第2章 Verilog 语法的基本概念

概述

Verilog HDL是一种用于数字系统设计的语言。用Verilog HDL描述的电路设计就是该电路的Verilog HDL模型，也称为模块。Verilog HDL既是一种行为描述的语言也是一种结构描述的语言。这也就是说，既可以用电路的功能描述也可以用元器件和它们之间的连接来建立所设计电路的Verilog HDL模型。
Verilog模型可以是实际电路的不同级别的抽象。

系统级:用语言提供的高级结构实现设计模块的外部性能的模型。

算法级:用语言提供的高级结构实现算法行为的模型。

RTL级:描述数据在寄存器之间流动和如何处理、控制这些数据流动的模型。

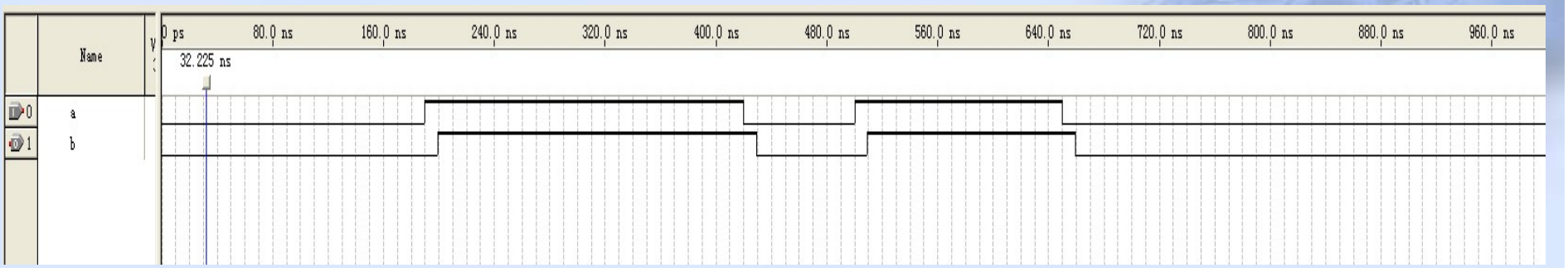
以上3种都属于行为描述，只有RTL级才与逻辑电路有明确的对应关系。

门级:描述逻辑门以及逻辑门之间的连接的模型。

开关级:描述器件中三极管和储存节点以及它们之间连接的模型。

2.1 Verilog模块的基本概念

```
module lx1(b,a);  //module lx1(a,b);
input a;
output b;

  assign b=a;  //assign b=~a;
endmodule

/*
force a 0 0,   1 23 ,0 96,1 132,0 205
run 300 ns
*/
```


Name
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
800.0 ns
880.0 ns
960.0 ns
32.225 ns
0
a
1
b

```
module lx2(b,a);
input a;
output b;

reg b;

   always @(a)
      b=a;

endmodule
```

always @(a)表示如果a有变化就执行下面的语句

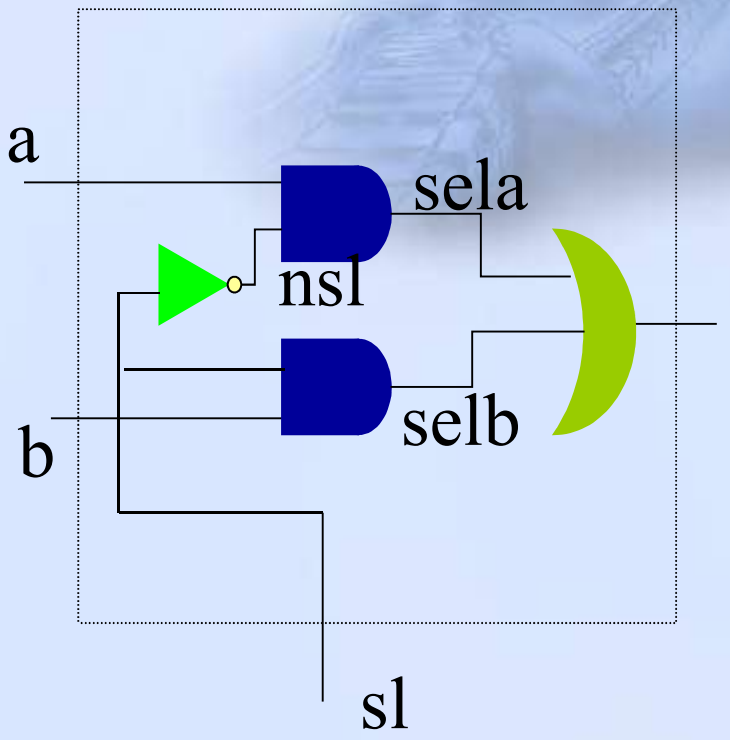
例2.1　二选一多路选择器

```
module muxtwo(out,a,b,sl);
  input a,b,sl;
  output out;
  reg out;

  always @(sl or a or b)
    if(!sl)
      out=a;
    else
      out=b;
endmodule
```

always @(sl or a or b)表示只要sl或a或b，其中若有一个变化时就执行下面的语句

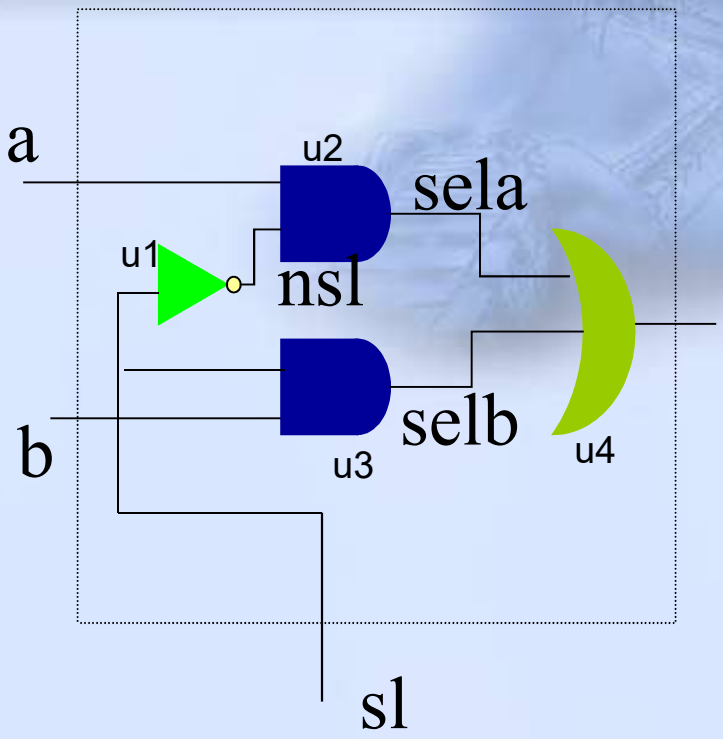
例2.2

```
module muxtwo(out,a,b,sl);
input a,b,sl;
output out;
wire nsl,sela,selb;

  assign nsl=~sl;
  assign sela=a&nsl;
  assign selb=b&sl;
  assign out=sela|selb; //
endmodule
```



例2.3

```
module muxtwo (out,a,b,sl);
   input a,b,sl;
   output out;
   wire nsl,sela,selb;//
   not u1(nsl,sl);
   and #1 u2(sela,a,nsl);
   and #1 u3(selb,b,sl);
   or  #2 u4(out,sela,selb);
endmodule
```



and、or和not都是Verilog语言的保留字，由Verilog语言的原语（primitive）规定了它们的接口顺序和用法，分别表示与门、或门和非门，#1和#2分别表示门输入到输出的延迟为1和2个时间单位。

如果在编写Verilog模块时符合一些基本规则，就可以通过软件把例2.1通过例2.2的中间形式自动转换为例2.3形式的模块，这个过程叫做综合。

例2.4 通过连续赋值语句描述3位加法器

```
module  adder ( count,sum,a,b,cin );
input [2:0] a,b;
input   cin;
output  count;
output [2:0] sum;
      assign {count,sum} = a + b + cin;
endmodule
```

例2.5 通过连续赋值语句描述一个比较器

```
module compare ( equal,a,b );
output  equal;     //声明输出信号equal
input [1:0] a,b;  //声明输入信号a,b
  assign  equal = (a = = b) ? 1: 0;
/*如果a、b 两个输入信号相等,输出为1。否则为0*/
endmodule
```

```
module compare_2 ( equal,a,b );
output  equal;     //声明输出信号equal
input [1:0] a,b;  //声明输入信号a,b
reg equal;//

  always @(a,b)
    if(a==b)
      equal=1;
    else
      equal=0;
/*如果a、b 两个输入信号相等,输出为1。否则为0*/
endmodule
/*
force a 00 0 ,10 100,  01 200,11 300
force  b 00 0,  01 50,  10 150,  01 250,  11 350
run 500
*/
```

```
module compare_3 ( equal,a,b );
output  equal;     //声明输出信号equal
input [1:0] a,b;   //声明输入信号a,b
reg equal;//

  always @(a,b)
  begin    //
    if(a==b)
      equal=1;
    else
      equal=0;
  end      //
//  assign  equal = (a = = b) ? 1: 0;
/*如果a、b 两个输入信号相等,输出为1。否则为0*/
endmodule
```

例2.6 三态门选择器

```
module trist2(out,in,enable);
output  out;
input   in, enable;
  bufif1  mybuf(out,in,enable);
endmodule
```

bufif1是Verilog语言中的原语库中现存的三态驱动器元件。这种引用现成元件或模块的方法叫做实例化或实例引用。

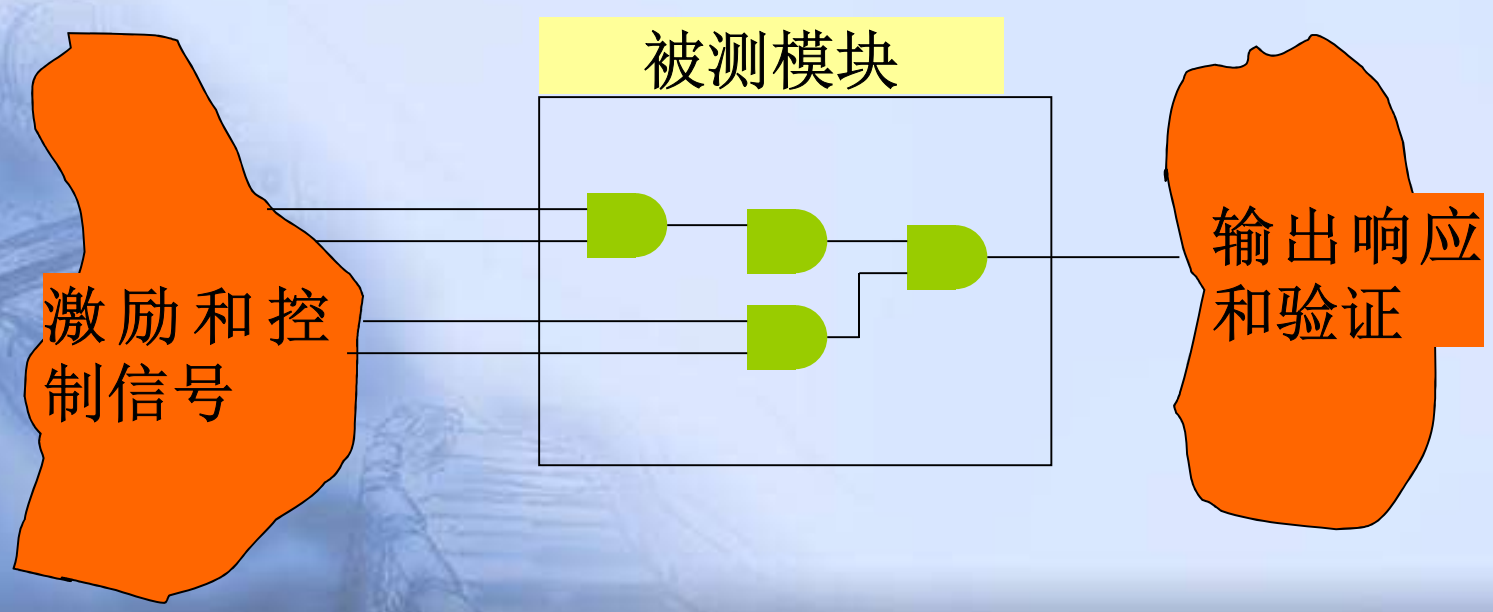
例2.7 采用二个模块的三态门选择器

```
module trist1(sout,sin,ena);
output  sout;
input  sin, ena;
   mytri  tri_inst(.out(sout),.in(sin),.enable(ena));
    //调用由mytri模块定义的实例元件tri_inst
    //在引用时用“.”符号，标明定义时规定的端口名
endmodule

module  mytri(out,in,enable);
output  out;
input  in, enable;
   assign  out = enable? in : 1'bz;
endmodule
```

```
module trist2(sout,sin,ena);
output  sout;
input  sin, ena;
   mytri  tri_inst(sout,sin,ena);
    //调用由mytri模块定义的实例元件tri_inst
    //在引用时严格按模块定义的端口顺序来连接,
    //不用标明原模块定义时规定的端口名
endmodule

module  mytri(out,in,enable);
output  out;
input  in, enable;
   assign  out = enable? in : 1'bz;
endmodule
```

2.2 Verilog用于模块的测试

Verilog还可以用来描述变化的测试信号。描述测试信号的变化和测试过程的模块也叫做测试平台（testbench），它可以对上面介绍的电路模块进行动态的全面测试。通过观测被测试模块的输出信号是否符合要求，可以调试和验证逻辑系统的设计和结构正确与否，并发现问题及时修改。



一个很简单的测试平台

```
module t_lx1;
wire b;
reg a;

initial
begin
  a=0;
  #23 a=1;
  #96 a=0;
end

lx1 u1(.a(a),.b(b));
endmodule
```

```
`include "muxtwo.v"
module t;
  reg ain,bin,select;
  reg clock;
  wire outw;

  initial
  begin
    ain=0;
    bin=0;
    select=0;
    clock=0;
  end

  always #50 clock=~clock;

  always @(posedge clock)
  begin
    #1 ain={$random} %2;     //~ain;
    #3 bin={$random} %2; //~bin;
  end

  always #10000 select=!select;

  muxtwo m(.out(outw),.a(ain),.b(bin),.sl(select));
endmodule
```

```
module muxtwo(out,a,b,sl);
  input a,b,sl;
  output out;
  reg out;

  always @(sl or a or b)
    if(!sl)
      out=a;
    else
      out=b;
endmodule
```