TKtimer: Fast & Accurate Clock Network Pessimism Removal

Christos Kalonakis*, Charalampos Antoniadis*, Panagiotis Giannakou*, Dimos Dioudis*,
Georgios Pinitas† and Georgios Stamoulis*
*Department of Electrical Engineering
University of Thessaly
{hrkalona, haadonia, panagian, ntioudis, georges}@inf.uth.gr
†Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
{g.pinitas}@student.tudelft.nl

***Abstract*—As integrated circuit process technology progresses into the deep sub-micron region, the phenomenon of process variation has a growing impact on the design and analysis of digital circuits and more specifically in the accuracy and integrity of timing analysis methods. The assumptions made by the analytical models, impose excessive and unwanted pessimism in timing analysis. Thus, the necessity of removing the inherited pessimism is of utmost importance in favour of accuracy. In this paper an approach to the common path pessimism removal timing analysis problem, TKtimer, is presented. By utilizing certain key techniques such as branch-and-bound, caching, tasklevel parallelism and enhanced algorithmic techniques, the approach described by this paper is able to handle any type and size of clock network trees and showed 100% accuracy combined with reasonable execution time within a straightforward solution context**

I. Introduction

In static timing analysis, the use of minimum and maximum delay cases (early and late mode analysis) to account for the impact of process variation can impose unnecessary pessimism on the evaluation of the circuits timing characteristics. More specifically, the dual mode analysis on certain parts of the clock network which are the common sub-path of data paths and clock paths, causes an overly pessimistic evaluation of the delay characteristics because the shared portion of the path between the data path and the clock path cannot be affected by both early and late mode operation impact. The main purpose of the Common Path Pessimism Removal (referred as CPPR) Static Timing Analysis is to remove this unnecessary pessimism imposed by the dual mode analysis. The common path pessimism removal problem has been identified as highly significant as the previous work on the field suggests. [7] examines the significance of CPPR for ATPG testing, [8] [9]propose a method for pessimism removal on clock networks of FPGSAs, while [4] proposes a method for hierarchical CPPR timing analysis. The proposed implementation is able to handle any type of clock network graphs. By making use of the dominator graph extraction the former can handle more complex cases of clock network trees[paper]. Furthermore, task level parallelism and the various further optimizations, allow TKtimer to provide high accuracy and perform reasonably well under a wide range of inputs. A detailed description of the above terminology used herein is provided in [1] [10].

In the rest of the paper, the software architecture is presented in section 2 along with detailed description of each part that it consists of. In section 3, the key features used for optimizing the performance and accuracy of the presented implementation are described. Section 4 describes the evaluation of the tool and finally, section 5 contains the conclusions.

II. Software Architecture

In the proposed tool, after arrival time propagation and required arrival time initialization, the setup/hold tests are performed and the relevant credits are computed. The amount of pessimism to be removed from each path examined both in setup and hold tests is path specific and is denoted as the aforementioned credit. After performing common path pessimism removal, the tests are reported according to their new criticality.

The software architecture block diagram of the proposed implementation is depicted in Figure 1. The shaded blocks form the static timing analysis engine of the tool. The other blocks are support IO modules responsible for the parsing of the input files and the formation of the output reports of the tool respectively.

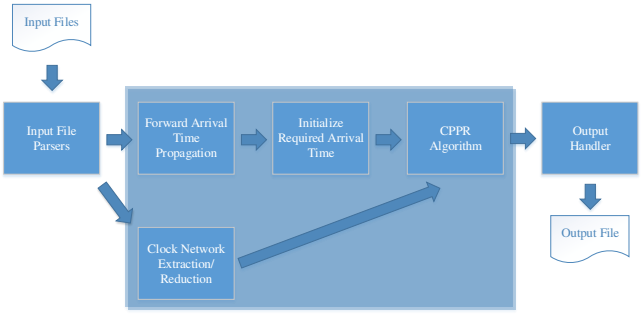

Fig. 1: Software Architecture

A. Input File Parsers

Initially, the implemented tool parses the given input files according to the TAU contests specifications and generates the internal representation of the circuit design, which is a bidirectional graph. Additionally, the parser identifies the clock source, assigns the initial arrival times in every primary input of the circuit, and forms a list of all the nodes of the circuit, the test pins list.

B. Arrival Time Propagation

Before calculating the slack at test pins it is required to determine the arrival time at those pins. Thus, the circuit graph is traversed breadth first and the arrival time information is conveyed from PIs to test pins. In late mode, the late arrival time will not be changed if the propagated arrival time is lower than the one already assigned to the adjacent node, while in early mode, the early arrival time will not be changed if the propagated arrival time is greater than the one already assigned to the adjacent node. After the end of the graph traversal, the latest/earliest time that a signal can reach every node of the circuit is known, from all the possible paths that converge onto them.

C. Clock Network Extraction/Reduction

The clock network graph (CNG) constitutes a simplified representation of complicated clock network structures. Before the application of CPPR algorithm the CNG must be isolated

Commencing from the identified clock source node, the tool traverses the circuit graph and elaborates the CNG. Nodes of the CNG can only be:

- Register clock pins
- Clocks
- Potential common nodes (pins with fanout greater than 1)

In order to improve the runtime, the clock network extraction and reduction, takes place in the same breadth first traversal that is used for the arrival time propagation.

D. Required Arrival Time Propagation

The required arrival times are determined by the circuit constraints, and along with the arrival times, indicate if the circuit can behave correctly under a specified clock frequency. Since the timing analysis is performed in late and early mode, the required arrival times are calculated by :

$$rat^{late}_{test\_pin} = T_{period} + at^{early}_{clk\_pin} - setup_{constraint} \quad (1)$$

$$rat^{early}_{test\_pin} = at^{late}_{clk\_pin} + hold_{constraint} \quad (2)$$

Where T is the period of the clock source, the arrival times used are from the clock pins of the sequential elements participating in the tests, and setup/hold are the constraints in the test pins. Our tool can calculate the slacks for setup/hold mode at each test pin, which are the pre-CPPR slacks for the whole test, using the required arrival times and arrival times, calculated in previous steps, by:

$$pre\_slack^{hold}_{test\_pin} = at^{early}_{test\_pin} - rat^{early}_{test\_pin} \quad (3)$$

$$pre\_slack^{setup}_{test\_pin} = rat^{late}_{test\_pin} - at^{late}_{test\_pin} \quad (4)$$

Since the arrival times calculated during the arrival time propagation where the earliest/latest, in this step our tool knows the worst slacks, for setup and hold, for each test so the aforementioned pruning can be performed to the examined test, if the slacks are positive.

E. CPPR Algorithm

The purpose of the CPPR algorithm is to remove the unnecessary pessimism that is introduced during the timing analysis in the presence of variation. Starting from a test pin, a backward depth first graph traversal is implemented, with the sole purpose of isolating one path at a time along with its slack, and also determine if there is a launching sequential element. The slack of the path in hold and setup mode are calculated by initializing

$$pre\_slack^{hold}_{path} = -rat^{early}_{test\_pin} \quad (5)$$

$$pre\_slack^{setup}_{path} = rat^{late}_{test\_pin} \quad (6)$$

and then adding/subtracting the delays of each timing arc across the path along with the arrival times at the primary inputs. At this point our tool has computed the pre-CPPR slack of a path along with the launching sequential element, if any. The next part of the process is to find the lowest common ancestor in the clock network tree between the capturing clock pin of our testing sequential element, and the launching clock pin of the sequential element, that the path passes through. If there is no launching clock pin, there is no common ancestor in the clock network tree, and in this case the credit is zero. In the case that a launching clock pin exists, the lowest common ancestor search algorithm is applied. A special case that arises when there is no launching sequential element, but the clock network tree reaches the test pin of the sequential element and the clock pin of the same sequential element, is handled by setting the launching pin to be the test pin. The algorithm traverses the clock tree backward both from the launching and the capturing node until it reaches a common node.

When a common node is reached, in the clock network tree, the credit is calculated by

$$credit_{hold} = at^{late}_{common\_point} - at^{early}_{common\_point} \quad (7)$$

$$credit_{setup} = credit_{hold} - (at^{late}_{clk\_source} - at^{early}_{clk\_source}) \quad (8)$$

where the arrival times at the clock source are the initial arrival times given by the input files. Having the credits for

hold and setup, the post-CPPR slacks, for each path, are calculated by

$$post\_slack_{path}^{hold} = pre\_slack_{path}^{hold} + credit^{hold} \quad (9)$$

$$post\_slack_{path}^{setup} = pre\_slack_{path}^{setup} + credit^{setup} \quad (10)$$

and total post-CPPR slacks for each test are calculated by,

$$post\_slack_{test\_pin}^{hold} = min_i(post\_slack_{path(i)}^{hold}) \quad (11)$$

$$post\_slack_{test\_pin}^{setup} = min_i(post\_slack_{path(i)}^{setup}) \quad (12)$$

Each path of a test, having a negative pre-CPPR slack, is stored in an ascending order, in a binary search tree, and the ordering is given by its post-CPPR slack. Each test, having a negative pre-CPPR slack, is stored in an ascending order, in a binary search tree, and the ordering is given by its post-CPPR slack. TKtimer stores the most critical tests, and within each test, the most critical paths. The reason behind using binary search trees, is that the ordering is kept intact during insertions and deletions of elements and their complexity is logarithmic. The CPPR algorithm can be executed either for setup mode, hold mode, or both modes. In the latter case the depth first graph traversal is only performed once to improve the runtime.

F. Output Handler

The output handler is responsible for shaping the output, into the form that was requested by the contest rules according to the TAU 2014 contest rules. More specifically the output handler is responsible for:

- Dumping the pre/post-slack of each path and all pins on this path from a given number of most critical paths for each setup, hold or both test under examination.
- Considering a given number of tests from the most critical tests

III. Key Features

Exploitation of the benefits that current multi-core architecture have to offer is of high importance for compute intensive problems.

CPPR can be compute intensive for complex designs and exhaustive in path tracing. For this reason a series of algorithmic optimizations which target multi-core architectures or reduce the problem space are deployed in order to reduce the required execution time. The most important optimization steps are presented and analyzed below.

A. Caching of Credit

The CPPR algorithm removes the pessimism of a given path, by adding credit to its computed slack. The credit is calculated by finding the lowest common ancestor, in the clock network tree, between the clock pin of the launching sequential element and the clock pin of the capturing sequential element, which belongs to the examined test pin. Each pair of capturing and launching nodes is not unique, and can be found numerous times during the same test. Thus, a cache-oriented approach is used to store the computed credit for each combination that is found for the first time during the test. When examining a single test, a cache is created that can store all the possible combinations, considering the tests capturing clock pin and all the available launching clock pins (including the capturing clock pin), and it is initialized as not calculated. Every single clock pin, belonging to a sequential element, has a unique identifier assigned to it, which can ensure constant time when accessing the cache. When a new pair is encountered for the first time, the cache is updated, and every time the same pair is identified during the test, the cache is accessed in order for TKtimer to acquire the corresponding credit without recomputation.

B. Multi-threading optimization

The CPPR algorithm removes the pessimism in every path that converges into a single test pin. Since the pessimism removal needs to be performed for every test pin of the circuit it becomes obvious that each test is completely independent from each other. Moreover, the fact that each test will not need any data from other tests makes the problem embarrassingly parallel. Each test does not require the exchange of any information with other tests, which means that the model is completely parallel. Initially TKtimer partitions the tests into chunks, one chunk for each thread, so that each thread can perform the CPPR algorithm for a number of tests, which has a significant positive impact on the runtime.

Contrary to intuition, partitioning the main problem into smaller sub-problems was not the optimum optimal way to use the multi-threading optimization, since every test has a variable number of paths; which means that performing the CPPR algorithm on different tests can yield poor scalability of the run times for each test, and can lead to work-unbalanced threads. A solution to the thread balance issue was to use a task-pool oriented approach; each thread gets the next available test from the pool and performs the CPPR algorithm on it. In this way all the threads can be occupied at all times. This technique achieved an even better speed-up in runtime. The parallelism was included during the steps of required arrival time initialization and the CPPR algorithm for each test.

C. Enhanced Breadth First Search (BFS)

During the stage of arrival time propagation, a breadth first graph traversal is used to propagate the latest/earliest arrival times in every node of the graph.

The basic notion behind breadth first traversal is that, a node should not be visited again if it was visited before during the traversal. That constraint led to erroneous arrival time calculation in some nodes, since the information that can be contributed from a new traversal to the already visited node, is

```
tests[NUM_TESTS]
j = 0

Thread i:
do
    cs
        test = test[j]
        j++
        if j ≥ NUM_TESTS then
            break
        end if
    endcs
    initialize rat(test)
    cppr(test)
while 1
```

Fig. 2: Multi-threading Algorithm

able to change the arrival times of the adjacent nodes. To fix this issue, the visited state check of the normal breadth first traversal is removed.

Using the new technique, our tool could calculate the correct arrival times for each node, but a major flaw emerged that increased the runtime. Even though a node would not change the arrival times of its adjacent nodes, those nodes could be revisited again for no reason. This flaw led to the reexamination of a whole sub-graph starting from the triggering node. As a solution to this emerged issue, an alternative logical condition is introduced when visiting a node; In more detail, a node should be visited if it has not been visited before or the arrival times contributed change the old ones, starting a chain reaction from this node, to the whole sub-graph. In this way the arrival time propagation algorithm would calculate the correct arrival times with the minimum runtime.

```
Queue Q
enqueue all inputs to Q
set all inputs as visited

while Q is not empty do
    t = top of Q
    for every u adjacent node of t connected with edge e do
        at = [at of t] + [delay of e]
        if (u not visited) or (at affects [at of u]) then
            [at of u] = at
            set u as visited
            enqueue u to Q
        end if
    end for
    pop Q
end while
```

Fig. 3: Extended BFS Algorithm

D. BnB (Branch and Bound)

Tracing all paths converging to a single test pin, requires the deployment of a depth first graph traversal that has exponential complexity. In order to reduce the search space of the graph, a branch and bound technique was utilized during the hold mode analysis. The slack in hold mode is initialized as,

$$pre\_slack^{hold}_{path} = -rat^{early}_{test\_pin} \tag{13}$$

The delays of the timing arcs are added into the slack, as the traversal moves through a single path. The condition of identifying if a path is not failing, and thus is not critical, is if its pre-CPPR slack is positive. Since the slack is monotonically increasing, if it reaches a value equal to or greater than zero, it will always be equal to or greater than zero, which means that further exploring of the graph will not alter the criticality of the test, since this path and every sub-path starting from this point will not be critical. In this case pruning of the whole sub-graph can be performed, starting from the node that contributed to the positive slack, thus reducing the runtime

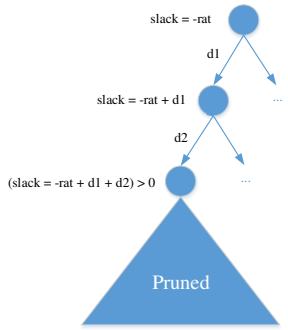

Fig. 4: Branch and Bound

E. Arbitrary CNG Handling: Dominator Graphs

In order to enhance TKtimer's ability to handle arbitrary CNGs, the SEMI-NCA algorithm [5] was utilised and the proposed implementation was incorporated in the tool. The former algorithm can transform an arbitrary graph into a tree, which in turn can be processed by the engine of the tool.

IV. Evaluation

TKtimer was extensively tested using synthetic benchmarks and under tests of varying complexity and different configurations in terms of test type (hold or setup) and number of tests/paths . The results of the TKtimer evaluation were drawed from [3]. Table I illustrates the evaluation results where #Tests is the number of performed tests, #Paths is the number of reported paths, T the execution time in minutes and A the percentage of accuracy.

TKtimer has perfect accuracy within a reasonable execution time budget. Thus, utilization of multi-core architectures makes analysis of complex designs with thousands or millions of gates feasible.

Benchmark	Test Type	#Tests	#Paths	T[min]	A[%]
Combo2v2	hold	10000	15	0.23	100
Combo2v2	setup	10000	15	0.29	100
Combo2v2	setup	20000	1	0.15	100
Combo3v2	setup	6000	20	0.16	100
Combo4v2	hold	15000	15	7.73	100
Combo4v2	hold	25000	1	7.47	100
Combo4v2	setup	15000	15	10.49	100
Combo4v2	setup	25000	1	9.73	100
Combo5v2	hold	20000	15	22.47	100
Combo5v2	hold	35000	1	21.87	100
Combo5v2	setup	20000	15	24.92	100
Combo5v2	setup	35000	1	23.33	100
Combo6v2	hold	35000	15	24.44	100
Combo6v2	hold	50000	1	26.69	100
Combo6v2	setup	35000	15	27.66	100
Combo6v2	setup	50000	1	24.6	100
Combo7v2	hold	35000	20	58.69	100
Combo7v2	hold	50000	1	54.93	100
Combo7v2	setup	35000	20	62.72	100
Combo7v2	setup	50000	1	59.18	100

TABLE I: Evaluation Results

V. Conclusion

We have presented a static timing analysis tool capable of removing inherent pessimism imposed during timing analysis in the clock network due to the presence of parametric variations on common paths. The software architecture is presented and analyzed along with algorithmic optimizations which reduce the problem space and also make the utilization of multi-core architectures possible. Moreover, the evaluation results guarantee high accuracy within an accurate sensible and acceptable timeframe.

References

[1] TAU Contest 2014. Contest education description, 2014.

[2] TAU Contest 2014. Contest file formats description, 2014.

[3] TAU Contest 2014. Contest results, 2014.

[4] S. Bhardwaj, K. Rahmat, and K. Kucukcakar. Clock-reconvergence pessimism removal in hierarchical static timing analysis, April 30 2013. US Patent 8,434,040.

[5] Loukas Georgiadis. *Linear-Time Algorithms for Dominators and Related Problems*. PhD thesis, Department of Computer Science, University of Princeton, Nov 2005.

[6] P. Ghanta, A. Goel, F.P. Taraporevala, M. Ovchinnikov, J. Liu, and K. Kucukcakar. Simultaneous multi-corner static timing analysis using samples-based static timing infrastructure, December 24 2013. US Patent 8,615,727.

[7] R. Kapur, J. Zejda, and T.W. Williams. Fundamentals of timing information for test: How simple can we get? In *Test Conference, 2007. ITC 2007. IEEE International*, pages 1–7, Oct 2007.

[8] W.A. Manaker. Method for computing hold and setup slack without pessimism, December 30 2008. US Patent 7,472,365.

[9] W.A. Manaker. Efficient method for computing clock skew without pessimism, January 6 2009. US Patent 7,475,297.

[10] J. Zejda and P. Frain. General framework for removal of clock network pessimism. In *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on*, pages 632–639, Nov 2002.