# UVC简介

* Each video function has a single VideoControl (VC) interface and can have several VideoStreaming (VS) interfaces

每个视频有且仅有1个VideoControl (VC)接口和可有多个 VideoStreaming (VS) 接口

* The **VideoControl** (VC) interface is used to access the device controls of the function whereas

the **VideoStreaming** (VS) interfaces are used to transport data streams into and out of the function.

VC接口用于设备功能控制,VS接口用于传输数据流进出

* Video Interface Class Code(A.1 P171)

视频接口类代码就是宏定义的USB\_CLASS\_VIDEO

总共有3种子类subclass

1.VideoControl Interface 视频控制接口子类

2.VideoStreaming Interface 视频数据流接口子类

3.Video Interface Collection 视频接口集合子类

* Units provide the basic building blocks to fully describe most video functions ,A Unit has one or more Input Pins and a single Output Pin,

Unit提供了基础模块来全面描述大部分的视频功能,一个Unit可以由一个或多个输入引脚和仅一个输出引脚(这里的每一个pin代表一个逻辑上的数据流)

Unit可以通过pin引脚连接在一起,一个输出pin可以连接多个输入pin,但一个输入pin只能连接一个输出pin

* An **Input Terminal** (IT) is an entity that represents a starting point for data streams inside the video function.

一个输入Terminal (IT)终端是一个实体代表数据流的开始端点

* An **Output Terminal** (OT) represents an ending point for data streams.

一个输出Terminal (OT)终端是一个实体代表数据流的结束端点

* Terminals have one Input or Output Pin that is always numbered one.

Terminal只有1个输入或一个输出引脚pin

* The **Camera Terminal** (CT) controls mechanical (or equivalent digital) features of the device component that transmits the video stream.

摄像头Terminal (CT)控制传输视频流的设备组件特性(Scanning Mode扫描模式 Auto-Exposure Mode自动曝光模式 Auto-Exposure Priority自动曝光优先级Exposure Time 曝光时间 Focus聚焦 Auto-Focus自动聚焦 Simple Focus简单聚焦 Iris红外 Zoom放大 Pan摇动 Roll滚动 Tilt倾斜 Digital Windowing数字窗口Region of Interest 感应区)

* The **Selector Unit** (SU) selects from n input data streams and routes them unaltered to the single output stream.

选择Unit (SU)选择多个输入数据流并路由它们到单一的输出流

* The **Processing Unit** (PU) controls image attributes of the video being streamed through it.

处理Unit (PU)控制流经它的视频流图像属性(【Brightness背光 Hue色度 Saturation饱和度 Sharpness锐度 Gamma伽马值 Digital Multiplier (Zoom)数字放大】

【White Balance Temperature白平衡色温 White Balance Component白平衡组件 Backlight Compensation背光补偿 Contrast对比度】

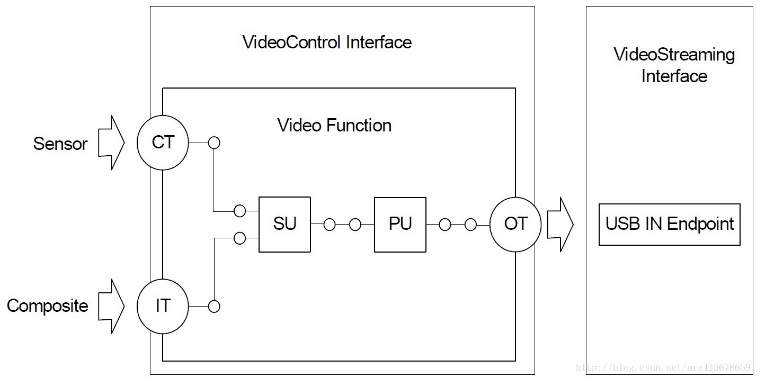
【Gain增益 Power Line Frequency电源线频率 Analog Video Standard模拟视频标准 Analog Video Lock Status模拟视频锁存状态】)

* The **Encoding Unit** controls attributes of the encoder that encodes the video being streamed through it.

编码Unit (EU)控制流过的视频流编码的编码器的属性(...)

* The **Extension Unit** (XU) is the method provided by this specification to add vendor-specific building blocks to the specification

扩展Unit (XU)提供厂商特殊控制模块方法



# USB描述符

## USB设备的插入检测机制

USB主机如何检测到设备插入的那？首先在USB集线器的每个下游端口的D+和D-上，分别接上一个15K的下拉电阻，而在USB设备端，在D+或者D-上接上1.5K的上拉电阻，高速设备接在D+上，低俗设备上拉接在D-上。这样当有设备插入到集线器时，就将差分数据线上的一条拉高了，集线器检测到这个状态后，它就报告给USB主控制器，这样就检测到设备插入了。

## USB设备的枚举过程

usb主机检测到USB设备插入后，就要对设备进行枚举了。枚举的作用就是从设备是那个读取一些信息，知道设备是什么样的设备，如果通信，这样主机就可以根据这些信息假造合适的驱动程序。调试USB设备，很重要的一点就是USB枚举过程，只要枚举成功了，那就成功一大半了。

USB的一种传输模式---控制传输

这种传输在USB中是非常重要的，它保证数据的正确性，在设备的枚举过程中都是控制传输。

控制传输分为三个过程：1.建立过程；2可选的数据过程；3状态过程。

建立过程都是由USB主机发起，它开始于一个Setup令牌包，后面紧跟着一个DATA0包，如果是控制输入传输，那么数据过程就是输入数据；如果控制输出传输，那么数据过程就是输出数据。数据过程之后是状态过程。状态过程刚好与数据过程的数据传输放喜爱那个相反。

首先：主机检测到USB设备插入后，就会先对设备进行复位，复位后，USB主机就会对地址为0的设备发送获取设备描述符的标准请求。所有的USB设备在总线复位后其地址都为0，这样主机就可以跟那些刚刚插入的设备通过地址0通信，获取玩设备描述符后，主机就会获取配置描述符9个字节， 主机获取到配置描述符后，根据里面的配置集合总长度，在获取配置结合。配置集合包括配置描述符，接口描述符，端点描述符等。

## USB描述符

### 设备描述符：

一个设备只有一个设备描述符

typedef struct \_USB\_DEVICE\_DESCRIPTOR\_

{

BYTE bLength, //描述符大小．固定为0x12．

BYTE bDescriptorType, //设备描述符类型．固定为0x01

WORD bcdUSB,

BYTE bDeviceClass,

BTYE bDeviceSubClass,

BYTE bDeviceProtol,

BYTE bMaxPacketSize0,

WORD idVenderI,

WORD idProduct,

WORD bcdDevice,

BYTE iManufacturer,

BYTE iProduct,

BYTE iSerialNumber,

BYTE iNumConfiguations

}USB\_DEVICE\_DESCRIPTOR;

bcdUSB : USB 规范发布号．表示了本设备能适用于那种协议，如2.0=0200，1.1=0110等．

bDeviceClass : 类型代码（由USB指定）。当它的值是0时，表示所有接口在配置描述符里，并且所有接口是独立的。当它的值是1到FEH时，表示不同的接口关联的。当它的值是FFH时，它是厂商自己定义的．

bDeviceSubClass : 子类型代码（由USB分配）．如果bDeviceClass值是0，一定要设置为0．其它情况就跟据USB-IF组织定义的编码．

bDeviceProtocol : 协议代码（由USB分配）．如果使用USB-IF组织定义的协议，就需要设置这里的值，否则直接设置为0。如果厂商自己定义的可以设置为FFH．

bMaxPacketSize0 : 端点０最大分组大小（只有8,16,32,64有效）．

idVendor : 供应商ID（由USB分配）．

idProduct : 产品ID（由厂商分配）．由供应商ID和产品ID，就可以让操作系统加载不同的驱动程序．

bcdDevice : 设备出产编码．由厂家自行设置．

iManufacturer : 厂商描述符字符串索引．索引到对应的字符串描述符． 为０则表示没有．

iProduct : :产品描述符字符串索引．同上．

iSerialNumber : 设备序列号字符串索引．同上．

bNumConfigurations : 可能的配置数．指配置字符串的个数

### 配置描述符：

配置描述符定义了设备的配置信息，一个设备可以有多个配置描述符

typedef struct \_USB\_CONFIGURATION\_DESCRIPTOR\_

{

BYTE bLength, //描述符大小．固定为0x09．

BYTE bDescriptorType, //配置描述符类型．固定为0x02

WORD wTotalLength,

BYTE bNumInterfaces,

BYTE bConfigurationValue,

BYTE iConfiguration,

BYTE bmAttributes,

BYTE MaxPower

}USB\_CONFIGURATION\_DESCRIPTOR;

wTotalLength : 返回整个数据的长度．指此配置返回的配置描述符，接口描述符以及端点描述符的全部大小．

bNumInterfaces : 配置所支持的接口数．指该配置配备的接口数量，也表示该配置下接口描述符数量．

bConfigurationValue : 作为Set Configuration的一个参数选择配置值．

iConfiguration : 用于描述该配置字符串描述符的索引．

bmAttributes : 供电模式选择．Bit4-0保留，D7:总线供电，D6:自供电，D5:远程唤醒．

MaxPower : 总线供电的USB设备的最大消耗电流．以2mA为单位．

### 接口描述符：

接口描述符说明了接口所提供的配置，一个配置所拥有的接口数量通过配置描述符的bNumInterfaces决定

typedef struct \_USB\_INTERFACE\_DESCRIPTOR\_

{

BYTE bLength, //描述符大小．固定为0x09

BYTE bDescriptorType, //接口描述符类型．固定为0x04

BYTE bInterfaceNumber,

BYTE bAlternateSetting,

BYTE bNumEndpoint,

BYTE bInterfaceClass,

BYTE bInterfaceSubClass,

BYTE bInterfaceProtocol,

BYTE iInterface

}USB\_INTERFACE\_DESCRIPTOR;

bInterfaceNumber: 该接口的编号．

bAlternateSetting : 用于为上一个字段选择可供替换的位置．即备用的接口描述符标号．

bNumEndpoint : 使用的端点数目．端点０除外．

bInterfaceClass : 类型代码（由USB分配）．

bInterfaceSubClass : 子类型代码（由USB分配）．

bInterfaceProtocol : 协议代码（由USB分配）．

iInterface : 字符串描述符的索引

### 端点描述符：

USB设备中的每个端点都有自己的端点描述符，由接口描述符中的bNumEndpoint决定其数量

typedef struct \_USB\_ENDPOINT\_DESCRIPTOR\_

{

BYTE bLength, //描述符大小．固定为0x07

BYTE bDescriptorType, //接口描述符类型．固定为0x05

BYTE bEndpointAddress,

BYTE bmAttributes,

WORD wMaxPacketSize,

BYTE bInterval

}USB\_ENDPOINT\_DESCRIPTOR;

bEndpointType : USB设备的端点地址．Bit7，方向，对于控制端点可以忽略，1/0:IN/OUT．Bit6-4，保留．BIt3-0：端点号．

bmAttributes : 端点属性．Bit7-2，保留．BIt1-0：00控制，01同步，02批量，03中断．

wMaxPacketSize : 本端点接收或发送的最大信息包大小．

bInterval : 轮训数据传送端点的时间间隔．对于批量传送和控制传送的端点忽略．对于同步传送的端点，必须为１，对于中断传送的端点，范围为１－２５５．

### 字符串描述符：

其中字符串描述符是可选的．如果不支持字符串描述符，其设备，配置，接口描述符内的所有字符串描述符索引都必须为０

typedef struct \_USB\_STRING\_DESCRIPTION\_

{

BYTE bLength,

BYTE bDescriptionType, //接口描述符类型．固定为0x03．

BYTE bString[1];

}USB\_STRING\_DESCRIPTION;

bLength : 描述符大小．由整个字符串的长度加上bLength和bDescriptorType的长度决定．

bString[1] : Unicode编码字符串．

## USB组合设备

USB复合设备(USB Compound Device) Compound Device内嵌Hub和多个Function，每个Function都相当于一个独立的USB外设，有自己的PID/VID。

USB组合设备(USB Composite Device)Composite Device内只有一个Function，只有一套PID/VID，通过将不同的interface定义为不同的类来实现多个功能的组合。

typedef struct \_USBInterfaceAssociationDescriptor {

BYTE bLength: 0x08 //描述符大小

**BYTE bDescriptorType: 0x0B //IAD描述符类型**

BYTE bFirstInterface: 0x00 //起始接口

BYTE bInterfaceCount: 0x02 //接口数

BYTE bFunctionClass: 0x0E //类型代码

BYTE bFunctionSubClass: 0x03 //子类型代码

BYTE bFunctionProtocol: 0x00 //协议代码

BYTE iFunction: 0x04 //描述字符串索引

}

Interface Association: ==============================USB复合设备

bLength 8

**bDescriptorType 11**  //IDA接口描述符

bFirstInterface 0 //起始接口

bInterfaceCount 2 //接口数

**bFunctionClass 14 Video**  //代码类型

bFunctionSubClass 3 Video Interface Collection //子代码类型

bFunctionProtocol 0 //协议代码

iFunction 5 Kingcome FHD Camera //描述字符串索引

Interface Association:

bLength 8

**bDescriptorType 11**

bFirstInterface 2

bInterfaceCount 2

**bFunctionClass 1 Audio**

bFunctionSubClass 2 Streaming

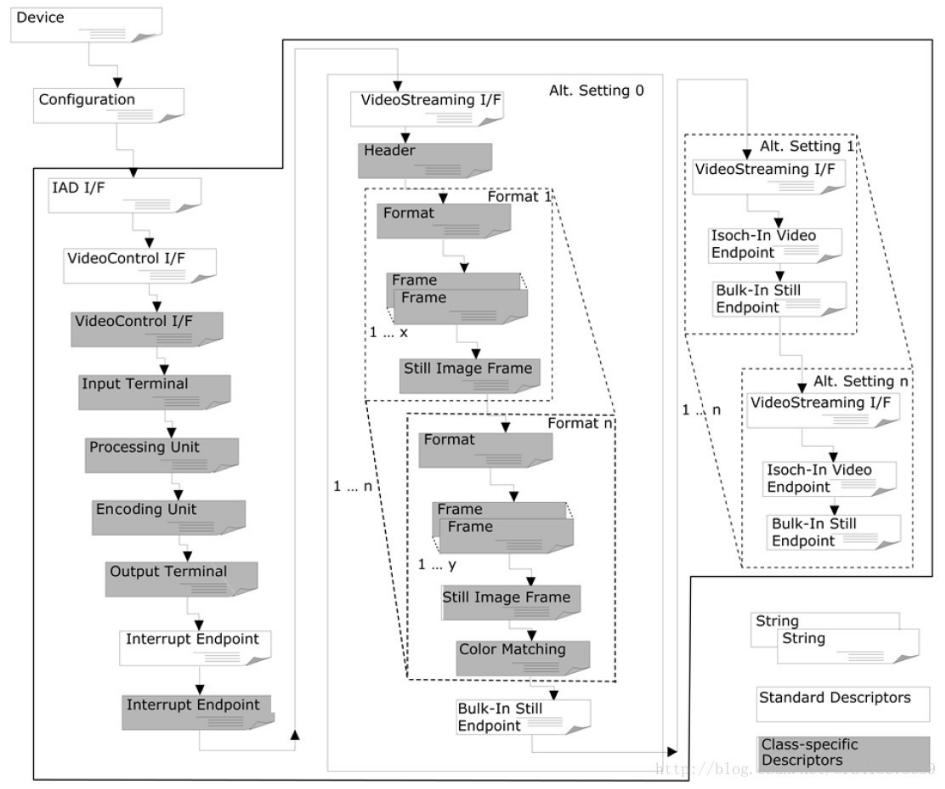
bFunctionProtocol 0

iFunction 6 Realtek USB2.0 MIC

## lsusb -d 2b7e:220f –v

可以通过Video Control描述符布局,分析出摄像头框架: IT(1)->PU(2)->XU(4)->OT(3)

可以通过Audio Control描述符布局,分析出MIC设备框架: **IT(1)->FU(3)->OT(2)**



### Device Descriptor

**bDeviceProtocol** =1，表示USB为组合设备，组合设备的详细描述参见**Interface Association**

**bNumConfigurations=1，** 有1个Configuration Descriptor

Bus 001 Device 004: ID 2b7e:220f

Device Descriptor: =================Device

bLength 18

**bDescriptorType 1**  //设备描述符

bcdUSB 2.00

bDeviceClass 239 Miscellaneous Device

bDeviceSubClass 2 ?

**bDeviceProtocol 1 Interface Association** //USB组合设备: Video & Audio

bMaxPacketSize0 64

**idVendor 0x2b7e**

**idProduct 0x220f**

bcdDevice 10.3a

iManufacturer 3 Kingcome

iProduct 1 KNC\_220F

iSerial 2 20170227

**bNumConfigurations 1**  //Configuration个数

### Configuration Descriptor

**bNumInterfaces=4,** USB设备一共有4个Interface Descriptor

Configuration Descriptor: =================Configuration:1

{

bLength 9

**bDescriptorType 2**  //配置描述符

wTotalLength 1331

**bNumInterfaces 4**  //Interface个数

bConfigurationValue 1 //作为Set Configuration的一个参数选择配置值

iConfiguration 4 USB Camera //用于描述该配置字符串描述符的索引

bmAttributes 0x80 //供电模式选择

(Bus Powered)

MaxPower 500mA

}

### Interface Association(IDA)

Video占用2个Interface, 第一个interface序号为0

Audio占用2个Interface, 第一个interface序号为2

Interface Association: ==============================USB复合设备: Video

{

bLength 8

**bDescriptorType 11**  //IDA接口描述符

**bFirstInterface 0 //起始接口**

**bInterfaceCount 2 //接口数**

**bFunctionClass 14 Video**  //代码类型

bFunctionSubClass 3 Video Interface Collection //子代码类型

bFunctionProtocol 0 //协议代码

iFunction 5 Kingcome FHD Camera //描述字符串索引

}

Interface Association: ==============================USB复合设备: Audio

{

bLength 8

**bDescriptorType 11**

**bFirstInterface 2**

**bInterfaceCount 2**

**bFunctionClass 1 Audio**

bFunctionSubClass 2 Streaming

bFunctionProtocol 0

iFunction 6 Realtek USB2.0 MIC

}

### Interface Descriptor

Interface 0: video control

Interface 1: video streaming

Interface 0: audio control

Interface 0: audio stream

Interface Descriptor:{

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 0**

bAlternateSetting 0 //备用的接口描述符标号

bNumEndpoints 1

**bInterfaceClass 14 Video**

**bInterfaceSubClass 1 Video Control**

bInterfaceProtocol 0

iInterface 5 Kingcome FHD Camera

}

Interface Descriptor: {

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 1**

bAlternateSetting 0

bNumEndpoints 0

**bInterfaceClass 14 Video**

**bInterfaceSubClass 2 Video Streaming**

bInterfaceProtocol 0

iInterface 0

}

Interface Descriptor:{

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 2**

bAlternateSetting 0

bNumEndpoints 0

**bInterfaceClass 1 Audio**

**bInterfaceSubClass 1 Control Device**

bInterfaceProtocol 0

iInterface 6 Realtek USB2.0 MIC

}

Interface Descriptor:{

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 3**

bAlternateSetting 0

bNumEndpoints 0

**bInterfaceClass 1 Audio**

**bInterfaceSubClass 2 Streaming**

bInterfaceProtocol 0

iInterface 0

}

#### Interface Descriptor 0: (bAlternateSetting 0) Video Control

可以通过Video Control描述符布局,分析出摄像头框架: IT(1)->PU(2)->XU(4)->OT(3)

Interface Descriptor:{

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 0**

bAlternateSetting 0

**bNumEndpoints 1**

**bInterfaceClass 14 Video**

**bInterfaceSubClass 1 Video Control**

bInterfaceProtocol 0

iInterface 5 Kingcome FHD Camera

}

##### VideoControl: HEADER(freg.=15Hz)

VideoControl Interface Descriptor:{

bLength 13

**bDescriptorType 36**

**bDescriptorSubtype 1 (HEADER)**

bcdUVC 1.00

wTotalLength 78

**dwClockFrequency 15.000000MHz**

bInCollection 1

baInterfaceNr( 0) 1

}

##### VideoControl: INPUT\_TERMINAL(bTerminalID=1)

VideoControl Interface Descriptor:{

bLength 18

**bDescriptorType 36**

**bDescriptorSubtype 2 (INPUT\_TERMINAL)**

**bTerminalID 1**

**wTerminalType 0x0201 Camera Sensor**

bAssocTerminal 0

iTerminal 0

wObjectiveFocalLengthMin 0

wObjectiveFocalLengthMax 0

wOcularFocalLength 0

bControlSize 3

**bmControls 0x0000000e**

Auto-Exposure Mode

Auto-Exposure Priority

Exposure Time (Absolute)

}

##### VideoControl: PROCESSING\_UNIT(bSourceID=1，bUnitID=2)

VideoControl Interface Descriptor:{

bLength 11

**bDescriptorType 36**

**bDescriptorSubtype 5 (PROCESSING\_UNIT)**

Warning: Descriptor too short

**bUnitID 2**

**bSourceID 1**

wMaxMultiplier 0

bControlSize 2

**bmControls 0x0000157f**

Brightness

Contrast

Hue

Saturation

Sharpness

Gamma

White Balance Temperature

Backlight Compensation

Power Line Frequency

White Balance Temperature, Auto

iProcessing 0

bmVideoStandards 0x 9

None

SECAM - 625/50

}

##### VideoControl: OUTPUT\_TERMINAL(bSourceID=4, bTerminalID=3)

VideoControl Interface Descriptor:{

bLength 9

**bDescriptorType 36**

**bDescriptorSubtype 3 (OUTPUT\_TERMINAL)**

**bTerminalID 3**

**wTerminalType 0x0101 USB Streaming**

bAssocTerminal 0

**bSourceID 4**

iTerminal 0

}

##### VideoControl: EXTENSION\_UNIT(baSourceID( 0) =2, bUnitID=4)

VideoControl Interface Descriptor:{

bLength 27

**bDescriptorType 36**

**bDescriptorSubtype 6 (EXTENSION\_UNIT)**

**bUnitID 4**

guidExtensionCode {8ca72912-b447-9440-b0ce-db07386fb938}

bNumControl 2

bNrPins 1

**baSourceID( 0) 2**

bControlSize 2

bmControls( 0) 0x00

bmControls( 1) 0x06

iExtension 0

}

##### Endpoint(Ep: addr=0x83, Interrupt, 1x16B)

Endpoint Descriptor:{

bLength 7

**bDescriptorType 5**  //端点描述符

**bEndpointAddress 0x83** EP 3 IN

bmAttributes 3

**Transfer Type Interrupt**

Synch Type None

Usage Type Data

**wMaxPacketSize 0x0010 1x 16 bytes**

bInterval 6

}

#### Interface Descriptor 1: (bAlternateSetting 0)

Interface Descriptor: {

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 1**

**bAlternateSetting 0**

**bNumEndpoints 0**

bInterfaceClass 14 Video

bInterfaceSubClass 2 Video Streaming

bInterfaceProtocol 0

iInterface 0

}

##### VideoStreaming: INPUT\_HEADER

bNumFormats =2： 表示有2种format 【FORMAT\_MJPEG, FORMAT\_UNCOMPRESSED】

VideoStreaming Interface Descriptor:{

bLength 15

**bDescriptorType 36**

**bDescriptorSubtype 1 (INPUT\_HEADER)**

**bNumFormats 2**

wTotalLength 685

**bEndPointAddress 129**

bmInfo 0

bTerminalLink 3

bStillCaptureMethod 2

bTriggerSupport 1

bTriggerUsage 0

bControlSize 1

bmaControls( 0) 11

bmaControls( 1) 11

}

##### VideoStreaming: FORMAT\_MJPEG

**bNumFrameDescriptors** =10： 当前format有10种分辨率 【FRAME\_MJPEG】

VideoStreaming Interface Descriptor:{

bLength 11

**bDescriptorType 36**

**bDescriptorSubtype 6 (FORMAT\_MJPEG)**

**bFormatIndex 1**

**bNumFrameDescriptors 10**

bFlags 1

Fixed-size samples: Yes

**bDefaultFrameIndex 1**

bAspectRatioX 0

bAspectRatioY 0

bmInterlaceFlags 0x00

Interlaced stream or variable: No

Fields per frame: 1 fields

Field 1 first: No

Field pattern: Field 1 only

bCopyProtect 0

}

###### VideoStreaming: FRAME\_MJPEG(1920x1080)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 7 (FRAME\_MJPEG)**

**bFrameIndex 1**

bmCapabilities 0x00

Still image unsupported

**wWidth 1920**

**wHeight 1080**

dwMinBitRate 995328000

dwMaxBitRate 995328000

dwMaxVideoFrameBufferSize 4147200

dwDefaultFrameInterval 333333

bFrameIntervalType 1

dwFrameInterval( 0) 333333

}

###### VideoStreaming: FRAME\_MJPEG(352x288)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 7 (FRAME\_MJPEG)**

**bFrameIndex 2**

bmCapabilities 0x00

Still image unsupported

**wWidth 352**

**wHeight 288**

dwMinBitRate 48660480

dwMaxBitRate 48660480

dwMaxVideoFrameBufferSize 202752

dwDefaultFrameInterval 333333

bFrameIntervalType 1

dwFrameInterval( 0) 333333

}

###### VideoStreaming: FRAME\_MJPEG(320x240)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 7 (FRAME\_MJPEG)**

**bFrameIndex 3**

bmCapabilities 0x00

Still image unsupported

**wWidth 320**

**wHeight 240**

dwMinBitRate 36864000

dwMaxBitRate 36864000

dwMaxVideoFrameBufferSize 153600

dwDefaultFrameInterval 333333

bFrameIntervalType 1

dwFrameInterval( 0) 333333

}

###### VideoStreaming: FRAME\_MJPEG(176x144)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 7 (FRAME\_MJPEG)**

**bFrameIndex 4**

bmCapabilities 0x00

Still image unsupported

**wWidth 176**

**wHeight 144**

dwMinBitRate 12165120

dwMaxBitRate 12165120

dwMaxVideoFrameBufferSize 50688

dwDefaultFrameInterval 333333

bFrameIntervalType 1

dwFrameInterval( 0) 333333

}

###### VideoStreaming: FRAME\_MJPEG(160x120)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 7 (FRAME\_MJPEG)**

**bFrameIndex 5**

bmCapabilities 0x00

Still image unsupported

**wWidth 160**

**wHeight 120**

dwMinBitRate 9216000

dwMaxBitRate 9216000

dwMaxVideoFrameBufferSize 38400

dwDefaultFrameInterval 333333

bFrameIntervalType 1

dwFrameInterval( 0) 333333

}

###### VideoStreaming: FRAME\_MJPEG(640x480)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 7 (FRAME\_MJPEG)**

**bFrameIndex 6**

bmCapabilities 0x00

Still image unsupported

**wWidth 640**

**wHeight 480**

dwMinBitRate 147456000

dwMaxBitRate 147456000

dwMaxVideoFrameBufferSize 614400

dwDefaultFrameInterval 333333

bFrameIntervalType 1

dwFrameInterval( 0) 333333

}

###### VideoStreaming: FRAME\_MJPEG(800x600)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 7 (FRAME\_MJPEG)**

**bFrameIndex 7**

bmCapabilities 0x00

Still image unsupported

**wWidth 800**

**wHeight 600**

dwMinBitRate 230400000

dwMaxBitRate 230400000

dwMaxVideoFrameBufferSize 960000

dwDefaultFrameInterval 333333

bFrameIntervalType 1

dwFrameInterval( 0) 333333

}

###### VideoStreaming: FRAME\_MJPEG(1280x720)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 7 (FRAME\_MJPEG)**

**bFrameIndex 8**

bmCapabilities 0x00

Still image unsupported

**wWidth 1280**

**wHeight 720**

dwMinBitRate 442368000

dwMaxBitRate 442368000

dwMaxVideoFrameBufferSize 1843200

dwDefaultFrameInterval 333333

bFrameIntervalType 1

dwFrameInterval( 0) 333333

}

###### VideoStreaming: FRAME\_MJPEG(1280x1024)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 7 (FRAME\_MJPEG)**

**bFrameIndex 9**

bmCapabilities 0x00

Still image unsupported

**wWidth 1280**

**wHeight 1024**

dwMinBitRate 629145600

dwMaxBitRate 629145600

dwMaxVideoFrameBufferSize 2621440

dwDefaultFrameInterval 333333

bFrameIntervalType 1

dwFrameInterval( 0) 333333

}

###### VideoStreaming: FRAME\_MJPEG(1920x1080)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 7 (FRAME\_MJPEG)**

**bFrameIndex 10**

bmCapabilities 0x00

Still image unsupported

**wWidth 1920**

**wHeight 1080**

dwMinBitRate 995328000

dwMaxBitRate 995328000

dwMaxVideoFrameBufferSize 4147200

dwDefaultFrameInterval 333333

bFrameIntervalType 1

dwFrameInterval( 0) 333333

}

###### VideoStreaming: STILL\_IMAGE\_FRAME (静态图像帧)

VideoStreaming Interface Descriptor:{

bLength 10

**bDescriptorType 36**

**bDescriptorSubtype 3 (STILL\_IMAGE\_FRAME)**

bEndpointAddress 0

bNumImageSizePatterns 1

**wWidth( 0) 1920**

**wHeight( 0) 1080**

bNumCompressionPatterns 1

}

###### VideoStreaming: COLORFORMAT(颜色格式帧)

VideoStreaming Interface Descriptor:{

bLength 6

**bDescriptorType 36**

**bDescriptorSubtype 13 (COLORFORMAT)**

bColorPrimaries 1 (BT.709,sRGB)

bTransferCharacteristics 1 (BT.709)

bMatrixCoefficients 4 (SMPTE 170M (BT.601))

}

##### VideoStreaming: FORMAT\_UNCOMPRESSED

VideoStreaming Interface Descriptor:{

bLength 27

**bDescriptorType 36**

**bDescriptorSubtype 4 (FORMAT\_UNCOMPRESSED)**

**bFormatIndex 2**

**bNumFrameDescriptors 10**

guidFormat {59555932-0000-1000-8000-00aa00389b71}

bBitsPerPixel 16

bDefaultFrameIndex 1

bAspectRatioX 0

bAspectRatioY 0

bmInterlaceFlags 0x00

Interlaced stream or variable: No

Fields per frame: 2 fields

Field 1 first: No

Field pattern: Field 1 only

bCopyProtect 0

}

###### VideoStreaming: FRAME\_UNCOMPRESSED(1920x1080)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 5 (FRAME\_UNCOMPRESSED)**

**bFrameIndex 1**

bmCapabilities 0x00

Still image unsupported

**wWidth 1920**

**wHeight 1080**

dwMinBitRate 132710400

dwMaxBitRate 132710400

dwMaxVideoFrameBufferSize 4147200

dwDefaultFrameInterval 2500000

bFrameIntervalType 1

dwFrameInterval( 0) 2500000

}

###### VideoStreaming: FRAME\_UNCOMPRESSED(352x288)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 5 (FRAME\_UNCOMPRESSED)**

**bFrameIndex 2**

bmCapabilities 0x00

Still image unsupported

**wWidth 352**

**wHeight 288**

dwMinBitRate 48660480

dwMaxBitRate 48660480

dwMaxVideoFrameBufferSize 202752

dwDefaultFrameInterval 333333

bFrameIntervalType 1

dwFrameInterval( 0) 333333

}

###### VideoStreaming: FRAME\_UNCOMPRESSED(320x244)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 5 (FRAME\_UNCOMPRESSED)**

**bFrameIndex 3**

bmCapabilities 0x00

Still image unsupported

**wWidth 320**

**wHeight 240**

dwMinBitRate 36864000

dwMaxBitRate 36864000

dwMaxVideoFrameBufferSize 153600

dwDefaultFrameInterval 333333

bFrameIntervalType 1

dwFrameInterval( 0) 333333

}

###### VideoStreaming: FRAME\_UNCOMPRESSED(176x144)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 5 (FRAME\_UNCOMPRESSED)**

**bFrameIndex 4**

bmCapabilities 0x00

Still image unsupported

**wWidth 176**

**wHeight 144**

dwMinBitRate 12165120

dwMaxBitRate 12165120

dwMaxVideoFrameBufferSize 50688

dwDefaultFrameInterval 333333

bFrameIntervalType 1

dwFrameInterval( 0) 333333

}

###### VideoStreaming: FRAME\_UNCOMPRESSED(160x120)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 5 (FRAME\_UNCOMPRESSED)**

**bFrameIndex 5**

bmCapabilities 0x00

Still image unsupported

**wWidth 160**

**wHeight 120**

dwMinBitRate 9216000

dwMaxBitRate 9216000

dwMaxVideoFrameBufferSize 38400

dwDefaultFrameInterval 333333

bFrameIntervalType 1

dwFrameInterval( 0) 333333

}

###### VideoStreaming: FRAME\_UNCOMPRESSED(640x480)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 5 (FRAME\_UNCOMPRESSED)**

**bFrameIndex 6**

bmCapabilities 0x00

Still image unsupported

**wWidth 640**

**wHeight 480**

dwMinBitRate 147456000

dwMaxBitRate 147456000

dwMaxVideoFrameBufferSize 614400

dwDefaultFrameInterval 333333

bFrameIntervalType 1

dwFrameInterval( 0) 333333

}

###### VideoStreaming: FRAME\_UNCOMPRESSED(800x600)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 5 (FRAME\_UNCOMPRESSED)**

**bFrameIndex 7**

bmCapabilities 0x00

Still image unsupported

**wWidth 800**

**wHeight 600**

dwMinBitRate 153600000

dwMaxBitRate 153600000

dwMaxVideoFrameBufferSize 960000

dwDefaultFrameInterval 500000

bFrameIntervalType 1

dwFrameInterval( 0) 500000

}

###### VideoStreaming: FRAME\_UNCOMPRESSED(1280x720)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 5 (FRAME\_UNCOMPRESSED)**

**bFrameIndex 8**

bmCapabilities 0x00

Still image unsupported

**wWidth 1280**

**wHeight 720**

dwMinBitRate 147456000

dwMaxBitRate 147456000

dwMaxVideoFrameBufferSize 1843200

dwDefaultFrameInterval 1000000

bFrameIntervalType 1

dwFrameInterval( 0) 1000000

}

###### VideoStreaming: FRAME\_UNCOMPRESSED(1280x1024)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 5 (FRAME\_UNCOMPRESSED)**

**bFrameIndex 9**

bmCapabilities 0x00

Still image unsupported

**wWidth 1280**

**wHeight 1024**

dwMinBitRate 104857600

dwMaxBitRate 104857600

dwMaxVideoFrameBufferSize 2621440

dwDefaultFrameInterval 2000000

bFrameIntervalType 1

dwFrameInterval( 0) 2000000

}

###### VideoStreaming: FRAME\_UNCOMPRESSED(1920x1080)

VideoStreaming Interface Descriptor:{

bLength 30

**bDescriptorType 36**

**bDescriptorSubtype 5 (FRAME\_UNCOMPRESSED)**

**bFrameIndex 10**

bmCapabilities 0x00

Still image unsupported

**wWidth 1920**

**wHeight 1080**

dwMinBitRate 132710400

dwMaxBitRate 132710400

dwMaxVideoFrameBufferSize 4147200

dwDefaultFrameInterval 2500000

bFrameIntervalType 1

dwFrameInterval( 0) 2500000

}

###### VideoStreaming: STILL\_IMAGE\_FRAME

VideoStreaming Interface Descriptor:{

bLength 10

**bDescriptorType 36**

**bDescriptorSubtype 3 (STILL\_IMAGE\_FRAME)**

bEndpointAddress 0

bNumImageSizePatterns 1

**wWidth( 0) 1920**

**wHeight( 0) 1080**

bNumCompressionPatterns 1

}

###### VideoStreaming: COLORFORMAT

VideoStreaming Interface Descriptor:{

bLength 6

**bDescriptorType 36**

**bDescriptorSubtype 13 (COLORFORMAT)**

bColorPrimaries 1 (BT.709,sRGB)

bTransferCharacteristics 1 (BT.709)

bMatrixCoefficients 4 (SMPTE 170M (BT.601))

}

#### Interface Descriptor 1: (bAlternateSetting 1) (Ep: addr=0x81, Isoc, 1x128B)

Interface Descriptor:

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 1**

**bAlternateSetting 1**

**bNumEndpoints 1**

bInterfaceClass 14 Video

bInterfaceSubClass 2 Video Streaming

bInterfaceProtocol 0

iInterface 0

Endpoint Descriptor:{

bLength 7

**bDescriptorType 5**  //端点描述符

**bEndpointAddress 0x81** EP 1 IN

bmAttributes 5

**Transfer Type Isochronous**

Synch Type Asynchronous

Usage Type Data

**wMaxPacketSize 0x0080 1x 128 bytes**

bInterval 1

#### Interface Descriptor 1: (bAlternateSetting 2) (Ep: addr=0x81, Isoc, 1x512B)

Interface Descriptor:

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 1**

**bAlternateSetting 2**

**bNumEndpoints 1**

bInterfaceClass 14 Video

bInterfaceSubClass 2 Video Streaming

bInterfaceProtocol 0

iInterface 0

Endpoint Descriptor:{

bLength 7

**bDescriptorType 5**  //端点描述符

**bEndpointAddress 0x81 EP 1 IN**

bmAttributes 5

**Transfer Type Isochronous**

Synch Type Asynchronous

Usage Type Data

**wMaxPacketSize 0x0200 1x 512 bytes**

bInterval 1

#### Interface Descriptor 1: (bAlternateSetting 3) (Ep: addr=0x81, Isoc, 1x1024B)

Interface Descriptor:

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 1**

**bAlternateSetting 3**

**bNumEndpoints 1**

bInterfaceClass 14 Video

bInterfaceSubClass 2 Video Streaming

bInterfaceProtocol 0

iInterface 0

Endpoint Descriptor:{

bLength 7

**bDescriptorType 5**  //端点描述符

**bEndpointAddress 0x81 EP 1 IN**

bmAttributes 5

**Transfer Type Isochronous**

Synch Type Asynchronous

Usage Type Data

**wMaxPacketSize 0x0400 1x 1024 bytes**

bInterval 1

#### Interface Descriptor 1: (bAlternateSetting 4) (Ep: addr=0x81, Isoc, 2x768B)

Interface Descriptor:

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 1**

**bAlternateSetting 4**

**bNumEndpoints 1**

bInterfaceClass 14 Video

bInterfaceSubClass 2 Video Streaming

bInterfaceProtocol 0

iInterface 0

Endpoint Descriptor:{

bLength 7

**bDescriptorType 5**  //端点描述符

**bEndpointAddress 0x81 EP 1 IN**

bmAttributes 5

**Transfer Type Isochronous**

Synch Type Asynchronous

Usage Type Data

**wMaxPacketSize 0x0b00 2x 768 bytes**

bInterval 1

#### Interface Descriptor 1: (bAlternateSetting 5) (Ep: addr=0x81, Isoc, 2x1024B)

对于音频跟视频这种同步传输方式，在2.0协议中，端点大小最大可以是1024，但D11~D12表示这个端点在一个Interval间隔中可以传输n次，所以在视频类中可以看到端点包大小有2x1024这样的格式。

Interface Descriptor:

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 1**

**bAlternateSetting 5**

**bNumEndpoints 1**

bInterfaceClass 14 Video

bInterfaceSubClass 2 Video Streaming

bInterfaceProtocol 0

iInterface 0

Endpoint Descriptor:{

bLength 7

**bDescriptorType 5**  //端点描述符

**bEndpointAddress 0x81 EP 1 IN**

bmAttributes 5

**Transfer Type Isochronous**

Synch Type Asynchronous

Usage Type Data

**wMaxPacketSize 0x0c00 2x 1024 bytes**

bInterval 1

#### Interface Descriptor 1: (bAlternateSetting 6) (Ep: addr=0x81, Isoc, 3x896B)

Interface Descriptor:

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 1**

**bAlternateSetting 6**

**bNumEndpoints 1**

bInterfaceClass 14 Video

bInterfaceSubClass 2 Video Streaming

bInterfaceProtocol 0

iInterface 0

Endpoint Descriptor:{

bLength 7

**bDescriptorType 5**  //端点描述符

**bEndpointAddress 0x81 EP 1 IN**

bmAttributes 5

**Transfer Type Isochronous**

Synch Type Asynchronous

Usage Type Data

**wMaxPacketSize 0x1380 3x 896 bytes**

bInterval 1

#### Interface Descriptor 1: (bAlternateSetting 7) (Ep: addr=0x81, Isoc, 3x1024B)

Interface Descriptor:

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 1**

**bAlternateSetting 7**

**bNumEndpoints 1**

bInterfaceClass 14 Video

bInterfaceSubClass 2 Video Streaming

bInterfaceProtocol 0

iInterface 0

Endpoint Descriptor:{

bLength 7

**bDescriptorType 5**  //端点描述符

**bEndpointAddress 0x81 EP 1 IN**

bmAttributes 5

**Transfer Type Isochronous**

Synch Type Asynchronous

Usage Type Data

**wMaxPacketSize 0x1400 3x 1024 bytes**

bInterval 1

#### Interface Descriptor 2: (bAlternateSetting 0) Audio Control

可以通过Audio Control描述符布局,分析出MIC设备框架: **IT(1)->FU(3)->OT(2)**

Interface Descriptor:{

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 2**

**bAlternateSetting 0**

**bNumEndpoints 0**

**bInterfaceClass 1 Audio**

**bInterfaceSubClass 1 Control Device**

bInterfaceProtocol 0

iInterface 6 Realtek USB2.0 MIC

}

##### AudioControl: HEADER

AudioControl Interface Descriptor:{

bLength 9

**bDescriptorType 36**

**bDescriptorSubtype 1 (HEADER)**

bcdADC 1.00

wTotalLength 39

bInCollection 1

baInterfaceNr( 0) 3

}

##### AudioControl: INPUT\_TERMINAL(bTerminalID=1)

AudioControl Interface Descriptor:{

bLength 12

**bDescriptorType 36**

**bDescriptorSubtype 2 (INPUT\_TERMINAL)**

**bTerminalID 1**

**wTerminalType 0x0201 Microphone**

bAssocTerminal 0

**bNrChannels 1**

wChannelConfig 0x0003

Left Front (L)

Right Front (R)

iChannelNames 0

iTerminal 0

}

##### AudioControl: OUTPUT\_TERMINAL(bSourceID=3, bTerminalID=2)

AudioControl Interface Descriptor:{

bLength 9

**bDescriptorType 36**

**bDescriptorSubtype 3 (OUTPUT\_TERMINAL)**

**bTerminalID 2**

**wTerminalType 0x0101 USB Streaming**

bAssocTerminal 1

**bSourceID 3**

iTerminal 0

}

##### AudioControl: FEATURE\_UNIT(bSourceID=1, bUnitID=3)

AudioControl Interface Descriptor:{

bLength 9

**bDescriptorType 36**

**bDescriptorSubtype 6 (FEATURE\_UNIT)**

**bUnitID 3**

**bSourceID 1**

bControlSize 2

bmaControls( 0) 0x03

**bmaControls( 0) 0x02**

Mute Control

Volume Control

Loudness Control

iFeature 0

}

#### Interface Descriptor 3: (bAlternateSetting 0)

Interface Descriptor:{

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 3**

**bAlternateSetting 0**

**bNumEndpoints 0**

bInterfaceClass 1 Audio

bInterfaceSubClass 2 Streaming

bInterfaceProtocol 0

iInterface 0

}

#### Interface Descriptor 3: (bAlternateSetting 1) (Ep: addr=0x82, Isoc, 1x120B, 22050)

Sample rate: 2chn \* 16bit 🡪 22050, PCM

Interface Descriptor:

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 3**

**bAlternateSetting 1**

**bNumEndpoints 1**

bInterfaceClass 1 Audio

bInterfaceSubClass 2 Streaming

bInterfaceProtocol 0

iInterface 0

AudioStreaming Interface Descriptor:

bLength 7

**bDescriptorType 36**

bDescriptorSubtype 1 (AS\_GENERAL)

bTerminalLink 2

bDelay 1 frames

**wFormatTag 1 PCM**

AudioStreaming Interface Descriptor:

bLength 11

**bDescriptorType 36**

bDescriptorSubtype 2 (FORMAT\_TYPE)

bFormatType 1 (FORMAT\_TYPE\_I)

**bNrChannels 2**

bSubframeSize 2

**bBitResolution 16**

bSamFreqType 1 Discrete

**tSamFreq[ 0] 22050**

Endpoint Descriptor:

bLength 9

**bDescriptorType 5**  //端点描述符

**bEndpointAddress 0x82 EP 2 IN**

bmAttributes 5

**Transfer Type Isochronous**

Synch Type Asynchronous

Usage Type Data

**wMaxPacketSize 0x0078 1x 120 bytes**

bInterval 4

bRefresh 0

bSynchAddress 0

AudioControl Endpoint Descriptor: …

#### Interface Descriptor 3: (bAlternateSetting 2) (Ep: addr=0x82, Isoc, 1x160B, 32000)

Sample rate: 2chn \* 16bit 🡪 32000, PCM

Interface Descriptor:

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 3**

**bAlternateSetting 2**

**bNumEndpoints 1**

bInterfaceClass 1 Audio

bInterfaceSubClass 2 Streaming

bInterfaceProtocol 0

iInterface 0

AudioStreaming Interface Descriptor:

bLength 7

**bDescriptorType 36**

bDescriptorSubtype 1 (AS\_GENERAL)

bTerminalLink 2

bDelay 1 frames

**wFormatTag 1 PCM**

AudioStreaming Interface Descriptor:

bLength 11

**bDescriptorType 36**

bDescriptorSubtype 2 (FORMAT\_TYPE)

bFormatType 1 (FORMAT\_TYPE\_I)

**bNrChannels 2**

bSubframeSize 2

**bBitResolution 16**

bSamFreqType 1 Discrete

**tSamFreq[ 0] 32000**

Endpoint Descriptor:

bLength 9

**bDescriptorType 5**  //端点描述符

**bEndpointAddress 0x82 EP 2 IN**

bmAttributes 5

**Transfer Type Isochronous**

Synch Type Asynchronous

Usage Type Data

**wMaxPacketSize 0x00a0 1x 160 bytes**

bInterval 4

bRefresh 0

bSynchAddress 0

AudioControl Endpoint Descriptor: …

#### Interface Descriptor 3: (bAlternateSetting 3) (Ep: addr=0x82, Isoc, 1x216B, 48000)

Sample rate: 1chn \* 16bit 🡪 48000, PCM

Interface Descriptor:

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 3**

**bAlternateSetting 3**

**bNumEndpoints 1**

bInterfaceClass 1 Audio

bInterfaceSubClass 2 Streaming

bInterfaceProtocol 0

iInterface 0

AudioStreaming Interface Descriptor:

bLength 7

**bDescriptorType 36**

bDescriptorSubtype 1 (AS\_GENERAL)

bTerminalLink 2

bDelay 1 frames

**wFormatTag 1 PCM**

AudioStreaming Interface Descriptor:

bLength 11

**bDescriptorType 36**

bDescriptorSubtype 2 (FORMAT\_TYPE)

bFormatType 1 (FORMAT\_TYPE\_I)

**bNrChannels 1**

bSubframeSize 2

**bBitResolution 16**

bSamFreqType 1 Discrete

**tSamFreq[ 0] 48000**

Endpoint Descriptor:

bLength 9

**bDescriptorType 5**  //端点描述符

**bEndpointAddress 0x82 EP 2 IN**

bmAttributes 5

**Transfer Type Isochronous**

Synch Type Asynchronous

Usage Type Data

**wMaxPacketSize 0x00d8 1x 216 bytes**

bInterval 4

bRefresh 0

bSynchAddress 0

AudioControl Endpoint Descriptor: …

#### Interface Descriptor 3: (bAlternateSetting 4) (Ep: addr=0x82, Isoc, 1x216B, 48000)

Sample rate: 2chn \* 16bit 🡪 48000, PCM

Interface Descriptor:

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 3**

**bAlternateSetting 4**

**bNumEndpoints 1**

bInterfaceClass 1 Audio

bInterfaceSubClass 2 Streaming

bInterfaceProtocol 0

iInterface 0

AudioStreaming Interface Descriptor:

bLength 7

**bDescriptorType 36**

bDescriptorSubtype 1 (AS\_GENERAL)

bTerminalLink 2

bDelay 1 frames

**wFormatTag 1 PCM**

AudioStreaming Interface Descriptor:

bLength 11

**bDescriptorType 36**

bDescriptorSubtype 2 (FORMAT\_TYPE)

bFormatType 1 (FORMAT\_TYPE\_I)

**bNrChannels 2**

bSubframeSize 2

**bBitResolution 16**

bSamFreqType 1 Discrete

**tSamFreq[ 0] 48000**

Endpoint Descriptor:

bLength 9

**bDescriptorType 5**  //端点描述符

**bEndpointAddress 0x82 EP 2 IN**

bmAttributes 5

**Transfer Type Isochronous**

Synch Type Asynchronous

Usage Type Data

**wMaxPacketSize 0x00d8 1x 216 bytes**

bInterval 4

bRefresh 0

bSynchAddress 0

AudioControl Endpoint Descriptor: …

#### Interface Descriptor 3: (bAlternateSetting 5) (Ep: addr=0x82, Isoc, 1x432B, 96000)

Sample rate: 2chn \* 16bit 🡪 96000, PCM

Interface Descriptor:

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 3**

**bAlternateSetting 5**

bNumEndpoints 1

bInterfaceClass 1 Audio

bInterfaceSubClass 2 Streaming

bInterfaceProtocol 0

iInterface 0

AudioStreaming Interface Descriptor:

bLength 7

**bDescriptorType 36**

bDescriptorSubtype 1 (AS\_GENERAL)

bTerminalLink 2

bDelay 1 frames

**wFormatTag 1 PCM**

AudioStreaming Interface Descriptor:

bLength 11

**bDescriptorType 36**

bDescriptorSubtype 2 (FORMAT\_TYPE)

bFormatType 1 (FORMAT\_TYPE\_I)

**bNrChannels 2**

bSubframeSize 2

**bBitResolution 16**

bSamFreqType 1 Discrete

**tSamFreq[ 0] 96000**

Endpoint Descriptor:

bLength 9

**bDescriptorType 5**  //端点描述符

**bEndpointAddress 0x82 EP 2 IN**

bmAttributes 5

**Transfer Type Isochronous**

Synch Type Asynchronous

Usage Type Data

**wMaxPacketSize 0x01b0 1x 432 bytes**

bInterval 4

bRefresh 0

bSynchAddress 0

AudioControl Endpoint Descriptor: …

#### Interface Descriptor 3: (bAlternateSetting 6) (Ep: addr=0x82, Isoc, 1x304B, 44100)

Sample rate: 2chn \* 24bit 🡪 44100, PCM

Interface Descriptor:

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 3**

**bAlternateSetting 6**

bNumEndpoints 1

bInterfaceClass 1 Audio

bInterfaceSubClass 2 Streaming

bInterfaceProtocol 0

iInterface 0

AudioStreaming Interface Descriptor:

bLength 7

**bDescriptorType 36**

bDescriptorSubtype 1 (AS\_GENERAL)

bTerminalLink 2

bDelay 1 frames

**wFormatTag 1 PCM**

AudioStreaming Interface Descriptor:

bLength 11

**bDescriptorType 36**

bDescriptorSubtype 2 (FORMAT\_TYPE)

bFormatType 1 (FORMAT\_TYPE\_I)

**bNrChannels 2**

bSubframeSize 3

**bBitResolution 24**

bSamFreqType 1 Discrete

**tSamFreq[ 0] 44100**

Endpoint Descriptor:

bLength 9

**bDescriptorType 5**  //端点描述符

**bEndpointAddress 0x82 EP 2 IN**

bmAttributes 5

**Transfer Type Isochronous**

Synch Type Asynchronous

Usage Type Data

**wMaxPacketSize 0x0130 1x 304 bytes**

bInterval 4

bRefresh 0

bSynchAddress 0

AudioControl Endpoint Descriptor: …

#### Interface Descriptor 3: (bAlternateSetting 7) (Ep: addr=0x82, Isoc, 1x336B, 48000)

Sample rate: 2chn \* 24bit 🡪 48000, PCM

Interface Descriptor:

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 3**

**bAlternateSetting 7**

bNumEndpoints 1

bInterfaceClass 1 Audio

bInterfaceSubClass 2 Streaming

bInterfaceProtocol 0

iInterface 0

AudioStreaming Interface Descriptor:

bLength 7

**bDescriptorType 36**

bDescriptorSubtype 1 (AS\_GENERAL)

bTerminalLink 2

bDelay 1 frames

**wFormatTag 1 PCM**

AudioStreaming Interface Descriptor:

bLength 11

**bDescriptorType 36**

bDescriptorSubtype 2 (FORMAT\_TYPE)

bFormatType 1 (FORMAT\_TYPE\_I)

**bNrChannels 2**

bSubframeSize 3

**bBitResolution 24**

bSamFreqType 1 Discrete

**tSamFreq[ 0] 48000**

Endpoint Descriptor:

bLength 9

**bDescriptorType 5**  //端点描述符

bEndpointAddress 0x82 EP 2 IN

bmAttributes 5

Transfer Type Isochronous

Synch Type Asynchronous

Usage Type Data

**wMaxPacketSize 0x0150 1x 336 bytes**

bInterval 4

bRefresh 0

bSynchAddress 0

AudioControl Endpoint Descriptor: …

#### Interface Descriptor 3: (bAlternateSetting 8) (Ep: addr=0x82, Isoc, 1x656B, 96000 )

Sample rate: 2chn \* 24bit 🡪 96000, PCM

Interface Descriptor:

bLength 9

**bDescriptorType 4**  //接口描述符

**bInterfaceNumber 3**

**bAlternateSetting 8**

bNumEndpoints 1

bInterfaceClass 1 Audio

bInterfaceSubClass 2 Streaming

bInterfaceProtocol 0

iInterface 0

AudioStreaming Interface Descriptor:

bLength 7

**bDescriptorType 36**

bDescriptorSubtype 1 (AS\_GENERAL)

bTerminalLink 2

bDelay 1 frames

**wFormatTag 1 PCM**

AudioStreaming Interface Descriptor:

bLength 11

**bDescriptorType 36**

bDescriptorSubtype 2 (FORMAT\_TYPE)

bFormatType 1 (FORMAT\_TYPE\_I)

**bNrChannels 2**

bSubframeSize 3

**bBitResolution 24**

bSamFreqType 1 Discrete

**tSamFreq[ 0] 96000**

Endpoint Descriptor:

bLength 9

**bDescriptorType 5**  //端点描述符

**bEndpointAddress 0x82 EP 2 IN**

bmAttributes 5

**Transfer Type Isochronous**

Synch Type Asynchronous

Usage Type Data

**wMaxPacketSize 0x0290 1x 656 bytes**

bInterval 4

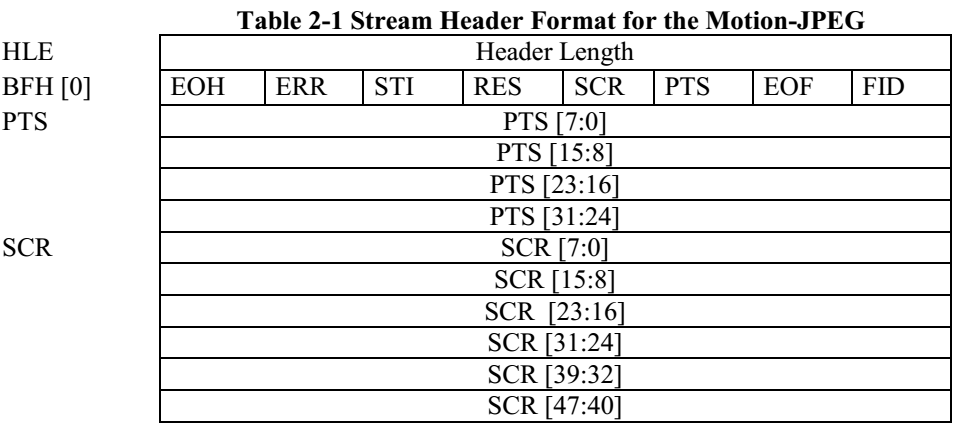
bRefresh 0

bSynchAddress 0

AudioControl Endpoint Descriptor: …

# Urb 数据格式

## MLPEG:



FID: Frame Identifier

This bit toggles at each frame start boundary and stays constant for the rest of the frame.

EOF: End of Frame

This bit indicates the end of a video frame and is set in the last video sample that belongs to a

frame.

PTS: Presentation Time Stamp

This bit, when set, indicates the presence of a PTS field.

SCR: Source Clock Reference

This bit, when set, indicates the presence of a SCR field

RES: Reserved.

Set to 0.

STI: Still Image

This bit, when set, identifies a video sample that belongs to a still image.

ERR: Error Bit

This bit, when set, indicates an error in the device streaming.

EOH: End of Header

This bit, when set, indicates the end of the BFH fields.

**Fid相同的packet数据属于同一个frame的数据。**

[14098.690958] [0] [L=1024] 0c cc 01 a6 a8 00 af 2c a9 00 d5 06 [fid:0][eof:0][sti:0][err:0]

[14098.695500] [0] [L=2048] 0c cc 01 a6 a8 00 5b 34 aa 00 d9 06 [fid:0][eof:0][sti:0][err:0]

[14098.699974] [0] [L=1024] 0c cc 01 a6 a8 00 06 3c ab 00 de 06 [fid:0][eof:0][sti:0][err:0]

[14098.704499] [0] [L=2048] 0c ce 01 a6 a8 00 b1 43 ac 00 e2 06 [fid:0][eof:2][sti:0][err:0]

[14098.708932] [0] [L= 76] 0c ce 01 a6 a8 00 5c 4b ad 00 e7 06 [fid:0][eof:2][sti:0][err:0]

[14098.713287] [0] [L= 12] 0c cc 01 a6 a8 00 06 53 ae 00 eb 06 [fid:0][eof:0][sti:0][err:0]

[14098.717745] [0] [L= 24] 0c cc 01 a6 a8 00 5d 53 af 00 ef 06 [fid:0][eof:0][sti:0][err:0]

[14098.722122] [0] [L= 12] 0c cc 01 a6 a8 00 b4 53 b0 00 f4 06 [fid:0][eof:0][sti:0][err:0]

[14098.726503] [0] [L=1024] 0c cd 62 46 b0 00 0d 54 b1 00 f8 06 [fid:1][eof:0][sti:0][err:0]

[14098.730967] [0] [L=1024] 0c cd 62 46 b0 00 66 54 b2 00 fd 06 [fid:1][eof:0][sti:0][err:0]

[14098.735367] [0] [L=1024] 0c cd 62 46 b0 00 10 5c b3 00 01 07 [fid:1][eof:0][sti:0][err:0]

[14098.739838] [0] [L=1024] 0c cd 62 46 b0 00 67 5c b4 00 05 07 [fid:1][eof:0][sti:0][err:0]

[14098.744239] [0] [L=1024] 0c cf 62 46 b0 00 12 64 b5 00 0a 07 [fid:1][eof:2][sti:0][err:0]

[14098.748707] [0] [L=1024] 0c cf 62 46 b0 00 69 64 b6 00 0e 07 [fid:1][eof:2][sti:0][err:0]

[14098.753066] [0] [L= 76] 0c cf 62 46 b0 00 13 6c b7 00 13 07 [fid:1][eof:2][sti:0][err:0]

[14098.757464] [0] [L=1024] 0c cc c3 e6 b7 00 6a 6c b8 00 17 07 [fid:0][eof:0][sti:0][err:0]

[14098.761912] [0] [L=1024] 0c cc c3 e6 b7 00 c2 6c b9 00 1c 07 [fid:0][eof:0][sti:0][err:0]

[14098.766313] [0] [L=1024] 0c cc c3 e6 b7 00 6c 74 ba 00 20 07 [fid:0][eof:0][sti:0][err:0]

[14098.770759] [0] [L=1024] 0c cc c3 e6 b7 00 c3 74 bb 00 24 07 [fid:0][eof:0][sti:0][err:0]

[14098.775143] [0] [L=1024] 0c ce c3 e6 b7 00 1b 75 bc 00 29 07 [fid:0][eof:2][sti:0][err:0]

[14098.779605] [0] [L=1024] 0c ce c3 e6 b7 00 73 75 bd 00 2d 07 [fid:0][eof:2][sti:0][err:0]

[14098.784063] [0] [L= 88] 0c ce c3 e6 b7 00 1c 7d be 00 32 07 [fid:0][eof:2][sti:0][err:0]

[14098.788491] [0] [L= 12] 0c cc c3 e6 b7 00 c6 84 bf 00 36 07 [fid:0][eof:0][sti:0][err:0]

[14098.792891] [0] [L=1024] 0c cd 23 87 bf 00 1f 85 c0 00 3b 07 [fid:1][eof:0][sti:0][err:0]

[14098.797291] [0] [L=1024] 0c cd 23 87 bf 00 78 85 c1 00 3f 07 [fid:1][eof:0][sti:0][err:0]

[14098.801740] [0] [L=1024] 0c cd 23 87 bf 00 22 8d c2 00 43 07 [fid:1][eof:0][sti:0][err:0]

[14098.806243] [0] [L=2048] 0c cd 23 87 bf 00 79 8d c3 00 48 07 [fid:1][eof:0][sti:0][err:0]

[14098.810692] [0] [L=1024] 0c cf 23 87 bf 00 24 95 c4 00 4c 07 [fid:1][eof:2][sti:0][err:0]

[14098.815053] [0] [L= 76] 0c cf 23 87 bf 00 ce 9c c5 00 51 07 [fid:1][eof:2][sti:0][err:0]

[14098.819454] [0] [L= 12] 0c cd 23 87 bf 00 26 9d c6 00 55 07 [fid:1][eof:0][sti:0][err:0]

# USB初始化

## Usb driver驱动的注册

### usb\_register()

#define usb\_register(driver) \

usb\_register\_driver(driver, THIS\_MODULE, KBUILD\_MODNAME)

int usb\_register\_driver(struct usb\_driver \*new\_driver, struct module \*owner, const char \*mod\_name)

{

new\_driver->drvwrap.for\_devices = 0;

new\_driver->drvwrap.**driver.name = (char \*) new\_driver->name;**

new\_driver->drvwrap.driver.bus = &usb\_bus\_type;

new\_driver->drvwrap.**driver.probe = usb\_probe\_interface;**

new\_driver->drvwrap.driver.remove = usb\_unbind\_interface;

new\_driver->drvwrap.driver.owner = owner;

new\_driver->drvwrap.driver.mod\_name = mod\_name;

spin\_lock\_init(&new\_driver->dynids.lock);

INIT\_LIST\_HEAD(&new\_driver->dynids.list);

retval = driver\_register(&new\_driver->drvwrap.driver);

}

usb\_register(&uvc\_driver.driver);

struct uvc\_driver uvc\_driver = {

.driver = {

**.name = "uvcvideo",**

**.probe = uvc\_probe,**

}

usb\_register(&hub\_driver);

static struct usb\_driver hub\_driver = {

**.name = "hub",**

**.probe = hub\_probe,**

}

usb\_register(&usbfs\_driver);

struct usb\_driver usbfs\_driver = {

**.name = "usbfs",**

**.probe = driver\_probe,**

}

## Usb device驱动注册标准接口

### usb\_register\_device\_driver()

int usb\_register\_device\_driver(struct **usb\_device\_driver** \*new\_udriver, struct module \*owner)

{

new\_udriver->drvwrap.for\_devices = 1;

new\_udriver->drvwrap.**driver.name = (char \*) new\_udriver->name;**

new\_udriver->drvwrap.driver.bus = &usb\_bus\_type;

new\_udriver->drvwrap.driver**.probe = usb\_probe\_device;**

new\_udriver->drvwrap.driver.remove = usb\_unbind\_device;

new\_udriver->drvwrap.driver.owner = owner;

retval = driver\_register(&new\_udriver->drvwrap.driver);

}

**usb\_register\_device\_driver**(&usb\_generic\_driver, THIS\_MODULE);

struct **usb\_device\_driver** usb\_generic\_driver = {

**.name = "usb",**

**.probe = generic\_probe,**

}

## usb\_init()

drivers\usb\core\usb.c

subsys\_initcall(usb\_init);

static int \_\_init usb\_init(void)

{

usb\_acpi\_register();

usb\_debugfs\_init();    //1.usb debugfs初始化

retval = bus\_register(&usb\_bus\_type);//2.usb总线注册

retval = bus\_register\_notifier(&usb\_bus\_type, &usb\_bus\_nb);//3.usb总线通知链注册

retval = usb\_major\_init();//4.注册usb主控器字符设备

retval = usb\_register(&usbfs\_driver);//5.注册usbfs驱动

retval = usb\_devio\_init(); //6.usb设备字符设备初始化

retval = usb\_hub\_init();//8.usb hub初始化

retval = usb\_register\_device\_driver(&usb\_generic\_driver, THIS\_MODULE);//9.usb注册同样设备驱动

usb\_hub\_cleanup();

}

## 注册Usb总线

kernel\drivers\media\usb\uvc\uvc\_driver.c

struct bus\_type usb\_bus\_type = {

.name = "usb",

**.match = usb\_device\_match,**

.uevent = usb\_uevent,

};

## 注册Usb主控制器字符设备

usb\_major\_init()

USB\_MAJOR=180

int usb\_major\_init(void)

{

error = register\_chrdev(USB\_MAJOR, "usb", &usb\_fops);

}

### struct file\_operations usb\_fops

cdev->ops = fops;

static const struct file\_operations usb\_fops = {

.owner = THIS\_MODULE,

.open = usb\_open,

.llseek = noop\_llseek,

};

## Usbfs驱动的注册

### usb\_driver usbfs\_driver

struct usb\_driver usbfs\_driver = {

**.name = "usbfs",**

.probe = driver\_probe,

.disconnect = driver\_disconnect,

.suspend = driver\_suspend,

.resume = driver\_resume,

};

## usb设备字符设备初始化

### usb\_devio\_init()

usbfs为用户提供了在用户空间直接访问usb硬件设备的接口

#define USB\_DEVICE\_MAJOR 189

#define USB\_DEVICE\_DEV MKDEV(USB\_DEVICE\_MAJOR, 0)

int \_\_init usb\_devio\_init(void)

{

//register\_chrdev\_region 是根据要求的范围进行申请，同时我们需要手动 cdev\_init cdev\_add

retval = register\_chrdev\_region(USB\_DEVICE\_DEV, USB\_DEVICE\_MAX, "usb\_device"); //分配设备号

cdev\_init(&usb\_device\_cdev, **&usbdev\_file\_operations**); //字符设备初始化

retval = cdev\_add(&usb\_device\_cdev, USB\_DEVICE\_DEV, USB\_DEVICE\_MAX); //添加字符设备

usb\_register\_notify(**&usbdev\_nb)**; //注册设备通知链

}

### struct file\_operations usbdev\_file\_operations

const struct file\_operations usbdev\_file\_operations = {

.owner = THIS\_MODULE,

.llseek = usbdev\_lseek,

.read = usbdev\_read,

.poll = usbdev\_poll,

.unlocked\_ioctl = usbdev\_ioctl,

#ifdef CONFIG\_COMPAT

.compat\_ioctl = usbdev\_compat\_ioctl,

#endif

.open = usbdev\_open,

.release = usbdev\_release,

};

### usbdev\_read()

static ssize\_t usbdev\_read(struct file \*file, char \_\_user \*buf, size\_t nbytes, loff\_t \*ppos)

{

struct dev\_state \*ps = file->private\_data;

struct usb\_device \*dev = ps->dev;

pos = \*ppos;

if (pos < sizeof(struct usb\_device\_descriptor)) {

struct usb\_device\_descriptor temp\_desc;

memcpy(&temp\_desc, &**dev->descriptor**, sizeof(dev->descriptor));

len = sizeof(struct usb\_device\_descriptor) - pos;

if (**copy\_to\_user**(buf, ((char \*)&temp\_desc) + pos, len)) {

ret = -EFAULT;

goto err;

}

\*ppos += len;buf += len;nbytes -= len;ret += len;

}

pos = sizeof(struct usb\_device\_descriptor);

for (i = 0; nbytes && i <**dev->descriptor.bNumConfigurations**; i++) {

struct usb\_config\_descriptor \*config = (struct usb\_config\_descriptor \*)dev->rawdescriptors[i];

unsigned int length = le16\_to\_cpu(config->wTotalLength);

if (\*ppos < pos + length) {

/\* The descriptor may claim to be longer than it really is. Here is the actual allocated length. \*/

unsigned alloclen = le16\_to\_cpu(dev->config[i].desc.wTotalLength);

len = length - (\*ppos - pos);

if (len > nbytes)

len = nbytes;

/\* Simply don't write (skip over) unallocated parts \*/

if (alloclen > (\*ppos - pos)) {

alloclen -= (\*ppos - pos);

if (**copy\_to\_user**(buf, **dev->rawdescriptors[i]** + (\*ppos - pos), min(len, alloclen))) {

ret = -EFAULT;

goto err;

}

}

\*ppos += len;buf += len;nbytes -= len;ret += len;

}

pos += length;

}

}

### struct notifier\_block usbdev\_nb

static struct notifier\_block usbdev\_nb = {

.notifier\_call = usbdev\_notify,

};

### usbdev\_notify()

static int usbdev\_notify(struct notifier\_block \*self, unsigned long action, void \*dev)

{

switch (action) {

case USB\_DEVICE\_**ADD**: break;

case USB\_DEVICE\_**REMOVE**: **usbdev\_remove**(dev); break;

}

return NOTIFY\_OK;

}

## Usb hub驱动的注册

### usb\_hub\_init()

int usb\_hub\_init(void)

{

usb\_register(&hub\_driver);

khubd\_task = kthread\_run(hub\_thread, NULL, "khubd");

}

### struct usb\_driver hub\_driver

static struct **usb\_driver** hub\_driver = {

**.name = "hub",**

.probe = hub\_probe,

.disconnect = hub\_disconnect,

.suspend = hub\_suspend,

.resume = hub\_resume,

.reset\_resume = hub\_reset\_resume,

.pre\_reset = hub\_pre\_reset,

.post\_reset = hub\_post\_reset,

.unlocked\_ioctl = hub\_ioctl,

.id\_table = hub\_id\_table,

.supports\_autosuspend = 1,

};

### hub\_thread()

static int hub\_thread(void \*\_\_unused)

{

/\* khubd needs to be freezable to avoid intefering with USB-PERSIST

\* port handover. Otherwise it might see that a full-speed device

\* was gone before the EHCI controller had handed its port over to

\* the companion full-speed controller.

\*/

set\_freezable();

do {

hub\_events();

wait\_event\_freezable(khubd\_wait,!list\_empty(&hub\_event\_list) || kthread\_should\_stop());

} while (!kthread\_should\_stop() || !list\_empty(&hub\_event\_list));

return 0;

}

### hub\_events()

static void hub\_events(void)

{

struct list\_head \*tmp;

struct usb\_device \*hdev;

struct usb\_interface \*intf;

struct usb\_hub \*hub;

struct device \*hub\_dev;

while (1) {

tmp = hub\_event\_list.next;

hub = list\_entry(tmp, struct usb\_hub, event\_list);

hdev = hub->hdev;

hub\_dev = hub->intfdev;

intf = to\_usb\_interface(hub\_dev);

//插入uvc camera: state 7 ports 1 chg 0000 evt 0002

dev\_dbg(hub\_dev, "state %d ports %d chg %04x evt %04x\n",

hdev->state,

hub->descriptor ? hub->descriptor->bNbrPorts : 0,

(u16) hub->change\_bits[0],

(u16) hub->event\_bits[0]);

ret = usb\_autopm\_get\_interface(intf); //自动唤醒interface

/\* deal with port status changes \*/

for (i = 1; i <= hub->descriptor->bNbrPorts; i++) {

connect\_change = test\_bit(i, hub->change\_bits);

if (connect\_change)

**hub\_port\_connect\_change(hub, i, portstatus, portchange);**

}

usb\_autopm\_put\_interface\_no\_suspend(intf);

} /\* end while (1) \*/

}

### hub\_port\_connect\_change()

static void hub\_port\_connect\_change(struct usb\_hub \*hub, int port1, u16 portstatus, u16 portchange)

{

//清除当前port的change\_bits标志；

clear\_bit(port1, hub->change\_bits);

//对于一些已经连接的设备，进行一些类似防抖动处理，处理机制：每隔25ms去读取port的status和change状态，查看port连接状态是否稳定，如果port处于稳定状态大于100ms，则认为当前port上的设备已经稳定，这种处理机制最长时间为1.5s;如果port处于稳定状态的时间小于100ms，则认为连接是不稳定的，则跳出当前port，去执行下一个port; 对于一个新插入的设备，并已经确定它的存在，则接下来会为这个usb设备进行枚举；

if (portchange & (USB\_PORT\_STAT\_C\_CONNECTION | USB\_PORT\_STAT\_C\_ENABLE)) {

status = hub\_port\_debounce\_be\_stable(hub, port1);

}

for (i = 0; i < SET\_CONFIG\_TRIES; i++) {

//通过usb\_alloc\_dev为新的USB设备申请资源，并进行一些初始化，将usb设置为USB\_STATE\_ATTACHED，表示设备已经连接；

udev = **usb\_alloc\_dev**(hdev, hdev->bus, port1);

usb\_set\_device\_state(udev, USB\_STATE\_POWERED); //设置usb设备为USB\_STATE\_POWERED态；

udev->bus\_mA = hub->mA\_per\_port; //设置usb设备可以从port上获取的电流量，及usb设备的拓扑层级；

udev->level = hdev->level + 1;

udev->wusb = hub\_is\_wusb(hub);

//如果hub支持超速，则设置usb设备的速度为超速，否则设置usb设备速度为unknow;

if (hub\_is\_superspeed(hub->hdev)) udev->speed = USB\_SPEED\_SUPER;

else udev->speed = USB\_SPEED\_UNKNOWN;

//一条usb总线下总共可以有128个设备，usb总线通过位图的形式来管理usb设备的地址，choose\_devnum是从usb 总线中找到一个usb地址，usb地址在1和128之间；

choose\_devnum(udev);

//通过hub\_port\_init对usb设备进行reset，并设置usb设备的地址，获取usb设备的设备描述符

status = **hub\_port\_init**(hub, udev, port1, i);

//如果当前的usb设备是一个hub，它由hub供电，而它从上一级hub上分配到的电流少于100mA，这种情况是不允许的，它会通过点亮AMBER指示灯来显示出现这种错误，指示灯由延时任务队列完成；

if (udev->descriptor.bDeviceClass == USB\_CLASS\_HUB && udev->bus\_mA <= unit\_load) {

status = usb\_get\_status(udev, USB\_RECIP\_DEVICE, 0,&devstat);

le16\_to\_cpus(&devstat);

if ((devstat & (1 << USB\_DEVICE\_SELF\_POWERED)) == 0) {

if (hub->has\_indicators) {

hub->indicator[port1-1] = INDICATOR\_AMBER\_BLINK;

schedule\_delayed\_work (&hub->leds, 0);

}

status = -ENOTCONN; /\* Don't retry \*/

goto loop\_disable;

}

}

//如果usb设备支持高速运行，而现在却工作在全速，它就会通过点亮绿色指示灯来指示这种错误；

if (le16\_to\_cpu(udev->descriptor.bcdUSB) >= 0x0200

&& udev->speed == USB\_SPEED\_FULL && highspeed\_hubs != 0)

check\_highspeed (hub, udev, port1);

//如果通过这些判断后，usb设备还是正常的，则将它放到用于表示hub每个port的指针数据hub->children里，并通过usb\_new\_device去获取usb设备的配置信息，最后将它注册到系统中；

hub->ports[port1 - 1]->child = udev;

if (!status)

status = **usb\_new\_device**(udev);

status = hub\_power\_remaining(hub);

}

### usb\_alloc\_dev()

struct usb\_device \*usb\_alloc\_dev(struct usb\_device \*parent, struct usb\_bus \*bus, unsigned port1)

{

struct usb\_hcd \*usb\_hcd = bus\_to\_hcd(bus);

struct usb\_device \*dev = kzalloc(sizeof(\*dev), GFP\_KERNEL);

device\_initialize(&dev->dev);

dev->dev.bus = &usb\_bus\_type;

**dev->dev.type = &usb\_device\_type;**

dev->dev.groups = usb\_device\_groups;

dev->dev.dma\_mask = bus->controller->dma\_mask;

set\_dev\_node(&dev->dev, dev\_to\_node(bus->controller));

dev->state = USB\_STATE\_ATTACHED;

dev->lpm\_disable\_count = 1;

atomic\_set(&dev->urbnum, 0);

INIT\_LIST\_HEAD(&dev->ep0.urb\_list);

dev->ep0.desc.bLength = USB\_DT\_ENDPOINT\_SIZE;

dev->ep0.desc.bDescriptorType = USB\_DT\_ENDPOINT;

usb\_enable\_endpoint(dev, &dev->ep0, false);

dev->can\_submit = 1;

if (unlikely(!parent)) {

**dev\_set\_name(&dev->dev, "usb%d", bus->busnum);**

root\_hub = 1;

} else {

**dev\_set\_name(&dev->dev, "%d-%s", bus->busnum, dev->devpath);**

}

dev->portnum = port1;

dev->bus = bus;

dev->parent = parent;

INIT\_LIST\_HEAD(&dev->filelist);

if (root\_hub) /\* Root hub always ok [and always wired] \*/

dev->authorized = 1;

else {

dev->authorized = usb\_hcd->authorized\_default;

dev->wusb = usb\_bus\_is\_wusb(bus)? 1 : 0; //wusb: wireless usb

}

return dev;

}

### struct device\_type usb\_device\_type

struct device\_type usb\_device\_type = {

**.name = "usb\_device",**

.release = usb\_release\_dev,

.uevent = usb\_dev\_uevent,

.devnode = usb\_devnode,

#ifdef CONFIG\_PM

.pm = &usb\_device\_pm\_ops,

#endif

};

### hub\_port\_init()

static int hub\_port\_init (struct usb\_hub \*hub, struct usb\_device \*udev, int port1, int retry\_counter)

{

struct usb\_hcd \*hcd = bus\_to\_hcd(hdev->bus);

//设置用于读取port状态的时间间隔，root hub需要50ms,普通hub需要10ms，低速的usb设备需要200ms

unsigned delay = HUB\_SHORT\_RESET\_TIME;

if (!hdev->parent) {

delay = HUB\_ROOT\_RESET\_TIME;

if (port1 == hdev->bus->otg\_port)

hdev->bus->b\_hnp\_enable = 0;

}

if (oldspeed == USB\_SPEED\_LOW)

delay = HUB\_LONG\_RESET\_TIME;

//通过hub\_port\_reset来reset设备，reset机制为：通过set\_port\_feature来传输USB\_PORT\_FEAT\_RESET指令，设置成功后循环延时由之前确定的时间间隔后去读取port的status和change状态，要确保usb设备在reset后还能正常存在，如reset还能正常，则通过port的status状态位来确定usb设备的速度，循环延时总时间为500ms,而usb设备reset次数为5次，

retval = hub\_port\_reset(hub, port1, udev, delay, false);

//根据usb设备的速度来初始化endpont0的最大发送数据长度；

switch (udev->speed) {

case USB\_SPEED\_SUPER:

case USB\_SPEED\_WIRELESS: udev->ep0.desc.wMaxPacketSize = cpu\_to\_le16(512); break;

case USB\_SPEED\_HIGH: udev->ep0.desc.wMaxPacketSize = cpu\_to\_le16(64); break;

case USB\_SPEED\_FULL: udev->ep0.desc.wMaxPacketSize = cpu\_to\_le16(64); break;

case USB\_SPEED\_LOW: udev->ep0.desc.wMaxPacketSize = cpu\_to\_le16(8); break;

}

//如果hub为高速，而usb设备为低速或全速，则在它们之间需要有一个速度转换设备tt

if (hdev->tt) {

udev->tt = hdev->tt;

udev->ttport = hdev->ttport;

} else if (udev->speed != USB\_SPEED\_HIGH && hdev->speed == USB\_SPEED\_HIGH) {

udev->tt = &hub->tt;

udev->ttport = port1;

}

//通过获取usb设备的设备描述符来得到usb设备的endpoint0的最大发送数据长度，设备描述符实际上有18个字节，而endpoint0的maxpacketsize在设备描述符里的第8个字节位，所以只要获取设备描述符的前8个字节数据就可以了，但有些USB设备它并不支持只获取8个字节这样不完整的usb请求，为解决这种问题usb驱动里采用了两种策略去获取maxpacketsize，一种是windows作法，它直接向usb设备请求64字节数据，对于一些maxpacketsize为32或64的，它可以直接一次性把18个字节的设备描述符传输回来，而对于像low speed的设备它的maxpacketsize只有8个字节，就需要进行多次发送; 而Linux作法是先设置usb设备的地址，然后再发送8个字节数据请求，从返回的8个字节里获取endpoint0的maxpacketsize;

for (i = 0; i < GET\_DESCRIPTOR\_TRIES; (++i, msleep(100))) {

if (USE\_NEW\_SCHEME(retry\_counter) && !(hcd->driver->flags & HCD\_USB3)) {

struct usb\_device\_descriptor \*buf;

buf = kmalloc(GET\_DESCRIPTOR\_BUFSIZE, GFP\_NOIO); //#define GET\_DESCRIPTOR\_BUFSIZE 64

for (j = 0; j < 3; ++j) {

buf->bMaxPacketSize0 = 0;

usb\_control\_msg(udev, usb\_rcvaddr0pipe(),USB\_REQ\_GET\_DESCRIPTOR, USB\_DIR\_IN,

USB\_DT\_DEVICE << 8, 0,buf, GET\_DESCRIPTOR\_BUFSIZE,initial\_descriptor\_timeout);

udev->descriptor.bMaxPacketSize0 = buf->bMaxPacketSize0;

kfree(buf);

retval = hub\_port\_reset(hub, port1, udev, delay, false);

}

if (udev->wusb == 0) {

for (j = 0; j < SET\_ADDRESS\_TRIES; ++j) {

retval = hub\_set\_address(udev, devnum);

msleep(200);

}

msleep(10);

if (USE\_NEW\_SCHEME(retry\_counter) && !(hcd->driver->flags & HCD\_USB3))

break;

}

retval = usb\_get\_device\_descriptor(udev, 8);

break;

}

//根据usb设备描述符确定endpoint0的maxpacketsize,对于无线usb设备和超速usb设备，maxpacketsize为512，而其它类型就为从设备侧获取的maxpacketsize;

if (udev->descriptor.bMaxPacketSize0 == 0xff || udev->speed == USB\_SPEED\_SUPER) i = 512;

elsei = udev->descriptor.bMaxPacketSize0;

if (usb\_endpoint\_maxp(&udev->ep0.desc) != i) {

udev->ep0.desc.wMaxPacketSize = cpu\_to\_le16(i);

usb\_ep0\_reinit(udev);

}

//根据maxpacketsize去获取完整的设备描述符

retval = usb\_get\_device\_descriptor(udev, USB\_DT\_DEVICE\_SIZE);

if (udev->wusb == 0 && le16\_to\_cpu(udev->descriptor.bcdUSB) >= 0x0201)

retval = usb\_get\_bos\_descriptor(udev);

/\* notify HCD that we have a device connected and addressed \*/

if (hcd->driver->update\_device)

hcd->driver->update\_device(hcd, udev);

}

### usb\_new\_device()

int usb\_new\_device(struct usb\_device \*udev)

{

if (udev->parent) device\_init\_wakeup(&udev->dev, 0);

err = **usb\_enumerate\_device**(udev); /\* Read descriptors \*/

udev->dev.devt = MKDEV(USB\_DEVICE\_MAJOR, (((udev->bus->busnum-1) \* 128) + (udev->devnum-1)));

**announce\_device**(udev);

if (udev->serial) add\_device\_randomness(udev->serial, strlen(udev->serial));

if (udev->product) add\_device\_randomness(udev->product, strlen(udev->product));

if (udev->manufacturer) add\_device\_randomness(udev->manufacturer, strlen(udev->manufacturer));

err = **device\_add**(&udev->dev); //register

(void) usb\_create\_ep\_devs(&udev->dev, &udev->ep0, udev);

}

### usb\_enumerate\_device()

static int usb\_enumerate\_device(struct usb\_device \*udev)

{

if (udev->config == NULL) err = **usb\_get\_configuration**(udev);

/\* read the standard strings and cache them if present \*/

udev->product = usb\_cache\_string(udev, udev->descriptor.iProduct);

udev->manufacturer = usb\_cache\_string(udev, udev->descriptor.iManufacturer);

udev->serial = usb\_cache\_string(udev, udev->descriptor.iSerialNumber);

err = usb\_enumerate\_device\_otg(udev);

usb\_detect\_interface\_quirks(udev);

}

### usb\_get\_configuration() //详见USb host config

int usb\_get\_configuration(struct usb\_device \*dev)

{

int ncfg = dev->descriptor.bNumConfigurations;

**dev->config** = kzalloc(ncfg \* sizeof(struct usb\_host\_config), GFP\_KERNEL);

dev->rawdescriptors = kzalloc(ncfg \* sizeof(char \*), GFP\_KERNEL);

struct usb\_config\_descriptor \*desc = kmalloc(USB\_DT\_CONFIG\_SIZE, GFP\_KERNEL);

for (cfgno = 0; cfgno < ncfg; cfgno++) {

result = usb\_get\_descriptor(dev, USB\_DT\_CONFIG, cfgno, desc, USB\_DT\_CONFIG\_SIZE);

length = max((int) le16\_to\_cpu(desc->wTotalLength), USB\_DT\_CONFIG\_SIZE);

/\* Now that we know the length, get the whole thing \*/

unsigned char \*bigbuffer = kmalloc(length, GFP\_KERNEL);

result = **usb\_get\_descripto**r(dev, USB\_DT\_CONFIG, cfgno, bigbuffer, length);

dev->rawdescriptors[cfgno] = bigbuffer;

result = **usb\_parse\_configuration**(dev, cfgno, &dev->config[cfgno], bigbuffer, length);

if (result < 0) {

++cfgno;

goto err;

}

}

err:

kfree(desc);

dev->descriptor.bNumConfigurations = cfgno;

}

## Usb device驱动的注册

### usb\_device\_driver usb\_generic\_driver

struct usb\_device\_driver usb\_generic\_driver = {

**.name = "usb",**

**.probe = generic\_probe,**

.disconnect = generic\_disconnect,

#ifdef CONFIG\_PM

.suspend = generic\_suspend,

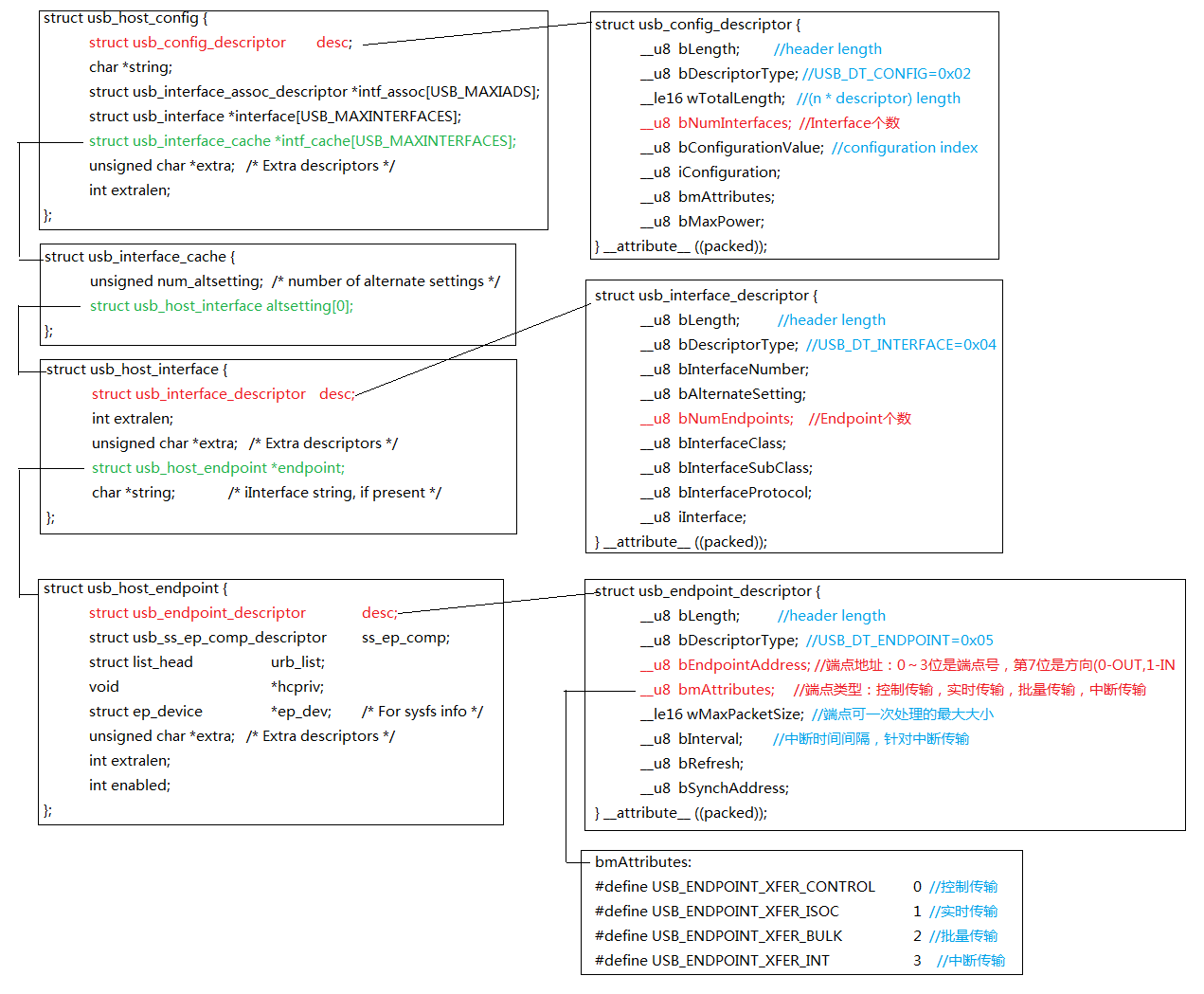
.resume = generic\_resume,

#endif

.supports\_autosuspend = 0,

};

# USB host config



## USB\_DT\_CONFIG: usb\_get\_descriptor

### usb\_get\_descriptor()

usb\_get\_descriptor(dev, USB\_DT\_CONFIG, cfgno, bigbuffer, length);

usb\_parse\_configuration(dev, cfgno, &dev->config[cfgno], bigbuffer, length);

//length = max((int) le16\_to\_cpu(desc->wTotalLength), USB\_DT\_CONFIG\_SIZE);

**//#define USB\_DT\_CONFIG\_SIZE 9;**

//bigbuffer = struct usb\_config\_descriptor;

### bDescriptorType

/\*

\* Descriptor types ... USB 2.0 spec table 9.5

\*/

#define USB\_DT\_DEVICE 0x01

#define USB\_DT\_CONFIG 0x02

#define USB\_DT\_STRING 0x03

#define USB\_DT\_INTERFACE 0x04

#define USB\_DT\_ENDPOINT 0x05

#define USB\_DT\_DEVICE\_QUALIFIER 0x06

#define USB\_DT\_OTHER\_SPEED\_CONFIG 0x07

#define USB\_DT\_INTERFACE\_POWER 0x08

/\* these are from a minor usb 2.0 revision (ECN) \*/

#define USB\_DT\_OTG 0x09

#define USB\_DT\_DEBUG 0x0a

#define USB\_DT\_INTERFACE\_ASSOCIATION 0x0b

/\* these are from the Wireless USB spec \*/

#define USB\_DT\_SECURITY 0x0c

#define USB\_DT\_KEY 0x0d

#define USB\_DT\_ENCRYPTION\_TYPE 0x0e

#define USB\_DT\_BOS 0x0f

#define USB\_DT\_DEVICE\_CAPABILITY 0x10

#define USB\_DT\_WIRELESS\_ENDPOINT\_COMP 0x11

#define USB\_DT\_WIRE\_ADAPTER 0x21

#define USB\_DT\_RPIPE 0x22

#define USB\_DT\_CS\_RADIO\_CONTROL 0x23

/\* From the T10 UAS specification \*/

#define USB\_DT\_PIPE\_USAGE 0x24

/\* From the USB 3.0 spec \*/

#define USB\_DT\_SS\_ENDPOINT\_COMP 0x30

### struct usb\_config\_descriptor //#define USB\_DT\_CONFIG\_SIZE 9

struct usb\_config\_descriptor {

\_\_u8 bLength;

\_\_u8 bDescriptorType;

\_\_le16 wTotalLength;

\_\_u8 bNumInterfaces; //Interface的个数

\_\_u8 bConfigurationValue;

\_\_u8 iConfiguration;

\_\_u8 bmAttributes;

\_\_u8 bMaxPower;

} \_\_attribute\_\_ ((packed))

### usb\_parse\_configuration()

static int usb\_parse\_configuration(struct usb\_device \*dev, int cfgidx, struct usb\_host\_config \*config, unsigned char \*buffer, int size)

{

memcpy(&**config->desc**, buffer, USB\_DT\_CONFIG\_SIZE);

cfgno = config->desc.bConfigurationValue;

buffer += config->desc.bLength;

size -= config->desc.bLength;

nintf = nintf\_orig = config->desc.bNumInterfaces;

for ((buffer2 = buffer, size2 = size); size2 > 0; (buffer2 += header->bLength, size2 -= header->bLength)) {

header = (struct usb\_descriptor\_header \*) buffer2;

if (header->bDescriptorType == USB\_DT\_INTERFACE) {

struct usb\_interface\_descriptor \*d = (struct usb\_interface\_descriptor \*) header;

int inum = d->bInterfaceNumber;

for (i = 0; i < n; ++i) if (inums[i] == inum) break;

if (i < n) { if (nalts[i] < 255) ++nalts[i];}

else if (n < USB\_MAXINTERFACES) { inums[n] = inum; nalts[n] = 1; ++n;}

}

else if (header->bDescriptorType == USB\_DT\_INTERFACE\_ASSOCIATION) {

config->intf\_assoc[iad\_num] = (struct usb\_interface\_assoc\_descriptor\*)header;

iad\_num++;

}

} /\* for ((buffer2 = buffer, size2 = size); ...) \*/

size = buffer2 - buffer;

**config->desc.wTotalLength** = cpu\_to\_le16(buffer2 - buffer0);

**config->desc.bNumInterfaces** = nintf = n;

/\* Allocate the usb\_interface\_caches and altsetting arrays \*/

for (i = 0; i < nintf; ++i) {

j = nalts[i];

len = sizeof(\*intfc) + sizeof(struct usb\_host\_interface) \* j;

**config->intf\_cache[i]** = intfc = kzalloc(len, GFP\_KERNEL);

kref\_init(&intfc->ref);

}

**config->extra** = buffer;

i = **find\_next\_descriptor**(buffer, size, **USB\_DT\_INTERFACE**, USB\_DT\_INTERFACE, &n);

**config->extralen** = i; buffer += i; size -= i;

/\* Parse all the interface/altsetting descriptors \*/

while (size > 0) {

retval = **usb\_parse\_interface**(ddev, cfgno, config, buffer, size, inums, nalts);

buffer += retval; size -= retval;

}

}

## USB\_DT\_INTERFACE : find\_next\_descriptor

### struct usb\_interface\_descriptor //#define USB\_DT\_INTERFACE\_SIZE 9

struct usb\_interface\_descriptor {

\_\_u8 bLength;

\_\_u8 bDescriptorType;

\_\_u8 bInterfaceNumber;

\_\_u8 bAlternateSetting;

\_\_u8 bNumEndpoints; //endpoint的个数

\_\_u8 bInterfaceClass;

\_\_u8 bInterfaceSubClass;

\_\_u8 bInterfaceProtocol;

\_\_u8 iInterface;

} \_\_attribute\_\_ ((packed));

### usb\_parse\_interface()

//usb\_parse\_interface(ddev, cfgno, config, buffer, size, inums, nalts);

static int usb\_parse\_interface(struct device \*ddev, int cfgno,

struct usb\_host\_config \*config, unsigned char \*buffer, int size, u8 inums[], u8 nalts[])

{

d = (struct usb\_interface\_descriptor \*) buffer;

buffer += d->bLength; size -= d->bLength;

struct usb\_interface\_cache \*intfc = NULL;

inum = d->bInterfaceNumber;

for (i = 0; i < config->desc.bNumInterfaces; ++i) if (inums[i] == inum) { **intfc = config->intf\_cache[i]**; break;}

struct usb\_host\_interface \*alt;

asnum = d->bAlternateSetting;

for ((i = 0, **alt = &intfc->altsetting[0])**; i < intfc->num\_altsetting; (++i, **++alt**)) {

if (alt->desc.bAlternateSetting == asnum) {

goto skip\_to\_next\_interface\_descriptor;

}

}

**++intfc->num\_altsetting;**

memcpy(&**alt->desc**, d, USB\_DT\_INTERFACE\_SIZE);

**alt->extra** = buffer;

i = find\_next\_descriptor(buffer, size, **USB\_DT\_ENDPOINT,** USB\_DT\_INTERFACE, &n);

**alt->extralen** = i; buffer += i; size -= i;

num\_ep = num\_ep\_orig = alt->desc.bNumEndpoints;

alt->desc.bNumEndpoints = 0; /\* Use as a counter \*/

if (num\_ep > 0) {

len = sizeof(struct usb\_host\_endpoint) \* num\_ep;

**alt->endpoint** = kzalloc(len, GFP\_KERNEL);

}

while (size > 0) {

if (((struct usb\_descriptor\_header \*) buffer)->bDescriptorType == USB\_DT\_INTERFACE) break;

retval = **usb\_parse\_endpoint**(ddev, cfgno, inum, asnum, alt, num\_ep, buffer, size);

++n; buffer += retval; size -= retval;

}

return buffer - buffer0;

}

## USB\_DT\_ENDPOINT : find\_next\_descriptor

### struct usb\_endpoint\_descriptor

#define USB\_DT\_ENDPOINT\_SIZE 7

#define USB\_DT\_ENDPOINT\_AUDIO\_SIZE 9 /\* Audio extension \*/

struct usb\_endpoint\_descriptor {

\_\_u8 bLength;

\_\_u8 bDescriptorType;

\_\_u8 bEndpointAddress;

\_\_u8 bmAttributes;

\_\_le16 wMaxPacketSize;

\_\_u8 bInterval;

\_\_u8 bRefresh;

\_\_u8 bSynchAddress;

} \_\_attribute\_\_ ((packed));

### struct usb\_host\_endpoint

struct usb\_host\_endpoint {

struct usb\_endpoint\_descriptor desc;

struct usb\_ss\_ep\_comp\_descriptor ss\_ep\_comp;

struct list\_head urb\_list;

void \*hcpriv;

struct ep\_device \*ep\_dev; /\* For sysfs info \*/

unsigned char \*extra; /\* Extra descriptors \*/

int extralen;

int enabled;

};

### usb\_parse\_endpoint()

static int usb\_parse\_endpoint(struct device \*ddev, int cfgno, int inum,

int asnum, struct usb\_host\_interface \*ifp, int num\_ep, unsigned char \*buffer, int size)

{

struct usb\_endpoint\_descriptor \*d = (struct usb\_endpoint\_descriptor \*) buffer;

buffer += d->bLength;

size -= d->bLength;

struct usb\_host\_endpoint \***endpoint = &ifp->endpoint**[ifp->desc.bNumEndpoints];

++**ifp->desc.bNumEndpoints;**

memcpy(&**endpoint->desc,** d, n);

INIT\_LIST\_HEAD(&endpoint->urb\_list);

/\* Fix up bInterval values outside the legal range. Use 32 ms if no proper value can be guessed. \*/

i = 0; /\* i = min, j = max, n = default \*/

j = 255;

if (usb\_endpoint\_xfer\_int(d)) {

i = 1;

switch (to\_usb\_device(ddev)->speed) {

case USB\_SPEED\_SUPER:

case USB\_SPEED\_HIGH: j = 16; n = fls(d->bInterval\*8); if (n == 0) n = 9;break;

/\* 32 ms = 2^(9-1) uframes \*/

default: n = 32; break;

}

} else if (usb\_endpoint\_xfer\_isoc(d)) {

i = 1; j = 16;

switch (to\_usb\_device(ddev)->speed) {

case USB\_SPEED\_HIGH: n = 9; break; /\* 32 ms = 2^(9-1) uframes \*/

default: n = 6; break; /\* 32 ms = 2^(6-1) frames \*/

}

}

if (d->bInterval < i || d->bInterval > j) **endpoint->desc.bInterval =** n;

if (to\_usb\_device(ddev)->speed == USB\_SPEED\_LOW && usb\_endpoint\_xfer\_bulk(d)) {

**endpoint->desc.bmAttributes** = USB\_ENDPOINT\_XFER\_INT;

**endpoint->desc.bInterval** = 1;

if (usb\_endpoint\_maxp(&endpoint->desc) > 8) **endpoint->desc.wMaxPacketSize** = cpu\_to\_le16(8);

}

if (to\_usb\_device(ddev)->speed == USB\_SPEED\_SUPER)

**usb\_parse\_ss\_endpoint\_companion**(ddev, cfgno, inum, asnum, endpoint, buffer, size);

**endpoint->extra** = buffer;

i = find\_next\_descriptor(buffer, size, USB\_DT\_ENDPOINT, USB\_DT\_INTERFACE, &n);

**endpoint->extralen** = i;

return (buffer - buffer0 + i);

}

# Uvcvideo驱动初始化

## uvc\_init(0

module\_init(uvc\_init);

static int \_\_init uvc\_init(void)

{

uvc\_debugfs\_init();

ret = usb\_register(&uvc\_driver.driver);

}

## struct uvc\_driver uvc\_driver

struct uvc\_driver uvc\_driver = {

.driver = {

**.name = "uvcvideo",**

**.probe = uvc\_probe,**

.disconnect = uvc\_disconnect,

.suspend = uvc\_suspend,

.resume = uvc\_resume,

.reset\_resume = uvc\_reset\_resume,

.id\_table = uvc\_ids,

.supports\_autosuspend = 1,

},

};

## usb\_register()

#define usb\_register(driver) \

usb\_register\_driver(driver, THIS\_MODULE, KBUILD\_MODNAME)

int usb\_register\_driver(struct usb\_driver \*new\_driver, struct module \*owner, const char \*mod\_name)

{

new\_driver->drvwrap.for\_devices = 0;

new\_driver->drvwrap**.driver.name = (char \*) new\_driver->name**;

new\_driver->drvwrap.driver.bus = &usb\_bus\_type;

new\_driver->drvwrap.driver.**probe = usb\_probe\_interface**;

new\_driver->drvwrap.driver.remove = usb\_unbind\_interface;

new\_driver->drvwrap.driver.owner = owner;

new\_driver->drvwrap.driver.mod\_name = mod\_name;

spin\_lock\_init(&new\_driver->dynids.lock);

INIT\_LIST\_HEAD(&new\_driver->dynids.list);

retval = driver\_register(&**new\_driver->drvwrap.driver**);

}

# Probe

## Usb device probe

### usb\_probe\_device()

static int usb\_probe\_device(struct device \*dev)

{

struct usb\_device\_driver \*udriver = to\_usb\_device\_driver(dev->driver);

error = udriver->probe(udev);

}

### generic\_probe()

static int generic\_probe(struct usb\_device \*udev)

{

c = usb\_choose\_configuration(udev);

if (c >= 0) {

err = usb\_set\_configuration(udev, c);

}

usb\_notify\_add\_device(udev);

}

### usb\_set\_configuration()

int usb\_set\_configuration(struct usb\_device \*dev, int configuration)

{

struct **usb\_host\_config** \*cp = &dev->config[i];

if (cp) {

nintf = cp->desc.bNumInterfaces;

new\_interfaces = kmalloc(nintf \* sizeof(\*new\_interfaces), GFP\_NOIO);

for (; n < nintf; ++n)

**new\_interfaces[n] = kzalloc**(sizeof(struct **usb\_interface**), GFP\_NOIO);

}

ret = usb\_autoresume\_device(dev);

ret = usb\_hcd\_alloc\_bandwidth(dev, cp, NULL, NULL);

for (i = 0; i < nintf; ++i) {

struct usb\_interface \*intf;

**cp->interface[i] = intf = new\_interfaces[i];**

struct usb\_interface\_cache \*intfc = cp->intf\_cache[i];

struct usb\_host\_interface \*alt = &intf->altsetting[0];

intf->altsetting = intfc->altsetting;

intf->num\_altsetting = intfc->num\_altsetting;

kref\_get(&intfc->ref);

intf->intf\_assoc = find\_iad(dev, cp, alt->desc.bInterfaceNumber);

intf->cur\_altsetting = alt;

usb\_enable\_interface(dev, intf, true);

intf->dev.parent = &dev->dev;

intf->dev.driver = NULL;

intf->dev.bus = &usb\_bus\_type;

intf->dev.type = &usb\_if\_device\_type;

intf->dev.groups = usb\_interface\_groups;

intf->dev.dma\_mask = dev->dev.dma\_mask;

INIT\_WORK(&intf->reset\_ws, \_\_usb\_queue\_reset\_device);

intf->minor = -1;

device\_initialize(&intf->dev);

pm\_runtime\_no\_callbacks(&intf->dev);

dev\_set\_name(&intf->dev, "**%d-%s:%d.%d"**,dev->bus->busnum, dev->devpath,configuration, alt->desc.bInterfaceNumber);

}

kfree(new\_interfaces);

ret = usb\_control\_msg(dev, usb\_sndctrlpipe(dev, 0), **USB\_REQ\_SET\_CONFIGURATION**, 0, configuration, 0, NULL, 0, USB\_CTRL\_SET\_TIMEOUT);

dev->actconfig = cp;

usb\_set\_device\_state(dev, USB\_STATE\_CONFIGURED);

for (i = 0; i < nintf; ++i) {

struct usb\_interface \*intf = cp->interface[i];

ret = **device\_add**(&intf->dev);

**create\_intf\_ep\_devs**(intf);

}

usb\_autosuspend\_device(dev);

}

### create\_intf\_ep\_devs()

static int create\_intf\_ep\_devs(struct usb\_interface \*intf)

{

struct usb\_host\_interface \*alt = intf->cur\_altsetting;

for (i = 0; i < alt->desc.bNumEndpoints; ++i)

(void) usb\_create\_ep\_devs(&intf->dev, &alt->endpoint[i], udev);

intf->ep\_devs\_created = 1;

}

### usb\_create\_ep\_devs()

int usb\_create\_ep\_devs(struct device \*parent, struct usb\_host\_endpoint \*endpoint, struct usb\_device \*udev)

{

struct ep\_device \*ep\_dev = kzalloc(sizeof(\*ep\_dev), GFP\_KERNEL);

ep\_dev->desc = &endpoint->desc;

ep\_dev->udev = udev;

ep\_dev->dev.groups = ep\_dev\_groups;

ep\_dev->dev.type = &usb\_ep\_device\_type;

ep\_dev->dev.parent = parent;

dev\_set\_name(&ep\_dev->dev, "**ep\_**%02x", endpoint->desc.bEndpointAddress);

retval = **device\_register**(&ep\_dev->dev);

device\_enable\_async\_suspend(&ep\_dev->dev);

endpoint->ep\_dev = ep\_dev;

}

## Usb driver probe

### usb\_probe\_interface()

static int usb\_probe\_interface(struct device \*dev)

{

struct usb\_driver \*driver = to\_usb\_driver(dev->driver);

struct usb\_interface \*intf = to\_usb\_interface(dev);

id = usb\_match\_id(intf, driver->id\_table);

if (!id) id = usb\_match\_dynamic\_id(intf, driver);

error = usb\_autoresume\_device(udev);

intf->condition = USB\_INTERFACE\_BINDING;

if (intf->needs\_altsetting0) {

error = **usb\_set\_interface**(udev, intf->altsetting[0]. desc.bInterfaceNumber, 0);

intf->needs\_altsetting0 = 0;

}

error = **driver->probe**(intf, id);

intf->condition = USB\_INTERFACE\_BOUND;

usb\_autosuspend\_device(udev);

}

### usb\_set\_interface()

int usb\_set\_interface(struct usb\_device \*dev, int interface, int alternate)

{

struct usb\_interface \*iface = usb\_ifnum\_to\_if(dev, interface); //config->interface[i];

struct usb\_host\_interface \*alt = usb\_altnum\_to\_altsetting(iface, alternate); //intf->altsetting[i]

ret = usb\_hcd\_alloc\_bandwidth(dev, NULL, iface->cur\_altsetting, alt);

ret = usb\_control\_msg(dev, usb\_sndctrlpipe(dev, 0),

**USB\_REQ\_SET\_INTERFACE**, USB\_RECIP\_INTERFACE, alternate, interface, NULL, 0, 5000);

usb\_disable\_interface(dev, iface, true);

iface->cur\_altsetting = alt;

usb\_enable\_interface(dev, iface, true);

}

### uvc\_probe()

static int uvc\_probe(struct usb\_interface \*intf, const struct usb\_device\_id \*id)

{

struct **uvc\_device** \*dev = **kzalloc**(sizeof \*dev, GFP\_KERNEL);

INIT\_LIST\_HEAD(&dev->entities);

INIT\_LIST\_HEAD(&dev->chains);

INIT\_LIST\_HEAD(&dev->streams);

atomic\_set(&dev->nstreams, 0);

atomic\_set(&dev->users, 0);

atomic\_set(&dev->nmappings, 0);

dev->udev = usb\_get\_dev(udev);

dev->intf = usb\_get\_intf(intf);

dev->intfnum = intf->cur\_altsetting->desc.bInterfaceNumber;

dev->quirks = (uvc\_quirks\_param == -1) ? id->driver\_info : uvc\_quirks\_param;

strlcpy(dev->name, udev->product, sizeof dev->name);

uvc\_parse\_control(dev);

v4l2\_device\_register(&intf->dev, &dev->vdev);

uvc\_ctrl\_init\_device(dev);

uvc\_scan\_device(dev);

**uvc\_register\_chains**(dev);

usb\_set\_intfdata(intf, dev);

**uvc\_status\_init**(dev);

uvcvideo\_create\_sysfs(&dev->vdev);

usb\_enable\_autosuspend(udev);

}

### uvc\_register\_chains()

static int uvc\_register\_chains(struct uvc\_device \*dev)

{

struct uvc\_video\_chain \*chain;

list\_for\_each\_entry(chain, &dev->chains, list)

ret = uvc\_register\_terms(dev, chain);

}

static int uvc\_register\_terms(struct uvc\_device \*dev, struct uvc\_video\_chain \*chain)

{

struct uvc\_streaming \*stream;

struct uvc\_entity \*term;

list\_for\_each\_entry(term, &chain->entities, chain) {

stream = uvc\_stream\_by\_id(dev, term->id);

stream->chain = chain;

ret = uvc\_register\_video(dev, stream);

term->vdev = stream->vdev;

}

}

### uvc\_register\_video()

static int uvc\_register\_video(struct uvc\_device \*dev, struct uvc\_streaming \*stream)

{

ret = **uvc\_video\_init**(stream);

uvc\_debugfs\_init\_stream(stream);

struct video\_device \*vdev = video\_device\_alloc();

vdev\_addr = vdev;

vdev->v4l2\_dev = &dev->vdev;

vdev->fops =**&uvc\_fops**;

vdev->release = uvc\_release;

vdev->prio = &stream->chain->prio;

set\_bit(V4L2\_FL\_USE\_FH\_PRIO, &vdev->flags);

strlcpy(vdev->name, dev->name, sizeof vdev->name);

stream->vdev = vdev;

video\_set\_drvdata(vdev, stream);

ret = video\_register\_device(vdev, VFL\_TYPE\_GRABBER, 1);

stream->chain->caps |= V4L2\_CAP\_VIDEO\_CAPTURE;

}

### struct v4l2\_file\_operations uvc\_fops

const struct v4l2\_file\_operations uvc\_fops = {

.owner = THIS\_MODULE,

**.open = uvc\_v4l2\_open,**

.release = uvc\_v4l2\_release,

**.unlocked\_ioctl = uvc\_v4l2\_ioctl,**

#ifdef CONFIG\_COMPAT

**.compat\_ioctl32 = uvc\_v4l2\_compat\_ioctl32,**

**#endif**

.read = uvc\_v4l2\_read,

**.mmap = uvc\_v4l2\_mmap,**

.poll = uvc\_v4l2\_poll,

#ifndef CONFIG\_MMU

.get\_unmapped\_area = uvc\_v4l2\_get\_unmapped\_area,

#endif

};

### uvc\_video\_init()

int uvc\_video\_init(struct uvc\_streaming \*stream)

{

struct uvc\_streaming\_control \*probe = &stream->ctrl;

ret = **uvc\_queue\_init**(&stream->queue, stream->type, !uvc\_no\_drop\_param);

usb\_set\_interface(stream->dev->udev, stream->intfnum, 0);

if (uvc\_get\_video\_ctrl(stream, probe, 1, UVC\_GET\_DEF) == 0)

uvc\_set\_video\_ctrl(stream, probe, 1);

ret = uvc\_get\_video\_ctrl(stream, probe, 1, UVC\_GET\_CUR);

for (i = stream->nformats; i > 0; --i) {

struct uvc\_format \*format = &stream->format[i-1];

if (format->index == probe->bFormatIndex) break;

}

for (i = format->nframes; i > 0; --i) {

struct uvc\_frame \*frame = &format->frame[i-1];

if (frame->bFrameIndex == probe->bFrameIndex) break;

}

probe->bFormatIndex = format->index;

probe->bFrameIndex = frame->bFrameIndex;

stream->def\_format = format;

stream->cur\_format = format;

stream->cur\_frame = frame;

if (stream->type == V4L2\_BUF\_TYPE\_VIDEO\_CAPTURE) {

if (stream->dev->quirks & UVC\_QUIRK\_BUILTIN\_ISIGHT) stream->decode = uvc\_video\_decode\_isight;

else if (stream->intf->num\_altsetting > 1) stream->decode = uvc\_video\_decode\_isoc;

else stream->decode = uvc\_video\_decode\_bulk;

}

}

### uvc\_queue\_init()

int uvc\_queue\_init(struct uvc\_video\_queue \*queue, enum v4l2\_buf\_type type, int drop\_corrupted)

{

queue->queue.type = type;

queue->queue.io\_modes = VB2\_MMAP | VB2\_USERPTR | VB2\_DMABUF;

queue->queue.drv\_priv = queue;

queue->queue.buf\_struct\_size = sizeof(struct uvc\_buffer);

queue->queue**.ops = &uvc\_queue\_qops;**

#ifdef CONFIG\_ASOC\_CAMERA

queue->queue.**mem\_ops = &asoc\_vb2\_ion\_memops;**

#else

queue->queue.mem\_ops = &vb2\_vmalloc\_memops;

#endif

queue->queue.timestamp\_type = V4L2\_BUF\_FLAG\_TIMESTAMP\_MONOTONIC;

ret = **vb2\_queue\_init**(&queue->queue);

mutex\_init(&queue->mutex);

spin\_lock\_init(&queue->irqlock);

INIT\_LIST\_HEAD(&queue->irqqueue);

queue->flags = drop\_corrupted ? UVC\_QUEUE\_DROP\_CORRUPTED : 0;

}

### struct vb2\_ops uvc\_queue\_qops

static struct vb2\_ops uvc\_queue\_qops = {

.queue\_setup = uvc\_queue\_setup,

.buf\_prepare = uvc\_buffer\_prepare,

.buf\_queue = uvc\_buffer\_queue,

.buf\_finish = uvc\_buffer\_finish,

.wait\_prepare = uvc\_wait\_prepare,

.wait\_finish = uvc\_wait\_finish,

};

### uvc\_status\_init()

#define UVC\_MAX\_STATUS\_SIZE 16

int uvc\_status\_init(struct uvc\_device \*dev)

{

struct usb\_host\_endpoint \*ep = dev->int\_ep;

**uvc\_input\_init**(dev);

dev->status = kzalloc(UVC\_MAX\_STATUS\_SIZE, GFP\_KERNEL);

dev->int\_urb = **usb\_alloc\_urb**(0, GFP\_KERNEL);

pipe = usb\_rcvintpipe(dev->udev, ep->desc.bEndpointAddress);

interval = ep->desc.bInterval;

**usb\_fill\_int\_urb**(dev->int\_urb, dev->udev, pipe,

dev->status, UVC\_MAX\_STATUS\_SIZE, **uvc\_status\_complete**, dev, interval);

}

### uvc\_input\_init()

static int uvc\_input\_init(struct uvc\_device \*dev)

{

struct input\_dev \***input = input\_allocate\_device**();

usb\_make\_path(dev->udev, dev->input\_phys, sizeof(dev->input\_phys));

strlcat(dev->input\_phys, "/button", sizeof(dev->input\_phys));

input->name = dev->name;

input->phys = dev->input\_phys;

usb\_to\_input\_id(dev->udev, &input->id);

input->dev.parent = &dev->intf->dev;

\_\_set\_bit(EV\_KEY, input->evbit);

\_\_set\_bit(KEY\_CAMERA, input->keybit);

ret = **input\_register\_device**(input);

**dev->input = input;**

}

### usb\_alloc\_urb()

struct urb \*usb\_alloc\_urb(int iso\_packets, gfp\_t mem\_flags)

{

struct urb \*urb = kmalloc(sizeof(struct urb) + iso\_packets\*sizeof(struct usb\_iso\_packet\_descriptor),mem\_flags);

usb\_init\_urb(urb);

return urb;

}

//**usb\_fill\_int\_urb**(dev->int\_urb, dev->udev, pipe, \

// dev->status, UVC\_MAX\_STATUS\_SIZE, **uvc\_status\_complete**, dev, interval);

static inline void usb\_fill\_int\_urb(struct urb \*urb,struct usb\_device \*dev,unsigned int pipe,

void \*transfer\_buffer,int buffer\_length,usb\_complete\_t complete\_fn,void \*context,int interval)

{

urb->dev = dev;

urb->pipe = pipe;

urb->transfer\_buffer = transfer\_buffer;

urb->transfer\_buffer\_length = buffer\_length;

**urb->complete = complete\_fn;**

urb->context = context;

if (dev->speed == USB\_SPEED\_HIGH || dev->speed == USB\_SPEED\_SUPER)

urb->interval = 1 << (interval - 1);

else

urb->interval = interval;

urb->start\_frame = -1;

}

# Buffer

## Uvc\_buffer state:

enum uvc\_buffer\_state {

UVC\_BUF\_STATE\_IDLE = 0,

UVC\_BUF\_STATE\_QUEUED = 1,

UVC\_BUF\_STATE\_ACTIVE = 2,

UVC\_BUF\_STATE\_READY = 3,

UVC\_BUF\_STATE\_DONE = 4,

UVC\_BUF\_STATE\_ERROR = 5,

};

## Uvc\_buffer error:

enum uvc\_buffer\_err {

UVC\_BUF\_ERR\_NONE = 0,

UVC\_BUF\_ERR\_ISOFRAM\_LOST = 1,

UVC\_BUF\_ERR\_ISOFRAM\_ERR = 2,

UVC\_BUF\_ERR\_OVERFLOW = 3,

UVC\_BUF\_ERR\_NOTFULL = 4,

};

## Isoc

Urb：**urb->transfer\_buffer + urb->iso\_frame\_desc[i].offset;**

**Fid相同的packet数据属于同一个frame的数据。**

[14098.690958] [0] [L=1024] 0c cc 01 a6 a8 00 af 2c a9 00 d5 06 [fid:0][eof:0][sti:0][err:0]

[14098.695500] [0] [L=2048] 0c cc 01 a6 a8 00 5b 34 aa 00 d9 06 [fid:0][eof:0][sti:0][err:0]

[14098.699974] [0] [L=1024] 0c cc 01 a6 a8 00 06 3c ab 00 de 06 [fid:0][eof:0][sti:0][err:0]

[14098.704499] [0] [L=2048] 0c ce 01 a6 a8 00 b1 43 ac 00 e2 06 [fid:0][eof:2][sti:0][err:0]

[14098.708932] [0] [L= 76] 0c ce 01 a6 a8 00 5c 4b ad 00 e7 06 [fid:0][eof:2][sti:0][err:0]

[14098.713287] [0] [L= 12] 0c cc 01 a6 a8 00 06 53 ae 00 eb 06 [fid:0][eof:0][sti:0][err:0]

[14098.717745] [0] [L= 24] 0c cc 01 a6 a8 00 5d 53 af 00 ef 06 [fid:0][eof:0][sti:0][err:0]

[14098.722122] [0] [L= 12] 0c cc 01 a6 a8 00 b4 53 b0 00 f4 06 [fid:0][eof:0][sti:0][err:0]

[14098.726503] [0] [L=1024] 0c cd 62 46 b0 00 0d 54 b1 00 f8 06 [fid:1][eof:0][sti:0][err:0]

[14098.730967] [0] [L=1024] 0c cd 62 46 b0 00 66 54 b2 00 fd 06 [fid:1][eof:0][sti:0][err:0]

[14098.735367] [0] [L=1024] 0c cd 62 46 b0 00 10 5c b3 00 01 07 [fid:1][eof:0][sti:0][err:0]

[14098.739838] [0] [L=1024] 0c cd 62 46 b0 00 67 5c b4 00 05 07 [fid:1][eof:0][sti:0][err:0]

[14098.744239] [0] [L=1024] 0c cf 62 46 b0 00 12 64 b5 00 0a 07 [fid:1][eof:2][sti:0][err:0]

[14098.748707] [0] [L=1024] 0c cf 62 46 b0 00 69 64 b6 00 0e 07 [fid:1][eof:2][sti:0][err:0]

[14098.753066] [0] [L= 76] 0c cf 62 46 b0 00 13 6c b7 00 13 07 [fid:1][eof:2][sti:0][err:0]

[14098.757464] [0] [L=1024] 0c cc c3 e6 b7 00 6a 6c b8 00 17 07 [fid:0][eof:0][sti:0][err:0]

[14098.761912] [0] [L=1024] 0c cc c3 e6 b7 00 c2 6c b9 00 1c 07 [fid:0][eof:0][sti:0][err:0]

[14098.766313] [0] [L=1024] 0c cc c3 e6 b7 00 6c 74 ba 00 20 07 [fid:0][eof:0][sti:0][err:0]

[14098.770759] [0] [L=1024] 0c cc c3 e6 b7 00 c3 74 bb 00 24 07 [fid:0][eof:0][sti:0][err:0]

[14098.775143] [0] [L=1024] 0c ce c3 e6 b7 00 1b 75 bc 00 29 07 [fid:0][eof:2][sti:0][err:0]

[14098.779605] [0] [L=1024] 0c ce c3 e6 b7 00 73 75 bd 00 2d 07 [fid:0][eof:2][sti:0][err:0]

[14098.784063] [0] [L= 88] 0c ce c3 e6 b7 00 1c 7d be 00 32 07 [fid:0][eof:2][sti:0][err:0]

[14098.788491] [0] [L= 12] 0c cc c3 e6 b7 00 c6 84 bf 00 36 07 [fid:0][eof:0][sti:0][err:0]

[14098.792891] [0] [L=1024] 0c cd 23 87 bf 00 1f 85 c0 00 3b 07 [fid:1][eof:0][sti:0][err:0]

[14098.797291] [0] [L=1024] 0c cd 23 87 bf 00 78 85 c1 00 3f 07 [fid:1][eof:0][sti:0][err:0]

[14098.801740] [0] [L=1024] 0c cd 23 87 bf 00 22 8d c2 00 43 07 [fid:1][eof:0][sti:0][err:0]

[14098.806243] [0] [L=2048] 0c cd 23 87 bf 00 79 8d c3 00 48 07 [fid:1][eof:0][sti:0][err:0]

[14098.810692] [0] [L=1024] 0c cf 23 87 bf 00 24 95 c4 00 4c 07 [fid:1][eof:2][sti:0][err:0]

[14098.815053] [0] [L= 76] 0c cf 23 87 bf 00 ce 9c c5 00 51 07 [fid:1][eof:2][sti:0][err:0]

[14098.819454] [0] [L= 12] 0c cd 23 87 bf 00 26 9d c6 00 55 07 [fid:1][eof:0][sti:0][err:0]

### uvc\_video\_decode\_isoc()

static void uvc\_video\_decode\_isoc(struct urb \*urb, struct uvc\_streaming \*stream, **struct uvc\_buffer \*buf**)

{

for (i = 0; i < urb->number\_of\_packets; ++i) {

if (urb->iso\_frame\_desc[i].status < 0) {

if (buf != NULL){

**buf->error** = **UVC\_BUF\_ERR\_ISOFRAM\_LOST**;

uvc\_trace(UVC\_TRACE\_FRAME\_ERR, "## USB isochronous frame lost .\n");

}

continue;

}

/\* Decode the payload header. \*/

**u8 \*mem = urb->transfer\_buffer + urb->iso\_frame\_desc[i].offset;**

do {

ret = **uvc\_video\_decode\_start**(stream, **buf**, mem,urb->iso\_frame\_desc[i].actual\_length);

if (ret == -EAGAIN) **buf** = uvc\_queue\_next\_buffer(&stream->queue,buf);

} while (ret == -EAGAIN);

if (ret < 0) continue;

/\* Decode the payload data. \*/

**uvc\_video\_decode\_data**(stream, **buf**, mem + ret,urb->iso\_frame\_desc[i].actual\_length - ret);

/\* Process the header again. \*/

**uvc\_video\_decode\_end**(stream, **buf**, mem,urb->iso\_frame\_desc[i].actual\_length);

if (buf->state == UVC\_BUF\_STATE\_READY) {

if (buf->length != buf->bytesused &&!(stream->cur\_format->flags & UVC\_FMT\_FLAG\_COMPRESSED))

buf->error = **UVC\_BUF\_ERR\_NOTFULL**;

**buf** = **uvc\_queue\_next\_buffer**(&stream->queue, buf);

}

}

}

### uvc\_video\_decode\_start()

static int uvc\_video\_decode\_start(struct uvc\_streaming \*stream, struct uvc\_buffer \*buf, const \_\_u8 \*data, int len)

{

\_\_u8 fid = data[1] & UVC\_STREAM\_FID;

if (stream->last\_fid != fid) { //

stream->sequence++;

if (stream->sequence) **uvc\_video\_stats\_update**(stream);

}

**uvc\_video\_clock\_decode**(stream, buf, data, len);

**uvc\_video\_stats\_decode**(stream, data, len);

if (data[1] & UVC\_STREAM\_ERR) buf->**error = UVC\_BUF\_ERR\_ISOFRAM\_ERR**;

if (buf->state != UVC\_BUF\_STATE\_ACTIVE) {

struct timespec ts;

if (uvc\_clock\_param == CLOCK\_MONOTONIC) ktime\_get\_ts(&ts);

else ktime\_get\_real\_ts(&ts);

buf->buf.v4l2\_buf.sequence = stream->sequence;

buf->buf.v4l2\_buf.timestamp.tv\_sec = ts.tv\_sec;

buf->buf.v4l2\_buf.timestamp.tv\_usec =ts.tv\_nsec / NSEC\_PER\_USEC;

buf->**state = UVC\_BUF\_STATE\_ACTIVE**;

}

if (fid != stream->last\_fid && buf->bytesused != 0) {

buf->state = UVC\_BUF\_STATE\_READY;

if((buf->bytesused != buf->length) &&!(stream->cur\_format->flags & UVC\_FMT\_FLAG\_COMPRESSED))

buf->**error = UVC\_BUF\_ERR\_NOTFULL**;

return -EAGAIN;

}

stream->last\_fid = fid;

return data[0];

}

### uvc\_video\_stats\_update()

static void uvc\_video\_stats\_update(struct uvc\_streaming \*stream)

{

struct uvc\_stats\_frame \*frame = &stream->stats.frame;

stream->stats.stream.nb\_frames++;

stream->stats.stream.nb\_packets += stream->stats.frame.nb\_packets;

stream->stats.stream.nb\_empty += stream->stats.frame.nb\_empty;

stream->stats.stream.nb\_errors += stream->stats.frame.nb\_errors;

stream->stats.stream.nb\_invalid += stream->stats.frame.nb\_invalid;

if (frame->has\_early\_pts) stream->stats.stream.nb\_pts\_early++;

if (frame->has\_initial\_pts) stream->stats.stream.nb\_pts\_initial++;

if (frame->last\_pts\_diff <= frame->first\_data) stream->stats.stream.nb\_pts\_constant++;

if (frame->nb\_scr >= frame->nb\_packets - frame->nb\_empty) stream->stats.stream.nb\_scr\_count\_ok++;

if (frame->nb\_scr\_diffs + 1 == frame->nb\_scr) stream->stats.stream.nb\_scr\_diffs\_ok++;

memset(&stream->stats.frame, 0, sizeof(stream->stats.frame));

}

### uvc\_video\_decode\_data()

static void uvc\_video\_decode\_data(struct uvc\_streaming \*stream,struct uvc\_buffer \*buf, const \_\_u8 \*data, int len)

{

int maxlen = buf->length - buf->bytesused;

void \***mem = buf->mem + buf->bytesused**;

nbytes = min((unsigned int)len, maxlen);

**memcpy(mem, data, nbytes);**

buf->bytesused += nbytes;

if (len > maxlen) {

buf->state = UVC\_BUF\_STATE\_READY;

buf->error = UVC\_BUF\_ERR\_OVERFLOW;

}

}

### uvc\_video\_decode\_end()

static void uvc\_video\_decode\_end(struct uvc\_streaming \*stream,struct uvc\_buffer \*buf, const \_\_u8 \*data, int len)

{

if (data[1] & UVC\_STREAM\_**EOF**&&**buf->bytesused** != 0) {

**buf->state = UVC\_BUF\_STATE\_READY**;

if (stream->dev->quirks & UVC\_QUIRK\_STREAM\_NO\_FID)

stream->last\_fid ^= UVC\_STREAM\_FID;

}

}

### uvc\_queue\_next\_buffer()

struct uvc\_buffer \*uvc\_queue\_next\_buffer(struct uvc\_video\_queue \*queue, struct uvc\_buffer \*buf)

{

struct uvc\_buffer \*nextbuf;

if ((queue->flags & UVC\_QUEUE\_DROP\_CORRUPTED) && buf->error) {

queue->framesdropped++;

buf->error = 0;

buf->state = UVC\_BUF\_STATE\_QUEUED;

buf->bytesused = 0;

vb2\_set\_plane\_payload(&buf->buf, 0, 0);

**return buf;**

}

**list\_del(&buf->queue); //qbuf: list\_add\_tail(&buf->queue, &queue->irqqueue);**

if (!list\_empty(&queue->irqqueue)) **nextbuf** = list\_first\_entry(&queue->irqqueue, struct uvc\_buffer,queue);

else nextbuf = NULL;

buf->state = buf->error ? VB2\_BUF\_STATE\_ERROR : **UVC\_BUF\_STATE\_DONE**;

**vb2\_set\_plane\_payload**(&buf->buf, 0, buf->bytesused);

**vb2\_buffer\_done**(&buf->buf, **VB2\_BUF\_STATE\_DONE**);

**return nextbuf;**

}

### vb2\_buffer\_done()

static inline void vb2\_set\_plane\_payload(struct vb2\_buffer \*vb,unsigned int plane\_no, unsigned long size)

{

if (plane\_no < vb->num\_planes) vb->v4l2\_planes[plane\_no].bytesused = size;

}

void vb2\_buffer\_done(**struct vb2\_buffer \*vb**, enum vb2\_buffer\_state state)

{

**struct vb2\_queue \*q** = vb->vb2\_queue;

if (vb->state != VB2\_BUF\_STATE\_ACTIVE) return;

if (state != VB2\_BUF\_STATE\_DONE && state != VB2\_BUF\_STATE\_ERROR) return;

for (plane = 0; plane < vb->num\_planes; ++plane)

call\_memop(q, finish, vb->planes[plane].mem\_priv); //finish()没有定义

**vb->state = state;//**VB2\_BUF\_STATE\_DONE

**list\_add\_tail(&vb->done\_entry, &q->done\_list);**

atomic\_dec(&q->queued\_count);

**wake\_up(&q->done\_wq);**

}

# End