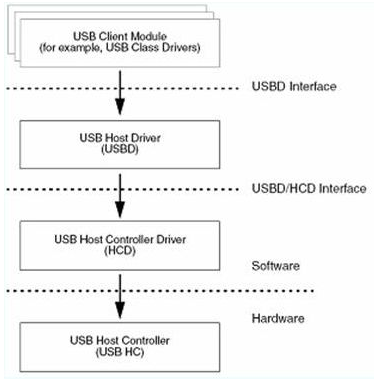
Vxworks中USB驱动栈-1

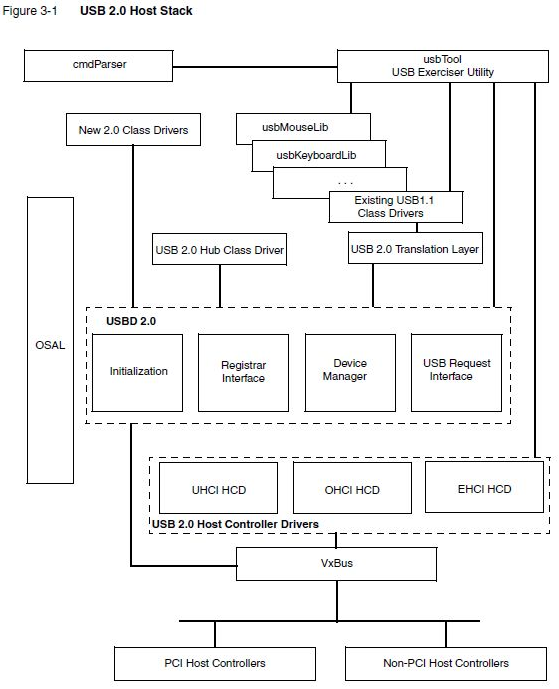
USB模块可以分为多端口主机（MPH）模块和双角色（DR）模块，它们都能够连接一个或者二个外部端口，这些模块和外部端口总称为USB接口.Mpc8379的USB模块采用的是DR，它的寄存器和**[数据结构](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/juana1/article/details/_blank)**均基于Intel的EHCI(Enhanced Host Controller Interface Specification for Universal Serial Bus)，DR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：

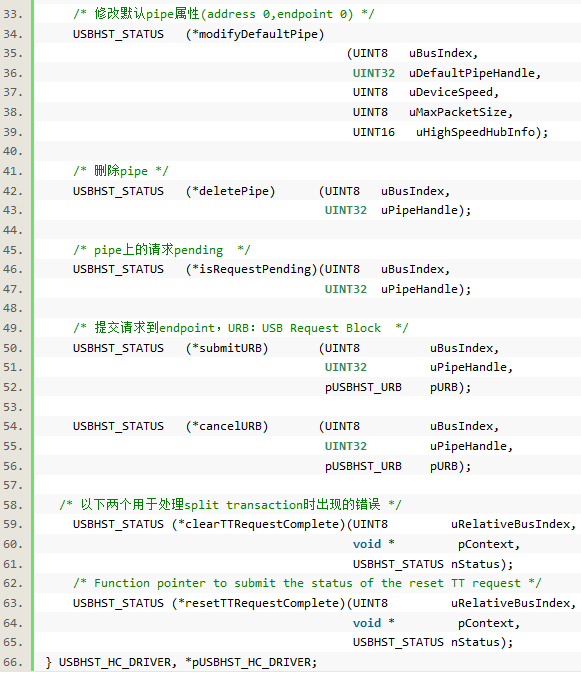


2、然后执行usbdInitialize初始化USBD，该函数在系统调用其他USBD函数前必须执行一次以上，有一个公用计数器guUSBDInited ，Initialize函数时会加1，ShutDown时会减1，大于0时表示初识化完成，注意：由于usbdInitialize可以嵌套执行，所以在执行计数器操作时，必须要遵循互斥访问规则。它用于准备访问URBs所需的USBD和传送单元。oss为O/S-independent services，用于保护互斥信号量。Host Stack的USBD2.0和Hub class模块的源码可以参考这两个目录：

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，

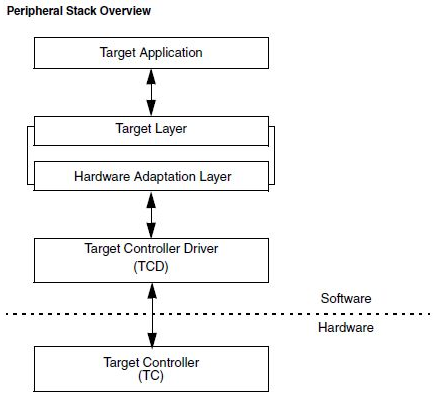
2、执行USBD入口函数usbdInit，

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：

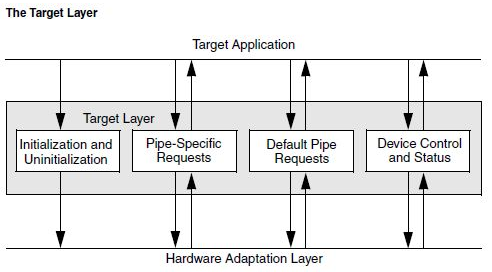
1、实现任何硬件相关的功能；

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。

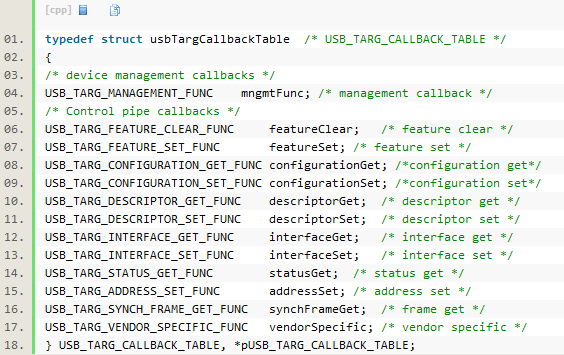
先看目标层，下图显示了目标层是如何串联应用层和HAL的，并描述了目标层的内部组成：



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：

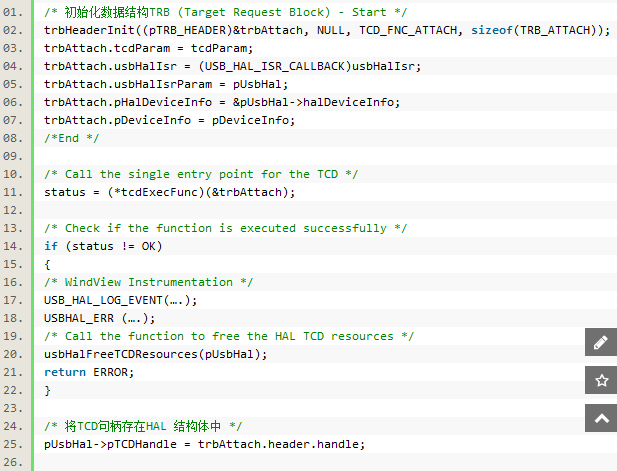
usbTargTcdAttach (USB\_TCD\_EXEC\_FUNC tcdExecFunc, pVOID tcdParam,

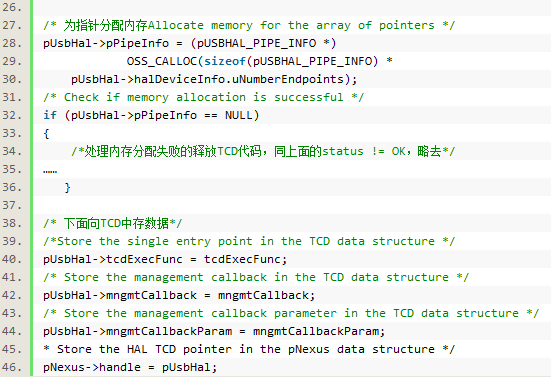
                                pUSB\_TARG\_CALLBACK\_TABLE pCallbacks,

                                pVOID callbackParam,

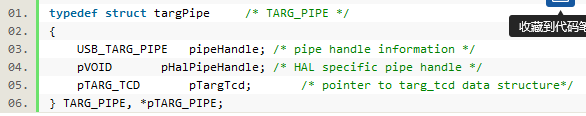
                                pUSB\_TARG\_CHANNEL pTargChannel);

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。