基于FPGA的信道编译码器设计与实现

预习报告 - (@2024-03-19)

实验目的

1. 理解信道特性，掌握信道编码的方法；
2. 能够设计并实现信道编译码器。

实验原理

M 序列发生器

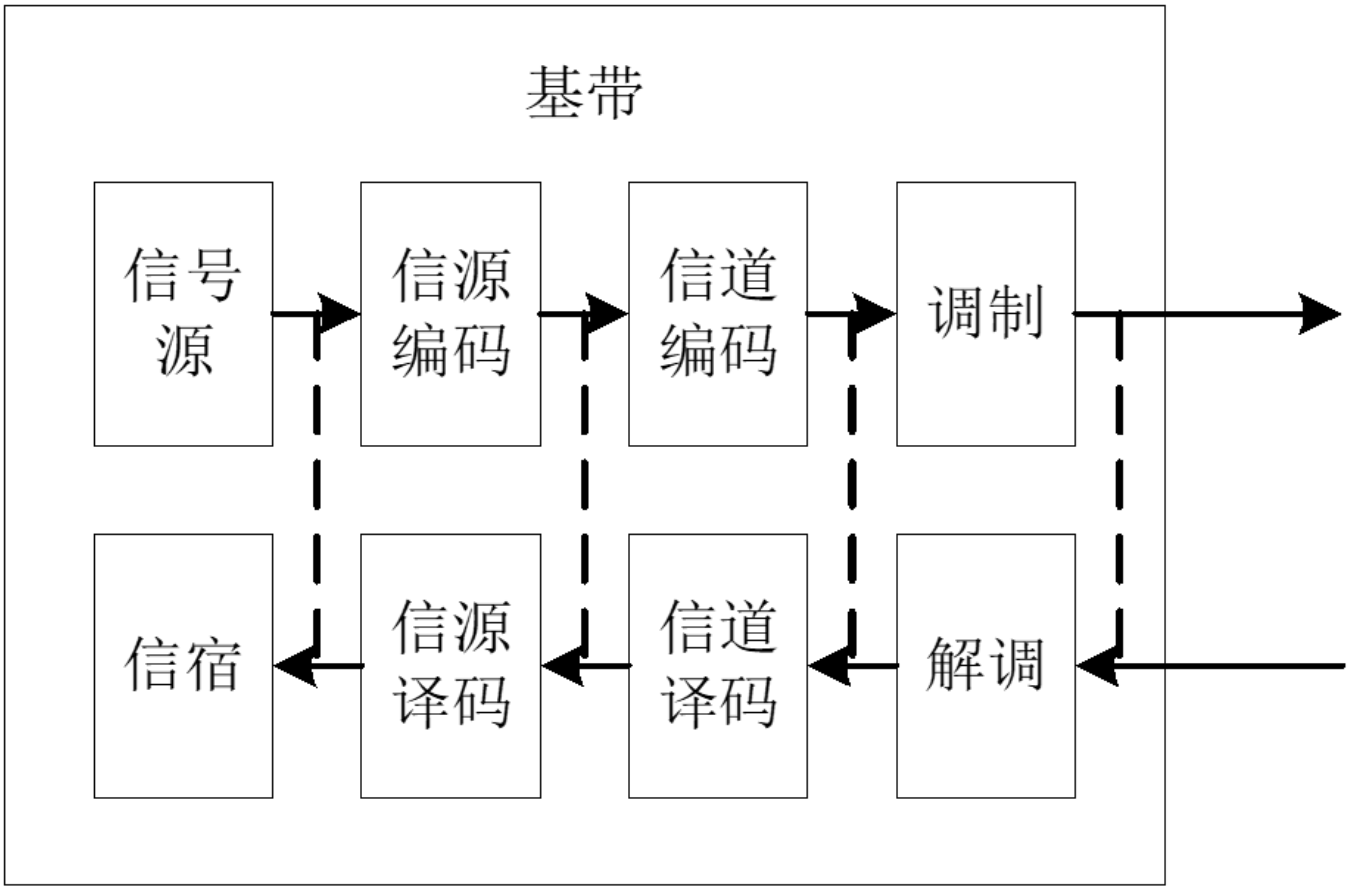
长度为 k 的 m 序列发生器会产生周期为 2^k-1 的序列串，**其中任取一个长度为 k 的序列总是不相同**，并且一个周期内长度为 k 的序列遍历除了 $(000\cdots0)_{k个0}$ 以外的全体序列。

信道编码

信道编码是一种在通信系统中常用的技术，旨在提高信息传输的可靠性，以抵御各种噪声和干扰。信道编码通过在原始数据中加入额外的冗余比特来实现这一目标，这样即使数据在传输过程中出现错误，接收端也可以通过这些冗余比特检测并纠正错误。

信道编码可以分为两大类：分组码和卷积码。

1. 线性分组码：
   - 汉明码、循环码、BCH 码、BS 码；
   - 分组码将可能的符号序列映射成长为n的码字。
2. 卷积码：
   - 卷积码则通过将数据序列与特定的码序列进行卷积编码来实现冗余性的增加。



数字基带系统框图

汉明码

汉明码是一种能够纠正单个错误的线性分组码。

它有以下特点：

（1）最小码距

$$d_{min} = 3$$

，可以纠正一位错误；
（2）码长 n 与监督元个数 r 之间满足关系式：

$$n = 2^r - 1$$

如果要产生一个系统汉明码，可以将矩阵 H 转换成典型形式的监督矩阵，进一步利用 Q = PT 的关系，得到相应的生成矩阵 G。通常二进制汉明码可以表示为：

$$(n, k) = (2^r - 1,\ 2^y - 1 - r)$$

根据上述汉明码定义可以看到，(7，4) 线性分组码实际上就是一个汉明码，它满足了汉明码的两个特点。

实验内容

实验任务

1. 将**实验2**中信源编码后的码元序列作为信道编码器的输入，在 QuartusII 中使用 verilog 或 VHDL 语言设计实现信道编码器，对输入序列进行编码和译码，利用 Modelsim 进行仿真测试；
2. 将程序加载至 FPGA（XSRP 平台）进行验证，利用示波器对编码前后、加噪后序列进行实测，对比分析仿真和实测结果。

思考与讨论

1. 阐述所设计的信道编译码器通常应用在什么样的通信环境中；
2. 讨论所设计的信道编码器的纠错能力和纠错极限。

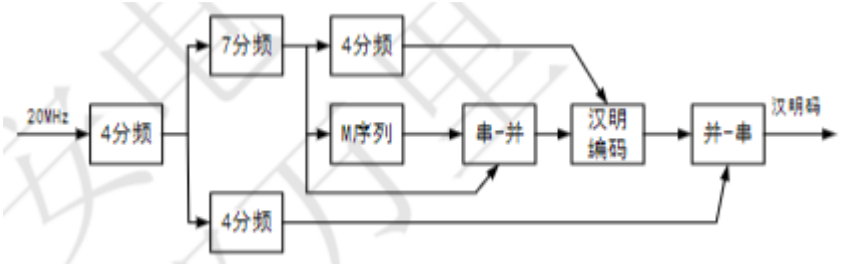
实验方案设计

汉明码

流程设计

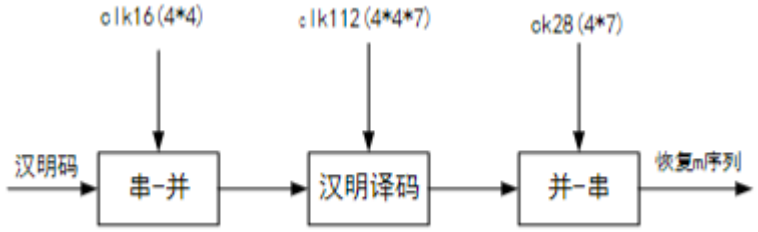
1. 汉明编码
2. 加噪
3. 汉明译码

汉明编码模块



汉明编码器原理框图

汉明译码模块



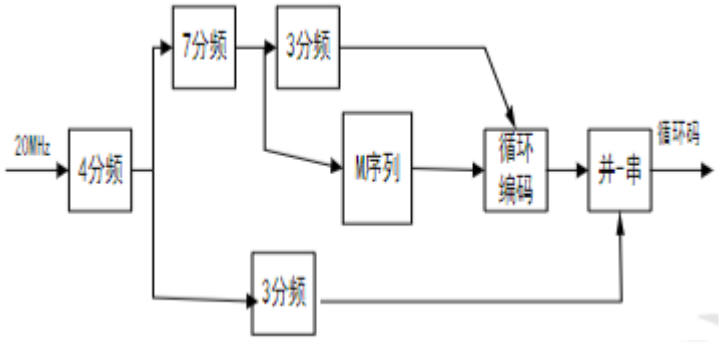
汉明译码器原理框图

(7, 3) 循环码

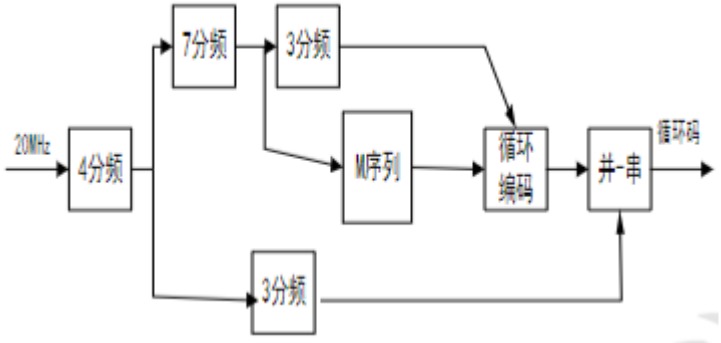
流程设计

1. 循环编码
2. 加噪
3. 循环译码

循环编码模块



循环译码模块

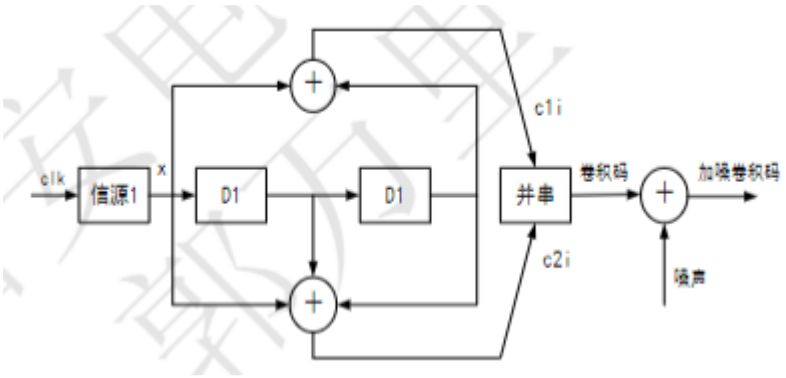


(2, 1, 2) 卷积码

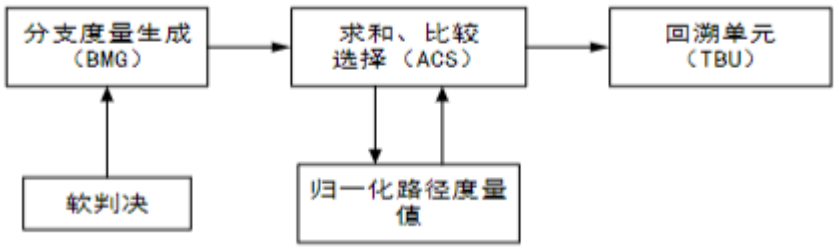
流程设计

1. 卷积编码
2. 加噪
3. 卷积译码

卷积编码模块



> 卷积编码器由 D1, D2 两个移位寄存器以及一个加法器构成
> 卷积译码模块



总结

各码的流程设计相同，分为以下三部分：

1. 编码
2. 加噪
3. 译码

对于各编译码模块而言，除各异的编译码器外，有以下通用模块：

1. M 序列发生器
2. 奇偶分频器
3. 串并转换器
4. 并串转换器
   上述模块应设计为可复用的通用模块。

因此对于各码而言，仅编译码器需单独设计。

通用模块

M 序列发生器

> 2 ~ 16 位 M 序列发生器

```
module M_series#(
        parameter       len = 4  // parameter range from 2 to 16
    )
    (
        input wire      clk,
        input wire      rst_n,
        output wire     Q
    );

    reg[(len-1):0] Q_r;

    assign Q = Q_r[(len-1)];

    always @(posedge clk or negedge rst_n) begin
```

```
        if (rst_n == 0) begin  // negedge reset
            Q_r <= ~(0);
        end

        else begin
            Q_r <= Q_r<<1;  // shift reg
            case(len)
                2 : Q_r[0] <= Q_r[1]^Q_r[0];
                3 : Q_r[0] <= Q_r[2]^Q_r[1];
                4 : Q_r[0] <= Q_r[3]^Q_r[2];
                5 : Q_r[0] <= Q_r[4]^Q_r[2];
                6 : Q_r[0] <= Q_r[5]^Q_r[4];
                7 : Q_r[0] <= Q_r[6]^Q_r[3];
                8 : Q_r[0] <= Q_r[7]^Q_r[5]^Q_r[4]^Q_r[3];
                9 : Q_r[0] <= Q_r[8]^Q_r[4];
                10: Q_r[0] <= Q_r[9]^Q_r[6];
                11: Q_r[0] <= Q_r[10]^Q_r[8];
                12: Q_r[0] <= Q_r[11]^Q_r[10]^Q_r[7]^Q_r[5];
                13: Q_r[0] <= Q_r[12]^Q_r[11]^Q_r[9]^Q_r[8];
                14: Q_r[0] <= Q_r[13]^Q_r[12]^Q_r[7]^Q_r[3];
                15: Q_r[0] <= Q_r[14]^Q_r[13];
                16: Q_r[0] <= Q_r[15]^Q_r[14]^Q_r[12]^Q_r[3];
                default: Q_r[0] <= Q_r[0];
            endcase
        end

    end

endmodule
```

奇偶分频器

> 可指定分频数与占空比

```
//rtl
module div(
    clk,
    rst_n,
    clk_div,
);
    input clk,rst_n;
    output clk_div;
    reg clk_div;
    reg [15:0] counter;

always @(posedge clk)
    if(!rst_n)
        counter <= 0;
    else if(counter==4)  // 4 分频
        counter <= 0;
    else counter <= counter+1;
```

```
always @(posedge clk)
  if(!rst_n)
    clk_div <= 0;
  else if(counter<2)  // 4 分频 * 50% 占空比
    clk_div <= 1;
  else
    clk_div <= 0;
 endmodule
```

串并转换器

> 分别实现指定位数的高位优先与低位优先串并转换器，以 8 位为例。

1. 低位优先

```
module Deserialize (
    input            clk,
    input            rst_n,
    input            data_i,
    output reg [7:0] data_o
);
  //lsb first
  always @(posedge clk or negedge rst_n) begin
    if (rst_n == 1'b0) begin
      data_o <= 8'b0;
    end else begin
      data_o <= {data_o[6:0], data_i};  //去掉了高位，让data_i来代替高位，相当于左移一位
    end
  end
endmodule
```

2. 高位优先

```
module Deserialize (
    input            clk,
    input            rst_n,
    input            data_i,
    output reg [7:0] data_o
);
  reg [2:0] cnt;

  always @(posedge clk or negedge rst_n) begin
    if (rst_n == 1'b0) begin
      data_o <= 8'b0;
      cnt    <= 3'd0;  // 三位十进制
    end else if (cnt == 3'b111) cnt <= 3'd0;
    else begin
```

```
      data_o[7-cnt] <= data_i;  // 用计数来进行7-0的移位操作
      cnt           <= cnt + 1'b1;
    end
  end
endmodule
```

并串转换器

> 实现可指定位数的并串转换器，以 8 位为例。

```
module right_shifter_reg
(
input               clk         ,
input               rst_n       ,
input       [7:0]   din         ,
input               din_vld     ,
output              dout
);

reg     [7:0]       shift       ;

always @(posedge clk or negedge rst_n) begin
    if(!rst_n)
        shift <= 8'b0;
    else if(din_vld) // 寄存
        shift <= din;
    else
        shift <= {shift[0], shift[7:1]}; // 右移
end

assign dout = shift[0];// 输出

endmodule
```