

§ 10 Static Timing Analysis and Clock Tree

§ 10.1 Static Timing Analysis

Comparison with Functional Simulations

Unlike functional simulation, static timing analysis (STA) analyzes the logic in a static manner, computing the delay times for each path through the logic. The path with the longest delay is called the critical path.

A timing analysis tool is more a logic calculator than a logic simulator. STA is necessary because of the limitations of logic simulators (including gate level simulations). For example, most logic simulators calculate logic cell delays once, before the simulations start. However, logic cell delays and signal ramp times often change depending on the state of certain control inputs. Such state-dependent timing is not accounted for by most logic simulators, and STA is needed for a complete timing analysis of the circuit.

Fortunately, STA does not require the creation of stimulus patterns, as does functional simulation. Note, also, that STA will not provide any stimulus patterns to activate the critical path. STA does not consider the function of the logic during analysis.

It logically follows that functional simulation may not find the critical path delay. Further, the critical path may be impossible to activate during functional simulation. The question must then be asked whether or not the delay on the critical path matters. If it does not, then the path is called a false path.

Dealing with Asynchronous Paths

STA works best when analyzing a fully synchronous system, where the longest delay path is the longest path through logic between FFs. In the real world (and in the design project), this is often not the case. Accordingly, logic with multiple clocks inherently have false paths from an STA perspective. These false paths are logic paths that exist between clock domains (at the asynchronous interface) where the delay on the path is irrelevant from an STA perspective.

To deal with this during STA, as well as during the generating of timing report using Synopsys Design Time, the command `set_false_path` is used.

```
set-false-path [-from from-list] [-through through-list] [-to to-list]
```

With respect to the design project, the set-false-path command would be used as follows.

```
set-false-path -from find(clock wclk) -to find(clock rclk)
```

```
set-false-path -from find(clock rclk) -to find(clock wclk)
```

If there are other logic paths that fail STA, the determination of whether or not they are false paths is on a case-by-case basis. By default, all logic paths in synchronous logic are assumed to be "true" paths.

Pre-Layout vs. Post-layout STA

STA at the IP block level is typically only done pre-layout (not post-layout) using an appropriate wire load model and enough timing margin to ensure that the likelihood of a post-layout timing violation is quite low. Over-constraining the clock by 20% is one example of such timing margin. Another is using a fairly pessimistic clock-uncertainty value that is expected to be higher than what is achievable in layout, worst case.

However, at the device level, both pre-layout and post-layout STA is necessary. What is typically done is pre-layout STA scripts are developed, using pre-layout timing information in standard delay files (SDFs), in such a manner that it's straightforward to switch-in a post-layout SDF and re-run the STA scripts. This approach is taken because post-layout timing information is not usually available until quite late in the design cycle, and we don't want to delay tapeout because of the unavailability of STA scripts.

Pre-layout STA at the device level continues to be done using overly pessimistic clock-uncertainty, worst case operating conditions (PVT), etc. Also, clock tree latency is modeled during pre-layout STA.

One commonly available STA tool is PrimeTime by Synopsys. PrimeTime STA scripts have a syntax very similar to the synthesis scripts you have created using Design Compiler. We won't be developing PrimeTime STA scripts in this course. Rather, we will stick to pre-layout timing reports generated by Design Time using wire load models to model interconnect delay, worst case operating conditions, and standard cell delays available from the 0.18 micron technology library.

Once pre-layout STA scripts are available, and as layout progresses, a post-layout SDF is the first timing information that is available for post-layout STA. You'll recall that SDF models interconnect delay using a "lumped" RC model. Thus, a post-layout SDF is useful for a first cut STA of the device. Note that this post-layout SDF has timing information for nets targeted for CTS -like clocks, resets, scan enables, etc.

Using this post-layout SDF with clock tree latency timing information included, STA will be able to show early signs of areas of the design that may fail to meet timing. Areas like:

- .Chip level setup, hold time violations
- .Chip level min/max propagation delay violations
- .Clock skew issues
- .Internal logic with timing problems, that may result from poor logic cell placement in layout, long interconnect nets with high RC values, poor placement of memories relative to the logic interfacing with the memory, etc. Think Physical

At this point, designers feed back the results of preliminary post-layout STA to layout, and the layout of the device is changed accordingly to improve the logic timing. In extreme cases, the circuit is re-synthesized with different synthesis constraints and a modified circuit is given to layout (one that is expected to have better timing).

As layout progresses with cell placement, routing interconnect, routing power, placing the device I/O pads, synthesizing clock trees, etc, a more accurate post-layout timing file is made available. It is called a SPF file and is different than a SDF in that the interconnect delays are "segmented", not "lumped" as in the SDF. Such segmented interconnect delays make the SPF file much larger than the SDF file, and it accordingly takes more CPU time and memory to perform STA using a SPF. However, SPF is much more accurate than SDF -in fact, SPF is accurate to within +/-5% of a SPICE transistor level analysis of the circuit, as compared with the +/-15% of SPICE transistor level accuracy of a SDF. For this reason, post-layout STA using SPF is done with relaxed clock period and clock skew, and must be exhaustively done on the design prior to tapeout.

At this late stage, if there are any timing violations, it usually makes more sense to manually edit the layout rather than to re-synthesize the netlist and start the process over again. Time is money, and getting to tapeout as soon as possible is obviously desirable.

The effect of process. voltage. temperature (PVT)

So far, all the STA I have been discussing is done using worst case operating conditions (worst case process, lowest voltage, highest temperature). This is because most timing violations result from a logic path that is too slow, not too fast. However, prior to tapeout, STA must also be performed using best case operating conditions, where things like hold times and minimum propagation delays may become an issue.

To perform STA using best case operating conditions, the same SDF and SPF files discussed above are used. This is because SDF and SPF files contain timing information for worst case, best case and typical case operating conditions. The designer performing the STA simply switches libraries to best case, and the associated operating conditions to best case. The STA scripts are re-run to ensure that no timing violations exist.

Typical operating conditions are not usually used for STA. It is deemed sufficient if STA successfully passes for both worst case and best case operating conditions.

§ 10.2 Clock Tree Delay -how it affects setup/hold/propagation delay

Consider the following circuit...

and associated timing diagram for a selected input of a device.

The higher the clock tree (CT) latency, the more this helps us meet any setup time requirement for the design. However, the more it hurts us meet any hold time requirement for the design. As a result, the higher the CT latency, the more buffering that may be required between the device input and the D-input of the FF to satisfy any hold time requirement.

As an example, let's assume the following delays:

Input pad	1 ns
Clock pad	1 ns
Logic Delay	1 ns
CT Latency	3 ns
FF setup	0.1 ns
FF hold	0 ns
Maximum Input to Clock setup specification	3 ns
Minimum Input to Clock hold specification	0 ns

Question - do we meet the setup time requirement? The hold time requirement? If not, how much buffering is required between the device input and the D-input of the FF?

Now consider the following circuit.

and associated timing diagram for a selected output of a device.

The higher the CT latency, the higher the minimum and maximum clock-to-output-valid propagation delay. As the CT latency increases (the larger the ASIC, the more FFs there are, and the higher the CT latency will be), the maximum propagation delay will be so large that it will be greater than a clock period. What do we do when the CT latency is so large? We clock the FF driving the device output using a clock edge that is earlier in time.

Using Delay-Locked-Loops (DLL) -why and how

We could do this by tapping off the clock to this FF at an earlier stage in the CT. But a more controlled method of doing this is to use a delay locked loop (DLL). A DLL is simply a piece of digital logic that allows clock edges to be moved ahead or behind in time based on certain control inputs. In this example, we wish to move the clock edge that clocks the device output FF ahead in time.

Such a DLL typically goes through layout as a block by itself. This is to ensure that the timing inside is very deterministic, post-layout. As you can imagine, the granularity of clock-adjustment increments must be quite granular and deterministic for the DLL to work reliably.

As an example, let's assume the following delays:

Output pad	1 ns	
Clock pad	1 ns	
Logic Delay	1 ns	
CT Latency	3 ns	
FF CP->Q delay	0.5 ns	
Minimum Clock to Output valid specification		0 ns
Maximum Clock to Output valid specification		3 ns

Question -do we meet the minimum clock-to-output-valid requirement? The maximum clock-to-output-valid requirement? If not, by how much time must the clock feeding the FF that clocks the device output be moved ahead in time?

The Effort of Output Loading

As you can imagine, the larger the capacitive load on the outputs of a device, the higher the clock-to-output-valid time becomes, and the problem discussed above worsens.

Generally speaking, for large ASICs with deep clock trees (and correspondingly large latencies), and/or for devices with large capacitive loads on output pins, and/or for devices with very aggressive clock-to-output-valid specifications, the requirement for a DLL becomes increasingly likely. As a data point for a device in 0.18 micron, if the ASIC is larger than 5 million gates with a couple of megabits of memory, and/or if the load on output pins is > 50 pF, and/or if the clock-to-output-valid is < 10 ns (this is equivalent to saying if the clock frequency is greater than something in the 50 to 100 MHz range), then the ASIC may require a DLL.

Pipelining to Fix Timing Problems - a last resort

Truly a last resort at this late stage of the development, but sometimes it is necessary to add a stage or pipelining to fix a static timing violation. This often happens when trying to squeeze too much logic into a single clock cycle, or trying to meet too aggressive I/O timing constraints without using a DLL. Also, a reminder that registering all of the primary I/O to both an IP block and at the device level, as well as registering memory I/O, all help avoid timing problems from cropping up late in the development cycle. Any timing violation discovered late in the development cycle (i.e. close to tapeout) that requires going back and re-synthesizing the netlist or re-architecting the VHDL code is truly disastrous, as the time required to repeat the steps required to implement the fix usually delay tapeout by several weeks.