FSK 调制与解调

通信原理实验

[无 52] **王冠儒** & [无 52] **万子牛**

https://github.com/WisdomFusion

1 实验目的

1. 了解 FSK 调制与解调原理;
2. 熟悉 FPGA 实验平台;
3. 熟悉编码与解码;
4. 利用 Quartus II 软件、Altera 实验平台实现 FSK 调制与解调的程序设计。

2 实验原理

本次实验中，我们利用实验板实现了一个基于 FSK 方式的通信系统。具体操作步骤如下：

1. **编码**。用户从实验板上的按键输入一组 4 位原码，这 4 位原码经过 FPGA 编码后，形成 10 位码元，即一帧编码结果; 在这 10 位中，前 3 位固定为 110，在解码时用来识别一帧的开头; 中间的 4 位是原码，最后 3 位是汉明码。
2. **调制**。这 10 位在编码后，串行输入到调制模块。调制模块的调制方式是 FSK 调制。
3. **解调**。调制完成后，输出到数模转换模块; 数模转换与模数转换模块相连，然后输出到解调模块。
4. **解码**。经过调制的编码结果经过解调后，输出到解码部分。被解码后，输出结果通过 4 个指示灯来显示，灯暗代表 0 ，灯亮代表 1 。同时输出到指示灯的还有一位示错信号，用来指示是否接收到的信号是否有错。

3 实验内容与仿真

3.1 调制模块（王冠儒实现）

调制模块本质上是一个多路选择器。它首先通过原始时钟信号分频生成两个时钟信号，分别是 1 分频和 2 分频。而多路选择器的控制信号则是编码结果信号。当编码信号为 0 时，输出 1 分

频时钟；当编码信号为 1 时，输出 2 分频时钟。其系统原理框图如下：

该部分代码如下：

</> CODE 1: FSK.v

```
module FSK (FSK_in, FSK_out, clk, rst_n);
  input FSK_in;
  input clk;
  input rst_n;
  output FSK_out;

  reg [1:0] cnt;
  reg clk_0, clk_1;

  always @(posedge clk or negedge rst_n)
  begin
    if(~rst_n)
      cnt <= 2'd0;
    else
      cnt <= (cnt < 2'd3)? (cnt + 2'd1): 2'd0;
  end

  always @(cnt)
  begin
    case(cnt)
      2'd0: begin clk_0 = 1; clk_1 = 1; end
      2'd1: begin clk_0 = 0; clk_1 = 1; end
      2'd2: begin clk_0 = 1; clk_1 = 0; end
      2'd3: begin clk_0 = 0; clk_1 = 0; end
    endcase
  end

  assign FSK_out = (~FSK_in)? clk_0: clk_1;


endmodule
```

3.2 解调模块（万子牛实现）

解调模块本质上是一个计数器。由于原始时钟的一分频代表信号 0，二分频代表信号 1；而编码信号则是由原始时钟的十六分频，即一个码字的持续时长为原始信号的 16 个周期。因此，只需在原始信号的 16 个周期内进行计数，每 16 个周期结束后对计数器进行检查：

1. 若计数器结果为 **16**，则代表该码字为 0；
2. 若计数器结果为 **8**，则代表该码字为 1；

为了使模块对噪声具有更强的鲁棒性，对以上策略进行修改：

1. 若计数器结果大于 **12**，则代表该码字为 0；
2. 若计数器结果小于 **12**，则代表该码字为 1；

其系统原理框图如下：

该部分代码如下：

</> CODE 2: deFSK.v

```
module deFSK (deFSK_in, deFSK_out, clk, rst_n);
  input deFSK_in;
  input clk;
  input rst_n;
  output deFSK_out;

  reg [4:0] cnt_period;
  reg [4:0] c_cnt_in, n_cnt_in;
  reg c_data, p_data;
  reg c_out, n_out;

  always @(posedge clk or negedge rst_n)
  begin
    if(~rst_n)
    begin
      cnt_period <= 5'd0;
      c_cnt_in <= 5'd0;
      c_data <= 1'd0;
      p_data <= 1'd0;
      c_out <= 1'd0;
    end
    else
    begin
      cnt_period <= (cnt_period < 5'b11111)? (cnt_period + 5'd1): 5'
   d0;
      c_cnt_in <= n_cnt_in;
      c_data <= deFSK_in;
      p_data <= c_data;
      c_out <= n_out;
    end
  end

  always @(c_cnt_in, c_data, p_data)
  begin
    if(p_data == 1'b0 && c_data == 1'b1)
      n_cnt_in = (cnt_period == 5'd0)? 5'd0: (c_cnt_in + 5'd1);
    else
```

```
      n_cnt_in = (cnt_period == 5'd0)? 5'd0: c_cnt_in;
  end

  always @(c_cnt_in, cnt_period, c_out)
  begin
    if(cnt_period == 5'd0)
      n_out = (c_cnt_in <= 5'd12)? 1'b1: 1'b0;
    else
      n_out = c_out;
  end

  assign deFSK_out = c_out;

endmodule
```

3.3 顶层封装（万子牛实现）

系统框图如下：

实现代码如下：

</> CODE 3: FSKSystem.v

```
module FSKSystem(inputData, outputData, clk, wr, rst_n, encode_en);

    input clk, rst_n, encode_en;
    input [3:0] inputData;
    output wr;
    output [3:0] outputData;

    wire clkCode;
    wire dataToFSK;
    wire dataToDeFSK;
    wire dataToDecoder;

    clk_code codeClk(clk, clkCode, rst_n);
    encode encoder(inputData, dataToFSK, encode_en, clkCode, rst_n);
    FSK FSKer(dataToFSK, dataToDeFSK, clk, rst_n);
    deFSK deFSKer(dataToDeFSK, dataToDecoder, clk, rst_n);
    decode decoder(dataToDecoder, outputData, wr, clkCode, rst_n);

endmodule
```

3.4 Testbench 仿真（万子牛实现）

3.4.1 编码器模块

</> CODE 4: encode_tb.v

```
module encode_tb;

  reg clk;
  reg[3:0] inputData;
  reg encode_en;
  reg rst_n;
  wire outputData;

  encode encoder(inputData, outputData, encode_en, clk, rst_n);

  always
    begin
      #5 clk <= ~clk;
    end

  initial
    begin
      clk <= 1'b0;
      inputData <= 4'b0101;
      encode_en <= 1'b0;
      rst_n <= 1'b0;
      #100
      encode_en <= 1'b1;
      rst_n <= 1'b1;
    end

endmodule
```

3.4.2 分频器模块

</> CODE 5: clk_tb.v

```
module clk_tb;

    reg clk, rst_n;
    wire clk_output;

    clk_code codeClk(clk, clk_output, rst_n);

    always
```

```
        begin
            #5 clk <= ~clk;
        end

    initial
    begin
      clk <= 1'b0;
      rst_n <= 1'b0;

      #40 rst_n <= 1'b1;

    end

endmodule
```

3.4.3 调制模块

</> CODE 6: FSK_tb.v

```
module FSK_tb;

  reg inputData, clk, rst_n;
  wire outputData;

  FSK FSKer(inputData, outputData, clk, rst_n);

  always
    begin
      #5 clk <= ~clk;
    end

  initial
    begin
      clk <= 1'b0;
      rst_n <= 1'b0;
      inputData <= 1'b0;

      #320 inputData <= 1'b1;
      #320 inputData <= 1'b0;
      #320 inputData <= 1'b1;
      #320 inputData <= 1'b1;
      #320 inputData <= 1'b0;
      #320 inputData <= 1'b0;
      #320 inputData <= 1'b0;
      #320 inputData <= 1'b1;
      #320 inputData <= 1'b1;
```

```
        #320 inputData <= 1'b1;

        #320 rst_n <= 1'b1;

        #320 inputData <= 1'b1;
        #320 inputData <= 1'b0;
        #320 inputData <= 1'b1;
        #320 inputData <= 1'b1;
        #320 inputData <= 1'b0;
        #320 inputData <= 1'b0;
        #320 inputData <= 1'b0;
        #320 inputData <= 1'b1;
        #320 inputData <= 1'b1;
        #320 inputData <= 1'b1;
    end
endmodule
```

3.4.4 解调模块

</> CODE 7: deFSK_tb.v

```
module deFSK_tb;

    reg inputSignal, clk, rst_n;
    wire outputData;

    deFSK deFSKer(inputSignal, outputData, clk, rst_n);

    always
        begin
            #5 clk <= ~clk;
        end

    initial
        begin
            clk <= 1'b0;
            inputSignal <= 1'b0;
            rst_n <= 1'b0;

            //rst_n test
            #20 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;
            #20 inputSignal <= 1'b1;

            rst_n <= 1'b1;
            //code = 1
```

```
#20 inputSignal <= 1'b1;
#20 inputSignal <= 1'b0;
#20 inputSignal <= 1'b1;
#20 inputSignal <= 1'b0;
#20 inputSignal <= 1'b1;
#20 inputSignal <= 1'b0;
#20 inputSignal <= 1'b1;
#20 inputSignal <= 1'b0;
#20 inputSignal <= 1'b1;
#20 inputSignal <= 1'b0;
#20 inputSignal <= 1'b1;
#20 inputSignal <= 1'b0;
#20 inputSignal <= 1'b1;
#20 inputSignal <= 1'b0;
#20 inputSignal <= 1'b1;
#20 inputSignal <= 1'b0;

//code = 0
#10 inputSignal <= 1'b1;
#10 inputSignal <= 1'b0;
#10 inputSignal <= 1'b1;
#10 inputSignal <= 1'b0;
#10 inputSignal <= 1'b1;
#10 inputSignal <= 1'b0;
#10 inputSignal <= 1'b1;
#10 inputSignal <= 1'b0;
#10 inputSignal <= 1'b1;
#10 inputSignal <= 1'b0;
#10 inputSignal <= 1'b1;
#10 inputSignal <= 1'b0;
#10 inputSignal <= 1'b1;
#10 inputSignal <= 1'b0;
#10 inputSignal <= 1'b1;
#10 inputSignal <= 1'b0;
#10 inputSignal <= 1'b1;
#10 inputSignal <= 1'b0;
#10 inputSignal <= 1'b1;
#10 inputSignal <= 1'b0;
#10 inputSignal <= 1'b1;
#10 inputSignal <= 1'b0;
#10 inputSignal <= 1'b1;
#10 inputSignal <= 1'b0;
#10 inputSignal <= 1'b1;
#10 inputSignal <= 1'b0;
#10 inputSignal <= 1'b1;
#10 inputSignal <= 1'b0;
```

```
            #10 inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;
            #10 inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;

            //code = 1
            #20 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;
            #20 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;
            #20 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;
            #20 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;
            #20 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;
            #20 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;
            #20 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;
            #20 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;


            //code = 0 (add noise)
            #10 inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;
            #10 inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;
            #10 inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;
            #3  inputSignal <= 1'b1;
            #4  inputSignal <= 1'b0;
            #3  inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;
            #10 inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;
            #10 inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;
            #10 inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;
            #10 inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;
            #10 inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;
            #10 inputSignal <= 1'b1;
```

```
            #10 inputSignal <= 1'b0;
            #10 inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;
            #3  inputSignal <= 1'b1;
            #4  inputSignal <= 1'b0;
            #3  inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;
            #10 inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;
            #10 inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;
            #10 inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;
            #10 inputSignal <= 1'b1;
            #10 inputSignal <= 1'b0;

            //code = 1 (add noise)
            #20 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;
            #20 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;
            #5  inputSignal <= 1'b1;
            #3  inputSignal <= 1'b0;
            #12 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;
            #20 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;
            #20 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;
            #20 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;
            #5  inputSignal <= 1'b1;
            #3  inputSignal <= 1'b0;
            #12 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;
            #20 inputSignal <= 1'b1;
            #20 inputSignal <= 1'b0;


        end
endmodule
```

3.4.5 解码器模块

</> CODE 8: decode_tb.v

```
module decode_tb;

  reg inputData, clk;
  reg rst_n;
  wire outputData, wrg_show;

  decode decoder(inputData, outputData, wrg_show, clk, rst_n);

  always
    begin
      #20 clk <= ~clk;
    end

  initial
    begin
      clk <= 1'b0;
      inputData <= 1'b0;
      rst_n <= 1'b0;

      #40 inputData <= 1'b1;
      #40 inputData <= 1'b1;
      #40 inputData <= 1'b0;
      #40 inputData <= 1'b1;
      #40 inputData <= 1'b0;
      #40 inputData <= 1'b0;
      #40 inputData <= 1'b0;
      #40 inputData <= 1'b1;
      #40 inputData <= 1'b1;
      #40 inputData <= 1'b1;

      //enable
      #40 rst_n <= 1'b1;
      #40 inputData <= 1'b0;

      //code = 1000|111 (Correct)
      #40 inputData <= 1'b1;
      #40 inputData <= 1'b1;
      #40 inputData <= 1'b0;
      #40 inputData <= 1'b1;
      #40 inputData <= 1'b0;
      #40 inputData <= 1'b0;
      #40 inputData <= 1'b0;
      #40 inputData <= 1'b1;
      #40 inputData <= 1'b1;
      #40 inputData <= 1'b1;
```

```
        #40 inputData <= 1'b0;

        //code = 1001|111 (Can be corrected)
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;

        #40 inputData <= 1'b0;

        //code = 1010|111 (Can be corrected)
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;

        #40 inputData <= 1'b0;

        //code = 0010|111 (Wrong)
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;

        #40 inputData <= 1'b0;

        //code = 1100|111 (Can be corrected)
```

```
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;

        #40 inputData <= 1'b0;

        //code = 1011|111 (Wrong, Would be corrected to 1111)
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;

        #40 inputData <= 1'b0;

        //code = 0000|111 (Can be corrected)
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b0;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
        #40 inputData <= 1'b1;
    end
endmodule
```

3.4.6 顶层封装

</> CODE 9: FSKSystem_tb.v

```
module FSKSystem_tb;
```

```
    reg clk, rst_n, encoder_en;
    reg [3:0] inputData;
    wire wr;
    wire [3:0] outputData;

    FSKSystem FSKSystemTest(inputData, outputData, clk, wr, rst_n,
   encoder_en);

    always
        begin
          #5 clk <= ~clk;
        end

    initial begin
        clk <= 1'b0;
        inputData <= 4'b1001;
        rst_n <= 1'b0;
        encoder_en <= 1'b0;

        #20 rst_n <= 1'b1;
        #20 encoder_en <= 1'b1;
    end

endmodule
```

可以看到，仿真结果与预期符合，设计的模块没有问题。

4 FSK 系统加噪测试

由于 verilog 无法实现产生随机信号，因此无法产生随机噪声。故我们加噪声的方式是将编码结果的某一位固定取反，然后观察解码信号是否正确。代码只需对 **encode.v** 略作修改即可：

</> CODE 10: encode.v

```
module encode (data_in, data_out, encode_en, clk, rst_n);
  input [3:0] data_in;
  input clk;
  input rst_n;
  input encode_en;
  output data_out;

  reg [3:0] c_state, n_state;
  reg [3:0] c_data, n_data;
  reg data_out;
```

```
always @(posedge clk or negedge rst_n)
begin
  if(~rst_n)
  begin
    c_state <= 4'd0;
    c_data <= 4'd0;
  end
  else
  begin
    c_state <= n_state;
    c_data <= n_data;
  end
end

always @(c_state, encode_en, c_data)
begin
  if(c_state == 4'd0)
  begin
    n_state = (encode_en)? 4'd1:4'd0;
    n_data = data_in;
  end
  else if(c_state < 4'd10)
  begin
    n_state = c_state + 4'd1;
    n_data = c_data;
  end
  else
  begin
    n_state = 4'd0;
    n_data = 4'd0;
  end
end

always @(c_state, c_data)
begin
  case(c_state)
    4'd0: data_out = 1'b0;
    4'd1: data_out = 1'b1;
    4'd2: data_out = 1'b1;
    4'd3: data_out = 1'b0;
    4'd4: data_out = c_data[3];
    4'd5: data_out = c_data[2];
    4'd6: data_out = ~c_data[1];
    4'd7: data_out = c_data[0];
    4'd8: data_out = c_data[3] ^ c_data[2] ^ c_data[1];
    4'd9: data_out = c_data[3] ^ c_data[2] ^ c_data[0];
```

```
        4'd10: data_out = c_data[3] ^ c_data[1] ^ c_data[0];
        default: data_out = 1'b0;
      endcase
    end

endmodule
```

测试结果如下图：

可以看到，虽然编码结果的一位被固定取反，解码信号仍然得到了正确的编码结果。

5 实验总结与分工

本次实验我们两人共同完成了这个 FSK 调制与解调的编码系统，虽然组里的人数比一般其他小组要少一人，但是凭借着成员扎实的 verilog 基础，我们仍然是完成了这次系统的设计。同时我们两人对汉明码的理解更深了一步。

在这次实验中，我们两人都积极对项目作出了贡献。在前期讨论的过程中，为了给自己更大的挑战，我们选择了难度比较大的汉明码编码，我们因此特别回顾了在“通信与网络”这门电子系必修课的内容，对汉明码有了一个更加全面的复习和认识，这对我们后来的设计工作带来了许多便利。

王冠儒同学主要负责编解码、调制模块的编写。他的 verilog 基本功非常扎实，代码风格也非常好，基本没有造成任何 BUG。

万子牛同学主要负责解调模块编写和 testbench 的设计。

最后，感谢通信原理实验的老师和助教老师的付出，我们也希望我们能在今后的学习过程中，秉承着相同的严谨实验的精神，实现自身更多的价值。

6 文件清单

表 1: 文件清单

文件	描述
encoder.v	编码器模块。
clk_code.v	原时钟信号的 16 分频时钟，用于生成编码模块的时钟输入。
FSK.v	FSK 调制模块。
deFSK.v	解调模块。
decode.v	解调过后的解码模块。
FSKSystem.v	顶层封装。
*_tb.v	对应模块的testbench。
encode_noise.v	对 encode 进行了加噪处理。