作业4

- 软件环境：Quartus Prime 18.0
- Device family：Cyclone IV E

作业内容

1. 设计 3-8 译码器
2. 8-3 优先编码器
3. 参数化的译码器
4. 参数化的编码器
5. 4 位格雷码计数器

设计译码器

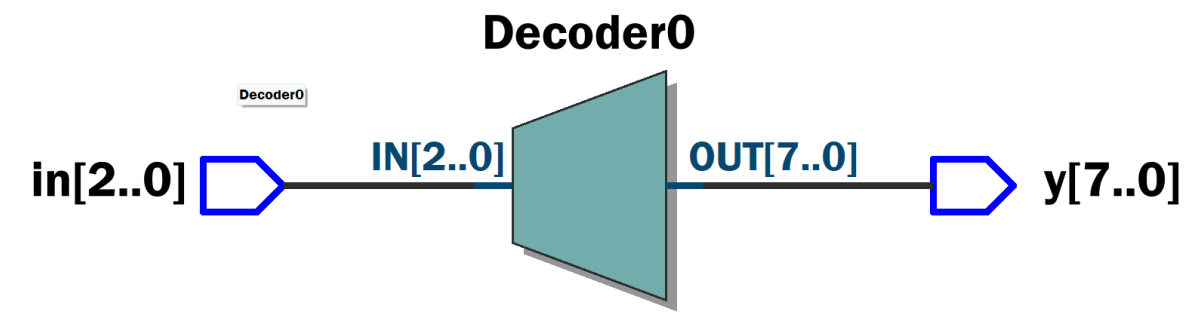
3-8 译码器

代码实现：

```
`timescale 1 ns/ 1 ps
module decoder_38(
    input wire[2:0]in,
    output reg[7:0]y
    );
    always@(*)
        case(in)
            3'b000: y = 8'b0000_0001;
            3'b001: y = 8'b0000_0010;
            3'b010: y = 8'b0000_0100;
            3'b011: y = 8'b0000_1000;
            3'b100: y = 8'b0001_0000;
            3'b101: y = 8'b0010_0000;
            3'b110: y = 8'b0100_0000;
            3'b111: y = 8'b1000_0000;
        endcase
endmodule
```

使用 case 实现输入输出一一对应的方式。

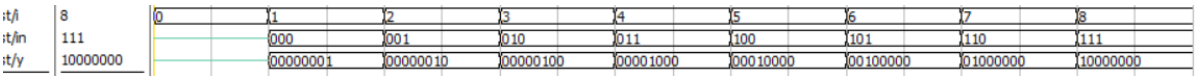
RTL viewer



> 使用的 testbench 与下一项相同，不重复列出

测试结果

波形图：



输出信息：

```
VSIM(paused)> run -all
# in = xxx ---> y = xxxxxxxx
# in = 000 ---> y = 00000001
# in = 001 ---> y = 00000010
# in = 010 ---> y = 00000100
# in = 011 ---> y = 00001000
# in = 100 ---> y = 00010000
# in = 101 ---> y = 00100000
# in = 110 ---> y = 01000000
# in = 111 ---> y = 10000000
```

> 可知结果正确

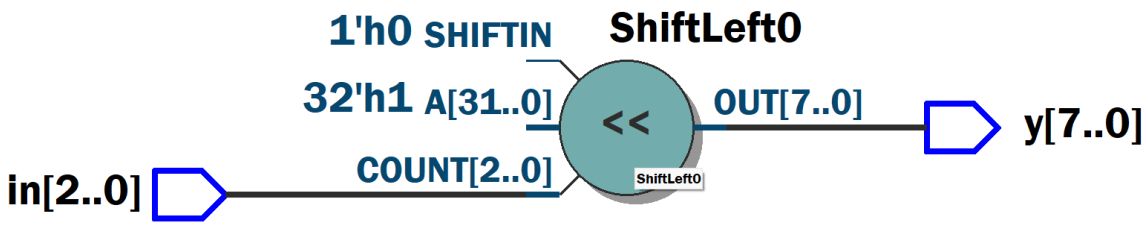
参数化译码器

代码实现

```
`timescale 1 ns/ 1 ps
module decoder #(
    parameter n=3,
              m=1<<n
    )(
    input wire[n-1:0]in,
    output reg[m-1:0]y
    );
    //one-hot
    always@(*)y=1<<in;  //1向左移in位
endmodule
```

其中，y 是独热码，即有多少个状态就有多少比特，而且只有一个比特为 1，其他全为 0 的一种码制。

RTL VIEWER：



testbench：

```
`timescale 1 ns/ 1 ps
module decoder_vlg_tst();
parameter N = 2;    //改变N的值可以测试不同的译码器
parameter M = 1<<N;
reg [N-1:0] in;
wire [M-1:0]  y;

decoder_coder #(.n(N), .m(M)) //更改参数
    i1(
    .in(in),
    .y(y)
```

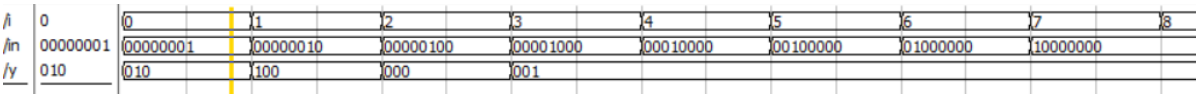
```
);

integer i;
initial
begin
    for(i=0; i<M; i=i+1) begin
        in = i;  //测试0---M-1的输入
        #10;
    end
    #10 $stop;  //使仿真暂停在这里，不会结束
end

initial
    $monitor("in = %b ---> y = %b ", in, y);

endmodule
```

通过改变 N 的值可以实现 N - 2^N 译码器，使用 for 循环测试所有可能的输入，即 0---->(2^N-1)，使用 monitor 查看输出的正确性。

测试结果

测试 N = 3：

波形图：

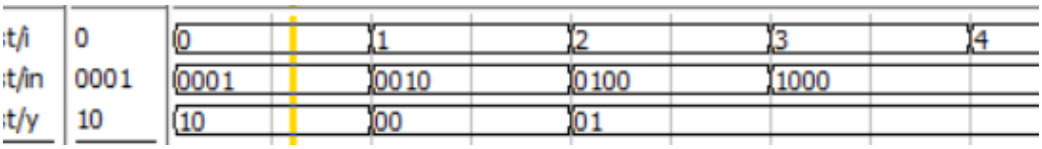


输出信息：

```
VSIM(paused)> run -all
# in = 00000001 ---> y = 010
# in = 00000010 ---> y = 100
# in = 00000100 ---> y = 000
# in = 00001000 ---> y = 001
# in = 00010000 ---> y = 001
# in = 00100000 ---> y = 001
# in = 01000000 ---> y = 001
# in = 10000000 ---> y = 001
```

测试 N = 2：

波形图：



输出信息：

```
VSIM(paused)> run -all
# in = 0001 ---> y = 10
# in = 0010 ---> y = 00
# in = 0100 ---> y = 01
# in = 1000 ---> y = 01
```

> 可知结果正确

设计编码器

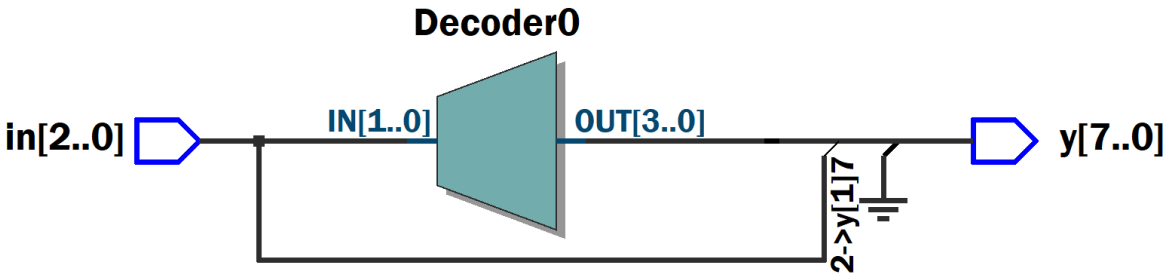
8-3 优先编码器

代码实现

```
`timescale 1 ns/ 1 ps
```

```
module decoder_coder(
    input wire[m-1:0]in,
    output reg[n-1:0]y
    );
    //one-hot
    always@(*)
        casez(in)
            8'b1???_????    : y = 3'b111;
            8'b01??_????    : y = 3'b110;
            8'b001?_????    : y = 3'b101;
            8'b0001_????    : y = 3'b100;
            8'b0000_1???    : y = 3'b011;
            8'b0000_01??    : y = 3'b010;
            8'b0000_001?    : y = 3'b001;
            8'b0000_0001    : y = 3'b000;
            default         : y = 3'b000;
        endcase
endmodule
```

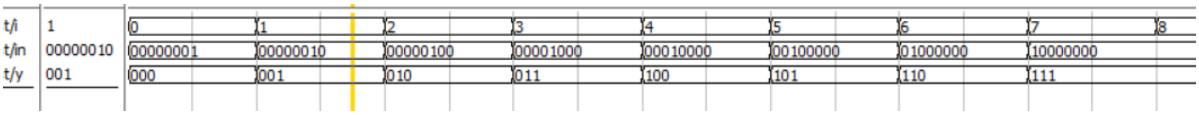
RTL VIEWER



> 使用的 testbench 与下一项相同，不重复列出

测试结果

波形图：



输出信息：

```
VSIM(paused)> run -all
# in = 00000001 ---> y = 000
# in = 00000010 ---> y = 001
# in = 00000100 ---> y = 010
# in = 00001000 ---> y = 011
# in = 00010000 ---> y = 100
# in = 00100000 ---> y = 101
# in = 01000000 ---> y = 110
# in = 10000000 ---> y = 111
```

> 可知结果正确

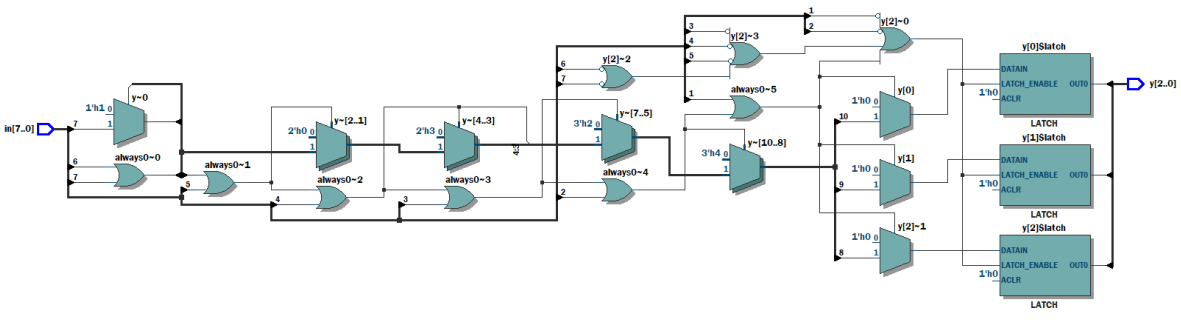
参数化编码器

代码实现

```
module decoder_coder #(
    parameter n=3,
              m=1<<n
)(
    input wire[m-1:0]in,
```

```
    output reg[n-1:0]y
);
    integer i;
    always@(*)
    begin:loop
        for (i=m-1;i>0;i=i-1)
            if(in[i]==1)
            begin
                y = i;
                disable loop;  //jump out of loop
            end
            else y = 0;
    end

endmodule
```

RTL VIEWER:



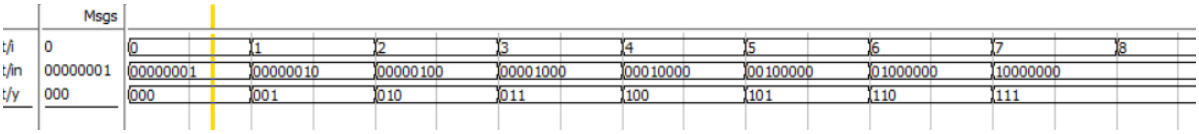
testbench:

```
`timescale 1 ns/ 1 ps
module decoder_coder_vlg_tst();
parameter N = 3;
parameter M = 1<<N;
reg [M-1:0] in;
wire [N-1:0]  y;

decoder_coder #(.n(N), .m(M))
    i1 (
    .in(in),
    .y(y)
);

integer i;
initial
begin
    for(i=0; i<M; i=i+1) begin
        in = 1<<i;    //这个前面不能加延时，关键是循环的第一个不能有延时，不然会导致波形不正确
        #10;
    end

    #10 $stop;
end

initial
    $monitor("in = %b ---> y = %b ", in, y);
endmodule
```

测试结果

测试 N = 3：

波形图：

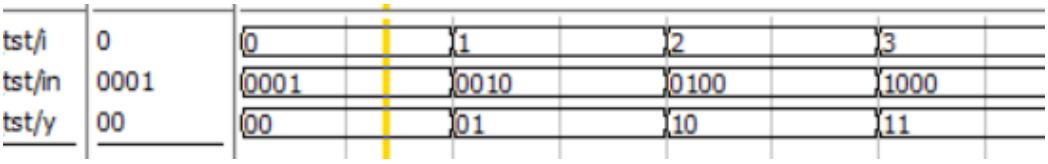


输出信息：

```
VSIM(paused)> run -all
# in = 00000001 ---> y = 000
# in = 00000010 ---> y = 001
# in = 00000100 ---> y = 010
# in = 00001000 ---> y = 011
# in = 00010000 ---> y = 100
# in = 00100000 ---> y = 101
# in = 01000000 ---> y = 110
# in = 10000000 ---> y = 111
```

测试 N = 2：

波形图：



输出信息：

```
VSIM(paused)> run -all
# in = 0001 ---> y = 00
# in = 0010 ---> y = 01
# in = 0100 ---> y = 10
# in = 1000 ---> y = 11
```

可知测试结果正确

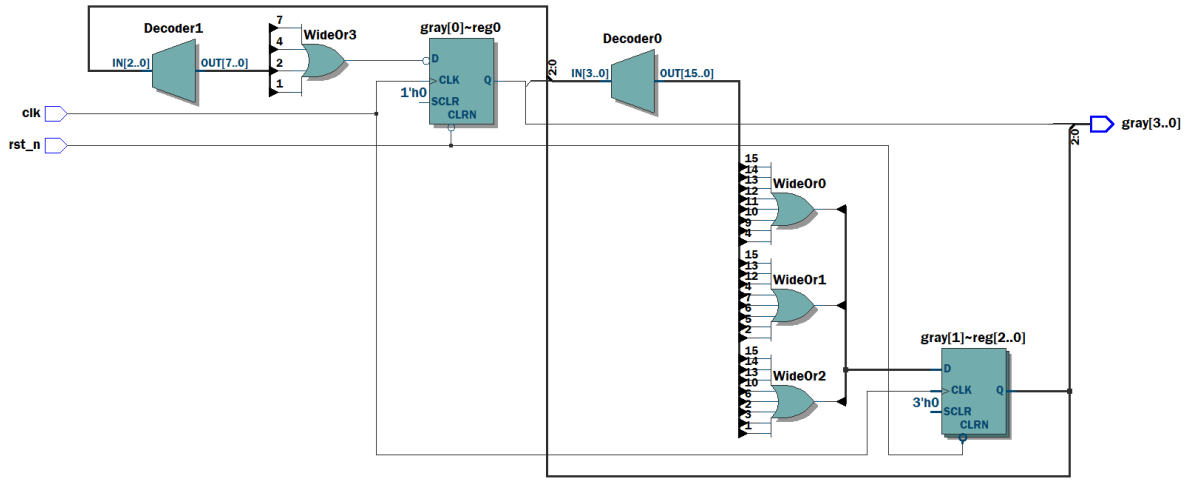
设计格雷码计数器

代码实现

```
module gray_counter(
    input clk,
    input rst_n,
    output reg[3:0]gray
);
    always@(posedge clk or negedge rst_n)
    if(!rst_n)  gray <= 4'b0000;
    else
        case(gray)
            4'b0000 : gray <= 4'b0001;
            4'b0001 : gray <= 4'b0011;
            4'b0011 : gray <= 4'b0010;
            4'b0010 : gray <= 4'b0110;
            4'b0110 : gray <= 4'b0111;
            4'b0111 : gray <= 4'b0101;
            4'b0101 : gray <= 4'b0100;
            4'b0100 : gray <= 4'b1100;
            4'b1100 : gray <= 4'b1101;
            4'b1101 : gray <= 4'b1111;
            4'b1111 : gray <= 4'b1110;
            4'b1110 : gray <= 4'b1010;
            4'b1010 : gray <= 4'b1011;
            4'b1011 : gray <= 4'b1001;
```

```
            4'b1001 : gray <= 4'b1000;
            4'b1000 : gray <= 4'b0000;
            default      : gray <= 4'bx;
        endcase

endmodule
```

RTL VIEWER:

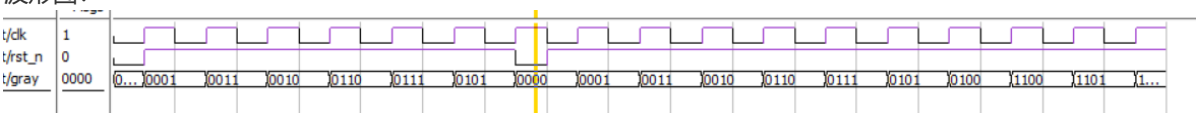


testbench:

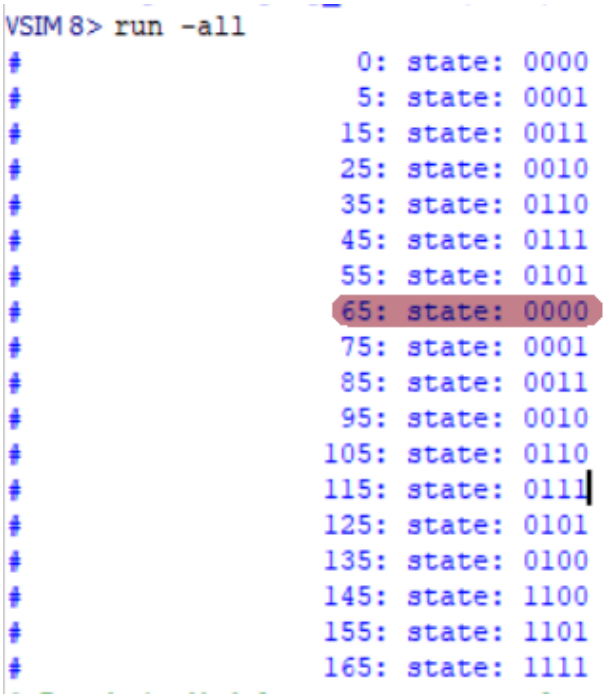
```
`timescale 1 ns/ 1 ps
module gray_counter_vlg_tst();
reg clk;
reg rst_n;
wire [3:0]  gray;

gray_counter i1 (
    .clk(clk),
    .gray(gray),
    .rst_n(rst_n)
);

initial
begin
    rst_n = 0;
    clk = 0;
    #5 rst_n = 1;
    #60 rst_n = 0;
    #5 rst_n = 1;
    #100 $stop;
end

always #5 clk = ~clk;

initial $monitor($time,": state: %b",gray);
endmodule
```

测试结果

波形图:



输出信息:



标红项为测试中复位信号作用的位置。

实验收获

1. $stop 的用法

$stop$任务得作用是把EDA工具(例如仿真器)置为暂停模式，在仿真环境下给出一个交互式的命令提示符，将控制权交给用户。格式与 finish 类似：
当$ stop 带参数时，根据不同的参数值，系统输出的特征信息：

- 0：不输出任何信息；
- 1：输出当前仿真时刻和位置；
- 2：输出当前仿真时刻、位置和仿真过程中所用的 memory 及 CPU 时间的统计。
  当$stop$后面不带参数时，则默认参数为1。使用 finish 会在仿真时间结束时，退出仿真器，而在 testbench 程序最后使用$ stop 则可以使仿真暂停，便于观察输出信息。

2. 含参例化
带参数模块可以通过普通例化方式例化，如果需要在例化时更改参数值，有两种方式：
第一种，使用 defparam 语句
比如：

```
defparam i1.n = 2;
decoder_coder i1 (
    .in(in),
    .y(y)
);
```

第二种，例化模块时，将新的参数值写入模块例化语句
比如：

```
decoder_coder #(.n(N), .m(M))
    i1 (
    .in(in),
    .y(y)
);
```

3. disable

quartus 的语法中不支持 break，实验中使用了 verilog 的 disable 语句。

disable 语句可以退出任何循环，也可以用来终止 task，但是必须要配合 begin...end... 指令使用。

verilog 的 disable 声明和 system verilog 的跳转声明的一个重要区别是：disable 声明适用于所有现行的 task 或者 block 的 invocation，然而 break，continue 和 return 只适用于当前的执行流程。