On Bounding the Delay of a Critical Path*

Leonard Lee
lylee@ece.ucsb.edu

Li-C. Wang
licwang@ece.ucsb.edu

Department of Electrical and Computer Engineering
University of California - Santa Barbara
Santa Barbara, CA, 93106-9560

ABSTRACT

Process variations cause different behavior of timing-dependent effects across different chips. In this work, we analyze one example of timing-dependent effects, cross-coupling capacitance, and the complex problem space created by considering coupling and process variations together. The delay of a critical path under these conditions is difficult to bound for design and test. We develop a methodology that analyzes this complex space by decomposing the problem space along three dimensions: the aggressor space, test space, and sample space. For design, we utilize an OBDD-based approach to prune the aggressor space based on logical constraints, which can be combined with a worst-case timing window simulator to prune based on both logical and timing constraints. After pruning, the reduced aggressor space can be used to derive a more accurate timing bound. Solving the problems in the test and sample spaces is postponed to the post-silicon stage, where we propose a test selection methodology for bounding the delay of every sample. This methodology is based on probability density estimation and has a tradeoff between the number of tests to apply and the tightness of the delay bound obtained. Experimental results based on benchmark examples are presented to show the effectiveness of the proposed methodology.

1. INTRODUCTION

In today's deep-submicron designs, process variations cause different timing configurations for different silicon chips. To complicate timing further, timing-dependent effects affect each chip differently since their effects depend on the timing of the signals involved. Process variations may cause the timing of two signals to match perfectly on one chip but not on another. Figure 1 depicts examples of timing-dependent effects. For example, past research has studied the multiple input switching (MIS) effect on delay [3, 5], the automatic test pattern generation (ATPG) problem under the effect of cross-coupling capacitance [8, 9, 15], and the impact of power noise [6, 11, 12, 14].

In this work, we focus on one of these effects, cross-coupling capacitance, mingled with process variations and solve two problems: (1) how to bound the timing of a critical path for design, and (2) how to bound the timing of a critical path with tests for speed binning.

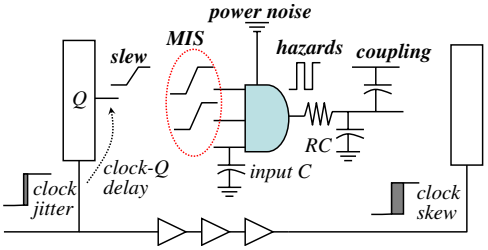


Figure 1: Path delay is affected by many different timing-dependent effects

Coupling's effect on the timing of a victim path is based on the timing alignment between its signals and the aggressors, where a full timing overlap has the largest impact and no overlap has the least. Many models multiply the coupling capacitance CC between an aggressor and a victim wire with a coupling factor, representing the severity of this alignment. This modified capacitance is added to the output loading of a victim gate to change its timing. In static analysis, setting a global coupling parameter of xCC is a simple but inaccurate model of coupling, as not all gates in a path are affected by xCC, causing overestimation of its timing.

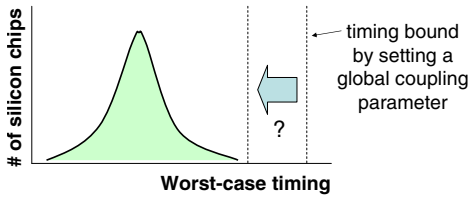


Figure 2: Can we push in the timing bound found by setting a global coupling parameter deterministically instead of empirically?

Figure 2 illustrates how an xCC model might overestimate the actual worst-case timing of a victim path. A common industrial practice to reduce this overestimation is to set the value x smaller than 2, where the value is determined empirically based on data from a previous design, process, or tapeout. In this work, we attempt to push in the timing bound for design and test without relying on this empirical data. We provide deterministic solutions to reduce the complexity of the cross-coupling capacitance problem space based on several different perspectives, and provide additional approximate solutions to estimate the delay bound of a critical path in test.

*This work was supported in part by NSF, Grant No. 0312701 and SRC, project 2004-TJ-1173.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICCAD'06, November 5-9, 2006, San Jose, CA
Copyright 2006 ACM 1-59593-389-1/06/0011 ...$5.00.

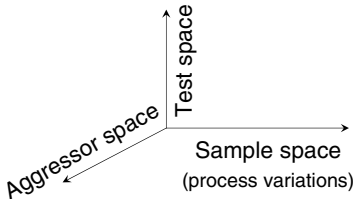


Figure 3: The 3 dimensions of the coupling problem

Figure 3 shows how the coupling problem space is created by the interaction between three dimensions:

- *Aggressor space*: the logical and timing constraints involved in activating combinations of aggressors.
- *Test space*: the various tests that can activate the required aggressors for a given combination.
- *Sample space*: process variations causing different delays for the same test on different chips.

In an earlier work [16], an attempt to develop a *statistical timed ATPG* revealed that searching the problem space with these dimensions mingled together is difficult. In this work, we propose to solve the problem along each dimension separately, reducing the complexity.

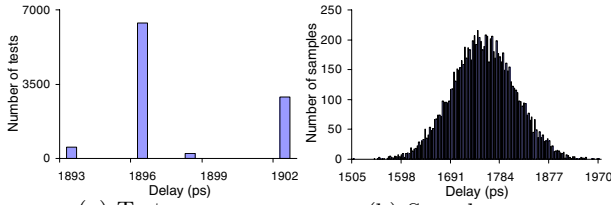
In the pre-silicon stage, where we cannot assume the availability of a statistical timing model that accurately models all effects from process variations, focusing only on pruning the aggressor space makes more sense. It is questionable to use *Timed ATPG* to search for tests that worsen the delay of a victim path because it requires an accurate timing model to find the alignment of timing-dependent effects. Hence, its result is sensitive to the timing model. A slight change in the model may result in a completely different test set.

Due to the difficulty in analyzing the path delay on a per-test basis, our methodology postpones the development of the final test set to expose the coupling effect on a path until the post-silicon stage. Our main idea is that in the pre-silicon stage, we only need to (and can only) develop a *superset* of tests. Then, a subset of these tests are selected for mass production, using post-silicon samples.

Pruning the aggressor space is essential for both design and test, and therefore it is the most fundamental step to bound the delay of a critical path. The logical and timing constraints imposed by the circuit can reduce the number of aggressors that need to be analyzed, resulting in pushing in the timing bound. Note that it is crucial to assume the worst-case in timing for pruning the aggressor space. Hence, the delay information on every signal is represented as a worst-case timing window. We cannot assume any knowledge of the actual delay distribution inside a timing window.

We propose an approach that combines Boolean Satisfiability (SAT) solving [18] and Ordered Binary Decision Diagrams (OBDD) [4] for pruning the aggressor space. On a given combination of aggressors, we rely on SAT to decide if a test can be produced to activate all aggressors simultaneously. We use OBDDs to store combinations that are possible or impossible to generate a test for. Information stored in OBDDs is then used to guide the selection of new combinations to try. The goal of this process is to identify all possible combinations that a test can be produced for.

To further identify a set of tests required to bound the delays (of a critical path) on silicon samples, we need to develop a silicon sample-based methodology. Figure 4 shows the complexity of the test space and the sample space individually by fixing the other dimensions. These are simulation results based on benchmark circuit c880 and on the most critical path reported by a statistical timing tool. To facilitate our study, we implement a statistical timing simulator [17] to emulate the behavior of the N sample chips, each with a different delay configuration statistically derived from a timing model of process variations. The statistical timing model is cell-based, which was characterized using Monte Carlo SPICE based on a 90nm CMOS technology. The delay values (their means and standard deviations) are stored as tables rather than timing rules.



(a) Test space (b) Sample space

Figure 4: Analyze a dimension by fixing two others

Figure 4-a shows that for a fixed aggressor combination on a fixed sample, the 10k randomly selected tests (that all activate the given combination) cluster into four groups. This suggests that test selection is needed to choose the desired worst-case test. It also shows that many tests produce the same effect on the path, if we consider only a particular combination of aggressors. Figure 4-b shows that for a fixed aggressor combination and a fixed test for the combination, 10k samples fit a Normal distribution well. The sample space problem can be solved by parameter estimation, where random sampling to find the mean and standard deviation of the distribution can capture its complexity.

We propose a sample-based methodology to bound the delay of every sample individually. In this methodology, we focus on the test space. In the post-silicon stage, we assume that a set of N (good) samples are available. Our objective is to develop with a high confidence a test set that is sufficient to be used to bound the delays of silicon chips in mass production. We generate a superset of tests based on the result from aggressor space pruning, and construct a smaller test set via silicon sample-based test selection.

To compare the quality of the superset and the selected set, we require a methodology to bound the delay of every sample based on a given set of tests. If such a methodology exists, then we can check to make sure that the bound given by a selected test set does not underestimate the delay bound from that given by the superset.

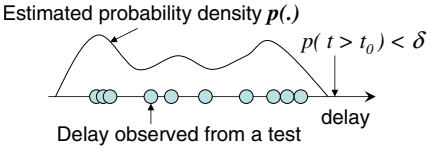


Figure 5: Applying density estimation to bound the delay of a sample

In this work, we turn the problem of bounding the delay of a sample into a *density estimation problem* [23]. Figure 5 illustrates the concept. Essentially, our goal is to estimate the distribution in the test space of a given sample based on a set of tests. The delays from these tests are treated as example delay points randomly drawn from a fixed and unknown distribution (in the test space). Once we can estimate this distribution as $p(.)$ based on these points, we can then bound the delay t by calculating the probability $p(t > t_0)$. For a given confidence level $(1 - \delta)$, we find the

smallest t_0 such that $p(t > t_0) < \delta$. Then, the value t_0 becomes the delay bound.

Note that in density estimation, we cannot rely on traditional *parametric estimation* by assuming a distribution model in advance, for example a Gaussian distribution or a mixture of Gaussian distributions. Figure 4-a shows that the distribution in the test space on a given sample does not fit into a parametric form of any continuous probability distribution. Hence, we need to apply *non-parametric estimation* to estimate $p(.)$. A popular approach for non-parametric density estimation is *kernel density estimation* [23].

Because our methodology to bound the delay of each sample relies on density estimation, we will see later in the paper that this methodology allows a tradeoff between the number of tests to apply and the tightness of the delay bound. Intuitively, if fewer tests are applied, we need to leave a bigger margin in the delay bound to account for the uncertainty in the unseen space. From this perspective, we see that the proposed methodology is not trying to find the absolute worst-case delay on a sample. Instead, it performs statistical inference to estimate a bound such that the probability of the actual delay exceeding this bound is extremely small.

We emphasize that in our work, density estimation is used only after aggressor space pruning. Otherwise, the combined aggressor and test spaces can be too large and complex for any density estimation method to work effectively.

The rest of the paper is organized as follows. Section 2 analyzes the aggressor space in more detail. Section 3 develops an OBDD-based solution for the logical constraints of the aggressor space, and Section 4 adds the timing constraints to the solution. After pruning the aggressor space, Section 5 discusses our test selection methodology using density estimation, and Section 6 concludes the work.

2. THE AGGRESSOR SPACE

The aggressor space consists of all possible aggressor combinations. It is not possible to produce a test for many of these combinations. The constraints imposed on this space can be illustrated with a Boolean Lattice. Given n aggressors $a_1, \ldots, a_n$, each node in the Lattice is associated with an n-bit vector where each $a_i \in \{0, 1\}$, $1 \leq i \leq n$. Each bit vector is an aggressor combination, where $a_i = 1$ means to activate aggressor i. Activating an aggressor means that a test can produce a transition on the aggressor opposite to the transition on the corresponding victim wire. An $a_i = 0$ means that the aggressor is *unconstrained* and hence can be activated or not activated.

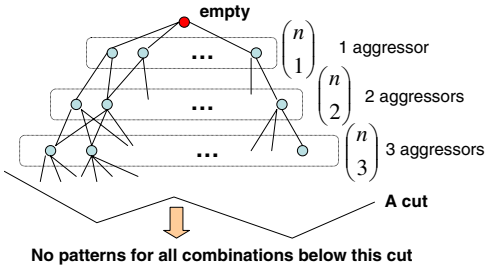


Figure 6: The Lattice view of the aggressor space

Figure 6 shows the Lattice space exploration problem. A node in the Lattice is assigned with an output value 0 or 1. For example, the root node has a value 1, meaning that a test exists to sensitize the victim path while leaving all aggressors unconstrained. The nodes in the j^{th} level represent all possible combinations of aggressor activations where j aggressors are constrained to be activated with the victim path. It is interesting to observe that if a set of selected aggressors cannot be activated together, then any aggressor combination containing the set of aggressors cannot be activated together either. Therefore, the problem of identifying all aggressor combinations for which a test can be produced becomes the problem of finding a *cut* on this Lattice. Below the cut, there exists no test for any combination and above the cut, a test exists for every combination. Section 3 discusses in detail how to find this cut.

The timing alignment of aggressor-victim pairs also introduces a timing-related problem space. Even if a test exists for a combination of aggressor activations, some of the transitions on the aggressors may not align timing-wise with the victim transitions. We discuss the methods to consider timing alignment in Section 4.

2.1 Cross-coupling modeling

The coupling capacitance CC is obtained from capacitance extraction on the layout. When there are opposite transitions on the two signals related to this CC, the delay gets pushed out because of the coupling effect. The exact amount of this push-out is some non-linear function of the timing alignment between the two signal transitions. The actual mechanisms causing the delay push-out can be quite complex [10, 13, 19, 20]. In actual design, people may not be able to afford using such a detailed analysis. However, using a simple xCC approach can be very pessimistic.

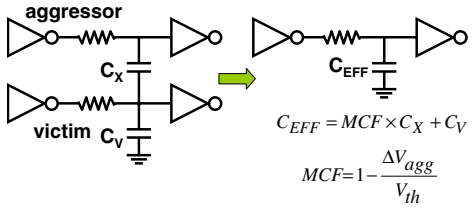


Figure 7: More sophisticated MCF-based practice

It is widely recognized that the xCC analysis is too rough. A more sophisticated method is to adopt the Miller Capacitance Factor (MCF) [7], as shown in Figure 7. MCF takes the timing alignment into account. ΔV_{agg} is the percentage of voltage change on the aggressor when the victim transitions from 0 to V_{th} (or from V_{dd} to V_{th}). For example, we can assume $V_{th} = 0.5V_{dd}$ in the formula. Under this assumption, $MCF \in [-1, 3]$, which also models the speed-up effect.

2.2 Search for the worst-case tests

The problem of searching for the tests to expose the worst-case delays of a victim path cannot be deterministically defined. This is because the worst-case test can be different from chip to chip. Finding the test set that can expose the worst-case delays for all chips can be a very difficult problem. Essentially, this is a *statistical-timed* ATPG problem [16]. In such an ATPG, search is carried out by mingling the aggressor, test, and sample spaces. Consider a 100-input circuit block with 50 aggressors and ±5% chip-to-chip delay variability. Searching for the right tests in such a large and complex space is extremely difficult. Adding to the difficulty is the lack of an accurate statistical timing model.

2.3 Searching the tests vs. bounding the delay

If we cannot afford to search for the worst-case tests, the alternative is to bound the delay of the path on every sample. This relaxes the requirement to find the exact test that

causes the worst case on every sample. However, searching for the tests that allow us to estimate a reliable delay bound on every sample is not trivial, so we need to begin by pruning the aggressor space.

Given a large number of n aggressors, pruning the aggressor space first makes more sense because it may be impossible for many aggressors to be activated together due to the circuit's logic constraints. Intuitively, the more aggressors we consider together, the less likely it is that they can all produce timing effects simultaneously. Therefore, an effective pruning methodology on the aggressor space can quickly reduce the number of aggressors to be considered and hence, reduce the complexity for solving the remaining problem.

In the pre-silicon stage, we can only afford pruning the aggressor space because, to further identify which tests are causing the worst-case delays, we need to carefully analyze the timing alignment among signal transitions. This demands an accurate timing model. With a fixed-delay timing model, the analysis of timing alignment is well defined. However, with a statistical timing model, the analysis cannot be deterministically defined.

If the aggressor space can be pruned effectively, the result can help to bound the path delay in design. For example, instead of considering n aggressors, the result of pruning may leave only $l \ll n$ aggressors to be considered. Then, an existing static timing analysis methodology can be applied with the l aggressors to obtain a tighter bound.

The discussion so far points to the importance of pruning the aggressor space. Hence, in this paper, we devote Sections 3 and 4 to discuss various methods for pruning the space. We first introduce a method to prune the aggressor space based on logical constraints only. Then we develop a method to prune further by also using timing constraints. However, because these two methods may individually take a long time (under 12 hours) to run, we introduce a more efficient methodology (an hour or two) in Section 4.2 that integrates both methods, combining both logical and timing constraints. Then, Section 5 discusses a sample-based method to obtain the test set for bounding the delay of the critical path on silicon samples.

3. PRUNING: LOGICAL CONSTRAINTS

We first focus on the constraints imposed by the circuit structure, ignoring any timing information. By analyzing the coupling capacitance information, we can obtain an initial set of aggressors that *affect the victim path* with non-zero coupling capacitance. For the two most critical paths of ISCAS85 benchmarks c880 and c1908, the coupling capacitance extraction at [1] reveals 72 and 76 aggressors affecting c880, and 88 and 81 aggressors affecting c1908. The first step in pruning using logical constraints simply involves removing aggressors that cannot *transition with the victim path*, resulting in 33 and 35 aggressors for c880, and 31 and 28 aggressors for c1908.

3.1 Pruning the remaining logical space

At this point the remaining aggressors can transition with the path but not necessarily with other aggressors. We propose a solution by combining SAT and OBDDs to collect aggressor combinations that are possible and impossible.

The Boolean Lattice in Figure 6 can be viewed as a diamond, with 1 combination at the top ($\binom{n}{0}$) and 1 combination at the bottom ($\binom{n}{n}$). For each combination, we use SAT to find if there exists a test to activate that combination together with the sensitization of the victim path (SAT), or if there does not exist any test to activate that combination (unSAT). This information is stored into two OBDDs, one representing the upper bound (UB OBDD) of the cut through the Boolean Lattice, and the other representing the lower bound (LB OBDD) of the cut.

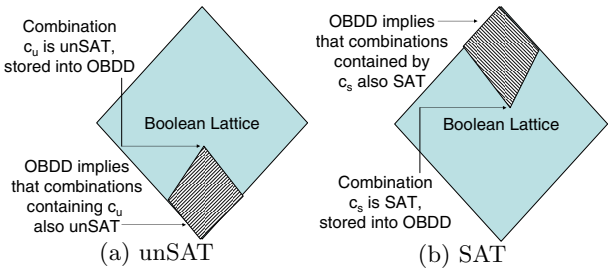


Figure 8: Storing combinations into an OBDD

Figure 8 shows how the two OBDDs can represent the upper and lower bounds of the cut. In Figure 8-a, when a combination c_u is found to be unSAT, any combination c_i that contains c_u is also unSAT. For example, if activating $\{a_1, a_2\}$ is impossible, then activating $\{a_1, a_2, a_3\}$ is also impossible. Similarly, Figure 8-b shows that when c_s is found to be SAT, any combination c_i that is contained by c_s is also SAT. For example, if activating $\{a_1, a_2, a_3\}$ is possible, activating only $\{a_1, a_2\}$ is also possible.

We can combine the unSAT combinations into the LB OBDD and the SAT combinations into the UB OBDD. Each variable in an OBDD represents one aggressor, where the $a_i = 1$ (or just a_i) branch signifies that the aggressor a_i is activated, and the $a_i = 0$ (or $\overline{a}_i$) branch signifies that it is unconstrained (can be activated or not).

The LB OBDD starts as the 1 terminal to represent that everything is still possible, especially since we have not even seen evidence that activating all aggressors simultaneously is impossible. When we find an unSAT combination of $\{a_1, a_2\}$, we create a new OBDD that has the branch $a_1 = 1$ and $a_2 = 1$ lead to the 0 terminal. All other branches still lead to the 1 terminal. By performing an OBDD AND operation on the LB OBDD and the newly created OBDD, we can modify the LB OBDD to represent that activating $\{a_1, a_2\}$ together is impossible. Notice how the LB OBDD now effectively represents any combination containing $\{a_1, a_2\}$ (like the $\{a_1, a_2, a_3\}$ example just described) is impossible.

Likewise, the UB OBDD starts as the 0 terminal to represent that everything still has not been tried. When we find a SAT combination of $\{a_1, a_2, a_3\}$, the new OBDD to create is not as simple as in the LB OBDD case. For example, if there are a total of 5 aggressors, the new OBDD would have the branch $a_4 = 0$ and $a_5 = 0$ lead to the 1 terminal (with all other branches leading to the 0 terminal). Notice how this OBDD also represents that the combination $\{a_1, a_2\}$ is also SAT because $\{\overline{a}_3, \overline{a}_4, \overline{a}_5\}$ contains $\{\overline{a}_4, \overline{a}_5\}$. The new OBDD represents that anything containing $\{\overline{a}_4, \overline{a}_5\}$ is SAT. Finally, by performing an OBDD OR operation on the UB OBDD and the newly created OBDD, we can modify the UB OBDD to represent SAT combinations.

Using the LB OBDD and the UB OBDD, we can bound where the cut of the Boolean Lattice is. Therefore, we can use these OBDDs to guide what combinations to attempt next. We create a *generator OBDD* by taking (LB OBDD) AND (UB OBDD)' as shown in Figure 9. Finding paths to the 1 terminal in the generator OBDD gives combinations that lie below the upper bound and above the lower bound. Each attempted combination prunes a sub-space that no

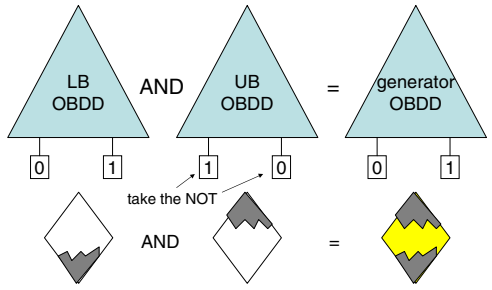


Figure 9: Creating the generator OBDD

longer needs to be searched, reducing the remaining problem space where possible combinations may still exist.

3.2 Finding the exact cut in the Lattice

Our iterative exhaustive methodology that utilizes the upper and lower bound OBDDs consists of the following steps:

1. Sort the aggressors based on their CC.
2. Use the aggressor with the largest CC as the only variable in the OBDD variable list.
3. Initialize LB OBDD to 1 and UB OBDD to 0.
4. Compute the current *generator OBDD*.
5. Generate all paths to the 1 terminal of the generator OBDD, considering the current OBDD variable list. Every path consists of some aggressors as 1, some as 0, and the rest as 2 (either 0 or 1). For each path with 2's, generate all permutations for that path. ex. 2201 will generate 0001, 0101, 1001, and 1101.
6. Pad the permutations with 0's for aggressors not yet in the OBDD variable list (recall 0's are don't cares).
7. Use a SAT solver to find tests for all permutations.
8. Add unSAT and SAT permutations to the OBDDs.
9. If the number of variables in the OBDD variable list is equal to the number of aggressors, terminate.
10. Add the next largest CC aggressor into the list.
11. Return to step 4.

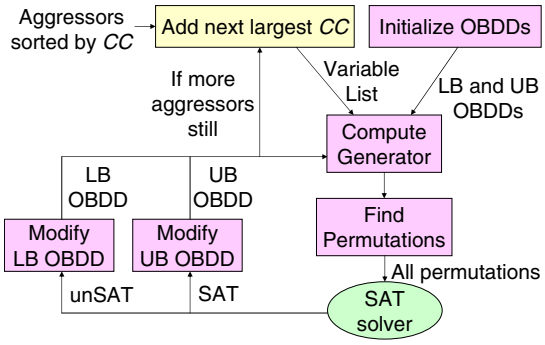


Figure 10: Our iterative methodology using OBDDs

This iterative methodology exhaustively explores the Boolean Lattice. The UB and LB OBDDs record which combinations are already known to be possible or impossible. Therefore, by using both OBDDs to find new combinations to try, we are focusing only on unexplored sections of the Lattice. However, since this is an iterative methodology, we also need to ensure that the UB and LB OBDDs are still accurate after adding another aggressor into the variable list.

For the LB OBDD, no change is necessary because it doesn't matter if we set the extra aggressor to 0 or 1 to make a new combination c_{new}, c_{new} still contains the original unSAT combination c_u. However, the UB OBDD requires a straightforward change to remain valid. Recall that to modify the UB OBDD for each combination, we need to know the total number of aggressors. Since we are modifying the total aggressor list by adding a_{new}, we need to AND the UB OBDD with $\overline{a}_{new}$. This fixes each path to the 1 terminal by requiring the new aggressor to also be unconstrained.

Although this exhaustive iterative methodology can find the exact cut, it takes a long time to run; it takes over a week to find the cut for the c1908 path with 28 aggressors, which has the least aggressors to begin with. This deterred us from running this exhaustive methodology on larger examples. Therefore, we introduce a more efficient alternative by relaxing the requirement for finding an exact cut.

3.3 Finding a bound of the cut for efficiency

To reduce the runtime, we loosen the constraints of finding the true cut of the Boolean Lattice. If our objective is simply to find a close upper and lower bound, we can relax step 5 in the methodology. Recall that each path to 1 in the generator OBDD contains 0's, 1's, and 2's. In the exhaustive methodology, we enumerated all permutations of 2's, and then used SAT on all these permutations. In our more efficient alternative, we instead generate only 1 permutation per path, where the 2's are randomly set to 0 or 1. This alternative reduces the run-time of from over a week to overnight (less than 12 hours), which is a vast improvement.

The increased performance comes at the cost of exhaustiveness. This methodology does not guarantee finding the exact cut. Instead, it finds the lower and upper bounds of where the cut might be. Comparing the alternative and exhaustive methods on c1908, the LB OBDD of the alternative has recorded only 2,831,280 fewer impossible combinations than the LB OBDD of the exhaustive method. Although this number may seem large, the total number of impossible combinations found by the exhaustive method is 267,390,976. Therefore, the alternative missed only 1% of the impossible combinations. This confirms our conjecture that the speed-up comes at the cost of not finding the true cut because not as many combinations are solved for compared to the exhaustive method. However, it does get close.

Because we do not have the true cut for examples other than c1908, we introduce a different way of evaluating the effectiveness based on the generator OBDD. At the completion of the exhaustive methodology, the generator OBDD will always have 0 unknown combinations, because all combinations are known to be either possible or impossible. Therefore, we evaluate our methodology by the number of unknown combinations still in its generator OBDD. Table 1 shows the percentage of unknown combinations to the total 2^n combinations. Although you would expect that the drastic speed-up introduces a large error in bounding the cut, our methodology is still very effective with less than 1% of the space left unexplored.

Table 1: Percentage of unknown combinations left in the generator OBDD to the total 2^n combinations

	c880-1	c880-2	c1908-1	c1908-2
# of Aggressors (n)	33	35	31	28
Percentage	0.026%	0.014%	0.247%	0.704%

Informal comment: Having a runtime less than 12 hours is important in practice as the methodology can be started before leaving the workplace for the day, and the results will be available upon returning the next day. Since the exhaustive methodology takes too long for practical use, the relaxed methodology for finding the tight bound on the cut can be useful for pruning the aggressor space. These examples with

less than 36 aggressors complete overnight. However, if we reduce the initial number of aggressors to be less than 25, we can prune the aggressor space in only a few hours.

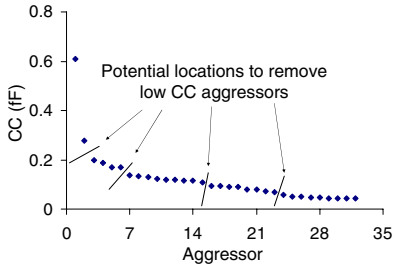


Figure 11: Aggressors sorted by CC

In practice, the runtime can be controlled by limiting the number of aggressors to begin with. Figure 11 shows an example case of the aggressors sorted by their CCs with the victim path. There are natural separations to throw out aggressors with small CCs. From experiments, we conclude that as long as the number of aggressors is limited within 35, the current method is able to approximate the true cut well within a 12-hour runtime limitation. The result of this can serve as the basis for further pruning by considering timing constraints. Below describes such a pruning approach.

4. PRUNING: TIMING CONSTRAINTS

Past research in ATPG that target timing-dependent effects [6, 8, 9, 11, 12, 14, 15] commonly utilize a pre-silicon fixed-delay timing model when trying to find the tests that exercise these effects. Unfortunately, when accounting for process variations, the accuracy to runtime tradeoff in pre-silicon models causes timing to be inaccurate. This uncertainty is especially problematic when finding worst-case delay tests. A small error or variance in the timing model can invalidate a test set found by a pre-silicon ATPG methodology. Therefore, ATPG may create a worst-case test set in pre-silicon that is completely disjoint from the actual worst-case test set on silicon.

We propose a pruning methodology that is not as sensitive to variations and uncertainties in the timing model. Assuming a fairly accurate worst-case timing model, we only prune aggressors by a conservative worst-case analysis.

4.1 Pruning with timing-window simulation

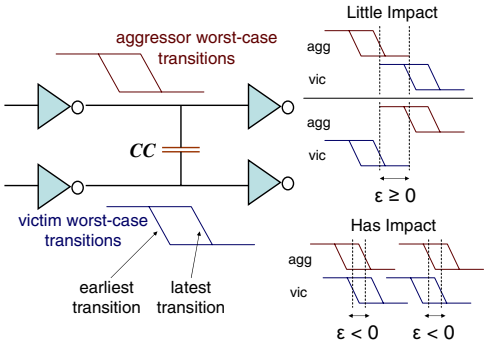


Figure 12: Using 3σ timing-window simulation to prune aggressors with little or no impact

By performing an earliest and a latest 3σ timing simulation for each test found in Section 3.3, every signal in the circuit will have an earliest and latest time of transition. Since our statistical timing model contains the mean μ and standard deviation σ of each pin-to-pin delay, we use $\mu \pm 3\sigma$ on every pin-to-pin delay to obtain the earliest and latest.

Figure 12 shows the two cases when we are sure that an aggressor has little or no impact on the victim under a specific test ($\varepsilon \geq 0$), and two cases when an aggressor has an impact ($\varepsilon < 0$). By simulating all the tests found in the logical pruning phase in this manner (around 1M tests for c880 and 500k tests for c1908), for each aggressor j on T tests, we have $\varepsilon_{j1}, \varepsilon_{j2}, ..., \varepsilon_{jT}$. To prune an aggressor j, it needs to have little or no effect on the victim path over all tests. That is, $\varepsilon_{jk} \geq 0 \forall k$. Moreover, if the uncertainty of the timing model is larger, the constraint $\varepsilon_{jk} \geq \kappa$, where $\kappa > 0$ and is a constant, makes pruning more conservative.

Note that simulation with patterns is very important to calculate timing alignments as our experiments without patterns (static analysis) did not effectively prune enough aggressors when assuming the worst-case MCF effects.

Table 2: Pruning with timing-window simulation

Pruning Method	c880-1	c880-2	c1908-1	c1908-2
Affects Path	72	76	88	81
Transitions With Path	33	35	31	28
Left After Additional Pruning	7	10	12	17

Table 2 shows results of pruning additional aggressors for the same two ISCAS85 benchmarks. Recall from Section 3.3 that the logical pruning may take overnight. Simulation of a million tests also may take many hours. This motivates us to find a more efficient alternative for combining logical and timing constraints in pruning the aggressor space.

4.2 A very efficient alternative for pruning

Previously, we perform logical pruning first, and then bring timing into the analysis afterwards. To improve the runtime of our pruning methodology, in this section, we repeatedly alternate between using logical constraints and timing constraints. We begin with all the aggressors as variables in the OBDD variable list and initialize the UB OBDD to 0 and the LB OBDD to 1. The first generator OBDD is simply 1, so instead of finding all paths to 1 like before, we find one random path to 1 from the generator OBDD and change all 2's into 0's. After the combination is found to be possible or impossible, the appropriate OBDD is changed and a new generator OBDD is calculated. This repeats M times, then the tests found during that time are simulated. Note for a very large M, this process degenerates back to Section 4.1, where all the tests are found first, and then simulated. In this section, we find that for a smaller $M = 1000$, the power of the timing constraints can be exploited early.

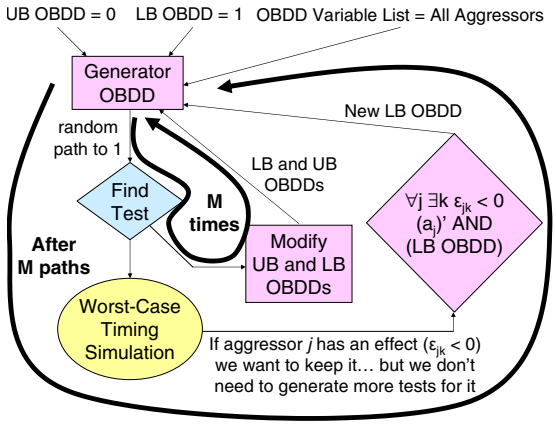


Figure 13: Combining logical and timing constraints

Figure 13 shows the flow of this methodology. We run logical pruning M times and then simulate the U tests found so far. This allows timing constraints to be fed back into the logical pruning via the LB OBDD. We add these steps:

- After finding U tests from the M combinations, simulate all U tests on the worst-case timing simulator.
- Stop trying to generate tests for each aggressor j that we determined *has an effect* ($\exists k, 0 < k \leq U | \varepsilon_{jk} < 0$).
- To ignore these, AND the LB OBDD with $\overline{a}_j$.

We know that each aggressor j that could have a timing effect in the worst-case timing simulator cannot be pruned by our methodology. Therefore, as soon as some of these aggressors are identified, this alternative methodology stops trying to prune those aggressors in logical analysis (they become unconstrained in the analysis). This allows the methodology to focus on aggressors that might have no effect instead of wasting runtime on activating aggressors that are already known to not be prune-able. Basically, each iteration of this methodology *selects aggressors to be kept because they have shown a timing effect on at least one test.* Consequently, termination must rely on empirical evidence, ex. terminating when no new aggressor has been found to have a timing effect in the last C iterations.

For $M = 1000$ and $C = 50$, each iteration takes only minutes to run. Therefore, this alternative completes in an hour or two, a vast improvement over the other methodologies described earlier. For the four examples in Table 2, it only takes 1 iteration for this alternative to run and it reaches the same number of aggressors as that in Table 2. Table 3 shows the effectiveness of this methodology on additional ISCAS85 benchmark examples.

Table 3: Pruning on selected ISCAS85 benchmarks

Path	Aff. Path	Trans. w/ Path	After Pruning	Iter.	p	Aggr. Left (l)
c880-1	72	33	7	1	0.00	7
c880-2	76	35	10	1	0.00	10
c880-3	78	41	16	3	0.25	10
c880-4	75	38	10	1	0.00	10
c880-5	79	40	12	1	0.00	12
c880-6	81	45	20	4	0.22	10
c1355-1	81	38	20	1	0.20	9
c1355-2	79	38	18	1	0.20	9
c1908-1	88	31	12	1	0.00	12
c1908-2	81	28	17	1	0.20	10
c7552-1	157	80	22	2	0.16	10
c7552-2	166	90	30	8	0.20	9

4.3 Preparation for post-silicon testing

Table 3 shows the remaining aggressors after aggressor space pruning. For any $l \leq 10$, $2^l < 1024$, which is a reasonable number of combinations to generate 10 tests for each combination, resulting in a superset test size of around 10k. (Note that the million tests found earlier correspond to a million different combinations and is prohibitively large, and so cannot be used as a superset for test selection.) For the other examples, the number of aggressors may still be large. Hence, we need to reduce l further. Note that given a set of aggressors left after the pruning, we do not exactly know in the pre-silicon phase which tests from which combinations may cause excessively long delays on silicon samples. Hence, a more reasonable solution is to reduce the number of aggressors and then produce tests for every combination of aggressors to be tried out on silicon samples.

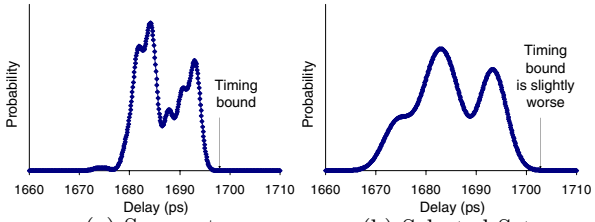
To reduce the number of aggressors further, we use a common industrial practice of removing aggressors that only have small coupling capacitances with the victim path. However, as opposed to using all the CC values from [1] and filtering out those smaller than a percentage p of the largest CC_{top}, our set consists of only those that have impact CC_{align}, where $\varepsilon_{align} < 0$ as shown in Figure 12. Table 3 shows that $p \approx 0.20$ is enough to reduce large examples to a reasonable number of aggressors.

5. TEST SELECTION

We have just analyzed and simplified the aggressor space from Figure 3. In this section, we propose a solution for test selection. Note that a global coupling parameter on the remaining aggressors after pruning is already a big improvement in the worst-case bound for design. Furthermore, the superset of tests based on the reduced aggressor set from Table 3 can be used to bound the timing for test. From 2^l combinations of aggressor activations, we can generate $C = 10$ tests per combination. However, this test size of $C2^l$ might still be too large for mass production. Therefore, in this section, we develop a test selection methodology to create a smaller test set and use density estimation to compare the effectiveness of the selected test set to the superset.

We apply the $C2^l$ tests onto N sample chips. In this work, we use a Monte Carlo statistical timing simulator, but our test selection can easily use actual silicon chips. We select the top K delay tests seen on each chip and collect them as the selected test set. Table 4 shows the sizes of the selected test set for $N = 50$ and $K = 10$.

Recall Figure 5 in the introduction, where the delays of the tests are used to find a timing bound for every sample individually through density estimation. Our goal is to ensure that the selected test set results in a more conservative bound than that given by the superset.



(a) Superset (b) Selected Set

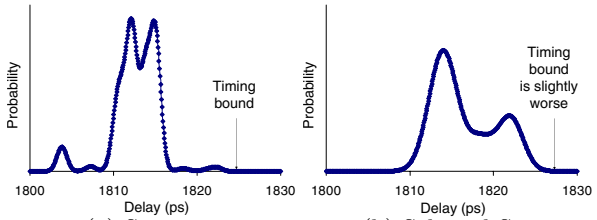
Figure 14: Density estimation on one sample

We utilize the *density* function in the *stat* package of the statistical tool R [2] for density estimation of a set of test delays on a sample. This function uses fast Fourier transforms to convolve an approximation with a discretized version of the kernel and linear approximations to evaluate the density at specific points [21, 22]. Figure 14 shows the density estimation for one sample chip of c880-1. We can clearly observe that the timing bound of the superset is tighter than that of the selected set. This is what we wanted.

It is interesting to note that deciding the timing bound based on density estimation naturally allows a tradeoff to be made between the number of tests to apply and the tightness of the bound obtained. With a smaller test set, because there are fewer data points to estimate the unknown distribution, the estimation is less precise and hence, results in a looser bound. If a tighter timing bound is required, then one can always include more tests. The tradeoff is very intuitive.

Figure 15 shows the density estimation for another sample chip of c880-1, and again we see the same tighter bound for the superset and looser bound for the selected set.

To measure the effectiveness of the selected set compared to the superset, we cannot just look at one sample. Therefore, we take the average timing bound over all samples for



(a) Superset (b) Selected Set

Figure 15: Density estimation on another sample

the superset and selected set. Table 4 shows that indeed, using the superset produces a slightly tighter bound than the selected set. However, note that the actual difference in the bound is much less than 1% over all examples. This shows that if we are allowed a 1-2% margin, the density estimation method demands a very small test set to determine a conservative timing bound on every sample.

Table 4: Difference in average timing bound

Circuit	Superset Bound (ps)	Selected Set Bound (ps)	Difference in Bound	Selected Set Size*
c880-1	1762.915	1765.97	0.1733%	60
c880-2	1758.186	1761.543	0.1909%	96
c880-3	1769.12	1770.931	0.1024%	68
c880-4	1753.318	1754.906	0.0906%	99
c880-5	1746.698	1749.692	0.1714%	73
c880-6	1753.572	1755.538	0.1121%	76
c1355-1	1739.008	1739.204	0.0113%	80
c1355-2	1721.848	1722.647	0.0464%	72
c1908-1	1806.478	1807.923	0.0800%	51
c1908-2	1783.986	1784.032	0.0026%	190
c7552-1	2233.757	2235.919	0.0968%	161
c7552-2	2277.706	2282.426	0.2072%	121

* The superset size is $\approx 10 \times 2^A$ where A is the number of aggressors left in Table 3

Informal comment: It is interesting to note that if we increase the sample size N from 50 to 500, a selected test set grows by very few tests. Hence, the *selected-test-to-sample* ratio approaches zero quickly as N increases. This is also true for using other small K values. Hence, we conjecture that with a reasonable sample size, say 2K, the selected test size would always be much smaller than the superset size. The study of the *test-to-sample* ratio requires careful analysis of the statistical complexity in sample behavior and hence, is left to future work.

6. CONCLUSION AND FUTURE WORK

We propose an effective pruning strategy utilizing OBDDs, SAT, and a timing-window simulator to significantly reduce the aggressor space for both design and test. Working on this reduced space is more accurate than setting an xCC global coupling parameter on the original aggressor space. Our most exhaustive pruning methodology is not feasible in practice. However, by relaxing the requirements slightly, we present a pruning methodology that finds a very tight bound in less than 12 hours (allowing to run overnight). Furthermore, if there exists a trusted worst-case timing model, we propose an aggressor pruning methodology that completes in an hour or two and finds aggressors that have a timing effect. Overall, our analysis is conservative and hence, the result does not depend on any assumption to model the actual statistical distribution of process variations.

From the reduced aggressor space, we can create a superset of tests to bound the timing for test. However, this superset can be quite large. If we allow a small margin to be added in the timing bound, the test set can be reduced drastically. The proposed kernel density estimation method to estimate timing bounds on chip samples is quite interesting. We plan to study more on the tradeoff between timing bound accuracy and test set size in the future.

7. REFERENCES

[1] Layout and parasitic information for ISCAS circuits. http://dropzone.tamu.edu/~xiang/iscas.html.

[2] The R project for statistical computing. http://www.r-project.org/.

[3] A. Agarwal et al. Statistical gate delay model considering multiple input switching. In *Design Automation Conf.*, pages 658–663, June 2004.

[4] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, 1986.

[5] V. Chandramouli and K. A. Sakallah. Modeling the effects of temporal proximity of input transitions on gate propagation delay and transition time. In *Design Automation Conf.*, pages 617–622, June 1996.

[6] T. S. Chang et al. Test generation for maximizing ground bounce for internal circuitry with recovergent fan-outs. In *VLSI Test Symp.*, pages 358–366, 2001.

[7] P. Chen et al. Miller factor for gate-level coupling delay calculation. In *ICCAD*, pages 68–74, Nov. 2000.

[8] W. Chen et al. Test generation in VLSI circuits for crosstalk noise. In *ITC*, pages 641–650, 1998.

[9] W. Chen et al. Test generation for crosstalk-induced delay in integrated circuits. In *ITC*, page 191, 1999.

[10] F. Dartu and L. T. Pileggi. Calculating worst-case gate delays due to dominant capacitance coupling. In *Design Automation Conf.*, pages 46–51, 1997.

[11] Y.-M. Jiang and K.-T. Cheng. Vector generation for power supply noise estimation and verification of deep submicron designs. *IEEE Trans. VLSI Syst.*, 9(2):329–340, Apr. 2001.

[12] Y.-M. Jiang et al. Estimation for maximum instantaneous current through supply lines for CMOS circuits. *Tran. VLSI Syst.*, 8(1):61–73, Feb. 2000.

[13] A. B. Kahng et al. Noise and delay uncertainty studies for coupled RC interconnects. In *ASIC/SOC*, 1999.

[14] A. Krstic et al. Delay testing considering power supply noise effects. In *ITC*, pages 181–190, 1999.

[15] A. Krstic et al. Delay testing considering crosstalk-induced effects. In *ITC*, page 558, Oct. 2001.

[16] L. Lee et al. On generating tests to cover diverse worst-case timing corners. In *Intl. Symp. on Defect and Fault Tolerance in VLSI*, pages 415–423, 2005.

[17] L. Lee et al. On silicon-based speed path identification. In *VLSI Test Symposium*, pages 35–41, May 2005.

[18] F. Lu et al. A circuit SAT solver with signal correlation guided learning. In *DATE*, page 892, 2003.

[19] J. Qian et al. Modeling the effective capacitance for the RC interconnect of CMOS gates. *IEEE Trans. Computer-Aided Design*, 13(12):1526–1555, Dec. 1994.

[20] T. Sakurai. Closed-form expressions for interconnection delay, coupling, and crosstalk in VLSI's. *Trans. Elect. Dev.*, 40(1):118–124, Jan. 1993.

[21] S. J. Sheather and M. C. Jones. A reliable data-based bandwidth selection method for kernel density estimation. *J. Roy. Statist. Soc. B*, 53:683–690, 1991.

[22] B. W. Silverman. *Density Estimation.* Chapman and Hall, 1986.

[23] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, 2nd edition, 1999.