来源: <http://blog.sina.com.cn/s/blog_640029b30100vvqb.html>

**USB的工作过程**

当usb设备接入到主机时，主机开始枚举usb设备，并向usb设备发出指令要求获取usb设备的相关描述信息，其中包括设备描述（device descriptor）、配置描述（configuration descriptor）、接口描述（interface descriptor）、端点描述（endpoint descriptor）等。这些信息是通过端点0（endpoint 0）传送到主机的。获取各种描述信息后，操作系统会为其配置相应的资源。这样主机就可以与设备之间进行通信了。

usb通讯有四种通讯方式控制（control）、中断（interrupt）、批量（bulk）和同步（ synchronous）。usb通讯是通过管道（pipe）实现的。管道是一个抽象的概念，指的是主机与设备之间通讯的虚拟链路。比如说一个usb通讯主机A和设备B，其中有bulk in（批量输入）、bulk out（批量输出）、control out（控制输出）三种通讯方式，那么A与B之间的通讯管道就有三个。（这里明确一个概念，在usb通信中数据流向都是相对主机来说的，in表示设备向主机传送数据，out表示表示主机箱设备传输数据）。在设备一端，每个管道对应一个端点，端点配置相关的寄存器和缓冲区。在通讯之前需对端点进行相关设置。在通信中，只需向缓冲写或读数据，并置位相关比特位即可。

下面具体从usb的中断输入输出来讲述基于keil C mdk开发环境的stm32的USB接口单片机程序设计。值得一提的是，st或相关公司给我们提供许多封装函数和相关例子，我们可以根据其中的例子并进行修改即可实现我们自己需要的usb通讯程序。

**1.usb描述符配置**

从上面的讲述可以看出，usb描述符是usb通讯的前提。主机必须先了解设备后才能与其进行通讯。在st提供的例子中，描述符都在usb\_des.c文件进行定义，下面就其中的Joystick例子说明usb描述符的配置。

**1.1设备描述符**

*const u8 Joystick\_DeviceDescriptor[JOYSTICK\_SIZ\_DEVICE\_DESC] =*

*{*

*0x12,*

*USB\_DEVICE\_DESCRIPTOR\_TYPE,*

*0x00,*

*0x02,*

*0x00,*

*0x00,*

*0x00,*

*0x40,*

*0x84,*

*0x19,*

*0x06,*

*0x04,*

*0x00,*

*0x02,*

*1,*

*2,*

*3,*

*0x01*

*}*

设备描述符两个重要参数是生产商ID和产品ID，主机将根据以上两个ID为设备选择相应驱动程序。在我们的应用中，我们一般只需修改例子中的这儿两个参数即可完成设备描述符的设置。

**1.2配置描述符**

*const u8 Joystick\_ConfigDescriptor[JOYSTICK\_SIZ\_CONFIG\_DESC] =*

*{*

*0x09,*

*USB\_CONFIGURATION\_DESCRIPTOR\_TYPE,*

*JOYSTICK\_SIZ\_CONFIG\_DESC,*

*0x00,*

*0x01,*

*0x01,*

*0x00,*

*0xE0,*

*0x32,*

*0x09,*

*USB\_INTERFACE\_DESCRIPTOR\_TYPE,*

*0x00,*

*0x00,*

*0x02,*

*0x00,*

*0x00,*

*0x00,*

*0,*

*0x07,*

*USB\_ENDPOINT\_DESCRIPTOR\_TYPE,*

*0x81,*

*0x03,*

*0x08,*

*0x00,*

*0x20,*

*0x07,*

*USB\_ENDPOINT\_DESCRIPTOR\_TYPE,*

*0x01,*

*0x03,*

*0x40,*

*0x00,*

*0x20,*

*}*

配置描述符中包括了接口、端点的配置。如果设备为HID设备，在配置描述符中还应加入HID描述，具体描述可以参照Joystick例子的配置。还有一些其他配置可以参可相关资料与例子加以理解。

**2.USB通讯的执行过程**

首先，当主机数据传送到USB设备，USB怎样接收命令和数据呢?USB首先会产生一个中断，这个中断在stm32fxxx\_it.c文件的USB\_HP\_CAN\_TX\_IRQHandler和USB\_LP\_CAN\_RX0\_IRQHandler中定义，一般使用USB\_LP\_CAN\_RX0\_IRQHandler。在这个函数中继续调用USB\_Istr()函数，这个函数是usb通讯的关键。它接收到主机命令，指派调度相应函数进行处理。对于这一点，详细过程我现在还不是很明白。如果以后搞懂了再补述。

当USB设备接入主机时，主机要枚举该USB设备，他将要求USB设备提供自身相关信息，这是通过endpoint0实现的。endpoint0是一个特殊的端点，每一个接口（interface）必须有endpoint0。一般情况下，我们需要使用多个端点（如前所述，配置描述符定义了端点的数目、类型、传输数据大小等）。在使用端点前需对端点进行初始化。这个过程在usb\_prop.c文件中的xxx\_reset()函数定义。如我定义端点1的两种传输方式：

*SetEPType(ENDP1, EP\_INTERRUPT);  
SetEPRxAddr(ENDP1, ENDP1\_RXADDR);  
SetEPRxCount(ENDP1, 8);  
SetEPRxStatus(ENDP1, EP\_RX\_VALID);  
SetEPType(ENDP1, EP\_INTERRUPT);  
SetEPTxAddr(ENDP1, ENDP1\_TXADDR);  
SetEPTxCount(ENDP1, 64);  
SetEPTxStatus(ENDP1, EP\_TX\_NAK);*

在定义完端点后，我们就可以使用端点进行数据传输了。

**向主机输入数据（in）：IN传输过程是**  
1.向缓冲区填入数据；  
2.设定USB数据计数器：  
3.设置USB输出有效。

*XXX\_send()  
{  
 UserToPMABufferCopy(sendBuffer, ENDP1\_TXADDR, 2);  
 SetEPTxCount(ENDP1, 2);  
 SetEPTxValid(ENDP1);  
}*

注意一般情况下，端点的输入输出缓冲区地址没有定义，须在usb\_conf.h中定义具体定义可以参考端点0的定义。  
  
**读从主机输出的数据（out）：out传输过程是**  
1.定义out回调函数；  
2.从缓冲区读出数据：  
3.设置USB输入有效。

*void EP1\_OUT\_Callback(void)  
{  
 u8 DataLen;     
 DataLen = GetEPRxCount(ENDP1);  
 PMAToUserBufferCopy(rcvData, ENDP1\_RXADDR, DataLen);  
 SetEPRxValid(ENDP1);  
}*

注意在一般情况下,EPX\_OUT\_Callback（）回调函数的申明为空执行函数。需将usb\_conf.h中#define EPX\_IN\_Callback   NOP\_Process隐掉。再在合适的地方从新定义void EP1\_OUT\_Callback(void)（合适的位置是指定义之后运行不会出现EP1\_OUT\_Callback为申明的错误就行）。