2.5 Verilog HDL基础及描述组合逻辑电路

2.5 硬件描述语言Verilog HDL基础

硬件描述语言HDL(Hardware Description Language) 类似于高级程序设计语言。是一种以文本形式来描述数字系统硬件的结构和行为的语言,用它可以表示逻辑电路图、逻辑表达式，复杂数字逻辑系统完成的逻辑功能

HDL是高层次自动化设计的起点和基础

1.HDL的产生

起源于美国国防部提出的超高速集成电路研究计划，目的是为了把电子电路的设计意义以文字或文件的方式保存下来，以便其他人能轻易地了解电路的设计意义 。

随着集成电路的亚微米和深亚微米制造、设计技术的飞速发展，集成电路已进入片上系统SOC（System on a chip）时代。SOC通常是由硬件电路和运行其上的系统软件构成。硬件电路一般使用HDL进行描述.

2.几种硬件描述语言

ABEL (Advanced Bolean Equation Language)

VHDL (V--Very High Speed Integrated Circuit)

Verilog HDL （简称Verilog）

VHDL 和Verilog的功能较强，属于行为描述语言。两种HDL均为IEEE标准。特别是Verilog由于其句法根源出自C语言,它相对VHDL好用、好学

计算机对HDL的处理:

逻辑仿真 是指用计算机仿真软件对数字逻辑电路的结构和行为进行预测.仿真器对HDL描述进行解释，以文本形式或时序波形图形式给出电路的输出。在仿真期间如发现设计中存在错误，就再要对HDL描述进行及时的修改。

逻辑综合 是指从HDL描述的数字逻辑电路模型中导出电路基本元件列表以及元件之间的连接关系（常称为门级网表）的过程。类似对高级程序语言设计进行编译产生目标代码的过程.产生门级元件及其连接关系的数据库，根据这个数据库可以制作出集成电路或印刷电路板PCB。

2.5.1 Verilog语言的基本语法规则

为对数字电路进行描述（常称为建模），Verilog语言规定了一套完整的语法结构。

1．**间隔符**：Verilog 的间隔符主要起分隔文本的作用，可以使文本错落有致，便于阅读与修改。

间隔符包括空格符（\b）、TAB 键（\t）、换行符（\n）及换页符。

2．**注释符**:注释只是为了改善程序的可读性,在编译时不起作用。

多行注释符(用于写多行注释): /* --- */;

单行注释符 :以//开始到行尾结束为注释文字。

3．标识符和关键词

标识符:给对象（如模块名、电路的输入与输出端口、变量等）取名所用的字符串。以英文字母或下划线开始

如，clk、counter8、_net、bus_A 。

关键词:是Verilog语言本身规定的特殊字符串，用来定义语言的结构。例如，module、endmodule、input、output、wire、reg、and等都是关键词。关键词都是小写，关键词不能作为标识符使用 。

4．逻辑值集合

为了表示数字逻辑电路的逻辑状态，Verilog语言规定了4种基本的逻辑值。

0	逻辑0、逻辑假
1	逻辑1、逻辑真
x或X	不确定的值（未知状态）
z或Z	高阻态

5．常量及其表示

常量
- 整数型
  - 十进制数的形式的表示方法:表示有符号常量
    例如：30、 - 2
  - 带基数的形式的表示方法:
    格式为：<＋/－><size>’<signed><base format><number>
    例如：3’b101、-5’o37、12’h13x，8’b1001_0011
- 实数型常量
  - 十进制记数法 如： 0.1、2.0、5.67
  - 科学记数法 如: 23_5.1e2、5E－4
    23510.0、 0.0005

Verilog允许用参数定义语句定义一个标识符来代表一个常量，称为**符号常量**。定义的格式为：

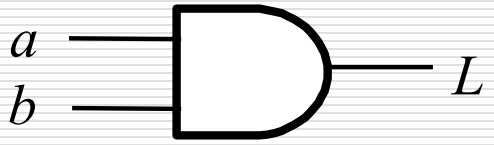
parameter 参数名1 = 常量表达式1，参数名2 = 常量表达式2，……；如 parameter BIT=1, BYTE=8, PI=3.14;

6．字符串:字符串是双撇号内的字符序列

2.5.2 变量的数据类型

1线网类型:是指输出始终根据输入的变化而更新其值的变量,它一般指的是硬件电路中的各种物理连接，是对硬件电路中元件之间实际存在物理连线的抽象

例:线网型变量L的值由与门的驱动信号a和b所决定，即L = a&b。a、b的值发生变化，线网L的值会立即跟着变化。



常用的线网类型由关键词wire定义
wire型变量的定义格式如下:
wire [n-1:0] 变量名1，变量名2，…，变量名n;

变量宽度

例:wire L; //将上述电路的输出信号L声明为线网型变量
wire [7:0] data bus; //声明一个8-bit宽的线网型总线变量

2、寄存器类型

寄存器型变量对应的是具有状态保持作用的电路元件,表示一个抽象的数据存储单元，如触发器寄存器。**寄存器型变量只能在initial或always内部被赋值。**

4种寄存器类型的变量

抽象描述，不对应具体硬件

寄存器类型	功能说明
reg	常用的寄存器型变量
integer	32位带符号的整数型变量
real	64位带符号的实数型变量
time	64位无符号的时间变量

例： reg clock；//定义一个1位寄存器变量

reg [3:0] counter; //定义一个4位寄存器变量

2.5.3 运算符及其优先级

1. 运算符

运算符分为算术运算符、逻辑运算符、关系运算符、移位运算符等

类型	符号	功能说明	类型	符号	功能说明
算术运算符	+ - - * /	二进制加 二进制减 2的补码 二进制乘 二进制除	关系运算符 (双目运算符)	>= <= == !=	大于 小于 大于或等于 小于或等于 相等 不相等
位运算符 (双目运算符)	~ & | ^ ^~ 或 ~^	按位取反 按位与 按位或 按位异或 按位同或	缩位运算符 (单目运算符)	& ~& | ~| ^ ^~ 或 ~^	缩位与 缩位与非 缩位或 缩位或非 缩位异或 缩位同或
逻辑运算符 (双目运算符)	! && ||	逻辑非 逻辑与 逻辑或	移位运算符 (双目运算符)	>> <<	右移 左移
位拼接运算符	{,} {{}}	将多个操作数拼接成为一个操作数	条件运算符 (三目运算符)	?:	根据条件表达式是否成立，选择表达式

位拼接运算符

作用是将两个或多个信号的某些位拼接起来成为一个新的操作数，进行运算操作。

设A=1’b1，B=2’b10，C=2’b00

则{B,C} = 4’b1000
{A,B[1],C[0]} = 3’b110
{A,B,C,3’b101}=8’b11000101。

对同一个操作数的重复拼接还可以双重大括号构成的运算符{{}}
例如{4{A}}=4’b1111，{2{A},2{B},C}=8’b11101000。

位运算符与缩位运算的比较

A：4’b1010 、
B：4’b1111，

位运算	~A = 0101 ~B = 0000	A&B= 1010	A|B= 1111	A^B= 0101	A~^B= 1010
缩位运算	&A=1& 0&1&0=0	~&A=1 &B=1	|A=1 ~|B=0	^A=0 ^B=0	~^A=1 ~^B=1

2. 运算符的优先级

优先级的顺序从下向上依次增加。

类型	符号	优先级别
取反	！ ~ -(求2的补码)	最高优先级
算术	* / + -	
移位	>> <<	
关系	< <= > >=	
等于	== !=	
缩位	& ~& ^ ^~ | ~|	
逻辑	&& ||	
条件	?:	最低优先级

条件运算符

是三目运算符，运算时根据条件表达式的值选择表达式。

一般用法：

condition_expr?expr1:expr2;

首先计算第一个操作数condition_expr的值，如果结果为逻辑1，则选择第二个操作数expr1的值作为结果返回，结果为逻辑0，选择第三个操作数expr2的值作为结果返回。

2.5.4 Verilog程序的基本结构

模块是Verilog描述电路的基本单元。对数字电路建模时，用一个或多个模块。不同模块之间通过端口进行连接。

1、每个模块以关键词module开始，以endmodule结束。

2、每个模块先要进行端口的定义，并说明输入（input)和输出（output),然后对模块功能进行描述。

3、除了endmodule语句外，每个语句后必须有分号。

4、可以用/* --- */和//.....对程序的任何部分做注释。

5、逻辑功能的描述方式有三种不同风格：结构描述方式（门级描述方式），数据流描述方式，行为描述方式。

模块定义的一般语法结构如下：

module 模块名（端口名1，端口名2，端口名3,…）；

端口类型说明(input, outout, inout)；
参数定义(可选)；
数据类型定义(wire, reg等)；

} 说明部分

实例化低层模块和基本门级元件；
连续赋值语句（assign）；
过程块结构（initial和always）
行为描述语句；

} 逻辑功能描述部分，其顺序是任意的

endmodule

2.5.5 Verilog内部的基本门级元件及门级建模

Verilog HDL内部预先定义了12个基本门级元件模型，引用这些基本门级元件对逻辑图进行描述，称为门级建模。

三态门　　Verilog基本门级元件　　多输出门　　多输入门

元件符号	功能说明	元件符号	功能说明
and	多输入端的与门	nand	多输入端的与非门
or	多输入端的或门	nor	多输入端的或非门
xor	多输入端的异或门	xnor	多输入端的异或非门
buf	多输出端的缓冲器	not	多输出端的反相器
bufif1	控制信号高电平有效的三态缓冲器	notif1	控制信号高电平有效的三态反相器
bufif0	控制信号低电平有效的三态缓冲器	notif0	控制信号低电平有效的三态反相器

多输入门：and、nand、or、nor、xor、xnor只有单个输出,1个或多个输入
多输出门：not、buf允许有多个输出,但只有一个输入
三态门：bufif0、bufif1、notif0、notif1有一个输出,一个数据输入和一个控制输入

Verilog基本门级元件模型

and	n-input AND gate
nand	n-input NAND gate
or	n-input OR gate
nor	n-input NOR gate
xor	n-input exclusive OR gate
xnor	n-input exclusive NOR gate

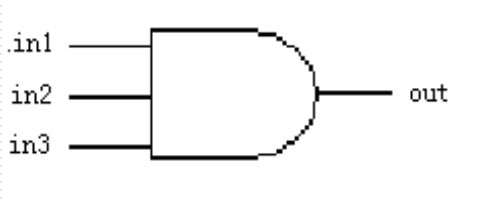
buf	n-output buffer
not	n-output inverter
bufif0	tri-state buffer; Io enable
bufif1	tri-state buffer; hi enable
notif0	tri-state inverter; Io enable
notif1	tri-state inverter; hi enable

1、多输入门

只允许有一个输出，但可以有多个输入。

调用名

and A1（out，in1，in2，in3）；



两输入and真值表

and		输入A			
		0	1	X	z
输入B	0	0	0	0	0
	1	0	1	x	x
	x	0	x	x	x
	z	0	x	x	x

X- 不确定状态　　Z- 高阻态

or真值表

or		输入1			
		0	1	X	Z
输入2	0	0	1	X	X
	1	1	1	1	1
	X	X	1	X	X
	Z	X	1	X	X

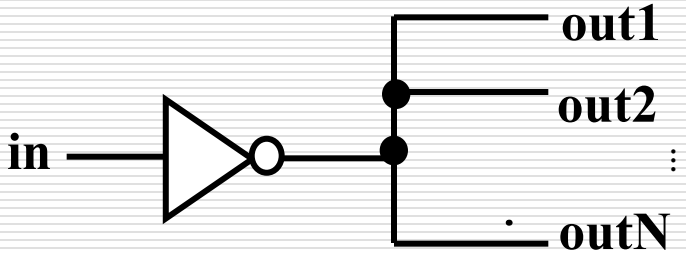
xor真值表

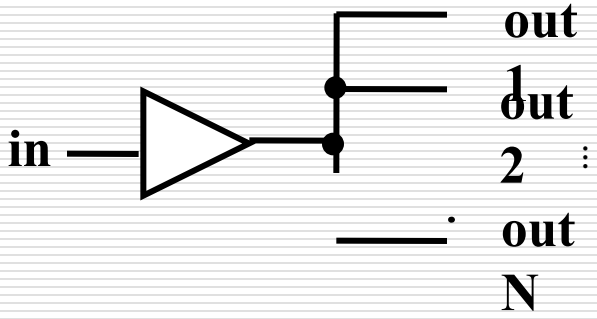
xor		输入1			
		0	1	X	Z
输入2	0	0	1	X	X
	1	1	0	X	X
	X	X	X	X	X
	Z	X	X	X	X

2、多输出门

允许有多个输出，但只有一个输入。

not N1（out1，out2，…，in）；buf B1（out1，out2，…，in）；





not真值表

not	输 入			
	0	1	x	z
输 出	1	0	x	x

buf真值表

buf	输 入			
	0	1	x	z
输 出	0	1	x	x

3、三态门

有一个输出、一个数据输入和一个输入控制。
如果输入控制信号无效，则三态门的输出为高阻态z。

图 4.6.3 三态门元件模型
（a）bufif1　　（b）notif1

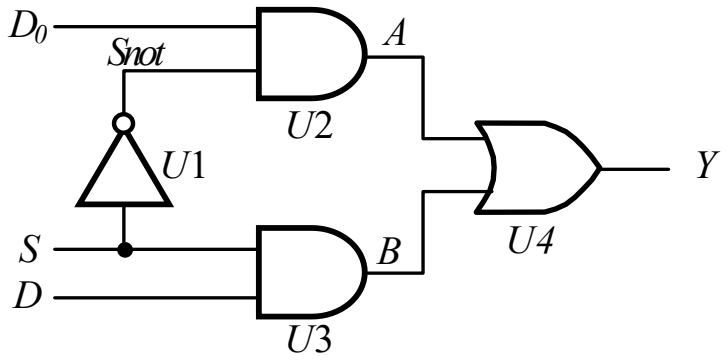
bufif1真值表

bufif1		控制输入			
		0	1	x	z
数据输入	0	z	0	0/z	0/z
	1	z	1	1/z	1/z
	x	z	x	x	x
	z	z	x	x	x

notif1真值表

notif1		控制输入			
		0	1	x	z
数据输入	0	z	1	1/z	1/z
	1	z	0	0/z	0/z
	x	z	x	x	x
	z	z	x	x	x

例：用结构描述方式建立门电路Verilog模型



模块名

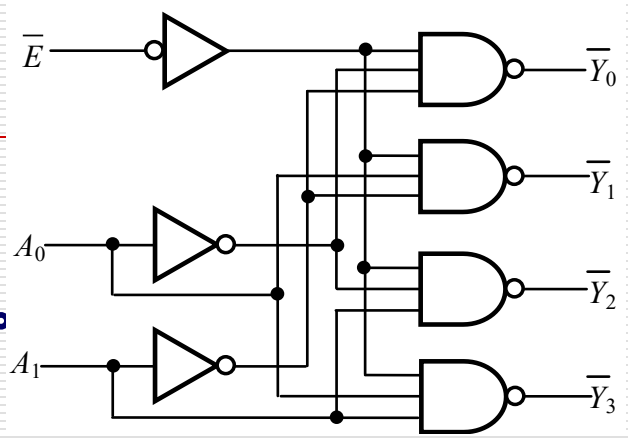
```
module mux2to1(D0, D1, S, Y );
  input D0, D1, S;  //定义输入信号
  output Y;    //定义输出信号
  wire Snot, A, B ; //定义内部节点信号数据类型
//下面对电路的逻辑功能进行描述
  not U1(Snot, S);
  and U2(A, D0, Snot);
  and U3(B, D1, S);
  or U4(Y, A, B);
endmodule
```

端口类型说明

数据类型说明

电路结构描述

思考：对2线-4线译码器进行门级建模。




```
Module Decoder2to4_gates(A,En,Y);
    input En;                          //声明输入变量
    input [1:0] A;                     //声明输入变量
    output [3:0] Y;                    //声明输出变量
    wire A1not, A0not;                 //声明电路内部节点
    not (A1not,A[1]);                  //实例引用非门
    not (A0not,A[0]);
    nand U0(Y[0],A1not,A0not,En);      //实例引用与非门
    nand U1(Y[1],A1not,A[0],En);
    nand U2(Y[2],A[1],A0not,En);
    nand U3(Y[3],A[1],A[0],En);
endmodule
```

2.5.6 数据流建模

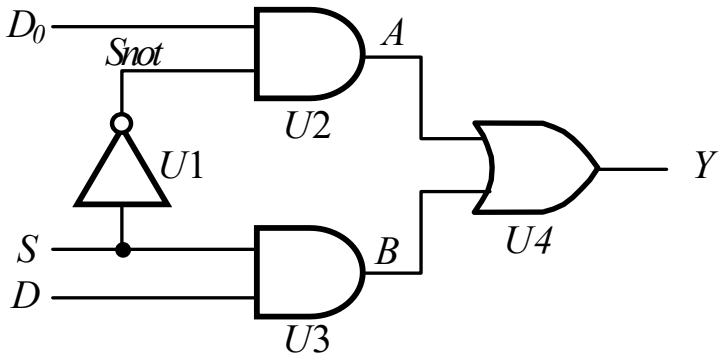
数据流建模使用的连续赋值语句由关键词assign开始，后面跟着由操作符和运算符组成的逻辑表达式。

一般用法为：

Wire [位宽说明] 变量名1，变量名2，…，变量名3
Assign 变量名=表达式

例 用数据流描述方式建立模型

$$Y = D_0 \cdot \overline{S} + D_1 \cdot S$$




```
module mux2to1_dataflow(D0, D1, S, Y );
  input D0, D1, S;
  output Y;
  wire Y ;
//下面是逻辑功能描述
  assign Y = (~S & D0) | (S & D1); //表达式左边Y必须是wire型
endmodule
```

端口类型说明

数据类型说明

电路结构描述

- 思考：使用数据流描述风格对2线-4线译码器进行建模。

```
Module Decoder2to4_DF(A,En,Y);
    input En;                          //声明输入变量
    input [1:0] A;                     //声明输入变量
    output [3:0] Y;                    //声明输出变量
    //下面用连续赋值语句描述模块的逻辑功能
    assign Y[0]=~(~A[1]&~A[0]&En);
    assign Y[1]=~(~A[1]&A[0]&En);
    assign Y[2]=~(A[1]&~A[0]&En);
    assign Y[3]=~(A[1]&A[0]&En);
endmodule
```

2.5.7 组合电路的行为级建模

行为级建模是将数字逻辑电路的功能和算法以比较抽象的形式描述出来，主要使用由关键词initial和always定义的两种结构类型的描述语句。

在两种结构内部，会使用条件语句（if-else）、多路分支语句（case-endcase）和循环语句等比较抽象的编程语句。

Verilog HDL行为描述方法

过程块的组成：

过程语句@(事件控制敏感表)

begin (：块名)

块内局部变量说明

一条或多条过程赋值或高级程序语句

end

1、条件语句（if语句）

条件语句就是根据判断条件是否成立，确定下一步的运算。

Verilog语言中有3种形式的if语句：

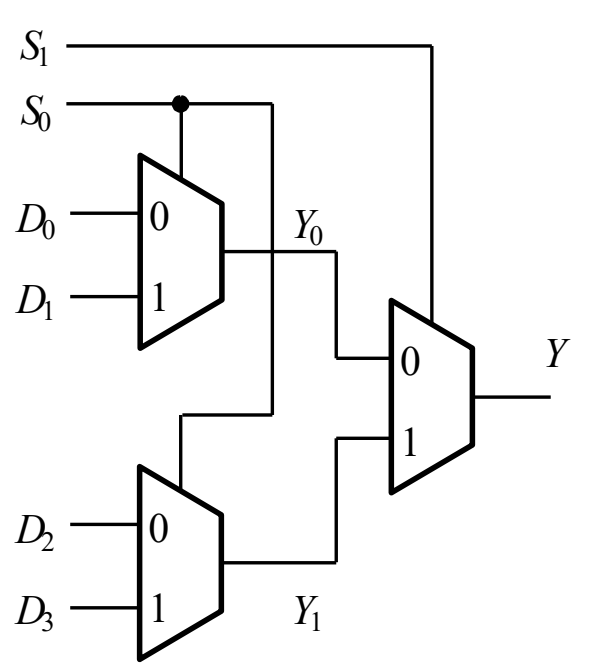
（1） if (condition_expr) true_statement;

（2） if (condition_expr)true_statement;
else fale_ statement;

（3） if (condition_expr1) true_statement1;
else if (condition_expr2) true_statement2;
else if (condition_expr3) true_statement3;
……
else default_statement;

if后面的条件表达式一般为逻辑表达式或关系表达式。执行if语句时，首先计算表达式的值，若结果为0、x或z，按“假”处理；若结果为1，按“真”处理，并执行相应的语句。

例：使用if-else语句对4选1数据选择器的行为进行描述




```
module mux4to1_bh(D, S, Y);
   input [3:0] D;  //输入端口
   input [1:0] S;  //输入端口
   output reg Y;  //输出端口及变量数据类型
   always @(D, S) //电路功能描述
     if (S == 2'b00)     Y = D[0];
     else if (S== 2'b01)  Y = D[1];
     else if (S== 2'b10)  Y = D[2];
     else                 Y = D[3];
endmodule
```

注意，过程赋值语句只能给寄存器型变量赋值，因此，输出变量Y的数据类型定义为reg。

2、多路分支语句（case语句）

是一种多分支条件选择语句，一般形式如下

```
case (case_expr)
        item_expr1: statement1;
        item_expr2: statement2;
          ……
        default: default_statement; //default语句可以省略
endcase
```

注意：当分支项中的语句是多条语句，必须在最前面写上关键词begin，在最后写上关键词end，成为顺序语句块。

另外，用关键词casex和casez表示含有无关项x和高阻z的情况。

例：对具有使能端En 的4选1数据选择器的行为进行Verilog描述。当En=0时，数据选择器工作，En=1时，禁止工作，输出为0。

```
module mux4to1_bh (D, S, Y);
  input [3:0] D，[1:0] S；
  output  reg  Y；
  always @(D, S, En)  //2001, 2005 syntax
begin
   if (En==1)  Y = 0; //En=1时，输出为0
   else              //En=0时，选择器工作
    case (S)
      2’d0: Y = D[0];
      2’d1: Y = D[1];
      2’d2: Y = D[2];
      2’d3: Y = D[3];
    endcase
end
endmodule
```

3、for循环语句

一般形式如下

for (initial_assignment; condition; step_assignment) statement;

initial_assignment为循环变量的初始值。

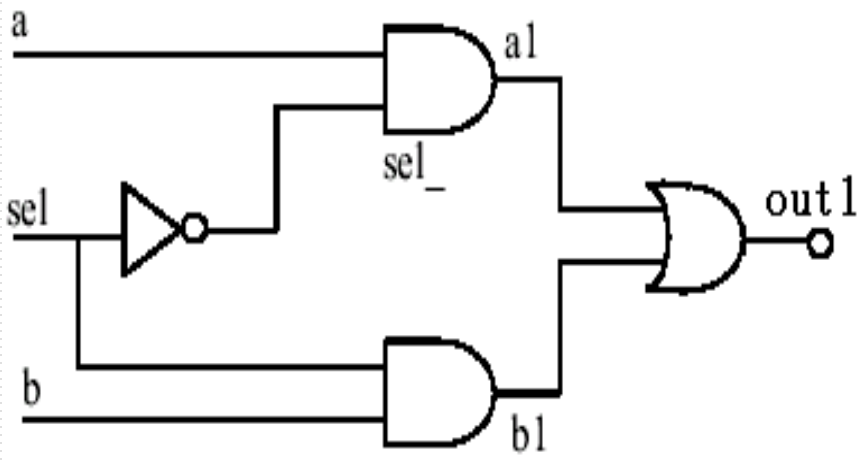
Condition为循环的条件，若为真，执行过程赋值语句statement，若不成立，循环结束，执行for后面的语句。

step_assignment为循环变量的步长，每次迭代后，循环变量将增加或减少一个步长。

试用Verilog语言描述具有高电平使能的3线-8线译码器.

```
module ecoder3to8_bh(A,En,Y);
      input [2:0] A，En;
      output reg  [7:0]Y;
      integer k;        //声明一个整型变量k
      always @(A, En)   //
        begin
            Y = 8'b1111_1111;     //设译码器输出的默认值
           for(k = 0; k <= 7; k = k+1) //下面的if-else语句循环8次
              if ((En==1) && (A== k) )
                  Y[k] = 0;   //当En=1时，根据A进行译码
              else
                  Y[k] = 1;   //处理使能无效或输入无效的情况
        end
   endmodule
```

循环8次




```
module mux2_1(out1,a,b,sel);
   output out1;
   input a,b,sel;
  wire sel_,a1,b1;
  not (sel_, sel);
  and U1(a1, a, sel_);
  and U2(b1, b, sel);
  or (out1, a1, b1);
endmodule
```

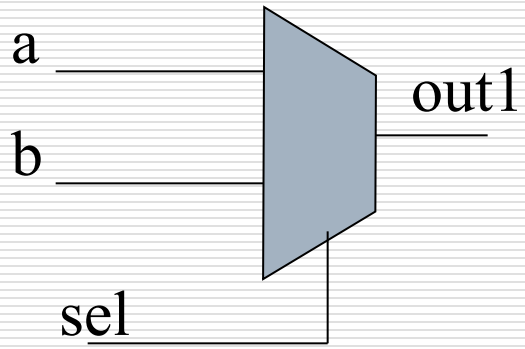
结构描述

```
module mux2_1(out1, a, b, sel) ;
    output   out1;
    input  a, b;
    input sel;

assign out1= sel ? b : a;

endmodule
```

数据流描述




```
module mux2_1(out1, a, b, sel) ;
    output   out1;
    input  a, b;
    input sel;

assign out1=(sel & b) | (~sel & a);

endmodule
```

数据流描述

行为描述

```
module mux2_1(out1, a, b, sel) ;
   output   out1;
   input  a, b;
   input sel;
    reg out1;

always @(sel or a or b)
  begin
     if (sel)
                out1 = b;
     else     out1 = a;
  end
endmodule
```

```
module mux2_1(out1, a, b, sel) ;
   output   out1;
   input  a, b;
   input sel;
    reg out1;
always @(sel or a or b)
begin
   case (sel)
      1'b0 :  out1 = a;
      1'b1 :  out1 = b;
   endcase
end
endmodule
```

•行为描述方式：

一般使用下述语句描述，可以对组合、时序逻辑电路建模。

1）initial 语句

2）always 语句

•数据流描述方式：

一般使用assign语句描述，主要用于对组合逻辑电路建模。

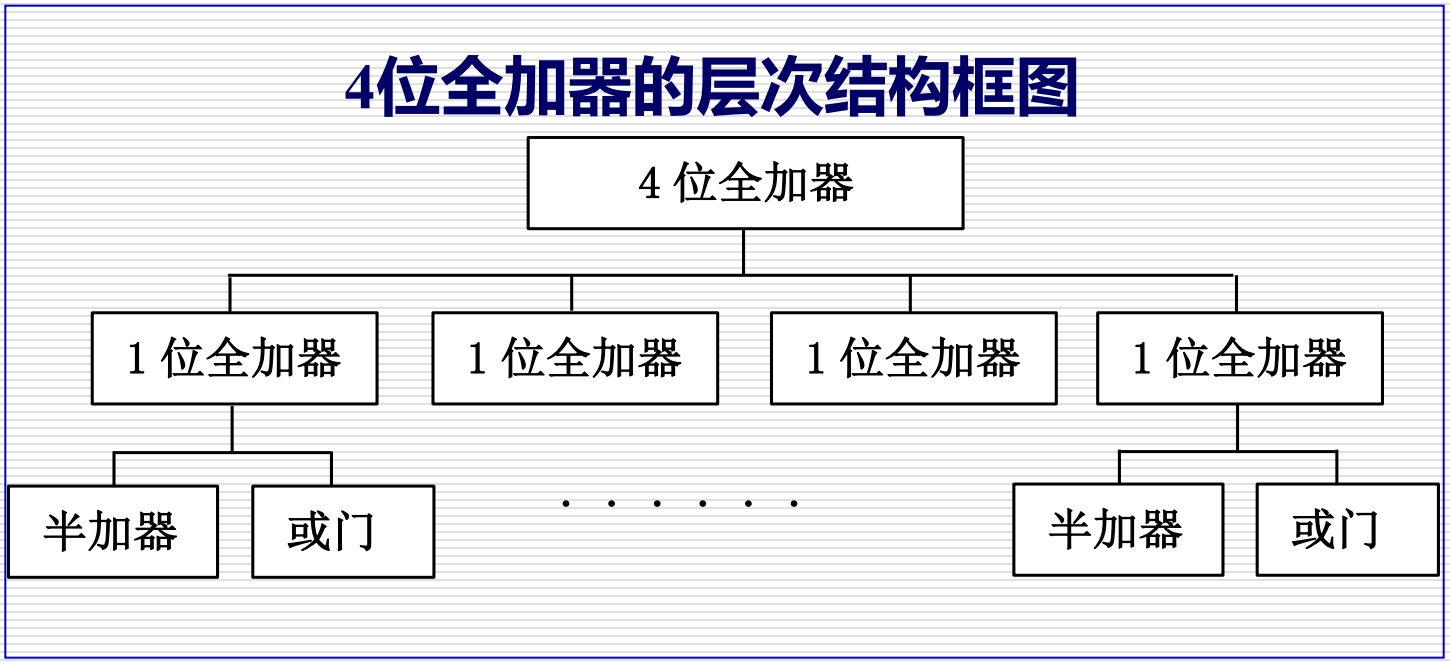
•结构描述方式：

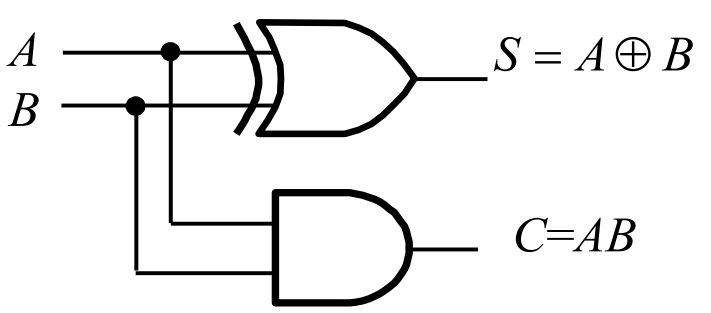
一般使用Primitive（内部元件）、自定义的下层模块对电路描述。主要用于层次化设计中。

2.5.8 分模块、分层次的电路设计

分层次的电路设计:在电路设计中，将两个或多个模块组合起来描述电路逻辑功能的设计方法。

设计方法：自顶向下和自底向上两种常用的设计方法



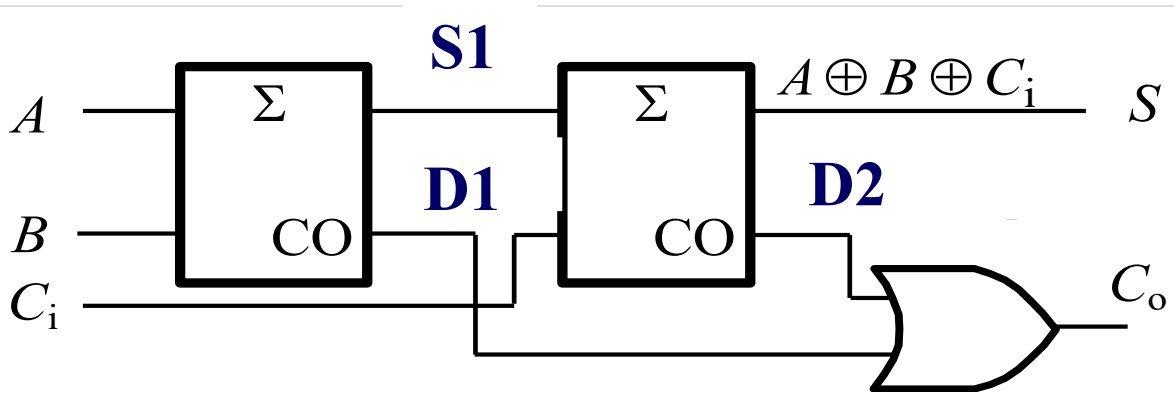



```
module halfadder (
    input A,B,
    output S,C
    );
    xor (S,A,B);
    and (C,A,B);
endmodule
```

半加器的门级描述

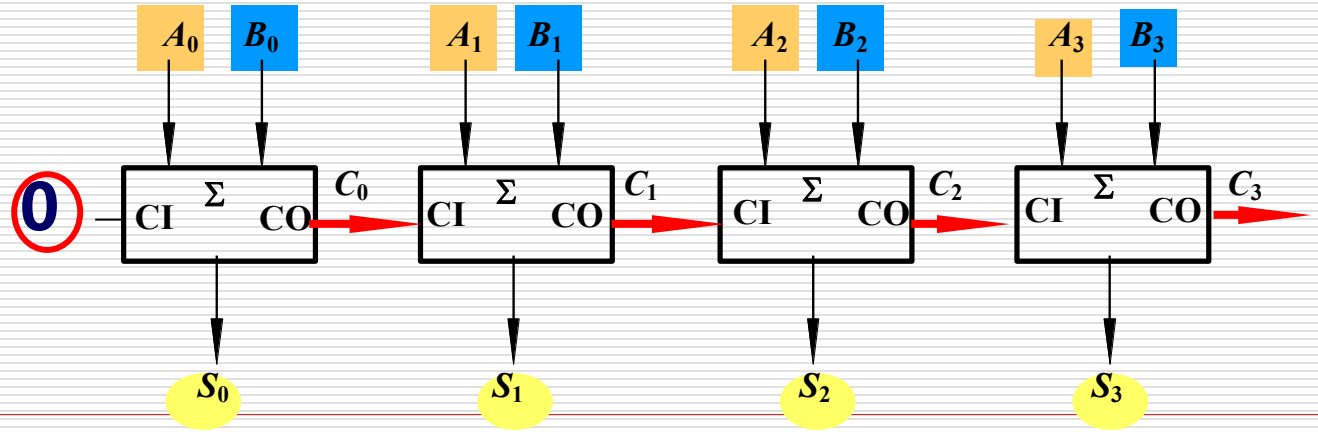
```
module fulladder (
    input A,B,CI,
    output S,CO
    );
    wire S1,D1,D2; //内部节点信号
    halfadder HA1 (.B(B),.S(S1),.C(D1),.A(A));
    halfadder HA2 (.A(S1),.B(Ci),.S(Sum),.C(D2));
    or  (Co,D2,D1);
    or g1(CO,D2,D1);
endmodule
```

全加器的描述-名称关联调用半加器,括号外是原名称，括号里是现在名称，位置时序可改变。



```
module _4bit_adder (S,C3,A,B,C_1);
        input [3:0] A,B;
        input C_1;
        output [3:0] S;
        output C3;
    wire C0,C1,C2;  //内部进位信号
fulladder  FA0 (S[0],C0,A[0],B[0],C_1),
           FA1 (S[1],C1,A[1],B[1],C0),
           FA2 (S[2],C2,A[2],B[2],C1),
           FA3 (S[3],C3,A[3],B[3],C2);
endmodule
```

4位全加器的描述--位置关联调用1位全加器，注意端口排列顺序



```
//带参数的4位加法器子模块
module adder #(paratermeter n=4 )
(
input [n-1:0] A,B,
input Ci,
output [n-1:0] S,
output Co
);
assign {Co,S} = A + B + Ci;

endmodule
```

4位全加器的描述

```
//按名称关联端口及参数
module adder_hier (
  input [7:0] A1,B1,
  input Cin1,
  output [7:0] Sum1,
  output Cout1
);
 adder #( .n(8)) U1
(
.A(A1),
.B(B1),
  .Ci(Cin1),
.S( Sum1),
.Co( Cout1)
);
```

8位全加器