# 数字电路实验一报告

Figure2.72

模块源码：

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

测试源码：

`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;

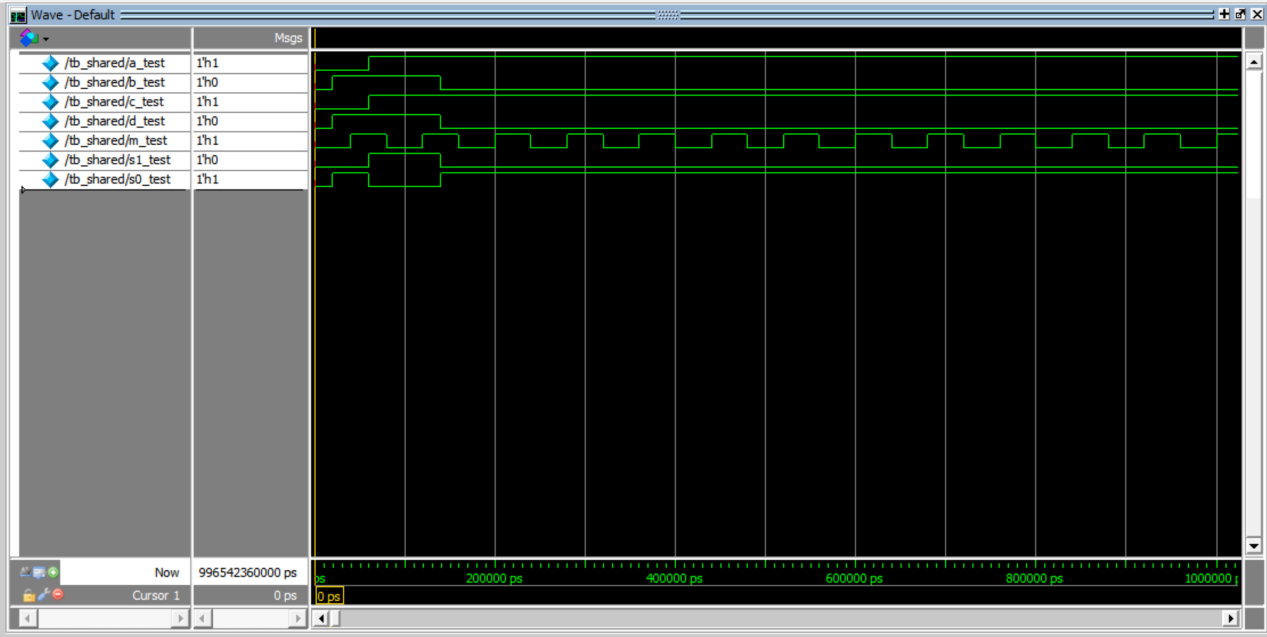
d\_test=0;

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

仿真波形：



波形分析：

Figure2.72中的电路为共享一个加法器实现两个不同和a+b和c+d的电路，结果存到s1s0中，其中s1表示十位，s0表示个位，当输入信号m=0时，电路实现S=a+b；当输入信号m=1时，电路实现S=c+d。从波形图中可以看出，前20ns内，m=0，当a=0,b=0时，s1=0,s0=0，所得结果为0；当a=0,b=1时，s1=0,s0=1，所得结果为1；20~40ns内，m=1，当c=0,d=1时，s1=0,s0=1，所得结果为1；当c=1,d=1时,s1=1,s0=0，所得结果为2。同理分析40~60ns，60~80ns内，当m=0时，所得结果均为a+b的值，当m=1时，所得结果均为c+d的值，因此可以说明模拟成功。

Figure2.40

模块源码：

module example3(x1,x2,s,f);

input x1,x2,s;

output f;

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

endmodule

测试源码：

`timescale 1ns/1ps

module tb\_text;

reg x1\_text;

reg x2\_text;

reg s\_text;

wire f\_text;

initial

s\_text=0;

always #160 s\_text=~s\_text;

initial

begin

x1\_text=0;

x2\_text=0;

#40

x1\_text=1;

x2\_text=0;

#40

x1\_text=1;

x2\_text=1;

#40

x1\_text=0;

x2\_text=1;

#40

x1\_text=0;

x2\_text=0;

#40

x1\_text=1;

x2\_text=0;

#40

x1\_text=1;

x2\_text=1;

#40

x1\_text=0;

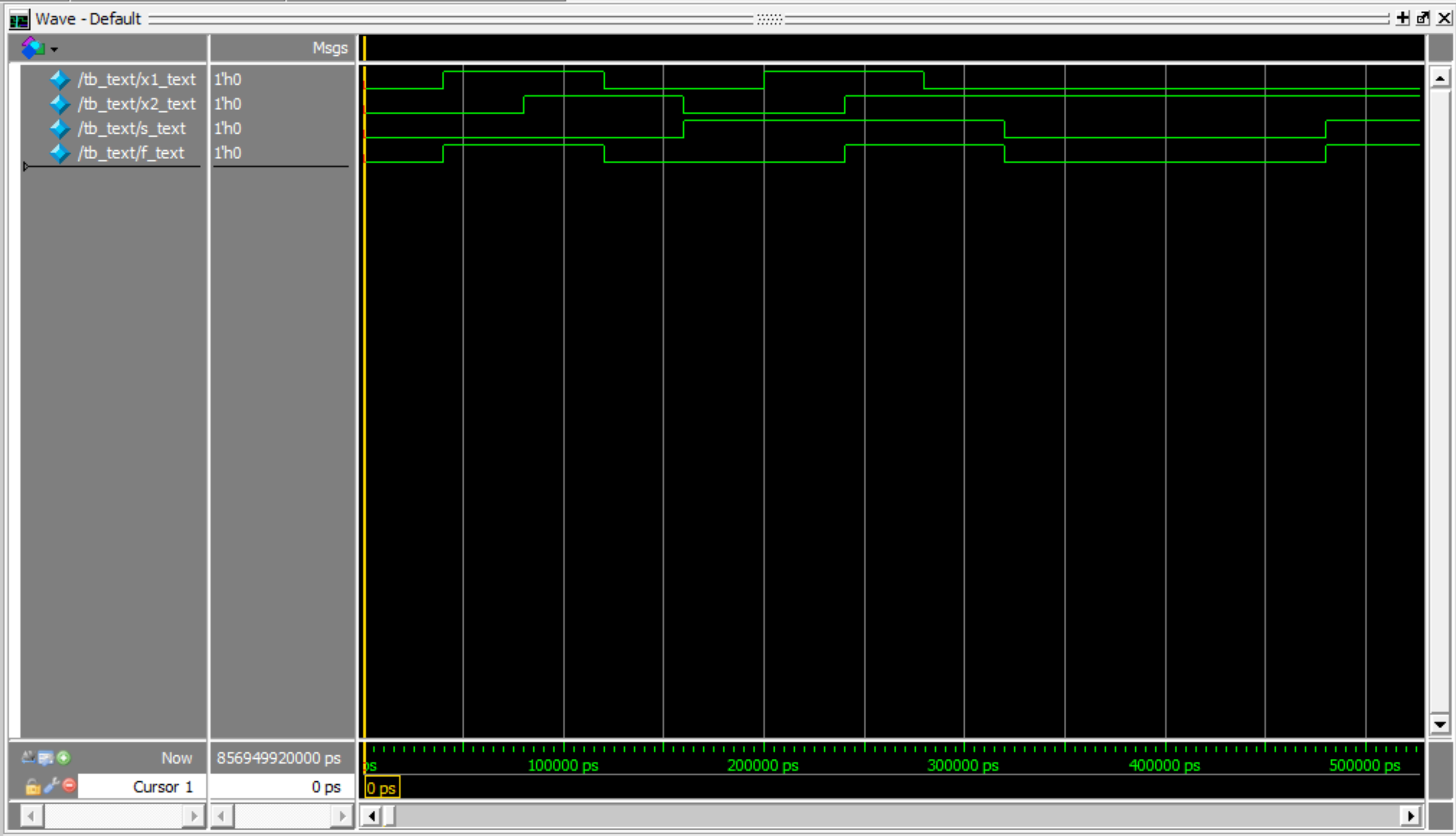
x2\_text=1;

end

example3 UUT\_example3(.x1(x1\_text),.x2(x2\_text),.s(s\_text),.f(f\_text));

endmodule

仿真波形：



波形分析：此电路为实现二选一功能的电路，当s=0时，f=x1；当s=1时，f=x2。从波形图中可以看出，前160ns内，s=0，f的波形与x1的波形一致；160~320ns内，s=1，f的波形与x2一致，类似地，往后的波形中，在s=0的时间内f的波形与x1的波形一致，在s=1的时间内f的波形与x2一致，因此可以认为模拟成功。

Figure2.42

模块源码：

module example3(x1,x2,s,f);

input x1,x2,s;

output f;

reg f;

always @(x1 or x2 or s)

if(s==0)

f=x1;

else

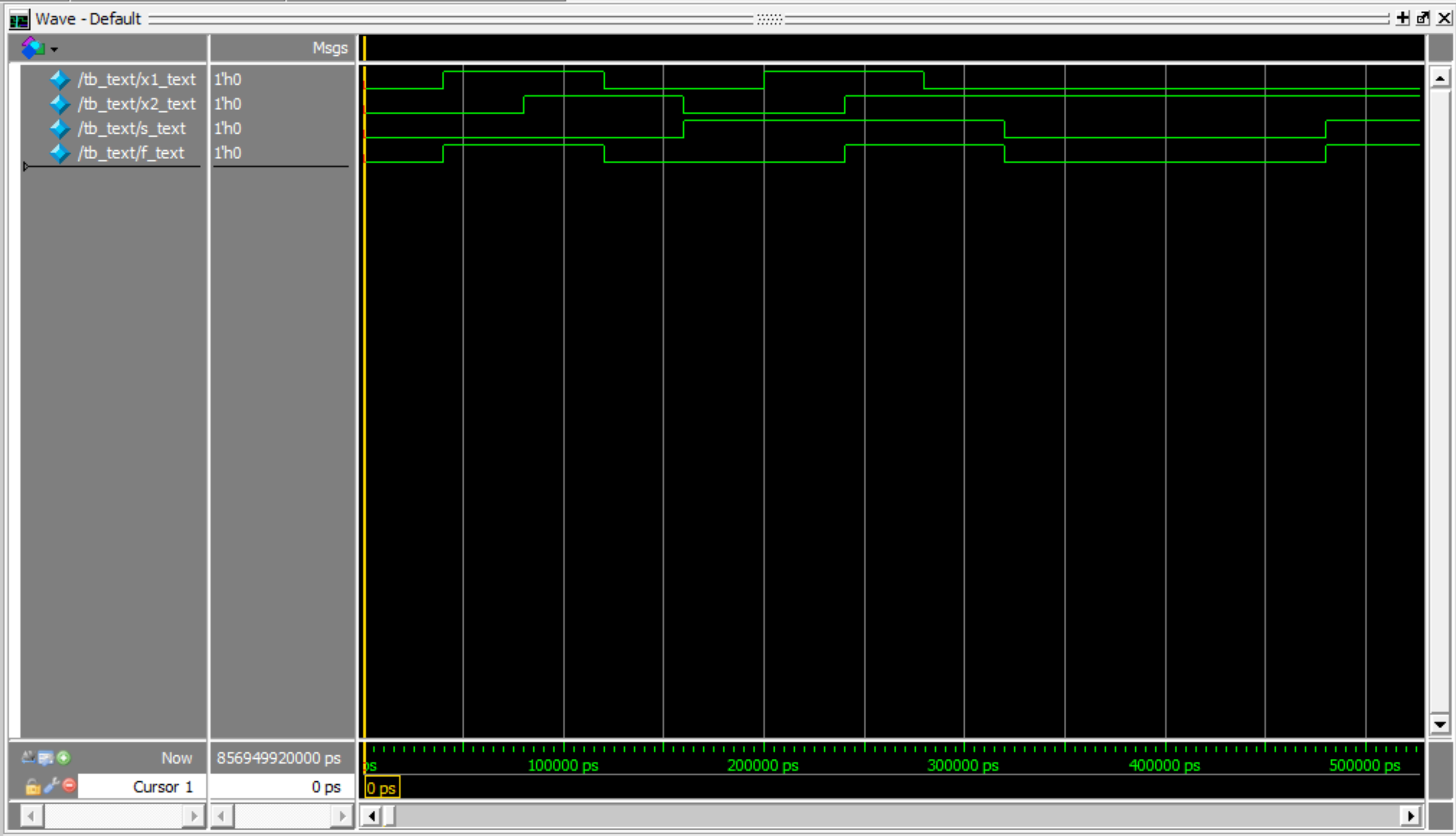
f=x2;

endmodule

测试源码：

同Figure2.40。

仿真波形：



波形分析：

可以看出此波形与Figure2.40的波形一模一样，这是因为他们的测试源码相同，模块源码一个采用连续赋值描述，另一个采用行为级描述但所实现的功能完全相同，故而他们的仿真波形一模一样，都是在s=0的时间内f的波形与x1的波形一致，在s=1的时间内f的波形与x2一致。

Figure2.45

模块源码：

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

input a,b;

output s1,s0;

assign s1=a&b;

assign s0=a^b;

endmodule

测试源码：

`timescale 1ns/1ps

module tb\_figure1;

reg a\_test;

reg b\_test;

wire s1\_test;

wire s0\_test;

initial

a\_test=0;

always #40 a\_test=~a\_test;

initial

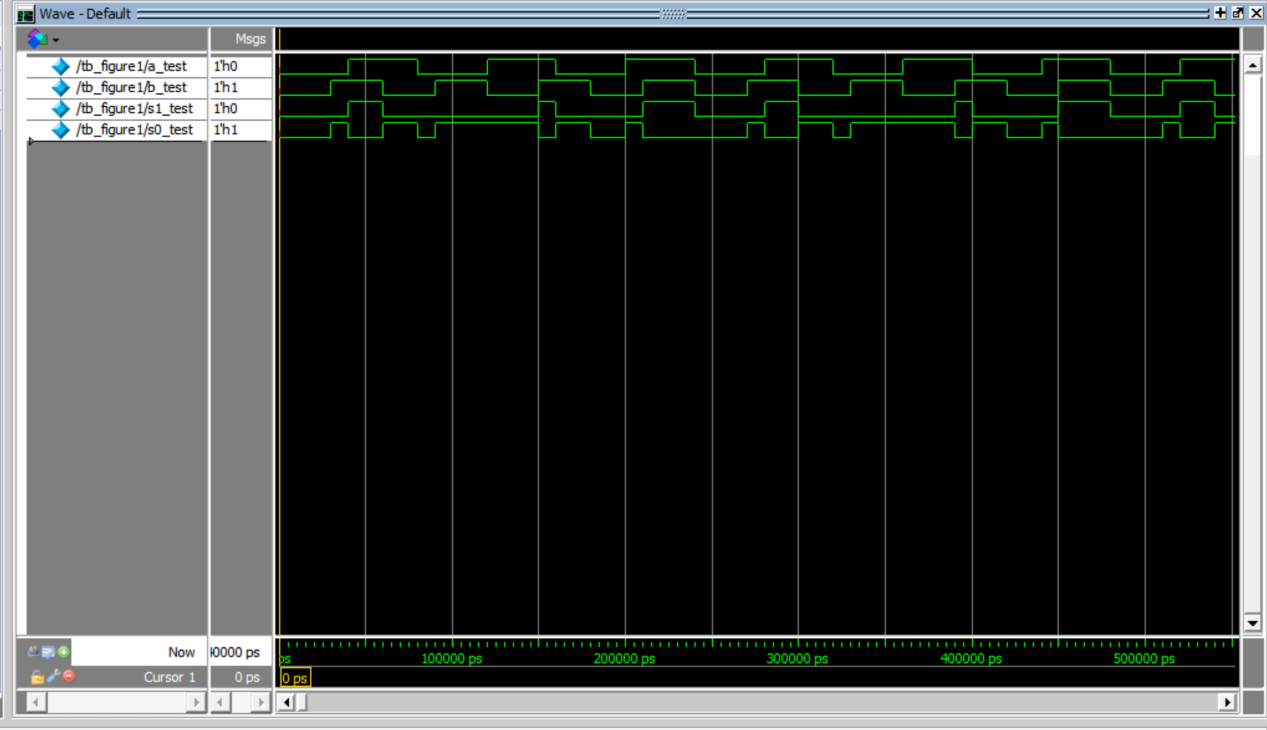
b\_test=0;

always #30 b\_test=~b\_test;

adder UUT\_adder(.a(a\_test),.b(b\_test),.s1(s1\_test),.s0(s0\_test));

endmodule

仿真波形：



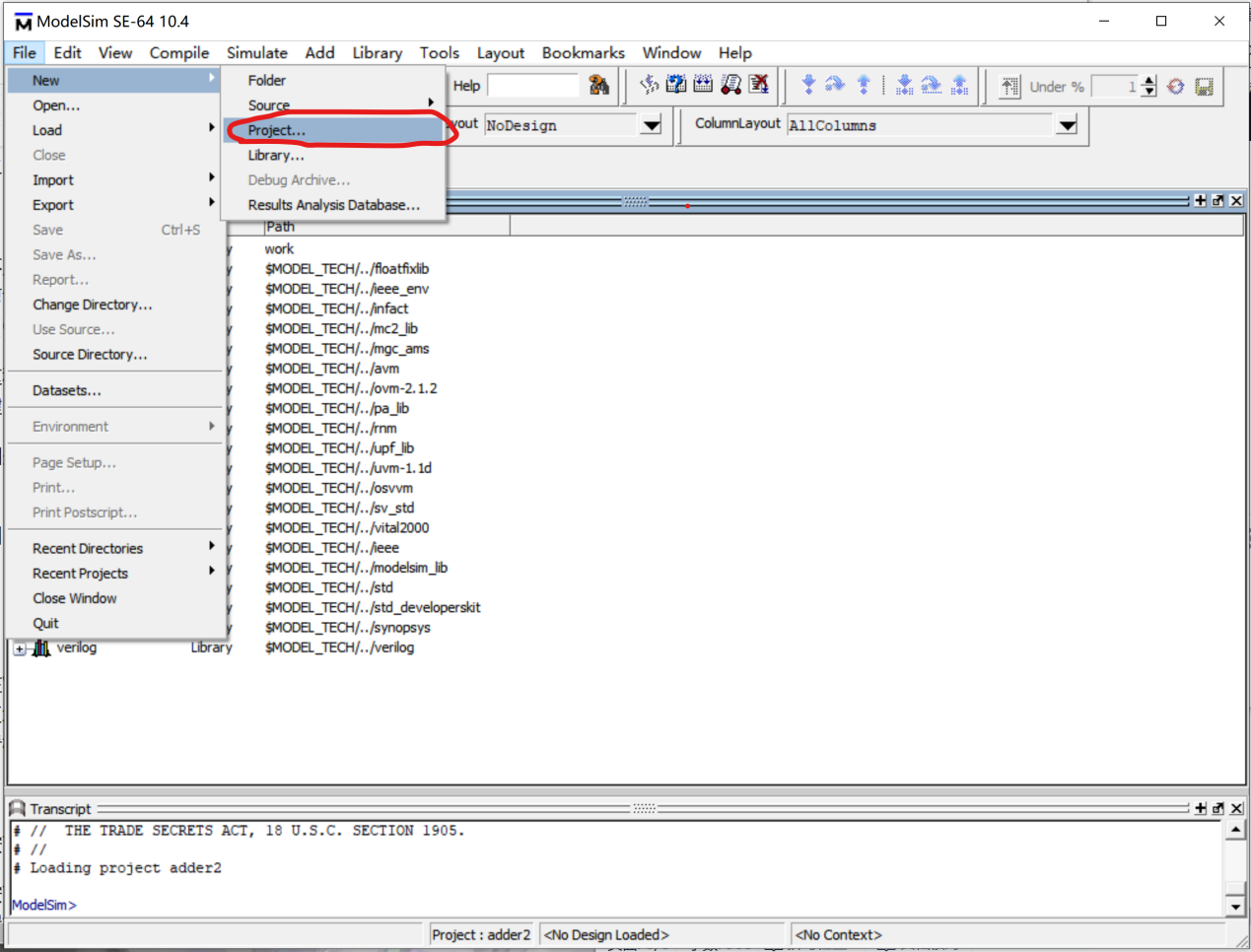
波形分析：

Figure2.12电路是实现二进制加法的一个电路，a+b的值存入s1s0中，其中，s1表示十位，s0表示个位。从波形图中可以看出，信号a每40ns变化一次，信号b每30ns变化一次。当a=0,b=0时，s1=0,s0=0;当a=0,b=1或者a=1,b=0时，s1=0,s0=1;当a=1,b=1时s1=1,s0=0，因此成功实现了两个一位二进制数加法的功能。

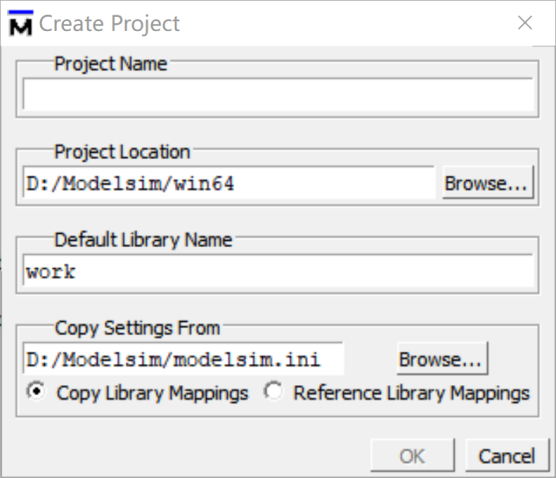
Modelsim软件使用详细步骤

一、建立工程

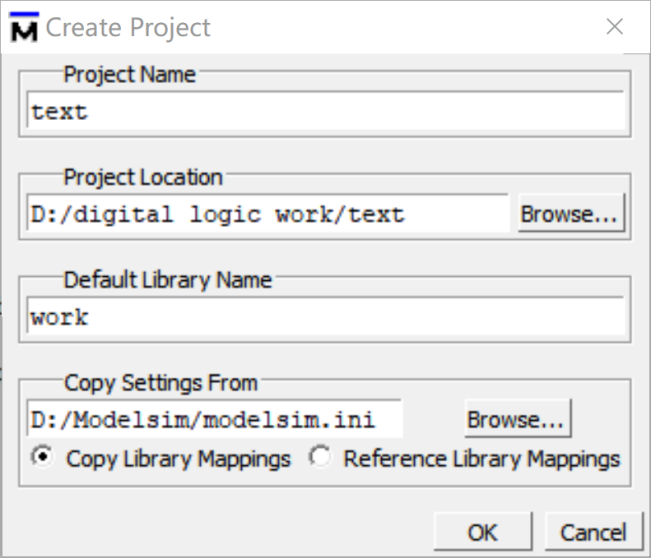
1. 双击打开Modelsim后，新建工程



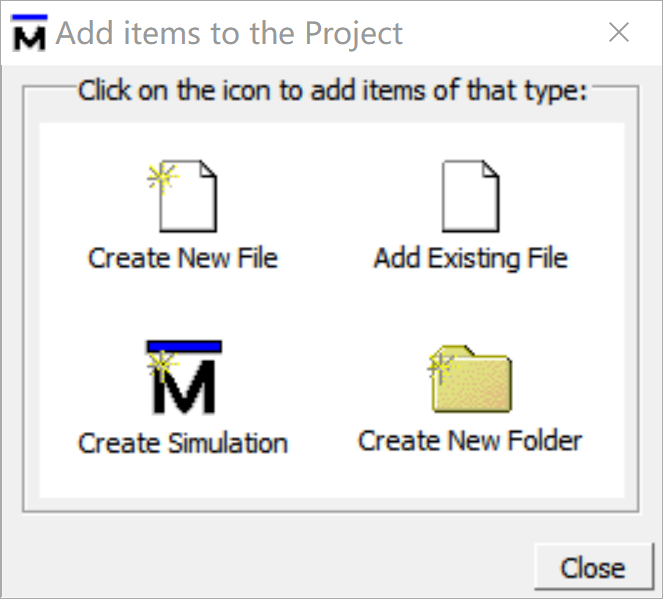
会弹出



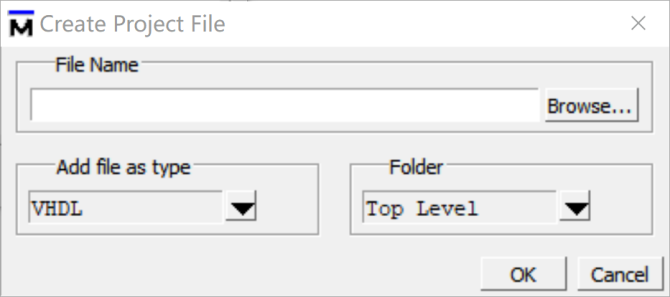
在Project Name中写入工程名，并选择工程保存位置，然后点击OK



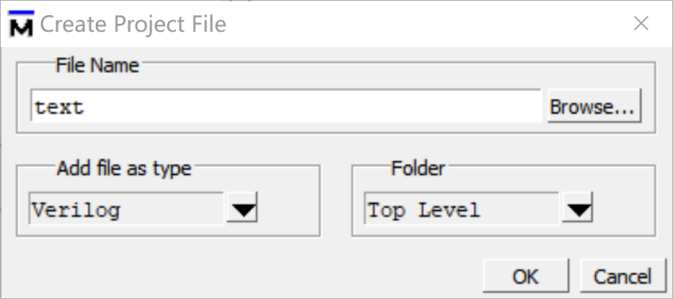
会出现



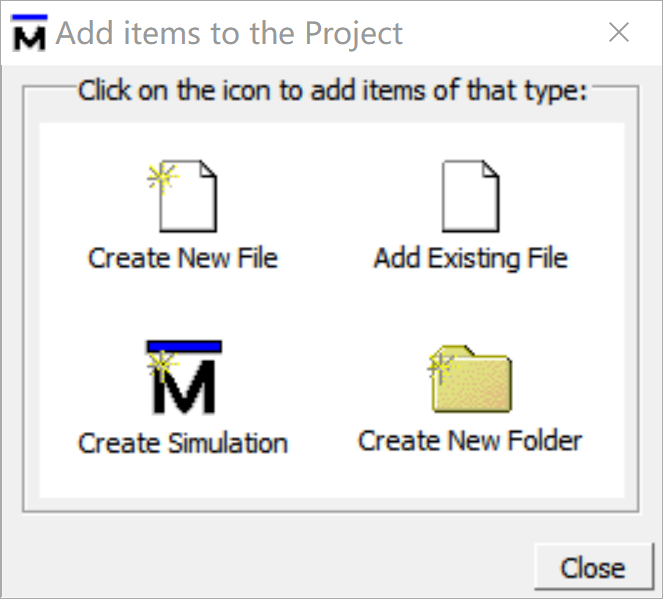
点击Create New File，或者选择在外部编辑器中编写代码后，选择Add Existing File，这里选择Create New File，会弹出



添加文件名并修改文件类型为Verilog后，点击OK



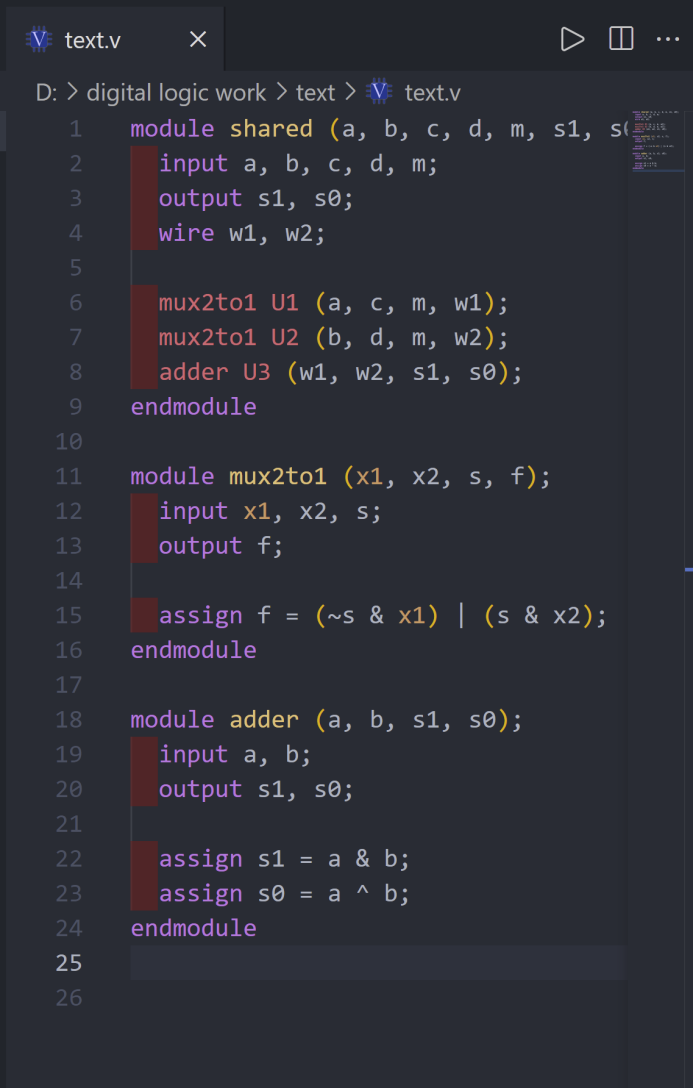
会弹出



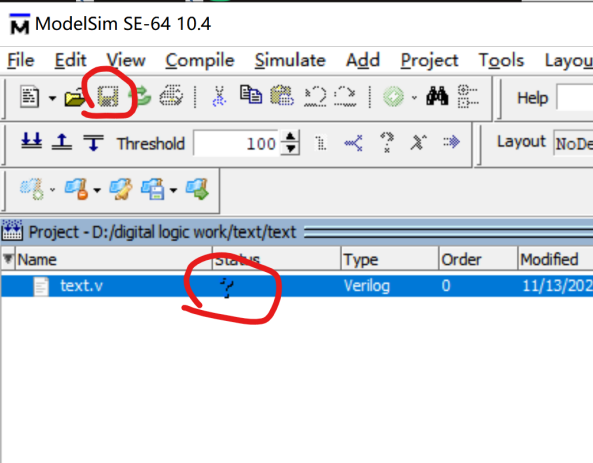
点击Close，在project中出现了一个text.v的文件，就是我们刚刚新建的那个file。

二、写代码

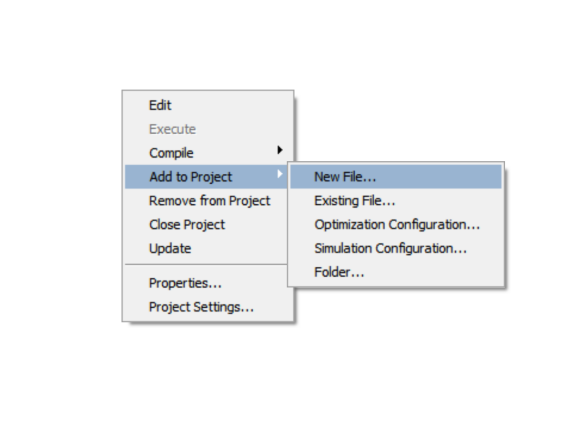
1.写主程序：双击刚刚新建的工程文件，这里使用vs code外部编译器，关于外部编译器的配置，见附表。



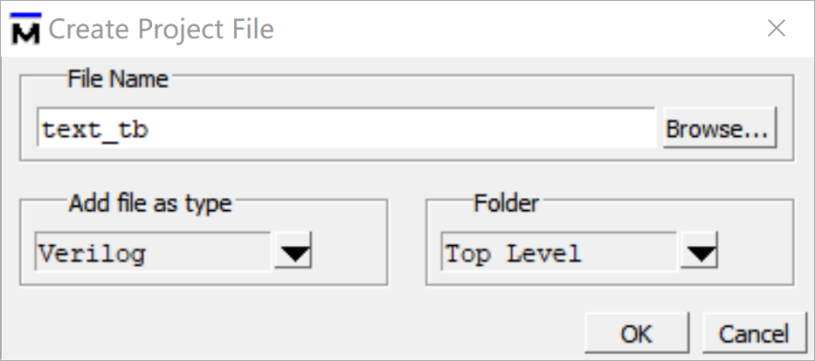
写好后不能马上编译，要先保存，否则编译无效。



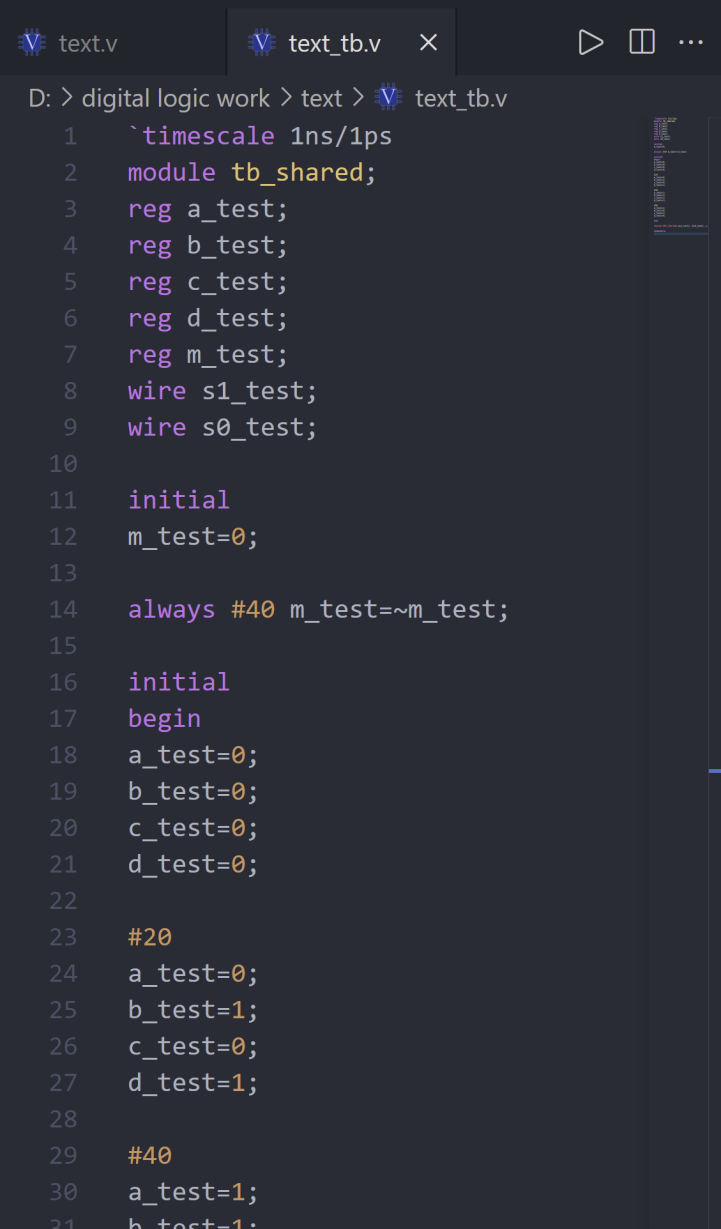
1. 写测试程序：在空白区内右键，选择Add to Project,选择New File



出现下框，写出测试程序名字test\_tb,tb是testbench的意思，注意选Verilog，点击OK

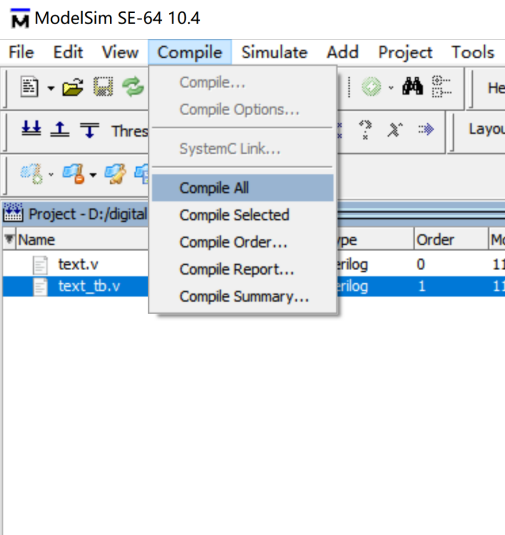


双击text\_tb.v,编写测试代码

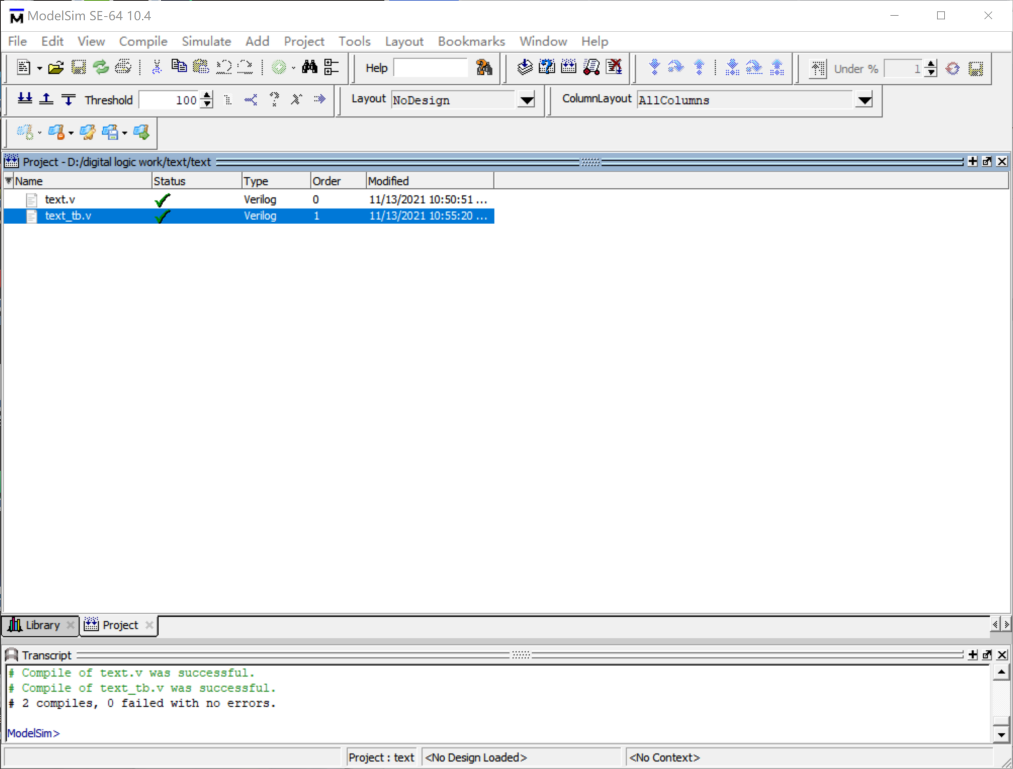


1. 编译代码

点击Compile,选择Compile All

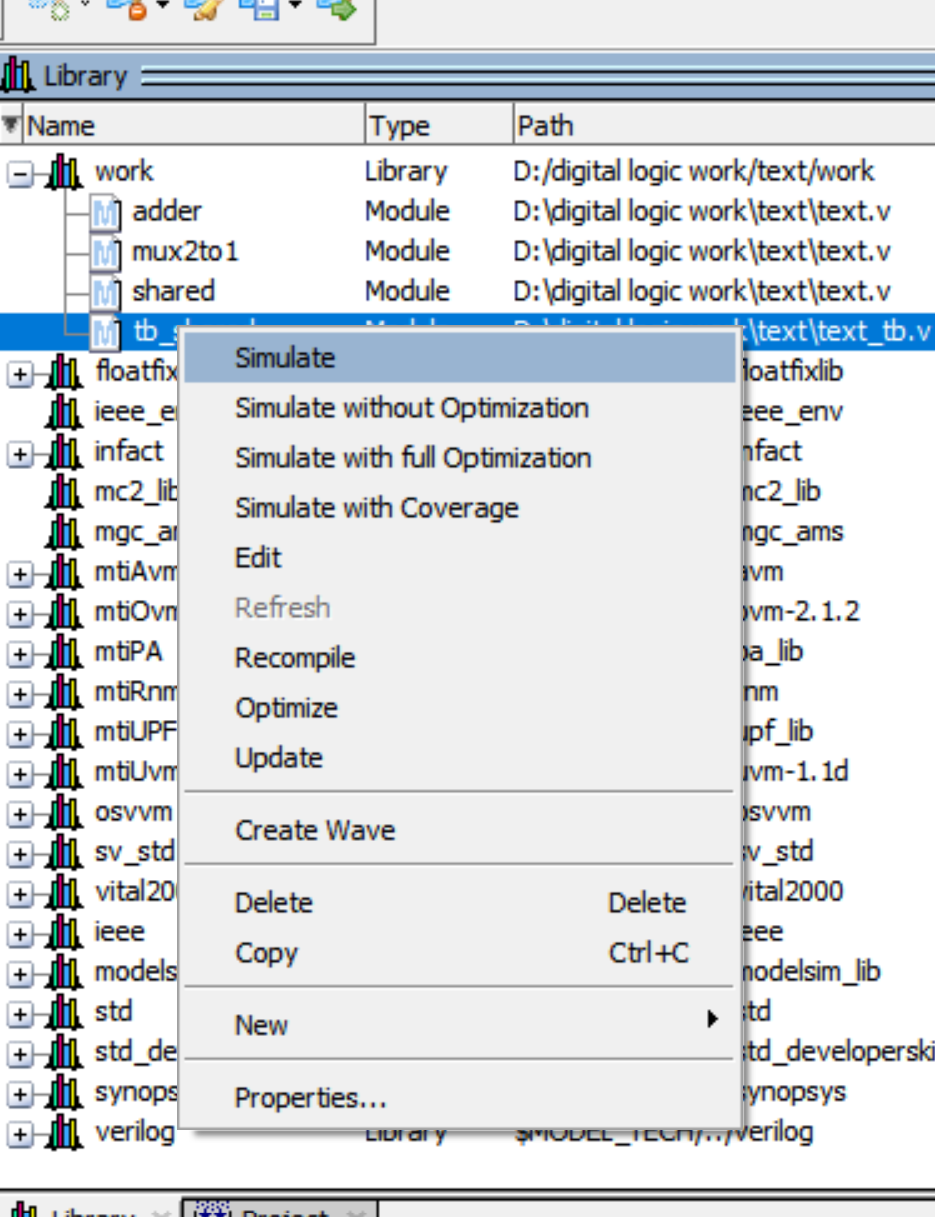


编译成功后，两文件后面的？变成了√，并且最下方的Transcript栏中出现了successful的字样，说明编译成功，否则修改文档。

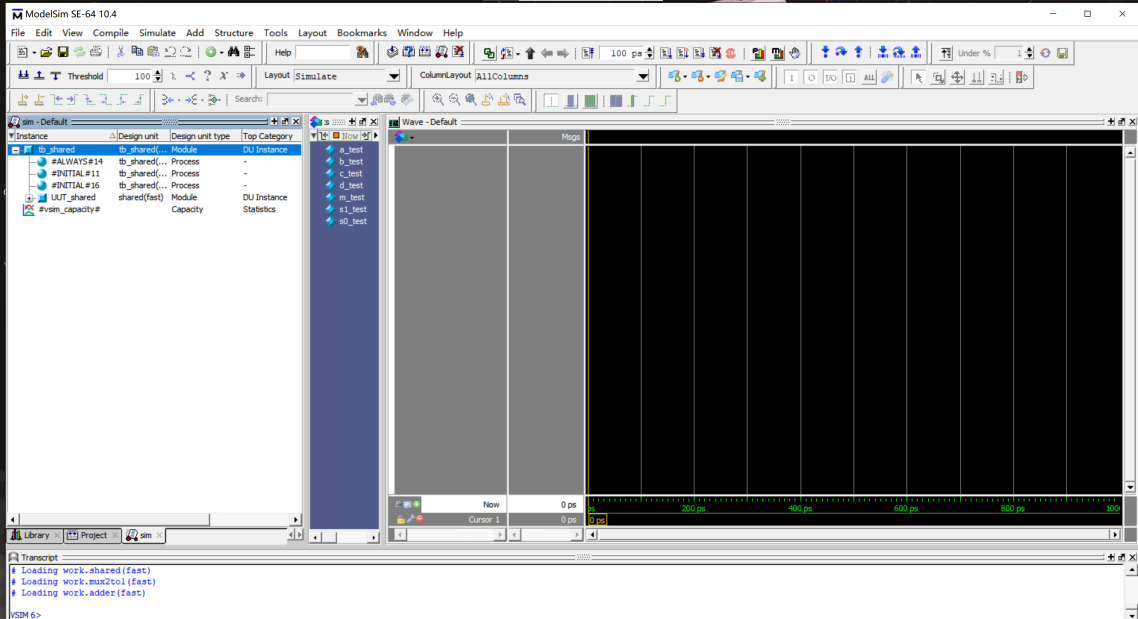


1. 仿真

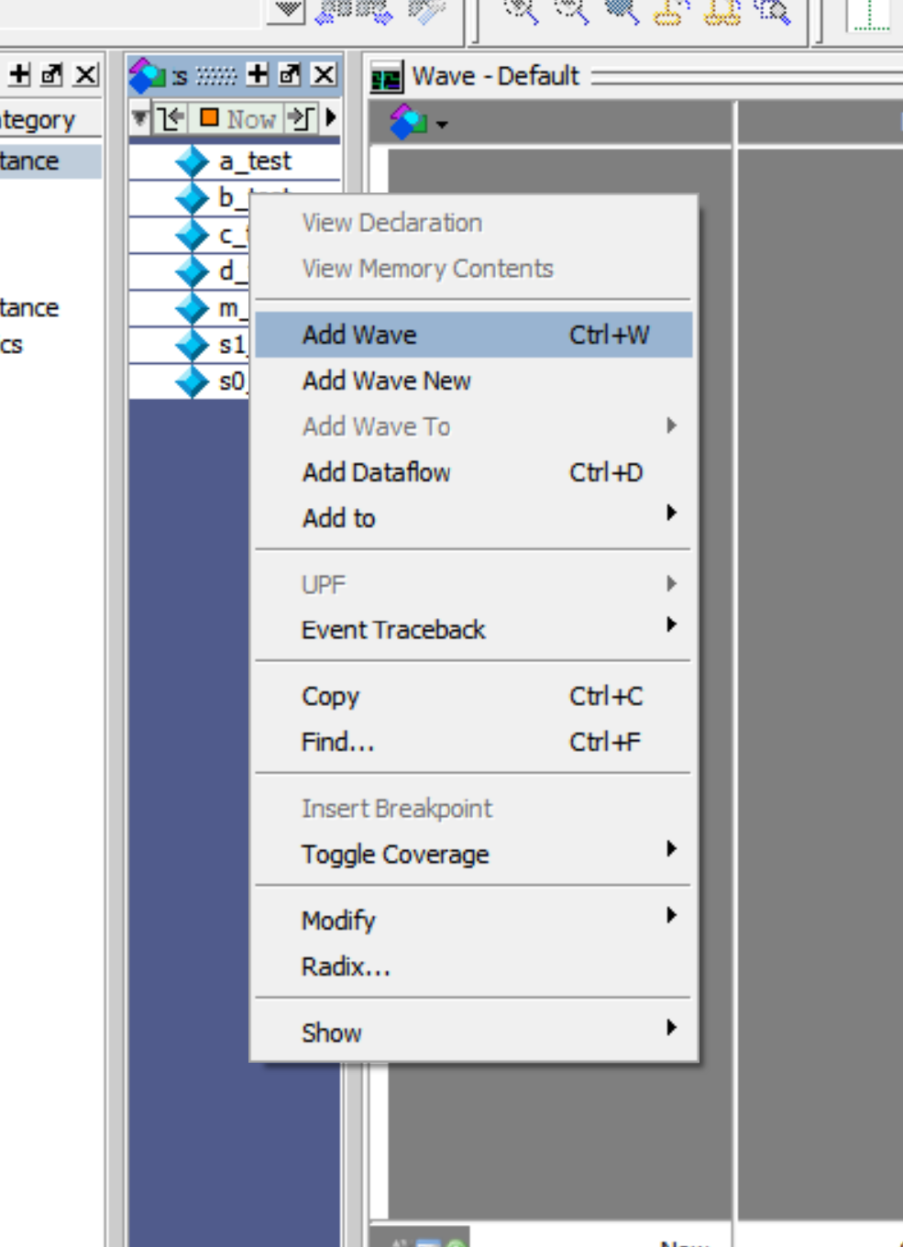
点击左下角的library按钮切换窗口，再点击work前的“+”号按钮，将其展开，看到text文件和tb\_text文件，选择后一个文件右键，选择simulate



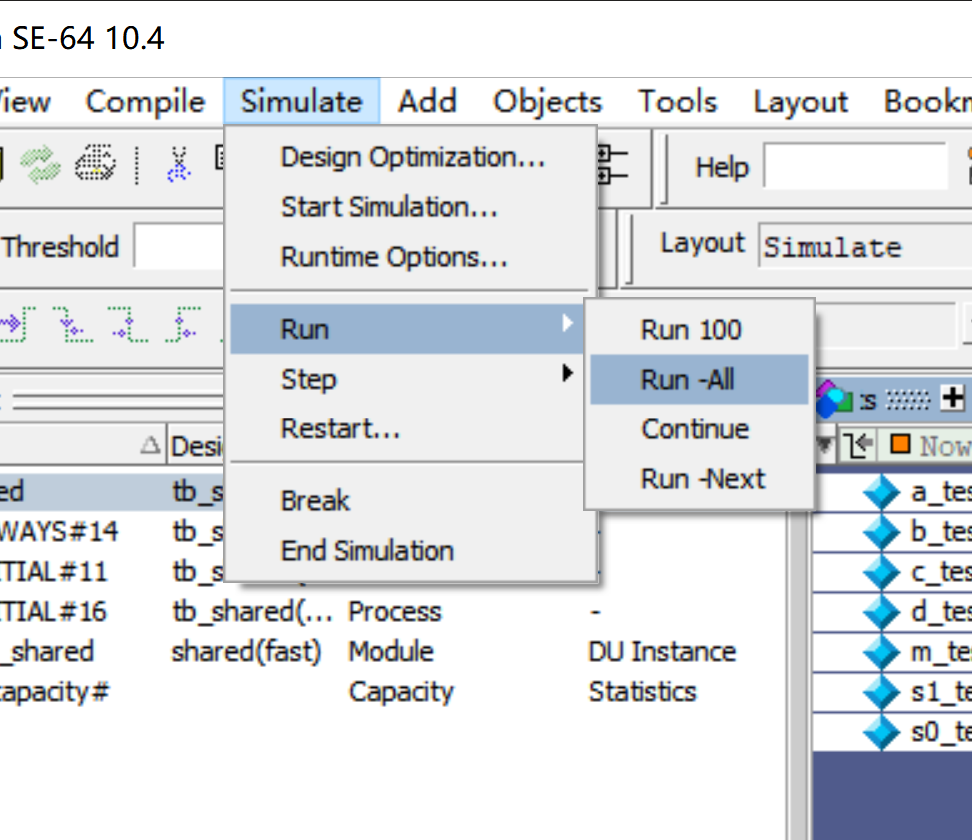
之后出现object框



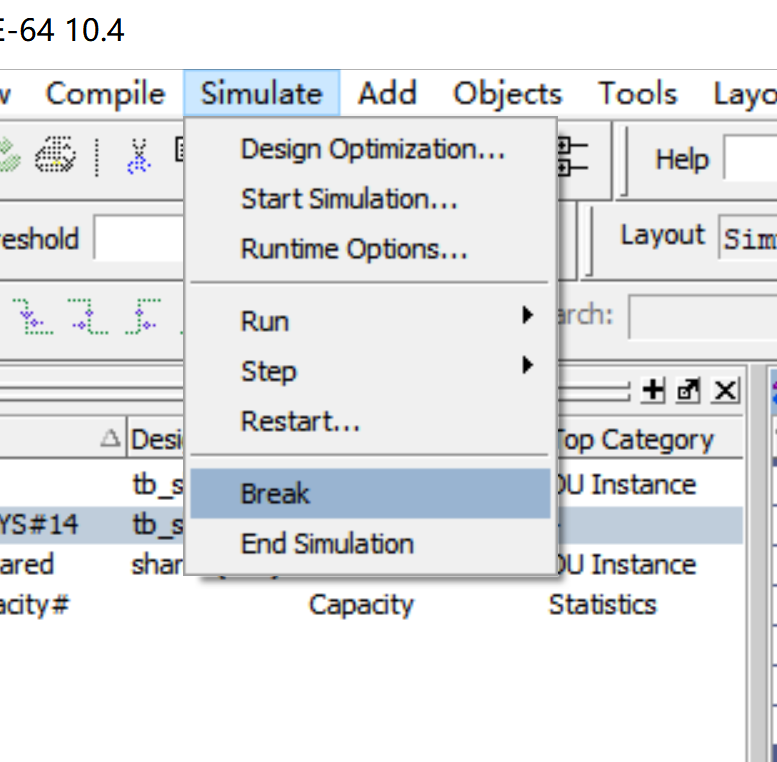
按住Ctrl键，选中所有信号，右键选择Add Wave



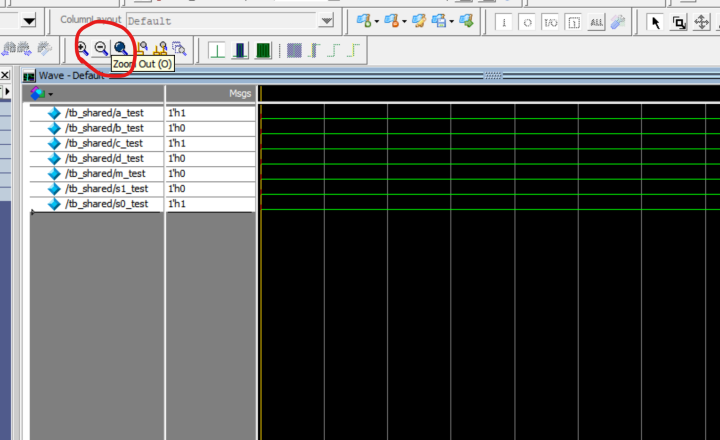
点击Simulate,选择Run,选择Run -All



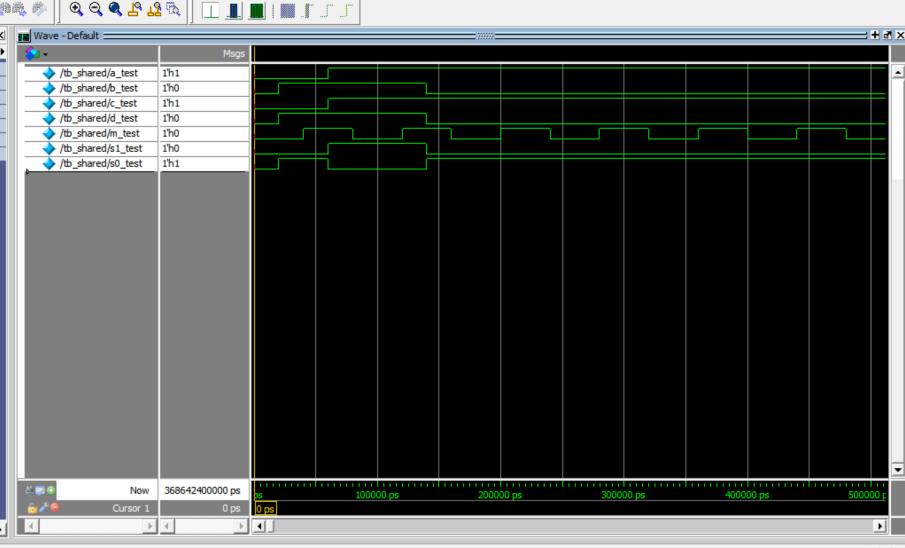
接着点击Break停止仿真



右边窗口出现波形，但看不清,点击缩小按钮



调整位置到最初时刻就能看到最终图像



本次实验收获和心得

本次实验学习到了ModelSim仿真工具的安装以及如何使用，学习了测试模块（Testbench）的结构和编写方法，体会到了仿真工具的强大功能，能够独立编写简单的代码模块和测试模块，对仿真图像进行一定的分析。在学习过程中要擅于查找资料，不能过分依赖别人的帮助，应该先自己琢磨，耐下心来都不难。

附表：

更换外部编译器

打开Modelsim，在Transcript命令行中输入如下命令：

proc external\_editor {filename linenumber} { exec "D:\\Microsoft VS Code Insiders\\Code - Insiders.exe" -g $filename:$linenumber}

其中，双引号之间为你编辑器所在的路径，注意：win路径中的\需要全部替换为\\

回车后，再键入：

set PrefSource(altEditor) external\_editor

再次回车，显示成功切换为外部编辑器，效果如图：