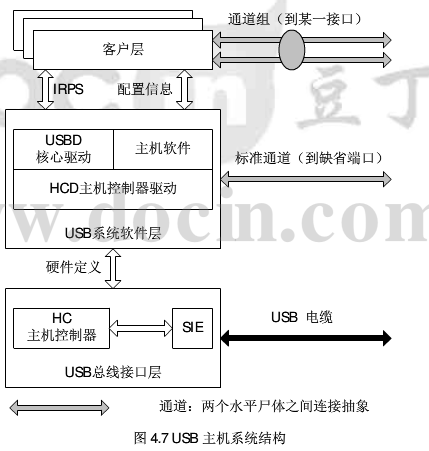
[机载数据采集器电路设计及驱动开发 - 豆丁网](http://www.docin.com/p-959635237-f7.html) <http://www.docin.com/p-959635237-f7.html>

一般来说USB仅仅是数据传输的接口，其本身不对数据的内容进行解析，数据的解析和应答由USB控制器后面的系统软件（驱动）来完成，系统软件从USB控制器FIFO读取请求和数据进行解释和响应。USB外设的各种功能的实现由主机上称为USB Client的驱动和外设中内嵌的系统软件（固件）配合完成。



从上图可以看出USB系统中主机3个主要的组成部分的关系，USB用户层是基于USB软件系统层次的USB主机组成部分，包括USB设备驱动程序（实现设备类协议）和用户软件。USB系统层是由操作系统提供的USB软件，用于控制主机控制器来管理主机和设备之间的数据传输，系统层包括了三个部分的软件，即USB主机控制器驱动程序（HCD）、USB核心驱动程序（USBD）和主机软件。USB的总线接口层主要是指以主机控制器（HC）为核心的硬件部分，包括USB主机控制器（HC）、串行接口引擎（Serial Interface Engine SIE）及其之间的电器连接等。其中主机控制器也集成了USB根Hub，以此提供了USB设备的连接点。

USB主机驱动(USBD)与底层硬件无关，通过HCD与底层控制器通信。USBD管理连接到主机上的每一个USB连接，并提供高层与USB设备通信的通 道。除此之外，USBD还自动对USB设备进行电源管理和分配带宽。在USB系统中，集线器(HUB)对USB系统的正常操作起到关键性作用，因此 USBD直接对集线器进行控制。这就意味着USBD还具有控制USB设备和集线器动态插拔的能力。（USBD）是客户驱动和HCD之间的中介，接受客户发 送来命令，发送给HCD，HCD在驱动硬件接受。USBD入口函数usbdCoreEntry(pURB\_HEADER pUrb)，所有请求函数都先进入此函数，根据pUrb->function的值调用相应函数。

在USB驱动中上层通过标准的函数接口调用底层函数完成相应的功能，而底层函数在接收到数据的时候则是通过调用上层函数的回调函数将接收到的数据交给上层。具体来说表现在以下的方面：

USBD层实现了一个同样通用接口函数库usbdLib，应用层可以通过该函数库usbdLib完成对与USB设备相应的层次的交互。

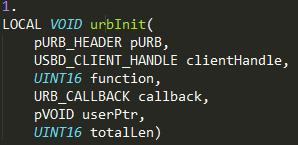
HCD层实现了一个通用接口函数库usbHcdLib，USBD层通过通用接口函数库实现对USB设备对应层次的交互；USBD层为上层保留的公共回调函数接口为pIrp->userCallback,对于control pipe,USBD层还为上层额外保留了一个回调函数接口pUrb->callback;

HCD层为USBD层保留的回调函数接口为pIrp->usbdCallback。

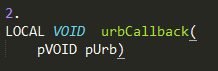
对于一个USB设备而言，只能够在USB总线上传输数据信号和控制信号，而不能够给出中断信号，这是因为USB总线并没有连接到计算机系统的PIC上。但是对于HC来说则是有严格意义上的中断信号，其中断信号线则是通过PCI接口接入到了PCI中，HC的中断处理函数是intHandler，他负责向任务intThread发送信号量pHost->intPending，而intThread在收到了信号量之后进行具体的中断处理，这样的处理方法使得中断处理程序简单且执行时间快，而消耗资源的部分则交给系统任务来实现，从而大大提高了中断响应的速度。

USBD层是一个抽象层，如同USB规范里所示，USBD层将USB设备抽象为一个node，从而将系统与USB设备的交互变成了client和USB node的交互。USBD层的实现可以两个子层：接口子层和实现子层。接口子层为上层提供了一系列的通用的接口，主要由函数库usbdLib来完成；而实 现子层则是实现了通用的接口函数的功能，主要由函数库usbdCoreLib来实现。

其实在函数库usbdCoreLib层已经实现了一个通用的接口函数urbExecBlock，只是这个接口函数过于复杂，不利于上层的调用，因此usbdLib通过调用函数urbExecBlock实现了一系列通用的接口，从而避开了应用层与usbd层隔离开来。



这个函数利用一个参数来初始化URB结构。URB结构将被用于通用接口函数的参数。



该函数被urbExecBlock()函数指定为URB结构的回调函数去调用函数usbdCoreEntry,函数usbdCoreEntry在结束前调用该函数，用于同步。



这个函数是usbdLIb函数库的核心，它负责初始化参数pUrb并以该参数调用函数usbdCoreEntry进行底层处理。

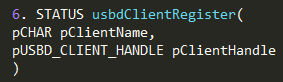


初始化函数库usbdLib。初始化的过程如下：

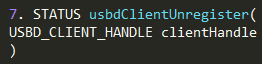
1. 创建semPoolQueue；
2. 调用urbInit(&urb.header,NULL,USBD\_FNC\_INITIALIZE,NULL,NULL,sizeof(urb))以及urbExecBlock（&urb.header）完成usbdCoreLib的初始化。



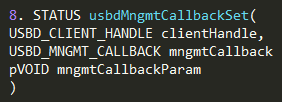
usbdInitialize函数的逆过程。先关闭usbdCoreLib层，然后再释放初始化函数分配的信号量队列空间，最后是关闭OSS函数库。



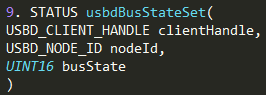
调用urbExecBlock函数（参数USBD\_FNC\_CLIENT\_REG）根据参数提供的pClientName生成一个USBD\_CLIENT结构，并返回该结构的handle。



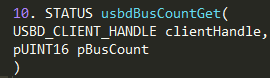
调用urbExecBlock（参数USBD\_FNC\_CLIENT\_UNREG，）删除一个clientHandle指定的USBD\_CLIENT结构。



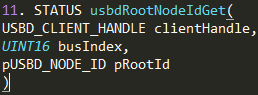
该函数为client设定一个management回调函数。该函数为USBD提供一种手段来通知USB上的异步管理事件。比如如果USB处于SUSPEND状态时，有一个USB设备发出了RESUME信号，那么这个事件将会通过management回调函数通知client。



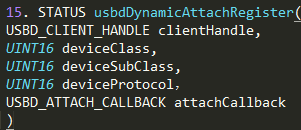
该函数允许一个client设置bus的状态（SUSPEND/RESUME），该总线由参数node指定（pNode->pBus）。 当设定一条bus为SUSPEND状态，必须清楚USBD不会自动将该bus恢复为RESUME状态，要想恢复为RESUME状态，还必须再次调用该函数 （使用RESUME参数）。这对于一个USB “remote wakeup”特性来说是很重要的，该特性允许一个远端设备驱动总线上的RESUME信号。Client正是通过management callback监控到该信号，并调用usbdBusStateSet函数来完成的。同样，这个函数还是调用了urbExecBlock（实际上是usbdCoreLib）来完成的。注意：client必须小心使用该函数，因为它会影响到一个bus的所有USB设备，从而影响到和这些设备通信的所有clients。



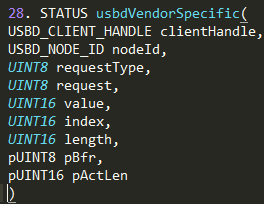
该函数是为了获取链接到系统中的所有USB host 控制器的数目。根据USB规范，每个host controller都有自己的root hub。系统中的HC的数目不是固定的，它可以通过usbdHcdAttach()函数和usbdHcdDetach()函数来修改。



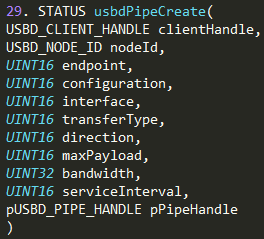
该函数是为了找到一个HC的root hub 对应的nodeID。busIndex指的是hcdList链上USBD\_HCD结构的下标。



该函数由client来调用执行，目的是告诉USBD：当USBD发现一个deviceClass/deviceSubClass/deviceProtocol的设备插拔时，通知该client。



该函数允许设备发出设备特殊的USB请求。特定的USB设备会有一些特殊的USB请求，这些请求不能用标准的USB功能来完成，因此就需要为client提供一个函数向该USB设备的控制pipe直接发送请求。



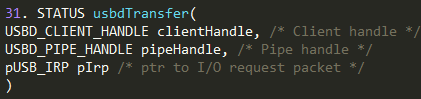
创建一个USBD\_PIPE。nodeId：指明了设备。endpoint：指明了设备的endpoint，而pipe正是client和一个设备的endpoint之间的联系通道。Configuration和interface正指定了设备的配置和接口。transferType：传输类型（control和BULK）

Direction：指明传输的方向（IN/OUT/INOUT）。一个pipe的方向特性是不会改变的。INOUT仅仅用于control Pipe。 maxPayload：该endpoint支持的最大的payLoad。这个值通常在设备的描述符中都标明了，因此这个参数是USBD首先从USB设备读取了配置描述符之后才直接将这个数值作为参数传递的。

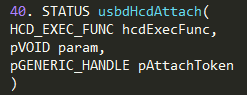
Bandwidth：对control和bulk Pipe来说，bandwidth为0；对interrupt pipe来说该数值为每个frame要传递的字节数；对于isochronous pipe，bandwidth指的是每秒钟传输的字节数。

serviceInterval：仅适用于中断传输pipe，标明该pipe的最大潜伏期（latency，单位毫秒）。如果一个设备的serviceInterval数值为20，说明该设备没20毫秒需要服务一次。

pPipeHandle：创建的PIPE的handle。



Client使用该函数来启动一个pipe上的传输，传输是由一个IRP结构或者一个I/O request packet来描述的，该IRP结构或者I/O request packet必须要在调用usbdTransfer()函数前被分配并初始化。

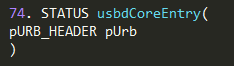


多数的系统在系统初始化的时候调用此函数，以便register一个或者是多个HCD。usbdHcdAttach函数是usbdLib函数库中为数不多的不需要首先调用usbdClientRegister()函数就可以使用的一个函数，因此就不需要向该函数传输一个USBD\_CLIENT\_HANDLE参数。

6.2.3 函数库usbdCoreLib

函数库usbdCoreLib是USBD层的具体实现。当USBD首次初始化时，它创建了一个内部client，该内部client主要被USBD用于控制每个 USB hub/device的控制pipe传输，这是因为当一个USB设备刚刚插上时，系统并不知道该设备是一个什么类型的设备，也不知道该设备需要什么类型的 pipe，而只能通过一个默认的pipe进行通信，而该内部client则是正好起到了利用默认pipe和新发现USB设备通信的作用，且该client自创建后一直保持active状态直至USBD被关闭（shutdown）。对于包括内部client在内的所有client，USBD会为每个client分配一个独立的任务clientThread，该任务继承了内部 client任务的优先级。该任务通常是unactive的（因等待消息而休眠），仅仅在一个client的回调函数被执行（发送消息）的时候才被激活.因此当一个client被其唯一的一个回调函数被激活时，其他的USBD任务不受影响。

对每个通过usbdHcdAttach()函数attach到USBD的HC来说，USBD同样创建了一个唯一的总线监控任务busThread。该任务 负责配置和监控一个给定USB总线上所有HUB的状态。系统不断向每个HUB发送一个状态查询IRP，当发现该HUB上一个端口出现了设备插拔状态改变 时，总线监控线程就负责更新内部数据结构，配置HUB，并触发动态插拔回调函数。



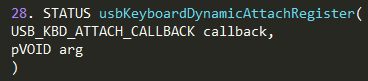
usbdCoreLib库的唯一入口函数。主要调用名字为fnc开始的函数。

从函数库usbdLib和函数库uabdCoreLib可以看出，在USBD层，所有的相关的概念都被抽象为几类：NODE、PIPE、CLIENT以及IRP，其中的一个USB设备抽象为一个NODE，对应的数据结构为USBD\_NODE。一个client即对应一个任务，主要负责和一个或者是多个USB设备的交互，而client和NODE交互的通道即是pipe一个pipe可以为IN、OUT或者INOUT类型，其中INOUT类型主要对应于控制PIPE。一个PIPE建立在一个client和USB设备的一个endpoint之间。一个USB设备可能有多个endpoint。一个IRP则是隶属于一个PIPE。



初始化usbKeyBoardLib库。整个初始化过程主要完成如下的工作：

1. 建立线程typematicThread，该线程的主要作用是：当持续按下某个按键的时候要进行特殊的处理，比如当持续按下字符按键的时候等于重复输入等等；
2. 调用usbdClientRegister创建USBD\_CLIENT结构并创建clientThread线程。这说明USB键盘设备共用一个clientThread线程
3. 调用usbdDynamaicAttachRegister在新创建的USBD\_CLIENT结构变量中注册一个USBD\_NOTIFY\_REQ结构变量，当USB系统中出现了符合USBD\_NOTIFY\_REQ结构变量的设备（USB键盘）插拔时，通过回调函数通知client。同时也要立刻对现有的USB设备进行检查，如果发现有符合USBD\_NOTIFY\_REQ变量的设备，马上通知client。这样既保证了当有USB键盘插拔时通知client又能保证即使在该函数被invoke之前插入的USB设备也能够正确的处理。



Vxworks操作系统可以支持多个USB键盘，在函数库usbKeyboardLib中的sioList链中记录了所有的USB键盘结构。由于USB设备是支持热插拔的， 因此当新插入一个设备时就需要对其进行初步识别并调用相应的管理程序以完成初始化等操作，因此需要在系统中首先进行注册，告诉系统“如果发现了某USB键 盘设备，调用这个函数通知我”，正如生活中的“我是xxx，这是我的名片，我的职责是XXX，如果有什么问题打这个电话通知我”，如果多个程序都对USB 键盘的插入感兴趣，则可以向reqList链中存放多个ATTACH\_REQUEST结构。函数。usbKeyboardDynamicAttachRegister的作用正是在reqList链中增加一个ATTACH\_REQUEST结构。注意：函数usbKeyboardDynamicAttachRegister只是在名片夹（reqLink链）里放了一张名片 （ATTACH\_REQUEST结构变量）；而每张名片又有自己的回调函数（如 usbKbdDrvAttachCallback）；usbKeyboardAttachCallback回调函数的作用则是遇到USB键盘设备插拔的时 候根据名片夹（reqLink链）的每张名片（每个ATTACH\_REQUEST结构变量）挨个通知；而 usbdDynamicAttachRegister函数的作用则是向USBD层注册了一个名片夹，当然如果需要可以注册多个名片夹。

在调用这个函数之前，可能就有匹配的设备存在了，因此需要注册reqList后检查sioList，如果有USB键盘设备则调用回调函数。

VxW0rks操作系统的USB2.0驱动分析

<http://www.elecfans.com/emb/app/20111125253627.html>

在协议栈的最底部是USB控制器USB HC(USBHost　Controller)，这是主机系统当中控制每一个USB设备的硬件部分，目前主要有两类的USB控制器：一种是通用主机控制器(UHCI)，另一种是开放主机控制器(OHCI)。硬件厂商一般根据这两个规范来设计USB主机控制器。对于每一种类型的主机控制器都有一个与硬件独立的USB主控制器驱动HCD（Host Controller Driver）。WindRiver公司提供了两类驱动：usbHcdUhciLib（UHCI主控制器库）和usbHcdOhciLib（OHCI主控制器库）。

Usb主机驱动(USBD)与底层的硬件没有关系，通过HCD与底层的控制器通信，USBD管理连接到主机上的每一个USB链接，并且提供高层与USB设备通信的通道。除此之外，USBD还自动对USB设备进行电源管理和分配带宽。在USB系统中，集线器（HUB）对USB系统的正常工作起到关键性的作用，因此USBD直接对集线器进行控制。这就意味着USBD还具有控制USB设备和集线器的动态插拔的能力。

在主机栈的最顶层是USB设备驱动程序。设备驱动程序依赖USBD提供的通信通道，驱动连接到USB系统中的USB设备。USB主机栈的设计关键是USB主机驱动（USBD）的设计，USBD为USB设备驱动程序提供标准的API，为主机控制器驱动提供HCD API。

1. USB主机栈的启动过程

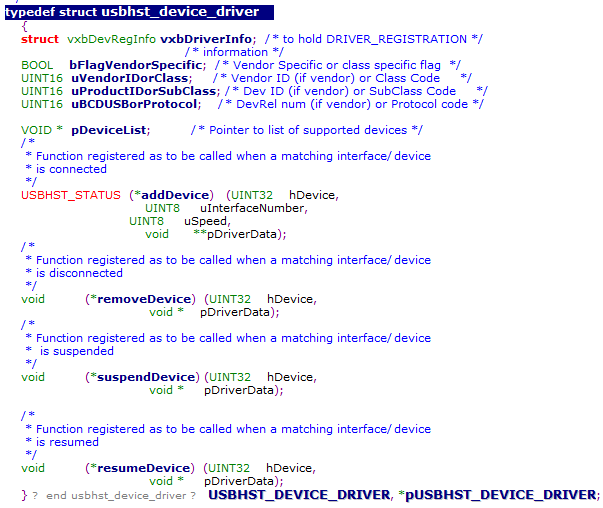
分为3步：1.初始化USB主机驱动（USBD）。2.加载USB主控制器驱动（USBD）；3.加载USB设备驱动程序。

1. USB主机驱动（USBD）分析

在使用USBD提供的接口之前，首先需要对USBD进行初始化，USBD的初始化分为3步，其步骤如下：1.调用USBD的入口函数usbdInit(),usbdInit()初始化USBD要用到的数据结构；2.调用USB集线器（HUB）的初始化函数(usbHubInit),初始化集线器驱动的数据结构；3.使用函数usbHCDInit()挂接相应的主机驱动。

对于每一个挂接到USBD上的主机驱动程序，集线器驱动程序都会为其创建一个优先级为100的任务来监视USB总线上的事件，这个任务一直处于睡眠状态，一旦集线器上的某一个端口状态发生了改变，该任务立即被激活。由于该任务的优先级为100，可以立刻获得资源变为运行状态将总线的状态报告给USBD。

USB设备类驱动首先需要在USBD中注册，USBD为每一个设备类维护着一个个设备类结构。该设备类的结构如下：



当USB设备插入或者是拔出USB系统的时候，集线器驱动程序将USB设备的插入或者是拔出的信息通知给USBD，USBD根据得到的设备信息（例如：设备类号，子类号，协议号等）查找设备的驱动列表，查找到该设备的驱动程序以后，调用USB设备的相应函数，进行设备的相应操作。当一个设备类不在系统中使用时，通过撤销函数来撤销注册，该操作的实质是删除相应的设备类的设备类结构。

在USB里面一共有四种传输：控制、中断、批量、等时传输。控制和批量端点用于异步的数据传输，驱动需要他们就立马工作。中断和等时端点是周期性的，即在固定时间段内连续的传输数据。