Vivado Design Analysis

The most crucial tips for Vivado FPGA circuit design

Gathered by:
Morteza Salimi Moghaddam

Samim Engineering Group

Website https://samimgroup.com

Vivado Design Analysis

General concept of timing

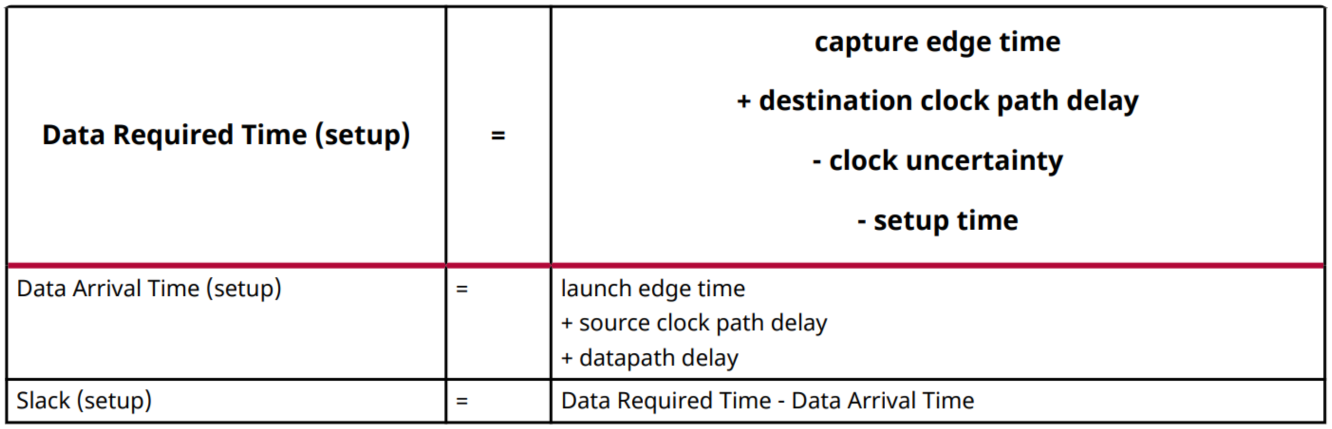
Understanding the fundamentals of timing analysis is our first step towards achieving timing closure in a design.
Timing paths are defined by the connectivity between the instances of the design. In digital designs, timing paths are formed by a pair of sequential elements controlled by the same clock, or by two different clocks. Each timing path is composed of three sections; source and destination clock paths along with data path. Data paths have startpoints and endpoints, both of which are supposed to be **clock pins or data ports.**
When transferring between sequential cells or ports, the data is:

- Launched by one of the edges of the source clock, which is called the launch edge.
- Captured by one of the edges of the destination clock, which is called the capture edge.

A few key concepts are to be followed:

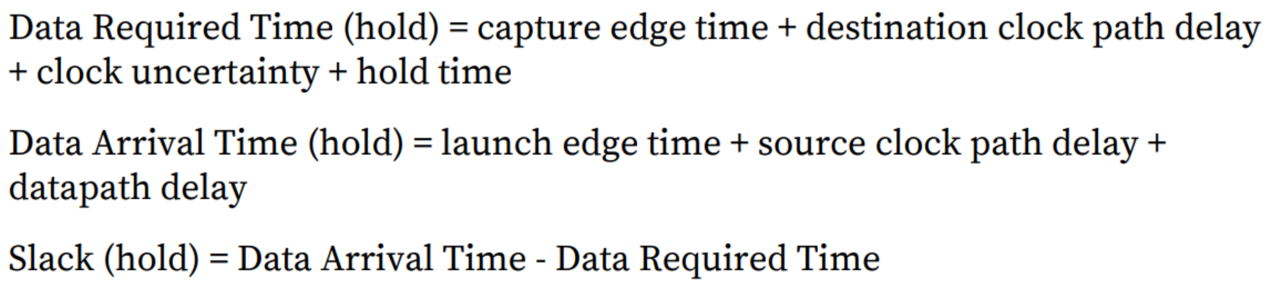
- Max Delay Analysis with Setup and Recovery Checks
- Min Delay Analysis with Hold and Removal Checks

We perform these analyses assuming the most pessimistic scenarios so that the result is guaranteed.



Data Required Time (setup)	**=**	**capture edge time** **+ destination clock path delay** **- clock uncertainty** **- setup time**
Data Arrival Time (setup)	=	launch edge time + source clock path delay + datapath delay
Slack (setup)	=	Data Required Time - Data Arrival Time

This section tells us what parameters are important when trying to reduce slack time out of setup duration. Bear in mind that negative slack time is not acceptable and means there's a problem regarding your hardware's capability to deliver results at the right time which leads to either metastability or incorrect outputs.

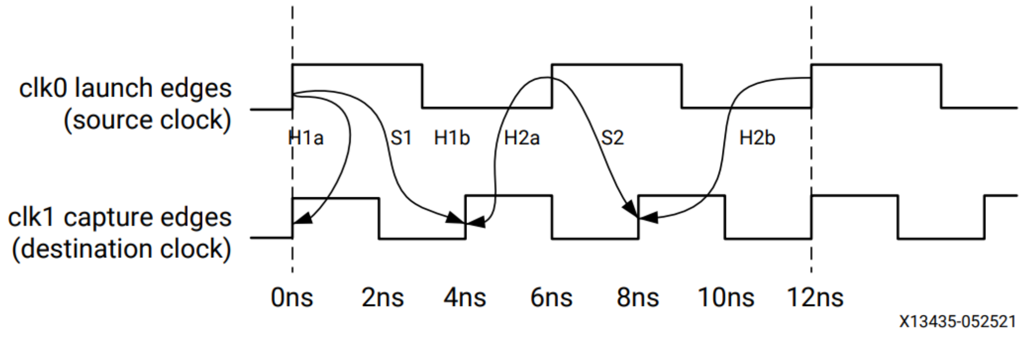


Data Required Time (hold) = capture edge time + destination clock path delay + clock uncertainty + hold time

Data Arrival Time (hold) = launch edge time + source clock path delay + datapath delay

Slack (hold) = Data Arrival Time - Data Required Time

The reason behind the differences between slack time calculations in hold and setup modes is that clock uncertainty can make clock arrival change both ways in time, so for the sake of having considered worst-case scenario, there's a difference in arithmetic sign of clock uncertainty in the two scenarios.



From the figure above, it's easy to infer the fact that worst-case analysis is found where the clocks make the closest value to zero for hold time, whereas the same parameter should be analyzed for hold time where it takes up time the most.
If a common period for the source and destination clocks is not found within 1000 cycles, the clocks are **unexpandable** and it's unlikely for the design to correspond to the most pessimistic scenario. We must review the paths between these clocks to assess their validity and determine if they can be treated as asynchronous paths instead.

Timing constraints

Timing constraints are a fundamental part of the design process, guiding the synthesis, placement, and routing stages to meet the desired performance goals. The key concepts, just to recapitulate, are setup and hold time, min/max delay constrains and clock and clock period constraints. Note that constraints can be applied to specific paths in the design, helping to focus optimization efforts on critical paths.
Timing constraints are important in many ways, but there are three factors that matter to us the most:

- Performance: Properly defined timing constraints allow designers to optimize the performance of their designs by ensuring signals meet required timing criteria.
- Reliability: Violations of timing constraints can lead to glitches, signal integrity issues, and even functional failures in the circuit. So taking them into consideration is a rather wise strategy.
- Cross-Clock Domain Synchronization: Constraints help manage interactions between signals belonging to different clock domains,

preventing metastability issues. Hence, CDC synchronization can be thought of as the most crucial factor of these three.

Another important idea to bear in mind when designing FPGA circuitry is **multicycle path**.
Multicycle paths are used to relax timing constraints on certain paths in a design. They are often employed in scenarios where meeting timing requirements on every clock cycle is unnecessary or overly restrictive.
Multicycle paths can apply to either setup time or hold time constraints. A setup multicycle path means that data can be captured by a flip-flop later than the normal setup time, and a hold multicycle path means that data can be released from the flip-flop earlier than the normal hold time.
The multicycle value specifies the number of clock cycles by which the path's timing constraints are relaxed. For example, a multicycle value of "2" would mean that the data must meet timing requirements every two clock cycles instead of every cycle. When performing timing analysis, the tools take into account the multicycle paths and ensure that the specified timing requirements are met according to the relaxed constraints.

Single and multiple bit transmission through CDC

Handling single or multiple-bit transmission across clock domains (CDCs) involves careful consideration of synchronization, timing constraints, and potential metastability issues. There are quite a few steps to transmit bits from a source with a specific clock frequency to a destination with another clock frequency:

1. Identify the signals that need to cross clock domains. Determine the source and destination clock domains, clock frequencies, and any clock relationships
2. Go through the synchronization process. This is where we have two different perspectives for single and multiple bit scenarios: If a single bit is in hand and frequencies of source is lower than the frequency of destination by a factor of at least 1.5, use a two-stage synchronizer (double flip-flop) to capture signals in the destination clock domain. For lower frequency differences and other scenarios of single-bit transmission, handshaking method works almost perfectly, the only problem with handshaking is that the acknowledgements are sent through the same bus as data which causes a tiny decrease in speed. This method works on multiple bit

transmission as well. There's another way to approach multiple bit transmission through CDC which is even better than handshaking, and that is using the asynchronous FIFO RAMs. The term "asynchronous" is used because the speed with which the RAM is reading differs from the speed with which it's writing.
3. Apply setup and hold timing constraints on the source and destination flip-flops to ensure reliable data capture and propagation.
4. For specific paths where the timing relationship is more complex, consider using the set_multicycle_path command to relax timing requirements.
5. If you're certain that a path is asynchronous or doesn't need to meet timing, use the set_false_path command to exclude it from timing analysis.
6. Simulate the design with different clock frequencies to verify proper synchronization and data transfer between clock domains. Observe the behavior of multiple-bit transmissions to detect any issues.
7. Use the FPGA design tool's timing analysis features to verify that the CDC paths meet timing constraints and avoid critical violations.
8. Use wider data paths for critical signals to enable error detection or correction capabilities.

More important SDC commands to handle CDC

1. set_clock_groups: This command defines clock groups to specify relationships between different clocks. It helps the tool understand which clocks are synchronous and which are asynchronous. To define synchronous clock groups:
2. set_false_path: As stated before, use this command to specify that a particular data path does not need to meet timing requirements. It's commonly used for asynchronous paths.
3. set_multicycle_path: This command, as stated before, can be used to define multicycle paths between asynchronous clock domains. It specifies that the data does not need to meet timing on every clock cycle. For an asynchronous path with setup relaxation.
4. set_clock_latency: Use this command to specify the latency of a clock relative to another clock in a different domain. This helps account for clock domain crossing delays.
5. set_input_delay / set_output_delay: These commands apply input and output delay constraints across clock domains, helping to align data correctly.
6. set_data_path_group: This command groups related data paths and allows you to define specific constraints for paths within the same group.

7. set_clock_skew: This command is used to specify the desired skew between two clock signals. Skew is often used to account for clock distribution delays, ensuring that different parts of the design have consistent timing relationships. It can be used to fix hold violations or setup violations in a design by adjusting the relative timing of different clock domains. It's crucial to use skew values that are within the limitations of the hardware. Excessive skew can lead to timing violations or unreliable operation.