

Chapter 5 Verilog HDL语言基础

主要内容

5.0 FPGA开发环境及开发板简介

5.1 Verilog HDL基本结构

5.2 数据类型及变量、常量

5.3 运算符

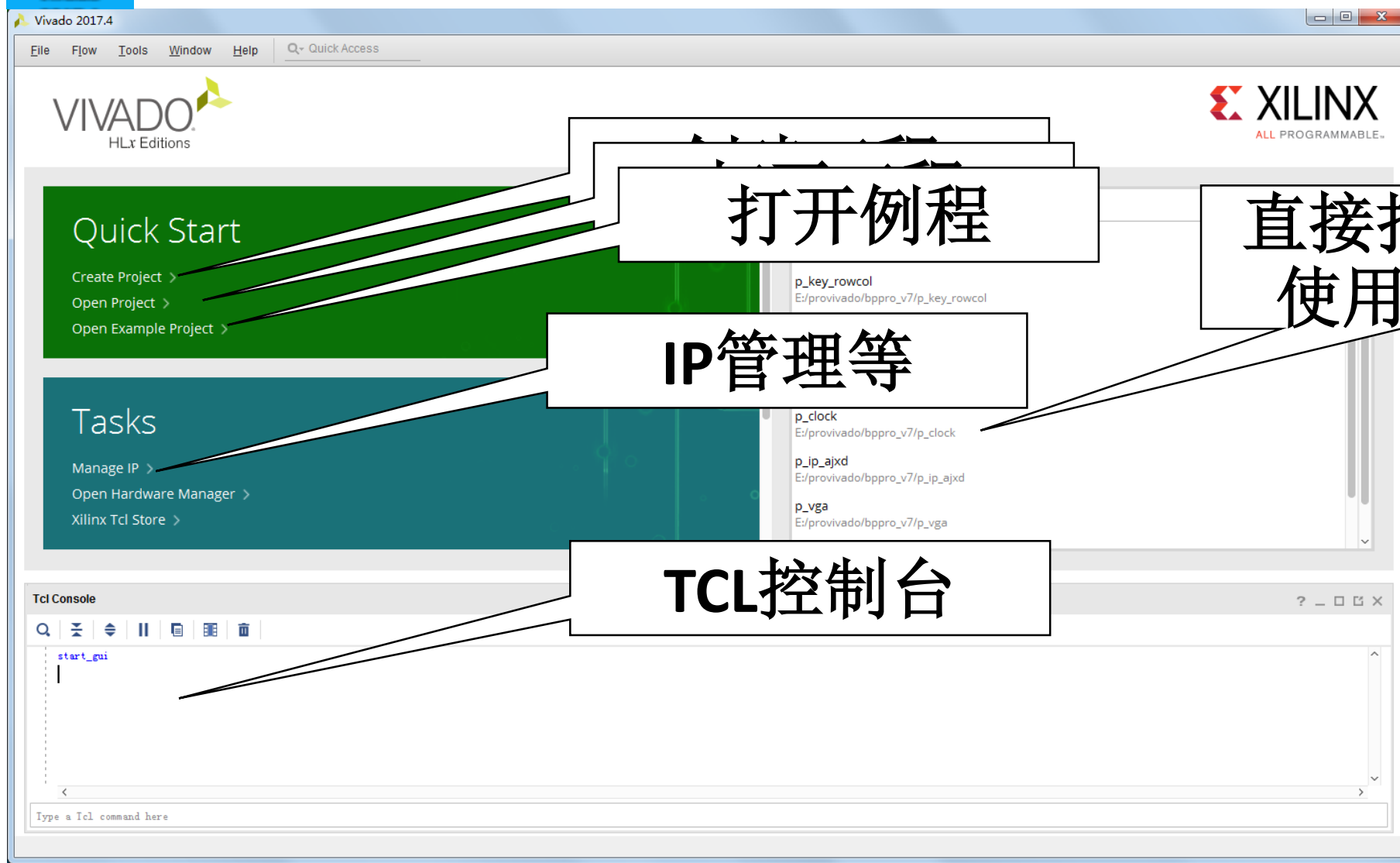
5.4赋值语句

5.0 FPGA开发环境及开发板简介



■ 进入VIVADO

VIVADO主界面



菜单

快速按钮

设置

IP管理

仿真

RTL分析

综合

实现

编程和调试

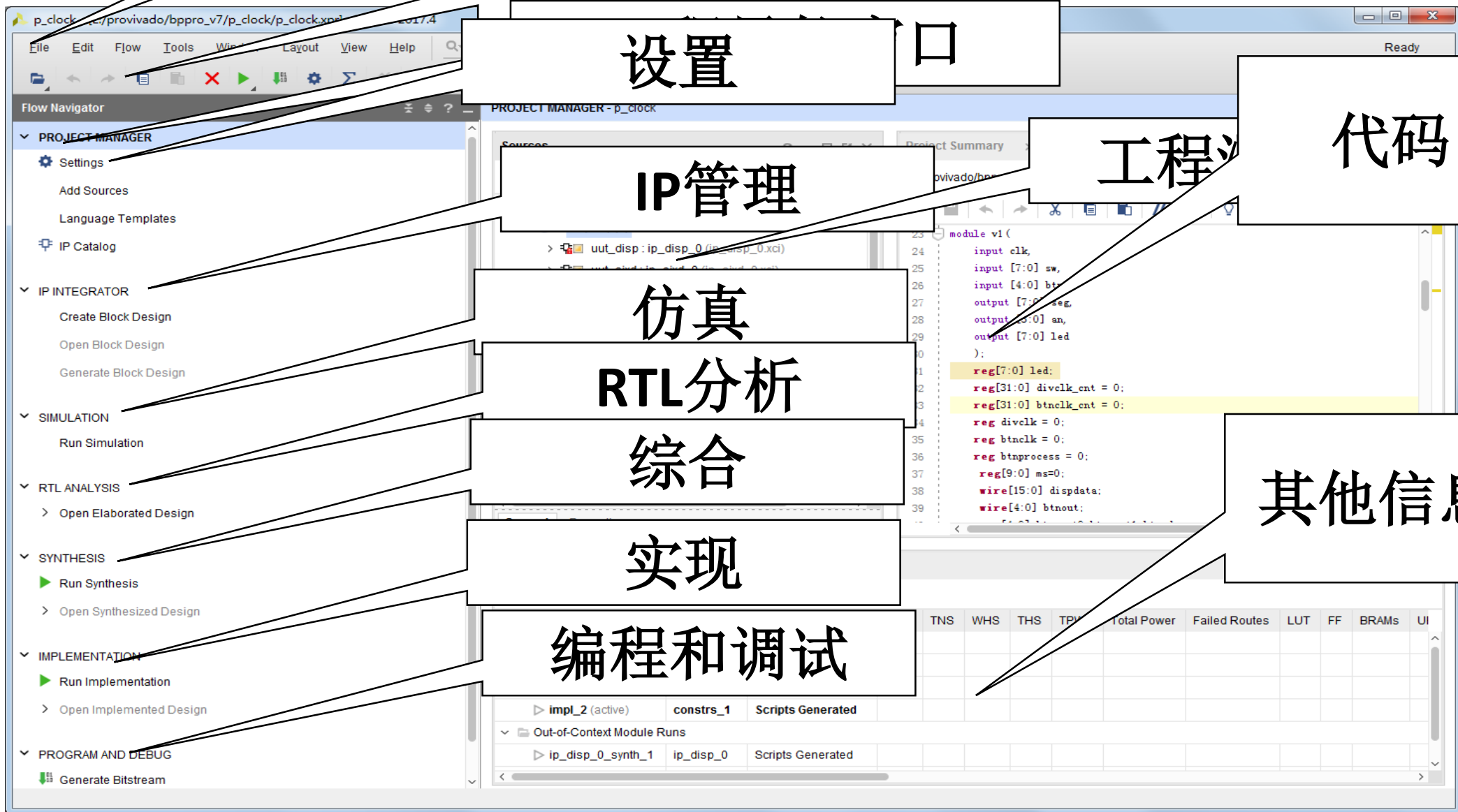
面

日

工程

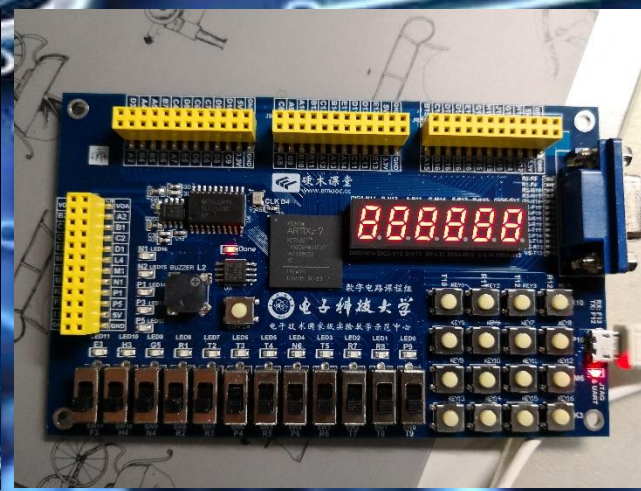
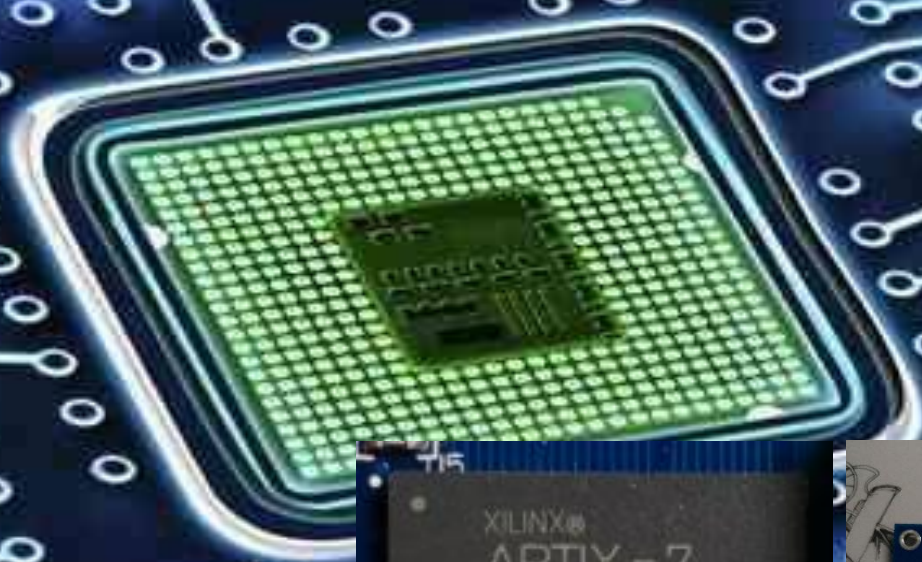
代码

其他信息



数字电路课程组与电子实验中心

电子科技大学



TLC7528

FPGA: xc7a35t

3.3V GND A3 B4 B5 B6 A7 E5 D6 E6 D8 C9

3.3V GND A8 B9 B10 B11 A12 A13 A14 C11 C13 E13

3.3V GND A15 B16 C16 D16 E16 F15 G15 J15 F14 J12 C13

硬木课堂
www.emooc.cc

DIG1-N11 R-N12 A-P14 G-M14 F-P13 F-K13 DIG6-G12 G-N16
DIG2-N14 DIG3-N13 D-K13 DP-L13 DIG4-M12 C-L14 DIG5-H13 VS-T13

XILINX®
ARTIX™-7
XC7A35T™
FTG256ABX1537
D5120993A
1C
TAIWAN
TCU535.00-00

数字电路课程组



电子科技大学

电子技术国家级实验教学示范中心

VOA A2 B1 C2 D1 L4 M1 N1 P1 P5 5V GND

75CDRTK
TLC7528C
3.3V

N1 LED16
N2 LED15 BUZZER L2
P1 LED14
P3 LED13
P5 LED12

Done

UB

CONF#

LED11 E3 LED10 H3 LED9 G5 LED8 R1 LED7 T2 LED6 T3 LED5 T4 LED4 N6 LED3 T5 LED2 R7 LED1 R8 LED0 P9

SW11 F3 SW10 H4 SW9 N4 SW8 R2 SW7 R3 SW6 P4 SW5 R5 SW4 P6 SW3 R6 SW2 T7 SW1 T8 SW0 T9

KEY1 KEY2 KEY3 KEY4
KEY5 KEY6 KEY7 KEY8
KEY9 KEY10 KEY11 KEY12
KEY13 KEY14 KEY15 KEY16

■ 5.1 Verilog HDL基本结构

- **HDL(Hardware Description Language)**硬件描述语言是对硬件电路进行行为描述、寄存器传输描述或者结构化描述的一种语言。
- **FPGA**作为可编程硬件，采用**HDL**语言作为编程语言。
- 通过**HDL**语言可以对**FPGA**的功能进行描述，描述完成后的源代码，通过综合（将高层次的寄存器传输级别的**HDL**设计转化为优化的低层次的**逻辑网表**）和实现及生成目标文件后，**下载**到**FPGA**以实现**对FPGA**进行配置，
- 配置后的**FPGA**实现了**HDL**语言描述的功能。

■ 5.1 Verilog HDL基本结构

4bit的2输入与运算

```
module AND4(a,b,out);  
output[3:0] out; //4位输出  
input [3:0] a,b; //4位输入a和b  
assign out=a&b; //out=a与b进行按位与  
endmodule
```

上述程序实例定义了一个模块**AND4**。

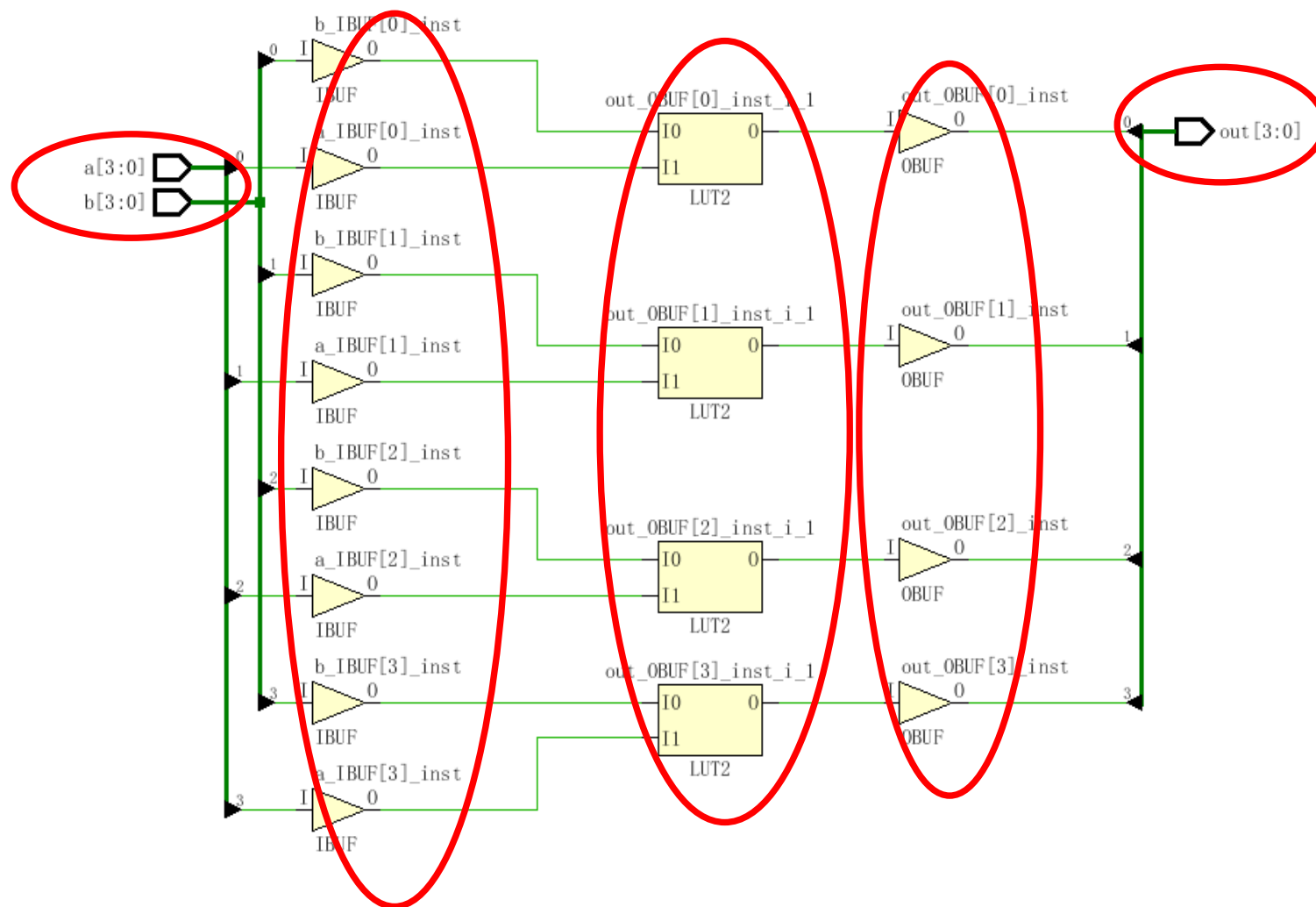
从模块的声明开始，最开始就是关键词**module**，然后是这个模块的名字叫**AND4**，最后是**endmodule**。

■ 5.1 Verilog HDL基本结构

4bit的2输入与运算

方法1: **module**
AND4(a,b,out);
output[3:0] out;
input [3:0] a,b;
assign out=a&b;
endmodule

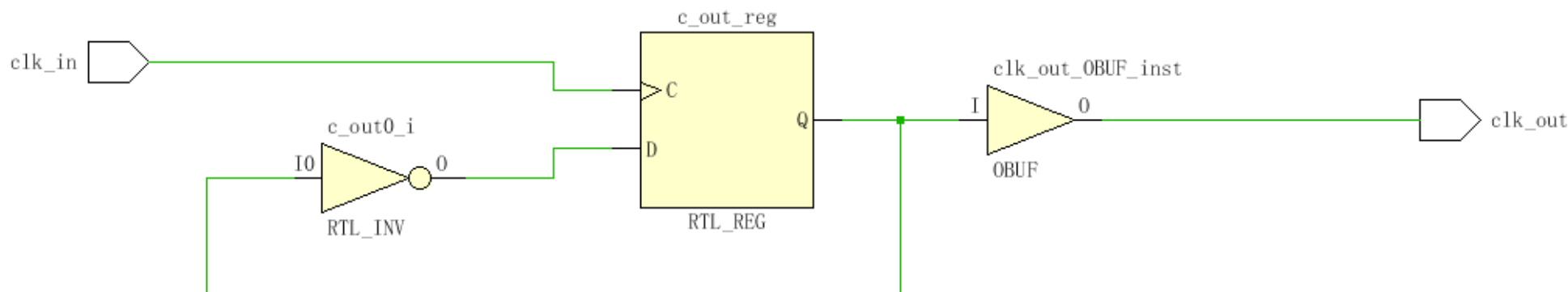
方法2: **module**
AND4(input [3:0] a,
input [3:0] b,
output[3:0] out);
assign out=a&b;
endmodule



■ 5.1 Verilog HDL基本结构

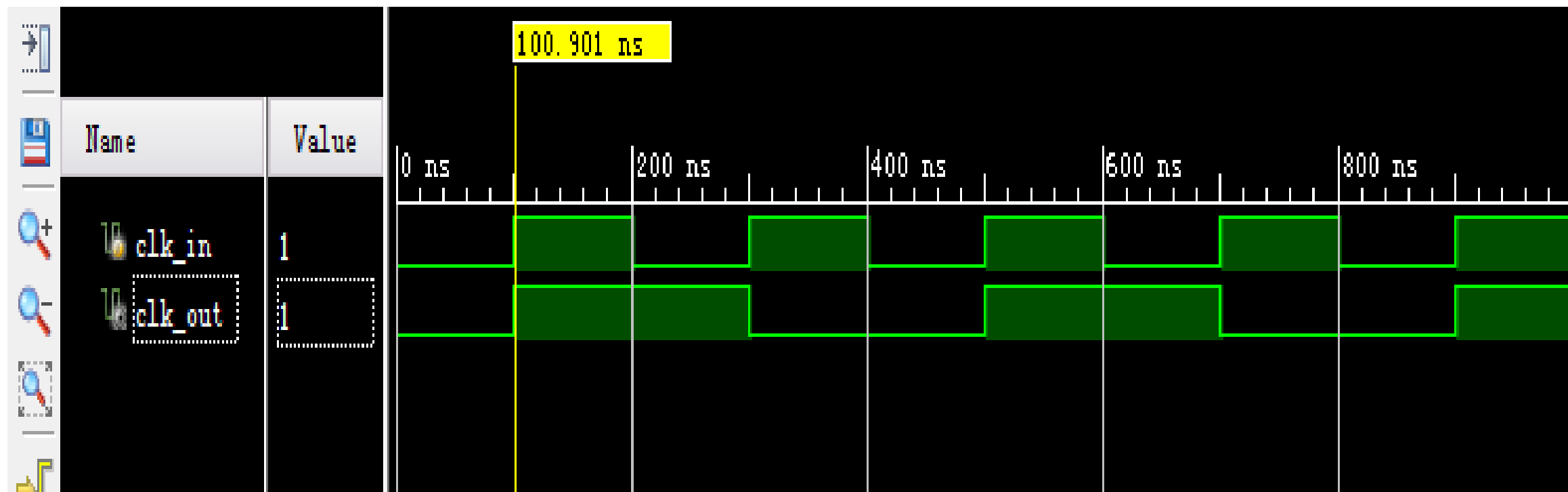
分频器

```
module FenPin(input clk_in, output clk_out);  
  reg c_out=0;  
  assign clk_out=c_out;  
  always @ (posedge clk_in)  
  begin  
    c_out=~c_out;  
  end  
endmodule
```



■ 5.1 Verilog HDL基本结构

分频器



■ 5.1 Verilog HDL基本结构

1. Verilog HDL程序是由模块构成的。每个模块嵌套在**module**和**endmodule**声明语句中。
2. 每个**Verilog HDL**源文件中只有一个顶层模块，其他为子模块。可以每个模块写一个文件。
3. 每个模块要进行端口定义，并说明输入输出端口，然后对模块的功能进行行为逻辑描述。
4. 模块中的时序逻辑部分在**always**块的内部，在**always**块中只能对寄存器变量赋值。
5. 模块中对端口或其他**wire**型变量的赋值，必须在**always**块的外部使用**assign**语句，通常是将寄存器的值送出。
6. 程序书写格式自由，一行可以写几个语句，一个语句也可以分多行写。
7. 除了**endmodule**语句、**begin_end**语句和**fork_join**语句外，每个语句和数据定义的最后必须有分号。
8. 可用**/*.....*/**和**//...**对程序的任何部分作注释。加上必要的注释，可以增强程序的可读性和可维护性。

5.1 Verilog HDL基本结构

```
1 module FenPin(clk_in, clk_out);  
2 input clk_in;  
3 output clk_out;  
4 reg c_out=0;  
5 assign clk_out=c_out;  
6 always @ (posedge clk_in)  
7 begin  
8     c_out=~c_out;  
9 end  
10 endmodule
```

端口定义

端口说明

信号类型声明

行为描述

5.2 数据类型及变量、常量

- **Verilog HDL**有两种常用的数据类型，线网（**Net**）类型及变量类型。
- 常量的值是不能够被改变的，变量的值是可以被改变的，例如寄存器型的变量**reg**。
- 从逻辑值和常量式开始，对于线网类型重点是常用的**wire**型变量，对于变量类型重点是**reg**型。

5.2 数据类型及变量、常量-逻辑值

逻辑值	含义
0	逻辑0
1	逻辑1
x	逻辑值未知
z	高阻

5.2数据类型及变量、常量-数的表达

表 达 方 式	说 明	举 例
<位宽> ' <进制> <数字>	完整的表达方式	4' b0101或4'h5
<进制> <数字>	缺省位宽，则位宽由机器系统决定，至少32位。	h05
<数字>	缺省进制为十进制，位宽默认为32位。	5

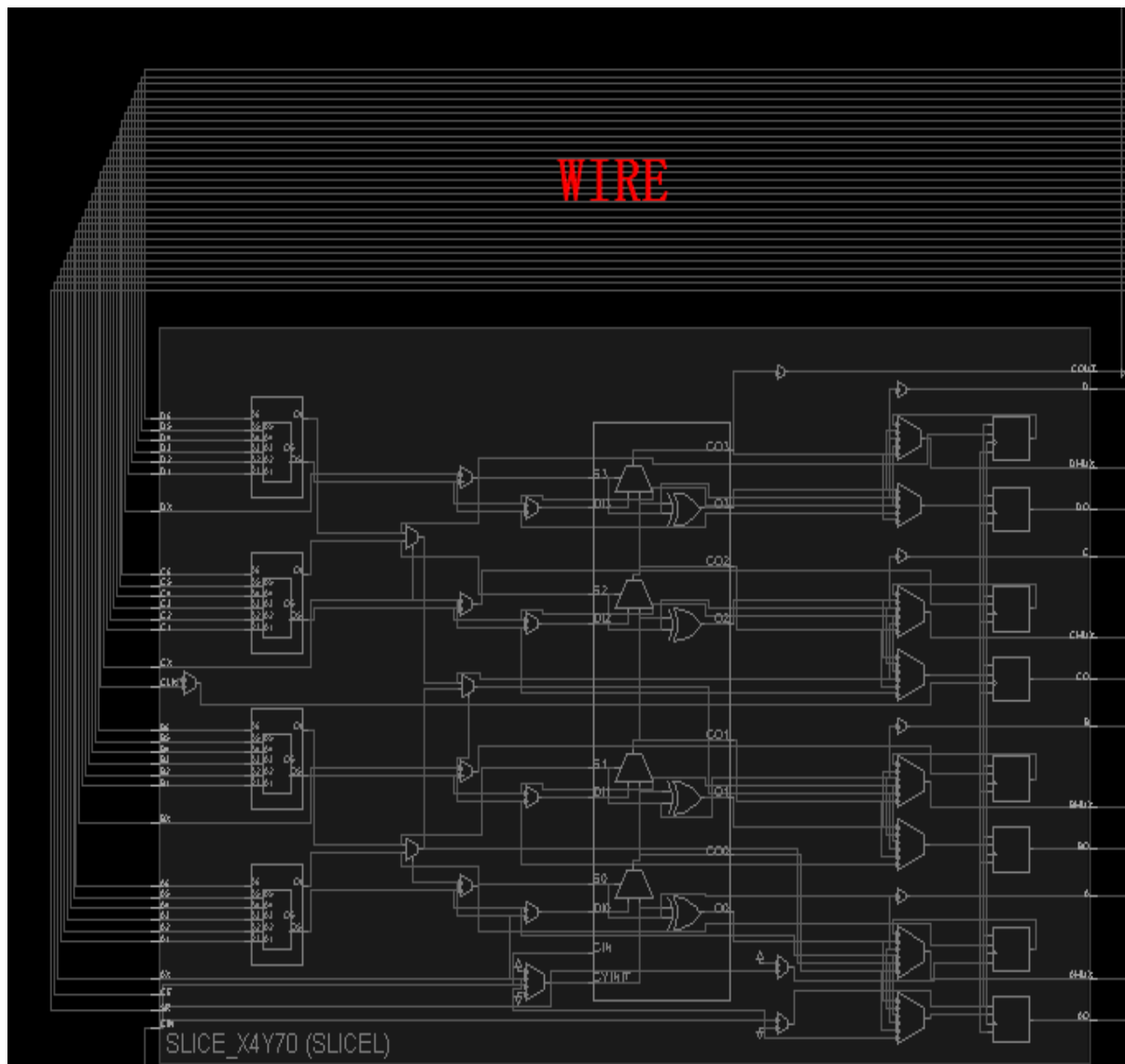
4'h5实际上就是4' b0101，只不过是用十六进制表示

8'h2x是8位二进制数，对应二进制数的值是8'b0010xxxx,低4位不确定。

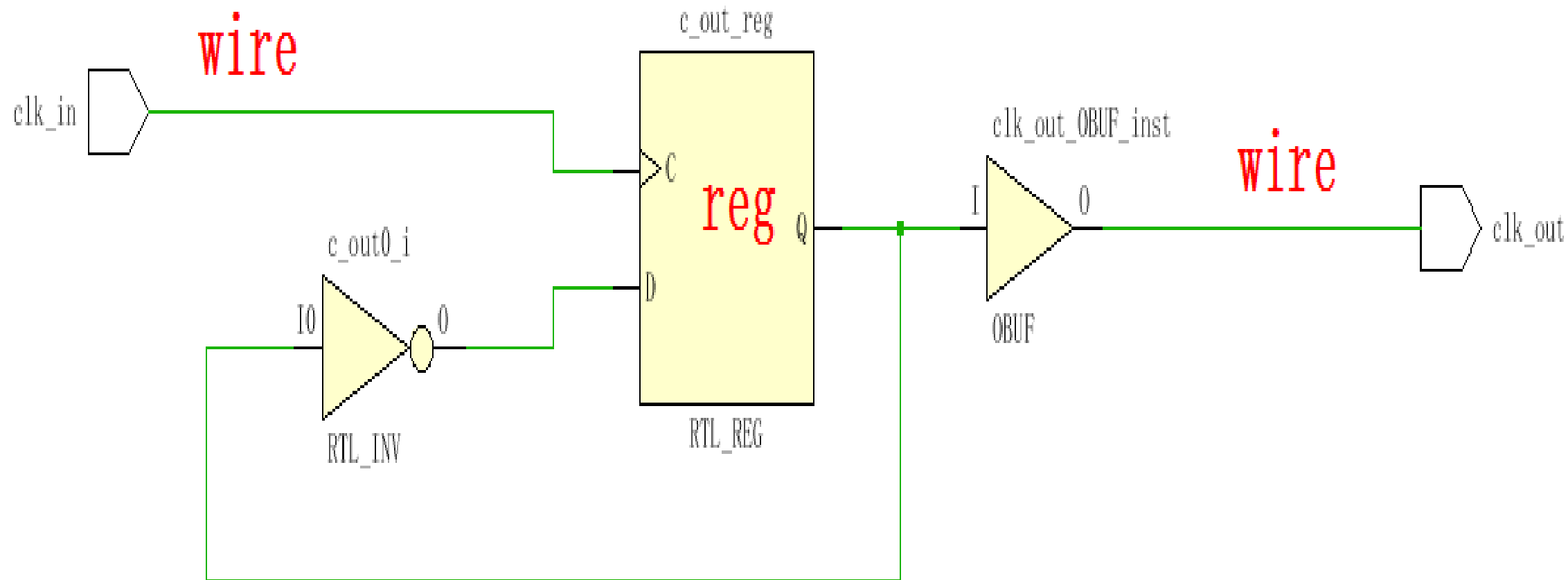
16是32位的二进制数，十进制的值是16

5.2 数据类型及变量、常量-wire

- 线网（**net**）型变量最常用的就是**wire**，最大的问题就是怎么去理解**wire**。可以将**wire**直接的理解为连线。
- 例如一个**D**触发器**reg1**的输出是**Q**，这个**Q**连接到端口**out1**上，那么**out1**的值始终跟随着**reg1**的值的而变化而变化。
- **wire clk_out;**
- **assign clk_out=c_out;**



5.2 数据类型及变量、常量-wire



5.2 数据类型及变量、常量-wire

wire型信号的定义格式如下：

定义一个n位的wire变量： `wire [n-1:0] 变量名;`

定义m个n位的wire变量： `wire [n-1:0] 变量名1, 变量名2,变量名m;`

下面给出2个例子：

`wire [7:0] a, b, c; // a, b, c都是位宽为8位的wire`

`wire d; //d是1位的wire`

5.2 数据类型及变量、常量-reg

reg型也称为寄存器型。
数字电路中的触发器只在时钟有效边沿到来的时候，保存的值才能够发生改变。

```
wire clk_out;
```

```
reg c_out;
```

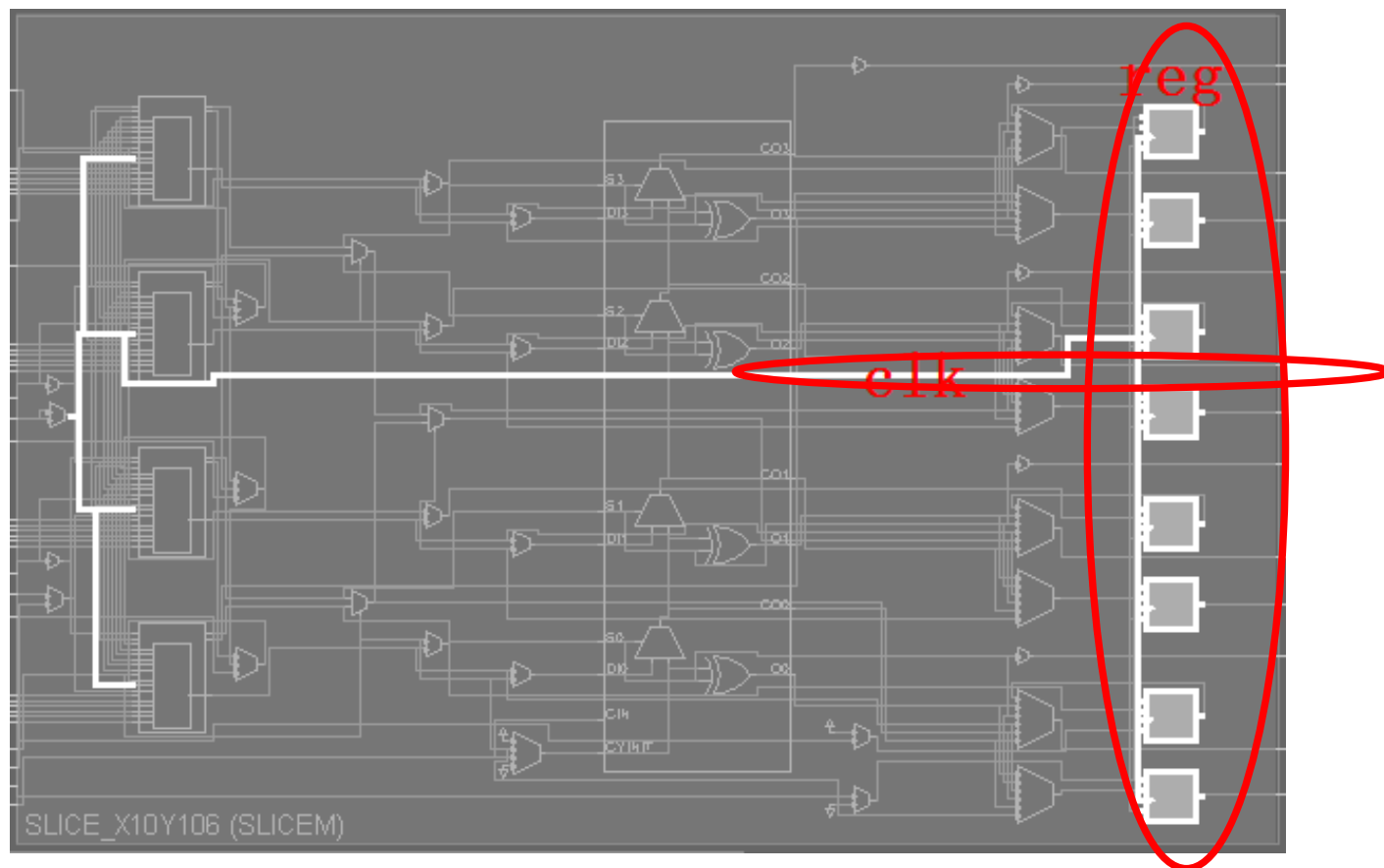
```
assign clk_out=c_out;
```

```
always @ (posedge clk_in)
```

```
begin
```

```
    c_out=~c_out;
```

```
end
```



5.2 数据类型及变量、常量-reg

reg型也称为寄存器型。
数字电路中的触发器只在时钟有效边沿到来的时候，保存的值才能够发生改变。

```
wire clk_out;
```

```
reg c_out;
```

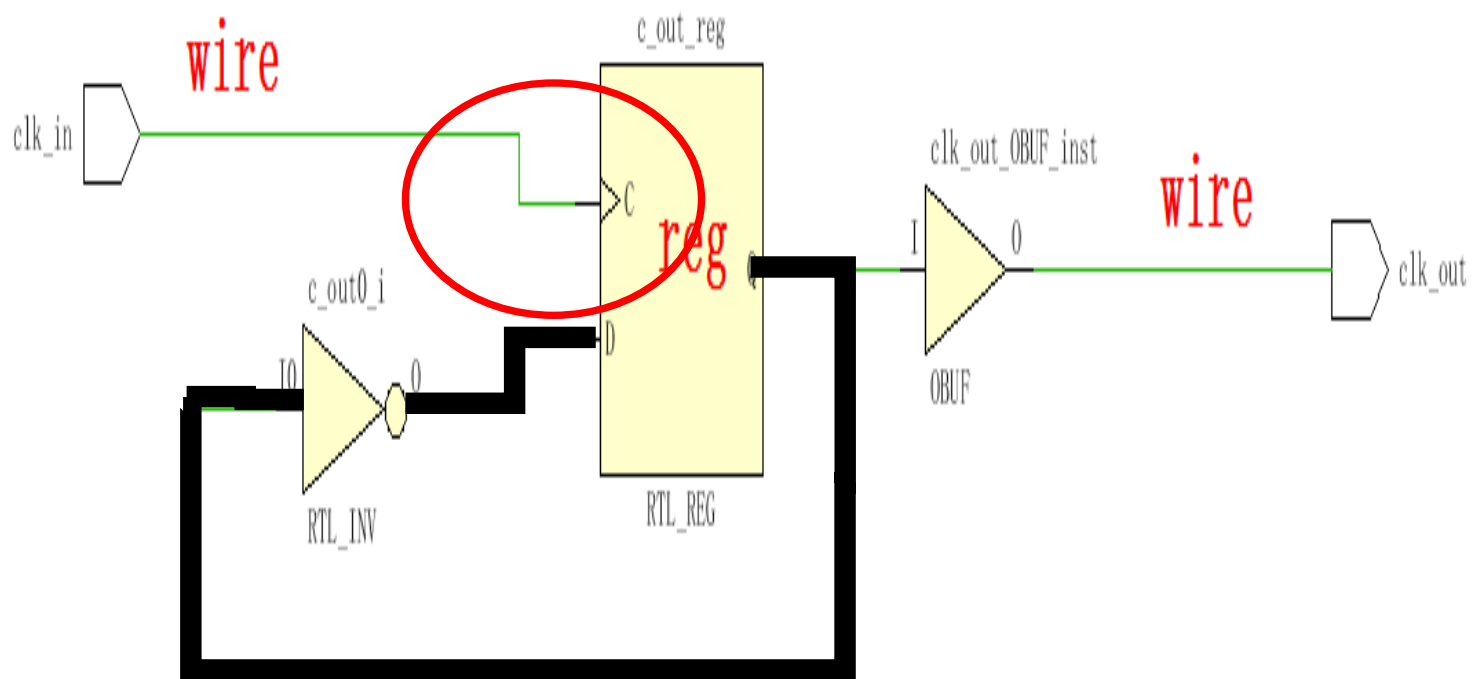
```
assign clk_out=c_out;
```

```
always @ (posedge clk_in)
```

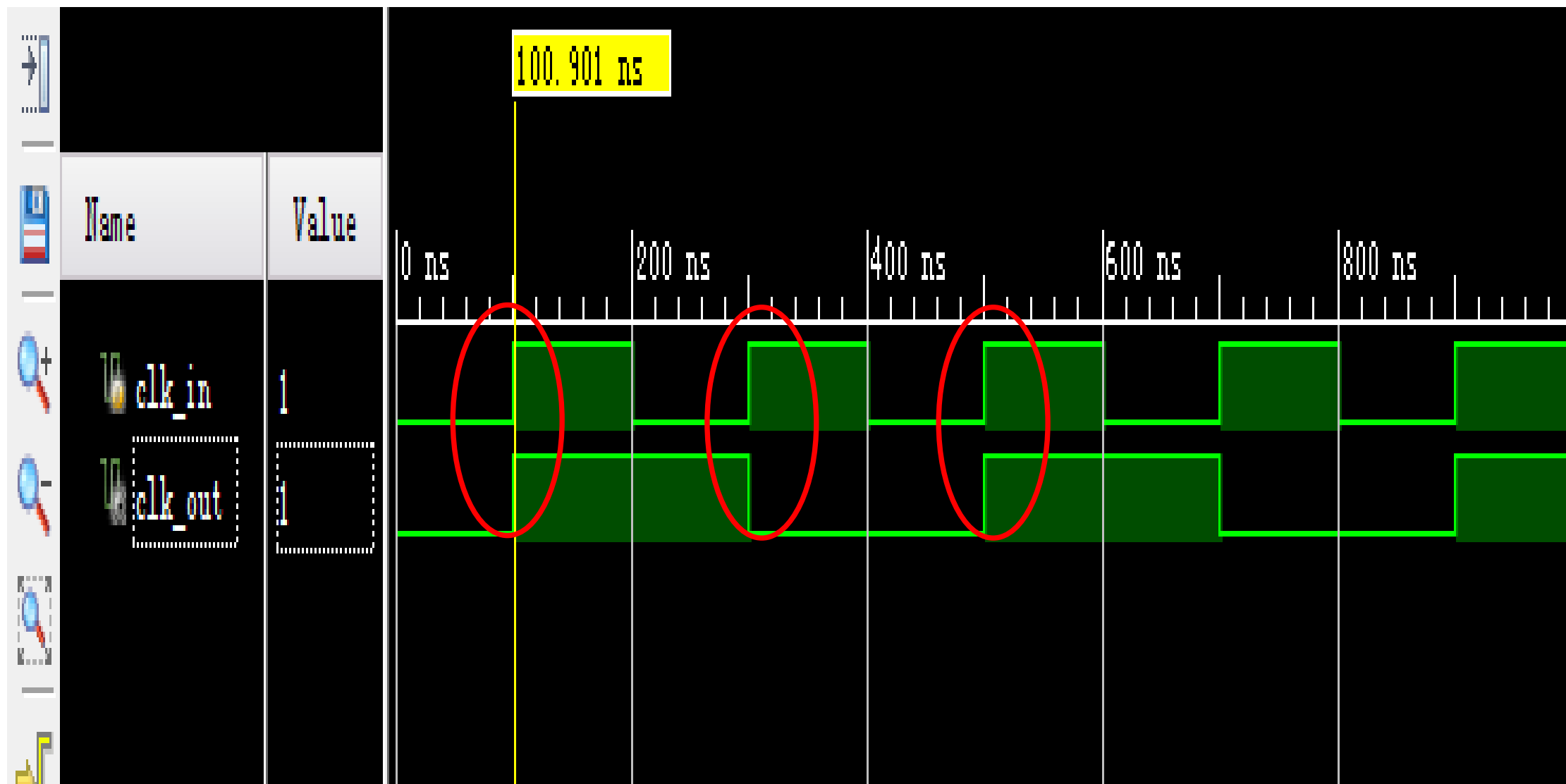
```
begin
```

```
    c_out=~c_out;
```

```
end
```



5.2 数据类型及变量、常量-**reg**



5.2 数据类型及变量、常量-reg

reg型信号的定义格式如下：

定义一个n位的寄存器变量： reg [n-1:0] 变量名；

定义m个n位的寄存器变量：

reg [n-1:0] 变量名1， 变量名2，变量名m；

下面给出2个例子：

reg [7:0] a, b, c; // a, b, c都是位宽为8位的寄存器

reg d; //1位的寄存器d

5.2 数据类型及变量、常量-符号常量

如果用**关键词parameter**来定义一个标识符，代表一个常量，这个常量就被称为符号常量。

例如：**parameter width=3;**//符号常量**width**的值是**3**，如果未进行重定义，当在程序中出现**width**时就用**3**代替。

parameter idle=1,one=2,two=3,stop=4; //定义了4个符号常量。如果未进行重定义，当代码中出现**idle**就用**1**代替，出现**one**就用**2**代替，出现**tow**就用**3**代替，出现**stop**就用**4**代替。

■**思考：****parameter width=3;**那么**width**就一定是**3**吗？

这是不一定的，因为使用**parameter**定义的常量，仍然可以**重定义**。

5.2 数据类型及变量、常量-符号常量

```
module adder(sum,a,b);
```

【1】 ↵

```
    parameter time_delay=5,time_count=10;
```

【2】 ↵

```
        .....↵
```

```
endmodule↵
```

```
module top;↵
```

```
    wire[2:0] a1,b1;↵
```

```
    wire[3:0] a2,b2,sum1;↵
```

```
    wire[4:0] sum2;↵
```

```
    adder  #(4,8)  AD1(sum1,a1,b1);//time_delay=4,time_count=8
```

【3】 ↵

```
    adder  #(12)  AD2(sum2,a2,b2);//time_delay=12,time_count=10
```

【4】 ↵

```
endmodule↵
```

5.2数据类型及变量、常量-存储器型

存储器实际上是一个寄存器数组。存储器使用如下方式定义

reg [msb: lsb] memory1 [upper1: lower1]

从高到低或从低到高均可

(**msb**是最高有效位，**lsb**是最低有效位)。

例如：

reg[3:0] mymem1[63:0] //mymem1为64个4位寄存器的数组。

reg dog [1:5] //dog为5个1位寄存器的数组。

dog[4] = 1 ; //合法赋值语句,对其中一个1位寄存器赋值。

dog[1:5] = 0 ; //合法赋值语句,对存储器大范围赋值。

5.3 运算符-算术运算符

算术运算符↵	说明↵
+↵	加↵
-↵	减↵
*↵	乘↵
/↵	除↵
%↵	取摸（取余数）↵

- 在进行整数的除法运算时，结果要略去小数部分，只取整数部分；而进行取模运算时（**%**，亦称作求余运算符）
- 结果的符号位采用模运算符中第一个操作数的符号。
- 例如，**-10%3** 结果 **-1**，**11%-3** 结果为**2**。
- 在进行算术运算时，如果某一个操作数有不确定的值**x**，则整个结果也为不确定值**x**。

5.3 运算符-逻辑运算符

逻辑运算符↵	说明↵
&& (双目)↵	逻辑与↵
(双目)↵	逻辑或↵
!(单目)↵	逻辑非↵

- 逻辑运算只区分真假，而不管是什么数值。逻辑运算的输入**4'ha1**和**4'h01**是没有区别的，都是逻辑真，而**0**为逻辑假。一般来说，逻辑运算的结果要么为真（**1**）要么为假（**0**）。
- 特例是如果有一个输入为未知**X**，那么结果也是**X**。
- 例如，**4'ha1&&4h01**是**1**，
4'ha1&&4h00是**0**。
- 只有两个输入都是**0**的时候，逻辑或的结果才是**0**。
- 对于逻辑非，当输入为非**0**值，输出就是**0**。
- 逻辑运算最常用于条件判断语句。

5.3运算符-按位运算符

位运算符	说明
\sim	按位取反
$\&$	按位与
$ $	按位或
\wedge	<u>按位异或</u>
$\wedge\sim, \sim\wedge$	按位同或

- 通常使用按位运算符完成基本的与、或、非、异或及同或逻辑运算。使用这些位运算符进行组合，很容易完成其他的逻辑运算。
- 按位运算要求对两个操作数的相应位逐位进行运算。
- 例如 **$0101\&1100=0100$** , **$0101|1100=1101$**

5.3 运算符-关系运算符

关系运算符	说明
<	小于
<=	小于或等于
>	大于
>=	大于或等于

- 关系运算符和逻辑运算符一般用于条件判断语句。
- 关系运算结果为1位的逻辑值**1**（真）或**0**（假），但也可能是**x**（未知）。关系运算符根据关系运算的结果是真还是假，用于条件判断。
- 关系运算时，若关系为真，则返回值为**1**；若声明的关系为假，则返回值为**0**；若某操作数为不定值**x**，则返回值也一定为**x**。

5.3 运算符-等式运算符

等式运算符	说明
==	等于
!=	不等于
===	全等
!==	不全等

- “==”和“!=”称作逻辑等式运算符，其结果由两个操作数的值决定。由于操作数可能是 x 或 z ，其结果可能为 x 。
- “===”和“!==”常用于**case**表达式的判别，又称作**case**等式运算符。其结果只能为**0**和**1**。如果操作数中存在 x 和 z ，那么操作数必须完全相同结果才为**1**，否则为**0**。
- “===”和“=”是完全不同的，“=”是对寄存器赋值使用的。

■5.3运算符-缩减运算符

缩减运算符	说明
$\&$	与
$\sim \&$	与非
$ $	或
$\sim $	或非
\sim	异或
$\sim \sim, \sim \sim$	同或

- 例如有4位的变量**b**，**c=&b**的含义是
- c=((b[0]&b[1]) &b[2]) & b[3];**



■5.3运算符-移位运算符

移位运算符	说明
\gg	右移
\ll	左移

■ $a \gg n$ 中 a 代表要进行向右移位的操作数， n 代表要移几位。
。 $a \ll n$ 表示将 a 逻辑左移 n 位。

这两种移位运算都用**0**来填补移出的空位。

■例如 a 是一个**5**位的寄存器，那么 $a = 4'b1001 \ll 1$, a 的结果就是**5'b10010**。

■但是当 a 是一个**4**位的寄存器，那么 $a = 4'b1001 \ll 1$, a 的结果就是**4'b0010**。

■ 5.3 运算符-条件运算符

- 条件运算符是书写效率非常高的运算符。

条件运算符为？，很像C语言里面的？运算符。它实现的是组合逻辑，或者说就是多路复用器。

- 用法： **assign wire**类型变量 = 条件？表达式1：表达式2；

- 例如 **assign out = sel? in1:in0**；描述了

当**sel**为**1**的时候 **out=in1**;

否则， **out=in0**;

是多路选择逻辑。

- 如果想对寄存器变量赋值，**不能**使用条件运算符，而要用**always**块中使用条件判断语句。


5.3运算符-拼接运算符

- 拼接运算是非常有意思的，非常高效率。
- 使用拼接运算符可以将变量任意组合后输出或送给另一个变量。
- 用法：{信号1的某几位，信号2的某几位，.....信号n的某几位}
-
- {a, b[3:0],w,3'b101}
- //等同于{a,b[3],b[2],b[1],b[0],w,1b'1,1'b0,1'b1}
- {4{w}} //等同于{w,w,w,w}
- {b,{3{a,b}}}
- //等同于{b,a,b,a,b,a,b}
- 这里的3、4必须是常量表达式。

5.3 运算符-拼接运算符

- 例如变量**r1**定义为 **reg[7:0] r1**，并且**a,b**都是**8**位的变量。
- **r1={a[3:0],b[3:0]}**
- 表示**r1**的 **r[7]=a[3],r[6]=a[2]...r[0]=b[0]**。
- 如果**a=17**即**8'b00010001**，**b=138**即**8'b10001010**，那么**r1=8'b0001 1010**。
- **r1={r1[3:0],r1[7:4]}**实现了将**r1**的高**4**位和低**4**位交换。

■ 5.3 运算符-运算符优先级

类 别	运 算 符	优先级
逻辑、位运算符	! ~	高
算术运算符	* / %	
	+ -	
移位运算符	<< >>	
关系运算符	< <= > >=	
等式运算符	= = ! = == = ! ==	
缩减、位运算符	& ~&	
	^ ^~	
	~	
逻辑运算符	&&	
条件运算符	?	低

■ 提高程序的可读性，建议使用括号来控制运算的优先级！