

ISPD 2013 Discrete Gate Sizing Contest Details

Last Modified: November 19, 2012

Look for slides with yellow box in top right corner
(examples below) to see what was modified.

Changed Nov-19-2012

Added Nov-19-2012

Outline

- Contest Overview
- Submission Information
- Contest Evaluation
- Benchmark Files & Parsers
- Sizer/Timer Interaction

Contest Overview

- Discrete Gate Sizing Contest
 - Simultaneous gate sizing and cell type assignment to optimize power under performance constraints
 - A standard cell library will be provided to define discrete cell sizes and types.
- **We reserve the right to make changes in the contest rules or benchmarks in the future.** It is your responsibility to check the contest announcements on the web page until the submission deadline

Major Changes from 2012 Contest

- A new set of more realistic benchmarks and constraints will be released
- The interconnect model for each net (except clock) will be a realistic RC tree instead of one lumped capacitance number
 - We provide additional API calls for you to query more detailed timing information such as effective capacitance Modified
- The contestants can choose to write their own PrimeTime[®] script to use any PrimeTime[®] feature

Submission Information

System Specification

- **Operation system:** Linux, version is TBD
- **Memory limit:** TBD (likely to be at least 16GB)
- **Number of cores:** TBD (likely to be at least 4)
- **Programming Language:** You are free to use any, as long as we can run your binary on our platform for evaluation
 - C/C++ are officially supported, for others please contact us
 - MATLAB will not be available
- **Library:** standard C/C++ library
- **Parallel library:** Check with the contest organizers first
- **Public-domain third party library:** check with the contest organizers first to see whether it is allowed or not
- **Binary size limits:** Reasonable limits may apply especially if you are statically linking external non-standard libraries in your binary

Timing Engine

- You are allowed to implement your own timer, but the final timing evaluation will be done using Synopsys PrimeTime[®]*
- You also have the choice to use PrimeTime[®] as your timer during sizing
 - We will provide a blackbox API (application programming interface) to send sizes to PrimeTime[®] and receive back timing information.
 - You may choose to write your own PrimeTime[®] script to use any PrimeTime[®] feature
 - We will not debug your scripts

* Other names and brands may be claimed as the property of others.

Submission Requirements

- **Files to be submitted:**
static binaries
- The submitted work can be either single threaded or multi-threaded version
- An alpha (preliminary) binary submission is required approximately two weeks before the final submission deadline
- Optional PrimeTime[®] script file

Contest Evaluation

Contest Evaluation

- Two separate rankings:
 - Primary ranking: Solution quality will be the main metric. Runtime will be used for tie-breaking.
 - Secondary ranking: Both solution quality and runtime will be important. Multi-core implementations are encouraged!
- There will be a hard runtime limit for each benchmark

Quality Metrics

- Quality metrics in order of importance:
 1. Timing, slew, and max-load violations
 - Note: We expect each benchmark to have a solution with zero violations.*
 2. Cell leakage power
- In other words:
 - A solution with no violations will always be better than another with some violations
 - If there are two solutions with zero violations, then the one with the smaller total leakage will be ranked higher.
 - If there are two solutions both of which have non-zero violations, then the one with the smaller total violation will be ranked higher.
 - The exact weights for different violation types will be published later.
- *Definitions:*
 - Timing violation: A timing endpoint has negative slack
 - Slew violation: A pin has transition time larger than the max limit
 - Max-load violation: A cell drives load larger than the max capacitance limit specified in the library for that cell.

Contest Benchmarks & Parsers

Overview of Benchmark Files and Parsers

- Benchmark files
 - Input files
 - .v file
 - .spef file
 - .sdc file
 - .lib file
 - Intermediate files
 - .int.sizes
 - .timing file
 - .ceff file Added
 - Output file
 - .sizes file
- *C++ parser helpers will be provided.*

Describes netlist, parasitics and constraints for each test bench

Library file, shared by all the test benches

Updated cell sizes from sizer to timer

PrimeTime[®] output file to sizer

PrimeTime[®] output file to sizer

Cell sizes for the test bench

Benchmark Features

- The provided std cell library includes multiple discrete choices of cell sizes and types
- Delay/slew values are defined as look up tables in the library (.lib file)
 - Delay/slew function of input slew and output cap
 - Not necessarily a convex function
- Interconnect parasitics are given in .spef file and modeled as distributed RC trees (without cross-coupling capacitances)
- Timing model: please see next few slides

Sequential Timing Model

- All sequentials in the benchmarks will be rising edge triggered flip-flops
- Sequential sizing is not allowed, the library has only one size for sequential cells
- Ideal clock network is assumed
 - Clock port at the top level is directly connected to all sequential clock pins (i.e. no clock buffers)
 - Zero skew
 - Arrival time of clock at all sequentials is the same
 - Sequential delay is independent of clock slew
 - Sequential clock to output delay table in the library has the same delay for all clock input transition times
 - Clock input pin caps are zero for all sequential cells
 - Clock net has zero parasitics (net will not be mentioned in .spef file)
- Setup time is always 0. There are no hold constraints.

Timing Model

- Interconnect will be modeled as a distributed RC tree (one tree per net)
 - The RC tree will not have any cross-coupling capacitances
 - Delay through interconnect is supposed to be non-zero (except for clock nets which don't have any RC)
- Cell timing arc delay is a function of input transition time and output load cap
 - Output load cap = “effective” capacitance seen by the driver
- Cell timing arc output transition time is a function of input transition time and output load cap
- Delay and transition time functions are implemented as lookup tables
 - A simple 2-dimensional linear interpolation model is assumed
- We will assume worst slew propagation
 - Since we are doing max timing (setup only), this is the largest slew (see the slide titled ‘Max Delay and Slew Propagation’)

Notes on delay calculation

- Cell library models contain delay and transition time as functions of input transition time and output load capacitance
- Since most nets (except clock) will be distributed RC, they must be mapped to some “effective” capacitance to look up cell delays and output transition time from cell model
 - There are several techniques to do this (please see references on next page)
 - You are free to implement whatever you want if you choose to implement your own timing engine
 - We will provide you API to query effective capacitance from PrimeTime® Modified
 - All submissions will be evaluated using PrimeTime®
- Since most nets will be modeled as distributed RC, there will be non-zero delay through them
 - There are several techniques to compute this delay
 - You are free to implement whatever you want if you choose to implement your own timing engine
 - We provide you API to query arrival/transition time at all pins through PrimeTime® (which you can then use to get interconnect delay) Modified
 - All submissions will be evaluated using PrimeTime®

Delay Calculation References

(This list is not comprehensive, you are free to choose whatever you want)

1. Reducing arbitrary RC trees to driving point pi-model (useful step for several “effective” capacitance algorithms)
 - a. Peter R. O’Brien and Thomas L. Savarino, *Modeling the Driving-Point Characteristic of Resistive Interconnect for Accurate Delay Estimation*, ICCAD 1989, pages 512-519
2. Effective capacitance algorithms
 - a. Jessica Qian, Satyamurthy Pullela, and Lawrence Pillage, *Modeling the “Effective Capacitance” for the RC Interconnect of CMOS Gates*, IEEE TCAD, vol. 13, no. 12, pages 1526-1535, December 1994
 - b. Florentin Dartu, Noel Menezes, and Lawrence T. Pileggi, *Performance Computation for Precharacterized CMOS Gates with RC Loads*, IEEE TCAD, vol. 15, no. 5, pages 544-553, May 1996
3. Interconnect delay calculation (algorithms and metrics)
 - a. Lawrence T. Pillage and Ronald A. Rohrer, *Asymptotic Waveform Evaluation for Timing Analysis*, IEEE TCAD, vol. 9, no. 4, pages 352-366, April 1990.
 - b. Chandramouli V. Kashyap, Charles J. Alpert, and Anirudh Devgan, *An “Effective” Capacitance Based Delay Metric for RC Interconnect*, ICCAD 2009, pages 229-235
 - c. Charles J. Alpert, Anirudh Devgan, and Chandramouli Kashyap, *A Two Moment RC Delay Metric for Performance Optimization*, ISPD 2000, pages 69-74
 - d. Tao Lin, Emrah Acar, and Larry Pileggi, *h-gamma: an RC delay metric based on a gamma distribution approximation of the homogenous response*, ICCAD 1998, pages 19-25

2-D Interpolation for Library LUTs

- Let's suppose we want to evaluate table value for (x,y) where $x_2 < x < x_3$ and $y_1 < y < y_2$
- Then the relevant entries are f_{21} , f_{31} , f_{22} , and f_{32}
- Procedure:
 - Compute weights along each dimension
 - Perform a weighted sum

Lookup Table

	x_1	x_2	x_3	x_4
y_1	f_{11}	f_{21}	f_{31}	f_{41}
y_2	f_{12}	f_{22}	f_{32}	f_{42}
y_3	f_{13}	f_{23}	f_{33}	f_{43}

Weights:

$$w_x = (x - x_2) / (x_3 - x_2)$$

$$w_y = (y - y_1) / (y_2 - y_1)$$

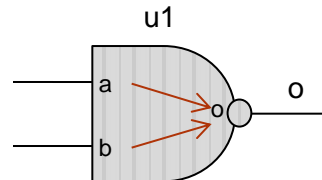
Value:

$$f(x,y) = (1-w_x)(1-w_y)f_{21} + w_x(1-w_y)f_{31} + (1-w_x)w_yf_{22} + w_xw_yf_{32}$$

Example if $x=x_2$ and $y=y_2$:

$$\rightarrow w_x=0, w_y=1, f(x,y) = 1*0*f_{21} + 0*0*f_{31} + 1*1*f_{22} + 0*1*f_{32} = f_{22}$$

Max Delay and Slew Propagation



- Let's suppose $a \uparrow$ to $o \downarrow$ arc has arrival time at $o \downarrow$ equal to 10ps and slew at $o \downarrow$ equal to 20ps
- Let's suppose $b \uparrow$ to $o \downarrow$ arc has arrival time at $o \downarrow$ equal to 12ps and slew at $o \downarrow$ equal to 18ps
- Therefore arrival time for $a \uparrow$ to $o \downarrow$ is better than that of $b \uparrow$ to $o \downarrow$ and slew of $a \uparrow$ to $o \downarrow$ is worse than that of $b \uparrow$ to $o \downarrow$
- For propagating an event to the next stage (i.e. inputs of cells driven by 'o') we need to select the worst case event
 - But which one is worse?
- For this contest, we will assume the worst for both:
 - We will use arrival time at $o \downarrow$ of 12ps ($b \uparrow$ to $o \downarrow$)
 - We will use slew at $o \downarrow$ of 20ps ($a \uparrow$ to $o \downarrow$)

C++ Parser Helpers

- The organizers are providing C++ helper functions and classes to extract the relevant data from the contest benchmarks.
- They are intended for parsing the contest benchmarks and contest output files (.timing/.ceff) only.
 - e.g. The provided Verilog parser will only work for the netlists of the ISPD13 benchmarks. It is not intended to parse general Verilog files.
- The contestants are free to use these parsers as is or make modifications. In any case, it is the contestants' responsibility to make sure that the parsers work as expected.
- ***The code provided here is mostly for description purposes. The organizers cannot guarantee that the provided code is free of bugs or defects.***

Verilog (.v) File

- Specifies cell instances and nets connecting them
- Simplified subset of the structured Verilog format
 - No hierarchy, no buses, no behavioral keywords
 - Single clock domain
 - Cell pins are only connected with wires
 - Inputs and outputs are implicitly connected to wires with the same name
 - No unconnected pins, no escape characters in names
 - No power or ground nets

simple.v File Example

```
module simple (  
  inp1,  
  inp2,  
  ispd_clk,  
  out  
);
```

verilog module name

interfaces
(inputs/outputs)

```
// Start PIs  
input inp1;  
input inp2;  
input ispd_clk;
```

inputs

```
// Start POs  
output out;
```

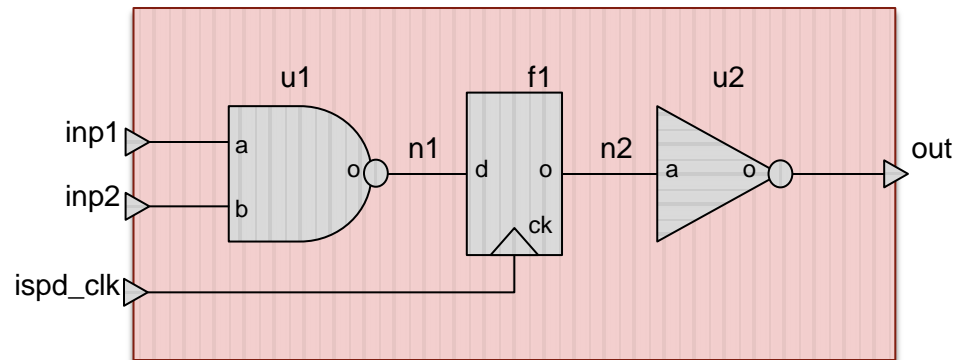
outputs

```
// Start wires  
wire n1;  
wire n2;  
wire inp1;  
wire inp2;  
wire ispd_clk;  
wire out;
```

intermediate
signals

```
// Start cells  
na02s01 u1 ( .a(inp1), .b(inp2), .o(n1) );  
ms00f80 f1 ( .d(n1), .ck(ispd_clk), .o(n2) );  
in01s01 u2 ( .a(n2), .o(out) );  
endmodule
```

end of module



cell instances and
connections between
cell pins and wires

Verilog Parser Helpers

- Provided C++ parsers:
 - `VerilogParser` class definition in `parser_helper.h`
 - See `test_verilog_parser()` function in `parser_helper.cpp` for an example on how to use this class.

Parasitics (.spef) File

- Specifies the interconnect parasitics of each net
 - We are using SPEF format as it is used by PrimeTime®
 - Even if you are not using PrimeTime®, the format is simple enough for you to parse
 - We will provide API to parse the SPEF file Modified
- File Format
 - Described on next page
- Recall: There will not be any cross coupling capacitances in the SPEF file and all nets (except clock nets) will have distributed RC trees. Clock nets will not have any parasitics and will not be mentioned in the SPEF file.

simple.spef File Format (by example)

Changed Nov-19-2012

Line #

```
1 *SPEF "IEEE 1481-1998"
2 *DESIGN "simple"
3 *DATE "Tue Sep 25 11:51:50 2012"
4 *VENDOR "ISPD 2013 Contest"
5 *PROGRAM "Benchmark Parasitic Generator"
6 *VERSION "0.0"
7 *DESIGN_FLOW "NETLIST TYPE VERILOG"
8 *DIVIDER /
9 *DELIMITER :
10 *BUS_DELIMITER []
11 *T_UNIT 1 PS
12 *C_UNIT 1 FF
13 *R_UNIT 1 KOHM
14 *L_UNIT 1 UH
```

Ignore lines 1-14

Note: All SPEF files will
have same units
C_UNIT = 1e-15 Farad
R_UNIT = 1000 Ohm

```
16 *D_NET inp1 5.4
17 *CONN
18 *P inp1 I
19 *I u1:a I
20 *CAP
21 1 inp1 1.2
22 2 inp1:1 1.3
23 3 inp1:2 1.4
24 4 u1:a 1.5
25 *RES
26 1 inp1 inp1:1 3.4
27 2 inp1:1 inp1:2 3.5
28 3 inp1:2 u1:a 3.6
29 *END
```

Changed direction from 'O' to 'I'

RC data for net
'inp1'

```
31 *D_NET inp2 2.0
32 *CONN
33 *P inp2 I
34 *I u1:b I
35 *CAP
36 1 inp2 0.2
37 2 inp2:1 0.5
38 3 inp2:2 0.4
39 4 u1:b 0.9
40 *RES
```

Changed direction from 'O' to 'I'

```
41 1 inp2 inp2:1 1.4
42 2 inp2:1 inp2:2 1.5
43 3 inp2:2 u1:b 1.6
44 *END
45
46 *D_NET out 0.7
47 *CONN
48 *I u2:o O
49 *P out O
50 *CAP
51 1 u2:o 0.2
52 2 out 0.5
53 *RES
54 1 u2:o out 1.4
55 *END
```

Changed direction from 'I' to 'O'

```
57 *D_NET n1 1.0
58 *CONN
59 *I u1:o O
60 *I f1:d I
61 *CAP
62 1 u1:o 0.2
63 1 n1:1 0.3
64 2 f1:d 0.5
65 *RES
66 1 u1:o n1:1 1.1
67 2 n1:1 f1:d 1.0
68 *END
```

RC data for net
'n1'

```
70 *D_NET n2 1.2
71 *CONN
72 *I f1:o O
73 *I u2:a I
74 *CAP
75 1 f1:o 0.7
76 2 u2:a 0.5
77 *RES
78 1 f1:o u2:a 2.1
79 *END
```

Explanation of RC data for net inp1

```

Line # 16 *D NET inp1 5.4
      17 *CONN
      18 *P inp1 I
      19 *I u1:a I
      20 *CAP
      21 1 inp1 1.2
      22 2 inp1:1 1.3
      23 3 inp1:2 1.4
      24 4 u1:a 1.5
      25 *RES
      26 1 inp1 inp1:1 3.4
      27 2 inp1:1 inp1:2 3.5
      28 3 inp1:2 u1:a 3.6
      29 *END
  
```

Net name and total capacitance: This means net 'inp1' has total lumped capacitance of $5.4 * C_UNIT = 5.4 * (1e-15) = 5.4e-15$ Farad

Changed direction from 'O' to 'I'

Connectivity section contains I/O pins of the net:

<pin-type> <name> <cell-pin-direction>
 <pin-type>: *P or *I
 <cell-pin-direction>: I or O

For net inp1:

- inp1 pin is of type port (*P) (i.e. port on the top module), and it is an input (I) of top module
- u1:a pin is of type internal (*I) (i.e. pin a of cell instance u1 inside the top module), and it is an input (I) of a cell (i.e. output of the net)

Capacitance section:

<cap-number> <node> <value>
 <cap-number>: 1,2,3, ...

- There is $1.2 * C_UNIT = 1.2e-15$ Farad capacitance from node inp1 to ground
- There is $1.3e-15$ Farad cap from node inp1:1 (internal node 1 of net inp1) to ground
- There is $1.5e-15$ Farad cap from pin a of cell u1 to ground (note that this is in addition to input pin cap coming from cell library)

Resistance section:

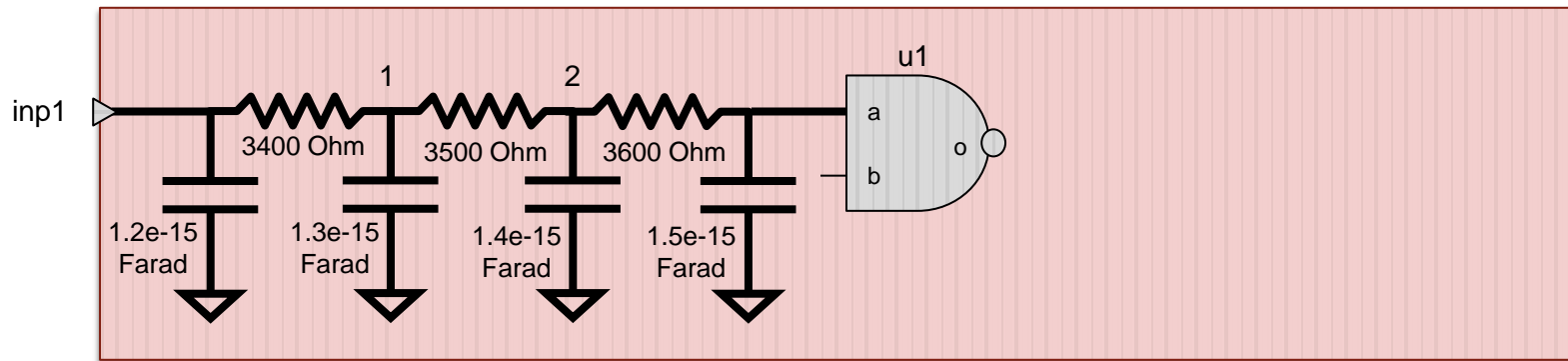
<res-number> <node1> <node2> <value>
 <res-number>: 1,2,3, ...

- There is $3.4 * R_UNIT = 3400$ Ohm resistance between node inp1 and node 1 of net inp1
- There is $3.6 * R_UNIT = 3600$ Ohm resistance between node 2 of net inp1 and pin a of cell instance u1

Notes on names of nets, nodes, pins in SPEF file

- All net names will match wire names from Verilog file
 - A net connected to primary input/output port on the top module will have the same name as the primary input/output port
- Nodes on net can be named three different ways
 1. `<port_name>` when the node is a port on the top module
 2. `<cell_name>:<pin_name>` when the node is a pin on an internal cell instance
 3. `<net_name>:<integer>` when the node is internal to the net

Visualization of RC data for net inp1



Spef Parser Helpers

- Provided C++ parsers:
 - `SpefParser` class definition in `parser_helper.h`
 - See `test_spef_parser()` function in `parser_helper.cpp` for an example on how to use this class.

Constraint (.sdc) File

- Describes timing and interface constraints for a block, including:
 - **clock period** and **clock input name**
 - **input delay*** – load-independent arrival time of the input relative to the rising edge of the clock
 - **output delay** – combinational delay from the output port to the next sequential (this sequential is external to the benchmark)
 - **driving cells for inputs*** – cell and pin names driving the input and transition time (slew rate) at the input of the driving cell (only one input cell will be used as driving cell)
 - **load capacitance for outputs**
- Timing and capacitance units are taken from the technology library (these are fixed as picosecond and femtoFarad)
- Max capacitance and max slew constraints are given in the library
- *see next 3 slides for details about input delay and driving cell

simple.sdc File Example

clock definition

```
create_clock -name mclk -period 50.0 [get_ports ispd_clk]
```

clock label (mclk), clock period (50.0) and port name (ispd_clk)

input delays

```
set_input_delay 0.0 [get_ports {inp1}] -clock mclk
```

```
set_input_delay 0.0 [get_ports {inp2}] -clock mclk
```

input delays w.r.t. clock (referred by label)

input drivers

```
set_driving_cell -lib_cell in01f80 -pin o [get_ports {inp1}] -input_transition_fall 80.0 -input_transition_rise 80.0
```

```
set_driving_cell -lib_cell in01f80 -pin o [get_ports {inp2}] -input_transition_fall 80.0 -input_transition_rise 80.0
```

driving cell/pin name and transition time at the cell input

output delays

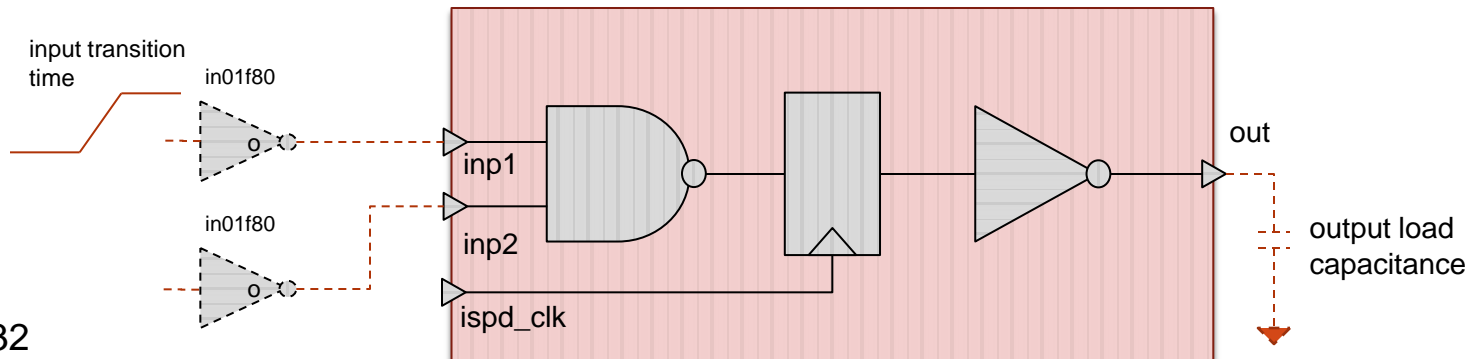
```
set_output_delay 0.0 [get_ports {out}] -clock mclk
```

output delay w.r.t. clock (referred by label)

output loads

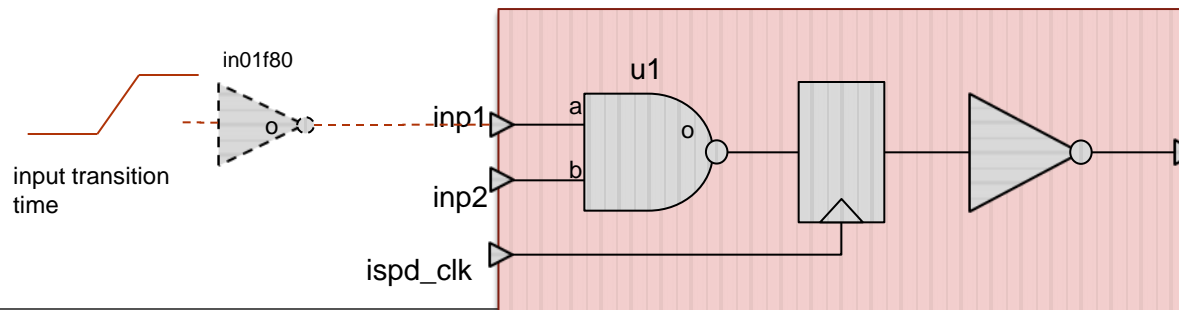
```
set_load -pin_load 4.0 [get_ports {out}]
```

output load capacitance



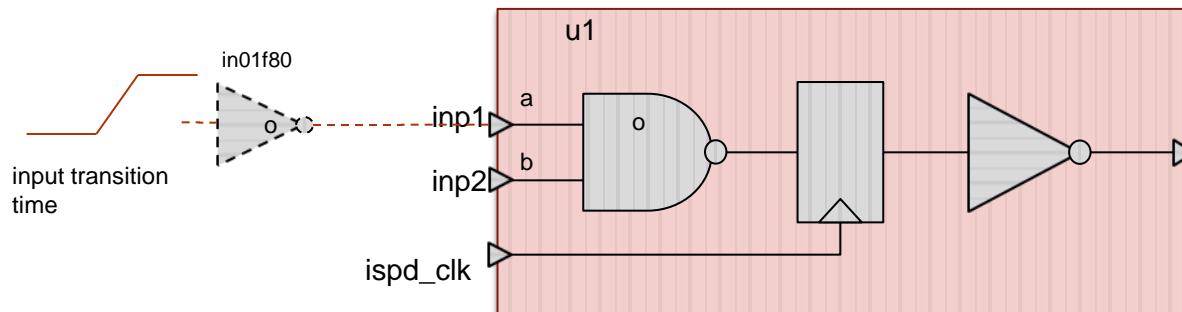
Constraint file (.sdc): Input delay and driving cell

- Let us consider arrival time calculation at input port `inp1`, for which the following is given in sdc-file:
 - `set_input_delay 0.0 [get_ports {inp1}] -clock mclk`
 - `set_driving_cell -lib_cell in01f80 -pin o [get_ports {inp1}] -input_transition_fall 80.0 -input_transition_rise 80.0`
- The real arrival time at port `inp1` ($ArrTime_{inp1}$) is computed as the sum of:
 - Load independent arrival time: Specified by `set_input_delay` command in sdc
 - Incremental delay due to capacitive load: Specified by `set_driving_cell` in sdc
- Detailed formulas:
 - $ArrTime_{inp1} = Load_Independent_ArrTime_{inp1} + Incremental_Driver_Delay_{inp1}$
 - $Load_Independent_ArrTime_{inp1}$: The value specified by `set_input_delay` in sdc
 - $Incremental_Driver_Delay_{inp1} = Delay_{in01f80}(Load_{inp1}, DriverInputTrans) - Delay_{in01f80}(0.0, DriverInputTrans)$
 - $Delay_{in01f80}(L, T)$: Delay of cell `in01f80` for output load L and input transition T (based on delay LUT of `in01f80`)
 - $Load_{inp1}$: Load at `inp1` (the effective capacitance for net `inp1`, calculated from RC data from .spef file and input capacitance of pin `a` of instance `u1`)
 - $DriverInputTrans$: The input transition specified in the `set_driving_cell` command in sdc
- In this example:
 - $ArrTime_{inp1} = 0.0 + Delay_{in01f80}(Load_{inp1}, 80.0) - Delay_{in01f80}(0.0, 80.0)$



Constraint file (.sdc): Input slew and driving cell

- Transition times (corresponding to slew rates) at the input ports are load dependent, and should be computed based on the driving cell specifications in sdc:
 - `set_driving_cell -lib_cell in01f80 -pin o [get_ports {inp1}] -input_transition_fall 80.0 -input_transition_rise 80.0`
- Detailed formulas:
 - $\text{TransitionTime}_{\text{inp1}} = \text{TT}_{\text{in01f80}}(\text{Load}_{\text{inp1}}, \text{DriverInputTrans})$
 - $\text{TT}_{\text{in01f80}}(L, T)$: The transition time at the output pin of *in01f80* for output load *L* and input transition *T* (based on the slew LUT of *in01f80*)
 - $\text{Load}_{\text{inp1}}$: Load at *inp1* (the effective capacitance for net *inp1*, calculated from RC data from .spef file and input capacitance of pin *a* of instance *u1*)
 - DriverInputTrans : The input transition specified in the *set_driving_cell* command in sdc
- In this example:
 - $\text{TransitionTime}_{\text{inp1}} = \text{TT}_{\text{in01f80}}(\text{Load}_{\text{inp1}}, 80.0)$



Sdc Parser Helpers

- Provided C++ parsers:
 - `SdcParser` class definition in `parser_helper.h`
 - See `test_sdc_parser()` function in `parser_helper.cpp` for an example on how to use this class.

.lib File

- We use a single .lib file to describe the timing and (leakage) power properties of all the standard cells in the library.
- Much of the information is boilerplate needed for Synopsys PrimeTime[®] to perform its timing analysis.
- The .lib file consists of a header (which for sizing purposes can be ignored), a specification of each cell, and then a footer (which can also be ignored.)

contest.lib File Example (header)

```
/*
 * Library File Format
 * Copyright © 2011, Synopsys, Inc. and others. All Rights reserved.
 */
library ("contest") {
  revision : 3.0.0 ;
  delay_model : table_lookup ;
  comment : "ISPD Contest Mock Library" ;
  library_features(report_delay_calculation, report_power_calculation);
  time_unit : "1ps" ;
  voltage_unit : "1V" ;
  current_unit : "1mA" ;
  leakage_power_unit : 1uW ;
  capacitive_load_unit(1,ff);
  pulling_resistance_unit : "1kohm" ;
  default_fanout_load : 1.0 ;
  default_inout_pin_cap : 0.0 ;
  default_input_pin_cap : 0.0 ;
  default_output_pin_cap : 0.0 ;
  slew_lower_threshold_pct_rise : 20.0 ;
  slew_lower_threshold_pct_fall : 20.0 ;
  slew_upper_threshold_pct_rise : 80.0 ;
  slew_upper_threshold_pct_fall : 80.0 ;
  input_threshold_pct_rise : 50.0 ;
  input_threshold_pct_fall : 50.0 ;
  output_threshold_pct_rise : 50.0 ;
  output_threshold_pct_fall : 50.0 ;
  nom_voltage : 0.7 ;
  nom_temperature : 70.0 ;
  nom_process : 1.0 ;
  in_place_swap_mode : match_footprint ;
  slew_derate_from_library : 1 ;

  default_cell_leakage_power : 0.0 ;
  default_leakage_power_density : 0.0 ;

  operating_conditions("typical_1.00") {
    process : 1.00 ;
    temperature : 70.0 ;
    voltage : 0.7 ;
    tree_type : "balanced_tree" ;
  }
  default_operating_conditions : "typical_1.00" ;
}
```

```
wire_load(10X10) {
  resistance : 0.00 ;
  capacitance : 0.00 ;
  area : 0.00 ;
  slope : 0.00 ;
  fanout_length(1, 1.0000);
  fanout_length(2, 1.0000);
  fanout_length(3, 1.0000);
  fanout_length(4, 1.0000);
  fanout_length(5, 1.0000);
  fanout_length(6, 1.0000);
  fanout_length(7, 1.0000);
  fanout_length(8, 1.0000);
  fanout_length(9, 1.0000);
}
default_wire_load_mode : enclosed ;
default_max_transition : 300.00 ;

scaling_factors("synop_lib_average_factors") {
  k_process_cell_rise : 1.0000 ; /* default */
  k_process_cell_fall : 1.0000 ; /* default */
  k_process_rise_transition : 1.0000 ; /* default */
  k_process_fall_transition : 1.0000 ; /* default */
  k_process_setup_rise : 0.0000 ; /* default */
  k_process_setup_fall : 0.0000 ; /* default */
  k_process_hold_rise : 0.0000 ; /* default */
  k_process_hold_fall : 0.0000 ; /* default */
  k_process_recovery_rise : 0.0000 ; /* default */
  k_process_recovery_fall : 0.0000 ; /* default */
  k_process_removal_rise : 0.0000 ; /* default */
  k_process_removal_fall : 0.0000 ; /* default */
  k_process_min_pulse_width_low : 0.0000 ; /* default */
  k_process_min_pulse_width_high : 0.0000 ; /* default */
}
```

```
lu_table_template (delay_outputslew_template_7X8) {
  variable_1 : total_output_net_capacitance ;
  variable_2 : input_net_transition ;
  index_1 ("1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6");
  index_2 ("2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7");
}
```

```
lu_table_template (relational_table_template_2X2X2) {
  variable_1 : related_pin_transition ;
  variable_2 : constrained_pin_transition ;
  variable_3 : related_out_total_output_net_capacitance ;
  index_1 ("1.0, 1.1");
  index_2 ("2.0, 2.1");
  index_3 ("3.0, 3.1");
}
```

The green fields might be important: the slope (transition time) limit, and the order of the variables for the timing tables.

contest.lib File Example (sizeable cell)

Cell name

```
/* Begin cell: na02s01 */
```

Leakage value for cell

Swapping group

```
cell ("na02s01") {
  cell_leakage_power : 2.00;
```

Output cap (always 0.0; contribution to delay included in delay tables)

Cap limit

```
  area : 2.00 ;
  cell_footprint : na02 ;
```

```
  pin ("o") {
    direction : output ;
    capacitance : 0.0 ;
    function : "!a+!b" ;
    max_capacitance : 6.40 ;
```

```
  min_capacitance : 0.00 ;
  timing() {
    cell_fall ("delay_outputslew_template_7X8") {
      index_1 ("0.00,0.40,0.80,1.60,3.20,6.40,12.80") ;
      index_2 ("5.00,30.00,50.00,80.00,140.00,200.00,300.00,500.00") ;
      values (\
        "51.00, 56.00, 60.00, 66.00, 78.00, 90.00, 110.00, 150.00",\
        "55.00, 60.00, 64.00, 70.00, 82.00, 94.00, 114.00, 154.00",\
        "59.00, 64.00, 68.00, 74.00, 86.00, 98.00, 118.00, 158.00",\
        "67.00, 72.00, 76.00, 82.00, 94.00, 106.00, 126.00, 166.00",\
        "83.00, 88.00, 92.00, 98.00, 110.00, 122.00, 142.00, 182.00",\
        "115.00, 120.00, 124.00, 130.00, 142.00, 154.00, 174.00, 214.00",\
        "179.00, 184.00, 188.00, 194.00, 206.00, 218.00, 238.00, 278.00"\
      );
    }
```

The rows (index_1) correspond to different caps; the columns (index_2) to different slopes

```
    cell_rise ("delay_outputslew_template_7X8") {
      <table syntax removed>
    }
    fall_transition ("delay_outputslew_template_7X8") {
      <table syntax removed>
    }
    rise_transition("delay_outputslew_template_7X8") {
      <table syntax removed>
    }
    timing_sense : negative_unate ;
    related_pin : "a" ;
  }
```

Timing arcs from pin "a" to pin "o"

```
/* End timing */
```

The library contains only inverting gates

```
  timing() {
    cell_fall ("delay_outputslew_template_7X8") {
      <table syntax removed>
    }
    cell_rise ("delay_outputslew_template_7X8") {
      <table syntax removed>
    }
    fall_transition ("delay_outputslew_template_7X8") {
      <table syntax removed>
    }
    rise_transition ("delay_outputslew_template_7X8") {
      <table syntax removed>
    }
    timing_sense : negative_unate ;
    related_pin : "b" ;
  }
/* End timing */
}
/* End pin */
pin ("a") {
  capacitance : 1.00 ;
  direction : input ;
}
/* End pin */
pin ("b") {
  capacitance : 1.00 ;
  direction : input ;
}
/* End pin */
}
/* End cell: na02s01 */
```

Input capacitance

contest.lib File Example (fixed sequential)

```

/* Begin cell: ms00f80 */
cell ("ms00f80") {
  ff (iq,iqn) {
    next_state : "d" ;
    clocked_on : "ck" ;
  }
  cell_leakage_power : 0.0;
  area : 0.0 ;
  cell_footprint : ms00 ;
  pin ("o") {
    direction : output ;
    capacitance : 0.0 ;
    function : "iq" ;
    max_capacitance : 2048.00 ;
    min_capacitance : 0.00 ;
    timing() {
      cell_fall ("delay_outputslew_template_7X8") {
        index_1 ("0.00,128.00,256.00,512.00,1024.00,2048.00,4096.00") ;
        index_2 ("5.00,30.00,50.00,80.00,140.00,200.00,300.00,500.00") ;
        values (\
          "16.00, 16.00, 16.00, 16.00, 16.00, 16.00, 16.00, 16.00",\
          "21.00, 21.00, 21.00, 21.00, 21.00, 21.00, 21.00, 21.00",\
          "26.00, 26.00, 26.00, 26.00, 26.00, 26.00, 26.00, 26.00",\
          "36.00, 36.00, 36.00, 36.00, 36.00, 36.00, 36.00, 36.00",\
          "56.00, 56.00, 56.00, 56.00, 56.00, 56.00, 56.00, 56.00",\
          "96.00, 96.00, 96.00, 96.00, 96.00, 96.00, 96.00, 96.00",\
          "176.00, 176.00, 176.00, 176.00, 176.00, 176.00, 176.00, 176.00"\
        );
      }
      cell_rise ("delay_outputslew_template_7X8") {
<table syntax removed>
      }
      fall_transition ("delay_outputslew_template_7X8") {
<table syntax removed>
      }
      rise_transition ("delay_outputslew_template_7X8") {
<table syntax removed>
      }
      timing_sense : non_unate ;
      timing_type : rising_edge ;
      related_pin : "ck" ;
    }
  }
}
/* End timing */
}
/* End pin */

```

Cell name

Swapping group (only one cell in group)

Cap limit

. Clock to output delay and slope are independent of clock slope.

Timing arcs from pin "ck" to pin "o"

Output cap (always 0.0; contribution to delay included in delay tables)

Appropriate annotations for a rising-edge triggered flip flop

```

pin ("ck") {
  clock : true ;
  capacitance : 0.0 ;
  direction : input ;
}
/* End pin */
pin ("d") {
  capacitance : 1.00 ;
  nextstate_type : data ;
  direction : input ;
  timing() {
    timing_type : setup_rising ;
    related_pin : ck ;
    related_output_pin : o ;
    rise_constraint ("relational_table_template_2X2X2") {
      index_1 ("5.00,500.00") ;
      index_2 ("5.00,500.00") ;
      index_3 ("5.00,500.00") ;
      values (\
        "0.00, 0.00",\
        "0.00, 0.00",\
        "0.00, 0.00",\
        "0.00, 0.00"\
      );
    }
    fall_constraint ("relational_table_template_2X2X2") {
<table syntax removed>
    }
  }
}
/* End timing */
timing() {
  timing_type : hold_rising ;
  related_pin : ck ;
  related_output_pin : o ;
  rise_constraint ("relational_table_template_2X2X2") {
<table syntax removed>
  }
  fall_constraint ("relational_table_template_2X2X2") {
<table syntax removed>
  }
}
/* End timing */
}
/* End pin */
}
/* End cell: ms00f80 */

```

Clock cap is always 0.

Data pin cap

Setup time is always 0 (complex 3D tables can be ignored.)

The hold time is also zero.

contest.lib File Example (footer)

```
/* Begin cell: vcc */
/* Description : vcc */
cell ("vcc") {
  dont_use : true ;
  dont_touch : true ;
  area : 0.00 ;
  cell_footprint : vcc ;
  pin(y) {
    direction : output ;
    function : " 1 " ;
    max_capacitance : 100000000.000 ;
  }
}
/* End pin */
}
/* End cell: vcc */
/* Begin cell: vss */
/* Description : vss */
cell ("vss") {
  dont_use : true ;
  dont_touch : true ;
  area : 0.00 ;
  cell_footprint : vss ;
  pin(y) {
    direction : output ;
    function : " 0 " ;
    max_capacitance : 100000000.000 ;
  }
}
/* End pin */
}
/* End cell: vss */
}
```

Define cells to produce constant values. Should be able to ignore these.

Lib Parser Helpers

- Provided C++ parsers:
 - `LibParser` class definition in `parser_helper.h`
 - See `test_lib_parser()` function in `parser_helper.cpp` for an example on how to use this class.
- Additional class definitions in `parser_helper.h` to store the extracted data:
 - `LibParserLUT`: Delay or slew look up table
 - `LibParserTimingInfo`: Timing arc data
 - `LibParserPinInfo`: Pin data
 - `LibParserCellInfo`: Cell data

Sizer Output (.sizes) File

- Optimization engine output file that includes the size/type for each cell
 - File format:
 - Must contain n lines where n is the number of cell instances in the benchmark, one line per instance
 - Each line must have two strings that are separated by a space in the middle
- <full-instance-name> <library-cell-name>**
- Each line must end with end-of-line (`\n`) character
 - This includes the last line in the file
 - There must not be any empty lines in the file
 - Example output for the given Verilog example simple.v:

Line #

```
1 u1 na02s20
2 f1 ms00f80
3 u2 in01s20
4
```

PrimeTime[®] output (.timing file)

- This applies only if you are using our blackbox API (see Option 2 in Sizer/Timer Interaction section) to call PrimeTime[®]
- Contains timing information (slacks, transition times, and arrival times) for pins (inputs/outputs of internal cells) and ports (inputs/outputs of top module)
- There is no particular order in which the pins/ports will be listed
- Timing information will only be printed for requested pins/ports (see Option 2 in Sizer/Timer interaction section)

- Each line will have this format:

```
<pin or port name> <riseSlack> <fallSlack> <riseTT> <fallTT> <riseAT> <fallAT>
```

```
<pin or port name>: <cell-name>/<pin-name> for inputs/outputs of internal cells
```

```
                <port-name> for inputs/outputs of top module
```

```
TT: transition time, AT: arrival time
```

- You may use the provided C++ parser to parse the .timing file (or you can write your own parser)

PrimeTime[®] output (.ceff file)

- This applies only if you are using our blackbox API (see Option 2 in Sizer/Timer Interaction section) to call PrimeTime[®]
- Contains effective capacitance for pins (outputs of internal cells) and ports (inputs of top module)
- There is no particular order in which the pins/ports will be listed
- Effective capacitance will only be printed for requested pins/ports (see Option 2 in Sizer/Timer interaction section)

- Each line will have this format:

```
<pin or port name> <riseCeff> <fallCeff>
```

```
<pin or port name>: <cell-name>/<pin-name> for inputs/outputs of internal cells  
                  <port-name> for inputs/outputs of top module
```

- You may use the provided C++ parser to parse the .ceff file (or you can write your own parser)

.timing/.ceff Parser Helpers

- Provided C++ parsers:
 - **TimingParser** class definition in `parser_helper.h`
 - See `test_timing_parser()` function in `parser_helper.cpp` for an example on how to use this class.
 - **CeffParser** class definition in `parser_helper.h`
 - See `test_ceff_parser()` function in `parser_helper.cpp` for an example on how to use this class.

Sizer Executable

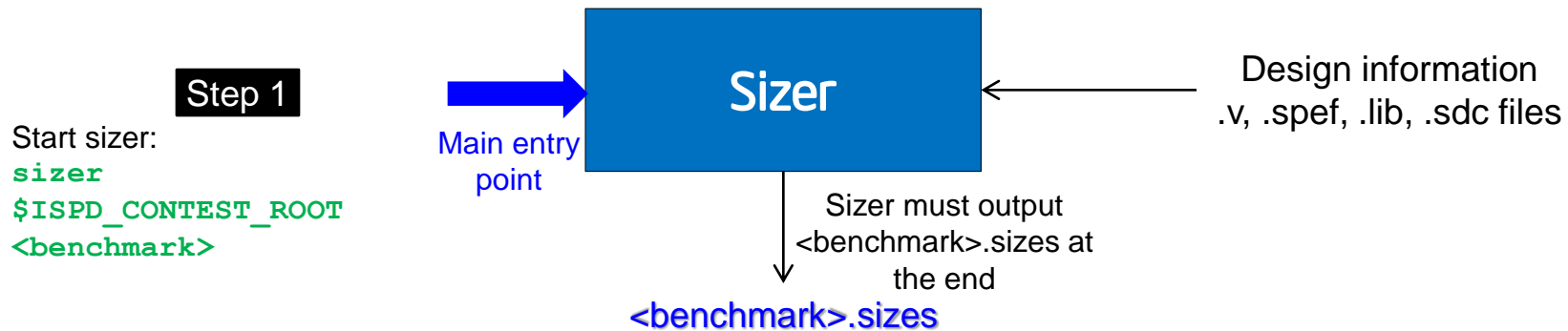
- Must be called sizer
- Must take exactly two arguments as input, the contest root directory and benchmark name
- We will invoke the sizer by calling this:
`sizer <ispd_contest_root> <benchmark>`
- Example:
`sizer ./ simple`
- Before calling the timer, you must write out .int.sizes file (same format as .sizes file) containing the cell sizes of all cell instances (timer reads this file)
 - simple/simple.int.sizes for our example
- You must output the final sizes for all cell instances in .sizes file
 - simple/simple.sizes for our example

Sizer/Timer Interaction

Sizer/Timer Interaction

- If you chose to implement your own timer, go to Option 1 (see following slides)
- If you chose to use Synopsys PrimeTime[®] as your timer using the API we provide you, go to Option 2 (see following slides)
- If you choose to use Synopsys PrimeTime[®] as your timer using your own scripts and API, go to Option 3 (see following slides)

Option 1: Sizer/Timer Flow if you are using your own timer



Notes:

1. Your sizer must look in the directory `$ISPD_CONTEST_ROOT/<benchmark>`, simple in our case, for design information (simple.v, simple.spef, and simple.sdc)
2. Your sizer must look in the `$ISPD_CONTEST_ROOT/lib` directory for cell library file contest.lib
3. Your sizer must output `$ISPD_CONTEST_ROOT/<benchmark>/<benchmark>.sizes`, simple.sizes in our example

Using Option 1

- Directory organization
 1. Make a directory (let's call it 'ispd2013'). You can pick whatever name you like. `setenv ISPD_CONTEST_ROOT ispd2013`
 2. Save your sizer under the same directory
 3. Save all design benchmark files (.v, spof, .sdc) provided by organizers in a directory whose name is the benchmark name (e.g., the "simple" benchmark in the earlier slide). Save this directory under `$ISPD_CONTEST_ROOT`.
 4. Save library file `contest.lib` provided by organizers under the `lib` directory under `$ISPD_CONTEST_ROOT`.
 5. `cd` to `$ISPD_CONTEST_ROOT`.
- Running the benchmark
 1. Call the sizer:
`sizer $ISPD_CONTEST_ROOT simple`
- At the end of final sizing
 1. Write out `$ISPD_CONTEST_ROOT/simple/simple.sizes` file for the benchmark

Option1: Directory hierarchy example for 'simple' benchmark

- ispd2013/sizer
- ispd2013/lib/contest.lib
- ispd2013/simple/simple.v
- ispd2013/simple/simple.spef
- ispd2013/simple/simple.sdc
- ispd2013/simple/simple.sizes (to be written by sizer)

Example Invocation:

```
sizer $ISPD_CONTEST_ROOT simple
```

Option 2: Sizer/Timer Flow if you are using PrimeTime[®] for timing using API we provide you

- We will provide all the utility scripts and helper functions to make this easy for you
- You may call PrimeTime[®] directly from your code, but then it is your responsibility to make sure it works properly.
- At the high level, we will provide you a C++ API to interact with PrimeTime[®] using file I/O
 - You must compile this C++ file into your final static binary
 - You are free to use the C++ API as is or make modifications
 - If you make modifications, you must ensure proper interaction
- The high level C++ API will interact with a timer process (running in parallel) using file I/O
 - For contest evaluation, the process will be invoked using the same TCL files we provide you
 - You must not modify any TCL files we provide you. You will not be submitting TCL files (we will use our versions, which we have already provided to you).

Option 2: Flow diagram

Sizer

Step 2

Start sizer and pass it benchmark name:

```
sizer  
$ISPD_CONTEST_ROOT  
simple
```

Design information
.v, .spef, .lib, .sdc files



Sizer must output
<benchmark>.sizes at
the end

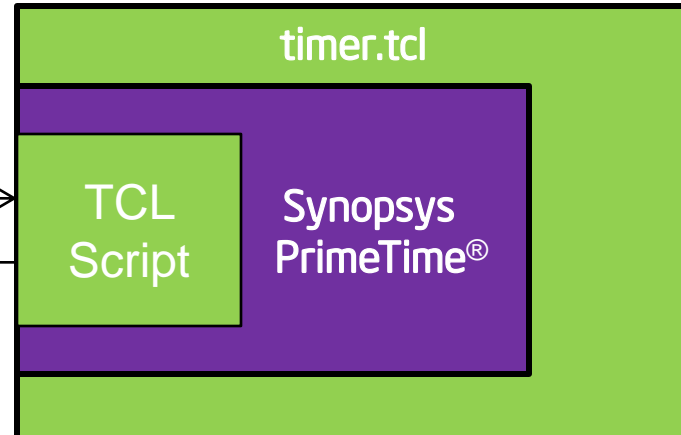
<benchmark>.sizes

Timer

Step 1


Start timer and pass it benchmark name and path to PrimeTime® executable:

```
tcl timer.tcl pt_shell
```



Note: You are not allowed to make any changes on this side

 Provided by contestants

 Provided by organizers

Option 2: Flow diagram (details)

Sizer

Timer

Step 2

Start sizer and pass it benchmark name:

```
sizer
$ISPD_CONTEST_ROOT
simple
```

Design information
.v, .spef, .lib, .sdc files



Sizer must output
↓ <benchmark>.sizes at the end

<benchmark>.sizes

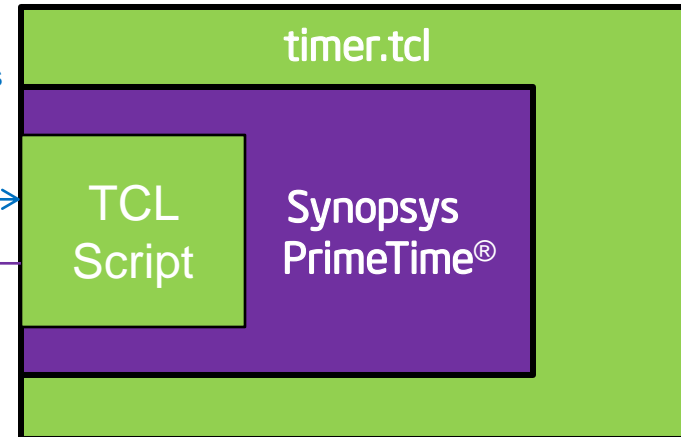
updated cell sizes
(.int.sizes file), ports/pins
for timing (.timing_pins)
and ceff (.ceff_pins)
requests

Timing (.timing) and
Ceff (.ceff) information

Step 1

Start timer and pass it benchmark name and path to PrimeTime® executable:

```
tcl timer.tcl pt_shell
```



Note: You are not allowed to make any changes on this side

Notes:

1. Your sizer must look in the directory \$ISPD_CONTEST_ROOT/<benchmark>, simple in our case, for design information (simple.v, simple.spef, and simple.sdc)
2. Your sizer must look in the \$ISPD_CONTEST_ROOT/lib directory for cell library file contest.lib
3. Your sizer must output \$ISPD_CONTEST_ROOT/<benchmark>/<benchmark>.int.sizes, simple/simple.int.sizes in our example, before each call to run timing through C++ API and write out .timing_pins and .ceff_pins files to request timing/ceff data from timer
4. At the end of each timing run, timer will output \$ISPD_CONTEST_ROOT/<benchmark>/<benchmark>.timing, simple/simple.timing in our example and \$ISPD_CONTEST_ROOT/<benchmark>/<benchmark>.ceff, simple/simple.ceff in our example
5. At the end of your sizer execution, you must output the final sizes in \$ISPD_CONTEST_ROOT/<benchmark>/<benchmark>.sizes file

Procedure for Timer/Sizer Interaction

- After timer.tcl and sizer are both invoked,
 - Step 1. timer.tcl will run and start Synopsys PrimeTime[®], which will keep running
 - Step 2. The provided TCL script (running inside PrimeTime[®] TCL shell) will monitor if the file `__TCMD_RUNTIMER_` exists in `<benchmark>` directory
 - Step 3. If `__TCMD_RUNTIMER_` exists, the TCL script will read cell sizes from `$ISPD_CONTEST_ROOT/<benchmark>/<benchmark>.int.sizes` and run timing analysis with the new sizes
 - Step 4. At the end of the timing run inside PrimeTime[®]
 - TCL script will read `$ISPD_CONTEST_ROOT/<benchmark>/<benchmark>.timing_pins` and `$ISPD_CONTEST_ROOT/<benchmark>/<benchmark>.ceff_pins` files (both written by sizer) to identify where timing/ceff information is requested by the sizer
 - a file named `__SIZERCMD_TIMERDONE_`, the timing information in the `$ISPD_CONTEST_ROOT/<benchmark>/<benchmark>.timing` file (e.g., `simple/simple.timing`), and the effective capacitance in `$ISPD_CONTEST_ROOT/<benchmark>/<benchmark>.timing` file (e.g., `simple/simple.ceff`) will be written out.
 - At any moment during the entire above process, if there is an error, an empty file called `__SIZERCMD_TIMERERROR_` will be written out in `$ISPD_CONTEST_ROOT/<benchmark>` directory
 - At all moments, timer.tcl will keep monitoring `__TCMD_SHUTDOWN_` in the benchmark directory. If the file exists then the timer process exits.
- **For easy interaction, use high level functions in the C++ API that we have provided (timer_interface.h)**

Format of .timing_pins and .ceff_pins files to be written by sizer

- These files are to be used by sizer to request timing/ceff information for specific ports/pins from PrimeTime®
 - We have done this to give you freedom to request timing and effective capacitance at some or all ports/pins
- Sometimes, PrimeTime® may not report timing/ceff at the port/pin you have requested. In this case, no information will be printed for that port/pin in the output .timing/.ceff files.
 - Example: effective capacitance only exists for outputs of cells (top module inputs are driven by cell mentioned in .sdc file so effective capacitance exists for top module input ports also)
- Format of .timing_pins and .ceff_pins files:
 - Each line must contain the port/pin name for which you want the information:

```
<pin or port name 1>
```

```
<pin or port name 2>
```

```
...
```

where

```
<pin or port name>: <cell-name>/<pin-name> for inputs/outputs of  
internal cells
```

```
<port-name> for inputs/outputs of top module
```


Using Option 2

- Directory organization
 1. Make a directory (let's call it 'ispd2013'). You can pick whatever name you like.
setenv ISPD_CONTEST_ROOT ispd2013
 2. Save all .tcl files (timing.tcl and pt_scripts.tcl) provided by organizers under the directory (\$ISPD_CONTEST_ROOT) that you created above
 3. Save your sizer under the same directory
 4. Save all design benchmark files (.v, .spef, .sdc) provided by organizers in a directory whose name is the benchmark name (e.g., the "simple" benchmark in the earlier slide). Save this directory under \$ISPD_CONTEST_ROOT.
 5. Save library file (contest.lib) provided by organizers under \$ISPD_CONTEST_ROOT/lib.
 6. cd to \$ISPD_CONTEST_ROOT.
- Running the benchmark
 1. Alias path to your TCL interpreter
Example:
`alias tcl /usr/bin/tcl`
 2. Call the timer by passing it the benchmark name and the path to PrimeTime® executable and the benchmark name
Example:
`tcl timer.tcl /usr/install/primetime/pt_shell &`
 3. PrimeTime® log will be saved in the current directory (pt.log file)

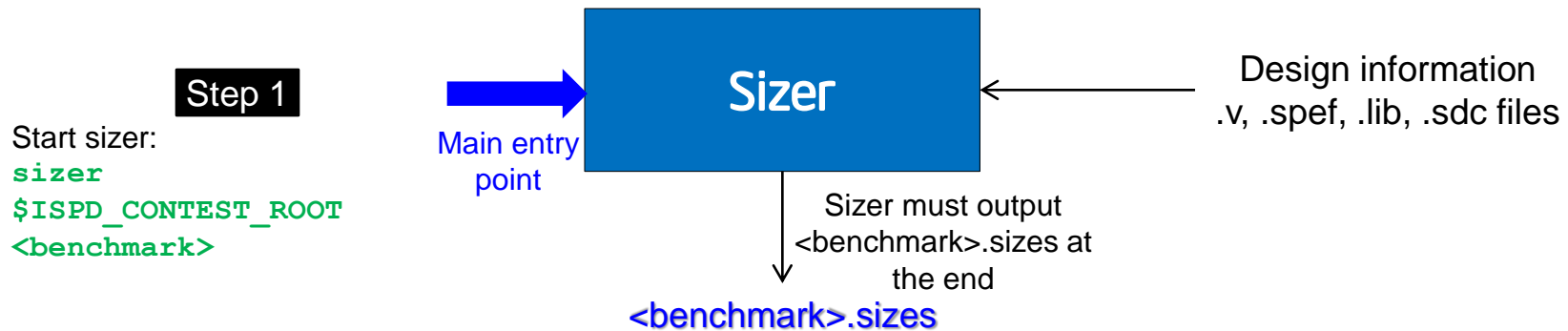
Option 2: Directory hierarchy example for 'simple' benchmark if using PrimeTime[®]

- ispd2013/sizer
- ispd2013/pt_scripts.tcl
- ispd2013/timer.tcl
- ispd2013/lib/contest.lib
- ispd2013/simple/simple.v
- ispd2013/simple/simple.spef
- ispd2013/simple/simple.sdc
- ispd2013/simple/simple.timing (written by PrimeTime[®])
- ispd2013/simple/simple.ceff (written by PrimeTime[®]) Added
- ispd2013/simple/simple.int.sizes (to be written by sizer)
- ispd2013/simple/simple.timing_pins (to be written by sizer) Added
- ispd2013/simple/simple.ceff_pins (to be written by sizer) Added
- ispd2013/simple/simple.sizes (to be written finally by sizer, will be used for evaluation) Added

Example Invocation:

```
alias tcl /usr/bin/tcl
cd ispd2013
tcl timer.tcl <full-path-to-pt_shell>
```

Option 3: Sizer/Timer Flow if you want to use your own scripts/PrimeTime[®] calls



Notes:

1. Your sizer must look in the directory `$ISPD_CONTEST_ROOT/<benchmark>`, simple in our case, for design information (simple.v, simple.spef, and simple.sdc)
2. Your sizer must look in the `$ISPD_CONTEST_ROOT/lib` directory for cell library file contest.lib
3. Your sizer must output `$ISPD_CONTEST_ROOT/<benchmark>/<benchmark>.sizes`, simple.sizes in our example
4. Use `$PRIMETIME_EXECUTABLE` environment variable to access PrimeTime[®] executable from within your binary

Using Option 3

- Directory organization
 1. Make a directory (let's call it 'ispd2013'). You can pick whatever name you like. `setenv ISPD_CONTEST_ROOT ispd2013`
 2. Save your sizer under the same directory
 3. Save all design benchmark files (.v, spof, .sdc) provided by organizers in a directory whose name is the benchmark name (e.g., the "simple" benchmark in the earlier slide). Save this directory under `$ISPD_CONTEST_ROOT`.
 4. Save library file `contest.lib` provided by organizers under the `lib` directory under `$ISPD_CONTEST_ROOT`.
 5. `cd` to `$ISPD_CONTEST_ROOT`.
- Running the benchmark
 1. Call the sizer:
`sizer $ISPD_CONTEST_ROOT simple`
- At the end of final sizing
 1. Write out `$ISPD_CONTEST_ROOT/simple/simple.sizes` file for the benchmark

Option3: Directory hierarchy example for 'simple' benchmark

- ispd2013/sizer
- ispd2013/lib/contest.lib
- ispd2013/simple/simple.v
- ispd2013/simple/simple.spef
- ispd2013/simple/simple.sdc
- ispd2013/simple/simple.sizes (to be written by sizer)

Example Invocation:

```
sizer $ISPD_CONTEST_ROOT simple
```

Contest Organizers

Chirayu Amin (Timing)

chirayu.s.amin _at_ intel com

Andrey Ayupov (Benchmarks)

andrey.ayupov _at_ intel com

Steven Burns (Cell Library)

steven.m.burns _at_ intel com

Mustafa Ozdal (Contest chair)

mustafa.ozdal _at_ intel com

Gustavo Wilke (Evaluations)

gustavo.r.wilke _at_ intel com

Cheng Zhuo (Communications)

cheng.zhuo _at_ intel com