

第 8 章 使用 ModelSim 进行设计仿真

ModelSim为HDL仿真工具，我们可以利用该软件来实现对所设计的VHDL或Verilog程序进行仿真，支持IEEE常见的各种硬件描述语言标准。可以进行两种语言的混合仿真，但推荐大家只对一种语言仿真。ModelSim常见的版本分为ModelSim XE和ModelSim SE两种，ModelSim版本更新很快，目前最新版本为 5.8 版本，该版本支持VHDL的 2002 标准以及Verilog的 2001 标准，此外，在该版本的Linux、HP和SUN工作站等平台支持VHDL、Verilog和SystemC的混合仿真，但在Windows平台上不支持SystemC的仿真。本章将对ModelSim5.7版本进行介绍，目的是希望看完本章，读者可以简单地使用ModelSim进行仿真，有关更深入地教程，还是参考ModelSim附带的文档。在网址<http://www.model.com/support/>上也可以找到深入的教程，在该页面上注册以后，会在电子邮件中收到发过来的密码，根据邮件地址和密码登陆后，会有一些高级教程和使用要点（Application Notes）下载。

ModelSim5.7SE 版本内部划分为更细的版本编号，从 ModelSim5.7aSE 到 ModelSim5.7gSE 等等。读者可以选择任一版本。另外，如果 ModelSim 是和 ISE 一起使用的话，你需要编译 Xilinx 的一些库文件，这些库文件包括 unisim、simprim、xilinxcorelib、aim、pls、cpld 等，有了这些库文件，可以在 ISE 中生成设计的行为仿真（将设计转换为 RTL 描述后进行的仿真）转换后仿真（将设计转换为 Xilinx 器件的基本模块来实现后进行的仿真）映射后仿真（将设计用 Xilinx 的具体器件的具体模块实现后进行的仿真，仿真中包含了器件的延时，但由于没有布局布线无法提取出互连线的信息，因此仿真中不包含互连线的延时、电容、电阻等信息）布局布线后仿真（将设计对 Xilinx 具体器件进行布局布线后，提取出互连线的信息，进行的仿真包含了器件的延时以及互连线的延时等信息）的各种模型，然后在 ModelSim 中对这些模型进行仿真，由于 ISE 和 ModelSim 已经实现了无缝的连接，在 ISE 中只需设置一个可视的波形文件，然后点几下鼠标就可以实现各种仿真，具体的方法在 ISE 的使用一章中有较为详细的介绍，可以参阅具体的章节。同时，也可以使用 ModelSim 的 XE（Xilinx Edition）版本，由于该版本是 Xilinx 版，其中自然就集成了 Xilinx 的各种库文件，使用时就不需要考虑库文件方面的问题了。对于 ISE 不同版本集成了不同版本的 ModelSim，虽然是集成，还是需要单独进行安装的。对于 ISE6.1 版本配套的 ModelSimXE 版本为 ModelSim5.7cXE。

本章为 ModelSim 的初级教程，读者读完本章可以较为熟练的使用 ModelSim 进行设计仿真，本章没有也不可能涉及 ModelSim 的各个方面，要想全面的掌握 ModelSim 可以参阅软件文档。

8.1 设计准备

在本节中将介绍 ModelSim5.7 的安装，Xilinx 仿真库的设置等方面的内容，不同的版本的安装过程大致相同，在此不再多讲，只是若你安装 XE 版本时，会出现选择是否是免费的 Starter 版本，如果不想购买可以选择该选项，其实，Starter 版本足够完成我们遇见的设计仿真。若你连在互联网上，可以在线申请 License 文件，选取开始->程序->ModelSimXEII5.7c->Submit License Request，会打开一个网页，如果你在 Xilinx 网站上未注册，可以先在线注册，已经注册的话先登录，会出现另一界面，显示你的注册信息，点击 Submit 就可以申请 License 了。过几分钟可以到你的电子邮箱里收取 License 文件。若安装 ModelSim 的机器没有联到互联网，可以点击开始->程序->ModelSimXEII5.7c->License Request Instructions，会打开一个文本文件，安装要求填写并发到指定的邮箱即可，但这种方法又是很难以得到回应。得到 License 文件后，再点击开始->程序->ModelSimXEII5.7c->Licensing Wizard 指定 License 文件即可。对于非 XE 版本的 ModelSim 安装同一般软件的安装。

ModelSim 的各个版本可以从互联网上免费得到，购买的只是 License 文件，ModelSim 的下载地址为 <http://www.model.com/>，进入这个界面可以点击 Download，用户填写完一张表格后可以得到一个小时的下载时间，读者在首页上点击 Evaluations 来得到评估版本的 license 文件。

ModelSim 软件的加密采用的是 Flexlm 方法，Flexlm (Flexible License Manager) 是由 Globetrotter 公司发明的软件加密方法，Globetrotter 公司向软件厂商出售相关开发软件，软件厂商把此加密程序集成到自己的软件中，该方法是目前最为流行的 EDA 软件加密方法，已被 80% 以上的 EDA 软件公司所采用，该方法可以锁定机器的硬盘号，网卡号，使用日期，支持加密狗等，以保护软件的知识产权。打开我们得到的 ModelSim Starter 版本得到的 License.dat 文件可以看到如下内容：

```
FEATURE xe-starter modeltech 2004.12 1-jan-00 uncounted \
```

```
9C0FA6415C9327086559 HOSTID=DISK_SERIAL_NUM=c61e85a ck=244
```

其中，FEATURE 是关键字；xe-starter 是 FEATURE 名，即允许开通的功能；modeltech 是厂商标识；2004.12 是版本标记，可以是时间，也可以是版本号；1-jan-00 为过期时间，如果是 permanent 或 0，表示永不过期，这里的 00 就表示不过期；uncounted 表示使用人数限制；9C0FA6415C9327086559 是根据以上条件和 Flexlm 加密算法计算出的密码；HOSTID=DISK_SERIAL_NUM=c61e85a 表示运行机器的特征，可以指定运行机器的硬盘号、网卡物理地址以及加密狗等，这里指定的是机器的硬盘号，只有硬盘号为 c61e85a 时才可以使⽤。ModelSim 不像有些 EDA 软件需要专门的管理工具来管理 License，大多数 EDA 软件是需要管理工具来管理的，如 PADS、ActiveHDL 等等，这些软件安装时会有安装 License Server 的选项，管理工具可以在控制面板里的 FLEXlm License Manager 进行设置。对于 ModelSim 只需要在环境变量里指明 license 文件的位置即可，运行 ModelSim LicenseWizard 可以自动设置环境变量，若需要手动设置，在 Win95/98 下在 autoexec.bat 中

加入如下一行:SET LM_LICENSE_FILE=c:\flexlm\license.dat, 如果在一台机器上安装了多个这种加密方式的 EDA 软件, 可以用分号隔开多个 license 文件, 如:

```
SET LM_LICENSE_FILE=c:\flexlm\license1.dat ; d:\altera\license2.dat
```

在 Win2000 和 NT 操作系统下, 直接在系统环境变量中设定这些参数。具体方法是在我的电脑上点击右键, 选择属性, 点击高级, 点击环境变量, 再新建一个变量, 变量名为 LM_LICENSE_FILE, 变量值为 license 文件的存放地址。

在使用由 Flexlm 加密的 EDA 文件时, 不要随意更改系统时间, 因为 Flexlm 加密系统会监测系统文件的时间。

另外 ModelSim 的版本很多, 基本的有 PE、LE、SE 版本, PE 为个人版本, 功能最少, 支持的操作系统为 32 位 WINDOWS 98/NT/ME/2000/XP; LE 版本支持的操作系统为 32 位 LINUX; SE 版本为全功能版本, 支持 32 位操作系统 AIX, HP-UX, LINUX + SOLARIS, WINDOWS 98/NT/ME/2000/XP, 64 位操作系统 AIX, LINUX (ITANIUM-2), HP-UX, SOLARIS, 以及 HP-UX, LINUX 等平台。

8.2 菜单和工具栏介绍

这里以 ModelSimXEH5.7c 为例来说明, 本节说明 ModelSim 的菜单和工具栏, 读者有一个初步的了解就可以了。可以通过点击开始->程序->ModelSim XEH5.7c->ModelSim 或点击桌面上的快捷方式来运行该软件, 出现的界面如图 8-2-1 所示。在图的最上端为标题栏; 下面一行为菜单栏; 再下面为工具栏; 左半部分为工作区 (Workspace), 在其中可以通过双击查看当前的工程及对库进行管理; 右半部分为命令窗口区, 在其中出现的命令行及提示信息称为脚本 (Transcript); 最下面一行为状态栏。这里要注意的是, 有些操作是无法通过菜单和工具栏来完成的, 学习 ModelSim 一定要学会使用命令行方式来操作, 常用的命令并不多, 不是很难掌握, 在后续章节将介绍仿真中的一些常见命令。因此, 本节内容读者略读一下就可以了, 实际试一下会更好。

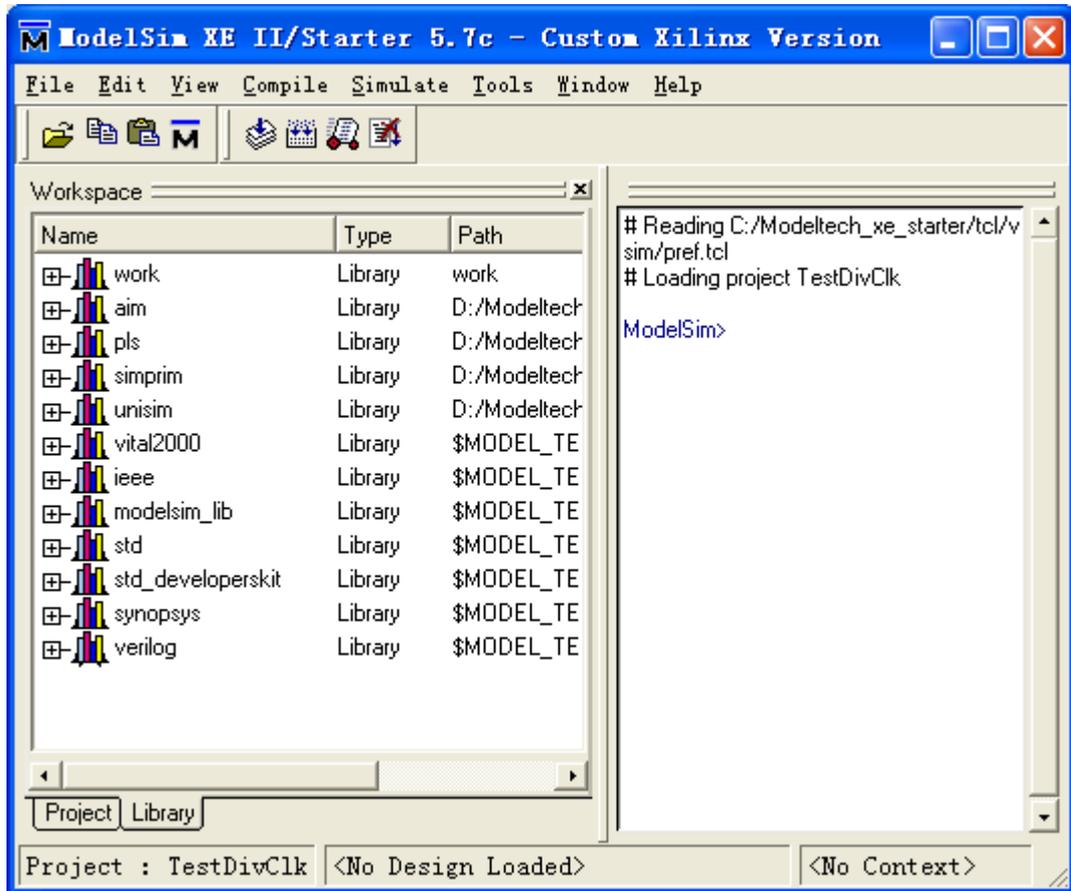


图 8-2-1 ModelSim 界面

8.2.1 标题栏

与一般的 Windows 窗口相同,界面的最上一行为标题栏,显示当前的应用程序的名称,通过点击标题栏的图标  (或 Alt 键 + SpaceBar 空格键) 可以对窗口进行诸如改变窗口大小、移动窗口位置、关闭窗口之类的操作,这些与 Windows 完全相同。

8.2.2 菜单栏

标题栏下方为菜单栏。菜单栏有八个菜单项,分别是:File (文件)、Edit (编辑)、View (视图)、Compile (编译)、Simulate (仿真)、Tools (工具)、Window (窗口)、Help (帮助)。下面分别罗列其具体选项。

1. File (文件) 菜单

文件菜单通常包含了对工程及文件等的操作。ModelSim 的文件菜单包含的命令有:

New (新建), Open (打开), Close (关闭), Import (导入), Save (保存), Delete (删除), Change Directory (更改路径), Transcript (对脚本进行管理), Add to Project (为工程添加文件), Recent Directories (最近几次的工作路径), Recen Projects (最近几次工程), Quit (退出)

(1) 新建文件命令 (File/ New)

单击 File/ New 命令, 将会出现一个子菜单, 共包含四个选项: 单击 Floder (新建文件夹) 后, 会出现对话框, 提示输入新建的文件夹的名字, 即可在当前目录下新建一个文件夹; 单击 Source (新建源文件) 后, 会出现源文件类型的选项 (VHDL, Verilog, Other), 点击可分别新建对应格式的源文件; 单击 Project (新建工程) 后, 会出现对话框, 提示在 Project Name 处输入新建工程的名称, 在 Project Location 处指定新建工程的存放路径, 在 Default Library Name 处指明默认的设计库的名称, 用户设计的文件将编译到该库中; 单击 Library (新建一个库) 后, 会出现对话框, 提示选择 Creat a New library and a logical mapping to it (新建一个库并建立一个逻辑映像) 或 A map to an existing library (新建一个到已存在库的映像), 在 Lirary name 处输入新建库的名称, 在 Library phycial name 处输入存放库的文件名称。

(2) Open (打开文件)

单击会出现子菜单选择打开 File (文件) Project (工程) 及 Dataset (WLF 文件)。

(3) Close (关闭)

单击会出现子菜单选择关闭 Project (工程) 或 Dataset (仿真数据文件)。

(4) Import (导入)

导入新的库, 在进行某些仿真时需要的一些库可以通过该方法导入, 根据提示指定源库路径及目标库路径, 一步步操作完成。注意 ModelSim 安装目录下的 modelsim.ini 文件不能为只读。在该文件中保存了 ModelSim 的一些设置信息, 后续章节将详细讨论该文件的内容及其含义。

(5) Save (保存)

保存当前仿真数据。

(6) Delete (删除)

删除指定的工程, 即删除.mpf 文件, mpf 是 ModelSim 工程的后缀名。

(7) Change Directory (改变路径)

改变当前工作路径, ModelSim 使用的是绝对路径, 而不是相对路径, 这与 ISE 不同, 在 ISE 中, 你可以将你的设计整个目录拷贝到其他任何地方, 只要目录完整, 你可以直接打开工程文件。而在 ModelSim 中, 若将整个目录拷贝到其他地方, 打开工程时其指向仍为原来工程的地址, 可以通过更改路径来设置新的路径。

(8) Transcript (脚本)

单击会出现子菜单选择操作 Save Transcript (保存主窗口中脚本) Save Transcript As (把主窗口中脚本另存为一个新文件) 或 Clear Transcript (清除主窗口中的脚本)。

(9) Add to Project (添加到工程)

单击会出现子菜单选择操作 File (添加文件到当前工程) Simulation Configuration (添

加设定的仿真配置) 或 Folder (添加文件夹)。

(10) Recent Directories (最近几次工作路径)

可以从中选取最近几次的工作路径。

(11) Recen Projects (最近几次工程)

可以打开最近几次的工程。

(12) Quit (退出)

退出 Model Sim.

2. Edit (编辑) 菜单

类似于 Windows 应用程序, 在编辑菜单中包含了对文本的一些常用的操作。

(1) Copy (复制)

复制选中的文档

(2) Paste (粘贴)

把剪切或复制的文档粘贴到当前插入点之前。

(3) Select All (全选)

选中主窗口中所有的抄本文档。

(4) Unselect All (取消全选)

取消已选文本的选中状态。

(5) Find (查找)

在命令窗口中查找字符或字符串。

3. View (视图) 菜单

类似于其他 Windows 应用程序, 视图菜单可以控制在屏幕上显示哪些窗口。

(1) All Windows (所有窗口)

打开所有的 Model Sim 窗口, 你试一下该命令会发现 ModelSim 打开了许多窗口, 包括波形窗口、信号列表窗口、源文件窗口等等。

(2) Dataflow (数据流)

打开 Dataflow 窗口, 在该窗口中显示数据的流向。

(3) List (列表)

打开列表窗口。

(4) Process (进程)

打开过程窗口, 该窗口显示了设计中的进程所在的位置。

(5) Signals (信号)

打开信号窗口。该窗口显示了设计中所有信号的列表

(6) Source (源文件)

打开源文件窗口, 可以在源文件窗口中显示设计中使用的源文件。

(7) Structure (结构)

打开结构窗口, 该窗口以列表方式显示了设计中所有到的结构, 双击某一结构, 可以

查找定义该结构的语句。

(8) Variables (变量)

打开变量窗口，该窗口以列表方式显示了设计中定义的所有变量。

(9) Wave (波形)

打开波形窗口，这是我们仿真时经常需要查看的窗口，在其中显示了输入和输出的波形。

(10) Datasets

打开 Dataset 浏览器来打开、关闭、重命名或激活一个 Dataset。你在使用的时候会发
现没有什么变化，这时候你可以看看 Workspace 窗口下是不是多了一个选项卡。该选项卡
显示的内容与 Structure 窗口显示的完全相同。

(11) Coverage (覆盖率)

查看仿真的代码覆盖率。

(12) Active Processes (活动的进程)

当前正在执行的进程。

(13) workspace (工作区)

打开当前的工作区。

(14) Encoding (编码)

以不同的编码查看。

(15) Properties

显示工作区中选中对象的属性。

4. Compile (编译) 菜单

(1) Compile (编译)

把 HDL 源文件编译到当前工程的工作库中。

(2) Compile Options (编译选项)

设置 VHDL 和 Verilog 编译选项，例如可以选择编译时采用的语法标准等。

(3) Compile All (全编译)

编译当前工程中的所有文件。

(4) Compile Select (编译选中的文件)

编译当前工程中的选中文件。

(5) Compile Order (编译顺序)

设置编译顺序，一般系统会根据设计对 VHDL 自动生成编译顺序，但对于 Verilog 需
要指定编译顺序。

(6) Compile Report (编译报告)

有关工程中已选文件的编译报告。

(7) Compile Summary (编译摘要)

有关工程中所有文件的编译报告。

5. Simulate (仿真) 菜单

这里的编译及运行命令类似于 VC 等高级语言的调试时候的命令。

(1) Simulate (仿真)

装载设计单元。

(2) Simulation Options (仿真选项)

设置仿真选项。

(3) Run (运行)

Run ***ns:在该仿真时间长度内进行仿真。若要改变长度,可在 Simulation Options 中设置或在工具栏中修改;

Run-All (运行所有仿真):进行仿真,直到用户停止它;

Continue (继续):继续仿真;

Run-Next (运行到下一事件):运行到下一个事件发生为止;

Step (单步):单步仿真;

Step-Over:仿真至子程序结束;

Restrat:重新开始仿真,重新加载设计模块,并初始化仿真时间为零。

(4) Break (停止)

停止当前的仿真。

(5) End Simulation (结束仿真)

结束当前仿真。

6. Tool (工具) 菜单

(1) Waveform Compare (波形比较)

在子菜单中有具体进行波形比较的命令。

(2) Coverage (覆盖率)

测试仿真的代码覆盖率,所谓代码覆盖率是指仿真运行到当前已运行的代码占有所有代码的比例,当然是越接近 100%越好。

(3) Breakpoints (断点设置)

单击此选项出现断点设置对话框,设置断点。

(4) Execute Macro (执行宏文件)

所谓的宏文件就是保存后的脚本,脚本保存起来,以后可以利用该命令来重新执行。

(5) Options (选项)

Transcript File:设置脚本文件的保存。

Command History:命令历史。

Save File:保存脚本文件。

Saved Lines:限制脚本文件的行数。

Line Prefix:设置每一行的初始前缀。

Update Rate:设置状态条的刷新频率。

ModelSim Prompt:改变 ModelSim 的命令提示符。

VSIM Prompt: 改变 VSIM 的命令提示符。

Paused Prompt: 改变 Paused 的命令提示符。

HTML Viewer:设置打开在线帮助的文件。

(6) Edit Preferences (编辑参数选取):
设置编辑参数。

(7) Save Preferences (保存参数选取):
设置保存用的参数。

7. Window (窗口) 菜单:

(1) Initial Layout (初始化版面)
恢复所有窗口到初始时的大小和位置。

(2) Cascade (层叠)
使所有打开的窗口层叠。

(3) Tile Horizontally (水平平铺)
水平分隔屏幕, 显示所用打开的窗口。

(4) Tile Vertically (垂直平铺)
垂直分隔屏幕, 显示所用打开的窗口。

(5) Layout Style (版面格式)
Default (默认格式): 与 Initial Layout 格式相同;
Classic (经典格式): 采样低于 5.5 版本的格式;
Cascade: 与 Cascade 格式相同;
Horizontally: 与 Tile Horizontally 格式相同;
Vertically: 与 Tile Vertically 格式相同。

(6) Icon Children
除了主窗口之外的其他窗口缩为图标。

(7) Icon All
将所有窗口缩为图标。

(8) Deicon All
将所用缩为图标的窗口还原。

8. Help (帮助) 菜单

(1) About ModelSim
显示 ModelSim 的版本、版权等信息。

(2) Release Notes
显示 ModelSim 的版本发布信息。

(3) Welocme Menu
显示欢迎画面。

(4) PDF Documentation

在子菜单中可以选择 ModelSim 的 PDF 文档。

SE HTML Documentation：ModelSim 的超文本文档。

(5) Tcl Help：Tcl 帮助文档。

Tcl 是 Tools Command Language 的缩写，它是一种可扩充的命令解释语言，具有与 C 语言的接口和命令的能力，应用非常广泛，这方面也有专门的书籍。

(6) Tcl Man Pages：Tcl 主页面。

(7) Technotes：技术文档。

8.2.3 工具栏

ModelSim 的工具栏如图 8-2-2 所示。从左到右依次为：打开、复制、粘贴、如何更新 ModelSim、编译选定、编译全部、仿真、停止仿真、重新开始仿真、仿真步长、运行一步、继续运行、运行所有、单步执行、主程序的单步执行。



图 8-2-2 ModelSim 的工具栏

8.2.4 状态栏

ModelSim 的状态条如图 8-2-3 所示，其中 Project 后面为当前工程的名称，Now 后面为当前仿真时间。最右边的为选定的仿真结构中变量。



图 8-2-3 ModelSim 的状态栏

8.3 从一个例子开始学习 ModelSim

前面两节简要介绍了有关 ModelSim 的安装以及用户界面的功能，初学者可能会觉得又很多名词看不懂，这一节我们来从一个简单的例子学习 ModelSim 的简单的使用。学完本节你会发现 ModelSim 不仅好用，而且易用。

这一节我们使用的例子是一个分频电路的设计。所谓分频电路是将较高频率的时钟分频，得到较低频率的时钟，分频电路的使用较为广泛，例如，我们要编写一个显示时间的电路就需要一个分频器，将晶振的频率分频得到 1Hz 的时钟信号。分频有几种方法，对于较为规则的分频，如 2 分频、4 分频、8 分频等可以调用 ISE 本身的库函数来实现，对于较不规则的分频，我们也有两种方法，一种是利用计数器的某一位来作为分频输出，一种是

计数器计数到某一数值时，分频时钟信号翻转来实现分频。两种方法的可以从其仿真结果得到。之所以选择这个例子，是因为这里例子有实用价值并且设计本身简单，仿真也较为简单。好了，下面我们开始我们的设计。

8.3.1 图形界面对设计进行仿真

考虑到初学者，这里的讲述较为详细，初学者可以按照如下步骤开始：

- 1、 运行 ModelSim，方法是点击开始->程序->ModelSim XE II 5.7c->ModelSim 或双击桌面上的快捷方式，会出现如图 8-3-1 所示的界面，如果上一次使用 ModelSim 建立过工程，这时候会自动打开上一次所建立的工程；

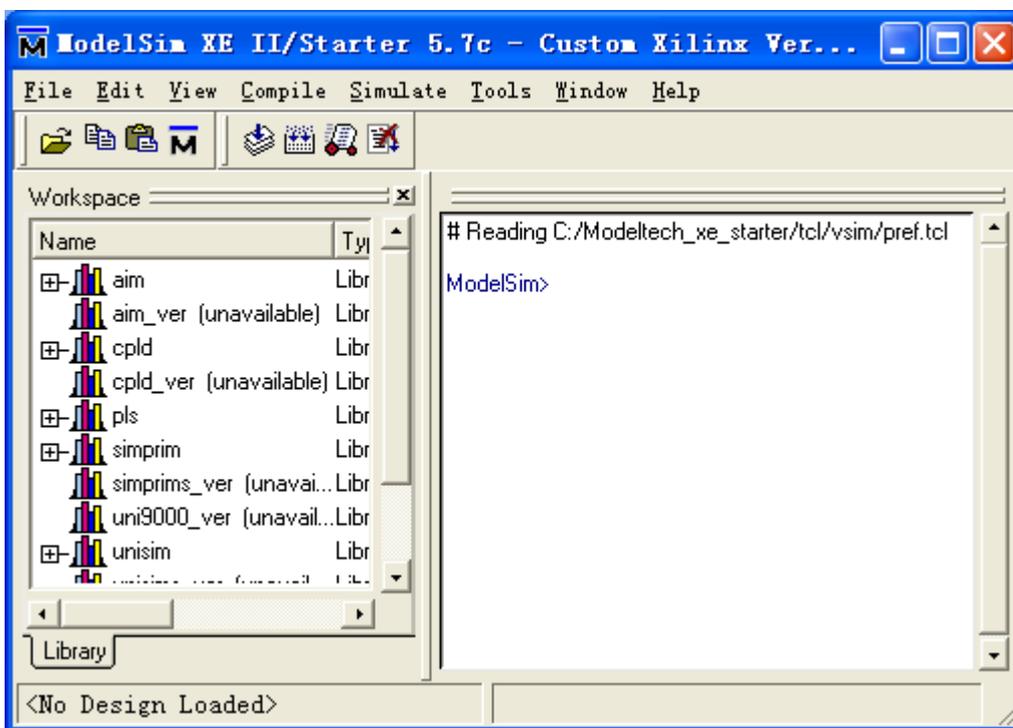


图 8-3-1 ModelSim 开始界面

- 2、 点击 File->New->Project，会出现如图 8-3-2 所示的界面，在 Project Name 中我们输入建立的工程名字为 DivClkSimu，在 Project Location 中输入工程保存的路径为 D:/yuProj/modelsim/DivClk，注意 ModelSim 不能为一个工程自动建立一个目录，这里我们最好是在 Project Location 中输入路径来为工程建立目录，在 Default Library Name 中为我们的设计编译到哪一个库中，这里我们使用默认值，这样，在我们编译设计文件后，在 Workspace 窗口的 Library 中就会出现 work 库。这里我们输入完以后，点击 OK；

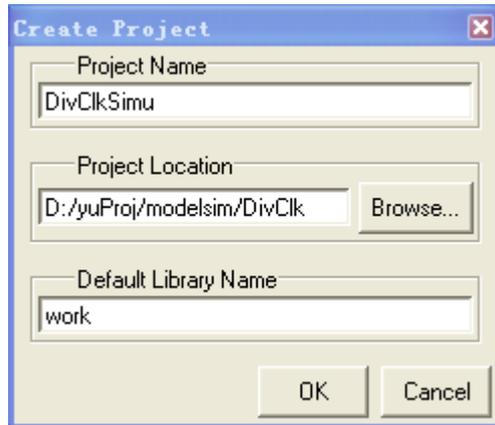


图 8.3.2 新建工程窗口

- 3、 这时有对话框如同 8-3-3 所示，提示我们给定的工程路径不存在，是否建立该路径，我们的目的就是为工程建立一个新目录，因此，点击确定；



图 8-3-3 确认建立新的工程目录

- 4、 这时候出现如同 8-3-4 所示的界面，可以点击不同的图标来为工程添加不同的项目，点击 Create New File 可以为工程添加新建的文件，点击 Add Existing File 为工程添加已经存在的文件，点击 Create Simulation 为工程添加仿真，点击 Create New Folder 可以为工程添加新的目录。这里我们点击 Create New File；



图 8-3-4 为工程添加项目

- 5、 出现界面如图 8-3-5 所示，我们在 File Name 中输入 DivClkHDL 作为文件的名称，Add file as type 为输入文件的类型为 VHDL、Verilog、TCL 或 text，这里我们使用默认设置 VHDL，Folder 为新建的文件所在的路径，Top Level 为在我们刚才所设定的工程路径下。点击 OK；并在 Add items to the Project 窗口点击 Close 关闭该窗口；

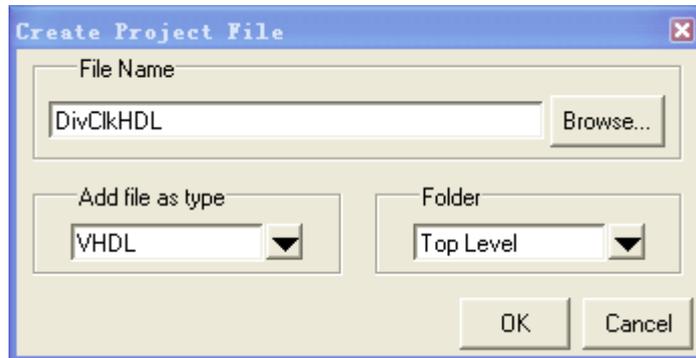


图 8-3-5 为工程添加新文件

- 6、 这时候在 Workspace 窗口中出现了 Project 选项卡，其中有 DivClkHDL.vhd，其状态栏有一个问号，表示未编译，我们双击该文件，这时候出现窗口 edit-DivClkHDL.vhd 的编辑窗口，在其中我们输入我们的设计文件如下：

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_ARITH.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL;  
  
entity divclk1 is
```

```

    Port ( clk : in std_logic;
          divclk : out std_logic);
end divclk1;

architecture Behavioral of divclk1 is
    signal counter : std_logic_vector(4 downto 0):="00000";
    signal tempdivclk: std_logic:='0';
begin
    process(clk)
    begin
        if clk'event and clk='1' then
            if(counter>="11000") then
                counter<="00000";
                tempdivclk<=not tempdivclk;
            else
                counter<=counter+'1';
            end if;
        end if;
    end process;
    divclk<=tempdivclk;
end Behavioral;

```

- 7、 点击 File->Save , 并退出该窗口 (File->Close);
- 8、 在 Workspace 窗口的 DivClkHDL.vhd 上点击右键 , 选择 Compile->Compile All , 如同 8-3-6 所示 ;

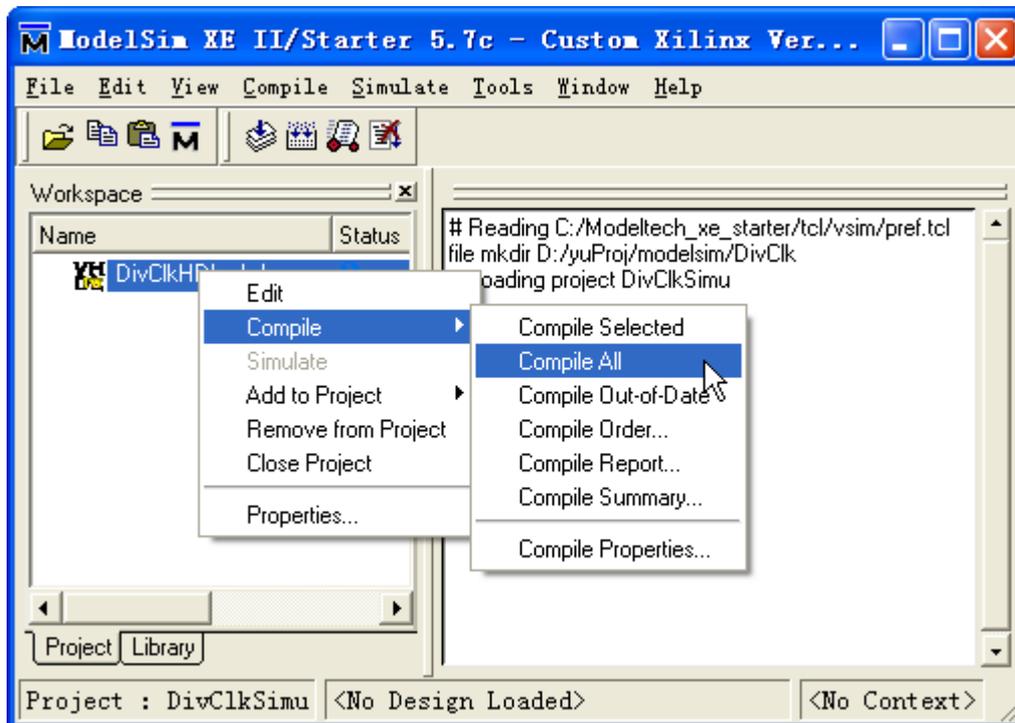


图 8-3-6 编译设计中的文件

- 9、 在脚本窗口中将出现一行绿色字体 Compile of DivClkHDL.vhd was successful.，说明文件编译成功，在该文件的状态栏后有一绿色的对号，表示编译成功；
- 10、 下面我们开始仿真，点击菜单 Simulate->Simulate，会出现如同 8-3-7 所示的界面，我们展开 Design 选项卡下的 work 库，并选中其中的 behavioral，这是在 Simulate 中出现了 work.divclk1(behavioral)表示我们所要仿真的对象，Resolution 为仿真的时间精度，这里我们使用默认值，点击 OK；

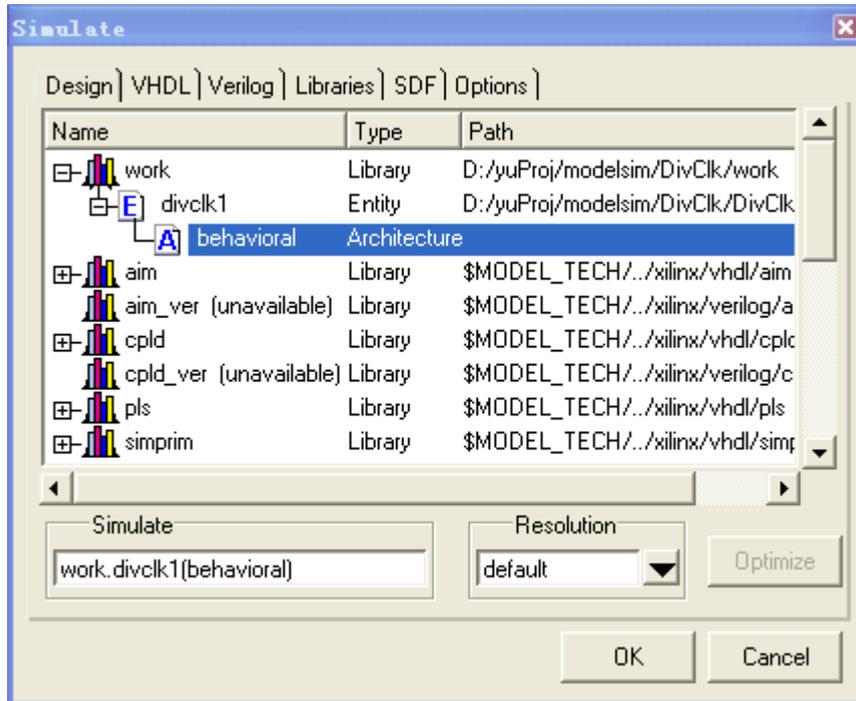


图 8-3-7 选择仿真对象

- 11、 为了观察波形窗口，我们点击菜单 View->Wave；
- 12、 这时候出现的 Wave 窗口为空，里面什么都没有，我们要为该窗口添加我们需要观察的对象，首先在主窗口而不是波形窗口中点击 View->Signals 打开信号列表窗口如图 8-3-8 所示，在改窗口中点击 Add->Wave->Signals in Design，这时候在波形窗口中就可以看到这些信号了；

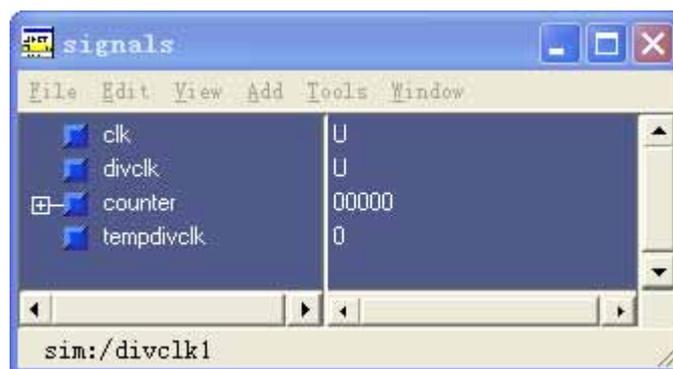


图 8-3-8 信号列表窗口

- 13、 下面我们就开始仿真了，在主窗口中输入命令对信号进行驱动，首先我们为时钟信号输入驱动：force clk 0 0,1 10000 -r 20000 其中 force 为命令，clk 表

8.3.2 使用命令行方式对设计进行仿真

在这一小节里我们主要通过主窗口中输入命令行来进行仿真，并不是所有的操作都是用命令行方式来实现的。我们所设计的例子是另外一种分频方式，即使用计数器的某一位作为分频得到的时钟。我们可以按照如下步骤来实现。

- 1、新建工程 DivClk2Proj，打开 ModelSim，点击 File->New->Project，在工程名中输入 DivClk2Proj，在工程保存路径中输入 D:/yuProj/modelsim/DivClk2，点击 OK，并在随后出现的对话框中点击确定来确认建立该目录；
- 2、在随后出现的 Add items to the Project 窗口中，点击 Create New File 出现 Create Project File 窗口，我们点击 Browse 找到我们刚刚建立的文件夹，并输入文件名，之后，在 File Name 框中应为 D:/yuProj/modelsim/DivClk2/DivClk2HDL，选择 Add file as type 为 VHDL，Folder 为 Top Level，点击 OK，并点击 Add items to the Project 窗口中的 Close 来将其关闭；
- 3、在主窗口中双击 Workspace 窗口中 DivClk2HDL.vhd，出现编辑窗口，在窗口中输入如下代码：

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity divclk1 is
    Port ( clk : in std_logic := '0';
          divclk : out std_logic);
end divclk1;

architecture Behavioral of divclk1 is
    signal counter : std_logic_vector(5 downto 0) := "000000";
begin
    process(clk)
    begin
        if clk'event and clk='1' then
            if(counter>="110001") then
                counter<="000000";
            else
                counter<=counter+'1';
            end if;
        end if;
    end process;
    divclk<=counter(5);
end Behavioral;
```

- 4、保存该文件并编译，方法同 8.3.1，在该文件上点击右键选择 Compile ->Compile All;
- 5、运行仿真，在主窗口输入命令：vsim work.divclk1，注意此处的 divclk1 表示设计中的实体名；
- 6、以下的仿真同 8.3.1 节，为时钟信号添加驱动，输入命令：force clk 0 0,1 10000 -r 20000，将仿真时钟设为 50MHz；
- 7、打开波形窗口，输入命令：view wave，这时会看到空的波形窗口已经打开；
- 8、为波形窗口添加信号，输入命令：add wave -hex *，这里的*表示添加设计中所有的信号，-hex 表示以十六进制来表示波形窗口中的信号值；
- 9、开始仿真，输入命令，run 3us，这时候在波形窗口中出现仿真波形，调整窗口大小，并添加鼠标线，得到如图 8-3-10 所示的波形窗口；

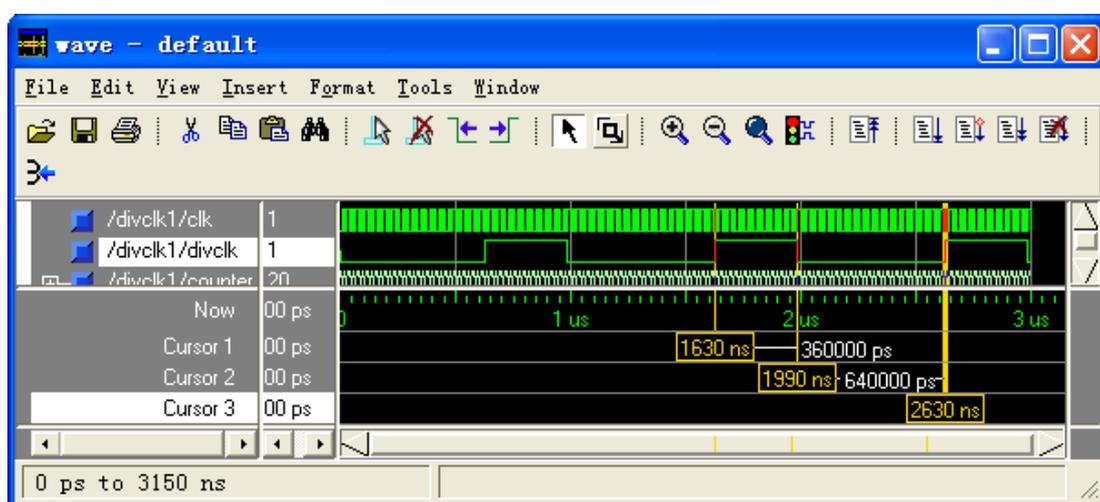


图 8-3-10 仿真波形

- 10、退出仿真，输入命令：quit -sim。
- 11、仿真结果分析，这里我们的输入时钟为 50MHz，周期为 20ns，分频得到的时钟周期为 1us，占空比不为 1。使用时可以调整分频语句 if(counter>="110001") then 中的值及位宽来调整分频后的时钟频率。设我们需要从周期为 T (ns) 的时钟得到周期为 X (ns) 的脉冲，可以用如下的方法计算出此处应有的值： $(X/T)-1$ ，例如此处我们要从周期为 20ns 的时钟得到周期为 1000ns (1us) 的脉冲， $(X/T)-1=(1000/20)-1=49=(110001)_{\text{Bin}}$ 因此可以得到该式中的值。

8.4 使用 Testbench 和 TEXTIO 对设计进行仿真

在上一节中，我们对使用 ModelSim 进行设计仿真有了一定的认识，这一节我们使用较为高级的仿真方式：TestBench 和 TEXTIO 来进行设计仿真。TestBench 可以理解为一个

平台，该平台包含待仿真的模块，具体一点是 TestBench 为一个电路板，在该电路板中包含了我们设计的用 HDL 语言描述的电路，这块电路板与外界没有任何的接口，其功能仅仅是仿真测试我们设计电路，我们可将设计的电路中的端口的信号描绘出来，进行仿真分析。也许看到这里你还不明白，这不要紧，看完后面一个例子就很清楚了。TEXTIO 是 VHDL 标准库 STD 中的一个程序包 (Package)，它提供了 VHDL 与磁盘文件直接访问的桥梁，我们可以利用它来读取或写入仿真数据到磁盘中的文件，TEXTIO 的使用是通过 TestBench 来进行的，即我们在 TestBench 中可以调用 TEXTIO 进行仿真，下面我们就介绍两者的使用。

8.4.1 使用 TestBench 对设计进行仿真

这里我们先看一个例子，我们的设计同上一节，即设计一个分频模块，其源代码如 8.3.2 节所述。这里我们为其编写一个 TestBench，代码如下：

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY divclk1_tb IS
END divclk1_tb;

ARCHITECTURE behavior OF divclk1_tb IS

    COMPONENT divclk1
    PORT(
        clk : IN std_logic;
        divclk : OUT std_logic
    );
    END COMPONENT;
    SIGNAL clk : std_logic := '0';
    SIGNAL divclk : std_logic;

    BEGIN
        uut: divclk1 PORT MAP(
            clk => clk,
            divclk => divclk
        );
        clk<= not clk after 10 ns;
    END;
```

下面我开始在 ModelSim 中使用 TestBench 对设计进行仿真；

- 1、打开 ModelSim，新建工程 TestBenchTest；
- 2、将 8.3.2 节的分频模块设计文件添加到当前工程中，如果你不知道怎么添加，方法

是在 Add items to the Project 窗口中点击 Add Existing Files 找到 8.3.2 节中的文件，拷贝到当前目录即可；

- 3、新建 TestBenchFile 文件，并将上述源代码添加进去保存，新建文件的方法是 File->New->Source->VHDL；
- 4、编译文件，方法是点击菜单 Compile ->Compile All;
- 5、如果代码与本书一样，且 ModelSim 的 License 正确的，应该编译没有问题，下面开始仿真；
- 6、在命令窗口中输入 vsim work.divclk1_tb ,注意与 8.3.2 不同之处在于这里的仿真对象为 divclk1_tb，它是 TestBenchFile 的实体名；
- 7、在命令窗口中输入 view wave，打开波形文件窗口；
- 8、在命令窗口中输入 add wave -hex *，将信号添加到波形文件中；
- 9、在命令窗口中输入 run 3us，在波形窗口中可以得到波形，添加鼠标线并调整大小可以得到如同 8.4.1 所示的波形；

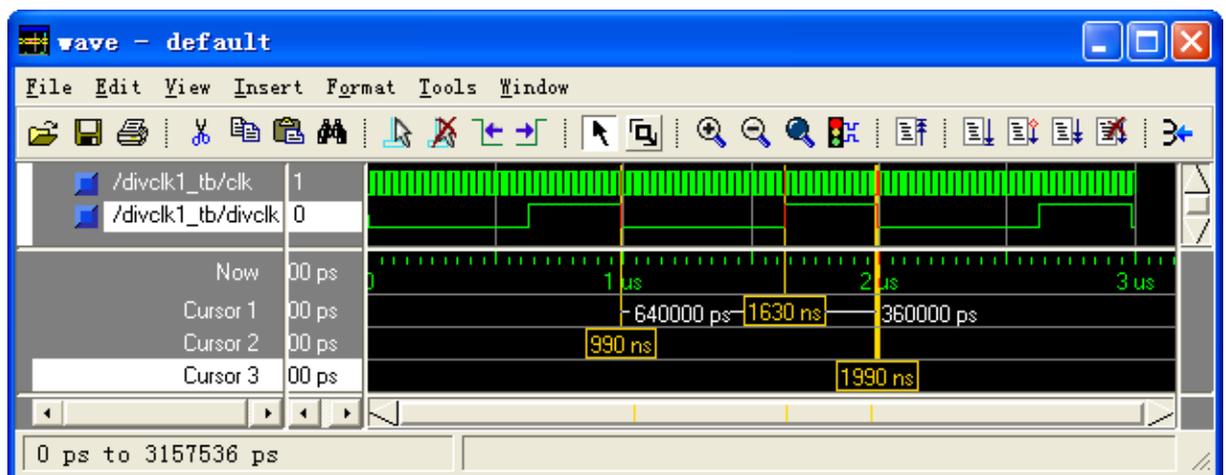


图 8.4.1 仿真结果波形

分析：

从仿真过程可以看出，TestBench 为波形输入自动添加了驱动，因此，在仿真时候就不需要再为信号进行驱动了。在我们这个例子中，并没有省很多事情，因为我们的设计输入只有一个时钟，但如果我们设计一个两个四位数相加的加法器，如果考虑到所有可能的输入一个个地输入会很麻烦，这时候我们可以利用 TestBench 来进行设置，我们在后面会举一个这样的例子。另外，我们要注意到 TestBench 文件中的实体与外界没有任何接口相连，因此：

```
ENTITY divclk1_tb IS
END divclk1_tb;
```

定义的实体中没有端口，我们把所要仿真的对象当成一个部件在 TestBench 中来使用，以实现 TestBench 与仿真对象之间的联系。

8.4.2 TEXTIO 介绍

TEXTIO 是 VHDL 标准库 STD 中的一个程序包 (Package)。在该包中定义了三个类型：LINE 类型、TEXT 类型以及 SIDE 类型；一个子类型 (subtype) WIDTH。此外，在该程序包中还定义了一些访问文件所必须的过程 (Procedure)。简单描述如下：

1. 类型的定义如下：

- type LINE is access string;

定义了 LINE 为存取类型的变量，它表示该变量是指向字符串的指针，它是 TEXTIO 中所有操作的基本单元，读文件时，先按行 (LINE) 读出一行数据，再对 LINE 操作来读取各种数据类型的数据；写文件时，先将各种的数据类型组合成 LINE，再将 LINE 写入文件。在用户使用时，必须注意只有变量才可以是存取类型的，而信号则不能是存取类型的。例如，我们可以定义

```
variable DLine : LINE;
```

但不能定义成：

```
signal DLine : LINE;
```

- type TEXT is file of string;

定义了 TEXT 为 ASCII 文件类型。定义成为 TEXT 类型的文件是长度可变的 ASCII 文件。例如在 TEXTIO 中定义了两个标准的文本文件。

```
file input : TEXT open read_mode is "STD_INPUT";
```

```
file output : TEXT open write_mode is "STD_OUTPUT";
```

定义好以后，就可以通过文件类型变量 input 和 output 来访问其对应的文件 STD_INPUT 和 STD_OUTPUT。

需要注意的是 VHDL'87 和 VHDL'93 在使用文件方面有较大的差异。在编译时注意选中对应的标准。

- type SIDE is (right, left);

定义了 SIDE 类型。表示定义了一个名为 SIDE 的数据类型，其中只能有两种状态，即 right 和 left，right 和 left 表示将数据从左边还是右边写入行变量。该类型主要在 TEXTIO 程序包包含的过程 (Procedure) 中使用。

- subtype WIDTH is natural;

定义 WIDTH 为自然数的子类型。所谓子类型表示其取值范围是父类型的范围的子集。

2. 过程 (Procedure) 的定义如下：

TEXTIO 提供了基本的用于访问文本文件的过程。类似于 C++，VHDL 提供了重载功能。即完成相近功能的不同的过程可以有相同的过程名，但其参数列表不同，或参数类型不同或参数个数不同。

TEXTIO 提供的基本过程有：

- procedure READLINE(文件变量;行变量);

用于从指定文件读取一行数据到行变量中。

- procedure WRITELINE(文件变量;行变量);

用于向指定文件写入行变量所包含的数据。

- procedure READ(行变量;数据类型);

用于从行变量中读取相应的数据类型的数据。

根据参数数据类型及参数个数的不同,有多种重载方式,TEXTIO 提供了 bit、bit_vector、BOOLEAN、character、integer、real、string、time 数据类型的重载。同时,提供了返回过程是否正确执行的 BOOLEAN 数据类型的重载。例如,读取整数的过程为

```
procedure READ(L:inout LINE; VALUE: out integer; GOOD : out BOOLEAN);
```

其中,GOOD 用于返回过程是否正确执行,若正确执行,则返回 TRUE。

- procedure WRITE(行变量;数据变量;写入方式;位宽);

该过程将数据写入行变量。其中写入方式表示写在行变量的左边还是右边,且其值只能为 left 或 right,位宽表示写入数据时占的位宽。例如:

```
write(OutLine,OutData,left,2);
```

表示将变量 OutData 写入 LINE 变量 OutLine 的左边占 2 个字节。

8.4.3 TEXTIO 在仿真中的应用

下面以一个简单的 8 位加法器来说明 TEXTIO 的使用。输入数据为两个 8bit 的有符号数,输出为 9bit 的有符号数,以防止溢出。在编写加法器的描述文件时,首先要对两个数进行位的扩展,再进行加法运算。在编写测试文件时,要注意读入数据与得到结果之间相差一个时钟周期,因此,需要在读出的结果与计算的结果之间需要插入一个时钟周期的等待。仿真步骤如下:

1. 生成输入及预定结果文件的 C++ 程序

我们可以使用 VC++、Matlab 等高级软件工具编写生成输入和预定结果文件的程序,由于设定输入为 8 位有符号数,因此,其范围为[-127,127]。C++ 程序如下:

```
#include "iostream.h"
#include "fstream.h"
void main(void)
{
    int i,j;
    ofstream fsIn("d:\\yuproj\\modelsim\\TextioTest\\TestData.dat");
    ofstream fsOut("d:\\yuproj\\modelsim\\TextioTest\\Result.dat");
    for(i=-127;i<128;i++)
    {
        for(j=-127;j<128;j++)
        {
            fsIn<<i<<" "<<j<<endl;
            fsOut<<i+j<<endl;
        }
    }
}
```

```

    }
}
fsIn.close();
fsOut.close();
}

```

在程序中，使用了 C++ 类库 `iostream.h` 和 `fstream.h`。主要使用了 “<<” 的输出功能。读者可以参考对应的 C++ 书籍。运行该程序可以在规定的目录下生成 `TestData.dat` 和 `Result.dat` 两个文本格式的文件。注意，一行输入多个数据时，之间以空格隔开即可。

2. 新建 ModelSim 工程

打开 ModelSim，新建工程 `Textio`，其存放目录为 `d:\yuproj\modelsim\TextioTest`；

3. 添加设计源文件

新建设计文件 `NineBitsAdder`，新建文件的方法是 `File->New->Source->VHDL`，源代码如下：

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity Add2In is
    port(
        D1 : in std_logic_vector(7 downto 0);
        D2 : in std_logic_vector(7 downto 0);
        Q  : out std_logic_vector(8 downto 0);
        Clk : in std_logic);
end Add2In;

architecture A_Add2In of Add2In is
begin
    process(Clk)
    begin
        if Clk = '1' and Clk'event then
            Q <= (D1(D1'left) & D1) + (D2(D2'left) & D2);

            end if;
        end process;
    end A_Add2In;

```

在进行加法前，首先进行位的扩展，再进行加法运算。在时钟的上升沿完成加法运算。

4. 添加测试文件

新建 `TestBench` 文件 `TestBench.vhd`，新建文件的方法是 `File->New->Source->VHDL`，代码如下：

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use std.TEXTIO.all;
```

```
entity tb is
end tb;
```

```
architecture a_tb of tb is
```

```
    component Add2In
```

```
    port(    D1 : in std_logic_vector(7 downto 0);
           D2 : in std_logic_vector(7 downto 0);
           Q  : out std_logic_vector(8 downto 0);
           Clk : in std_logic);
```

```
end component;
```

```
    signal D1 : std_logic_vector(7 downto 0):=(others=>'0');
```

```
    signal D2 : std_logic_vector(7 downto 0):=(others=>'0');
```

```
    signal Q  : std_logic_vector(8 downto 0);
```

```
    signal Clk : std_logic:='0';
```

```
    signal Dlatch : boolean :=false;
```

```
    signal SResult : integer;
```

```
begin
```

```
    dut : Add2In
```

```
    port map(    D1=>D1,
               D2=>D2,
               Q => Q,
               Clk => Clk);
```

```
    Clk<=not Clk after 20 ns;
```

```
process
```

```
    file InputD : text open read_mode is "TestData.dat";
```

```
    variable DLine : LINE;
```

```
    variable good   : Boolean;
```

```
    variable Data1  : integer;
```

```
    variable Data2  : integer;
```

```
begin
```

```
    wait until Clk='1' and Clk'event;
```

```
        readline(InputD,DLine);
```

```
        read(DLine,Data1,good);
```

```

        read(DLine,Data2,good);
        if ( good )then
            D1 <= CONV_STD_LOGIC_VECTOR(Data1,8);
            D2 <= CONV_STD_LOGIC_VECTOR(Data2,8);
        else
            assert false report "End of Reading Input File!"
                severity error;
        end if;

    end process;

process
    file InputR : text open read_mode is "Result.dat";
    variable RLine : LINE;
    variable Result : integer;
begin
    wait until Clk='1' and Clk'event;
    Dlatch<=true;
    if Dlatch then
        readline(InputR,RLine);
        read(RLine,Result);
        SResult<=Result ;
        if SResult /=Q then
            assert false report "Two values are different"
                severity warning;
        end if;
    end if;
end process;

end a_tb;

```

5. 编译

下面开始仿真，先编译所有文件，方法是点击菜单 Compile ->Compile All;

6. 装载设计

在命令窗口中输入 vsim work.tb ;

7. 打开波形窗口

在命令窗口中输入 view wave , 打开波形文件窗口 ;

8. 添加信号到波形窗口中

在命令窗口中输入 add wave -dec * , 将信号添加到波形文件中 ;

9. 运行仿真

在命令窗口中输入 `run -all` 该命令将使仿真一直运行下去；

10. 仿真结果

仿真结束后在主界面窗口中将会看到如下信息：

```
# ** Fatal: (vsim-3551) TEXTIO : Read past end of file "TestData.dat".  
#   Time: 2601020 ns Iteration: 0 Process: /tb/line__34 File:  
D:/yuProj/modelsim/TextioTest/TestBench.vhd  
# Fatal error at D:/yuProj/modelsim/TextioTest/TestBench.vhd line 41  
#
```

表示在读完 `TestData.dat` 后，因读空出现错误。其中没有出现我们程序中所设定的 `warning`。表示加法器的仿真结果与高级软件得到的预定结果相符合。我们可以改进程序，在其中加入 `ENDFILE ()` 函数来判断是否读取到文件的结尾，仿真结果如图 8-4-2 所示；

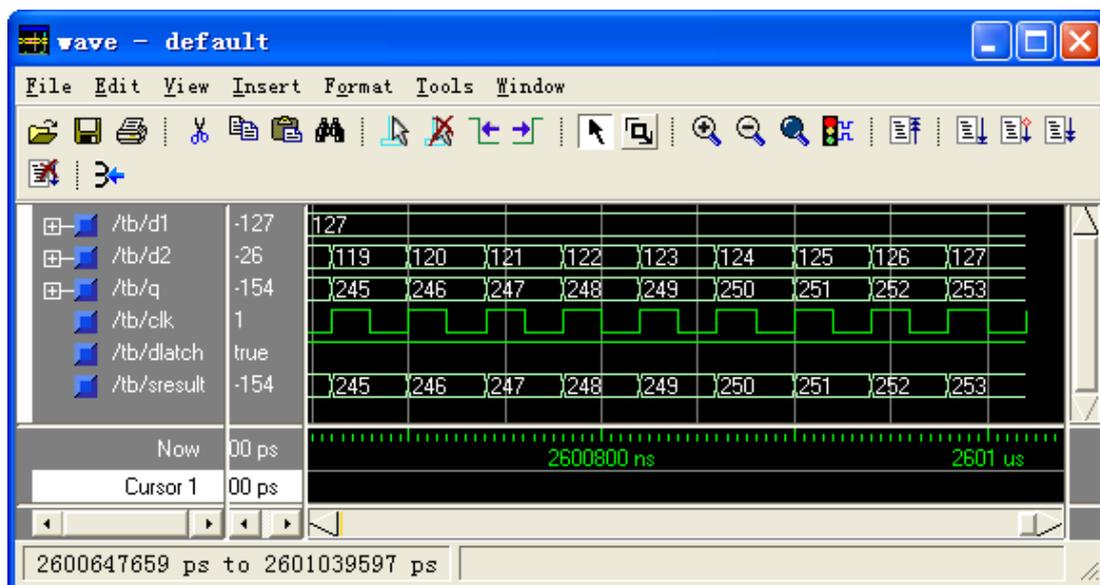


图 8-4-2 仿真结果波形

11. 结束仿真

输入命令 `quit -sim` 结束仿真。

12. 修改数据文件

将 VC 生成的文件 `Result.dat` 打开，将文件中的第 100 行的 -155 为 100，保存该文件；

13. 仿真

同上，在 ModelSim 主窗口中输入如下命令：

```
vsim work.tb
```

```
view wave
add wave -dec *
run -all
```

14. 仿真结果

运行完毕会在主窗口中显示如下警告信息，我们双击对应的警告信息，可以看到，仿真结果与 Result.dat 中的预定结果不一致的地方。仿真图形如图 8-4-2 所示。

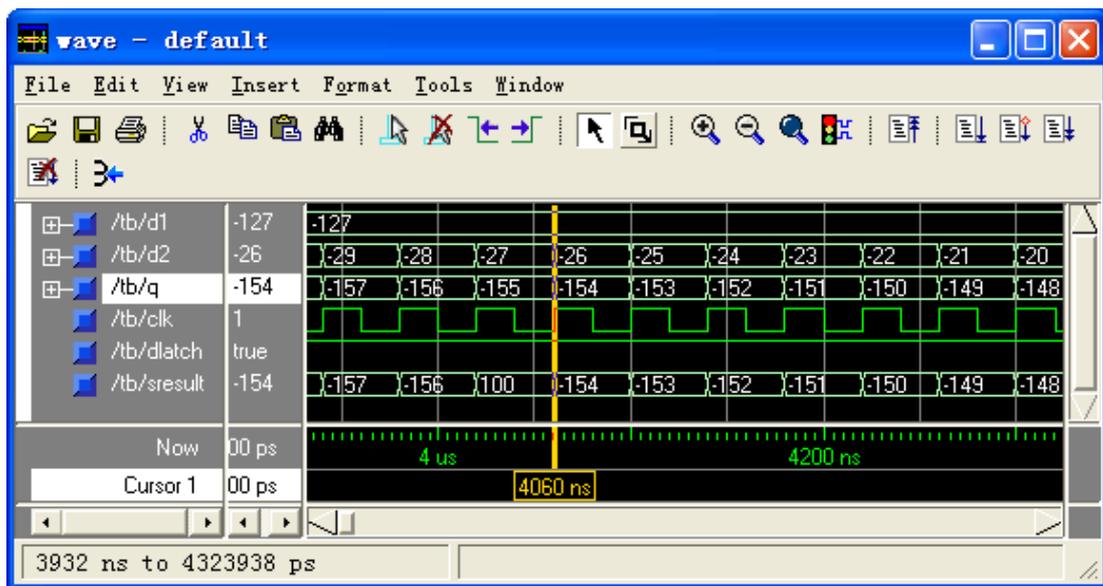


图 8-4-2 警告出现处的仿真结果波形

```
# ** Warning: Two values are different
#   Time: 4060 ns Iteration: 0 Instance: /tb
# ** Fatal: (vsim-3551) TEXTIO : Read past end of file "TestData.dat".
#   Time: 2601020 ns Iteration: 0 Process: /tb/line__34 File:
D:/yuProj/modelsim/TextioTest/TestBench.vhd
# Fatal error at D:/yuProj/modelsim/TextioTest/TestBench.vhd line 41
#
```

15. 退出仿真

输入命令 `quit -sim ;`

仿真分析：

在该程序中，首先读出输入文件的一行内容，再从该行中提取出两个值输入加法器。从预定结果文件中提取出一个值，将加法器计算结果与该值比较，若两者不同则输出警告信息。我们也可以将输出写入一个文本文件，再比较两个文本文件的异同以获知出错地方。这里要注意的是要使用 TEXTIO 程序包，另外，测试文件的实体内的端口为空，相当于一块独立的电路板，使用 Component 在其中包含了上面定义的加法器，该独立的电路板所完

成的功能是对设计的加法器进行测试。在该程序中使用了 `assert` 断言语句，要注意该语句后的表达式或变量为真时不执行后续的输出，为假时执行后续的输出。在该程序中，还使用了类型转换函数 `CONV_STD_LOGIC_VECTOR()` 将整数转换为 8 位的标准类型。另外，在程序中定义了变量 `Dlatch`，该变量的作用是将计算结果与预定结果的比较延迟一个时钟周期，周期地与预定结果比较。

8.5 ModelSim 的配置

在上一节的仿真中，如果你在编译时出现如下错误：

```
** Error: D:/yuProj/modelsim/TextioTest/TestBench.vhd(34): Unknown
identifier: read_mode
** Error: D:/yuProj/modelsim/TextioTest/TestBench.vhd(34): FILE
declaration using 1076-1993 syntax. Recompile using -93 switch.
** Error: D:/yuProj/modelsim/TextioTest/TestBench.vhd(34): VHDL Compiler
exiting
```

也许你会没有一点办法，你可能会问，我明明是照着书本上的例程，为什么会出现这些错误呢？其实，只需动一动鼠标，改一下 ModelSim 的配置就可以了。在 Workspace 窗口中，我们在出错的文件 `TestBench.vhd` 上点击右键，选择 `Properties`，会出现如图 8-5-1 所示的窗口，这里我们可以看到 ModelSim 的目录之间的分割符为“/”，而 DOS 下目录的分割符为“\”，有点不同，我们点击 `VHDL` 选项卡，会看到如图 8-5-2 所示的界面，这里关键的地方是在 `Use 1993 Language Syntax` 前打勾，表示我们的设计在编译时候按照 VHDL'93 的语法标准，打勾之后，你可以再编译一下，这次应该没有问题了吧。此外，在图 8-5-2 中，还可以设定什么情况下出现警告信息等选项。

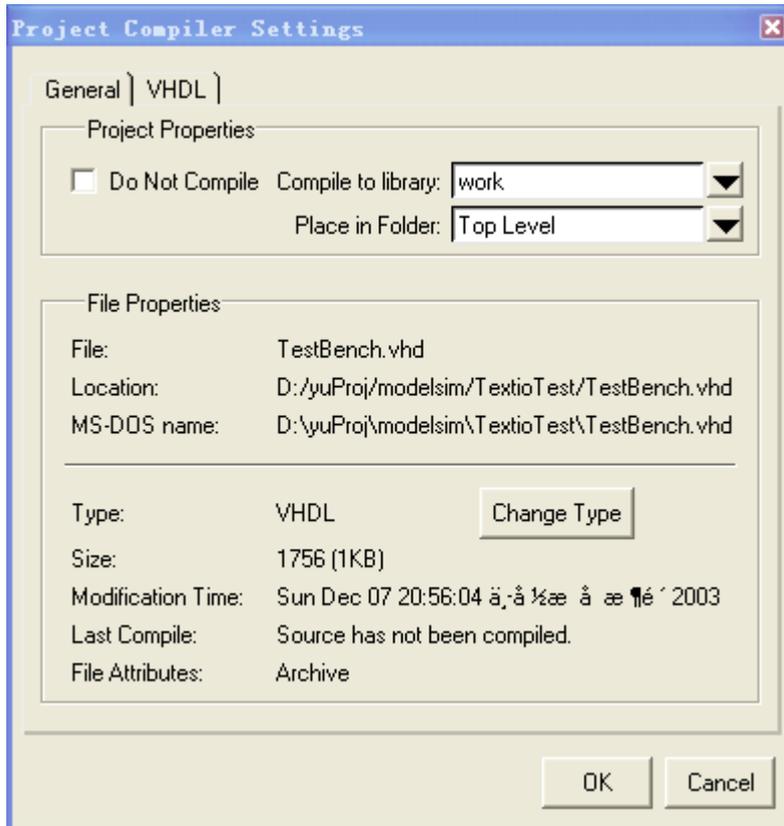


图 8-5-1 工程编译设定

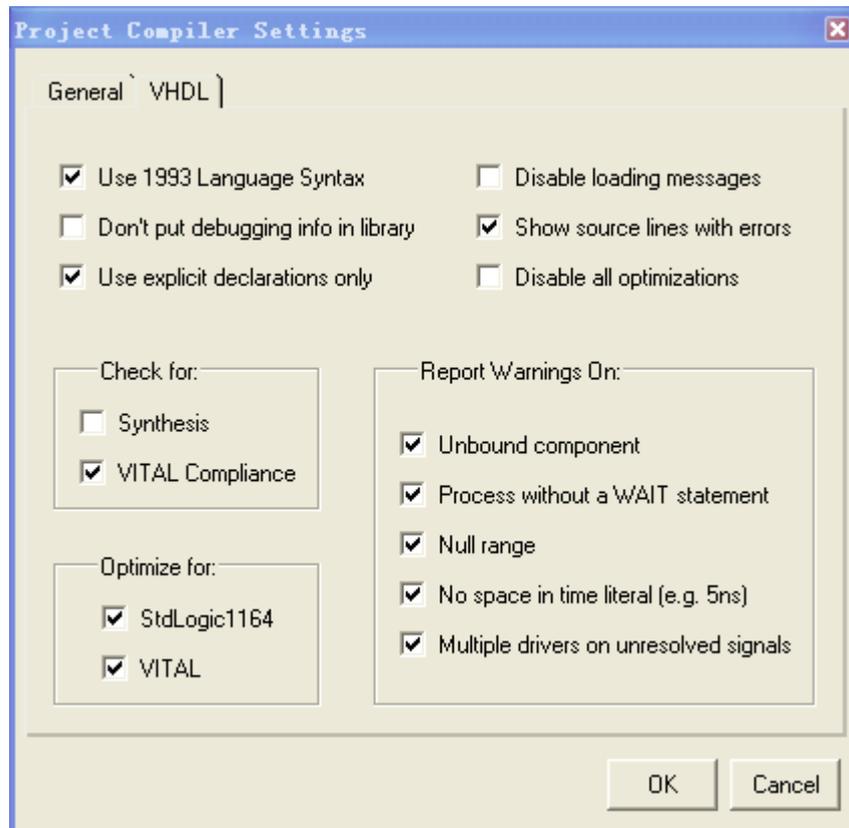


图 8-5-2 VHDL 方面的工程编译设定

在仿真时，是不是还记得仿真时候有一些默认的选项，如仿真的时间单位，每次单步仿真的时间等信息，这里我们可以对其进行修改。点击菜单 Simulate->Simulation Options，我们可以看到如图 8-5-3 所示的界面，在 Default Radix 下为添加信号到波形文件中默认的信号格式，如 Symbolic 表示符号型，Binary 表示以二进制显示，Octal 表示以八进制表示，Decimal 表示以十进制表示，Unsigned 表示以无符合数表示，Hexadecimal 表示以十六进制表示，ASCII 表示以 ASCII 码来表示信号的值。在 Default Run 框中的值表示默认的单步运行时间。Iteration Limit 表示在一定时间内循环最大的次数，以避免无限循环系统无法承受。

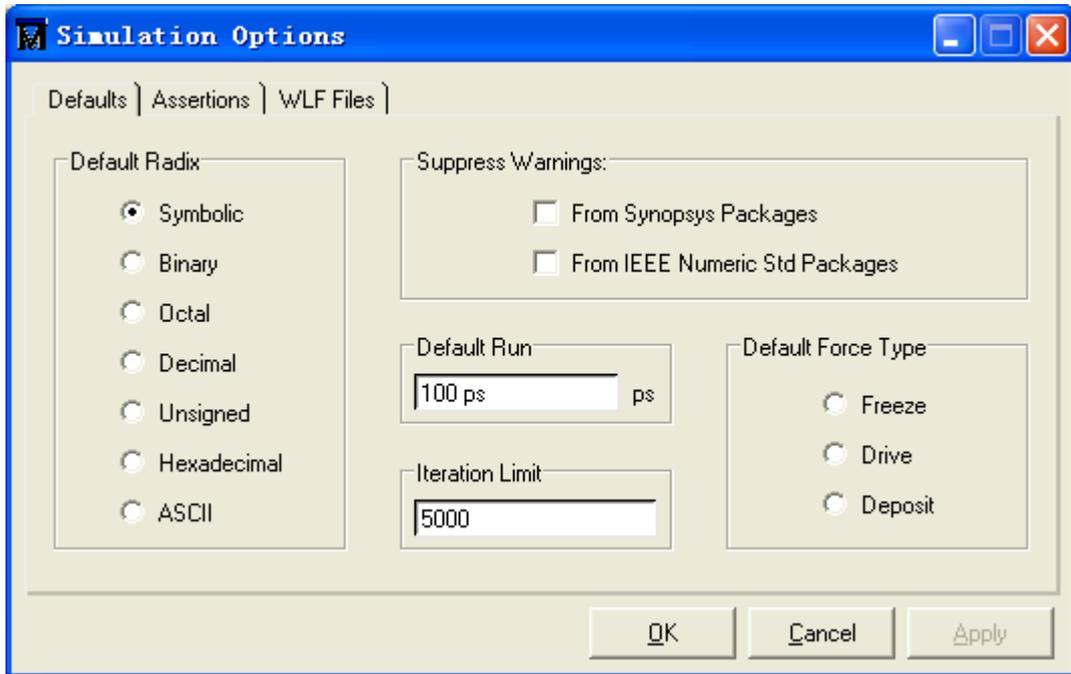


图 8-5-3 仿真选项设定

点击图 8-5-3 中的 Assertions 选项卡，可以看到如图 8-5-4 所示的窗口，在其中可以对仿真遇到何种情况中止，并可以设置忽略一些情况。这些设置有利于我们调试分析不同的情况。

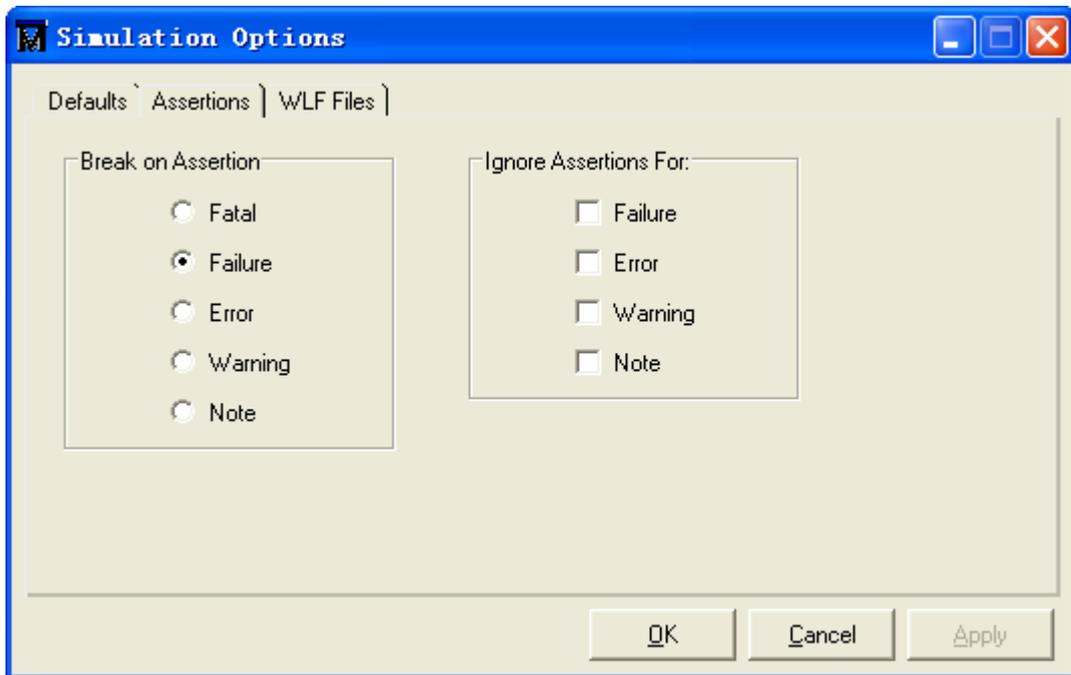


图 8-5-4 仿真中的断言设置

其实上述的设置都保存在 ModelSim 安装目录下的 modelsim.ini 文件中,我们用记事本可以查看该文件的内容,该文件中较为重要的是[Library]、[Vcom]以及[Vsim]。

在[Library]后面是各个库的名字及其存放目录,如下:

```
[Library]
std = $MODEL_TECH/./std
ieee = $MODEL_TECH/./ieee
verilog = $MODEL_TECH/./verilog
vital2000 = $MODEL_TECH/./vital2000
std_developerskit = $MODEL_TECH/./std_developerskit
synopsys = $MODEL_TECH/./synopsys
modelsim_lib = $MODEL_TECH/./modelsim_lib

; VHDL Section
unisim = $MODEL_TECH/./xilinx/vhdl/unisim
simprim = $MODEL_TECH/./xilinx/vhdl/simprim
xilinxcorelib = $MODEL_TECH/./xilinx/vhdl/xilinxcorelib
aim = $MODEL_TECH/./xilinx/vhdl/aim
pls = $MODEL_TECH/./xilinx/vhdl/pls
cpld = $MODEL_TECH/./xilinx/vhdl/cpld

; Verilog Section
unisims_ver = $MODEL_TECH/./xilinx/verilog/unisims_ver
uni9000_ver = $MODEL_TECH/./xilinx/verilog/uni9000_ver
simprims_ver = $MODEL_TECH/./xilinx/verilog/simprims_ver
xilinxcorelib_ver = $MODEL_TECH/./xilinx/verilog/xilinxcorelib_ver
aim_ver = $MODEL_TECH/./xilinx/verilog/aim_ver
cpld_ver = $MODEL_TECH/./xilinx/verilog/cpld_ver
```

在[vcom]后面有一些编译时的选项设置,值为零表示为 OFF,为 1 表示 ON,例如 VHDL93=1 表示编译时按照 VHDL93 标准,Show_source=1 表示编译出错时是否将出错的那一行显示出来,我们可以根据其注释来了解其含义,该部分内容如下:

```
[vcom]
; Turn on VHDL-1993 as the default. Normally is off.
VHDL93 = 1

; Show source line containing error. Default is off.
; Show_source = 1

; Turn off unbound-component warnings. Default is on.
```

```

; Show_Warning1 = 0

; Turn off process-without-a-wait-statement warnings. Default is on.
; Show_Warning2 = 0

; Turn off null-range warnings. Default is on.
; Show_Warning3 = 0

; Turn off no-space-in-time-literal warnings. Default is on.
; Show_Warning4 = 0

; Turn off multiple-drivers-on-unresolved-signal warnings. Default is on.
; Show_Warning5 = 0

; Turn off optimization for IEEE std_logic_1164 package. Default is on.
; Optimize_1164 = 0

; Turn on resolving of ambiguous function overloading in favor of the
; "explicit" function declaration (not the one automatically created by
; the compiler for each type declaration). Default is off.
Explicit = 1

; Turn off VITAL compliance checking. Default is checking on.
; NoVitalCheck = 1

; Ignore VITAL compliance checking errors. Default is to not ignore.
; IgnoreVitalErrors = 1

; Turn off VITAL compliance checking warnings. Default is to show warnings.
; Show_VitalChecksWarnings = false

; Turn off "loading..." messages. Default is messages on.
; Quiet = 1

; Turn on some limited synthesis rule compliance checking. Checks only:
; -- signals used (read) by a process must be in the sensitivity list
; CheckSynthesis = 1

```

在[Vsim]后是一些仿真的选项，例如 Resolution=ps 表示仿真最小的分辨率为 1ps，UserTimeUnit=default 表示分辨率的单位，default 表示默认的单位，而默认的单位为分辨率所使用的单位，RunLength=100 表示执行一次 Run 所仿真的时间，我们可以根据文件中的说明来分析其含义，该部分内容如下：

```
[vsim]
```

```

; Simulator resolution
; Set to fs, ps, ns, us, ms, or sec with optional prefix of 1, 10, or 100.
Resolution = ps

; User time unit for run commands
; Set to default, fs, ps, ns, us, ms, or sec. The default is to use the
; unit specified for Resolution. For example, if Resolution is 100ps,
; then UserTimeUnit defaults to ps.
UserTimeUnit = default

; Default run length
RunLength = 100

; Maximum iterations that can be run without advancing simulation time
IterationLimit = 5000

; Directive to license manager:
; vhdl      Immediately reserve a VHDL license
; vlog      Immediately reserve a Verilog license
; plus      Immediately reserve a VHDL and Verilog license
; nomgc     Do not look for Mentor Graphics Licenses
; nomti     Do not look for Model Technology Licenses
; noqueue   Do not wait in the license queue when a license isn't
available
; License = plus

; Stop the simulator after an assertion message
; 0 = Note 1 = Warning 2 = Error 3 = Failure 4 = Fatal
BreakOnAssertion = 3

; Assertion Message Format
; %S - Severity Level
; %R - Report Message
; %T - Time of assertion
; %D - Delta
; %I - Instance or Region pathname (if available)
; %% - print '%' character
; AssertionFormat = "*** %S: %R\n  Timf: %T Iteration: %D%I\n"

; Assertion File - alternate file for storing assertion messages
; AssertFile = assert.log

```

```
; Default radix for all windows and commands...
; Set to symbolic, ascii, binary, octal, decimal, hex, unsigned
DefaultRadix = symbolic

; VSIM Startup command
; Startup = do startup.do

; File for saving command transcript
TranscriptFile = transcript

; File for saving command history
;CommandHistory = cmdhist.log

; Specify whether paths in simulator commands should be described
; in VHDL or Verilog format. For VHDL, PathSeparator = /
; for Verilog, PathSeparator = .
PathSeparator = /

; Specify the dataset separator for fully rooted contexts.
; The default is ':'. For example, sim:/top
; Must not be the same character as PathSeparator.
DatasetSeparator = :

; Disable assertion messages
; IgnoreNote = 1
; IgnoreWarning = 1
; IgnoreError = 1
; IgnoreFailure = 1

; Default force kind. May be freeze, drive, or deposit
; or in other terms, fixed, wired or charged.
; DefaultForceKind = freeze

; If zero, open files when elaborated
; else open files on first read or write
; DelayFileOpen = 0

; Control VHDL files opened for write
; 0 = Buffered, 1 = Unbuffered
UnbufferedOutput = 0

; Control number of VHDL files open concurrently
```

```
; This number should always be less than the
; current ulimit setting for max file descriptors
; 0 = unlimited
ConcurrentFileLimit = 40

; This controls the number of hierarchical regions displayed as
; part of a signal name shown in the waveform window. The default
; value or a value of zero tells VSIM to display the full name.
; WaveSignalNameWidth = 0

; Turn off warnings from the std_logic_arith, std_logic_unsigned
; and std_logic_signed packages.
; StdArithNoWarnings = 1

; Turn off warnings from the IEEE numeric_std and numeric_bit
; packages.
; NumericStdNoWarnings = 1

; Control the format of a generate statement label. Don't quote it.
; GenerateFormat = %s__%d

; Specify whether checkpoint files should be compressed.
; The default is to be compressed.
; CheckpointCompressMode = 0

; List of dynamically loaded objects for Verilog PLI applications
; Veriuser = veriuser.sl
```

8.6 小结

本章以简单的例子讲述了 ModelSim 的仿真流程，以及使用 Testbench 和 Textio 对设计进行仿真的流程。适合初学者入门，较为深入的教程可以参阅软件的用户手册。