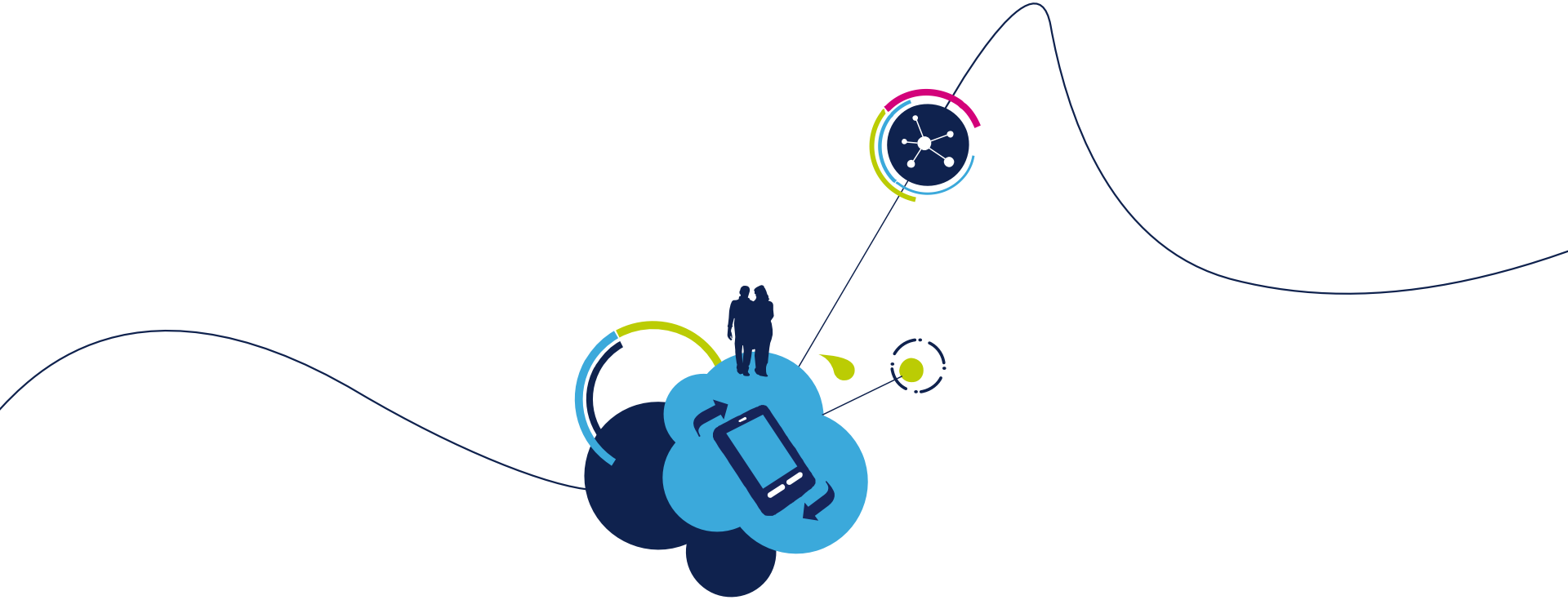
STM32 USB OTG _FS/HS模块

2018年5月

life.augmented



OTG模块概览

OTG_FS和OTG_HS模块通用特性比较

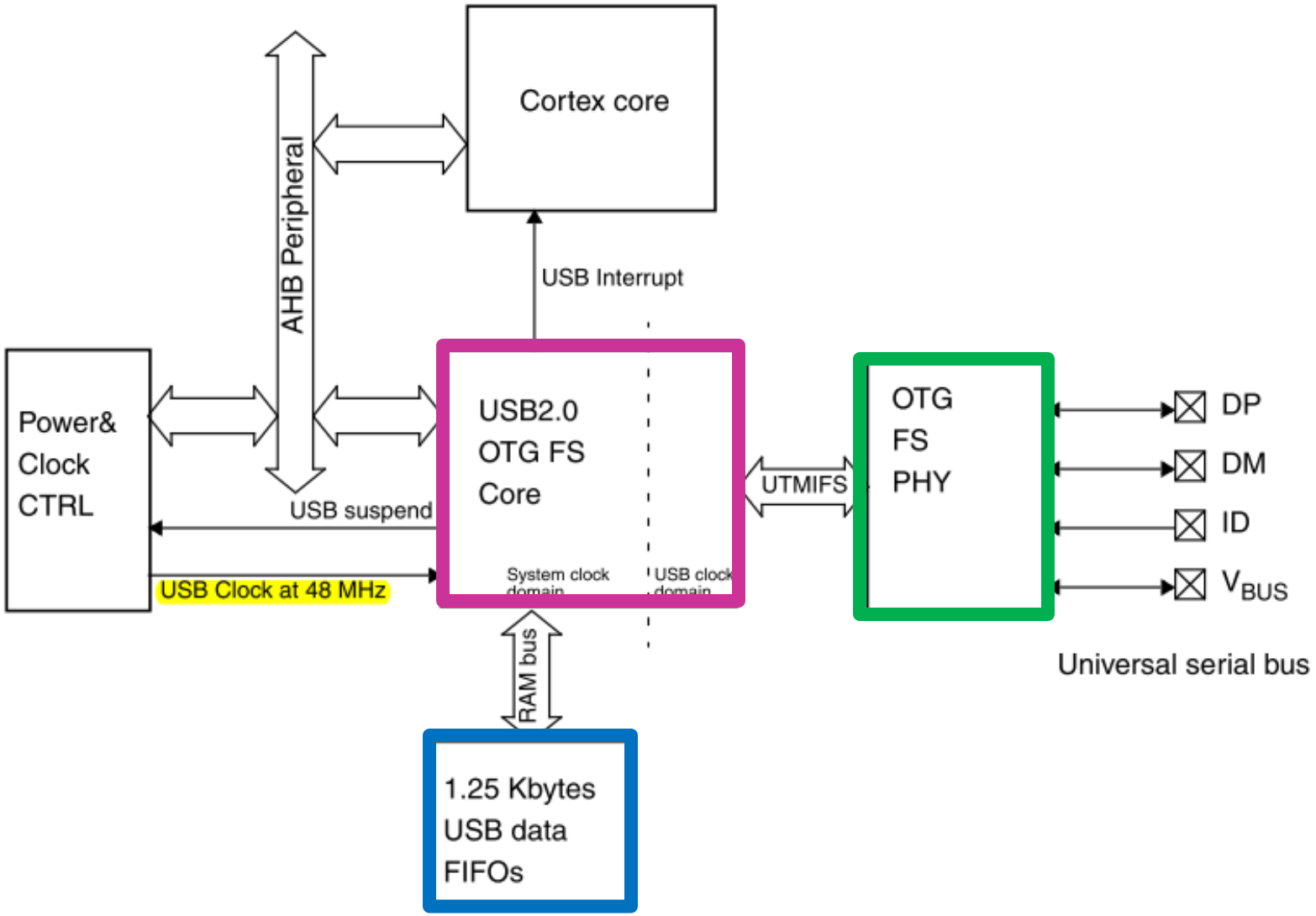
OTG_FS	OTG_HS
USB 2.0协议，OTG 1.3协议（支持HNP和SRP）	
可作为USB主机、USB设备、OTG设备(A类/B类)使用	
输出SOF信号，供各种同步应用（to PAD, to TIM2）	
相同的省电特性	
AHB主频必须高于14.2MHz	AHB主频必须高于30MHz
FIFO使用1.25KB专用RAM	FIFO使用4KB专用RAM
	内置独立的DMA管理FIFO的数据传输
可使用内部FS PHY做FS通信	
	具有ULIP接口，可和外部HS PYH连接做HS通信

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	
可作为FS、LS主机	可作为HS、FS、LS主机
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
1.25KB专用RAM：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 4KB专用RAM	
一个共享的RXFIFO用以接收数据	
设备特性比较	
可作为FS设备	可作为HS、FS设备
4个双向端点(包括端点0)	6个双向端点(包括端点0)
1.25KB专用RAM：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：4KB专用RAM
1个共享的RX FIFO	
支持软件断开	

OTG_FS功能框图描述



全速OTG核心模块

- 从RCC模块接收48MHz+/- 0.25%精度的时钟，必须在配置全速OTG核心模块之前将时钟使能
- CPU通过AHB总线访问核心模块的寄存器；USB中断事件由单独一条“OTG中断线”连到NVIC
  - CPU往“push register”中的写操作 → 数据自动写入数据发送FIFO
    - 设备模式下，每个IN端点有各自的一个“push register”
    - 主机模式下，每个OUT通道有各自的一个“push register”
  - CPU从“pop register”中的读操作 → 数据自动从共享RX-FIFO中读取出来
    - 设备模式下，每个OUT端点有各自的一个“pop register”
    - 主机模式下，每个IN通道有各自的一个“pop register”
- 片上PHY内集成的FS/LS收发模块负责硬件实现USB协议层

FIFO读写是基于专用地址区域的操作

- 对FIFO的读(pop)写(push)操作，是通过对特定地址区域的访问完成的
- 根据端点/通道的方向，决定对FIFO的操作是读还是写
  - e.g. OTG Device, IN端点对应的是发送FIFO，只能push操作，即软件写对应区域

FIFO access register section	Address range (offset)	Access
Device IN Endpoint 0/Host OUT channel 0 : FIFO write access Device OUT Endpoint 0/Host IN channel 0 : FIFO read access	0x1000 – 0x1FFC	W R
Device IN Endpoint 1/Host OUT channel 1 : FIFO write access Device OUT Endpoint 1/Host IN channel 1 : FIFO read access	0x2000 – 0x2FFC	W R
…		…
Device IN Endpoint x(1) /Host OUT channel x(1) : FIFO write access Device OUT Endpoint x(1) /Host IN channel x(1) : FIFO read access	0xX000 – 0xXFFC	W R

1. Where x= 3 for device mode and 7 for host mode

全速OTG PHY的组件

- 作为主机或设备时使用的FS/LS收发模块
- ID线上集成上拉电阻，用以在识别A/B设备时采样ID信号线电平
- D+/D-线上集成上拉和下拉电阻，根据当前角色，由核心模块控制使能
  - 设备角色：检测到Vbus有效电平(B-session valid)就使能D+上的上拉电阻---->FS
  - 主机角色：使能D+/D-上的下拉电阻
  - 上下拉电阻可在HNP协议下根据设备当前角色动态使能、关闭
- 带上下拉电阻的ECN电路
  - 对D+上拉阻值的动态调整可以提高噪声抑制能力和 Tx/Rx 信号质量
- 带滞回的Vbus检测比较器
  - 用于检测：Vbus有效电压门限、A-B会话有效电压门限、会话结束电压门限
- Vbus脉冲电路
  - 用于在SRP期间通过电阻对 VBUS 充电/放电(驱动力较弱）……

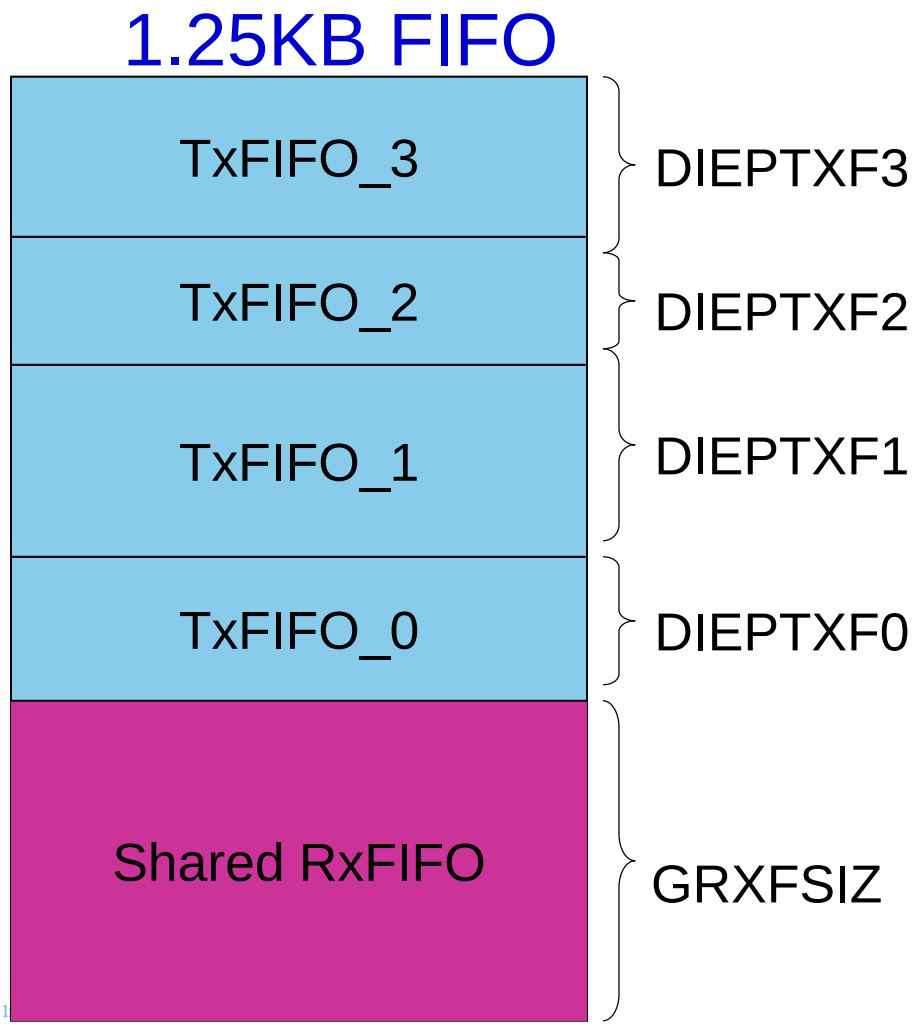
为使FS OTG PHY正常工作，AHB主频不能低于14.2MHz

全速OTG数据FIFO

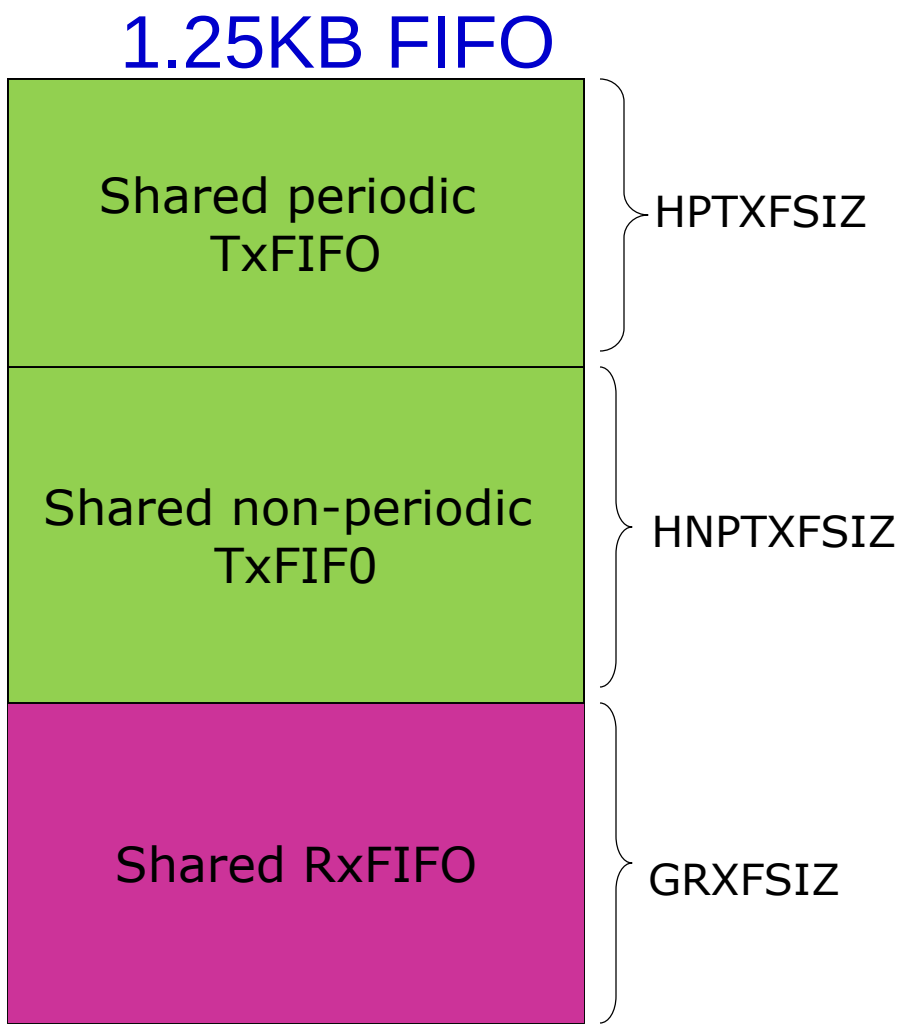
- 1.25KB专用RAM，精巧的FIFO控制机制
- 多个Tx-FIFO
  - 在通过USB总线发送出去之前，应用把数据写到这里(push)做临时存储
- 一个共享的Rx-FIFO
  - 从USB总线收下来的数据，在被应用读取(pop)之前，临时存储的地方
- FIFO的大小可软件配置
- FIFO的组织架构和分配取决于模块当前角色

FIFO空间的分配

- 设备模式下



- 主机模式下



清空FIFO

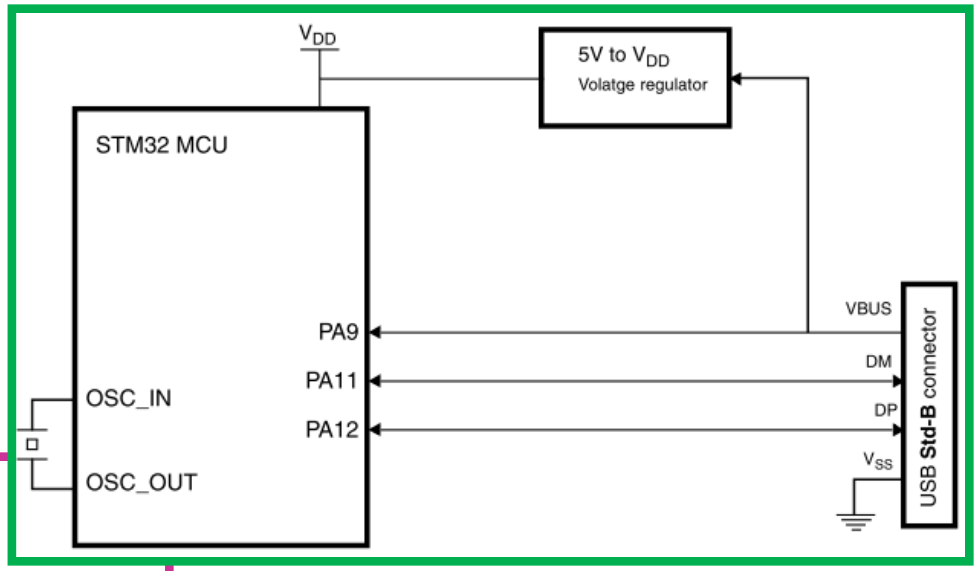
- 通过全局复位寄存器来清空FIFO（主机、设备都适用）
  - TX FIFO
    - TXFNUM：选择待清空的TX FIFO编号，或所有TX FIFO(0x10000)
    - TXFFLSH：程序要先确认当前待清空的TX FIFO编号没有在使用，再置位该位
    - 如何确认没有被使用？
      - 通过EP上的**NAK Effective**中断，来确认MAC没有在读取TX FIFO
      - 通过AHBIDL@GRSTCTL来确认内核没有在写TX FIFO
  - RX FIFO
    - 程序要先确认当前没有对RX FIFO的使用，再置位该位；然后等待该位被硬件清零（一般需要8个clock cycle）再往下继续
    - Clock cycle = PHY或AHB clock最短的那个

OTG_FS reset register (OTG_FS_GRSTCTL)

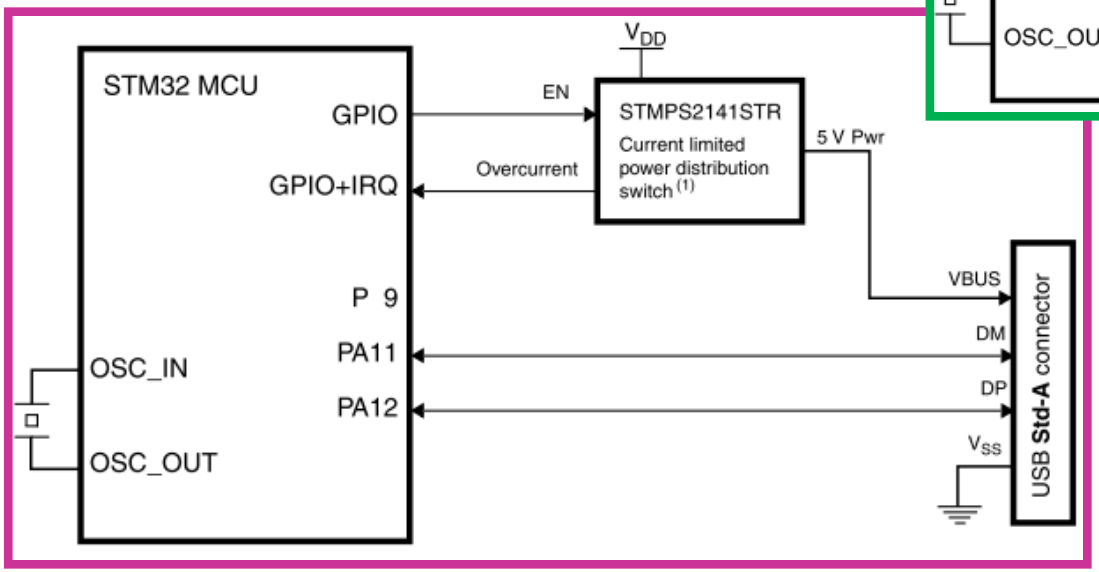
10 9 8 7 6	5	4
TXFNUM	TXFFLSH	RXFFLSH
rw	rs	rs

典型硬件连接框图

- 仅作USB设备

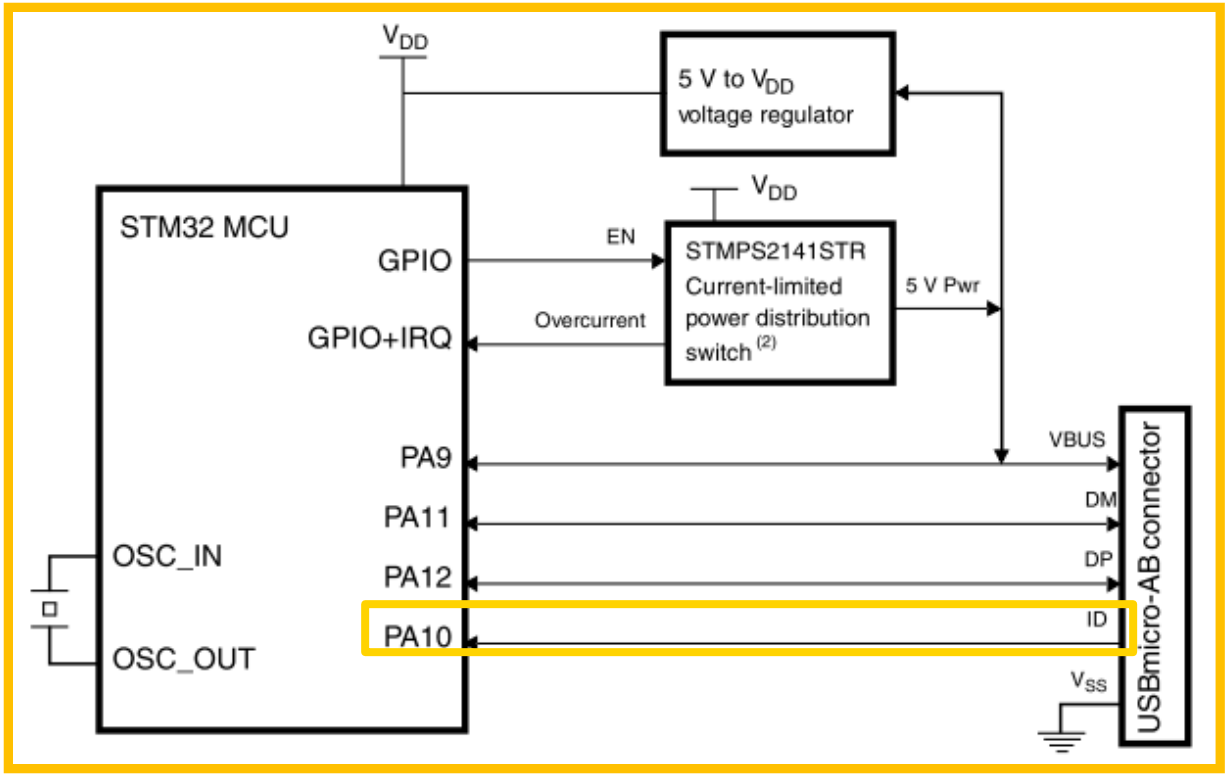


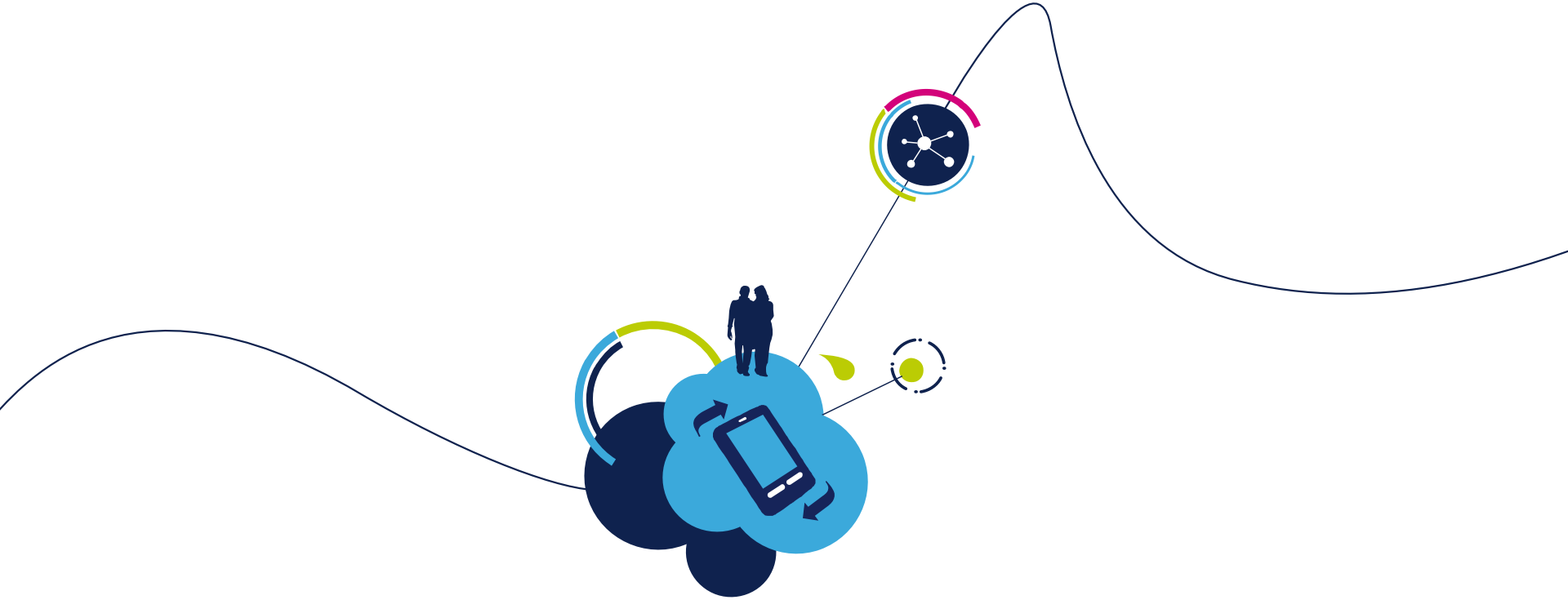
- 仅作USB主机



典型硬件连接框图

- 全角色OTG

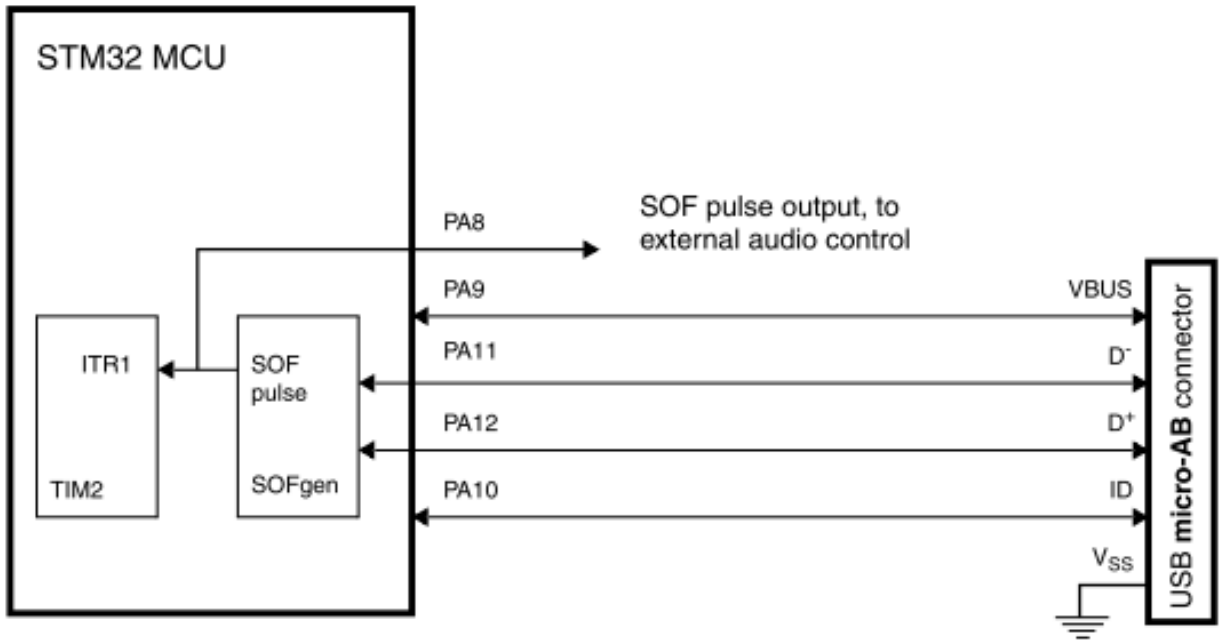




OTG模块的SOF特性

SOF触发信号

- 全速OTG模块可在主机或设备角色下监测、跟踪、以及在主机角色下配置两个相邻SOF之间的帧周期长度，并且还可以输出SOF脉冲
- 适用于自适应音频时钟产生技术
  - 音频从设备需要根据和主机的音频数据流保持同步
  - 音频主设备需要根据从设备当前的要求动态调整帧率



主、从角色下的SOF功能

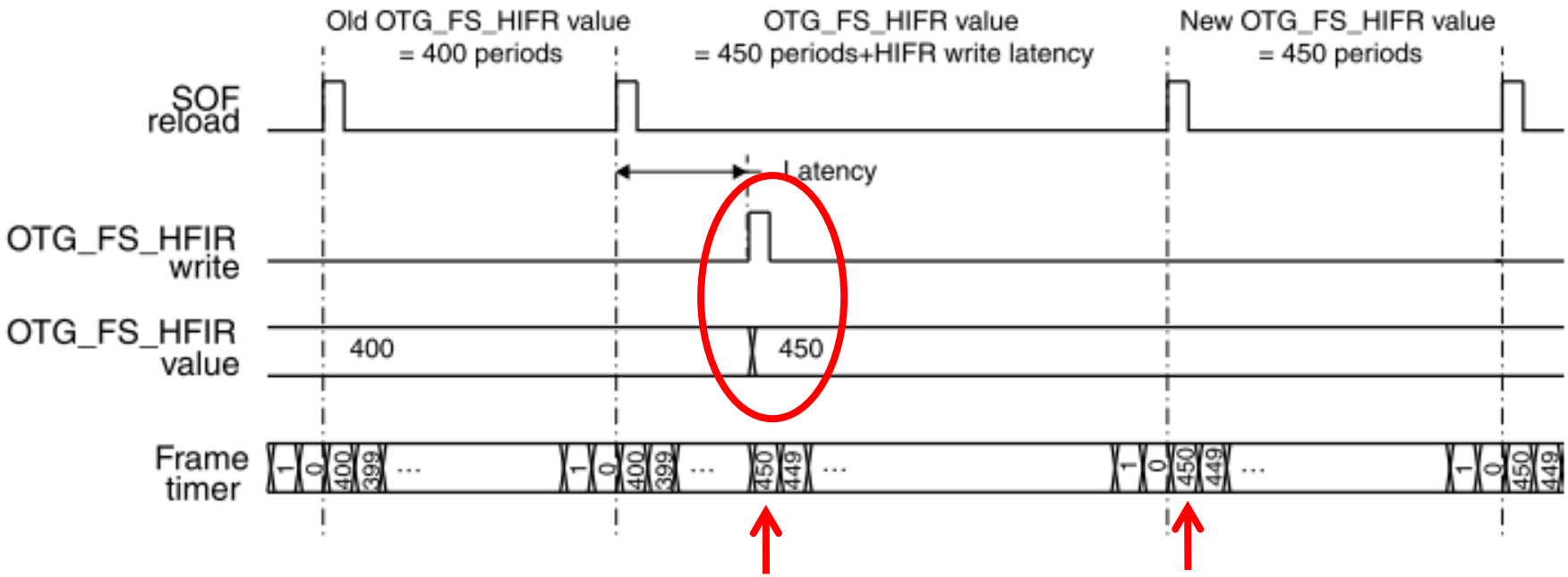
- 主机
  - 主机可以完全控制两个连续SOF令牌之间的时间（多少个PHY时钟）
  - **每发出一个SOF令牌，会产生对应SOF中断**（SOF@OTG_FS_GINTSTS）
    - 可从HFNUM读出当前帧号，以及当前帧剩余时间（多少个PHY时钟）
  - **还会产生一个SOF脉冲信号**
    - 宽度为12个系统时钟周期
    - 内联到TIM2的输入触发：TIM2的触发、输入捕获、输出比较都可由SOF脉冲触发
    - 也可通过SOFOUTEN@OTG_FS_GCCFG从SOF引脚输出到芯片外部

见下页

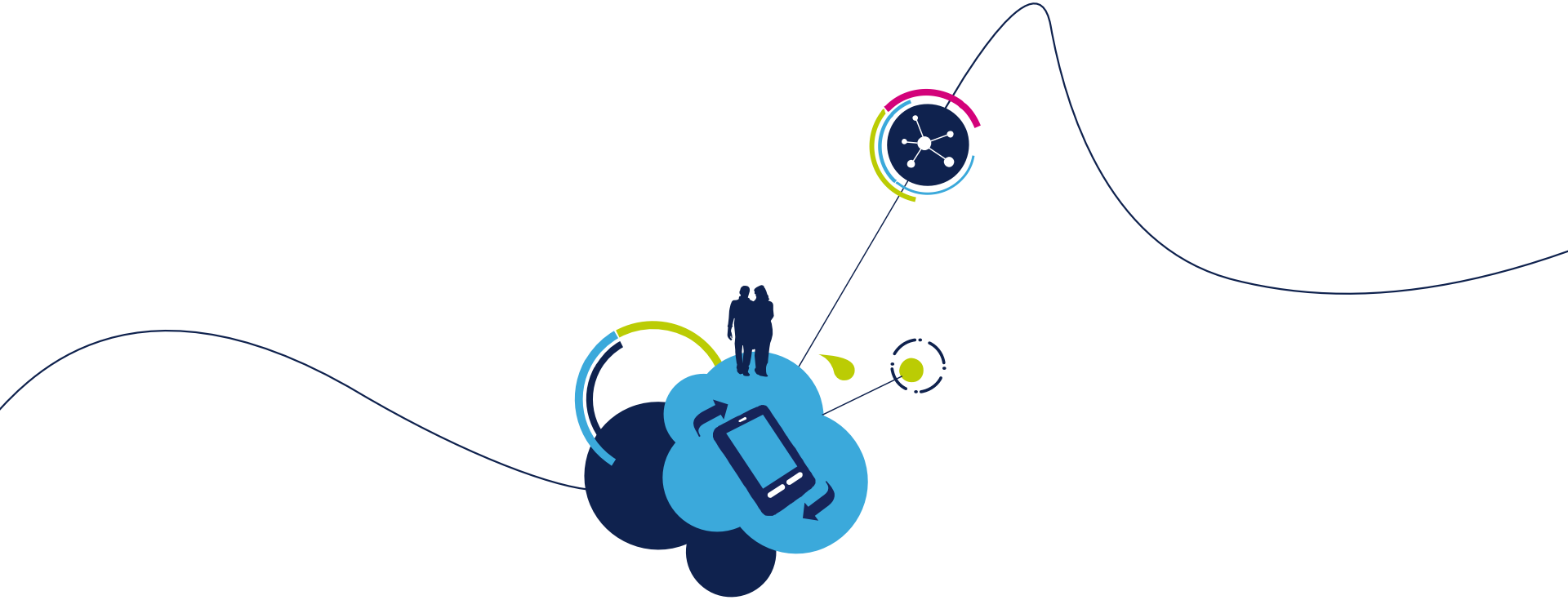
- 从机
  - **每收到SOF令牌就会产生SOF中断**（SOF@OTG_FS_GINTSTS）
    - 在此中断ISR可以读取当前帧号（FNSOF@OTG_FS_DSTS）
  - **还会产生一个SOF脉冲信号**
    - 宽度为12个系统时钟周期
    - 内联到TIM2的输入触发：TIM2的触发、输入捕获、输出比较都可由SOF脉冲触发
    - 也可通过SOFOUTEN@OTG_FS_GCCFG从SOF引脚输出到芯片外部
  - **周期帧结束中断**（EOPF@OTG_FS_GINTSTS）
    - 可用于通知应用程序；一帧间隔的80%,85%,90%,95%时间已经过去

动态调整帧周期长度

- 主机可以动态调整(micro-)SOF帧周期
  - 当前帧修改OTG_FS_HFIR寄存器的值
  - 下一帧才开始真正生效



life.augmented

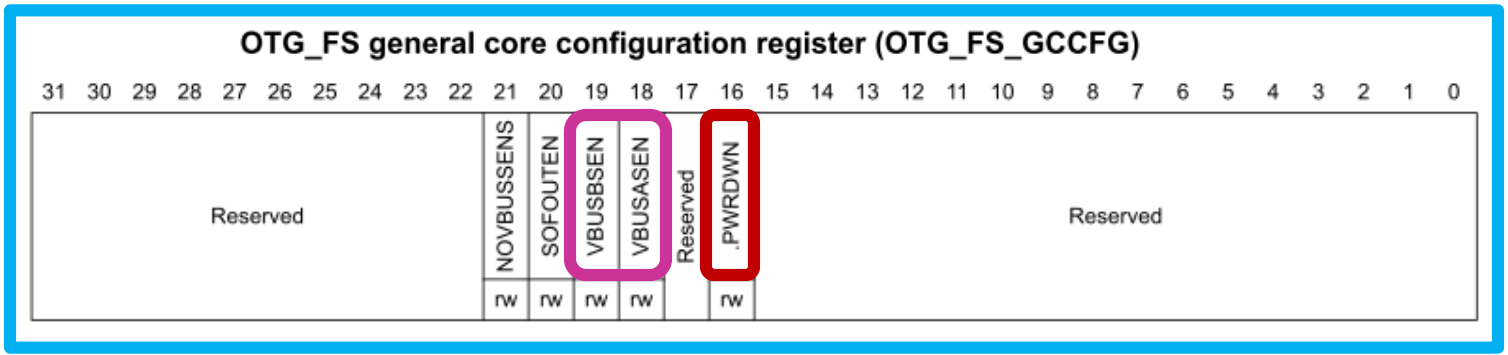


OTG模块的功耗特性

两个模块的功耗特性相同

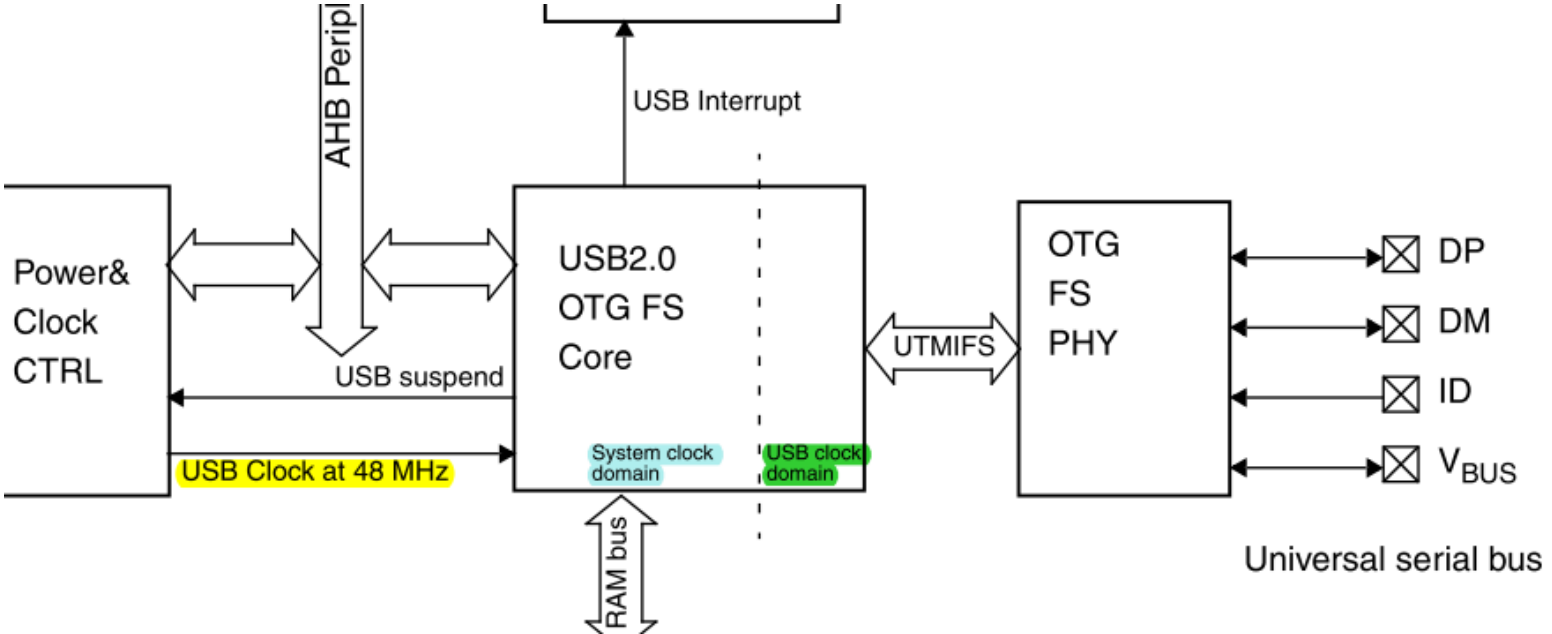
- 节电特性
  - USB挂起模式下停止系统时钟
  - 关闭数字模块、PHY和DFIFO的时钟

- OTG_PHY的功耗
  - PHY掉电： GCCFG/PWRDWN
    - 关闭PHY中全速收发模块
    - USB操作之前必须先把这位打开
  - A-VBUS监控： GCCFG/VBUSASEN
    - 关闭和A设备操作相关的Vbus比较器
    - 主机模式下或者HNP过程中，必须开启
  - B-VBUS监控： GCCFG/VBUSASEN
    - 关闭和B设备操作相关的Vbus比较器
    - 设备模式下或者HNP过程中，必须开启

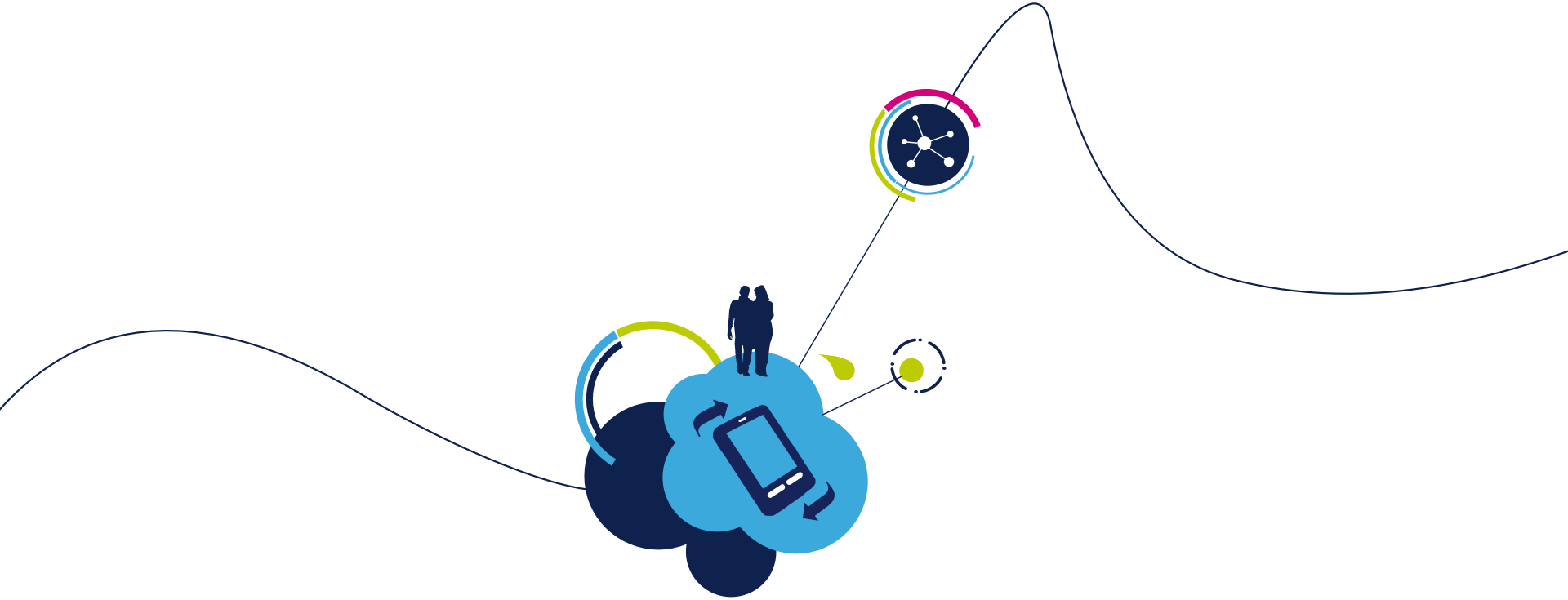


功耗特性

- 挂起模式下的功耗
  - 停止PHY时钟(**48MHz时钟区域**)：STPPCLK@OTG_FS_PCGCCTL
    - 节省掉时钟信号切换造成的动态功耗，即使此时还有48MHz时钟的输入
    - 收发模块的大部分被禁止，只有负责检测异步resume或者远程唤醒信号的功能模块还工作
  - 停止系统时钟(**HCLK时钟区域**)：GATEHCLK
    - 节省掉时钟信号切换造成的动态功耗，即时此时还有系统时钟的输入
    - 只有寄存器读写接口还保持工作
  - USB系统停止模式
    - 先设置“停止PHY时钟”位，再进入芯片的系统深睡眠@PWR
    - 当检测到异步的远程唤醒或者继续信号，OTG自动回复系统和USB时钟



life.augmented



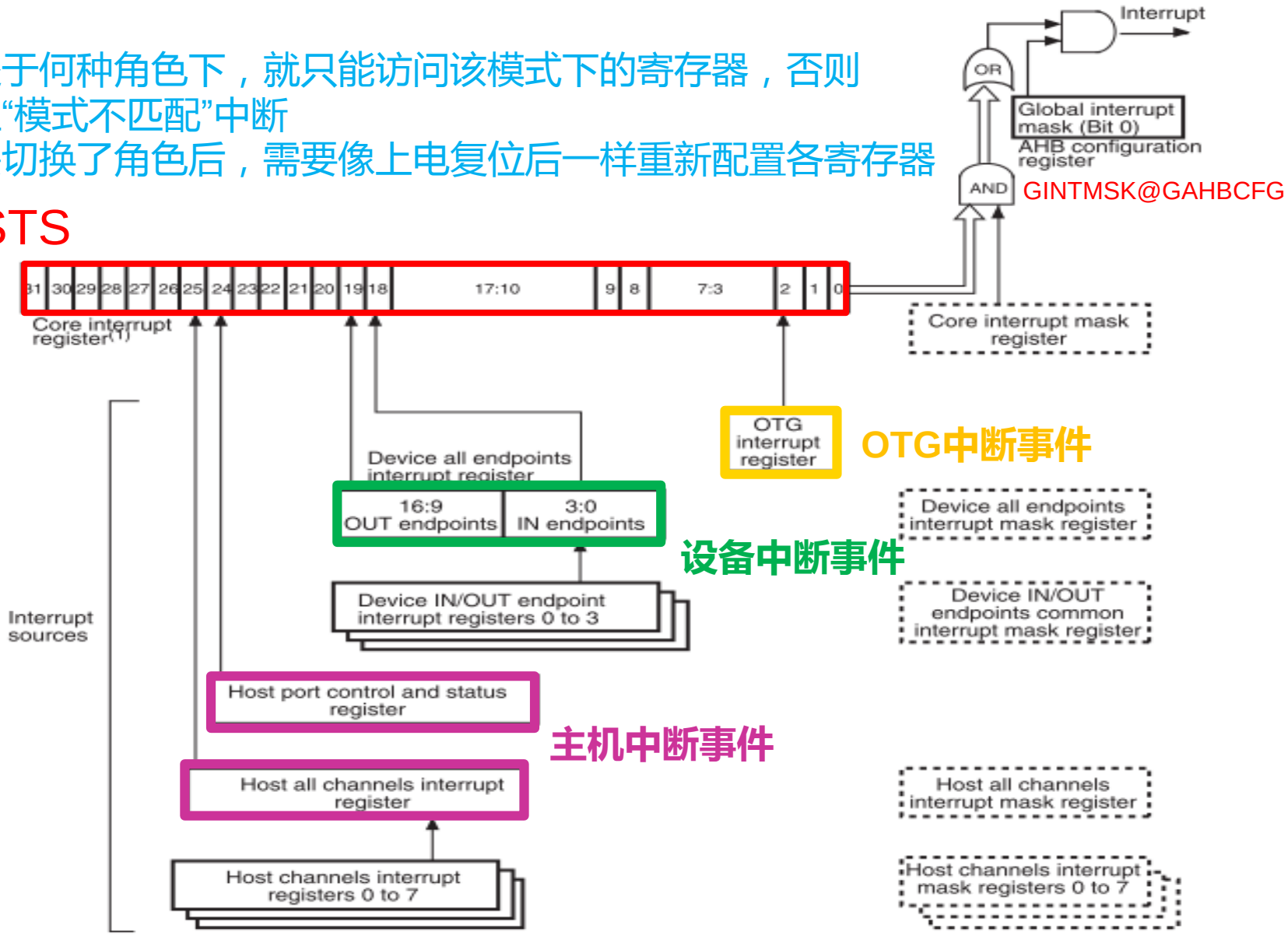
OTG模块的中断层级

OTG FS模块的中断层级

模块处于何种角色下，就只能访问该模式下的寄存器，否则会产生“模式不匹配”中断

当模块切换了角色后，需要像上电复位后一样重新配置各寄存器

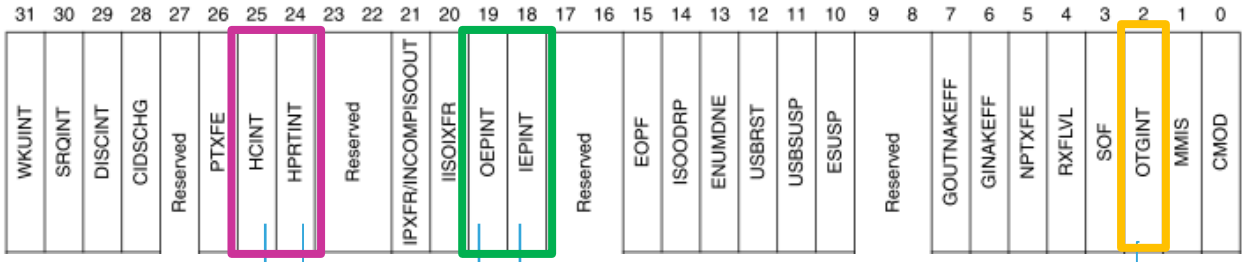
GINTSTS



life.augmented

内核中断寄存器

OTG_FS core interrupt register (OTG_FS_GINTSTS)



OTG_FS interrupt mask register

(OTG_FS_GINTMSK)

OTG事件中断

查看详细事件信息@ OTG_FS_GOTGINT

OUT端点中断

IN端点中断

查看具体端点号@ OTG_FS_DAINT

OTG_FS all endpoints interrupt mask register

(OTG_FS_DAINTMSK)

查看具体中断源 @ OTG_FS_DIEPINTx

OTG_FS device IN endpoint common interrupt mask

(OTG_FS_DIEPMSK)

查看具体中断源 @ OTG_FS_DOEPINTx

OTG_FS device OUT endpoint common interrupt mas

(OTG_FS_DOEPMSK)

主机通道中断

主机端口中断

查看具体中断源@ OTG_FS_HPRT

查看具体通道号@ OTG_FS_HAINT

查看具体中断源@ OTG_FS_HCINTx

OTG_FS Host all channels interrupt mask register

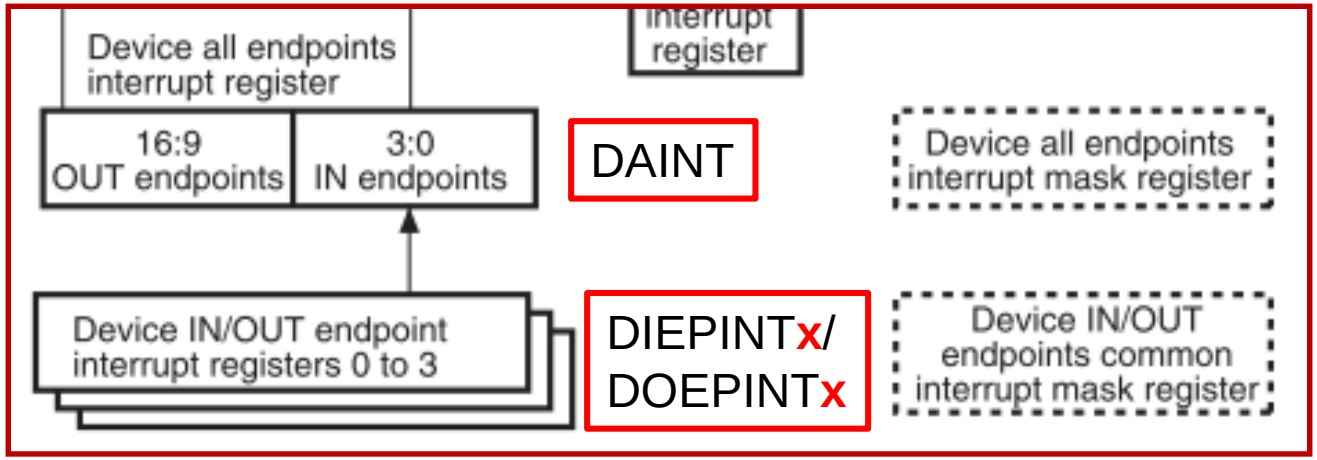
(OTG_FS_HAINTMSK)

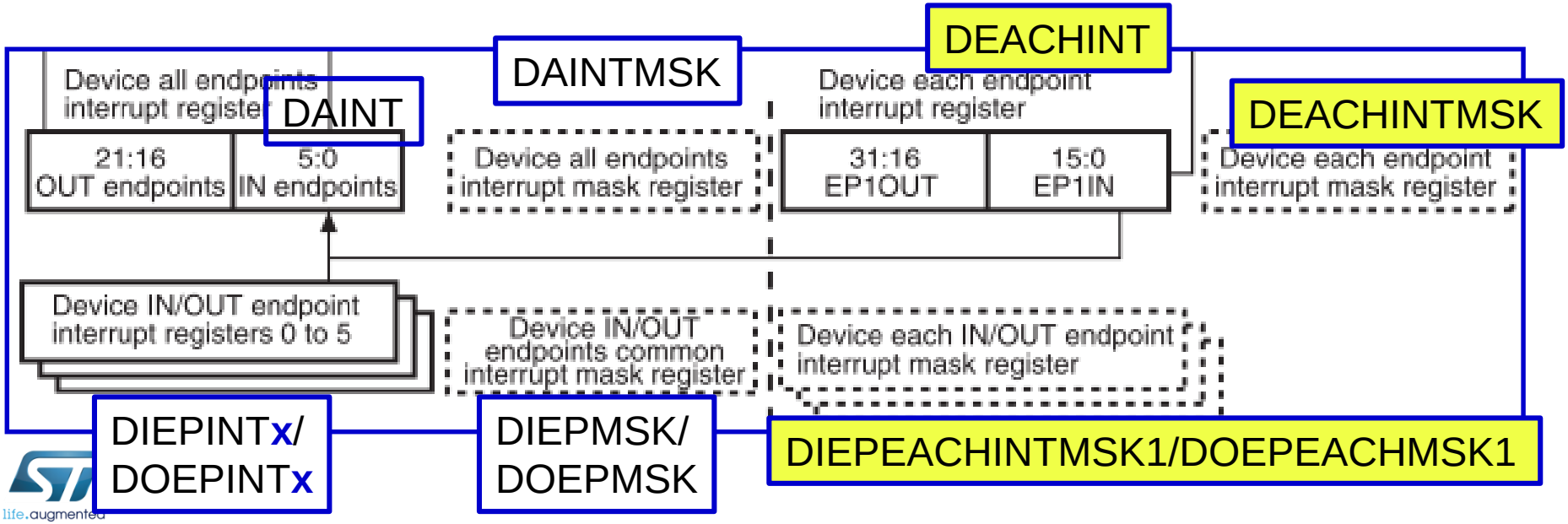
OTG_FS Host channel-x interrupt mask register

(OTG_FS_HCINTMSKx)

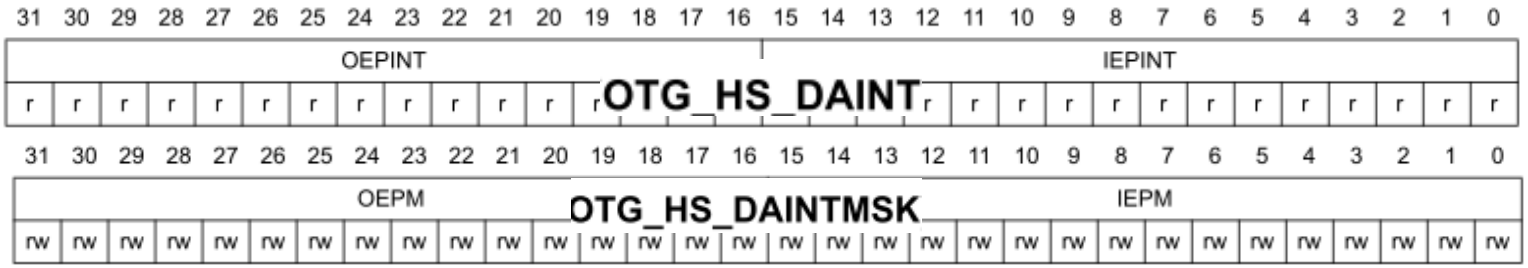
life.augmented

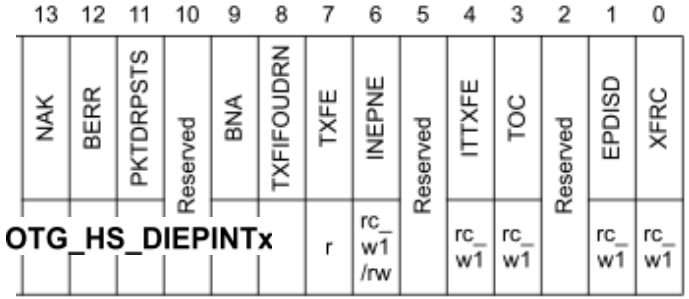
中断层级结构.设备

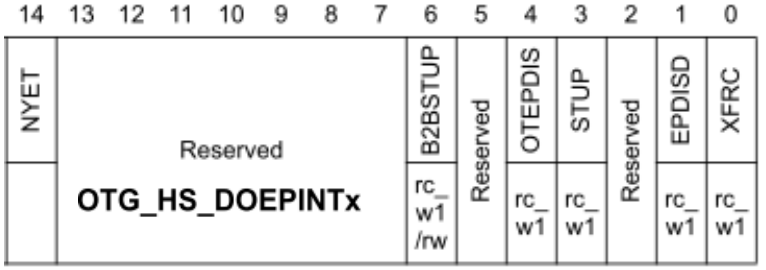


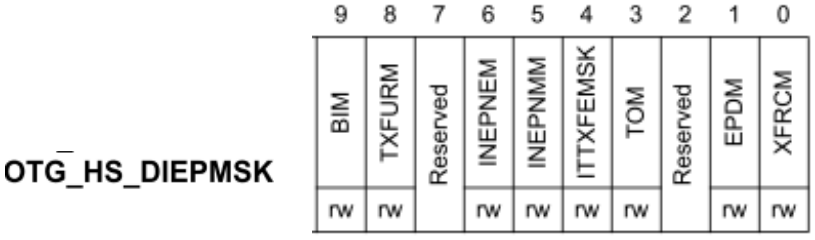


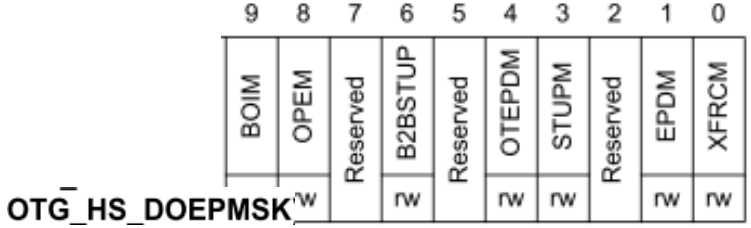
设备.中断寄存器

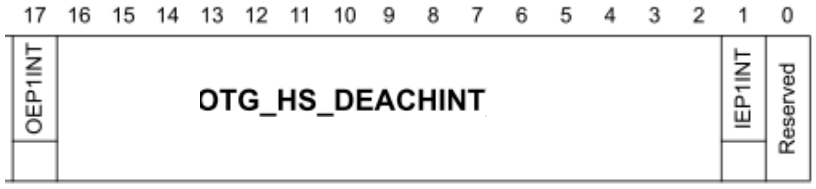


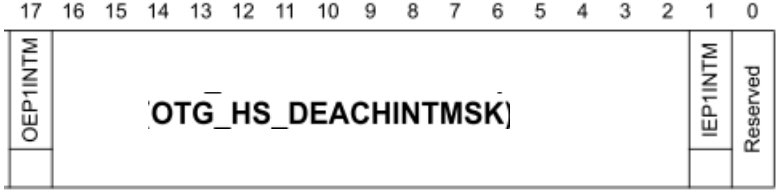


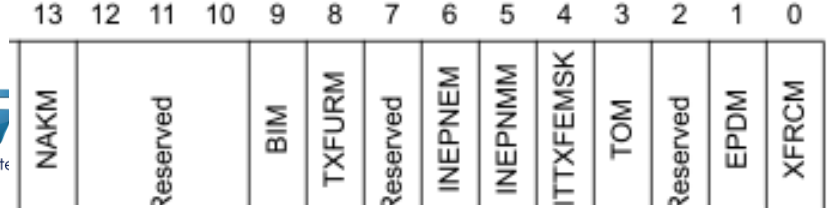


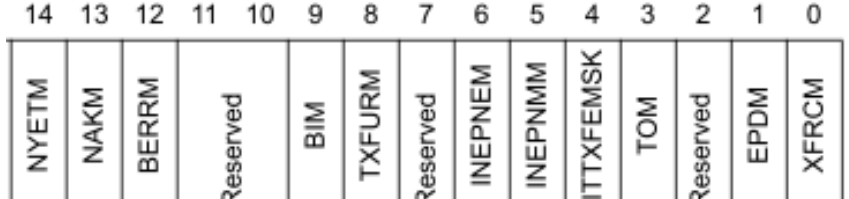












GINTSTS

位	域名		
31	WKUINT	检测到Resume/wkup信号	【H】【D】
30	SRQINT	会话请求/检测到新会话	【H】【D】
29	DISCINT	检测到设备从我这个主机断开连接	【H】
28	CIDSCHG	ID线状态变化	【H】【D】
26	PTXFE	周期传输发送FIFO空	【H】
25	HCINT	主机通道事件	【H】继续查看OTG_FS_HAINT和OTG_FS_**HCINTx**来确定哪个通道上的具体中断
24	HPRTINT	主机端口事件	【H】继续查看OTG_FS_**HPRT**来确定具体中断
21	IPXFER/INCOMPISOOUT	周期传输未完成	【H】/【D】
20	IISOIXFR	未完成的同步IN传输	【D】
19	OEPINT	OUT端点中断	【D】继续查看OTG_FS_DAINT和OTG_FS_**DOEPINTx**来确定哪个端点上的具体中断
18	IEPINT	IN端点中断	【D】继续查看OTG_FS_DAINT和OTG_FS_**DIEPINTx**来确定哪个端点上的具体中断
15	EOPF	周期帧结束了	【D】
14	ISOODRP	同步OUT包丢包(RxFIFO空间不够)	【D】

GINTSTS(2)

位	域名		
13	ENUMDNE	枚举完成	【D】读取OTG_FS_DSTS获得枚举出来的速度
12	USBRST	检测到USB复位	【D】
11	USBSUSP	检测到USB挂起状态	【D】
10	ESUSP	检测到早期USB挂起状态	【D】
7	GONAKEFF	“设置全局OUT NAK”位生效	【D】
6	GINAKEFF	“设置全局IN NAK”位生效	【D】
5	NPTXFE	非周期传输发送FIFO空	【H】
4	RXFLVL	接收FIFO非空	【H】【D】
3	SOF	发出/检测到SOF信号	【H】【D】
2	OTGINT	OTG中断事件	【H】【D】
1	MMIS	角色不匹配中断	【H】【D】
0	CMOD	当前角色	【H】【D】

HostChannelINTx (x=0…7)

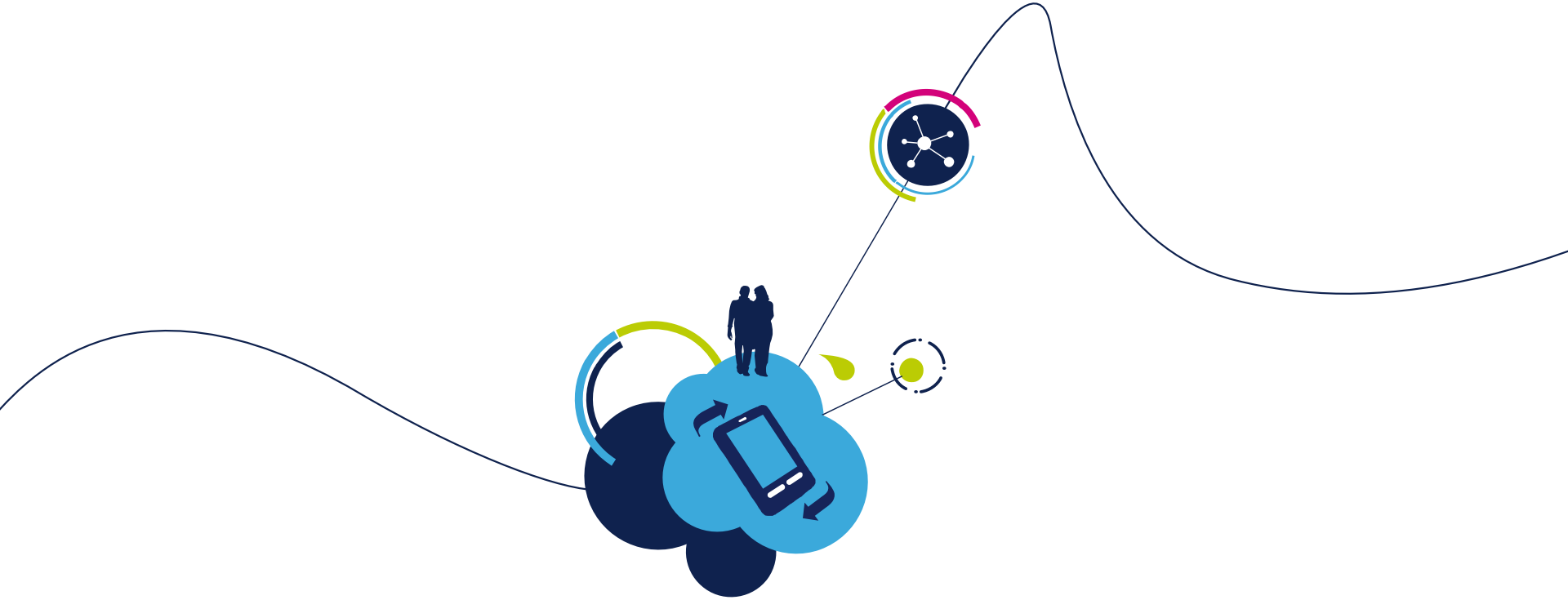
位	域名		
10	DTERR	数据toggle错误	应用需要先读取OTG_FS_HAINT来知晓发送中断的是哪个通道
9	FRMOR	Frame溢出错误	
8	BBERR	Babble错误	
7	TXERR	传输错误 >> CRC校验失败 >> 超时 >> 比特填充错误 >> 假的EOP	
6	NYET	Stm32f407xx.h	
5	ACK	发出或者收到ACK应答	
4	NAK	收到NAK应答	
3	STALL	收到STALL应答	
2	AHBERR	Stm32f407xx.h	
1	CHH	通道halt住，通信非正常中止 >> USB总线错误 >> 应用主动为之	
0	**XFRC**	**传输正确完成**	

DIEPINTx (x=0…3)

位	域名		
7	TXFE	该端点的发送FIFO空	应用需要先读取OTG_FS_DAINT来知晓发送中断的是哪个端点
6	INEPNE	该端点上的NAK位生效了	
4	ITTXFE	发送FIFO空，却收到了IN令牌	
3	TOC	**EP0的IN端点上**检测到超时(从上一个收到的IN令牌开始，发送超时了)	
1	EPDISD	端点被应用关闭	
0	**XFRC**	**该端点上的传输成功结束**	

DOEPINTx (x=0…3)

位	域名		
6	B2BSTUP	**EP0的OUT端点上**收到3个以上的背靠背SETUP包	应用需要先读取OTG_FS_DAINT来知晓发送中断的是哪个端点
4	OTEPDIS	**EP0的OUT端点上**收到OUT令牌，但是端点还未使能	
3	STUP	**EP0的OUT端点上**的SETUP阶段结束了，且没有背靠背的SETUP包，应用可以解码SETUP中的数据内容了	
1	EPDISD	端点被应用关闭	
0	**XFRC**	**该端点上的传输成功结束**	



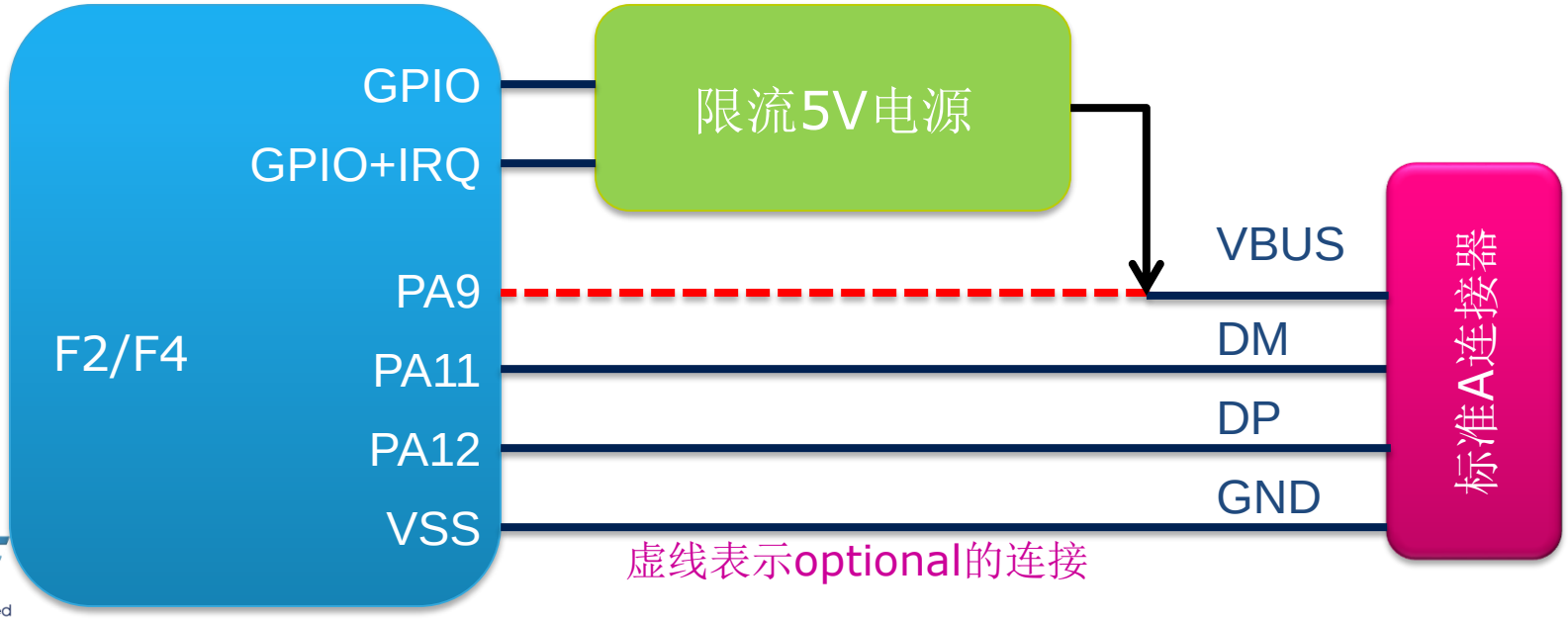
OTG模块作为USB主机

作为USB主机的四种情形

- OTG A主机
  - OTG A器件（连接的是A-side电缆）的默认状态，即插上A-side电缆时的状态
- OTG B主机
  - OTG B器件（连接的是B-side电缆）在HNP之后把角色切换成USB主机
- A器件
  - 连接的是A-side电缆，HNPCAP位被清零（角色不会变动了）
  - DP/DM线集成的下拉自动使能
- 仅作为USB主机
  - FHMOD被置位，强制处于USB主机角色
  - 此时是否有ID信号都无所谓，被忽略
  - DP/DM线集成的下拉自动使能
- OTG A主机、A器件、仅作为USB主机，三种情况下需要负责提供5V

OTG_FS模块作为”host only”的连接

- 芯片不支持5V输出，因此需要外部电源给Vbus提供5V
  - 由GPIO控制电压输出，由GPIO检测过流以在发送过流时切断供电
- PA9用以监测VBUS的供电
  - 使能HNP和SRP时必须连接Vbus
  - 取消该监控时(NOVBUSSENS)，PA9可用作普通I/O口；此时，VBUS被默认永远有效(5V)



USB主机的若干状态

- **给端口供电**
  - 通过GPIO控制外部charge pump给Vbus供电，还必须设置PPWR@OTG_FS_HPRT
  - 取消Vbus供电时，也必需清除PPWR
- **有效的总线电压**
  - HNP或SRP使能时，PA9必须作为Vbus输入监测
    - 在电压意外掉下去时（低于4.25V）可**触发中断**
    - 中断标志：SEDET@GOTGINT，
    - application必需关掉Vbus，并且清除端口供电位PPWR
  - HNP和SRP都未使能时，PA9不用连到Vbus，而可作为GPIO使用
  - 电压泵的过流输出可通过MCU上的任意空闲GPIO来告知主机，并产生**相应中断**
    - 这种情况下，application也必需关掉Vbus，并且清除端口供电位PPWR
- **设备断开和连上**
  - 主机端口变化：**触发中断**：HPRTINT@GINTSTS，表示主机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（ PCDET@HPRT ）
  - 设备断开：**触发中断** @ DISCINT@GINTSTS

主机状态：枚举

- 主机等待设备插入去抖完成，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置位是才有效
- 发送USB复位信号
  - 置位PRST@HPRT并保持至少10ms，不超过20ms后再复位该位
- 复位序列完成后，触发主机**中断**
  - PENCHNG@HPRT（该位是rc_w1）
  - 以此告知设备的速度已经获取，并可从PSPD@HPRT读取
  - 主机开始发送SOF或者Keep alives
  - 主机可以开始对设备发送枚举命令

	位域	位域名称	位域属性		备注
OTG_FS_HPRT	Bit5	POC CHNG	rc_w1，可产生中断	端口过流变化	Bit4变化会置位
	Bit4	POC A	r	端口是否过流	
	Bit3	PEN CHNG	rc_w1，可产生中断	端口使能变化	Bit2变化会置位
	Bit2	PEN A	rc_w0	端口是否使能	

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
  - **应用置位PRES@HPRT 来使得主机发送resume信号**
  - 应用需要掌控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

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……
- Case1：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~~~
- Case2：如何能够通过一直连在其上的设备唤醒主机
  - 设备通过remote wakeup机制在总线发起Resume信号来唤醒USB总线，会触发主机的EXTI18中断 (OTG_FS_WKUP_IRQ)

主机通道及相关寄存器

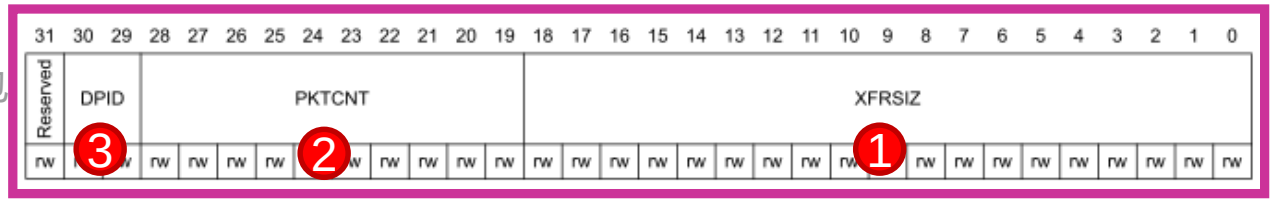
- OTG_FS模块有8个主机通道
  - 每个通道有各自的【通道属性】、【传输配置】、【状态/中断】、【掩码】寄存器
- 通道属性寄存器：HCCHARx
  - 通道enable/disable
  - 与该通道所连接的USB设备的速度FS/LS
  - 与该通道所连接的USB设备的地址
  - 与该通道所连接的USB设备上的端点号
  - 通道传输方向IN/OUT
  - 通道传输类型CTL/BLK/INT/ISO
  - 通道最大包长MPS
  - 黄色位：只针对周期传输（INT、ISO）有效

该通道的静态属性

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHENA	CHDIS	ODDFRM	DAD							MCNT		EPTYP		LSDEV	Reserved	EPDIR	EPNUM				MPSIZ										
rs	rs	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

主机通道及相关寄存器

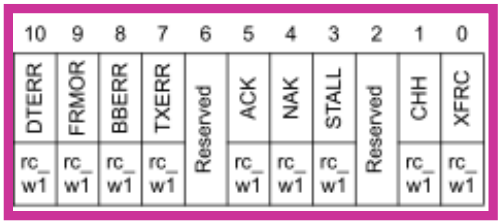
- 通道传输配置寄存器：HCTSIZx
  - 应用在此配置传输参数，完成后才能enable该通道（前页）
    1. 此次传输字节数
    2. 此次传输有多少个数据包
    3. 此次传输的初始数据PID



  - 一旦传输开始，应用可在此读取传输状态

XFRSIZ：【OUT传输】主机要发送的字节数
【IN传输】应用为接收数据预留的空间，设置成MPZ的整数倍

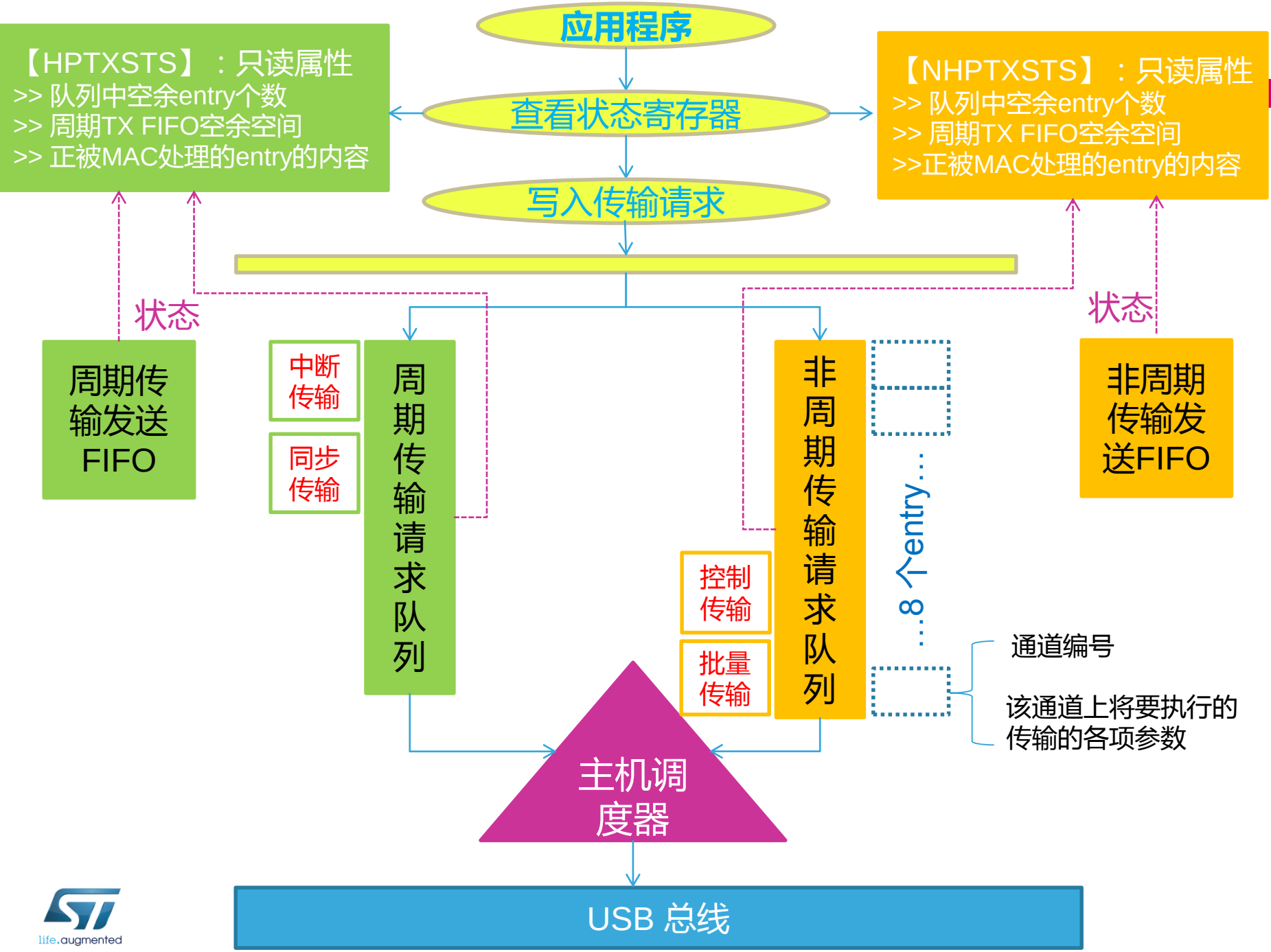
- 通道状态/中断寄存器：HCINTx
  - 硬件首先置位HCINT@GINTSTS
  - 应用读取HCAINT来获知到底哪个通道的事件触发了中断
  - 然后应用再读取HCINTx才能确定具体发生了些什么事件
    - Transfer完成
    - 通道disable（可由transfer完成、transaction出错，来自应用的disable命令导致）
    - 如果是IN方向通道：相关发送FIFO半空或者全空
    - 收到ACK
    - 收到NAK
    - 收到STALL
    - USB transaction出错（可由CRC失败、超时、比特填充、错误EOP导致）
    - Babble错误；Frame溢出错误、数据toggle错误



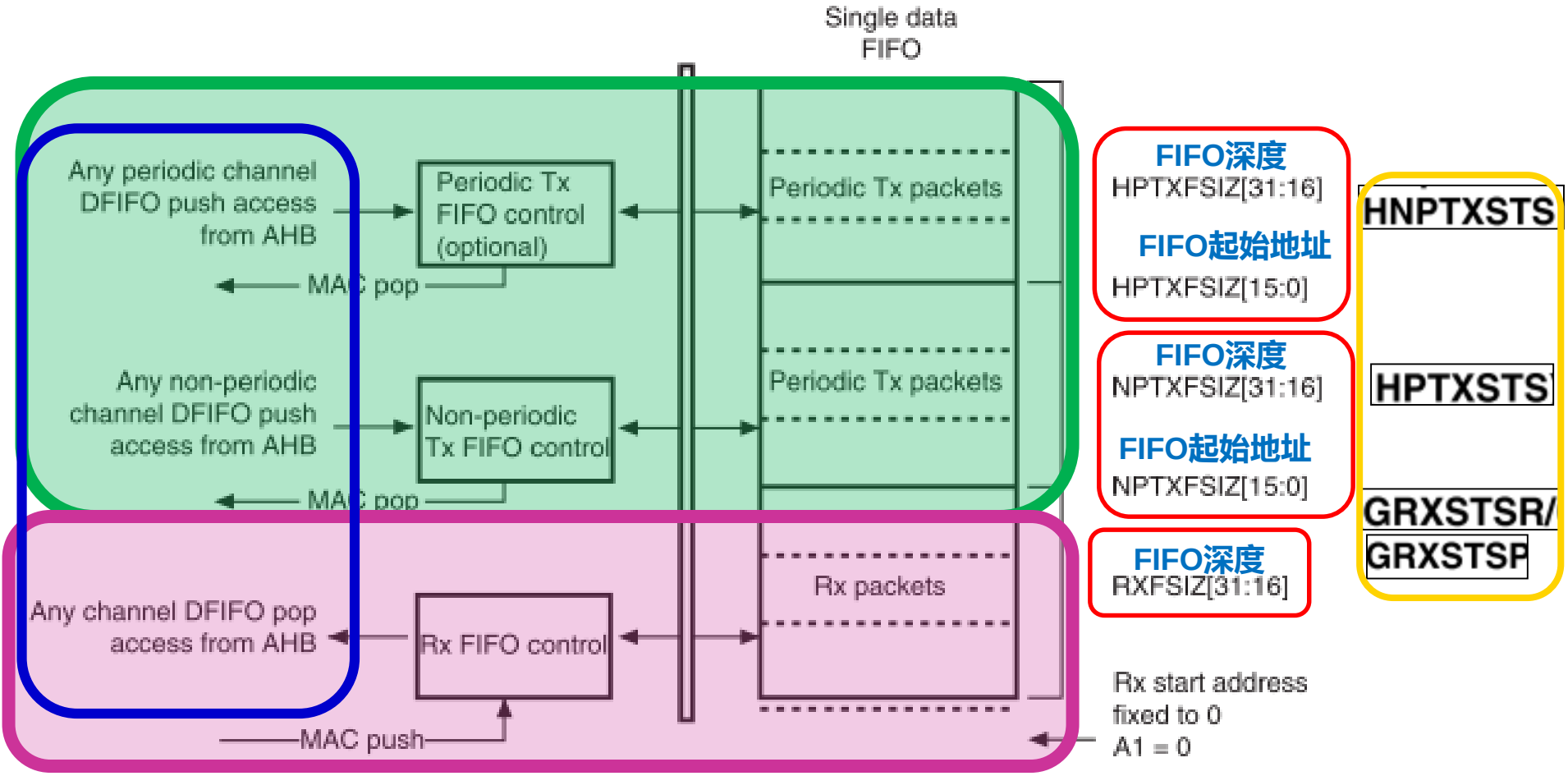
life.augmented

主机调度器 & 传输请求队列

- 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）；再处理非周期传输（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
    - 队列中各个传输请求的顺序，就是这些传输出现在总线上的顺序
  - 如果当前帧结束时，计划在当前帧执行的同步或中断类型的 USB 传输事务请求仍处于挂起状态，则主机 将发出未完成周期性传输中断IPXFR@GINTSTS
  - 应用读取每个**请求队列**的状态，均是只读寄存器
    - 非/周期传输发送FIFO和队列状态寄存器： HNPTXSTS / HPTXSTS
    - NPTQXSAV：非周期传输请求队列中还有几个空的entry（应用发请求之前必须先查它）
    - NPTXFSAV：如果要发起OUT传输，非周期发送FIFO中有几个空闲word
    - NPTXQTOP：非周期传输队列顶部正被MAC处理的那个entry中的内容
      - 通道号、【IN/OUT令牌、0长度数据包、通道挂起命令】、是否是最后一个entry


应用程序
查看状态寄存器
写入传输请求
【HPTXSTS】：只读属性
>> 队列中空余entry个数
>> 周期TX FIFO空余空间
>> 正被MAC处理的entry的内容
【NHPTXSTS】：只读属性
>> 队列中空余entry个数
>> 周期TX FIFO空余空间
>>正被MAC处理的entry的内容
状态
状态
周期传输发送FIFO
非周期传输发送FIFO
中断传输
同步传输
周期传输请求队列
非周期传输请求队列
控制传输
批量传输
8个entry
通道编号
该通道上将要执行的传输的各项参数
主机调度器
USB 总线

作为USB主机时的FIFO架构



life.augmented

主机FIFO的使用（1）

	FIFO始址和大小的配置	FIFO当前状态的查看	FIFO的使用备注
接收FIFO	**GRXFSIZ** >> 接收FIFO始址固定是0	**GRXSTSP** >> 收到的数据包状态 >> 收到的数据包PID >> 收到的数据包字节数 >> 收到的数据包所属的通道的编号	收到的数据包一个挨一个的存放； Packet的状态信息和data payload一起存放； 只要RX FIFO中有数据就不断产生非空中断RXFLVL@GINTSTS
非周期发送FIFO	**HNPTXFSIZ** >> FIFO起始地址 >> FIFO大小 >> 和EP0的发送FIFO尺寸寄存器复用	**HNPTXSTS** >> **非周期发送请求队列…** >> FIFO空闲空间 0 → 全满 X → X个字可用 (x!=0)	该缓存区用于存放发送packet中的数据负载； 只要TX FIFO半空或全空就会触发TX FIFO空中断:(N)PTXFE@GINTSTS； 只有TX FIFO和**请求队列**都有空闲位置，应用才可提前把要发送的数据和请求放进去
周期发送FIFO	**HPTXFSIZ** >> FIFO起始地址 >> FIFO大小	**HPTXSTS** >> **周期发送请求队列…** >> FIFO空闲空间 0 → 全满 X → X个字可用 (x!=0)	
共性	宽度为32位字，最小容量64B；最大容量1KB		

life.augmented

作为USB主机时的FIFO分配

- 一共1.25KB，即1280B，以32-bit为单位 → 320个location
- RX FIFO
  - 接收一个数据包的预留空间
    - 最小size要是(最大的MPS/4+1)个表项，因为接收包的状态信息也一起存入FIFO
    - 如果有多个ISO通道，最小size要是 [2*(最大MPS/4) +1]个表项
  - 每个transfer complete status
    - 一个表项
- TX FIFO
  - 非周期发送FIFO
    - 最小size要是主机所支持的众多非周期OUT通道中最大MPZ
    - 通常是取两倍的量（一个包在发到USB总线的过程中，CPU可以填入下一个包）
  - 周期发送FIFO
    - 最小size要是主机所支持的众多周期OUT通道中最大MPZ
    - 通常是取两倍的量（如果有ISO传输的话）

CDC Host Demo.设置FIFO大小

- USB_HostInit () ← HAL_HCD_Init() ← USBH_LL_Init() ← USBH_Init()

	RX FIFO	始址	HNPTXF	始址	HPTXF
FS OTG	**128-word** 512B	0x80	**96-word** 384B	0xE0	**64-word** 256B
HS OTG	512-word	-	-	-	-

@ <stm32f4xx_ll_usb.c>

```
if(USBx == USB_OTG_FS)
  {
    /* set Rx FIFO size */
    USBx->GRXFSIZ  = (uint32_t )0x80;
    USBx->DIEPTXF0_HNPTXFSIZ = (uint32_t )(((0x60 << 16)& USB_OTG_NPTXFD) | 0x80);
    USBx->HPTXFSIZ = (uint32_t )(((0x40 << 16)& USB_OTG_HPTXFSIZ_PTXFD) | 0xE0);
  }
  else
  {
    /* set Rx FIFO size */
    ……..
}
```

主机FIFO的使用（2）
读取通道的RX FIFO/写通道的TX FIFO

>> 这些寄存器，在设备和主机模式下都可访问

>> 用于对特定端点或者通道的DFIFO的读或者写访问

>> 若该通道是IN或者该端点是OUT，则只能对该FIFO执行写操作

>>若该通道是OUT或者该端点是IN，则只能对该FIFO执行读操作

例如：
设备IN EP0/ 主机OUT通道0 的写地址
也是
设备OUT EP0/ 主机IN 通道0 的读地址

life.augmented

```
void *USB_ReadPacket (*USBx, uint8_t *dest, uint16_t len)
{
  uint32_t i=0;
  uint32_t count32b = (len + 3) / 4;

  for ( i = 0; i < count32b; i++, dest += 4 )
  {
    *(__packed uint32_t *)dest = USBx_DFIFO(0);
  }
  return ((void *)dest);
}
```

为何是固定0？因为接收FIFO是共享的！

```
                                        0x1000                          0x1000
#define USBx_DFIFO(i) (USBx + USB_OTG_FIFO_BASE + (i) * USB_OTG_FIFO_SIZE)
```

```
USB_WritePacket (*USBx, uint8_t *src, uint8_t ch_ep_num, uint16_t len, uint8_t dma)
{
  uint32_t count32b= 0 , i= 0;
  if (dma == 0)
  {
    count32b =  (len + 3) / 4;
    for (i = 0; i < count32b; i++, src += 4)
    {
      USBx_DFIFO(ch_ep_num) = *((__packed uint32_t *)src);
    }
  }
  return HAL_OK;
}
```

```
                                        0x1000                          0x1000
#define USBx_DFIFO(i) (USBx + USB_OTG_FIFO_BASE + (i) * USB_OTG_FIFO_SIZE)
```

GINTSTS中适用于主机的中断

位	域名		
31	WKUINT	检测到Resume/wkup信号	【H】【D】
29	DISCINT	检测到设备从我这个主机断开连接	【H】
26	PTXFE	周期传输发送FIFO空	【H】
25	HCINT	主机通道事件	【H】继续查看OTG_FS_HAINT和OTG_FS_**HCINTx**来确定哪个通道上的具体中断
24	HPRTINT	主机端口事件	【H】继续查看OTG_FS_**HPR**T来确定具体中断
21	IPXFR/INCOMPISOOUT	周期传输未完成	【H】/【D】
5	NPTXFE	非周期传输发送FIFO空	【H】
4	RXFLVL	接收FIFO非空	【H】【D】
3	SOF	发出/检测到SOF信号	【H】【D】
1	MMIS	角色不匹配中断	【H】【D】
0	CMOD	当前角色	【H】【D】

主机通道中断标志HCINTx (x=0…7)

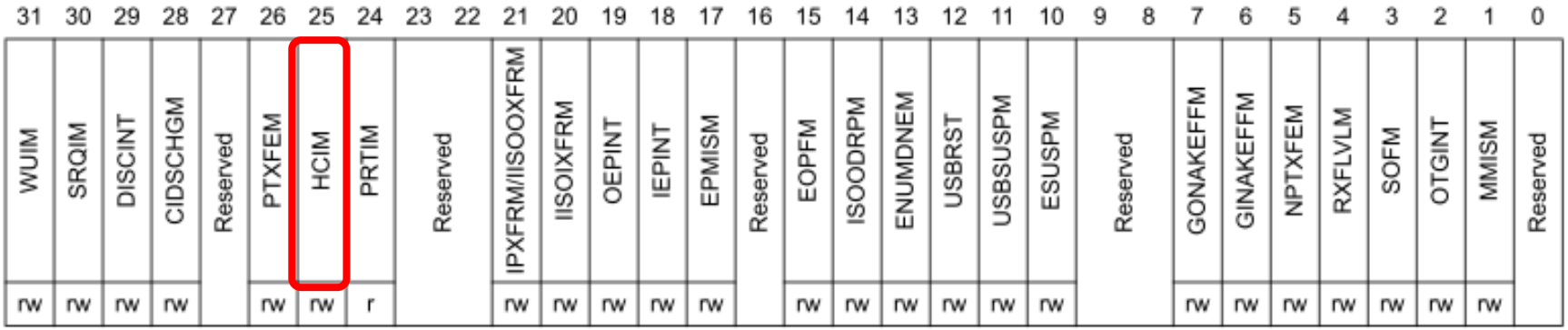
位	域名		备注	HC_IN_IRQ	HC_OUT_IRQ
10	DTERR	数据toggle错误	应用需要先读取OTG_FS_HAINT来知晓发送中断的是哪个通道		
9	FRMOR	Frame溢出错误			
8	BBERR	Babble错误		X	X
7	TXERR	传输错误 >> CRC校验失败 >> **超时** >> 比特填充错误 >> 假的EOP			
6	NYET	Stm32f407xx.h		X	
5	ACK	发出或者收到ACK应答			
4	NAK	收到NAK应答			
3	STALL	收到STALL应答			
2	AHBERR	Stm32f407xx.h			
1	CHH	通道halt住，通信非正常中止 >> USB总线错误 >> 应用主动为之			
0	**XFRC**	**传输正确完成**			

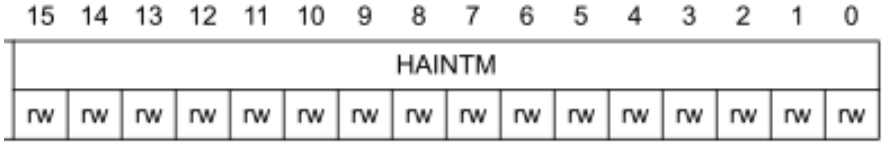
主机端口中断标志HPRT

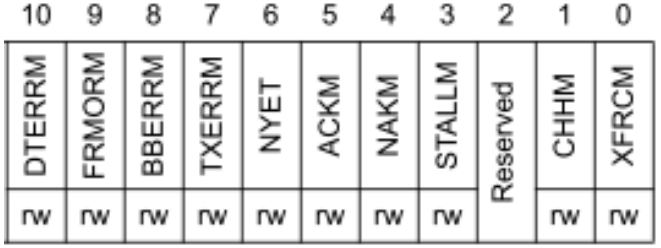
位	域名			Cube库中处理
5	P OC CHNG	端口过流标志改变	rc_w1，可产生中断	Y
4	P OC A	端口过流	r	
3	P EN CHNG	端口使能标志改变	rc_w1，可产生中断	Y
2	P EN A	端口使能	r	
1	P C DET	检测到端口有连接	rc_w1，可产生中断	Y
0	P C STS	端口连接状态	r	

主机编程模型.Initialize通道

- 应用必须初始化通道之后才能和所连设备（的端点）通信
  - 在OTG_FS_GINTMSK中打开**【主机通道】**中断
  - 在OTG_FS_GAINTMSK中打开**所选择的的通道**上的中断
  - 在OTG_FS_HCINTMSKx中打开**感兴趣的transaction相关的中断**
  - 在OTG_FS_HCTSIZx中设置传输的字节数、期望的数据包的个数、初始PID
  - 在OTG_FS_HCCHARx中设置该通道所连设备及端点的静态属性







life.augmented

Cube库.Initialize通道

Middleware/Core
<usbh_pipes.c>
USBH_OpenPipe
(USBH_HandleTypeDef *phost,
uint8_t pipe_num,
uint8_t epnum,
uint8_t dev_address,
uint8_t speed,
uint8_t ep_type,
uint16_t mps)

call

Application
<usbh_conf.c>
USBH_LL_OpenPipe
(USBH_HandleTypeDef *phost,
uint8_t pipe,
uint8_t epnum,
uint8_t dev_address,
uint8_t speed,
uint8_t ep_type,
uint16_t mps)

Driver/STM32F4xx_HAL_Driver
<stm32f4xx_hal_hcd.c>
HAL_HCD_HC_Init
(HCD_HandleTypeDef *hhcd,
uint8_t ch_num,
uint8_t epnum,
uint8_t dev_address,
uint8_t speed,
uint8_t ep_type,
uint16_t mps)

Driver/STM32F4xx_HAL_Driver
<stm32f4xx_ll_usb.c>
USB_HC_Init
(USB_OTG_GlobalTypeDef *USBx,
uint8_t ch_num,
uint8_t epnum,
uint8_t dev_address,
uint8_t speed,
uint8_t ep_type,
int16_t mps)

主机编程模型.Halt通道

- 应用程序采取以下操作来Halt通道
  - CHDIS=1, CHENA=1 @ OTG_FS_HCCHARx
  - 应用需要等到CHH@OTG_FS_HCINTx中断后才能再次为传输分配通道
  - 主机会把传输请求队列清空，并产生通道halt中断
  - 已经在USB总线上开始的transaction不会被中断
  - Halt通道之前，程序必须保证传输请求队列中至少有一个空闲entry。如果队列是满，则可以通过CHDIS=1, CHENA=0清空队列
- 以下情形下，应用需要halt通道
  - 通道上收到STALL回复中断和各种错误中断 (TXERR/BBERR/DTERR)
  - 检测到设备断开，应用必须halt掉所有之前使能的通道
    - 【说明】这就是host stack在检测到设备断开后，调用USBH_CDC_InterfaceDeInit()
  - 当应用想在传输正常结束前中止传输

Cube库.Halt通道

	CHENA	CHDIS
清空队列	0	1
Halt通道	1	1
使能通道	1	0

Middleware/Core
<usbh_pipes.c>
USBH_ClosePipe
(USBH_HandleTypeDef *phost,
uint8_t pipe_num)

call

Application
<usbh_conf.c>
USBH_LL_ClosePipe
(USBH_HandleTypeDef *phost,
uint8_t pipe)

Driver/STM32F4xx_HAL_Driver
<stm32f4xx_hal_hcd.c>
HAL_HCD_HC_Halt
(HCD_HandleTypeDef *hhcd,
uint8_t ch_num)

Driver/STM32F4xx_HAL_Driver
<stm32f4xx_ll_usb.c>
USB_HC_Halt
(USB_OTG_GlobalTypeDef *USBx,
uint8_t ch_num)

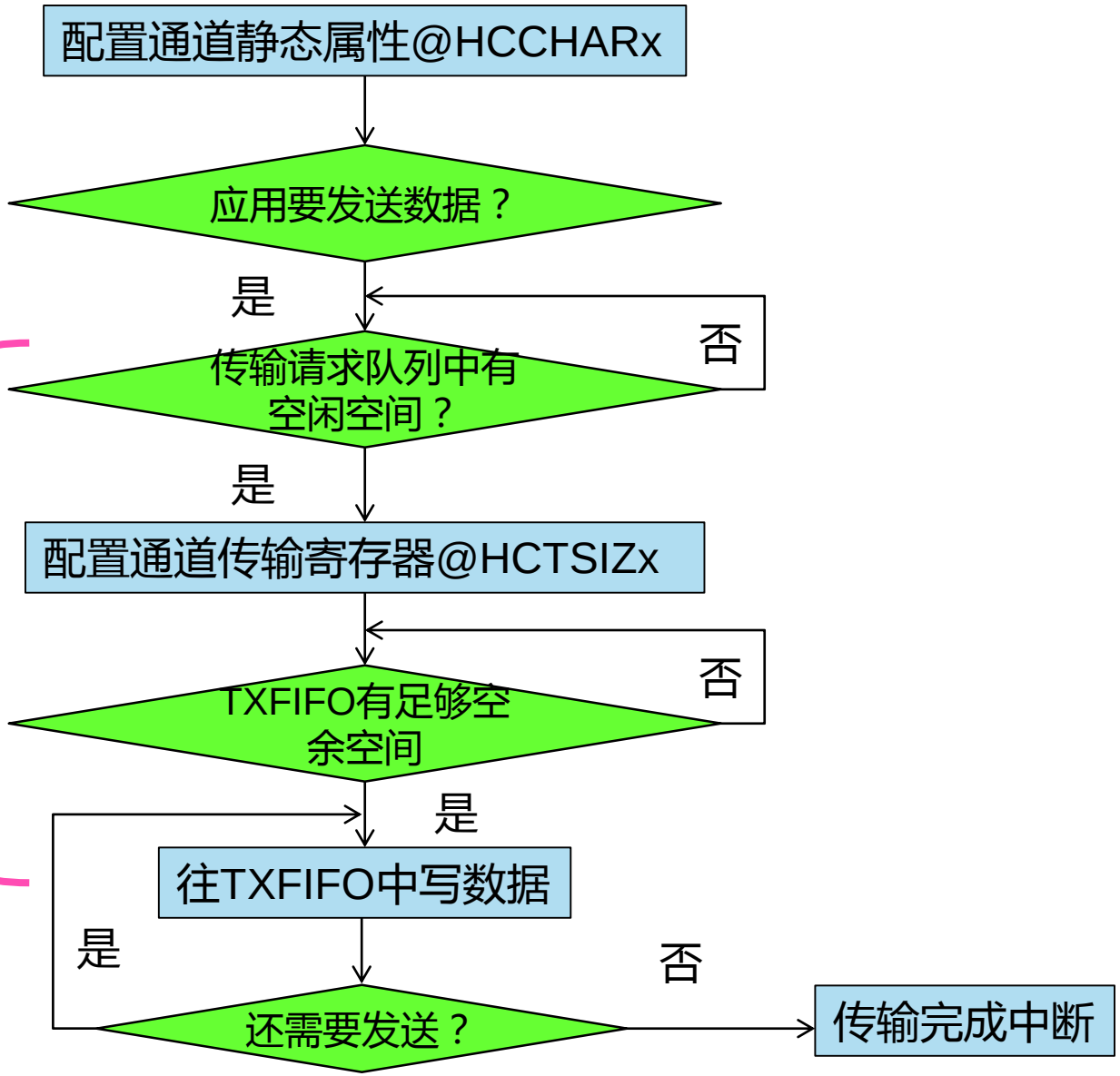
Driver/STM32F4xx_HAL_Driver
<stm32f4xx_hal_hcd.c>
HCD_HC_OUT_IRQHandler (*hhcd, ch_num)
HCD_HC_IN_IRQHandler (*hhcd, ch_num)

主机编程模型.开启一次OUT传输

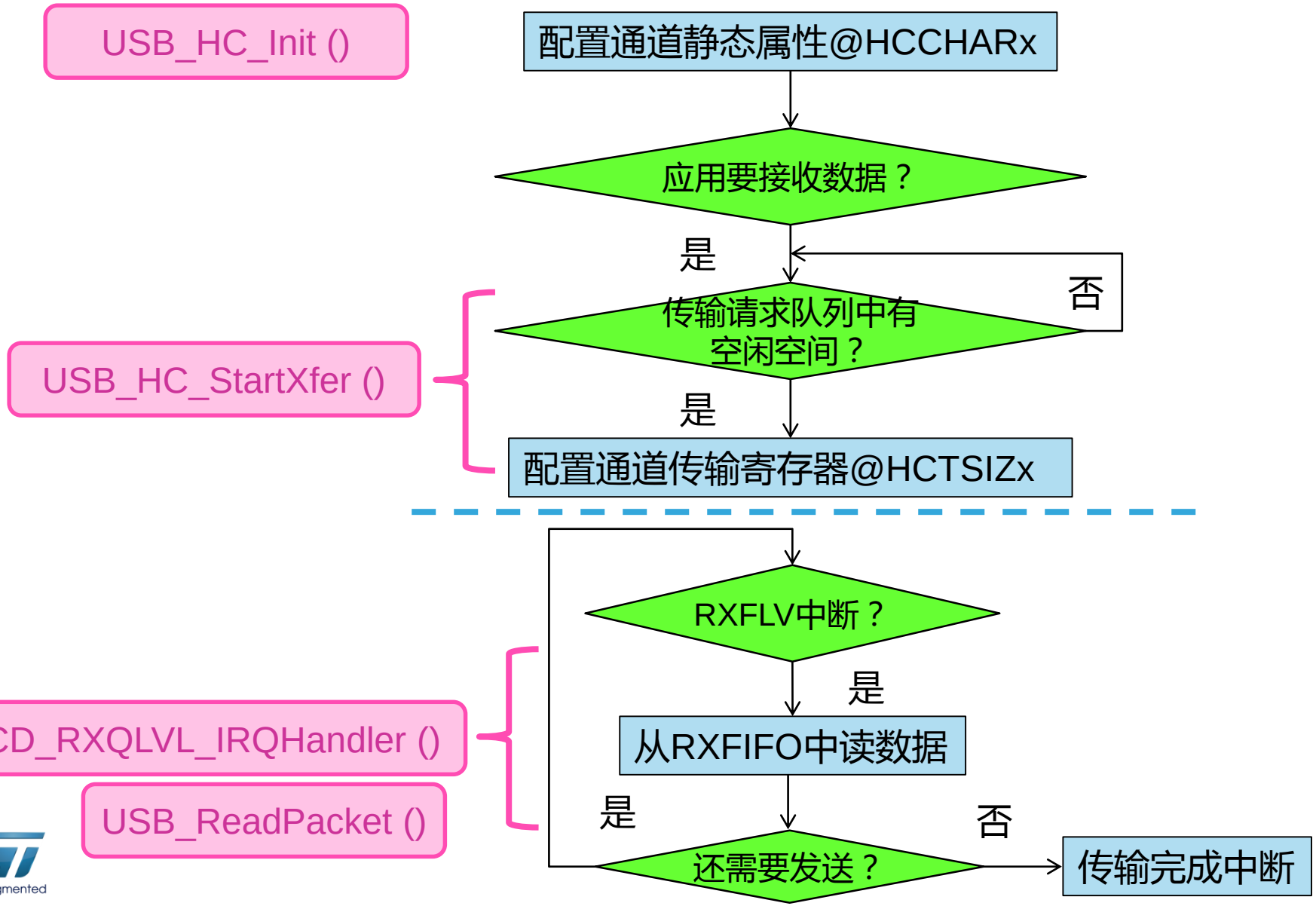
USB_HC_Init ()

USB_HC_StartXfer ()

USB_WritePacket ()



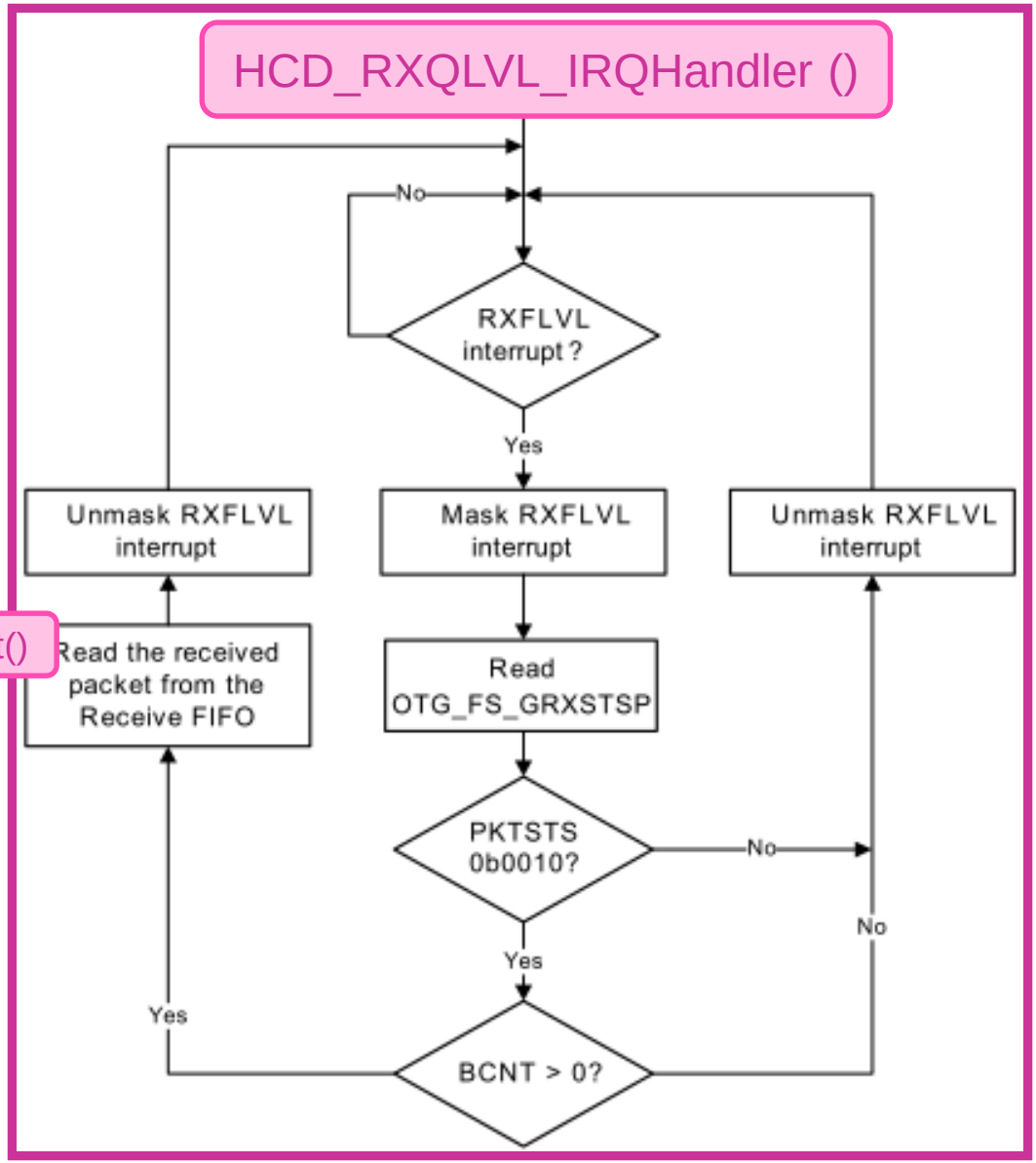
主机编程模型.开启一次IN传输



life.augmented

主机编程模型.读接收FIFO

- 应用收到接收FIFO非空中断
- 读取**GRXSTSP**
  - 是IN数据包？PKSTS=0010
  - 非0数据包？BCNT>0
- 从接收FIFO中读取数据



PKTSTS: Packet status

Indicates the status of the received packet

0010: IN data packet received

0011: IN transfer completed (triggers an interrupt)

0101: Data toggle error (triggers an interrupt)

0111: Channel halted (triggers an interrupt)

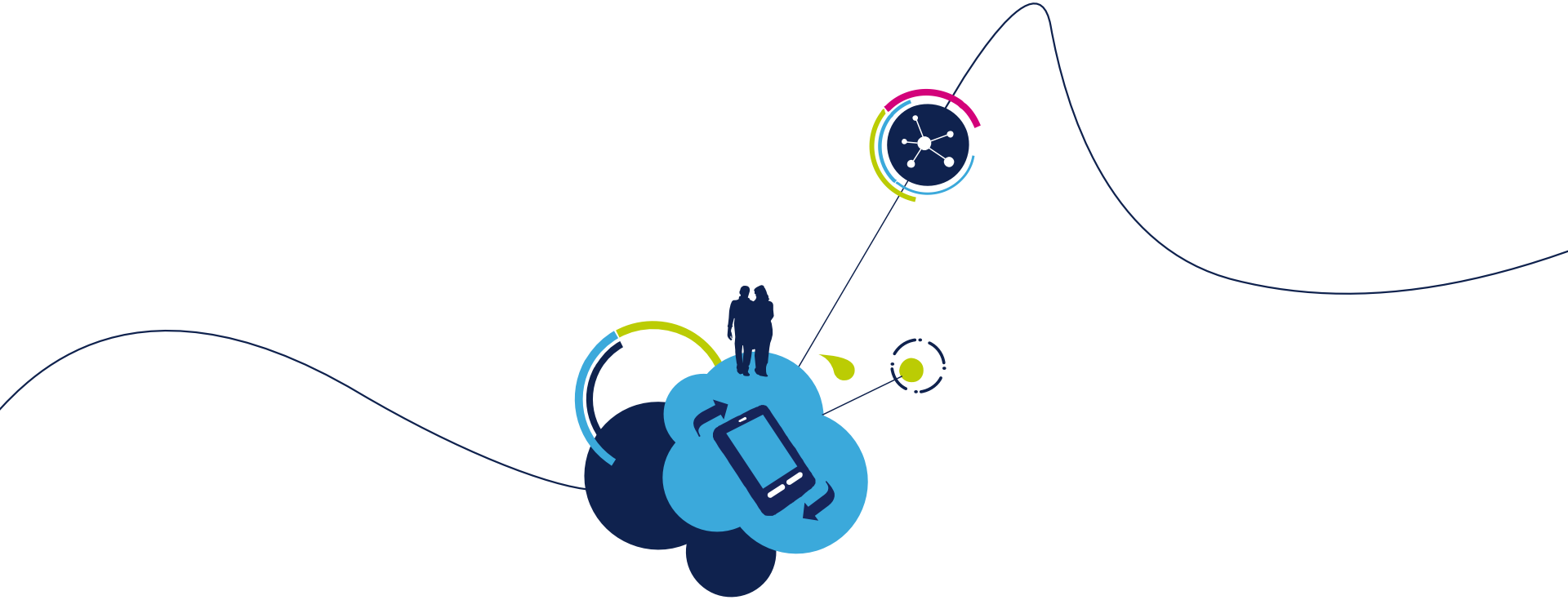
Others: Reserved

BCNT: Byte count

Indicates the byte count of the received IN data packet.

OTG作为**主机**的使用小结

- 典型硬件连接
- 主机状态
  - 端口供电、总线电压有效、枚举，挂起…
- 主机通道即相关寄存器
  - 通道静态属性配置，动态传输配置
- 主机调度器：传输请求队列
- 主机FIFO的使用
- 主机中断
- 主机编程模型
  - 通道的初始化和挂起
  - 主机如何进行OUT传输
  - 主机如何进行IN传输

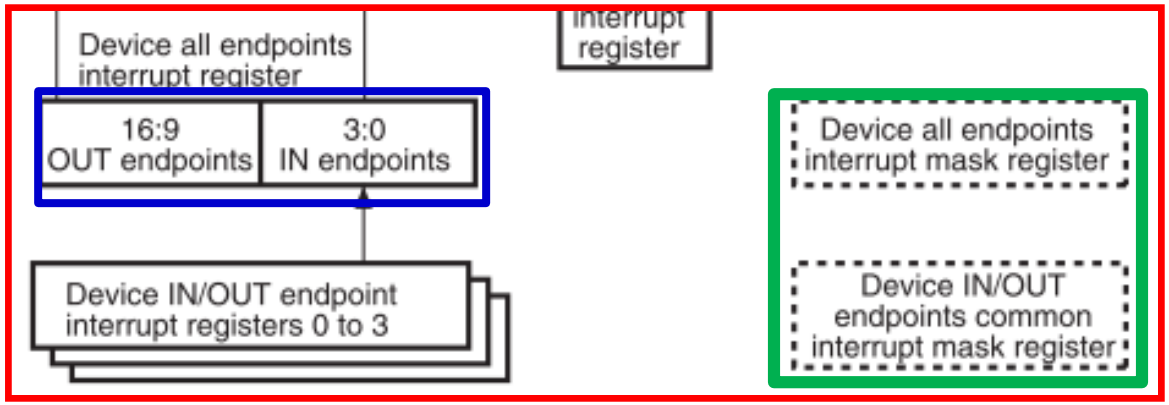


OTG模块的控制和状态寄存器

Core Global	
OTG_FS_GOTG**CTL**	A/BSVLD…DHNPEN, HSHNPEN, SRQ…
OTG_FS_GOTG**INT**	DBCDNE, HNGDET, SEDET,
OTG_FS_G**AHBCFG**	上电或角色切换后要重新配置：PTXFELVL，TXFELVL，**GINTMSK（关全局中断）**
OTG_FS_G**USBCFG**	上电或角色切换后要重新配置：FDMOD，FHMOD，HNPCAP，SRPCAP，TRDT(USB turnaround time)，TOCAL(调整用于**检测Inter packet timeout**的时间，PHY时钟个数)
OTG_FS_G**RSTCTL**	复位core内各个硬件特性：TXFNUM， TXFFLSH，RXFFLSH、FCRST，HSRST、CSRST
OTG_FS_G**INTSTS**	内核全局中断各个标志位
OTG_FS_G**INTMSK**	
OTG_FS_G**RXSTSR** OTG_FS_G**RXSTSP**	接收状态读取和弹出寄存器 Read @ 0x1c; Pop @ 0x20 Read R返回的是接收FIFO顶端的内容；Read P返回接收FIFO顶端的数据项
OTG_FS_G**RXFSIZ**	**接收FIFO大小（单位：字；最大值：256；最小值：16）**
OTG_FS_HNPTXFSIZ OTG_FS_DIEPTXF0	**主机非周期发送FIFO大小和起始地址（同上）** **端点0发送FIFO大小和起始地址（同上）**
OTG_FS_HNPTXSTS	主机非周期发送FIFO和队列状态寄存器
OTG_FS_GCCFG	NOVBUSSENS、SOFOUTEN, VBUSA/BEN, PWRDEN
OTG_FS_CID	
OTG_FS_HPTXFSIZ	**主机周期发送FIFO和大小和起始地址**
OTG_FS_DIEPTXFx (X=1~3)	**端点1~3发送FIFO大小和起始地址**

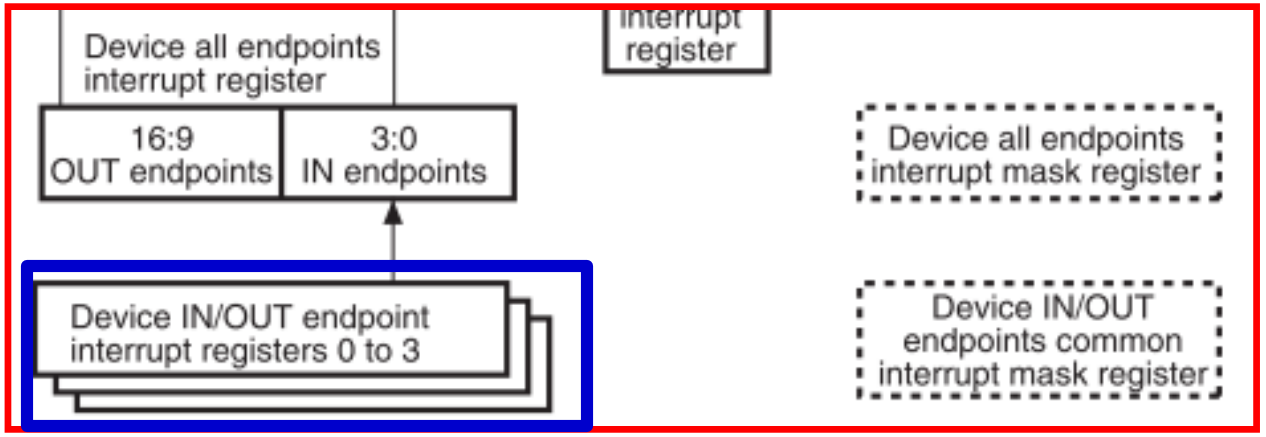
	Host mode Register	
主机其它	OTG_FS_HCFG	FS-LS-support，FS-LS-PHY-clk-select
	OTG_FS_HFIR	Frame间隔寄存器：FRIVL（设置2个SOF之间有多少个PHY CLK）
	OTG_FS_HFNUM	Frame time remaining, frame number
主机队列和FIFO状态	OTG_FS_HPTXSTS	主机周期发送FIFO和队列状态寄存器
主机通道	OTG_FS_HAINT	当HCINT@GINTSTS置位，查看这里看中断是来自哪个通道
	OTG_FS_HAINTMSK	
主机端口	OTG_FS_HPRT	端口速度、D+/D-数据线状态； 端口供电、端口复位、端口挂起，端口恢复、端口使能、端口过流
主机通道	OTG_FS_HCCHARx (X=0~7)	通道使能、关闭； 所连设备地址、端点号、速度、传输类型、传输方向、MPZ
	OTG_FS_HCINTx (x=0~7)	通道上的各自中断标志： 4种error、收到的3种应答、通道挂起(disable)、transfer完成
	OTG_FS_HINTMSKx (x=0~7)	
	OTG_FS_HCTSIZx (x=0~7)	通道上此次传输的大小（几个字节，几个数据包）

	Device mode Register –group1	
设备其他	OTG_FS_DCFG	设备地址、应用期望的速度；帧间隔、NZL-SO-HSK
	OTG_FS_DCTL	远程唤醒操作、软件断开操作、各种握手应答的回复…
	OTG_FS_DSTS	设备枚举出的速度、进入挂起状态、收到的SOF帧号；…
设备端点、发送FIFO中断	OTG_FS_DIEP**MSK**	
	OTG_FS_DOEP**MSK**	
	OTG_FS_DAINT	
	OTG_FS_DAINT**MSK**	
	OTG_FS_DIEPEMP**MSK**	IN端点对应每个TXFIFO的空中断屏蔽控制
设备发送FIFO状态	OTG_FS_DTXFSTSx,0~3	IN端点对应的每个TXFIFO空间的可用空间查询
OTG/SRP操作	OTG_FS_DVBUSDIS	SRP过程中用到…
	OTG_FS_DVBUSPULSE	SRP过程中用到…



life.augmented

	Device mode Register –group2	
设备端点控制	OTG_FS_DIEPCTL0	
	OTG_FS_DIEPCTLx, 1~3	
	OTG_FS_DOEPCTL0	
	OTG_FS_DOEPCTLx,1~3	
设备端点中断	OTG_FS_DIEPINTx, 1~3	
	OTG_FS_DOEPINTx, 1~3	
设备端点传输尺寸	OTG_FS_DIEPTSIZ0	
	OTG_FS_DOEPTSIZ0	
	OTG_FS_DIEPTSIZx, 1~3	
	OTG_FS_DOEPTSIZx,1~3	



life.augmented

Feature HS Vs. FS

	OTG HS IP	OTG FS IP
设备	HS/FS	FS
主机	HS/FS/LS	FS/LS
OTG	FS	
Internal DMA	模块内部集成	X
FIFO大小	4KB	1.25KB
通道数目	12	8
Build-in scheduler	2 queue, each hold up to 8 entries	
端点数目	1(双向EP0) + 5*2	1(双向EP0) + 3*2
设备端的软断开	Y	
全速通信PHY	模块内部已集成	
高速通信PHY	模块集成ULPI接口	X
AHB最低工作频率	30MHz	14.2MHz

端点中断状态寄存器

DIEPINTx (x=0…3)

位	域名		
13	NAK	设备发出或者收到NAK； 【ISO IN EP】由于TX FIFO空间不够，只能发了一个0长度数据包	OTG HS模块新增的中断标志位域
12	BEER		
11	PKTDRPSTS	表示应用丢失一个ISO OUT包；没有中断屏蔽位，也不产生中断	
9	BNA	要访问的描述符没有就绪（e.g. host busy or DMA done）???	
8	TXFIFOUNDRN	IN端点的TX FIFO下溢（使能thresholding时才有效）	
7	TXFE	该端点的发送FIFO空	应用需要先读取OTG_FS_DAINT来知晓发送中断的是哪个端点；再到对应的DIEPINTX来查看具体中断事件
6	INEPNE	该端点上的NAK位生效了	
4	ITTXFE	发送FIFO空，却收到了IN令牌	
3	TOC	**控制类型IN端点上**检测到超时(从上一个收到的IN令牌开始，发送超时了)	
1	EPDISD	端点被应用关闭	
0	XFRC	**该端点上的传输成功结束**	

端点中断状态寄存器

DOEPINTx (x=0…3)

位	域名		
14	NYET	**非同步OUT端点上**发出一个NYET应答	OTG HS新增
6	B2BSTUP	**EP0的OUT端点上**收到3个以上的背靠背SETUP包	应用需要先读取OTG_FS_DAINT来知晓发送中断的是哪个端点；再到对应的DIEPINTX来查看具体中断事件
4	OTEPDIS	**EP0的OUT端点上**收到OUT令牌，但是端点还未使能	
3	STUP	**控制类型OUT端点上**的SETUP阶段结束了，且没有背靠背的SETUP包，应用可以解码SETUP中的数据内容了	
1	EPDISD	端点被应用关闭	
0	**XFRC**	**该端点上的传输成功结束**	