1．包格式

包的概念在前面已经介绍了，包是帧的基本成分。常用的包有令牌包、数据包和握手包。对于高速传输，还定义了事务分割专用令牌包(事务分割开始令牌包和事务分割完成令牌包)。

1)令牌包格式

在 USB 系统中，所有的通信都是由主机发出相应的令牌所引起的。令牌包格式图 10．33 所示。 其中 PID 为包标识，ADDR 为设备地址，ENDP 为端点号，CRC5 是对 ADDR 和 ENDP 域进行校验的 5 位 CRC 校验码，校验多项式为：G(X)=X5+X2+1。

2)数据包格式

数据包用于主机与设备之间的数据传输。数据包格式如图 10．34 所示。 其中 PID 为包标识，DATA 为数据位，最多为 8 192 个位，DATA 应是字节的整数倍。CRCl6 是对 DATA 域进行校验的 16 位 CRC 校验码，校验多项式为：G(X)=X16+X15+X2+1.

www.juqingshow.com

(LSB) (MSB)

域	PID	ADDR	ENDP	CRC5
位数	8	7	4	5

图 10.33 令牌包的格式

(LSB) (MSB)

域	PID	DATA	CRC16
位数	8	0~8192	16

图 10.34 数据包的格式

3)握手包格式

握手包用来指示数据被成功接收、命令被接收或被拒绝等事务状态。握手包格式如图 10.35 所示。 握手包仅由 PID)组成。有四种常用握手包(ACK、NAK、STALL 和 NYET)和一个专用握手包，握手包的类型是通过 PID 的编码来体现的。

·ACK 包表示接收器已成功接收数据。

·NAK 包表示接收设备不能接收数据或发送设备不能发送数据。

·STALL 包表示端点已终止或不支持控制管道请求。

·NYET 包表示接收器还没有任何响应。

4)帧开始包 SOF(Start Of Frame)

在全速或低速时，主机每隔 1 ms±0．0005 ms 发出一个帧开始包 SOF，在高速时，每隔 125 μs±0．0625μs 发出一个 SOF，以表示开始一个新帧。SOF 包的格式如图 10．36 所示。

www.juqingshow.com

(LSB) (MSB)

域	PID
位数	8

图 10.35 握手包的格式

www.juqingshow.com

(LSB) (MSB)

域	PID	Frame Number	CRC5
位数	8	11	5

图 10.36 SOF 包的格式

注：令牌包：标示下一个数据包的传输方向，标示下一个数据包的收/发地址。

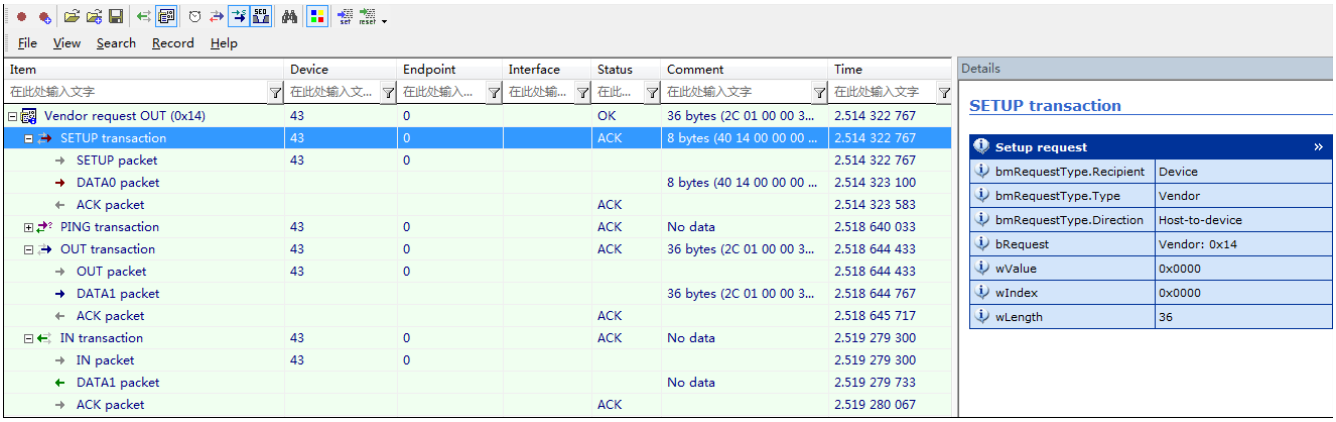
令牌包请求协议:由硬件 phy 去解析

USB 软件请求协议(标准 USB 描述符请求，用户自定义请求):由控制器驱动解析

struct dwc2_ep *dep //usb 端点描述符
struct dwc2_request *request //usb 请求描述符
//俩者之间的关系如下：
1.request = next_request(&dep->urb_list);
2.request->dep = dep; list_add_tail(&request->queue,&dep->urb_list)
下面以烧录工具发送 args 数据过程为例做 uboot 中 usb 驱动架构和 usb 控制器驱动分析：
（控制传输）

```
int status = usb_control_msg(
                                handle,
          /* bmRequestType */ USB_ENDPOINT_OUT|USB_TYPE_VENDOR|USB_RECIP_DEVICE,
          /* bRequest      */ VR_UPDATE_CFG,
          /* wValue        */ 0,
          /* wIndex        */ 0,
          /* Data          */ (char *)&cmd,
          /* wLength       */ sizeof(union cmd),
                                5000);
```

以下就是上面这个函数所导致的 usb 传输流程：



send args length

HOST **DEVICE**

SETUP_package
DATA0_package setup stage complete
ACK_package----------> DEP_XFER_COMP(什么也没做)
----------> DEP_SETUP_PHASE_DONE
\|/
parse_setup
\|/
dev->driver->setup(composite_setup)
\|/
f_cloner_setup_handle
\|/
用 handle_cmd 作 complete 构造 usb_request(即 ep0req),调用 usb_ep_queue 将 ep0 与 ep0req 绑定
这里就是“建立阶段”做的目的工作（就是将”数据阶段”的端点描述符与请求描述符绑定，让“数据阶段”能调用请求描述符中的方法去处理”数据阶段“接收的数据）

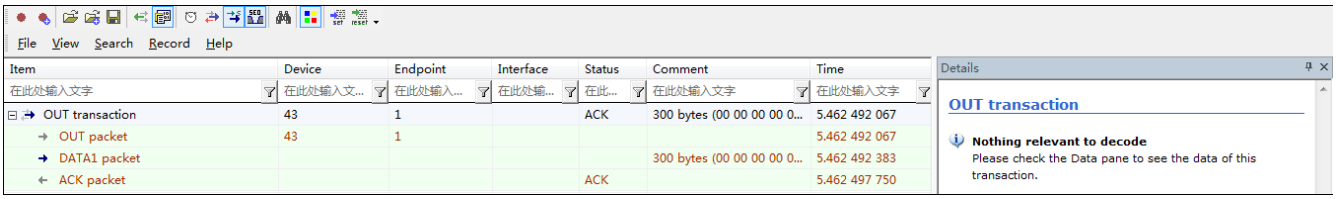
OUT_package
DATA1_package　　data stage complete
ACK_package----------->DEP_XFER_COMP
　　\|/
outep0_transfer_complete
　　\|/
dwc2_giveback_urb
　　\|/
request->req.complete(**handle_cmd**)
　　\|/
VR_UPDATE_CFG
cloner->args_req->length = cmd->update.length;
(这个 cmd 就是数据阶段接收到的数据,与烧录工具端结合着看这个 cmd)
usb_ep_queue(cloner->ep_out, cloner->args_req,0)
（数据阶段完成后，又为接下来 BULK 传输绑定了数据处理函数，即 cloner->args_req.complete）

IN_package
DATA1_package　　status stage complete
ACK_package----------->DEP_XFER_COMP

（BULK 传输）

```
status = usb_bulk_write(
                        handle,
    /* endpoint        */ INGENIC_OUT_ENDPOINT,
    /* *data           */ (char *)data,
    /* length          */ length,
    /* timeout         */ 5000 * factor);
```

以下就是上面这个函数所导致的 usb 传输流程：



send args