阶段一 数字电路设计讲义

目录

[一、实验任务 2](#_Toc137248254)

[二、实验目的： 2](#_Toc137248255)

[三、实验内容 2](#_Toc137248256)

[3.1 组合逻辑电路 2](#_Toc137248257)

[3.1.1 3-8译码器 2](#_Toc137248258)

[3.1.2 带有优先级的83编码器 3](#_Toc137248259)

[3.1.3 多路选择器 4](#_Toc137248260)

[3.2 时序逻辑电路设计 5](#_Toc137248261)

[3.2.1最简单的D触发器 5](#_Toc137248262)

[3.2.2带低电平有效异步复位端的触发器 5](#_Toc137248263)

[3.3.3带同步复位端的D触发器 6](#_Toc137248264)

[3.3.4 带使能端的D触发器 7](#_Toc137248265)

[3.3 面向设计的verilog语法 8](#_Toc137248266)

[3.3.1 模块定义 8](#_Toc137248267)

[3.3.2 信号方向 9](#_Toc137248268)

[3.3.3 常用的两个基本的信号数据类型 reg型、wire型。 9](#_Toc137248269)

[3.3.4 常量 10](#_Toc137248270)

[3.3.5运算符 10](#_Toc137248271)

[3.3.6条件语言 11](#_Toc137248272)

[3.3.7连续赋值语句 assign 用来实现简单的组合逻辑电路。 11](#_Toc137248273)

[3.3.8过程 12](#_Toc137248274)

[3.4 面向测试的Verilog语法 13](#_Toc137248275)

[3.4.1 测试示意图 13](#_Toc137248276)

[3.4.2完整的测试模块 13](#_Toc137248277)

[3.4.3测试举例 13](#_Toc137248278)

[3.4.4元件例化 14](#_Toc137248279)

[3.4.5产生激励向量 15](#_Toc137248280)

[3.4.6 显示输出结果 15](#_Toc137248281)

[3.4.7特殊符号# 15](#_Toc137248282)

[3.4.8 “ ` ”表示编译引导语句 15](#_Toc137248283)

# 一、实验任务

1.学会Vivado使用。

2.翻阅《夏宇闻数字逻辑设计.pdf》对课上讲到的重要关键字进行复习。module,endmodule,input,output,inout,wire,reg,assign,alaways,initial,begin,end,posedge,negedge,case,endcase,default,if,else,for,`define,`include,`timescale。

3.完成组合逻辑电路设计。38译码器，83编码器，多路选择器。

4.完成时序逻辑电路D触发器的设计（四种）。使用verilog语言进行描述并且仿真,得到正确的波形图。

# 二、实验目的：

熟悉并运用Verilog语言进行电路设计。

（1）RTL代码编写电路。

RTL（Register Transfer Level）直译为寄存器传输级，顾名思义，也就是在这个级别下，描述各级寄存器（受clk约束的触发器），以及寄存器之间的组合逻辑。

通俗来讲，RTL代码不是在“写代码”，是在画电路结构。RTL代码需要“画”出输入输出端口，各级寄存器，寄存器之间的组合逻辑，以及这三者之间的连接关系。

（2）testbench和测试激励编写

（3）运行功能仿真

# 三、实验内容

## 3.1 组合逻辑电路

### 3.1.1 3-8译码器

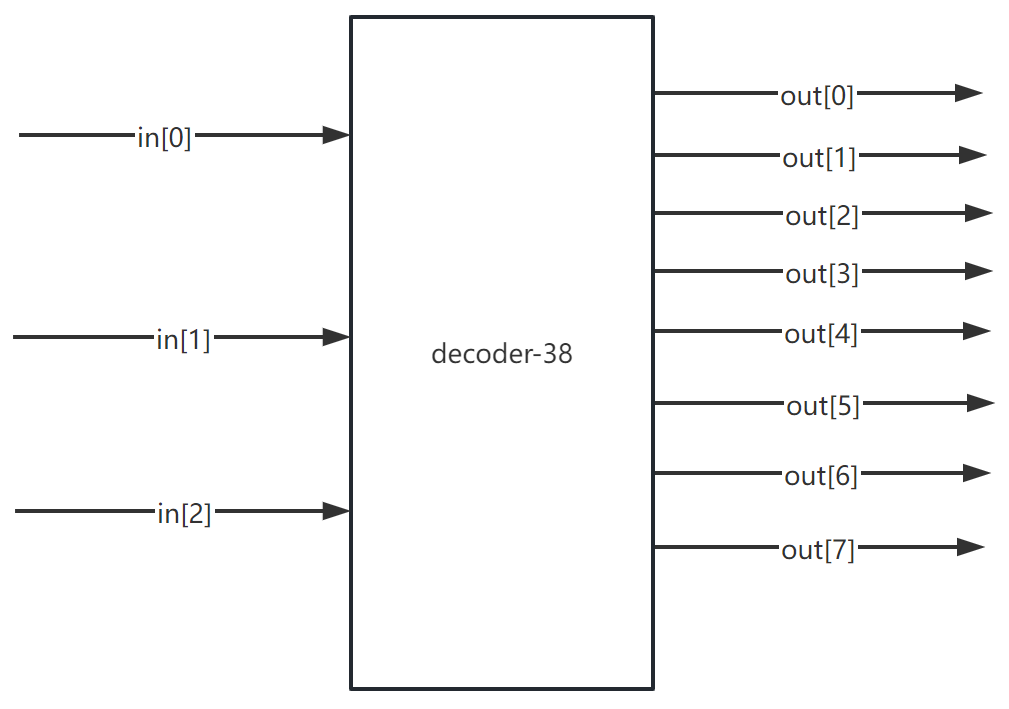


图 3.1 38译码器模块图

如图3.1所示，为38译码器的电路模块图。38译码器有3线输入，8线输出。3线输入会组成000到111共8个不同的数字，输出对应会有8种状态。表3.1是38译码器的真值表。译码器的应用场景是在cpu会根据指令操作码去译码，从而识别不同指令。

表 3.1 38译码器真值表

|  |  |
| --- | --- |
| 输入 | 输出 |
| in[2] in[1] in[0] | out[7] out[6] out[5] out[4] out[3] out[2] out[1] out[0] |
| 000 | 0000 0001 |
| 001 | 0000 0010 |
| 010 | 0000 0100 |
| 011 | 0000 1000 |
| 100 | 0001 0000 |
| 101 | 0010 0000 |
| 110 | 0100 0000 |
| 111 | 1000 0000 |

### 3.1.2 带有优先级的83编码器

如图3.2所示，为带优先级的83编码器模块图，83编码器有8线输入，3线输出。某一位为1，则输出对应的数字。应用场景是：根据指令译码后的结果生成ALU模块的操作码alu\_op。

表3.2为83编码器的真值表。

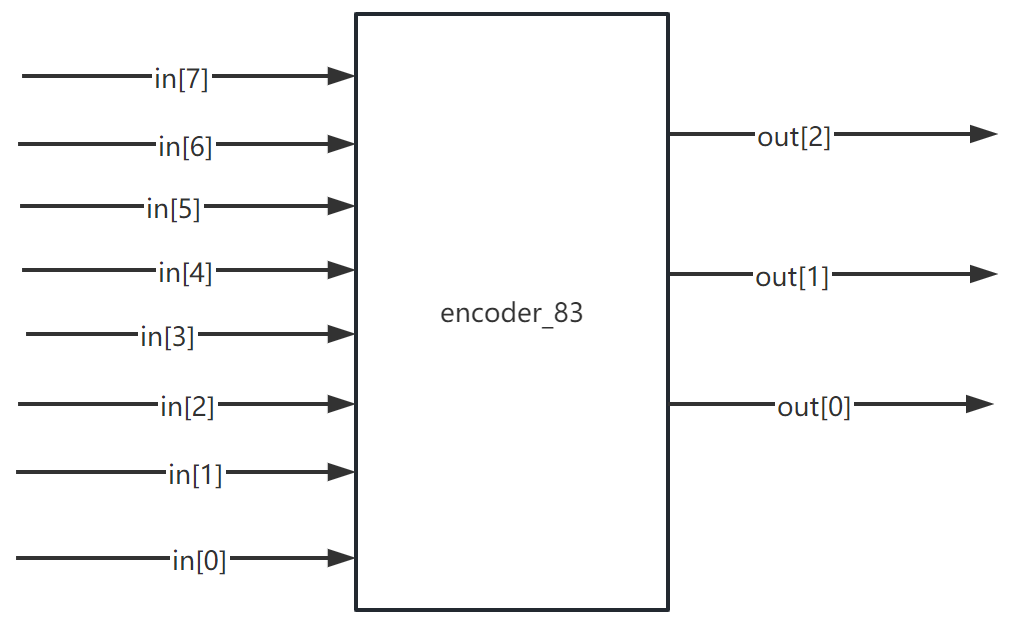


图 3.2 83编码器模块图

表 3.2 83编码器真值表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 输入 | | | | | | | | 输出 | | |
| In[7] | In[6] | In[5] | In[4] | In[3] | In[2] | In[1] | In[0] | Out[2] | Out[1] | Out[0] |
| x | x | x | x | x | x | x | 1 | 0 | 0 | 0 |
| x | x | x | x | x | x | 1 | 0 | 0 | 0 | 1 |
| x | x | x | x | x | 1 | 0 | 0 | 0 | 1 | 0 |
| x | x | x | x | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| x | x | x | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| x | x | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| x | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

### 3.1.3 多路选择器

多路选择器是cpu常用的一种逻辑。在我们1阶段的电路模块中会多次用到多路选择器。图3.3是5选1多路选择器的模块图，表3.3是5选1多路选择器的真值表。其中sel[2:0]是选择子，用来控制输出out[7:0]应为哪一路输入。多路选择器在CPU的设计中会多次使用，多用来作为多路数据的选择，以及结果的选择。

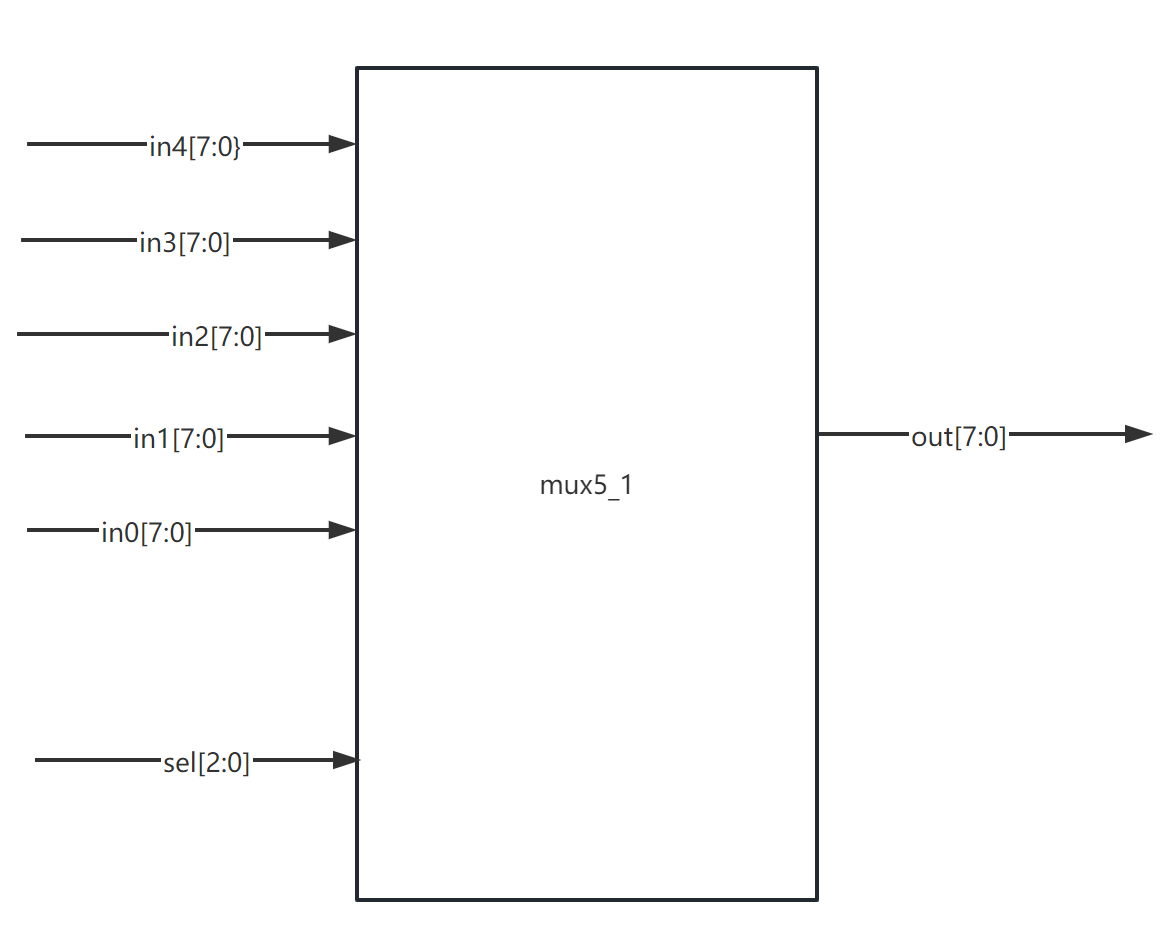


图 3.3 5选1多路选择器

表 3.3 5选1多路选择器

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 选择子 | | | 输入 | | | | | 输出 |
| sel[2] | sel[1] | sel[0] | in4[7:0] | in3[7:0] | in2[7:0] | in1[7:0] | in0[7:0] | out[7:0] |
| 0 | 0 | 0 | in4[7:0] | in3[7:0] | in2[7:0] | in1[7:0] | in0[7:0] | in0[7:0] |
| 0 | 0 | 1 | in4[7:0] | in3[7:0] | in2[7:0] | in1[7:0] | in0[7:0] | in1[7:0] |
| 0 | 1 | 0 | in4[7:0] | in3[7:0] | in2[7:0] | in1[7:0] | in0[7:0] | in2[7:0] |
| 0 | 1 | 1 | in4[7:0] | in3[7:0] | in2[7:0] | in1[7:0] | in0[7:0] | in3[7:0] |
| 1 | 0 | 0 | in4[7:0] | in3[7:0] | in2[7:0] | in1[7:0] | in0[7:0] | in4[7:0] |

## 3.2 时序逻辑电路设计

Cpu设计中时序电路设计都是D触发器的基础上去设计的。

### 3.2.1最简单的D触发器

如下为最简单的D触发器的实现代码，图3.4为其模块示意图。表3.4为其真值表。

always @(posedge clk) begin

q <= d ;

end

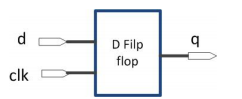


图 3.4 简单D触发器示意图

表 3.4 简单D触发器的真值表

|  |  |  |
| --- | --- | --- |
| **时钟信号CLK** | **输入D** | **输出Q** |
| **其余时刻** | **X** | **Qprev** |
| **上升沿** | **1** | **1** |
| **上升沿** | **0** | **0** |

### 3.2.2带低电平有效异步复位端的触发器

如下为带低电平有效异步复位端的触发器实现代码，图3.5为其模块示意图。表3.5为其真值表。

always @(posedge clk or negedge reset )

begin

if ( !reset )

begin

q <= 0;

end

else//对应了时钟边沿

begin

q <= d ;

end

end

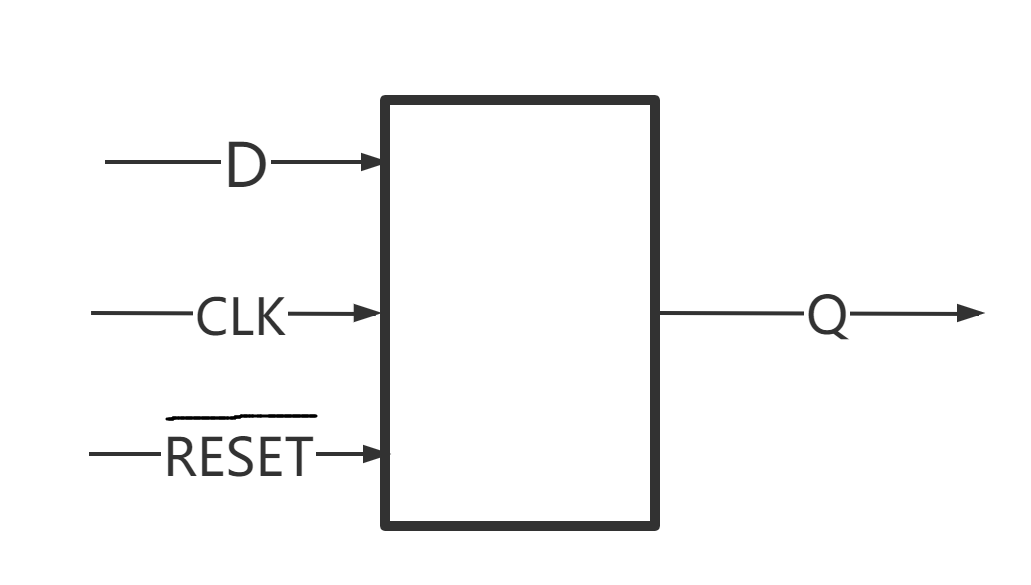


图 3.5 带低电平有效异步复位端的触发器模块图

表 3.5 带低电平有效异步复位端的触发器真值表

|  |  |  |  |
| --- | --- | --- | --- |
| **时钟信号CLK** | **复位信号reset’** | **输入D** | **输出Q** |
| **任意时刻** | **0** | **X** | **0** |
| **上升沿** | **1** | **1** | **1** |
| **上升沿** | **1** | **0** | **0** |
| **其余时刻** | **1** | **X** | **Qprev** |

### 3.2.3带同步复位端的D触发器

如下为带同步复位端的D触发器实现代码，图3.6为其模块示意图。表3.6为其真值表。

always @(posedge clk ) //reset没在敏感表中。在时钟上升沿和复位信号同时有效，才可以复位。

begin

if ( !reset )

q <= 0;

else

q<= d ;

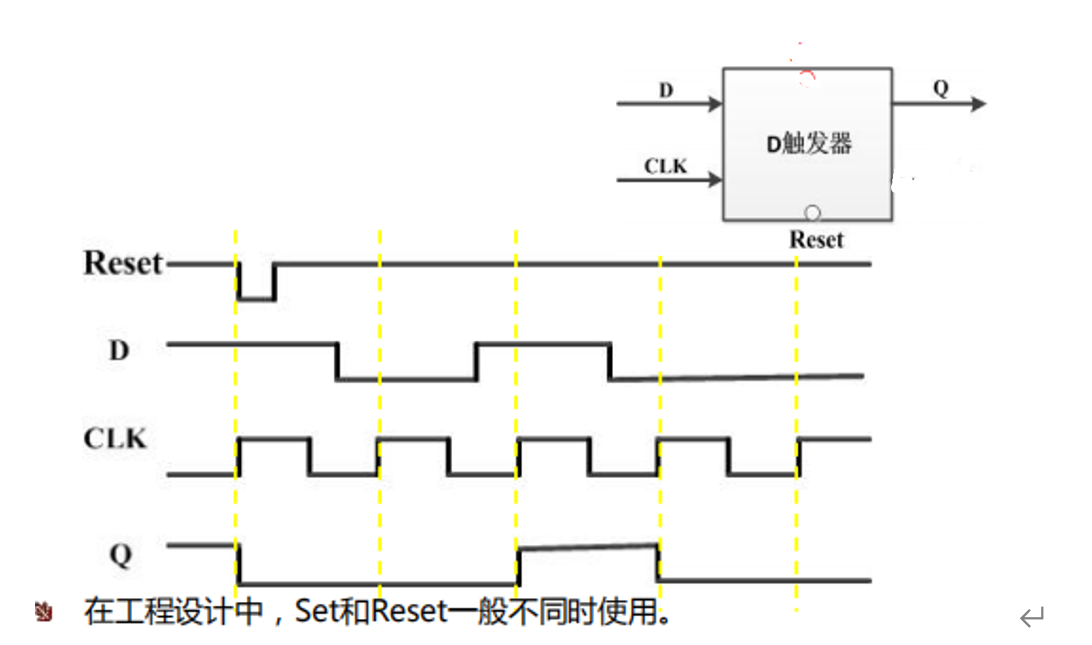
end 

图 3.6 带同步复位端的D触发器模块图和波形图

表 3.6 带同步复位端的D触发器真值表

|  |  |  |  |
| --- | --- | --- | --- |
| **时钟信号CLK** | **复位信号reset’** | **输入D** | **输出Q** |
| **上升沿** | **0** | **X** | **0** |
| **上升沿** | **X** | **1** | **1** |
| **上升沿** | **X** | **0** | **0** |
| **其余时刻** | **X** | **X** | **Qprev** |

### 3.2.4 带使能端的D触发器

如下为带使能端的D触发器，图3.7为其模块示意图。表3.7为其真值表。图3.8为其波形图。

always @(posedge clk) begin

if (en) begin

q <= d;

end

end

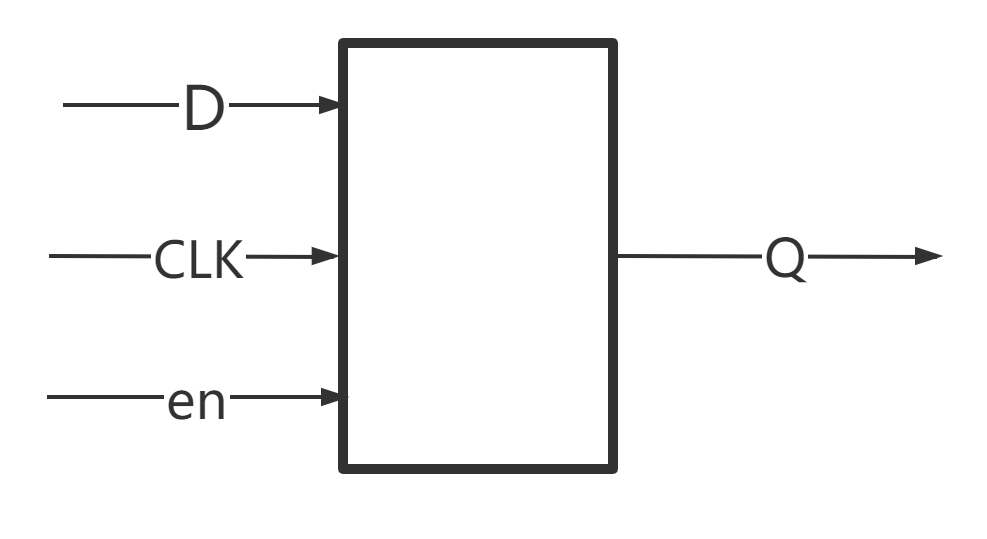


图 3.7 带使能端的D触发器模块图

表 3.7 带使能端的D触发器真值表

|  |  |  |  |
| --- | --- | --- | --- |
| **时钟信号CLK** | **使能信号en** | **输入D** | **输出Q** |
| **上升沿** | **1** | **1** | **1** |
| **上升沿** | **1** | **0** | **0** |
| **其余时刻** | **1** | **X** | **Qprev** |
| **任意时刻** | **0** | **X** | **Qprev** |

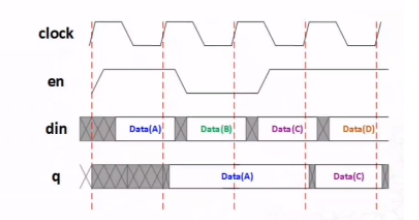


图3.8 带使能端的D触发器的波形图

## 3.3 面向设计的verilog语法

### 3.3.1 模块定义

module模块名(端口1，端口2，端口3，端口4, ………);

endmodule

模块的内容包括I/O说明、内部信号声明、功能定义。

完成组合逻辑电路裁判表决器设计。设计一个比赛裁判表决电路，设比赛有三个裁判，一个主裁判和两个副裁判。只有当两个或两个以上裁判判明成功，并且有一个为主裁判时，表明成功。使用verilog语言进行描述并且仿真，得到正确的波形图。图3.9为表决器电路模块图。

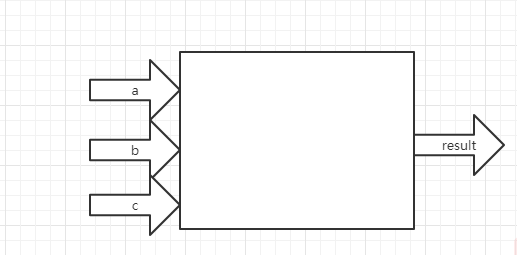


图 3.9 表决器电路模块图

module judge(

input wire a,

input wire b,

input wire c,

output reg result

);

always@(a or b or c)begin

result=(a&b)|(a&c);

end

endmodule

### 3.3.2 信号方向

input,output,inout双向信号

### 3.3.3 常用的两个基本的信号数据类型 reg型、wire型。

（1）wire

表示元件之间的物理连接，缺省值是Z

wire [n-1:0] 数据名1,数据名2,…数据名i;

wire a; //定义了一个一位的wire型数据

wire [7:0] b; //定义了一个八位的wire型数据

1. reg(具有存储功能的一根线，可以看成代码书写规范要求，而不是把它看成D触发器)

always/initial过程的输出中用reg型。reg 型数据的缺省初始值是不定值。

reg rega; //定义了一个一位的名为rega的reg型数据

reg [3:0] regb; //定义了一个四位的名为regb的reg型数据

1. 除了应该定义为reg的都定义为wire

### 3.3.4 常量

图3.10为verilog的四值逻辑,四值逻辑中的四个数值分别为0, 1, X, Z

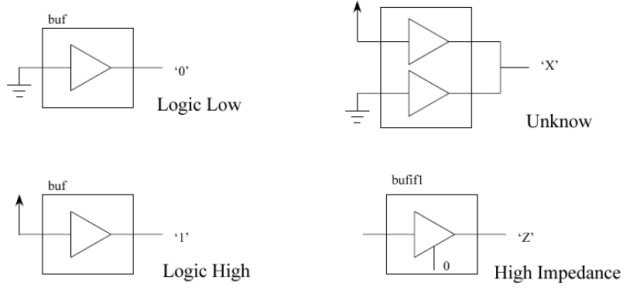


图3.10 verilog四值逻辑

常量的表示方法为：<size>’<base><value>

• Size: 以bit为单位

• Base: b(二进制),o(八进制),d(十进制),h(16进制)

• Value:和进制相应的数值，x, z, ? (x,z不区分大小写)

例

• 16 //只给出一个数字，默认位宽为32位

• 8’d16 //位宽为8的十进制数16

• 8’h10 //位宽为8的十六进制数10h

• 16’b1010\_1011\_1111\_1010 //合法格式

• 32’bx //32位x

• 4'b10x0 //位宽为4的二进制数从低位数起第二位为不定值

• 4'b101z //位宽为4的二进制数从低位数起第一位为高阻值

• 12'dz //位宽为12的十进制数其值为高阻值(第一种表达方式)

• 12'd? //位宽为12的十进制数其值为高阻值(第二种表达方式)

• 8'h4x //位宽为8的十六进制数其低四位值为不定值

### 3.3.5运算符

和C语言一样。（看看即可，用到哪个再学哪个），表3.8为verilog运算符。表3.9为verilog运算符的优先级。

表3.8 verilog运算符

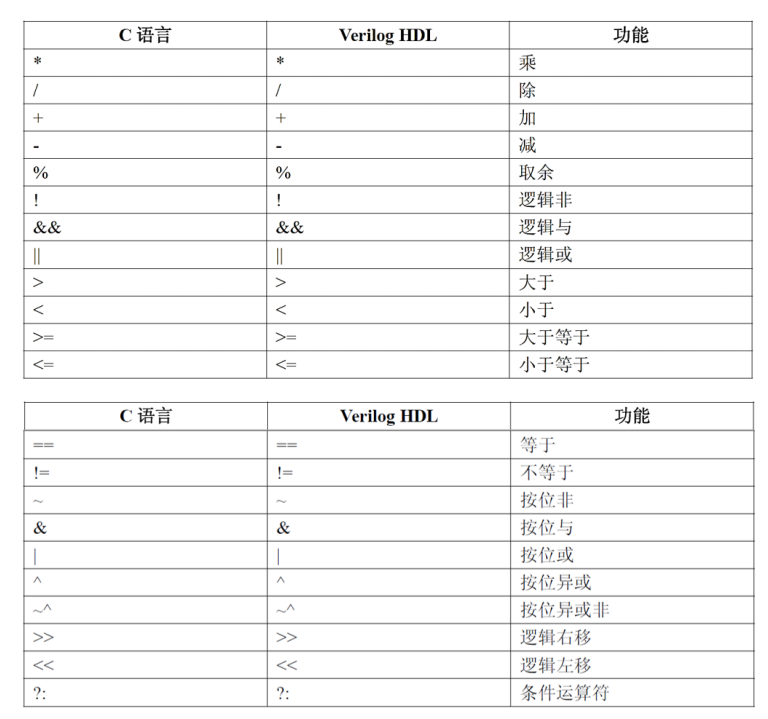
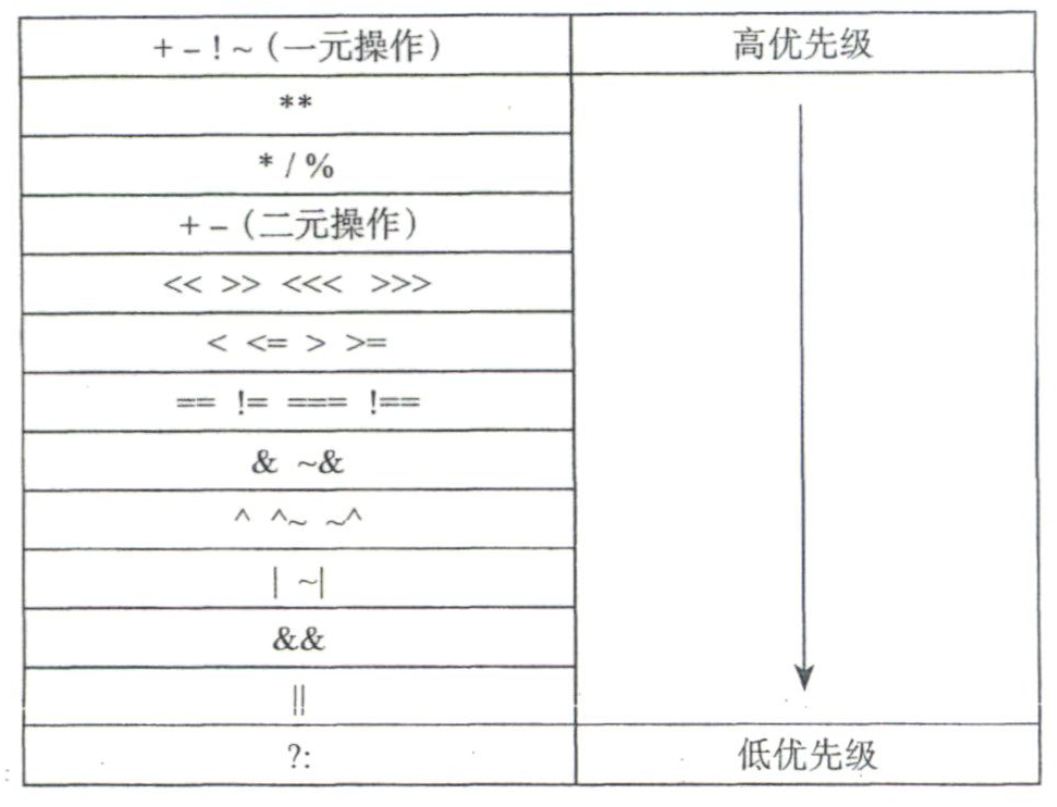


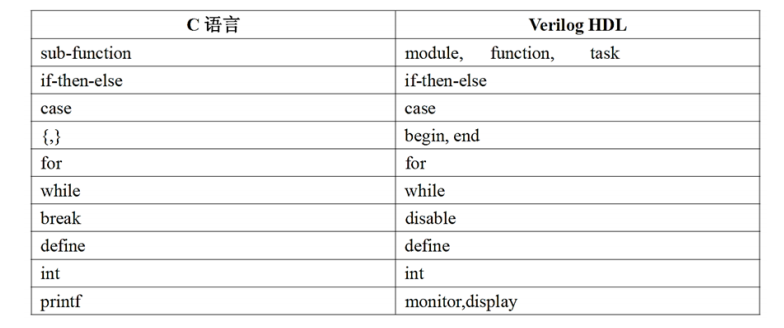
表3.9 verilog运算符优先级



### 3.3.6条件语言

看看即可，用到哪个再学哪个。表3.10为verilog的条件语言。

表3.10 verilog条件语言



### 3.3.7连续赋值语句 assign 用来实现简单的组合逻辑电路。

module block (

input wire a,

input wire b,

output wire c,

output wire d);

wire e=1’b1;//declare and assign

assign c= a | b;

assign d= e?a:b;//二选一

endmodule

要避免出现反馈电路assign a=b+a;

### 3.3.8过程

过程 (always)

• 阻塞赋值(=)

always @(a or b) //由输入信号中任意一个电平发生变化所引起，所有在赋值语句右边出现的信号我们都要放到敏感表中。if,case里面的条件信号也要放到敏感表中。

begin

语句块

(=, if, case)

end

• 非阻塞赋值(<=) //由单个跳变沿所引起

always @(posedge/negedge sig or…)

begin

语句块 (<=, if ,case)

end

(1)两个或更多always模块是同时执行的，模块内部阻塞赋值是顺序执行

(2)always模块描述组合逻辑电路时，用阻塞赋值语句，和c语言一样。

(3)always模块描述时序逻辑电路时，用非阻塞赋值语句，多条非阻塞语句并行执行。计算过程和赋值过程分开进行，先把右边计算过程的全部计算出来，再并行赋值。

(4)在同一个always/initial里面不要同时出现阻塞赋值和非阻塞赋值。

## 3.4 面向测试的Verilog语法

### 3.4.1 测试示意图

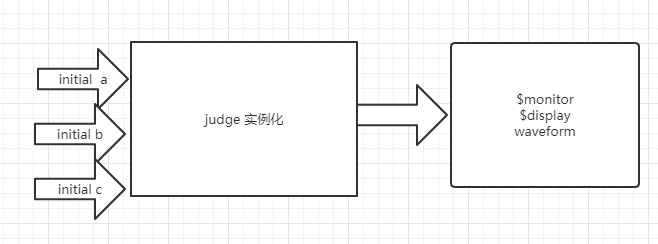


图 3.11 表决电路测试示意图

### 3.4.2完整的测试模块

产生输入信号，观察输出信号

`timescale 1ns/1ps //显示的单位为1ns,精度为1ps,可以想象成时间步长是1ps,时间1ps,1ps这样走。

module tb; //没有输入信号，输出信号。

// 信号定义

// 元件例化

// 产生激励向量

// 显示输出结果

endmodule

### 3.4.3测试举例

module test();

reg a;

reg b;

reg c;

wire result;

judge judge1(.a(a),.b(b),.c(c),.result(result));

initial begin

a=0;

b=0;

c=0;

#10

a=0;

b=0;

c=1;

#10

a=0;

b=1;

c=0;

#10

a=0;

b=1;

c=1;

#10

a=1;

b=0;

c=0;

#10

a=1;

b=0;

c=1;

#10

a=1;

b=1;

c=0;

#10

a=1;

b=1;

c=1;

#10 $finish;

end

initial begin

$monitor($time,",a=%b,b=%b,c=%b,result=%b",a,b,c,result);

end

endmodule

### 3.4.4元件例化

相当于把一个电路例化。可以用在测试代码，也可以用在RTL级代码设计。

reg a;

reg b;

reg c;

wire result;

judge judge1(.a(a),.b(b),.c(c),.result(result));

### 3.4.5产生激励向量

initial:非周期信号

always:周期信号

initial begin

clk = 0;

forever

#10 clk = ~clk ;

end

initial begin

rst = 1;

#15 rst = 0;

#10 rst = 1;

#175 $finish;

end

initial begin

$monitor($time,,

“%b %b %b“, rst,clk,dout);

end

### 3.4.6 显示输出结果

(1)特殊符号$－系统任务和函数

$<标识符> 表示Verilog 的系统任务和函数

常用的系统任务和函数有下面几种：

– $time //当前的仿真时间

– $display, $monitor //显示和监视信号值的变化

– $stop //暂停仿真

– $finish //结束仿真

1. 观察波形

### 3.4.7特殊符号#

在实际的物理电路中，一般是不允许使用延时语句来描述电路的时间特性。但在仿真的时候我们需要描述电路的时间特性和时序特性。例如Clock信号周期，频率。

#特殊符号 “#” 表示延迟 ，用于过程赋值语句里的延迟

initial begin

#10 rst=1; //10个时间单位之后，rst变为1

end

### 3.4.8 “ ` ”表示编译引导语句

用于指导仿真编译器在编译时采取一些特殊处理。

常用的编译引导有：

• `define 宏定义 `define RegAddrBus 4:0

• `include 包含文件 `include"defines.v"

• `timescale 时间单位/时间精度 `timescale 1ns / 1ps ,1ns是时间单位，1ps是时间精度