西安交通大學

XI'AN JIAOTONG UNIVERSITY

微电子学实验室

实验教程

VerilogHDL/VHDL 仿真器 ModelSim 实验

2006-7

目 录

1. 前言

ModelSim 仿真软件是由 MentorGraphic 公司的子公司 Model 技术公司开发的工业界上最为通用的仿真器之一，它可以用于 Verilog 仿真，VHDL 仿真或者两者的混合仿真。

ModelSim 仿真软件产品的类型很多，我们在这里要介绍的是 ModelSim/SE，它是 ModleSim 主要的版本，功能最为强大，包含了 ModelSim/PLUS 的所有功能及其附加功能。ModelSim/SE 随着时间的推移不断地推出新的版本，我们要介绍的是 Mentor Graphis 公司于 2005 年 6 月推出的 ModelSim/Se 6.1 版本的使用。

首先要准备本实验教程的实验数据，请每个同学将目录/cad/Labs/mentor/modelsim/下的的 modelsimLab 文件夹(内部包含 lab1、lab2、lab3、lab4 四个文件夹)复制到自己的 home 目录下。执行以下命令：

```
unix%cd /cad/Labs/mentor/modelsim
unix%cp -r modelsimLab ~
unix%cd ~/modelsimLab/lab1
```

输入 fpga.setup 命令，然后输入 vsim 命令，便可以得到图 1.1 所示的 ModelSim 图形用户界面。fpga.setup 命令是为了设置环境变量，在每一个要运行 modelsim 的 Terminal 中只要执行一次。

```
unix%fpga.setup
unix%vsim &
```

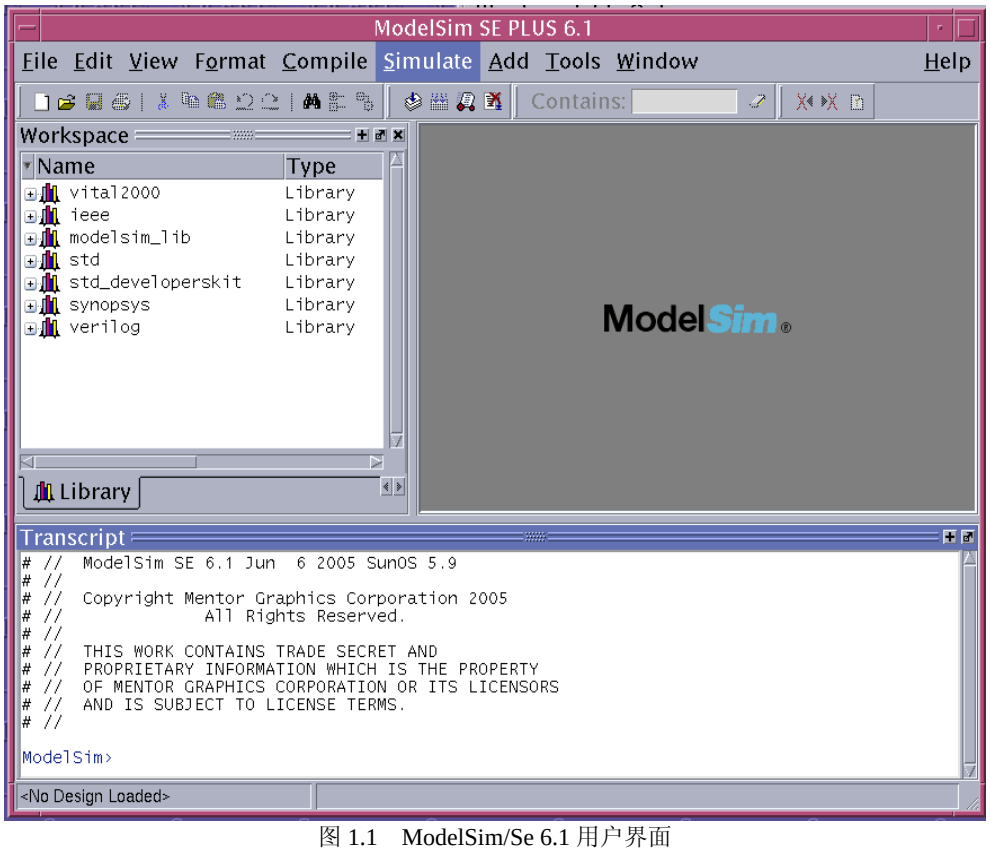

图 1.1 ModelSim/Se 6.1 用户界面

2. 基本的使用步骤

ModelSim 有三种实现方法。第一种是交互式的命令行，这种操作方法没有用户界面，唯一的界面是控制台的命令行。第二种是用户界面(UI)，它能够接收菜单输入和命令行输入。第三种是批处理模式，是用 DOS 或 UNIX 命令行运行批处理文件。我们在这里主要讨论第二种——用户界面的方式。

2.1 建立 ModelSim 库（物理库）

从主菜单里面：File->New->Library 点击 Library 得到 Create a New Library 对话框。选择 a new library and a logical mapping to it ，在 Library Name 中输入 work，相应的在 Library Physical Name 中也会出现物理名 work。然后点击 OK 确定。

此时在工作空间 workspace 的 Library 栏内会出现一个名为 work 的库。

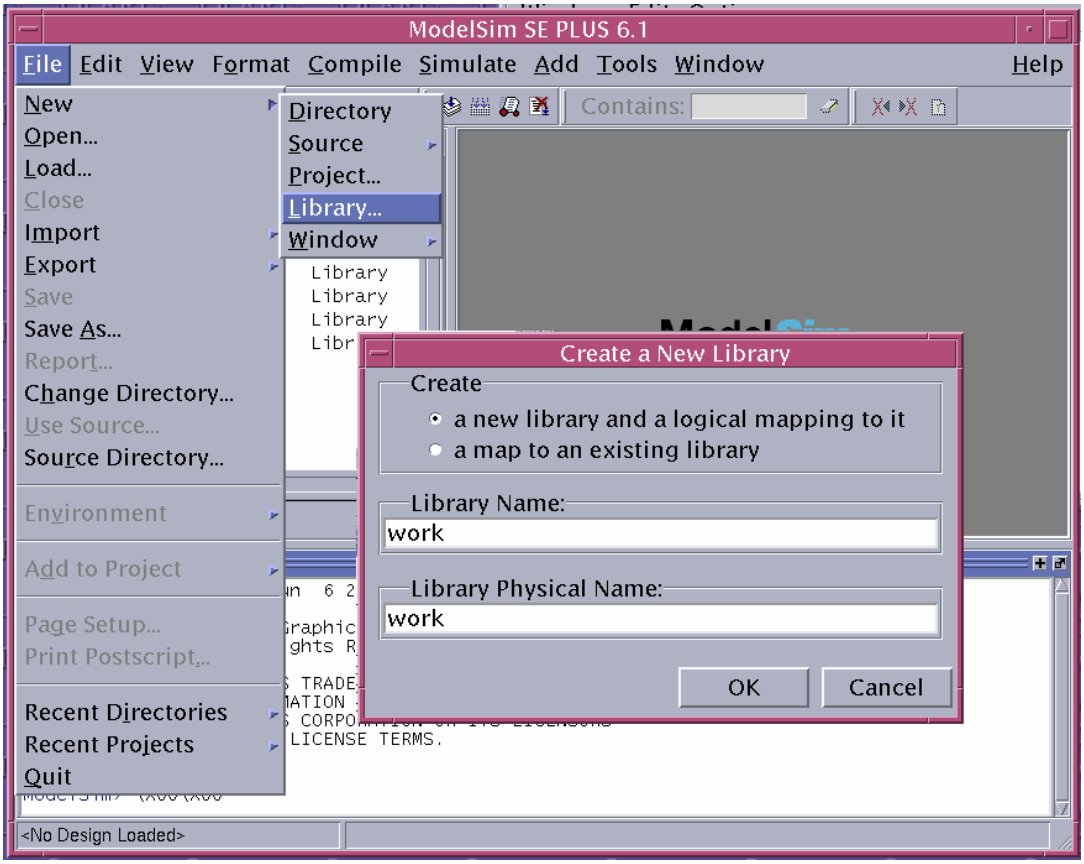

图 2.1 建立库

2.2 建立工程管理

从主菜单里面：File->New->Project 点击 Project 得到 Create Project 窗口，在 Project Name 栏输入工程名 project1，在 Project Location 栏输入新建工程所处的位置（默认为用户当前所

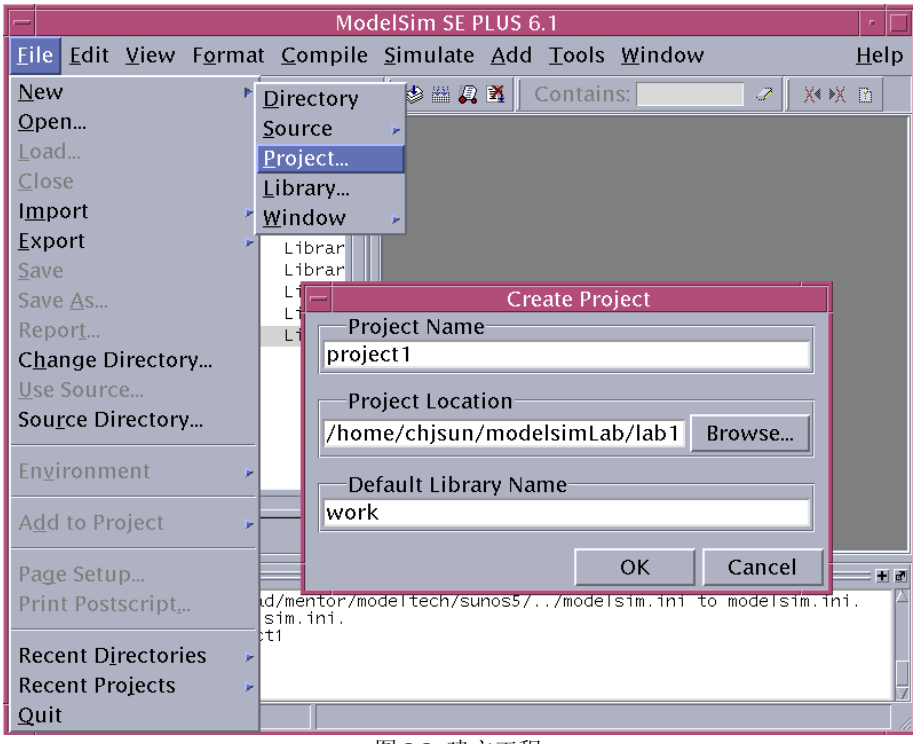

图 2.2 建立工程

在的工作目录下），在 Default Library Name 栏中输入所使用的库名（默认为 work 库）。输入完毕以后点击 OK 确认。如果用户第一次使用的话，ModelSim 会提示你选择 Use Default Ini 或者 Use Current Ini，选择 Use Default Ini。

此时在工作空间 workspace 内会出现一个名为 Project 的栏，如图 2.3 所示。同时出现了一个 Add items to the Project 的对话框，选择 Add Existing File 后又会弹出一个 Add file to Project 的对话框。

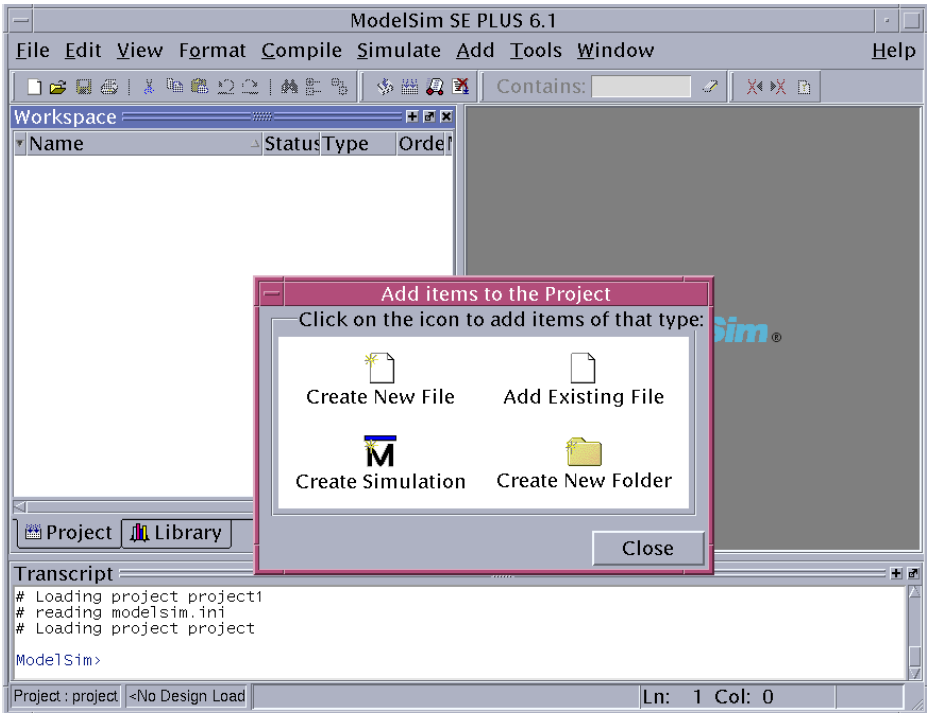

图 2.3 添加文件

点击浏览 Browse 弹出 Select files to add to project 对话框。选择文件 lab1 下的 nand2.v 文件后，按住 Ctrl 键继续选择 tb_nand2.v 文件，此时点击 Open 确定。这是一个很简单的二输入与非门的 verilog 模型 nand2.v 和二输入与非门的测试激励 tb_nand2.v。代码很简单，只是用来演示如何使用 modelsim。

进一步点击 Add file to Project 对话框中的 OK 确定。同时关闭 Add items to the project

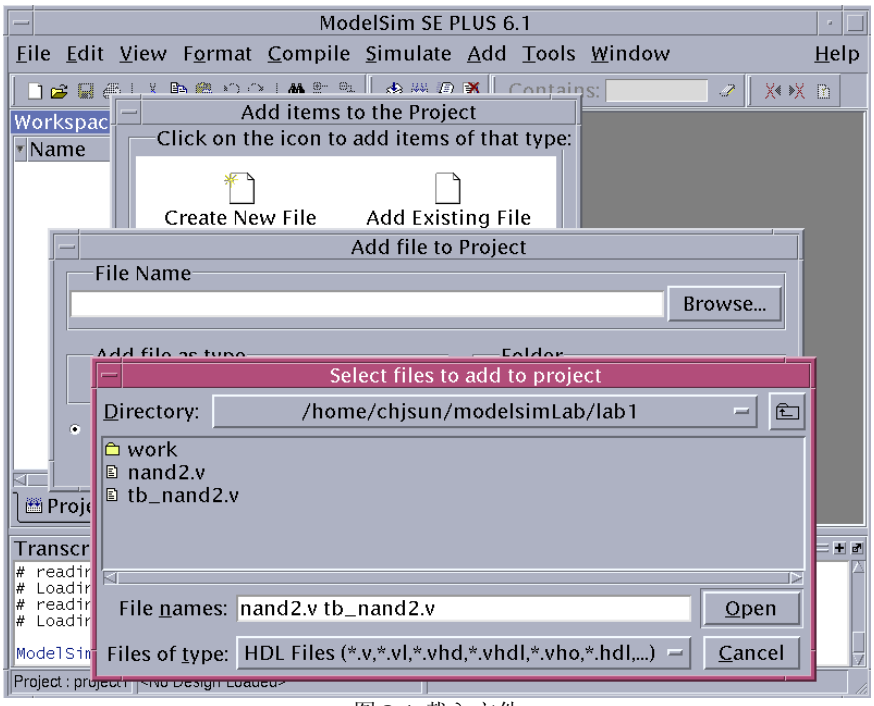

图 2.4 载入文件

对话框。此时在 Project 栏中出现了刚才被选入的 nand2.v、tb_nand2.v 两个文件。同时它们的状态 Status 被显示为？，代表了这两个文件现在还没有被编译。

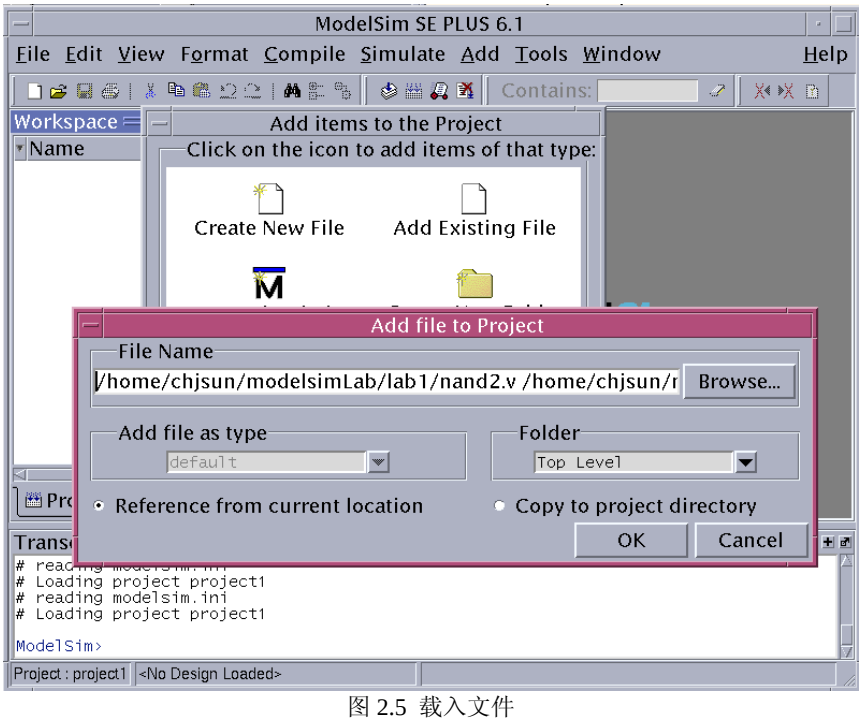

图 2.5 载入文件

2.3 编译源代码

选中 nand2.v 文件以后点击右键在下拉菜单中点击 Compile->Compile All，此时两个文件都会被编译。也可以分别选中两个文件右键选择 Compile Selected 对两个文件分别编译。

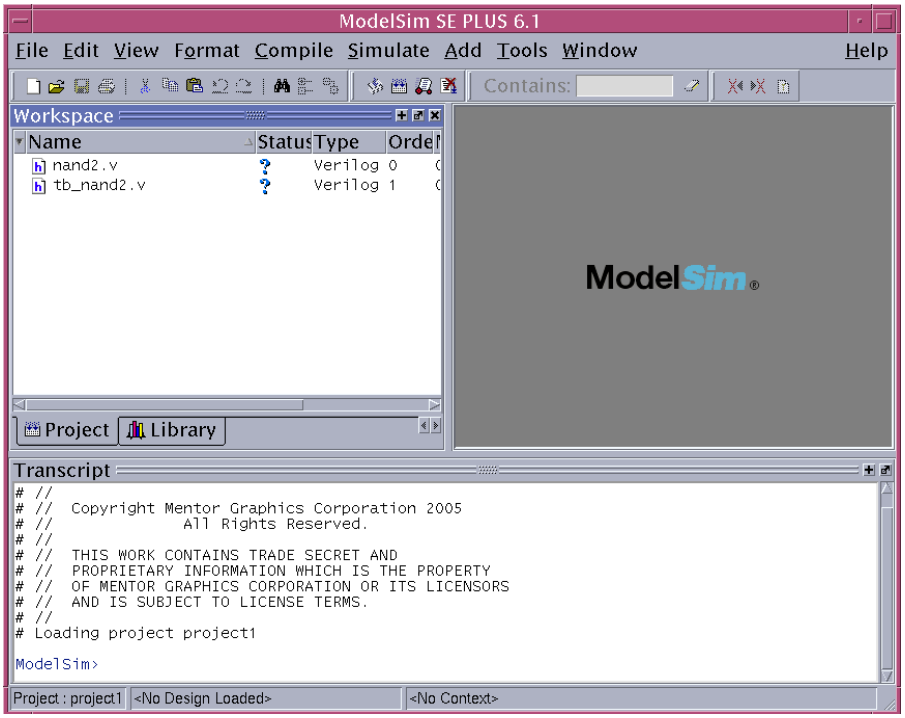

图 2.6 编译前

编译完以后用户界面如图 2.7 所示：此时状态 Status 显示为绿色的对号（√)表示编译已通过。

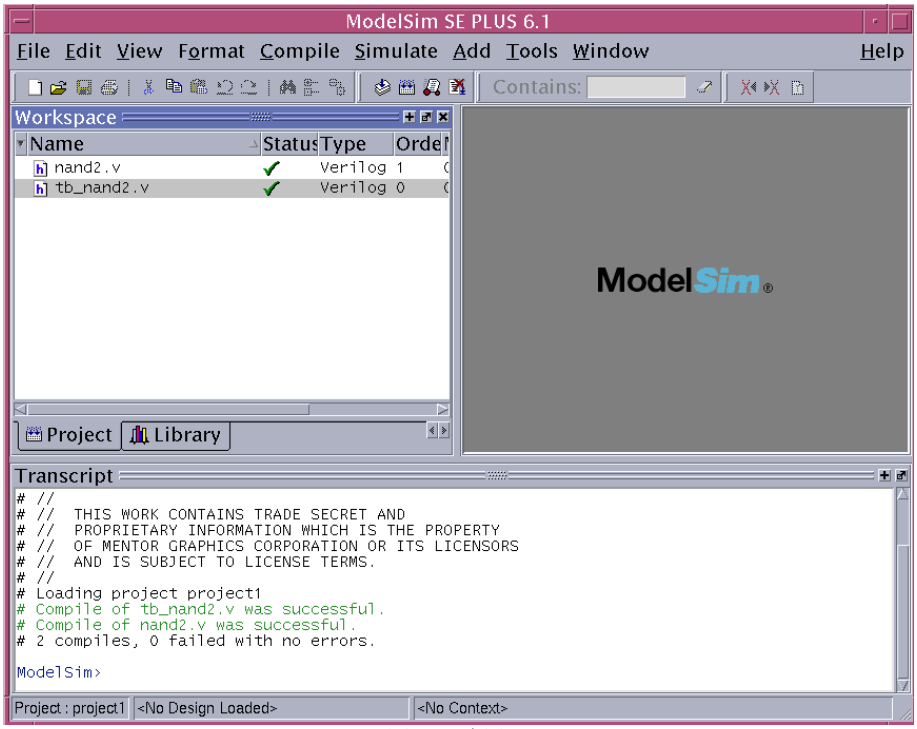

图 2.7 编译后

2.4 启动仿真器

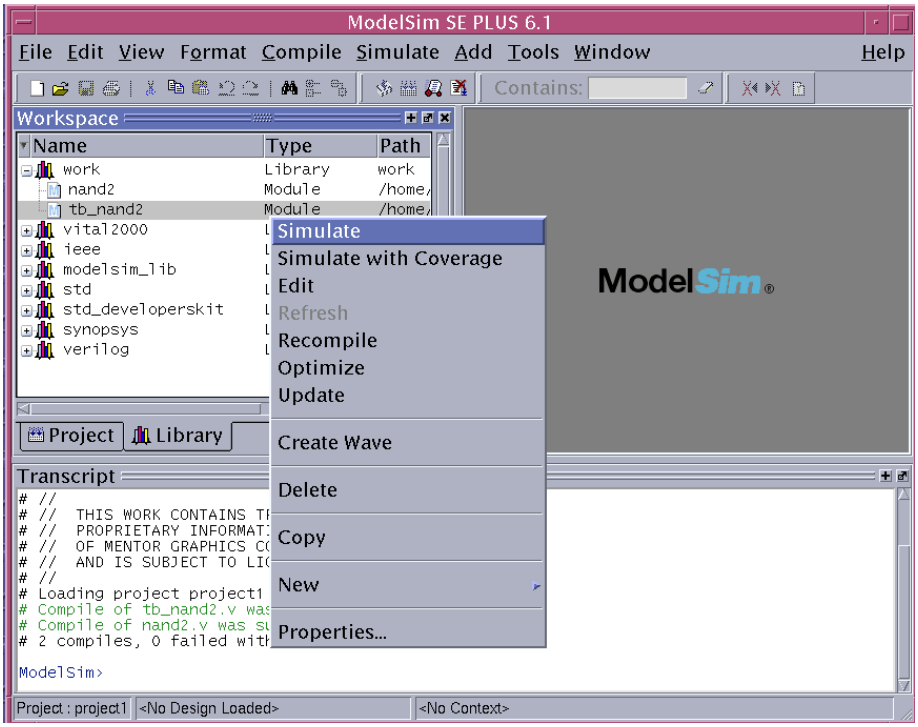

图 2.8 启动仿真

编译通过以后，点击 Library 栏，展开 work 库。发现此时在 work 库内部出现了 nand2 与 tb_nand2 两个模块(图标表示是 Verilog 的 module)。选择测试模块 tb_nand2（注意：必须选择顶层的测试模块）点击右键得到下拉菜单，选择 Simulate，就可以加载要仿真的设计

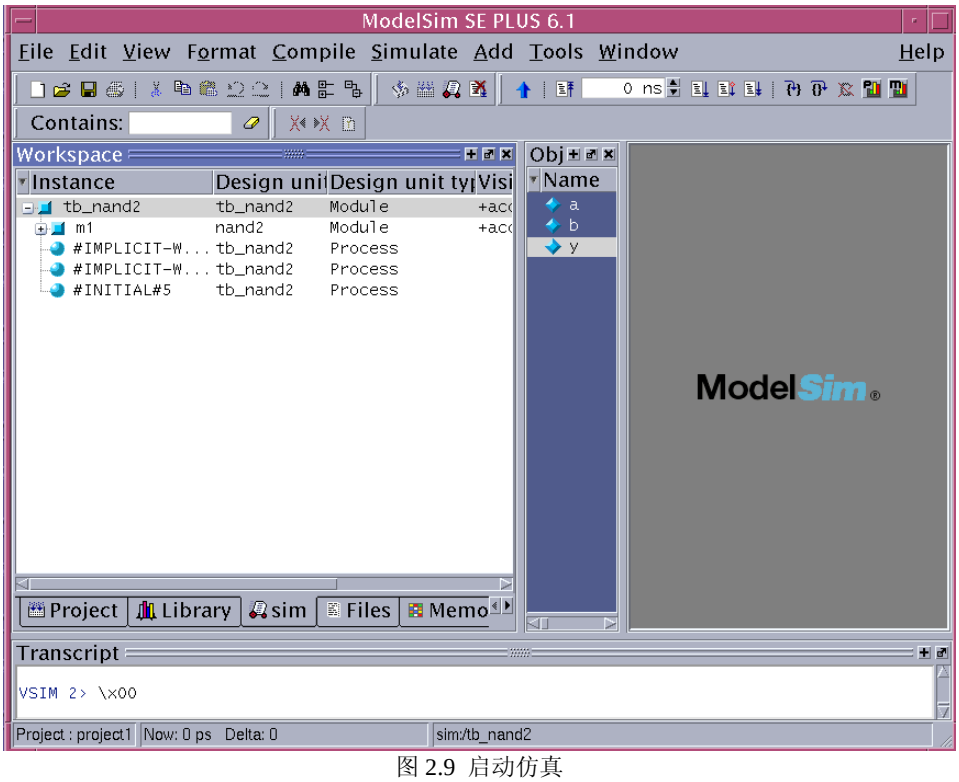

图 2.9 启动仿真

了。点击 Simulate 后在 Workspace 工作空间内出现了 3 个新的栏，它们分别是 sim、Files、Memories。如图 2.9 所示

选择 sim 栏，右键点击 tb_nand2，出现下拉菜单，选择 Add->Add to Wave 如图 2.10 所示。

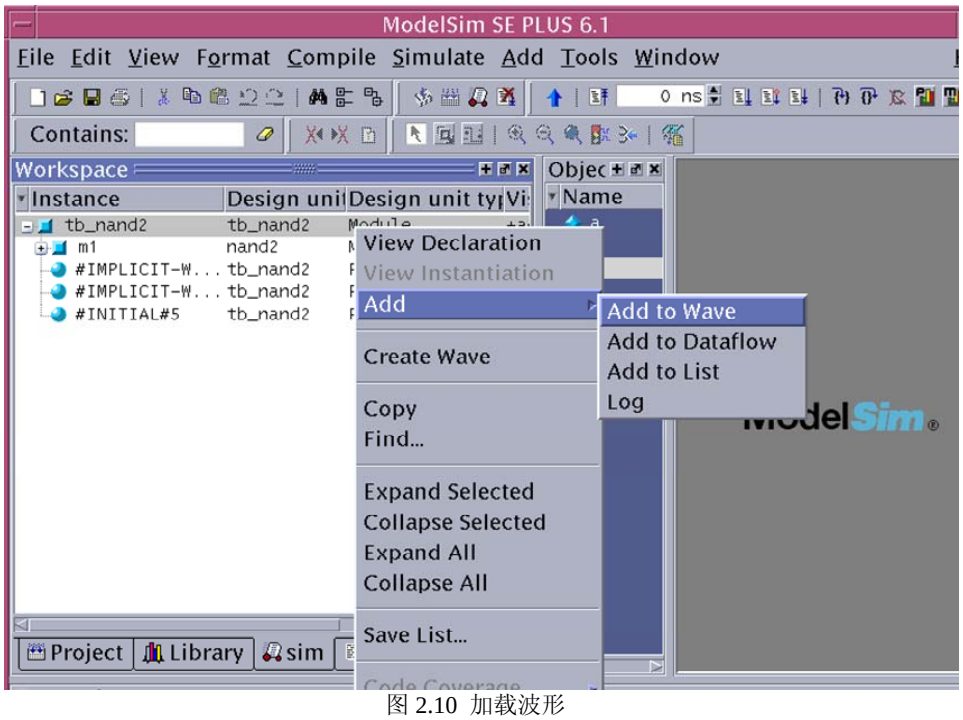

图 2.10 加载波形

此时用户界面出现 Wave-default 新的内嵌窗口，如图 2.11 右方所示，波形窗中出现了待观察的信号 a、b、y 信号的名字。

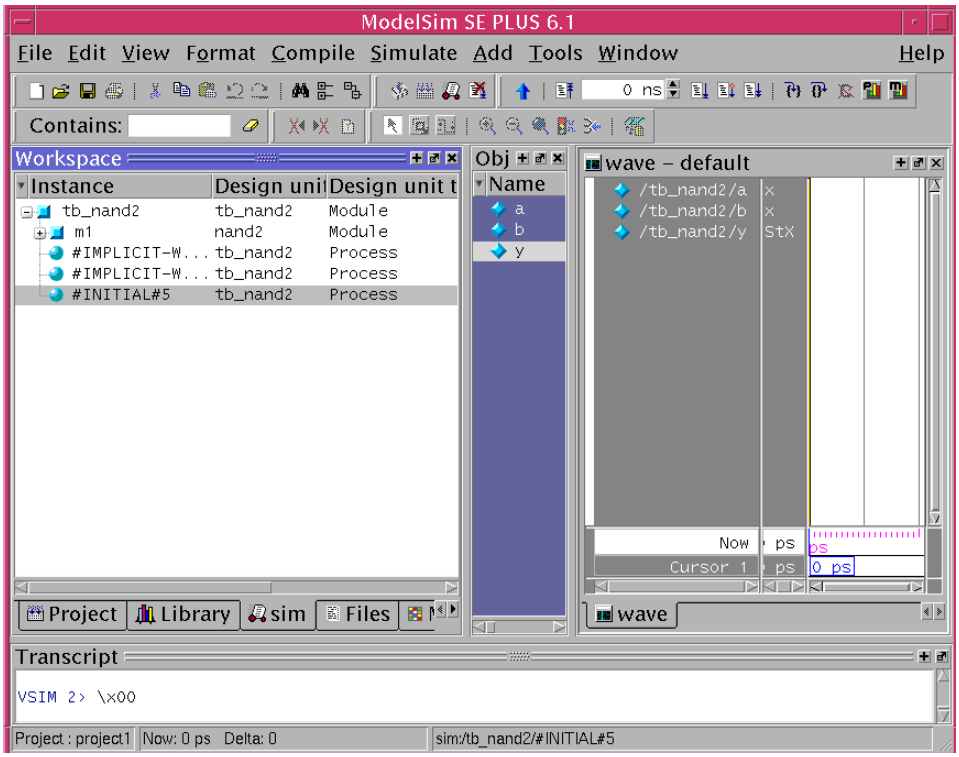

图 2.11 加载波形

2.5 运行仿真器

在 Transcript 窗口中输入 run -all，然后 enter 键确定(或点击工具栏按钮)，运行仿真。在仿真结束时会弹出 Finish Vsim 窗口，选择 No，否则会退出仿真器。

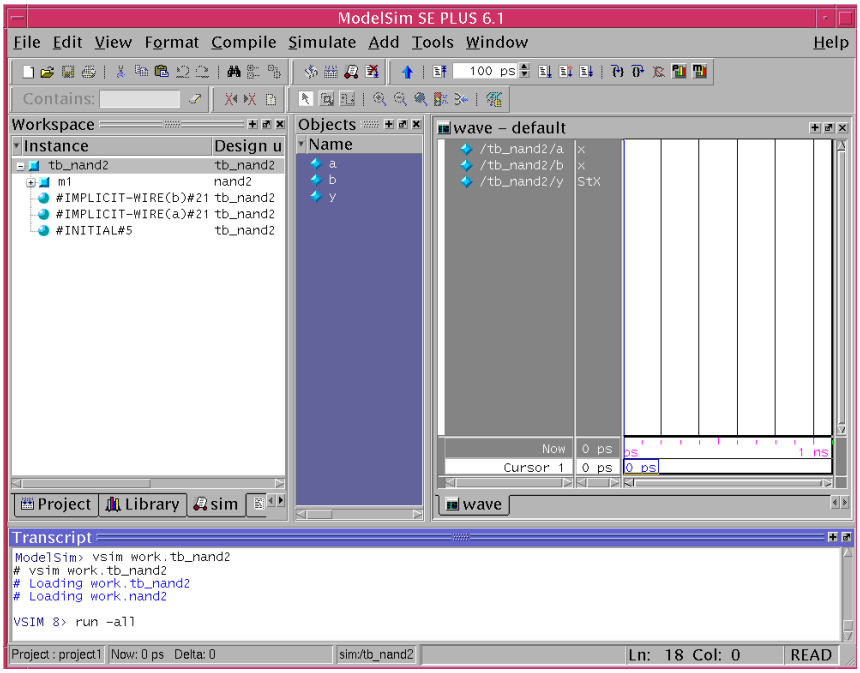

图 2.12 运行仿真

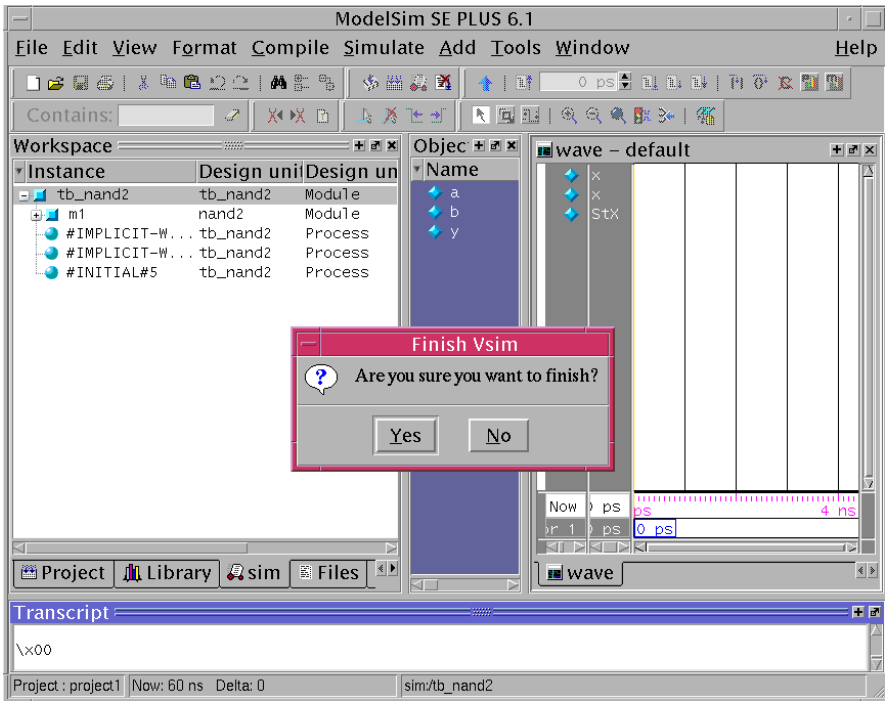

图 2.13 停止仿真

波形窗内嵌在主窗口内很不容易观察，可以点击图 2.14 中波形窗右上角的按钮(Dock/Undock)将波形窗单独调出来观察。

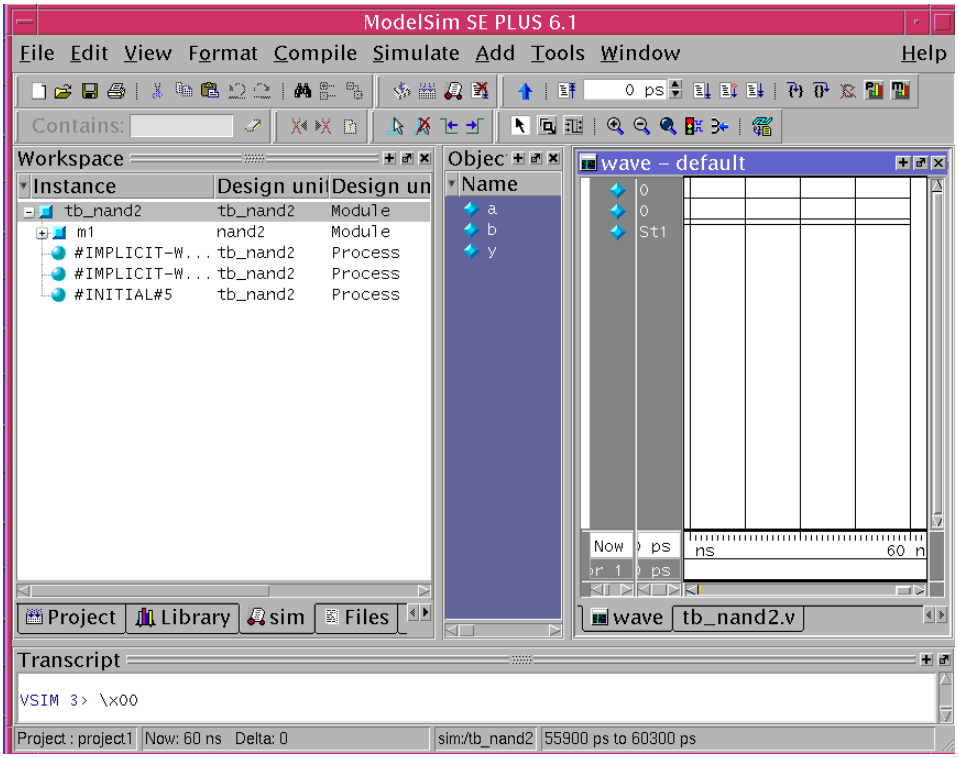

图 2.14 波形观察

调出来的波形窗口如图 2.15 所示，可以通过拉动波形窗下面的滚卷条来调节显示波形的位置时刻，点击工具栏中的按钮可以从整体上对波形进行观察。

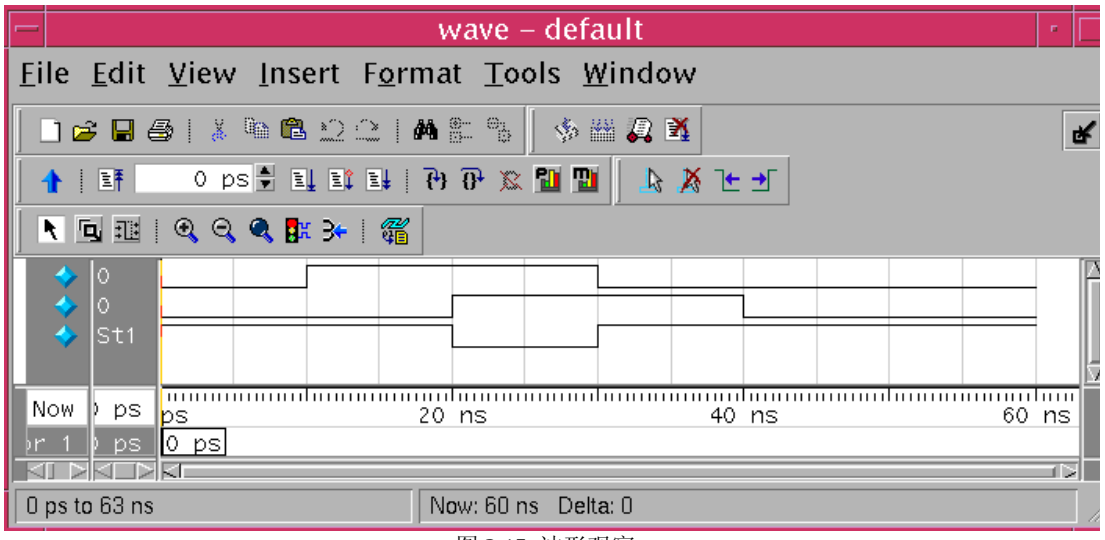

图 2.15 波形观察

观察波形，如果波形是错误的，可以在用户工作区修改源代码。修改完代码以后保存，重新运行仿真。重新运行仿真时点击工具栏中的 Restart 按钮，弹出 Restart 对话框如图 2.16 所示。

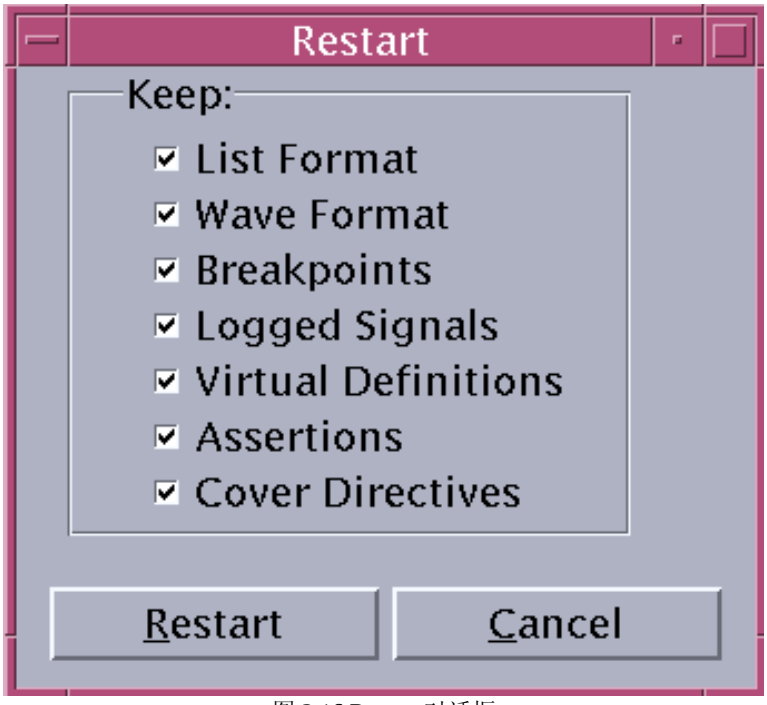

图 2.16 Restart 对话框

点击 Restart 按钮以后，重新点击按钮执行仿真。

如果观察波形正确，说明设计达到了要求，可以退出仿真器。首先在 Transcript 窗口中输入命令 quit –sim 后 enter 键确定执行(也可以利用菜单 Simulate->End Simulation 菜单完成，不过此时要在弹出的对话框中选中 YES 确定退出仿真)，退出仿真，如图 2.17 所示。

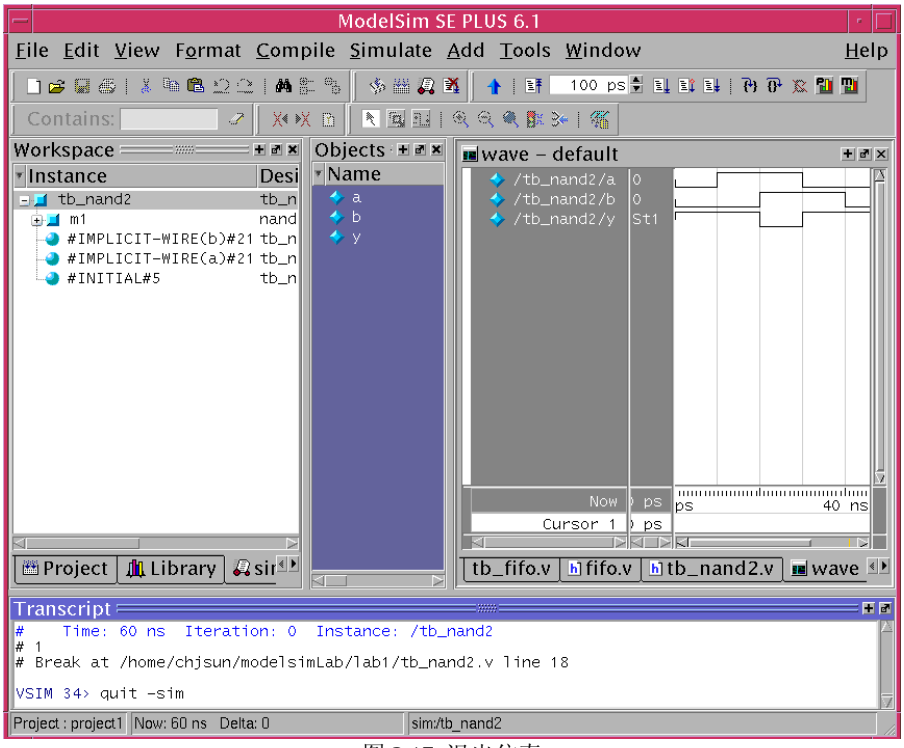

图 2.17 退出仿真

利用 quit -sim 命令退出仿真以后的界面又恢复到了启动仿真前的状态，如图 2.18 所示。注意右边用户界面处的几个窗口同学们可以自己关掉。

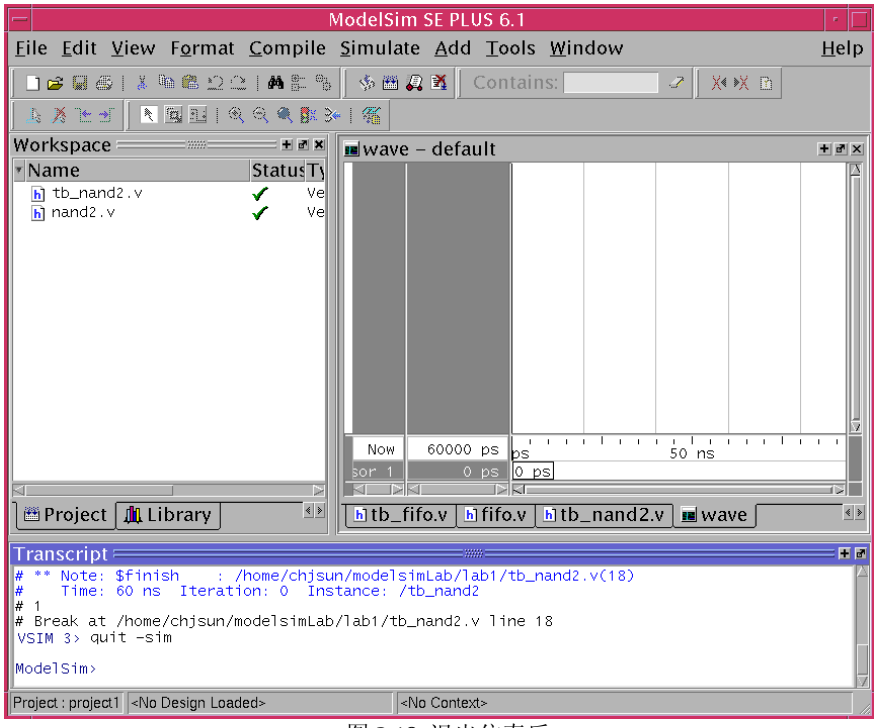

图 2.18 退出仿真后

退出仿真以后用户可以进而退出整个仿真器，只需要在 Transcript 窗口中继续输入 quit 命令确定执行即可，不过为了完成后续的实验请同学们不要退出仿真器。

3. ModelSim 的用户界面

3.1 ModelSim 的 Debug 窗口

上面我们对 modelsim 怎么仿真 verilog 代码有了一个大概的了解，以下我们更详细的介绍 modelsim 的不同仿真、调试的功能。

点击主窗口上的 View->DebugWindow->All Windows…，可以看到下拉菜单中共有 7 个窗口。它们分别是：Dataflow 窗口、List 窗口、Wave 窗口、Active 窗口、Process 窗口、Locals 窗口、Objects 窗口、Watch 窗口。点击 All Window 以后这些窗口都显示出来。

分别通过点击这些窗口右上方的 Dock/Undock 按钮可以将这些窗口从主窗口中调出来单独观察。下面我们将简单介绍这几个窗口的功能。

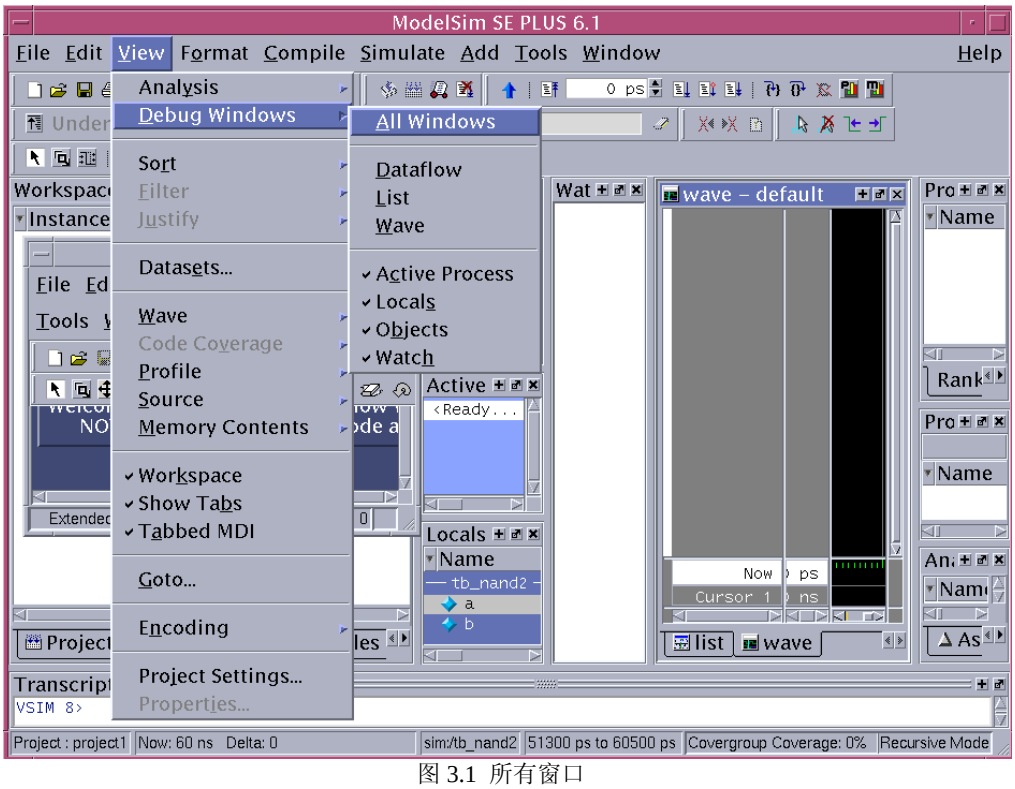

图 3.1 所有窗口

3.2 Main 窗口

在介绍 debug 窗口之前首先介绍一下主窗口。如图 3.2 所示(可能每个人启动 vsim 后的主窗口会略有不同)。

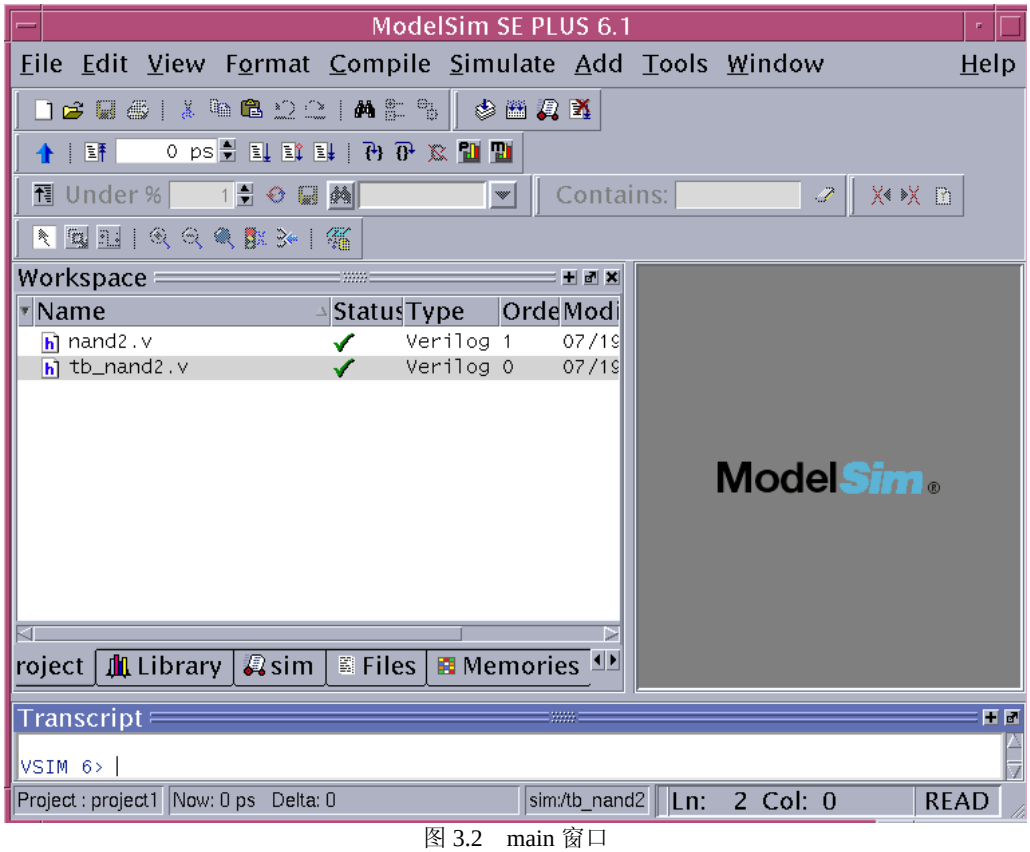

图 3.2　main 窗口

主窗口最下面的 Transcript 窗口最初的提示符为 ModelSim>，在加载设计以后，显示的提示符为 VSIM 6>，VSIM 后面的数字表示执行命令的次数。它可以告诉我们仿真器目前的状态，不同状态有不同的行为，包括命令和输出信息都有不同。主窗口的 workspace 区最开始只有 Library 一个活页栏，我们可以在这个 Library 栏中加入新库或者编辑已有的库，也可以浏览和编辑库目录。从前面的仿真实例我们也可以看到，在建立工程以后 workspace 中会出现 Project 一栏，在启动仿真以后 workspace 中又会陆续出现 sim、Files、Memories 等栏。主窗口右端是一个用户工作区，可以在此处放置 verilog 源代码、仿真波形等 debug 窗口。如果把所有的窗口都打开的话，此时的主窗口如图 3.3 所示：可以发现主窗口内嵌了很多其它的 Debug 窗口，可以把它们一一调出来观察。

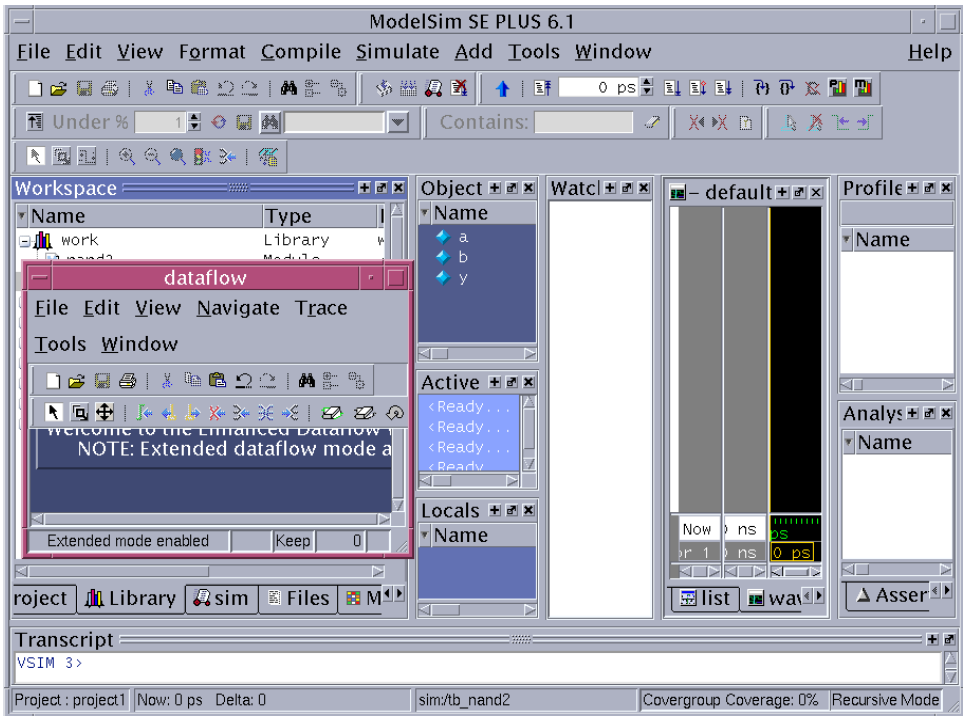

图 3.3 所有窗口

3.3 Wave 窗口

Wave 窗口用于显示待观察的信号波形。对于 VHDL，Wave 窗可以显示信号与过程变量，对于 Verilog，波形窗可以显示的对象是线网、寄存器变量和命名事件。利用 Wave 窗口可以观察任意指定信号的波形；可以打开新的波形窗以便观察更多的信号；可以对指定的信号进行组合，建立虚拟的总线；可以改变信号和向量的基数(例如将二进制显示形式变成十六进制形式显示)便于查看；还可以将信号的波形打印出来。工具栏中提供了大量的工具按钮便于指定波形缩放，选定观察波形的范围。大家可以一一尝试点击一下这些按钮，可以很容易的了解这些按钮的功能。

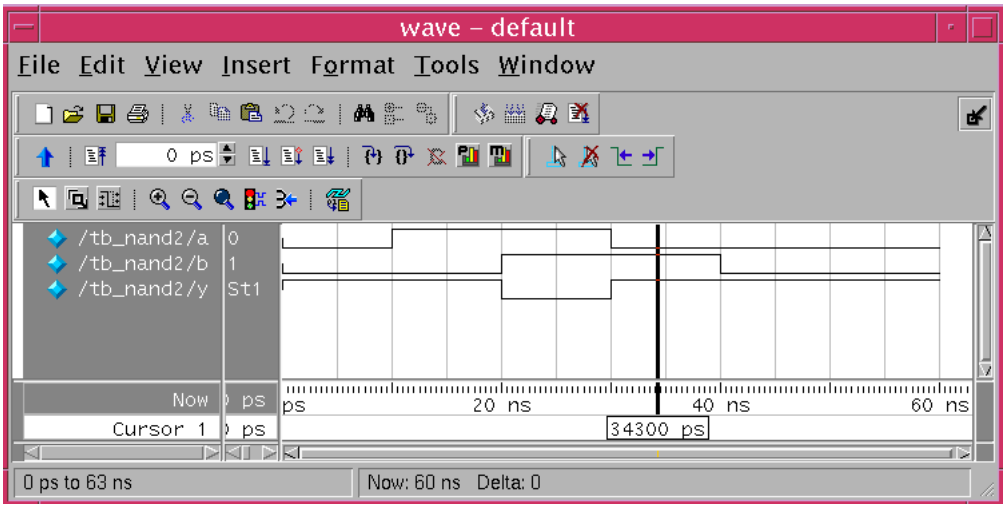

图 3.4 波形窗口

3.4 Process 窗口

进程窗口可以显示外部和内部的进程，通过工具栏中的 View 按钮可以点击选择两种观察模式。它们分别是 View Active 和 View In Region 两种观察模式。

如图 3.5 所示的是 View Active 模式下的窗口。这时会将当前仿真周期内所有的将要执行的进程罗列出来。

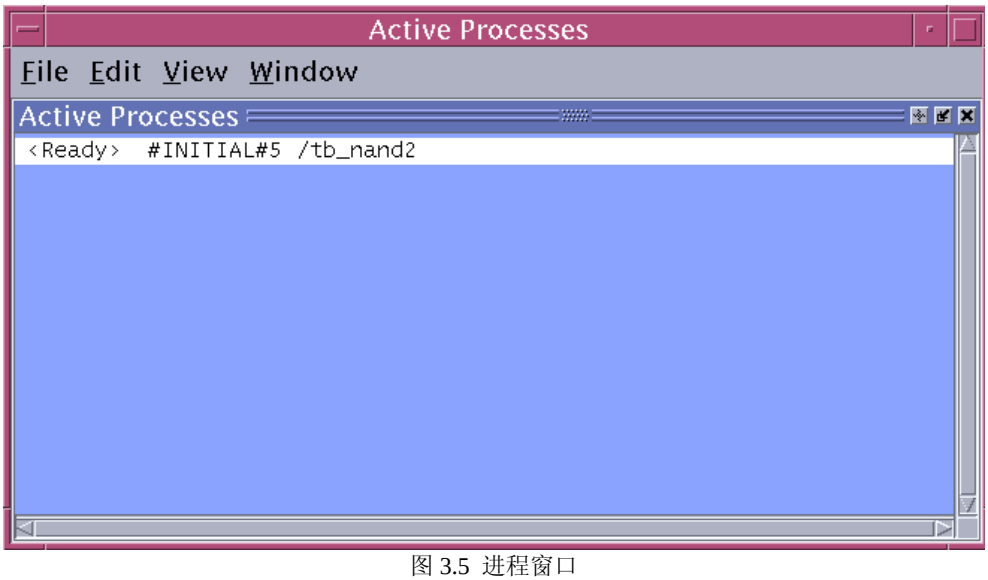

图 3.5 进程窗口

如图 3.6 所示的是 View In Region 模式下的窗口，这时将选择的结构中所有的进程列出来。

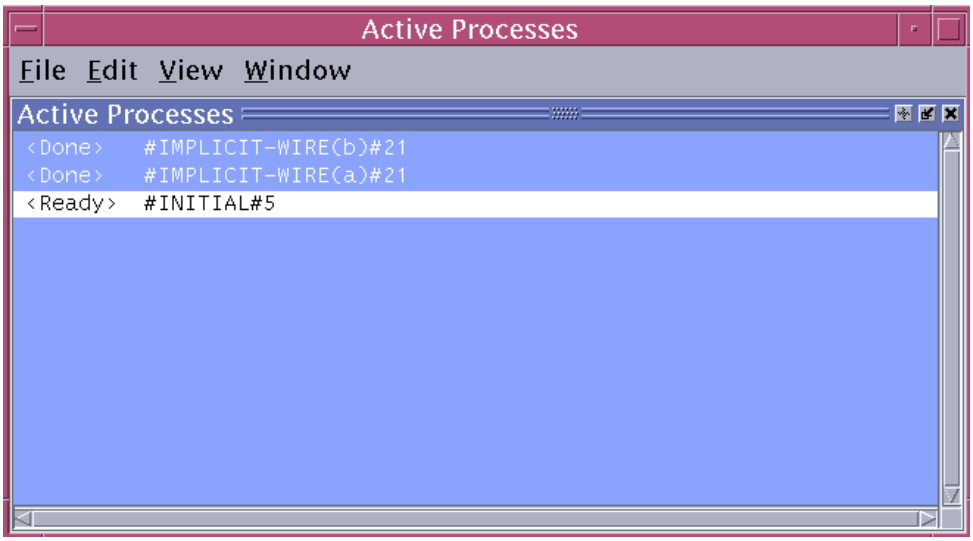

图 3.6 进程窗口

图 3.6 中最左边的一列有 3 种标记显示不同的进程，<Ready>代表在当前 delta 时间内要被执行的进程，<Wait>代表进程正在等待 VHDL 信号或者 Verilog 线网和变量的改变，或等待到超时，<Done>代表进程在没有超时或者没有敏感列表的情况下，执行了 VHDL 的 Wait 语句。

3.5 Objects 窗口

Objects 窗口显示了被选中的设计层次模块的信号名以及它们的值。信号可以是 VHDL 信号，可以是 Verilog 线网、寄存器变量和命名事件。窗口中的信号能够支持“拖放”功能，即将信号拖放到 Wave 窗或者 List 窗。Edit 菜单中的 Force 菜单可以用于产生激励。View 菜单中的 Filter 可以帮助快速显示或者不显示想要观察的信号类型，例如输入，输出，或者内部信号等。

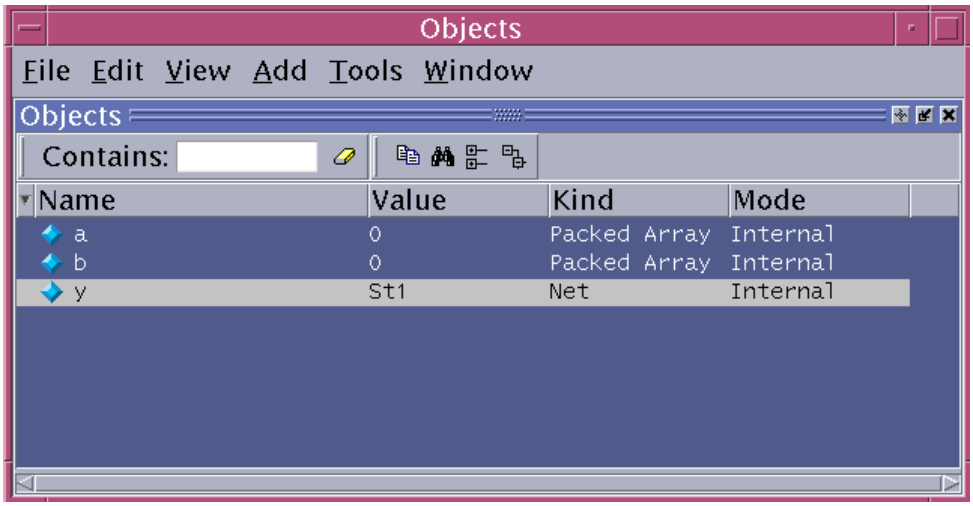

图 3.7 object 窗口

3.6 Locals 窗口

Locals 窗口显示下面即将被执行的语句中的可以马上被看到的数据对象及其值。（即将被执行的语句在源程序中是被绿色的箭头标示的）。Locals 窗口包含了两列，第一列列出了数据对象的名字，第二列是其数值。

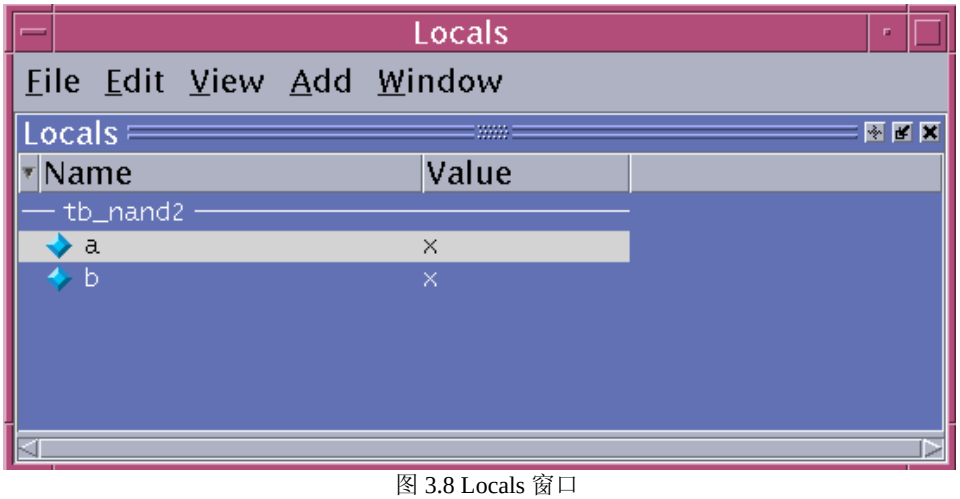

图 3.8 Locals 窗口

3.7 Watch 窗口

Watch 窗口显示了当前仿真时间下的信号与变量的值，与 Objects 窗口和 Locals 窗口不同的是，Watch 窗口允许你观察当前设计中任意的信号与变量的值。

在 Watch 窗口中被观察的值可以是 VHDL 的信号、变量、generic，也可以是 Verilog 的线网、寄存器、命名事件与模块参数。要在 Watch 窗口观察信号或者变量，只需要从 Object 窗口或者 Locals 窗口中将信号拖放进来即可。

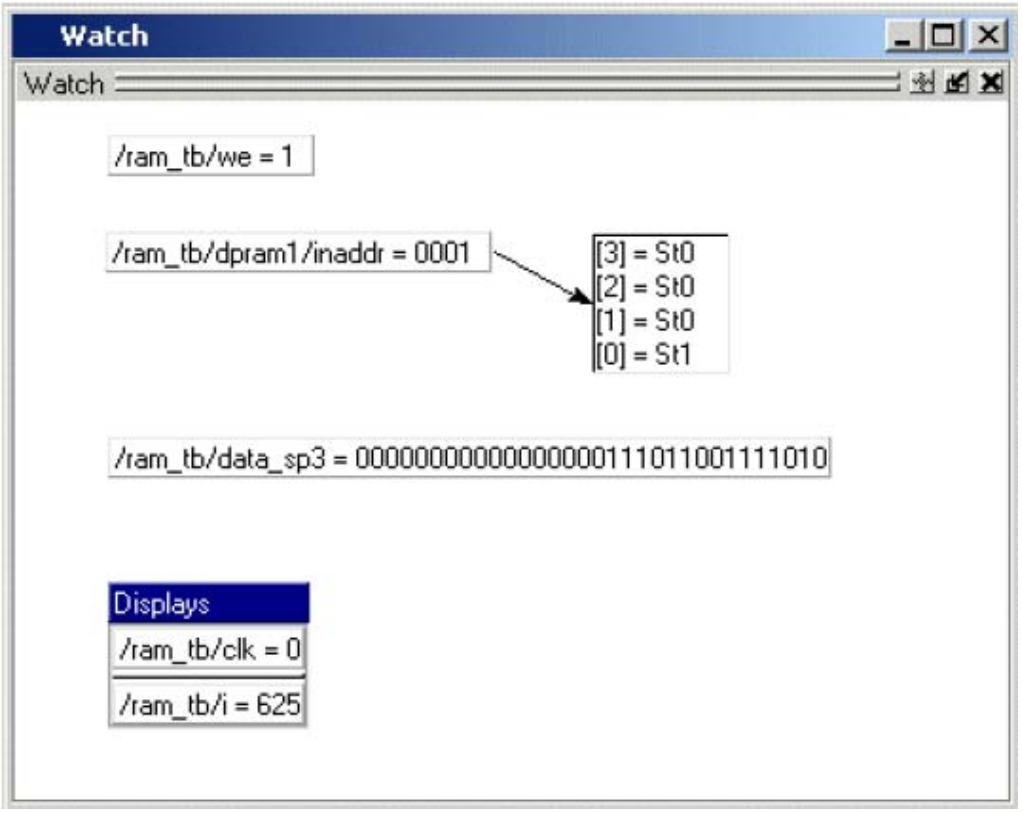

图 3.9 watch 窗口

3.8 List 窗口

List 窗口使用表格方式显示仿真结果。其中 VHDL 显示信号和过程变量，Verilog 显示线网和寄存器变量。可以从 List 窗口将信号拖放到其它的窗口，或者将信号从其它的窗口拖放到 List 窗口。这个窗口支持查找功能，可以通过 Edit 菜单下的 Find 进行查找。可以用 Tools 下的 Combine Signals 来建立用户定义的虚拟总线。通过 File 菜单中的 Write List 可以将 List 中的内容导出。如图 3.10 显示了 List 窗口，在窗口左侧的数据显示了时间信息。在窗口右侧的数据显示了在指定时间下，各个信号及变量的值。

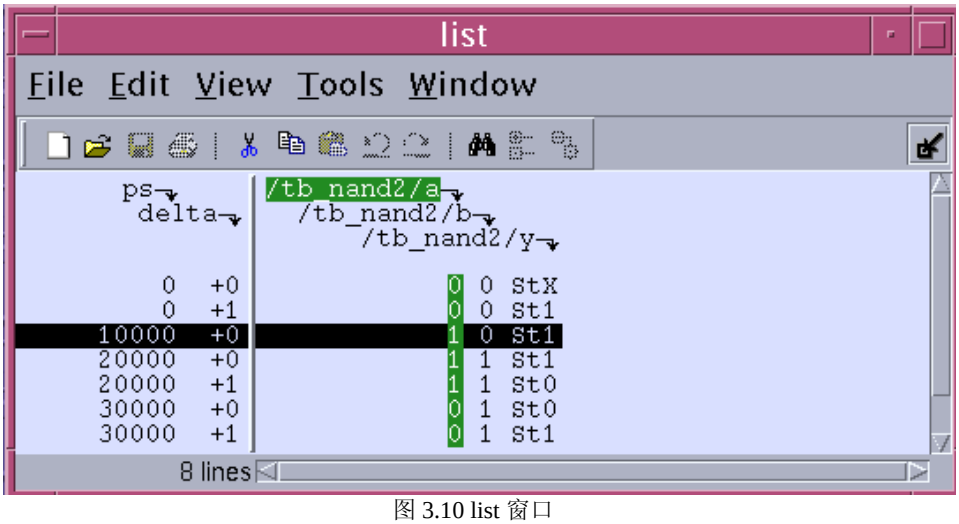

图 3.10 list 窗口

3.9 Dataflow 窗口

Dataflow 窗口对 VHDL 信号或者 Verilog 线网的信号驱动进行图示化的跟踪。其中驱动信号或线网的进程位于左边，读取的进程或被线网触发的进程放在右边。对进程而言，被读取的信号或触发该进程的网线在左边，被进程驱动的信号或网线在右边。Dataflow 窗口内嵌了波形窗口，将波形与图示的模型之间建立了动态的链接，光标所处的波形上的信号的值可以在图上动态的得到显示。更为重要的是 Dataflow 具有强大的波形跟踪功能，在 Trace 菜单下的各个子菜单命令可以很方便的进行波形跟踪，尤其是 ChaseX 功能能够对未知态 X 进行跟踪，后面的教程内容将对此进行演示。

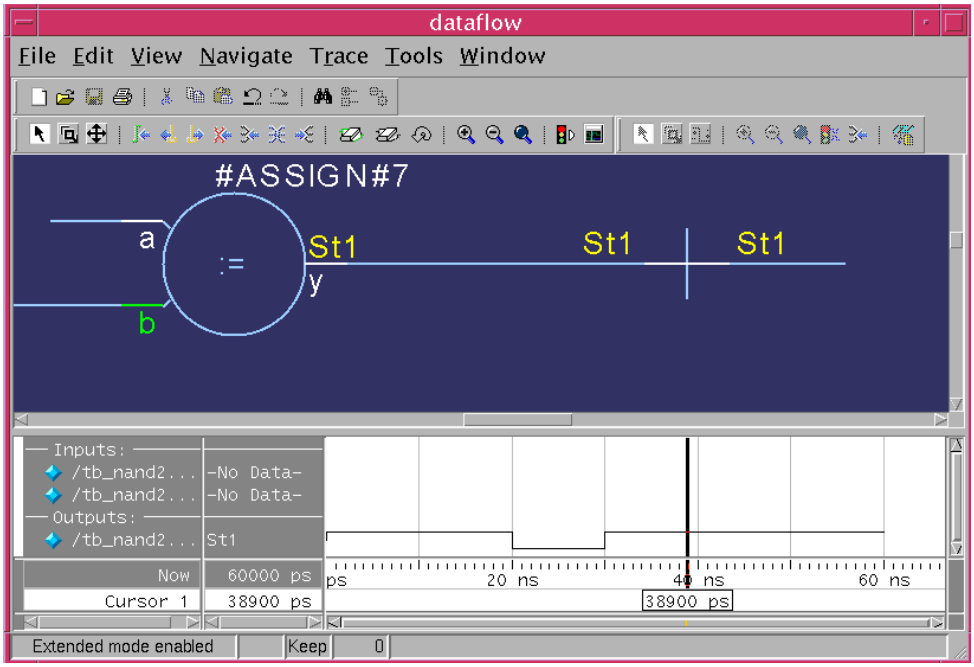

图 3.11 dataflow 窗口

3.10 ModelSim 调试窗口特点

ModekSim/SE6.1 版本的 debug 窗口共有七个，它可以支持任何窗口的多个副本，支持拖放，在一个窗口选择 HDL 项后，用鼠标左键，这些选项能被从一个窗口拖放到另一个窗口。HDL 项可以从 Dataflow、List、Signals、Source、Objects 和 Wave 窗口中拖出来，然后把它们放到 List 或者 Wave 窗口中。

对于 Dataflow 窗口，当一个进程被选到这个窗口时 Process、Objects、Source 窗口会被更新。

对于 Process 窗口，当一个进程被选择，Dataflow、Objects 窗口将会被更新。

对于 Objects 窗口，当 Object 窗口被选中，Dataflow 窗口的值将会被更新。

对于大部分的窗口，都允许用户通过 Edit 下的 Find 选项进行查找。

4. 功能仿真和时序仿真

在这里我们通过一个 8 位加法器的仿真验证来演示如何使用 modelsim 进行功能仿真和时序仿真。

4.1 功能仿真

首先进行功能仿真。我们为接下来要做的这个功能仿真实验建一个工程名为 project2 的工程。在建立 project2 工程之前我们首先关闭 project1 工程。过程如图 4.1 所示：右键点击当前 project 栏中的任意一处得到下拉菜单，点击 Close Project，在弹出的对话框中点击 Ok 确定关闭，当前的 project1 工程便会被关闭。

关闭 project1 以后系统又恢复到了没有 project 栏的状态。建立一个新的工程名为 project2

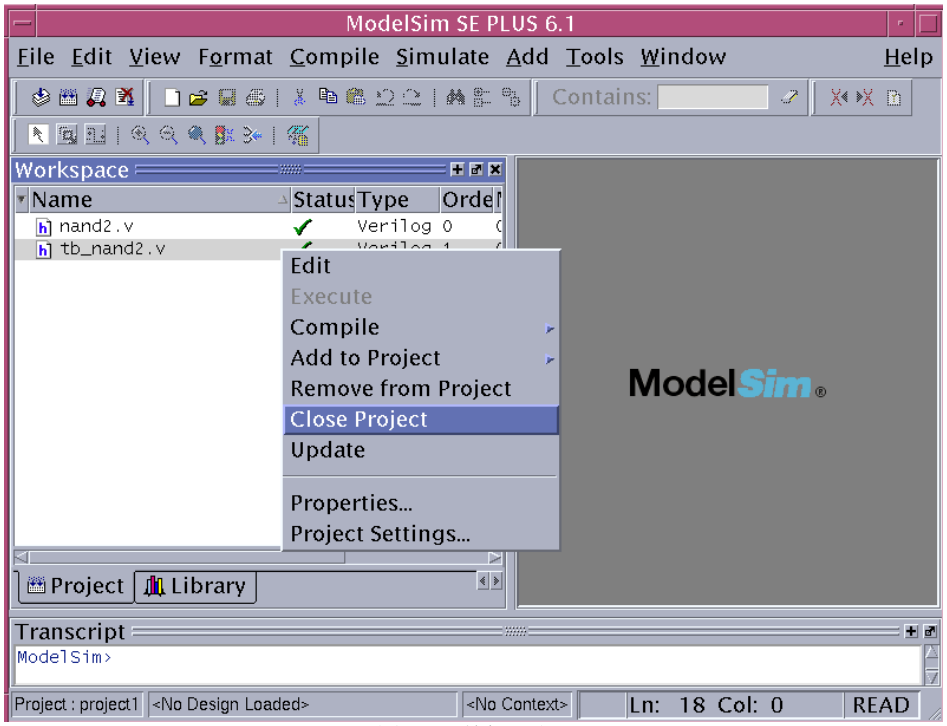

图 4.1　关闭工程

的工程。这里的步骤和前面的新建工程的方式是一样的。注意在建立这个工程的时候修改一下 Project Location，将工程指定到 lab2 的目录下面，如图 4.2 所示。

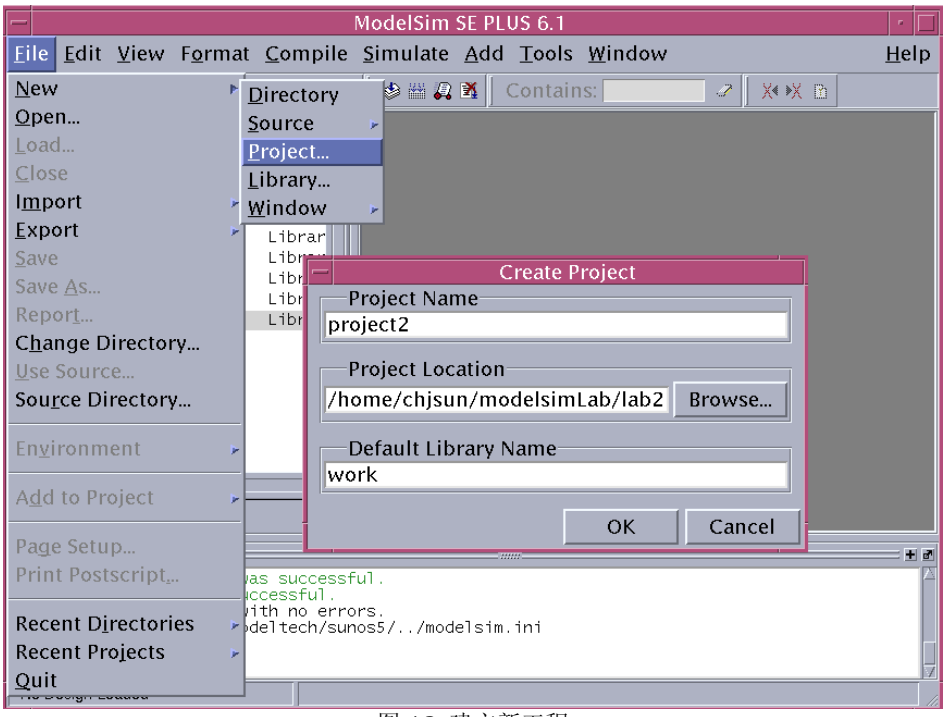

图 4.2 建立新工程

点击 Ok 确定，重复前面的步骤，如图 4.3 所示。我们选择 Add Existing File 后在 File Name 栏点击浏览，进入 lab2 文件，里面有几个 Verilog 源程序文件，按住 Ctrl 键把 add4.v、add8.v、fa.v、addertb.v 四个源程序选入刚建立好的 project2 工程中来。进一步进行编译、加载设计、仿真等步骤，这些过程前面已经介绍过了，这里就不再赘述。

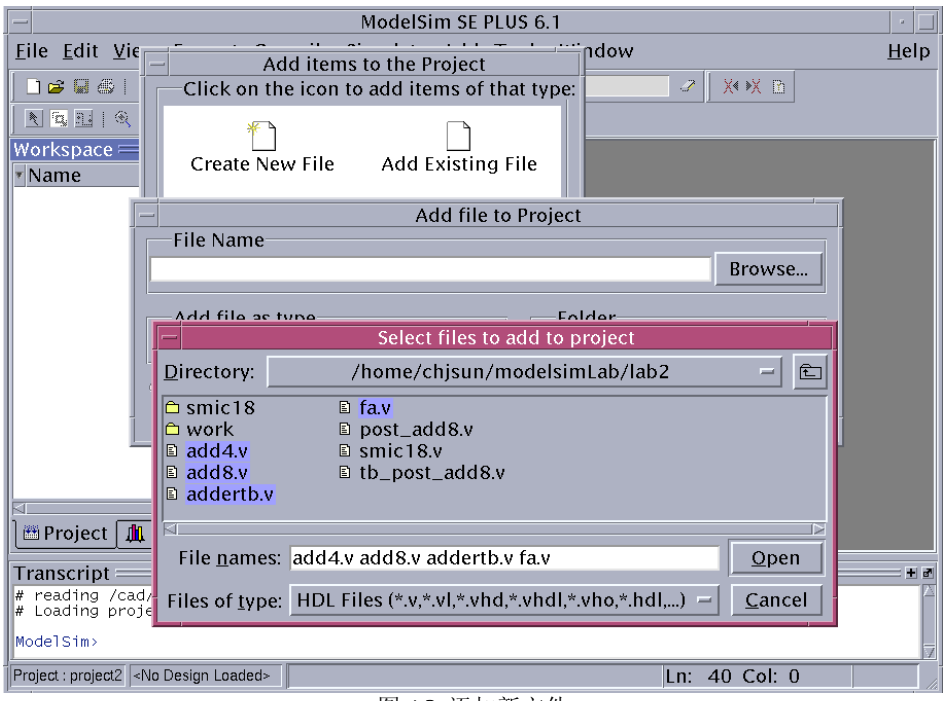

图 4.3 添加新文件

得到的总体波形如图 4.4 所示。

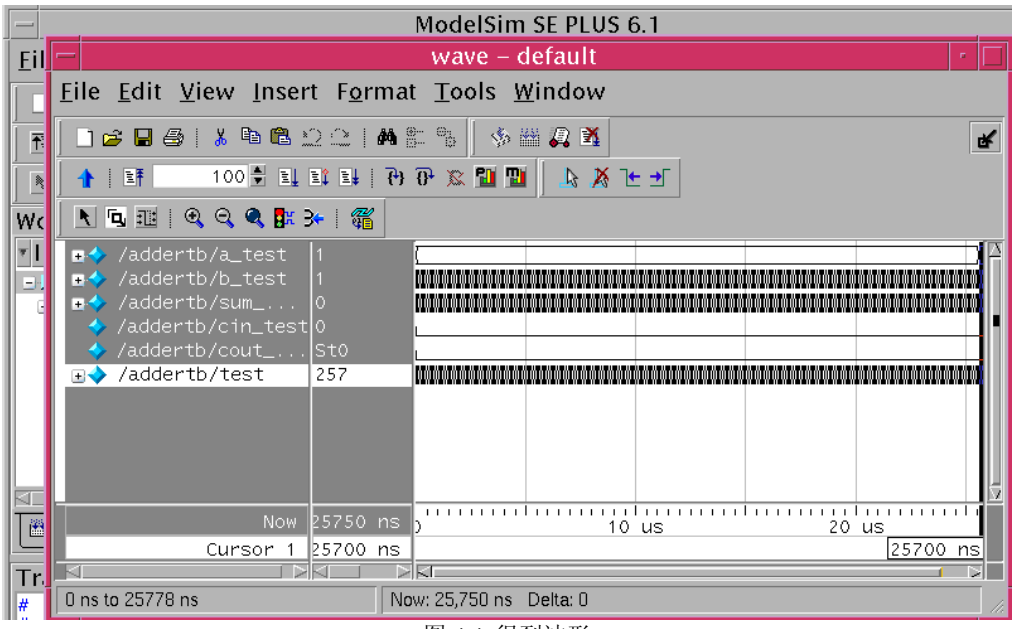

图 4.4 得到波形

从得到的局部放大的波形可以看到功能仿真的特点——没有时间延时，输入与输出之间的波形是对齐的，正如图 4.5 所示。这正是我们预料之中的事情，由于功能仿真只针对我们设计的功能进行仿真，因此它是不包含延时的。

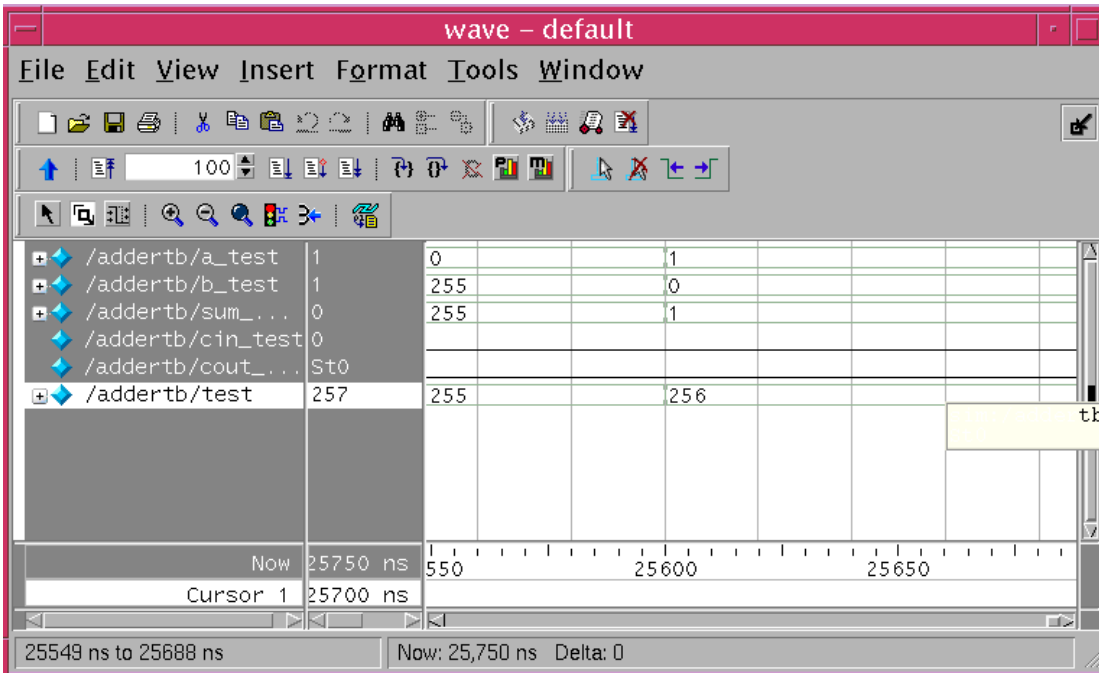

图 4.5 局部放大波形

4.2 时序仿真

选择 File->Add to Project->Existing File…菜单，进一步在 project2 工程中加入 lab2 目录中的以下 3 个文件：post_add8.v、smic18.v、tb_post_add8.v，其中，post_add8.v 是综合后的网表，smic18.v 是标准单元的 verilog 仿真模型，tb_post_add8.v 是网表的测试文件。

为了把 smic18.v 模块单独编译到另外一个库中，使它与自己设计的模块区分开来，我们在这里新建一个名为 smic18 的库。完成以后在 library 栏中可以看到多了 smic18 库。

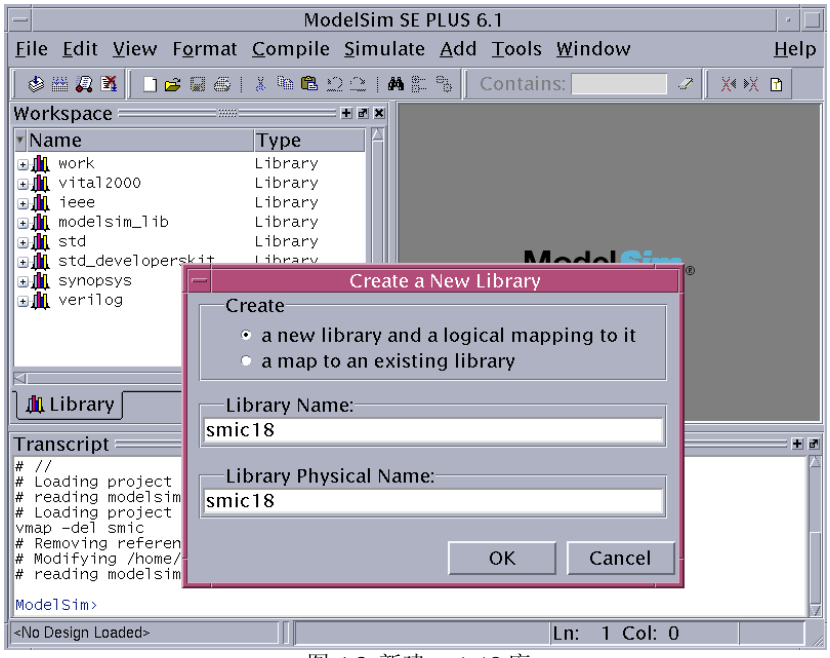

图 4.6 新建 smic18 库

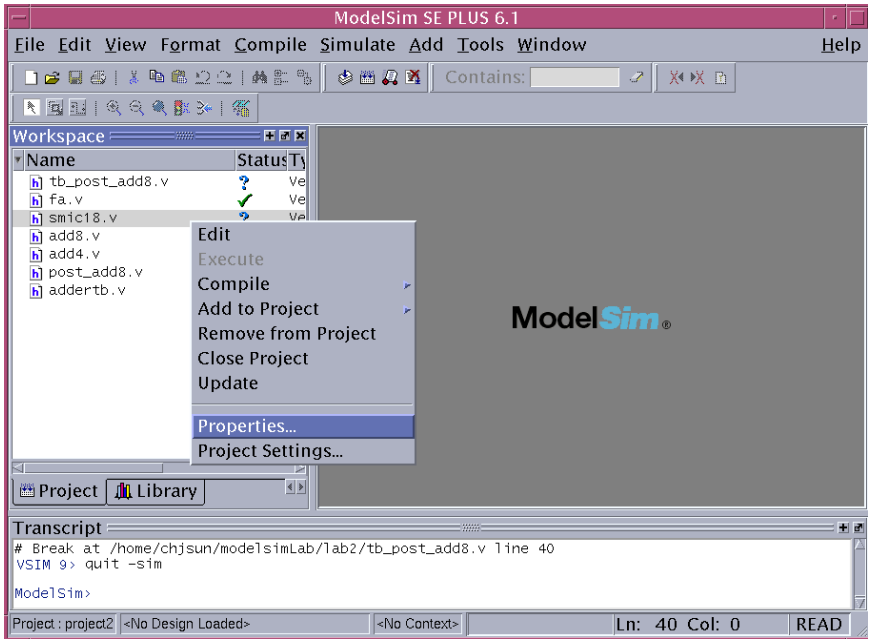

图 4.7 更改编译属性

将 simc18.v 编译到 smic18 库里来，要更改 smic18.v 的编译属性。选中 project 栏中 smic18.v，点击右键得到下拉菜单，如图 4.7，选择 properties，弹出如图 4.8 所示的对话框。将 Compile to library 右边的下拉菜单中选择 smic18，点击 OK 确定。此时对 smic18.v 进行编译，编译结果将进入指定的 smic18 库中，不会与自己的设计混淆。

对 post_add8.v 与 tb_post_add8.v 的编译属性不作更改，对两个模块进行编译，编译结果会自动地进入默认的 work 库中。

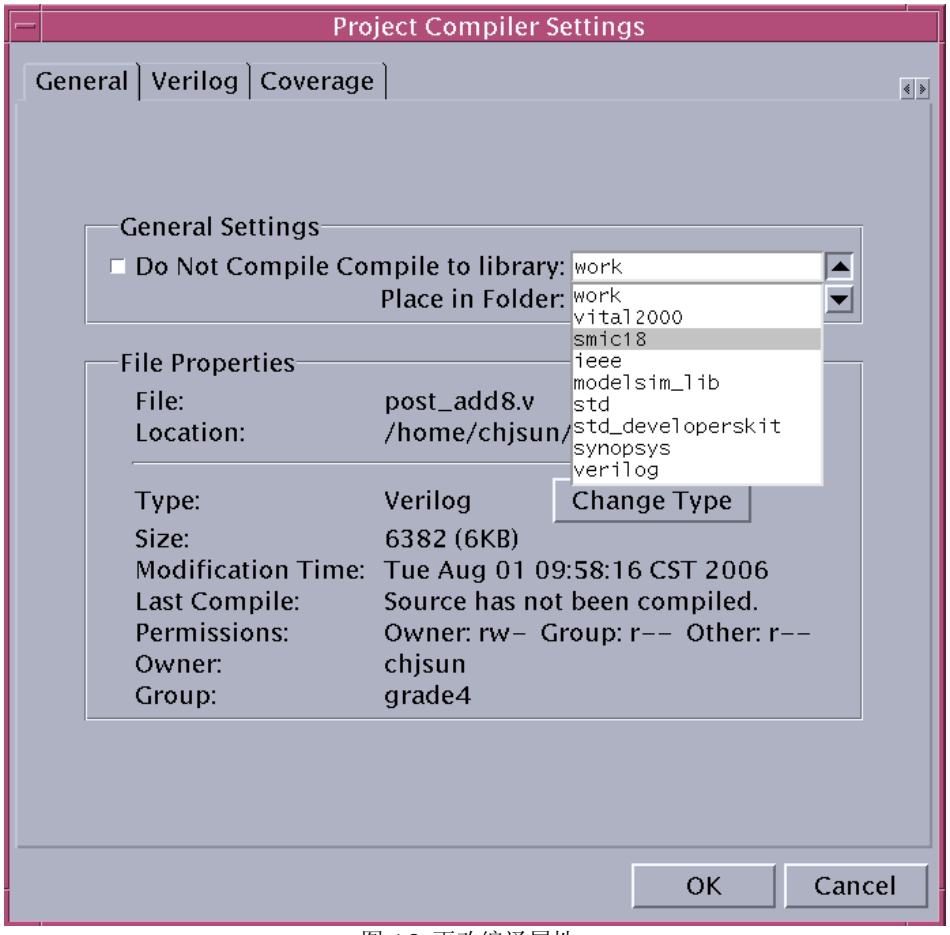

图 4.8 更改编译属性

编译完毕以后，在 Simulate 下拉菜单中选中 Start Simulation…，出现了如图 4.9 所示的 Start Simulation 对话框，在出现的对话框中首先选择 SDF 栏，然后点击 ADD 出现了 Add SDF Entry 对话框，点击浏览将 lab2 文件夹中的 readd8.sdf 文件选中打开。在 Add SDF Entry 对

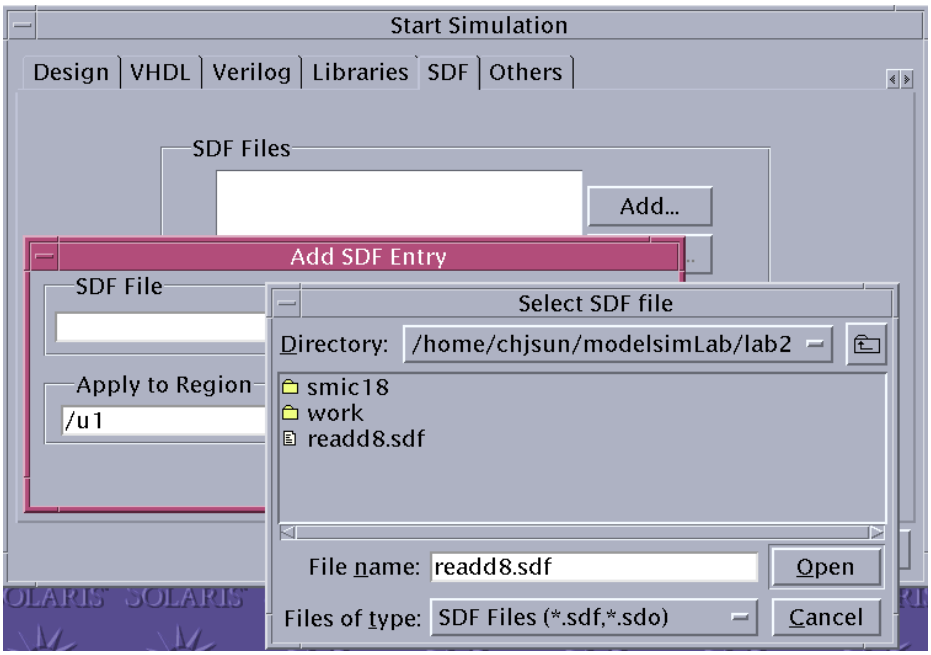

图 4.9 添加 sdf 文件

话框中的 Apply to Region 区填入“/u1”(u1 是被测元件在测试激励中的实例名，可参考 tb_post_add8.v 源代码)，点击 OK 确定。

然后点击 Library 栏，继续点击 Add 跳出 Select Library 对话框，选择 smic18 后点击 OK 确定。

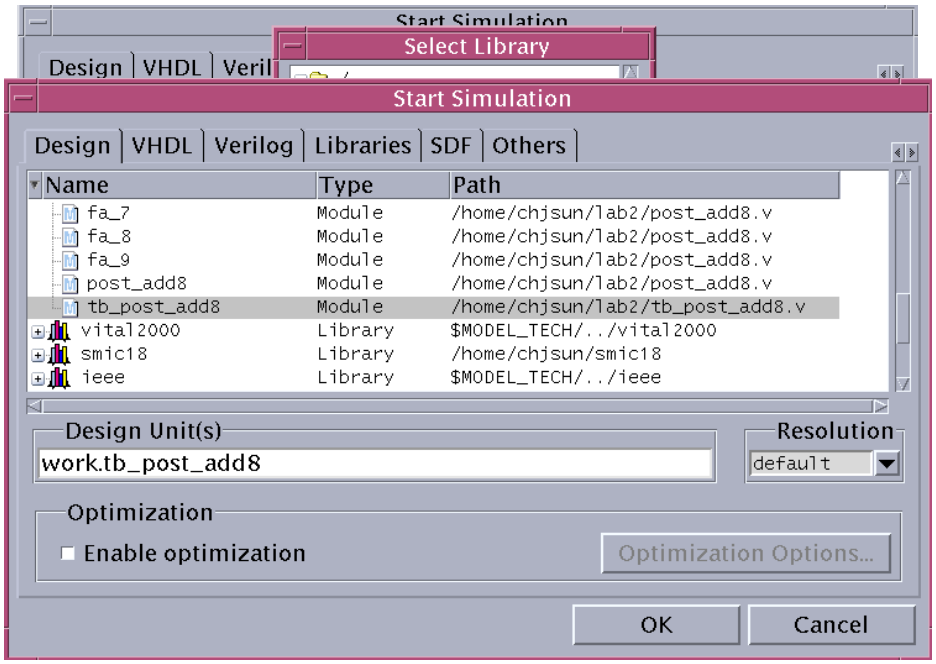

图 4.11 启动仿真

最后点击 Start Simulation 对话框中的 Design 栏，选择 work 库中的 tb_post_add8 模块，

点击 OK 启动仿真。如图 4.11 所示。

启动仿真以后，按照前面相同的步骤加载波形，运行仿真。由于加入了延时信息，仿真需要的时间较长。执行完毕以后得到波形结果。利用按钮来放大所要观察的波形如图 4.12 所示：

我们看到了波形中的输入输出信号已经不再是对齐的了，它们之间出现了延时。(注意利用工具栏中的这个按钮可以产生一个新的光标，然后可以利用鼠标来拖动新生成的光标到达指定的位置，此时自动显示原有光标与新光标之间的相对位置大小与它们各自的绝对位置，可以利用这两个光标测量信号之间的延时。)

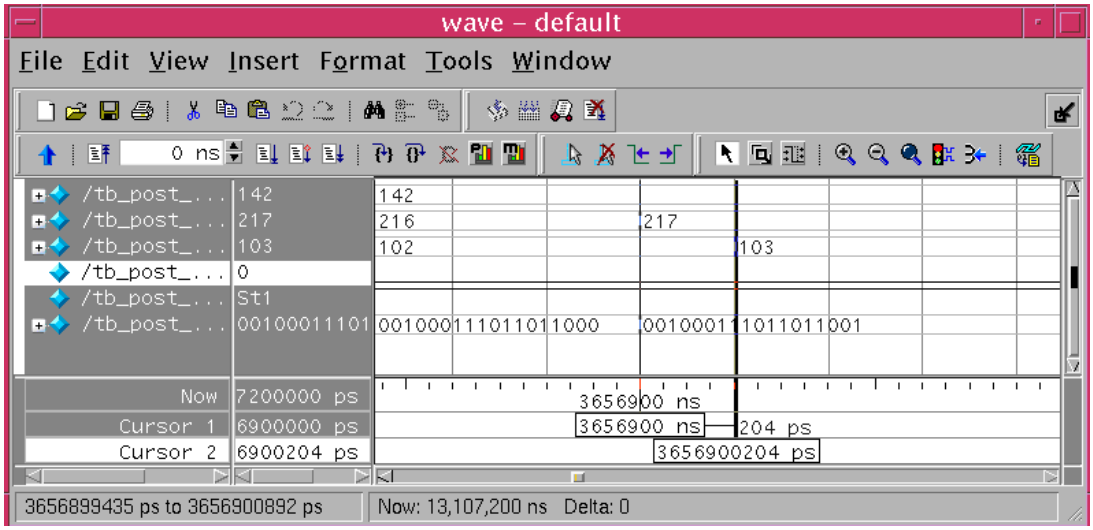

图 4.12 局部放大波形

5. 高级功能

5.1 波形追踪（ChaseX）

在这里我们要为大家演示 ModelSim 的高级功能之一即波形追踪的 ChaseX 功能，利用这一功能，ModelSim 可以迅速的帮设计者找到电路中未知态 X 的来源，便于电路的调试。

我们先在 lab3 目录下新建一个 project3 的工程，同上一个实验一样，在建立 project3 之前先把 project2 关闭掉。然后把 lab3 目录中的内容 bistcellreg.v 与 tb_bistcellreg.v 加载到 project3 中来。这里的步骤同前面是一样的，在这里不再重复。同样的，在添加了两个模块以后要进行编译，启动仿真的步骤。在这里要注意一点，不但要把顶层的模块 tb_bistcellreg 的所有信号加载到波形窗来进行观察，还必须要把被测元件 bitcellreg（例化名为 m1）加到波形窗。这在追踪未知态 X 的过程中是相当重要的，因为波形追踪功能只能追踪连续的未知态 X 值，中间不能有 X 值的间断，否则 ModelSim 无法实现对 X 值得追踪功能。这个过程如图 5.1，图 5.2 所示。

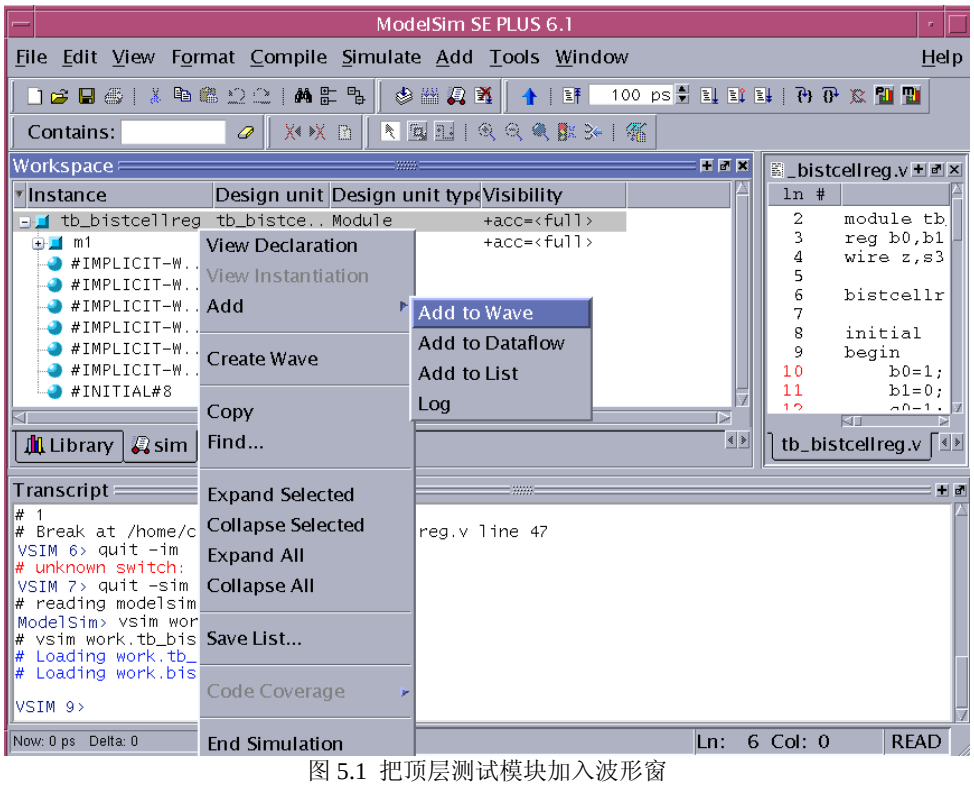

图 5.1 把顶层测试模块加入波形窗

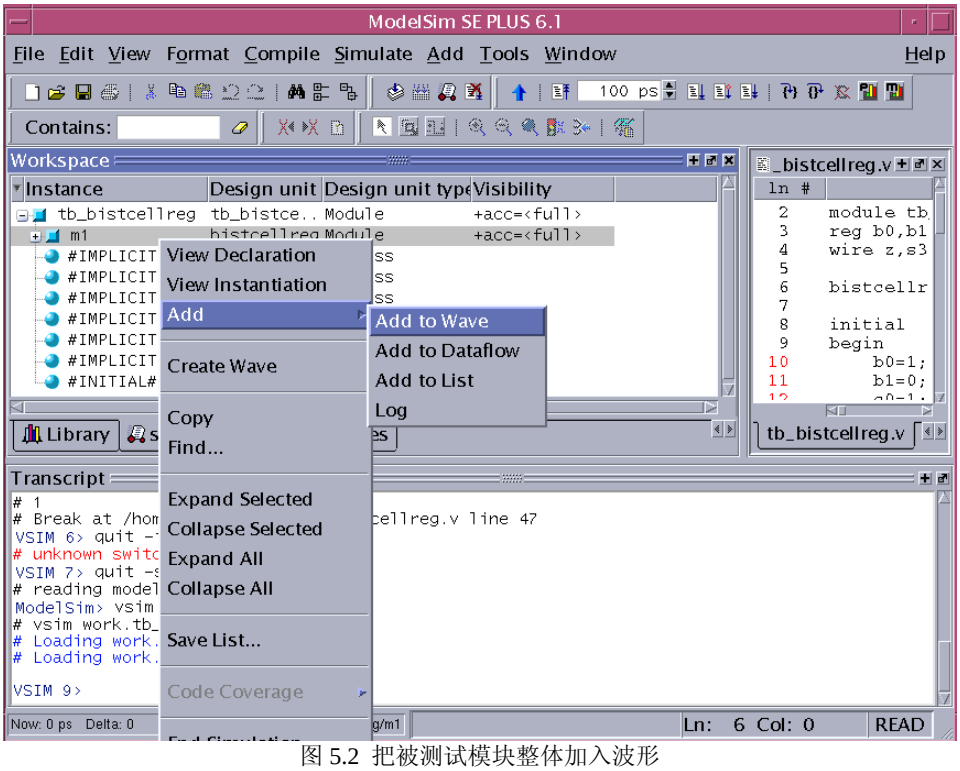

图 5.2 把被测试模块整体加入波形

按照与前面一样的步骤执行仿真，得到波形图，单独把波形窗调出来观察，可以发现 tb_bitcellreg 模块的输出端口 z 在 30ns 到 40ns 之间处于未知态 X.，此时我们想要找到使得输出为未知态的根源，如图 5.3 所示：

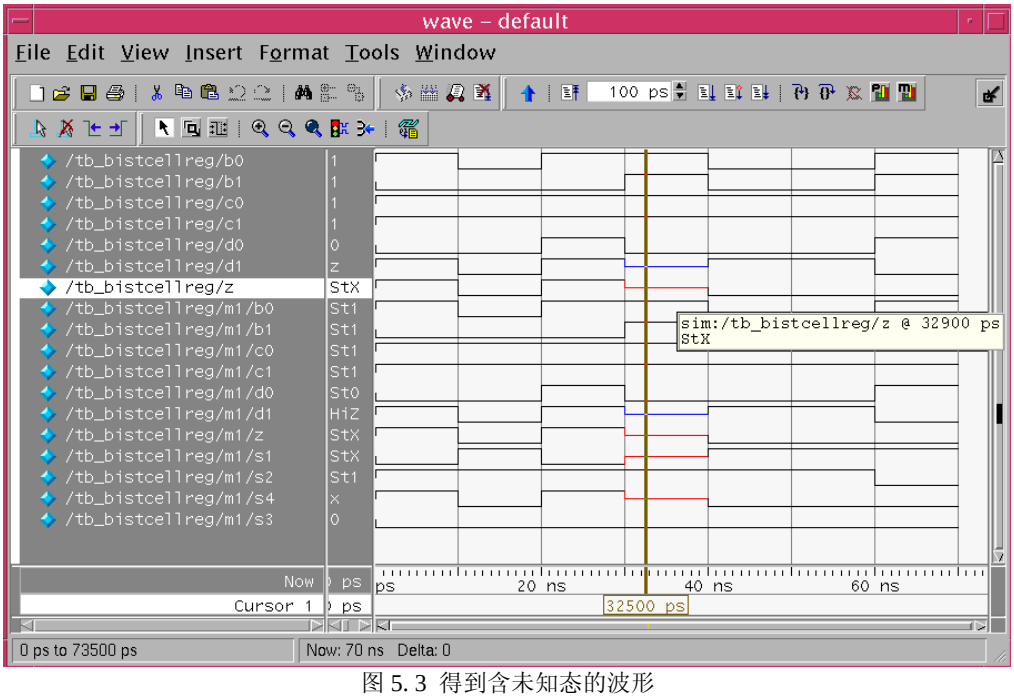

图 5. 3 得到含未知态的波形

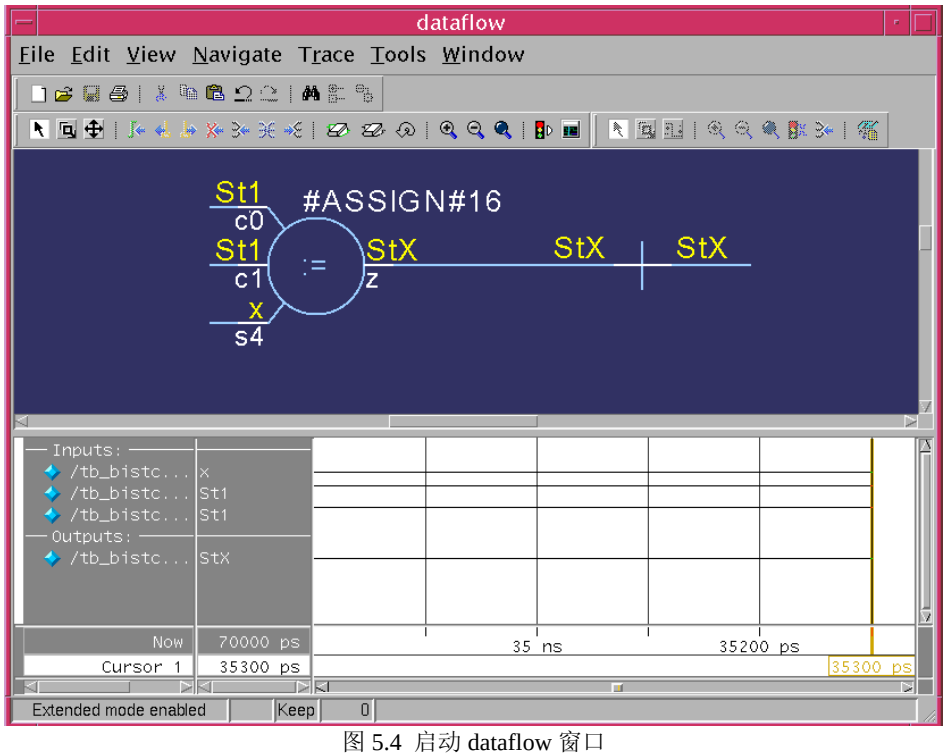

图 5.4 启动 dataflow 窗口

首先选中输出部分含有未知态 X 的输出信号 z，然后点击按钮  来启动 dataflow 窗口。启动后的 dataflow 窗口如图 5.4 所示：

得到这个窗口以后我们可以观察的 dataflow 窗口的特点，电路驱动位于图形的左边，电路负载部分处于电路的右边。图形中白色的文字代表了信号的名称，黄色的部分代表了在当前光标所处的位置时刻对应变量的值。信号的值与光标所处的时刻之间建立了动态的连接，注意要使用 ChaseX 功能时，必须要使光标位于未知态的位置，也就是波形中显示为红色的区域。此时图中黄色信号值对应的显示为 StX 值。下面我们为大家演示 ModelSim 强大的波形追踪功能。如图 5.5 所示，首先在 dataflow 窗口中选中最右端标记为 StX 的输出端，如果被选中，则显示为红色。然后点击菜单 Trace->ChaseX。

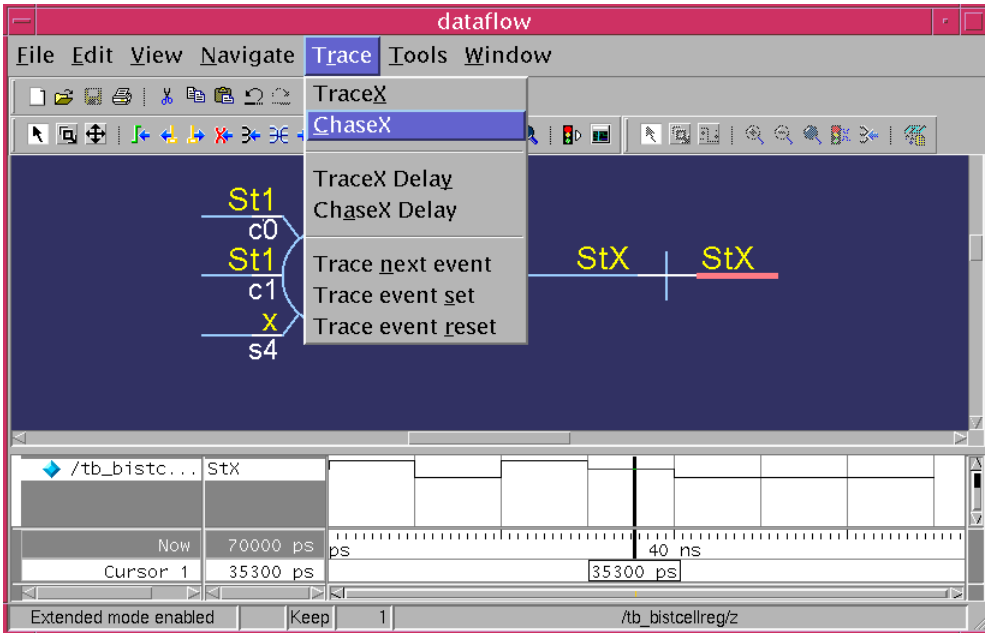

图 5.5 启动 ChaseX 功能

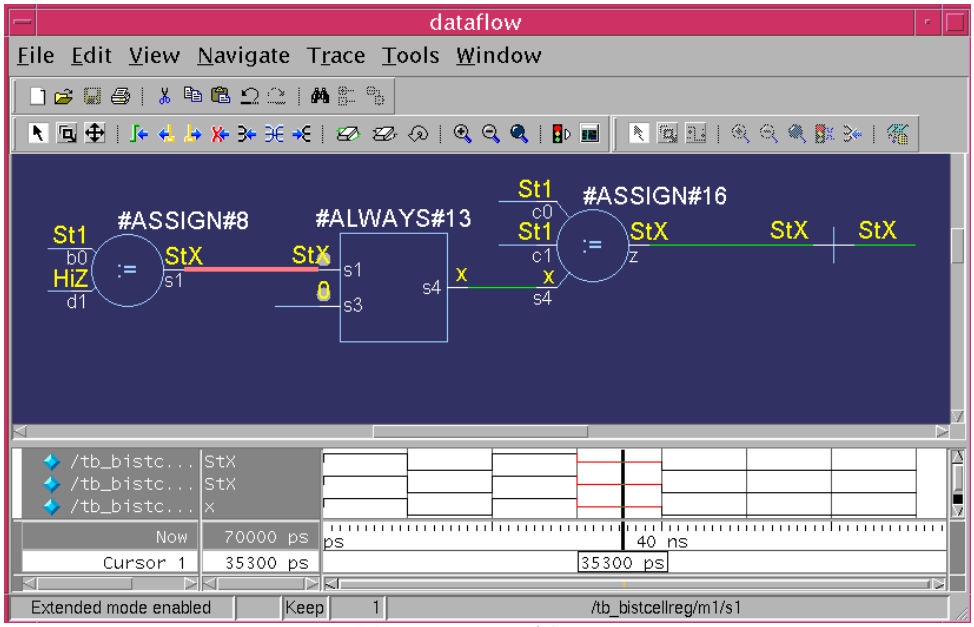

图 5.6 ChaseX 功能显示

点击以后得到图 5.6 所示图形：

通过显示出的图形窗口，我们发现原来在 30ns 到 40ns 之间输入 d1 的值为 HiZ，使得输出 z 在这一段时刻的值为未知态 X，点击#ASSIGN#16 进程符号，相应的在 dataflow 下面的波形窗也显示出对应的进程输入输出信号以及相应的值。从这里我们看到了 ModelSim 的强大的 ChaseX 功能。

5.2 代码覆盖（Code Coverage）

代码覆盖率是验证激励是否完备，检验代码质量的一个重要手段。测试激励的代码覆盖率至少要达到 95%以上，才能基本认为代码在逻辑上是通过质量控制的，才能进入综合步骤。

代码覆盖率是保证高质量代码的必要条件，却不是充分条件。即便是代码覆盖和分支覆盖都能够达到 100%，也不能肯定地说代码已经得到 100%的验证，除非所有的分支覆盖都能进行组合遍历。在大的设计中，想通过一个激励就验证完一个设计或者一个模块是不现实的，通常的做法是每一个激励都只验证电路功能的某个方面，整个电路的功能验证有数个激励共同完成，在这种验证方法中代码覆盖率更显得重要，因为可以通过代码覆盖率来控制激励对功能的覆盖程度。ModelSim 的 Code coverage 不但能记录各个激励对代码的“行覆盖”和“分支覆盖”，而且能够将各个激励的覆盖记录进行合并，做到对覆盖率的全面覆盖。

我们下面的这个实验为大家演示如何观察仿真过程中的代码覆盖率。

首先关闭当前的工程库 project3，在 lab4 目录下新建一个 project4 的工程，在工程中添加 lab4 中的 fifo.v 与 tb_fifo.v 两个模块。

为了观察代码覆盖，在编译之前首先要设置一下编译属性，在 project 栏中利用 Ctrl 键选中 fifo.v 与 tb_fifo.v 两个文件，点击右键得到下拉菜单，选择 Properties…，如图 5.7 所示。

跳出 Project Compiler Settings 对话框，选择 Coverage 活页夹，将 Enable Statement Coverage、Enable Branch coverage、Enable Condition Coverage、Enable Expression Coverage、Enable 0/1Toggle Coverage、Ignore VHDL Subprograms 等选项打上勾，如图 5.8 所示，点击 OK 确定。

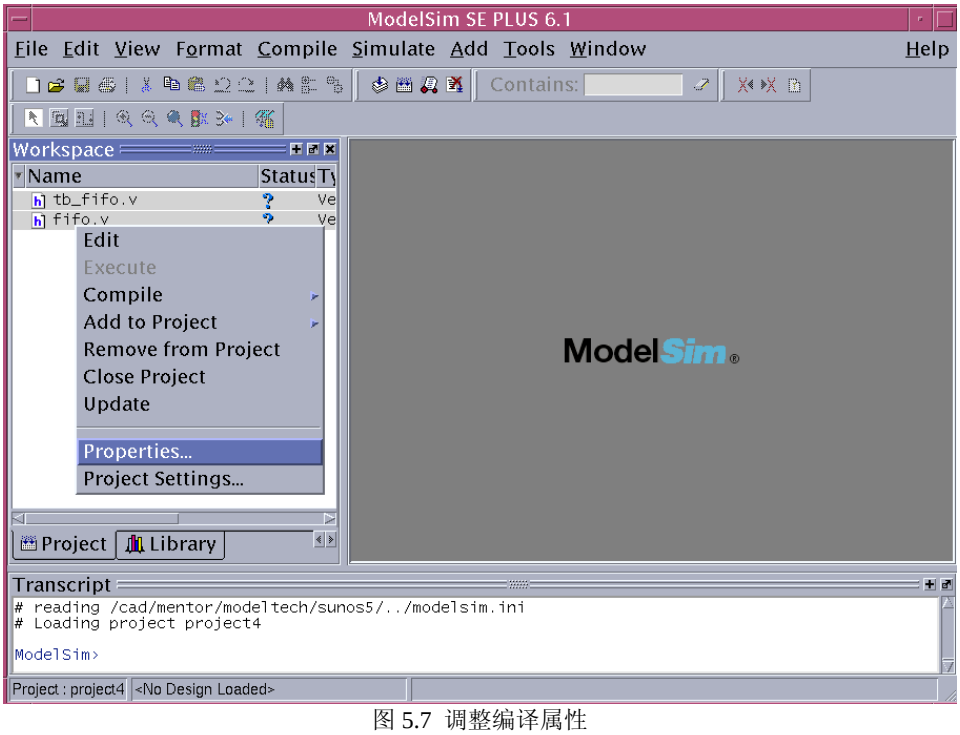

图 5.7 调整编译属性

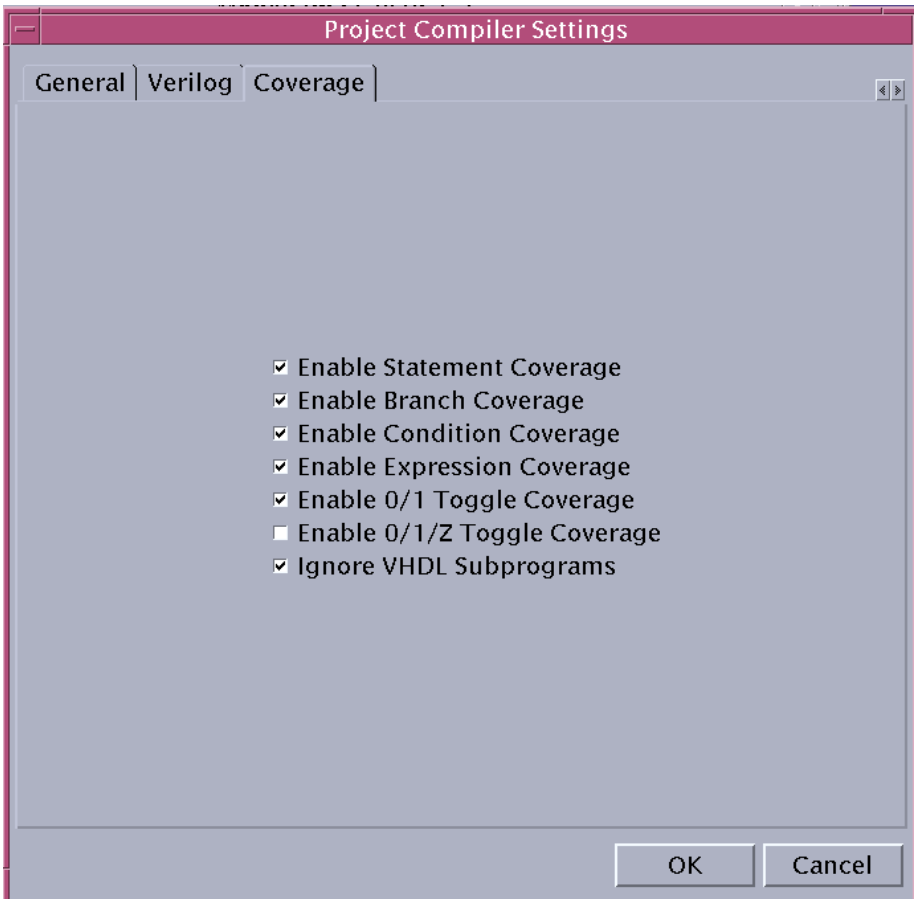

图 5.8 编译覆盖设置

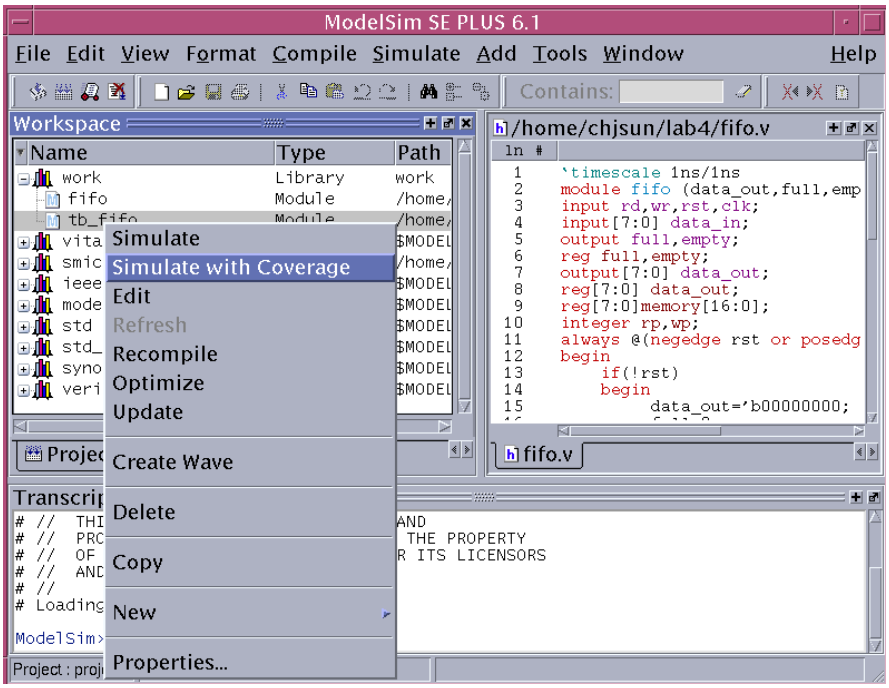

图 5.9 启动仿真

然后与一般的编译方法一样，点击右键选择 Compile All，进行编译。在下一步启动仿

真的时候，如图 5.9 所示：不要选择 Simulate，选择 Simulate with Coverage，此时出现的界面与前述的有所不同，出现了如 Missed Coverage、Details、Instance Coverage 等内嵌的窗口。在启动仿真后，没有执行仿真之前，所有的语句都没有被覆盖到，因此它们都被打上了红色的叉。如图 5.10 所示。

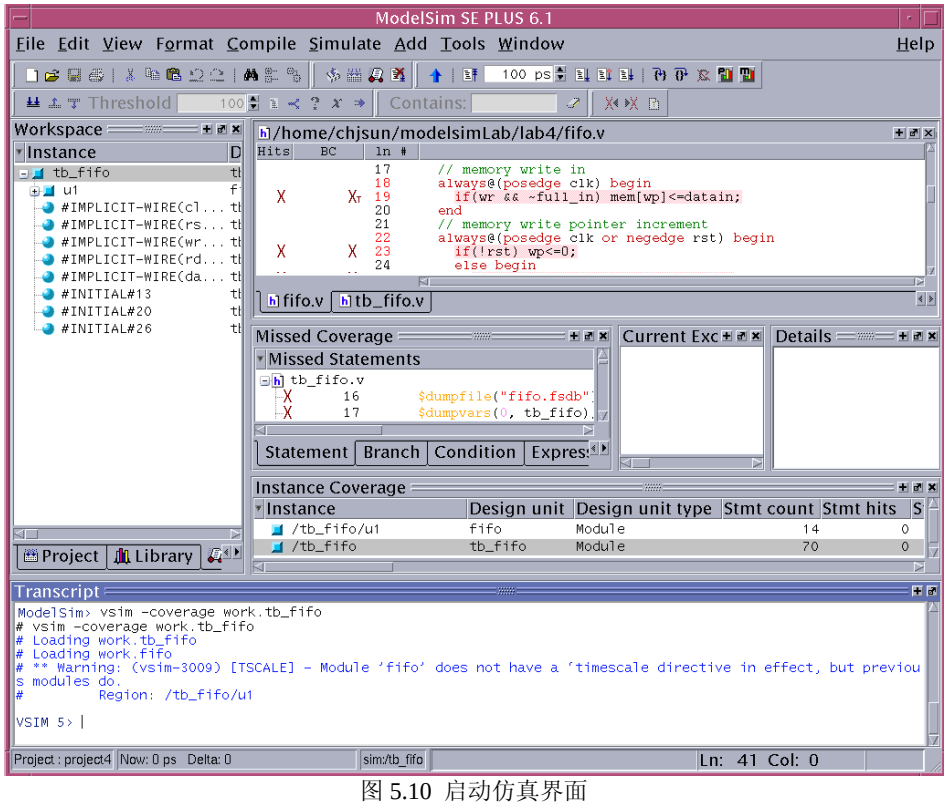

图 5.10 启动仿真界面

执行仿真，执行完以后界面如图 5.11 所示。

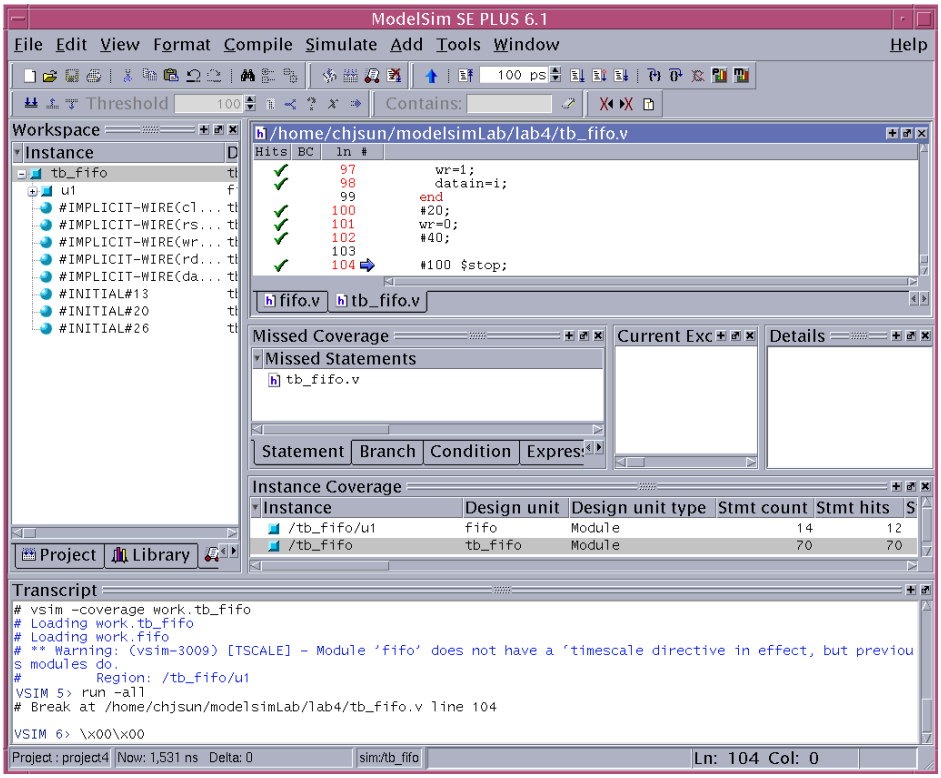

图 5.11 仿真后结果

此时，分别选择 Instance Coverage 框中的 tb_fifo 或者 tb_fifo/u1(也就是被测元件 fifo)两个设计单元，Missed Coverage 框中的内容会相应的发生改变，以显示选中的设计单元中没有被覆盖到的内容。Missed Coverage 框中相应的有 Statement、Branch、Condition、Expression、Toggle 等分支，选择不同的分支便可以对应的显示这一类没有被覆盖到的语句。没有覆盖到的语句在 Missed Statements 框中被打上了红色的 X。在上面的源代码中没有被覆盖到的语句也同样地被打上了红色的 X。源代码中被执行过的语句前被打上了绿色的对号。点击 Missed Coverage 中的语句，Details 框中会显示相应语句的详细的信息。

针对我们的实验内容，我们首先点击 Instance Coverage 框中的 tb_fifo，把 Missed Coverage 单独调出来观察，可以看到 tb_fifo 中所有的语句都被执行过了。因此并没有 Missed Coverage，如图 5.12 所示。

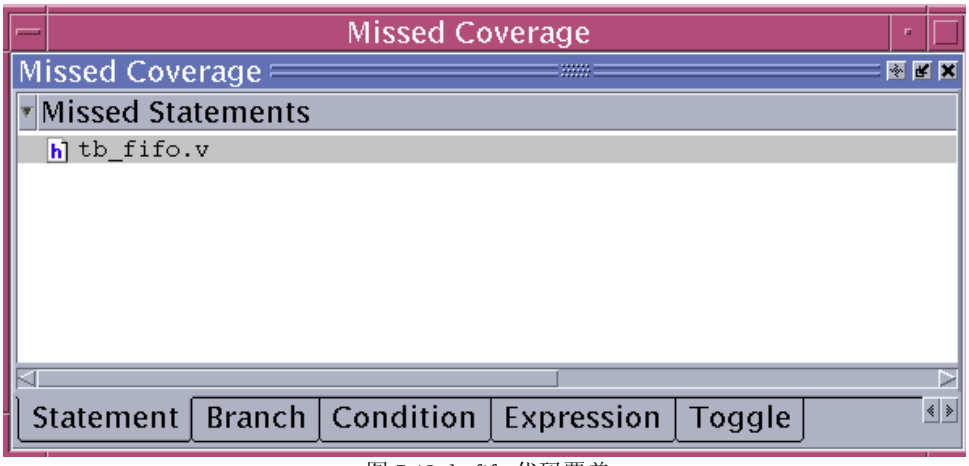

图 5.12 tb_fifo 代码覆盖

选中 Instance Coverage 中的 tb_fifo/u1(也就是被测元件 fifo)，把 Missed Coverage 窗口单独调出来观察，可以看到被测元件 fifo 的源代码中有没有被覆盖到的语句，如图 5.13 所示。

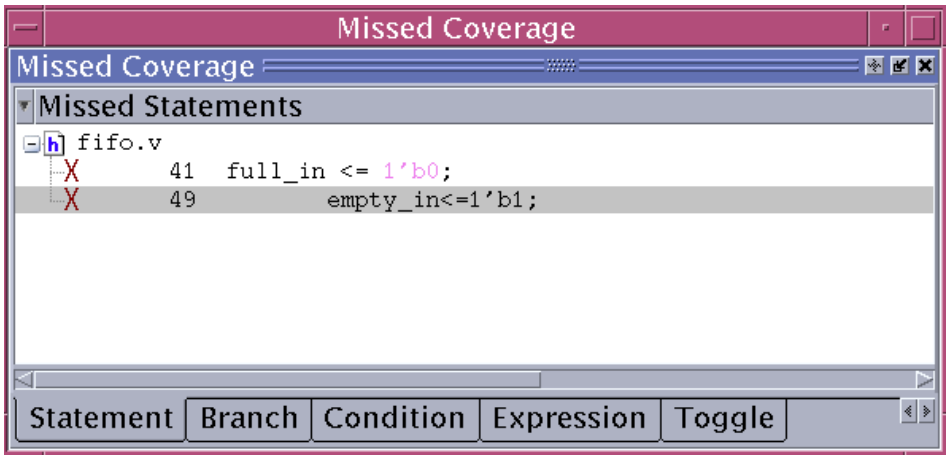

图 5.13 fifo 代码覆盖

分别选中 Statement、Branch、Condition、Expression、Toggle 等活页夹，可以看到各种类型的没有被覆盖到的语句。

同时在 Workspace 中也可以通过拉动滚卷条，观察全部的关于代码覆盖的信息，如图 5.14 所示。

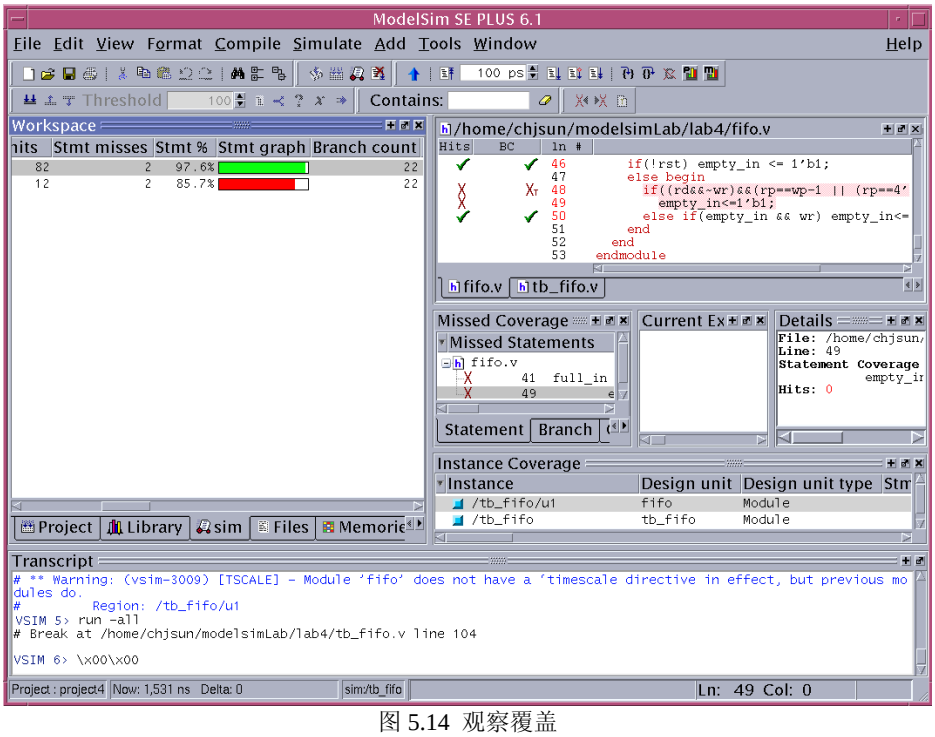

图 5.14 观察覆盖

同学们可以换一个测试激励文件，把 tb_fifo.v 文件换成 tb_fifo_full.v 文件(tb_fifo_full.v 文件也处于目录 lab4 下)。tb_fifo_full.v 测试激励对 fifo.v 的测试代码覆盖可以达到 100%。同学们可以对比一下这两个测试激励的不同。