Chapter 14
Timing Closure

Srinivasan Dasasathyan

14.1 Introduction to Timing Concepts

Timing closure involves modifying constraints, design, or tool flow/settings to meet timing requirements. In Vivado tool, the timing constraints are entered in *XDC* format. XDC constraints are based on the standard Synopsys Design Constraints (SDC) format.

For brevity all the constraints that Vivado supports are not explained in this chapter but only few are given to help understand topics discussed later in this chapter. For details on XDC constraints and syntax, please refer to UG903 published by Xilinx.

14.1.1 Creating and Defining a Clock

create_clock Tcl command allows user to define clock on a certain port and also allows users to specify properties like period, waveform, root, etc. Unless a clock is defined using the *create_clock* command, static timing analysis is not performed on the clock. Also, *create_clock* command defines primary clocks, and all *derived* clocks are automatically inferred. Usually the *derived* clocks come from the clock modifying blocks like MMCM and PLL.

S. Dasasathyan (✉)
Xilinx Inc., San Jose, CA
e-mail: srini.das@gmail.com

© Springer International Publishing Switzerland 2017
S. Churiwala (ed.), *Designing with Xilinx® FPGAs*,
DOI 10.1007/978-3-319-42438-5_14

14.1.2 Defining Clock Relationships

Like all other SDC-based tools, Vivado also does timing analysis on all the cross-clock paths. However, designers in certain occasions would want to ignore certain paths, because those paths are either static paths (no signal transition happens) or the paths are asynchronous and hence should not be timed. In such cases *set_clock_groups* or *set_false_path* commands are used to preclude certain portions of the designs from timing analysis. This is an essential step as ISE (the previous Xilinx tool) which used UCF constraints, assumed the opposite, i.e., unless clock relationship was specified, timing analysis was not done on cross-clock paths.

14.1.3 Timing Analysis

Given these basic definitions of creating clock constraints and specifying clock relationships, Vivado's timing analysis engine does several checks under the static timing analysis engine. The timing analysis engine analyzes and reports slack at the timing path endpoints. The slack is the difference between the data required time and the data arrival time at the path endpoint. A data is safely transferred between two registers if both the setup and hold relationships are successfully verified on that path. In other words, if both setup and hold slacks are positive, the path is considered good from a timing point of view. The following are the checks performed by Vivado's timing analysis engine:

- Setup check
- Hold check
- Pulse-width check

14.2 Generating Timing Reports

The first step in timing closure is to understand whether the design has met all the timing checks or not. In order to generate timing reports to view failing paths, the following options are available in Vivado.

14.2.1 Report Timing Summary

Report timing summary gives an overall picture of timing on the design. It performs *setup*, *hold*, *pulse-width* checks, and gives a summary on whether some or all of these checks have failed. Even if one of the checks has failed, this command reports that the design has failed to meet timing. Based on this report, it can be decided if further steps are needed to achieve timing closure. Figure 14.1 gives a sample snapshot of the command, where *setup*, *hold*, and *pulse-width* violations are checked.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-1.120 ns	Worst Hold Slack (WHS):	0.016 ns	Worst Pulse Width Slack (WPWS):	0.381 ns
Total Negative Slack (TNS):	-6027.573 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	34121	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	490437	Total Number of Endpoints:	490437	Total Number of Endpoints:	154291

Fig. 14.1 Report timing summary output

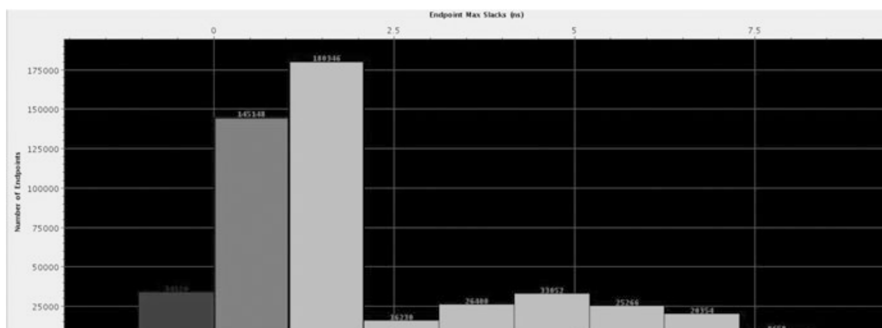

Fig. 14.2 Slack histogram

Once the design is determined to have not met timing requirements, you can further analyze failing timing paths in the design by running report timing or slack histogram command.

14.2.2 Report Timing

Report timing summary only gives a top-level report on timing failures; however, *report timing* gives details of all the paths that fail timing checks (setup and hold). By default *report timing* reports on all path groups and prints the top 10 paths in each path group and sort it by slack in ascending order. Additional filters can be added to customize timing analysis on different *from*, *through*, or *to* points as well as select more paths to view. *Report timing* only works for *setup* and *hold* checks. *Pulse-width* checks are reported in Vivado log file indicating where the errors are.

14.2.3 Slack Histogram

Another way to see the failing timing paths is to generate *slack histogram*. *Slack histogram* gives a concise view of all the timing paths across all path groups. Figure 14.2 shows a sample slack histogram plot. Slack histogram divides the slacks into different bins. The *X*-axis represents different slack bins and the *Y*-axis represents the number of paths in each bin. Clicking on each of the bars filters the paths in that bin, where you can examine paths in each of the bin.

In both report timing and slack histogram, you can click and double-click any of the paths to examine each of the timing path in detail, including characteristics of the path as well as placement and connectivity details.

14.3 Timing Paths and Constraint Correctness

Timing paths are defined by the connectivity between the instances of the design. In digital designs, timing paths are formed by a pair of sequential elements controlled by the same clock or by two different clocks.

In order to debug and fix the timing paths, it is important to first check whether these paths are valid or not. Checking constraints is one of the key and easy steps in getting to timing closure. One of the common issues in writing of XDC constraint is related to incorrect cross-clock domain crossing paths. Timer takes the worst case requirements for timing analysis. Hence if cross-clock paths are getting wrongly timed (very often they needn't be timed), they might have very tough requirement, resulting in a big negative slack. *Report CDC* and *report clock interaction* are two very useful commands to check if the interclock paths are being timed correctly.

14.3.1 Clock Interaction

Report clock interaction gives a matrix and specifies where all the clock pairs in the design are considered for interaction. Each entry in the matrix is color coded. All the entries across the diagonal are the paths within the same clock group. It is important to examine if there are any unexpected cross-clock domain paths, and fix them by adding proper XDC constraints (*set_false_path*, *set_clock_groups*). Xilinx published UG903 has more details.

14.3.2 Report Clock Domain Crossing

Report CDC (clock domain crossing) performs a structural analysis of the clock domain crossings in your design. You can use this information to identify potentially unsafe CDCs, which will lead to metastability or data coherency issues. While the CDC report is similar to the clock interaction report, the CDC report focuses on structures and their timing constraints, but does not provide information related to timing slack.

Before generating the CDC report, you must ensure that the design has been properly constrained and there are no missing clock definitions. *Report CDC* only analyzes and reports paths where both source and destination clocks have been defined. *Report CDC* performs structural analysis on:

1. On all paths between asynchronous clocks
2. Only on paths between synchronous clocks that have the timing exceptions (e.g., clocks coming out of MMCM)

Synchronous clock paths with no such timing exception are assumed to be safely timed and are not analyzed by the CDC engine. The report CDC operates without taking into consideration any net or cell delays.

14.4 Timing Closure Techniques

14.4.1 Critical Path Analysis

Timing reports can be generated at any stage during the synthesis and/or implementation phase. You should generate timing reports at each stage after synthesis, placement, and routing and analyze the paths to make sure that the design is converging. Catching and fixing issues earlier in the flow will save several iterations of the subsequent stages. For example, fixing issues at synthesis will save time in place and route stage.

A timing failure might happen due to multiple different reasons. Based on the analysis of the timing paths, fixes may be required at synthesis stage or the placement and routing stage. Hence it is important to study the characteristic of top failing paths to determine the reasons and fixes. Below are some of the important characteristics in the timing paths that can be examined and remedies that can be taken to mitigate them.

14.4.2 Logic vs. Wire Delay

Critical path delay can be broken down into logic delay and wire delay. The percentage of logic and wire delay in critical path can help to determine where to reduce delays. A low logic delay component usually means that wire delay is higher, where potentially floor planning the design can help in timing closure. A higher logic delay component means that there are too many logic levels in the design.

14.4.3 Reducing Logic Levels

For paths with higher levels of logic, looking at the levels of logic in the top failing paths can reveal if there are any issues in the RTL or inferring of the logic.

Synthesis step in Vivado infers structures in optimal way to balance between area and speed. Different RTL coding styles guide the tool to infer structures that are sometimes area optimal or performance optimal. By observing the logic levels in

critical path, we can identify if we need to change either RTL coding style or guide the tool to infer for performance as opposed to area. To reduce the levels of logic, you can return to the RTL and check for the following general issues. In addition, refer to Chap. 9 for controlling synthesis behavior.

- Use *FSM_ENCODING* in your RTL to infer ONE_HOT FSM, which are usually better for speed.
- Use *CASE* statements instead of nested *IF-ELSE* statements; though the former takes more area, it has efficient inferences of Muxes which leads to better delays.
- Add pipeline registers to the critical path.

Any change to RTL will require resynthesizing the design. Several iterations may be needed to get optimal depth of logic.

14.4.4 Clock Skew

Clock skew is the difference between delays that clock takes from *common source* to capture flop/sequential element and the launch flop/sequential element. Examining the magnitude of clock skew can reveal issues in clocking structure. A design with high clock skew in critical paths usually means that the clocking structure needs to be revisited. Using MMCMs to multiply/divide clocks is recommended than using LUTs. UltraScale and newer devices have a very flexible clock architecture and offer lots of clocks to the user. To ease the issue of reducing clock skew and to generate *H-tree* clocking structures, the device offers *CLOCK_ROOT* which is the center tap points from where clock distribution happens. *CLOCK_ROOT* is chosen by Vivado for set of clock loads such that clock skew for the set of loads is minimal. However, in some cases where the paths are legal cross-clock domain paths, clock skew might be higher. In these cases user can choose *CLOCK_ROOT* manually to reduce the clock skew. UG912 from Xilinx explains the mechanism to modify *CLOCK_ROOT* location.

14.4.5 Reducing High-Fanout Signals

High-fanout signals typically pose a challenge to the place and route tools, as due to the very nature they have many connections, and the placement will be spread out. Due to this, delay on the net would be relatively higher. If the top several critical paths have some commonality that all of them involve high-fanout signal, some optimization can be done at RTL level to reduce the fanout coupled with options to synthesis tool. Some options are:

Duplicate the driver and tell the synthesis tool not to remove the duplicate logic (attribute *DONT_TOUCH*).

For the signals other than control signals such as reset, set, and clock enable, using *max_fanout* in synthesis will direct synthesis to replicate the driver.

Another option is to use *phys_opt_design* (post-placement). This command performs timing-based logic replication of high-fanout drivers and critical-path cells. Drivers are replicated, then loads are distributed among the replicated drivers, and the replicated drivers are automatically placed. This optional command can be run after placement and before routing.

14.4.6 Control Sets and Control Set Optimization

In Xilinx FPGA architecture (for 7 series and UltraScale), each *slice* has eight flip-flops (FFs). These eight FFs share control signals, so the FFs that are placed in the same slice should have same control sets. Hence the flops in the same slice have to share the control set. Placer algorithm honors this constraint by placing FFs of the same control sets together. Xilinx FPGAs can accommodate several thousand control sets; however, the higher the number of control sets, the more complex the job for placer to place flops into slices without wasting flops. *report_control_sets* command can be used to assess the number of unique control sets in the design. Under verbose options, the command gives details on the distribution of the fanouts of the control signal.

Vivado synthesis has an option which is used to specify threshold for synchronous control set optimization to lower number of control sets. The number set to this value specifies how large the fanout of a control set should be before it starts using it as a control set. For example, if *control_set_opt_threshold* is set to 5, a synchronous reset that only fans out to 5 registers would be moved to the *D* input's logic rather than using the reset line of a register. The default threshold value is currently set to 4.

Other ways to reduce control sets is to use *resets* judiciously. Be selective on the use of *resets* by observing the following points:

- Have resets only where they have impact on functionality.
- Use synchronous resets rather than asynchronous reset.

14.4.7 Floor Planning

Examining the critical path in the Vivado GUI will show the placement of the logic in the path. Sometimes, placer while trying to optimize several constraints might yield a suboptimal placement. Examining the top several critical paths in the GUI will give an idea if the placer indeed did a suboptimal job in placement of critical-path object. If so, floor planning can be done to guide the placer. A hierarchical floor plan can reduce the route delay in the critical logic. A good starting point when floor planning for the first time is to floor plan only the logic that the implementation tools consider timing critical. Generally start with the lower-level hierarchies that the place and route stage finds to be timing critical. More often it is useful to look at

the placement of block RAMs and DSP blocks, as these are not distributed throughout the FPGA. Floor planning them not only gives better performance but also predictive results in future iterations of the same project. When the design meets timing, it is also possible to reuse the placement.

For SSI devices, floor planning poses additional requirements to consider, which are explained in Chap. 13.

14.4.8 Physical Optimization

Physical optimization performs optimization on the paths that fail to meet timing. Optimizations involve replication, retiming, hold fixing, and placement improvement. Physical optimization is usually run after placement when the timing picture is reasonably accurate. These optimizations are invoked by explicitly running the optional *phys_opt_design* command. This command performs the following physical optimizations.

High-Fanout Optimization: High-fanout nets, with negative slack within a percentage of the WNS, are considered for replication. The drivers are replicated and the replicated drivers are placed near to cluster of loads.

Placement-Based Optimization: Cells on the critical path are replaced to reduce wire delays.

Rewire: LUT connections are swapped to reduce the number of logic levels for critical signals. LUT equations are modified to maintain design functionality.

Critical-Cell Optimization: Cells in failing paths are replicated. If the loads on a specific cell are placed far apart, the cell may be replicated with new drivers placed closer to load clusters. High fanout is not a requirement for this optimization to occur, but the path must fail timing with slack within a percentage of the worst negative slack.

DSP Register Optimization: Registers are moved out of the DSP cell into the logic array or from logic to DSP cells if it improves the delay on the critical path.

Block RAM Register Optimization: Registers are moved out of the block RAM cell into the logic array or from logic to block RAM cells if it improves the delay on the critical path.

Retiming: Registers are moved across combinational logic to provide better timing.

Forced Net Replication: Net drivers are replicated, regardless of timing slack. Replication is based on load placements and requires manual analysis to determine if replication is sufficient. If further replication is required, nets can be replicated repeatedly by successive commands. Although timing is ignored, the net must be in a timing-constrained path to trigger the replication.

The above optimizations are run only during post-placement physical optimization steps; however, Vivado also allows to run physical optimization at post-route stage also. Only a subset of the optimizations are run at post-route stage, as the runtime of physical optimization post-routing is higher.

14.4.9 Strategy and Directives

Directives are powerful features that are available with every implementation step (synthesis, optimize design, placement, physical optimization, and routing). Directives give the implementation step to direct behavior of the algorithms toward alternate goal. It changes the implementation step by using:

- Different flows
- Different algorithms
- Different objectives

Directives allow each implementation step to enable more design space exploration than in the default mode. Directives have different objectives such as *reduce area*, *reduce runtime*, *improve performance*, *and improve power.*

Directives are enabled by running any synthesis and implementation step with the option *-directive*. Usually the names of the directive are chosen to indicate how different they are compared to the default behavior and their objective. Every implementation step has the directive *explore*. Explore allows the implementation step to work in a high effort mode to meet the timing objective at the expense of runtime. For designs with very tight requirements, it is recommended to use *explore* directive for most of the implementation steps (especially placement and physical optimization). Directives related to placement usually give the biggest improvement for performance. Please refer to UG904 from Xilinx for details on the list of directives and what each of the directive's objectives is.

Strategies define the flow of Vivado and customize the different implementaiton steps, and how each of these steps are configured. As each synthesis and implementation step has varieties of options and directives, strategies configure the best possible combination of these switches. You can also define your own custom strategy. Strategies are categorized into the following:

- Performance
- Area
- Power
- Flow
- Congestion

Each of the above strategy categories has several strategies which can be used to extract the last mile performance from the tools. In the context of timing closure, categories related to performance and congestion are applicable. One way is to run all the available performance strategies and pick the best results.

14.4.10 Congestion and Congestion Alleviation

FPGA routing architecture has different kinds of routing resources to service different scenarios seen in placement of the design. Congestion can happen when in a region there is more demand of certain kinds or all kinds of routing resources than

their availability. Extent of the congestion regions defines whether the congestion is local or global. Router and placement algorithms, in order to alleviate congestion, introduce *white spaces* and *detours*. These changes may impact the routing delays by worsening them, which impact the timing of the design. There are certain steps you can take to reduce the effect of congestion on timing. Congested regions can be determined by running congestion reporting using *report design analysis*. Also designs with heavy utilization of block RAMs, MuxF7s, and MuxF8s and distributed RAMs have a tendency to have congestion. Care should be taken to reduce the utilization of any block with high connectivity. Blocks with high connectivity increase number of signals coming in a region where the blocks are placed. If there are many high connectivity blocks placed in a small region, one can increase the size of a region by defining a *pblock*. The size of the pblock can be increased to make it large enough to have enough routing resources to complete routing all nets and thereby alleviating congestion.

14.4.11 Report Design Analysis

Report design analysis is a command that summarizes several important details on the critical paths. Commonly occurring issues in critical paths are summarized in a tabular format. By looking at the characteristics of several critical paths, issues can be deduced. *Report design analysis* has three modes of operation:

- Timing
- Congestion
- Complexity

Timing mode is used to find out the characteristics of critical paths. For each of the path, many important characterisitcs are printed. For example, it is easy to determine if the top critical paths have block RAMs and whether they are registered or not. Or, if the top several critical paths have LUTs which are combined in synthesis stage (we can turn this off by using *-lc off* option). Xilinx published UG906 provides information on other meaningful information that can be obtained from this report.

Congestion mode gives the post-placement and post-routing congestion windows, and *complexity* computes the *rent's* exponent of the netlist or modules specified. Congestion combined with complexity can determine whether the netlist itself is inherently congested, or the congestion is placement induced. Using congestion mode, you can find the congested window and also determine what modules are placed in the region. Later you can run complexity on these modules and compute the *rent's* complexity on them. Rule of thumb says that any rent's complexity over *0.7* can be considered as an issue in netlist.

14.4.12 Timing Closure and Hold Violation

The previous section covered several techniques related to closure of timing which mainly focused on setup violations. Hold violations are also another kind of timing failures that you need to be aware of. Hold violations are severe, as reducing the clock frequency will not help in timing closure. Vivado tool is hold aware and tries to mitigate the violations by detouring and adding extra delay to the paths failing *hold*. However, you should be aware of these requirements and not solely depend on tool to fix the issues. Buffers can be added in hold failing path with *DONT_TOUCH* attribute so that synthesis tool does not optimize them away. Further post-route physical optimization and few router directives can also help to reduce the hold violation. Figure 14.3 provides a top-level flow chart for achieving timing closure on your design.

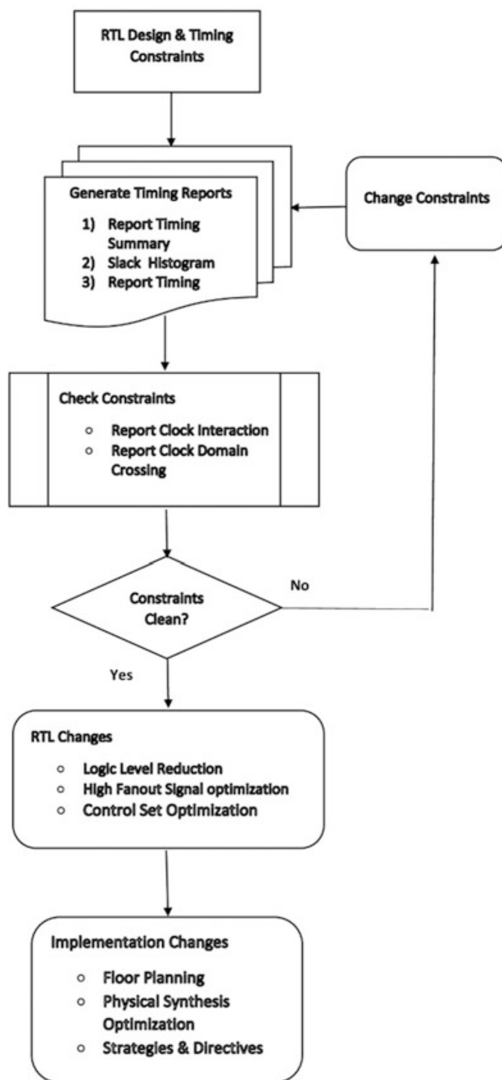

Fig. 14.3 Flow chart for timing closure