SYNOPSYS®
Predictable Success

Static Timing Verification of Custom Blocks Using Synopsys' NanoTime® Tool

September 2009

Author
Dr. Larry G. Jones,
Implementation
Group,
Synopsys, Inc.

Introduction

With the continued evolution of processes reaching levels below 90-nanometers (nm), nanometer effects are having a significant and growing impact on delay and require a number of changes in the approaches used to predict delays. Delays are now dominated by large and complex networks of parasitic devices present on the interconnect. In an accurate extraction, the number of parasitic devices can be orders of magnitude larger than the number active devices, requiring longer runtimes from analysis tools. Coupling between unrelated parasitic networks can result in crosstalk and introduce significant delays that must be accounted for. This paper introduces NanoTime, Synopsys' next-generation static timing verification solution for custom designs and is intended for use on designs at 90-nm and below to specifically addresses the complex issues occurring in nanometer design.

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 or not 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 the dynamic simulation approach, 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 subcircuits within the block and use graph analysis techniques to quickly seek out the slowest and fastest paths in the block. The result is that an STA tool can typically find all timing violations in a block in a fraction of the time it would take a dynamic circuit simulator.

Synopsys offers the NanoTime analysis tool for custom designs and the PrimeTime® tool for gate-level designs to address these STA challenges.

Verification of Custom Designs

The approaches required for timing verification of custom designs differ significantly from those required for the gate-level design approach. Gate-level STA relies on an upfront process called characterization that uses circuit simulation to determine delays for the individual gates in the library under a pre-determined variety of input slopes and output loads. The resulting delay models are then used during the trace phase of the gate-level tool.

Circuit styles used by custom designers often include the use of structures that can prove to be a challenge to accurately characterize for delay without full knowledge of the surrounding circuitry. Examples of circuits that can be difficult to characterize in isolation include dynamic logic such as domino, pass-transistor logic, tri-state, transparent latches, register files, etc.

NanoTime's approach to verifying custom designs is to combine circuits that need to be considered as a unit for accuracy purposes and determine delays during tracing by simulating circuits in their exact input transition and output loading context.

Introducing the NanoTime Static Timing Analysis Tool:

NanoTime includes significant enhancements to the following important features from PathMill®, the previous generation custom design STA tool offered by Synopsys:

- NanoTime was designed to accept the same netlist, parasitic, and technology formats that PathMill accepts.
- A path-based tracing approach is used to propagate delay information through the design. Although the underlying trace engine is based on more advanced algorithms than PathMill, both are path-based
- An internal delay simulation engine is used to dynamically compute delays of circuits as they are encountered during the trace. This means that delays are determined using simulation on actual slopes and actual loads rather than derived from pre-characterized tables
- An embedded dynamic fast-spice simulation engine is available for obtaining high-accuracy for complex clock networks or circuits with near-analog behavior
- Automatic circuit recognition of common structures that require special attention, including domino logic, muxes, latches and register files
- Detailed analysis of side-branch logic used to determine the input-logic that will have the most impact on circuit delay

Additional features introduced in NanoTime:

- Automatic calculation of the delay impact caused by cross-talk
- Automatic adjustment of slacks to account for variation in the arrival of signals re-converging from a common point
- A seamless hierarchical flow with PrimeTime, including support for Synopsys Design Constraints (SDC) for the analysis of designs containing both custom and gate-level blocks
- An underlying logic engine that automatically avoids false-paths that introduce undue pessimism in the analysis results
- NanoTime is an interactive tool. Designers can read in a design, modify the setup, and make changes to clock definitions, generate detailed reports and query the tool to ensure the design is appropriately configured before initiating a trace
- Multiple analyses are allowed in a single session, boosting designer productivity. NanoTime significantly increases designer productivity. Not only does the tool enable interaction, but it also allows multiple analyses to occur in a single session
- NanoTime uses new algorithms that result in a five-fold increase in speed

NanoTime Features

User 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, latches, muxes, timing checks, and paths can be queried, assigned properties, gathered into collections or lists, and reported. Designers can easily create scripts to simplify routine tasks.

Many of the commands in PrimeTime are directly supported by NanoTime, including the use of Synopsys Design Constraints (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 and many of the reports generated by NanoTime also look similar to 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.

Phases

A typical run of the NanoTime tool is accomplished in phases. Each phase represents a specific focus on certain tasks. For example, in the netlist phase, the designer reads in and links the design netlist and uses certain commands to condition the netlist for further processing. A session consists of five major phases:

- Netlist phase – the design is read in and prepared for further processing
- Clock propagation and topology recognition phase – clocks are identified and common or critical circuits are recognized
- Timing constraint specification phase – timing constraints such as setup and holds are imposed on the design
- Path tracing and simulation phase – the design is partitioned into subcircuits, paths are traced through the design, and delays are calculated. During this phase longest and shortest arrival times are propagated and timing constraints are checked
- Analysis reporting phase – after tracing is completed, the designer can request generation of reports and query the tool for specific information

Interactive phases guide the designer to correct setup and analysis as well as increase overall productivity by simplifying the tasks required.

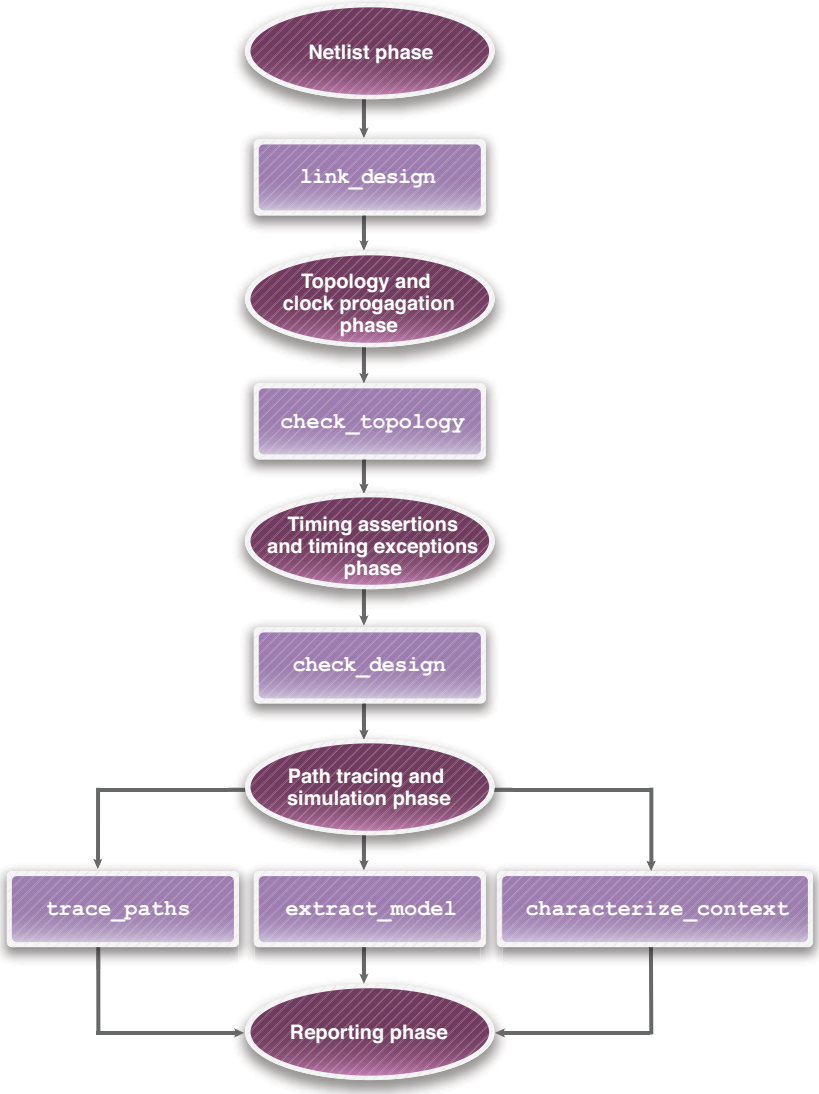

Figure 1. Timing analysis flow based on NanoTime phases

Netlist and Technology Management

NanoTime recognizes netlists in a wide variety of formats (for example: SPICE, Verilog, and EDIF). NanoTime also accepts industry standard formats for parasitic and device information in the Standard Parasitic Exchange Format (SPEF) and the Device Parameter Format (DPF).

Because delays are determined by dynamic simulation, technology information that describes the transistor behavior is required. Transistor models can be expressed as either standard BSIM3/BSIM4 SPICE models or as a technology file that captures device properties in tabular form.

As geometries shrink, the impact of parasitic devices on delay has become more important. Use of extractor tools such as Synopsys' StarRC™ tool to determine the parasitic resistors and capacitors in a design is now a standard part of design flows. The number of parasitic devices extracted is usually much larger than the number of transistors in a design, often one or two orders of magnitude more. Since the runtime of dynamic simulation is sensitive to the total number of devices, not just the number of transistors, the growth in parasitics has had a significant impact on the speed of simulation based transistor-level analysis tools. One way this problem is mitigated is by applying parasitic reduction techniques prior to the analysis tool to reduce the number of parasitic devices while maintaining an approximation of the delay impact caused by the parasitics. However, it is difficult to know the extent of reduction that is possible for a given accuracy target

without knowing the delay context that the parasitics are situated in. To address this, the NanoTime tool has an internal parasitic reduction capability that uses its knowledge of delay to guide the reduction process in a controlled, delay-accurate manner. While the internal parasitic reduction defaults to an accurate reduction balanced against runtime, the designer is given controls to allow more aggressive reduction in the early stages of a project to provide quick delay estimates, or less aggressive reduction to improve simulation accuracy on tricky circuits. The reduction controls are in the hands of the designer.

Other useful features offered by NanoTime that support analysis of advanced technologies include:

- Multivoltage – supports the use of multiple power supplies
- Programmable switching thresholds – controls to independently adjust trip points for slew and delay
- Netlist objects – object level access to netlist elements. Transistors and nets can be used in lists and collections and can be accessed and manipulated in Tcl scripts

Clock Specification and Analysis

NanoTime provides a complete set of SDC compliant commands to simplify the specification of origin and properties of clocks in the design. Properties of clocks include the period, waveform, latency, uncertainty, and transition time. The supplied clock information is used to automatically trace clock networks and identify clocked subcircuits within the design. Appropriate setup and hold constraints are automatically applied to the clocked subcircuits and are checked during path tracing to identify timing violations.

NanoTime supports a variety of clock types and clocking concepts including:

- Propagated clocks – automatically identifies complete clock networks originating from a defined source and propagating through inverters and gating logic. NanoTime automatically computes clock latency and transition times with SPICE-like accuracy for the propagated clock signals
- Ideal clocks – uses propagated clocks by default, however, it also allows the use of ideal clocks where latency and transition times for the clock signals in a clock network may be directly annotated instead of computed
- Generated clocks – supports the use of clocks derived from a master clock. This is commonly used to create clocks with increased frequency (multiply-by) or decreased frequency (divide-by). It is also possible to create new asymmetric clocks with edges that are timed to specific edges of the master clock. Generated clocks can also be used to support the specification of pulse-clocks
- Clock uncertainty – uncertainty refers to the variation in arrival time of each edge of the clock. Designers can specify the amount of variation between edges of the same or different clocks
- Clock net merging – clock networks often include parallel drivers. Merging insures that these drivers share common characteristics and switch in unison

Topology Recognition

In order to accurately analyze a design, a transistor-level timing-analysis tool must be able to intelligently partition the design into subcircuits whose devices are required to be analyzed together or require specific techniques. Accurately partitioning the devices requires not only an understanding of the properties and direct connectivity of the individual devices, but also their collective topology and context, along with the logic in the immediate region. For example, accurate analysis of a latch requires simultaneously analyzing all of the transistors in the subcircuits that the latch is composed of and identifying key elements of the latch such as the data and clock inputs and the feedback devices. It may also require inclusion of surrounding logic that feed the controls of the latch. In another example, correct analysis of a pass-transistor mux may require an understanding of the direction of signal flow through the transistors within the mux along with inclusion of inverters controlling the select lines.

NanoTime also includes an advanced recognition engine capable of automatically recognizing most common topologies used in custom design that require special attention. These cover a wide variety of structures from different circuit styles, including, among others:

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

By automatically recognizing most structures commonly used in custom design, NanoTime significantly reduces the effort required to achieve the highest accuracy.

Recognized topologies are accessible objects in the NanoTime Tcl interface. This means that they may be manipulated as part of lists and collections and can be examined or modified. A wide variety of reports can be generated that list and count the various topologies and the designer can apply specific techniques or try what-if experiments on specific topologies.

Once topologies are recognized, signal flow through the circuit structures can be determined. Previous techniques used by other analysis tools often had difficulty analyzing circuits using pass-gate logic techniques commonly used to implement mux and decoder structures. The result is that many transistors were erroneously identified as bi-directional. This can have a significant negative impact as the analysis tool may attempt to determine delays along paths in the subcircuit that cannot actually occur, resulting in long runtimes and overly-pessimistic delay estimations. As a work-around designers must manually inspect reported bi-directional devices to verify and correct the directionality.

NanoTime uses a significantly improved algorithm to automatically detect the signal flow direction. One improvement in this algorithm takes into account 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.

Constraining the Design

The main goal of an STA tool is to verify that timing expectations are met. This is accomplished by laying down timing constraints on the design, determining signal arrival times, and checking to ensure that the constraints have been 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 in order to speed 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 is capable of evaluating logical relationships between nets of the design. By laying down logical constraints on the design, the user can establish these relationships. In addition, many net-to-net logical relationships are established automatically, based on the switch-level logic of the circuitry.

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 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 the user has set a logical one on net A and an invert constraint 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 one

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. By including the switch-level evaluation of subcircuits, the logic evaluation engine can propagate logical assignments across subcircuits 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, a number of timing exceptions are available to manually 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 – Automatically detects and avoids a wide variety of false-paths, however, these exceptions give ultimate control to the user
- No-check exceptions – used to suppress reporting of specific paths
- Multi-cycle path exceptions – used to control setup and hold checks across multiple clock cycles on transparent paths
- Phasing exceptions – used to control setup and hold checks across clock phases

Timing Analysis

Path-Based Exploration

As stated earlier, static timing verification is the process of determining that a given design can be operated at a specific clock frequency without timing errors. To verify the performance and check for errors NanoTime explores the design calculating and propagating worst-case arrivals. Worst-case does not necessarily mean longest delay. For example, to check a setup constraint, it is necessary to determine the earliest time the opening edge of the clock can arrive, but the latest time the data can change. Similarly, to check a hold constraint, it is necessary to determine the latest time the closing edge of the clock can arrive and the earliest time the data can change. It is therefore necessary to determine both the earliest and the latest time a signal can arrive.

NanoTime uses a path-based approach for exploring the design. This means that it attempts to explore every path from a design input (referred to as a source) to a design output or timing check (referred to as a sink). Since generally there are too many paths in a design to explore all paths, NanoTime prunes searches that it determines will not lead to a faster or slower source-to-sink path.

There are several advantages to using a path-based approach. First, many common false-path scenarios can be avoided by using the internal logic evaluation engine to exploit the knowledge of the transitions that occurred on the path being explored. Second, more accurate delays can be determined by using knowledge of transitions of side inputs in cases of reconvergent fan-out, a very common scenario in custom design. Third, complex clocking schemes involving dynamic circuits and transparencies are more accurately accounted for.

Pruning non-productive paths during a path-based trace is a key technology in NanoTime. Without pruning, path-based tracing would take time exponential in the number of transistors in the design, making it impractical for all but the smallest circuits. Careful pruning can reduce that time to nearly linear making it possible to explore designs containing millions of transistors in reasonable time. The pruning algorithm represents an accumulation of years of experience applying the technique to thousands of designs, including many high-speed microprocessors.

Paths are collected into a path database as they are found. Usually only one fastest and one slowest path is kept in the path database for each source-sink pair. When a new path is found, it is compared to the previous path with the same source and sink and the worst (fastest or slowest) of the two is kept and the other is discarded. However, it is also possible to keep multiple paths for each source-sink pair by defining limits on the number of paths and the maximum difference from the worst timing found so far. Both limits are extremely useful when trying to make optimization tradeoffs and reducing the time required to repair ‘onion-peeling’ violations in which a potential violation is just below a reported violation.

Simulation based delays

NanoTime uses its internal circuit simulation engine during path-based tracing to compute the delay on a subcircuit with SPICE-like accuracy. Since circuits may be visited multiple times under similar circumstances, NanoTime maintains a cache of previous simulations that it exploits to keep the number of circuit simulations to a minimum. An advantage of creating and maintaining this cache is that it can be used in later traces in the same session to great speed advantage, significantly reducing the total number of simulations required to retrace the design. NanoTime is also capable of saving simulation data created in one session for later use within another session, even after changes to the netlist. The cache reuse technology is capable of automatically determining where the cache may be reapplied, providing a fast, incremental approach to transistor-level design and verification.

Reporting

Paths collected in the path database can be manipulated as objects in the Tcl interface. In addition, there are a number of reporting features for displaying paths in standard formats consistent with PrimeTime to ease the use of both tools in a chip-level flow. NanoTime has a very flexible custom reporting capability that gives the user controls over the selection of fields, field orders, and sort orders. Paths in the database can also be exported as complete spice decks ready for simulation in an external dynamic simulator such as HSPICE.

Support for Hierarchical Flows

Mixed gate and transistor level analysis

NanoTime is the first transistor-level STA tool to fully enable the timing verification of system-on-a-chip (SOC) designs containing both custom and gate-level blocks using a seamless flow with the PrimeTime gate-level STA tool. Following are the benefits offered by a hierarchical timing flow constructed using both NanoTime and PrimeTime:

- NanoTime generates Extracted Timing Models (ETM's) that capture transparencies and can be natively analyzed directly by PrimeTime
- The same delay calculator is used for both NanoTime and PrimeTime for extracting delays from models. This golden delay calculator is also used by other Synopsys tools including Design Compiler® and Astro™ tools. Consistency in treatment of delays is key to achieving timing closure
- NanoTime leverages Synopsys standard formats, generates and accepts constraints and models in .lib, .db and SDC
- NanoTime can be used in both a downward analysis flow and an upward one. Specifically, NanoTime accepts SDC constraints from PrimeTime to express the boundary conditions and to constrain the analysis and extraction of a lower-level model. The resulting extracted lower-level model is expressed in .lib or .db and can in turn be used by PrimeTime in the analysis of a higher-level block
- For complex interactions between custom and gate-level blocks, NanoTime can accept PrimeTime models for hybrid analysis in the context of a containing custom block and the resulting extracted hybrid models can then be imported back into PrimeTime

Model merging

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 different modes may be different, a complete hierarchical analysis may require extracting a model for each timing mode of the block. Rather than maintain multiple models, NanoTime offers the ability to merge these models into a single mode-sensitive model. The resulting merged model can than be used in a higher-level analysis by setting the operating mode. Merged models can also be imported into PrimeTime and Design Compiler.

Context characterization

The context characterization feature captures the timing context of a lower-level block within the timing environment of a containing higher-level block. The resulting context includes clock waveform information, input arrival times, output required times, timing exceptions, logic constraints, and capacitive loads. Context characterization can be used to perform timing analysis of the lower-level block while observing the chip-level timing constraints.

Dynamic Simulation

Under normal conditions an internal circuit simulator is used by NanoTime to analyze and determine delays for subcircuits in the design. In certain scenarios, such as complex clock networks or circuits with near-analog behavior, it is desirable to boost accuracy by using the full-power of a dynamic simulator. To accommodate this need, an embedded version of the NanoSim® fast-spice simulator is used.

Dynamic clock simulation

Clock trees are normally traced by breaking them into subcircuits, simulating the subcircuits individually, and deriving arrivals by combining the observed delays. While this fast method is accurate enough for most clock tree situations, 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 subcircuit and uses the embedded NanoSim

engine to accurately determine the waveforms of the clock network. Use of the DCS feature is very simple, the user merely turns on the feature and NanoTime automatically sets up the simulation, starts the embedded simulator, and then back-annotates the results on the clock network.

Dynamic Delay Simulation

NanoTime's Dynamic Delay Simulation (DDS) is a powerful feature that allows the user to carve out specific regions of the design and send these to the embedded NanoSim engine to accurately determine delays within the region. This feature is particularly useful for circuits with complex interactions between multiple signals such as occur in RAM sense-amps or certain simultaneous-switching scenarios where timing is tricky to analyze and especially critical.

NanoTime provides flexible controls over the application of DDS. The designer marks regions of the design requiring dynamic simulation and provides (if required) any special controls over the simulation that is desired. Vectors for side-inputs to the region can be automatically generated or supplied by the user.

When a region requiring DDS is encountered during path tracing, the NanoSim run consisting of the region, its input vectors and the simulation controls are automatically configured, and the embedded NanoSim simulator is invoked, the required delays are extracted and lastly the results are incorporated into the trace.

Advanced Features

Signal Integrity – Crosstalk Delay

As geometries become smaller, the capacitive coupling that occurs between adjacent parallel 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. The result is that the impact on delay of capacitive coupling becomes significant at 180-nm and below.

Crosstalk delay effects between two coupled nets occur when the transition time of a victim net is impacted by a nearly simultaneous transition on an aggressor net. If the aggressor switches in the opposite direction as the victim, the coupling capacitor acts as an additional load and can cause the transition on the victim net to occur later. If the aggressor switches in the same direction as the victim, the coupling capacitor acts as an additional driver and can cause the transition on the victim to occur earlier.

As mentioned, the crosstalk delay effects occur when the aggressor and victim nets transition at nearly the same time. If the aggressor switches too early or too late, it will not impact the timing of the transition of the aggressor. In addition if the aggressor slew is faster than that of the victim, the impact of the capacitive coupling is maximized, if the aggressor slew is slower than that of the victim, the impact is minimized and may even be negligible. Another consideration is that large nets may be victims of many aggressors and the relative transition times and slew rates of each influence the crosstalk-delay impact on the victim.

In summary, the overall timing impact of an aggressor on a victim depends on a number of factors:

- The relative amount of cross-coupled capacitance between the aggressor and the victim
- The direction of the aggressor's transition compared to the victim's transition
- The relative switching times and slew rates of the aggressor and the victim
- The combination of effects from multiple aggressor nets on a single victim

Considering these factors, it's clear that signal timing influences the impact of crosstalk delay and crosstalk delay impacts signal timing. For this reason, NanoTime provides a concurrent static timing analysis and signal-integrity solution.

NanoTime takes an iterative refinement approach: With each iteration, signal arrivals and slews are simultaneously calculated using path tracing while determining delay impact due to crosstalk. Initially aggressors are pessimistically assumed to switch sometime within an infinite arrival window so that they can have maximum impact on victims. After the first iteration, arrivals have been established for all nets, and the

windows can be adjusted to reflect each net's earliest and latest possible arrival. The impact of this is that the arrival windows can shrink dramatically. Since simultaneous transitions of aggressors and victims can occur only if their arrival windows overlap, the effect of shrinking windows is that fewer aggressor-victim pairs have overlapping windows reducing pessimism and providing improved accuracy with each iteration. Experience has shown that convergence is reached within only a few iterations and, in fact, two iterations usually provide good results in a reasonable amount of time.

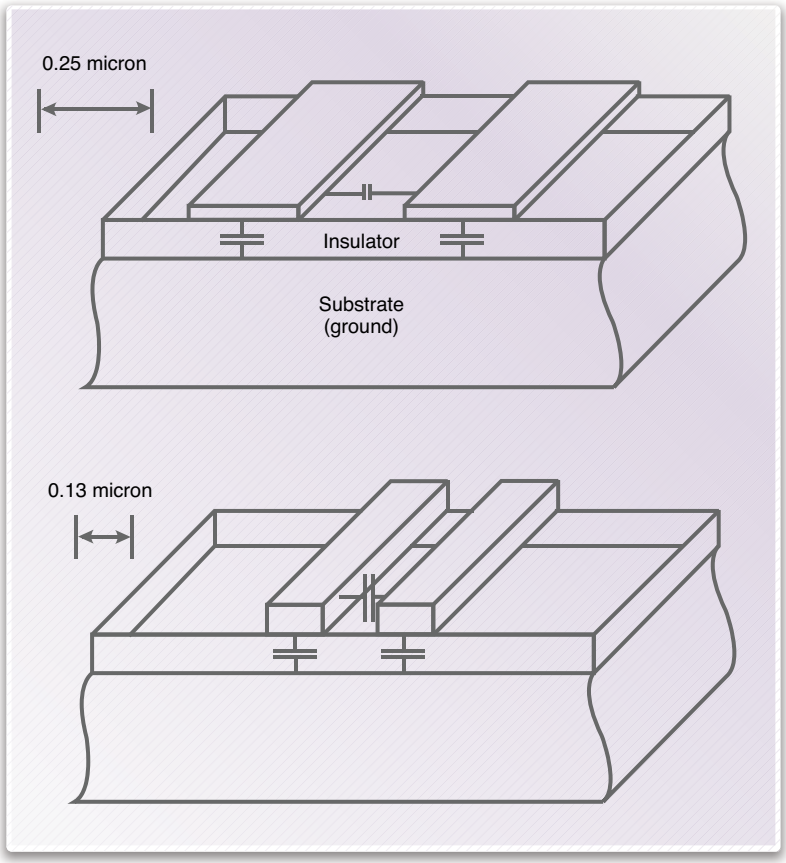

Figure 2. Relative importance of coupling capacitance grows with shrinking geometries

Path-Based Slack Adjustment

The slack of a timing path is the amount of time by which a violation is avoided. Variation in the arrival times of a clock edge (referred to as clock uncertainty) can reduce the amount of slack on a path causing an increase in timing violations. For example variation on a common clock edge arrival can result in the violation of a setup check caused by the late launch of data and a subsequent early capture.

One approach to assessing the impact of clock uncertainty is to simply adjust delays at the launch and capture points by the worst-case skew between the appropriate clock edges. However, large clock trees may have a significant variation in clock arrival times across the chip. Using the designs worst-case clock skew value between the launch and capture points of every path can lead to undue pessimism since the worst-case skew is much larger than the typical skew. The difference between the actual skew and the worst-case can be even more pronounced when the launch clock and capture clock share a large common segment in the clock tree.

NanoTime's Path-Based Slack Adjustment (PBSA) feature gives the user a powerful capability for accurately assessing the impact of clock uncertainty due to variation while avoiding pessimism. Instead of only adjusting delays by a global worst-case skew, adjustments can also be made based on the path length (in number of gates) from a common point. Skew adjustments are determined by partitioning paths into four distinct types of path segments:

- Common segment – from the source of a clock to the last common point in the clock tree before the branch
- Launch segment – from the common point to the clock node at the launch point
- Capture segment – from the common point to the clock node at the capture point
- Data segment – from the output node at the launch point to the input node of the capture point

Parameters are provided that control the amount of adjustment applied to each segment and the impact of the segment path length on the adjustment.

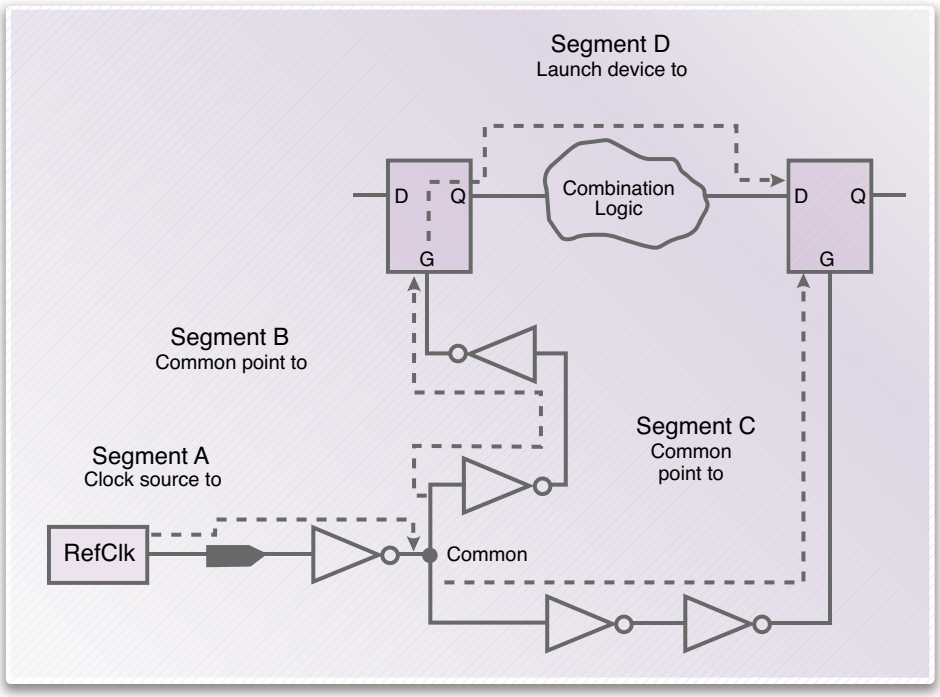

Figure 3. Segments considered in NanoTime's path-based Slack Adjustment feature

Conclusion

NanoTime is Synopsys' next-generation static timing verification solution for custom designs at 90 nm and below. NanoTime provides the designer with accurate delay analysis in response to the increased challenges caused by nanometer effects such as crosstalk delay. Faster runtime, an interactive user interface and a seamless hierarchical flow with PrimeTime significantly enhance the throughput of analysis leading to quick turn-around and an overall increase in the level of verification and confidence.

SYNOPSYS®
Predictable Success
Synopsys, Inc. • 700 East Middlefield Road • Mountain View, CA 94043 • www.synopsys.com