UNIVERSITY OF RHODE ISLAND

Using VHDL on the Mentor Graphics Tools - A Tutorial -

Version 1.01

December 4, 1996

Gus Uht

1 Introduction

Greetings and welcome to the fun-filled world of VHDL at URI!

For the newcomer, VHDL stands for "VHSIC Hardware Description Language", with VHSIC standing for "Very High-Speed Integrated Circuit". VHDL was originally developed a decade or two ago as part of the Defense Department's VHSIC program. VHDL is now standardized by the IEEE, and is one of the most (if not the most) common hardware description languages in use in the US today.

Before reading this document, you should be familiar with the basic Mentor Graphics setup in the ELE department at URI, as well as the ELE computing environment (especially the use of the UNIX operating system). These topics are covered in other documents.

I must emphasize that this is not a tutorial or reference on VHDL itself. Many textbooks cover VHDL, and there are some online tutorials from other schools that may help, including:

`http://www.erc.msstate.edu/~reese/vhdl_synthesis/index.htm`

`http://www.eng.auburn.edu/department/ee/mgc/vhdl.html`

Further, there is online Mentor help via Bold Browser, accessible by entering:

`bold_browser`

The primary online documents that are of use are: *Getting Started with QuickHDL and VHDLwrite* (Modules 1 and 2 in general, and pages ix to xi, 1-1 to 1-4, 2-4 to 2-5, and 2-40 to 2-47 in particular; this tutorial differs in some details from that in pages 2-1 to 2-33) and the *QuickHDL User's and Reference Manual.*

The purpose of this document is to guide the first-time user through the maze of Mentor documents, tools and systems to be able to write VHDL descriptions, compile them, and simulate them. Rather than deal with all of the options, I'll simply present a quicky tutorial, elucidating on some of the more interesting options.

There are two main programs used for VHDL: Design Architect (DA) and QuickHDL. Design Architect is used both to enter VHDL code and to compile it. QuickHDL is then used to simulate it. Note that it is possible to combine both VHDL- and schematic-based designs; this is a bit complex, so it will be covered in another document. Also note that Mentor also supports the use of Verilog, the other major hardware description language. However, since this support was just added, Verilog is much less seamlessly integrated into the Mentor suite than VHDL, so I recommend using VHDL.

I will now go through a sample VHDL entry, compilation and simulation session. The VHDL modelling of a simple multiplexor will be used as an example.

2 Entering and Compiling VHDL

You can use either the built-in VHDL editor of Design Architect, or your own favorite text editor. In the latter case, you will have to import the text file. Using the built-in editor has a couple of advantages: (1) it is tightly coupled with the compiler; lines with errors can be immediately highlighted and located; (2) pre-formatted VHDL code *templates* can be used to build your code on; e.g., the skeletal components of an *entity* can be inserted into your program by calling up the entity template.

I'll now go through the required steps, assuming that the built-in editor is used.

1. Within your "`project`" directory, create the following two directories, using either Unix or Design Manager commands:

   `.../project/qhdl_n/src`

```
.../project/qhdl_n/work
```

The `src` directory will hold the source code for all of your VHDL descriptions, while the `work` directory is the *library* used to hold the compiled code.

2. `cd` to the `qhdl_n` directory, and invoke DA:

   ```
   da
   ```

   Once in DA, make sure you are working from the correct directory, by clicking and navigating from the drop-down menus on the menu bar:

   ```
   MGC>Location Map>Set Working Directory:[..../qhdl_n]
   ```

   Also set the display by clicking:

   ```
   MGC>Setup>Session...[Up Down Tiling]
   ```

3. You'll now enter a sample VHDL description of a one-line 2-to-1 MUX. Click the "OPEN VHDL" palette button. (This is on the "Session" palette.) An entry panel will come up. Set the "VHDL Source Name" to:

   ```
   .../qhdl_n/src/muxentity
   ```

   click "Yes" on "Options", and click the "New Numbered Fileset" button. Note: when you reopen the file, click on "Use Existing Type", instead.

   A window will open into which you can either directly enter your VHDL, or copy it from another file that you created earlier. For now, enter the following VHDL entity description:

   ```
   --one bit MUX

   library ieee;
   use ieee.std_logic_1164.ALL;

   entity mux2to1 is
   port(
           signal s,zero,one:      in      std_ulogic;
           signal y:               out     std_ulogic);
   end mux2to1;
   ```

   Save it via the menu bar's "File" drop-down menu.

4. OK, let's compile it.

   (a) First, in the vhdl_palette, click on "Show Current Maps"; this will display the mappings between the logical names of the libraries and their physical path names. Don't worry about the "quickhdl.ini" or the "work" warnings. Ensure that "work" is mapped to "`./work`".

   (Note: if you do any involved VHDL work, you will probably want to add a logical link to the absolute, not relative, pathname of your library. This is a two-step process. First, copy the default "`/usr/local/packages/MGraphics/lib/quickhdl.ini`" initialization file to your `qhdl_n` directory, keeping the same file name. You then add the new mapping to this file, either by editing it with an ASCII text editor [carefully] to add one line for the new mapping, or by using the "Map..." command on the DA menu bar's "QuickHDL" drop-down menu [this appears when DA has no subwindows open]. In the latter case, click on "set", then enter a logical name you will use for your library [arbitrary] in the first box, followed by the full path name of the physical location of the library, e.g., ".../project/qhdl_n/work".)

   Close the display.

   (b) Next, in the Palette, click on "Set Compiler Options...". In the panel, set the Work Library to: `.../qhdl_n/work`, set the QuickHDL Options for "Constraint Checking", "VHDL93" and "Explicit Scoping"; also, click the "Simulation" button and set "Conditional Compilation" to "All"; then click "OK".

(c) Now, actually do the compilation, by clicking "Compile." in the Palette. A new window will come up for the compiler itself, in which any errors will be indicated. Ignore the Warning about "quickhdl.ini". (You might try introducing an error, like changing `signal` to `signa`, to see how this works. The neat thing is that if you click on an "ERROR:..." message, and then click the "Highlight" button, the source line in question will be shaded yellow.)

(d) Once you have compiled with no errors, close the source window; the compilation window will then also close automatically.

5. OK, we've got an entity, let's get an architecture. Enter the following into a new VHDL file called: `.../qhdl_n/src/muxarch`, using the same approach as with the entity file:

```
--my first architecture

architecture first of mux2to1 is
begin
        y <= one when (s='1') else zero;
end first;
```

   Now compile it, following the same steps you did as above for the entity file. (You may have to reset the simulator options.)

   Note: recall that one of the features of VHDL is its ability to have multiple architectures for the same entity (the latter being the I/O sepcification of the design). Normally the architecture(s) are compiled after the entity. If a change is made to an architecture, neither the entity nor other architectures need be recompiled. For simulation, a single architecture of an entity must be chosen to simulate.

3 VHDL Simulation

At this point, you can simulate your new VHDL MUX directly with the QuickHDL simulator. To invoke it, type `qhsim` on a Unix command line (from the `.../qhdl_n` directory), or double-click its button in Design Manager (DMGR). (Note: to keep the execution time down, you might want to exit DA [and DMGR].)

In the "Startup" window, select the desired library ("work"), entity ("mux2to1") and architecture ("first") (note: recall you can have more than one architecture for an entity, so in general you can make the selection here of which to test). Then click on the "Load" button.

Now, `qhsim` has many windows. To see them all, type

```
view *
```

in the (main) simulator window. See the documentation starting at page 5-1 in the *QuickHDL User's and Reference Manual* for a complete description of the different windows.

To get you started, we'll do a quicky simulation, focussing on the "Signals", "Wave" and "List" windows. Normally, you can input stimuli from a file. Here, we'll do it manually, via the "Signals" window.

1. First, we have to get the signals to appear in the Wave and List windows. Do this by clicking the following from the drop-down menus in the menu bar of the Signals window:
   `Wave>Signals in region`
   followed by:
   `List>Signals in region`
   (Alternatively, single signals could have added.)

2. Now, we'll create the stimulus on the three MUX inputs "s", "zero" and "one". This is done via repeated use of the "Force" drop-down menu in the Signals window. First, select "s". Then, click on "Force", and enter the following in the dialog box: "Value:" 0, "Delay:" 0, "Repeat Every:" 1000. "Kind:" is "Freeze", and is that value for all of the following entries; click the "Force" button. (NOTE: the signals are not displayed in the Wave or List windows until the simulation begins.)

   Similarly, enter the following, each line of settings in a different "Force" invokation:

for "s":
0, 0, 1000 -(already done)
1, 500, 1000

for "zero":
0, 100, 200
1, 200, 200

for "one":
0, 100, 400
1, 200, 400

Now let's simulate! In the main window (command window, the first one), click on the "Run" button repeatedly, until you have simulated about 1000 ns. In the Wave window, click:

```
Zoom>Full Size
```

to see a big picture of the trace. Convince yourself that what you see indicates that the MUX is working.

Congratulations! You've done it!

One last thing: print the timing chart by writing a Postscript version of it to a file:

```
File>Write Postscript...[OK]
```

and then printing the file, e.g.:

```
lpr -Pqms wave.ps
```

I've attached a sample output for the MUX, following everything done in the tutorial.

That's it!

Last note: you can exit all of the QuickHDL windows at once by killing the main QuickHDL window.

Good luck!