Static Timing Analysis (STA) Guide

Introduction

This guide outlines the process for performing basic Static Timing Analysis (STA) using Cadence Tempus. STA is a vital verification technique used to validate whether a digital design meets required timing constraints across various process, voltage, and temperature conditions. This guide walks through preparing the environment, importing files, running timing checks, and interpreting reports.

1. Overview of Genus Synthesis

Static Timing Analysis (STA) is a simulation-independent method used to evaluate the timing performance of digital circuits. It verifies that signals propagate through combinational logic within defined clock periods, ensuring correct functionality. STA detects timing violations such as setup and hold errors between sequential elements. The analysis can be performed for different operating corners to ensure design robustness under all expected conditions.

2. Setting Up the Environment

Before executing STA in Tempus, the environment must be properly configured. The following setup parameters are required:

- **Top Module Name**: Defines the hierarchical entry point of the design for timing analysis.
- **File Paths**: Specify the exact locations of all required design and constraint files.
- **Output Directories**: Designate folders for saving reports and analyzed design snapshots.

Harsh A Patil
For any queries, feel free to contact me at harshpatilhp28@gmail.com.

3. Input File Requirements

STA depends on a variety of design and constraint files that describe circuit structure and behaviour:

- **Library Files (.lib)**: Contains technology-specific standard cell definitions.
- **RTL Files (.v)**: Represents the circuit in Verilog hardware description language.
- **SDC Files (.sdc)**: Defines design constraints such as clocking and timing requirements.
- **SPEF File (.spef)** *(optional)*: Provides parasitic resistance and capacitance data for post-layout timing accuracy.

Tempus uses these files to construct an internal timing graph, enabling comprehensive analysis of data paths and constraints.

4. Design Loading and Setup

The design loading phase involves reading the required input files into the Tempus environment using the following sequence:

1. **Read Library Files**: Load standard cell timing libraries (.lib).
2. **Read Verilog Netlist**: Import the synthesized structural netlist.
3. **Set Top Module**: Assign the top-level module for STA processing.
4. **Read SDC File**: Apply design constraints including clock and I/O specifications.
5. **Read SPEF File (Optional)**: Load parasitic data for post-layout analysis.

Each step ensures that the timing model is accurate and reflects real-world physical effects where applicable.

Harsh A Patil
For any queries, feel free to contact me at harshpatilhp28@gmail.com.

5. Timing Analysis Execution

Once the design and constraints are loaded, timing analysis is performed. Tempus automatically computes path delays, clock skews, and slack values across all timing paths. It evaluates both setup and hold timing for every register-to-register path and reports any violations.

Typical STA modes include:

- **Setup Analysis**: Checks whether data arrives before the clock edge.
- **Hold Analysis**: Ensures data remains stable after the clock edge.
- **Path-based Analysis**: Provides detailed slack breakdown for each path.

6. Report Generation

After executing the timing checks, Tempus generates a variety of reports:

- **Timing Report**: Highlights critical paths, worst negative slack (WNS), and total negative slack (TNS).
- **Violations Report**: Lists all timing constraint violations.
- **Slack Report**: Summarizes slack values across endpoints and timing paths.
- **Cross-Probing Reports**: Provide GUI integration with waveforms and netlist views for debugging.

These reports help identify timing bottlenecks and guide optimization strategies in the design.

7. Final Outputs and Verification

The final step involves reviewing and saving all reports for further action. Designers may use this data to:

- Adjust timing constraints in the SDC.
- Perform gate-level optimizations or buffering.
- Provide feedback to synthesis or place-and-route teams.

Verification of the STA results ensures that the design is timing-clean and ready for signoff or physical implementation.

Best Practices for Effective STA

To ensure accurate and efficient STA, consider the following recommendations:

- **Accurate Constraints**: Ensure the SDC file reflects real-world operating conditions and interfaces.
- **Use Post-Layout Data**: For final signoff, always include parasitics via the SPEF file.
- **Corner Analysis**: Run STA across multiple PVT corners to guarantee reliability.
- **Incremental Analysis**: Continuously perform STA after every major design change.
- **Script Automation**: Use TCL scripts for reproducible, consistent analysis.

By following these best practices, STA with Tempus becomes an efficient and reliable part of the ASIC/SoC verification flow.

Harsh A Patil
For any queries, feel free to contact me at harshpatilhp28@gmail.com.