TAU 2014 Contest on Removing Common Path Pessimism during Timing Analysis

Jin Hu, Debjit Sinha, Igor Keller†
IBM Systems and Technology Group, Hopewell Junction, USA
†Cadence Design Systems, San Jose, USA
tau.contest@gmail.com

ABSTRACT

To margin against modeling limitations in considering design and electrical complexities (e.g., crosstalk coupling, voltage drops) as well as variability (e.g., manufacturing process, environmental), "early" and "late" signal propagation delays in static timing analysis are often made pessimistic by addition of extra guard bands. While these forced "early-late splits" provide desired margins, the splits applied across the entire design introduce excessive and undesired pessimism. To this end, "common path pessimism removal (CPPR)" eliminates the redundant pessimism during timing analysis.

The aim of the TAU 2014 timing contest is to seek novel ideas for fast CPPR by: (*i*) introducing the concept and importance of common path pessimism removal while highlighting the exponential run-time complexity of an optimal solution, (*ii*) encouraging novel parallelization techniques (including multi-threading), and (*iii*) facilitating the creation of a timing analysis and CPPR framework with benchmarks to further advance research in this area.

Categories and Subject Descriptors

B.8.2 [**Hardware**]: Performance and Reliability—*Performance Analysis and Design Aids*

Keywords

Timing analysis; Common path pessimism removal; Algorithms; Design; Experimentation; Performance

1. INTRODUCTION

Static timing analysis is a key component of any integrated circuit (IC) chip design-closure flow, and is employed to obtain bounds on the fastest (*early*) and slowest (*late*) signal transition times for various timing tests and paths in the chip design. Growing chip design sizes and complexities (e.g., increased number of clock domains, increased significance of crosstalk coupling, voltage islands), as well as more complex and accurate timing models (e.g., current source models) lead to longer timing analysis run-times, thereby hindering designer productivity.

ISPD'14, March 30–April 2, 2014, Petaluma, California, USA.

http://dx.doi.org/10.1145/2560519.2565876

Variability in the deep sub-micron chip manufacturing process adds another dimension of timing analysis complexity, and ideally necessitates variation aware timing analysis. Manufacturing sources of variability include device front-end variability (e.g., variations in channel length, oxide thickness, dopant concentration) and back-end-of-line variability (e.g., metal). In addition, environmental sources of variation (e.g., voltage, temperature) impact circuit timing. While variation aware timing analysis approaches like statistical timing analysis and multi-corner timing analysis have been proposed to model variability, not all sources of variability are accurately modeled either due to increased modeling complexity or increased number of multi-corner runs.

Trade-offs are naturally performed on timing model complexity to achieve practical turn-around-times for chip static timing analysis. To margin against the ignored (or traded-off or uncertain) modeling limitations that are not explicitly and accurately modeled in the native timing models, *early* and *late* signal propagation delays (for both gates and wires) are made further pessimistic by the addition of extra guard bands. While these forced *early-late splits* provide the desired safety margins, applying the splits for the full path of some timing test introduces excessive and undesired pessimism if the data path shares an overlap with the clock path. *Common path pessimism removal (CPPR)* [1] attempts to remove this pessimism by tracing these potentially problematic paths and discarding some of the early-late difference along the common sub-path. An optimal solution to CPPR may require analysis of all paths in the design. For modern multi-million gate designs, this exponential performance complexity is impractical, and demands an intelligent selective analysis on only a sub-set of paths.[1]

This paper describes the TAU 2014 timing contest on common path pessimism removal, which seeks novel ideas for fast and accurate CPPR. The TAU 2014 timing contest (*i*) introduces the concepts and importance of CPPR while highlighting its challenges, such as exhibiting an exponential run-time complexity of an optimal solution, (*ii*) encourages novel parallelized analysis techniques, including the use of multiple threads, and (*iii*) provides a publicly available timing analysis and CPPR framework with benchmarks.

The paper is organized as follows. Section 2 provides a conceptual understanding of static timing analysis. Section 3 introduces the basic concept of CPPR. Section 4 discusses commonly-used filters in CPPR. Section 5 provides an example illustrating CPPR. Section 6 outlines the file formats of the TAU 2014 contest. Section 7 tabulates the benchmarks used in the contest. Section 8 describes the evaluation process. Section 9 concludes the paper.

[1]CPPR is alternatively known as clock reconvergence pessimism removal (CRPR).

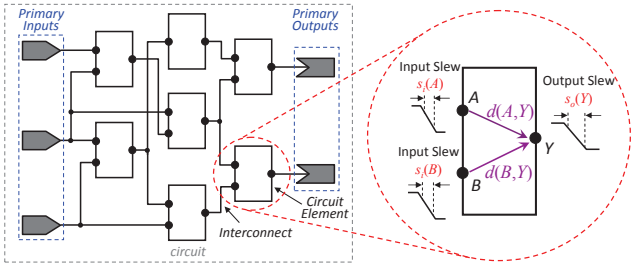

Figure 1: Generic circuit (left) and delay model representation of a combinational element (right).

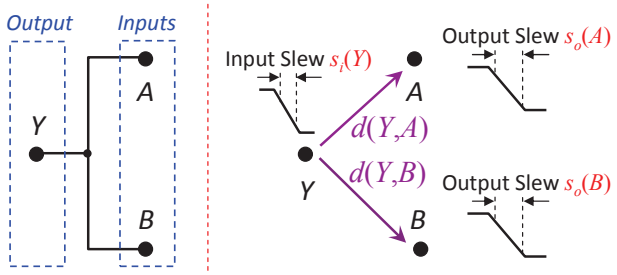

Figure 2: Generic output to input connection (left) and delay model (right).

2. STATIC TIMING ANALYSIS

A static timing analysis of a design typically provides a profile of the design's performance by measuring the timing propagation from inputs to outputs. This analysis provides a pessimistic bound, and thus facilitates further analysis of only problematic portions of the design.

2.1 Def nitions

Timing analysis of a circuit computes the amount of time required for signals to propagate from *primary inputs (PIs)* to *primary outputs (POs)* through various *circuit elements* and *interconnect*. Signals arriving at an input of an element will be available at its output(s) at some later time; each pin-to-pin connection therefore introduces a *delay* during signal propagation. For example, as shown in Figure 1 (right), the delay across the circuit element from input A to output Y is designated by $d(A,Y)$. A *timing path* is a set of directed connections through circuit elements and interconnect, and its delay is quantified by the sum of those components' delays.

A signal transition is characterized by its *input slew* and *output slew*, where slew is defined as the amount of time for the signal to transition from *high-to-low* or *low-to-high*.[2] For example, in Figure 1, the input slew at A is denoted by $s_i(A)$, and the output slew at Y is denoted by $s_o(Y)$.

To account for timing modeling limitations in considering design and electrical complexities, as well as multiple sources of variability, such as manufacturing variations, temperature fluctuation, voltage drops, and electromigration, timing analysis is typically done using an *early-late split*, where each circuit node has an early (lower) bound and a late (upper) bound on its time.[3] By convention, if the mode (early or late) is not explicitly specified, both modes should be considered. Both slew and delay are computed separately on early and late modes. For example, in early mode, an output slew s_o^E is computed using the input slew taken from the early mode, s_i^E, and, similarly, in late mode, the output slew s_o^L is computed using s_i^L. 互连与元件都会传播信号

Circuit elements. A *combinational* circuit element (e.g., OR gate) and interconnect propagate signals from input to output. Each combinational element has a defined delay d and output slew s_o for an input/output pin-pair. A *sequential* circuit element (e.g., flip-flop (FF)) captures data from the output(s) of a combinational block, and injects it into the input(s) of the combinational block in the next stage (Figure 3). This operation is synchronized by *clock signals* generated by one or multiple clock sources. Clock signals that reach distinct flip-flops (e.g., sinks in the clock tree), are delayed from the clock source by a *clock latency*.

A (D) flip-flop captures a given logic value at its input data pin D, when a given clock edge is detected at its clock pin CK, and subsequently presents the captured value[4] at the output pin Q. For correct operation, a flip-flop requires the logic value of the input data pin to be stable for a specific period of time *before* the capturing clock edge, and is denoted by *setup time* (t^S). Additionally, the logic value of the input data pin must also be stable for a specific period of time *after* the capturing clock edge. This period of time is designated by the *hold time* (t^H). These constraints are modeled by timing *setup* and *hold* tests (denoted as T_S and T_H, respectively), between the data and clock pins of the flip-flop, as shown in Figure 3 (left).

2.2 Timing Propagation

Starting from the primary input(s), the instant that a signal reaches an input or output of a circuit element is quantified as the *arrival time (at)*. Similarly, starting from the primary output(s), the limits imposed for each arrival time to ensure proper circuit operation is quantified as the *required arrival time (rat)*. Given an arrival time and a required arrival time, the *slack* at a circuit node quantifies how well timing constraints are met. That is, a positive slack means the required time is satisfied, and a negative slack means the required time is in violation.

Actual arrival time. Starting from the primary inputs, arrival times (at) are computed by adding delays across a path, and performing the minimum (in early mode) or maximum (in late mode) of such accumulated times at a convergence point. This establishes bounds on the time that a signal transition can reach any given circuit node. For example, let $at^E(A)$ and $at^E(B)$ denote the early arrival times at pins A and B, respectively, in Figure 1 (right). The most pessimistic early mode arrival time at the output pin Y is:

convergence point指的是汇点

early mode下求最小值，用min，有1个信号到了就算OK

$$at^E(Y) = \min\big(at^E(A) + d^E(A,Y),\ at^E(B) + d^E(B,Y)\big). \quad (1)$$

Conversely, in late mode, the latest time that a signal transition can reach any given circuit node is computed. Following the same example in Figure 1 (right), the most pessimistic late mode arrival time at Y is:

late mode下求最大值，用max，所有信号到了才行

$$at^L(Y) = \max\big(at^L(A) + d^L(A,Y),\ at^L(B) + d^L(B,Y)\big). \quad (2)$$

[2]A *low* (*high*) signal is defined as 10% (90%) of the voltage.
[3]This is commonly accomplished by *derating* an existing delay value (e.g., by ±5%).
对于组合电路元件，每一对input和output之间，存在一个delay和output slew

[4]The complement output $\overline{Q}$, is usually available as well.

关于early mode下的RAT和AT。每个signal到达pin之后不需要等待，立刻往下一个pin传播，因此AT的计算使用min操作。反向计算RAT的时候，从end point出发，RAT=AT，相当于这里假设，每个从到达endpoint的信号，都是按照endpoint的AT到达的，因此反向传播的时候，使用max操作，比如input pin为Y，output pin为A和B，从Y出发的信号，只需要有1个，能够使AB达到early mode下的AT，就可以满足最后endpoint达到AT，因此只需要max操作就可以了。在整个运算中，每个信号的RAT都按照endpoint那种十分苛刻的AT来计算，因此需要AT >= RAT就算满足

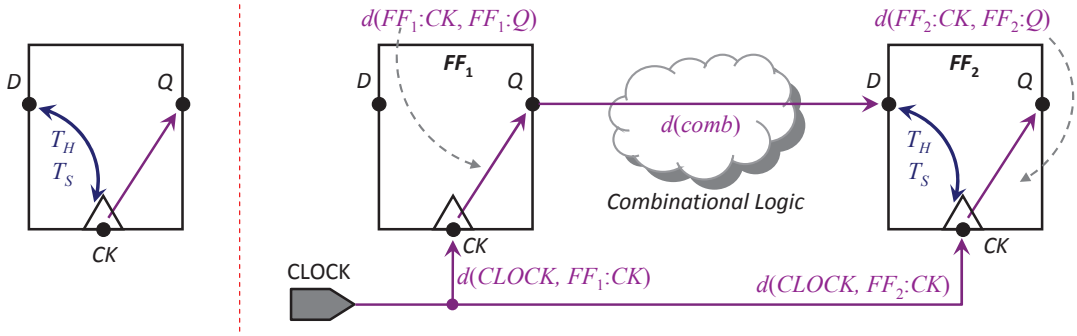

Figure 3: Generic D FF and its timing model (left), and two FFs in series and their timing models (right).

Required arrival time. Starting from the primary outputs, required arrival times (rat) are computed by subtracting the delays across a path, and performing the maximum (minimum) in early (late) mode of such accumulated times at a convergence point. That is, in early mode, the earliest time that a signal transition must reach any circuit node is computed. For example, in Figure 2, the most pessimistic early mode required arrival time at the output pin Y is:

early mode用max，late mode用min

$$rat^E(Y) = \max\big(rat^E(A) - d^E(Y,A),\; rat^E(B) - d^E(Y,B)\big). \tag{3}$$

Conversely, in late mode, the earliest time that a signal transition must reach a given circuit node is computed. Following the example in Figure 2, the most pessimistic late mode required arrival time at the input pin Y is:

late模式等价于AOE网络中的最早开始时间与最晚开始时间

$$rat^L(Y) = \min\big(rat^L(A) - d^L(Y,A),\; rat^L(B) - d^L(Y,B)\big). \tag{4}$$

Slack. For proper circuit operation, the following conditions *must* be met:

$$at^E \geq rat^E, \tag{5}$$
$$at^L \leq rat^L. \tag{6}$$

To quantify how well timing constraints are met at each circuit node, slacks ($slack$) can be computed based on the aforementioned conditions. That is, slacks are positive when the required times are met, and negative otherwise.

$$slack^E = at^E - rat^E \tag{7}$$
$$slack^L = rat^L - at^L \tag{8}$$

Slew propagation. As circuit element delays and interconnect delays are functions of input slew (s_i), subsequent output slew (s_o) must be propagated. One approach to slew propagation at a convergence point is *worst-slew* propagation, wherein the smallest (largest) slew in early (late) mode is propagated. Following the example in Figure 1 (right), the early and late output slew at output pin Y, respectively, are:

output slew与input pin对应的input slew有关

$$s_o^E(Y) = \min\left(s_o^E(A,Y), s_o^E(B,Y)\right), \tag{9}$$
$$s_o^L(Y) = \max\left(s_o^L(A,Y), s_o^L(B,Y)\right), \tag{10}$$

where $s_o^E(A,Y)$ is a function of $s_i^E(A)$, $s_o^E(B,Y)$ is a function of $s_i^E(B)$, $s_o^L(A,Y)$ is a function of $s_i^L(A)$, and $s_o^L(B,Y)$ is a function of $s_i^L(B)$.

Sequential signal propagation. Signal transition between two flip-flops is illustrated in Figure 3 (right). Assuming that the clock edge is generated at the source at time 0, it reaches the injecting (launching) flip-flop FF_1 at time $d(CLOCK, FF_1{:}CK)$, making the data available at the input of the combinational block $d(FF_1{:}CK, FF_1{:}Q)$ time later. If the propagation delay in the combinational block is $d(comb)$, then the data is available at the input of the capturing flip-flop FF_2 at time:

$$d(CLOCK, FF_1{:}CK) + d(FF_1{:}CK, FF_1{:}Q) + d(comb).$$

Assuming the clock period to be a constant P, the next clock edge reaches FF_2 at time $P + d(CLOCK, FF_2{:}CK)$. For correct operation, the data must be be available at the input pin $(FF_2{:}D)$ t^S time units before the next clock edge. Therefore, the late arrival time and the late required arrival time at the data pin are:

$$\begin{aligned} at^L(FF_2{:}D) &= d^L(CLOCK, FF_1{:}CK) \\ &\quad + d^L(FF_1{:}CK, FF_1{:}Q) + d^L(comb), \end{aligned} \tag{11}$$

这里有一个时钟周期

$$\begin{aligned} rat^{Setup}(FF_2{:}D) &= rat^L(FF_2{:}D) \\ &= P + at^E(FF_2{:}CK) - t^S \\ &= P + d^E(CLOCK, FF_2{:}CK) - t^S. \end{aligned} \tag{12}$$

A similar condition is derived for ensuring that the hold time is respected. The data input pin D of FF_2 must remain stable for at least t^H time after the clock edge reaches the corresponding CK pin. Therefore, the early arrival time and early required arrival time at the data pin are:

$$\begin{aligned} at^E(FF_2{:}D) &= d^E(CLOCK, FF_1{:}CK) \\ &\quad + d^E(FF_1{:}CK, FF_1{:}Q) + d^E(comb), \end{aligned} \tag{13}$$

同一个时钟周期

$$\begin{aligned} rat^{Hold}(FF_2{:}D) &= rat^E(FF_2{:}D) \\ &= at^L(FF_2{:}CK) + t^H \\ &= d^L(CLOCK, FF_2{:}CK) + t^H. \end{aligned} \tag{14}$$

The aforementioned arrival times from Equations (11) and (13) and required arrival times from Equations (12) and (14) induce hold and setup slacks, which are derived from Equations (7) and (8), respectively.

The timing model used for this contest is a simplified derivation of the model employed in the TAU 2013 variation-aware timing contest. Additional details are provided in [2].

3. COMMON PATH PESSIMISM REMOVAL (CPPR)

The early-late split based timing analysis effectively accounts for modeling limitations at the cost of added *pessimism*. Consequently, this can lead to an overly conservative design. Consider the slack computation example in Figure 4. The early (late) data's arrival time is compared with the late (early) clock's arrival time for the hold (setup) test. However, along the physically-common portion of the data path and clock path, the signal cannot simultaneously experience all the effects accounted for during early and late mode operation, (e.g., the signal cannot be both at high voltage and low voltage). As a result, this unnecessary pessimism can lead to tests being marked as failing (having negative slack), when in actuality, they could be passing (having positive slack). This unnecessary pessimism should thus be avoided when reporting final timing results.

Generally, the amount of pessimism for a given test can be approximated by the difference in the early and late arrival times at the *common point* (Figure 4). The common point is found by backwards tracing from the data and clock for the *path with the worst slack*. In the general case, there are multiple paths converging at the data input of a flip-flop, and every path will have its own amount of undue pessimism. Therefore, to find the correct *credit* or slack, the minimum credit found across all paths needs to be computed.

Hold tests. For tests that compare the data arrival against the clock arrival in the *same clock cycle* (e.g., data must be stable *after* the clock signal arrives at the capturing flip-flop in this case), the total pessimism incurred is the difference between the early and late arrival times at the last meeting point between the data path and clock path. That is, the signal propagation to this common point is identical for both clock and data, and thus does not warrant any early-late split. Thus, the credit for a hold test with one data path dp sharing a sub-path with clock path cp is:

$$C^{Hold}(dp) = at^L(CP) - at^E(CP), \tag{15}$$

where CP is the point when cp and dp diverge (Figure 4).

Setup tests. For tests that compare the data arrival against the clock arrival in *different clock cycles* (e.g., data must be stable *before* the subsequent cycle clock-signal arrives at the capturing flip-flop in this case), the total pessimism incurred is the sum of the partial difference of late and early delays upstream of the common point. While the data and clock path share the same physical components, they are launched at different clock cycles. Therefore, only cycle-independent contributors to the early-late split can be removed (e.g., due to global chip-to-chip variation), while cycle-dependent ones cannot be removed (e.g., due to crosstalk coupling adjusts, clock primary input arrival times, and intra-chip voltage variation). If the early-late split is limited to cycle-independent variation,[5] the credit for a setup test with one data path dp sharing a sub-path with clock path cp is:

$$C^{Setup}(dp) = \sum_{p \in \{cp \cap dp\}} d^L(p) - d^E(p), \tag{16}$$

Test slack. As removing pessimism could require analyzing many paths, the post-CPPR test slack is the minimum slack of all paths that converge at the data point of the test.[6]

[5]Considering crosstalk noise is out of scope for this contest.
[6]Only hold and setup tests are considered in the contest.

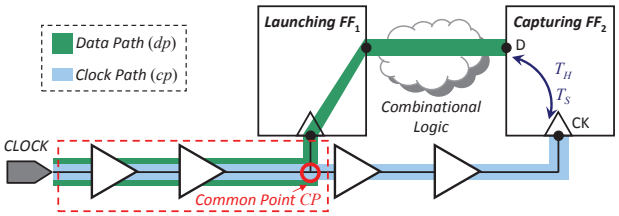

Figure 4: Pessimism due to early-late timing split.

4. CPPR RUN-TIME IMPROVEMENTS

A complete common path pessimism removal analysis necessitates investigating all paths for every (failing) test in this design. This naive procedure, however, is not used in practice by commercial timing analysis tools due to the impractical run-time for modern chip designs. As alluded to in Section 2, performing an upfront static timing analysis to find a guaranteed lower bound for all test slacks is instrumental to quickly narrow the analysis space to specific paths that causing timing failures.

A set of filters or *fast-outs* is then employed by CPPR algorithms to further prune the search space. Among common filters used in CPPR, (*i*) *path-limit*, (*ii*) *pre-CPPR slack-limit*, and (*iii*) *post-CPPR path-slack-limit*, contribute to the largest run-time savings without significantly impacting the final solution accuracy. This condition applies to the scope of this paper wherein CPPR can only improve the test slack by application of non-negative common-path credits.

Path-limit filters. Path-limit filters enforce tracing at most a given threshold number of paths for each timing test analyzed by CPPR. Ideally, if there were a known correspondence between pre- and post-CPPR paths and slacks, this filter would be able to guarantee tracing and analysis of the most critical path(s) for each test. However, such a correspondence does not always exist, and only heuristically over a representative set of designs, can a threshold for the path-limit filter be determined (which is "likely" to trace and analyze critical paths for tests). This filter is primarily used for reducing CPPR run-time while taking some risk of an incorrect test slack being reported. It should be noted that any reported test slack in this case is guaranteed to be an upper bound of the ideal test slack (Figure 5). For any given test with slack $slack_{pre}$ prior to CPPR,

$$slack_{pre} \leq slack^* \leq slack^j_{inCPPR}, \tag{17}$$

where $slack^j_{inCPPR}$ denotes the worst slack among the post-CPPR path slacks for j paths traced by CPPR, and $slack^*$ denotes the golden (correct) answer. If CPPR traces all N paths for the test, the following is guaranteed:

$$slack^N_{inCPPR} = slack^*. \tag{18}$$

It can be observed from the figure that it is not essential to trace all N paths to obtain $slack^*$. However, it is not trivial to determine if the worst post-CPPR path has already been traced, and this is what necessitates tracing all paths.

Slack-limit filters. The goal of the pre-CPPR slack-limit and *post-CPPR path-slack-limit* filters is to predict if timing tests meet specified requirements (non-negative slacks). That is, if the slack of a test is guaranteed to be above a threshold (e.g., 0), the test is considered passing and the actual value of the slack is no longer relevant. This forms

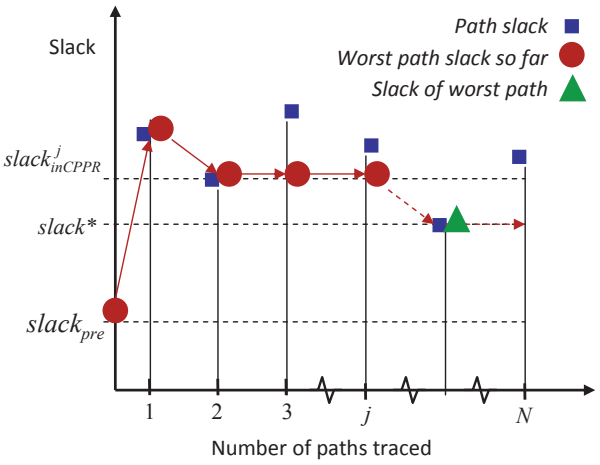

Figure 5: Illustration of worst path slack as function of paths traced during CPPR.

the basis of the *pre-CPPR slack-limit*, which denotes the threshold above which a test is considered passing. CPPR is performed on a test only if the test slack is smaller than this threshold, since it is known that CPPR can only increase the final slack. This filter can also be used during CPPR, wherein if paths are being traced in a non-decreasing order of pre-CPPR path slacks, tracing is performed only as long as for each path traced, the pre-CPPR path slack is smaller than the *pre-CPPR slack-limit* threshold.

In contrast, a post-CPPR path slack that is smaller than a threshold *post-CPPR path-slack-limit* may indicate a design problem that forces a designer to perform design updates or optimization. In such cases, the precise worst slack value is likewise irrelevant.[7] For such a test, it is commonly no longer beneficial to trace additional paths since that test's slack is guaranteed to be at most the last encountered post-CPPR path slack (which is indicating a failing test).

Commercial timing analysis tools employ a large number of similar filters to improve CPPR run-time. In this contest, a *pre-CPPR slack-limit* filter threshold of 0 is used for generation of golden results as well as for output comparisons. While useful in controlling run-time, filters should be used in CPPR with discretion to avoid optimistic results wherein the true critical path(s) is (are) not traced.

Other important factors (e.g., crosstalk, waveform effects) that impact timing and CPPR are considered by commercial timing analysis tools, but are ignored for simplicity in the contest. In addition, the topology of the clock structure is limited to a tree; clock path reconvergence and clock-grid structures are not considered for the contest.

Path tracing under variability is a considerably more difficult problem. In statistical timing, the concept of a critical path becomes less well-defined; there is now a probability p that a path is critical (e.g., $0 \leq p \leq 1$). Path ordering in this situation is a difficult problem, and destroys the properties that facilitate the use of the aforementioned slack filters in deterministic timing analysis. Statistical common path pessimism reduction is out of scope of this contest and could be a natural extension of the contest in the future.

[7]It should be noted that the pre-CPPR slack is still a slack lower bound.

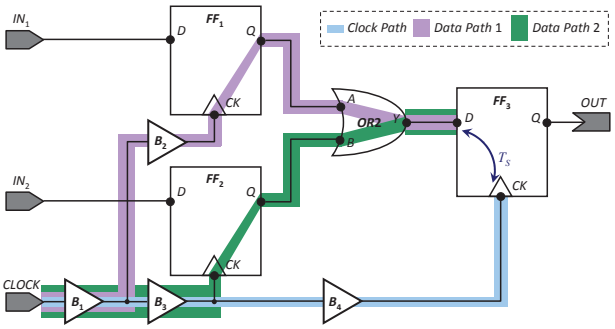

Figure 6: Example sequential circuit.

5. EXAMPLE

Consider a sample circuit as shown in Figure 6, where two data paths feed a common flip-flop (FF_3). Using the following timing information, the final CPPR credit calculation for the setup test T_S at FF_3 is illustrated. Assume that all wire delays and all primary input arrival times are zero.

$$\begin{aligned} d^E(B_1) &= 20, \quad d^L(B_1) = 25 \\ d^E(B_2) &= 10, \quad d^L(B_2) = 30 \\ d^E(B_3) &= 10, \quad d^L(B_3) = 45 \\ d^E(B_4) &= 10, \quad d^L(B_4) = 30 \\ d^L(FF_1{:}CK, FF_1{:}Q) &= d^L(FF_2{:}CK, FF_2{:}Q) = 40 \\ d^L(OR2{:}A, OR2{:}Y) &= d^L(OR2{:}B, OR2{:}Y) = 50 \\ P &= 120 \\ t^S(FF_3) &= 30 \end{aligned}$$

Pre-CPPR test slack computation. The pre-CPPR test slack is computed as follows:

$$slack^L_{pre}(FF_3{:}D) = rat^L(FF_3{:}D) - at^L(FF_3{:}D). \quad (19)$$

Using Equation (12) in Equation (19),

$$\begin{aligned} slack^L_{pre}(FF_3{:}D) &= P + at^E(FF_3{:}CK) - t^S(FF_3) \\ &\quad - at^L(FF_3{:}D) \quad (20) \\ &= 120 + 40 - 30 - \max(145, 160) = -30. \end{aligned}$$

Pre-CPPR path slack computation. Consider data path 1 (dp_1):

$$CLOCK \rightarrow B_1 \rightarrow B_2 \rightarrow FF_1 \rightarrow OR2 \rightarrow FF_3.$$

Using Equations (20) and (11), its pre-CPPR path slack is:

$$\begin{aligned} slack^L_{pre}(FF_3{:}D)|_{dp_1} &= P + \Big(d^E(B_1) + d^E(B_3) + d^E(B_4)\Big) \\ &\quad - t^S(FF_3) \\ &\quad - \big(d^L(OR2{:}A, OR2{:}Y) \\ &\quad + d^L(FF1{:}CK, FF1{:}Q) + \\ &\quad + d^L(B_2) + d^L(B_1)\big) \\ &= 120 + (40) - 30 - (145) = -15. \end{aligned}$$

Similarly, consider data path 2 (dp_2):

$$CLOCK \rightarrow B_1 \rightarrow B_3 \rightarrow FF_2 \rightarrow OR2 \rightarrow FF_3.$$

The pre-CPPR path slack for this path is:

$$\begin{aligned} slack^L_{pre}(FF_3{:}D)|_{dp_2} &= P + \Big(d^E(B_1) + d^E(B_3) + d^E(B_4)\Big) \\ &\quad - t^S(FF_3) \\ &\quad - \big(d^L(OR2{:}B, OR2{:}Y) \\ &\quad + d^L(FF_2{:}CK, FF_2{:}Q) \\ &\quad + d^L(B_3) + d^L(B_1)\big) \\ &= 120 + (40) - 30 - (160) = -30. \end{aligned}$$

Post-CPPR path slack computation. Without loss of generality, paths are traced in order of criticality, i.e., most negative to least negative. Starting with dp_2, its common portion with the clock path contains B_1 and B_3. Therefore, its credit is:

$$\begin{aligned} C^{Setup}(dp_2) &= \Big(d^L(B_1) - d^E(B_1)\Big) + \Big(d^L(B_3) - d^E(B_3)\Big) \\ &= (25 - 20) + (45 - 10) = 40. \end{aligned}$$

Similarly for dp_1, its CPPR credit is computed by finding the common portion, i.e., B_1:

$$C^{Setup}(dp_1) = \Big(d^L(B_1) - d^E(B_1)\Big) = (25 - 20) = 5.$$

The post-CPPR path slacks for dp_1 and dp_2 are:

$$\begin{aligned} slack^L_{post}(FF_3{:}D)|_{dp_1} &= slack^L_{pre}(FF_3{:}D)|_{dp_1} \\ &\quad + C^{Setup}(dp_1) \\ &= -15 + 5 = -10, \\ slack^L_{post}(FF_3{:}D)|_{dp_2} &= slack^L_{pre}(FF_3{:}D)|_{dp_2} \\ &\quad + C^{Setup}(dp_2) \\ &= -30 + 40 = 10. \end{aligned}$$

Post-CPPR test slack computation. The post-CPPR test slack for the setup test T_S between $FF_3{:}D$ and $FF_3{:}CK$ is the minimum post-CPPR path slack across all (data) paths through $FF_3{:}D$:

$$\begin{aligned} slack^S_{post}(FF_3{:}D) &= \min\big(slack^L_{post}(FF_3{:}D)|_{dp_1}, \\ &\qquad slack^L_{post}(FF_3{:}D)|_{dp_2}\big) \\ &= \min\big(-10, 10\big) = -10. \end{aligned}$$

Final CPPR credit computation. The final CPPR credit for the test is the difference between the post- and pre-CPPR test slacks:

$$\begin{aligned} C^{Setup} &= slack^S_{post}(FF_3{:}D) - slack^S_{pre}(FF_3{:}D) \\ &= -10 - (-30) = 20. \end{aligned}$$

Three observations are made from this example. (*i*) The most critical path *pre*-CPPR is not necessarily reflective of the true critical path. This emphasizes the importance of CPPR analysis, and highlights its impact on chip performance. (*ii*) During CPPR, analyzing the single-most critical path can be insufficient. In the example, if CPPR was only performed on the most critical pre-CPPR path dp_2, the final test slack would incorrectly seem positive, i.e., passing. In actuality, the next-most critical path is still failing. (*iii*) The final CPPR credit for the test need not match the credit for any one path traced during CPPR.

6. FILE FORMATS

Given a set of input files, the TAU 2014 contest requests development of a timer that reports post-CPPR critical tests and paths. The number of tests and paths to be reported is controlled via a set of pre-defined inputs to the tool.

6.1 User-def ned Parameters

The command line for the timer allows the following user-defined parameters: (*i*) the types of relevant tests (`<test-Type>`), (*ii*) the number of tests (`-numTests`), and (*iii*) the number of paths for each test (`-numPaths`).

- `-hold`, `-setup`, `-both`: input command switches that specify which set of tests to report in the output file. The option `-hold` indicates only hold tests, the option `-setup` indicates only setup tests, and the option `-both` indicates both hold and setup tests.
- `-numTests (int)`: input command where `(int)` is the number of *most critical* tests to be reported for each type of test. A test is considered critical if it has a non-positive *pre-CPPR* slack. The ordering of criticality, however, should be based on its *post-CPPR* slack. Non-critical tests should not be printed.
- `-numPaths (int)`: input command where `(int)` is the number of *most critical* paths per test to be reported for each type of test. The ordering of criticality is based on the *post-CPPR* slack, i.e., consistent with the test slack. Non-critical paths should not be printed.

6.2 Input Files

Each design consists of (*i*) a delay and (*ii*) a timing input file. The former describes the timing behavior of the circuit; the latter asserts the initial operating conditions.

Delay file. The delay file contains the description of the circuit cells as well as the delays and constraints for direct connection. Circuit topology is implicitly derived from this file, which has the following format.

---

input `<PI>`
output `<PO>`
`<source>` `<sink>` d^E d^L
hold `<data>` `<clock>` t^H
setup `<data>` `<clock>` t^S

Keywords:

- `input (output)`: primary input (output).
- `setup (hold)`: setup (hold) test.

Variable fields:

- `<PI>` `<PO>` `<source>` `<sink>` `<data>` `<clock>` specify a node or pin name. `<PI>` (`<PO>`) is a primary input (output) of the design. `<source>` and `<sink>` specify a directed connection between a source pin and sink pin of a delay segment; `<data>` and `<clock>` specify the data and clock pins of a flip-flop.
- d^E (d^L) specify the early (late) delays between `<source>` and `<sink>`.
- t^H (t^S) specify the hold (setup) test time between `<data>` and `<clock>`.

Timing file. The timing file contains the relevant timing information needed to start signal propagation. The format of this file is as follows.

clock <PI> P
at <PI> at^E(<PI>) at^L(<PI>)

Keywords:

- `clock`: clock pin.
- `at`: arrival time.

Variable fields:

- P is the clock period.
- <PI> is a primary input node.
- at^E(<PI>) (at^L(<PI>)) is the early (late) arrival time of <PI>.

6.3 Output File

The output file contains the paths for each type of test specified in accordance with the input commands. The format for each critical test T with critical paths $DP = \{dp_1, dp_2, \ldots\}$, where $slack_{post}(dp_i) < slack_{post}(dp_j),\ i < j$, is as follows.

<type> $slack_{pre}^{\text{<type>}}(T)$ $slack_{post}^{\text{<type>}}(T)$ $|DP|$
$slack_{pre}(dp_1)$ $slack_{post}(dp_1)$ $|dp_1|$
dp_1
$slack_{pre}(dp_2)$ $slack_{post}(dp_2)$ $|dp_2|$
dp_2
...

Variable fields:

- <type> is the test type, either **hold** or **setup**.
- $slack_{pre}^{\text{<type>}}(T)$ ($slack_{post}^{\text{<type>}}(T)$) is the pre- (post-)CPPR test slack for <type> test T.
- $|DP|$ is the number of reported data paths in DP.
- $slack_{pre}(p_i)$ ($slack_{post}(p_i)$) is the pre- (post-)CPPR path slack for path dp_i, $1 \leq i \leq |DP|$.
- $|dp_i|$ is the length of dp_i.
- dp_i is the node by node trace of the critical path, originating from the data pin of the FF and terminating at a primary input.

7. CONTEST BENCHMARKS

Table 1 presents the list of all benchmarks used in the TAU 2014 timing contest. The table shows number of primary inputs (PIs), primary outputs (POs), timing segments or arcs, and the number of timing tests for each benchmark. The larger benchmarks have close to a million (M) segments and several thousand (K) timing tests. Benchmark *Combo6* has 3.8M segments and 128.2K timing tests. A subset of the larger benchmarks are used for the final contest evaluation.

Figure 7 illustrates the impact of CPPR on some setup and hold test slacks for benchmarks *des_perf* and *vga_lcd*. The horizontal and vertical axes in the plots denote the pre-CPPR and post-CPPR slacks, respectively. Using the line indicating identical slack (with slope = 1.0) as reference, it is observed that each post-CPPR slack is at least the pre-CPPR slack value. Slack improvements of up to 100 picoseconds are seen from the figure. The plots indicate the importance of CPPR during design closure from a designer's perspective.

Table 1: Contest benchmarks and statistics.

Design	Number of:			
	PIs	POs	Segments	Tests
s27	6	1	112	6
s344	11	11	658	30
s349	11	11	682	30
s386	9	7	701	12
s400	5	6	813	42
s510	21	7	1091	12
s526	5	6	1097	42
s1196	16	14	2.4K	36
s1494	10	19	2.9K	12
systemcdes	132	65	13.3K	380
wb_dma	217	215	17.4K	1374
tv80	14	32	23.7K	838
systemcaes	260	129	29.6K	2.5K
mem_ctrl	115	152	45.0K	3.7K
ac97_ctrl	84	48	55.7K	9.3K
usb_funct	128	121	66.1K	4.3K
pci_bridge32	162	207	78.2K	16.4K
aes_core	260	129	86.7K	2.5K
des_perf	235	64	404.2K	19.7K
vga_lcd	89	109	525.6K	50.1K
Combo2	170	218	284.4K	29.5K
Combo3	353	215	216.2K	8.2K
Combo4	260	169	866.3K	53.5K
Combo5	432	164	2229.6K	79.0K
Combo6	486	174	3843.9K	128.2K
Combo7	459	148	3012.3K	109.6K

8. EVALUATION

A timer is evaluated on its (i) accuracy relative to the "golden" reference and (ii) relative run-time. Each output file o is uniquely identified by the benchmark design and the command-line settings of `-numTests`, `-numPaths`, and <testType>. The final score is the average across the set of all output files O.

Test and path accuracy. The slack accuracy $A(T)$ ($A(dp)$) for each reported test (path) with negative post-CPPR slack in o is compared with the golden. The accuracy score is a function of the slack difference, where if the difference falls outside of a specified threshold, no score is awarded.

Design accuracy. Given all test and path accuracy scores, the (raw) accuracy measurement $A(o)$ for an output file is the average of the following five terms.

- average of $A(T)$ of $T \in TEST$, where $TEST$ is bounded by `-numTests`
- average of $A(dp)$ of $dp \in DP$, where DP is bounded by `-numPaths`
- average of $A(critDP(T))$ of $T \in TEST$, where $critDP(T)$ is the most critical path for T
- $A(critT)$ of $critT \in TEST$, where $critT$ is the most critical test in the design
- minimum of $A(T)$ of $T \in TEST$

The first three terms quantify the overall accuracy quality, while the last two terms quantify the worst accuracy quality.

Run-time. Timer run-time is used as a modifier to $A(o)$.

Final ranking. A timer's overall score is the average across each output's weighted accuracy. For each benchmark design, multiple output files that emphasize both the importance of finding the most critical tests, and the set of paths that cause each test's timing violations, are generated.

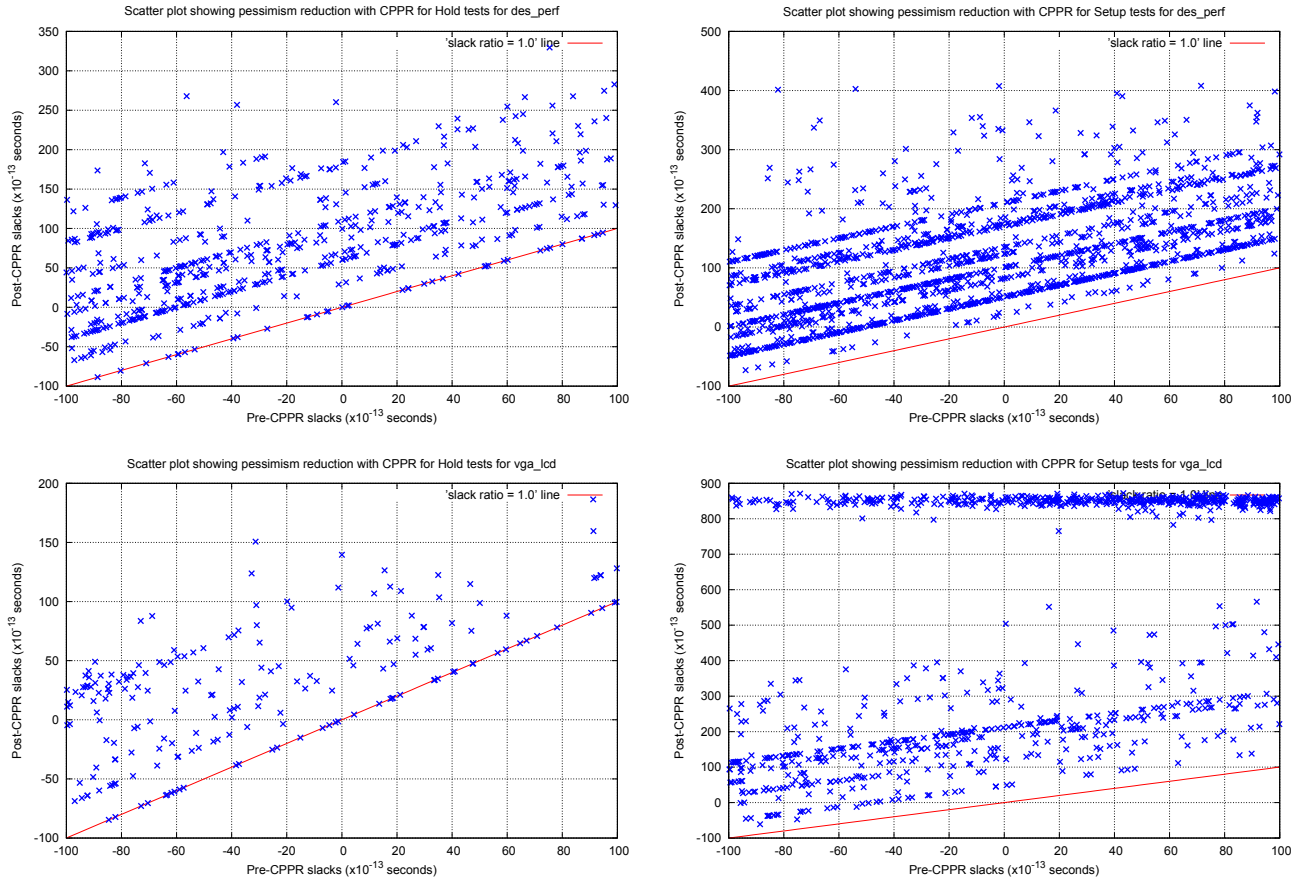

Figure 7: Impact of CPPR on test slacks.

9. SUMMARY

Common path pessimism reduction is an essential step during static timing analysis which reduces undesired pessimism from forced "early-late splits" in the common data and clock portion of timing tests' paths. An exhaustive approach may required analysis of all paths in the design, which is run-time impractical for modern large designs. This paper describes the TAU 2014 timing contest on common path pessimism reduction.

The aim of the TAU 2014 timing contest is to seek novel ideas for fast common path pessimism removal (CPPR) by: (*i*) introducing the concept and importance of common path pessimism removal while highlighting its associated challenges, (*ii*) encouraging novel parallelization techniques, and (*iii*) facilitating the creation of a publicly available timing analysis and CPPR framework, along with a set of benchmarks. Simplified delay models are used in the contest to steer focus towards algorithm development that aim to reduce run-time (including parallelization or multi-threading). Additional important factors for CPPR (e.g., crosstalk, waveform effects, and clock-grid structures) are ignored for simplicity in the contest.

Acknowledgments

The netlist and library formats used in the contest are simplified derivations of those used in the TAU 2013 variation aware timing analysis contest [2]. The authors acknowledge the help and support of the TAU 2013 contest committee for providing sample benchmarks and library.

The authors also acknowledge Jobin Jacob Kavalam, Sudharshan V, Prof. Nitin Chandrachoodan and Prof. Shankar Balachandran (IITimer team) from the Indian Institute of Technology, Madras, India for sharing the source code of their timing analysis tool (winners of [2]) with the participants of the TAU 2014 timing contest, and helping with the initial benchmark conversion from the prior year's contest.

10. REFERENCES

[1] J. Bhasker and R. Chadha, *Static timing analysis for nanometer designs: A practical approach*, Springer, 2009.

[2] D. Sinha, L. Guerra e Silva, J. Wang, S. Raghunathan, D. Netrabile and A. Shebaita, "TAU 2013 variation aware timing analysis contest", *Proc. International Symposium on Physical Design*, 2013, pp. 171-178.