上讲回顾

- 有限状态机分类
- 编码的方法（顺序编码和独热码的区别）
- 有限状态及设计步骤

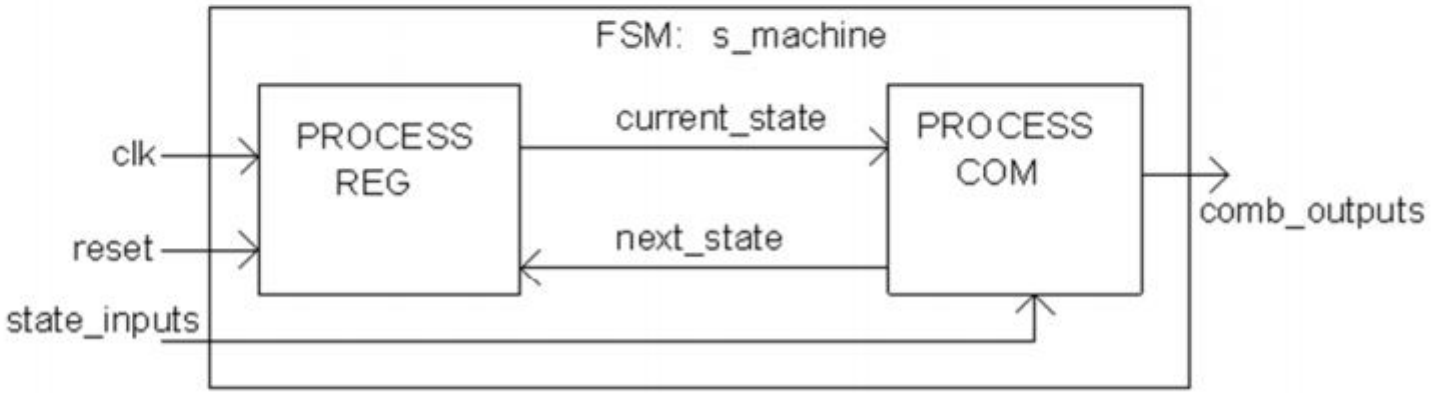
一般有限状态机的结构

1. 说明部分，如：

```
parameter st0 = 0，st1 = 1，st2 = 2，st3 = 3，st4 = 4;
reg [2:0] state，nxstate ;
```

2. 主控时序过程

3. 主控组合进程



4. 辅助进程

第三章
基于Verilog HDL语言的设计（六）

仿真过程一般包括以下的工作：

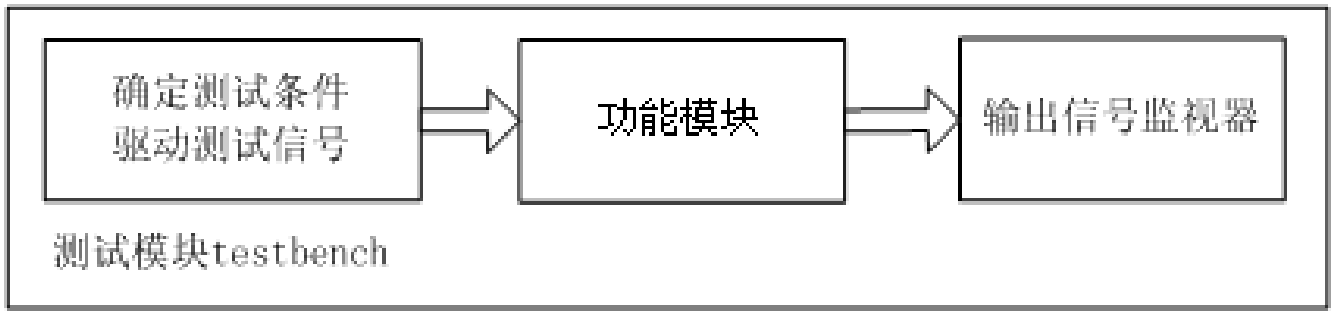
（1）产生仿真激励(波形)；

（2）将仿真的输入激励加入到被测试模块端口并观测其输出响应；

（3）将被测模块的输出与期望值进行比较，验证设计的正确性。

1．测试平台的搭建

（1）测试模块是顶层模块，它直接调用待测的功能模块，并生成测试激励信号添加

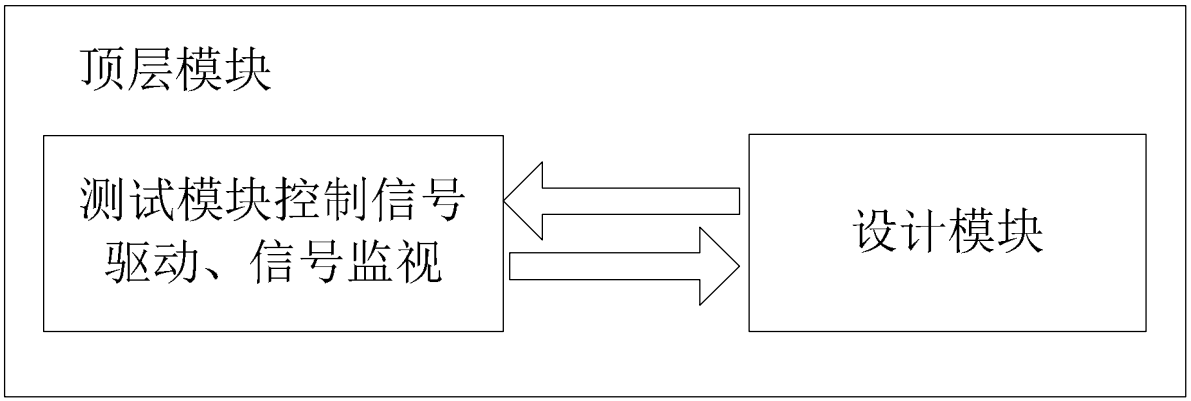


例：用图3.16的测试平台对例3-1的半加器进行测试。

```
`timescale 1ns/100ps
module testbench;
reg a，b；
wire co，s;
halfadder u1 ( .A(a),.B(b),  .CO(co), .S(s));

initial begin
        a = 0;  b = 0;
    #10 begin a=1;b=1;end
    #10 begin a=1;b=0;end
    #10 begin a=0;b=1;end
    #10 begin a=0;b=0;end
    #10 $stop；
    end
endmodule
```

（2）将测试模块和设计模块分别设计完成，然后在一个虚拟的顶层模块中进行调用，将相应端口进行连接。



例：用图3.17的测试平台对例3-1描述的半加器进行测试。
第一步：编写针对半加器的测试模块

```
`timescale 1ns/100ps
module testhalfadder(a,b,co,s);
input co,s;
output a,b;
  reg a，b;
   // 输出
    wire s,co;
   initial
begin
        a = 0;  b = 0;
       #10 begin a=1;b=1;end
       #10 begin a=1;b=0;end
       #10 begin a=0;b=1;end
       #10 begin a=0;b=0;end
       #10 $stop;
     end
endmodule
```

第二步：实例化半加器及测试模块，建立两个模块在同一个层次上的连接。

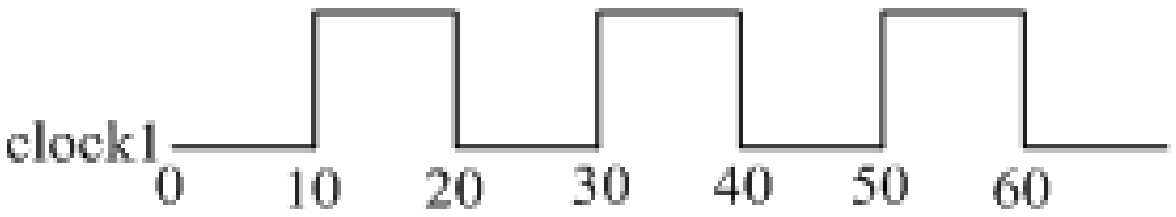
```
module testbench;
testhalfadder u1 (.co(co),.s(s),.a(a),.b(b));
halfadder u2 (.A(a),.B(b),.S(s),.CO(co));
endmodule
```

- 测试文件中时钟波形的设计是最基本的设计，常利用initial、always、forever、assign等语句产生时钟信号

（1）要产生周期性的时钟方波，

例：产生周期为20ns的时钟

```
`timescale 1ns/100ps
module Gen_clock1 (clock1) ;
output clock1;
reg clock1;
parameter T=20;
   initial    clock1 =0;
   always    # (T/2) clock1=~clock1;
endmodule
```



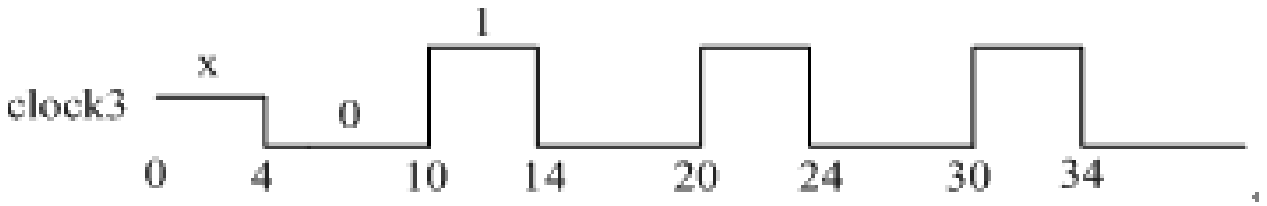
例：

```
`timescale 1ns/100ps
…
initial
begin clock2 = 0;
 forever #10 clock2=~clock2;  end
endmodule
```

（2）采用always语句产生高低电平持续时间不同的时钟:

例：

```
module Gen_clock3 (clock3);
output clock3;
reg clock3;
always
begin
# 4 clock3=0;  //时延4个单位时间后后，clock3赋值0
# 6 clock3=1;  //时延6个单位时间后后，clock3赋值1
end
endmodule
```



例：利用forever也能产生如图3.20的时钟波形

```
forever
begin
# 4 clock4=1;
# 6 clock4=0; end
endmodule
```

（3）产生具有相移的时钟。

例：module Gen_clock1 (clock_pshift,clock1) ;

output clock_pshift,clock1;

reg clock1;

wire clock_pshift;

parameter T=20;

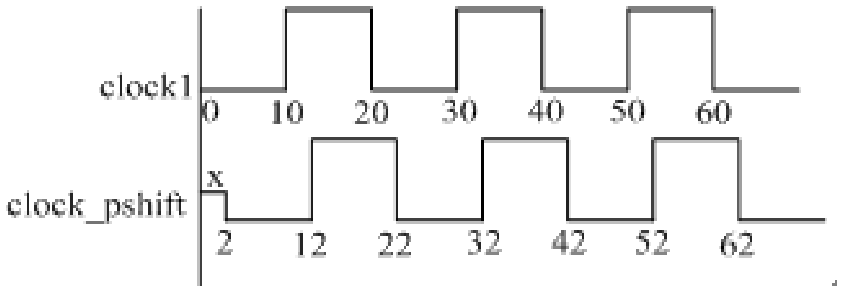
parameter pshift=2;

initial clock1 =0;

always # (T/2) clock1=~clock1;

assign #PSHIFT clock_pshift=clock1;

endmodule



（1）输入信号取值数据量较少时可以使用initial语句对输入信号的变化进行穷举式的描述。

```
`timescale 1ns/100ps
module testbench();
//定义输入输出端口
reg 输入激励端口罗列；
wire 输出端口罗列；
//例化被测功能模块；
//输入端加入激励信号
initial
#延迟时间  begin
        输入激励信号赋值；end
#延迟时间  begin
        输入激励信号赋值；end
…
end
endmodule
```

（2）激励数据量较多时，可以通过编写、调用任务和函数完成重复性操作，减少程序书写工作量。

例

```
`timescale 1ns / 100ps
…
task writeburst;
   input [7:0] wdata;
   begin
      @(posedge clockin) begin
         write_enable = #2 1;
         write_data = #2 wdata;     end
   end
endtask
//可以调用低层任务，构筑更复杂的任务操作
task writeburst4;
begin
      writeburst(128); writeburst(129); writeburst(130); writeburst(131);
       …
end
```

```
//调用任务，进行较多测试数据的输入
initial
begin
    writeburst4;
 …
end
endmodule
```

3、使用系统函数读入数据，完成测试激励的输入

例：

```
`timescale  1ns / 100ps
module testbench;
…
//将数据文件中的值读入某寄存器中.
initial
begin
      $readmemb(“d:\test\mydata .dat", my_memory);
…
end
//使用数组中的数据作为输入激励
always @ (posedge clk)
begin
    data_in <= my_memory[i]
    i=<i+1;
…
end
initial
begin
```

```
//打开一个文件，准备接收仿真的输出数据
   file_descriptor= $fopen(“d:\simulus.dat");
…
$fwrite(file_descriptor ，"%b\n",result);    //将仿真数据写入输出文件
…
$fclose(file_descriptor);
 end
```

3.7 VerilogHDL设计实例

- 本节课通过示例讲解采用Verilog HDL为设计输入方式，在ISE环境下，进行数字系统的FPGA开发的方法。
- 通过本节课内容，复习ISE软件的使用，掌握设计输入方式进行数字系统的FPGA开发。

- 一般而言，在对FPGA芯片进行开发时，要注意以下几点：

1. 理解设计目标，用“自顶向下”的思路将设计目标分解成多个实现难度合适的子模块，确定各个子模块的输入/输出端口，实现的功能等。
2. 分别对各个子模块进行设计、验证。
3. 将低层模块组合、连接构成上一层系统，最后组合得到目标系统，应注意，在每一级的连接完成后，都应该进行功能仿真、时序仿真，及早发现设计中的问题

- 例3-65：系统输入一个5MHz的时钟源，设计分频电路产生一个2MHz的时钟信号 。
- 1．任务分析：
- 由于分频系数是2.5，考虑设计一个模3的计数器，再设计一个扣除脉冲电路，加在模3计数器后，每来两个脉冲就扣除一个脉冲，就可以得到分频系统为2.5的小数分频器。这个数字电路系统可以由异或门、模3计数器、2分频器3个模块组成。

- 系统框图如下图所示：

- 2．设计流程：
- 1）创建工程
- 启动ISE工程管理器，新建一个工程。

- 2） 各个子模块设计：

（1）异或门模块，直接调用Verilog HDL提供的逻辑原语xor。

（2）模3计数器:

源文件名为“mod3_counter.v”

（3）设计2分频器:

源文件名为“devide.v”

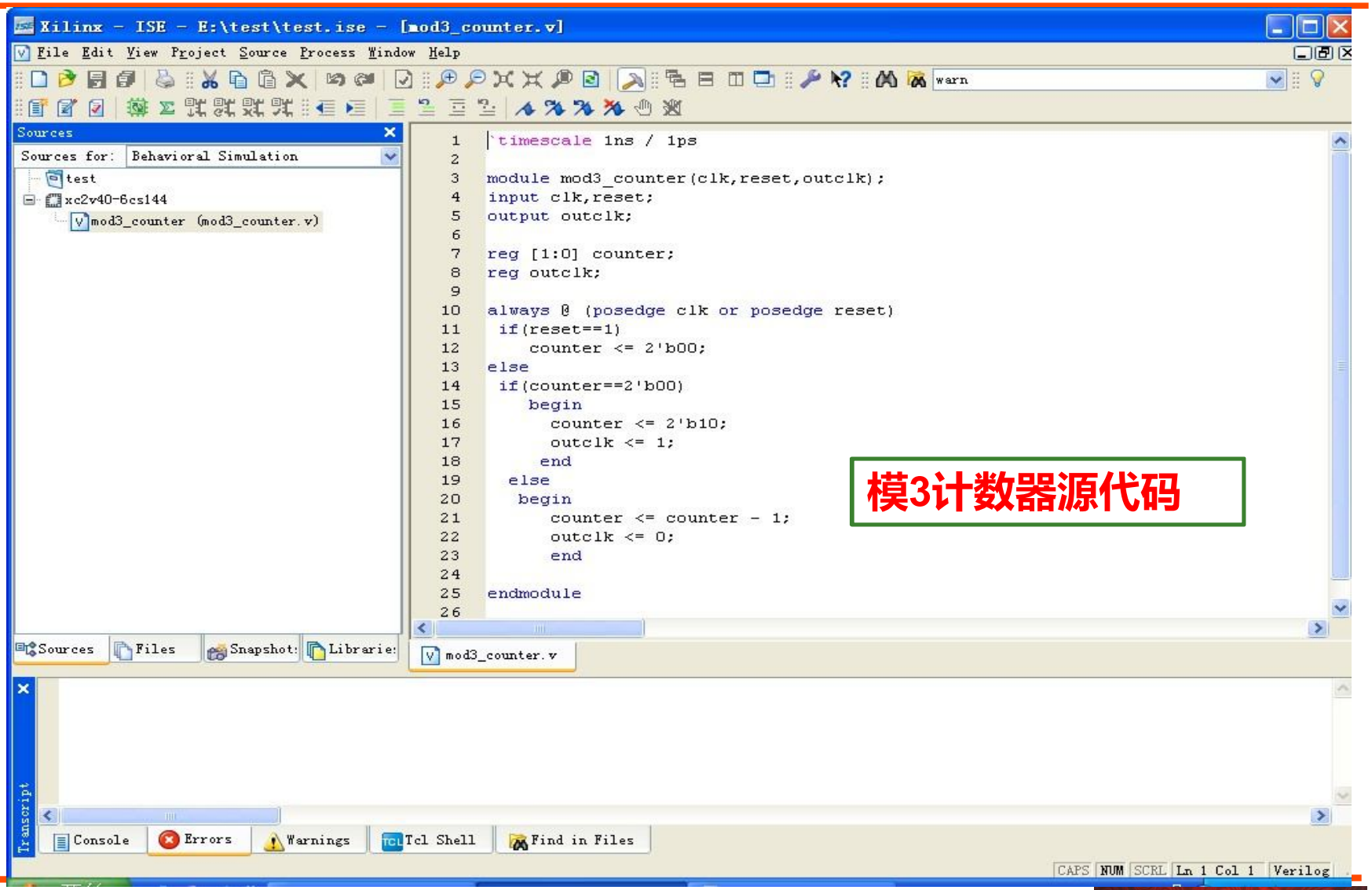
- （2）模3计数器设计步骤：

①创建和编写功能模块。

② 创建和编写测试文件。

③ 用测试文件对被测试模块进行功能仿真。

④ 模块的时序仿真。


Xilinx - ISE - E:\test\test.ise - [mod3_counter.v]
File Edit View Project Source Process Window Help
warn
Sources
Sources for: Behavioral Simulation
test
xc2v40-6cs144
mod3_counter (mod3_counter.v)
Sources
Files
Snapshot
Librarie
1 `timescale 1ns / 1ps
2
3 module mod3_counter(clk,reset,outclk);
4 input clk,reset;
5 output outclk;
6
7 reg [1:0] counter;
8 reg outclk;
9
10 always @ (posedge clk or posedge reset)
11 if(reset==1)
12 counter <= 2'b00;
13 else
14 if(counter==2'b00)
15 begin
16 counter <= 2'b10;
17 outclk <= 1;
18 end
19 else
20 begin
21 counter <= counter - 1;
22 outclk <= 0;
23 end
24
25 endmodule
26
mod3_counter.v
模3计数器源代码
Transcript
Console
Errors
Warnings
Tcl Shell
Find in Files
CAPS NUM SCRL Ln 1 Col 1 Verilog

```
module test_mod3_counter;
    reg clk;
    reg reset;
    wire outclk;
    mod3_counter uut (
        .clk(clk),
        .reset(reset),
        .outclk(outclk)        );
```

模3计数器顶层测试文件

```
initial begin // Initialize Inputs
        clk = 0;
        reset = 1;
        #100;
        #10 clk=1;
                    #10 clk=0;
                      reset=0;
                    #10 clk=1;
                    #10 clk=0;
                    #10 clk=1;
                    #10 clk=0;
                    #10 clk=1;
                    #10 clk=0;
                    #10 clk=1;
                   #10 clk=0;
                    end
                   endmodule
```

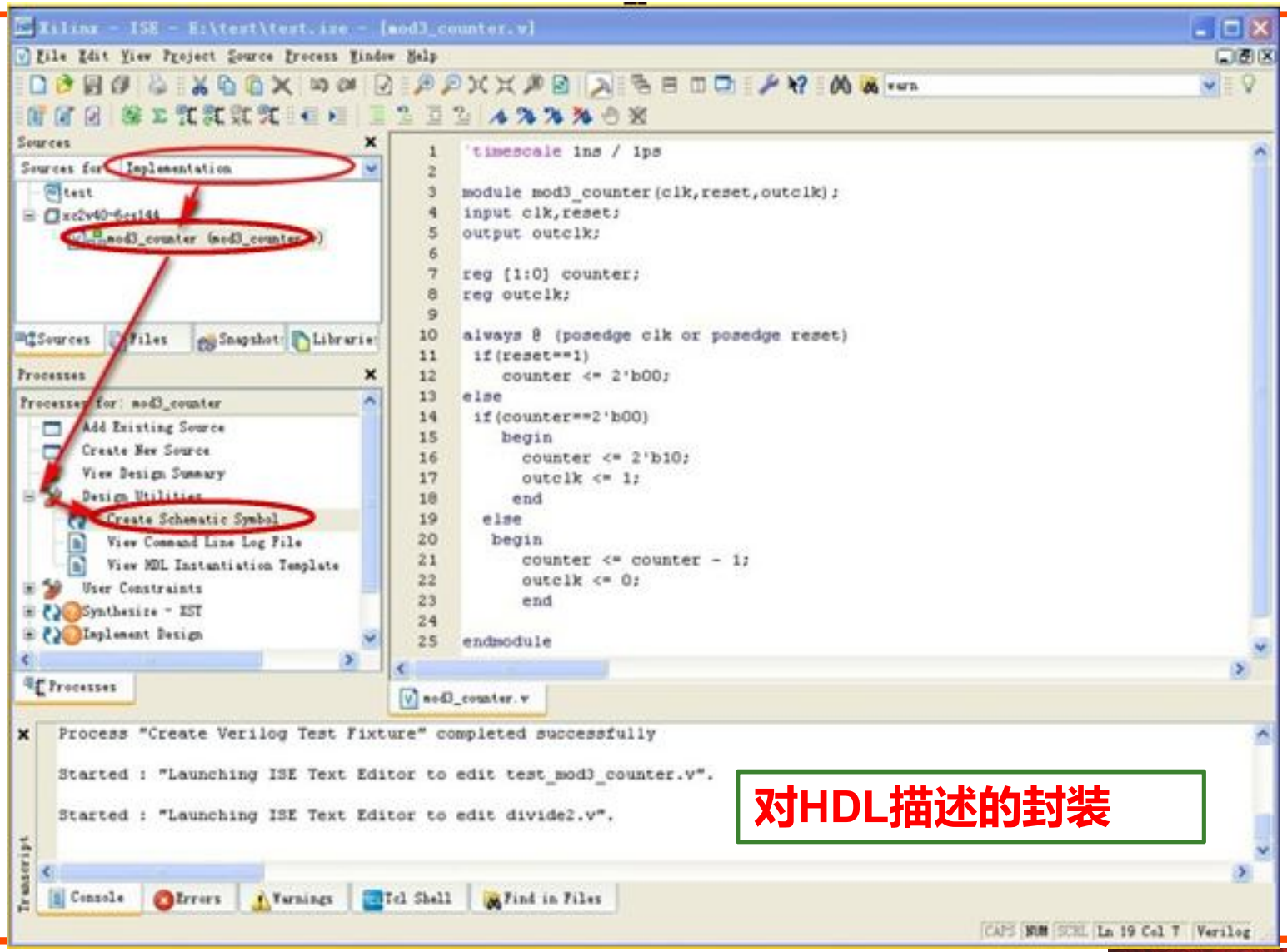
- （3）设计2分频器：按照模3计数器的设计步骤进行设计，仿真验证结果。

- 3）顶层模块设计
- 顶层模块设计可采用原理图描述方法或HDL语言描述方法。
- (1)原理图描述方法需要将用HDL描述的各个子模块封装成元件，然后在顶层模块设计调用。
- （2）HDL语言描述方法是将各个子模块用HDL语言描述连接起来，方法和前面子模块设计类似。

原理图描述方法设计步骤如下：

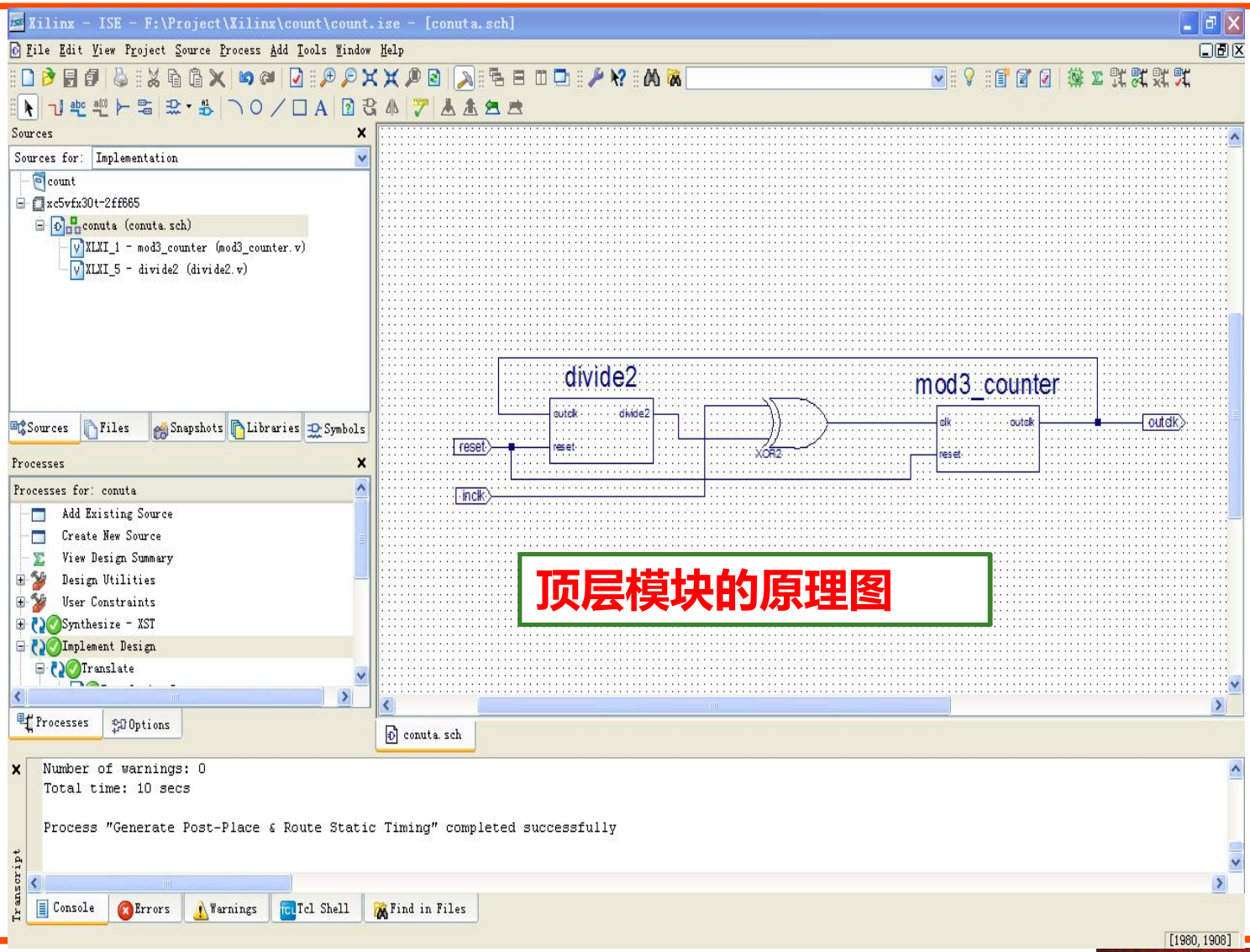
① ①在ISE工程管理器的【Sources】窗口中【Sources for】栏选择【Implementation】选项，并选中要封装的功能块对应的源程序，如“mod3_counter.v”。

② 在工程管理器的【Processes】窗口相应就出现了综合、适配等设计环节的标识。展开【Design Utilities】栏，双击【Create Schematic Symbol】选项，对【Sources】窗口中选择的HDL模块进行封装。整个封装的过程如下图：


Xilinx - ISE - E:\test\test.ise - [mod3_counter.v]
File Edit View Project Source Process Window Help
Sources
Sources for: Implementation
test
xc2v40-6cs144
mod3_counter (mod3_counter.v)
Sources
Files
Snapshot
Libraries
Processes
Processes for: mod3_counter
Add Existing Source
Create New Source
View Design Summary
Design Utilities
Create Schematic Symbol
View Command Line Log File
View HDL Instantiation Template
User Constraints
Synthesize - XST
Implement Design
Processes
1 'timescale 1ns / 1ps
2
3 module mod3_counter(clk,reset,outclk);
4 input clk,reset;
5 output outclk;
6
7 reg [1:0] counter;
8 reg outclk;
9
10 always @ (posedge clk or posedge reset)
11 if(reset==1)
12 counter <= 2'b00;
13 else
14 if(counter==2'b00)
15 begin
16 counter <= 2'b10;
17 outclk <= 1;
18 end
19 else
20 begin
21 counter <= counter - 1;
22 outclk <= 0;
23 end
24
25 endmodule
mod3_counter.v
Process "Create Verilog Test Fixture" completed successfully
Started : "Launching ISE Text Editor to edit test_mod3_counter.v".
Started : "Launching ISE Text Editor to edit divide2.v".
Transcript
Console
Errors
Warnings
Tcl Shell
Find in Files
Ln 19 Col 7
Verilog

对HDL描述的封装

- 封装得到的元件将自动保存在当前工程的存储路径下，当使用该模块符号进行顶层模块的原理图设计时，在时可以像使用系统库中的模块一样被调用。
- 依次将以上3个子模块都封装成元件，使用ECS绘制出顶层模块的原理图，方法详见2.2.2节。顶层模块的原理图如下图所示。

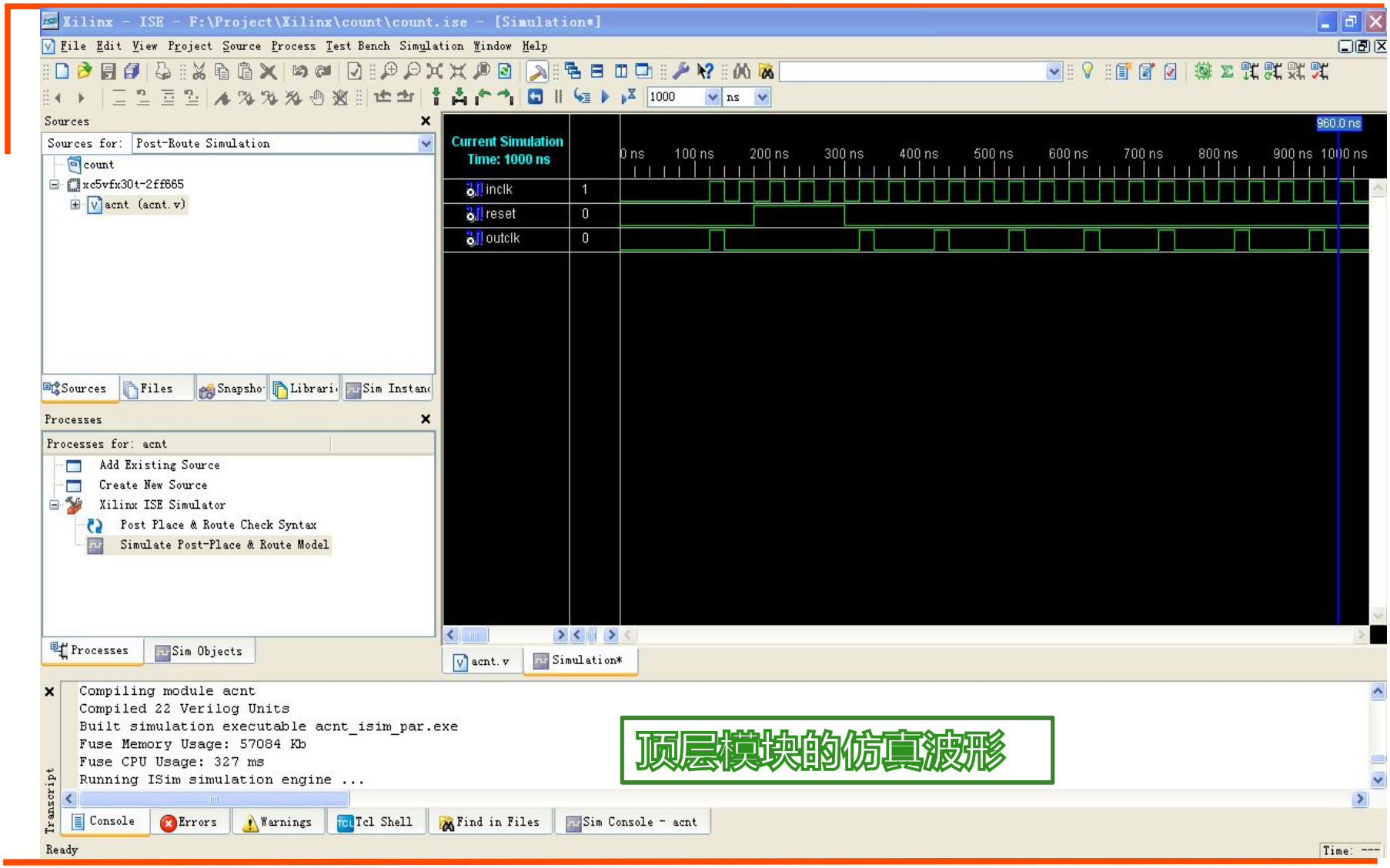

Xilinx - ISE - F:\Project\Xilinx\count\count.ise - [conuta.sch]
File Edit View Project Source Process Add Tools Window Help
Sources
Sources for: Implementation
count
xc5vfx30t-2ff665
conuta (conuta.sch)
XLXI_1 - mod3_counter (mod3_counter.v)
XLXI_5 - divide2 (divide2.v)
Sources Files Snapshots Libraries Symbols
Processes
Processes for: conuta
Add Existing Source
Create New Source
View Design Summary
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Translate
Processes Options
divide2
outclk
divide2
reset
mod3_counter
clk
outclk
reset
XOR2
reset
inclk
outclk
顶层模块的原理图
conuta.sch
Number of warnings: 0
Total time: 10 secs
Process "Generate Post-Place & Route Static Timing" completed successfully
Transcript
Console Errors Warnings Tcl Shell Find in Files
[1980, 1908]

（2）HDL语言描述方法是将各个子模块用HDL语言描述连接起来，方法和前面子模块设计类似。打开一个的文本编辑界面，编写顶层模块的参考程序如下

顶层模块：

```
module devide(inclk,reset,outclk);
//端口声明
input inclk;
input reset;
output outclk;
wire  clk;
//子模块例化
xor (clk,inclk,divide2);
mod3_counter u1(clk,reset,outclk);
divide2 u0(outclk,reset,divide2);
endmodule
```

- 在完成顶层模块的连接后，按照前面各个子模块设计步骤，进行功能仿真、锁定引脚、综合、布局布线、时序仿真、下载等。锁定引脚和下载的方法详见2.2.4节和2.2.6节。本例中主要介绍Verilog HDL在ISE软件中设计步骤，一些模块的编程请读者自行完成设计，顶层模块的时序仿真结果可以参考下图。


Xilinx - ISE - F:\Project\Xilinx\count\count.ise - [Simulation*]
File Edit View Project Source Process Test Bench Simulation Window Help
Sources
Sources for: Post-Route Simulation
count
xc5vfx30t-2ff665
acnt (acnt.v)
Sources Files Snapsho Librari Sim Instanc
Processes
Processes for: acnt
Add Existing Source
Create New Source
Xilinx ISE Simulator
Post Place & Route Check Syntax
Simulate Post-Place & Route Model
Processes Sim Objects
Current Simulation Time: 1000 ns
0 ns 100 ns 200 ns 300 ns 400 ns 500 ns 600 ns 700 ns 800 ns 900 ns 1000 ns
960.0 ns
inclk 1
reset 0
outclk 0
acnt.v Simulation*
Compiling module acnt
Compiled 22 Verilog Units
Built simulation executable acnt_isim_par.exe
Fuse Memory Usage: 57084 Kb
Fuse CPU Usage: 327 ms
Running ISim simulation engine ...
Transcript
Console Errors Warnings Tcl Shell Find in Files Sim Console - acnt
Ready
Time: ---

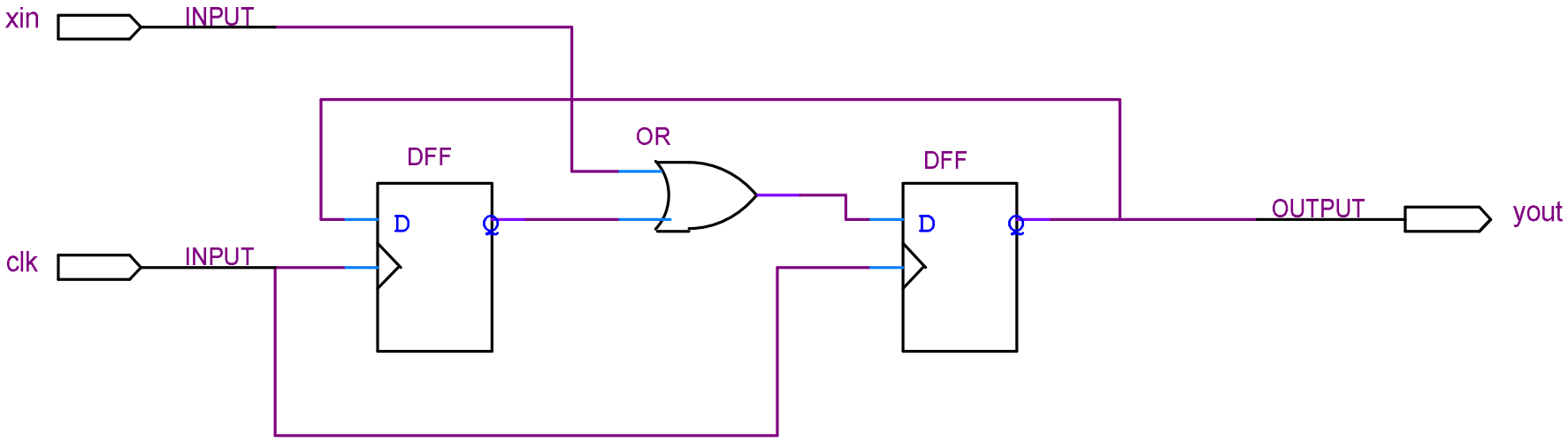
顶层模块的仿真波形

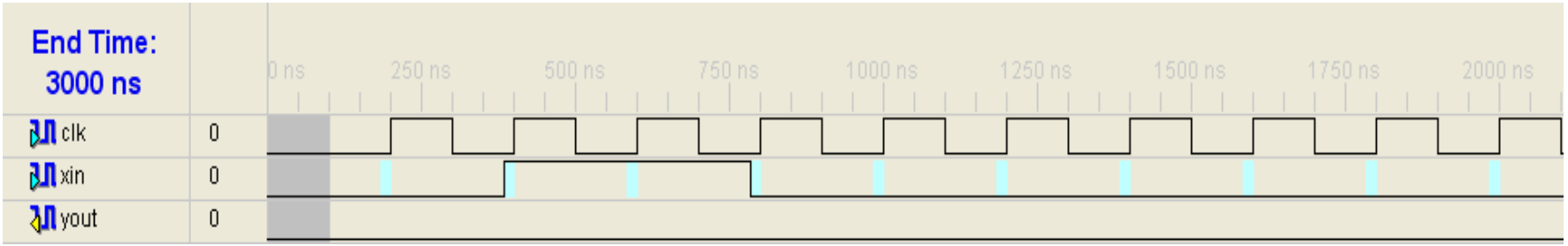
小结：

1 明确设计目标后才开始编程；
2 用硬件电路系统的思想来编写HDL；
3 理解HDL的可综合性；
4.语法掌握贵在精，不在多；

作业

- 1、根据图所示的原理图写出相应的Verilog HDL程序，其中DFF模块是上升沿触发的触发器。然后根据给出的输入波形写出测试代码，并画出输出仿真波形。





作业

2、用Verilog HDL设计一个五分频电路，

1）输入信号：

clk-------- 输入时钟

reset--------同步复位信号，当reset=1’b1时，系统输出置零，当reset=1’b0时，系统正常工作。

2）输出信号：

clk_out --------输出信号，其频率是输入时钟的五分之一。

作业

3、用Verilog HDL设计能实现自动测频十进制数字频率计具体功能如下：

1）测量范围: 1Hz～9999Hz

2）测量的数值通过4个数码管显示

3）频率超过9999Hz时，溢出指示灯亮，可以作为扩大测量范围的接口。

编写测试代码进行仿真验证。

1、测频原理

若某一信号在T秒时间里重复变化了N 次，则根据频率的定义可知该信号的频率fs 为：fs=N/T 通常测量时间T取1秒或它的十进制时间。

- 当T=1s，N就是测得的频率。

测定信号的频率必须有一个脉宽为1秒的对输入信号脉冲计数允许的信号；1秒计数结束后，计数值锁入锁存器的锁存信号和为下一测频计数周期作准备的计数器清0信号。这清0个信号可以由一个测频控制信号发生器产生

分析

- Cnt10d.v 10进制计数器，用于计算分频结果，并连接数码管显示。
- TESTCTL .v【例1】--测频控制器，使得频率计能自动测频，输出信号分别是：
  - CNT_EN，连接计数器，控制计数器的使能端；
  - RST_CNT,连接计数器，控制计数器的清零端
  - LOAD，连接锁存器，锁存器的锁存信号
- reg4 .v【例2】锁存器，将频率计的每位数锁存后输出。

分析

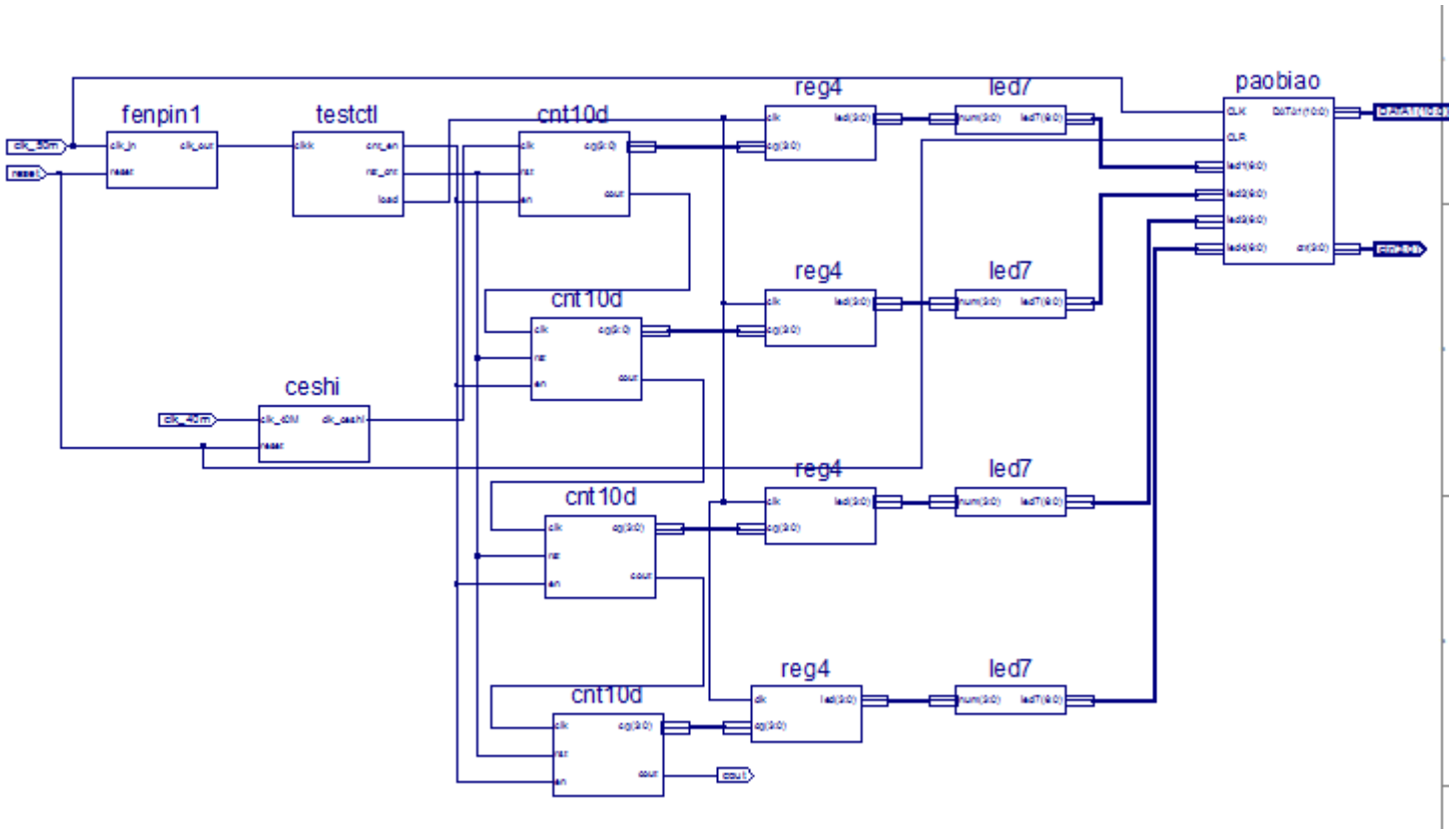
TESTCTL的计数使能信号CNT_EN能产生一个1秒脉宽的周期信号，并对频率计的每一计数器CNT10的EN使能端进行同步控制。当CNT_EN高电平时，允许计数；低电平时停止计数，并保持其所计的脉冲数。在停止计数期间，首先需要一个锁存信号LOAD的上跳沿将计数器在前1秒钟的计数值锁存进各锁存器REG4B中，并由外部的7段译码器译出，显示计数值。设置锁存器的好处是，显示的数据稳定，不会由于周期性的清零信号而不断闪烁。锁存信号之后，必须有一清零信号RST_CNT对计数器进行清零，为下1秒钟的计数操作作准备。

输入端口：

- F_in为待测频率输入端口
- CLK为1Hz的基准时钟信号，经过处理后可以产生用于自动测频所需的计数允许、锁存数据和清零三个控制信号。
- 测试时：这两个信号可以由一个分频模块同时产生，也可单独编写。

输出端口：

- 选择发光二极管分别对应个位、十位、百位、千位的数码管显示。自行编写数码显示模块。
- CARRY_OUT为溢出指示端口，也为扩大测量范围的留接口。


fenpin1
testctl
cnt10d
reg4
led7
paobiao
ceshi
cnt10d
reg4
led7
cnt10d
reg4
led7
cnt10d
reg4
led7