2 Verilog HDL入门与功能仿真

2.1 硬件描述语言简介

2.2 Verilog HDL程序的基本结构

2.3 逻辑功能的仿真验证过程

2.4 ModelSim仿真软件的使用

2.5 Verilog HDL功能仿真常用命令

2.1 硬件描述语言简介

- 硬件描述语言HDL(Hardware Description Languag) 类似于高级程序设计语言. 它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，复杂数字逻辑系统的逻辑功能。用HDL编写设计说明文档易于存储和修改，并能被计算机识别和处理.
- HDL是高层次自动化设计的起点和基础.目前， IEEE推出两种标准：VHDL和Verilog HDL

(1) VHDL的起源与发展

- ***V***ery high speed integrated ***H***ardware ***D***escription ***L***anguage (**VHDL**)
- **它是70年代末和80年代初，起源于美国国防部提出的超高速集成电路VHSIC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 。 1981年6月成立了VHDL小组。**
- **1983年第三季度，由IBM公司、TI公司、Intermetric 公司成立开发小组。**
- **1986年3月，IEEE开始致力于VHDL的标准化工作，讨论VHDL语言标准。**
- **IEEE于1987年12月公布了VHDL的标准版本（IEEE STD 1076/1987）；**
- **1993年VHDL修订，形成新的标准即IEEE STD 1076-1993)。**

(2) Verilog HDL的起源与发展

- 1981年Gateway Automation硬件描述语言公司成立；
- 1983～84年间该公司发布“Verilog HDL”及其仿真器Verilog -XL；
- 1986年Phil Moorby提出快速门级仿真的XL算法并获得成功，Verilog语言迅速得到推广。 Verilog-XL较快，特别在门级，能处理万门以上的设计。
- 1987年Synonsys公司开始使用Verilog行为语言作为它综合工具的输入；
- 1989年12月 Cadence公司并购了Gateway公司；
- 1990年初Cadence公司把Verilog HDL和Verilog-XL分成单独产品，公开发布了Verilog HDL,与VHDL竞争。并成立Open Verilog International(OVI)组织，负责Verilog的发展和标准的制定。

(2) Verilog HDL的起源与发展

- 1993年 几乎所有ASIC厂商支持Verilog HDL,认为Verilog-XL是最好的仿真器。OVI推出2.0版本的Verilog HDL规范，IEEE接受了将OVI的Verilog2.0作为IEEE标准的提案。
- 1995年12月，定出Verilog HDL的标准IEEE 1364。
- 2001年3月IEEE正式批准了Verilog-2001标准（即IEEE 1364-2001）。
- Verilog-2001标准在Verilog-1995的基础上有几个重要的改进。新标准有力地支持可配置的IP建模，大大提高了深亚微米（DSM）设计的精确性，并对设计管理作了重大改进。

（3）两种语言的比较（能力、数据类型、易学性、效率）

- 能力（capability）

VHDL

结构建模

抽象能力强

系统级－算法级－RTL级－逻辑级－门级

Verilog

结构建模

具体物理建模能力强

算法级－RTL级－逻辑级－门级－版图级

• 数据类型（data type）

VHDL

是一种数据类型性极强的语言。支持用户定义的数据类型。当对象的数据类型不一样时必须用类型转换函数转换。可以使用抽象（比如枚举）类型为系统建模。能利用数据类型检查编程的错误。

Verilog

数据类型简单。只能由语言本身定义，不能由用户定义。适于硬件结构的建模，不适于抽象的硬件行为建模。

• 易学性（easiest to learn）

VHDL

是一种数据类型很强的语言，欠直观。加之同一种电路有多种建模方法，通常需要一定的时间和经验，才能高效的完成设计。

VHDL根植于ADA，有时简洁，有时冗繁，如行为描述简洁，结构描述冗繁。

Verilog

由于Verilog为直接仿真语言，数据类型较简单，语法很直观，故Verilog更易理解和好学。

Verilog更像C，约有50％的结构来自C，其余部分来自ADA。

• 效 率

VHDL 由于数据类型严格，模型必须精确定义和匹配数据类型，这造成了比同等地verilog效率要低。

Verilog 不同位宽的信号可以彼此赋值，较小位数的信号可以从大位数信号中自动截取自己的位号。在综合过程中可以删掉不用的位，这些特点使之简洁，效率较高。

（4）VHDL语言的新进展

近年来，VHDL又有了一些新的发展。例如，为了大幅度提高EDA 工具的设计能力，出现了一系列对HDL语言的扩展。OO-VHDL(Object-Oriteted VHDL，即面向对象的VHDL）模型的代码比VHDL模型短30%～50%，缩短了开发时间，提高了设计效率。

美国杜克大学扩展的DE-VHDL(Duke Extended VHDL)通过增加3条语句，使设计者可以在VHDL描述中调用不可综合的子系统（包括连接该子系统和激活相应功能）。杜克大学用DE-VHDL进行一些多芯片系统的设计，极大地提高了设计能力。

（5）Verilog HDL语言的新进展

OVI组织1999年公布了可用于模拟和混合信号系统设计的硬件描述语言Verilog-AMS语言参考手册的草案，Verilog-AMS语言是符合IEEE 1364标准的Verilog HDL子集。目前Verilog-AMS还在不断的发展和完善中。

结 论

HDL主要用于数字电路与系统的建模、仿真和自动设计。目前有两种标准的硬件描述语言：Verilog和VHDL。由于Verilog简单易学，所以我建议大家学习Verilog HDL语言。

我国国家技术监督局于1998年正式将《集成电路/硬件描述语言Verilog》列入国家标准，国家标准编号为GB/T18349-2001，从2001年10月1日起实施。相信该标准的制定对我国集成电路设计技术的发展有重要的推动作用。

2.2 Verilog HDL程序的基本结构

2.2.1 简单Verilog HDL程序实例

2.2.2 Verilog HDL程序的基本结构

module (···);

2.2.1 简单Verilog HDL程序实例

end module

Verilog使用大约100个预定义的关键词定义该语言的结构

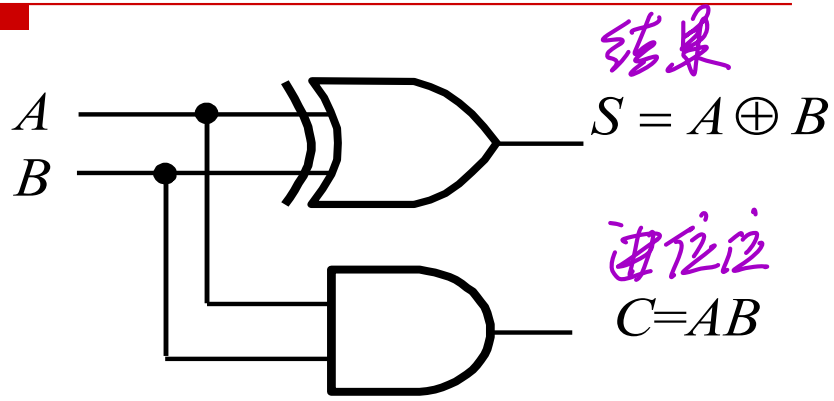
1. VerilogHDL程序由模块构成。每个模块的内容都是嵌在关键词module和endmodule两个语句之间。每个模块实现特定的功能。

2. 每个模块先要进行端口的定义，并说明输入（input）、输出（output)和双向（inout)，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VerilogHDL程序的任何部分做注释。

2.2.1 简单Verilog HDL程序实例



半加器逻辑图

模块名

```
/* Gate-level description of a half adder */
module HalfAdder_GL(A, B, Sum, Carry);
   input  A ,B ;                    //输入端口声明
   output  Sum, Carry ;             //输出端口声明
   wire A ,B , Sum ,Carry ;
   xor X1 (Sum, A, B );
   and A1 (Carry, A, B);
endmodule
```

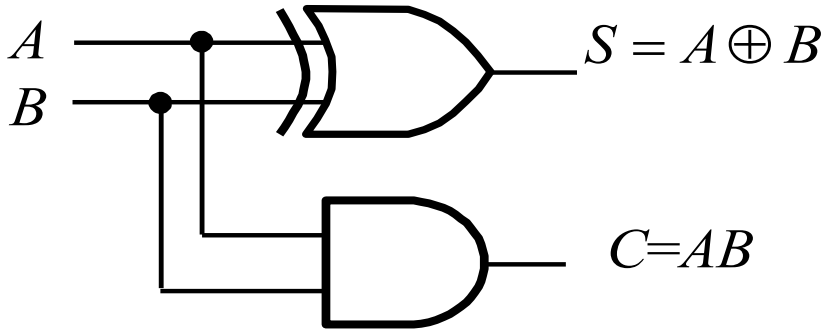
端口类型说明

数据类型说明

功能描述

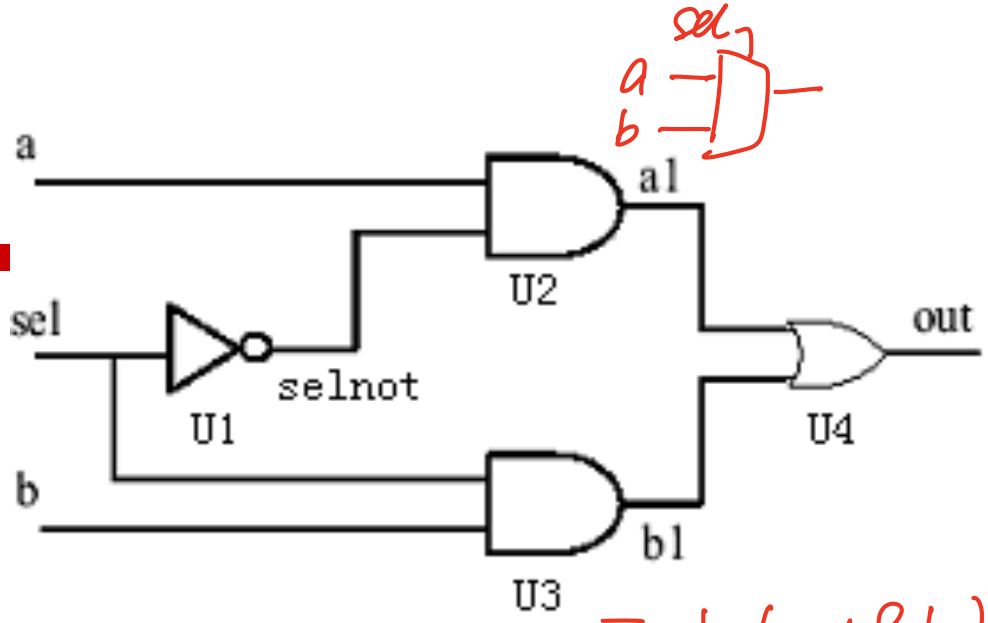
2.2.1 简单Verilog HDL程序实例

```
/* Dataflow description of a half adder */
module HalfAdder_DF(A, B, Sum, Carry);
   input  A ,B ;
   output  Sum ,Carry ;
   wire A ,B, Sum ,Carry ;
   assign  Sum = A ^ B;
   assign  Carry = A & B;
endmodule
```




```
/* Behavioral description of a half adder */
module HalfAdder_BH(A, B, Sum, Carry);
   input  A ,B ;
   output  Sum ,Carry ;
   reg Sum ,Carry ;      //声明端口数据类型为寄存器
   always @(A or B)  begin
      Sum = A ^ B;  //用过程赋值语句描述逻辑功能
      Carry = A & B;
   end
endmodule
```

例 2选1数据选择器的程序实例



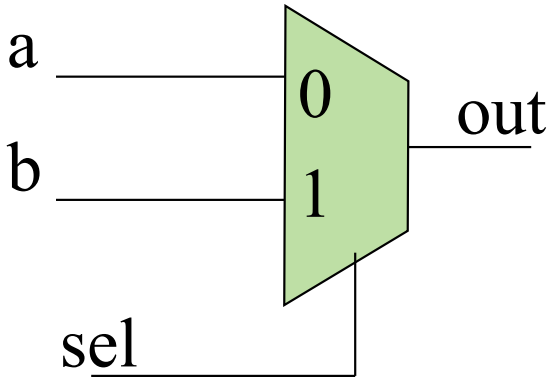
结构描述

(门级描述)

```
module mux2to1(a, b, sel, out);
  input a, b, sel; //定义输入信号
  output out; //定义输出信号
  wire selnot,a1,b1; //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(selnot, sel);
  and U2(a1, a, selnot);
  and U3(b1, b, sel);
  or  U4(out, a1, b1);
 endmodule
```

电路结构描述

例 2选1数据选择器的程序实例




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

assign out= sel ? b : a;

endmodule
```

数据流描述

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

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

endmodule
```

数据流描述

例 2选1数据选择器的程序实例

行为描述

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

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

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

2.2.2 Verilog HDL程序的基本结构

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,…）；

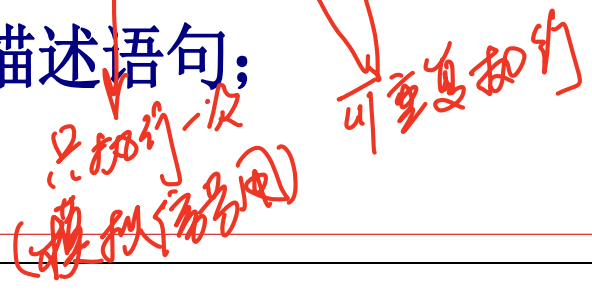
端口类型说明(input, output, inout)；
参数定义(可选)；
数据类型定义(wire, reg等)；

} 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



几种描述方式小结：

•结构描述（门级描述）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。

•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。

•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

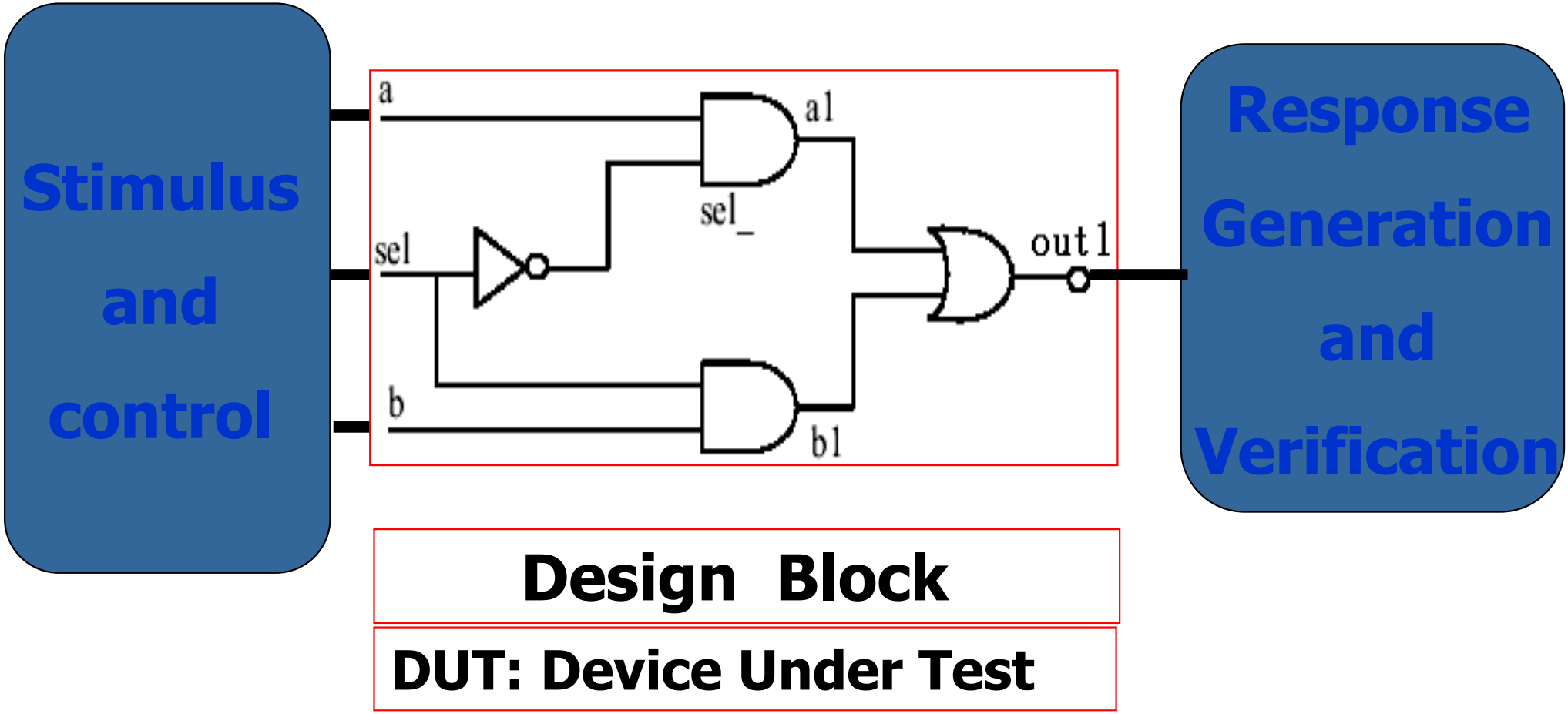
2.3 逻辑功能的仿真验证过程

HDL产生的最初动因就是为了能够模拟硬件系统，可以分析系统的性能，验证其功能是否正确。

要测试一个设计块是否正确，就要用Verilog再写一个测试模块。这个测试模块应包括以下三个方面的内容：

- 测试模块中要调用到设计块，只有这样才能对它进行测试；
- 测试模块中应包含测试的激励信号源；
- 测试模块能够实施对输出信号的检测，并报告检测结果；

写出测试模块的过程又称为搭建测试平台（test bench）



例：2选1数据选择器的测试模块：

```
module test_mux;
   reg a,b,s;
   wire out;
   mux2to1 u1(out, a, b, s);
   initial
   begin
      a=0; b=1; s=0;
#10   a=1; b=1; s=0;
#10   a=1; b=0; s=0;
#10   a=1; b=0; s=1;
#10   a=1; b=1; s=1;
#10   a=0; b=1; s=1;
#10   $finish;
    end
  initial
$monitor($time, “a=%b
b=%b s=%b out=%b” ,
a,b,s,out);
endmodule
```

时间延迟

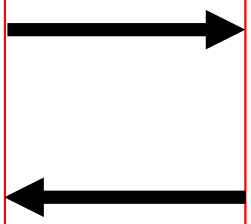
```
module
mux2to1(out,a,b,sel);
  output out;
  input a,b,sel;

  not (selnot, sel);
  and (a1, a, selnot);
  and (b1, b, sel);
  or (out1, a1, b1);
endmodule
```

```
0   a=0 b=1 s=0 out=0
10  a=1 b=1 s=0 out=1
20  a=1 b=0 s=0 out=1
30  a=1 b=0 s=1 out=0
40  a=1 b=1 s=1 out=1
50  a=0 b=1 s=1 out=1
```

测试激励块与设计块之间的关系

```
module test_mux;
reg a,b,s;
wire out;
mux2to1 u1(out, a, b, s);
```



```
module mux2to1(out,a,b,sel);
input a,b,sel;
output out;
```

- 仿真时，信号线**a**、**b**、**s**上要加一组测试激励信号，这组激励信号的产生，是通过**initial**内部的过程语句产生的，而过程语句只能给**reg**型变量赋值。
- 仿真时，信号线**a**、**b**、**s**上的激励信号是不能消失的，需要有“寄存”效应，能够描述这种“寄存”行为的，只能是**reg**型。

2.3.2 仿真过程简介

- 使用软件ModelSim-Altera 6.5b Starter Edition进行仿真验证的大致过程

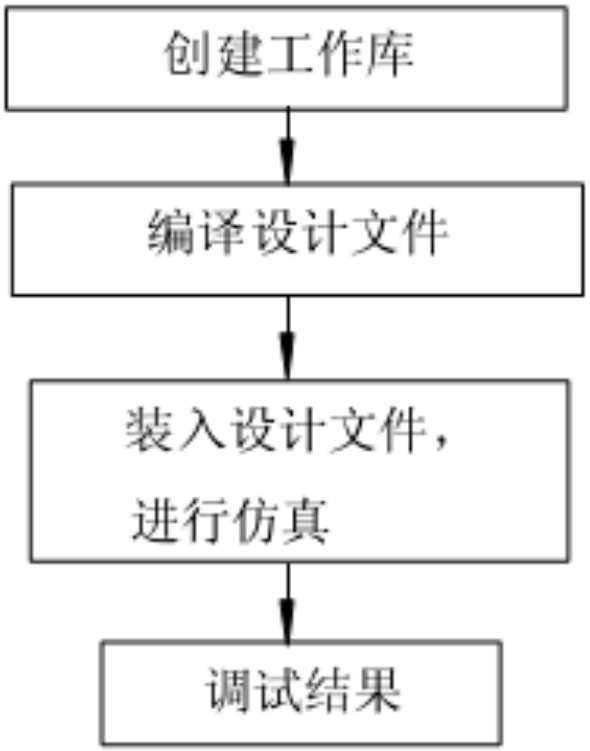


图 2.3.1 基于工作库仿真的基本流程

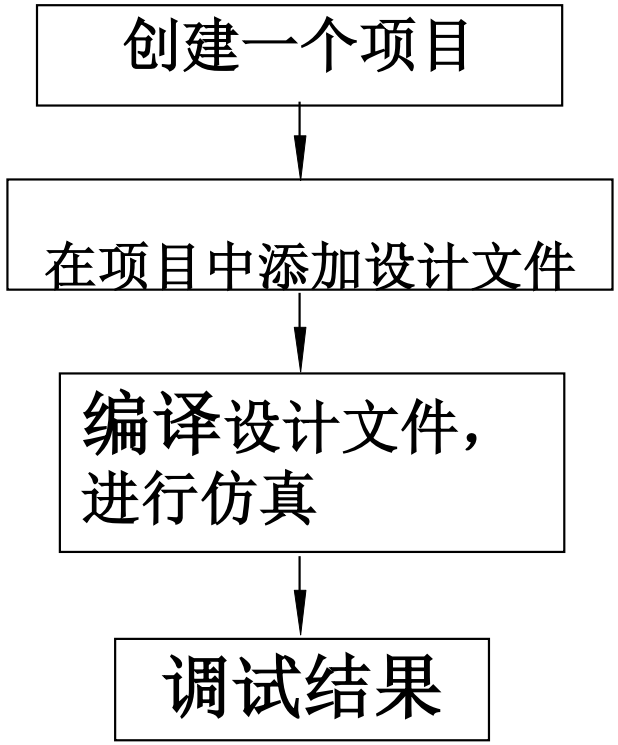
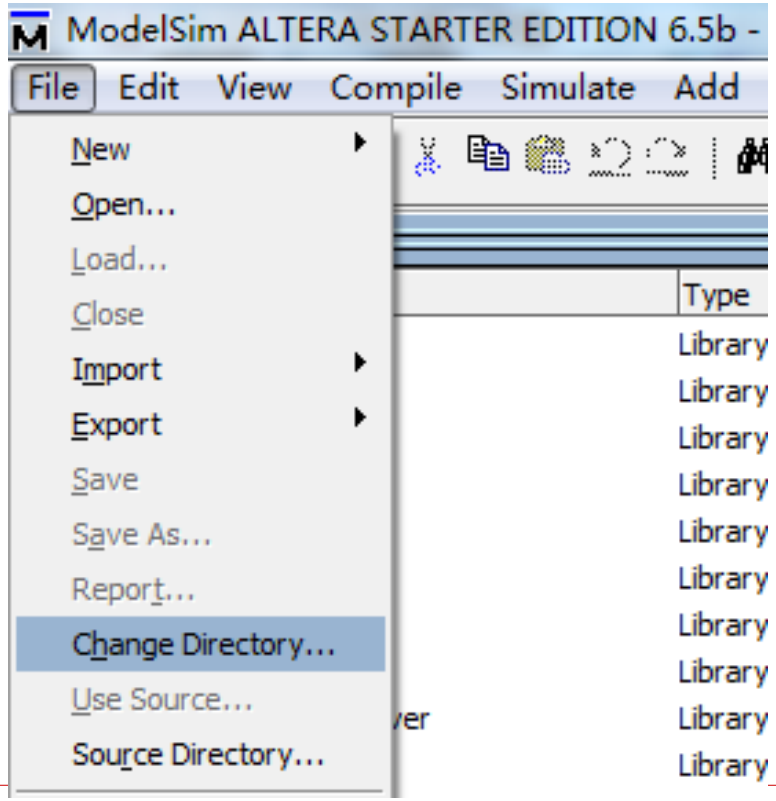
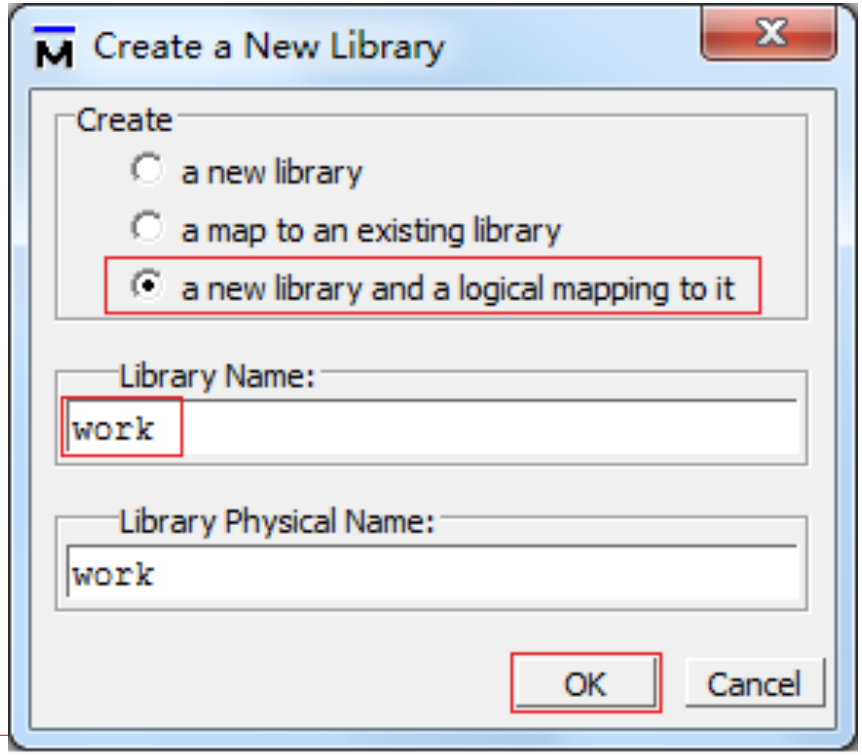


图2.3.2 基于工程项目仿真的基本流程

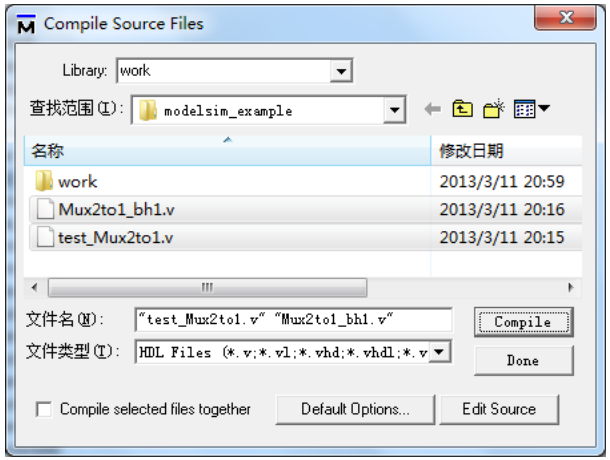
2.4 ModelSim仿真软件的使用

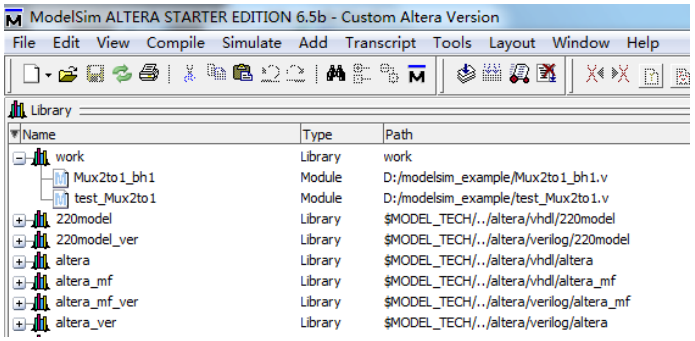
- 创建一个工作目录
- 输入源文件
- 建立工作库



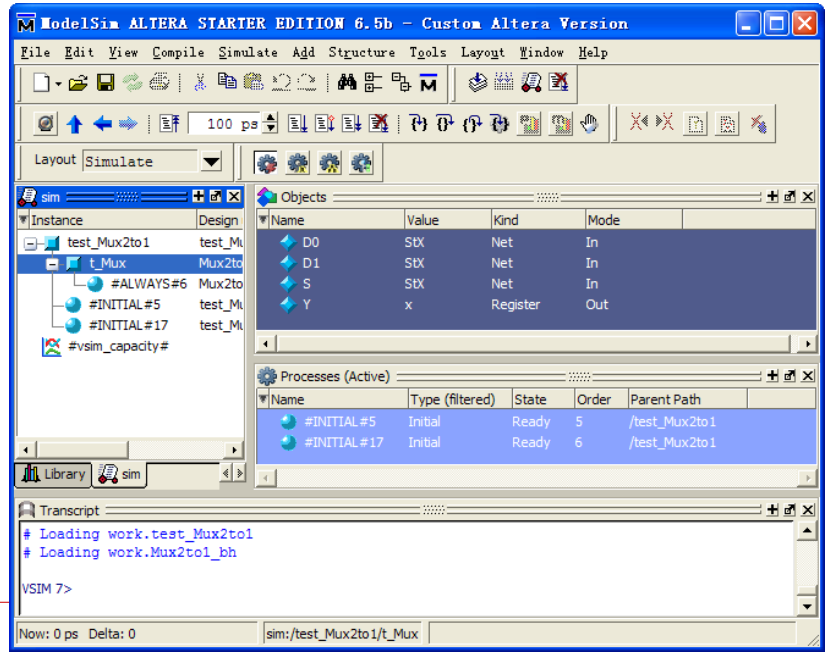


编译设计文件

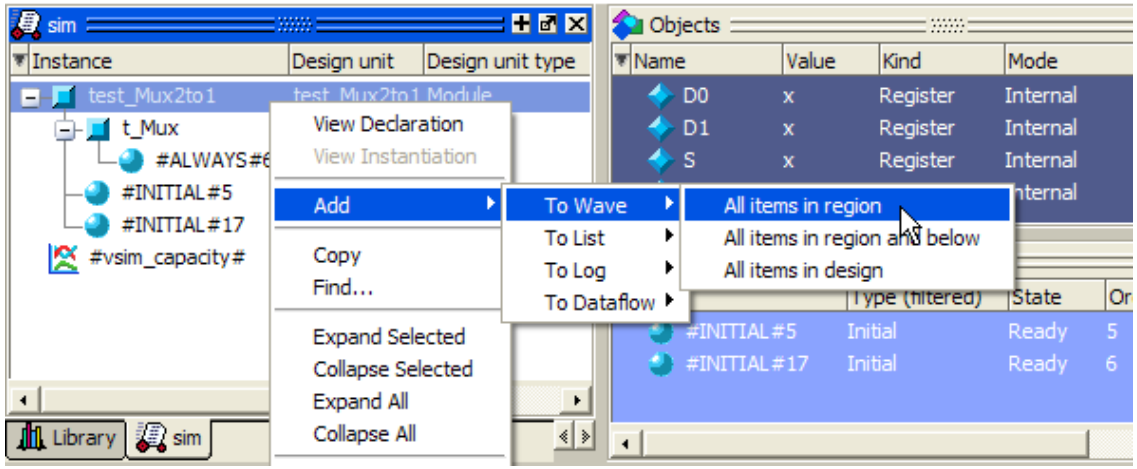


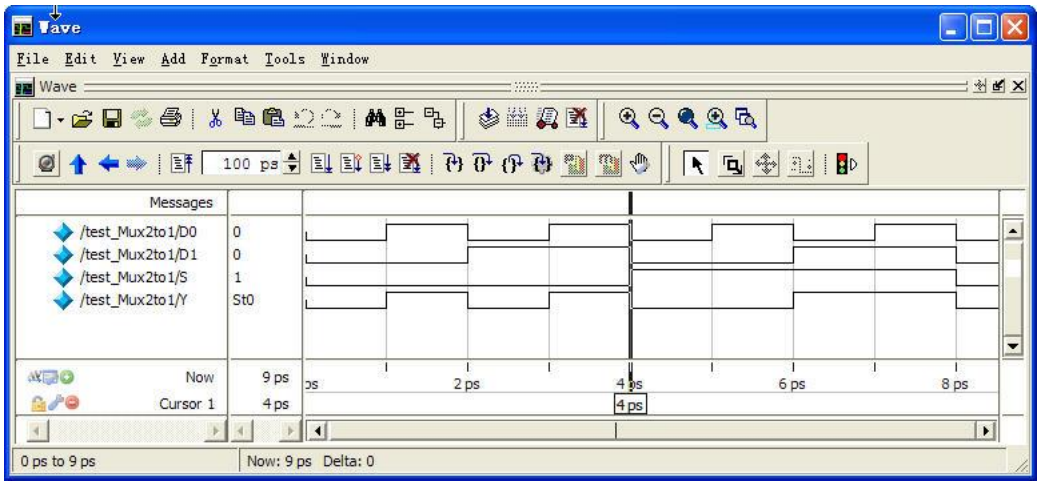


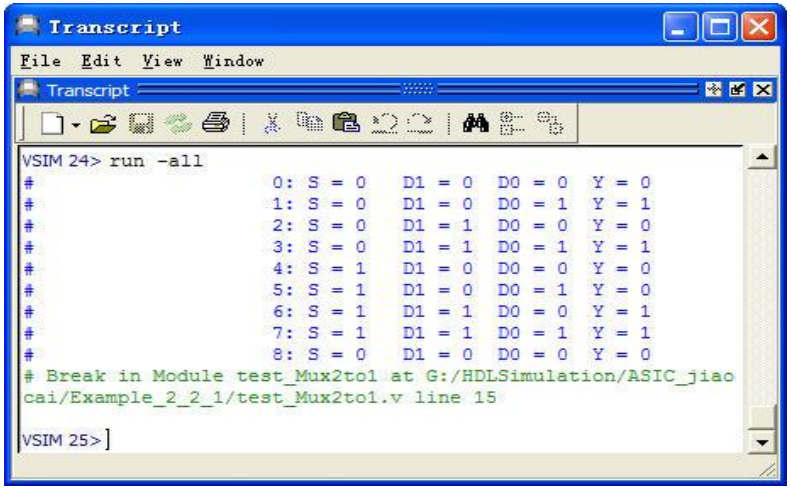
装入设计文件到仿真器



运行仿真器


sim
Instance
Design unit
Design unit type
test_Mux2to1
test_Mux2to1 Module
t_Mux
#ALWAYS#6
#INITIAL#5
#INITIAL#17
#vsim_capacity#
View Declaration
View Instantiation
Add
Copy
Find...
Expand Selected
Collapse Selected
Expand All
Collapse All
To Wave
To List
To Log
To Dataflow
All items in region
All items in region and below
All items in design
Library
sim
Objects
Name
Value
Kind
Mode
D0
D1
S
Register
Internal
Type (filtered)
State
#INITIAL#5
#INITIAL#17
Initial
Ready


Wave
File Edit View Add Format Tools Window
Messages
/test_Mux2to1/D0
/test_Mux2to1/D1
/test_Mux2to1/S
/test_Mux2to1/Y
Now
Cursor 1
2 ps
4 ps
6 ps
8 ps
0 ps to 9 ps
Now: 9 ps Delta: 0


Transcript
File Edit View Window
VSIM 24> run -all
0: S = 0 D1 = 0 D0 = 0 Y = 0
1: S = 0 D1 = 0 D0 = 1 Y = 1
2: S = 0 D1 = 1 D0 = 0 Y = 0
3: S = 0 D1 = 1 D0 = 1 Y = 1
4: S = 1 D1 = 0 D0 = 0 Y = 0
5: S = 1 D1 = 0 D0 = 1 Y = 0
6: S = 1 D1 = 1 D0 = 0 Y = 1
7: S = 1 D1 = 1 D0 = 1 Y = 1
8: S = 0 D1 = 0 D0 = 0 Y = 0
Break in Module test_Mux2to1 at G:/HDLSimulation/ASIC_jiaocai/Example_2_2_1/test_Mux2to1.v line 15
VSIM 25>

2.5 Verilog HDL功能仿真常用命令

2.5.1系统任务（System Tasks）

1．显示任务（Display Task）

$display是Verilog中最有用的任务之一，用于将指定信息（被引用的字符串、变量值或者表达式）以及结束符显示到标准输出设备上。其格式如下：

```
$display (format_specification1, argument_list1,
          format_specification2, argument_list2,
          ……,) ;
```

$monitor任务的参数格式与$display的相同

2.5 Verilog HDL功能仿真常用命令

表 2.5.1 字符串的显示格式说明

格式符	显示说明	格式符	显示说明
%d or %D	用十进制数显示变量	%v or %V	显示强度
%b or %B	用二进制显示变量	%o or %O	用八进制显示变量
%s or %S	显示字符串	%t or %T	显示目前时间格式
%h or %H	用十六进显示变量	%e or %E	用科学记数方式显示实数（例如 3e10）
%c or %C	显示 ASCII 字符	%f or %F	用十进制方式显示实数（例如 2.13）
%m or %M	显示层次名(不需参数)	%g or %G	选择科学记数和十进制方式中较短的来显示实数

2.5.1系统任务（System Tasks）

3. 仿真的中止 (Stopping) 和 完成(Finishing)任务

- $stop；//在仿真期间，停止执行，未退出仿真环境。
- $finish；//仿真完成，退出仿真环境，并将控制权返回给操作系统
- 系统任务$stop使得仿真进入交互模式，然后设计者可以进行调试。当设计者希望检查信号的值时，就可以使用$stop使仿真器被挂起。然后可以发送交互命令给仿真器继续仿真。

2.5.2 编译器指令(Compiler Directives)

以`（反撇号，用键盘上左边与~共用的键输入）开头的标识符就是编译指令，用来控制代码的整个过程。在Verilog代码编译的整个过程中，编译器指令始终有效（编译过程可能跨越多个文件），直至遇到其他不同的编译器指令为止。

```
`timescale time_unit/time_precision

`include "../../header.v “

`define  WORD_SIZE  32 //定义文本宏
```