International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(22s) 2026 pp. 1273-1279
https://doi.org/10.70917/ijcisim-2026-5518
Article

Logic-Correlated Static Timing Analysis

Sagar Mallik

Independent Researcher, USA

Abstract: Traditional Static Timing Analysis (STA) has largely remained functionally agnostic. Industry-standard STA tools generally fail to resolve functional correlations across multiple levels of logical gating. Functionality-aware STA reduces the analytical burden, but standard flows must enable all timing arcs because arrival delay differences across arcs can cause functional race conditions, manifesting as functional glitches and crosstalk. These glitches and crosstalk-induced delays significantly impact both timing and power integrity. In addition, arc-based STA introduces excessive pessimism and computational overhead during timing updates. In sub-10nm technologies, where timing signoff is sensitive to even minor pessimism, more refined methods are required. A block-based STA methodology is presented that works alongside arc-based analysis to reduce pessimism in timing, crosstalk, dynamic power, and EMIR analyses. The method captures arrival times at all input pins of combinational gates or logic building blocks (LBBs). By evaluating the relationships between these arrival times and the functional behavior of the logic, the engine dynamically determines which arcs should be enabled for accurate edge propagation.
Keywords: Logic-Correlated STA, Logic Building Blocks (LBBs), Static Timing Analysis (STA), Crosstalk Analysis, Dynamic Power Integrity, Timing Path Optimization

1. Introduction

Static Timing Analysis (STA) forms the basis of testing the timing characteristics of a digital circuit. This analysis offers a way of assessing delay times of signals in an objective manner without necessarily conducting simulations. The main concept behind static timing analysis revolves around representing a logic gate using the various timing arcs of that gate. Fig. 1 illustrates an AND gate with its physical arcs. Each physical arc comprises two logical transitions, rise-to-rise and fall-to-fall. Consequently, a standard two-input AND gate is characterized by four distinct timing arcs. These arcs are defined within the library model of the cell; their existence is typically independent of the state of other inputs. Certain library cells utilize sdf_cond (SDF conditions) to model arc delays based on the logical state of side inputs. When gates are connected in a cascade, the number of potential timing paths increases multiplicatively as STA propagates rise and fall edges through every available arc. Early contributions emphasized the need to capture statistical variation sources in VLSI circuits [1]. By extending max-operation formulations, it became possible to account for correlated delay distributions, improving accuracy in deep sub-micron technologies. Subsequent work highlighted that ignoring inter-arc dependencies leads to pessimistic timing margins [2]. These pioneering works laid down the mathematical groundwork for statistical STA, thus paving the way for scaling techniques that could address complicated correlation phenomena.

Variability plays an important role in achieving timing closure at the gate level. Research on statistical STA to model the behavior of logic gates has shown that delay distributions should be considered from the gate level instead of abstracting from paths [3]. Predictive methods have been developed accordingly. Prediction-based multi-corner STA using machine learning reduced closure time by predicting the delay at each corner, making it possible to achieve quicker sign-off in SoC design [4]. These are a few examples that highlight the development of STA from deterministic arc models to statistical and predictive models.

Conventional STA flows often enable all timing arcs indiscriminately, leading to excessive pessimism. For example, if a fall edge is guaranteed to arrive earlier than a rise edge at a specific stage, downstream timing paths

originating from the suppressed transition can be disabled. This motivates functionality-aware STA, where relative arrival times are evaluated dynamically to prune infeasible arcs. Such pruning reduces computational overhead and mitigates pessimism in timing closure.

As per STA, deterministic timing arcs have become progressively sophisticated due to the incorporation of predictive computation. STA designs that incorporate considerations for manufacturing and environment variability are also considered adaptive. The block-based STA technique analyzes arrival times at every input of a combinational circuit. By dynamically determining which arcs should be enabled for edge propagation, pessimism in timing closure is reduced while accuracy is maintained. The concept builds on statistical models [1], [2], extends gate-level evaluation [3], leverages predictive learning [4], and resonates with broader static evaluation applications. Through this integration, computational overhead and functional pessimism inherent in conventional STA flows are addressed.

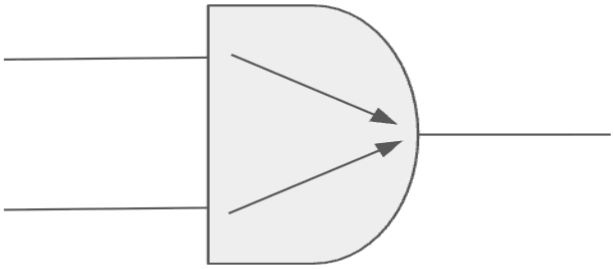

Fig. 1. Timing arcs in a two-input AND gate

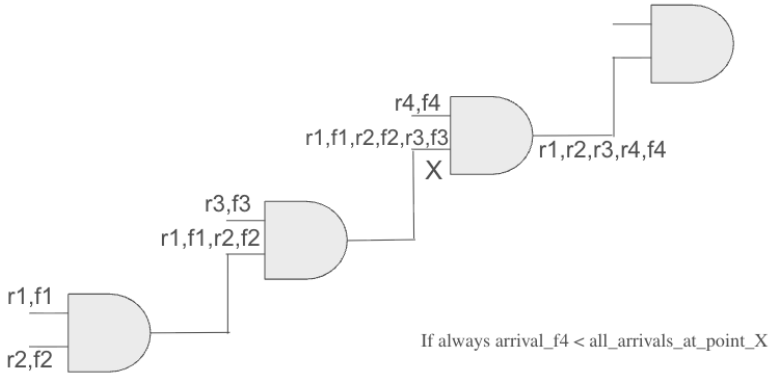

Fig. 2. Timing path propagation in cascaded logic

2. Case Study 1

Fig. 2 illustrates a cascaded structure of AND gates and their associated timing arcs. By default, Static Timing Analysis (STA) evaluates both rise and fall arcs at every input. For a gate with N inputs, this results in $2(n+1)$ potential edges at the output of a single stage. Extending this to the n-th stage of two-input AND gates, the complexity can reach $2(n+1)$ edges. An exponential increase shows how path explosion can occur for timing analysis, leading to the creation of an extremely hard-to-manage graph. The effect of the above discussion has a direct implication on design closure. Timing analysis engines have to trace all the paths, regardless of their feasibility. This leads to overly conservative timing margins and unnecessarily pessimistic analysis cycles. As integrated circuits scale in complexity, this exponential expansion of paths becomes a bottleneck for both accuracy and efficiency. Functional pruning

provides a mechanism to address this challenge. By enforcing logical conditions at specific stages, redundant arcs can be removed. In Fig. 2, arcs $f1$, $f2$, and $f3$ at the output of the third stage are eliminated once functional and delay constraints are applied. This approach eliminates many paths and prevents exponential growth as well as simplifies the timing graph. The idea behind this approach is simple: if transitions will definitely happen earlier or will be logically disabled, then propagating them downstream would be redundant and can be eliminated. This saves resources without sacrificing accuracy. Glitching of internal node's signals represents another problem worth mentioning. Node X in Fig. 2 shows how glitches happen. A glitch occurs if input signals are switched at different times, thus causing temporary and unstable output signals. Transient states can occur in combinational clouds; however, they do not affect the correctness of sequential circuits because flip-flop outputs are always stable due to their ability to sample signals during clock edges. Therefore, even though glitches might occur in internal nodes, they do not affect the sequential circuit because glitches are automatically filtered out in such a case. But, this affects dynamic power and crosstalk timing to a great extent in sub-10nm technologies. Another example from FPGAs shows that invariant-based safety analysis also guarantees that no transients are harmful to the system's performance [7]. The advantages of pruning are not limited to improving timing analysis. Transitions, which cause glitches, consume additional dynamic power by charging and discharging capacitances. As the number of infeasible arcs is decreased by pruning, the frequency of errors is also minimized. The combination of both benefits successful timing closure and energy saving- makes pruning an integral part of today's design process. The example above illustrates that pruning is not just an option but a necessity for advanced designs. Without pruning, the explosive growth of paths will make it extremely difficult for timing engines in lower technologies to close cycles. The removal of arcs $f1$, $f2$, and $f3$ exemplifies how functional conditions can simplify timing graphs while preserving correctness. This principle can be generalized to other gate types. OR, NAND, and XOR structures exhibit similar arc multiplicity, and pruning can be applied in the same way. Identifying infeasible transitions based on logical conditions and suppressing their propagation ensures that timing graphs remain tractable across diverse design contexts. It is essential to have automated incorporation of pruning within STA delay calculation and graph building process. Automated tools need to integrate functionality-based pruning strategies capable of determining arrival times and logical values. This integration makes it possible for pruning to yield consistent results regardless of the project's complexity level. The discussion presented in Case Study 1 lays down the groundwork for functionality-based STA. It highlights the role of pruning in controlling the number of paths exponentially growing within designs, the impact of sequential circuits on glitching issues, and the ability of pruning to conserve energy by minimizing the number of transitions. Case Study 2 continues the discussion by exploring the glitching phenomenon further and proving that sequential outputs do not change despite any internal node changes.

Stage (n)	Inputs per Gate	Potential Edges per Stage	Observed Issue
1	2	4	Not an issue
2	2	8	Growth of edges begins
3	2	16	Growth of redundant arcs f_1, f_2, f_3
n	2	$2^{(n+1)}$	Exponential path explosion

Table 2: Path Explosion in Cascaded AND Gates

3. Case Study 2

Let us illustrate this more with a simple scenario shown in Fig. 3. and Fig. 4. There are two registers, "A" and "B." "A" and "B" are "OR"-ed. "B" drives the OR gate through a buffer. The output of the OR gate and "A" are "AND"-ed to drive the output. "M" and "N" are inputs of the OR gate, and "X" and "Y" are inputs of the AND gate.

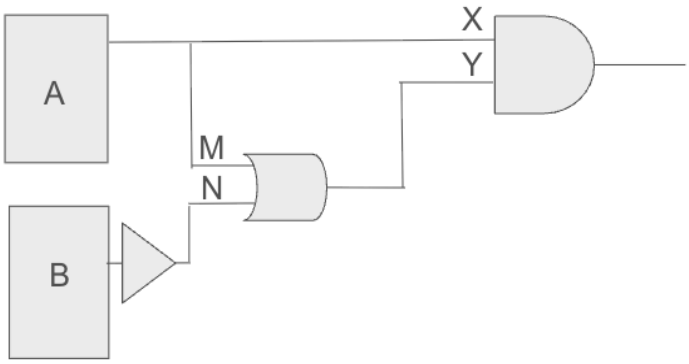

Fig. 3. Functional correlation in complex logic structures

Fig. 4: Showing the actual physical path tracing done by regular STA. For each of Path1, Path2, and Path3, there are two directions, rise and fall. So, there are in total 3 * 2 = 6 timing paths.

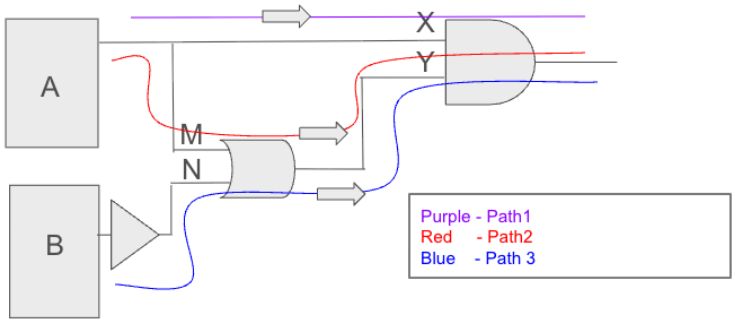

Fig. 4. Showing the Physical Timing Paths in the Circuit in Fig. 3

Path Name	Edge Type	Status After Pruning
Path1r	Rise	Retained
Path1f	Fall	Retained
Path2r	Rise	Retained
Path2f	Fall	Removed
Path3r	Rise	Removed
Path3f	Fall	Removed

Table 3: Path Reduction in OR-AND Circuit

Let's name the paths as Path1r, Path1f, Path2r, Path2f, Path3r, and Path3f. For an OR gate output state (rise or fall) , a decisive input is the "rise" edge. Similarly, for an AND gate output state, the decisive input state is the "fall" edge.

Let's call the arrival time delays for different edges at the pins as

Mr = Rise Arrival Time at M

Mf = Fall Arrival Time at M

Nr = Rise Arrival Time at N

Nf = Fall Arrival Time at N

Xr = Rise Arrival Time at X

Xf = Fall Arrival Time at X

Yr = Rise Arrival Time at Y

Yf = Fall Arrival Time at Y

Let us consider a scenario, which is picked at random and not a special case, where

Always Mr < Nr (I)

Always Xf < Yf (II)

As can be seen clearly, as the rise edge at M always arrives before the rise edge at N (relation (I)) in a particular STA corner, the rise edge propagation through “N” is redundant for both setup and hold analyses. This removes the need for the timing of path "Path3r."

Similarly, as the fall edge at X always arrived before the fall edge at Y (relation (II)), the fall edge propagation through “Y” is redundant for both setup and hold. This removes the need for timing the paths Path2f and Path3f.

Hence, we remain with Path1r, Path1f, and Path2r in our static timing analysis.

As can be seen, for a very simple circuit the number of timing paths could be cut down to half.

The STA tool can disable these paths from timing analysis or in fact can remove them altogether from active memory.

The advantages of the removal of these paths and, hence, the edges of the nodes in the design are various. Firstly, it helps in removing a lot of narrow arrival timing windows of functional race conditions in SDF (Standard Delay Format) files, removing pessimism in modeling glitch power and EMIR violations. Secondly, by removing unnecessary edges, the crosstalk analysis becomes less pessimistic. In sub-10nm technologies, pessimism is a crosstalk to be borne to avoid missing potential violations. This penalizes the design overall. The third advantage is that the STA tool can remove part of the design from active memory altogether to save memory and runtime. STA runtime is a huge blocker for reticle size HPC SOCs. Almost no SOCs can be analyzed in a flat (non-hierarchical) STA run.

Advantage Area	Benefit Description
Timing Closure	Reduces pessimism, fewer infeasible paths
Glitch Power Modeling	Eliminates narrow arrival windows, lowers dynamic power
Crosstalk Analysis	Less pessimistic, fewer false violations
Memory Footprint	Reduced active memory usage, faster runtime
Large-Scale SoCs	Enables hierarchical STA, impractical in flat runs

Table 4: Advantages of Functional Pruning in Timing and Power Integrity

4. Algorithm

The first step to identify building blocks of combinations logic in the design. In the case studies above, we have assumed simple logic gates as building blocks, but these can as well be a group of logic gates, each of which has been optimized for the propagation of timing paths through them. Fig. 5 is a high-level schematic of such a setup. The yellow boxes show the groups of logic blocks; let’s call them “Logic Building Blocks” (LBB).

The second step is to identify decisive input edges for each output of the LBBs. For example, let’s take output X of an LBB and

X = f(A1r, A2f, A3r, A4r ….., ANf) where A1 to N.

Where AI is an input of the LBB.

The third step is to determine if the decisive edge arrives at an input AI before the others and propagate that edge to the output and discard all other inputs.

The fourth step is to load the minimized netlists of the LBBs into top-level STA.

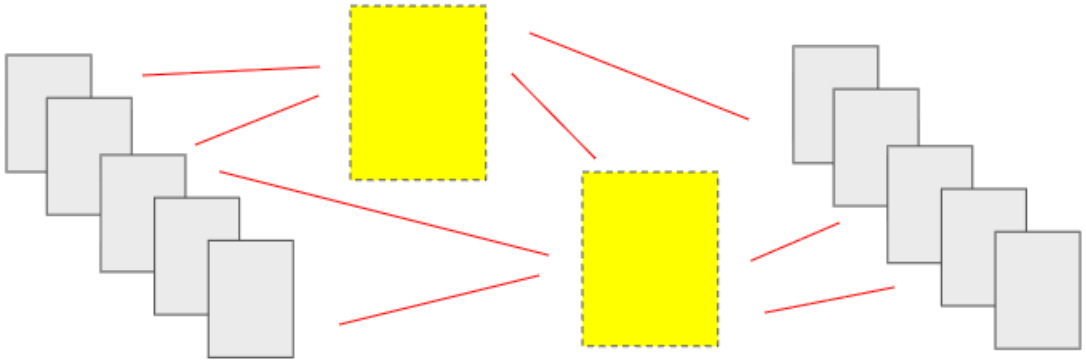

Fig. 5. Breaking the Combinational Logic in to LBBs

5. Results

The industry-standard EDA tools are not open source; hence, the method of set_disable_timing is used to disable the redundant timing paths. This should be about 30% saving on average crosstalk delta (total negative slack/number of violating nets) and 30% saving on the update_timing memory footprint. The saving in memory can be up to 70% if a reduced netlist and parasitics can be loaded into the active memory, which will not necessitate a set_diable_timing.

Metric	Value
Instance Design	100K
Timing Endpoints	30K
Maximum Logic Depth	40
Minimum Logic Depth	2
Timing Paths Reported Pre-Opt	2M
Timing Paths Post-Opt (set_disable_timing)	1M
Average Crosstalk Delay Pre-Opt	1X
Average Crosstalk Delay Post-Opt	0.7X
Peak Memory Pre-Opt	1X
Peak Memory Post-Opt (set_disable_timing)	0.7X

Table 3: Pre- and Post-Optimization Timing and Resource Metrics for 100K Instance Design

6. Conclusion

Functionality-aware static timing analysis introduces a practical refinement to conventional timing flows by dynamically pruning redundant arcs and transitions and building LBBs. From the example cases, it is clear that in even simple logic paths, exponential growth in path creation can be effectively minimized using a decisive edge selection scheme. Not only does this simplify timing graphs, but accuracy is achieved by eliminating all the impossible paths in the process. Glitching paths in combinational clouds can be cut down to a large extent. The reduced number of paths also helps in ensuring better power integrity. The proposed block-based method expands on these advantages to more complex designs by creating logic groups as building blocks and propagating only those that are decisive edges. The integration of pruning strategies into automated tools ensures scalability, allowing consistent improvements across diverse design contexts. Functionality-aware STA presents a perfect middle ground in addressing the various issues facing modern-day timing signoff, especially due to the adoption of a combination of deterministic arc characterization and adaptive pruning techniques and LBB creations. This technique fits into the current wave of technological advancement and adaptive design, making it highly relevant in advanced technologies. With this paper, the way is paved for future timing flows that are both efficient and accurate with regard to functionality.

REFERENCES

1. Abu M. Baker, “Max Operation in Statistical Static Timing Analysis on the Non-Gaussian Variation Sources for VLSI Circuits," Ph.D. dissertation, Univ. of Nevada, Las Vegas, NV, USA, Dec. 2013.
2. Shuji Tsukiyama et al., A Statistical Static Timing Analysis Considering Correlations Between Delays, Proc. IEEE Int. Symp. on Quality Electronic Design (ISQED), 2001. https://people.eecs.berkeley.edu/~alanmi/research/timing/papers/sta_del_corr.pdf
3. Bhushan Malkapurkar et al., “Statistical Static Timing Analysis for Performance of Logic Gates,” Int. J. Recent Innovation Trends Comput. Commun., vol. 3, no. 5, pp. 2862–2865, May 2015. https://www.ijritcc.org/index.php/ijritcc/article/view/4349/4349
4. Zhenyu Zhao et al., “Machine-Learning-Based Multi-Corner Timing Prediction for Faster Timing Closure," Electronics, vol. 11, no. 10, p. 1571, May 2022. https://www.mdpi.com/2079-9292/11/10/1571
5. Peng Cao et al., “Timing Analysis and Optimization Method with Interdependent Flip-Flop Timing Model for Near-Threshold Design," Electronics, vol. 11, no. 22, p. 3670, Nov. 2022. https://www.mdpi.com/2079-9292/11/22/3670
6. Xuemei Fan et al., “Design of Light-Weight Timing Error Detection and Correction Circuits for Energy-Efficient Near-Threshold Voltage Operation,” Electronics, vol. 11, no. 18, p. 2879, Sep. 2022. https://www.mdpi.com/2079-9292/11/18/2879
7. Vyacheslav Kharchenko et al., “Invariant-Based Safety Assessment of FPGA Projects: Conception and Technique," Computers, vol. 10, no. 10, p. 125, Oct. 2021. https://www.mdpi.com/2073-431X/10/10/125