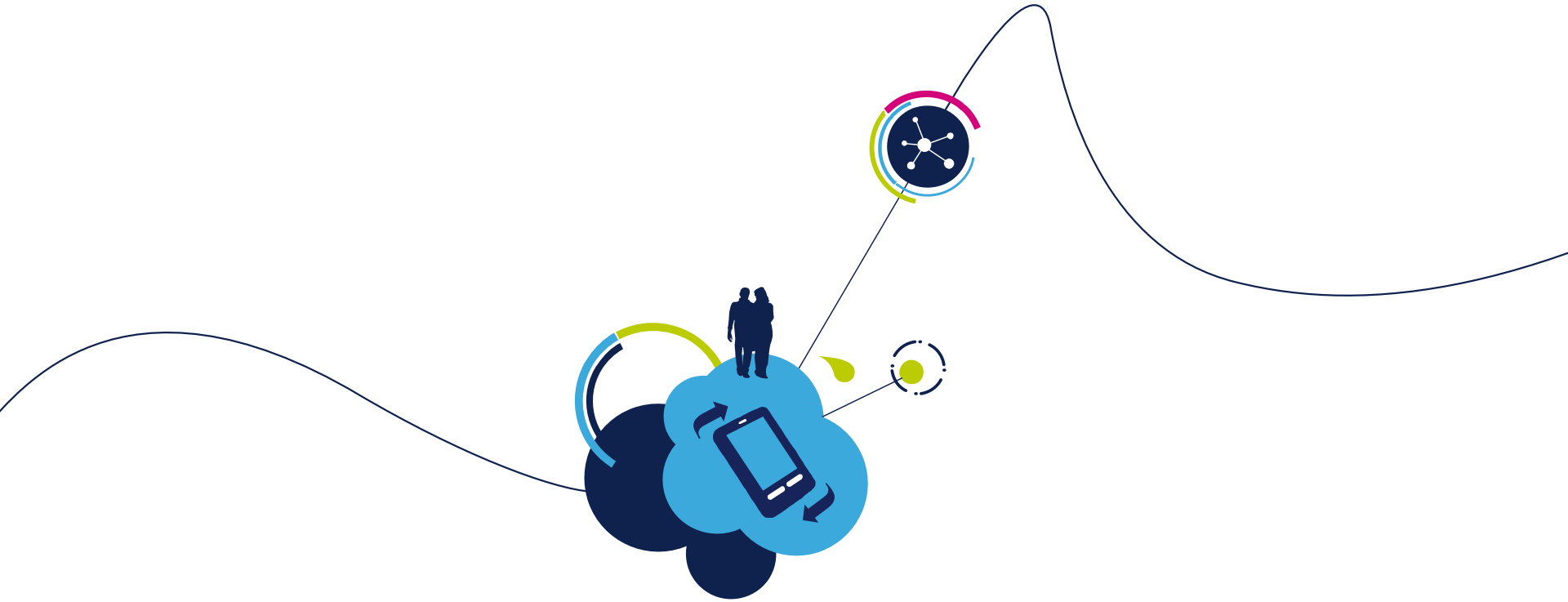


OTG模块作为USB主机

life.augmented

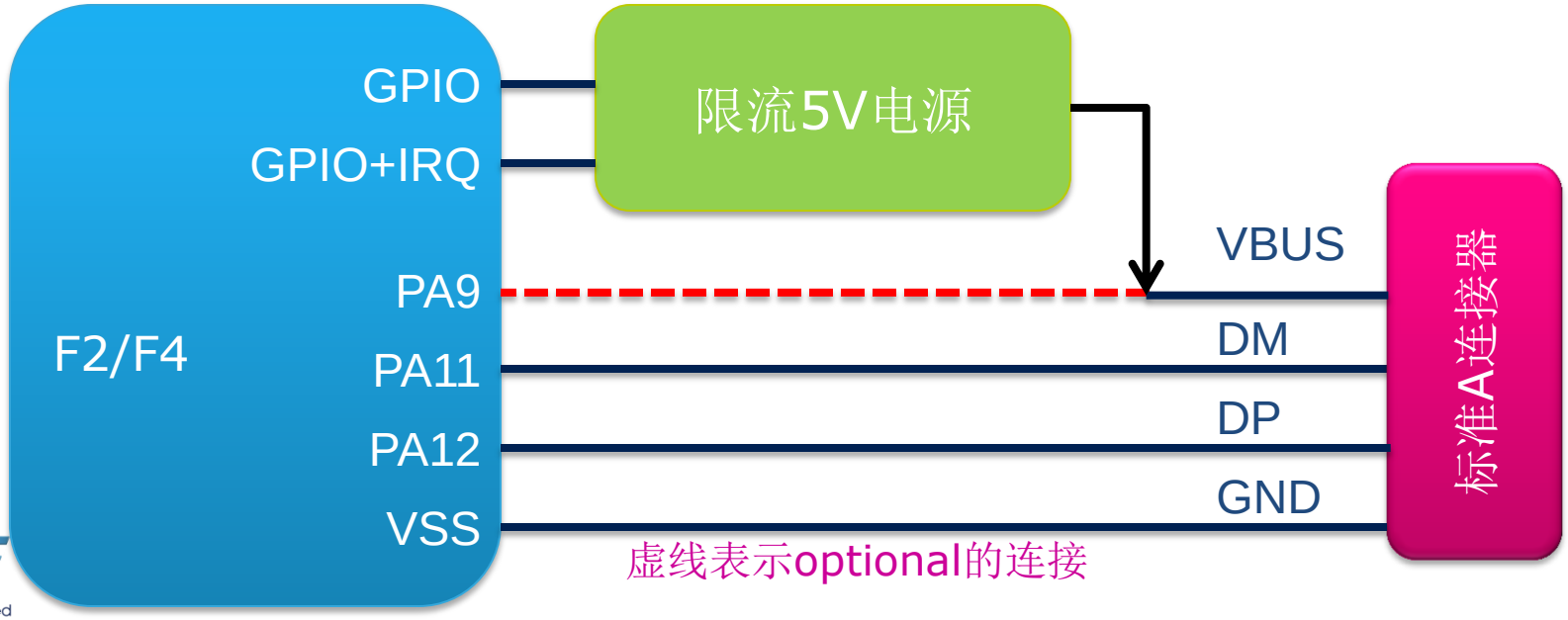
作为USB主机的四种情形

- 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

life.augmented

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)



life.augmented

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
  - 设备断开：**触发中断**：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（ PCDET@HPRT ）
  - 设备断开：**触发中断** @ DISCINT@GINTSTS

life.augmented

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_wq）
  - 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

	bit5	POC CHNG	Interrupt	端口过流变化	Bit4变化会置位该位
OTG_FS_HPRT	Bit4	POC A	r	端口是否过流	
	Bit3	PEN CHNG	Interrupt	端口使能变化	Bit2变化会置位该位
	Bit2	PEN A	r	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

life.augmented

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)

life.augmented

主机通道及相关寄存器

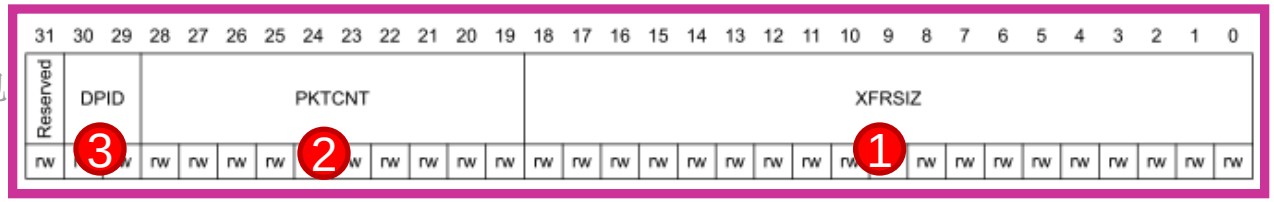
- OTG_FS模块有8个主机通道
  - 每个通道有各自的【通道控制】、【传输配置】、【状态/中断】、【掩码】寄存器
- 通道属性寄存器：HCCHARx
  - 通道enable/disable
  - 与该通道所连接的USB设备的速度FS/LS
  - 与该通道所连接的USB设备的地址
  - 与该通道所连接的USB设备上的端点号
  - 通道传输方向IN/OUT
  - 通道传输类型CTL/BLK/INT/ISO
  - 通道最大包长MPS
  - 黄色位：只针对周期传输（INT、ISO）有效

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

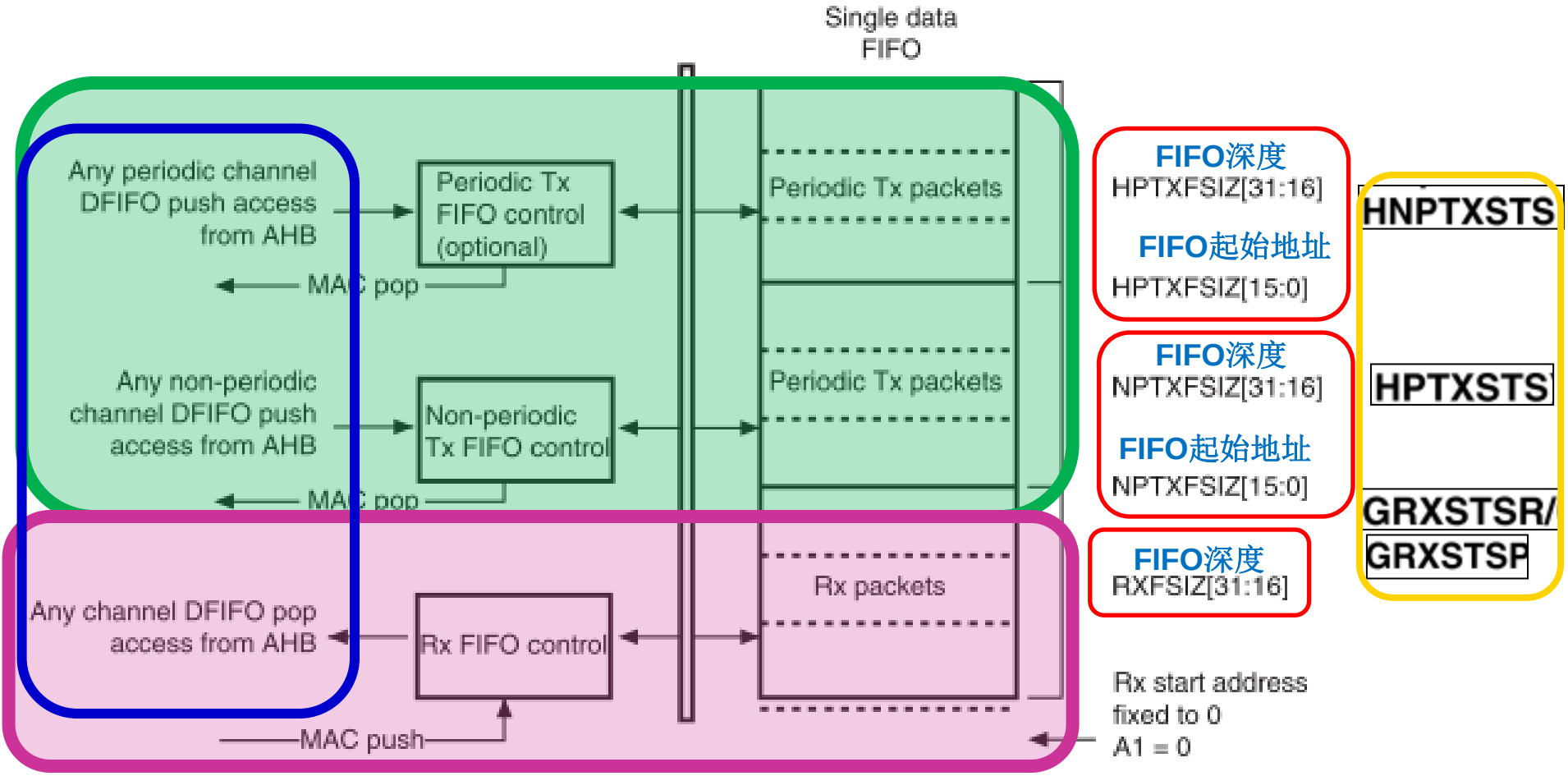
life.augmented

主机调度器

- 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
  - 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
    - 非/周期传输发送FIFO和队列状态寄存器： HNPTXSTS / HPTXSTS
    - NP TQX SAV：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）
    - NP TXF SAV：如果要发起OUT传输，非周期发送FIFO中有几个空闲word
    - NP TXQ TOP：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

life.augmented

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 >> 收到的数据包的目标通道	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** >> 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)	
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）

life.augmented

CDC Host Demo.设置FIFO大小

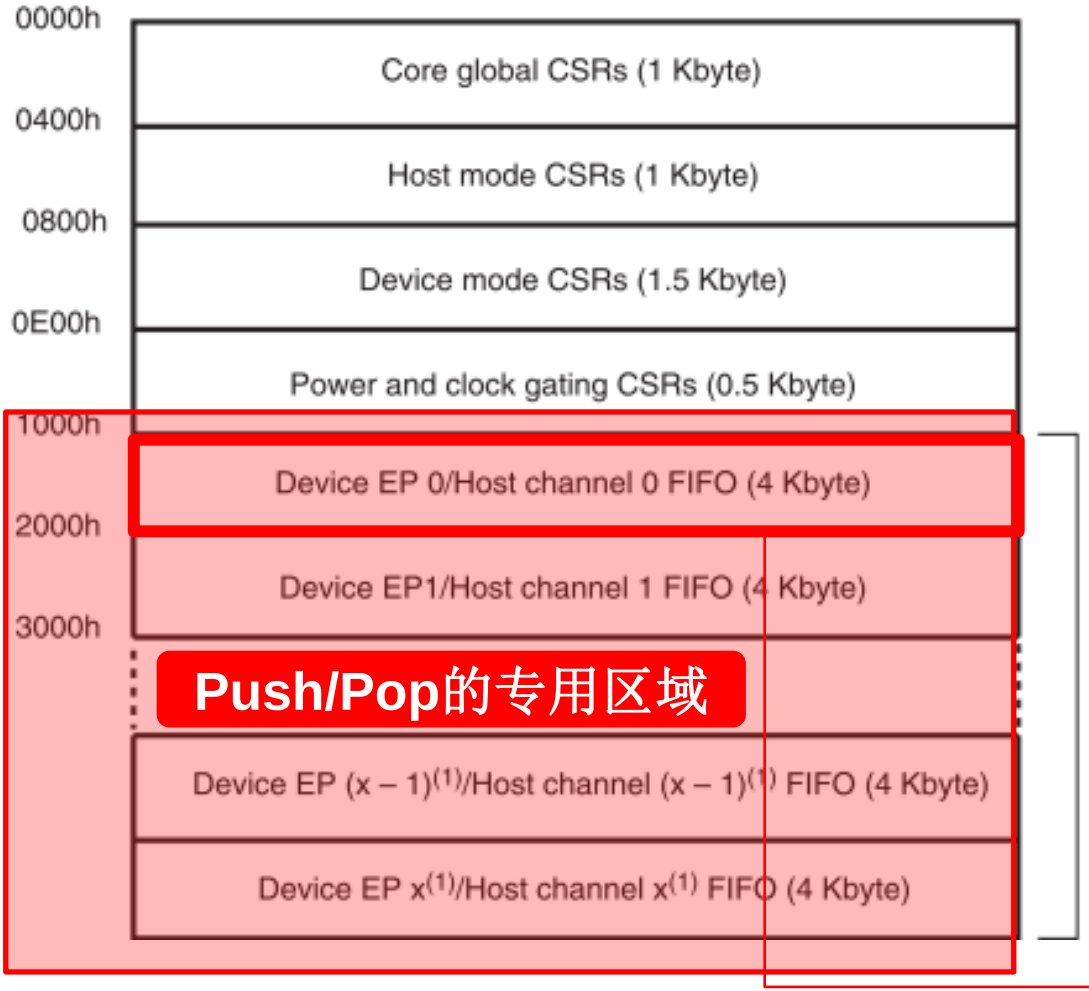
- USB_HostInit () ← HAL_HCD_Init() ← USBH_LL_Init() ← USBH_Init()

	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)
```

life.augmented

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_**HPRT**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】

life.augmented

主机通道中断标志HCINTx (x=0…7)

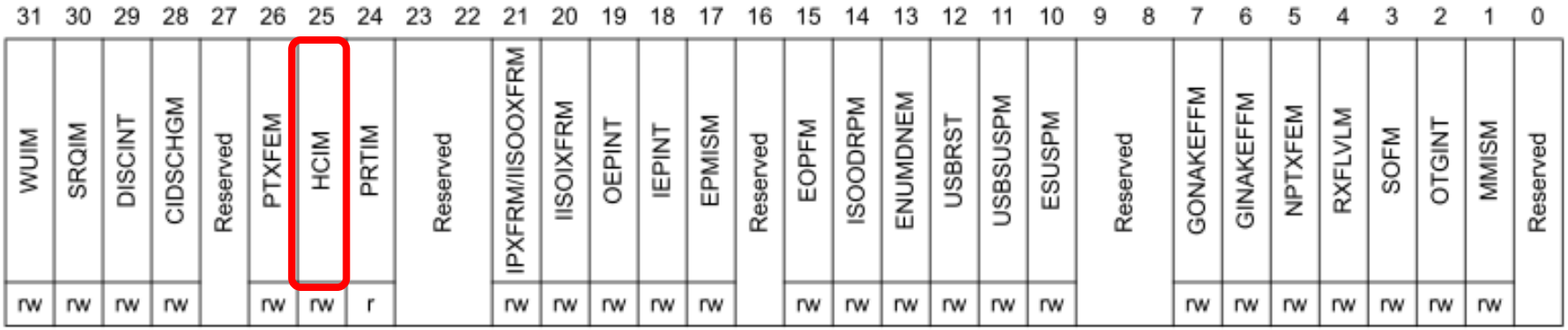
位	域名		备注	HC_IN_IRQ	HC_OUT_IRQ
10	DTERR	数据toggle错误	应用需要先读取OTG_FS_HAINT来知晓发送中断的是哪个通道		
9	FRMOR	Frame溢出错误			
8	BBERR	Babble错误		X	X
7	TXERR	传输错误 >> CRC校验失败 >> **超时** >> 比特填充错误 >> 假的EOP			
6	NYET	Stm32f407xx.h		X	
5	ACK	发出或者收到ACK应答			
4	NAK	收到NAK应答			
3	STALL	收到STALL应答			
2	AHBERR	Stm32f407xx.h			
1	CHH	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**			

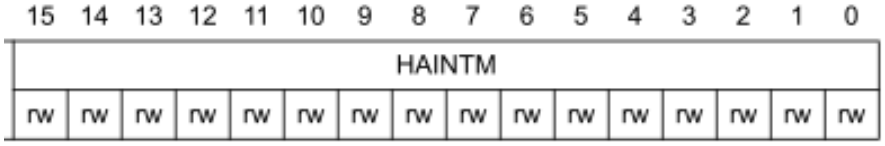
主机端口中断标志HPRT

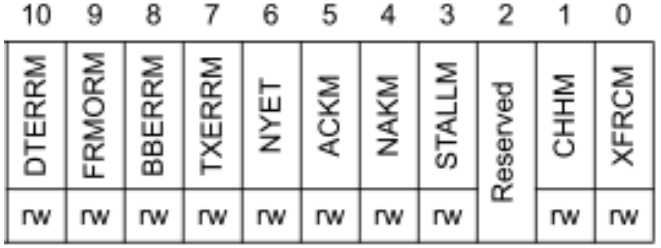
位	域名			Cube库中处理
5	P OC CHNG	端口过流标志改变	rc_w1，可产生中断	Y
4	P OC A	端口过流	r	
3	P EN CHNG	端口使能标志改变	rc_w1，可产生中断	Y
2	P EN A	端口使能	r	
1	P C DET	检测到端口有连接	rc_w1，可产生中断	Y
0	P C STS	端口连接状态	r	

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

life.augmented

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_ll_usb.c>
USB_HC_Halt
(USB_OTG_GlobalTypeDef *USBx,
uint8_t ch_num)

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

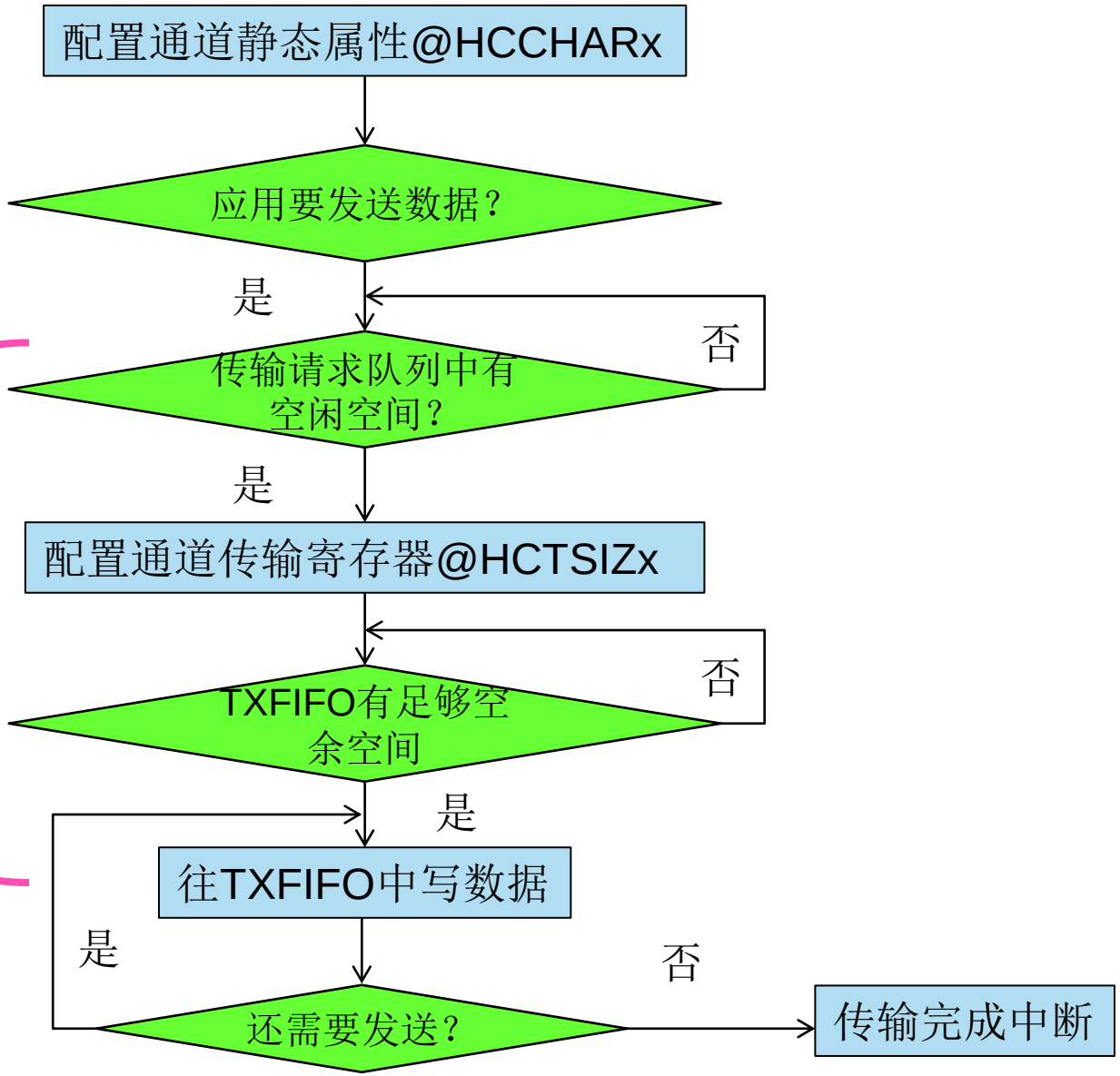
life.augmented

主机编程模型.开启一次OUT传输

USB_HC_Init ()

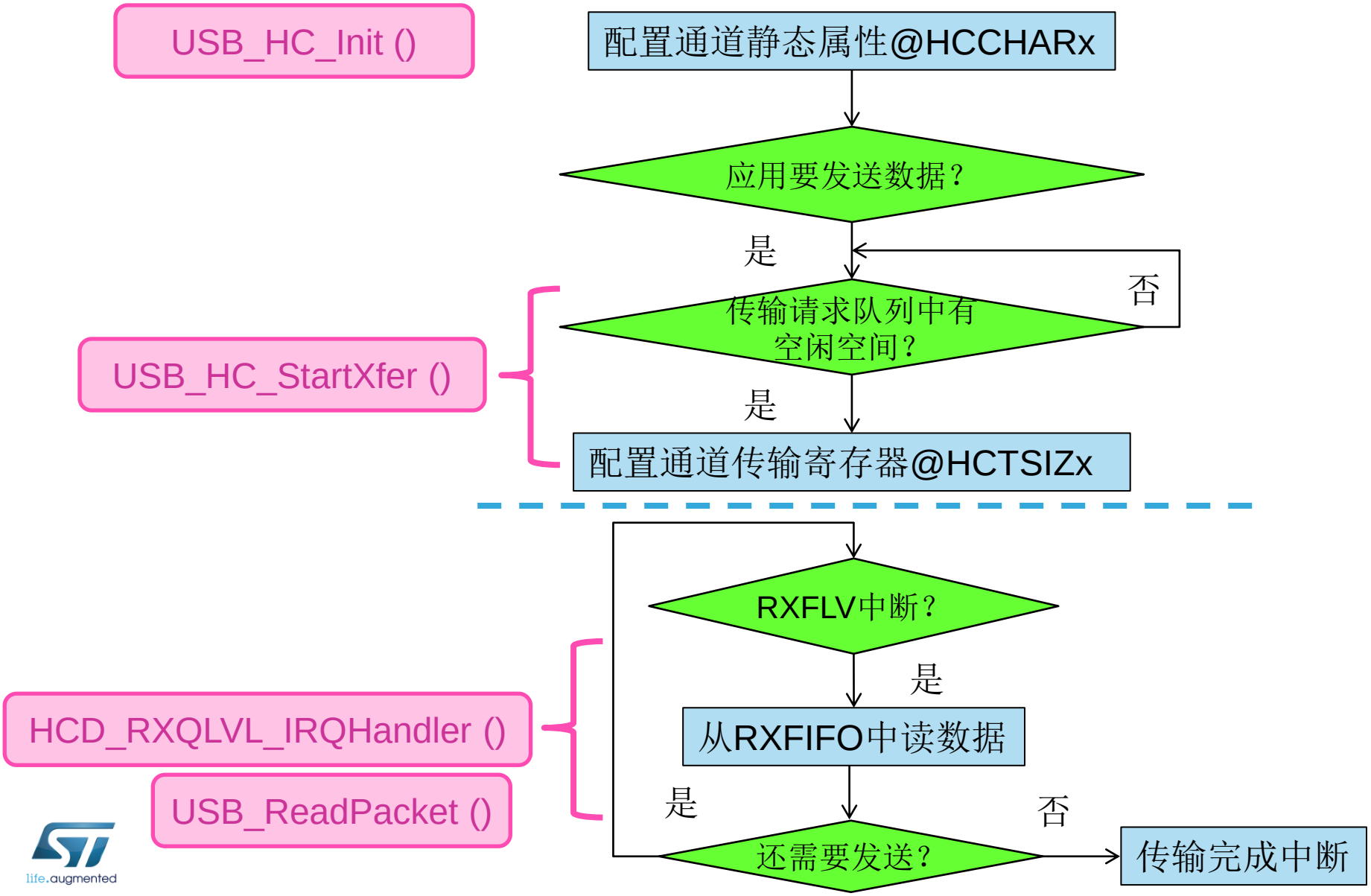
USB_HC_StartXfer ()

USB_WritePacket ()



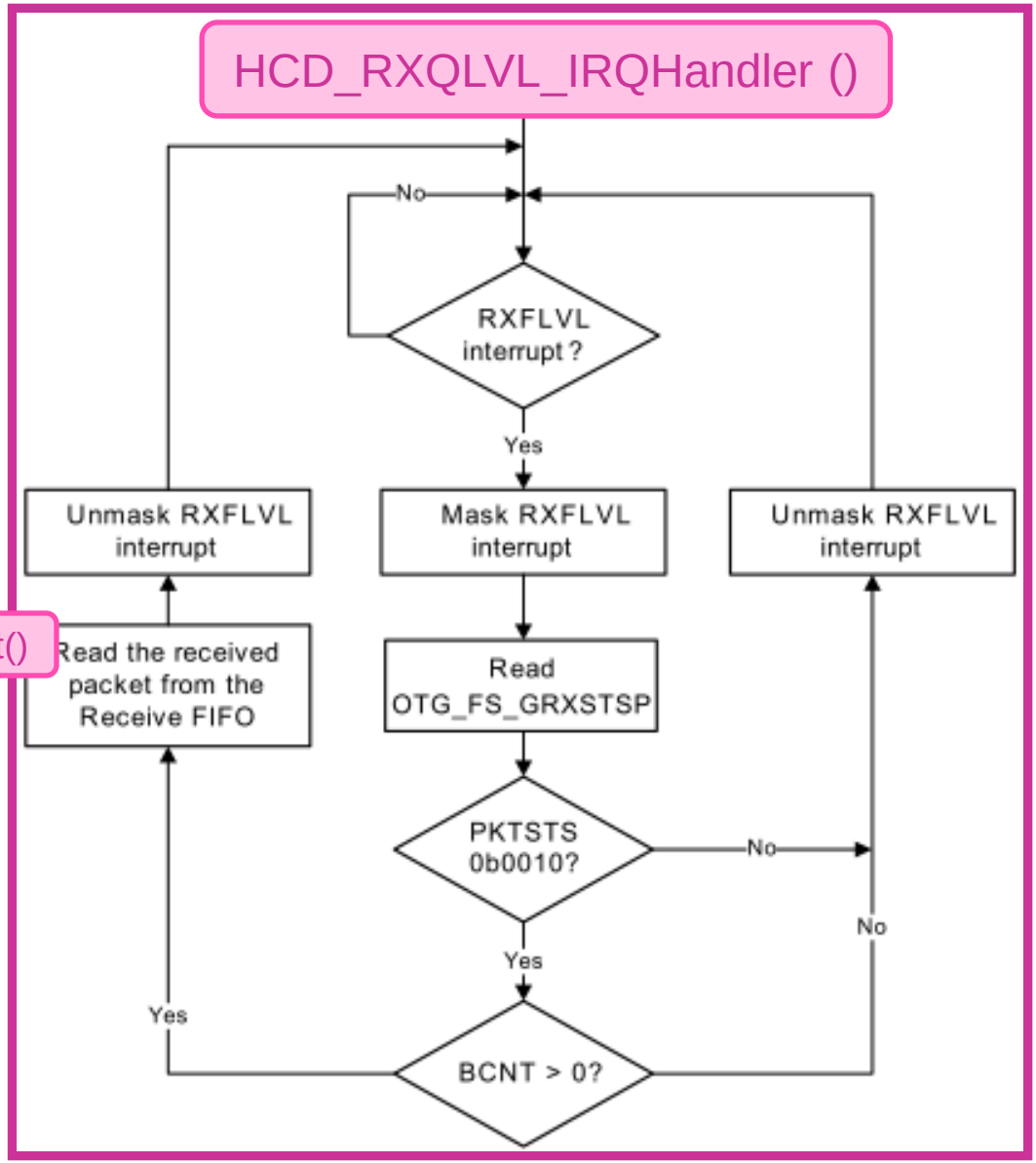
life.augmented

主机编程模型.开启一次IN传输



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

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.

Information furnished is believed to be accurate and reliable. However, STMicroelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of STMicroelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. STMicroelectronics products are not authorized for use as critical components in life support devices or systems without express written approval of STMicroelectronics.

The ST logo is registered trademark of STMicroelectronics

All other names are the property of their respective owners

© 2015 STMicroelectronics - All Rights Reserved

STMicroelectronics group of companies Australia - Brazil - Canada - China - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States.

www.st.com

life.augmented