TAU 2015 Contest on Incremental Timing Analysis

Invited Paper: Incremental Timing and CPPR Analysis

Jin Hu, Greg Schaeffer, Vibhor Garg[†]
IBM Systems and Technology Group, Hopewell Junction, NY, USA
[†]Cadence Design Systems, San Jose, CA, USA
{jinhu, gmschaef}@us.ibm.com, [†]gargv@cadence.com

***Abstract*—Among timing analysis applications, timing-driven operations are imperative for the success of optimization flows, such as placement, routing, logic synthesis, and physical synthesis. Optimization transforms change the design, and therefore have the potential to significantly affect timing information. As such, timing must be kept current to ensure slack integrity and timing closure. For reasonable turnaround and performance, the timer should *incrementally* update the affected portion of the design.**

The aim of the TAU 2015 timing contest is to seek novel ideas for incremental timing analysis by: (i) introducing the concept and motivating the importance of incremental timing analysis, (ii) encourage novel parallelization techniques (including multi-threading), and (iii) facilitating the creation of an incremental timing analysis framework with benchmarks to further advance this research area.

I. Introduction

Static timing analysis (STA) is essential to the manufacturing process of any integrated circuit. It is used to determine bounds on the *early* and *late* signal transitions through circuit elements across hundreds of millions of paths. As circuit design size increases and chips become more susceptible to variation effects (e.g., crosstalk coupling, temperature and process complexities), the required time to perform STA can be hindering to designer productivity. Improvements have been developed to standalone timing analysis and common path pessimism removal (CPPR) in recent timing analysis contests [4], [9]. However, if designers need instantaneous feedback after a small set of changes, re-analyzing timing for hundreds of millions of gates to obtain an updated timing profile for tens of gates is prohibitively slow.

Various stages of the design flow (e.g., logic synthesis, placement, routing, physical synthesis and optimization) facilitate a need for *incremental timing analysis*. During these stages, local operations such as gate repowering, buffering, or net rerouting can modify small portions of the design, and may significantly change both the local and global timing landscape. In these environments where the timing information may be queried after each operation (i.e., so as to ensure timing closure), the performance and turnaround time of timing analysis is crucial, and timing information must be kept up-to-date in an incremental fashion.

This paper describes the TAU 2015 timing contest on incremental timing and incremental common path pessimism removal, which seeks novel ideas for maintaining the timing profile of a design after design modifications. The TAU 2015 timing contest (i) introduces the concepts and importance of incremental timing analysis 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 incremental timing analysis and CPPR framework with benchmarks.

The paper is organized as follows. Section II provides a conceptual understanding of static timing analysis and timing propagation. Section III introduces the basic concept of CPPR. Section IV discusses the impact of incremental timing and CPPR. Section V states the delay and slew models used for timing propagation. Section VI outlines the file formats of the TAU 2015 contest. Section VII tabulates the benchmarks used in the contest. Section VIII describes the evaluation process. Section IX concludes the paper.

II. 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.

pessimistic 悲观的

A. Definitions

Timing analysis of a circuit analyzes the amount of time required for signals to propagate from *primary inputs (PIs)* to *primary outputs (POs)* through various *circuit elements* and *interconnect* (Figure 1). Signals arriving at an input of an element will be available at its output at some later time; each pin-to-pin connection therefore introduces a *delay* during signal propagation. For example, the delay across the combinational OR gate from input A to output Z is designated by $d_{A\rightarrow Z}$. (Figure 1). A *timing path* is a set of directed connections through circuit elements and interconnect, and its delay is 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*.[1] For example, the input slew at A is denoted by s_{iA}, and the output slew at Z is denoted by s_{oZ} (Figure 1).

To account for timing modeling limitations such as design and electrical complexities and multiple sources of variability, such as manufacturing variations, temperature fluctuation,

[1]A *low* (*high*) signal is defined as 10% (90%) of the voltage.

delay考虑的是pin-to-pin之间需要的时间
slew考虑的是signal transition需要的时间，low-to-high和high-to-low

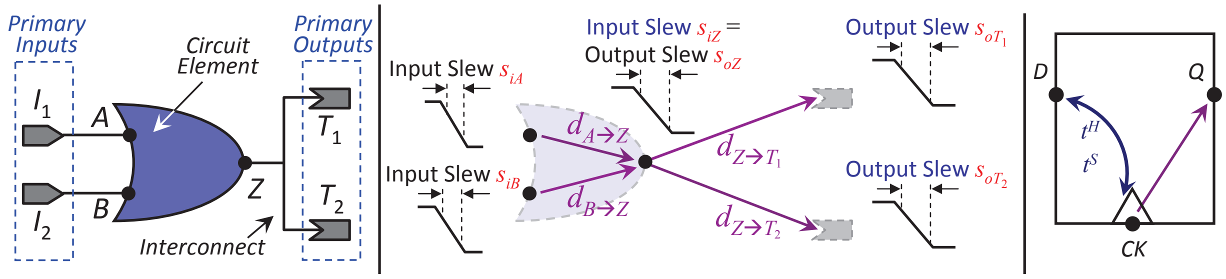


Fig. 1. Example circuit (left), its timing model (center), and generic D FF timing model (right).

考虑到这种时间特性建模的限制以及电路的多变性，通常使用early-late split这种方法

voltage drops, and electromigration, the analysis is typically performed using an *early-late split*, where each circuit node has an early (lower) bound and a late (upper) bound on its timing characteristics.[2] By convention, if the mode (early or late) is not explicitly specified, both modes should be considered. Both delay and slew 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.

Circuit elements. A *combinational* circuit element 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 2). This operation is synchronized by *clock signals* generated by one or multiple clock sources. Clock signals that reach distinct FFs (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 at the output pin Q. For correct operation, a FF 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 between the data and clock pins of the FF (Figure 1 (right)).

B. Timing Propagation

Starting from the primary inputs, the time at which a signal reaches an input or output of a circuit element is quantified as the *arrival time (at)*. Starting from the primary outputs, the limits imposed for each arrival time to ensure proper circuit operation is quantified as the *required arrival time (rat)*. Given an *at* and a *rat*, 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. The in-depth computation of the *at*, *rat*, *slack*, *slew*, *setup guard time*, and *hold guard time* values is further discussed in [4].

[2]This is commonly done by *derating* an existing delay value (e.g., by ±5%).

Transitions. For each timing arc, delay and output slew values will propagate only for transitions that exist. For example, if the first timing arc propagates rise-to-rise and fall-to-fall, and the second timing arc propagates fall-to-rise, then the only output transition at the second timing arc will be rise. Similarly, setup and hold tests are checked against the specified transitions that are present in their definitions.

III. 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 example in Figure 2. 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.

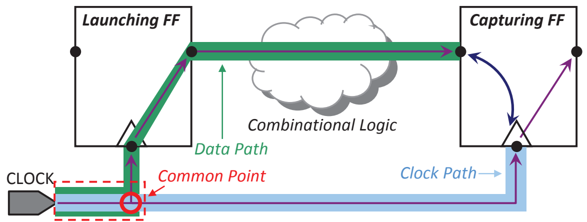


Fig. 2. Two FFs in series showing the clock and data paths for CPPR analysis.

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 2). 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. For *early-mode* or *hold* tests that compare the data arrival against the clock arrival in the *same clock cycle*, the pessimism incurred is the difference between the early and late arrival times at the common point between the data path and clock path. For *late-mode* or *setup* tests that compare the data arrival against the clock arrival in *different clock cycles*, the credit is the sum of the partial difference of late and early delays upstream of the common point. For details on how CPPR credit is computed, see [4].

IV. Challenges and Impact of Incremental Timing and CPPR

Once the critical paths in a design have been identified based on a timing analysis, an optimization tool may be used to optimize logic so as to overcome the timing violations on violating paths, or reduce area and power consumption by changing cells on positive slack paths. As these transformations are performed, the design timing for the path or endpoint under consideration changes, but it may also impact timing for another related path or unrelated endpoint. A transformation is not accepted if it causes additional slack violations. Since an optimization tool can perform thousands of logic transformations, it is imperative that the timing profile is updated accurately and efficiently. As shown in Figure 3 (left), a change in delay δ can affect up to the majority of a circuit, but depending on where the critical paths are, only a small fraction of the circuit's timing points would need to be updated. For example, if δ does not change the arrival time at `n1:o`, e.g., the critical path was through `n1:a` $\rightarrow$ `n1:o`, then every downstream timing point after `n1:o` is unaffected. However, if δ caused the critical path to change, e.g., improved the slack from the original critical path `n1:b` $\rightarrow$ `n1:o`, then the downstream arrival times would need to be updated accordingly. Figure 4 demonstrates the potential performance benefit from incremental timing on the `netcard_iccad` design based on a random number of operations.

Logic transformations on the data network do not impact CPPR adjustment for various timing checks as the clock paths

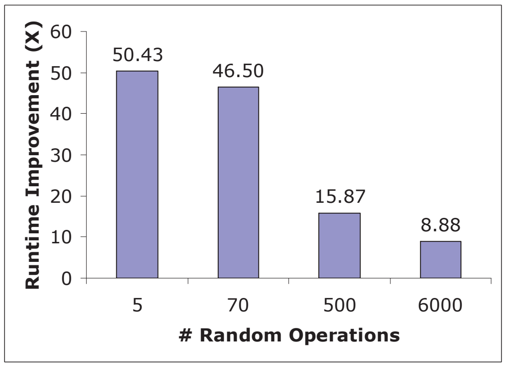


Fig. 4. Performance improvement of incremental timing to full timing on `netcard_iccad` after a set of random operations.

are not changed, and any cached value for CPPR adjustment for a given launch-capture pair can be re-used. However, if the transformations are on the clock network, example re-sizing a buffer or adding or deleting buffers or inverters on clock tree to meet slack or skew targets, they can potentially impact a large number of data paths and slacks for these end points must be re-calculated with updated CPPR adjustment values. Further, in some cases, changes to the clock network may not even impact CPPR for any end point at all. Therefore, the challenge with incremental changes to the clock network is correctly identifying what end points are affected by which changes. As shown in Figure 3 (right), a change in delay δ_1 at instance `i2` can impact CPPR adjustment for both launch-capture flop pairs `f1` and `f3`, as well as `f2` and `f3`, while a change in delay δ_2 at instance `i5` only causes an update for pair `f2` and `f3`. A third change in delay δ_3 at `i6`, however, does not impact the CPPR adjustment for any FF pair.

V. Delay and Slew Modeling

The TAU 2015 contest uses the following models to calculate the delay and slew of each net and circuit element. The parasitics of each net are stored in the **.spef** file (Section VI-C), and the gate definitions are stored in the **.lib** files (Section VI-A). Additional details are provided in [9].

A. Interconnect Modeling

The basic instance of interconnect (wire) is a *net*, which is assumed to have an input pin $Port$ and one or more output pins *Taps* (Figure 5 (left)). The net has *parasitics* (e.g., resistors and capacitors), which define its delay and output slew. The TAU 2015 contest limited parasitic RC networks to only include grounded capacitors and no floating resistors.

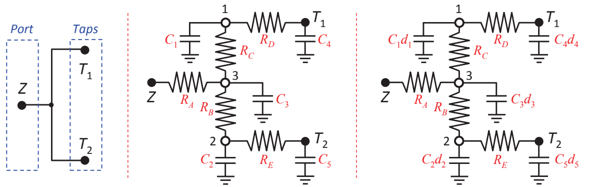
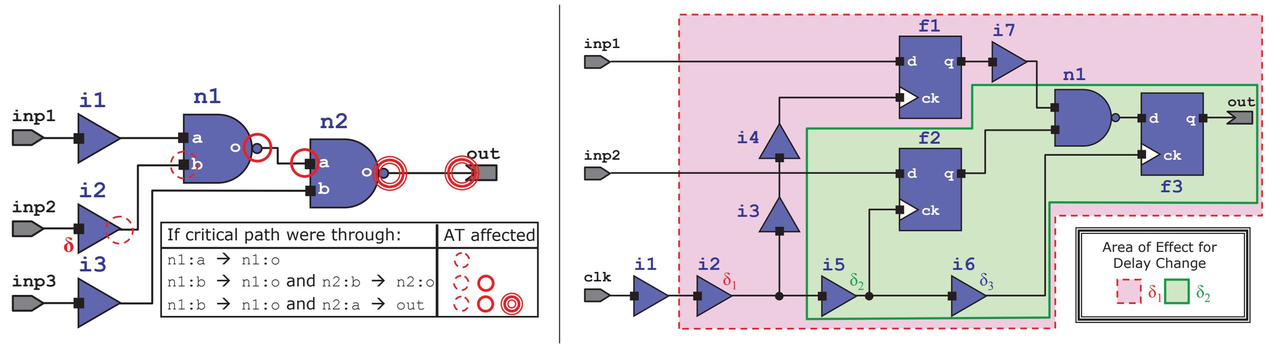


Fig. 5. Generic interconnect timing model (left), its delay RC network (center) and its slew RC network (right).

Delay. The computation of port-to-tap delays can be accurately performed through electrical simulation. However, for simplicity, the TAU 2015 contest assumes the Elmore delay model [2], where the delay is approximated by the symmetric of the value of the first moment of the impulse response. To compute the delay of RC tree networks, we summarize the topological method provided in [7].

In an RC network, consider any two nodes e and k. Let C_k be the lumped capacitance at node k, and let $R_{k\rightarrow e}$ be the total resistance of the common path between the paths from Z to e and Z to k. For example, in Figure 5 (center), the resistance between nodes 1 and T_2 ($R_{1\rightarrow T_2}$) is R_A, as that is the only

net假设有1个input（port）和1个或多个output（tap），通过这种方法计算port-to-tap的delay以及每个tap的slew



If critical path were through:	AT affected
n1:a → n1:o	◌
n1:b → n1:o and n2:b → n2:o	◌ ○
n1:b → n1:o and n2:a → out	◌ ○ ◎

Fig. 3. Illustration of incremental timing (left) and CPPR impact (right) due to delay changes.

common resistor between the paths Z to 1 and Z to T_2. The Elmore delay at node e is

$$d_e = \sum_{k \in N} R_{k \to e} C_k \tag{1}$$

where N is the set of all nodes in the RC network. For the example net illustrated in Figure 5 (right), the delay at node T_2 (tap) is (visiting in order nodes 1, T_1, 3, 2, T_2):

$$\begin{aligned} d_{T_2} &= R_A C_1 + R_A C_3 + R_A C_4 + \\ &\quad (R_A + R_B) C_2 + (R_A + R_B + R_E) C_5 \\ &= R_A (C_1 + C_3 + C_4) + \\ &\quad (R_A + R_B) C_2 + (R_A + R_B + R_E) C_5 \end{aligned} \tag{2}$$

Output slew. The value of the output slew (s_o) on any given tap node T can be approximated by a two-step process. First, compute the output slew of the impulse response on T, which was observed [2], [3] to be well-approximated by

$$\hat{s}_{oT} \approx \sqrt{2\beta_T - {d_T}^2} \tag{3}$$

where β_T is the second moment of the input response at node T, and d_T is the corresponding Elmore delay from Equation 1. Second, compute the slew of the response to the input ramp by the expression given in [5]

$$s_{oT} \approx \sqrt{{s_i}^2 + \hat{s}_{oT}^2} \tag{4}$$

where s_i is the input slew. The value of β_T can be computed through the efficient path-tracing algorithm for moment computation proposed in [8], which is a generalization of the algorithm proposed in [2]. To calculate β_T, first replace all capacitance values C_k in the RC network by $C_k d_k$, where d_k is the Elmore delay from Equation 1 (Figure 5 (right)). Second, follow the same procedure as before for finding β_T

$$\beta_T = \sum_{k \in N} R_{k \to T} C_k d_k \tag{5}$$

Following the example in Figure 5 (right), at node T_2,

$$\begin{aligned} \beta_{T_2} &= R_A (C_1 d_1 + C_3 d_3 + C_4 d_4) + \\ &\quad (R_A + R_B) C_2 d_2 + (R_A + R_B + R_E) C_5 d_5 \end{aligned} \tag{6}$$

B. Circuit Element Modeling

The delay and output slew for any timing arc are represented as non-linear delay models, and encapsulated in multi-dimensional tables in the **.lib** file. To find the corresponding timing information, extrapolation or interpolation will be necessary.

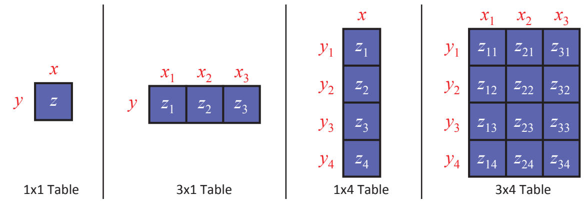


Fig. 6. Illustration of different tables: scalar, one-dimensional, and two-dimensional.

If the table contains a single value, i.e., a 1×1 table (Figure 6 (left)), no interpolation is necessary. That is, regardless of input x and y, the corresponding value is constant. If the table is one-dimensional, i.e., a $1 \times n$ table or a $m \times 1$ table (Figure 6 (center)), then the value will depend only on the non-scalar dimension. For example, consider the 1×4 table in Figure 6. If $y < y1$, then the corresponding output z value will be the linear extrapolation between z_1 and z_2. If $y_2 \leq y \leq y_3$, then z will be the linear interpolation between z_2 and z_3. If $y_4 < y$, then z will be the linear extrapolation between z_3 and z_4.

$$\begin{aligned} & z_1 - (y_1 - y)\frac{z_2 - z_1}{y_2 - y_1} && \text{if} \quad y < y_1 && (7) \\ & z_1 && \text{if} \quad y = y_1 && (8) \\ & z_1 + (y - y_1)\frac{z_2 - z_1}{y_2 - y_1} && \text{if} \quad y_1 < y < y_2 && (9) \\ & z_2 && \text{if} \quad y = y_2 && (10) \\ & z_2 + (y - y_2)\frac{z_3 - z_2}{y_3 - y_2} && \text{if} \quad y_2 < y < y_3 && (11) \\ & z_3 && \text{if} \quad y = y_3 && (12) \end{aligned}$$

$$z_3 + (y - y_3)\frac{z_4 - z_3}{y_4 - y_3} \quad \text{if} \quad y_3 < y < y_4 \tag{13}$$

$$z_4 \quad \text{if} \quad y = y_4 \tag{14}$$

$$z_4 + (y - y_4)\frac{z_4 - z_3}{y_4 - y_3} \quad \text{if} \quad y > y_4 \tag{15}$$

If the table is two-dimensional, perform linear interpolation on the x values first, then perform linear interpolation on the y values. For example, consider the 3x4 table in Figure 6. If $x_2 < x < x_3$ and $y_2 < y < y_3$, then (i) determine z_{first} by linear interpolation on z_{22} and z_{32}, (ii) determine z_{second} by linear interpolation on z_{23} and z_{33}, and then (iii) determine z by linear interpolation using z_{first} and z_{second}.

For a delay or an output slew query, the tables are referenced by the cell's *input slew* s_i and *output* or *driving load* C_L, where C_L is the equivalent downstream capacitance as seen from the output pin of the cell. For simplicity, C_L is assumed to be the sum of all the capacitances in the parasitic RC tree, including the cell pin capacitances at the taps of the interconnect network. For a setup or hold guard time query on a sequential cell element, the tables are referenced by the cell's *clock-side input slew* and *data-side input slew*.

VI. File Formats

Given a set of input files, the TAU 2015 contest requests the development of an incremental timer that both (i) maintains the current state of the design after a set of design modifiers and (ii) reports the post-CPPR timing of any queried data path or timing point. The initial design is defined by its cell library (**.lib**), netlist and topology (**.v**), parasitics (**.spef**), and boundary timing and load conditions (**.timing**). The sets of expected operations and timing queries are encapsulated in an input command file (**.ops**). All four files are listed in a wrapper file for ease of processing (**.tau2015**).

A. .lib (Liberty)

The set of available gates is encapsulated using the Liberty (**.lib**) industry standard. Cell information includes the necessary delay and output slew information for each combinational and sequential element, as well as the setup and hold guard times for sequential elements. The following outlines the relevant portion of the **.lib** syntax, and is further explained in [10]. The Liberty format follows the *non-linear delay model (NLDM)*, where each calculation is based on the interpolation and extrapolation of multi-dimensional *lookup-tables (LUTs)*. An example of an LUT definition is:

```
index_1 ("1.05, 2, 5, 5.5, 9");
index_2 ("1.00");
  values ( "2.1", "4", "8", "11", "18");
```

where `index_1` references the rows of the table, and `index_2` references the columns of the table.

Combinational timing arcs. Combinational arcs propagate delay and output slew from a source pin to a sink pin. They are found in common combinational logic gates, e.g., `NAND2` or as a clock-trigger segment in flip-flops. A propagate segment's syntax is:

```
cell(<cell name>) {
 pin (<sink pin name>) {
  direction : <direction> ; input, output, internal
  timing() {
   related_pin   : <source pin name> ; source pin
   timing_sense : <timing sense> ;
   timing_type   : <timing type> ;
   cell_<transition> (<table label>) {
    <LUT instance> cell delay
   }
   <transition>_transition(<table label>) {
    <LUT instance> output slew
   }
  }
 }
}
```

where the `related_pin` refers to the source, and the `pin` refers to the sink of the timing arc, and `direction` is either `input`, `output`, or `internal`. The `timing_sense` specifies the transition mode: (i) `positive_unate`, where the source and sink transitions are the same (e.g., rise-to-rise), (ii) `negative_unate`, where the source and sink transitions are opposite (e.g., rise-to-fall), and (iii) `non_unate`, where the source transition has no relation to the sink transition. The `timing_type` specifies if (i) the arc is `combinational`, where the unateness is be defined as either `positive_unate` or `negative_unate`, or (ii) `<edge_type>_edge`, where the unateness is defined as `non_unate` and `<edge_type>` is either `rising` or `falling`, and refers to the source.

The `cell_<transition>` table references the arc's delay values, whereas the `<transition>_transition` table references the arc's output slew. In both tables, `<transition>` refers to the sink, and is either `rise` or `fall`. In the case of `non_unate`, both `<timing_sense>` and `<transition>` are used, where the former is the source, and the latter is the sink. The `<table label>` is (i) a LUT template or (ii) `scalar`, indicating a constant.

Sequential timing checks. Timing tests or checks ensure that timing is consistent between two pins, commonly between the clock and data pins of a sequential element. Common checks include *hold tests* that ensure the data signal is held long enough after the clock launches, and *setup tests* that ensure the data signal arrives safely before the clock signal. A test's syntax is:

```
cell(<cell name>) {
 pin (<data pin name>) {
  direction: <direction> ;
  timing() {
   related_pin: <clock pin name> ;
   timing_type: <timing type>;
   <transition>_constraint (<table label>) {
     <LUT instance>
   }
  }
 }
}
```

where the `related_pin` is the clock side, and the `pin` is the data side of the test. The `timing_type` states if the test is a hold or setup test, and checks against the clock-side transition with `hold_rising`, `hold_falling`, `setup_rising` or `setup_falling`. The `<transition>_constraint` tables refer to the guard times, where `<transition>` refers to the data side of the test.

B. Design Topology (**.v**)

The design netlist is encoded using Verilog (**.v**), an industry-standard format. The contest uses a small subset of Verilog keywords, and follows the format below.

```
module <circuit name> (<input>, <output>);
 <cell type> <instance> (.<pin> (<net>));
end module
```

The Verilog file defines the interface of design `<circuit name>` with as many `<input>` and `<output>` as necessary. The netlist is comprised of gates with `<cell type>` and `<instance>`, where the pins of each gate is connected through `<net>`. The nets of the design are optionally defined using the keywords `input`, `output` and `wire`.

C. Design Parasitics (**.spef**)

The design parasitics are encoded using the *Standard Parasitic Exchange Format* (**.spef**), an industry-standard format, with each net using the following syntax:

```
*D_NET <net name> <total net capacitance>
*CONN
<pin type> <pin name> <pin direction>
*CAP
<cap id> <node> <capacitance>
*RES
<res id> <node1> <node2> <res>
*END
```

The parasitics of each `<net name>` comprises of a series of resistors and capacitors. Each capacitor is associated with a unique identifier `<cap id>`, and has value `<capacitance>` at `<node>`. Each resistor is associated with a unique identifier `<res id>`, has value `<res>`, and connects `<node1>` and `<node2>`.

D. Design Example

The two files `simple.v` and `simple.spef` encode the `simple` design (Figure 7).

```
// simple.v
module simple (inp1, inp2, clk, out);
 NAND2_X1 u1 (.a(inp1), .b(inp2), .o(n1));
 DFF_X80 f1 (.d(n2), .ck(clk), .q(n3));
 INV_X1 u2 (.a(n3), .o(n4));
 INV_X2 u3 (.a(n4), .o(out));
 NOR2_X1 u4 (.a(n1), .b(n3), .o(n2));
endmodule
```

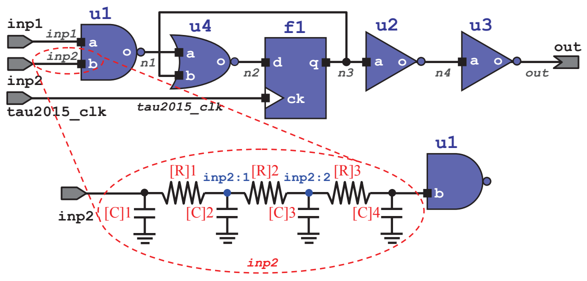


Fig. 7. `simple` design encoded with `simple.v` and `simple.spef`.

```
// simple.spef
*D_NET inp2 2.0
*CONN
*P inp2 I
*I u1:b I
*CAP
1 inp2 0.2
2 inp2:1 0.5
3 inp2:2 0.4
4 u1:b 0.9
*RES
1 inp2 inp2:1 1.4
2 inp2:1 inp2:2 1.5
3 inp2:2 u1:b 1.6
*END
```

E. Design Operating Conditions (**.timing**)

The design's boundary operating conditions are asserted through the **.timing** file, and follows the format below.

```
clock <PI> <clock period> <low %>
at <PI> <ER> <EF> <LR> <LF at>
slew <PI> <ER> <EF> <LR> <LF>
rat <PO> <ER> <EF> <LR> <LF>
load <PO> <capacitance>
```

The `clock` denotes that `<PI>` is the clock input with `<clock period>` and duty cycle `<low %>`. Each `at` and `slew` (`rat`) asserts the corresponding initial condition on the `<PI>` (`<PO>`) for each early-rise `<ER>`, early-fall `<EF>`, late-rise `<LR>`, and late-fall `<LF>` transition. The `load` asserts a fixed capacitance on the `<PO>` pin.

F. Commands and Operations (**.ops**)

The operations file (**.ops**) contains the set of commands that are supported. There are two types of operations – those that modify the design, and those that query timing information.
Circuit modification commands. This set of commands modifies the design's topology at the (*i*) gate-level, (*ii*) net-level, and (*iii*) pin-level.
Gate-level:

- `insert_gate`: adds an unconnected gate.
- `repower_gate`: changes the size of a gate.
- `remove_gate`: deletes a disconnected gate.

Net-level:

- `insert_net`: creates an empty (no parasitics) and disconnected net in the design.
- `read_spef`: asserts parasitics on existing nets.
- `remove_net`: removes a net from the design.

Pin-level:

- `connect_pin`: connects a pin to a net in the design.
- `disconnect_pin`: disconnects a pin from its net.

Timing queries. This set of commands probe the design to report timing information.

- `report_at`: prints the arrival time at a pin for any transition and mode.
- `report_rat`: prints the required arrival time at a pin for any transition and mode.
- `report_slack`: prints the post-CPPR slack at a pin for any transition and mode.
- `report_worst_paths`: prints paths with the worst non-positive post-CPPR slacks either through a specified pin or having the worst slacks in the design.

G. Output

From the set of commands present in the **.ops** file, only the timing queries have expected output. The first three commands `report_at`, `report_rat`, and `report_slack` prints a single scalar value. The format for `report_worst_paths` has the following syntax:

```
report_worst_paths: <paths reported>
Path <#>: <type> <path slack> <len> <mode>
<Deepest pin in path> <transition>
<Immediate upstream pin> <transition>
...
```

where `<paths reported>` is the number of printed paths (with non-negative slacks). For each path, the `<type>` can either be `Setup` or `Hold`, where the path originates from a timing test, or `RAT`, where the path is traced from a primary output. The `<path slack>` is the path slack after CPPR analysis, the `<len>` is the number of pins through the path, and the `<mode>` is either early (`E`) or late (`L`). The path is traced in an upstream manner, starting with the either the data pin of a sequential element or a primary output, and ending with a primary input. The `<transition>` specifies the transition of the path at a particular pin. The following shows an example output:

```
report_worst_paths: 1
Path 1: hold -30.0 6 E
f1:d F
u4:o F
u4:a R
u1:o R
u1:a F
inp1 F
```

TABLE I. CONTEST BENCHMARKS AND STATISTICS.

Design	Number of:			
	PIs	POs	Gates	Nets
c3_slack	4	3	3	7
c17	5	2	6	11
c17_slack	5	2	6	11
c432	36	7	134	170
c499	41	32	176	217
c1355	41	32	180	221
c1908	33	25	222	255
c2670	157	63	344	501
c3540	50	22	691	741
c5315	178	123	918	1.1K
c6288	32	32	1.7K	1.7K
c7552	206	107	1.1K	1.4K
c7552_slack	206	107	1.1K	1.4K
ff	2	3	11	14
s27	6	1	28	34
s344	11	11	182	193
s349	11	11	194	205
s386	9	7	177	186
s400	5	6	221	226
s510	21	7	312	291
s526	5	6	304	309
s1196	16	14	641	657
s1494	10	19	804	814
ac97_ctrl	84	48	14.3K	14.4K
aes_core	260	129	22.9K	23.2K
des_perf	235	64	105.4K	106.5K
mem_ctrl	115	152	10.5K	10.7K
pci_bridge32	162	207	19.1K	19.3K
systemcaes	260	129	6.5K	6.8K
systemcdes	132	65	3.4K	3.6K
tv80	14	32	5.3K	5.3K
usb_funct	128	121	15.7K	15.9K
vga_lcd	89	109	139.5K	139.6K
wb_dma	217	215	4.2K	4.4K
cordic_ispd	34	64	45.4K	45.4K
des_perf_ispd	234	140	138.9K	139.1K
edit_dist_ispd	2.6K	12	147.6K	150.2K
fft_ispd	1.0K	2.0K	38.2K	39.2K
matrix_mult_ispd	3.2K	1.6K	155.3K	167.2K
pci_bridge_32_ispd	160	201	40.8K	41.0K
usb_phy_ispd	15	19	923	938
cordic_dut	80	78	3.6K	3.6K
crc32d16N	19	32	478	495
softusb_navre	34	51	6.9K	7.0K
tip_master	778	869	37.7K	38.5K
b19_iccad	22	25	255.3K	255.3K
mgc_edit_dist_iccad	2.6K	12	161.7K	164.2K
mgc_matrix_mult_iccad	3.2K	1.6K	171.3K	174.5K
vga_lcd_iccad	85	99	259.1K	259.1K
netcard_iccad	1.8K	10	1496.0K	1497.8K
leon2_iccad	615	85	1616.4K	1517.0K
leon3mp_iccad	254	79	1247.7K	1248.0K

VII. CONTEST BENCHMARKS

Table I presents the list of all benchmarks used in the TAU 2015 timing contest. The table shows the number of primary inputs (PIs), primary outputs (POs), nets and gates for each benchmark. The first set of benchmarks (e.g., c17, c2670) are purely combinational benchmarks. The second set (e.g., s27, s386) are simple sequential designs. The third set (e.g., systemcaes) are more complex sequential designs based on OpenCore designs. The fourth set (e.g., fft_ispd) are larger sequential benchmarks based on industrial designs. The fifth set (e.g., cordic_dut) are industry-based sequential designs that have both clock buffers and inverters. The sixth set (e.g., b19_iccad) are based on the ICCAD 2014 timing-driven placement benchmarks.

The first two sets of benchmarks are those based on the TAU 2013 v1.0 benchmarks, and were released in the initial phase of the contest. The third set, based on the TAU 2013 v2.0 benchmarks and the fourth set, based on the ISPD 2013 Gate Sizing Contest [6] were released during the contest to further stress contestant submissions. The fifth and sixth sets, based on designs from Cadence Design Systems and the ICCAD 2014 contest, were partially released to the contestants, and used during evaluation.

VIII. Evaluation

The timer is evaluated on its (i) accuracy relative to a "golden" output, and (ii) overall performance, which is quantified by run-time and memory usage. To measure the accuracy, the timer's output file, uniquely identified by the benchmark and associated **.ops** file, is evaluated based on path-based and gate-based timing metrics.

Path-based accuracy is based on the output from the `report_worst_paths` command. The slack accuracy A_{path} of all the reported critical paths is compared with the golden. If the slack difference falls outside of a set threshold, no score is awarded. Similarly, if the timer's report does not include a path that is present in the golden, the timer will not receive credit.

Gate-based accuracy is evaluated using the `report_at`, `report_rat`, and `report_slack` commands. Their values are compared to the golden output, and are measured by A_{at}, A_{rat}, and A_s, respectively.

Design accuracy. Given all path-based and gate-based accuracy scores, the (raw) accuracy measurement $A(o)$ for an output file is the conglomerate of the different path-based and gate-based accuracy scores that both evaluate the overall quality as well as the worst quality of the timer. This includes, but is not limited to, the average of the accuracy scores of all pin-based timing queries and the accuracy score of the most critical path. The output files are targeted towards queries that stress both breadth (e.g., reporting the most critical paths in the design) and depth (e.g., reporting the most critical paths through a specified pin).

Run-time and memory usage. The timer's performance — run-time and memory usage — are applied as modifiers to $A(o)$, yielding a weighted accuracy score $A_w(o)$ for a given benchmark and **.ops** file.

Ranking. The final score is the average over all $A_w(o)$.

IX. Summary

Incremental timing analysis and incremental common path pessimism removal are essential tools during any timing-driven design-flow step such as physical design or synthesis. When small portions of the design has been modified (e.g., gate repowering, logic resynthesis, buffering), re-analyzing the full design is infeasible from a turnaround time and productivity perspective.

The aim of the TAU 2015 timing contest is to explore novel ideas for quickly updating the timing profile of a design in the midst of physical or logical design changes. The TAU 2015 timing contest (i) introduces the concepts and importance of incremental timing analysis 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 incremental timing analysis and CPPR framework with benchmarks.

Acknowledgments

The authors acknowledge Jobin Jacob Kavalam, Neel Gala, Sudharshan V, Professor Nitin Chandrachoodan and Professor Shankar Balachandran (IITiMer team) from the Indian Institute of Technology, Madras, India for sharing the source code of their timing analysis tool (winners of [9]). The authors also acknowledge Tsung-Wei Hwang, Pei-Ci Wu and Professor Martin D. F. Wong (UITimer team) from the University of Illinois at Urbana-Champaign for sharing their source code (winners of [4]).

References

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

[2] W. C. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wide-band Ampliers", *Journal of Applied Physics*, 19(1)(1948), pp. 55-63.

[3] R. Gupta, B. Tutuianu and L. T. Pileggi, "The Elmore Delay as a Bound for RC Trees with Generalized Input Signals", *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 16(1)(1997), pp. 95-104.

[4] J. Hu, D. Sinha and I. Keller, "TAU 2014 contest on removing common path pessimism during timing analysis", *Proc. International Symposium on Physical Design*, 2014, pp. 153-160.

[5] C. V. Kashyap, C. J. Alpert, F. Liu and A. Devgan, "Closed-form Expressions for Extending Step Delay and Slew Metrics to Ramp Inputs for RC Trees", *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 23(4)(2004), pp. 509-516.

[6] M. M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke and C. Zhuo, "An Improved Benchmark Suite for the ISPD-2013 Discrete Cell Sizing Contest", *Proc. of International Symposium on Physical Design*, pp. 168-170, 2013.

[7] P. Penfield Jr. and J. Rubinstein, "Signal Delay in RC Tree Networks", *Proc. Design Automation Conference*, 1981, pp. 613-617.

[8] C. L. Ratzlaff and L. T. Pillage, "RICE: Rapid Interconnect Circuit Evaluation Using AWE", *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 13(6)(1994), pp. 763-776.

[9] 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.

[10] The TAU 2015 Contest Website. `https://sites.google.com/site/tau2015contest`