第5章 逻辑电路仿真设计

5.1 硬件仿真软件使用

一、实验目的

掌握硬件仿真软件的使用方法。

二、实验条件和环境

微型计算机，Windows 操作系统，tecs-software-suite-2.5 软件。

三、实验任务和要求

实现异或门。

四、实验步骤和操作指导

1. 有关硬件仿真软件

本章的微机组成实验采用了以色列科学家 Noam Nisan 等人开发的开源软件包 tecs-software-suite-2.5，该软件包用 Java 语言编写，最新版为 2.5，文件约有 600KB，可以在 Windows、UNIX 和 Mac OS 等操作系统中运行。

tecs-software-suite-2.5 软件包中的硬件仿真器可以根据 HDL（硬件描述语言）的描述在内存中构造芯片的内存映像，利用由不同测试场景构成的测试脚本对芯片进行仿真，最后通过输出文件与比较文件的对比结果以验证芯片设计的正确性。

2. 有关 HDL

硬件描述语言（Hardware Description language，HDL）是一种用于定义和测试芯片的规范：芯片的接口是由负责传输布尔信号的输入以及输出引脚构成的，而芯片的主体则是由互联的底层芯片组成的。

虽然从不同角度考虑，如效率、速度、成本等，芯片的物理结构可以多种多样，但通常

在设计芯片时普遍遵循的原则应该是：用尽可能少的门电路实现尽可能多的功能。

图 5-1 表示一位异或门的逻辑结构，由于 $a \oplus b = (a \wedge \bar{b}) \vee (\bar{a} \wedge b)$，则该异或门可以用 2 个非门、2 个与门和 1 个或门实现，该异或门也可以用下面的 Xor. hdl 文件进行描述。

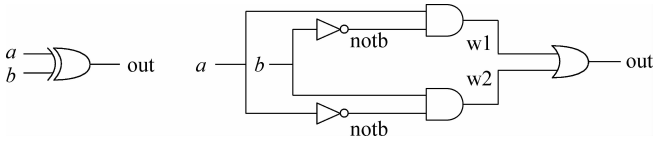

图 5-1　一位异或门的逻辑结构

```
CHIP Xor {     //芯片名称
    IN a, b;   //a、b 是异或门的输入引脚
    OUT out;   //out 是异或门的输出引脚
PARTS:
    Not(in=a,out=nota);        //非门的输入引脚 in 连接 a,输出引脚 out 连接 nota
    Not(in=b,out=notb);        //非门的输入引脚 in 连接 b,输出引脚 out 连接 notb
    And(a=a,b=notb,out=w1);  //与门的输入引脚 a、b 分别连接 a 和 notb,输出引脚 out 连
                                 接 w1
    And(a=nota,b=b,out=w2);  //与门的输入引脚 a、b 分别连接 nota 和 b,输出引脚 out 连
                                 接 w2
    Or(a=w1,b=w2,out=out);   //或门的输入引脚 a、b 分别连接 w1 和 w2,输出引脚 out 连
                                 接 out
}
```

对于上述 HDL 文件，需要说明的是：

(1) Xor 定义在以. hdl 为后缀的文本文件中。芯片的定义由描述头(header)和描述体(parts)组成。描述头作为芯片的 API，定义了芯片的接口，包括芯片的名称、输入输出引脚的名称，芯片和引脚的名称可以是由任意字母和数字组成的序列，但不能以数字开头，通常芯片的名称以大写字母开头，引脚以小写字母开头。描述体描述了所有底层芯片的名称和物理结构，每条语句描述了芯片的名称以及与其他芯片的连接方式。

(2) 本例中的 nota、notb、w1 和 w2 是内部引脚，用于将一个芯片的输出引脚连接到其他芯片的输入引脚上。每个内部引脚只能作为一个芯片的输出引脚，但可以作为多个芯片的输入引脚。

(3) 芯片引脚的信号源可以是芯片的引脚或者内部引脚，如 Not 芯片的输入信号源是 Xor 芯片的输入引脚，Not 芯片的输出信号源是内部引脚 nota。注意：每个输入引脚只能有一个信号源，如 And(a=a,a=b,…)就是一条错误的语句。

(4) 引脚的默认数据宽度为 1b，如 a、b 和 out 的数据宽度均为 1b。

(5) 通常选择仿真器中的 load file 菜单命令或者单击快捷按钮可以打开一个已经设计好的 HDL 文件，例如通过该方式可以在仿真器中打开 Xor. hdl。如果本例中 And、Or 和 Not 等芯片是自己用 HDL 文件设计的，则在加载 Xor. hdl 时，这些 HDL 文件也应该加载到仿真器中，最简单的方法是将这些芯片的 HDL 文件与 Xor. hdl 放在同一目

录中。

(6) 如果在本例中想使用系统提供的 And、Or 和 Not 等芯片，只要在存放 Xor. hdl 的目录中不包含这些芯片的 HDL 文件，系统就会自动调用 builtInChips 目录中各芯片的内置版本，内置版本的接口可以通过查看该模块的 HDL 文件了解，其功能已经用 Java 语言实现，实现细节已被 HDL 接口屏蔽。如本例中 And 和 Or 模块的内置版本定义的输入引脚是 *a* 和 *b*，输出引脚是 out；Not 内置版本定义的输入引脚是 in，输出引脚是 out。

(7) 芯片的内置版本不仅可以用于构建其他芯片，还可以减少芯片开发复杂性，加快仿真速度。开发复杂芯片时，为了使设计者专注于构建和测试芯片的逻辑功能，无须考虑底层芯片的实现细节，使用底层芯片的内置版本可以提高芯片设计的准确性。由于芯片内置版本的速度和内存方面比自己设计的芯片占有优势，因此使用内置版本还可以加快仿真速度。此外，由于内置版本的芯片都具有 GUI 功能，因此当这些芯片加载到硬件仿真器时，可以图形化方式显示芯片的内容和变化，如内存单元的存储内容等。

3. 有关测试脚本

硬件仿真器使用脚本语言编写的测试脚本测试芯片的功能。测试脚本中说明了所要加载的 HDL 文件、输出信息以及一系列的测试场景。每个测试场景都是一组输入值的组合，硬件仿真器会自动计算出相应的测试结果，并将该结果记录到指定的输出文件中。对于一些简单的门电路，可以用穷举法列出所有可能的输入组合；对于较复杂的芯片，可以使用一些有代表性的输入组合。

选择仿真器中的 Load Script 菜单命令或者单击快捷按钮可以打开一个已经设计好的测试脚本。图 5-2 是一位异或门的测试脚本 Xor. tst，该脚本与 Xor. hdl、Xor. out 和 Xor. cmp 等文件存放在同一目录中。

头部	load Xor.hdl, //将 Xor.hdl 加载到硬件仿真器中 output-file Xor.out, //将仿真结果输出到文件 Xor.out 中 compare-to Xor.cmp, //对仿真结果与比较文件 Xor.cmp 进行比较 output-list a%B3.1.3 b%B3.1.3 out%B3.1.3; //设置测试结果的输出格式			
场景	场景 1	场景 2	场景 3	场景 4
	set a 0,//将 a 设置为 0 set b 0,//将 b 设置为 0 eval, //仿真 output;//输出	set a 0, //将 a 设置为 0 set b 1, //将 b 设置为 1 eval, //仿真 output; //输出	set a 1, //将 a 设置为 1 set b 0, //将 b 设置为 0 eval, //仿真 output; //输出	set a 1, //将 a 设置为 1 set b 1, //将 b 设置为 1 eval, //仿真 output; //输出

图 5-2 Xor. tst 脚本

对于上述脚本文件，需要说明的是：

(1) 脚本命令可以分为两种，其中设置命令用于加载文件以及对模拟过程初始化，如图 5-2 中头部出现的各种命令；模拟命令用于说明测试场景的测试步骤，如图 5-2 中场景部分出现的各种命令。

(2) 脚本命令以“,”或者“;”结尾，其中“,”表示结束一条命令，“;”表示结束一条命令和一个模拟步骤(由一条或者多条命令组成)。

(3) output-list 命令对于每个输出变量的格式规定如下：padL.len.padR，其中 padL 和 padR 分别表示输出变量左边和右边的空格数，len 表示输出变量的列宽，此外，也可以在该格式前加入%X、%B 和%D 等前缀，分别表示十六进制、二进制和十进制数，如 output-list a%B3.1.3 表示输出结果为“ a ”且 a 用二进制数表示。如果用%S 作为前缀，表示数据的类型是字符，字符的内容必须包含在双引号中。

(4) set 命令可以给变量赋值，如引脚等，参与赋值操作的数值应该与变量的数据宽度保持一致，如 set a 0 表示将引脚 a 设置为 0，由于 a 能够表示 1 位二进制数，因此该赋值正确。

(5) eval 命令将指示仿真器应用所构建的芯片对当前输入引脚的数值进行计算。

(6) out 命令将获取 output-list 命令列出的所有变量值并按照格式要求在输出文件中输出一行。如果输出行与比较文件不一致，将显示错误信息并停止脚本执行。

a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

图 5-3 Xor.out

4. 有关比较文件

比较文件是用测试脚本对 HDL 文件进行测试时应该输出的正确信息。图 5-3 是用测试脚本 Xor.tst 对 Xor.hdl 进行仿真时生成的输出文件 Xor.out，如果它与比较文件 Xor.cmp 中的信息一致，则说明所设计的异或门符合要求。

5. 启动硬件仿真软件

将 tecs-software-suite-2.5 软件包解压后，双击 HardWareSimulator.bat 即可打开硬件仿真器。如果系统没有安装 Java 运行环境(JRE)，用户还需要安装 JRE，JRE 可以自由下载，但要保证操作系统的 PATH 变量中必须包含 JDK 的安装路径。提示：选择“控制面板”→“系统”→“高级系统设置”。在“系统属性”的“高级”选项页中，单击“环境变量”按钮，在“系统变量”中选择 path，单击“编辑”按钮，将 JDK 的安装路径加入到“变量值”内容的最后。

6. 仿真实现

硬件仿真器的界面如图 5-4 所示，由菜单栏、快捷工具栏、输入引脚区、输出引脚区、HDL 区、内部引脚区、测试脚本区等构成。

- 菜单栏：主要由 File、View 和 Run 等子菜单构成，其中 File 菜单用于加载芯片和脚本；View 菜单可以设置执行方式、显示和数字的格式；Run 菜单用于设置测试脚本运行的方式。
- 快捷工具栏：位于菜单栏下方，从左至右的快捷按钮依次表示加载芯片、单步执行、执行、停止、重置、重新计算、定时器、加载脚本、打开断点模板、设置速度、执行方式、设置表示方式和显示方式。
- 输入引脚区：用于设置输入引脚的名字和数值。
- 输出引脚区：显示输出引脚的名字和数值。

- HDL 区：显示 HDL 描述的芯片逻辑。
- 内部引脚区：显示芯片内部引脚的名字和数值。
- 测试脚本区：用于实时显示脚本的执行过程。

在硬件仿真器中，选择 Load Chip 菜单命令或者单击快捷按钮打开所选择的 HDL 文件，图 5-4 是打开 Xor. hdl 后的状态。单击 HDL 区中文件的 PARTS 时，可以查看描述体中用到的所有模块的属性信息，如芯片名称、类型、是否是时序芯片等。如果单击 HDL 区中 PARTS 中的模块，还可以查看该模块的引脚及其取值。如果单击 Input pins 中的引脚，可以更改其值，单击 Eval 图形按钮，可以在 Output pins 中查看输出引脚值的变化。

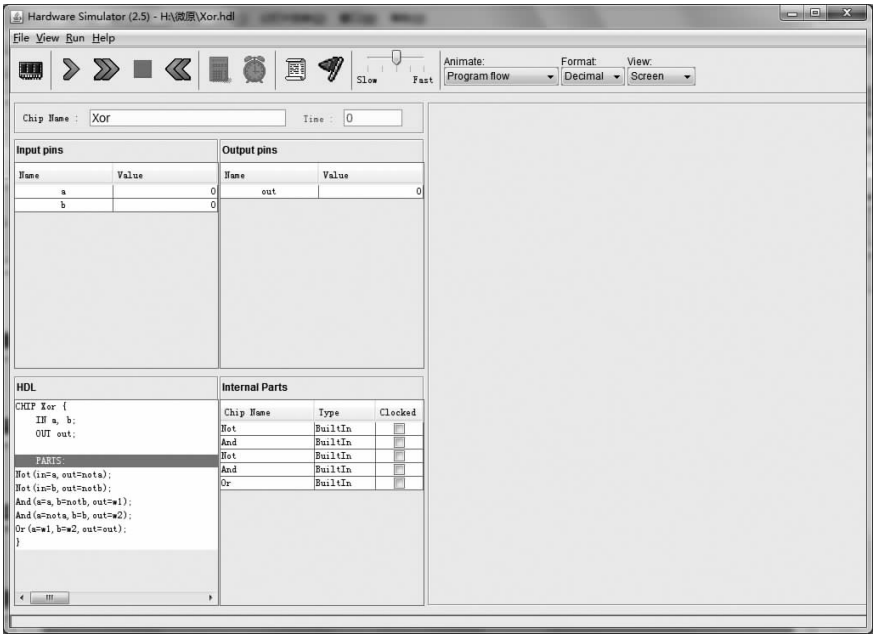

图 5-4　在硬件仿真器中记载 HDL 文件后的状态

在硬件仿真器中选择 Load Script 菜单命令或者单击快捷按钮打开所选择的脚本文件，图 5-5 是打开 Xor. tst 后的状态。按 F5 键或者单击(Run)按钮，将按照测试脚本中的场景验证芯片设计是否正确。如果测试通过，会在状态栏中显示“End of script-comparison ended successful”信息；如果测试未通过，可以单击(Reset)按钮后，再单击(Single Step)按钮单步跟踪仿真过程，定位出错的场景，以便修改 HDL 文件。如果在 View 下拉框中选择 Output，还可以查看测试过程中生成的输出文件。

五、实验报告要求

(1) 将实验仿真运行结果截图粘贴到实验报告中，并分析测试脚本的含义。

(2) 写出实验过程中遇到的问题及原因。

(3) 本次实验过程中最大的体会是什么？学会了哪些技能？

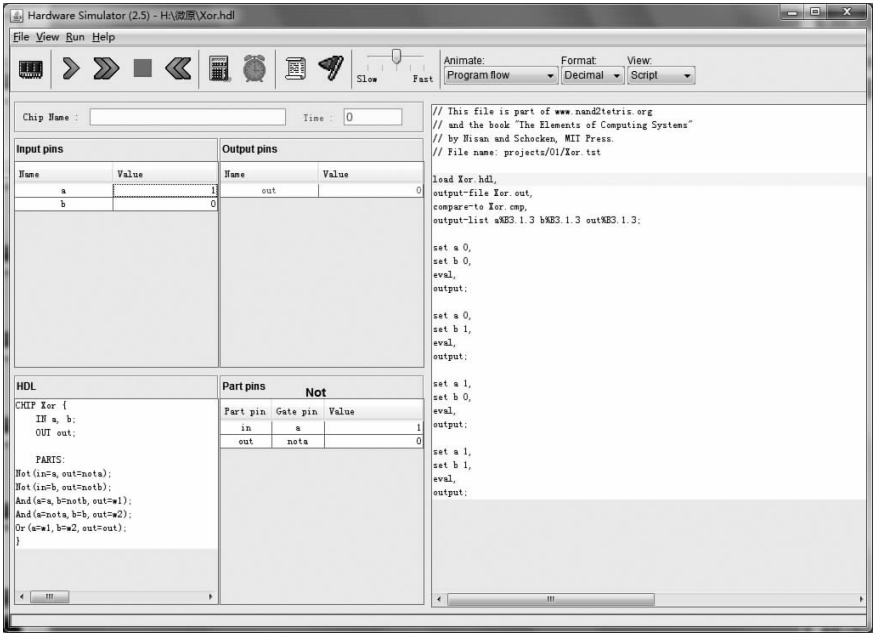

图 5-5　在硬件仿真器中加载脚本文件后的状态

六、实验思考题

还可以怎样设计异或芯片的物理结构？请画出原理图。

5.2　实现一位逻辑门

一、实验目的

(1) 理解逻辑运算。

(2) 掌握构建一位非门、与门、或门、复用器和选择器的方法。

二、实验条件和环境

微型计算机，Windows 操作系统，tecs-software-suite-2.5 软件。

三、实验任务和要求

(1) 实现一位非门。

(2) 实现一位与门。

(3) 实现一位或门。

(4) 实现一位多路复用器。

(5) 实现一位多路选择器。

四、实验步骤和操作指导

1. 一位逻辑门的实现原理

计算机的底层物理结构是由一组基本的逻辑门构成的，本节所有实验都建立在与非门基础之上，在 HDL 文件中可以直接使用与非门，仿真器会自动调用与非门的实现文件。与非门 Nand 的输入引脚是 a 和 b，输出引脚是 out，输入和输出之间的关系为：if (a ==1 and b==1) set out=0 else set out=1。

无论布尔函数多么复杂，都可以使用与、或、非 3 种基本运算实现，而实现这 3 种逻辑运算的逻辑门又可以使用与非门实现，例如非运算 $\bar{a}=\overline{a \wedge a}$，与运算 $a \wedge b=\overline{\overline{a \wedge b}}=\overline{\overline{a \wedge b} \wedge \overline{a \wedge b}}$，或运算 $a \vee b=\overline{\overline{a \vee b}}=\overline{\bar{a} \wedge \bar{b}}=\overline{\overline{a \wedge a} \wedge \overline{b \wedge b}}$。一位非门、与门和或门的结构如图 5-6 所示，其中非门输入和输出之间的关系为：out = not in；与门输入与输出之间的关系为：if (a == 1 and b == 1) set out = 1 else set out = 0；或门输入与输出之间的关系为：if (a == 1 or b == 1) set out = 1 else set out = 0。它们分别可以用 1 个、3 个、3 个与非门实现。

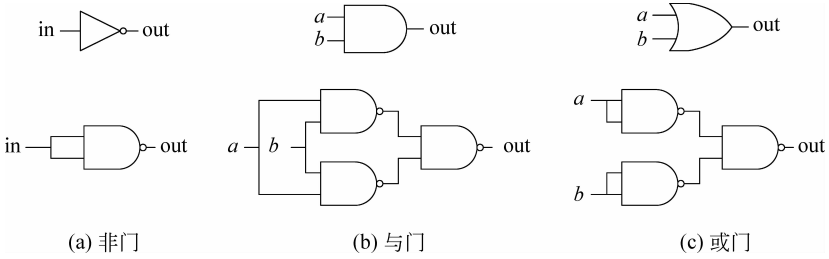

图 5-6　非门、与门、或门的逻辑结构和物理结构

另一种常用的组合逻辑门是复用器，它主要用于分时传递数据，由选择信号确定由哪一个支路输出。一位复用器的结构如图 5-7 所示，输入与输出之间的关系为：if (sel == 0)set out = a；else set out = b，即当选择信号 sel 为 0 时，确定输出由输入 a 确定，否则输出由输入 b 确定，可以用 1 个非门、2 个与门和 1 个或门实现其功能。

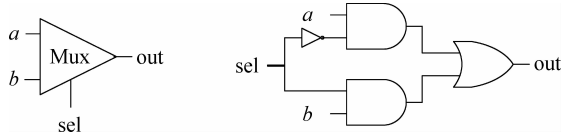

图 5-7　一位复用器逻辑结构和物理结构

选择器是另一种常用的组合逻辑门，但它与复用器的功能正好相反，由选择信号确定输入由哪一个支路输出。一位选择器的结构如图 5-8 所示，输入与输出之间的关系为：if (sel == 0)set {a, b} = {in, 0} else set {a, b} = {0, in}，即当选择信号 sel 为 0 时，确定输入由 a 输出，否则输入由 b 输出，可以用 1 个非门和 2 个与门实现其功能。

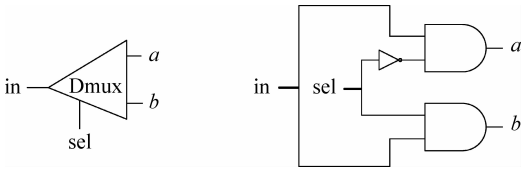

图 5-8　一位选择器的逻辑结构和物理结构

2. 一位逻辑门的仿真实现

按照实验所提供的 HDL 框架文件，在文本编辑器中完成逻辑门的设计。在硬件仿真器中，打开所设计的 HDL 文件(.hdl)和对应的测试脚本(.tst)，并利用测试脚本验证芯片设计的正确性。建议将 HDL 文件、测试脚本和比较文件放在同一路径中。注意：在设计芯片时，如果要用到其他芯片，为了保证该芯片的正确性，建议使用这些芯片的内置版本，即在路径中仅包含当前所设计芯片的.hdl 文件。由于后面设计的芯片会用到前面设计的芯片，因此建议按照实验编排的顺序完成所有实验内容。注意：资源软件中包含了本章实验用到的所有 HDL 文件框架、测试脚本和比较文件。

1) 一位非门

一位非门的 HDL 文件为 Not.hdl，以下是该 HDL 文件的框架：

```
CHIP Not {
    IN in;
    OUT out;
    PARTS:
    // Put your code here:
}
```

一位非门的测试文件和比较文件如下所示：

<table>
<tr><th colspan="3">测试脚本 Not.tst</th><th>比较文件 Not.cmp</th></tr>
<tr><td>头部</td><td colspan="2">load Not.hdl,
output-file Not.out,
compare-to Not.cmp,
output-list in%B3.1.3 out%B3.1.3;</td><td rowspan="2">| in | out |
| 0 | 1 |
| 1 | 0 |</td></tr>
<tr><td>场景</td><td>set in 0,
eval,
output;</td><td>set in 1,
eval,
output;</td></tr>
</table>

2) 一位与门

一位与门的 HDL 文件为 And.hdl，以下是该 HDL 文件的框架：

```
CHIP And {
    IN a, b;
    OUT out;
```

```
    PARTS:
    // Put your code here:
}
```

一位与门的测试脚本和比较文件如下所示：

测试脚本 And. tst				比较文件 And. cmp
头部	load And.hdl, output-file And.out, compare-to And.cmp, output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;			| a | b | out | | 0 | 0 | 0 | | 0 | 1 | 0 | | 1 | 0 | 0 | | 1 | 1 | 1 |
场景	set a 0, set b 0, eval, output;	set a 0, set b 1, eval, output;	set a 1, set b 0, eval, output;	set a 1, set b 1, eval, output;

3）一位或门

一位或门的 HDL 文件为 Or. hdl，以下是该 HDL 文件的框架：

```
CHIP Or {
    IN a, b;
    OUT out;
    PARTS:
    // Put your code here:
}
```

一位或门的测试脚本和比较文件如下所示：

测试脚本 Or. tst				比较文件 Or. cmp
头部	load Or.hdl, output-file Or.out, compare-to Or.cmp, output-list a%B3.1.3 b%B3.1.3 out%B3.1.3;			| a | b | out | | 0 | 0 | 0 | | 0 | 1 | 1 | | 1 | 0 | 1 | | 1 | 1 | 1 |
场景	set a 0, set b 0, eval, output;	set a 0, set b 1, eval, output;	set a 1, set b 0, eval, output;	set a 1, set b 1, eval, output;

4）多路复用器的 Mux. hdl

一位多路复用器的 HDL 文件为 Mux. hdl，以下是该 HDL 文件的框架：

```
CHIP Mux {
    IN a, b, sel;
    OUT out;
    PARTS:
    // Put your code here:
```

```
}
```

一位多路复用器的测试脚本和比较文件如下所示：

测试脚本 Mux.tst				比较文件 Mux.cmp
头部	load Mux.hdl, output-file Mux.out, compare-to Mux.cmp, output-list a%B3.1.3 b%B3.1.3 sel%B3.1.3 out%B3.1.3;			| a | b | sel | out | | 0 | 0 | 0 | 0 | | 0 | 0 | 1 | 0 | | 0 | 1 | 0 | 0 | | 0 | 1 | 1 | 1 | | 1 | 0 | 0 | 1 | | 1 | 0 | 1 | 0 | | 1 | 1 | 0 | 1 | | 1 | 1 | 1 | 1 |
场景	set a 0, set b 0, set sel 0, eval, output; set sel 1, eval, output;	set a 0, set b 1, set sel 0, eval, output; set sel 1, eval, output;	set a 1, set b 0, set sel 0, eval, output; set sel 1, eval, output;	set a 1, set b 1, set sel 0, eval, output; set sel 1, eval, output;	

5）多路选择器的 DMux.hdl

一位多路选择器的 HDL 文件为 DMux.hdl，以下是该 HDL 文件的框架：

```
CHIP DMux {
    IN in, sel;
    OUT a, b;
    PARTS:
    // Put your code here:
}
```

一位多路选择器的测试脚本和比较文件如下所示：

测试脚本 Dmux.tst				比较文件 Dmux.cmp
头部	load DMux.hdl, output-file DMux.out, compare-to DMux.cmp, output-list in%B3.1.3 sel%B3.1.3 a%B3.1.3 b%B3.1.3;			| in | sel | a | b | | 0 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 1 | 0 | 1 | 0 | | 1 | 1 | 0 | 1 |
场景	set in 0, set sel 0, eval, output;	set in 0, set sel 1, eval, output;	set in 1, set sel 0, eval, output;	set in 1, set sel 1, eval, output;	

五、实验报告要求

（1）写出所设计的 HDL 文件，并说每条语句的含义。

（2）将实验仿真运行结果截图粘贴到实验报告中，并分析测试脚本的含义。