EDA技术

第3章 组合电路设计



《EDA技术实用教程——Verilog HDL版》
(第七版)
配套课件

EDA Lab

内容提要

- 半加器电路的Verilog描述
- 多路选择器的Verilog描述
- Verilog加法器设计
- 组合逻辑乘法器设计
- RTL概念

EDA Lab

3.1 半加器电路的Verilog描述

$$SO = A \oplus B \quad ; \qquad CO = A \cdot B$$

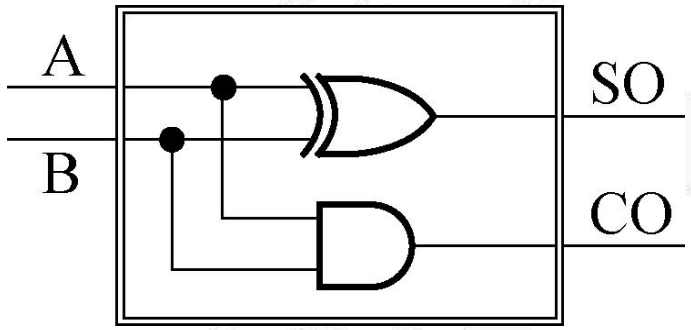


图3-1 半加器的电路结构

A	B	SO	CO
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

图3-2 半加器的真值表

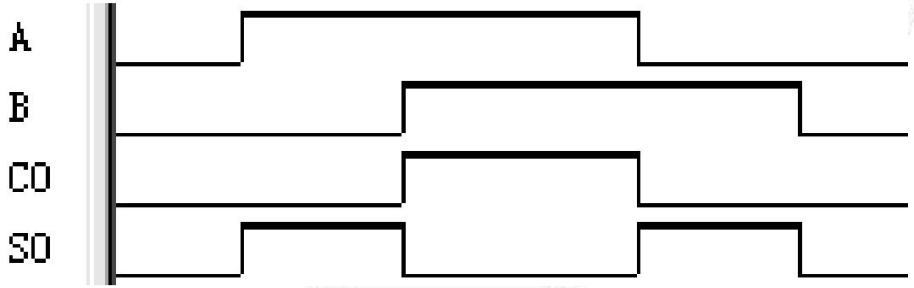


图3-3 半加器电路的时序波形

3.1 半加器电路的Verilog描述



【例3-1】

```
module h_adder (A,B,SO,CO) ;
      input A,B;
      output SO,CO;
      assign SO = A ^ B;     //将变量A和B执行异或逻辑后将结果赋给输出信号SO
      assign CO = A & B;     //将变量A和B执行与逻辑后将结果赋给输出信号CO
endmodule
```

半加器电路的Verilog描述具有三个部分：

1）以关键词module_endmodule引导的完整的电路模块
2）以input/output等关键词引导的对模块外部端口描述的语句
3）以assign引导的赋值语句

EDA Lab

3.1 半加器电路的Verilog描述



1. 模块语句及其表达方式

```
module   模块名  （模块端口名表）;
    模块端口和模块功能描述
endmodule
```

mux21a、counter4b 、h_adder √

or2、latch、74LS160、模块1 ×

EDA Lab

3.1 半加器电路的Verilog描述



2．端口语句、端口信号名和端口模式

```
input   端口名1,端口名2,... ;                    输入端口
output   端口名1,端口名2,... ;                   输出端口
inout   端口名1,端口名2,... ;                    双向端口，如RAM的数据口
input [msb : lsb]   端口名1,端口名2,... ;        多信号端口、总线端口
```

ZHOU DIANZI UNIVERSITY 1956

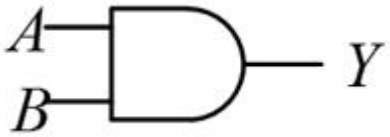
EDA Lab

3.1 半加器电路的Verilog描述

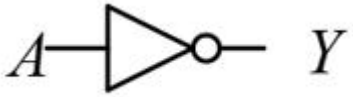
3．逻辑操作符

逻辑运算电路与Verilog描述

- 与：
- Verilog: `assign y = a & b;`



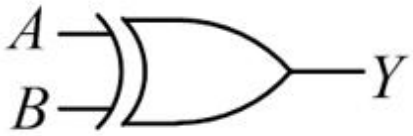
- 非：
- Verilog: `assign y = ~ a;`



- 或：
- Verilog: `assign y = a | b;`



- 异或：
- Verilog: `assign y = a ^ b;`

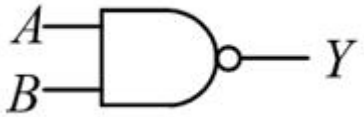


3.1 半加器电路的Verilog描述

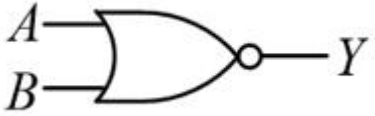
3．逻辑操作符

逻辑运算电路与Verilog描述

- 与非门（2输入）：
- Verilog: assign y = ~(a&b);



- 或非门（2输入）：
- Verilog: assign y = ~(a|b);



- 同或：
- Verilog: assign y = a ~^ b;



3.1 半加器电路的Verilog描述



4. 连续赋值语句

➡ assign 目标变量名 = 驱动表达式;

```
assign   y = a;          //将信号 a 向 y 赋值
assign   y = a & b ;     //将信号 a 和信号 b 逻辑相与后向 y 赋值
assign   y = (S? a : b);    条件表达式  ?  表达式1 : 表达式2
```

```
assign DOUT = a & b | c ;
assign DOUT = e & f | d ;
```



➡ assign [延时] 目标变量名 = 驱动表达式;

```
`timescale 10ns/100ps
 assign #6 R1 = A & B;
```

延时值在综合器中被忽略，不参与综合

EDA Lab

3.1 半加器电路的Verilog描述



5. 关键词

- 关键词全部用小写字母定义！
- 如always，assign，begin，case，casex，else，end，for，function，if，input，output，repeat，table，time，while，wire
- INPUT、OUTPUT、MODULE、ASSIGN 不是关键词

EDA Lab

3.1 半加器电路的Verilog描述



6．标识符

- 设计者在Verilog程序中自定义的、用于标识不同名称的词语。
- 标识符可由字母、数字、下划线和$符号构成；但第一个字符必须是字母或下划线，不能是数字或$符号！
- 在Verilog HDL中变量名是区分大小写的！

EDA Lab

3.1 半加器电路的Verilog描述

■合法的标识符:

```
Decoder_1, FFT, Sig_N, Not_Ack, State0, _Decoder_, REG
```

■非法的标识符:

```
2FFT          //Start with a number
Sig_#N        //"#" cannot be a component of an identifier
Not-Ack       //"-" cannot be a component of an identifier
data__BUS     //No double underscore in an identifier
reg           //Keyword
ADDER*        //"*" cannot be a component of an identifier
```

3.1 半加器电路的Verilog描述



7．注释符号

- /*　　*/　可换行
- //　　注释文字只能放在同一行

8．规范的程序书写格式

- 良好的代码风格
- 书上的书写格式是不推荐的

EDA Lab

3.1 半加器电路的Verilog描述



9．文件取名和存盘

- 路径与文件名中不包含中文与空格
- 文件后缀名必须是“.v”
- 文件取名的大小写也是敏感的，h_adder.v，H_ADDER.v不同
- 建议程序名和该程序的模块名一致，尤其对于Quartus必须满足！

EDA Lab

3.2 多路选择器的Verilog描述

3.2.1 4选1多路选择器及其case语句表述方式

4选1多路选择器的电路模型或元件图如图3-4所示；
图3-5是此模块电路（设此模块名是MUX41a）的时序波形。

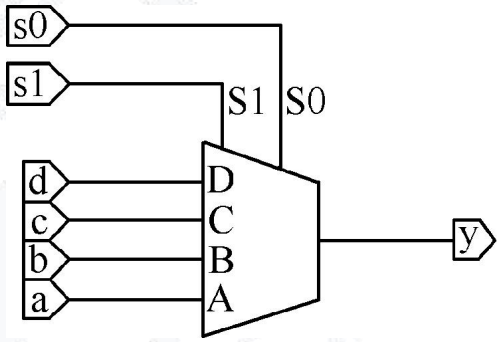


图3-4 4选1多路选择器

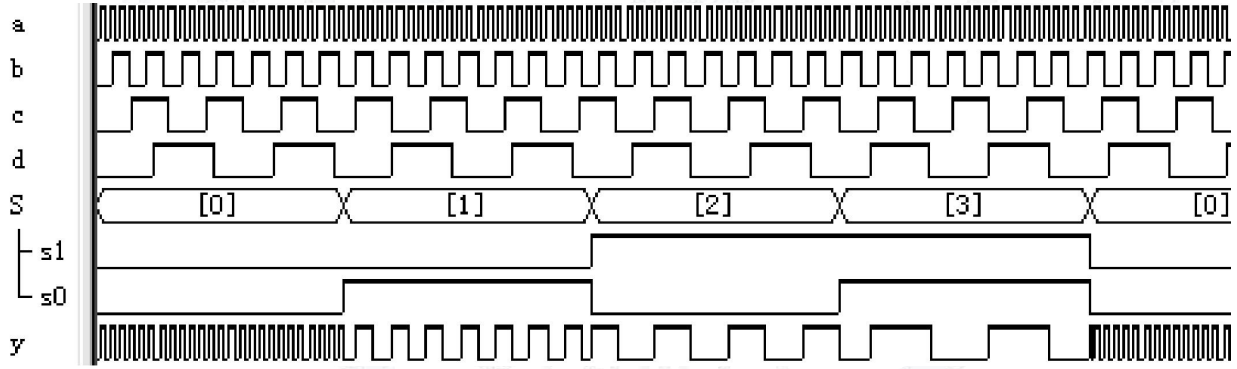


图3-5 4选1多路选择器MUX41a的时序波形

3.2 多路选择器的Verilog描述



3.2.1 4选1多路选择器及其case语句表述方式

【例3-2】

```
module MUX41a(a,b,c,d,s1,s0,y);
      input    a,b,c,d;
      input    s1,s0;
      output   y;
      reg  y;
      always@(a or b or c or d or s1 or s0)
         begin     :  MUX41    //块语句开始
             case({s1,s0})
                  2'b00: y=a;
                  2'b01: y=b;
                  2'b10: y=c;
                  2'b11: y=d;
                default: y=a;
             endcase
         end
endmodule
```

电路模块端口说明和定义段

信号类型定义段

具体描述电路功能的Verilog语句描述段

电路模块功能描述段

Verilog表述的可综合的完整电路模块

EDA Lab

3.2 多路选择器的Verilog描述



3.2.1 4选1多路选择器及其case语句表述方式

1. reg型变量定义

```
reg   变量名1，变量名2，... ;
reg   [msb:lsb] 变量名1，变量名2，... ;
```

```
module seg_7 (input[3:0] num, input en, output reg[6:0] seg);
```

- reg: 寄存器型变量；wire: 网线型变量。
- 如果未在模块中显式的定义信号类型，端口信号默认为wire型。
- 在过程语句always@引导的顺序语句中，被赋值信号规定必须是reg型变量。
- assign引导的并行赋值语句中信号类型规定为wire型。
- 输入或双向口信号不能定义为reg型。

EDA Lab

3.2 多路选择器的Verilog描述



3.2.1 4选1多路选择器及其case语句表述方式

2. 过程语句

- Verilog中有两类能引导顺序语句的过程语句，
  - always在可综合语句中最为常用
  - 另一类过程语句是initial语句

- 本章着重讲解always语句：

```
always @ （敏感信号及敏感信号列表或表达式）
        包括块语句的各类顺序语句
```

  - 通常要求将过程语句中所有的输入信号都放在敏感信号列表中

EDA Lab

3.2 多路选择器的Verilog描述



3.2.1 4选1多路选择器及其case语句表述方式

2. 过程语句

- 敏感信号
  - 用关键词or连接所有敏感信号，如(a or b or c or d or s1 or s0);
  - 用逗号区分或连接所有敏感信号，如(a, b, c, d, s1, s0);
  - 省略形式。always @ (*) 或 always @ *

EDA Lab

3.2 多路选择器的Verilog描述



3.2.1 4选1多路选择器及其case语句表述方式

2．过程语句

- 当某一敏感信号变化时，always @引导的语句被执行一遍，然后等待下一次敏感信号的变化。

- 对于一个module模块中的多个always @过程语句结构，本身都属于并行语句。而任一个过程内的语句结构属于顺序语句。

EDA Lab

3.2 多路选择器的Verilog描述



3.2.1 4选1多路选择器及其case语句表述方式

3．块语句begin _end

```
begin  [: 块名]
    语句1;语句2;. . . ;语句n;
end
```

- 块语句begin end仅限于在always引导的过程语句结构中使用，通常用来组合顺序语句块，相当于一个括号。
- 以上方括号中的“：块名”可以省略，如“: MUX41”

EDA Lab

3.2 多路选择器的Verilog描述



3.2.1 4选1多路选择器及其case语句表述方式

4. case条件语句

多分支case语句的一般格式：

```
case （表达式）
    取值1 : begin 语句1; 语句2; ... ; 语句n;     end
    取值2 :  begin 语句n+1; 语句n+2; ... 语句n+m;   end
      ...
    default : begin 语句n+m+1; ... ;  end
endcase
```

1956 HANGZHOU DIANZI UNIVERSITY

EDA Lab

3.2 多路选择器的Verilog描述



3.2.1 4选1多路选择器及其case语句表述方式

5. 并位操作运算符 {}

- 将两个或多个信号按二进制位拼接为一个数据信号来使用
- 可嵌套使用

```
{s1,s0}=2'b00
```

```
{a1,b1,4{a2,b2}}={a1,b1,{a2,b2},{a2,b2},{a2,b2},{a2,b2}}=
{a1,b1,a2,b2,a2,b2,a2,b2,a2,b2}
```

EDA Lab

3.2 多路选择器的Verilog描述



3.2.1 4选1多路选择器及其case语句表述方式

6. Verilog的数字表达形式

<位宽> '<进制> <数字>

B：二进制Binary， O：八进制Octal，

D：十进制Decimal， H：十六进制Hexadecimal

不分大小写，如：2’b10、4’B1011、4’HA等

EDA Lab

3.2 多路选择器的Verilog描述



3.2.2 4选1多路选择器及assign语句表述方式

【例3-3】

```
module MUX41a (a,b,c,d,s1,s0,y);
      input a,b,c,d,s1,s0;          output y;
      wire[1:0] SEL;                 // 定义2元素位矢量SEL为网线型变量wire
      wire AT,BT,CT,DT;             //定义中间变量，以作连线或信号节点
      assign SEL = {s1,s0};    //对s1,s0进行并位操作，即SEL[1]=s1;SEL[0]=s0
      assign AT = (SEL==2'D0);       assign BT = (SEL==2'D1);
      assign CT = (SEL==2'D2);       assign DT = (SEL==2'D3);
      assign  y = (a&AT)|(b&BT)|(c&CT)|(d&DT);//4个逻辑信号相或
endmodule
```

EDA Lab

3.2 多路选择器的Verilog描述

3.2.2 4选1多路选择器及assign语句表述方式

1. 按位逻辑操作符

表 3-1 逻辑操作符

逻辑操作符	功能	A,B 逻辑操作结果	C,D 逻辑操作结果	C,E 逻辑操作结果
~	逻辑取反	~A = 1'b1	~C = 4'b0011	~E = 6'b101001
|	逻辑或	A | B = 1'b1	C | D = 4'b1111	C | E = 6'b011110
&	逻辑与	A & B = 1'b0	C & D = 4'b1000	C & E = 6'b000100
^	逻辑异或	A ^ B = 1'b1	C ^ D = 4'b0111	C ^ E = 6'b011010
~^ 或 ^~	逻辑同或	A ~^ B = 1'b0	C ~^ D = 4'b1000	C ~^ E =6'b100101
设：A=1'b0；B=1'b1；C[3:0]=4'b1100；D[3:0]=4'b1011；E[5:0]=6'b010110				

3.2 多路选择器的Verilog描述



3.2.2 4选1多路选择器及assign语句表述方式

2. 等式操作符

表3-2 等式操作符

等式操作符	含义	等式操作示例
= =	等于	(3= =4)= 0; (A= =4'b1011)= 1; (B= =4'b1011)=0;
!=	不等于	(D!=C)= 0; (3!=4)= 1;
= = =	全等	(D= = =C)=1 ; (E= = =4'b0x10)= 0;
!= =	不全等	(E!= =4'b0x10)= 1;
设：A=4'b1011; B=4'b0010; C=4'b0z10; D=4'b0z10; E=3'bx10		

- 等式操作符运算的结果是1位逻辑值，采用逐位比较的方式。
- 对于双等于“==”，如位数不等则将较少位数的高位补0对齐，每位都相等才输出1，否则为0；如果有的位是未知值x或高阻值z，则判定为假，输出0。
- 对于全等于“===”，位数不等时不补零，直接判定为假；将x或z当成确定值进行比较，当表述完全相同时输出1。

EDA Lab

3.2 多路选择器的Verilog描述



3.2.2 4选1多路选择器及assign语句表述方式

3．wire定义网线型变量

- 如果assign语句中需要有端口以外的信号或连接线性质的变量（由于端口都已默认为网线型变量），则必须用网线型变量定义语句事先给出显式定义。

```
wire   变量名1, 变量名2,... ;
wire   [msb:lsb] 变量名1, 变量名2,... ;
```

- 还可直接用wire来表达assign语句

```
wire  a1, a2; assign  Y = a1 ^ a2;
```

EDA Lab

3.2 多路选择器的Verilog描述

3.2.3 4选1多路选择器及条件赋值语句表述方式

【例3-4】

```
module MUX41a (A,B,C,D,S1,S0,Y);
      input A,B,C,D,S1,S0;
      output Y;
      wire AT = S0 ? D : C ;
      wire BT = S0 ? B : A ;
      wire  Y = (S1 ? AT : BT);
endmodule
```

条件操作符的用法：

条件表达式？表达式1：表达式2

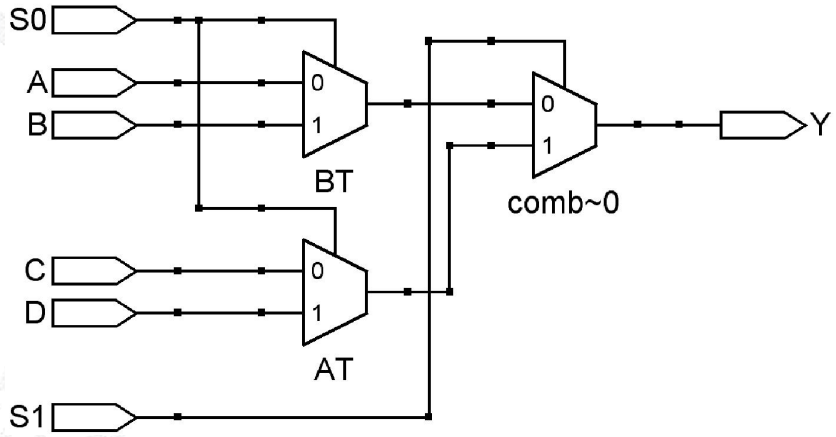


图3-6 例3-4的RTL电路图

3.2 多路选择器的Verilog描述



3.2.4 4选1多路选择器及其条件语句表述方式

【例3-5】

```
module MUX41a (A,B,C,D,S1,S0,Y);
     input A,B,C,D,S1,S0;      output Y;
     reg[1:0] SEL;     reg Y;
     always @(A,B,C,D,SEL)
       begin                              //块语句起始
            SEL = {S1,S0};                //把S1,S0并位为2元素矢量变量SEL[1:0]
         if (SEL==0)   Y = A;             //当SEL==0成立，即(SEL==0)=1时，Y=A;
   else  if (SEL==1)   Y = B;             //当(SEL==1)为真，则Y=B;
   else  if (SEL==2)   Y = C;             //当(SEL==2)为真，则Y=C;
   else                  Y = D;           //当SEL==3，即SEL==2'b11时，Y = D;
       end                                //块语句结束
endmodule
```

EDA Lab

3.2 多路选择器的Verilog描述



3.2.4 4选1多路选择器及其条件语句表述方式

1. if条件语句

- 例3-5条件表达式计算方法：
  - 当SEL等于00，则（SEL==0）=1, 满足条件，执行Y=A;
  - 当SEL不等于00，则（SEL==0）=0，随即执行else后的if语句。以如此顺序执行下去，直到完成所有if条件语句。
- 若只有一个条件，可表示为：

```
if (S)  Y=A;  else  Y=B;
```

- 当需执行多条语句，用begin_end块括起来：

```
if (S)  Y=A;  else   begin Y=B; Z=C; Q=1'b0; end
```

EDA Lab

3.2 多路选择器的Verilog描述



3.2.4 4选1多路选择器及其条件语句表述方式

2．过程赋值语句

（1）阻塞式（blocking）赋值“=”

如y=b;一旦执行完当前的赋值语句，赋值目标变量y即刻获得来自等号右侧表达式的计算值。

如果在一个块语句中含有多条阻塞赋值语句，而当执行到其中某条赋值语句时，其他语句被禁止执行，这时其他语句如同被阻塞了一样。

该语句特点与C语言等软件语言类似，都属顺序执行语句。

EDA Lab

3.2 多路选择器的Verilog描述



3.2.4 4选1多路选择器及其条件语句表述方式

2．过程赋值语句

（2）非阻塞式（non-blocking）赋值 “<=”

如y<=b; 特点是必须在块语句执行结束时才整体完成赋值操作。

在执行当前语句时，对于块中的其他语句的执行情况一律不加限制，不加阻塞。可理解为在begin块中的所有赋值语句都可以“并行”运行。

EDA Lab

3.2 多路选择器的Verilog描述



3.2.4 4选1多路选择器及其条件语句表述方式

3．数据类型表示方式

- 当所赋的值大于变量已定义的矢量位可能的值，综合器会首先将赋值符号右侧的数据折算成二进制数，然后根据被赋值变量所定义的位数，向左截去多余的数位
  - 如Y[1:0]，当赋值Y<=9时，(9=4‘b1001)，编译后，Y得到赋值为2’b01。

ZHOU DIANZI UNIVERSITY 1956

3.3 Verilog加法器设计

3.3.1 全加器设计及例化语句应用

1. 全加器原理图结构

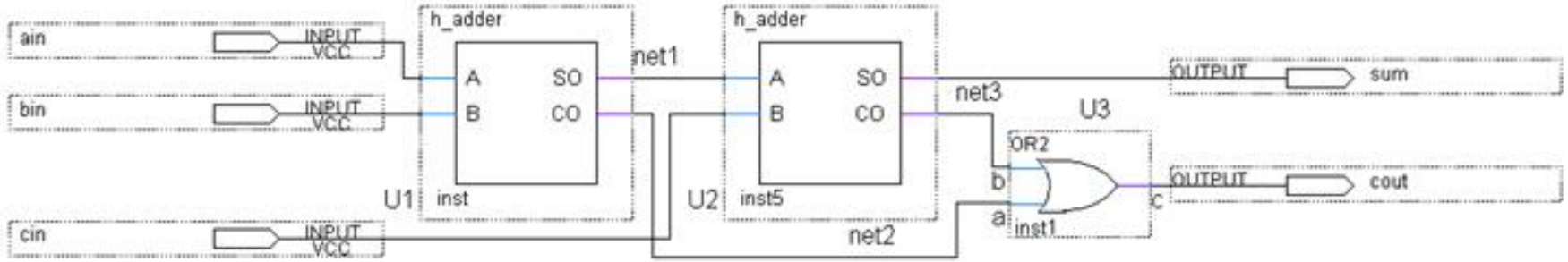


图3-7 全加器f_adder电路图

sum=第一个半加器的和，加上Cin (即两个半加器的级联，最终的和输出)。

cout=第一个半加器的进位，“或”第二个半加器的进位。

3.3 Verilog加法器设计



3.3.1 全加器设计及例化语句应用

```
module h_adder(a,b,so,co); //半加器描述(3)：直接用算符完成的描述
    input a,b ;
    output so,co ;
    assign {co,so} = a + b;  //两位二进制数直接相加，进位进入并位后的 co
endmodule
```

```
module or2a(a,b,c);          //或门逻辑描述
    output c;  input a,b ;
    assign c = a | b ;
endmodule
```

EDA Lab

3.3 Verilog加法器设计

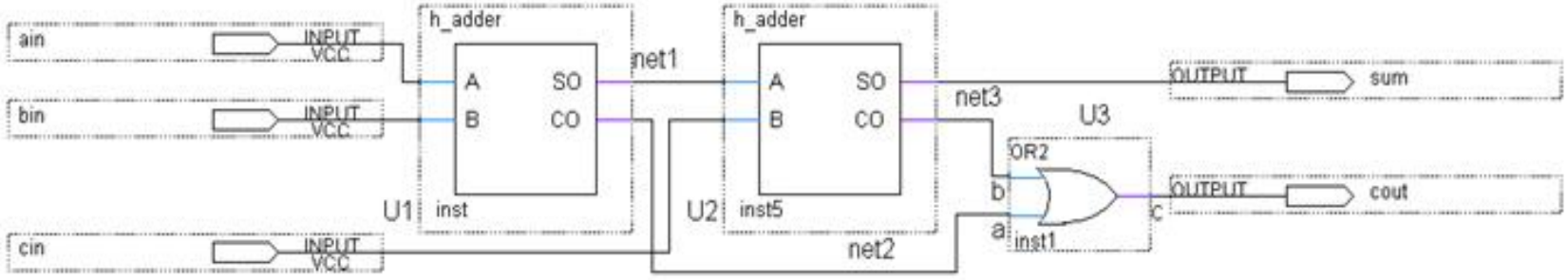
3.3.1 全加器设计及例化语句应用

2. 全加器顶层设计文件

【例3-6】

```
module f_adder(ain, bin, cin, cout, sum);
   output cout, sum;    input ain, bin, cin;
    wire  net1, net2, net3;
    h_adder  U1(ain, bin, net1, net2);
    h_adder  U2(.A(net1), .SO(sum), .B(cin), .CO(net3));
        or   U3(cout, net2, net3);
endmodule
```

用顶层的总模块连接涉及的底层子模块，通过例化语句实现。



3.3 Verilog加法器设计



3.3.1 全加器设计及例化语句应用

3. Verilog例化语句及其用法

⌘ 元件例化（Instantiation）：引入一种连接关系，将预先设计好的设计模块定义为一个元件，然后利用特定的语句将此元件与当前的设计实体中指定端口相连接，从而为当前设计实体引进一个新的、低一级的设计层次。

<模块元件名> <例化元件名>（.例化元件端口（例化元件外接端口名）,...）;

EDA Lab

3.3 Verilog加法器设计



3.3.1 全加器设计及例化语句应用

3. Verilog例化语句及其用法

信号名映射法（端口名关联法）：括号中的信号名是外部端口名，括号外带点的信号名是待连接的元件自己的端口名，位置不影响连接结果。

```
h_adder  U2(.A(net1), .SO(sum), .B(cin),.CO(net3));
h_adder  U2(.B(cin), .CO(net3), .A(net1), .SO(sum));
```

位置关联法，关联表述的信号位置十分重要，注意端口顺序，不能放错。

```
h_adder  U1(ain, bin, net1, net2);
```

EDA Lab

3.3 Verilog加法器设计

3.3.2 8位加法器设计及算术操作符应用

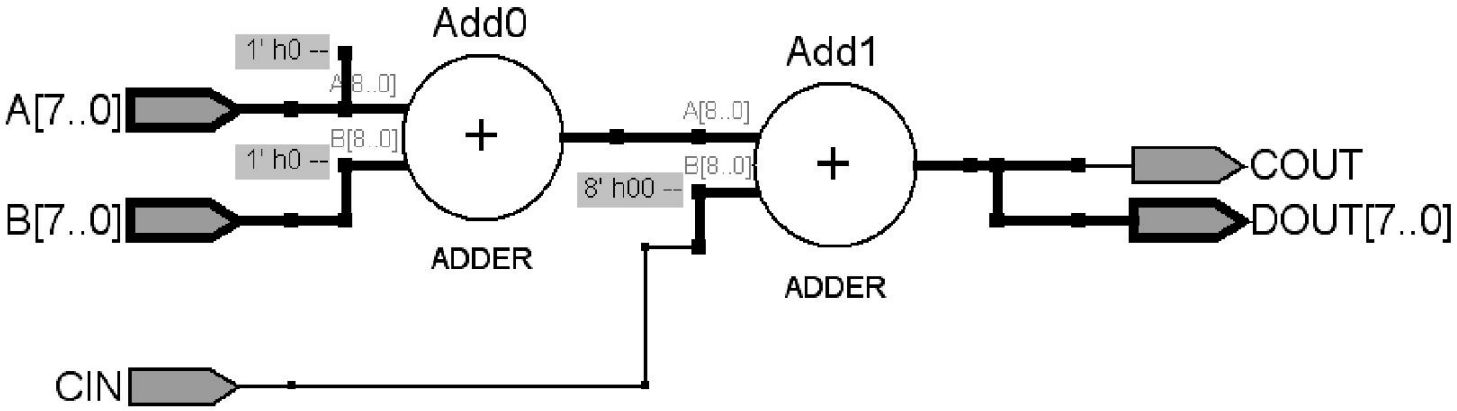


图3-10 8位加法器的RTL电路图

3.3 Verilog加法器设计



3.3.2 8位加法器设计及算术操作符应用

【例3-7】

```
module
    ADDER8B(A,B,CIN,COUT,DOUT);
    output[7:0] DOUT;  output COUT;
    input[7:0] A,B; input  CIN;
    wire[8:0]  DATA;
    //加操作的进位自动进入DATA[8]
    assign DATA = A + B + CIN;
    assign  COUT = DATA[8];
    assign  DOUT = DATA[7:0];
endmodule
```

【例3-8】

```
module
    ADDER8B (A,B,CIN,COUT,DOUT);
    output[7:0] DOUT;
    output COUT;
    input[7:0] A,B;
    input  CIN;
    //加操作的进位进入并位COUT
    assign {COUT,DOUT} = A + B + CIN;
endmodule
```

EDA Lab



3.3.3 算术运算操作符

表3-3 算术操作符

算术操作符	功能	说明	操作示例
+	加		S = A + B = 8'b00011000
-	减		S = B - A = 8'b11111110
*	乘		S = A * B = 8'b10001111=2'H8F
/	除	结果：小数抛弃	S = A / 3 = 8'b00000100
%	求余	除法求余数	S = A%3 = 8'b00000001
设： A[3:0]=4'b1101；B[3:0]=4'b1011；定义S为S[7:0]			

所有算数运算都是按无符号操作数进行的，如果是减法运算，输出的结果是补码。对于乘法，若为无符号数，可直接用乘法算符（*）；若为有符号数乘，则需将操作数和输出结果用signed定义为有符号数，乘法结果为补码。

3.3 Verilog加法器设计



3.3.4 BCD码加法器设计

8421BCD码就是十进制数的4位二进制码。BCD码加法是十进制加法，不存在A（4'b1010）~F(4'b1111)这几种状态，超过9随即进一位，相当于二进制加6。

一位BCD码: 0~9

Hexadecimal --> BCD:

If Hexadecimal <10, BCD=Hexadecimal;

If Hexadecimal >=10, BCD=Hexadecimal+6

EDA Lab

3.3 Verilog加法器设计

【例3-10】

```
module BCD_ADDER (A,B,D);
  input[7:0] A,B;    output[8:0] D;
  wire[4:0] DT0, DT1 ; reg[8:0] D; reg S;
 always@ (DT0)
     begin  if (DT0[4:0] >= 5'b01010)
    //如果低位BCD码的和大于等于10,则使和加上6，且有进位，使进位标志S等于1
                begin D[3:0] = (DT0[3:0]+4'b0110); S=1'b1; end
                 else  begin D[3:0] = DT0[3:0] ; S=1'b0; end
     end //否则，将低位值赋予低位BCD码D[3:0]输出，无进位，使进位标志S等于0
 always@ (DT1)    begin
     if (DT1[4:0]>=5'b01010)
    begin D[7:4] = (DT1[3:0]+4'b0110); D[8]=1'b1; end
        else begin D[7:4] = DT1[3:0] ; D[8]=1'b0;  end      end
   assign DT0 = A[3:0] + B[3:0];              //设没有来自低位的进位
   assign DT1 = A[7:4] + B[7:4] + S;        //S是来自低位BCD码相加的进位
endmodule
```

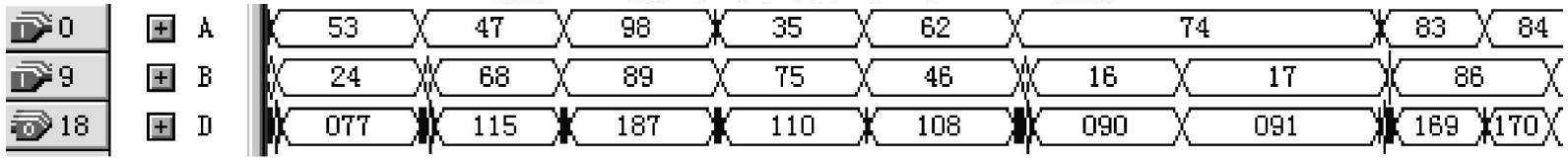


图3-12 例3-10的仿真波形

3.3 Verilog加法器设计

表3-4 不等式操作符

不等式操作符	含义	操作示例
>	大于	（A > B）= 1; （A > 12）=1;
<	小于	（A < B）= 0; （A < 20）= 1;
<=	小于或等于	（A <= 13）= 1;
>=	大于或等于	（A >= 14）= 0;
设：A=4'B1101， B=4'B0110		

3.4 组合逻辑乘法器设计



3.4.1 参数定义关键词parameter和localparam

- 在Verilog中，用关键词parameter来定义参数常量，如延时、变量的位宽等，从而成为一个符号化常量，便于改变整个设计
- parameter的一般定义格式：

```
parameter   标识符名1 = 表达式或数值1, 标识符名2 = 表达式或数值2, ... ;
```

```
parameter   A=15, B=4'b1011, C=8'hAC;
parameter   d=8'b1001_0011, e=8'sb10101101;
```

在模块中使用参数定义的常数只能被赋值一次。

- localparam：局部参数定义关键词，无法通过外部程序的数据传递来改变其定义的常量

EDA Lab

3.4 组合逻辑乘法器设计



3.4.2 整数型寄存器类型定义

```
integer   标识符1,标识符2,...,标识符n [msb: lsb];
```

- 定义为integer类型的变量多数被用于表达循环变量，用于指示循环次数。
- 与reg类型的定义不同，reg类型必须明确定义其位数，但integer类型的定义不必特指位数，因为其被默认为32位宽的二进制寄存器类型。

EDA Lab

3.4 组合逻辑乘法器设计



3.4.3 for语句用法

for（循环初始值设置表达式； 循环控制条件表达式；循环控制变量增值表达式）

begin 循环体语句结构 end

（1）根据“循环初始值设置表达式”获得循环次数初始值。

（2）根据“循环控制条件表达式”判断是否满足继续循环的条件

（3）根据“循环控制变量增值表达式”计算出循环控制变量的数值，然后跳到以上步骤（2）。

EDA Lab

3.4 组合逻辑乘法器设计

3.4.4 移位操作符及其用法

➡ V >> n 或 V <<n　右移位操作符 或 左移位操作符

右移或左移，移出腾空的
位用0填补。

➡ V >>> n 或 V <<< n　有符号数右移或左移

有符号数左右移，右移一律将符号位，即最高位填补移出的位。左移操作同普通左移<<

3.4 组合逻辑乘法器设计



3.4.5 两则乘法器设计示例

【例3-11】

```
module MULT4B(R,A,B);
 parameter S=4;
 output[2*S:1] R;
 input[S:1] A,B;
 reg[2*S:1] R;
 integer i;
   always @(A or B)
    begin
    R = 0;
    for(i=1;  i<=S; i=i+1)
    if(B[i])  R=R+(A<<(i-1));
    end
endmodule
```

【例3-12】

```
module MULT4B (R,A,B);
 parameter S=4;
 output[2*S:1] R;
 input[S:1] A,B;
 reg[2*S:1] R,AT; reg[S:1] BT,CT;
 always @(A,B )   begin
    R=0; AT = {{S{1'B0}},A};
    BT = B;  CT = S;
    for(CT=S; CT>0; CT=CT-1)
      begin  if(BT[1]) R=R+AT;
        AT = AT<<1;
        BT = BT>>1;
end  end   endmodule
```

EDA Lab

3.4 组合逻辑乘法器设计

3.4.5 两则乘法器设计示例

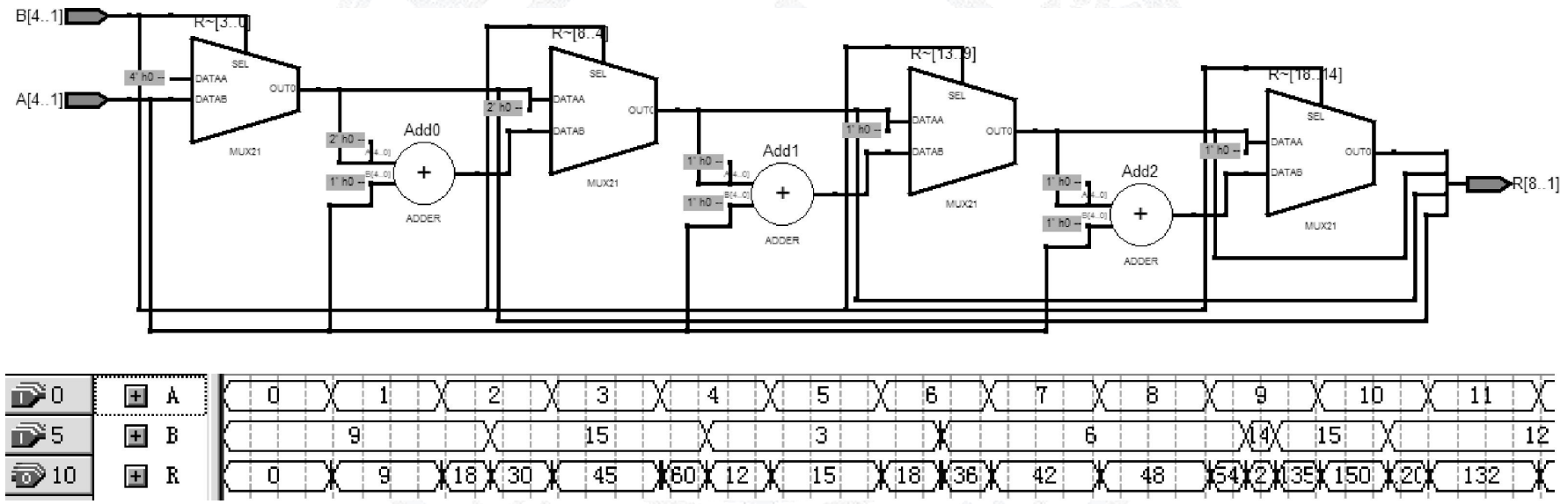


图3-13 上图是4位乘法器的RTL逻辑电路图，下图是其时序仿真图

3.4 组合逻辑乘法器设计



3.4.6 repeat语句用法

```
repeat（循环次数表达式）
  begin    循环体语句结构   end
```

“循环次数表达式”可以是数值确定的整数、变量或定义了常数的参数标识符等。

repeat语句的循环次数是在进入此语句执行以前就已决定的，无需循环次数控制增量表达式及其计算。

【例3-13】

```
module MULT4B(R,A,B);
  parameter S=4;
  output[2*S:1] R;
input [S:1] A,B;
  reg[2*S:1] TA,R;
  reg[S:1] TB;
    always @(A or B)
     begin
     R = 0; TA = A;  TB = B;
     repeat(S) begin
      if(TB[1]) begin R=R+TA; end
        TA = TA<<1;
        TB = TB>>1;
                end
    end
endmodule
```

EDA Lab

3.4 组合逻辑乘法器设计



3.4.7 while语句用法

```
while（循环控制条件表达式）
  begin   循环体语句结构 end
```

（1）根据“循环控制条件表达式”判断是否满足继续循环的条件。

（2）对于此种循环语句，必须在“循环体语句结构”中包含类似for语句的“循环控制变量增值表达式”。

【例3-14】

```
module MULT4B(A,B,R);
  parameter S=4;
  input[S:1] A,B;
  output[2*S:1] R;
  reg[2*S:1] R,AT;
  reg[S:1] BT,CT;
  always@(A or B)
    begin
    R=0; AT={{S{1'b0}},A};
     BT=B; CT=S;
     while(CT>0)  begin
      if(BT[1]) R=R+AT; else R=R;
begin  CT= CT-1;
       AT=AT<<1;
       BT=BT>>1;  end
                          end
    end
endmodule
```

EDA Lab

3.4 组合逻辑乘法器设计



3.4.8 parameter的参数传递功能

parameter的目的除了能用标识符定义一些常数外，另一重要用途是通过例化语句来传递参数。以便仅通过上层设计中的相关参数的改变来轻易地改变底层电路的结构功能与逻辑规模。

```
module MULT4B #(parameter S=4)(R,A,B);
```

或

```
module MULT4B #(parameter S)(R,A,B);
```

【例3-15】

```
module MULT4B
 #(parameter S)(R,A,B);
  output [2*S:1] R;
 input[S:1] A,B;
  reg[2*S:1] AT,R;
  reg[S:1] BT,CT;
        ...   //以下与例3-16相应部分相同
```

【例3-16】

```
module MULTB (RP,AP,BP);
 output[15:0] RP;
 input[7:0] AP,BP;
   MULT4B #(.S(8))
  U1(.R(RP), .A(AP), .B(BP));
endmodule
```

EDA Lab

3.4 组合逻辑乘法器设计

3.4.8 parameter的参数传递功能

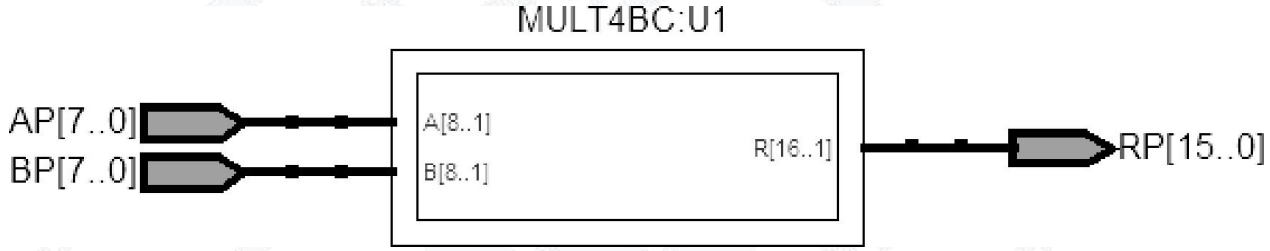


图3-14 例3-16的RTL图，内部结构如图3-15所示

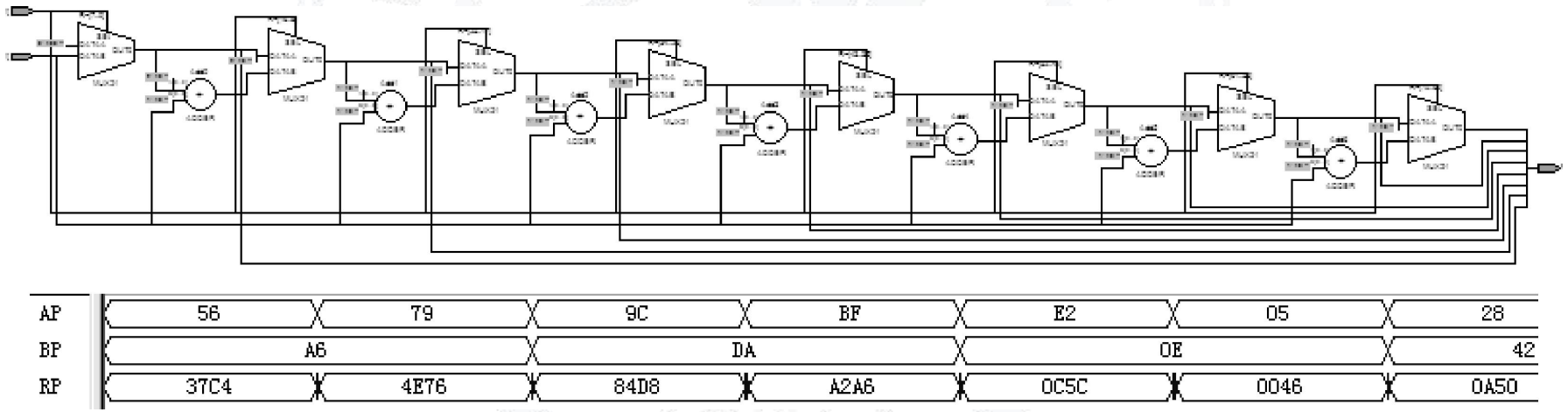


图3-15 上图是8位乘法器的RTL逻辑电路图，下图是其仿真波形图

3.4 组合逻辑乘法器设计



3.4.8 parameter的参数传递功能

例如若底层文件的模块语句和参数表述是

```
module SUB_E
  #(parameter S1=4, parameter S2=5, parameter S3=2)(A,B,C);
```

则在例化语句中应作如下类似表述：

```
SUB_E  #(.S1(8), .S2(9), .S3(7))
         U1(.C(CP), .A(AP), .B(BP));
```

EDA Lab

3.5 RTL概念

RTL（Register Transport Level）的概念最初产生于对某类电路的描述

- 传统概念下的RTL电路的构建特色是由一系列组合电路模块和寄存器模块相间级联而成。
- 此后，RTL引申为一切用各种独立的组合电路模块和独立的寄存器模块，但不涉及底层具体逻辑门结构或触发器电路细节，来构建描述数字电路的形式。

HANGZHOU DIANZI UNIVERSITY

3.5 RTL概念



- 进入EDA技术领域后，有了进一步的引申，且向两个不同的方向分化：

（1）RTL描述。一切用可综合的语句表述的、可由综合器生成实际电路的HDL代码形式称为RTL描述。

（2）RTL级或RTL电路。将RTL表述确定为数字系统表述的某一层次级别，即RTL级。

- 习惯上将功能仿真称为RTL级仿真，时序仿真称为门级仿真
- 对应综合而言，通常将可综合的Verilog描述称为RTL描述

EDA Lab

3.5 RTL概念

- 数字电路描述层次（设计层次），按序排列
  - 系统级（ESL）
  - 行为级（BL）
  - RTL（寄存器船速级）
  - 门级
  - 开关机（管子级/MOS级）
  - 物理级别（版图级）

1956 XIDIAN UNIVERSITY
EDA Lab