**1、并行扰码/解扰码器的设计 3**

**1.1设计任务和要求 3**

**1.1.1、扰码/解扰码器并行算法原理； 3**

**1.1.2、扰码/解扰码器并行算法的Verilog HDL程序设计。 3**

**1.2、扰码/解扰码器并行算法原理 3**

**1.2.1、并行加扰器设计 3**

**1.2.2、并行解扰器设计 6**

**1.3、仿真输出波形 6**

**1.4、并行扰码/解扰码器的Verilog HDL程序 7**

**1.4.1、扰码器的程序 7**

**1.4.2、解扰码器的程序 7**

**1.4.3、测试文件的Verilog HDL程序 8**

**2、CRC\_16并行算法的Verilog HDL程序设计 9**

**2.1、设计任务和要求 9**

**2.1.1、CRC\_16并行算法原理 9**

**2.1.2、CRC16并行算法的Verilog HDL程序设计。 9**

**2.2、CRC\_16并行算法原理 9**

**2.2.1、CRC编码原理 9**

**2.2.2、CRC\_16并行算法原理 10**

**2.3、CRC\_16并行仿真输出波形 13**

**2.4、CRC\_16并行算法的Verilog HDL程序 13**

**2.4.1、CRC\_16校验码的并行编码的程序 13**

**2.4.2、CRC\_16测试文件程序 15**

# **1、并行扰码/解扰码器的设计**

# **1.1设计任务和要求**

## **1.1.1、扰码/解扰码器并行算法原理；**

## **1.1.2、扰码/解扰码器并行算法的Verilog HDL程序设计。**

# **1.2、扰码/解扰码器并行算法原理**

在数字通信系统中，若经常出现长的“0”或“1”系列，将会影响位同步的建立和保持。为了解决这个问题以及限制电路中存在的不同程度的非线性特性对其他电路通信造成的串扰，要求数字信号的最小周期足够长。将数字信号变成具有近似于白噪声统计特性的数字序列即可满足要求，这通常用加扰来实现。而所谓加扰，就是不用增加冗余而扰乱信号，改变数字信号统计特性，使其具有近似白噪声统计特性的一种技术。这种扰乱是有规律可循的，因此也是可以解除的。

扰码的作用就是对输入的信号进行随机化处理，以减少数据的连0连1数目，确保接收端的位同步提取，并同时扩展基带信号频率，起到加密效果。这种加扰的基础建立在伪随机序列理论的基础上，其工作原理就是在发送端用加扰来改变原来数字的统计特性，而在接收端用解扰器恢复原始的数字信号。

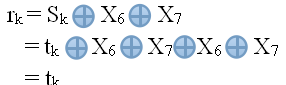
扰码产生是通过循环移位寄存器来实现的，而扰码生成多项式决定循环移位寄存器的结构。这次设计以ITU-T推荐的SDH（同步数字体系）使用的扰码生成多项式：

X7 + X6 + 1

设扰码的输入数字序列为tk ，输出为Sk ；解码器的输入为Sk ，输出为rk 。

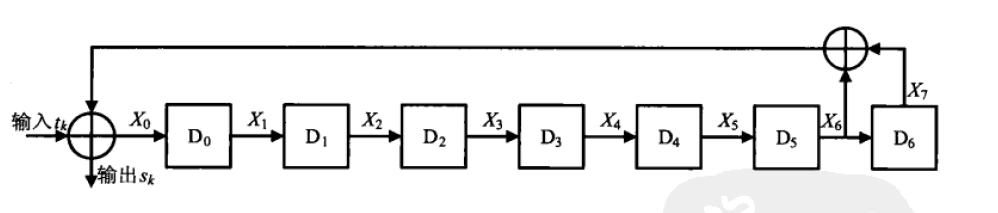
扰码器的输入和输出序列关系为：

解扰码器的输入和输出序列关系为：

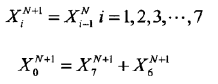


## ***1.2.1、并行加扰器设计***

在高速的数据传输过程中，为了满足数据高速处理的需要，一般扰码器和解码器通常采用并行的方式来实现。并行扰码的原理基于特征多项式X7 + X6 + 1，加扰器电路结构如图所示。



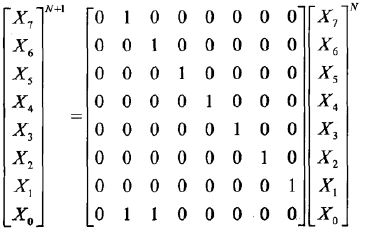
假设每个触发器的输出端分别是假设为最低位触发器的输入，那么每个触发器的关系如下

：

令i=7以及i=6代入上式可得：



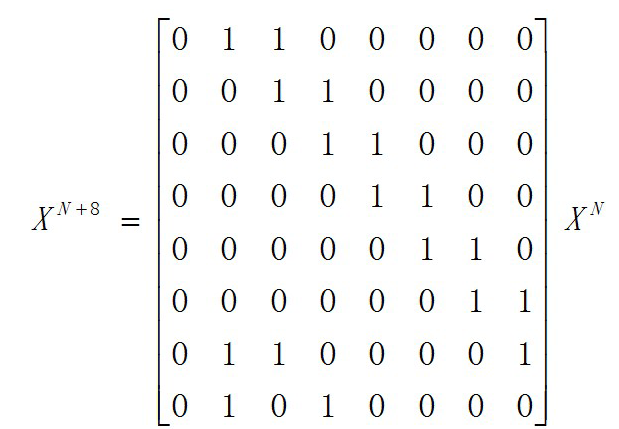
对于上述的几个关系式可以用矩阵乘法来表示。



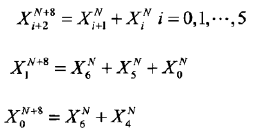
对于8位并行扰码，需要知道（N+8）时刻输出相对于N时刻的关系，在上式中令



经过计算可得：

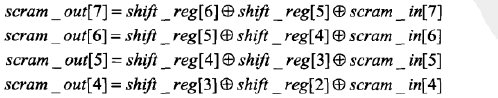


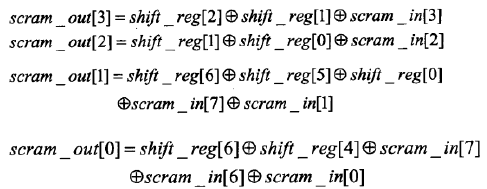
即



上述式子就是并行扰码的理论基础。

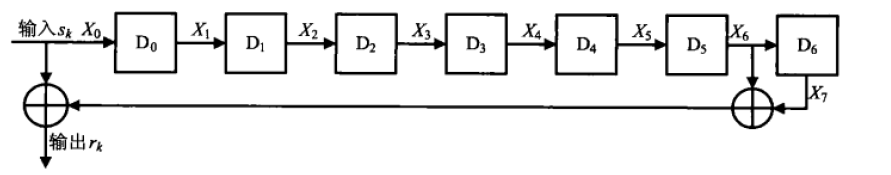
根据上述式子，假设并行8位输入数据为scram\_in[7:0]，扰码器的并行8位输出数据为scram\_out[7:0]，shift\_reg[6:0]，是加扰器的移位寄存器的初始状态加扰器复位后初始状态一般为shift\_reg[6:0]=7b’111\_1111，则可以得并行8位数据输入后加扰器输出数据和加扰器输入数据以及移位寄存器初始状态的关系式：



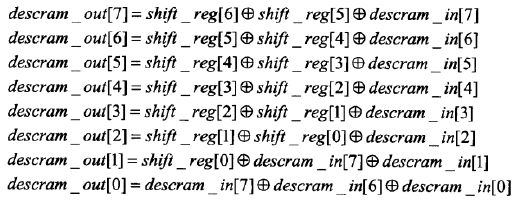


## **1.2.2、并行解扰器设计**

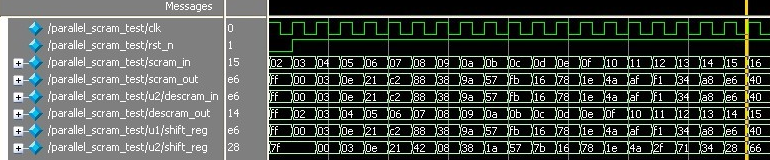
并行加扰器电路设计原理和并行加电路设计原理一样，如图所示为解扰码器的电路图：

 解扰器电路结构

假设解扰器电路输入时descram\_in[7:0]，解码电路输出是descram\_out[7:0]，移位寄存器是shift\_ren[7:0]，可以根据并行扰码的推导方法导出解扰器输出和解扰器输入以及移位寄存器初始状态的逻辑关系式：



# **1.3、仿真输出波形**



如图所示是并行加扰/解扰器的仿真输出波形，从图中可以看出测试文件输入初始为0000 0001的8位数据，每来一个时钟该数据加1，该数据经过加扰器并行扰码后将数据送给并行解码器解扰，加扰器的输出作为解扰器的输入，由图中可以看出，连0和连1的个数最多8个，同时加扰器输入数据与解扰器输出数据相同，只差一个时钟周期的运算延时。

# **1.4、并行扰码/解扰码器的Verilog HDL程序**

## **1.4.1、扰码器的程序**

module parallel\_scrambler(clk,

rst\_n,

scram\_in,

scram\_out);

input clk,rst\_n; //rst\_n为复位信号，低电平有效

input [7:0] scram\_in; //扰码数据输入

output [7:0] scram\_out; //扰码数据输出

reg [7:0] shift\_reg; //反馈移位寄存器

//输出的反馈异或关系

assign scram\_out[7:0] = shift\_reg [7:0];

always @ (posedge clk or negedge rst\_n) begin

if(!rst\_n) //已将rst改为!rst（考虑到可综合性问题）

shift\_reg <= 8'b1111\_1111;

else begin

//寄存器反馈异或关系

shift\_reg[7:2] <= shift\_reg[6:1] ^ shift\_reg[5:0] ^ scram\_in[7:2];

shift\_reg[1] <= shift\_reg[0] ^ shift\_reg[6] ^ shift\_reg[5] ^ scram\_in[7] ^ scram\_in[1];

shift\_reg[0] <= shift\_reg[6] ^ shift\_reg[4] ^ scram\_in[7] ^ scram\_in[6] ^ scram\_in[0];

end

end

Endmodule

**1.4.2**、解扰码器的程序

module parallel\_decrambler(clk,rst\_n,descram\_in,descram\_out);

input clk,rst\_n; //rst\_n为复位信号，低电平有效

input [7:0] descram\_in; //扰码数据输入

output [7:0] descram\_out; //扰码数据输出

reg [6:0] shift\_reg; //反馈移位寄存器

assign descram\_out[7:2] = shift\_reg[6:1] ^ shift\_reg[5:0] ^ descram\_in[7:2];

assign descram\_out[1] = shift\_reg[0] ^ descram\_in[7] ^ descram\_in[1];

assign descram\_out[0] = descram\_in[7] ^ descram\_in[6] ^ descram\_in[0];

always@(posedge clk or negedge rst\_n)begin

if(!rst\_n) //已将rst改为!rst（考虑到可综合性问题）

shift\_reg [6:0] <= 7'b111\_1111;

else begin

shift\_reg[6:0] <= descram\_in[6:0];

end

end

Endmodule

**1.4.3、测试文件的Verilog HDL**程序

module parallel\_scram\_test;

reg clk;

reg rst\_n;

reg [7:0] scram\_in;//扰码器的输入端

wire [7:0] scram\_out; //既是扰码器的输出端，也是解扰器的输入端

wire [7:0] descram\_out; //解扰器的输出端

parameter period = 20;

initial begin

clk = 1;

rst\_n = 0; //已将rst\_n = 1改为rst\_n = 0

scram\_in[7:0] = 8'b000\_0001;

#period rst\_n = 1; //已将rst\_n = 0改为rst\_n = 1

end

//根据第8章，伪随机序列产生的原理，产生一个伪随机的序列作为加扰器的输入数据，

//该伪随机序列的生成多项式为x^8 + x^4 + x^3 + x^2 + 1

always #(period/2) clk =~clk;

always @ (posedge clk) begin

scram\_in [7:0] <= scram\_in[7:0]+1;

end

//加扰、解扰器模块调用

parallel\_scrambler u1 (.clk(clk),

.rst\_n(rst\_n),

.scram\_in(scram\_in),

.scram\_out(scram\_out));

parallel\_decrambler u2 (.clk(clk),

.rst\_n(rst\_n),

.descram\_in(scram\_out),

.descram\_out(descram\_out));

endmodule

**2、CRC\_16并行算法的Verilog HDL**程序设计

**2.1**、设计任务和要求

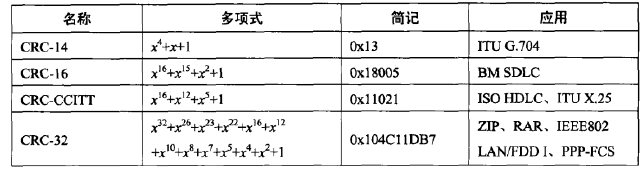
**2.1.1、CRC\_16**并行算法原理

**2.1.2、CRC16并行算法的Verilog HDL**程序设计。

**2.2、CRC\_16**并行算法原理

**2.2.1、CRC**编码原理

在数字通信中，可能会因为各种原因导致数据在传输过程中或接收时发生错误，为了保证数据传输的可靠性和数据校验的高效性，常常采用一些差错控制方法。冗余校验（CRC）就是一种被广泛采用的差错控制方法和数据编码方法。它具有编码和解码方法简单，剪错和纠错能力强等特点，能有效地对数据进行编码，并可以显著的提高系统的剪错能力，从而保证数据的可靠性和正确性，因此在大多数的以太网协议中都采用了CRC的校验来进行差错控制。CRC主要有两种，即非标准的由用户定义的CRC的生成多项式和国际标准化组织规定的标准的生成多项式。其中第二中应用广泛，国际标准化组织规定的几种主要常见的CRC如表所示。



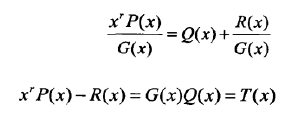
CRC编码是根据CRC检验原理得出的一种编码方法，其基本思想是：利用线性编码理论，在发送方向根据要传输的K位二进制序列，以一定的规则产生r位校验用的监督吗（CRC码），并附在信息位之后，构成一个新的二进制代码序列共n=k+r位。在接收方，则根据信息码和CRC码之间的规则进行校验，以确定传输中是否出现错误。

对一个数据进行编码，就是原始数据通过某种算法，得到一个新的数据。而这个新的数据与原始数据有着固定的内在联系。通过把原始数据和新的数据组合在一起形成新的数据，因此这个数据具有自我校验的能力。将原始数据表示为P（x），它是一个n阶多项式表示为；



式子中，为数据位；x为伪变量，用指明各位间的排列位置。

在对这个数据进行编码的时，CRC生成多项式G（x），并在带编码的二进制多项式P（x）的末尾添加r个0，这样对应的二进制多项式升幂为。再用生成多项式G（x）去除，求得余数为r-1阶的二进制多项式R（x）。R（x）即为所求的CRC校验码多项式。最后用以模2的方法减去R（x），则对应的二进制序列就是包含了CRC校验码的待发送字符串，也就是编码后的数据。

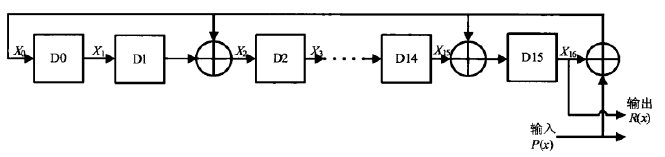


式中：Q(x）是商多项式；R(x)是余数多项式；T(x)是编码后的数据。

**2.2.2、CRC\_16**并行算法原理

在利用CRC\_16串行编码电路对输入进行编码时，它的输入数据是串行输入的，因此数据传输效率低，它的运算速度的提高完全依赖于系统时钟频率的提高，不能满足现在的低功耗下的数据快速传输，如果将数据进行并行处理，就可以在较低的时钟频率下快速的处理数据，因此，本次设计采用并行的CRC\_16。

最常见的CRC校验码电路采用的是线性反馈寄存器LFSR实现，对于CRC\_16，其生成多项式为，所以对应的系数0—16为：:1010 0000 0000 00011，由此可以得到CRC\_16编码电路结构图，如图所示。

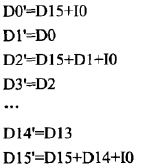


从图中可以看出，各位移位寄存器当前存储的数据为CRC余数值，16个D触发器输出从右至左依次为。输入的信息码元为I。令表示编码以为寄存器当前所处的状态，I=[I0 I1...I7]表示第1至第8个时钟的信息码元输入，向量表示编码器的下一个状态，D(8)表示第8个时钟周期后CRC编码器中移位寄存器所处的状态，则设计8bit并行的CRC逻辑编码器就是找出函数关系。

第1个时钟周期后的编码寄存器状态为：



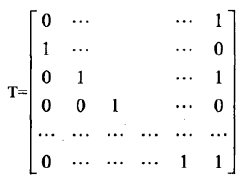




可以将以上表达式组成矩阵乘法，有：



其中，





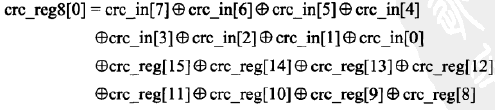
第2个时钟周期后的编码寄存器状态为：

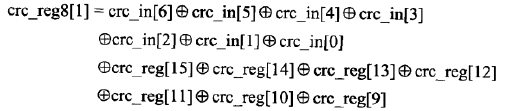


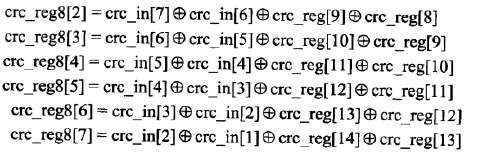
以此类推则有：



假设用crc\_reg[15:0]表示移位寄存器初始状态，crc\_in[8:0]表示并行输入的8bit数据，crc\_reg8[15:0]表示8bit并行编码后的移位寄存器状态，则

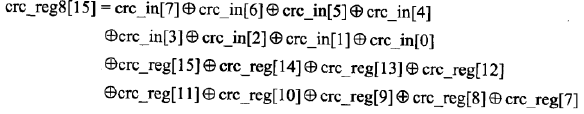






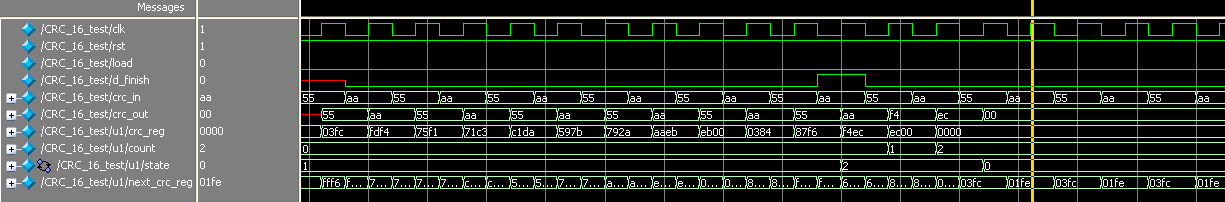






**2.3、CRC\_16并行**仿真输出波形

如图所示是CRC\_16并行编码电路测试程序的仿真结果，输入信号时crc\_in，没有完全输入编码器是，输入信号crc\_in作为crc\_out端口输出数据；当d\_finish信号为高电平指示数据输入完成时，将线性移位寄存器crc\_reg中的校验位并行8位输出到crc\_out端口。由图可以看出输入信号与输出存在一个时钟周期的延时。



2.4、CRC\_16并行算法的Verilog HDL程序

**2.4.1、CRC\_16**校验码的并行编码的程序

module CRC\_16(clk,rst,load,d\_finish,crc\_in,crc\_out);

input clk,rst;

input load; //开始编码信号

input d\_finish; //编码结束信号

input [7:0] crc\_in; //编码器并行8bit数据输入

output [7:0] crc\_out; //编码器并行8bit数据输出

reg [7:0] crc\_out; //数据输出寄存器

reg [15:0] crc\_reg; //编码器移位寄存器

reg [1:0] count;

reg [1:0] state;

wire [15:0] next\_crc\_reg; //移位寄存器输入变量

parameter idle = 2'b00;

parameter compute = 2'b01;

parameter finish = 2'b10;

//寄存器输出和寄存器初始状态以及输入码字组合逻辑关系

assign next\_crc\_reg[0] = (^crc\_in[7:0]) ^ (^crc\_reg[15:8]);

assign next\_crc\_reg[1] = (^crc\_in[6:0]) ^ (^crc\_reg[15:9]);

assign next\_crc\_reg[2] = crc\_in[7] ^ crc\_in[6] ^ crc\_reg[9] ^ crc\_reg[8];

assign next\_crc\_reg[3] = crc\_in[6] ^ crc\_in[5] ^ crc\_reg[10] ^ crc\_reg[9];

assign next\_crc\_reg[4] = crc\_in[5] ^ crc\_in[4] ^ crc\_reg[11] ^ crc\_reg[10];

assign next\_crc\_reg[5] = crc\_in[4] ^ crc\_in[3] ^ crc\_reg[12] ^ crc\_reg[11];

assign next\_crc\_reg[6] = crc\_in[3] ^ crc\_in[2] ^ crc\_reg[13] ^ crc\_reg[12];

assign next\_crc\_reg[7] = crc\_in[2] ^ crc\_in[1] ^ crc\_reg[14] ^ crc\_reg[13];

assign next\_crc\_reg[8] = crc\_in[1] ^ crc\_in[0] ^ crc\_reg[15] ^ crc\_reg[14] ^ crc\_reg[0];

assign next\_crc\_reg[9] = crc\_in[0] ^ crc\_reg[15] ^ crc\_reg[1];

assign next\_crc\_reg[14:10] = crc\_reg[6:2];

assign next\_crc\_reg[15] = (^crc\_in[7:0]) ^ (^crc\_reg[15:7]);

always @ (posedge clk or negedge rst) begin

if(!rst) begin

state <= idle;

count <= 2'b00;

end

else begin

case(state)

idle : begin

if(load)

state <= compute;

else

state <= idle;

end

compute : begin

if(d\_finish)

state <= finish;

else

state <= compute;

end

finish : begin

if(count == 2) begin

state <= idle;

end

else begin

count <= count + 1;

state <= finish;

end

end

endcase

end

end

always @ (posedge clk or negedge rst) begin

if(!rst) begin

crc\_reg[15:0] <= 16'b0000\_0000\_0000\_0000;

end

else begin

case(state)

idle : begin //寄存器装初值状态

crc\_reg[15:0] <= 16'b0000\_0000\_0000\_0000;

end

compute : begin //编码计算状态

crc\_reg[15:0] <= next\_crc\_reg[15:0];

crc\_out[7:0] <= crc\_in[7:0];

end

finish : begin //编码结束状态

crc\_reg[15:0] <= {crc\_reg[7:0],8'b0000\_0000};

crc\_out[7:0] <= crc\_reg[15:8];

end

endcase

end

end

Endmodule

**2.4.2、CRC\_16**测试文件程序

module CRC\_16\_test;

reg clk;

reg rst;

reg load;

reg d\_finish;

reg [7:0] crc\_in;

wire [7:0] crc\_out;

parameter clk\_period = 40;

initial begin

#clk\_period clk = 1;

#clk\_period rst = 0;

#clk\_period rst = 1;

#clk\_period crc\_in[7:0] = 8'b1010\_1010; //输入待编码数据

#clk\_period load = 1;

#clk\_period load = 0;

#clk\_period d\_finish = 0;

#(10\*clk\_period) d\_finish = 1;

#clk\_period d\_finish = 0;

end

always #(clk\_period/2) clk = ~clk;

always #(clk\_period) crc\_in[7:0] = ~crc\_in[7:0]; //输入待编码数据

CRC\_16 u1(.clk(clk),

.rst(rst),

.load(load),

.d\_finish(d\_finish),

.crc\_in(crc\_in),

.crc\_out(crc\_out));

Endmodule