

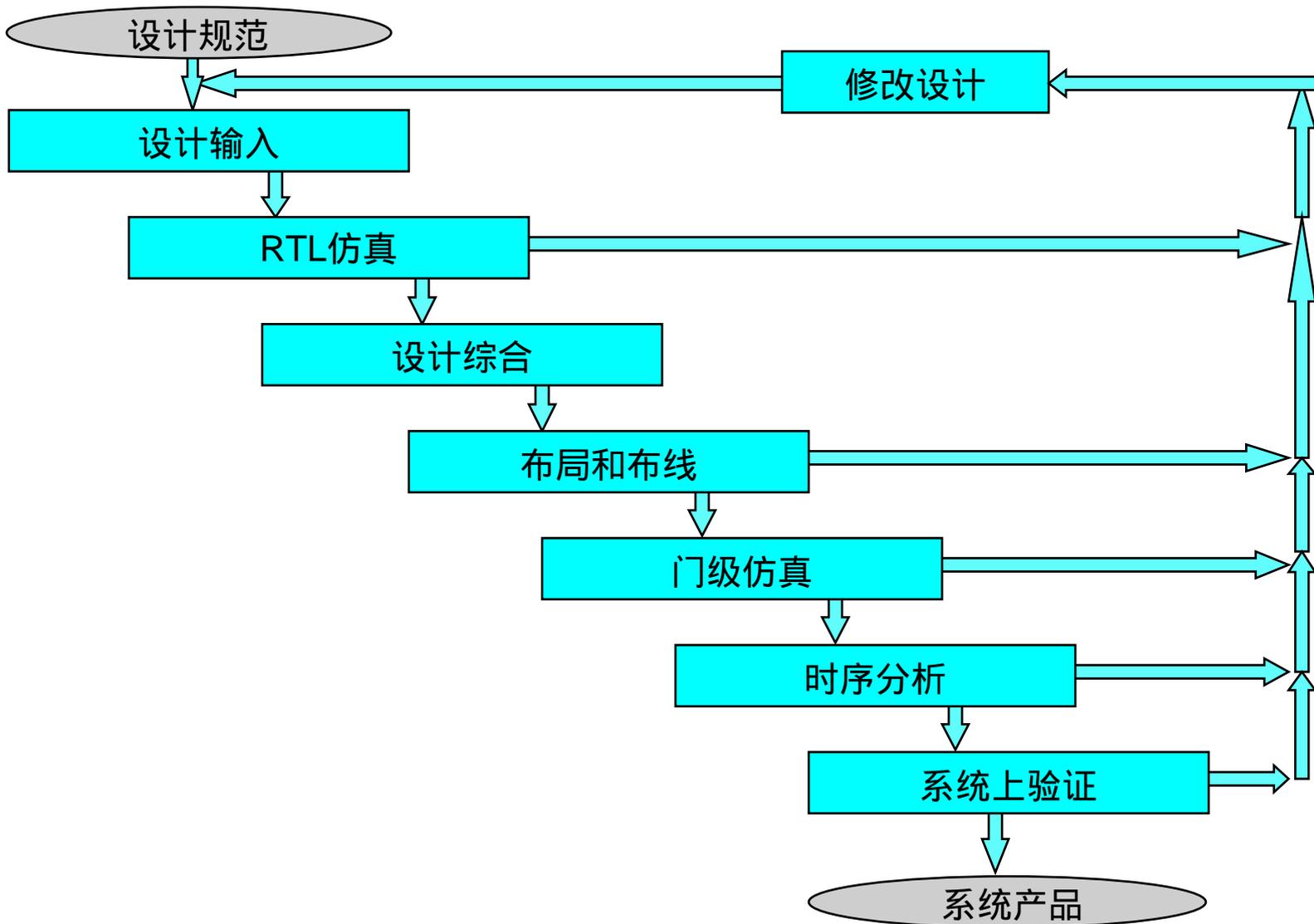
---

# Model*Sim* 使用教程

---

# 典型的PLD设计流程

# 典型的PLD流程



# 典型的PLD设计流程

## ■ 设计输入

- 设计的行为或结构描述

## ■ RTL仿真(ModelSim)

- 功能仿真
- 验证逻辑模型(没有使用时间延迟)
- 可能要求编辑设计

## ■ 综合

- 把设计翻译成原始的目标工艺
- 最优化
  - 合适的面积要求和性能要求

## ■ 布局和布线

- 映射设计到目标工艺里指定位置
- 指定的布线资源应被使用

# 典型的PLD设计流程

- 门级仿真 (ModelSim)
  - 时序仿真
  - 验证设计一旦编程或配置将能在目标工艺里工作
  - 可能要求编辑设计
- 时序分析
- 验证合乎性能规范
  - 可能要求编辑设计
- 版图设计
  - 仿真版图设计
  - 在板编程和测试器件

# Model*Sim*概览



# Model*Sim* 仿真工具

- 由Model技术公司开发
- 工业上最通用的仿真器之一
- 可在Verilog 和 VHDL仿真
  - OEM版本允许Verilog仿真 **或者** VHDL 仿真



**Model Technology**  
A MENTOR GRAPHICS COMPANY

# Model*Sim* 产品

- Model*Sim*/VHDL 或者 Model*Sim*/Verilog
  - OEM
- Model*Sim*/LNL
  - 许可 Verilog 或者 VHDL , 但是不同时许可
- Model*Sim*/PLUS
  - 设计者能立刻混合仿真Verilog 和 VHDL
- Model*Sim*/SE
  - 首要的版本
  - PLUS的所有功能连同附加功能



# Model*Sim* OEM 功能

## ■ 提供完全的标准

- '87 VHDL
- '93 VHDL
- IEEE 1364-'95 Verilog
- SDF 1.0 - 3.0
- VITAL 2.2b
- VITAL '95

## ■ 易用的界面

- 通用的平台

# 用Model*Sim* 仿真



**Model Technology**  
A MENTOR GRAPHICS COMPANY

# 课程安排

---

- 基本的仿真步骤
- 用户界面
- 功能仿真
- Quartus输出仿真文件
- 时序仿真

# Model 技术公司的 ModelSim



# Model*Sim*实现方法

- 交互式的命令行 (Cmd)
  - 唯一的界面是控制台的命令行, 没有用户界面
- 用户界面 (UI)
  - 能接受菜单输入和命令行输入
  - 课程主要讨论
- 批处理模式
  - 从DOS或UNIX命令行运行批处理文件
  - 不讨论

# 基本仿真步骤

1 ⇒ 建立库

2 ⇒ 映射库到物理目录

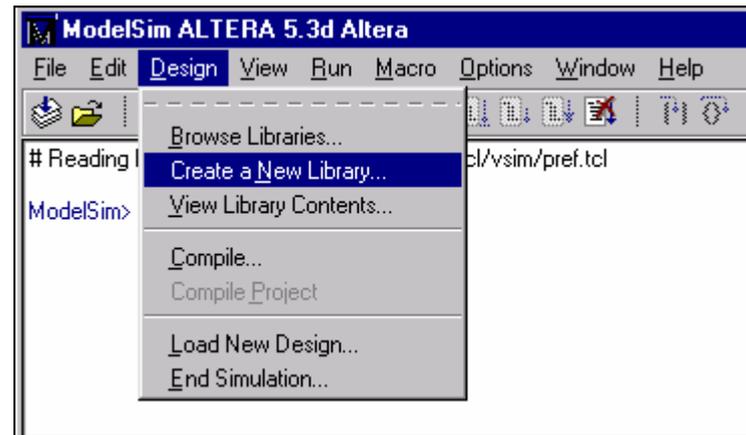
3 ⇒ 编译源代码

- 所有的HDL代码必须被编译
- Verilog和VHDL是不同的

4 ⇒ 启动仿真器

5 ⇒ 执行仿真

# 1 ⇒ 建立ModelSim库



UI) 从主菜单里面:

*Design -> Create a New Library*

Cmd) 从main, 记录窗口:

ModelSim> *vlib* <库名>

# ModelSim 库

- 包含编译设计单元的目录
  - VHDL 和 Verilog 都被编译到库里
- 两个类型
  - Working (缺省值 *work*)
    - 包含当前被编译的设计单元
    - 编译前必须建立一个working库
    - 每个编译只允许一个
  - Resource
    - 包含能被当前编译引用的设计单元
    - 在编译期间允许多个
    - VHDL库能通过LIBRARY和USE子句引用

# ModelSim 设计单元

## ■ 主要

- 在一个特定的库中必须有唯一的名字
- VHDL
  - Entities(实体)
  - Package Declarations(包声明)
  - Configurations(结构)
- Verilog
  - Modules(模块)
  - User Defined Primitives (用户定义原语)

## ■ 次要

- 在相同的库里单元可以用一个普通名称
- VHDL
  - Architectures(体系)
  - Package bodies
- Verilog没有次要单元

# VHDL 预先确定库

## ■ VHDL

- Library *std* 包含 *packages standard* 和 *textio*
  - 这些 *packages* 初学者不要去修改

## ■ IEEEpure

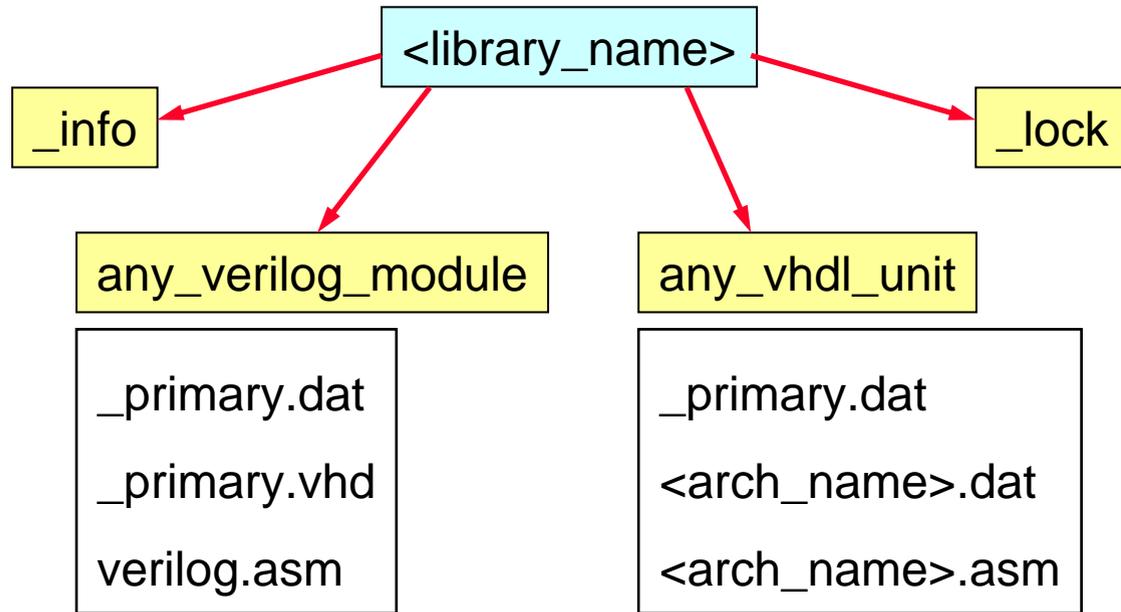
- 包含唯一IEEE 认可的 *std\_logic\_1164 packages*
- 用于仿真加速

## ■ IEEE

- 包含预编译的Synopsys和IEEE 算法包
- 给*std\_logic*的基本类型
- 用于仿真加速

# *vlib* <library\_name> 命令

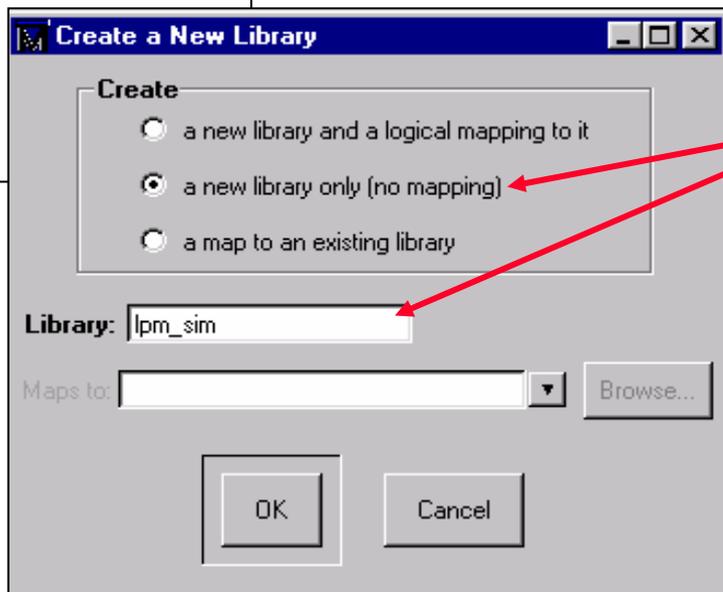
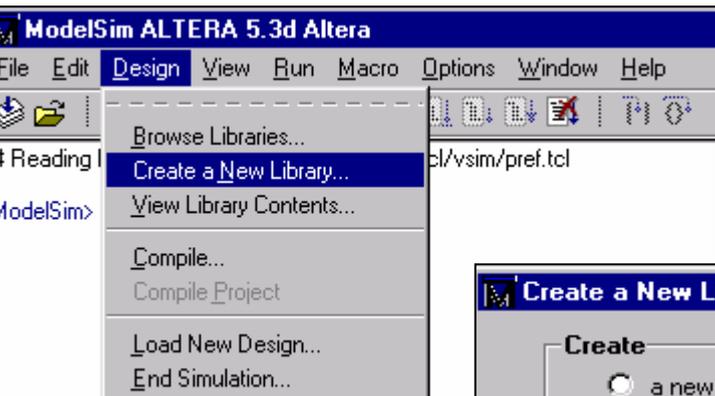
- 建立库
- 缺省值是 work



## ■ Where

- \_primary.dat - Verilog module 或 VHDL entity的编码格式
- \_primary.vhd - Verilog 端口的VHDL entity陈述
- <arch\_name>.dat - VHDL体系的编码格式
- verilog.asm 和 <arch\_name>.asm - 执行代码文件

# 建立库(UI)



选择 **a new library only** 和  
输入库名

这个命令在局部目录  
建立一个库子目录

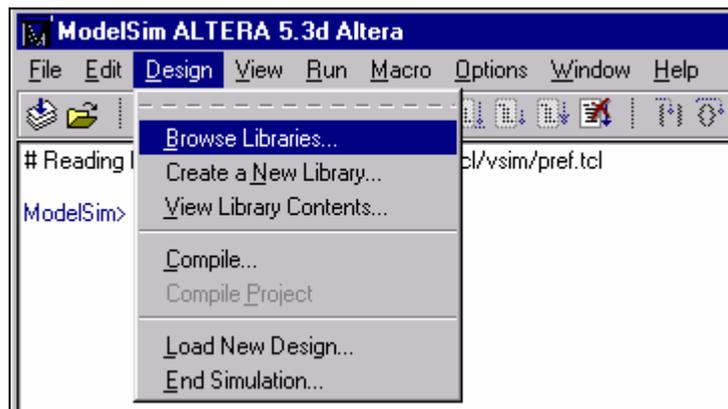
-> vlib lpm\_sim

# 映射逻辑库名

- 必须映射一个逻辑库名到库路径(定位)
  - 在库路径里的文件必须已经被编译
  - 支持相对的, 绝对的, 和软件路径名
- 需要库在工作目录里没有定位
- 用 *vmap* 命令

## 2 ⇒ 映射逻辑库名

- 语法: *vmap* <logical\_name> <directory\_path>



UI) 从主菜单:

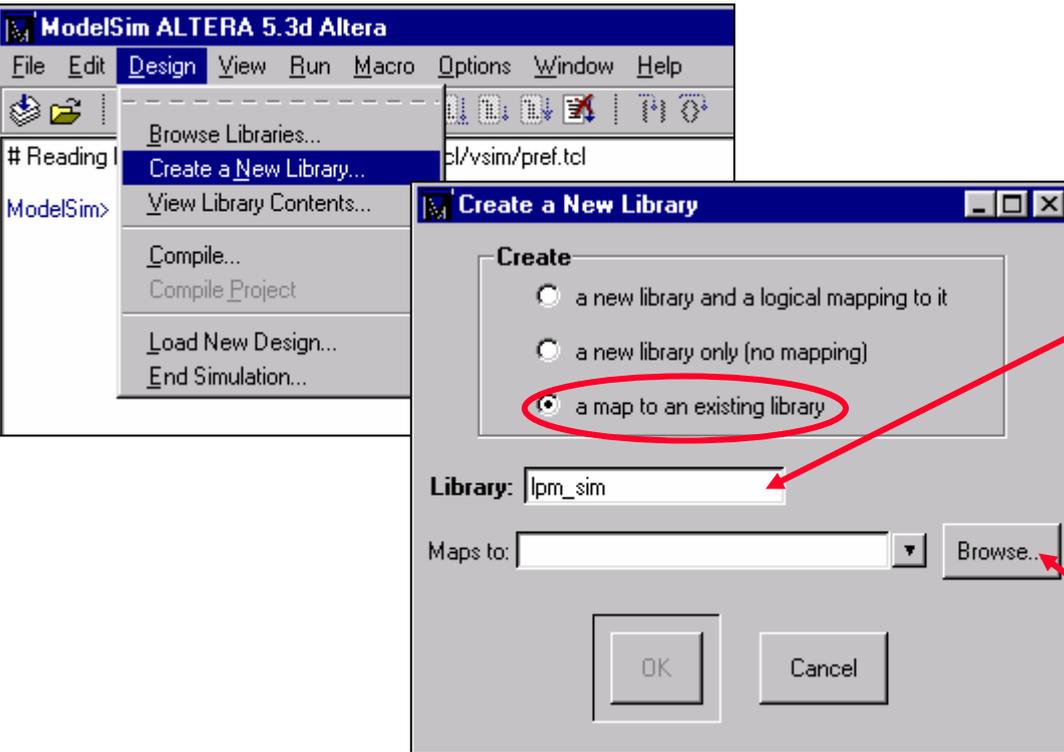
*Design -> Browse Libraries*

*Design -> Create a New Library*

Cmd) 从主体的记录窗口:

```
ModelSim> vmap my_work c:/my_design/my_lib
```

# 映射现有的库(UI)



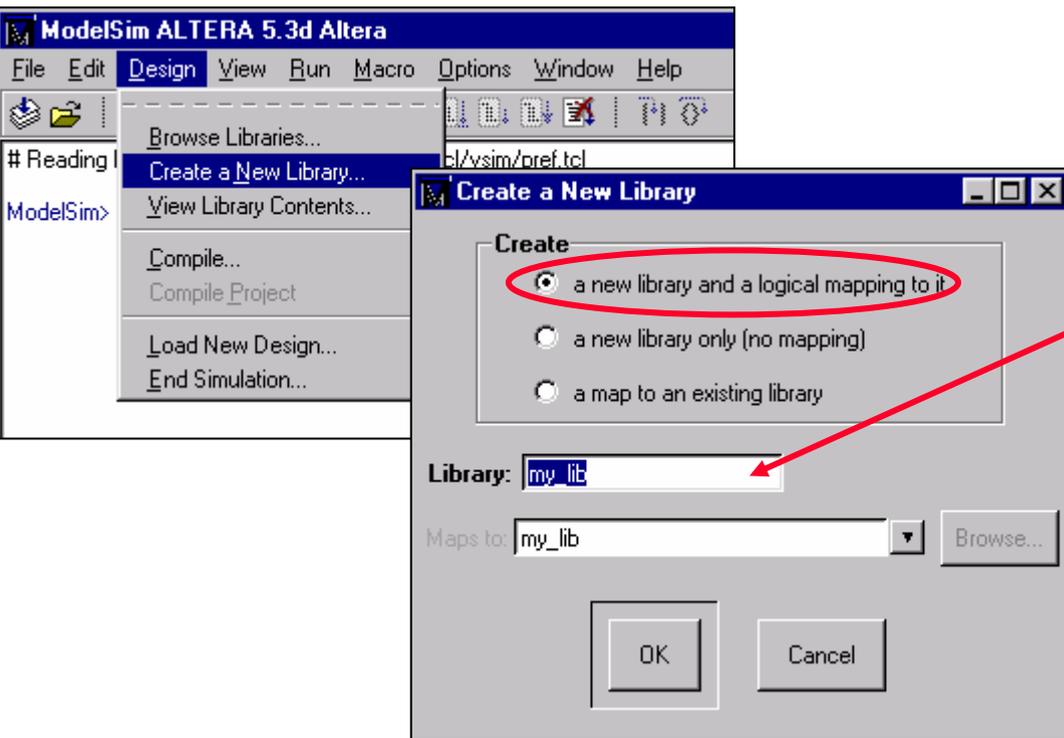
选择 *a map to an existing library* 和输入库名

用这个命令可映射一个设计单元已经预编译的库目录

浏览库目录

```
-> vmap lpm_sim c:/Quartus/library/lpm
```

# 映射现有的库(UI)

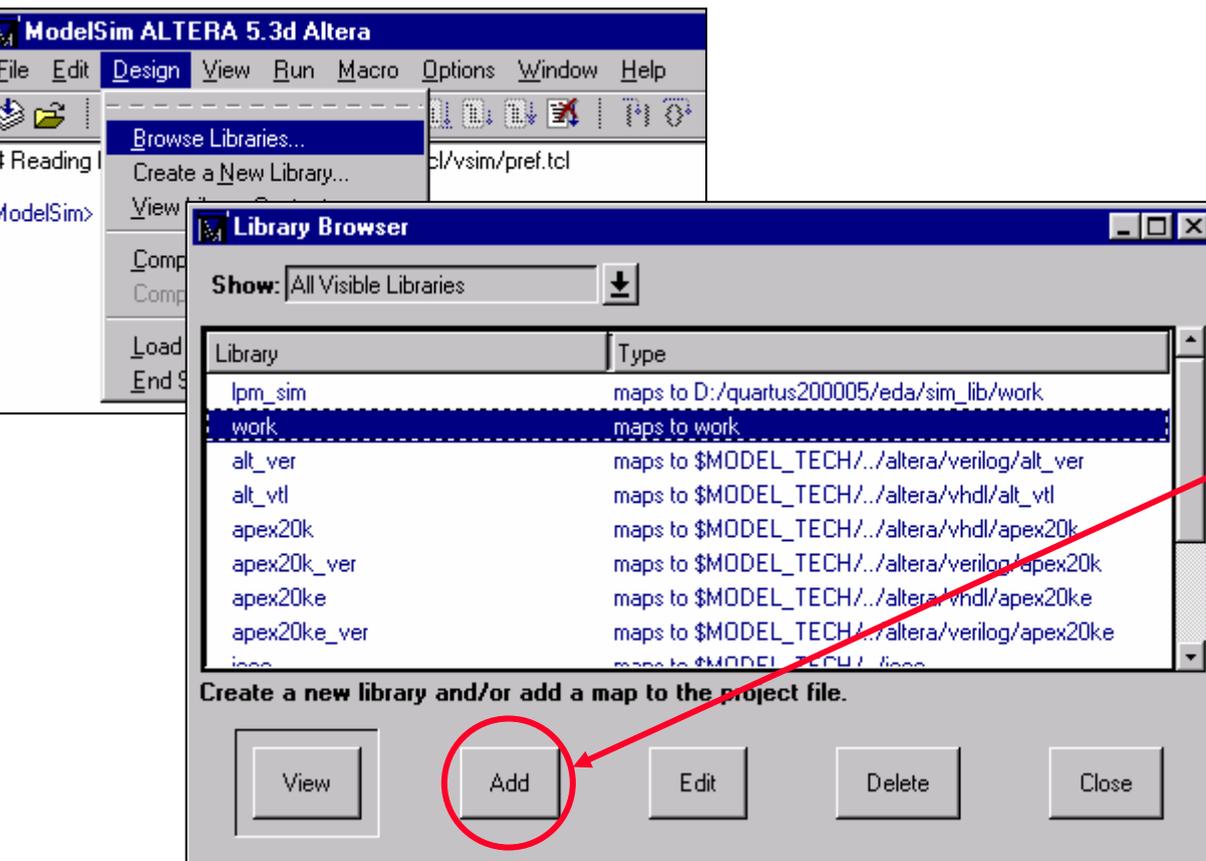


选择 *a new library and a logical mapping to it* 输入库名

这个命令在局部目录建立一个库目录然后为它设立映射

```
-> vlib my_lib  
-> vmap my_lib my_lib
```

# 映射库 (UI)



用Add 按钮建立一个新的库和映射

# 其他的库命令

## ■ *vdel*

- 从指定的库中删除一个完整的库或者一个设计单元
- UI) *Design -> Library Browser* (删除库或者映射)
- UI) *Design -> View Library Contents* (删除设计单元)
- Cmd) *vdel-lib* <library\_name> <design\_unit>

## ■ *vdir*

- 显示指定库的内容
- UI) *Design -> View Library Contents*
- Cmd) *vdir-lib* <library\_name>

## 3 ⇒ 编译源代码(VHDL)

- UI) *Design -> Compile*
- Cmd) *vcom -work* <library\_name> <file1>.vhd <file2>.vhd
  - 文件按出现的顺序被编译
  - Compilation order/dependencies (next slide)
- '87 VHDL是缺省的
  - UI) 用*Default Options* 按钮设为'93
  - Cmd) 用 *-93* 选项(必须是第一个参数)
- 缺省编译到工作库
  - 例如. *Vcom -93* my\_design.vhd

*注意: 当库中涉及的设计单元被改变时设计单元必须重新分析。*

# VHDL Design Units Dependencies

- Entity before Architecture
- Package Declaration before Package Body
- Design units must be compiled before being referenced
  - Packages before Entity/Architectures using them
  - Entities/Configurations before Architectures referencing them
- Configurations are compiled last

## 3 ⇒ 编译源代码(Verilog)

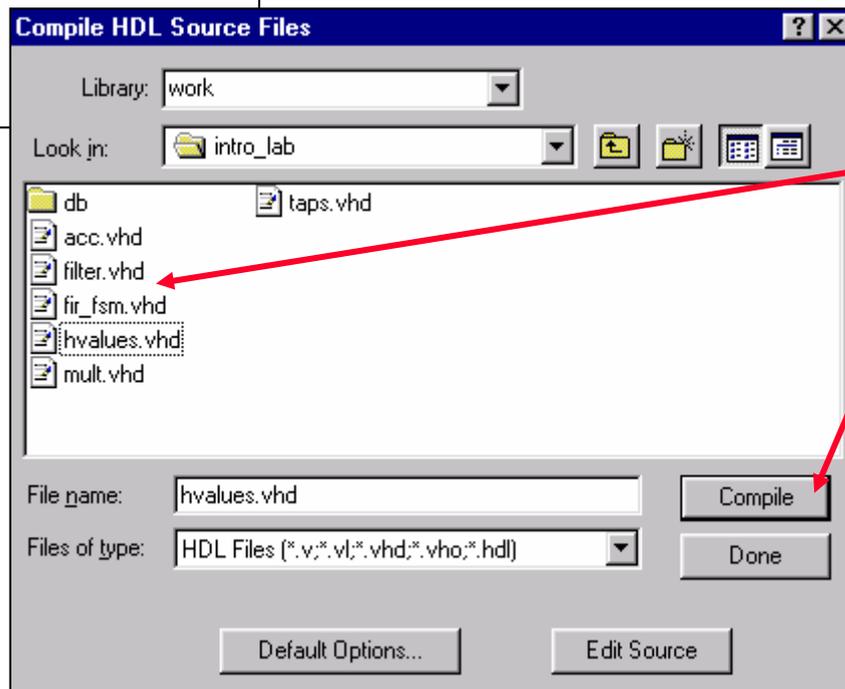
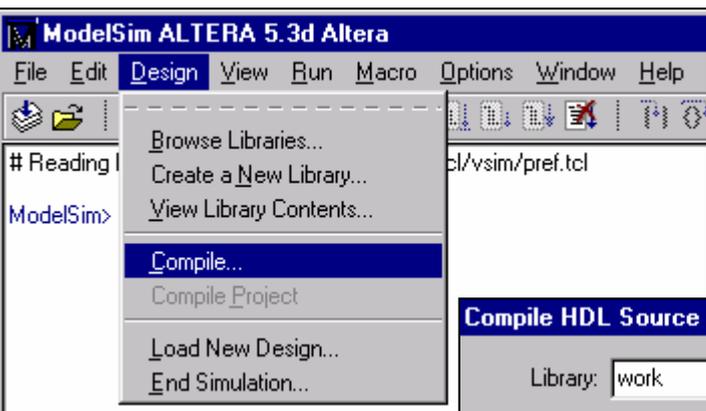
- UI) *Design -> Compile*
- Cmd) *vlog* -work <library\_name> <file1>.v <file2>.v
  - 文件按出现的顺序被编译
  - 文件的顺序或者编辑的顺序不重要
- 支持增量式编译
- 缺省编译到work库
  - 例如. *vlog* my\_design.v

*注意:当库中涉及的设计单元被改变时设计单元必须重新分析。*

# Verilog 增量编译

- 只有被改变的设计单元被编译
- 自动或者手动
  - 手动更有效
  - 自动在命令行用 `-incr` 选项
- Module 和 UDP实例和 hierarchical references可在仿真时解决
  - 一些错误在编辑时将不能察觉
    - 引用的模块没有编译
    - 不正确的端口连接
    - 不正确的hierarchical references

# 编译 (UI)



点亮一个或多个文件并点击 **Compile**

# 错误信息

错误信息在 Main 窗口显示

```
ModelSim ALTERA 5.3d Altera
File Edit Design View Run Macro Options Window Help
100000
# Region /lab2_fifo/u2/prj_ran_dq_component
# WARNING: E:/Quartus_MS_test/lab2/Verilog/lab2_crt.v(54) (TFMPC) - Too few port connections
# Region /lab2_fifo/u1/prj_counter_component
vlog reportprogress 300 -work work E:/Quartus_MS_test/lab2/Verilog/lab2_fifo.v
# Model Technology ModelSim ALTERA vlog 5.3d Altera Compiler 2000.04 May 17 2000
# - Compiling module lab2_fifo_tb
# ERROR: E:/Quartus_MS_test/lab2/Verilog/lab2_fifo_tb.v(43) near "": expecting ";"
# ERROR: E:/Quartus_MS_test/lab2/Verilog/lab2_fifo_tb.v(45) near "endmodule": expecting END
vlog reportprogress 300 -work work E:/Quartus_MS_test/lab2/Verilog/lab2_fifo.v
# Model Technology ModelSim ALTERA vlog 5.3d Altera Compiler 2000.04 May 17 2000
# - Compiling module lab2_fifo_tb
# ERROR: E:/Quartus_MS_test/lab2/Verilog/lab2_fifo_tb.v(45) near "endmodule": expecting END
vlog reportprogress 300 -work work E:/Quartus_MS_test/lab2/Verilog/lab2_fifo.v
# Model Technology ModelSim ALTERA vlog 5.3d Altera Compiler 2000.04 May 17 2000
# - Compiling module lab2_fifo_tb
# Top level modules:
# lab2_fifo_tb
```

```
source_edit_5 - lab2_fifo_tb.v
File Edit Object Options Window
30
31 #50 tdata <= 20;
32
33 #50 tdata <= 24;
34
35 #50 tdata <= 28;
36
37 #50 tdata <= 32;
38
39 #50 tdata <= 36;
40
41 #50 tdata <= 40;
42
43 #100 creset <= 1;
44
45 end
46
47 endsodule
48
```

在信息上双击，引起错误的代码在 Source 窗口被点亮

## 4 ⇒ 启动仿真器

■ UI) *Design -> Load New Design*

■ Cmd) *vsim -lib* <library\_name> <top\_level\_design>

### ■ VHDL

– *vsim* top\_entity top\_architecture

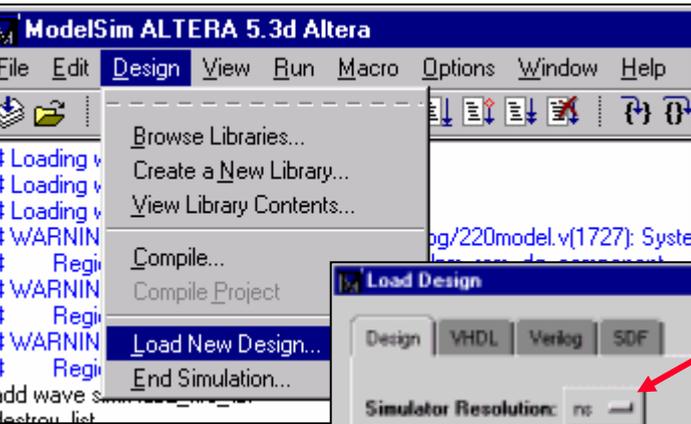
- 仿真 Entity/Architecture 对
- 也能选择一个结构

### ■ Verilog

– *vsim* top\_level1 top\_level2

- 仿真多个top级模块

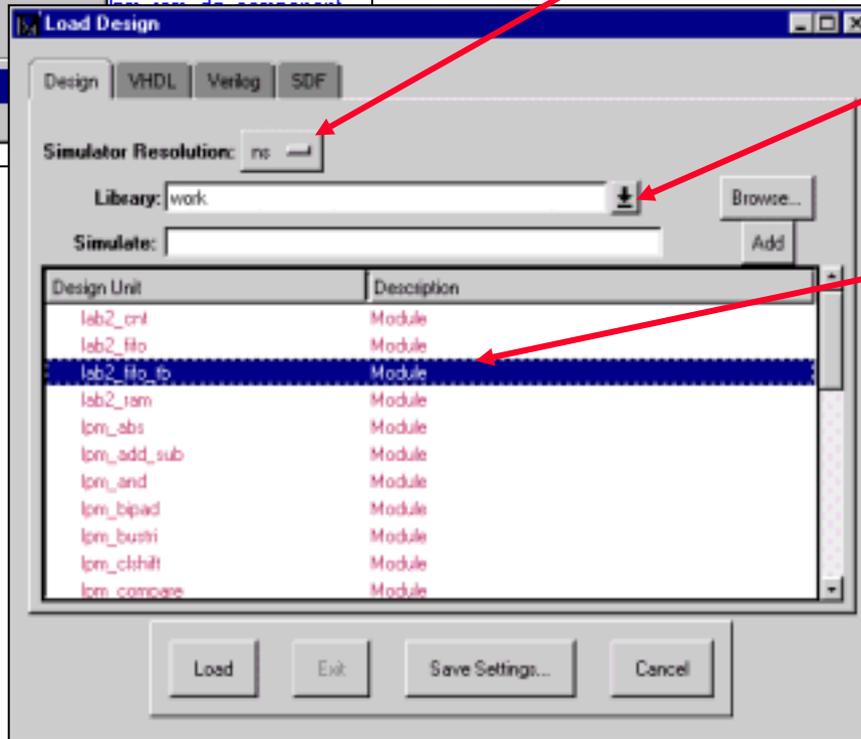
# 启动仿真器 (UI)



选择仿真器的分辨率

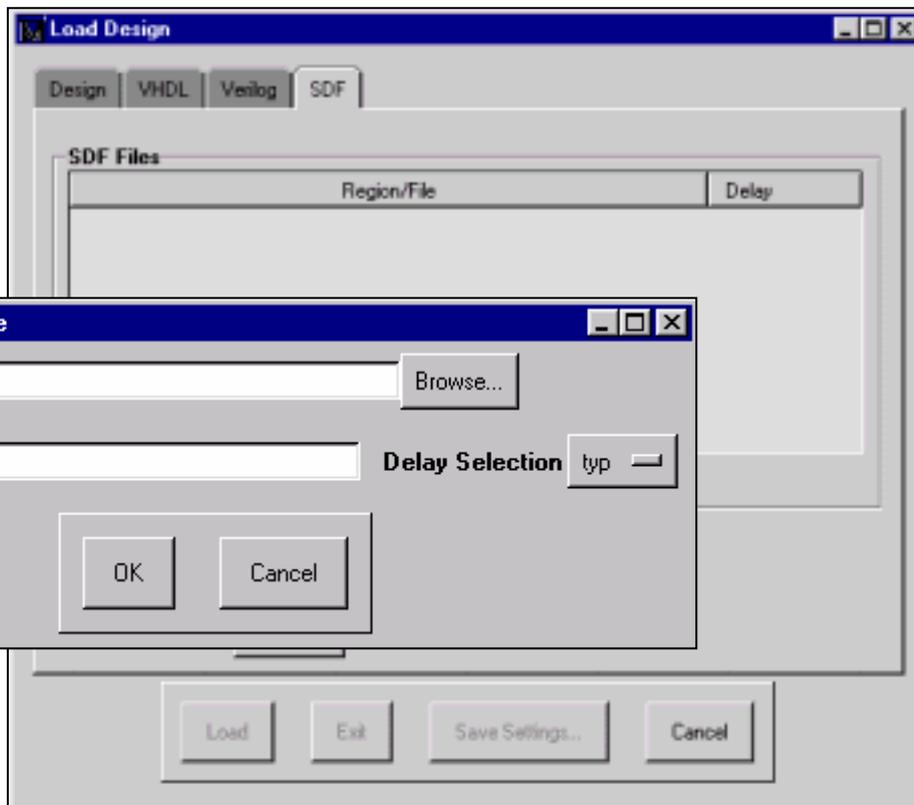
选择库

选择顶级module 或  
entity/architecture



# 启动仿真器(UI)

指定 SDF文件



使用定时值的等级的类型 (如果不是顶级)

# *vsim* 命令的参数

## ■ 参数

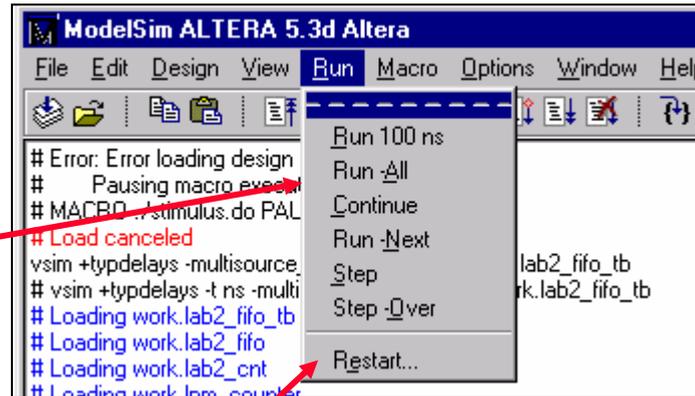
- `-t <time_unit>`
  - 指定仿真的时间分辨率
  - 单位可以是{fs, ps, ns, ms, sec, min, hr}
  - 如果用了 Verilog 的 'timescale 指令, 将使用整个设计中的最小的时间精度
  - 可选项(缺省是 ns)
  
- `-sdfmin | -sdftyp | -sdfmax <instance>=<sdf_filename>`
  - 注释SDF文件
  - 可选项
  - 使用实例名也是可选项; 如果没有使用, SDF用于顶级

## 5 ⇒ 执行仿真

- UI) *Run*
- COM) *run* <time\_step> <time\_units>
- 按timesteps指定的时间长度执行仿真

# 执行仿真 (UI)

选择 timesteps 数量就可以执行仿真



***Restart*** – 重装任何已改动的设计元素并把仿真时间设为零

COM) **restart**

# run 命令参数

## ■ 可选的参数

- `<timesteps>` `<time_unit>`
  - 指定运行的timesteps数量
  - 单位可用{fs, ps, ns, ms, sec}
- `-step`
  - Steps to the next HDL statement
- `-continue`
  - 继续上次在`-step`或断点后的仿真
- `-all`
  - 运行仿真器直到没有其他的事件

# *run* 命令举例

- *run* 1000
  - 从当前位置运行仿真 1000 timesteps
- *run* 2500 ns
  - 从当前位置运行仿真2500 ns
- *run* @3000
  - 运行仿真到 timestep 3000

# 仿真器激励

## ■ 测试台

- Verilog 或 VHDL
- 非常复杂的仿真 交互式仿真

## ■ force命令

- 简单的模块仿真
- 直接从命令控制台输入
- .DO 文件 (宏文件)

# force 命令

- 允许用户给VHDL信号和Verilog线网予以激励

- 常规语法:

- *force* <item\_name> <value> <time>, <value> <time>

- 参数

- *item\_name*

- 被激励的HDL项的名称
    - 必需的
    - 必须是一个Must be a scalar or one-dimensional array of characters
      - Can be an indexed array, array slice, or record sub-element as long as its of the above type
    - Can use wildcards as long as only one match is obtained

# force 命令(继续...)

## ■ 其他参数

### – value

- 被强制的项的值
- 必须适合项的数据类型
- 必需的

### – time

- 指定值的时间单位
- 相对于当前的仿真时间
  - 用 @ character指定绝对时间
- 时间单位能被指定
  - 缺省值是仿真分辨率
- 可选的

Value	Description
1111	character sequence
2#1111	binary radix
10#15	decimal radix
16#F	hexadecimal radix

# force 命令(继续...)

## ■ 其他参数

- `-r[epeat]` <period>
  - 在指定周期重复force命令
  - 可选的
- `-cancel` <period>
  - 在指定周期后取消强制force命令
  - 可选的

# force 命令举例

- **force** clr 0
  - 在当前仿真时间强制 clr 到 0
- **force** bus1 01XZ 100 ns
  - 在当前仿真时间后100ns强制 bus1到 01XZ
- **force** bus2 16#4F @200
  - 仿真启动后强制 bus2到 4F直到200时间单位 , 分辨率在仿真启动时选择
- **force** clk 0 0, 1 20 -repeat 50 -cancel 1000
  - 在当前仿真后0时间单位强制clk到0和在20时间单位强制到1. 每50时间单位重复直到1000. 因此, 下一个 1 将在70时间单位发生
- **force** clk2 1 10 ns, 0 {20 ns} -r 100 ns
  - 和上一个例子相似。-r前面的时间单位表达式必须放在大括号里

# DO 文件

- 自动完成仿真步骤的宏文件
  - 库设置
  - 编译
  - 仿真
  - 强制仿真激励
- 能在所有的ModelSim 模式里被调用
  - UI) *Macro -> Execute*
  - COM) *do* <filename>.do
- 能调用其他的DO文件

```
cd c:\mydir
vlib work
vcom counter.vhd
vsim counter
view *
add wave /*
add list /*
do run.do
```

# DO文件举例

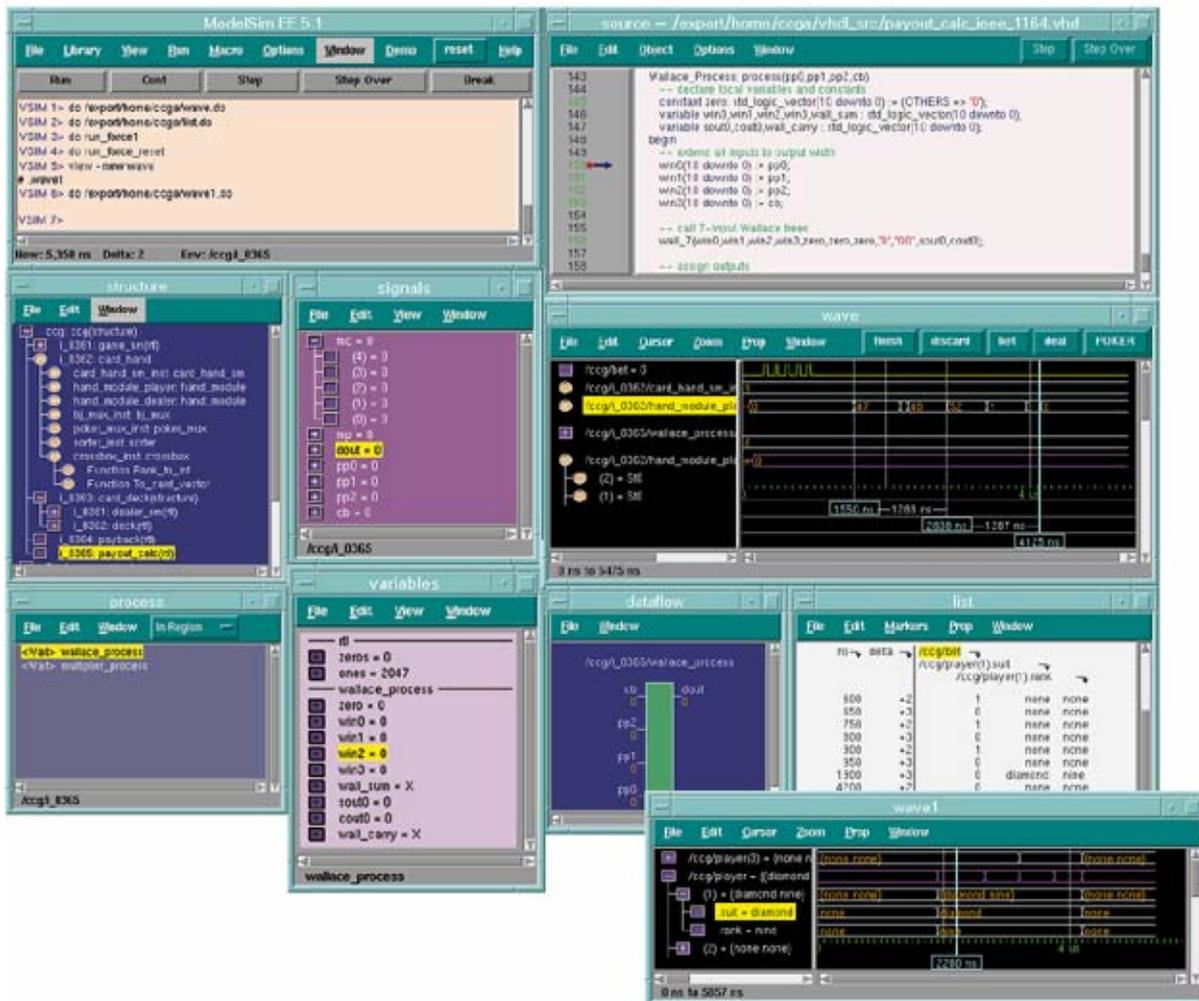
my\_sim.do

```
cd c:\mydir  
vlib work  
vcom counter.vhd  
vsim counter  
view *  
do stimulus.do
```

stimulus.do

```
add wave /clk  
add wave /clr  
add wave /load  
add wave -hex /data  
add wave /q  
force /clk 0 0, 1 50 -repeat 100  
force /clr 0 0, 1 100  
run 500  
force /load 1 0, 0 100  
force /data 16#A5 0  
force /clk 0 0, 1 50 -repeat 100  
run 1000
```

# ModelSim 用户界面



# ModelSim 用户界面特征

- 有九个窗口: *main, structure, source, signals, process, variables, dataflow, wave, 和 list* 窗口

- 支持任何窗口的多个副本

•注意: Main窗口只允许存在一个,  
因为这个窗口控制仿真器

- 拖放

- 在一个窗口选择HDL项后, 用鼠标左键, 这些项能被从一个窗口拖和放到另一个窗口.
- HDL项可从 *Dataflow, List, Signals, Source, Structure, Variables,* 和 *Wave*窗口拖出.
- 可把它们放到 *List* 或者 *Wave* 窗口

# Main 窗口

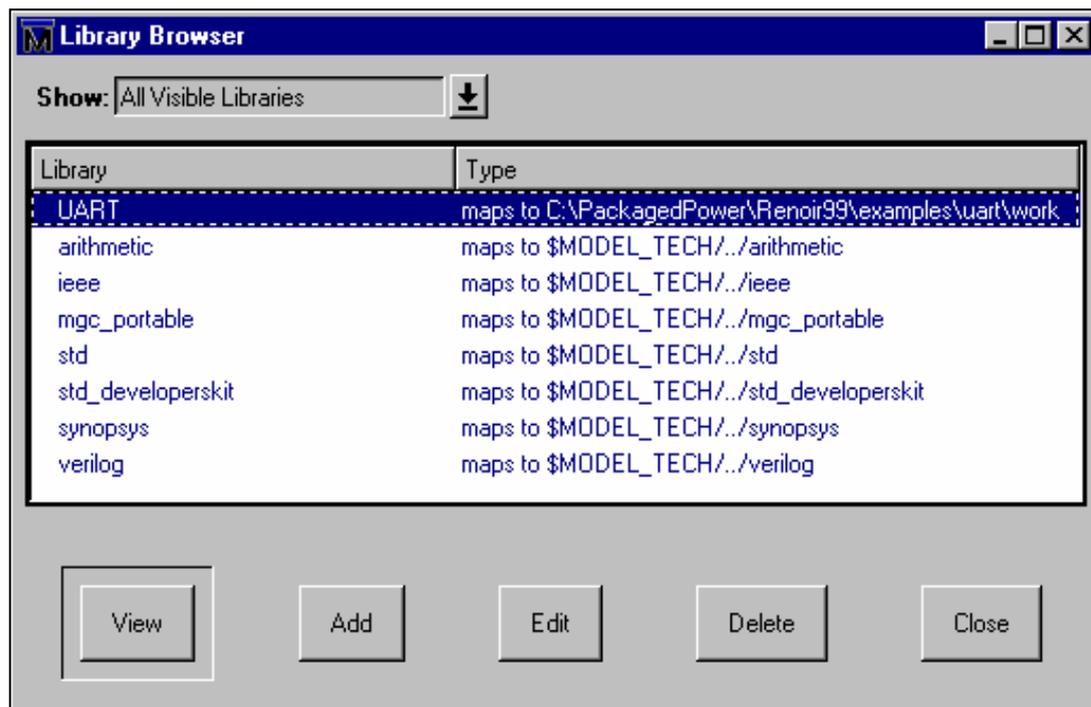
- ModelSim>      这是设计加载前的提示符
  - 能浏览帮助, 编辑库, 编辑源代码而不用调用一个设计
- VSIM>      设计加载后显示的提示符
- 告诉我们仿真器的行为动作
  - 命令
  - 信息
  - 声明



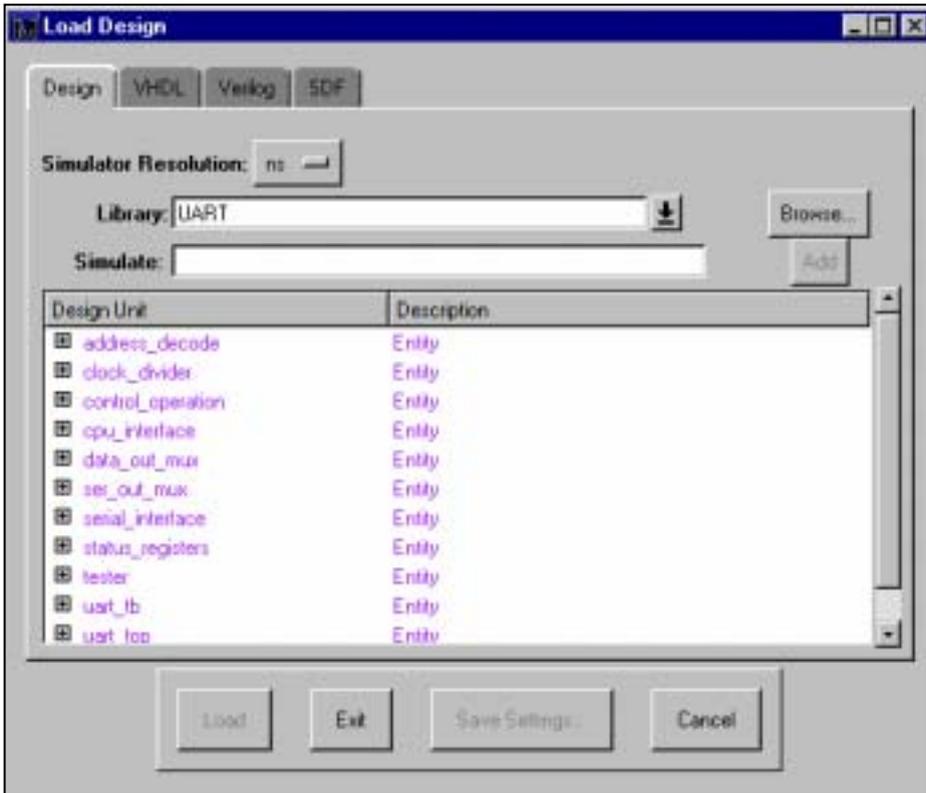
# Main 窗口: 库

## ■ Design Menu -> Browse Libraries

- 加入新库或编辑已有的库
- 浏览和编辑库目录



# Main 窗口: 启动窗口



- 用于选择要加载的设计
- 选择:
  - 时间分辨率
    - Supports multipliers of 1, 10, and 100 each time scale.
  - 包含顶级设计单元的库
  - 顶级设计单元
    - Entity/Architecture对
    - 构造
    - 模块

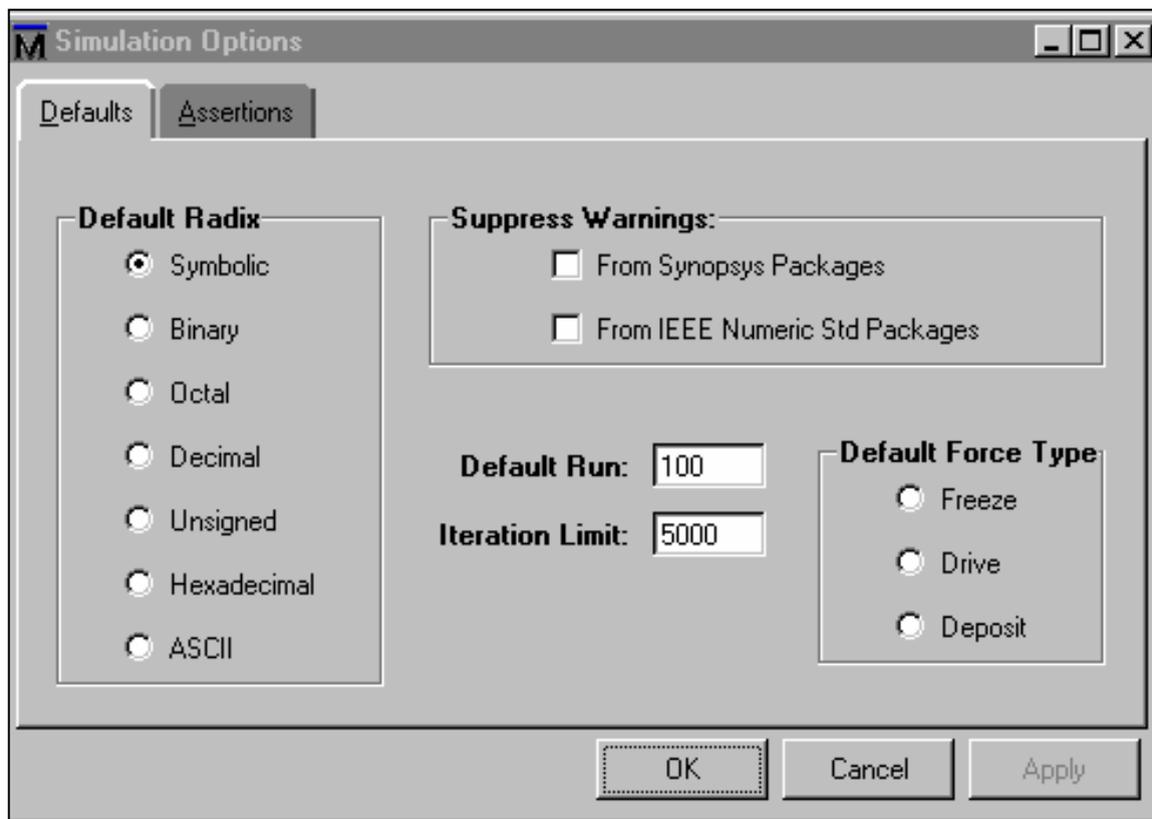
## ■ *Design Menu*

*-> Load New Design*

```
COM) vsim <library_name> <top_level_design_unit>
```

# Main 窗口: 选项

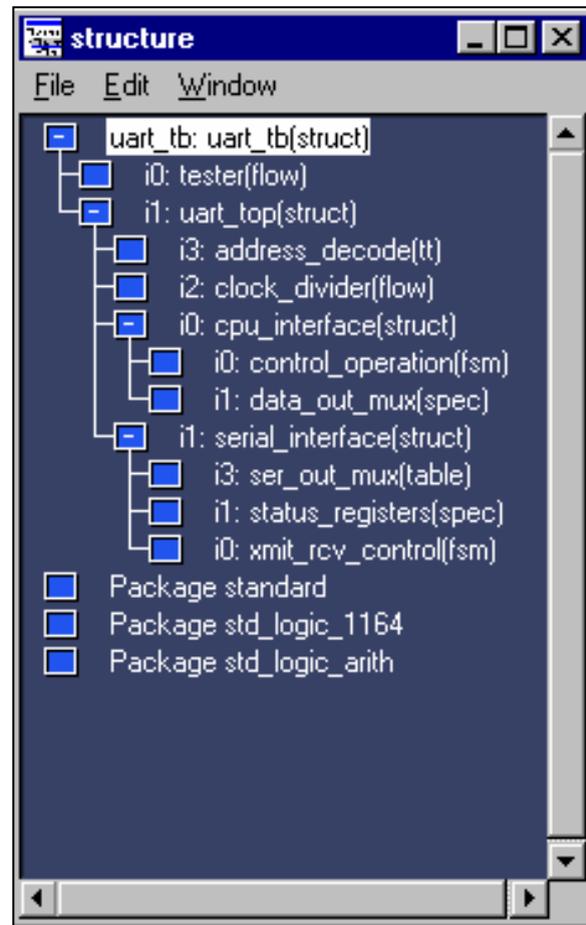
## ■ Options Menu -> Simulation Options



# Structure 窗口

## ■ 设计的结构多层浏览

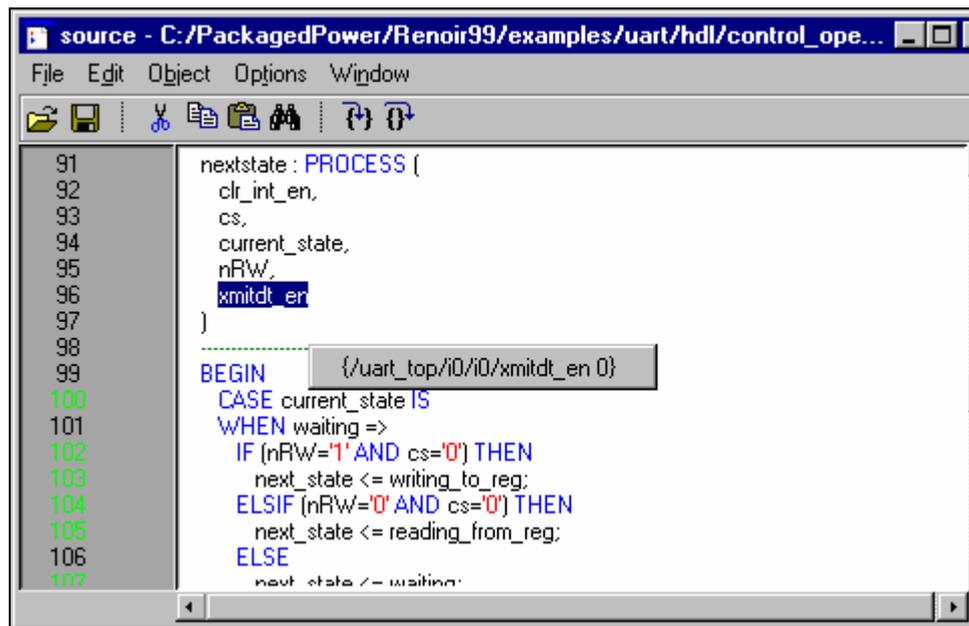
- VHDL (□) - Package, component instantiation, generate and block statements
- Verilog (○) – module实例, named fork, named begin, task, 和 function
- Instantiation label, entity/module, architecture
- 成为当前层 for *Source* 和 *Signals* 窗口, updates *Process* 和 *Variables* 窗口



COM) view structure

# Source 窗口

- 从Structure窗口选择
- Options 菜单 (源代码的控制浏览)
- Color-coded
  - 注释, 关键字, 字符串, 数字, 执行行, 标识符, 系统任务, 文本
- 完全的编辑能力
  - 保存 编译和重启
- 拖放
- 描述/检查
  - VHDL – 信号, 变量和常数



```
91  nextstate : PROCESS (  
92  clr_int_en,  
93  cs,  
94  current_state,  
95  nRW,  
96  xmitdt_en  
97  )  
98  -----  
99  BEGIN {/uart_top/iO/iO/xmitdt_en 0}  
100  CASE current_state IS  
101  WHEN waiting =>  
102  IF (nRW='1' AND cs='0') THEN  
103  next_state <= writing_to_reg;  
104  ELSIF (nRW='0' AND cs='0') THEN  
105  next_state <= reading_from_reg;  
106  ELSE  
107  next_state <= waiting;
```

# Source 窗口

## ■ 描述

- 显示所选的HDL项的信息

## ■ 检查

- 显示所选HDL项当前仿真值

- 1) 高亮信号,  
变量, 常数,  
线网, 或寄存器
- 2) 右击鼠标并选择 Now  
或  
*Object Menu ->*  
*Examine/Description*

```
91  nextstate : PROCESS (  
92  clr_int_en,  
93  cs,  
94  current_state,  
95  nRW,  
96  xmitdt_en  
97  )  
98    
99  BEGIN  
100  CASE current_state IS  
101  WHEN waiting =>  
102  IF (nRW='1' AND cs='0') THEN  
103  next_state <= writing_to_reg;  
104  ELSIF (nRW='0' AND cs='0') THEN  
105  next_state <= reading_from_reg;  
106  ELSE  
107  next_state <= waiting;
```

# Process窗口



## ■ 显示外部和内部的处理

### ■ *View -> Active*

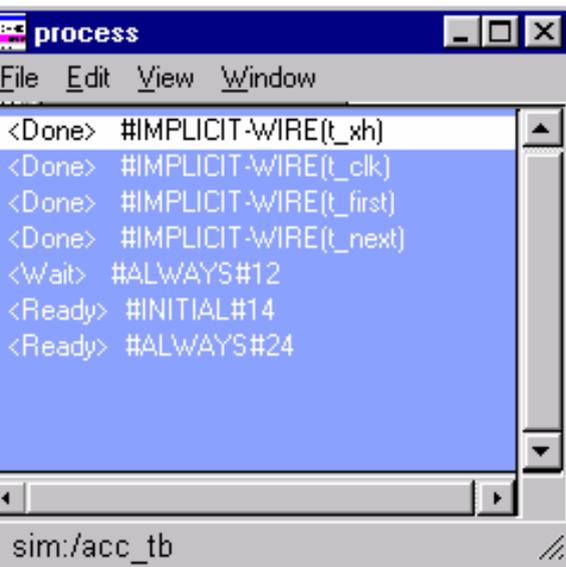
- Shows all processes scheduled for execution during the current simulation cycle

### ■ *View -> In Region*

- Shows names of all processes in the region selected in the Structure window

COM) *view* process

# Process窗口



```
process
File Edit View Window
<Done> #IMPLICIT-WIRE(t_xh)
<Done> #IMPLICIT-WIRE(t_clk)
<Done> #IMPLICIT-WIRE(t_first)
<Done> #IMPLICIT-WIRE(t_next)
<Wait> #ALWAYS#12
<Ready> #INITIAL#14
<Ready> #ALWAYS#24
sim:/acc_tb
```

## ■ 指示器

- <Ready>
  - Process is scheduled to be executed
- <Wait>
  - 处理正等待 VHDL 信号或Verilog 线网改变或等待超时
- <Done>
  - Process has executed a VHDL wait statement without a time-out or sensitivity list

COM) view process

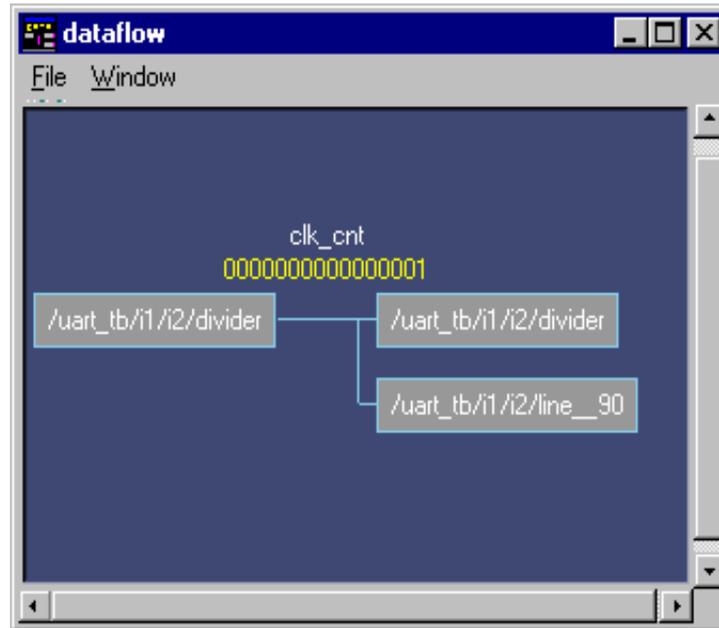
# Signals 窗口

- 紧跟Structure窗口
  - 显示Structure窗口的当前层HDL项的名称和值
- 排序 - 升序, 降序或声明顺序
- 层次 - (+)可展开的, (-)已展开的
- VHDL 信号
- Verilog 线网, 寄存器变量和已命名的时间
- “拖放”
  - Wave和List窗口
- Force
  - 用于激励
- Filter
  - 为浏览选择信号类型  
(输入, 输出, 内部, 等)



COM) *view* signals

# Dataflow 窗口



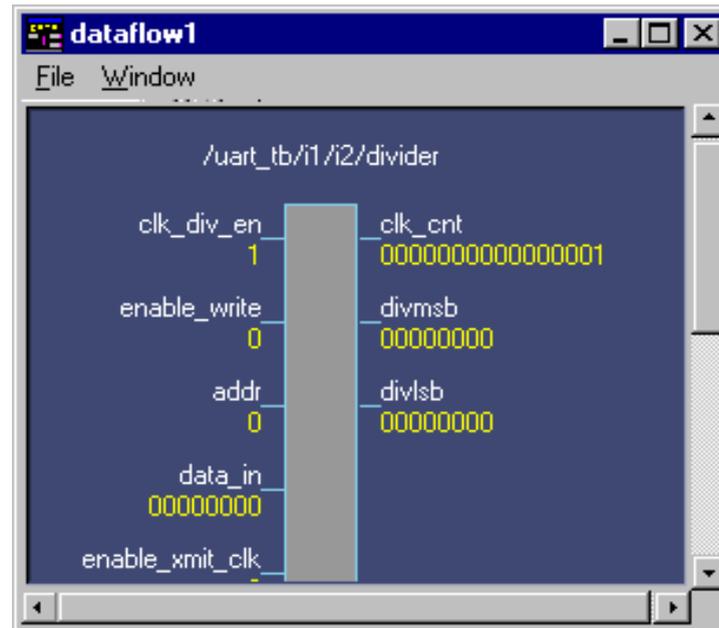
## ■ VHDL信号或Verilog线网的图形描绘

### — 信号或线网在窗口中央

- Processes that drive signal or net on the left
- Processes that read the signal or are triggered by the net on the right

COM) *view* dataflow

# Dataflow 窗口



## ■ Processes

- Signals read by or nets that trigger the process on the left
- Signals or nets driven by the process on the right

## ■ Single-Double clicking update of signals and processes

# Wave 窗口

- 用波形浏览仿真结果的图形化的历史记录
  - VHDL – 信号和过程变量
  - Verilog – 线网, 寄存器变量, 已命名事件
- 对于更多的逻辑信号多个波形窗口用于更多的逻辑信号
- 改变信号和向量的基数已方便查看
- 打印波形

**Verilog** (points to the code list)

**Adjustable spacing between signals** (points to the gap between signal traces)

**VHDL Variable** (points to a signal trace)

**Color-coding on a signal-by-signal basis** (points to a multi-colored signal trace)

**Cursor Measurement** (points to a measurement box on the waveform)

**缩放菜单** (Zoom menu)

**拖放** (Drag and drop)

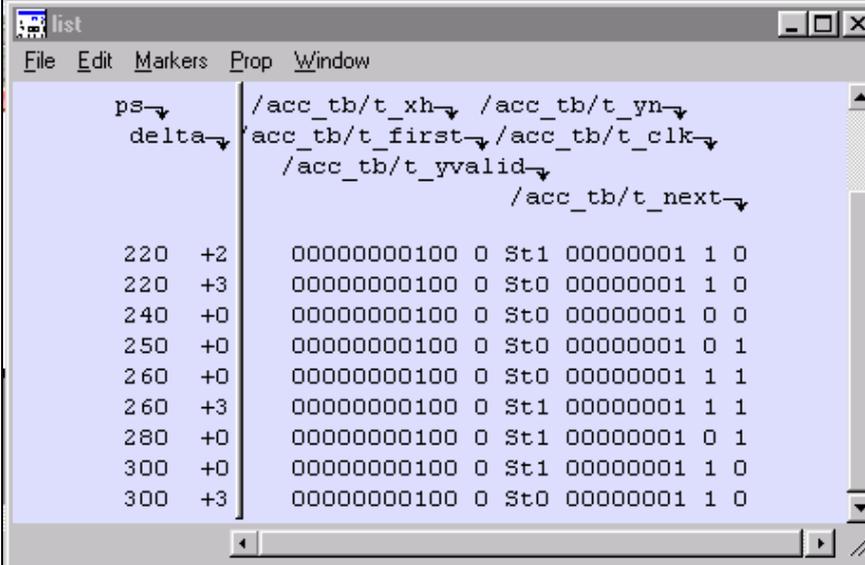
**项格式化** (Item formatting)

**多个指针** (Multiple pointers)

强大的编辑和查找能力

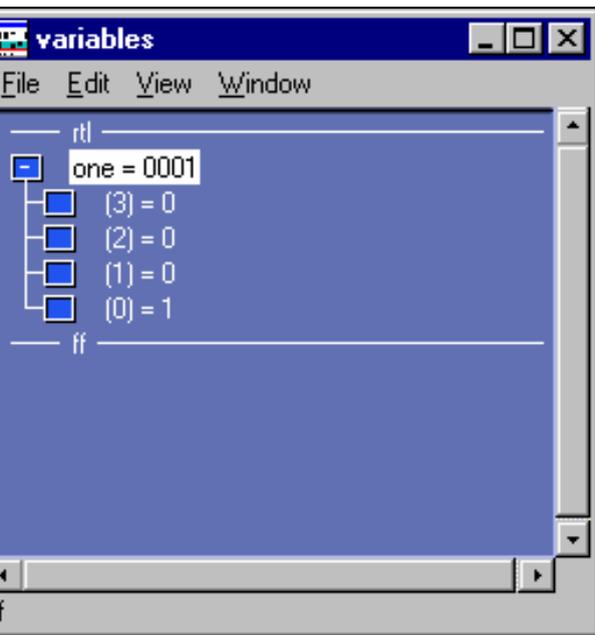
# List 窗口

- 用表格显示仿真结果
  - VHDL – 信号和过程变量
  - Verilog – 线网和寄存器变量
- 从这个窗口或到这个窗口“拖放”
- 编辑功能 – 查找
- 建立用户定义的总线 - Edit>Combine
- 设置触发和选通
- Write List - Tabular, Event or TSSI
- Marker - Add, Delete or Goto



```
list
File Edit Markers Prop Window
ps→ /acc_tb/t_xh→ /acc_tb/t_yn→
delta→ /acc_tb/t_first→ /acc_tb/t_clk→
/acc_tb/t_yvalid→
/acc_tb/t_next→
220 +2 00000000100 0 St1 00000001 1 0
220 +3 00000000100 0 St0 00000001 1 0
240 +0 00000000100 0 St0 00000001 0 0
250 +0 00000000100 0 St0 00000001 0 1
260 +0 00000000100 0 St0 00000001 1 1
260 +3 00000000100 0 St1 00000001 1 1
280 +0 00000000100 0 St1 00000001 0 1
300 +0 00000000100 0 St1 00000001 1 0
300 +3 00000000100 0 St0 00000001 1 0
```

# Variables 窗口



- 列出 HDL 项的名称
  - VHDL – 常数, generics 和变量
  - Verilog – 寄存器变量
- 到当前过程的路径被显示在左下
- 树层次 - (+)可展开的, (-)已展开的
- 分类 – 升序, 降序或声明顺序
- 改变 – 选择HDL项改变值
- 浏览 – 在Wave 或 List 窗口 或 log 文件的项
  - 选择变量或层中的变量

COM) *view* variables

# ModelSim 用户界面特征(继续...)

## ■ 自动更新窗口

- Dataflow 窗口: 当一个进程被选到这个窗口的中央, *Process*, *Signals*, *Source*, *Structure*, 和 *Variables* 窗口会被更新.
- Process 窗口: 当一个进程被选择, *Dataflow*, *Signals*, *Structure*, 和 *Variables*窗口被更新.
- Signals 窗口: 当Signals窗口被选择, *Dataflow* 窗口是唯一被更新的窗口.
- Structure 窗口: 当从你的设计结构中层次浏览中的一个被选择, *Signals* 和 *Source* 窗口将自动更新.

# ModelSim 用户界面(继续...)

## ■ 查找名称或搜索值

- 除两个窗口外其他窗口都允许用户通过菜单 *Edit>Find* 查找项名称。只有 *Main* 和 *Dataflow* 窗口没有这个功能。
- 在 *List* 和 *Wave* 窗口, 能通过 *Edit>Search* 搜索HDL项值。

## ■ 排序HDL 项

- 用 *Edit>Sort* 菜单选项, HDL项能被排序(按升序,降序, 或声明顺序)。
- 缺省, 这些项按被声明的顺序排序。

# ModelSim 用户界面特征 (继续...)

## ■ 多个窗口副本

- 从主窗口, 用 *View>New* 菜单选项建立额外的相同窗口的副本.

# 设计调试

## ■ 什么时候调试？

- 编译失败
- 不正确或意外的仿真结果

## ■ Model*Sim* 调试能力举例

- 信号监视
- 断点

# 监视更多的信号

给跟踪加入附加的信号或变量

- 在Structure 窗口选择层
- 从Source, Signals 或Variables窗口 “拖放”到:
  - Wave窗口
  - List 窗口

# 断点

## ■ 支持两种类型的断点

### – 在源代码窗口设置断点

- Toggles – 再次点击删除断点
- 没有断点数量的限制
- 用 bp 命令

*bp* <file\_name> <line#>

### – 条件断点

- *when* <condition> <action>
- *when* {b=1 and c/=0}
- 与VHDL信号和Verilog 线网和寄存器一起使用
- 也可用 bp 命令

*bp* <file\_name> <line#> {if{\$snow/=100}then{cont}}

# ModelSim 项目

- 一个集合:
  - 根目录和子目录
  - HDL 仿真文件
  - 库
  - 仿真器设置
- 允许你保留当前工作
- 多用 UI
- 在项目目录里保存为 .MPF 文件
  - 仍然支持.INI文件
- 项目操作
  - File -> New / Open / Delete

# modelsim.ini文件

- Model*Sim*使用 ASCII文件, 由用户控制
- 在Model*Sim*的安装目录一个缺省文件被提供
- modelsim.ini被编译器和仿真器使用
  - 存有初始信息
    - 库定位
    - 启动文件的定位
    - Model*Sim*其他缺省设定
- Model*Sim* 按下列顺序搜索modelsim.ini:
  1. 环境变量Environment variable called MODELSIM which points directly to the modelsim.ini file to be used
  2. A file called modelsim.ini located in the current working directory
  3. 缺省的 modelsim.ini文件在ModelSim软件安装树目录里

# startup.do 文件

- 一个 DO 脚本自动执行通过 *vsim*

- 一个例文件 startup.do 就象这样:

```
view source
```

```
view structure
```

```
view wave
```

```
do wave.do
```

- 调用一个启动文件, 未注释 (移去“;”)下面 modelsim.ini 文件行给 do 文件提供路径:

```
;Startup = do /path .... /startup.do
```