Synplify Guide for Model Technology - Model*Sim*

Section 1. Introduction

As today's designs increase in complexity, the ability to find and fix design problems through hardware decreases. Designers can't easily probe internal logic or trace back problems to the source of the problem when looking at a chip. The solution to finding and fixing these problems is through software simulation. Software simulation emulates real time operation of the internal logic as well as the external pins. By probing different locations within the chip, simulation allows designers to trace problems back to the source.

The most accurate simulation model is created by the back-end placement tool. However, for larger FPGA devices, the back-end compile times may be significantly longer than the synthesis compiles through Synplify. Synplify is uniquely capable of providing accurate functional simulation models with quick compile times thereby allowing designers to accurately debug functional design problems in a timely manner.

This application note describes how to successfully integrate simulation into your design methodology. This integration not only involves the simulation process but additionally uses test benches to run the different types of simulations.

Topics to be covered:

- Simulation Flow
- Simulation Features
- HDL Testbenches
- Example 1: Functional Simulation (Model*Sim*)
- Example 2: Compiling the Mapped Netlist (Synplify)
- Example 3: Mapped-Functional Simulation (Model*Sim*)
- Example 4: Generating Gate Level VHDL/Verilog (Xilinx Alliance)
- Example 5: Gate-Level Simulation With Timing (Model*Sim*)

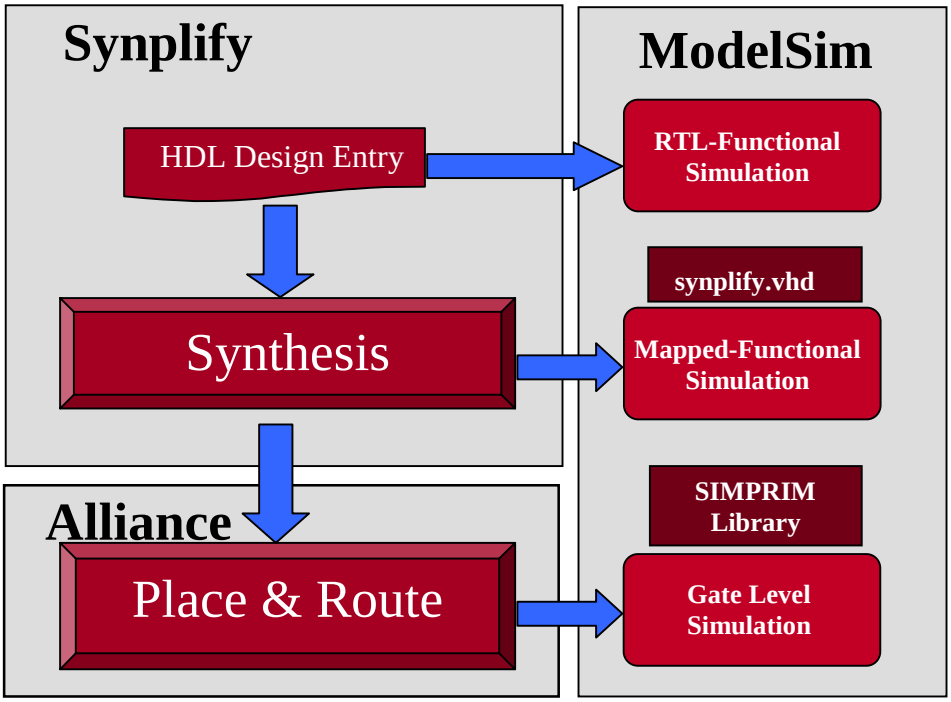

Figure 1: Design Flow

Section 2. The Design Flow

Designers have traditionally simulated the source code to check for syntax and functional problems. With the HDL editor in Synplify designers may easily enter and simulate the mapped-functional netlist. The functional simulation of the source code allows designers to check the general functionality without the full design flow.

After synthesis, Synplify generates mapped functional simulation that uses unit delays. This mapped netlist adds architecture specific structures to the functional simulation. With Synplify's linear compile times, even the largest Xilinx devices may be compiled in minutes. Designers can iterate more quickly through their design cycle using the mapped, functional simulation generated from Synplify without running into the longer Alliance place and route times.

The Alliance tools will additionally generate a VHDL or Verilog netlist with the routing delays annotated into a Standard Delay Format (SDF) file. The simulation netlist settings may be set in the Alliance tools in the IMPLEMENTATION OPTIONS window under the INTERFACE tab.

Simulation Features

As a leader in FPGA synthesis solutions, Synplify offers an efficient simulation flow for Model*Sim*. Coupled with other debugging features and fast compile times, designers can quickly debug problems within Synplify and Model*Sim*, minimizing the iterations through the Alliance tools.

Because a good simulation models true device behavior, functionally simulating the source code may not be enough. If the simulation does not accurately represent device operation, a bad simulation may cause designers to debug non-existent problems seen only during simulation or to miss critical board problems during simulation. Because VHDL and Verilog were both originally created as a simulation language, functional simulation of the source code will simulate following a set of guidelines for simulation and will not be able to model true device behavior.

Only after using Synplify for synthesis will the device characteristics be incorporated into a mapped-functional simulation model that includes device characteristics. Due to the mapping, architecture specific details will be mapped making the simulation accurately model true device behavior functionally.

Designers should simulate at the mapped-functional level to bypass longer compile times associated with placing and routing the FPGA device through the Alliance tools. Synplify's fast compile times allow designers to run multiple compiles to isolate and debug problems before having to go through the full back-end design flow. When the design is functionally correct, the designer takes the design through the Alliance tools to place and route the Xilinx device.

To make the simulation accurately model Xilinx devices, Synplify creates a mapped simulation netlist for both VHDL and Verilog. These mapped simulation netlists use unit delays to model propagation delays. Both VHDL and Verilog are structured down to the base cells, e.g. look-up-tables and carry chains.

To enhance simulations, both Synplify and Model*Sim* support VHDL '93. Concerning library requirements for simulation, only a small three-component VHDL library from Synplicity will be required (<synplcty>/lib/vhdl_sim/synplify.vhd). The remainder of the VHDL is self-contained within the simulation file. Synplify's Verilog netlist is fully self-contained and does not require any other design files to be compiled into Model*Sim*.

About Testbenches

A testbench is a separate set of VHDL or Verilog code that connects up to the inputs and outputs of a design. Since the inputs and outputs of the design stay constant throughout the Synthesis and Place & Route process, the testbench can usually be used at each stage to verify the functionality of the design. The Testbench has two main purposes:

First, a testbench provides the stimulus and response information (clocks, reset, input data, etc.) that the design will encounter when implemented in a FPGA device. Secondly, the testbench contains regression checking constructs which allow key functionality to be tested throughout the FPGA HDL simulation flow (whether a counter design is counting; whether a floating point unit has calculated the correct value; etc.).

```
use std.textio.all;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity TGUESSER is
end TGUESSER;

architecture SIMULATE of TGUESSER is
      component GUESSER
      port (
          DATA, CLK, RST                  : in  std_logic;
          GUESS                           : out std_logic);
      end component;

      constant CLK_PULSE                  : time := 50 ns;
      signal   TB_DATA,TB_RST,TB_CLK      : std_logic;
      signal   TB_GUESS                   : std_logic;

begin
      DUT: GUESSER port map( TB_DATA, TB_CLK, TB_RST, TB_GUESS );

      TB_RST  <=   '0' after 0 ns,
                   '1' after 50 ns,
                   '0' after 100 ns;

      TB_DATA <= '0' after 0 ns,
                   '1' after 150 ns,
                   '0' after 550 ns,
                   '1' after 1000 ns;

--Create the clock
      process (TB_RST, TB_CLK)
      begin
          if (TB_RST = '1' ) then
                   TB_CLK <= '0';
          else
                   TB_CLK <= not TB_CLK after CLK_PULSE;
          end if;
      end process;
end SIMULATE;

-- Configuration for simulation
-- Configuration for simulation
library WORK;
configuration CFG_TB_GUESSER of TGUESSER is
  FOR SIMULATE
    FOR DUT: GUESSER
      use entity work.GUESSER(behave);
    END FOR;
  END FOR;
end CFG_TB_GUESSER;
```

Figure 2: tstate_mch2.vhd

Section 3. The Design Example

The design file can be found in the Synplify examples directory <synplcty>\examples\vhdl\statmchs\statmch2.vhd. Create a working directory for this tutorial and copy the design file.

For Model*Sim*, both the menu commands as well as the command lines "*VSIM>*" have been shown.

Example 1: Functional Simulation (Model*Sim*)

Prior to synthesizing and place & routing with the Xilinx tools, you will first compile and simulate the "LAB" RTL design with Model Technology's Model*Sim* simulator.

1) Open Model*Sim*
2) Change directories to the working directory <working directory>:
   Select FILE=>DIRECTORY menu.
   VSIM> cd <path>
3) Create a WORK directory: Select LIBRARY=>NEW and enter in WORK
   VSIM>vlib work
4) Open up VHDL Compiler menu: Click on the VCOM button.
5) Switch to VHDL'93: Click on the OPTIONS button and check "Use 1076-1993 language standard" and ACCEPT.

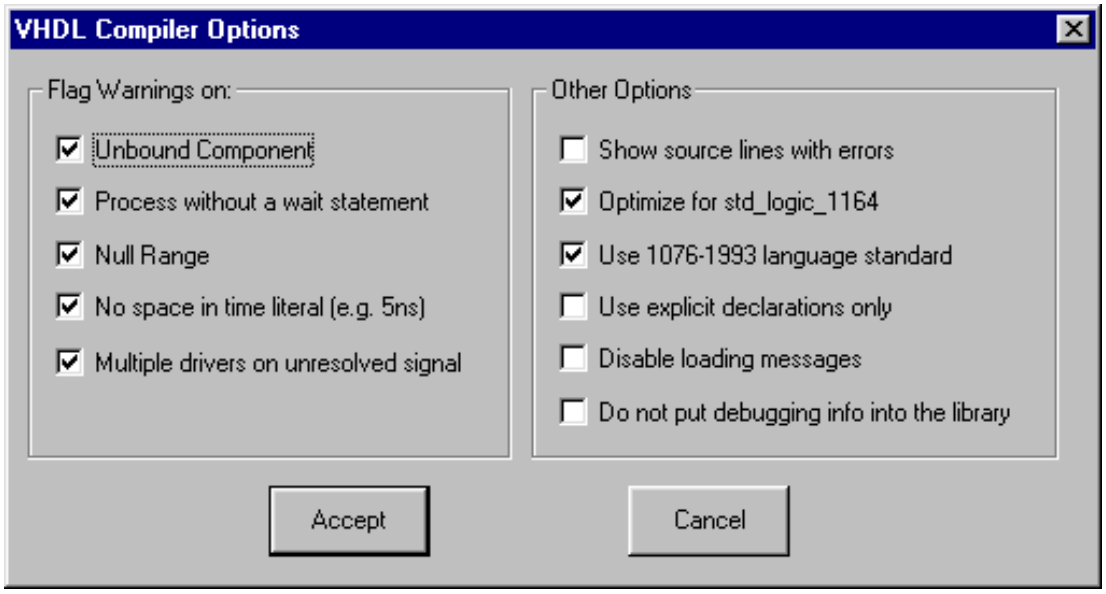

Figure 3: Setting Compiler Options

6) Compile the VHDL design file and testbench:
   a) Select STATMCH2
   b) Click COMPILE
   c) Select TSTATE_MCH2.VHD
   d) Click COMPILE
   e) Click on DONE. The design is now ready for simulation.
   VSIM> vcom –93 statmch2.vhd
   VSIM> vcom –93 tstate_mch2.vhd

7) Start simulation: Click on VSIM and select the configuration cfg_tb_guesser.
VSIM> vsim cfg_tb_guesser
8) Open the simulator windows: Select VIEW=>ALL
*VSIM> view **
9) Clean up windows by tiling: Select Window=>Tile Horizontally
10) Add signals to waveform:
Select SIGNALS=>ADD TO WAVEFORM=>SIGNALS IN DESIGN
*VSIM> wave **
VSIM> wave /dut/state
11) Run for 2000 ns
VSIM> run 2000

Note: See appendix A for a brief description of the ModelSim debugging windows

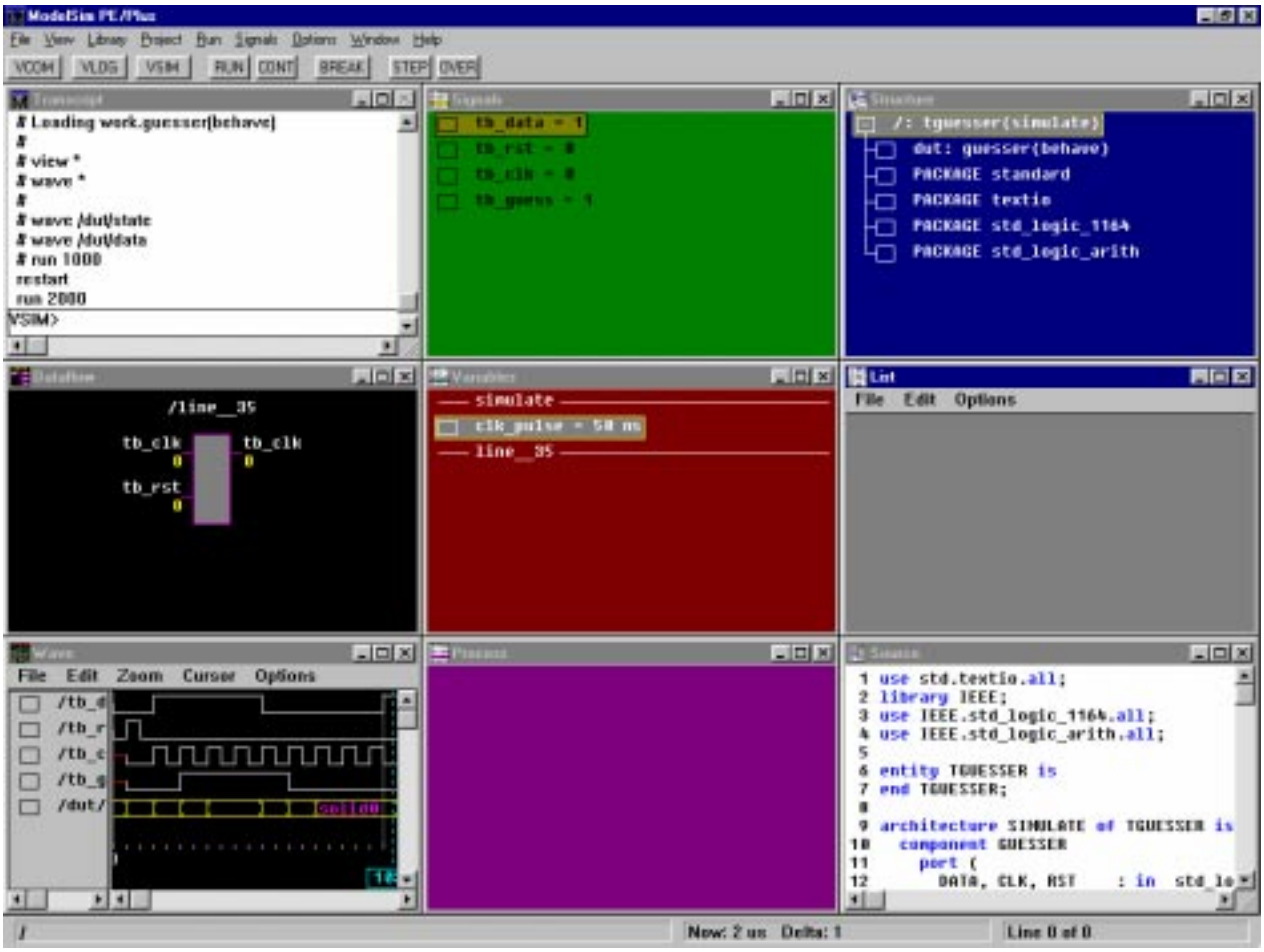

Figure 4: Model*Sim* Simulation

Example 2: Compiling the Mapped Netlist (Synplify)

1) Open Synplify
2) Create a new project file and save to working directory:
a) Select FILE=>NEW "Project"
b) Select FILE=>SAVE AS

3) Enable the VHDL or Verilog mapped netlists from the menus:
OPTIONS=>WRITE MAPPED VHDL/VERILOG NETLIST

Figure 5: Mapping To VHDL/Verilog in Synplify

Note: Synplify will create a vhdl (.vhm) or verilog (.vm) netlist in the working directory.

4) Add the design files to be compiled:
Note: Add the designs in the reverse order of hierarchy with the top level design file added last to the bottom of the list
Click on ADD (Source Files) "statmch2.vhd" to the compiler window

5) Start Compiling: Click on the Run button

Example 3: Mapped-Functional Simulation (Model*Sim*)

1) Open Model*Sim*

2) **(MAPPING LIBRARY ONLY NEEDS TO BE DONE FIRST TIME)**
Pre-Compile Synplicity Synplify library:
a) Choose a directory to keep compiled libraries: <modelsim>/SYNPLIFY
b) Change to <modelsim> directory
c) Create a SYNPLIFY library: LIBRARY=>NEW and type SYNPLIFY
d) Open VHDL Compiler window: Click VCOM button
e) Change to <synplcty>/lib/vhdl_src directory
f) Change TARGET LIBRARY to SYNPLIFY
g) Compile synplify.vhd
j) Change TARGET LIBRARY back to WORK

k) Map Compiled SIMPRIM library to directory:
Select LIBRARY=>MAPPING=>NEW
Library: synplify
Directory: <modelSim>/synplify
l) Click OK in both windows

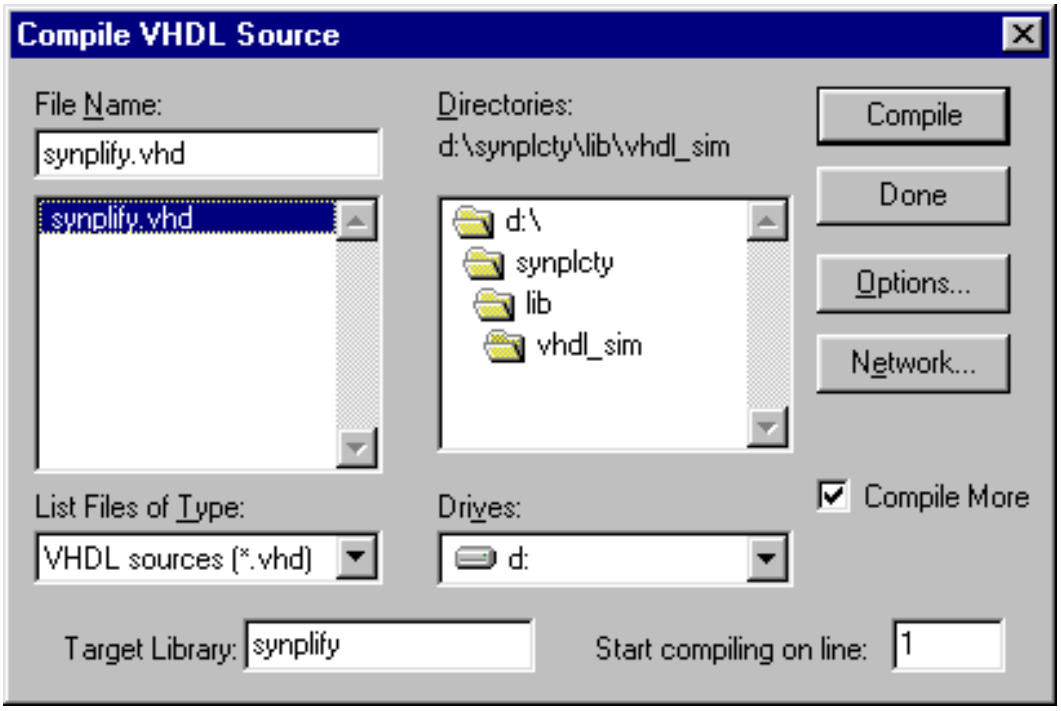

Figure 6: Compiling Synplify Library

3) Change directories to the working directory <path>:
Open the FILE=>DIRECTORY menu
VSIM> cd <path>

4) Create WORK library:
b) Select LIBRARY=>NEW and type WORK
VSIM>vlib work

5) Switch to VHDL'93 (use –93 option when using vcom in command line):
a) Click on VCOM button
b) Click on the OPTIONS button and check "Use 1076-1993 language standard" and ACCEPT.

6) Compile VHDL files:
a) Show all the files: In SELECT FILES OF TYPE select ALL FILES
b) Select statmch2.vhm
c) Click on COMPILE
d) Select TSTATE_MCH2.VHD
e) Click on COMPILE
f) Close the VHDL Compiler window by clicking on DONE. The design is now ready for simulation.
VSIM> vcom –93 statmch2.vhd
VSIM> vcom –93 tstate_mch2.vhd

7) Start Simulation: Click on VSIM and select the configuration cfg_tb_guesser.
VSIM> vsim cfg_tb_guesser

8) Simulate:
*VSIM> view **
*VSIM> wave **

VSIM> wave /dut/state
VSIM> run 2000

9) Setup display: Select Window=>Tile Horizontally

Example 4: Generating Gate Level VHDL/Verilog (Xilinx Alliance)

1) Open Alliance
2) Create a new project: Select File=>New Project
   Input Design: <path>/statmch2.xnf
   Work Directory: <path>
   Comment: <optional>
3) Enable Timing Simulation to create gate level netlist for HDL Simulation:
   a) Select DESIGN=>IMPLEMENT
   b) Click on OPTIONS
   c) Check PRODUCE TIMING SIMULATION DATA (Optional Targets)
   d) Keep DESIGN=>IMPLEMENT=>OPTIONS menu open

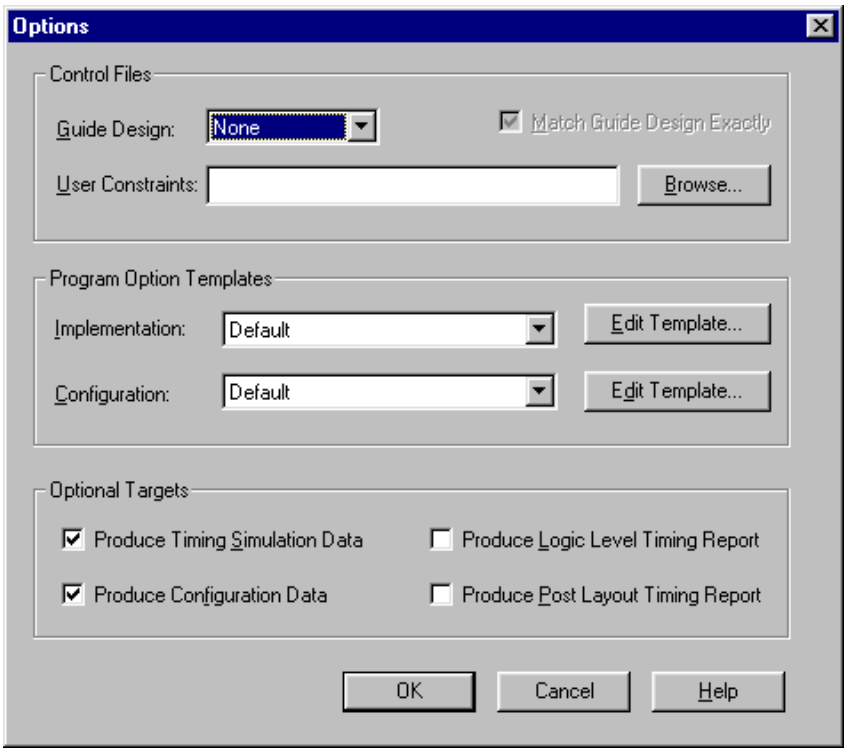

Figure 7: Setting Simulation Settings

4) Select the simulation format in DESIGN=>IMPLEMENT=>OPTIONS menu:
   a) Click on EDIT TEMPLATE (Implementation)
   b) Select INTERFACE tab
   c) Select netlist preference (VHDL/Verilog)
   d) Click OK to close EDIT TEMPLATE menu
   e) Click OK to close OPTIONS menu
   f) Keep IMPLEMENT menu open
5) Start compiling: Click on RUN
   Alliance will start compiling STATMCH2.XNF and generate the gate-level simulation netlist files automatically in the directory <path>/xproj/ver1/rev1

TIME_SIM.VHD: VHDL Netlist
TIME_SIM.SDF: Annotated Timing Delays

6) Close Flow Engine and Alliance Tools

Example 5: Gate-Level Simulation With Timing (Model*Sim*)

1) Open Model*Sim*
2) **(MAPPING LIBRARY ONLY NEEDS TO BE DONE ONCE)**
Pre-Compile Xilinx SIMPRIMS library:
a) Choose a directory to keep compiled libraries: <modelsim>/XILINX
b) Change to <modelsim> directory
c) Create a XILINX library: LIBRARY=>NEW and type XILINX
d) Open VHDL Compiler window: Click VCOM button
e) Change to <xilinx>/vhdl/src/simprims directory
f) Change TARGET LIBRARY to XILINX
g) Compile simprim_Vpackage.vhd
h) Compile simprim_VITAL.vhd
i) Compile simprim_Vcomponents
j) Change TARGET LIBRARY back to WORK
k) Map Compiled SIMPRIM library to directory:
Select LIBRARY=>MAPPING=>NEW
Library: Xilinx
Directory: <modelsim>/XILINX
l) Click OK in both windows
3) Change directories to the working directory <path>/xproj/ver1/rev1 by using the FILE=>DIRECTORY menu.
VSIM> cd <path>/xproj/ver1/rev1
4) Create the WORK directory: Select LIBRARY=>NEW and type WORK
VSIM> vlib work
5) Switch to VHDL'93 (use –93 in command line):
a) Click on VCOM
b) Click on the OPTIONS button and check "Use 1076-1993 language standard" and ACCEPT.
6) Compile VHDL File
a) Select TIME_SIM.VHD
b) Click on COMPILE
c) Select TSTATE_MCH2_GATE.VHD
d) Click on COMPILE
e) Close the VHDL Compiler window by clicking on DONE. The design is now ready for simulation.
VSIM>vcom –93 TIME_SIM.VHD
VSIM>vcom –93 TSTATE_MCH2_GATE.VHD
7) Start Simulation:
a) Click on VSIM
b) Click on Config CFG_TB_GUESSER
c) Select the SDF tab
SDF File: TIME_SIM.SDF
Apply to region: DUT
VSIM> vsim –sdftype DUT=time_sim.sdf cfg_tb_guesser
8) Simulate:
*VSIM> view **
*VSIM> wave **
VSIM> run 2000
9) Setup display: Windows=>Tile Horizontally

```
use std.textio.all;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity TGUESSER is
end TGUESSER;

architecture SIMULATE of TGUESSER is
 component STATMCH2
  port (
   DATA, CLK, RST    : in  std_logic;          -- Clock
   GUESS             : out std_logic);  -- STATMCH2 result
 end component;

 constant CLK_PULSE              : time := 50 ns;
 signal   TB_DATA,TB_RST,TB_CLK    : std_logic;
 signal   TB_GUESS               : std_logic;

begin
 DUT: STATMCH2 port map( TB_DATA, TB_CLK, TB_RST, TB_GUESS );

 TB_RST  <= '0' after 0 ns,
          '1' after 50 ns,
          '0' after 100 ns;
 TB_DATA <= '0' after 0 ns,
          '1' after 150 ns,
          '0' after 550 ns,
          '1' after 1000 ns;

--Create the clock
 process (TB_RST, TB_CLK)
 begin
  if (TB_RST = '1' ) then
     TB_CLK <= '0';
  else
     TB_CLK <= not TB_CLK after CLK_PULSE;
  end if;
 end process;

end SIMULATE;

-- Configuration for simulation
library WORK;
configuration CFG_TB_GUESSER of TGUESSER is
 FOR SIMULATE
    FOR DUT : STATMCH2
       use entity work.STATMCH2(STRUCTURE);
    END FOR;
 END FOR;
END CFG_TB_GUESSER;
```

Figure 8: tstate_mch2_gate.vhd

Appendix A

The following is a brief description of the Model*Sim* debugging windows.

Transcript – The command-line window; displays a transcript of all command activity.

Source – Displays the HDL source code for the design.

Structure – Displays the hierarchy of structural elements such as VHDL component instances, packages, blocks, generate statements, and Verilog model instances, names blocks, tasks and functions.

Wave – Displays waveforms and current values for the VHDL signals, and Verilog nets and register variables you have selected.

List – Shows the simulation values of selected VHDL signals, and Verilog nets and register variables in tabular format.

Dataflow – Allows you to trace VHDL signals or Verilog nets through your design.

Signals – Shows the names and current values of VHDL signals, and Verilog nets and register variables in the region currently selected in the Structure window.

Variables – Displays VHDL constants, generics, variables, and Verilog register variables in the current process and their current values.

Process – Displays a list of processes that are scheduled to run during the current simulation cycle.