Chapter 5
Performance (Timing) Analysis

Abstract Timing analysis can be static or dynamic. Dynamic timing analysis (DTA) verifies functionality of the design by applying input vectors and checking for correct output vectors whereas static timing analysis (STA) checks static delay requirements of the circuit without any input or output vectors. In this chapter, STA techniques is focused since it is widely used in FPGA design flow to make sure the timing requirements are met.

5.1 Overview

Dynamic timing analysis (DTA), also known as simulation-based timing analysis technique, is complicated for even small FPGAs because of huge number of input vectors and unbearable long simulation time, while static timing analysis (STA), which could analyze a design in a very short time, is then thriving. As a mainstay of modern FPGA design flows, STA breaks a design down into timing paths, calculates the signal propagation delay along each path, and checks for violations of timing constraints inside the design and at the input/output interface. STA also has been integrated with timing-driven EDA engines to optimize FPGA's timing performance.

The target design checkpoint (containing timing constraints and timing graph) and device library (containing timing models) are the main inputs that a timing analysis engine needs. The final output is the timing report (Fig. 5.1).

Standard Delay Format (SDF) is another optional output of timing engine. SDF is an IEEE standard for the representation and interpretation of timing data (both cell delays and interconnect delays) for use at any stage of the electronic design process [1]. This can be used along with the netlist in a simulator to verify that design meets its functional and timing requirements.

Given that device timing model has been discussed in Sect. 2.2.2, timing constraints has been discussed in Sect. 3.2.2, another main input–timing graph, derived from target design checkpoint, will be introduced in the following section.

Before we dive into the timing calculation algorithms, here are some basic concepts about STA. Figure 5.2 is the most common used picture to illustrate this.

K. Tu et al., *FPGA EDA*, https://doi.org/10.1007/978-981-99-7755-0_5

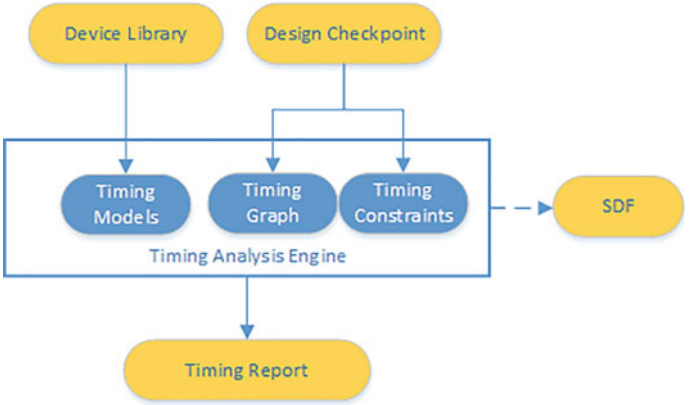

Fig. 5.1 Typical inputs and outputs in timing analysis flow

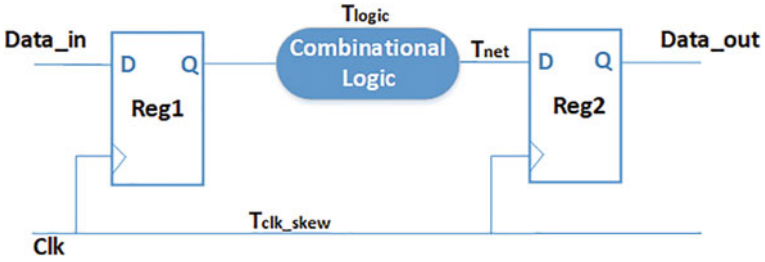

Fig. 5.2 Typical setup/hold timing analysis

Equations 5.1 and 5.2 can accurately represents the calculations of setup time slack ($\text{Slack}_{\text{setup}}$) and hold time slack ($\text{Slack}_{\text{hold}}$).

$$\text{Slack}_{\text{setup}} = T_{\text{period}} - (T_{\text{cq}} + T_{\text{logic}} + T_{\text{net}} + T_{\text{setup}} - T_{\text{clk}_{\text{skew}}}) \tag{5.1}$$

$$\text{Slack}_{\text{hold}} = T_{\text{cq}} + T_{\text{logic}} + T_{\text{net}} - T_{\text{hold}} - T_{\text{clk}_{\text{skew}}}) \tag{5.2}$$

where T_{period} is clock period, T_{cq} is defined as time it takes for data to appear on output Q once clock is triggered (pos edge or neg edge), T_{logic} is the delay of the combinational logic, T_{net} is the delay of the routing net, $T_{\text{clk}_{\text{skew}}}$ is the time difference between the clock arriving time at the two flip-flops.

To simplify the equation, T_{net} and $T_{\text{clk}_{\text{skew}}}$ can be ignored. In order to make sure that $\text{Slack}_{\text{setup}}$ and $\text{Slack}_{\text{hold}}$ are positive, we can derive Eqs. 5.3 and 5.4 (plus T_{setup} on both sides) from Eqs. 5.1 and 5.2.

$$T_{\text{period}} > T_{\text{cq}} + T_{\text{logic}} + T_{\text{setup}} \tag{5.3}$$

$$T_{cq} + T_{logic} + T_{setup} > T_{hold} + T_{setup} \tag{5.4}$$

Combine Eqs. 5.1 and 5.2, we can have Eq. 5.5.

$$T_{hold} + T_{setup} < T_{cq} + T_{logic} + T_{setup} < T_{period} \tag{5.5}$$

$T_{cq} + T_{logic} + T_{setup}$ is the data propagation delay, if it is greater than T_{period}, the data will not arriving when the second register is sampling, on the other hand, if it is smaller than the register sampling window ($T_{hold} + T_{setup}$), the registers could fall into metastability.

In FPGA design, STA can be performed in different stages: post-synthesis (logical level) and post-implementation (physical level). Post-synthesis STA (based on ideal implementation information) is faster but less accurate than post-implementation STA (based on real implementation information).

5.2 Timing Analysis Techniques

STA usually requires a timing graph that describes the target design from the timing perspective, identifying all the timing paths. The timing graph consists of nodes and edges, nodes correspond to component pins or input/output ports, and edges are the timing path between them. Edges have attached weights that can denote some characteristics such as delay values [2].

Timing Graph Definition: A timing graph G = N, E, s, t is a directed graph having exactly one source node s and one sink node t , where N is a set of nodes, and E is a set of edges. The weight associated with an edge corresponds to either the gate delay or the interconnect delay (Fig. 5.3).

Traditional STA is deterministic (DSTA) and compute the circuit delay for a specific condition. In practice, the worst-case slow or best-case fast process is typically used and this could lead to over-design, leaving a lot of margin on the table in terms of PPA. Statistical STA (SSTA) then come out to address this problem. It combines

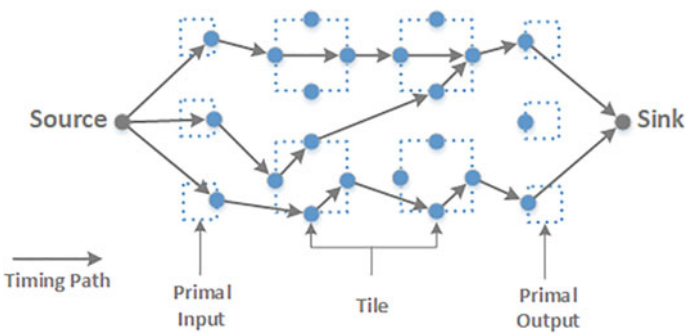

Fig. 5.3 Example of timing graph

the delays along the timing paths which is expressed statistically (with mean and standard deviations) to obtain the overall delay data.

SSTA is also employed by Intel in its Quartus Prime software to mitigate the effect of random variation on longer paths [3]. By discounting the minimum/maximum delay spread on these paths, the FPGA performance reported by STA may increase. There are two main categories of SSTA techniques–path-based and block-based.

1. Path-based
   In path-based STA technique, critical path is searched in an exhaustive way. The statistical calculation is simple, but the paths of interest must be identified prior to running the analysis [4–6].
2. Block-based
   In block-based STA technique, the circuit timing graph is traversed in a topological manner. In [7], two basic graph traversal algorithms–depth first search (DFS) and breadth first search (BFS) are applied to STA module and the runtime efficiencies is compared by testing a large number of sequential circuit instances. The conclusion is that BFS algorithm can implement STA module more efficiently than DFS algorithm. Due to its runtime advantage, many research [8–11] and commercial efforts have taken the block-based approach. The advantage is completeness, and no need for path selection, however, to compute statistical max (or min) of random variables is not trivial.

The choice of using path-based analysis or block-based analysis depends on several factors, such as the design complexity, stage, and goal. Generally, path-based analysis is more suitable for small or medium-sized designs, where the number of paths is manageable and the accuracy is important. It can also be used for final verification or optimization, where the timing margins are tight and the details are needed. On the other hand, block-based analysis is more suitable for large designs, where the number of paths is overwhelming and the runtime is important. It can also be used for FPGA architecture exploration, where the timing budget is loose and the trends are sufficient.

In some cases, it could be more optimized to combine both techniques and use them in different stages or levels of the design. For example, one can use block-based analysis for the system-level design, where the blocks are abstracted and the overall timing is estimated. Then, one can use path-based analysis for the block-level design, where the paths are detailed. The balance between accuracy and efficiency can be obtained in this way [12].

5.3 Summary and Trends

The state-of-the-art STA engines still can not replace DTA (simulation) completely because there are some aspects of timing verification that cannot yet be completely captured and verified in STA [13]. Some of these limitations include:

1. Inaccurate timing models
   The timing models used in FPGA STA may not accurately represent the behavior of the actual circuit due to the complexity of the FPGA architecture.
2. Lack of support for dynamic circuits
   FPGA STA assumes that the circuit is static and does not take into account dynamic circuits such as state machines or circuits with feedback paths.
3. Impact of environmental conditions
   FPGA STA assumes ideal environmental conditions, such as constant temperature and voltage, which may not hold true in the real world.

Although FPGA STA has been matured for many years, it still benefits from emerging technologies. The following are some of the recent trends in FPGA STA:

1. Parallel acceleration
   Parallel STA on different computing platforms is one of the researching hot spots, such as multi-core CPUs [4, 14–17] and GPUs [16, 18].
2. AI (machine learning) acceleration
   ML algorithms are increasingly being used to analyze the timing characteristics of FPGA designs [19–21]. ML-based timing analysis can quickly identify critical paths in the design, predict the timing behavior of the design, and optimize the design for timing performance.

References

1. IEEE standard for standard delay format (SDF) for the electronic design process, in *IEEE Std. 1497-2001* (2001), pp. 1–80
2. J.L.M. Lee, A scalable method to measure similarity between two EDA-generated timing graphs, in *2015 International Conference on Computer, Communications, and Control Technology (I4CT)* (2015), pp. 44–48
3. Intel, Guaranteeing silicon performance with FPGA timing models. https://cdrdv2-public.intel.com/650314/wp-01139-timing-model.pdf
4. T.-W. Huang, M.D.F. Wong, UI-timer 1.0: an ultrafast path-based timing analysis algorithm for CPPR. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **35**(11), pp. 1862–1875 (2016)
5. D. Mishagli, E. Koskin, E. Blokhina, Path-based statistical static timing analysis for large integrated circuits in a weak correlation approximation, in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)* (2019), pp. 1–5
6. L.-W. Chen, Y.-N. Sui, T.-C. Lee, Y.-L. Li, M.C.-T. Chao, I.-C. Tsai, T.-W. Kung, E.-C. Liu, Y.-C. Chang, Path-based pre-routing timing prediction for modern very large-scale integration designs, in *2022 23rd International Symposium on Quality Electronic Design (ISQED)* (2022), pp. 1–6
7. J. Lu, N. Xu, J. Yu, T. Weng, Research of timing graph traversal algorithm in static timing analysis based on FPGA, in *2017 IEEE 3rd Information Technology and Mechatronics Engineering Conference (ITOEC)* (2017), pp. 334–338
8. L. Zhang, Y. Hu, C.-P. Chen, Block based statistical timing analysis with extended canonical timing model, in *Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference, 2005.*, vol. 1 (2005), pp. 250–253

9. R. Chen, H. Zhou, New block-based statistical timing analysis approaches without moment matching, in *Proceedings of the ASP-DAC 2007—Asia and South Pacific Design Automation Conference 2007*, Series. Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC (2007), pp. 462–467
10. G. Luo, B. Jin, W. Zhang, A fast and simple block-based approach for common path pessimism removal in static timing analysis, in *2015 14th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)* (2015), pp. 234–235
11. L. Jin, W. Fu, Y. Zheng, H. Yan, A precise block-based statistical timing analysis with max approximation using multivariate adaptive regression splines, in *2019 IEEE 13th International Conference on ASIC (ASICON)* (2019), pp. 1–4
12. T.-W. Huang, M.D.F. Wong, OpenTimer: a high-performance timing analysis tool, in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (2015), pp. 895–902
13. J. Bhasker, R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*, 1st edn. (Springer, 2009)
14. Y.-M. Yang, Y.-W. Chang, I.H.-R. Jiang, iTimerC: common path pessimism removal using effective reduction methods, in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (2014), pp. 600–605
15. T.-W. Huang, M.D.F. Wong, D. Sinha, K. Kalafala, N. Venkateswaran, A distributed timing analysis framework for large designs, in *Proceedings of the 53rd Annual Design Automation Conference*, Series DAC '16 (Association for Computing Machinery, New York, NY, USA, 2016). Available https://doi.org/10.1145/2897937.2897959
16. K. E. Murray, V. Betz, Tatum: parallel timing analysis for faster design cycles and improved optimization, in *2018 International Conference on Field-Programmable Technology (FPT)* (2018), pp. 110–117
17. T.-W. Huang, G. Guo, C.-X. Lin, M.D.F. Wong, OpenTimer v2: a new parallel incremental timing analysis engine. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **40**(4):776–789 (2021)
18. Z. Guo, T.-W. Huang, Y. Lin, GPU-accelerated static timing analysis, in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)* (2020), pp. 1–9
19. S. Bian, M. Shintani, M. Hiromoto, T. Sato, LSTA: learning-based static timing analysis for high-dimensional correlated on-chip variations, in *Proceedings of the 54th Annual Design Automation Conference 2017*, Series DAC '17 (Association for Computing Machinery, New York, NY, USA, 2017). Available https://doi.org/10.1145/3061639.3062280
20. A.B. Kahng, U. Mallappa, L. Saul, Using machine learning to predict path-based slack from graph-based timing analysis, in *2018 IEEE 36th International Conference on Computer Design (ICCD)* (2018), pp. 603–612
21. M.A. Savari, H. Jahanirad, NN-SSTA: a deep neural network approach for statistical static timing analysis, Expert Syst. Appl. **149**, 113309 (2020). Available https://www.sciencedirect.com/science/article/pii/S0957417420301342