A Fast and Accurate Approach for Common Path Pessimism Removal in Static Timing Analysis

Baihong Jin[†], Guojie Luo,[†‡§] and Wentai Zhang[†]
[†]Center for Energy-efficient Computing and Applications, School of EECS, Peking University
[‡]PKU-UCLA Joint Research Institute in Science and Engineering
[§]Collaborative Innovation Center of High Performance Computing, National Univ. of Defense Tech.
Email: happyucb@gmail.com, gluo@pku.edu.cn, rchardx@gmail.com

***Abstract*—The dual-mode delay model, while being effective for characterizing on-chip timing variations, also yields timing analysis results that are overly pessimistic due to the Common Path Pessimism (CPP). In this paper, we develop a fast and accurate block-based algorithm for removing this pessimism in timing analysis, when the dual-mode delay model is used. We illustrate the effectiveness of our algorithm on a set of benchmarks from the TAU 2014 Contest [1].**

I. Introduction

Timing closure has always been the main concern of digital designers. A design team may spend months on the design iterations to achieve the timing target. The purpose of Static Timing Analysis (STA) techniques is to estimate the maximum frequency that a digital circuit can achieve, and an accurate estimation will reduce the number of design iterations and improve the design quality.

As device scaling continues, the process variations have become more significant and also harder to model. Dual-mode analysis, or "early-late split", provides a simple yet effective way to model the the On-Chip Variation (OCV) in timing analysis, where each timing edge in the timing graph has an early-mode delay and an late-mode delay that bounds the possible delay values. However, as will be introduced shortly, this delay modeling approach, when directly incorporated into block-based timing analysis schemes, will bring about undesirable Common Path Pessimism (CPP).

Over the past decades, there have been a number of efforts at Common Path Pessimism Removal (CPPR) techniques [2], [3], [4], which can be traced back to the early work of Hathaway et al [2]. The recent TAU 2014 Contest [1] has inspired a number of new approaches for CPPR. Among the contest winners, Yang et al. proposed using a dynamic branch-and-bound technique to retrieve the true critical paths [5], based on the pre-CPPR timing analysis results. Huang et al. [6] adopted a sophisticated path-based method and also achieved a good acceleration performance on parallel platforms.

Different from the aforementioned approaches, our approach to be introduced in this paper can directly derive accurate timing analysis results without CPP from a block-based timing analysis flow. To the best of our knowledge, this approach has not been seen in the previous literature. To demonstrate the effectiveness of the proposed approach, we prototyped the algorithm and compared it with the winners in the TAU 2014 Contest. It is worthy to note that in this paper we restrict our scope to, as in the TAU 2014 Contest, edge-triggered sequential circuits with a tree-structured clock network. For discussions on more general clock networks, readers are referred to [2], [3], [4].

The remainder of this paper is structured as follows: Sec. II gives the necessary background on STA. In Sec. III, we formulate the CPPR problem under bounded delay models. Detailed descriptions and analysis on our algorithm are given in Sec. IV and the experimental setup and results are presented in Sec. V. We conclude the paper in Sec. VI.

II. Preliminaries

A. Timing Graph

A *timing graph* abstracts the timing properties of a circuit. Formally, a timing graph $\mathcal{G} = (V, E)$ is a Directed Acyclic Graph (DAG) where V is the set of vertices (nodes) and E the set of edges. The *pins* of circuit elements, as well as the Primary Inputs (PIs) and the Primary Outputs (POs) of the circuit, are represented as vertices. Gate and wire delay information is described by the weights on the corresponding edges. In dual-mode analysis, each edge carries a weight $w_i = \{\underline{\delta}_i, \overline{\delta}_i\}$ that bounds all possible delay values δ_i on edge e_i, i.e.

$$\underline{\delta}_i \leq \delta_i \leq \overline{\delta}_i. \tag{1}$$

Besides, it is often convenient to connect all PIs to a virtual *source node* v_{src}, which results in a timing graph with a single source. The weights on virtual edges connecting v_{src} and PIs can be used to model the *input arrival times*.

A *path* p in $\mathcal{G}$ can be defined as an alternating sequence of vertices and edges,

$$p = \{v_0, e_0, v_1, e_1, \ldots, e_n, v_{n+1}\}. \tag{2}$$

In this paper, a path unless otherwise stated is assumed to start at v_{src}. The weight of a path $W(p) = \{\underline{D}(p), \overline{D}(p)\}$ in dual-mode analysis, where

$$\underline{D}(p) = \sum_{i=0}^{n} \underline{\delta}_i, \ \overline{D}(p) = \sum_{i=0}^{n} \overline{\delta}_i \tag{3}$$

are respectively the lower and the upper bounds of the path delay $D(p)$. A path is said to be *complete* if it connects v_{src} and a PO. In other words, a complete path cannot be extended in the timing graph. An *incomplete* path is a prefix of one or several complete paths.

B. Sequential Circuits and Common Path Pessimism (CPP)

A sequential circuit is made up of computational nodes and memory elements [7], which in general can be divided into two parts, the *clock network* and the *datapath*. The clock network distributes the clock signal to the clock input (CK port) of each flip-flop (FF). In this study, we assume the clock network to be a tree. The datapath is comprised of the rest of the circuit.

As shown in Fig. 1, between two successive clock cycles a data signal is transmitted from the launch FF to the capture FF through the datapath. For the signal to be correctly latched into FF_2, *setup time* and *hold time* constraints need to be satisfied.

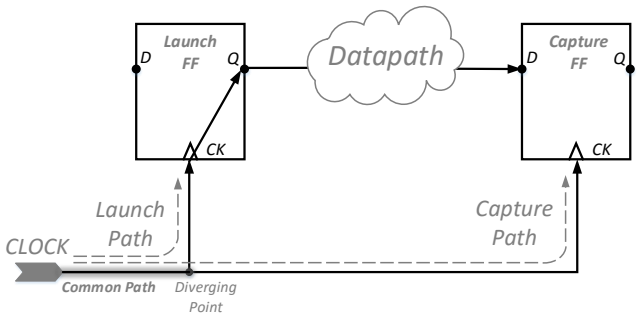

Fig. 1. Illustration of a simple sequential circuit that shows how the signal is latched into the capture FF from the launch FF.

In STA, a *test* checks whether or not a timing constraint (setup time, hold time, etc.) is satisfied. The degree of satisfaction can be quantified by the *test slack*, which is the difference between the *required arrival time* and the *earliest/latest arrival time* at a given node.

In dual-mode analysis, the CPP arises from the common portion between the launch and the capture clock paths. Taking the setup test as an example, when analyzing the *worst-case* situation we implicitly assume the launch path and the data path in late mode while the capture path in early mode. However, a signal cannot simultaneously experience two different delay values on the common clock path. Therefore, the CPP renders an underestimated slack value. The case of hold tests can be analyzed in a similar way.

C. Block-based STA

In general, STA algorithms fall into two categories, *path-based* and *block-based* approaches. Path-based approaches search for the critical path in an exhaustive way. As a result, they usually achieve better accuracy and less pessimism; however, due to the exponential number of paths in the timing graph, runtime will be an issue for path-based approaches.

Unlike path-based approaches, a block-based approach is based on a technique known as the Critical Path Method (CPM) [8] and only needs a linear topological traversal for finding the critical path in the timing graph modeling a combinational circuit. Let P be the set of paths in the graph, and P_i be the set of paths ending at v_i. If $\overline{D}(p)$ is used as the metric to evaluate the criticality of each path $p \in P_i$, a partial ordering relation can be defined on P, i.e.

$$\forall i,\ p \preceq p' \Leftrightarrow \overline{D}(p) \leq \overline{D}(p') \text{ where } p, p' \in P_i \tag{4}$$

The idea behind block-based approaches is to exclude sets of paths that cannot be critical at early stages in the search, by comparing their prefixes during the traversal. Let Π_i denote the index set of the parents of v_i, and $\delta_{j,i}$ be the delay from v_j to v_i, as illustrated in Fig. 2. We have the following recurrence relation,

$$\begin{aligned}\overline{D}(p_i^*) = \overline{d}(v_{\text{src}}, v_i) &= \max_{j \in \Pi_i} \{\overline{d}(v_{\text{src}}, v_j) + \overline{\delta}_{j,i}\} \\ &= \max_{j \in \Pi_i} \{\overline{D}(p_j^*) + \overline{\delta}_{j,i}\}. \end{aligned} \tag{5}$$

As a result, we can find the critical path to v_i by first finding the critical path to each of its parent node v_j. This provides us with a way to construct p_i^* incrementally through a topological traversal over the timing graph.

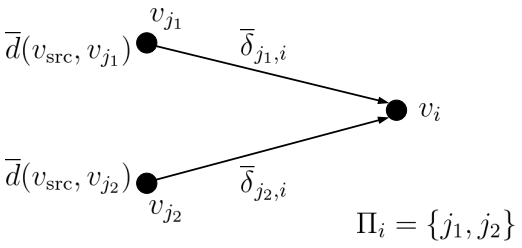

Fig. 2. In a block-based approach, the critical path to v_i can be constructed using the information of the critical paths to its parents nodes v_j where $j \in \Pi_i$.

The above approach can be easily generalized to find the top K critical paths in a graph, at the cost of increased memory consumption. Instead of storing only the information of the most critical path, we can keep track of the most critical K paths by storing multiple delay values and backtracking pointers at each node.

The block-based approach can also be extended to the analysis of sequential circuits, when the CPP is not considered. Using the case in Fig. 1 as an example, we can compute the required arrival time at the D port of FF_2 for setup tests

$$R(FF_2.D) = T + \underline{d}(v_{\text{src}}, FF_2.CK) - FF_2.t_{\text{setup}}, \tag{6}$$

where T is the minimum clock period to be tested and t_{setup} is the setup time. Here, the "." symbol is used to reference a port or property of an FF. The block-based approach can then be used to find the critical path $p^*_{\text{pre-CPPR}}$ to node $FF_2.D$, and the pre-CPPR setup test slack of FF_2

$$\begin{aligned}\text{slack}^{\text{pre-CPPR}}(FF_2.D) &= R(FF_2.D) - \overline{d}(v_{\text{src}}, FF_2.D) \\ &= R(FF_2.D) - \overline{D}(p^*_{\text{pre-CPPR}}). \end{aligned} \tag{7}$$

Because the pessimism arises from the common portion between the launch and the capture paths and is thus path-dependent, the original partial order relation (4) no longer compares the true criticality between any two paths. Moreover, the true critical path $p^*_{\text{post-CPPR}}$ after CPPR can differ from $p^*_{\text{pre-CPPR}}$. As a result, it is non-trivial to incorporate CPPR techniques into a block-based approach. In Sec. IV, we will provide a solution to that while keeping the conventional block-based STA framework.

III. Problem Formulation

The Common Path Pessimism Removal (CPPR) problem, based on the problem setup in the TAU 2014 Contest [1], can be formulated as follows: *Given the topology as well as the dual-mode delay information of a sequential circuit, and the type of test to conduct, identify N capture FFs with the most critical post-CPPR test slacks, and retrieve from each of these FFs K paths that are most critical.*

IV. Our Block-Based CPPR Approach

To obtain the true slack value of a complete path p, a *credit* should be added to its pre-CPPR slack value. In setup tests, the launch and the capture paths are sensitized in two successive clock cycles while in hold tests they are sensitized in the same cycle, which results in two different ways of calculating the credit.

$$\begin{aligned}\text{credit}(p) &= \text{slack}^{\text{post-CPPR}}(p) - \text{slack}^{\text{pre-CPPR}}(p) \\ &= \begin{cases} \overline{d}(v_{\text{clk}}, v_{\text{cp}}) - \underline{d}(v_{\text{clk}}, v_{\text{cp}}) \text{ in setup tests,} \\ \overline{d}(v_{\text{src}}, v_{\text{cp}}) - \underline{d}(v_{\text{src}}, v_{\text{cp}}) \text{ in hold tests,} \end{cases} \end{aligned} \tag{8}$$

where v_{cp} is the diverging point of the launch and the capture paths.

In a block-based approach, we rely on the below property for constructing the critical path to v_i

$$\forall i,\ \exists j \in \Pi_i \text{ s.t. } p_j^* \subset p_i^*, \tag{9}$$

where $p_j^* \subset p_i^*$ means that p_j^* is a prefix of p_i^*. When the CPP is not being considered, the criticality of a path is evaluated by the path delay D, as in (4). To incorporate the effects of CPP on the final slack value, we introduce an alternative delay metric D^{CPPR} for comparing paths,

$$D^{\text{CPPR}}(p_i) = D(p_i) + \text{credit}(p_i), \tag{10}$$

where the credit of an incomplete path p_i is decided by both its launch FF and its capture FF. However, p_i may have several possible downstream capture FFs, and a different credit should be given for each possible destination. As a result, for the block-based approach to be applicable in the CPPR setting, it is not enough to propagate only a single delay value (when $K = 1$) at a node that has multiple downstream capture FFs. In our implementation, the delay information at each node is stored in a *delay table*, which is a collection of *delay vectors* for each downstream capture FF, as shown in Fig. IV.

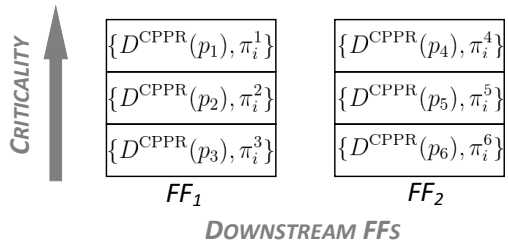

Fig. 3. An illustration of the delay table of a node v_i that has two downstream FFs. Each downstream FF corresponds to a unique delay vector that has K entries. In this case $K = 3$. Each entry corresponds to a critical path towards the associated capture FF. The modified delay value D^{CPPR} stored in each entry is used to compare the criticality between different paths. A backtracking pointer π_i is also stored in each entry for reconstructing the corresponding critical path.

The proposed block-based approach for delay propagation over the datapath is given in Algorithm 1, where the DelayPropagation procedure processes the datapath nodes in a topological order. During the traversal, the DelayTableInit procedure is invoked to initialize the delay tables at the starting points of the datapath, and the DelayTableMax procedure propagates the D^{CPPR} values over the datapath to trim away the paths that cannot be the most critical ones.

After the traversal, the post-CPPR test slacks at each capture FF can be directly computed in the same manner as in (7); however, D^{CPPR} instead of D is used here as the metric for comparing paths.

Algorithm 1 Block-based Traversal Algorithm for CPPR

Precondition: $\mathcal{D}$ is the set of topo-sorted data path nodes
Precondition: $v_i.T$ is the delay table at vertex v_i

```
1: function DelayPropagation
2:     for all i ∈ D do
3:         if Π_i ∩ D = ∅ then
4:             DelayTableInit(i)
5:         else
6:             DelayTableMax(i)
7:         end if
8:     end for
9: end function

10: function DelayTableMax(i)
11:     F ← DownstreamFF(v_i)
12:     for all f ∈ F do
13:         v_i.T[f] ← K-max_{j∈Π_i} {v_j.T[f] + δ̄(v_j, v_i)}
14:     end for                                  ▷ T: the delay table
15:     return δ
16: end function

17: function DelayTableInit(i)
18:     F ← DownstreamFF(v_i)    ▷ potential capture FFs
19:     for all f ∈ F do
20:         v_i.T[f][0] ← credit(f_0, f)    ▷ f_0: the launch FF
21:     end for
22: end function
```

Compared to the conventional block-based approach that is able to find the pre-CPPR critical path using linear time and space, our approach consumes more computational and memory resources due to the increased number of delay vectors stored at each node. In the worst case, we need to store at each node as many delay vectors as the total number of FFs in the circuit. However, this is unlikely to happen in real-world circuits. Assuming that on average each datapath node has m downstream FFs, the overhead on time and space complexity induced by our CPPR algorithm will be a factor of m compared to the pre-CPPR case.

V. Experiment Results

To validate our proposed approach, we prototyped the above algorithm in C++ and performed an experiment on the seven benchmarks released by the TAU 2014 Contest [1]. Some statistics of the benchmarks are given in Table I.

To improve the performance of our implementation, we first apply a graph reduction algorithm to obtain a smaller

TABLE II. RUNTIME COMPARISON

Benchmarks	Type	Parameters		UI-timer		iTimerC		Our Timer	
		#tests (N)	#paths (K)	Runtime (s)	Normalized	Runtime (s)	Normalized	Runtime (s)	Normalized
Combo2v2	setup	10000	15	15.30	1.31	13.69	1.17	11.70	1.00
		20000	1	8.95	1.51	6.44	1.09	5.93	1.00
	hold	10000	15	12.50	1.27	11.71	1.19	9.81	1.00
		20000	1	7.46	1.35	5.87	1.06	5.54	1.00
Combo3v2	setup	6000	20	6.89	1.02	7.73	1.14	6.78	1.00
		8000	1	2.92	1.00	3.60	1.23	3.16	1.08
	hold	6000	20	5.95	1.12	6.28	1.18	5.32	1.00
		8000	1	2.18	1.00	3.41	1.56	2.99	1.37
Combo4v2	setup	15000	15	82.88	1.43	90.03	1.55	57.99	1.00
		25000	1	43.55	1.52	43.71	1.53	28.66	1.00
	hold	15000	15	67.52	1.79	37.79	1.00	44.86	1.19
		25000	1	42.30	2.47	17.14	1.00	27.37	1.60
Combo5v2	setup	20000	15	222.87	1.85	169.23	1.40	120.56	1.00
		35000	1	150.78	2.17	89.33	1.29	69.38	1.00
	hold	20000	15	167.84	1.92	87.25	1.00	90.30	1.03
		35000	1	134.84	2.75	48.97	1.00	64.64	1.32
Combo6v2	setup	35000	15	584.83	2.40	244.05	1.00	253.71	1.04
		50000	1	433.90	3.73	116.48	1.00	165.49	1.42
	hold	35000	15	500.47	2.89	173.08	1.00	208.93	1.21
		50000	1	378.29	4.63	81.72	1.00	154.64	1.89
Combo7v2	setup	35000	20	484.15	2.04	364.20	1.54	237.26	1.00
		50000	1	299.60	2.21	136.31	1.00	135.85	1.00
	hold	35000	20	383.47	2.19	174.92	1.00	190.26	1.09
		50000	1	260.65	3.90	66.81	1.00	126.61	1.90
GEOMEAN	–	–	–	63.46	1.88	38.84	1.15	38.79	1.15

TABLE I. BENCHMARK PROFILES

Benchmark	$\|V\|$	$\|V'\|$	#FFs	#Datapath Vertices
Combo2v2	260638	56991	14957	42079
Combo3v2	181833	42586	4500	36407
Combo4v2	778640	163991	27020	127954
Combo5v2	2051806	284018	39957	226447
Combo6v2	3577928	447551	64619	357219
Combo7v2	2817563	350456	55243	276147

timing graph $\mathcal{G}'$ by eliminating the "single fan-in single fan-out" nodes. As can be seen from Table I, after the reduction the number of vertices has decreased by 86.1%.

UI-timer [6] and iTimerC [5] are two top timers in the TAU 2014 Contest. We ran the benchmarks on a Linux platform with dual Intel Xeon E5-2430 2.2GHz CPUs and 32GB RAM, and compared the performance of our timer to theirs using the results reported in [5]. The comparison in runtime is shown in Table II. All three timers produced timing analysis results with 100% accuracy. Although this is not a direct comparison on the same platform, the machine used in [5] has similar specifications to ours. It is worthy to note that our implementation is single-threaded, while UI-timer is reported to be multi-threaded [6]. It can be seen from the results that our timer is in overall as efficient as iTimerC, and outperforms the two timers in some test cases.

VI. CONCLUSION

In this paper, we proposed a block-based algorithm for removing Common Path Pessimism (CPP) during STA. We have shown that, by using an alternative delay metric, CPP can be removed under the block-based STA framework at a cost of slightly increased runtime and memory consumption. Our future work includes parallelization of the proposed algorithm on platforms such as multi-core CPUs or GPUs for further acceleration, and techniques for handling circuits with clock reconvergence.

ACKNOWLEDGMENT

This work is partly supported by National Natural Science Foundation of China (NSFC) Grant 61202073, Research Fund for the Doctoral Program of Higher Education of China (MoE/RFDP) Grant 20120001120124, and Beijing Natural Science Foundation (BJNSF) Grant 4142022.

REFERENCES

[1] J. Hu, D. Sinha, and I. Keller, "TAU 2014 contest on removing common path pessimism during timing analysis," in *Proceedings of the 2014 on International symposium on physical design*. ACM, 2014, pp. 153–160.

[2] D. Hathaway, J. Alvarez, and K. Belkhale, "Network timing analysis method which eliminates timing variations between signals traversing a common circuit path," Jun. 3 1997, uS Patent 5,636,372.

[3] 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*, Nov 2002, pp. 632–639.

[4] R. Chen, L. Zhang, V. Zolotov, C. Visweswariah, and J. Xiong, "Static timing: back to our roots," in *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*. IEEE Computer Society Press, 2008, pp. 310–315.

[5] Y.-M. Yang, Y.-W. Chang, and I. H.-R. Jiang, "iTimerC: Common path pessimism removal using effective reduction methods," in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 600–605.

[6] T.-W. Huang, P.-C. Wu, and M. D. F. Wong, "UI-timer: An ultra-fast clock network pessimism removal algorithm," in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 758–765.

[7] S. Sapatnekar, *Timing*. Springer Science & Business Media, 2004.

[8] J. E. Kelley Jr and M. R. Walker, "Critical-path planning and scheduling," in *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*. ACM, 1959, pp. 160–173.