ICCAD 2015 Contest: Incremental Timing-driven Placement
File formats for Benchmarks and Contest Deliverables

Myung-Chul Kim and Jin Hu

v1.1 – June 2^{nd}, 2015

http://cad-contest.el.cycu.edu.tw/problem_C/default.html

Contents

1 Introduction

The contest will primarily deal with localized set of keywords for each file format, both for input and output. It will also provide a more detailed explanation of the evaluation script.

2 Input Files: Benchmarks

Each benchmark is comprised of a set of seven (7) files with the following extensions: (*i*) `.iccad2015`, (*ii*) `.v`, (*iii*) `.lef`, (*iv*) `.def`, (*v*) `.sdc`, (*vi*) `_Early.lib`, and (*vii*) `_Late.lib`.

2.1 .iccad2015 (wrapper)

This file contains the list/location of the other six files needed for benchmark parsing:

```
simple.v simple.lef simple.def simple.sdc simple_Early.lib simple_Late.lib
```

It is used as the only input file when loading the benchmark, e.g., for the evaluation script and for your binary. The order of the file names is not important, e.g.,

```
simple.v simple.sdc simple.def simple.lef simple_Late.lib simple_Early.lib
```

is the same as the above. Each contained filename will have the same directory path location as the `.iccad2015` file. That is, if the call to load the above `.iccad2015` file is

```
%> myPlacer -settings <.parm> -input ../simple/simple.iccad2015 -ut
  <target_utilization> -max_disp <maximum_cell_displacement in μm>
```

then the six filenames in `simple.iccad2015` will have the same directory path:

```
./simple/simple.v
../simple/simple.lef
../simple/simple.def
../simple/simple.sdc
../simple/simple_Early.lib
../simple/simple_Late.lib
```

2.2 .v (verilog)

The verilog file specifies the top-level hierarchy of the design. For this contest, we will be using a small set of keywords with the verilog language. If you are implementing your own verilog parser, you will be expected to support the set of keywords found within the `simple.v` file (reproduced below for clarity). The design implemented in simple.v corresponds to the design found in Figure 1.

```
01.  module simple (
02.  inp1,
03.  inp2,
04.  iccad_clk,
05.  out
06.  );
07.
08.  // Start PIs
09.  input inp1;
10.  input inp2;
11.  input iccad_clk;
12.
13.  // Start POs
14.  output out;
15.
16.  // Start wires
17.  wire n1;
18.  wire n2;
19.  wire n3;
20.  wire n4;
21.  wire inp1;
22.  wire inp2;
23.  wire iccad_clk;
24.  wire out;
25.  wire lcb1_fo;
26.
27.  // Start cells
28.  NAND2_X1 u1 ( .a(inp1), .b(inp2), .o(n1) );
29.  NOR2_X1 u2 ( .a(n1), .b(n3), .o(n2) );
30.  DFF_X80 f1 ( .d(n2), .CK(lcb1_fo), .q(n3) );
31.  INV_X1 u3 ( .a(n3), .o(n4) );
32.  INV_X1 u4 ( .a(n4), .o(out) );
33.  INV_Z80 lcb1 ( .a(iccad_clk), .o(lcb1_fo) );
34.
35.  endmodule
```

Lines 01 and 35 define the start and end of the specified design with the keywords `module` and `endmodule`. Lines 01-06 specify the input and output connection names of the module (note that the direction is not specified here). Lines 09-11 specify the primary inputs (PIs) of the module with the keyword `input`. These names must match the ones stated with module (lines 01-06). Line 14 specifies the primary output (PO) of the module with the keyword `output`. This name must match the one stated with `module` (lines 01-06). Lines 17-25 specify the

connections or nets within the module with the keyword `wire`. These connections specify both the external PIs and POs as well as the internal connections between gates (explained further after lines 28-33). Lines 28-33 specify the cells used in the design, as well as how the cells are connected. Each definitions syntax is

cellName instanceName (.*pinName* (*netName*))

where every *cellName* and *.pinName* should be a specified library file (e.g., `simple.def`), and every *netName* should match one of the specified `wire` statements (lines 17-25). For example, on line 28, a `NAND2_X1`-type cell instance of `u1` is specified, its `a` pin is fed by primary input `inp1`, its `b` pin is fed by primary input `inp2`, and its `o` pin feeds the internal net `n1`. On line 29, `n1` feeds the `a` pin of the `NOR2_X1`-type cell instance `u2`.

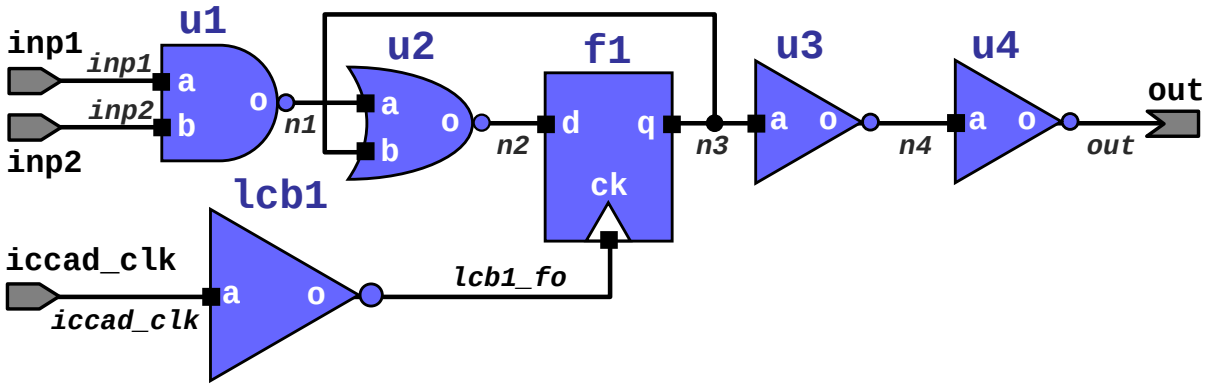

Figure 1: Implementation of `simple.v`.

2.3 .lef (LEF)

The `.lef` file (in Library Exchange Format (LEF) specifies the library definitions of the different cells, metal layers, and placement layout of the design. For this contest, we will be using a very small subset of the keywords in the LEF language. If you are implementing your own parser, you will be expected to support what is defined in `simple.lef` (in the simple benchmark), as well as defined comments beginning with the character '`#`'. The following explains the general format of the necessary keywords, with snippets of `simple.lef` reproduced for clarity. The relevant keywords are those listed in the header, as well as **SITE**, **LAYER**, and **MACRO**.

LEF structure. These initial statements are used to set the parsing rules of the LEF file.

```
01. VERSION 5.7 ;
02. NAMESCASESENSITIVE ON ;
03. BUSBITCHARS "[]" ;
04. DIVIDERCHAR "/" ;
05.
06. UNITS
07.    DATABASE MICRONS 2000 ;
08. END UNITS
09.
10. MANUFACTURINGGRID 0.0025 ;
...
99. END LIBRARY
```

Line 01 specifies the version number of the LEF language used. Line 02 specifies the mode of name parsing. By default, we will assume this to be "ON". Line 03 specifies the special character set used for stating grouped signals e.g., as buses. For instance, `myInputs[31]` specifies that `myInputs` spans 32 bits. For this contest, we will not be defining bus signals (but the keyword may be part of the LEF file). Line 04 specifies the hierarchy divider special character, e.g., `myBox/a` states that pin `a` is contained within the instance `myBox`. Line 06-08 specify the unit conversion from LEF database (distance) units to microns (μm). In this example, every 1000 LEF database units is equivalent to 1 (μm). Line 10 specifies the grid granularity in μm – all shapes' and cells' locations must be snapped to a multiple of that value in the grid. This value must be positive. Line 99 specifies the end of the file.

SITE. Each site defines a placement grouping in the design, i.e., the placement grid for a family of macros and cells. These sites will be used in the keyword `ROW` in the Design Exchange Format (DEF) file (Section 2.4). For this contest, the **SITE** keyword is limited to the following:

```
01. SITE siteName
02.    CLASS { PAD | CORE } ;
03.    SYMMETRY {X | Y} ;
04.    SIZE width BY height ;
05. END siteName
```

The following keywords are expected to have the following fields:

- **SITE** (Lines 01 and 05): *siteName* is an alphanumeric string no longer than 64 characters.
- **CLASS** (Line 02): PAD – specifying an I/O pad site – or CORE – specifying a core site.
- **SYMMETRY** (Line 03): X (Y) – specifying the row's symmetry about the X- (Y-) axis. As this contest does not allow cell flipping, this value will not be used.
- **SIZE** (Line 04): *width* and *height* are the dimensions (size) of the site specified in microns.

The following shows an example site (the keywords specified within `SITE` can be in any order).

```
01.  SITE core
02.     SIZE 0.19 BY 1.71 ;
03.     CLASS CORE ;
04.     SYMMETRY Y ;
05.  END core
```

Line 01 and line 05 define the site with name `core`. Line 02 defines the dimensions width and height of the site to be 0.19 μm $\times$ 1.71 μm. Line 03 specifies the site is of type CORE. Line 04 specifies the symmetry of the site to be about the Y-axis.

LAYER. Each metal layer that is available in the design will be specified here. This information is used to specify routing layers and where pins are located. For this contest, we will only use the ROUTING type with a limited set of keywords.

```
01.  LAYER layerName
02.     TYPE ROUTING ;
03.     DIRECTION {HORIZONTAL | VERTICAL} ;
04.     PITCH {distance | xDistance yDistance} ;
05.     WIDTH defaultWidth ;
06.     OFFSET {distance | xDistance yDistance } ;
07.  END layerName
```

The following keywords are expected to have the following fields:

- **LAYER** (Lines 01 and 07): *layerName* is an alphanumeric string no longer than 64 characters.
- **TYPE** (Line 02): ROUTING is currently the only expected option.
- **DIRECTION** (Line 03): the layer either has routing tracks parallel to the X-axis (HORIZONTAL) or the Y-axis (VERTICAL).
- **PITCH** (Line 04): specifies the minimum distance between each routing track in microns (float). If only *distance* is specified, it is for both vertical and horizontal tracks. Otherwise, *xDistance* and *yDistance* specify for the horizontal and vertical tracks, respectively.
- **WIDTH** (Line 05): *defaultWidth* specifies the routing-track width for all (regular) routing tracks in microns.
- **PITCH** (Line 06): specifies the offset distance of the routing grid to align with the standard cell grid in distance units (float). If only *distance* is specified, it is for both x-offset and y-offset. Otherwise, *xDistance* and *yDistance* specify for the x-offset and y-offset, respectively.

The following shows an example definition of a metal layer.

```
01.  LAYER metal1
02.     TYPE ROUTING ;
03.     DIRECTION HORIZONTAL ;
04.     PITCH 0.200 ;
05.     OFFSET 0.000 ;
06.     WIDTH 0.100 ;
07.  END metal1
```

Lines 01 and 07 specify the definition of the layer with name `metal1`. Line 02 specifies the layer type to be ROUTING. Line 03 specifies the routing tracks on this layer are HORIZONTAL. Line 04 specifies the minimum spacing between routing tracks is 0.2 μm. Line 05 specifies no offset with the routing grid and the standard-cell boundaries. Line 06 specifies the track width to be 0.1 μm.

MACRO. Each definition specifies the definition of a cell type, which can be instantiated in other files, e.g., `simple.v`. For this contest, we will be using a limited set of keywords associated with **MACRO**.

```
01.  MACRO macroName
02.     CLASS {PAD | CORE} ;
03.     ORIGIN point ;
04.     SIZE width BY height ;
05.     SITE siteName ;
06.     PIN pinName DIRECTION {INPUT | OUTPUT}
07.        PORT
08.        LAYER layerName ;
09.        RECT point point ;
10.        END
11.     END pinName
12.  END macroName
```

The following keywords are expected to have the following fields:

- **MACRO** (Lines 01 and 12): *layerName* is an alphanumeric string no longer than 64 characters.
- **CLASS** (Line 02): Specifies either an I/O pad (PAD) or standard-cell area (CORE).
- **ORIGIN** (Line 03): Specifies a point (x,y) where to align the origin of the macro to the DEF COMPONENTS placement point (Section 2.4). The macro is shifted by (x,y) before alignment. For example, if point is (0,$-$1), then all macro objects at (0,1) are shifted to (0,0).
- **SIZE** (Line 04): Specifies the dimensions (size) *width* and *height* of the macro in microns.
- **SITE** (Line 05): Specifies the macro's associated *siteName* (defined as using **SITE**).

- **PIN** (Lines 06 and 11): Specifies a input (INPUT) or output (OUTPUT) pin with name *pinName*. Lines 07-10 are associated with this keyword.
- **PORT** (Lines 07 and 10): Required (no following variable input).
- **LAYER** (Line 08): Specifies the macro pin's associated *layerName* (defined with **LAYER**).
- **RECT** (Line 09): Specifies two points (x_1,y_1) and (x_2,y_2) that define the locations of two opposite corners of the macro pins (specified in microns). If multiple point locations are specified (e.g., to construct a polygon-shape pin), the pin location (e.g., used for routing purposes) will be the center of a bounding box of all **RECT** statements.

The following is an example definition of a macro.

```
01.  MACRO INV_X1
02.     CLASS CORE ;
03.     ORIGIN 0 0 ;
04.     SIZE 0.760 BY 1.71
05.
06.     SITE core ;
07.     PIN o DIRECTION OUTPUT ;
08.        PORT
09.        LAYER metal1 ;
10.        RECT 0.465 0.150 0.53 1.255
11.        RECT 0.415 0.150 0.61 0.28
12.        END
13.     END o
14.     PIN a DIRECTION INPUT ;
15.        PORT
16.        LAYER metal1 ;
17.        RECT 0.210 0.340 0.34 0.405
18.        END
19.     END a
20.  END INV_X1
```

Lines 01 and 19 specify the definition of the macro `INV_X1`. Line 02 specifies the macro type to be CORE. Line 03 specifies the origin of `INV_X1` to be at (0,0). Line 04 specifies the size of the macro to be 0.76 μm $\times$ 1.71 μm. Line 06 specifies that this macro belongs to the site `core`. Lines 07-13 specify an output pin `o` located on layer `metal1`. It is comprised of two rectangles, the first with lower-left and upper-right coordinates being (0.465 μm, 0.15 μm) and (0.53μm, 1.255 μm), respectively, and the second with lower-left and upper-right coordindates being (0.415 μm, 0.15 μm) and (0.61μm, 0.28 μm), respectively. Its effective pin location will be the center of the smallest bounding box that encloses these two rectangles. Lines 14-19 specify an input pin `a`, located on layer `metal1`, with lower-left and upper-right coordinates being (0.21 μm, 0.34 μm) and (0.34μm, 0.405 μm), respectively.

2.4 .def (DEF)

The .def file (in Design Exchange Format (LEF) specifies the different cells and connectivity of the design. For this contest, we will be using a very small subset of the keywords in the DEF language. If you are implementing your own parser, you will be expected to support what is defined in simple.def, as well as defined comments beginning with the character '`#`'. The following explains the general format of the necessary keywords, with snippets of `simple.def` reproduced for clarity. The relevant keywords are those listed in the header, as well as **ROW**, **COMPONENTS**, **PINS**, and **NETS**.

DEF structure. These initial statements are used to set the parsing rules of the DEF file.

```
01. VERSION 5.7 ;
02. DIVIDERCHAR "/" ;
03. BUSBITCHARS "[]" ;
04. DESIGN simple ;
05. UNITS DISTANCE MICRONS 2000 ;
06.
07. DIEAREA ( 0 0 ) ( 13680 13680 ) ;
...
99. END DESIGN
```

Line 01 specifies the DEF language version used. Line 02 specifies the hierarchy delimiter character. For instance, `myBox/a` defines the `a` to be within the instance `myBox`. Line 03 specifies the character set used to specify groups of signals, e.g., bus signals. For instance, `myInputs[31]` specify a signal that spans 32 bits. For this contest, we will not be defining bus signals (but the keyword may be part of the DEF file). Line 04 specifies the design name. Line 05 specifies the unit conversion between DEF database units to microns (μm). In this example, every 2000 DEF units is equivalent to 1 μm. Line 07 specifies the lower-left and upper-right coordinates of the total layout area in DEF database units (integers). Line 99 specifies the end of the DEF file.

ROW. Each row specifies a region where standard cells are placed. It has the following syntax:

ROW *rowName siteName origX origY siteOrient* `DO` *numX* `BY` *numY* `STEP` *stepX stepY*

Here, *rowName* defines the row name, *siteName* defines the associated site name (as defined by **SITE**, (*origX*, *origY*) defines the origin (lower-left) of the row in DEF database units (integers), and *siteOrient* defines site orientation (up to two characters). The `DO` *numX* `BY` *numY* segment specifies the number of sites in **ROW**. At least one of *numX* and *numY* must equal 1; if *numX* = 1, then the row is vertical, and if *numY* = 1, then the row is horizontal. If both *numX* = *numY* = 1, then the row is a single site. The `STEP` *stepX stepY* segment specifies the spacing between sites in the horizontal (*stepX*) or vertical (*stepY*) row in DEF database units. The following shows an example definition of a row with name `core_SITE_ROW_2` in the site core at

(0,6840) with 'N' orientation. The row is horizontal with 36 sites and 380 DEF database units between each site.

ROW `core_SITE_ROW_2 core 0 6840 N DO 36 BY 1 STEP 380 0 ;`

COMPONENTS. All cell instances are defined here with DEF database units.

```
01.  COMPONENTS numComponents ;
02.     – instanceName cellName
03.        + {PLACED | FIXED} ( xLoc yLoc ) orientation ;
04.  END COMPONENTS
```

In line 01, *numComponents* specifies the total number of components or cell instances there are in the design. Line 02 specifies the instance *instanceName* and cell-type *cellName* names of the component, e.g., those defined in the verilog file. Line 03 specifies if the cell instance is movable (PLACED) or not movable (FIXED), along with the lower-left coordinate of the cell (*xLoc*,*yLoc*) in DEF database units and its orientation. Line 04 specifies the end of the **COMPONENTS** keyword. The following shows an example definition of components:

```
01.  COMPONENTS 6 ;
02.     - u1 NAND2_X1
03.        + PLACED ( 3420 6840 ) N ;
04.     - u2 NOR2_X1
05.        + PLACED ( 3420 3420 ) N ;
06.     - f1 DFF_X80
07.        + FIXED ( 684 0 ) N ;
08.     - u3 INV_X1
09.        + PLACED ( 6840 3420 ) N ;
10.     - u4 INV_X1
11.        + PLACED ( 12160 6840 ) N ;
12.     - lcb1 INV_Z80
13.        + PLACED ( 0 12160 ) N ;
14.  END COMPONENTS
```

Lines 01 and 12 specify six (6) cell instances in the components definition. Lines 02-03 define a movable instance of `u1` of cell-type `NAND2_X1` at location (3420,6840) with orientation 'N'. Lines 04-05 define a movable instance of `u2` of cell-type `NOR_X1` at location (3420,3420) with orientation 'N'. Lines 06-07 define a non-movable instance of `f1` of cell-type `DFF_X80` at location (684,0) with orientation 'N'. Lines 08-09 define a movable instance of `u3` of cell-type `INV_X1` at location (6840,3420) with orientation 'N'. Lines 10-11 define a movable instance of `u4` of cell-type `INV_X1` at location (12160,6840) with orientation 'N'. Lines 12-13 define a movable instance of `lcb1` of cell-type `INV_Z80` at location (0,12160) with orientation 'N'.

PINS. In the DEF file, only the primary input and outputs pins of the design are specified.

```
01.  PINS numPins ;
02.     - pinName + NET netName
03.        + DIRECTION {INPUT | OUTPUT}
04.        + {PLACED | FIXED} point orientation
05.        + LAYER layerName ( point ) ( point ) ;
06.  END PINS
```

Line 01 specifies the number of primary input and output pins in the design (*numPins*). Line 02 specifies the individual pin name (*pinName*) and its connected net (*netName*). These names should match the ones specified in the other files, e.g., verilog. Line 03 specifies the pin as either a primary input (INPUT) or primary output (OUTPUT). Line 04 specifies if the pin is movable (PLACED) or non-movable (FIXED), along with the lower-left coordinate of the pin (x,y) in DEF database units and the orientation. Line 05 specifies the layer (*layerName*) the pin is located on, as well as its geometry, defined in DEF database units, as the two opposite corners. Note that the semi-colon (;) character denotes the end of the pin read – if lines 03 and 05 were in reverse order, the semi-colon would be after {INPUT — OUTPUT}. Line 06 specifies the end of the **PINS** keyword. The following is example pins definition:

```
01.  PINS 4 ;
02.     - out + NET out
03.        + DIRECTION OUTPUT
04.        + FIXED ( 13680 6000 ) N
05.        + LAYER metal3 ( 0 0 ) ( 100 100 ) ;
06.     - inp1 + NET inp1
07.        + DIRECTION INPUT
08.        + FIXED ( 3420 13680 ) N
09.        + LAYER metal3 ( 0 0 ) ( 100 100 ) ;
10.     - inp2 + NET inp2
11.        + DIRECTION INPUT
12.        + FIXED ( 6840 13680 ) N
13.        + LAYER metal3 ( 0 0 ) ( 100 100 ) ;
14.     - iccad_clk + NET iccad_clk
15.        + DIRECTION INPUT
16.        + FIXED ( 0 6480 ) N
17.        + LAYER metal3 ( 0 0 ) ( 100 100 ) ;
18.  END PINS
```

Lines 01 and 18 asserts four (4) primary inputs and outputs in the design. Lines 02-05 define the primary output out pin connected to net out with fixed location (13680,6000) and orientation 'N' on `metal3` with size 100 × 100. Lines 06-09 define the primary input `inp1` pin connected to net `inp1` with fixed location (3420,13680) and orientation 'N' on `metal3` with size 100 ×

100. Lines 10-13 define the primary input `inp2` pin connected to net `inp2` with fixed location (6840,13680) and orientation 'N' on `metal3` with size 100 × 100. Lines 14-17 define the primary input `iccad_clk` pin connected to net `iccad_clk` with fixed location (0,6480) and orientation 'N' on `metal3` with size 100 × 100.

2.5 .lib (Timing Liberty format) and .parm

The early / late timing information associated with standard cells and macro blocks is contained in Synopsys Liberty format. The contest timer reads verilog(`.v`), parasitic information (`.spef`), and timing information(`.lib`) to measure timing. The spef files are internally generated by the contest evaluation script. The parameter file (`.parm`) specifies technology-dependent parameters, such as local (intermediate) wire resistances / capacitances per m. Those parameters are used to extract parasitic information of a given placement and then to write the spef files.

2.6 .sdc (Constraints)

The Synopsys Design Constraints (SDC) file specifies the set of assertions or constraints used as initial timing conditions for the design. For this contest, we will be using a very limited set of commands. If you are writing your own parser, you are expected to support the commands and keywords specified in `simple.sdc`, as well as the comments starting with '`#`'. All time values are specified in picoseconds (ps). We expect to be using the following five commands and their associated options in a restricted manner:

- **get_ports** {*pinName pinName* ...}:
  returns a set of pins specified by the list of *pinNames*, which are enclosed in {}. A single pinName does not require {}, but can have them.
- **create_clock** `-name` *clockName* `-period` *val* [**get_ports**...]:
  creates a clock signal *clockName* with a corresponding period of *val*, and associates it with the primary input (clock) pin returned from **get_ports**.
- **set_input_delay** *val* `-clock` *clockName* [**get_ports**...]:
  sets the arrival time of the primary input pins with *val* time with respect to *clockName*.
- **set_driving_cell** `-lib_cell` *cellName* `-pin` *pinName*
  `-input_transition_fall` *val* `-input_transition_rise` *val* [**get_ports**...]:
  sets the cell type (*cellName*) and pin name (*pinName*) with associated input slews for rise and fall.
- **set_output_delay** `-clock` *clockName val* [**get_ports**...]:
  sets the required arrival time of the primary output signals with respect to *clockName*.
- **set_load** `-pin_load` *val* [**get_ports**...]:
  asserts the capacitance (specified in fF) at the output pin.

This is similar to that shown in Figure 2. The following shows an example set of assertions.

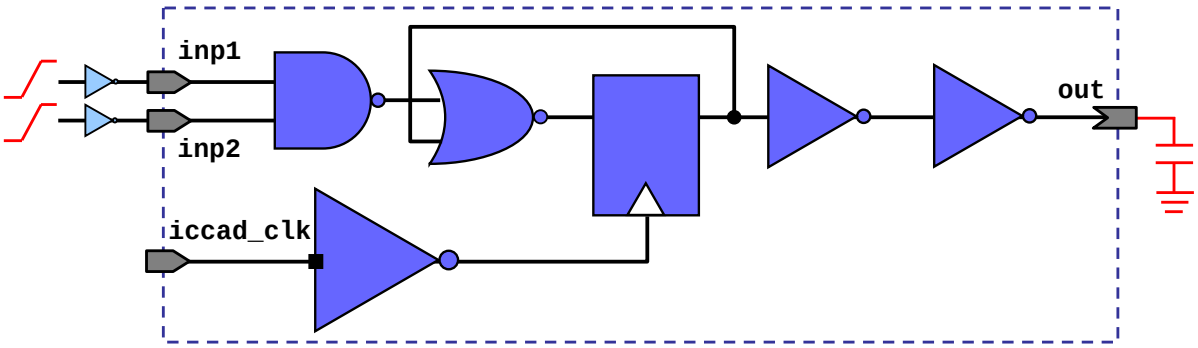

Figure 2: Circuit in `simple.v` with assertions.

```
01.  create_clock -name mclk -period 150.0 [get_ports iccad_clk]
02.  set_input_delay 0.0 [get_ports {inp1}] -clock mclk
03.  set_input_delay 0.0 [get_ports {inp2}] -clock mclk
04.  set_driving_cell -lib_cell INV_X80 -pin o [get_ports {inp1}]
   -input transition fall 10.0 -input transition rise 10.0
05.  set_driving_cell -lib_cell INV_X80 -pin o [get_ports {inp2}]
   -input_transition_fall 10.0 -input_transition_rise 10.0
06.  set_output_delay 0.0 [get_ports {out}] -clock mclk
07.  set_load -pin_load 4.0 [get_ports {out}]
```

Line 01 creates a clock signal `mclk` with a period of 150 ps, and assigns it to the primary input `iccad_clk`. Line 02 (03) sets the arrival time of `inp1` (`inp2`) to 0, with respect to the `mclk` signal. Line 04 (05) sets the input slew of 10 ps at the driving cell *INV_X80* feeding `inp1` (`inp2`) for both rise and fall. Line 06 sets the required time at out to 0, with respect to the `mclk` signal. Line 07 sets the output capacitance attached to out to 4 fF.

3 Output Files

The expected output files (that are generated by your placer) are (*i*) a simplified DEF file (.def) that reports the placement locations in DEF database units of all cell instances in the design, and (*ii*) circuit modification file (.ops) that reports any changes in the netlist in terms of flipflip-to-LCB associations.

3.1 A simplified DEF (.def)

In this DEF output file, you should include all cells, i.e., those with either the `FIXED` or `PLACED` property. The expected keywords are (along with any comments): **VERSION**, **DESIGN**, and **COMPONENTS**. For details regarding the syntax of these keywords and a full explanation of DEF, please refer to Section 2.4. Below shows an example output DEF file:

```
01.  VERSION 5.7 ;
02.  DESIGN simple ;
03.
04.  COMPONENTS 6 ;
05.     - u1 NAND2_X1
06.        + PLACED ( 3420 6840 ) N ;
07.     - u4 NOR2_X1
08.        + PLACED ( 3420 3420 ) N ;
07.     - f1 DFF_X80
08.        + FIXED ( 684 0 ) N ;
09.     - u3 INV_X1
10.        + PLACED ( 6840 3420 ) N ;
11.     - u4 INV_X1
12.        + PLACED ( 12160 6840 ) N ;
13.     - lcb1 INV_Z80
14.        + PLACED ( 0 12160 ) N ;
15.  END COMPONENTS
16.
17.  END DESIGN
```

Line 01 documents the DEF language version. Line 02 states the design name. Lines 04 and 15 state the set of five (6) components or cells in the design. Lines 05-06 state the cell instance `u1` is at (3420,6840). Lines 07-08 state the cell instance `u2` is at (3420,3420). Lines 07-08 state the cell instance `f1` is at (684,0). Lines 09-10 state the cell instance `u3` is at (6840,3420). Lines 11-12 state the cell instance `u4` is at (12160,6840). Lines 13-14 state the cell instance `lcb1` is at (0,12160). The orientations should be 'N', as cell flipping/rotation is disallowed in the contest.

3.2 A circuit modification file (.ops)

The file contains a list of circuit modification operations in case of changing original FF-to-LCB associations. The file shares the same format with OPS file from TAU 2015 contest, but only two pin-level modification operations are allowed.

- `disconnect_pin <pin name>` : removes the FF's clk pin from its driving LCB's fanout.
- `connect_pin <pin name> <net name>` : adds the FF's clk pin to new LCB's fanout.

The listed circuit modification operations in OPS file occur in order, before the evaluation script generates SPEF file to run a contest timer. For a FF's clock pin, note that `disconnect_pin` operation must precede `connect_pin` operation. The evaluation script checks whether all FF's clock pins are properly connected to LCBs. If you do not want to make any changes in terms of FF-to-LCB associations, please leave the file empty. Below shows an example output OPS file (final.ops) for the `simple` benchmark, which is effectively no-op:

```
01.  disconnect_pin f1:ck
02.  connect_pin f1:ck lcb1_fo
```

Line 01 detaches the pin `f1:ck` from its driving net (`lcb1_fo`). Line 02 attaches the pin `f1:ck` back to the net `lcb1` fo as a sink.

4 Updates

- 06/02/15 [v1.1]: early and late timing library files included in `.iccad2015`
- 05/07/15 [v1.0]: initial version.