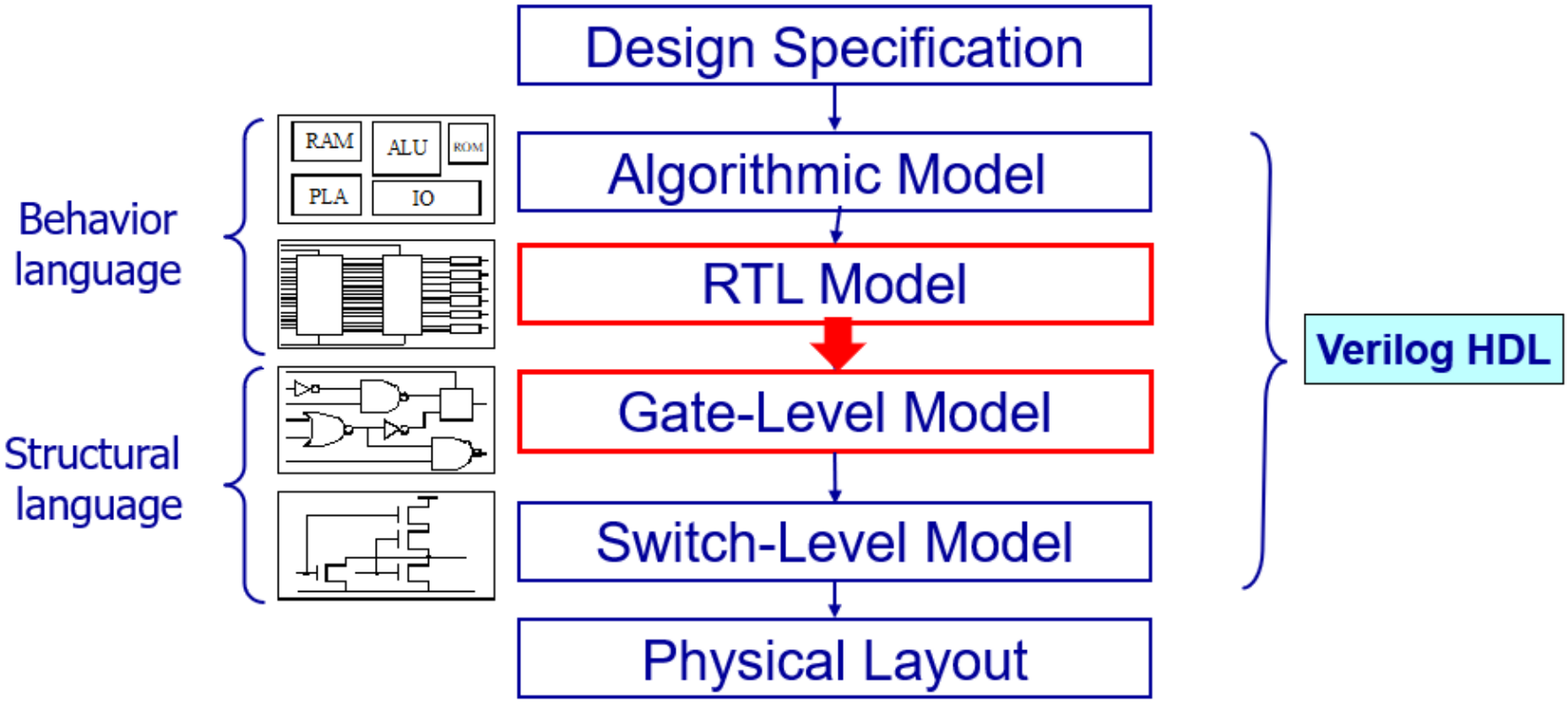
Introduction to Synthesis Flow with Synopsys Design Compiler

Lecturer: Shao-Hua Lian

Review : IC design flow



Outline

- **Section 1 Design Compiler Introduction**
- **Section 2 Basic Synthesis Flow**
  - Develop HDL files
  - Specify libraries
  - Read design
  - Develop design environment
  - Set design constraints
  - Select compile strategy
  - Optimize the design
  - Analyze and resolve design problems
  - Save the design database
- **Section 3 Generate & For Loop**

Outline

- **Section 1 Design Compiler Introduction**
- **Section 2 Basic Synthesis Flow**
  - Develop HDL files
  - Specify libraries
  - Read design
  - Develop design environment
  - Set design constraints
  - Select compile strategy
  - Optimize the design
  - Analyze and resolve design problems
  - Save the design database
- **Section 3 Generate & For Loop**

Design Compiler Introduction

✔ Design compiler

- It synthesizes your HDL designs (Verilog) into optimized technology-dependent(D35,U18…), gate-level designs.
- It can optimize both combinational and sequential designs for speed, area, and power.

RTL(.v) → **Netlist(.v)**

```
module adder_1b(
        input clk,
        input rst_n,
        input x1,x2,
        output reg [1:0] sum
);

always@(posedge clk or negedge rst_n)begin
    if(!rst_n)
        sum <= 0;
    else
        sum <= x1 + x2;
end

endmodule
```

```
////////////////////////////////////////////////////////////////////
// Created by: Synopsys DC Ultra(TM) in wire load mode
// Version   : K-2015.06-SP1
// Date      : Sun Oct 28 18:55:52 2018
////////////////////////////////////////////////////////////////////


module adder_1b ( clk, rst_n, x1, x2, sum );
  output [1:0] sum;
  input clk, rst_n, x1, x2;
  wire   n1, n2, n3, n4;

  DFFSX2 sum_reg_1_ ( .D(n4), .CK(clk), .SN(rst_n), .QN(sum[1]) );
  DFFSX2 sum_reg_0_ ( .D(n3), .CK(clk), .SN(rst_n), .QN(sum[0]) );
  AND2XL U5 ( .A(x1), .B(x2), .Y(n2) );
  AOI2BB1XL U6 ( .A0N(x1), .A1N(x2), .B0(n2), .Y(n1) );
  INVXL U7 ( .A(n1), .Y(n3) );
  INVXL U8 ( .A(n2), .Y(n4) );
endmodule
```

No always block or assign

Logic Synthesis

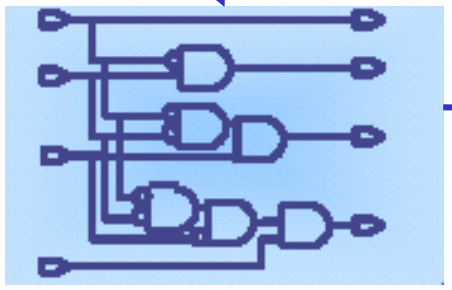
Logic synthesis

- A process by which behavioral model of a circuit is turned into an implementation in terms of logic gates
- Synthesis = **Translation + Mapping + Optimization**

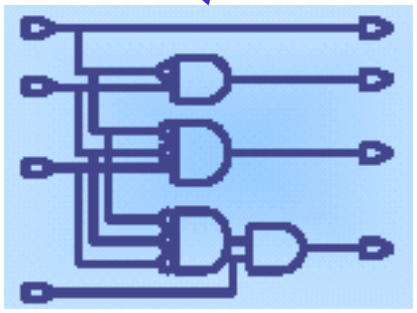
```
assign avg=sum/total;
always_ff @(posedge clk)
begin
    sum=sum+score*weight;
end
```

HDL Source

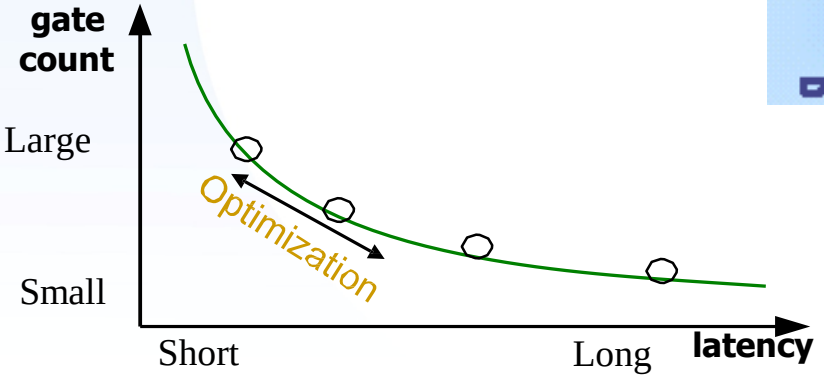
Translate



Map + Optimize



Target Technology



GTECH

- GTECH Netlist

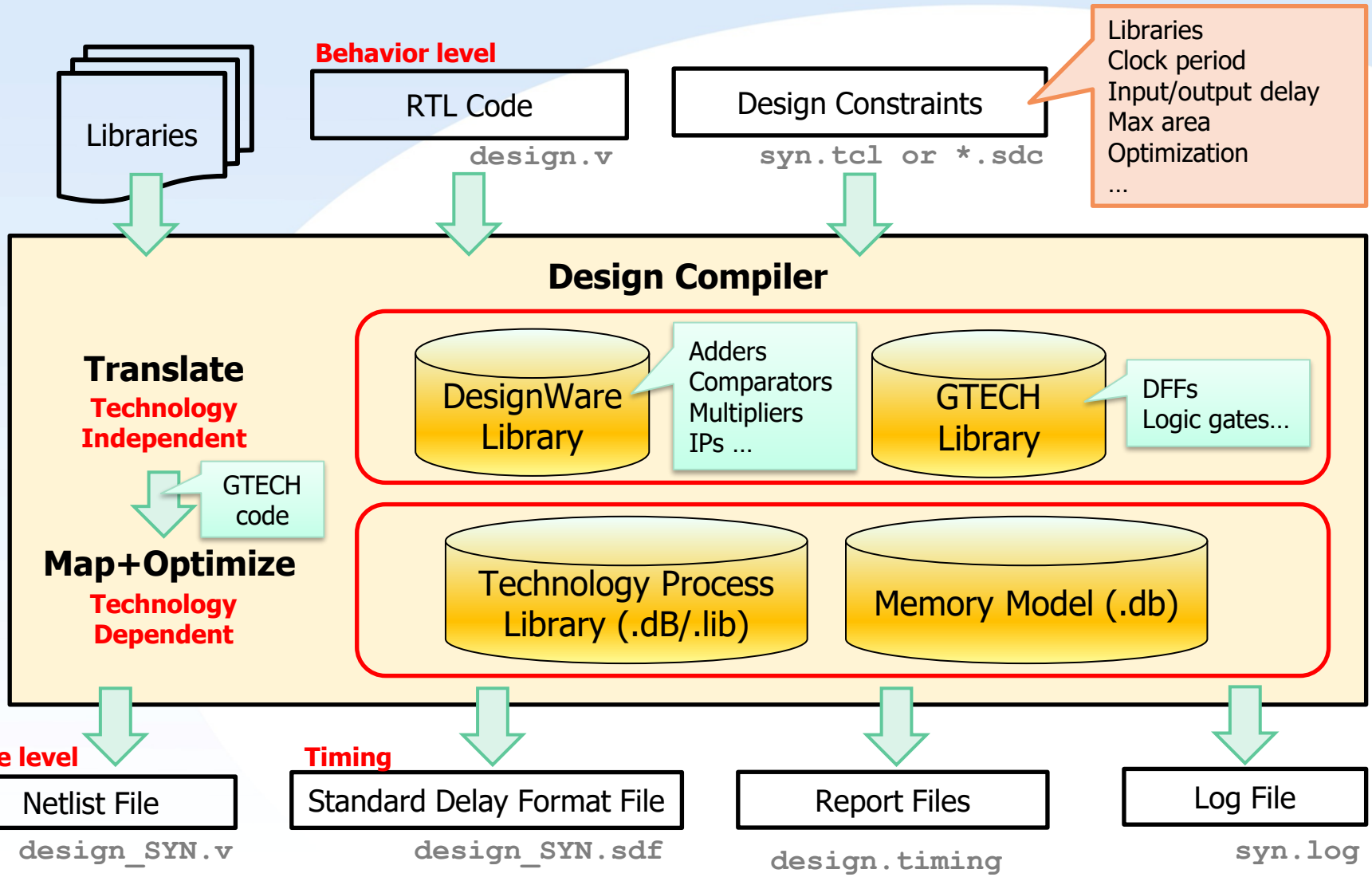
```
module mux ( out, sel, b, a );
  input sel, b, a;
  output out;
  wire   sel_, selb, sela;

  GTECH_NOT I_0 ( .A(sel), .Z(sel_) );
  GTECH_AND2 __tmp101 ( .A(sel), .B(b), .Z(selb) );
  GTECH_AND2 __tmp102 ( .A(sel_), .B(a), .Z(sela) );
  GTECH_OR2 __tmp103 ( .A(selb), .B(sela), .Z(out) );
endmodule

module dffr ( q, q_, d, clk, rst_ );
  input d, clk, rst_;
  output q, q_;
  wire   N0, N1, N2, N3, N4, N5, dl, qe, de, dl_;

  GTECH_AND2 n1 ( .A(dl), .B(qe), .Z(N0) );
  GTECH_NOT I_0 ( .A(N0), .Z(de) );
  GTECH_AND3 n2 ( .A(clk), .B(de), .C(rst_), .Z(N1) );
  GTECH_NOT I_1 ( .A(N1), .Z(qe) );
  GTECH_AND3 n3 ( .A(d), .B(dl_), .C(rst_), .Z(N2) );
  GTECH_NOT I_2 ( .A(N2), .Z(dl) );
  GTECH_AND3 n4 ( .A(dl), .B(clk), .C(qe), .Z(N3) );
  GTECH_NOT I_3 ( .A(N3), .Z(dl_) );
  GTECH_AND2 n5 ( .A(qe), .B(q_), .Z(N4) );
  GTECH_NOT I_4 ( .A(N4), .Z(q) );
  GTECH_AND3 n6 ( .A(dl_), .B(q), .C(rst_), .Z(N5) );
  GTECH_NOT I_5 ( .A(N5), .Z(q_) );
endmodule
```

- Normal Netlist

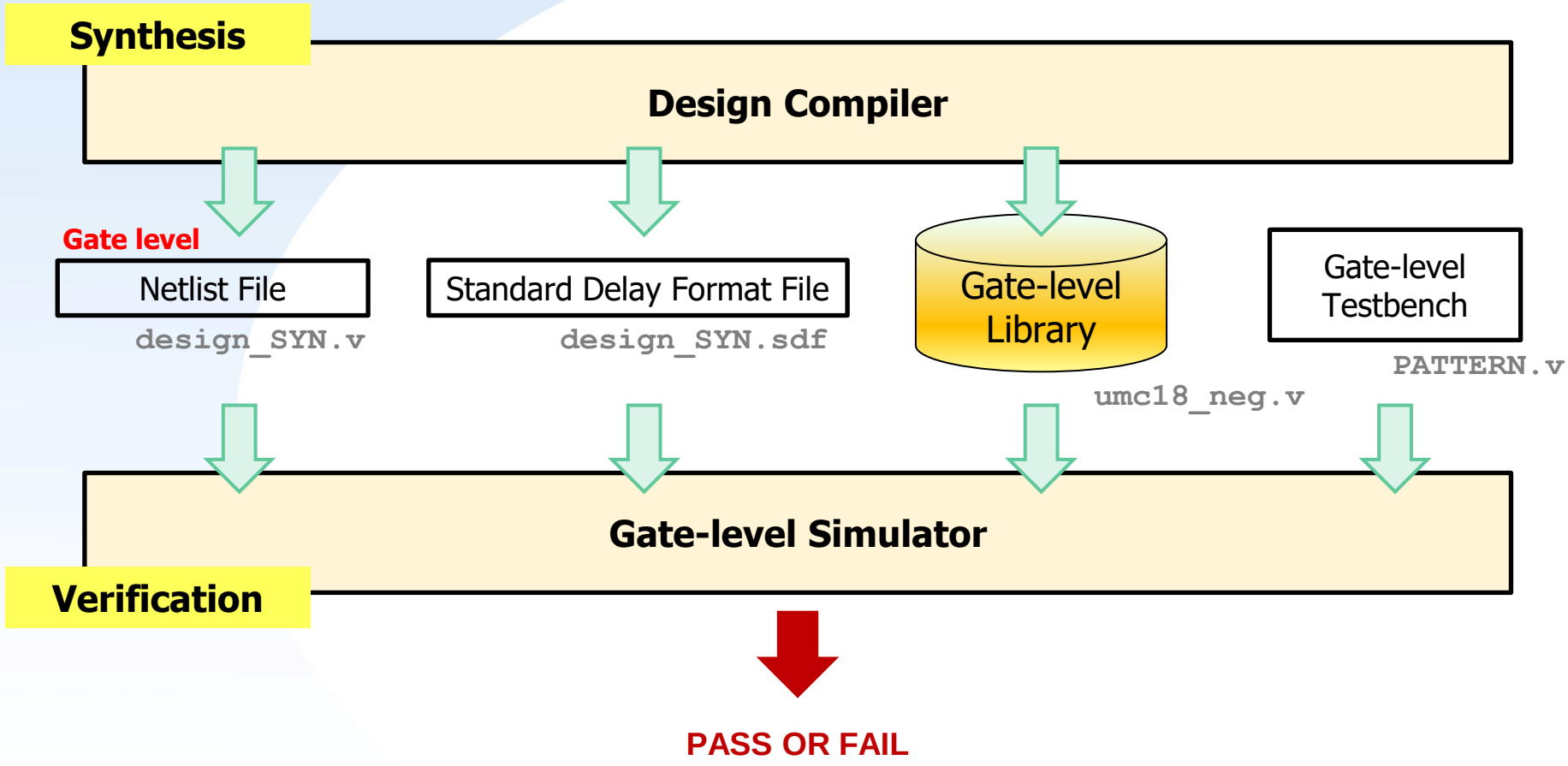
```
module mux_0 ( out, sel, b, a );
  input sel, b, a;
  output out;
  wire   sela, n2, n3;

  AND2HD4X __tmp102 ( .A(n2), .B(a), .Z(sela) );
  INVHDPX U1 ( .A(n3), .Z(out) );
  AOI21HD1X U2 ( .A(b), .B(sel), .C(sela), .Z(n3) );
  INVHDPX U3 ( .A(sel), .Z(n2) );
endmodule

module dffr_0 ( q, q_, d, clk, rst_ );
  input d, clk, rst_;
  output q, q_;
  wire   N0, N1, N3, N5, dl, qe, de, dl_;

  AND3HD2X n6 ( .A(dl_), .B(q), .C(rst_), .Z(N5) );
  AND3HD2X n4 ( .A(dl), .B(clk), .C(qe), .Z(N3) );
  AND3HD2X n2 ( .A(clk), .B(de), .C(rst_), .Z(N1) );
  AND2HD4X n1 ( .A(dl), .B(qe), .Z(N0) );
  NAND2HD4X U1 ( .A(q_), .B(qe), .Z(q) );
  INVCLKHD3X U3 ( .A(N5), .Z(q_) );
  INVCLKHD2X U2 ( .A(N0), .Z(de) );
  INVHD2X U4 ( .A(N1), .Z(qe) );
  NAND3HD1X U5 ( .A(d), .B(dl_), .C(rst_), .Z(dl) );
  INVCLKHD2X U6 ( .A(N3), .Z(dl_) );
endmodule
```

Reference : https://zhuanlan.zhihu.com/p/605629775?utm_id=0

Data Flow in Design Compiler



Gate-level Simulation



Design Compiler Introduction(cont.)

- **Design compiler provides two user interfaces**
  - command-line interface

    - **dctcl mode :** Applying a script based on tool command language (tcl) which packages all the commands needed to implement specific Design Compiler functionality.

      %dc_shell-t

    Used in this course

  - graphical user interfaces (GUI)
    - Design Vision (can execute in both modes) **%dv**
    - **%dc_shell> gui_start**
    - **%dc_shell> gui_stop**

Some Design Compiler comment

- **Get topic command help**
  - %dc_shell> help
- **Get a particular command help**
  - %dc_shell> command_name -help
- **Get topic command help**
  - %dc_shell> man command_name
    - ex: man set
- **More information about design compiler in user guide:**
  - http://archive.eclass.uth.gr/eclass/modules/document/file.php/MHX303/Documentation/dcug.pdf

Synthesize by Design Compiler

- **Prepare RTL code(.v) and script file(.tcl)**
- **Invoke design compiler**
  - Command :
    %dc_shell-t -f script_filename.tcl |tee log_filename.log
  - Example :
    %dc_shell-t -f syn.tcl |tee syn.log → ./01_run_dc
  - -f :format
  - tee: store the result into file and print on the screen
- **Check log file(.log) to see if there are error(ex: Latch) messages**
- **Check report file(.report) to see if timing or area are met**
- **Run gate-level simulation**

System Object (Gate level)

```
module TOP(in1,in2,clk,out);
    input in1,in2,clk;
    output [1:0] out;

    wire inv0,inv1,bus0,bus1;
    ADDER U_ADD1(.AIN(in1), .BIN(in2), .Q0(bus0), .Q1(bu1));

INV U_INV1(.A(bus0), .Z(inv0));
INV U_INV2(.A(bus1), .Z(inv1));

REGFILE U_REG(.D0(inv0),.D1(inv1),.CLK(clk), Q(OUT));

endmodule
```

design

clock

port

net

pin

reference/design

Design Objects

Design : A circuit that performs one or more logical functions

Cell : An instantiation of a design within another design

Reference: The original design that a cell “point to”

Port : The input or output of a design

Pin : The input or output of a cell

Net : The wire that connects ports or pins

Clock : Waveform applied to a port or pin identified as a clock source

No always block or assign

SDF File

✔ Standard Delay Format file (.sdf file)

```
(DELAYFILE
(SDFVERSION "OVI 2.1")
(DESIGN "CP")
(DATE "Wed Feb 26 18:12:26 2020")
(VENDOR "slow")
(PROGRAM "Synopsys Design Compiler cmos")
(VERSION "K-2015.06-SP1")
(DIVIDER /)
(VOLTAGE 1.62:1.62:1.62)
(PROCESS "slow")
(TEMPERATURE 125.00:125.00:125.00)
(TIMESCALE 1ns)
(CELL
  (CELLTYPE "ADDHXL")
  (INSTANCE U9469)
  (DELAY
    (ABSOLUTE
    (IOPATH A CO (0.218262:0.218911:0.218911) (0.221926:0.224530:0.224530))
    (IOPATH B CO (0.218498:0.219461:0.219461) (0.207622:0.211288:0.211288))
    (COND B == 1'b1 (IOPATH A S (0.298010:0.300566:0.300566) (0.206581:0.207653:0.207653)))
    (COND B == 1'b0 (IOPATH A S (0.355384:0.356453:0.356453) (0.336261:0.338810:0.338810)))
    (IOPATH (posedge A) S (0.355384:0.356453:0.356453) (0.336556:0.337945:0.337945))
    (IOPATH (negedge A) S (0.355157:0.357118:0.357118) (0.336261:0.338810:0.338810))
    (COND A == 1'b1 (IOPATH B S (0.235169:0.238284:0.238284) (0.141208:0.142291:0.142291)))
    (COND A == 1'b0 (IOPATH B S (0.207619:0.208926:0.208926) (0.230646:0.234490:0.234490)))
    (IOPATH (posedge B) S (0.238927:0.240244:0.240244) (0.235283:0.236908:0.236908))
    (IOPATH (negedge B) S (0.235169:0.238284:0.238284) (0.230646:0.234490:0.234490))
    )
  )
)
```

design

min : typ : max

rising edge transition

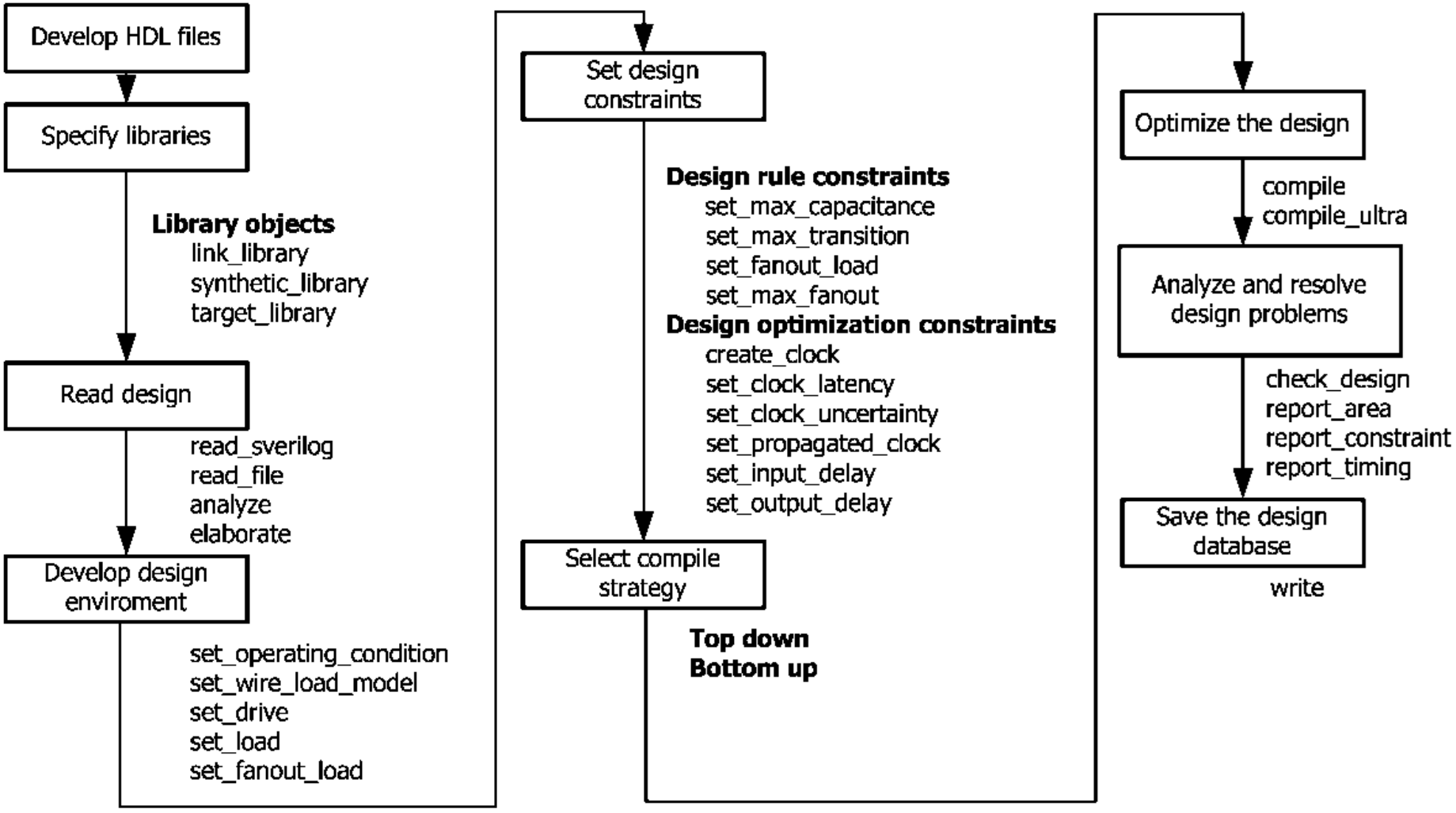
falling edge transition

Outline

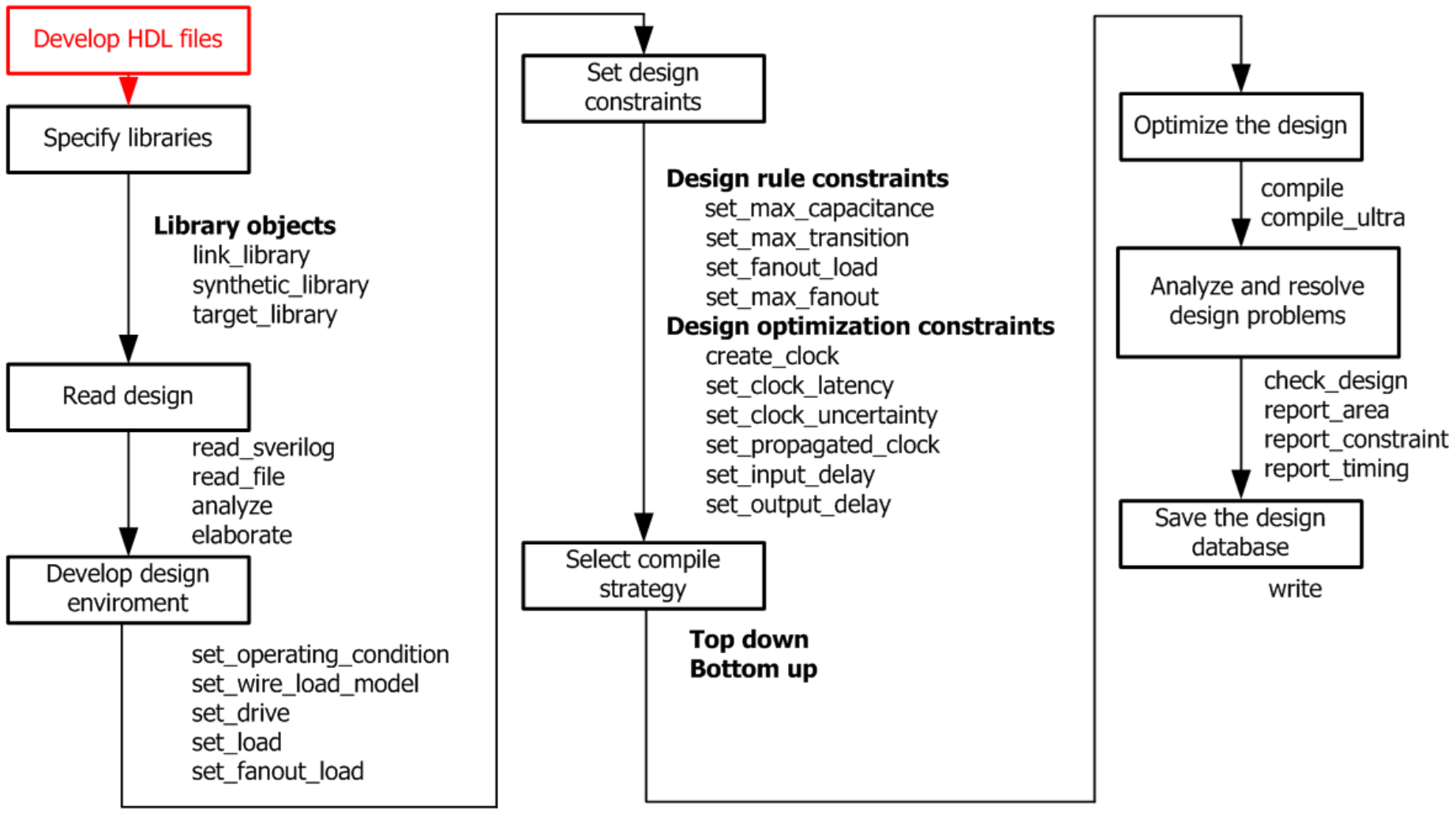
- ✔ Section 1 Design Compiler Introduction
- ✔ **Section 2 Basic Synthesis Flow**
  - Develop HDL files
  - Specify libraries
  - Read design
  - Develop design environment
  - Set design constraints
  - Select compile strategy
  - Optimize the design
  - Analyze and resolve design problems
  - Save the design database
- ✔ Section 3 Generate & For Loop

More details in page10 pdf

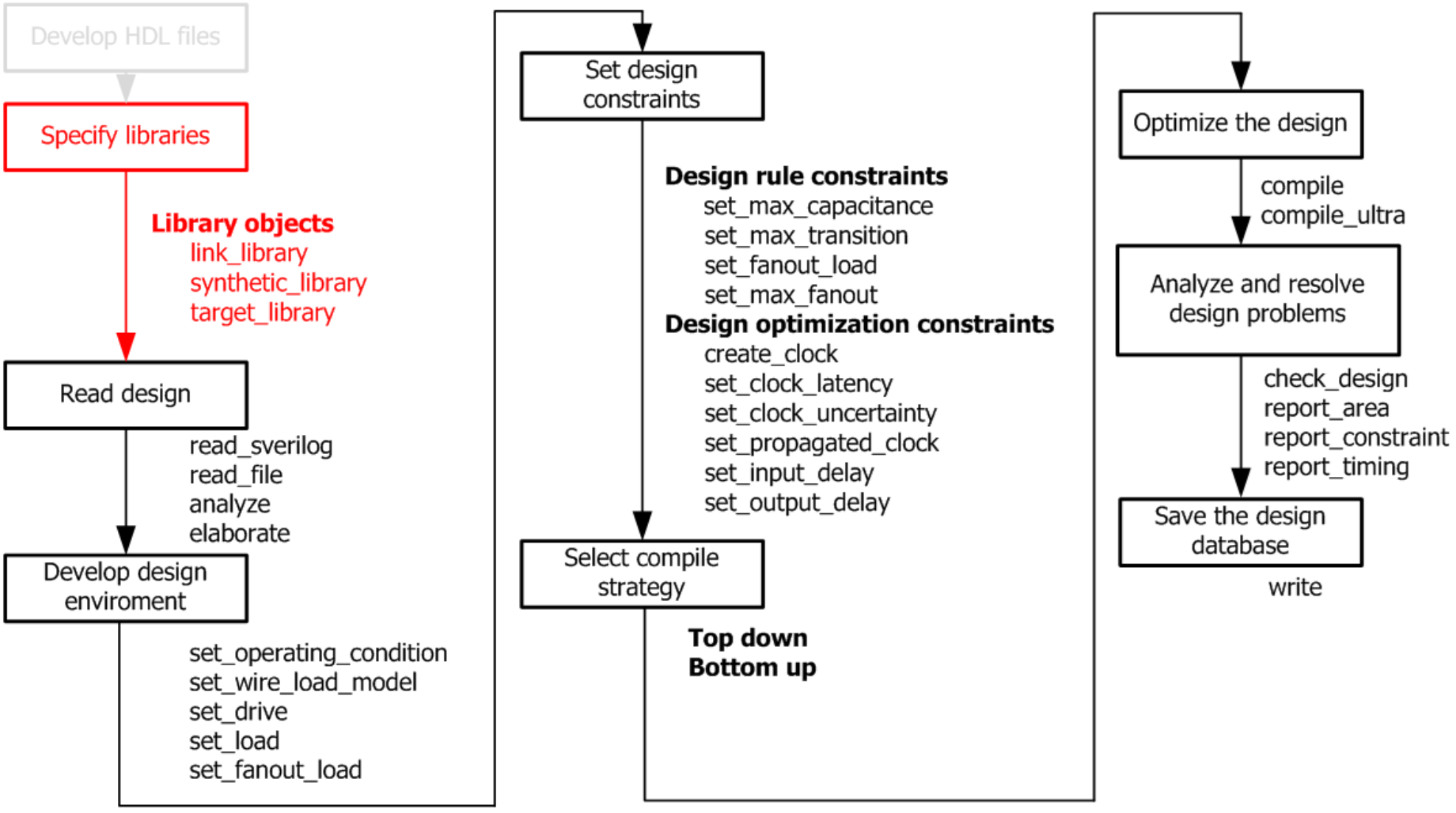
Basic Synthesis Flow



Basic Synthesis Flow



Basic Synthesis Flow



Specify Libraries

Specify libraries

1. Synthetic Library
   - Specifies additional **DesignWare libraries** for optimization purposes.
   - Efficient implementations for adders, comparators, multipliers
2. Link Library
   - Specifies a list of libraries that Design Compiler can use to resolve design references.
3. Target Library
   - During synthesis, compiler selects gates from target library.
   - It also calculates the timing of the circuit, using the vendor-supplied (UMC, TSMC …) timing data of the lib.

Synthesize in worst case. (xx.sldb)

```
set search_path {./../01_RTL \
                 ~iclabta01/umc018/Synthesis/ \
                 /usr/synthesis/libraries/syn/ \
                 /usr/synthesis/dw/ }

1 set synthetic_library {dw_foundation.sldb}

2 set link_library {* dw_foundation.sldb standard.sldb slow.db}

3 set target_library {slow.db}
```

.lib file

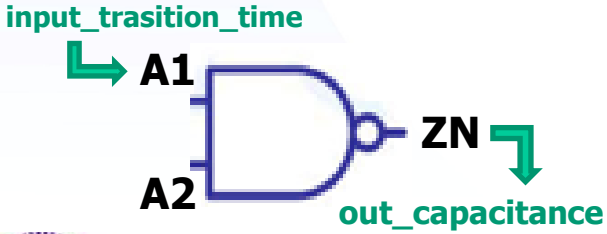
- ✔ Cell name
- ✔ Drive strength
- ✔ Area
- ✔ Pin
  - Internal delay
  - Leakage power
  - Internal power

```
cell (NANDX1) {
  pin(A1) {
    direction : input;
    capacitance : 0.00683597;
  }
  pin(A2) {
    direction : input;
    capacitance : 0.00798456;
  }
  pin(ZN) {
    direction : output;
    capacitance : 0.0;
    internal_power() {
      timing() {
        cell_rise(table10){
          values ("0.020844,0.02431,0.030696,0.039694,0.048205,0.072168,0.10188",\
                  "0.024677,0.027942,0.035042,0.045467,0.054973,0.082349,0.11539",\
                  "0.032068,0.035394,0.042758,0.055361,0.065991,0.090936,0.13847",\
                  "0.046811,0.049968,0.057164,0.064754,0.086481,0.11676,0.15744",\
                  "0.073919,0.078805,0.080873,0.091007,0.11655,0.1579,0.21448",\
                  "0.13162,0.13363,0.1383,0.14793,0.1685,0.22032,0.30054",\
                  "0.24661,0.24835,0.25294,0.26221,0.282,0.32417,0.42783");
```

Same information as .db file

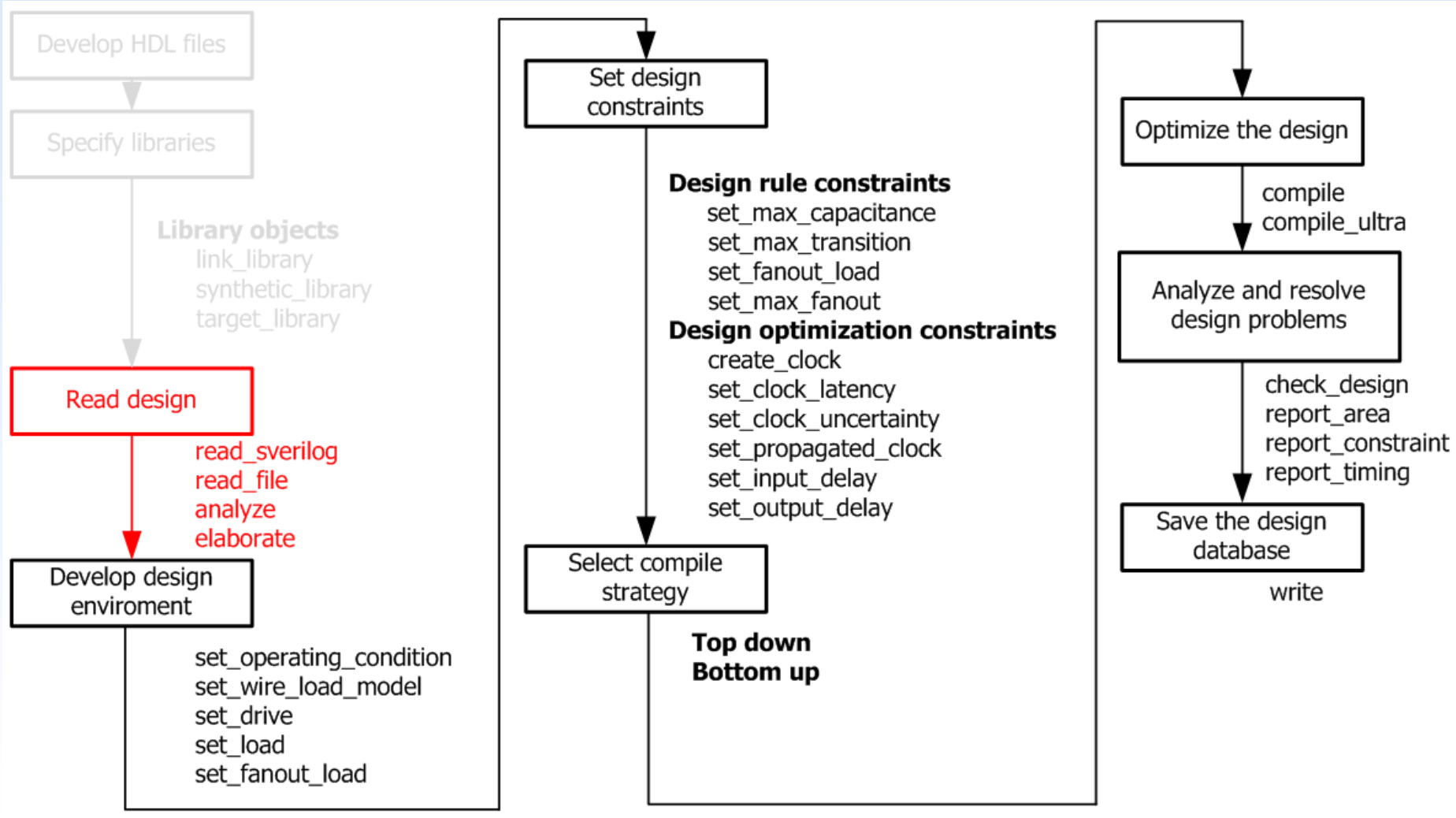
out_capacitance

input_trasition_time




```
lu_table_template(table10){
  variable_1 : total_output_net_capacitance;
  variable_2 : input_transition_time;
  index_1 ("0.001400,0.003000,0.006200,0.012500,0.025100,0.050400,0.101000");
  index_2 ("0.0208,0.0336,0.06,0.1112,0.2136,0.4192,0.8304");
}
```

Basic Synthesis Flow



Read Design

Read design

- The Design Compiler optimization process works only on the designs loaded in memory.

Method 1: read_file , read_verilog , read_sverilog
Method 2: analyze & elaborate
(Please refer to the appendix for the detail usage of them)

Setting the Current Design

- After the read file command, set the current design on the module you want to focus on.

Method 1

```
#=====================================
#  Read RTL Code
#=====================================
set hdlin_auto_save_templates TRUE
read_verilog {$DESIGN\.v  INV_IP.v}
current_design $DESIGN
```

Method 2

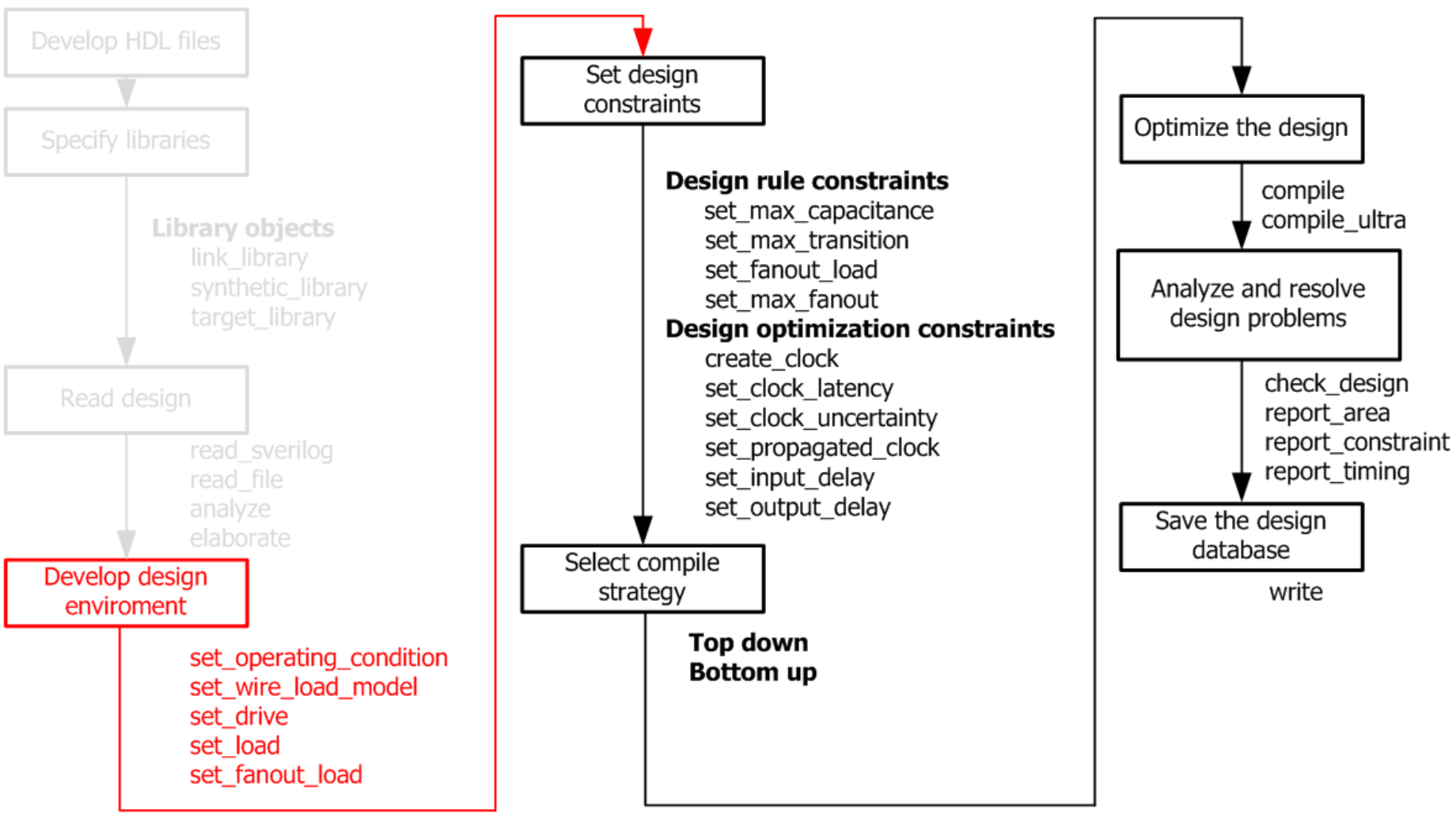
```
analyze -f verilog $DESIGN\.v
analyze -f verilog INV_IP.v
elaborate $DESIGN
current_design $DESIGN (optional)
```

Note: read_sverilog is compatible to read the Verilog format

Read Design

	read_file	analyze and elaborate
Input format	All formats	VHDL, Verilog
When to use	Netlist, precompiled designs	Synthesize VHDL or Verilog
Generics	Cannot pass parameters (must use directives in HDL)	Allow user to set parameter values on the elaborate command line e.g. `elaborate $DESIGN -parameter "IP_WIDTH=6"` Use Carefully!
Design libraries	Cannot store analyzed results	Can store analyzed results in specified design libraries
Commands	**dc_shell>read_file –f keyword file_name** **e.g.:** **dc_shell>read_file –f verilog name**	**dc_shell>analyze –f keyword file_name** **dc_shell>elaborate design_name**
Architecture	Cannot specify architecture to be elaborated	Allow user to specify architecture to be elaborated

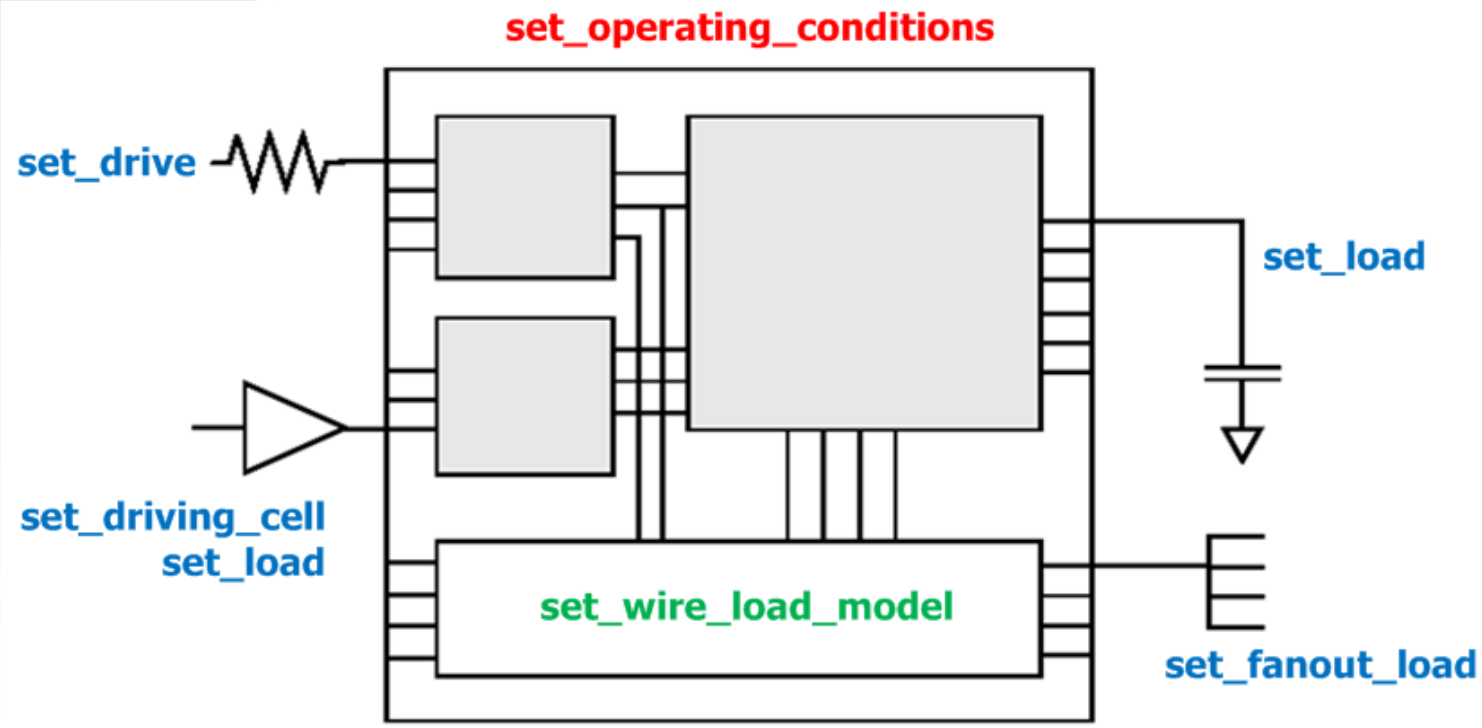
Basic Synthesis Flow



Define Design Environment

Design Environment

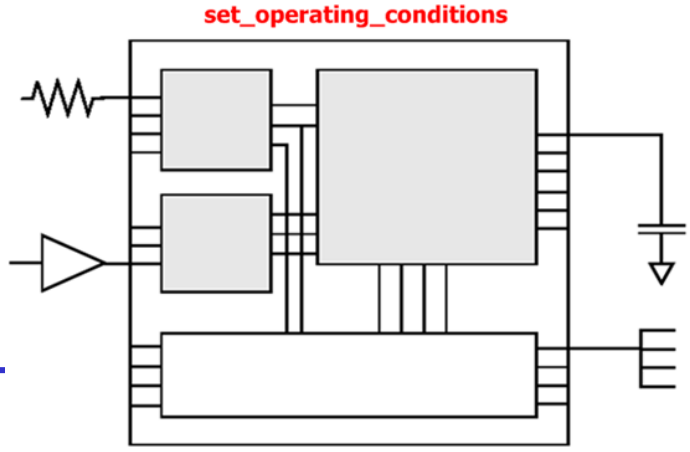
- Define the environment in which the design is expected to operate in by specifying operating conditions, wire load models, and system interface characteristics.



Commands Used to Define the Design Environment

Define Design Environment (Operating Conditions)

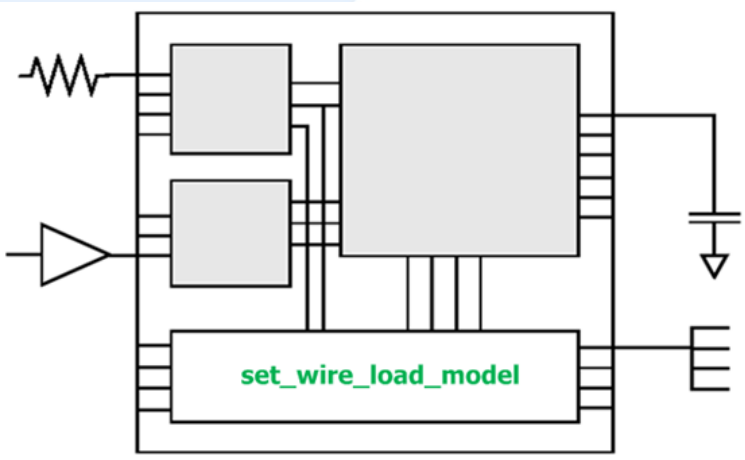
- **Defining the Operating Conditions (PVT Variations)**
  - An operating condition is defined as a combination of **Process, Voltage, Temperature(PVT).**
  - There are three kinds of manufacturing process models that are provided by the semiconductor foundry for digital designs: *slow* process model, *typical* process model, *fast* process model.
  - For robust design, the design should be validated at the extreme corners (*slow*, *fast* process model) of the manufacturing process at last.
  - Generally, for synthesis stage, we use worst-case to ensure that the timing (setup time) is met under strict constraints.

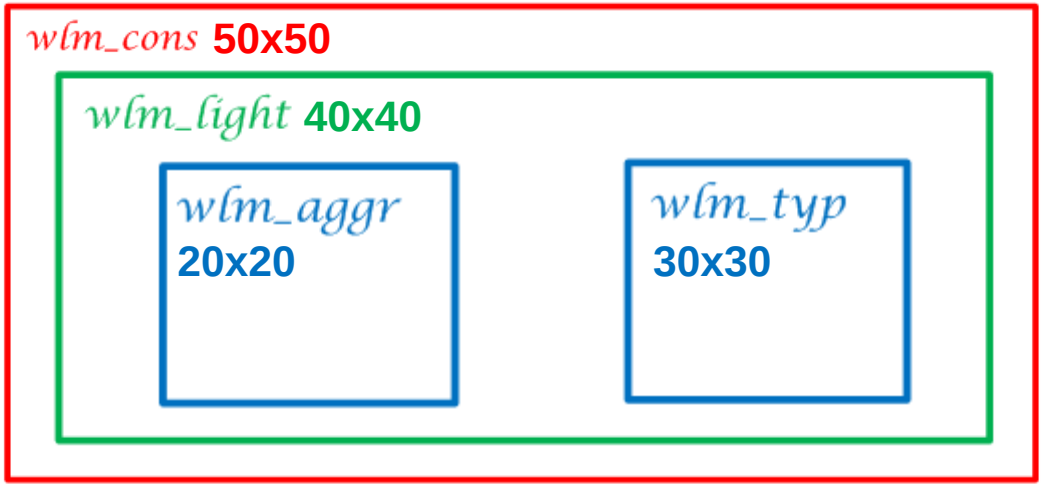


Define Design Environment (Wire Load Models)

Defining Wire Load Models

- Before layout(APR), wire load models can be used to estimate capacitance, resistance and the area overhead due to interconnection.





e.g.

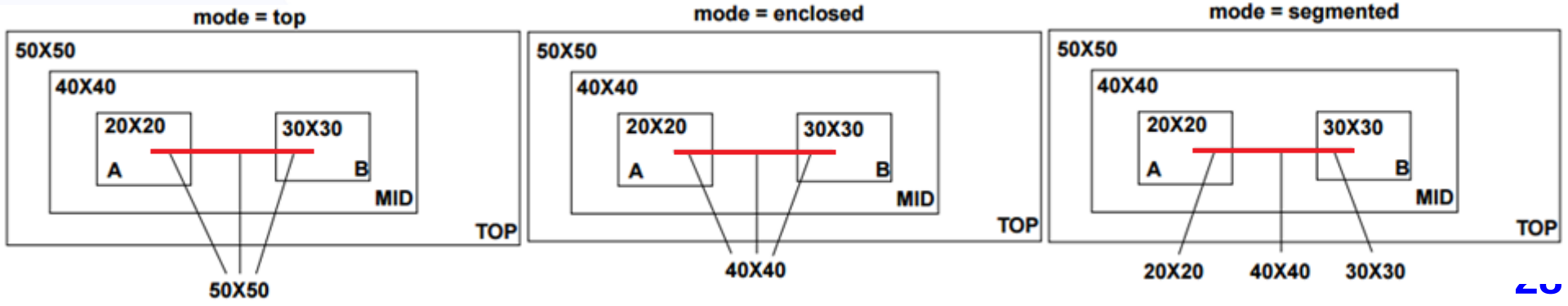
set_wire_load_model –name “umc18_wl50”

Define Design Environment (Wire Load Models)

- **Design Compiler uses physical values to calculate wire delays and circuit speeds, there are three types of wire load mode.**

```
set_wire_load_mode top
```

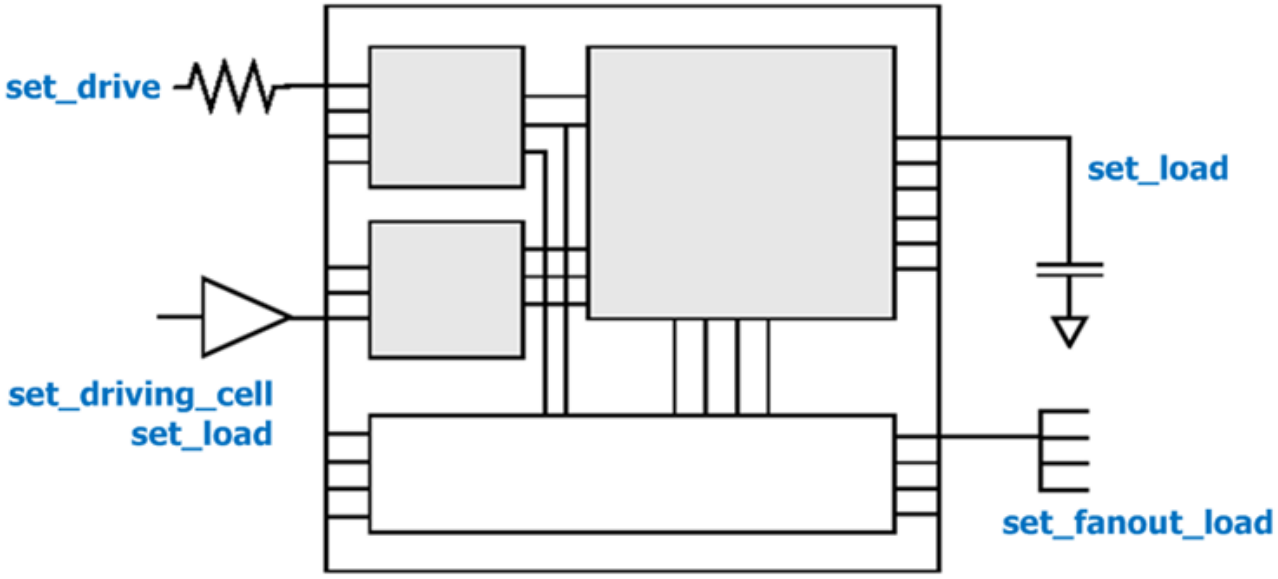
  - **Top**
    - Uses the wire load model specified for the top level of the design hierarchy for all nets
  - **Enclosed**
    - Uses the wire load model of the smallest module that fully encloses the net
  - **Segmented**
    - Each segment of the net gets its wire load model from the block that encompasses the net



Define Design Environment (System Interface)

Modeling the System Interface

- Design Compiler supports the following ways to model the design's interaction with the external system:
  - Defining drive characteristics for input ports
  - Defining fan out loads on output ports
  - Defining loads on input and output ports

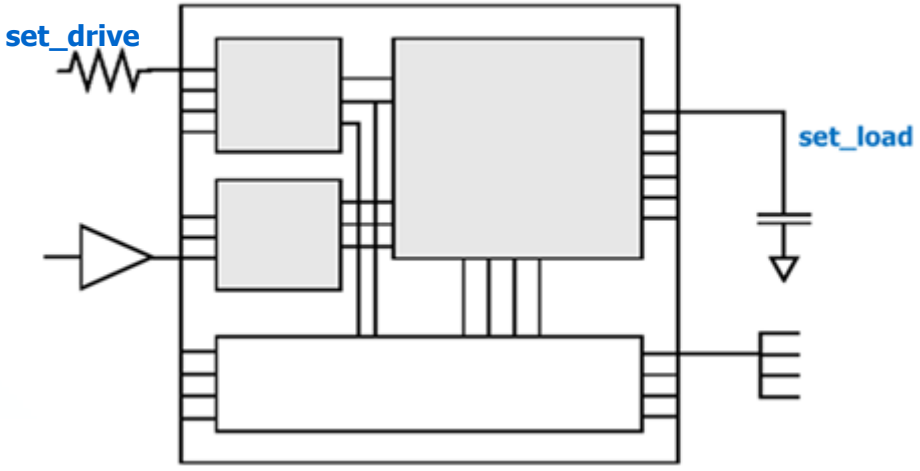


Define Design Environment (System Interface)

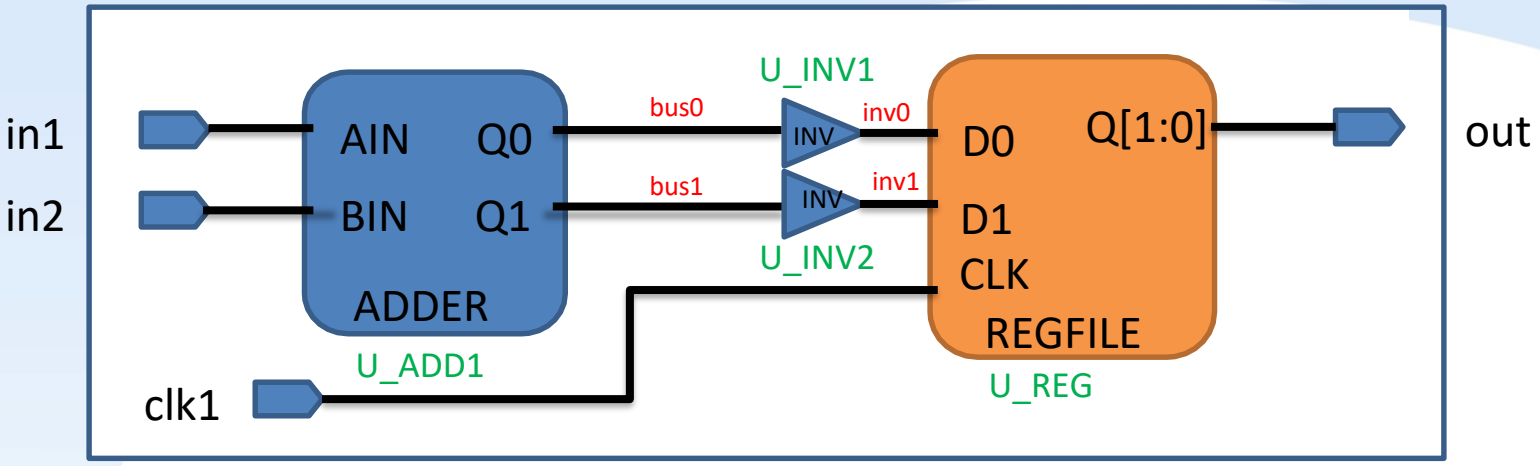
- **e.g. defining loads on input and output ports**
  - By default, it assumes zero capacitive load on input and output ports.
  - Use the set_load command to set a capacitive load value on input and output ports of the design.
  - Example : Set a load of 0.05 picofarads on all output ports, set a drive of 1.5 kilo-ohms on all input ports.

set_drive 1.5 [all_inputs]

set_load 0.05 [all_outputs]



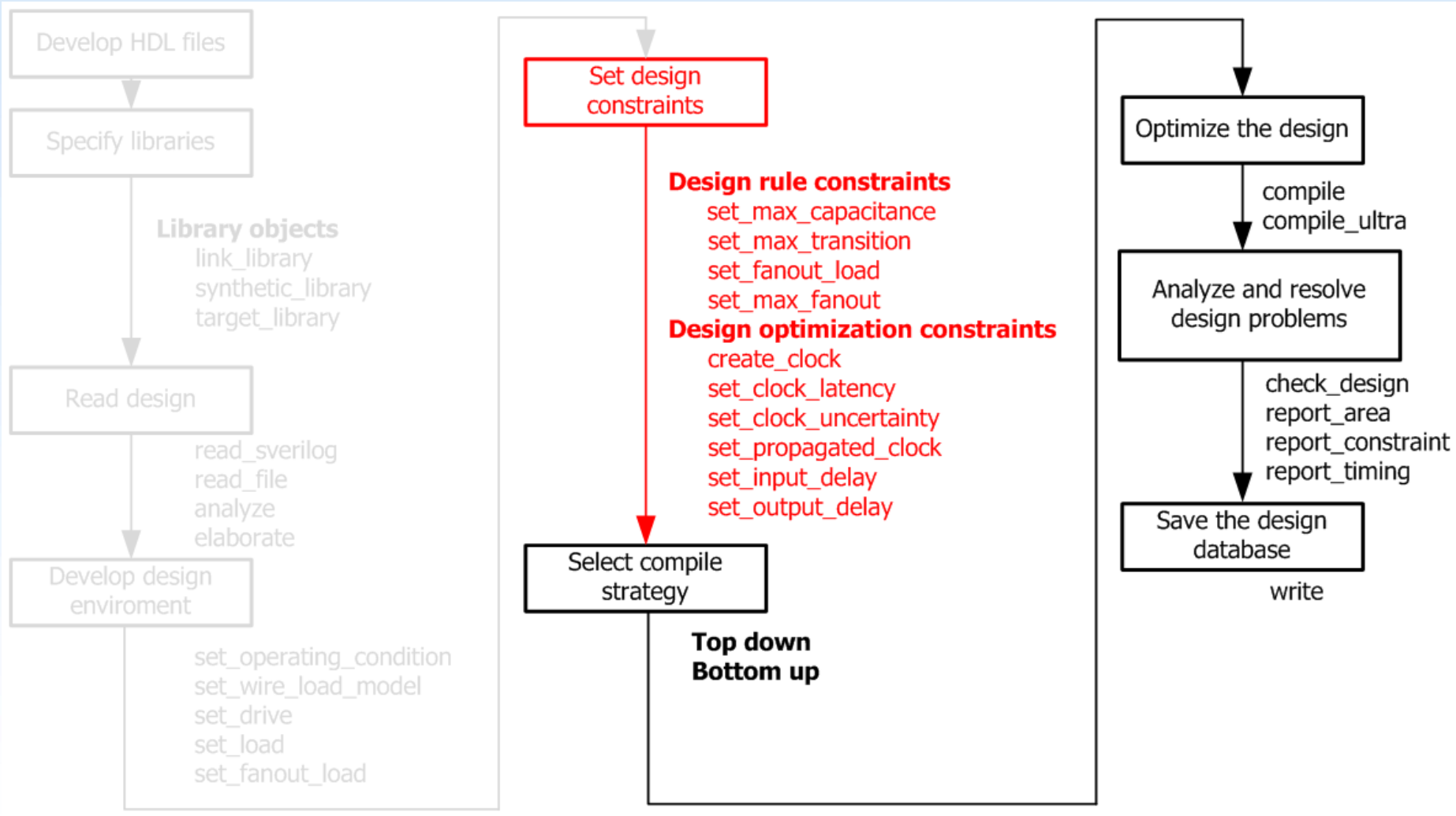
Select Design Object




```
Example:
[get_ports *]          : in1, in2, out, clk1
[get_designs *]        : top
[get_pins U_ADD1/*]    : AIN, BIN, Q0, Q1
[get_pins U_REG/*]     : D0, D1, CLK, Q
[get_cells *]          : U_ADD1, U_REG, U_INV1, U_INV2

[get_clocks *]         : clk1
[get_nets *]           : bus0, bus1, inv0, inv1

Frequently Used :
[all_inputs]           : in1, in2, clk1
[all_outputs]          : out
[all_ports]            : in1, in2, clk1, out
```

Reserved Word: clk, rst_n

Basic Synthesis Flow



Set Design Constraints

✔ Constraint Priorities: (default)

Priority (descending order)	Notes
connection classes	
multiple_port_net_cost	
min_capacitance	Design rule constraint
max_transition	Design rule constraint
max_fanout	Design rule constraint
max_capacitance	Design rule constraint
cell_degradation	Design rule constraint
max_delay	Optimization constraint
min_delay	Optimization constraint
power	Optimization constraint
area	Optimization constraint
cell count	

Design rule constraint

Optimization constraint

Set Design Constraints(cont.)

- **There are two categories of design constraints**

Design rule constraints

- Design rule constraints reflect technology-specific restrictions your design must meet in order to function as intended.
- Most technology libraries specify default design rules.
- You can apply more restrictive design rules, but you cannot apply less restrictive ones.

Design optimization constraints

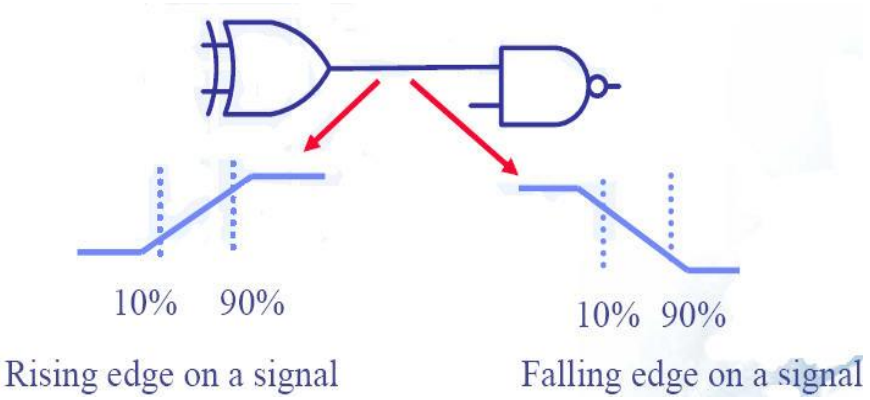
- User defines speed(timing) and area optimization goals for Design Compiler.
- Speed(timing) constraints have higher priority than area.(The priority can be changed)
- Optimization constraints are secondary to design rule constraints.

Set Design Constraints(cont.)

- **Design Constraints**
  - **Design rule constraints**
    - **set_max_capacitance**
    - **set_max_transition**
    - **set_fanout_load**
    - **set_max_fanout**
  - **Design optimization constraints**

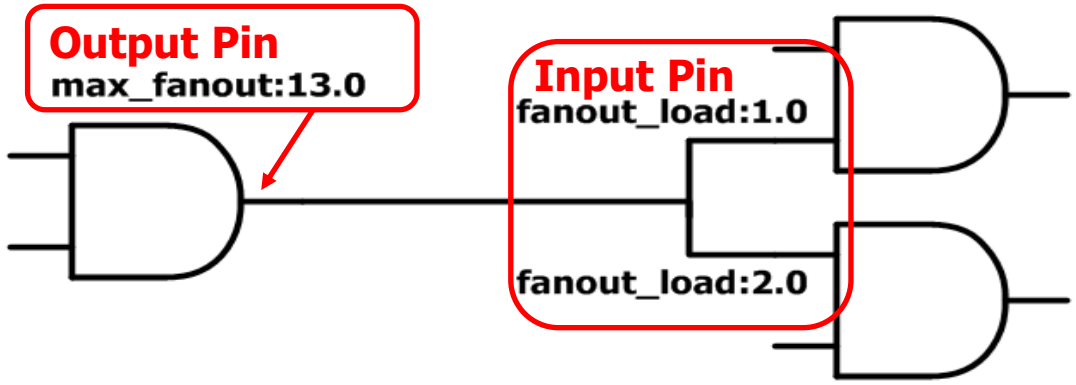
Design Rule Constraints

- **Maximum capacitance**
  - **dc_shell>set_max_capacitance cap_value port_list**
  - It is set as a pin-level attribute that defines the maximum total capacitive load that an output pin can drive.
- **Maximum transition**
  - **dc_shell>set_max_transition trans_value port_list**
  - The maximum transition time for a net is the longest time required for its driving pin to change logic values.



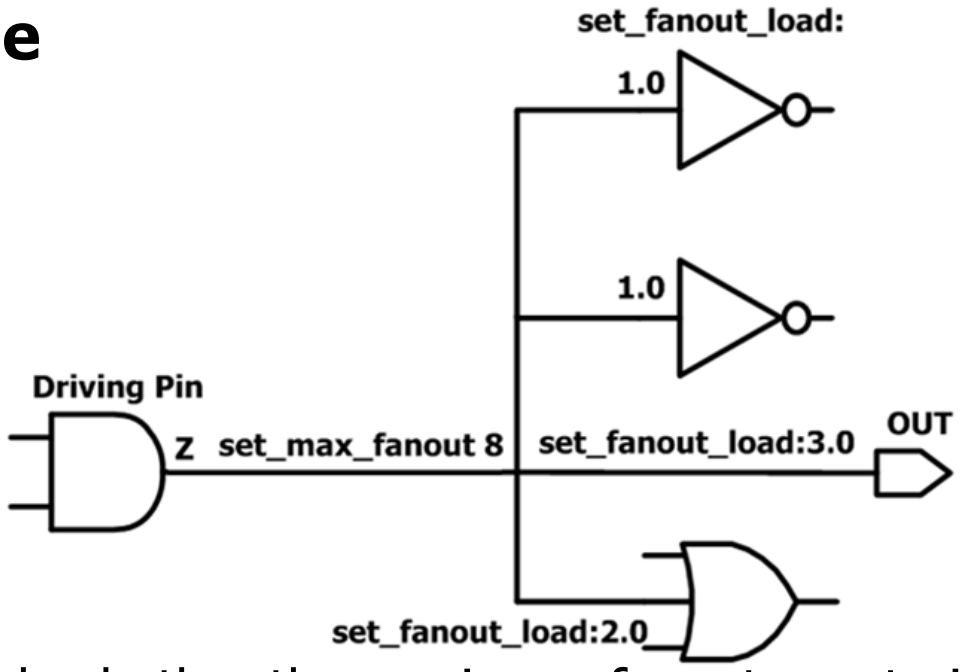
Design Rule Constraints(cont.)

- **Fan-out load**
  - **dc_shell>set_fanout_load cap_value port_list**
  - Fan-out load is a dimensionless number, not a capacitance. It represents a numerical contribution to the total effective fan-out.
- **Maximum fan-out :**
  - **dc_shell>set_max_fanout cap_value port_list**
  - The maximum fan-out load for a net is the maximum number of loads the net can drive.
  - If a library fan-out constraint exists and a max_fanout attribute is specified, Design Compiler tries to meet the more restrictive value.



Design Rule Constraints(cont.)

Example



- 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 constraint is met.

$$8 \geq \underbrace{1.0 + 1.0 + 3.0 + 2.0}_{\text{Total Fanout Load}}$$

$$7$$

Design Rule Constraints(cont.)

- **In some cases, the nets should be set to ideal.**
  - Nets that are assigned ideal timing conditions—that is, latency, transition time, and capacitance are assigned a value of zero..
  - Such nets are exempt from timing updates, delay optimization, and design rule fixing.
  - **set_ideal_network net_list**
  - **e.g. set_ideal_network {clk}**

Set Design Constraints(cont.)

- **Design Constraints**
  - Design rule constraints
  - **Design optimization constraints**
    - create_clock
    - set_clock_latency
    - set_clock_uncertainty
    - set_propagated_clock
    - set_input_delay
    - set_output_delay
    - set_max_delay
    - set_fix_delay
    - set_false_path

Design Optimization Constraints

create_clock

- Defines the period and waveform for the clock

  Syntax :

  **create_clock –name "clk name for tcl" source_objects **
  -period period_value -waveform { rise fall }

  - source_objects : A list of pins or ports on which to apply this clock.
  - -period period_value : the period of the clock waveform in library time units(ns).
  - -waveform option : set the rising edge time and the falling edge time. If you do not specify the clock waveform, default waveform is a 50 percent duty cycle.

Example:

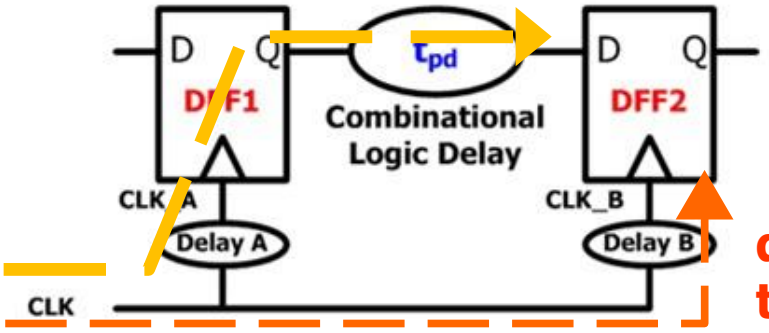
```
set CYCLE 10.0

create_clock -name "clk" [get_ports clk1]  -period $CYCLE
```

Design Optimization Constraints(cont.)

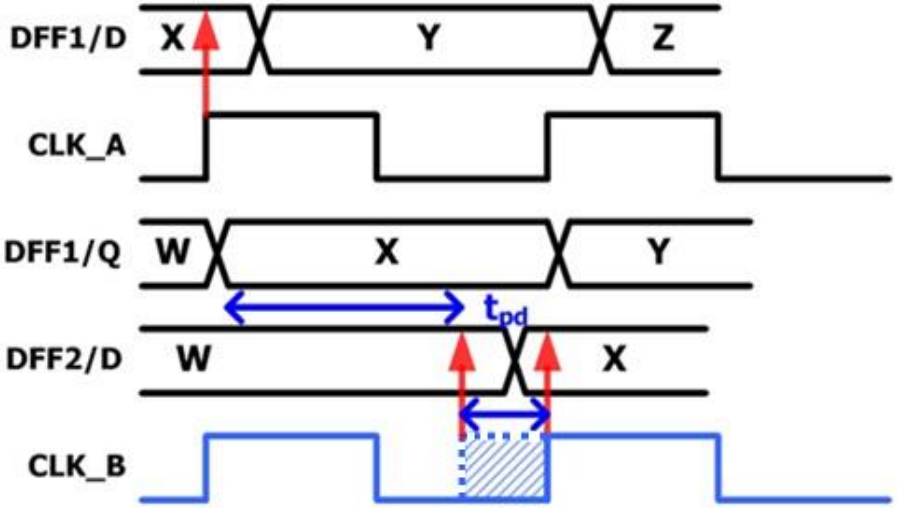
- **Design Compiler treats clock networks as ideal (having no delay) by default.**
- **You can override the default behavior to obtain nonzero clock network delay and specify information about the clock network delays, whether the skews are predicted or actual.**
- **dc_shell>set_clock_latency * (more details in Chapter 7)**
  - Define the delay from CLK to the register
- **dc_shell>set_clock_uncertainty * (more details in Chapter 7)**
  - Used to model various factors that can reduce the effective clock period.
- **dc_shell>set_propagated_clock * (more details in Chapter 7)**
  - Specify the clock latency be propagated throughout the clock network

Static Timing Analysis

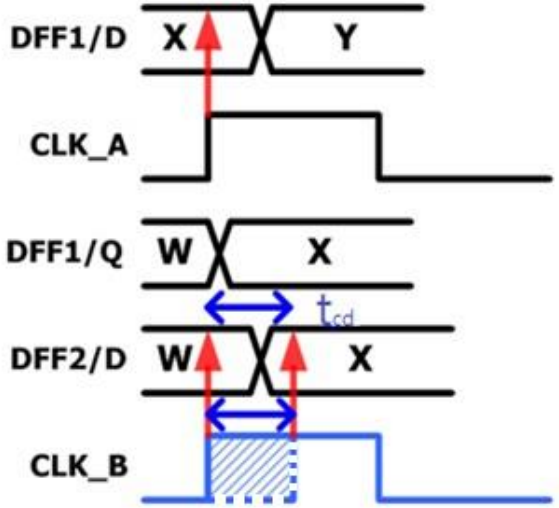
data arrival time (Launch path)



data required time (Capture path)



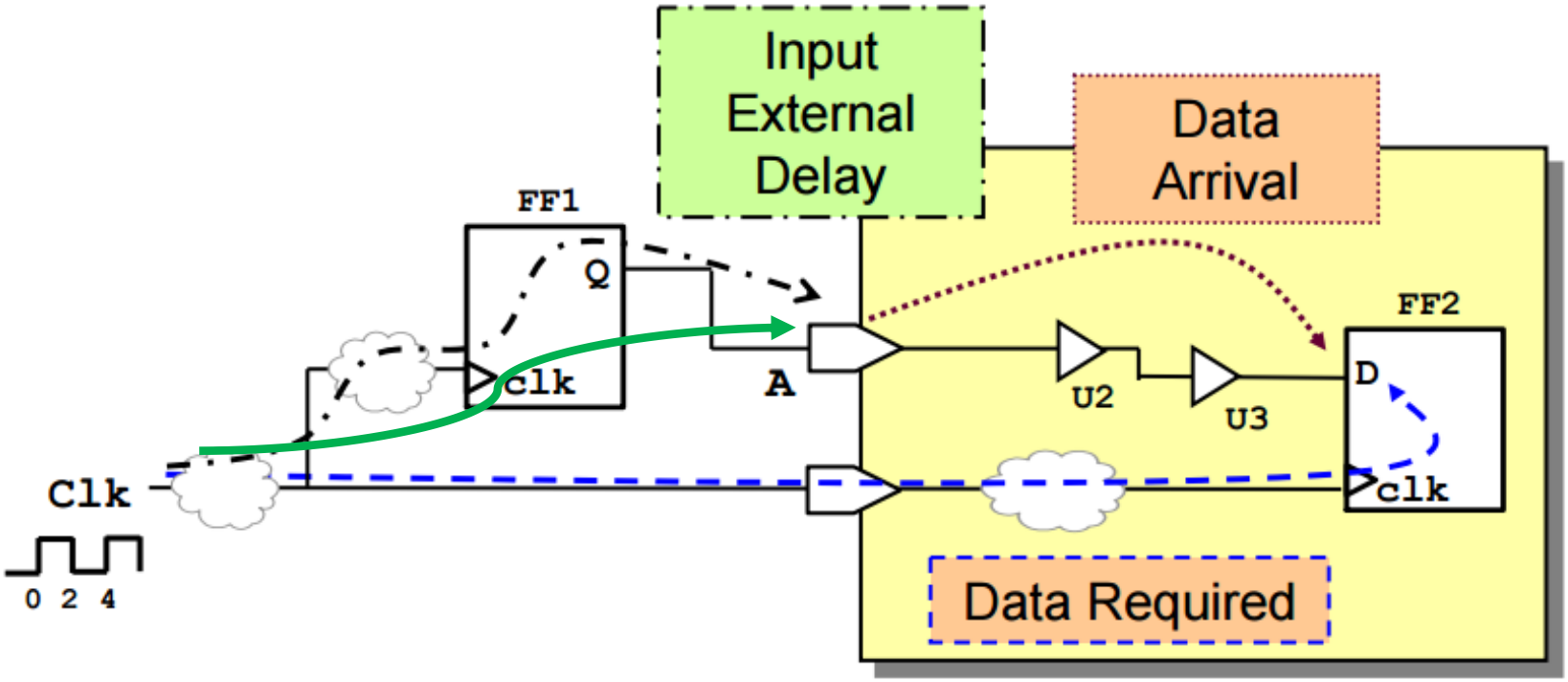
Setup Time Check



Hold Time Check

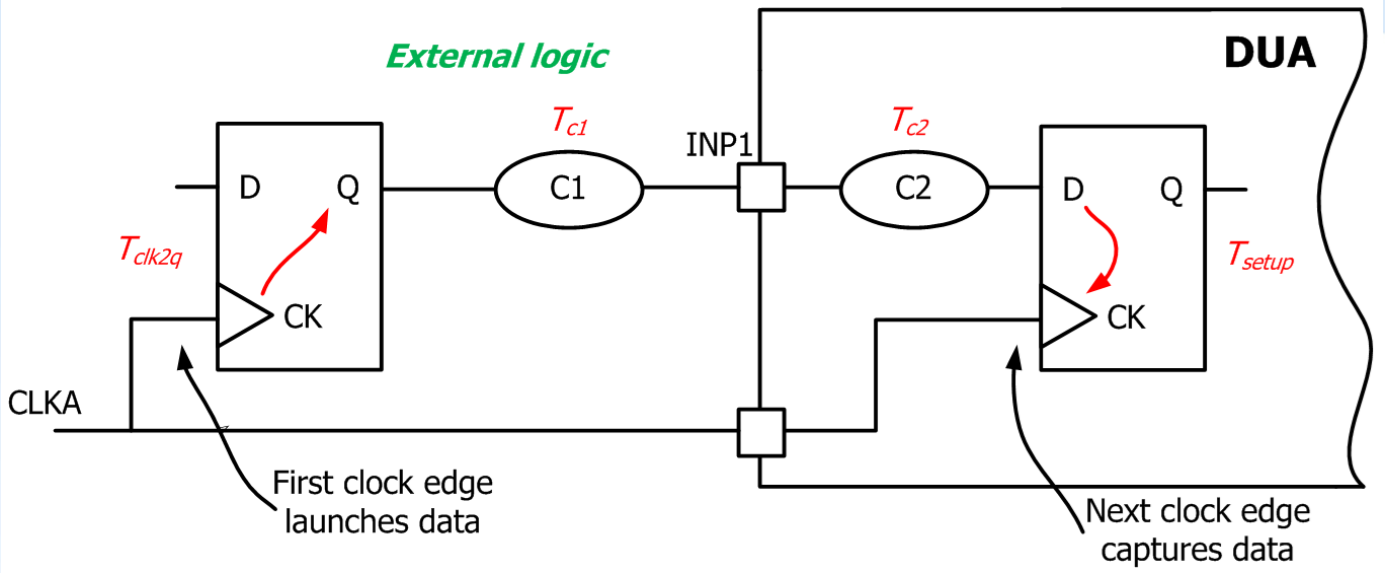
Constraining Input Paths

Specify the arrival time at the input ports of the design.



Constraining Input Paths(cont.)

- e.g.



DUA (Design Under Analysis)

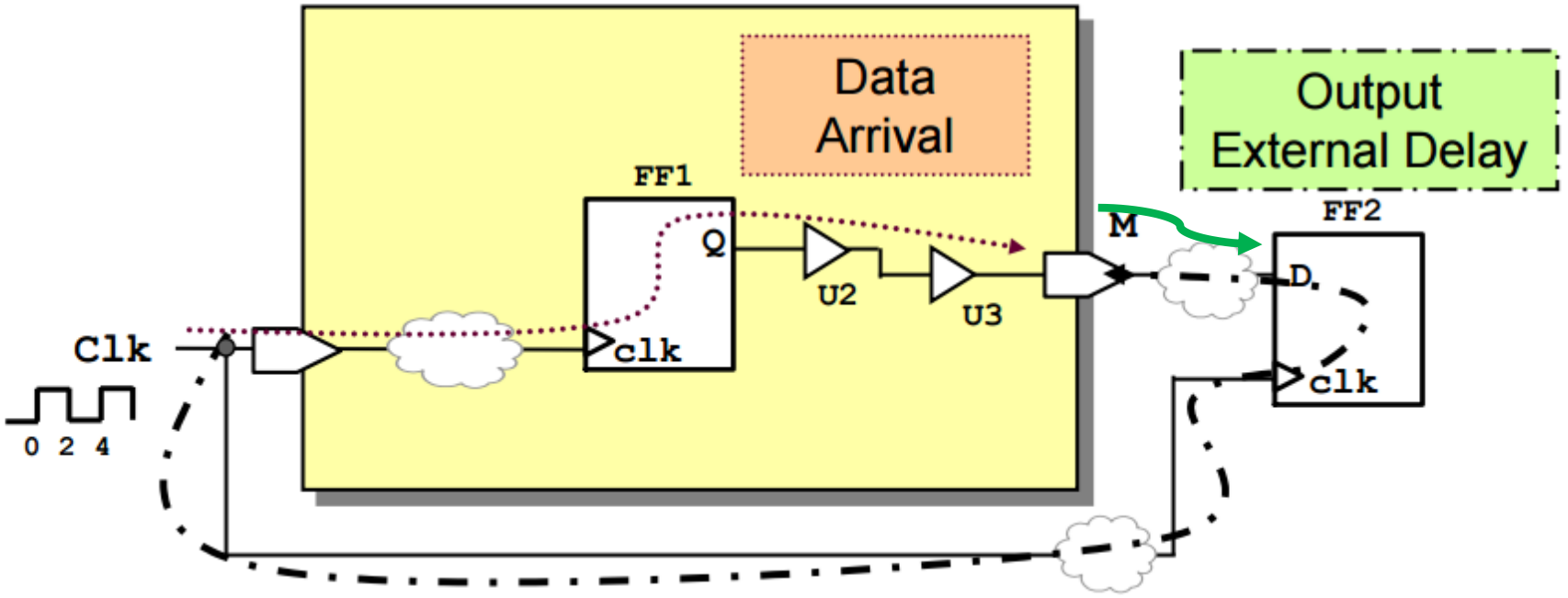
- **Input delay = (T_{clk2q} + T_{C1})**
- clock cycle ≧ **(T_{clk2q} + T_{C1})** + T_{C2} + Tsetup

- dc_shell>set T_{clk2q} 0.9
- dc_shell>set T_{C1} 0.6
- dc_shell>set_input_delay –clock CLKA –max [expr $$T_{clk2q}$+$$T_{C1}$] [all_inputs]

```
set CYCLE 7.0

set_input_delay  [ expr $CYCLE*0.5 ] -clock clk [all_inputs]
```

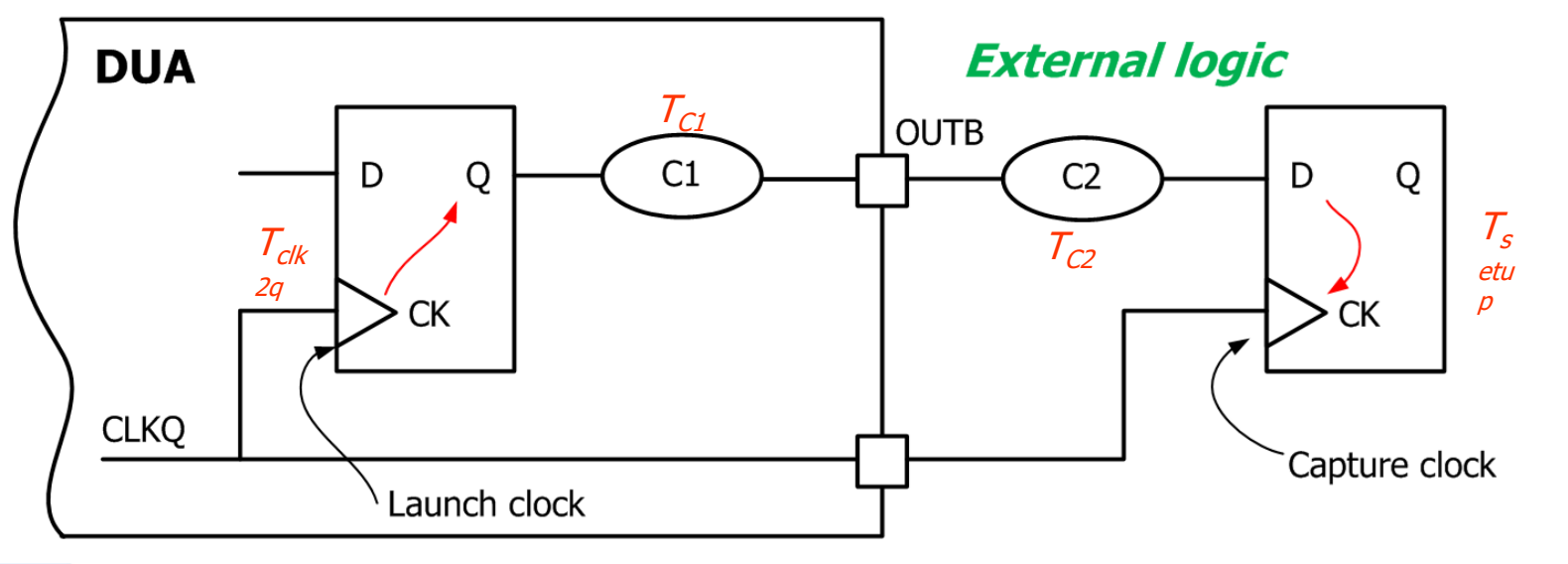
Constraining Output Paths

You specify the path required time at the output ports of the design.



Constraining Output Paths (cont.)

- e.g.



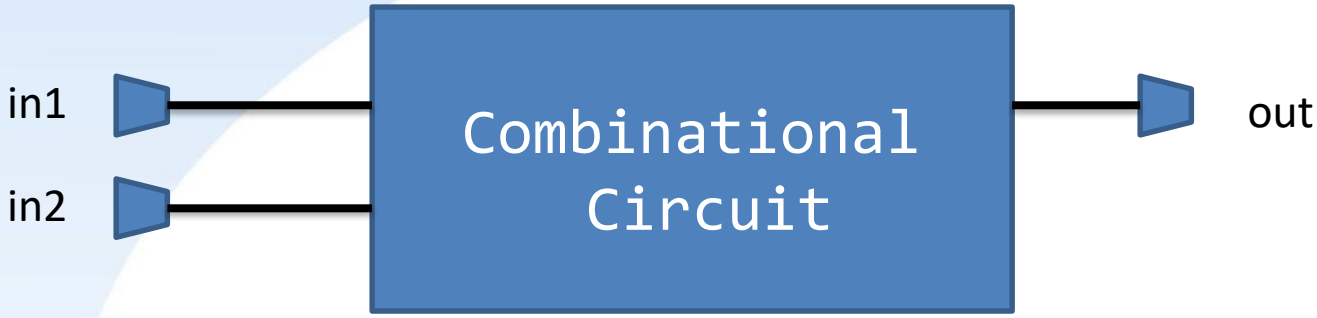
DUA (Design Under Analysis)

- **Output delay = (T_{C2} + T_{setup})**
- clock cycle ≧ T_{clk2q} + T_{C1} + **(T_{C2} + T_{setup})**

- dc_shell>set T_{C2} 3.9
- dc_shell>set T_{setup} 1.1
- dc_shell>set_output_delay –clock CLKQ –max [expr $$T_{C2}$+$$T_{setup}$] [all_outputs]

```
set CYCLE 7.0

set_output_delay [ expr $CYCLE*0.5 ] -clock clk [all_outputs]
```

Combinational Design Constraint



Syntax:

```
set_max_delay max_delay_value –from object –to object
```

!!! This only applies for Combinational Circuit Only !!!

Example:

```
set_max_delay $MAX_Delay -from [all_inputs] -to [all_outputs]
```

Design Optimization Constraints(cont.)

- **Design Compiler fix hold violations at register during compilation**
  - Set_fix_hold informs compile that hold time violations of the specified clocks should be fixed.
  - To fix a hold violation requires slowing down data signals.
  - Design Compiler considers the minimum delay cost only if the set_fix_hold command is used.
  - Generally, fixing and optimizing setup time violation are more important than hold time violation before CTS.

  - Syntax : **set_fix_hold clock_list**

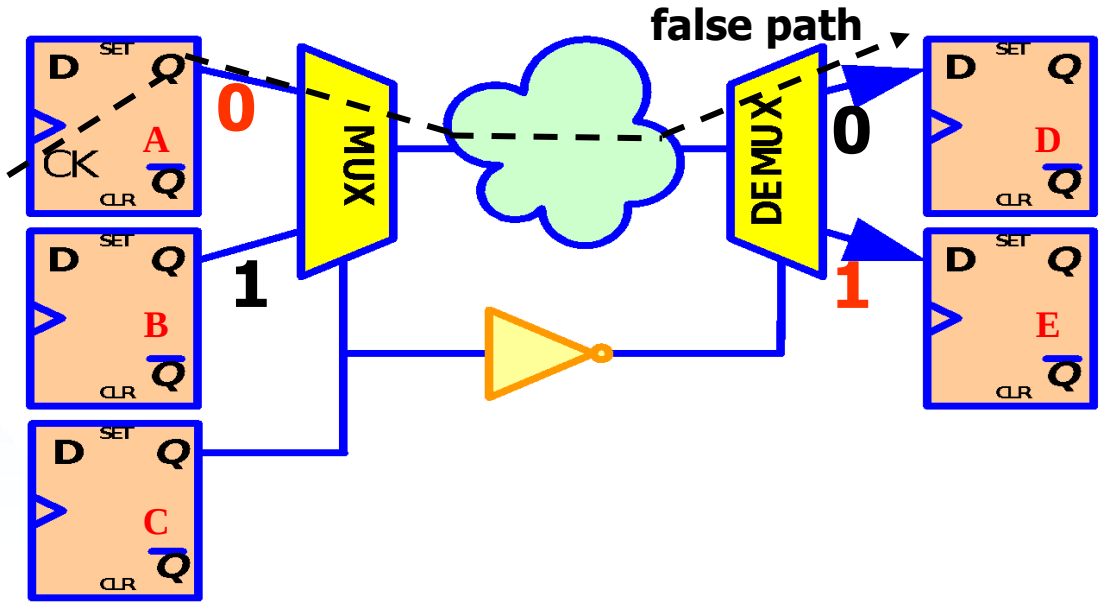
  e.g. :

  The following command sets a fix_hold attribute on clock "clk1".

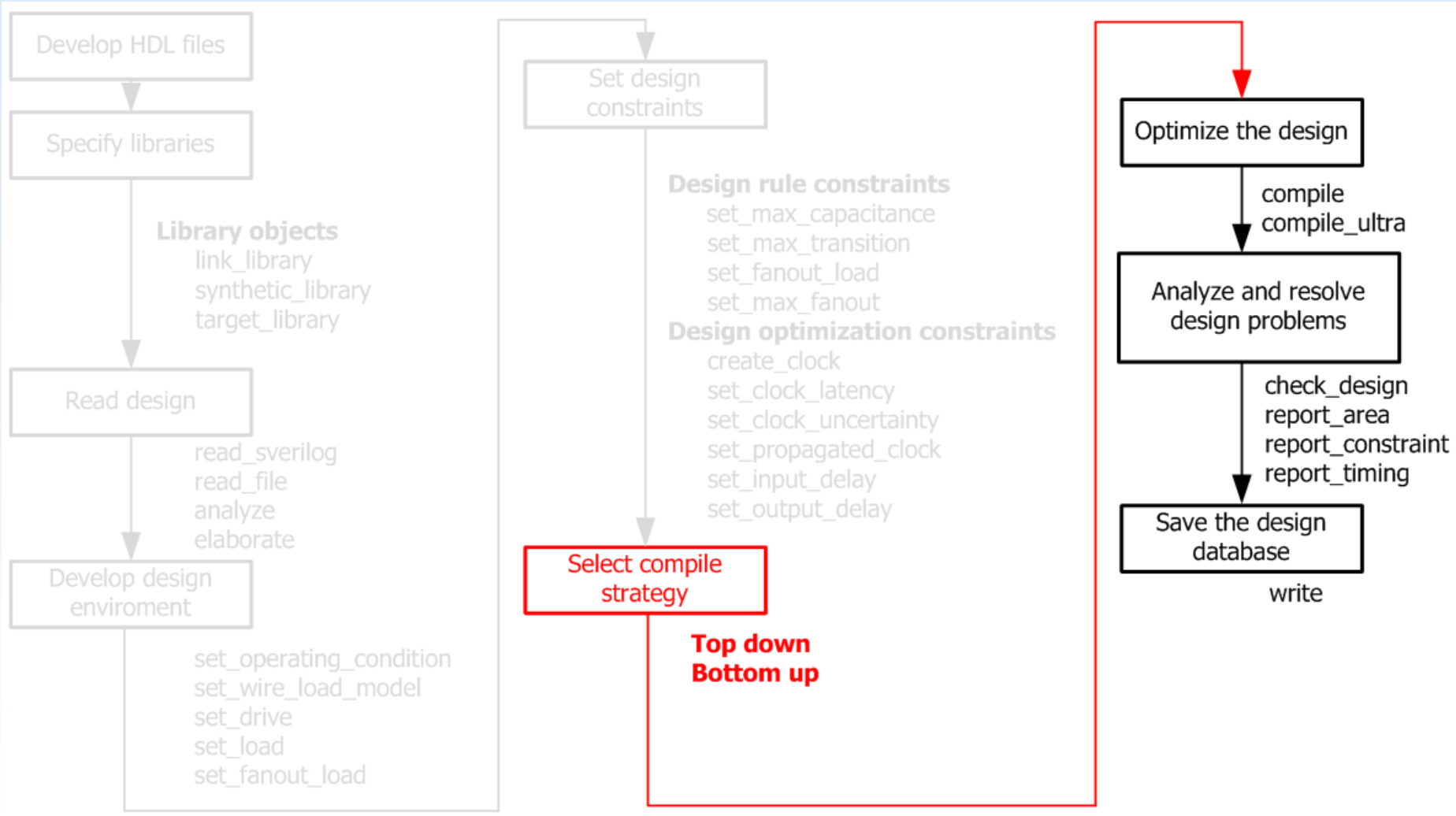
  dc_shell> set_fix_hold clk1

Specify False Path

- **Remove timing constraints from particular path**
  - Make compiler to ignore paths that never occur in normal operation
  - Timing checks of false path will be disabled. Therefore, use this command carefully
- **dc_shell>set_false_path –from FF_A/CK –to FF_D/D**



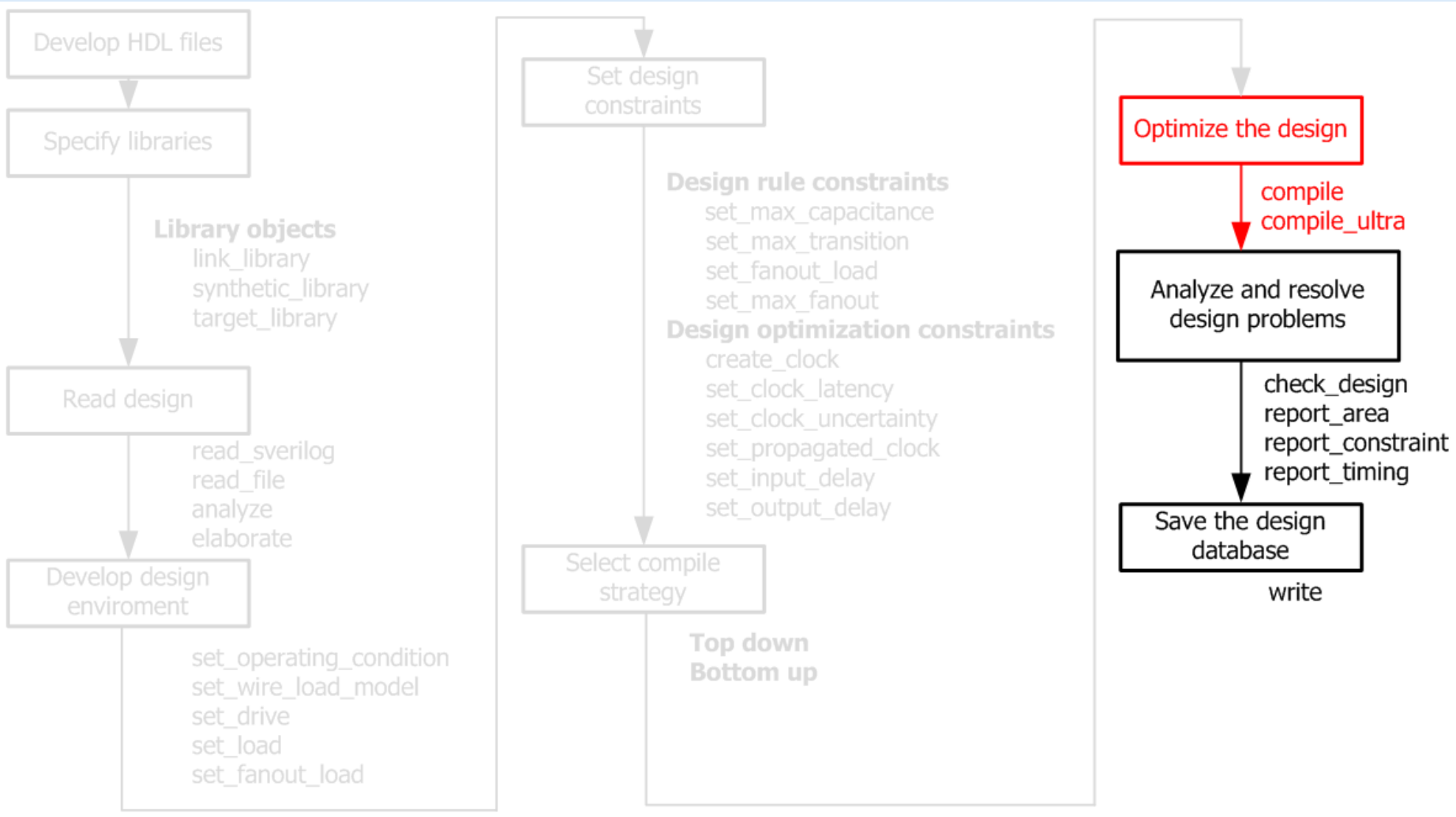
Basic Synthesis Flow



Select Compile Strategy

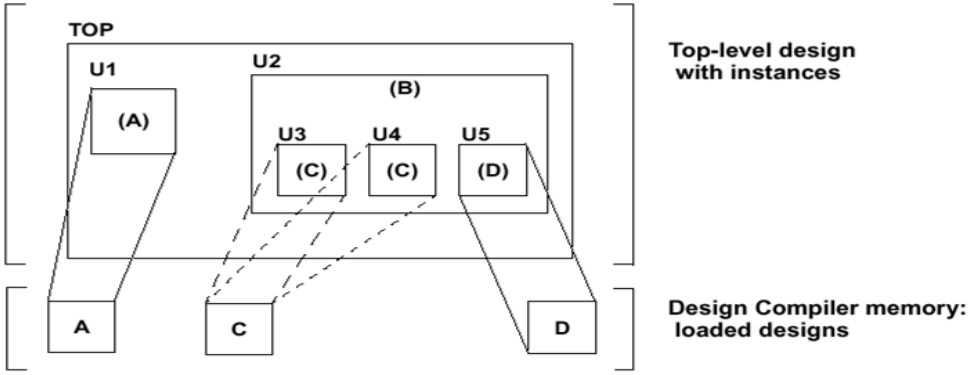
- **You can use various strategies to compile your hierarchical design.**
- **The basic strategies are**
  - Top-down compile, in which the top-level design and all its subdesigns are compiled together.
  - Bottom-up compile, in which the individual subdesigns are compiled separately, starting from the bottom of the hierarchy and proceeding up through the levels of the hierarchy until the top-level design is compiled.
  - Mixed compile, in which the top-down or bottom-up strategy, whichever is most appropriate, is applied to the individual subdesigns.

Basic Synthesis Flow



Multiple Instances of a Design Reference

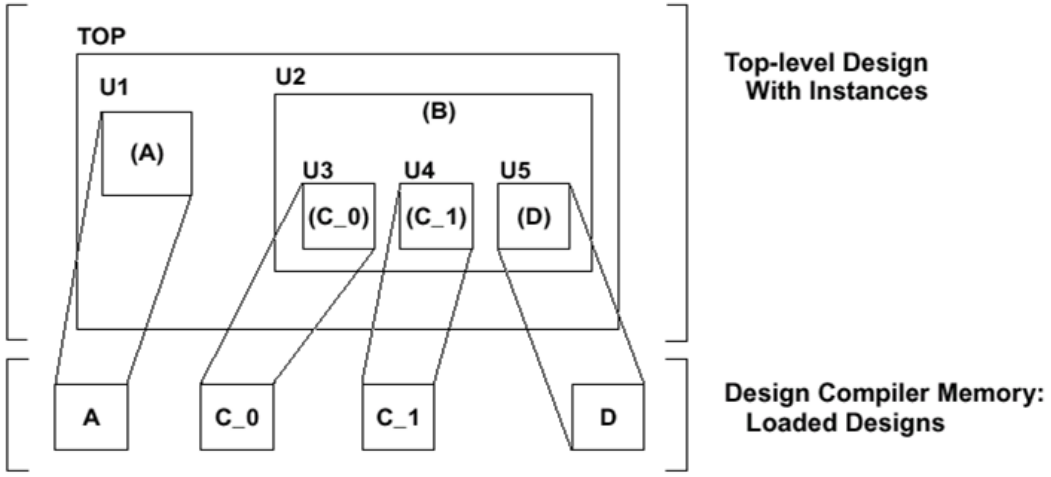
- **Multiple instances of a design reference**
  - In a hierarchical design, subdesigns are often referenced by more than one cell instance, that is, multiple references of the design can occur.
  - Ex: design C is referenced twice (U2/U3 and U2/U4).



- **Use any of the following methods resolve multiple instances before running the compile command**
  - The uniquify method
  - The compile-once-dont-touch method
  - The ungroup method

Uniquify Method

- **Uniquify (default)**
  - The command is to duplicate and rename the multiple referenced design so that each instance references a unique design.
    - Requires more memory
    - Takes longer to compile
  - **dc_shell> uniquify**
    dc_shell> compile



Uniquify Method (cont.)

✔ Example:

```
module ch6_ex(input clk, INF.DESIGN inf);

adder_1b add1(.clk(clk),.rst_n(inf.rst_n),.x1(inf.x1),.x2(inf.x2),.Sum(inf.Sum1));
adder_1b add2(.clk(clk),.rst_n(inf.rst_n),.x1(inf.x1),.x2(inf.x2),.Sum(inf.Sum2));
adder_1b add3(.clk(clk),.rst_n(inf.rst_n),.x1(inf.x1),.x2(inf.x2),.Sum(inf.Sum3));

endmodule :ch6_ex

module adder_1b ( clk, rst_n, x1, x2, Sum );
  output [1:0] Sum;
  input clk, rst_n, x1, x2;
  wire   N1, N2;

  DFFTRX1 Sum_reg_1 ( .D(N2), .RN(rst_n), .CK(clk), .Q(Sum[1]) );
  DFFTRX1 Sum_reg_0 ( .D(N1), .RN(rst_n), .CK(clk), .Q(Sum[0]) );
  XOR2X2 U3 ( .A(x2), .B(x1), .Y(N1) );
  AND2X2 U4 ( .A(x1), .B(x2), .Y(N2) );
endmodule

//module adder_1b(
//  input clk,
//  input logic rst_n,
//  input logic x1,
//  input logic x2,
//  output logic [1:0] Sum
//);
//
//always_ff@(posedge clk)
//begin
//  if(!rst_n)
//    Sum  <= 0;
//  else
//    Sum <= x1+x2;
//end
//
//endmodule :adder_1b
```

script(.tcl)

```
#============================
#  Optimization
#============================
uniquify
#set_dont_touch {add1 add2}
#ungroup {add1 add2}
compile
```

```
module ch6_ex ( clk, inf_rst_n, inf_x1, inf_x2, inf_Sum1, inf_Sum2, inf_Sum3
 );
  output [1:0] inf_Sum1;
  output [1:0] inf_Sum2;
  output [1:0] inf_Sum3;
  input clk, inf_rst_n, inf_x1, inf_x2;

  adder_1b_2 add1 ( .clk(clk), .rst_n(inf_rst_n), .x1(inf_x1), .x2(inf_x2),
      .Sum(inf_Sum1) );
  adder_1b_1 add2 ( .clk(clk), .rst_n(inf_rst_n), .x1(inf_x1), .x2(inf_x2),
      .Sum(inf_Sum2) );
  adder_1b_0 add3 ( .clk(clk), .rst_n(inf_rst_n), .x1(inf_x1), .x2(inf_x2),
      .Sum(inf_Sum3) );
endmodule
```

```
module adder_1b_1 ( clk, rst_n, x1, x2, Sum );
  output [1:0] Sum;
  input clk, rst_n, x1, x2;
  wire   N2, N1;

  DFFTRX1 Sum_reg_1 ( .D(N2), .RN(rst_n), .CK(clk), .Q(Sum[1]) );
  DFFTRX1 Sum_reg_0 ( .D(N1), .RN(rst_n), .CK(clk), .Q(Sum[0]) );
  XOR2X1 U1 ( .A(x2), .B(x1), .Y(N1) );
  AND2X1 U2 ( .A(x1), .B(x2), .Y(N2) );
endmodule
```

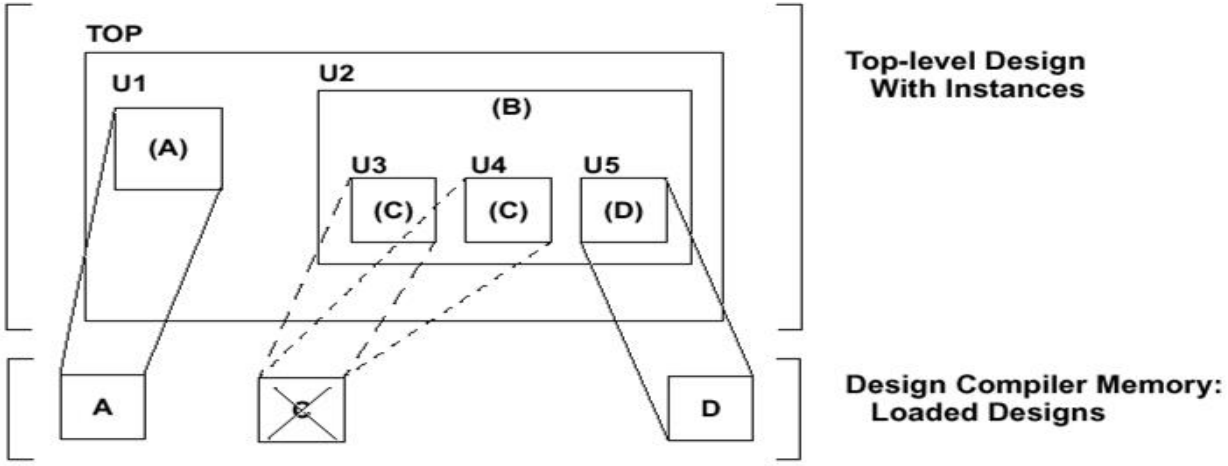
```
module adder_1b_0 ( clk, rst_n, x1, x2, Sum );
  output [1:0] Sum;
  input clk, rst_n, x1, x2;
  wire   N2, N1;

  DFFTRX1 Sum_reg_1 ( .D(N2), .RN(rst_n), .CK(clk), .Q(Sum[1]) );
  DFFTRX1 Sum_reg_0 ( .D(N1), .RN(rst_n), .CK(clk), .Q(Sum[0]) );
  XOR2X1 U1 ( .A(x2), .B(x1), .Y(N1) );
  AND2X1 U2 ( .A(x1), .B(x2), .Y(N2) );
endmodule
```

```
module adder_1b_2 ( clk, rst_n, x1, x2, Sum );
  output [1:0] Sum;
  input clk, rst_n, x1, x2;
  wire   N2, N1;

  DFFTRX1 Sum_reg_1 ( .D(N2), .RN(rst_n), .CK(clk), .Q(Sum[1]) );
  DFFTRX1 Sum_reg_0 ( .D(N1), .RN(rst_n), .CK(clk), .Q(Sum[0]) );
  XOR2X1 U1 ( .A(x2), .B(x1), .Y(N1) );
  AND2X1 U2 ( .A(x1), .B(x2), .Y(N2) );
endmodule
```

RTL(.v) → **Netlist(.v)**

Compile-Once-Don't-Touch Method

- **set_dont_touch**
  - It places the dont_touch attribute on cells, nets, references, and designs in the current design to prevent these objects from being modified or replaced during optimization.
    - Compiles the reference design once
    - Requires less memory and less time than the uniquify method
  - **dc_shell> set_dont_touch {U2/U3 U2/U4}**



Compile-Once-Don't-Touch Method(cont.)

- Example: {add1 add2} are compiled by using the environment of one of its instances. In this case, no copies of the original sub-design are loaded into memory when running this command sequence.

```
module ch6_ex(input clk, INF.DESIGN inf);

adder_1b add1(.clk(clk),.rst_n(inf.rst_n),.x1(inf.x1),.x2(inf.x2),.Sum(inf.Sum1));
adder_1b add2(.clk(clk),.rst_n(inf.rst_n),.x1(inf.x1),.x2(inf.x2),.Sum(inf.Sum2));
adder_1b add3(.clk(clk),.rst_n(inf.rst_n),.x1(inf.x1),.x2(inf.x2),.Sum(inf.Sum3));

endmodule :ch6_ex

module adder_1b ( clk, rst_n, x1, x2, Sum );
  output [1:0] Sum;
  input clk, rst_n, x1, x2;
  wire   N1, N2;

  DFFTRX1 Sum_reg_1 ( .D(N2), .RN(rst_n), .CK(clk), .Q(Sum[1]) );
  DFFTRX1 Sum_reg_0 ( .D(N1), .RN(rst_n), .CK(clk), .Q(Sum[0]) );
  XOR2X2 U3 ( .A(x2), .B(x1), .Y(N1) );
  AND2X2 U4 ( .A(x1), .B(x2), .Y(N2) );
endmodule

//module adder_1b(
//  input clk,
//  input logic rst_n,
//  input logic x1,
//  input logic x2,
//  output logic [1:0] Sum
//);
//
//always_ff@(posedge clk)
//begin
//  if(!rst_n)
//    Sum  <= 0;
//  else
//    Sum <= x1+x2;
//end
//
//endmodule :adder_1b
```

script(.tcl)

```
#=========================
#  Optimization
#=========================
#uniquify
set dont_touch {add1 add2}
#ungroup {add1 add2}
compile
```

```
module ch6_ex ( clk, inf_rst_n, inf_x1, inf_x2, inf_Sum1, inf_Sum2, inf_Sum3
 );
  output [1:0] inf_Sum1;
  output [1:0] inf_Sum2;
  output [1:0] inf_Sum3;
  input clk, inf_rst_n, inf_x1, inf_x2;

  adder_1b add1 ( .clk(clk), .rst_n(inf_rst_n), .x1(inf_x1), .x2(inf_x2),
        .Sum(inf_Sum1) );
  adder_1b add2 ( .clk(clk), .rst_n(inf_rst_n), .x1(inf_x1), .x2(inf_x2),
        .Sum(inf_Sum2) );
  adder_1b_0 add3 ( .clk(clk), .rst_n(inf_rst_n), .x1(inf_x1), .x2(inf_x2),
        .Sum(inf_Sum3) );
endmodule
```

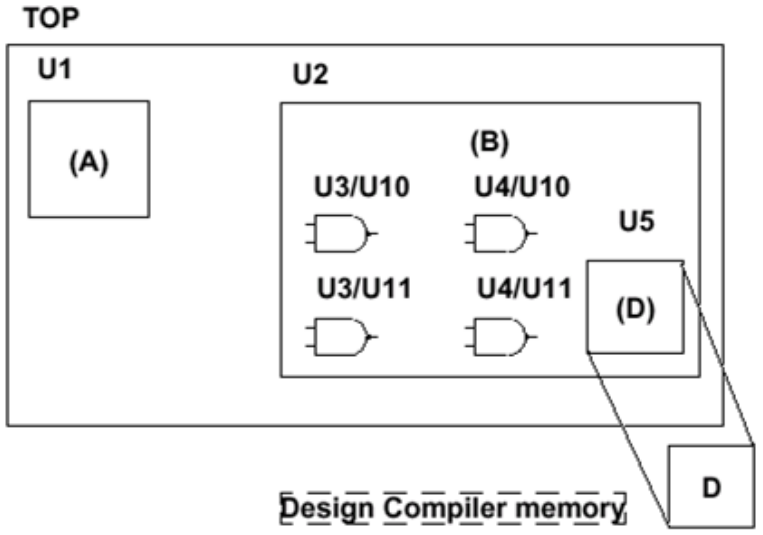
```
module adder_1b_0 ( clk, rst_n, x1, x2, Sum );
  output [1:0] Sum;
  input clk, rst_n, x1, x2;
  wire   N2, N1;

  DFFTRX1 Sum_reg_1 ( .D(N2), .RN(rst_n), .CK(clk), .Q(Sum[1]) );
  DFFTRX1 Sum_reg_0 ( .D(N1), .RN(rst_n), .CK(clk), .Q(Sum[0]) );
  XOR2X1 U1 ( .A(x2), .B(x1), .Y(N1) );
  AND2X1 U2 ( .A(x1), .B(x2), .Y(N2) );
endmodule
```

```
module adder_1b ( clk, rst_n, x1, x2, Sum );
  output [1:0] Sum;
  input clk, rst_n, x1, x2;
  wire   N2, N1;

  DFFTRX1 Sum_reg_1 ( .D(N2), .RN(rst_n), .CK(clk), .Q(Sum[1]) );
  DFFTRX1 Sum_reg_0 ( .D(N1), .RN(rst_n), .CK(clk), .Q(Sum[0]) );
  XOR2X2 U3 ( .A(x2), .B(x1), .Y(N1) );
  AND2X2 U4 ( .A(x1), .B(x2), .Y(N2) );
endmodule
```

RTL(.v) → **Netlist(.v)**

Ungroup Method

- **Ungroup**
  - use the ungroup command to ungroup one or more designs before optimization
    - Requires more memory and takes longer to compile than the compile-once-don't-touch method
    - Provides the best synthesis results
    - May increase the difficulty for ECO(Engineering change order).
  - **dc_shell> current_design B**
    dc_shell> ungroup {U3 U4}
    dc_shell> current_design top
    dc_shell> compile



Ungroup Method(cont.)

- Example: the following command sequence uses the ungroup method to resolve the multiple instances of {add1 add2} except for add3.

```
module ch6_ex(input clk, INF.DESIGN inf);

adder_1b add1(.clk(clk),.rst_n(inf.rst_n),.x1(inf.x1),.x2(inf.x2),.Sum(inf.Sum1));
adder_1b add2(.clk(clk),.rst_n(inf.rst_n),.x1(inf.x1),.x2(inf.x2),.Sum(inf.Sum2));
adder_1b add3(.clk(clk),.rst_n(inf.rst_n),.x1(inf.x1),.x2(inf.x2),.Sum(inf.Sum3));

endmodule :ch6_ex

module adder_1b ( clk, rst_n, x1, x2, Sum );
  output [1:0] Sum;
  input clk, rst_n, x1, x2;
  wire   N1, N2;

  DFFTRX1 Sum_reg_1 ( .D(N2), .RN(rst_n), .CK(clk), .Q(Sum[1]) );
  DFFTRX1 Sum_reg_0 ( .D(N1), .RN(rst_n), .CK(clk), .Q(Sum[0]) );
  XOR2X2 U3 ( .A(x2), .B(x1), .Y(N1) );
  AND2X2 U4 ( .A(x1), .B(x2), .Y(N2) );
endmodule

//module adder_1b(
//  input clk,
//  input logic rst_n,
//  input logic x1,
//  input logic x2,
//  output logic [1:0] Sum
//);
//
//always_ff@(posedge clk)
//begin
//  if(!rst_n)
//    Sum  <= 0;
//  else
//    Sum <= x1+x2;
//end
//
//endmodule :adder_1b
```

script(.tcl)

```
#=============================
#  Optimization
#=============================
#uniquify
#set_dont_touch {add1 add2}
ungroup {add1 add2}
compile
```

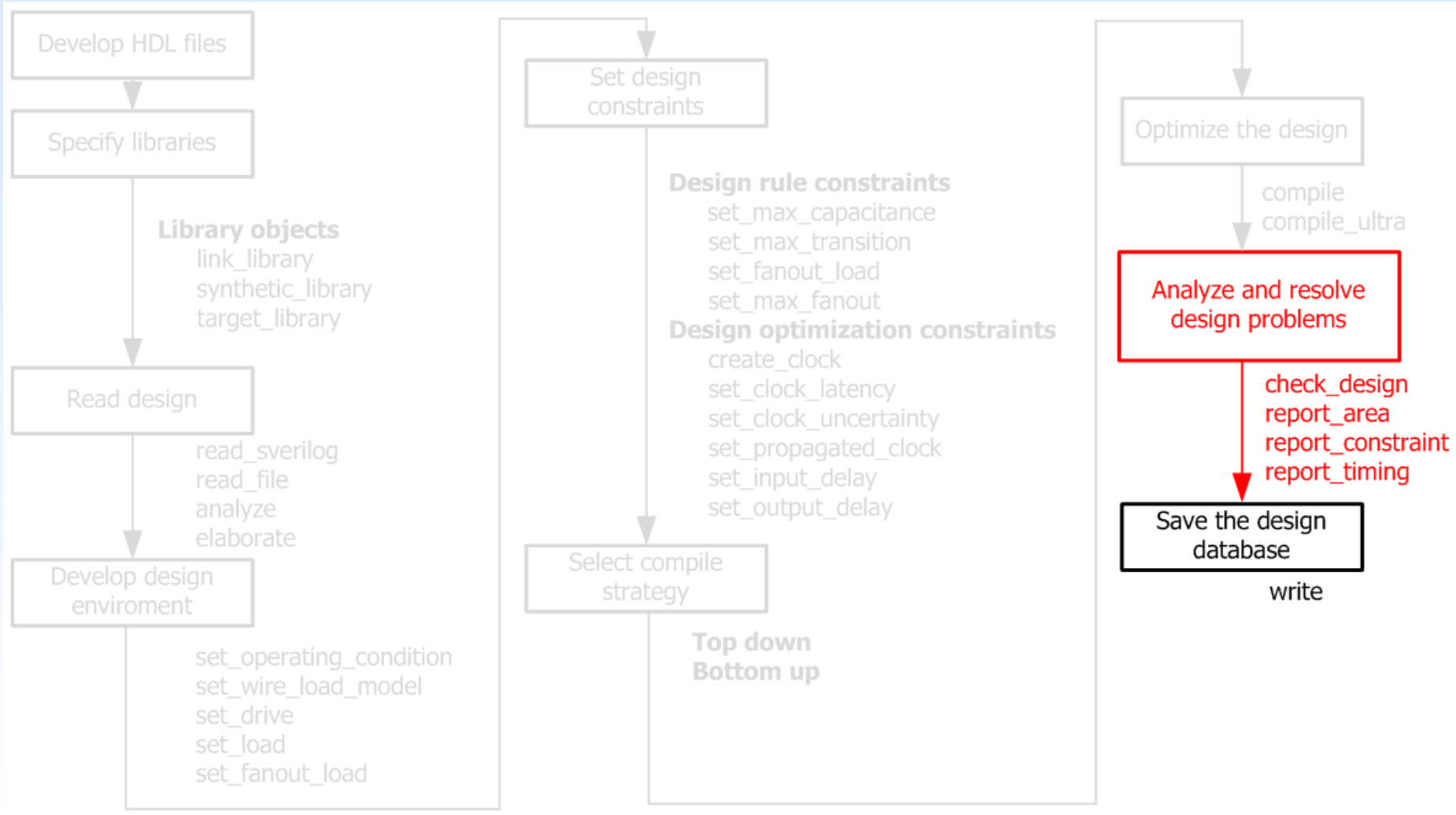
```
module ch6_ex ( clk, inf_rst_n, inf_x1, inf_x2, inf_Sum1, inf_Sum2, inf_Sum3
 );
  output [1:0] inf_Sum1;
  output [1:0] inf_Sum2;
  output [1:0] inf_Sum3;
  input clk, inf_rst_n, inf_x1, inf_x2;
  wire   add2_N1, add2_N2;

  adder_1b add3 ( .clk(clk), .rst_n(inf_rst_n), .x1(inf_x1), .x2(inf_x2),
       .Sum(inf_Sum3) );
  DFFTRX1 add1_Sum_reg_0 ( .D(add2_N1), .RN(inf_rst_n), .CK(clk), .Q(
       inf_Sum1[0]) );
  DFFTRX1 add1_Sum_reg_1 ( .D(add2_N2), .RN(inf_rst_n), .CK(clk), .Q(
       inf_Sum1[1]) );
  DFFTRX1 add2_Sum_reg_0 ( .D(add2_N1), .RN(inf_rst_n), .CK(clk), .Q(
       inf_Sum2[0]) );
  DFFTRX1 add2_Sum_reg_1 ( .D(add2_N2), .RN(inf_rst_n), .CK(clk), .Q(
       inf_Sum2[1]) );
  XOR2X1 U3 ( .A(inf_x1), .B(inf_x2), .Y(add2_N1) );
  AND2X1 U4 ( .A(inf_x2), .B(inf_x1), .Y(add2_N2) );
endmodule

module adder_1b ( clk, rst_n, x1, x2, Sum );
  output [1:0] Sum;
  input clk, rst_n, x1, x2;
  wire   N2, N1;

  DFFTRX1 Sum_reg_1 ( .D(N2), .RN(rst_n), .CK(clk), .Q(Sum[1]) );
  DFFTRX1 Sum_reg_0 ( .D(N1), .RN(rst_n), .CK(clk), .Q(Sum[0]) );
  XOR2X1 U1 ( .A(x2), .B(x1), .Y(N1) );
  AND2X1 U2 ( .A(x1), .B(x2), .Y(N2) );
endmodule
```

RTL(.v) → **Netlist(.v)**

Compile

- **Default synthesis algorithm**
  - **dc_shell> compile**
- **Advanced synthesis algorithm**
  - **dc_shell> compile_ultra**
    - Automatically ungroups logical hierarchies
    - High-performance design
    - Maximum performance
    - Minimum area
    - Data path
- More information
  - Refer to "Design Compiler Optimization Reference Manual"

Basic Synthesis Flow



Report Analysis

Constraint report

- Syntax : report_constraint [-all_violators] [-verbose]
  - -all_violators : Displays a summary of all of the optimization and design-rule constraints with violations in the current design.
  - -verbose : Indicates to show more detail about constraint calculations

Area report

- dc_shell-t> report_area

```
……..
Constraint                                  Cost
---------------------------------------------------

max_transition                          0.00 (MET)
max_capacitance                         0.00 (MET)
max_delay/setup                         0.00 (MET)
critical_range                          0.00 (MET)
```

```
****************************************
Report : area
Design : DAG
Version: 2003.06
Date   : Sun Oct 10 15:59:26 2004
****************************************

Library(s) Used:

    slow (File: /RAID/Manager/lib.18/SynopsysDC/slow.db)

Number of ports:            40
Number of nets:            220
Number of cells:           160
Number of references:       21

Combinational area:      3775.465576
Noncombinational area:   2471.515869
Net Interconnect area:      2.482625

Total cell area:         6246.979492

Total area:              6249.463867
```

Report Analysis (cont.)

report_timing

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : DAG
Version: 2003.06
Date   : Sun Oct 10 16:06:19 2004
****************************************

Operating Conditions: slow   Library: slow
Wire Load Model Mode: segmented

  Startpoint: COUNT_LOAD_reg[1]
              (rising edge-triggered flip-flop clocked by CLK)
  Endpoint: ADDR_reg[0]
              (rising edge-triggered flip-flop clocked by CLK)
  Path Group: CLK
  Path Type: max
```

Maximum delay analysis

Point	Incr	Path	
clock CLK (rise edge)	0.00	0.00	
clock network delay (ideal)	0.00	0.00	
COUNT_LOAD_reg[1]/CK (DFFX1)		0.00	0.00 r
COUNT_LOAD_reg[1]/QN (DFFX1)		0.56	0.56 r
U134/Y (NAND3X1)	0.23	0.79 f	
U160/Y (INVX2)	0.90	1.69 r	
U137/Y (MX2X2)	0.52	2.21 f	
.....			
.....			
U226/Y (OR4X4)	0.57	8.24 f	
U206/Y (NOR2X4)	0.97	9.21 r	
U149/Y (AOI22X1)	0.27	9.49 f	
U148/Y (INVX2)	0.16	9.64 r	
ADDR_reg[0]/D (EDFFX1)	0.00	9.64 r	
data arrival time	9.64		
clock CLK (rise edge)	10.00	10.00	
clock network delay (ideal)	0.00	10.00	
ADDR_reg[0]/CK (EDFFX1)	0.00	10.00 r	
library setup time	-0.29	9.71	
data required time	9.71		
data required time	9.71		
data arrival time	-9.64		
slack (MET)	0.07		

Fix Glitch Suppression

- **When you meet glitch suppression at 03_GATE_SIM**

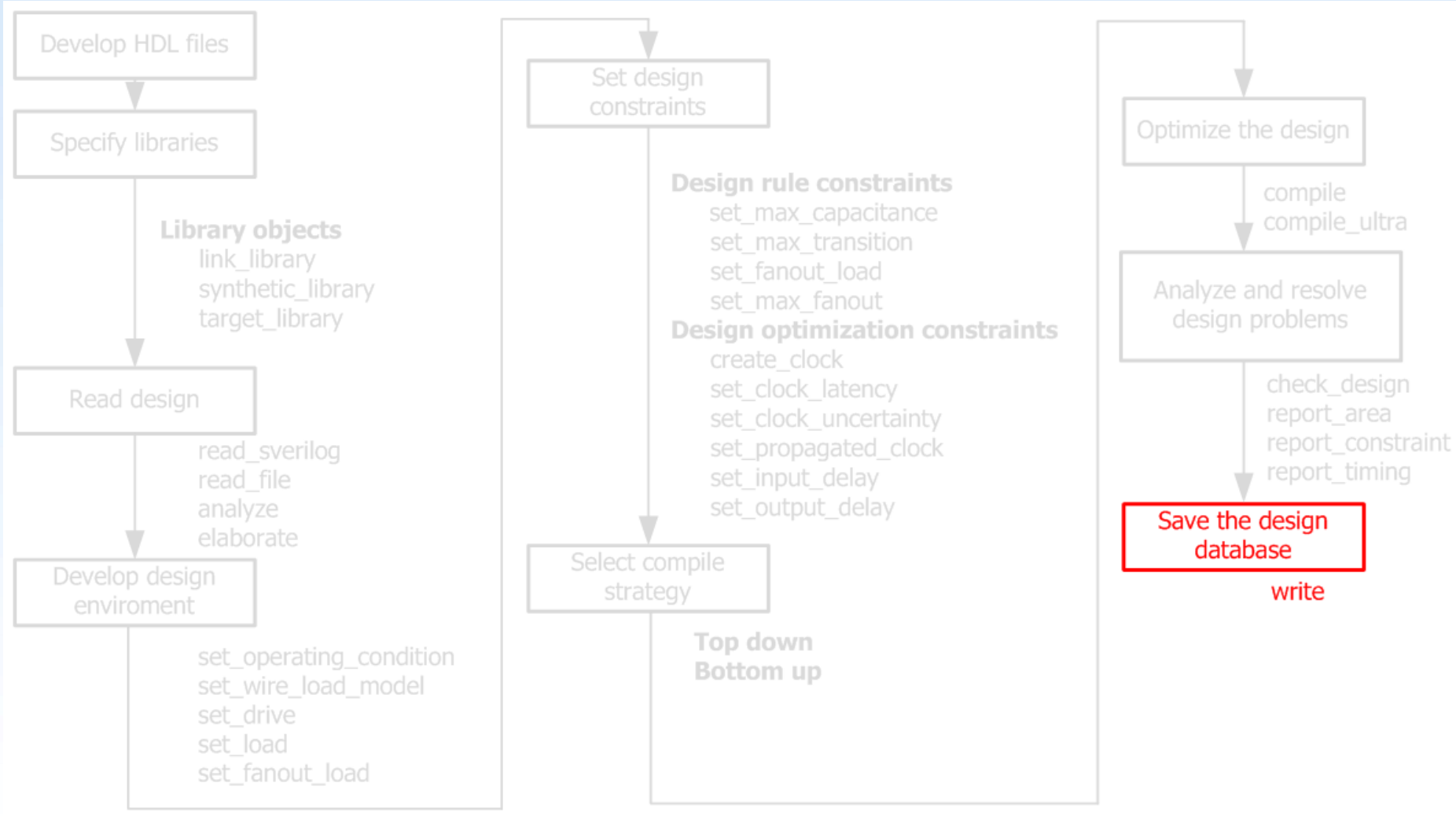
```
Warning!  Glitch suppression
          Scheduled event for delayed signal of net "dD" at time 148218322 PS was canceled!
           File: /misc/RAID2/COURSE/icsum/icsumta01/umc018/Verilog/umc18_neg.v, line = 10043
          Scope: TESTBED.U_CORE.y_reg1_reg_7_
           Time: 148217861 PS
```

  - Always occur in "dD" umc18_neg.v, line = 10043
  - Add "-nontcglitch" in 03_GATE ./01_run

- **To specify cells in the target library to be excluded during optimization**
  - **dc_shell> set_dont_use [get_lib_cells "slow/JKFF*"]**

```
set_dont_use slow/JKFF*
```

Basic Synthesis Flow



Save Design

- **Change naming rule script**
  - Make all net and port names conform to the naming conventions for layout tool
  - Execute the script after compile your design

```
set bus_inference_style “%s\[%d\]”
set bus_naming_style “%s\[%d\]”
set hdlout_internal_busses  true
change_names –hierarchy –rule verilog
define_name_rules name_rule –allowed “a-z A-Z 0-9 _” –max_length 255 –type cell
define_name_rules name_rule –allowed “a-z A-Z 0-9 _[ ]” –max_length 255 –type net
define_name_rules name_rule –map {{“\\*cell\\*” “cell”}}
define_name_rules name_rule _case_insensitive  # if you want to run spice after APR
change_names –hierarchy –rules name_rule
```

Save Design

Save design

- You use the write command to save the synthesized designs. Remember that Design Compiler does not automatically save designs before exiting.

Save a gate level Verilog file

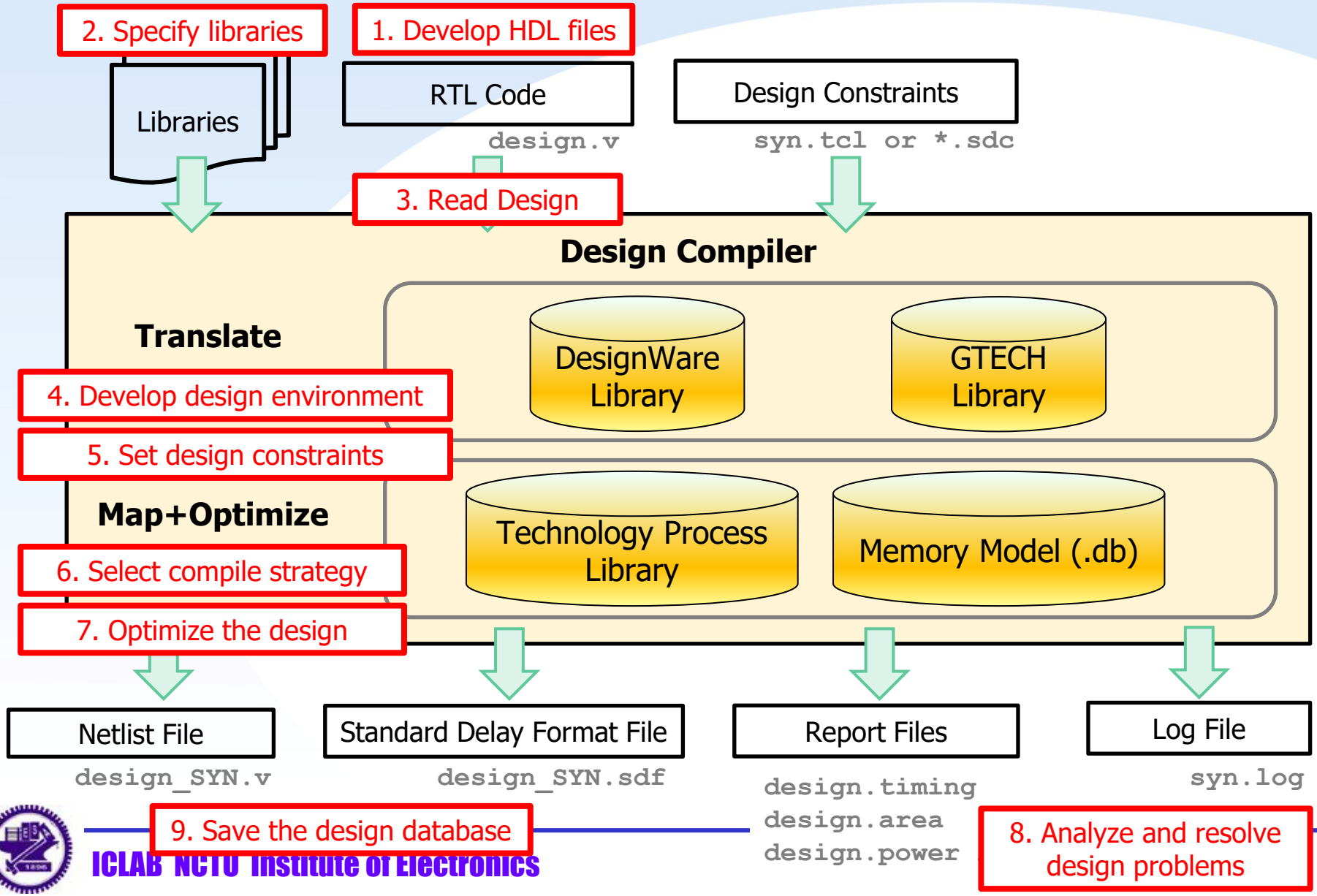
- Syntax : write –f verilog –o ***file_name.v*** –hierarchy
  - -f : format
  - -o : output file name
  - -hierarchy : Indicates to write all designs in the hierarchy

```
write -format verilog -output Netlist/$DESIGN\_SYN.v -hierarchy
```

Gate-level Simulation

- **Post synthesis timing simulation**
  - The post synthesis design must be simulated with estimated delays from synthesis. A SDF(standard delay format) can be generated from design compiler for this purpose. Use the following command to generate the SDF file.
  - Syntax : **write_sdf -version sdf_version file_name**
    - -version sdf_version : Selects which SDF version to use. Supported SDF versions are 1.0, 2.1 or 3.0. SDF 2.1 is the default.
    - file_name : Specifies the name of the SDF file to write.
  - **dc_shell> write_sdf –version 3.0 CHIP.sdf**
- **Modify your test file (TESTBED.v)**
  - $sdf_annotate("the_SDF_file_name¨, the_instance_name);
  - ex : $sdf_annotate("CHIP.sdf", I_DAG);

Data Flow in Design Compiler



Syn.tcl Example

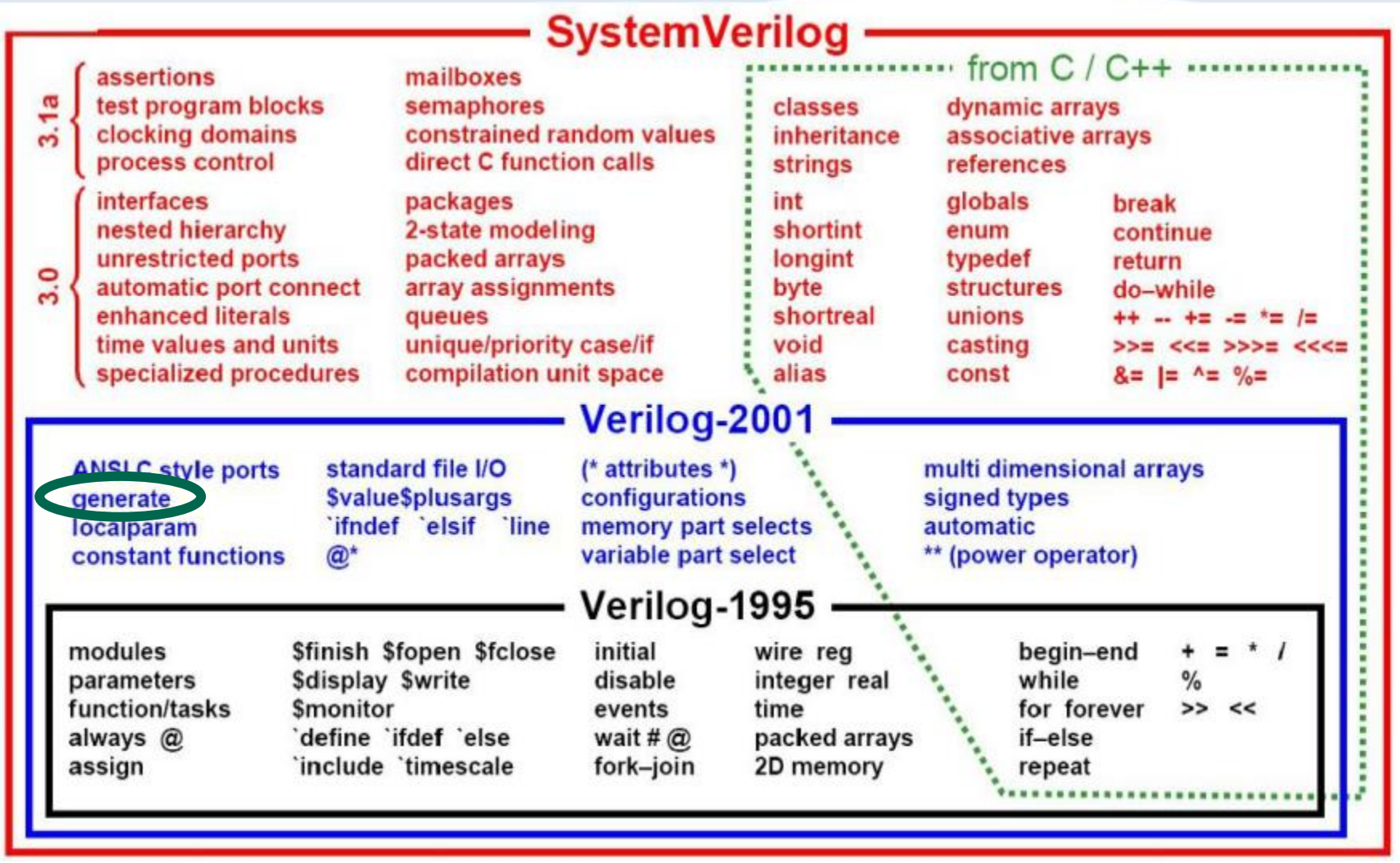
```
#================================================
#
# Synopsys Synthesis Scripts (Design Vision dctcl mode)
#
#================================================

#================================================
#  Set Libraries
#================================================
set search_path {./../01_RTL \
                 ~iclabta01/umc018/Synthesis/ \
                 /usr/synthesis/libraries/syn/}

set synthetic_library {dw_foundation.sldb}
set link_library {* dw_foundation.sldb standard.sldb slow.db}
set target_library {slow.db}

#================================================
#  Global Parameters
#================================================
set DESIGN "INV_IP_demo"
set MAX_Delay 60

#================================================
#  Read RTL Code
#================================================
analyze -f verilog $DESIGN\.v
analyze -f verilog INV_IP.v
elaborate $DESIGN
current_design $DESIGN

#================================================
#  Global Setting
#================================================
set_wire_load_mode top
#================================================
#  Set Design Constraints
#================================================
set_max_delay $MAX_Delay -from [all_inputs] -to [all_outputs]
set_load 0.05 [all_outputs]

#================================================
#  Optimization
#================================================
uniquify
set_fix_multiple_port_nets -all -buffer_constants
compile_ultra

#================================================
#  Output Reports
#================================================
report_timing > Report/$DESIGN\.timing
report_area > Report/$DESIGN\.area
report_resource > Report/$DESIGN\.resource

#================================================
#  Change Naming Rule
#================================================
set bus_inference_style "%s\[%d\]"
set bus_naming_style "%s\[%d\]"
set hdlout_internal_busses true
change_names -hierarchy -rule verilog
define_name_rules name_rule -allowed "a-z A-Z 0-9 _" -max_length 255 -type cell
define_name_rules name_rule -allowed "a-z A-Z 0-9 _[]" -max_length 255 -type net
define_name_rules name_rule -map {{"\\*cell\\*" "cell"}}
change_names -hierarchy -rules name_rule

#================================================
#  Output Results
#================================================
set verilogout_higher_designs_first true
write -format verilog -output Netlist/$DESIGN\_SYN.v -hierarchy
write_sdf -version 3.0 -context verilog -load_delay cell Netlist/$DESIGN\_SYN.sdf -significant_digits 6

#================================================
#  Finish and Quit
#================================================
report_area
report_timing
exit
```

2. 3. 4. 5. 6. 7. 8. 9.

Outline

- ✔ Section 1 Design Compiler Introduction
- ✔ Section 2 Basic Synthesis Flow
  - Develop HDL files
  - Specify libraries
  - Read design
  - Develop design environment
  - Set design constraints
  - Select compile strategy
  - Optimize the design
  - Analyze and resolve design problems
  - Save the design database
- ✔ **Section 3 Generate & For Loop**

Generate


SystemVerilog
3.1a
assertions
test program blocks
clocking domains
process control
mailboxes
semaphores
constrained random values
direct C function calls
3.0
interfaces
nested hierarchy
unrestricted ports
automatic port connect
enhanced literals
time values and units
specialized procedures
packages
2-state modeling
packed arrays
array assignments
queues
unique/priority case/if
compilation unit space
from C / C++
classes
inheritance
strings
dynamic arrays
associative arrays
references
int
shortint
longint
byte
shortreal
void
alias
globals
enum
typedef
structures
unions
casting
const
break
continue
return
do–while
++ -- += -= *= /=
>>= <<= >>>= <<<=
&= |= ^= %=
Verilog-2001
ANSI C style ports
generate
localparam
constant functions
standard file I/O
$value$plusargs
`ifndef `elsif `line
@*
(* attributes *)
configurations
memory part selects
variable part select
multi dimensional arrays
signed types
automatic
** (power operator)
Verilog-1995
modules
parameters
function/tasks
always @
assign
$finish $fopen $fclose
$display $write
$monitor
`define `ifdef `else
`include `timescale
initial
disable
events
wait # @
fork–join
wire reg
integer real
time
packed arrays
2D memory
begin–end
while
for forever
if–else
repeat
+ = * /
%
>> <<

Review : For Loop

- **For loop in Verilog**
  - Duplicate same function
  - Very useful for doing reset and iterated operation

```
reg [3:0] temp;
integer i;
always @(posedge clk) begin
  for (i = 0; i < 3 ; i = i + 1) begin: for_name
    temp[i] <= 1'b0;
  end
end
```

=

```
always @(posedge clk) begin
  temp[0] <= 1'b0;
  temp[1] <= 1'b0;
  temp[2] <= 1'b0;
end
```

Generate – generate vs regular for loop

- **How to use for loop with generate?**
  - For loop in generate : three always blocks
  - Regular for loop : one always block

```
reg [3:0] temp;
genvar i;
generate
for (i = 0; i < 3 ; i = i + 1) begin: for_name
   always @(posedge clk) begin
      temp[i] <= 1'b0;
   end
end
endgenerate
```

Generate block

```
reg [3:0] temp;
integer i;
always @(posedge clk) begin
  for (i = 0; i < 3 ; i = i + 1) begin:
    temp[i] <= 1'b0;
  end
end
```

Regular for loop

Generate

```
wire [3:0] o_data;

genvar i;
generate
for (i=0 ; i < 4; i = i+1)begin : loop_1
    wire tmp_result;
    assign tmp_result = operand1[i] | operand2[i];
    if(i == 0)begin
       assign o_data[0] = tmp_result;
    end
    else begin
       assign o_data[i] = tmp_result & loop_1[i-1].tmp_result;
    end
end
endgenerate
```

always block in for loop with genvar

If-else in generate for can only use for select circuit but not logic determination

How to use wire in for loop ?

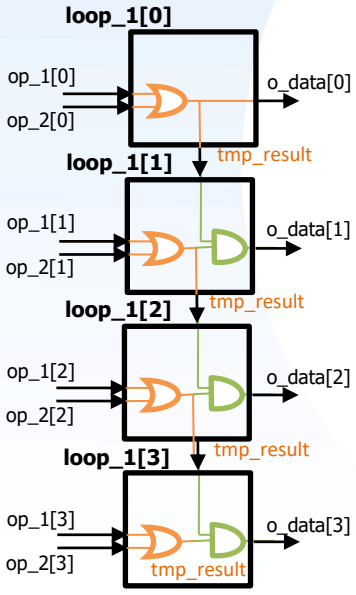
for_name[i].wire_name

Ex : loop_1[0].tmp_result

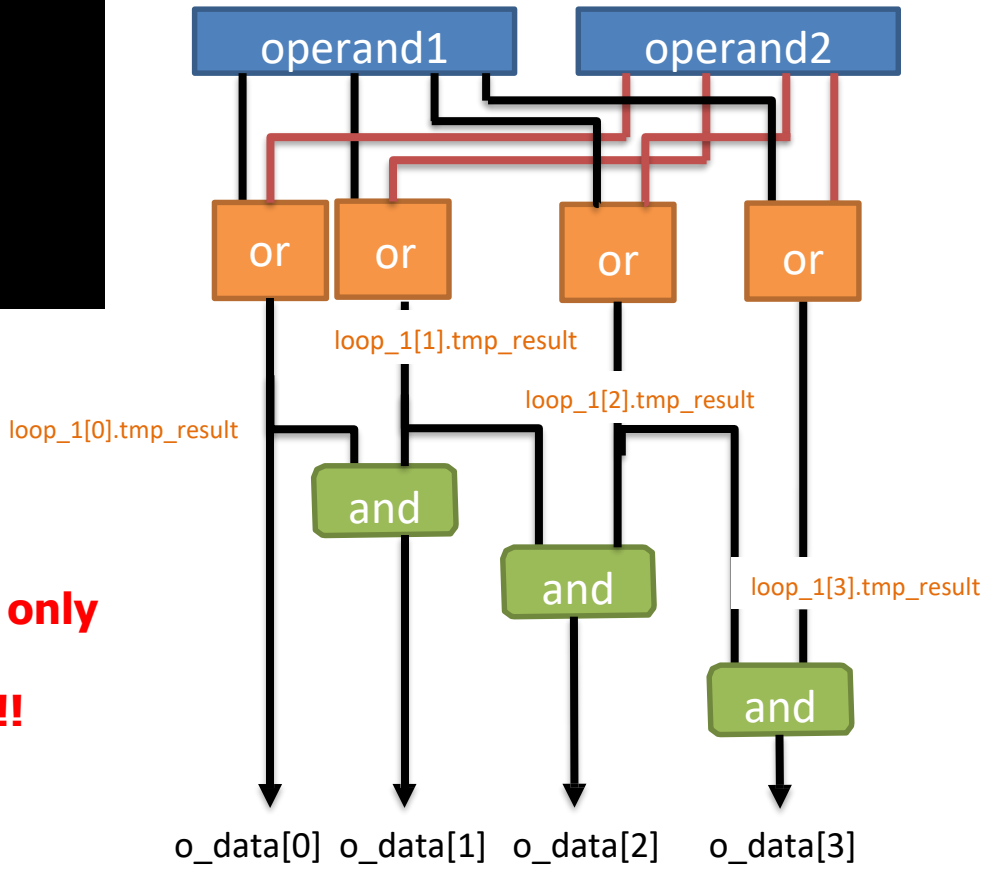
Generate

A little complicated but scalable design example

```
wire [3:0] o_data;

genvar i;
generate
for (i=0 ; i < 4; i = i+1)begin : loop_1
    wire tmp_result;
    assign tmp_result = operand1[i] | operand2[i];
    if(i == 0)begin
       assign o_data[0] = tmp_result;
    end
    else begin
       assign o_data[i] = tmp_result & loop_1[i-1].tmp_result;
    end
end
endgenerate
```



Generate for can only use increase idx but not decrease!!



Generate

- **Generate blocks are useful when change the physical structure of module via parameters.**
  - We can modify the parameter for different application

```
module top
...
adder add_8bit #(8) (.a(a0), .b(b0),
                     .sum(sum_8bit));
adder add_16bit #(16) (.a(a1), .b(b1),
                      .sum(sum_16bit));
...
endmodule
```

```
module FA
(  input          a, b, cin,
   output  reg    sum, cout  );

always @(*) begin
        {cout, sum} = a + b + cin;
end
endmodule
```

```
module adder
#(parameter LENGTH = 16)
(  input      [LENGTH-1:0]        a,
   input      [LENGTH-1:0]        b,
   output     [LENGTH:0]          sum   );
wire [LENGTH-1:0]  c ;

genvar i;
generate
for (i=0; i<LENGTH; i=i+1) begin: loop_fa
    if (i==0)
       FA u0 (.a(a[i]), .b(b[i]), .cin(1'b0), .sum(sum[i]), .cout(c[i]));
    else
       FA u1 (.a(a[i]), .b(b[i]), .cin(c[i-1]), .sum(sum[i]), .cout(c[i]));
end
endgenerate

assign sum[LENGTH] = c[LENGTH-1];

endmodule
```

Reference

- STA - Static Timing Analysis
  - http://www.ee.bgu.ac.il/~digivlsi/slides/STA_9_1.pdf (p62, p63 圖)
- Static Timing Analysis for Nanometer Designs - A Practical Approach, Bhasker, J./ Chadha, Rakesh
  - https://vlsitesting.files.wordpress.com/2017/02/ref-for-unit6.pdf
- Design compiler user guide
  - http://archive.eclass.uth.gr/eclass/modules/document/file.php/MHX303/Documentation/dcug.pdf

Revision History

Author : 2004 Wan-Chun Liao (celesta@si2lab.org)
2006revised Ming-Wei Lai (mwlai@si2lab.org)
2008revised Chien-Ying Yu (cyyu@si2lab.org)
2008revised Jia-Lung Lin (jllin@si2lab.org)
2009revised Yung-Chih Chen (ycchen@oasis.ee.nctu.edu.tw)
2010revised Jen-Wei Lee (circus@si2lab.org)
2011revised Chia-Ching Chu (ccchu@oasis.ee.nctu.edu.tw)
2013revised Rong-Jie Liu (johnny510.ee98@g2.nctu.edu.tw)
2014revised Ming-Feng Shiu (mango@si2lab.org)
2015revised De-An Chen (and0350283@gmail.com)
2016revised Chien-Tung Shih (ct.shihg@gmail.com)、Heng-Wei Hsu (hengwzx@gmail.com)
2016revised Yun-Sheng Chan (yschan.ee03g@g2.nctu.edu.tw)
2022revised Heng-Yu, Liu (nine87129.ee10@nycu.edu.tw)
2023revised Zhi-Ting, Dong (yjdzt918.ee11@nycu.edu.tw)
2023revised Shao-Hua Lian (eed0810766.eed08@nctu.edu.tw)