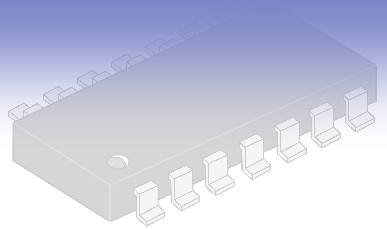
第 12 章

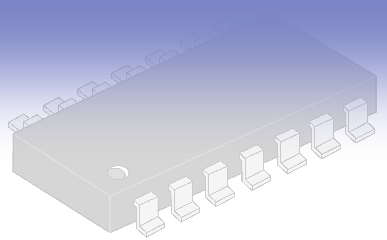
使用 Verilog HDL 进行综合

内容



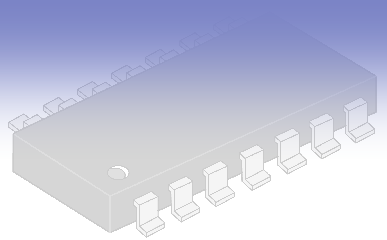
- 逻辑综合的概念
- Verilog 综合

逻辑综合的概念



- 基础
  - 标准单元库
    - 包含基本的单元，例如：与门、或门和或非门等基本逻辑门
    - 包含宏单元，例如：加法器、多路选择器和触发器。
  - 特定的设计约束条件
- 综合
  - 把设计的高层次（Verilog HDL / System Verilog / VHDL / System C ）描述转换成优化的门级网表
- 工具
  - 综合工具
    - 自动完成从高层次描述向逻辑门的转化过程
    - 将 Verilog 描述的电路设计转换成标准的门级结构网表
  - 布线工具
    - 根据综合后生成的标准的门级结构网表，产生具体的电路

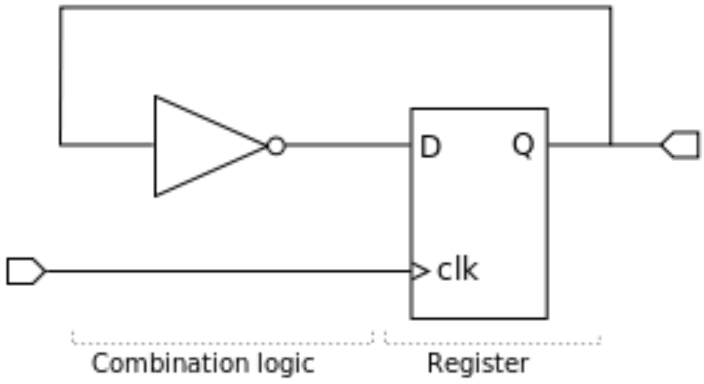
可综合性



- 可综合的 Verilog HDL 只是 Verilog 标准中的一个子集
  - 各厂商的的综合工具所支持的 HDL 也不尽相同
- 面向综合的 Verilog 模块设计
  - 目标是能够综合、生成电路
  - 设计时必须考虑使用可综合性问题
  - 使用 Verilog 中的可综合子集
  - 采用寄存器传输级（RTL）的编码风格进行设计
  - 必须符合所使用综合工具的要求
- 面向仿真测试的Verilog模块设计
  - 不考虑可综合性
  - 用于描述对模块功能进行仿真的测试平台的设计
  - 几乎可以使用Verilog提供的所有语句和全部功能
    - 部分语法特征受到使用的仿真工具的限制

寄存器传输级

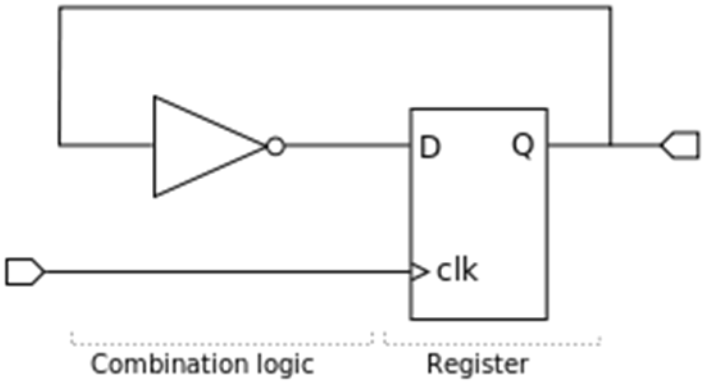
- 采用寄存器传输级（RTL）的HDL设计能够用于逻辑综合
- 寄存器传输级（RTL）
  - RTL ——基于同步逻辑，HDL设计的一种编码风格
  - 三个主要部分
    - 保存状态信息的寄存器、产生下一个状态的组合逻辑、控制状态变化的时钟
  - 综合工具能够将寄存器传输级HDL描述转化为优化的门级网表
- 高级综合（HLS）
  - 用于把基于C++/C的算法描述转换成RTL描述
- 目前采用的主要设计方法
  - 基于RTL的综合
  - HLS 综合

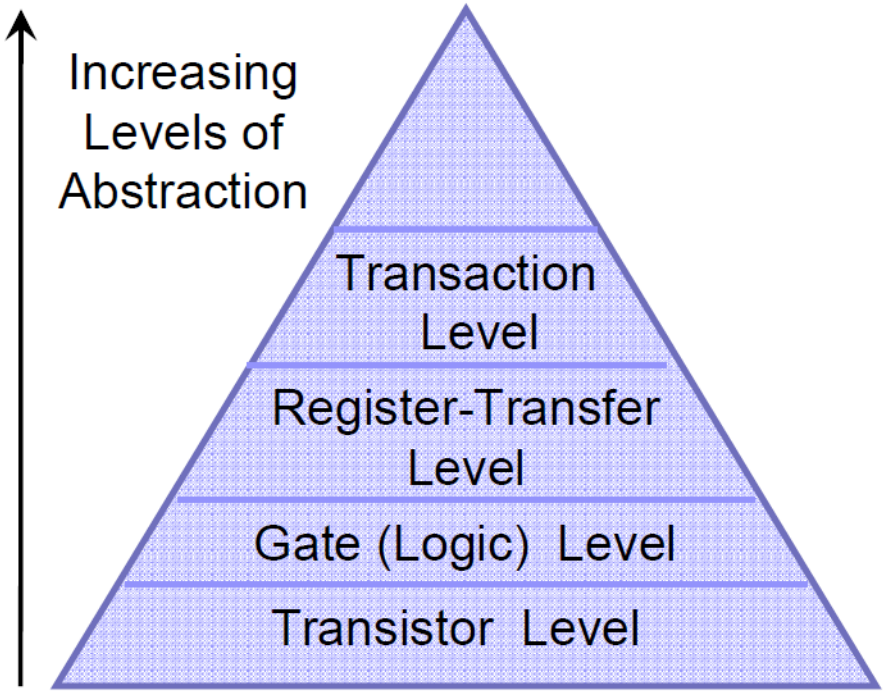


数字系统设计抽象层次

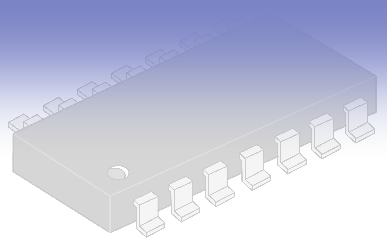
数字系统设计的几个抽象层次

- 事务级建模（Transaction Level Modeling）
  - 抽象的通信机制
- 寄存器传输级
  - 使用寄存器、数据通道（datapath ）和控制器
  - 控制数据通过数据通道（datapath）在寄存器直接进行传输
- 门级
  - 使用基本的门



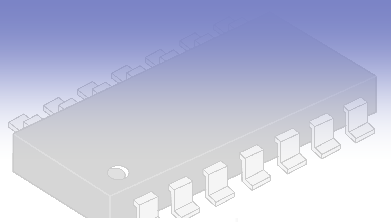


Verilog 结构



- 用于逻辑综合的Verilog模块设计不能随意编写
  - 并非符合Verilog语法标准的结构描述都能综合
- 可进行逻辑综合的Verilog HDL结构
  - 端口 —— input，inout，output
  - 参数 —— parameter
  - 模块定义——module
  - 信号和变量——wire, reg，tri，允许使用向量表示
  - 调用（实例引用）—— module instance primitive gate instance
  - 函数和任务 —— function，task，不考虑时序结构
  - 过程 —— always，if, then, else, case，不支持initial
  - 过程块—— begin, end, named blocks, disable，命名块的禁止是允许的
  - 数据流——assign，不考虑延迟信息
  - 循环 —— for, while，forever
    - while 和 forever 循环必须包含 @(posedge clock) 或 @(negedge clock)
- 不同综合工具支持的 Verilog 结构也不完全相同

Verilog Support —— Quartus II v13.1



Quartus II Help v13.1 > Quartus II Verilog HDL Support

Show All

Quartus II Verilog HDL Support

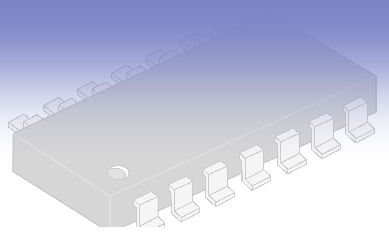
Quartus II support for Verilog HDL is described for the following categories of Verilog HDL constructs. These sections match those in the IEEE Std 1394-2001 *IEEE Hardware Description Language Based on the Verilog Hardware Description Language* manual.

- **Section 3—Data Types:**
- **Section 4—Expressions:**
- **Section 6—Assignments:**
- **Section 7—Gates and Switches:**
- **Section 8—User Defined Primitives:**
- **Section 9—Behavioral Modeling:**
- **Section 10—User Defined Tasks & Functions:**
- **Section 11—Disable Statements:**
- **Section 12—Hierarchical Structures:**
- **Section 13—Specify Blocks:**
- **Section 14—System Tasks & Functions:**
- **Section 16—Compiler Directives:**

Note: Support for each Verilog HDL construct is described with one of the following terms:

- **Supported**— The Quartus II software offers full support for the construct.
- **Not-supported**— The construct cannot be used in a Verilog Design File (.v). If used, the construct causes an error when the Quartus II software compiles the file.

Verilog Support —— ISE Foundation v8.1



Verilog Language Support

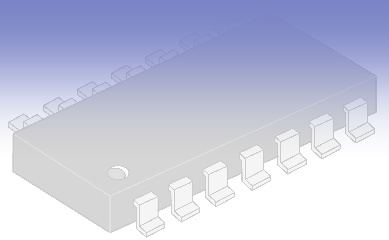
This chapter contains the following sections.

- "Introduction"
- "Behavioral Verilog Features"
- "Variable Part Selects"
- "Structural Verilog Features"
- "Parameters"
- "Parameter/Attribute Conflicts"
- "Verilog Limitations in XST"
- "Verilog Meta Comments"
- "Verilog Language Support Tables"
- "Primitives"
- "Verilog Reserved Keywords"
- "Verilog-2001 Support in XST"

For detailed information about Verilog design constraints and options, refer to Chapter 5, "Design Constraints". For information about the Verilog attribute syntax, see "Verilog Meta Comment Syntax" in Chapter 5.

For information on setting Verilog options in the Process window of Project Navigator, refer to "General Constraints" in Chapter 5.

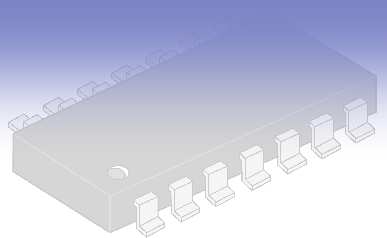
Verilog Support —— Synplify pro v9.6.2



Verilog Language Support

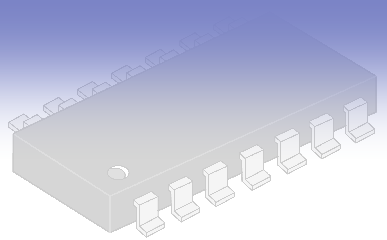
This chapter discusses Verilog support in the synthesis tool (SystemVerilog support is discussed in a separate chapter, *SystemVerilog Language Support*). This chapter includes the following topics:

可综合设计的基本原则



- 尽量减少使用可能导致前后仿真结果不一致的描述风格
  - 原因 —— 综合工具忽略所有由#<delay>制定的时序延时
  - 导致 —— 综合前后的Verilog的仿真结果可能不同
- 初始化
  - 逻辑综合工具不支持initial结构
  - 必须使用复位机制来取代initial结构，进行电路信号的初始化
- 应当明确地指定信号和变量宽度
  - 定义未指定宽度的变量可能产生庞大的门级网表
  - 因为综合工具可能根据变量定义生成不必要的逻辑。

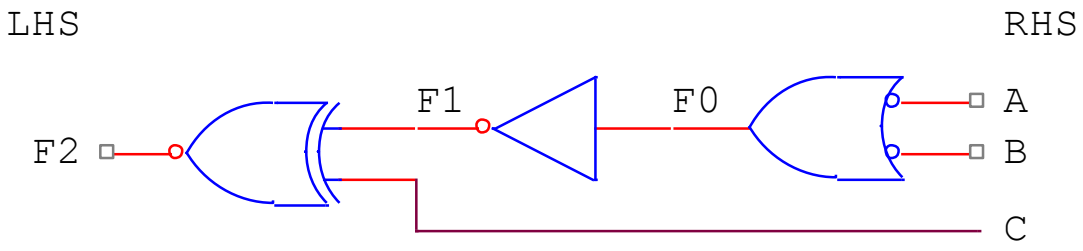
使用Verilog 操作符



- 与 x 和 z 相关的两种的操作符不能用于综合
  - ===
  - !==
  - 逻辑值 x 和 z 在逻辑综合中没有意义
- 其它操作符都可用于逻辑综合
- 表达式
  - 不要依赖操作符来确定逻辑运算的优先级
  - 应当使用圆括号明确指定逻辑运算的优先级
  - 例如：
    - `y = a & b | c`
    - 应当改写为：`y = ( a & b ) | c`

Verilog 模块设计原则 1

- 例、过程阻塞赋值
  - always @(event)
  - begin
  - …
  - F0 = ~(A & B);
  - F1 = ~F0;
  - F2 = F1 ^~ C;
  - …
  - end
- 对应的电路如下图所示：



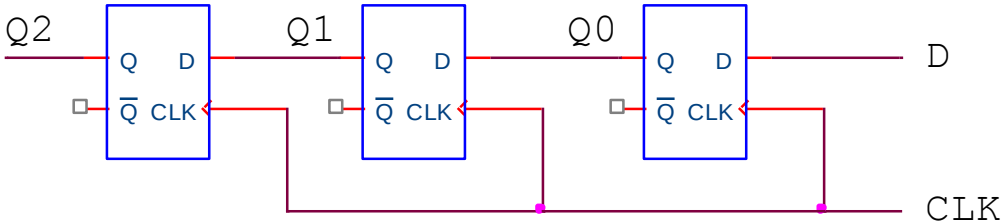
- 阻塞赋值操作
  - 用 always 语句块建立组合逻辑模型时，用阻塞赋值

Verilog 模块设计原则 2

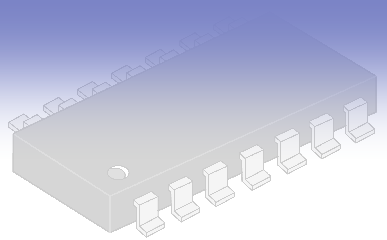
- 非阻塞赋值
  - 用 always 语句块建立时序电路模型时，用非阻塞赋值
- 例

```
always @(posedge clk)
    begin
        ...
        Q0 <= D;
        Q1 <= Q0;
        Q2 <= Q1;
        ...
    end
```

- 对应的电路如下图所示：



Verilog 模块设计原则(3~4)

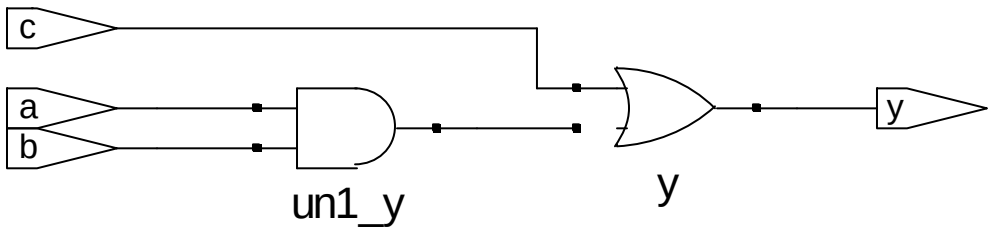


- 原则 3
  - 在同一个 always 过程语句块中，同时描述时序电路和组合电路时，均使用非阻塞赋值
    - 原则 3’
      - 在同一个 always 过程语句块中，不要同时使用阻塞赋值和非阻塞赋值
- 原则 4
  - 不要在多个 always 块中为同一个变量赋值

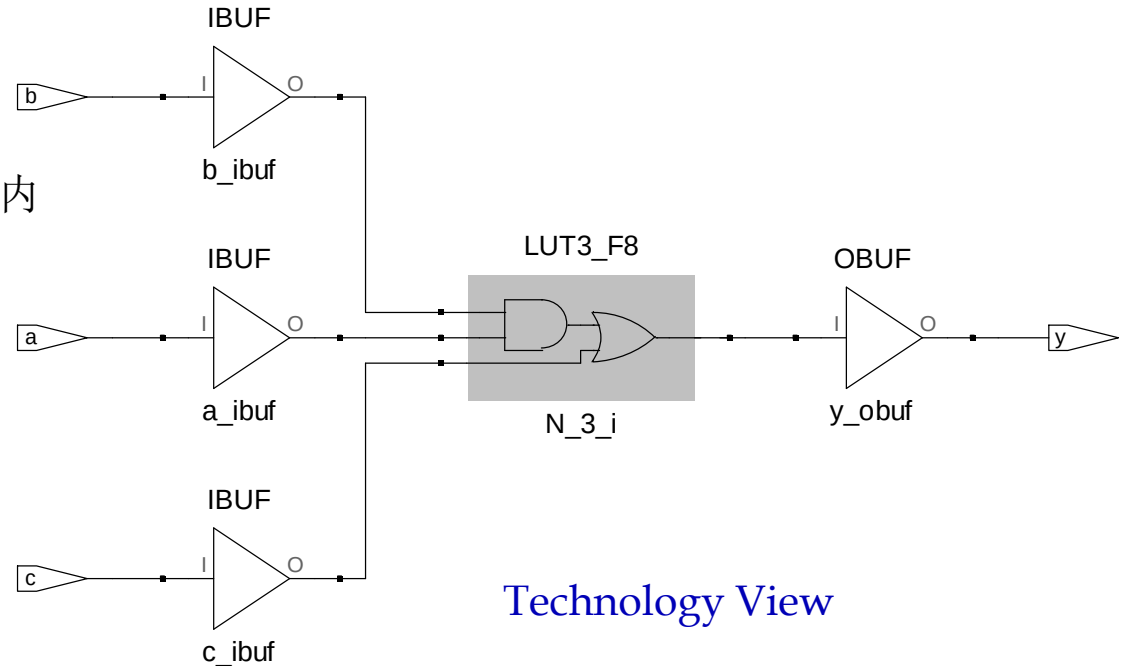
```
always @*
begin
    y = a + b;
end

always @*
begin
    y = c ^ d
end
```

Verilog 综合 —— 简单逻辑运算

- assign —— 描述组合逻辑
  - 综合工具：Synplify Proto 8.1
  - FPGA：
    - Xilinx Spartan2-XC2S100-6-PQ208
- RTL View
  - 将设计转换为基本部件的连接电路图
  - 基本部件
    - ALU
    - Register
    - 存储器等
- Techology View
  - 将RTL映射到具体的FPGA器件内部逻辑单元后的电路连接图



RTL View

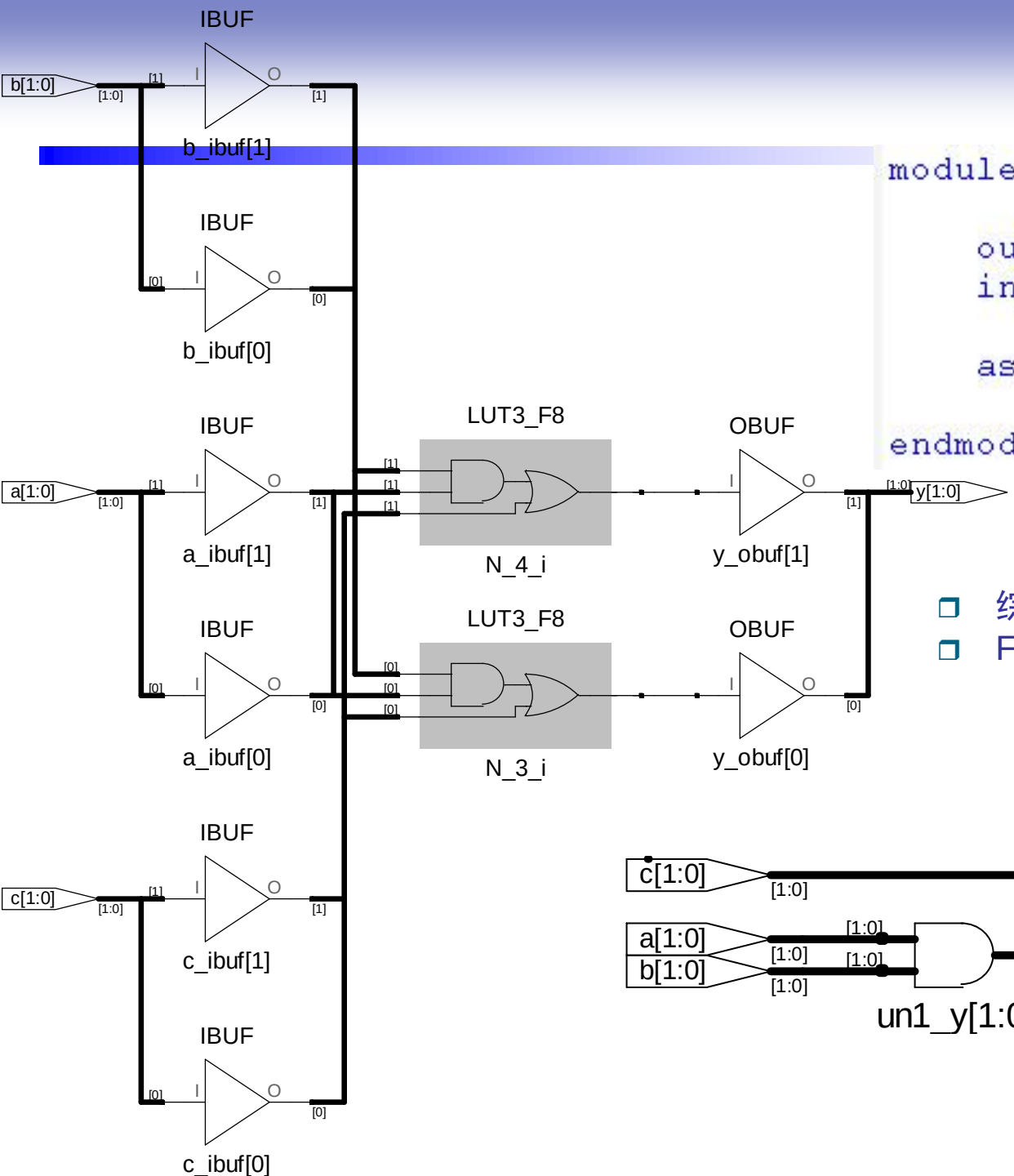


Technology View

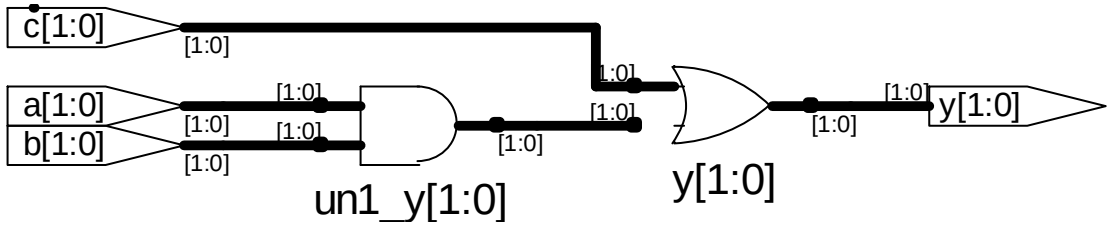
```
module comb01( y, a, b, c);
    output y;
    input a, b, c;

    assign y = ( a & b ) | c;

endmodule
```

向量的逻辑运算

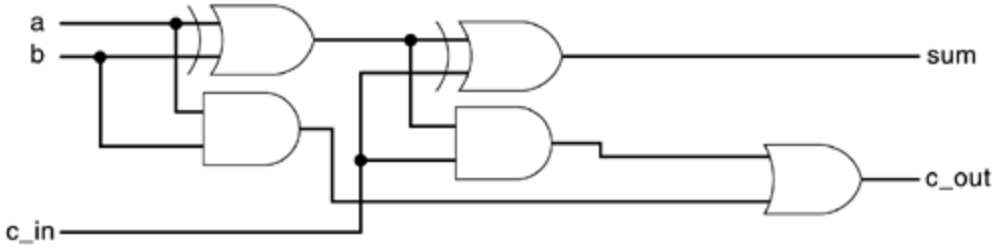
```
module comb02( y, a, b, c);

    output [1:0] y;
    input [1:0] a, b, c;

    assign y = ( a & b ) | c;

endmodule
```



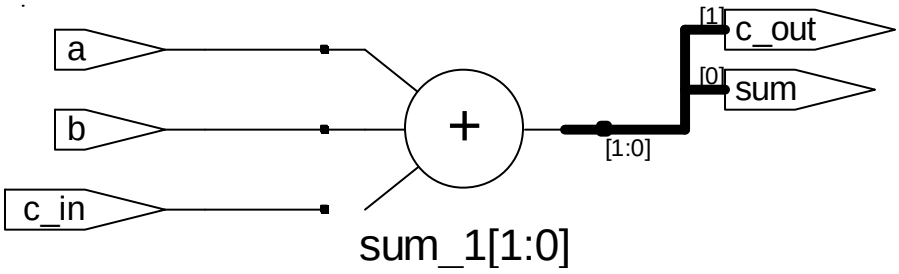
- 综合工具：Synplify Proto 8.1
- FPGA：
  - Xilinx Spartan2-XC2S100-6-PQ208

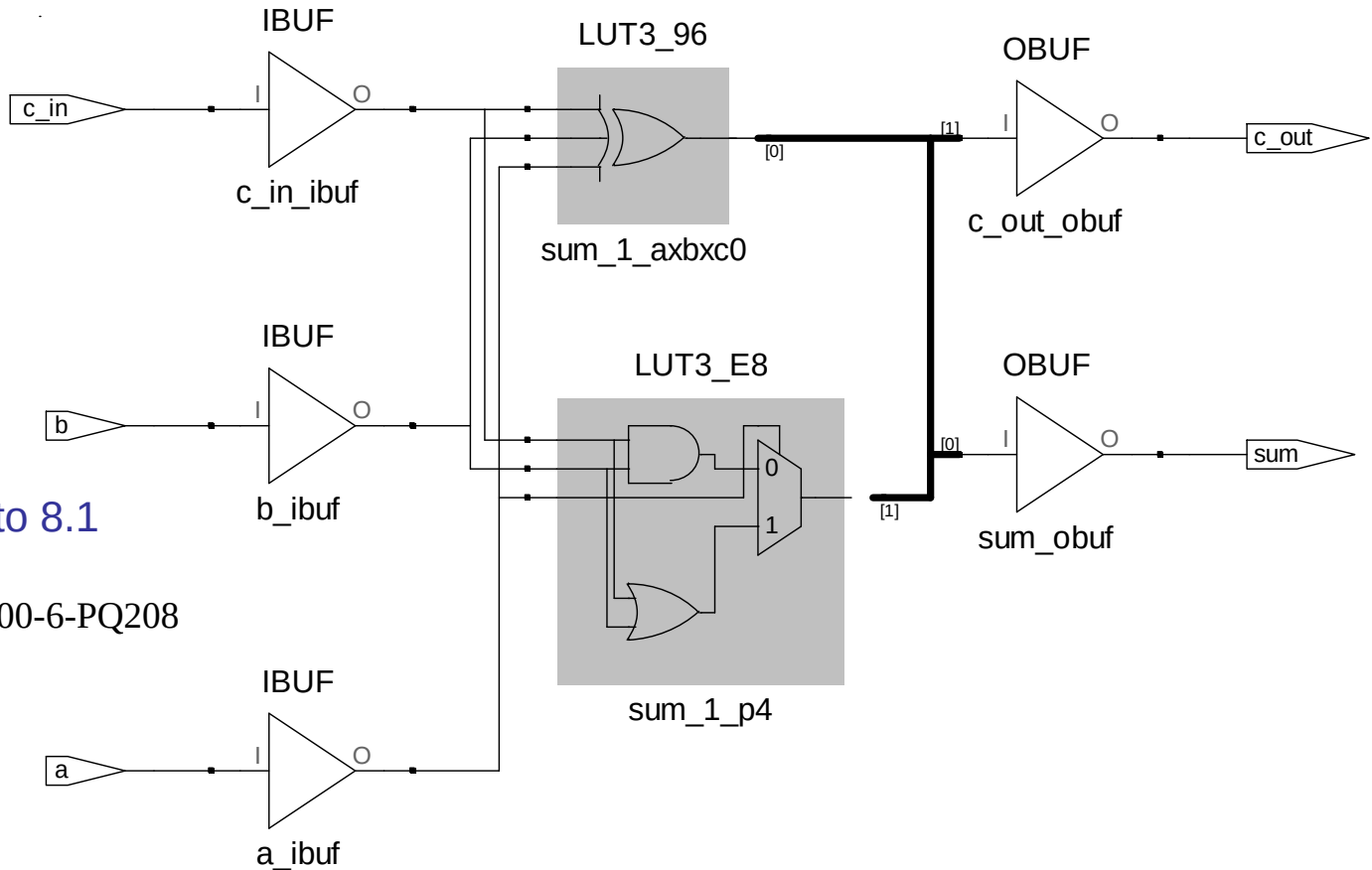


一位全加器（1）




```
module fulladder1b ( sum, c_out, a, b, c_in );

    output sum, c_out;
    input  a, b, c_in;

    assign { c_out, sum } =  a + b + c_in;

endmodule
```

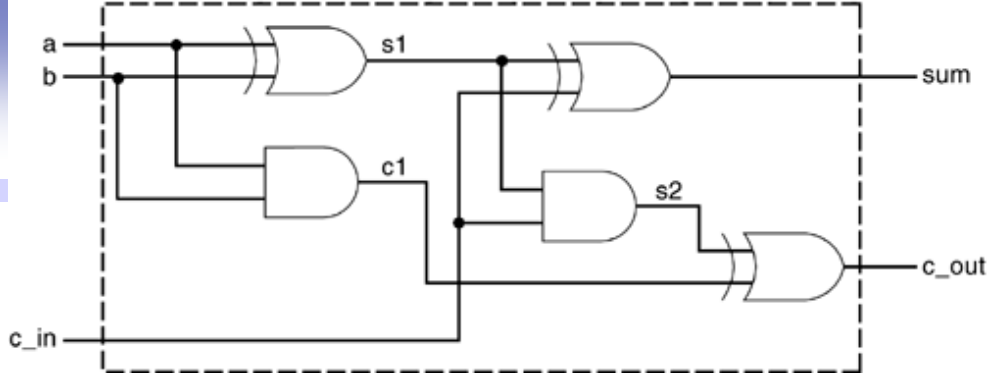


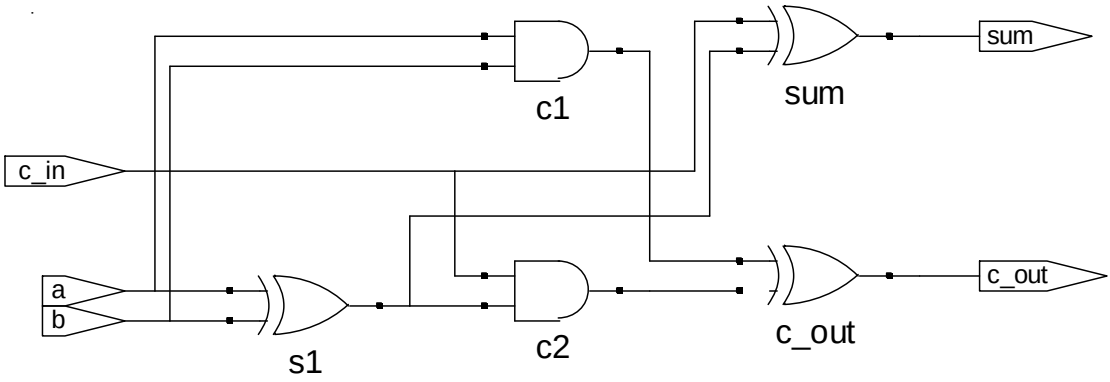


- 综合工具：Synplify Proto 8.1
- FPGA：
  - Xilinx Spartan2-XC2S100-6-PQ208

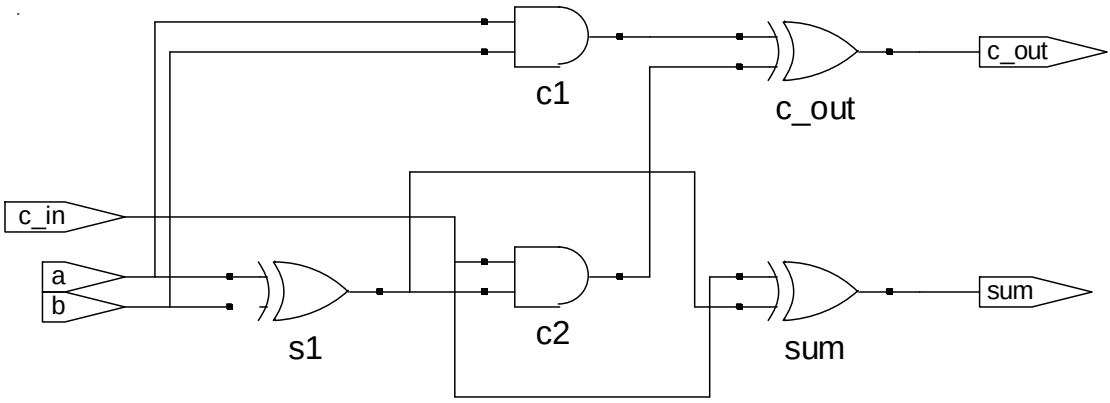
一位全加器（3）

```
// Define a 1-bit full adder
module fulladder1bg(sum, c_out, a, b, c_in);

    // I/O port declarations
    output sum, c_out;
    input a, b, c_in;

    // Internal nets
    wire s1, c1, c2;

    // Instantiate logic gate primitives
    xor u1(s1, a, b);
    and u2(c1, a, b);

    xor u3(sum, s1, c_in);
    and u4(c2, s1, c_in);

    xor u5(c_out, c2, c1);

endmodule
```





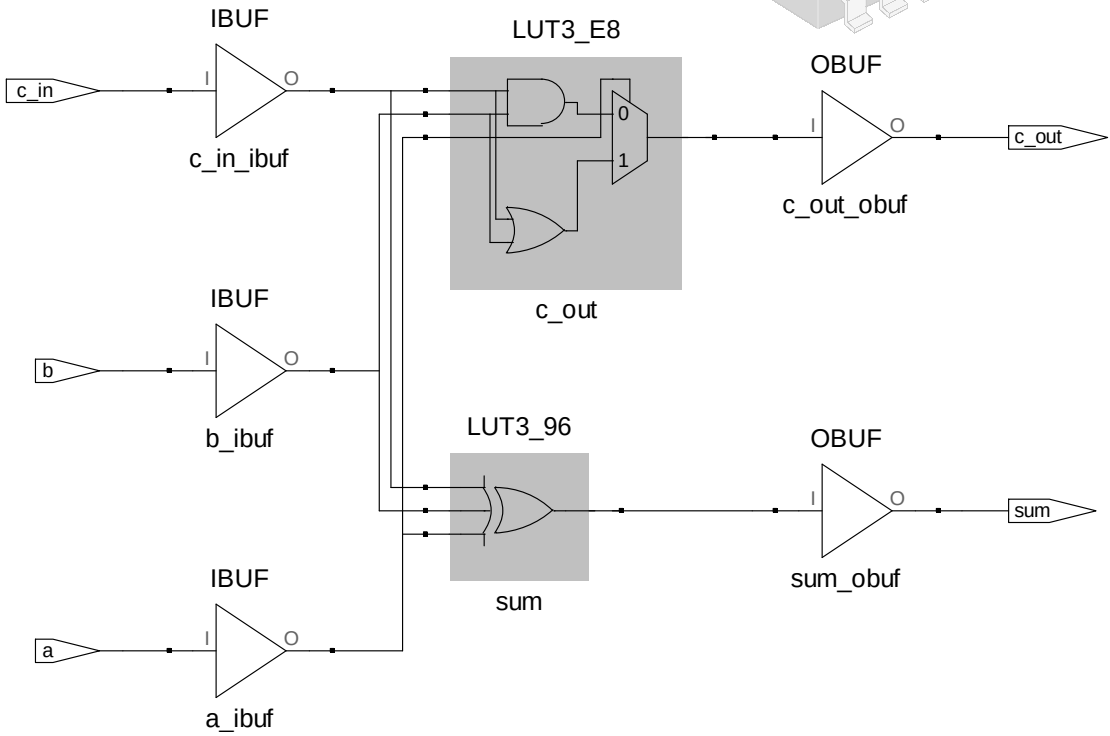
- 综合工具：Synplify Proto 8.1
- FPGA：
  - Altera MX7000-EPM7128B-7-TC144

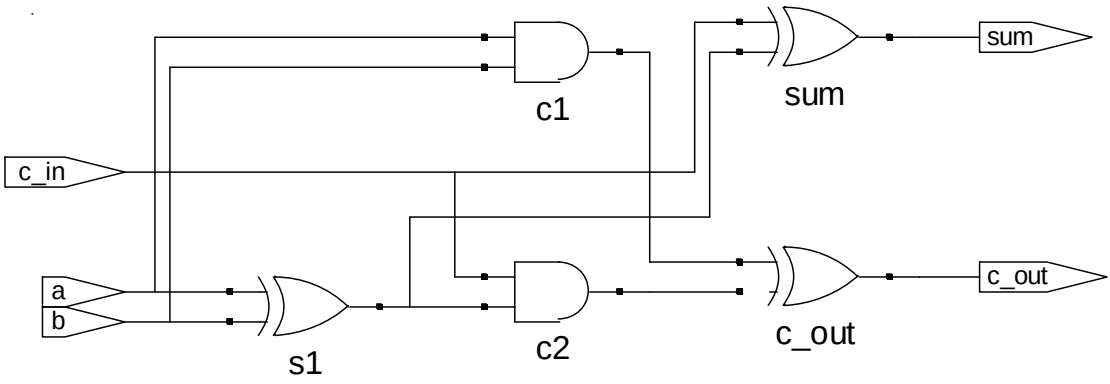


一位全加器（4）

- 综合工具：Synplify Proto 8.1
- FPGA：
  - Xilinx Spartan2-XC2S100-6-PQ208

```
// Define a 1-bit full adder
module fulladder1bg(sum, c_out, a, b, c_in);

    // I/O port declarations
    output sum, c_out;
    input a, b, c_in;

    // Internal nets
    wire s1, c1, c2;

    // Instantiate logic gate primitives
    xor u1(s1, a, b);
    and u2(c1, a, b);

    xor u3(sum, s1, c_in);
    and u4(c2, s1, c_in);

    xor u5(c_out, c2, c1);

endmodule
```

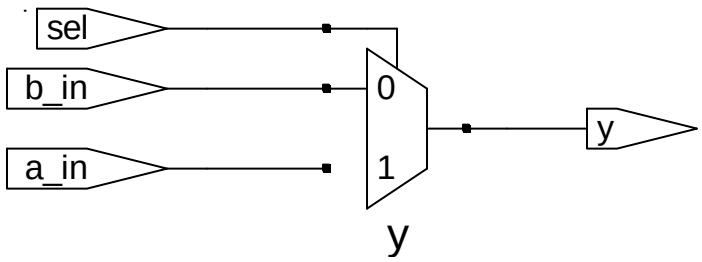


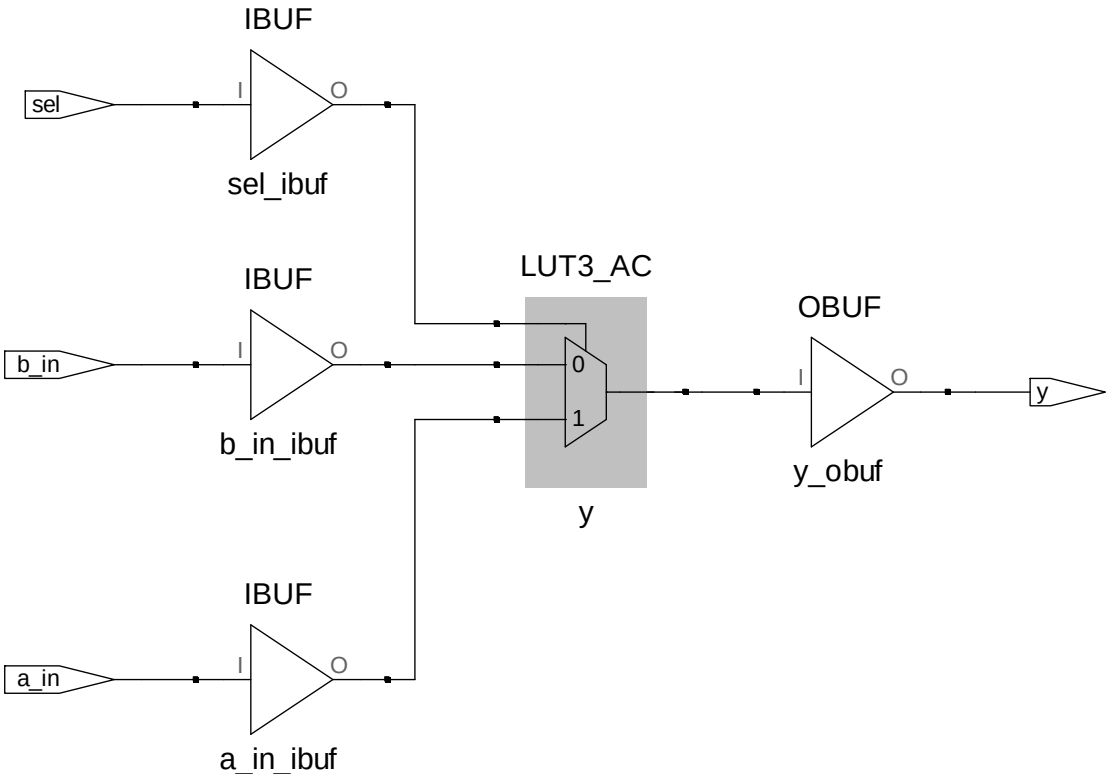


条件操作符

- 一般综合成多路选择器

```
module comb04( output y, input sel, input a_in, input b_in );

    assign y = ( sel ) ? a_in : b_in;

endmodule
```

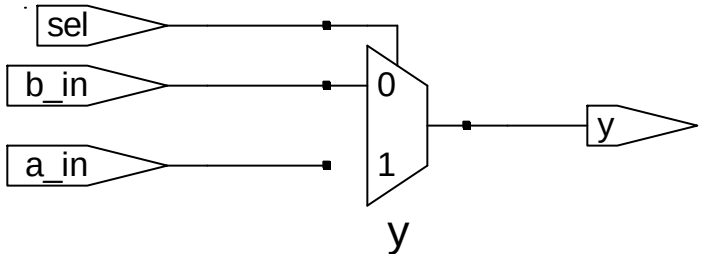




条件语句和多项选择语句

```
module comb05a( output reg y, input sel, input a_in, input b_in );

    always @(*)
    begin
        if (sel) y = a_in;
        else y = b_in;
    end

endmodule
```

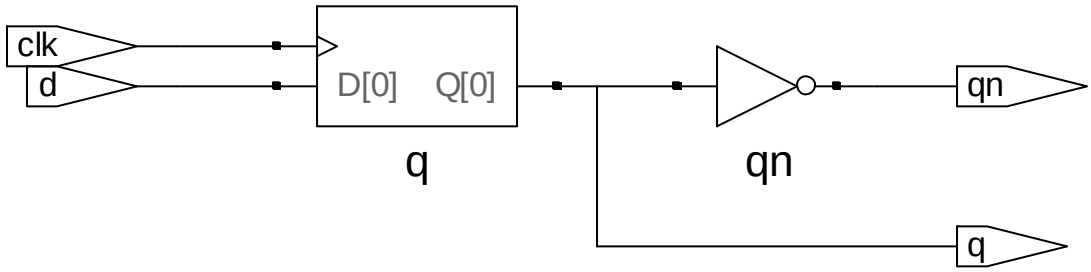
```
module comb05b( output reg y, input sel, input a_in, input b_in );

    always @(*)
    begin
        case (sel)
            1'b0:    y = a_in;
            1'b1:    y = b_in;
            default:y = a_in;
        endcase
    end

endmodule
```

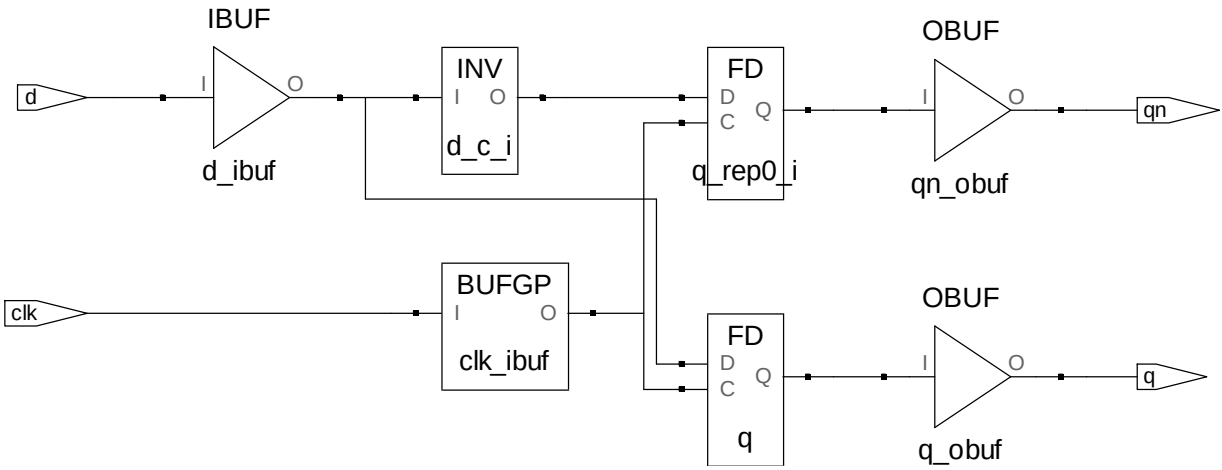


always 语句

- 组合逻辑 —— 电平控制，触发信号都列在敏感量列表中
- 时序逻辑 —— 时钟的变化控制：上升沿、下降沿
- 例、D触发器

```
module dff (output reg q,
            output qn,
            input  clk,
            input  d );

    always @(posedge clk) q <= d;

    assign qn = ~q;

endmodule
```

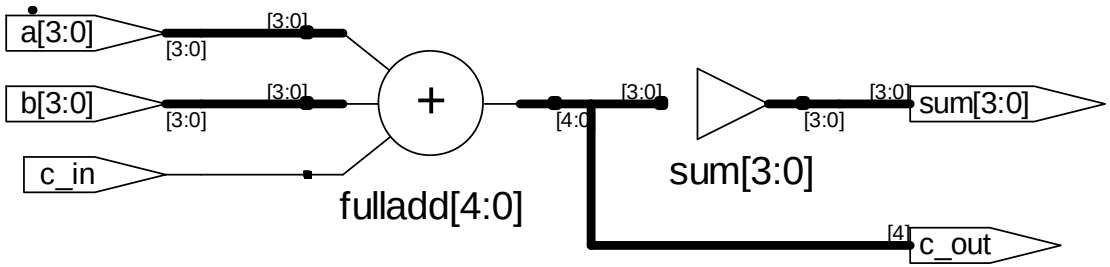




函数语句（1）

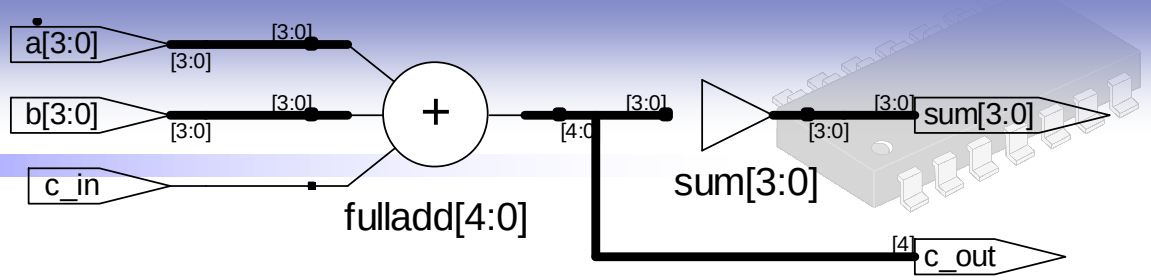
- 综合为具有一个输出变量（标量/矢量）的组合逻辑模块

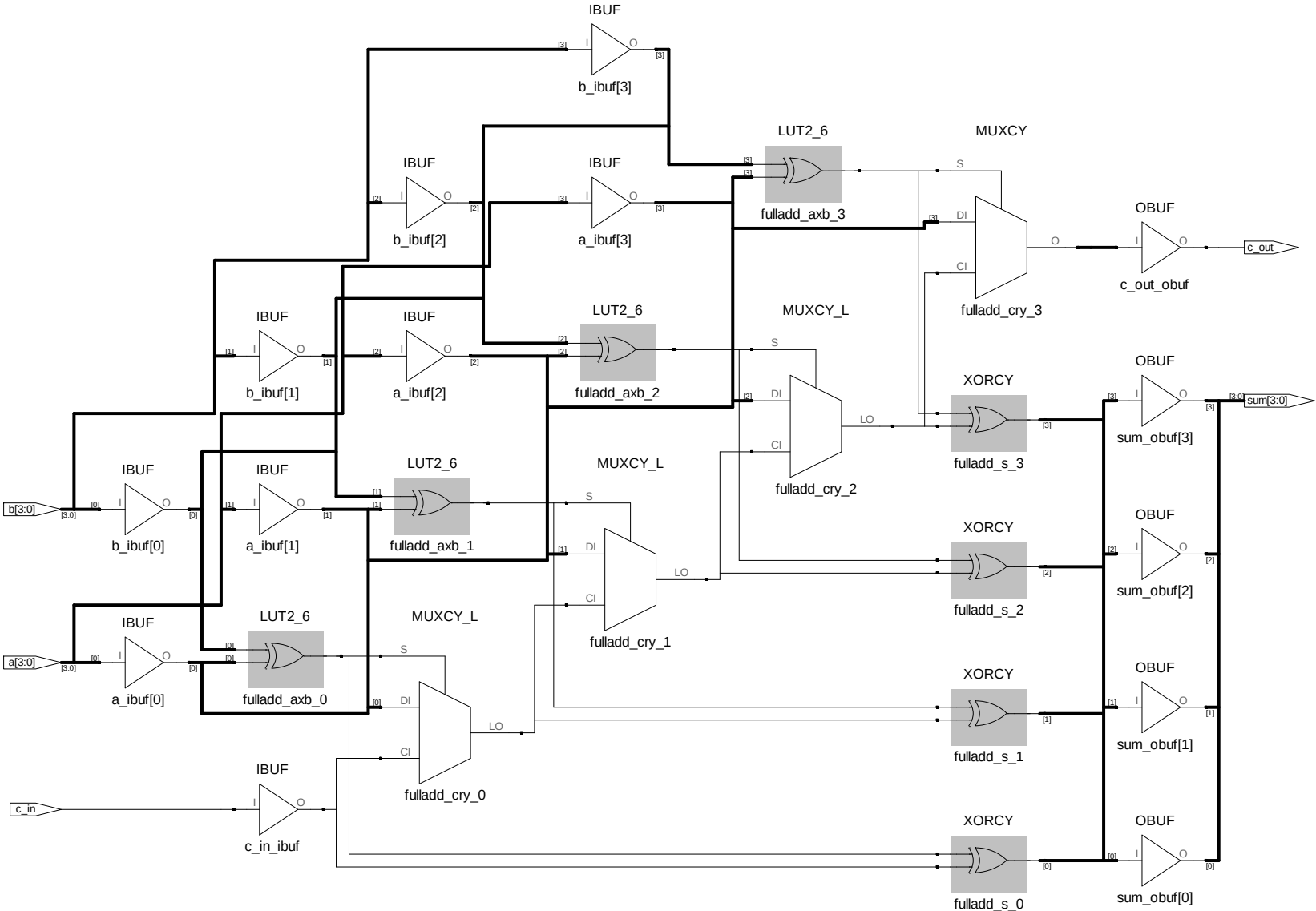
```
module fulladder4b ( sum, c_out, a, b, c_in );

    output reg [3:0] sum;
    output reg c_out;
    input  [3:0] a, b;
    input  c_in;


    always @(*)
    begin
        {c_out, sum } =  fulladd( a, b, c_in );
    end

    function [4:0] fulladd;
        input [3:0] p_a, p_b;
        input p_cin;
        begin
            fulladd = p_a + p_b + c_in;
        end
    endfunction

endmodule
```



函数语句（2）

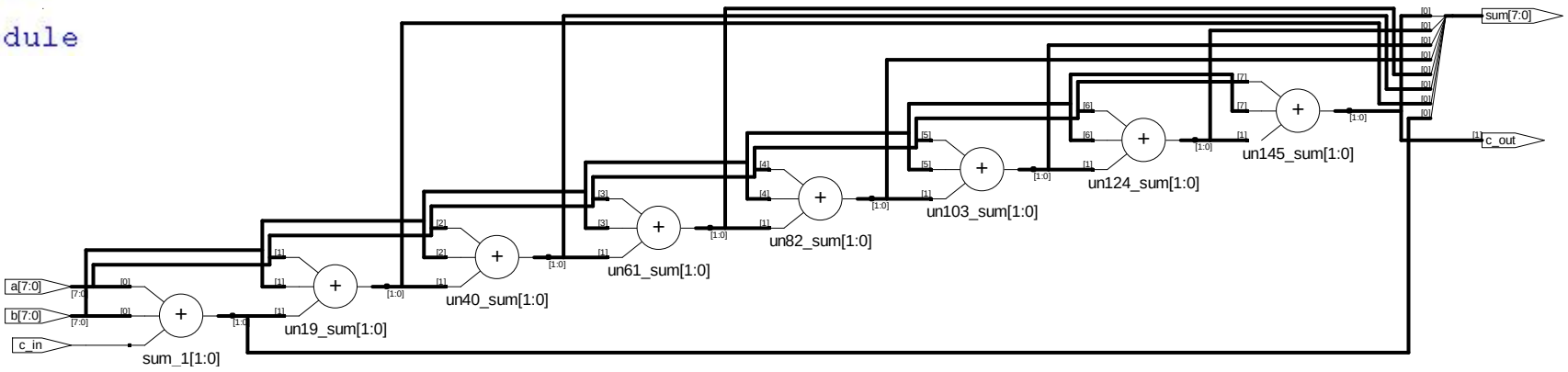
- 两种视图


a[3:0]
b[3:0]
c_in
[3:0]
+
[4:0]
fulladd[4:0]
sum[3:0]
c_out
[4]

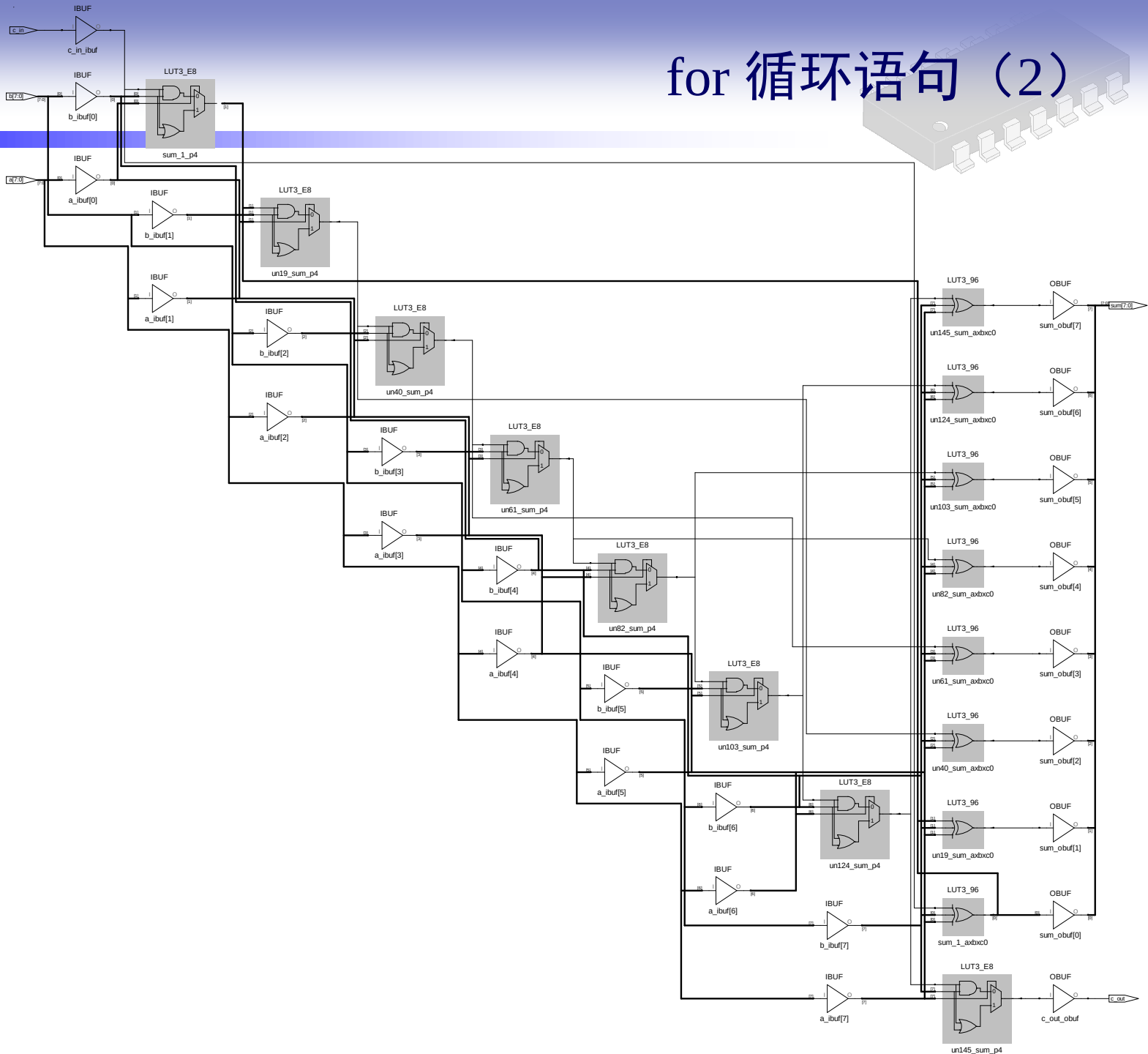

IBUF
b_ibuf[3]
b_ibuf[2]
a_ibuf[3]
LUT2_6
fulladd_axb_3
MUXCY
fulladd_cry_3
OBUF
c_out_obuf
c_out
b_ibuf[1]
a_ibuf[2]
fulladd_axb_2
MUXCY_L
fulladd_cry_2
XORCY
fulladd_s_3
sum_obuf[3]
sum[3:0]
b[3:0]
b_ibuf[0]
a_ibuf[1]
fulladd_axb_1
fulladd_cry_1
fulladd_s_2
sum_obuf[2]
a[3:0]
a_ibuf[0]
fulladd_axb_0
fulladd_cry_0
fulladd_s_1
sum_obuf[1]
c_in
c_in_ibuf
fulladd_s_0
sum_obuf[0]

for 循环语句（1）

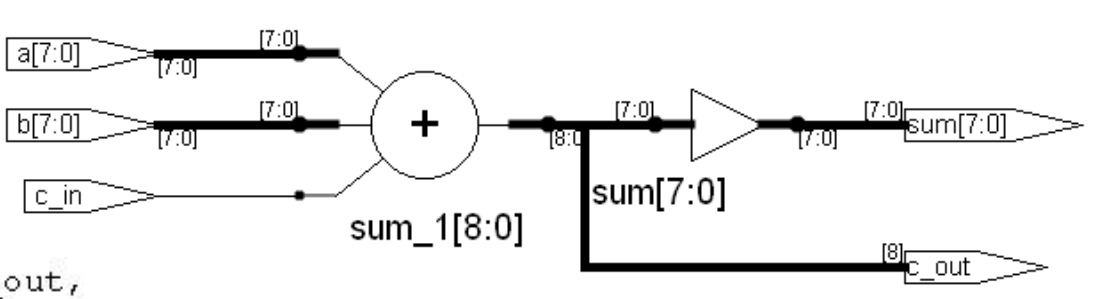
```
module fulladder8b ( c_out, sum, a, b, c_in );

    output reg c_out;
    output reg [7:0] sum;

    input  [7:0] a, b;
    input  c_in;
    reg c_buf;

    integer i;

    always @(*)
    begin
        c_buf = c_in;

        for ( i= 0; i <= 7; i = i + 1 )
            {c_buf, sum[i]} =  a[i] + b[i] + c_buf;

        c_out = c_buf;
    end

endmodule
```



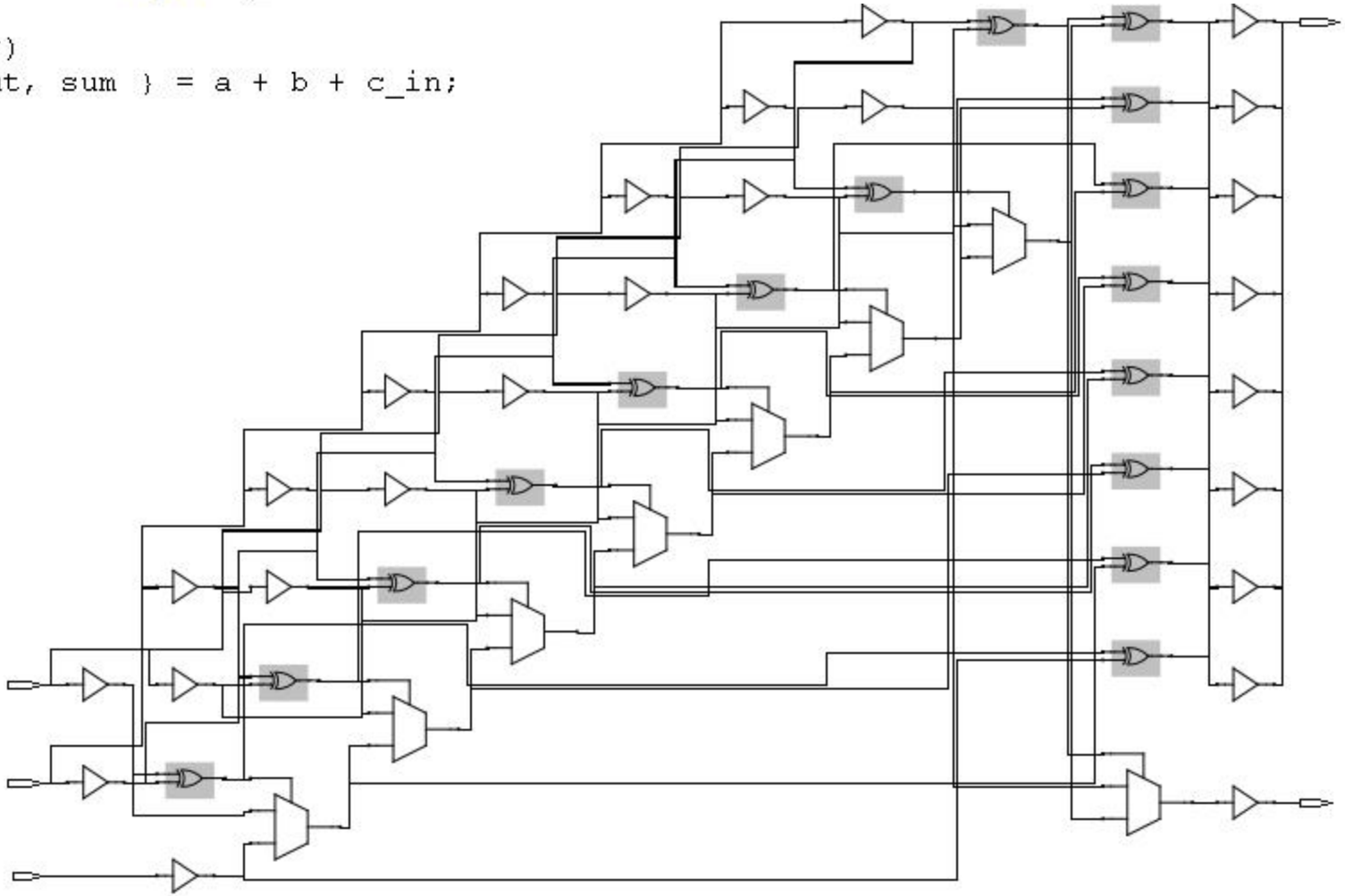
for 循环语句（2）


IBUF
c_in_ibuf
b[7:0]
b_ibuf[0]
LUT3_E8
sum_1_p4
a[7:0]
a_ibuf[0]
b_ibuf[1]
un19_sum_p4
a_ibuf[1]
b_ibuf[2]
un40_sum_p4
a_ibuf[2]
b_ibuf[3]
un61_sum_p4
a_ibuf[3]
b_ibuf[4]
un82_sum_p4
a_ibuf[4]
b_ibuf[5]
un103_sum_p4
a_ibuf[5]
b_ibuf[6]
un124_sum_p4
a_ibuf[6]
b_ibuf[7]
a_ibuf[7]
un145_sum_p4
LUT3_96
un145_sum_axbxc0
un124_sum_axbxc0
un103_sum_axbxc0
un82_sum_axbxc0
un61_sum_axbxc0
un40_sum_axbxc0
un19_sum_axbxc0
sum_1_axbxc0
OBUF
sum[7:0]
sum_obuf[7]
sum_obuf[6]
sum_obuf[5]
sum_obuf[4]
sum_obuf[3]
sum_obuf[2]
sum_obuf[1]
sum_obuf[0]
c_out
c_out_obuf

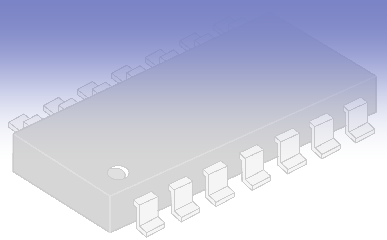
再次综合 8 位全加器




```
2  module fulladder8b( output reg c_out,
3                      output reg [7:0] sum,
4                      input [7:0] a, b,
5                      input c_in );
6
7      always @(*)
8          { c_out, sum } = a + b + c_in;
9
10 endmodule
```



Verilog编码风格

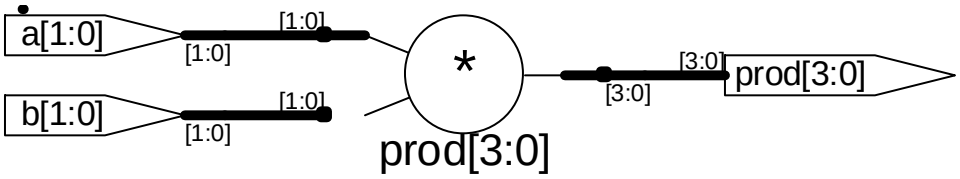


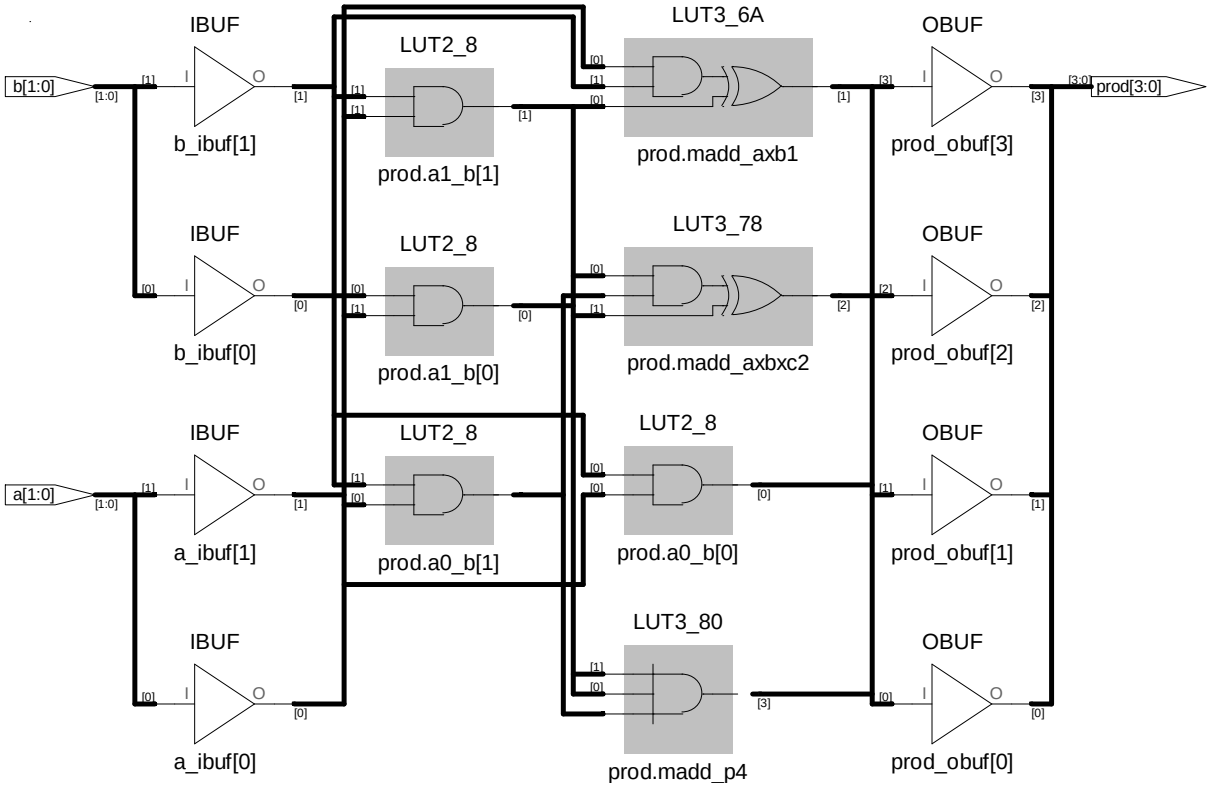
- 要点
  - 在抽象层次和控制逻辑综合结构之间进行折中
  - 更高的抽象层次
    - 易于设计，但可能产生不可预料的逻辑结构
  - 较低的抽象层次
    - 不利于设计，很可能依赖于具体的工艺和设计工具
- 基本原则
  - 使用有意义的信号和变量名称
  - 避免混合使用上升沿和下降沿触发的触发器
  - 使用圆括号优化逻辑结构
  - 在 if-else 和 case 语句中，说明各种可能的条件分支
  - 不要在多个 always 过程块中对同一个变量进行赋值
  - 将复杂的模块划分成较小的模块进行设计

使用操作符 *、/、%

- 实现依赖于所采用的设计方法

```
module multi2b ( prod, a, b );

    output reg [3:0] prod;

    input  [1:0] a, b;


    always @(*)
    begin
        prod =  a * b;
    end

endmodule
```

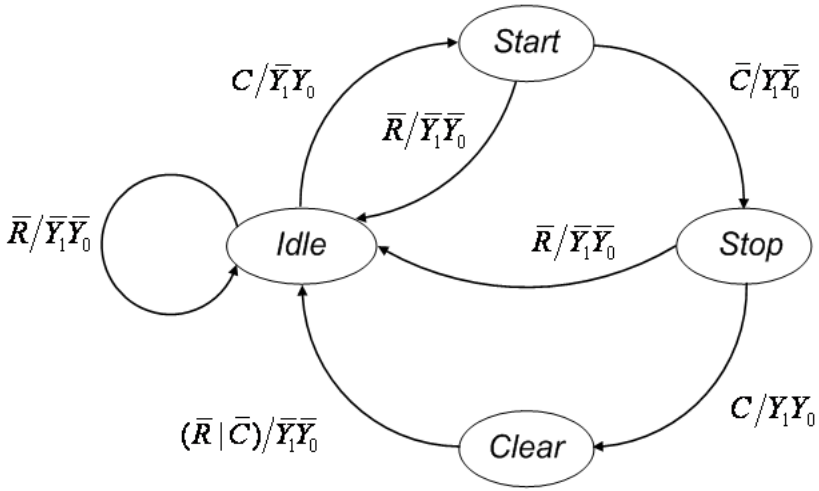




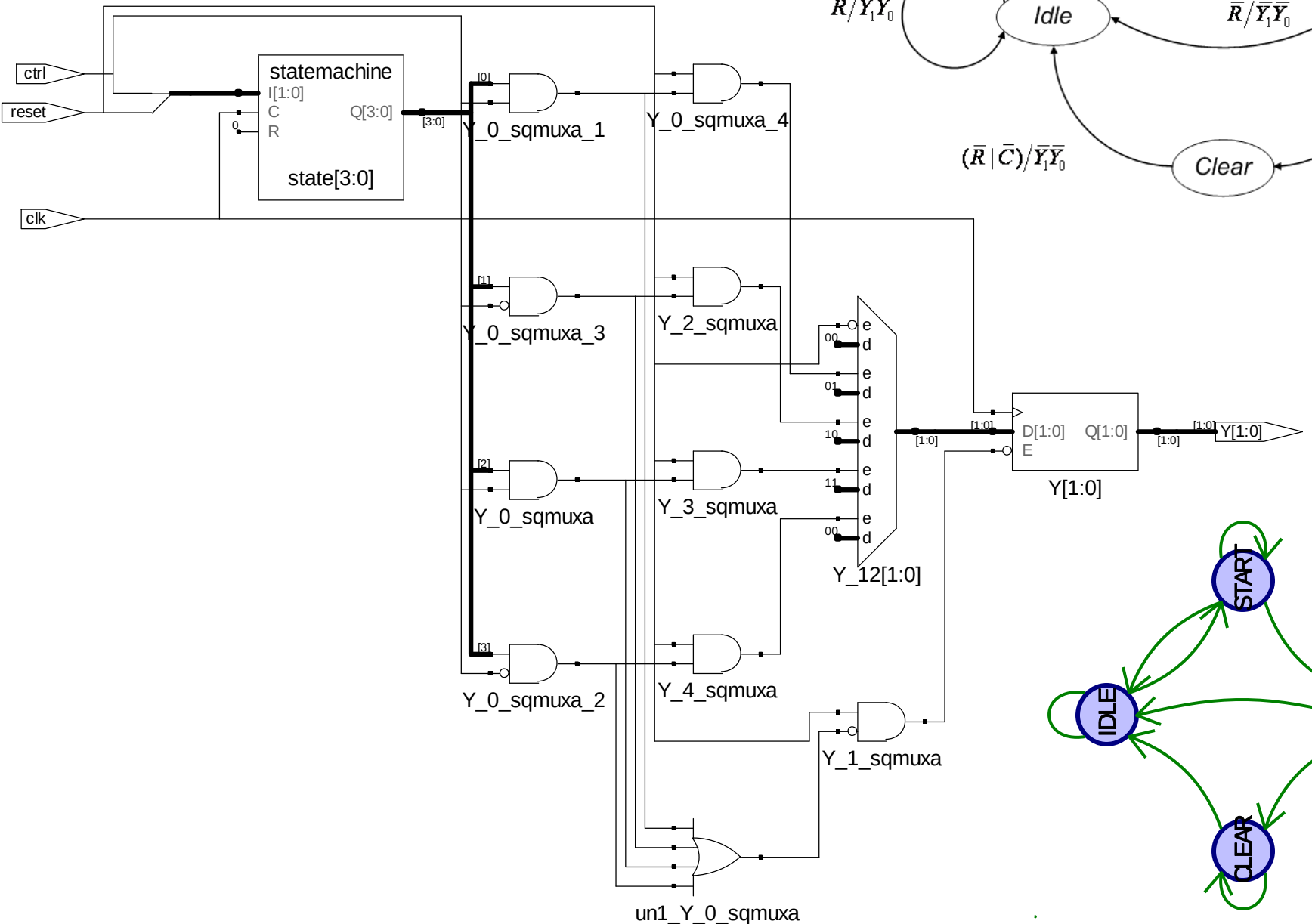
状态机综合

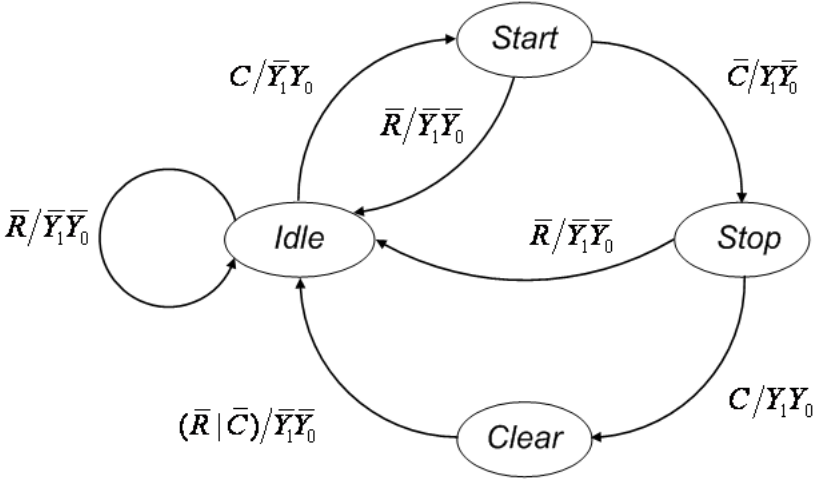
```
 3 module fsm_4s_m1 ( Y, clk, ctrl, reset );
 4     output reg [1 : 0] Y;
 5     input clk, ctrl, reset;
 6     reg [3 : 0] state;
 7     parameter  IDLE  = 4'b0001, START = 4'b0010, STOP  = 4'b0100, CLEAR = 4'b1000;
 8     always @(posedge clk)
 9         if (!reset) begin
10             state <= IDLE; Y <= 2'b00;
11         end
12         else
13             case (state)
14                 IDLE:
15                     begin
16                         if (ctrl) begin
17                             state <= START; Y <= 2'b01;
18                         end
19                         else
20                             state <= IDLE;
21                     end
22                 START:
23                     begin
24                         if (!ctrl) begin
25                             state <= STOP; Y <= 2'b10;
26                         end
27                         else state <= START;
28                     end
29                 STOP:
30                     begin
31                         if (ctrl) begin
32                             state <= CLEAR; Y = 2'b11;
33                         end
34                         else state <= STOP;
35                     end
36                 CLEAR:
37                     begin
38                         if (!ctrl) begin
39                             state <= IDLE; Y = 2'b00;
40                         end
41                         else state <= CLEAR;
42                     end
43                 default:
44                     state <= IDLE;
45             endcase
46 endmodule
```

- 输入
  - 复位 Reset：记为 R
  - 控制 Ctrl：记为 C
- 输出
  - Y_1、Y_0
- 使用同步复位



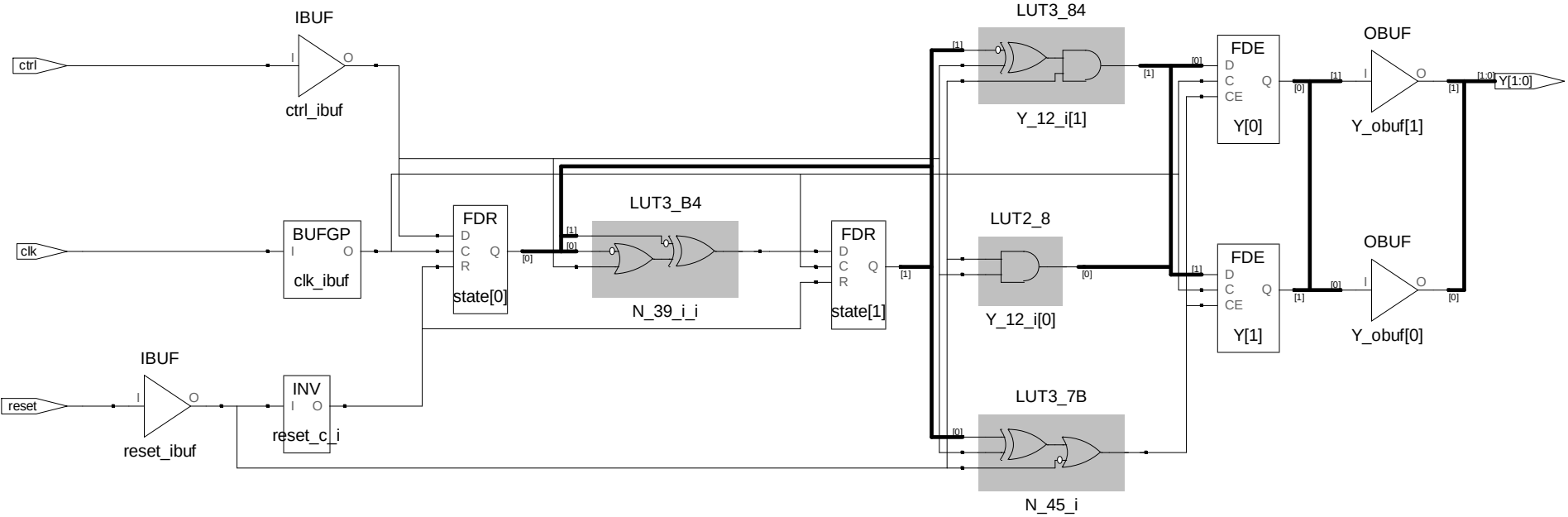
FSM RTL View


ctrl
reset
clk
statemachine
I[1:0]
C
R
Q[3:0]
state[3:0]
[3:0]
[0]
[1]
[2]
[3]
Y_0_sqmuxa_1
Y_0_sqmuxa_4
Y_0_sqmuxa_3
Y_2_sqmuxa
Y_0_sqmuxa
Y_3_sqmuxa
Y_0_sqmuxa_2
Y_4_sqmuxa
Y_1_sqmuxa
un1_Y_0_sqmuxa
e
d
00
01
10
11
Y_12[1:0]
D[1:0]
Q[1:0]
E
Y[1:0]


Start
Idle
Stop
Clear
$C/\bar{Y}_1\bar{Y}_0$
$\bar{C}/Y_1\bar{Y}_0$
$\bar{R}/\bar{Y}_1\bar{Y}_0$
$\bar{R}/\bar{Y}_1\bar{Y}_0$
$\bar{R}/\bar{Y}_1\bar{Y}_0$
$(\bar{R}\,|\,\bar{C})/\bar{Y}_1\bar{Y}_0$
C/Y_1Y_0

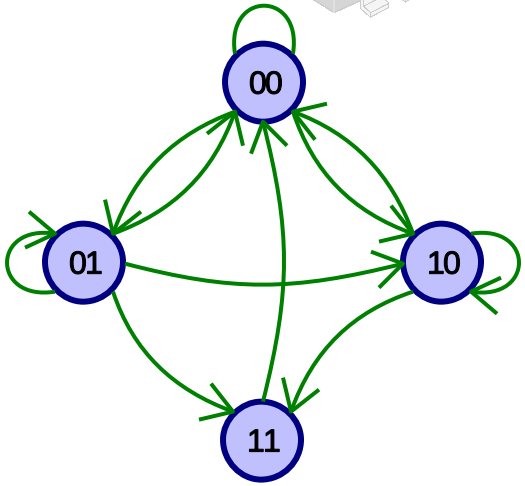
START
IDLE
STOP
CLEAR

FSM Technology View



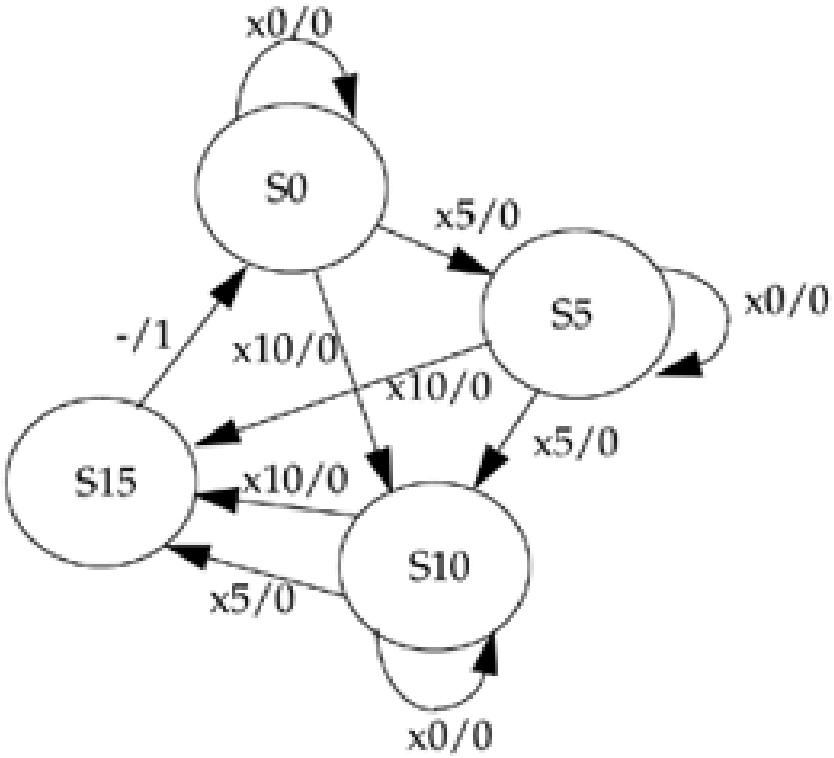
例、报纸售卖机投币器

- 功能：
  - 报纸价格：15分
  - 接受硬币：5分、1角
  - 不找零。



State	Money
S0	0 cents
S5	5 cents
S10	10 cents
S15	15 cents

Input	coin[1:0]
x0	2'b00
x5	2'b01
x10	2'b10
-	don't care

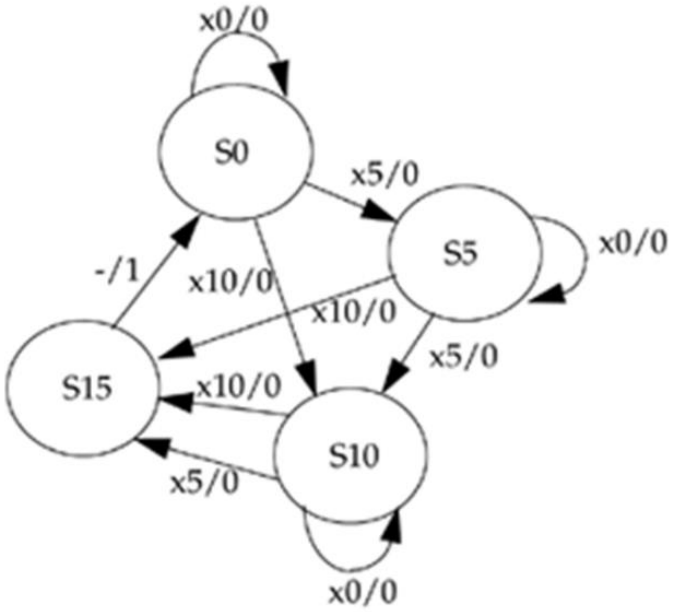


Verilog模块设计（1）

```
module vend(
             input [1:0] coin, //Input output port declarations
             input clock,
             input reset,
             output newspaper
           );
    //internal FSM state declarations
    wire [1:0] NEXT_STATE;
    reg [1:0] PRES_STATE;

    //state encodings
    parameter s0 = 2'b00,
              s5 = 2'b01,
              s10 = 2'b10,
              s15 = 2'b11;
```

State	Money
S0	0 cents
S5	5 cents
S10	10 cents
S15	15 cents

Input	coin[1:0]
x0	2'b00
x5	2'b01
x10	2'b10
-	don't care



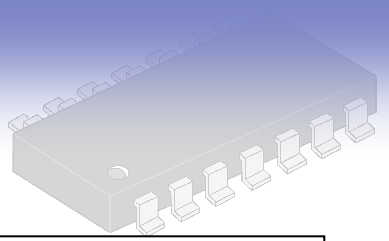
Verilog模块设计（2）

```
function [2:0] fsm (input [1:0] fsm_coin, fsm_PRES_STATE);
    reg fsm_newspaper;
    reg [1:0] fsm_NEXT_STATE;
begin
    case (fsm_PRES_STATE)
    s0: begin    //state = s0
            if (fsm_coin == 2'b10) begin
                fsm_newspaper = 1'b0;  fsm_NEXT_STATE = s10;
            end
            else if (fsm_coin == 2'b01) begin
                fsm_newspaper = 1'b0;  fsm_NEXT_STATE = s5;
            end
            else begin
                fsm_newspaper = 1'b0; fsm_NEXT_STATE = s0;
            end
        end
    s5: begin    //state = s5
            if (fsm_coin == 2'b10)  begin
                fsm_newspaper = 1'b0;  fsm_NEXT_STATE = s15;
            end
            else if (fsm_coin == 2'b01) begin
                fsm_newspaper = 1'b0;  fsm_NEXT_STATE = s10;
            end
            else begin
                fsm_newspaper = 1'b0;  fsm_NEXT_STATE = s5;
            end
        end
```

Verilog模块设计（3）

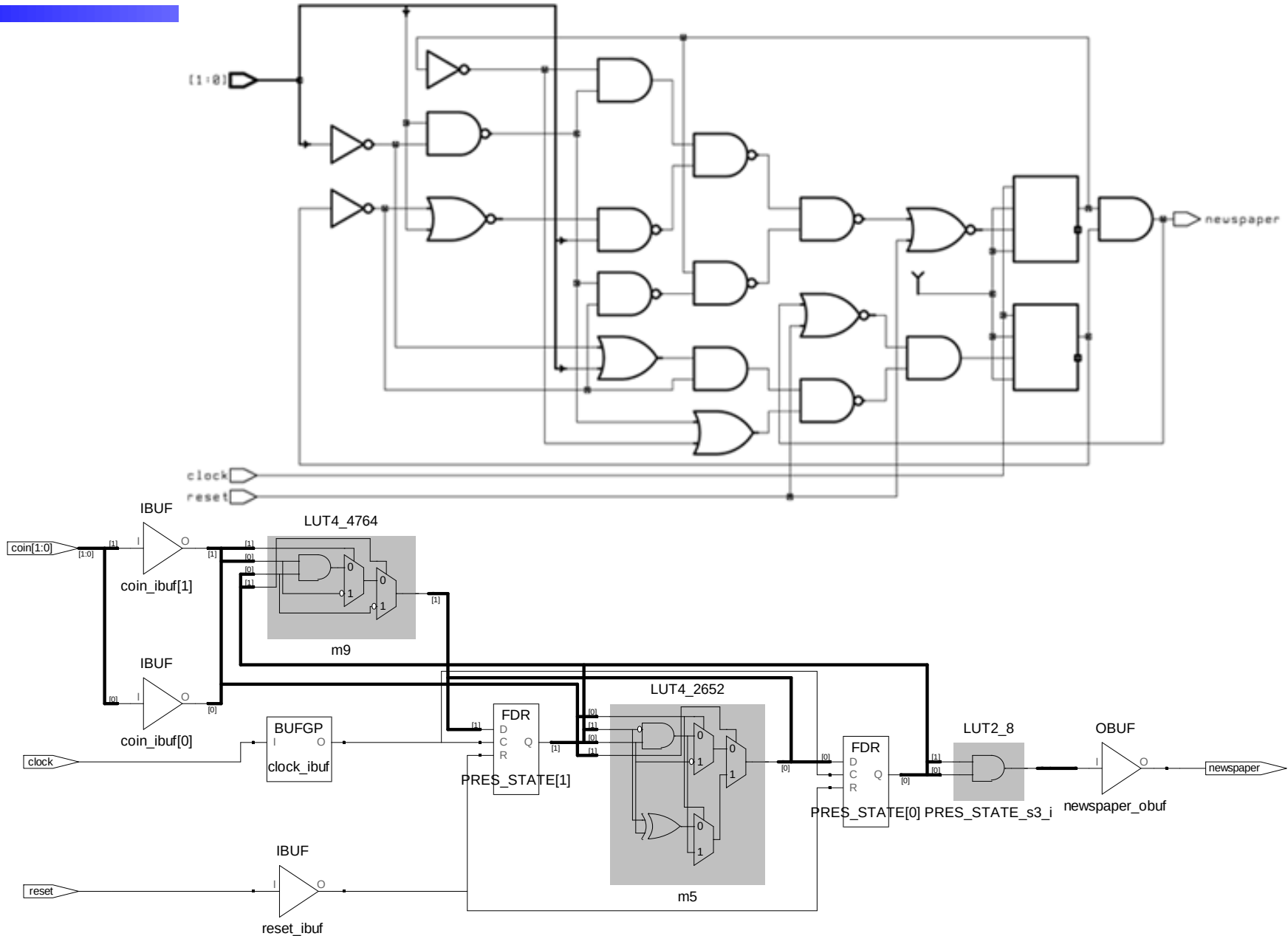
```
    s10:begin    //state = s10
                 if (fsm_coin == 2'b10) begin
                     fsm_newspaper = 1'b0;     fsm_NEXT_STATE = s15;
                 end
                 else if (fsm_coin == 2'b01) begin
                     fsm_newspaper = 1'b0;     sm_NEXT_STATE = s15;
                 end
                 else begin
                     fsm_newspaper = 1'b0;     fsm_NEXT_STATE = s10;
                 end
            end
        s15:begin    //state = s15
                 fsm_newspaper = 1'b1;     fsm_NEXT_STATE = s0;
            end
        endcase
        fsm = {fsm_newspaper, fsm_NEXT_STATE};
    end
    endfunction
```

Verilog模块设计（4）



```
    assign {newspaper, NEXT_STATE} = fsm(coin, PRES_STATE);

    //clock the state flip-flops.
    //use synchronous reset
    always @(posedge clock)
    begin
        if (reset == 1'b1) PRES_STATE <=  s0;
        else PRES_STATE <=  NEXT_STATE;
    end
endmodule
```

投币器电路的综合


[1:0]
newspaper
clock
reset
IBUF
coin[1:0]
coin_ibuf[1]
LUT4_4764
m9
IBUF
coin_ibuf[0]
BUFGP
clock
clock_ibuf
FDR
PRES_STATE[1]
LUT4_2652
m5
FDR
PRES_STATE[0]
LUT2_8
PRES_STATE_s3_i
OBUF
newspaper_obuf
newspaper
IBUF
reset
reset_ibuf