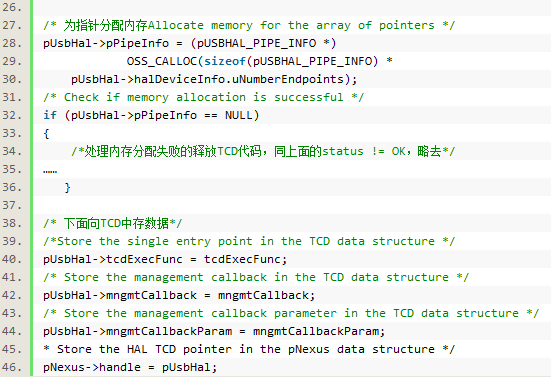
                                pUSB\_TARG\_CALLBACK\_TABLE pCallbacks,

                                pVOID callbackParam,

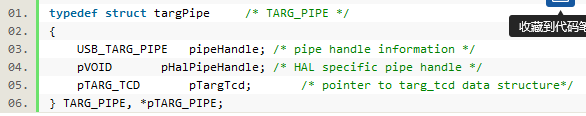
                                pUSB\_TARG\_CHANNEL pTargChannel);

需要传入的参数包括TCD的Single Entry Point指针，TCD-defined属性值，目标应用程序callback表的指针以及callback函数的参数。当目标控制器TC成功的 attach到TCD后，TCD会返回自己的句柄，保存在USB\_TARG\_CHANNEL中，应用程序可以通过该句柄与TCD进行接下来的通讯。在 HAL层，函数会调用usbHalTcdAttach真正与TCD连接上，该函数主要源码如下：





1. 使能该TCD：在attach成功后，应用程 序会调用usbTargEnable使能TCD，和第三步的attach类似，此时TCD也会使能底层的目标控制器，HAL会通过 TCD\_FNC\_ENABLE执行TCD的single entry point(usbHalTcdEnable)。
2. 创建管道pipes：上层应用程序会调用函数**usbTargPipeCreate**创建管道，管道的信息存储在结构体TARG\_PIPE中，定义如下：



pipeHandle只是管道的标示符，用于应用程序执行USB传输到终端；pHalPipeHandle是HAL信息的指针，用于在目标层中的内部记录，只要管道创建成功，指向结构体**USB\_HAL\_PIPE\_INFO**的指针就会被存在该句柄中；

1. 传输数据：数据的传输有两种形式，通用的和控制的。前者使用目标层函数**usbTargTransfer**实现USB外围驱动栈与主机间的数据传输；而如果应用程序要通过默认控制管道与主机通信，即为后者，需调用**usbTargControlResponseSend**、**usbTargControlStatusSend**和**usbTargControlPayloadRcv**三个函数。**usbTargTransfer**函数通过pipeHandle初始化一个管道上的传输，需要传输的数据用结构体USB\_ERP描述，结构体定义如下：



至于后者，目标应用程序用**usbTargControlResponseSend**发送控制管道回应主机用**usbTargControlResponseSend**发出的请求，应用程序在处理主机发送的各种Control-IN请求时都会利用之前发送的数据。例如，一个GET\_DESCRIPTOR请求，应用程序会调用API发送回应数据来响应主机的请求。**usbTargControlStatusSend**用于当控制传输没有数据段时向主机发送状态；**usbTargControlPayloadRcv**用于注册一个回调函数接收主机发来的控制管道回应，接收到回应后，回调函数就会被调用，pBfr指向控制数据。

简单来说，Vxworks设备驱动程序是硬件设备和Vxworks操作系统进行通信的一系列的方法，目前风河公司的vxworks6.x支持两种设备驱动模型。

(1):VxBus-Enabled Device Driver

从VxWorks 6.x之后，该模型被加入到VxWorks系统中来，风河公司最近提供的所有设备驱动都是采用VxBus模型的。该模型为驱动程序定义一些标准的接口来与操作系统和硬件设备。值得一提的是，如果驱动程序是在SMP环境下工作的话，风河公司建议驱动程序采用VxBus模型，风河公司并不保证传统的驱动程序模型在SMP(对称多处理器结构)环境下是安全的，这样就必须要求开发者在驱动程序里面考虑到SMP因素。

1. :Legacy Device Driver

这种传统的驱动模型都是在VxWorks6.x之前，比如5.5一直被采用的。传统的驱动模型并不采用统一的接口提供给操作系统或者硬件设备。在VxWorks6.6之后，风河公司只推荐在单处理器环境下采用传统驱动模型。但是无论怎样，风河公司还是建议开发者采用VxBus模型开发最新的驱动程序。

关于VxBus

术语VxBus通常情况下指的是VxWorks中对支持设备驱动程序的一些具体的基础设施，它包括：

（1）允许设备驱动程序自动与设备匹配

（2）为其他软件环境（设备驱动程序之外）访问设备功能提供机制

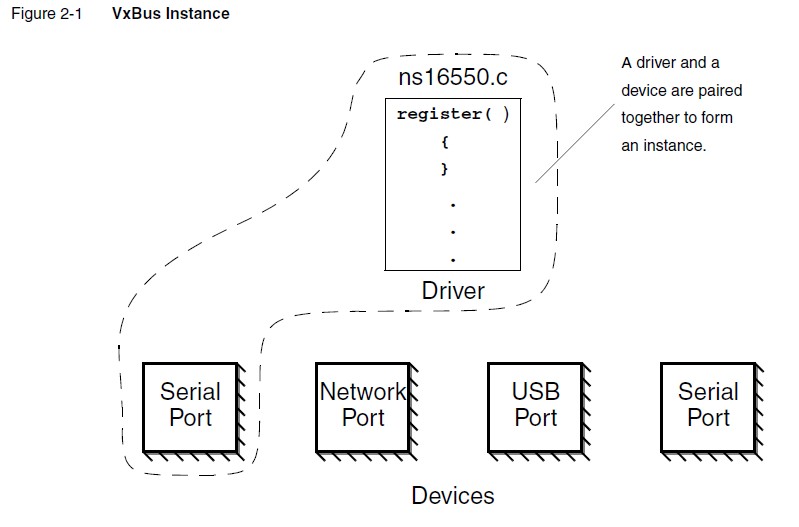
（3）为VxWorks系统中的设备驱动程序的多样性提供一些必须的支持

（4）另外VxBus有时候也指VxWorks系统的的组件集合，包括WorkBench开发平台vxprj命令行工具，VxWorks镜像项目等

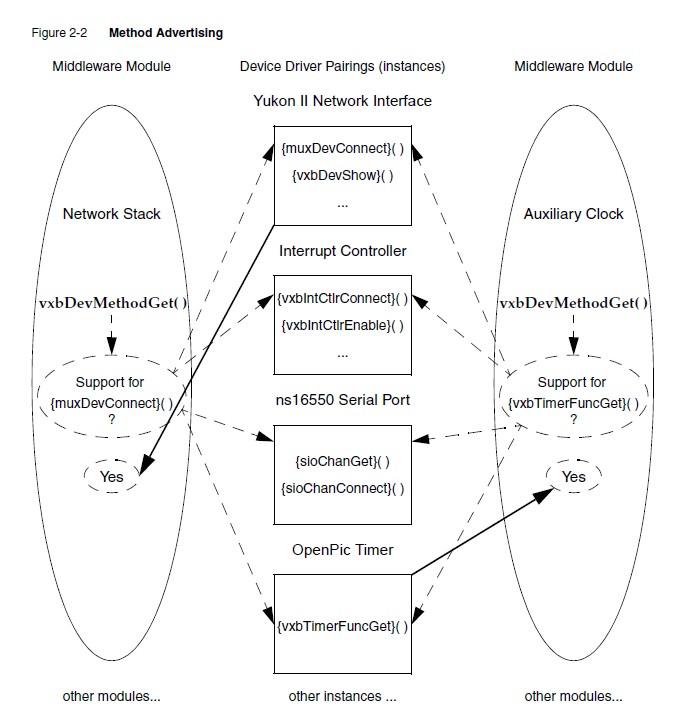
VxBus最核心的功能是组件功能，它把每个设备驱动程序和VxBus支持的模块都抽象成一 个组件，所有的这些组件都可以单独在Workbench中进行配置（添加和删除）。在VxWorks6.2以前，设备驱动程序并没有集成vxWorks项 目配置中，程序员为了添加和删除对特定设备的支持必须具备足够的BSP、驱动程序开发知识，当设备驱动程序被添加或者删除时，它同样也要求额外的功夫去管 理VxWorks工程。作为组件的集合，VxBus通过允许在workBench中选择非常多的驱动程序和支持模块来减少上面提到的大部分工作量，而且它 并不要求程序员具备良好的BSP和驱动程序知识，也不要求增删设备驱动时对工程的附加管理等。

## 3、VxBus Device Driver

对于很好的理解VxBus 设备驱动程序，有三个概念非常重要：设备，驱动程序，实例。设备概念大家都很清楚，一般就是指硬件；驱动程序是指使硬件设备能够被**[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "http://blog.csdn.net/htyang725/article/details/_blank)**访问的可执行代码和必须的配置信息；每一个驱动程序可以和0或者多个设备相关联，术语实例指的是这种关联的其中一个。这和蓝牙设备配对过程是一样的，驱动程序也要和设备进行配对，系统可以同时存在多个这样的实例。以下是VxBus实例的示意图：

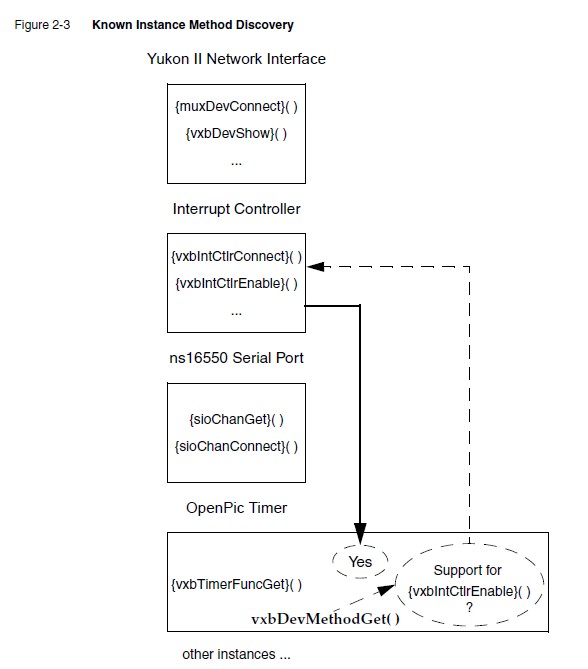


如上图所见，一个设备驱动程序和一个设备配对后形成一个实例。设备驱动程序中的方法构成了一种机制，它为软件其他部分访问硬件设备功能提供支持。当使用驱动程序的一个方法时，发起请求的模块能够询问一个或者多个实例。这种询问可以查询一些如何完成一个动作的信息，也可以请求驱动程序去完成某一个动作。对于上层来说，这些查询主要包括指定实例能否支持这样一个动作，哪些实例能够支持这样一个动作。 下图演示了vxworks系统中的一个子系统的device/driver/OS的通信过程。



上图主要包括网络栈和辅助时钟两个中间模块，它们都尝试与系统中的硬件进行通信。注意到，实际的系统中，一般会有多个实例和很多中间模块，上图只是实际系统的一个子集而已。

一个实例通过广播它支持的驱动方法使得这些方法都能够被整个VxWorks系统访问。在上图 中，网络栈使用vxbDevMethodGet( )方法来查询系统中的每一个实例，在这个例子中，网络栈寻找支持{muxDevConnect}( )设备方法的实例，如果该实例支持该驱动方法，则会返回驱动中实现该方法的函数的指针，如果不支持，则返回NULL。此例中，网络栈找到一个支持该方法的 网络接口（Yukon II Network Interface）。上图同样演示了辅助时钟的询问过程，辅助时钟寻找支持{vxbTimerFuncGet}( )驱动方法的实例，并且得知系统中的OpenPic timer instance支持该方法。上图中的虚线表示查询系统中的实例，实线表示得到了一个positive结果（支持）。



## 2.1 串行设备驱动程序

串行设备驱动程序管理面向终端和使用串行接口（RS232，RS422)的设备，这些 设备都被连接到VxWorks的IO系统，并可以在控制台中进行配置，软件可以使用open,read,write,ioctl,close等标准接口来 访问这些设备，在VxBus的框架下，串行设备驱动程序的安装目录在：

installDir/vxworks-6.x/target/src/hwif/sio

## 2.8 USB驱动程序

USB功能被划分为两个不同类型：USB HOST适配器和USB Class Drivers。USB HOST适配器是一种总线控制器，其实就是一个桥，它把USB总线通过USB HOST适配器挂接到PLB或者PCI总线。在VxBus框架下，USB HOST适配器的驱动程序安装目录是：

installDir/vxworks-6.x/target/src/hwif/busCtlr/usb/hcd

USB Class Drivers主要包括USB Storage，USB network等，但是在Vx6.8中USB Class Drivers并没有被集成到VxBus的框架下，因此，它们的主要源码目录在：

installDir/vxworks-6.x/target/src/drv/usb

## 2.9 中断控制器驱动程序

中断控制器的主要任务就是管理中断源，当外部设备向中断控制器请求中断后，中断控制器会在适合时把中断传递给CPU，让CPU处理该中断请求。在VxBus框架下，中断控制器驱动程序的安装目录是在：

installDir/vxworks-6.x/target/src/hwif/intCtlr

中断控制器负责识别哪些设备连接到了中断输入引脚，配置中断输入属性，禁止或者使能中断等。

注意VxWorks支持I/O重定向的功能，我们可以使用函数：ioGlobalStdSet(stdFd, newFd )来将系统保留的描述符stdFd（0,1,2）重定向到newFd上，这样做的好处是可以将标准输入/输出/错误输出重新定向到任何一个你喜欢的I/O 设备上，如：串口，socket，文件等等，便于你跟踪调试。自然也可以使用ioGlobalStdGet（stdFd）读出现在系统的标准输入/输出 /错误输出定向在哪个描述符上。更灵活的是我们可以使用ioTaskStdSet(tasked,stdFd,newFd)重定向某个指定任务的标准I /O.

2. Pipe Devices

管道是利用VxWorks的I/O系统进行任务间通讯的虚拟设备。任务写消息到管道上，这些消息可以由其它任务读出。

我们可以使用pipeDevCreate（）函数来创建管道，并给定最大消息数，及每个消息的最大长度。VxWorks的管道被设计成与任务级代码一样的方式允许ISRs写管道，许多VxWorks的代码，包括除了管道外的其他I/O设备都不允许ISRs写 。ISRs通过调用write()函数来写管道，但是一旦管道满了，由于ISRs无法阻塞，消息将被丢弃。

驱动程序的工作流程

不同设备在操作系统中完成的工作是不同的，但是就是工作流程来说，大致可以分为两个阶段。 第一个阶段是初始化阶段，在初始化阶段，驱动程序主要完成硬件以及设备驱动相关数据结构的初始化。 第二个阶段是硬件的访问阶段，根据设备工作模式的不同，可以分为中断模式和轮询模式，无论何种模式都可以通过与硬件设备无关的通用接口进行硬件设备的访问。

USB主驱动USBD(USB host driver，简称USBD)和HCD之间的接口允许一个或超过一个的底层主控制器。而且，WindRiver的USBD能够同时连接多个USB HCD。这样的设计特点可以使开发者建立复杂的USB系统。USBD是在HCD之上的与硬件独立的模块。USBD管理每一个与主机相连的USB设备， 向更高层次提供了可与USB设备通信的路径。它还负责自动处理USB电力管理以及USB带宽管理。而且，USBD还管理USB hub，Hub功能是一个驱动能否对USB正确操作的评价之一。因此WindRiver的USBD设计者要使USBD透明地处理hub的功能。这意味着，USBD 还能处理USB hub和设备的动态插拔。

USB Client模块在USB主驱动栈的顶端。USB类驱动(USB Class Driver)是Client模块的典型例子。USB类驱动负责管理连接到USB上的不同类型的设备； 它们依靠USBD来提供与每个设备的通信路径。USB client模块的其他例子就是那些利用USBD与USB设备通信的应用程序。

4、 USBD驱动详解

这一部分将要描述USBD(USB Host Driver)的典型应用。例如初始化，client注册，动态连接注册，设备配置，数据传输，同时还探讨了USBD内部设计的关键特性。这部分是VxWorks下USB驱动的核心。

4.1 初始化USBD：分为两步

1. 必须至少调用一次函数usbdInitialize()。(通常在Vxworks系统启动时已经调用过了该函数，在其他的设备模块中是不是不需要再调用该函数？)在一个给定的系统中，usbdlnifialize()初始化内部USBD数据结构，并依次调用 其它USB驱动栈模块的入口。usbdinitialize()可以在启动时调用一次，也可以对每一个设备各调用一次。USBD 自己记录了调用usbdInitialize()(?+?)和usbdShutDown()(?-?)的次数。只有大于等于1时才是真正初始化了，而等于 0是关闭了。
2. 用USBD 的~~lisbdHedAttaeh()~~（没有找到该函数，但是存在usbdDynamicAttachregister函数和usbdHcdAttach函数）函数来把至少一个HCD连接到USBD上。这一过程既可以在VxWorks启动时，也可以在运行时把HCD 连接到USBD 上去。后一种机制可以支持“热插拔”，而不用象前一种那样需要重新启动。

4.2 HCD的连接(attaching)与断开(detaching)

当 HCD连接到USBD 时，调用者为usbdHcdattach函数传递HCD执行入口(表HCD\_EXEC\_FUNC)和HCD连接参数(HCD attach parameter)。USBD用HCD FNC ATYACH 服务请求依次激活HCD的执行入口，传递同样的HCD attach参数。

需要强调虽然可以改变用HCD定义的参数，但是最好不应该有所改变。对于WindRiver提供的UHCI和OHCI的HCD，HCI attach参数是一个指向结构PCI\_CFG\_HEADER (定义在pciConstants.h) 的指针。该结构用UHCI和OHCI主控制器的PCI配置头来初始化，而HCD用这个结构中的信息来定位，管理特定的主控制器。典型的，调用者用 usbPciClassFind ()和usbPciConfigHeaderGet()来得到想要的主控制器的PCI配置头- 这两个函数定义在usbPciLib.h 中。如果有UHCI或OHCI要连接到USBD，就要调用这些函数来获得每一个主控制器的 PCI\_CFG\_HEADER。然后利用usbdHcdAttaeh来激活已鉴别出的每一个主控制器。注意：底层BSP可能不支持USB的HCD断开，因为当中断向量表重新使能时，如果还应用的是过期的向量表，会导致错误。

4.3 启动顺序

必须在所有USBD函数前执行函数usbdInitialize()。存在以下两种调用方式：

(1)传统的“启动”初始化。执行顺序与其意义如下：

a．usbdInitialize()；

b．usbdPciClassFind()：定位一个USB主控制器；

c．usbdPeiConfigHeaderGet()：读USB主控制器配置头；

d．usbdHedAttaeh()：连接HCD，将其作为特定的主控制器：

e．调用USB class driver初始化入口点；

f．USB class driver调用usbdlnitialize()。

(2)“热插拔”调用。执行顺序与其意义如下：

Boot Code里调用：

a．USB class driver初始化入口点；

b．USB class driver调用usbdlnitialize()；

Hot-Swap code调用：

c．Hot-Swap 鉴别USB主控制器的连接或断开；

d．Usbdlnitialize()；

e．UsbdPciConfigHeaderGet()：读USB主控制器配置头；

f．UsbdHcdAttach()：连接HCD，将其作为特定的主控制器。

因为热插拔可以在任何时刻发生，所以USBD和其Client都必须被写成可以动态识别USB设备被插入还是被拔出。当主控制器连接到系统时，USBD 自动地鉴别与其相连的设备，并通知相关的client；同样，拔出设备时，也要通知相关设备。重要的是，USBD 的client，比如USB class driver，在client初始化时，从不设想特定的设备已经出现；而在其他时候，这些驱动随时检查设备是否已经连接到系统上。

4.4 总线任务

对每一个连接到USBD 的主控制器，例如插入或拔出设备，USBD都会产生一个总线任务，来监控总线事件。一般情况下，这些任务是休眠的(不消耗CPU)，只有当USB hub报告它的一个端口有变化时，它们才被唤醒。每一个USBD总线任务有VxWorks任务名：UsbdBus。 虽然HCD委托USBD来管理，但有可能HCD 亲自监视主控制器事件。例如WindRiver提供了UHCI和OHCI的HCD来创造这样的任务。对于WindRiver的UHCI模块 (usbHcdUheiLib)，后台任务只是被周期地唤醒，目的是为了检查超时IRP(用一个中断来通知OHCI根hub发生改变)。用以在USBD和USB之问进行通信的client模块，除了调用usbdlnitialize()外，必须调用usbClientRegister() 使其在USBD注册。当一个client注册到USBD时，USBD把每一个以后将要用到的client的数据结构定位，并跟踪那个client的请求。对于每一个client，在client注册过程中，USBD还创建了一个callback任务。在成功注册client后，USBD返回一个句柄 USBD\_CLIENT\_HANDLE。以下对USBD的调用，将会用到这个句柄。当所有句柄都不需要时，可以调用 usbdClientUnregister()来释放每一个client的数据结构和callback任务。注意：此时所有client要求的任务都会被 取消。例如：注册一个叫USBD\_TEST的client，再注销。

注册：usbdClientRegister(USBD\_TEST，&usbdClientHandle)；

注销：usbdClientUnregister(usbdClientHandle)；

4.5 client回调(callback)任务

USB 操作是严格遵守时序的。例如为使中断传输和同步传输正确工作．需要依靠时钟中断。在一个有几个不同client出现的主系统中．总是有可能出现一个 client打断其它client传输事件的发生。WindRiver USBD建议用client callback任务来解决这个问题。许多USB事件可以导致一个USB client的callback任务。例如， 每当USBD 完成USB IRP后，client的IRP callback函数被激活。同样，当USBD识别出一个动态连接事件后，会激活一个或更多的动态attach callback操作。但不是马上激活这些回调操作， 而是安排合适的相应的USBD client的回调任务来执行callback。

一 般的情况下，每一个client的callback任务处于“休眠”态(阻塞态)。每一个client的callback，继承了 usbdClientRegister()产生的VxWorks任务优先级。这确保了每一个callback按其client的任务优先级来执行，而且可 以利用优先级来写client，保证对时间要求严格的USB传输。由于每一个client有它自己的callback任务，因此在callback期间， 它们有很大的灵活性决定可以做什么。例如，允许在不破坏USBD或其它USBD client性能的条件下，使callback执行代码运行至阻塞态。