EDA技术

第12章 Verilog知识拾遗

《EDA技术实用教程——Verilog HDL版》（第七版）

配套课件

EDA Lab

12.1 Verilog文字规则

- 在Verilog程序中，其值不能被随意改变的量称为常量或常数，主要包括三种类型：整数类型（integer）、实数类型（real）和字符串类型（strings）。
- 其中，整数类型可综合，实数和字符串类型不可综合

12.1 Verilog文字规则

1 整数

- 较长的数间可以用下划线分开，如10’b10_1010_1011，但数字最高位前和位宽、进制处不能用
- 若不注明位宽和进制，或仅用D注明进制，都是十进制数，默认32位，如

-7’d30=7’H62，-5=32’hFFFFFFFB,

123=32’h0000_007B, ‘D12=32’h0000000C

- 若未注明某整数的位宽，仅标注了数制，则其宽度即为此数制规定的数值中对应的位数，如

’o466(=‘b1_0011_0110)=9’o466

12.1 Verilog文字规则

1 整数

- 整数可以在其签名，即左边带正负符号，但不能放在数制内。如

  7’d-30错误，-7’d30正确， -7’d30=7’H62

- 如果定义的位宽比实际短，则多余位数从高位被截，如

  5’b11011011实际是5’b11011

- 如果定义的位宽比实际长，则从高位补0；如果此数最高位的一位为x或z，则相应的用x或z在高位补位，如

  5’b11实际为5’b00011，5’bx1xz实际为5’bxx1xz

12.1 Verilog文字规则

1 整数

- reg [3:0] A; reg [5:0] B; reg [31:0] C;

A<=6'B11_0110; //A实际获得赋值4'B0110，高2位被截去。进制符号b或B大小写都可

A<='o466; //'o466='H136，A实际获得低4位：4'B0110。高位被截去

A<=123; //123=32'h0000_007B，转换为32位二进制数，A实际获得赋值4'B1011

A<=8'hAC; //A实际获得赋值4'h1100，高4位被截去

C<=-5; //-5=32'hFFFFFFFB，A即获得赋值32'hFFFFFFFB

B<=-7'd30; //-7'd30=7'H62，A实际获得赋值=6'H22，高1位被截去

12.1 Verilog文字规则

2 实数

- 实数属于十进制数，通常其表述必须带上小数点，且小数点两侧必须有数字

  错误 17.

  正确 1.335 88_670_551.453_909(= 88670551.453909)

  1.0 44.99e-2(=0.4499) 0.1 3E-4(=0.0003)

- Verilog可通过四舍五入将实数转为整数

  13.44 13.43 → 13

  7

  51.6 51.68 → 52

  9

  - → -23

  23.34

12.1 Verilog文字规则

3 字符串

- 包括文字字符串、数位字符串，是一维字符数组，需放在双引号中

（1）文字字符串是用双引号括起来的一串文字，字符串不能分成多行书写

“ERROR” “Both S and Q equal to 1”

“X” “BB&CC”

（2）数位字符串也称位矢量，与文字字符串类似，但所代表的是二进制、八进制、十六进制的数组

12.1 Verilog文字规则

4 标识符

- 可以是常数、变量、信号、端口、参数名等
- 有效字符：26个英文字母、0~9数字、&、_ 及其组合，最长1203个字符
- 任何标识符必须以英文字母或下划线开头
- 单一下划线，前后必须有英文字母或数字
- 字母区分大小写
- 允许包含图形符号和空格

12.1 Verilog文字规则

- 合法的标识符:

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

- 非法的标识符:

```
2FFT            // 起始为数字
Sig_#N          // 符号“#”不能成为标识符的构成
Not-Ack         // 符号“-”不能成为标识符的构成
data_ _BUS      // 标识符中不能有双下划线
reg             // 关键词
ADDER*          // 标识符中不允许包含字符 *
```

12.2 数据类型

- 数据类型是Verilog用来表示数字电路硬件中的物理连线、数据存储对象和传输单元等。
- 包括网线型（net）和寄存器型（reg）
- 注：Verilog-1995标准中的reg数据类型被Verilog-2001标准中的variable类型替代，或variable型变量即称为reg型

12.2 数据类型

2.1 net网线类型

01

定义为网线型(net)的变量常被综合成硬件电路中的物理连接，其特点是输出紧跟输入变化，因此常被用来表示以assign关键词引导的组合电路描述

02

net另一个使用场合是在结构描述中将其连接到一个门元件或模块的输出端，如net变量未连接到驱动，其值为高阻z。

12.2 数据类型

12.2.1 net网线类型

- Verilog程序模块中输入输出型变量都默认为net型变量的子类型wire
- net型包含wire、tri、supply0、supply1等四种可综合的子类型
  - tri与wire功能用法相同
  - supply0、supply1表示地线（逻辑0）、电源线（逻辑1）

EDA Lab

12.2.2 register寄存器类型

- 具有寄存器特性，即在接受下一次赋值前，可以保持原值不变

- reg类型必须放在initial或always引导的过程语句中

EDA Lab

12.2.2 register寄存器类型

- 包含五种数据类型
  - 可综合的有reg、integer
  - 不可综合的有
    - 时间寄存器类型time，定义32位带符号整形寄存器变量
    - 实数寄存器类型real，定义64位带符号实数形寄存器变量
    - 实数时间寄存器类型realtime，定义64位带符号实数形寄存器变量

12.2.3 memory存储器类型

- 可看成由reg定义的二维矢量，即由一组寄存器构成的阵列，若干个相同宽度的寄存器矢量（一维矢量）构成的阵列构成一个存储器
  - reg [7:0] mem[1023:0];
  - $readmemh(“sinrom.dat”

12.3 操作符

- Verilog的操作符包括逻辑操作符、算术操作符、关系操作符、等式操作符、条件操作符、位操作符、缩位操作符、移位操作符、位拼接操作符等
- 按操作符所带的操作数的个数，分为：
  - 单目操作符：带一个操作数，如逻辑取反“~”
  - 双目操作符：带两个操作数，如逻辑与“&”
  - 三目操作符：带三个操作数，如条件操作符“?:”

12.3 操作符

1. 逻辑操作符

- 逻辑操作符属于双目操作符
- 逻辑操作符有三类，逻辑与：&& 逻辑或：|| 逻辑非：！。
- 逻辑非“！”属于单目操作符，“&&”和“||”都属于双目操作符，且优先级高于算术运算操作符。

12.3 操作符

1. 逻辑操作符

- 逻辑操作符与按位逻辑操作符的区别是，逻辑操作符对应的操作数如果是位矢量，则无论有多少位，操作后输出只有1位，即1或0。如果是1，说明声明的关系为真；如果是0，则说明声明的关系为假。

12.3 操作符

1. 逻辑操作符

- 逻辑操作符的操作特点就是，首先分别对两操作数位矢中的所有位进行按位逻辑或操作，最后对两数进行逻辑操作。

  - A=4’b1001, B=4’b0001
  - 首先计算 1 | 0 | 0 | 1 =1
  - 然后计算 0 | 0 | 0 | 1 =1
  - 最后计算 1&1 =1

  按位逻辑或

  A && B=(1|0|0|1) & (0|0|0|1)=1&1=1’b1

  A||B = 1|1 = 1’b1　　! A = ~1 =0

12.3 操作符

1. 逻辑操作符

- 如果一个矢量中除了0外还有z，则认为它是逻辑z，有如下关系：
  - 1&z = 1'bz，0&z = 1'b0，1|z = 1'b1，0|z = 1'bz

12.3 操作符

2. 缩位操作符

- 缩位操作符有6种类型，包括&（与）、~&（与非）、|（或）、~|（或非）、^（异或）、^~，~^（同或）。
- 缩位操作符属于单目操作符，其操作的输出结果也是一个位。

EDA Lab

12.3 操作符

2. 缩位操作符

- 缩位运算符的操作方法与位操作符的逻辑运算法则一样，但是缩位运算符是对单个位矢（如果是的话）操作数中的每一个位值进行与、或、非递推运算。表述时操作符放在操作数的前面。缩位运算符将一个位矢缩减为一个标量。
- 例如，若A=8'b11101111，则 &A=1&1&1&0&1&1&1&1=0，因为只有A的各位都为1时，其与的缩减操作值才为1。

12.4 常用语句补充

- Verilog主要的编程语句包括过程语句、块语句、赋值语句、条件语句（if、case）、循环语句等
- 不同的HDL综合器所支持的HDL语句不尽相同，可综合的语句只是HDL语句的一部分
- 随着EDA技术的发展，可综合的Verilog正逐渐走向标准化

12.4 常用语句补充

12.4.1 initial过程语句使用示例

- 用于产生指定激励信号的测试模块

【例12-1】

```
`timescale 1ns/100ps        //声明仿真时间单位是1ns，仿真精度也是100ps
module test;                //名为test的测试模块
reg A, B, C;
initial                     //定义initial过程语句结构
  begin
   A=0;B=1;C=0     //在过程中分别定义A、B、C在时刻0的初始值
  #50  A=1;B=0;    //经过50ns延时后，在仿真时刻50ns时A和B的输入值分别是1,0
  #50  A=0;C=1;    //又经过50ns延时后，在时刻100ns时A和C的输入值分别是0,1
  #50  B=1;        //再经过50ns延时后，在时刻150ns时B的输入值分别是1
  #50  B=0;C=0    //再经过50ns延时后，在时刻200ns时B和C的输入值都是0
  #50  $finish    //又经过50ns延时后，结束
  end
endmodule
```

·timescale 仿真时间单位/仿真时间精度，注意：

（1）仿真时间单位和仿真时间精度取值范围仅三种，即1/10/100；单位是秒s/毫秒ms/微秒us/纳秒ns/皮秒ps

（2）设计前了解所使用的综合器是否要求在程序前加·timescale语句。对于ModelSim等仿真器则必须加入此语句

12.4 常用语句补充

12.4.2 forever循环语句

```
forever 语句;

forever begin 语句; end
```

- forever循环语句可以连续不断的执行其后的语句或语句块，从而产生周期性的波形，作为仿真激励信号
- forever循环语句通常在initial语句中使用
- 解决前面例子中的繁琐波形生成方式

12.4 常用语句补充

12.4.3 编译指示语句

- Verilog允许在程序中使用特定的编译指示语句（compiler directives）
- 在综合前，通常先对编译指示语句进行预处理，再将预处理的程序和源程序一并交付综合器进行编译
- 编译指示语句及被定义后调用的宏名都以符号“`”开头
- Verilog的编译指示语句除了`define，还包括`include、`ifdef、`else、`endif

12.4 常用语句补充

1. 文件包含语句`include

`include 功能是将一个文件全部包含到另一个文件中，格式如下

`include “文件名”

【例12-2】

```
`include "h_adder.v"
`include "or2a.v"
    module f_adder(input ain,bin,cin,output cout,sum);
      wire  e,d,f ;
    h_adder  u1(ain, bin, e, d);
    h_adder  u2(.a(e), .so(sum), .b(cin),.co(f));
         or2a  u3(.a(d), .b(f),     .c(cout));
    endmodule
```

使用`include 调用半加器和或门模块，对于Quartus来说，由于其综合器会自动根据例化语句的表述在工作库中调用例化语句指示的模块，文件包含语句是非必要的

12.4 常用语句补充

1. 文件包含语句`include

使用`include语句需注意：

（1）一条`include语句只能指定一个被包含的文件，语句中要给出全面和后缀

（2）`include语句可出现在程序的任何地方

EDA Lab

12.4 常用语句补充

1. 文件包含语句`include

（3）如果被包含的文件不在当前工程所在文件夹中，需标明此文件的路径，如`include “e:/ADDER/h_adder.v”

（4）`include语句的文件包含允许多层次包含，如文件1包含文件2，文件2又包含文件3

（5）不同编译器和综合器对`include语句的要求不同

EDA Lab

12.4 常用语句补充

2. 条件编译语句`ifdef、`else、`endif

- 条件编译语句`ifdef、`else、`endif的功能是，命令综合器将此语句指定的部分参与Verilog源程序一同编译综合，有两种使用格式

```
`ifdef 宏名
    语句块
`endif
```

功能：当用`define语句定义的宏名在程序中被定义，则由语句`ifdef、`endif涵盖的Verilog语句块可以参与源文件的编译综合，否则该语句不参加编译综合

```
`ifdef 宏名
    语句块1
`else 语句块2
`endif
```

功能：当用`define语句定义的宏名在程序中被定义，则语句块1将被编译到源文件中参与综合，否则语句块2将被编译到源文件中参与综合。

12.4 常用语句补充

【例12-3】

```
`define AND
module andd (out,A,B);
    input [1:0] A,B;
        output [1:0] out;
        `ifdef AND
            assign out=A&B;
    `else assign out=A|B;
        `endif
endmodule
```

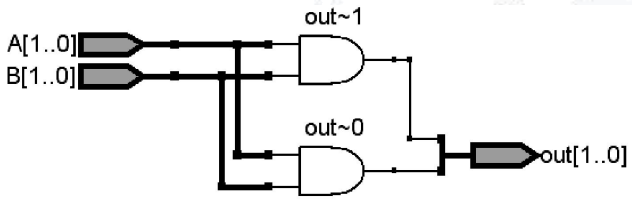


图12-1 对应例12-3的RTL图

`define语句定义了宏名AND，命令语句中也规定了宏名AND，则执行“assign out=A&B”

【例12-4】

```
`define OR1
module andd (out,A,B);
    input [1:0] A,B;
        output [1:0] out;
        `ifdef AND
            assign out=A&B;
    `else assign out=A|B;
        `endif
endmodule
```

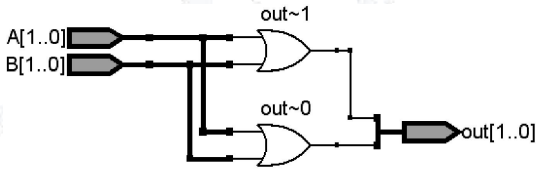


图12-2 对应例12-4的RTL图

`define语句定义了宏名OR1，命令语句中也规定了宏名AND，则执行“assign out=A|B”

12.4 常用语句补充

12.4.4 任务task和函数function语句

- 任务和函数具备将程序中的反复被用的语句结构聚合起来的能力，因此其功能类似于C语音的子程序
- 通过任务和函数语句结构来替代重复性大的语句可以有效的简化程序结构
- 任务和函数可以把一个大的程序模块分解成许多小的任务函数，以利于调试

EDA Lab

12.4.4 任务task和函数function语句

1. 任务语句task

task定义语句格式

```
task <任务名>;
    端口及数据类型声明语句
    begin 过程语句; end
endtask
```

task调用格式

```
<任务名> (端口1,端口2,…,端口N);
```

任务语句中不能出现由always或initial引导的过程语句结构，因此在任务中无法描述时序电路，可综合的任务语句结构只能描述组合电路

12.4.4 任务task和函数function语句

1. 任务语句task

【例12-5】

```
module TASKDEMO (S,D,C1,D1,C2,D2);   //主程序模块及端口定义
input S;   input[3:0] C1,D1,C2,D2;
output[3:0] D;                       //端口定义数目不受限制
reg[3:0] out1,out2;
task CMP;                            //任务定义，任务名CMP，此行不能出现端口定义语句
 input[3:0] A,B;  output[3:0] DOUT;  //注意任务端口名的排序
 begin if (A>B) DOUT=A;             //任务过程语句描述一个比较电路
  else DOUT=B; end                  //在任务结构中可以调用其他任务或函数，甚至自身
 endtask                            //任务定义结束
 always @ (*) begin                 //主程序过程开始
CMP(C1,D1,out1);                    //调用一次任务。任务调用语句只能出现在过程结构中
CMP(C2,D2,out2);   end              //第二次调用任务
assign D=S? out1:out2;
endmodule
```

12.4.4 任务task和函数function语句

1. 任务语句task

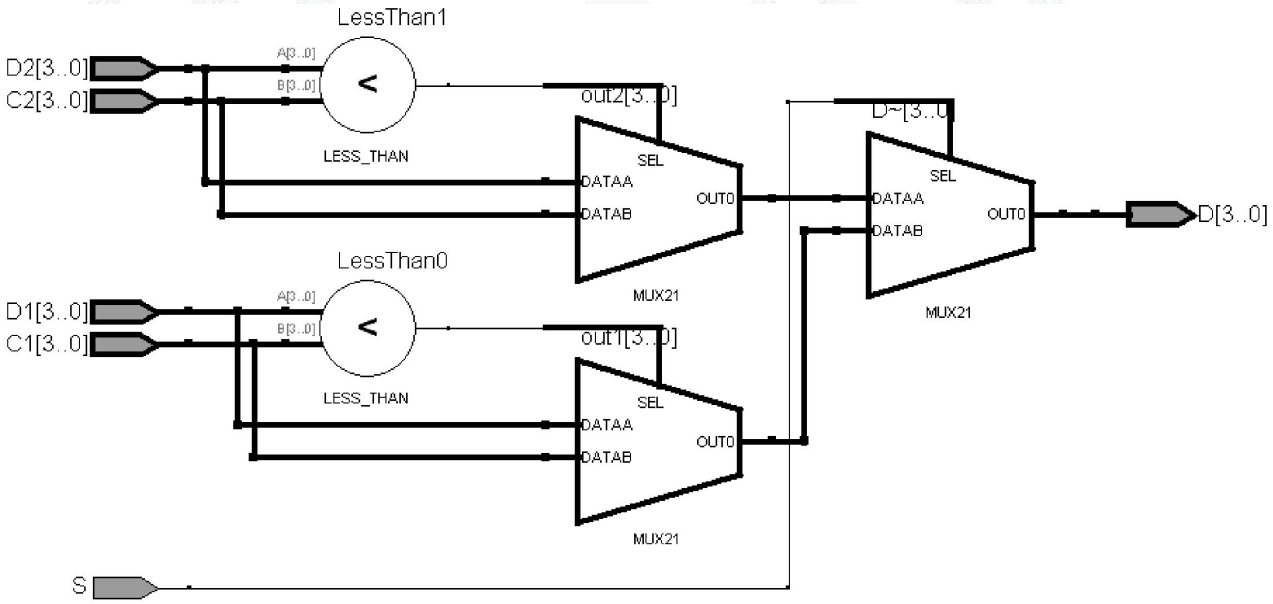


图12-3 例12-5的RTL图

12.4.4 任务task和函数function语句

2. 函数语句function

函数定义语句格式

```
function <位宽范围声明> 函数名;
      输入端口说明, 其他类型变量定义;
      begin   过程语句;    end
endfunction
```

函数调用格式

<函数名> (输入参数1, 输入参数2, …)

EDA Lab

12.4.4 任务task和函数function语句

2. 函数语句function

【例12-6】

```
module CN (input[3:0] A, output[2:0] OUT);
function[2:0] GP;    //定义一个函数名为GP的函数，GP同时作为位宽为3的输出参数
input[3:0] M;         //M定义为此函数的输入值，位宽是4
 reg[2:0] CNT,N;
 begin CNT=0;  for(N=0; N<=3; N=N+1)        //for循环语句
if(M[N]==1)  CNT=CNT+1;  GP=CNT;  end        //含1的位个数累加
endfunction
assign OUT=(~|A) ? 0:GP(A); //主程序输入A或非缩位，若为1则输出函数计数结果
endmodule
```

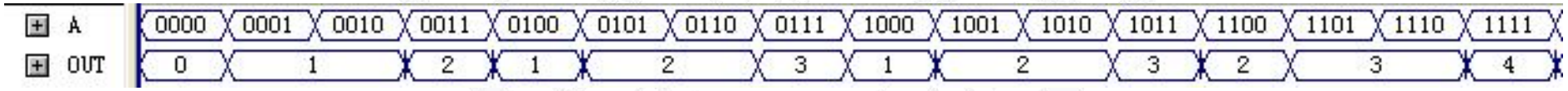


图12-4 例12-6的仿真图

12.4.4 任务task和函数function语句

2. 函数语句function

函数定义应该注意：

（1）函数定义语句只能放在模块中，不能放在过程结构中。

（2）函数内部可以调用函数，但不可调用任务。

（3）由于被调用的函数相当于一个操作数，所以在过程语句和连续赋值语句中都可以调用函数。

（4）同样是由于被调用的函数是一个操作数，所以不能作为语句单独出现。

12.5 库元件和UDP用法介绍

- Verilog程序模块中，可以通过例化方法直接调用库中的基本元件或模块，将它们连接起来实现既定的功能，即结构建模方式，无需描述电路的行为或功能
- 如果描述的元件不在库中，则必须进行元件创建，并将其放在元件库中，以便后来的设计可以从库中调用

EDA Lab

12.5 库元件和UDP用法介绍

12.5.1 Verilog原语库元件与用法

基于结构描方式的Verilog程序可通过以下方式构建电路并实现其功能：

1) 调用Verilog内置的基本门元件（门级结构描述）
2) 调用开关级元件（晶体管级结构描述）
3) 用户自定义元件（门级结构描述）
4) 通过例化方式调用以不同方式表述的模块元件，属于更常用的结构描述

EDA Lab

12.5 库元件和UDP用法介绍

12.5.1 Verilog原语库元件与用法

对于调用Verilog内置的基本门元件：

Verilog中预先定义的基本逻辑单元称为原语（primitive），即现成的门级库元件，包括and/nand/or/nor/xor/xnor/not等，特点是有一个到多个输入端口，只有一个输出端口，且默认输出口的位置排在最左侧，如

```
and (out, in1, in2)
and (out, in1, in2, in3)
```

EDA Lab

12.5 库元件和UDP用法介绍

12.5.1 Verilog原语库元件与用法

- Verilog中定义了26个基本元件，包括14个门级元件和12个开关级元件
- 14个门级元件：
  1) 多输入门类6个：与门and、与非门nand、或门or、或非门nor、异或门xor、同或门xnor
  2) 多输出门类2个：缓冲门buf、非门not
  3) 三态门类4个：高电平使能三态门bufif1、低电平使能三态门bufif0、低电平使能三态非门notif0、高电平使能三态非门notif1
- 必须是位置关联法例化

12.5 库元件和UDP用法介绍

调用门元件的格式是： 基本门元件名 <门例化名> (<端口关联列表>)

【例12-7】

```
module LOGICGATE (input A,B,C,S , output OUT);
wire a1,a2,a3,a4;
    not u1 (a1,B);
    and u2 (a2,A,a1);
    or  u3 (a3,C,B);
    xor u4 (a4,a3,a2);
  notif1 u5 (OUT,a4,S);
endmodule
```

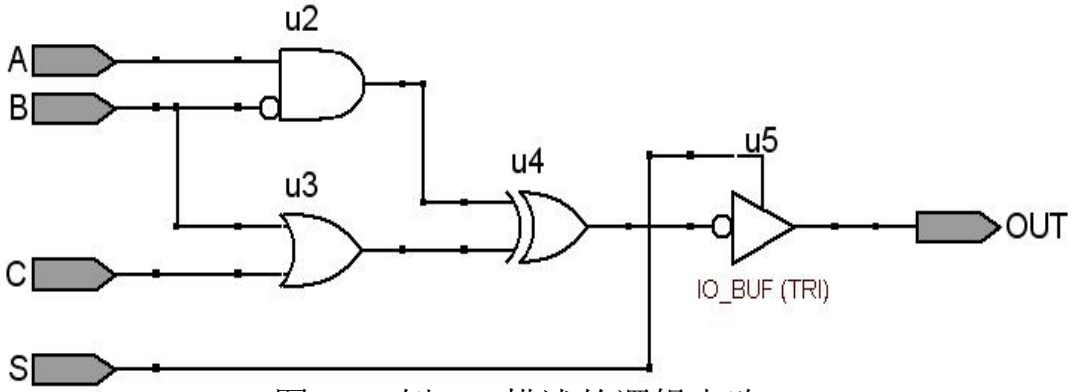


图12-5 例12-7描述的逻辑电路

12.5 库元件和UDP用法介绍

调用门元件的格式是： 基本门元件名 <门例化名> (<端口关联列表>)

其中普通门的端口关联列表按以下的顺序列出：
(输出， 输入1， 输入2， 输入3， ...);

例如3输入与门和2输入与门的例化语句如下：

```
and U1 (out,in1,in2,in3);          //三输入与门，例化名是U1
and U2 (out,in1,in2);              //二输入与门，例化名是U2
```

对于三态门，则按以下顺序列出输入/输出端口，例如：

```
bufif1 U1(out,in,enable);  //高电平使能的三态门
bufif2 U2(out,a,ctr1);             //低电平使能的三态门
```

至于buf和not两种元件的调用需注意，它们允许有多个输出，但只能有一个输入，如：

```
not IC1 (out1,out2,in);            //1输入in, 2输出out1,out2
buf IC2 (out1,out2, out3,in);      //1输入in, 3输出out1,out2, out3
```

12.5.2 用户自定义原语UDP及用法

- 用户自定义原语UDP即用户自定义的基础元件
- UDP由关键词primitive起始，引导出这个元件的结构和功能描述，以关键词endprimitive结束
- 关键词primitive旁的标识符是元件名，元件名右侧的括号是端口表，表中的标识符是端口名

12.5.2 用户自定义原语UDP及用法

- 调用UDP元件时，应严格按照源文件中端口定义的顺序来定义外部信号的连接
- Verilog规定UDP元件只能有一个输出端，且输出端口名要放在端口表括号的最左侧，而在右侧只能列出所有输入端口
- 关键词table_endtable引导该UDP的逻辑功能真值表

12.5 库元件和UDP用法介绍

【例12-8】

```
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
```

【例12-9】

```
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
```

例12-9描述的是一个半加器，利用位置关联法的例化语句调用了例12-8的异或门和库元件中的与门，从而实现了图3-1所示的半加器电路结构。

12.5.3 利用UDP元件设计多路选择器

【例12-10】

```
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12-11】

```
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
```

?代表“不关心”的意思，可以用问号“?”替代一些位值，以表示在逻辑关系中对这些位不在乎是什么值，以便简化逻辑表述。（?另一种含义：表示高阻值）。

12.5.4 用UDP表述D触发器

含异步复位RST控制的边沿触发型D触发器

【例12-12】

```
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12-13】

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

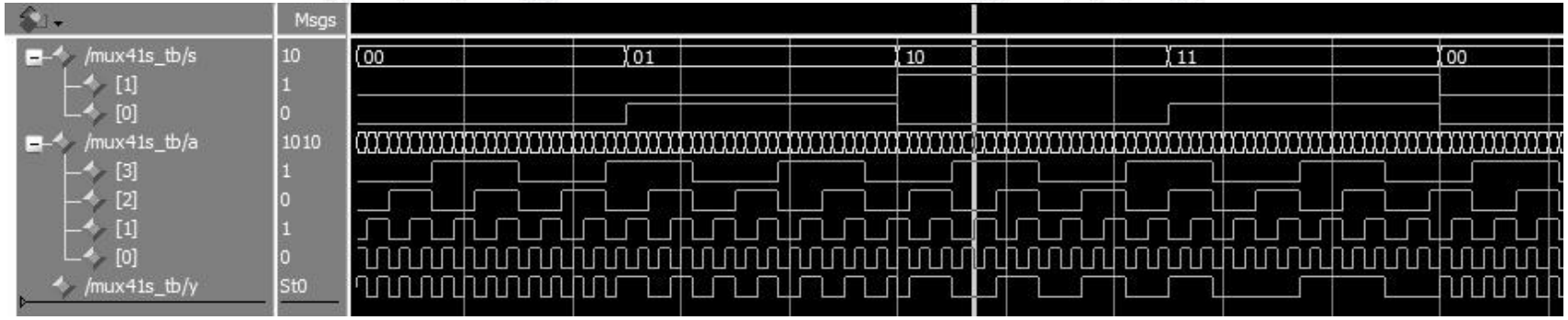
(01):上升沿触发；(10):下降沿触发；Q: 现态；Q+: 次态；-: 保持原状态；？：任意数据

12.6 常用模块的Verilog 简洁描述

12.6.1 多路选择器/复用器

【例12－14】

```
module mux41s
( input [1:0] s,
  input [3:0] a, output y);
assign y=a[s]; //注意s 是一个输入信号，是可变的
endmodule
```

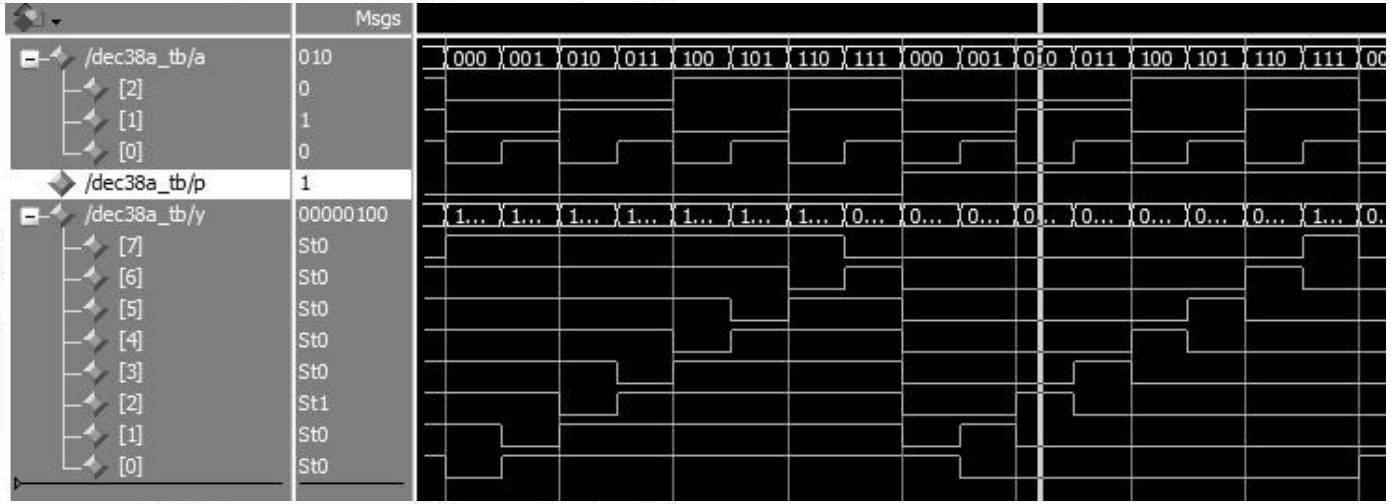


12.6 常用模块的Verilog 简洁描述

12.6.2 译码器

【例12-15】

```
module dec38a
( input [2:0] a,input p, // Polarity
  output [7:0] y);
  wire [7:0] temp_y;
  assign temp_y=1'b1<<a;
  assign y=(p)?temp_y:~temp_y;
endmodule
```




```
reg [7:0] temp_y;
always@(a) begin temp_y=0; temp_y[a]=1'b1; end
```

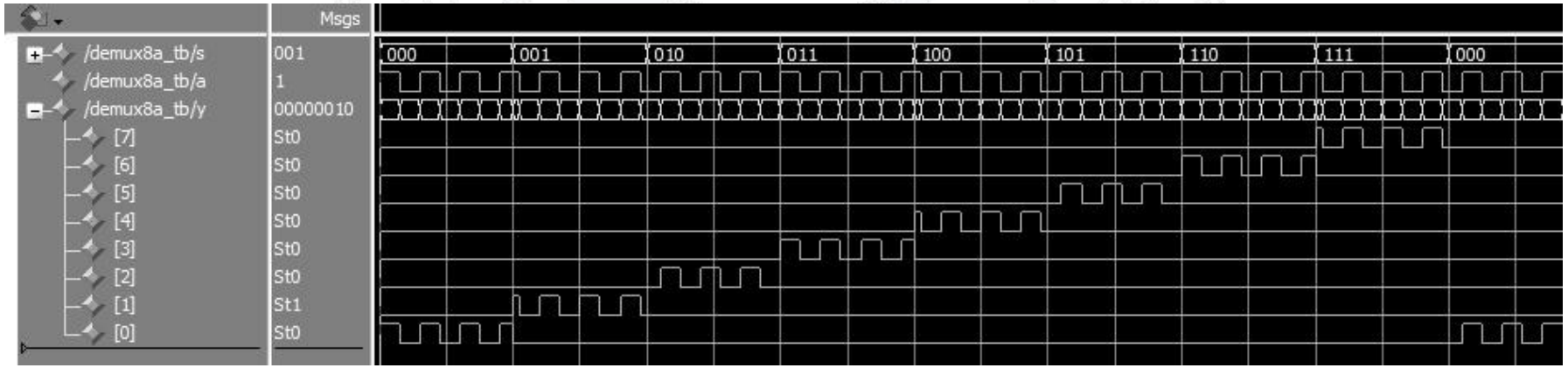
12.6 常用模块的Verilog 简洁描述

12.6.3 数据分配器/解复用器

【例12-16】

```
module demux8a
(input [2:0] s, input a, output [7:0] y);
assign y=a<<s;
endmodule
```

```
reg y;
always@* begin y=0; y[s]=a; end
```



12.6.4 PWM 发生器

【例12-17】

```
module cnt
#(parameter NUM=9_9999_9999)
(input clk,en,load,reset_n,input [31:0] d,output reg[31:0] q);
reg [31:0] qmax;
always@(posedge clk,negedge reset_n)
   if(!reset_n) qmax<=0; else if(load) qmax<=(d>NUM)?NUM:d;
always @(posedge clk,negedge reset_n)
   if(!reset_n) q<=0; else if(en) if(q<qmax ) q<=q+1;
   else q<=0;
endmodule
```

实验与设计

- 12-1 SPWM脉宽调制控制系统设计
- 12-2 数字彩色液晶显示控制电路设计
- 12-3 串行ADC/DAC控制电路设计
- 12-4 AM幅度调制信号发生器设计
- 12-5 VGA简单图像显示控制模块设计

EDA Lab

12-1 SPWM脉宽调制控制系统设计

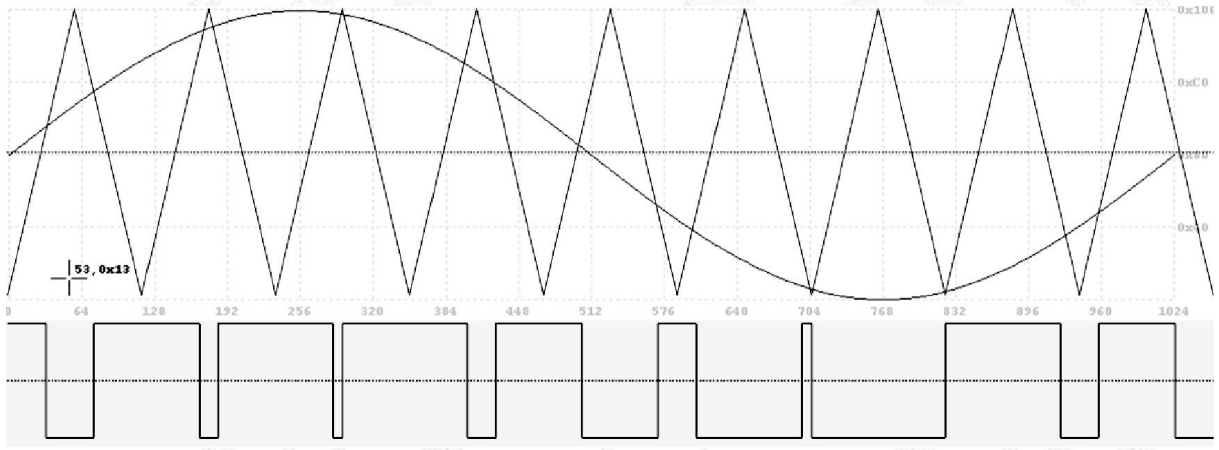


图12-11 SPWM波生成原理图

12-1 SPWM脉宽调制控制系统设计

【例12-18】

```
module TRANG (input[9:0] ADR，  output[9:0] OUTD);
  reg[9:0] OT1;   reg[10:0] CC;
  always @(ADR or CC)  begin
    if (ADR<10'H200)  begin  OT1[9:1]<=ADR[8:0]; OT1[0]<=1'b0;end
     else  begin  CC<=11'b10000000000 + (~ADR);
      OT1[9:1] <= CC[8:0];  OT1[0] <= 1'b0;     end   end
    assign OUTD = OT1;
endmodule
```

12-1 SPWM脉宽调制控制系统设计

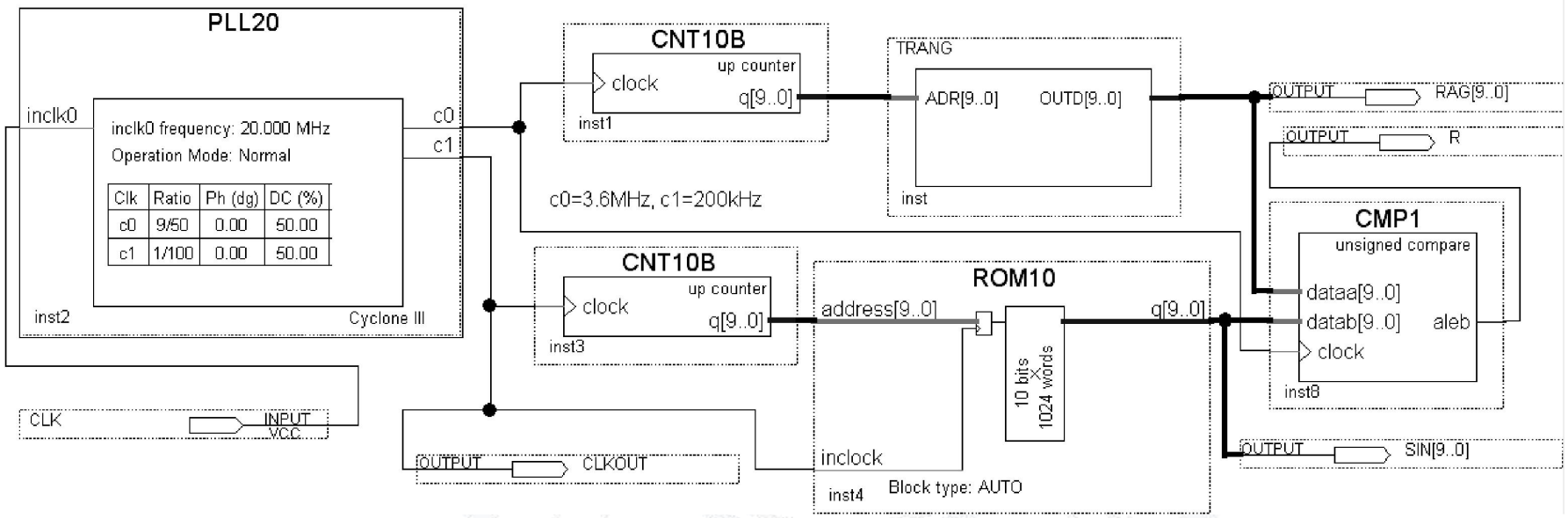


图12-12 SPWM波发生器基本电路图

12-1 SPWM脉宽调制控制系统设计

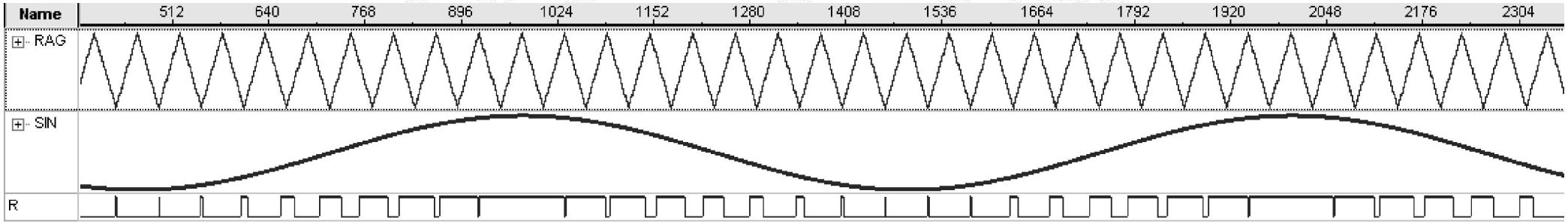


图12-13 图12-12电路的SignalTap II实测波形

12-1 SPWM脉宽调制控制系统设计

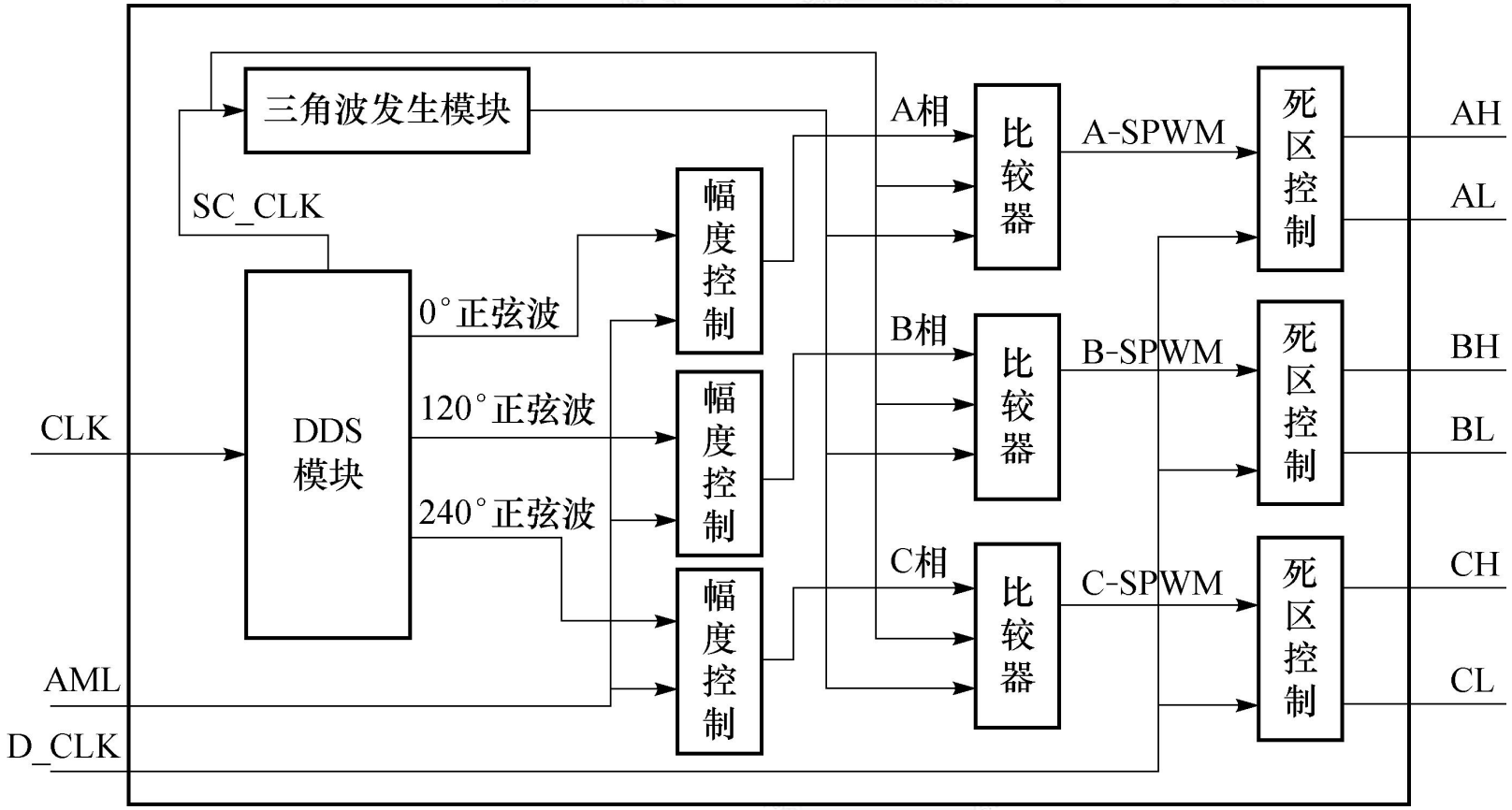


图12-14 三相SPWM控制器电路模块图

12-4 AM幅度调制信号发生器设计

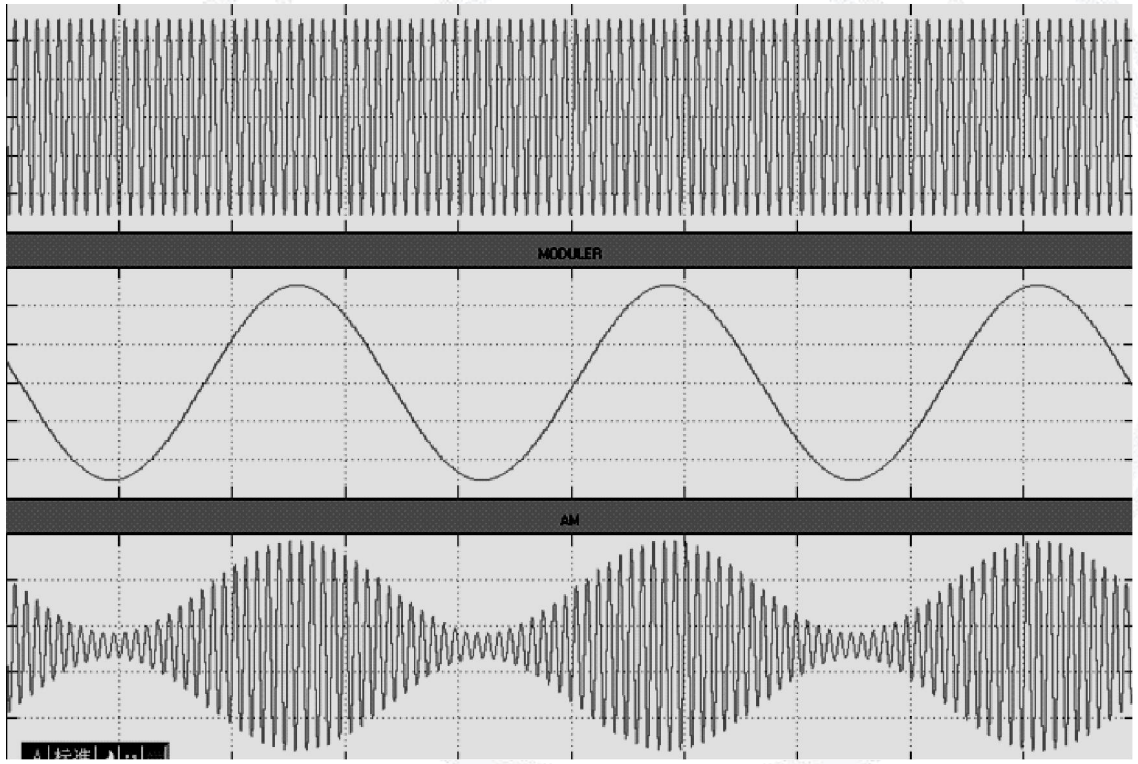


图12-15　AM模型仿真波形

12-5 VGA简单图像显示控制模块设计

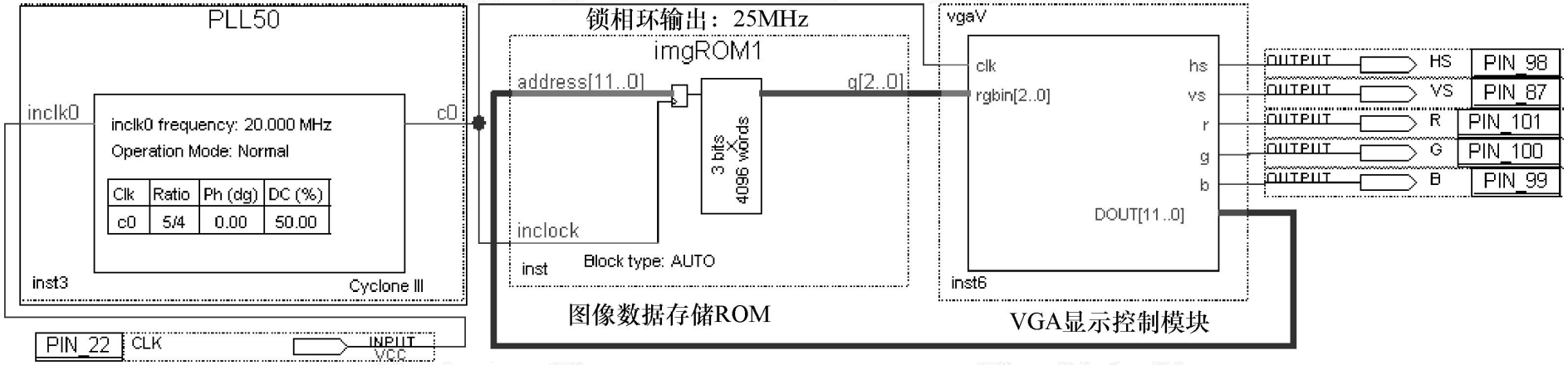


图12-16 VGA图像显示控制模块原理图

实验与设计

【例12-19】

```
module vgaV (clk, hs, vs, r, g, b, rgbin, DOUT);
  input clk;      //工作时钟25MHz
  output hs,vs;//场同步、行同步信号
  output r,g,b; // 红、绿、蓝信号
  input[2:0] rgbin;                //像素数据
  output[11:0] DOUT;               //图像数据ROM的地址信号
  reg[9:0] hcnt, vcnt;     reg r,g,b;     reg hs,vs;
  assign DOUT = {vcnt[5:0], hcnt[5:0]};
  always @(posedge clk)  begin  //水平扫描计数器
    if (hcnt<800)   hcnt<=hcnt+1;
    else            hcnt<={10{1'b0}};
    end
  always @(posedge clk)   begin  //垂直扫描计数器
    if (hcnt==640+8)    begin
      if (vcnt<525)  vcnt<=vcnt+1;
      else   vcnt<={10{1'b0}};  end   end
  always @(posedge clk)   begin //场同步信号发生
    if ((hcnt>=640+8+8) & (hcnt<640+8+8+96))
     hs<=1'b0; else  hs<=1'b1;    end
  always @(vcnt)   begin        //行同步信号发生
    if ((vcnt>=480+8+2) & (vcnt<480+8+2+2))
      vs<=1'b0;  else  vs<=1'b1;  end
  always @(posedge clk)   begin
    if (hcnt<640 & vcnt<480)   //扫描终止
    begin  r<=rgbin[2];  g<=rgbin[1];  b<=rgbin[0];   end
    else begin  r<=1'b0; g<=1'b0;  b<=1'b0;  end
  end
endmodule
```

后修课程

- 高年级本科生课程
  - FPGA应用于实践
- 研究生课程
  - FPGA算法优化与硬件加速

- 参考MOOC课程
  - 中国大学MOOC平台“FPGA算法优化与引进加速”课程