OTG IP设备库介绍（上）

STM32_f105-07_f2_f4_usb-host-device_lib

STSW-STM32046 (UM1021)

FS HID Device

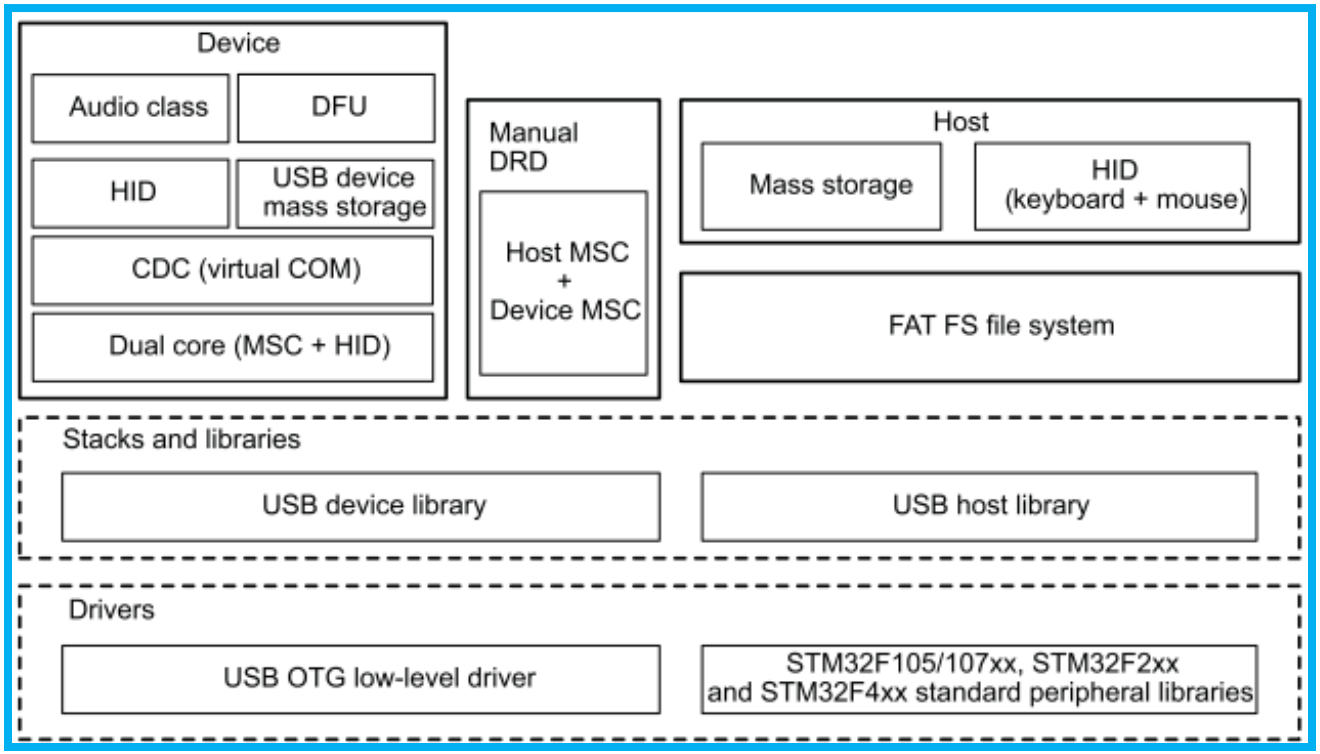
life.augmented

OTG IP设备库讲解要点

- 项目结构
  - 库函数、USB库函数
  - 类相关文件
  - 应用相关文件
- 库代码结构
  - 初始化
  - USB中断处理流程
  - 回调接口函数
- 枚举通信的代码流程
- 如何基于已有例程做自己的应用裁剪

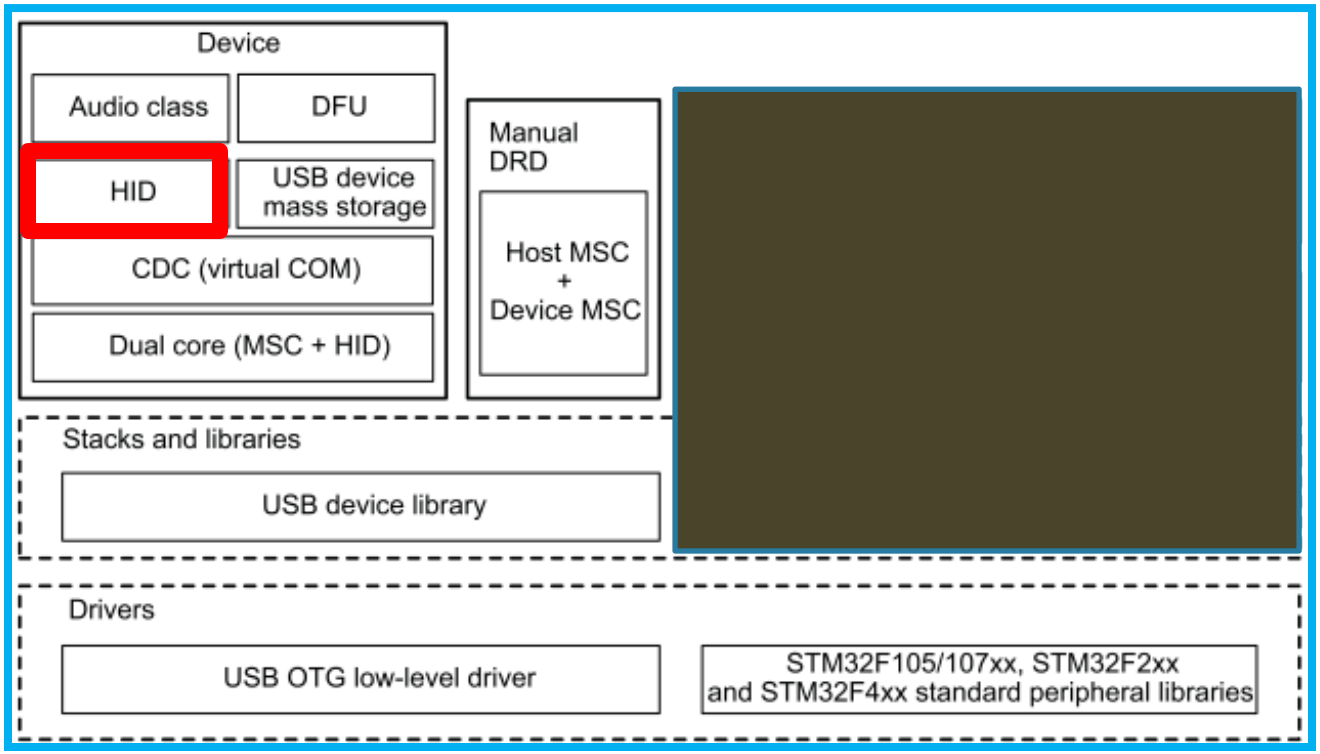
STSW-STM32046库概览

- 主要特性
  - 易于扩展以支持OTG功能
  - 结构通用而简单易用
    - 可添加用户特定类
    - 支持复合设备应用
  - 支持多个OTG IP同时工作 → Dual core demo



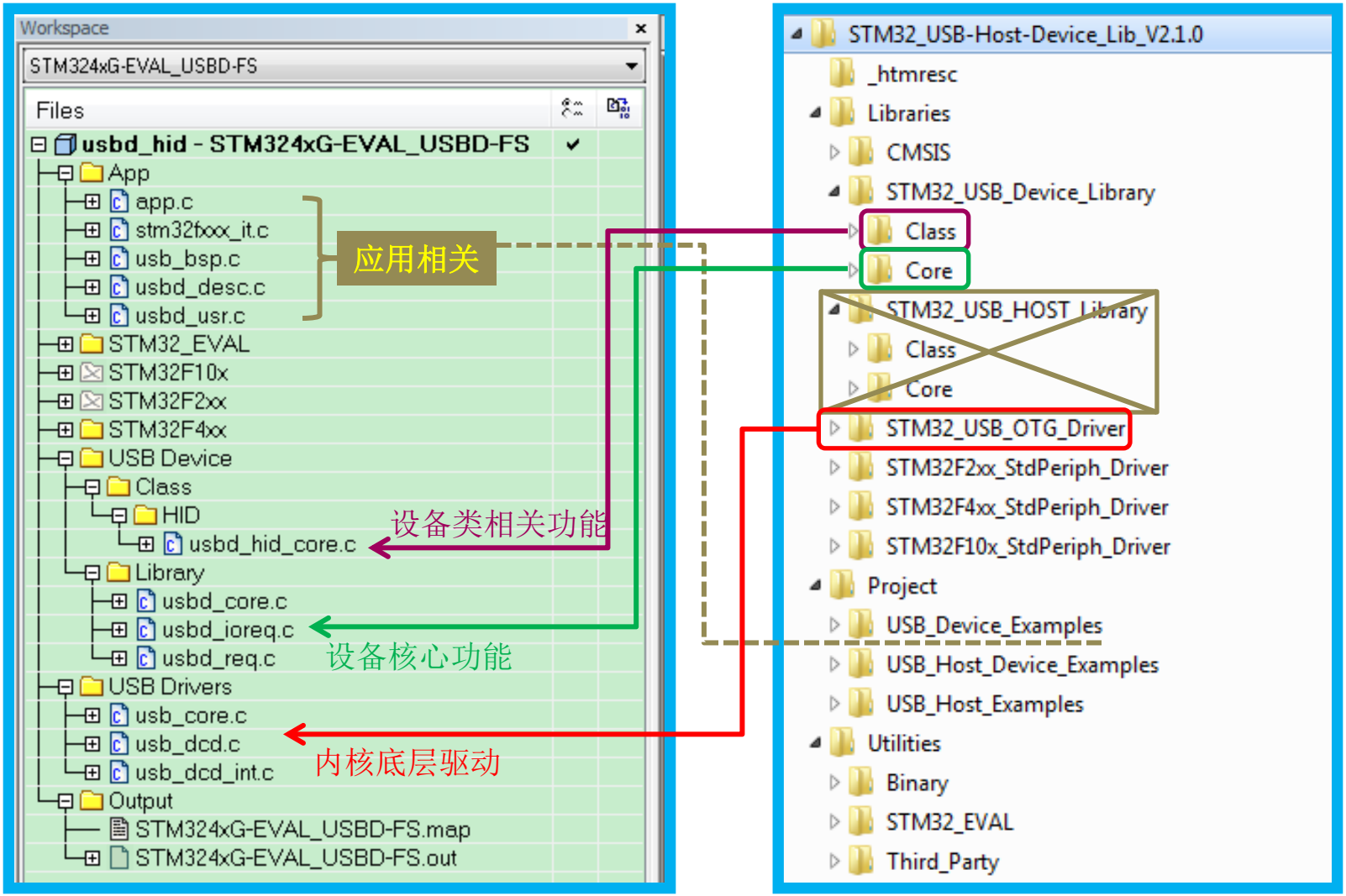
STSW-STM32046库概览

- 主要特性
  - 易于扩展以支持OTG功能
  - 结构通用而简单易用
    - 可添加用户特定类
    - 支持复合设备应用
  - 支持多个OTG IP同时工作 → Dual core demo



项目结构

- 运行在STM324xG-EVAL板上的HID **Device**例程



usb_core.c
内核层，寄存器封装

- USB_OTG_CoreInit
- USB_OTG_CoreInitHost
- USB_OTG_CoreReset
- USB_OTG_DisableGlobalInt
- USB_OTG_DriveVbus
- USB_OTG_EnableCommonInt
- USB_OTG_EnableGlobalInt
- USB_OTG_EnableHostInt
- USB_OTG_FlushRxFifo
- USB_OTG_FlushTxFifo
- USB_OTG_GetMode
- USB_OTG_HC_DoPing
- USB_OTG_HC_Halt
- USB_OTG_HC_Init
- USB_OTG_HC_StartXfer
- USB_OTG_InitFSLSPClkSel
- USB_OTG_IsDeviceMode
- USB_OTG_IsEvenFrame
- USB_OTG_IsHostMode
- USB_OTG_ReadCoreItr
- USB_OTG_ReadHostAllChannels_intr
- USB_OTG_ReadHPRT0
- USB_OTG_ReadOtgItr
- USB_OTG_ReadPacket
- USB_OTG_ResetPort
- USB_OTG_SelectCore
- USB_OTG_SetCurrentMode
- USB_OTG_StopHost
- USB_OTG_WritePacket

- DCD_DevConnect
- DCD_DevDisconnect
- DCD_EP_Close
- DCD_EP_ClrStall
- DCD_EP_Flush
- DCD_EP_Open
- DCD_EP_PrepareRx
- DCD_EP_SetAddress
- DCD_EP_Stall
- DCD_EP_Tx
- DCD_GetEPStatus
- DCD_Init
- DCD_SetEPStatus

usb_dcd.c
设备接口层

内核底层驱动

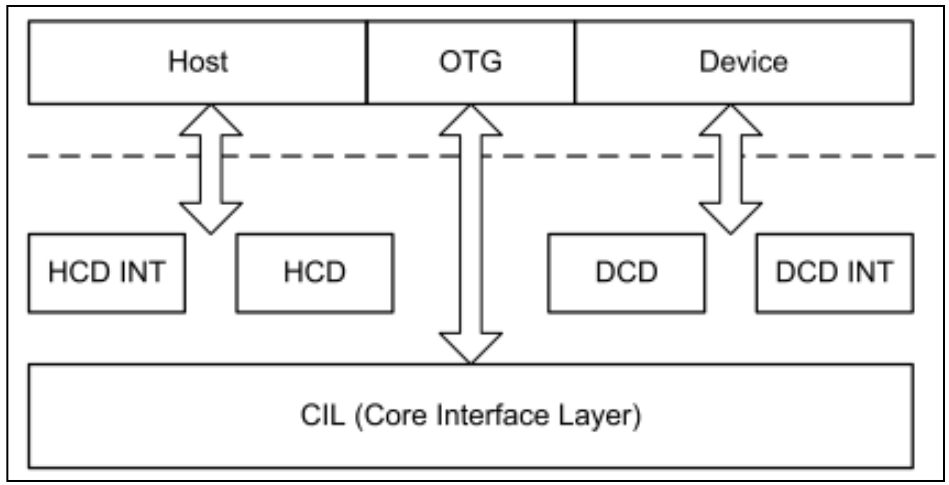
usb_dcd_int.c
设备驱动中断子程序

Call them

- DCD_HandleEnumDone_ISR
- DCD_HandleInEP_ISR
- DCD_HandleOutEP_ISR
- DCD_HandleResume_ISR
- DCD_HandleRxStatusQueueLevel_ISR
- DCD_HandleSof_ISR
- DCD_HandleUsbReset_ISR
- DCD_HandleUSBSuspend_ISR
- DCD_IsoINIncomplete_ISR
- DCD_IsoOUTIncomplete_ISR
- DCD_OTG_ISR
- DCD_ReadDevInEP
- DCD_SessionRequest_ISR
- DCD_WriteEmptyTxFifo
- USBD_OTG_ISR_Handler

USB OTG底层驱动文件

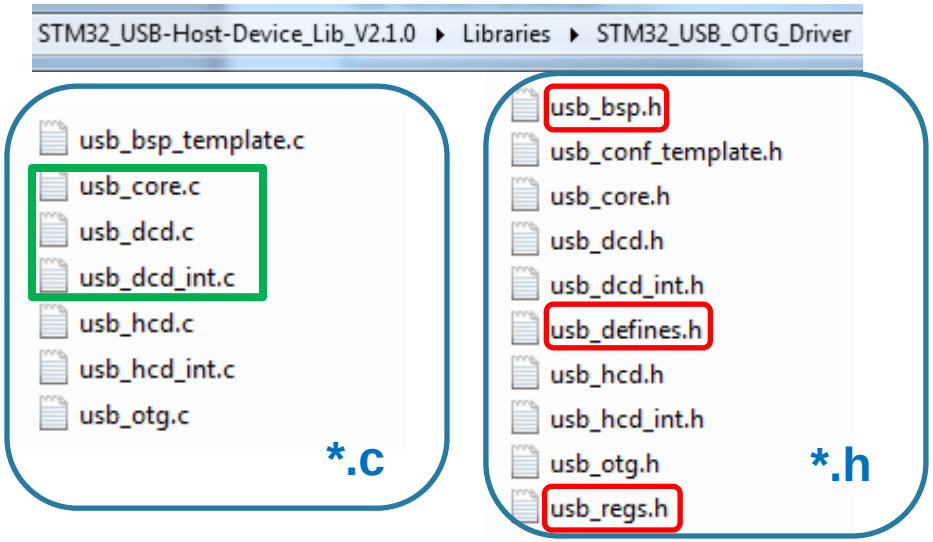
- 驱动文件架构概览



上层：Stack、上层软件

底层驱动：把OTG硬件模块和上层协议栈联系起来

- 底层驱动文件



USB OTG底层驱动文件.描述

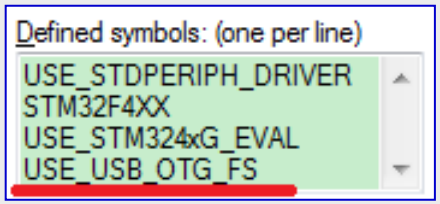
角色	文件	描述
Common	usb_core.c/.h	硬件抽象层，对OTG寄存器的封装
	usb_**conf**_template.h	配置文件：发送、接收FIFO大小；模块角色；所选的特性。 **用户要把该文件拷贝到应用文件夹下，并根据应用需要修改配置**
	usb_bsp_template.c/.h	模块底层配置：中断、GPIO… **用户要把该文件拷贝到应用文件夹下，并根据应用需要修改配置**
Host	usb_hcd.c/.h	主机接口层：stack使用它来访问内核
	usb_hcd_int.c/.h	主机角色下的中断处理
Device	usb_dcd.c/.h	设备接口层：stack使用它来访问内核
	usb_dcd_int.c/.h	设备角色下的中断处理
OTG	usb_otg.c/.h	对SPR、HNP协议的实现，以及和OTG角色相关的中断

USB OTG底层驱动文件.配置

usb_conf.h

1. USB OTG FS PHY配置

```
#ifndef USE_USB_OTG_FS
 //#define USE_USB_OTG_FS
#endif /* USE_USB_OTG_FS */
```




```
#ifdef USE_USB_OTG_FS
 #define USB_OTG_FS_CORE
#endif
```

2. USB OTG HS PHY配置

```
#ifndef USE_USB_OTG_HS
 //#define USE_USB_OTG_HS
#endif /* USE_USB_OTG_HS */

#ifndef USE_ULPI_PHY
 //#define USE_ULPI_PHY
#endif /* USE_ULPI_PHY */

#ifndef USE_EMBEDDED_PHY
 //#define USE_EMBEDDED_PHY
#endif /* USE_EMBEDDED_PHY */

#ifdef USE_USB_OTG_HS
 #define USB_OTG_HS_CORE
#endif
```

3. USB OTG FS FIFO配置

```
#ifdef USB_OTG_FS_CORE
 #define RX_FIFO_FS_SIZE                128
 #define TX0_FIFO_FS_SIZE               64
 #define TX1_FIFO_FS_SIZE               128

 #define USB_OTG_FS_LOW_PWR_MGMT_SUPPORT
// #define USB_OTG_FS_SOF_OUTPUT_ENABLED
#endif
```

4. USB OTG HS FIFO配置

```
#ifdef USB_OTG_HS_CORE
 #define RX_FIFO_HS_SIZE                512
 #define TX0_FIFO_HS_SIZE               128
 #define TX1_FIFO_HS_SIZE               372

#define USB_OTG_HS_LOW_PWR_MGMT_S
// #define USB_OTG_HS_SOF_OUTPUT_ENA

// #define USB_OTG_INTERNAL_VBUS_ENAB
#define USB_OTG_EXTERNAL_VBUS_ENABL
#define USB_OTG_HS_INTERNAL_DMA_ENA
#define USB_OTG_HS_DEDICATED_EP1_EN

#ifdef USE_ULPI_PHY
 #define USB_OTG_ULPI_PHY_ENABLED
#endif
 #ifdef USE_EMBEDDED_PHY
   #define USB_OTG_EMBEDDED_PHY_ENAB
 #endif
#endif
```

5. USB OTG 角色配置

```
//#define USE_HOST_MODE
#define USE_DEVICE_MODE
//#define USE_OTG_MODE
```

USB OTG底层驱动文件.编程模型

- OTG模块句柄
  - 全局结构体
  - 所有模块用到的变量、状态、缓冲区，用于处理内部状态机和传输流
  - 必须由用户在应用层分配句柄实例

```
#ifdef USB_OTG_HS_INTERNAL_DMA_ENABLED
  #if defined ( __ICCARM__ ) /*!< IAR Compiler */
    #pragma data_alignment=4
  #endif
#endif /* USB_OTG_HS_INTERNAL_DMA_ENABLED */
__ALIGN_BEGIN USB_OTG_CORE_HANDLE          USB_OTG_dev __ALIGN_END ;
```

```
typedef struct USB_OTG_handle
{
  USB_OTG_CORE_CFGS    cfg;
  USB_OTG_CORE_REGS    regs;
#ifdef USE_DEVICE_MODE
  DCD_DEV     dev; // 设备驱动的核心结构体
#endif
#ifdef USE_HOST_MODE
  HCD_DEV     host;
#endif
#ifdef USE_OTG_MODE
  OTG_DEV     otg;
#endif
}
USB_OTG_CORE_HANDLE , *PUSB_OTG_CORE_HANDLE;
```

USB OTG底层驱动文件.编程模型.设备

Usb_dcd.c/.h

- 设备驱动初始化
  - **DCD_Init** (**USB_OTG_CORE_HANDLE *pdev** , USB_OTG_CORE_ID_TypeDef coreID)
- 端点配置
  - **DCD_EP_Open** (**USB_OTG_CORE_HANDLE *pdev** , ep_addr, ep_mps, ep_type)
  - **DCD_EP_Close** (**USB_OTG_CORE_HANDLE *pdev** , ep_addr, ep_mps, ep_type)
- 端点USB数据流
  - **DCD_EP_PreprareRx** (**USB_OTG_CORE_HANDLE *pdev** , ep_addr, *pbuf, buf_len)
  - **DCD_EP_Tx** (**USB_OTG_CORE_HANDLE *pdev**, ep_addr, *pbuf, buf_len)
  - **DCD_EP_Stall** (**USB_OTG_CORE_HANDLE *pdev**, ep_addr)
  - **DCD_EP_ClrStall** (**USB_OTG_CORE_HANDLE *pdev**, ep_addr)
  - **DCD_EP_Flush** (**USB_OTG_CORE_HANDLE *pdev**, ep_addr)
  - **DCD_GetEPStatus** (**USB_OTG_CORE_HANDLE *pdev,** ep_addr)
- 软件控制USB设备的连接和断开
  - **DCD_DevConnect** (**USB_OTG_CORE_HANDLE *pdev**)
  - **DCD_DevDisconnect** (**USB_OTG_CORE_HANDLE *pdev**)
- USB设备中断
  - **USBD_OTG_ISR_Handler** (**USB_OTG_CORE_HANDLE *pdev**)

底层驱动文件 ← 设备核心功能 & 类功能

usb_dcd.c 底层驱动的设备接口层	设备核心功能 / 类功能	具体文件	功能类别
DCD_Init()	USBD_Init	**usbd_core.c**	设备核心功能
DCD_EP_Close()	USBD_Reset (clear feature)	**usbd_core.c**	
DCD_EP_Open()	USBD_**HID**_DeInit (0x81/0x01)	**usbd_hid_core.c**	类功能
	USBD_**HID**_Init (0x81/0x01)	**usbd_hid_core.c**	
DCD_EP_Flush()	USBD_**HID**_DataIn	**usbd_hid_core.c**	
DCD_EP_Tx()	USBD_**HID**_SendReport (0x81)	**usbd_hid_core.c**	
	USBD_CtlSendData (0)	**usbd_ioreq.c**	设备核心功能
	USBD_CtlContinueSendData (0)		
	USBD_CtlSendStatus (0)		
DCD_EP_Stall()	USBD_StdEPReq	**usbd_req.c**	
	USBD_CtlError	**usbd_req.c**	
DCD_EP_ClrStall()	USBD_StdEPReq	**usbd_req.c**	
DCD_EP_PreprareRx()	not used	N.A	

USB OTG底层驱动文件.编程模型.设备

- OTG模块句柄**USB_OTG_CORE_HANDLE**
  - → 设备驱动的核心结构体**DCD_DEV**

typedef struct _DCD {}		
uint8_t	device_config	设备当前工作于哪个configuration下
uint8_t	device_state	设备在一次控制传输中的阶段标志
uint8_t	device_status	设备在枚举过程中的阶段标志
uint8_t	device_old_status	
uint8_t	device_address	主机分配的设备地址
uint8_t	connection_status	设备和主机的连接状态
uint8_t	test_mode	
uint32_t	DevRemoteWakeup	该设备当前是否被通过SetFeature使能了远程唤醒功能
USB_OTG_EP	**in_ep** [15]	num, type,is_in, is_stall, data_pid_start, even_odd_frame, tx_fifo_num, MPZ // *xfer_buff, dma_addr, xfer_len, xfer_count // total_data_len, ctl_data_len, rem_data_len
USB_OTG_EP	**out_ep** [15]	
uint8_t	setup_packet [8*3]	存储来自主机的8字节request
USBD_Class_cb_TypeDef	* class_cb	类回调函数， 在<usbd_XXX_core.c>中定义
USBD_Usr_cb_TypeDef	* usr_cb	用户回调函数，连接枚举的各个阶段标志
USBD_DEVICE	* usr_device	设备描述符集合
uint8_t	* pConfig_descriptor	

usb_core.c：开始端点上的传输

- 无论是0端点还是非0端点，无论是IN方向还是OUT方向：在函数(**USB_OTG_EPStartXfer** / **USB_OTG_EP0StartXfer**)中都是要配置
  - 对应端点传输寄存器中的 xfersize & pktcnt 域
  - 对应端点控制寄存器中的 cnak =1 & epena =1 域
  - 如果IN方向，且len>0，还要使能发送FIFO空中断
- 假设MPZ=64

		0端点				非0端点			
IN方向		Len=0	Len=63	Len=64	Len=65	Len=0	Len=63	Len=64	Len=65
	修改len	--	--	--	64	--	--	--	--
	xfersize	0	63	64	MPZ64	0	Len=63	Len=64	Len=65
	pckcnt	1	1	1	1	1	1	1	2
OUT方向		Len=0	Len=63	Len=64	Len=65	Len=0	Len=63	Len=64	Len=65
	修改len	--	--	--	64	--	--	--	--
	Xfersize	64	64	64	64	64	1*64	1*64	2*64
	pckcnt	1	1	1	1	1	1	1	2

usb_core.c：开始端点上的传输

- 无论是0端点还是非0端点，无论是IN方向还是OUT方向：在函数(**USB_OTG_EPStartXfer** / **USB_OTG_EP0StartXfer**)中都是要配置
  - 对应端点传输寄存器中的 xfersize & pktcnt 域
  - 对应端点控制寄存器中的 cnak =1 & epena =1 域
  - 如果IN方向，且len>0，还要使能发送FIFO空中断
- 假设MPZ=64

		0端点				说明
IN方向		Len=0	Len=63	Len=64	Len=65	端点0上的传输，无论IN还是OUT方向， >> pckcnt始终是1 >> xfersize取值则是 [0, 64]
	修改len	--	--	--	64	
	xfersize	0	63	64	MPZ64	
	pckcnt	1	1	1	1	
OUT方向		Len=0	Len=63	Len=64	Len=65	
	修改len	--	--	--	64	
	Xfersize	64	64	64	64	
	pckcnt	1	1	1	1	

usb_core.c：开始端点上的传输

- 无论是0端点还是非0端点，无论是IN方向还是OUT方向：在函数(**USB_OTG_EPStartXfer** / **USB_OTG_EP0StartXfer**)中都是要配置
  - 对应端点传输寄存器中的 xfersize & pktcnt 域
  - 对应端点控制寄存器中的 cnak =1 & epena =1 域
  - 如果IN方向，且len>0，还要使能发送FIFO空中断
- 假设MPZ=64

		说明	非0端点			
IN方向		非0端点上的传输，无论IN还是OUT方向， >> pckcnt 不会被固定在1，取决于实际要传输的数据长度需要要几个包	Len=0	Len=63	Len=64	Len=65
	修改len		--	--	--	--
	xfersize		0	Len=63	Len=64	Len=65
	pckcnt		1	1	1	2
OUT方向			Len=0	Len=63	Len=64	Len=65
	修改len		--	--	--	--
	Xfersize		64	1*64	1*64	2*64
	pckcnt		1	1	1	2

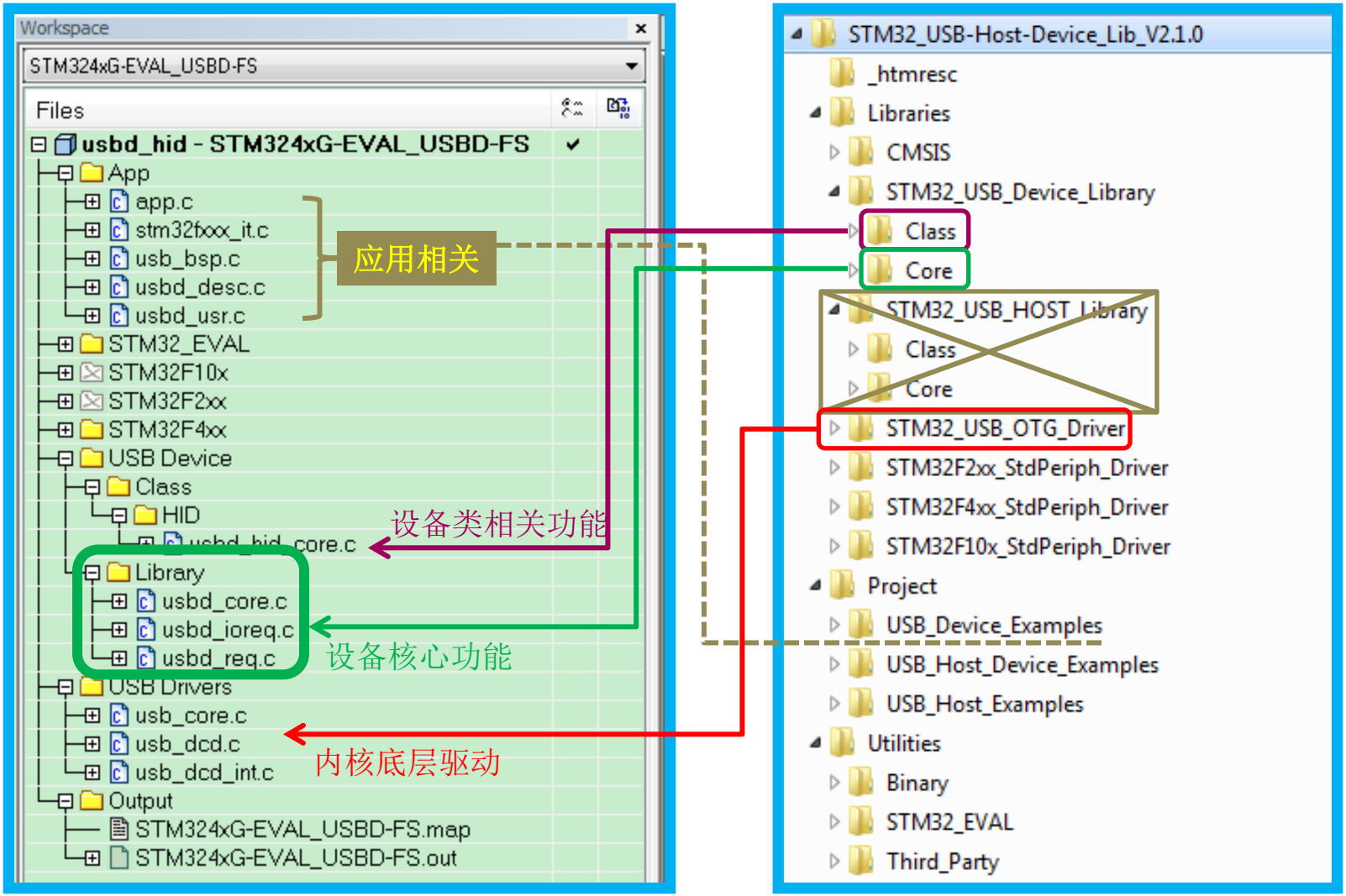
usb_core.c：开始端点上的传输

- 无论是0端点还是非0端点，无论是IN方向还是OUT方向：在函数(**USB_OTG_EPStartXfer** / **USB_OTG_EP0StartXfer**)中都是要配置
  - 对应端点传输寄存器中的 xfersize & pktcnt 域
  - 对应端点控制寄存器中的 cnak =1 & epena =1 域
  - 如果IN方向，且len>0，还要使能发送FIFO空中断
- 假设MPZ=64

		0端点				非0端点			
IN方向		说明：OUT方向，无论是0端点还是非0端点，xfersize不是len的实际长度，而一定是MPZ的整数倍 = N * MPZ , N>0							
	修改len								
	xfersize								
	pckcnt								
OUT方向		Len=0	Len=63	Len=64	Len=65	Len=0	Len=63	Len=64	Len=65
	修改len	--	--	--	64	--	--	--	--
	Xfersize	64	64	64	64	64	1*64	1*64	2*64
	pckcnt	1	1	1	1	1	1	1	2

项目结构

- 运行在STM324xG-EVAL板上的HID **Device**例程



设备核心功能

usbd_core.c

~~USBD_ClrCfg~~
USBD_DataInStage
USBD_DataOutStage
USBD_DeInit
USBD_DevConnected
USBD_DevDisconnected
USBD_Init
USBD_IsoINIncomplete
USBD_IsoOUTIncomplete
USBD_Reset
USBD_Resume
USBD_RunTestMode
~~USBD_SetCfg~~
USBD_SetupStage
USBD_SOF
USBD_Suspend

用户接口API

USBD_DCD_INT_cb结构体成员
@ <usbdc_core.c>
在中断里被调用

usbd_req.c
枚举过程中对标准Request的处理

USBD_ClrFeature
USBD_CtlError
USBD_GetConfig
USBD_GetDescriptor
USBD_GetLen
USBD_GetStatus
USBD_GetString
USBD_ParseSetupRequest
USBD_SetAddress
USBD_SetConfig
USBD_SetFeature
USBD_StdDevReq
USBD_StdEPReq
USBD_StdItfReq

USBD_CtlContinueRx
USBD_CtlContinueSendData
USBD_CtlPrepareRx
USBD_CtlReceiveStatus
USBD_CtlSendData
USBD_CtlSendStatus
USBD_GetRxCount

usbd_ioreq.c
处理控制传输的各个阶段

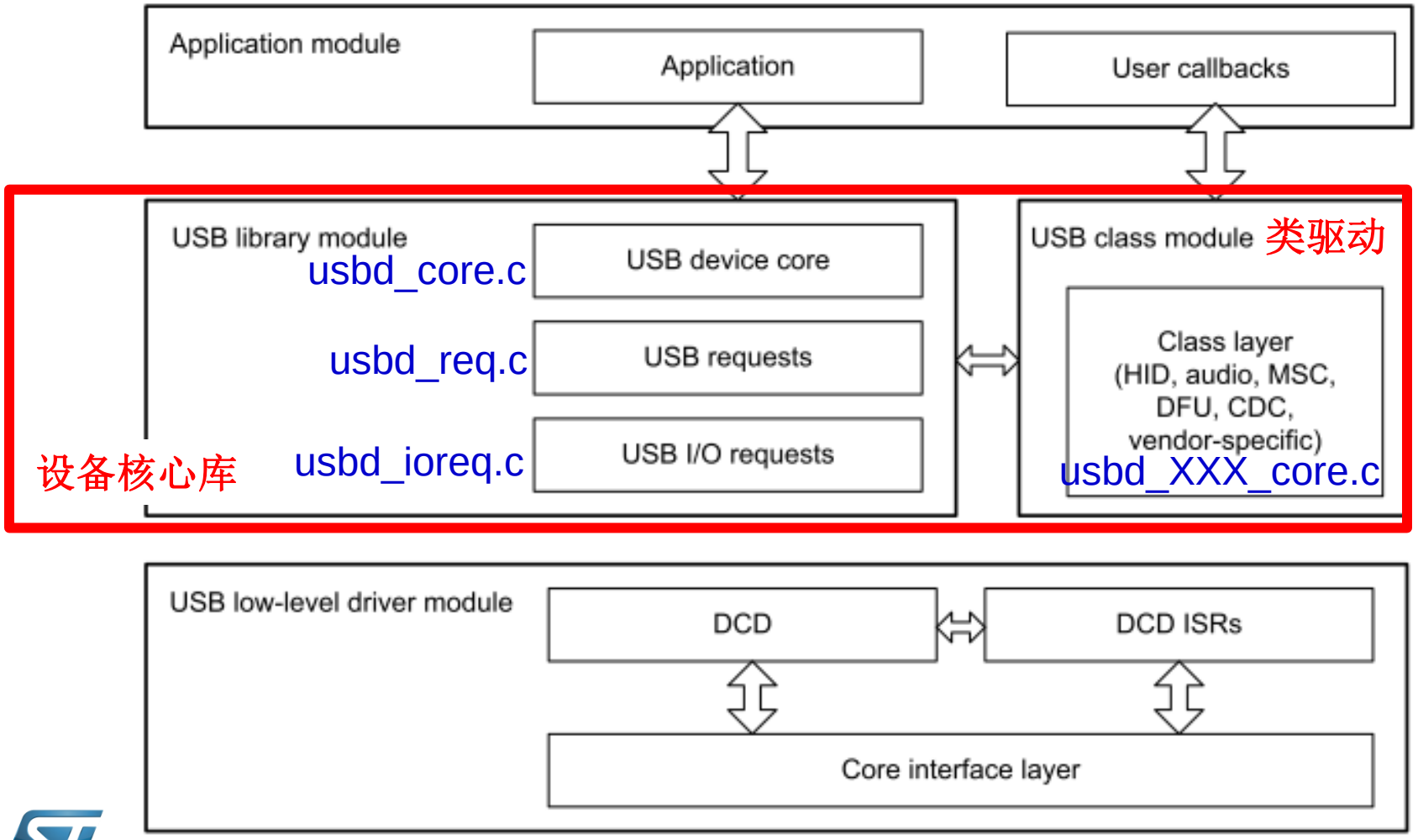
life.augmented

USB OTG设备库特性概览

- 支持多包传输：使能大数据量传输，无需根据最大包长事先拆分数据
- 支持控制端点上连续收到3个transfer（兼容OHCI控制器）
- 32位对齐的数据结构，以处理高速模式下的DMA传输
- 支持多个OTG模块实例同时使用

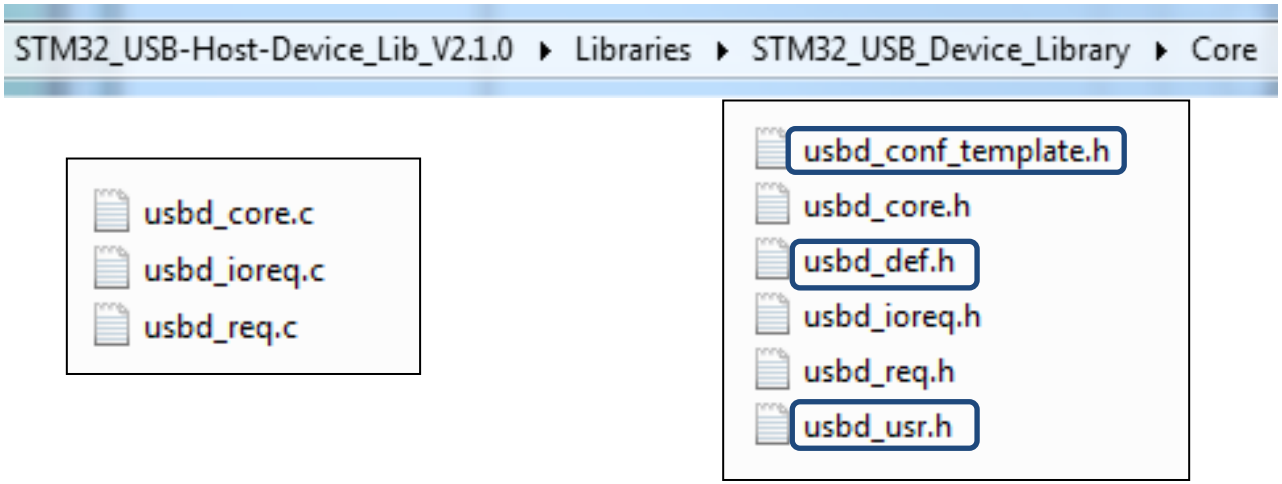
USB OTG设备核心库.架构

- 设备核心功能文件架构概览



USB OTG设备核心库.文件.配置

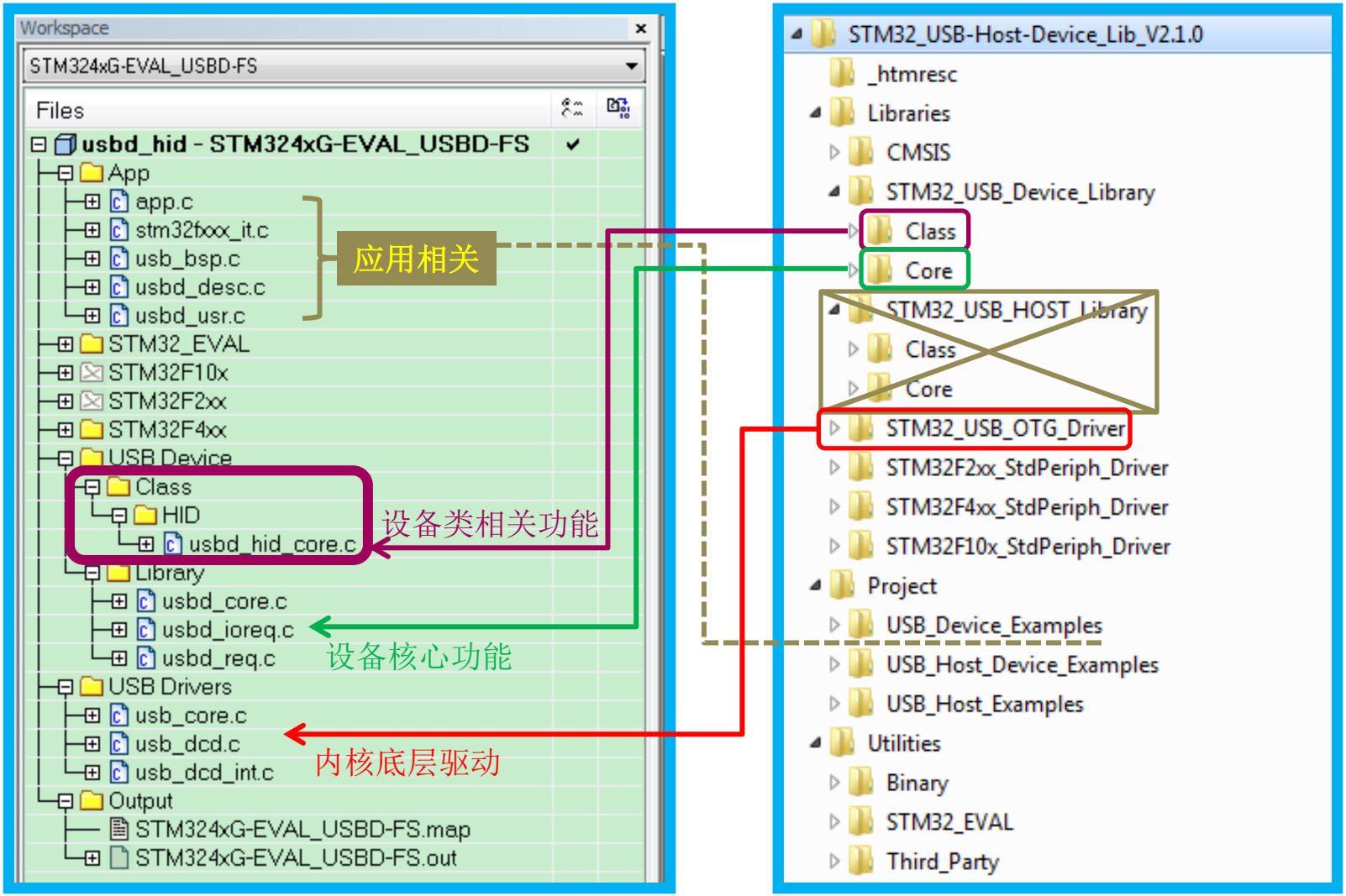
- 底层驱动文件及描述



文件	描述
usbd_core.c/.h	处理USB设备在整个通信过程中对各个中断事件的响应
usbd_req.c/.h	对USB标准请求的实现
usbd_ioreq.c/.h	处理control传输中四个类型的transaction流程
usbd_conf.h	所支持设备的Interface, configuration的个数、非0端点的定义 #define USBD_CFG_MAX_NUM 1 #define USBD_ITF_MAX_NUM 1 #define HID_IN_EP 0x81 #define HID_OUT_EP 0x01

项目结构

- 运行在STM324xG-EVAL板上的HID **Device**例程



usbd_hid_core.c
USB设备HID类响应回调函数
配置描述符
类相关描述符（e.g. Report描述符）

该文件中定义的结构体
USBD_HID_cb的成员函数


```
USBD_Class_cb_TypeDef  USBD_HID_cb =
{
  USBD_HID_Init,
  USBD_HID_DeInit,
  USBD_HID_Setup,
  NULL,                 /*EP0_TxSent*/
  NULL,                 /*EP0_RxReady*/
  USBD_HID_DataIn, /*DataIn*/
  NULL,                 /*DataOut*/
  NULL,                 /*SOF */
  NULL,
  NULL,
  USBD_HID_GetCfgDesc,
#ifdef USB_OTG_HS_CORE
  USBD_HID_GetCfgDesc,
#endif
};
```

设备的类相关（HID）功能

设备的类相关回调

- 在初始化时指定类回调

```
USBD_Init (&USB_OTG_Core_dev,  USB_OTG_FS_CORE_ID,
           &USR_desc,    &USBD_HID_cb,    &USR_cb);
```

@ <usbd_hid_core.c>

```
USBD_Class_cb_TypeDef  USBD_HID_cb =
{
  USBD_HID_Init,
  USBD_HID_DeInit,
  USBD_HID_Setup,
  NULL,
  NULL,
  USBD_HID_DataIn,
  NULL,
  NULL,
  NULL,
  NULL,
  USBD_HID_GetCfgDesc,
#ifdef USB_OTG_HS_CORE
  USBD_HID_GetCfgDesc,
#endif
};
```

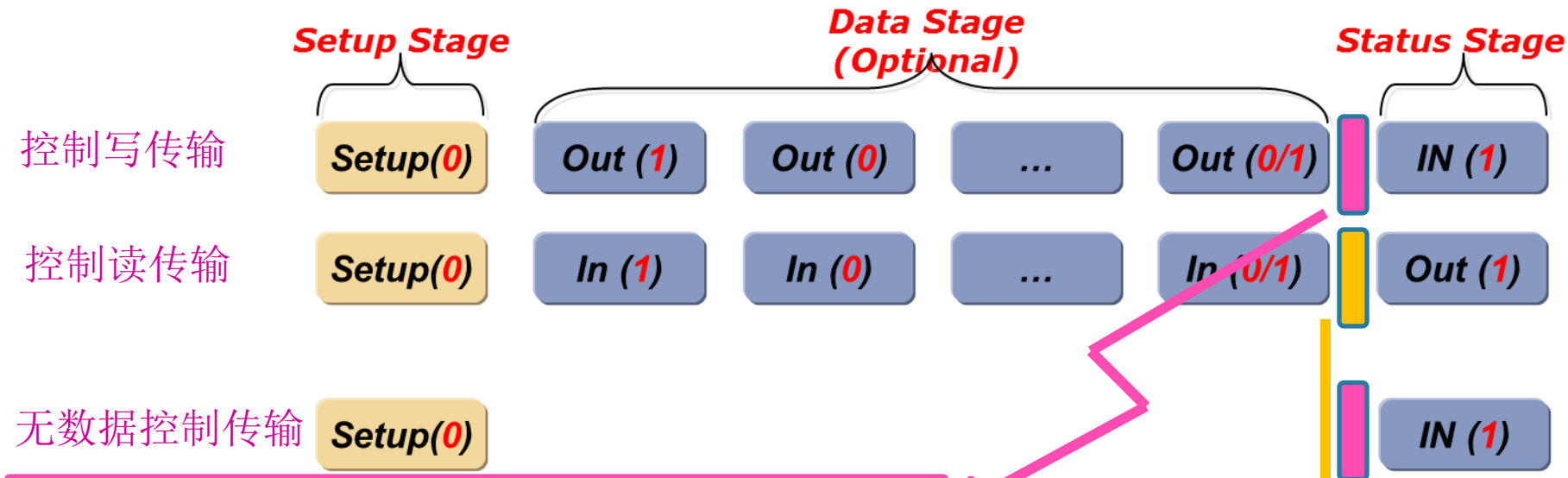
```
typedef struct _USBD_Class_cb
{
  uint8_t  (*Init);
  uint8_t  (*DeInit);
  uint8_t  (*Setup);
  uint8_t  (*EP0_TxSent);
  uint8_t  (*EP0_RxReady);
  uint8_t  (*DataIn);
  uint8_t  (*DataOut);
  uint8_t  (*SOF);
  uint8_t  (*IsoINIcomplete);
  uint8_t  (*IsoOUTIcomplete);
  uint8_t  *(*GetConfigDescriptor);

  uint8_t  *(*GetOtherConfigDescriptor);
} USBH_Class_cb_TypeDef;
```

HID类 Vs. MSC类

函数		HID	VCP	MSC	被…调用	描述
Init	控制端点	Y	Y	Y	收到Set Configuration命令	打开类用到的端点
DeInit		Y	Y	Y	收到Clear Configuration命令	关闭类用到的端点
Setup		Y	Y	Y	收到特殊的类相关request	
EP0_TxSent		NUL	NUL	NUL	控制传输中接收方向Status阶段结束	
EP0_RxReady		NUL	Y	NUL	控制传输中发送方向Status阶段结束	
DataIn	类相关的端点	Y	Y	Y	执行非控制端点上的data in阶段	
DataOut		NUL	Y	Y	执行非控制端点上的data out阶段	
SOF		NUL	Y	NUL	收到SOF中断	可用于同步其他模块
IsoINIcomplete		NUL	NUL	NUL		
IsoOUTIcomplete		NUL	NUL	NUL		
GetConfigDescriptor		Y	Y	Y	返回USB配置描述符	
GetOtherConfigDescriptor		Y	Y	Y	返回HS模式下用到的类的描述符	

控制传输结构



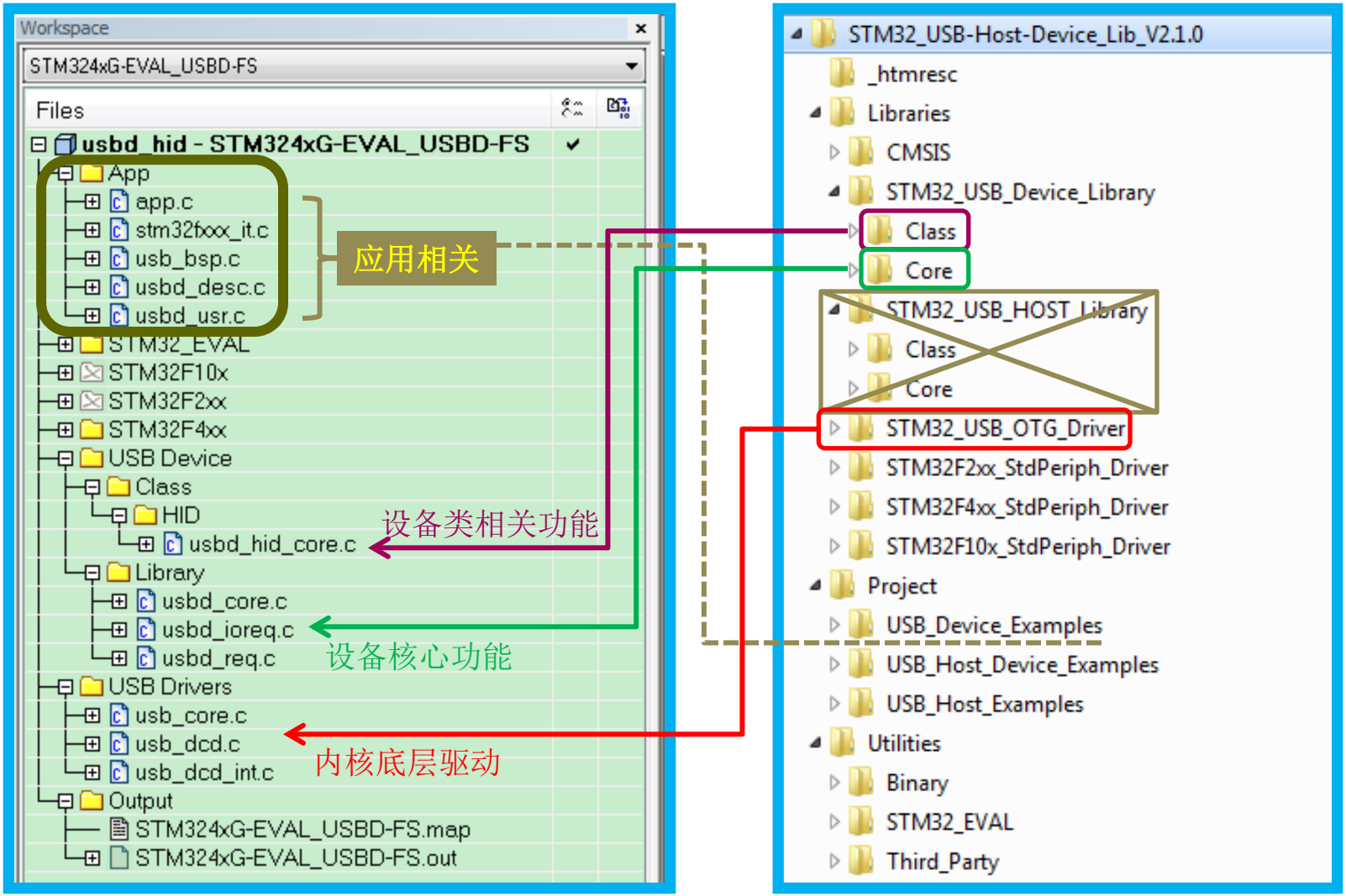

```
{
  if((pdev->dev.class_cb->EP0_RxReady != NULL)&&
     (pdev->dev.device_status == USB_OTG_CONFIGURED))
  {
    pdev->dev.class_cb->EP0_RxReady(pdev);
  }
  USBD_CtlSendStatus(pdev);
}
```

```
if((pdev->dev.class_cb->EP0_TxSent != NULL)&&
   (pdev->dev.device_status == USB_OTG_CONFIGURED))
       {
         pdev->dev.class_cb->EP0_TxSent(pdev);
       }
       USBD_CtlReceiveStatus(pdev);
```

life.augmented

项目结构

- 运行在STM324xG-EVAL板上的HID **Device**例程



app.c
USB设备初始化

```
USBD_Init();

While(1)
{
  ……
}
```

USB_OTG_BSP_EnableInterrupt
USB_OTG_BSP_Init
USB_OTG_BSP_mDelay
USB_OTG_BSP_uDelay

Usb_bsp.c
板级支持文件：
延迟功能、中断使能

Stm32fxxx_it.c
应用相关中断处理入口

```
Void SysTick_Handler(void)
{
 USBD_HID_SendReport (pdev, buf, 4)
}

void EXTI15_10_IRQHandler(void)
{
 // 执行设备远程唤醒
}

void OTG_FS_WKUP_IRQHandler (void)
Void OTG_HS_WKUP_IRQHandler (void)
{
  // 设备被唤醒，恢复时钟配置
}

void OTG_HS_IRQHandler (void)
void OTG_FS_IRQHandler (void)
{ // OTG模块中断总入口
  USBD_OTG_ISR_Handler (pdev)
}

void OTG_HS_EP1_IN_IRQHandler (void);
void OTG_HS_EP1_OUT_IRQHandler (void);
```

USBD_USR_DeviceConfigured
USBD_USR_DeviceConnected
USBD_USR_DeviceDisconnected
USBD_USR_DeviceReset
USBD_USR_DeviceResumed
USBD_USR_DeviceSuspended
USBD_USR_Init

用户回调函数

Usbd_usr.c
用户回调函数

应用相关文件

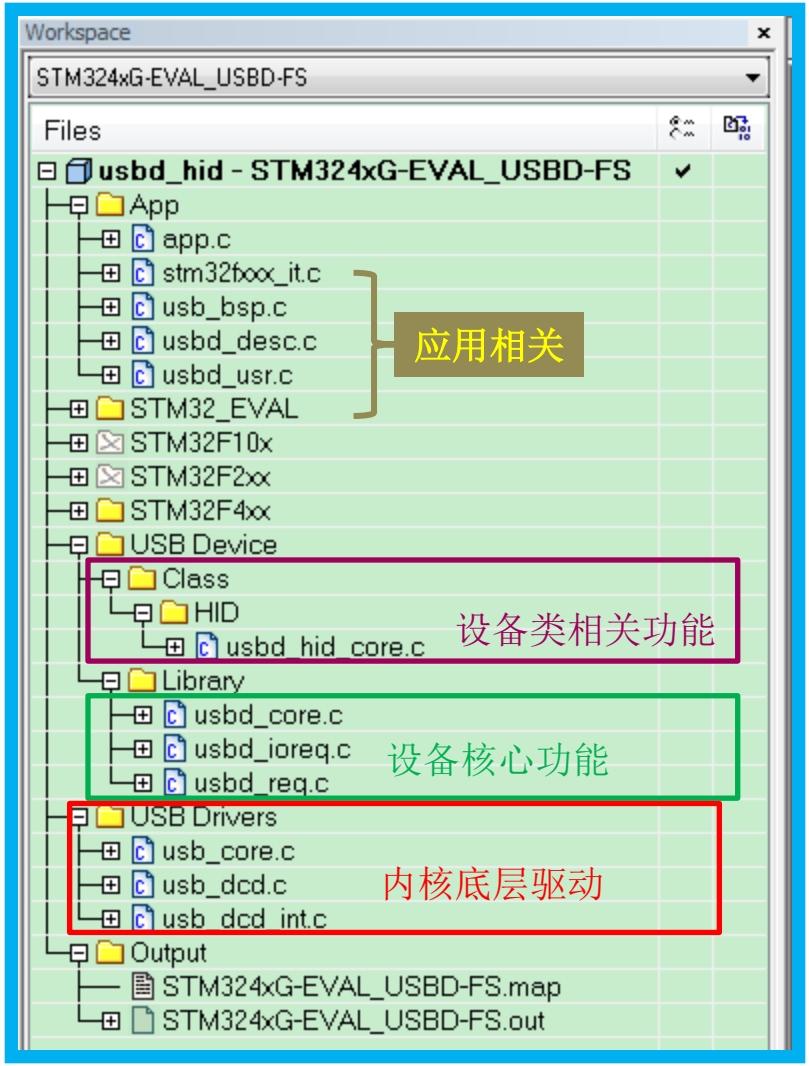
USBD_USR_ConfigStrDescriptor
USBD_USR_DeviceDescriptor
USBD_USR_InterfaceStrDescriptor
USBD_USR_LangIDStrDescriptor
USBD_USR_ManufacturerStrDescriptor
USBD_USR_ProductStrDescriptor
USBD_USR_SerialStrDescriptor

Usbd_desc.c
描述符文件

设备的描述符

代码层	应用相关层	设备类相关功能层
文件	usbd_desc.c	usbd_XXX_core.c
描述符	设备描述符	全部配置描述符
	各个字符串描述符…	类相关描述符

```
USBD_DEVICE USR_desc =
{
  USBD_USR_DeviceDescriptor,
  USBD_USR_LangIDStrDescriptor,
  USBD_USR_ManufacturerStrDescriptor,
  USBD_USR_ProductStrDescriptor,
  USBD_USR_SerialStrDescriptor,
  USBD_USR_ConfigStrDescriptor,
  USBD_USR_InterfaceStrDescriptor,
};
```



life.augmented

用户回调

- 在初始化时指定用户回调函数组

```
USBD_Init (&USB_OTG_Core_dev,  USB_OTG_FS_CORE_ID,
           &USR_desc,    &USBD_HID_cb,    &USR_cb);
```

@ <usbd_usr.c>

```
USBD_Usr_cb_TypeDef  USR_cb =
{
  USBD_USR_Init,
  USBD_USR_DeviceResett,
  USBD_USR_DeviceConfigured,
  USBD_USR_DeviceSuspended,,
  USBD_USR_DeviceResumed,,

  USBD_USR_DeviceConnected,
  USBD_USR_DeviceDisconnected
};
```

```
typedef struct _USBD_USR_PROP
{
  uint8_t  (*Init);
  uint8_t  (*DeviceReset);
  uint8_t  (*DeviceConfigured);
  uint8_t  (*DeviceSuspended);
  uint8_t  (*DeviceResumed );

  uint8_t  (* DeviceConnected);
  uint8_t  (* DeviceDisconnected);
} USBD_Usr_cb_TypeDef;
```

小结：回调函数

- 用户回调函数 @ <usbd_usr.c>
- 类回调函数 @ <usbh_XXX_core.c>

Usbd_usr.c

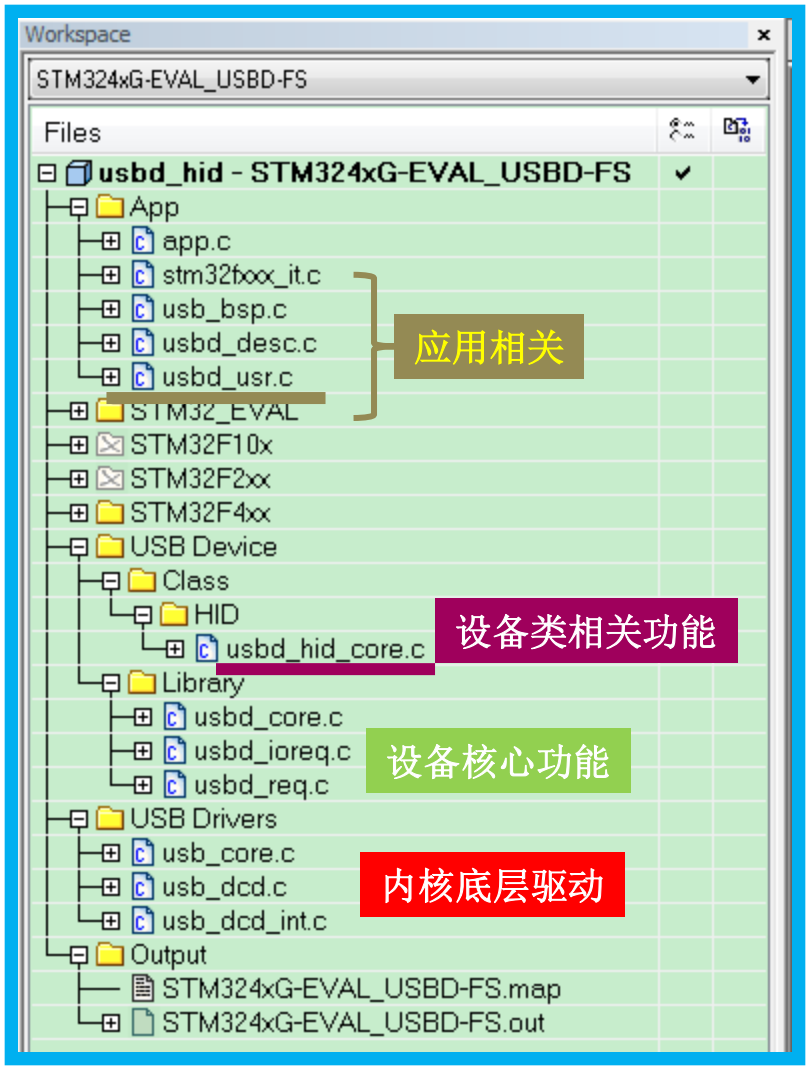
USBD_USR_DeviceConfigured
USBD_USR_DeviceConnected
USBD_USR_DeviceDisconnected
USBD_USR_DeviceReset
USBD_USR_DeviceResumed
USBD_USR_DeviceSuspended
USBD_USR_Init

用户回调函数
该文件中定义的结构体
USR_cb的成员函数

usbd_hid_core.c

USBD_HID_DataIn
USBD_HID_DeInit
USBD_HID_GetCfgDesc
USBD_HID_Init
USBD_HID_SendReport
USBD_HID_Setup

类回调函数
该文件中定义的结构体
USBD_HID_cb的成员函数



life.augmented

OTG IP设备库讲解要点

- 项目结构
  - 库函数、USB库函数
  - 类相关文件
  - 应用相关文件
- **库代码结构**
  - **初始化**
  - **USB中断处理流程**
  - **回调接口函数**
- 枚举通信的代码流程
- 如何基于已有例程做自己的应用裁剪

Device Stack初始化

- 一个用户API
  - void **USBD_Init** (P1, P2, P3, P4, P5)
    - P1 = pdev，指向USB OTG模块句柄的指针
    - P2 = CoreID，实名使用哪个OTG IP
    - P3 = pDevice，指向USB设备描述符集合
    - P4 = class_cb，**类回调函数**， @ <usbh_XXX(CLASS)_core.c>
    - P5 = usr_cb，**用户普通回调函数**，和类无关的回调，主要在枚举过程的不同阶段被调用，通常用于显示枚举阶段到哪一步了
- 用户回调函数 @ <usbd_usr.c>
  - USBD_Init()的P5

OTG IP设备库讲解要点

- 项目结构
  - 库函数、USB库函数
  - 类相关文件
  - 应用相关文件
- 库代码结构
  - 初始化
  - USB中断处理流程
  - 回调接口函数
- **枚举通信的代码流程**
- 如何基于已有例程做自己的应用裁剪

运行USB Device Stack

- 拷贝usb_conf_template.h和usb_bsp_template.c到应用文件夹下并根据应用要求修改，重命名为usb_conf.h和usb_bsp.c
- 定义用户**回调函数** @ <usbd_usr.c> & <usbd_desc.c>
- 定义一个结构体 @ <app.c>
- 调用一个API @ <app.c>

```
USB_OTG_CORE_HANDLE          USB_OTG_dev;

int main(void)
{
   USBD_Init (&USB_OTG_dev,  USE_USB_OTG_FS, &USR_desc, &USBD_MSC_cb,  &USR_cb);

   while (1)
    {
      …. other task…
    }
}
```

CONFIDENTIALITY OBLIGATIONS:

THIS DOCUMENT CONTAINS SENSITIVE INFORMATION.

IT IS CLASSIFIED “MICROCONTROLLERS, MEMORIES & SECURE MCUs (MMS) RESTRICTED AND ITS DISTRIBUTION IS SUBMITTED TO ST/MMS AUTHORIZATION

AT ALL TIMES YOU SHOULD COMPLY WITH THE FOLLOWING SECURITY RULES:

- DO NOT COPY OR REPRODUCE ALL OR PART OF THIS DOCUMENT
- KEEP THIS DOCUMENT LOCKED AWAY
- FURTHER COPIES CAN BE PROVIDED ON A “NEED TO KNOW BASIS”, PLEASE CONTACT YOUR LOCAL ST SALES OFFICE OR DOCUMENT WRITTER.