刘恋　2022302181085

## 3.4、

### （1）if_else 语句：

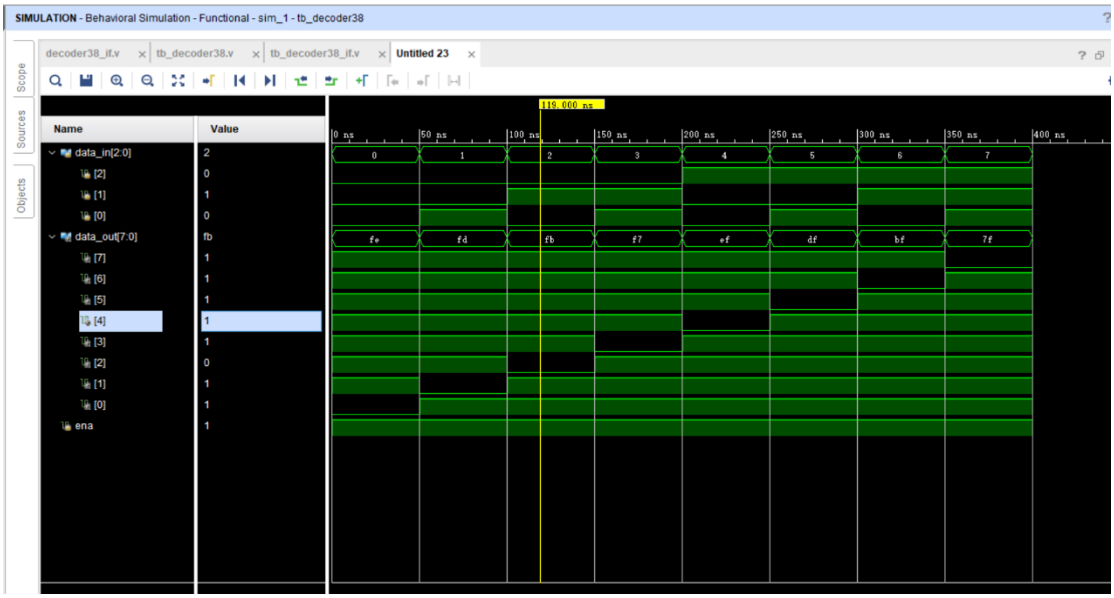
源代码：

```
module decoder38(data_in,data_out,ena);
        input [2:0] data_in;
        input ena;
        output reg [7:0] data_out;
        always@(data_in or ena)begin
            if(ena==1)
                if(data_in==3'b000)    data_out = 8'b1111_1110;
                    else if(data_in==3'b001)    data_out = 8'b1111_1101;
                    else if(data_in==3'b010)    data_out = 8'b1111_1011;
                    else if(data_in==3'b011)    data_out = 8'b1111_0111;
                    else if(data_in==3'b100)    data_out = 8'b1110_1111;
                    else if(data_in==3'b101)    data_out = 8'b1101_1111;
                    else if(data_in==3'b110)    data_out = 8'b1011_1111;
                    else if(data_in==3'b111)    data_out = 8'b0111_1111;
                else    data_out = 8'bxxxxxxxx;
            else data_out = 8'b000000;
         end
endmodule
```

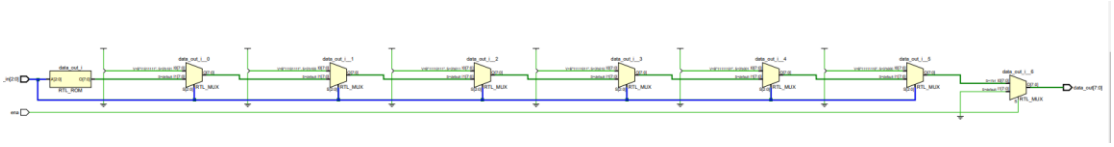
仿真代码：

```
module tb_decoder38;
        reg [2:0] data_in;    wire [7:0] data_out;    reg ena;
        initial begin
                data_in = 3'b000;ena = 1;
                #50         data_in = 3'b001;ena = 1;
                #50         data_in = 3'b010;ena = 1;
                #50         data_in = 3'b011;ena = 1;
                #50         data_in = 3'b100;ena = 1;
                #50         data_in = 3'b101;ena = 1;
                #50         data_in = 3'b110;ena = 1;
                #50         data_in = 3'b111;ena = 1;
                #50         $stop;
            end
         decoder38 txt1(.data_in(data_in), .data_out(data_out), .ena(ena));
endmodule
```

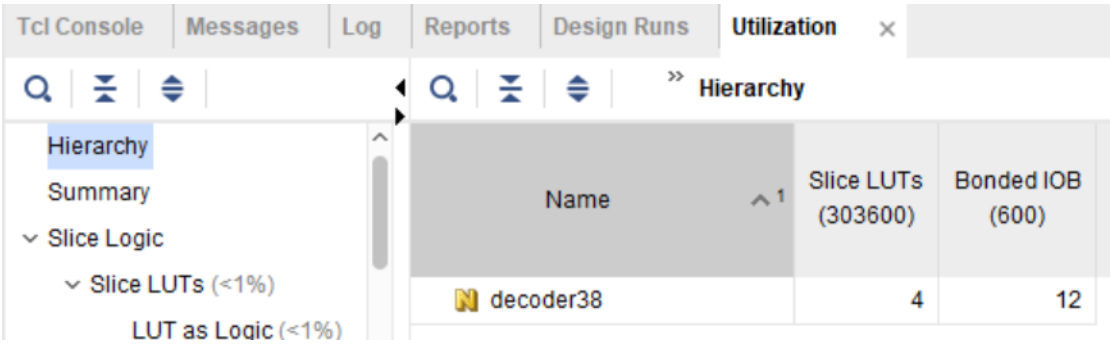
仿真图：




RTL 原理图：



资源开销：




## case 语句：

源代码：

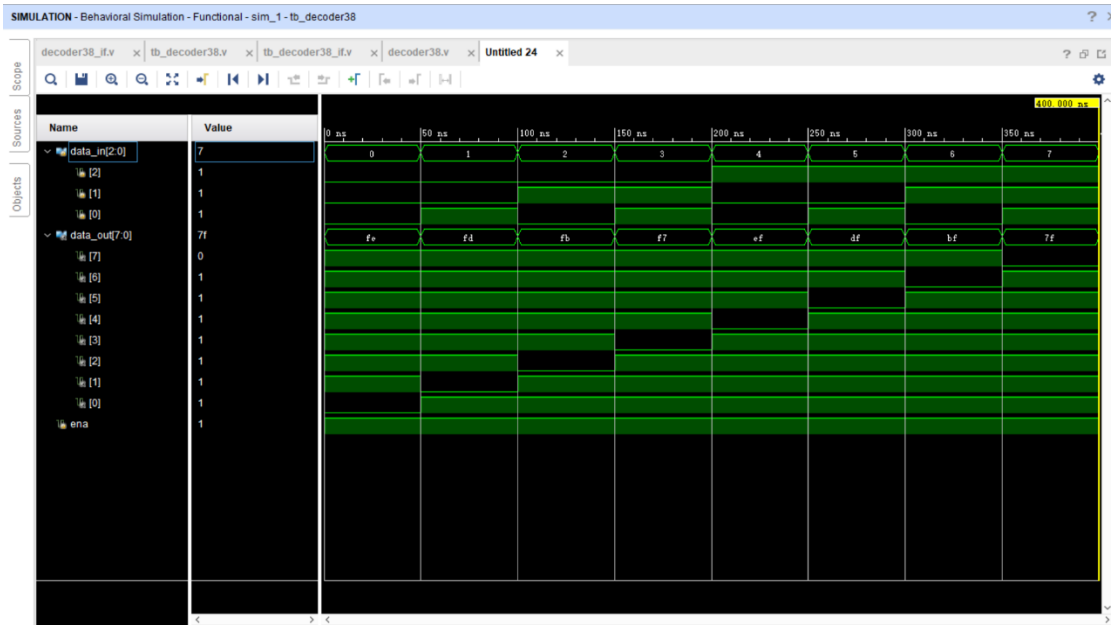
```
module decoder38(data_in,data_out,ena);
        input [2:0] data_in;    input wire ena;    output reg [7:0] data_out;
        always@(data_in)begin
            case(data_in)
```

```
                3'b000 : data_out = 8'b1111_1110;
                3'b001 : data_out = 8'b1111_1101;
                3'b010 : data_out = 8'b1111_1011;
                3'b011 : data_out = 8'b1111_0111;
                3'b100 : data_out = 8'b1110_1111;
                3'b101 : data_out = 8'b1101_1111;
                3'b110 : data_out = 8'b1011_1111;
                3'b111 : data_out = 8'b0111_1111;
                default : data_out = 8'b0000_0000;
            endcase
        end
endmodule
```

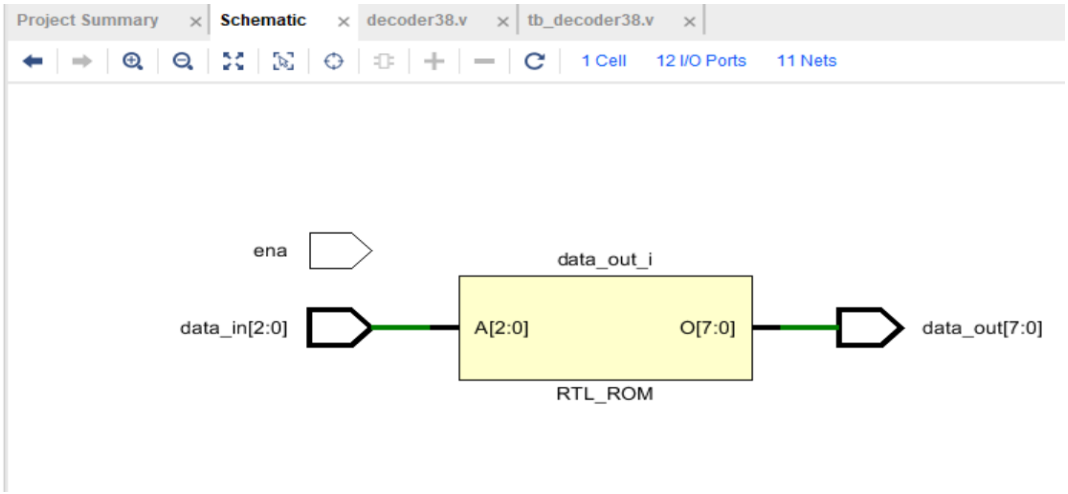
仿真代码：

```
module tb_decoder38;
        reg [2:0] data_in;    wire [7:0] data_out;    reg ena;
        initial begin
                data_in = 3'b000;
                ena = 1;
                #50
                data_in = 3'b001;
                ena = 1;
                #50
                data_in = 3'b010;
                ena = 1;
                #50
                data_in = 3'b011;
                ena = 1;
                #50
                data_in = 3'b100;ena = 1;
                #50
                data_in = 3'b101;ena = 1;
                #50
                data_in = 3'b110;ena = 1;
                #50
                data_in = 3'b111;ena = 1;
                #50
                $stop;
            end
        decoder38 txt1(.data_in(data_in), .data_out(data_out), .ena(ena));
endmodule
```

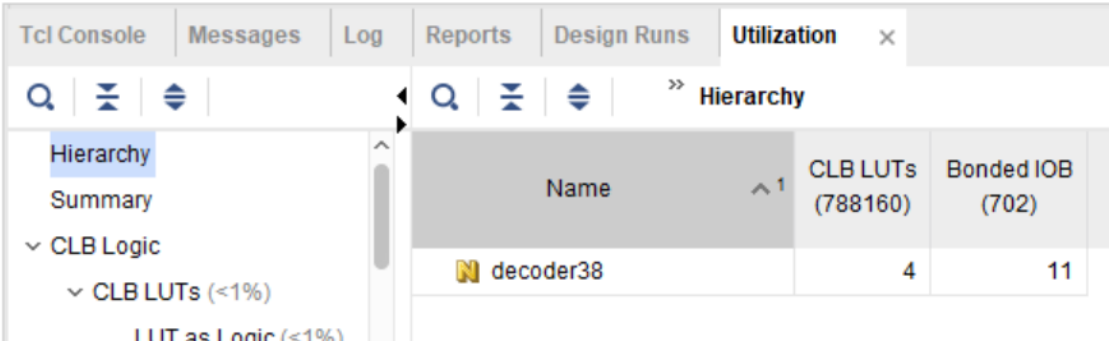
仿真图：




RTL 原理图：




资源开销：



比较：if_else 语句的逻辑判断是有优先级地，case 的逻辑判断是并列的。在该题中两者的仿真代码与仿真图形是一样的。但是两者 RTL 原理图以及资源开销不同，此题中，case 语句资源开销比 if_else 语句少了用一个 IOB。

3.8、

源代码：

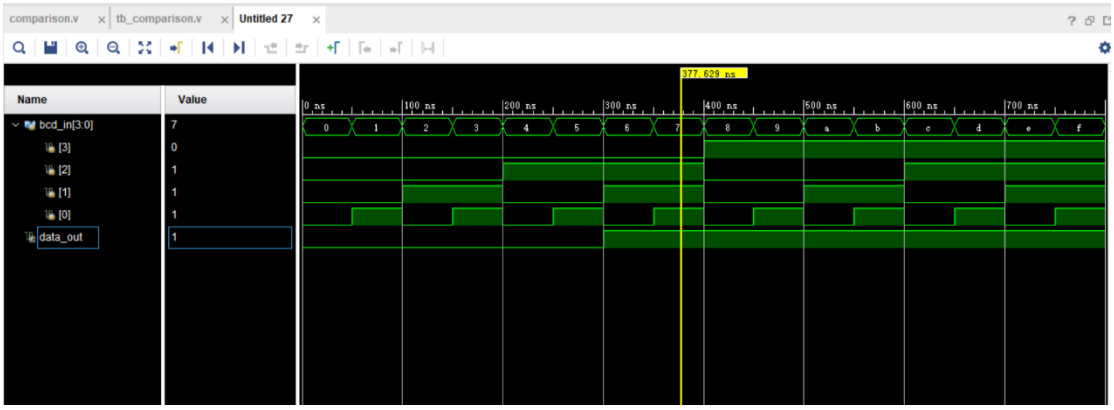
```
module comparison(bcd_in, data_out);
        input [3:0] bcd_in;
        output reg data_out;
        always@(*)begin
        if(bcd_in > 5)
              data_out = 1;
        else
              data_out = 0;
      end
endmodule
```

仿真代码：

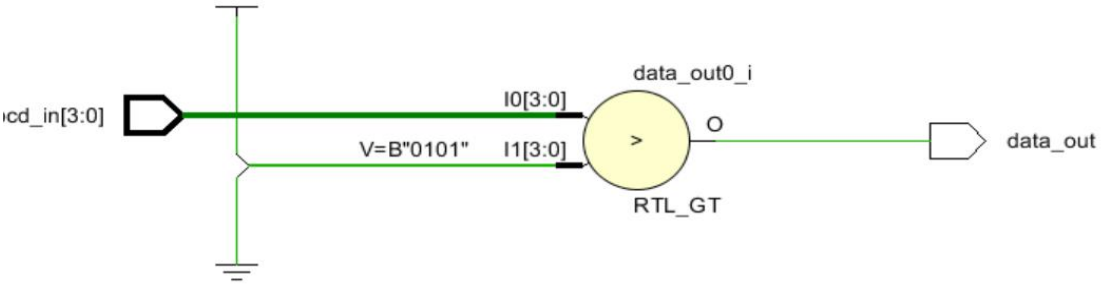
```
module tb_comparison;
            reg [3:0] bcd_in;
            wire data_out;
            initial begin
                        bcd_in = 4'b0000;
                        #50     bcd_in = 4'b0001;
                        #50     bcd_in = 4'b0010;
                        #50     bcd_in = 4'b0011;
                        #50     bcd_in = 4'b0100;
                        #50     bcd_in = 4'b0101;
                        #50     bcd_in = 4'b0110;
                        #50     bcd_in = 4'b0111;
                        #50     bcd_in = 4'b1000;
                        #50     bcd_in = 4'b1001;
                        #50     bcd_in = 4'b1010;
                        #50     bcd_in = 4'b1011;
                        #50     bcd_in = 4'b1100;
                        #50     bcd_in = 4'b1101;
```

```
        #50     bcd_in = 4'b1110;
        #50     bcd_in = 4'b1111;
        #50     $stop;
    end
    comparison txt1(.bcd_in(bcd_in), .data_out(data_out));
endmodule
```

仿真图：




RTL 原理图：




资源开销：

## 3.19、

源代码：

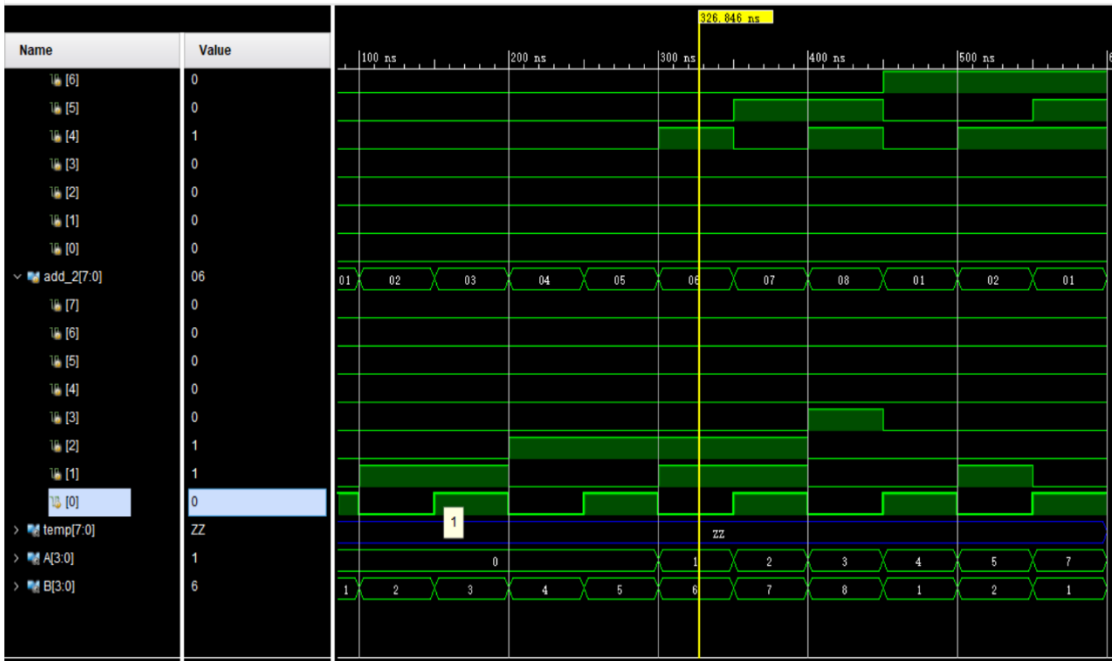
```
`timescale 1ns / 1ps
module add_full(A,B,C,Carry,S);
        input A,B,C;
        output Carry,S;
        assign S = A^B^C;
        assign Carry = (A&B)|(B&C)|(A&C);
endmodule

module add(add_1, add_2,A,B);
    input [7:0] add_1;
    input [7:0] add_2;
    //input cin;
    output   [3:0] A;
    output   [3:0] B;
    wire [7:0] temp;
    wire [8:0] C;
    assign C[0] = 0;
    add_full u1(add_1[0], add_2[0], C[0], C[1], temp[0]),
             u2(add_1[1],add_2[1],C[1], C[2], temp[1]),
             u3(add_1[2],add_2[2],C[2],C[3], temp[2]),
             u4(add_1[3],add_2[3],C[3],C[4], temp[3]),
             u5(add_1[4],add_2[4],C[4],C[5], temp[4]),
             u6(add_1[5],add_2[5],C[5],C[6], temp[5]),
             u7(add_1[6],add_2[6],C[6],C[7], temp[6]),
             u8(add_1[7],add_2[7],C[7],C[8], temp[7]);
    assign A = {temp[7],temp[6],temp[5],temp[4]};
    assign B = {temp[3],temp[2],temp[1],temp[0]};
endmodule
```

仿真代码：

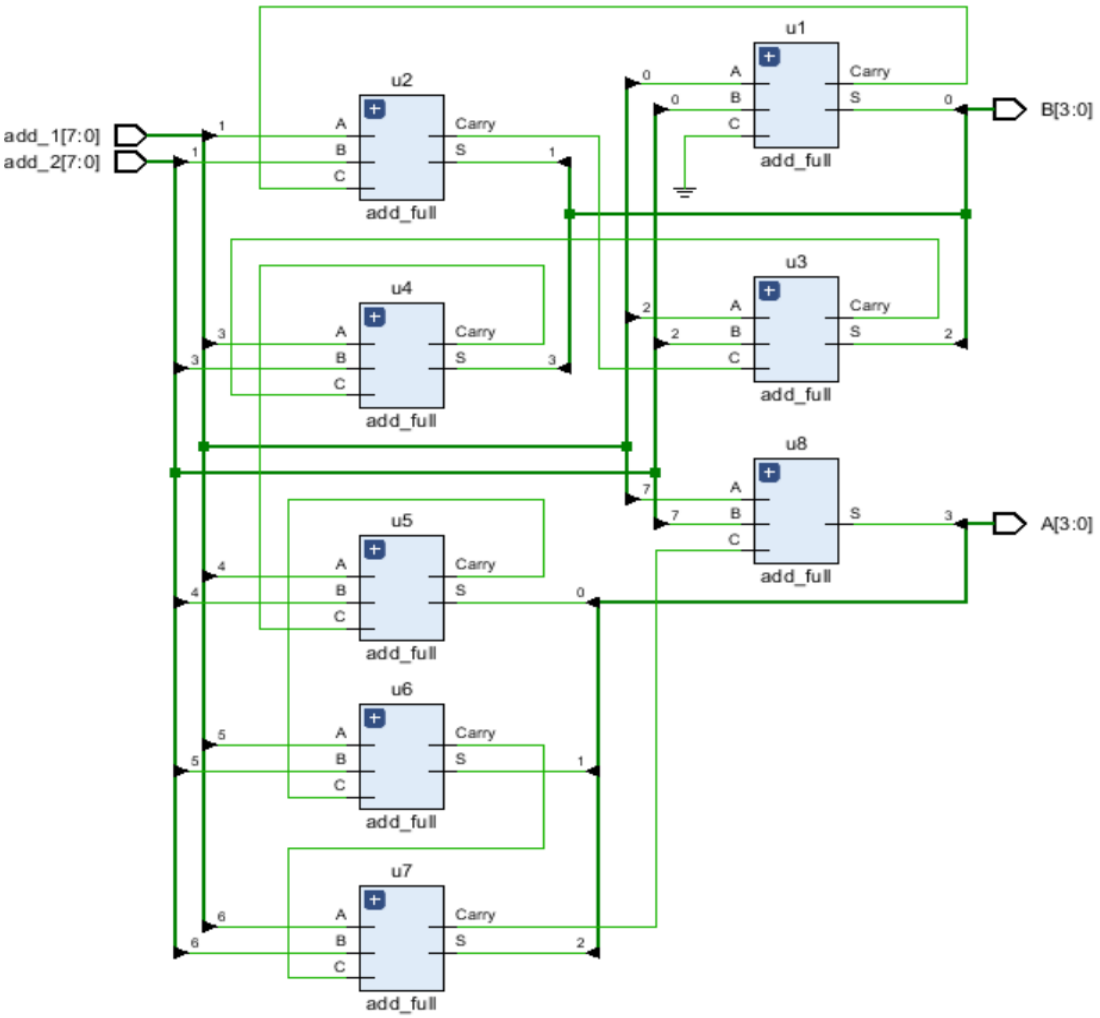
```
`timescale 1ns / 1ps
module tb_add;
        reg [7:0] add_1; reg [7:0] add_2;
        wire [7:0] temp; wire [3:0] A;
        wire [3:0] B;
        initial begin
                add_1 = 8'b0000_0000;   add_2 = 8'b0000_0000;
                #50
                add_1 = 8'b0000_0000;   add_2 = 8'b0000_0001；
                #50
```

```
        add_1 = 8'b0000_0000;    add_2 = 8'b0000_0010;
        #50
        add_1 = 8'b0000_0000;    add_2 = 8'b0000_0011;
        #50
        add_1 = 8'b0000_0000;    add_2 = 8'b0000_0100;
        #50
        add_1 = 8'b0000_0000;    add_2 = 8'b0000_0101;
        #50
        add_1 = 8'b0001_0000;    add_2 = 8'b0000_0110;
        #50
        add_1 = 8'b0010_0000;    add_2 = 8'b0000_0111;
        #50
        add_1 = 8'b0011_0000;    add_2 = 8'b0000_1000;
        #50
        add_1 = 8'b0100_0000;    add_2 = 8'b0000_0001;
        #50
        add_1 = 8'b0101_0000;    add_2 = 8'b0000_0010;
        #50
        add_1 = 8'b0111_0000;    add_2 = 8'b0000_0001;
        #50      $stop;
    end
    add txt1(.add_1(add_1), .add_2(add_2),.A(A),.B(B) );
endmodule
```

仿真图：



RTL 原理图：




资源分析：

Reports | Design Runs | Utilization

Hierarchy

| Name | Slice LUTs (303600) | Bonded IOB (600) |
|---|---|---|
| add | 8 | 24 |

## 5.4、

源代码：

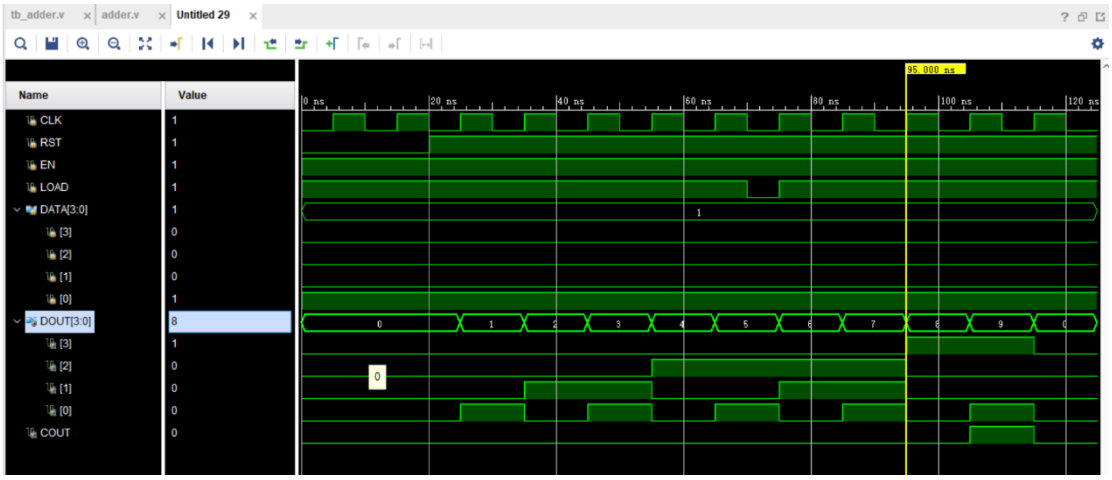
```
`timescale 1ns / 1ps
module adder( CLK,RST,EN,LOAD,COUT,DOUT,DATA);
        input CLK,EN,RST,LOAD;
        input [3:0] DATA;
        output [3:0] DOUT;
        output [3:0] COUT;
        reg [3:0] Q1;
        reg COUT;
        assign DOUT = Q1;
        always @(posedge CLK or negedge RST)
          begin
          if(!RST) Q1 <= 0;//RST=0 时，对内部寄存器单元异步清 0
          else if (!LOAD) Q1 <= DATA;
          else if(EN) begin
                if(Q1 < 9) Q1 <= Q1 + 1;
                else Q1 <= 4'b0000;
                end
             end
           always@(Q1)
           if(Q1 == 4'h9) COUT = 1'b1;
           else COUT = 1'b0;
endmodule
```

仿真代码：

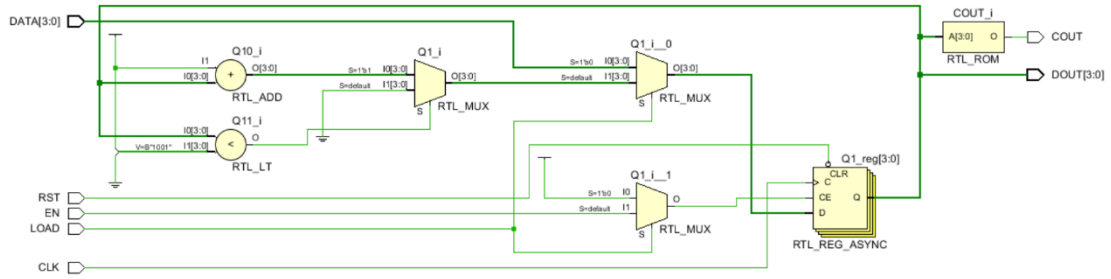
```
`timescale 1ns / 1ps
module tb_adder;
          reg CLK,RST,EN,LOAD;      reg [3:0] DATA;
          wire [3:0] DOUT;      wire COUT;
          initial
              CLK = 0;
          always
             #5 CLK = ~CLK;

          initial begin
                    RST = 0;
                    LOAD = 1;
                    EN = 1;
                    DATA = 4'h11;
                     #20    RST = 1;
                     #50    LOAD = 0;;
```

```
        #5      LOAD = 1;
        #50     $finish;
    end
adder txt1(.CLK(CLK),.RST(RST),.EN(EN),.LOAD(LOAD),.COUT(COUT),.DOUT(DOUT),.DATA(DATA));
endmodule
```

仿真图：




RTL 原理图：




资源分析：


| Name | Slice LUTs (303600) | Slice Registers (607200) | Bonded IOB (600) | BUFGCTRL (32) |
|---|---|---|---|---|
| adder | 7 | 4 | 13 | 1 |

## 5.5、

源代码：

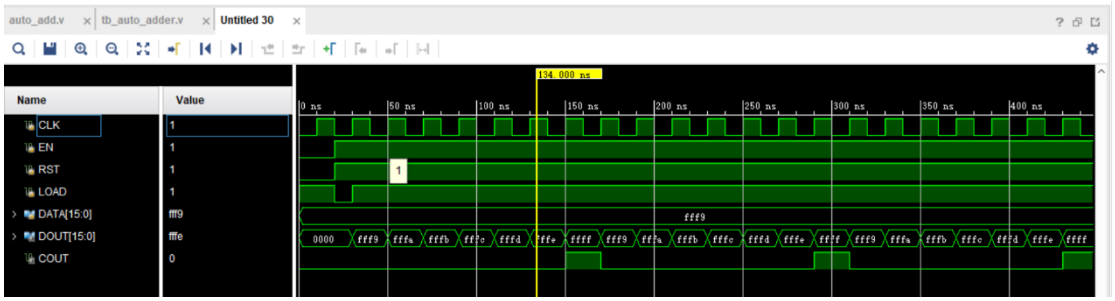
```
`timescale 1ns / 1ps
module auto_add(CLK, RST, EN, LOAD, COUT, DOUT, DATA);
    input CLK, EN, RST, LOAD;    input [15:0] DATA;
    output [15:0] DOUT, COUT;
    reg[15:0] Q1;    reg COUT;    reg FULL;
    assign DOUT = Q1;
    always @(posedge CLK or posedge LOAD or negedge RST)
    begin
        if(!RST)  //异步复位信号
        begin
            Q1 <= 0;    FULL <= 0;
        end
        else if(LOAD)  //异步加载信号
        begin
            Q1 <= DATA;
        end
        else if(EN)  //同步使能信号
        begin
            Q1 <= Q1 + 1;
        end
    end

    always @(Q1)
    begin
        if(Q1 == 16'b1111111111111111)
            COUT = 1'b1;
        else
            COUT <= 1'b0;
    end
    assign LOAD = (Q1 == 16'd0) ; //产生加载信号
    assign DOUT = Q1;                  //数据
endmodule
```

仿真代码：

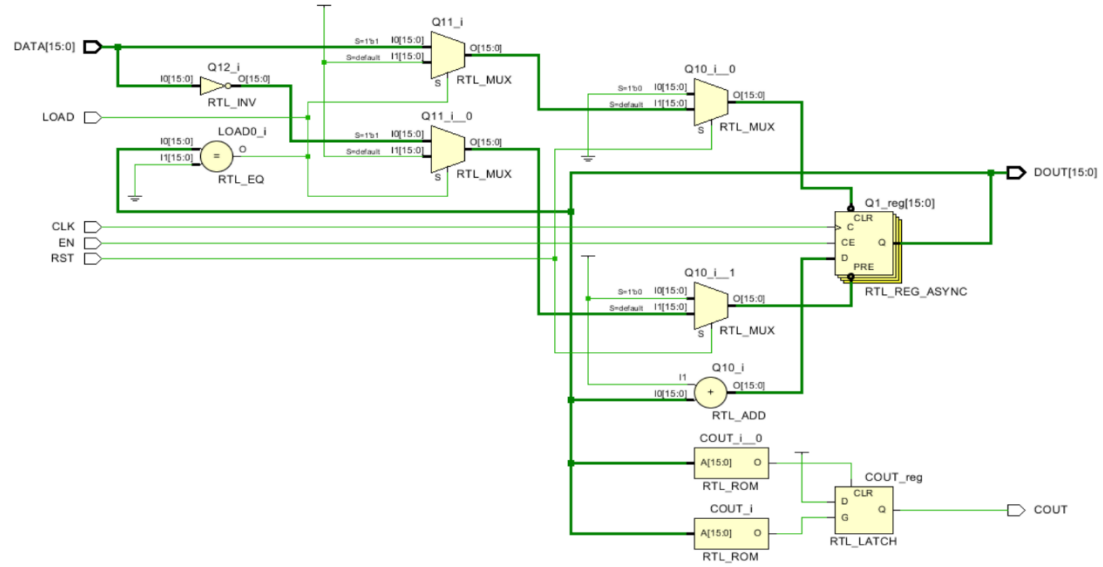
```
`timescale 1ns / 1ps
module tb_auto_adder();
    reg CLK, EN, RST, LOAD;    reg [15:0] DATA;
    wire [15:0] DOUT;    wire COUT;
    initial  begin
        RST = 0;    CLK = 0;
        LOAD = 1;    EN = 0;
```

```
        DATA = 16'b1111111111111001;
        # 20    RST = 1;    EN = 1;    LOAD = 0;
        # 10    LOAD = 1;
    end
    always
     # 10    CLK = ~CLK;
    auto_add
txt1(.CLK(CLK),.RST(RST),.EN(EN),.LOAD(LOAD),.COUT(COUT),.DOUT(DOUT),.DATA(DATA));
endmodule
```

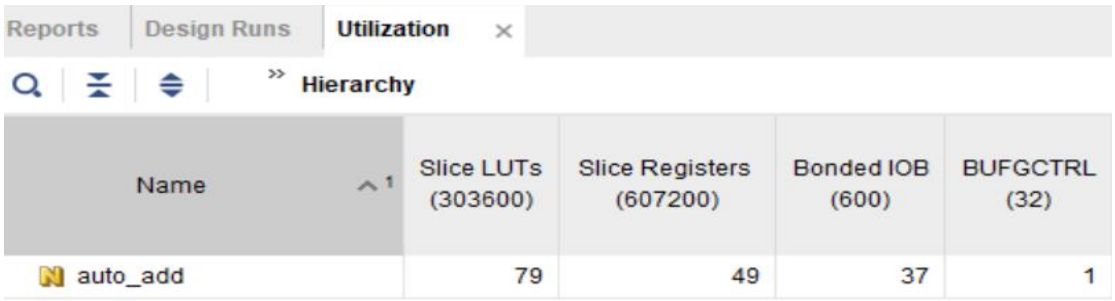
仿真图：




RTL 原理图：




资源分析：




| Name | Slice LUTs (303600) | Slice Registers (607200) | Bonded IOB (600) | BUFGCTRL (32) |
|---|---|---|---|---|
| auto_add | 79 | 49 | 37 | 1 |

## 5.8、

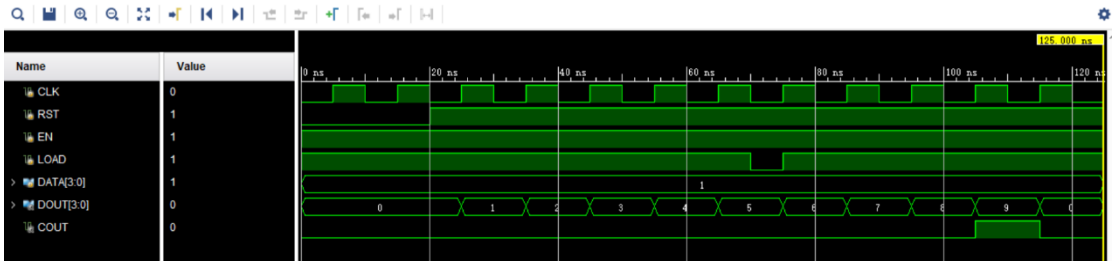
源代码：

```
`timescale 1ns / 1ps
module counter_74LS160(CLK, RST, EN, LOAD, COUT, DOUT, DATA);
    input CLK, RST, EN, LOAD;       input [3:0] DATA;
    output [3:0] DOUT;      output COUT;
    reg[3:0] Q1;       reg COUT;
    assign DOUT = Q1;
    always @(posedge CLK or negedge RST)
    begin
        if(!RST)     //异步清零
            Q1 <= 0;
        else if(EN)    //同步使能
        begin
            if(!LOAD)    Q1 <= DATA;
            else if(Q1 < 9)    Q1 <= Q1 + 1;
            else      Q1 <= 4'b0000;
        end
    end
    always @(Q1)
        if(Q1 == 4'h9)     COUT = 1'b1;
        else      COUT = 1'b0;
endmodule
```

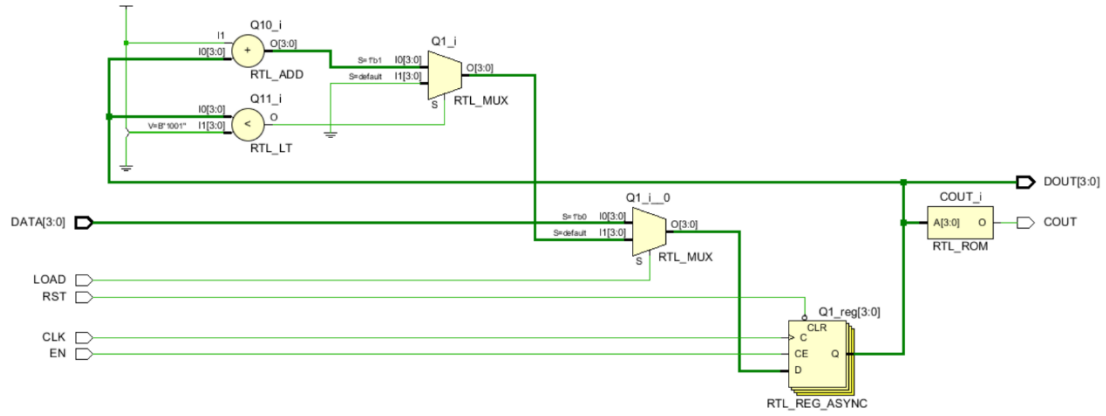
仿真代码：

```
`timescale 1ns / 1ps
module tb_counter_74LS160;
        reg CLK,RST,EN,LOAD;       reg [3:0] DATA;
        wire [3:0] DOUT;       wire COUT;
        initial
            CLK = 0;
        always
           #5 CLK = ~CLK;
        initial begin
                RST = 0;       LOAD = 1;       EN = 1;       DATA = 4'h11;
                 #20    RST = 1;
                 #50    LOAD = 0;;
                 #5    LOAD = 1;
                 #50    $finish;
            end
         counter_74LS160
txt1(.CLK(CLK),.RST(RST),.EN(EN),.LOAD(LOAD),.COUT(COUT),.DOUT(DOUT),.DATA(DATA));
endmodule
```

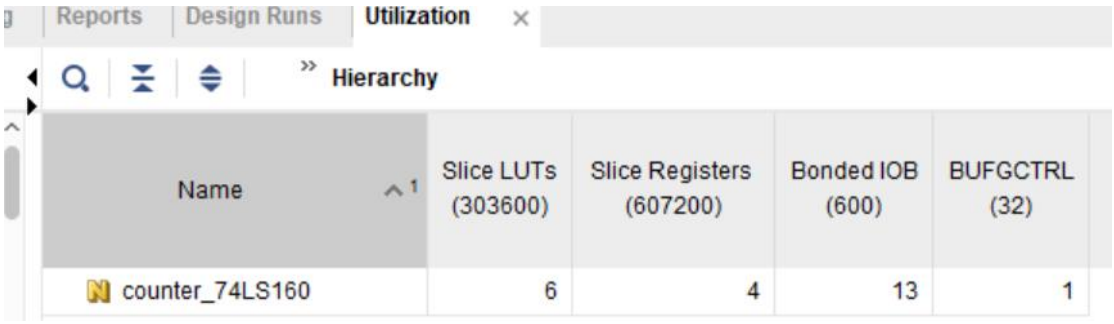
仿真图：




RTL 原理图：




资源分析：




## 5.9、

源代码：

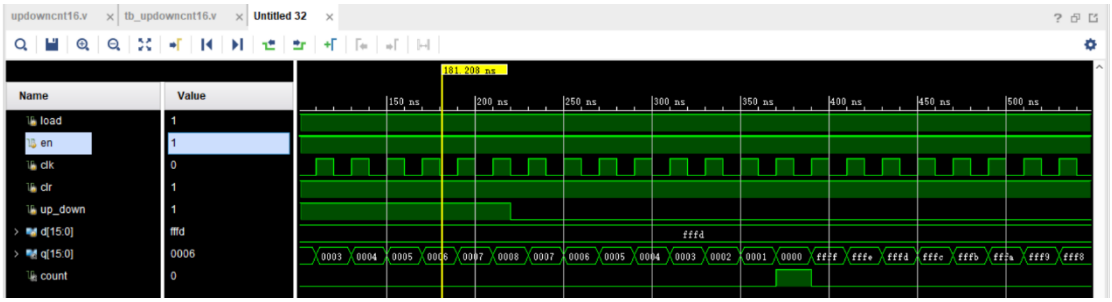
```
`timescale 1ns / 1ps
module updowncnt16(q, count, d, load, en, clk, clr, up_down);
        input [15:0] d;
        input load, en, clk, clr, up_down;
        output [15:0] q;      reg [15:0] q;      output count;
        always @(posedge clk or negedge clr)
        begin
```

```
            if(!clr)        q = 16'd0;
            else if(en)
            begin
                if(!load)       //同步加载数据        q = d;
                if(up_down)       q = q + 1;     //做加法
                else        q = q - 1;     //做减法
                end
              end
              assign count = up_down ? &q : ~|q;
endmodule
```

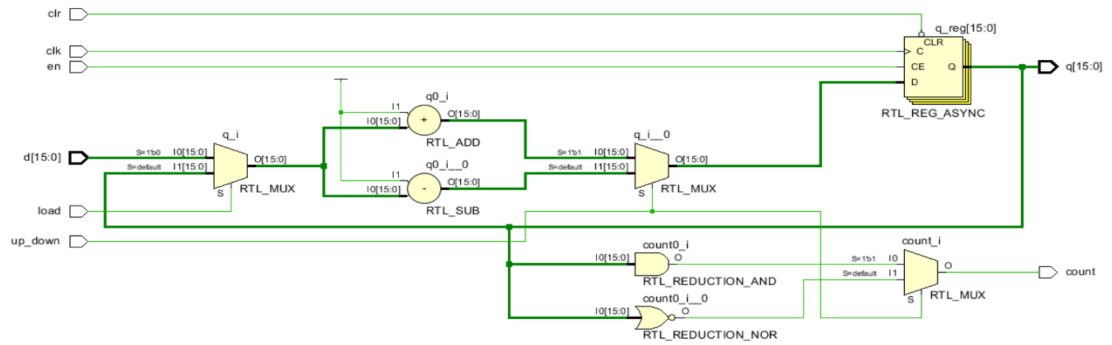
仿真代码：

```
`timescale 1ns / 1ps
module tb_updowncnt16;
        reg load,en,clk,clr,up_down;      reg [15:0] d;      wire [15:0] q;      wire count;
        initial begin
            clr = 0;    load = 1;    en = 0;    clk = 0;
            d = 16'b1111_1111_1111_1101;    up_down = 1;
            #10    clr = 1;    load = 0;    en = 1;
            #10    load = 1;
            #200    up_down = 0;      end
          always    #10    clk = ~clk;
         updowncnt16 txt1(.q(q),.count(count),.d(d),.load(load),.en(en),.clk(clk),.clr(clr),.up_down(up_down));
endmodule
```

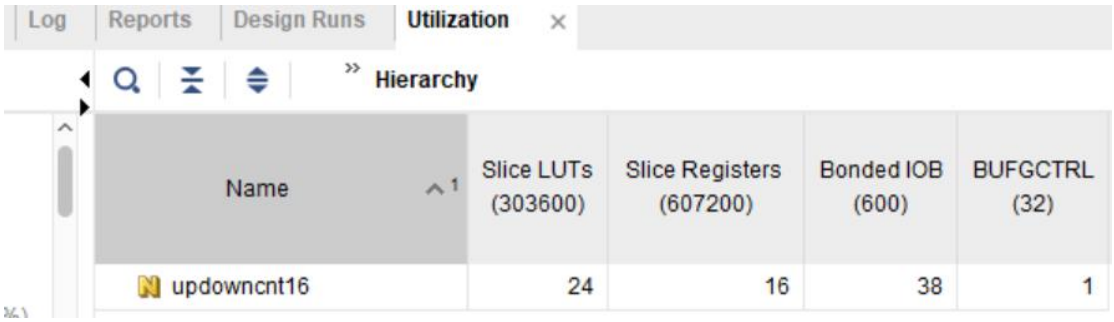
仿真图：




RTL 原理图：



资源分析：




10.1、

代码如下：

```
`timescale 1ns / 1ps
module inprovement10_1(clk,reset,state_inputs,comb_outputs);
        input clk,reset;        input [0:1] state_inputs;
        output [3:0] comb_outputs;        reg [3:0] comb_outputs;
        parameter s0=0,s1=1,s2=2,s3=3,s4=4;
        reg [4:0] c_st,next_state;
        always @(posedge clk or negedge reset) begin
            if(!reset) c_st <= 0;
            else    c_st <= next_state;
            end
        //过程 1：负责状态转换
        always @(c_st or state_inputs) begin
         case(c_st)
         s0:begin
                    if(state_inputs == 2'b00)       next_state <= s0;
                    else         next_state <= s1;
                end
         s1:begin
                    if(state_inputs == 2'b01)       next_state <= s1;
                    else       next_state <= s2;
                end
         s2:begin
                    if(state_inputs == 2'b10)       next_state <= s0;
                    else       next_state <= s3;
                end
         s3:begin
                    if(state_inputs == 2'b11)       next_state <= s3;
                    else       next_state <= s4;
                end
```

```
            s4:begin
                    next_state <= 0;
                end
            default:next_state <= s0;
            endcase
            end

            //过程 2：负责输出控制信号
             always @(c_st or state_inputs) begin
                    case(c_st)
                    s0:begin
                            comb_outputs <= 5;
                        end
                    s1:begin
                            comb_outputs <= 8;
                        end
                    s2:begin
                            comb_outputs <= 12;
                        end
                    s3:begin
                            comb_outputs <= 14;
                        end
                    s4:begin
                            comb_outputs <= 9;
                        end
                    endcase
                    end
endmodule
```