STM32 OTG_FS/HS模块

Lilian YAO

life.augmented

STM32 MCU有两种带USB功能的IP

- USB IP
  - 可作为全速或低速的USB设备
  - 存在于STM32F102、STM32F103
- FS OTG IP
  - 可作为全速和低速USB主机
  - 可作为全速USB设备
  - 存在于STM32F105、STM32F107、STM32F2、STM32F4
- HS OTG IP
  - 可作为高速、全速和低速USB主机
  - 可作为高速和全速USB设备
  - 存在于STM32F2、STSM32F4

本PPT讲解OTG IP

	USB	OTG	
	FS	FS	HS
STM32F102/103	Y		
STM32F105/107		Y	
STM32F2/F4		Y	Y

OTG_FS和OTG_HS模块通用特性比较

OTG_FS	OTG_HS
USB 2.0协议，OTG 1.3协议	
可作为USB主机、USB设备、OTG设备(A类/B类)使用	
	支持3种PHY
可使用内部FS PHY做FS通信	
	具有ULIP接口，可和外部HS PYH连接做HS通信
输出SOF信号，供各种同步应用	
省电功能	
FIFO使用1.25KB专用RAM	FIFO使用4KB专用RAM
	内置独立的DMA管理FIFO的数据传输

life.augmented

两个模块的主机、设备特性比较

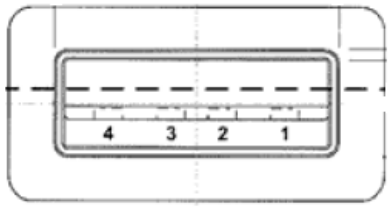
OTG_FS	OTG_HS
主机特性比较	
需要外接电源芯片为所连的USB设备供电	
2个请求队列 >> 周期性队列：管理最多8个ISO、INTERRUPT传输请求 >> 非周期性队列：管理最多8个CONTROL、BULK传输请求	
8个主机通道	12个主机通道
专用TXFIFO >> 周期性TXFIFO：存储需要传输的ISO、INTERRUPT传输数据 >>非周期性TXFIFO：存储需要传输的CONTROL、BULK传输数据	
一个共享的RXFIFO用以接收数据	
设备特性比较	
4个双向端点(包括端点0)	6个双向端点(包括端点0)
4个独立的TX FIFO对应于4个IN端点	6个独立的TX FIFO对应于6个IN端点
1个共享的RX FIFO	
支持软件断开	

两个模块的省电特性相同

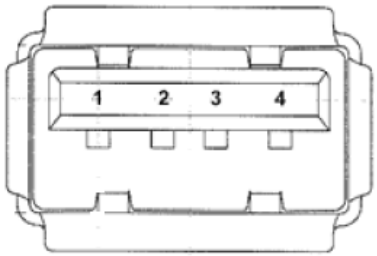
- OTG_PHY的功耗
  - PHY： PWRDWN@GCCFG
  - A-VBUS监控： VBUSASEN@GCCFG
  - B-VBUS监控： VBUSASEN@GCCFG

- 挂起模式下的功耗
  - 停止PHY时钟(48MHz时钟区域)：STPPCLK@PCGCCTL
  - 停止系统时钟(HCLK时钟区域)：GATEHCLK@PCGCCTL
  - 进入系统停止模式

USB 连接器：标准接口/Mini接口

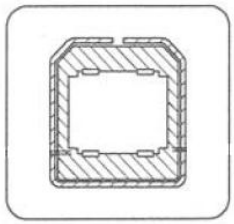


Standard A Plug

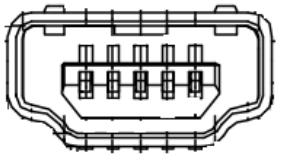


Standard A Receptacle

在PC端

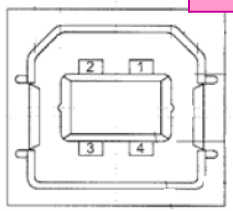


Standard B Plug

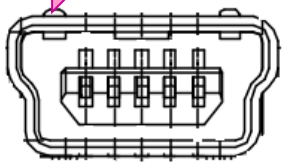


Mini B Plug

体积更小……



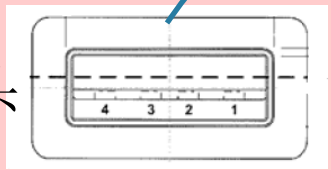
Standard B Receptacle



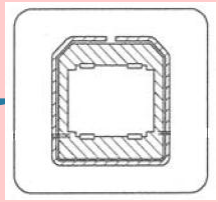
Mini B Receptacle

在设备端

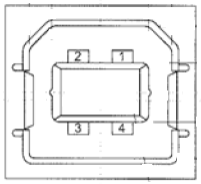
一根电缆



标准A插头

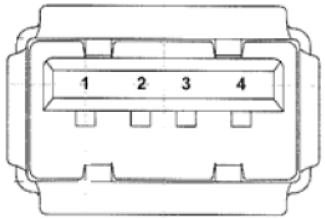


标准B插头

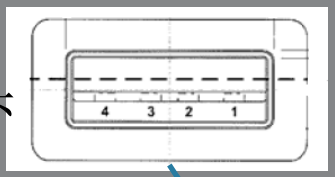


设备板子上的
标准B插座

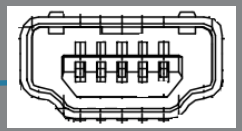
PC上的
标准A插头



标准A插头

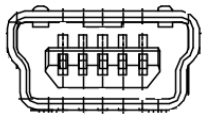


一根电缆

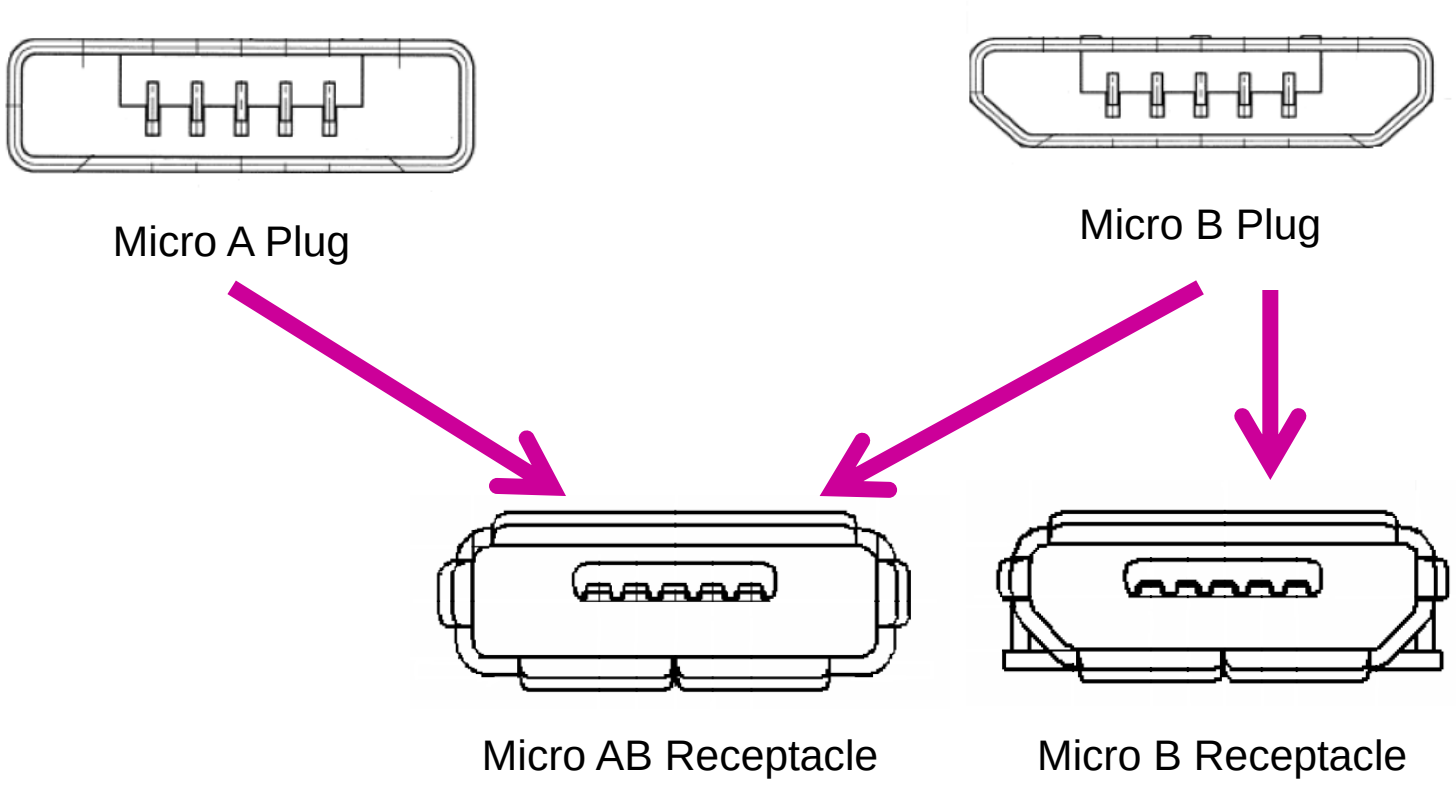


miniB插头

设备板子上的
miniB插座

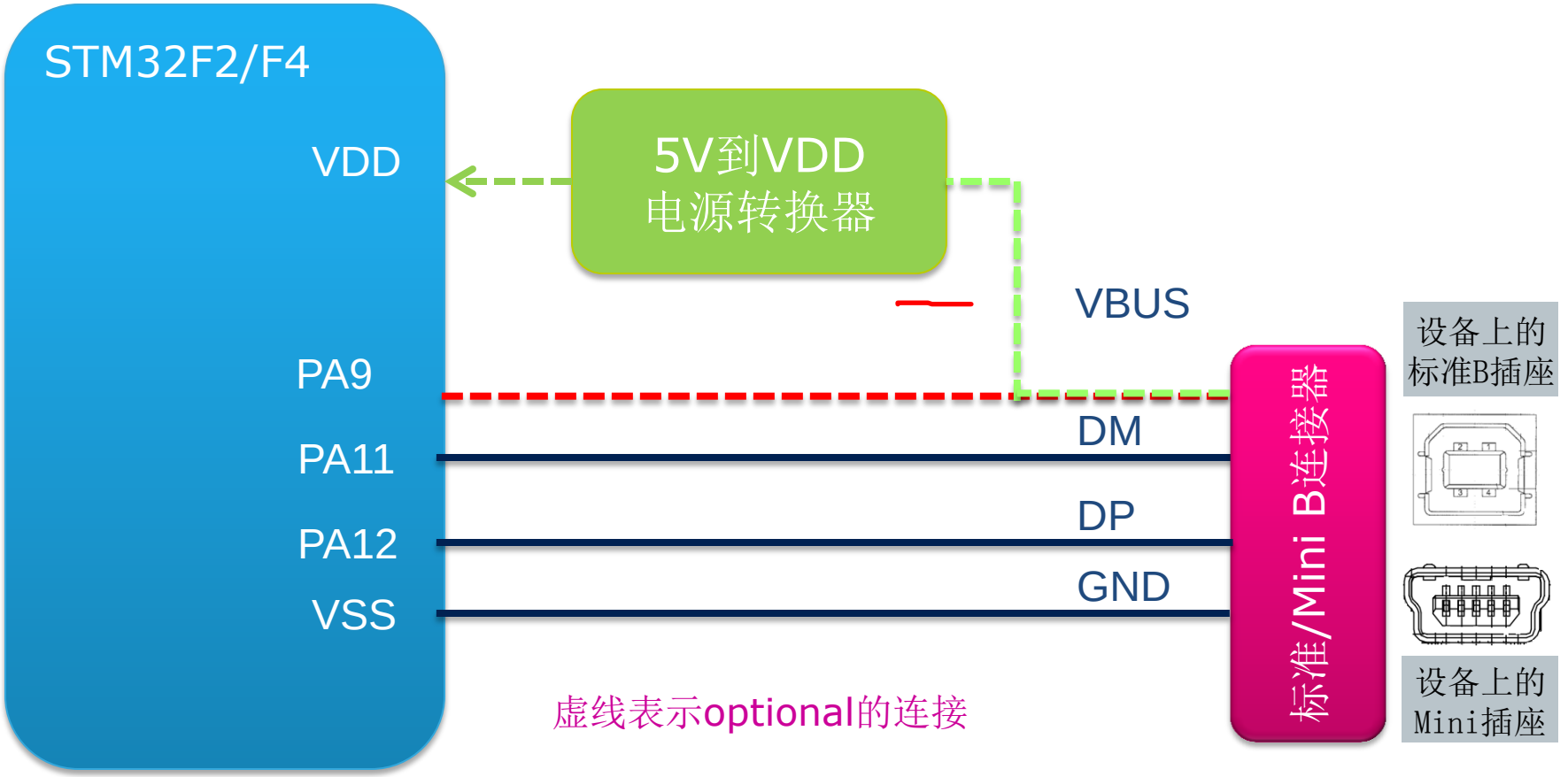


USB 连接器：Micro接口



life.augmented

OTG_FS模块作为设备的连接示意图



>> 电源转换器可从插座的Vbus获取电源作为MCU供电→总线供电设备

>> PA9用以监测VBUS的供电：可用作监测是否和主机断开连接

>> 取消该监控时(NOVBUSSENS)，PA9可用作普通I/O口，则VBUS默认一直存在

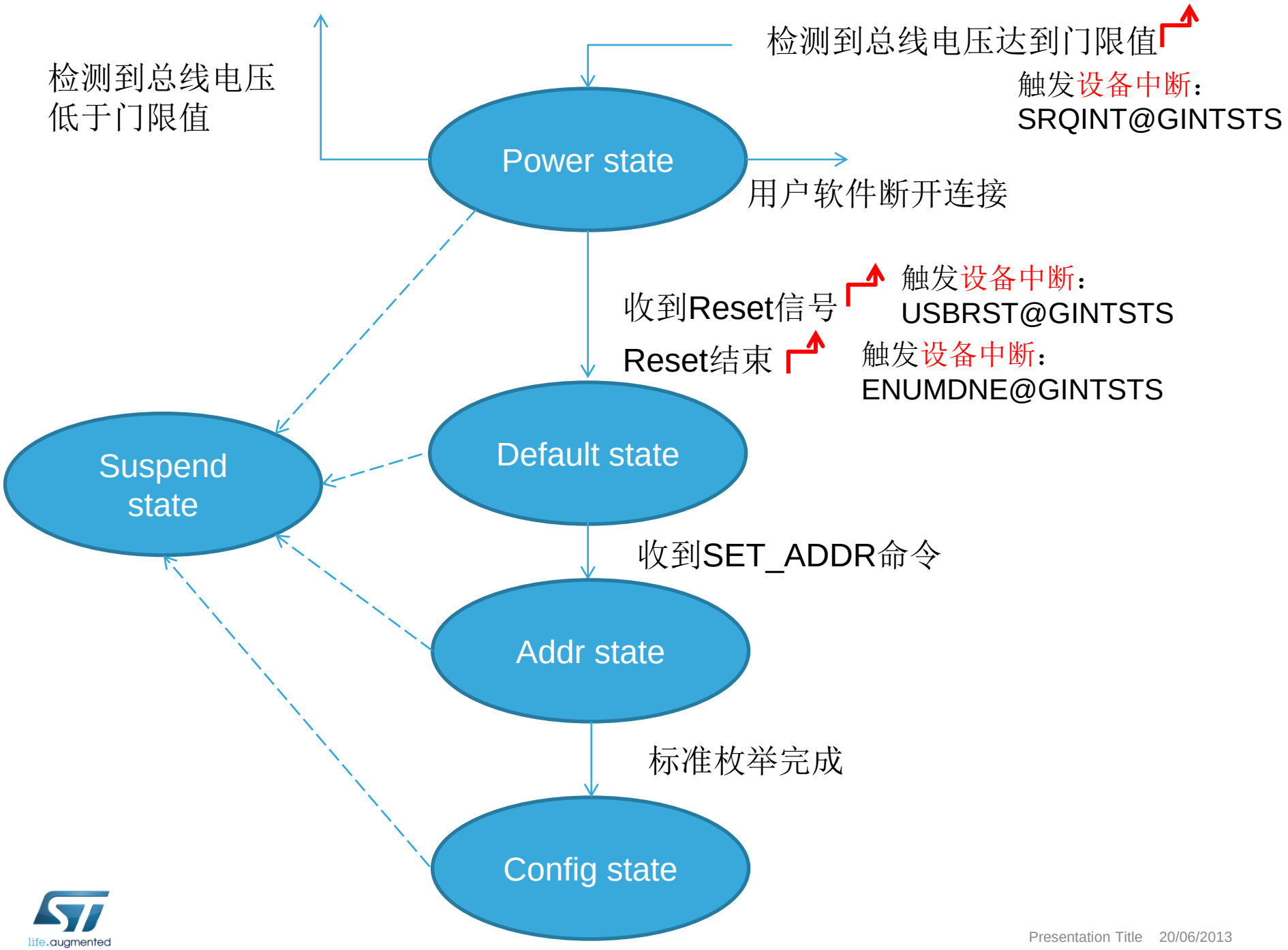
life.augmented

OTG_FS模块作为设备

- 使用PA.9监测USB总线电压，是可选的
  - PA.9连到插槽的Vbus引脚
    - 检测到B-session有效电压，自动使能D+的上拉电阻；触发**设备中断**：SRQINT@GINTSTS
    - 检测到Vbus低于B-session有效电压，自动断开上拉电阻；触发**设备中断**：SEDET@GOTGINT
  - PA.9也可作为GPIO使用（需搭配软件的配置：NOVBUSSENS@OTG_FS_GCCFG）
- 支持软件控制断开和主机的连接（关闭D+上拉）
  - SDIS@DCTL，可触发主机端的设备断开中断
- 无需在D+和D-数据线上放串行电阻（阻抗匹配已满足USB规范）
- 设备的挂起模式
  - 总线空闲3ms → 触发**设备中断**：ESUSP@GINTSTS
  - 3ms后 → 触发**设备中断**：USBSUSP@GINTSTS
  - 可通过主机发送的resume信号唤醒（触发设备中断：RWUSIG@GINTSTS）
  - 也可通过设备自身发送唤醒信号（置位WKUPINT@DCTL并在1~15ms后复位它）

Vbus sensing的作用

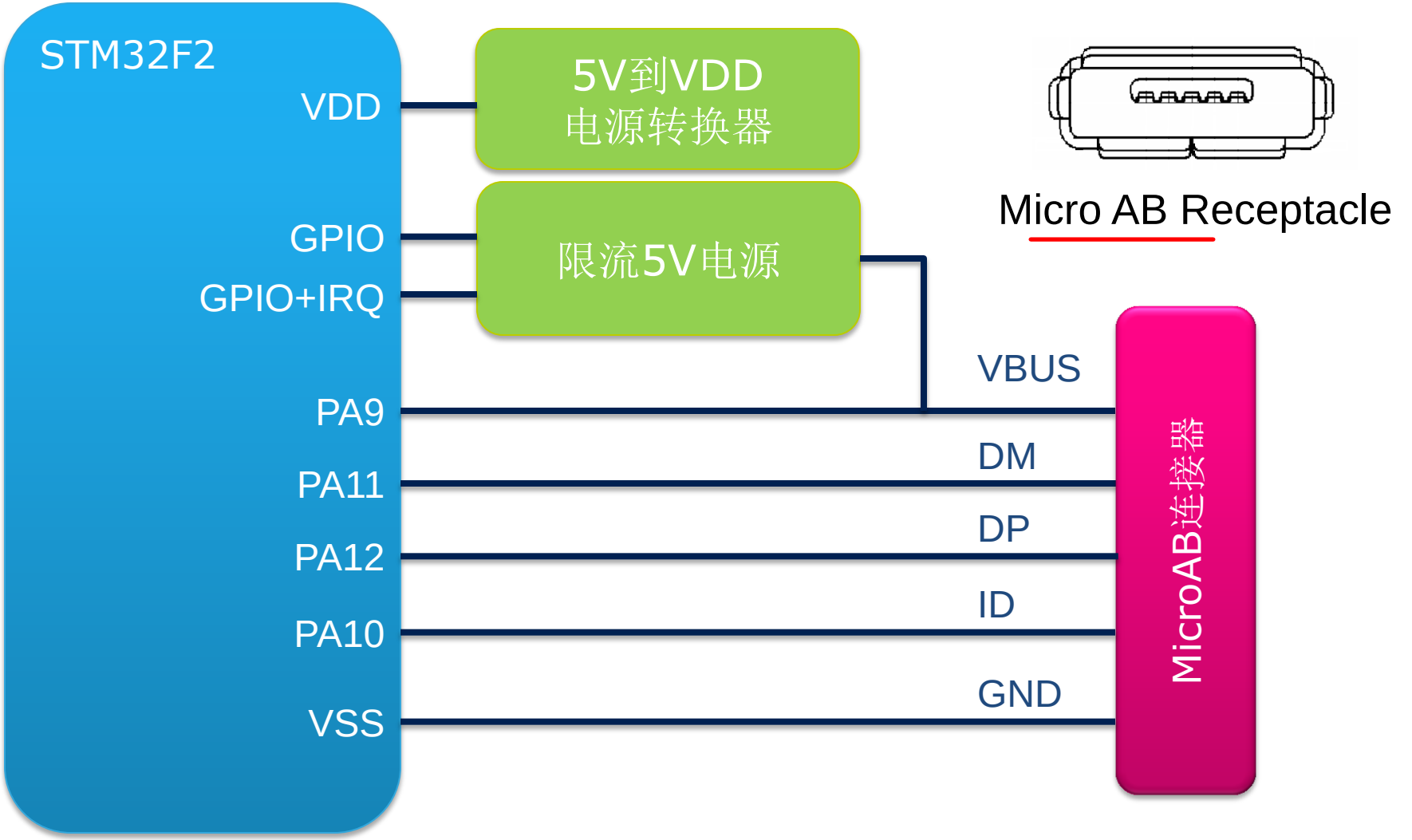
- OTG模式
  - 用于检测SRP信号
- Host模式
- Device模式
  - 用于检测和host连接或断开连接 by SRQINT@OTG_FS_GINTSTS
  - 如果软件禁用该功能（硬件上不连PA9）
    - USB模块会默认认为USB总线一直由主机供电的，因此内嵌的DP数据线上的上拉就会自动有效；这种情况下，让可以通过设置SDIS@OTG_FS_DCTL来手动断开该上拉
    - 这样可以使用某个空闲GPIO连到插座的Vbus引脚，通过检测Vbus的上升下降沿确定设备是否和主机连接
    - 还可以用D+对应GPIO对应的EXTI检测上升、下降沿（如果该设备严格符合USB规范：在检测到Vbus时才会使能D+的上拉）


检测到总线电压
低于门限值
检测到总线电压达到门限值
触发设备中断：
SRQINT@GINTSTS
Power state
用户软件断开连接
收到Reset信号
触发设备中断：
USBRST@GINTSTS
Reset结束
触发设备中断：
ENUMDNE@GINTSTS
Suspend
state
Default state
收到SET_ADDR命令
Addr state
标准枚举完成
Config state

OTG作设备的四种情况

- OTG A 器件设备
  - OTG A器件（OTG器件被插入MicroA插头）经过HNP切换到了设备角色
- OTG B 器件设备
  - OTG B 器件（OTG器件被插入MicroB插头）在电缆刚刚连接时是作为设备的角色
- B 器件
  - OTG 器件被插入MicroB插头（ID线使得此次该器件为设备角色），但HNP功能被关闭（HNPCAP@GUSBCFG=0）
  - 在保持此连接的情况下，只能作为主机角色了
- 只作为设备
  - 该器件工作于“只作为设备”模式：FDMOD@GUSBCFG=1
  - 即使插座上有ID线，也被忽略，不作为此器件角色的决定因素

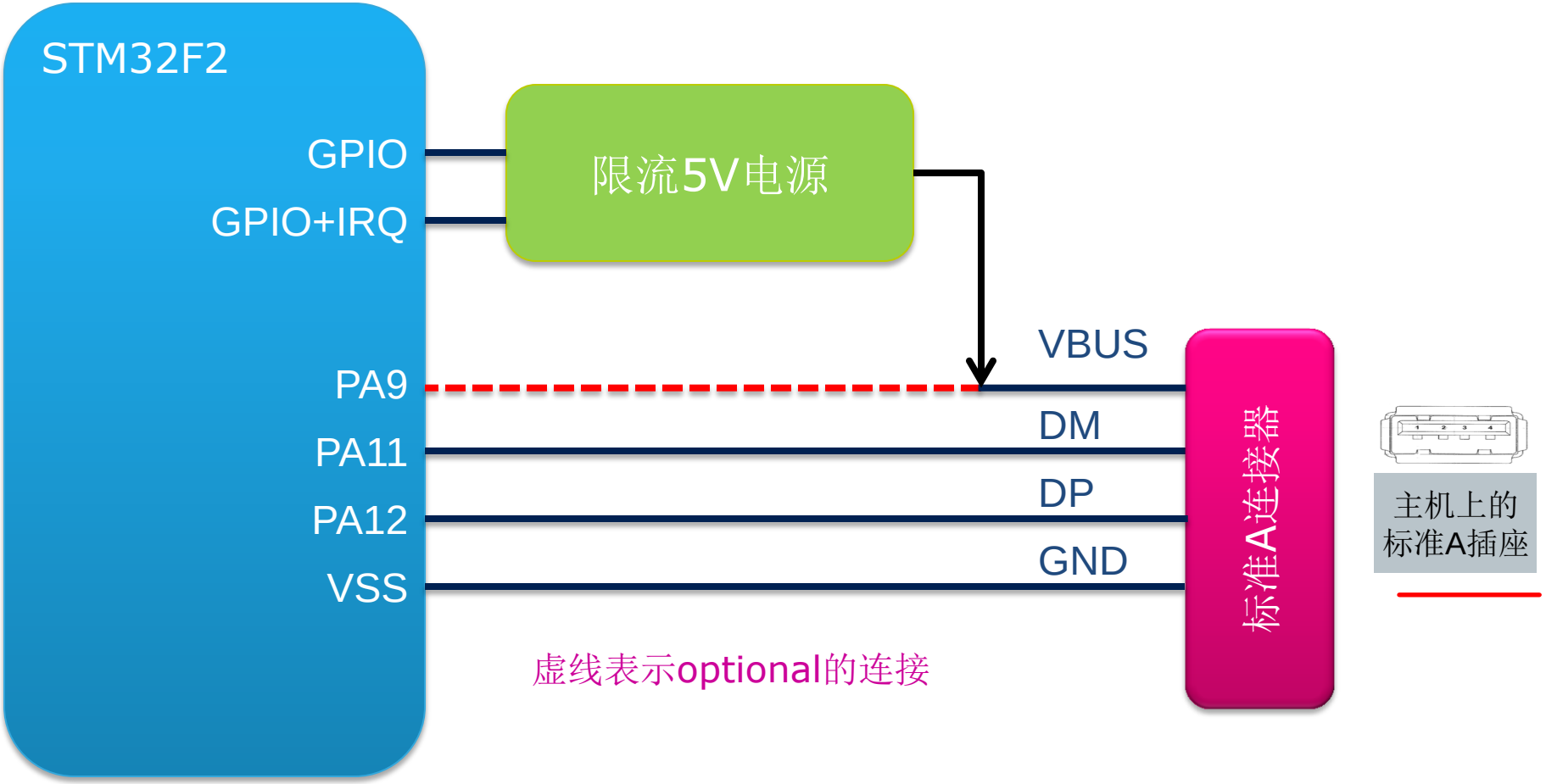
OTG_FS模块作为OTG



Micro AB Receptacle

life.augmented

OTG_FS模块作为主机的连接示意图



>> PA9用以监测VBUS的供电：可用作监测是否和主机断开连接

>>取消该监控时(NOVBUSSENS)，PA9可用作普通I/O口，则VBUS默认一直存在

life.augmented

OTG作主机的四种情况

- OTG A 器件主机
  - OTG A设备（OTG器件被插入MicroA插头）在电缆刚刚连接时是作为主机的角色
- OTG B 器件主机
  - OTG B 设备（OTG器件被插入MicroB插头）经过HNP切换到了主机角色
- A 器件
  - OTG 器件被插入MicroA插头（ID线使得此次该器件为主机角色），但HNP功能被关闭（HNPCAP@GUSBCFG=0）
  - 在保持此连接的情况下，只能作为主机角色了
- 只作为主机（上页如图）
  - 该器件工作于“只作为主机”模式：FHMOD@GUSBCFG=1
  - 即使插座上有ID线，也被忽略，不作为此器件角色的决定因素

做USB主机的四种情况之一

- USBH_MSC Demo只作为主机
  - 内部集成的D+、D-上的下拉电阻被自动使能
  - FHMOD@GUSBCFG=1

USBH_Init() → HCD_Inint() → USB_OTG_SetCurrentMode(pdev, HOST_MODE)

```
USB_OTG_STS USB_OTG_SetCurrentMode(USB_OTG_CORE_HANDLE *pdev , uint8_t mode)
{
……….

  if ( mode == HOST_MODE)
    usbcfg.b.force_host = 1;

  else if ( mode == DEVICE_MODE)
    usbcfg.b.force_dev = 1;

  USB_OTG_WRITE_REG32(&pdev->regs.GREGS->GUSBCFG, usbcfg.d32);
……..
}
```

FHMOD: Force host mode

Writing a 1 to this bit forces the core to host mode irrespective of the OTG_FS_ID input pin.

0: Normal mode

1: Force host mode

After setting the force bit, the application must wait at least 25 ms before the change takes effect.

主机的状态：给端口供电

- 给USB端口供电
  - 通过GPIO打开电源泵给插座上的Vbus引脚供电时，要置位PPWR@HPRT

USBH_Init() → HCD_Inint() →USB_OTG_CoreInitHost() → USB_OTG_DriveVbus(pdev, 1)

```
void USB_OTG_DriveVbus(USB_OTG_CORE_HANDLE *pdev , uint8_t state)
{ ……

USB_OTG_BSP_DriveVBUS(pdev, state);    //Set PH.5来使能charge pump的5V输出

hprt0.d32 = USB_OTG_ReadHPRT0(pdev);
if ((hprt0.b.prtpwr == 0 ) && (state == 1 ))
  { hprt0.b.prtpwr = 1;
    USB_OTG_WRITE_REG32(pdev->regs.HPRT0, hprt0.d32);}

if ((hprt0.b.prtpwr == 1 ) && (state == 0 ))
  { hprt0.b.prtpwr = 0;
    USB_OTG_WRITE_REG32(pdev->regs.HPRT0, hprt0.d32);  }
…… }
```

PPWR: Port power

The application uses this field to control power to this port, and the core clears this bit on an overcurrent condition.

0: Power off

1: Power on

主机的[redacted]：总线电平有效

- 有效的总线电压V_{BUS}
  - 插座上的Vbus pad可以连到OTG模块[redacted]引脚，使得主机能监控在USB通信过程中电源泵的输出是否保持在有效电压
    - 在电压意外掉下去时（低于4.25V）可
    - 中断标志：SEDET@GOTGINT
    - ISR必需关掉电源泵Vbus的输出，并复位端口供电位PPWR
  - 电压泵的过流可通过MCU上任意空闲GPIO来告知主机，并产生**相应EXTI中断**
    - ISR必需关掉Vbus的输出，并复位端口供电位PPWR

主机状态：设备的连接和断开

- 主机在检测到有效的总线电压时，才能检测设备的【连接与否】
  - PA.9和USB插座上的Vbus信号pad相连 & NOVBUSSENS = 0（复位值）时
    - 主机根据插槽上的Vbus pad上的电压， sense总线电压是否在有效范围内
  - PA.9没有和Vbus信号pad相连，用做其他功能 & NOVBUSSENS = 1时
    - 主机内部认为Vbus一直是有效的
- 【设备连接】
  - 硬件置位（检测到端口连接）PCDET@HPRT：软件需要写1清零
  - **触发中断**：（主机端口中断）HPRTINT@GINTSTS，表示FS端口有状态变化了，软件需要查看HPRT来确定发送了什么具体事件
- 【设备断开连接】
  - **触发中断**：（检测到断开连接）DISCINT@GINTSTS

主机如何检测到设备插入...

- HCD_IsDeviceConnected(pdev)
  - return (pdev->host.ConnSts)
- 何时该状态会被置位？ By interrupt...

@<stm32fxxx_it.c>

```
void OTG_FS_IRQHandler(void)
{
  USBH_OTG_ISR_Handler(&USB_OTG_Core);
}
```

```
……
 if (gintsts.b.portintr)
 {
   retval |= USB_OTG_USBH_handle_port_ISR (pdev);
 }
……
```

```
……
/* Port Connect Detected */
 if (hprt0.b.prtconndet)
 {

   hprt0_dup.b.prtconndet = 1;
   USBH_HCD_INT_fops->DevConnected(pdev);
   retval |= 1;
 }
 ……
```

```
uint8_t USBH_Connected (USB_OTG_CORE_HANDLE
*pdev)
{
  pdev->host.ConnSts = 1;
  return 0;
}
```

@ <usbh_core.c>

life.augmented

主机如何检测到设备拔除…

- HCD_IsDeviceConnected(pdev)
  - return (pdev->host.ConnSts)
- 何时该状态会被复位？By interrupt…

@<stm32fxxx_it.c>

```
void OTG_FS_IRQHandler(void)
{
  USBH_OTG_ISR_Handler(&USB_OTG_Core);
}
```

```
……
 if (gintsts.b.disconnect)
 {
   retval |= USB_OTG_USBH_handle_Disconnect_ISR (pdev);
 }
……
```

```
{
  ……
 USBH_HCD_INT_fops->DevDisconnected(pdev);

  /* Clear interrupt */
  gintsts.b.disconnect = 1;
  USB_OTG_WRITE_REG32(&pdev->regs.GREGS->GINTSTS, gintsts.d32);
}
```

```
uint8_t USBH_Disconnected (USB_OTG_CORE_HANDLE
*pdev)
{
  pdev->host.ConnSts = 0;
  return 0;
}
```

@ <usbh_core.c>

唤醒STOP模式下的USB主机

- STM32F105/107或者STM32F2/F4作为USB主机时，平时处于STOP模式……
- 如何能够通过设备的插入唤醒主机
  - 当FS/HS设备连上主机后，主机端的D+(PA11)会有个上升沿；可使能EXTI12的上升沿检测：一旦设备插入，EXTI12把MCU从STOP模式唤醒
  - 注意：EXTI不是AF管辖范围内，任何AF功能都可以附加上EXTI功能~~~

主机状态：枚举

- 设备连接并完成去抖阶段，**触发中断**表示总线再次稳定
  - DBCDNE@GOTGINT（只对主机模式有效）
  - 该位只在HNPCAP或SRPCAP@GUSBCFG置位时才有效
  - 如果类似USBH_MSC Demo作为“host only”模式，无需检查该信号
- SW发送USB复位信号
  - 置位PRST@HPRT并保持10ms~20ms，然后复位该位
- 复位序列完成后，**触发中断**表示主机可以读取设备速度了
  - PENCHNG@HPRT
    - 复位序列之后，内核会自动使能端口
    - 过流或断开连接时，内核自动禁止端口
  - SW可从PSPD@HPRT读取所连设备速度
  - 主机开始发送SOF或者Keep alives
  - 主机可以开始发送配置命令来完成对设备的枚举

USBH_Process() →

```
……
Case HOST_DEV_ATTACHED:
   if ( HCD_ResetPort(pdev) == 0)
   {  ……
      phost->device_prop.speed = HCD_GetCurrentSpeed(pdev);
      ……
   }
```

HCD_GetCurrentSpeed(*pdev)

```
USB_OTG_HPRT0_TypeDef  HPRT0;
HPRT0.d32 = USB_OTG_READ_REG32(pdev->regs.HPRT0);
return HPRT0.b.prtspd;
```

HCD_ResetPort(*pdev) → USB_OTG_ResetPort(*pdev)

```
hprt0.b.prtrst = 1;
USB_OTG_WRITE_REG32(pdev->regs.HPRT0, hprt0.d32);
USB_OTG_BSP_mDelay (10);
hprt0.b.prtrst = 0;
USB_OTG_WRITE_REG32(pdev->regs.HPRT0, hprt0.d32);
USB_OTG_BSP_mDelay (20);
```

USBH_HandleEnum() →

```
……
Case ENUM_IDLE:
   if (USBH_Get_DevDesc(pdev , phost, 8) == USBH_OK)
   {  ……
      HCD_ResetPort(pdev);              //复位USB设备
      ……
   }
```

主机状态：挂起

- 主机可以通过控制位来停止发送SOF，把总线挂起
  - 控制位：PSUSP@HPRT
- 总线的挂起状态可以通过【主机】发起退出
  - SW置位PRES@HPRT 来使得主机发送resume信号
  - SW需要掌控resume的时间，然后再复位PRES位
- 总线的挂起状态也可由【设备】发起而退出
  - 设备发出“远程唤醒信号”
  - 主机检测到后，触发WKUPINT@GINTSTS中断
  - 硬件自动置位PRES@HPRT来自动发送resume信号
  - 应用需要掌控resume的时间，然后手动复位PRES位

主机通道

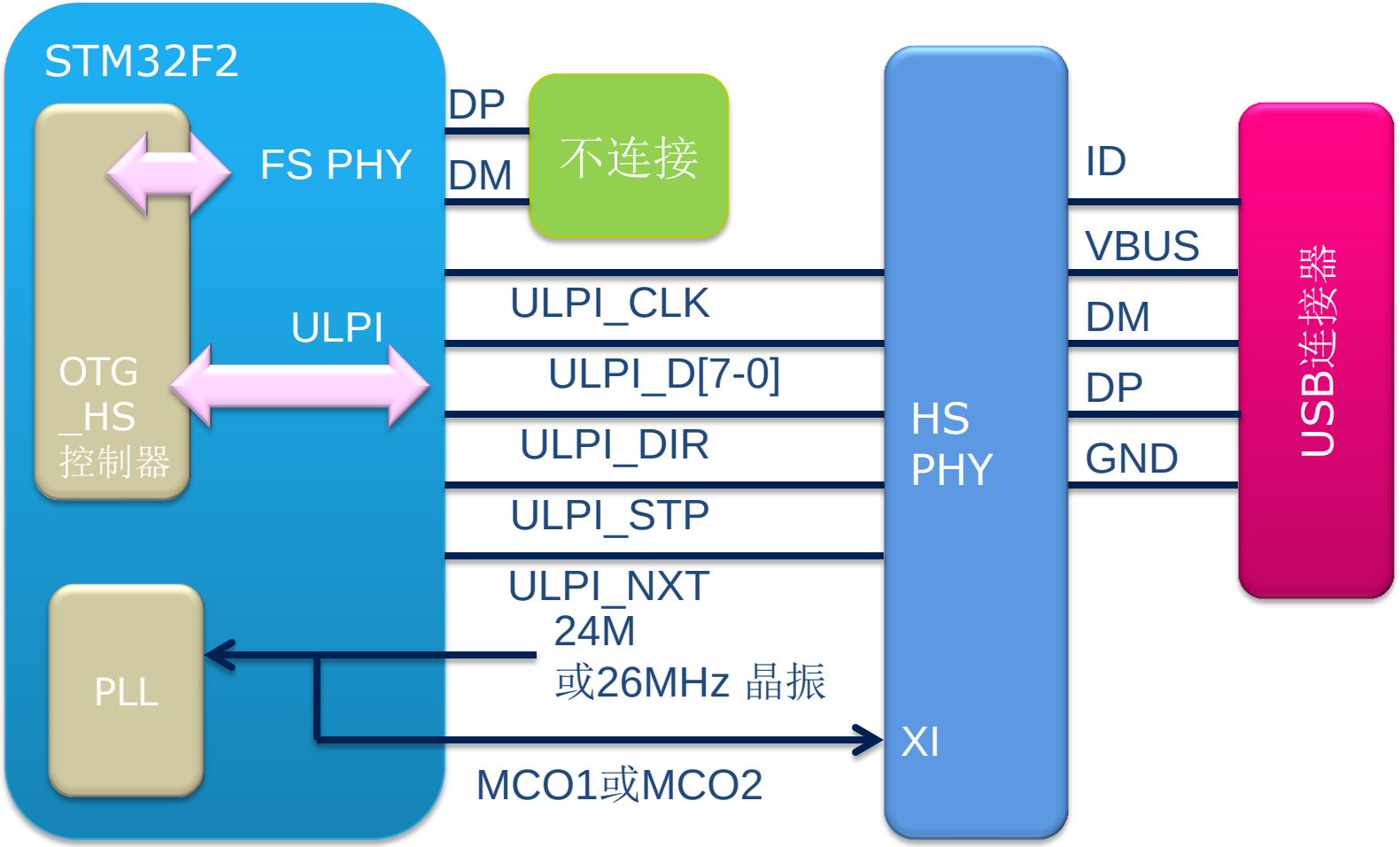
- OTG_FS模块有8个主机通道
  - 最多可同时处理来自应用的8个输出请求
  - 每个通道有各自的【控制】、【配置】、【状态/中断】、【掩码】寄存器
- 通道控制寄存器：HCCHARx
  - 通道enable/disable
  - 目标USB设备的速度FS/LS
  - 目标USB设备的地址
  - 目标USB设备目标端点号
  - 通道传输方向IN/OUT
  - 通道传输类型CTL/BLK/INT/ISO
  - 通道最大包长MPS
  - Periodic transfer to be executed during odd/even frames

- 通道传输配置寄存器：HCTSIZx
  - 应用在此配置传输参数，完成后才能enable该通道（前页）
  - 一旦传输开始，应用可在此读取传输状态
  - 配置此次传输字节数
  - 组成整个传输的数据包的个数
  - 初始的数据PID
- 通道状态/中断寄存器：HCINTx
  - 硬件首先置位HCINT@GINTSTS
  - 应用读取HCAINT来获知到底哪个通道的事件触发了中断
  - 然后应用再读取HCINTx才能确定具体发生了些什么事件

主机调度器

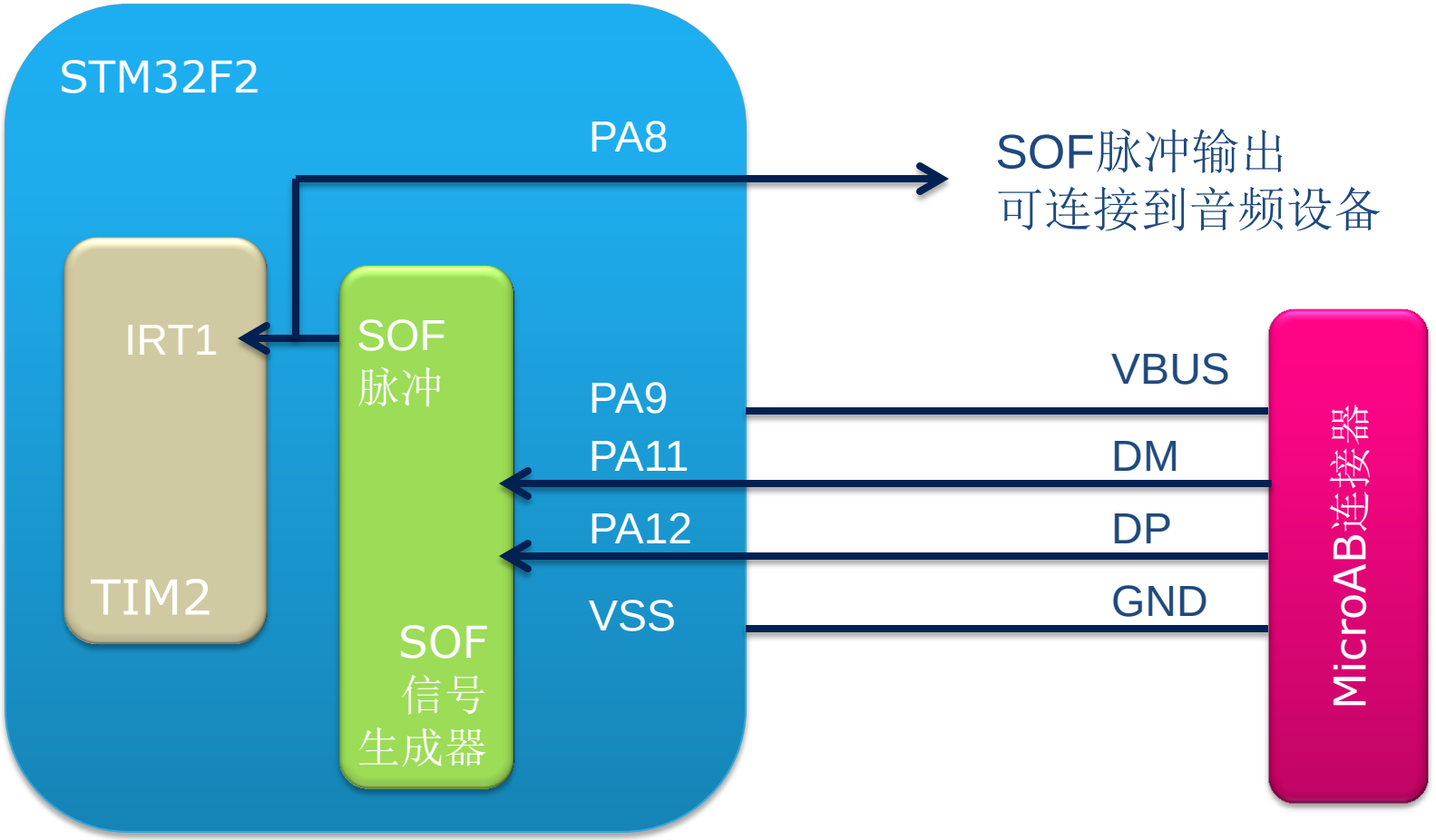
- 主机内核集成一个硬件调度器，对来自应用的USB传输请求进行自动排序和管理
  - 每个frame中，先处理周期传输（ISO、INT）；在处理非周期传输（CTL、BULK）
  - 调度器通过请求队列处理USB传输请求
    - 一个周期传输请求队列
    - 一个非周期传输请求队列
    - 每个队列可包含8个entry，对应每个请求所对应的通道号码，和传输相应参数
  - 若当前frame结束之前host还没有完成计划在当前frame处理的周期，则触发中断IPXFR@GINTSTS
  - 应用读取每个请求队列的状态，只读寄存器
    - 周期传输发送FIFO和队列状态寄存器：HPTXSTS
    - 非周期传输发送FIFO和队列状态寄存器：HNPTXSTS
  - 应用需要在发出传输请求之前先检查队列有空余entry
    - 通过读取PTXQSAV@HPTXSTS
    - NPTXQXSAV @HNPTXSTS

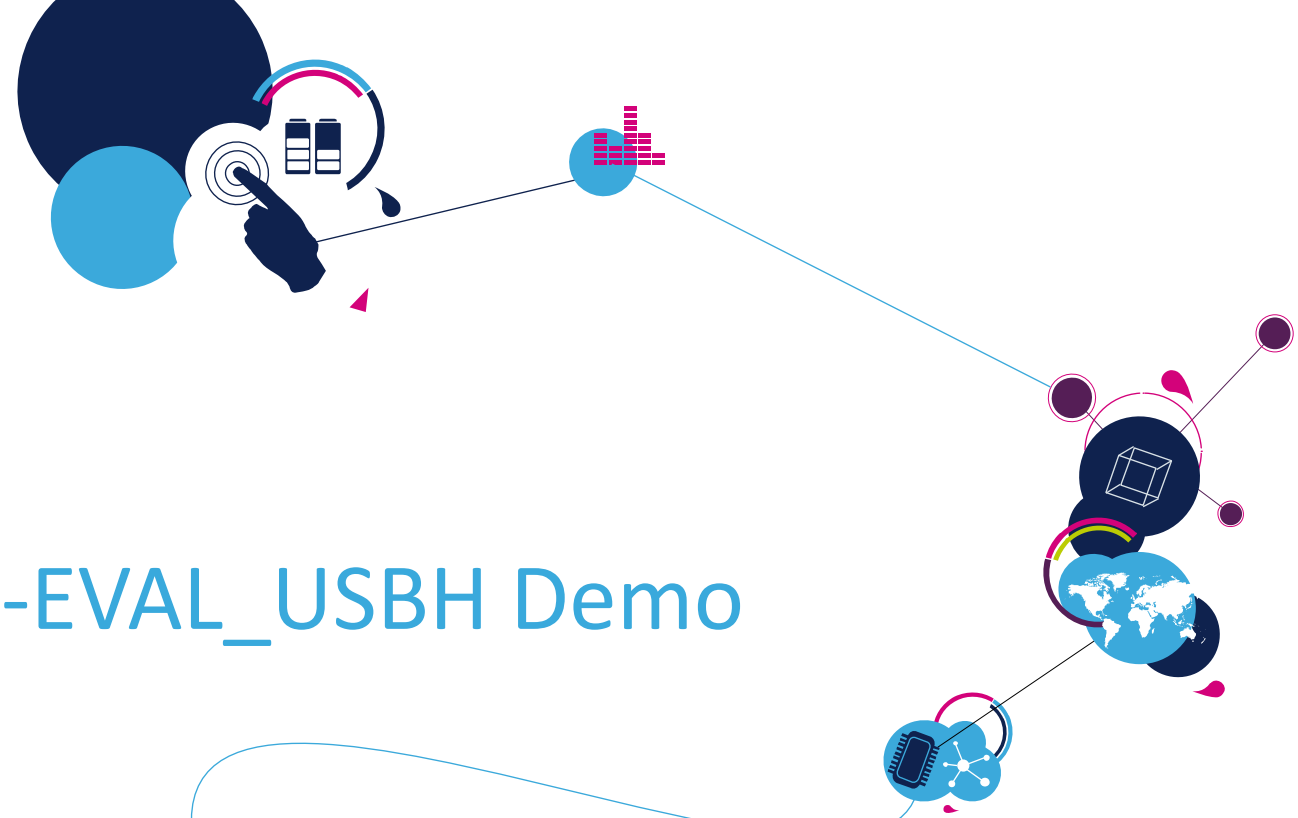
OTG_HS模块外接HS PHY



- 使用MCO1或MCO2输出时钟至外接PHY，可节省外接晶振

STM32 OTG_FS模块特性 —— SOF信号连接



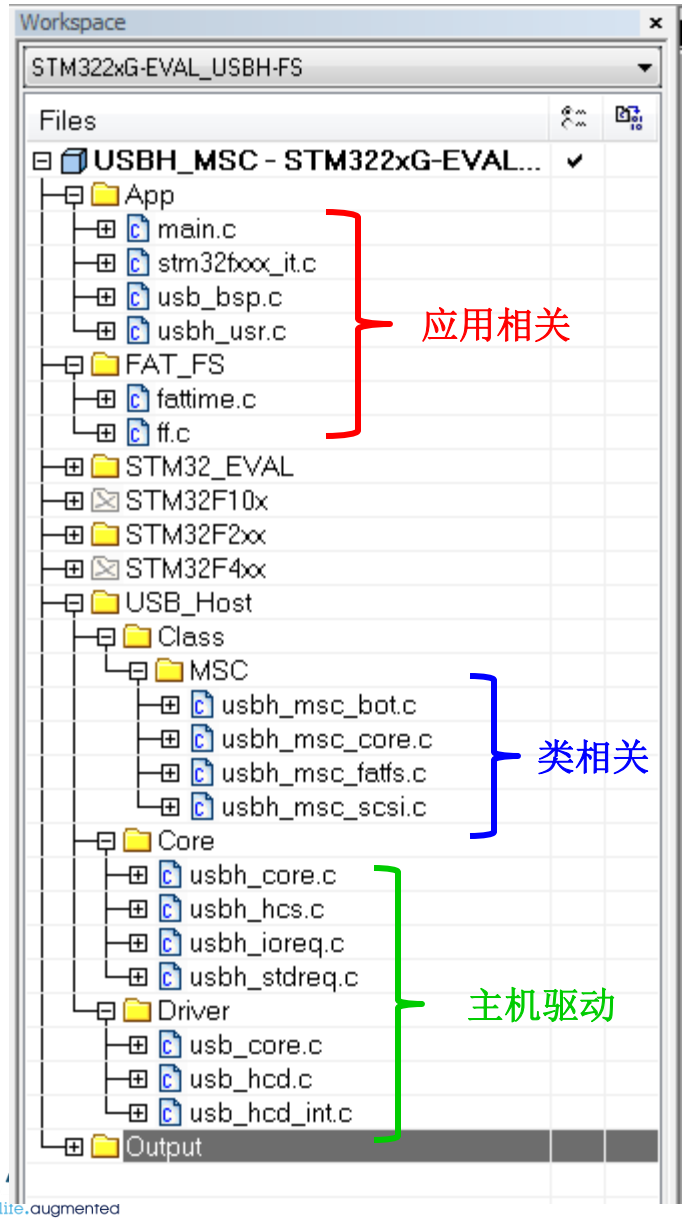


STM322xG-EVAL_USBH Demo

Lilian YAO

life.augmented

USBH-MSC：接U盘



应用相关文件：

<usb_bsp.c> 板子相关

>> GPIO连接：DM、DP、Vbus、SOF…

>> 系统中断NVIC配置和使能

>> 系统用到的延时功能

<usbh_usr.c> 应用相关

>> 各种用户回调函数

>> 应用相关的初始化、实现等

MSC类相关文件：

<usbh_msc_bot.c> 处理BOT流程

<usbh_msc_core.c> MSC类特有命令和request处理

<usbh_msc_fatfs.c> 存储介质访问

<usbh_msc_scsi.c> 各种SCSI命令处理

```
typedef struct _USBH_Class_cb
{
  (*Init) (*pdev , void *phost);
  (*DeInit) (*pdev , void *phost);
  (*Requests) (*pdev , void *phost);
  (*Machine) (*pdev , void *phost);
} USBH_Class_cb_TypeDef;
```

```
USBH_Class_cb_TypeDef  USBH_MSC_cb =
{
  USBH_MSC_InterfaceInit,
  USBH_MSC_InterfaceDeInit,
  USBH_MSC_ClassRequest,
  USBH_MSC_Handle,
};
```

```
typedef struct _USBH_USR_PROP
{
  void (*Init)(void);
  void (*DeInit)(void);
  void (*DeviceAttached)(void);
  void (*ResetDevice)(void);
  void (*DeviceDisconnected)(void);
  void (*OverCurrentDetected)(void);
  void (*DeviceSpeedDetected)(DeviceSpeed);
  void (*DeviceDescAvailable)(void *);
  void (*DeviceAddressAssigned)(void);
  void (*ConfigurationDescAvailable
  void (*ManufacturerString)(void)
  void (*ProductString)(void *);
  void (*SerialNumString)(void *);
  void (*EnumerationDone)(void);
  USBH_USR_Status (*UserInput)(void);
  int  (*UserApplication) (void);
  void (*DeviceNotSupported)(void);
  void (*UnrecoveredError)(void);
} USBH_Usr_cb_TypeDef;
```

```
USBH_Usr_cb_TypeDef USR_cb =
{
  USBH_USR_Init,
  USBH_USR_DeInit,
  USBH_USR_DeviceAttached,
  USBH_USR_ResetDevice,
  USBH_USR_DeviceDisconnected,
  USBH_USR_OverCurrentDetected,
  USBH_USR_DeviceSpeedDetected,
  USBH_USR_Device_DescAvailable,
  USBH_USR_DeviceAddressAssigned,
  USBH_USR_Configuration_DescAvailable,
  USBH_USR_Manufacturer_String,
  USBH_USR_Product_String,
  USBH_USR_SerialNum_String,
  USBH_USR_EnumerationDone,
  USBH_USR_UserInput,
  USBH_USR_MSC_Application,
  USBH_USR_DeviceNotSupported,
  USBH_USR_UnrecoveredError
};
```

用户应用程序只需调用…

- 使用高速OTG模块实现USB主机（MSC类主机）
  - USBH_Init(&USB_OTG_Core,
  - USB_OTG_HS_CORE_ID
  - &USB_Host,
  - &USBH_MSC_cb,
  - &USR_cb);

初始化

```
USBH_Class_cb_TypeDef  USBH_MSC_cb =
{
  USBH_MSC_InterfaceInit,
  USBH_MSC_InterfaceDeInit,
  USBH_MSC_ClassRequest,
  USBH_MSC_Handle,
};
```

MSC类的回调函数

```
typedef enum
{
  USB_OTG_HS_CORE_ID = 0,
  USB_OTG_FS_CORE_ID = 1
}USB_OTG_CORE_ID_TypeDef;
```

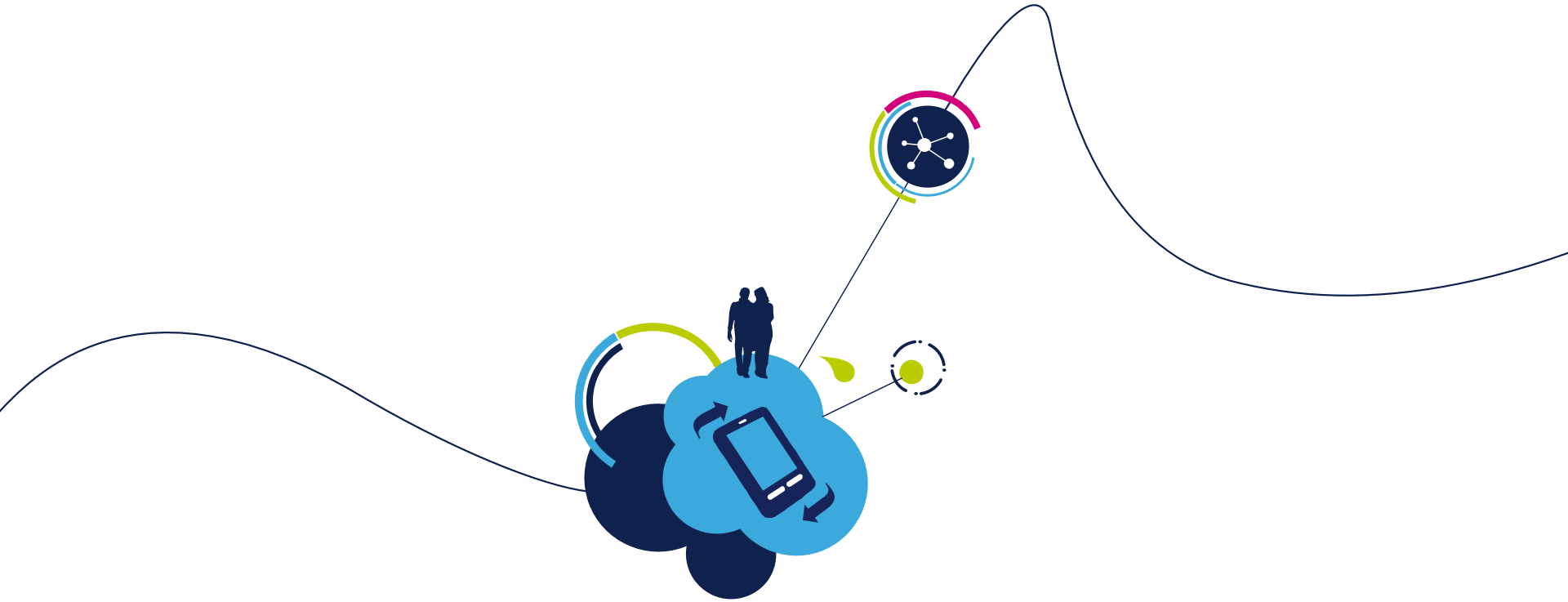
用户回调函数

```
USBH_Usr_cb_TypeDef USR_cb =
{
  USBH_USR_Init,
  USBH_USR_DeInit,
  USBH_USR_DeviceAttached,
  USBH_USR_ResetDevice,
  USBH_USR_DeviceDisconnected,
  USBH_USR_OverCurrentDetected,
  USBH_USR_DeviceSpeedDetected,
  USBH_USR_Device_DescAvailable,
  USBH_USR_DeviceAddressAssigned,
  USBH_USR_Configuration_DescAvailable,
  USBH_USR_Manufacturer_String,
  USBH_USR_Product_String,
  USBH_USR_SerialNum_String,
  USBH_USR_EnumerationDone,
  USBH_USR_UserInput,
  USBH_USR_MSC_Application,
  USBH_USR_DeviceNotSupported,
  USBH_USR_UnrecoveredError

};
```

- While(1) {
- USBH_Process(&USB_OTG_Core, &USB_Host);
- }

运行

life.augmented



初始化.参数结构

初始化

USB_OTG_Core的定义

- 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;
- #endif
- #ifdef USE_OTG_MODE
- OTG_DEV otg;
- #endif
- }
- USB_OTG_CORE_HANDLE , *PUSB_OTG_CORE_HANDLE;
- USB_OTG_CORE_HANDLE USB_OTG_Core

```
typedef struct _HCD
{
  uint8_t              Rx_Buffer [MAX_DATA_LENGTH];
  __IO uint32_t        ConnSts;
  __IO uint32_t        ErrCnt[USB_OTG_MAX_TX_FIFOS];
  __IO uint32_t        XferCnt[USB_OTG_MAX_TX_FIFOS];
  __IO HC_STATUS       HC_Status[USB_OTG_MAX_TX_FIFOS];
  __IO URB_STATE       URB_State[USB_OTG_MAX_TX_FIFOS];
  USB_OTG_HC           hc [USB_OTG_MAX_TX_FIFOS];
  uint16_t             channel [USB_OTG_MAX_TX_FIFOS];
}
HCD_DEV , *USB_OTG_USBH_PDEV;
```

- <main.c> while(1) { USBH_Process(&**USB_OTG_Core**, &USB_Host);…}
- void USBH_Process(USB_OTG_CORE_HANDLE ***pdev*** , USBH_HOST *phost)

life.augmented

USBH_HOST的定义

```
typedef struct _Host_TypeDef
{
  HOST_State                gState;       /* USBH_Process大循环的switch */
  HOST_State                gStateBkp;
  ENUM_State                EnumState;    /* 用于USBH_HandleEnum循环 */
  CMD_State                 RequestState; /* USBH_CtlReq循环 */
  USBH_Ctrl_TypeDef         Control;
  USBH_Device_TypeDef       device_prop;
  USBH_Class_cb_TypeDef     *class_cb;
  USBH_Usr_cb_TypeDef       *usr_cb;
} USBH_HOST, *pUSBH_HOST;
USBH_HOST    USB_Host
```

MSC、HID类的回调函数

用户回调函数

- <main.c> while(1) { USBH_Process(&USB_OTG_Core, &**USB_Host**);…}
- void USBH_Process(USB_OTG_CORE_HANDLE *pdev , USBH_HOST ***phost**)

```
typedef enum {

  HOST_IDLE =0,
  HOST_DEV_ATTACHED,
  HOST_DEV_DISCONNECTED,
  HOST_DETECT_DEVICE_SPEED,
  HOST_ENUMERATION,
  HOST_CLASS_REQUEST,
  HOST_CLASS,
  HOST_CTRL_XFER,
  HOST_USR_INPUT,
  HOST_SUSPENDED,
  HOST_ERROR_STATE

} HOST_State;
HOST_State          gState, gStateBkp;
```

```
typedef enum {

 ENUM_IDLE = 0,
 ENUM_GET_FULL_DEV_DESC,
 ENUM_SET_ADDR,
 ENUM_GET_CFG_DESC,
 ENUM_GET_FULL_CFG_DESC,
 ENUM_GET_MFC_STRING_DESC,
 ENUM_GET_PRODUCT_STRING_DESC,
 ENUM_GET_SERIALNUM_STRING_DESC,
 ENUM_SET_CONFIGURATION,
 ENUM_DEV_CONFIGURED

} ENUM_State;
ENUM_State          EnumState;
```

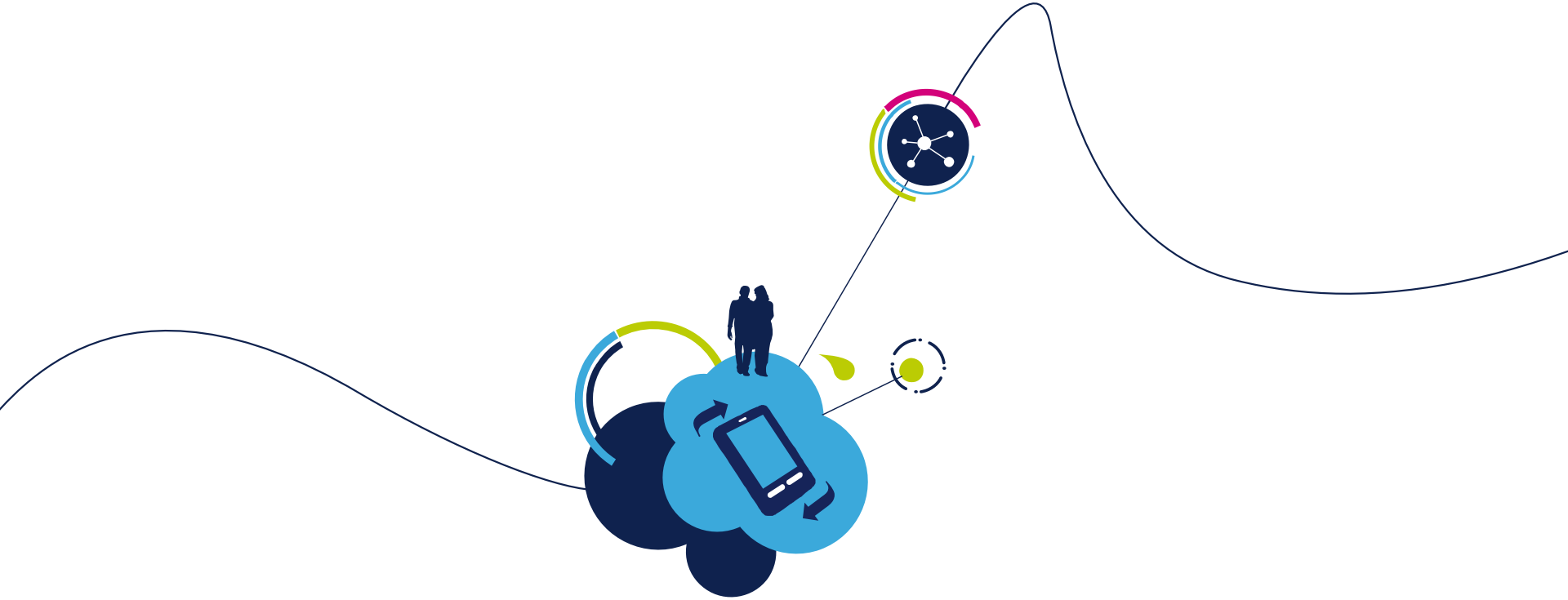
```
typedef enum {

 CMD_IDLE =0,
 CMD_SEND,
 CMD_WAIT

} CMD_State;
CMD_State          RequestState;
```

```
typedef struct _Ctrl{

  uint8_t                 hc_num_in;
  uint8_t                 hc_num_out;
  uint8_t                 ep0size;
  uint8_t                 *buff;
  uint16_t                length;
  uint8_t                 errorcount;
  uint16_t                timer;
  CTRL_STATUS             status;
  USB_Setup_TypeDef       setup;
  CTRL_State              state;

} USBH_Ctrl_TypeDef;

USBH_Ctrl_TypeDef     Control;
```

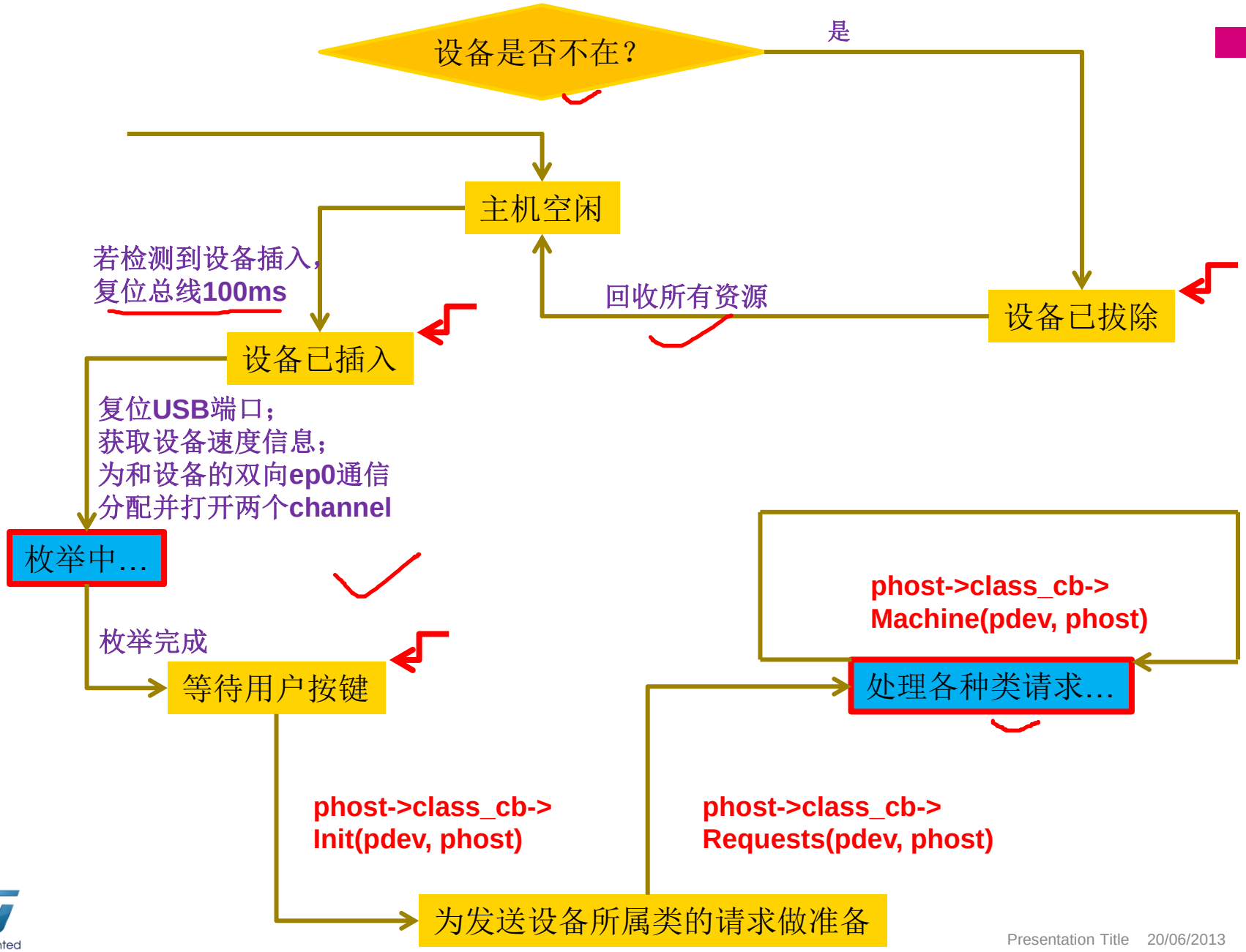
对插上来的设备的属性的记录

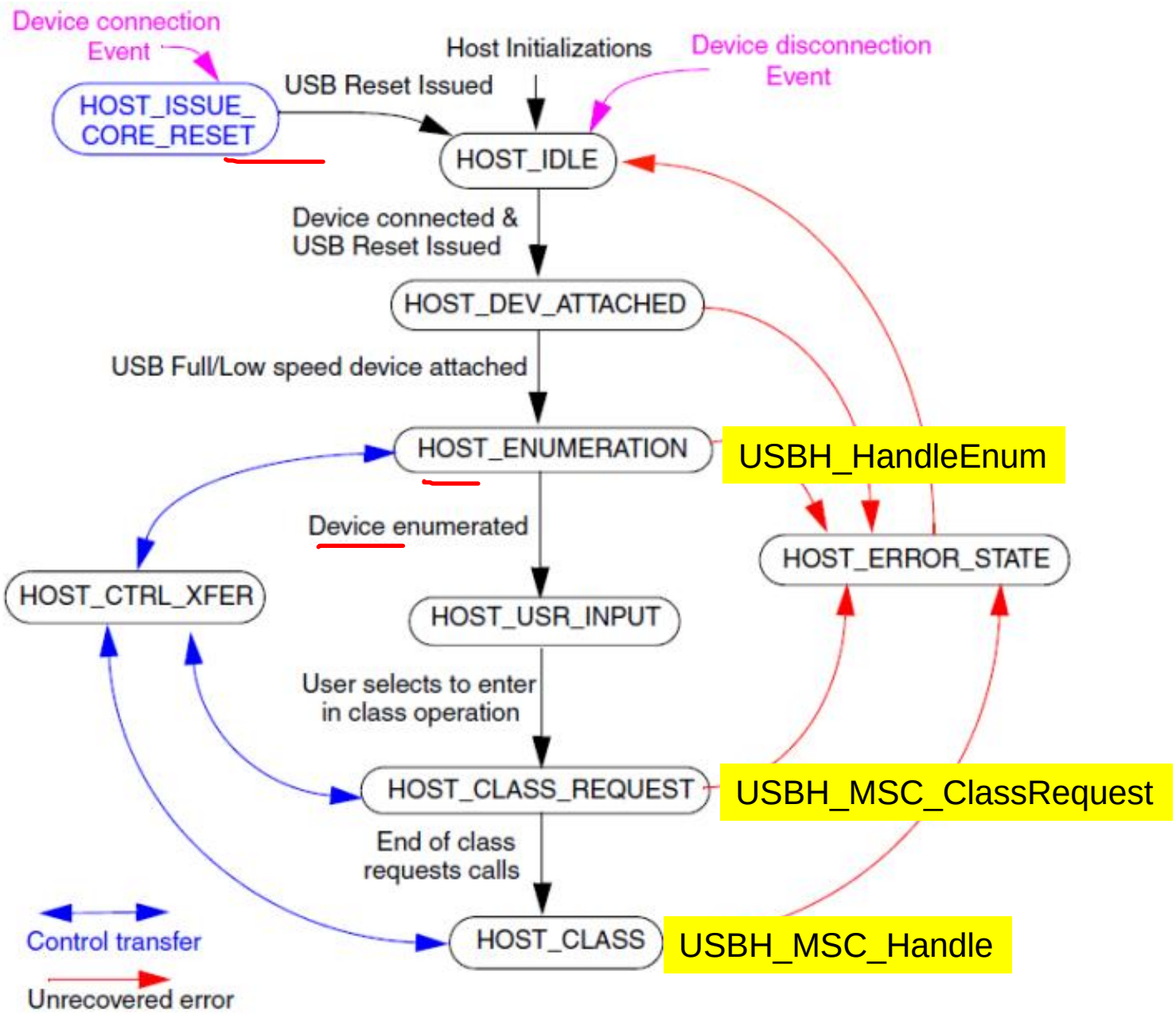
- 设备插上后检测到其速度：LS/FS/HS
- 枚举过程中USB主机给设备分配的设备地址
- 从枚举过程中获得关于设备各种信息
  - 设备、配置、接口、端点描述子(Descriptor)

```
typedef struct _DeviceProp {

  uint8_t                          address;
  uint8_t                          speed;
  USBH_DevDesc_TypeDef             Dev_Desc;
  USBH_CfgDesc_TypeDef             Cfg_Desc;
  USBH_InterfaceDesc_TypeDef       Itf_Desc[USBH_MAX_NUM_INTERFACES];
 USBH_EpDesc_TypeDef               Ep_Desc[USBH_MAX_NUM_INTERFACES][USBH_MAX_NUM_ENDPOINTS];
 USBH_HIDDesc_TypeDef              HID_Desc;

}USBH_Device_TypeDef;

USBH_Device_TypeDef   device_prop;
```



开始运行.状态机


设备是否不在？
是
主机空闲
若检测到设备插入，
复位总线100ms
回收所有资源
设备已拔除
设备已插入
复位USB端口；
获取设备速度信息；
为和设备的双向ep0通信
分配并打开两个channel
枚举中…
枚举完成
等待用户按键
phost->class_cb->
Machine(pdev, phost)
处理各种类请求…
phost->class_cb->
Init(pdev, phost)
phost->class_cb->
Requests(pdev, phost)
为发送设备所属类的请求做准备


Device connection Event
Host Initializations
Device disconnection Event
USB Reset Issued
HOST_ISSUE_ CORE_RESET
HOST_IDLE
Device connected & USB Reset Issued
HOST_DEV_ATTACHED
USB Full/Low speed device attached
HOST_ENUMERATION
USBH_HandleEnum
Device enumerated
HOST_ERROR_STATE
HOST_CTRL_XFER
HOST_USR_INPUT
User selects to enter in class operation
HOST_CLASS_REQUEST
USBH_MSC_ClassRequest
End of class requests calls
HOST_CLASS
USBH_MSC_Handle
Control transfer
Unrecovered error

USBH_Process

	MSC
主机空闲	若检测到设备插入，复位总线100ms
设备已插入	复位USB端口；获取设备速度信息； 为和设备的双向ep0通信分配并打开两个channel
枚举中…	USBH_HandleEnum(pdev , phost) →ENUM_IDLE、ENUM_GET_FULL_DEV_DESC、ENUM_SET_ADDR、ENUM_GET_CFG_DESC、ENUM_GET_FULL_CFG_DESC、ENUM_GET_MFC_STRING_DESC、ENUM_GET_PRODUCT_STRING_DESC、ENUM_GET_SERIALNUM_STRING_DESC、ENUM_SET_CONFIGURATION、ENUM_DEV_CONFIGURED
等待用户按键	phost->class_cb->Init(pdev, phost) @ <usbh_xxx_core.c>
	即 USBH_MSC_InterfaceInit 为和设备进行通信分配并打开两个channel
为发送设备所属类的请求做准备	phost->class_cb->Requests(pdev, phost)
	即 USBH_MSC_ClassRequest
	为处理MSC类的BOT通信过程初始化BOT状态机的初始状态

USBH_Process(2)

	MSC
处理各种类请求	**phost->class_cb->Machine(pdev, phost)**
	即 **USBH_MSC_Handle** @<usbh_msc_core.c>
	处理**MSC**类的**BOT**通信过程 Switch (USBH_MSC_BOTXferParam.MSCState) USBH_MSC_BOT_INIT_STATE、USBH_MSC_BOT_RESET、USBH_MSC_GET_MAX_LUN、 USBH_MSC_CTRL_ERROR_STATE、USBH_MSC_TEST_UNIT_READY、 USBH_MSC_READ_CAPACITY10、USBH_MSC_MODE_SENSE6、 USBH_MSC_REQUEST_SENSE、USBH_MSC_BOT_USB_TRANSFERS、 USBH_MSC_DEFAULT_APPLI_STATE、 → **pphost->usr_cb->UserApplication()** **即USBH_USR_MSC_Application()** USBH_MSC_UNRECOVERED_STATE

各状态下的用户/类相关回调函数

- 主机空闲：HOST_IDLE
  - phost->**usr_cb**->DeviceAttached()
- 设备已插入：HOST_DEV_ATTACHED
  - phost->**usr_cb**->DeviceAttached()
  - phost->**usr_cb**->ResetDevice()
  - phost->**usr_cb**->DeviceSpeedDetected(phost->device_prop.speed)
- 枚举中：HOST_ENUMERATION
  - phost->**usr_cb**->EnumerationDone()
- ……

各状态下的用户/类相关回调函数

- 等待用户输入：HOST_USR_INPUT
  - phost->**usr_cb**->UserInput()
  - phost->**class_cb**->Init(pdev, phost)
- 为发送设备所属类的请求做准备：HOST_CLASS_REQUEST
  - phost->**class_cb**->Requests(pdev, phost)
- 处理各种类请求：HOST_CLASS
  - phost->**class_cb**->Machine(pdev, phost)
- 设备已拔除：HOST_DEV_DISCONNECTED
  - phost->**usr_cb**->DeviceDisconnected()
  - phost->**usr_cb**->DeInit()
  - phost->**class_cb**->DeInit(pdev, &phost->device_prop)

主机如何检测到设备插入...

- HCD_IsDeviceConnected(pdev)
  - return (pdev->host.ConnSts)
  - 如果检测到插入，状态从**HOST_IDEL → HOST_DEV_ATTACHED**
    - phost->gState = HOST_DEV_ATTACHED
- 何时该状态会被置位？ By interrupt...

@<stm32fxxx_it.c>

```
void OTG_FS_IRQHandler(void)
{
  USBH_OTG_ISR_Handler(&USB_OTG_Core);
}
```

```
……
 if (gintsts.b.portintr)
 {
   retval |= USB_OTG_USBH_handle_port_ISR (pdev);
 }
……
```

```
……
/* Port Connect Detected */
 if (hprt0.b.prtconndet)
 {

   hprt0_dup.b.prtconndet = 1;
   USBH_HCD_INT_fops->DevConnected(pdev);
   retval |= 1;
 }
 ……
```

主机如何检测到设备拔除…

- HCD_IsDeviceConnected(pdev)
  - return (pdev->host.ConnSts)
  - 如果检测到拔除，状态从**XXX → HOST_DEV_DISCONNECTED**
    - phost->gState = HOST_DEV_DISCONNECTED
- 何时该状态会被复位？By interrupt…

@<stm32fxxx_it.c>

```
void OTG_FS_IRQHandler(void)
{
  USBH_OTG_ISR_Handler(&USB_OTG_Core);
}
```

```
……
 if (gintsts.b.disconnect)
  {
    retval |= USB_OTG_USBH_handle_Disconnect_ISR (pdev);
  }
……
```

```
{
  ……
 USBH_HCD_INT_fops->DevDisconnected(pdev);

  /* Clear interrupt */
  gintsts.b.disconnect = 1;
  USB_OTG_WRITE_REG32(&pdev->regs.GREGS->GINTSTS, gintsts.d32);
}
```

主机检测到设备后...

- 为设备的双向0端点各分配一个channel，并打开

```
phost->Control.hc_num_out = USBH_Alloc_Channel(pdev, 0x00);      <0>
phost->Control.hc_num_in = USBH_Alloc_Channel(pdev, 0x80);       <1>

USBH_Open_Channel (pdev,
                   phost->Control.hc_num_in,                     <1，刚分配的>
                   phost->device_prop.address,                   <默认值：0>
                   phost->device_prop.speed,                     <已检测到>
                   EP_TYPE_CTRL,                                 <固定=控制传输类型>
                   phost->Control.ep0size);                      <默认值：64>

USBH_Open_Channel (pdev,
                   phost->Control.hc_num_out,                     <0，刚分配的>
                   phost->device_prop.address,                    <默认值：0>
                   phost->device_prop.speed,                      <已检测到>
                   EP_TYPE_CTRL,                                  <固定=控制传输类型>
                   phost->Control.ep0size);                       <默认值：64>
```

- 复位该USB端口长达10ms

- 获取插入设备的速度（如何获知？中断ISR中读取寄存器）
  - 读取状态寄存器 pdev->regs.HPRT0，其中有2个bit域表示速度信息，由硬件设置

```
phost->device_prop.speed = HCD_GetCurrentSpeed(pdev);
```

life.augmented

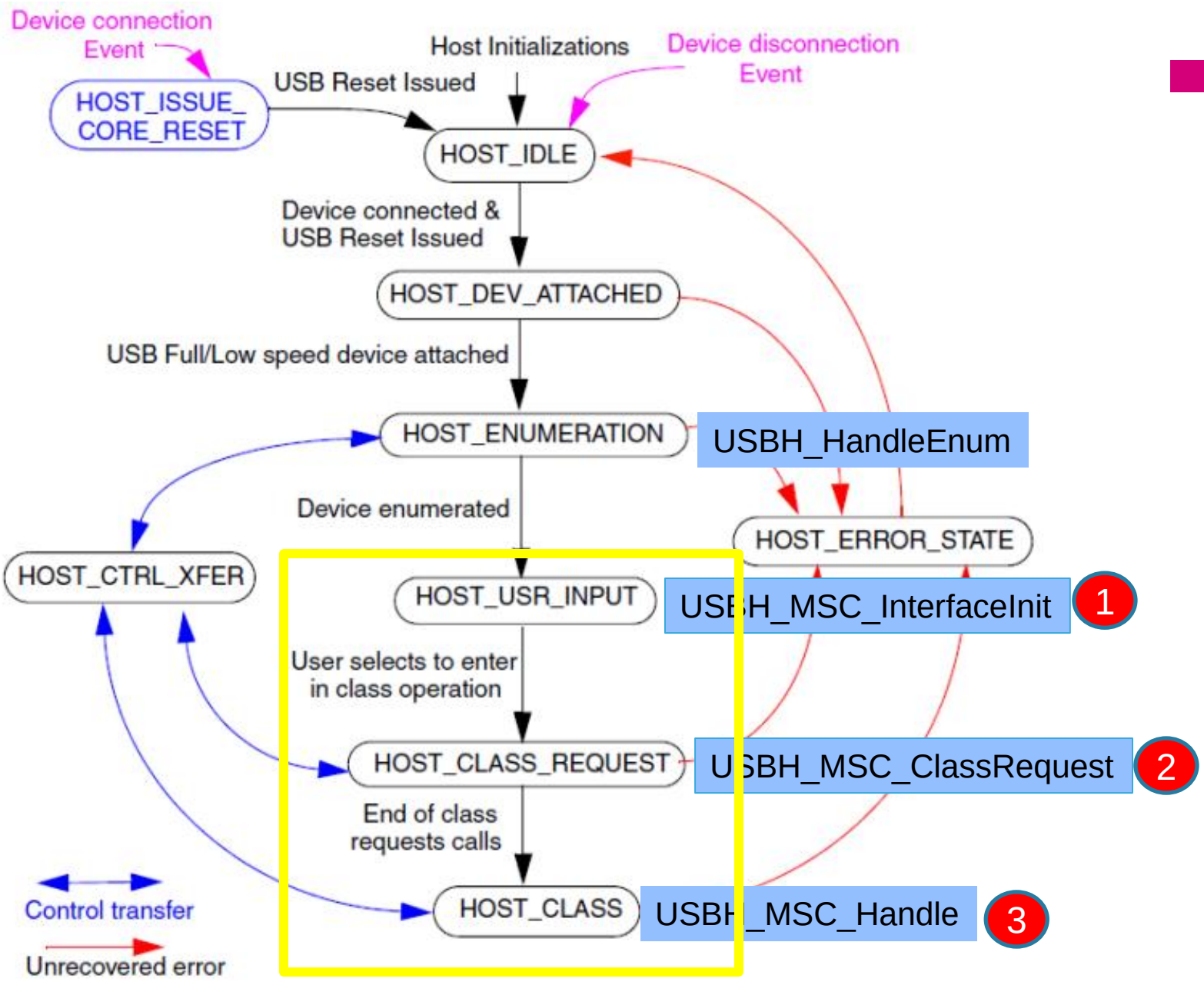
枚举过程中...

枚举结束

- 主机为设备分配地址：0x01
- 主机获知设备的各种信息（Descriptor）
  - ep0支持的最大包长度：64
  - 非自供电设备，不支持远程唤醒 (remote control)；最大功耗100mA
  - 属于MSC类，支持SCSI命令集、使用BOT协议
  - 设备使用2个端点 (ep)
    - Ep1，IN， 支持最大包长度512字节，传输类型= BULK
    - Ep2，OUT，支持最大包长度512字节，传输类型= BULK
- 等待用户按键，以继续…（进入“发送类相关request”阶段）

发送类相关Request


```
case HOST_USR_INPUT:

if ( phost->usr_cb->UserInput() == OK)
   {
     if ( (phost->class_cb->Init (pdev, phost)) == OK)
     { phost->gState  = HOST_CLASS_REQUEST;  }
   }

case HOST_CLASS_REQUEST:
    status = phost->class_cb->Requests (pdev, phost);

    if(status == USBH_OK)
    { phost->gState  = HOST_CLASS; }

    else
    { USBH_ErrorHandle(phost, status); }

 case HOST_CLASS:

   status = phost->class_cb->Machine (pdev, phost);
   USBH_ErrorHandle(phost, status);
   break;
```


Device connection Event
Host Initializations
Device disconnection Event
USB Reset Issued
HOST_ISSUE_ CORE_RESET
HOST_IDLE
Device connected & USB Reset Issued
HOST_DEV_ATTACHED
USB Full/Low speed device attached
HOST_ENUMERATION
USBH_HandleEnum
Device enumerated
HOST_ERROR_STATE
HOST_CTRL_XFER
HOST_USR_INPUT
USBH_MSC_InterfaceInit
1
User selects to enter in class operation
HOST_CLASS_REQUEST
USBH_MSC_ClassRequest
2
End of class requests calls
HOST_CLASS
USBH_MSC_Handle
3
Control transfer
Unrecovered error

MSC：为后续发起类相关传输做准备

phost->class_cb->Init (pdev, phost) 1

```
USBH_MSC_InterfaceInit ( USB_OTG_CORE_HANDLE *pdev,  void *phost)
{
  ……

   MSC_Machine.hc_num_out  =  USBH_Alloc_Channel (pdev,  0x02);          <2>
   MSC_Machine.hc_num_in   =  USBH_Alloc_Channel (pdev,  0x81);          <3>

   USBH_Open_Channel (pdev,
                      MSC_Machine.hc_num_out,                            <2，刚分配的>
                      pphost->device_prop.address                        <枚举过程中分配：1>
                      pphost->device_prop.speed,                         <已检测到>
                     EP_TYPE_BULK,                                       <固定=BULK传输类型>
                      MSC_Machine.MSBulkOutEpSize);                      <从枚举获知：512>

   USBH_Open_Channel (pdev,
                      MSC_Machine.hc_num_in,                             <3，刚分配的>
                      pphost->device_prop.address,                       <枚举过程中分配：1>
                      pphost->device_prop.speed,                         <已检测到>
                     EP_TYPE_BULK,                                        <固定=BULK传输类型>
                     MSC_Machine.MSBulkInEpSize);                         <从枚举获知：512 >
  ……
}
```

phost->class_cb->Requests(pdev, phost) 2

```
USBH_Status USBH_MSC_ClassRequest (USB_OTG_CORE_HANDLE *pdev , void *phost)
{
  USBH_MSC_BOTXferParam.MSCState = USBH_MSC_BOT_INIT_STATE;
}
```

MSC：发送类相关Request

phost->class_cb->Machine (pdev, phost) 3

USBH_MSC_Handle (USB_OTG_CORE_HANDLE *pdev, void *phost)

```
switch(USBH_MSC_BOTXferParam.MSCState)
  {
  case USBH_MSC_BOT_INIT_STATE:
    USBH_MSC_Init(pdev);
    USBH_MSC_BOTXferParam.MSCState = USBH_MSC_BOT_RESET;
    break;

  case USBH_MSC_BOT_RESET:
    status = USBH_MSC_BOTReset(pdev, phost);

    if(status == USBH_OK )
      USBH_MSC_BOTXferParam.MSCState = USBH_MSC_GET_MAX_LUN;

    if(status == USBH_NOT_SUPPORTED )
    {
      USBH_MSC_BOTXferParam.MSCStateBkp = USBH_MSC_GET_MAX_LUN;
      USBH_MSC_BOTXferParam.MSCState = USBH_MSC_CTRL_ERROR_STATE;
    }
    break;

  ..............................
```

填充CBW结构体的一部分

如果该命令不支持，在主机clear feature后，继续走下一个命令

MSC：发送类相关Request（2）

phost->class_cb->Machine (pdev, phost)

USBH_MSC_Handle (USB_OTG_CORE_HANDLE *pdev, void *phost)

```
switch(USBH_MSC_BOTXferParam.MSCState)
  {
  .....................
  case USBH_MSC_GET_MAX_LUN:
    status = USBH_MSC_GETMaxLUN(pdev, phost);

    if(status == USBH_OK )
    {
      MSC_Machine.maxLun = *(MSC_Machine.buff) ;

      if((MSC_Machine.maxLun > 0) && (maxLunExceed == FALSE))
      {
        maxLunExceed = TRUE; pphost->usr_cb->DeviceNotSupported(); break;
      }
      USBH_MSC_BOTXferParam.MSCState = USBH_MSC_TEST_UNIT_READY;
    }

    if(status == USBH_NOT_SUPPORTED )
    {
      USBH_MSC_BOTXferParam.MSCStateBkp = USBH_MSC_TEST_UNIT_READY;
      USBH_MSC_BOTXferParam.MSCState = USBH_MSC_CTRL_ERROR_STATE;
    }
    break;

  case USBH_MSC_CTRL_ERROR_STATE:
    status = USBH_ClrFeature(pdev, phost, 0x00, pphost->Control.hc_num_out);

    if(status == USBH_OK )
    {
      MSC_Machine.maxLun = 0;
      USBH_MSC_BOTXferParam.MSCState = USBH_MSC_BOTXferParam.MSCStateBkp;
    }
    break;
```

填充Setup包 + USBH_CtlReq(pdev, phost, MSC_Machine.buff, 1)

本主机demo不支持多盘符的U盘

如果该命令不支持，在主机clear feature后，继续走下一个命令

在此发送clear feature标准命令

“有些只有一个盘符的U盘，是可以对GetMaxLUN命令STALL的”

执行完clear feature后就直接走下一个命令了

用户应用：读取文件

- 基于开源的嵌入式文件系统FatFs进行文件操作

pphost->usr_cb->UserApplication()

```
int USBH_USR_MSC_Application(void)
{
  USBH_USR_ApplicationState

  = USH_USR_FS_INIT：          f_mount( 0, &fatfs )

  = USH_USR_FS_READLIST：      Explore_Disk("0:/", 1)
                               f_opendir(&dir, path);
                               f_readdir(&dir, &fno);

  = USH_USR_FS_WRITEFILE：     f_mount(0, &fatfs )
                               f_open(&file, "0:STM32.TXT",FA_CREATE_ALWAYS | FA_WRITE)
                               f_write (&file, writeTextBuff, bytesToWrite, (void *)&bytesWritten);
                               ……
                               f_close(&file)
                               f_mount(0, NULL)

  = USH_USR_FS_DRAW：          f_mount( 0, &fatfs )
                               Image_Browser("0:/")

}
```

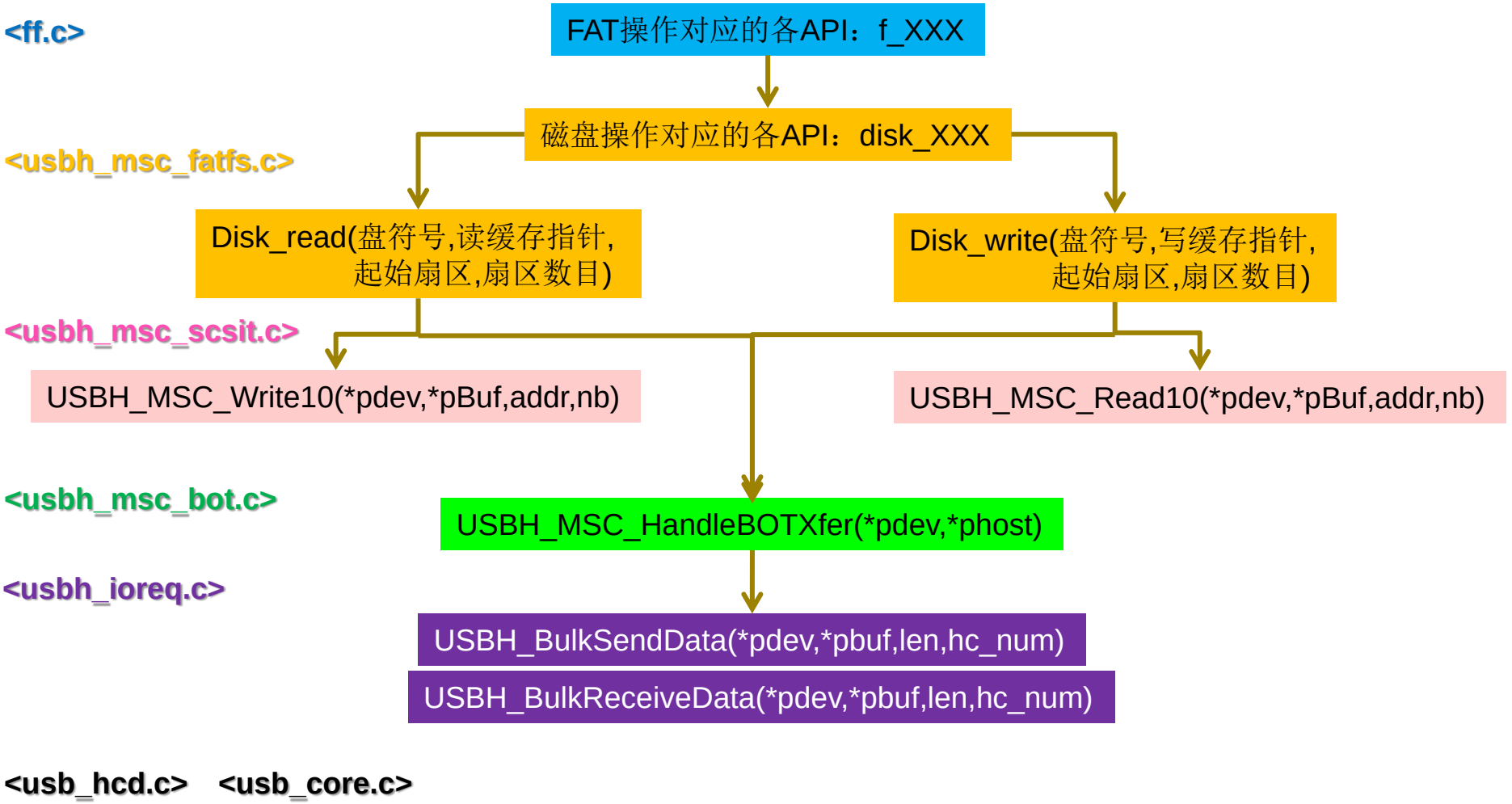
FatFs应用接口


```
FRESULT f_mount     (BYTE, FATFS*);                          /* Mount/Unmount a logical drive */
FRESULT f_open      (FIL*, const XCHAR*, BYTE);              /* Open or create a file */
FRESULT f_read      (FIL*, void*, UINT, UINT*);              /* Read data from a file */
FRESULT f_write     (FIL*, const void*, UINT, UINT*);        /* Write data to a file */
FRESULT f_lseek     (FIL*, DWORD);                           /* Move file pointer of a file object */
FRESULT f_close     (FIL*);                                  /* Close an open file object */
FRESULT f_opendir   (DIR*, const XCHAR*);                    /* Open an existing directory */
FRESULT f_readdir   (DIR*, FILINFO*);                        /* Read a directory item */
FRESULT f_stat      (const XCHAR*, FILINFO*);                /* Get file status */
FRESULT f_getfree   (const XCHAR*, DWORD*, FATFS**);         /* Get number of free clusters on the drive */
FRESULT f_truncate  (FIL*);                                  /* Truncate file */
FRESULT f_sync      (FIL*);                                  /* Flush cached data of a writing file */
FRESULT f_unlink    (const XCHAR*);                          /* Delete an existing file or directory */
FRESULT f_mkdir     (const XCHAR*);                          /* Create a new directory */
FRESULT f_chmod     (const XCHAR*, BYTE, BYTE);              /* Change attribute of the file/dir */
FRESULT f_utime     (const XCHAR*, const FILINFO*);          /* Change time-stamp of the file/dir */
FRESULT f_rename    (const XCHAR*, const XCHAR*);            /* Rename/Move a file or directory */
FRESULT f_forward   (FIL*, UINT(*)(const BYTE*,UINT), UINT, UINT*);  /* Forward data to the stream */
FRESULT f_mkfs      (BYTE, BYTE, WORD);                      /* Create a file system on the drive */
FRESULT f_chdir     (const XCHAR*);                          /* Change current directory */
FRESULT f_chdrive   (BYTE);                                  /* Change current drive */
```

Disk I/O Interface

```
disk_initialize     初始化的磁盘驱动器
disk_status -       获取磁盘状态
disk_read           读扇区
disk_write          写扇区
disk_ioctl          关于存储介质的其他杂项功能
```

调用结构、层次框图


<ff.c>
FAT操作对应的各API：f_XXX
磁盘操作对应的各API：disk_XXX
<usbh_msc_fatfs.c>
Disk_read(盘符号,读缓存指针,
起始扇区,扇区数目)
Disk_write(盘符号,写缓存指针,
起始扇区,扇区数目)
<usbh_msc_scsit.c>
USBH_MSC_Write10(*pdev,*pBuf,addr,nb)
USBH_MSC_Read10(*pdev,*pBuf,addr,nb)
<usbh_msc_bot.c>
USBH_MSC_HandleBOTXfer(*pdev,*phost)
<usbh_ioreq.c>
USBH_BulkSendData(*pdev,*pbuf,len,hc_num)
USBH_BulkReceiveData(*pdev,*pbuf,len,hc_num)
<usb_hcd.c> <usb_core.c>