初涉 USB，初学者 USB 入门总结（1）枚举

一，概述

现在很多的主控上都带有 USB 的功能，但是对于初学者来说，这方面应用还是比较棘手，因为 usb 的不但固件程序需要编写，PC 端的驱动也要编写，而且驱动写好了还要写个上位机才能看出效果。这样调试起来十分困难，建议从 USB 的键盘，鼠标开始做，了解清楚了，再做自己的协议就比较简单了。

USB 的概念历史啥的这里就不说了。我们先不管具体的数据包格式，这一节先从整个包的层面上简单的说，过程是这样的，

----------设备插入----------

1） 主机会轮回查询各个 USB 端口，主机检测到 D+与 D-之间有电压差，就认为有新的设置接入。主机等待 100ms 后发出复位请求。设备接到复位请求后将产生一个外部中断信号。

----------枚举过程----------

2） 主机这时候只是知道有新的设备插入了，但是不知道插进来个什么东西，所以就开始询问它是什么设备，怎么用，负荷能力怎么样。这个时侯就进入了枚举过程。因为刚刚插入的设备没有分配地址，就用默认地址 0，首先发送一个 Get_descriptor（获取设备描述符）指令包，设备接到包后就开始解析包（其实就是你在固件程序里判断处理），然后按固定格式返回自己设备的设备描述符，这一步主要是主机知道你的 USB 设备的基础属性，比如支持的传输数据长度，电流负荷多少，支持那个 USB 版本，以及以后方便电脑找驱动的 PID，VID。

3） 这时候主机知道你（你做的设备，简称你吧）的数据长度还有电流大小后，下一步就是给你分配一个属于你的地址。

4） 给你一个地址后就开始询问你的具体配置。首先发送一个试探性的设备配置请求 Get_configuration（要求固定返回 9 个设备配置字），你接到后就开始发送 9 字节的设备配置字，其中包括你的配置字的总长度，这样主机就知道你的配置到底有多长，然后再发一次设备配置请求，这时你就开始上传所有的配置字。这个时侯主机就已经很明白你的工作方式就各种特性，然后就可以正常工作了

5） 如果你在前面的某些配置（以后章节详细说明）要求要说明自己的名字什么的，这里还要上传字符串描述符。

6） 如果是鼠标或者键盘还要上传报告描述符

----------正常数据阶段----------

7） 这个时侯你已经被主机正式接受并且注册了，你可以通过自己写测驱动或通用驱动与电脑进行通讯了。

以上是简单的描述，详细的后面章节再做介绍，学习一个东西关键是首先要知道这个东西是什么，简单的工作原理。对于 USB 的工作我这里做个比方，

主机好比一个公司，你就是 USB 设备，要进入公司首先要面试（枚举），你到了面试现场（第一次插入设备），面试官首先了解到你的外表，性别已经你要应聘的岗位（设备描述符），然后给你一个号，以后就开始按号叫人，当你被叫到就开始问你的专业知识，性格等（配置描述符），如果你比较合适（通过了枚举）你就会录取了，并且注册一个你的信息到公司（驱动安装，并且写入注册表）。等你下次来公司，只要把工号（PID，VID）报上，就知道是你来了。

初涉 USB，初学者 USB 入门总结（2） 设备固件程序

二，实际数据过程测试

（图片颜色显示不出，可以到我博客 http://blog.csdn.net/arthur05611/archive/2009/02/23/3929778.aspx）

这节主要是对固件里的 USB 请求处理有个概念，还有就是调试的方法。大篇幅的程序配合，如果不关心这一块的话可以跳过，呵呵。

为了更好的说明整个 USB 启动过程，我们可以用串口实时的跟踪各个 USB 中断。不过这里先不用串口进行测试，只是简单的用一组变量记录过程。测试程序如下（以下会有程序的说明）：

```
uchar test[100];//100 长度的变量，记录过程
uchar conters=0;//记录计数值，
/*-------------------------------------------------------------
        高校电子联盟--肖继达
     QQ：258347765
-------------------------------------------------------------*/

void EXT_int(void)//USB 中断响应函数
{
    /*-------------------------------------------------------------
          Check interrupt status register to know interrupt
       source.
    -------------------------------------------------------------*/

    if (USB_BUSRESET_ASS_INT())
    {   /* USB bus reset */
        /*  for USB Rev.1.1
                    After USB bus reset released, 10msec recoverly time we have.
                    Follwing request must be processed normally.
        */

        CLR_BUS_RESET_STATE();  /* USB bus reset status clear */

        /*-------------------------------------------------------------
           Endpoint0 setting
        -------------------------------------------------------------*/
        /* Tx/Rx payload size setting */
        /* Rx payload is fixed as 8-byte or 32-byte, therefor the
           setting has no meaninig */

        SET_PAYLOAD_EPn(EP0RX, device_descriptor.bMaxPacketSize0);
        SET_PAYLOAD_EPn(EP0TX, device_descriptor.bMaxPacketSize0);
        /* Stall bit, the value undefined after reset, cleared */
        CLR_STALL_EPn(EP0);
```

```
        /*-------------------------------------------------------------
           Misceronous status variable initialization
        -------------------------------------------------------------*/
        usb_status.configuration = NULL;
        usb_status.remote_wakeup = 0;
        usb_status.address = 0;
        usb_status.dvcstate = DEFAULT_STATE;    /* Device state :DEFAULT */
        usb_status.stall_req = 0;
               #ifdef Debug
                      test[conters]='!';
                      conters++;
                      #endif

        /*-------------------------------------------------------------
           Callback to application layer
        -------------------------------------------------------------*/
        (*usb_status.callback)();
    }
    else if (SUSPENDED_INT())
    {   /* suspended state */
        /* for USB Rev.1.1
                      Transit to suspended state after detect the USB line has kept
idle over 3msec.
                      After resume detected, end suspend state in 3msec to be able
to respond
                      the host request.
        */
              CLR_SUSPENDED_STATE();
                      #ifdef Debug
                      test[conters]='@';
                      conters++;
                      #endif

    }

    else if (AWAKE_INT())
    {   /* Deveice awake state */
        /* AWAKE procedure */

        CLR_AWAKE_STATE();          /* Request clear */
                      #ifdef Debug
                      test[conters]='#';
                      conters++;
```

```
                    #endif

    }
    else if (USB_BUSRESET_DES_INT())
    {   /* USB bus reset deassert */
        /* Procedure for USB bus reset de-assert */

        CLR_BUS_RESET_DES_STATE();  /* Request clear */
                    #ifdef Debug
                    test[conters]='$';
                    conters++;
                    #endif

    }

    else if (SOF_INT())
    {   /* SOF interrupt status */
        CLR_B_SOF_STATE();
               #ifdef Debug
                    test[conters]='%';
                    conters++;
                    #endif
       /* SOF interrupt status clear */
    }   /* SOF interrupt status */

    if (SETUP_RDY_INT())
    {   /* setup ready */
                  #ifdef Debug
                    test[conters]='^';
                    conters++;
                    #endif

        read_Device_Requests();
    }

    else if(EP1_PKTRDY_INT())
    {   /* EP1 packet ready */
        read_FIFO(EP1);
       }
    else if (EP2_PKTRDY_INT())
    {   /* EP2 packet ready */

        write_FIFO(EP2);
    }
```

```
    else if (EP0_RXPKTRDY_INT())
    {   /* EP0 receive packet ready */
        read_FIFO(EP0RX);
    }
    else if (EP0_TXPKTRDY_INT())
    {   /* EP0 transmit packet ready */
        write_FIFO(EP0TX);
    }

}
```

计录的结果在变量查看中显示如下：

首先我解释一下，这段程序是我在做 USB 设备时的中断函数。主控（就是你往里面写固件程序的那个东西）会在要求设备进行操作时，产生一个相应的中断（我们可以用中断的方式，也可以用查询的方式，中断的方式的好处就是主机有需要操作的都会叫你，而用查询你必须不断的问主机“有事么”，这里采用中断方式），比如主机给设备设置地址，主机会通过固定的通道（point0）发送一个‘设定地址’包，设备主控接到包后会产生中断，并且把响应的状态保存在相应的寄存器中，我们只要在中断程序判断各个寄存器就能完成主机的任务。程序中蓝色字是中断类型的判断，其对应的宏定义就不列出来了。如果是这个中断就会执行相应的中断操作。并且一次中断只有一种中断类型，我们在每个中断响应中加一段红色字的程序，是为了保存每次中断的状态，比如刚插上设备，来了一次 BUSRESET 总线复位中断，就会进入相应的中断操作，完了后记录状态 test[conters]='!'; conters++;意思是进入了这个中断就在这一组数的当前位置设成'!'，并且位置记录的变量加一，以便下一次记录到下一个位置。这样 USB 的过程我们就记录了下来，

下面看一下记录结果（其中的数字和字母是响应标准请求时的程序产生的这里不罗列程序了）。

file:///C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/msohtml1/01/clip_image002.jpg

可以看到，一开始是一次总线复位，然后 USB bus reset de-assert，然后再挂起总线。重复了两次，然后就是上一节的具体配置了。

这节主要是对固件里的 USB 请求处理有个概念，还有就是调试的方法。

初涉 USB，初学者 USB 入门总结（3） 数据包阐述

对于 USB 传输大体有个概念，下一步就来看看到底 USB 上传的什么东西，以什么格式传数据，先不涉及端点的概念。

各种总线的数据传输都是以固定的层次协议进行的，USB 当然也不例外。所谓的层次也只是个抽象的概念罢了，就是表达一种依附关系，上层要依赖与底层，上层以底层为基础，上层只需要关心自己的东西就行了，如果你还不明白，那就继续看，学习一个东西不可能一两句话说的明白一个点，需要全面了解后才能清楚各个点。

要实现两个机器（机器的范围比较广，可以是电脑，交换机，单片机）的通信总是要有一个载体才可以，对于机器当然是电平高低为载体，具体的说机器甲要告诉机器乙一件事情（比如说一条指令），那么机器甲可以通过一根线（串行数据总线）连到机器乙的一个 IO 口上，

甲发送一个个的高低电平，乙固定时间检测自己的这个 IO 口，然后逐个记录下放到自己的缓冲里，这样乙就收到甲送的数据了。上述就是一个简单的数据链路层（计算机网络里这么叫）的描述，这一层要保证的就是甲发的每一位数据，乙都可以正确及时的接受，并且对在传输过程中出错的数据做出反应。其实比数据连路更底层的还有物理层，这就是真正的物理介质，对于机器就是电线了，数据就是电线上传输的电压,USB 是用的四线，两个电源，两个数据线。

这里也打个比方，比如人与人进行交流，我们当然是通过说话了，物理层就是空气和传输的声波，数据链路层就是我们说的每一个字，物理层就是空气，负责把我们说的话转换成声波传给对方，数据链路层负责让对方能正确的听到每个字，如果听的不清可以告诉对方重新说一遍。

经过上述的两个底层，就可以保证每一位数据可以正确的传到对方那里去。下一步的工作当然是解析数据代表了什么，一般来说，数据都是以一串数为单位，一般称为一个包，机器间传输都是以一个包为单位传出，就像人们说话都是以一句话为单位输出一样。每一个包包含有许多位数据，这些数据又分段表示不同的意义，如图一，这是一个 USB 令牌阶段的包，Sync 是同步数据（相当于说话时先打个招呼，告诉对方要跟他说话了），PID 是包标示（告诉对方这个包是干什么用的），ADDR 是对方的地址（叫对方的名字），ENDP 是用端点几通讯（先不介绍这个），CRC5 是校验位（判断这个包是否在传输中出错），EOP 是包结束。

Sync	PID	ADDR	ENDP	CRC5	EOP

图一

USB 的数据包又分为三种，一个是令牌包，一个是数据包，另一个是握手包。每一次的 USB 通讯事务处理都是以令牌包开头，告诉对方要跟谁说话，这句话是用来干嘛的。如果要求有数据传输，则下一步就是数据包，另外如果要求对方要有反馈，则会发出握手包。令牌包又简单的包括 OUT，IN，STEP 三种类型，OUT 是用于主机告诉设备主机要向 USB 设备发送数据，IN 是用于主机告诉设备要上传数据，而 STEUP 是用于主机向 USB 设备发送配置信息，在枚举过程中会用到。另外数据包和握手包的具体格式什么的，可以参照详细的协议。

可以看到在所以的通讯过程中，主机都是发起者，不管是主机发送数据到 USB 设备还是 USB 设备发送数据到主机，都必须收主机控制。图二为一次事务的过程

令牌阶段　——》　数据阶段　——》　握手阶段

图二

这个过程可以这样描述，甲和乙对话，甲是老板，乙是职员。第一节已经讲过了，乙面试就是枚举，在这个过程中，甲多段的发送 STEP 令牌包给乙，乙收到后如果要反馈数据，就发数据包给甲，甲正确接收后，跟甲握握手，表示这次对话成功。

乙被正式录取后，甲会分派任务（OUT），这时甲对乙说有任务给你（令牌阶段），然后乙就开始听，甲说你的任务就是记录数据并且上报（这段话就是数据包），乙说好的（握手包）。乙开始正式工作，并且记录数据。过了一段时间，甲开始要求提交数据（IN），乙把数据报告给甲（数据阶段），甲说好（握手成功）。这里乙不能主动的去向老板汇报，只能被动的干活。

上面已经讲 USB 主机和设备间数据传输的过程，都是我个人理解，有不正确和不到位的大家提出，方便初学者理解，谢谢・・

初涉 USB，初学者 USB 入门总结（4）USB 通讯设备快速开发

经过上述三节的描述，对 USB 应该已经有了初步的认识，其中具体的协议（比如各个描述符的定义什么的）这里不做描述了，网上一搜一大堆。下面我以一个实例来详细说明快速开发 USB 设备的步骤，

一，设定规划

凡事预则立，不预则费，所以开发一个小小的 USB 也要稍微规划一下，比如想象要实现什么功能，传输的数据协议什么的。

二，固件编程，

固件编程说白了就是写单片机程序，要实现 USB 一般可以使用带 USB 功能的单片机，再个就是加一个专用的 USB 芯片。这里以内部集成 USB 功能单片机为例

固件的 USB 开发一般就是先使能 USB，使能 USB 时钟，使能各个 USB 控制中断（挂起，复位，标准请求，写入，写出等）然后 USB 就能正常工作了，这时候不如不写别的东西，电脑就可以检测出有 USB 设备插入了，具体的反应是在设备管理器里会发现闪了一下说明发现了新的 USB 设备，接下来电脑会发送各种标准请求，因为这个时候你的程序还没写完整，对这些请求不会有反应，所以电脑不可能识别出是什么东西。

接下来的工作就是在中断中响应电脑传来的各种标准请求。当必要的请求都被正确的响应的话，这个时候如果电脑里有正确的驱动，电脑就会去加载这个驱动，如果是第一次插入这个设备，还要把驱动安装一下，然后设备就进入正常工作了，电脑会显示“这个 USB 已经成功安装并可以应用了”。

这里捎带着说一下端点（endpoint）的概念，一般一个 USB 设备都会有数个端点，端点就是一个数据缓冲控制区（FIFO），每个缓冲区相当于有一个出口一个进口的池子，数据通过进口进入到池子，然后你再在固件里去用这些数据。固件往电脑写数据，也是把数据先放到池子里，然后打开出口，就可以干自己的事情，不用一个个的把数据发出了，池子的出口自动把数据流出。

一般的端口 0 是用来做标准请求响应用的，也就是在枚举阶段用到。我一般把端口 1 定义为出（OUT），端口 2 定义为入（IN）（注意，这个 OUT 和 IN 是相对与电脑的，也就是说 OUT 是数据从电脑出去到设备，IN 是设备的数据进入电脑）。这些定义也是在标准请求中去告诉电脑的。

接下来就可以实现与电脑的通讯了，你把数据放到相应的池子里就行了。下面就可以自己定义通讯的数据格式了。比如控制开发板上的 8 个 LED 的第一个灯亮，那么上位机发送数据 0x55，0x01，0x80，0xaa。我们就可以规定第一个数据是启示位，遇到这个表明开始一次控制指令，0x01 表示这个是控制灯亮暗的指令，0x80 表示 LED 的控制数据，最高位是 1，表示第一个亮，其他位是 0，表示都暗。最后一个数据是 0xaa，表示这是结束。其实所谓的数据协议不过就是自己定义的一套让通讯双方都能正确理解对方的数据格式。电脑比较是电脑，什么都要规定好了，它才能正确的工作。

三，驱动程序

对于快速开发用 Driverstudio 就可以了，我先装了 VC6.0，然后装了 DDK2600，最后装了 Driverstudio，网上有说这个顺序不容易出问题，我也没时间去试别的顺序会出怎么样的特效，姑且不管他是否在忽悠，先这样按了没坏处。

我一开始比较新潮的装了 DriverStudio3.2 版本，然后按网上的方法破解了，生成了驱动是能打开设备，但是就是传输不了数据，搞了两天还是不行，后来想到是不是 3.2 版本太新了？或者破解没完整？然后卸载了 3.2 装了 3.1，果然可以了，真不知道是 Compuware 做了手脚故意玩我还是本人愚笨弄错了哪里。

驱动生成的步骤可以在百度，Google 里搜“10 分钟完成一个 USB 驱动程序”能出来一

大堆，要是你嫌搜索麻烦就直接点这个算了 http://www.4oa.com/Article/html/6/33/482/2005/17317.html 按那个步骤操作就可以了，根据向导操作完了以后，VC 就会出来一个驱动程序框架了，如果你在这个时候编译一下就可你会碰到很多问题，我的操作是这样的。首先把 DDK 的库编译一下，操作网上有， 网上有云：

1. 启动 Visual C++ 。
2. 选择菜单 File|Open Workspace 。打开位于 DriverStudio/DriverWorks/Source/vdwlibs.dsw 的工作空间文件。
3. 选择菜单 Build|Batch Build,在弹出的对话框中选择你想编译的库。
4. 点击 Build 编译你选择的库。

然后在 VC 的 Driverstudio 的工具条点击“change environment variables”，在第一个选型卡把 DDK 的路径选上，我的是 C:\WINDDK\2600。然后点 OK，接下来点 DriverStudio 工具条的编译，就可以了，如果你还是碰到问题，你可以把 VC 显示的错误复制到百度。

==========

推荐：**EDN 助学－USB2.0 CY7C68013 小组** **http://group.ednchina.com/1754/**

资料】中文 datasheet: cy7c68013 资料译文.pdf

http://group.ednchina.com/1754/28965.aspx

等等*****

- EDN 助学 SoEZ-USB2.0 开发板介绍：带 IDE 移动硬盘功能，以及 HID 键盘功能。驱动以及固件源码丰富，赠送视频教程。丰富的电子书等资源，助您一臂之力！

注意：CY7C68013A 支持 USB2.0 高速传输模式(480Mbps)

详情点击：SoEZ-USB 2.0 开发板

另有特价版：SoEZ-USB 精简版