EE/CprE 465 -- Fall 2015

Lab 4: Synthesis & Static Timing Analysis

What is synthesis?

In the last 2 labs, we built our design in Cadence Virtuoso analog design environment and simulated it with Cadence Spectre engine. You must have spent a lot of time to build the schematic and layout for the DFF. It will cost more time if we have a huge design to make. For example, if we need to implement our circuit of Lab 1 which contains many DFFs, INVs, NANDs, adders, etc., we may spend days to build the schematic, not to mention the time for layout drawing and simulation. To reduce the design time, the solution is circuit synthesis: translation of your Verilog codes to gate-level circuit by software tool automatically.

How does synthesis work?

In synthesis, the synthesis tool will create the schematic for us based on what we wrote in our Verilog codes. That means the tool will translate our Verilog codes into schematics automatically. We called this kind of schematic netlist. Netlist contains the information of which logic gates are used and how they are connected. Synthesis is based on a standard cell library. It can only use the logic gates that exist in the library. For example, if there is no NAND gate in the library but there are AND gates and inverters, the tool will use one AND gate and one inverter wherever it needs a NAND gate to build the schematic. You have already had an overview of standard cell library in the last two labs. In a library, all integrated cells' properties (such as timing, power, and area information) have been characterized into lookup tables. So the tool knows every cell's characteristics. The tool is like a construction worker who knows all kinds of bricks that he has very well. He will use different kind of bricks to build different parts of the building depends on optimization objectives and constraints.

Constraints for synthesis

Does the synthesis tool translate our Verilog codes into gate level circuit based on logic only? No. Typically, the tool first translates the Verilog codes into a generic netlist without considering the standard cell library. Next the tool performs technology mapping, which maps the modules in the generic netlist into the cells in the libray. This mapped netlist may not be good in terms of various design objectives like area and power, and may not satisfy various constraints like setup time and hold time constraints of flip-flops. (Sometimes, power is specified as a constraint.) So the tool will apply different optimization techniques like gate sizing, buffering, cloning, pinswapping and restructuring to optimize the netlist. In other words, how the constraints are set will affect the result of synthesis. We will explore this issue in next lab.

Power estimation

There are two kinds of power information to be reported by the tool after synthesis. One is dynamic power and the other is leakage power. However, the tool just reports estimated values to both of them. The reason is that it is impossible for the tool to calculate the powers exactly without knowing the inputs. For example, the dynamic power when the circuit is idle and when it is fully running will not be the same. Similarly, for the leakage power, it will not be the same when a 2-input NAND has inputs of logic 11 and when it has logic 10. But at the synthesis stage, the tool does not know the input values of the circuit. So the tool can only estimate the dynamic and leakage powers based on some assumptions on the typical behaviors of the circuit.

For every standard cell in the library, there are dynamic power information (in power lookup tables, just like timing lookup tables we learned) and leakage power information (please search for them using the keyword "leakage") in different situations.

Static Timing Analysis

Do you remember the simulation we did in Lab 1? That is called functional simulation. Functional simulation only considers the *function* of the codes and has no timing involved. Then we perform synthesis, which turns the design into a more detailed, gate level netlist. Should we do functional simulation again to make sure the function is correct at this point? The answer is "not necessary". It is because synthesis should generate a netlist which is functionally equivalent to the Verilog codes. Although the netlist should be functionally correct, it may have timing issues.

During synthesis, as mentioned above, the tool will try to construct the circuit considering area, power, setup time and hold time constraints, etc. But the timing information during synthesis stage is still very crude. It is possible that the tool has done its best but the timing still fails. How can we make sure the circuit will not violate the setup time and hold time constraints and will work correctly when it is made into real silicon? We need to check the timing based on more accurate analysis. After synthesis, we have the gate level description of the circuit. We also have delays for every gate, which are characterized into lookup tables in the library. We may determine the timing of the circuit by simulation, but simulation is very time consuming and is input-dependent. In practice, Static Timing Analysis (STA) is performed instead. The word “static” alludes to the fact that this timing analysis is carried out in an input-independent manner. STA purports to find the worst-case delay of the circuit over all possible input combinations to ensure setup time and hold time constraints are never violated. (Note that it is impossible to obtain accurate timing until after layout, which will be studied in Lab 6.)

3 corners for a library

Because of variations in process, voltage and temperature (PVT), a fabricated cell may operate faster or slower than normal. In order to ensure that a circuit functions correctly under all circumstances, it is necessary to analyze a circuit under various extreme process, voltage and temperature conditions, i.e., corners. As a result, a standard cell library needs to be characterized for multiple corners. Regarding process, two letters are usually used to specify how the speed of transistors is affected due to process variation. The first letter is about the NMOS transistors and the second letter is about the PMOS transistors. For each letter, F means “fast”, T means “typical” and S means “slow”. For example, FS means NMOS transistors are fast and PMOS transistors are slow in fabricated chips. Typically, for each library in a given process, there are at least 3 characterizations associated with 3 different corners. We call them best, worst, and typical corners.

- Best corner assumes lower working temperature, higher working voltage and favorable process parameters (i.e., FF process corner), so all cells will run faster in such condition.
- Worst corner assumes higher working temperature and lower working voltage and unfavorable process parameters (i.e., SS process corner), so all cells will run slower in such condition.
- Typical corner specifies typical process parameters (i.e., TT process corner) and operating condition inbetween the last 2 cases.

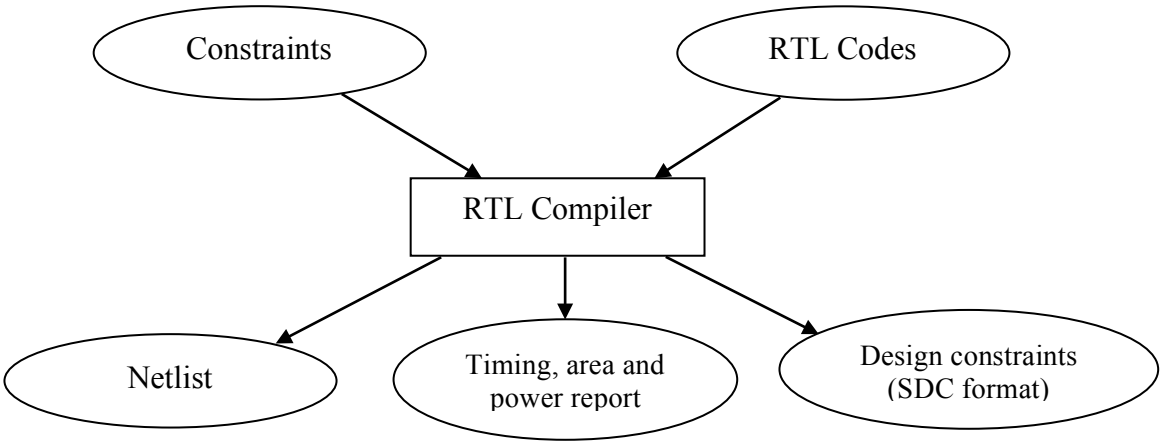
Typical corner is used when the designers want to get some idea on how the circuit would perform under normal condition. In theory, to check if the circuit has any timing violation, we should run synthesis based on both best and worst corners. But in practice, we just focus on fixing setup time violations during synthesis and leave hold time violations to next stage -- placement and routing. So at synthesis stage in this lab, we only use the worst corner library.

Lab Tasks:

The objective of this lab is to get a hands-on experience with:

1) RTL to netlist synthesis
2) Circuit timing, area, power analysis

In this lab, you will synthesize the Verilog code you designed in the Lab 1 with the TSMC 65nm standard cell library posted in Lab 3 using Cadence RTL Compiler. Please refer to the document RTL_compiler_tutorial.docx on how to use RTL compiler. The inputs and outputs of RTL Compiler are described in the diagram below.



1. Synthesize the Verilog codes of Lab 1 using the attached scripts in rc.rar. (Please note that you may need to modify some parts of the scripts if you used different port names from Lab 1 instruction.) The clock period constraint is already set to 0.8ns by the following command in the design.sdc file:

   ```
   create_clock -period 0.8 -waveform {0 0.4} [get_ports {iCLK}]
   ```

   where {0 0.4} means the duty cycle is 50% according to the clock period of 0.8ns.

2. Exam the schematic you get after synthesis. Please make sure you have those DFFs at the input ports and output ports just like the diagram in the Lab 1 instruction. If there is not, find out why the synthesis tool gives you such a synthesized netlist without those DFFs based on your Verilog codes. You will get to know that your Verilog coding style will affect the synthesis result.

3. Afterwards, look at the timing report either by the method shown in the tutorial or by typing "report timing" in the command window. This command will show you the details of the worst timing path in your design based on the clock period you set. A document How_to_read_timing_report.pdf is posted. It describes how to read a timing report in great detail. Although the document is about timing reports in Synopsys PrimeTime format generated by the Altera HardCopy Design Center, the concepts are the same. You should find this document very useful. Please answer the following questions:

   - What kind of timing violation checking will be carried out by the command "report timing"? Setup time or hold time? How do you know that?
   - Please exam the worst path diagram (if you use the method shown in the tutorial to report timing) or the timing report (if you used "report timing" in the command window). Describe where the start point of this path is and where the end point is. Please roughly mark this path in the diagram of Lab 1. In this way, you may get a sense of which part of your circuit has the tightest timing.
   - According to the timing report, what is the slack value of the worst path? Is it a positive or negative value? What does it mean? Do you think this path has timing violation or not?

4. We have automatically stored the report of worst timing path to a file named timing.rpt in the folder run_dir by running the scripts. Please copy it to another file as you will need it later. There is one command "dc::set_dont_use FA1D*" in the design.sdc file. It is a constraint that prevents the tool to use FA1D* cells to synthesize the circuit. FA1D* stands for a series of cells named FA1D1, FA1D2, etc. Please open the file DBTCBN65GPLUSBC_121B.pdf (TSMC 65nm Core Library Databook) from Lab 2. It contains the general information of all standard cells, including the functional description. Please find the FA1D* cells in the databook and explain what it is. Now remove this constraint by just deleting it. Re-run synthesis and generate a new timing report. Please note we have not changed the clock period yet. It is still 0.8ns.

   By comparing the details of this new timing report to the one last time, you should not find a cell named FA1D1 in the last report, but there is such a cell in the report this time because we allow the tool to use this kind of cell during synthesis. Does the slack get improved or not comparing to that of the last time of synthesis? Does it still have violation? If the answer is yes, one possible (but perhaps undesirable) solution is to relax the clock period you set in the constraint file. Based on what you have learnt about setup time and hold time, can you estimate a minimal amount of relaxation for the period to eliminate the violation according to the current values of clock period and slack? Then execute it to verify your estimation. Please re-synthesize until there is no timing violation in the report. Please report the slack and clock period values. In this way, you can get the highest frequency your circuit can run with.

5. Now you have two versions of synthesis. One allows the tool to use FA1D* cells and the other does not. For synthesis without using FA1D* cells, please also modify the clock period and re-run synthesis until there is no timing violation in the report. In this way, you can get the highest clock frequency for both versions. Now, we compare the timing reports of these two versions. You should see that one version can give us a faster circuit than the other. But timing is not everything. Now report the area and power consumption of these two versions respectively. Compare area and power consumption of these two synthesis results. You should find out that the version with better timing has worse power consumption and uses more area. For the other version, it may have better power consumption and uses less area, but has worse timing. You have already experienced this kind of trade-off in Lab 3 when you tried to optimize your DFF. Can you explain why this FA1D1 cell can bring such a difference in the synthesis result? And why would the tool try to use it if it is allowed?