TAU 2018 Contest

Path Reporting

**Contest Education**

v1.0 – December 7th, 2017

<https://sites.google.com/view/taucontest2018>

[Introduction 2](#_Toc500446790)

[1 Static Timing Analysis (STA) 2](#_Toc500446791)

[1.1 Timing Propagation 3](#_Toc500446792)

[1.2 Interconnect Modeling 5](#_Toc500446793)

[1.3 Circuit Element Modeling 6](#_Toc500446794)

[1.4 Timing Reports 10](#_Toc500446795)

[1.4.1 Overview 10](#_Toc500446796)

[1.4.2 A typical timing report 10](#_Toc500446797)

[2 Common Path Pessimism Removal (CPPR) 11](#_Toc500446798)

[2.1 CPPR Example 12](#_Toc500446799)

[3 Timing Macro Modeling 14](#_Toc500446800)

[3.1 Serial Compression 15](#_Toc500446801)

[4 Timing-driven Operations 16](#_Toc500446802)

[4.1 Sample Cases to Consider 18](#_Toc500446803)

[5 Things to Consider 19](#_Toc500446804)

[Updates 21](#_Toc500446805)

1

# Introduction

The TAU 2018 contest will focus on the reporting capabilities of a timing tool. The specific objective is to build a runtime and memory efficient timing path enumeration and reporting capability which allows user to generate top N critical timing reports on a specific cone of logic.

This document outlines the technical concepts related to static timing analysis, including timing calculation and propagation and common path pessimism removal used by *OpenTimer*. For contest file formats, including input files and deliverables, please refer to contest file formats.pdf, which is posted on the contest website above. For contest logistics, deadlines and any other updates, please refer to the contest website.

The remainder of this document is structured as follows. Section 2 introduces *static timing* *analysis (STA)*. Section 3 provides a conceptual understanding of *common path pessimism removal (CPPR)*. Section 4 outlines the basics of creating a timing model of a design. Section 5introduces the relevant commands that are supported by the reference timer.

# 1 Static Timing Analysis (STA)

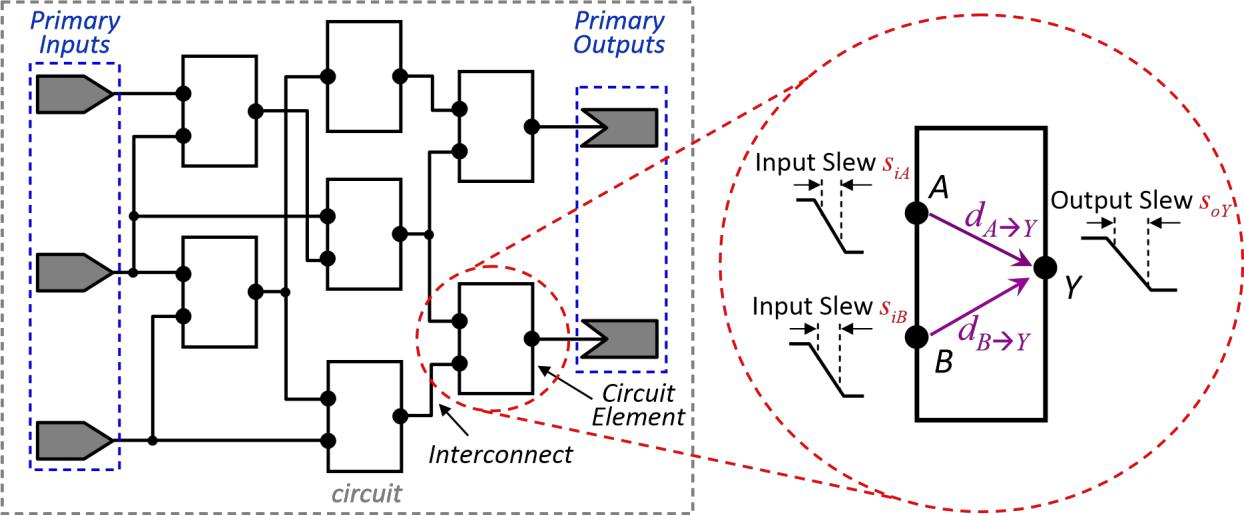


Figure 1: Generic circuit (left) and delay model representation of a circuit element (right).

Timing analysis computes the amount of time signals propagate in a circuit from its **primary** **inputs (PIs)** to its **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 somelater time; each element therefore introduces a *delay* during signal propagation. Furthermore,

2

assume that signal transitions are characterized by their *input slew* and their *output slew*, which is defined as the amount of time required for a signal to transition from **high-to-low** or **low-to-high**.1For example, as shown in Figure 1 (right), the delay across the circuit element from input*A* to output *Y* is designated by *dA*→*Y* , the input slew at *A* by *siA*, and the output slew at *Y* by *soY* .Here, both the delay and the output slew are functions of input slew. Section

2.1 will outline timing propagation, while Sections 2.2 and 2.3 will describe delay modeling.

## 1.1 Timing Propagation

Starting from the primary input(s), we quantify the instant that a signal reaches an input or output of a circuit element as the *arrival time (at)*. Similarly, starting from the primary output(s), we quantify the limits imposed for each arrival time to ensure proper circuit operation as the *required arrival time (rat)*. Given an arrival time and a required arrival time, we define the *slack* at a circuit node as a measurement of 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.

To account for multiple sources of within-chip variation, 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.2 By convention, if the early or late mode is not explicitly stated, both modes will be need to be considered. For example, a generic output slew *so* that is a function of input

slew *si* implies that the early mode *searlyo* is a function of early mode *searlyi* , and the late mode  is a function of late mode *slatei* .

**Actual arrival time.** Starting from the primary inputs, arrival times (*at*) are computed byadding delays across a path, and performing the minimum (in early mode) or maximum (in late mode) of such accumulated times at a convergence point. That is, in early mode, we are concerned with computing the earliest time instant that a signal transition can reach any given

circuit node. For example, let and  to be the early arrival times at pins *A* and *B* in Figure 1 (right). Then the early mode arrival time at the output pin *Y* will be



(1)

Conversely, in late mode, we are concerned with computing the latest time instant that a signal transition can reach any given circuit node. Following the same example in Figure 1 (right), the late mode arrival time at *Y* will be



(2)

1. Here, a **low (high)** signal is defined as 10% (90%) of the voltage.
2. This is typically done by *derating* an existing delay value, e.g., by ±5%. For the purposes of this contest, we will base the delays on early and late input slews (see Sections 2.2 and 2.3)

3

**Required arrival time.** Starting from the primary outputs, required arrival times (*rat*) arecomputed by subtracting the delays across a path, and performing the maximum (in early mode) or minimum (in late mode) of such accumulated times at a convergence point. That is, in early mode, we are concerned with computing the earliest time instant that a signal transition must reach any circuit node. For example, in Figure 2 (left), the early mode required arrival time at the input pin *Z* will be



(3)

Conversely, in late mode, we are concerned with computing the latest time instant that a signal transition must reach any given circuit node. Following the same example in Figure 2 (left), the late mode required arrival time at the input pin *Z* will be



|  |  |  |
| --- | --- | --- |
|  | (4) |  |
| **Slacks.** For proper circuit operation, the following conditions*must*hold: |  |  |
| *atearly* ≥ *ratearly* | (5) |  |
|  |  |
| *atlate* ≤ *ratlate* | (6) |  |

To quantify how well timing constraints are met at each circuit node, slacks (*slack*) can be computed based on Equations 5 and 6. That is, slacks are positive when the required times are met, and negative otherwise.

|  |  |
| --- | --- |
| *slackearly* = *atearly* − *ratearly* | (7) |
| *slacklate* = *ratlate* − *atlate* | (8) |

**Slew propagation.** As circuit element delays and interconnect delays are a function of the inputslew (*si*), the subsequent output slew (*so*) must be propagated. In this contest, we will assume *worst-slew propagation*, where we propagate the smallest (largest) slew in early (late) mode.Following the example in Figure 1 (right), the early mode and late output slew at output pin *Y* are, respectively:



(9)

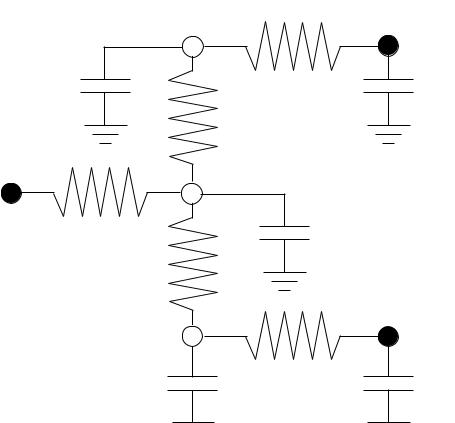


(10)

**Transitions.** For each timing arc, delay and output slew values will propagate only fortransitions that exist. For example, suppose there two timing arcs in serial, where the first timing arc propagates rise-to-rise (R→R) and fall-to-fall (F→F), and the second timing arc propagates fall-to-rise (F→R). After timing analysis, the only valid output transition at the second timing arc will be rise (R). The delay through both the timing arcs is the sum of the delay for the F→F transition in first arc plus the delay for the second arc for the F→R transition in the second arc. Note that the delay for the R→R delay from the first arc is not used, and the fall arrival time for the second arc is undefined. For this contest, an *undefined early (late) arrival*

4

*time* is set as 987654 (-987654), and an *undefined early (late) required arrival time* is set as (-987654) (987654).



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 1 | *T*1 |  |
|  |  |  |  |
| *oT* 1 | *C* 1 | *RD* | *C*4 |  |
|  |  | *RC* |  |  |
|  |  | 3 |  |  |
| *Z* | *RA R* |  | *C*3 |  |
|  | *B* |  |  |  |
| *oT* 2 |  |  |  |  |
|  | 2 |  | *T*2 |  |
|  | *C*2 | *RE* | *C*5 |  |

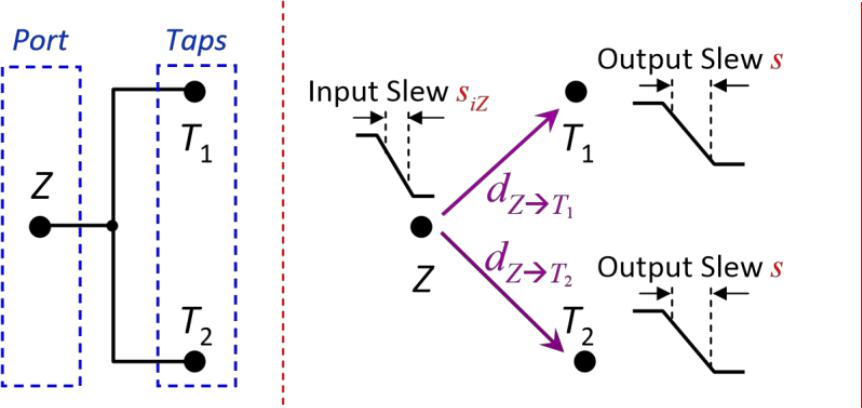


Figure 2: Generic interconnect (left), its timing model (center) and *RC* network (right).

## 1.2 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*), as illustrated in Figure 2 (left). Parasitic *RC* trees only contain grounded capacitors and floating resistors (we will not include the discussion of coupling capacitors or grounded resistors).

**Delay.** The computation of port-to-tap delays can be accurately performed through electricalsimulation. However, and for the sake of simplicity (and speed), we will assume the simpler Elmore delay model [1], 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 [4].

In an RC network, consider any two nodes *e* and *k*. Let *Ck* be the lumped capacitance at node *k*, and let *Rk*→*e* be the total resistance of the common path between the paths from *Port* to *e* and *Port* to *k*. For example, in Figure 2 (right), the resistance between nodes 1 and *T*2

(*R*1→*T*2) is *RA*, as that is the only common resistor between the paths *Z* to 1 and *Z* to *T*2. The Elmore delay at node *e* is

|  |  |
| --- | --- |
| *de* = X *Rk*→*eCk* | (11) |
| *k*∈*N* |  |

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

*dT*2= *RAC*1+ *RAC*3+ *RAC*4+ (*RA* + *RB*)*C*2+ (*RA* + *RB* + *RE*)*C*5(12) = *RA*(*C*1+ *C*3+ *C*4) + (*RA* +

*RB*)*C*2+ (*RA* + *RB* + *RE*)*C*5

5

**Output slew.** The value of the output slew (*so*) on any given tap node*T*can be approximated bya two-step process. First, compute the output slew of the impulse response on *T*, which was observed [1,2] to be well-approximated by

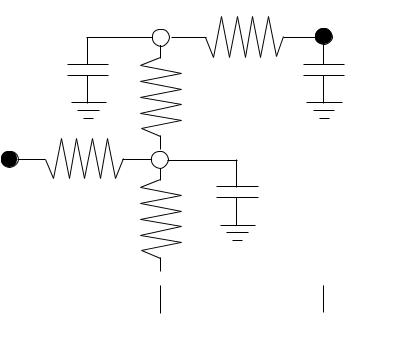
|  |  |
| --- | --- |
| q |  |
| *s*ˆ*oT* ≈2*βT* − *dT* 2 | (13) |

where *βT* is the second moment of the input response at node *T*, and *dT* is the corresponding Elmore delay from Equation 11. Second, compute the slew of the response to the input ramp by the expression given in [3]

q

|  |  |
| --- | --- |
| *soT* ≈ *si*2+ *s*ˆ2*oT* | (14) |

where *si* is the input slew.



|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 | *T*1 |  |
|  |  |  |
| *C* 1*d*1 | *RD* | *C* 4*d*4 |  |
|  | *RC* |  |  |
|  | 3 |  |  |
| *ZR A R B* |  | *C* 3*d*3 |  |

1.  *T*2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *C* 2*d*2 |  |  |  |  |  |  | *RE* |  |  |  |  |  |  |  |  | *C* 5*d*5 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Figure 3: Modified RC network for output slew calculation.

The value of *βT* can be computed through the efficient path-tracing algorithm for moment computation proposed in [5], which is a generalization of the algorithm proposed in [1]. To calculate *βT* , first replace all capacitance values *Ck* in the RC network by *Ckdk*, where *dk* is the Elmore delay from Equation 11 (see Figure 3). Second, follow the same procedure as before for finding *βT*

|  |  |
| --- | --- |
| *βT* = X *Rk*→*T Ckdk* | (15) |
| *k*∈*N* |  |

Following the example in Figure 3, at node *T*2, we have *βT*2 = *RA*(*C*1*d*1 + *C*3*d*3 + *C*4*d*4) + (*RA* + *RB*)*C*2*d*2

+ (*RA* + *RB* + *RE*)*C*5*d*5 (16)

## 1.3 Circuit Element Modeling

For delay and output slew calculations between two pins, the information will be given in the **.lib** file as two-dimensional tables. To find the corresponding timing information, extrapolationor interpolation will be necessary.

6

If the table contains a single value, i.e., a 1x1 table (Figure 4 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 1x*n* table or a *m*x1 table (Figure 4 center), then the value will depend only on the non-scalar dimension. For example, consider the 1x4 table in Figure 4. If *y < y*1, then the corresponding output *z* value will be the linear extrapolation between *z*1 and *z*2. If *y*2 ≤ *y* ≤

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *y*3, then *z* will be | | | | | the linear interpolation between *z*2 and | | | | | | | | | | | *z*3. If *y*4 *< y*, then *z* | | | | | | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | *xx*1 |  |
|  |  |  |  | *x*3 |  |  |  |  |  |  |  |  | *z*1 |  |  |  |  | *z*11 |  |  | *z*21 |  |  | *z*31 |  | *x*2 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | *z*2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | *z*12 |  |  | *z*22 |  |  | *z*32 |  |  |  |
|  |  | *z* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | *z*1 | *z*2 |  | *z*3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | *z*3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | *y*1*y*1 *xx*1 | | | *x*2 | | |  |  |  |  |  |  | *z*13 |  |  | *z*23 |  |  | *z*33 |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | *z*4 |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | *x*3*y*2*y*2 | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | *z*14 |  |  | *z*24 |  |  | *z*34 |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

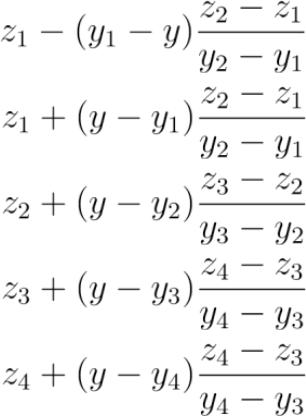
*yy*

*y*3*y*3 *y*4*y*4

1x1 Table3x1 Table1x4 Table3x4 Table

Figure 4: Illustration of different tables: scalar, one-dimensional, and two-dimensional. will

|  |  |  |  |
| --- | --- | --- | --- |
| be the linear extrapolation between *z*3 and *z*4. |  |  |  |
| if *y < y*1 (17) *z*1 if *y* = *y*1 | | (18) |  |
| if *y*1 *< y < y*2 | (19) *z*2 if *y* = *y*2 | | (20) |
| if *y*2 *< y < y*3 | (21) *z*3 if *y* = *y*3 | | (22) |
| if *y*3 *< y < y*4 | (23) *z*4 if *y* = *y*4 | | (24) |
| if *y>y*4 (25) |  |  |  |



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 4. If *x*2 *< x < x*3 and *y*2 *< y < y*3, then (*i*) determine *zfirst* by linear interpolation on *z*22 and *z*32, (*ii*) determine *zsecond* by linear interpolation on *z*23 and *z*33, and then (*iii*) determine *z* by linear interpolation using

*zfirst* and *zsecond*.

7

**Combinational elements.** For a given combinational cell, e.g., OR gate, let the delay*d*andoutput slew *so* for a input/output pin-pair (see Figure 5) be calculated by non-linear delay model interpolation/extrapolation. These delay and output slew tables are stored in the **.lib**, and are referenced by the input slew (*x*) and driving load (*y*). *CL* denotes the equivalent downstream capacitance seen from the output pin of the cell. Several sophisticated models have been proposed for computing *CL*. For simplicity, the application of such models is considered to be out of the scope of the present contest, and a simple model is adopted. *CL* 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.

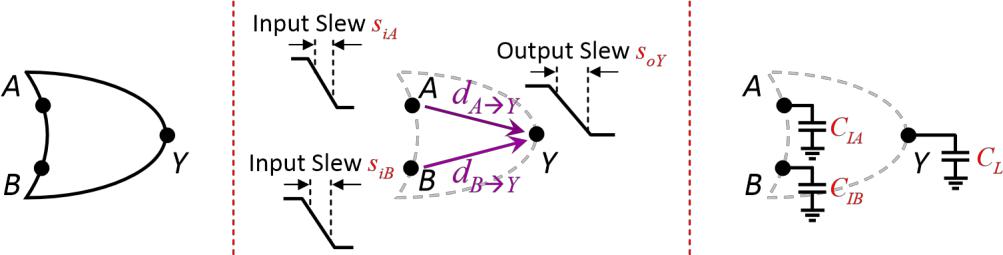


Figure 5: Combinational OR gate (left), its timing model (center) and capacitances (right).

**Sequential elements.** Sequential circuits consist of combinational blocks interleaved by*registers*, usually implemented with *flip-flops (FFs)*. Typically, sequential circuits are composedof several stages, where a register captures data from the outputs of a combinational block from a previous stage, and injects it into the inputs of the combinational block in the next stage. Register 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 l*.

A (D) flip-flop is a storage element that 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 and its complement at the output pins *Q* and *Q*. The flip-flop also enables asynchronous preset (set) and clear (reset) of the output pins through the respective *S* and *R* input pins.3

1. The complement, preset and clear signals are stated here for completeness. For the purposes of the contest, their behavior will be ignored.

8

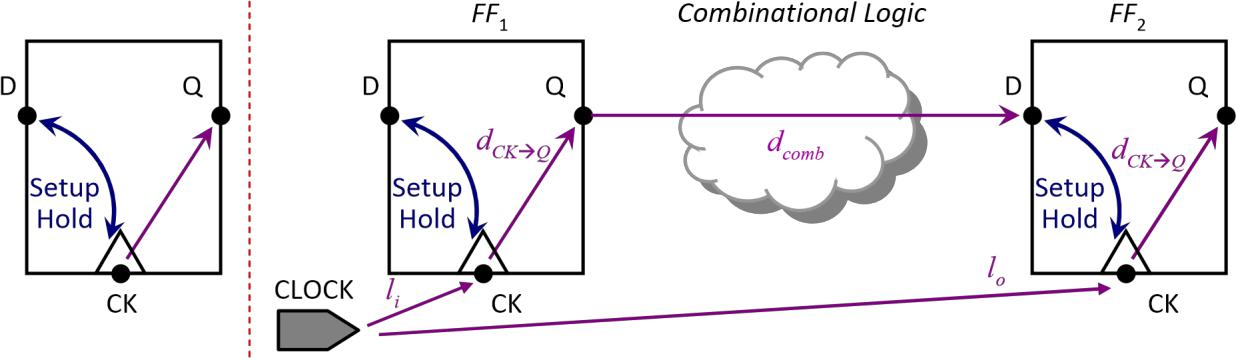


Figure 6: Generic D flip-flop and its timing model (left), and two FFs in series and their timing models (right).

**Setup and hold constraints.** Proper operation of a flip-flop requires the logic value of the inputdata pin to be stable for a specific period of time *before* the capturing clock edge. This period of time is designated by the *setup time tsetup*. 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 thold*. The flip-flop timing models are depicted in Figure 6 (left). The test time are given in the **.lib** as two-dimensional tables, and are referenced by the clock-side input slew (*x*) and the data-side input slew (*y*).

**Signal propagation.** Consider the standard signal transition between two flip-flops asillustrated in Figure 6 (right). Assuming that the clock edge is generated at the source at time 0, it will reach the injecting (launching) flip-flop *FF*1 at time *li*, making the data available at the input of the combinational block *dCK*→*Q* time later. If the propagation delay in the combinational block is *dcomb*, then the data will be available at the input of the capturing flip-flop *FF*2 at time *li*

* *dCK*→*Q* + *dcomb*. Let the clock period to be a constant *T*. Then the next clock edge will reach *FF*2at time *T* +*lo*. For correct operation, the data must be be available at the input pin *D* of *FF*2 *tsetup* time before the next clock edge. Therefore, at the data input pin *D* of *FF*2, we have the following:



(26)



(27)

A similar condition can be derived for ensuring that the hold time is respected. The data input pin *D* of *FF*2 must remain stable for at least *thold* time after the clock edge reaches the corresponding *CK* pin. Therefore, at the data input pin *D* of *FF*2, we have the following:

*atearlyD* = *liearly* + *dCK*→*Q* + *dearlycomb* (28) *rathold* = *ratearlyD* = *lolate* +

*thold* (29)

Note that when computing the required arrival times in Equations 27 and 29, the value *lo* is specific to Figure 6. In the general case, *lo* should be replaced with *atC*. The previous arrival times and required arrival times induce setup and hold slacks, which can be computed from Equations

9

7 and 8. For the clock pins of the flip-flop, the required arrival time is derived from the test slack. For early mode, the slack at the clock pin is the setup or late test slack, and for late mode, the slack at the clock pin is the hold or early test slack. From the corresponding test slack and arrival time, the clock required arrival time can be derived, and appropriately propagated.

## 1.4 Timing Reports

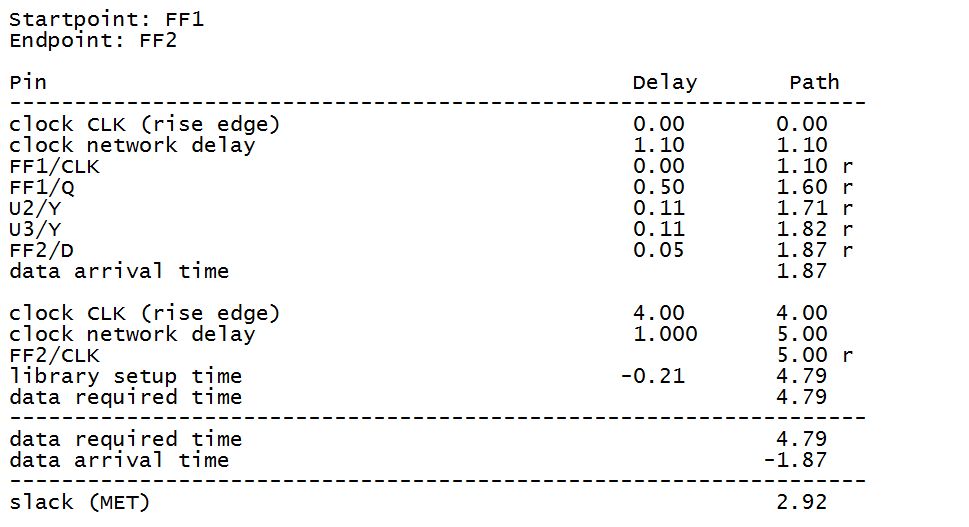
### 1.4.1 Overview

The timing reports show the timing paths in the current design that have the worst slack. These are the paths that violate the timing constraints by the largest amounts, or paths with positive slack that come closest to causing a timing violation.

Each path has a startpoint and an endpoint. Data is launched by a clock edge at the path startpoint, propagated through combinational logic in the path, and then captured at the path endpoint by another clock edge. The startpoint can be a register clock pin or an input port. The end point can be a register data input pin or an output port. To restrict the scope of the report to only the paths that start, pass through, or end on specific pins, ports, nets, or cell instances, use reporting commands often have the “from”, “though”, and “to” options.

While different timing tools might have different reporting options, they all need to constraint the number of paths. By default, the reporting command usually reports the single path in the design with the worst violation. To control the number of paths reported, the reporting command needs to have the “num\_worst” option, which specifies the maximum number of worst path reported per endpoint; and the “max\_paths” option, which specifies the overall maximum number of paths reported by the command.

### 1.4.2 A typical timing report



Data

arrival

Data

required

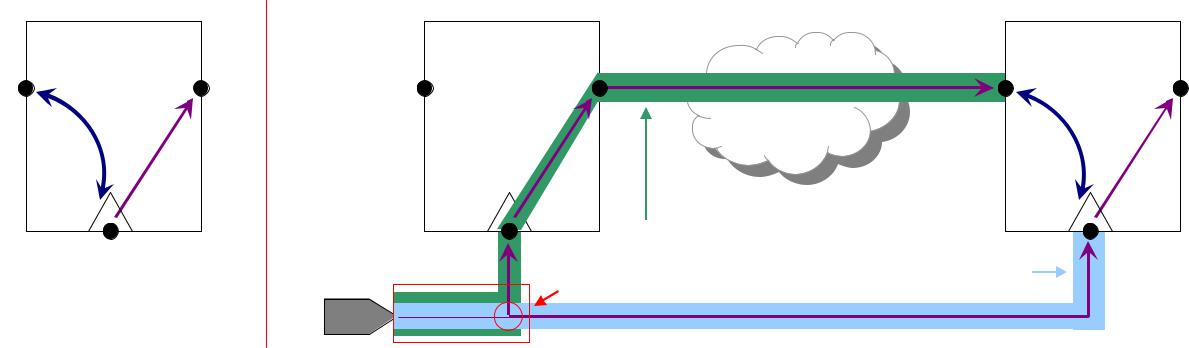
Slack

value

The above timing report shows the incremental delay contributed by each pin-to-pin segment of the path (Delay column) and the cumulative path delay up to each pin in the timing path (Path column). The table also shows the data launch clock arrival time, the data capture clock arrival time, an accounting of arrival time versus required time, and the resulting timing slack for the path. A positive slack (2.92) indicates a non-violated path.

# 2 Common Path Pessimism Removal (CPPR)

The early-late split-timing analysis has greatly enabled timers to effectively account for *any* within-chip variation effects. However, this dual-mode analysis inherently embeds *unnecessary* *pessimism*, which can lead to an over-conservative design. Consider the example in Figure 7,where we analyze the slack of a typical hold (setup) test. When we perform our dual-mode analysis, we compare the early (late) data’s arrival time against the late (early) clock’s arrival time. However, along the physically-common portion of the data path and clock path in the clock tree, the signal cannot simultaneously experience 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 should be passing (having positive slack). Therefore, this unnecessary pessimism should not be included when reporting final timing results.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *D* | *Q* | ***Launching FF*** | ***Capturing FF*** |  |
|  |  |  |
|  | *tH* |  |  |  |
|  | *tS* |  |  |  |
|  |  | *Combinational Logic* |  |  |
|  | *CK* | *Data Path* |  |  |
|  | *Common Point* | *Clock Path* |  |
|  | CLOCK |  |



Figure 7: Inherent but unnecessary pessimism introduced by the early/late timing split.

To the first order, 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 (see Figure 7). However, 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 can be 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, we need to take the minimum credit found across all paths.

**Hold tests.** For tests that compare the data point against the clock point*for the same cycle*, e.g.,where data must be stable *after* the clock signal arrives at the capturing flip-flop, the total pessimism incurred is the difference between the early and late arrival times at the common point. That is, any on-chip variation incurred is spatially and temporally the same. For a hold test *Thold* that has one data path that share common elements *and* common edges, the amount of credit given back is

10



(30)

where *cp* is the last point before the data path and clock path diverge (see Figure 7).

**Setup tests.** For tests that compare the data point against the clock point*in subsequent cycles*,e.g., where the data must be stable *before* the clock signal arrives at the capturing flip-flop, the total pessimism incurred is the summation of the difference of early and late delays up through the common point. While the data and clock path share the same physical components, they are launched at different clock cycles. Therefore, if some on-chip variation, e.g., temperature fluctuation occurs when the data is launched, but does not occur when the data is captured, the pessimism was now necessary, and cannot be removed. For a setup test *Tsetup* that has one data path sharing common elements *and* common edges, the amount of credit given back is

*creditsetup* = X *dlatep* − *dearlyp* (31)

*p*∈*P*(*V,E*)

where *P*(*V,E*) is the physically-common path between the data and clock paths. Here, *V* is the set of all common circuit elements, and *E* is the set of all common interconnect. Following the example in Figure 7, the amount of credit for the setup test is



**Total test credit.** As removing pessimism for each test could require investigating multiplepaths, the amount of credit per test will be defined as the difference between the post- and pre-CPPR test slack. For this contest, we will only consider hold and setup tests.



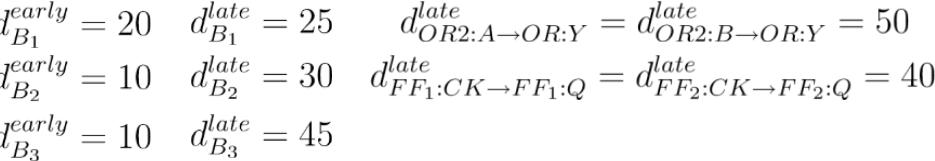
(33)

 (34) When analyzing CPPR credit, only consider the credit that comes from paths with

nonpositive slack.

## 2.1 CPPR Example

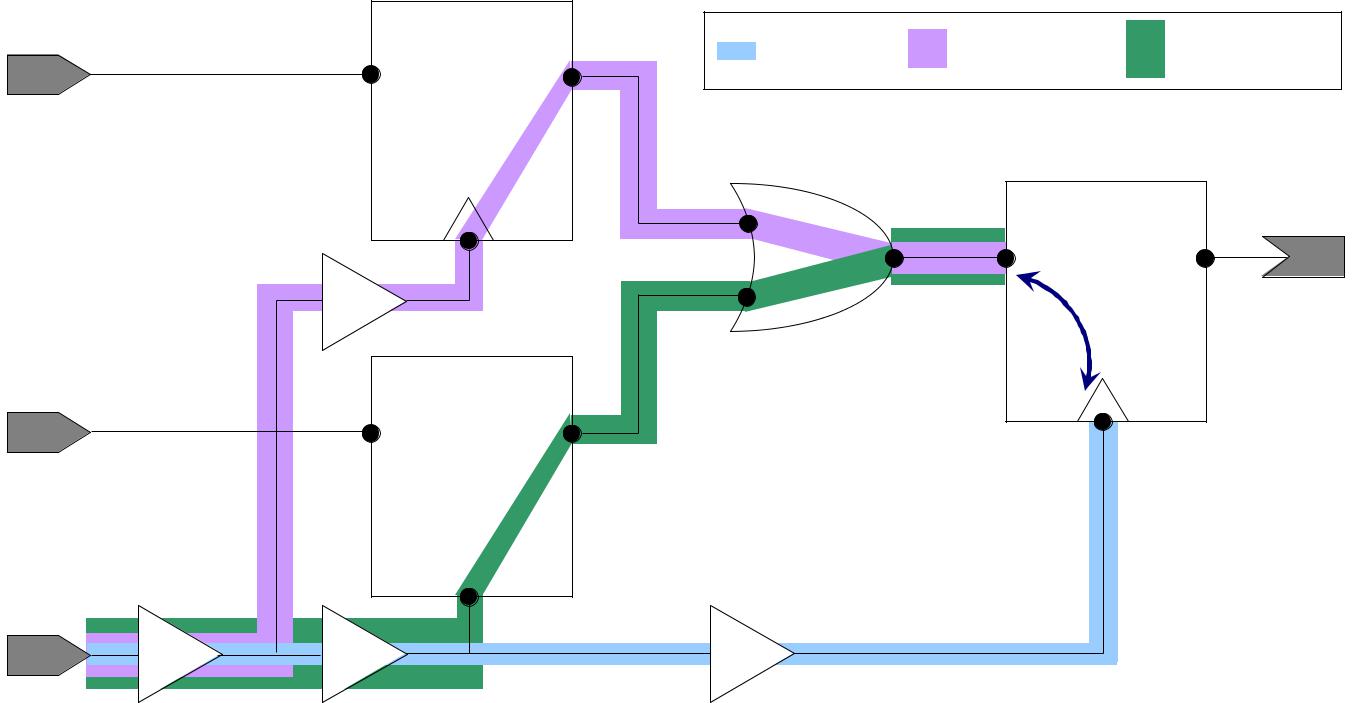
Consider the following sample circuit in Figure 8, where two data paths feed a common latch (*FF*3). Using the following timing information, we will determine the CPPR credit for the setup test at *FF*3. Please note that this example is used for conceptual understanding only. Assume that all wire delays are zero, and all arrival times are zero.



*TCLOCK* = 120



11



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *IN*1 | D | ***FF*1** | Q | *Clock Path* | *Data Path* 1 | *Data Path* 2 |  |
|  |  |  |
|  |  |  | CK | *A* | ***FF*3** | *OUT* |  |
|  |  |  |  | ***OR*2***Y* | D | Q |  |
|  | ***B*2** |  |  | *B* | Setup |  |  |
|  |  |  | Hold |  |  |
|  |  |  |  |  |  |  |
| *IN*2 | D | ***FF*2** | Q |  | CK |  |  |
|  |  |  |  |  |
|  |  |  |  |  |  |

CK

*CLOCK*

***B*1**  ***B*3**  ***B*4**

Figure 8: Example sequential circuit.

First, we find the data path slacks at *FF*3:*D*. We do this by comparing the late data arrival

time and the late required arrival time at *FF*3:*D*.



(35)

Using Equation 27 in Equation 35, we obtain

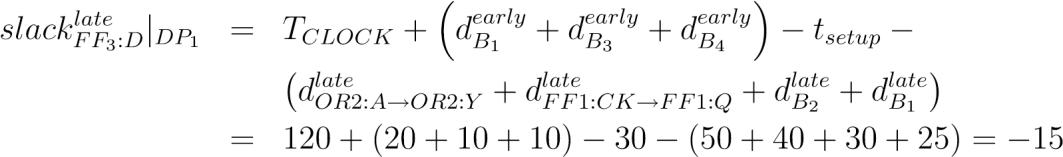


(36)

We now find the pre-CPPR slacks for each path. Consider Data Path 1 (*DP*1), where its timing is determined by going through the path

*CLOCK* → *B*1→ *B*2→ *FF*1→ *OR*2 → *FF*3

Using Equations 36 and 26, we obtain:

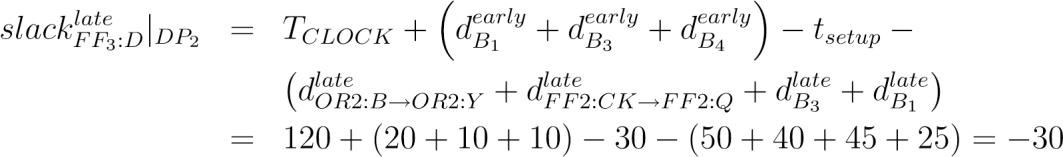


Similarly, for Data Path 2 (*DP*2), its path

*CLOCK* → *B*1→ *B*3→ *FF*2→ *OR*2 → *FF*3

12

has pre-CPPR path slack



We now compute the CPPR credit for each path (Equation 33). Without loss of generality, we will process paths in order of criticality, i.e., most negative to least negative. Starting with *DP*2, its common portion with the clock path contains *B*1and *B*3. Therefore, its credit is



Similarly, for *DP*1, its CPPR credit can be computed by finding the common portion, i.e., *B*1:



The post-CPPR path slacks for *DP*1 and *DP*2 are:

*late* *post*−*CPPR* *late*

*slackFF*3:*D*|*DP*1 = *slackFF*3:*D*|*DP*1 + *creditsetup*|*DP*1 = −15 + 5 = −10

*slackFFlate*3:*D*|*postDP*2−*CPPR* = *slackFFlate*3:*D*|*DP*2+ *creditsetup*|*DP*2= −30 + 40 = 10

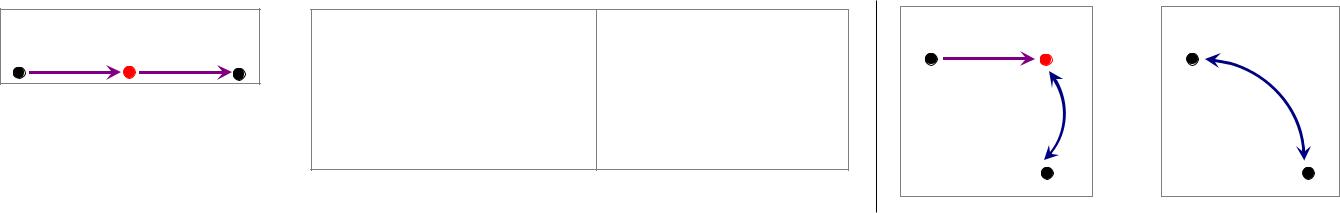
From this example, we make two observations. (*i*) The most critical path *pre-*CPPR is not necessarily reflective of the true critical path. This emphasizes the importance of CPPR analysis, and highlights its impact on chip performance. (*ii*) During CPPR, analyzing the single-most critical path can be insufficient. In our example, if we only applied credit according to *DP*2, the most critical pre-CPPR path, we would assume the test slack would be positive, i.e., passing. In actuality, the next-most critical path is still failing.

# 3 Timing Macro Modeling

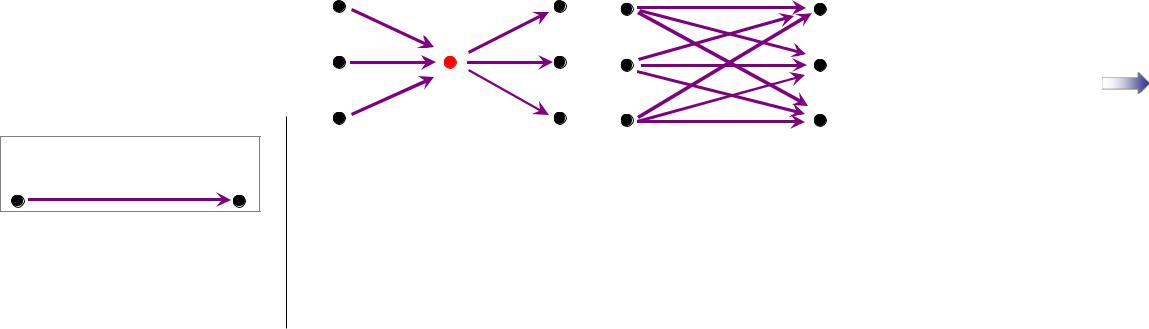
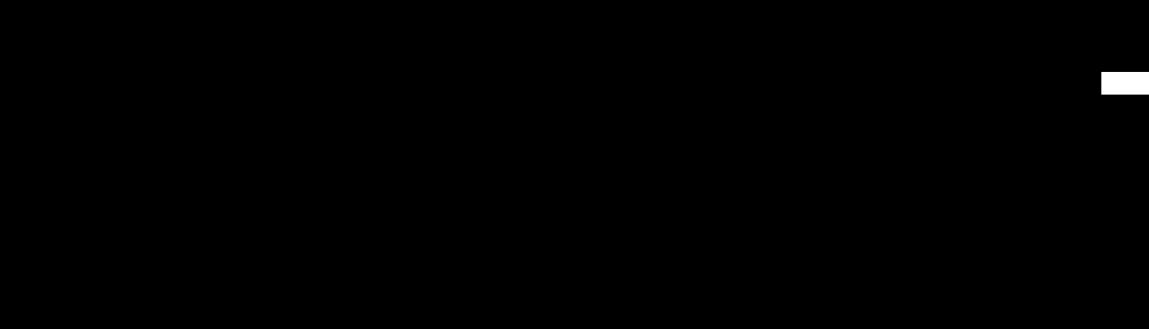
This section outlines the basics of creating a timing model for a design. Given the initial circuit, you will need to convert the benchmark into a timing graph. To model each combinational and sequential element, please refer to Section 2.3. One such timing graph is where there is a bijection between every gate and wire in the original circuit and every timing arc and test in the model, is the most detailed and the most accurate model you can construct. Another such timing graph is where there only exists direct connections between the input and outputs and timing tests between inputs. Your goal is to construct a timing model that successfully balances accuracy and runtime. Common methods to reduce the model size is to compression and

13

pruning of arcs and tests. Do not limit yourself to just this method – you are encouraged to find other methods and accuracy-runtime tradeoffs.



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A | B | C | A | B | A |  |
|  |  |  |  |
|  |  |  |  |  |  |
| A |  | C |  | C | C |  |



*7 nodes + 6 psegs = 13* *6 nodes + 9 psegs = 15*

*Retain Node* *Remove Node*

## 3.1 Serial Compression

Figure 9: Merging two timing arcs in serial (left), eliminating an intermediate node (center), and merging a timing arc and timing test in serial (right).

To combine two timing arcs in serial into one timing arc, the resultant timing arc must exhibit the same behavior as its original parents. Consider the example case in Figure 9 (left). Suppose the delay for arc A → B is 10, and the delay for B → C is 20. The resultant arc A → C then should have delay 30. By merging arcs A → B and B → C, we have reduced the model size by two

– two timing arcs and three nodes became one timing arc and two nodes. This compression method effectively has eliminated the intermediate node B. However, as shown in Figure 9 (center), it may not always be efficient to eliminate all intermediate nodes, as you must preserve all paths that cross from its fanin cone to its fanout cone. In the example, by eliminating the intermediate point, the model size has increased by two. From a model size perspective, this is undesired. However, if you believe that this tradeoff is worthwhile in terms of using the model (in-context), then you may choose to keep this overhead.

To combine a timing arc and a timing test, the resultant timing test must exhibit the same behavior as its original parents. Consider the example case in Figure 9 (right), where A and B are data points and C is a clock point. Suppose the timing arc A → B has early delay of 10, and the test B ↔ C is a hold test with guard time of 30. The resultant hold test A ↔ C must preserve the original (slack) check. Therefore, it must have an updated guard time of 30 - 10 = 20. Similarly, if the timing arc A → B has a late delay of 10 and the test B ↔ C is a setup test with a guard time of 30, then the resultant A ↔ C setup test must have a guard time of 30 - 10 = 20.

**3.2** **Parallel Compression**

For this contest, you are restricted to having *at most one* timing arc between two points (with the exception described in next paragraph), and *at most one* hold test and *at most one* setup test between two points. That is, between two points, you have a maximum of three connections: (*i*) one timing arc, (*ii*) one setup test, and (*iii*) one hold test. Therefore, if you have multiple timing

14

arcs or tests, you will need to eliminate all but one of them. To do this, you will need to determine which timing arcs or tests cause the worst-case or behavior in the appropriate mode.

If a timing arc between two points are non-unate arc, because the limitation of liberty syntax, if you model it with a single timing arc, you will have to merge the input rise/fall. In this case, we allow model this arc with two liberty timing arcs, one with positive unate (r->r, f->f) and one with negative unate (r->f, f->r). See file format document for more details on liberty format.

# 4 Timing-driven Operations

This section lists the supported set of operations for *OpenTimer* that will be used during the evaluation process. All commands can be used in the .ops file. There two sets of commands: timing-based operations and circuit-based operations. The former will be used to query timing during evaluation, and may or may not be helpful for you to use during your implementation. The latter have the ability to modify the circuit. They will only be used to add in extra logic during evaluation; you should not need them during your evaluation. These commands are only listed as a reference. For a detailed explanation of each command and syntax, please refer to contest file formats.pdf.

**Timing assertions.** This set of commands sets a custom timing value at a specific pin on thedesign.

* set at -pin *<*PI pin name*>* [-rise|-fall] [-early|-late] *<*value*>*: sets the arrival time in picoseconds (*ps*) at *<*PI pin name*>*, where the pin will be a primary input in the current design. Options -rise and -fall are mutually exclusive, and respectively specify the desired transition. Options -early and -late are mutually exclusive, and respectively specify the desired mode. By default, the -early -rise options are selected. In this command, the -pin switch is required. All other switches are optional.
* set slew -pin *<*pin name*>* [-rise|-fall] [-early|-late] *<*value*>*: sets the slew in picoseconds (*ps*) at *<*pin name*>*, where the pin will be a primary input in the current design. Options -rise and -fall are mutually exclusive, and respectively specify the desired transition. Options - early and -late are mutually exclusive, and respectively specify the desired mode. By default, the -early -rise options are selected. In this command, the -pin switch is required. All other switches are optional.
* set rat -pin *<*pin name*>* [-rise|-fall] [-early|-late] *<*value*>*: sets the required arrival time in picoseconds (*ps*) at *<*pin name*>*, where the pin will be a primary output in the current design. Options -rise and -fall are mutually exclusive, and respectively specify the desired transition. Options -early and -late are mutually exclusive, and respectively specify the desired mode. By default, the -early -rise options are selected. In this command, the -pin switch is required. All other switches are optional.

15

* set load -pin *<*PO pin name*> <*value*>*: sets the output load in femtofarads (*fF*) at *<*pin name*>*, where the pin will be a primary output in the current design. The value will be used for both early and late load. In this command, the -pin switch is required.

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

* report at -pin *<*pin name*>* [-rise|-fall] [-early|-late]: reports the arrival time in picoseconds (*ps*) at *<*pin name*>*, where the pin will be in the current design, i.e., no internal spef nodes. Options -rise and -fall are mutually exclusive, and respectively specify the desired transition. Options -early and -late are mutually exclusive, and respectively specify the desired mode. By default, the -early -rise options are selected. In this command, the -pin switch is required. All other switches are optional.
* report rat -pin *<*pin name*>* [-rise|-fall] [-early|-late]: reports the required arrival time in picoseconds (*ps*) at *<*pin name*>*, where the pin will be in the current design, i.e., no internal spef nodes. Options -rise and -fall are mutually exclusive, and respectively specify the desired transition. Options -early and -late are mutually exclusive, and respectively specify the desired mode. By default, the -early -rise options are selected. In this command, the -pin switch is required. All other switches are optional.
* report slack -pin *<*pin name*>* [-rise|-fall] [-early|-late]: reports the postCPPR slack in picoseconds (*ps*) at *<*pin name*>*, where the pin will be in the current design, i.e., no internal spef nodes. Options -rise and -fall are mutually exclusive, and respectively specify the desired transition. Options -early and -late are mutually exclusive, and respectively specify the desired mode. By default, the -early -rise options are selected. In this command, the -pin switch is required. All other switches are optional.

**Circuit modification commands.** This set of commands have the ability to change a design’stopology at the (*i*) gate-level, (*ii*) net-level, and (*iii*) pin-level. No output is expected. **Gate**-level:

* insert gate *<*new cell instance name*> <*new cell type*>*: creates a new gate in the design. This newly-created cell is *not* connected to any other gates or wires. The new gate name is guaranteed not to conflict with any existing names in the current design.
* repower gate *<*cell instance name*> <*new cell type*>*: changes the size or level of an existing gate, e.g., NAND2 X2 to NAND2 X3. The gate’s logic function and topology is guaranteed to be the same, along with the currently-connected nets. However, the pin capacitances of the new cell type could be different.
* remove gate *<*cell instance name*>*: removes a gate from the design. This is guaranteed to be called after the gate has been disconnected from the design (see pin-level commands below).

**Net**-level:

16

* insert net *<*net name*>*: creates an empty net object with the unique identifier *<*net name*>*. By default, it will not be connected to any pins, and have no parasitics (**.spef**). This net will be connected to existing pins in the design by the command connect pin, and parasitics will be loaded by the command read spef.
* read spef *<*.spef*>*: reads in a **.spef** file which includes a set of net parasitics. If the spef file contains a net that already has parasitics in the design, it should be overwritten. Any **.spef** files will be located in the same directory as the wrapper (**.tau2016**).
* remove net *<*net name*>*: removes a net from the design. By default, if a net is connected to pins, the pins should be automatically disconnected from the net. The corresponding parasitics should also be removed.

**Pin**-level:

* connect pin *<*pin name*> <*net name*>*: connects the pin to the corresponding net. The *<*pin

name*>* will either have the *<*cell name*>*:*<*cell pin name*>* syntax (e.g., u4:ZN), or be a primary input (e.g., inp1). The *<*net name*>* will match an existing net read in from a **.spef** file.

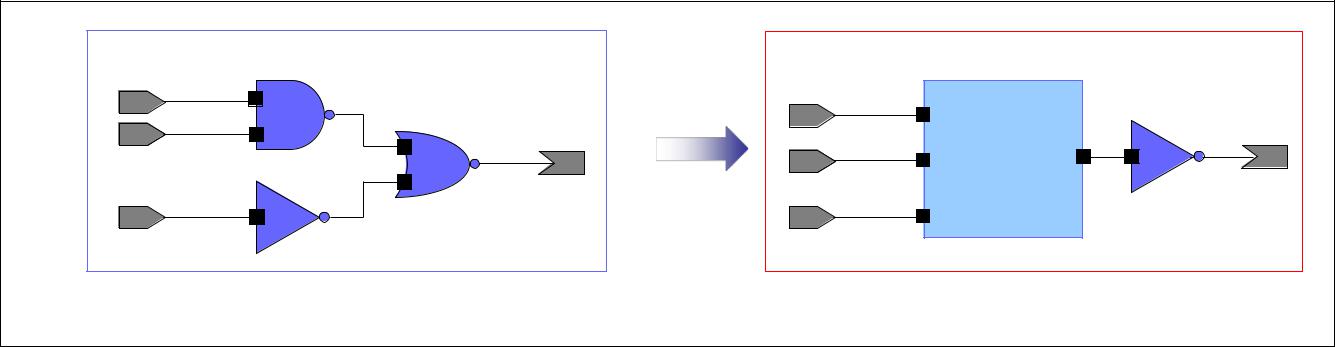
* disconnect pin *<*pin name*>*: disconnects the pin from the net it is connected to. The *<*pin

name*>* will either have the *<*cell name*>*:*<*cell pin name*>* syntax (e.g., u4:ZN) or be a primary input (e.g., inp1).

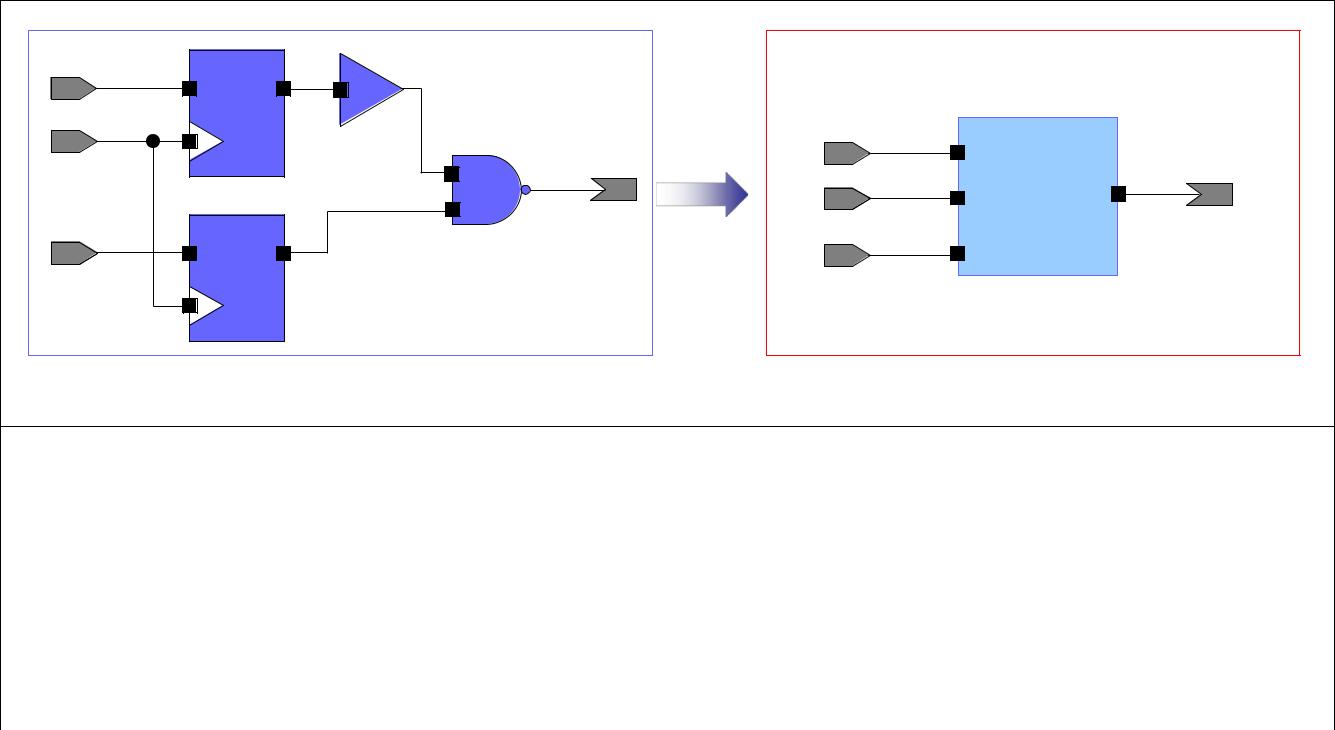
## 4.1 Sample Cases to Consider

When generating timing models, you will need to consider various types of topologies in the design, as well as the potential topologies that you could encounter when using the timing models. This is especially important for correct timing propagation and removing pessimism. Figure 10 illustrates a few cases for you to consider when considering which timing components (e.g., arcs and pins) you are allowed to eliminate, and which ones you must retain.

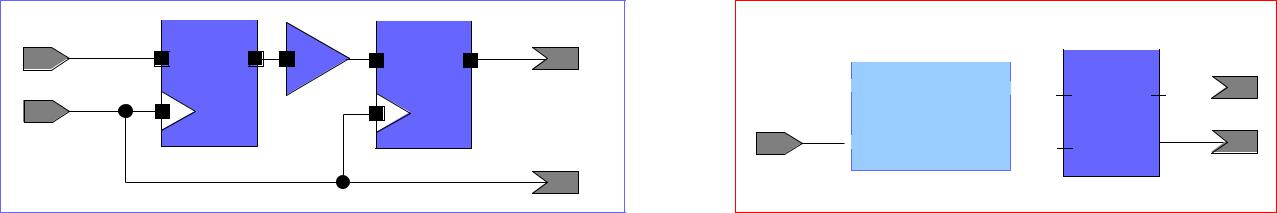
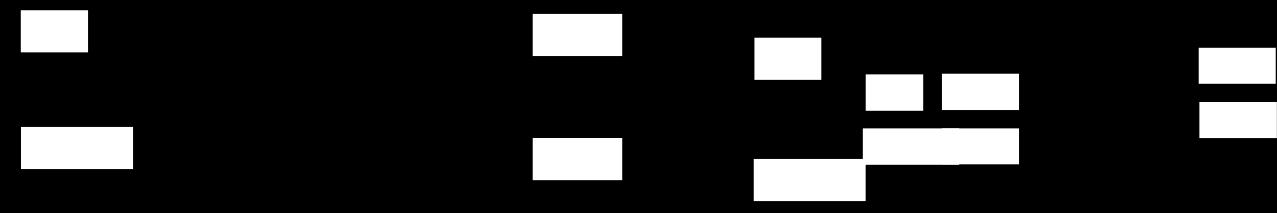
|  |  |  |  |
| --- | --- | --- | --- |
| **Original Design** | **In-context Usage** |  |  |
| **inp1** | **inp1** |  |  |
|  |  |  |
| **out** | **inp1** | **out** |  |
| **inp2 out** |  |
| **inp2** |  |  |
|  | **inp2** |  |  |
|  | **inp3** |  |  |
| **inp3** | **inp3** |  |  |



17



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **inp1** |  |  |  |  |
|  |  | **inp1** |  |  |
| **clock** | **out** | **inp1** | **out** |  |
|  | **inp2 out** |  |
|  |  |  |  |
|  |  | **inp2** |  |  |
|  |  | **clock** |  |  |
| **inp2** |  | **clock** |  |  |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **inp** |  |  |  | **out1** |  |
|  |  |  |

**inp**  **out1**  **inp out1** 



|  |  |  |
| --- | --- | --- |
| **clock** | **out2** |  |
|  |  |

**clock**

 **clock out2** 

**out2**

Figure 10: Different circuit topologies to consider.

# 5 Things to Consider

In the process of developing your macro modeler, you are strongly encouraged to do your own research on relevant topics and generate new ideas. In addition, we have provided some food for thought.

* **Model size reduction**: when generating your timing graph, what level of detail shouldyou start with and/or end with? You may start with the original design’s timing graph, and then start compressing and/or pruning edges. You may also start with a blackbox model and have only direct connections between input and outputs. Consider the advantages and disadvantages of both approaches (and any hybrid ones).
* **Model size**: when characterizing an arc or a test, how many table data entries do youneed? What is the range that you will need to maintain accuracy? How should you determine what your characterization points should be? Does the timing for each arc and each test change with different slews and loads at the primary inputs and primary outputs?
* **Model size**: the final evaluation will only compare slacks on all the PIs and POs except theclock PI. Can you use this to your advantage in order to reduce the model size?
* **CPPR**: how does CPPR affect the timing analysis, both in the original design and whenusing the macro model? Do you need to retain any logic in specific cases?

In addition, consider perusing the following references to aid you in your development.

18

* A. J. Daga, L. Mize, S. Sripada, C. Wolff, Q. Wu, “Automated timing model generation”, *DAC* 2002, pp. 146-151.
* C. W. Moon, H. Kriplani, “Timing model extraction of hierarchical blocks by graph reduction”, *DAC* 2002, pp. 152-157.
* D. Sinha, A. Bhanji, B. L. Dorfman, K. Kalafala, N. Venkateswaran, C. Visweswariah,

“Performing a statistical timing abstraction for a hierarchical timing analysis of VLSI circuits”, Patent 8122404 B2, [http://www.google.com/patents/US8122404.](http://www.google.com/patents/US8122404)

* <http://www.vlsi-expert.com/2011/02/etm-extracted-timing-models-basics.html>

19

# Updates

20

**References**

1. W. C. Elmore, “The Transient Response of Damped Linear Networks with Particular Regard to Wide-band Amplifiers”, *Journal of Applied Physics*, 19(1)(1948), pp. 55-63.
2. 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.
3. 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.
4. P. Penfield Jr. and J. Rubinstein, “Signal Delay in RC Tree Networks”, *Proc. Design* *Automation Conference*, 1981, pp. 613-617.
5. 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.

21