Clock Sensitivity Test

Lei Han @Synopsys

October 4, 2024

Abstract

Customers and internal may experience QoR instability due to minor changes in flow, settings, or design. These fluctuations complicate the debugging process and compromise the stability of release-to-release results. A sensitivity test was initialized to assess the engine's response to external stimuli in IC Compiler II. Improvements were accomplished from this experiment, encompassing CTS, routing, extraction, data optimization, etc. This guarantees a stable and predictable tool performance and enables adaptability to customer scenarios. Recent customer feedback has confirmed the success of this approach in streamlining the migration process and reducing development costs. As we look to the future, an AI model for predictive analysis based on historical sensitivity data is underway.

1 Background

In the process of testing an engine/feature, a standard, referred to as "GOLDEN", is introduced to facilitate the comparison of testing results. Although the current testing methodology has proven effective, there remains a possibility that customers may encounter challenges in migrating smoothly between different branches. From an EDA perspective, both customers and internal stakeholders may experience Quality-of-Results (QoR) instability due to minor changes in flow, setting, or design. These fluctuations can complicate the debugging process and jeopardize the stability of release-to-release results. This concern, centered on the customer experience, can be attributed to the high sensitivity of the EDA tools.

2 Idea Initialization

To address this issue, a sensitivity test has been initiated to ascertain how an engine will respond in terms of QoR to minor interferences. This experiment aims to identify the root causes of sensitivity and rectify them, offering an opportunity to enhance the quality of the software. Furthermore, it represents a step towards achieving the global optimal solution. To delve deeper into this methodology, consider a design pool comprising a set of unit cases created for testing. While a stable environment with consistent flow, build, setup, and constraints would yield predictable outcomes, customers using our tool have various unpredictable inputs. Consequently, predicting the response of designs to these interferences becomes challenging. However, formulating hypotheses such as the impact of QoR increasing with the size of the interference or exhibiting linear growth, whether intuitive or counterintuitive, could guide our understanding. Conducting tests from this perspective is uncommon, as the conventional approach involves applying the aforementioned GOLDEN method. However, to ensure the broader stability and consistency of the tools, such tests appear essential.

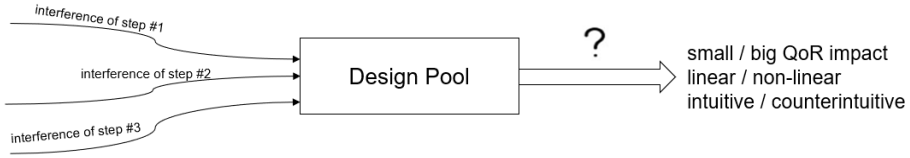

Figure 1: Formulating Hypotheses

3 Implementation

In the domain of IC Compiler II backend clock tree synthesis(CTS), the primary objective of CTS is to establish a delay-balanced, timing-compliant, and area-power-optimized tree structure that encompasses all registers within the design.

3.1 Interferences

When addressing interference types for CTS, it is imperative to ensure that these interferences are sufficiently minor to have an inconsequential impact on the clock tree. Some interferences for CTS could be delay margin changes, balance point settings, register locations, and placement blockages. The implementation process can be delineated into the following key segments.

3.2 Design Suite

The initial aspect pertains to the design suite, which plays a pivotal role in customizing the test cases. The first dimension relates to the selection of the library to be utilized. To align with the evolving design requisites of customers, diverse new tech nodes will be embraced to cover varied benchmarks.

3.3 Clock Structure

For the clock structures, it is essential to strike a balance between simplicity for unit testing, ensuring runtime and debugging cost-effectiveness, and complexity to simulate customer clock scenarios effectively.

3.4 Automation

Following the establishment of the library and clock structure, the adoption of automation tools becomes imperative to enhance productivity, enabling seamless automatic mapping and the formulation of metrics for result evaluation.

3.5 Flow

After the suite is developed, the subsequent step involves delineating the test flow to simulate customer usage. This encompasses integrating real CTS environments and applying diverse types and sizes of interferences, followed by the execution of CTS.

4 Result Analysis

Upon completion of all preparatory measures, scrutinizing the results becomes paramount. It is crucial to elucidate expectations and identify unforeseen behaviors necessitating attention to discern and ascertain their root causes. The observed sensitivity phenomena can be classified into three distinct situations, each exemplifying specific changes induced by minor interferences: register cluster changes(2), buffering selection differences and branching points. These changes are discernible and affirm that sensitive situations may be intricately interwoven. However, upon meticulous dissection, they can be understood as composites of the aforementioned causes. Consequently, these sensitivity tests unveil optimality issues that are intrinsic to advancing our tool toward optimality. Therefore, the sensitivity test is an indispensable phase in fortifying the optimality of our tool.

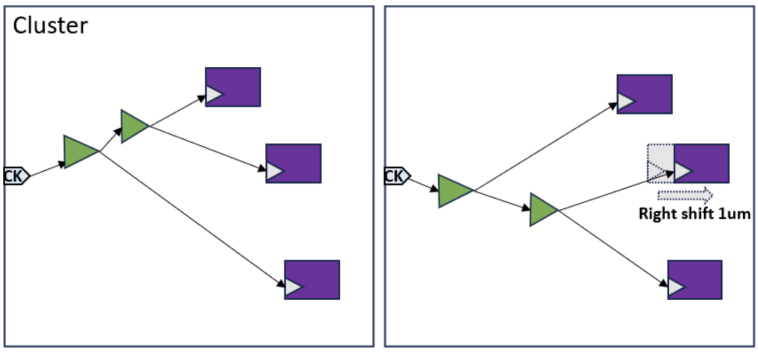

Figure 2: Register Cluster Changes

5 Benefits

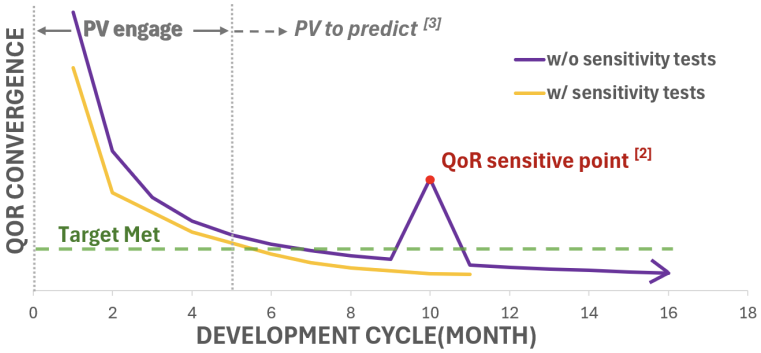

Figure 3: Left-shifting Development Cycle

Sensitivity tests significantly broaden the scope of testing and can be applied across various domains and types of engines. This methodology ensures consistent and optimal product performance, allowing the tool to accommodate diverse customer scenarios and intricate design requirements while enhancing product reliability in practical applications. Recent customer feedback has confirmed the success of this approach in streamlining migration processes and reducing development costs. It is conceivable that an AI model, be it a neural network, decision tree, or support vector machine, could undergo training to conduct predictive analyses on new designs based on historical sensitivity tests and anticipate potential risks in advance.

References

[1] P. V. Vishnu, A. R. Priyarenjini, and N. Kotha, "Clock tree synthesis techniques for optimal power and timing convergence in soc partitions," in *2019 4th International Conference on Recent Trends on Electronics, Information, Communication Technology (RTEICT)*, 2019, pp. 276–280.

[2] G. M. Madhuri, J. Selvakumar, and K. S. Krishna, "Performance analysis on skew optimized clock tree synthesis," in *2022 Fourth International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT)*, 2022, pp. 01–06.

[1] [2]