



“十二五”普通高等教育本科国家级规划教材

EDA技术实用教程

——Verilog HDL版(第五版)

潘 松 黄继业 潘 明 编著



- 讲技术，授技能，求职就业的帮手
- 布情景，述过程，教学改革的能手
- 举示例，重实践，能力培养的强手



科学出版社

日期: /

PLD (可编程逻辑器件)

1. 分类 { ① 按集成度分类 { 简单 < 复杂

② 按结构分类 { 乘积项结构器件 \Rightarrow 与-或阵列
基于查找表结构器件

③ 按编程工艺划分 { 熔丝型器件

反熔丝型器件

EPROM型: 紫外线擦除

EEPROM型: 电可擦除

SRAM型:

Flash型

2. 逻辑单元符号表示




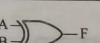



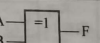
	非门	与门	或门	异或门
IEEE-1991版 标准逻辑符号	 \bar{A}	 F	 F	 F
IEEE-1984版 标准逻辑符号	 \bar{A}	 F	 F	 F
逻辑表达式	$\bar{A} = \text{NOT } A$	$F = A \cdot B$	$F = A + B$	$F = A \oplus B$

图 2-3 两种不同版本的国际标准逻辑门符号对照图

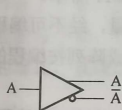


图 2-4 PLD 的互补缓冲器

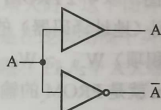


图 2-5 PLD 的互补输入

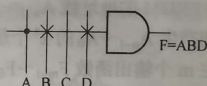


图 2-6 PLD 中与阵列的表示

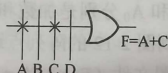


图 2-7 PLD 中或阵列的表示



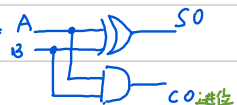
图 2-8 阵列线连接表示

日期:

器件名

组合电路的Verilog设计

一 半加器 (h-adder)

1. 电路结构:  $\Rightarrow S0 = A \oplus B$
 $CO = A \cdot B$

2 真值表:

A	B	S0	CO
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

波形图:

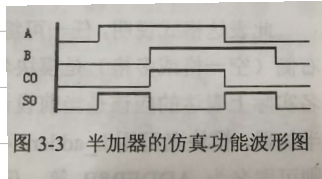


图 3-3 半加器的仿真功能波形图

3 代码: (模块)

模块名称

module h-adder (A, B, S0, CO)

端口表

器件的个何都即所有输入输出

input A, B;

input/output/inout 即输入/输出
后缀 [msb:lsb] 代表有 msb-lsb+1 位

output S0, CO;

成对出现

assign S0 = A ^ B

assign CO = A & B

endmodule

所有的 assign 同时执行 \Rightarrow 数据流描述方式
引导赋值语句 连续赋值语句

延时: 'timescale 10ns/100ps \rightarrow 仿真时间精度

assign #6 R1 = A & B; 计算出值后隔6个时间单元再赋值
惯性延时

日期: /

二、多路选择器:

1. 4选1多路选择器及 case 语句表述方法

```
module MUX41a(a,b,c,d,s1,s0,y);
```

```
input a,b,c,d;
```

```
input s1,s0;
```

```
output y;
```

定义为变量
寄存器型变量

```
reg y;
```

y在always中
要定义为reg

敏感信号表,里面的信号一旦改变则重新执行一遍
可以不用写,不能用其调整功能

```
always @(a or b or c or d or s1 or s0)
```

```
begin: MUX41
```

块级名称

引导块级

```
case ({s1,s0})
```

case必须放在always中执行
把s1,s0合并成一个2位数(并位操作运算符)

```
2'b00: y <= a;
```

非阻塞式赋值,

```
2'b01: y <= b;
```

always块后所有的<=语句同时赋值
一个变量不可有多个非阻塞式赋值

```
2'b10: y <= c;
```

```
2'b11: y <= d;
```

```
default: y <= a;
```

可靠,防止机器以为y值不变
一般要写完整不然会出现时序电路

```
endcase
```

```
end
```

可去掉

```
endmodule
```

日期: /

2. 4选1多路选择器及assign语句表达

```
module MUX4a (a,b,c,d,s1,s0,y);
```

```
    input a,b,c,d,s1,s0;
```

```
    output y;
```

```
    wire [1:0] SEL; 网线型变量
```

```
    wire AT, BT, CT, DT; 中间变量
```

```
    assign SEL = {s1,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);
```

```
endmodule
```

XT表示选择哪一个
XT=1 则输出X

日期: /

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	$\sim A = 1'b1$	$\sim C = 4'b0011$	$\sim E = 6'b101001$
	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	$A \wedge B = 1'b1$	$C \wedge D = 4'b0111$	$C \wedge E = 6'b011010$
~^ 或 ~^	逻辑同或	$A \sim \wedge B = 1'b0$	$C \sim \wedge D = 4'b1000$	$C \sim \wedge E = 6'b100101$

: $A=1'b0$; $B=1'b1$; $C[3:0]=4'b1100$; $D[3:0]=4'b1011$; $E[5:0]=6'b010110$

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	$(E!=4'b0x10)=1$;

每位相等
设: $A=4'b1011$; $B=4'b0010$; $C=4'b0z10$; $D=4'b0z10$; $E=3'bx10$

日期: /

3. 4选1多路选择器及条件赋值语句

```
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
```

$\frac{S_1 S_0}{2}$

$S_1 S_0$	Y
00	A
01	B
10	C
11	D

4. 4选1多路选择器及条件语句表达方式

```
【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  
        SEL = {S1,S0};  
        if (SEL==0) Y = A;  
        else if (SEL==1) Y = B;  
        else if (SEL==2) Y = C;  
        else Y = D;  
    end  
endmodule
```

//块语句起始
//把 S1,S0 并位为 2 元素矢量变量 SEL[1:0]
//当 SEL==0 成立, 即 (SEL==0)=1 时, Y=A;
//当 (SEL==1) 为真, 则 Y=B;
//当 (SEL==2) 为真, 则 Y=C;
//当 SEL==3, 即 SEL==2' b11 时, Y = D;
//块语句结束

阻塞式赋值

日期:

/

5. 利用UDP元件设计多路选择器

口排列顺序必须与例3-9一致

【例 3-9】

```
primitive
MUX41_UDP(Y,D3,D2,D1,D0,S1,S0);
input D3,D2,D1,D0,S1,S0; output Y;
table //D3 D2 D1 D0 S1 S0 : Y
    ? ? ? 1 0 0 : 1;
    ? ? ? 0 0 0 : 0;
    ? ? 1 ? 0 1 : 1;
    ? ? 0 ? 0 1 : 0;
    ? 1 ? ? 1 0 : 1;
    ? 0 ? ? 1 0 : 0;
    1 ? ? ? 1 1 : 1;
    0 ? ? ? 1 1 : 0;
endtable
endprimitive
```

【例 3-10】

```
module MUX41UDP (D,S,DOUT);
input[3:0] D;
input[1:0] S;
output DOUT;
MUX41_UDP (DOUT,D[3],D[2],
            D[1],D[0],S[1],S[0]);
endmodule
```


日期: /

三、加法器:

1. 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
```

加数 被加数 → 进位输入 → 进位输出 → 进位
顺序交换
第2个半加器的SO与SUM接在一起
例化名

2. 半加器的UDP结构建模描述方式.

【例 3-7】	【例 3-8】
<pre>primitive XOR2(DOUT,X1,X2); input X1,X2; output DOUT; table // X1 X2 : DOUT 0 0 : 0; 0 1 : 1; 1 0 : 1; 1 1 : 0; endtable endprimitive</pre>	<pre>module H_ADDER (A,B,SO,CO); input A,B; output SO,CO; XOR2 U1(SO,A,B); // 调用元件 XOR2 and U2(CO,A,B); // 调用元件 and endmodule</pre>

日期: /

3. 8位加法器及算术操作符

【例 3-11】

```
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 3-12】

```
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
```

算术操作符

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]

BCD加法器

```
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
    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
  end
endmodule
```

日期: /

两条语句 module MULT4B(R,A,B)和 parameter S=4 改

四. 乘法器

1. 参数定义关键词

① parameter: 定义常量

parameter A=15, B=4'b1011;

② localparam 局部参数定义

2 整数型寄存器类型定义

integer: 多用于表达 循环变量指示循环次数, 默认 32 位宽

3. for 语句

for (循环初始值; 循环控制条件; 循环控制变量增值)

begin

...

end

【例 3-15】

```
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 3-16】

```
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
```

→ for 语句实现的 4 位乘法器

4. 移位操作符

$V \gg n \rightarrow V$ 右移 n 位

$V \ggg n \rightarrow V$ (有符号数) 右移 n 位

日期: /

5. repeat语句

repeat (循环次数表达式)

begin

...

end

6. while 语句

while ()

begin

end

【例 3-17】

```
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
```

【例 3-18】

```
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
        endmodule
```

日期: /

时序电路的Verilog设计

基本D触发器的Verilog表述:

```
module DFF1 (clk, D, Q);
```

```
output Q;
```

```
input clk, D;
```

```
reg Q;
```

```
always @ (posedge clk)
```

```
Q <= D;
```

```
end module
```

↑ 上升沿有效, posedge的信号且下always包含模块中没有出现则认定为时钟信号

D触发器的UDP表示

【例 5-2】

```
primitive EDGE_UDP (Q,D,CLK,RST);
input D,CLK,RST; output Q; reg Q;
table//D CLK RST : Q : Q+
0 (01) 0 : ? : 0;
1 (01) 0 : ? : 1;
? (1?) 0 : ? : -;
? (?0) 0 : ? : -;
1 0 1 : ? : 0;
1 1 1 : ? : 0;
0 0 1 : ? : 0;
0 1 1 : ? : 0;
endtable
endprimitive
```

【例 5-3】

```
module DFF_UDP (Q,D,CLK,RST);
input D,CLK,RST;
output Q;
EDGE_UDP U1 (Q,D,CLK,RST);
endmodule
```

↓
(01) 表示上升沿触发

(10) 下降沿触发

日期: /

含异步复位和时钟使能的D触发器 — Verilog

↓
RST

↓
EN → 只有EN=1时, 上升沿才有效

独立于时钟控制的复位控制端
无论何时只要RST=0则清零

```
module DFF2 (CLK, D, Q, RST, EN);
```

```
    output Q;
```

```
    input CLK, RST, EN;
```

```
    reg Q;
```

```
    always @ (posedge CLK or negedge RST)
```

```
    begin
```

```
        if (!RST) Q <= 0;
```

```
        else if (EN) Q <= D;
```

```
    end
```

```
end module
```

日期: /

→ 上升沿清零

同步复位控制的D触发器 — Verilog

控制信号只有在时钟信号有效时才起作用

由RST控制多路选择器 RST=0时为0信号, RST=1时为, 清零信号

法一: module DFF3 (CLK, D, Q, RST);

Output Q;

input CLK, D, RST;

reg Q;

always @(posedge CLK)

if (RST == 1) Q = 0;

else if (RST == 0) Q = D;

endmodule

法二: module DFF1 (CLK, D, Q, RST);

Output Q;

input CLK, D, RST;

reg Q, Q1;

always @(RST)

if (RST == 1) Q1 = 0; else Q1 = D;

always @(posedge CLK)

Q <= Q1;

endmodule

日期: /

基本锁 — verilog

↓ CLK为高电平时有效

```
module LATCH1 (CLK, D, Q);
```

```
    output Q;
```

```
    input CLK, D;
```

```
    reg Q;
```

```
    always @(D or CLK)
```

```
        if (CLK) Q <= D;
```

```
endmodule
```

含清0控制的锁存器 ~ verilog

【例 5-9】

```
module LATCH2 (CLK, D, Q, RST);  
    output Q;    input CLK, D, RST;  
    assign Q = (!RST)? 0: (CLK ? D: Q);  
endmodule
```

【例 5-10】

```
module LATCH3 (CLK, D, Q, RST);  
    output Q;  
    input CLK, D, RST;  
    reg Q;  
    always @(D or CLK or RST)  
        if (!RST) Q <= 0;  
        else if (CLK) Q <= D;  
endmodule
```


日期: /

异步时序电路 ~ Verilog

过程 2 被启动。

【例 5-11】

```
module AMOD(D,A,CLK,Q);  
  output Q; input A,D,CLK;  
  reg Q,Q1;  
  always @(posedge CLK) Q1 = ~(A|Q);  
  always @(posedge Q1) Q = D;  
endmodule
```

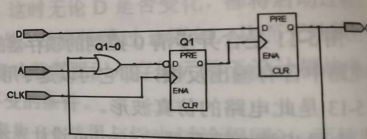


图 5-14 例 5-11 的时序电路图

时序模块的 Verilog 设计规律:

- ① 若将 A 定义为边沿敏感时钟信号就必需加上 `posedge A/negedge A` 但在 `always` 过程中不能出现 A
- ② 若将 B 定义为电平敏感异步控制信号需用 `posedge A/negedge A` 且在 `always` 过程中明示 B 的逻辑行为
- ③ 时钟控制的信号不能出现在敏感表中
- ④ 敏感信号表中不允许出现混合信号

日期: /

二 进制计数器的 Verilog 表示

1. 简单加法计数器 → 4位 从 0000 → 1111 来一个时钟信号就加 1

【例 5-13】

```
module CNT4 (CLK, Q);  
    output[3:0] Q;    input  CLK;  
    reg[3:0] Q1;  
    always @(posedge CLK)  
        Q1 <= Q1+1;  
    assign Q=Q1;  
endmodule
```

【例 5-14】

```
module CNT4 (CLK, Q);  
    output[3:0] Q;  
    input  CLK;  
    reg[3:0] Q;  
    always @(posedge CLK)  
        Q <= Q+1;  
endmodule
```

2. 实用加法计数器

程序中，对于相关语句的用法。
点关注程序中的 if 语句的用法。

【例 5-15】

```
module CNT10 (CLK, RST, EN, LOAD, COUT, DOUT, DATA);  
    input CLK, EN, RST, LOAD;    // 时钟, 时钟使能, 复位, 数据加载控制信号  
    input[3:0] DATA;            // 4 位并行加载数据  
    output[3:0] DOUT;            // 4 位计数输出  
    output COUT;                 // 计数进位输出  
    reg[3:0] Q1;    reg COUT;  
    assign DOUT = Q1;            // 将内部寄存器的计数结果输出至 DOUT  
    always @(posedge CLK or negedge RST) // 时序过程  
    begin  
        if (!RST) Q1 <= 0;    // RST=0 时, 对内部寄存器单元异步清 0  
        else if (EN) begin    // 同步使能 EN=1, 则允许加载或计数  
            if (!LOAD) Q1 <= DATA;    // 当 LOAD=0, 向内部寄存器加载数据  
            else if (Q1 < 9) Q1 <= Q1+1;    // 当 Q1 小于 9 时, 允许累加  
            else Q1 <= 4'b0000; end    // 否则一个时钟后清 0 返回初值  
        end  
    always @(Q1)                // 组合过程  
        if (Q1==4'h9) COUT = 1'b1; else COUT = 1'b0;  
endmodule
```

日期: /

移位寄存器 Verilog

1. 含同步预置功能的移位寄存器

【例 5-16】

```
module SHFT1(CLK,LOAD,DIN,QB);  
    output QB; input CLK,LOAD; input[7:0] DIN; reg[7:0] REG8;  
    always @(posedge CLK)  
        if (LOAD) REG8<=DIN; else REG8[6:0]<=REG8[7:1];  
    assign QB = REG8[0];  
endmodule
```

CLK 上升沿来时, LOAD 为高电平 将 DIN 作为预置数据放入

来一个时

2. 使用移位操作符设计移位寄存器

【例 5-16】

```
module SHFT1(CLK,LOAD,DIN,QB);  
    output QB; input CLK,LOAD; input[7:0] DIN; reg[7:0] REG8;  
    always @(posedge CLK)  
        if (LOAD) REG8<=DIN; else REG8[6:0]<=REG8[7:1];  
    assign QB = REG8[0];  
endmodule
```

日期: /

可预置型计数器

1. 同步加载计数器 → 4位

【例 5-18】

```
module FDIW0 (input CLK,RST,input[3:0]D,output PM,output[3:0]DOUT);
    reg[3:0] Q1;    reg FULL;

    (* synthesis,keep *) wire LD; //设定 LD 为仿真可测试属性
    always @(posedge CLK or negedge RST)
        if (!RST) begin Q1<=0; FULL<=0; end
        else if (LD) begin Q1<=D; FULL<=1; end
        else begin Q1 <= Q1+1; FULL<=0; end
    assign LD=(Q1==4'b1111); assign PM=FULL; assign DOUT=Q1;
endmodule
```

2. 异步加载计数器

【例 5-20】

```
module fdiv1 (CLK,PM,D);
    input CLK; input[3:0] D;    output PM; reg FULL;
    (* synthesis,probe_port,keep *) reg[3:0] Q1;
    (* synthesis,probe_port,keep *) wire RST;
    always @(posedge CLK or posedge RST)
        if (RST) begin Q1<=0;
            FULL<=1; end
        else begin Q1<=Q1+1;
            FULL<=0; end
    assign RST = (Q1==D);
    assign PM = FULL;
endmodule
```

同步清0加载计数器

always @(posedge CLK)

日期: /

Verilog 状态机