Applications Note

HDL Simulation FPGA Design Methodology

October 15, 1998

Revision 1.0

Overview

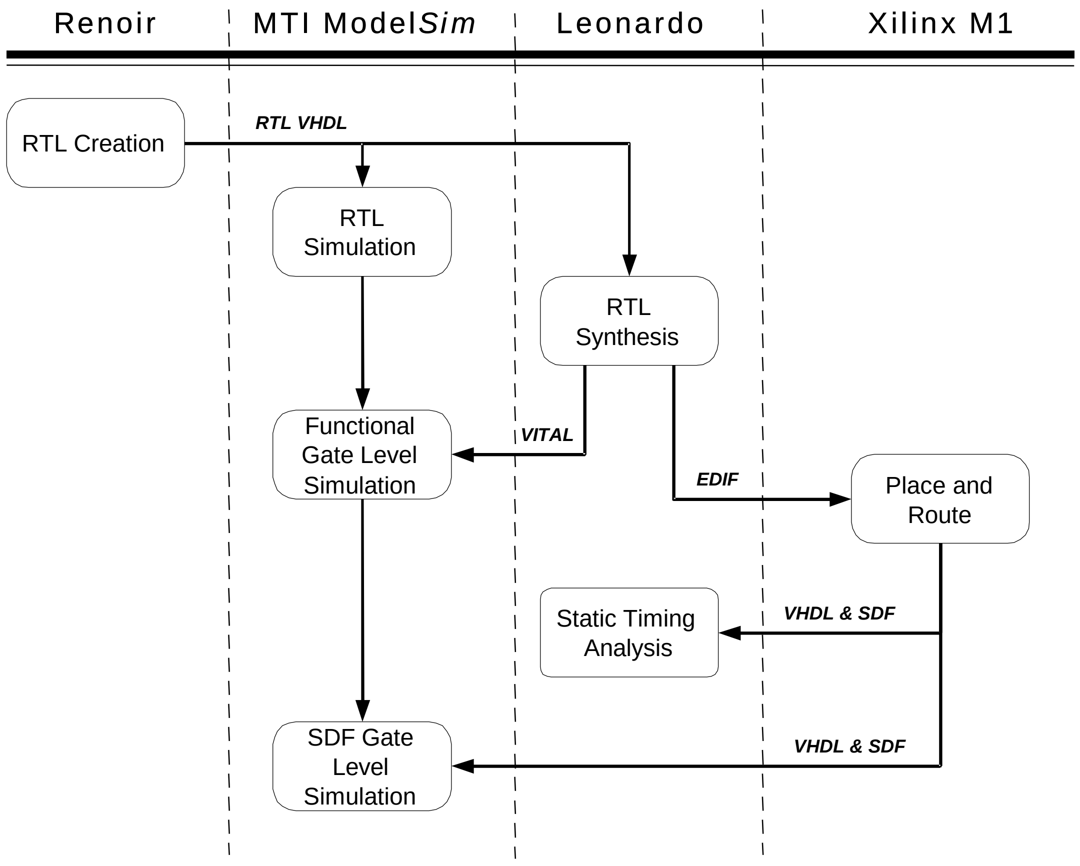
ModelSim performs pre-synthesis RTL simulation and post-place and route gate-level timing simulation with SDF backannotation of timing. ***Leonardo*** performs architecture specific synthesis and optimization for all Xilinx devices. ***Xilinx Alliance Series*** performs placement and routing of the synthesized netlist. This applications note discusses methodology and optimization settings for Leonardo, Alliance Series and ModelSim when targeting Xilinx devices. The intent of this appnote is not to exhaustively explore all the different options in the Leonardo, Alliance Series and Model*Sim* toolsets but to present a single methodology that works. For information beyond the scope of this document, refer to the following web pages:

www.exemplar.com	***Exemplar Logic***
www.model.com	***Model Technology***
www.xilinx.com	***Xilinx***

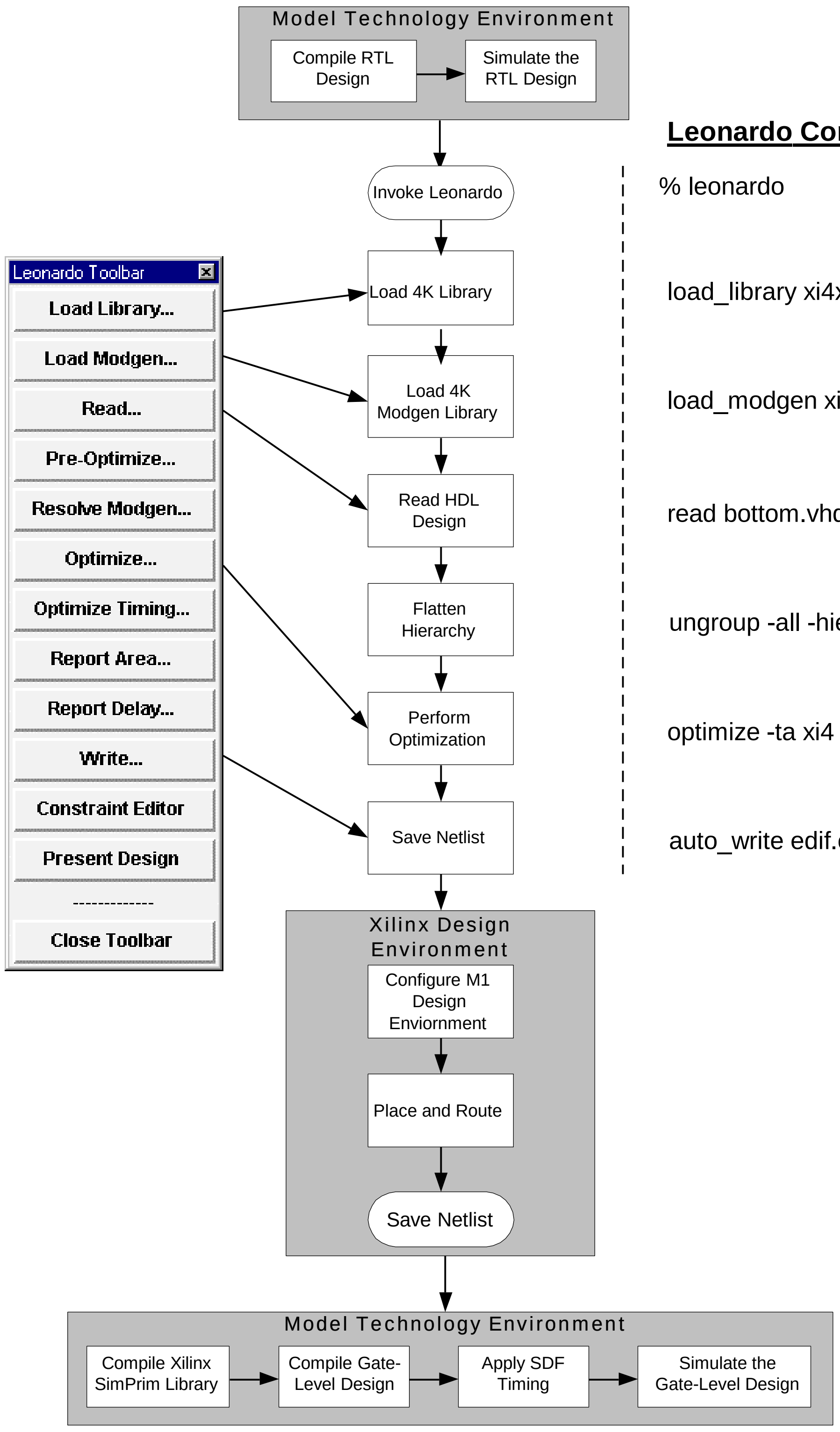
When targeting Xilinx devices, Leonardo will map the design into Xilinx lookup tables. Both Galileo Extreme and Alliance Series offer configuration options that dictate how this mapping takes place. In some cases both tools can perform the same functions but, as we'll see, with different results.

The Alliance Series allows the creation of VHDL or Verilog of the placed and routed design along with timing information in a SDF file. This gate-level timing design can then be compiled (along with the Xilinx SIMPRIM library) and simulated in Model*Sim*.

Model Technology , Exemplar & Xilinx Toolflow Overview



Synthesis Process Design Flow



Leonardo Commands

```
% leonardo

load_library xi4xl

load_modgen xi4e

read bottom.vhd middle.vhd top.vhd

ungroup -all -hierarchy

optimize -ta xi4 -area

auto_write edif.out
```

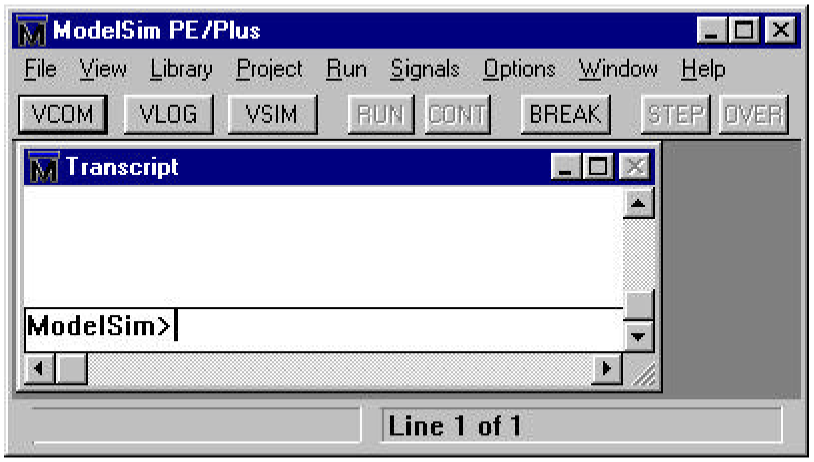
Example Session

Example Model*Sim* RTL Simulation Session

This section describes the basic steps to compiling and simulating the pre-synthesis RTL design. It is in this step that the functionality of the design is verified prior to synthesis.

1) Invoke Model*Sim*

   % vsim **-or-** **Programs > Model Tech > ModelSim**


ModelSim PE/Plus
File View Library Project Run Signals Options Window Help
VCOM VLOG VSIM RUN CONT BREAK STEP OVER
Transcript
ModelSim>
Line 1 of 1

2) Set Model*Sim* to the directory where the RTL design resides.

   ModelSim> **cd c:\mydesign** **-or-** **File > Directory**

3) Create a working library to store the compiled RTL design.

   ModelSim> **vlib work** **-or-** **Library > New**

4) Compile the RTL design.

   ModelSim> **vcom bottom.vhd middle.vhd top.vhd**

   -or- **VCOM** button

5) Start the Model*Sim* simulator.

   ModelSim> **vsim top** **-or-** **VSIM** button

6) View all the Model*Sim* debugging windows.

   VSIM> **view *** **-or-** **View > All**

7) Wave and list signals of interest in the design.

   VSIM> **wave /*** -- Adds all top level signals to the wave window
   VSIM> **list /*** -- Adds all top level signals to the list window

8) Unless you have a VHDL testbench which stimulates the RTL design, you will need to force the inputs of the RTL design.

   VSIM> **force /clk 0 @ 0 ns, 1 @ 50 ns –repeat 100 ns**
   VSIM> **force /input 0**

9) Run the simulation and analyze the information in the Model*Sim* debugging windows.

   VSIM> **run –all** **-or-** **Run > Run Forever**

Example Leonardo Session

This section provides a detailed example of using Leonardo with the Xilinx Alliance Series toolset targeting the XC4000XL device family. All commands are easily accessible via the toolbar, pulldown menus or the Xilinx specific flow guide. This example will demonstrate Leonardo shell commands.

10) Invoke leonardo

    % leonardo

    LEONARDO{1}

11) Load the Xilinx xi4e library. This will load cell data only

    LEONARDO{1} **load_library xi4xl**

12) Load the Xilinx Modgen library. Modgen is a library of handcrafted implementations for all the inferred design elements. This includes operators, RAMs and counters. There are typically multiple architectures for each element. If this library is not loaded Leonardo will use a generic Modgen library which will not be able to take advantage of Xilinx specific cells.

    LEONARDO{2**} load_modgen xi4e**

13) Read in the HDL files. VHDL design files must be listed in their bottom-up order. Verilog users enjoy "auto-top detection" which means that Leonardo will automatically detect the top-level module from files listed in any order.

Note*: Leonardo uses file suffixes to figure out file formats; VHDL files = .vhd, .vhdl; Verilog files = .v, .ver; EDIF files = .edn, .edf, .edif.*

    LEONARDO{3} **read bottom.vhd middle.vhd top.vhd**

14) Flatten the design. Hierarchical boundaries prevent or limit important optimizations from occuring. Sometimes there are good reasons to preserve hierarchy, i.e., design size or to separate out speed critical blocks. Only a minimum of hierarchy should be kept. It is recommended to have no more than 50K gates per hierarchical block.

    LEONARDO{4} **ungroup -all -hierarchy**

15) Perform optimization. Leonardo can perform both area and timing optimization. In this example we will be performing optimization to achieve the smallest design. Additionally, the effort level can be specified. Quick performs 1 pass and standard performs 4 passes and will take 4 times longer to complete.

    LEONARDO{5} **optimize -ta xi4e -area -effort quick**

16) Generate area and timing reports. The optimization runs will display a single area and worst case timing number. Reports are only necessary if more information is required.

    LEONARDO{6} **report_area**

    LEONARDO{7} **report_delay**

17) Generate an EDIF netlist for Alliance Series. A Netlist pre-processor is built into Leonardo. Because the Xilinx XC4000E technology is specified the correct netlist pre-processing will take place.

Example Alliance Series Place and Route Session

The Xilinx graphical tools are designed to behave, look, and feel like the XACT 6.0 tools. So despite the fact that the core technology algorithms have been redesigned, the graphical tools allow users to run the software in the same way as previous PC versions. For PC customers, the learning curve should be short.

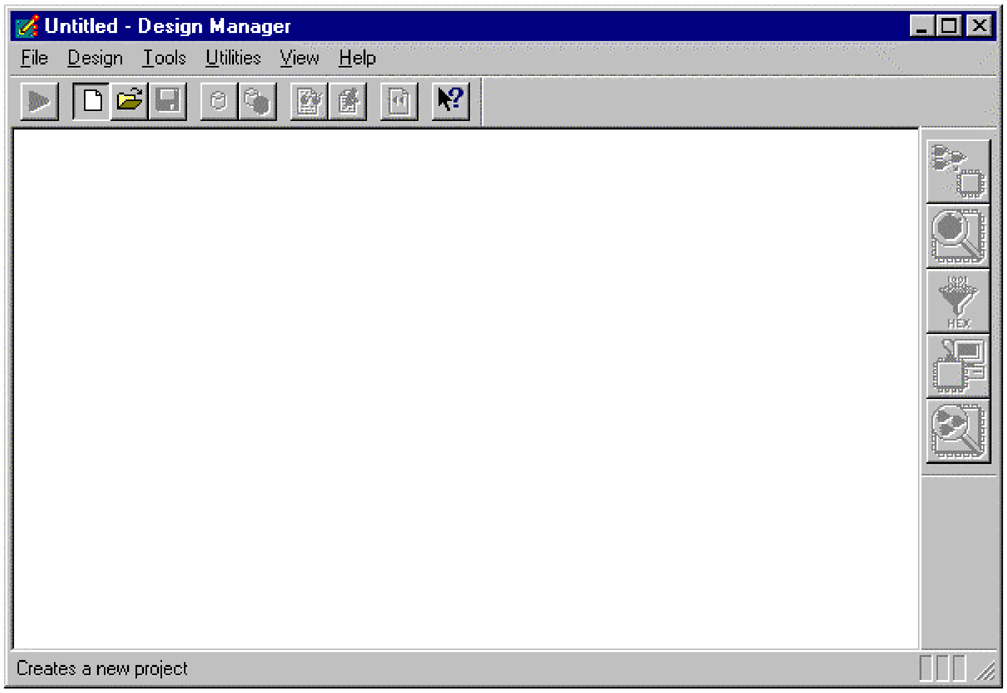
The Design Manger (DM) is the graphical tool that manages the design files that are created during design implementation. The DM also provides push button access to the following Xilinx tools: Flow Engine, Prom File Formatter, Timing Analyzer, Hardware Debugger, and JTAG Programmer.

Start the Design Manager from the Windows 95 or NT desktop by executing the command:

Selecting Start > **Programs** > **Xilinx** > **Design Manager**

From a shell invoke Design Manager by typing:

dsgnmgr

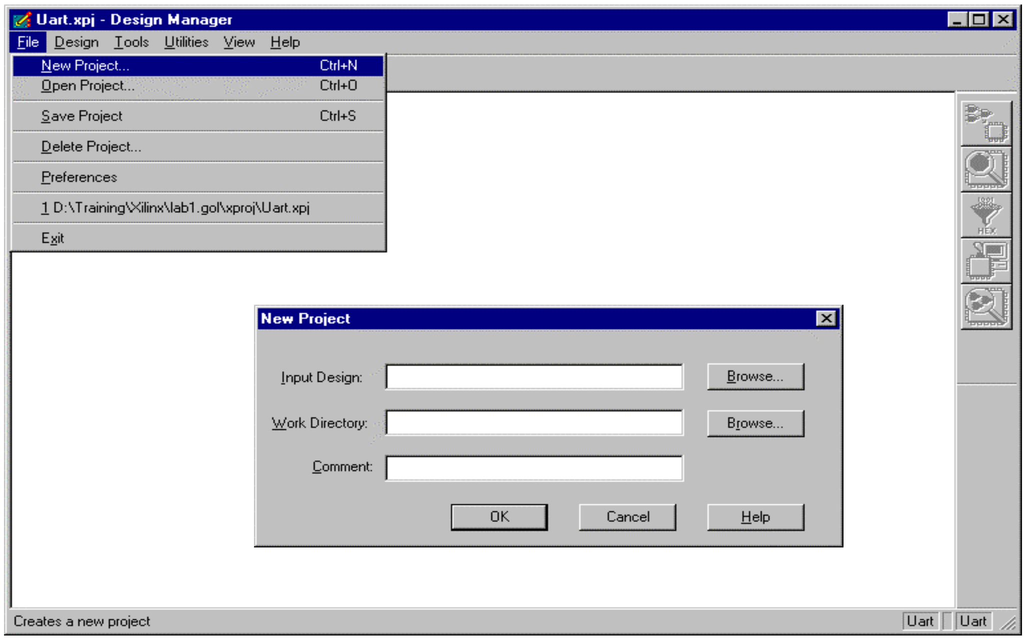

Untitled - Design Manager
File Design Tools Utilities View Help
HEX

Creates a new project

1) Create a New Project in Alliance Series. From the Design Manager toolbar, execute the pulldown menu command

```
FILE > New Project
```

Push the ***"input design"*** button and navigate to the EDIF file generated by Leonardo. This file should have the extension, ".edn".



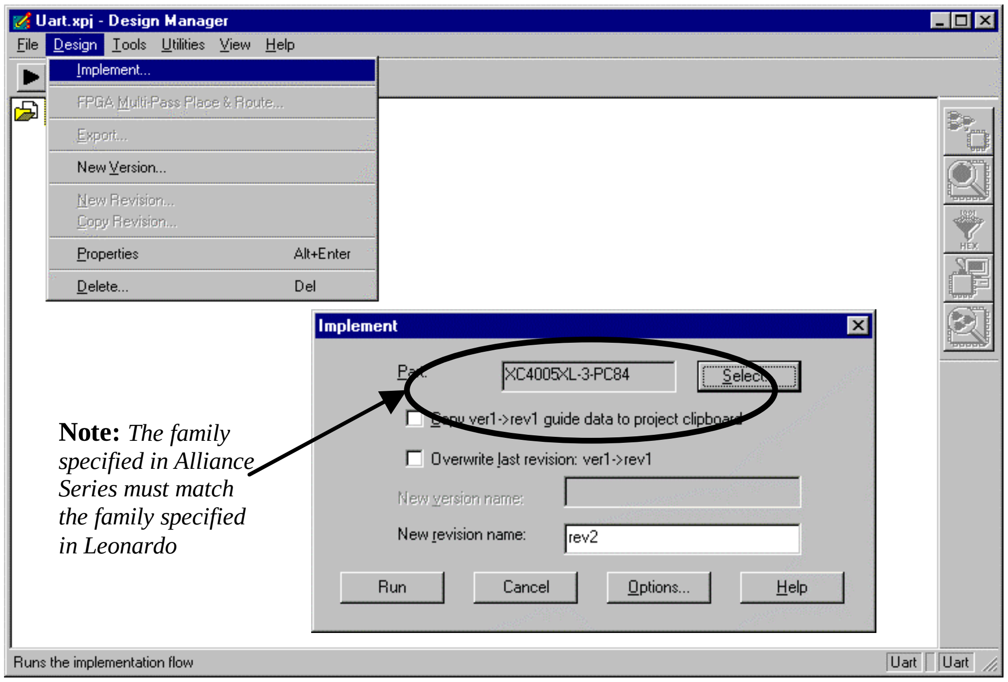
2) Perform "Implement" on the design. From the pulldown menu, execute the command,

```
Design > Implement
```

- Select a specific Xilinx part
- Push the *Select* button. In the “Part Selector” dialog and choose the appropriate member, speed and package combination from those available for the XC4000XV family

Note: *If the Leonardo command, "generate_timespec" is issued after optimization and before saving the EDIF netlist then clock frequency timing data is included in the EDIF netlist.*

3) Click "*OK, but do not hit "Run" on the Implement dialog box*

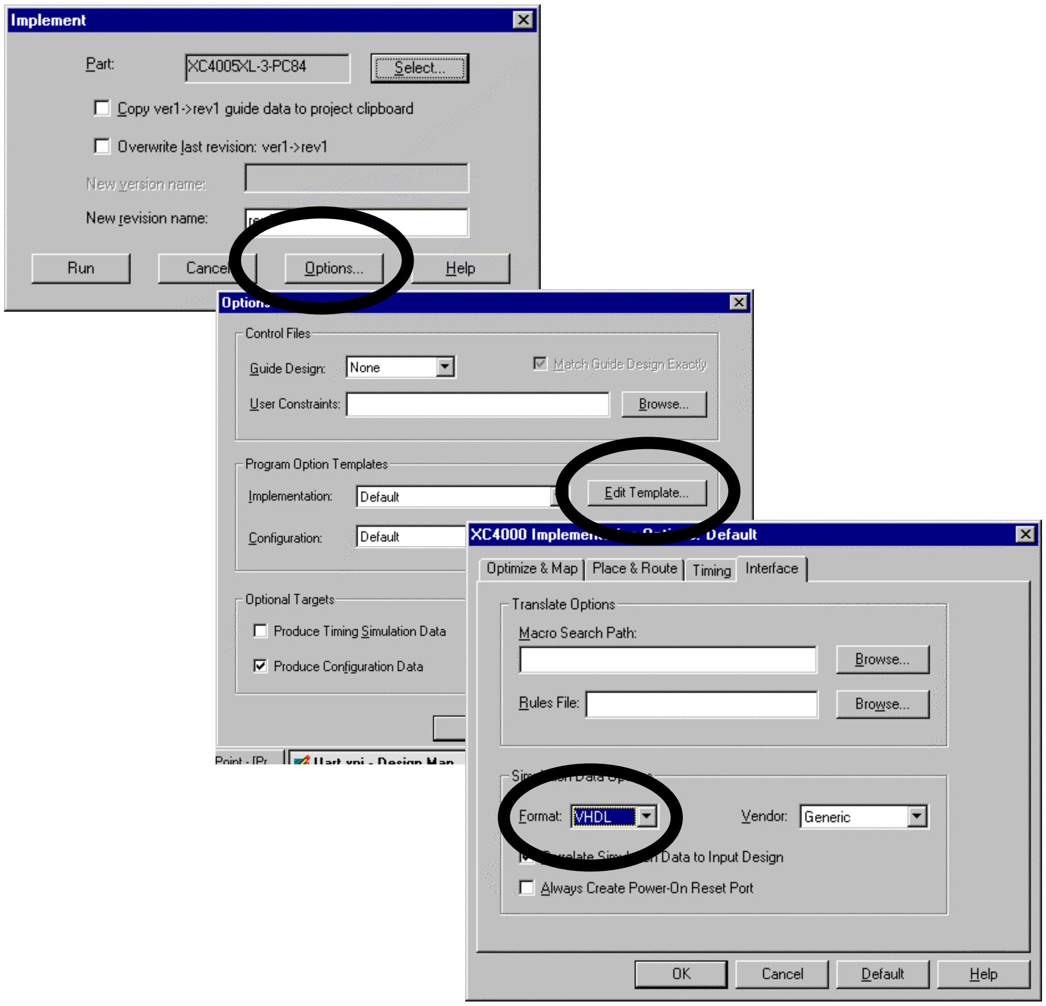


Note: *The family specified in Alliance Series must match the family specified in Leonardo*

Note: *The first step in implementing a design is the selection of a target device. If a valid PART has been specified in Leonardo, it will be pre-selected in the Part selection dialog box. For designs that do not have the PART specified in the netlist (or the Design Manager is unable to detect its presence), the user must identify the target part using this dialog box. Users may define the part in Leonardo by setting the "part" variable, i.e.,* ***"set part xc4005xl-3-PC84".*** *Setting the part variable is a step of convenience and will not effect optimization results*

Setup Alliance Series to generate VITAL VHDL Simulation Model

1) From the Implement dialog box select the ***"Options"*** button. This will bring up the "Options" dialog box.
2) In the "Optional Targets" field, check the box labeled***, "Produce Timing Simulation Data"***
3) In the "Program Option Templates" field, select the ***"Edit Template***" button for "Implementation". This will bring up the "XC4000 Implementation Options: Default" dialog box.
4) Once up Select the ***"Interface"*** tab and Set the simulation data output to "***VHDL"***
5) ***"OK"*** all the dialog boxes



Implement

Part: XC4005XL-3-PC84 Select...

☐ Copy ver1->rev1 guide data to project clipboard

☐ Overwrite last revision: ver1->rev1

New version name:

New revision name:

Run Cancel Options... Help

Options

Control Files

Guide Design: None ☑ Match Guide Design Exactly

User Constraints: Browse...

Program Option Templates

Implementation: Default Edit Template...

Configuration: Default

Optional Targets

☐ Produce Timing Simulation Data

☑ Produce Configuration Data

XC4000 Implementation Options: Default

Optimize & Map | Place & Route | Timing | Interface

Translate Options

Macro Search Path: Browse...

Rules File: Browse...

Format: VHDL Vendor: Generic

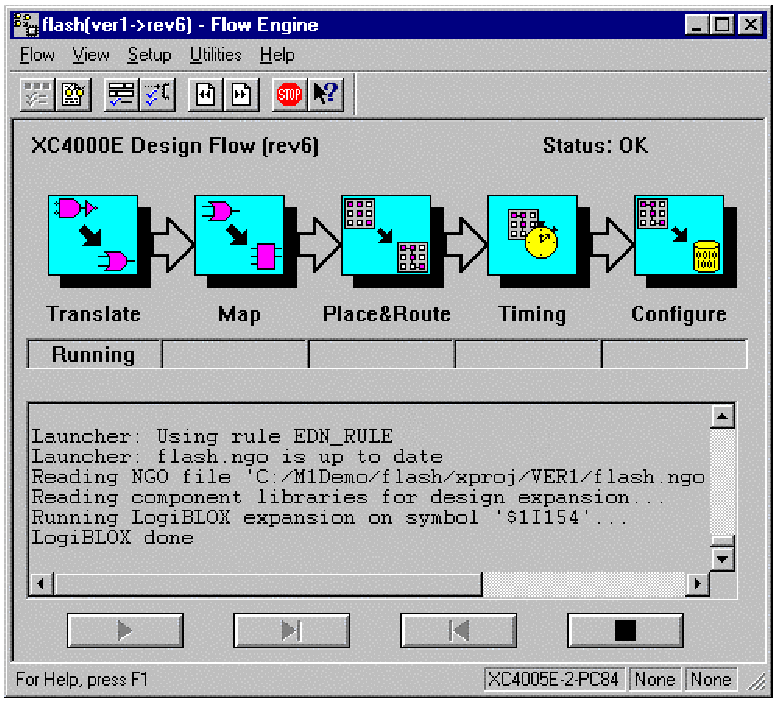
Data to Input Design

☐ Always Create Power-On Reset Port

OK Cancel Default Help

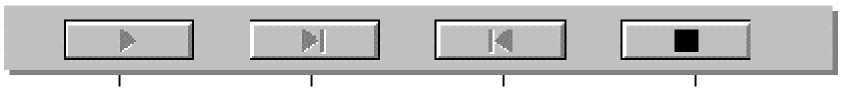
Run the "Implement" Command

To launch the implementation process, in the "Implement" dialog, push the ***"Run"*** button. This will cause the "Flow Engine" graphical interface to appear. The design is now processed through a 5 step sequence.



```
Launcher: Using rule EDN_RULE
Launcher: flash.ngo is up to date
Reading NGO file 'C:/M1Demo/flash/xproj/VER1/flash.ngo
Reading component libraries for design expansion...
Running LogiBLOX expansion on symbol '$1I154'...
LogiBLOX done
```

Notice the arrow buttons on the bottom of the window. These look like CD Player buttons and provide a similar function.



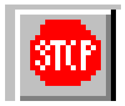
Run a step

Advance to next step

Backup to previous step

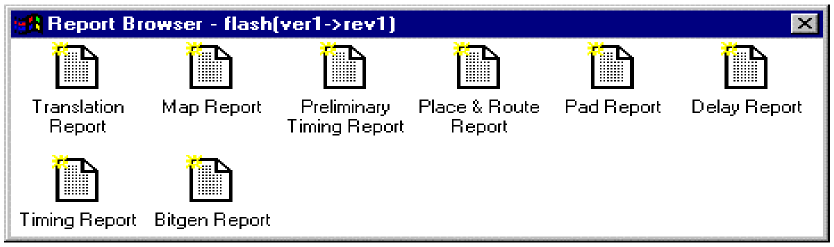
Stop

An "Implement" run can be stopped after any step by hitting the "Stop" button which appears in the form of a stop sign



Review the Alliance Series "Implement" Results

Once the Flow Engine has completed the "Implement" process, the "Implement Status" dialog box is posted. To review the processing which has occurred, review the log file. In the "Implement Status" dialog box, push the ***"View Logfile"*** button that will bring up the "Report Browser". Any report can be quickly viewed with a simple "double click" of the mouse

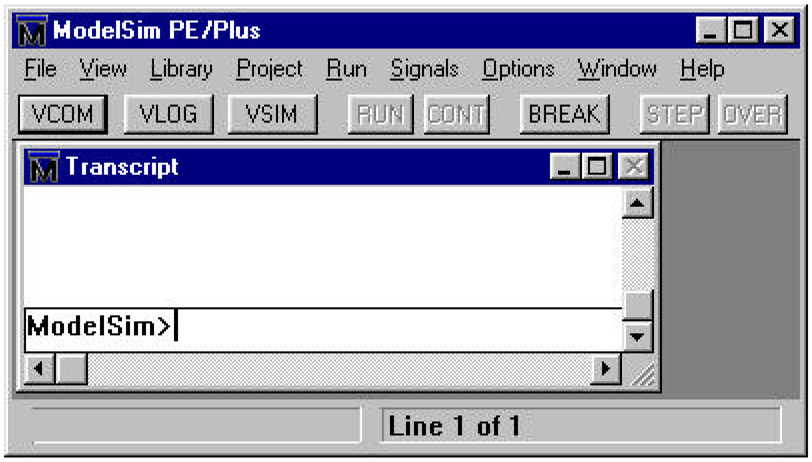


Example Model*Sim* Gate-Level Simulation (with SDF) Session

This section describes the basic steps to compiling and simulating the post-synthesis/post-place&route gate-level design with SDF timing back annotation.

1) Invoke Model*Sim*

   % vsim **-or-** **Programs > Model Tech > ModelSim**


ModelSim PE/Plus
File View Library Project Run Signals Options Window Help
VCOM VLOG VSIM RUN CONT BREAK STEP OVER
Transcript
ModelSim>
Line 1 of 1

2) Set Model*Sim* to the directory that contains the gate-level VHDL netlist (from Xilinx Design Manager).

   ModelSim> **cd c:\mygatedesign** **-or-** **File > Directory**

3) Create a working library to store the compiled design.

   ModelSim> **vlib work** **-or-** **Library > New**

4) Create a Simprim library to store the compiled Xilinx Simprim packages.

   ModelSim> **vlib simprim_lib** **-or-** **Library > New**

Note: *A separate library is not required for the Simprim packages. You could compile them into your* work *directory. If you did this, you would map the simprim library to* work *instead of* simprim_lib *as is shown in the next step.*

5) Map the Library name "simprim" to the library simprim_lib that was just created. This will allow ModelSim to know where to look when it encounters a "**library SIMPRIM;"** statement in the VHDL design.

   ModelSim> **vmap simprim simprim_lib**

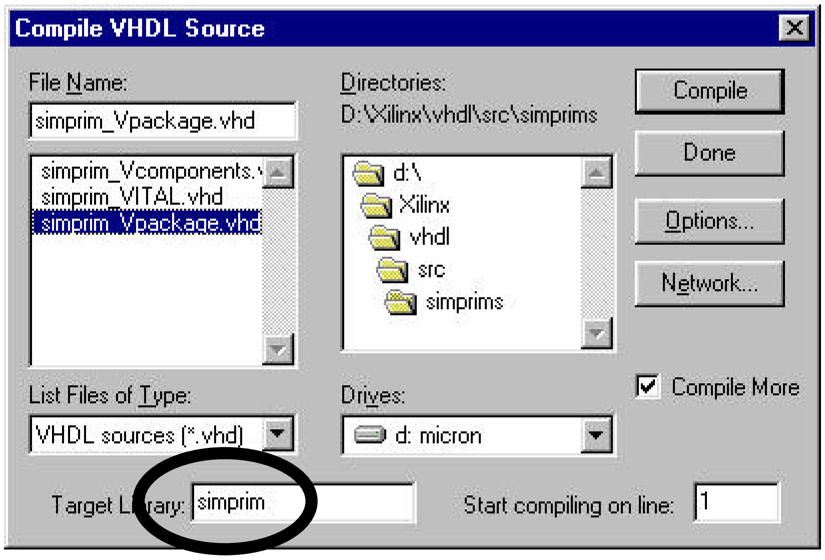
6) Compile the Simprim packages into the simprim_lib library.

ModelSim> **vcom -work simprim –explicit **
<Xilinx dir>\vhdl\src\simprims\simprim_Vpackage.vhd

ModelSim> **vcom -work simprim –explicit **
<Xilinx dir>\vhdl\src\simprims\simprim_VITAL.vhd

ModelSim> **vcom -work simprim –explicit **
<Xilinx dir>\vhdl\src\simprims\simprim_Vcomponents.vhd

This step is easier using the ModelSim VCOM button which brings up the following dialog. Make sure you compile in the proper order (Vpackage, VITAL, Vcomponents). Note the Target Library setting of *simprim* instead of *work*.



7) Compile the VHDL netlist created by the Xilinx Design Manager.

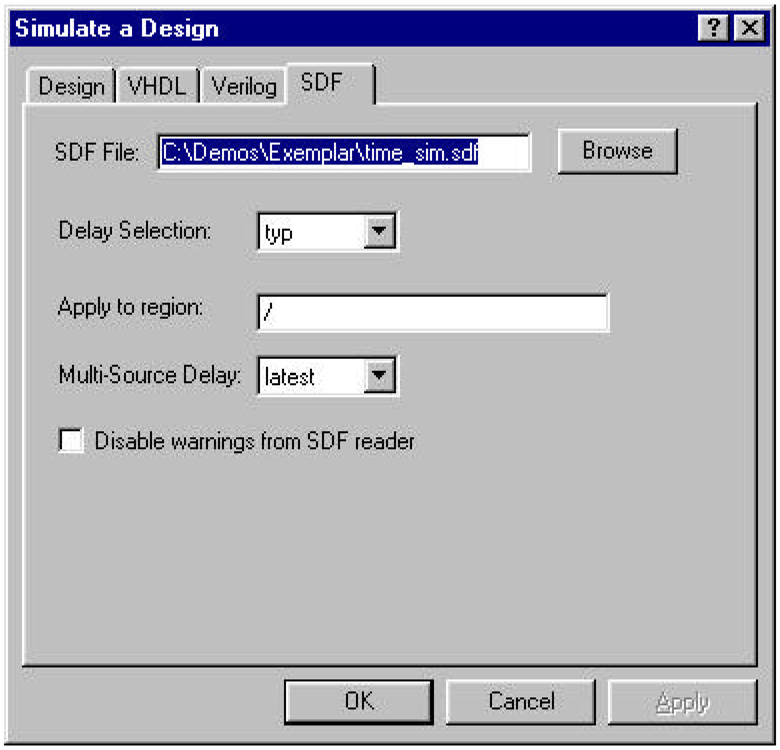
ModelSim> **vcom time_sim.vhd** **-or-** **VCOM** button

Note: *The place and routed* time_sim.vhd *gate-level design uses the* IEEE & SIMPRIM *VHDL libraries. The IEEE library comes pre-built in the ModelSim simulator. The SIMPRIM library was built in step 6 above. If the SIMPRIM libraries were not successfully compiled or if the library name* SIMPRIM *was not properly mapped to the SIMPRIM_LIB library above in step 5, then the ModelSim compiler would issue error messages complaining about '*Library simprim not found.*"*

8) Start the Model*Sim* simulator applying the sdf information in the file tim_sim.sdf to the root level of the design (/).

   ModelSim> **vsim –sdftyp /=time_sim.sdf top**

   Or use the ModelSim VSIM button to start the simulator and apply the SDF info:



Simulate a Design

SDF File: C:\Demos\Exemplar\time_sim.sdf

Delay Selection: typ

Apply to region: /

Multi-Source Delay: latest

☐ Disable warnings from SDF reader

9) View all the Model*Sim* debugging windows.

   VSIM> **view *** -or- **View > All**

10) Wave and list signals of interest in the design.

```
VSIM> wave /*          -- Adds all top level signals to the wave window
VSIM> list /*          -- Adds all top level signals to the list window
```

11) As was done in the RTL simulation, either use a VHDL testbench to stimulate the RTL design, or force the inputs directly in ModelSim.

```
VSIM> force /clk 0 @ 0 ns, 1 @ 50 ns –repeat 100 ns
VSIM> force /input 0
```

12) Run the simulation and analyze the information in the Model*Sim* debugging windows to verify the results are the same as in the RTL simulation.

   VSIM> **run –all** -or- **Run > Run Forever**