ModelSim Guide

Introduction

This document is a review and reference guide for using ModelSim. The Intel ModelSim Starter Edition is free and comes bundled with the (also free) Lite version of the Quartus Prime software from Intel. It is an important tool for testing, validating, and debugging your hardware.

Compiling Your Design

Before you can simulate, you must compile your source files (Verilog (.v) and SystemVerilog(.sv)) using the `vlog` command. This is a wholly separate, and unrelated, process to compiling in Quartus Prime. By default, files will be searched for in the current directory, which can be changed or viewed with the `cd` and `pwd` commands, respectively.

Compiled versions of (System)Verilog modules are placed into a *library*, with the default one being named "`work`" by convention, and will be created in a subfolder of the same name. You must also create this library before you can compile:

```
cd ../path/to/source/files
vlib work      # creates 'work' library
vlog file1.v   # compiles source files into 'work' library
vlog file2.sv
vlog *.sv      # wildcards are okay
```

For convenience, you can put all the `vlib` and `vlog` commands into a separate script file, and run it from ModelSim with '`do` *`scriptfile`*'.

Starting the Simulator

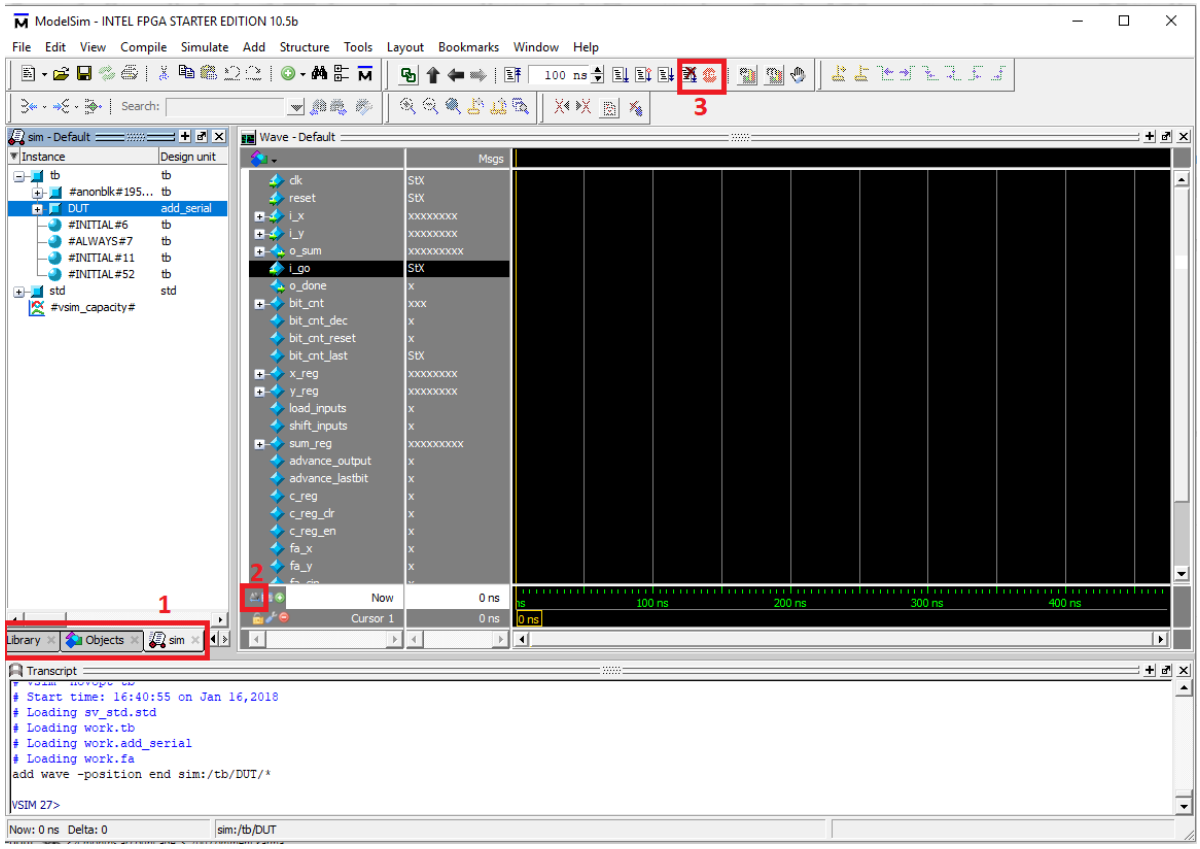
After compiling, use the `vsim` command to enter simulation mode. The `-novopt` flag will disable optimization, allowing you to peek at internal signals in your hardware, rather than just the input/output pins. This is similar to how when writing C/C++ code, you pass the `-g` flag to `gcc` or `g++` to disable optimization and allow debugging.

The main argument to `vsim` is the name of the module you wish to simulate. This will automatically include the sub-modules that it instantiates as well, as they are part of its design. An ideal candidate for simulation is a specially-designed Verilog module, called a *testbench*, that instantiates the actual hardware you wish to test, and contains special code to exercise/stimulate it with test cases. It is generally not good design practice to simulate your FPGA top-level module (the one that's at the top of the hierarchy in your Quartus Prime project), as it contains generic, non-project-specific signal names like `SW` and `KEY`, some of which might be active-low.

```
# Starts simulation of the previously-compiled module called 'tb'.
# Additional errors in your code may be caught here, rather than during vlog
vsim -novopt tb
```

Simulation Mode GUI Layout

Shown below is a recommended layout of GUI elements when in simulation mode (after invoking a successful `vsim` command). The windows can be dragged and combined. The Wave window might not be initially visible the first time you run ModelSim, and can be shown from the `View` menu.



1. The Libary, Objects, and sim tabs window are grouped as tabs on the left-hand side. The 'sim' tab lets you browse and select a module instance from the design hierarchy, and the Objects tab then shows the signals within that instance.

2. This button toggles Full vs. Leaf names of the signals in the wave window. Shorter names are easier to read (enabled in the screenshot).

3. These buttons will let you break or stop a running simulation if it goes on too long.

Adding Signals

There are two ways to add signals to the Wave window: with commands, or by dragging-and-dropping (an entire module instance from the 'sim' tab, or individual signals from the Objects tab). You can right-click the grey area with the signal names to add dividers or Groups to better organize your signals. This is also possible with commands. When using commands, you specify the full path to a signal or module hierarchically, using '/' as a separator.

```
add wave /tb/DUT/clk  # adds one signal
add wave /tb/DUT/*    # adds all the signals in the 'DUT' instance
add wave -divider "Control Signals"  # create a named divider
add wave -group FSM /tb/DUT/state /tb/DUT/nextstate  # create group with 2 signals
```

Running the Simulation

After adding the signals you wish to monitor, use the `run` command to begin, or continue, the simulation:

```
run 5us   # run for a specific amount of time
run -all  # run until the testbench code explicitly stops the simulation
```

The most common thing you will be doing is fixing/modifying your source code and then re-simulating. This is possible to do **without** having to exit and re-enter simulation mode:

```
<modify and save some Verilog files>
vlog *.sv      # recompile one or more files...
do compile.do  # ... or run your compile script again
restart -f     # restarts simulator at time=0, keeping the same Wave signals
```

Finally, to exit simulator mode:

```
quit -sim
```

GUI Hotkeys

F	Zooms to fit entire simulation into the Wave window
C	Centers and zooms Wave window on active yellow cursor
+/-	Zoom in and out on position pointed at by mouse cursor
Tab	Advances yellow cursor to next transition of selected signal
Shift+Tab	... previous transition ...

Another useful feature is changing the number base of signals displayed in the Wave window. Select one or more signal names, right click, and select the *Radix* menu. *Unsigned* interprets the signal as unsigned decimal, and *Decimal* as signed decimal.