Wiki »

FOUR BIT ADDER TEST BENCH

Exercise Goals

Through the creation of a new ModelSim project including a unit to test and its test bench, you will get knowledge about:

- The creation of combinatorial test benches in the ModelSim IDE
- The use of assertions and wait functions and other test bench constructs.
- ModelSim
- Intent model generation for describing combinatorial behaviour
- TextIO for importing test vectors from external data sources

Prerequisites

- Quartus and ModelSim IntelFPGA software must be installed and working.
- Have the Modelsim tutorial and handbook near by for assistance (See ModelSim Help Menu)
- Have completed the four bit adder exercise

Exercise Steps

1 Create a new ModelSim project

Open ModelSim and see follow the ModelSim Introduction to create an initial project. Name the project: "fourbitadder". Do NOT start adding files or running simulations yet!

2 Add four bit adder and create the Test Bench

A) With a new bare ModelSim project created it is time to add new VHDL files. In the project window, right-click and select "add to project"->"existing file". Add the .vhd files from the four bit adder project

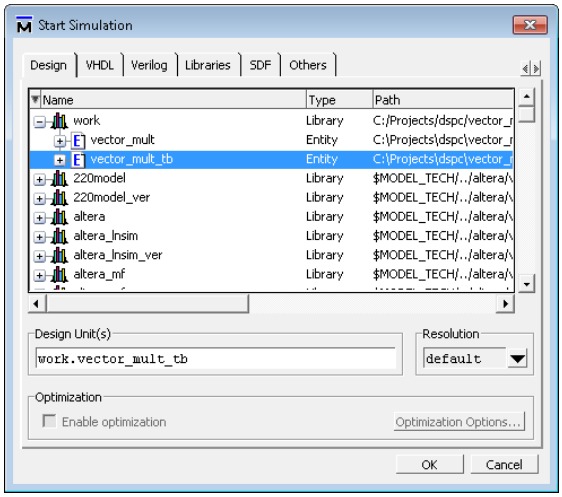
C) Create a new test bench file and add it to the modelsim project by right-clicking and selecting "add to project"->"new file" in the project window. Name the file four_bit_adder_tb.vhd. Knowing the entity decleration of the unit under test (UUT), we are able to write the test bench skeleton code. We know the signals involved, the component that must be instantiated and the test bench's entity decleration.

D) Write the test bench skeleton code. You may use Notepad++ and the VHDL Plugin to copy the entity interface and create a template test bench in the test bench file. The basic test bench will not do any actions on the UUT, but we'll come back to that later.

F) Compile the file in Modelsim. Correct any errors.

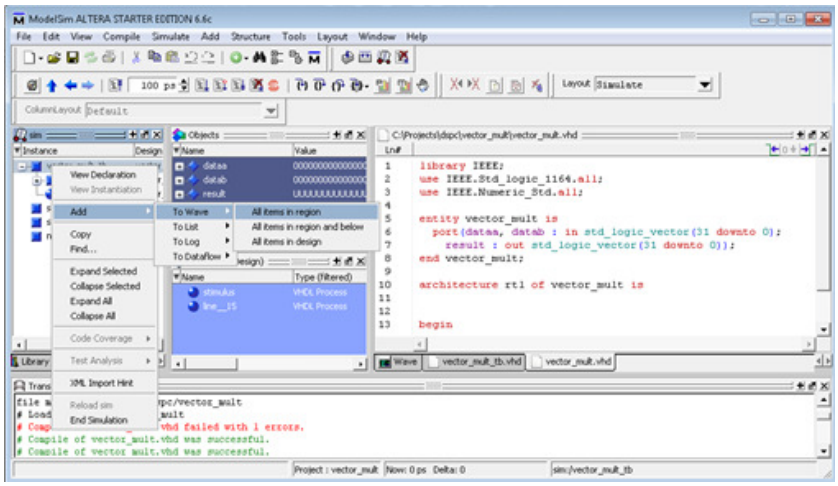
G) Now we would to add stimuli to the test bench. In the testbench process (with NO sensitivity list), set the inputs "A" and "B" to a given value and use "wait for xx ns" to evaluate the outputs. Add several value assignments and wait statements. Compile and debug until compilation is successful.

H) In ModelSim, select "Simulate", this will bring up the following window:



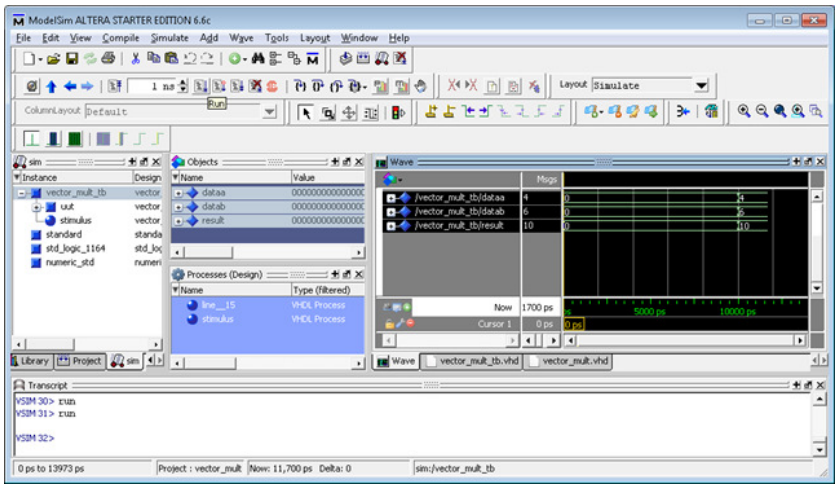
You must select the [E]ntity you wish to simulate. You must pick the _tb file as your top-level entity. Leave Resolution to default and press OK.

I) The simulation view will bring up a nuber of windows. You'll need to select which signals to show in the waveform viewer:



Note the structure of the "Instance" window, it corresponds to your structural design. Select all items in the top-level region.

J) Finally, the waveform viewer is brought to view:



Set a step length that makes sense when thinking back on your "wait for xx ns" statements in the test bench. Press the "Run" button to forward one step. If you modify the testbench or the UUT, you will have to recompile them and press the "restart" button in the waveform window.

You should be able to see a waveform changing its values when running a sufficient number of steps. If you have errors and need to know details of the interiour of the UUT, then select the uut in the Instance window and add the internal signals of the UUT to the waveform viewer.

3 Intent Model Generation

A) Use an intent model to generate stimuli data and output values. Check the relevant chapters in "VHDL for Engineers", there is plenty stuff about the topic.

B) Add assertions to the testbench, check if the four bit adder performs as expected. Try adding a wrong assertion or an assertion straight after a signal assignment, simulate and note how you are notified in ModelSim when simulating. When doing intense simulation, the waveform windows becomes too overwhelming and only through assertions will you have a certain guarantee that your design still complies with the rules set up.

4 File Access

Working with real-world (or Matlab) you will have to import this data into your simulation. VHDL itself provides a set of "textIO" functions for doing so. These are however very limited. As with a lot of programming environments, people create their own tool to get the job done, when they are not satisfied with what's available. In this exercise step you are allowed to do one of two possible solutions:

- Use VHDL's textIO to read a text file with data
- Write your own Python script (or use another suitable language) to generate a VHDL file with a "ROM" (an array of std_logic_vectors) with data

TextIO

See the “ModelSim User’s Manual” page UM-71 for details on textIO. You can find inspiration here. You can also check Test Bench for the Edge Detector in the repository. See the _tb.vhd and io_utils.vhd in particular.

Note! When you add “use std.textio.all” to your testbench and compile your design in ModelSim, the textIO library will appear in the simulation tab of the worksheet window. Clicking on this opens the package header file, and you’ll be able to see the function prototypes.

TextIO uses two functions for reading a file:

```
readline(file, line); -- Reads a line
read(line, bit/bit_vector/integer/etc.); -- Reads one element in
                                         -- the line and moves the pointer
                                         -- to the next element
```

TextIO does not support std_logic, but does support “bit”. Conversion is done with:

```
<std_logic_vector> <= to_stdlogicvector(<bit_vector>);
<std_logic> <= to_stdUlogic(<bit>);
```