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

5.4赋值语句

- **1** 连续赋值语句**assign**
- **assign**语句
- 用于对**wire**型变量赋值，是描述组合逻辑最常用的方法之一。
- 例如，**assign c=a&b;**
- **//a、b**可以是**wire**型变量或寄存器变量，**c**必须是**wire**型变量或其他线网型变量。

- **2** 过程赋值语局“**=**”和“**<=**”
- 用于对**reg**型变量赋值
- 在过程块中使用过程赋值语句。

结构说明语句always

- always块包含一个或一个以上的语句(如:过程赋值语句、条件语句和循环语句等），在运行的全过程中，在时钟控制下被反复执行。
- 时钟有效边沿来了就执行。
- 在always块中被赋值的只能是寄存器reg型变量。
- always块的写法是:always @ (敏感信号表达式)
- 例如：
- always @ （clk） //只要clk发生变化就触发
- always @ （posedge clk） //clk上升沿触发
- always @ （negedge clk） //clk下降沿触发
- always @ （negedge clk1 or posedge clk2） // clk1下降沿触发，clk2上升沿也触发
- always @ （*）该语句所在模块的任何输入信号变化了都触发。

结构说明语句initial

initial语句用于对寄存器变量赋予初值

```
module ifblock(clk,i_a,o_b,o_c);
input clk,i_a;
output o_b,o_c;
reg b,c;                                    【1】
assign o_c=c;
assign o_b=b;
initial                                     【2】
begin
        b=0;c=0;
end;
always @(posedge clk)
    begin
        b<=i_a; //非阻塞赋值
        c<=b;
    end
endmodule
```

阻塞与非阻塞

- 阻塞的概念：
- 在一个块语句中，如果有多条阻塞赋值语句，在前面的赋值语句没有完成之前，后面的语句就不能被执行，就像被阻塞了一样，因此称为阻塞赋值方式。

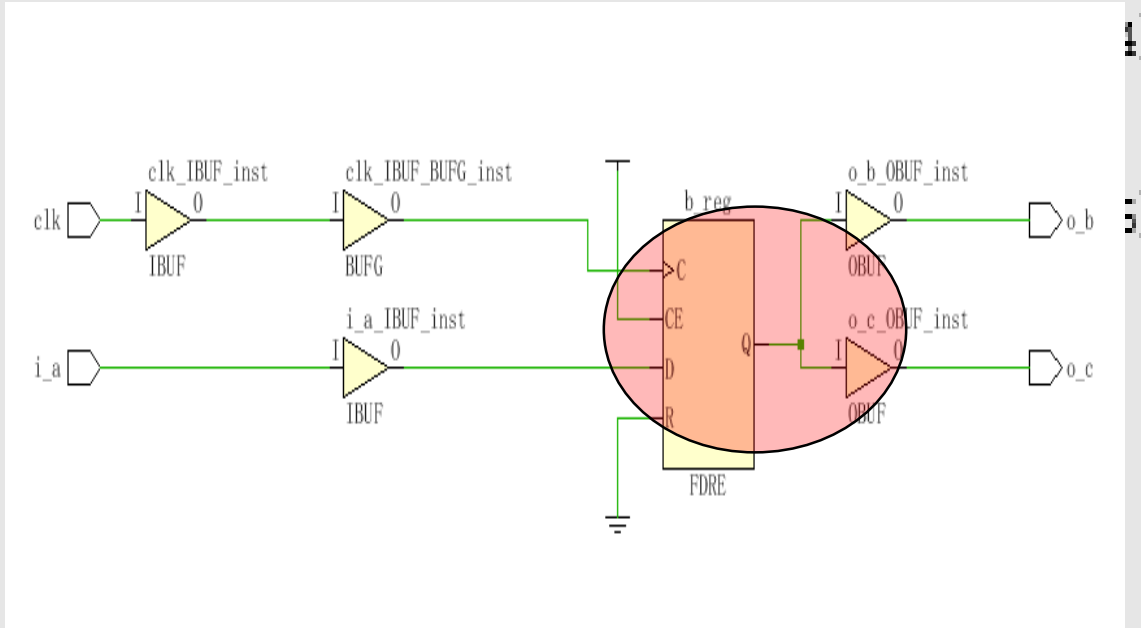
- 非阻塞的概念：
- 多条非阻塞赋值在过程块内同时完成赋值操作,多条语句相当于同时执行！

- 非阻塞（**non-blocking)**赋值方式：
- 赋值符号为**<=**，如 **b <= a** ；
- 阻塞（**blocking)**赋值方式：
- 赋值符号为**=**，如 **b = a** ；
- 注意，非阻塞和阻塞是截然不同的！

阻塞与非阻塞

阻塞的研究

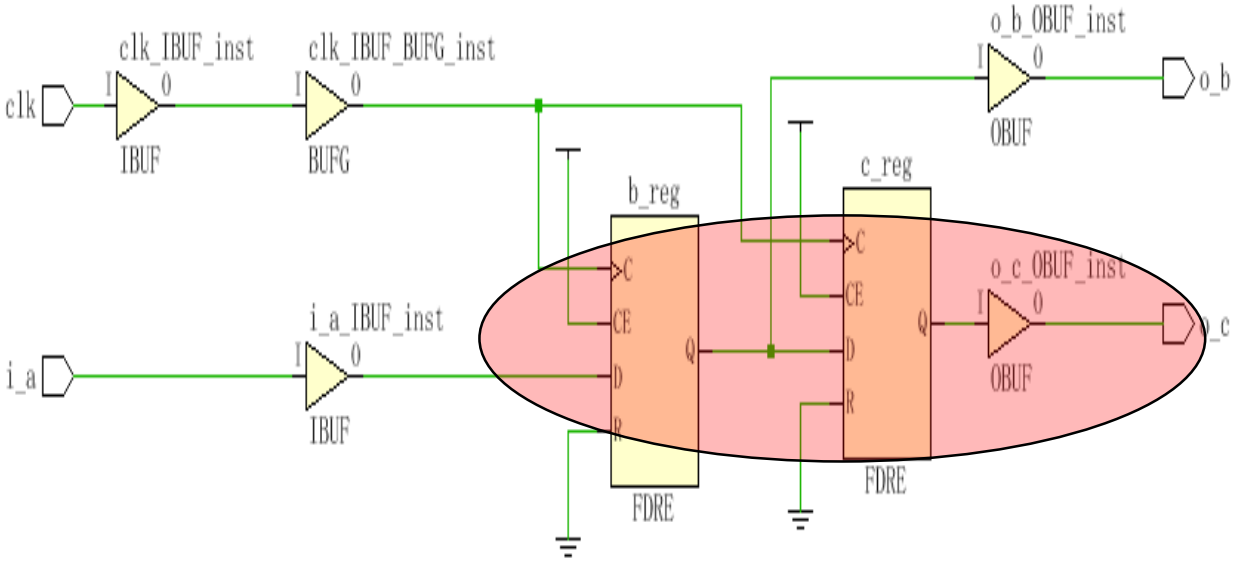
```
module ifblock(clk,i_a,o_b,o_c);
input clk,i_a;
output o_b,o_c;
reg b=0,c=0;
assign o_c=c;
assign o_b=b;
always @(posedge clk)
    begin
        b=i_a; //阻塞赋值
        c=b;
    end
endmodule
```



阻塞与非阻塞

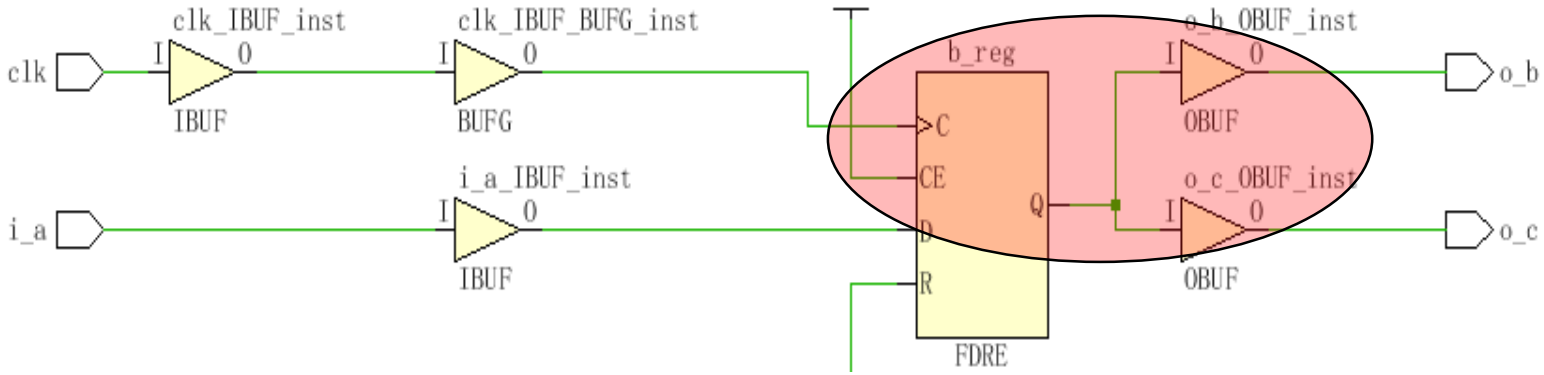
非阻塞的研究

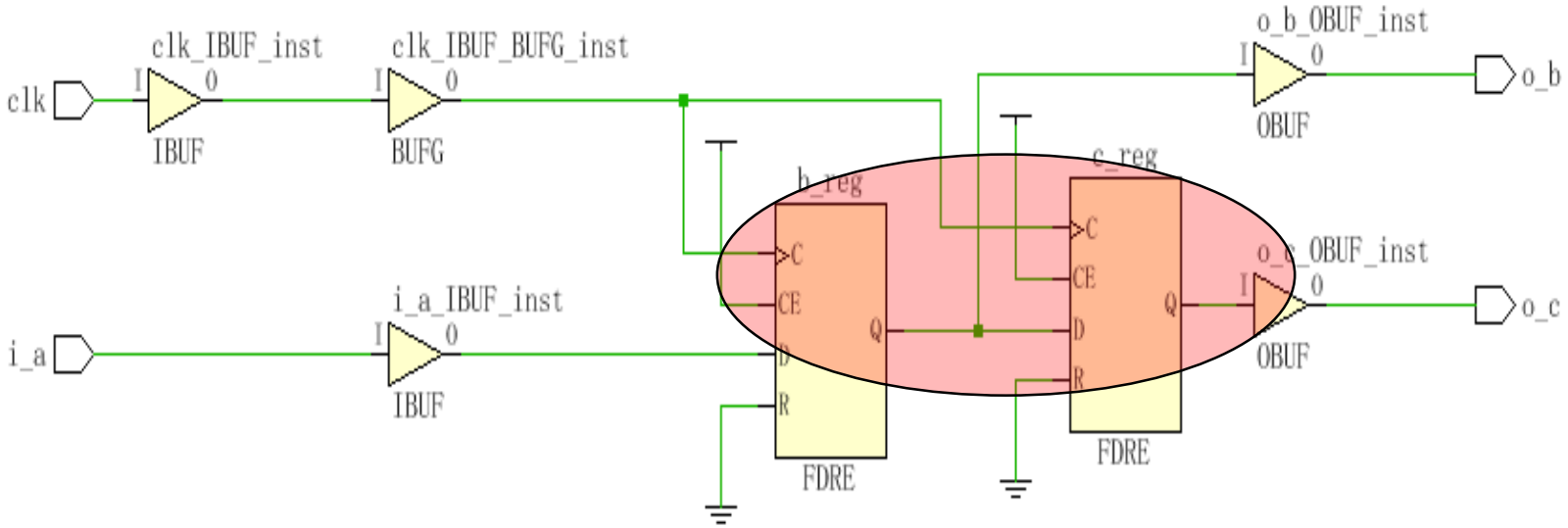
```
module ifblock(clk,i_a,o_b,o_c);
input clk,i_a;
output o_b,o_c;
reg b=0,c=0;
assign o_c=c;
assign o_b=b;
always @(posedge clk)
    begin
        b<=i_a; //非阻塞赋值
        c<=b;
    end
endmodule
```



阻塞与非阻塞

非阻塞及阻塞的比较



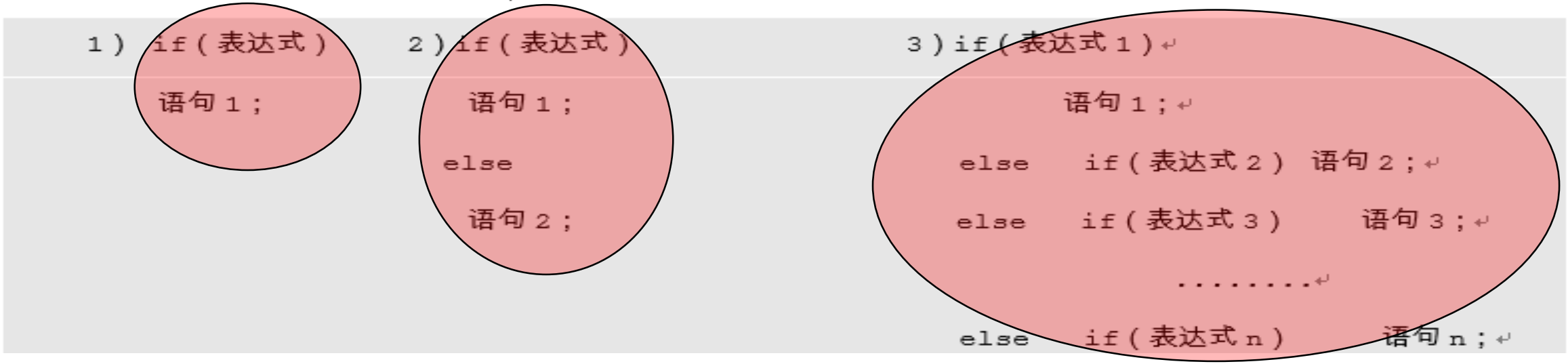


条件语句if

- 条件语句用于always或Initial过程块内部，
- 主要包含if-else语句和case语句。
- if-else语句
- 用于判定所给条件是否满足，根据判定的结果（真或假）决定执行给出的两种操作之一。if-else语句有3种形式。

条件语句if

表达形式1：if-else语句有3种形式



- 如果语句有多条组成，必须包含在**begin**和**end**之内。
- **3**种形式的**if**语句后面都有表达式，一般为逻辑表达式或关系表达式。当表达式的值为**1**，按真处理，若为**0**、**x**、**z**，按假处理。
- **else**语句不能单独使用，它是**if**语句的一部分。

条件语句if

```
always@(posedge clk) //把系统时钟分频 50M/1000=50000 1000HZ
begin
    if(cnt1==26'd25000)
    begin
        clk_ms =~ clk_ms;
        cnt1 = 0;
    end
    else
    begin
        cnt1 = cnt1+1'b1;
    end
end
```

case语句

- **case**语句是一种多分支选择语句，**if**只有两个分支可以选择，但是**case**可以直接处理多分支语句，这样程序看起来更直观简洁。

表达形式 2：case 语句的 3 种形式

```
1）case（表达式）      <case 分支项>   endcase
2）casex（表达式）     <case 分支项>   endcase
3）casez（表达式）     <case 分支项>   endcase
```

- **case**(表达式)
- 分支表达式：　　语句；
- 分支表达式：　　语句；
- …… ……
- 默认项（**default**）　　语句；
- **endcase**

case语句

```
module mux4to1(out,a,b,c,d,select);
       output out;
       input a,b,c,d;
       input[3:0] select;
       reg out; //该寄存器名称与输出信号 out 同名，实际将其输出送 out 端口
      always@ (select[3:0] or a or b or c or d)
      begin
          casex (select)
               4'b???1: out = a;
               4'b??1?: out = b;
               4'b?1??: out = c;
               4'b1???: out = d;
          endcase
      end
endmodule
```

case语句

- **case**语句的所有表达式的值的位宽必须相等。
- 在**case**语句中，分支表达式每一位的值都是确定的（或者为**0**，或者为**1**）；
- 在**casez**语句中，若分支表达式某些位的值为高阻值**z**，则不考虑对这些位的比较；
- 在**casex**语句中，若分支表达式某些位的值为**z**或不定值**x**，则不考虑对这些位的比较。

case	0	1	x	z
0	1	0	0	0
1	0	1	0	0
x	0	0	1	0
z	0	0	0	1

casez	0	1	x	z
0	1	0	0	1
1	0	1	0	1
x	0	0	1	1
z	1	1	1	1

casex	0	1	x	z
0	1	0	1	1
1	0	1	1	1
x	1	1	1	1
z	1	1	1	1

循环语句

- 在**Verilog**中存在着多种循环语句，用来控制执行语句的执行次数。
- 这些语句**C**语言中很常见，也是必须的。
- 在**FPGA**设计中，循环语句不一定能被综合，
- 多用于在仿真代码生成仿真激励信号。

循环语句forever

- **forever**语句： 连续执行的语句。
- 格式：**forever begin**语句块**end**
- **forever**常用于仿真代码中。
- 如下仿真程序代码中的【1】和【2】是等价的，都是产生**20**个时间单位的方波，占空比**50%**。
- 仿真的时间单位，可以在系统中设置，也可以在仿真文件的开始加上：
- 比如，**`timescale 1ns / 1ps //**时间单位**1ns,** 精度**1ps**

```
forever                                    【1】
begin
#10 clk=1;
#10 clk=0;
end

always #10 clk=~clk                        【2】
```

循环语句repeat

- **repeat**语句： 连续执行**n**次的语句。
- 格式：**repeat**（表达式） **begin**语句块**end**

- 其中“表达式”用于指定循环次数，可以是一个整数、变量或者数值表达式。如果是变量或者数值表达式，其数值只在第一次循环时得到计算，从而得到确定循环次数。
- **repeat**语句也常用于仿真。

例如

```
repeat (10)
begin
  #10 clk=0;
  #10 clk=1;
end
```

循环语句while

- **while**语句：　执行语句，直至某个条件不满足。
- 　　格式：**while**（表达式）　**begin**语句块**end**
- 表达式是循环执行条件表达式，代表了循环体得到继续重复执行时必须满足的条件，通常是一个逻辑表达式。
- 在每一次执行循环体之前，都需要对这个表达式是否成立进行判断。**while** 语句在执行时，首先判断循环执行条件表达式是否为真，如果真，执行后面的语句块， 然后再重新判断循环执行条件表达式是否为真，直到条件表达式不为真为止退出循环。
- 在执行语句中，如果没有改变循环执行条件表达式的值的语句，循环就成为死循环。

循环语句while

```
module  cnt1 (in,clk,cnt );
        output[3:0] cnt;
        input [7:0] in;
        input clk;
        reg[3:0]  cnt;
        reg[7:0] temp;          //用作循环执行条件表达式
      always @(posedge clk)
     begin
            cnt = 0;                    //cnt 初值为 0
            temp = in;              //  tempreg 初值为 rega
            while(temp>0)           // 若 tempreg 非 0，则执行以下语句
            begin
                  if(temp[0]) cnt = cnt+1;   //只要 tempreg 最低位为 1，则 count 加 1
                  temp=temp>>1;  //右移 1 位
            end
         end
endmodule
```

循环语句for

- **for** 语句： 三个部分
- **for**（表达式1；表达式2；表达式3）
- **for**（循环变量赋初值；循环执行条件；循环变量增值）。
- 例如
- **for(i=0;i<=7;i=i+1)**。

- 如果要让系统能够综合，那么循环的次数一定是固定的。
- 程序实例实现了统计输入的8位数据in中1的个数，每个时钟上升沿统计一次。

循环语句for

```
module cnt2(in,clk,cnt );
        output[3:0] cnt;
        input [7:0] in;
        input clk;
        reg[3:0] i;
        reg[3:0]  cnt;
        always @(posedge clk)
        begin
                cnt = 0;                          //cnt 初值为 0
                for (i=0;i<=7;i=i+1)              // 循环 8 次
                begin
                        if(in[i]) cnt = cnt+1;    //如果位 i 为 1，则 cnt 加 1
                end
           end
endmodule
```

综合结果为时钟同步状态机