通用串行总线(USB) 2.0

2010

STM32F10x USB 开发者培训

2010年7月

内容

- **USB技术简介**
  - 技术背景
  - 架构、系统、协议和供电
  - 设备的枚举、识别
  - 传输类型

- **STM32 USB模块和函数库**
  - 模块的特性
  - 各类描述符解析
  - 模块的中断源及相关的中断处理函数
  - 模块的其他相关函数库

技术背景(1)

- 通用串行总线(USB)协议的设计目标：
  - 易于使用的PC外设
  - 以低成本的方案支持高达480Mbps的传输速率
  - 满足声音，音频和视频类传输的实时需求
  - 灵活的协议，能混合同步和异步的消息数据传递
  - 集成商品设备技术
  - 能适应于任意外形和配置的PC
  - 提供一个标准接口，能快速应用于产品中
  - 允许扩展出新的USB设备类，以提升PC的功能
  - UBS2.0协议必需向下兼容，以容纳早期版本的设备

技术背景(2)

应用范围

性能	应用	特性
低速: ✓交互式设备 ✓**10-100kbps**	➢键盘，鼠标 ➢手写笔 ➢游戏手柄 ➢虚拟设备 ➢外设	•极低的成本 •易于使用 •热插拔 •同时使用多个外设
全速: ✓电话，音频类 ✓压缩的视频类 ✓**500kbps – 10Mbps**	➢话音 ➢宽带 ➢音频 ➢麦克风	•较低的成本 •易于使用 •热插拔 •同时使用多个外设 •可保证的带宽 •可保证的延迟
高速: ✓视频，大容量存储 ✓**25 – 400Mbps**	➢视频 ➢大容量存储 ➢图像 ➢宽带	•低成本 •易于使用 •热插拔 •同时使用多个设备 •可保证的带宽 •可保证的延迟 •高带宽

技术背景(3)

特性：

简单易用　　使用统一制式的电缆和连接插座，支持热插拔。

应用广泛　　支持从几kbps到几百Mbps的数据带宽，支持同步和异步的传输，支持连接多达127个设备，并支持复合设备。

同步带宽　　提供保证的带宽和低延迟。

使用灵活　　支持不同大小的数据包和各种传输速率。

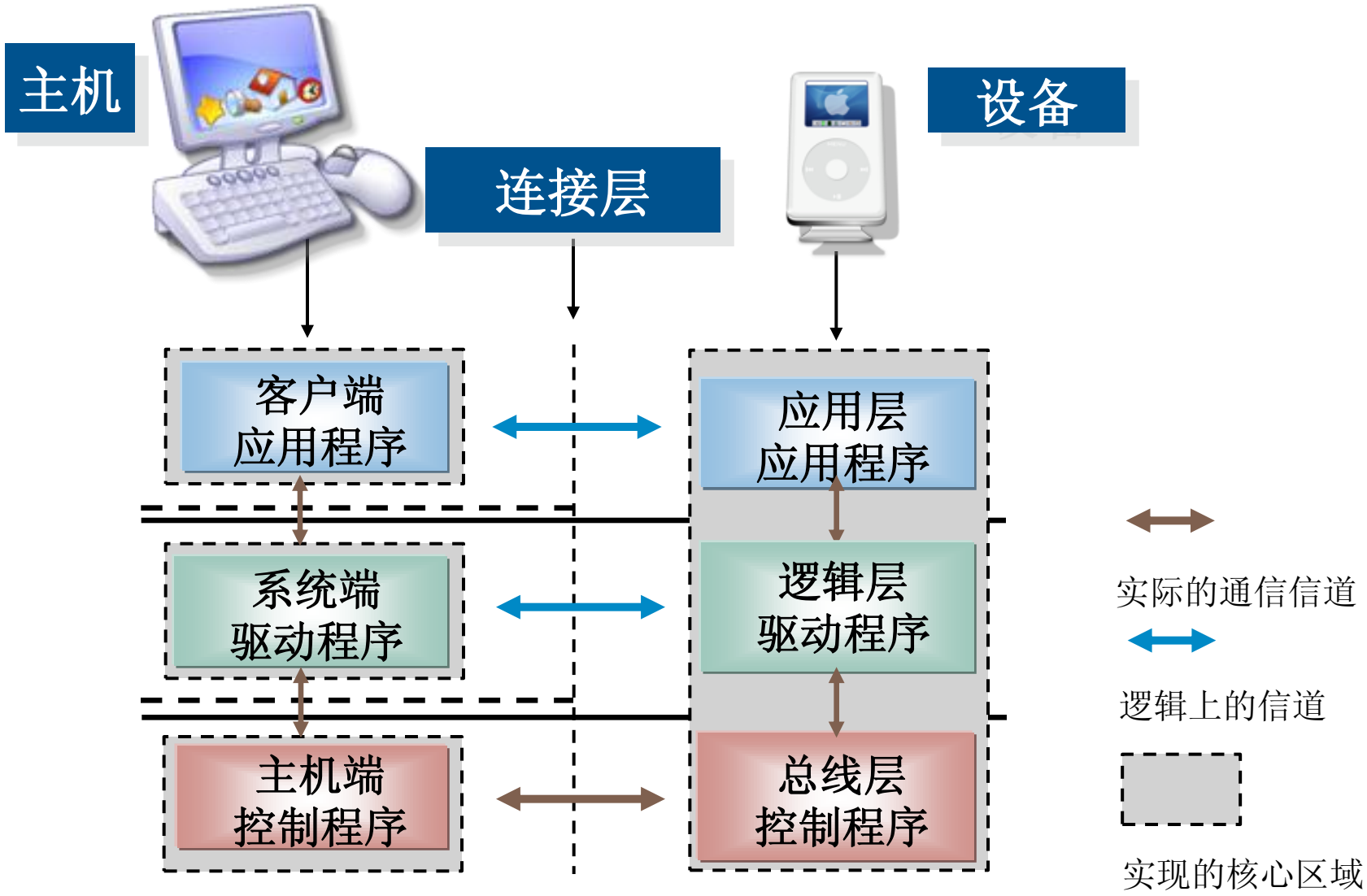
鲁棒性佳　　多种的错误校验和恢复机制。

协同PC产业　协议易于实现和整合，并支持热插拔机制

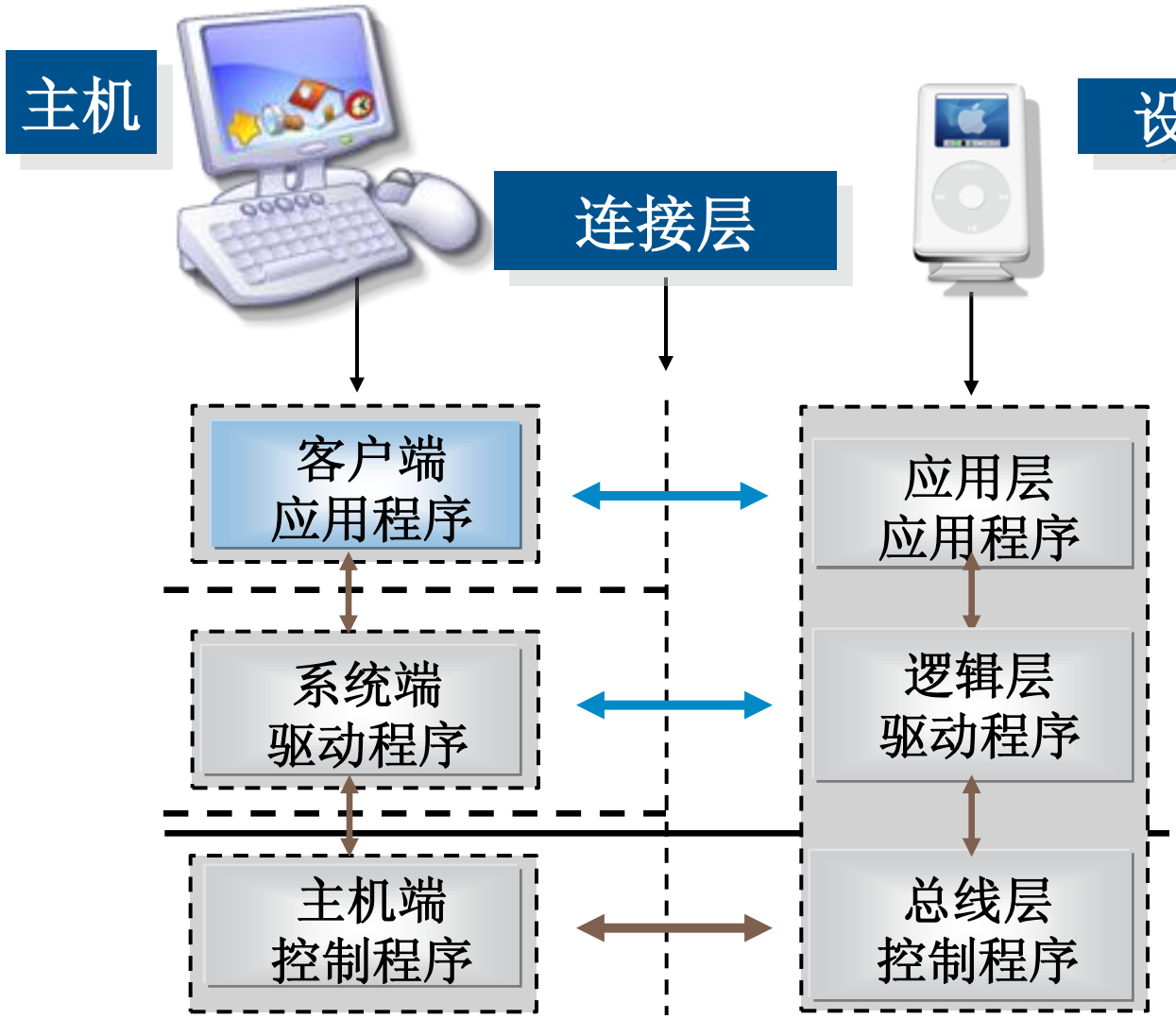
低成本实现　低成本的电缆和连接插座，商品化的实现技术

易于升级　　整体结构易于升级，能适应各类新生的应用。

USB 系统 (1)


主机
设备
连接层
客户端
应用程序
系统端
驱动程序
主机端
控制程序
应用层
应用程序
逻辑层
驱动程序
总线层
控制程序
实际的通信信道
逻辑上的信道
实现的核心区域

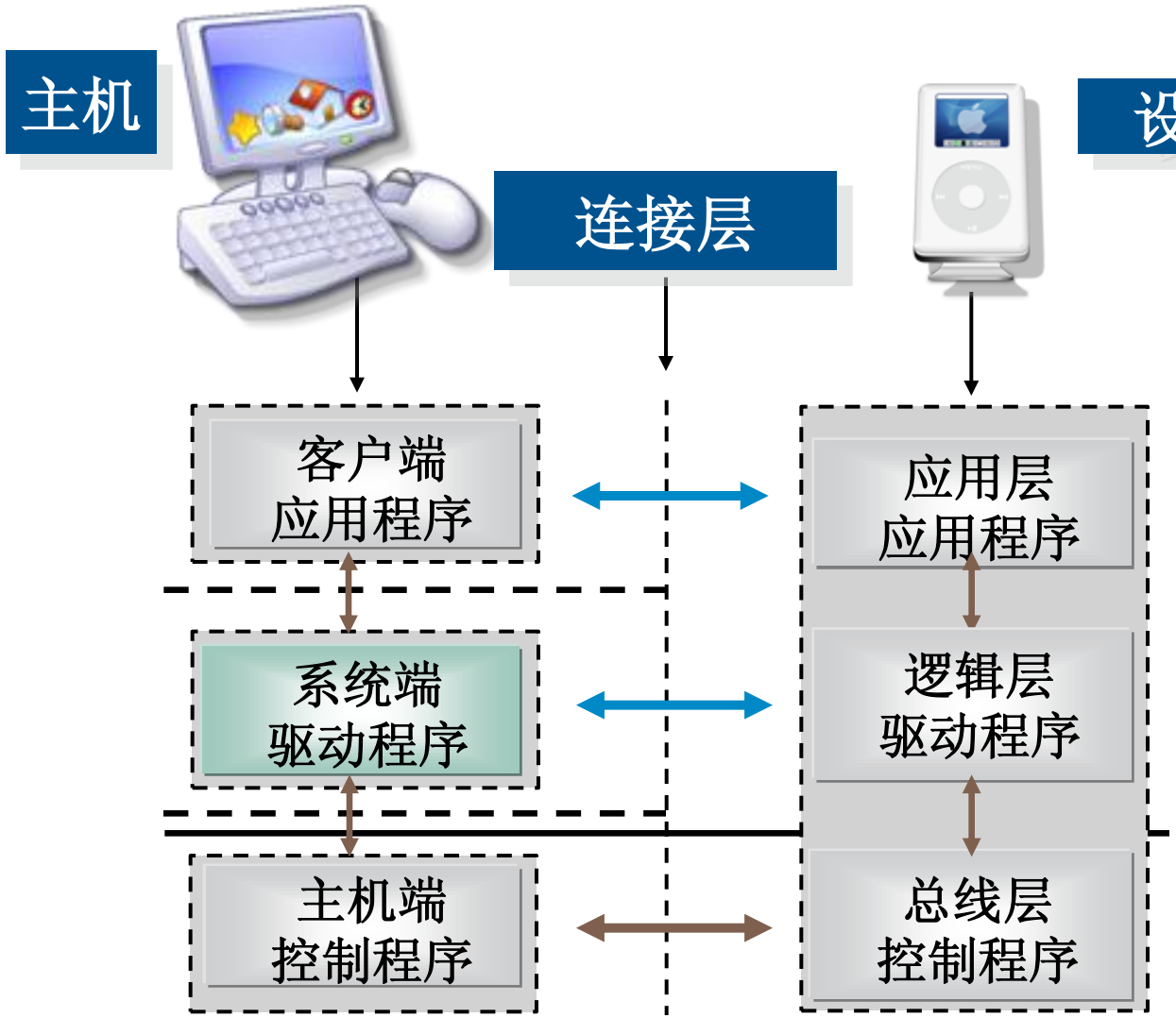
USB 系统 (2)



应用程序:

应用程序运行于USB设备所连接的主机上。部分应用程序会由主机的操作系统提供，大部分的应用程序由USB设备的供应商提供。

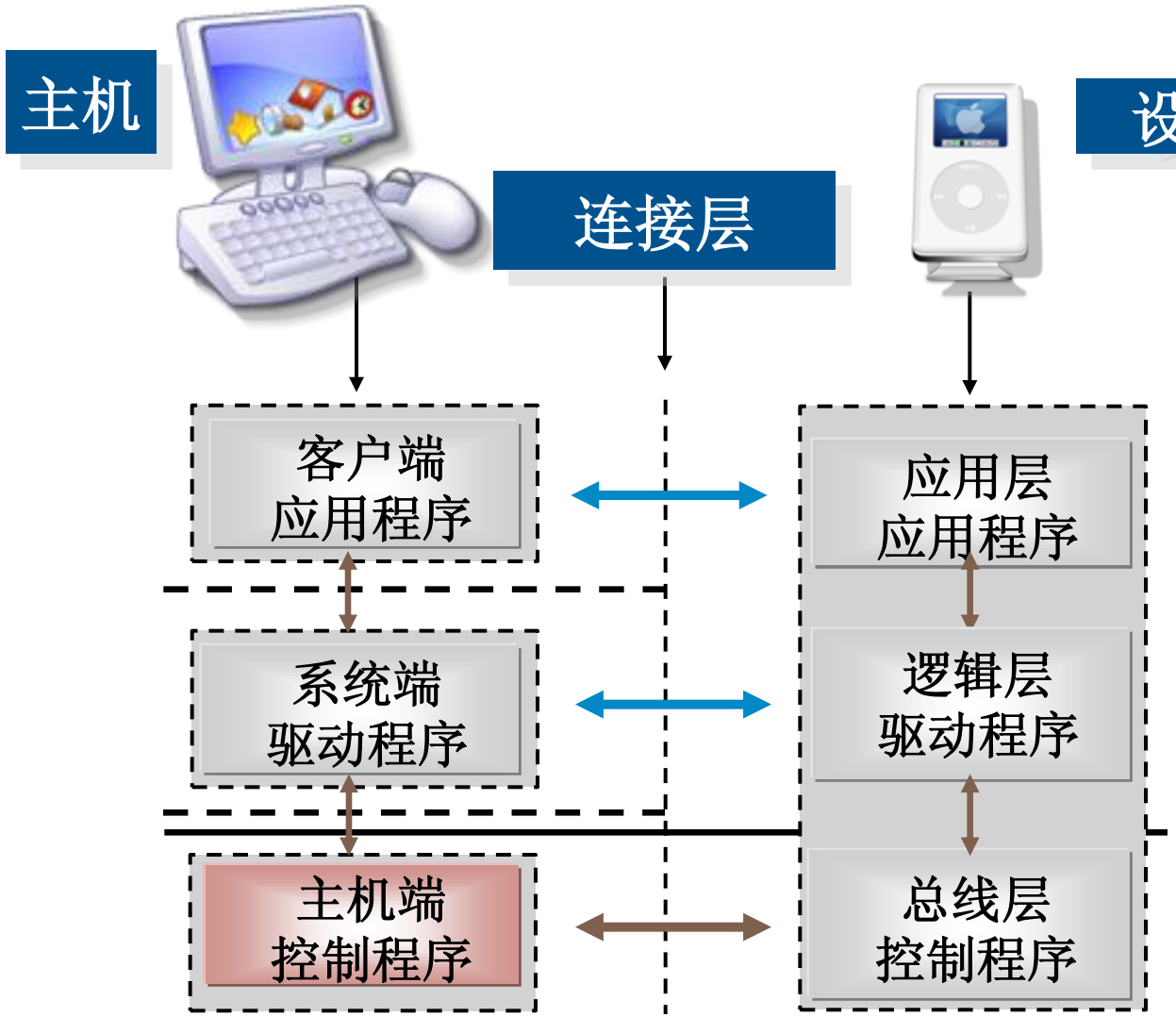
USB 系统 (3)



驱动程序:

驱动程序服务于特定的操作系统，为USB设备提供支持。
驱动程序通常由操作系统提供，独立于特定的应用程序和USB设备。

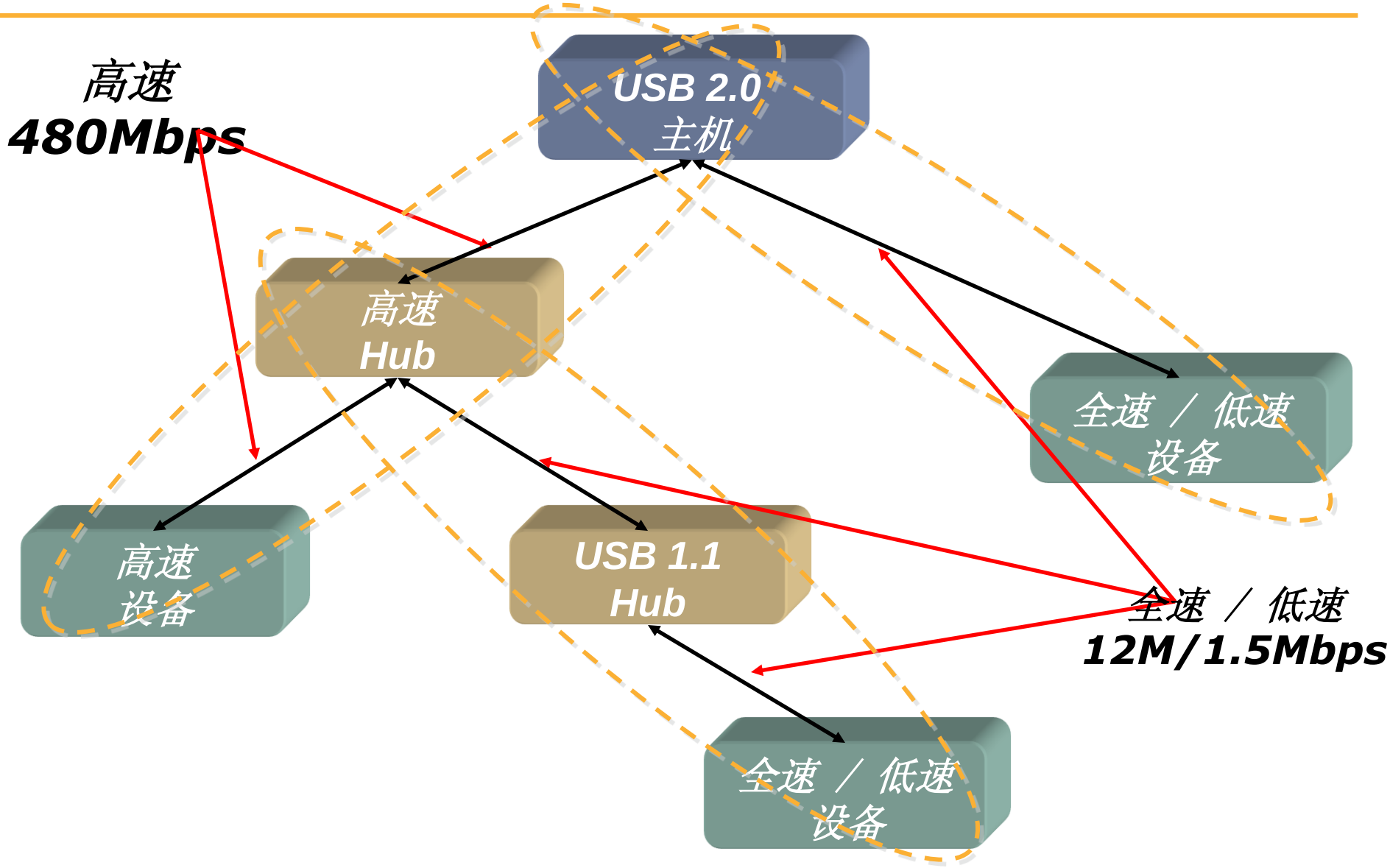
USB 系统 (4)



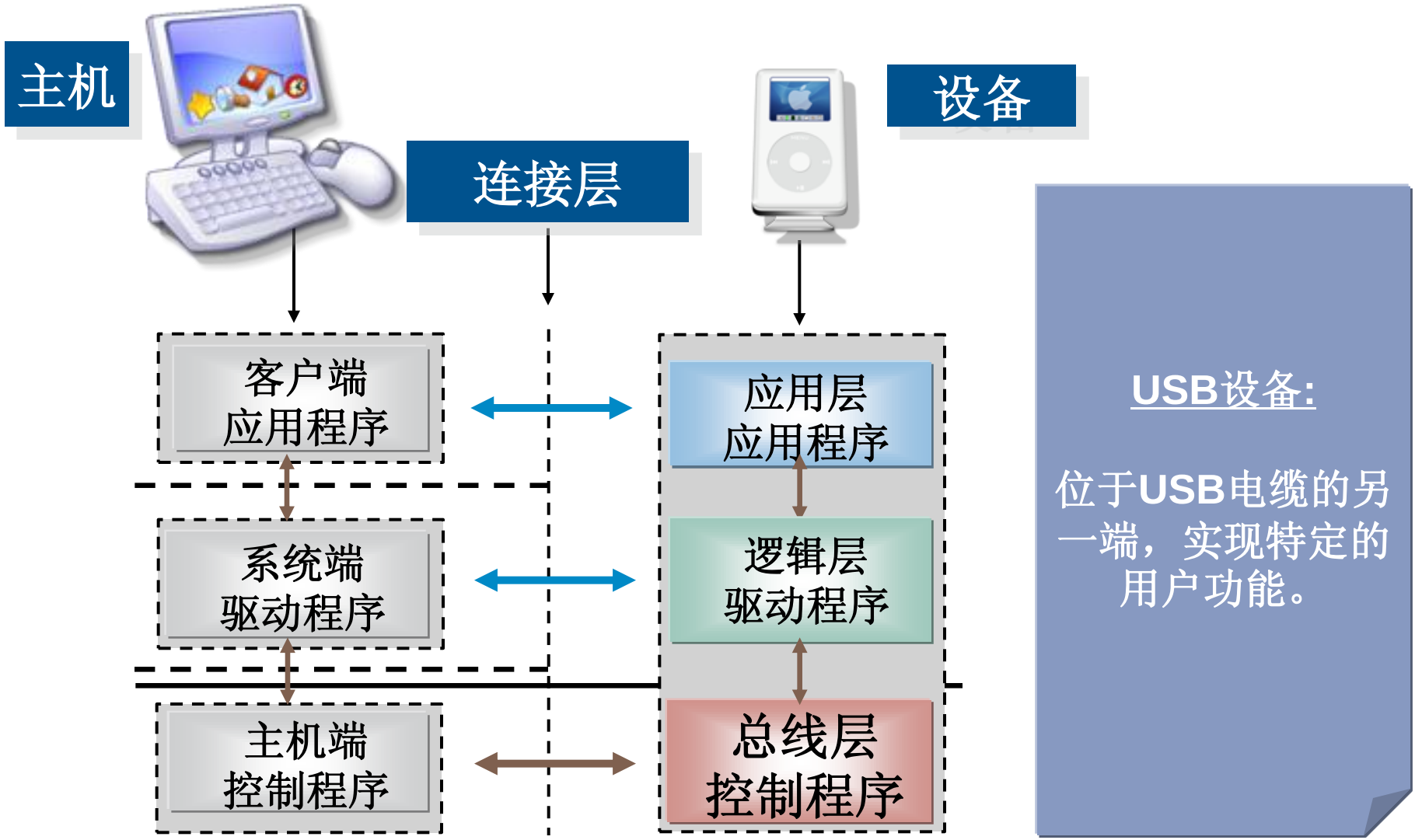
主机端控制程序:

包括硬件和软件，使USB设备能够被主机识别。

USB 拓扑结构


高速
480Mbps
USB 2.0
主机
高速
Hub
全速 / 低速
设备
高速
设备
USB 1.1
Hub
全速 / 低速
12M/1.5Mbps
全速 / 低速
设备

USB 系统 (5)

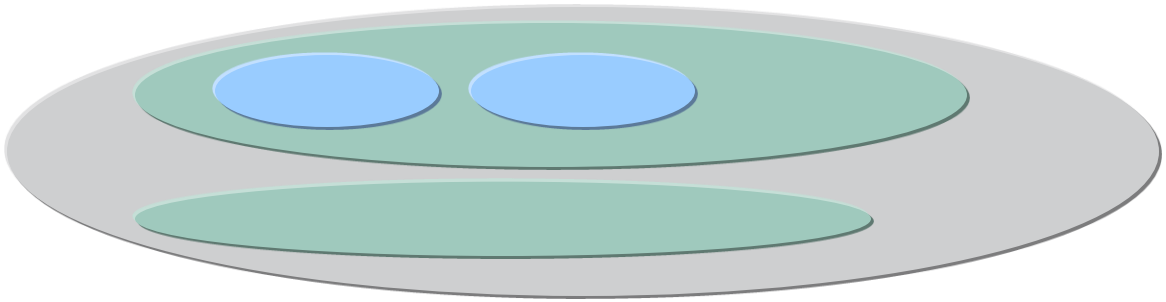

主机
连接层
设备
客户端
应用程序
系统端
驱动程序
主机端
控制程序
应用层
应用程序
逻辑层
驱动程序
总线层
控制程序
USB设备:
位于USB电缆的另一端，实现特定的用户功能。

USB 逻辑部件 (1)

设备

接口

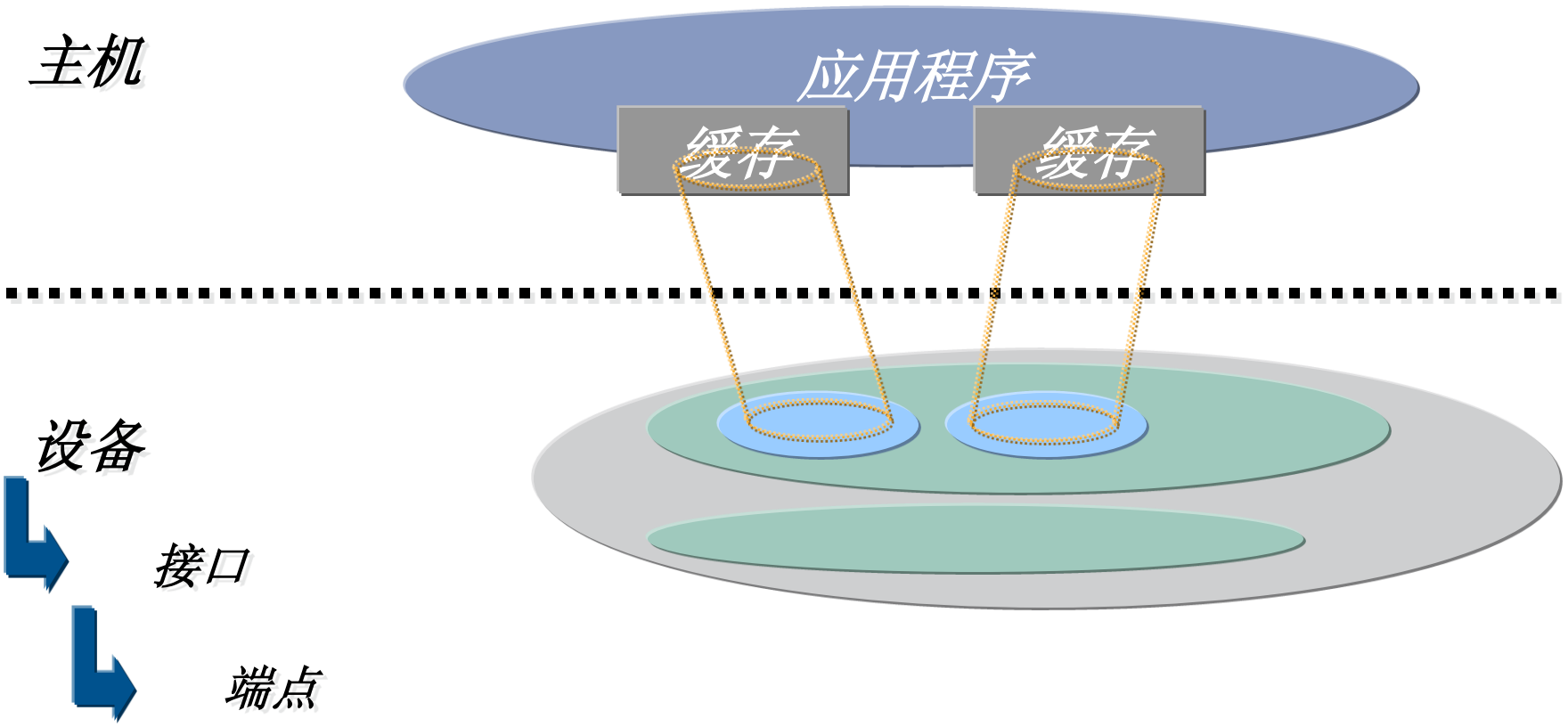
端点



接口：

接口用于描述特定的功能，每个接口都有一个端点集，用于实现接口功能。

USB 逻辑部件 (2)



端点：

通信频率　带宽　端点号　错误处理　最大包长度

传输类型　方向

USB 类 (1)

- 音频类 (Audio)
- 通信类—虚拟串口类 (CDC)
- 设备固件升级类 (DFU)
- 人机接口类 (HID)
- 大容量存储设备类 (Mass Storage)

- http://www.usb.org/developers/devclass_docs#approved

USB 类 (2)

- 智能卡接口设备类 (CCID)
- 图像类 (Image)
- IrDA桥接设备类 (IrDA Bridge)
- 监视设备类 (Monitor)
- 个人保健设备类 (Personal Health Care)
- 电源设备类 (Power Device)
- 物理接口设备类 (Physical Interface)
- 打印设备类 (Printer)
- 视频类 (Video)
- 测试测量类 (Test & Measurement)

供电 (1)

- **USB设备按供电类别分：**
  - 主hub：直接连接到USB主机控制器，与主机控制器使用同一供电电源，为下游端口提供1个单位(100mA)－5个单位(500mA)的供电。
  - 总线供电类hub：从上游端口取电，为自身及其下游设备供电。
  - 自供电hub：从外部电源取电，为自身及其下游设备供电。
  - 低功耗总线供电类设备：所有的耗电都从上游总线获取，在任意时刻的耗电都不能超过1个单位。
  - 高功耗总线供电类设备：所有的耗电都从上游总线获取，在上电初始时，耗电不能超过1个单位，当设备正确配置以后，可以从上游获得最多5个单位的供电。
  - 自供电设备：能从上游总线获得1个单位的供电，设备需要消耗的其他供电从外部电源获取。

供电 (2)

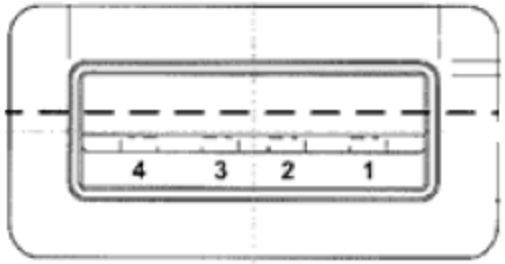
- 挂起模式
  - 高功耗设备的挂起模式：

    在此模式下，耗电最多不能超过2.5mA。

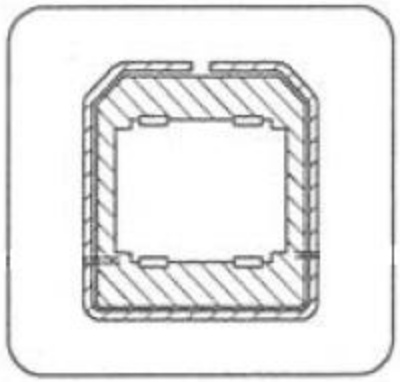
  - 低功耗设备的挂起模式：

    在此模式下，耗电最多不能超过500uA。

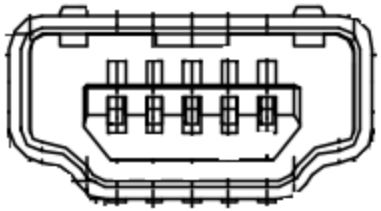
USB 连接器(1) – Standard / Mini



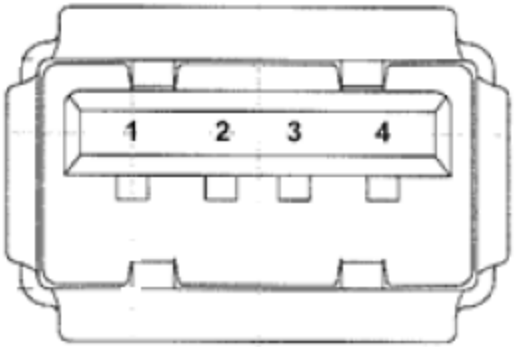
Standard A Plug



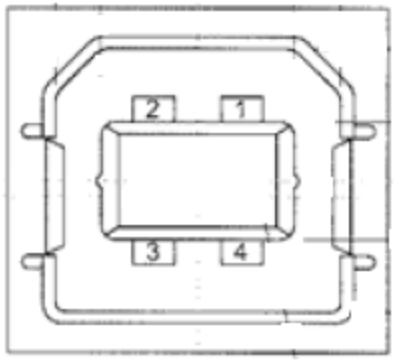
Standard B Plug



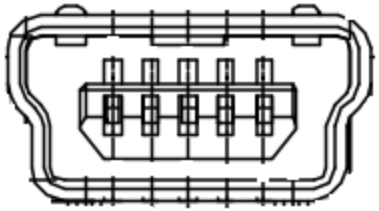
Mini B Plug



Standard A Receptacle

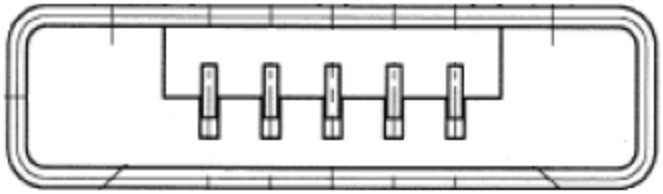


Standard B Receptacle

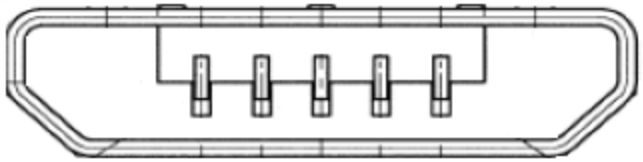


Mini B Receptacle

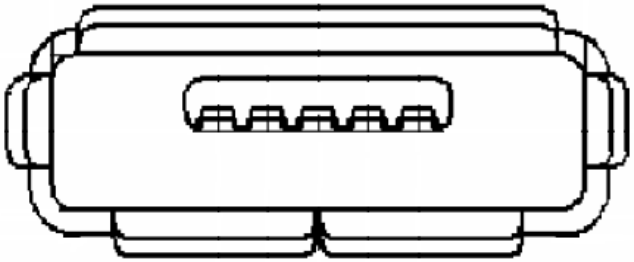
USB 连接器 (2) -- Micro



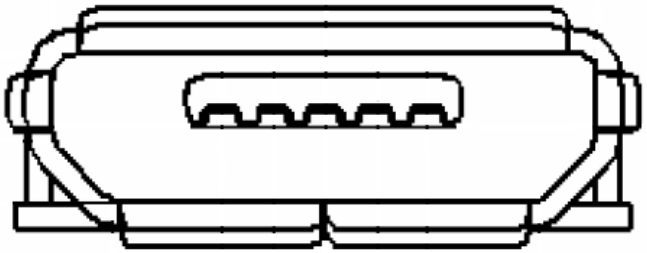
Micro A Plug



Micro B Plug



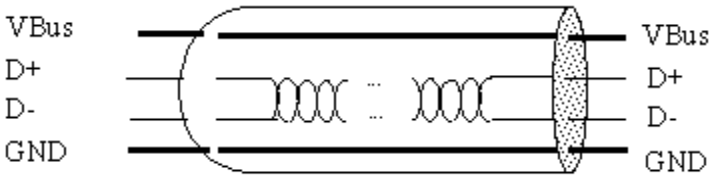
Micro AB Receptacle



Micro B Receptacle

USB 信号 (1)

电缆:



差分信号: 1: D+ > V_{OH} (2.8V)and D- < V_{OL}(0.3V)

0: D- > V_{OH} and D+ < V_{OL}

J状态: Low Speed: Differential 0

Full Speed: Differential 1

K状态: Low Speed: Differential 1

Full Speed: Differential 0

USB 信号 (2)

SE0 (Single-ended0): D+ and D- < V_{OL}

SE1 (Single-ended1): D+ and D- > V_{SE1}(0.8V)

复位 (Reset): D+ and D- < V_{OL} for >= 10ms

恢复 (Resume): K 状态

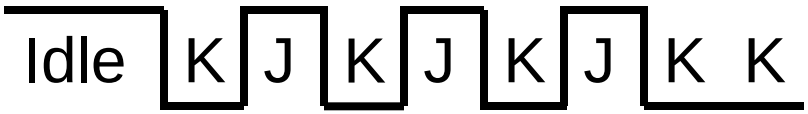
SOP (Start of Packet):

数据线从IDLE状态切换到K状态

EOP (End of Packet):

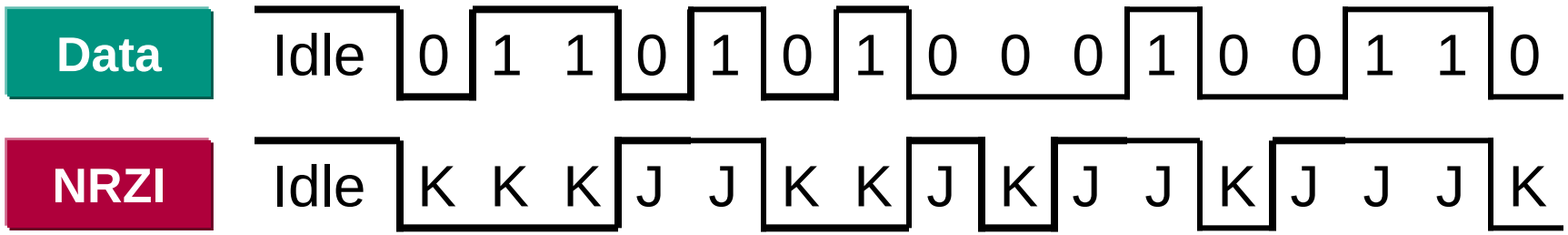
持续2位时间的SE0信号，后跟随1位时间的J状态。

SYNC: 3个重复的KJ状态切换，后跟随2位时间的K状态，共持续8位时间。 (低速/全速设备)



USB 信号 (3)

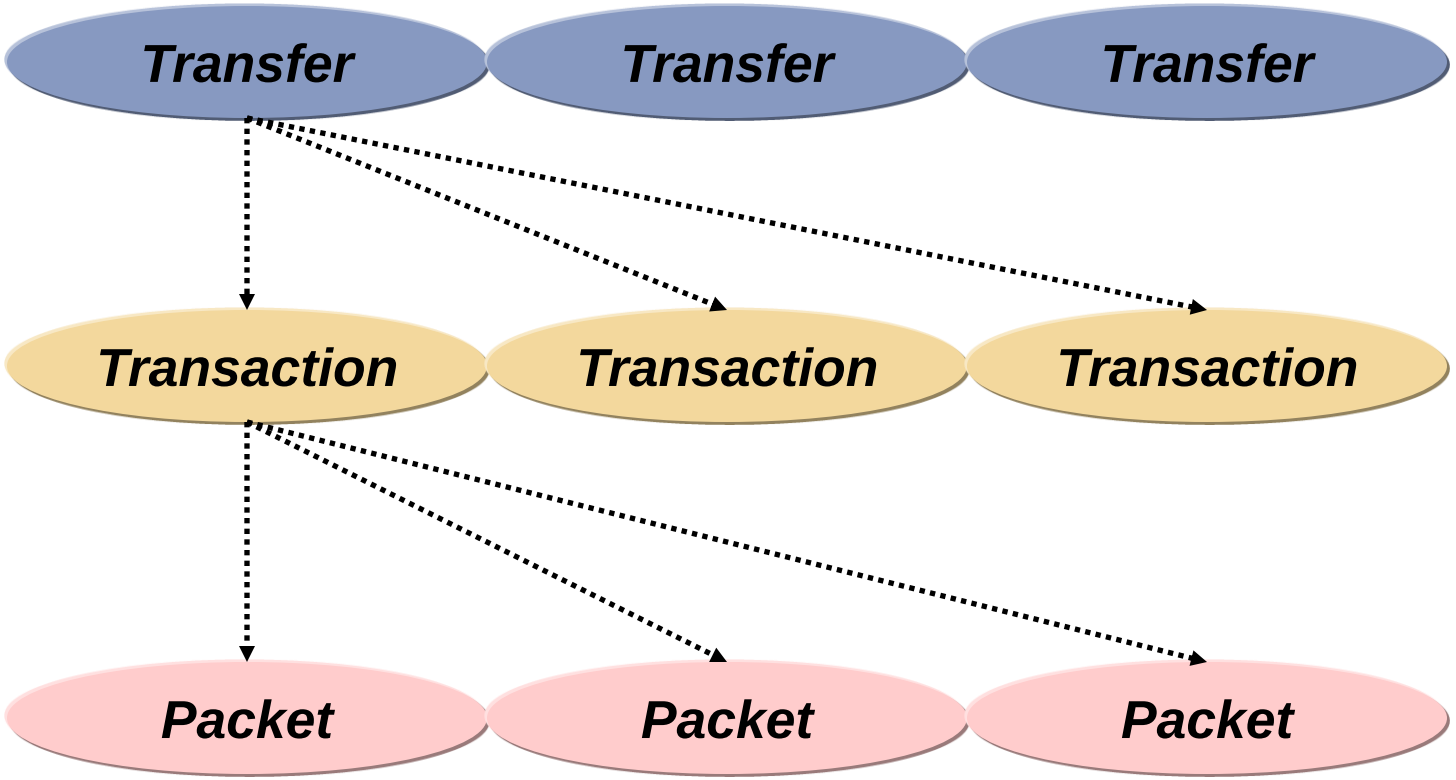
数据的编码 / 解码: 反向不归零码(NRZI)



Bit 填充:

在数据进行**NRZI**编码前，每**6**个连续的**1**信号之后都会插入**1**个**0**信号，以免丢失同步。

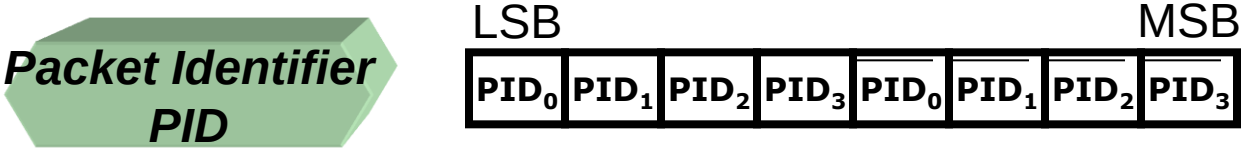
USB 数据格式


Transfer
Transfer
Transfer
Transaction
Transaction
Transaction
Packet
Packet
Packet

Packet 格式 (1)

Packet 格式 (2)

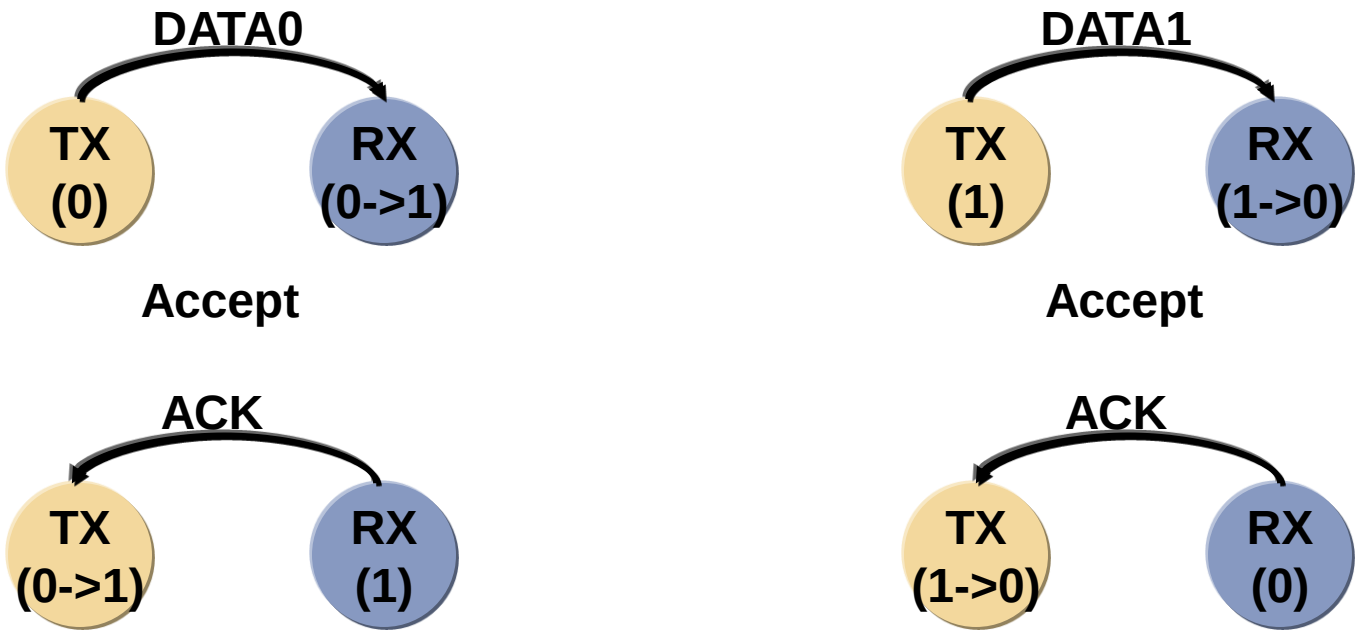
Packet 格式 (3)



PID Type	PID Name
Token	OUT / IN / SOF / SETUP
Data	DATA0 / DATA1 / DATA2 / MDATA
Handshake	ACK / NAK / STALL / NYET
Special	PRE / ERR / SPLIT / PING

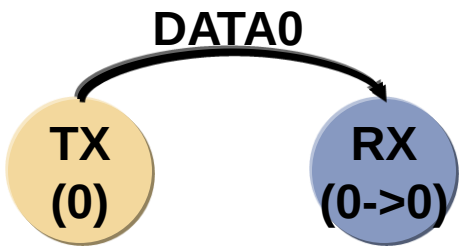
Packet 格式 (3) -- Data PID Toggle(1)

- **Data Toggle** 用于数据的同步和重发
  - 正确的数据传输流程

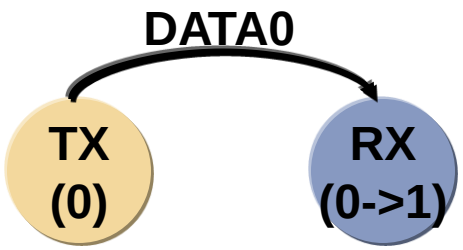


Packet 格式 (3) -- Data PID Toggle(2)

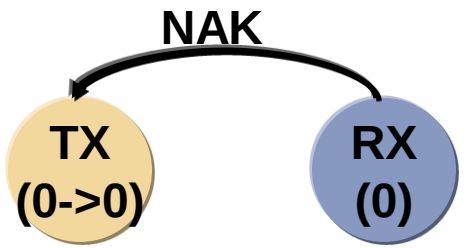
- **Data Toggle** 用于数据的同步和重发
  - 当数据被破坏或者没有正确接收

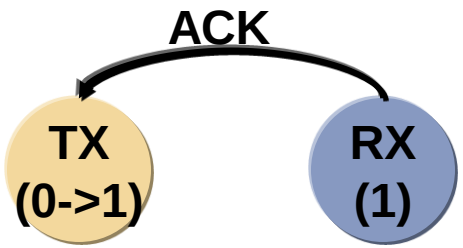


Reject



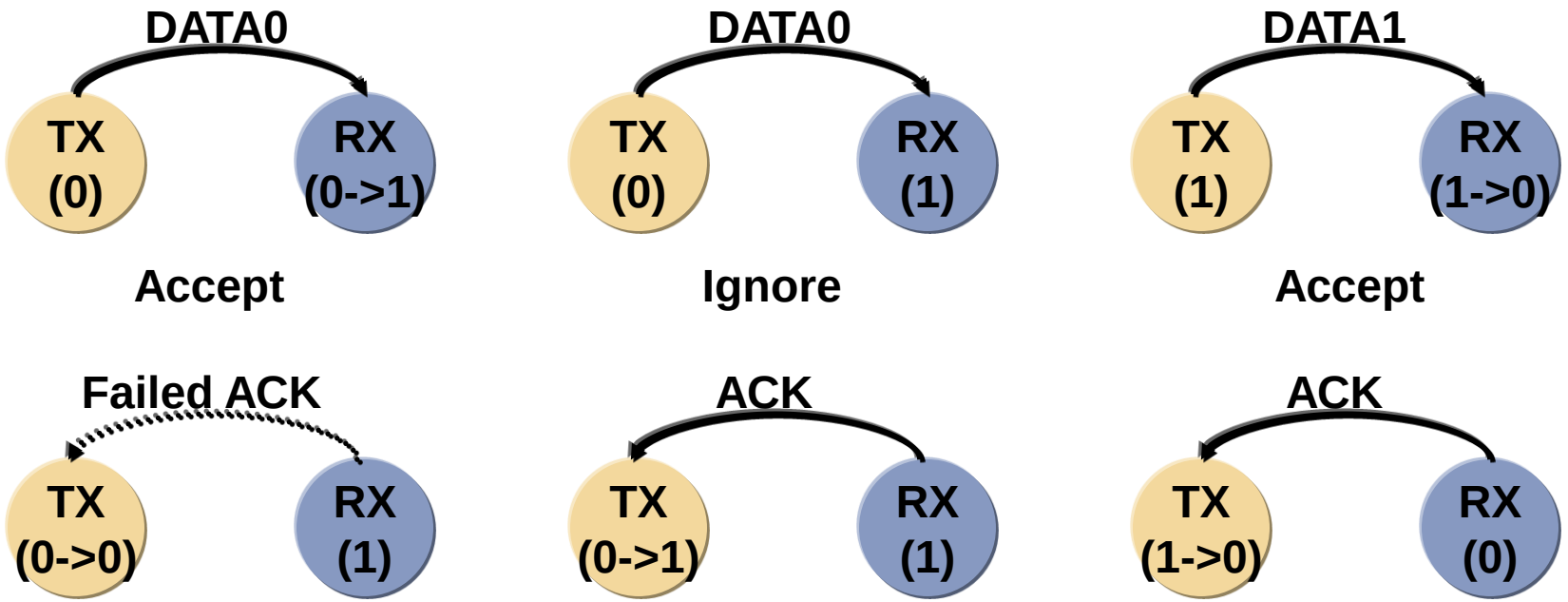
Accept



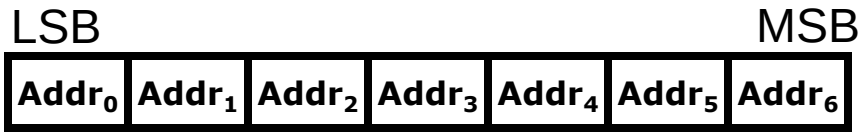


Packet 格式 (3) -- Data PID Toggle(3)

- **Data Toggle 用于数据的同步和重发**
  - **当ACK的传输被破坏**

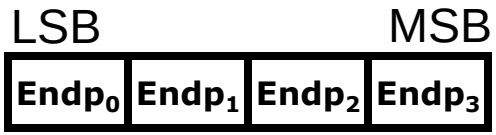


Packet 格式 (4)



设备地址

+



端点地址

低速设备:

支持最多3个端点

全速和高速设备:

支持最多16个IN和OUT端点

Packet 格式 (5)

- 11 位
- 主机每发出一个帧，帧号都会自加1
- 当帧号达到7FFH时，将归零重新开始计数
- 仅在每个SOF帧的帧首传输帧号

Packet 格式 (6)

- 根据传输类型的不同，数据域的数据长度从0到1024字节不等。

Packet 格式 (7)

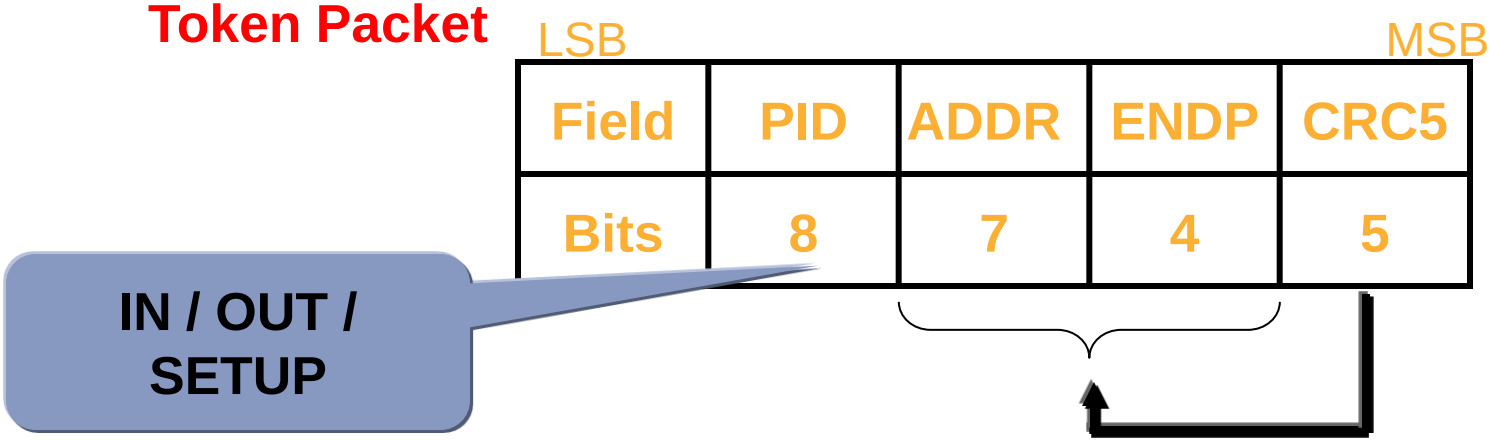
- Token CRC
  - 计算IN，OUT，和SETUP Token地址域的CRC
  - 计算SOF Token的帧号域的CRC
  - $G(X) = X^5 + X^2 + 1$
- Data CRC
  - 计算所有数据域数据的CRC
  - $G(X) = X^{16} + X^{15} + X^2 + 1$

Packet类型

- **Packet分四大类：**
  - 命令 (Token) Packet
  - 帧首 (Start of Frame) Packet
  - 数据 (Data) Packet
  - 握手 (Handshake) Packet

Packet类型(1): Token Packet

Token Packet

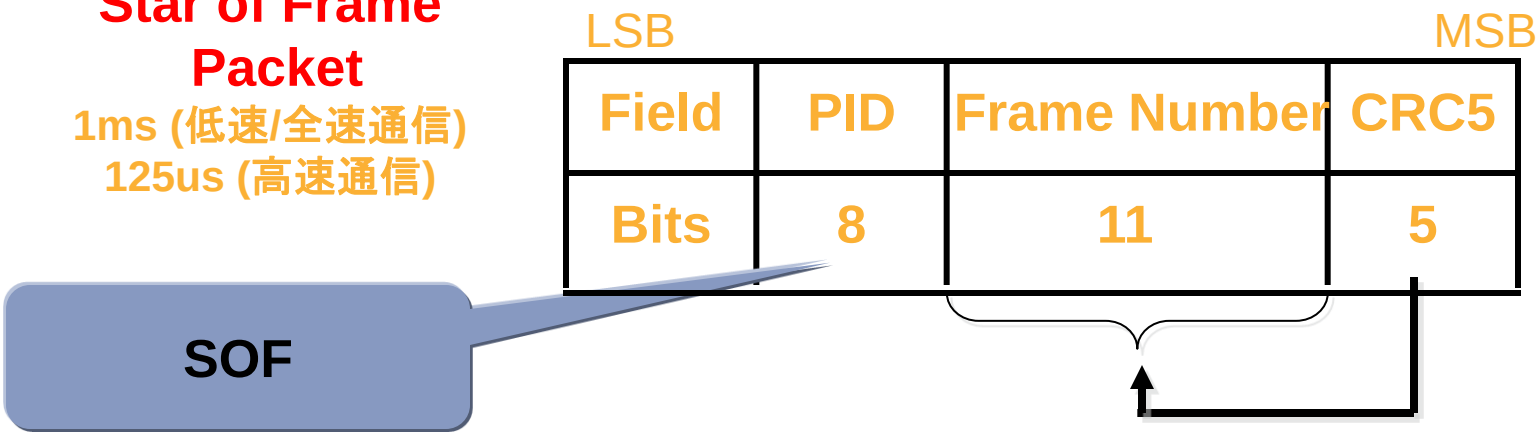


Sync	SETUP	ADDR	ENDP	CRC5	EOP
00000001	0xB4	3	0	0x0A	250.000 ns

Packet类型(2): Start of Frame Packet

Star of Frame Packet

1ms (低速/全速通信)
125us (高速通信)



Sync	SOF	Frame #	CRC5	EOP
00000001	0xA5	1611	0x11	250.000 ns

Packet类型(3): Data Packet

Data Packet LSB MSB

Field	PID	DATA	CRC16
Bits	8	0-1024	16

DATA0 / DATA1

Sync	DATA0	Data	CRC16	EOP
00000001	0xC3	80 06 00 01 00 00 12 00	0x072F	250.000 ns

Packet类型(4): Handshake Packet

Handshake Packet

LSB MSB

Field	PID
Bits	8

ACK:
传输正确完成

NAK:
设备不能接收数据或者没有准备好数据上传

STALL:
设备不能进行传输

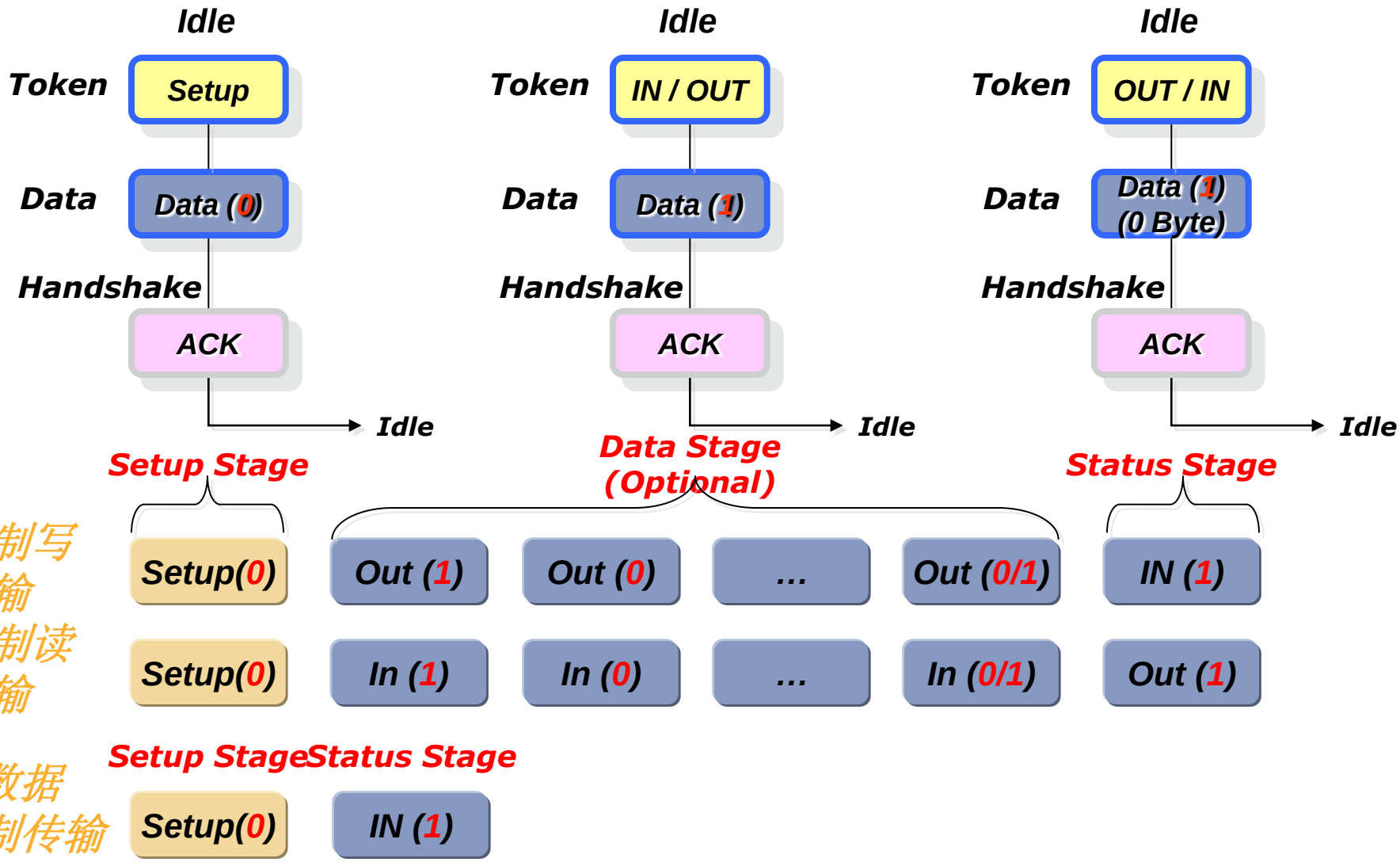
NYET / ERR:
仅用于高速传输设备没有准备好或者出错

Sync	ACK	EOP
00000001	0x4B	233.330 ns

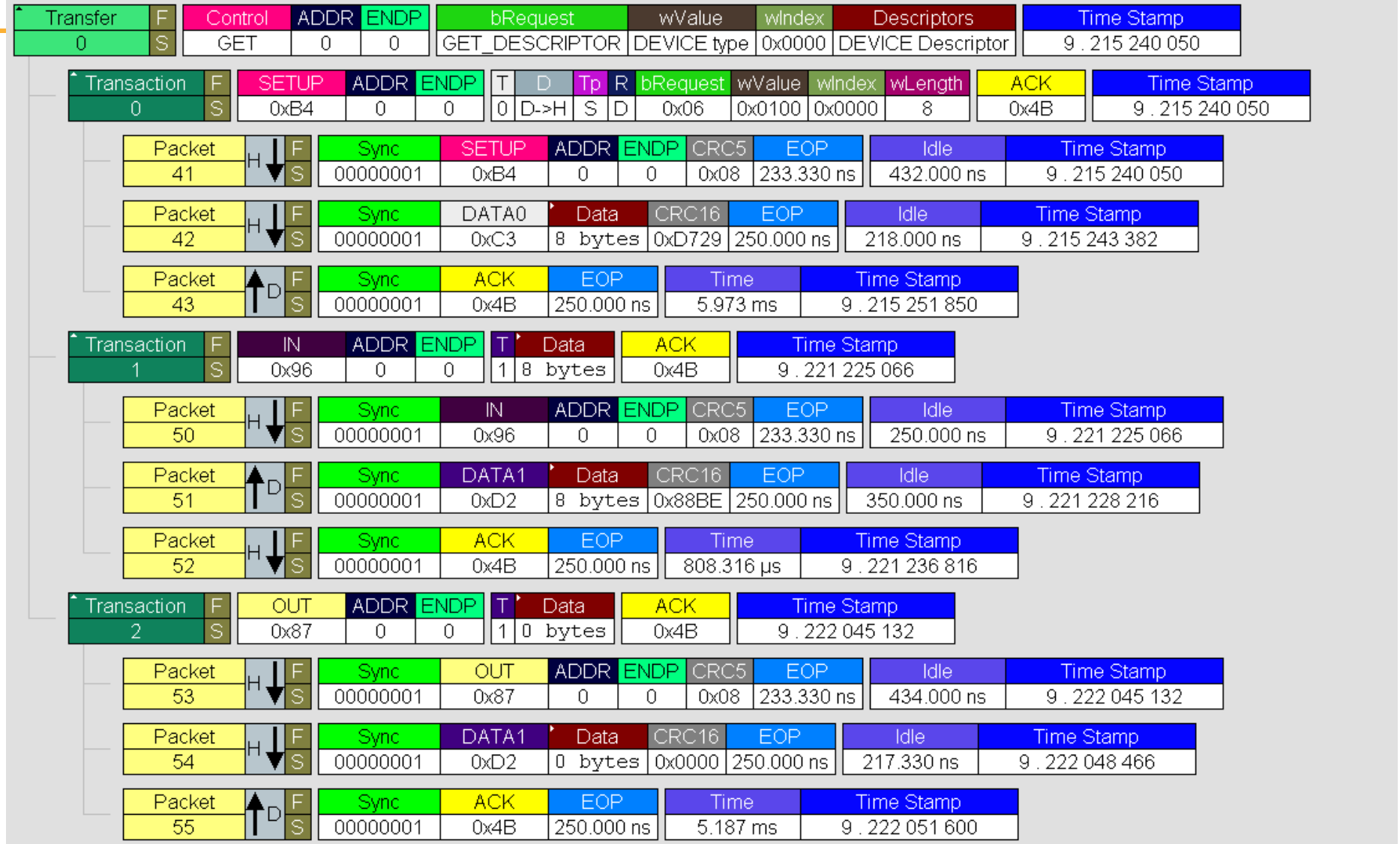
传输类型

- **USB协议定义了四种传输类型：**
  - **控制传输(Control Transfers):** 突发，非周期性，由主机发起，用于命令和状态的传输。
  - **同步传输(Isochronous Transfers):** 周期性，持续性的传输，用于传输与时效相关的信息，并且在数据中保存时间戳的信息。
  - **中断传输(Interrupt Transfers):** 周期性，低频率，允许有限延迟的通信。
  - **大容量数据传输(Bulk Transfers):** 非周期性，大容量突发数据的通信，数据可以占用任意带宽，并容忍延迟。

控制传输 (1)


Idle
Token
Setup
Data
Data (0)
Handshake
ACK
Idle
Setup Stage
Idle
Token
IN / OUT
Data
Data (1)
Handshake
ACK
Idle
Data Stage
(Optional)
Idle
Token
OUT / IN
Data
Data (1)
(0 Byte)
Handshake
ACK
Idle
Status Stage
控制写传输
Setup(0)
Out (1)
Out (0)
…
Out (0/1)
IN (1)
控制读传输
Setup(0)
In (1)
In (0)
…
In (0/1)
Out (1)
Setup Stage
Status Stage
无数据控制传输
Setup(0)
IN (1)

控制传输 (2)



Transfer	F	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors	Time Stamp
0	S	GET	0	0	GET_DESCRIPTOR	DEVICE type	0x0000	DEVICE Descriptor	9 . 215 240 050

Transaction	F	SETUP	ADDR	ENDP	T	D	Tp	R	bRequest	wValue	wIndex	wLength	ACK	Time Stamp
0	S	0xB4	0	0	0	D->H	S	D	0x06	0x0100	0x0000	8	0x4B	9 . 215 240 050

Packet	H	F	Sync	SETUP	ADDR	ENDP	CRC5	EOP	Idle	Time Stamp
41		S	00000001	0xB4	0	0	0x08	233.330 ns	432.000 ns	9 . 215 240 050

Packet	H	F	Sync	DATA0	Data	CRC16	EOP	Idle	Time Stamp
42		S	00000001	0xC3	8 bytes	0xD729	250.000 ns	218.000 ns	9 . 215 243 382

Packet	D	F	Sync	ACK	EOP	Time	Time Stamp
43		S	00000001	0x4B	250.000 ns	5.973 ms	9 . 215 251 850

Transaction	F	IN	ADDR	ENDP	T	Data	ACK	Time Stamp
1	S	0x96	0	0	1	8 bytes	0x4B	9 . 221 225 066

Packet	H	F	Sync	IN	ADDR	ENDP	CRC5	EOP	Idle	Time Stamp
50		S	00000001	0x96	0	0	0x08	233.330 ns	250.000 ns	9 . 221 225 066

Packet	D	F	Sync	DATA1	Data	CRC16	EOP	Idle	Time Stamp
51		S	00000001	0xD2	8 bytes	0x88BE	250.000 ns	350.000 ns	9 . 221 228 216

Packet	H	F	Sync	ACK	EOP	Time	Time Stamp
52		S	00000001	0x4B	250.000 ns	808.316 µs	9 . 221 236 816

Transaction	F	OUT	ADDR	ENDP	T	Data	ACK	Time Stamp
2	S	0x87	0	0	1	0 bytes	0x4B	9 . 222 045 132

Packet	H	F	Sync	OUT	ADDR	ENDP	CRC5	EOP	Idle	Time Stamp
53		S	00000001	0x87	0	0	0x08	233.330 ns	434.000 ns	9 . 222 045 132

Packet	H	F	Sync	DATA1	Data	CRC16	EOP	Idle	Time Stamp
54		S	00000001	0xD2	0 bytes	0x0000	250.000 ns	217.330 ns	9 . 222 048 466

Packet	D	F	Sync	ACK	EOP	Time	Time Stamp
55		S	00000001	0x4B	250.000 ns	5.187 ms	9 . 222 051 600

控制传输 (3)

- **特性:**
  - 每个USB设备都必须有控制端点，支持控制传输来进行命令和状态的传输。USB主机驱动将通过控制传输与USB设备的控制端点通信，完成USB设备的枚举和配置。

- **方向:**
  - 控制传输是双向的传输，必须有IN和OUT两个方向上的特定端点号的控制端点来完成两个方向上的控制传输。

控制传输 (4)

- 有效的数据长度:
  - 最大的有效数据长度:
    - 低速设备: 8 字节
    - 全速设备: 8, 16, 32, 64 字节
    - 高速设备: 64 字节

- 带宽:
  - 对于低速和全速设备，10%的帧带宽将保留给控制传输， 对于高速设备，20%的微帧带宽将保留给控制传输。
    - 低速设备: 在每个帧内只能有少于4个的任意数据长度的控制传输
    - 全速设备: 在每个帧内能有最多32个的1字节长度的控制传输，和最多13个64字节长度的控制传输
    - 高速设备: 在每个微帧内能有最多43个1字节长度的控制传输，和最多31个64字节长度的控制传输

数据的拆分 (1)

256字节

对于高速设备：
最大有效数据长度为：64字节

64字节 + 64字节 + 64字节 + 64字节

数据的拆分 (2)：数据传输完毕的判定

250字节

对于高速设备：
最大有效数据长度为：64字节

64字节 + 64字节 + 64字节 + 58字节

数据的拆分 (3)：数据传输完毕的判定

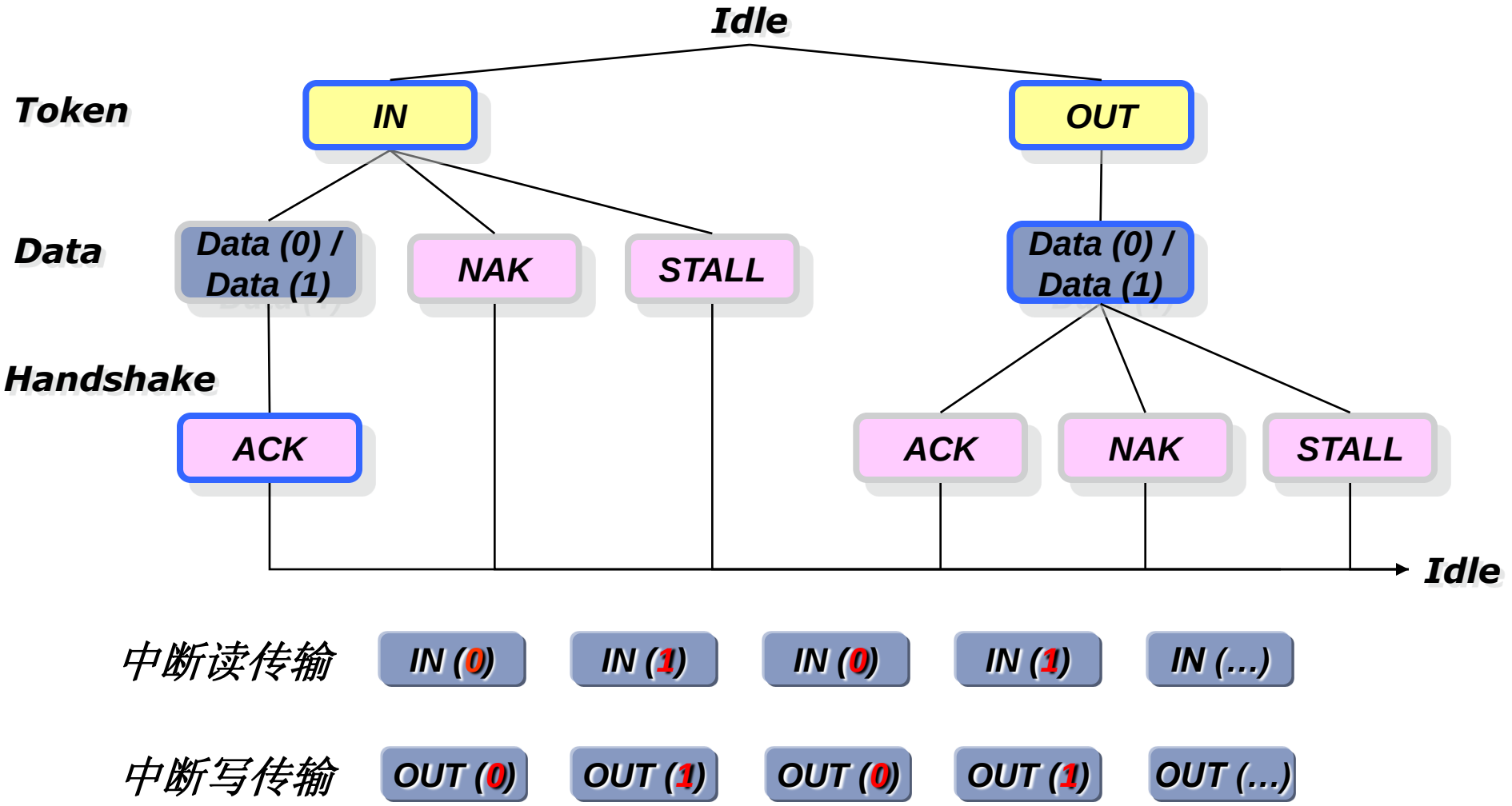
256字节

对于高速设备：
最大有效数据长度为：64字节

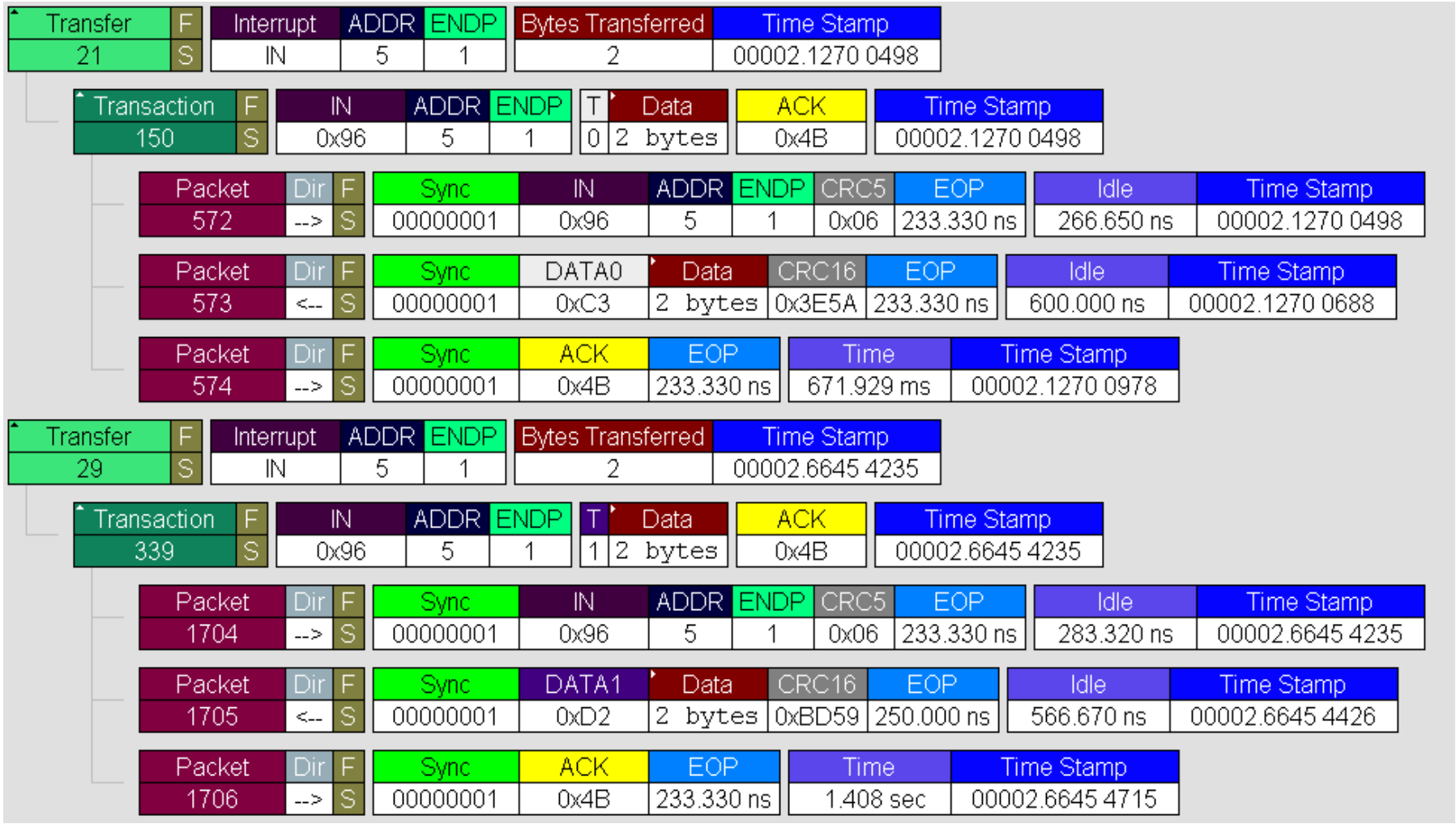
64字节 + 64字节 + 64字节 + 64字节

+ 0字节

中断传输 (1)


Idle
Token
IN
OUT
Data
Data (0) / Data (1)
NAK
STALL
Data (0) / Data (1)
Handshake
ACK
ACK
NAK
STALL
Idle
中断读传输
IN (0)
IN (1)
IN (0)
IN (1)
IN (…)
中断写传输
OUT (0)
OUT (1)
OUT (0)
OUT (1)
OUT (…)

中断传输 (2)



Transfer	F	Interrupt	ADDR	ENDP	Bytes Transferred	Time Stamp
21	S	IN	5	1	2	00002.1270 0498

Transaction	F	IN	ADDR	ENDP	T	Data	ACK	Time Stamp
150	S	0x96	5	1	0	2 bytes	0x4B	00002.1270 0498

Packet	Dir	F	Sync	IN	ADDR	ENDP	CRC5	EOP	Idle	Time Stamp
572	-->	S	00000001	0x96	5	1	0x06	233.330 ns	266.650 ns	00002.1270 0498

Packet	Dir	F	Sync	DATA0	Data	CRC16	EOP	Idle	Time Stamp
573	<--	S	00000001	0xC3	2 bytes	0x3E5A	233.330 ns	600.000 ns	00002.1270 0688

Packet	Dir	F	Sync	ACK	EOP	Time	Time Stamp
574	-->	S	00000001	0x4B	233.330 ns	671.929 ms	00002.1270 0978

Transfer	F	Interrupt	ADDR	ENDP	Bytes Transferred	Time Stamp
29	S	IN	5	1	2	00002.6645 4235

Transaction	F	IN	ADDR	ENDP	T	Data	ACK	Time Stamp
339	S	0x96	5	1	1	2 bytes	0x4B	00002.6645 4235

Packet	Dir	F	Sync	IN	ADDR	ENDP	CRC5	EOP	Idle	Time Stamp
1704	-->	S	00000001	0x96	5	1	0x06	233.330 ns	283.320 ns	00002.6645 4235

Packet	Dir	F	Sync	DATA1	Data	CRC16	EOP	Idle	Time Stamp
1705	<--	S	00000001	0xD2	2 bytes	0xBD59	250.000 ns	566.670 ns	00002.6645 4426

Packet	Dir	F	Sync	ACK	EOP	Time	Time Stamp
1706	-->	S	00000001	0x4B	233.330 ns	1.408 sec	00002.6645 4715

中断传输 (3)

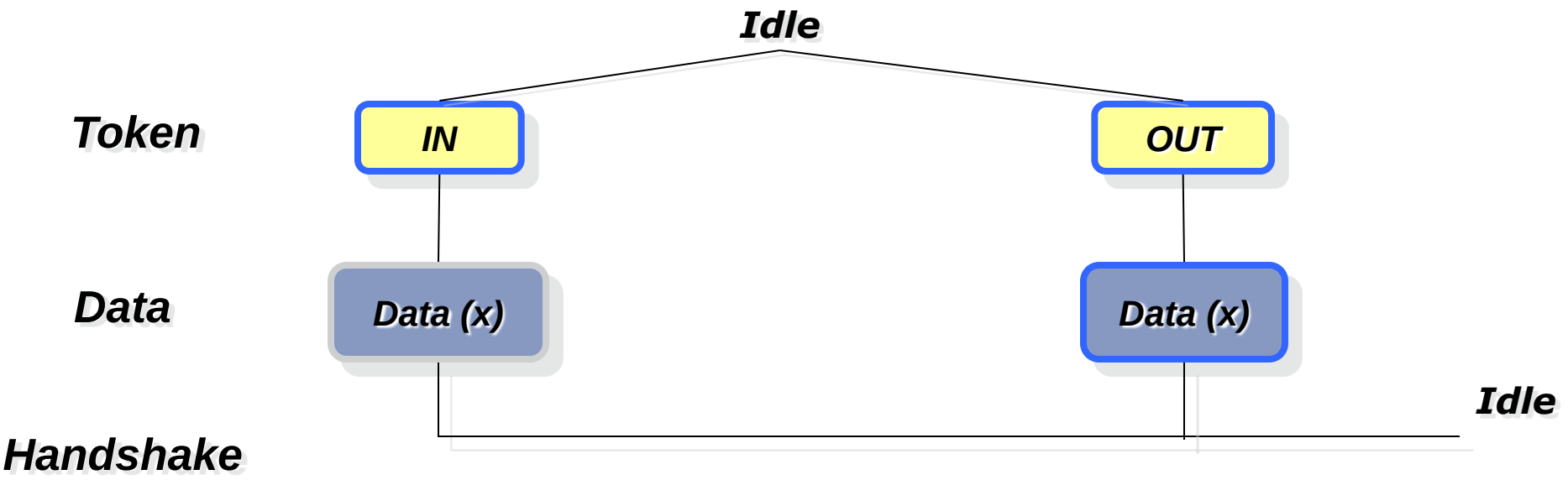
- **特性:**
  - 中断传输用于那些频率不高，但对周期有一定要求的数据传输。具有保证的带宽，并能在下个周期对先前错误的传输进行重传。
  - 对于全速端点，中断传输的间隔时间在1ms到255ms之间, 对于低速端点，间隔时间限制在10ms到255ms之间, 对于高速端点，间隔为$2^{bInterval-1} \times 125us$, bInterval的值在1到16之间。
- **方向:**
  - 中断传输总是单向的，可以用单向的中断端点来实现某个方向上的中断传输。

中断传输 (4)

- 有效的数据长度:
  - 最大的有效数据长度:
    - 低速设备: 8字节
    - 全速设备: 64字节
    - 高速设备: 1024字节

- 带宽:
  - 对于低速和全速设备，有90%的帧带宽是保留给中断传输和同步传输的，对于高速设备，有80%的微帧带宽是保留给中断传输和同步传输的。
    - 低速设备: 每个帧内可以有最多8个1字节长度的中断传输，或者有最多6个8字节长度的中断传输。
    - 全速设备: 每个帧内可以有最多107个1字节长度的中断传输，或者有最多19个64字节长度的中断传输。
    - 高速设备: 每个微帧内可以有最多199个1字节长度的中断传输，或者有最多63个64字节长度的中断传输。

同步传输 (1)



同步读传输 IN (0) IN (0) IN (0) IN (0) IN (…)

同步写传输 OUT (0) OUT (0) OUT (0) OUT (0) OUT (…)

同步传输 (2)

- **特性:**
  - 同步传输用于传输那些需要保证带宽，并且不能忍受延迟的信息。整个带宽都将用于保证同步传输的数据完整，并且不支持出错重传

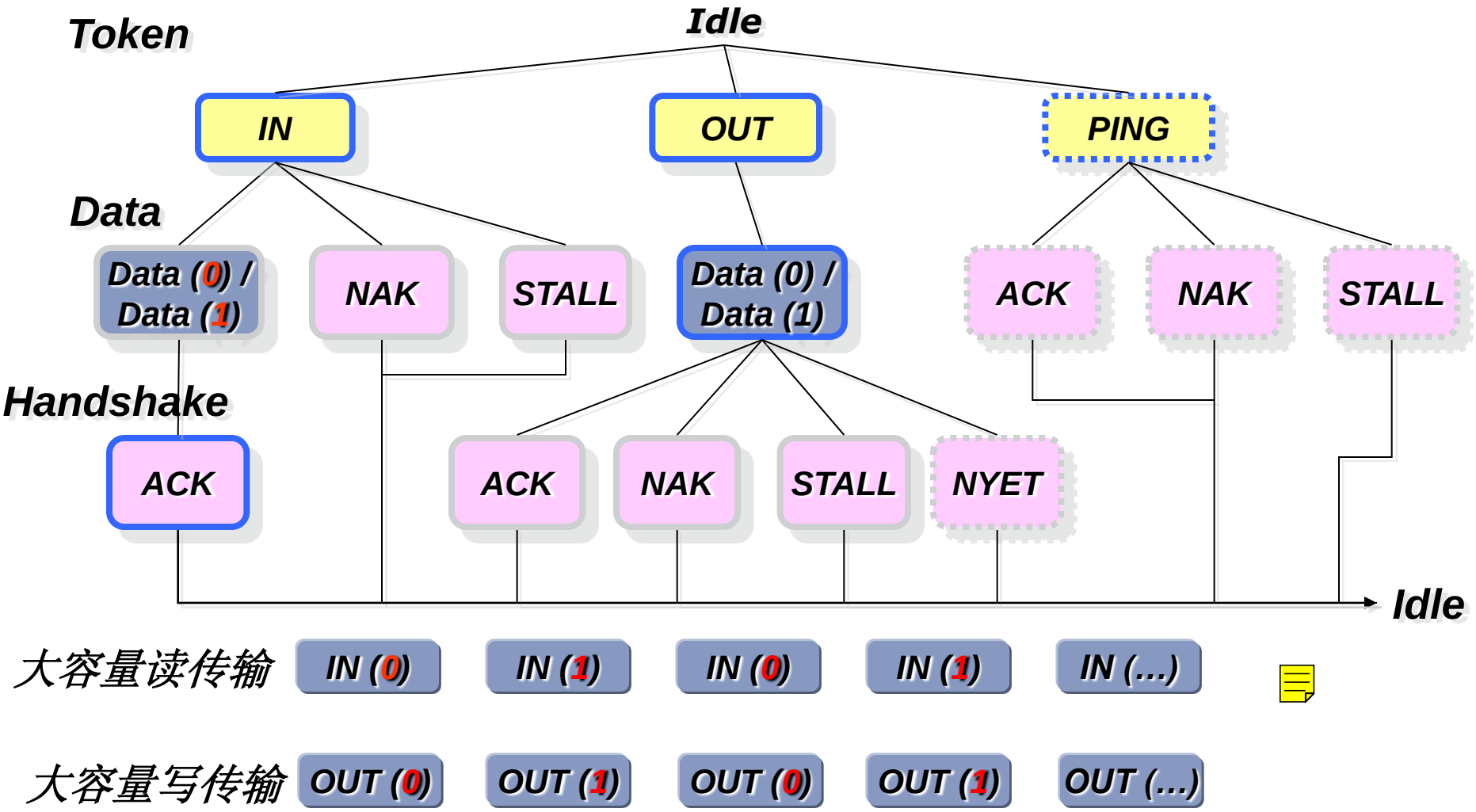
- **方向:**
  - 同步传输总是单向的，可以使用单向的同步端点来实现某个方向上的同步传输。

同步传输 (3)

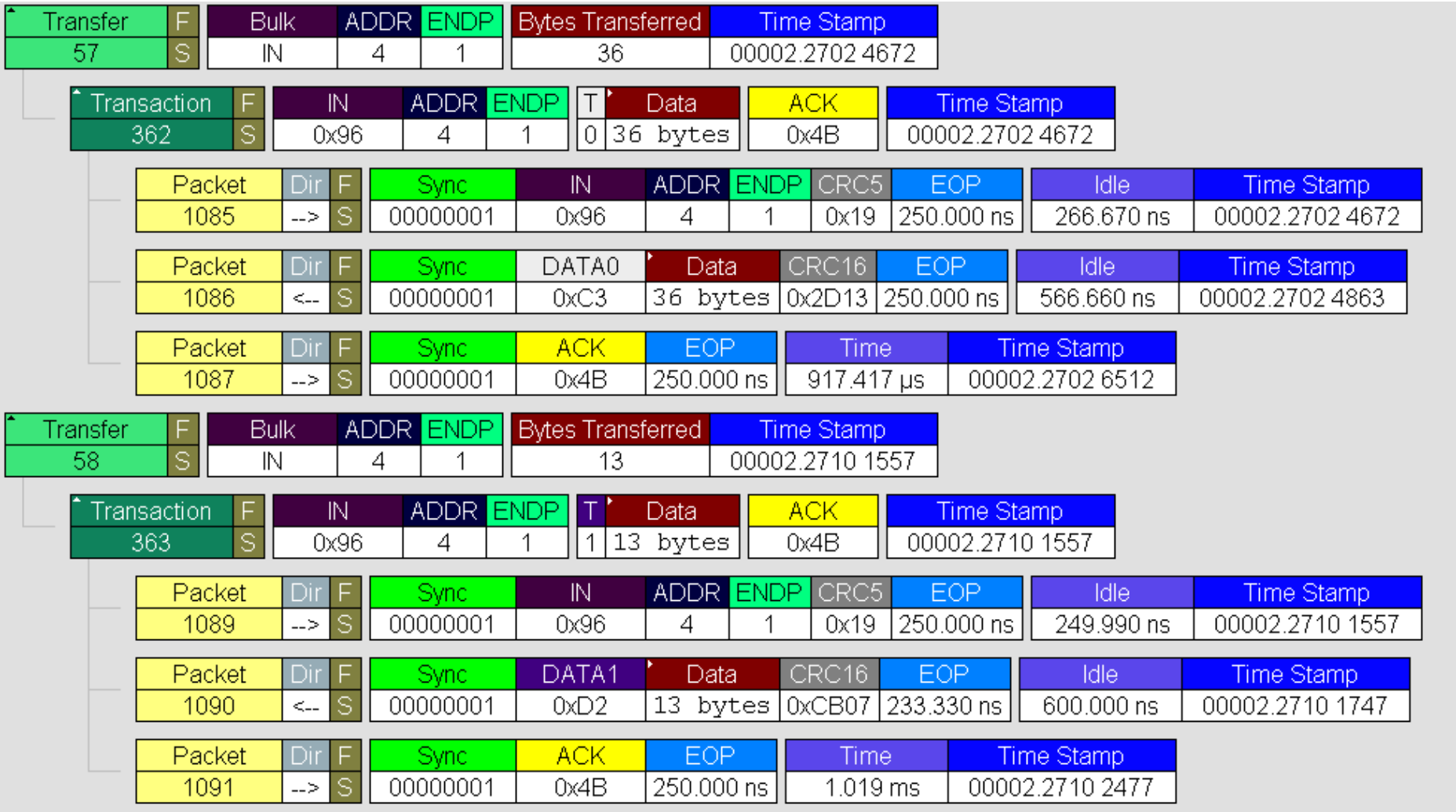
- 有效数据长度:
  - 最大的有效数据长度:
    - 全速设备: 1023 字节
    - 高速设备: 1024 字节

- 带宽:
  - 对于全速设备，有90%的帧带宽是保留给中断传输和同步传输的，对于高速设备，有80%的微帧带宽是保留给中断传输和同步传输的。
    - 全速设备: 在每个帧内，可以有最多150个1字节长度的同步传输，或者有最多1个1023字节长度的同步传输。
    - 高速设备: 在每个微帧内，可以有最多192个1字节长度的同步传输，或者有最多7个1024字节长度的同步传输。

大容量数据传输 (1)


Token
Idle
IN
OUT
PING
Data
Data (0) / Data (1)
NAK
STALL
Data (0) / Data (1)
ACK
NAK
STALL
Handshake
ACK
ACK
NAK
STALL
NYET
Idle
大容量读传输
IN (0)
IN (1)
IN (0)
IN (1)
IN (…)
大容量写传输
OUT (0)
OUT (1)
OUT (0)
OUT (1)
OUT (…)

大容量数据传输 (2)



Transfer	F
57	S

Bulk	ADDR	ENDP	Bytes Transferred	Time Stamp
IN	4	1	36	00002.2702 4672

Transaction	F
362	S

IN	ADDR	ENDP	T	Data	ACK	Time Stamp
0x96	4	1	0	36 bytes	0x4B	00002.2702 4672

Packet	Dir	F
1085	-->	S

Sync	IN	ADDR	ENDP	CRC5	EOP	Idle	Time Stamp
00000001	0x96	4	1	0x19	250.000 ns	266.670 ns	00002.2702 4672

Packet	Dir	F
1086	<--	S

Sync	DATA0	Data	CRC16	EOP	Idle	Time Stamp
00000001	0xC3	36 bytes	0x2D13	250.000 ns	566.660 ns	00002.2702 4863

Packet	Dir	F
1087	-->	S

Sync	ACK	EOP	Time	Time Stamp
00000001	0x4B	250.000 ns	917.417 µs	00002.2702 6512

Transfer	F
58	S

Bulk	ADDR	ENDP	Bytes Transferred	Time Stamp
IN	4	1	13	00002.2710 1557

Transaction	F
363	S

IN	ADDR	ENDP	T	Data	ACK	Time Stamp
0x96	4	1	1	13 bytes	0x4B	00002.2710 1557

Packet	Dir	F
1089	-->	S

Sync	IN	ADDR	ENDP	CRC5	EOP	Idle	Time Stamp
00000001	0x96	4	1	0x19	250.000 ns	249.990 ns	00002.2710 1557

Packet	Dir	F
1090	<--	S

Sync	DATA1	Data	CRC16	EOP	Idle	Time Stamp
00000001	0xD2	13 bytes	0xCB07	233.330 ns	600.000 ns	00002.2710 1747

Packet	Dir	F
1091	-->	S

Sync	ACK	EOP	Time	Time Stamp
00000001	0x4B	250.000 ns	1.019 ms	00002.2710 2477

大容量数据传输 (3)

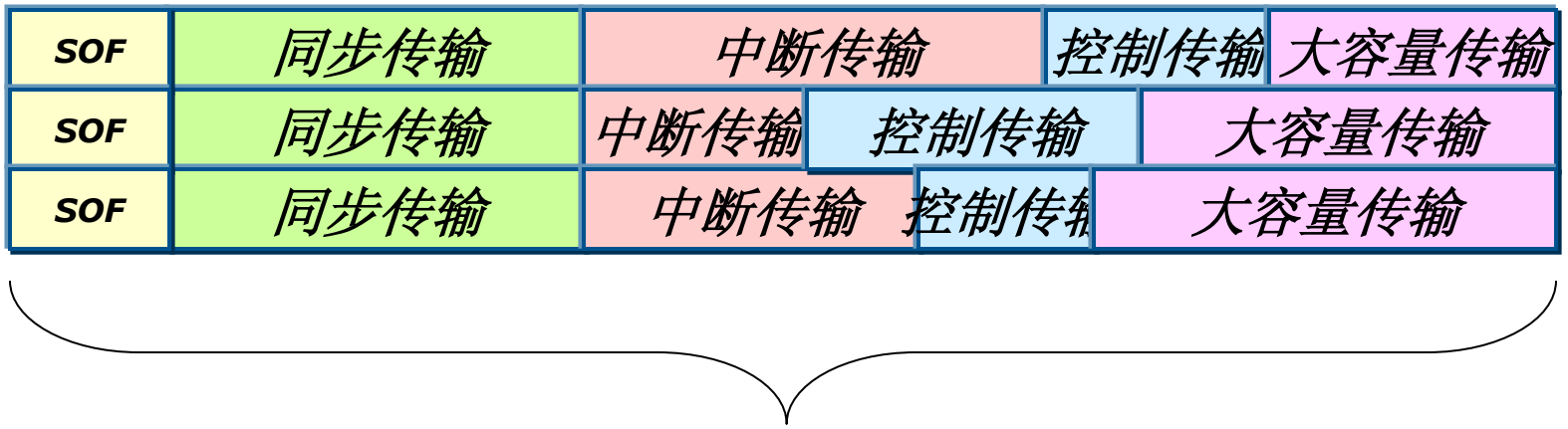
- **特性:**
  - 大容量数据传输适用于那些需要大数据量传输，但是对实时性，对延迟性和带宽没有严格要求的应用。大容量传输可以占用任意可用的数据带宽。

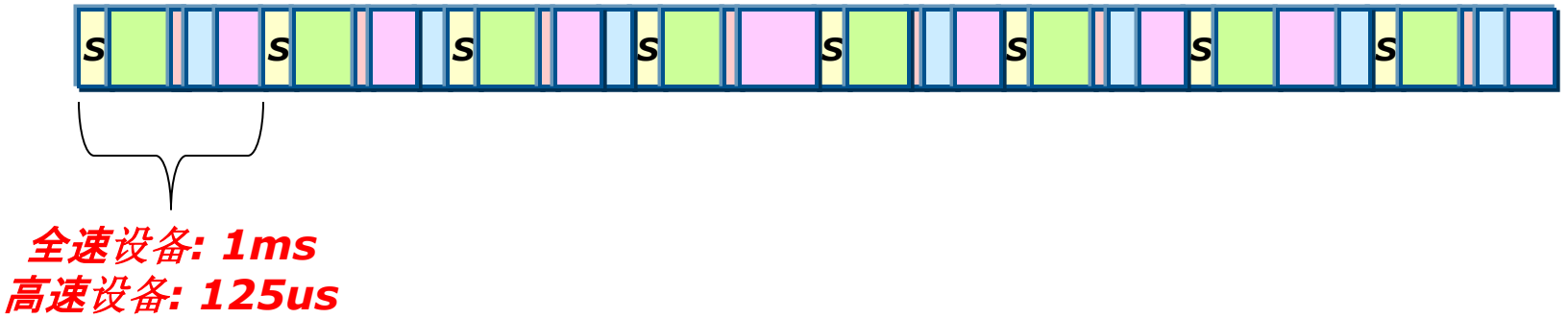
- **方向:**
  - 大容量传输是单向的，可以用单向的大容量传输端点来实现某个方向的大容量传输。

大容量数据传输 (4)

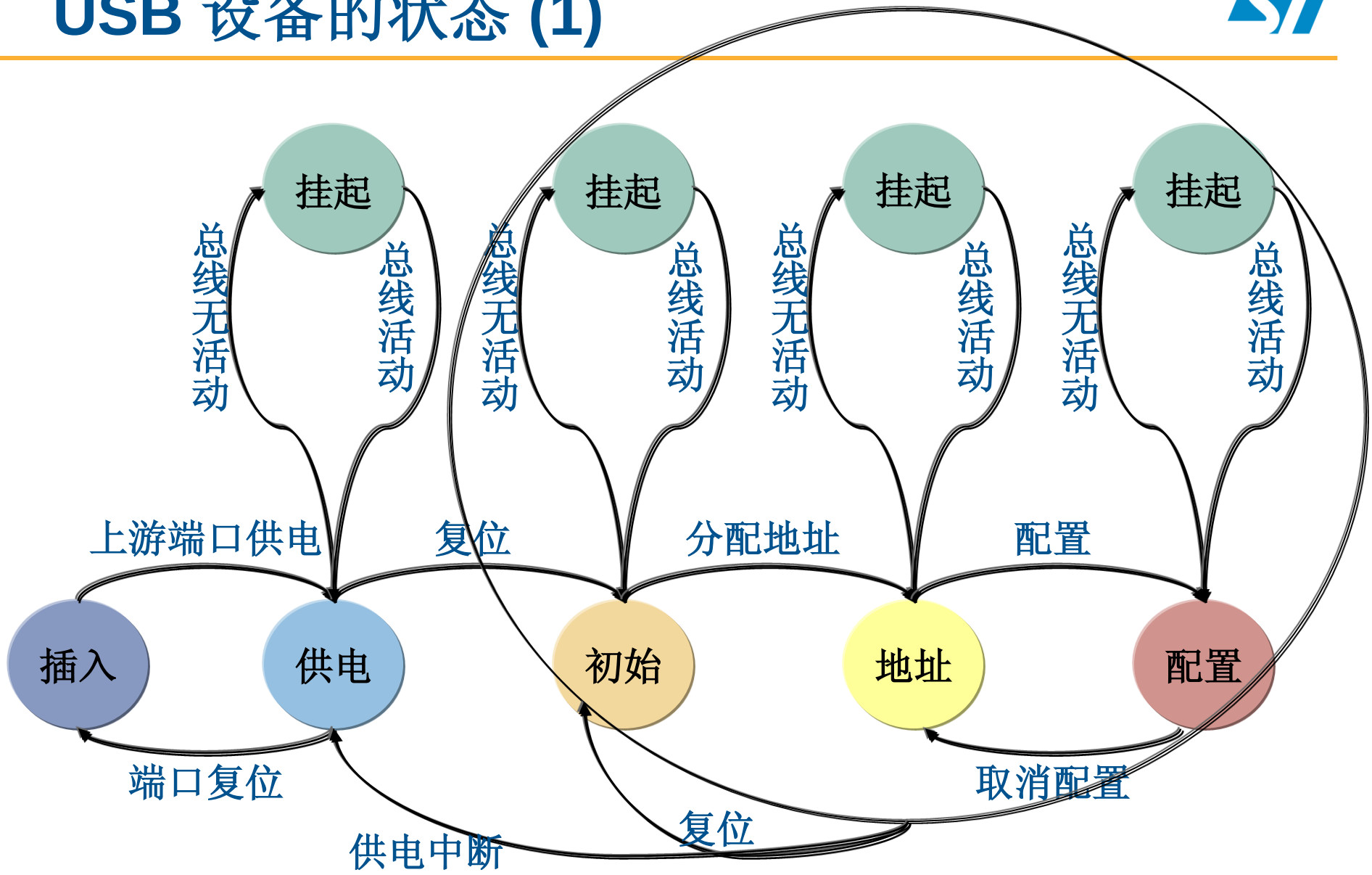
- 有效数据长度:
  - 最长的有效数据长度:
    - 全速设备: 8, 16, 32, 64 字节
    - 高速设备: 512 字节
- 带宽:
  - 没有专门的带宽会为大容量传输保留，只要当前帧有空间没有被其他传输占用，就可以被大容量传输占用。
    - 全速设备: 在每个帧内，可以有最多107个1字节长度的大容量传输或最多19个64字节长度的大容量数据传输。
    - 高速设备: 在每个微帧内，可以有最多133个1字节长度的大容量传输，或最多13个512字节长度的大容量数据传输。

帧格式


SOF
同步传输
中断传输
控制传输
大容量传输
SOF
同步传输
中断传输
控制传输
大容量传输
SOF
同步传输
中断传输
控制传输
大容量传输
全速设备: 1ms
高速设备: 125us


S
S
S
S
S
S
S
S
S
全速设备: 1ms
高速设备: 125us

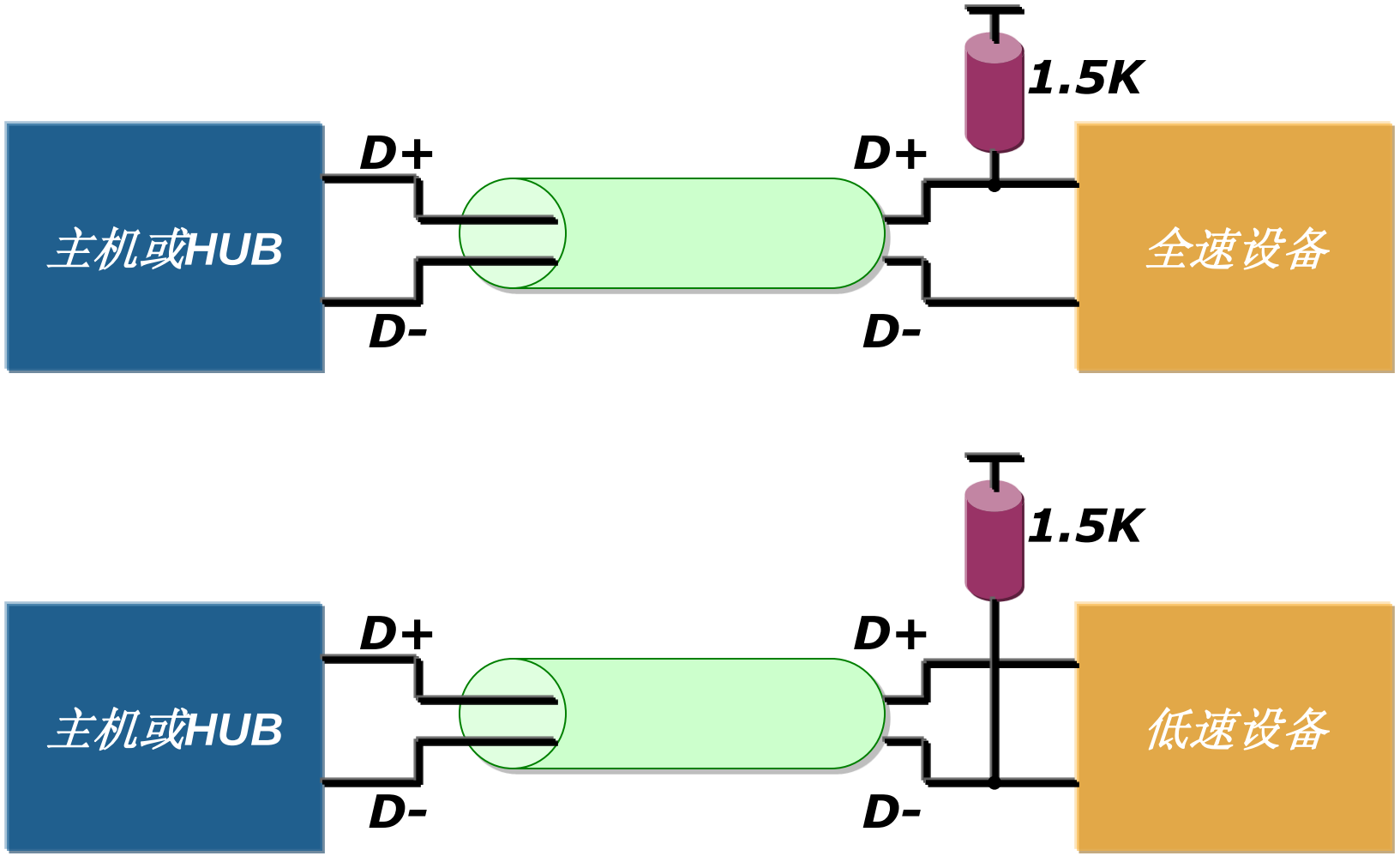
USB 设备的状态 (1)


挂起
挂起
挂起
挂起
总线无活动
总线活动
总线无活动
总线活动
总线无活动
总线活动
总线无活动
总线活动
上游端口供电
复位
分配地址
配置
插入
供电
初始
地址
配置
端口复位
取消配置
复位
供电中断

USB 设备的状态 (2)

插入	供电	初始	地址	配置	挂起	状态
No						设备未插入
Yes	No					设备已插入，但未供电
Yes	Yes	No				设备已插入并供电，但未复位
Yes	Yes	Yes	No			设备已插入，供电并复位，但未分配地址
Yes	Yes	Yes	Yes	No		设备已被分配地址，但没有被配置
Yes	Yes	Yes	Yes	Yes	No	设备已被配置，此时可以使用设备的功能
Yes	Yes	Yes	Yes	Yes	Yes	总线无活动超过3ms

USB 设备的插入检测



USB 设备枚举

当一个USB设备插入主机后，会有以下活动:

- 供电
- 复位
- 获取Device Descriptor
- 复位 (可选)
- 分配地址
- 获取Device Descriptor
- 获取Configuration Descriptor
- 获取String Descriptor (可选)
- 配置

USB 设备描述符(Device) - Joystick

Joystick_DeviceDescriptor[ ] =

```
{
  0x12,              /* 整个Descriptor的长度：18字节 */
  0x01,              /* Descriptor的类别：Device Descriptor(0x01) */
  0x00, 0x02,        /* 设备所遵循的USB协议的版本号：2.00 */
  0x00,              /* 设备所实现的类：由每个接口描述符描述所实现的类*/
  0x00,              /* 设备所实现的子类：由每个接口描述符描述 */
  0x00,              /* 设备所遵循的协议类别：由每个接口描述符描述*/
  0x40,              /* 端点0的最大数据包长度：64字节*/
  0x83, 0x04,        /* IDVendor:  0x0483 (for ST) */
  0x10, 0x57,        /* IDProduct: 0x5710 */
  0x00, 0x02,        /* bcdDevice: 2.00*/
  1,                 /* 用于描述生产厂商的字符描述符的索引号 */
  2,                 /* 用于描述产品的字符描述符的索引号*/
  3,                 /* 用于描述产品系列号的字符描述符的索引号*/
  0x01               /* 设备所支持的配置数目：1*/
}
```

USB 配置描述符(Configuration) – Joystick(1)

Joystick_ConfigDescriptor[ ] =

{

配置描述符：**Configuration Descriptor**

+

接口描述符：**Interface Descriptor**

+

类描述符：**Class Descriptor**

+

端点描述符：**Endpoint Descriptor**

}

USB 配置描述符(Configuration) – Joystick(2)

配置描述符(Configuration Descriptor):

0x09,
/* 描述符的长度：9字节 */

USB_CONFIGURATION_DESCRIPTOR_TYPE,
/* 描述符的类型: 0x02 配置描述符(Configuration) */

JOYSTICK_SIZ_CONFIG_DESC, 0x00,
/* 完整的描述符包括接口描述符、端点描述符和类描述符的长度 */

0x01,
/* 配置所支持的接口数目： 1*/

0x01,
/* 用SetConofiguration()选择此配置，所指定的配置号： 1*/

0x00,
/* 用于描述此配置的字符描述符的索引号：0 */

0xE0,
/* 供电配置：B7(1 保留), B6(自供电), B5(远程唤醒),
B4-B0(0 保留) */

0x32,
/* 最大功耗，以2mA为单位计算：0x32表示 50×2 ＝100mA */

USB 配置描述符(Configuration) – Joystick(3)

接口描述符 (Interface Descriptor):

```
0x09,
/* 描述符的长度：9字节 */
USB_INTERFACE_DESCRIPTOR_TYPE,
/* 描述符的类型：0x04接口描述符(Interface) */
0x00,
/* 选择此接口的索引号，从0开始计算：0 */
0x00,
/* 用于选择此设置的索引号：0 */
0x01,
/* 实现此接口需要使用的端点数目：1 */
0x03,
/* 此接口所遵循的类：HID Class */
0x01,
/* 此接口所遵循的子类：1=BOOT, 0=no boot: requiring BIOS support */
0x02,          /* 此接口所支持的协议：0：自定义、1：键盘、2：鼠标 */

0,     /* 用于描述此接口的字符描述符的索引号 */
```

USB 配置描述符(Configuration) – Joystick(4)

类描述符(Class Descriptor):

```
0x09,
/* 描述符长度：9字节 */
HID_DESCRIPTOR_TYPE,
/* 描述符类型：HID类描述符 */
0x00, 0x01,
/* 所遵循的HID协议版本：1.00 */
0x00,
/* 国家代码：无*/
0x01,
/* 按照类定义，后续所需要的描述符的数目：1*/
0x22,
/* 后续的描述符的类型：报告描述符 */
JOYSTICK_SIZ_REPORT_DESC, 0x00,
/* 后续的描述符的长度：*/
```

USB 配置描述符(Configuration) – Joystick(5)

端点描述符(Endpoint Descriptor):

0x07,

/* 描述符长度：7字节 */

USB_ENDPOINT_DESCRIPTOR_TYPE,

/* 描述符类型：端点描述符*/

0x81,

/* 端点的特性：

B3-B0(端点号), B6-B4(0), B7(1=IN, 0=OUT): 0x81：Endpoint1/ IN */

0x03,

/* 端点的类型：

B1-B0(00=控制 01=同步 10=大容量 11=中断): 0x03：中断端点 */

0x04, 0x00

/* 此端点的最大有效数据长度：4 字节 */

0x20,

/* 主机查询此端点数据的间隔时间：(1ms或125us单位): 0x20：32 ms */

USB 字符描述符(String) - Joystick

```
Joystick_StringVendor[ ] =
 {
     JOYSTICK_SIZ_STRING_VENDOR,
     /* 描述符的长度 */

     USB_STRING_DESCRIPTOR_TYPE,
    /* 描述符的类型：字符描述符 */

     'S', 0, 'T', 0, 'M', 0, 'i', 0, 'c', 0, 'r', 0, 'o', 0, 'e', 0,
     'l', 0, 'e', 0, 'c', 0, 't', 0, 'r', 0, 'o', 0, 'n', 0, 'i', 0,
     'c', 0, 's', 0
    /* 描述符所描述的内容："STMicroelectronics" */
 }
```

内容

- USB技术简介
  - 技术背景
  - 架构、系统、协议和供电
  - 设备的枚举、识别
  - 传输类型

- **STM32 USB模块和函数库**
  - 模块的特性
  - 各类描述符解析
  - 模块的中断源及相关的中断处理函数
  - 模块的其他相关函数库

STM32 USB模块特性

- 遵循USB2.0全速设备标准
- 支持双向8个端点，即8个IN端点和8个OUT端点
- 硬件实现CRC自动生成/校验, NRZI 编码/解码和bit-stuffing.
- 支持控制传输，中断传输，大容量传输和同步传输四种传输类型，并内置专为大容量传输和同步传输所设定的双缓存区。
- 支持USB挂起/唤醒

STM32103 USB 模块中断源

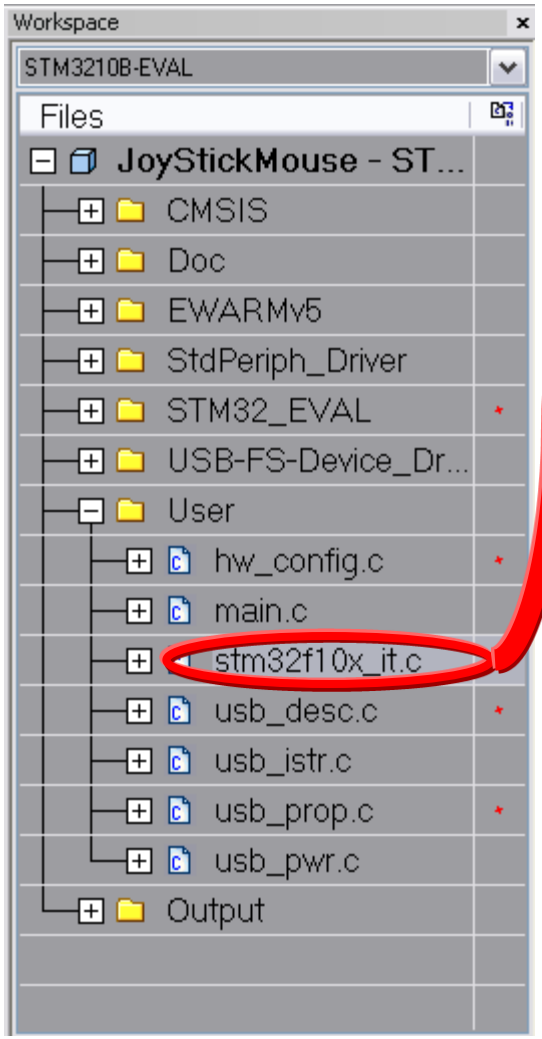
USB_HP_CAN_TX (19) —— 由同步传输，或双缓存大容量传输的CTR事件触发

USB_LP_CAN_RX0 (20)

- CTR: 正确的传输
- PMAOVR: 包溢出
- ERR: NANS / CRC / BST / FVIO
- RESET: 复位信号
- WKUP: 唤醒信号
- SOF: SOF包
- ESOP: SOF包丢失
- SUSP: 总线超过3ms无通信

USBWakeUp (42) —— 设备处于挂起模式时，由唤醒事件触发

STM32 USB 标准库函数 (1)

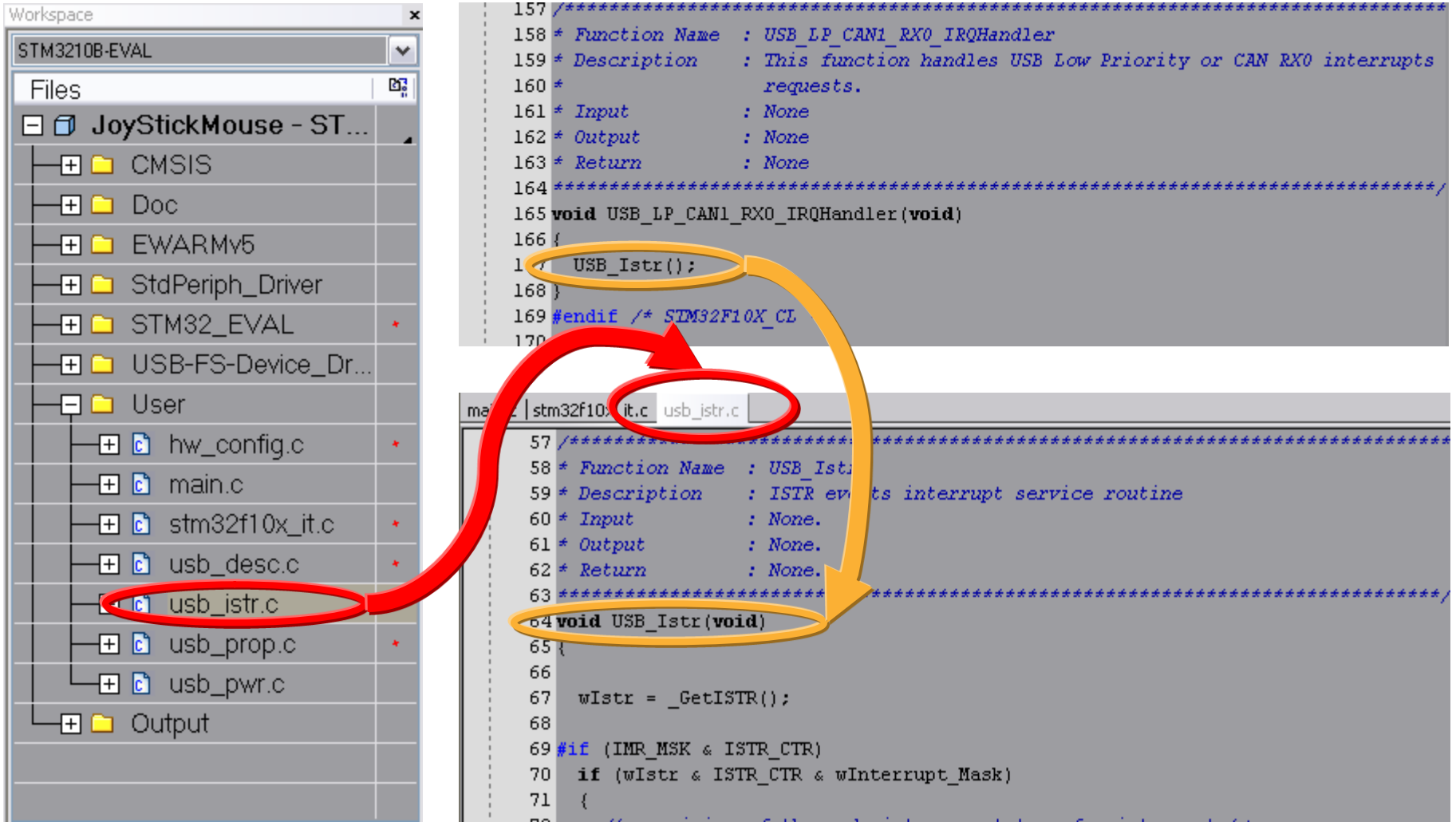


USB_LP_CAN_RX0 (20)

stm32f10x_it.c

```
152 /*******************************************************************************/
153 /*            STM32F10x Peripheral Interrupt Handlers                          */
154 /*******************************************************************************/
155
156 #ifndef STM32F10X_CL
157 /*******************************************************************************
158 * Function Name  : USB_LP_CAN1_RX0_IRQHandler
159 * Description    : This function handles USB Low Priority or CAN RX0 interrupts
160 *                  requests.
161 * Input          : None
162 * Output         : None
163 * Return         : None
164 *******************************************************************************/
165 void USB_LP_CAN1_RX0_IRQHandler(void)
166 {
167   USB_Istr();
168 }
169 #endif /* STM32F10X_CL */
170
171 /*******************************************************************************
```

STM32 USB 标准库函数 (2)



STM32 USB 标准库函数 (3)

```
void USB_Istr(void) {

 wIstr = _GetISTR();

...
#if (IMR_MSK & ISTR_RESET)
  if (wIstr & ISTR_RESET &
     wInterrupt_Mask) {
     SetISTR((u16)CLR_RESET);
    Device_Property.Reset();
#ifdef RESET_CALLBACK
    RESET_Callback();
#endif
  }
#endif
...
```

usb_conf.h :

```
#define IMR_MSK
CNTR_CTRM      | CNTR_WKUPM |
 CNTR_SUSPM    | CNTR_ERRM  |
 CNTR_SOFM     | CNTR_ESOFM |
 CNTR_RESETM
```

usb_pwr.c:

```
wInterrupt_Mask =
CNTR_RESETM | CNTR_SUSPM |
CNTR_WKUPM;
```

STM32 USB 标准库函数 (4)

```
void USB_Istr(void) {

 wIstr = _GetISTR();

...

#if (IMR_MSK & ISTR_CTR)
if (wIstr & ISTR_CTR &
    wInterrupt_Mask)
{
    CTR_LP();
    #ifdef CTR_CALLBACK
        CTR_Callback();
    #endif
}
#endif
```

usb_int.c:

```
void CTR_LP(void) {
...
    wEPVal =
    _GetENDPOINT(EPindex);

    if ((wEPVal & EP_CTR_RX) != 0) {
        (*pEpInt_OUT[EPindex-1])();
    }
    if ((wEPVal & EP_CTR_TX) != 0) {
        (*pEpInt_IN[EPindex-1])();
    }
...
}
```

STM32 USB 标准库函数 (5)

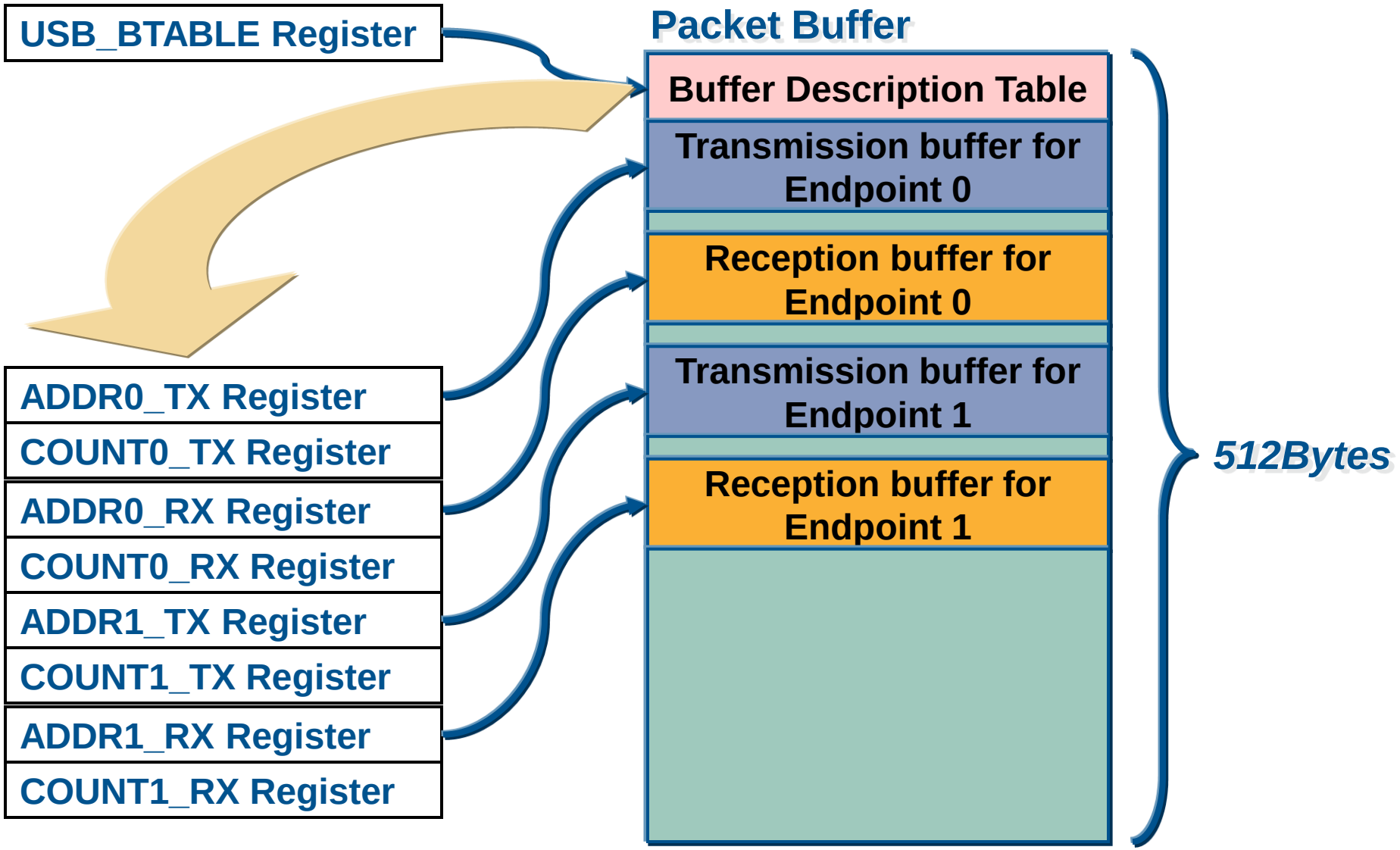
Usb_istr.c

```
void (*pEpInt_IN[7])(void) =
 {
   EP1_IN_Callback,
   EP2_IN_Callback,
   EP3_IN_Callback,
   EP4_IN_Callback,
   EP5_IN_Callback,
   EP6_IN_Callback,
   EP7_IN_Callback,
 };

void (*pEpInt_OUT[7])(void) =
 {
   EP1_OUT_Callback,
   EP2_OUT_Callback,
   EP3_OUT_Callback,
   EP4_OUT_Callback,
   EP5_OUT_Callback,
   EP6_OUT_Callback,
   EP7_OUT_Callback,
 };
```

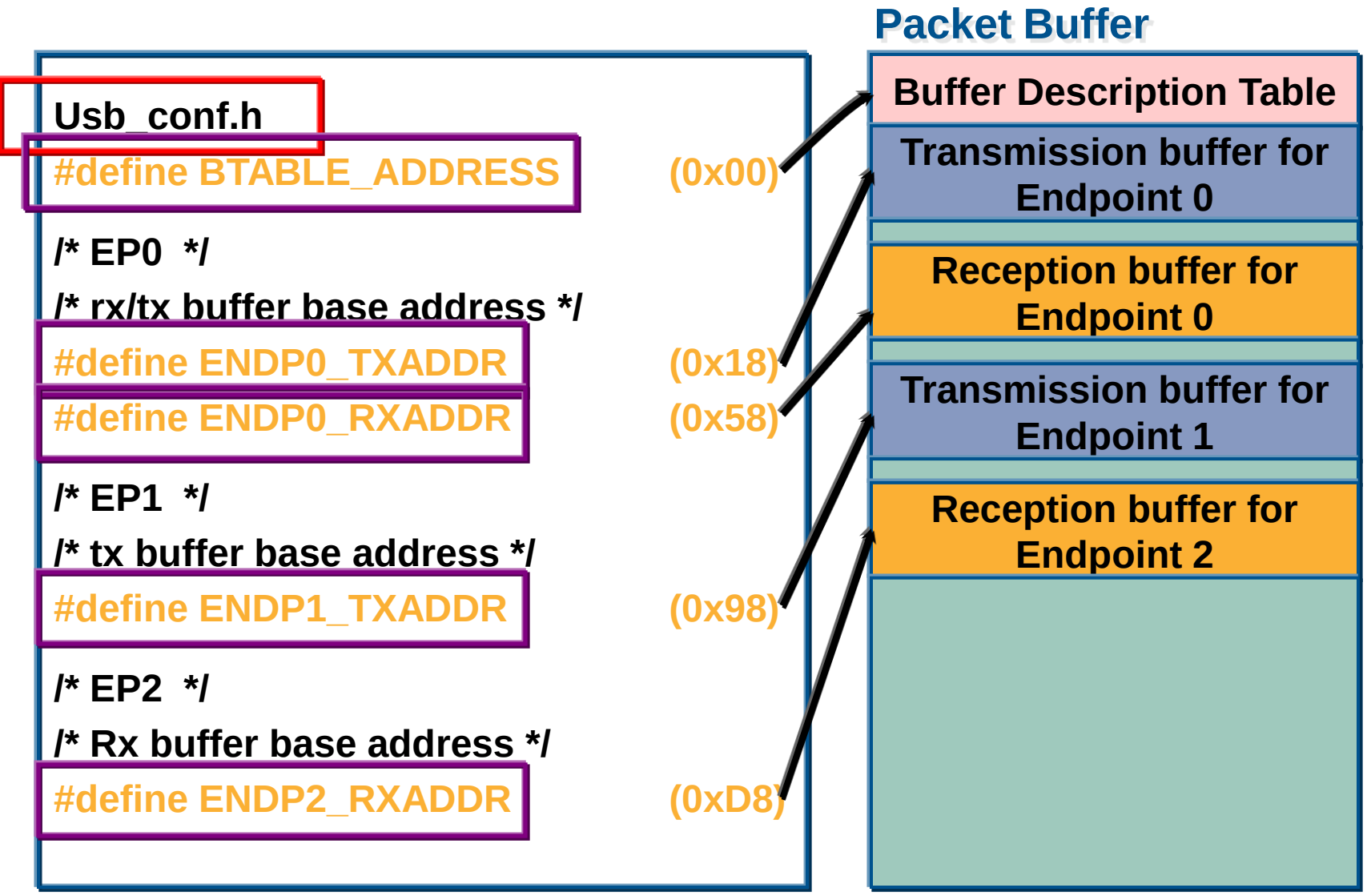
Usb_conf.h

```
#define  EP1_IN_Callback   NOP_Process
#define  EP2_IN_Callback   NOP_Process
#define  EP3_IN_Callback   NOP_Process
#define  EP4_IN_Callback   NOP_Process
#define  EP5_IN_Callback   NOP_Process
#define  EP6_IN_Callback   NOP_Process
#define  EP7_IN_Callback   NOP_Process

/*#define  EP1_OUT_Callback   NOP_Process*/
#define  EP2_OUT_Callback   NOP_Process
#define  EP3_OUT_Callback   NOP_Process
#define  EP4_OUT_Callback   NOP_Process
#define  EP5_OUT_Callback   NOP_Process
#define  EP6_OUT_Callback   NOP_Process
#define  EP7_OUT_Callback   NOP_Process
```

Packet Buffer & Buffer description table



Packet Buffer 的设置



实现一个USB设备的步骤：初始化

1

/* 根据应用定义需使用的端点数 */

usb_conf.h

```
#define EP_NUM     (2)
```

/* 这个数应该是所使用的非0的端点数+1 */

2

/* 初始化端点 */

usb_prop.c

```
SetEPType(ENDP1, EP_BULK);
SetEPTxAddr(ENDP1, ENDP1_TXADDR);
SetEPTxStatus(ENDP1, EP_TX_NAK);
SetEPRxStatus(ENDP1, EP_RX_DIS);
```

usb_regs.h:

```
#define EP_BULK          (0x0000)
#define EP_CONTROL       (0x0200)
#define EP_ISOCHRONOUS   (0x0400)
#defie   EP_INTERRUPT    (0x0600)
```

```
#define EP_TX_DIS        (0x0000)
#define EP_TX_STALL      (0x0010)
#define EP_TX_NAK        (0x0020)
#define EP_TX_VALID      (0x0030)
#define EP_RX_DIS        (0x0000)
#define EP_RX_STALL      (0x1000)
#define EP_RX_NAK        (0x2000)
#define EP_RX_VALID      (0x3000)
```

实现一个USB设备的步骤：使能端点

3

/* 使能需要的端点 */

对于IN端点：

```
USB_SIL_Write(EP1_IN, UserBuffer, Count);

SetEPTxStatus (ENDP1, EP_TX_VALID);
```

对于OUT端点：

```
SetEPRxStatus(ENDP1, EP_RX_VALID);
```

主机

IN

OUT

设备

实现一个USB设备的步骤：处理中断(1)

主机

设备

IN传输

IN Token

NAK

步骤3，使能端点

IN Token

DATA

ACK

产生CTR中断

调用EPx_IN_Callback

表示传输完毕，用户可以重复步骤3，发起下一次传输

实现一个USB设备的步骤：处理中断(2)

主机

设备

OUT传输

OUT Token

DATA

NAK

步骤3，使能端点

OUT Token

DATA

ACK

产生CTR中断

调用EPx_OUT_Callback

表示传输完毕，用户需要将数据从端点缓存区中拷贝至用户数组

实现一个USB设备的步骤：处理中断(3)

4

/ EPx_OUT/IN_Callback */*

IN传输:

```
EPx_IN_Callback(void):
    USB_SIL_Write(EP1_IN, UserBuffer, Count);
    SetEPTxStatus(ENDP1, EP_TX_VALID);
```

OUT传输:

```
EPx_OUT_Callback(void):
    Count =USB_SIL_Read(EP1_OUT, UserBuffer)
    SetEPRxStatus(ENDP1, EP_RX_VALID );
```

实现一个USB设备的步骤

1. 根据应用选择合适的USB类实现
2. 根据所选择的USB类协议，完成各个描述符)
   (包括设备描述符，配置描述符，接口描述符，
   端点描述符和字符描述符)
3. 根据描述符，初始化端点数目，
   分配各端点所需使用的Packet Buffer
4. 初始化所使用的端点，配置端点的传输类型，
   方向，Packet Buffer地址，和初始状态
5. 在需要发送或接收数据的时候，使能端点
6. 在该端点的中断回调函数中，处理数据，
   如果需要则使能下一次传输

Thank You !