An Introduction to VHDL

Dhruv Ilesh Shah

January 12-13, 2017

VHDL → VHSIC *(Very High Speed Integrated Chip)* Hardware Description Language
A great resource/guide would be The GHDL Homepage

Important Features

- Ports and Signals are pretty much the same, as far as the usage in a *driver* is concerned. Simply carry a literal of the data-type mentioned.
- An input port cannot be assigned a value by a concurrent statement. Similarly, an output port cannot serve as an input to another driver. To do so, another port type called *buffer* must be used.
- A driver event is triggered only when one of the dependencies undergo a change in value, unlike sequential execution of a program.
- Each uninitialised signal has a default value corresponding to the the left-most value in description. (`type bit: {'0', '1'}`).
- For explicit definition, you can declare it like `signal w: bit := '0'`.
- VHDL treats each continuous sequence of values of the ports/signals as a waveform. The transactions are piled up in a *queue*. The algorithm is as follows.
  - Once you execute (compile?) the events at $t = 0$, you create a bunch of transactions that need to be performed. These are pushed into a T-queue.
  - The pointer moves to the head of the queue and implements the transactions at that point in time. The resultant driver changes make further additions to the T-queue, and the loop goes on forever.
  - Delay δ can be explicitly mentioned. `a <= not a after 5ns;`
  - The approach is kind of a breadth-first approach. All the transactions/events at the current point in time executed before moving at a point in time.
- The standard way of simulation works only on a fixed value of inputs as defined. What we'd want is to create events on the inputs, to capture the complete picture. For this, we create a *testbench*.
- Testbench is self-contained, with no entities.
  - In the interior, we have the 6 signals that are the entities of the DUT.
  - To this, we also add a component (which is the DUT), which is defined separately
  - A port map is defined, which maps the signals of the testbench to the ports of the component.
- Comments in the code begin with `--`.
- **Simulators**: `GHDL` (free), `ModelSim` (proprietary).

Hands-On

- The `compile_ghdl.sh` file provided lists all the files that need to be compiled, which basically includes the main file and testbench.
- The testbench reads the file `TRACEFILE` and writes the output in `OUTPUTS`. This basically automates all possible test cases and checks whether the simulation is working fine.
- For usage instructions on the changes to be made, refer to `../Generic Testbench/README.txt`

Running The Code

Using the generic testbench provided for the `TwoBitAdder`, any circuit of the sort 8-input, 2-output can be synthesised. In this case, the changes required would be:

- Let us say the entity is `component`, declared in `file.vhd`. Note that the definitions in the testbench and DUT would be of the `component`, and the filename is only used for the compilation.
- The definition of the circuit changes in the `file.vhd`, which is the real logic definition of the circuit.
- The testbench holds for this (if the DUT is used, the Testbench is pretty generic, as the name suggests). However, `TRACEFILE` needs to be updated, so that verification can be done.
- In case of a failure, check the `OUTPUT` can be used to verify what went wrong.
- `compile_ghdl.sh` is a wrapper to compile all the required `.vhd` files. This must include the following: (`-a` mode is for analysing; `-m` mode is for)
  - The file `file.vhd`
  - The generic testbench
  - DUT declarations, for port mapping
  - Running the testbench executable?
- *... to be continued.*