Verilog HDL语言简介

2020年秋

内容大纲

- Verilog语言初步
- Verilog语法与要素
- Verilog行为语句
- Verilog设计层次与风格
- Verilog有限状态机设计

VERILOG设计初步

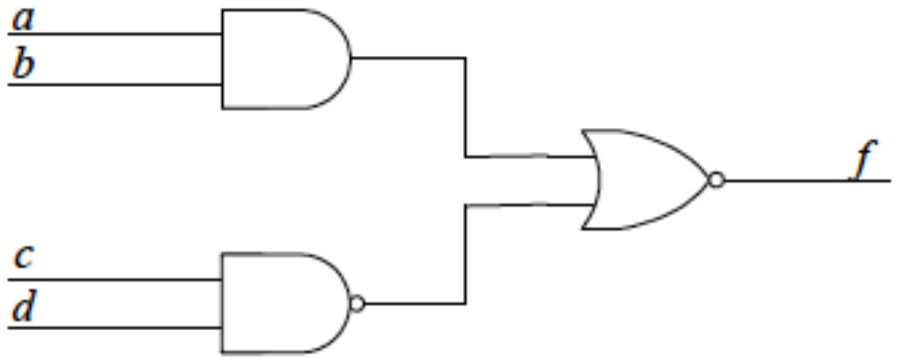
Verilog设计初步

- Verilog简介
- Verilog模块的结构
- Verilog基本组合电路设计
- Verilog基本时序电路设计

Verilog历史

- Verilog语言是1983年由GDA（Gateway Design Automation）公司的Phil Moorby首创的，之后Moorby又设计了Verilog-XL仿真器，Verilog-XL仿真器大获成功，也使得Verilog语言得到推广使用。
- 1989年，Cadence收购了GDA，1990年，Cadence公开发表了Verilog HDL，并成立了OVI组织专门负责Verilog HDL的发展。
- Verilog于1995年成为IEEE标准，称为IEEE Standard 1364-1995（Verilog-1995）
- IEEE“1364-2001”标准（Verilog-2001）也获得通过，多数综合器、仿真器都已支持Verilog-2001标准

Verilog模块结构(p001.v)




```
module aoi(a,b,c,d,f);
/* 模块名为aoi，端口列表a，b，c，d，f */
input a,b,c,d;                    //模块的输入端口为a，b，c，d
output f;                         //模块的输出端口为f
wire a,b,c,d,f;                   //定义信号的数据类型
assign f=~((a&b)|(~(c&d)));       //逻辑功能描述
endmodule
```

Verilog模块的结构

- 模块定义
- 输入输出定义
- 类型定义
- 书写自由
- 末尾分号
- 注释：同c++ /* */, //

逻辑功能定义

- 模块中最核心的部分是逻辑功能定义。
- 定义逻辑功能的几种基本方法:
  - （1）用assign持续赋值语句定义：assign语句多用于组合逻辑的赋值，称为持续赋值方式。
  - （2）用always过程块定义：always过程语句既可以用来描述组合电路，也可以描述时序电路。
  - （3）调用元件（元件例化）：调用元件的方法类似于在电路图输入方式下调入图形符号来完成设计，这种方法侧重于电路的结构描述。

Verilog模块的模板

- module <顶层模块名> (<输入输出端口列表>);
- output 输出端口列表; //输出端口声明
- input 输入端口列表; //输入端口声明
- /*定义数据，信号的类型，函数声明*/
- reg 信号名;
- //逻辑功能定义
- assign <结果信号名> = <表达式>; //使用assign语句定义逻辑功能
- //用always块描述逻辑功能

- always @ (<敏感信号表达式>)
- begin
- //过程赋值
- //if-else，case语句
- //while，repeat，for循环语句
- //task，function调用
- end
- //调用其他模块
- <调用模块名module_name > <例化模块名> (<端口列表port_list >);
- //门元件例化
- 门元件关键字 <例化门元件名> (<端口列表port_list>);
- endmodule

Verilog基本组合电路设计

三人表决电路的Verilog描述 p005.v

```
module vote(a,b,c,f);              //模块名与端口列表
   input a,b,c;                    //模块的输入端口
   output f;                       //模块的输出端口
   wire a,b,c,f;                   //定义信号的数据类型
   assign f=(a&b)|(a&c)|(b&c);     //逻辑功能描述
endmodule
```

Verilog基本组合电路设计

- 4位二进制加法器的Verilog描述 p006.v

```
module add4_bin(cout,sum,ina,inb,cin);
input cin;
input[3:0] ina,inb;
output[3:0] sum;
output cout;
    assign {cout,sum}=ina+inb+cin;      /*逻辑功能定义*/
endmodule
```

Verilog基本组合电路设计

BCD码加法器 p007.v

```
module add4_bcd(cout,sum,ina,inb,cin);
input cin; input[3:0] ina,inb;
output[3:0] sum; reg[3:0] sum;
output cout; reg cout;
reg[4:0] temp;
always @(ina,inb,cin)                          //always过程语句
    begin temp<=ina+inb+cin;
    if(temp>9)
        {cout,sum}<=temp+6;                    //两重选择的IF语句
    else
        {cout,sum}<=temp;
end
endmodule
```

Verilog基本时序电路设计

- 基本D触发器的Verilog描述 p008.v

```
module dff(q,d,clk);
input d,clk;
output reg q;
always @(posedge clk)
begin
    q<=d;
end
endmodule
```

Verilog基本时序电路设计p009.v

- 带同步清0/同步置1（低电平有效）的D触发器

```
module dff_syn(q,qn,d,clk,set,reset);
input d,clk,set,reset; output reg q,qn;
always @(posedge clk)
begin
    if(~reset) begin q<=1'b0;qn<=1'b1;end
    //同步清0，低电平有效
    else if(~set) begin q<=1'b1;qn<=1'b0;end
    //同步置1，低电平有效
    else begin q<=d; qn<=~d; end
end
endmodule
```

Verilog基本时序电路设计p010.v

- 带异步清0/异步置1（低电平有效）的D触发器

```
module dff_asyn(q,qn,d,clk,set,reset);
input d,clk,set,reset; output reg q,qn;
always @(posedge clk or negedge set or negedge
reset)
begin
    if(~reset) begin q<=1'b0;qn<=1'b1; end
    //异步清0，低电平有效
    else if(~set) begin q<=1'b1;qn<=1'b0; end
    //异步置1，低电平有效
    else begin q<=d;qn<=~d; end
end
endmodule
```

Verilog基本时序电路设计p011.v

- 4位二进制加法计数器

```
module count4(out,reset,clk);
input reset,clk;
output reg[3:0] out;
always @(posedge clk)
begin
    if(reset) out<=0;          //同步复位
    else out<=out+1;           //计数
end
endmodule
```

Verilog基本时序电路设计p012.v

- 十进制加法计数器

```
module count10(cout,qout,reset,clk);
input reset,clk;
output reg[3:0] qout;
output cout;always @(posedge clk)
begin
    if(reset)
        qout<=0;
    else if (qout<9)
        qout<=qout+1;
    else
        qout<=0;
    end
    assign cout = (qout==9)?1:0;
endmodule
```

VERILOG语法与要素

Verilog语法与要素

- Verilog语言要素
- 常量
- 数据类型
- 参数
- 向量
- 运算符

Verilog语言要素

- Verilog程序由符号流构成，符号包括
- 空白符(White Space)：空格，tab，换行，换页
- 注释(Comments)：// /* */
- 操作数(Operators)
- 数字(Numbers)
- 字符串(Strings)
- 标识符(Identifiers)
- 关键字(Keywords)：关键字都是小字，保留字不能被使用

标识符（Identifiers）

- Verilog中的标识符可以是任意一组字母、数字以及符号$和_（下划线）的组合，但标识符的第一个字符必须是字母或者下划线。另外，标识符是区分大小写的。

- count
- COUNT //COUNT与count是不同的
- _A1_d2 //以下划线开头
- R56_68
- FIVE

常量

- 程序运行中，值不能被改变的量称为常量（contants）。
- Verilog的常量主要有如下3种类型

- 整数
- 实数
- 字符串

整数

- 整数按照如下方式书写
- +/-<size>’<base><value>
- 即+/-<位宽>’<进制><数字>
- size为对应二进制数的宽度；base为进制；value是基于进制的数字序列
- 进制有如下4种表示形式
- 二进制b或B
- 十进制d或者D或者缺省
- 十六进制h或者H
- 八进制o或者O

整数的例子

- 8'b11000101
- 8'hd5
- 5'O27
- 4'D2
- 4'B1x_01
- 5'Hx　　xxxxx
- 4'hZ　　zzzz

实数

- 实数有下面两种表示方法
- 十进制表示法
- 2.0
- 0.1
- 2. //这个非法
- 科学计数法
- 43_5.1e2 //43510.0
- 9.6E2 //960.0
- 5E-4 //0.0005

字符串

- 字符串是双引号内的字符序列
- 字符串不能分成多行书写。
- 字符串: “ INTERNAL ERROR”
  - 字符串的作用主要是用于仿真时，显示一些相关的信息，或者指定显示的格式

数据类型

- 数据类型（Data Type）是用来表示数字电路中的物理连线，数据存储和传输单元等物理量的
- Verilog有下面四种基本的逻辑状态
  - 0：低电平，逻辑0，或者逻辑非
  - 1：高电平，逻辑1，或者真
  - x或X：不确定或者未知的逻辑状态
  - z或Z：高阻态
- Verilog中的所有数据类型都在上述4类逻辑中取值，其中x和z都不区分大小写，也就是说0x1z与值0X1Z是等同的

数据类型

- Verilog中的变量分为如下两种数据类型：
- net型
- variable型
- net型中常用的有wire, tri;
- variable型包括reg, integer等;

net型

- net型数据相当于硬件电路中的各种物理连线，其特点是输出的值紧跟输入值的变化而变化。对连线型有两种驱动方式，一种方式在结构描述中将其连到一个门原件或模块的输出端；另一种方式是用持续赋值语句assign对其进行赋值
- wire是最常用的net型变量
- wire型变量的定义格式如下：
- wire数据名1，数据名2，……数据名n;
- 例如：wire a,b;　　//定义了两个wire型变量a和b

Variable型

- variable型变量必须放在过程语句（如initial, always)中，通过过程赋值语句赋值；在always, inital等过程块内被赋值的信号也必须定义成variable型
  - 注意：variable型变量并不意味着一定对应硬件上的一个触发器或寄存器等存储元件，在综合器进行综合的时候，variable型变量会根据具体情况来确定是映射成连线还是映射为触发器或者寄存器
- reg 型变量是最常用的一种variable型变量
- reg 数据名1，数据名2，……数据名n;
- 例如：reg a,b;　//定义了两个reg型变量a和b

参数

- 在Verilog语言中，使用参数parameter来定义符号常量，即用parameter来定义一个标识符代表一个常量。参数常用来定义时延和变量的宽度。
- parameter 参数名1 = 表达式1，参数名2=表达式2
- 例如
- parameter SEL=8, CODE=8s'ha3;

采用参数定义的数据比较器
p013.v

```
module compare_w(a,b,larger,equal,less);
parameter SIZE=6;                    //参数定义
input[SIZE-1:0] a,b;
output larger,equal,less;
wire larger,equal,less;
    assign larger=(a>b);
    assign equal=(a==b);
    assign less=(a<b);
endmodule
```

采用参数定义的加法器
p014.v

```
module add_w(a,b,sum);
parameter MSB=15;          //参数定义
input[MSB:0] a,b;
output[MSB+1:0] sum;
    assign sum=a+b;
endmodule
```

采用参数定义的二进制计数器

```
module count_w(en,clk,reset,out);
input clk,reset,en;
parameter WIDTH=8;                          //参数定义
output[WIDTH-1:0] out;
reg[WIDTH-1:0] out;
always @(posedge clk or negedge reset)
    if(!reset) out=0;
    else if(en) out=out+1;
endmodule
```

向量

- 标量与向量
- 宽度为1位的变量称为标量，如果在变量声明中没有指定位宽，则默认为标量（1位）。举例如下：
  - wire a; //a为标量
  - reg clk; //clk为标量reg型变量
- n线宽大于1位的变量（包括net型和variable型）称为向量（vector）。向量的宽度用下面的形式定义：
  - [msb : lsb]
- 比如：
  - wire[3:0] bus; //4位的总线

位选择和域选择

- 在表达式中可任意选中向量中的一位或相邻几位，分别称为位选择和域选择，例如：
- A=mybyte[6]; //位选择
- B=mybyte[5:2]; //域选择
- 再比如：
- reg[7:0] a,b; reg[3:0] c; reg d;
- d=a[7]&b[7]; //位选择
- c=a[7:4]+b[3:0]; //域选择

运算符（1）

- 1 算术运算符（Arithmetic operators）
  - 常用的算术运算符包括：
  - + 加
  - - 减
  - * 乘
  - /
- 2．逻辑运算符（Logical operators）
  - && 逻辑与
  - || 逻辑或
  - ! 逻辑非

运算符（2）

- 3. 位运算符（Bitwise operators）
  - 位运算，即将两个操作数按对应位分别进行逻辑
  - 运算。
  - ~ 按位取反
  - & 按位与
  - | 按位或
  - ^ 按位异或
  - ^~,~^按位同或（符号^~与~^是等价的）
- 4. 关系运算符（Relational operators）
  - < 小于
  - <= 小于或等于
  - > 大于
  - >= 大于或等于

运算符（3）

- 5．等式运算符（Equality Operators）
  - == 等于
  - != 不等于
  - === 全等
  - !== 不全等
- 6．缩位运算符（Reduction operators）
  - & 与
  - ~& 与非
  - | 或
  - ~| 或非
  - ^ 异或
  - ^~,~^ 同或
- 7．移位运算符（shift operators）
  - >> 右移
  - << 左移

运算符（4）

- 8．条件运算符（conditional operators）
- ？ ：
  - 三目运算符，其定义方式如下：signal=condition?true_expression:false_expression;
  - 即：信号=条件?表达式1:表达式2;当条件成立时，信号取表达式1的值，反之取表达式2的值。
- 9．位拼接运算符（concatenation operators）
- { }
  - 该运算符将两个或多个信号的某些位拼接起来。{信号1的某几位，信号2的某几位，……，信号n的某几位}

运算符的优先级

运 算 符	优先级
! ~	高优先级
* / %	
+ -	
<< >>	
< <= > >=	
== != === !==	
& ~&	
^ ^~	
| ~|	
&&	
||	
?:	低优先级

VERILOG行为语句

Verilog行为语句

- 过程语句（initial，always）
- 块语句（begin-end，fork-join）
- 赋值语句（assign、=、<=）
- 条件语句（if-else、case、casez、casex）
- 循环语句（for, forever, repeat, while)
- 编译指导语句（`define, `include, `ifdef, `else, `endif)
- 任务（task）和函数（function）
- 顺序执行与并发执行

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	√

过程语句

- initial
- always
- 在一个模块（module）中，使用initial和always语句的次数是不受限制的。initial语句场用于仿真中的初始化，initial过程块中的语句仅执行一次；always块内的语句则是不断重复执行的。

always过程语句使用模板

- always@(<敏感信号表达式event-expression>)
- begin
- //过程赋值
- //if-else, case, casex, casez选择语句
- //while, repeat, for循环
- //task, function调用
- end
- “always”过程语句通常是带有触发条件的，触发条件写在敏感信号表达式中，只有当触发条件满足时，其后的”begin-end”块语句才能被执行。

敏感信号表达式

- 若有两个或两个以上信号时，使用or连接
- 例如：
- @(a)
- @(a or b)
- @(posedge clock)
- @(negedge clock)
- @(posedge clk or negedge reset)

敏感信号列表举例p016.v

- 4选1数据选择器

```
module mux4_1(out,in0,in1,in2,in3,sel);
output out;
input in0,in1,in2,in3;
input[1:0] sel; reg out;
always @(in0 or in1 or in2 or in3 or sel)        //敏感信号列表
    case(sel)
        2'b00: out=in0;
        2'b01: out=in1;
        2'b10: out=in2;
        2'b11: out=in3;
        default: out=2'bx;
    endcase
endmodule
```

posedge和negedge关键字p017.v

- 对于时序电路，事件通常是由时钟边沿触发的，为表达边沿这个概念，Verilog提供了posedge和negedge关键字来描述
- 同步置数、同步清零的计数器

```
module count(out,data,load,reset,clk);
output[7:0] out; input[7:0] data;
input load,clk,reset; reg[7:0] out;
always @(posedge clk)                    //clk上升沿触发
begin
    if(!reset) out=8'h00;                //同步清0，低电平有效
    else if(load) out=data;              //同步预置
    else out=out+1;                      //计数
end
endmodule
```

块语句

- 块语句是由块标识符begin-end或fork-join界定的一组语句，当块语句只包含一条语句时，块标识符可以缺省
- begin-end串行块中的语句按串行方式顺序执行，比如
  - begin
  - regb=rega;
  - regc=regb;
  - end
- 由于begin-end块内的语句顺序执行，在最后，将regb,regc的值都更新为rega的值，该begin-end块执行完后，regb,regc的值是相同的。

赋值语句

- 持续赋值语句（continuous assignments）

- assign为持续赋值语句，主要用于对wire型变量的赋值。
- 比如：assign c=a&b;
- 在上面的赋值中，a,b,c三个变量皆为wire型变量，a和b信号的任何变化，都将随时反映到c上来

过程赋值语句

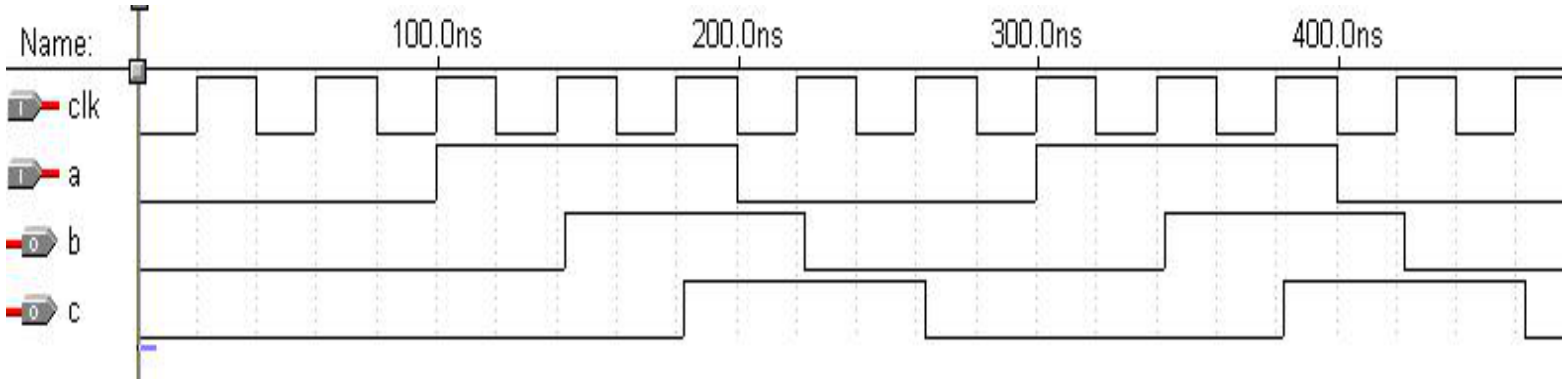
- 过程赋值语句多用于对reg型变量进行赋值。
- （1）非阻塞（non_blocking）赋值方式
- 赋值符号为”<=“，比如：b<=a; 非阻塞赋值在整个过程块结束时才完成赋值操作，即b的值并不是立刻就改变的。
- （2）阻塞（blocking）赋值方式
- 赋值符号为”=”，如：b=a; 阻塞赋值语句在该语句结束时就立即完成赋值操作，即b的值在该条语句结束后立刻改变。如果在一个块语句中，有多条阻塞赋值语句，那么在前面的赋值语句没有完成之前，后面的语句不能被执行，仿佛被阻塞了（blocking）一样，因此被称为是阻塞赋值方式。

阻塞赋值与非阻塞赋值

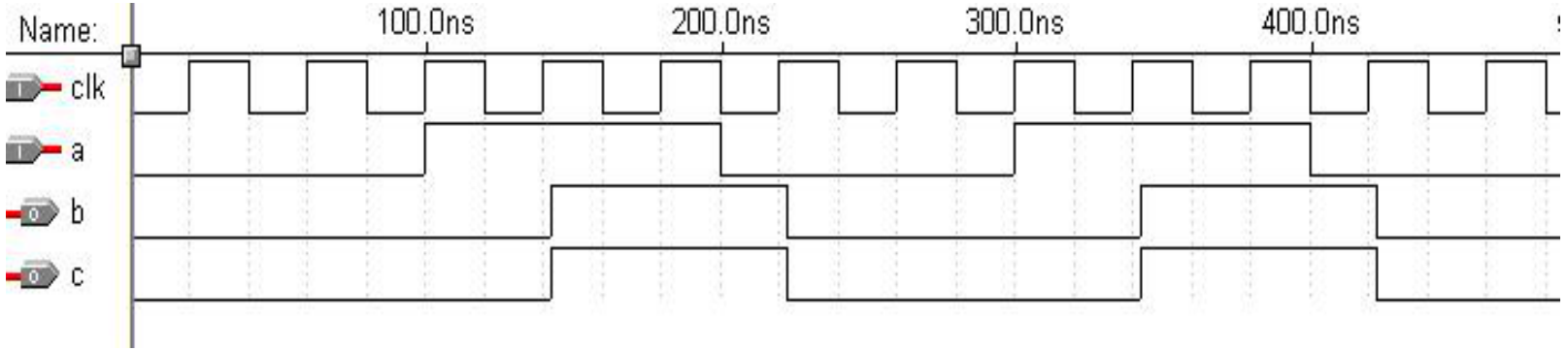
```
module
non_block(c,b,a,clk);
   output c,b;
   input clk,a;
   reg c,b;
   always @(posedge clk)
   begin
      b<=a;
      c<=b;
   end
endmodule
```

```
module block(c,b,a,clk);
   output c,b;
   input clk,a;
   reg c,b;
   always @(posedge clk)
   begin
      b=a;
      c=b;
   end
endmodule
```

阻塞赋值与非阻塞赋值的仿真波形

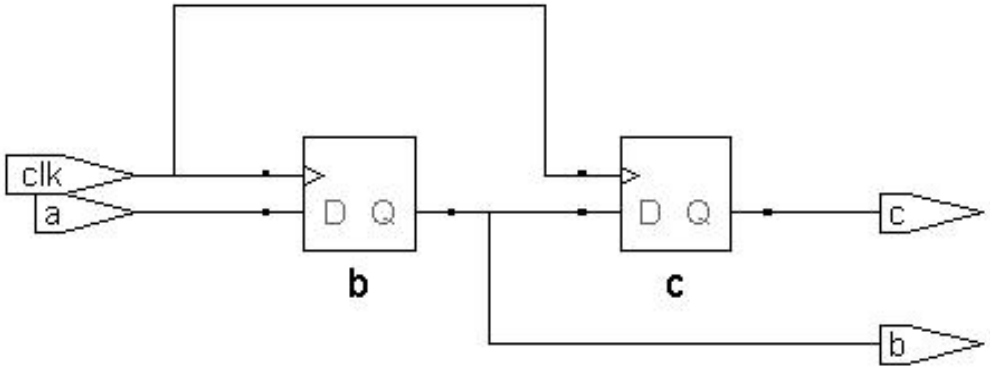


非阻塞赋值仿真波形图

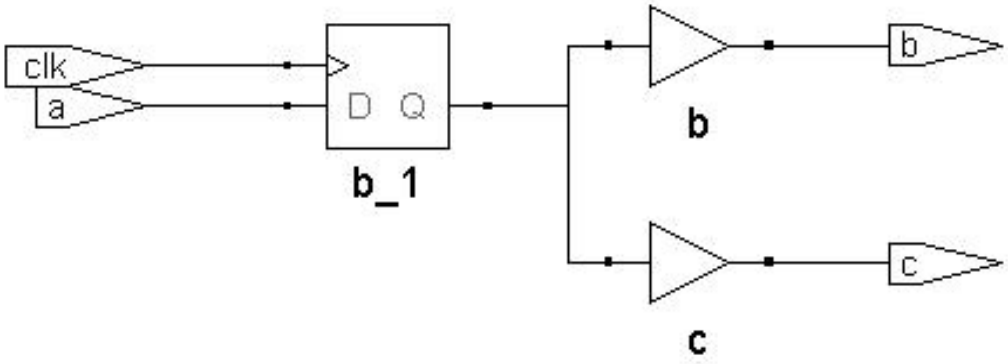


阻塞赋值仿真波形图

阻塞赋值和非阻塞赋值的综合结果



非阻塞赋值综合结果



阻塞赋值综合结果

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;

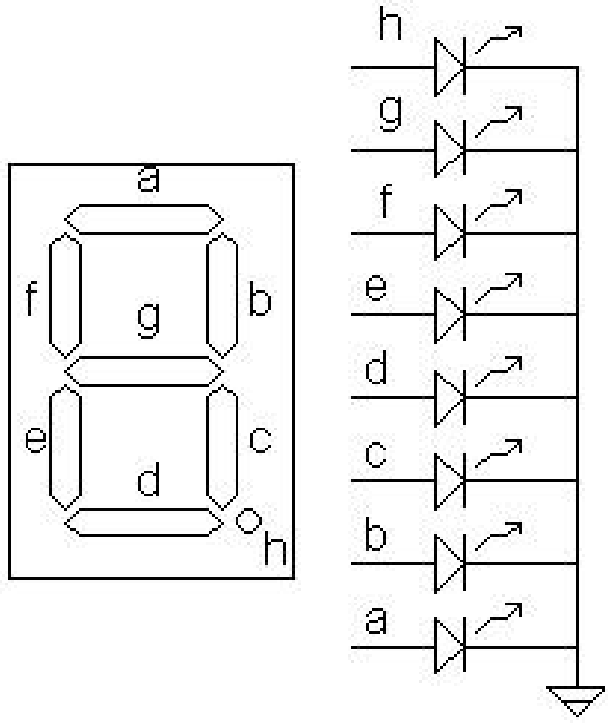
case语句

- 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

BCD码—七段数码管显示译码器

```
module decode4_7(decodeout,indec);
output[6:0] decodeout;
input[3:0] indec; reg[6:0] decodeout;
always @(indec)
begin case(indec) //用case语句进行译码
    4'd0:decodeout=7'b1111110;
    4'd1:decodeout=7'b0110000;
    4'd2:decodeout=7'b1101101;
    4'd3:decodeout=7'b1111001;
    4'd4:decodeout=7'b0110011;
    4'd5:decodeout=7'b1011011;
    4'd6:decodeout=7'b1011111;
    4'd7:decodeout=7'b1110000;
    4'd8:decodeout=7'b1111111;
    4'd9:decodeout=7'b1111011;
    default: decodeout=7'bx;
endcase end
endmodule
```

p018.v



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 @(vote)
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
```

p019.v

repeat语句

- repeat语句的使用格式为：
- repeat（循环次数表达式） 语句；
- 或者
- repeat（循环次数表达式）
- begin
- ……
- end

用repeat实现8位二进制数乘法

```
module mult_repeat(outcome, a, b);
parameter size=8; input[size:1] a,b;
output[2*size:1] outcome;
reg[2*size:1] temp_a,outcome;
reg[size:1] temp_b;
always @(a or b)
    begin
        outcome=0; temp_a=a; temp_b=b;
        repeat(size)                          //repeat语句，size为循环次数
        begin
            if(temp_b[1])      //如果temp_b的最低位为1，就执行下面的加法
            outcome=outcome +temp_a;
            temp_a=temp_a<<1;                 //操作数a左移一位
            temp_b=temp_b>>1;                 //操作数b右移一位
        end
    end
endmodule
```

p020.v

编译指导语句

- Verilog允许程序中使用特殊的编译向导（Compiler Directives）语句，在编译的时候，通常先对这些向导语句进行“预处理”，然后再将预处理的结果和源程序一起进行编译。
- 向导语句以符号 ` 开头，以区别于其它语句。Verilog提供了十几条编程向导语句，如 `define,`ifdef,`else,`endif,`restall。比较常用的有 `define,`include，和`ifdef,`else,`endif等
- 宏替换：`define
- `include 1)只能指定一个文件；2）可以出现在源文件任意位置；3）允许多重包含

任务和函数

- 任务（task）
- 任务定义格式：
- 　task<任务名>;　　//注意没有端口列表
- 　端口及数据类型声明语句
- 　其它语句
- 　endtask
- 任务调用的格式为：
- 　<任务名>（端口1，端口2，端口3……）
- 需要注意的是：任务调用时和定义时的端口变量应该是一一对应的

使用任务时需要注意

- 任务的定义与调用必须在一个module模块内。
- 定义任务时，没有端口名列表，但需要紧接着进行输入输出端口和数据类型的说明
- 当任务被调用时，任务被激活。任务的调用和模块调用一样通过任务名调用实现，调用时，需要列出端口名列表，端口名的排序和类型必须与任务定义中的相一致。
- 一个任务可以调用别的任务和函数，可以调用的任务和函数个数不限

函数

- 函数的目的是返回一个值，以用于表达式计算
- 函数的定义格式：
- function <返回值位宽或者类型说明> 函数名；
- 端口说明
- 局部变量定义
- 其它语句
- endfunction
- <返回值位宽或者类型说明>是一个可选项，如果缺省，则返回值为1位寄存器类型的数据

函数举例p021.v

```
function[7:0] get0;
input[7:0] x; reg[7:0] count;
integer i;
begin
    count=0;
    for (i=0;i<=7;i=i+1)
        if(x[i]=1'b0) count=count+1;
    get0=count;
end
endfunction
```

- 上面的get0函数循环核对输入数据x的每一位，计算出x
- 中0的个数，并返回一个适当的值。

使用函数时需要注意

- 函数的定义与调用必须在一个module模块内
- 函数只允许有输入变量且必须至少有一个输入变量，输出变量有函数名本身承担，在定义函数时，需对函数名说明其类型和位宽。
- 定义函数时，没有端口名列表，但是调用函数时，需列出端口名列表，端口名的排序和类型必须与定义时的相一致。这一点与任务相同。
- 函数可以出现在持续赋值assign的右端表达式中。
- 函数不能调用任务，但是任务可以调用别的任务和函数，且调用任务和函数的个数不受限制

任务与函数的比较

	任务（task）	函数（function）
输入与输出	可有任意个各种类型的参数	至少有一个输入，不能将inout类型作为输出
调用	任务只可在过程语句中调用，不能在连续赋值语句assign中调用	函数可作为表达式中的一个操作数来调用，在过程赋值和连续赋值语句中均可以调用
定时事件控制（#，@和wait）	任务可以包含定时和事件控制语句	函数不能包含这些语句
调用其他任务和函数	任务可以调用其它任务和函数	函数可调用其它函数，但不可以调用其它任务
返回值	任务不向表达式返回值	函数向调用它的表达式返回一个值

顺序执行与并发执行

- 两个或者多个always过程块，assign持续赋值语句，实例元件调用等操作都是同时执行的。
- 在always模块内部，其语句如果是非阻塞赋值，也是并发执行的；而如果是阻塞赋值，则语句是按照指定的顺序执行的，语句的书写顺序对程序执行结果有着直接的影响。

顺序执行的例子

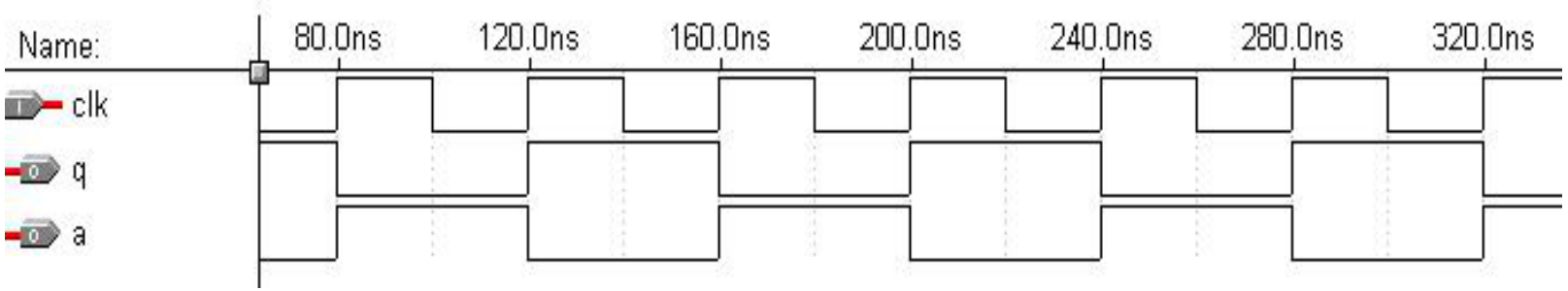
- 顺序执行模块1

```
module serial1(q,a,clk);
output q,a;
input clk;
reg q,a;
always @(posedge clk)
    begin
        q=~q;
        a=~q;
    end
endmodule
```

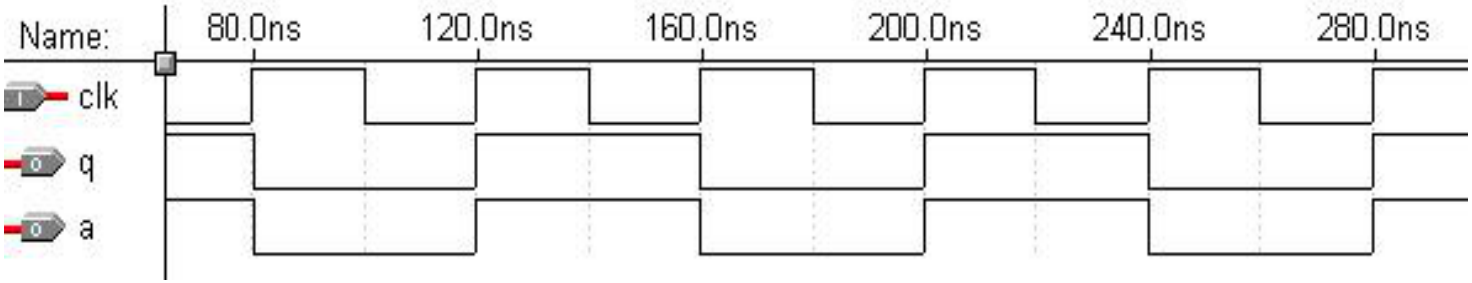
- 顺序执行模块2

```
module serial2(q,a,clk);
output q,a;
input clk;
reg q,a;
always@(posedge clk)
    begin
        a=~q;
        q=~q;
    end
endmodule
```

顺序执行的时序效果

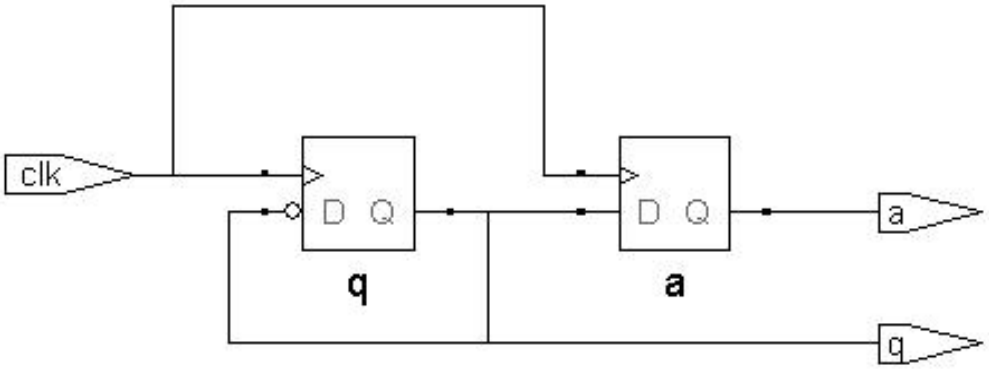


顺序执行模块1仿真波形图

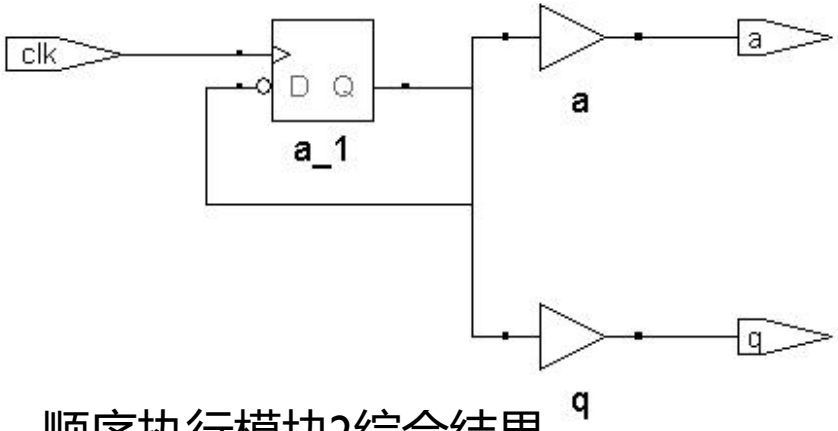


顺序执行模块2仿真波形图

顺序执行模块的综合结果



顺序执行模块1综合结果



顺序执行模块2综合结果

VERILOG设计层次与风格

Verilog设计的层次与风格

- Verilog设计的层次
- 门级结构描述
- 行为描述
- 数据流描述
- 不同描述风格的设计
- 多层次结构电路的设计
- 基本组合电路设计
- 基本时序电路设计
- 三态逻辑设计

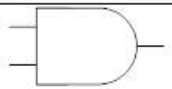
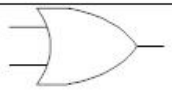
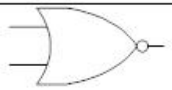
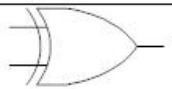
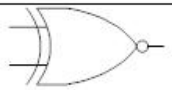
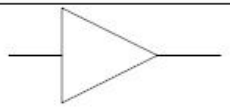
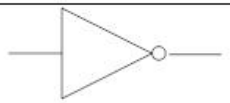
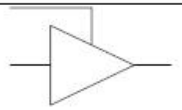
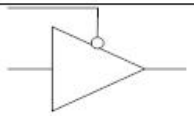
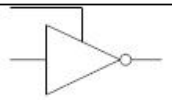
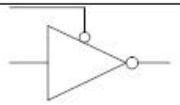
Verilog设计的描述风格

- 结构（Structural）描述
- 行为（Behavioral）描述
- 数据流（Data Flow）描述

结构描述

- 在Verilog程序中可通过如下方式描述电路的结构
- 调用Verilog内置门元件（门级结构描述）
- 调用开关机元件（晶体管级结构描述）
- 用户自定义元件UDP（也在门级）

Verilog的内置门元件

类　　别	关　键　字	符号示意图	门　名　称
多输入门	and		与门
	nand		与非门
	or		或门
	nor		或非门
	xor		异或门
	xnor		异或非门
多输出门	buf		缓冲器
	not		非门
三态门	bufif1		高电平使能三态缓冲器
	buif0		低电平使能三态缓冲器
	notif1		高电平使能三态非门
	notif0		低电平使能三态非门

门元件的调用

- 调用门元件的格式为：
- 门元件名字<例化的门名字>(<端口列表>)
- 其中普通门的端口列表按下面的顺序列出：
- （输出，输入1，输入2，输出3，……）；
- 比如
- and a1(out,in1,in2,in3);
- 对于三态门，则按如下顺序列出输入输出端口：
- （输出，输入，使能控制端）；
- 比如：
- bufif1 mytri1(out,in,enable); //高电平使能的三态门

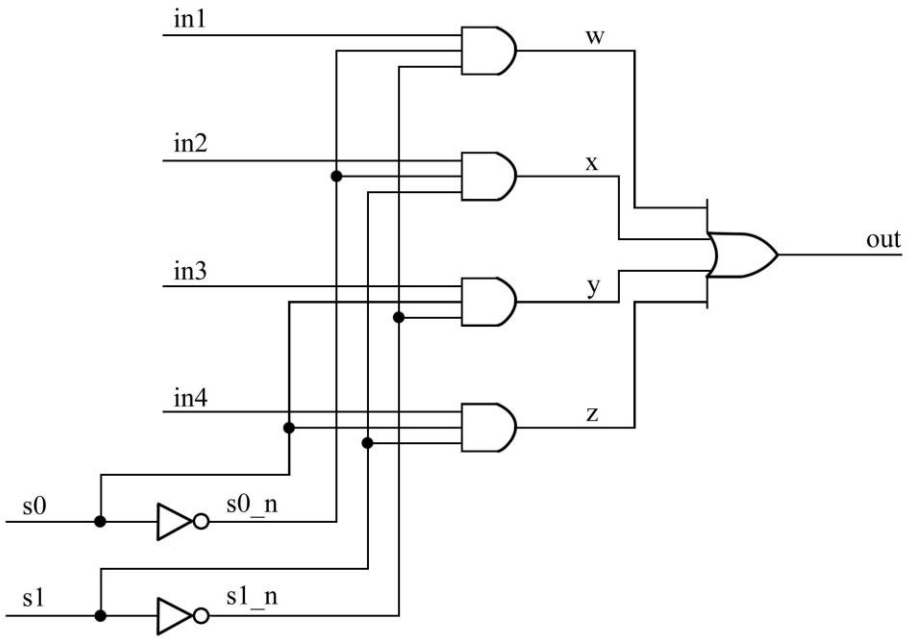
门元件的调用

- 对于buf和not两种元件的调用，需注意的是：它们允许有多个输出，但只能有一个输入。比如：
- not N1 （out1,out2,in)
- buf B1(out1,out2,out3,in);

调用门元件实现的4选1 MUX

```
module mux4_1a(out,in1,in2,in3,in4,s0,s1);
input in1,in2,in3,in4,s0,s1; output out;
wire s0_n,s1_n,w,x,y,z;
    not (sel0_n,s0),(s1_n,s1);
    and (w,in1,s0_n,s1_n),(x,in2,s0
(y,in3,s0,s1_n),(z,in4,s0,s1);
    or (out,w,x,y,z);
endmodule
```

p022.v



行为描述

- 就是针对设计实体的数学模型的描述，其抽象程度远高于结构描述方式。行为描述类似于高级编程语言，单描述一个设计实体的行为时，无需知道具体电路的结构，只需要描述清楚输入与输出信号的行为，而不需要花费更多的精力关注设计功能的门级实现

用case语句描述的4选1 MUX

p023.v

```
module mux4_1b(out,in1,in2,in3,in4,s0,s1);
input in1,in2,in3,in4,s0,s1;
output reg out;
always@(*) //使用通配符
    case({s0,s1})
        2'b00:out=in1;
        2'b01:out=in2;
        2'b10:out=in3;
        2'b11:out=in4;
        default:out=2'bx;
    endcase
endmodule
```

采用行为描述方式时需注意

- 用行为描述模式设计电路，可以降低设计难度。行为描述只需表示输入和输出之间的关系，不需要包含任何结构方面的信息。设计者只需写出源程序，而挑选电路方案的工作由EDA软件自动完成。
- 在电路的规模较大或者需要描述复杂的逻辑关系时，应首先考虑用行为描述方式设计电路，如果设计的结果不能满足资源占有率的要求，则应该变描述方式。

数据流描述

- 数据流描述方式主要使用持续赋值语句，多用于描述组合逻辑电路，其格式为：
- assign LHS_NET = RHS_expression;
- 右边表达式中的操作无论何时发生变化，都会引起表达式值的重新计算，并将重新计算后的值赋予左边表达式的net型变量

数据流描述的4选1 MUX

```
module mux4_1c(out,in1,in2,in3,in4,s0,s1);
input in1,in2,in3,in4,s0,s1;
output out;
    assign out=(in1 & ~s0 & ~s1)|(in2 & ~s0 & s1)|
    (in3& s0 & ~s1)|(in4 & s0 & s1);
endmodule
```

p024.v

数据流描述

- 用数据流描述模式设计电路与用传统的逻辑方程设计电路很相似。设计中只要有了布尔代数表达式就很容易将它用数据流方式表达出来。表达方式是用Verilog中的逻辑运算符置换布尔逻辑运算符即可。
- assign F=(a&b)|(~(c&d))。

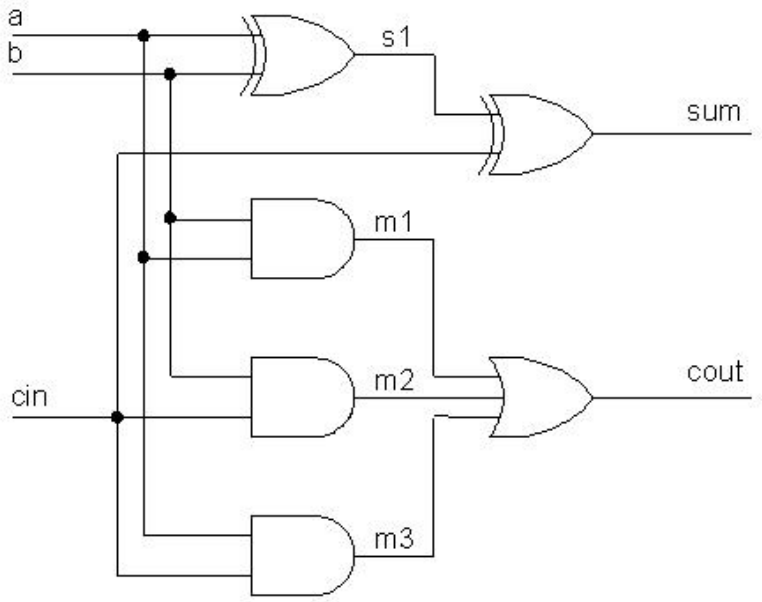
不同描述风格的设计

- 对于设计者而言，采用的描述级别越高，设计越容易；对于综合器而言，行为级的描述为综合器的优化提供了更大的空间，较之门级结构描述更能发挥综合器的性能，所以在电路设计中，除非一些关键路径的设计采用门级结构描述外，一般更多采用行为建模方式。

调用门元件实现的1位全加器

```
module full_add1(a, b, cin, sum, cout);
input a, b, cin;
output sum, cout;
wire s1,m1, m2, m3;
    and (m1, a, b),
        (m2, b, cin),
        (m3, a, cin);
    xor (s1, a, b),
        (sum, s1, cin);
    or (cout, m1, m2, m3);
endmodule
```

p025.v



数据流描述的1位全加器

```
module full_add2(a,b,cin,sum,cout);
input a, b, cin;
output sum, cout;
    assign sum = a ^ b ^ cin;
    assign cout = (a & b ) | (b & cin ) | (cin & a );
endmodule
```

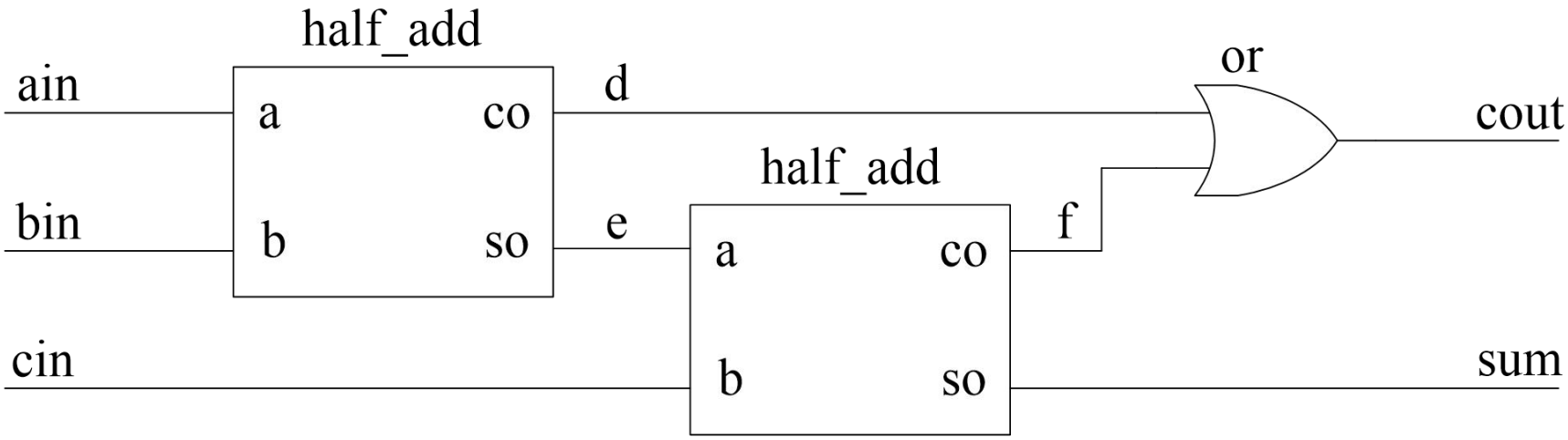
- p026.v

行为描述的1位全加器

```
module full_add3(a,b,cin,sum,cout);
input a,b,cin;
output reg sum,cout;
always @*
//或写为always @(a or b or cin)
    begin
        {cout,sum}=a+b+cin;
    end
endmodule
```

- p027.v

采用层次化方式设计的1位全加器



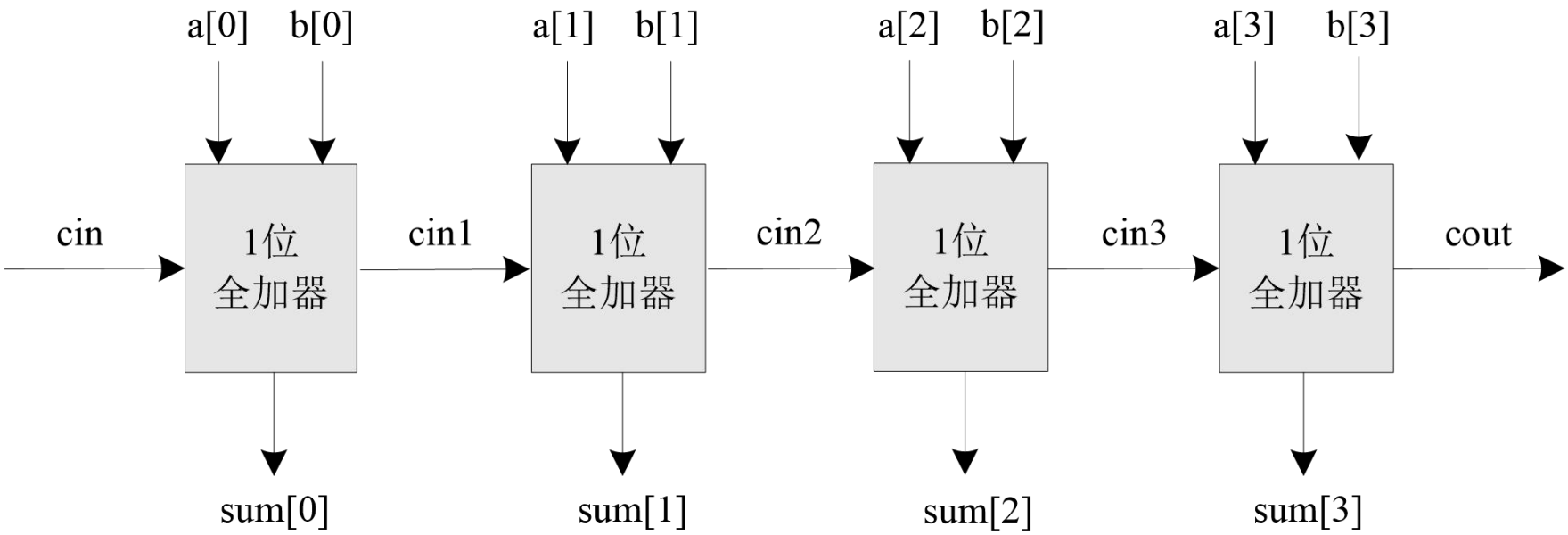
- 两个半加器构成全加器

模块例化方式设计的1位全加器

```
module full_add(ain,bin,cin,sum,cout);
input ain,bin,cin;
output sum,cout;
wire d,e,f; //用于内部连接的节点信号
    half_add u1(ain,bin,e,d);
    //半加器模块调用，采用位置关联方式
    half_add u2(e,cin,sum,f);
or u3(cout,d,f); //或门调用
endmodule
```

```
module half_add(a,b,so,co);
input a,b;
output so,co;
    assign co=a&b;
    assign so=a^b;
endmodule
```

4位加法器设计


a[0]
b[0]
a[1]
b[1]
a[2]
b[2]
a[3]
b[3]
cin
1位
全加器
cin1
1位
全加器
cin2
1位
全加器
cin3
1位
全加器
cout
sum[0]
sum[1]
sum[2]
sum[3]

结构描述的4位级连全加器

```
module add4_1(sum,cout,a,b,cin);
output [3:0] sum;
output cout;
input [3:0] a,b;
input cin;
    full_add1 f0(a[0],b[0],cin,sum[0],cin1);
    full_add1 f1(a[1],b[1],cin1,sum[1],cin2);
    full_add1 f2(a[2],b[2],cin2,sum[2],cin3);
    full_add1 f3(a[3],b[3],cin3,sum[3],cout);
endmodule
```

p029.v

数据流描述的4位加法器

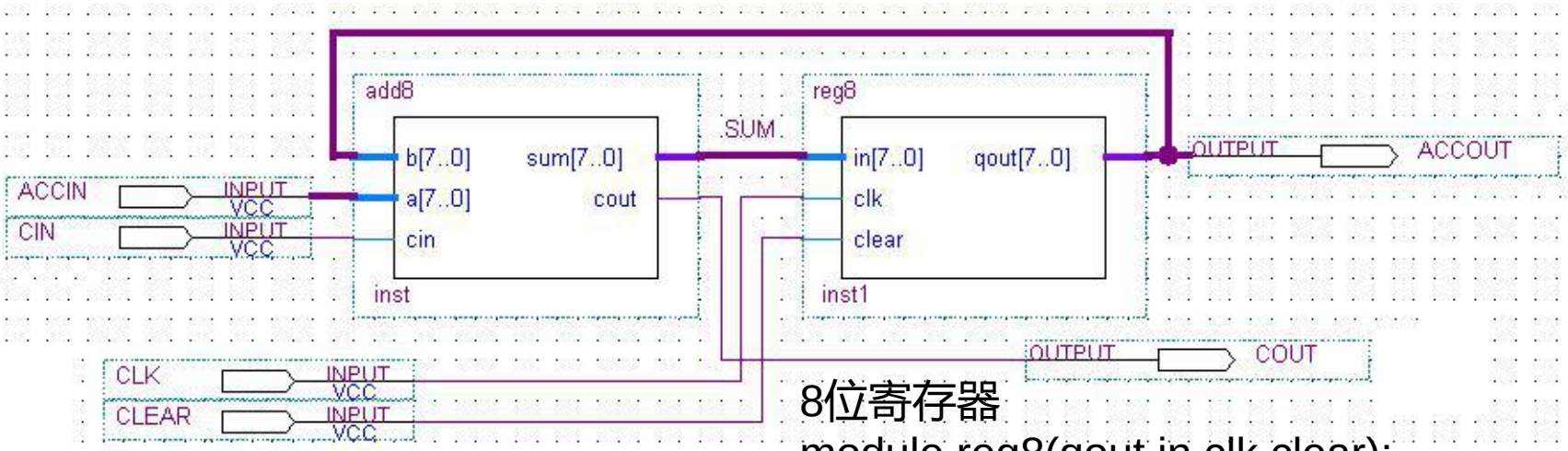
```
module add4_2(cout,sum,a,b,cin);
input cin; input[3:0] a,b;
output[3:0] sum;
output cout;
    assign {cout,sum}=a+b+cin;
endmodule
```

- p030.v

多层次结构的电路设计

- 如果数字系统比较复杂，可采用”top-down”的方法进行设计。首先把系统分为几个模块，每个模块再分为几个子模块，以此类推，直到易于实现为止。这种“Top-Down”的方法能够把复杂的设计分解为许多简单的逻辑来实现，同时也适合于多人进行分工合作。如同C语言编写大型软件一样，Verilog语言能够很好地支持这种”Top-down”的设计方法。
- 多层次结构电路的描述既可以采用文本方式，也可以使用图形和文本混合设计的方式。用一个8位累加器的设计为例说明这两种设计方式。

图形和文本混合设计



8位全加器

```
module add8(sum,cout,b,a,cin);
output[7:0] sum;
output cout;
input[7:0] a,b;
input cin;
    assign {cout,sum}=a+b+cin;
endmodule
```

8位寄存器

```
module reg8(qout,in,clk,clear);
output[7:0] qout;
input[7:0] in;
input clk,clear;
reg[7:0] qout;
always @(posedge clk or posedge clear)
begin
    if(clear) qout<=0; //异步清0
    else qout<=in;
    end
endmodule
```

顶层文本设计

- 累加器顶层文本描述

```
module acc(accout,cout,accin,cin,clk,clear);
output[7:0] accout;
output cout;
input[7:0] accin;
input cin,clk,clear;
wire[7:0] sum;
    add8 accadd8(sum,cout,accout,accin,cin);
    //调用add8子模块
    reg8 accreg8(accout,sum,clk,clear);
    //调用reg8子模块
endmodule
```

模块调用

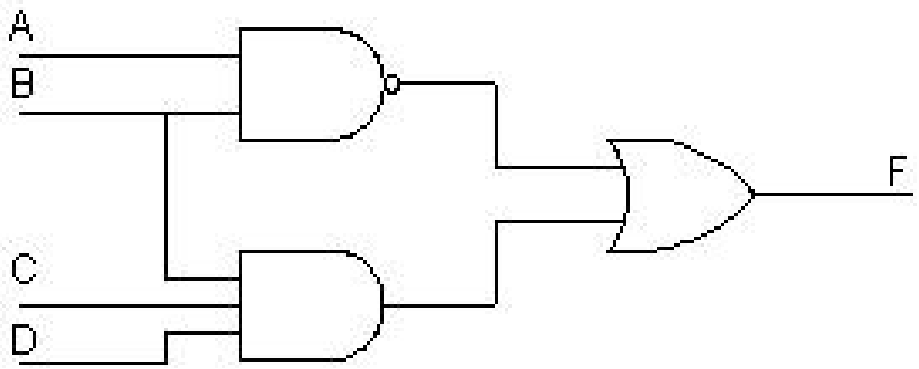
- 对于上面的模块调用，可采用位置对应的方式，即调用时模块端口列表中信号的排列顺序与模块定义时端口列表中的信号排列顺序相同；也可以采用信号名对应方式，此时不必按顺序，例如上面对reg8的调用：
- module reg8(qout,in,clk,clear);
- reg8 accreg8(accout,sum,clk,clear);

- reg8 accreg8(.qout(accout),.clear(clear),.in(sum),.clk(clk));

基本组合逻辑电路设计

门级结构描述

```
module gate1(F,A,B,C,D);
input A,B,C,D;
output F;
    nand(F1,A,B); //调用门元件
    and(F2,B,C,D);
    or(F,F1,F2);
endmodule
```



数据流描述

```
module gate2(F,A,B,C,D);
input A,B,C,D;
output F;
    assign F=(A&B)|(B&C&D);
endmodule
```

译码器74138的Verilog描述

p032.v

```
module ttl74138(a,y,g1,g2a,g2b);
input[2:0] a; input g1,g2a,g2b; output reg[7:0] y;
always @(*)
begin
  if(g1 & ~g2a & ~g2b)          //只有当g1、g2a、g2b为100时，译码器使能
    begin case(a)
        3‘b000:y=8’b11111110;        //译码输出
        3'b001:y=8'b11111101;
        3'b010:y=8'b11111011;
        3'b011:y=8'b11110111;
        3'b100:y=8'b11101111;
        3'b101:y=8'b11011111;
        3'b110:y=8'b10111111;
        3'b111:y=8'b01111111;
        default:y=8'b11111111;
    endcase end
  else y=8'b11111111;
end
endmodule
```

8线—3线优先编码器74148的Verilog描述

```
module ttl74148(din,ei,gs,eo,dout);
input[7:0] din; input ei; output reg gs,eo; output reg[2:0]
dout;
always @(ei,din)
    begin
        if(ei) begin dout<=3'b111;gs<=1'b1;eo<=1'b1; end
        else if(din==8'b11111111)
            begin dout<=3'b111;gs<=1'b1;eo<=1'b0;end
        else if(!din[7]) begin dout<=3'b000;gs<=1'b0;eo<=1'b1;end
        else if(!din[6]) begin dout<=3'b001;gs<=1'b0;eo<=1'b1;end
        else if(!din[5]) begin dout<=3'b010;gs<=1'b0;eo<=1'b1;end
        else if(!din[4]) begin dout<=3'b011;gs<=1'b0;eo<=1'b1;end
        else if(!din[3]) begin dout<=3'b100;gs<=1'b0;eo<=1'b1;end
        else if(!din[2]) begin dout<=3'b101;gs<=1'b0;eo<=1'b1;end
        else if(!din[1]) begin dout<=3'b110;gs<=1'b0;eo<=1'b1;end
        else begin dout<=3'b111;gs<=1'b0;eo<=1'b1;end
    end
endmodule
```

p033.v

奇偶校验位产生器

```
module parity(even_bit,odd_bit,a);
input[7:0] a;
output even_bit,odd_bit;
    assign even_bit=^a;
    //生成偶校验位
    assign odd_bit=~even_bit;
    //生成奇校验位
endmodule
```

带异步清0/异步置1的JK触发器

```
module jkff_rs(clk,j,k,q,rs,set);
input clk,j,k,set,rs; output reg q;
always @(posedge clk, negedge rs, negedge set)
begin
    if(!rs) q<=1'b0;
    else if(!set) q<=1'b1;
    else case({j,k})
        2'b00:q<=q;
        2'b01:q<=1'b0;
        2'b10:q<=1'b1;
        2'b11:q<=~q;
        default:q<=1'bx;
    endcase
end
endmodule
```

p034.v

基本的时序电路

电平敏感的1位数据锁存器

```
module latch1(q,d,le);
input d,le; output q;
    assign q=le?d:q;          //le为高电平时，将输入端数据锁存
endmodule
```

带置位/复位端的1位数据锁存器

```
module latch2(q,d,le,set,reset);
input d,le,set,reset;
output q;
      assign q=reset?0:(set? 1:(le?d:q));
endmodule
```

8位数据锁存器

- 8位数据锁存器（74LS373）

```
module ttl373(le,oe,q,d);
input le,oe; input[7:0] d; output reg[7:0] q;
a l w a y s @ *     //或写为always @(le,oe,d)
begin
    if(~oe & le) q<=d;    //或写为if((!oe) && (le))
    else q<=8'bz;
end
endmodule
```

- p035.v

数据寄存器

```
module reg_w(dout,din,clk,clr);
parameter WIDTH=7;
input clk,clr; input[WIDTH:0] din;
output reg[WIDTH:0] dout;

always @(posedge clk, posedge clr)
begin
    if(clr) dout<=0;else dout<=din;
end

endmodule
```

- p036.v

可变模加法/减法计数器

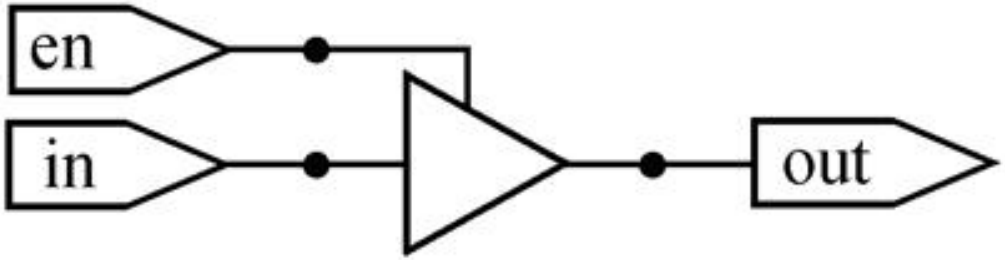
```
module updown_count(d,clk,clear,load,up_down,qd);
input clk,clear,load,up_down;
input[7:0] d; output[7:0] qd; reg[7:0] cnt;
assign qd=cnt;
always @(posedge clk)
    begin if(!clear) cnt<=8'h00;          //同步清0，低电平有效
        else if(load) cnt<=d;             //同步预置
        else if(up_down) cnt<=cnt+1;      //加法计数
        else cnt<=cnt-1;                  //减法计数
    end
endmodule
```

- p037.v

三态逻辑设计

- 行为描述的三态门

```
module tristate1(in,en,out);
input in,en;
output reg out;
always @(in or en)
    begin if(en) out<=in;
    else out<=1'bz; end
endmodule
```



三态逻辑设计

调用门元件bufif1描述的三态门

```
module tristate2(in,en,out);
input in,en; output out; tri out;
    bufif1 b1(out,in,en);
    //注意三态门端口的排列顺序
endmodule
```

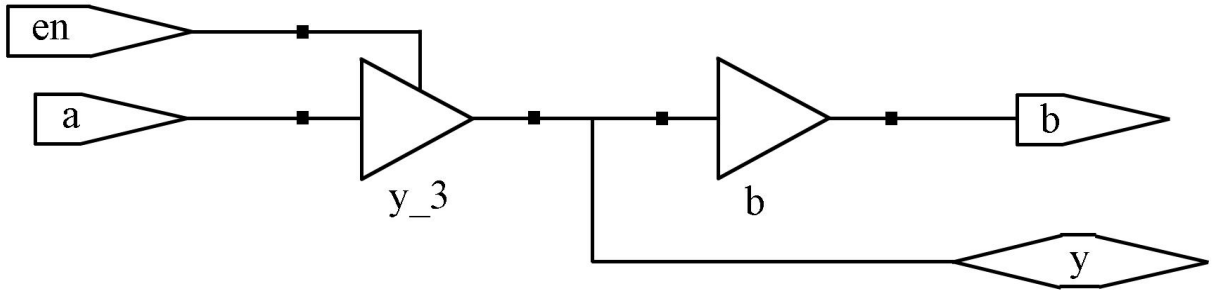
数据流描述的三态门

```
module tristate3(out,in,en);
input in,en; output out;
assign out=en?in:1'bz;          //若en=1，out=in；
                                //若en=0，out为高阻态

endmodule
```

三态双向驱动器

```
module bidir(y,a,en,b);
input a,en; output b;
inout y;
    assign y=en?a:'bz;
    assign b=y;
endmodule
```



三态双向总线缓冲器

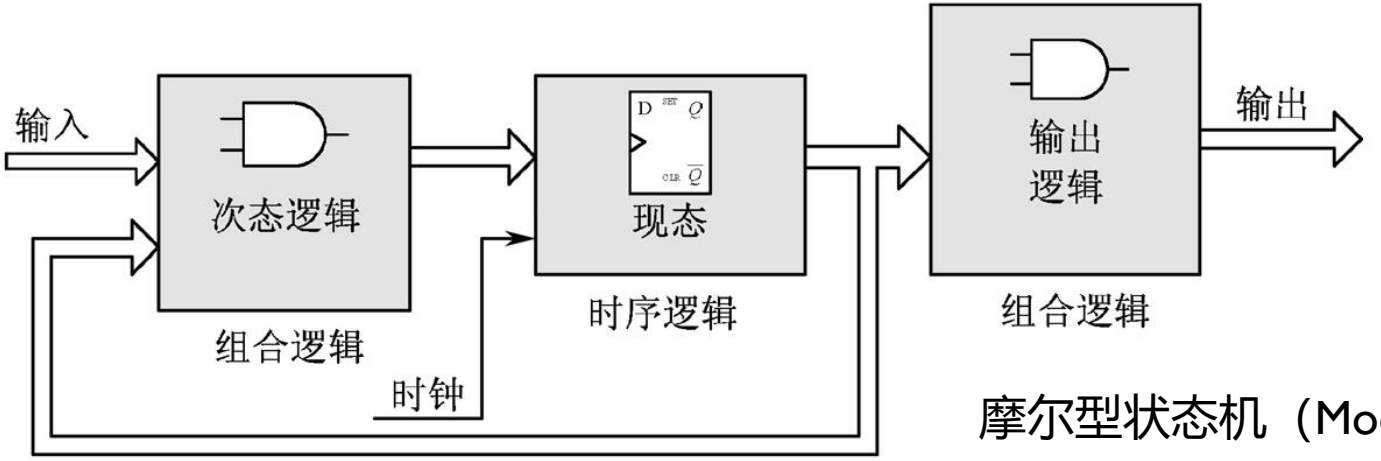
```
module ttl245(a,b,oe,dir);
input oe,dir; //使能信号和方向控制
inout[7:0] a,b; //双向数据线
    assign a=({oe,dir}==2'b00)?b:8'bz;
    assign b=({oe,dir}==2'b01)?a:8'bz;
endmodule
```

VERILOG有限状态机设计

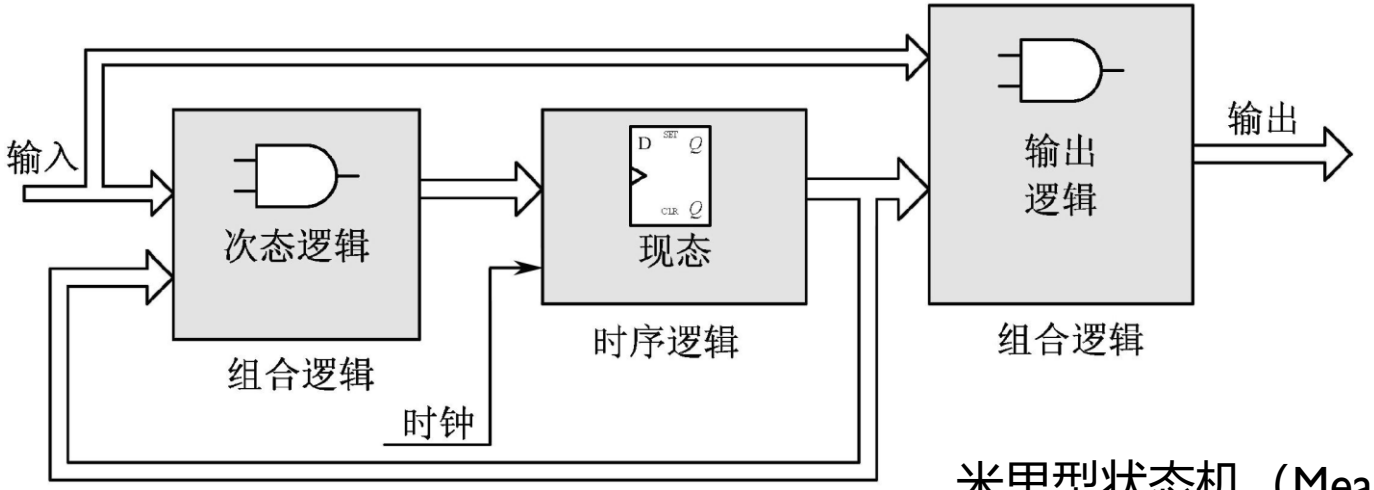
Verilog有限状态机设计

- 有限状态机
- 有限状态机的Verilog描述
- 状态编码
- 有限状态机设计要点

有限状态机



摩尔型状态机（Moore）



米里型状态机（Mealy）

用状态机设计模5计数器

```
module fsm(clk,clr,z,qout);
input clk,clr; output reg z; output reg[2:0] qout;
always @(posedge clk or posedge clr) //此过程定义状态转换
begin
    if(clr) qout<=0; //异步复位
    else case(qout)
        3'b000: qout<=3'b001;
        3'b001: qout<=3'b010;
        3'b010: qout<=3'b011;
        3'b011: qout<=3'b100;
        3'b100: qout<=3'b000;
        default: qout<=3'b000; /*default语句*/
    endcase
end

always @(qout) /*此过程产生输出逻辑*/
begin
    case(qout)
        3'b100: z=1'b1;
        default:z=1'b0;
    endcase
end
endmodule
```

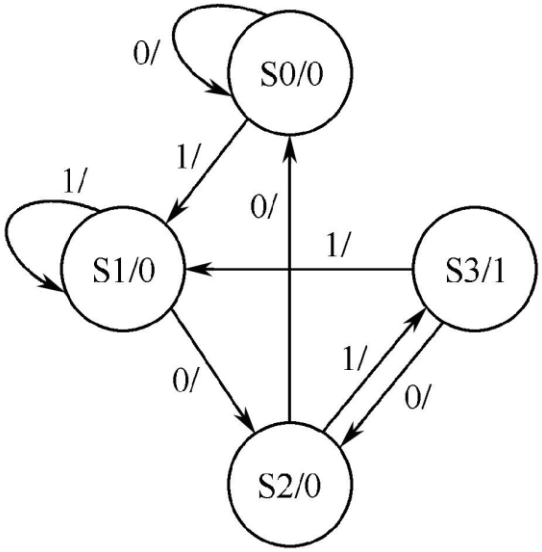
p038.v

有限状态机的几种描述方式

- （1）用三个过程描述：即现态（CS），次态（NS），输出逻辑（OL）各用一个always过程描述。
- （2）双过程描述（CS+NS，OL双过程描述）：使用两个always过程来描述有限状态机，一个过程描述现态和次态时序逻辑（CS+NS）；另外一个过程描述输出逻辑（OL）。
- （3）双过程描述（CS，NS+OI双过程描述）：一个过程用来描述现态（CS）；另一个过程描述次态和输出逻辑（NS+OL）。
- （4）但过程描述：在单过程描述方式中，将状态机的现态、次态和输出逻辑（CS+NS+OL）放在一个always过程中进行描述。

101序列检测器的Verilog描述（三个过程）

- p039.v



101序列检测器单过程描述

- p040.v

状态编码

- 常用的编码方式

- 顺序编码
- 格雷编码
- 约翰逊编码
- 一位热码

	State Variables		
State	**One-Hot Code**	**Binary Code**	**Gray Code**
S0	00001	000	000
S1	00010	001	001
S2	00100	010	011
S3	01000	011	010
S4	10000	100	110

Table 1:An example of state Encoding for a 4 state Machine

状态编码的定义

在Verilog语言中，有两种方式可用于定义状态编码，分别用parameter和'define语句实现，比如要为state0、state1、state2、state3四个状态定义码字为：00、01、11、10，可采用下面两种方式。

方式1：用parameter参数定义

```
parameter
state1=2'b00,state2=2'b01,state3=2'b11,state4=2'b10;
……
case(state)
state1: …; //调用
state2: …;
……
```

状态编码的定义方式2：用'define语句定义

```
'define state1 2'b00 //不要加分号“;”
'define state2 2'b01
'define state3 2'b11
'define state4 2'b10
case(state)
'state1: …; //调用，不要漏掉符号“`”
'state2: …;
……
```

要注意两种方式定义与调用时的区别，一般情况下，更倾向于采用方式I来定义状态编码。一般使用case，casez和casex语句来描述状态之间的转换，用case语句表述比用if-else语句更加清晰明了

有限状态机设计要点

- 1. 起始状态的选择：起始状态是指电路复位后所处的状态，选择一个合理的起始状态将使整个系统简洁，高效。多数EDA软件会自动为基于状态机的设计选择一个最佳的起始状态。
- 2. 有限状态机的同步复位
- 2. 有限状态机的异步复位

多余状态的处理

- 一般有如下两种处理多余状态的方法：
- （1）在case语句中使用default分支决定如果进入无效状态所采取的措施；
- （2）编写必要的Verilog源代码明确定义进入无效状态所采取的行为。

谢谢