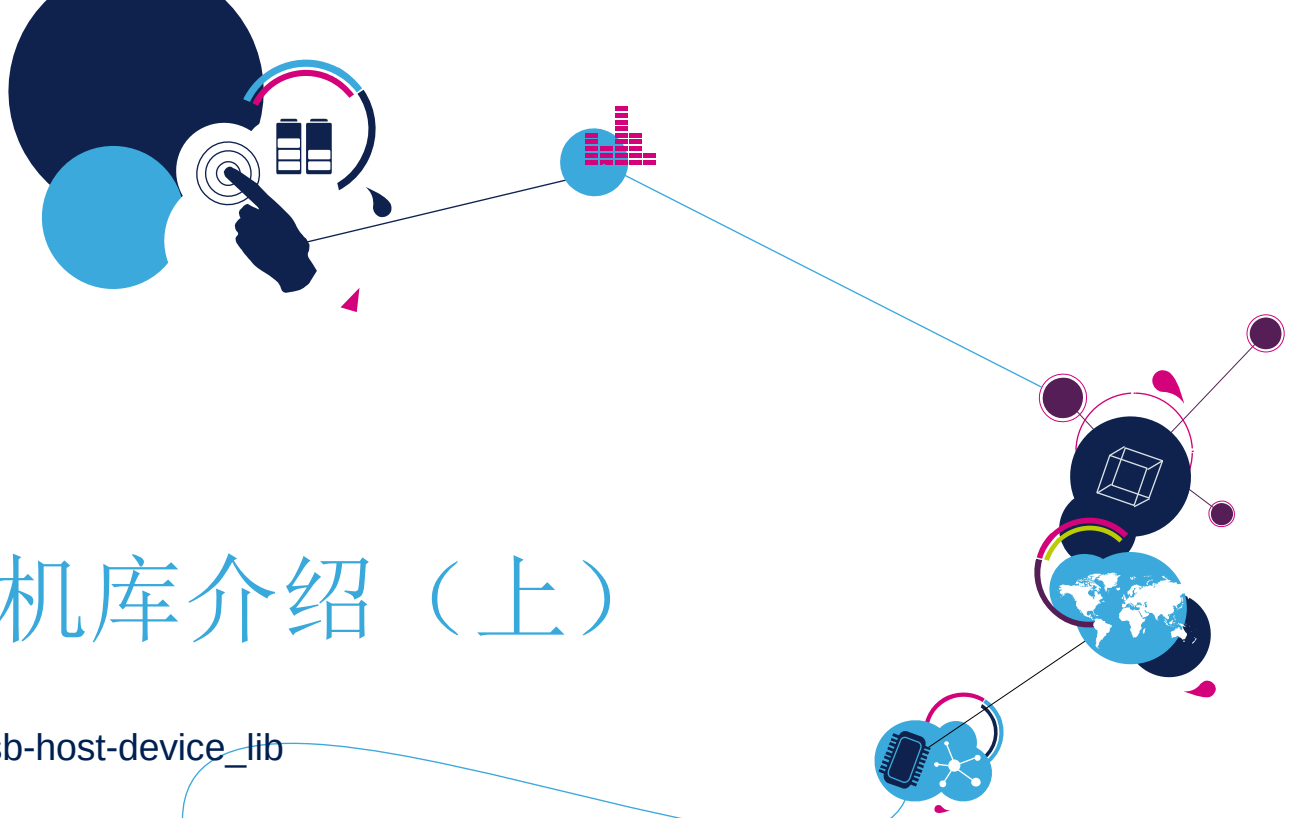


OTG IP主机库介绍（上）

STM32_f105-07_f2_f4_usb-host-device_lib

STSW-STM32046 (UM1021)

FS HID Host

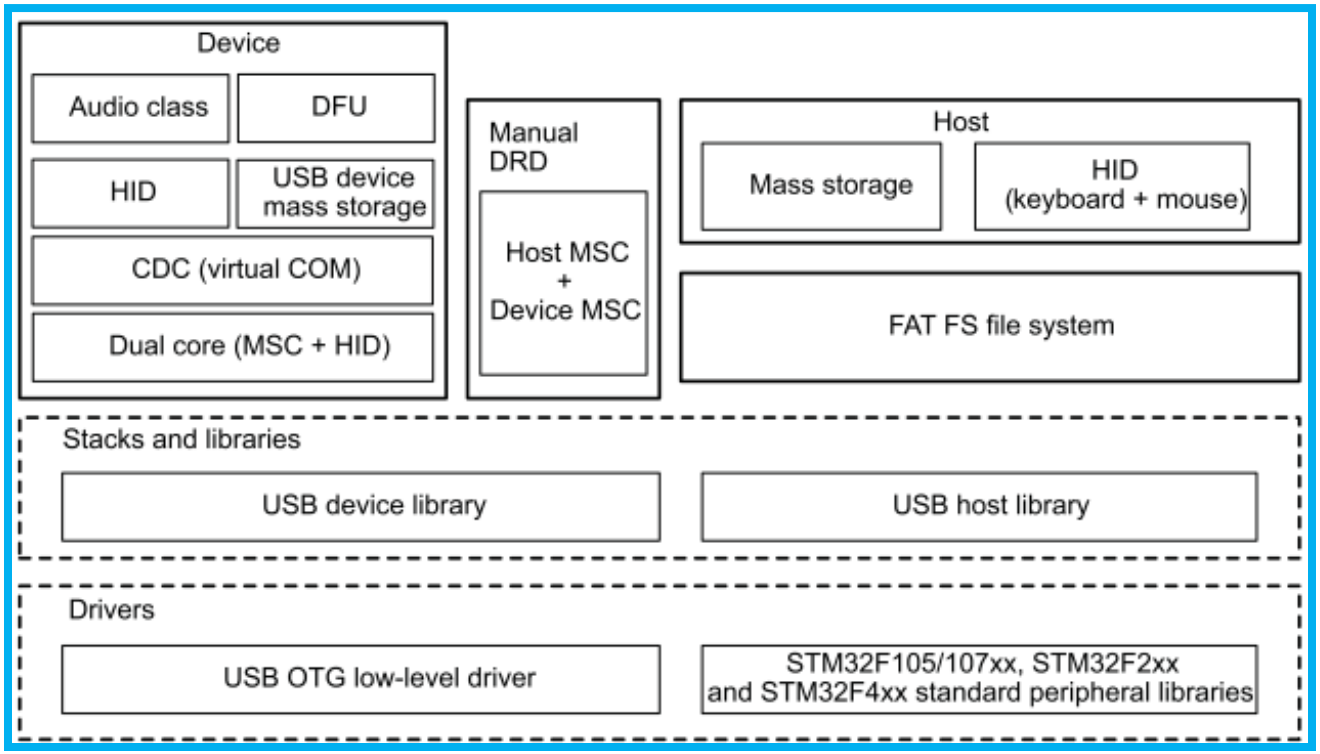
life.augmented

OTG IP主机库讲解要点

- 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
- 如何基于已有例程做自己的应用裁剪

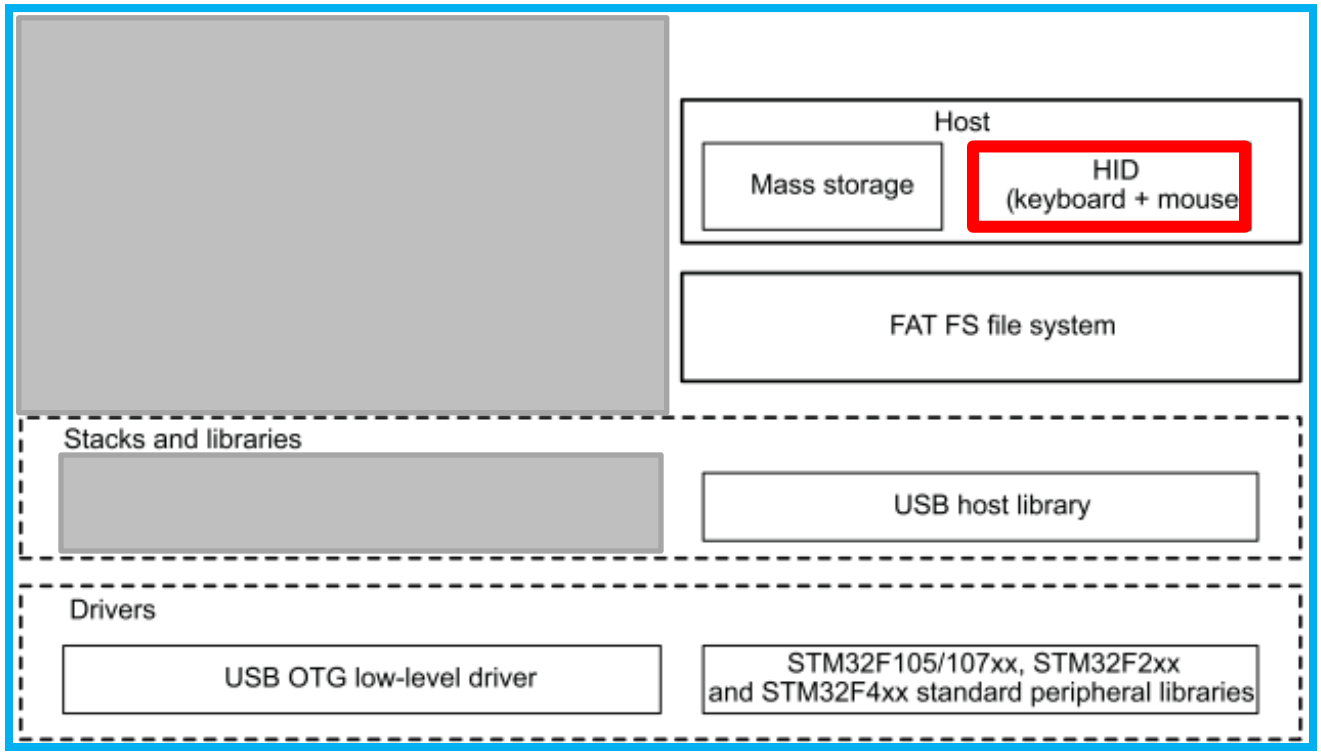
STSW-STM32046库概览

- 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



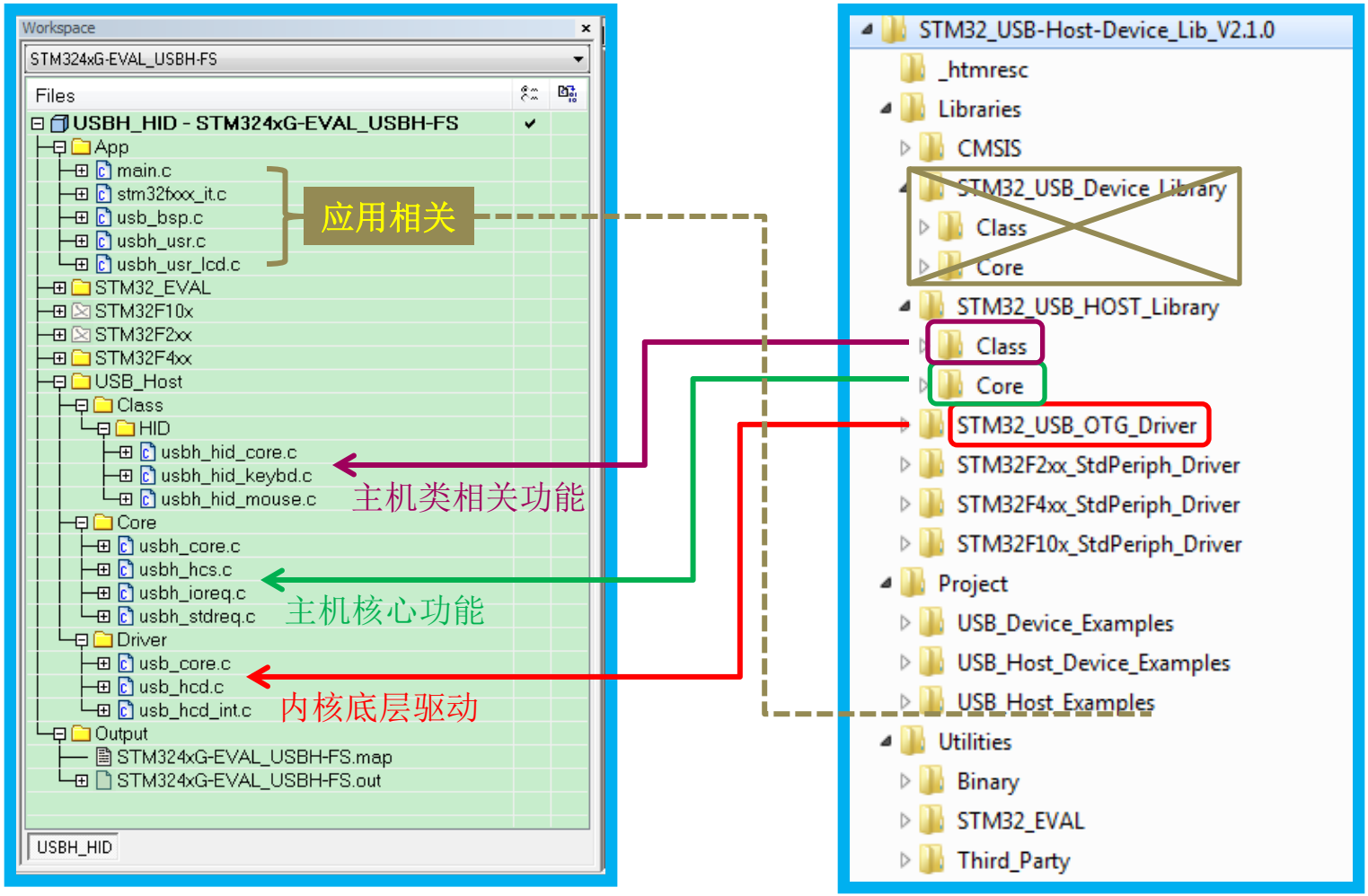
STSW-STM32046库概览

- 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

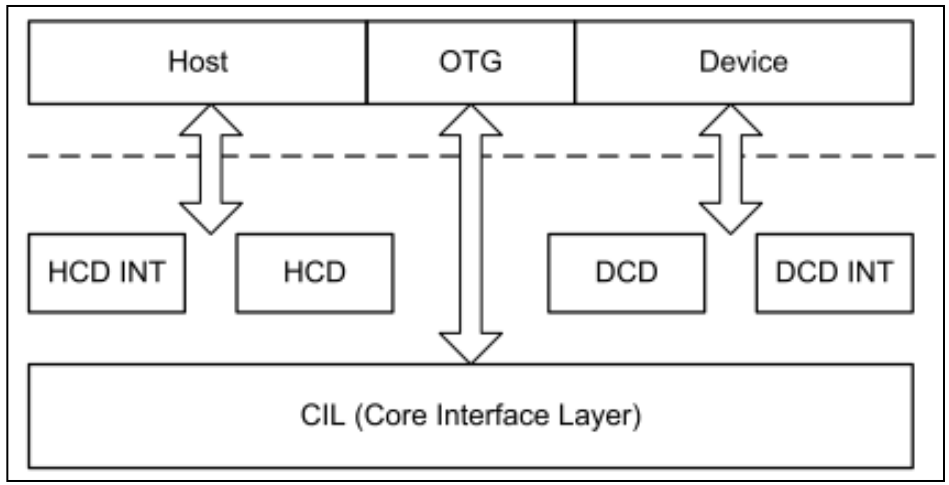
- 运行在STM324xG-EVAL板上的HID **Host**例程



切换到IAR Workbench窗口…

USB OTG底层驱动文件

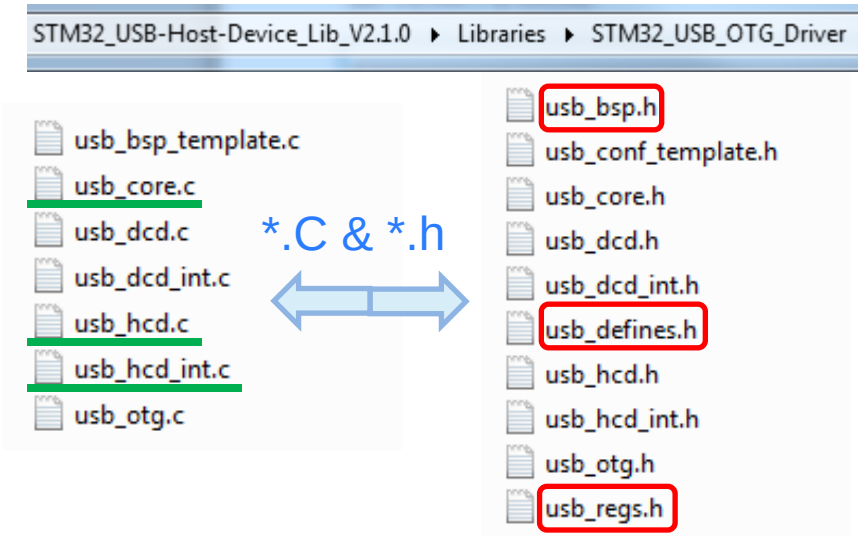
- 驱动文件架构概览



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



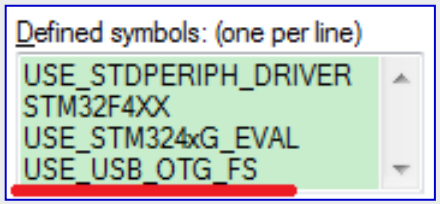
USB OTG底层驱动文件.描述

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SRP、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

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
```

<usb_conf.h>

3. USB OTG FS FIFO配置

```
#ifdef USB_OTG_FS_CORE
 #define RX_FIFO_FS_SIZE                128
 #define TXH_NP_FS_FIFOSIZ              96
 #define TXH_P_FS_FIFOSIZ               96

// #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 TXH_NP_HS_FIFOSIZ              256
 #define TXH_P_HS_FIFOSIZ               256

// #define USB_OTG_HS_LOW_PWR_MGMT_
// #define USB_OTG_HS_SOF_OUTPUT_ENA

// #define USB_OTG_INTERNAL_VBUS_ENAB
#define USB_OTG_EXTERNAL_VBUS_ENABL

#ifdef USE_ULPI_PHY
 #define USB_OTG_ULPI_PHY_ENABLED
#endif
#ifdef USE_EMBEDDED_PHY
  #define USB_OTG_EMBEDDED_PHY_ENAB
 #endif
 #define USB_OTG_HS_INTERNAL_DMA_ENA
// #define USB_OTG_HS_DEDICATED_EP1_E
#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
  - 包含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

<main.c>

```
#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_Core_dev __ALIGN_END ;
```

<usb_core.h>

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

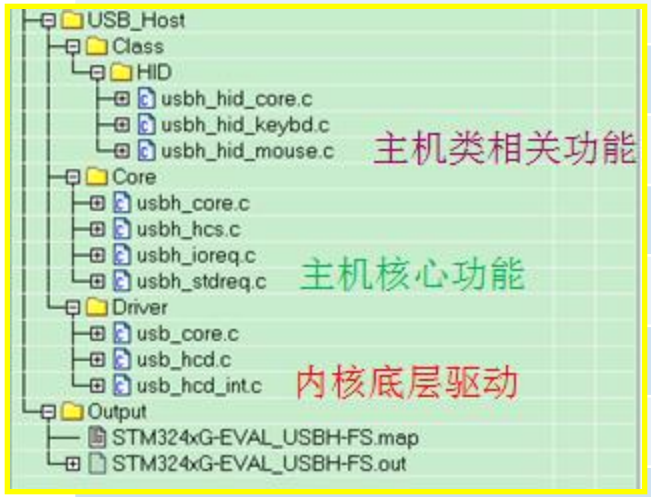
USB OTG底层驱动文件.编程模型.主机

Usb_hcd.c/.h

- 主机模块初始化
  - **HCD_Init** (**USB_OTG_CORE_HANDLE *pdev** , USB_OTG_CORE_ID_TypeDef coreID)
- 主机通道初始化
  - **HCD_HC_Init** (**USB_OTG_CORE_HANDLE *pdev** , uint8_t hc_num)
- 启动主机传输
  - **HCD_SubmitRequest** (**USB_OTG_CORE_HANDLE *pdev** , uint8_t hc_num)
  - 之后传输流就由HCD中断处理(usb_hcd_int.c/.h)，用户可通过以下API监测传输状态
  - **HCD_GetURB_State** (**USB_OTG_CORE_HANDLE *pdev** , uint8_t ch_num)
  - **HCD_GetHCState** (**USB_OTG_CORE_HANDLE *pdev** , uint8_t ch_num)
  - **HCD_GetXferCnt** (**USB_OTG_CORE_HANDLE *pdev**, uint8_t ch_num)
- USB主机监测
  - **HCD_IsDeviceConnected** (**USB_OTG_CORE_HANDLE *pdev**)
- USB主机中断
  - **USBH_OTG_ISR_Handler** (**USB_OTG_CORE_HANDLE *pdev**)

底层驱动文件 ← 主机核心功能

usbh_hcd.c 底层驱动的主机接口层	主机核心功能	具体文件
HCD_Init()	**USBH_Init**	usbh_core.c
HCD_HC_Init()	USBH_Open_Channel	usbh_hcs.c
	USBH_Modify_Channe	
HCD_SubmitRequest	USBH_CtlSendSetup	usbh_ioreq.c
	USBH_CtlSendData	
	USBH_CtlReceiveData	
	USBH_BulkSendData	
	USBH_BulkReceiveData	
	USBH_InterruptReceiveData	
	USBH_InterruptSendData	
	USBH_IsocReceiveData	
	USBH_IsocSendData	
HCD_IsDeviceConnected	**USBH_Process**	usbh_core.c
USBH_OTG_ISR_Handler	OTG_FS/HS_IRQHandler	stm32fxxx_it.c



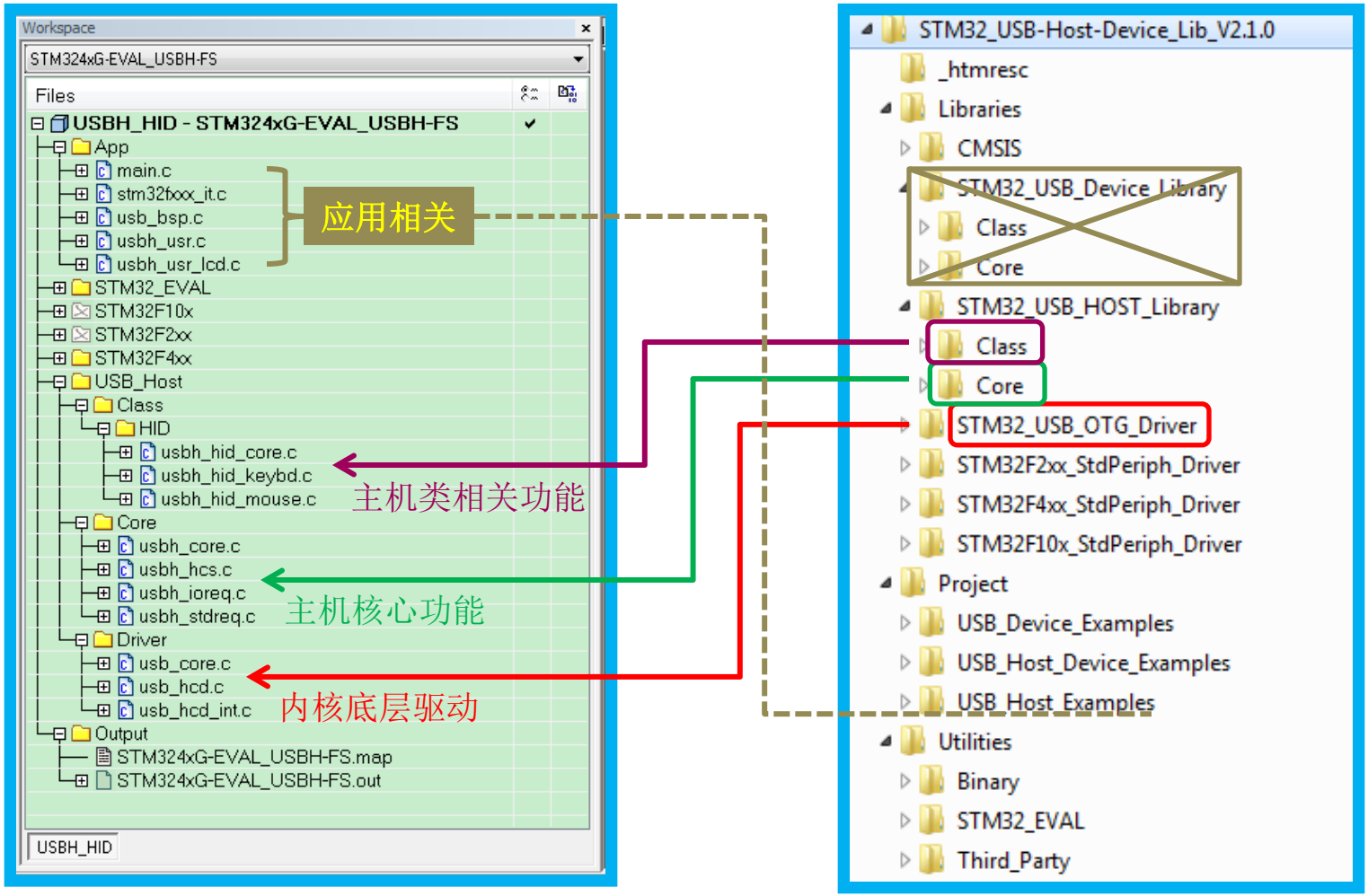
USB OTG底层驱动文件.编程模型.主机

- OTG模块句柄**USB_OTG_CORE_HANDLE**
  - → 主机驱动的核心结构体**HCD_DEV**

typedef struct _HCD {}		
uint8_t	Rx_Buffer[200]	[MAX_DATA_LENGTH], pdev→host.Rx_Buffer
__IO uint32_t	ConnSts	连接状态；HCD_IsDeviceConnected()
__IO uint32_t	ErrCnt[15]	一个通道上一次传输的错误计数
__IO uint32_t	XferCnt[15]	收到并存放在Rx_Buffer中的IN数据个数：GetXferCnt()
__IO HC_STATUS	HC_Status[15]	一个通道当前transaction的状态；driver内部使用、也可由上层应用调用
__IO URB_STATE	URB_State[15]	一个通道当前transfer的状态
USB_OTG_HC	hc[15]	一个通道的属性：所连设备地址、端点编号、速度、方向、传输类型、传输记录
uint16_t	channel[15]	记录主机每个通达的状态(used or free)
USB_OTG_hPort_TypeDef	*port_cb	主机端口回调函数，用户用来处理连接和断开事件的应用相关操作

项目结构

- 运行在STM324xG-EVAL板上的HID **Host**例程



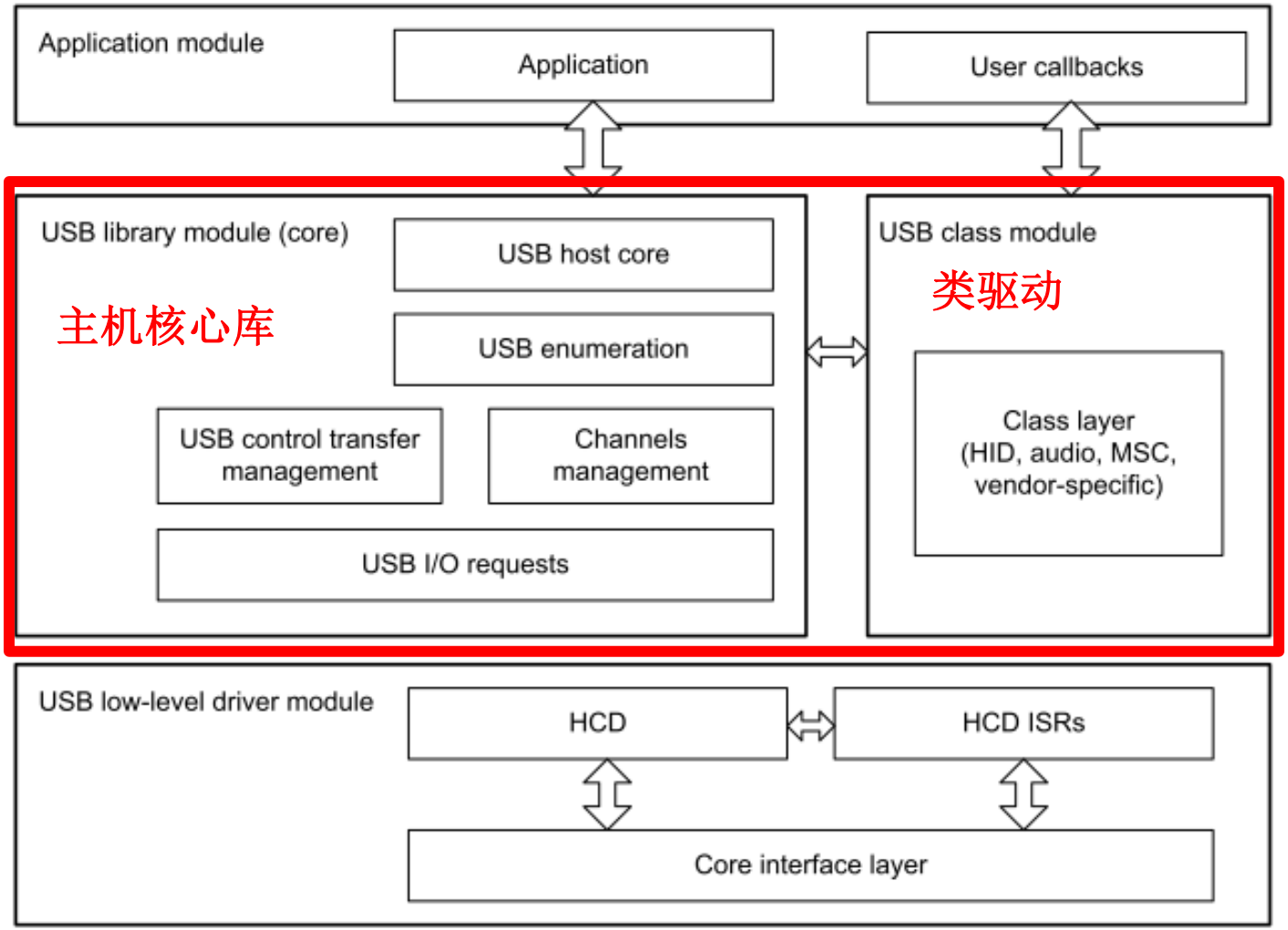
切换到IAR Workbench窗口…

USB OTG主机库特性概览

- 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
- 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
- 基于全局状态机
- 对RTOS全兼容

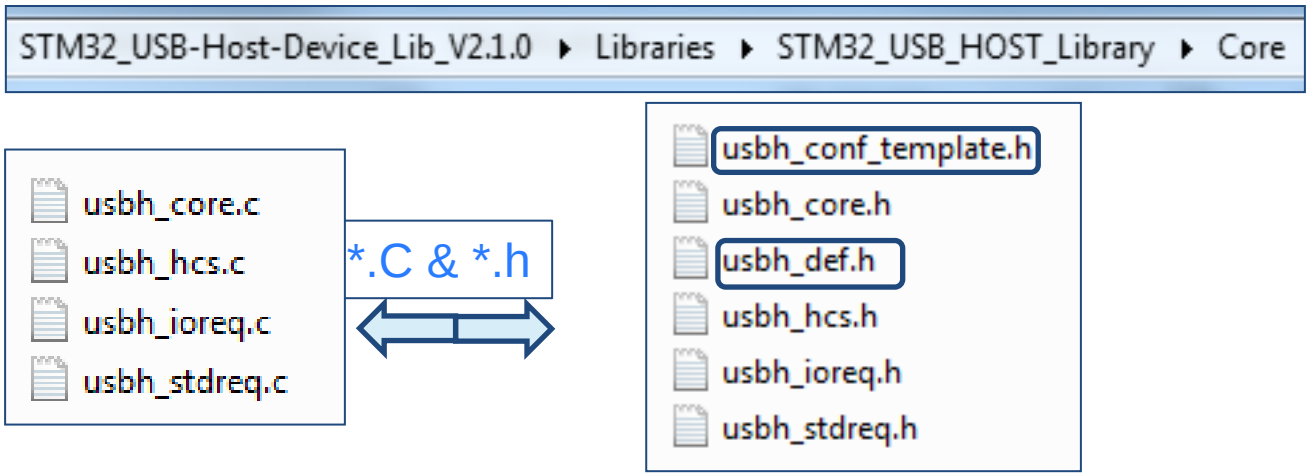
USB OTG主机核心库.架构

- 主机核心功能文件架构概览



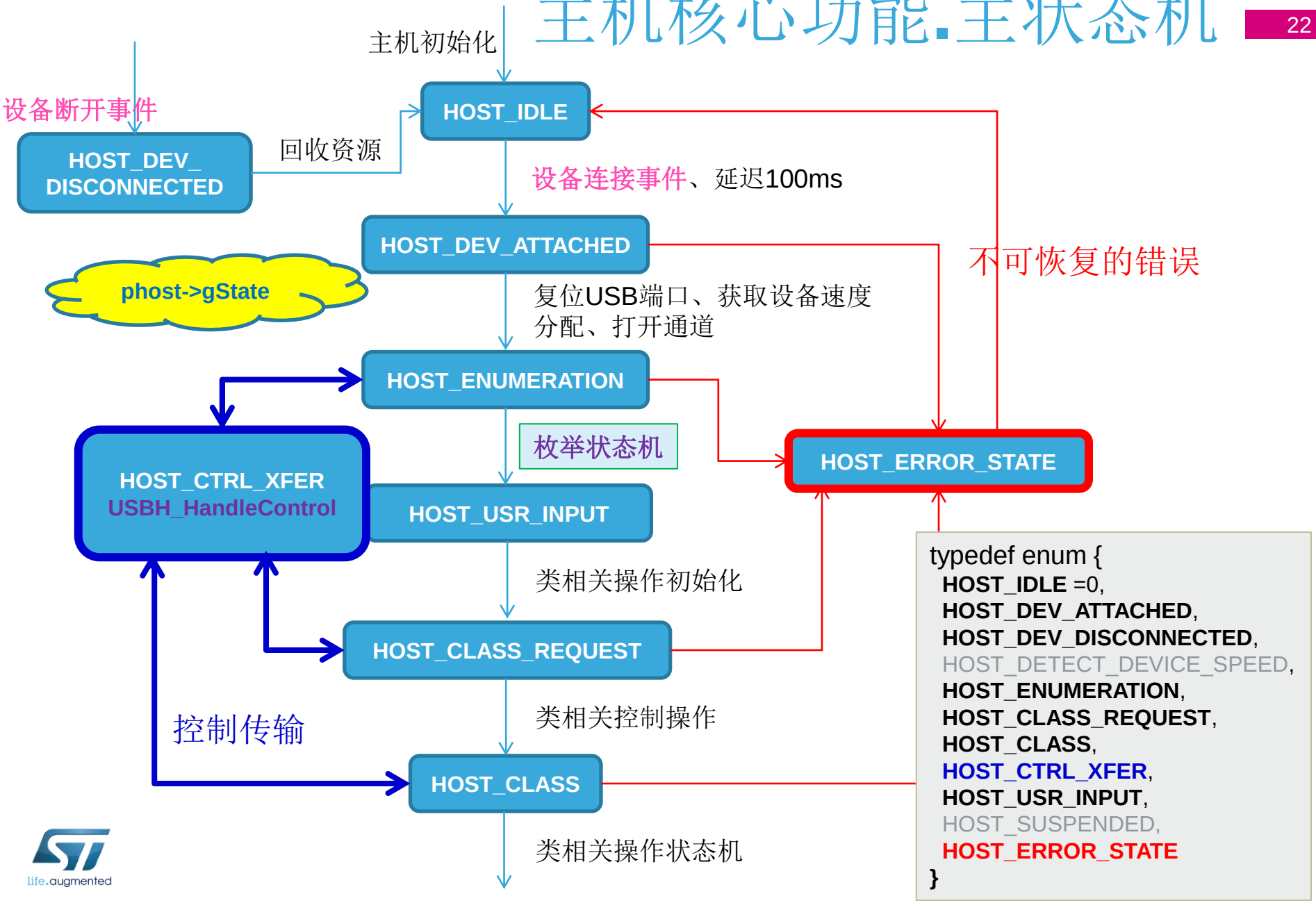
USB OTG主机核心库.文件&配置

- 主机核心功能文件及描述



文件	描述
usbh_core.c/.h	处理所有USB通信： >> 枚举状态机、控制传输状态机 >> 两个供用户调用主要API
usbh_stdreq.c/.h	USB 2.0协议第九章中的标准request
usbh_ioreq.c/.h	处理USB 2.0协议的四种传输
usbh_hcs.c/.h	处理主机通道的分配和处理
usbh_conf.h	所支持设备的Interface, configuration的个数 #define USBH_MAX_NUM_ENDPOINTS 2 #define USBH_MAX_NUM_INTERFACES 2

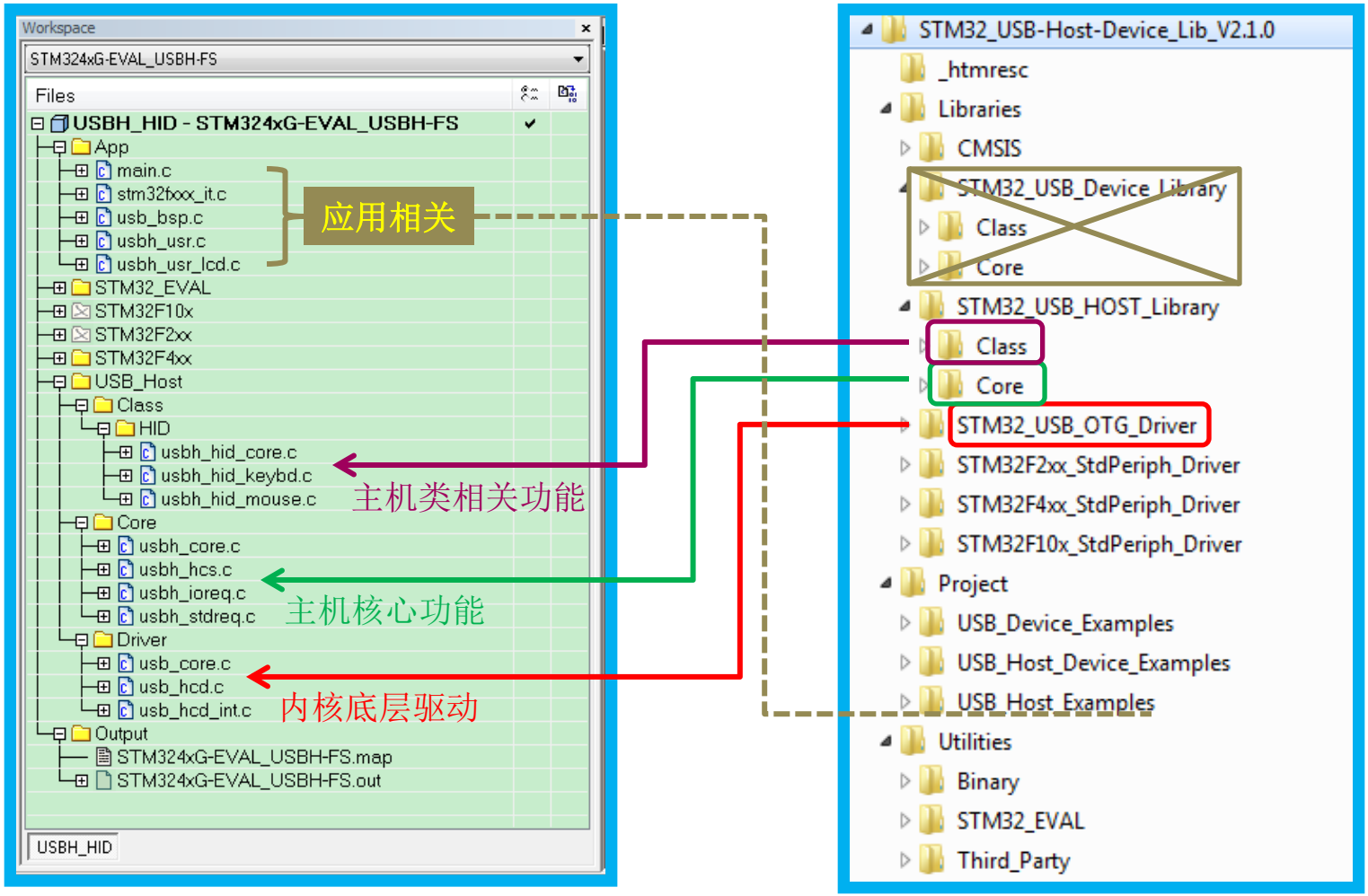
主机核心功能.主状态机



life.augmented

项目结构

- 运行在STM324xG-EVAL板上的HID **Host**例程



切换到IAR Workbench窗口…

主机的类驱动回调

- 在主机库初始化时指定类驱动

```
USBH_Init (&USB_OTG_Core_dev,  USB_OTG_FS_CORE_ID,
           &USB_Host,    &HID_cb,    &USR_Callbacks);

@ <usbh_hid_core.c>
USBH_Class_cb_TypeDef  HID_cb =
{
  USBH_HID_InterfaceInit,
  USBH_HID_InterfaceDeInit,
  USBH_HID_ClassRequest,
  USBH_HID_Handle

};
```

```
typedef struct _USBH_Class_cb
{
  USBH_Status  (*Init);
  void         (*DeInit);
  USBH_Status  (*Requests);
  USBH_Status  (*Machine);

} USBH_Class_cb_TypeDef;
```

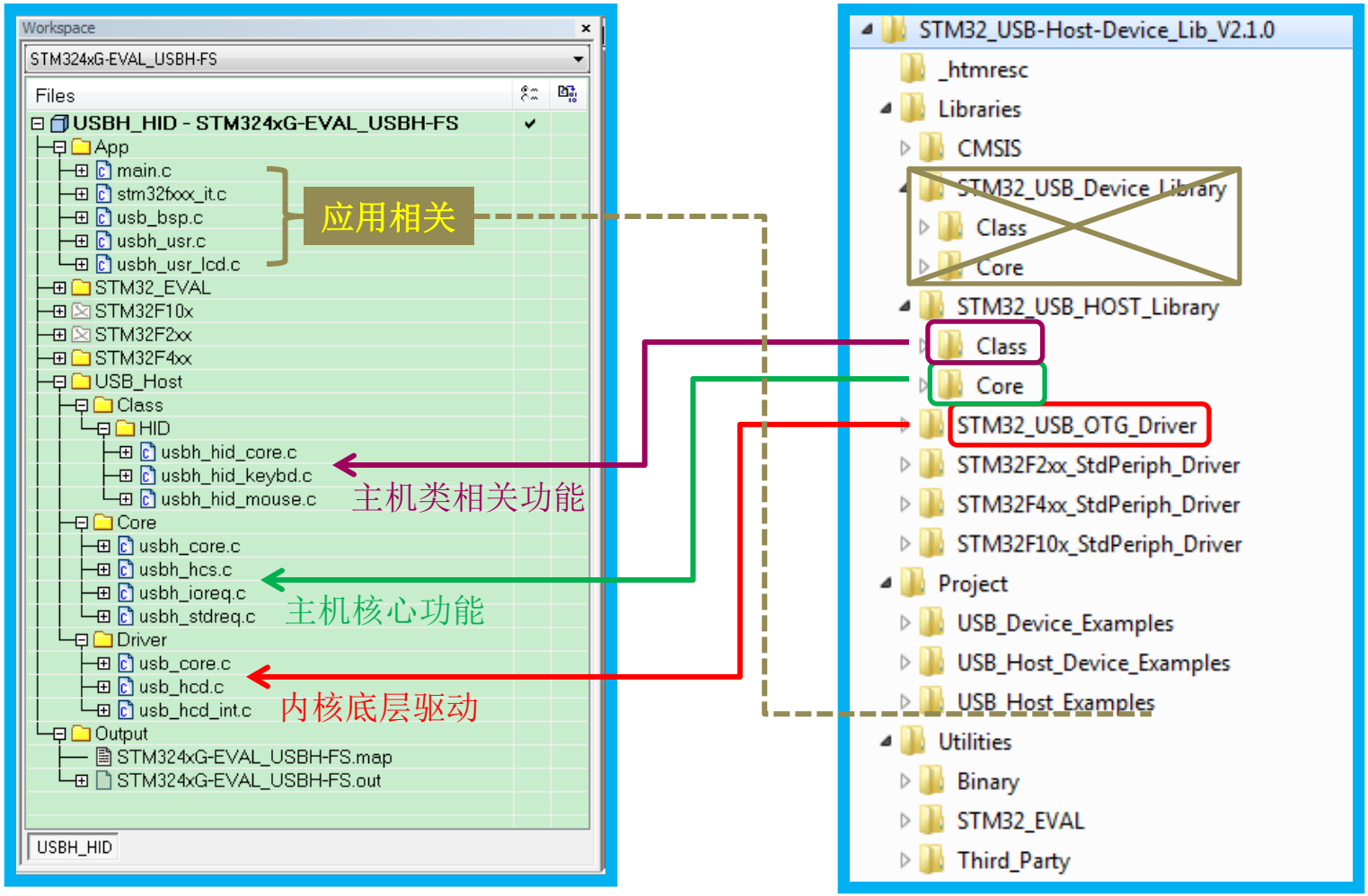
HID类

- 支持HID boot鼠标和键盘设备
- 通过中断IN传输来获得HID report (而不是Get report的控制传输)

函数	描述
USBH_HID_InterfaceInit	解析接口和端点描述符，配置主机通道以建立获取HID report所需的中断传输IN管道
USBH_HID_InterfaceDeInit	释放之前分配的IN管道
USBH_HID_ClassRequest	HID类相关命令：获得HID report描述符；设置IDLE时间；设置Protocol…
USBH_HID_Handle	HID类核心功能状态机：定时发起中断IN传输
USBH_Get_HID_Descriptor	HID类命令：获得HID类描述符
USBH_Get_HID_ReportDescriptor	HID类命令：获得HID report描述符
USBH_ParseHIDDesc	解析HID report描述符
USBH_Set_Idle	HID类命令：设置IDLE时间
USBH_Set_Report	HID类命令：发送Report OUT数据（demo中未用）
USBH_Set_Protocol	HID类命令：设置HID协议

项目结构

- 运行在STM324xG-EVAL板上的HID **Host**例程



切换到IAR Workbench窗口…

USB Host Stack用户接口

- 两个用户API
  - void **USBH_Process** (void) 在用户应用程序主循环中循环调用
  - void **USBH_Init** (P1, P2, P3, P4, P5)
    - P1 = pdev，指向USB OTG模块句柄的指针
    - P2 = CoreID，实际使用哪个OTG IP
    - P3 = phost，指向USB主机状态机的指针
    - P4 = class_cb，**类驱动回调函数**，在主机内核完成标准枚举后，由它来进行类相关的操作 @ <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 @ <usbh_usr.c>
  - 用户类回调函数
  - 用户普通回调函数- USBH_Init()的P5

用法举例

@ <MSC Host demo>

```
USBH_Init(&USB_OTG_Core,
#ifdef USE_USB_OTG_FS
          USB_OTG_FS_CORE_ID,
#else
          USB_OTG_HS_CORE_ID,
#endif
          &USB_Host, &USBH_MSC_cb, &USR_cb);
```

@ <HID Host demo>

```
USBH_Init(&USB_OTG_Core_dev,
#ifdef USE_USB_OTG_FS
          USB_OTG_FS_CORE_ID,
#else
          USB_OTG_HS_CORE_ID,
#endif
          &USB_Host,  &HID_cb,  &USR_Callbacks);
```

@ <Dual Core Host demo>

```
/* Init HS Core */
 USBH_Init(&USB_OTG_Core,
           USB_OTG_HS_CORE_ID,
           &USB_Host,
           &USBH_MSC_cb,
           &USR_MSC_cb);


 /* Init FS Core */
 USBH_Init(&USB_OTG_FS_Core,
           USB_OTG_FS_CORE_ID,
           &USB_FS_Host,
           &HID_cb,
           &USR_HID_cb);
```

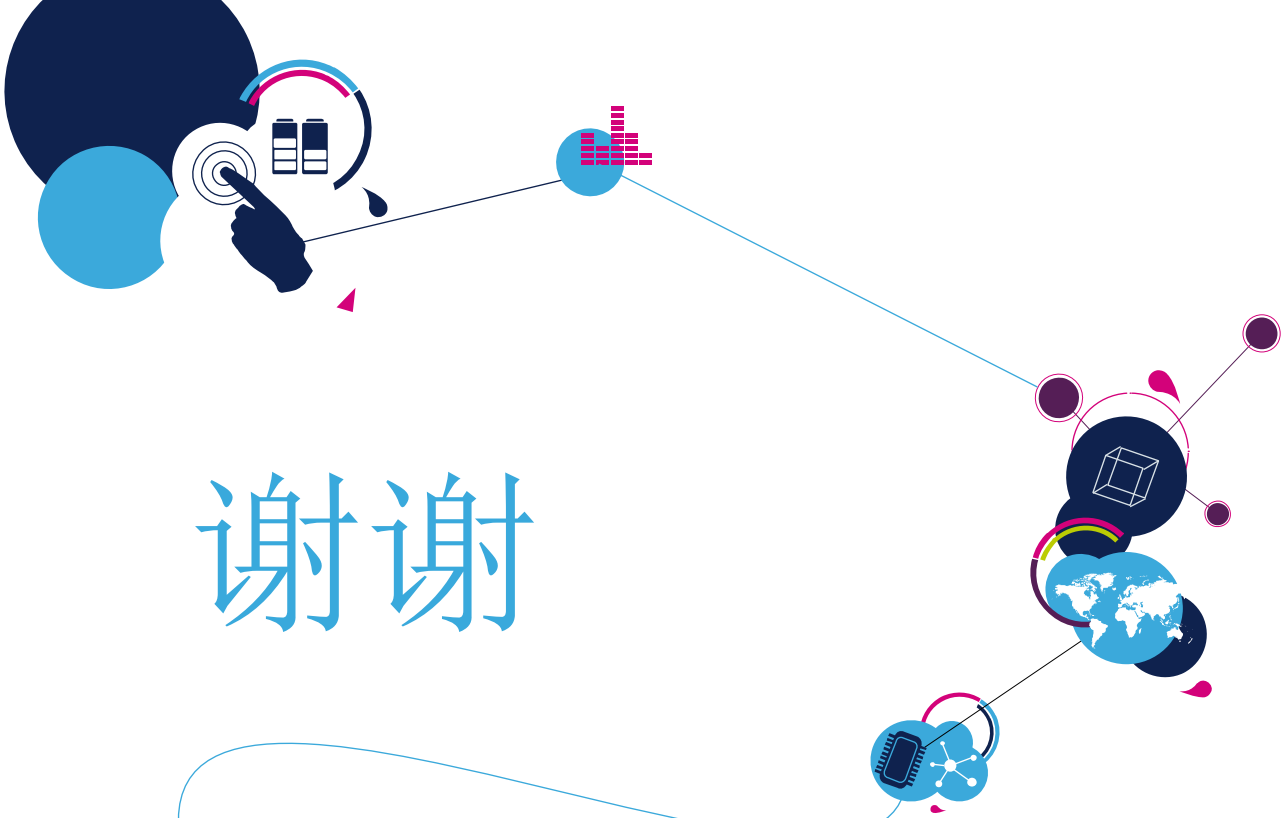
运行USB Host Stack

- 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 @ <usbh_usr.c>
- 定义两个结构体 @ <app.c>
- 调用两个API @ <app.c>

```
USB_OTG_CORE_HANDLE          USB_OTG_Core_dev;
USBH_HOST                    USB_Host;

int main(void)
{
   USBH_Init (&USB_OTG_Core_dev,  USE_USB_OTG_FS, &USB_Host, &HID_cb,  &USR_Callbacks);

   while (1)
    {
      USBH_Process (&USB_OTG_Core_dev , &USB_Host);
      …. other task…
    }
}
```



谢谢

life.augmented

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.