Modelsim Simulation & Example VHDL Testbench

Agenda

- Simulating a VHDL design with a VHDL Testbench
- Generating a sample testbench from Quartus
- Modifying the testbench
- Procedure creation and Procedure calls
- Create a script for easy recompiling and simulation within Modelsim
- Adding self checking and reporting via a VHDL monitor process

Top Level Design File

- Top level FPGA vhdl design, our test bench will apply stimulus to the FPGA inputs.
- The design is an 8 bit wide 16 deep shift register.

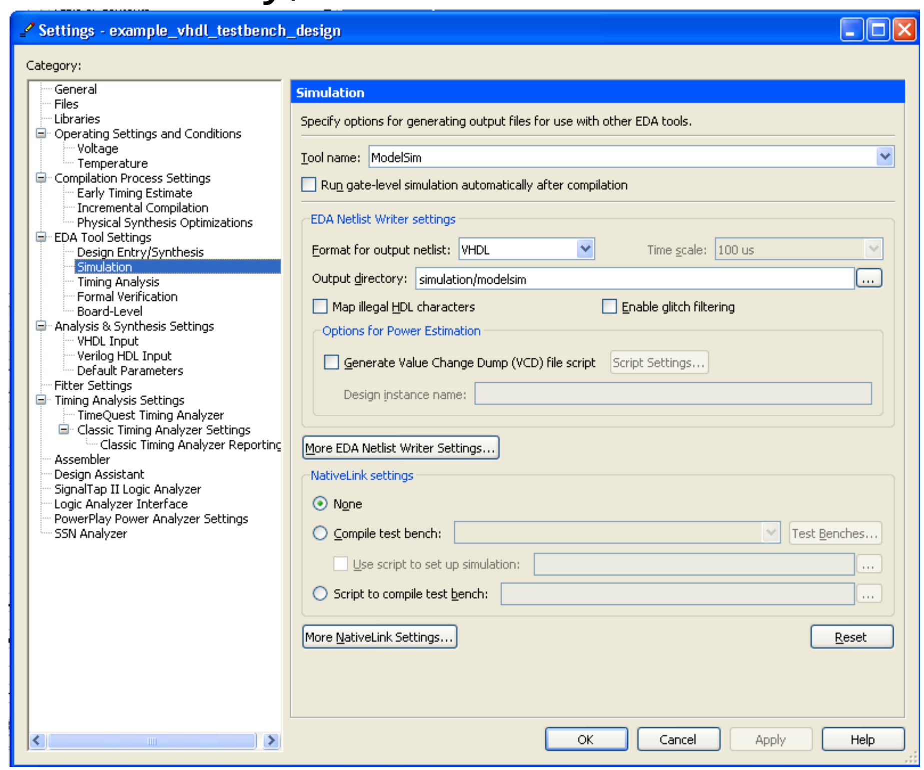
```
-- example design for showing vhdl testbench

library ieee;
use ieee.std_logic_1164.all;

entity example_vhdl is
  Port
    (
    Inputclk  : in std_logic; -- input clock
    sclr      : in std_logic; -- active high clear
    en        : in std_logic; -- active high enable
    data_in   : in std_logic_vector(7 downto 0); -- input data
    data_out  : out std_logic_vector(7 downto 0) -- output data
    );
end example_vhdl;

architecture rtl of example_vhdl is

  signal mem_data : std_logic_vector(7 downto 0);

begin

-- 16 deep shift register, shifts in the 8 bits and delays
-- by 16 clock cycles, and then shifts out
-- direct instantiation, do not need to define the component
   shift_reg_inst : entity work.shift_reg(SYN)
   port map
      (
      aclr      => sclr,
      clken     => en,
      clock     => Inputclk,
      shiftin   => data_in,
      shiftout  => mem_data,
      taps      => open
      );

    data_out <= mem_data; -- send the shifted data off chip

end architecture rtl;
```

I/O portion of the design

Design instantiates an alt_shift_taps megawizard function, 16 deep, 8 bit wide shift register, will require altera_mf library For simulation.

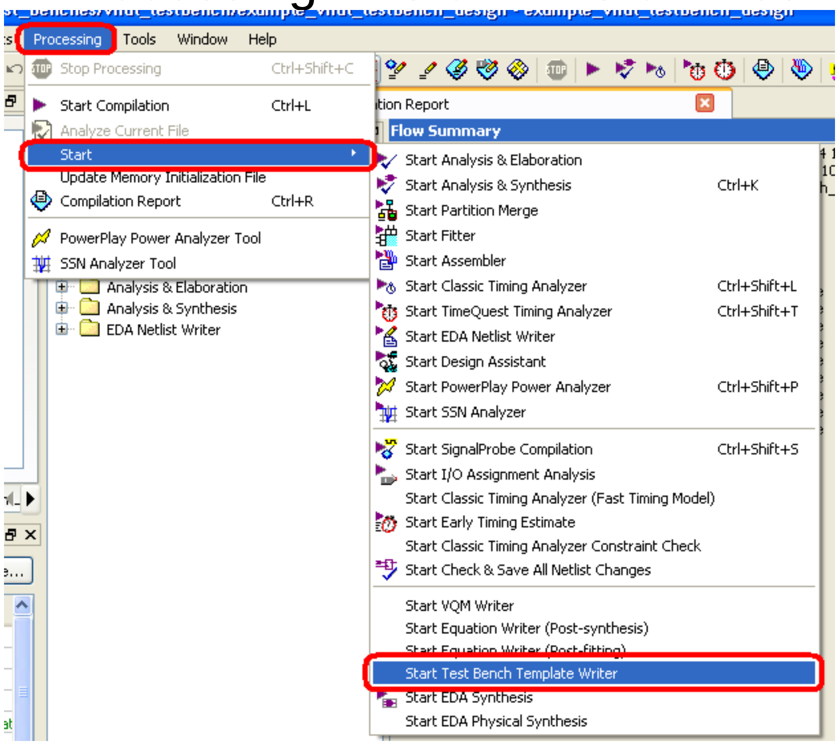
Set-up Quartus to Generate Sim Directory

- Setup Quartus to generate a simulation directory for Modelsim
  - Simulation .vho (structural netlist) and .vht (testbench) files are generated and placed in this directory, default is ./simulation/modelsim
  - Assignments->Settings
  - Then [Simulation]



Create and Example Testbench

- Perform and Analysis and Elaboration on the design in Quartus, then generate the testbench structure, which is a good place to start the testbench design
  - Processing -> Start -> Start Test Bench Template Writer

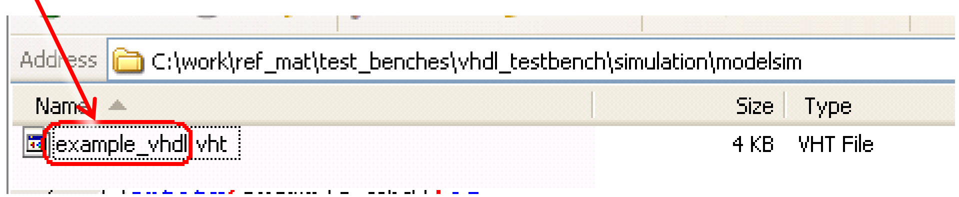


Only run this once to get the structure. If you run again, you will overwrite all Your changes, so may be a good idea To change the file name to prevent Overwriting.

Example Testbench File

- The ./simulation/modelsim directory now contains the example_vhdl.vht file. The file name (example_vhdl) is derived from the top level entity name.

```vhdl
-- example design for showing vhdl testbench

library ieee;
use ieee.std_logic_1164.all;

entity example_vhdl is
  Port
    (
    Inputclk   : in std_logic; -- input clock
    sclr       : in std_logic; -- active high clear
    en         : in std_logic; -- active high enable
    data_in    : in std_logic_vector(7 downto 0); -- input data
    data_out   : out std_logic_vector(7 downto 0) -- output data
    );
end example_vhdl;
```



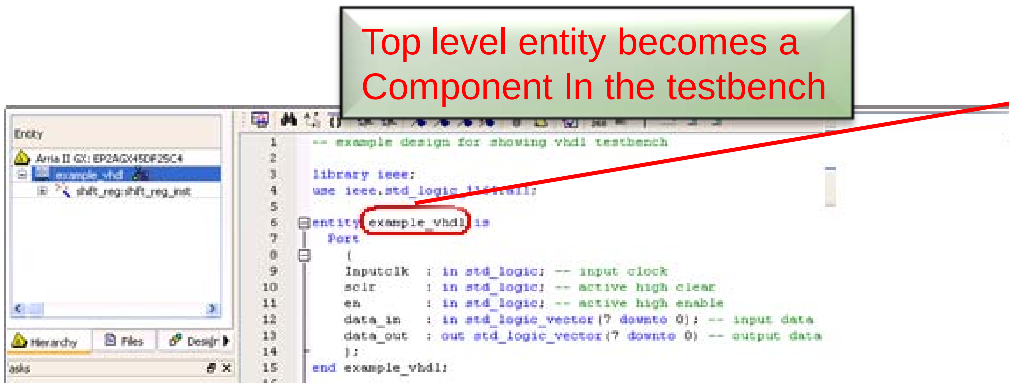
Example Testbench File

- The first thing you'll notice about the testbench, is that the top level entity has no I/O.
  - It is simply an "entity name is", and "end entity name".
  - This makes sense as there is no I/O in a testbench

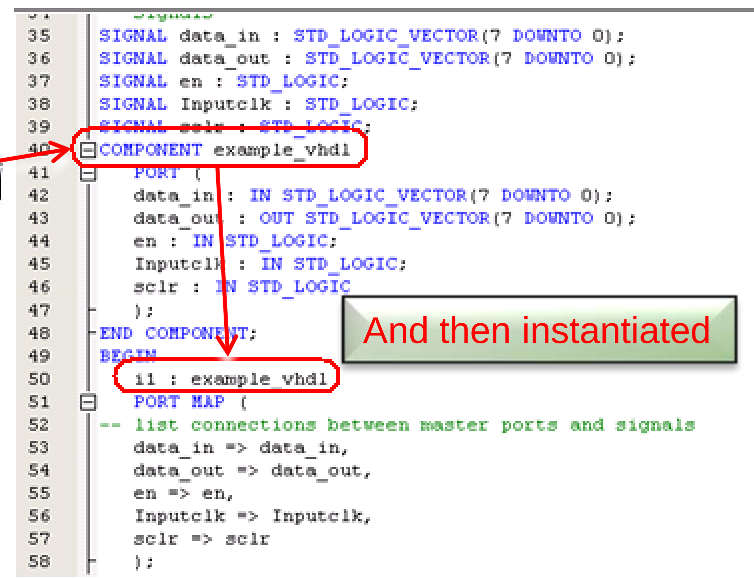
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY example_vhdl_vhd_tst IS
END example_vhdl_vhd_tst;
ARCHITECTURE example_vhdl_arch OF example_vhdl_vhd_tst IS
```

Example Testbench File

- The testbench creates some signals to connect the stimulus to the Device Under Test (DUT) component. The DUT is the FPGA’s top level design. In our case example_vhdl. (example_vhdl is the top level entity of our FPGA design)



Quartus
example_vhdl.vhd
(top level design file)



example_vhdl.vht
(testbench file)

Example Testbench File

- The next section is where the stimulus will reside, the Quartus generated .vht (testbench file) does not contain any stimulus, this must be added to perform a simulation
- The .vht generated file provides the structure

```
init : PROCESS
-- variable declarations
BEGIN
        -- code that executes only once
WAIT;
END PROCESS init;
always : PROCESS
-- optional sensitivity list
-- (          )
-- variable declarations
BEGIN
        -- code executes for every event on sensitivity list
WAIT;
END PROCESS always;
END example_vhdl_arch;
```

Creating Tesbench Clock

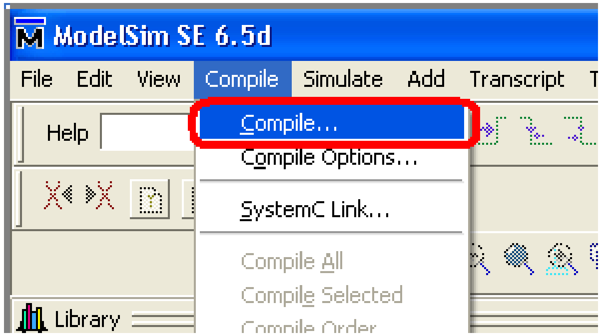
- We now need to add in some stimulus into the testbench. This design is simply a shift register with data in, data out, clock, clear, and enable.
- Let's start with a free running clock. Directly after the DUT (example_vhdl) instantiation, add line 61 below, this will create a free running 20Mhz clock, but we need to supply a default value. We can do this at the signal declaration, add a ":= '0'" to set the signal to a logic '0'

```
35  SIGNAL data_in : STD_LOGIC_VECTOR(7 DOWNTO
36  SIGNAL data_out : STD_LOGIC_VECTOR(7 DOWNT
37  SIGNAL en : STD_LOGIC;
38  SIGNAL Inputclk : STD_LOGIC := '0';
39  SIGNAL sclr : STD_LOGIC;
40  COMPONENT example_vhdl
```

```
58      );
59
60  -- add in a free running clock of 20Mhz/50ns cycle
61  inputclk <= not inputclk after 50 ns;
62
63  init : PROCESS
64  -- variable declarations
```

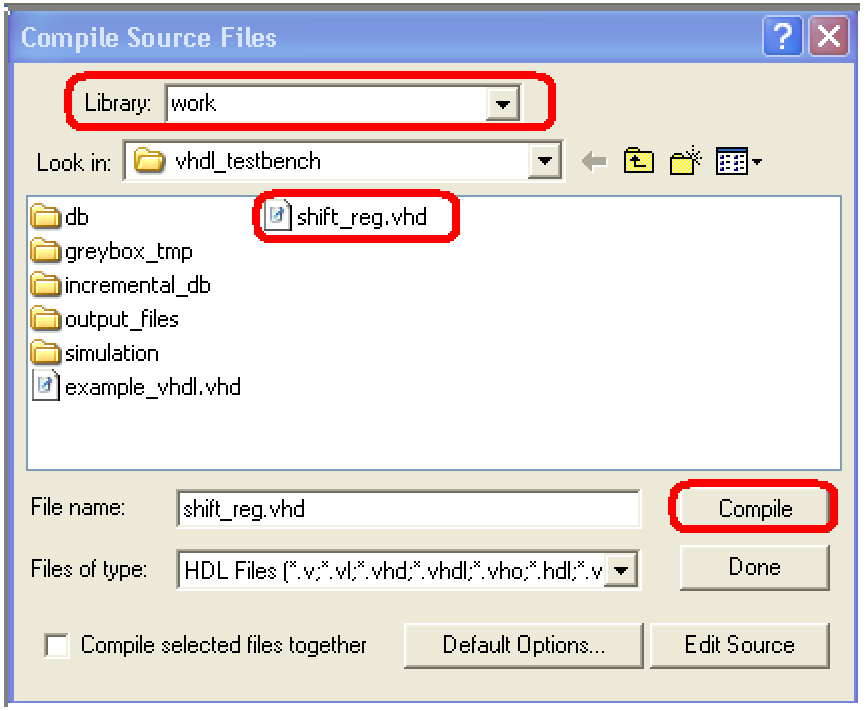
Compiling Design Files in Modelsim

- Let's now take the design and testbench into Modelsim
- Open up Modlesim and from the prompt:
  - Change directory into your modelsim directory (the directory created by Quartus)
    - ModelSim> cd C:/work/ref_mat/test_benches/vhdl_testbench/simulation/modelsim
  - create a new library called work, creates a directory called work (do only once)
    - ModelSim> vlib work
  - Map logical library work to directory work (do only once)
    - ModelSim> vmap work work
  - Compile the underlying design files, including other libraries, start with the lowest level design file in the project, and the last file compiled will be the testbench.
    - Go to Compile->Compile…



Compiling Design Files in Modelsim

- The Following GUI pops up, specify library work, shift_reg.vhd, and click compile



Compiling Design Files in Modelsim

- The following Error will occur if not using Altera Modelsim
  - Altera Modelsim includes the Altera pre-compiled libraries

```
vcom -reportprogress 300 -work work C:/work/ref_mat/test_benches/vhdl_testbench/shift_reg.vhd
# Model Technology ModelSim SE vcom 6.5d Compiler 2009.11 Nov 18 2009
# -- Loading package standard
# -- Loading package std_logic_1164
# ** Error: C:/work/ref_mat/test_benches/vhdl_testbench/shift_reg.vhd(39): Library altera_mf not found.
# ** Error: C:/work/ref_mat/test_benches/vhdl_testbench/shift_reg.vhd(40): (vcom-1136) Unknown identifier "altera_mf".
# ** Error: C:/work/ref_mat/test_benches/vhdl_testbench/shift_reg.vhd(42): VHDL Compiler exiting

ModelSim>
```

  - The shift_reg.vhd file calls out the library altera_mf, the shift_reg.vhd was created from a megawizard

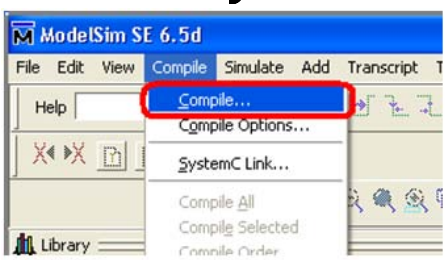
```vhdl

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.all;

ENTITY shift_reg IS
    PORT
```

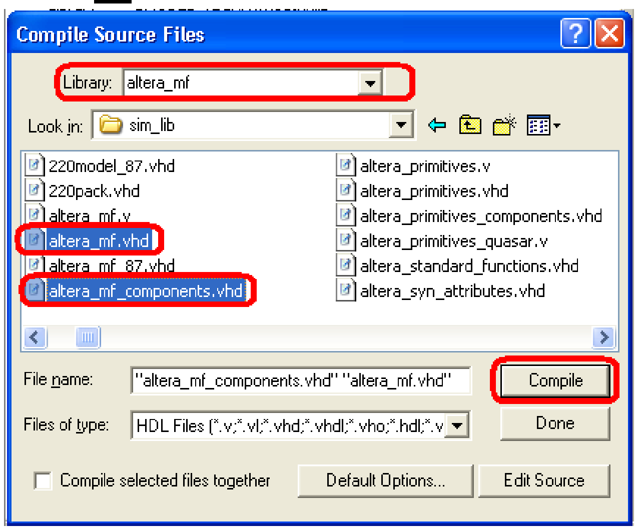
The shift_reg.vhd calls out the altera_mf library

  - So we need to create another library called altera_mf and compile the altera_mf files into that library

Compiling Library Files in Modelsim

- Create the altera_mf library
  - Modelsim> vlib altera_mf
  - Modelsim> vmap altera_mf altera_mf
- The library files we need to compile are located in the Quartus install directory, under sim_lib, see below:





- Then compile into altera_mf
  - altera_mf.vhd
  - altera_mf_components.vhd
    - Can select both at the same time

Compiling Design Files in Modelsim

- Now that we have altera_mf compiled, we can now compile shift_reg.vhd again.
- You can use the up arrow from the Modelsim command line prompt to find the command to run, then hit enter
  - ModelSim> vcom -reportprogress 300 -work work C:/work/ref_mat/test_benches/vhdl_testbench/shift_reg.vhd

```
ModelSim> vcom -reportprogress 300 -work work C:/work/ref_mat/test_benches/vhdl_testbench/shift_reg.vhd
# Model Technology ModelSim SE vcom 6.5d Compiler 2009.11 Nov 18 2009
# -- Loading package standard
# -- Loading package std_logic_1164
# -- Compiling entity shift_reg
# -- Compiling architecture syn of shift_reg

ModelSim>
```

No library errors

Compiling Design Files in Modelsim

- Compile the top level design file; example_vhdl.vhd, either from command line or GUI

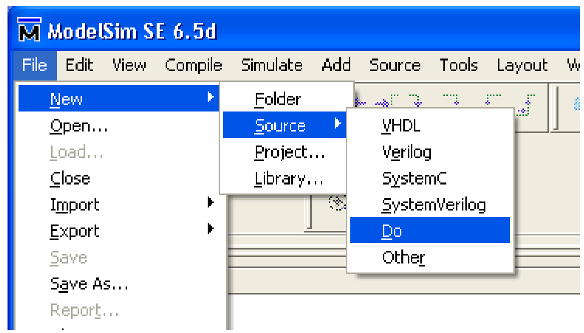
```
vcom -reportprogress 300 -work work C:/work/ref_mat/test_benches/vhdl_testbench/example_vhdl.vhd
# Model Technology ModelSim SE vcom 6.5d Compiler 2009.11 Nov 18 2009
# -- Loading package standard
# -- Loading package std_logic_1164
# -- Compiling entity example_vhdl
# -- Compiling architecture rtl of example_vhdl
# -- Loading entity shift_reg

ModelSim>
```

- And finally compile the testbench, from the ./simulation/modelsim directory, example_vhdl.vht

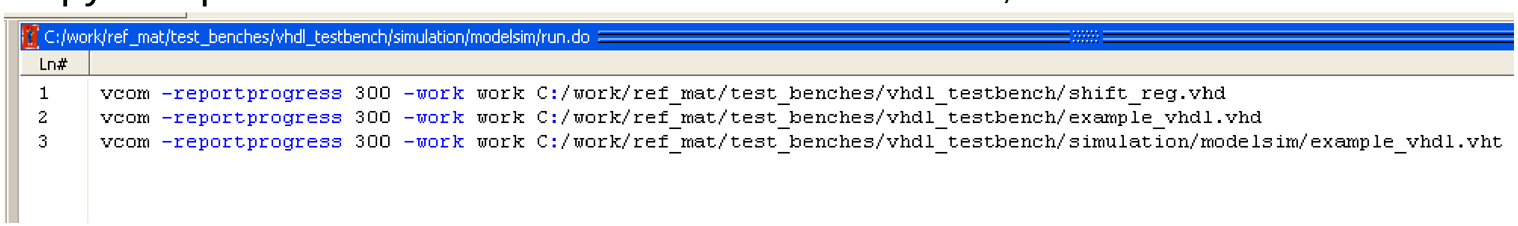
```
vcom -reportprogress 300 -work work C:/work/ref_mat/test_benches/vhdl_testbench/simulation/modelsim/example_vhdl.vht
# Model Technology ModelSim SE vcom 6.5d Compiler 2009.11 Nov 18 2009
# -- Loading package standard
# -- Loading package std_logic_1164
# -- Compiling entity example_vhdl_vhd_tst
# -- Compiling architecture example_vhdl_arch of example_vhdl_vhd_tst

ModelSim>
```

Creating Compile Script

- Since simulating your design is an iterative process, you will find yourself compiling the design files regularily. So we will create a run.do file so we can script the compilation, and eventually the running of the simulation.
  - New->Source->Do



  - Copy and paste the vcom commands into the .do file, and then save as run.do

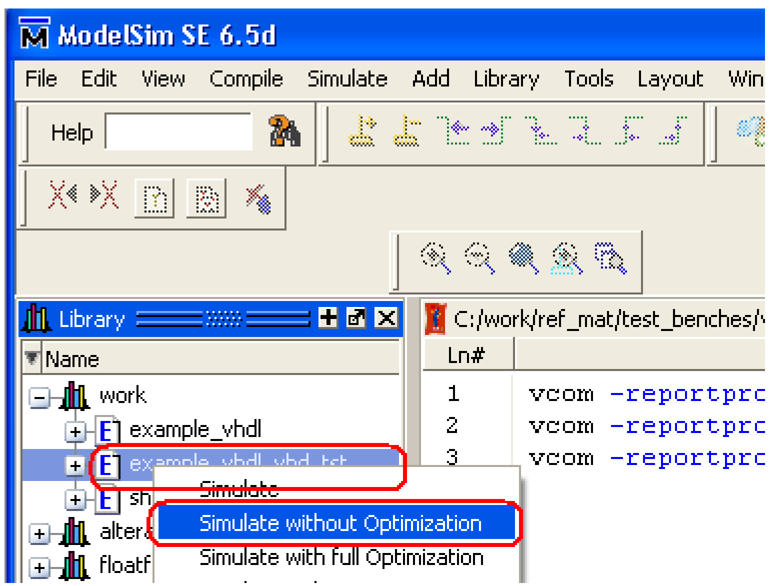


  - To run, type “do run.do” from the modelsim prompt

Simulating the Design

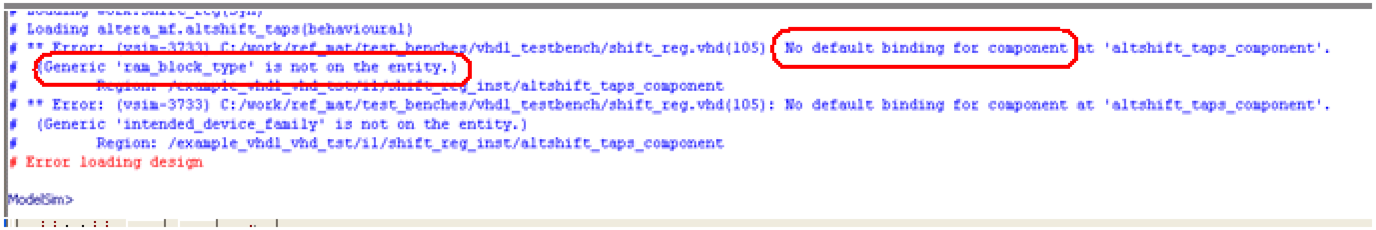
- Let's run the simulation and see what we get
- From the library, highlight "example_vhdl_vhd_test" and right click and "simulate without Optimization"
  - Why example_vhdl_vhd_test?
  - That is the entity name of the
  - Testbench.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY example_vhdl_vhd_tst IS
END example_vhdl_vhd_tst;
ARCHITECTURE example_vhdl_arch OF example_vhdl_vhd_tst IS
```



Simulating the Design

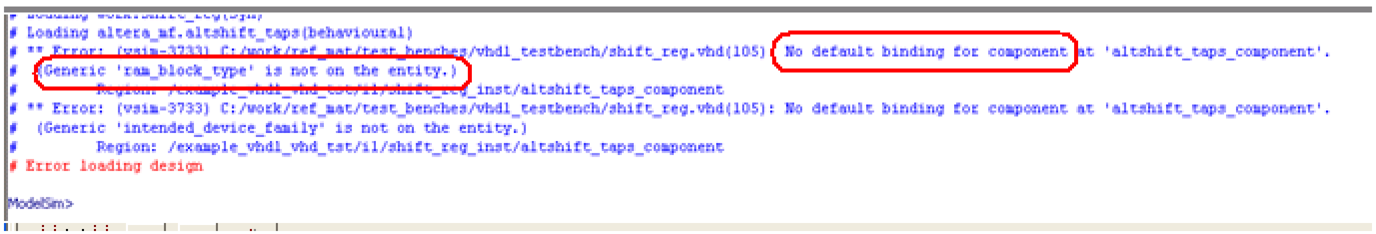
- we have an error…



- The "No default binding for component" is pretty common, it simply means that the component that we have instantiated, does not match up with any entity that we compiled. They have to match exactly for the component to bind with the lower level entity. The lower level entity is part of the altera_mf library for the altshift_taps_compenent. Typically means that we are compiling an out of date file.

Simulating the Design

- The Quartus10.0 altera_mf altshift_taps_component was missing a VHDL generic “ram_block_type”. Quartus 10.1 has fixed this issue, so we will recompile the 10.1 altera_mf library, follow the same steps from slide 14 above, except point to the 10.1 directory structure



- Another way around these types of issues is to simply edit the VHDL.

Simulating the Design

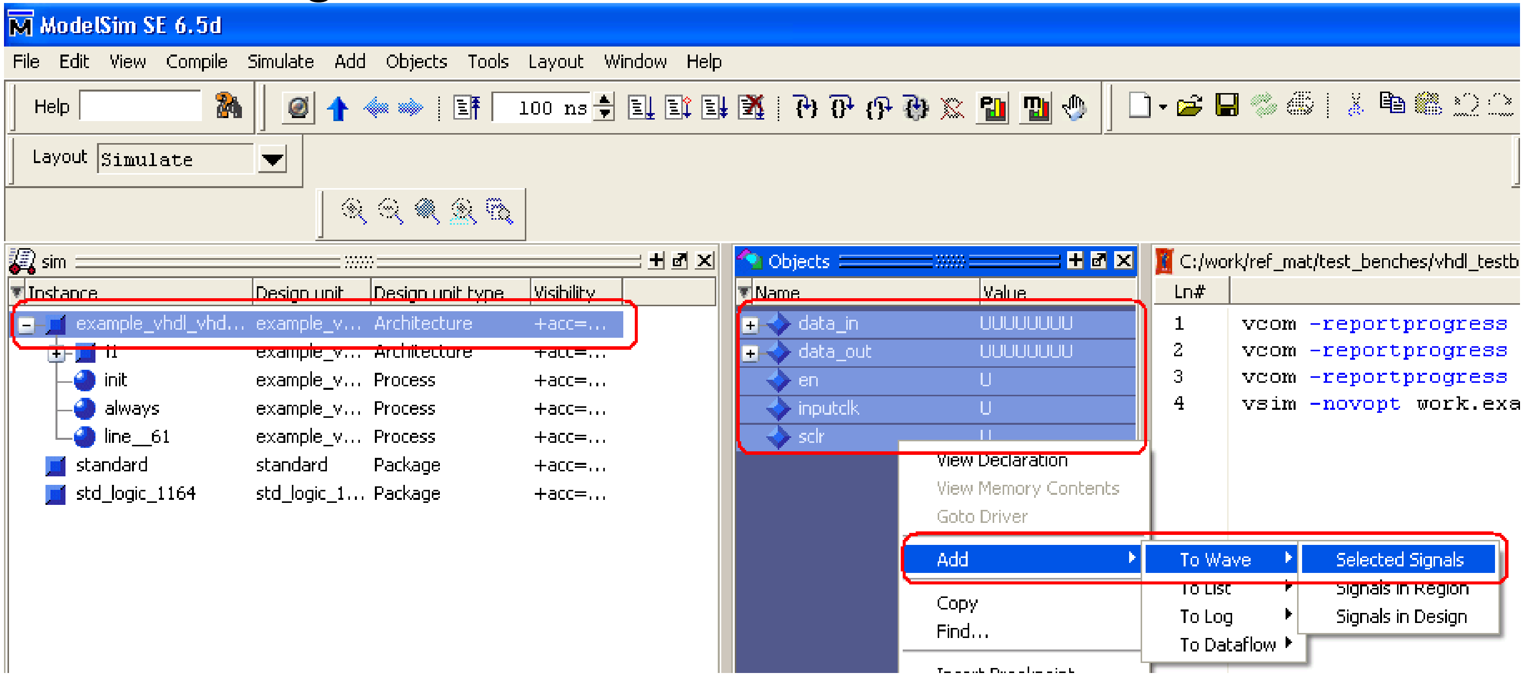
- We have now successfully compiled all the design files and loaded all design files for simulation, with the top level file being the testbench.

```
ModelSim> vsim -novopt work.example_vhdl_vhd_tst
# vsim -novopt work.example_vhdl_vhd_tst
# Loading std.standard
# Loading ieee.std_logic_1164(body)
# Refreshing C:\work\ref_mat\test_benches\vhdl_testbench\simulation\modelsim\work.example_vhdl_vhd_tst(example_vhdl_arch)
# Loading work.example_vhdl_vhd_tst(example_vhdl_arch)
# Refreshing C:\work\ref_mat\test_benches\vhdl_testbench\simulation\modelsim\work.example_vhdl(rtl)
# Loading work.example_vhdl(rtl)
# Refreshing C:\work\ref_mat\test_benches\vhdl_testbench\simulation\modelsim\work.shift_reg(syn)
# Loading work.shift_reg(syn)
# Loading altera_mf.altshift_taps(behavioural)

VSIM 20>
```

- Let’s add the vsim command to our run.do script. Simply copy the “vsim –novopt work.example_vhdl_vhd_tst” command to the run.do file

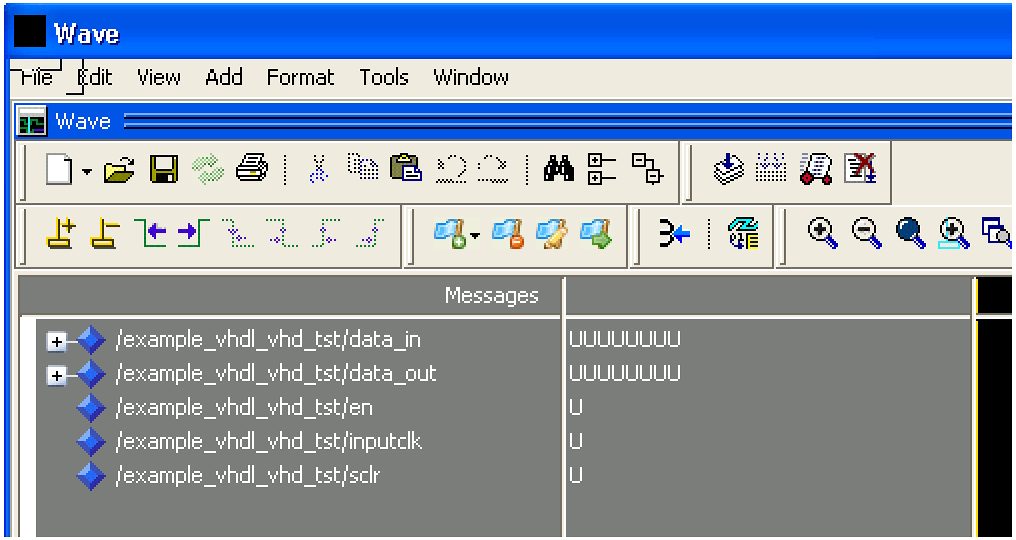
Adding Signals to the Wave Window

- Let's add signals so we can view what is going on. From modelsim, with the top level tesbench highlighted, select all the Objects (data_in, data_out, etc) and right click and add the signals to the wave



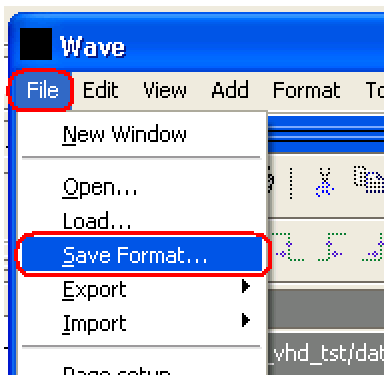
Wave Window

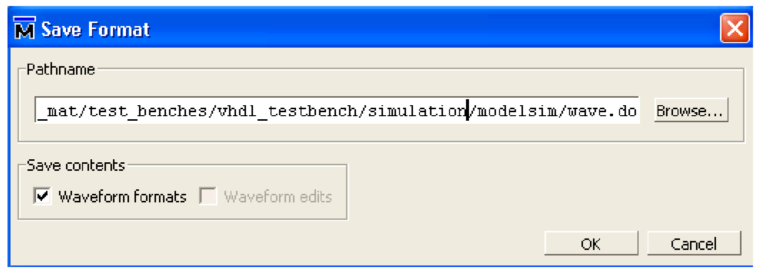
- The Wave window should now open up with our desired signals, these are the signals in our device under test (DUT); our FPGA.



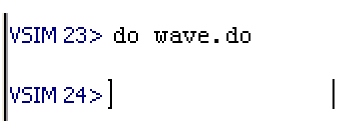
Save the Wave Window

- We now want to save the wave window, so we don't lose our signals and then modify the run.do script.
- File->Save Format… save the wave.do file



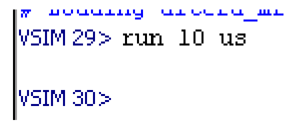


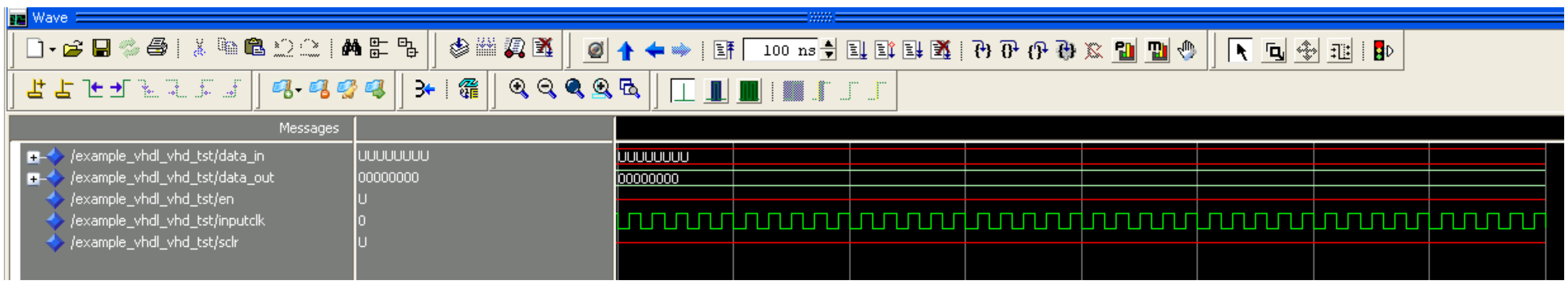
- To reload the signals from our saved wave.do, type
  - do wave.do
  - Then add the command "do wave.do" to the run.do script.



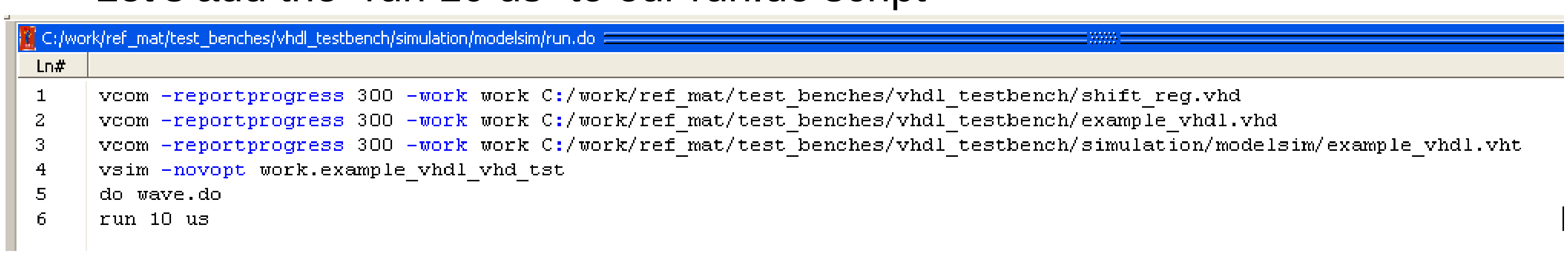
Wave Window with Clock Generated

- We are now ready to run and advance the simulator time
- From the modelsim window type
  - run 10 us
- This will run the simulation for 10 us.





  - From above, you can see we have a free running clock, and the data_out is at 0's.
  - Let's add the "run 10 us" to our run.do script



More Testbench Design

- Note: There are many ways to do testbenches, this will show one way using procedures.
- Let's create a procedure to assign all signals and a procedure to sync up to the rising edge of the clock and wait for x number of clock cycles
  - Procedures are defined within a process
- Then an easy way to exit the simulation will be added to the end of the testbench.

Testbench VHDL Procedures

- Below is the beginning of of our stimulus

```
-- create a process with no sensitivily list, this will be where we create the
-- stimulus for the te
stimulus : process

  -- put all signals in here to give them a defaults value, this are signals that
  -- inputs to your FPGA design, as your I/O grows in your design, continue to add
  -- in the signals.  this procedure will be called out first.
  procedure p_stable is
    begin
      en        <= '0';
      data_in   <= (others => '1');
      sclr      <= '0';
  end procedure p_stable;

  -- call this procedure to wait a variable number of clock cycles
  -- and to sync up to the rising edge of the clock, it will wait
  -- until the inputclk has a rising edge and then will loop based
  -- on the input integer value
  procedure p_sync_app (constant c_loop : integer) is
    variable i:integer:=0;
  begin
    loop1: while i <= c_loop loop -- for i in 1 to c_loop loop
      wait until inputclk'event and inputclk='1';
       i:=i+1;
     end loop loop1;
   end p_sync_app;

  begin
    -- call p_stable once to get all your signals set at time 0
    p_stable;
    -- this will sync
    p_sync_app(5);

    -- assert statement
    -- will stop the simulation, this is a handy way to stop the simulation at the end of all your

    assert false report "----- Not a Failure it is the END OF SIMULATION --------"
     severity failure;

  end process stimulus;
```

Process with no sensitivity list

Assign all inputs to give them a default value

Procdure to sync up to the rising edge clock
And then wait an integer number of clock cycles

After the process' begin statement, call p_stable and then
p_sync_app (5) which will take the simulator to the 6th rising edge

Handy way to exit from the simulation
Allows the use of "run –all"

End the process

Example VHDL Testbench

- Since we modifed our testbench design, we will need to recompile that file and reload the simulation. We could do this a step at a time, but we have created our run.do script, let's make a small change, change to "run –all" instead of "run -10 us"

C:/work/ref_mat/test_benches/vhdl_testbench/simulation/modelsim/run.do

```
vcom -reportprogress 300 -work work C:/work/ref_mat/test_benches/vhdl_testbench/shift_reg.vhd
vcom -reportprogress 300 -work work C:/work/ref_mat/test_benches/vhdl_testbench/example_vhdl.vhd
vcom -reportprogress 300 -work work C:/work/ref_mat/test_benches/vhdl_testbench/simulation/modelsim/example_vhdl.vht
vsim -novopt work.example_vhdl_vhd_tst
do wave.do
run -all
```

- From the modelsim prompt type
  - restart –f …. This will restart the simulator and clean out the wave window
  - do run.do …. Recompiles all our files, reloads for simulaion and then loads the wave window and runs

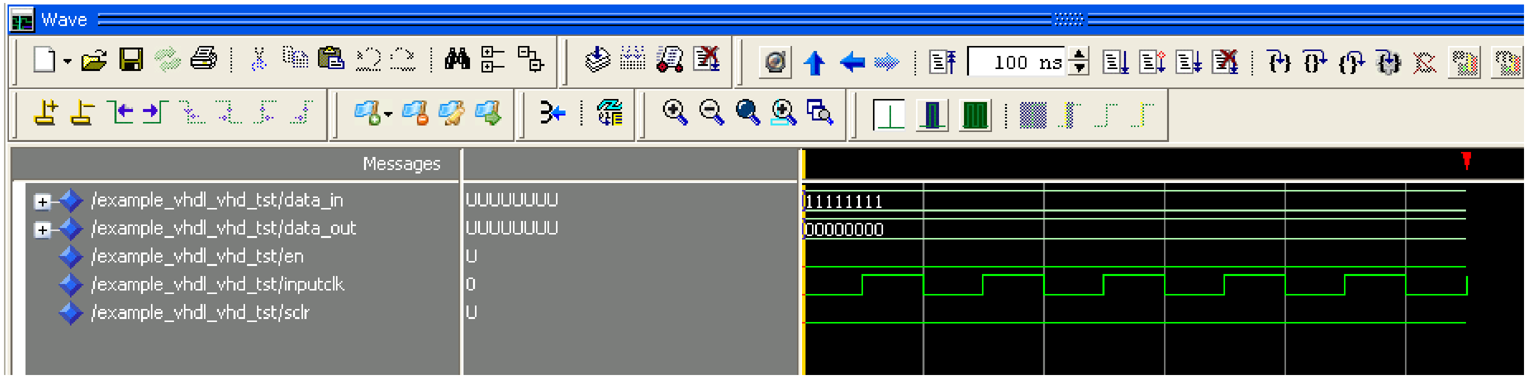
Iterating With Design Changes in Modelsim

- Results of “restart –f” and “do run.do”
- The simulation stops on the assert failure command

```
VSIM(paused)> restart -f
VSIM(paused)> do run.do
# Model Technology ModelSim SE vcom 6.5d Compiler 2009.11 Nov 18 2009
# -- Loading package standard
# -- Loading package std_logic_1164
# -- Compiling entity shift_reg
# -- Compiling architecture syn of shift_reg
# Model Technology ModelSim SE vcom 6.5d Compiler 2009.11 Nov 18 2009
# -- Loading package standard
# -- Loading package std_logic_1164
# -- Compiling entity example_vhdl
# -- Compiling architecture rtl of example_vhdl
# -- Loading entity shift_reg
# Model Technology ModelSim SE vcom 6.5d Compiler 2009.11 Nov 18 2009
# -- Loading package standard
# -- Loading package std_logic_1164
# -- Compiling entity example_vhdl_vhd_tst
# -- Compiling architecture example_vhdl_arch of example_vhdl_vhd_tst
# vsim -novopt work.example_vhdl_vhd_tst
# Loading std.standard
# Loading ieee.std_logic_1164(body)
# Refreshing C:\work\ref_mat\test_benches\vhdl_testbench\simulation\modelsim\work.example_vhdl_vhd_tst(example_vhdl_arch)
# Loading work.example_vhdl_vhd_tst(example_vhdl_arch)
# Refreshing C:\work\ref_mat\test_benches\vhdl_testbench\simulation\modelsim\work.example_vhdl(rtl)
# Loading work.example_vhdl(rtl)
# Refreshing C:\work\ref_mat\test_benches\vhdl_testbench\simulation\modelsim\work.shift_reg(syn)
# Loading work.shift_reg(syn)
# Loading altera_mf.altshift_taps(behavioural)
# ** Failure: ----- Not a Failure it is the END OF SIMULATION --------
#    Time: 550 ns  Iteration: 0  Process: /example_vhdl_vhd_tst/stimulus File: C:/work/ref_mat/test_benches/vhdl_testbench/simulation/modelsim/example_vhdl.vht
# Break in Process stimulus at C:/work/ref_mat/test_benches/vhdl_testbench/simulation/modelsim/example_vhdl.vht line 99
# Simulation Breakpoint: Break in Process stimulus at C:/work/ref_mat/test_benches/vhdl_testbench/simulation/modelsim/example_vhdl.vht line 99
# MACRO ./run.do PAUSED at line 6
```

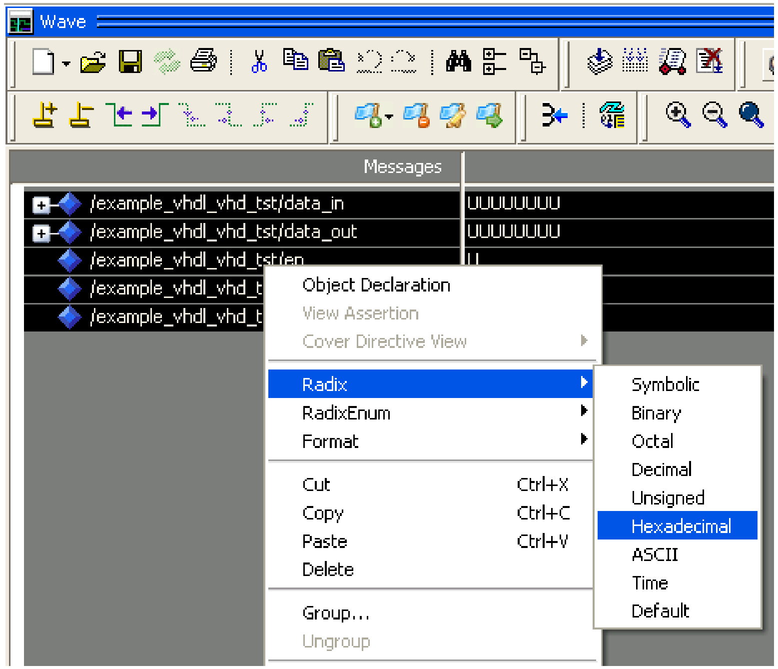
Simulation Results

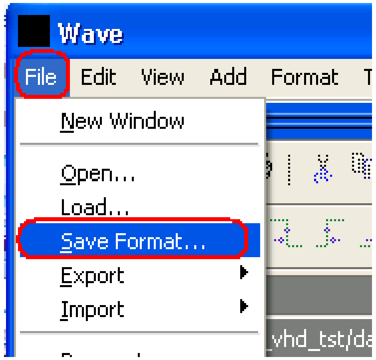
- Wave window results:
  - All of our inputs are now driven to a known value
  - Simulation ran until the 6th rising edge clock which is what we expected
  - The procedure call to p_sync_app waited until it sees a rising edge and then loops until it hits the 5th edge after.
  - Then we have the assert command to exit the simulator



Changing the Wave Window Radix

- The data is in binary, lets change to hex, highlight the signals, and right click and change to hex, then save the wave.do file (Save Format), so they will be hex from now on





More Testbench procedures

- Lets now add some more procedures to do some shifting in of data and control the clock enable.

```
-- procedure to test shifting with the enable
procedure p_shift is
begin
  wait for 50 ns;  -- good idea to drive the signals off of the rising edge, so wait 1/2 clock
  en <= '1';  -- need to enable the shift register
  data_in <= x"11";
  wait for (100 ns); -- basically advance the simulator 100 ns
  data_in <= x"22";
  wait for (100 ns);
  en <= '0'; -- turn off the enable
 end p_shift;
```

Shift in some data, hex '11' and hex '22'

```
-- procedure to test shifting with no enable, shift should not occur
procedure p_no_shift is
begin
  wait for 50 ns;  -- good idea to drive the signals off of the rising edge, so wait 1/2 clock
  en <= '0';  -- keep the enable off to make sure the data is not shifted through
  data_in <= x"33";
  wait for (100 ns); -- basically advance the simulator 100 ns
  data_in <= x"44";
  wait for (100 ns);
 end p_no_shift;
```

Make sure that when en is low, the data is not shifted in

```
 -- procedure to shift everything out and shift in 0's
 procedure p_shift_all_out is
 begin
     wait for 50 ns;    -- move simulation to the falling edge clock to keep off of rising edge
     en <= '1';         -- enable the shift register
     data_in <= x"00"; -- shift in 0's
     p_sync_app(20);    -- wait 20 clock cycles
 end p_shift_all_out;
```

We need to run through some clocks to shift the data out
It is 16 deep shift register…

Calling the Procedures in Testbench

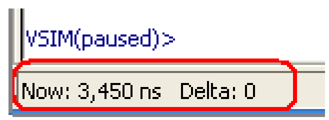
- Now add the procedure calls to the process

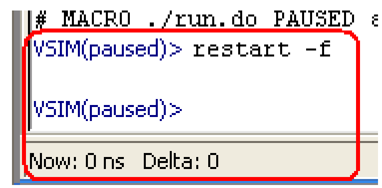
```
begin -- begin our stimulus process
  -- call p_stable once to get all your signals set at time 0
  p_stable;
  -- this will sync to the clock rising edge and then run 5 clock cycles before continuing
  p_sync_app(5);
  -- drive our data in for shifting
  p_shift; -- one line in are process will peform all the procesing in the p_shift procedure
  -- sync up to the rising edge
  p_sync_app(0);
  -- test no enable
  p_no_shift;
  -- sync up to the rising edge and then another 2 rising edges
  p_sync_app(2);
  -- shift everthing out
  p_shift_all_out;

  -- assert statements can control simuation, the default in modelsim is that severity failure
  -- will stop the simulation, this is a handy way to stop the simulation at the end of all your
  -- stimulus, this allows you to do a 'run -all' instead of a certain duration like 'run -10 us'
  assert false report "----- Not a Failure it is the END OF SIMULATION --------"
   severity failure;
```

Call our new procedures, shift in the data, test the no shift, and then Shift the data out…

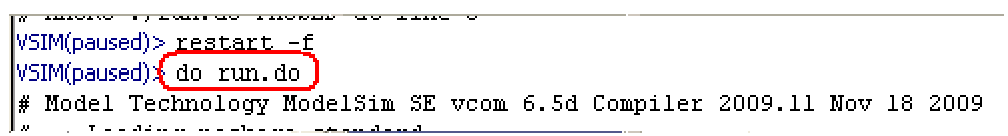
Re-running the Simulation

- We are now ready to re-run the simulation with out testbench changes, again we have our “run.do” script, recall we need to restart the simulator with “restart –f”
- If you look at the lower left portion of the simulator, you can see the Now: value, this is the current time of the simulator, when you do the restart –f, you will see it clear back to 0 ns.



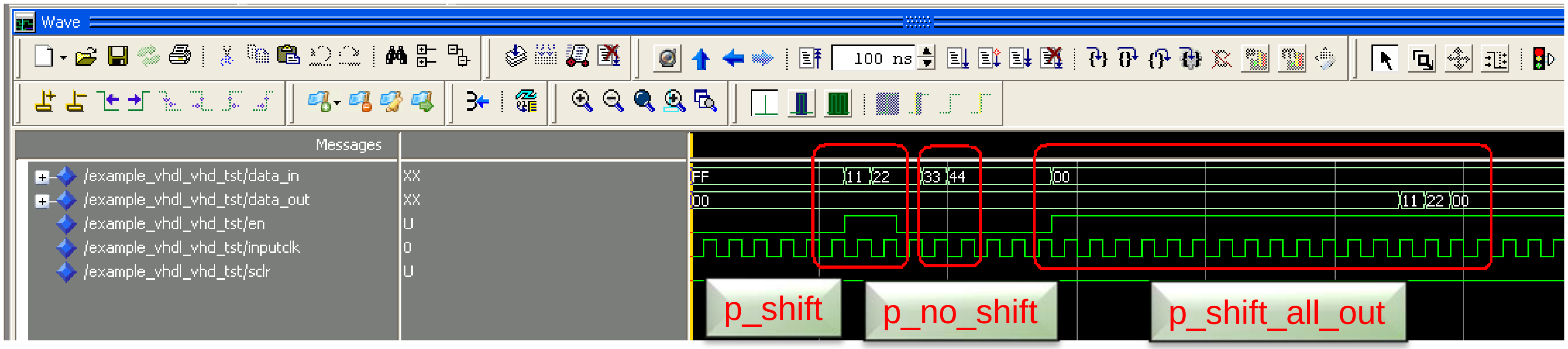


- Run the script
  - do run.do



Simulation Results

- Results of the simulation, you can see the results of calling the various procedures



Adding in Self Checking into testbench

- For self checking we will be adding:
  - A package to help convert std_logic_types for writing
  - Shared variables to allow passing of expected values
  - Signal to invoke the self check
  - Another process to perform the self check
  - Modifying current procedures to start the self check

Additional library Statements

- Added Library Statements and the image_pkg to the testbench file to help with converting std_logic_vectors to strings

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------------------------------
-- the following is added to support writing with write/writeline
-----------------------------------------------------------------
USE ieee.std_logic_textio.all;
USE std.textio.all;
-- package to help in using textio
USE work.image_pkg.all; -- compile image_pb.vhd into library work in modelsim

---------------------------------------------------------------------
-- top level entity, no sesitivity list, can add generics that can be
-- passed in at simulation time
---------------------------------------------------------------------
```

Libraries for writing out messages and Package to help with conversions

Additional Signal and Variable for Self-Checking

- Boolean added for invoking the checking process
- A Shared Variable, essentially a global variable, that can be passed throughout the design, not just within a process like a standard variable

```
SIGNAL data_out : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL en : STD_LOGIC;
SIGNAL Inputclk : STD_LOGIC := '0';  -- give the clock a default value
SIGNAL sclr : STD_LOGIC;

------------------------------------------
-- signals/variable used for self checking
------------------------------------------
-- boolean to invoke a self check
SIGNAL b_check : boolean := false;
-- essentially a global variable used to pass expected values aro
SHARED VARIABLE sv_check_value : std_logic_vector(7 downto 0);
-------------------------------------------------------
-- this is the actual fpga design component declaration
-------------------------------------------------------
```

b_check used for invoking the checking process
sv_check_value is used to hold the expected results

Example VHDL Testbench

- Added the setting of the shared variable and the boolean to the procedures

```
----------------------------------------------
-- procdure to test shifting with the enable
----------------------------------------------
procedure p_shift is
begin
  wait for 50 ns;  -- good idea to drive the signals off of the rising edge, so wait 1/2 clock
  en <= '1';  -- need to enable the shift register
  data_in <= x"11";
  wait for (100 ns); -- basically advance the simulator 100 ns
  data_in <= x"22";
  sv_check_value := X"22";  -- expected value
  b_check <= true;          -- enable a check
  wait for (100 ns);
  en <= '0'; -- turn off the emable
  b_check <= false;
 end p_shift;

------------------------------------------------------------------
-- procedure to test shifting with no enable, shift should not occur
------------------------------------------------------------------
procedure p_no_shift is
begin
  wait for 50 ns;  -- good idea to drive the signals off of the rising edge, so wait 1/2 clock
  en <= '0';  -- keep the enable off to make sure the data is not shifted through
  data_in <= x"33";
  wait for (100 ns); -- basically advance the simulator 100 ns
  data_in <= x"44";
  sv_check_value := X"44"; -- expected value
  b_check <= true;         -- enable a check
  wait for (100 ns);
  b_check <= false;
 end p_no_shift;
```

Set the shared variable, and set the boolean to true to Invoke the test, the check process will use the shared variable

Set the shared variable, and set the boolean to true to Invoke the test, the check process will use the shared variable

Testbench Monitor Process – Additions to the testbench file

```
-----------------------------------------
-- monitor for performing a check on data
-----------------------------------------
generic_monitor : process (b_check)
 variable l:line;
```

Monitor process, sensitive to b_check, need variable for outputting strings

```
begin
  if b_check = true then
   if data_out /= sv_check_value then
```

Is checking enabed, if so, does the data_out match the expected value?

```
    assert false
        report "data_out is NOT correct"
      severity warning;
--    severity failure;          -- if you want the simulation to stop
```

Can use assert statements to report problems

```
      writeline(output,l);            -- prints a blank line, since l is blank
      write(l, string'("****************************************************************"));
      writeline(output,l);
      write(l, string'("******   FAILURE FAILURE FAILURE FAILURE FAILURE FAILURE  ******"));
      writeline(output,l);
      write(l, string'("**************************************************
      writeline(output,l);
      write(l, string'                              simulation
      write(l, now);
      write(l, string'(", the data did NOT match!"));
      writeline(output,l);
      ----------------------------------------------------------------
      -- use the image package so that we can convert std_logic_vector
      -- to a string for writing out
      ----------------------------------------------------------------
      write(l, string'("Expected binary value:" & image(sv_check_value)));
      writeline(output,l);
      write(l, string'("  Actual binary value:" & image(data_out)));
      writeline(output,l);            -- print some blank lines for better formatting
      writeline(output,l);
      writeline(output,l);
```

The write and writeline write values out the modelsim console window, needs the textio packages and the image

Now is the current simulation time

Sending the check value and data_out value to the *image* function to convert to a string

```
   else               -- data out is correct
    assert false     -- data is correct
      report "data_out is correct"
    severity warning;
   end if;           -- data out check
  end if;            -- b_check
end process generic_monitor;
```

The data matched, just using assert to report this

Image Package: image_pb.vhd

- The image_pkg package is used to convert various types to strings for using write/writeline.
- The package contains an overloaded function called image, you can send in different types, and a string is returned.

```
package Image_Pkg is
   function Image(In_Image : Time) return String;
   function Image(In_Image : Bit) return String;
   function Image(In_Image : Bit_Vector) return String;
   function Image(In_Image : Integer) return String;
   function Image(In_Image : Real) return String;
   function Image(In_Image : Std_uLogic) return String;
   function Image(In_Image : Std_uLogic_Vector) return String;
   function Image(In_Image : Std_Logic_Vector) return String;
   function Image(In_Image : Signed) return String;
   function Image(In_Image : UnSigned) return String;

end Image_Pkg;
```

- Need to compile the image_pb.vhd file into library work, then add the vcom command to the run.do script
  - vcom -reportprogress 300 -work work C:/work/ref_mat/test_benches/vhdl_testbench/simulation/modelsim/image_pb.vhd
  - Add the above command before the testbench file compilation

New run.do file

- Below is the final run.do file, with relative paths added
- You can create multiple "*.do" files to break up compilation, and running, etc
  - For a small design, recompiling source files is quick, this can become lengthy on larger designs, and thus time consuming

C:/work/ref_mat/test_benches/vhdl_testbench/simulation/modelsim/run.do

```
vcom -reportprogress 300 -work work C:/work/ref_mat/test_benches/vhdl_testbench/shift_reg.vhd
vcom -reportprogress 300 -work work C:/work/ref_mat/test_benches/vhdl_testbench/example_vhdl.vhd
vcom -reportprogress 300 -work work C:/work/ref_mat/test_benches/vhdl_testbench/simulation/modelsim/image_pb.vhd
vcom -reportprogress 300 -work work C:/work/ref_mat/test_benches/vhdl_testbench/simulation/modelsim/example_vhdl
vsim -novopt work.example_vhdl_vhd_tst
do wave.do
run -all
```

C:/work/ref_mat/test_benches/vhdl_testbench/simulation/modelsim/run.do

```
vcom -reportprogress 300 -work work ../../shift_reg.vhd
vcom -reportprogress 300 -work work ../../example_vhdl.vhd
vcom -reportprogress 300 -work work ./image_pb.vhd
vcom -reportprogress 300 -work work ./example_vhdl.vht
vsim -novopt work.example_vhdl_vhd_tst
do wave.do
run -all
```

Results of the Testbench Monitor Process

- Results from running the run.do file “do run.do” from the modelsim command prompt

Monitor process was called twice with mismatched data, hence, two sets of failures

```vhdl
---------------------------------------------
-- monitor for performing a check on data
---------------------------------------------
generic_monitor : process (b_check)
 variable l:line;
begin
  if b_check = true then
   if data_out /= sv_check_value then
    assert false
        report "data_out is NOT correct"  -- not really needed just added for completeness
      severity warning;                   -- will not break out of simulation
--    severity failure;                   -- if you want the simulation to stop
      writeline(output,l);                -- prints a blank line, since l is blank
      write(l, string'("******************************************************************"));
      writeline(output,l);
      write(l, string'("******    FAILURE FAILURE FAILURE FAILURE FAILURE FAILURE   ******"));
      writeline(output,l);
      write(l, string'("******************************************************************"));
      writeline(output,l);
      write(l, string'("at time: "));
      write(l, now);                      -- now is the current simulation time
      write(l, string'(", the data did NOT match!"));
      writeline(output,l);
      ----------------------------------------------------------------
      -- use the image package so that we can convert std_logic_vector
      -- to a string for writing out
      ----------------------------------------------------------------
      write(l, string'("Expected binary value:" & image(sv_check_value)));
      writeline(output,l);
      write(l, string'("  Actual binary value:" & image(data_out)));
      writeline(output,l);                -- print some blank lines for better formatting
      writeline(output,l);
      writeline(output,l);
   else              -- data out is correct
    assert false  -- data is correct
      report "data_out is correct"
    severity warning;
   end if;           -- data out check
  end if;            -- b_check
end process generic_monitor;
```

Testbench File

```
#
# ****************************************************************
# ******    FAILURE FAILURE FAILURE FAILURE FAILURE FAILURE   ******
# ****************************************************************
# at time: 700 ns, the data did NOT match!
# Expected binary value:00100010
#   Actual binary value:00000000
#
#
# ** Warning: data_out is NOT correct
#    Time: 1 us  Iteration: 1  Instance: /example_vhdl_vhd_tst
#
# ****************************************************************
# ******    FAILURE FAILURE FAILURE FAILURE FAILURE FAILURE   ******
# ****************************************************************
# at time: 1000 ns, the data did NOT match!
# Expected binary value:01000100
#   Actual binary value:00000000
#
#
# ** Failure: ----- Not a Failure it is the END OF SIMULATION --------
#    Time: 3450 ns  Iteration: 0  Process: /example_vhdl_vhd_tst/stimulus
# Break in Process stimulus at C:/work/ref_mat/test_benches/vhdl_testbencl
# Simulation Breakpoint: Break in Process stimulus at C:/work/ref_mat/tes
# MACRO ./run.do PAUSED at line 7

VSIM(paused)>
```

Modelsim Output

Conclusion

- Simple way to create a testbench structure from Quartus
- Create one process and add procedures to perform the stimulus to the FPGA under test
- Built up a “do” file to easily iterate through design and test bench changes “run.do”
- Created a process to perform self checking of the results
- Wrote outputs to the modelsim console window
- There are many ways to perform tesbenches, this is one example