Timing Analysis: Dynamic, Static, and an Augmented Approach

By Sheila Tran, Apex Semiconductor

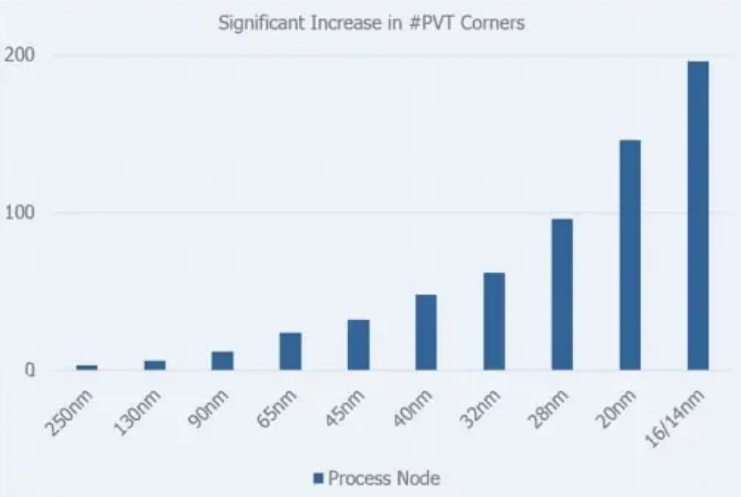
VLSI designs have become increasingly complex with every new process node, leading to complications in timing analysis techniques. In this paper, we'll discuss how timing analysis methods have evolved to meet designer needs and examine how new technologies like the Apex Semiconductor STA Timing Estimator are providing innovative solutions to industry-wide challenges.

Introduction

One of the most crucial and challenging aspects of VLSI design is timing closure, which ensures that a circuit can operate at its intended frequency and meet power, performance, and area specifications. After designing the chip, designers must validate the speed that it runs, the speed at which it interacts with other chips, and other timing interactions that affect performance.

To test these timing interactions, designers use timing analysis to measure the delay of the design at various stages and make subsequent design fixes. As VLSI designs become increasingly complex and the number of timing paths and operating conditions has grown, however, modern timing analysis methods have developed runtime, coverage, and accuracy issues.

Figure 1: Increase in PVT corners with shrinking process nodes



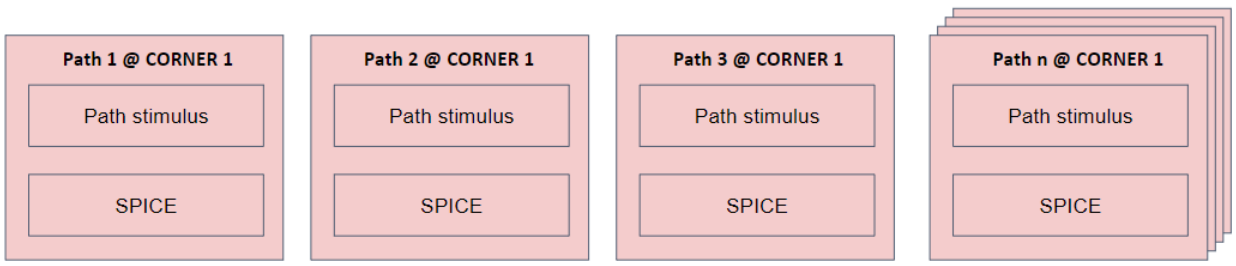
Dynamic timing analysis

The traditional timing analysis method is dynamic timing analysis (DTA), which uses tools such as SPICE and Spectre to simulate the entire logical operation of a circuit. Given a set of comprehensive input vectors, this method simulates the circuit and verifies timing by checking for correct output vectors. Conceived to model circuit behavior of non-linear circuit devices and components in the 1960s, DTA is the gold standard of accuracy for passive and active circuit elements alike.

While DTA has strengths in accuracy, it suffers from coverage and runtime issues. Because this method only performs timing analysis on circuitry that is exercised by test stimulus, it may leave some critical paths untested and timing issues undiscovered. DTA also relies on the time-consuming generation of test vectors, which cannot be generated through a manual process due to the complexity of modern designs.

The increasing complexity of chip designs has exacerbated the runtime issues of DTA. DTA cannot reasonably scale to more than 100~1000 timing paths, while the modern design now has at least 10 million timing paths. There are so many simulation combinations of gate and timing paths that it is no longer feasible to comprehensively simulate the entire design and identify the critical paths in a reasonable timeframe.

Figure 2: DTA analysis architecture model



In the modern chip design process, DTA now serves as the foundation for a newer timing analysis method: static timing analysis.

Static timing analysis

Becoming prevalent in VLSI in the 1980s, static timing analysis (STA) was developed to alleviate the runtime and coverage issues of dynamic timing analysis. Given characterized libraries from DTA and a design netlist, STA uses tools such as PrimeTime and Innovus to identify all timing paths through graph- or path-based algorithms.

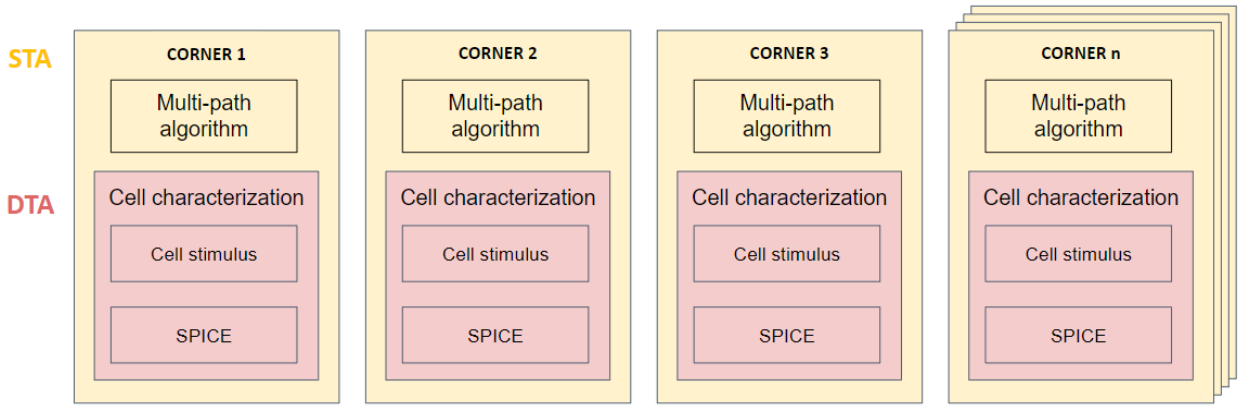
The process of STA begins by abstracting cell-based design components into smaller building blocks. This method relies on DTA to perform library characterization for all possible combinations of the building blocks. Because these building blocks are simpler compared to full timing paths, their timing relationships can be simulated by DTA in a more reasonable timeframe. STA is then performed on the unique combinations of the simple building blocks rather than the traditional process of using DTA to simulate every possible full timing path.

STA does not require simulating the entire circuit, which greatly reduces runtime compared to DTA. STA also features improved coverage because it checks all possible timing paths rather than only the logical conditions that are sensitized by a set of test vectors.

However, a major drawback of STA is that it depends on library characterization to obtain an accurate representation of the cell. When a circuit behaves differently in different operating conditions, library characterization must be performed before any analysis can begin. As modern integrated circuits become increasingly complex, however, the number of corners has also grown with each process node. The average number of sign-off corners in modern designs is now 20~30, while STA can only manually analyze 1~3 corners at a time.

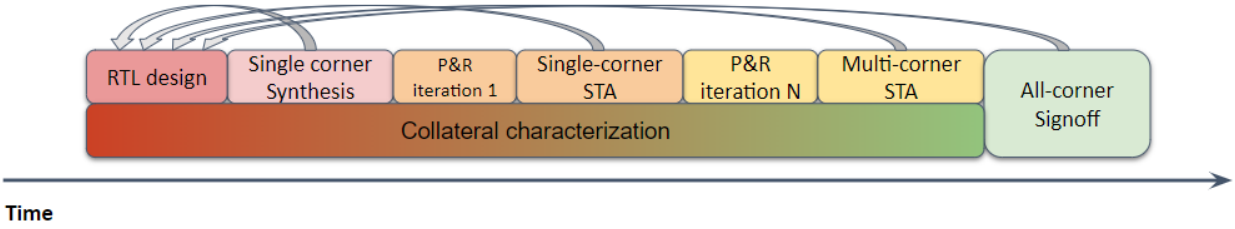
Figure 3 below displays the STA analysis architecture model, which incorporates the cell characterization of DTA. See Figure 2 for the DTA analysis architecture model.

Figure 3: STA analysis architecture model



The conventional design process now requires numerous runs of library characterization and consists of a long and continuous loop of design iteration, library characterization, and STA. Runtime has significantly increased, which has subsequently led to issues with integration. Because timing results are generated so late in the design cycle, designers cannot make meaningful fixes to early design stages.

Figure 4: Conventional STA design flow



The increasing complexity of chip designs has led to runtime, accuracy, and integration issues in the two classic STA integration models: single-corner analysis and all-corner analysis for sign-off.

Single-corner static timing analysis

Single-corner analysis is used when characterized libraries for some corners are unavailable. In this model, designers develop the chip based on currently available information while waiting for the remaining characterized libraries. Because the process of library characterization is so lengthy and only 1~3 operating conditions can be tested at a time, timing violations are typically revealed too late in the design cycle for designers to correct all errors.

Figure 5: Single-corner STA design phase integration

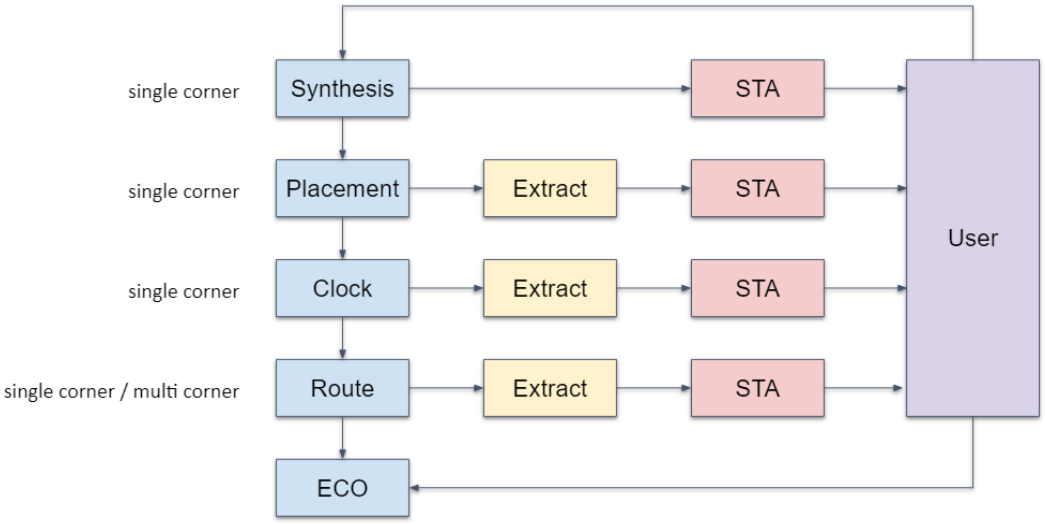


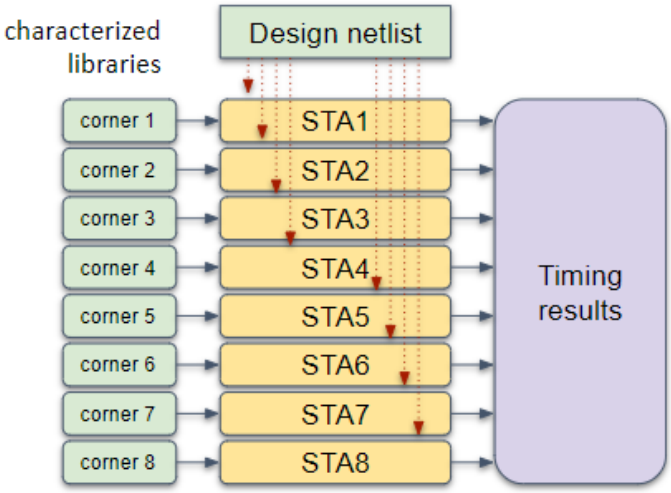
Figure 5 above displays that single-corner timing information is typically used to drive logic design and design optimization beginning from the synthesis phase into subsequent phases. At various stages of the design cycle, designers must work with a limited subset of timing information and

ultimately make design optimization trade-offs. The process of waiting for corners to become available also incurs a significantly longer runtime because library characterization can take weeks or even months to complete.

All-corner static timing analysis for sign-off

All-corner STA for sign-off requires all characterized libraries to be available before performing any analysis. The drawback of this sign-off integration model is that the lengthy process of library characterization typically isn't completed until late in the design cycle, when the design is already finalized. As a result, the results of all-corner STA cannot be used to meaningfully improve early stages of the design.

Figure 6: All-corner STA for sign-off usage model



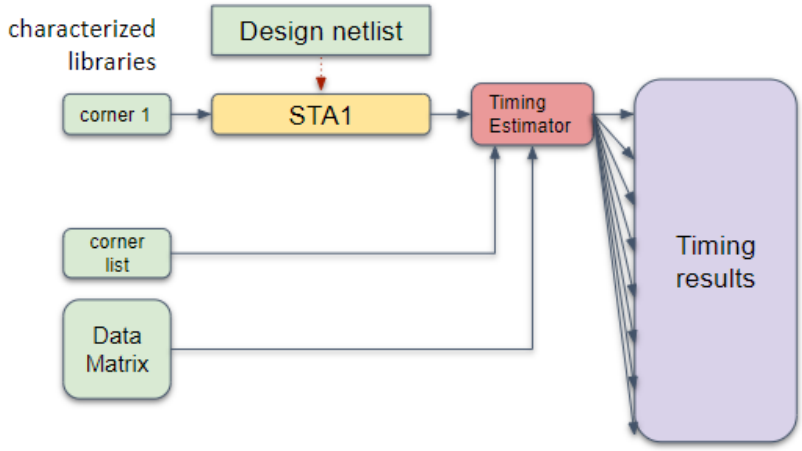
Designers can alternatively avoid waiting for library characterization by mixing and matching currently available corners with STA conditions. In Figure 6 above, a designer may want to check corner 6 with STA condition 6. If the characterized libraries for corner 6 are not yet available, the designer can use corner 5 with STA condition 6 to generate timing results. While this usage model decreases runtime, it incurs a considerable decrease in accuracy. Ultimately, the classic STA integration models suffer from runtime, accuracy, and integration issues.

Apex Semiconductor's solution for efficient timing analysis

The Apex Semiconductor STA Timing Estimator is an early-stage timing feedback tool that builds on DTA and STA while alleviating the runtime and integration issues of both timing analysis methods. Using machine learning technology, the Timing Estimator uses DTA characterized libraries and STA results from a single corner to estimate frequency and critical paths in all corners.

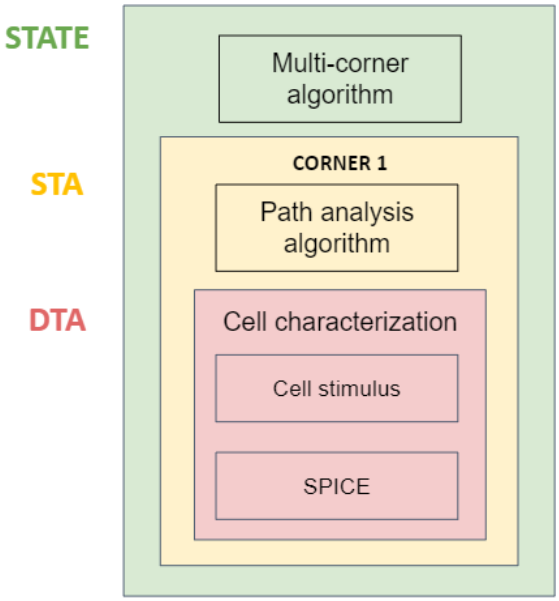
Because the Timing Estimator does not rely heavily on characterized DTA data, it has a significantly lower runtime compared to the classic STA usage model displayed in Figure 6.

Figure 7: STA Timing Estimator usage model



The Timing Estimator leverages STA's accurate and comprehensive coverage engine while using a multi-corner algorithm to supplement its reliance on the slow DTA cell library characterization process. Figure 8 below displays the STA Timing Estimator analysis architecture model, which builds on the path analysis of STA and cell characterization of DTA. See Figure 2 for the DTA analysis architecture model and Figure 3 for the STA analysis architecture model.

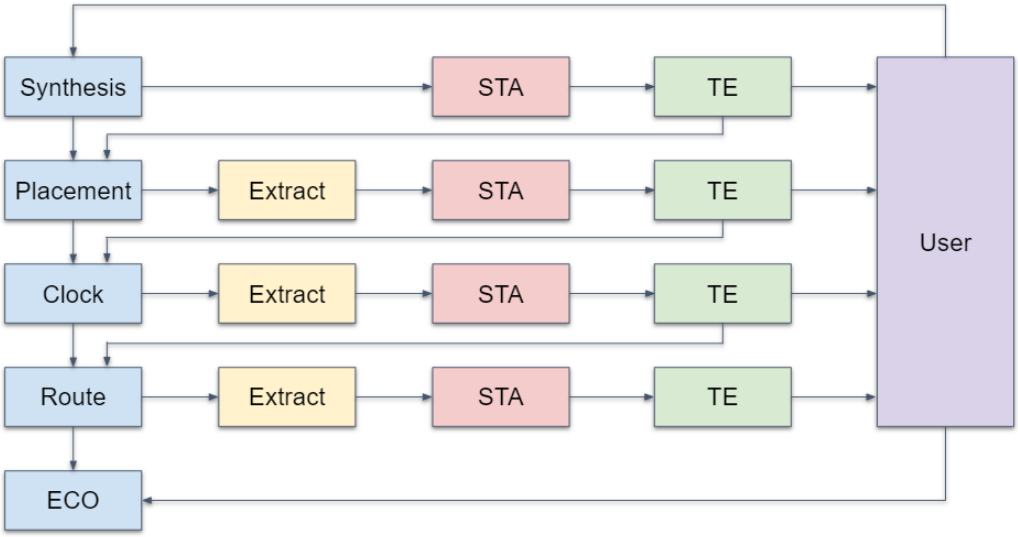
Figure 8: STA Timing Estimator analysis architecture model



The Timing Estimator also alleviates the integration issues of classic STA displayed in Figure 5. Using the Timing Estimator, designers can identify corners containing critical paths as early as physical synthesis and throughout the design process. Because Apex's solution has a significantly lower runtime than classic STA, timing results can be integrated into each individual design phase for

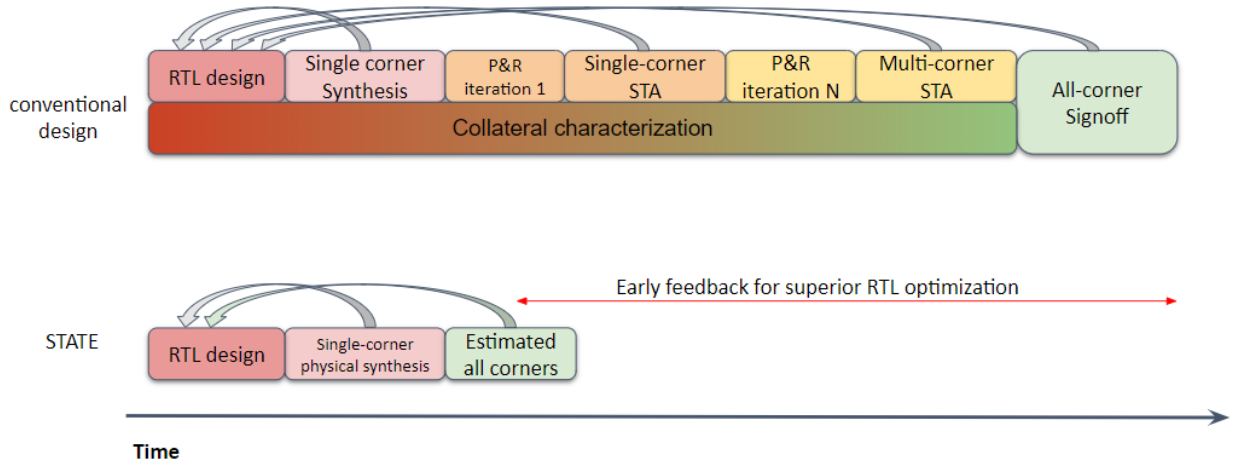
meaningful and early design fixes before advancing to subsequent phases. As a result, the Timing Estimator greatly reduces the number of design iterations and enables a higher-performance design.

Figure 9: Timing Estimator design phase integration



By bypassing the conventional STA model's long loop of design iteration, characterization, and STA, the Timing Estimator allows for efficient timing sign-off and design convergence over a large number of PVT design conditions. Furthermore, the tool is not constrained by the growing number of operating conditions in advanced technologies.

Figure 10: Timing Estimator design flow comparison



The Timing Estimator does not require IP or cell library characterization, simulator license, or STA license. The tool is built with a customized dataset that features expansive coverage of a variety of timing paths, gate and topological permutations, and delay and slew property permutations. This customized dataset ensures accurate coverage across a wide range of designs.

In addition, the dataset allows the Timing Estimator to identify paths in the design that are critical in operating conditions outside of the typical logic design scope. These timing outliers impact design performance in the field, but are typically analyzed and reported too late in the design cycle for practical design fixes.

Conclusion

As chip technologies become increasingly complex, timing analysis methods have evolved to meet designer needs. The traditional and most accurate method, dynamic timing analysis, now suffers from runtime issues due to the exponential growth in timing paths. Static timing analysis was built on DTA to offer a faster alternative, but its reliance on DTA to perform library characterization has led to challenges with runtime and integration as the number of design operating conditions has also increased.

The current static timing analysis model can take months to complete, forcing designers to make design and timing trade-offs late in the design cycle. As VLSI designs become increasingly advanced with each shrinking process node, new timing analysis methods are needed to develop high-performance chips in an efficient timeframe.

Apex Semiconductor's STA Timing Estimator uses AI-powered technology to further augment the dynamic and static timing analysis processes. The Timing Estimator identifies timing violations months faster than classic STA with little to no accuracy loss while also capturing timing outliers that are typically missed in traditional timing analysis methods. Apex Semiconductor's timing analysis solution enables designers to reach timing sign-off and design convergence in record time for the advanced technologies of today and beyond.

Figure 11: Timing analysis methods comparison

	Dynamic timing analysis	**Static timing analysis**	**STA Timing Estimator**
Coverage	Moderate	High	High
Accuracy	Very high	High	High
Runtime	Very high	High	Low
Usage	Single corner	Single corner	Multi-corner
Integration with design cycle	No longer feasible	Incremental; results generated too late	Fully integrated