General Framework for Removal of Clock Network Pessimism

Jindrich Zejda
Synopsys, Inc.
700 East Middlefield Road
Mountain View, CA 94043, U.S.A.
+1 650 584-5067
zejdaj@synopsys.com

Paul Frain
Synopsys, Inc.
Blanchardstown Corporate Park
Dublin 15, Ireland
+353 1 809-8816
pfrain@synopsys.com

Abstract

The paper presents a simple yet powerful general theoretical framework and efficient implementation for removal of clock network timing pessimism. We address pessimism in static timing analysis (STA) tools caused by considering delay variation along common segments of clock paths. The STA tools compute setup (hold) timing slack based on conservative combinations of late (early) launching and early (late) capturing arrival times. To avoid exponential-time path-based analysis the STA tools use both early and late arrival times on gates common to both launching and capturing paths. It is impossible in real circuit and is observed as the clock network pessimism in STA. Our approach supports any kind of delay variation though the typical causes of the pessimism are process, voltage, and temperature on-chip variation, and reconvergence in clock network. We propose a new theoretical framework that allows to apply known graph algorithms instead of time consuming forward and backward multi-pass tracing algorithms and heuristics that are limited to some network topologies [4]. The new graph-based framework supports clock networks of virtually any size and type, e.g., tree, mesh, hybrid, clock gating, chains of multipliers and dividers, loops in such chains, etc. The implementation based on the proposed framework has proven its strength in a commercial sign-off static timing analyzer and thus is helping hundreds of designers to achieve faster clock speeds of their chips.

General Terms

Verification, Design, Algorithms.

Keywords

Clock network reconvergence, static timing analysis, process, voltage and temperature delay variation, deep sub-micron.

1. Introduction

In static timing analysis, delay variation in clock network contributes to unwanted pessimism that skews the actual timing properties of the circuit. This results in “leaving performance on the table”, i.e., concluding the circuit can operate at lower frequency than the actual silicon implementation.

The paper is organized as follows: Section 1 defines the terminology, explains the difficulty in removing clock network pessimism in STA, and surveys the existing solutions. Section 2 describes the new graph-based framework. It also shows how dominator and common predecessor algorithms are used for pessimism removal in this framework, and how it interacts with the STA engine in slack calculation. Section 3 describes the practical implementation and experimental results on several state of the art ASIC designs taped out by various vendors in past 2 years. Pros and cons are discussed in Section 4, and Section 5 lists references.

1.1. Terminology - Circuit Description

For any sequential timing check (e.g., setup, hold, clock gating, recovery) we define launching clock path, data path and destination clock path (Figure 1). The launching path is the path from the clock source to the clock pin of the register that launches the data path signal. The data path is the path from the clock pin of the launching register to the data pin of the register where the sequential check is performed (called capturing register). The capturing clock path is the path from the clock source to the clock pin of the capturing register. Then satisfaction of the timing constraint is determined from the arrival times along the launching clock path, the data path, the destination clock path, and the constraint itself.

It is fairly straightforward to evaluate a constraint under the assumption that all delays are constant. In practice, however, an STA tool must consider many instances of the physical circuit in

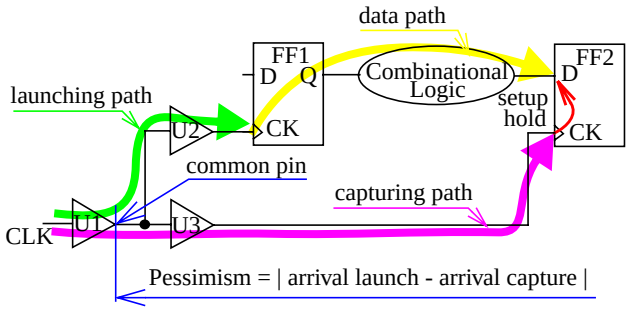

Figure 1: Clock network pessimism

its model. That is achieved by modeling delay variation, by e.g., interval delay model or delay distribution [5, 7]. In the interval delay model each gate, net, and constraint is assigned two delay values: minimum (shortest) and maximum (longest). The delay distribution model assigns probability to each value of delay, usually as a probabilistic function for a continuous range of delay [8].

When evaluating, e.g., setup constraint on pin FF2/D the longest delay along the launching path, the data path, the longest constraint, and the shortest delay along the capturing path are used.

1.2. Pessimism on Common Segment of Clock Network

But a problem arises with buffer U1. It was assigned both the shortest and the longest delays at the same time which is physically impossible. Actually, the launching and capturing clock transitions may propagate at the same time only for zero-cycle hold checks. But in general, the launching and capturing transitions are one or at most few clock periods apart. Since the largest delay variations are not caused by any effects that change during few clock periods[1] they appear as a pessimism in STA model when compared to the physical circuit. Without pessimism removal the STA tool reports worse violation than the physical circuit has and forces synthesis and optimization tools to waste effort on paths that meet the target clock frequency.

It is easy to remove the clock pessimism in our simple example in Figure 1. The pessimism is the max delay of U1 minus the min delay of U1. In general, it is the difference of the latest and earliest arrival times at the last common pin of the launching and capturing clock paths (Figure 2, or [4]). The clock pessimism can be removed by adding its absolute value to the final timing slack of the constraint.

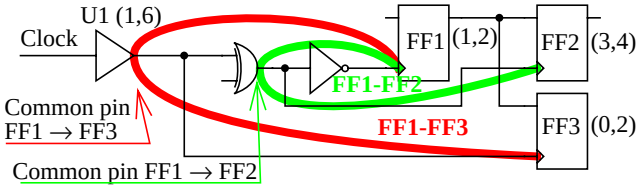

Figure 2: Common pin of the clock network

1. E.g., period-to-period clock jitter is thus excluded from pessimism removal.

1.3. Difficulty in Pessimism Removal in STA

The real problem is that the amount of clock pessimism that needs to be removed is path-specific. E.g., in Figure 2 register FF1 launches signal that is captured by both FF2 and FF3. The common pin and thus the clock pessimism are different for each capturing register.

Example: Assume that all combinational gates in Figure 2 have shortest delay of 1, longest of 6, denoted (1, 6). All registers have delay clock to Q of (1, 2), and setup constraint of (3, 4); except FF3 which has setup (0, 2). Data path has delay of (3.5, 5), nets have zero delay (0, 0). Clock period is T. Then timing slack on FF2 is T+1+1 - (6+6+6+2+5+4) = T-27. Common point clock pessimism is 6+6 - (1+1) = 10, and slack after removal of pessimism is T-17. On FF3 slack is T+1 - (6+6+6+2+5+2) = T-26. Clock pessimism is 6 - 1 = 5, and slack after removal of pessimism is T-21.

Note that the critical path without clock network pessimism removal is FF1→ FF2. The real critical path once the pessimism is considered is FF1 → FF3. Because in deep submicron data paths are shorter, clocks are faster, and clock networks are longer to accommodate larger chips, it is essential in STA to consider such pessimism. Not only to get more accurate path timing, but also to identify the critical paths.

1.4. Existing Solutions

Many other methods exist for calculation of delays considering min-max variation and their correlation (starting with [7]), and even for exact delay calculation under such conditions (the first comprehensive framework was [5]). But, first, these methods address only delays in data path. Second, even without extending for clock network, they are very complex in terms of CPU time which makes them unsuitable for full chip STA.

The existing practical solutions for removal of clock network pessimism fall into two categories: path-based and backtracking-based. The *path-based* solution is the most straightforward. Critical paths are identified without considering the pessimism first. Then for each path the common pin is found by a simple walk through the launching and capturing segments of the path. Finally, slack of each path is adjusted by the amount of pessimism on the common pin. The obvious disadvantage is runtime since the number of paths is exponential. Thus practical implementations must rely on limiting the number of tested paths, therefore may not be able to identify the most critical paths at all. The advantages are the simplicity and that it can be integrated with any STA tool as a postprocessing step.

The *backtracking methods* rely on modified arrival time propagation and backtracking to identify the common pin. An excellent description of such a method is given in [4]. The key idea is to perform initial timing analysis to identify potential common pins (called CCPPS, or critical common path point set in [4]). For each violating slack backtracking is performed to identify which CCPPS the worst arrivals passed through. The arrival time is propagated then again forward and is labeled by the CCPPS the trace passes through. The CCPPS defines bounds on pessimism included in the arrival time at any point on the data and clock paths. Slack calculation on registers or other constrained

points is done as usual by comparing early/late arrival and required (arrival along the capturing clock path) times. Then slack is increased by the amount of pessimism. If the arrival and required times have same CCPPS then the right-most CCPP determines the pessimism. If however, they are not (which is common in complex circuits), a backtrace is needed to identify the right-most CCPP. This process of several forward and backward tracing is repeated iteratively for all slacks from the initial timing analysis. The iteration stops when the overall worst slack is positive or the last computed slack adjusted for clock network pessimism is worse than the next slack without pessimism removal. The advantages are that paths do not need to be enumerated, and bounds on the pessimism are known for any pin in the design. The disadvantage is the costly processing of one slack at a time, each with multiple forward and backward traces. Further, some of the traces are exponential for non-tree clock networks. Also, as [4] points out, finding CCPPS is costly. Therefore [4] proposes heuristics and manual user input to eliminate this bottleneck.

In past we have used a path-based solution. As mentioned earlier, it was extremely slow, or even never finished on some designs, or did not always identify the critical path thus preventing static timing sign off.

Today's circuits are large (tens of millions of gates), have large clock networks (hundreds of thousands of gates), and long chains of dependent clocks that are derived from each other by dividers and multipliers (an example is node 5 in Figure 5). Such chains can be very long in system on chip (SOC) designs, usually 5 to 100 clocks per chain. Such clock circuits cannot be handled by backtracking methods because the CCPP sets would have hundreds or thousands elements and the number of needed backtracks becomes unacceptable.

2. Solution

Based on our past experience with sizes and types of circuits that need to be handled by an STA tool it was clear that path enumeration or backtracking based algorithms are not an acceptable solution. The key idea of the new solution is to eliminate the path tracing and backtracking and perform as much work up front, without timing the entire design.

2.1. Overview of Pessimism Removal Engine

The steps involved in STA with pessimism removal are described in Figure 3.

First, the circuit clock network is represented by a directed cyclic graph. Gate pins are nodes of the graph, nets and gates are edges. The graph is reduced as explained later in Section 2.2. The reduction leaves only potential common nodes and some key nodes such as clock sources and register clock pins. Arrival times are propagated in a single forward pass while treating the potential common nodes as through points of timing exceptions as described in [2]. Once the arrival times on register clock pins and data pins are known then the timing engine evaluates all relevant [2] combinations of arrival and required times. The arrival time propagation is similar to the second forward propagation in [4]. However, instead of labeling the arrival times with a pointer to the set of all potentially common nodes the arrival trace passes through, only a reference to a representative register clock pin node in the clock network graph is used. Thus any slack calculation uniquely identifies the launching and capturing groups of register clock nodes. Then the closest common clock node is identified by graph operations as explains in Section 2.3. This is the most difficult part and the main focus of the paper. Once the common node and transition sense on it are known, the min and max arrival times on the node are used to calculate clock pessimism. The clock pessimism is added to the timing slack. This approach theoretically takes more memory than [4] since more slacks need to be calculated. As discussed later, the increase is relatively low in practical circuits. The main benefit is that the powerful combination of node grouping, arrival propagation [2], and dominator analysis eliminates backtracking and the iterative search for the "next worst slack" which results in faster performance.

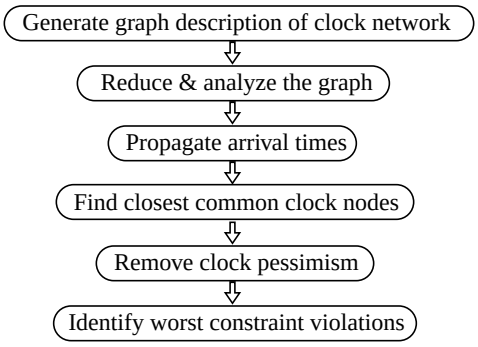

Figure 3: Overview of the pessimism removal engine

2.2. Graph Description of Clock Network

The Clock Network Graph (CNG) is built to represent complicated clock network as simply as possible, therefore only the following objects are represented as nodes in the CNG:

1. Register clock pins
2. Clocks
3. Potential common nodes (pins with fanout >1)
4. Clock dividers and multipliers

Figure 4 below shows how the circuit in Figure 2 would be represented as a CNG.

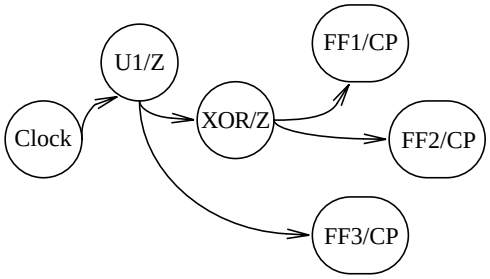

Figure 4: Circuit of Figure 2 represented as a CNG

The CNG is built by tracing forward from all clocks in the design and adding nodes for pins that fit the previously mentioned conditions. Together with topological information the CNG also stores on each node a record of what edges (rising, falling, or both) result from individual clock edges.

An example of full CNG for a small circuit is shown in Figures 5 and 6. The clock node 1 is the start node of the graph. Leaf nodes such as 8 and 9 represent register clock pins. Node 5 is clock of half the frequency of clock in node 1.

The graph is reduced in several ways. First, nodes that do not drive register clock pins, clock gating cells, or chains of generated clocks are eliminated. Second, nodes are grouped according to their arrival time distance from their predecessors. The maximum arrival time difference among all the nodes in each group determines how much pessimism is not removed (per path; they do not accumulate). The limit is fully controllable and thus allows the user to further trade off speed for pessimism removal. A reasonable limit is few picoseconds for designs below 1 GHz. The grouping step is essential and greatly reduces the number of potential common nodes that the arrival propagation engine has to consider. The key advantage is that only the clock network needs to be timed. There is no need to time the entire circuit (no datapath or slack calculation is done).

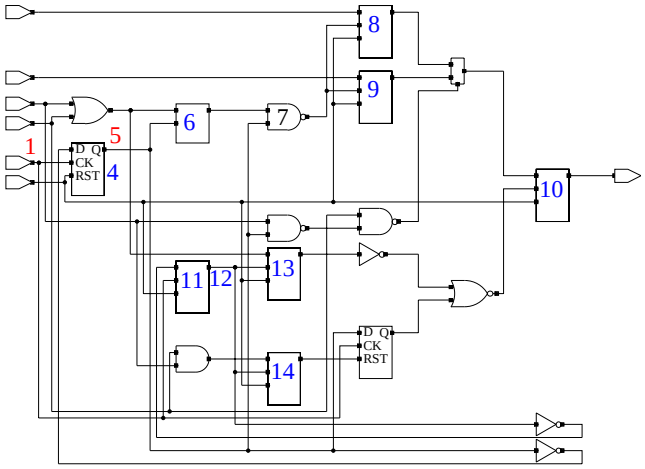

Figure 5: Sample circuit - schematic

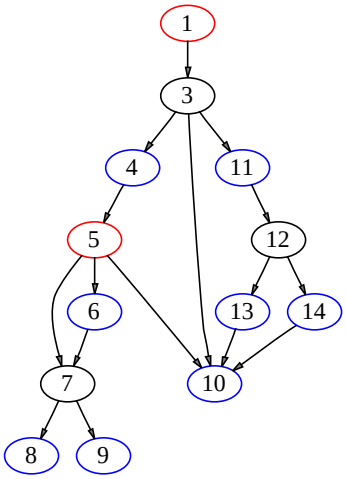

Figure 6: Sample circuit - clock network graph

2.3. Finding Common Clock Nodes

The graph representation of the circuit allows us to translate topological search in the circuit onto a graph problem. In graph theory the closest common clock node can be defined based on the paths from start nodes to register clock pin nodes or, as we show later in this section, based on graph dominators. In this section we define a theoretical graph-based framework for identification of common clock nodes. The framework allows us to quickly answer question "given 2 register clock pins, what is the common clock node".

2.3.1. Theoretical definition of common clock node

The clock network circuit is described by graph $G = (V, E, S)$, where V is set of vertices (nodes), E set of edges, $E \subseteq V \times V$, S set of start nodes, $S \subseteq V$.

Path $p(v_1, v_m)$ between two nodes v_1, v_m in graph G is defined by a set of nodes $(v_1, v_2, \ldots, v_m)$ and edges $E_p \subseteq E$ such that $e(v_i, v_{i+1}) \in E_p$, $1 \le i < m$, and $i \ne j \Rightarrow v_i \ne v_j$, $1 \le j < m$. The m is the length of path p.

For each node v_i on path p we define distance d from the final node v_m on the path, $d(p, v_i) = m - i$.

Intersection of n paths Π is a set of nodes common to all the paths, $\Pi(p_1, \ldots, p_n) = \{v : v \in p_1 \wedge \ldots \wedge v \in p_n\}$.

Then closest common clock node $\gamma(R)$ of set of nodes $R \subseteq V$ is the node in the intersection of all paths starting in S and ending in R, closest to the final node of any such path containing it:

$$\gamma(R) = \left\{ \begin{array}{l} v\text{: } \min d(p_i, v), v \in \Pi p_i \\ \forall p_i = (s, \ldots, r), r \in R, s \in S \end{array} \right\} \quad (1)$$

For a given launching register clock pin (node L) and capturing register clock pin (node C) the closest common clock node is then $\gamma(\{L, C\})$, denoted $\gamma(L, C)$. The (1) also defines a simple path-enumeration method to compute $\gamma(L, C)$. This, indeed, is not practical since the set P of all paths in the circuits is exponential in size, $P \subseteq E^{|V|}$, or $P_s \subseteq S \times E^{|V|-1}$ for paths from start nodes.

Let's relax (1) by removing the requirement of minimum distance and reducing size of R to a single node r. Then $\gamma(R)$ is reduced to the definition of set $\delta(r)$ of dominators of node r:

$$\delta(r) = \{v\text{: } v \in \Pi p_i, \forall p_i = (s, \ldots, r), s \in S\} \quad (2)$$

Dominator sets define a new relation in the graph, "being a dominator of" or dominance Δ: $\Delta \subseteq V \times V$. Dominance is reflexive, antisymmetric, and transitive.

Dominator graph Φ is a graph representation of Δ:

$$\Phi(G) = \left\{ \begin{array}{c} (W, F, T) : W \subseteq V, F \subseteq E, T \subseteq S \\ f(w_1, w_2) \in F : w_1 \in \delta(w_2) \end{array} \right\} \quad (3)$$

Dominator graph $\Phi(G)$ is a forest for any non-trivial graph G because relation Δ is antisymmetric and transitive for an arbitrary graph. In literature, Φ is usually referred to as "dominator

tree" because most algorithms only need to consider cases with a single start node, or root. For clock network analysis there are usually multiple start nodes corresponding to individual clocks, hence the forest.

Assume that we can construct the dominator graph (3). Then the closest common clock node (1) can be defined using dominator definition (2) as follows:

$$\gamma(R) = \left\{ \begin{array}{c} w : w \in \bigcap_{\forall r} \delta(r), r \in R, w \in W \\ \text{and } \forall w' \in \bigcap_{\forall r} \delta(r), w' \le w \end{array} \right\} \quad (4)$$

where the $w' \le w$ is an ordering relation defined on $\Phi(G)$ based on the distance from the start nodes T. Since $\Phi(G)$ is a forest then there is exactly one path $\pi = (t, \ldots, w', \ldots, w, \ldots, r)$, $t \in T$, $r \in R$, and $d(\pi, w') > d(\pi, w)$.

The (4) tells us how to find the closest common clock node γ: first identify dominator forest, then intersect dominator sets of register pins and find the dominator closest to the register pins.

2.3.2. Example of dominator based search for γ(L,C)

An example of closest common clock node is given in Figure 7. Note that it may seem counter-intuitive to calculate γ(5, 7) = {1} rather than 2. Unlike a simulator, STA tools typically cannot analyze all possible reconvergent arrival times thus the signal arrival on node 4 would be determined by both paths through nodes 2 and 3 in the STA model of the circuit. Hence definition of the γ ensures that STA with pessimism removal remains conservative.

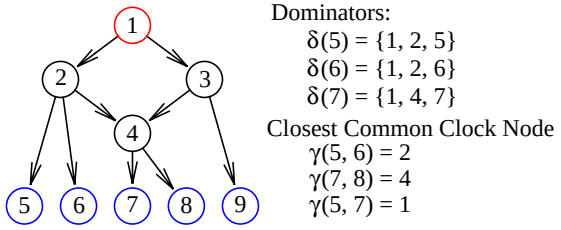

Figure 7: Example of closest common dominator

Computing dominators alone is, however, costly. Despite of being a well understood common tool in design of software compilers, there is no practical linear-time, linear-memory, low-overhead algorithm. Compared to basic blocks in software compilers the clock network analysis is very different. First, software programs have tens or hundreds of blocks for which dominators need to be calculated. Our needs are several orders of magnitude higher. Second major difference is that we do not need to compute full dominator graph, but per (4) only relevant subsets $\delta(L)$ and $\delta(C)$, i.e., dominators of register clock pins.

Several powerful dominator algorithms exist. There are even linear time algorithms such as [1] which is however a purely theoretical study since to achieve linear behavior in time it requires exponential amount of memory. Our goal is not to search for the best theoretical dominator algorithm but rather use and extend strengths of existing algorithms to solve a very hard practical problem in hardware verification. If better dominator algorithms are invented in the future, they can be plugged into our engine as a new dominator solver.

2.4. Practical Dominator and Common Node Analysis Engine

It is not practically possible to compute all the closest common clock nodes for all pairs of register clock pins. First, most dominator algorithms require too much memory. Second, not all pairs of registers have data path between them.

The first problem is addressed by limiting the search space in which dominators are computed. We use a partial on-demand dominator analysis. The γ is always computed for only two nodes at a time, but as an (intended) side effect the search can produce dominators for other nodes. The second is addressed by on-demand calculation driven by the timing engine arrival time tracing procedure with aggressive caching.

We use multiple solvers to get the γ. Each solver either finds the solution or indicates that it cannot operate on the given circuit topology. In such case the pair of register clock pins is passed to the next solver.

2.4.1. Trivial case solver

Trivial closest common clock node solver is used when the two register clock pins *L* and *C* have a common predecessor node and fanin of 1. Typical example are register clock pin nodes 7 and 8 in Figure 7. It correspond to the same buffer driving both register clock pins.

2.4.2. Tree based solver

This solver is used for H-type clock trees or hybrid mesh-H (high-level mesh, lower level H tree) clock networks but not hybrid H-mesh. The hybrid mesh-H clock networks must have the closest common clock node within the tree part of the network.

The advantage of the multiple solver architecture is that no explicit knowledge of the type of clock network is needed up front. The tree solver either finds the γ or detects that the relevant part of the clock network is not a tree.

Assume clock network in Figure 8. The top node (1) is a clock node. The leaf level nodes are register clock pins (7 to 13). The γ(7, 10) can be found by traversing backward from each register node until any common node is found. In this case the paths are 7, 4, 2, 1 and 10, 5, 2, 1. The first common node is 2 which is the γ(7, 10).

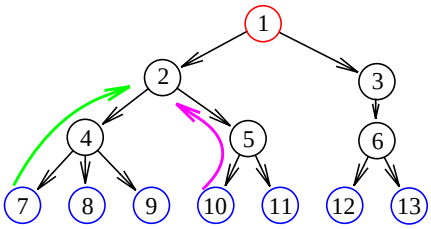

Figure 8: Tree clock network

A general solution for finding closest common predecessor in trees is described in [3]. However, it is a solution for arbitrary

number of nodes. We only need closest common node for two registers therefore the implementation complexity of [3] can be easily avoided. Our algorithm is described in Figure 9. It is a simple backward graph traversal of a single pair of paths. The complexity of the search is linear in the length of the path from register node to the γ. That means that the worst complexity for the entire circuit is bounded by length of the longest clock path and the number of register clock pin pairs. That indeed would be unacceptable in practice. In practice, the average complexity for the entire circuit is linear in the number of register clock pins (not pairs of pins) because of both caching of γ and use of an STA approach.

```
While exist predecessors {
  extend path from launch node backward
  if the reached node is on the backward capture path
    return node
  extend path from capture node backward
  if the reached node is on the backward launch path
    return node
  if any reached node has >1 predecessor then
    return not_a_tree
}
return no_γ
```

Figure 9: Tree-based common clock node solver

Note that, in example in Figure 8, node 1 is never visited. In this case node 2 is the γ, thus the search stops there. Therefore the fast tree algorithm works even on hybrid mesh-H clock networks. If the γ is not in the tree portion of the clock network then the next solver is used.

2.4.3. Arbitrary graph solver

The solver is based on the basic Tarjan-Lengauer algorithm (TJ, [6]). The advantage of using TJ is that it can handle any directed graph - including graphs with loops. It may sound as an excessive requirement for clock network analysis tools but many practical circuits do have loops in the clock network graph. The reason are paths shared for both clock and data, clock gating and especially clock shaping (divider and multiplier) circuits with their feedback logic. Handling arbitrary directed graphs is an essential feature.

The TJ performs first a DFS to identify semi-dominators (sets of nodes that include all dominators but also some non-dominators) and then refine them iteratively based on immediate dominator relation on each node.

The TJ in its simplest form has memory complexity square in the number N of graph nodes but good speed - average CPU complexity is roughly[1] O(N log(N)), worst $O(N^2)$. In graphs that describe practical clock networks the memory complexity is the limiting factor.

One natural way to cope with the TJ memory complexity is by identifying the smallest possible relevant part of the clock network graph for the two given register nodes and then apply TJ algorithm. That works fine for individual pairs of registers. However, the identified space for any register pair usually overlaps with the space for another register pair. Thus average complexity of the calculation for the entire design would be $O(N^2 \log(N))$ assuming that the STA timing engine analyzes about O(N) register pairs which is the case for full chip in most cases. Though it might not be the case for individual tiny subblocks like barrel shifters, multipliers or graphics processors - but for us is important only the relationship for the entire analyzed chip.

One way to reduce the overall complexity is to give up some of the search space reduction with the overall speed goal in mind. The search space is extended to the fanout of the intersection of fanin cones of the registers (Figure 10).

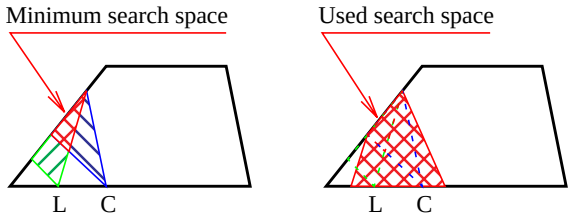

Figure 10: Reduction of search space

The computed dominators $\delta(L)$, $\delta(C)$ and closest common clock nodes $\gamma(L, C)$ are cached independently. Thus one TJ dominator calculation results in knowing dominators of all nodes in the search space. Then any other γ calculation in the same search space lookups up dominators and finds the γ by ordering dominators as defined in (4).

You can imagine that there are clock network topologies for which each search space is easily 30 to 90% of the clock network. That can mean tens of thousands or even millions of clock nodes. The basic TJ algorithm cannot handle such cases.

If the number of nodes in the search space is greater than about hundred then we further reduce the search space by mapping part the physical search space onto a virtual search space. The virtual search space represents various subgraphs of the graph by a smaller equivalent representation. An example of such mapping rule is that a fanin cone of node is replaced by set of its dominators if they are already known. The virtual space used during the search in Figure 10 could then map to the corresponding physical space depicted in Figure 11.

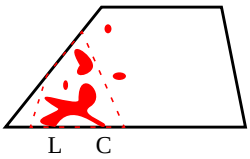

Figure 11: Virtual search space

Note that the theoretical worst case memory complexity of this solver is again $O(N^2)$. However, the worst case is not a practical circuit. For example, to construct a circuit that is equivalent to full graph, all drivers in the clock network would have to be bidirectional and have fanout of N each. We have encountered clock networks with many bidirectional drivers, and also nodes with fanout of tens of thousands (e.g., a block that does not have

1. Roughly - because the complexity depends on the type of graph. We consider average practical types of clock networks, thus abstracting from the number of edges in the graph.

clock network synthesized yet). But such features cannot exist on all nodes, neither they can all be combined in one circuit to be a practical clock distribution network.

2.5. Transition Sense Propagation

To remove the pessimism due to delay variation we use stricter requirements than that the clock signal passes through the common pin γ as was shown in Figure 1. It must also pass through the same transistor (pull up or pull down) of the output pin of the gate. That translates to a requirement of same (rising or falling) transition on γ in the gate-level representation (Figure 12).

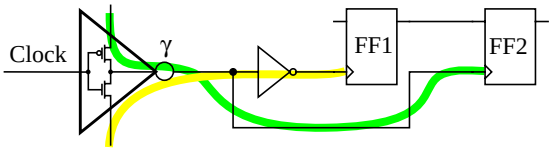

Figure 12: Mismatching transition senses on γ

2.5.1. Finding transition sense on γ

The transition sense on register clock pins is known. For rising edge-triggered register the sense is rising, for falling edge-triggered register the sense is falling, for level-sensitive latches we use a set of both rising and falling.

The sense relative to the clock is precomputed at the time of creation of the clock network graph. The absolute transition sense on the γ is computed from the clock phase and the relative sense between the clock and the γ and the relative sense between the clock and the register clock pins.

2.5.2. Delay correlation

Even for mismatching senses some pessimism can be removed. We provide an option to consider a simple correlation as shown in Figure 13 because it is conservative, easy to understand and implement. Indeed, it is conservative as long as such correlation exists. Therefore many silicon vendors adhere to the strict (same transition sense) correlation model in CMOS circuits. They use the correlation model in Figure 13 only for removal of pessimism due to wire delay of tester probes. The presented framework for removal of clock network pessimism is independent of the correlation model. Any model can be used.

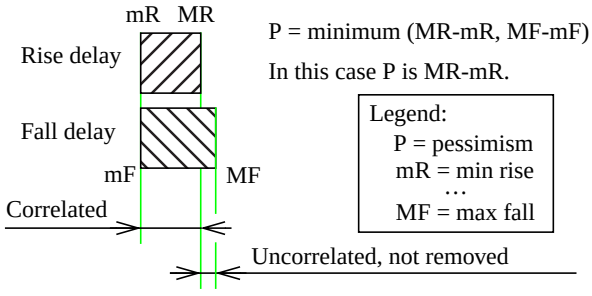

Figure 13: Simple rise/fall gate or path delay correlation

3. Experimental Results

The clock network pessimism removal engine was implemented in C as a part of a commercial STA tool. The results are presented for a set of customer designs on hand that were provided to us for testing the tool or as testcases for bug reports. The designs include a 128-bit graphics chip, signal processor, MPEG decoder, and various chips with large clock networks and complicated clocking schema. The purpose of several other chips is not known to us. Many more designs that had CPU time and memory problems with the path-based solution were run off-site by individual customers to their satisfaction. CPU time results are for UltraSparc-III 750MHz processor (Table 1). The "CPU time for γ" is the key measurement. It is the CPU time for building CNG, grouping nodes, finding dominators, finding common clock nodes, propagating sense, and computing pessimism. Since it does not include arrival time and slack calculation the total CPU time is important as well. The "# of pins in the clock network" is the size of the problem that the γ algorithm has to deal with. The total memory and CPU time are for complete run of the tool including reading design files, delay calculation, and reporting. The very last column is the total CPU time for our previous path-based solution. The ∞ in that column means that the run could not finish in 3 days. We do not present any results for commonly used ISCAS or MCNC circuits because of two reasons: 1) they are too small, 2) in their most publicly used form they do not have synthesized clock network, gate and net delay, or post-layout parasitics.

Table 1: Experimental results

Circuit	# clocks	# gates	# pins	# pins in clock network	CPU time for γ [s]	Total CPU time [s]	Total Memory [MB]	Path-based CPU [s]
d1	19	320921	1384933	42450	159	5047	1444	∞
d2	6	72142	304043	14873	3	74	227	122
d3	28	295012	1228821	148879	115	762	357	∞
d4	1	143142	686867	21022	178	718	408	∞
d5	37	198254	682102	29159	5	380	235	54377
d6	7	48177	179525	6732	11	313	539	∞
d7	2	64611	247007	9558	2	110	117	303
d8	4	28687	121463	5659	1	65	55	2603
d9	4	30038	136324	5452	2	62	83	1466
d10	19	320921	1384933	42450	157	5011	1445	∞
d11	1	37158	271327	187622	429	727	442	13982
d12	24	360449	1529113	66751	15	2122	898	∞
d13	1	45757	154824	14000	2	41	62	28989
d14	26	285697	963358	30548	6	267	283	24246
d15	16	851464	3109263	123057	44	3560	952	∞
d16	10	500966	1886949	54566	10	858	558	33405
d17	1	35965	267498	187574	120	239	473	∞

The experimental results show very good performance of the pessimism removal solver. In many cases most closest common nodes are found by the trivial or tree-based solver (cases where CPU time is around 10 seconds). In all other cases a mix of all solvers is used because the designs have more complex meshed or hybrid clock networks involving thousands of nodes. The total runtime is much faster compared to the previous path-based solution. Since no implementation of [4] was available to us we were not able to compare the runtime. The predicted performance is that [4] is faster on small circuits, and significantly slower on medium and large ones.

An important question is how much pessimism is removed. An example of a large circuit with many violations is shown in Figure 14. The endpoint slacks for all checks including setup, hold, clock gating, … are binned according to their slack into 10 bins. The left-most bin is the worst slack, -4.5ns to about -4ns, the right-most bin is the best slack, about -0.4ns to 0. Non-critical endpoints are not shown. The figure shows that several thousand violations were due to clock network pessimism. Also the overall design worst slack has improved by at least 10% once the clock network pessimism has been removed (the left-most bin is empty). The number of violating endpoints decreased from 91937 to 26564. Similar improvement in slack was observed on other designs. In all 17 designs, the number of violating endpoints decreased from 352296 to 169048, i.e., to less than half.

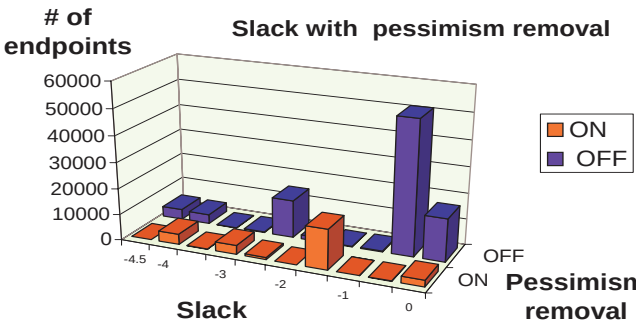

Figure 14: Violating endpoints with and without clock network pessimism removal for circuit d1

4. Conclusions

We have presented a new theoretical framework and a practical solution for fast removal of pessimism in clock network due to on-chip variation of delays in static timing analysis. The key idea is to initially time only clock network, describe it by abstract CNG model, and perform node grouping up front. Then propagate sufficient number of arrival times identifying groups of registers with significant amount of clock network pessimism. The difficult part, identification of the common points in clock network during slack calculation is addressed by a systematic approach based on dominator graph algorithms which ensures support of non-tree clock networks. The second most difficult part, the identification and analysis of the “next worst slack” (which the path-based and backtracking [4] methods suffer from) is solved by a combination of up front analysis of clock network graph and use of a timing exception-aware arrival propagation [2].

On a set of real customer designs the new method far outperformed the previous path-based solution and showed significant reduction of pessimism. Compared to backtracking methods [4] the main advantage is elimination of time consuming multiple forward and backward tracing, elimination of the iteration over “next worst slack”, ability to remove pessimism even on non-violating paths which is useful for budgeting, and support of meshed and hybrid clock networks. The advantage of [4] is that bounds on pessimism are included in arrival times on each pin which may help during in place optimization (IPO).

5. References

[1] Alstrup, S., Harel, D., Lauridsen, P.W., Thorup, M. Dominators in linear time. SIAM Journal on Computing (Vol. 28(6), 1999), 2117-2132.

[2] Craven, T.L., Baylor, D.M., Rindenau, Y. Static timing analysis of digital electronic circuits using non-default constraints known as exceptions. United States Patent 6,237,127 (May 2001).

[3] Harel, D., Tarjan, R.E. Fast algorithm for finding nearest common ancestors. SIAM Journal on Computing (Vol. 13(2), May 1984), 338-355.

[4] Hathaway, D., Alvarez, J. P., Belkbale, K. P., Network timing analysis method which eliminates timing variations between signals traversing a common circuit path, United States patent 5,636,372 (June 1997).

[5] Lam, W.K.C., Brayton, R.K. Timed boolean functions - a unified formalism for exact timing analysis. ISBN 0-7923-9454-2 (Kluwer Academic Publishers, 1994).

[6] Lengauer, T., Tarjan, R.E. A Fast algorithm for finding dominators in a flow graph. ACM Transactions on Programming Languages and Systems (Vol. 1(1), July 1979), 121-141.

[7] Sivaraman, M., Strojwas, A.J. Accurate timing verification with correlated component delays. Workshop on Design Methodologies for Microelectronics and Signal Processing October 1993).

[8] Zasio, J.J., Choy, K.C., Parham, D.R. Static timing analysis of semiconductor digital circuits. United States Patent 4,924,430 (May 1990).