Transistor Level Static Timing Analysis with NanoTime

Authors

Eduardo Flores,
Jim McCanny,
Synopsys

Background

Timing verification is essential to make sure your design will function and perform as expected when it is manufactured in silicon. To ensure complete coverage of all timing paths, static timing analysis needs to be used to complement dynamic simulation. For custom designs such as datapaths blocks, register files, embedded memory and mixed signal blocks this requires performing analysis at the transistor level. This paper discusses the capabilities of NanoTime, Synopsys' transistor-level static timing analyzer.

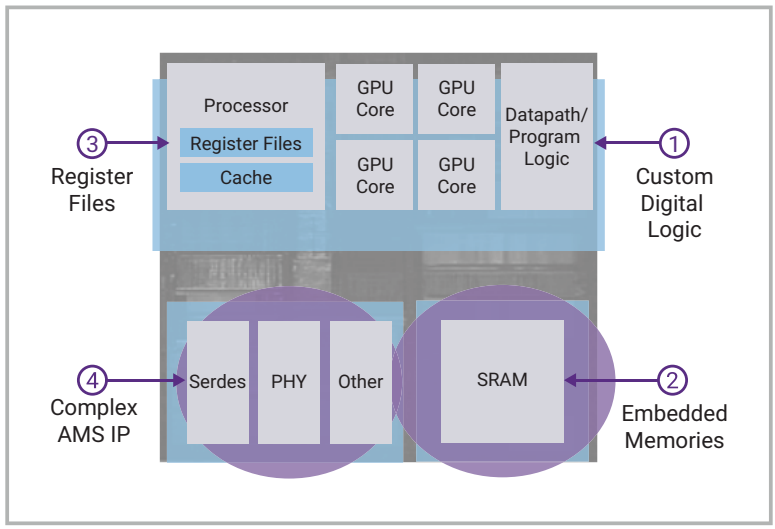


Figure 1: Large SOC design with custom blocks suitable for analysis with NanoTime

Introduction

The cost of silicon failure for designs using advanced process technology such as FinFET is very significant especially when time to market is accounted for. Hence signoff analysis is critical to ensure that the design is free from fatal timing problems such as hold violations. Additionally, at advanced nodes functional failures due to noise caused by increased wire coupling need to be found and fixed before tapeout. For cell-based designs, static timing analysis is the typical way to verify and signoff a chip's timing and noise immunity. However, often for custom design blocks, verification is performed using only vector-based simulation. While simulation is essential to ensure a chip is functioning correctly, it is usually not exhaustive enough to catch all timing or noise scenarios that can lead to silicon failures. For custom design blocks, where possible, dynamic simulation should be complemented with static transistor level timing and noise analysis.

Item	NanoTime	Dynamic Simulation
Design Style: Analog, RF	No	✓
Functionality Analysis	No	✓
Performance	**Hours**	Days
Timing Constraint Checks	✓	No
Noise Glitch Analysis	✓	No
Variability	✓	Very few paths
Design Coverage	**Exhaustive**	Vector-dependent
Signoff	**Comprehensive**	Limited by designer ability to pick worst path

Figure 2: Comparing dynamic simulation to static analysis

Static Timing Verification

Timing verification is the process of determining that a given design can be operated at a specific clock frequency without errors caused by a signal arriving too soon or too late. For example, if the data input of a latch arrives after the closing edge of the clock (a setup violation), or if the data input changes before the closing edge of the previous clock (a hold violation), the latch may not store the data correctly. Given a specific scenario of input and clock transitions, a dynamic simulator can be used to determine if a particular sensitization leads to a timing violation in the circuit block. By simulating under all possible sequences of transitions, it can be determined whether the block can operate at the given clock frequency without any timing violations. Unfortunately, the number of input and clock sensitizations required for such an exhaustive validation is exponential in the number of inputs and state elements and so this method is impractical for all but very small blocks.

Unlike dynamic simulation, Static Timing Analysis (STA) tools remove the need for simulating the entire block under all possible scenarios. Instead, STA tools use fast, but accurate approaches to estimate the delay of sub-components within the block and use graph analysis techniques to quickly seek out the slowest and fastest paths. The result is that STA can find all timing violations in a fraction of the time it would take a dynamic circuit simulator. For cell-based designs the underlying sub-components are represented as pre-characterized timing and noise tables in a cell library. For transistor level designs, these sub-components are created and simulated on-the-fly as paths are traced. Hence transistor level static analysis can be applied to a much wider range of design styles, i.e. those where the underlying behavior of the block is pre-dominantly digital with full rail swings. If a design is a mixture of digital and analog (e.g. a mixed signal IP block) it can still be analyzed if the analog portion is identified so it can be by-passed. NanoTime is Synopsys' tool for transistor level signoff, complementing PrimeTime® the tool for cell-based designs.

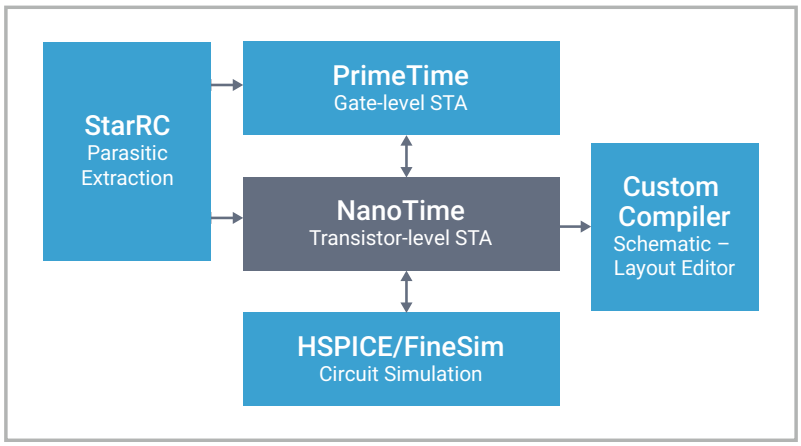


Figure 3: Synopsys signoff solution

NanoTime Inputs and Outputs

NanoTime uses industry standard formats for transistor-level analysis such as standard netlist formats (HSPICE, CDL, Verilog etc.), standard parasitic file formats (dspf, SPEF) and standard SPICE device model formats. NanoTime also reads Synopsys Design Constraint format (SDC) to specify the clocks, inputs and outputs of the design being analyzed.

NanoTime reports a list of the most critical paths, both max and min, including how much timing slack each path has available. Timing slack is the difference between the required delay for a path and the actual delay. If a path has negative slack it is deemed a violation and needs to be repaired. NanoTime can also generate a timing and noise model in Liberty format (.lib) so that the results from analyzing a custom block can be incorporated in a cell-based signoff flow with PrimeTime. Each path in a timing report can be output as a ready to simulate SPICE netlist for further detailed analysis and to aid fixing of violations.

NanoTime also reports noise violations (above low, below high, below low, above high) to each net, including injected and propagated noise impact. Noise violation sources (i.e. aggressors) are detailed so designers can find and fix circuits which are contributing the most for noise violations at victim nets.

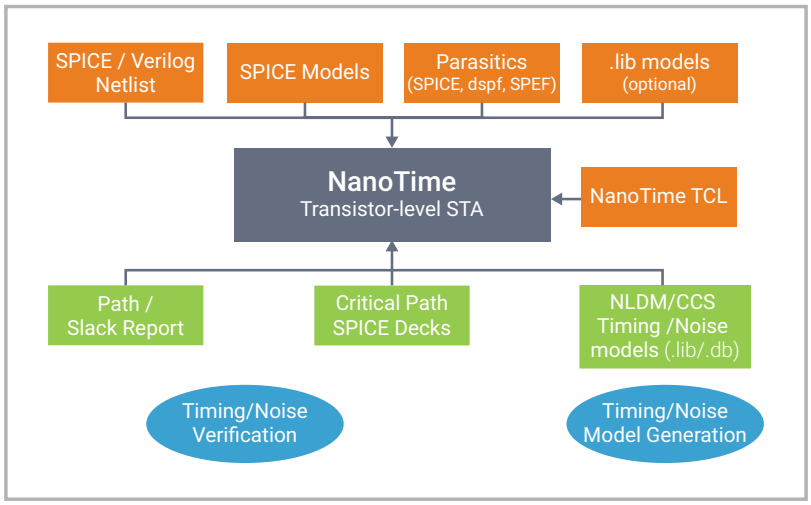


Figure 4: NanoTime inputs/outputs

Path Tracing

NanoTime uses path-based tracing to propagate delay and noise information through the design. Each path starts from an input port and tracing continues until it reaches an output port or a timing element (e.g. a flip-flop) that prevents the path from propagating further. As the path is propagated from net to net it passes through clusters of transistors.

A transistor cluster is a small sub-component that NanoTime creates that is made up of transistors that can be locally isolated from their surrounding circuitry and simulated without any accuracy loss. Each transistor cluster will be simulated during the path traversal in the context of the path that it belongs to. This ensures that both the input transition time and the effective loading of the cluster is accounted for. The simulation of the cluster is performed by a built-in simulator that is optimized for computing delay and transition. Alternatively, the HSPICE or FineSim® simulators can be used including selectively, e.g. for the clock tree.

When a path reaches a timing element such as a gated clock or a latch, it is checked against the required arrival time. The required arrival timing is determined from the clock signal that controls the timing element. Hence, NanoTime must identify clocks and understand how the ideal clock signal waveform is modified along the path to each timing element. NanoTime will check for setup time violations when it is tracing the slowest (max) path and hold time violations when tracing the fastest (min) paths. For setup, the slowest data path will be checked against the fastest clock path to model the worst-case scenario. For hold, the fastest data path is checked against the slowest clock path. A path is considered a data path if it is not part of the clock network or a power supply. The difference between the arrival time of a data path and the required time is known as slack. Positive slack means that the path meets its timing goal, negative slack indicates a timing violation.

In high-speed designs it is common to have latches that are 'transparent'. In these cases, when data arrives between the leading and trailing edge of the clock, the path tracing will continue through that latch until it reaches an output port or another non-transparent timing element. If the data arrives after the closing edge of the clock it is flagged as a violation. However, to test for possible violations downstream from this latch, NanoTime adjusts the data arrival time back to the required value and continues tracing the path forwards. This allows the downstream paths to be checked in the same analysis run without having to iterate on repairs to the failing path. This feature is known as "latch error recovery". When this occurs, the impacted latches will be appropriately marked in NanoTime's path reports.

If a data path originates from a clock signal of a timing element, NanoTime will account for the common clock portion of the data path and the clock signal it is being checked against so that the timing check is reasonable and not overly pessimistic. This automatic adjustment of slacks to account for variation in the arrival of signals re-converging from a common point is known as Path Based Slack Adjustment (PBSA). This is similar to the Clock Re-convergence Pessimism Removal (CRPR) concept that is used in cell-based timing analysis.

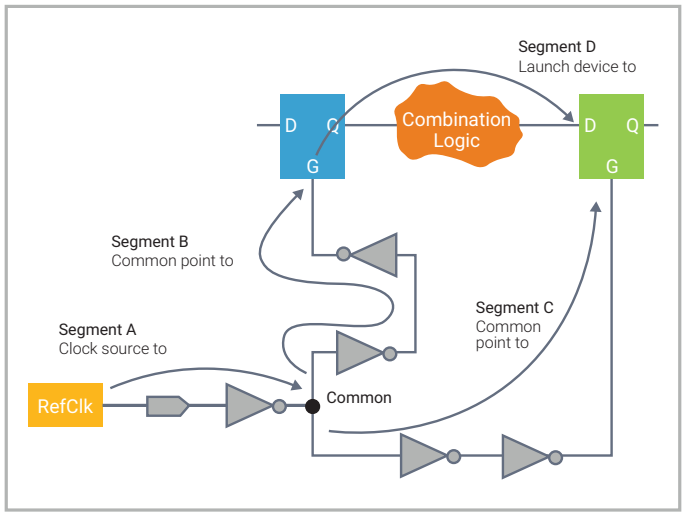


Figure 5: Path Based Slack Adjustment (PBSA)

In addition, to recognizing the clock network, NanoTime must identify all the timing elements in the design. Any transistor cluster that has both a data input and a clock input will be considered a timing element. How timing is checked at each timing element can vary depending on the element type such as a latch, gated clock, flip-flop or domino logic stage. Hence NanoTime needs to identify the type of timing element each transistor cluster represents. NanoTime has built-in recognition of a wide variety of common circuit elements. It also supports user-defined elements via topology marking commands.

Clock Network Analysis

To perform timing checks, NanoTime must identify and analyze the clock network. The primary ports for each clock and their ideal waveform shape are user specified using SDC commands. NanoTime will then propagate the clock property and waveform through all the relevant transistor clusters until it reaches a timing element. For certain timing elements such as clock gating elements, the clock tree will be propagated forward unless explicitly blocked e.g. by a user defined logic condition. For other timing elements such as a latches or flip-flops, the clock tree propagation will stop at the clock pins of that element. As the clock tree is propagated forward it is delayed accordingly to ensure that clock skew and logic inversion is accounted for.

Clock trees are normally traced by breaking them into clusters, simulating the clusters individually, and deriving arrivals by combining the observed delays. This fast method is accurate enough for most clock trees. When the clock contains interactions between parallel paths within the network or complex generation circuitry using feedback loops, simultaneous switches, latches, flip-flops, choppers, etc. it is often desirable to simulate the entire clock network as a single entity, fully capturing the complex interaction of the elements of the clock network. To address this, NanoTime provides the Dynamic Clock Simulation (DCS) feature that automatically creates the clock network as a single sub-circuit and uses dynamic simulation to accurately model the full network and determine the exact waveforms. The results of the dynamic clock tree simulation are imported back into NanoTime to be used for all the static timing checks.Topology Recognition

To perform the correct timing checks for each timing element and to reduce the number of possible false paths (paths that are logically not possible), NanoTime will recognize and classify transistor clusters into known circuit types. For example, invertors are marked to ensure that the logic relationship between inverting signals is maintained. Similarly, for transmission gates where if the relationship between the N and P transistors wasn't known, the worst-case delay would be pessimistic due to passing a logic 0 via the P when the N transistor is off. By knowing that the transistors form a transmission gate, NanoTime will ensure that both transistors are on simultaneously. The following is a list of many of the circuit types that NanoTime recognizes.

For those structures that NanoTime doesn't explicitly recognize, the user can define their own structures and define the timing checks

- Inverters
- Transmission gates
- Flip-flops
- Latches
- Muxes
- Clock-gating structures
- Weak pullups
- Domino precharge
- Turn-off structures
- Register file structures
- Static RAM cells
- Feedback structures
- Xor structures
- Three-state logic

to apply at that structure. NanoTime supports two methods of marking a structure, by sub-circuit name or by transistor pattern. The former is typically used when NanoTime is run with a hierarchical netlist where the RC parasitics have been back-annotated via a separate dspf or SPEF file. The later method is most often used for flat extracted netlists.

By automatically recognizing most structures commonly used in custom design, NanoTime significantly reduces the effort required to achieve the highest accuracy. Once topologies are recognized, signal flow through the circuit structures will be determined. NanoTime uses a built-in algorithm to automatically detect the signal flow direction. This algorithm accounts for surrounding logic to automatically spot mutual exclusion and prevent erroneous signal flows, thus preventing unidirectional devices in pass-gate logic from being identified as bi-directional. This greatly improves the accuracy and performance of the analysis. If a transistor is left as bi-directional, NanoTime may attempt to determine delays along paths that may not actually occur, resulting in longer runtimes and potentially overly-pessimistic delay estimations. For transistors that NanoTime cannot automatically determine direction, there are user commands to specify directions manually.

Timing Model Extraction

Transistor- and cell-level static timing analysis need to work seamlessly together to achieve full chip timing verification. NanoTime can analyze and characterize transistor-level blocks and generate an extracted timing model (ETM) for inclusion in full-chip timing by PrimeTime. The ETM model uses the standard Liberty format (.lib) and includes composite current source (CCS) models for timing, composite current source models for noise (CCSN) and Liberty Variation Format (LVF) for process variation. The model also contains pin capacitance, non-linear delay models and timing constraints.

Blocks in a design can have more than one operating mode, for example a normal mode and a test mode.Since the timing requirements of each mode may be different, a complete hierarchical analysis may require extracting a separate timing model for each one. NanoTime can generate models for each mode and then combine them into a single mode-sensitive model. The resulting merged model can be used in a higher-level analysis by setting the operating mode. Merged models can be imported into PrimeTime and Design Compiler.

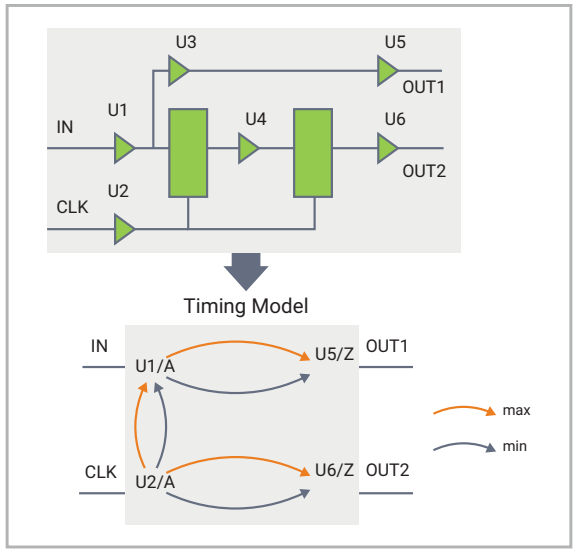


Figure 6: Timing model extraction

NanoTime Tcl Interface

NanoTime has a friendly interactive interface based on the Tcl scripting language. Tcl provides a full range of programming constructs such as flow of control (if-then-else, while loops, for loops, etc.), user variables, collections, lists, procedures, etc. Objects such as nets, transistors, circuit structures such as latches, muxes, timing checks, and paths can be queried, gathered into collections or lists and reported. Designers can easily create scripts to simplify routine tasks and perform custom formatting and checking of NanoTime's results.

Many of the commands in PrimeTime are directly supported by NanoTime, including the use of SDC. Commands specific to custom design and analysis techniques that have no equivalent in PrimeTime are constructed in a way to be consistent with the PrimeTime approach. Many of the reports generated by NanoTime also look like those generated by PrimeTime. The result is that NanoTime's interface is very familiar to users of PrimeTime and the two tools have a very similar look-and feel.

Logic and Timing Constraints

The main goal of an STA tool is to verify that timing expectations are met. Once the clock networks have been established and topologies recognized, NanoTime automatically attaches a variety of timing constraints between the nets of the design. These constraints, when checked during tracing, establish a timing relationship between the attached nets. For example, a setup check verifies that a signal launched by a clock edge arrives at its destination before the capture clock edge. A hold check verifies that the same signal arrives late enough not to be captured by the previous capture clock edge. Other timing checks supported include min pulse-width checks which ensure pulse-driven sequential logic receives valid pulse-clock signals, and domino precharge checks that perform a variety of checks to ensure proper setup and hold conditions in domino logic.

NanoTime uses a path-based tracing technique to calculate and propagate timing information through a design. It is often desirable to exclude certain paths from the analysis, for example if it is known that certain logic conditions required by the path will never be met, or if the designer wants to exclude certain paths from consideration to speed up analysis time and focus attention on a specific region of the design. NanoTime provides several mechanisms to constrain the paths considered during analysis. One of these mechanisms is an internal logic evaluation engine that understands logical relationships between many of the nets of the design. Users can also specify logical constraints on the design between signals or based on the underlying topology structure, for example a multiplexer.

Case analysis is the technique of using logical constraints to constrain the design. Several different logical constraints are available to the user for case analysis. The simplest one allows the user to explicitly set the logical value of a net to one or zero. The user can also apply invert constraints to a pair of nets, asserting that the logical values of the two nets must be opposite. For example, if a logical one is set on net A and an invert constraint exists between net A and net B, then NanoTime will determine that net B must be a logical zero. Other constraints supported include:

- Equal constraints – nets must have the same logical value
- One-hot constraints – exactly one net in a list is logical one, all others must be logical zero
- At-most-one-hot constraints – at most one net in a list is logical one, all others must be logical zero
- One-off constraints – exactly one net in a list is logical zero, all others must be logical one
- At-most-one-off constraints – at most one net in a list is logical zero, all others must be logical on

Logical constraints are applied immediately, i.e. the constraint is passed to the internal logic evaluation engine which propagates logical assignments as far as possible. When a logical value is applied to the gate-pin of a transistor, it effectively turns on or off the transistor. The logic evaluation engine can propagate logical assignments across transistor clusters and completely exclude large sections of the design from consideration during path tracing, reducing runtime and memory while focusing reports on only the region of interest.

In addition to logical constraints, many timing exceptions are available to remove specific paths from consideration or to alert the user when certain paths were attempted. Types of timing exceptions supported by NanoTime include:

- False path exceptions – While NanoTime automatically detects and avoids a wide variety of false paths, the false path exception commands give ultimate control to the user.
- No-check exceptions – used to suppress reporting of specific paths.
- Multi-cycle path exceptions – used to control hold checks across multiple clock cycles on transparent paths.
- Phasing exceptions – used to control setup and hold checks across clock phases.

Signal Integrity Analysis

As geometries shrink, coupling capacitive between wires grows due to the shrinking separation between wires and the increasing relative heights of the wires. At the same time, loads caused by parasitic wire-to-substrate and gate-to-substrate capacitances shrink as wires become narrower and transistors become smaller. For advanced nodes, analyzing the impact of signal integrity (SI) on timing is essential because crosstalk from adjacent signals can change path delays by as much as +/-25%.

NanoTime identifies potential victim and aggressor nets and accounts for both the speeding up and slowing down of signals based on their worst-case switching alignment. For accurate analysis of the signal integrity induced delay change, NanoTime considers the arrival times of the aggressor nets to ensure that the victim and aggressors can switch together. To account for changes in aggressor arrival times caused by changes to each victim's arrival time, NanoTime will iterate until the timing windows of the victims and aggressors converge or a user set limit is reached.

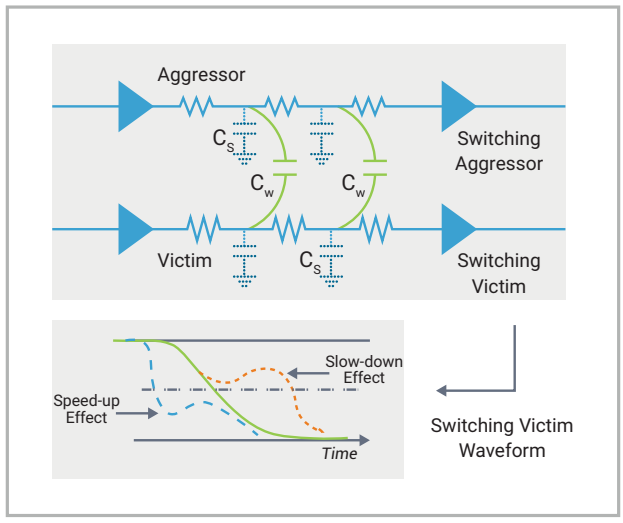


Figure 7: Signal integrity impact on delay

Crosstalk can also induce noise glitches that might cause fatal functional failures if they propagate and disturb stored logic states in latches or flip-flops. When aggressor nets switch while the victim is quiet, a noise glitch is injected onto the victim net. It is essential to measure all potential noise injections to ensure that they will be attenuated and not amplified by the downstream path. NanoTime will exhaustively examine all nets and will calculate the worst-case possible glitch caused by aggressor switching at their fastest possible switching rate. Any glitch that exceeds noise immunity thresholds will be flagged so that it can be repaired prior to signoff.

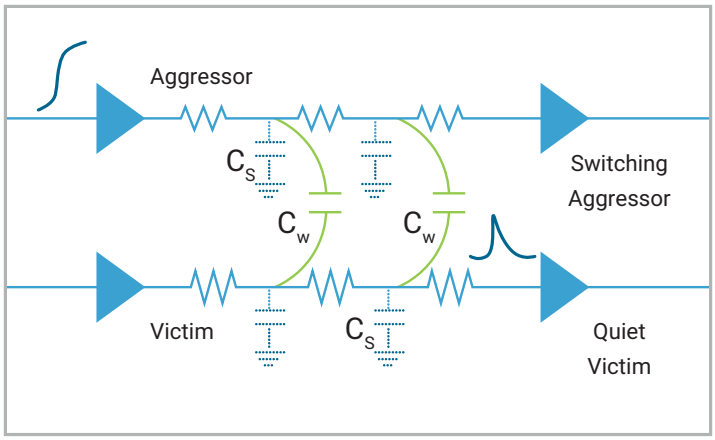


Figure 8: Injected noise caused by aggressor switching

NanoTime will also check for propagated noise by simulating significant glitches that occur at the inputs of a stage to calculate how the noise bump transfers to the output. NanoTime uses the results from the injected noise analysis to determine noise bumps to be applied at the input of the stage. It also enables users to define specific noise aggressions for selective nets. This enables users to verify how much a noise attack can be attenuated or amplified by noise-sensitive stages.

Process Variation Analysis

For advanced designs, managing the impact of process variation on timing is essential to improve yield. Simply derating delays by constant on-chip variation (OCV) factors across the design is too pessimistic for most paths but not pessimistic enough for some others. Understanding how process variation impacts each component within the context of a path enables a much more realistic analysis which helps to optimize a design.

The traditional method of modeling local process variation is to use Monte Carlo simulation with thousands of trails. NanoTime avoids this costly Monte Carlo simulation by pre-analyzing various transistor scenarios to create unique variation coefficients. These coefficients are then using to calculate the impact of process variation on the delay and slew of each path stage and statistically accumulates the impact over the complete path. NanoTime's variation analysis can be performed in typically less than 50% of the runtime of a nominal NanoTime run, unlike Monte Carlo simulation where the run time increases with the number of samples. The variation results can also be written into the timing model as Liberty Variation Format (LVF) for delay, slew and timing constraints.

Embedded Memory Design Analysis

Embedded memory designs present unique challenges for transistor-level static timing analysis due to the large size of the memory array and the analog nature of the sense amplifier. NanoTime addresses these challenges by identifying the memory array (bit cells, word line, sense ampifier, and mulitiplexer circuitry) and using a direct interface to distributed dynamic simulation (using the HSPICE or FineSim) to determine the worst-case paths through the array. The results of the dynamic simulation are then seamlessly integrated and reported along with the built-in analysis of the remaining circuitry (such as address decoder circuits). Furthermore, special SRAM setup/hold timing checks (Figure 9) are also performed as part of the static timing analysis process to ensure accuracy.

Optionally for faster throughput, NanoTime allows the memory array to be skipped from the analysis. In this mode, the memory periphery (including the sense amplifier, write drivers, read muxes etc.) is fully analyzed along with all the associated timing checks.

Memory Setup Checks	Memory Hold Checks
wordline off before precharge on	wordline on after precharge off
wordline off before precharge on	write enable off after wordline off
wordline on before read mux on	data valid after wordline off
write enable on before wordline on	precharge on after read mux off
data valid before wordline off	sense amp precharge on after read mux off
read mux on before sense amp enable on	sense amp out precharge on after read mux off
sense amp enable on before read mux off	input valid after write enable off
precharge off before read mux on	sense amp out precharge on after read mux off
sense amp precharge off before read mux on	sense amp enable on after precharge off
sense amp out precharge off before read mux on	sense amp enable on after sense amp precharge off
sense amp enable off before precharge on	precharge on after write enable off
sense amp enable off before sense amp precharge on	
sense amp enable off before sense amp out precharge on	
precharge off before write enable on	
input valid before write enable on	

Figure 9: Memory specific timing checks

Using traditional dynamic simulation, simple checks like those listed above would take a long time to setup and execute without the confidence of completeness. However, with NanoTime, all checks are done automatically, and timing models are generated quickly for full-chip SoC signoff.

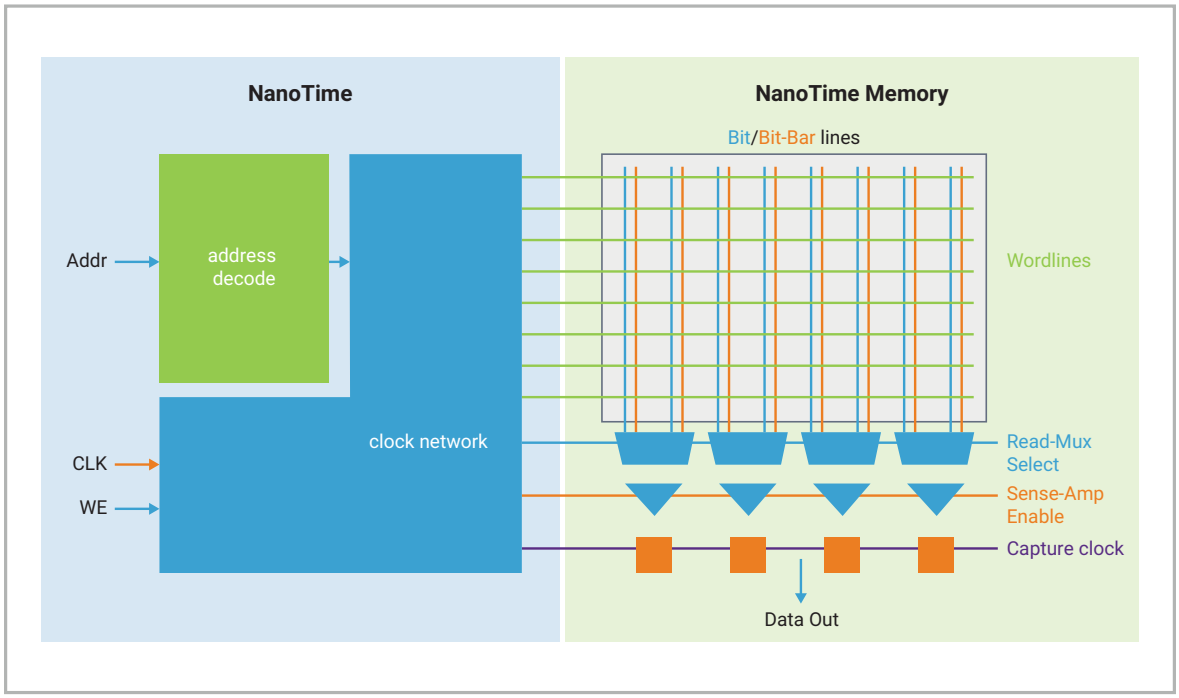


Figure 10: NanoTime's memory option for analyzing paths through the array

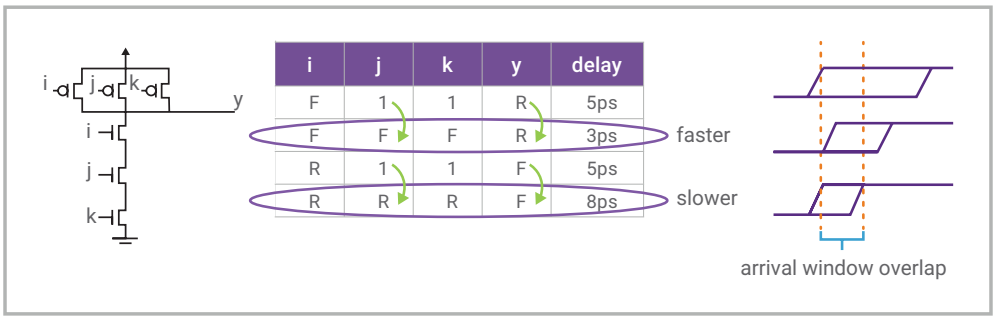
Mixed Level Analysis

NanoTime supports mixed-level analysis oftransistor and cell-level designs. This allows NanoTime to leverage pre-characterized libraries in Liberty format (.lib) for portions of the design and utilize detailed transistor level analysis for the most critical parts. This capability is especially useful for analyzing mixed signal blocks like SerDes and PLLs. In order to prevent the path analysis from traversing through the analog portions of the block, NanoTime permits those portions to be black boxed using a timing model (.lib) or by marking it to be ignored during path tracing. This black-box model model can include just capacitance loading or it can contain delay and/or contraint arcs. The black-boxing of the analog components allows NanoTime to ignore the analog nature of the underlying circuitry and focus on the digital blocks, identifying worst-case paths, performing timing checks and creating a timing model.

The results of this hybrid analysis can be extracted into a block-level timing model and used for synthesis, place and route or by full-chip level timing and noise analysis.

Simultaneous Switching Signals

Static timing analysis typically assumes that each signal switches independently. However, sometimes signals are designed to switch simultaneously to boost performance; in other cases, signals inadvertently coincide. NanoTime can analyze the impact on timing from multiple simultaneous signal transitions. This includes support for full-swing differential circuits and the impact of multiple input switching (MIS). For both differential analysis and MIS, NanoTime can iterate to model the skew between the various switching inputs to ensure the most accurate result. These iterations will also incorporate SI delay effects, ensuring that changes in signal arrival times (timing windows) are fully accounted for.



i	j	k	y	delay
F	1	1	R	5ps
F	F	F	R	3ps
R	1	1	F	5ps
R	R	R	F	8ps

Figure 11: The Impact of multi-input switching on delay

A common case of simultaneous switching is due to differential signals. NanoTime automatically detects cases where the signals are synchronized and accounts for this during its analysis.

Summary

Transistor level static timing verification should be used to complement dynamic timing simulation for custom designs. It allows for a more exhaustive analysis and is the only efficient means to check the impact of signal integrity, multiple input switching and process variation. For cell-based designs, PrimeTime is the recommended signoff tool. For all other designs styles, which the exception of pure analog and RF blocks, NanoTime should be used. By deploying static transistor level analysis, fatal design flaws such as hold time violations, pulse width issues, race conditions and noise induced functional failures can be captured during signoff and costly silicon re-spins can be avoided.

03/14/19.CS324170537-Transistor Level Static Timing Analysis with NanoTime® White Paper.

Pub: March 2019