**USB Device 设计文档**

**撰写人：陈灵佳**

**时间：2021年11月**

**文档更新记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **更新人** | **版本** | **备注** |
| 2021.11.8 | 陈灵佳 | 0.1 | 创建初稿 |
| 2021.12.7 | 陈灵佳 |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1 USB\_PHY 3](#_Toc90213871)

[1.1 USB PHY TX 5](#_Toc90213872)

[1.1.1 nrzi\_stuf\_encoder模块 6](#_Toc90213873)

[1.1.2 tx\_ctrl模块 7](#_Toc90213874)

[1.2 USB PHY\_RX 9](#_Toc90213875)

[1.2.1 de\_glitch模块 12](#_Toc90213876)

[1.2.2 信号采集/发送相位调整模块 12](#_Toc90213877)

[1.2.3 信号转换模块 13](#_Toc90213878)

[1.2.4 SYNC段检测模块 14](#_Toc90213879)

[1.2.5 非归零反相编码——填充位解码模块 15](#_Toc90213880)

[2. USB\_PROTOCALL 16](#_Toc90213881)

[2.1 packet\_de\_asemmbler 16](#_Toc90213882)

[2.1.1功能介绍 16](#_Toc90213883)

[2.1.2端口描述 16](#_Toc90213884)

[2.1.3实现原理 17](#_Toc90213885)

# 1 USB\_PHY

#### 框图

其模块框图如下所示：



#### 子模块描述

|  |  |  |  |
| --- | --- | --- | --- |
| 模块名 | | 功能 | |
| phy\_tx | nrzi\_stuf\_encoder | 将来自上层模块的数据进行非归零反相编码并发送到物理层 | 将来自上层模块的数据发送到物理层 |
| tx\_ctrl | 控制tx发送数据的流程 |
| phy\_rx | de\_glitch | 去除来自物理层的信号中小于等于一个phy\_clk宽度的毛刺 | 将来自物理层的数据发送到上层模块 |
| phase\_tune | 自适应来自物理层的数据，调整采样与发送的相位 |
| signal2state | 将0/1信号转换为总线状态 |
| sync\_detector | 检测同步段 |
| nrzi\_stuf\_decoder | 将来自物理层经过非归零反相编码后的数据解码 |
| reset\_syncer | | 同步系统复位信号至USB\_PHY时钟域 | |
| gen\_syncer | | 实现部分信号在系统时钟域和USB\_PHY时钟域的转换 | |

#### 接口描述

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 名称 | 信号描述 |
| 1 | clk\_i | system clock | 系统时钟域时钟 |
| 2 | rst\_n\_i | system reset | 系统时钟域复位信号，低有效 |
| 3 | phy\_clk\_i | phy clock | phy时钟域时钟，48MHz |
| 4 | phy\_rst\_n\_i | phy reset | phy时钟域复位信号，低有效 |
| 5 | rxdp\_i | 物理层总线输入 | 差分信号（positive） |
| 6 | rxdn\_i | 物理层总线输入 | 差分信号（negative） |
| 7 | en\_i | enable | 模块使能 |
| 8 | rx\_data\_o | rx data output | rx接收数据输出 |
| 9 | rx\_err\_o | rx data error | 为高时，rx收数发生错误 |
| 10 | rx\_active\_o | rx active | 为高时，rx busy |
| 11 | rx\_data\_vld\_o | 接收数据有效 | 为高时，rx收数有效 |
| 12 | rx\_data\_rdy\_i | rx data ready | 为高时，上层模块允许rx收数 |
| 13 | tx\_data\_i | tx data input | n byte,来自上层模块 |
| 14 | tx\_data\_vld\_i | tx data valid | 来自上层模块，为高时说明待发送数据有效 |
| 15 | tx\_data\_rdy\_o | tx data ready | 反馈给上层模块，为高时说明数据已发送 |
| 16 | txdp\_o | txdp | 发送到物理层总线差分中的positive |
| 17 | txdn\_o | txdn | 发送到物理层总线差分中的negative |
| 18 | tx\_en\_o | tx enable | phy\_\_tx模块使能 |

## 1.1 USB PHY TX

#### 框图

其框图如下图所示：

#### 功能描述

该模块将来自上层模块的数据，按照包格式即在DATA前加上同步段 ，在数据后加上两个SE0，并对DATA进行非归零反相编码后发往物理层差分总线。

#### 端口描述

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 名称 | 信号描述 |
| 1 | clk\_i | phy clock | phy时钟域时钟，48MHz |
| 2 | rst\_n\_i | phy reset | phy时钟域复位，低有效 |
| 3 | tx\_data\_i | tx data input | 来自上层模块的数据，n byte |
| 4 | tx\_data\_vld\_i | tx data valid | 来自上层模块，为高时说明待发送数据有效 |
| 5 | tx\_data\_rdy\_o | tx data ready | 反馈上层模块，为高时说明数据发送完毕 |
| 6 | en\_i | enable | 模块使能 |
| 7 | send\_en\_i | send enable | 来自phy\_rx中的phase\_tune，为高时说名tx可以发数 |
| 8 | txdp\_o | txdp | 发送数据至物理层差分总线的positive |
| 9 | txdn\_o | txdn | 发送数据至物理层差分总线的negative |
| 10 | tx\_en\_o | tx enable | 为高时，tx busy |
| 11 | rx\_en\_o | rx enable | 为高时，rx busy,与tx\_en互斥 |

#### 实现原理

通过tx\_ctrl将tx发数的任务分为三段：SOP,DATA,EOP以控制tx发数的流程。在各流程与nrzi\_stuf\_encoder进行握手，tx\_ctrl将待发送的数据发给nrzi\_stuf\_encoder，nrzi\_stuf\_encoder发送完毕后，反馈tx\_ctrl进入下一流程。

### 1.1.1 nrzi\_stuf\_encoder模块

#### 功能描述

该模块将SYNC,DATA,SE0发送至物理层差分总线，其中对DATA进行非归零反相编码。

#### 端口描述

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 名称 | 信号描述 |
| 1 | clk\_i | phy clock | phy时钟域时钟，48MHz |
| 2 | rst\_n\_i | phy reset | phy时钟域复位，低有效 |
| 3 | en\_i | enable | 模块使能 |
| 4 | send\_en\_i | send enable | 来自phy\_rx，为高时说明tx可以发数 |
| 5 | tx\_data\_updt\_i | tx data update input | 来自tx\_ctrl，待发送数据 |
| 6 | tx\_data\_updt\_vld\_i | tx data update valid | 来自tx\_ctrl，为高时说明待发送数据有效 |
| 7 | tx\_data\_updt\_rdy\_o | tx data update ready | 反馈tx\_ctrl，为高时说明tx发送数据完毕 |
| 8 | txdp\_o | txdp | 发送数据至物理层差分总线的positive |
| 9 | txdn\_o | txdn | 发送数据至物理层差分总线的negative |
| 10 | stuff\_en\_i | stuff bit enable | 来自tx\_ctrl，为高时说明tx将要发送DATA |
| 11 | se0\_en\_i | SE0 enable | 来自tx\_ctrl，为高时说明tx将要发送SE0 |
| 12 | sync\_en\_i | SYNC enable | 来自tx\_ctrl，为高时说明tx将要发送SYNC |
| 13 | send\_done\_o | send done | 来自nrzi\_stuf\_encoder，为高时说明tx发送SYNC或两个SE0或每字节DATA完毕 |

#### 实现原理

### 1.1.2 tx\_ctrl模块

#### 功能描述

该模块通过状态机控制tx向物理层总线发送数据的流程（IDLE,SOP,DATA,EOP），与上层模块（maybe FIFO）进行握手，锁存上层模块欲发送数据，与nrzi\_stuf\_encoder握手，传递其待发送数据（SYNC,DATA）。

#### 端口描述

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 名称 | 信号描述 |
| 1 | clk\_i | phy clock | phy时钟域时钟，48MHz |
| 2 | rst\_n\_i | phy reset | phy时钟域复位，低有效 |
| 3 | en\_i | enable | 模块使能 |
| 4 | tx\_data\_i | tx data input | 来自上层模块，n byte |
| 5 | tx\_data\_vld\_i | tx data valid | 来自上层模块，为高时说明待发送数据有效 |
| 6 | tx\_data\_rdy\_o | tx data ready | 反馈上层模块，为高时说明数据发送完毕 |
| 7 | tx\_data\_updt\_o | tx data update output | 去往nrzi\_stuf\_encoder，待发送数据（SYNC或DATA） |
| 8 | tx\_data\_updt\_vld\_o | tx data updata valid | 去往nrzi\_stuf\_encoder，为高时说明tx\_data\_updt有效 |
| 9 | tx\_data\_updt\_rdy\_i | tx data update ready | 反馈tx\_ctrl，为高时说明tx\_data\_updt已被nrzi\_stuf\_encoder发送 |
| 11 | send\_done\_i | send enable | 来自nrzi\_stuf\_encoder，为高时说明tx发送SYNC或两个SE0或每字节DATA完毕 |
| 12 | stuff\_en\_o | stuff bit enable | 去往nrzi\_stuf\_encoder，为高时说明tx将要发送DATA |
| 13 | se0\_en\_o | SE0 enbale | 去往nrzi\_stuf\_encoder，为高时说明tx将要发送SE0 |
| 14 | sync\_en\_o | SYNC enable | 去往nrzi\_stuf\_encoder,为高时说明tx将要发送SYNC |
| 15 | tx\_en\_o | tx enable | 为高时，tx busy,与tx\_en互斥 |
| 16 | rx\_en\_o | rx enable | 为高时，rx busy,与tx\_en互斥 |

#### 实现原理

##### 状态机实现

定义状态变量及状态跳转方向

以包格式对tx发数的任务进行划分：SYNC,DATA,SE0。由此定义状态变量及状态跳转方向如下图所示。



IDLE：tx空闲状态

SOP：tx发送SYNC状态

DATA：tx发送DATA状态

EOP：tx发送两个SE0状态

##### 规定状态跳转条件

IDLE跳SOP：上层模块向tx\_ctrl发出待发数有效信号的同时，nrzi\_stuf\_encoder告诉tx\_ctrl其上一次发数结束，现在可以发数（SYNC）

SOP跳DATA：nrzi\_stuf\_encoder告诉tx-ctrl SYNC发送完毕

DATA跳EOP：nrzi\_stuf\_encoder告诉tx\_ctrl DATA发送完毕，且上层模块告诉tx\_ctrl没有DATA待发送了。

EOP跳IDLE：nrzi\_stuf\_encoder告诉tx\_ctrl 两个SE0发送完毕。

状态跳转图如下图所示：



##### 编写状态机输出

状态机输出：tx\_data\_rdy\_o

tz\_data\_updt\_o

tx\_data\_\_updt\_vld\_o

se0\_en\_o

stuff\_en\_o

sync\_en\_o

tx\_en\_o

rx\_en\_o

## 1.2 USB PHY\_RX

#### 框图

其框图如下图所示：



#### x.x.1功能描述

该模块接收来自物理层的数据，在去除毛刺后对数据进行解码，最后以8bit并行输出至上层模块。同时与tx共享phase\_tune模块，发出send\_en供tx使用。

x.x.2端口描述

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 名称 | 信号描述 |
| 1 | clk\_i | phy时钟 | phy\_rx模块工作时钟 |
| 2 | rst\_n\_i | phy复位 | phy\_rx模块复位信号，低有效 |
| 3 | rxdp\_i | 物理层总线输入 | 差分信号（positive） |
| 4 | rxdn\_i | 物理层总线输入 | 差分信号（negative） |
| 5 | en\_i | 使能 | 模块使能 |
| 6 | send\_en\_o | 发送使能 | 为高时，tx可以发数 |
| 7 | rx\_data\_o | 已接收数据 | rx接收数据输出 |
| 8 | rx\_err\_o | 接收错误 | 为高时，rx收数发生错误 |
| 9 | rx\_active\_o | rx模块活跃 | 为高时，rx busy |
| 10 | rx\_data\_vld\_o | 接收数据有效 | 为高时，rx收数有效 |
| 11 | rx\_data\_rdy\_i | rx data ready | 为高时，上层模块允许rx收数 |

#### x.x.3实现原理

该模块包括de\_glitch、phase\_tune、signal2state、sync\_detect和nrzi\_stuf\_decoder子模块。其中de\_glitch将总线差分输入的glitch去除；phase\_tune根据差分输入调整phase以确保采样质量；采样后的信号由signal2state生成总线状态；sync\_detector以类似序列检测的方式检测同步段（SYNC）；当同步段检测通过后，nrzi\_stuf\_decoder开始工作，对DATA进行非归零反相编码的解码，最终得到以byte为单位的二进制的DATA，并将其发送至上层模块。

模块功能描述

x.1 CRC循环冗余校验码

x.1.1功能介绍

首先，CRC是一种优秀的检错码，即它只能检测错误不能纠正错误。它检测错误的过程是这样的，当我们发送数据时，以该数据产生一个被除数，在以某种约束选择一个固定的除数，这样我们就得到一个余数，我们将余数连同被除数一起发送出去，当接收端收到数据时，将加上余数的被除数再次除以除数，如果此时新余数为0，则说明发送没有错误，反之则有错误。所以CRC工作的过程就是做除法的过程，CRC码就是余数。

说的规范些就是，把比特流看作多项式的系数。设定一个生成多项式（generator polynomial）作为除数。数据流看作被除数。发送方需要在数据流末尾加上一段冗余码，使得组合后的新数据流能够整除除数。这段冗余码就是所谓的CRC。发送方计算好CRC后，把它加到末尾。然后接收方通过传过来的数据做除法计算余数，如果余数不为0，就说明有错误发生。

那么，如何计算CRC码？在数据流末尾补CRC长度的0，也就是在低位添加CRC长度的0，所谓CRC长度就是选定的多项式的阶数，然后做除法得到的余数就是了。只是...这个除法与普通的十进制除法不同，它是模2除法。

换句话说,CRC的计算就是：基于有限域GF(2)（即除以2的同余）的多项式环。总结一下，我们把数据流看作一个被除数，同时约定了一个除数，把通过计算获得一段对应的CRC码字。 然后呢，计算过程中的除法有点特殊，是模2除法。

以一个式子表达：原始数据左移m位（m为除数多项式的阶数） %（模2除法） 多项式 = CRC码

这个生成多项式（也就是除数）的选择是非常有讲究的，如果你的多项式选择很愚蠢，那么很可能你的检查范围可能只在最后几位。这个要展开讲就太复杂了。不同的生成多项式对应不同的规范。

CRC算法参数模型解释：

NAME:参数模型名称。

WIDTH:宽度，即生成的CRC数据位宽，如CRC-8，生成的CRC为8位。

POLY:十六进制多项式，省略最高位1，如x^8+x^2+x+1，二进制位100000111，省略最高位1，转换位十六进制为0x07。

INIT:CRC初始值，和WIDTH位宽一致。

REFIN:true或false，在进行计算之前，原始数据是否发生翻转。

REFOUT:true或false，运算完成后，得到的CRC值是否进行翻转。

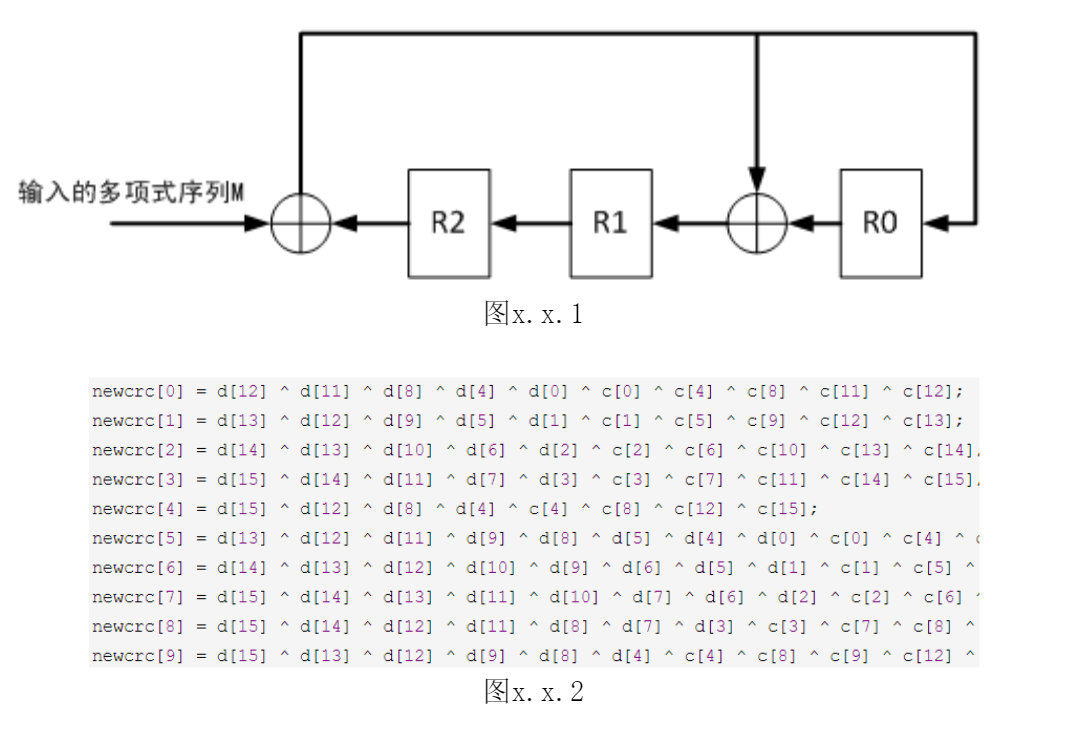
XOROUT:计算结果与此参数进行异或运算得到最终的CRC值，和WIDTH位宽一致。

x.x.2 接口描述

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 名称 | 信号描述 |
| 1 | clk\_i | 系统时钟 | 同步模块工作时钟 |
| 2 | rst\_n\_i | 系统复位 | 模块复位信号，低有效 |
| 3 | crc\_en\_i | CRC校验使能 | 为高时，模块开始CRC计算 |
| 4 | crc\_clr\_i | CRC校验清除 | 为高时，模块清除当前CRC计算结果 |
| 5 | data\_i | CRC待计算数据 | 需要进行CRC计算的数据（宽度为11bit或8bit） |
| 6 | crc\_o | CRC计算结果 | CRC结果输出（宽度与data\_i对应，5bit或16bit） |

x.x.3 实现原理

CRC计算需要“模2计算”，其可以转换为伽罗华域运算规则进行计算，而LFSR现行反馈移位寄存器亦满足伽罗华域运算规则，故可以通过LFSR实现CRC计算规则；模块采用并行线性反馈移位寄存器，并通过异或关系完成反馈电路功能。



### 1.2.1 de\_glitch模块

#### x.2.1 功能介绍

物理层总线上的信号因为种种原因可能会产生glitch，该模块将小于等于一个phy\_clk 宽度的脉冲（USB中的glitch都为0→1）视为glitch，并将其去除，并且可以恢复信号的宽度，不因de\_glitch的操作而缩短或拉长。

#### x.2.2 端口描述

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 名称 | 信号描述 |
| 1 | clk\_i | phy时钟 | de\_glitch模块工作时钟 |
| 2 | rst\_n\_i | phy复位 | 模块复位信号，低有效 |
| 3 | signal\_i | 待de\_glitch的输入信号 | 已同步，glitch宽度为1个时钟cycle，有效信号宽度大于等于2个时钟cycle |
| 4 | signal\_o | 已de\_glitch的信号输出 | de\_glitch完成，信号宽度应保持不变 |

#### x.2.3 实现原理

##### x.2.3.1 去除glitch

由于信号已经经过同步，所以可以通过将signal\_i打一拍得到signal\_d0，再打一拍得到signal\_d1，然后将这两个信号相与，得到signal\_de\_glitch，从而去除glitch。

##### x.2.3.2 还原信号宽度

根据x.2.3.1的描述，可以去除glitch，但是因此信号的宽度会有所损失。通过将signal\_de\_glitch打一拍得到signal\_de\_glitch\_d0，再将二者相或，得到宽度还原的signal\_o。

### 1.2.2 信号采集/发送相位调整模块

#### x.3.1 功能描述

状态变量PHASE\_0,PHASE\_1,PHASE\_2,PHASE\_3由clk\_i控制。理想情况下，输入observed\_signal\_i应该与phase对齐，但由于电气特性，phase会存在偏差，所以需要phase\_tune。该模块通过修改状态机的phase来适应输入信号obseerved\_signal\_i。

#### x.3.2 端口描述

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 名称 | 信号描述 |
| 1 | clk\_i | phy时钟 | phase\_tune模块工作时钟，48MHz |
| 2 | rst\_n\_i | phy复位 | 模块复位信号，低有效 |
| 3 | en\_i | 相位调整使能 | 为高时，模块开始相位调整 |
| 4 | observed\_signal\_i | 待采集的信号 | 此信号在何phase toggle的不确定性导致了phase调整的必要性 |
| 5 | sample\_en\_o | 采集使能信号 | 为高时，开始采集 |
| 6 | send\_o | 发送使能信号 | 为高时，开始传输 |

#### 1.2.3 实现原理

具体说明，状态机是跟着clk\_i的，是死的，所以sample\_en\_o和send\_en\_o也是死的，它们分别在phase\_1和phase\_3拉高。而observed\_signal\_i是活的，所以我们需要根据observed\_siganl\_i来调整phase，还原理想情况。观察一帧observed\_signal\_i（4个phy\_clk周期），称它的第一个toggle沿为起始沿，那么它的起始沿会落入那个phase，一共有四种情况。当起始沿落入phase\_0时，此为理想情况，不需要phase\_tune。而当起始沿落入其他三个phase的时候，phase\_tune模块开始工作。

让phase适应observed\_signal\_i，强行还原理想情况。那么理想情况是什么样的。数据的起始对应phase\_0，结尾对应phase\_3。而自适应就意味着先发现问题，再解决问题，也就是说，只有发现数据的起始不对应phase\_0，我们才能接着采取措施，换言之，数据的起始不对应phase\_0已经发生了，我们要做的是强行让下一个phase是phase\_1，然后让phase正常跳转即可。

### 1.2.3 信号转换模块

#### x.5.1 功能描述

该模块根据总线rxdp和rxdn上的0/1值，输出四种状态，分别是J,K,SE0,SE1。

#### x.5.2 端口描述

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 名称 | 信号描述 |
| 1 | rxdp\_i | 总线输入 | 差分信号positive端 |
| 2 | rxdn\_i | 总线输入 | 差分信号negtive端 |
| 3 | j\_o | J状态输出 | 为高时，总线为J态 |
| 4 | k\_o | K状态输出 | 为高时，总线为K态 |
| 5 | se0\_o | SE0状态输出 | 为高时，总线为SE0态 |
| 6 | se1\_o | SE1状态输出 | 为高时，总线为SE1态 |

#### x.5.3 实现原理

|  |  |  |
| --- | --- | --- |
| rxdp | rxdn | state |
| 1 | 0 | J |
| 0 | 1 | K |
| 0 | 0 | SE0 |
| 1 | 1 | SE1 |

根据上方表格中描述的对应关系，将rxdp和rxdn进行逻辑运算得到state。

### 1.2.4 SYNC段检测模块

#### x.4.1 功能描述

检测SYNC段，本模块检测的序列已经过signal2state,所以不是0/1而是state（J,K,SE0,SE1）。当检测输入为K,J,K,J,K,J,K,K或K,J,K,J,K,K时，输出同步段检测成功信号，否则输出检测同步段失败信号，等待下一次检测。

#### x.4.2 端口描述

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 名称 | 信号描述 |
| 1 | clk\_i | phy时钟 | sync\_detector模块工作时钟 |
| 2 | rst\_n\_i | phy复位 | 模块复位信号，低有效 |
| 3 | en\_i | 模块使能 | 为高时，模块开始检测SYNC段 |
| 4 | sample\_en\_i | 采集使能 | 来自模块phase\_tune，为高时，采集信号 |
| 5 | j\_i | 总线输入 | 为高时，总线为J态 |
| 6 | k\_i | 总线输入 | 为高时，总线为K态 |
| 7 | sync\_acpt\_o | 同步段检测成功 | 为高时，同步段检测成功 |
| 8 | sync\_err\_o | 同步段检测失败 | 为高时，同步段检测失败 |

#### x.4.3 实现原理

该模块检测SYNC段，SYNC段为8’b1000\_0000，由于是LSB，所以对于RX来说，接收到的应是0000\_0001，又由于非归零反向编码，所以实际检测的是K,J,K,J,K,J,K,K。当检测到K,J,K,J,K,J,K,K，输出同步段检测成功信号。反之，输出同步段检测失败信号。特别的，将检测到K,J,K,J,K,K也视为检测同步段成功。因为K,J,K,J,K,K对应的同步段为6’b10\_0000，而5个零足以让模块进行phase\_tune，所以该情况亦可。具体实现类似于序列检测。

### 1.2.5 非归零反相编码——填充位解码模块

#### x.6.1 功能描述

该模块在同步段检测成功后工作，将来自物理层的数据进行非归零反相编码的解码，并将解码得出的 data发送至上层模块。

#### x.6.2 端口描述

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 名称 | 信号描述 |
| 1 | clk\_i | phy时钟 | 模块工作时钟，48MHz |
| 2 | rst\_n\_i | phy复位 | 模块复位信号，低有效 |
| 3 | en\_i | 模块使能 | 为高时，模块开始解码 |
| 4 | sample\_en\_i | 采集使能 | 来自模块phase\_tune，为高时，采集信号 |
| 5 | sync\_acpt\_i | 同步段检测成功 | 为高时，同步段检测成功 |
| 6 | rxd\_i | 总线输入 | 为高时，总线为J态；为低时，总线为K态 |
| 7 | se0\_i | SE0状态 | 为高时，总线为SE0态 |
| 8 | rx\_data\_o | rx输出 | 8bit，LSB |
| 9 | rx\_data\_vld\_o | rx输出有效 | 为高时，rx输出有效 |
| 10 | rx\_data\_rdy\_i | rx待采集数据准备 | 为高时，rx待采集数据已准备 |
| 11 | rx\_active\_o | rx工作中 | 为高时，rx正在工作 |
| 12 | sync\_err\_i | 同步段检测错误 | 为高时，同步段检测错误 |
| 13 | rx\_err\_o | rx错误 | 为高时，rx发生错误。可能是同步段检测错误，可能是填充位错误，可能是bit数错误 |

#### x.6.3 实现原理

##### 解码

以传输8’hbf为例，由于非归零反相编码和LSB，所以同步段8’b10000\_0000和DATA 8’b1011\_1111在物理层差分总线上呈现为rxdp，rxdn如下图所示。经过signal2state模块后，0/1信号转换为总线状态，如rxd所示。将rxd打一拍之后再与自身做同或，得出解码后的数据，如de\_nrzi所示。



##### 输出数据

使用one\_cnt对de\_nrzi进行连续个‘1’的计数，当计数到6的时候，拉高flag信号stuf\_bit。rx在收取de\_nrzi并整合成rx\_data\_o的过程中避开stuf\_bit拉高时的de\_nrzi，从而得到剔除填充位的数据。

# 2. USB\_PROTOCALL

## 2.1 packet\_de\_asemmbler

### 2.1.1功能介绍

该模块将来自usb\_phy\_rx的数据分析出PID字段、数据字段和CRC字段。将DATA packet中的数据字段发往usb\_endpoint。分别计算CRC5与CRC16并与参考值进行比较，将比较结果上报。

### 2.1.2端口描述

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 名称 | 信号描述 |
| 1 | clk\_i | phy clock | phy时钟域时钟，48MHz |
| 2 | rst\_n\_i | phy reset | phy时钟域复位信号，低有效 |
|  | | | |
| 3 | en\_i | enable | 为高时，模块开始工作 |
| PHY RX | | | |
| 4 | rx\_data\_i | rx data | [7:0]，来自usb\_phy\_rx |
| 5 | rx\_data\_vld\_i | rx data valid | 为高时，rx data 有效 |
| 6 | rx\_data\_rdy\_o | rx data ready |  |
| 7 | rx\_err\_i | rx err | 为高时，rx收数有误 |
| 8 | rx\_active\_i | rx active | 为高时，rx busy |
| PID | | | |
| 9 | pid\_IN\_o | PID IN | 为高时，当前接收的是IN包 |
| 10 | pid\_OUT\_o | PID OUT | 为高时，当前接收的是OUT包 |
| 11 | pid\_DATA0\_o | PID DATA0 | 为高时，当前接收的是DATA0包 |
| 12 | pid\_DATA1\_o | PID DATA1 | 为高时，当前接收的是DATA1包 |
| 13 | pid\_ACK\_o | PID ACK | 为高时，当前接收的是ACK包 |
| 14 | pid\_chk\_err\_o | PID check error | 为高时，PID检测未知 |
| DATA FIELD IN DATA PACKET | | | |
| 15 | rx\_data\_st\_o | rx data store | 需要发往usb\_endpoint的DATA0/1包中的数据字段 |
| 16 | rx\_data\_st\_vld\_o | rx data store valid | rx\_data\_store\_o 有效 |
| 17 | crc16\_err\_o | CRC16 error | 为高时，CRC16错误 |
| DATA FIELD IN TOKEN PACKET | | | |
| 18 | token\_addr\_o | TOKEN ADDR | TOKEN包中的ADDR域 |
| 19 | token\_endp\_o | TOKEN ENDP | TOKEN包中的ENDP域 |
| 20 | token\_vld\_o | TOKEN valid | ADDR/ENDP 有效 |
| 21 | crc5\_err\_o | CRC5 error | 为高时，CRC5错误 |

### 2.1.3实现原理

根据协议规定，包格式如下图所示：



而usb\_phy\_rx已经将SYNC和EOP剔除，发出rx\_data\_o，所以rx\_data\_o中只包含



该模块的目标就是将这些字段从rx\_data\_o中分解出来：



说明：为了简化设计，TOKEN packet只设计IN/OUT packet。且由于本设计为USB DEVICE，所以只会收到HANDSHAKE packet中的ACK。

根据上图所示，抽象程度最高的是PID类型，所以设计者将PID作为状态变量。在各状态下，对抽象程度低一些的字段进行处理。

状态变量如下图所示：



说明：1. ACTIVE作为非PID类型，添加它的作用是在ACTIVE状态下查看PID类型，这样才能决定下一个状态跳往哪一种PID。2. 由于本设计为USB DEVICE，所以对于HOST发来的ACK,系统只会回到IDLE，没有其他操作，所以索性在ACTIVE下检测出PID为ACK时直接跳转到IDLE，故省略状态ACK。

状态跳转条件

|  |  |  |
| --- | --- | --- |
| 现态 | 跳转条件 | 次态 |
| IDLE | rx\_data\_vld\_i | ACTIVE |
| ACTIVE | pid\_ACK || rx\_err || pid\_chk \_err | IDLE |
| pid\_IN ||pid\_OUT | TOKEN |
| pid\_DATA0 || pid\_DATA1 | DATA |
| TOKEN | rx\_err || !rx\_active | IDLE |
| DATA | rx\_err || !rx\_active | IDLE |



UNDER ACTIVE

在ACTIVE状态下，我们需要查看PID类型，实现状态跳转并输出PID类型。

首先，为了方便处理，我们需要对rx\_data对应PID的部分进行store。由于PID字段是8bit，且是packet的第一个字段，所以一个frame中的第一个rx\_data一定是PID字段。

得到PID后，我们将PID的值与协议规定的值进行比较，得出PID类型或者PID错误。

UNDER TOKEN

当PID是IN或OUT时，状态由ACTIVE跳转至TOKEN。在TOKEN下，我们需要从rx\_data中拆解出数据字段：ADDR（7 bit）和ENDP（4 bit），还有CRC字段：CRC5（5 bit）。

由于TOKEN（IN/OUT）包的数据字段是定长（2个byte），所以我们关注的是处于TOKEN状态的两个rx\_data。

