

ModelSim Tutorial Using VHDL

Nick Gamroth

April 2005

Abstract

Here's a tutorial on using ModelSim. I like VHDL, so all the code is VHDL. If you're using Verilog or SystemC or anything else, you can go to hell.

1 Introduction

I'm not all that good at this stuff, so you might want to find a better tutorial. Anyways, I'm going to make a 14 bit 2's complement converter. I'm even going to test it! To do this conversion, all we need to do is invert our input signal and add 1 to it. Simps!

Special note: I'm doing all this stuff on windows, even though my backslashes are forward slashes. This is because I'm lazy and there's not a simple control sequence in LaTeX to get a backslash. Or something like that, I don't know.

2 Create a Project

So I guess the first thing to do is create a ModelSim project. You do this with File — New — Project. In the dialog box that opens, give the project a name. I'm naming mine `two_comp` for two's complement converter. I made a directory in my ModelSim home directory (`c:/modeltech_ae/work`) that holds all of my ModelSim files. So, back to the dialog box. I've already got the project name, now I put in `c:/modeltech_ae/work/two_comp` for the project directory. Leave the Default Library Name as 'work'. Then press OK! When you do this, the main ModelSim window should change some, so that you have a tab on the left side named 'Project'.

I like to keep things organized, so I make a folder for my VHDL code files. To do this, right click in the Project window and click Add

to Project — Folder. I name this folder vhdl. Then click OK. Now you've got folder in your project!

Next let's make some VHDL!

3 Writin' on some VHDL

First, we need to make a file to add to the project. We'll start off with a 14 bit adder. In ModelSim, click File — New — Source — VHDL. You'll get a VHDL editor window. If you want, you can paste my code into it, or write your own 14 bit adder, I don't care! Here's the code:

```
library ieee;
use ieee.std_logic_1164.all;
entity half_add is port(
    x:      in std_logic;
    y:      in std_logic;
    s:      out std_logic;
    c:      out std_logic
);
end half_add;

architecture ha_arch of half_add is
begin
    s <= x xor y;
    c <= x and y;
end ha_arch;

library ieee;
use ieee.std_logic_1164.all;
entity full_add is port(
    x:      in std_logic;
    y:      in std_logic;
    z:      in std_logic;
    s:      out std_logic;
    c_o:    out std_logic
);
end full_add;

architecture fa_arch of full_add is
component half_add port(
    x:      in std_logic;
    y:      in std_logic;
```

```
        s:      out std_logic;
        c:      out std_logic
    );
end component;
signal hs, hc, tc: std_logic;
begin
    ha1: half_add
        port map(x, y, hs, hc);
    ha2: half_add
        port map(hs, z, s, tc);
    c_o <= tc or hc;
end fa_arch;

library ieee;
use ieee.std_logic_1164.all;
entity add_14 is port(
    a:      in std_logic_vector(13 downto 0);
    b:      in std_logic_vector(13 downto 0);
    cin:    in std_logic;
    s:      out std_logic_vector(13 downto 0);
    cout:   out std_logic
);
end add_14;

architecture add_arch of add_14 is
component full_add port(
    x:      in std_logic;
    y:      in std_logic;
    z:      in std_logic;
    s:      out std_logic;
    c_o:    out std_logic    -- carry out
);
end component;
signal c: std_logic_vector(14 downto 0);
begin
    b_0: full_add
        port map(a(0), b(0), c(0), s(0), c(1));
    b_1: full_add
        port map(a(1), b(1), c(1), s(1), c(2));
    b_2: full_add
        port map(a(2), b(2), c(2), s(2), c(3));
    b_3: full_add
        port map(a(3), b(3), c(3), s(3), c(4));
```

```
b_4: full_add
      port map(a(4), b(4), c(4), s(4), c(5));
b_5: full_add
      port map(a(5), b(5), c(5), s(5), c(6));
b_6: full_add
      port map(a(6), b(6), c(6), s(6), c(7));
b_7: full_add
      port map(a(7), b(7), c(7), s(7), c(8));
b_8: full_add
      port map(a(8), b(8), c(8), s(8), c(9));
b_9: full_add
      port map(a(9), b(9), c(9), s(9), c(10));
b_10: full_add
      port map(a(10), b(10), c(10), s(10), c(11));
b_11: full_add
      port map(a(11), b(11), c(11), s(11), c(12));
b_12: full_add
      port map(a(12), b(12), c(12), s(12), c(13));
b_13: full_add
      port map(a(13), b(13), c(13), s(13), c(14));
c(0) <= cin;
cout <= c(14);
end add_arch;
```

You're probably saying, that's a lot of code for a stupid adder. Well, it is! But that's an adder. Anyways, save that file with File — Save. Make sure it goes into your project directory. Oh, and I called mine `add_14.vhd`.

We also need to add that file to the project, so back in the main ModelSim window, in the Project tab on the left, right click on the `vhdl` folder and go Add to Project — Existing File. Then browse to your file, and make sure the Folder drop-down says `vhdl`. Then click OK!

4 Stimulate That Adder

Let's run a quick simulation of the adder to make sure it adds stuff. Since this project only needs to add 1 to things, I'm going to be lazy and only add 1 to every number between 0 and 2^{14} . This is usually a bad idea most of the time!!1

4.1 DO it with DO Files

Wow, ModelSim is awesome. You can make TCL files that it'll process to do your bidding. I really only use this awesome power to automate simulations. Anyways, these files are called do files, and they're pretty simple. We're going to make one that opens up some windows and puts some signals into our adder. I'll just list the code for it here:

```
# you need this line to start the simulation
vsim work.add_14
# if you want to run it again, you need to restart the simulation
# to clear out the results from the old one
restart -force -nowave

# these lines add signal traces in the wave window
add wave a
add wave b
add wave cin
add wave s
add wave cout

# this mess forces signals onto one of the adder inputs
# i don't know a way to make a binary counter signal
# so i did it like this, just force a clock signal on
# each input so that it comes out like a counter
force -deposit /add_14/a(0) 0 0, 1 5 -r 10
force -deposit /add_14/a(1) 0 0, 1 10 -r 20
force -deposit /add_14/a(2) 0 0, 1 20 -r 40
force -deposit /add_14/a(3) 0 0, 1 40 -r 80
force -deposit /add_14/a(4) 0 0, 1 80 -r 160
force -deposit /add_14/a(5) 0 0, 1 160 -r 320
force -deposit /add_14/a(6) 0 0, 1 320 -r 640
force -deposit /add_14/a(7) 0 0, 1 640 -r 1280
force -deposit /add_14/a(8) 0 0, 1 1280 -r 2560
force -deposit /add_14/a(9) 0 0, 1 2560 -r 5120
force -deposit /add_14/a(10) 0 0, 1 5120 -r 10240
force -deposit /add_14/a(11) 0 0, 1 10240 -r 20480
force -deposit /add_14/a(12) 0 0, 1 20480 -r 40960
force -deposit /add_14/a(13) 0 0, 1 40960 -r 81920

# for this simulation, i put 1 on the second input and 0 on cin
# i think you can use hex values, but i'm not sure how it works
# with 14 bits, so i just did the bits
force -freeze /add_14/b 00000000000001 0
```

```
force -freeze /add_14/cin 0 0

# this line opens up the wave window to look at waves
view wave

# the run command makes the simulation run!
run 81920
```

OBVIOUSLY, you should save that file. I saved it as `test_adder.do`, and added it to my project under the top level. You probably want to run it, which you can do by typing:

```
do test_adder.do
```

A wave window should appear with the results of your simulation. It's a lot of data. I just looked at a bunch of them and decided the adder works.

5 The Rest of the Logic

Ok, the only other component we need right now is a 14 bit inverter. I'm not going to explain anything about it since you set it up the same as the adder. Here's the VHDL:

```
library ieee;
use ieee.std_logic_1164.all;

entity inv_14 is port(
    input:  in std_logic_vector(13 downto 0);
    output: out std_logic_vector(13 downto 0)
);
end inv_14;

architecture inv_arch of inv_14 is
begin
    process(input)
    begin
        output <= not input;
    end process;
end inv_arch;
```

I saved that file as `inv_14.vhd` and put it in the `vhdl` folder. You can simulate it if you want:

```
vsim work.inv_14
restart -force -nowave

add wave input
add wave output

force /inv_14/input 00000000000000 0
force /inv_14/input 01010101010101 100
force /inv_14/input 10101010101010 200
force /inv_14/input 00110011001100 300
force /inv_14/input 11001100110011 400

view wave

run 500
```

I cleverly called that file test_inv.do. The important part about this file is the first line:

```
vsim work.inv_14
```

If you don't put that inv_14 part in there, it'll probably try and simulate your adder. In any case, something will be screwed up if you don't change that line.

6 Put it Together

This section is probably more about VHDL than ModelSim, but it's something I forget a lot, so I'll put it in here.

Sometimes, I like to draw wires and stuff because it's fun, but you can't do that in ModelSim because it only likes VHDL. But we still want to wire that adder and inverter together, and we don't want to leave ModelSim. SO! Let's make a VHDL file that crams those two modules together and makes a real live (well, not really) two's complement converter!

The astute reader will have noticed that this is exactly what we did in the adder. It's made up of a bunch of half adders and stuff, which are all crammed into a complete 14 bit adder using port maps. So, let's do the same thing here:

```
-- 14 bit 2's complement converter
-- Nick Gamroth

library ieee;
```

```
use ieee.std_logic_1164.all;

entity two_comp is port(
    input:  in std_logic_vector(13 downto 0);
    output: out std_logic_vector(13 downto 0);
    carry:  out std_logic
);
end two_comp;

architecture tc_arch of two_comp is
component inv_14 port(
    input:  in std_logic_vector(13 downto 0);
    output: out std_logic_vector(13 downto 0)
);
end component;

component add_14 port(
    a:      in std_logic_vector(13 downto 0);
    b:      in std_logic_vector(13 downto 0);
    cin:    in std_logic;
    s:      out std_logic_vector(13 downto 0);
    cout:   out std_logic
);
end component;

-- this is the temporary signal between the inverter
-- and the adder
signal temp: std_logic_vector(13 downto 0);

begin
    inv: inv_14
        port map(input, temp);
    adder: add_14
        portmap(temp, "00000000000001", '0', output, carry);

end tc_arch;
```

And there it is! You can see that this looks different than `add_14.vhd`. It doesn't have every component defined in the same file, which is just fine. You can put them all in one file, or break them into multiple files, and ModelSim will figure it out, as long as you use the same names and everything.

So you can simulate this whole thing and see if it works. It works for me, so I'm happy! Bye!