STM32F103 上 USB 的端点资源灵活使用

前言

理解 STM32F103 上 USB 模块的端点资源，灵活在应用中的配置。

问题

某客户使用 STM32F103 的 USB 模块做设备时和上位机 PC 连接时碰到一个问题：PC 端驱动已经固定好，是对下位机 USB 设备上的地址编号为 0x0A 和 0x0B 的两个端点通信，从 0x0A 端点读取数据，向 0x0B 端点写数据。而 STM32F103 的 USB 模块只有 8 个双向端点，能否支持这样的寻址。

1. 问题调研

我们先来看看 STM32F103 上的 USB 端点资源。从 STM32F103 参考手册（RM0008）可知，一共有 8 个双向端点，对应 8 个寄存器来控制其属性和表征其状态。如下图，可知每一对端点必须配置成相同的端点地址，这个地址位域是 4 位，取值从 0x0 到 0x0F 范围。

USB endpoint n register (USB_EPnR), n=[0..7]

Address offset: 0x00 to 0x1C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTR_RX	DTOG_RX	STAT_RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR_TX	DTOG_TX	STAT_TX[1:0]		EA[3:0]			
rc_w0	t	t	t	r	rw	rw	rw	rc_w0	t	t	t	rw	rw	rw	rw

和以下摘录的 USB 规范符合：

8.3.2.2 Endpoint Field

An additional four-bit endpoint (ENDP) field, shown in Figure 8-3, permits more flexible addressing of functions in which more than one endpoint is required. Except for endpoint address zero, endpoint numbers are function-specific. The endpoint field is defined for IN, SETUP, and OUT tokens and the PING special token. All functions must support a control pipe at endpoint number zero (the Default Control Pipe). Low-speed devices support a maximum of three pipes per function: a control pipe at endpoint number zero plus two additional pipes (either two control pipes, a control pipe and a interrupt endpoint, or two interrupt endpoints). Full-speed and high-speed functions may support up to a maximum of 16 IN and OUT endpoints.

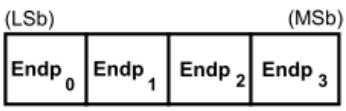

Figure 8-3. Endpoint Field

客户使用的是 STSW-STM32121（STM32F10x, STM32L1xx and STM32F3xx 全速 USB 设备库），那么应该修改哪些代码呢？

2. 问题分析

首先，USB 设备通过端点描述符向主机 PC 报告它所使用的端点有哪些：每个端点的地址（即 USB 规范里，以及参考手册的寄存器中规定的那 4 位地址域）、传输方向、传输类型、最大包长等。以 STSW-STM32121 库中的 Mass_Storage 例程为例，需要把<usb_desc.c>中的端点描述符做如下修改：0x0A 地址的端点作为 IN 端点（PC 从它读取数据），0x0B 地址的端点作为 OUT 端点（PC 向它写数据）。

```
const uint8_t MASS_ConfigDescriptor[MASS_SIZ_CONFIG_DESC] =
  {
    。。。。。。
    /* 18 */
    0x07,   /*Endpoint descriptor length = 7*/
    0x05,   /*Endpoint descriptor type */
    0x8A,   //0x81,   /*Endpoint address (IN, address 1) */
    0x02,   /*Bulk endpoint type */
    0x40,   /*Maximum packet size (64 bytes) */
    0x00,
    0x00,   /*Polling interval in milliseconds */
    /* 25 */
    0x07,   /*Endpoint descriptor length = 7 */
    0x05,   /*Endpoint descriptor type */
    0x0B,   //0x02,   /*Endpoint address (OUT, address 2) */
    0x02,   /*Bulk endpoint type */
    0x40,   /*Maximum packet size (64 bytes) */
    0x00,
    0x00     /*Polling interval in milliseconds*/
    /*32*/
  };
```

接下来就是考虑使用 STM32F103 USB 模块提供的 8 个双向端点的哪个端点了。我们刚才从参考手册关于寄存器描述的截图中看到，每一对端点具有相同的地址。在库函数里，对端点寄存器的地址位域的操作在这里：

```
void SetDeviceAddress(uint8_t Val)
{
  uint32_t i;
  uint32_t nEP = Device_Table.Total_Endpoint;

  /* set address in every used endpoint */
  for (i = 0; i < nEP; i++)
  {
    _SetEPAddress((uint8_t)i, (uint8_t)i);
  } /* for */
  _SetDADDR(Val | DADDR_EF); /* set device address and enable function */
}
```

这个函数的名称是“设置(USB)设备地址”，但是其中除了最后一句是在设置 USB 设备的地址，前面的 for 循环是在设置该设备内的端点地址。

从以上绿色标注的代码段可以看到，库代码固定给 1 号端点”0x01”这个地址，2 号端点”0x02”这个地址，以此类推。这里的”1 号”、”2 号”指的是端点的编号，对应的就是之前提到的 8 个寄存器的编号，即下图中的 n=0~7。n 在这里就是端点的编号。

USB endpoint n register (USB_EPnR), n=[0..7]

Address offset: 0x00 to 0x1C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTR_RX	DTOG_RX	STAT_RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR_TX	DTOG_TX	STAT_TX[1:0]		EA[3:0]			
rc_w0	t	t	t	r	rw	rw	rw	rc_w0	t	t	t	rw	rw	rw	rw

那么在这个应用中，需要用到地址为 0x0A 和 0x0B 的两个端点，但是端点编号最多只能到 7，因此需要修改库代码中关于端点地址设置的地方如下：这里，我们使用编号为 1 和 2 的两个端点。

- ➔ 为啥不用编号 0？因为编号 0 默认给双向 0 端点，即用于控制传输的 0 端点。
- ➔ 为啥用 2 个编号？因为这里需要 2 个不同的端点地址，必须用 2 个编号。一个编号对应的 2 个端点必须共享同样的端点地址。

```
void SetDeviceAddress(uint8_t Val)
{
  /* set address in every used endpoint */
  _SetEPAddress((uint8_t)0, (uint8_t)0);
  _SetEPAddress((uint8_t)1, (uint8_t)0x0A); // 1 号端点是 0x8A,即地址为 A 的 IN 端点
  _SetEPAddress((uint8_t)2, (uint8_t)0x0B); // 2 号端点是 0x0B,即地址位 B 的 OUT 端点

  _SetDADDR(Val | DADDR_EF); /* set device address and enable function */
}
```

既然这里指定了使用编号 1 和编号 2 的端点，那么需要在<usb_conf.h>中设置这两个端点的硬件收发缓冲区地址

```
/* EP0  */
/* rx/tx buffer base address */
#define ENDP0_RXADDR        (0x18)
#define ENDP0_TXADDR        (0x58)

/* 1 号端点，IN 端点，发送缓冲区如下  */
#define ENDP1_TXADDR        (0x98)
/* 2 号端点，OUT 端点，接收缓冲区如下 */
#define ENDP2_RXADDR        (0xD8)
```

当然如果你很任性，一定要使用编号为 6 和 7 的端点，也可以，那么代码就如下修改：

```
void SetDeviceAddress(uint8_t Val)
{
  /* set address in every used endpoint */
```

```
  _SetEPAddress((uint8_t)0, (uint8_t)0);
  _SetEPAddress((uint8_t)6, (uint8_t)0x0A);
  _SetEPAddress((uint8_t)7, (uint8_t)0x0B);

  _SetDADDR(Val | DADDR_EF); /* set device address and enable function */
}
```

相应地，需要在<usb_conf.h>中指明编号为 6 和 7 的这两个端点的硬件收发缓冲区地址。那么如法炮制做如下修改，**就可以了吗？就可以了吗？就可以了吗？**

```
/* EP0  */
/* rx/tx buffer base address */
#define ENDP0_RXADDR        (0x18)
#define ENDP0_TXADDR        (0x58)

/* 6 号端点，IN 端点，发送缓冲区如下  */
#define ENDP6_TXADDR        (0x98)
/* 7 号端点，OUT 端点，接收缓冲区如下 */
#define ENDP7_RXADDR        (0xD8)
```

答案是否定的！以下的代码才 OK。欲知详情，请参考下一条应用技巧《STM32F103 上 USB 模块的包缓冲区详解》

```
/* EP0  */
/* rx/tx buffer base address */
#define ENDP0_RXADDR        (0x40)
#define ENDP0_TXADDR        (0x80)

/* 6 号端点，IN 端点，发送缓冲区如下  */
#define ENDP6_TXADDR        (0xC0)
/* 7 号端点，OUT 端点，接收缓冲区如下 */
#define ENDP7_RXADDR        (0x100)
```

重要通知 – 请仔细阅读

意法半导体公司及其子公司（“ST”）保留随时对ST 产品和/ 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于ST 产品的最新信息。ST 产品的销售依照订单确认时的相关ST 销售条款。

买方自行负责对ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的ST 产品如有不同于此处提供的信息的规定，将导致ST 针对该产品授予的任何保证失效。

ST 和ST 徽标是ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。