ModelSim/Quartus Simulation Tutorial

By Dr. P. Chu, Dept EECS, Cleveland State University

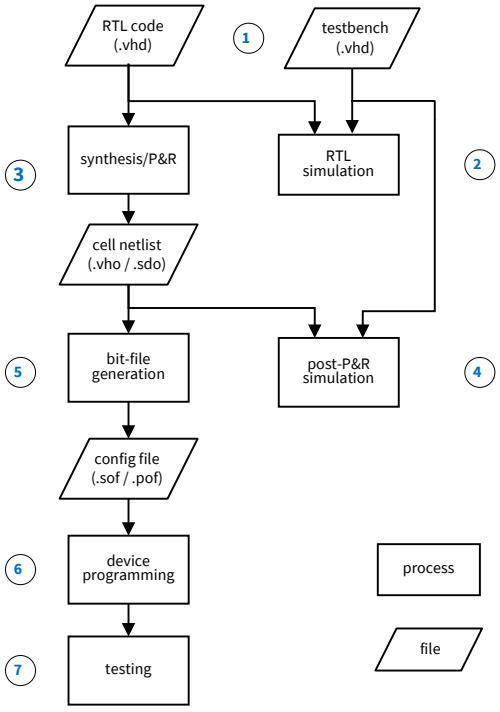
1 INTRODUCTION

ModelSim is a mixed-language HDL simulator manufactured by Mentor Graphics. It is the most widely used HDL simulation program. "*ModelSim – Intel FPGA Starter Edition*" is included in Quartus suite distribution and can be downloaded from Intel PSG site. It is a free "watered-down" version configured with Intel FPGA device model libraries. The tutorial uses this version. The tutorial includes covers two types of simulation:

- RT-level simulation
- Post synthesis (cell-level) simulation

2 OVERVIEW OF DEVELOPMENT FLOW

The simplified FPGA development flow is



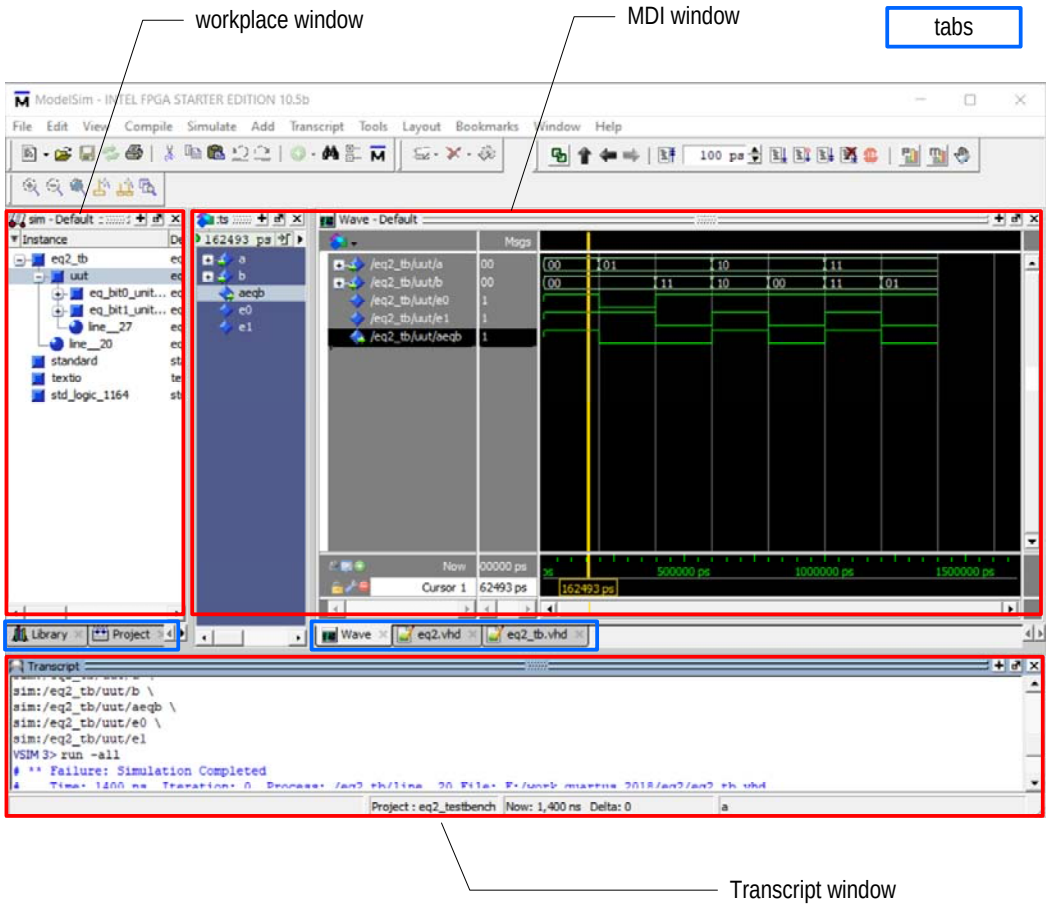
The basic steps are

1. Develop VHDL codes for RT-level design and testbench (both .vhd file).
2. Perform RT-level simulation.
3. Perform synthesis and placement-and-routing to software generate a cell-level netlist file (.vho) and timing delay file (.sdo).
4. Perform post-placement-and-routing simulation.
5. Generate FPGA configuration file (.sof or .pof).
6. Download the file (i.e., "program") the FPGA device.
7. Test the physical circuit.

The simulation is on the right track, including Steps 1 and 2 for RT-level simulation and Steps 1, 3, and 4 for the post-synthesis simulation. This tutorial focuses on this track. Note that the simulation is performed twice, one at the RT-level (original HDL code) and one at the cell level (synthesized physical implementation).

3 OVERVIEW OF MODELSIM LAYOUT

ModelSim is a standalone HDL simulator and can run independently without Quartus. The default ModelSim window is shown below



It is divided into three subwindows:

- Transcript window (bottom)
- Workspace window
- Multiple-document-interface (MDI) window

The Workspace window displays information on the current process. The bottom tab is used to select the desired process page, which can be Project, Library}, Sim, and so on. The Transcript window keeps track of command history and messages. It can also be used as a command-line interface to enter ModelSim commands. The MDI window is an area to display HDL text, waveform, and so on. Its bottom tab selects the desired pages.

Each subwindow may be resized, moved, docked, or undocked. Additional windows may appear for some operations. The default layout can be restored by selecting Window ⇒ Initial Layout.

4 RT-LEVEL SIMULATION

The RT-level simulation process includes three basic steps:

1. Create a simulation project.
2. Compile the HDL codes.
3. Perform simulation and examine the waveform.

The detailed procedure is demonstrated in the following subsections.

We use the 2-bit comparator and its testbench discussed in Chapter 1 for the tutorial. The files are

- eq1.vhd: 1-bit comparator
- eq2.vhd: 2-bit comparator composed of two 1-bit comparators
- eq2_tb.vhd: testbench for the 2-bit comparator

The listing of these files is in the Appendix.

4.1 FILE STRUCTURE

Both synthesis and simulation generate a lot of intermediate and auxiliary files. It is a good practice to use separate folders for them. The recommended file structure for a simple project (say project #1) is

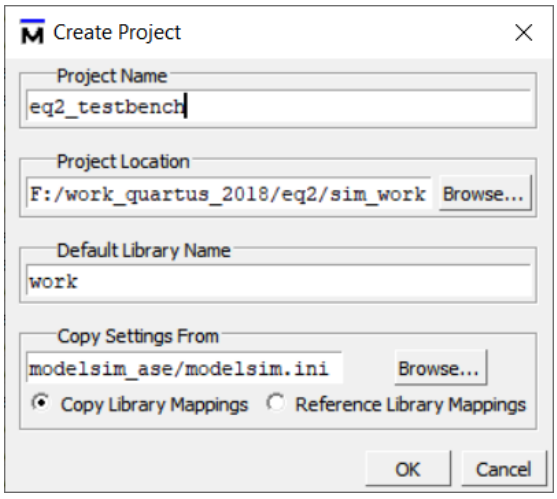
- ../proj1/src: folder for the HDL source files, constraint file, and test bench
- ../proj1/syn_work: working directory for synthesis
- ../proj1/sim_work: working directory for simulation

Only the files in the src folder need to be saved or backed up.

4.2 CREATE A SIMULATION PROJECT

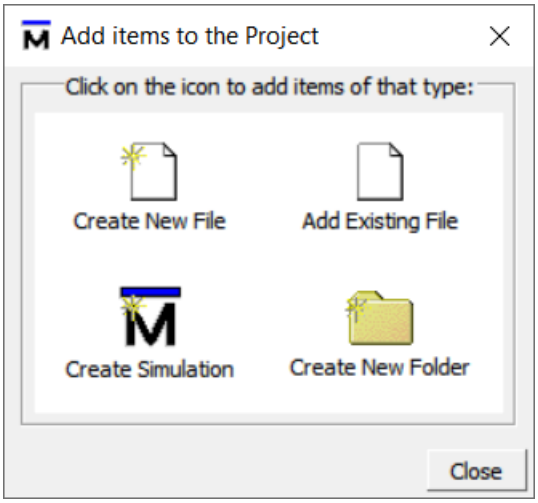
A ModelSim simulation project consists of the library definition and a collection of HDL files. A testbench is an HDL program and can be created by using the Quartus text editor or any VHDL editor. We assume that all HDL files are already constructed. The procedure to create a project is as follows:

1. Launch ModelSim from the Quartus installation folder (the default folder is Intel FPGA …).
2. Select File ⇒ New ⇒ Project and the Create Project dialog appears:

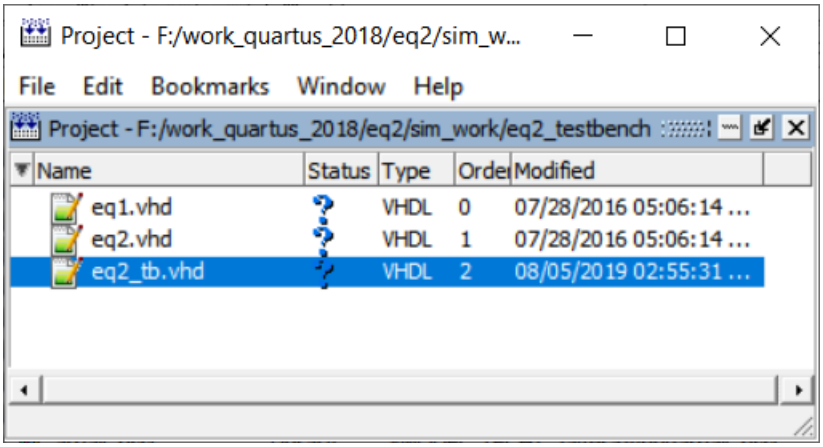


- Enter the project name, say eq2_testbench
- Select the project location for a simulation work folder (something like .../sim_work)
- Keep Default Library Name to work.
- Click OK.

3. The Project page appears in the main window and the Add items to the project dialog appears:



4. In the Add items to the project dialog
   - Click Add Existing File.
   - Add eq1.vhd, eq2.vhd, and eq2_tb.vhd.
   - Click Close. The project tab appears in the workplace subwindow and displays the selected files:



4.3 COMPILE THE HDL CODE

The *compile* term here means to convert the HDL code into ModelSim's internal format. In VHDL, the compiling is done on the *design unit* basis. Each entity and architecture is considered as one design unit. The procedure is:

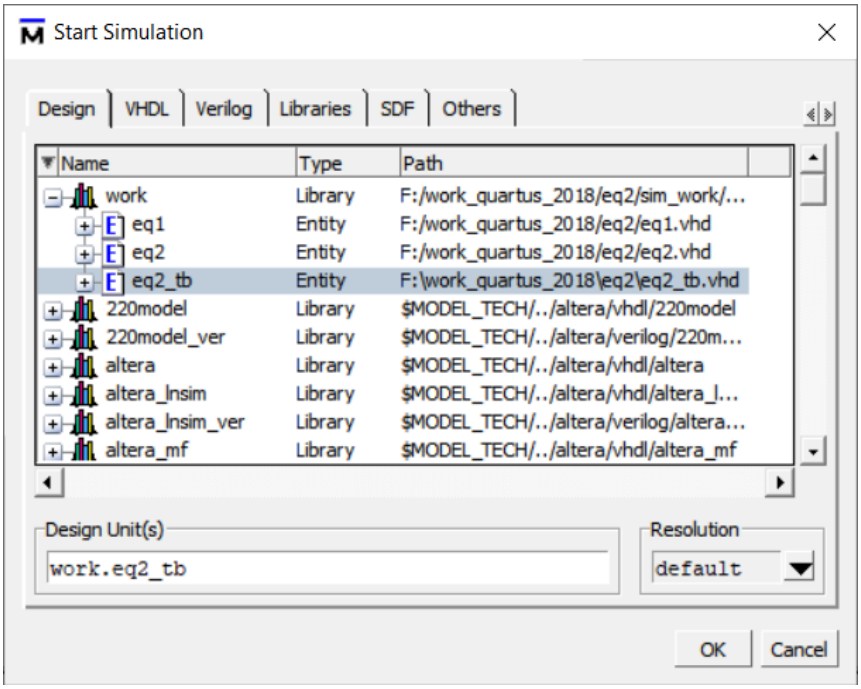
1. Highlight the eq1 file and right-click the mouse.

2. Select Compile ⇒ Compile Selected. Note that the compiling should be started from the modules at the bottom of the design hierarchy. The progress and messages are displayed in the transcript window.
3. If the file contains no syntactical error, a green checkmark ✓ shows up.
4. If an X mark shows up, click the red error line in the transcript window to locate the errors. Correct the problems, save the file, and recompile the file.
5. Repeat the preceding steps to compile the eq2 file and then the eq_tb file.

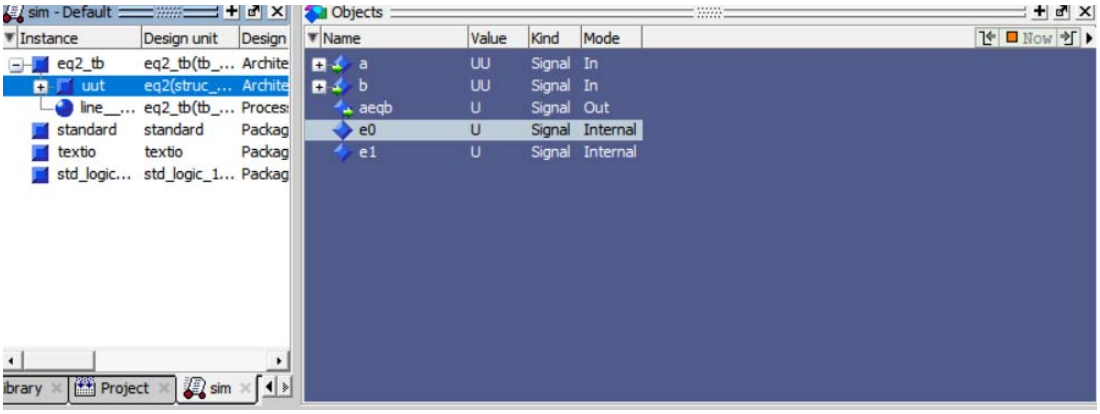
4.4 PERFORM A SIMULATION AND EXAMINE THE WAVEFORM

After compiling the testbench and corresponding files, we can perform the simulation and examine the resulting waveform. This corresponds to running the circuit in a virtual lab bench and checking the waveform in a virtual logic analyzer. The procedure is:

1. Select Simulate ⇒ Start Simulate and the Simulate dialog appears.
2. In the Design tab, find and expand the work library, which is the one defined when we create the project. All the compiled units are displayed:

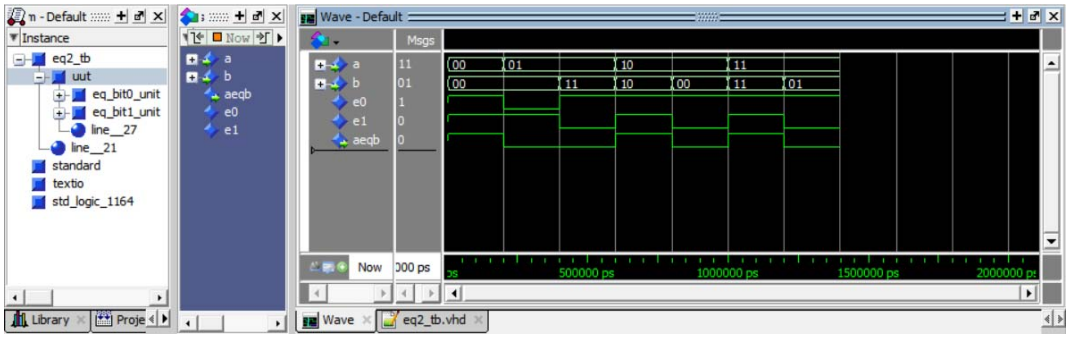


3. Select eq2_testbench and click OK.
4. If the Objects panel does not show up, select View ⇒ Objects in the menu.
5. The sim tab appears in the workplace window and the corresponding page displays the structure of the eq2_testbench module:



6. Highlight the uut unit from the previous window. The ports and signals from the uut unit show up on the Objects panel.
7. Select the desired ports/signals and right-click the mouse. Select Add ⇒ Add to Wave. This adds signals/ports it to the waveform page. The waveform page appears in the MDI window.
8. If necessary, rearrange the order of the signals and set them to the proper formats (decimal, hex, and so on).
9. Select Simulate ⇒ Run in the menu. There are several commands to control the simulation:
   - Restart: restart the simulation
   - Run: run the simulation one step
   - Continue run: resume the run from the interrupt
   - Run All: run the simulation forever
   - Break: (break the simulation).

   These commands are also shown as icons at the top of the window.
10. The waveform window displays the simulated result:



We can scroll the window, zoom in, or zoom out to check the correctness of the design.

4.5 FURTHER READING

ModelSIm is a complex software package. The tutorial just helps you "jump-start" the leaning process. Mentor Graphics provides a more detailed tutorial and user guide:

- *ModelSim Tutorial*
- *ModelSim User Manual*

The tutorial and manual and other relevant documents can be accessed in ModesSim by selecting Help ⇒ PDF documentation in the menu.

5 POST-SYNTHESIS SIMULATION

Post-synthesis simulation verifies the correctness of the synthesized circuit. Since only a small subset of VHDL constructs can be realized in hardware, not all VHDL code cannot be correctly constructed. The synthesized circuit may not function correctly even the RT-level simulation is fine. Post-synthesis simulation can be used to check for the error and detect the discrepancy.

Post-synthesis simulation includes the following steps:

1. Create a design project with HDL codes.
2. Configure the simulation setting.
3. Compile the project.
4. Create a simulation project.
5. Compile the HDL codes.
6. Perform simulation and examine the waveform.

The detailed procedure is demonstrated in the following subsections.

We use the 2-bit comparator and its testbench for the tutorial. The files used for synthesis are

- eq1.vhd: 1-bit comparator
- eq2.vhd: 2-bit comparator composed of two 1-bit comparators

After synthesis, Quartus generates the eq2.vho file, which is the cell-level netlist. The eq2.vho is a VHDL file and the .vho suffix stands for "VHDL output." The files used for post-synthesis are

- eq2.vho: cell-level netlist of 2-bit comparator
- eq2_tb.vhd: testbench for the 2-bit comparator

5.1 CREATE A DESIGN PROJECT WITH HDL CODES.

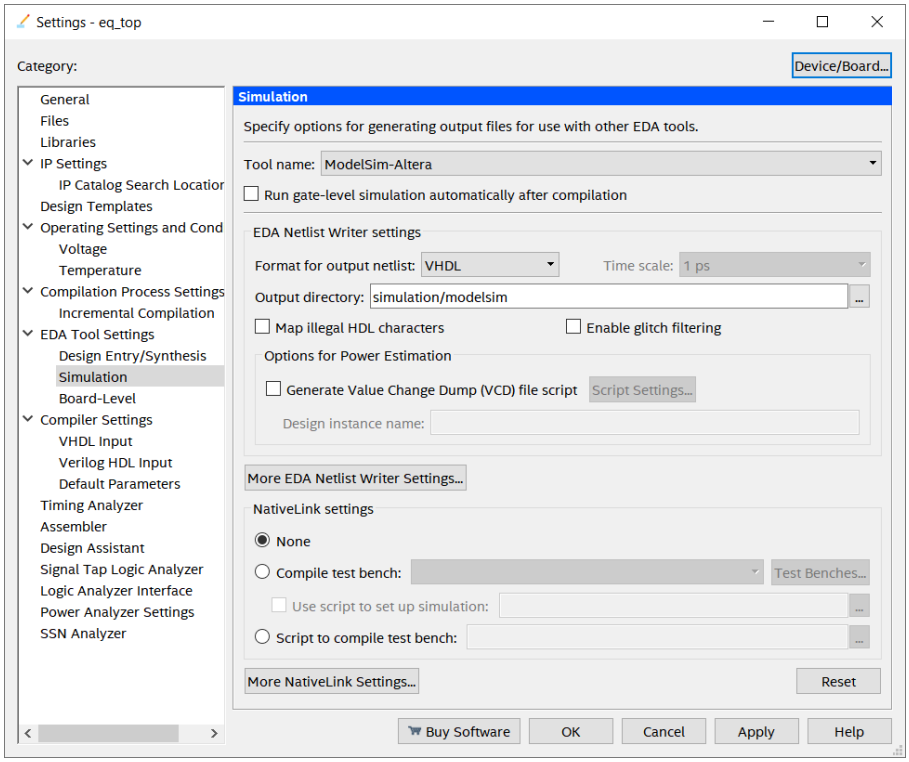
This step is the same as the one discussed in "*Quartus Synthesis Tutorial"* except for the following:

- The top-level wrapping file (eq2_top.vhd) is not needed and the original 2-bit comparator (the eq2 entity of eq2.vhd) will be set up as the top-level entity.
- The "Import a pin-assignment constraint file" task can be omitted.

5.2 CONFIGURE THE SIMULATION SETTING

In order to obtain a cell-level netlist file (.vho) to facilitate ModelSim, we need to configure the synthesis software. The procedure is:

- In the Quartus II menu, select Assignments ⇒ Settings…
- The Settings dialog appears:



- On the left panel, expand EDA Tool Settings and the select Simulation.
- In the Tool name: field, use the pull-down menu and select ModelSim-Altera.
- In the Format for output netlist: field, use the pull-down menu and select VHDL.
- Click the OK button to save the setting.

5.3 COMPILE THE PROJECT

There are three tasks in this step:

1. Specify the top-level module.
2. Perform synthesis and placement-and-routing.
3. Verify the .vho file

5.3.1 Specify the top-level module

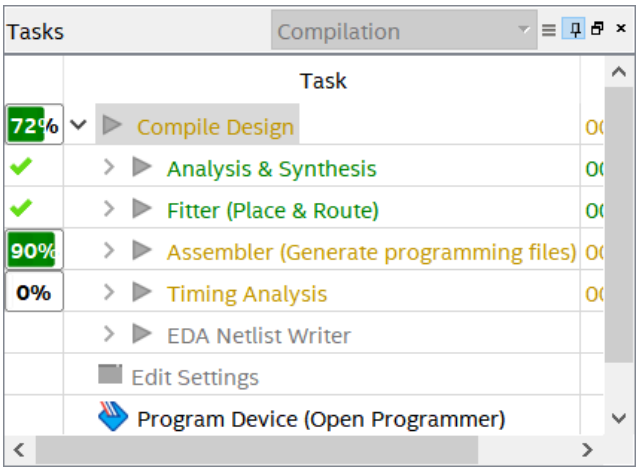
In Quartus, the default top-level module name is set to be the top-level HDL file name. After adding and creating all HDL files, we can change the top-level module if needed. The procedure is:

- In Quartrus menu, select Assignments ⇒ Settings…. The Settings dialog appears.
- In the left panel, click the General item.
- Enter the top-level entity name, eq2, in the Top-level entity field.
- Click the OK button to complete the process.

The top-level wrapping file (eq2_top.vhd) is not used because it is a wrapping file used to accommodate the DE10-lite peripherals and the signal names are not very meaningful for simulation).

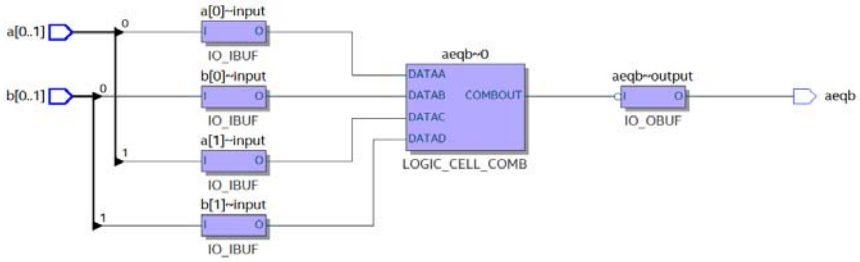
5.3.2 Perform synthesis and placement-and-routing

It can be invoked by selecting Processing ⇒ Start Compilation in Quartus menu. the eq2.vho (where eq2 is the Quartus project name) file is generated when the EDA Netlister Writer task is completed.



5.3.3 Verify the .vho file

After compiling, a cell-level netlist (i.e., implementation with FPGA's logic cells) is generated. The scheme diagram looks like



The eq2.vho file is the VHDL description of the schematic. It is placed in the .../syn_work/simulation/modelsim/ folder. Open the file with the Quartus editor and verify the architecture name. Following are some code segments:

```
architecture structure OF eq2 is
...
begin
   ...
   -- a logic cell
   \aeqb~0\ : fiftyfivenm_lcell_comb
   generic map (
      lut_mask => "0110111111110110",
      sum_lutc_input => "datac")
   port map (
      dataa => \a[0]~input_o\,
      datab => \b[0]~input_o\,
      datac => \a[1]~input_o\,
      datad => \b[1]~input_o\,
      combout => \aeqb~0_combout\);
   ...
end structure;
```

Note that the entity declaration remains the same (i.e., eq2) but a new architecture body, structure, is created.

5.4 CREATE A SIMULATION PROJECT

The step is the same as the RT-level simulation but just add the eq2.vho and eq2_tb.vhd. In the eq2_tb.vhd file, following statement is used to define the desired architecture:

```
uut : entity work.eq2(struc_arch)
```

The struc_arch is the architecture name of the original RT-level design (in eq2.vhd). It needs to be replaced with structure for the cell-level code:

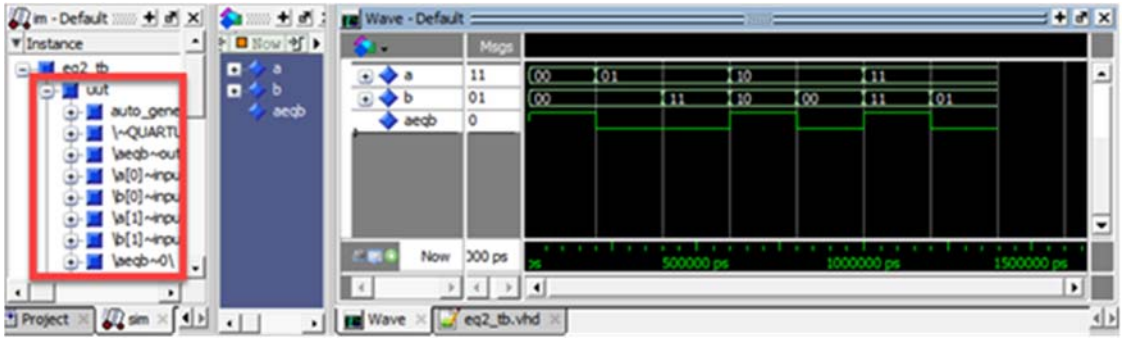
```
uut : entity work.eq2(structure)
```

5.5 COMPILE THE HDL CODES

The step is the same as the RT-level simulation.

5.6 PERFORM FUNCTIONAL SIMULATION AND EXAMINE THE WAVEFORM.

The step is the same as the RT-level simulation. If we expand uut on the left panel (highlighted by the red square), it shows that uut is composed of logic elements. However, since the synthesized circuit is correct, the waveform is the same.



6 REFERENCE

- *FPGA Prototyping by VHDL Examples 2nd edition: Xilinx MicroBlaze MCS SoC*
- *FPGA Prototyping by VHDL Examples: Xilinx Spartan-3*

APPENDIX: CODE LISTINGS

Following are the three files used in this tutorial.

```
-- File eq1.vhd
library ieee;
use ieee.std_logic_1164.all;
entity eq1 is
   port(
      i0, i1 : in  std_logic;
      eq     : out std_logic   );
end eq1;
```

```
architecture sop_arch of eq1 is
   signal p0, p1 : std_logic;
begin
   -- sum of two product terms
   eq <= p0 or p1;
   -- product terms
   p0 <= (not i0) and (not i1);
   p1 <= i0 and i1;
end sop_arch;

-- File eq2.vhd
library ieee;
use ieee.std_logic_1164.all;
entity eq2 is
   port(
      a, b : in  std_logic_vector(1 downto 0);
      aeqb : out std_logic );
end eq2;

architecture struc_arch of eq2 is
   signal e0, e1 : std_logic;
begin
   -- instantiate two 1-bit comparators
   eq_bit0_unit : entity work.eq1(sop_arch)
      port map(
         i0 => a(0),
         i1 => b(0),
         eq => e0
      );
   eq_bit1_unit : entity work.eq1(sop_arch)
      port map(
         i0 => a(1),
         i1 => b(1),
         eq => e1
      );
   -- a and b are equal if individual bits are equal
   aeqb <= e0 and e1;
end struc_arch;

-- File eq2_tb.vhd
library ieee;
use ieee.std_logic_1164.all;
entity eq2_tb is
end eq2_tb;

architecture tb_arch of eq2_tb is
   signal a : std_logic_vector(1 downto 0);
   signal b : std_logic_vector(1 downto 0);
   signal aeqb : std_logic;
```

```
begin
   -- instantiate the circuit under test
   uut: entity work.eq2(struc_arch)
   --uut: entity work.eq2(structure)
      port map(
         a    => a,
         b    => b,
         aeqb => aeqb
      );
   -- test vector generator
   process
   begin
      -- test vector 1
      a <= "00";
      b <= "00";
      wait for 200 ns;
      -- test vector 2
      a <= "01";
      b <= "00";
      wait for 200 ns;
      -- test vector 3
      a <= "01";
      b <= "11";
      wait for 200 ns;
      -- test vector 4
      a <= "10";
      b <= "10";
      wait for 200 ns;
      -- test vector 5
      a <= "10";
      b <= "00";
      wait for 200 ns;
      -- test vector 6
      a <= "11";
      b <= "11";
      wait for 200 ns;
      -- test vector 7
      a <= "11";
      b <= "01";
      wait for 200 ns;
      -- terminate simulation
      assert false
         report "Simulation Completed"
         severity failure;
   end process;
end tb_arch;
```