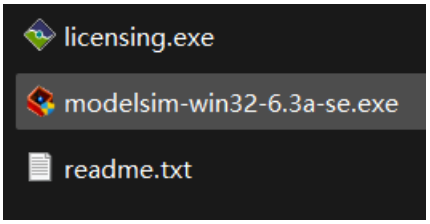
实验一　实验环境搭建与简单组合逻辑设计

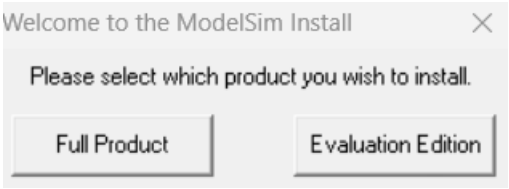
学号 2023370018
姓名 梁桐
班级 10012209
课程代码 U10M21001.01

modelsim-win32-6.3a 版本的安装

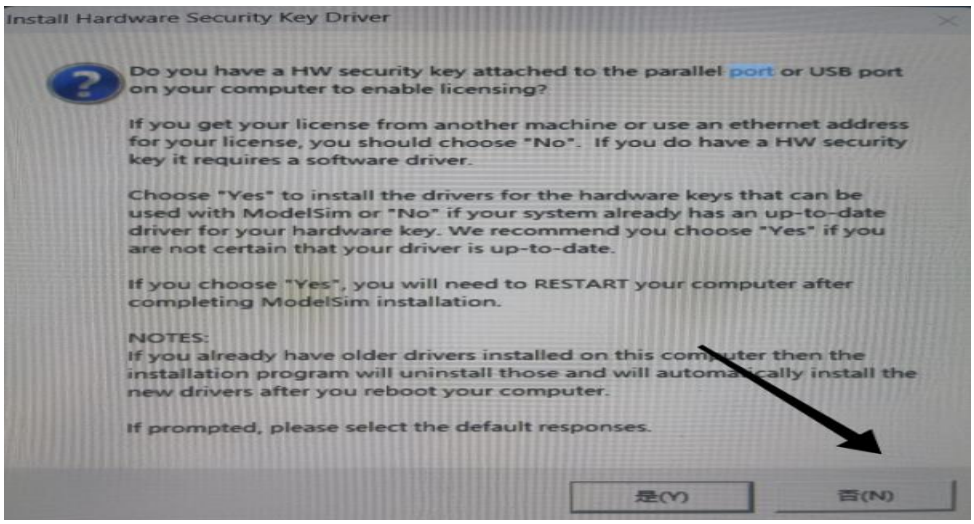
1. 双击运行 modelsim-win32-6.3a-se.exe



2. 点击 Full Product 启动安装

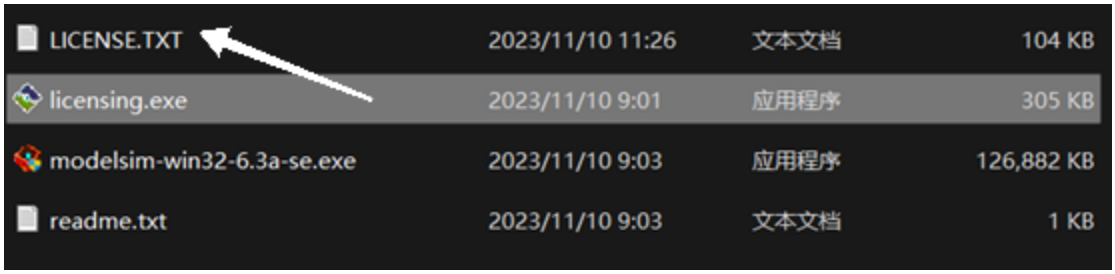


3. 遇到此窗口点击 否

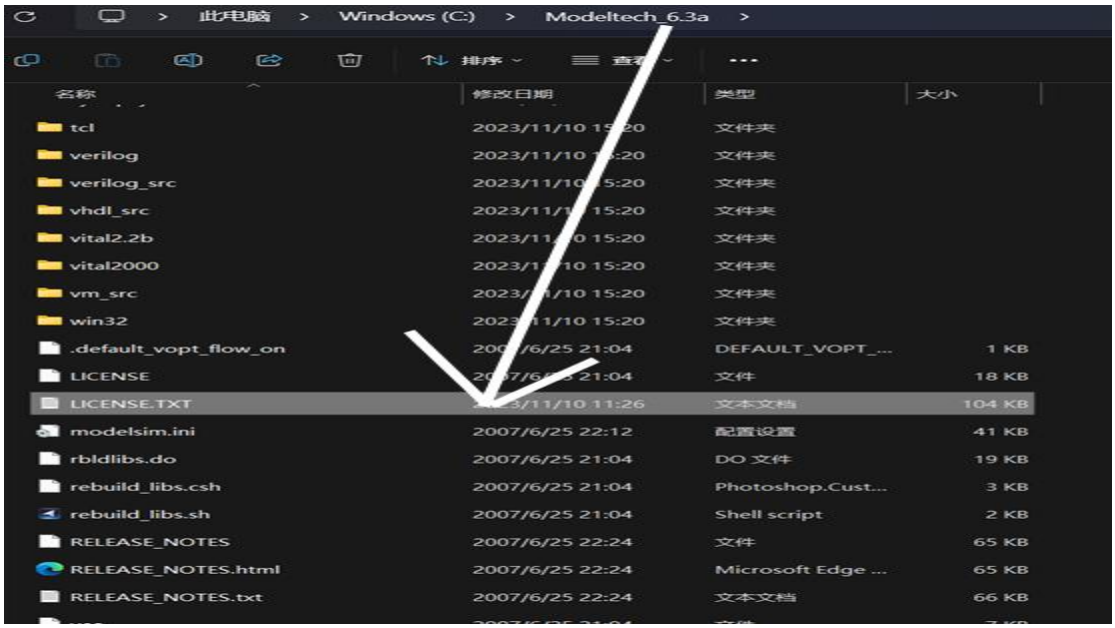


4. 遇到此窗口点击 Exit

5. 双击 licensing.exe 等待其跳转后刷新文件 LICENSE.TXT

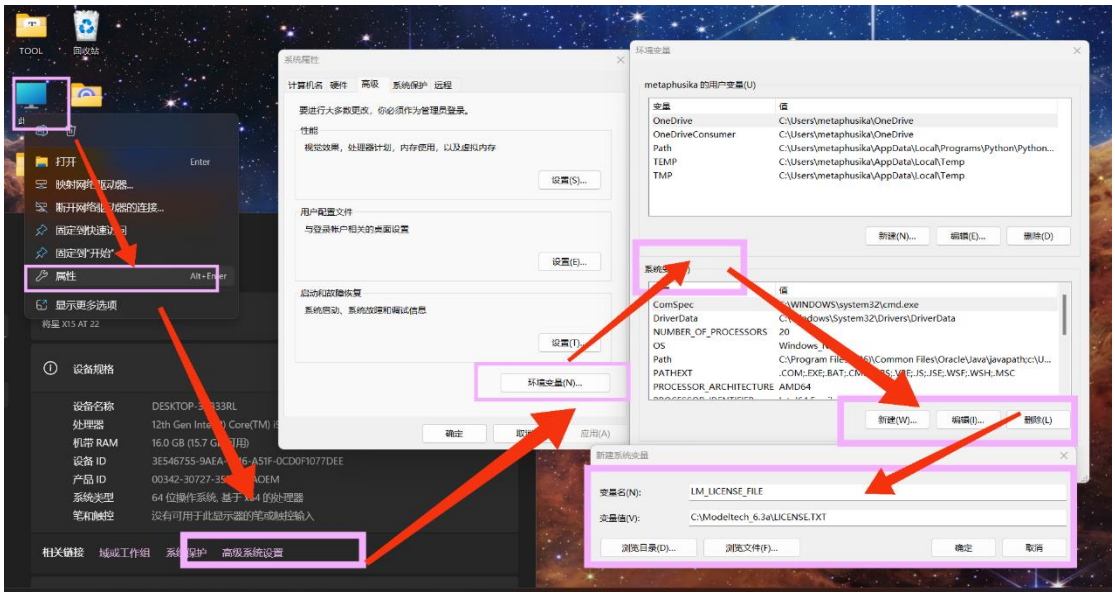


6. 将 LICENSE.TXT 文件粘贴至 C:\Modeltech_6.3a 目录



7. **按如图操作进行环境变量设置**：右击计算机图标选择“属性”->高级系统设置->环

境变量->新建系统变量->变量名：LM_LICENSE_FILE　　变量值：C:\Modeltech_6.3a\LICENSE.TXT



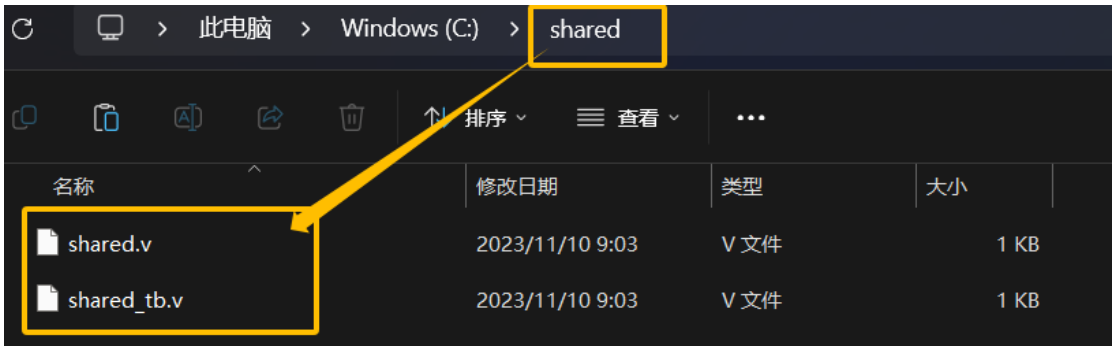
8. 安装完成

ModelSimSetup-15.0.0.145-windows 版本的安装

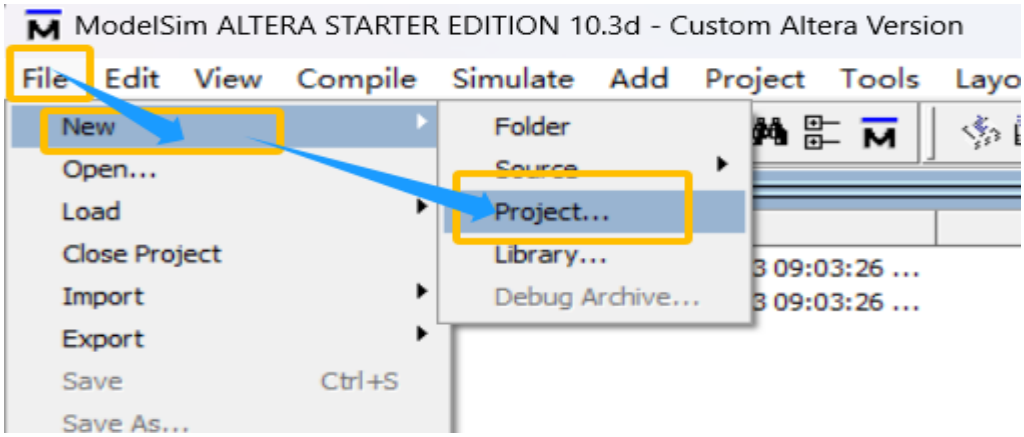
1. 双击应用程序按照默认选项操作安装即可

Figure2.72 的实现（使用 ModelSimSetup-15.0.0.145-windows 版本）

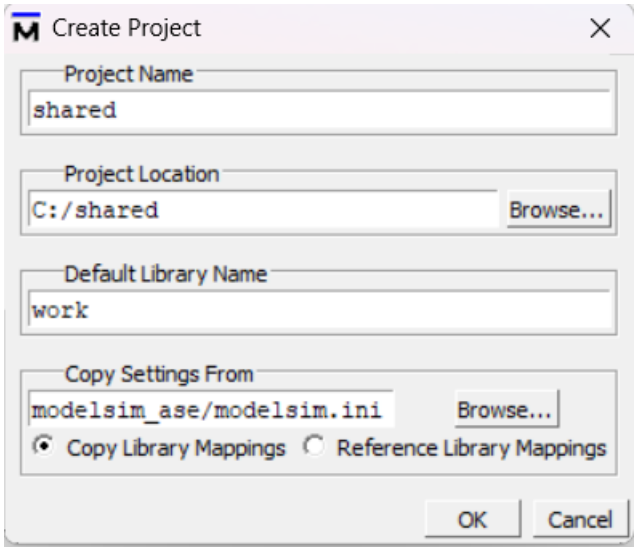
1. **编写设计文件** shared.v 和测试文件 tb_shared.v，新建目录 C:\shared，将以上两个文件复制到该目录



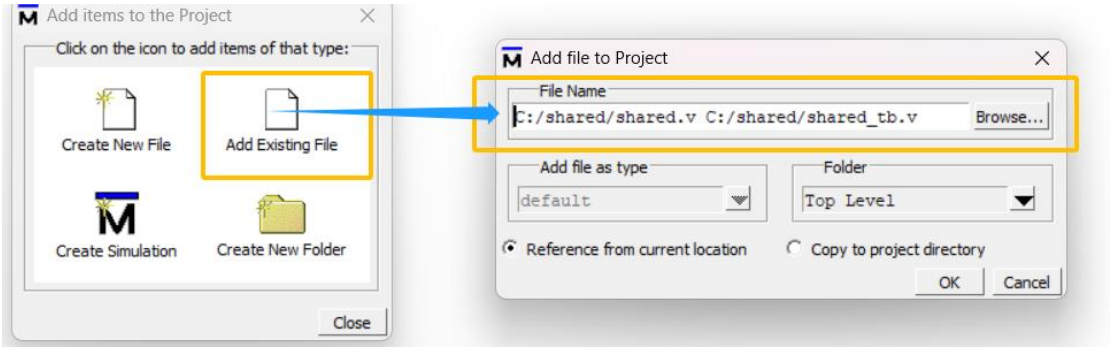
2. 按下图操作新建工程 New project



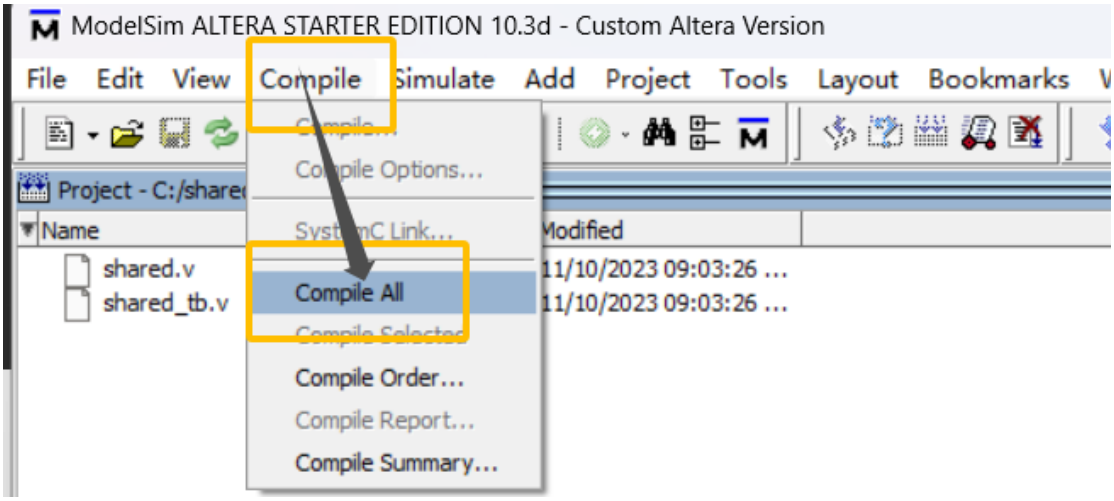
3. 工程名 shared，指定路径到之前目录 C:\shared



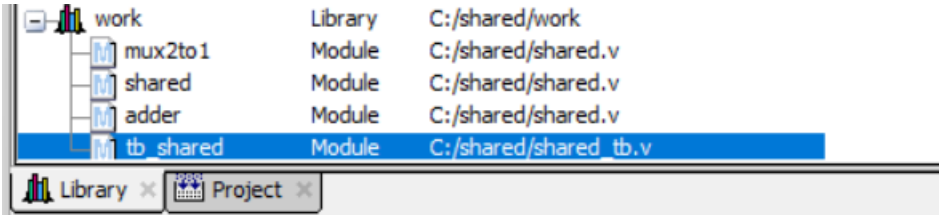
4. 单击 Add exiting file to project，将 shared.v 和 tb_shared.v 添加到工程中



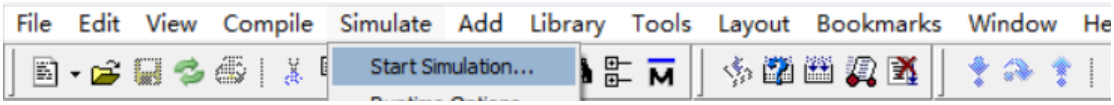
5. 单击 Compile 再单击 Compile All，编译源文件和测试文件



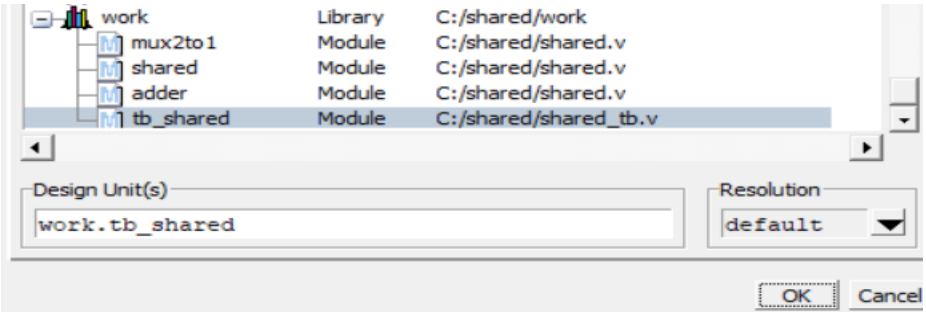
6. 编译无误后在 Library 中选择 work 库，选中 tb_shared



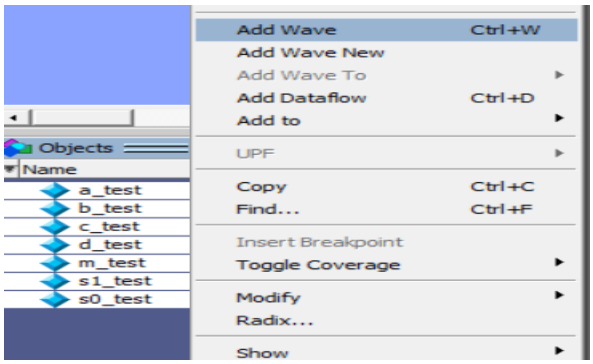
7. 菜单中选择 simulation ->start simulation



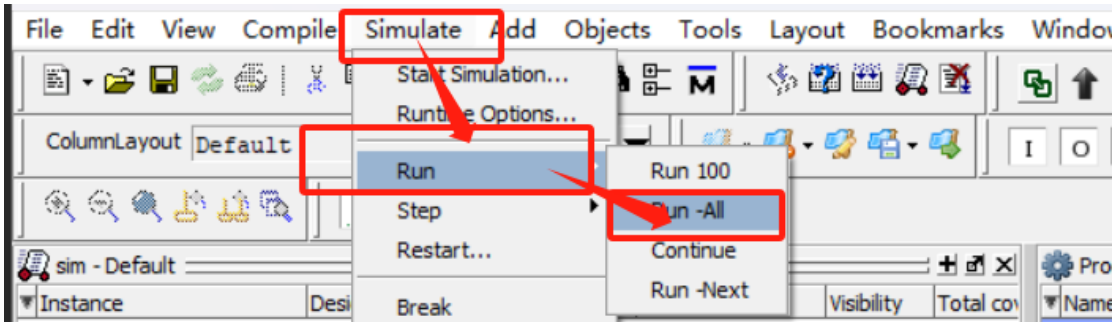
8. 再次选中 work 库中的 tb_shared，（modelsim-win32-6.3a 版本需要同时去掉优化选项的小勾），单击 ok

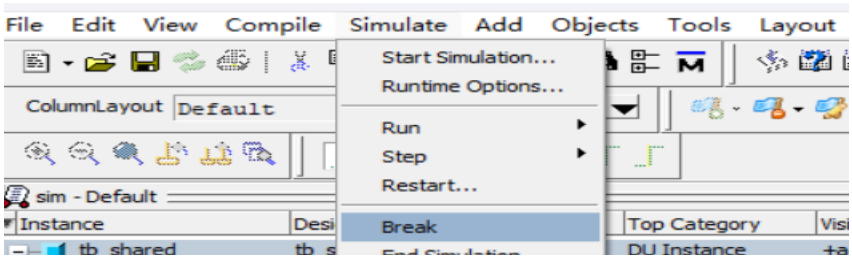


9. 在 object 选择全部信号,右击选择 Add wave

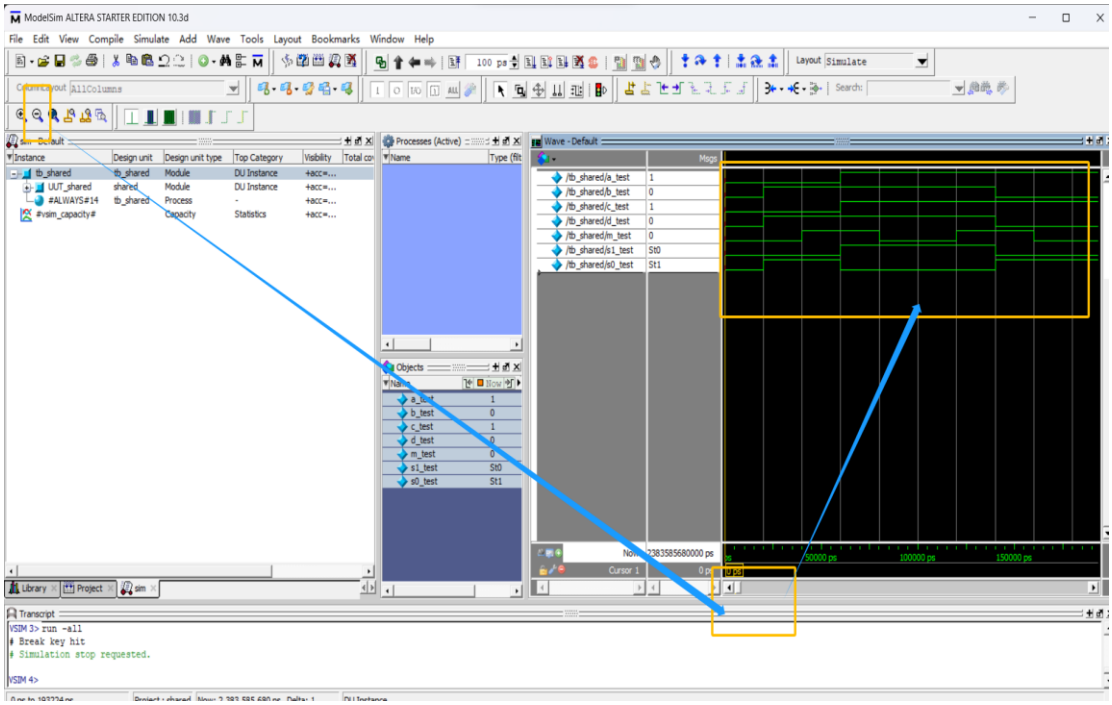


10. 菜单中选择 simulation ->start run All, 然后在菜单中选择 simulation ->start break





11. 将 wave 窗口滚动条横向拖至最左侧 0ns 处，用工具栏缩小镜图标调整视图到合适大小



程序代码实现与波形分析

Figure2.72 代码

```
module shared (a, b, c, d, m, s1, s0);
        input a, b, c, d, m;
        output s1, s0;
        wire w1, w2;

        mux2to1 U1 (a, c, m, w1);
        mux2to1 U2 (b, d, m, w2);
        adder U3 (w1, w2, s1, s0);
endmodule

module mux2to1 (x1, x2, s, f);
        input x1, x2, s;
        output f;

        assign f = (~s & x1) | (s & x2);
endmodule

module adder (a, b, s1, s0);
        input a, b;
        output s1, s0;

        assign s1 = a & b;
```

```
    assign s0 = a ^ b;
endmodule
```

Figure2.72testbench 测试代码

```
`timescale 1ns/1ps
module tb_shared;
reg a_test;
reg b_test;
reg c_test;
reg d_test;
reg m_test;
wire s1_test;
wire s0_test;

initial
m_test=0;

always #40 m_test=~m_test;

initial
begin
a_test=0;
b_test=0;
c_test=0;
d_test=0;

#20
a_test=0;
b_test=1;
c_test=0;
d_test=1;

#40
a_test=1;
b_test=1;
c_test=1;
d_test=1;

#80
a_test=1;
b_test=0;
c_test=1;
```

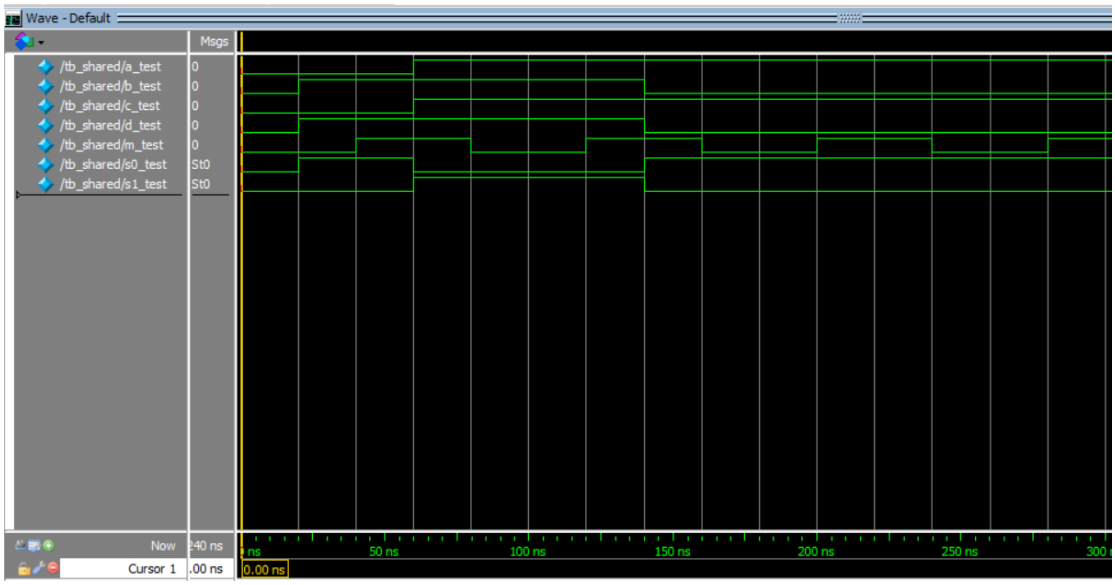
d_test=0;

end

shared UUT_shared(.a(a_test),.b(b_test),.c(c_test),.d(d_test),.m(m_test),.s1(s1_test),.s0(s0_test));

endmodule

Figure2.72 波形分析



这段代码实现了代码中一个 2-1 多路选择器加法器子电路的结合，如果多路选择器的选择输入信号 m=0，则电路实现和 S=a+b;如果 m=1，则电路实现 S=c+d。

0 到 20ns 时，由于 m 为 0，因此和为 a+b，a 和 b 在此时的值均为 0，因此 s1，s2 为 0，和为 0，输出正确。

20 至 40ns 时，m 为 0，a 为 0，b 为 1，输出为 a+b=1，s0 为 1，s1 为 0，和为 1，输出正确。

40 至 60ns 时，m 为 1，则和为 c+d，此时 c 为 0，d 为 1，s0 为 1，s1 为 0，和为 1，输出正确。

60 至 80ns 时，m 为 1，则和为 c+d，此时 c 为 1，d 为 1，因此和为 2，对应第 0 位 s0 为 0，第一位 s1 为 1，输出正确。

80 至 120ns 时，m 为 0，则和为 a+b，此时 a 为 1，b 为 1，因此和为 2，对应第 0 位 s0 为 0，第一位 s1 为 1，输出正确。

120 至 140ns 时，m 为 1，则和为 c+d，此时 c 为 1，d 为 1，因此和为 2，对应第 0 位 s0 为 0，第一位 s1 为 1，输出正确。

140ns 之后，a b c d 分别为 1 0 1 0，因此 a+b 和 c+d 均为 1，无论 m 如何跳变，和都为 1，则 s0 为 1，s1 为 0，输出正确。

Figure2.40 代码

```
module example3 (x1, x2, s, f);
    input x1, x2, s;
    output f;
    assign f = (~s & x1) | (s & x2);
endmodule
```

Figure2.40testbench 测试代码

```
`timescale 1ns/1ps

module tb_example3;
reg x1_test;
reg x2_test;
reg s_test;
wire f_test;

initial
begin
    x1_test = 0;
    x2_test = 0;
    s_test = 0;

    #20;
    x1_test = 1;
    x2_test = 1;
    s_test = 1;

    #10;
    x1_test = 1;
    x2_test = 0;
    s_test = 1;

    #30;
    x1_test = 0;
    x2_test = 1;
    s_test = 1;

    #20;
    x1_test = 0;
    x2_test = 0;
    s_test = 1;
```

```
        #20;
        x1_test = 1;
        x2_test = 1;
        s_test = 0;

        #20;
        x1_test = 1;
        x2_test = 1;
        s_test = 0;
end

example3 UUT_example3(.x1(x1_test), .x2(x2_test), .s(s_test), .f(f_test));

endmodule
```

Figure2.42 代码

```
// Behavioral specification
module example5 (x1, x2, s, f);
   input x1, x2, s;
   output f;
   reg f;

   always @(x1 or x2 or s)
   begin
      if (s == 0)
         f = x1;
      else
         f = x2;
   end

endmodule
```

Figure2.42testbench 测试代码

```
`timescale 1ns/1ps

module tb_example5;
reg x1_test;
reg x2_test;
reg s_test;
wire f_test;

initial
```

```
begin
    x1_test = 0;
    x2_test = 0;
    s_test = 0;

    #20;
    x1_test = 1;
    x2_test = 1;
    s_test = 1;

    #10;
    x1_test = 1;
    x2_test = 0;
    s_test = 1;

    #30;
    x1_test = 0;
    x2_test = 1;
    s_test = 1;

    #20;
    x1_test = 0;
    x2_test = 0;
    s_test = 1;

    #20;
    x1_test = 1;
    x2_test = 1;
    s_test = 0;

    #20;
    x1_test = 1;
    x2_test = 1;
    s_test = 0;
end

example5 UUT_example5(.x1(x1_test), .x2(x2_test), .s(s_test), .f(f_test));

endmodule
```

Figure 2.40 波形

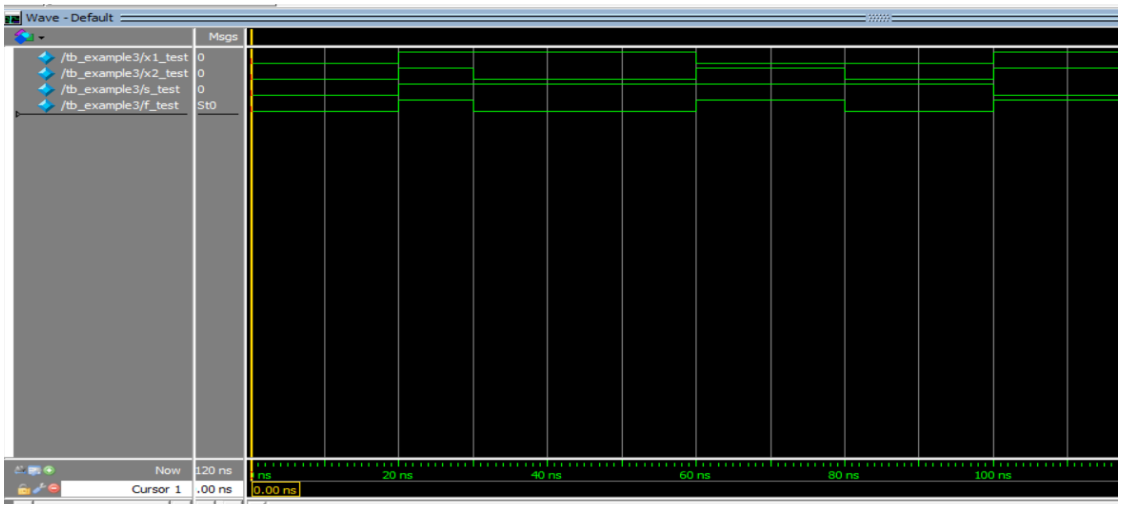
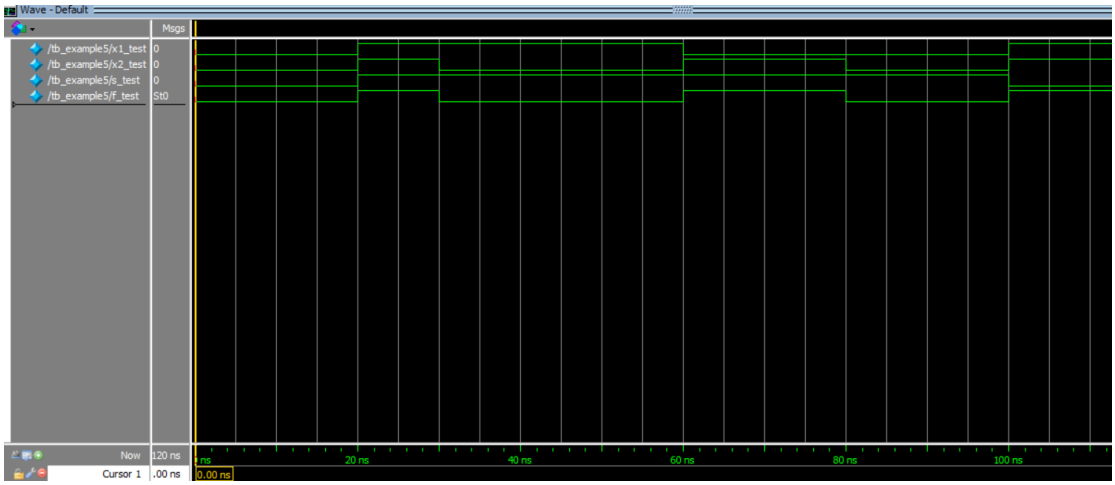


Figure 2.42 波形



波形分析

观察可发现两段代码波形相同

此代码实现了一个二选一多路选择器，s 为 0 时输出为 x1，s 为 1 时输出 x2

0 到 20ns 时，s 为 0，应该输出 x1，即 f 应该与 x1 相同，此时 x1 为 0，f 为 0，输出正确。

20 到 100ns 时，s 为 1，输出应为 x2，x2 的波形与 f 在此时间段内同步变化，输出正确。

100 到 120ns 时，s 为 0，输出应为 x1，此时 x1 为 1，f 为 1，输出正确。

Figure 2.45 代码

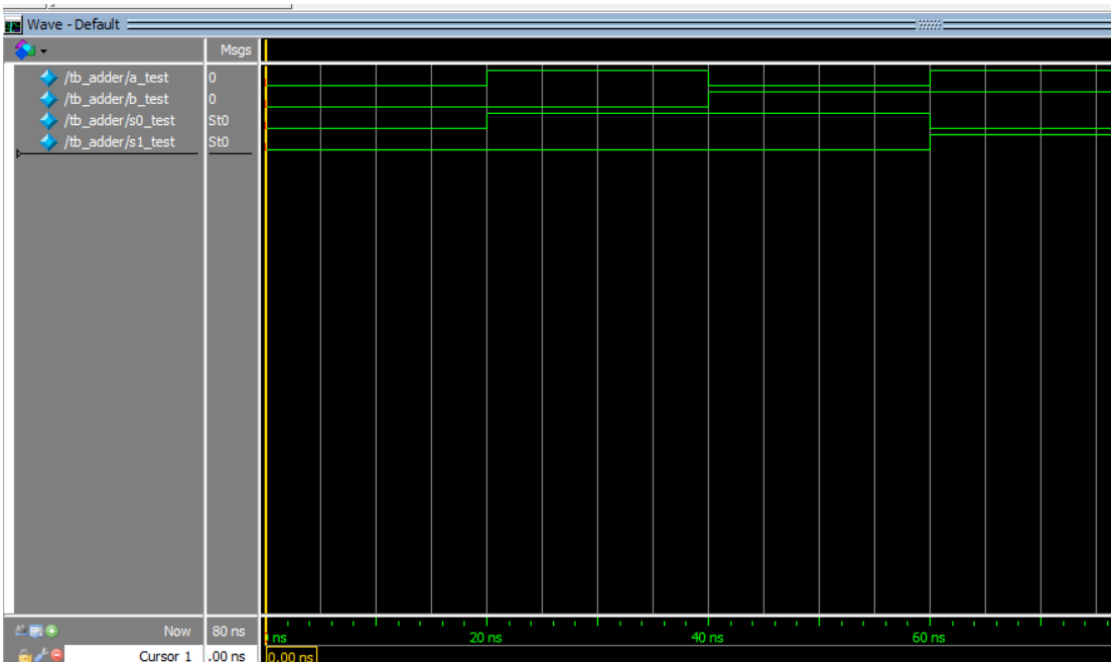
```
// An adder module
module adder (a, b, s1, s0);
   input a, b;
   output s1, s0;

   assign s1 = a & b;
   assign s0 = a ^ b;
```

```
endmodule
```

Figure2.45testbench 测试代码

```
`timescale 1ns/1ps
module tb_adder;
reg a_test;
reg b_test;
wire s0_test;
wire s1_test;

initial
begin
	a_test = 0;
	b_test = 0;

	#20;
	a_test = 1;
	b_test = 0;

	#20;
	a_test = 0;
	b_test = 1;

	#20;
	a_test = 1;
	b_test = 1;

	  #20;
	a_test = 1;
	b_test = 1;
end

adder UUT_adder(.a(a_test), .b(b_test), .s0(s0_test),.s1(s1_test));

endmodule
```

Figure2.45 波形分析



此代码实现了一个简单的加法器，输出 s0,s1 分别代表 a+b 的二进制第 0 位和第 1 位
如图，0 到 20ns 时 a 为 0，b 为 0，它们的和也应为 0，上图 s0,s1 均为 0，波形正确。
20 到 40ns 时 a 为 1，b 为 0，它们的和应 01，上图 s0 为 1，s1 为 0，波形正确。
40 到 60ns 时 a 为 0，b 为 1，它们的和应 01，上图 s0 为 1，s1 为 0，波形正确。
60 到 80ns 时 a 为 1，b 为 1，它们的和应 10，上图 s0 为 0，s1 为 1，波形正确。

实验心得与感悟

本次实验我完成了 ModelSim 环境的搭建，根据文档的指导成功运行了 shared 代码，得出了正确的波形，课后我完成了 2.40，2.42，2.45 的测试代码的编写，并运行出了相应的波形。在此过程中我有以下心得与感悟：

1. 环境搭建过程中务必细心，每一步都应按照说明仔细执行。
2. 严格按照说明步骤运行第一部代码可以为后续所编写代码的运行减少很多麻烦，在 ModelSim 上运行代码的步骤大致相同。
3. 在编写完测试代码之后可以在纸上手画出心中的目标代码，待 ModelSim 上的波形显示出来之后与之比对可以节省时间，快速发现波形错误的地方。
4. 在编写测试代码时需要注意变量的类型（代码的语句应该与变量的类型相匹配,reg 型和 wire 型需仔细判断），否则程序会报错，纠错很耗费时间。
5. 编写测试代码时考虑尽可能多的输入情况组合可以为你找到难以发现的错误。
6. 得知代码有错误后一步步仔细审查，总能发现纰漏的地方。
7. 代码编写和运行时需要沉着冷静，条理分明，逻辑清晰，因为慌乱和急躁可能会引起不必要的错误，反而会浪费时间。