

Post-Synthesis Validation

ECE 551: Digital System Design and Synthesis
Fall 2021, Hoffman

Overview

You have already finished your behavior or RTL level Verilog design, and synthesized it using design_compiler mapping it to the Synopsys **saed32lvt_tt0p85v25c** technology library. It is time to simulate your synthesized design and compare the result with your original design. This tutorial demonstrates how to do post-synthesis simulation in ModelSim with your synthesized design.

Extract Synthesized Netlist

Assuming your synthesis was done with a script based method in dc_shell. Then the final line of your synthesis script should be to write out the gate level verilog netlist.

```
write -format verilog <top_module> -output <top_module>.vg.
```

For instance if your top level module was **uart** this would be:

```
write -format verilog uart -output uart.vg
```

Post-Synthesis Functional Simulation

You should put the post synthesis verilog file into a new directory and create a new project for your simulation. You can copy the original testbench to use.

Timescale

Synopsys cells use ``timescale 1ns/1ps` compiler directive. Therefore, it is a good design practice to add the ``timescale 1ns/1ps` compiler directive in **both** your synthesized netlist (uart.vg in our example) and testbench (uart_tb.v in our example). Typically this is done as the first line in the Verilog file.

```
-----  
`timescale 1ns/1ps  
uart_tb();  
.  
.  
.  
-----
```

Testbench

Besides modifying ``timescale` compiler directive, your original testbench might need a fix of your reset timing.

- **RESET**

If you use only positive clock edge in your design, you might think that you can reset all the flip-flops as long as the reset pulse covers a positive clock edge. This is not always true, because in LSI Logic cells, both positive and negative clock edge timing constraints are given. Your RESET pulse should cover both a positive and a negative clock edge to properly reset

your circuit. Since we are using many async reset flops, it is good practice to deassert reset on the negative edge of clock. Below is an example of an initial block that works well.

```
-----
initial begin
    clk = 0;           // clock must be initialized
    rst_n = 0;         // assert reset
    ...               // initialize other inputs to your DUT
    @(posedge clk);    // wait for a positive edge of clock
    @(negedge clk);    // wait for negative edge of clock
    rst_n = 1;         // deassert reset
    .
    .
    .
-----
```

- **Initial Assignment to all inputs of DUT from testbench**

Sometimes if an input from the testbench is X it can propagate in an unexpected manner through the gates. Ensure all input to your DUT from the testbench have a logical 0 or 1 assignment prior to reset deasserting.

Cell Library (very important)

The synthesized netlist you wish to simulate is made up of a bunch of library cells from the Synopsys library. How does ModelSim know the function of a **NAND2X1_LVT** gate? We need to reference a pre-compiled library of all the cells in the Synopsys library (**saed32lvt_tt0p85v25c**) that our synthesis run targeted.

It is best if you make a local copy of the library so you have full linux file permissions on the compiled library you will reference. You can copy the library from the linux user ejhoffman.

Copy the **SAED32_lib** somewhere in your home directory structure. Perhaps at your **<root>/ece551** level

```
unix_prompt>cd ~/ece551
unix_prompt>cp -r ~ejhoffman/ece551/public/SAED32_lib .
```

Search Library (very important)

When you launch simulation in ModelSim (Simulate → Start Simulation). You have to add the precompiled **SAED32_lib** library you just copied to the search path. The Simulate → Start Simulation form has many tabs across the top. One of them is the “Libraries Tab”.

NOTE: The **SAED32_lib** was compiled on linux. This will only work with vsim on linux. Do not try to run this using ModelSim using CAE Windows machines.

select **Simulate**-> **Simulate** and select **Library** tab
 click **Add** under the Search Libraries section
 Now click **Browse** and a “Select Library Browser” form comes up.
 In this form type in:
 ~/ece551 in the Library path search string at the bottom of the form, and hit enter.
 Now select the **SAED32_lib** library you copied.

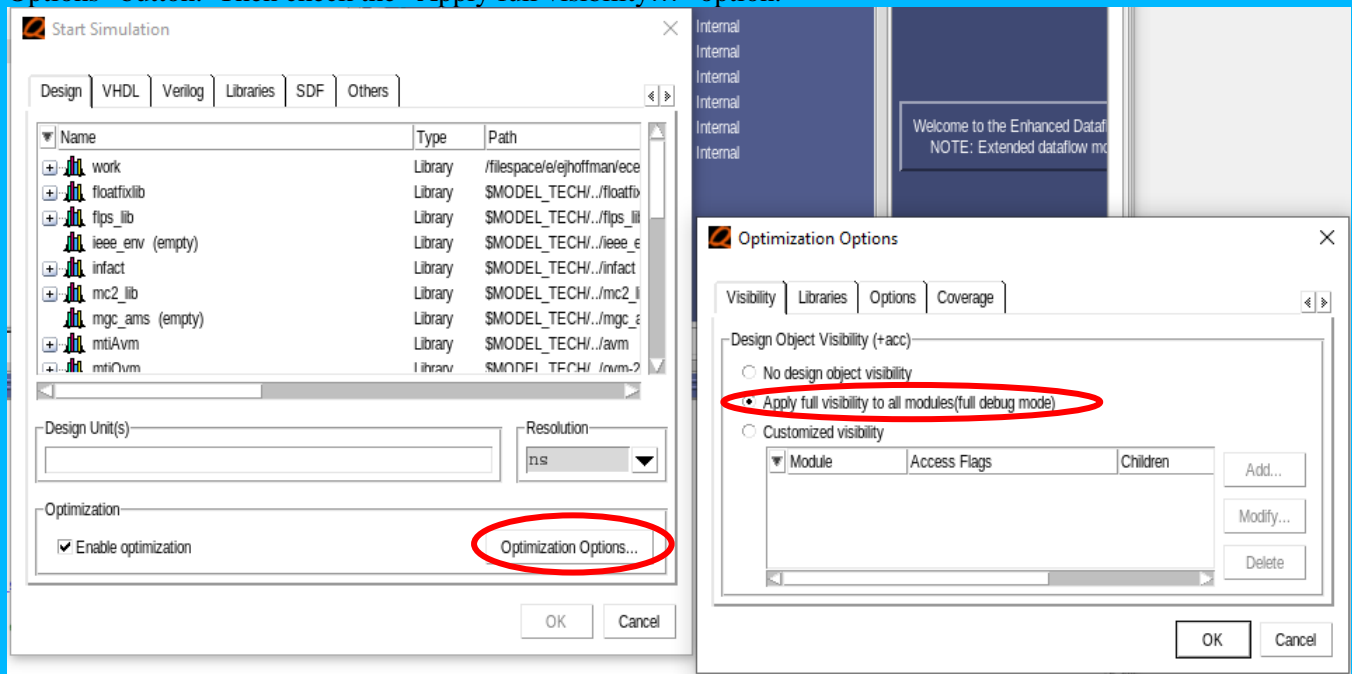
Disable Timing (under the Verilog Tab of Start Simulation) (also very important)

Synopsys is the final arbiter of setup and hold time violations, not the timing checks in the library files. We want to disable the timing checks in the library files so they don't cause problems.

Under the “Verilog” tab of Start Simulation, **check the box that says:**
“Disable timing checks in specify blocks(+notimingchecks)”

Select **Design** tab and select your module (testbench) to simulate.
 Change the **Resolution** dropdown box to ns.

Enable optimization should be checked on Linux vsim version, but you should take the “Optimization Options” button. Then check the “Apply full visibility...” option.



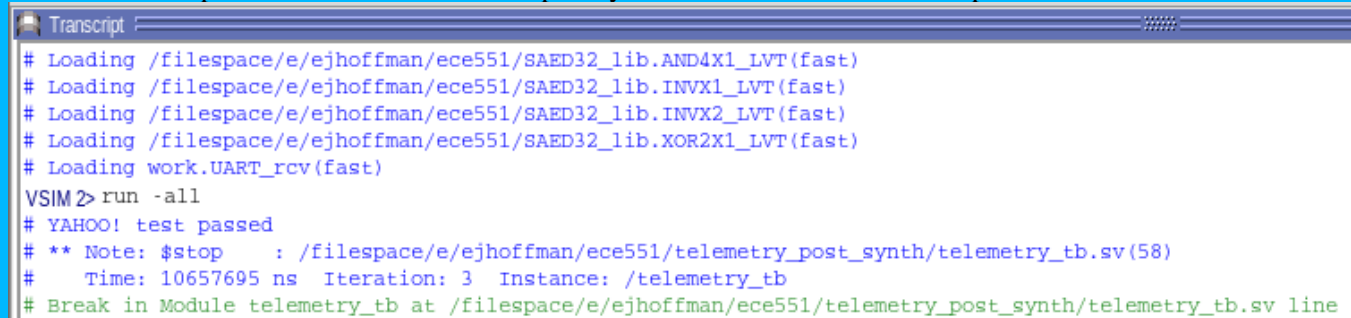
Click **OK** on the Optimization options form and then on the Start Simulation form.

NOTE: on linux we need enable optimizations checked. However, under the optimizations options button choose “full visibility”.

The design should now load using cell modules out of the **saed32lvt_tt0p85v25c** library. You may receive many timescale directive warnings, but these can be ignored. You may also receive warnings like: “missing connections for port Q”. The library contains flops with both Q and QN outputs, and sometimes Synopsys only uses one of the connections, so these are expected warnings.

Now run hit the “run all” button and run the simulation as you normally would. It will take longer because post synthesis simulation is slow (it is simulating all those gates, not a behavioral design).

Finally get a screen grab of the transcript window. Make the transcript window large enough in the vertical such that I can see it loading the cell library cells, and the “Yahoo” test passed messages from the self check(s) of your testbench. This proves to me it was indeed a post synthesis simulation. See example below:



```
Transcript
# Loading /fileSPACE/e/ejhoffman/ece551/SAED32_lib.AND4X1_LVT(fast)
# Loading /fileSPACE/e/ejhoffman/ece551/SAED32_lib.INVX1_LVT(fast)
# Loading /fileSPACE/e/ejhoffman/ece551/SAED32_lib.INVX2_LVT(fast)
# Loading /fileSPACE/e/ejhoffman/ece551/SAED32_lib.XOR2X1_LVT(fast)
# Loading work.UART_rcv(fast)
VSIM 2> run -all
# YAHOO! test passed
# ** Note: $stop      : /fileSPACE/e/ejhoffman/ece551/telemetry_post_synth/telemetry_tb.sv(58)
#      Time: 10657695 ns  Iteration: 3   Instance: /telemetry_tb
# Break in Module telemetry_tb at /fileSPACE/e/ejhoffman/ece551/telemetry_post_synth/telemetry_tb.sv line
```

Summary

1. Extract synthesized netlist
2. Make necessary changes
 - a. Timescale
 - b. Testbench
 - c. Copy the SAED32_lib from ~ece551 account
 - d. Search library → Reference the SAED32_lib you copied
 - e. Disable Timing Checks
3. Simulate Design

If your design does not work with post-synthesis simulation:

1. Are there latches in the design?
 - a. Run “**report_register –level_sensitive**” inside design_vision. If this report has no results, you have no latches (good).
2. Is every flip flop being reset?
 - a. Start looking at main important registers such as a state register or counters used. All flops that are involved in control functions will need a asynch reset from **rst_n**. If not X’s will propagate.
 - b. If something is not being reset, try using **force...release** statements inside your testbench to force the register to a known value so you can see if other parts work
3. Does your testbench change input signals concurrent with the clock edge?
 - a. You might be creating a race. Using @ (posedge clk); statements to wait to change stimulus is fine because the flops have already been evaluated by the time the change occurs. However if you are using delays (#clock_period strt_tx = 1;) you might be changing input stimulus right at the clock edge and creating race conditions. Remember flops need hold time, so input stimulus should transition slightly after clock edges.
4. Did you assign a value to all input stimulus in your testbench at time zero? Remember your stimulus signals in your testbench are of type reg, and therefore assigned X at time zero. Behavioral simulation is more forgiving about how X’s propagate. In post synthesis simulation X’s propagate in nasty ways. Assign all input stimulus to a known value.
5. Are there other warnings from synthesis about unused connections, wire connected to GROUND/VCC?
 - a. These are an indication that your design may not be connected as you thought

6. Is the clock speed of your testbench faster than the clock speed your design could achieve?