《数字通信基础》

[《数字通信基础》 1](#_Toc182391484)

[第一节：线路编码 4](#_Toc182391485)

[1.1不归零码 4](#_Toc182391486)

[1.1.1不归零码编码应用 5](#_Toc182391487)

[1.2归零码 6](#_Toc182391488)

[1.3其它高级编码 7](#_Toc182391489)

[1.3.18B/10编码原理 8](#_Toc182391490)

[1.3.1.1编码表 8](#_Toc182391491)

[1.3.1.2运行一致性计算 10](#_Toc182391492)

[1.4 8B/10B编码器HDL模型 10](#_Toc182391493)

[1.4.13B4B译码电路设计与实现 11](#_Toc182391494)

[1.4.2 5B6B译码电路设计与实现 12](#_Toc182391495)

[1.4.3 运行一致性计算电路程序实现 13](#_Toc182391496)

[1.5 8B10B编码模型验证 14](#_Toc182391497)

[第二节：SOC总线 14](#_Toc182391498)

[2.1SOC总线通信规则 15](#_Toc182391499)

[2.1.1下游无反制通信 15](#_Toc182391500)

[2.1.2下游无反制通信 15](#_Toc182391501)

[2.1.3下游快速反制通信 15](#_Toc182391502)

[2.2AXI总线 16](#_Toc182391503)

[2.2.1 axi mm 16](#_Toc182391504)

[2.2.2AXI St 18](#_Toc182391505)

[2.3AXI ST例子 19](#_Toc182391506)

在电路设计中，数据都是通过数字信号传递，通过总线或者接口传递数据，根据不同的应用形式可将柱子通信进行分类。

按照传输数据的形式分类：

**单比特：**一个比特一个比特传输，又叫做串行传输，

缺点：

如果在同样的速率情况下，一个比特一个比特的传输，相对于多比特传输慢。

优点：

由于只有一根信号线，不存在竞争冒险，信号传递相对更稳定，并且有利用同步，尤其传输数据速率较高的情况下，可将数据和时钟进行编码传输。

多比特：多个比特同时传输，又叫做并行行传输，

缺点：存在竞争冒险，速率较高的情况，传输距离较远的时，数据误码率高。

优点：可以在慢时钟下传输

在实际应用中，如果是片外数字通信，一般是使用串行传输，如果是片上数字通信，一般选择并行传输。

按照驱动电路分类：

低速传输：驱动电路通过普通的驱动电平，比如LVCOS3.3 , LVCOS2.5 ….

高速传输 :驱动电路需要为高速驱动电路，比如：TGX GTP, GTZ TGH，ETH phy ,usb phy

按照通信模式分类;

主从传输：通信系统中有主电路和从电路，整个通信由主机控制

对等传输：通信系统中所有电路之间都可以控制通信

# 第一节：线路编码

在数字通信中，当通信速率较高的情况下，片外传输采用并行，误码率较高，所以按照串行方式传输，所以现在的片外高速通信都采用串行传输。

当串行传输，如果采用异步传输，存在高频采样问题，现实需要采用同步传输，同步传输需要时钟信号，时钟信号如果直接在硬件上通过线传递，这样很难让时钟和数据之间延迟保持一致，解决这一问题的方法就是编码

发送端：线路编码

目前高速接口都有自己的编码方式，比如USB2.0,nrzi编码，USB3.0 8B/10B编码，PCIE2.0 8B/10B ,pcie 3.0 128/130编码

接收端: cdr

由于接收端需串行接收时钟，时钟和数据需要保持一致，根据编码特性，从编码中进行时钟同步，从而恢复数据，本质上用PLL实现。

## 1.1不归零码

不归零码通过2个驱动电平，电平要求如下：

单极性：

正电平：+v

零电平 ：0

0

1

1

0

1

Y（v）

X（ns）

0

0

0

1

1

0

1

Y（v）

X（ns）

双极性

正电平：=V

负电平：-v

### 1.1.1不归零码编码应用

不归零码常见的应用有NRZI，和曼彻斯特编码

1. NRZI编码

USB2.0采用的编码形式，是常见的一种不归零码，一般我们通过电平大小表示逻辑值，而NRZI通过电平状态便是逻辑值，

保持：表示逻辑值1

翻转：表示逻辑值0

X（ns）

Y（v）

1

0

1

0

1

Y（v）

X（ns）

1

0

1. 曼彻斯特编码

曼彻斯特编码是不归零码的一种，用于光纤以太网，它也使用电平状态表示逻辑值

0-1：表示逻辑值0

1-0：表示逻辑值1

0

1

0

1

## 1.2归零码

归零码需要3个电平驱动，如下：

单极性：

正电平：-v

零电平：0

零电平：0

Y（v）

Y（v）

1

0

0

1

0

1

X（ns）

X（ns）

双极性：

正电平：-v

零电平：0

负电平：0

0

0

1

## 1.3其它高级编码

以上编码不管采用什么形式，都有很大的瑕疵，我们力求完美，绝对完美的编码是很难找到的，但是可以寻求一种平衡的完美编码，比如8B/10B

这里我们介绍8B10B编码，简单来说就是将8比特的数据编码位10比特的数据，有效率占80%。

以前的基本编码都是按照常规的思维，我们换一种思维，IBM公司的人员经过研究发现，从1024个数中寻找256个数作为基本编码，这个256个数是按照一定的排列组合，从而达到完美平衡，这个排列组合是工程师通过比对寻找得出的结果。

由于基本通信是按照字节通信，所以选择以字节为单位进行编码，直接从10bit数中寻找平衡规律难度稍等，可以将8位数据分成2部分，分别是3位和5位，把3位编码4位，相当于从16个数中寻找8个数，另外把5位编码位6位，从64个数中寻找32个数。这样相对更简单。

### 1.3.18B/10编码原理

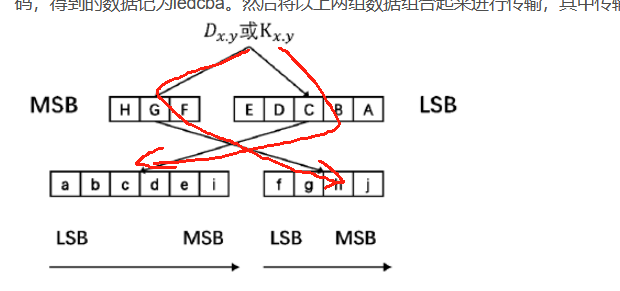
#### 1.3.1.1编码表

8B/10B = 3B/4B + 5B/6B

比如：data[7:0]

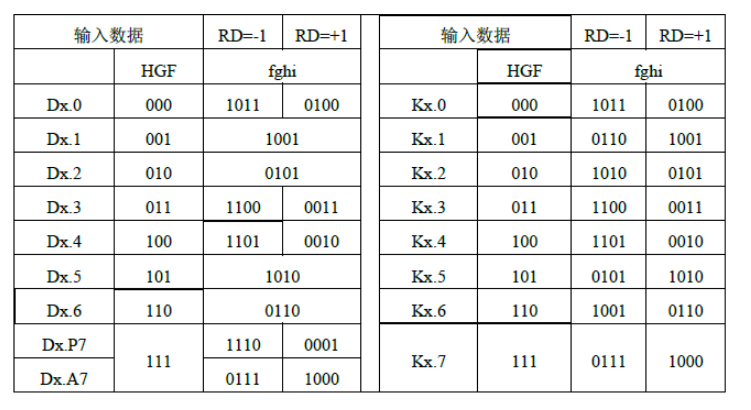
Data[7:5] :按照3B/4B编码

Data[4:0] :按照5B/6B编码



数据编码

控制编码





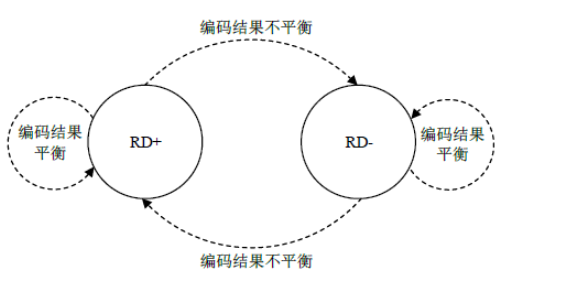
比如data = 8’b10000\_001

rd= -rd

code = 10’b1011\_011011

#### 1.3.1.2运行一致性计算

运行一致性是为了DC平衡，如果前一个码10比特数中，1比0多，为rd+，否则rd-，在实际使用中我们需要按照要求计算rd的极性。



## 1.4 8B/10B编码器HDL模型

要求：

对有效数据进行编码，我们只对数据进行编码

rst\_n

rst\_n

rst\_n

clk

clk

clk

rd\_vanue

case\_5b6b

case\_3b4b

Code\_8b10b

dout[3:0]

dout[9:4]

din[1:0]

din[7:5]

rd

valid\_din

din[7:0]

flag\_dout

### 1.4.13B4B译码电路设计与实现

rst\_n

clk

dout[3:0]

din[7:5]

case\_3b4b

valid\_din

rd

|  |  |
| --- | --- |
| 信号名 | 描述 |
| 系统相关 | |
| clk | 编码器时钟 |
| rst\_n | 系统复位 |
| 编码数据输入 | |
| valid\_din | 当==1表示din[7:5]有效 |
| din[7:5] | 被编码数据 |
| 编码输出 | |
| dout[3:0] | 编码数据 |
| 编码极性 | |
| rd | Rd  0:表示rd-  1:表示rd+ |

### 1.4.2 5B6B译码电路设计与实现

rd

rst\_n

clk

dout[3:0]

din[7:5]

case\_5b6b

valid\_din

Flag\_dout

|  |  |
| --- | --- |
| 信号名 | 描述 |
| 系统相关 | |
| clk | 编码器时钟 |
| rst\_n | 系统复位 |
| 编码数据输入 | |
| valid\_din | 当==1表示din[7:5]有效 |
| din[4:0] | 被编码数据 |
| 编码输出 | |
| dout[9:4] | 编码数据 |
| flag\_dout | 当flag\_dout  ==1表示dout有效 |
|  |  |
| 编码极性 | |
|  | |
| rd | Rd  0:表示rd-  1:表示rd+ |

### 1.4.3 运行一致性计算电路程序实现

dout[7:0]

rd

clk

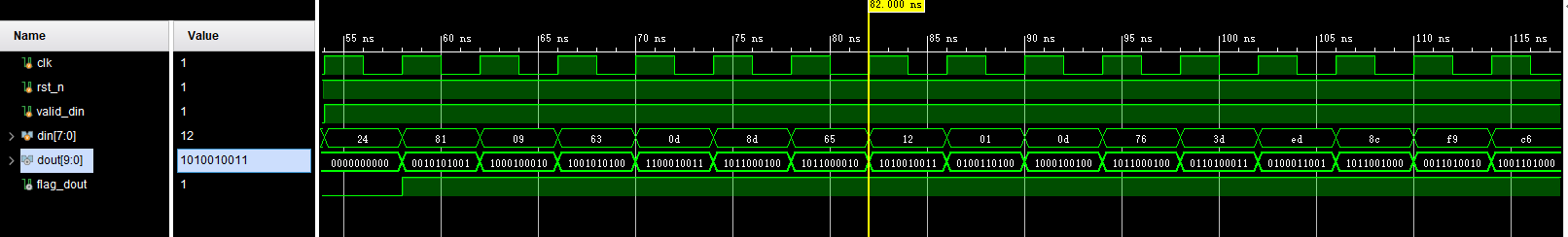
rd\_vanue

rst\_n

|  |  |
| --- | --- |
| 信号名 | 描述 |
| 系统相关 | |
| clk | 编码器时钟 |
| rst\_n | 系统复位 |
| 编码输出 | |
| dout[9:4] | 编码数据 |
| flag\_dout | 当flag\_dout  ==1表示dout有效 |
|  |  |
| 编码极性 | |
|  | |
| rd | Rd  0:表示rd-  1:表示rd+ |

## 1.5 8B10B编码模型验证

本设计为并行编码器电路模型，非串行编码器电路，验证结果如下;



# 第二节：SOC总线

随着数字电路设计的高度集成化，并随着SOC的发展，数字电路设计的硬件系统越来越复杂，就需要一种公共数字通信的传输总线（SOC总线），目前比较流行的总线有**AMBA总线，Wishbone总线，Avalon总线**，对于FPGA开发而言，以及ASIC开发，最为流行的是AMBA总线（AXI）。

Soc总线作为片上系统互联的总线，采用并行主从传输，并且单主系统。

## 2.1SOC总线通信规则

总线或者接口是数字通信的具体表现，每一种总线或者接口都有自己的通信规则或者协议，作为SOC总线，我们希望它是通用的，因此需要寻求一种通用的数字通信规则，下来我们将通信规则进行分析

Souce

上游

Sink

下游

valid

data[k-1:0]

ready

### 2.1.1下游无反制通信

1. 上游任何时候可以发送数据，valid==1表示发送数据有效
2. 下游只要检测到valid==1就接收数据

### 2.1.2下游无反制通信

1. 上游检测ready==1的情况下，采可以发送数据
2. 当上游可以发送数据的时候，valid==1表示发送数据有效
3. 当下游ready=1的时候检测到valid==1开始接收数据，直到数据接收完成（valid==0）

### 2.1.3下游快速反制通信

1. 上游任何时刻都可以发数据，valid==1表示发送数据有效
2. 下游任何时刻都可以反制，ready=0反制
3. 只有当valid==1，ready=1发送的数据采会被真正的接收

以上是SOC常用的数字通信规则，AXI作为目前最流行的片上总线，采用的是下游快速反制通信规则。

## 2.2AXI总线

AXI作为通用的SOC总线，标准的总线采用32位或者64位的并行主从传输总线，根据应用需求设计了**2种总线，分别axi mm ,aix st**.

axi mm : **针对随机读写而设计的总线**，支持突发以及一些软件特性，在设计中有的时候我们需要突发，有的时候不需要突发，当不需要突发的时候我们可以axi mm总线

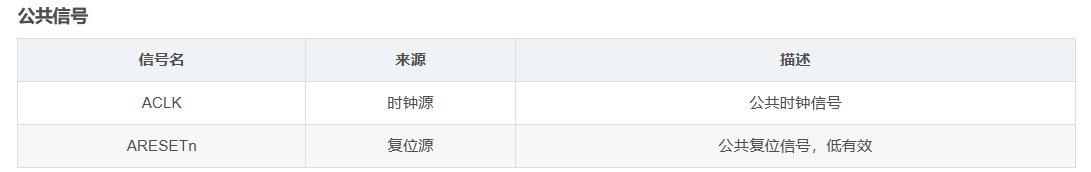
axi mm full :支持突发的随机读写总线，主要用来读写大批量数据，比如存储器

axi mm lite :不支持突发的随机读写总线，主要用来读单个数据，比如寄存器

axi st ：针对顺序读写而设计的总线。一般读写fifo

### axi mm

AXI mm 采用独立通道设计，采用了5个通道进行数据读写



1. 写操作（3个通道）

写地址通道



软件特性

写数据通道



写响应通道



1. 读操作（2个通道）

读地址通道



软件特性

读数据通道



### 2.2.2AXI St



## 2.3AXI ST例子

