ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
ĐẠI HỌC BÁCH KHOA
NGÀNH KỸ THUẬT ĐIỆN TỬ

Chapter 4
Logic Synthesis

PGS.TS. HOÀNG TRANG
Bộ môn Kỹ Thuật Điện Tử
hoangtrang@hcmut.edu.vn

Static Timing Analysis

- ❖Dynamic vs. Static Timing Analysis
- ❖Performance of Design
- ❖Synthesis step

Dynamic vs. Static Timing Analysis

❖Timing analysis is integral part of ASIC/VLSI design flow. Anything else can be compromised but not timing! Timing analysis can be **static** or **dynamic**.

❖**Dynamic timing analysis verifies functionality** of the design by applying input vectors and checking for correct output vectors whereas **Static Timing Analysis** checks **static delay requirements** of the circuit without any input or output vectors.

❖**Dynamic timing analysis(DTA)** has to be accomplished and functionality of the design must be cleared **before** the design is subjected to **Static Timing Analysis (STA).**

❖**Dynamic Timing Analysis (DTA)** and **Static Timing Analysis (STA)** are **not alternatives** to each other.

❖**Quality of the Dynamic Timing Analysis (DTA) increases** with the **increase of input test vectors**. Increased test vectors increase simulation time. Dynamic timing analysis can be **used for synchronous as well as asynchronous designs.** Dynamic Timing Analysis (DTA) is also **best suitable for designs having clocks crossing multiple domains**

❖**Static Timing Analysis (STA) can't run on asynchronous deigns** and hence Dynamic Timing Analysis (DTA) is the best way to analyze asynchronous designs. .

Dynamic vs. Static Timing Analysis

❖In **Static Timing Analysis (STA)** static delays such as **gate delay and net delays** are considered in each path and these delays are compared against their required maximum and minimum values.

❖ Circuit to be analyzed is broken into different timing paths constituting of gates, flip flops and their interconnections. Each timing path has to process the data within a clock period which is determined by the maximum frequency of operation.

❖**Cell delays are available in the corresponding technology libraries.** Cell delay values are tabulated based on input transition and fanout load which are characterized by simulation tools.

❖ **Net delays are calculated based on the Wire Load Models(WLM)** or extracted resistance R and capacitance C. Wire Load Models(WLM) are available in the Technology File. These values are Table Look Up(TLU) values calculated based on the net fanout length.

Dynamic vs. Static Timing Analysis

Advantages of STA:

❖All timing paths are considered for the timing analysis. This is not the case in simulation.

❖Analysis times are relatively short when compared with event and circuit simulation.

❖Timing can be analyzed for worst case, best case simultaneously. This type of analysis is not possible in dynamic timing analysis.

❖Static Timing Analysis (STA) works with timing models. STA has more pessimism and thus gives maximum delay of the design. DTA performs full timing simulation. The problem associated with DTA is the computational complexity involved in finding the input patterns (vectors) that produce maximum delay at the output and hence it is slow.

Dynamic vs. Static Timing Analysis

Disadvantages of STA:

❖All paths in the design may not run always in worst case delay. Hence the analysis is pessimistic.

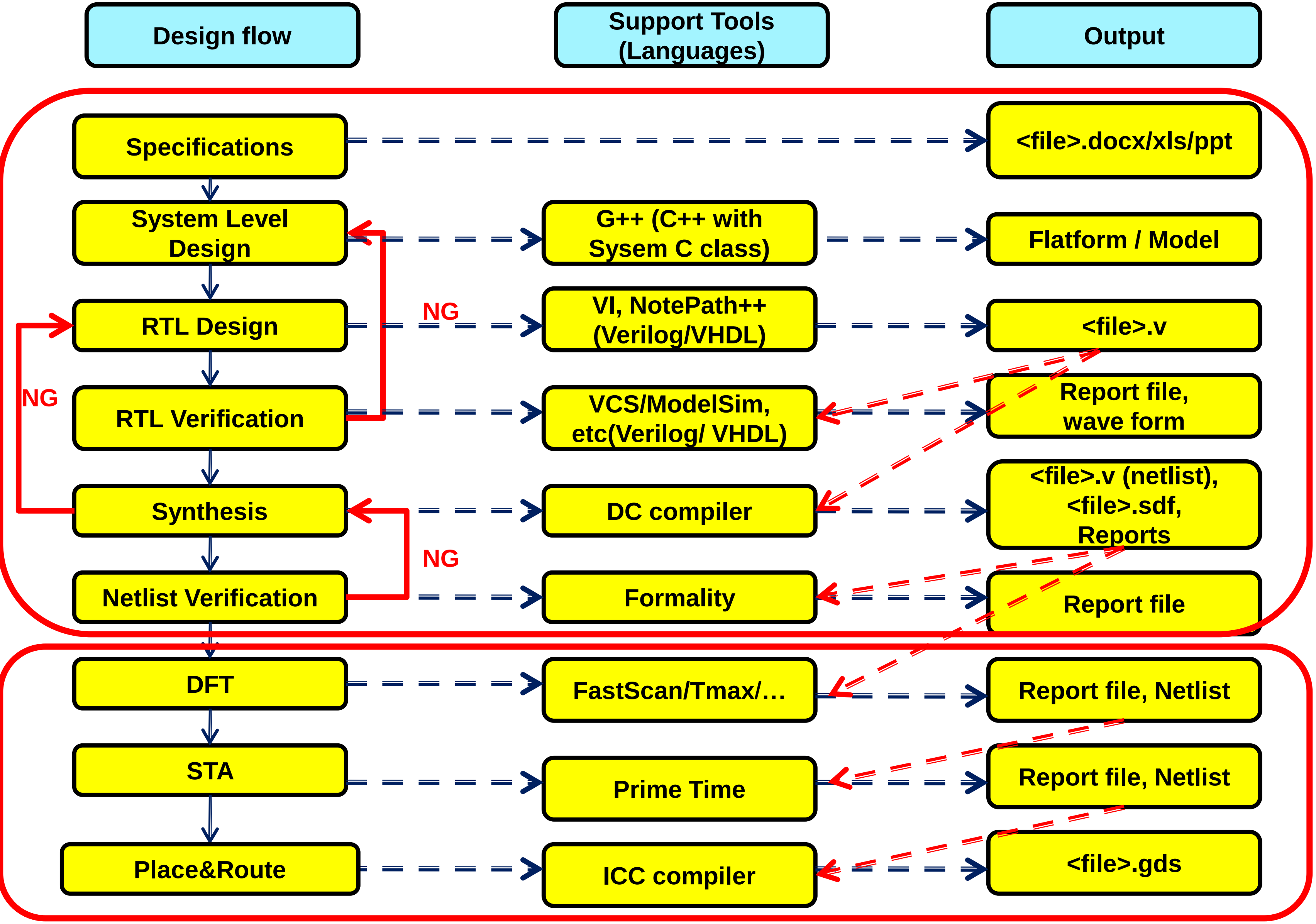
❖Clock related all information has to be fed to the design in the form of constraints.

❖Inconsistency or incorrectness or under constraining of these constraints may lead to disastrous timing analysis.

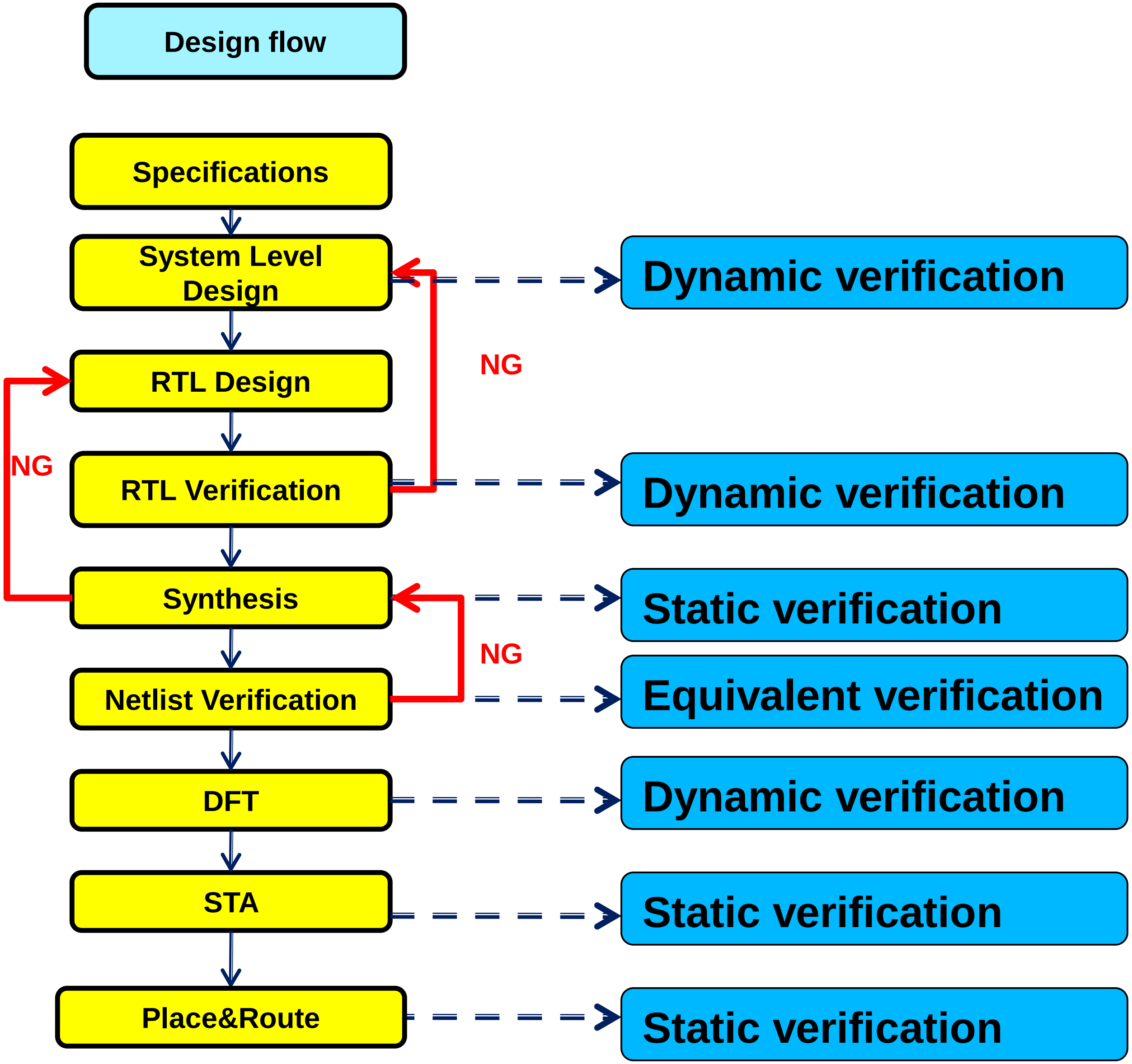
❖STA does not check for logical correctness of the design.

❖STA is not suitable for asynchronous circuits.

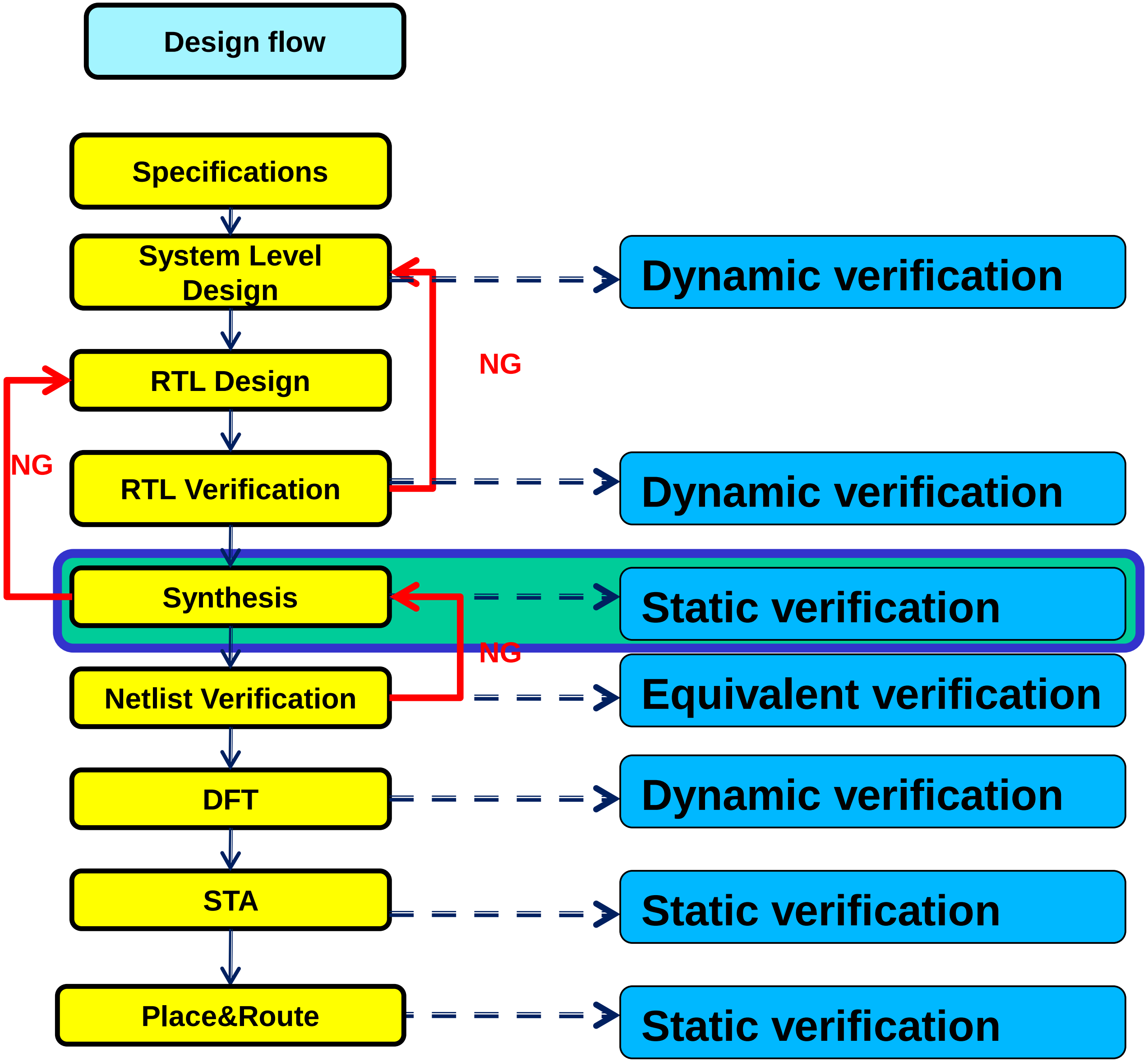
Dynamic vs. Static Timing Analysis



Dynamic vs. Static Timing Analysis



Dynamic vs. Static Timing Analysis



Design flow

Specifications

System Level Design

RTL Design

RTL Verification

Synthesis

Netlist Verification

DFT

STA

Place&Route

Dynamic verification

Dynamic verification

Static verification

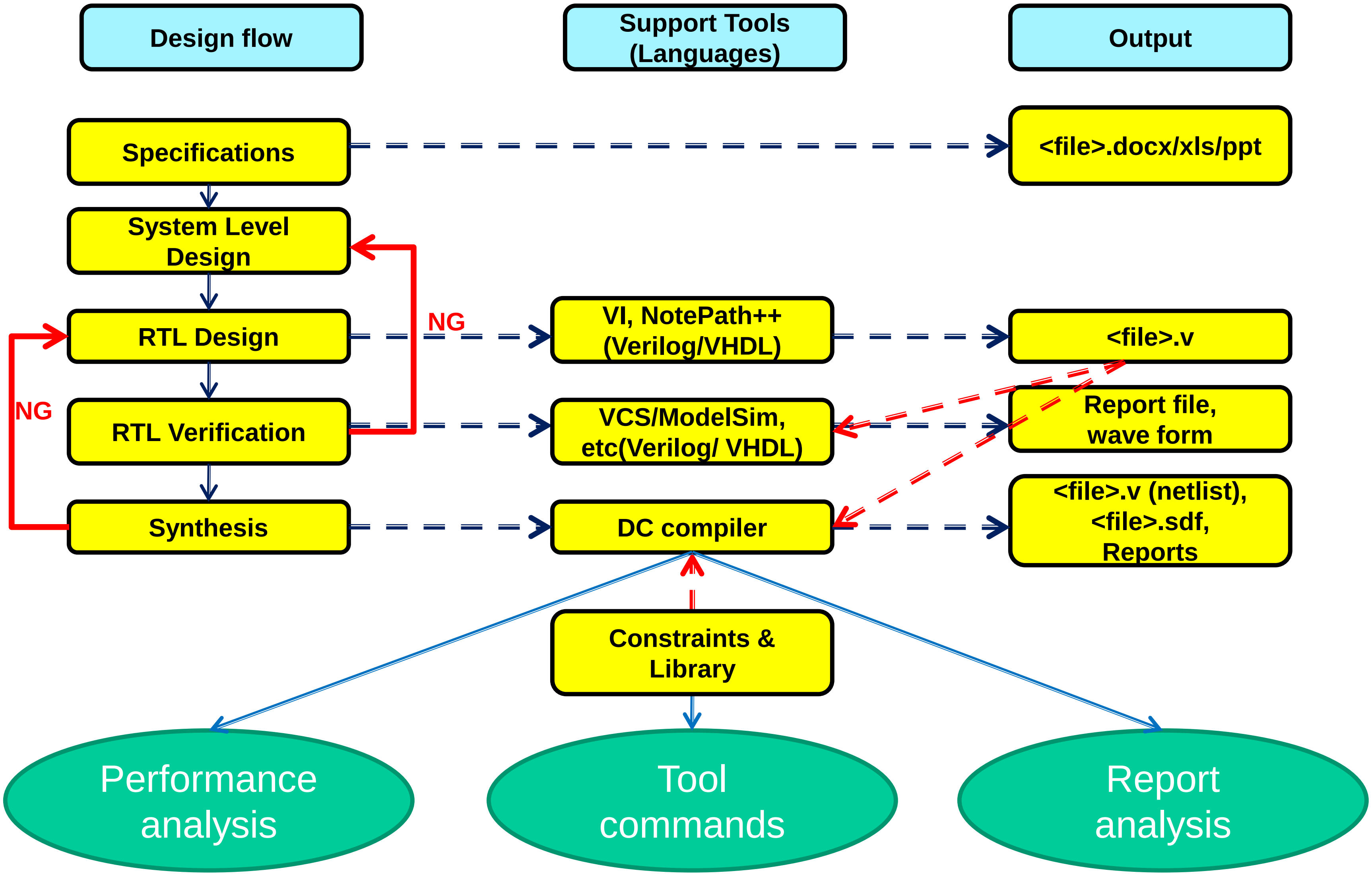
Equivalent verification

Dynamic verification

Static verification

Static verification

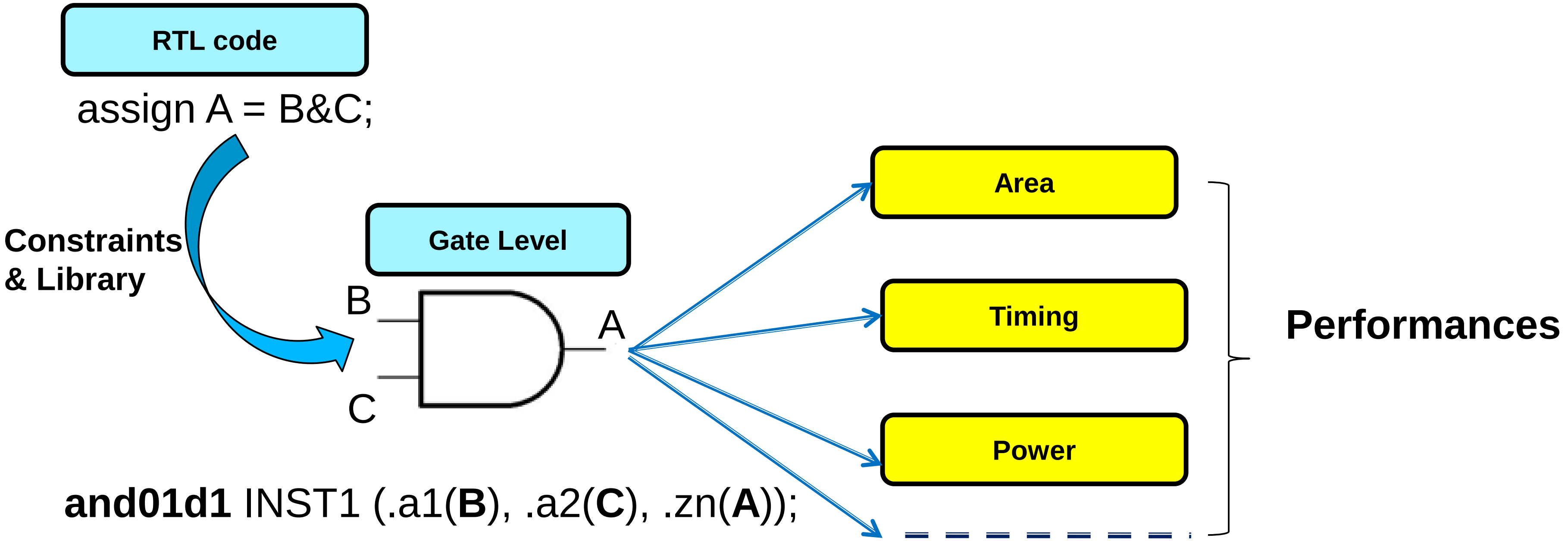
Dynamic vs. Static Timing Analysis



Static Timing Analysis

- ❖Dynamic vs. Static Timing Analysis
- ❖**Performance of Design**
- ❖Synthesis step

Performance of Design



RTL code

assign A = B&C;

Constraints
& Library

Gate Level

and01d1 INST1 (.a1(**B**), .a2(**C**), .zn(**A**));

Area

Timing

Power

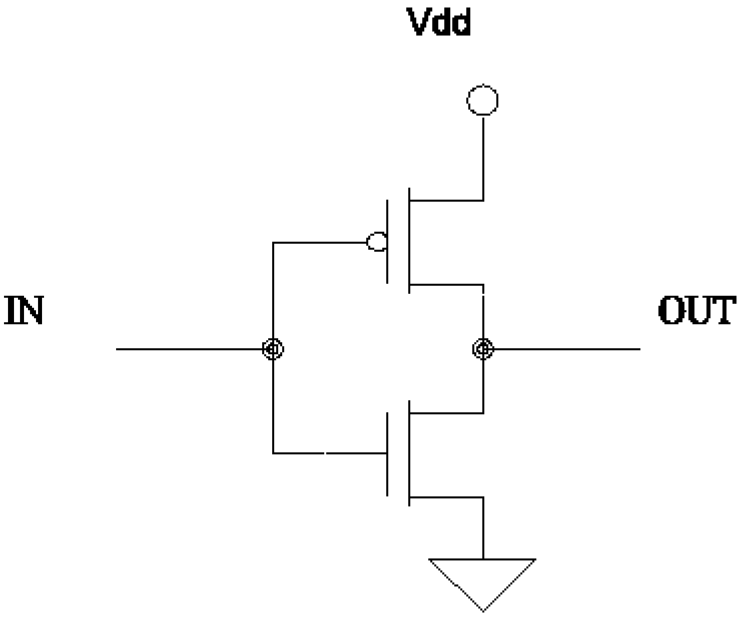
Performances

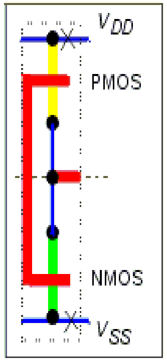
Q1. Why the gates have different performances?

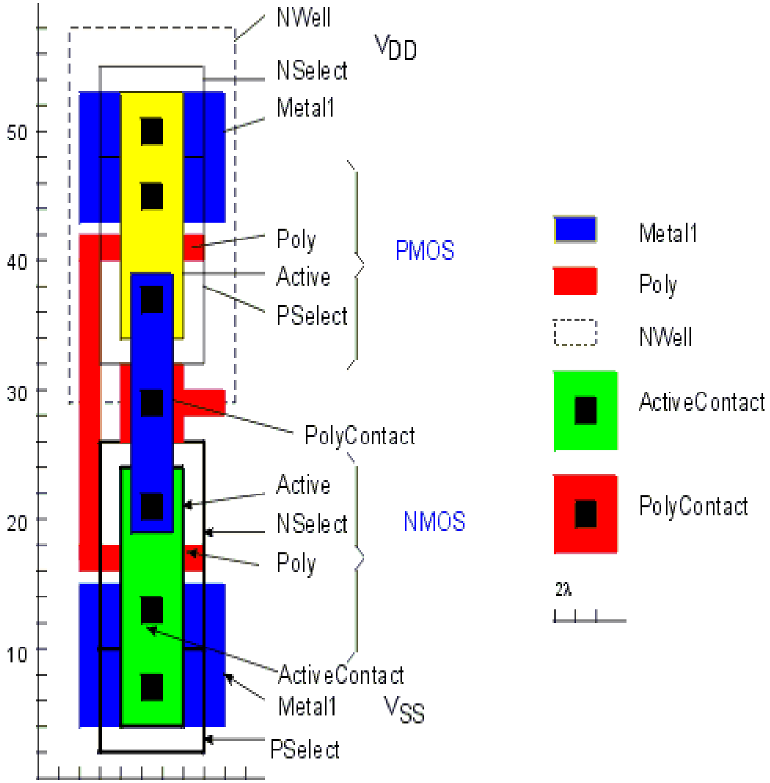
Q2. How to get the best performance ?

Performance of Design

Q1. Why same gates have different performances?



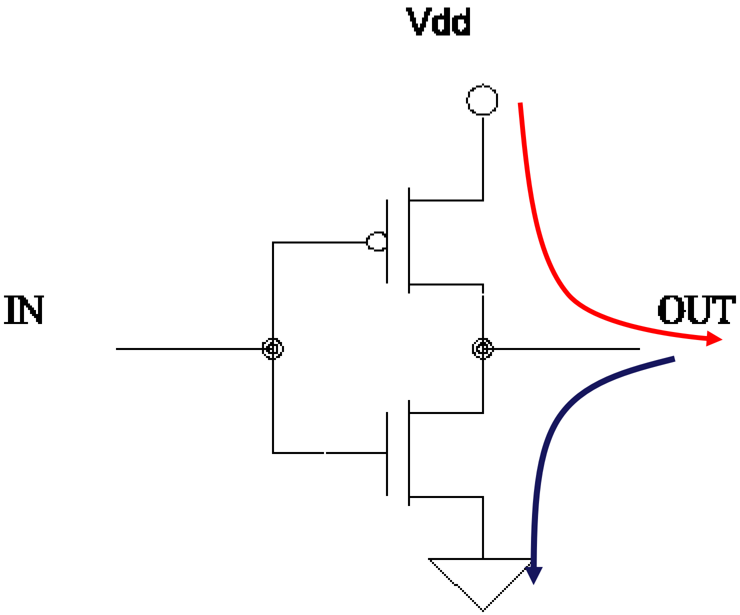


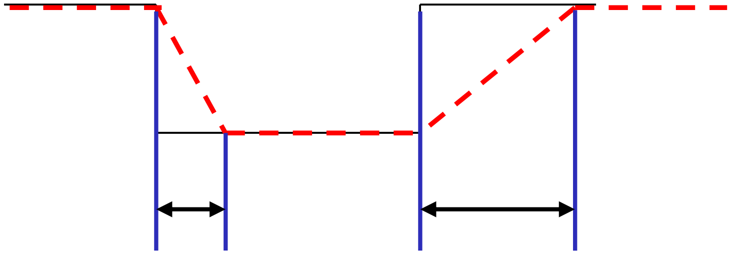


→ **The different layouts make the different performances**

Performance of Design

Q1. Why the gates have different performances?

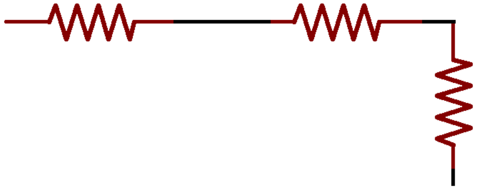




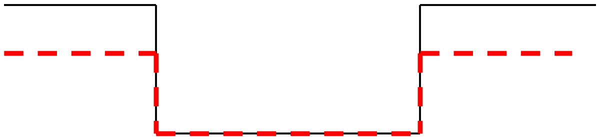
→ The different layouts make the different performances

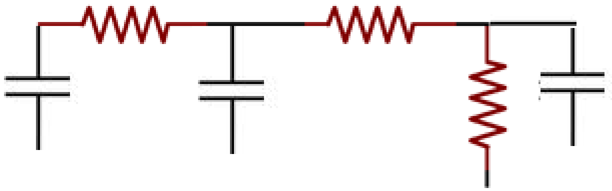
Dynamic vs. Static Timing Analysis

Q1. Why the gates have different performances?

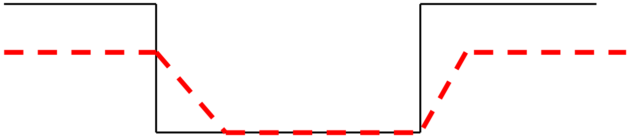


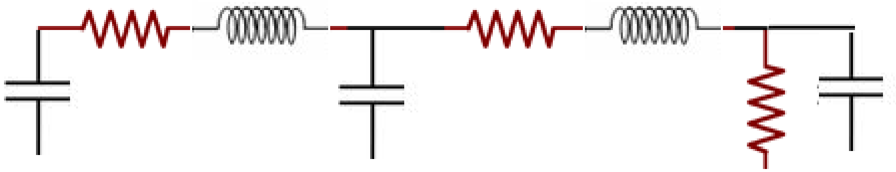
The effect of R



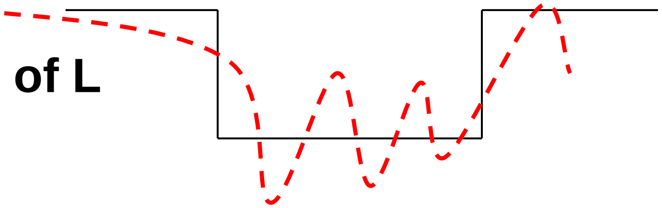


The effect of C





The effect of L

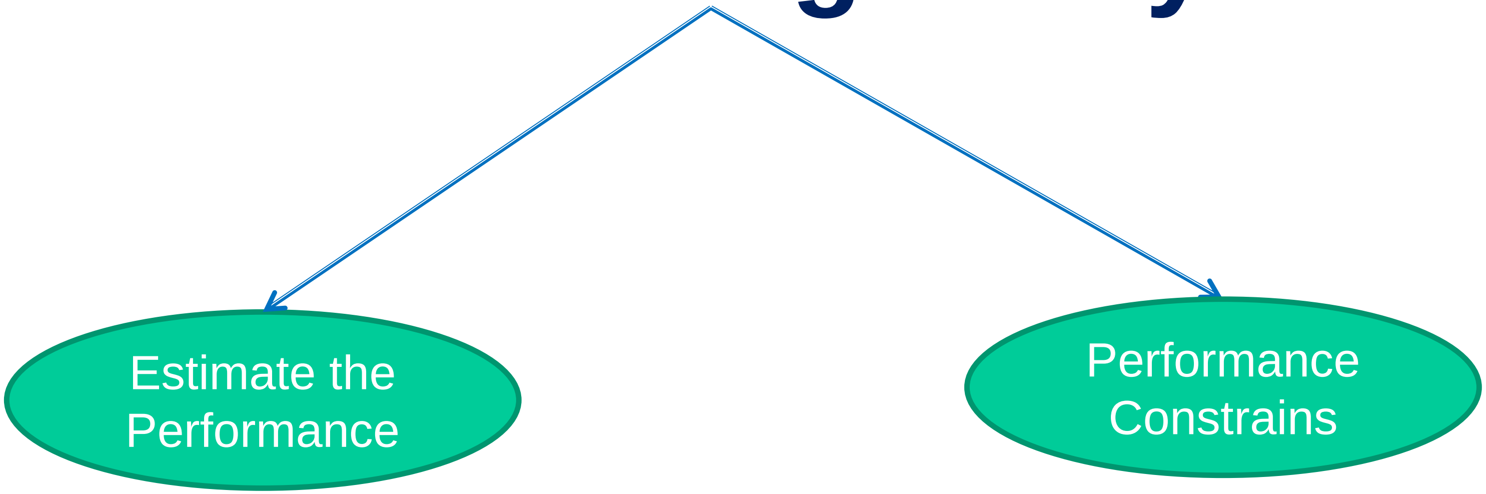


→ The different parasitic C, L, R

Performance of Design

Q2. How to get the best performance ?

Static Timing Analysis



Estimate the Performance

Performance Constrains

Area, Fanout, Trainsition, Path, Constrain, Clock skew, Setup Time …

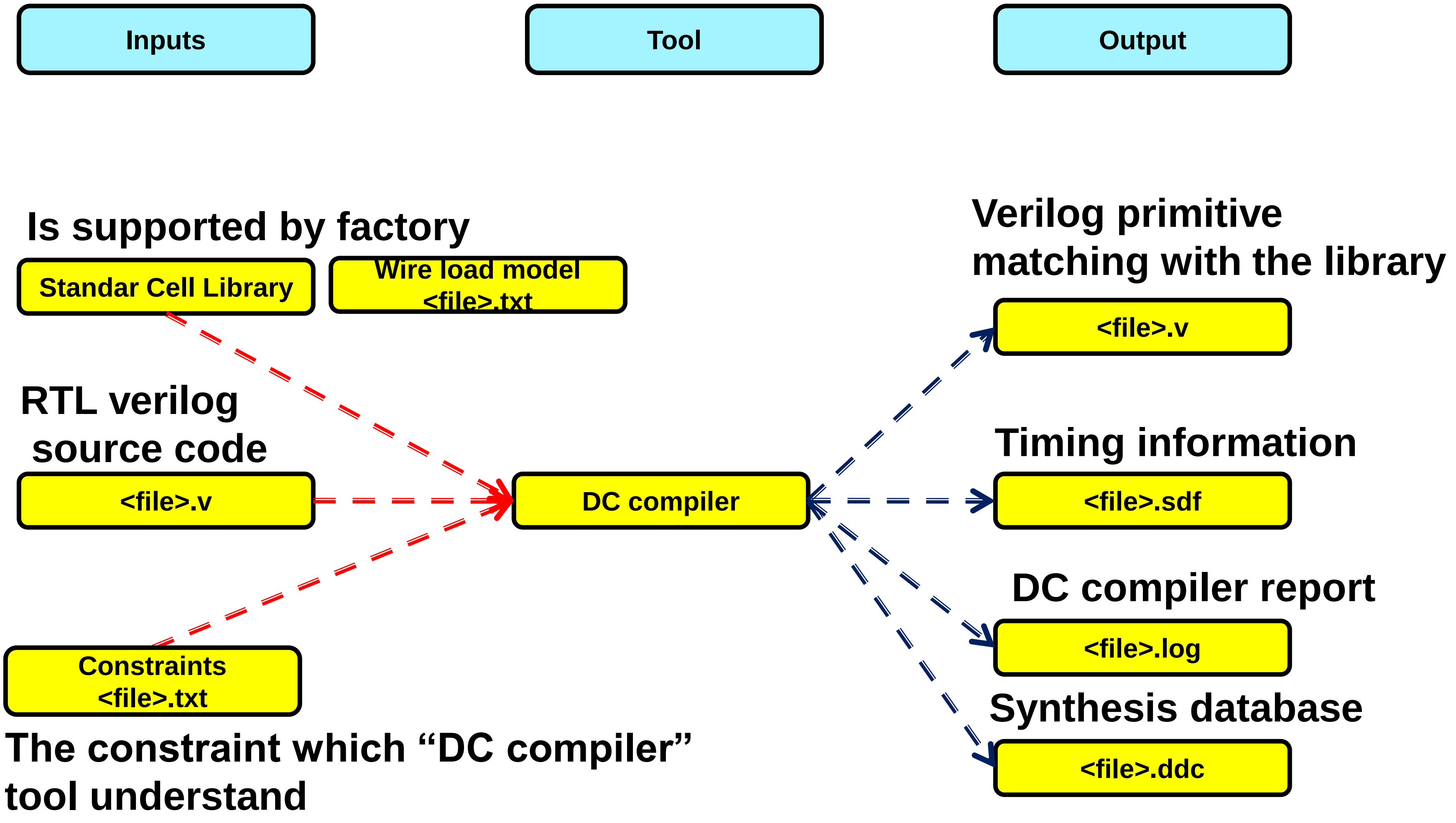
Static Timing Analysis

- Dynamic vs. Static Timing Analysis
- Performance of Design
- **Synthesis step**

Static Timing Analysis

- ❖Dynamic vs. Static Timing Analysis
- ❖Synthesis Step
  - Synthesize RTL to gate level netlist
  - Estimate performance of design approximately
- ❖Analysis
- ❖Constraint
- ❖Violation
- ❖Using DC Tool in Command/GUI mode

Synthesis step



Inputs

Tool

Output

Is supported by factory

Standar Cell Library

Wire load model <file>.txt

RTL verilog source code

<file>.v

Constraints <file>.txt

The constraint which "DC compiler" tool understand

DC compiler

Verilog primitive matching with the library

<file>.v

Timing information

<file>.sdf

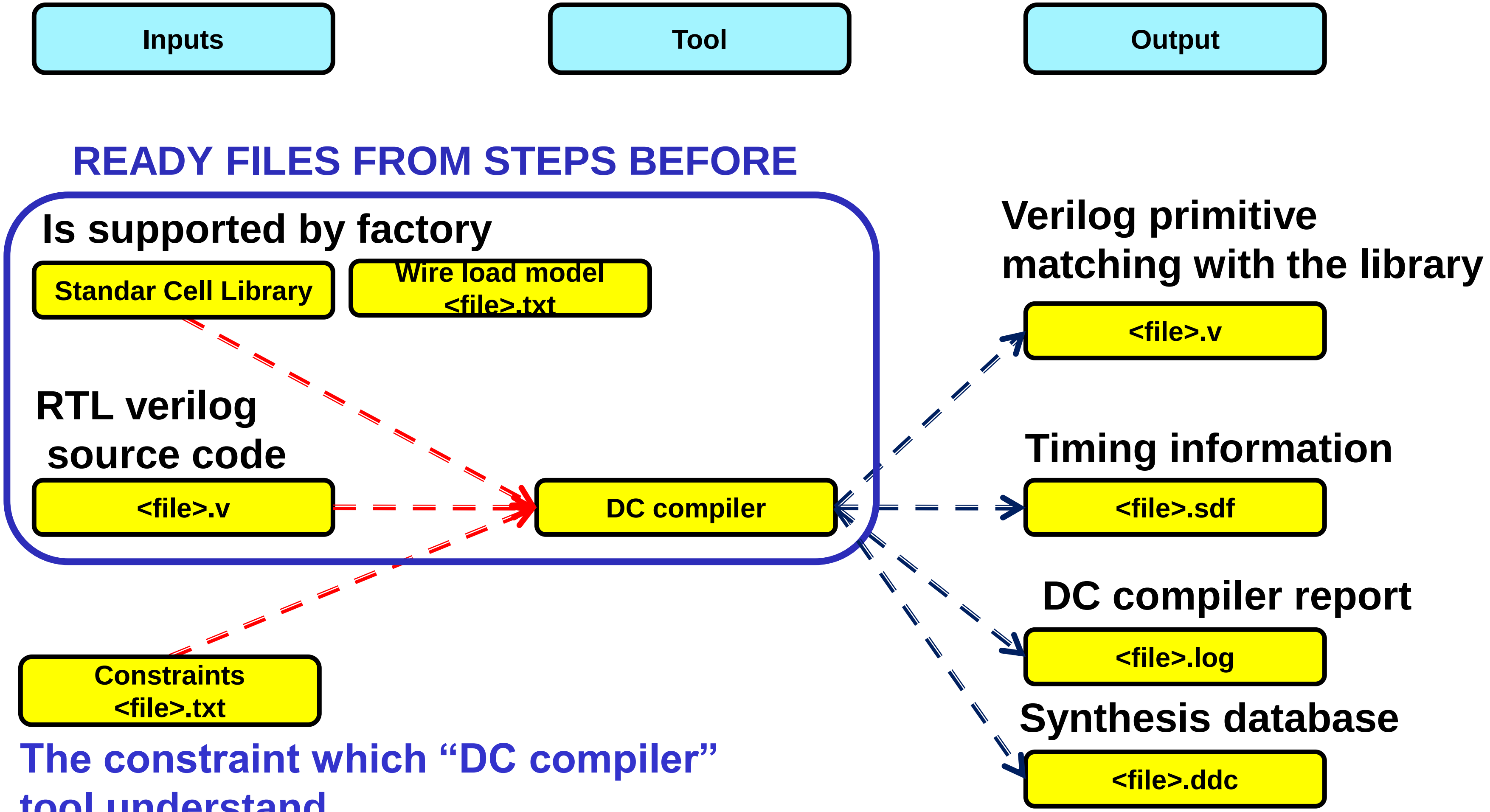
DC compiler report

<file>.log

Synthesis database

<file>.ddc

Synthesis step



Inputs

Tool

Output

READY FILES FROM STEPS BEFORE

Is supported by factory

Standar Cell Library

Wire load model <file>.txt

RTL verilog source code

<file>.v

DC compiler

Constraints <file>.txt

Verilog primitive matching with the library

<file>.v

Timing information

<file>.sdf

DC compiler report

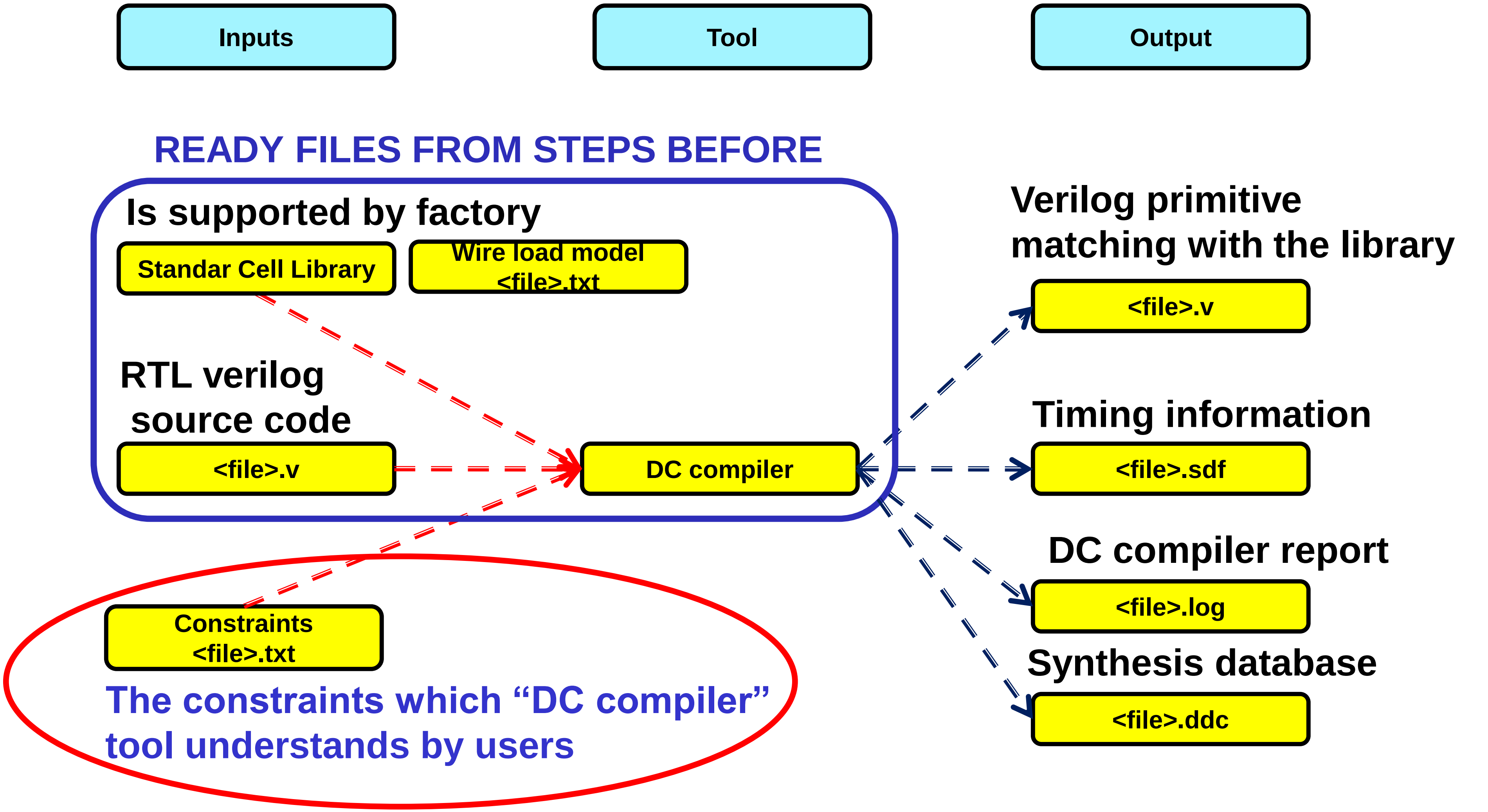
<file>.log

Synthesis database

<file>.ddc

The constraint which “DC compiler” tool understand

Synthesis step



Inputs

Tool

Output

READY FILES FROM STEPS BEFORE

Is supported by factory

Standar Cell Library

Wire load model <file>.txt

RTL verilog source code

<file>.v

DC compiler

Constraints <file>.txt

The constraints which “DC compiler” tool understands by users

Verilog primitive matching with the library

<file>.v

Timing information

<file>.sdf

DC compiler report

<file>.log

Synthesis database

<file>.ddc

Q1. WHICH FEATURES ARE CONSTRAINED IN CIRCUIT ?

Q2. HOW TO COMPOSE THE CONSTRAINS ?

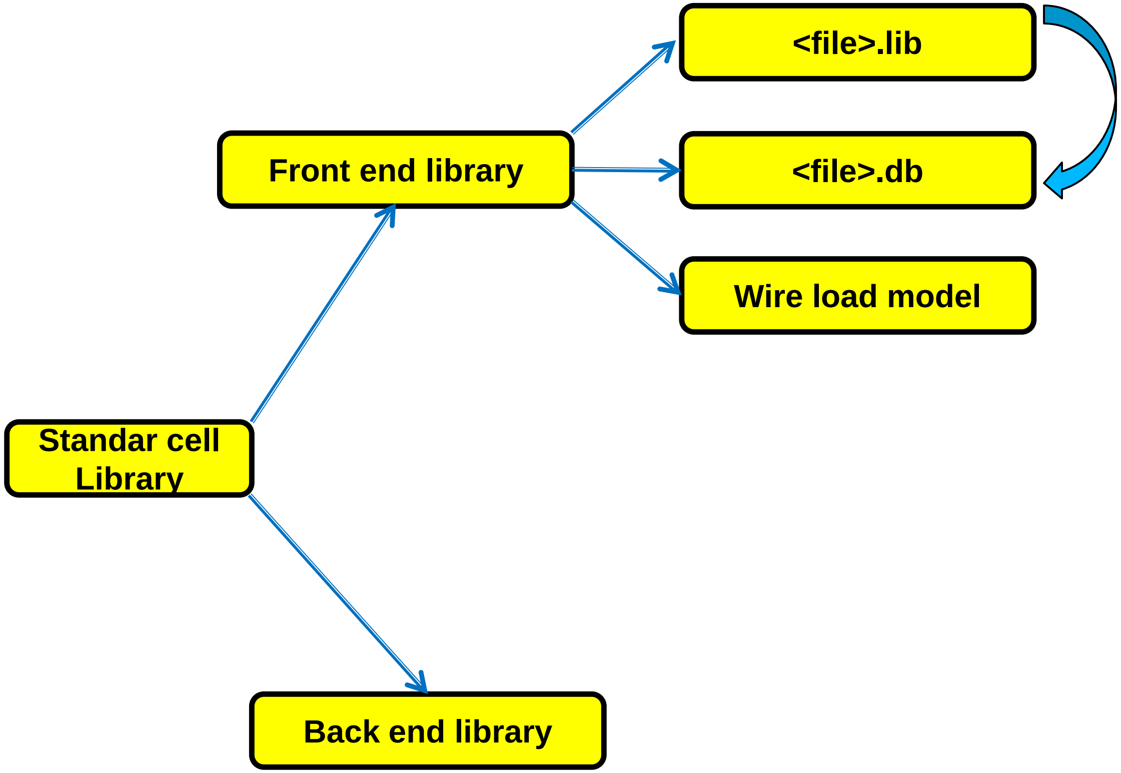
Static Timing Analysis

- **Library – file.lib**

- **Wire load model**

Synthesis step

Library



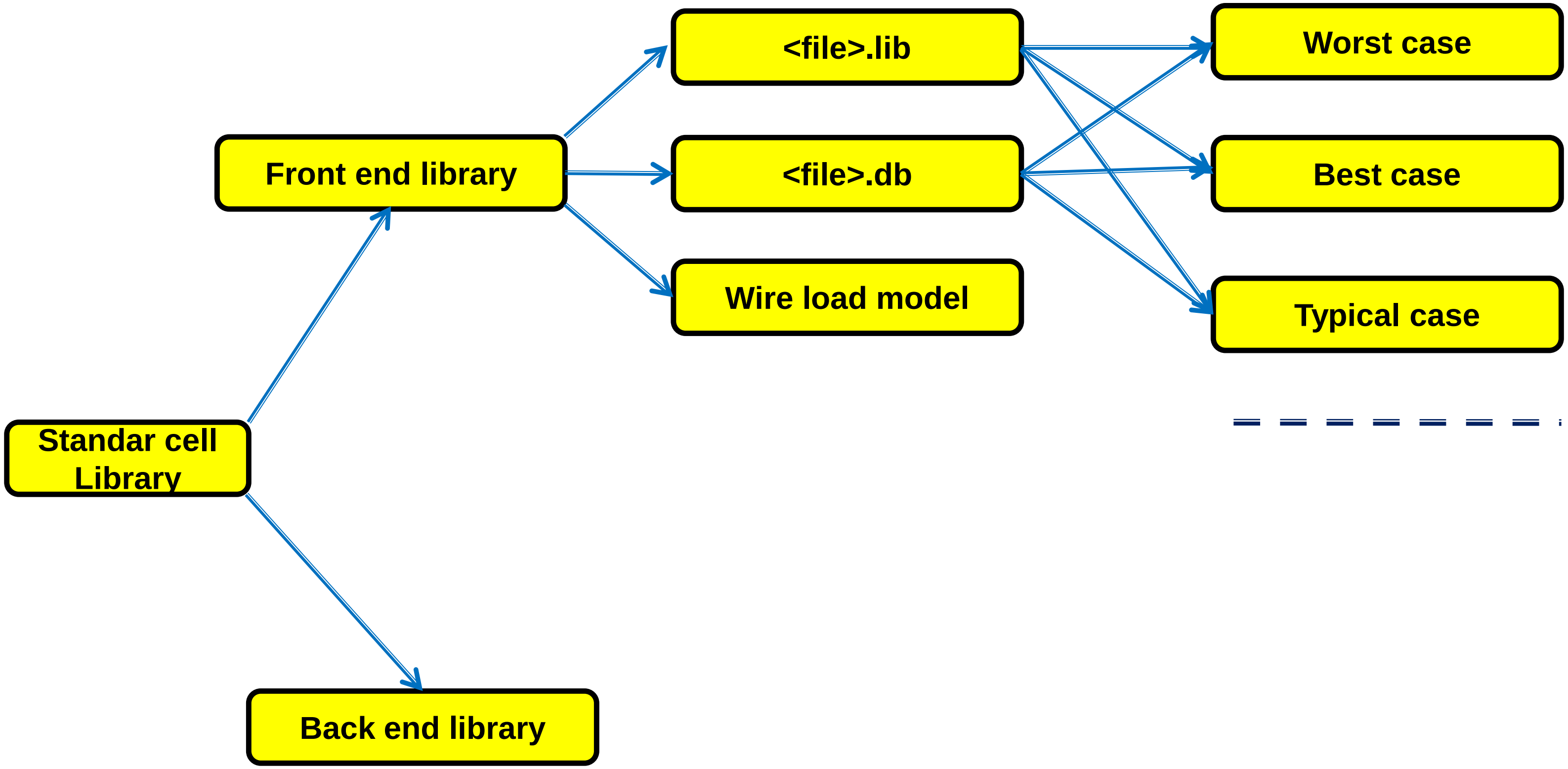
DC tool

- ➢ read_lib … ?
- ➢ write_lib … ?

read_lib ./path/library_file_name.lib (ex: TSM018AG_AASW.lib)
write_lib library_name -f db -o ./path/library_file_name.db

Synthesis step

Library



Standar cell Library

Front end library

Back end library

<file>.lib

<file>.db

Wire load model

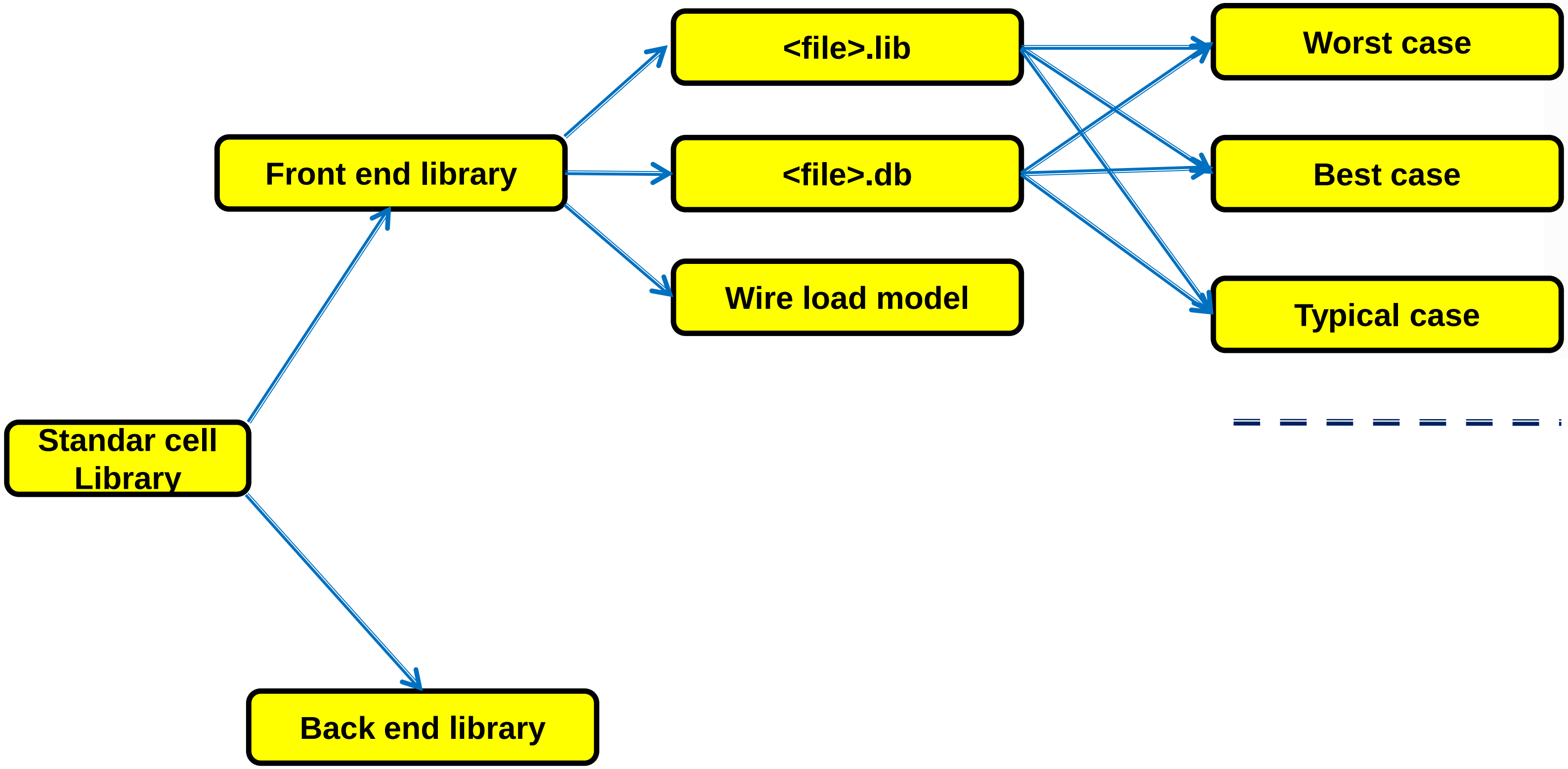
Worst case

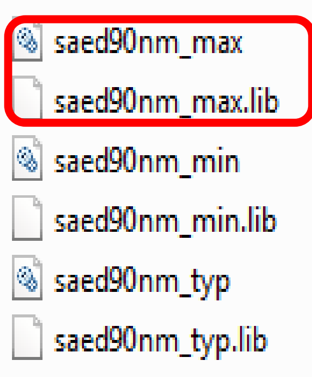
Best case

Typical case

Synthesis step

Library

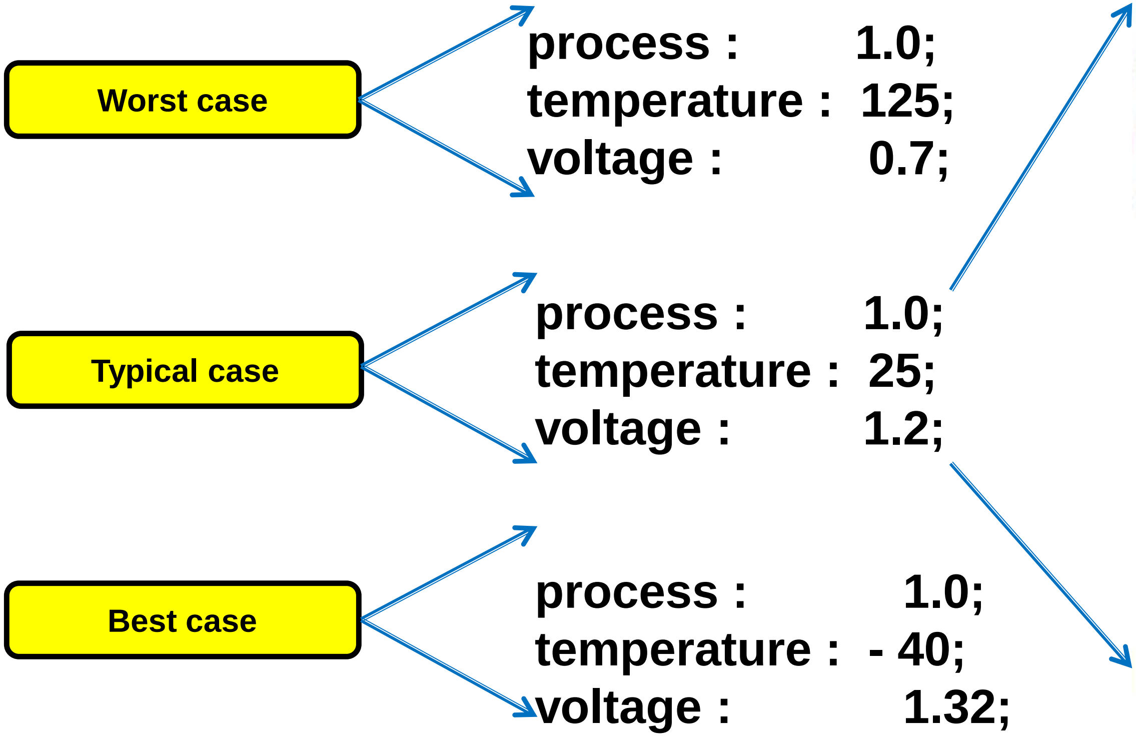




saed90nm_max
saed90nm_max.lib
saed90nm_min
saed90nm_min.lib
saed90nm_typ
saed90nm_typ.lib

Synthesis step

Library



Worst case

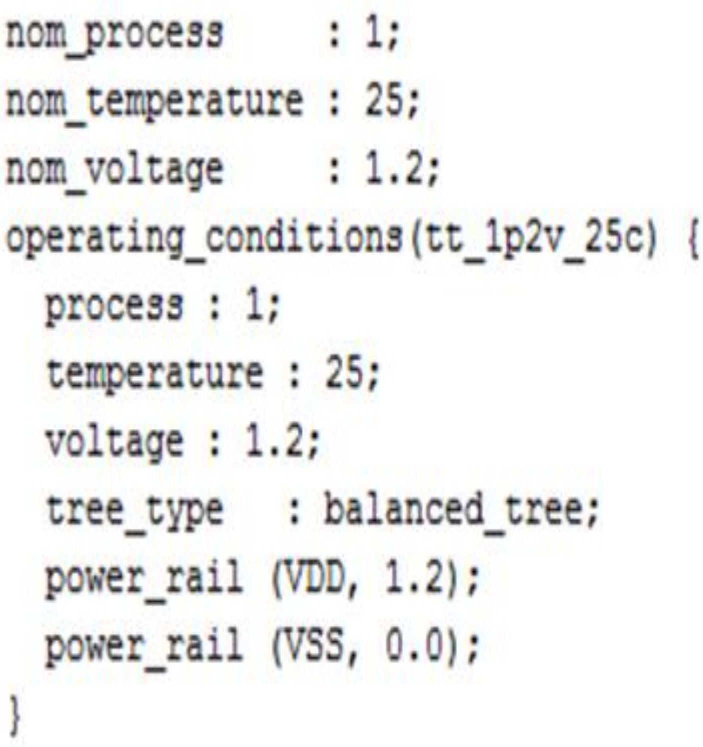
process : 1.0;
temperature : 125;
voltage : 0.7;

Typical case

process : 1.0;
temperature : 25;
voltage : 1.2;

Best case

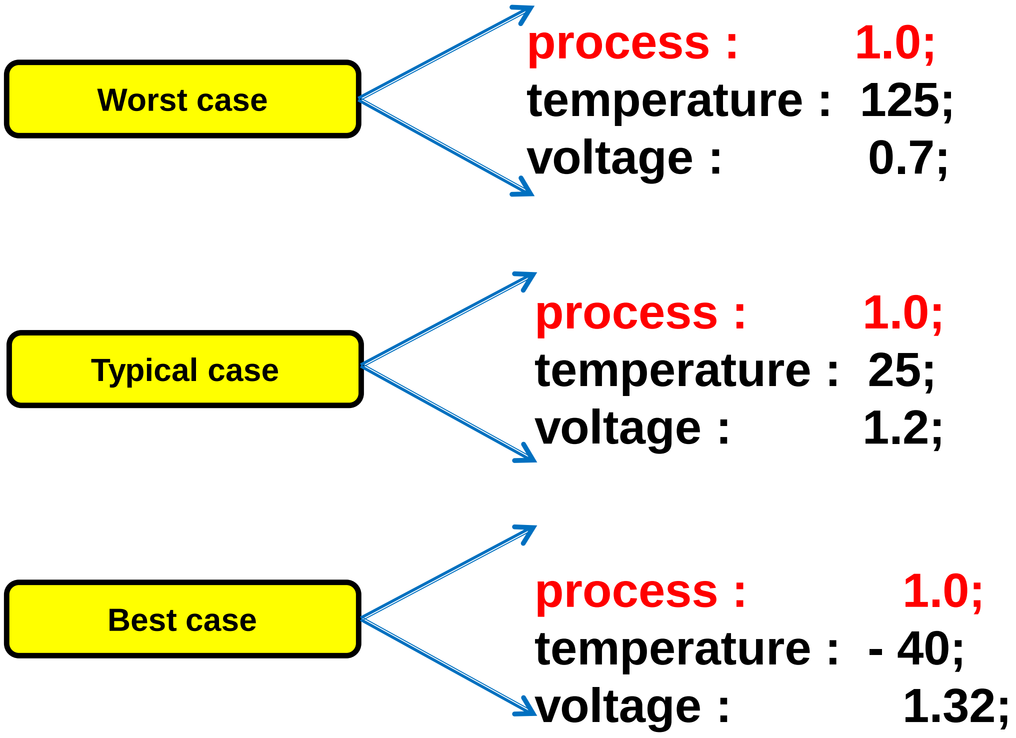
process : 1.0;
temperature : - 40;
voltage : 1.32;



```
nom_process      : 1;
nom_temperature : 25;
nom_voltage      : 1.2;
operating_conditions(tt_1p2v_25c) {
  process : 1;
  temperature : 25;
  voltage : 1.2;
  tree_type    : balanced_tree;
  power_rail  (VDD, 1.2);
  power_rail  (VSS, 0.0);
}
```

Synthesis step

Library



Worst case

process : 1.0;
temperature : 125;
voltage : 0.7;

Typical case

process : 1.0;
temperature : 25;
voltage : 1.2;

Best case

process : 1.0;
temperature : - 40;
voltage : 1.32;

Process

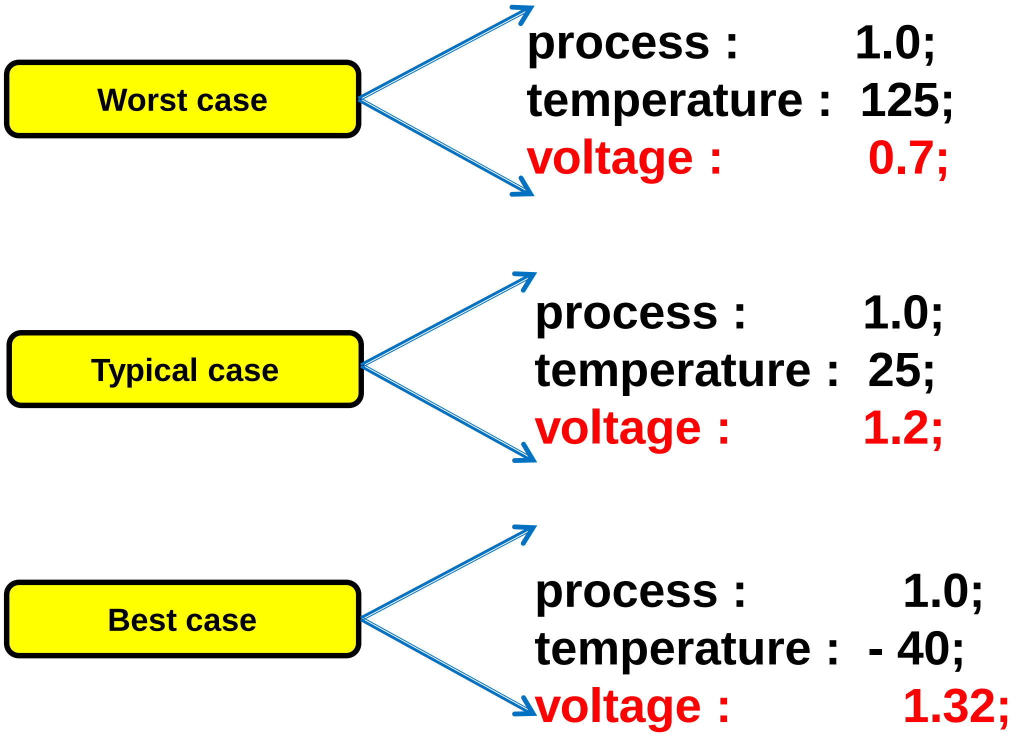
Library

This variation accounts for deviations in the semiconductor fabrication process. Usually process variation is treated **as a percentage variation in the performance calculation**. Variations in the process parameters can be impurity concentration densities, oxide thicknesses and diffusion depths. These are caused bye non uniform conditions during depositions and/or during diffusions of the impurities. This introduces variations in the sheet resistance and transistor parameters such as threshold voltage. Variations are in the dimensions of the devices, mainly resulting from the limited resolution of the photolithographic process. This causes (W/L) variations in MOS transistors.

Process variations are due to variations in the manufacture conditions such as temperature, pressure and dopant concentrations. The ICs are produced in lots of 50 to 200 wafers with approximately 100 dice per wafer. The electrical properties in different lots can be very different. There are also slighter differences in each lot, even in a single manufactured chip. There are variations in the process parameter throughout a whole chip. As a consequence, the transistors have different transistor lengths throughout the chip. This makes the propagation delay to be different everywhere in a chip, because a smaller transistor is faster and therefore the propagation delay is smaller.

Synthesis step

Library



Worst case

process : 1.0;
temperature : 125;
voltage : 0.7;

Typical case

process : 1.0;
temperature : 25;
voltage : 1.2;

Best case

process : 1.0;
temperature : - 40;
voltage : 1.32;

Voltage

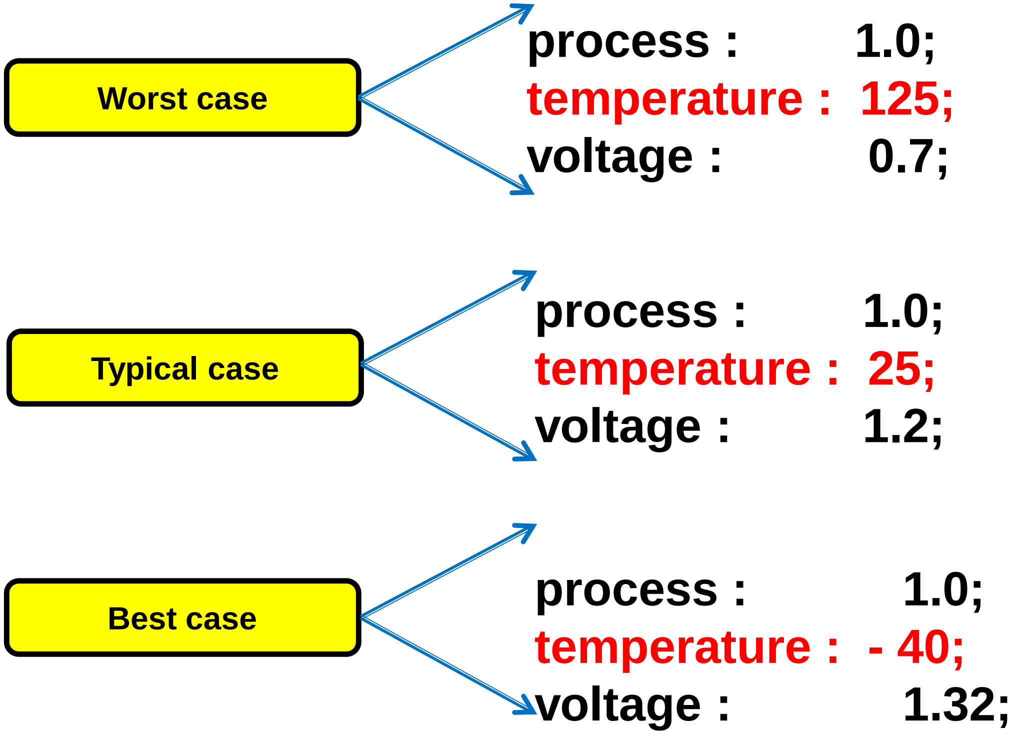
Library

The design's supply voltage can vary from the established ideal value during day-to-day operation. Often a complex calculation (using a shift in threshold voltages) is employed, but a simple linear scaling factor is also used for logic-level performance calculations.

The saturation current of a cell depends on the power supply. The delay of a cell is dependent on the saturation current. In this way, the power supply inflects the propagation delay of a cell. Throughout a chip, the power supply is not constant and hence the propagation delay varies in a chip. The voltage drop is due to nonzero resistance in the supply wires. A higher voltage makes a cell faster and hence the propagation delay is reduced. The decrease is exponential for a wide voltage range. The self-inductance of a supply line contributes also to a voltage drop. For example, when a transistor is switching to high, it takes a current to charge up the output load. This time varying current (for a short period of time) causes an opposite self-induced electromotive force. The amplitude of the voltage drop is given by .V=L*dI/dt, where L is the self inductance and I is the current through the line.

Synthesis step

Library



Worst case

process : 1.0;
temperature : 125;
voltage : 0.7;

Typical case

process : 1.0;
temperature : 25;
voltage : 1.2;

Best case

process : 1.0;
temperature : - 40;
voltage : 1.32;

Temperature

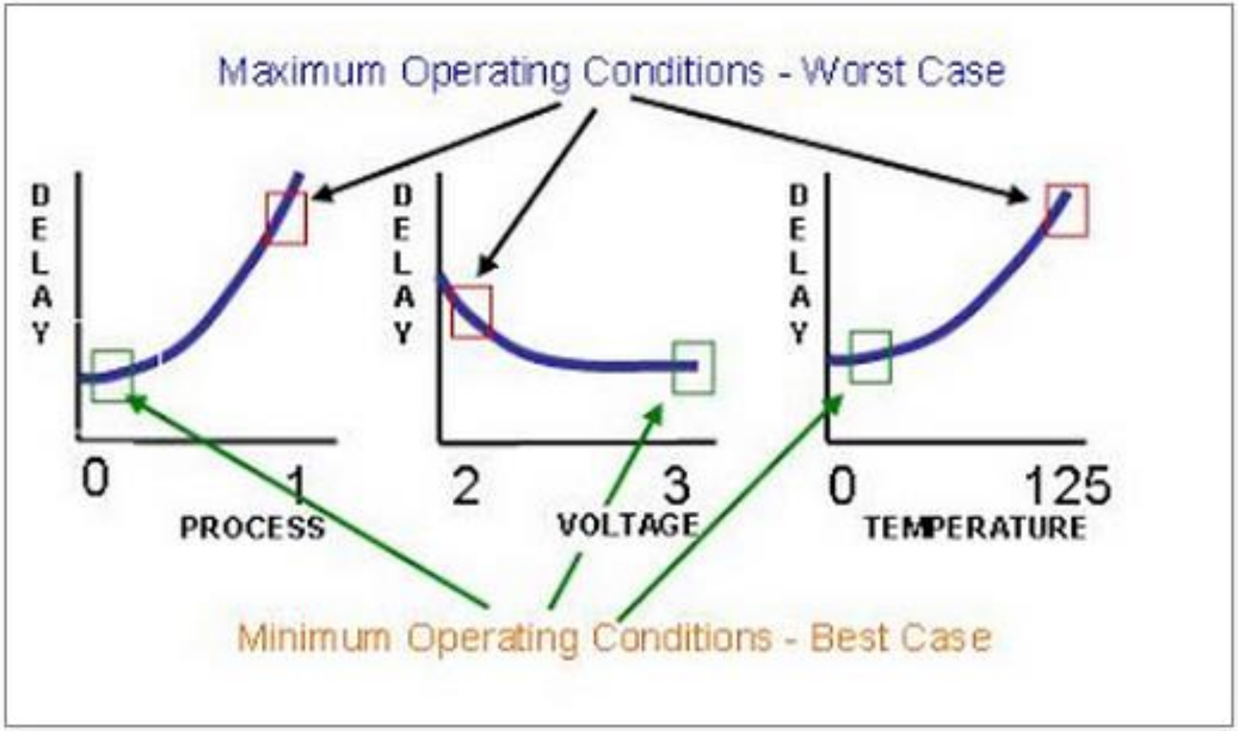
Library

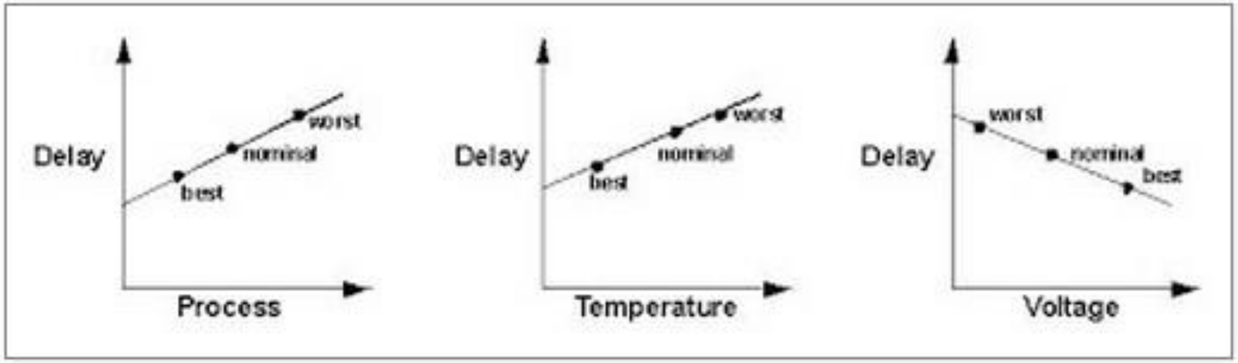
Temperature variation is unavoidable in the everyday operation of a design. Effects on performance caused by temperature fluctuations are most often handled as linear scaling effects, but some submicron silicon processes require nonlinear calculations.

When a chip is operating, the temperature can vary throughout the chip. This is due to the power dissipation in the MOS-transistors. The power consumption is mainly due to switching, short-circuit and leakage power consumption. The average switching power dissipation (approximately given by Paverage = Cload*Vpower supply 2*fclock) is due to the required energy to charge up the parasitic and load capacitances. The short-circuit power dissipation is due to the finite rise and fall times. The nMOS and pMOS transistors may conduct for a short time during switching, forming a direct current from the power supply to the ground. The leakage power consumption is due to the nonzero reverse leakage and sub-threshold currents. The biggest contribution to the power consumption is the switching. The dissipated power will increase the surrounding temperature. The electron and hole mobility depend on the temperature. The mobility (in Si) decreases with increased temperature for temperatures above –50 °C. The temperature, when the mobility starts to decrease, depends on the doping concentration. A starting temperature at –50 °C is true for doping concentrations below 1019 atoms/cm3. For higher doping concentrations, the starting temperature is higher. When the electrons and holes move slower, then the propagation delay increases. Hence, the propagation delay increases with increased temperature. There is also a temperature effect, which has not been considered. The threshold voltage of a transistor depends on the temperature. A higher temperature will decrease the threshold voltage. A lower threshold voltage means a higher current and therefore a better delay performance. This effect depends extremely on power supply, threshold voltage, load and input slope of a cell. There is a competition between the two effects and generally the mobility effect wins

Operation Condition

Library

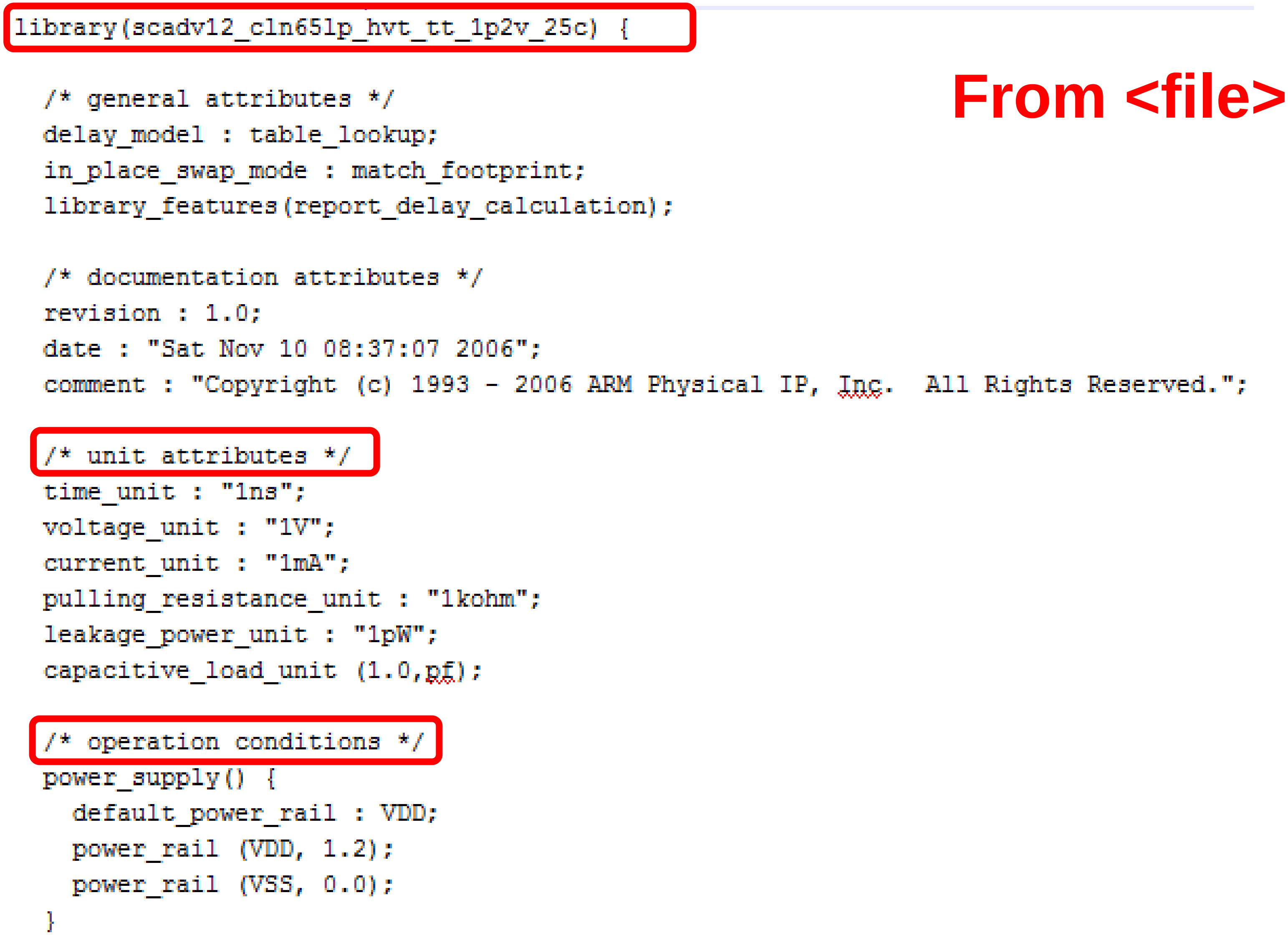




Library – file.lib

<file>.lib

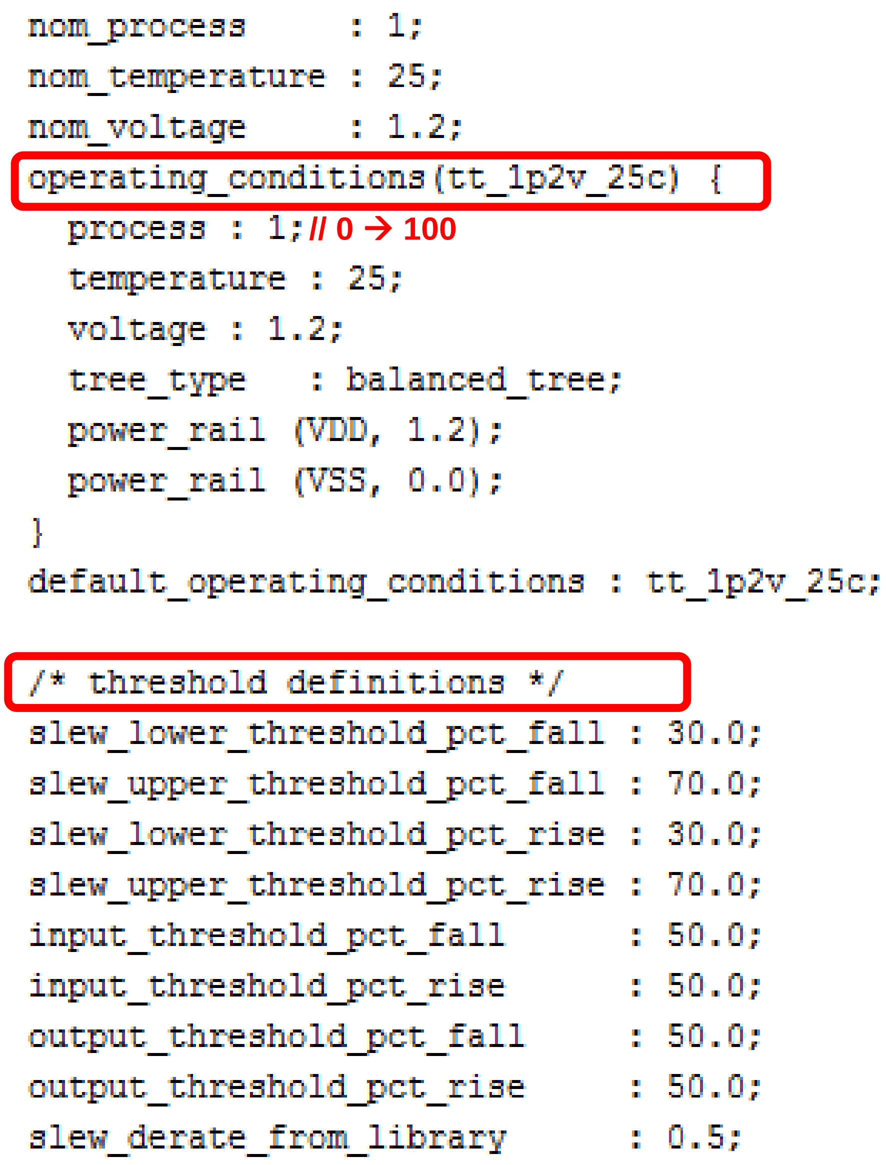
From <file>.lib



```
library(scadv12_cln65lp_hvt_tt_1p2v_25c) {

  /* general attributes */
  delay_model : table_lookup;
  in_place_swap_mode : match_footprint;
  library_features(report_delay_calculation);

  /* documentation attributes */
  revision : 1.0;
  date : "Sat Nov 10 08:37:07 2006";
  comment : "Copyright (c) 1993 - 2006 ARM Physical IP, Inc.  All Rights Reserved.";

  /* unit attributes */
  time_unit : "1ns";
  voltage_unit : "1V";
  current_unit : "1mA";
  pulling_resistance_unit : "1kohm";
  leakage_power_unit : "1pW";
  capacitive_load_unit (1.0,pf);

  /* operation conditions */
  power_supply() {
    default_power_rail : VDD;
    power_rail (VDD, 1.2);
    power_rail (VSS, 0.0);
  }
```

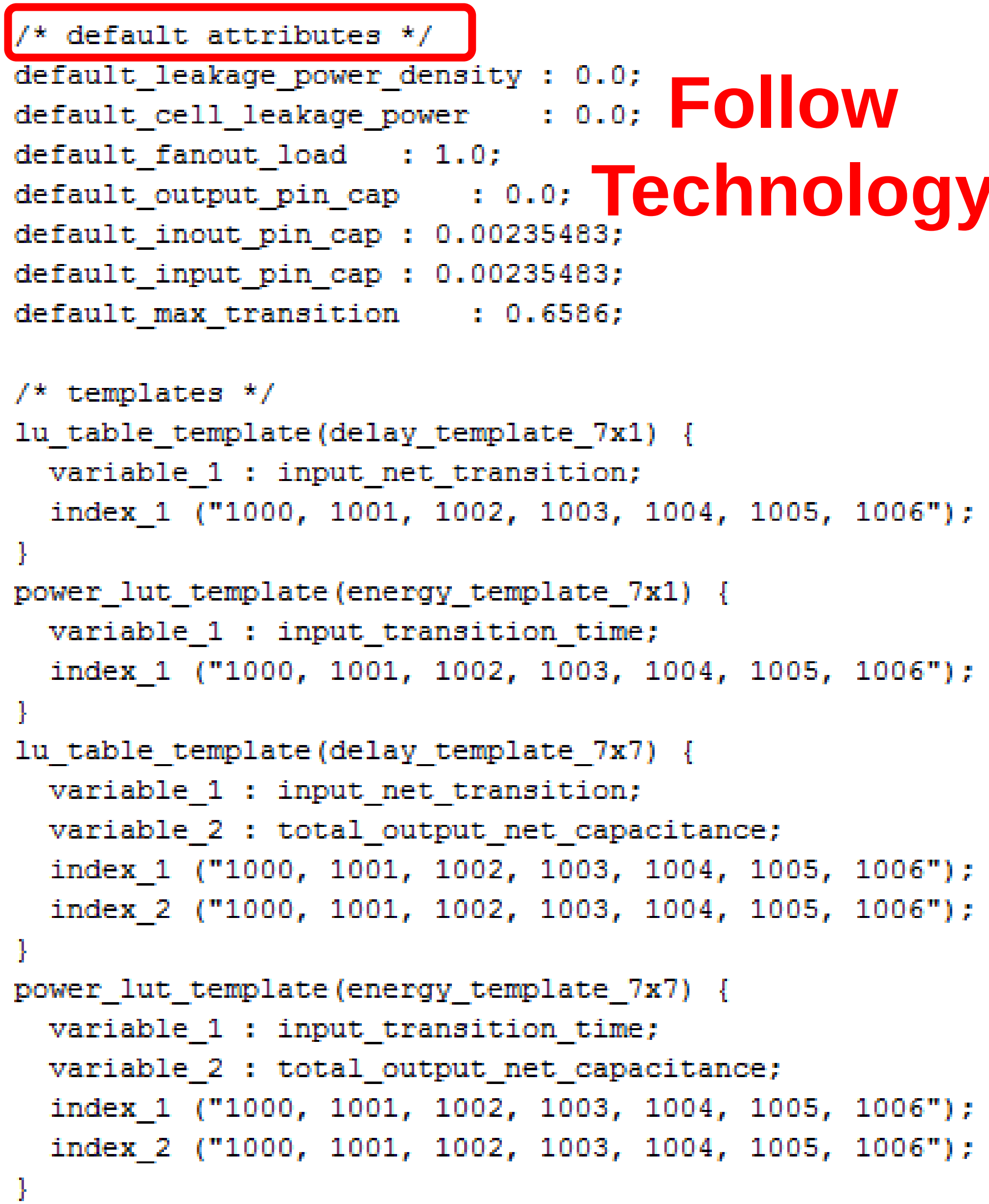
Library – file.lib

<file>.lib



```
nom_process      : 1;
nom_temperature : 25;
nom_voltage      : 1.2;
operating_conditions(tt_1p2v_25c) {
  process : 1;
  temperature : 25;
  voltage : 1.2;
  tree_type    : balanced_tree;
  power_rail (VDD, 1.2);
  power_rail (VSS, 0.0);
}
default_operating_conditions : tt_1p2v_25c;

/* threshold definitions */
slew_lower_threshold_pct_fall : 30.0;
slew_upper_threshold_pct_fall : 70.0;
slew_lower_threshold_pct_rise : 30.0;
slew_upper_threshold_pct_rise : 70.0;
input_threshold_pct_fall       : 50.0;
input_threshold_pct_rise       : 50.0;
output_threshold_pct_fall      : 50.0;
output_threshold_pct_rise      : 50.0;
slew_derate_from_library       : 0.5;
```

// 0 → 100



```
/* default attributes */
default_leakage_power_density : 0.0;
default_cell_leakage_power     : 0.0;
default_fanout_load     : 1.0;
default_output_pin_cap      : 0.0;
default_inout_pin_cap : 0.00235483;
default_input_pin_cap : 0.00235483;
default_max_transition      : 0.6586;

/* templates */
lu_table_template(delay_template_7x1) {
  variable_1 : input_net_transition;
  index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
}
power_lut_template(energy_template_7x1) {
  variable_1 : input_transition_time;
  index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
}
lu_table_template(delay_template_7x7) {
  variable_1 : input_net_transition;
  variable_2 : total_output_net_capacitance;
  index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
  index_2 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
}
power_lut_template(energy_template_7x7) {
  variable_1 : input_transition_time;
  variable_2 : total_output_net_capacitance;
  index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
  index_2 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
}
```

Follow Technology

Library – file.lib

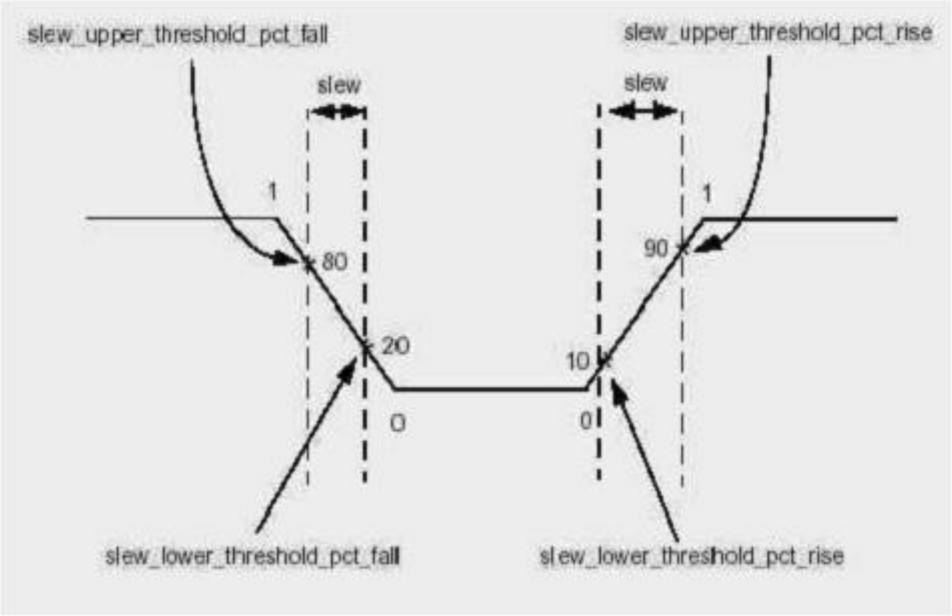
<file>.lib

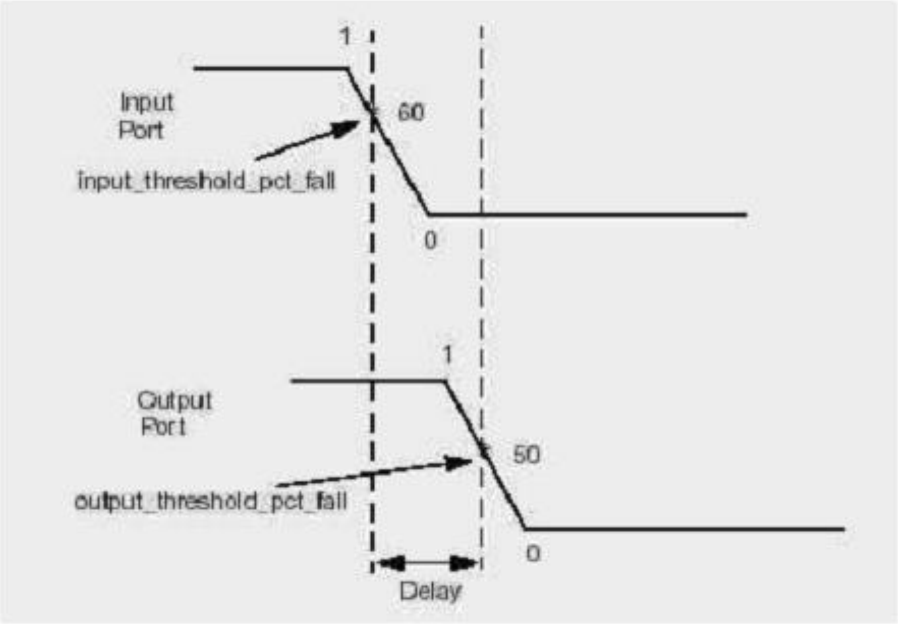
Syntax:

slew_lower_threshold_pct_rise : trip_point;
slew_upper_threshold_pct_rise : trip_point;
slew_lower_threshold_pct_fall : trip_point;
slew_upper_threshold_pct_fall : trip_point;

Example:

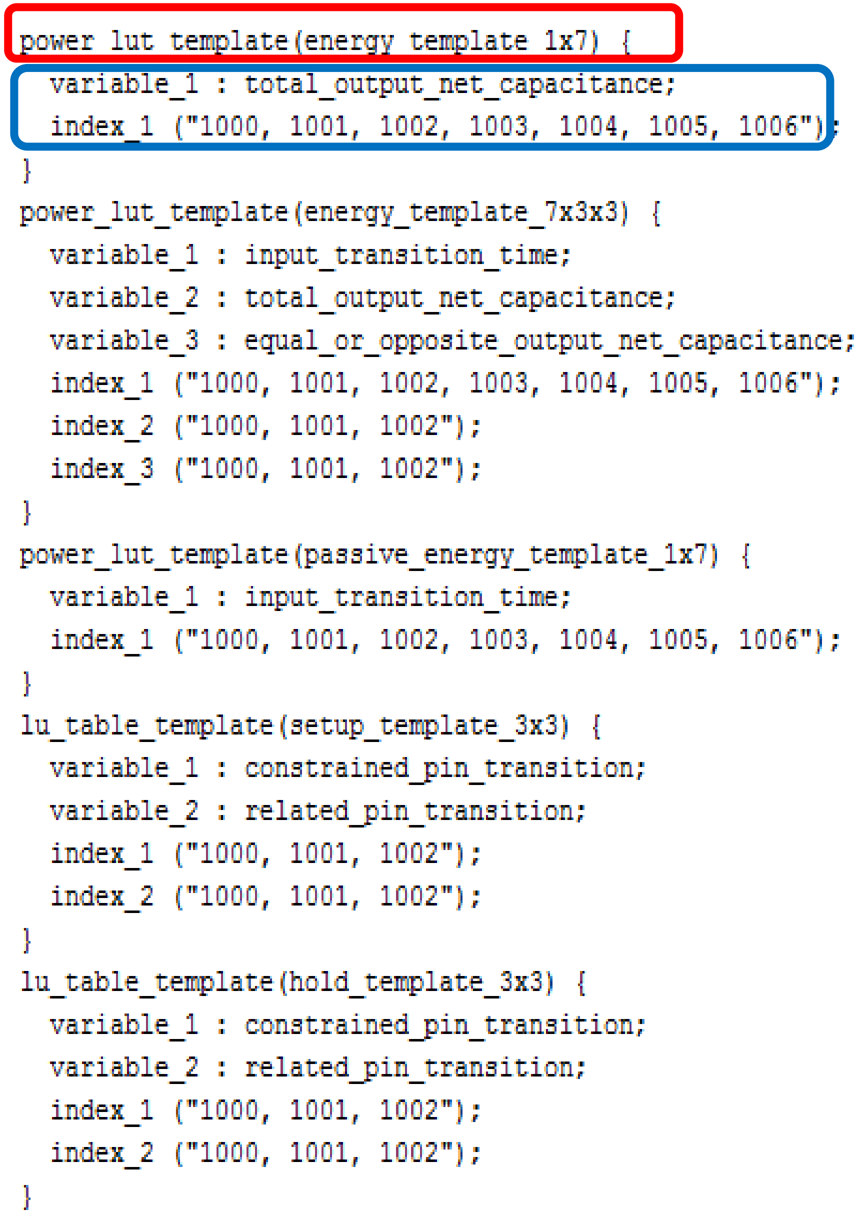
slew_lower_threshold_pct_rise : 10.00;
slew_upper_threshold_pct_rise : 90.00;
slew_lower_threshold_pct_fall : 10.00;
slew_upper_threshold_pct_fall : 90.00;


slew_upper_threshold_pct_fall
slew
slew
slew_upper_threshold_pct_rise
1
80
20
0
1
90
10
0
slew_lower_threshold_pct_fall
slew_lower_threshold_pct_rise

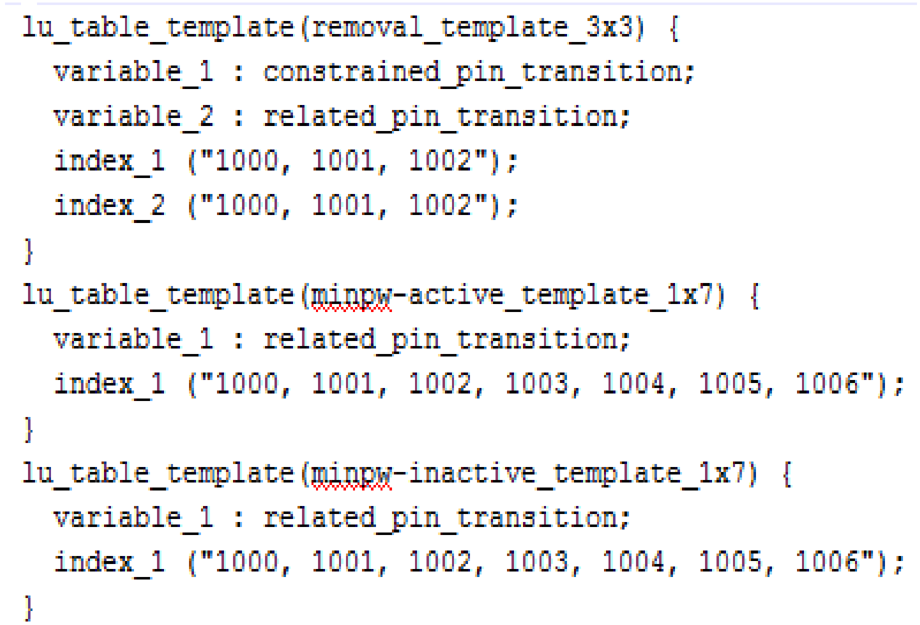


Library – file.lib

<file>.lib



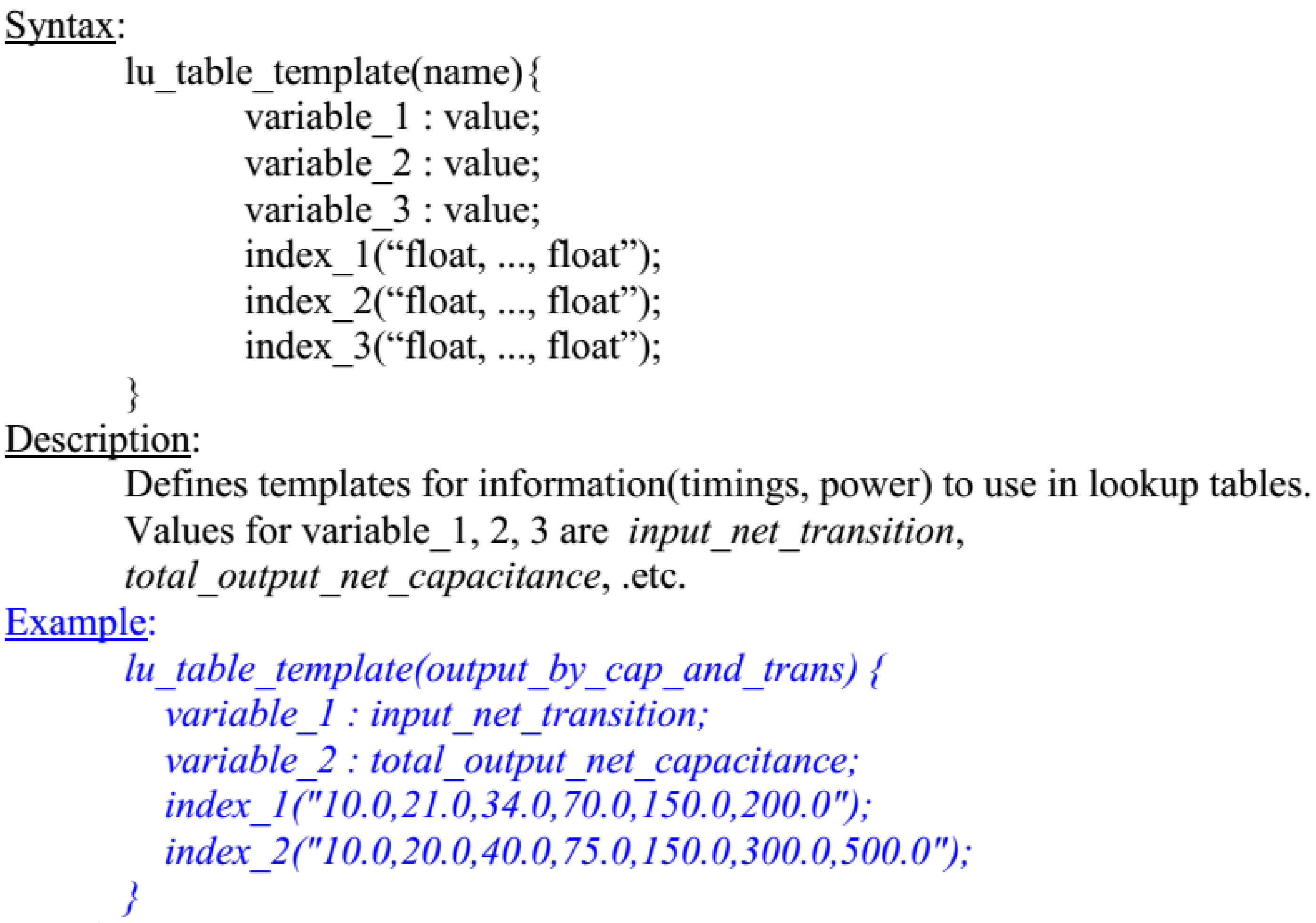
```
power_lut_template(energy_template_1x7) {
  variable_1 : total_output_net_capacitance;
  index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
}
power_lut_template(energy_template_7x3x3) {
  variable_1 : input_transition_time;
  variable_2 : total_output_net_capacitance;
  variable_3 : equal_or_opposite_output_net_capacitance;
  index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
  index_2 ("1000, 1001, 1002");
  index_3 ("1000, 1001, 1002");
}
power_lut_template(passive_energy_template_1x7) {
  variable_1 : input_transition_time;
  index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
}
lu_table_template(setup_template_3x3) {
  variable_1 : constrained_pin_transition;
  variable_2 : related_pin_transition;
  index_1 ("1000, 1001, 1002");
  index_2 ("1000, 1001, 1002");
}
lu_table_template(hold_template_3x3) {
  variable_1 : constrained_pin_transition;
  variable_2 : related_pin_transition;
  index_1 ("1000, 1001, 1002");
  index_2 ("1000, 1001, 1002");
}
```



```
lu_table_template(removal_template_3x3) {
  variable_1 : constrained_pin_transition;
  variable_2 : related_pin_transition;
  index_1 ("1000, 1001, 1002");
  index_2 ("1000, 1001, 1002");
}
lu_table_template(minpw-active_template_1x7) {
  variable_1 : related_pin_transition;
  index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
}
lu_table_template(minpw-inactive_template_1x7) {
  variable_1 : related_pin_transition;
  index_1 ("1000, 1001, 1002, 1003, 1004, 1005, 1006");
}
```

Library – file.lib

<file>.lib



Syntax:

```
lu_table_template(name){
        variable_1 : value;
        variable_2 : value;
        variable_3 : value;
        index_1("float, ..., float");
        index_2("float, ..., float");
        index_3("float, ..., float");
}
```

Description:

Defines templates for information(timings, power) to use in lookup tables. Values for variable_1, 2, 3 are *input_net_transition*, *total_output_net_capacitance*, .etc.

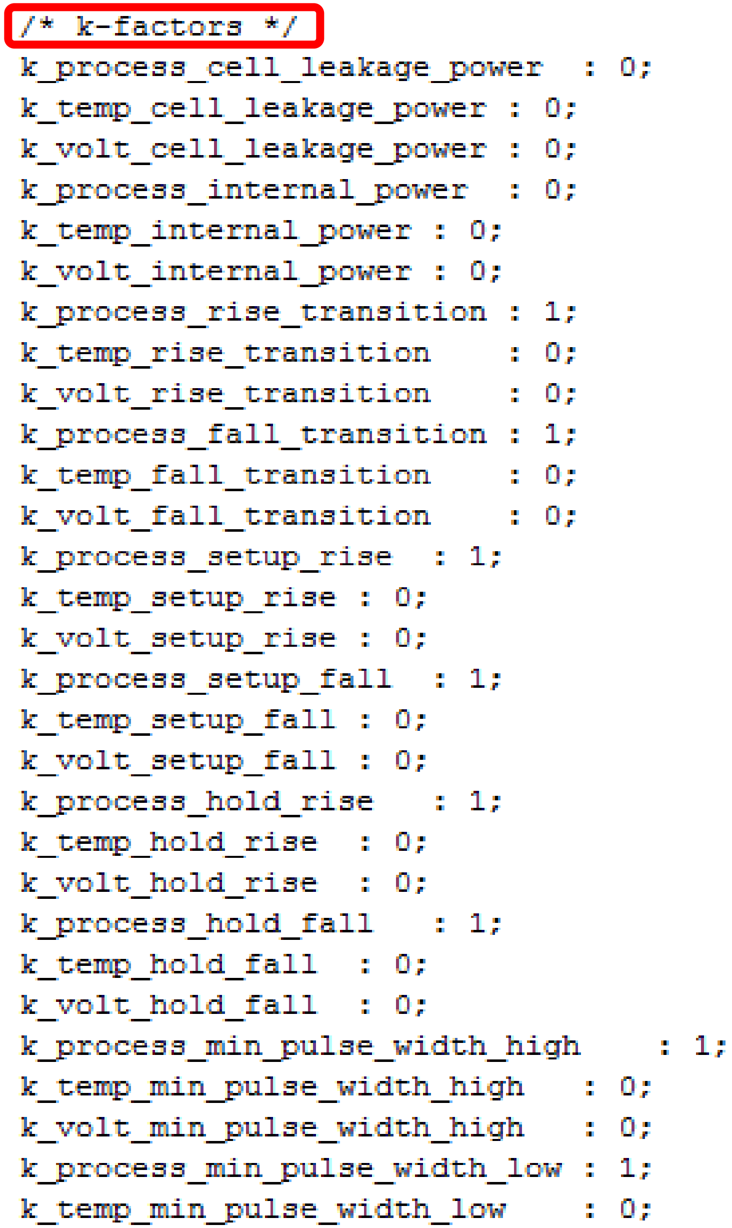
Example:

```
lu_table_template(output_by_cap_and_trans) {
   variable_1 : input_net_transition;
   variable_2 : total_output_net_capacitance;
   index_1("10.0,21.0,34.0,70.0,150.0,200.0");
   index_2("10.0,20.0,40.0,75.0,150.0,300.0,500.0");
}
```

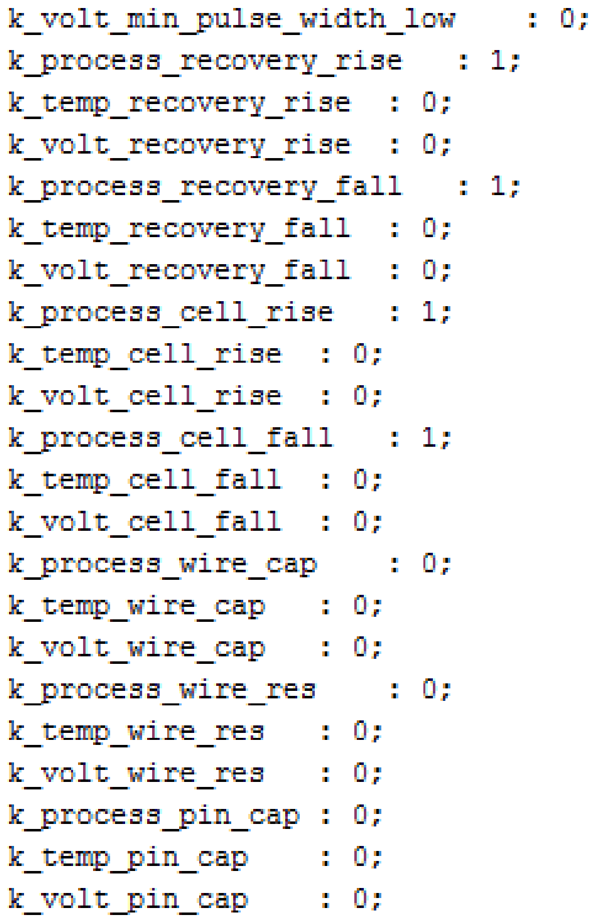
This example means that clock rates for characterization are 10.0 ps, 21.0 ps, 34.0 ps, 70.0 ps, 150.0 ps and 200.0 ps (suppose that time_unit is "ps") and output loading for characterization are 10.0 fF, 20.0 fF, 40.0 fF,75.0 fF,150.0 fF, 300.0 fF and 500.0 fF (suppose *capitive_load_unit (1,fF);*). There are 6x7 = 42 pairs of (clock rate, output load).

Library – file.lib

<file>.lib



```
/* k-factors */
k_process_cell_leakage_power  : 0;
k_temp_cell_leakage_power : 0;
k_volt_cell_leakage_power : 0;
k_process_internal_power  : 0;
k_temp_internal_power : 0;
k_volt_internal_power : 0;
k_process_rise_transition : 1;
k_temp_rise_transition    : 0;
k_volt_rise_transition    : 0;
k_process_fall_transition : 1;
k_temp_fall_transition    : 0;
k_volt_fall_transition    : 0;
k_process_setup_rise  : 1;
k_temp_setup_rise : 0;
k_volt_setup_rise : 0;
k_process_setup_fall  : 1;
k_temp_setup_fall : 0;
k_volt_setup_fall : 0;
k_process_hold_rise   : 1;
k_temp_hold_rise  : 0;
k_volt_hold_rise  : 0;
k_process_hold_fall   : 1;
k_temp_hold_fall  : 0;
k_volt_hold_fall  : 0;
k_process_min_pulse_width_high    : 1;
k_temp_min_pulse_width_high   : 0;
k_volt_min_pulse_width_high   : 0;
k_process_min_pulse_width_low : 1;
k_temp_min_pulse_width_low    : 0;
```



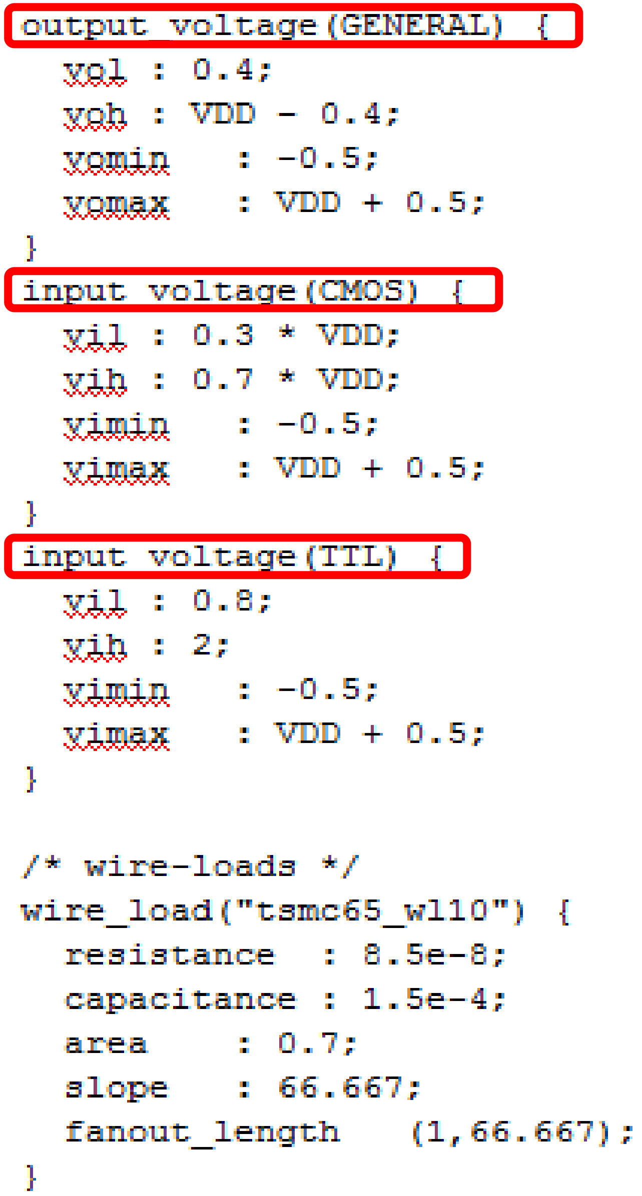
```
k_volt_min_pulse_width_low    : 0;
k_process_recovery_rise    : 1;
k_temp_recovery_rise   : 0;
k_volt_recovery_rise   : 0;
k_process_recovery_fall    : 1;
k_temp_recovery_fall   : 0;
k_volt_recovery_fall   : 0;
k_process_cell_rise    : 1;
k_temp_cell_rise   : 0;
k_volt_cell_rise   : 0;
k_process_cell_fall    : 1;
k_temp_cell_fall   : 0;
k_volt_cell_fall   : 0;
k_process_wire_cap     : 0;
k_temp_wire_cap    : 0;
k_volt_wire_cap    : 0;
k_process_wire_res     : 0;
k_temp_wire_res    : 0;
k_volt_wire_res    : 0;
k_process_pin_cap  : 0;
k_temp_pin_cap     : 0;
k_volt_pin_cap     : 0;
```

For Scale

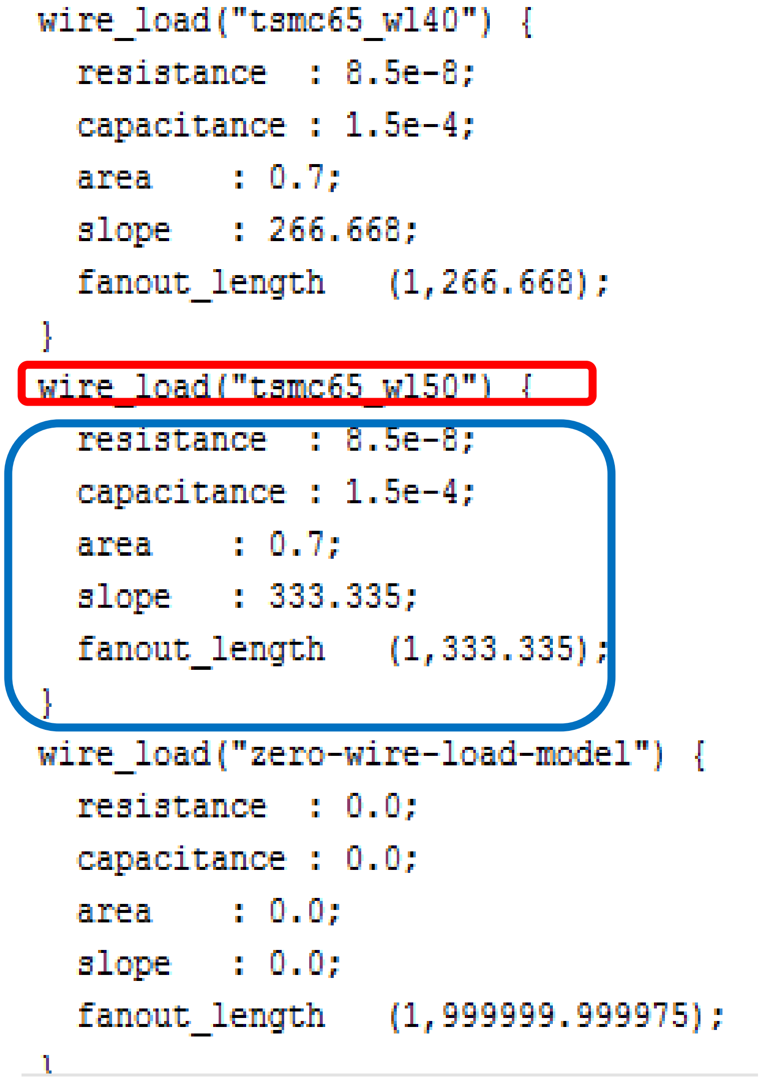
k_process_cell_fall : 0.3702;
→ average of 210 max=0.5230 min=0.2803

Library – file.lib

<file>.lib



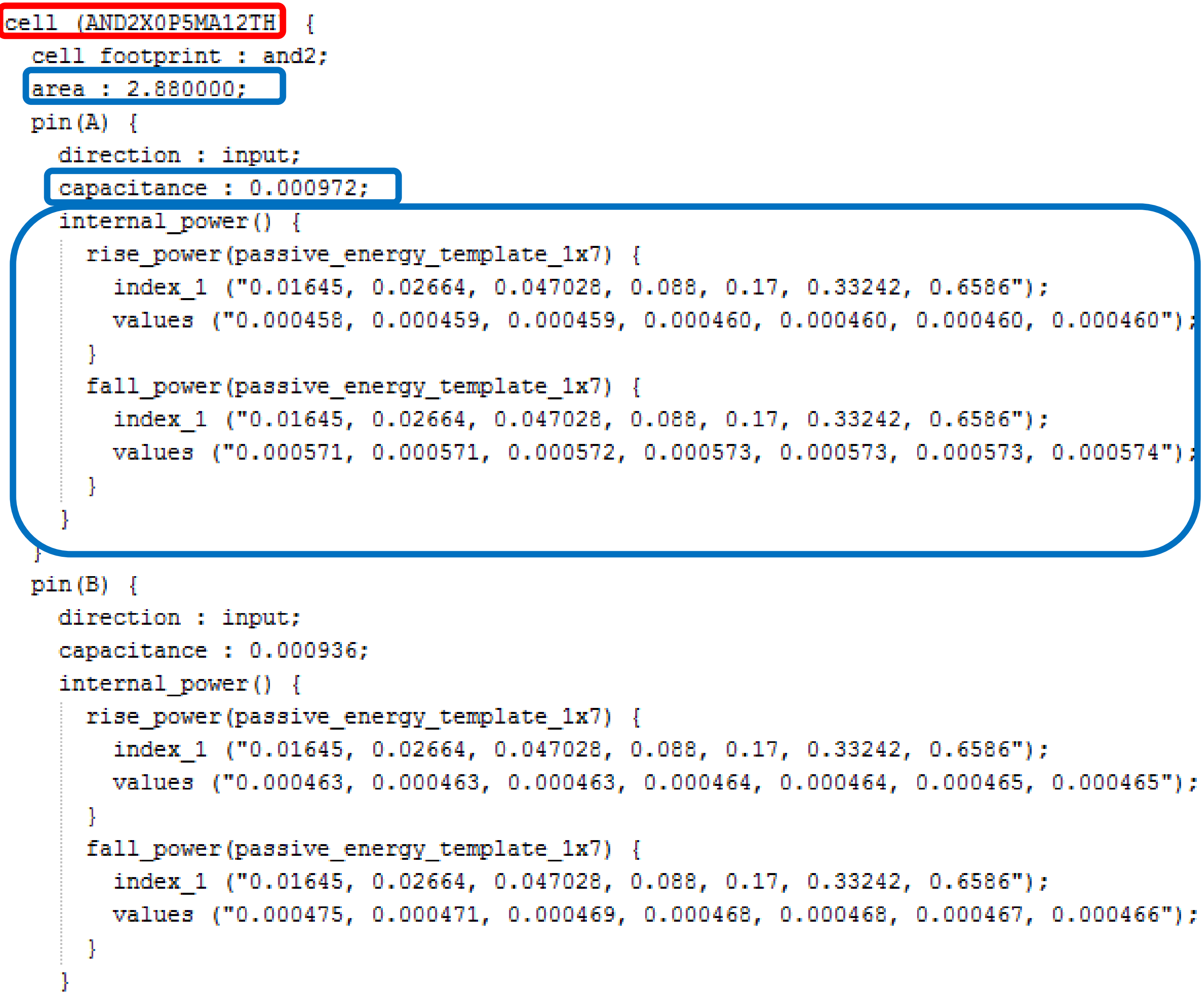
```
output voltage(GENERAL) {
   vol : 0.4;
   voh : VDD - 0.4;
   vomin    : -0.5;
   vomax    : VDD + 0.5;
}
input voltage(CMOS) {
   vil : 0.3 * VDD;
   vih : 0.7 * VDD;
   vimin    : -0.5;
   vimax    : VDD + 0.5;
}
input voltage(TTL) {
   vil : 0.8;
   vih : 2;
   vimin    : -0.5;
   vimax    : VDD + 0.5;
}

/* wire-loads */
wire_load("tsmc65_wl10") {
   resistance   : 8.5e-8;
   capacitance : 1.5e-4;
   area     : 0.7;
   slope    : 66.667;
   fanout_length    (1,66.667);
}
```



```
wire_load("tsmc65_wl40") {
   resistance   : 8.5e-8;
   capacitance : 1.5e-4;
   area     : 0.7;
   slope    : 266.668;
   fanout_length    (1,266.668);
}
wire_load("tsmc65_wl50") {
   resistance   : 8.5e-8;
   capacitance : 1.5e-4;
   area     : 0.7;
   slope    : 333.335;
   fanout_length    (1,333.335);
}
wire_load("zero-wire-load-model") {
   resistance   : 0.0;
   capacitance : 0.0;
   area     : 0.0;
   slope    : 0.0;
   fanout_length    (1,999999.999975);
}
```

Library – file.lib

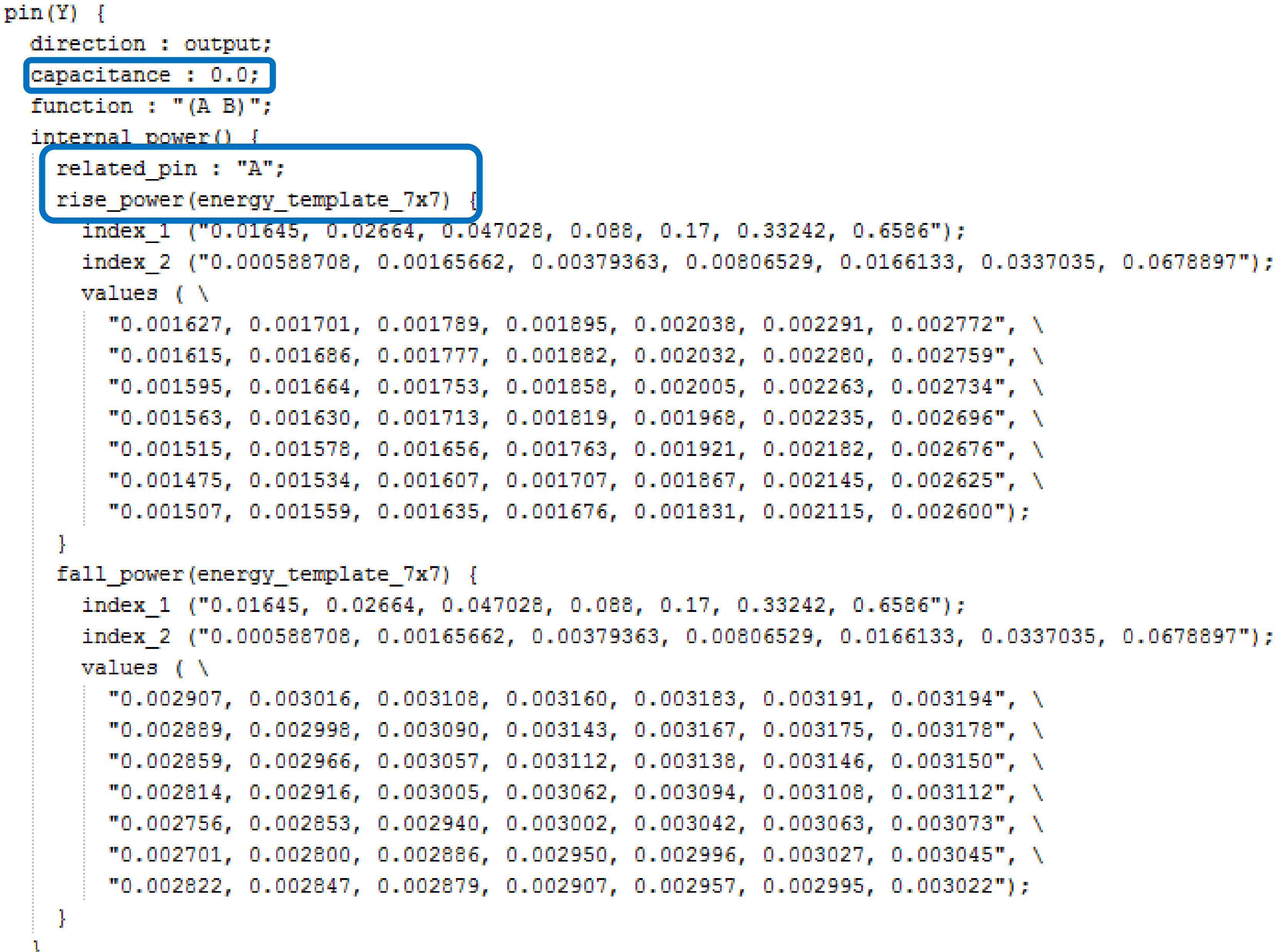
<file>.lib



```
cell (AND2X0P5MA12TH) {
  cell footprint : and2;
  area : 2.880000;
  pin(A) {
    direction : input;
    capacitance : 0.000972;
    internal_power() {
      rise_power(passive_energy_template_1x7) {
        index_1 ("0.01645, 0.02664, 0.047028, 0.088, 0.17, 0.33242, 0.6586");
        values ("0.000458, 0.000459, 0.000459, 0.000460, 0.000460, 0.000460, 0.000460");
      }
      fall_power(passive_energy_template_1x7) {
        index_1 ("0.01645, 0.02664, 0.047028, 0.088, 0.17, 0.33242, 0.6586");
        values ("0.000571, 0.000571, 0.000572, 0.000573, 0.000573, 0.000573, 0.000574");
      }
    }
  }
  pin(B) {
    direction : input;
    capacitance : 0.000936;
    internal_power() {
      rise_power(passive_energy_template_1x7) {
        index_1 ("0.01645, 0.02664, 0.047028, 0.088, 0.17, 0.33242, 0.6586");
        values ("0.000463, 0.000463, 0.000463, 0.000464, 0.000464, 0.000465, 0.000465");
      }
      fall_power(passive_energy_template_1x7) {
        index_1 ("0.01645, 0.02664, 0.047028, 0.088, 0.17, 0.33242, 0.6586");
        values ("0.000475, 0.000471, 0.000469, 0.000468, 0.000468, 0.000467, 0.000466");
      }
    }
```

Library – file.lib

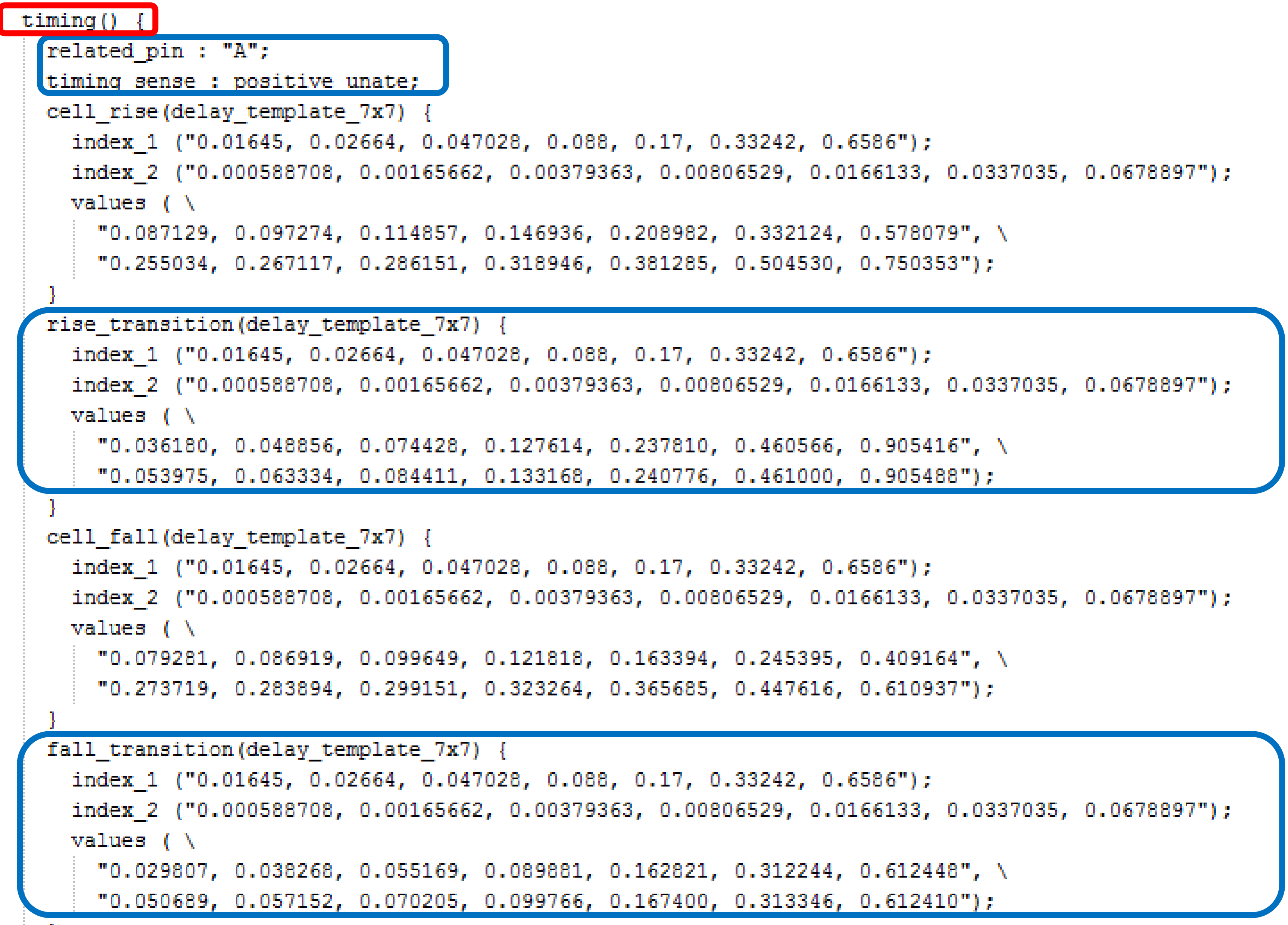
<file>.lib



```
pin(Y) {
  direction : output;
  capacitance : 0.0;
  function : "(A B)";
  internal_power() {
    related_pin : "A";
    rise_power(energy_template_7x7) {
      index_1 ("0.01645, 0.02664, 0.047028, 0.088, 0.17, 0.33242, 0.6586");
      index_2 ("0.000588708, 0.00165662, 0.00379363, 0.00806529, 0.0166133, 0.0337035, 0.0678897");
      values ( \
        "0.001627, 0.001701, 0.001789, 0.001895, 0.002038, 0.002291, 0.002772", \
        "0.001615, 0.001686, 0.001777, 0.001882, 0.002032, 0.002280, 0.002759", \
        "0.001595, 0.001664, 0.001753, 0.001858, 0.002005, 0.002263, 0.002734", \
        "0.001563, 0.001630, 0.001713, 0.001819, 0.001968, 0.002235, 0.002696", \
        "0.001515, 0.001578, 0.001656, 0.001763, 0.001921, 0.002182, 0.002676", \
        "0.001475, 0.001534, 0.001607, 0.001707, 0.001867, 0.002145, 0.002625", \
        "0.001507, 0.001559, 0.001635, 0.001676, 0.001831, 0.002115, 0.002600");
    }
    fall_power(energy_template_7x7) {
      index_1 ("0.01645, 0.02664, 0.047028, 0.088, 0.17, 0.33242, 0.6586");
      index_2 ("0.000588708, 0.00165662, 0.00379363, 0.00806529, 0.0166133, 0.0337035, 0.0678897");
      values ( \
        "0.002907, 0.003016, 0.003108, 0.003160, 0.003183, 0.003191, 0.003194", \
        "0.002889, 0.002998, 0.003090, 0.003143, 0.003167, 0.003175, 0.003178", \
        "0.002859, 0.002966, 0.003057, 0.003112, 0.003138, 0.003146, 0.003150", \
        "0.002814, 0.002916, 0.003005, 0.003062, 0.003094, 0.003108, 0.003112", \
        "0.002756, 0.002853, 0.002940, 0.003002, 0.003042, 0.003063, 0.003073", \
        "0.002701, 0.002800, 0.002886, 0.002950, 0.002996, 0.003027, 0.003045", \
        "0.002822, 0.002847, 0.002879, 0.002907, 0.002957, 0.002995, 0.003022");
    }
  }
```

Library – file.lib

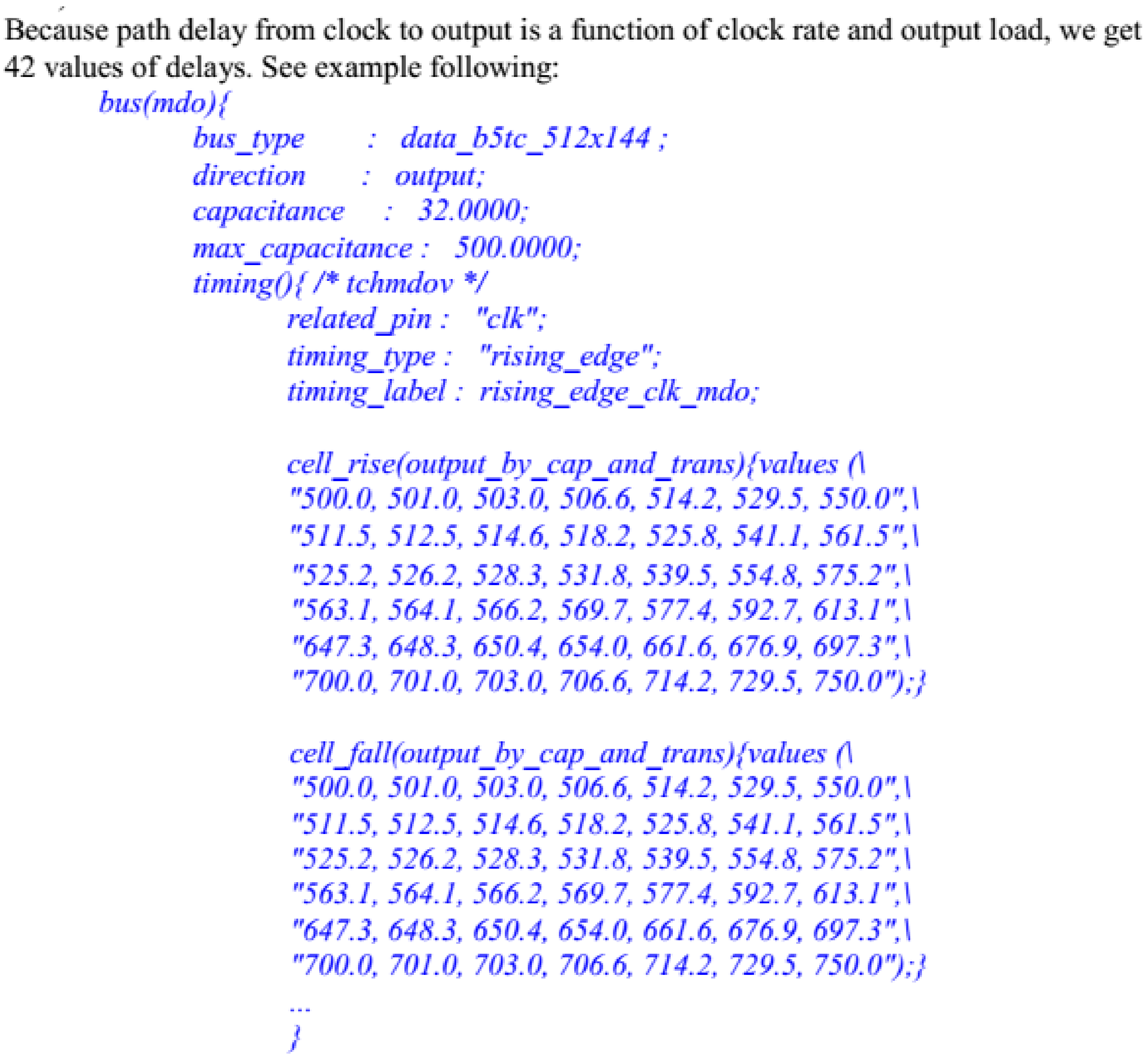
<file>.lib



```
timing() {
  related_pin : "A";
  timing_sense : positive_unate;
  cell_rise(delay_template_7x7) {
    index_1 ("0.01645, 0.02664, 0.047028, 0.088, 0.17, 0.33242, 0.6586");
    index_2 ("0.000588708, 0.00165662, 0.00379363, 0.00806529, 0.0166133, 0.0337035, 0.0678897");
    values ( \
      "0.087129, 0.097274, 0.114857, 0.146936, 0.208982, 0.332124, 0.578079", \
      "0.255034, 0.267117, 0.286151, 0.318946, 0.381285, 0.504530, 0.750353");
  }
  rise_transition(delay_template_7x7) {
    index_1 ("0.01645, 0.02664, 0.047028, 0.088, 0.17, 0.33242, 0.6586");
    index_2 ("0.000588708, 0.00165662, 0.00379363, 0.00806529, 0.0166133, 0.0337035, 0.0678897");
    values ( \
      "0.036180, 0.048856, 0.074428, 0.127614, 0.237810, 0.460566, 0.905416", \
      "0.053975, 0.063334, 0.084411, 0.133168, 0.240776, 0.461000, 0.905488");
  }
  cell_fall(delay_template_7x7) {
    index_1 ("0.01645, 0.02664, 0.047028, 0.088, 0.17, 0.33242, 0.6586");
    index_2 ("0.000588708, 0.00165662, 0.00379363, 0.00806529, 0.0166133, 0.0337035, 0.0678897");
    values ( \
      "0.079281, 0.086919, 0.099649, 0.121818, 0.163394, 0.245395, 0.409164", \
      "0.273719, 0.283894, 0.299151, 0.323264, 0.365685, 0.447616, 0.610937");
  }
  fall_transition(delay_template_7x7) {
    index_1 ("0.01645, 0.02664, 0.047028, 0.088, 0.17, 0.33242, 0.6586");
    index_2 ("0.000588708, 0.00165662, 0.00379363, 0.00806529, 0.0166133, 0.0337035, 0.0678897");
    values ( \
      "0.029807, 0.038268, 0.055169, 0.089881, 0.162821, 0.312244, 0.612448", \
      "0.050689, 0.057152, 0.070205, 0.099766, 0.167400, 0.313346, 0.612410");
```

Library – file.lib

<file>.lib

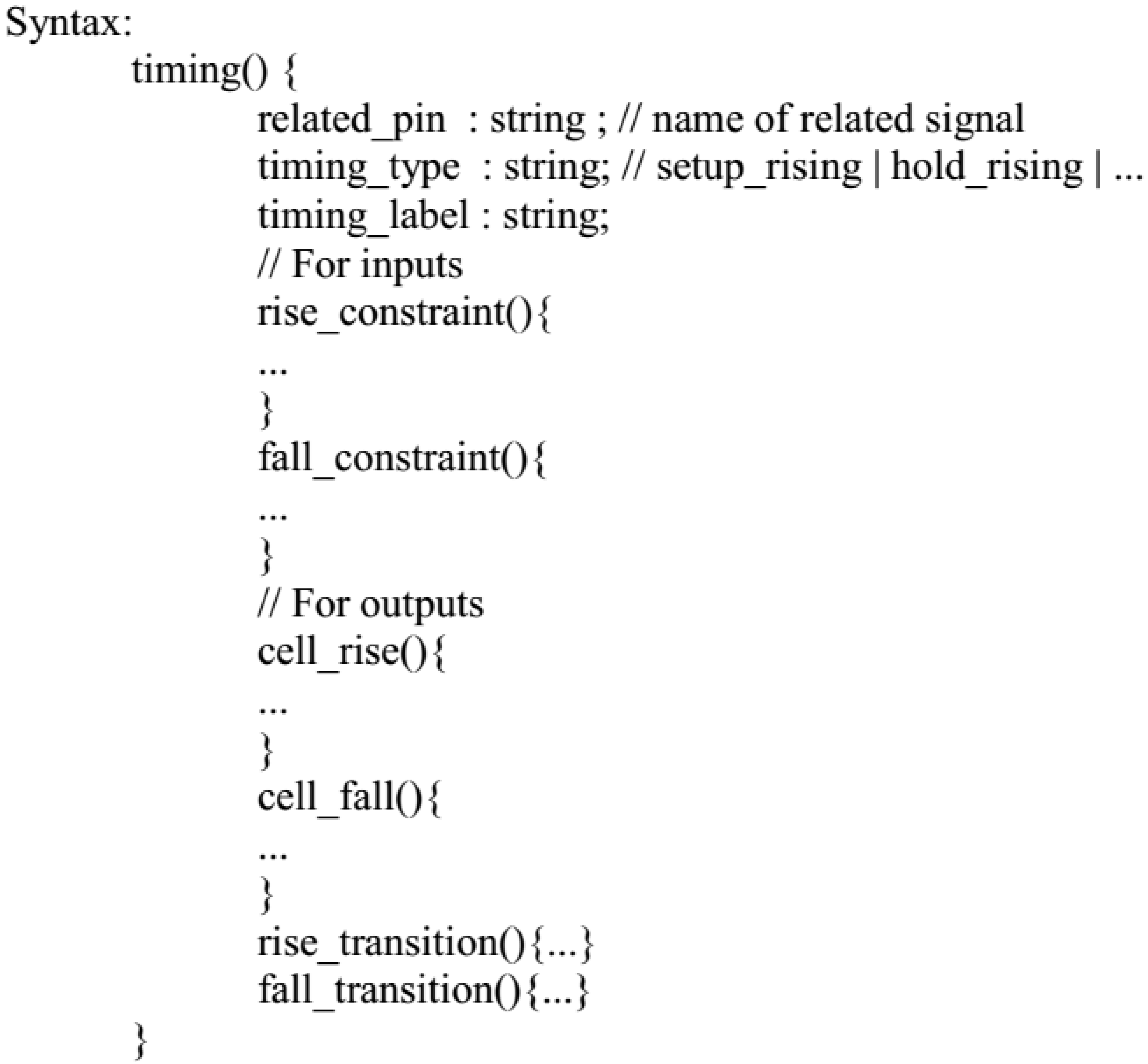


Because path delay from clock to output is a function of clock rate and output load, we get 42 values of delays. See example following:

```
bus(mdo){
        bus_type        :  data_b5tc_512x144 ;
        direction       :  output;
        capacitance     :  32.0000;
        max_capacitance :  500.0000;
        timing(){ /* tchmdov */
                related_pin :  "clk";
                timing_type :  "rising_edge";
                timing_label :  rising_edge_clk_mdo;

                cell_rise(output_by_cap_and_trans){values (\
                "500.0, 501.0, 503.0, 506.6, 514.2, 529.5, 550.0",\
                "511.5, 512.5, 514.6, 518.2, 525.8, 541.1, 561.5",\
                "525.2, 526.2, 528.3, 531.8, 539.5, 554.8, 575.2",\
                "563.1, 564.1, 566.2, 569.7, 577.4, 592.7, 613.1",\
                "647.3, 648.3, 650.4, 654.0, 661.6, 676.9, 697.3",\
                "700.0, 701.0, 703.0, 706.6, 714.2, 729.5, 750.0");}

                cell_fall(output_by_cap_and_trans){values (\
                "500.0, 501.0, 503.0, 506.6, 514.2, 529.5, 550.0",\
                "511.5, 512.5, 514.6, 518.2, 525.8, 541.1, 561.5",\
                "525.2, 526.2, 528.3, 531.8, 539.5, 554.8, 575.2",\
                "563.1, 564.1, 566.2, 569.7, 577.4, 592.7, 613.1",\
                "647.3, 648.3, 650.4, 654.0, 661.6, 676.9, 697.3",\
                "700.0, 701.0, 703.0, 706.6, 714.2, 729.5, 750.0");}
                ...
                }
```

Library – file.lib

<file>.lib

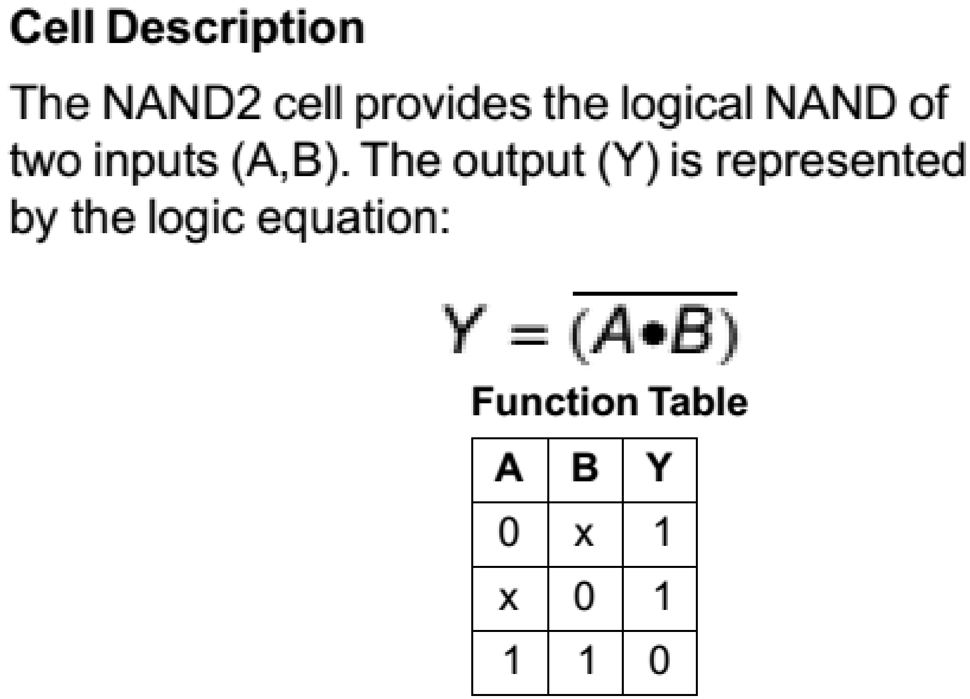


Syntax:

```
timing() {
        related_pin  : string ; // name of related signal
        timing_type  : string; // setup_rising | hold_rising | ...
        timing_label : string;
        // For inputs
        rise_constraint(){
        ...
        }
        fall_constraint(){
        ...
        }
        // For outputs
        cell_rise(){
        ...
        }
        cell_fall(){
        ...
        }
        rise_transition(){...}
        fall_transition(){...}
}
```

Library – file.lib

<file>.lib



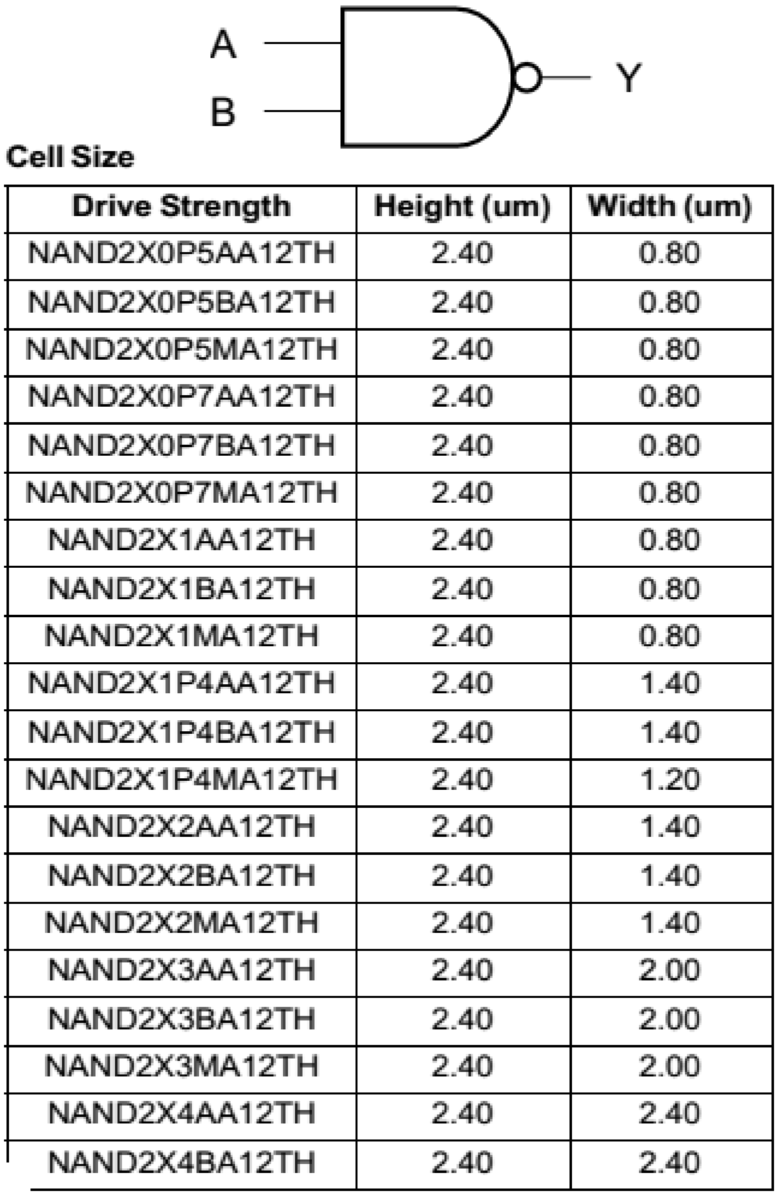
Cell Description

The NAND2 cell provides the logical NAND of two inputs (A,B). The output (Y) is represented by the logic equation:

$$Y = \overline{(A \bullet B)}$$

Function Table

A	B	Y
0	x	1
x	0	1
1	1	0



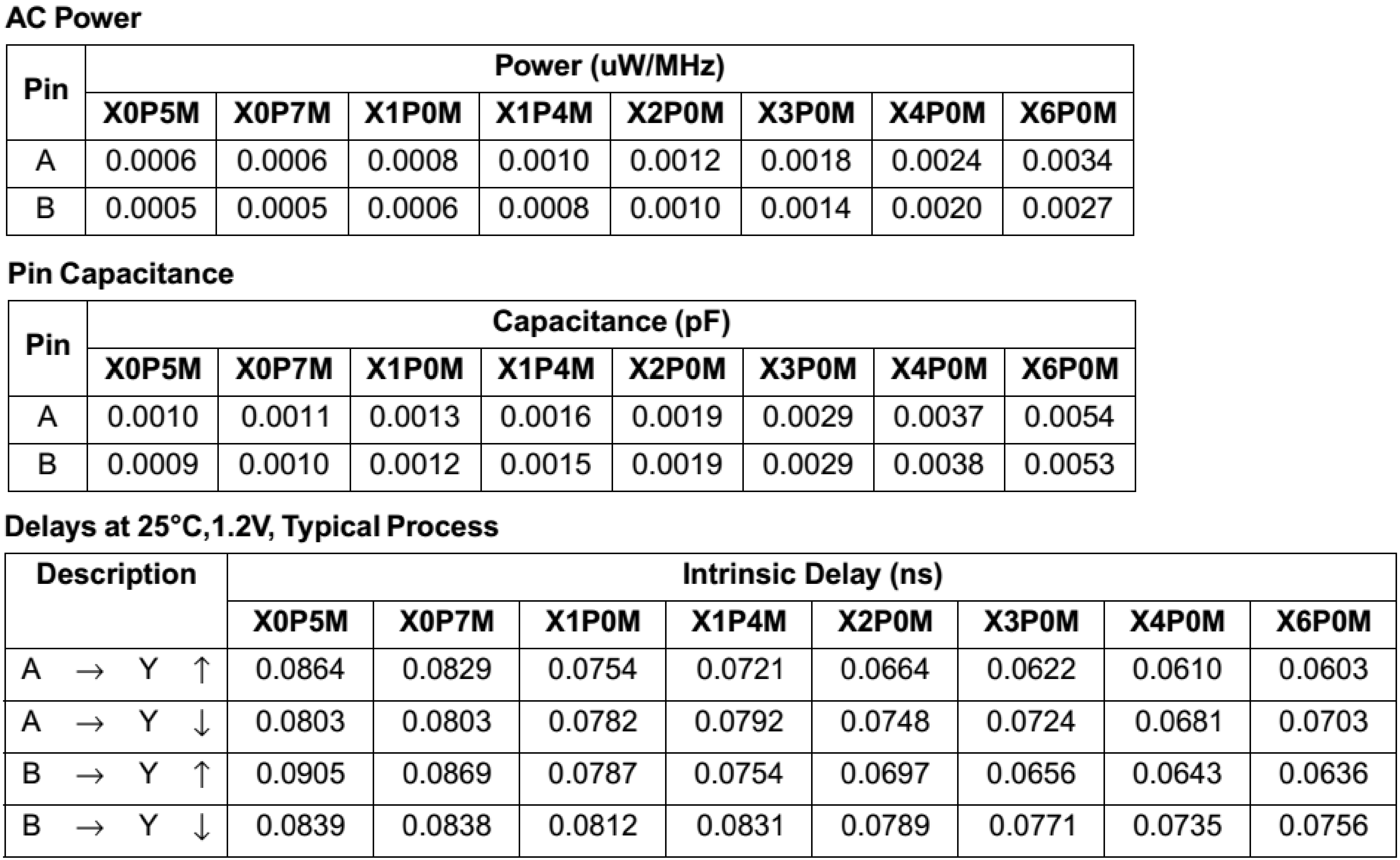
Cell Size

Drive Strength	Height (um)	Width (um)
NAND2X0P5AA12TH	2.40	0.80
NAND2X0P5BA12TH	2.40	0.80
NAND2X0P5MA12TH	2.40	0.80
NAND2X0P7AA12TH	2.40	0.80
NAND2X0P7BA12TH	2.40	0.80
NAND2X0P7MA12TH	2.40	0.80
NAND2X1AA12TH	2.40	0.80
NAND2X1BA12TH	2.40	0.80
NAND2X1MA12TH	2.40	0.80
NAND2X1P4AA12TH	2.40	1.40
NAND2X1P4BA12TH	2.40	1.40
NAND2X1P4MA12TH	2.40	1.20
NAND2X2AA12TH	2.40	1.40
NAND2X2BA12TH	2.40	1.40
NAND2X2MA12TH	2.40	1.40
NAND2X3AA12TH	2.40	2.00
NAND2X3BA12TH	2.40	2.00
NAND2X3MA12TH	2.40	2.00
NAND2X4AA12TH	2.40	2.40
NAND2X4BA12TH	2.40	2.40

From <file>.doc/pdf

Library – file.lib

<file>.lib



AC Power

Pin	Power (uW/MHz)							
	X0P5M	X0P7M	X1P0M	X1P4M	X2P0M	X3P0M	X4P0M	X6P0M
A	0.0006	0.0006	0.0008	0.0010	0.0012	0.0018	0.0024	0.0034
B	0.0005	0.0005	0.0006	0.0008	0.0010	0.0014	0.0020	0.0027

Pin Capacitance

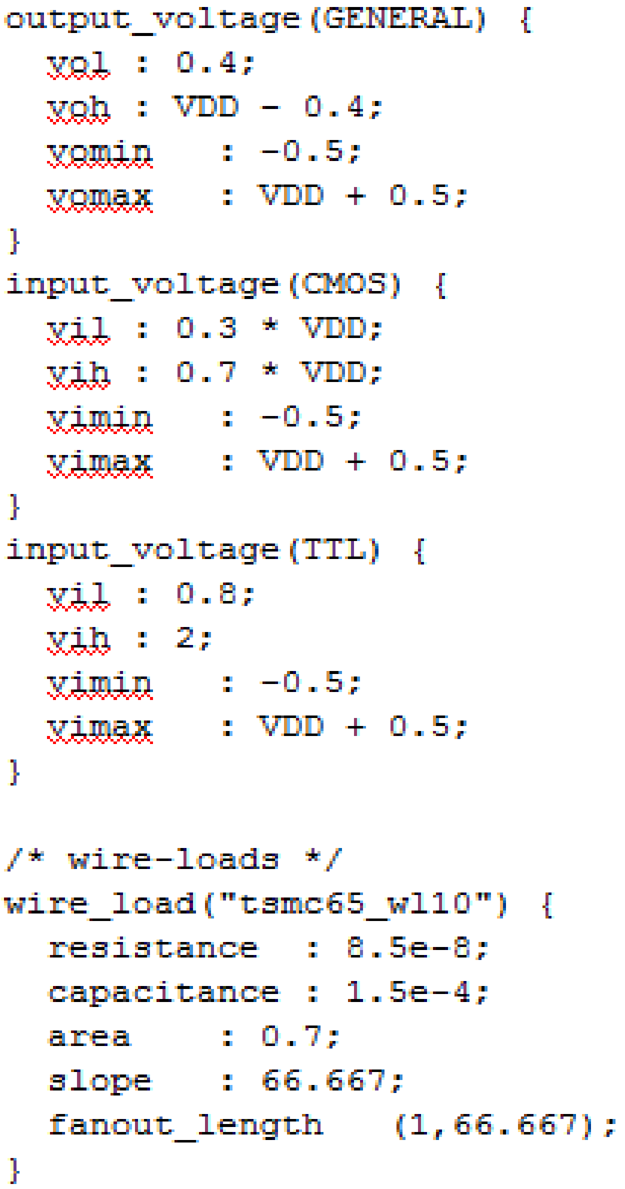
Pin	Capacitance (pF)							
	X0P5M	X0P7M	X1P0M	X1P4M	X2P0M	X3P0M	X4P0M	X6P0M
A	0.0010	0.0011	0.0013	0.0016	0.0019	0.0029	0.0037	0.0054
B	0.0009	0.0010	0.0012	0.0015	0.0019	0.0029	0.0038	0.0053

Delays at 25°C,1.2V, Typical Process

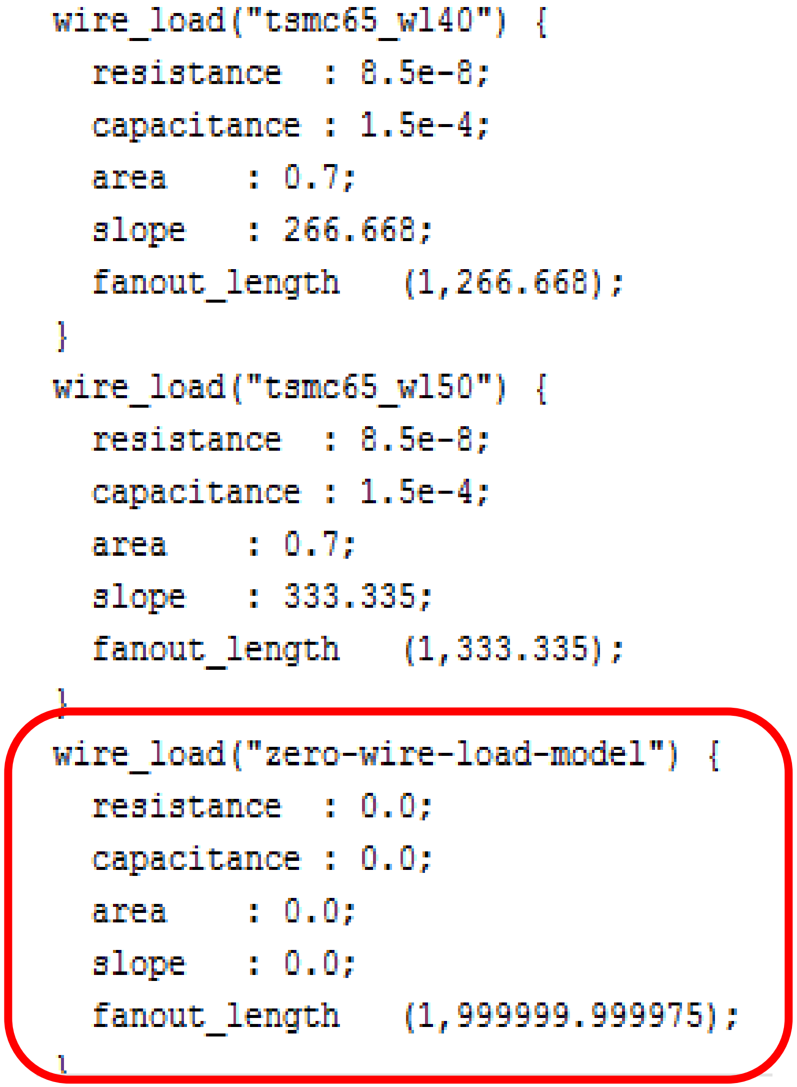
Description	Intrinsic Delay (ns)							
	X0P5M	X0P7M	X1P0M	X1P4M	X2P0M	X3P0M	X4P0M	X6P0M
A → Y ↑	0.0864	0.0829	0.0754	0.0721	0.0664	0.0622	0.0610	0.0603
A → Y ↓	0.0803	0.0803	0.0782	0.0792	0.0748	0.0724	0.0681	0.0703
B → Y ↑	0.0905	0.0869	0.0787	0.0754	0.0697	0.0656	0.0643	0.0636
B → Y ↓	0.0839	0.0838	0.0812	0.0831	0.0789	0.0771	0.0735	0.0756

Library – file.lib

<file>.lib

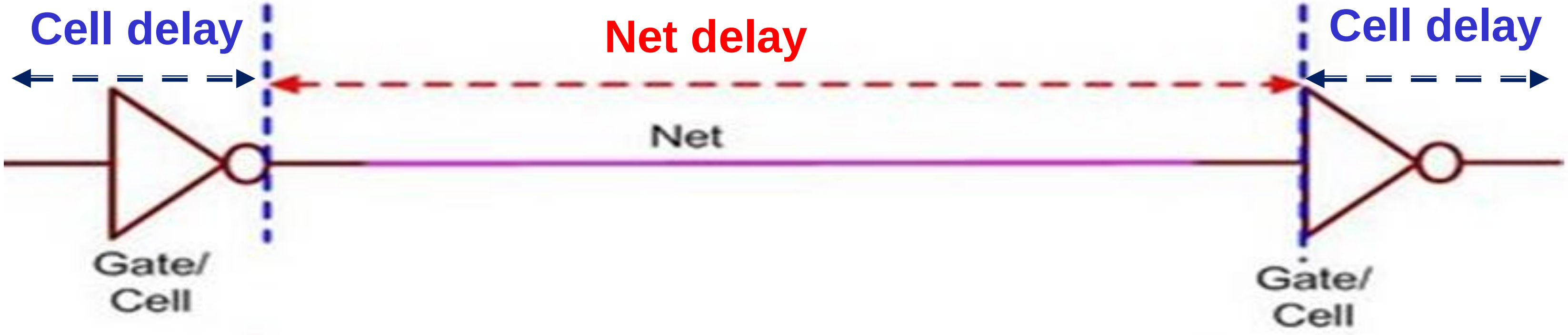


```
output_voltage(GENERAL)  {
   vol : 0.4;
   voh : VDD - 0.4;
   vomin     : -0.5;
   vomax     : VDD + 0.5;
}
input_voltage(CMOS)  {
   vil : 0.3 * VDD;
   vih : 0.7 * VDD;
   vimin     : -0.5;
   vimax     : VDD + 0.5;
}
input_voltage(TTL)  {
   vil : 0.8;
   vih : 2;
   vimin     : -0.5;
   vimax     : VDD + 0.5;
}

/* wire-loads */
wire_load("tsmc65_wl10")  {
   resistance   : 8.5e-8;
   capacitance  : 1.5e-4;
   area      : 0.7;
   slope     : 66.667;
   fanout_length    (1,66.667);
}
```



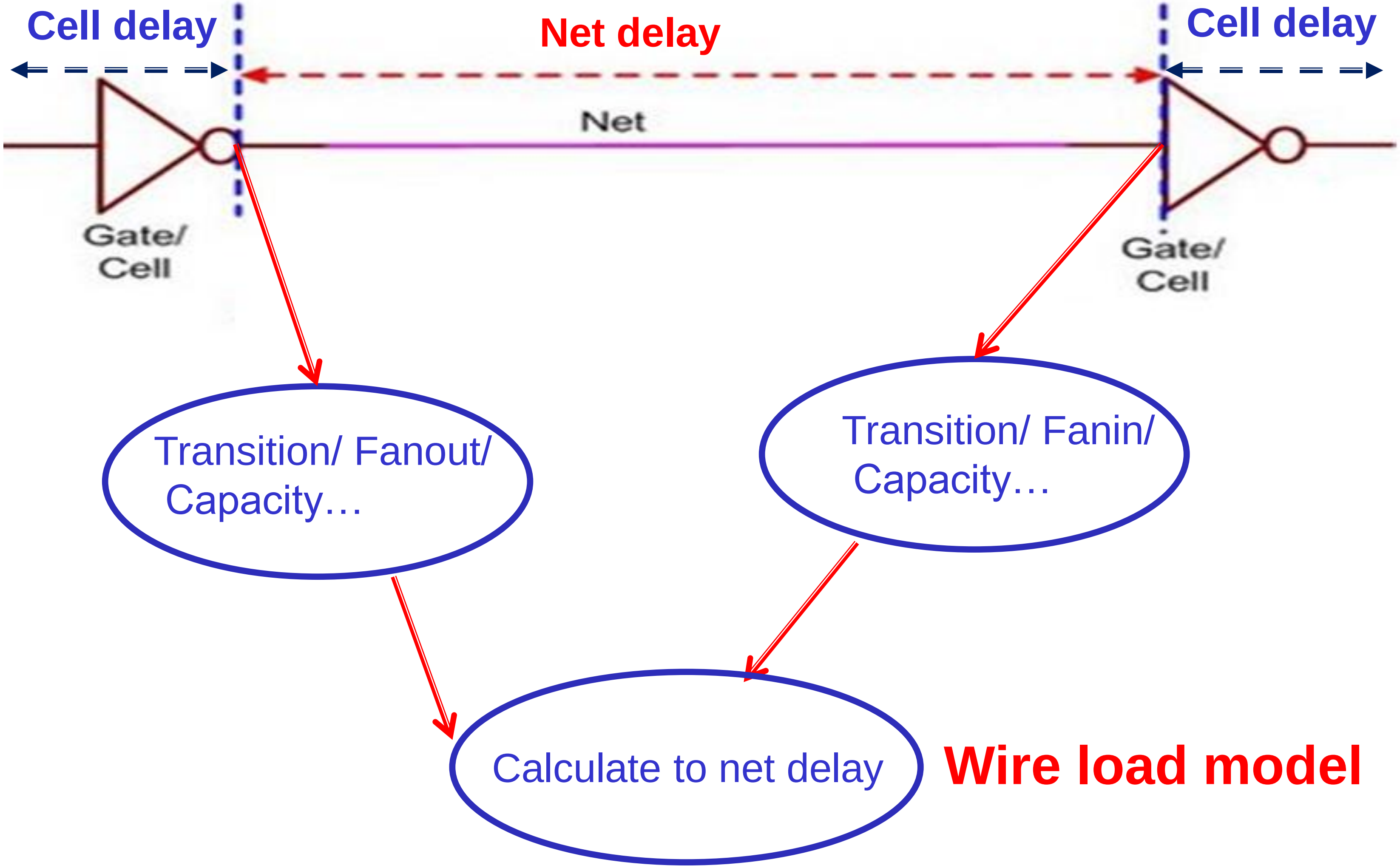
```
wire_load("tsmc65_wl40")  {
   resistance   : 8.5e-8;
   capacitance : 1.5e-4;
   area      : 0.7;
   slope     : 266.668;
   fanout_length    (1,266.668);
}
wire_load("tsmc65_wl50")  {
   resistance   : 8.5e-8;
   capacitance : 1.5e-4;
   area      : 0.7;
   slope     : 333.335;
   fanout_length    (1,333.335);
}
wire_load("zero-wire-load-model")  {
   resistance   : 0.0;
   capacitance : 0.0;
   area      : 0.0;
   slope     : 0.0;
   fanout_length    (1,999999.999975);
}
```

Wire load model



Q1. Before P&R step, which timing/power … information for net delay?

Wire load model



Cell delay

Net delay

Cell delay

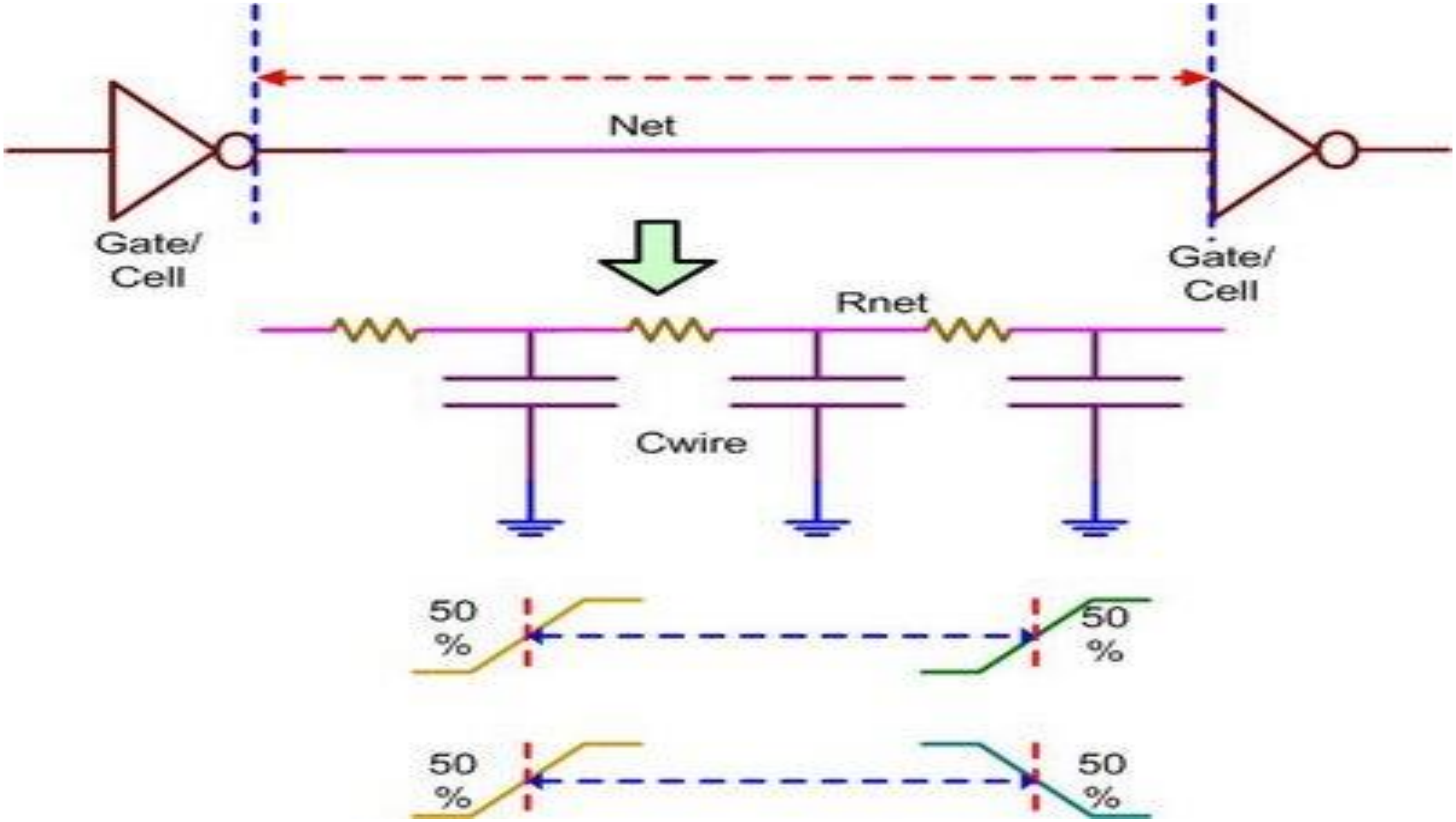
Transition/ Fanout/ Capacity…

Transition/ Fanin/ Capacity…

Calculate to net delay

Wire load model

Wire load model

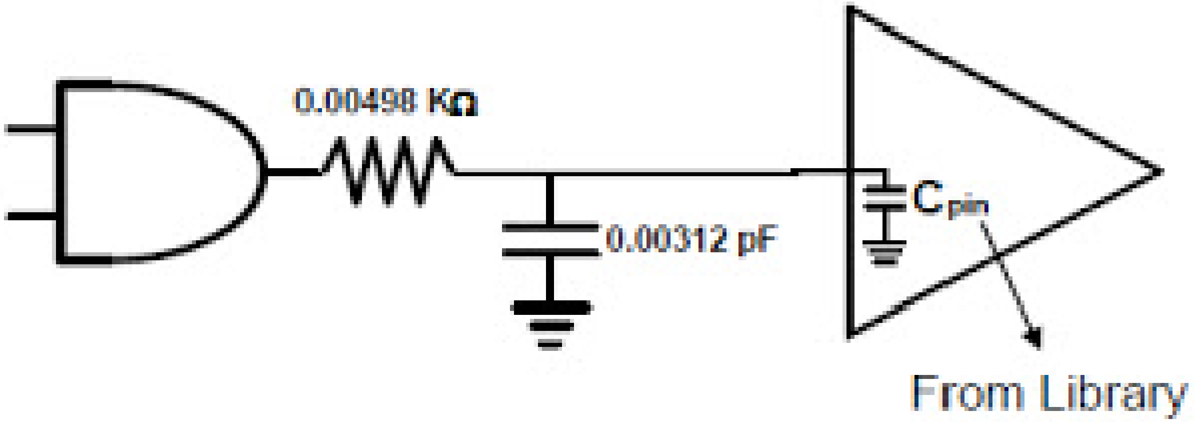


Wire load model

- ❑ Used to estimate the interconnect wire delay during pre-layout in a design cycle.
- ❑ Wire load information is based on statistics from physical layout parasitic
  - ✓ Information from the statistics is used in both conservative and aggressive tables.
  - ✓ The conservative tables are based on “mean value” plus 3-sigma; the aggressive tables on “mean value” plus 1-sigma.
- ❑ Different for different technology.
  - ✓ Wire load models are approximated from one technology to another based on scaling factors. Due to these approximations, the accuracy of these models diminish over multiple technology nodes
- ❑ Describes effect of wire length and fanout on
  - ✓ Resistance
  - ✓ Capacitance
  - ✓ Area of the nets.
- ❑ All attributes (R, C and Area) are given per unit length wire.
- ❑ Slope value is used to characterize linear fanout.
- ❑ Basically a set of tables
  - ✓ Net fanout vs load
  - ✓ Net fanout vs resistance
  - ✓ Net fanout vs area

Wire load model

Net Fanout	Resistance KΩ	Capacitance pF
1	**0.00498**	**0.00312**
2	0.01295	0.00812
3	0.02092	0.01312
4	0.02888	0.01811



Wire load model

```
wire_load("WLM1")     {
resistance  :        0.0006       ;------>R per unit length
capacitance :        0.0001       ;------> C per unit length
area :               0.1          ;------> Area per unit length
slope        :       1.5          ;------> Used for linear extrapolation
fanout_length(1,  0.002)          ; ------> at fanout “1” length of the wire is 0.002
fanout_length(2,  0.006);
fanout_length(3,  0.009);
fanout_length(4,  0.015);
fanout_length(5,  0.020);
fanout_length(7,  0.028);           ------> at fanout “7” length of the wire is 0.028
fanout_length(8,  0.030);
fanout_length(9,  0.035);
fanout_length(10, 0.040);
}
```

Wire load model

In general, not all fan outs are mentioned in a given WLM lookup table. For example, in above WLM1 and WLM2 lookup table, capacitance and resistance values for fan outs 1, 2, 3, 4, 5, 7, 8, 9, 10 is given. If we want to estimate the values at fan outs in the gaps (e.g. from 6) or outside the fanout range specified in the table (e.g Fan out 20), we have to calculated those value using (linear) interpolation and extrapolation.

- ❑ **For WLM1 →For Fanout = 20 :** Since its more than the max value of Fanout available in table (i.e 10) , so we have to perform extrapolation.

  - ✓ Net length = <length of net at fanout 10> + (20-10) x Slope
  - ✓ Resistance = <new calculated Net length at fanout 6> x Resistance or Capacitance value per unit length
  - ✓ Capacitance = <new calculated Net length at fanout 6> x Capacitance value per unit length

  - ✓ Net length = 0.040 + 10 x 1.5 (slope) = 15.04 → length of net with fanout of 20
  - ✓ Resistance = 15.04 x 0.0006 = 0.009024 units
  - ✓ Capacitance = 15.04 x 0.0001 = 0.001504 units

Wire load model

- ❑ **For Fan out=6 :** Since it's between 5 and 7 and corresponding fanout Vs length is available, we can do the interpolation.

  - ✓ Net length = ((net length at fanout 5) + (net length at fanout 7)) / 2
  - ✓ Resistance = <new calculated Net length at fanout 20> x Resistance value per unit length
  - ✓ Capacitance = <new calculated Net length at fanout 20> x Capacitance value per unit length

  - ✓ Net length = (0.0020 + 0.0028)/2=0.0048/2=0.0024 ----------> length of net with fanout of 6
  - ✓ Resistance = 0.0024 x 0.0006 = 0.00000144 units
  - ✓ Capacitance = 0.0024 x 0.0001 = 0.00000024 units

Synthesis step

Wire load model

WLM Types : For flows that run timing-based logic optimization before placement, there are three basic types of WLMs that can be used:

1. Statistical WLMs : Are based on averages over many similar designs using the same or similar physical libraries.
2. Structural WLMs: Use information about neighboring nets, rather than just fanout and module size information.
3. Custom WLMs: Are based on the current design after placement and routing, but before the current iteration of pre_placement synthesis.

Normally the semiconductor vendors will develop the models. ASIC vendors typically develop wireload models based on statistical information taken from a variety of example designs. For all the nets with a particular fanout, the number of nets with a given capacitance is plotted as a histogram. A single capacitance value is picked to represent this fanout value in the wireload model. If a very conservative wireload model is desired, the 90% decile might be picked (i.e. 90% of the nets in the sample have a capacitance smaller than that value). Similar statistics are gathered for resistance and net area. Usually the vendor supplies a family of wireload models, each to be used for a different size design. This is called area-based wireload selection

Wire load model

WLM Types : For flows that run timing-based logic optimization before placement, there are three basic types of WLMs that can be used:

1. Statistical WLMs : Are based on averages over many similar designs using the same or similar physical libraries.
2. Structural WLMs: Use information about neighboring nets, rather than just fanout and module size information.
3. Custom WLMs: Are based on the current design after placement and routing, but before the current iteration of pre_placement synthesis.

Normally the semiconductor vendors will develop the models. ASIC vendors typically develop wireload models based on statistical information taken from a variety of example designs. For all the nets with a particular fanout, the number of nets with a given capacitance is plotted as a histogram. A single capacitance value is picked to represent this fanout value in the wireload model. If a very conservative wireload model is desired, the 90% decile might be picked (i.e. 90% of the nets in the sample have a capacitance smaller than that value). Similar statistics are gathered for resistance and net area. Usually the vendor supplies a family of wireload models, each to be used for a different size design. This is called area-based wireload selection

Wire load model

Few Advance concepts: Till now we have discussed that for a particular Net you can estimate the RC value as per the WLM.
Let me ask you one question. What if your design is hierarchical? Do you think even in that case you can use the same WLM for a particular net which is crossing the hierarchical boundaries?
Short ANS is: you can use it but you will lose the accuracy. Just to solve this problem, Vendors usually supplies multiple WLMs. There are different Modes for WLM analysis- few important are: WLM analysis has three modes:

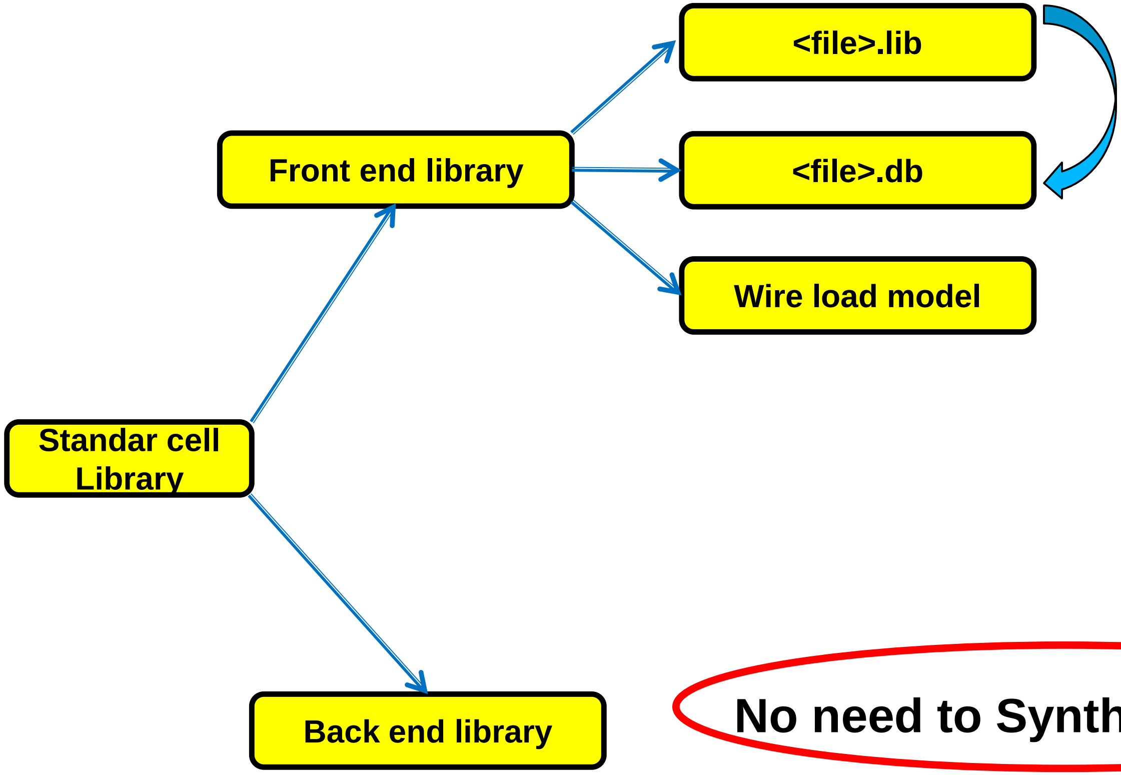
1. Top: Consider the design as it has no hierocracy and use the WLM for the top module to calculate delays for all modules. Any low level WLM is ignored.

2. Enclosed: Use the WLM of the module which completely encloses the net to compute delay for that net.

3. Segmented: If a net goes across several WLM, use the WLM that corresponds to that portion of the net which it encloses only.

Synthesis step

Library



Standar cell Library

Front end library

<file>.lib

<file>.db

Wire load model

Back end library

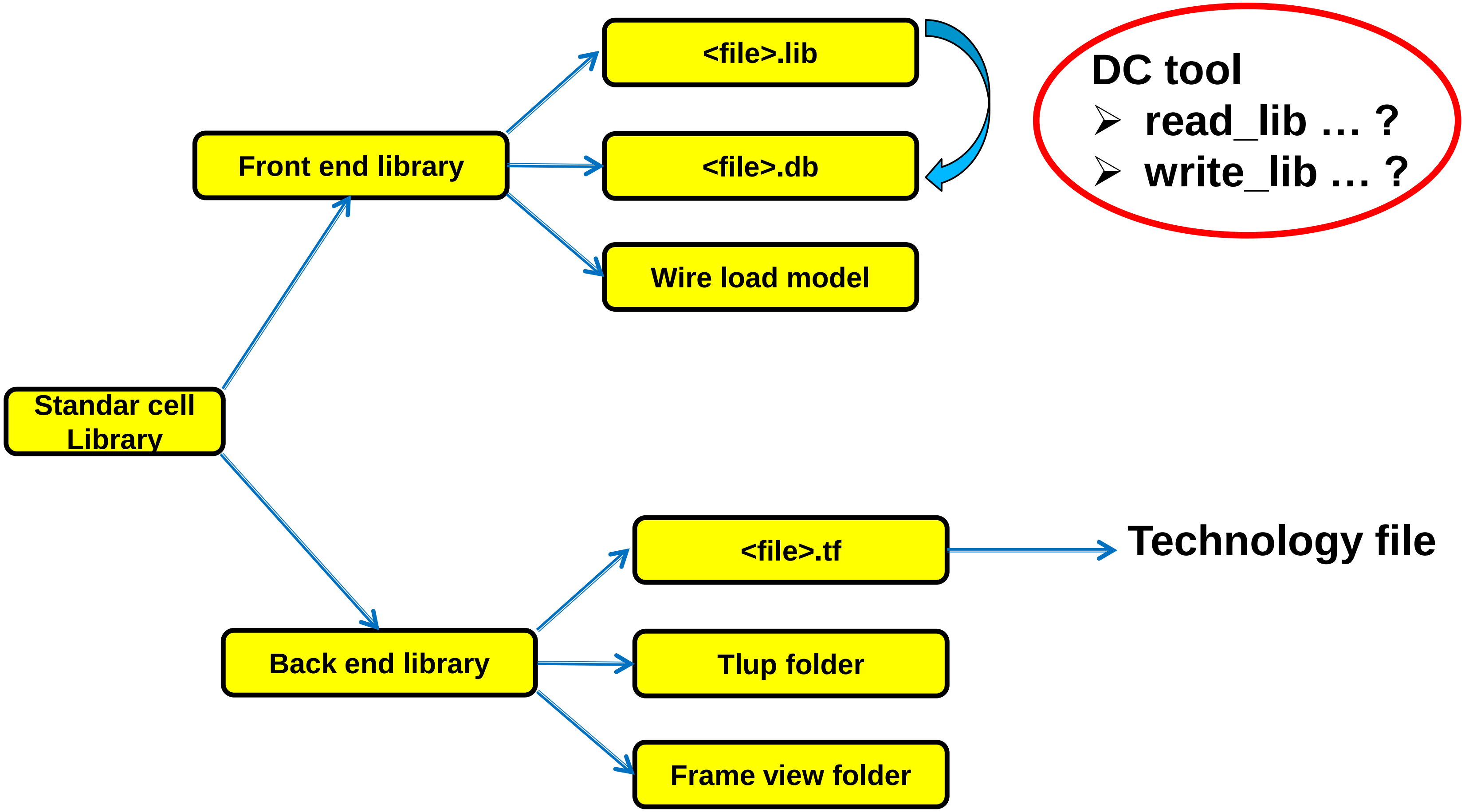
DC tool

- ➢ read_lib … ?
- ➢ write_lib … ?

No need to Synthesized task

Synthesis step

Library

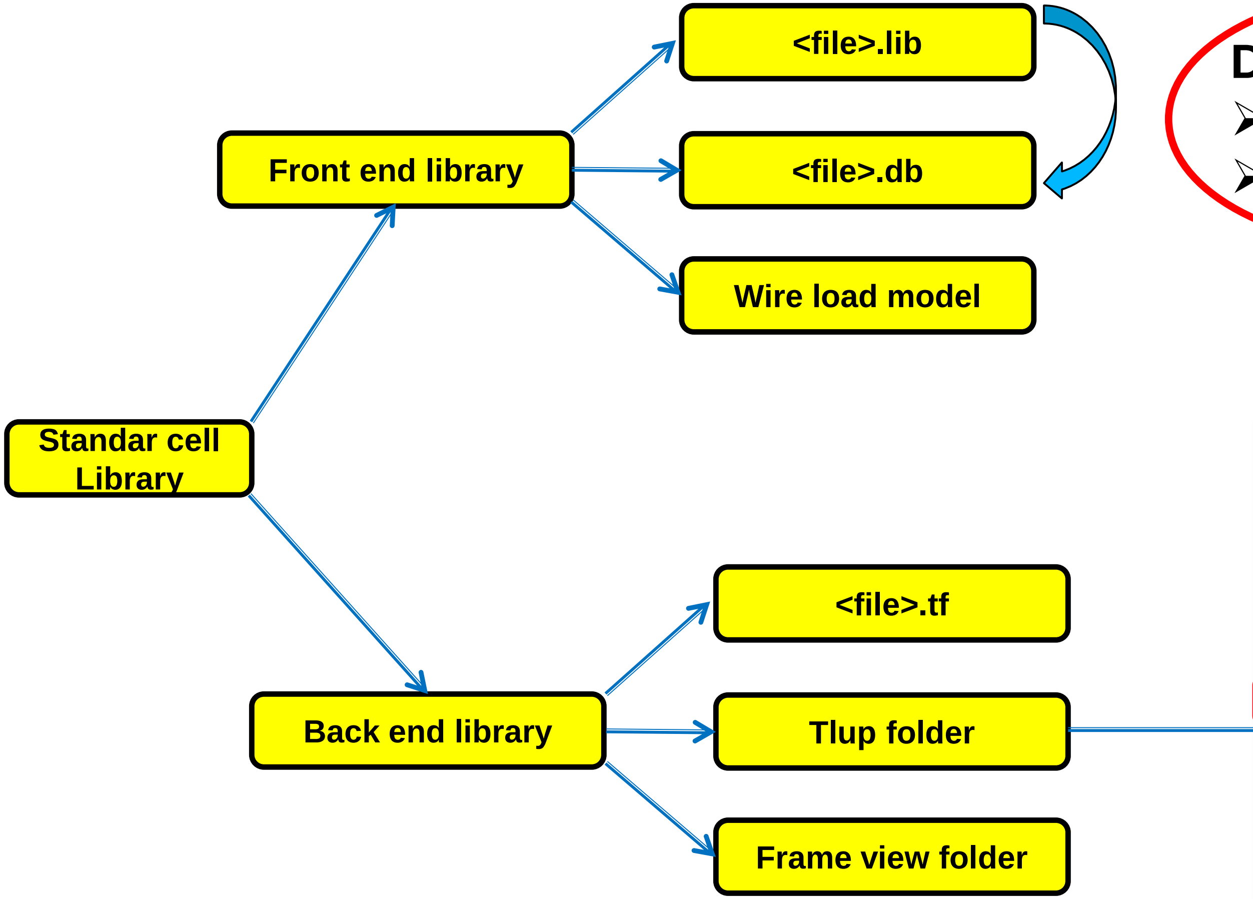


DC tool

- ➢ read_lib … ?
- ➢ write_lib … ?

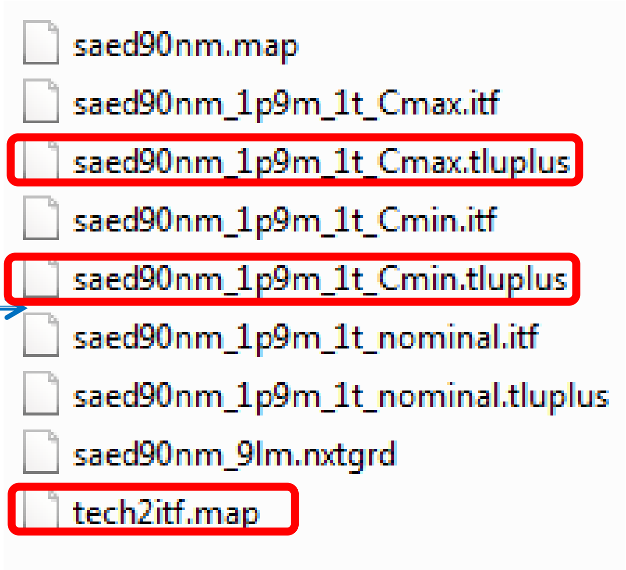
Synthesis step

Library



DC tool

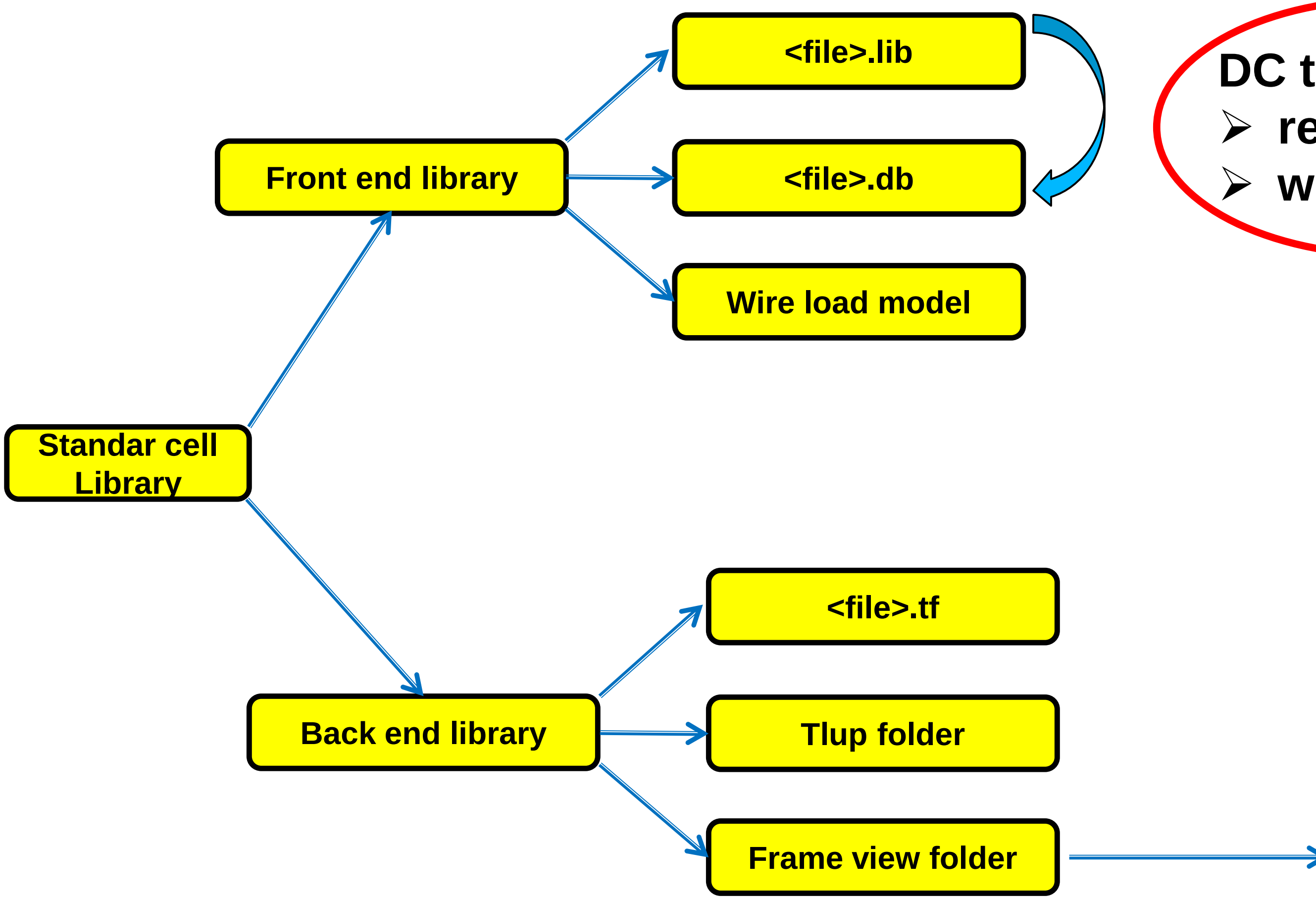
- read_lib … ?
- write_lib … ?



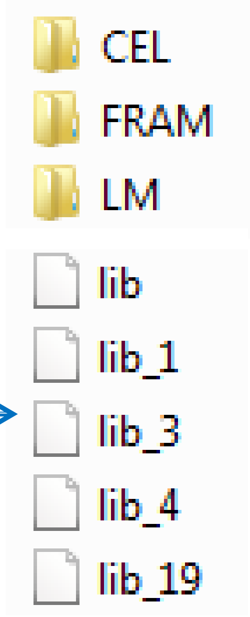
saed90nm.map
saed90nm_1p9m_1t_Cmax.itf
saed90nm_1p9m_1t_Cmax.tluplus
saed90nm_1p9m_1t_Cmin.itf
saed90nm_1p9m_1t_Cmin.tluplus
saed90nm_1p9m_1t_nominal.itf
saed90nm_1p9m_1t_nominal.tluplus
saed90nm_9lm.nxtgrd
tech2itf.map

Synthesis step

Library



DC tool
- ➢ read_lib … ?
- ➢ write_lib … ?

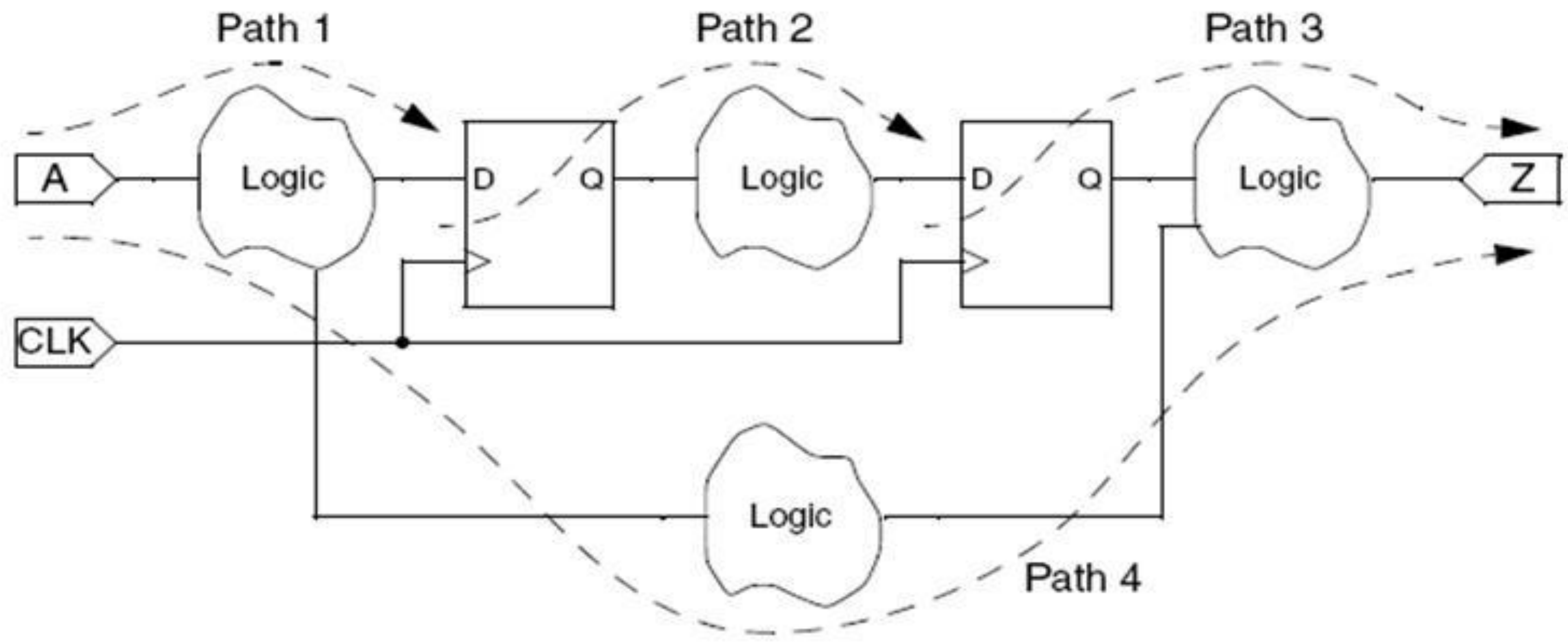

CEL
FRAM
LM
lib
lib_1
lib_3
lib_4
lib_19

Refer Library Directory

Static Timing Analysis

- ❖Dynamic vs. Static Timing Analysis
- ❖Synthesis Step
- ❖**Analysis**
- ❖Constraint
- ❖Violation
- ❖Using DC Tool in Command/GUI mode

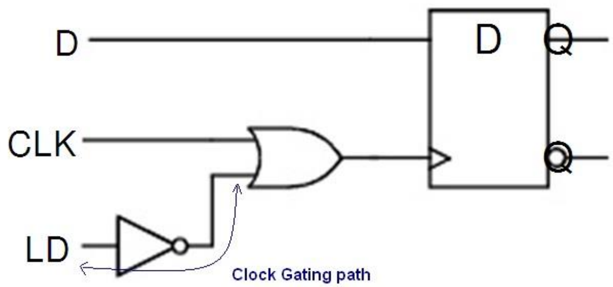
Timing Analysis – Data Path



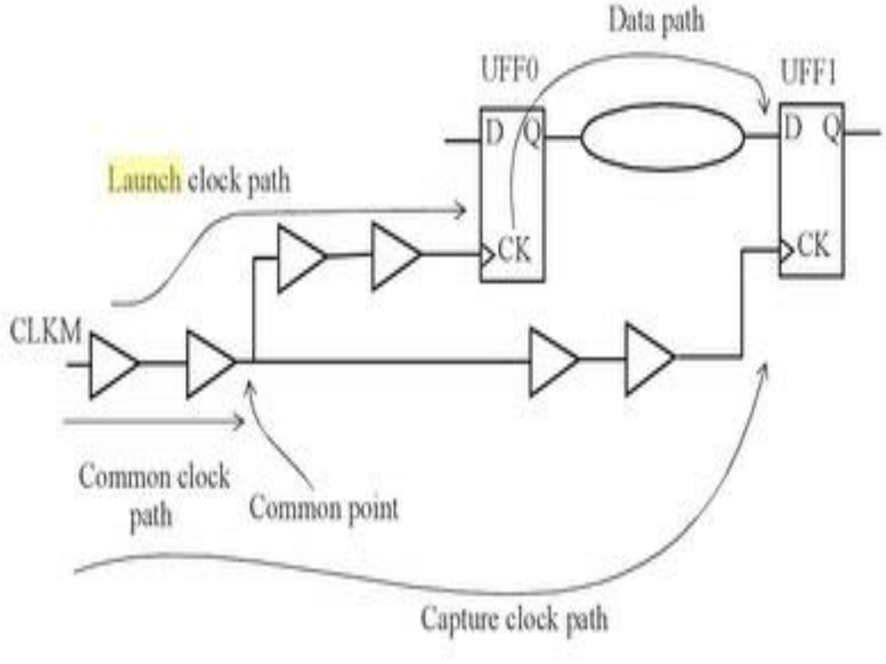
❖**Start point:** Data input, clock port of DFF

❖**End point:** Data output, Data input of DFF

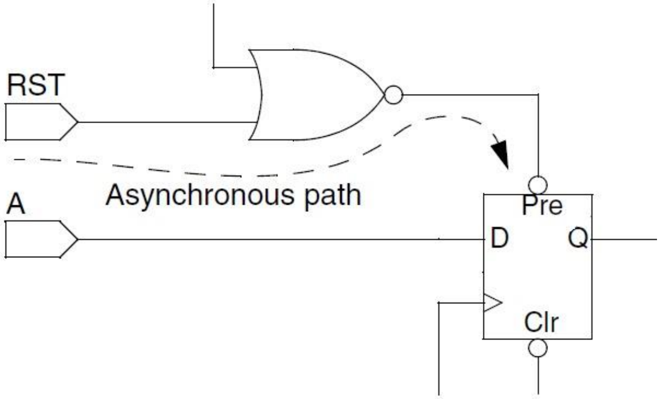
Timing Analysis – Clock paths



Clock gating path

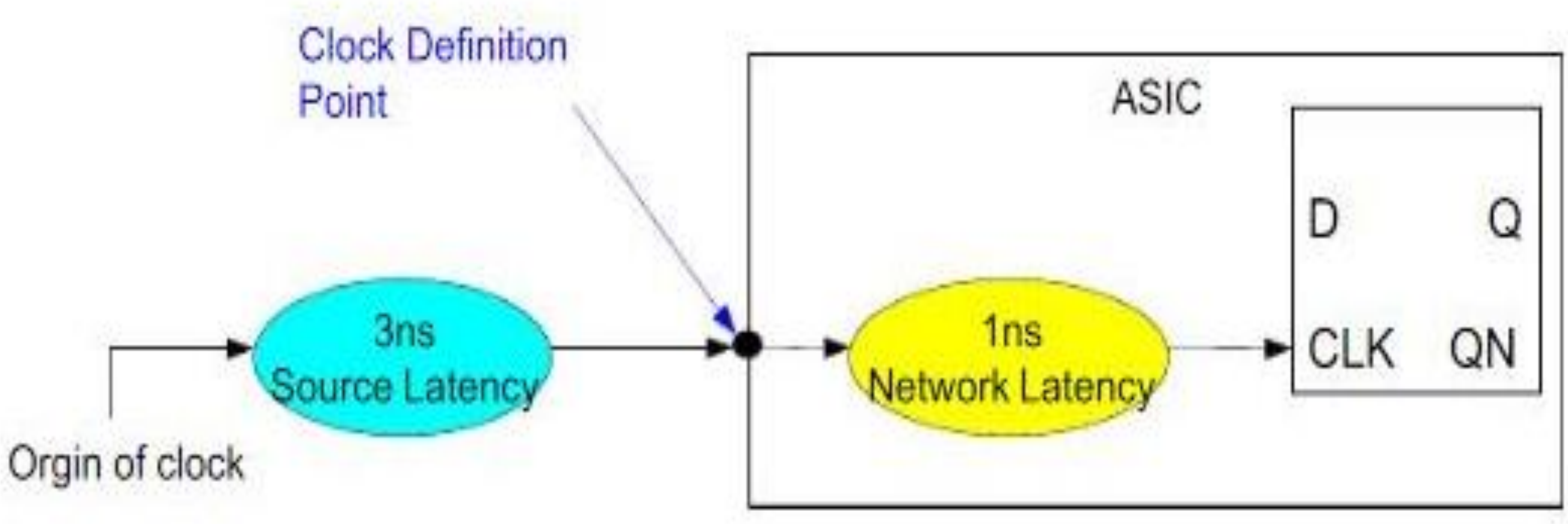


Clock path

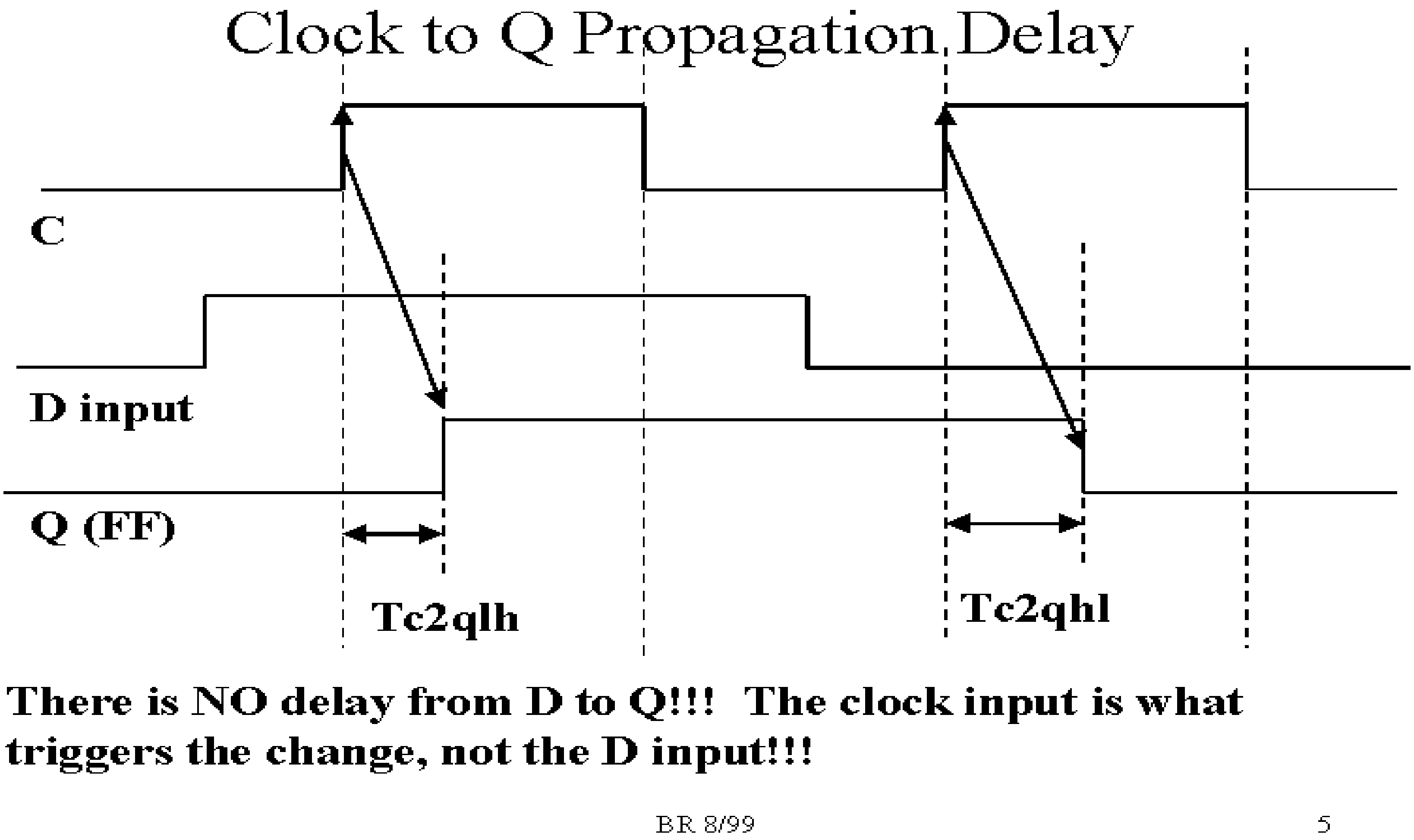


Asynchronous path

Timing Analysis – Clock latency



Timing Analysis – Propagation Delay

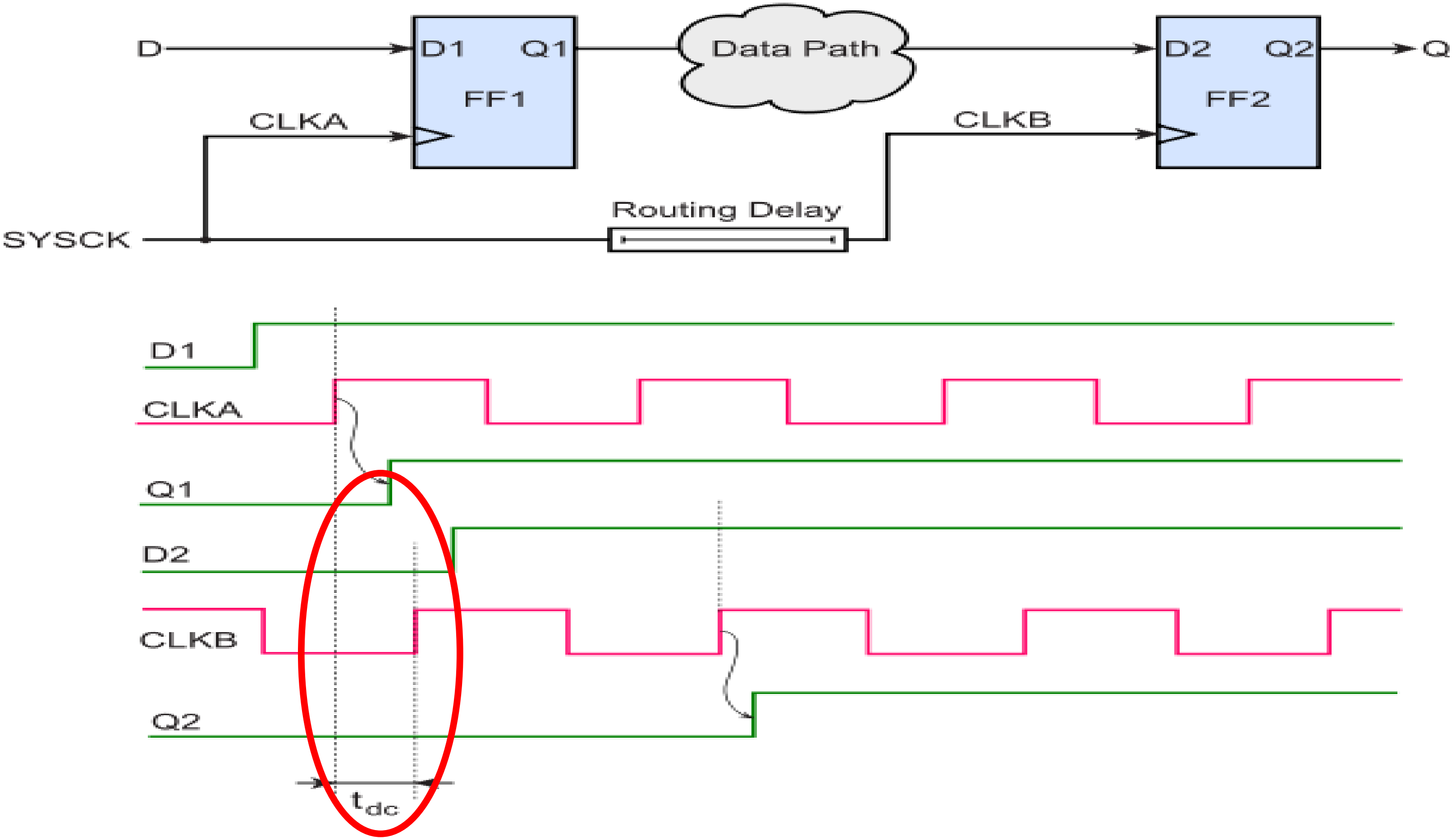


Clock to Q Propagation Delay

There is NO delay from D to Q!!! The clock input is what triggers the change, not the D input!!!

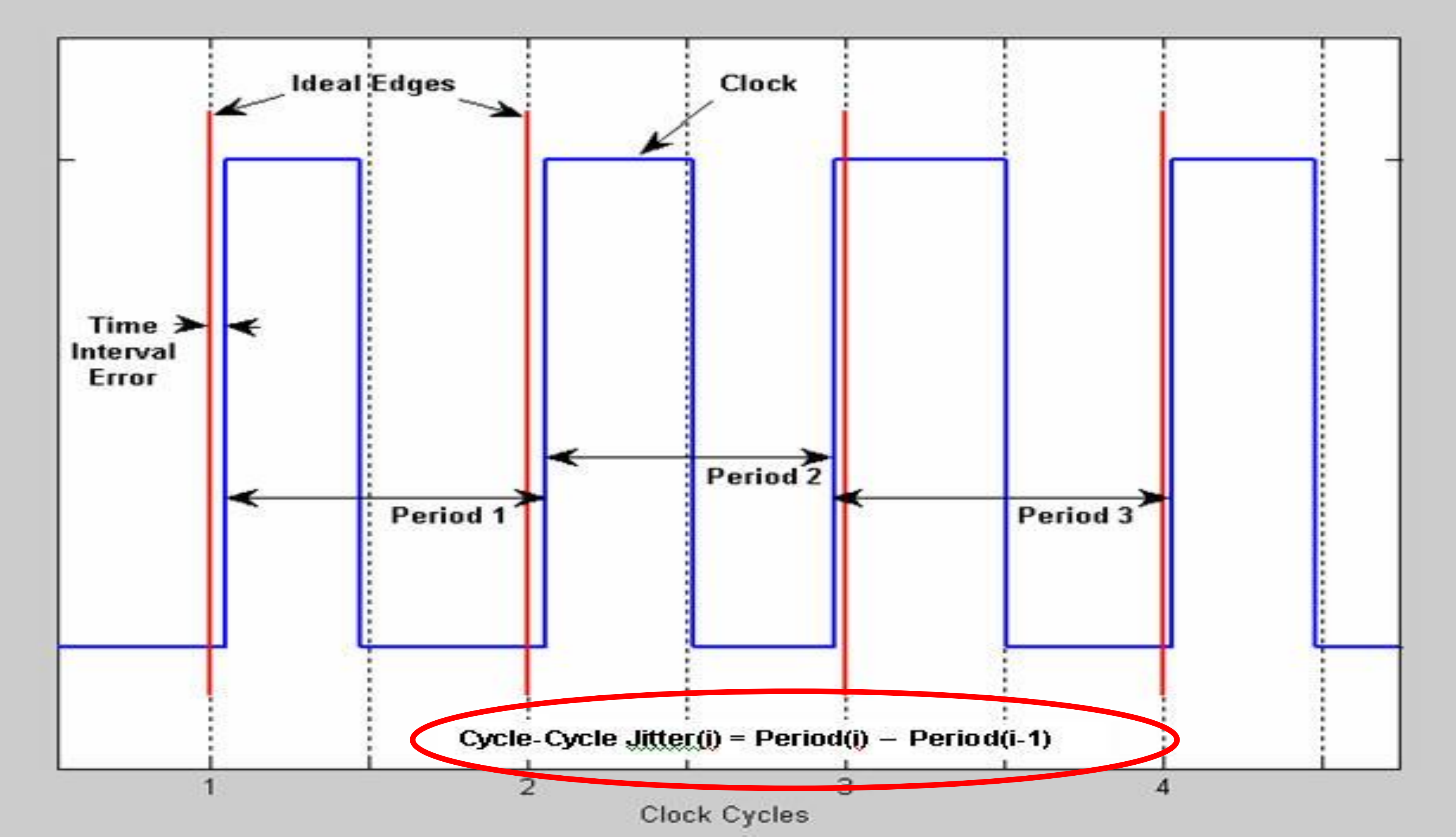
BR 8/99 5

Timing Analysis – Clock skew

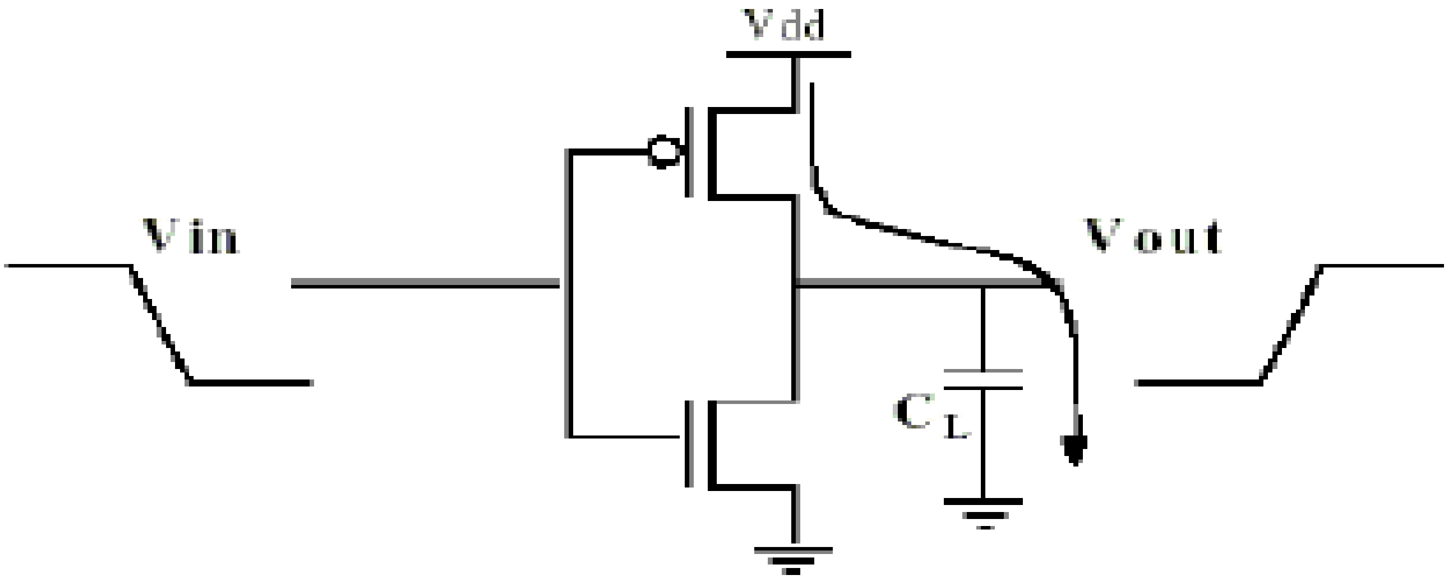


The $skew system task specifies the maximum delay allowable between two signals.

Timing Analysis – Clock jitter

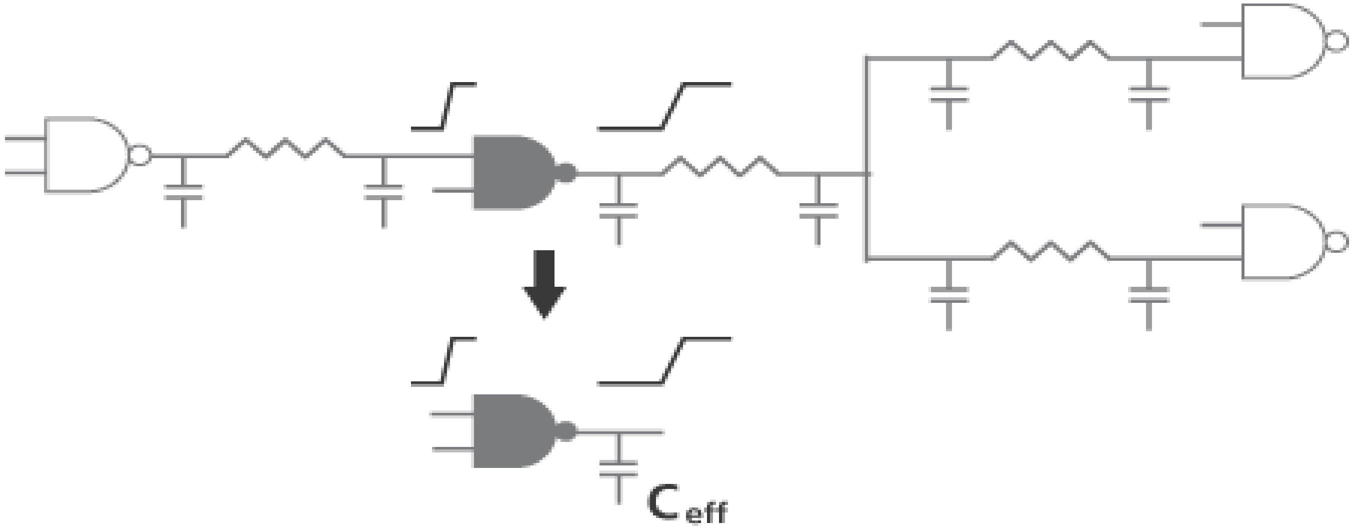


Timing Analysis – Cell delay

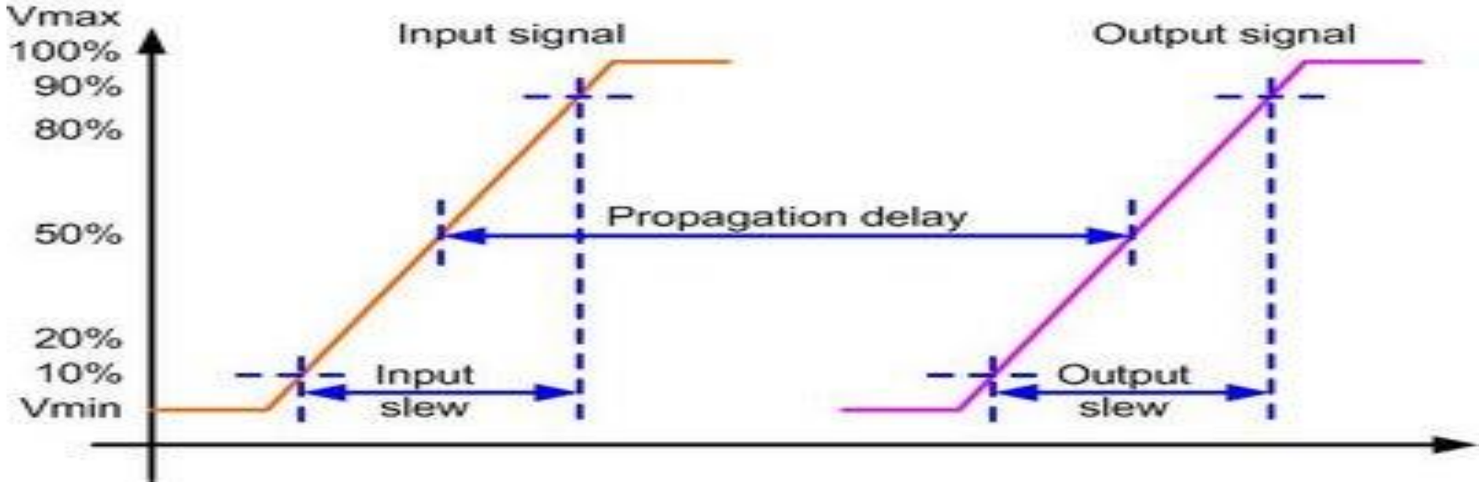


Library		Output load (pF)	
		0.05	0.1
Input Trans (ns)	0.5	0.1	0.05
	0.25	0.25	0.3

Timing Analysis – Cell delay

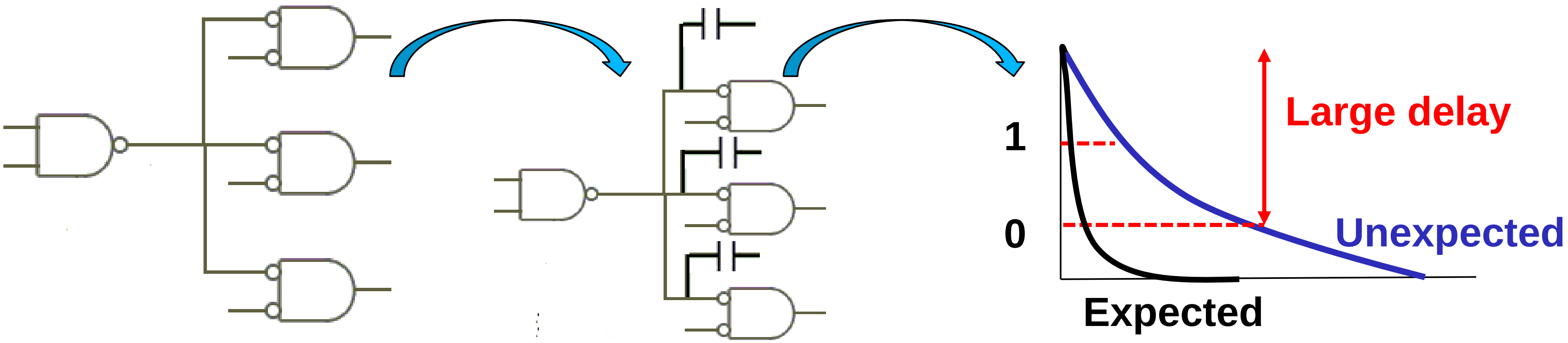


The input transition and output load effect the cell delay
→ increase the cell transition

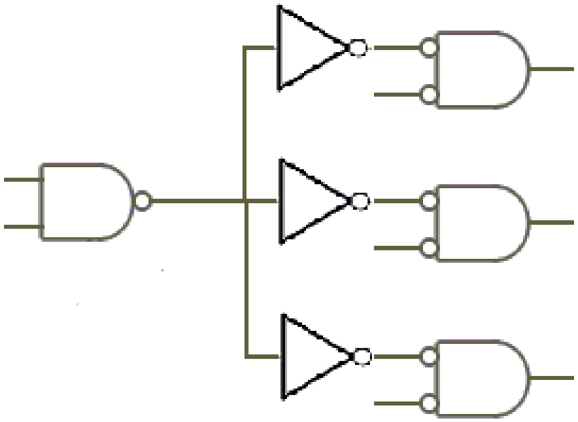


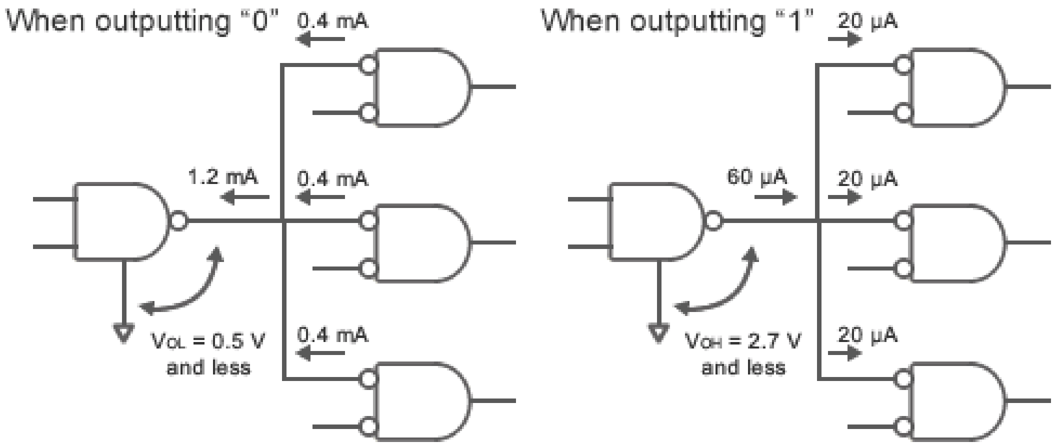
Timing Analysis – Cell delay

Fan-out = 3



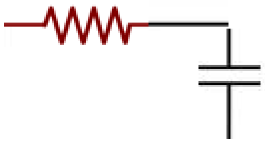
Solution





$$\text{TTL Fan-out} = \frac{\text{L output current } (I_{OL})}{\text{L input current } (I_{IL})} , \frac{\text{H output current } (I_{OH})}{\text{H input current } (I_{IH})}$$

Power Analysis



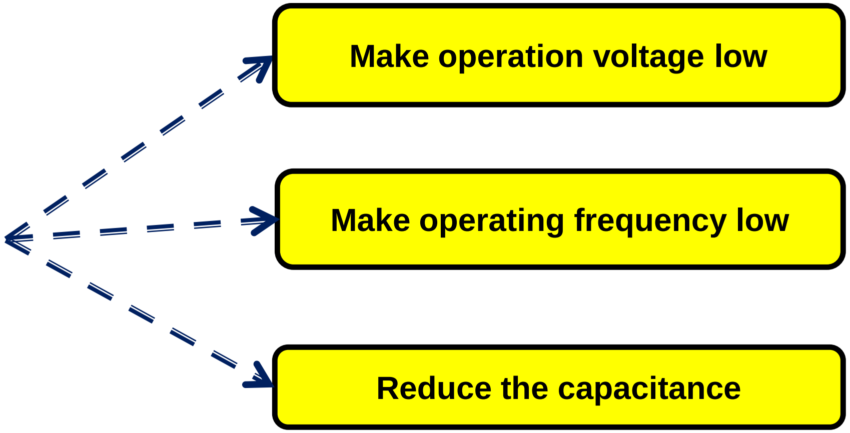
P = V.I

= V.Q/T

= V.C.V/T

= V.C.V.f

= f.C.V.V



Make operation voltage low

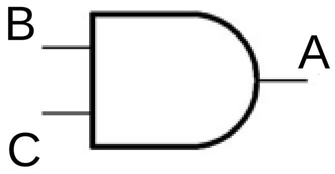
Make operating frequency low

Reduce the capacitance

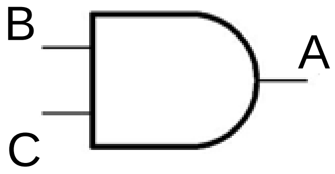
Static Timing Analysis

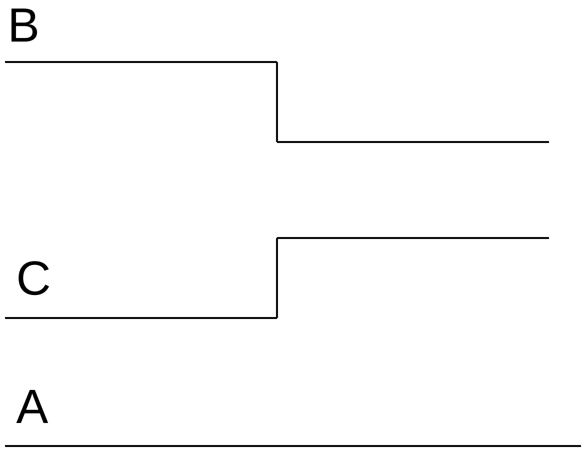
- ❖Dynamic vs. Static Timing Analysis
- ❖Synthesis Step
- ❖Analysis
- ❖**Constraint**
- ❖Timing Violation
- ❖Using DC Tool in Command/GUI mode

Timing Constrains – Setup/Hold time

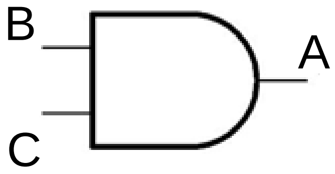


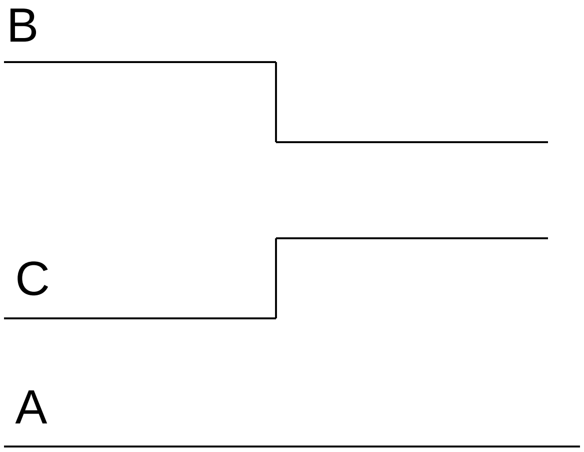
Timing Constrains – Setup/Hold time

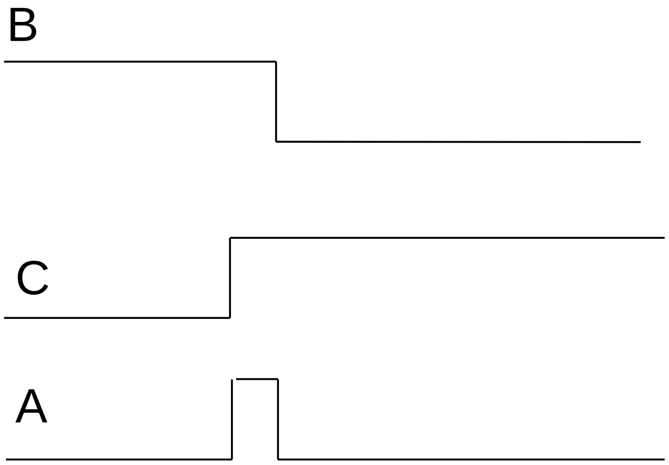




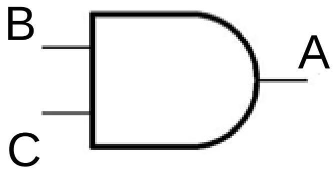
Timing Constrains – Setup/Hold time



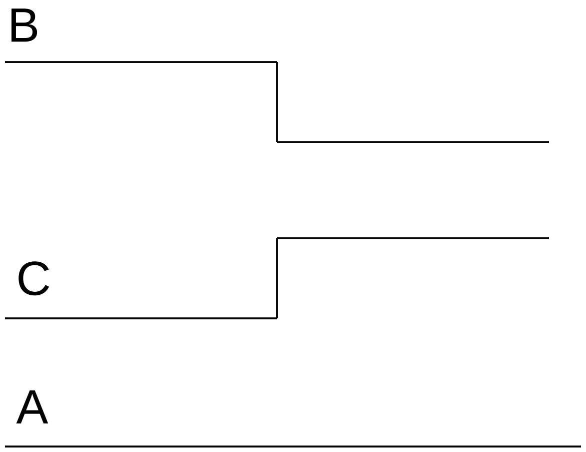


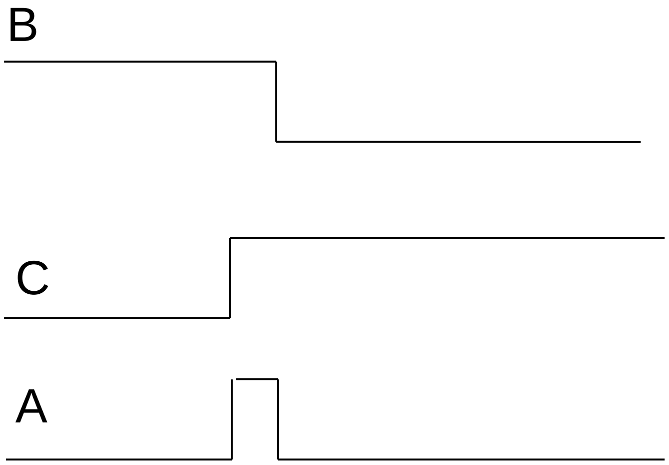


Timing Constrains – Setup/Hold time

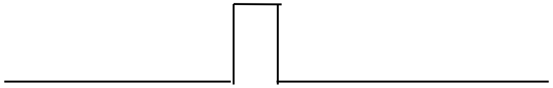


Hazard

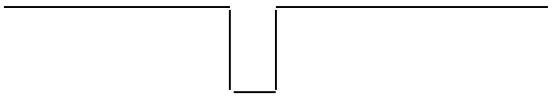




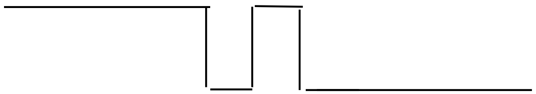
Static zero



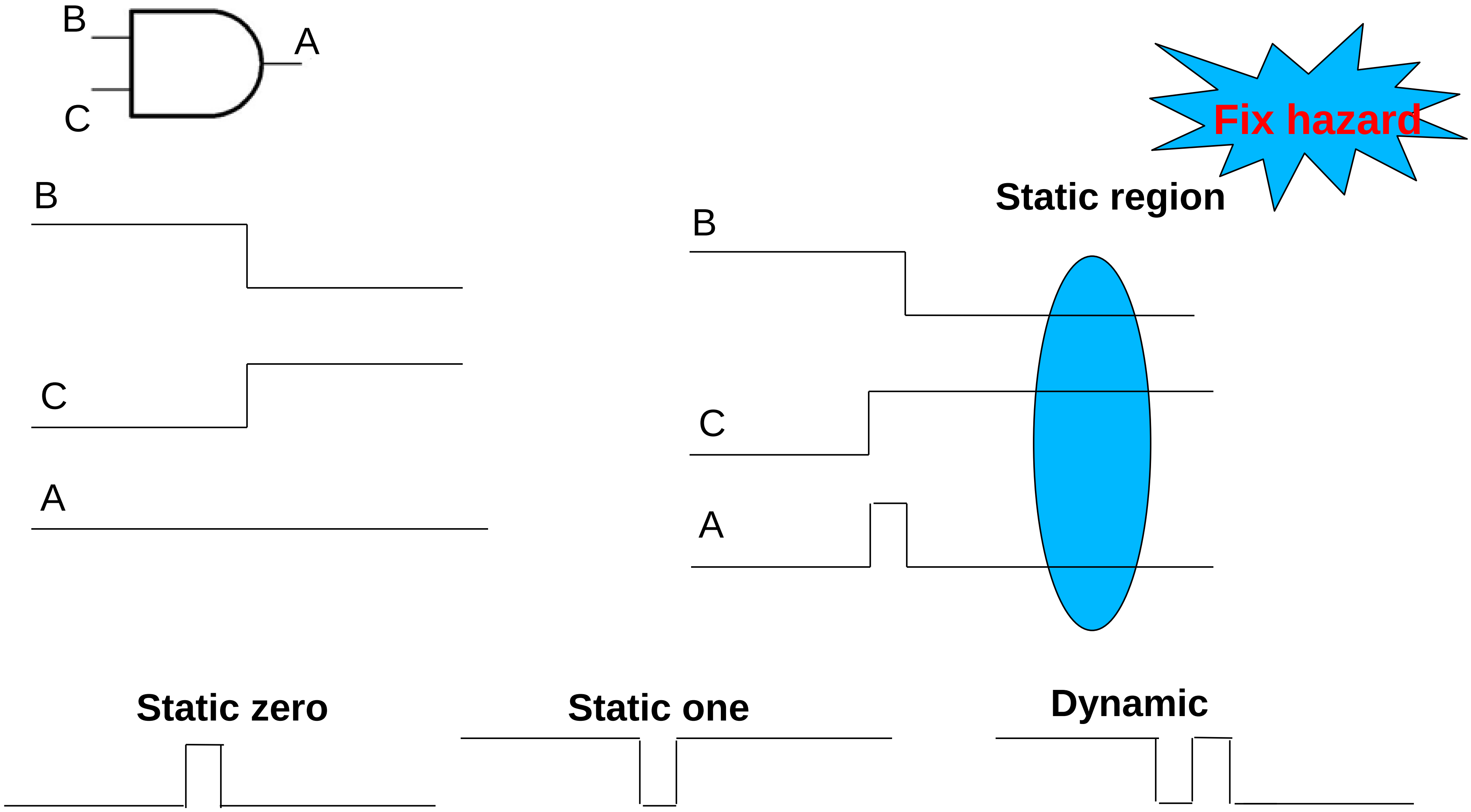
Static one



Dynamic

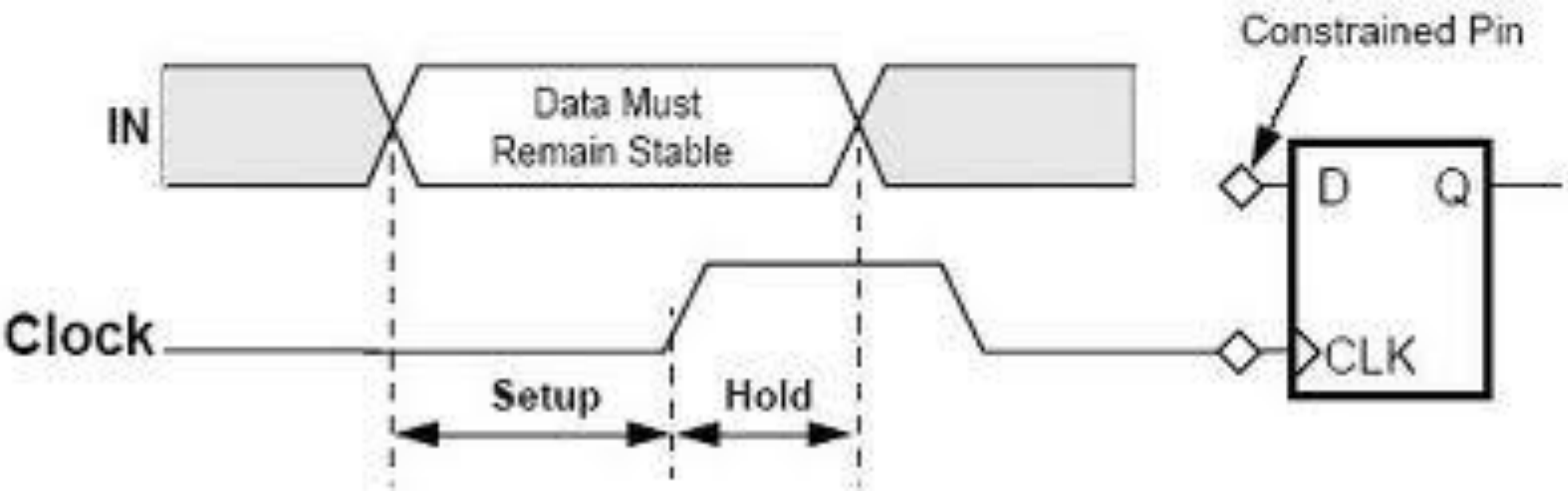


Timing Constrains – Setup/Hold time



→ **Need enough time for the static output**

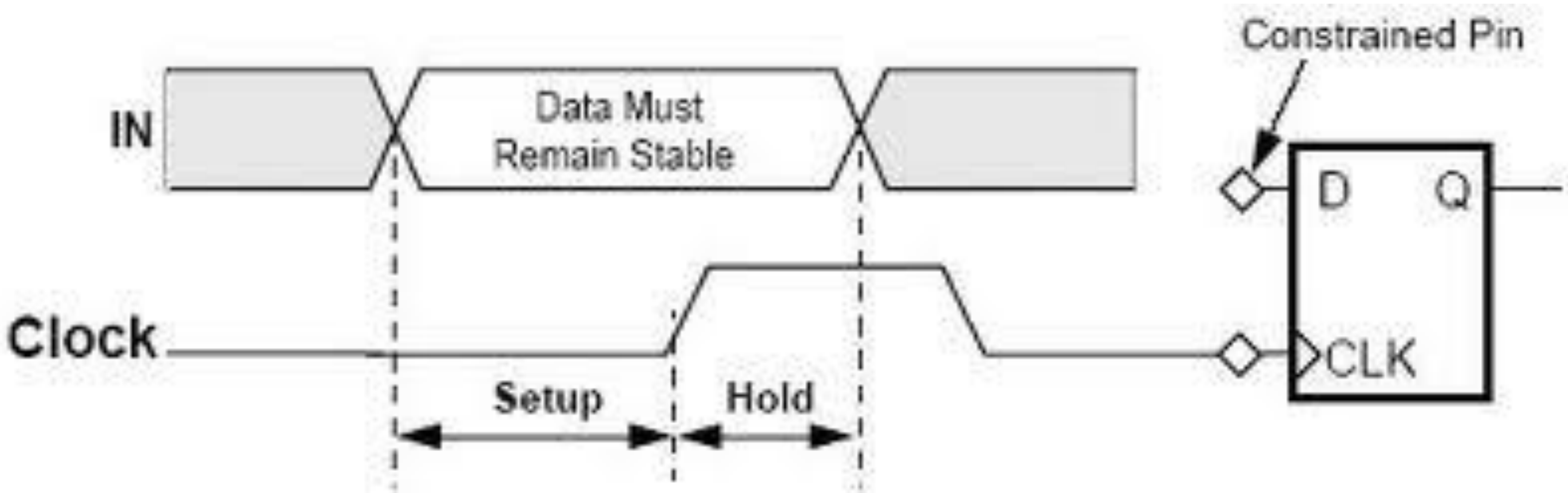
Timing Constraint – Setup/Hold time



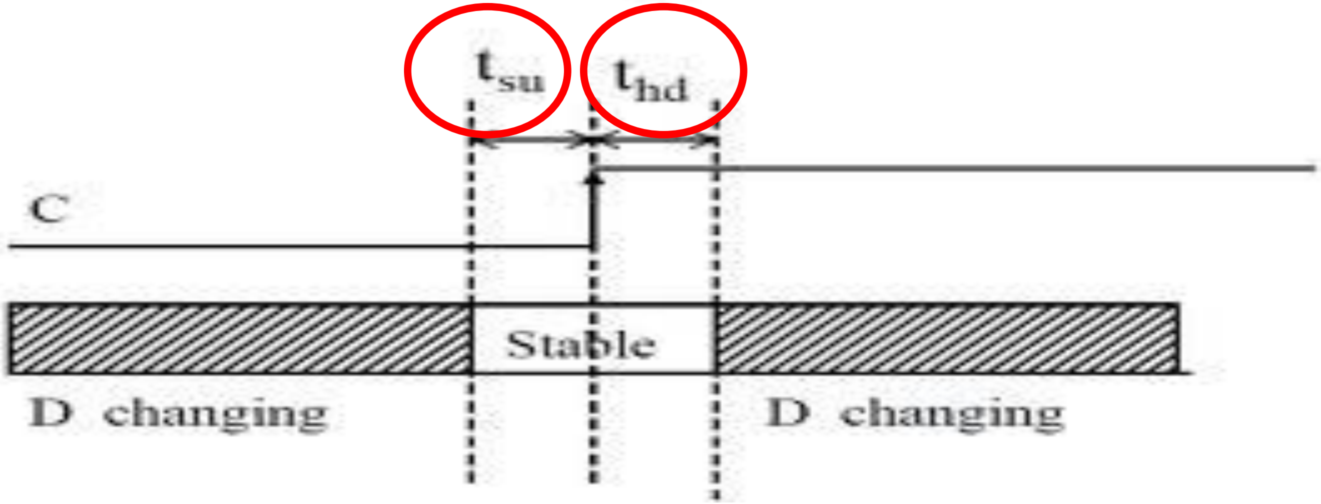
❖The **$hold** system task determines whether a data signal remains stable for a minimum specified time after a transition in an enabling signal.

❖The **$setup** system task determines whether a data signal remains stable before a transition in an enabling signal.

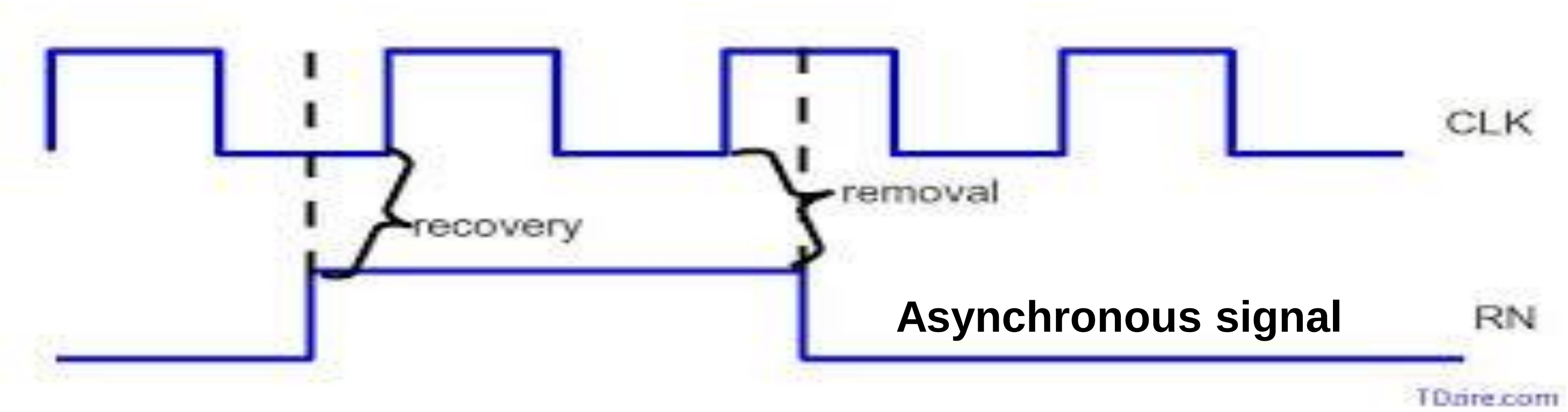
Timing Constrains – Setup/Hold time



Setup, Hold Time



Timing Constraint - Recovery & Removal



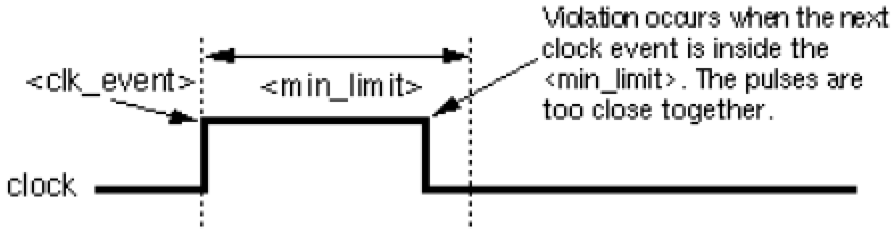
❖The $removal system task specifies a time constraint between an asynchronous control signal and a clock signal

❖The $recovery system task specifies a time constraint between an asynchronous control signal and a clock signal

Static Timing Analysis

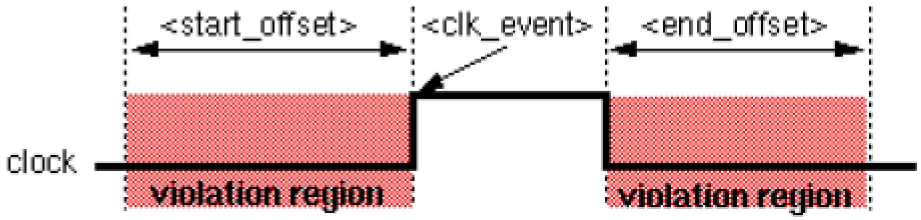
- ❖Dynamic vs. Static Timing Analysis
- ❖Synthesis Step
- ❖Analysis
- ❖Constraint
- ❖**Violation**
- ❖Using DC Tool in Command/GUI mode

Timing Violation

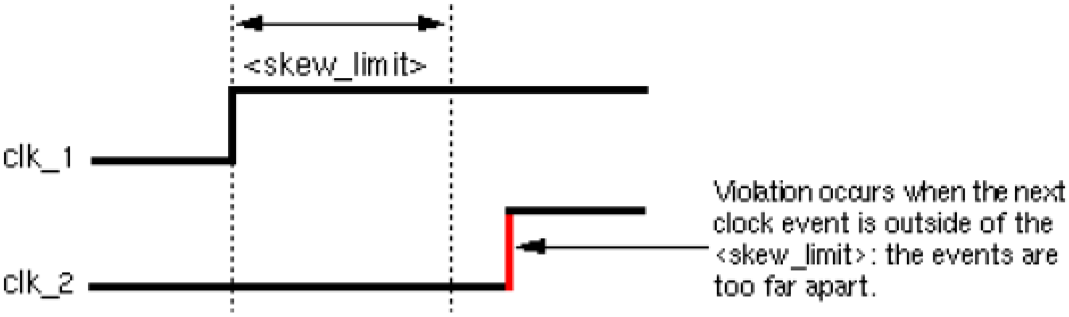


Violation occurs when the next clock event is inside the <min_limit>. The pulses are too close together.

Clock Width Violation

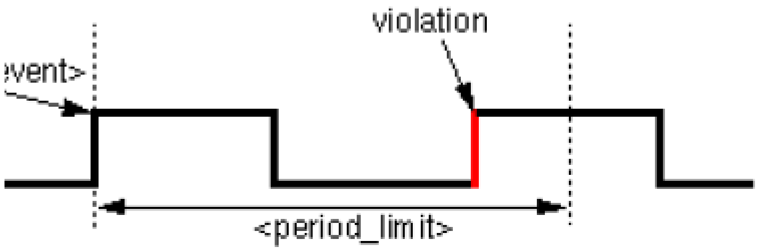


Clock Width Violation



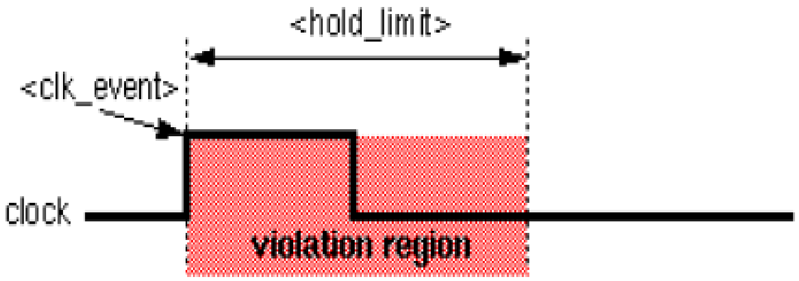
Violation occurs when the next clock event is outside of the <skew_limit>: the events are too far apart.

Clock Skew Violation

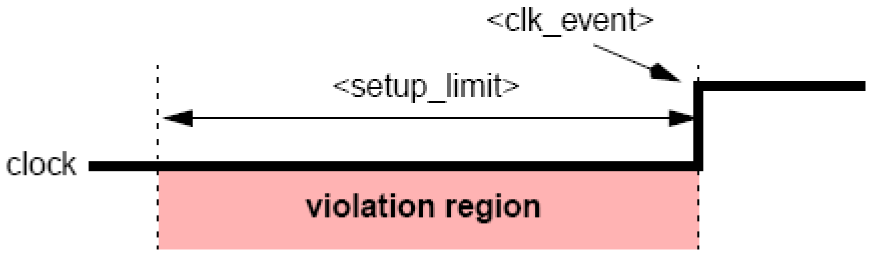


Clock Period Violation

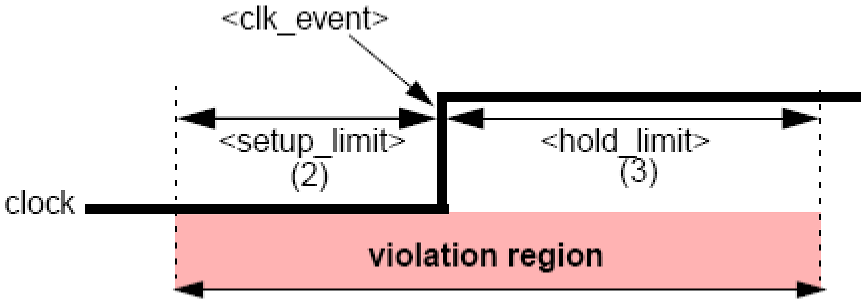
Timing Violation



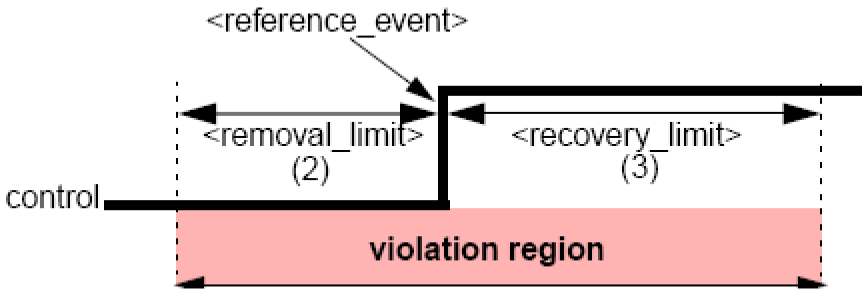
Hold Time Violation



Setup Time Violation



Setup/Hold time violation

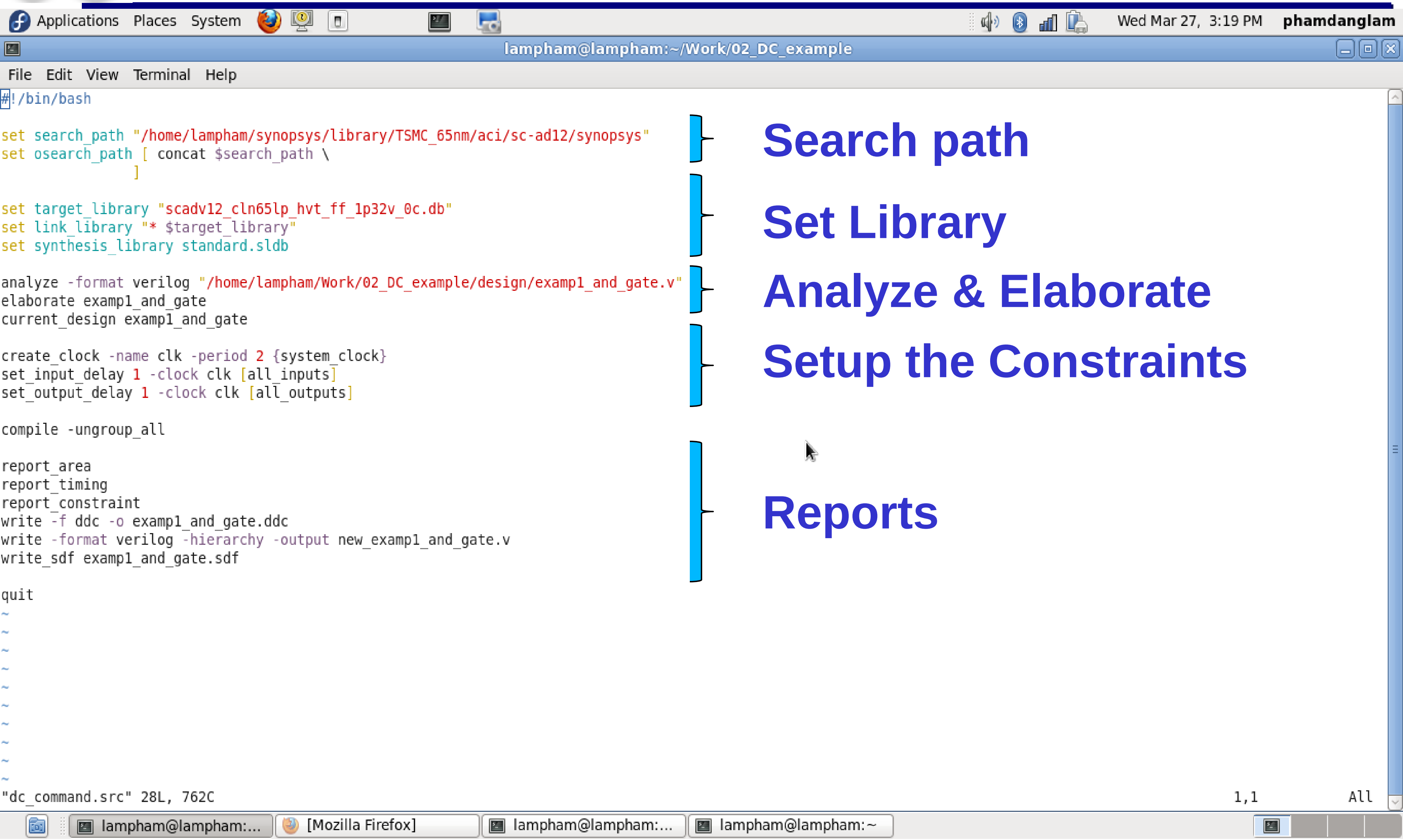


Removal/Recovery Violation

Static Timing Analysis

- Dynamic vs. Static Timing Analysis
- Synthesis Step
- Analysis
- Constraint
- Violation
- **Using DC Tool in Command/GUI mode**

Using DC Tool In Command Mode



```
#!/bin/bash

set search_path "/home/lampham/synopsys/library/TSMC_65nm/aci/sc-ad12/synopsys"
set osearch_path [ concat $search_path \
                ]

set target_library "scadv12_cln65lp_hvt_ff_1p32v_0c.db"
set link_library "* $target_library"
set synthesis_library standard.sldb

analyze -format verilog "/home/lampham/Work/02_DC_example/design/exampl_and_gate.v"
elaborate exampl_and_gate
current_design exampl_and_gate

create_clock -name clk -period 2 {system_clock}
set_input_delay 1 -clock clk [all_inputs]
set_output_delay 1 -clock clk [all_outputs]

compile -ungroup_all

report_area
report_timing
report_constraint
write -f ddc -o exampl_and_gate.ddc
write -format verilog -hierarchy -output new_exampl_and_gate.v
write_sdf exampl_and_gate.sdf

quit
~
~
~
~
~
~
~
~
~
~
"dc_command.src" 28L, 762C                                    1,1           All
```

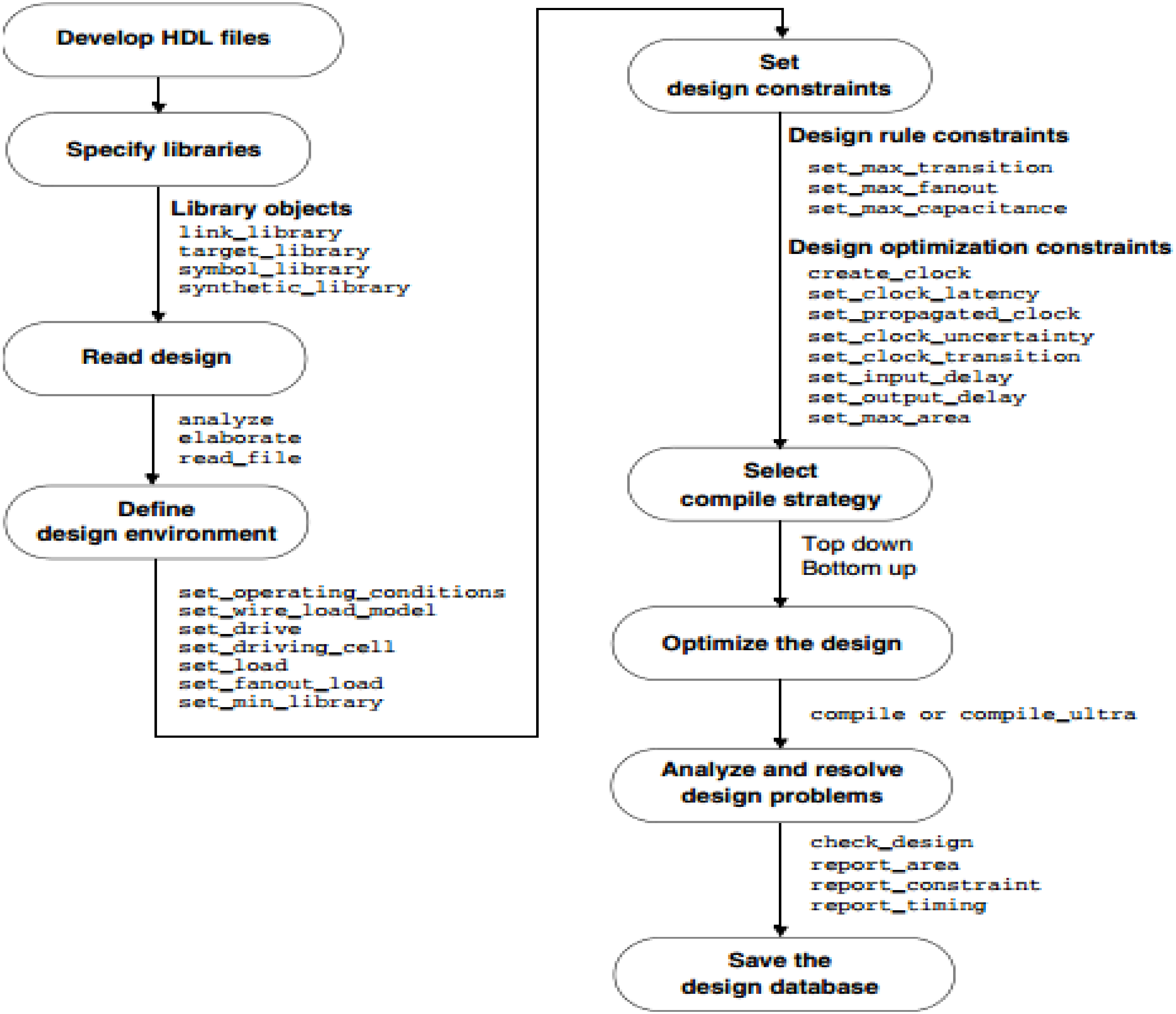
Search path

Set Library

Analyze & Elaborate

Setup the Constraints

Reports



Develop HDL files

Specify libraries

Library objects

```
link_library
target_library
symbol_library
synthetic_library
```

Read design

```
analyze
elaborate
read_file
```

Define design environment

```
set_operating_conditions
set_wire_load_model
set_drive
set_driving_cell
set_load
set_fanout_load
set_min_library
```

Set design constraints

Design rule constraints

```
set_max_transition
set_max_fanout
set_max_capacitance
```

Design optimization constraints

```
create_clock
set_clock_latency
set_propagated_clock
set_clock_uncertainty
set_clock_transition
set_input_delay
set_output_delay
set_max_area
```

Select compile strategy

Top down
Bottom up

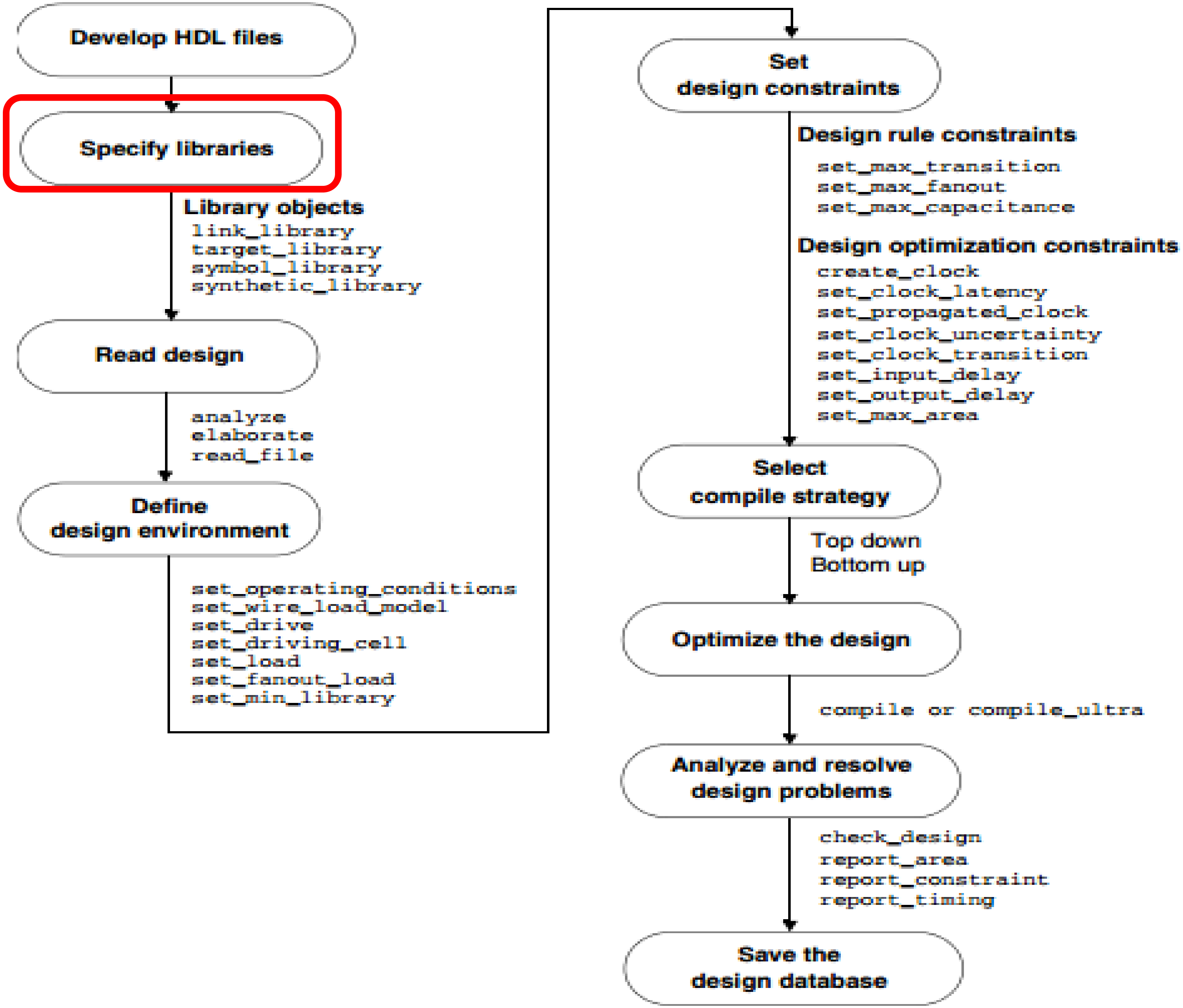
Optimize the design

```
compile or compile_ultra
```

Analyze and resolve design problems

```
check_design
report_area
report_constraint
report_timing
```

Save the design database



Develop HDL files

Specify libraries

Library objects

link_library
target_library
symbol_library
synthetic_library

Read design

analyze
elaborate
read_file

Define
design environment

set_operating_conditions
set_wire_load_model
set_drive
set_driving_cell
set_load
set_fanout_load
set_min_library

Set
design constraints

Design rule constraints

set_max_transition
set_max_fanout
set_max_capacitance

Design optimization constraints

create_clock
set_clock_latency
set_propagated_clock
set_clock_uncertainty
set_clock_transition
set_input_delay
set_output_delay
set_max_area

Select
compile strategy

Top down
Bottom up

Optimize the design

compile or compile_ultra

Analyze and resolve
design problems

check_design
report_area
report_constraint
report_timing

Save the
design database

Using DC Tool In Command Mode

- **Target Library:** Design Compiler uses the target library to build a circuit. During mapping, Design Compiler selects functionally correct gates from the targetlibrary. It also calculates the timing of the circuit, using the vendor-supplied timing data for these gates. Use the target_libraryvariable to specify the target library.

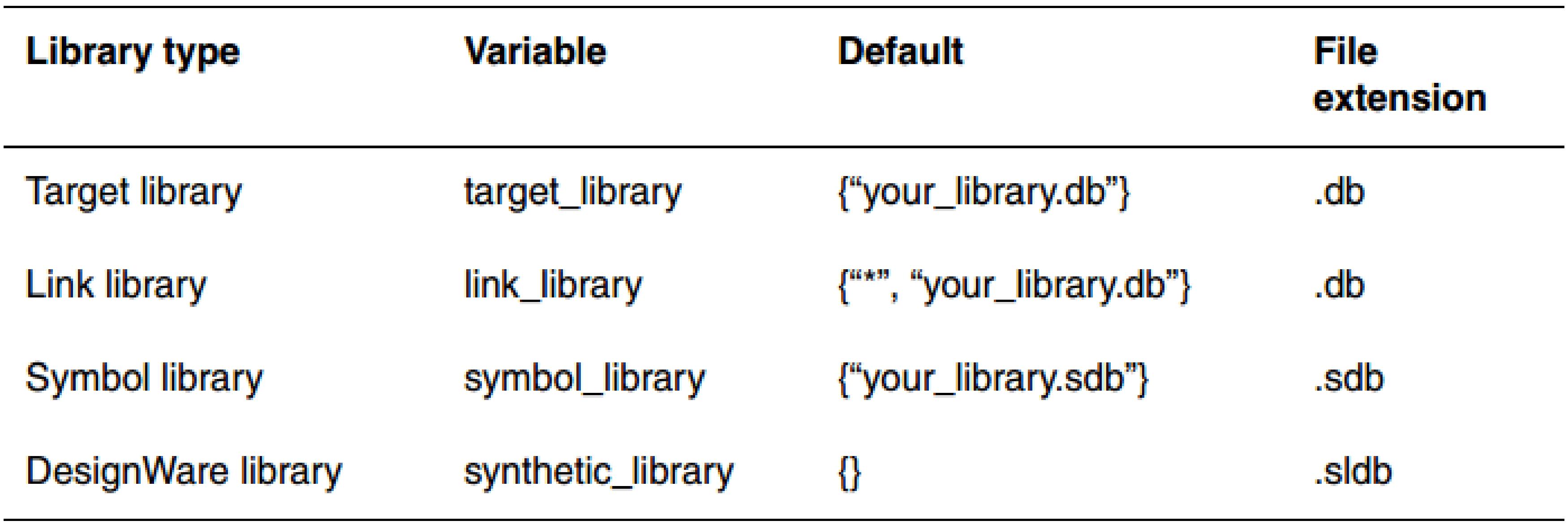
- **Link Library:** Design Compiler uses the link library to resolve references. For a design to be complete, it must connect to all the library components and designs it references. This process is called linking the design or resolving references. During the linking process, Design Compiler uses the link_library system variable, the local_link_library attribute, and the search_pathsystem variable to resolve references

- **Symbol libraries:** Contain definitions of the graphic symbols that represent library cells in the design schematics. Semiconductor vendors maintain and distribute the symbol librariesThe symbol library defines the symbols for schematic viewing of the design. You need this library if you intend to use the Design Vision GUI.

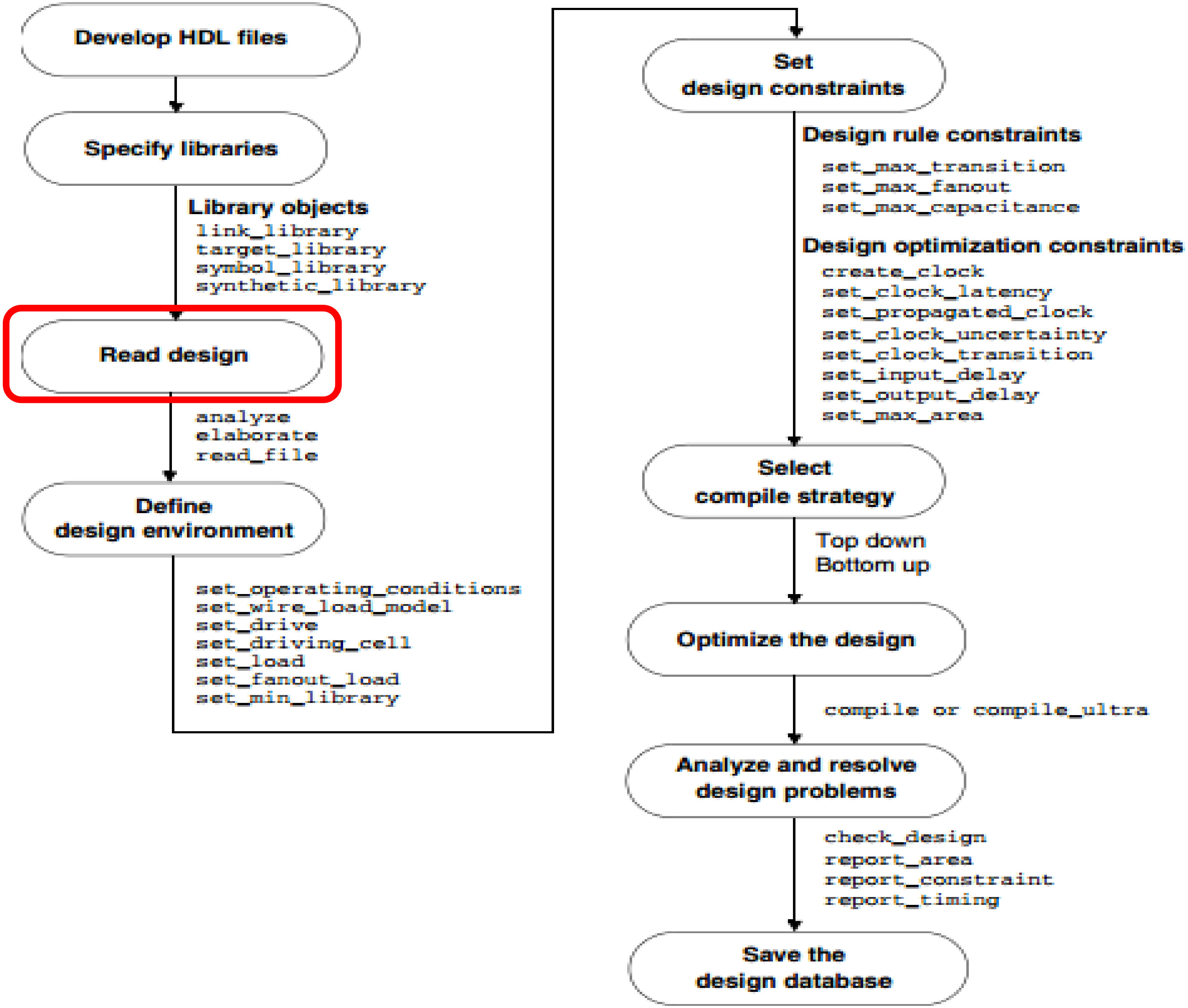
- **DesignWare Library**: A DesignWare library is a collection of reusable circuit-design building blocks (components) that are tightly integrated into the Synopsys synthesis environment. During synthesis, Design Compiler selects the right component with the best speed and area optimization from the DesignWare Library. For more information, see the DesignWare Library documentation.

Using DC Tool In Command Mode

```
set target_library lsi_10k.db
set symbol_library lsi_10k.sdb
set synthetic_library dw_foundation.sldb
set link_library “* $target_library $synthetic library”
set search_path [concat $search_path ./src]
set designer “test.v"
```



Library type	Variable	Default	File extension
Target library	target_library	{“your_library.db”}	.db
Link library	link_library	{“*”, “your_library.db”}	.db
Symbol library	symbol_library	{“your_library.sdb”}	.sdb
DesignWare library	synthetic_library	{}	.sldb



Develop HDL files

Specify libraries

Library objects

```
link_library
target_library
symbol_library
synthetic_library
```

Read design

```
analyze
elaborate
read_file
```

Define design environment

```
set_operating_conditions
set_wire_load_model
set_drive
set_driving_cell
set_load
set_fanout_load
set_min_library
```

Set design constraints

Design rule constraints

```
set_max_transition
set_max_fanout
set_max_capacitance
```

Design optimization constraints

```
create_clock
set_clock_latency
set_propagated_clock
set_clock_uncertainty
set_clock_transition
set_input_delay
set_output_delay
set_max_area
```

Select compile strategy

Top down
Bottom up

Optimize the design

```
compile or compile_ultra
```

Analyze and resolve design problems

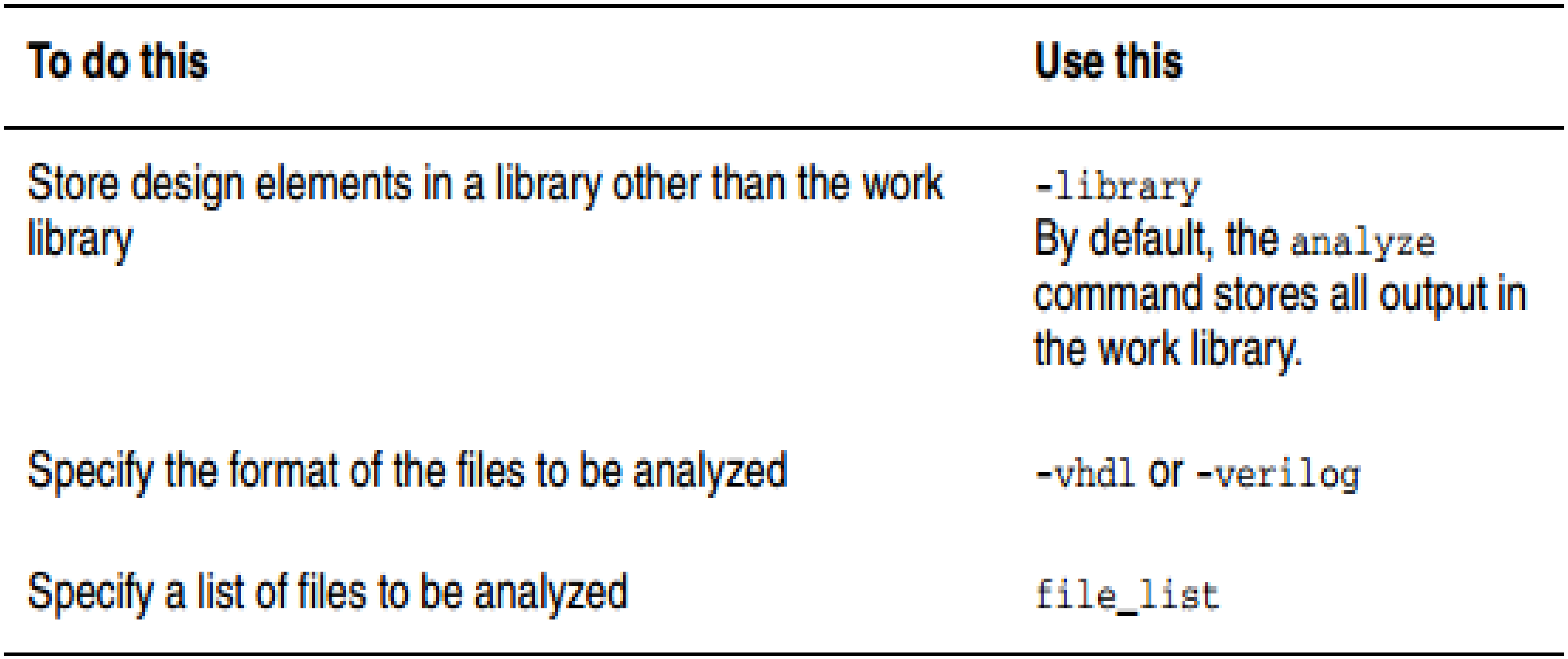
```
check_design
report_area
report_constraint
report_timing
```

Save the design database

Using DC Tool In Command Mode

The **analyze** command does the following:

- Reads an HDL source file
- Checks it for errors (without building generic logic for the design)
- Creates HDL library objects in an HDL-independent intermediate format
- Stores the intermediate files in a location you define

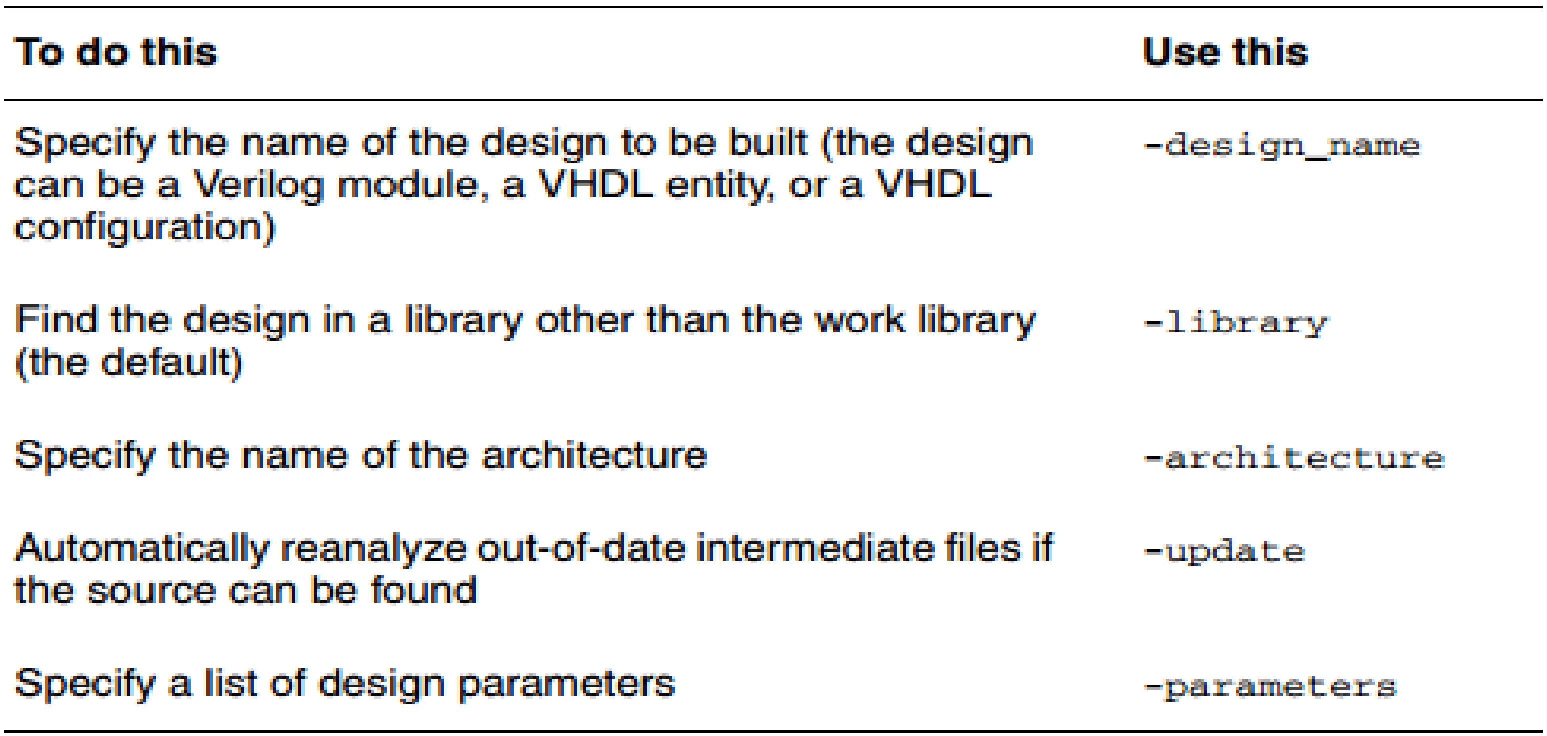


To do this	Use this
Store design elements in a library other than the work library	`-library` By default, the `analyze` command stores all output in the work library.
Specify the format of the files to be analyzed	`-vhdl` or `-verilog`
Specify a list of files to be analyzed	`file_list`

Using DC Tool In Command Mode

The **elaborate** command does the following:

- Translates the design into a technology-independent design (GTECH) from the intermediate files produced during analysis
- Allows changing of parameter values defined in the source code
- Allows VHDL architecture selection
- Replaces the HDL arithmetic operators in the code with DesignWare components
- Automatically executes the link command, which resolves design references



To do this	Use this
Specify the name of the design to be built (the design can be a Verilog module, a VHDL entity, or a VHDL configuration)	`-design_name`
Find the design in a library other than the work library (the default)	`-library`
Specify the name of the architecture	`-architecture`
Automatically reanalyze out-of-date intermediate files if the source can be found	`-update`
Specify a list of design parameters	`-parameters`

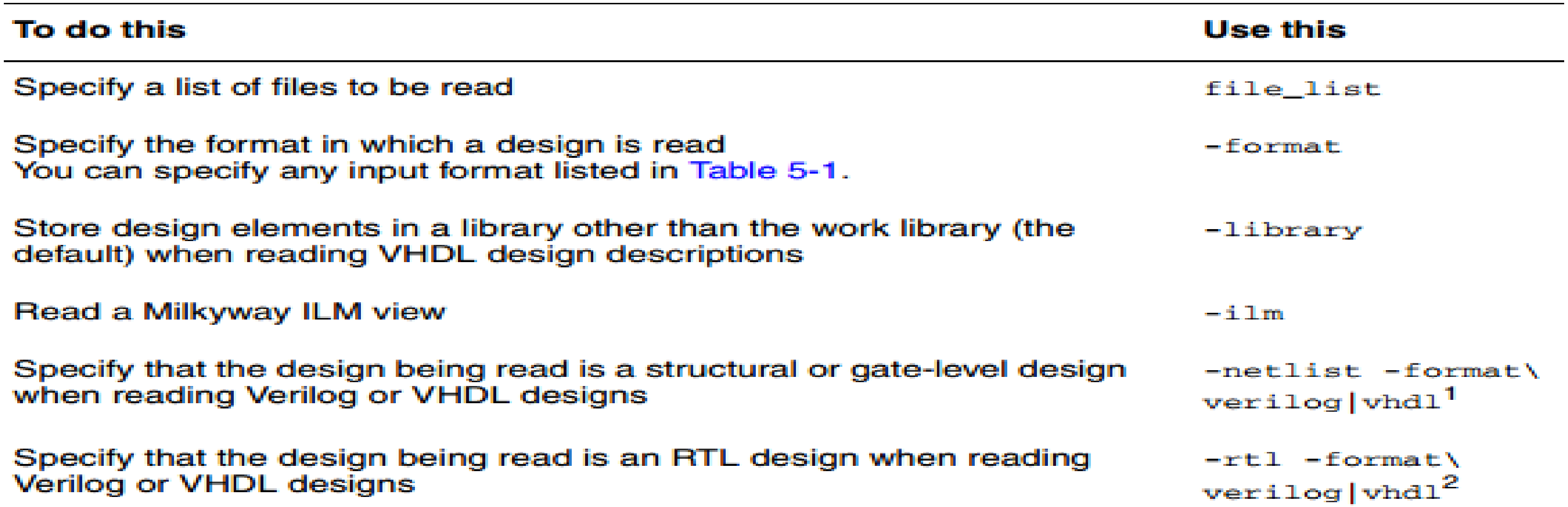
Using DC Tool In Command Mode

The **read_file** command does the following:

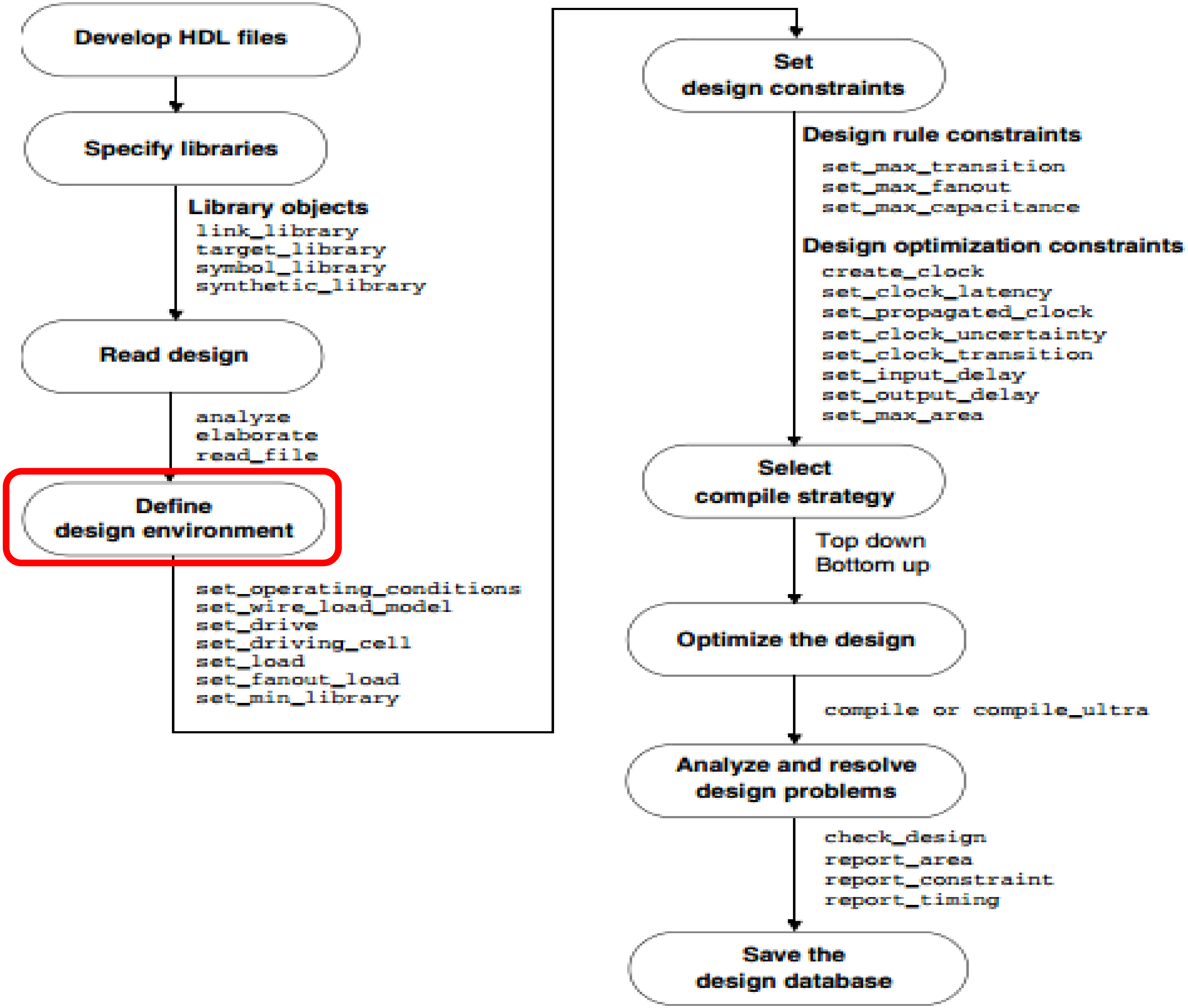
- Reads several different formats
- Performs the same operations as analyzeand elaboratein a single step
- Creates .mr and .st intermediate files for VHDL
- Does not execute the linkcommand automatically (see “Linking Designs” on page 5-13)
- Does not create any intermediate files for Verilog

The following **formats** are supported:

- ddc format: Uses the .ddc extension
- db format: Uses the .db, .sldb, .sdb, .db.gz, .sldb.gz, and .sdb.gz extensions
- Verilog format: Uses the .v, .verilog, .v.gz, and .verilog.gz extensions
- SystemVerilog: Uses the .sv, .sverilog, .sv.gz, and .sverilog.gz extensions
- VHDL: Uses the .vhd, .vhdl, vhd.gz, and .vhdl.gz extensions



To do this	Use this
Specify a list of files to be read	`file_list`
Specify the format in which a design is read You can specify any input format listed in Table 5-1.	`-format`
Store design elements in a library other than the work library (the default) when reading VHDL design descriptions	`-library`
Read a Milkyway ILM view	`-ilm`
Specify that the design being read is a structural or gate-level design when reading Verilog or VHDL designs	`-netlist -format\ verilog|vhdl`[1]
Specify that the design being read is an RTL design when reading Verilog or VHDL designs	`-rtl -format\ verilog|vhdl`[2]



Develop HDL files

Specify libraries

Library objects

```
link_library
target_library
symbol_library
synthetic_library
```

Read design

```
analyze
elaborate
read_file
```

Define
design environment

```
set_operating_conditions
set_wire_load_model
set_drive
set_driving_cell
set_load
set_fanout_load
set_min_library
```

Set
design constraints

Design rule constraints

```
set_max_transition
set_max_fanout
set_max_capacitance
```

Design optimization constraints

```
create_clock
set_clock_latency
set_propagated_clock
set_clock_uncertainty
set_clock_transition
set_input_delay
set_output_delay
set_max_area
```

Select
compile strategy

Top down
Bottom up

Optimize the design

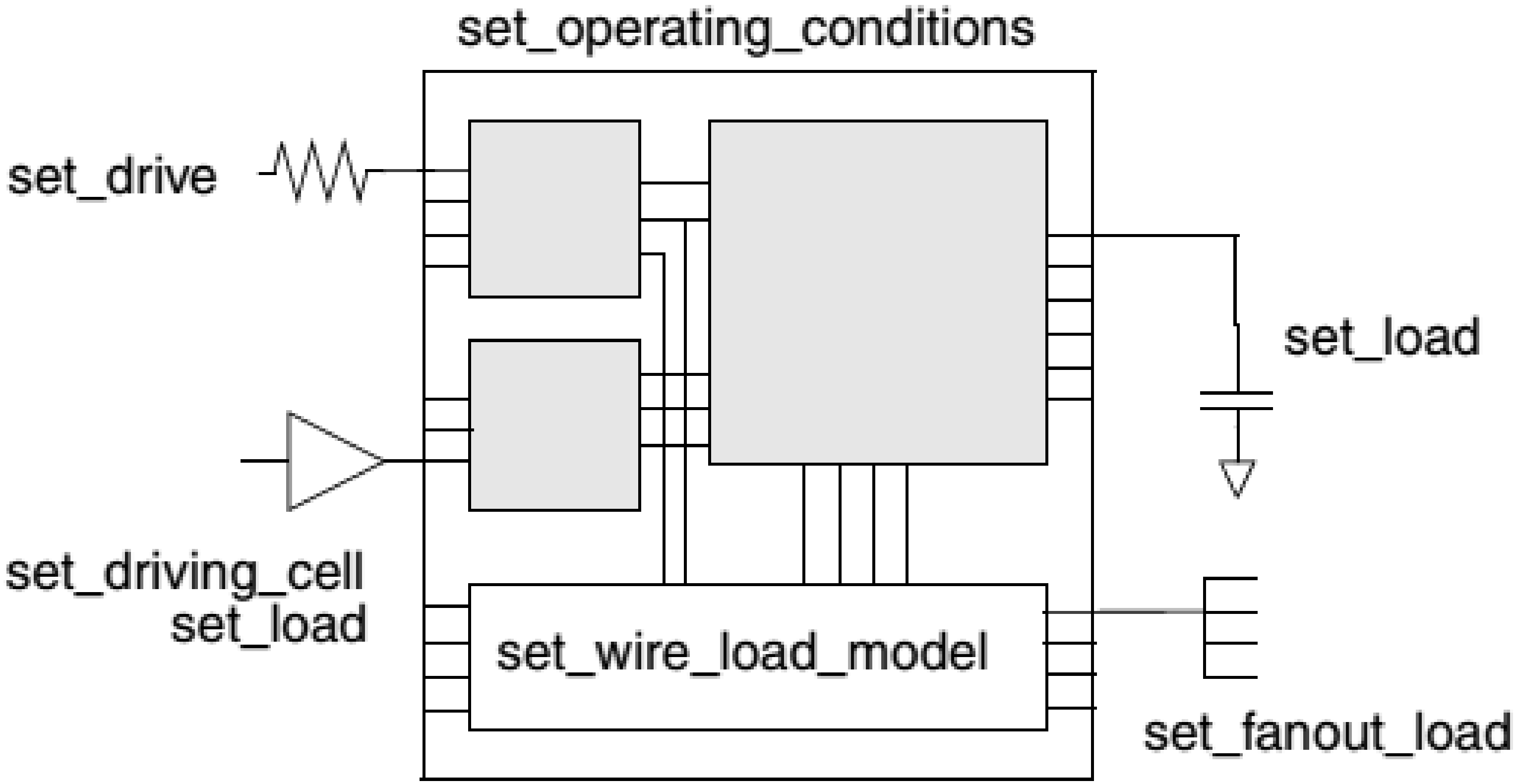
```
compile or compile_ultra
```

Analyze and resolve
design problems

```
check_design
report_area
report_constraint
report_timing
```

Save the
design database

Using DC Tool In Command Mode



Using DC Tool In Command Mode

- Operating temperature variation

  Temperature variation is unavoidable in the everyday operation of a design. Effects on performance caused by temperature fluctuations are most often handled as linear scaling effects, but some submicron silicon processes require nonlinear calculations.

- Supply voltage variation

  The design's supply voltage can vary from the established ideal value during day-to-day operation. Often a complex calculation (using a shift in threshold voltages) is employed, but a simple linear scaling factor is also used for logic-level performance calculations.

- Process variation

  This variation accounts for deviations in the semiconductor fabrication process. Usually process variation is treated as a percentage variation in the performance calculation.

```
dc_shell> read_file my_lib.db
dc_shell> report_lib my_lib
```

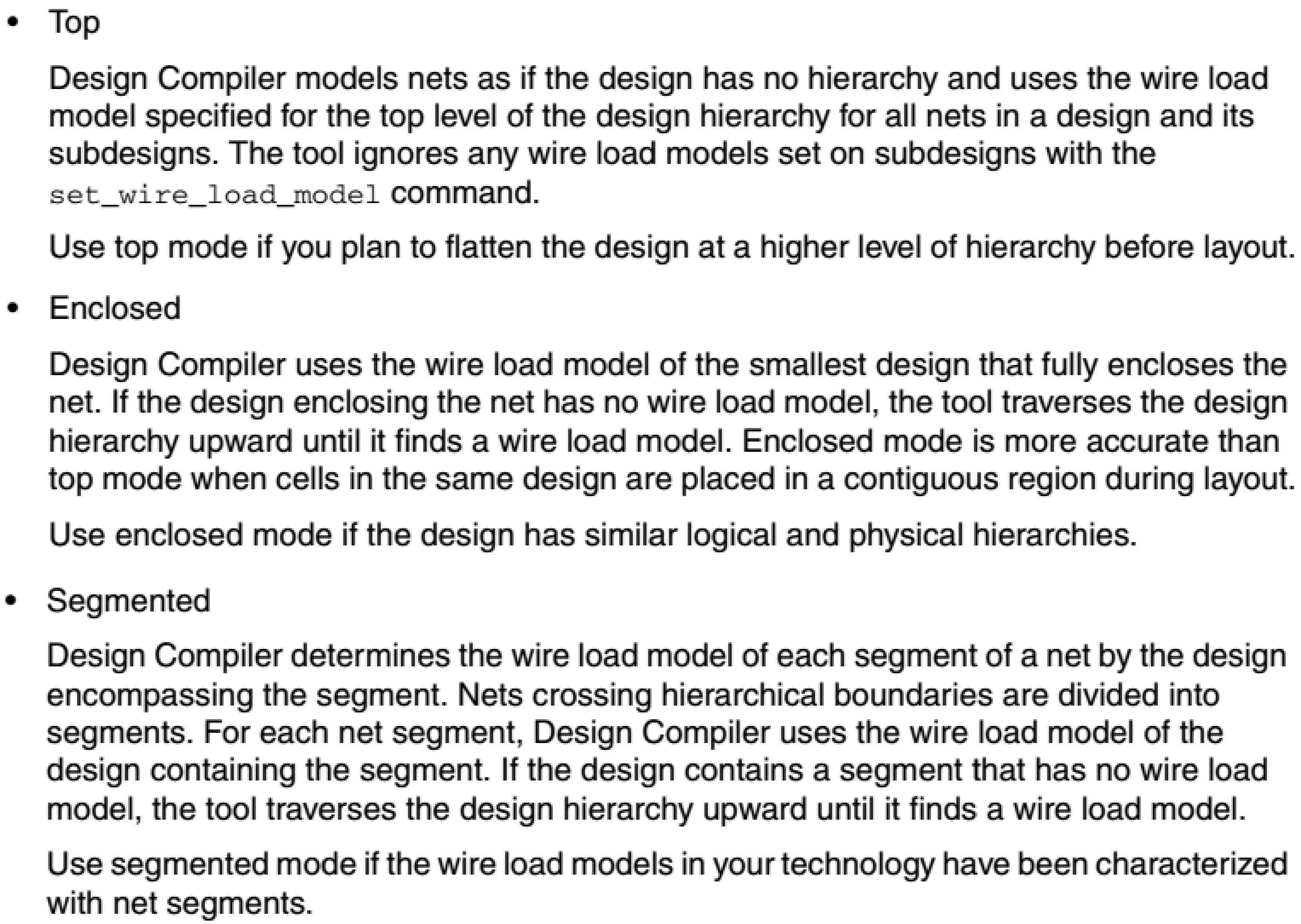
Using read_lib command to clarify the chosen operating condition

```
Operating Conditions:

    Name         Library          Process      Temp     Volt     Interconnect Model
    ---------------------------------------------------------------------------------
    WCCOM        my_lib              1.50      70.00    4.75      worst_case_tree
    WCIND        my_lib              1.50      85.00    4.75      worst_case_tree
    WCMIL        my_lib              1.50     125.00    4.50      worst_case_tree
```

For example, to set the operating conditions for the current design to worst-case commercial, enter

```
dc_shell> set_operating_conditions WCCOM -lib my_lib
```

Using DC Tool In Command Mode



- Top

  Design Compiler models nets as if the design has no hierarchy and uses the wire load model specified for the top level of the design hierarchy for all nets in a design and its subdesigns. The tool ignores any wire load models set on subdesigns with the `set_wire_load_model` command.

  Use top mode if you plan to flatten the design at a higher level of hierarchy before layout.

- Enclosed

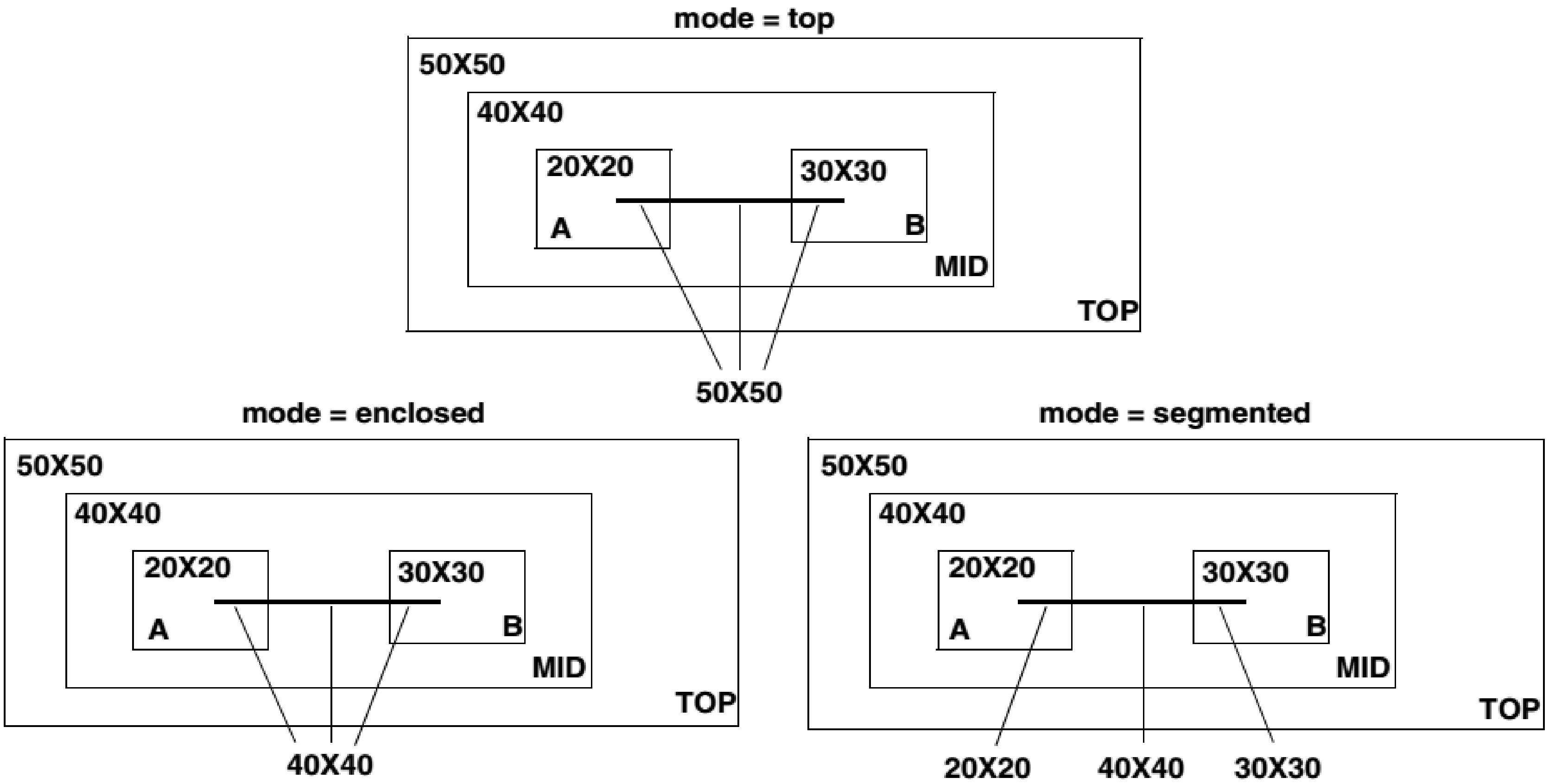
  Design Compiler uses the wire load model of the smallest design that fully encloses the net. If the design enclosing the net has no wire load model, the tool traverses the design hierarchy upward until it finds a wire load model. Enclosed mode is more accurate than top mode when cells in the same design are placed in a contiguous region during layout.

  Use enclosed mode if the design has similar logical and physical hierarchies.

- Segmented

  Design Compiler determines the wire load model of each segment of a net by the design encompassing the segment. Nets crossing hierarchical boundaries are divided into segments. For each net segment, Design Compiler uses the wire load model of the design containing the segment. If the design contains a segment that has no wire load model, the tool traverses the design hierarchy upward until it finds a wire load model.

  Use segmented mode if the wire load models in your technology have been characterized with net segments.



You can turn off automatic selection of the wire load model by setting the `auto_wire_load_selection` variable to false. For example, enter the following commands:

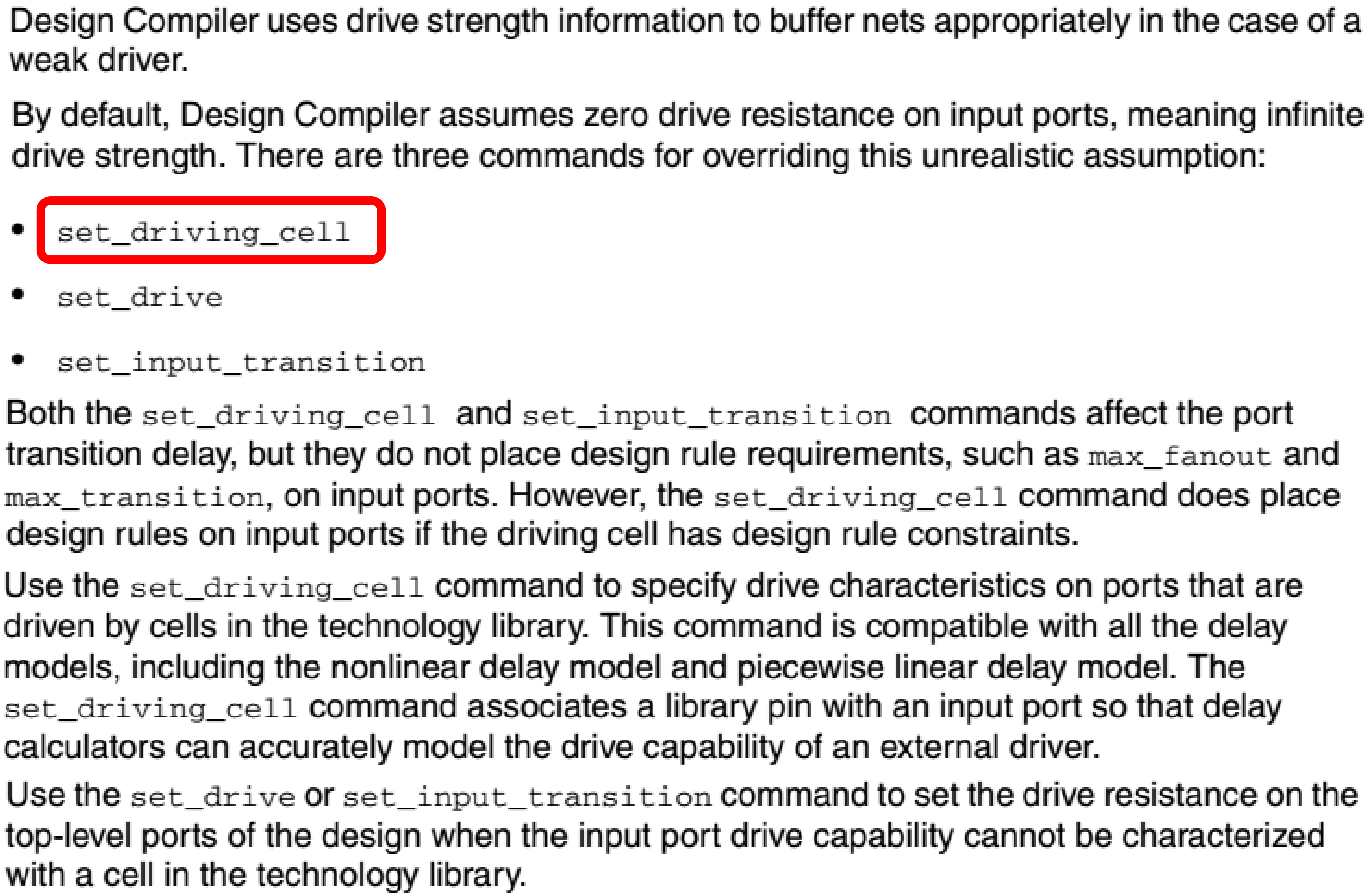
For example, to select the 10x10 wire load model, enter

```
dc_shell> set_wire_load_model "10x10"
```

To select the 10x10 wire load model and specify enclosed mode, enter

```
dc_shell> set_wire_load_mode enclosed
```

Using DC Tool In Command Mode



Design Compiler uses drive strength information to buffer nets appropriately in the case of a weak driver.

By default, Design Compiler assumes zero drive resistance on input ports, meaning infinite drive strength. There are three commands for overriding this unrealistic assumption:

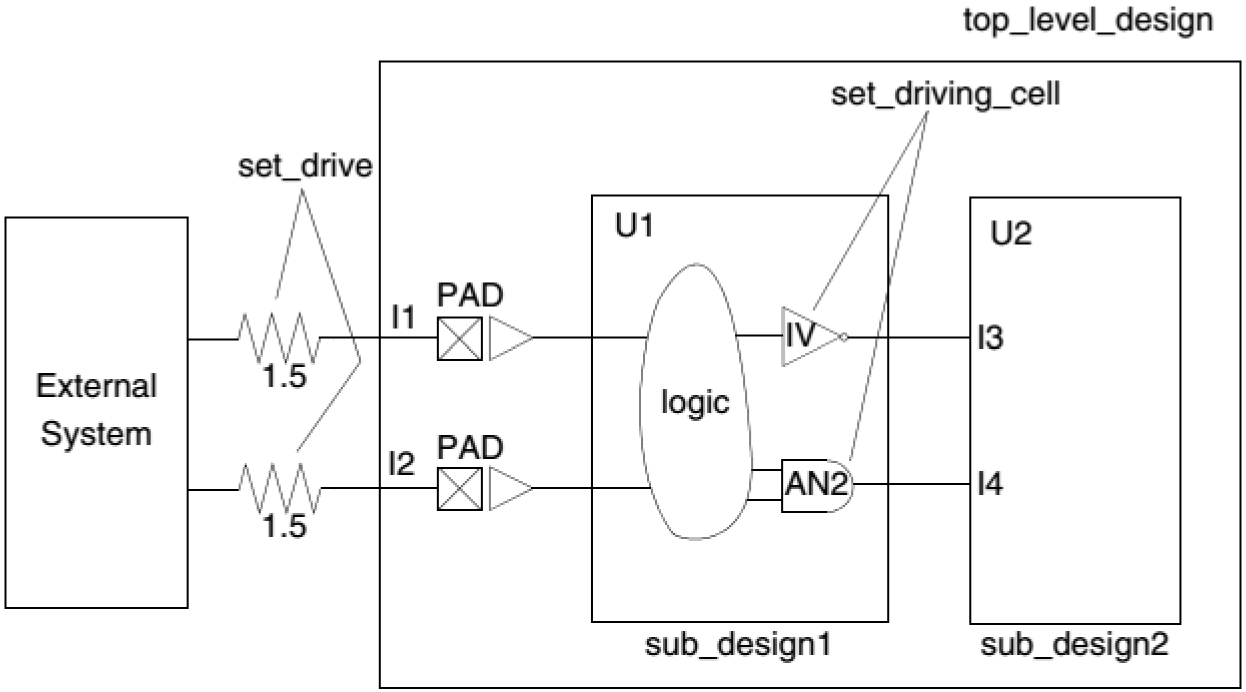
- `set_driving_cell`
- `set_drive`
- `set_input_transition`

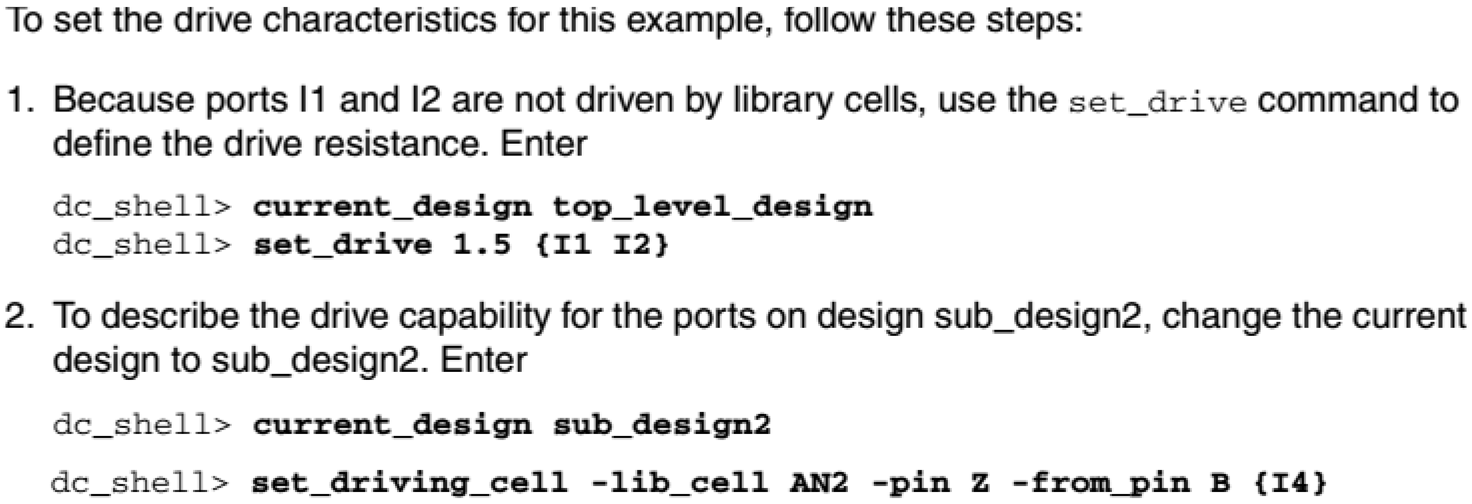
Both the `set_driving_cell` and `set_input_transition` commands affect the port transition delay, but they do not place design rule requirements, such as `max_fanout` and `max_transition`, on input ports. However, the `set_driving_cell` command does place design rules on input ports if the driving cell has design rule constraints.

Use the `set_driving_cell` command to specify drive characteristics on ports that are driven by cells in the technology library. This command is compatible with all the delay models, including the nonlinear delay model and piecewise linear delay model. The `set_driving_cell` command associates a library pin with an input port so that delay calculators can accurately model the drive capability of an external driver.

Use the `set_drive` or `set_input_transition` command to set the drive resistance on the top-level ports of the design when the input port drive capability cannot be characterized with a cell in the technology library.

Using DC Tool In Command Mode




To set the drive characteristics for this example, follow these steps:

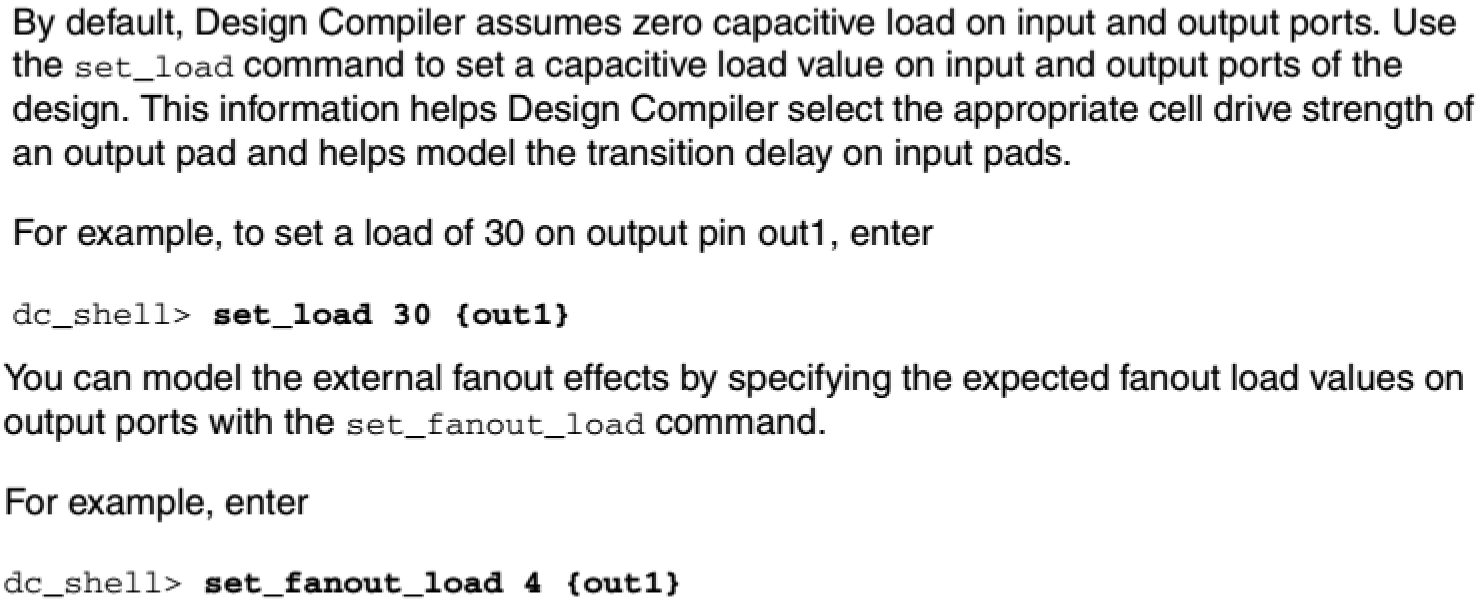
1. Because ports I1 and I2 are not driven by library cells, use the `set_drive` command to define the drive resistance. Enter

```
dc_shell> current_design top_level_design
dc_shell> set_drive 1.5 {I1 I2}
```

2. To describe the drive capability for the ports on design sub_design2, change the current design to sub_design2. Enter

```
dc_shell> current_design sub_design2

dc_shell> set_driving_cell -lib_cell AN2 -pin Z -from_pin B {I4}
```

Using DC Tool In Command Mode



By default, Design Compiler assumes zero capacitive load on input and output ports. Use the `set_load` command to set a capacitive load value on input and output ports of the design. This information helps Design Compiler select the appropriate cell drive strength of an output pad and helps model the transition delay on input pads.

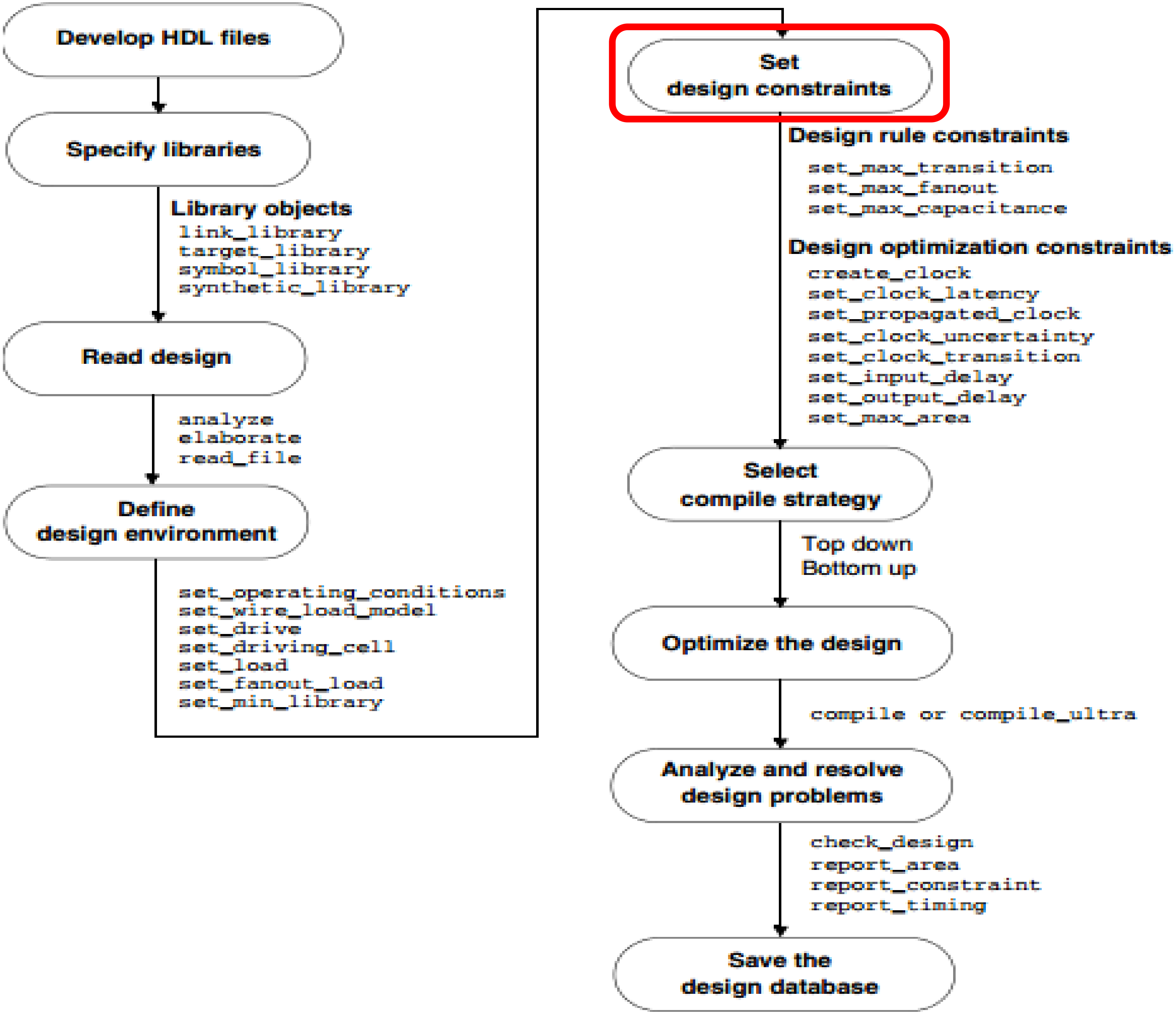
For example, to set a load of 30 on output pin out1, enter

```
dc_shell> set_load 30 {out1}
```

You can model the external fanout effects by specifying the expected fanout load values on output ports with the `set_fanout_load` command.

For example, enter

```
dc_shell> set_fanout_load 4 {out1}
```



Develop HDL files

Specify libraries

Library objects

```
link_library
target_library
symbol_library
synthetic_library
```

Read design

```
analyze
elaborate
read_file
```

Define design environment

```
set_operating_conditions
set_wire_load_model
set_drive
set_driving_cell
set_load
set_fanout_load
set_min_library
```

Set design constraints

Design rule constraints

```
set_max_transition
set_max_fanout
set_max_capacitance
```

Design optimization constraints

```
create_clock
set_clock_latency
set_propagated_clock
set_clock_uncertainty
set_clock_transition
set_input_delay
set_output_delay
set_max_area
```

Select compile strategy

Top down
Bottom up

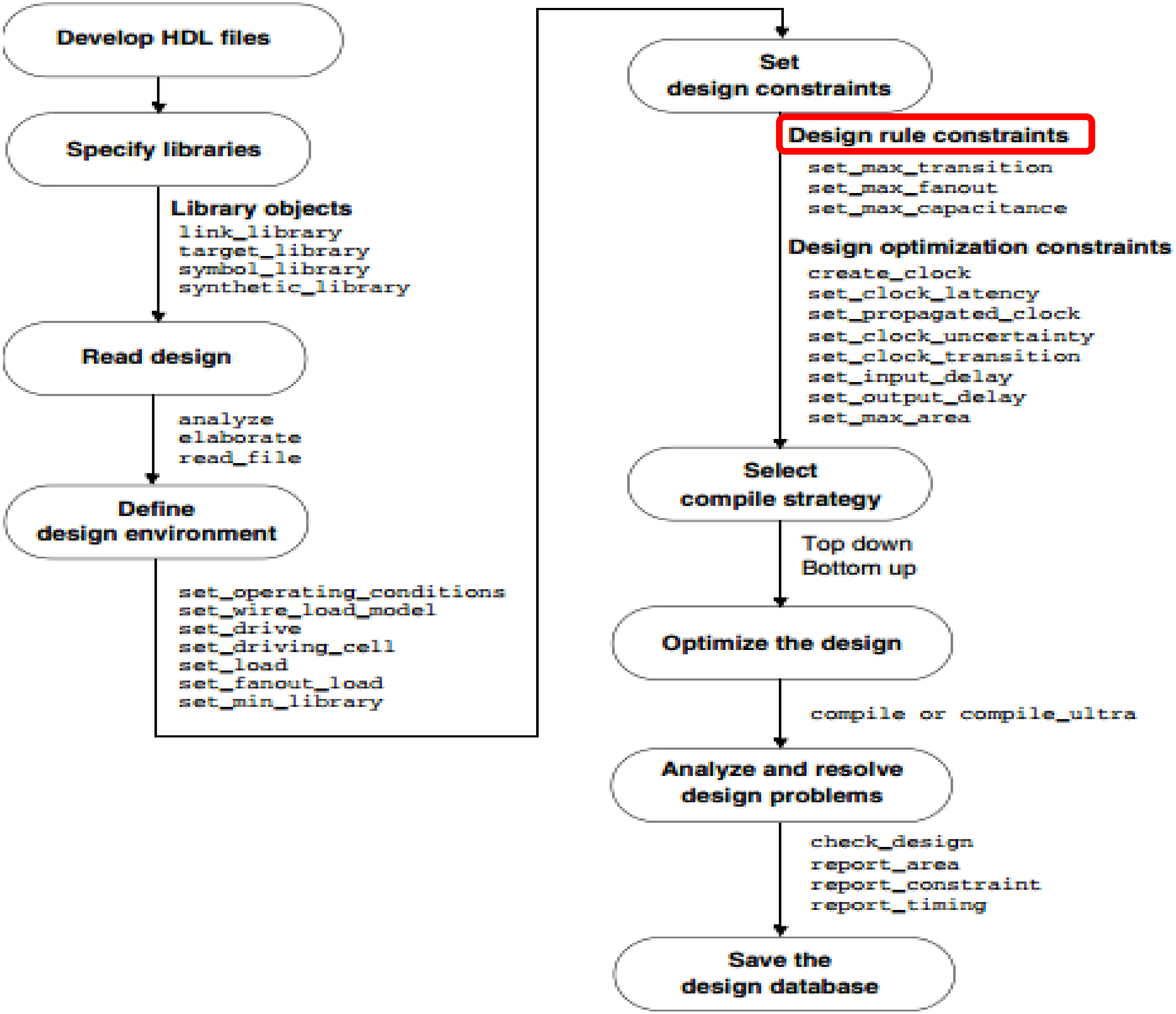
Optimize the design

```
compile or compile_ultra
```

Analyze and resolve design problems

```
check_design
report_area
report_constraint
report_timing
```

Save the design database



Develop HDL files

Specify libraries

Library objects

```
link_library
target_library
symbol_library
synthetic_library
```

Read design

```
analyze
elaborate
read_file
```

Define
design environment

```
set_operating_conditions
set_wire_load_model
set_drive
set_driving_cell
set_load
set_fanout_load
set_min_library
```

Set
design constraints

Design rule constraints

```
set_max_transition
set_max_fanout
set_max_capacitance
```

Design optimization constraints

```
create_clock
set_clock_latency
set_propagated_clock
set_clock_uncertainty
set_clock_transition
set_input_delay
set_output_delay
set_max_area
```

Select
compile strategy

Top down
Bottom up

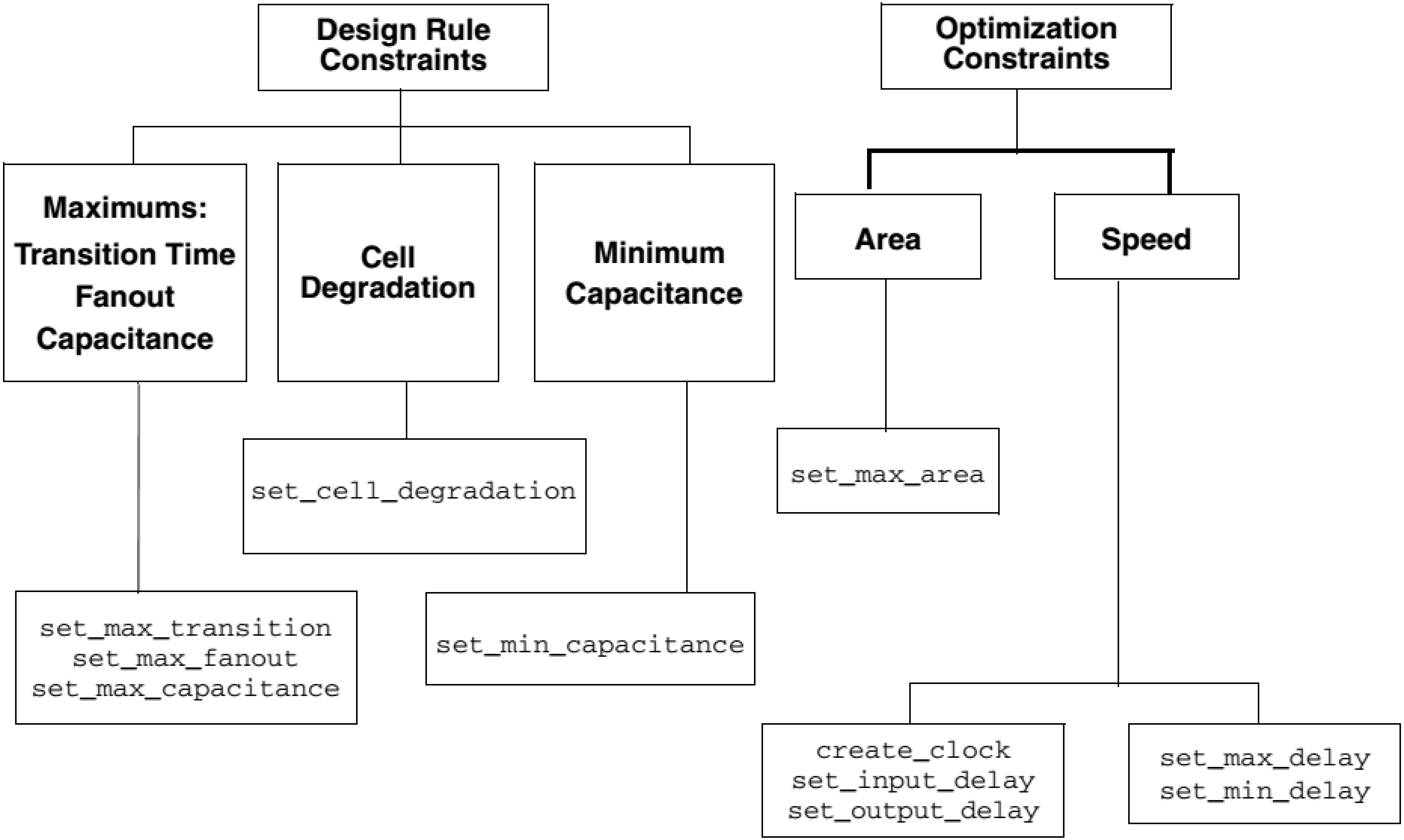
Optimize the design

```
compile or compile_ultra
```

Analyze and resolve
design problems

```
check_design
report_area
report_constraint
report_timing
```

Save the
design database



Design Rule Constraints

Maximums:
Transition Time
Fanout
Capacitance

```
set_max_transition
set_max_fanout
set_max_capacitance
```

Cell Degradation

```
set_cell_degradation
```

Minimum Capacitance

```
set_min_capacitance
```

Optimization Constraints

Area

```
set_max_area
```

Speed

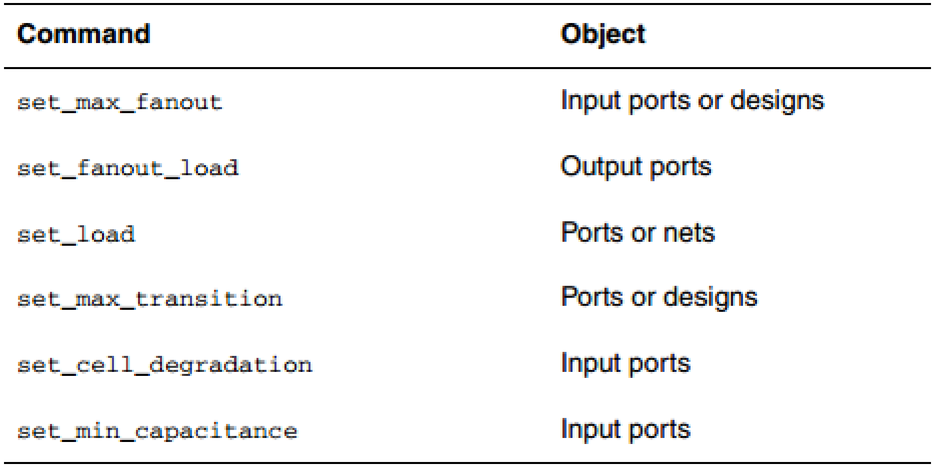
```
create_clock
set_input_delay
set_output_delay
```

```
set_max_delay
set_min_delay
```

Using DC Tool In Command Mode

The **design rule** constraints comprise

- **Maximum transition time**
- **Maximum fanout**
- **Minimum and maximum capacitance**
- **Cell degradation**



Command	Object
`set_max_fanout`	Input ports or designs
`set_fanout_load`	Output ports
`set_load`	Ports or nets
`set_max_transition`	Ports or designs
`set_cell_degradation`	Input ports
`set_min_capacitance`	Input ports

Maximum transition time

- To set a maximum transition time of 3.2 for the design adder, enter the following command:

  dc_shell> set_max_transition 3.2 [get_designs adder]

- To undo a set_max_transitioncommand, use remove_attribute. Enter the following command:

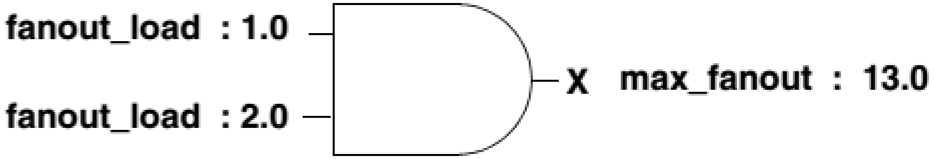
  dc_shell> remove_attribute [get_designs adder] max_transition

- For example the following command sets a max_transitionvalue of 5 on all pins belonging to the Clk clo group:

  dc_shell> set_max_transition 5[get_clocks Clk]

Using DC Tool In Command Mode

Design Compiler models fanout restrictions by associating a fanout_load attribute with each input pin and a max_fanoutattribute with each output (driving) pin on a cell. show these fanout attributes.

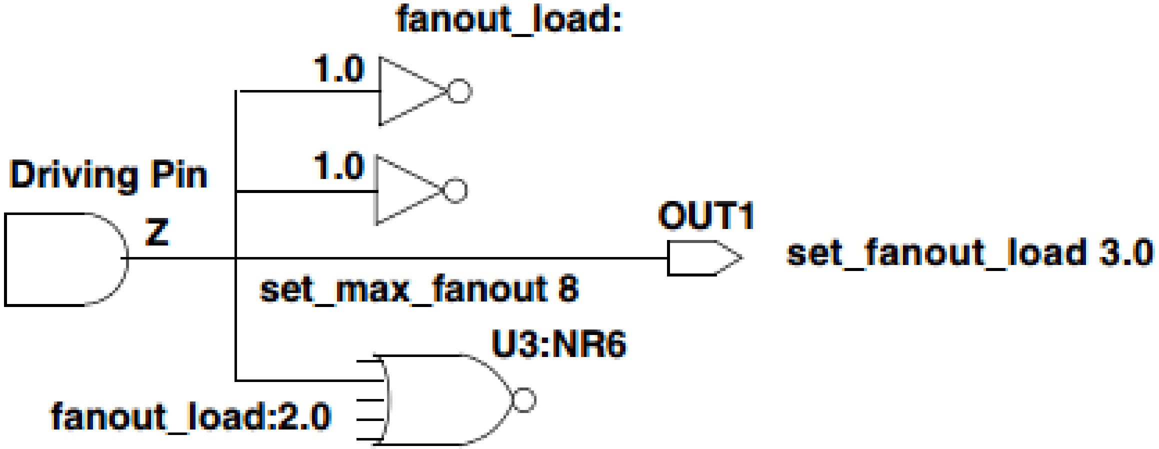


To evaluate the fanout for a driving pin, Design Compiler calculates the sum of all the fanout_load attributes for inputs driven by pin X and compares that number with the number of max_fanout attributes stored at the driving pin X.

• If the sum of the fanout loads is not more than the max_fanout value, the net driven by X is valid.

• If the net driven by X is not valid, Design Compiler tries to make that net valid, perhaps by choosing a higher-drive component.

Using DC Tool In Command Mode



You can set a maximum fanout constraint on every driving pin and input port as follows:

```
dc_shell> set_max_fanout 8 [get_designs ADDER]
```

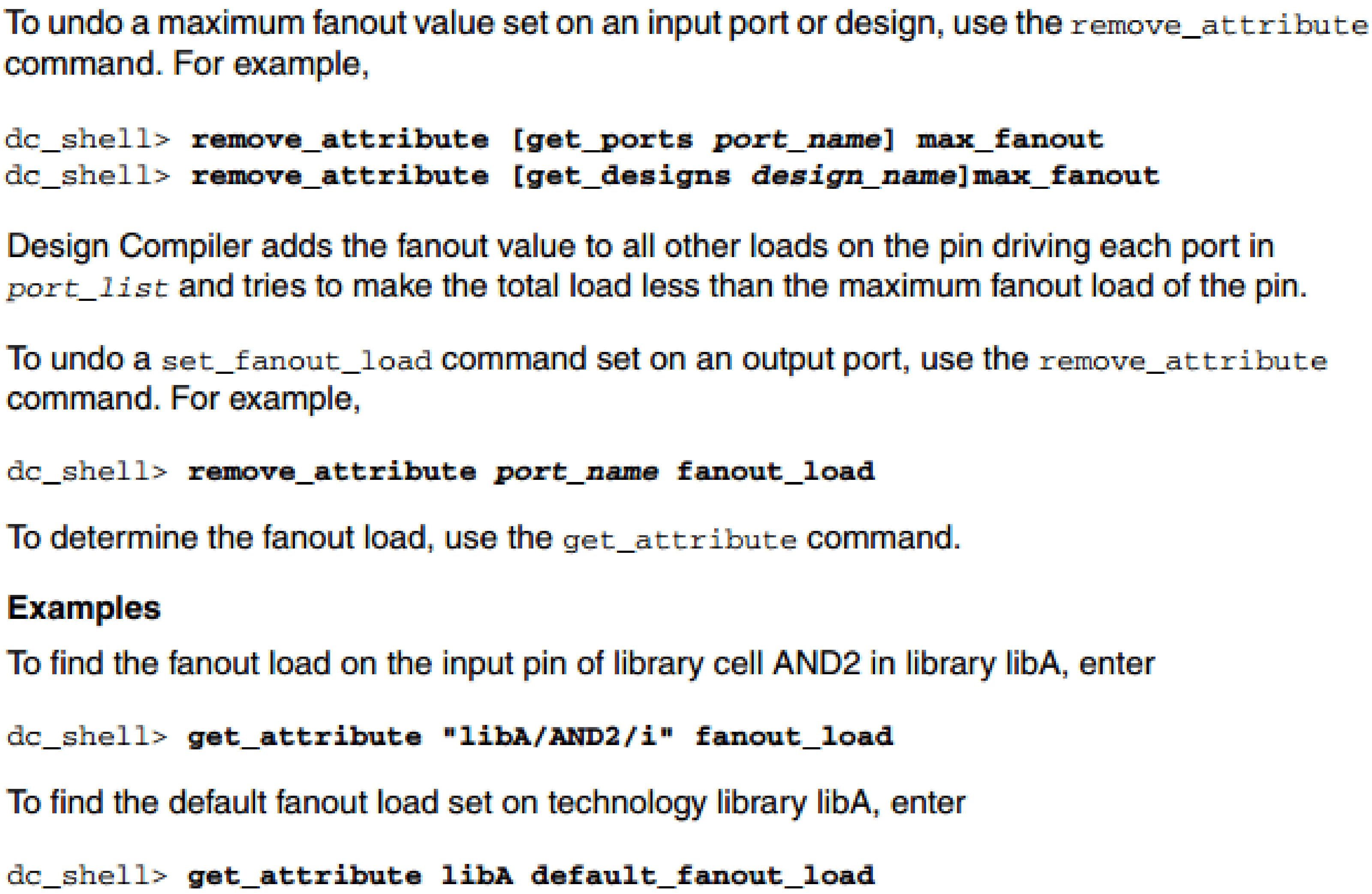
To check whether the maximum fanout constraint is met for driving pin Z, Design Compiler compares the specified `max_fanout` attribute against the fanout load.

In this case, the design meets the constraints.

Total Fanout Load

$$\underbrace{8 \geq 1.0 + 1.0 + 3.0 + 2.0}_{7}$$

Using DC Tool In Command Mode



To undo a maximum fanout value set on an input port or design, use the `remove_attribute` command. For example,

```
dc_shell> remove_attribute [get_ports port_name] max_fanout
dc_shell> remove_attribute [get_designs design_name]max_fanout
```

Design Compiler adds the fanout value to all other loads on the pin driving each port in *port_list* and tries to make the total load less than the maximum fanout load of the pin.

To undo a `set_fanout_load` command set on an output port, use the `remove_attribute` command. For example,

```
dc_shell> remove_attribute port_name fanout_load
```

To determine the fanout load, use the `get_attribute` command.

Examples

To find the fanout load on the input pin of library cell AND2 in library libA, enter

```
dc_shell> get_attribute "libA/AND2/i" fanout_load
```

To find the default fanout load set on technology library libA, enter

```
dc_shell> get_attribute libA default_fanout_load
```

Using DC Tool In Command Mode

- The **set_max_capacitance** command sets a maximum capacitance for the nets attached to named ports or to all the nets in a design. This command allows you to control capacitance directly and places a **max_capacitance** attribute on the listed objects.

- Design Compiler calculates the capacitance on a net by adding the wire capacitance of the net to the capacitance of the pins attached to the net. To determine whether a net meets the capacitance constraint, Design Compiler compares the calculated capacitance value with the max_capacitancevalue of the pin driving the net.

- To set a maximum capacitance of 3 for the design adder, enter one the following command:

```
dc_shell> set_max_capacitance 3 [get_designs adder]
```

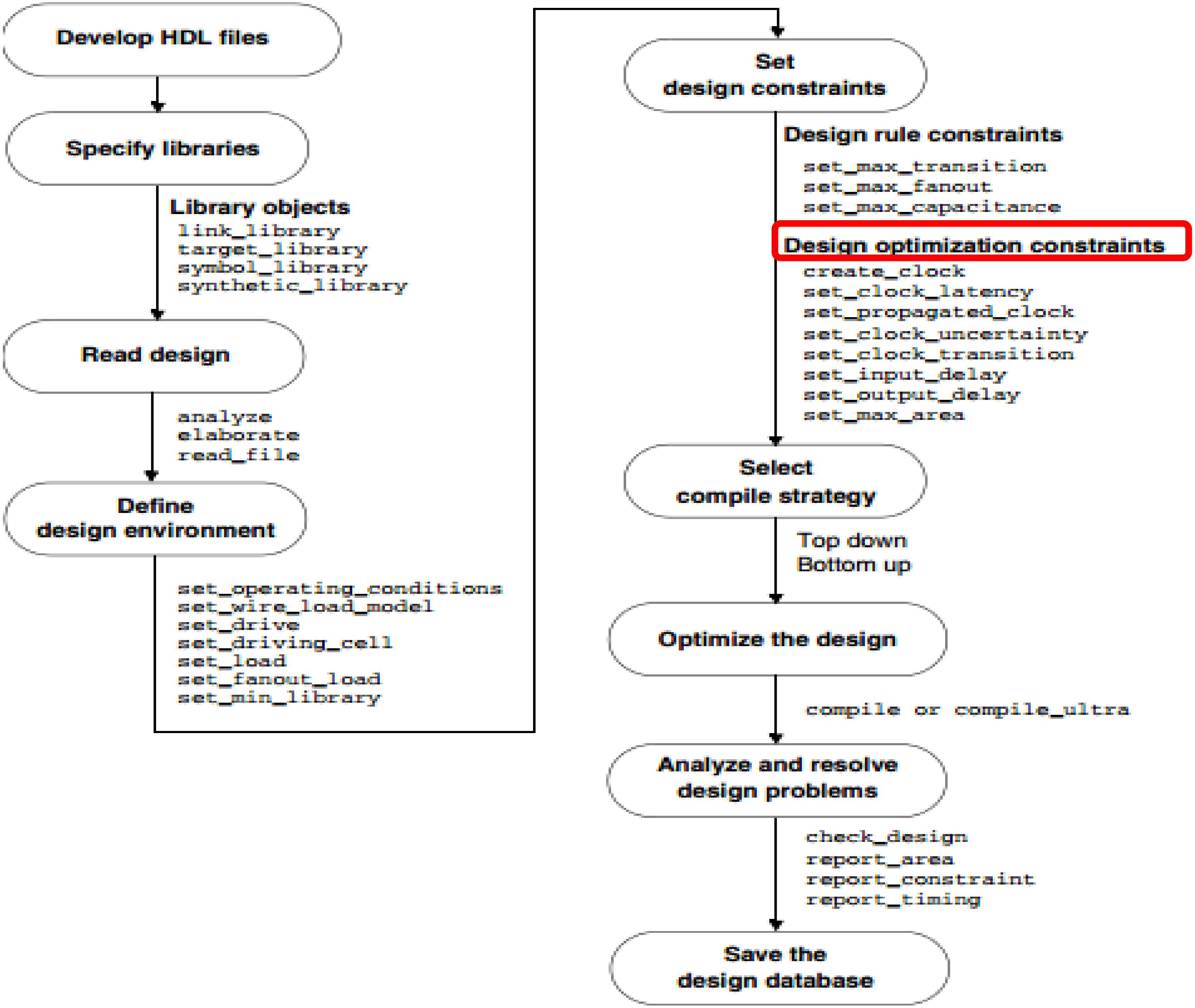
- To undo a set_max_capacitancecommand, use remove_attribute. For example, enter the following command:

```
dc_shell> remove_attribute [get_designs adder] max_capicitance
```

Using DC Tool In Command Mode

- **Cell Degradation**: Cell degradation is a design rule constraint. Some technology libraries contain cell degradation tables. The tables list the maximum capacitance that can be driven by a cell as a function of the transition times at the inputs of the cell.
- The cell_degradation design rule specifies that the capacitance value for a net is less than the cell degradation value.
- This command sets a maximum capacitance value of 2.0 units on the port named late_riser:

```
dc_shell> set_cell_degradation 2.0 late_riser
```



Develop HDL files

Specify libraries

Library objects

```
link_library
target_library
symbol_library
synthetic_library
```

Read design

```
analyze
elaborate
read_file
```

Define design environment

```
set_operating_conditions
set_wire_load_model
set_drive
set_driving_cell
set_load
set_fanout_load
set_min_library
```

Set design constraints

Design rule constraints

```
set_max_transition
set_max_fanout
set_max_capacitance
```

Design optimization constraints

```
create_clock
set_clock_latency
set_propagated_clock
set_clock_uncertainty
set_clock_transition
set_input_delay
set_output_delay
set_max_area
```

Select compile strategy

Top down
Bottom up

Optimize the design

```
compile or compile_ultra
```

Analyze and resolve design problems

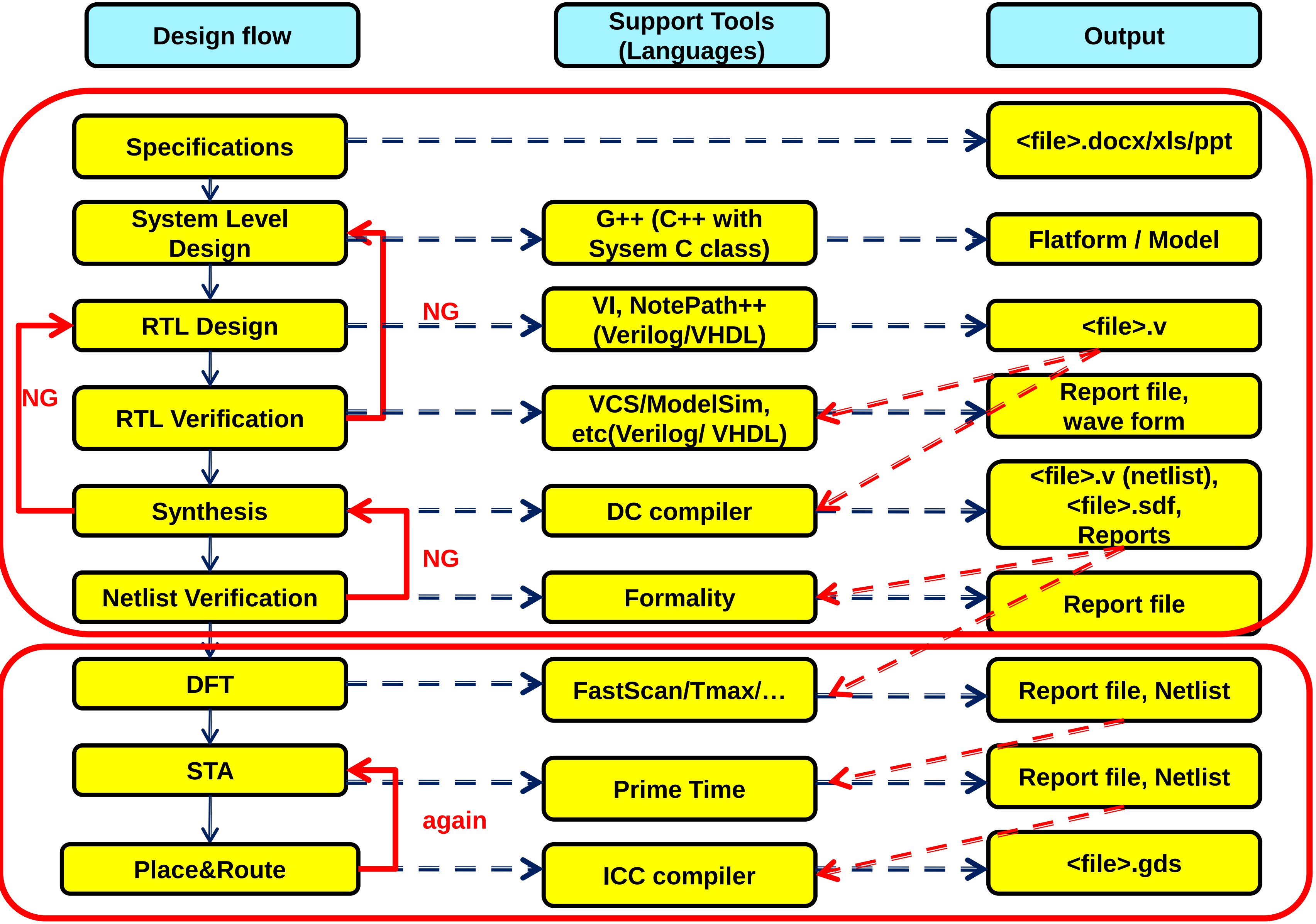
```
check_design
report_area
report_constraint
report_timing
```

Save the design database

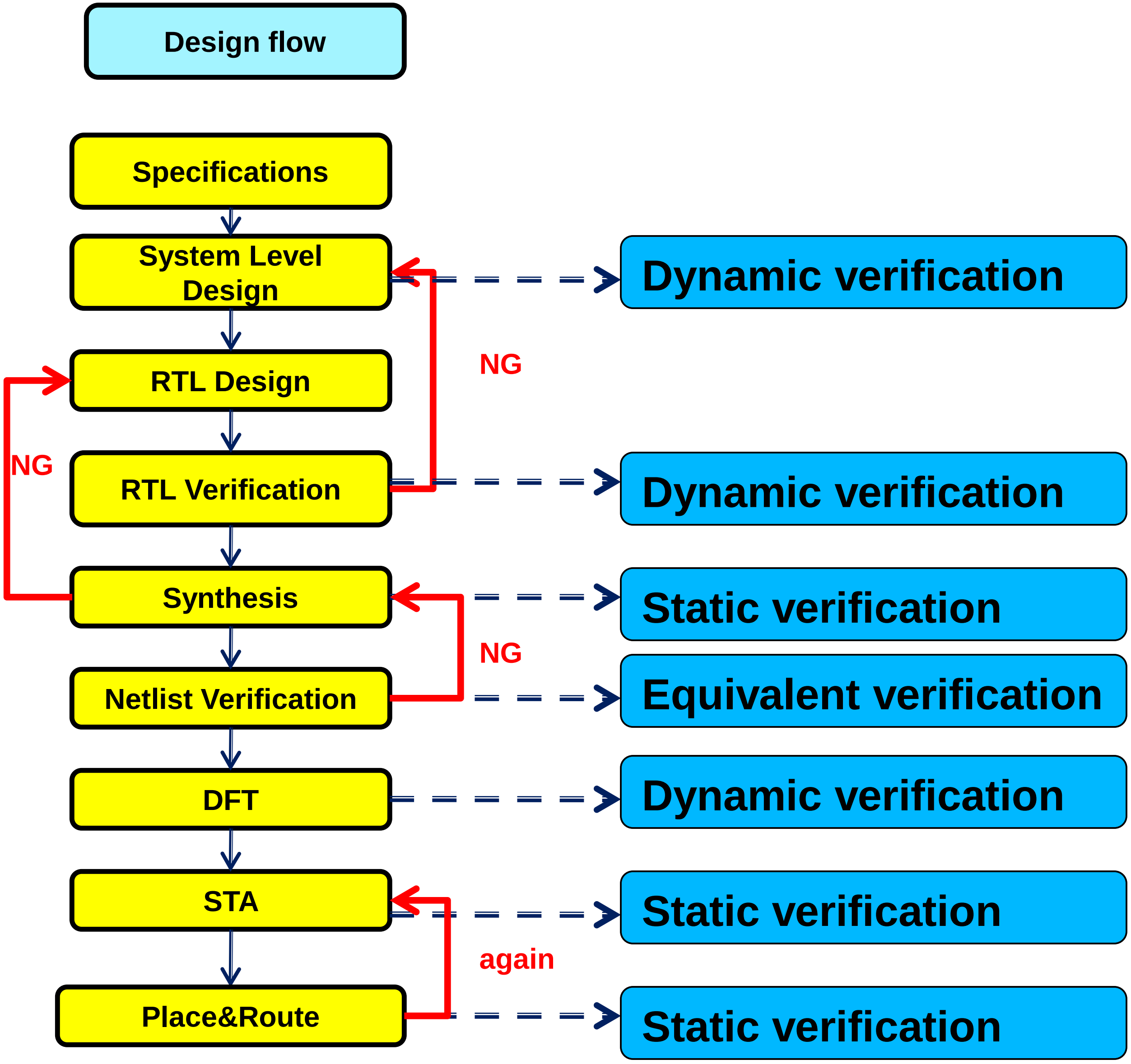
Static Timing Analysis

- ❖STA step
- ❖Concepts
- ❖STA Report Analysis
- ❖Using DC Tool in Command/GUI mode

STA step


Design flow
Support Tools (Languages)
Output
Specifications
<file>.docx/xls/ppt
System Level Design
G++ (C++ with Sysem C class)
Flatform / Model
RTL Design
VI, NotePath++ (Verilog/VHDL)
<file>.v
NG
RTL Verification
VCS/ModelSim, etc(Verilog/ VHDL)
Report file, wave form
NG
Synthesis
DC compiler
<file>.v (netlist), <file>.sdf, Reports
NG
Netlist Verification
Formality
Report file
DFT
FastScan/Tmax/…
Report file, Netlist
STA
Prime Time
Report file, Netlist
again
Place&Route
ICC compiler
<file>.gds

STA step



Design flow

Specifications

System Level Design

RTL Design

RTL Verification

Synthesis

Netlist Verification

DFT

STA

Place&Route

Dynamic verification

Dynamic verification

Static verification

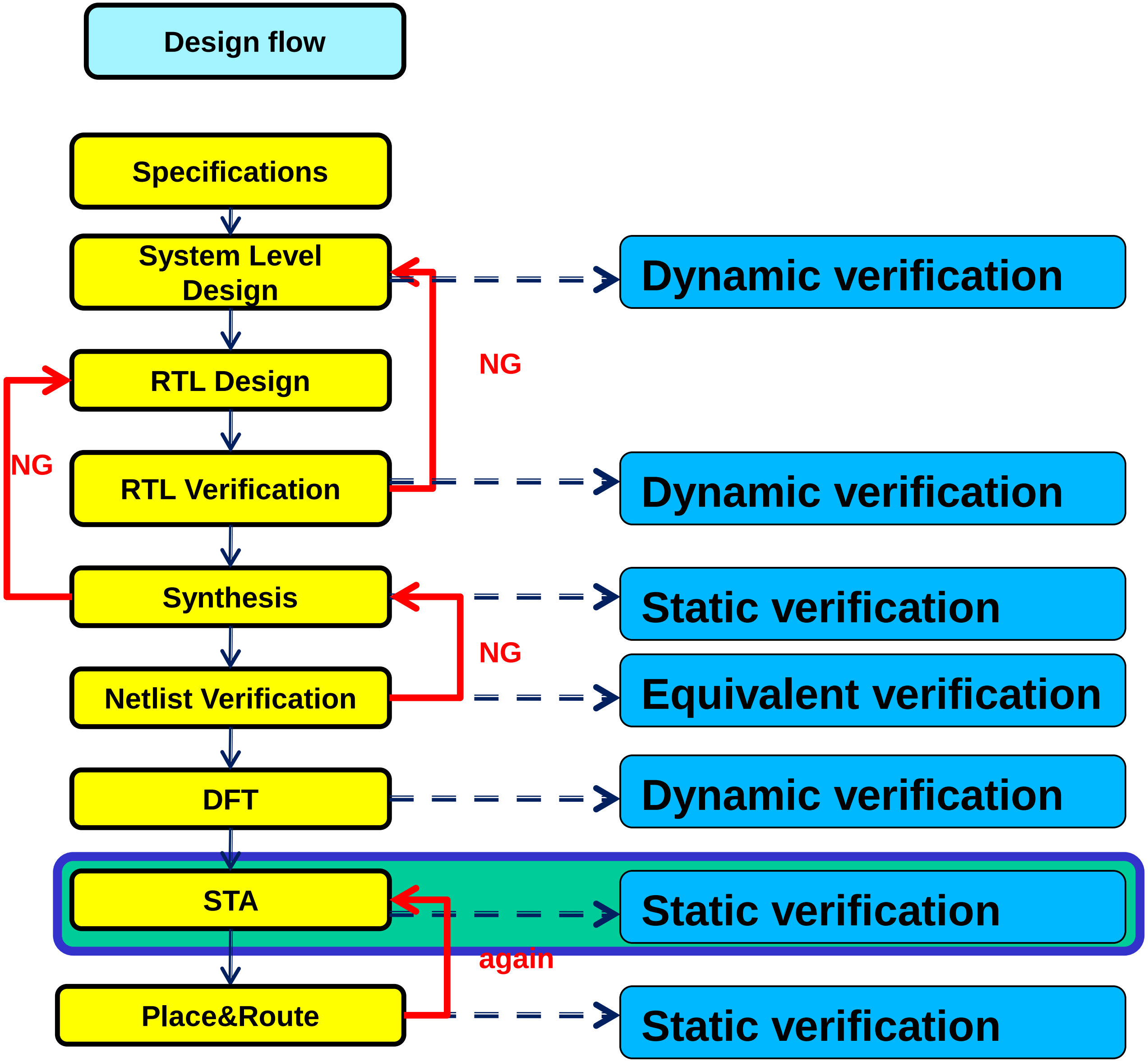
Equivalent verification

Dynamic verification

Static verification

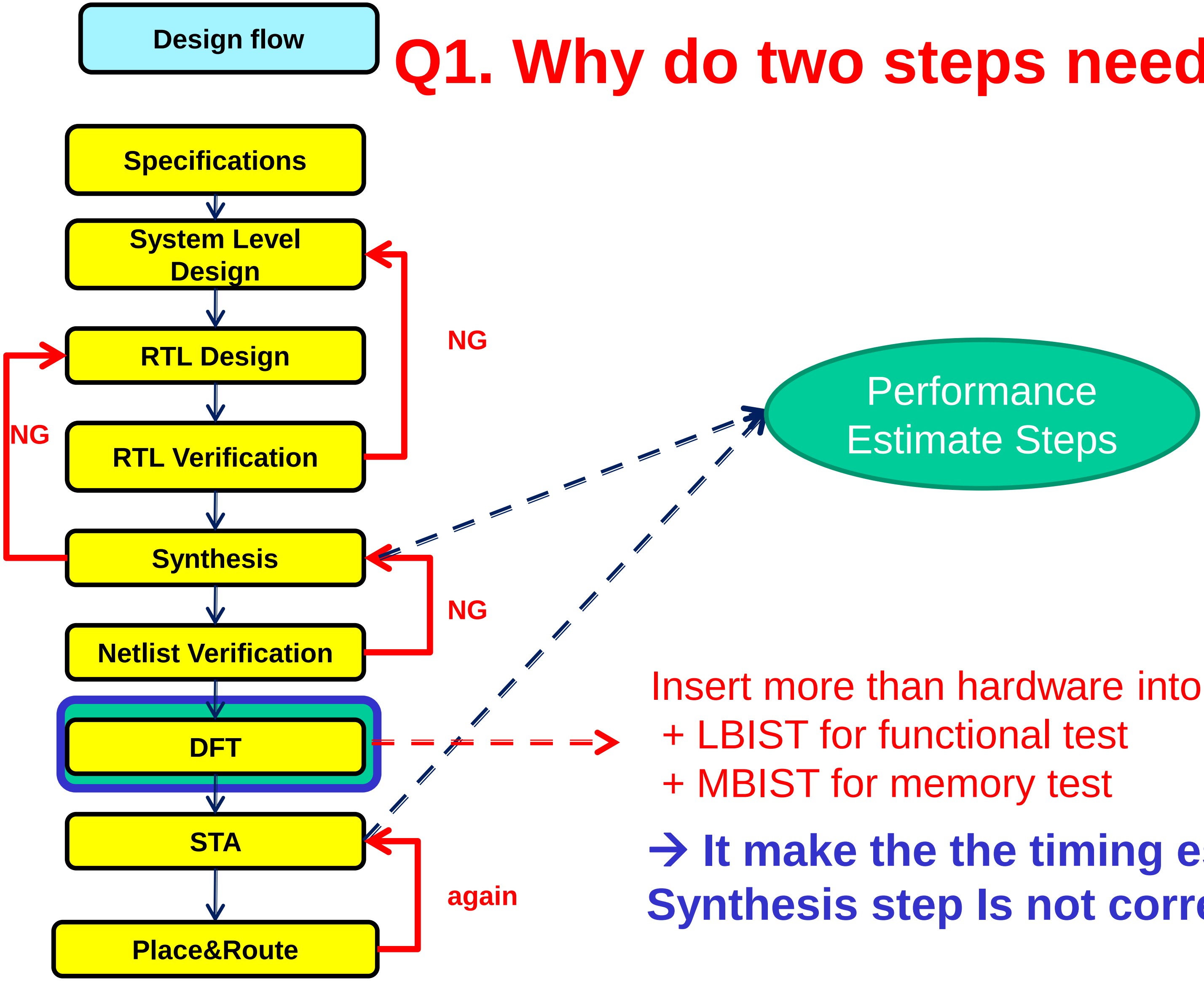
Static verification

STA step


Design flow
Specifications
System Level Design
Dynamic verification
NG
RTL Design
NG
RTL Verification
Dynamic verification
Synthesis
Static verification
NG
Netlist Verification
Equivalent verification
DFT
Dynamic verification
STA
Static verification
again
Place&Route
Static verification

STA step

Q1. Why do two steps need to estimate?



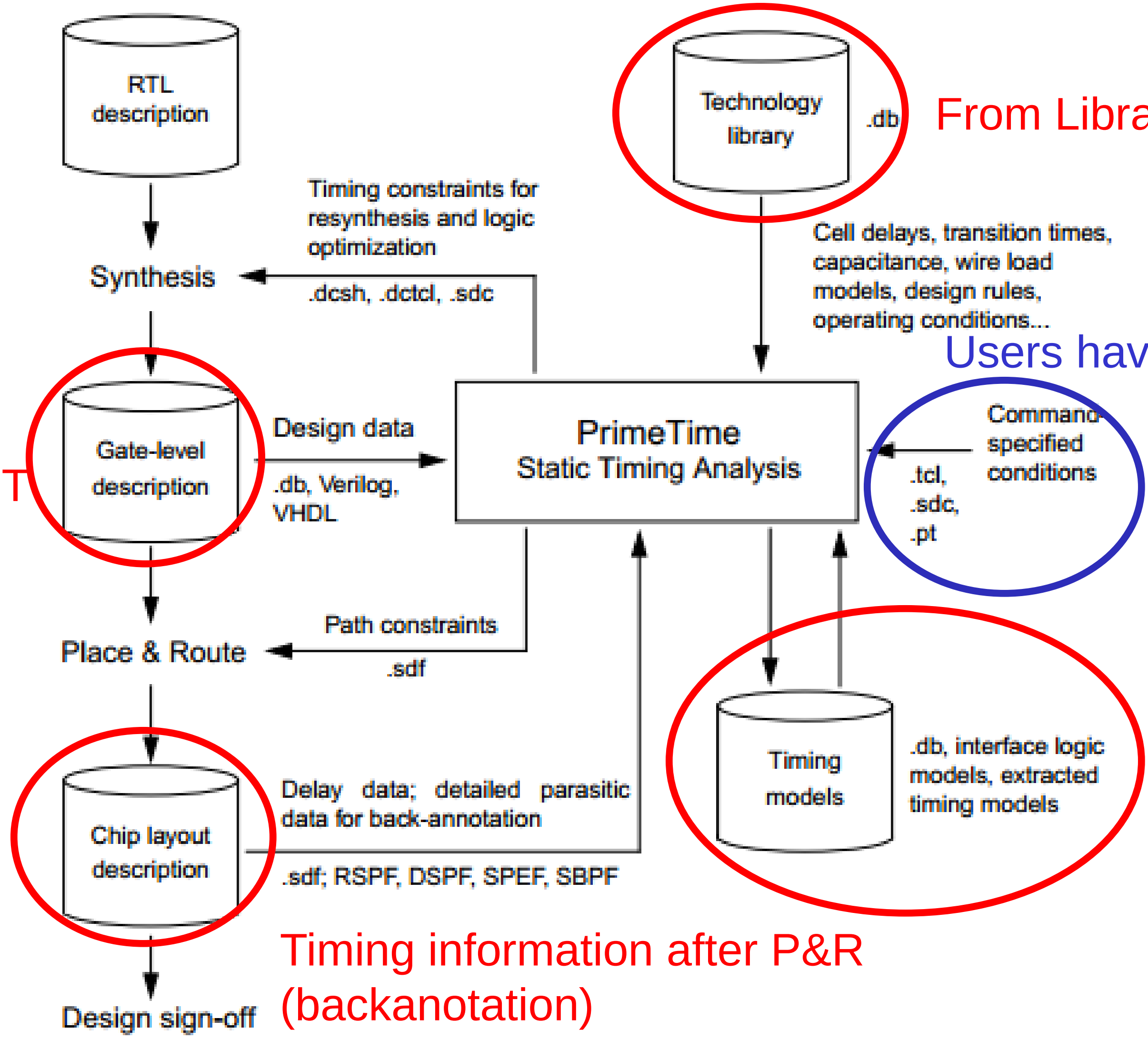
Insert more than hardware into the netlist for self test

+ LBIST for functional test

+ MBIST for memory test

→ It make the the timing estimation at Synthesis step Is not correct

STA step



RTL description

Synthesis

Timing constraints for resynthesis and logic optimization

.dcsh, .dctcl, .sdc

Technology library

.db

Cell delays, transition times, capacitance, wire load models, design rules, operating conditions...

Gate-level description

Design data

.db, Verilog, VHDL

PrimeTime
Static Timing Analysis

Command specified conditions

.tcl, .sdc, .pt

Place & Route

Path constraints

.sdf

Timing models

.db, interface logic models, extracted timing models

Chip layout description

Delay data; detailed parasitic data for back-annotation

.sdf; RSPF, DSPF, SPEF, SBPF

Design sign-off

From Library

Users have to define

Platen RTL code after inserting DFT

From Library

Timing information after P&R (backanotation)

Concepts

❖**Type of checking**

+ Timing contrains : Setup time, Hold time, removal and recovery
+ data-to-data timing constrains
+ Clock gating setup & hold constrains
+ Minimum period/pulse width of clock
+ Design rule (minimum/maximum transition time, capacitance and fanout …)

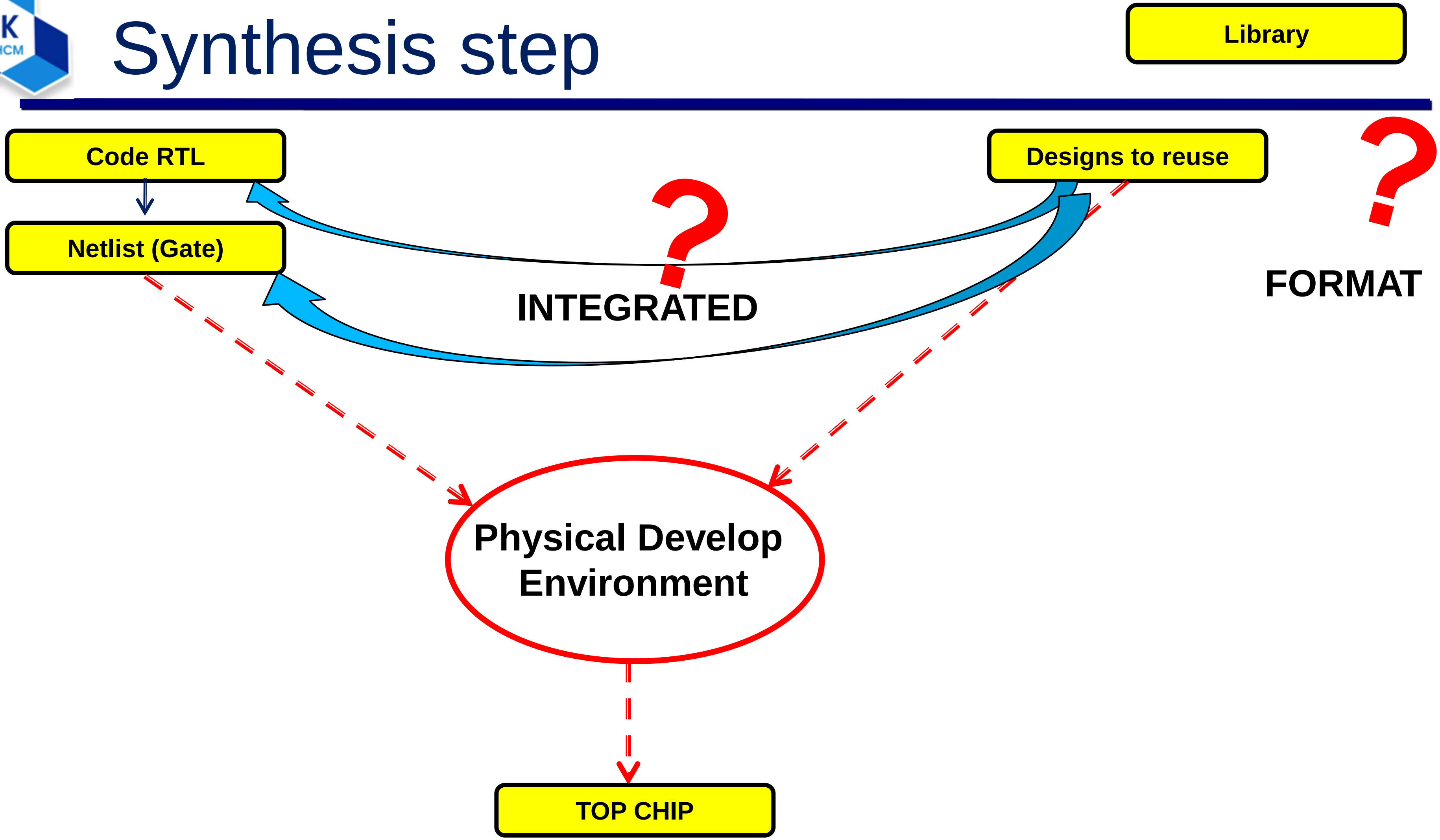
❖**Analysis features**

+ Multiple clock
+ Multicycle part timing
+ False path timing
+ Min/max delay (setup/hold time)
+ On chip variation of process/voltage/Temperature
+ Mode analysis
+ Bottleneck analysis
+ ECO analysis
+ Crosstalk effect

❖**Backanotation extracted file**

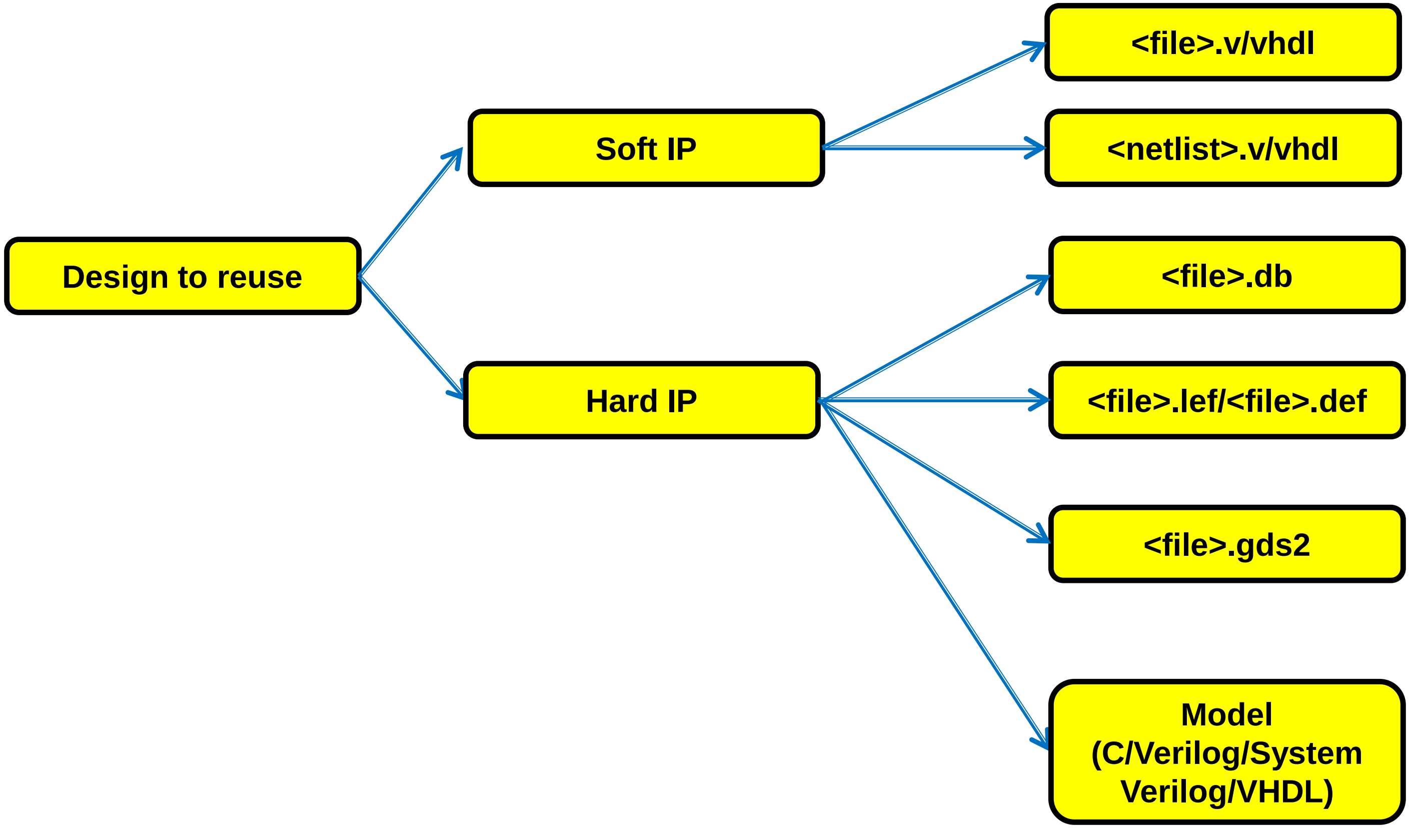
+ <file>.SDF : Standard Parasitic Format
+ <file>.RSPF: Reduced Standard Parasitic Format
+ <file>.DSPF: Detailed Standard Parasitic Format
+ <file>.SPEF: Standard Parasitic Exchange Format
+ <file>.SBPF: Synopsis Binary Parasitic Format

Synthesis step



Synthesis step

Library


Design to reuse

Soft IP

Hard IP

<file>.v/vhdl

<netlist>.v/vhdl

<file>.db

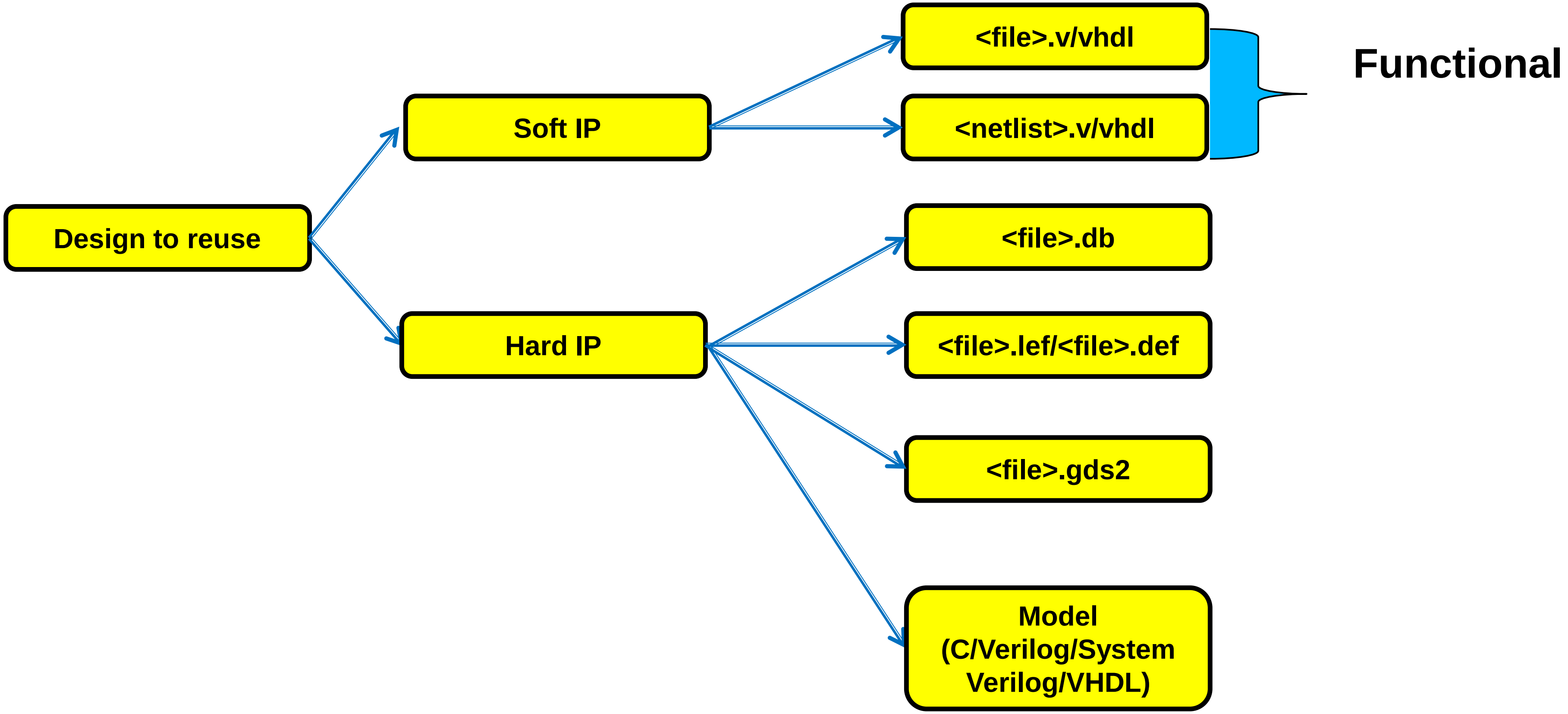
<file>.lef/<file>.def

<file>.gds2

Model (C/Verilog/System Verilog/VHDL)

Synthesis step

Library



Design to reuse

Soft IP

Hard IP

<file>.v/vhdl

<netlist>.v/vhdl

Functional

<file>.db

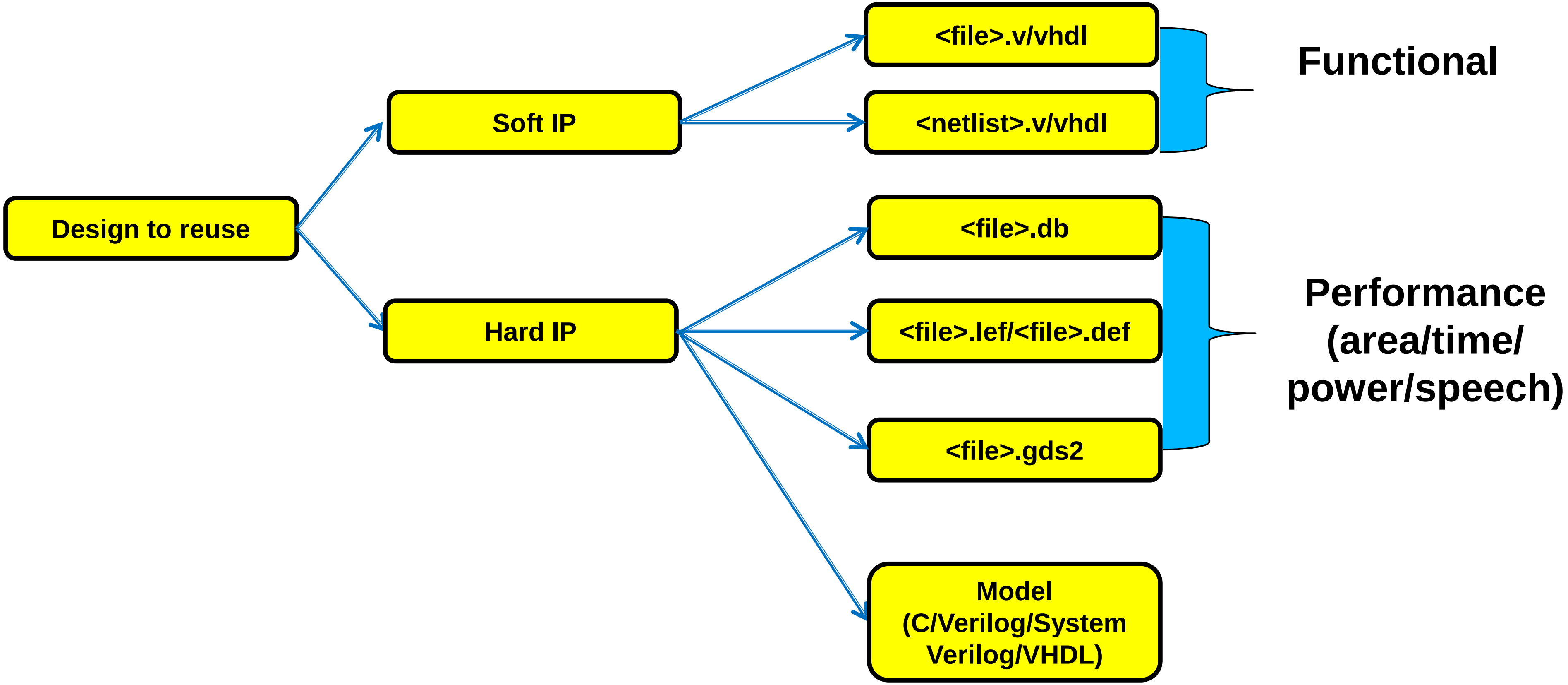
<file>.lef/<file>.def

<file>.gds2

Model (C/Verilog/System Verilog/VHDL)

Synthesis step

Library



Design to reuse

Soft IP

Hard IP

<file>.v/vhdl

<netlist>.v/vhdl

Functional

<file>.db

<file>.lef/<file>.def

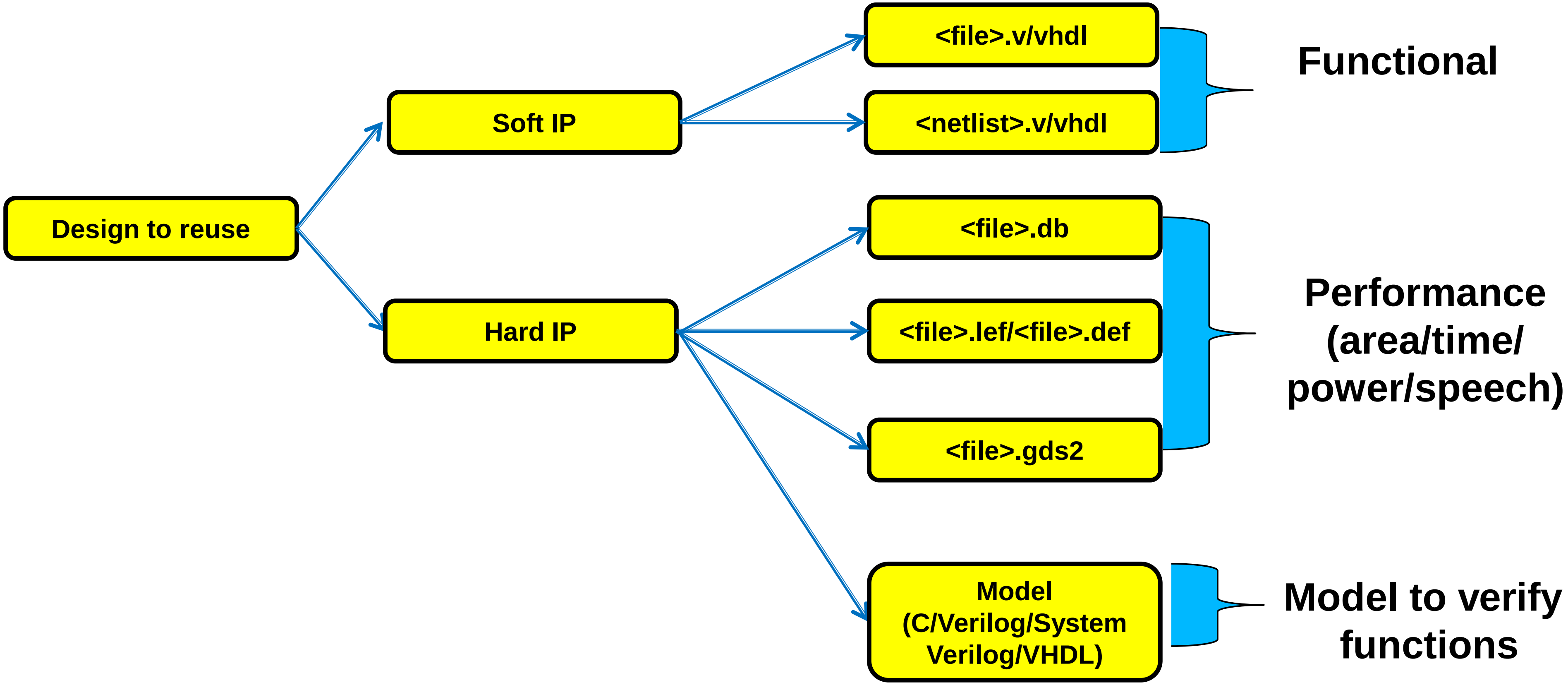
<file>.gds2

Performance (area/time/ power/speech)

Model (C/Verilog/System Verilog/VHDL)

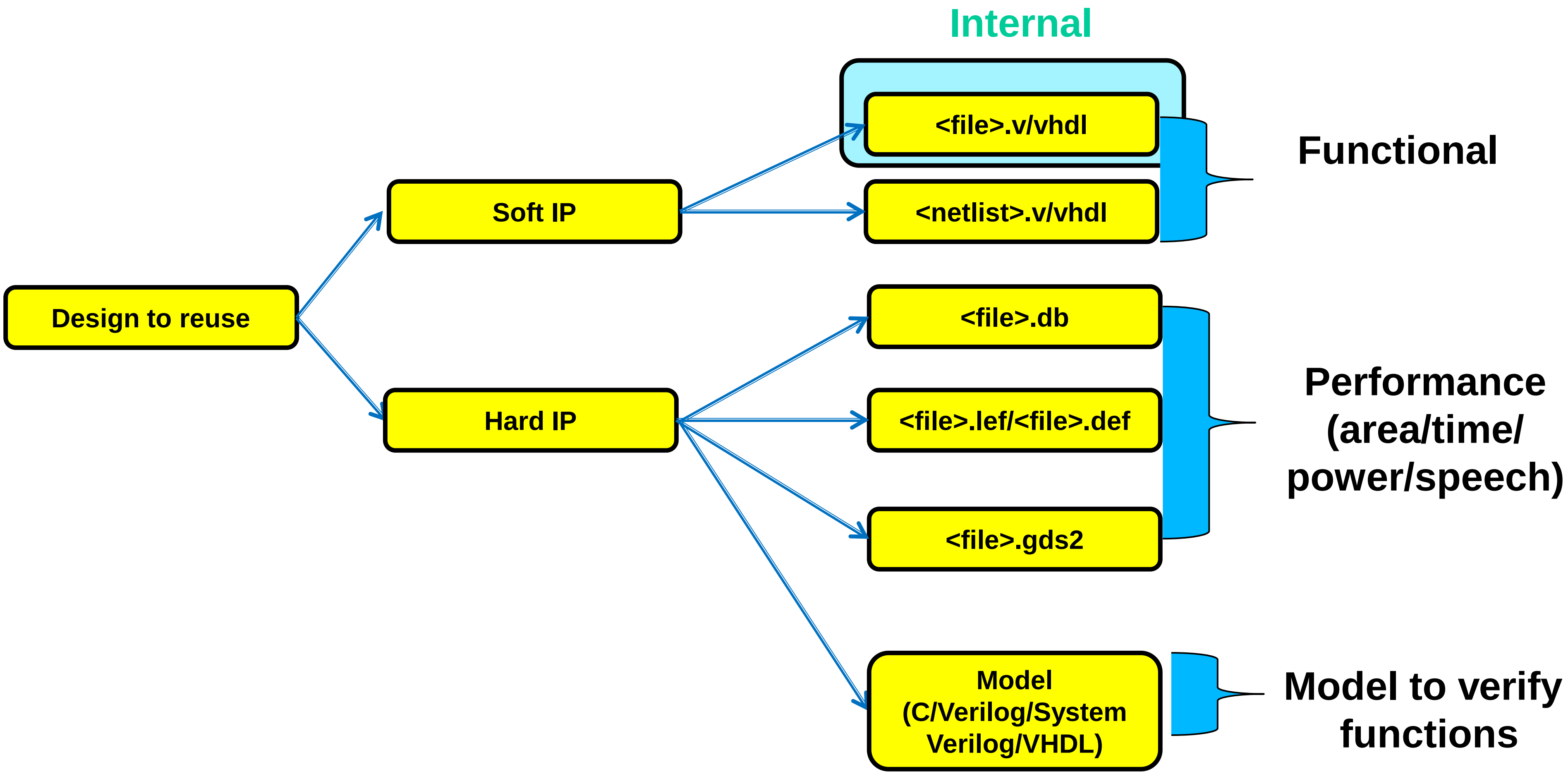
Synthesis step

Library



Synthesis step

Library



Internal

Design to reuse

Soft IP

Hard IP

<file>.v/vhdl

<netlist>.v/vhdl

Functional

<file>.db

<file>.lef/<file>.def

<file>.gds2

Performance (area/time/ power/speech)

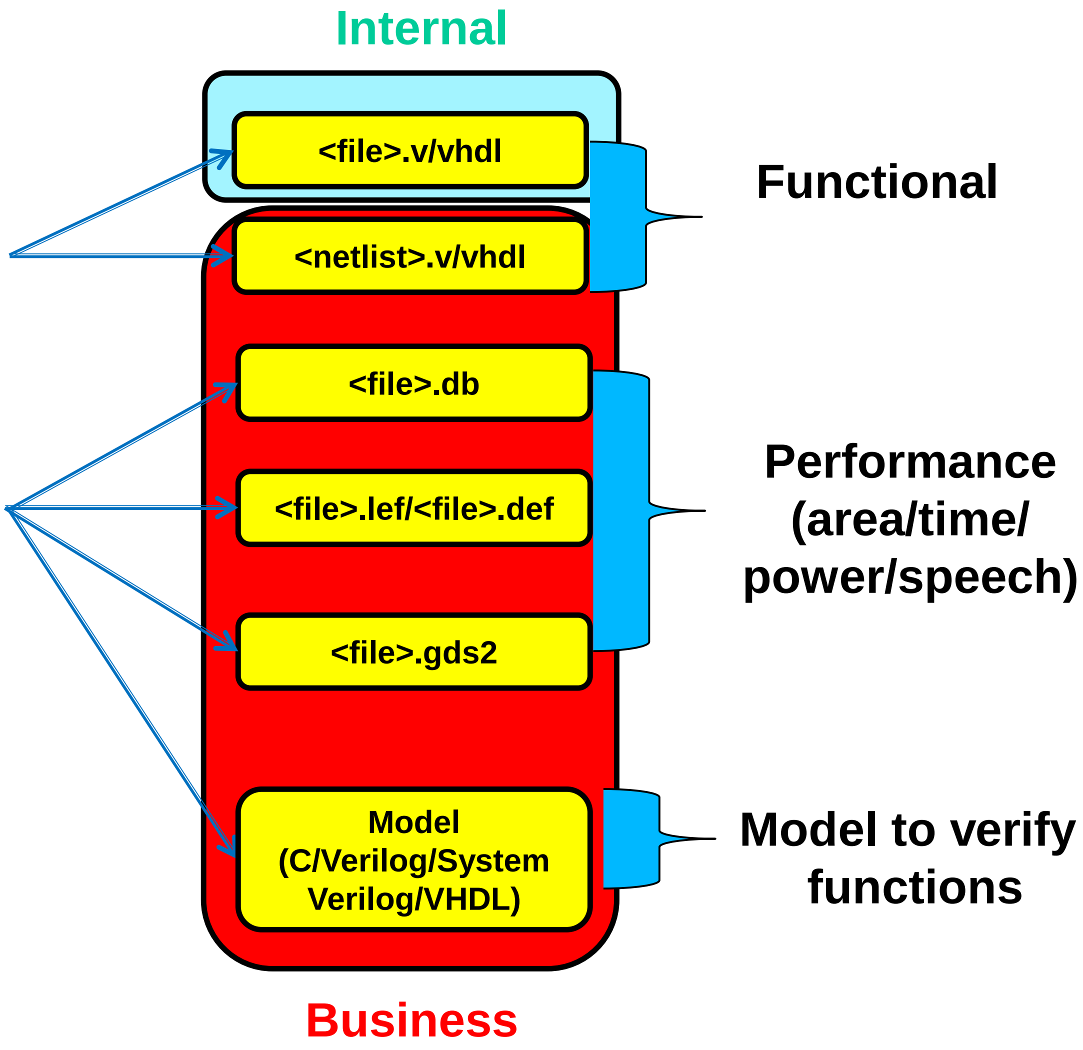
Model (C/Verilog/System Verilog/VHDL)

Model to verify functions

Synthesis step

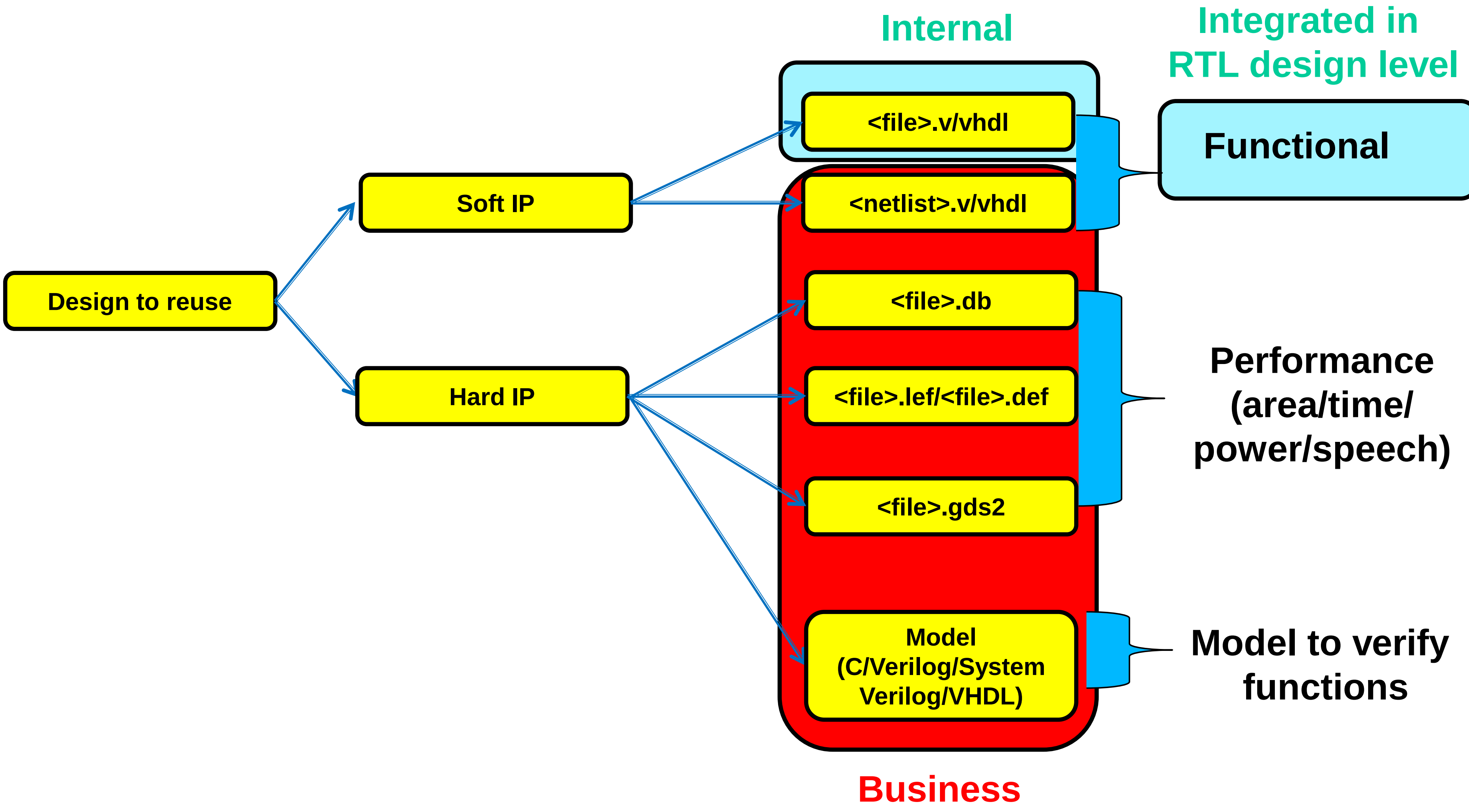
Library

Design to reuse


Internal
<file>.v/vhdl
<netlist>.v/vhdl
Functional
<file>.db
<file>.lef/<file>.def
<file>.gds2
Performance (area/time/ power/speech)
Model (C/Verilog/System Verilog/VHDL)
Model to verify functions
Business

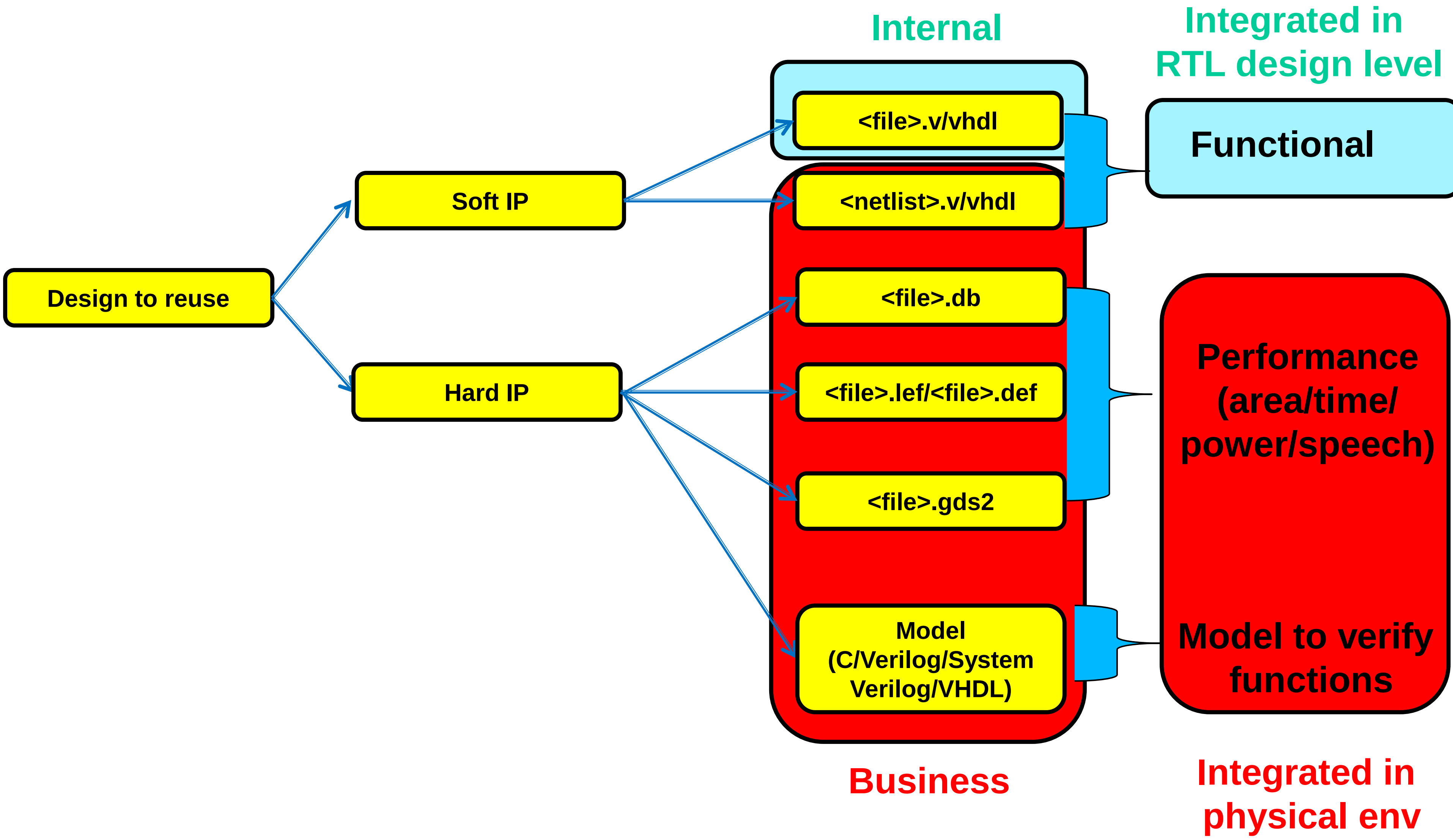
Synthesis step

Library


Internal
Integrated in RTL design level
<file>.v/vhdl
Functional
Soft IP
<netlist>.v/vhdl
Design to reuse
<file>.db
Hard IP
<file>.lef/<file>.def
Performance (area/time/ power/speech)
<file>.gds2
Model (C/Verilog/System Verilog/VHDL)
Model to verify functions
Business

Synthesis step

Library



Internal

Integrated in RTL design level

Design to reuse

Soft IP

Hard IP

<file>.v/vhdl

<netlist>.v/vhdl

<file>.db

<file>.lef/<file>.def

<file>.gds2

Model (C/Verilog/System Verilog/VHDL)

Functional

Performance (area/time/ power/speech)

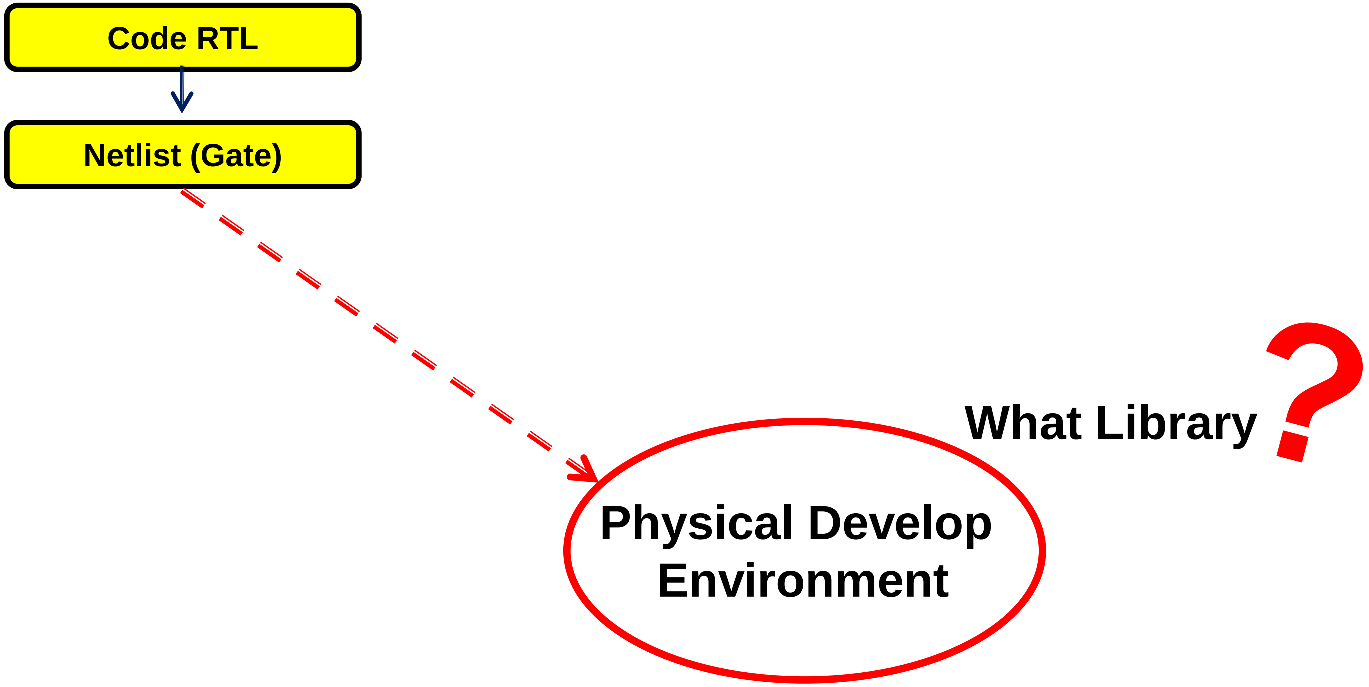
Model to verify functions

Business

Integrated in physical env

Synthesis step

Library



Code RTL

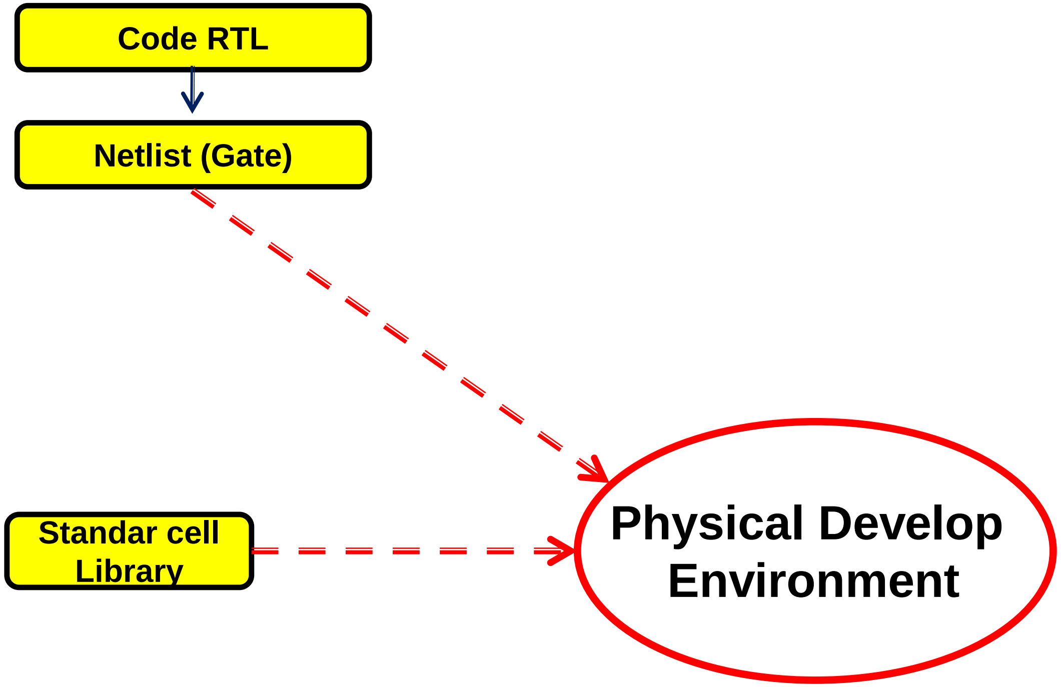
Netlist (Gate)

What Library ?

Physical Develop Environment

Synthesis step

Library



Code RTL

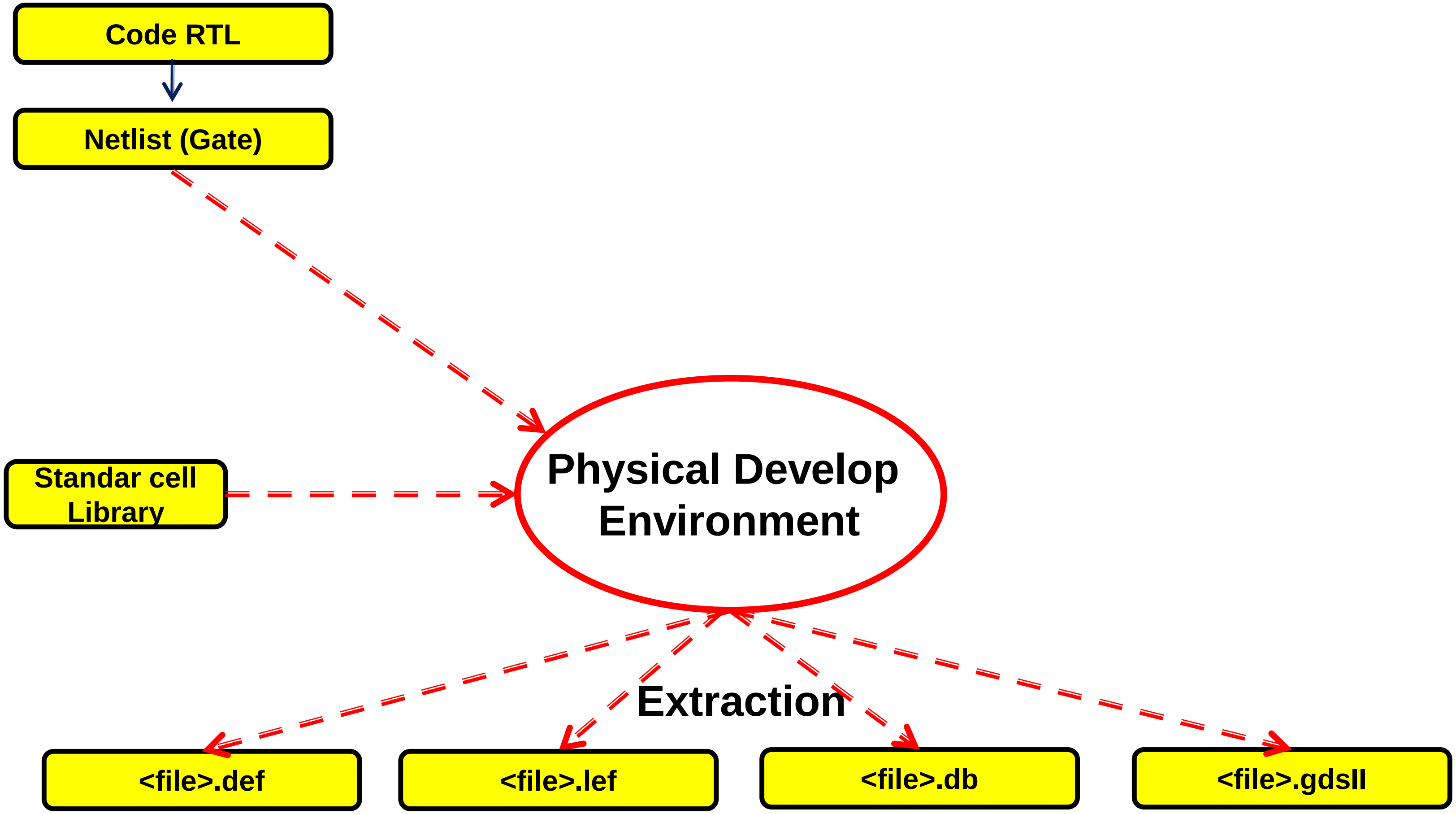
Netlist (Gate)

Standar cell Library

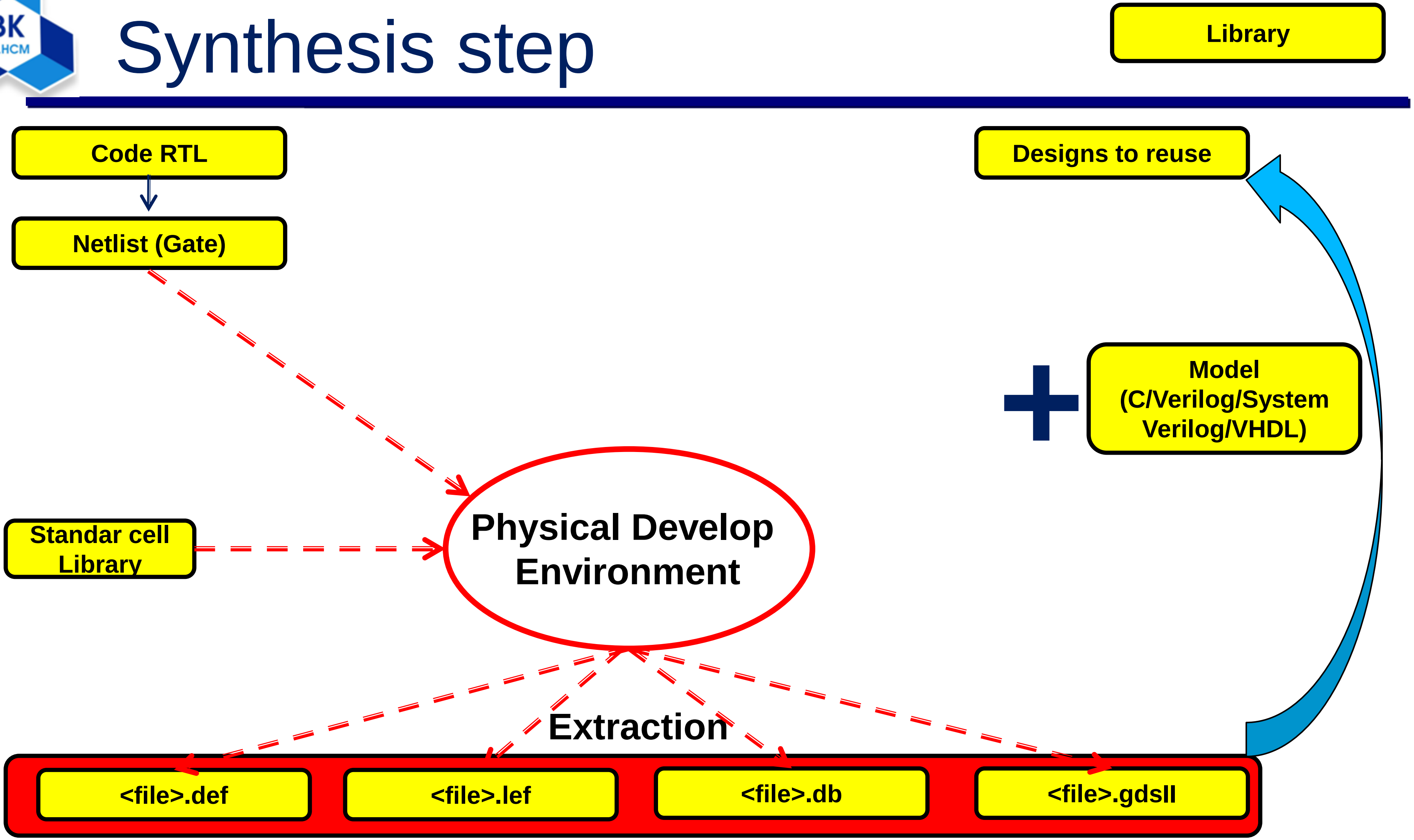
Physical Develop Environment

Synthesis step

Library



Synthesis step



Synthesis step

Library

- **LEF: Library Exchange Format:** LEF is used to define an ICprocess and a logic cell library. For example, you would use LEF to describe a gate array: the base cells, the logic macros with their size and connectivity information, the interconnect layers and other information to set up the database that the physical design tools need.

- **DEF: Design Exchange format:** DEF is used to describe all the physical aspects of a particular chip design including the netlist and physical location of cells on the chip. For example, if you had a complete placement from a floorplanning tool and wanted to exchange this information with another tool, you would use DEF.

- **GDSII: Generic Data Structures Library**: This file is used by foundry for fabricating the ASIC.

- **SPF: Standard Parasitic Format:** This file is generated by the Parasitic Extraction Tool. It contains all the parasitic information of the design such as capacitance, resistance etc. This file is generated after the layout. It is back annotated and the timing analysis of the design is performed again to get the exact timing information.

- **<file>.lib: Liberty Format** This library format is from Synopsys (Transfer to <file>.db berore repairing the library to physical environment)

Q & A