

System Verilog入门（一）

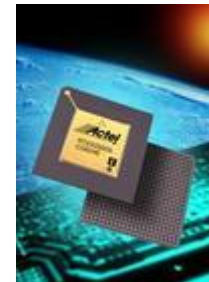
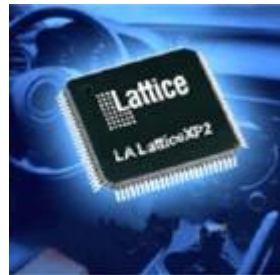
实验目的

- 了解 FPGA
- 学习 system verilog。
- 尝试使用 vivado 进行可编程器件开发。

可编程器件简介

- 概述

- PLD是电子设计领域中最具活力和发展前途的一项技术。
- PLD能做什么呢？可以毫不夸张的讲，PLD能完成任何数字器件的功能，上至高性能CPU,下至简单的位片电路，都可以用PLD来实现。
- 目前有多家公司生产CPLD/FPGA，主要有：ALTERA(Intel)，XILINX，Lattice，Actel。



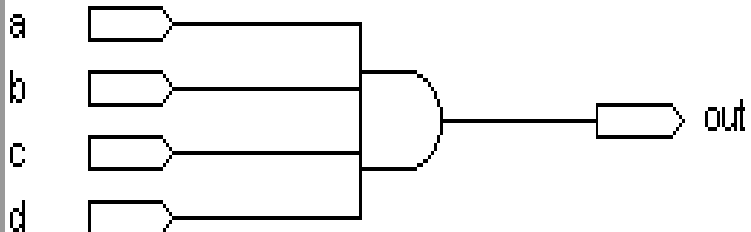
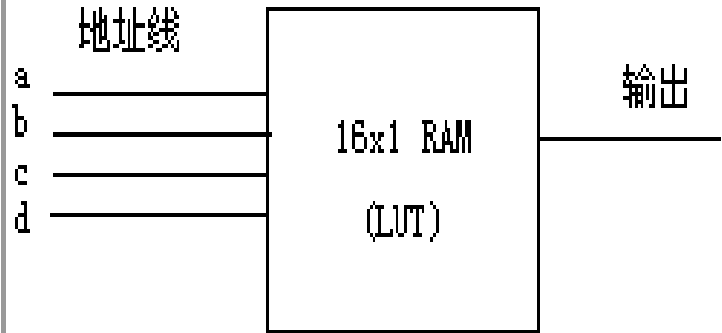
可编程器件

- **FPGA**
 - Field Programmable Gate Array 现场可编程门阵列
 - FPGA基于SRAM的架构，集成度高，以LE（包括查找表、触发器及其他）为基本单元，有内嵌Memory、DSP等，支持IO标准丰富。

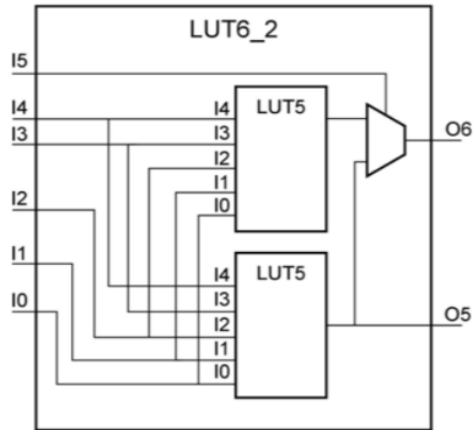
查找表

- 基于查找表（Look-Up-Table)的原理与结构：
 - 采用这种结构的PLD芯片如altera的ACEX,APEX系列, xilinx的Spartan,Virtex系列等。
 - 查找表（Look-Up-Table)简称为LUT，LUT本质上就是一个RAM
 - 工作原理
 - 当用户通过原理图或HDL语言描述逻辑电路
 - 软件会自动计算逻辑电路的所有可能的结果，并把结果事先写入RAM
 - 每输入一个信号进行逻辑运算就等于输入一个地址进行查表，找出地址对应的内容，然后输出即可。

4输入与门

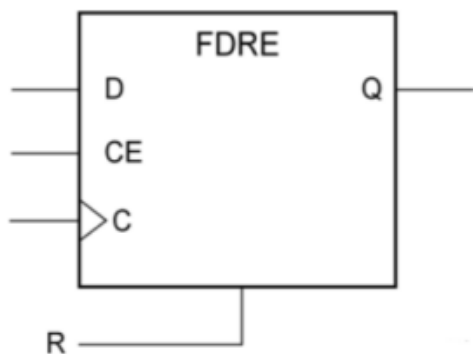
实际逻辑电路		LUT的实现方式	
			
a, b, c, d 输入	逻辑输出	地址	RAM中存储的内容
0000	0	0000	0
0001	0	0001	0
....	0	...	0
1111	1	1111	1

LUT结构



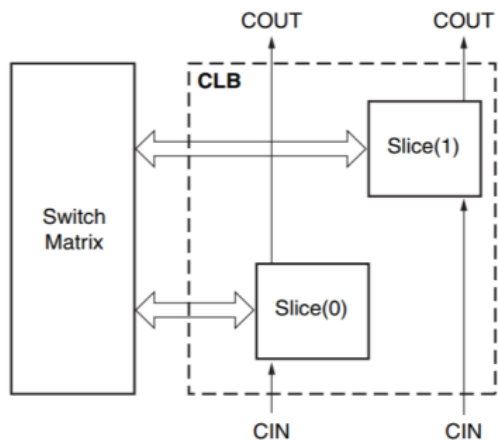
- 5输入，2输出
- 6输入，1输出

FF结构（寄存器）

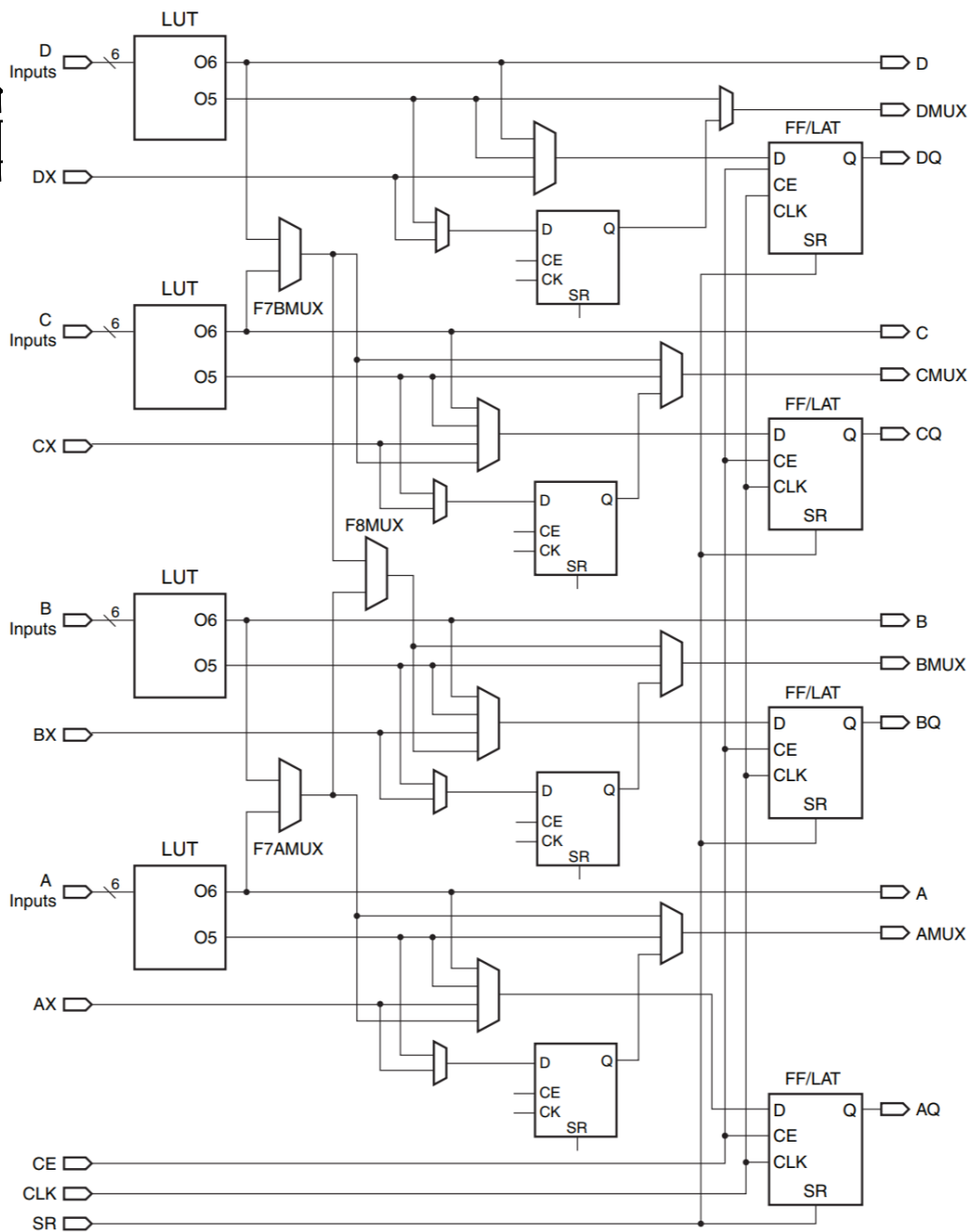


- D是数据输入
- CE是使能信号
- C是时钟信号
- Q是输出
- R是复位信号

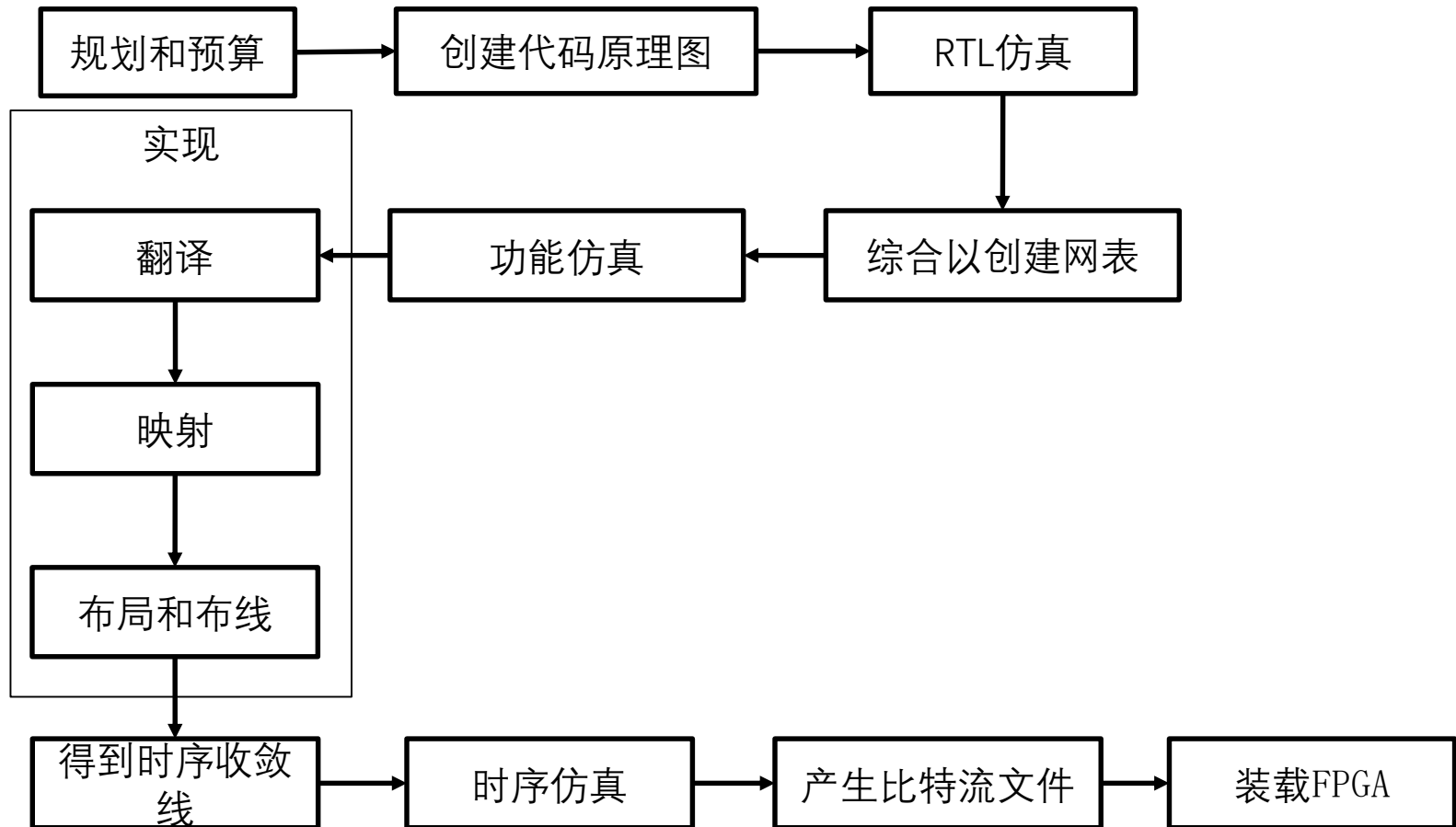
逻辑



每两个Slice被组织成一个可配置逻辑块（CLB），最后大量的CLB之间再由开关阵列连接起来。这样的架构使得FPGA片上的LUT、FF可以自由地连接，形成一个更大规模、可配置的逻辑电路



开发流程



System Verilog

- 硬件描述语言
- Verilog + c++
- 具有很强的电路描述与建模能力，能从多个层次对数字系统进行描述和建模。

为什么需要硬件描述语言？

- 硬件极其复杂，不可能直接用门电路搭接出最终的硬件电路，通过硬件描述语言进行抽象，使得对硬件的描述成为了可能
 - 晶体管（门级），连线
- 硬件描述语言
 - 可用于描述复杂的硬件设计
 - 可用于行为仿真（包括功能和时序）
 - 可用于硬件的综合
- 硬件描述语言被设计出来完成上述的目的
 - 不同的硬件描述语言有很多的相似性（VHDL, Verilog），完成相同的目标
 - 语言功能通常可以直接映射，特别是对于常用的子集
 - Verilog（SystemVerilog）在工业界常用

System Verilog

- 区分大小写
- 用 `//` 进行单行注释,用 `/*` 与 `*/` 进行跨行注释
- 标识符 (**identifier**) 可以是任意一组字母、数字、**\$** 符号和 `_`(下划线)符号的合, 但标识符的第一个字符必须是字母或者下划线, 不能以数字或者**\$** 符开始

System Verilog

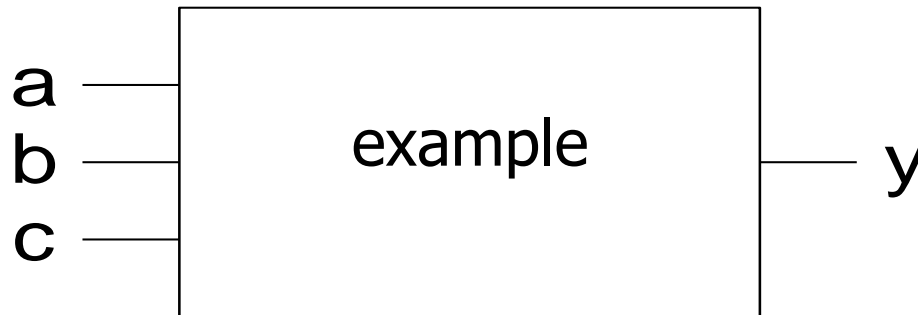
- 四种基本的值来表示硬件电路中的电平逻辑：
 - 0: 逻辑 0 或 "假"
 - 1: 逻辑 1 或 "真"
 - x 或 X: 未知
 - z 或 Z: 高阻
- 整数数值表示方法
 - 十进制('d 或 'D), 十六进制('h 或 'H), 二进制 ('b 或 'B), 八进制 ('o 或 'O)
 - 4'b1011 *// 4bit 数值*
 - 32'h3022_c0de *// 32bit 的数值*

数据类型

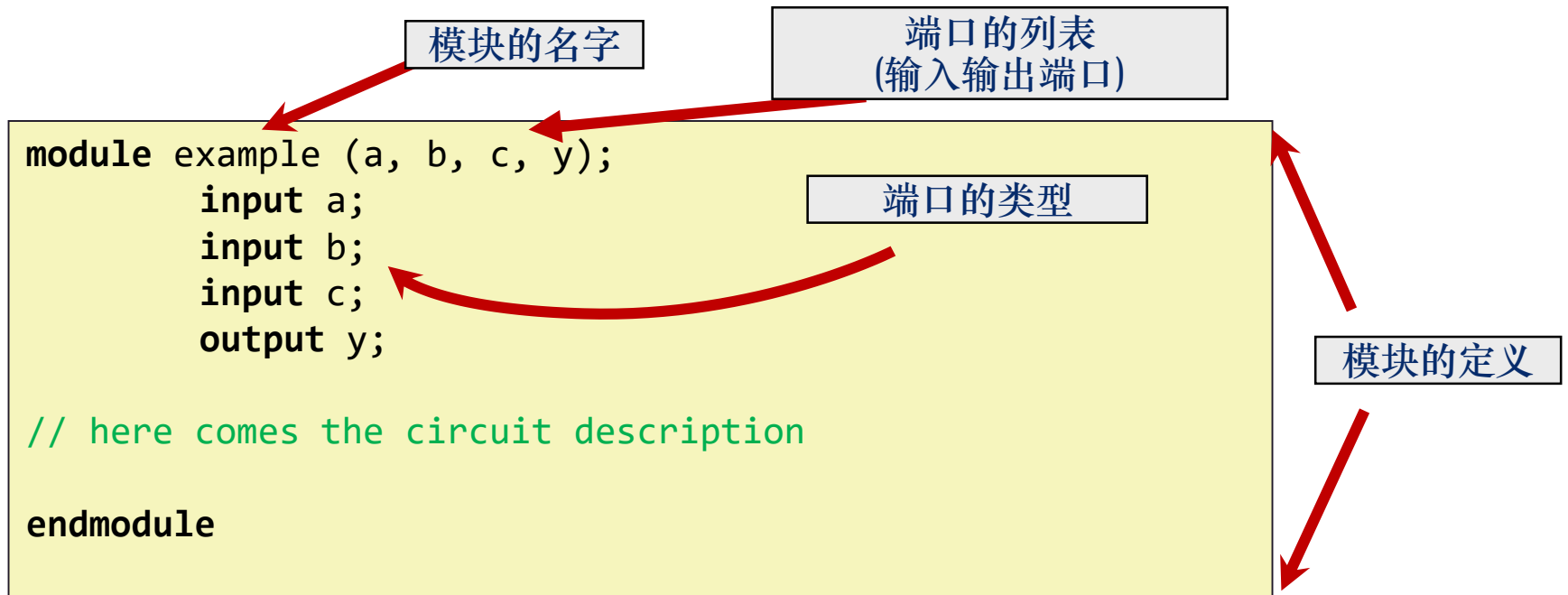
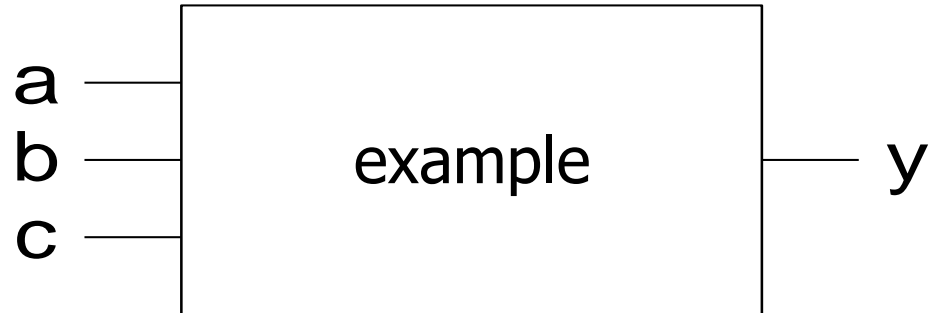
- **wire**
 - 表示硬件单元之间的物理连线，由其连接的器件输出端连续驱动
- **reg**
 - 用来表示存储单元，它会保持数据原有的值，直到被改写
- **logic**

模块

- 模块module是verilog(systemverilog)中的基本编程单元
- 模块的定义：
 - 模块的名字
 - 模块端口的名字
 - 模块端口的方向（输入端口还是输出端口）
- 描述模块的功能



模块



端口接口定义

```
module test ( a, b, y );  
    input a;  
    input b;  
    output y;  
  
endmodule
```

```
module test ( input a,  
              input b,  
              output y );  
  
endmodule
```

端口名字和方向可以放在一起，还可以加上类型，如
input wire a

向量

- 多位的输入输出Input/Output (Bus)

```
input  [31:0] a;    // a[31], a[30] .. a[0]
output [15:8] b1;   // b1[15], b1[14] .. b1[8]
output [7:0]  b2;   // b2[7], b2[6] .. b2[0]
input                c;    // single signal
```

- 通常使用 [31:0] 比 [0:31] 普遍
- 无论如何定义，要保持在程序中是一致的

连续赋值

- 用于对 **wire** 型变量进行赋值
- **assign LHS_target = RHS_expression ;**
 - **LHS_target** 必须是一个标量或者线型向量，而不能是寄存器类型。
 - **RHS_expression** 的类型没有要求，可以是标量或线型或寄存器向量，也可以是函数调用。
 - 只要 **RHS_expression** 表达式的操作数有事件发生（值的变化）时，**RHS_expression** 就会立刻重新计算，同时赋值给 **LHS_target**。

```
module and_gate (  
    input wire a_i,  
    input wire b_i,  
    output wire c_o  
);  
  
    assign c_o = a_i & b_i;  
endmodule
```

RTL ANALYSIS

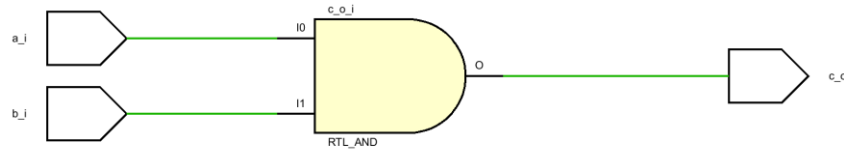
Open Elaborated Design

Report Methodology

[Report DRC](#)

Report Noise

[Schematic](#)



Cell Properties

c_o_i

Name: c_o_i
Reference name: RTL_AND
Type: RTL Gate
Number of cell pins: 3
Number of nets: 3

General Properties Nets Cell Pins

SYNTHESIS

Run Synthesis

Open Synthesized Design

Constraints Wizard

Edit Timing Constraints

[Set Up Debug](#)

[Report Timing Summary](#)

Report Clock Networks

Report Clock Interaction

[Report Methodology](#)

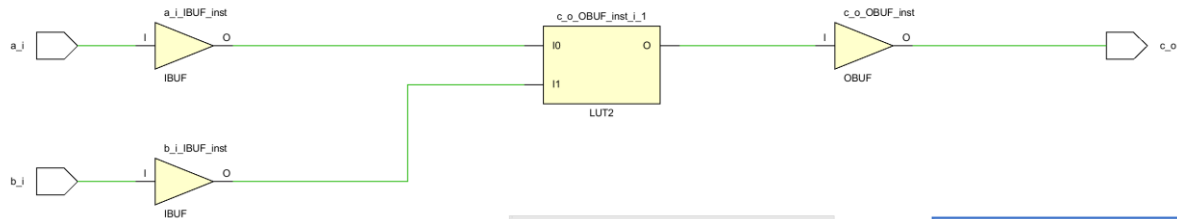
Report DRC

Report Noise

Report Utilization

[Report Power](#)

[Schematic](#)



Cell Properties

c_o_OBUF_inst_i_1

Name: c_o_OBUF_inst_i_1
Reference name: LUT2
Type: LUT
Number of cell pins: 3
Number of nets: 3

General Properties Power Nets Cell Pins Truth Table

Cell Properties

c_o_OBUF_inst_i_1

I1	I0	O=I0 & I1
0	0	0
0	1	0
1	0	0
1	1	1

Edit LUT Equation...

General Properties Power Nets Cell Pins Truth Table

IMPLEMENTATION

▶ Run Implementation

▼ Open Implemented Design

[Constraints Wizard](#)

Edit Timing Constraints

🕒 Report Timing Summary

Report Clock Networks

Report Clock Interaction

📋 Report Methodology

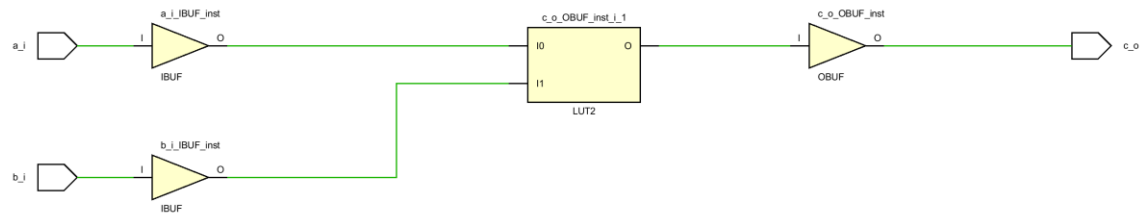
Report DRC

Report Noise

Report Utilization

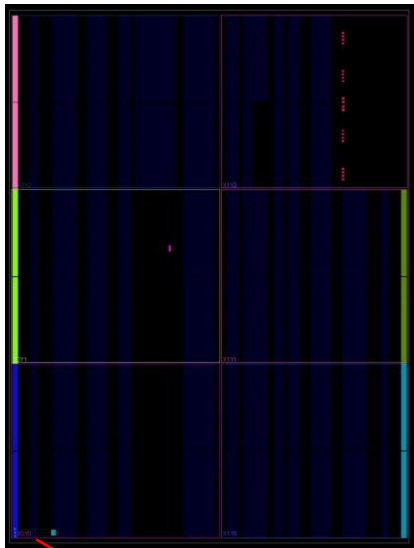
🔌 Report Power

🔌 Schematic



Cell Properties	
Name:	c_o_OBUF_inst_I_1
Reference name:	LUT2
Type:	LUT
BEL:	<input checked="" type="checkbox"/> A6LUT <input type="checkbox"/> Fixed
Site:	<input checked="" type="checkbox"/> SLICE_X0Y1
Title:	<input checked="" type="checkbox"/> CLBLL_L_X2Y1
Clock region:	<input checked="" type="checkbox"/> X0Y0
Number of cell pins:	3
Number of nets:	3

General | Properties | Power | Nets | Cell Pins



Cell Properties x Clock Regions

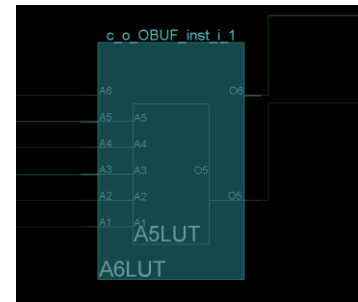
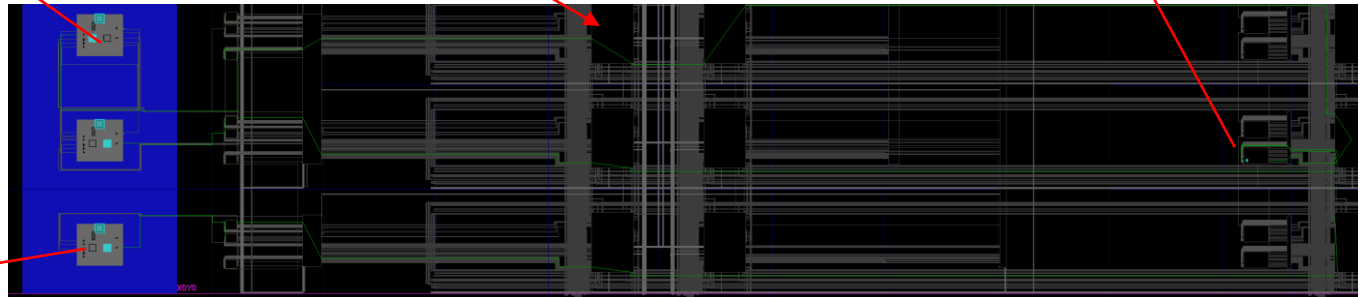
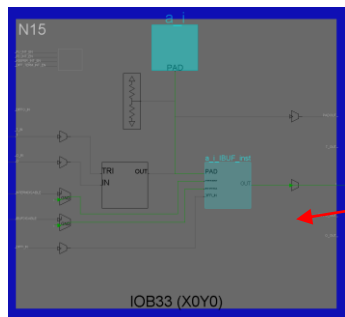
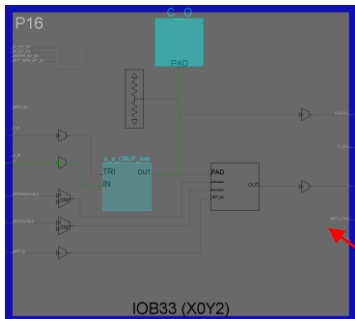
c_o_OBUF_inst_L1

I1 I0 O=I0 & I1

0	0	0
0	1	0
1	0	0
1	1	1

Edit LUT Equation...

General Properties Power Nets Cell Pins **Truth Table**



Verilog HDL行为语句

类别	语句	可综合性
过程语句	initial	
	always	√
块语句	串行块 begin-end	√
	并行块 fork-join	
赋值语句	持续赋值 assign	√
	过程赋值 = , <=	√
条件语句	if - else	√
	case	√
循环语句	for	√
	repeat	
	while	
	forever	
编译指示语句	`define	√
	`include	
	`ifdef, `else, `endif	√

always 语句

- **always 语句是重复执行的。** always 语句块从 0 时刻开始执行其中的行为语句；当执行完最后一条语句后，便再次执行语句块中的第一条语句，如此循环反复。
- **always_comb 适用于组合逻辑电路**

```
module and_gate(  
    input wire a_i,  
    input wire b_i,  
    output reg c_o  
);  
  
    always_comb begin  
        if (a_i && b_i )  
            c_o = 1;  
        else  
            c_o = 0;  
        end  
    endmodule
```

条件语句

if-else语句使用方法有以下3种

(1) if (表达式) 语句1;

(2) if (表达式) 语句1;

else 语句2;

(3) if (表达式1) 语句1;

else if (表达式2) 语句2;

else if (表达式3) 语句3;

.....

else if (表达式n) 语句n;

else 语句n+1;

case语句

case语句的使用格式如下

case (敏感表达式)

值1: 语句1;

值2: 语句2;

.....

值n: 语句n;

default: 语句n+1

endcase

循环语句

在verilog中存在四种类型的循环语句，用来控制语句的执行次数。这四种语句分别为：

(1) **forever**: 连续执行语句；多用在initial块中，以生成时钟等周期性波形。

(2) **repeat**: 连续执行一条语句n次。

(3) **while**: 执行一条语句直到某个条件不满足

(4) **for**: 有条件的循环语句

```
initial
begin
  for(i=0;i<4;i
    =i+1)
    out=out+1;
end
```

```
initial
begin
  repeat(5)
    out = out +1;
end
```

```
initial
begin
  i=0;
  while(i<0)
    i = i + 1 ;
end
```

for语句

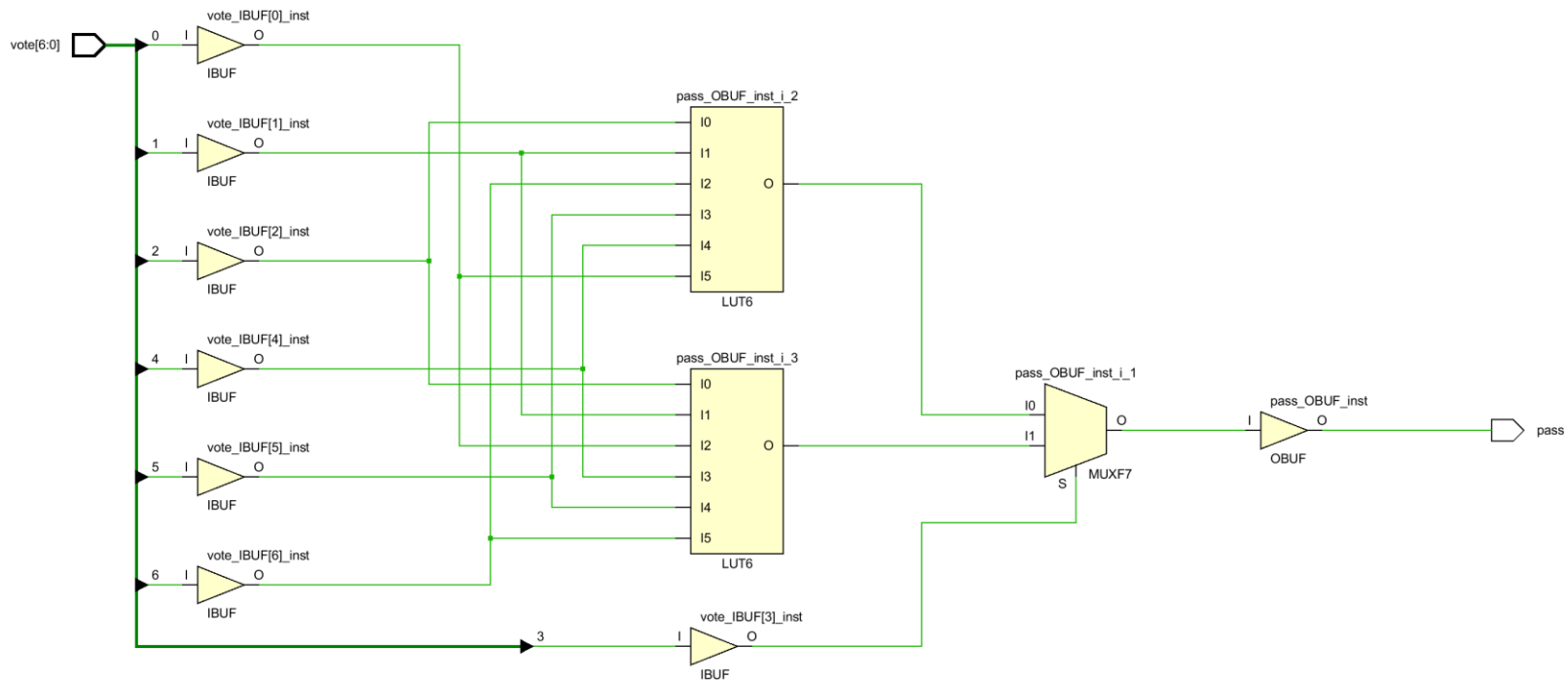
for语句的使用格式如下（同C语言）：

for (表达式1； 表达式2； 表达式3) 语句；

即: **for**（循环变量赋初值； 循环结束条件； 循环变量增值） 执行语句；

用for语句描述七人投票表决器

```
module voter7(pass,vote);  
    output pass;  
    input[6:0] vote;  
    reg[2:0] sum;  
    integer i;  
    reg pass;  
  
    always_comb  
    begin  
        sum=0;  
        for(i=0;i<=6;i=i+1)           //for语句  
            if(vote[i]) sum=sum+1;  
            if(sum[2]) pass=1;          //超过4人赞成，则通过  
            else pass=0;  
    end  
endmodule
```

实验内容

- 同时点亮一个经过译码的数码管和一个未经过译码的数码管
- 数码管根据输入显示从0到f（带译码的显示0到9）

