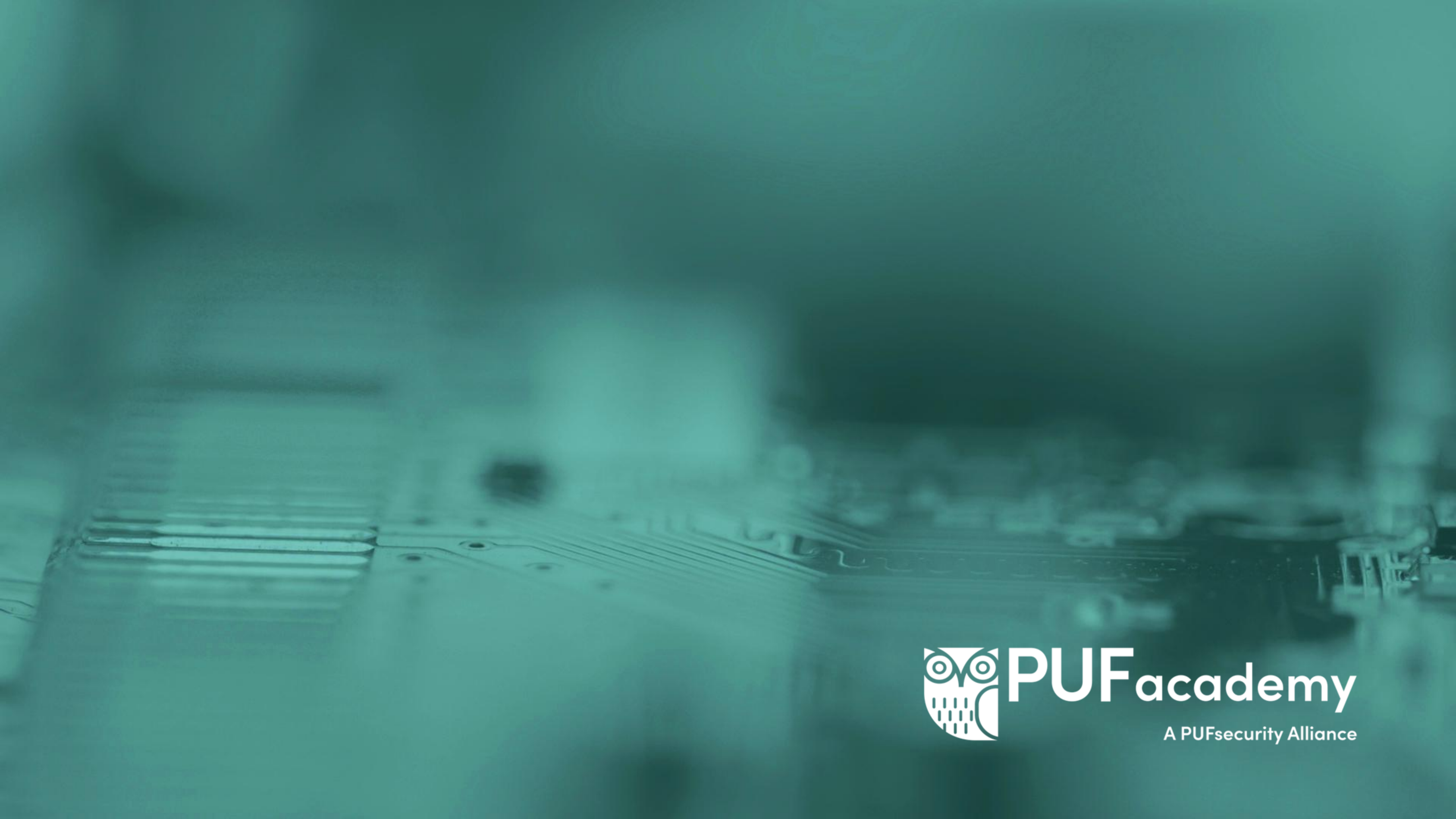


PUFacademy

A PUFsecurity Alliance

# Digital Logic Design
# - Lecture 6
# - Synthesis .

2025 Spring

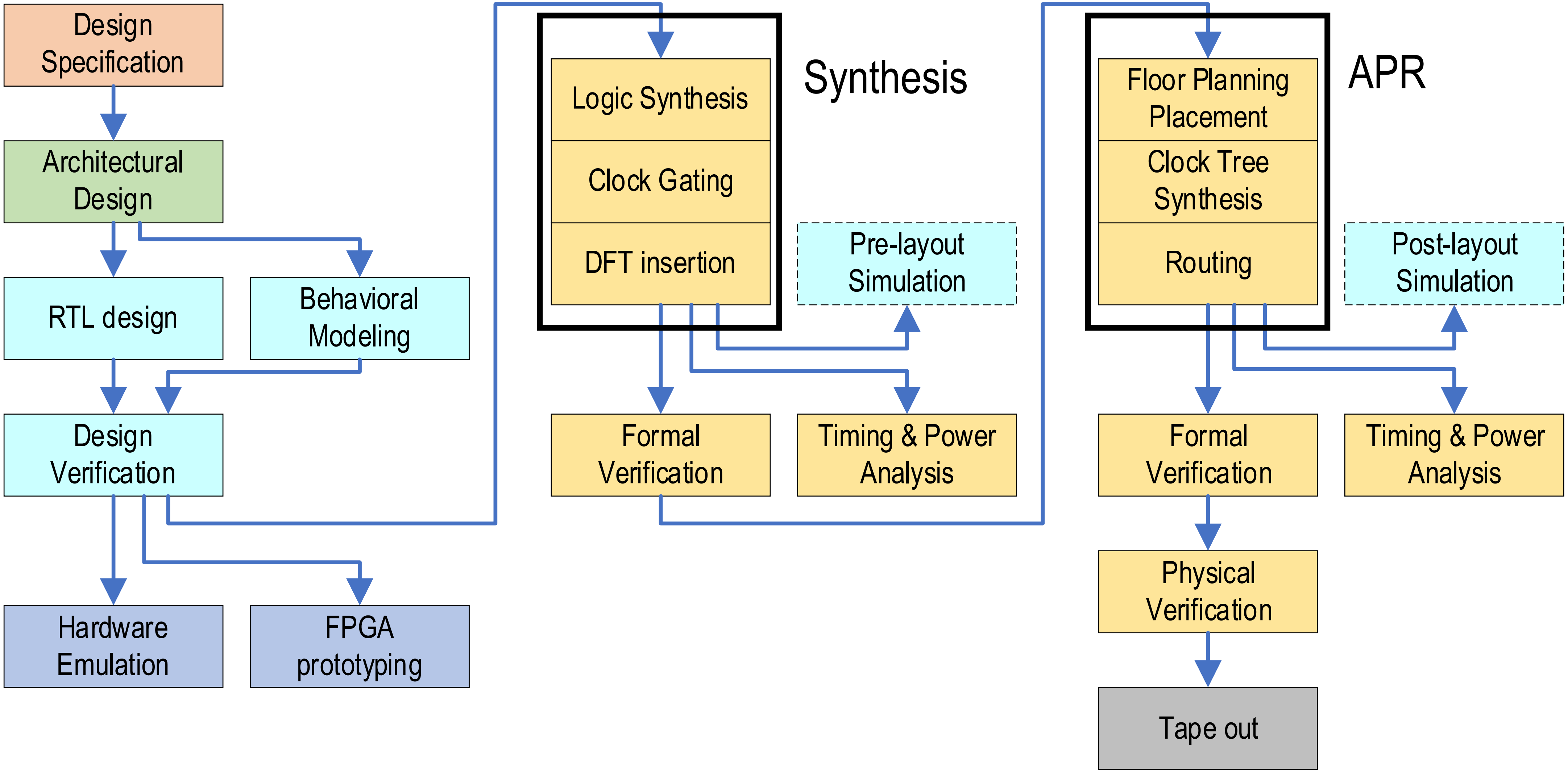
PUFacademy

A PUFsecurity Alliance

# Agenda

1. Synthesis Flow
2. Design Compiler

# Basic Design Flow.



Design Specification

Architectural Design

RTL design

Behavioral Modeling

Design Verification

Hardware Emulation

FPGA prototyping

Synthesis

Logic Synthesis

Clock Gating

DFT insertion

Pre-layout Simulation

Formal Verification

Timing & Power Analysis

APR

Floor Planning Placement

Clock Tree Synthesis

Routing

Post-layout Simulation

Formal Verification

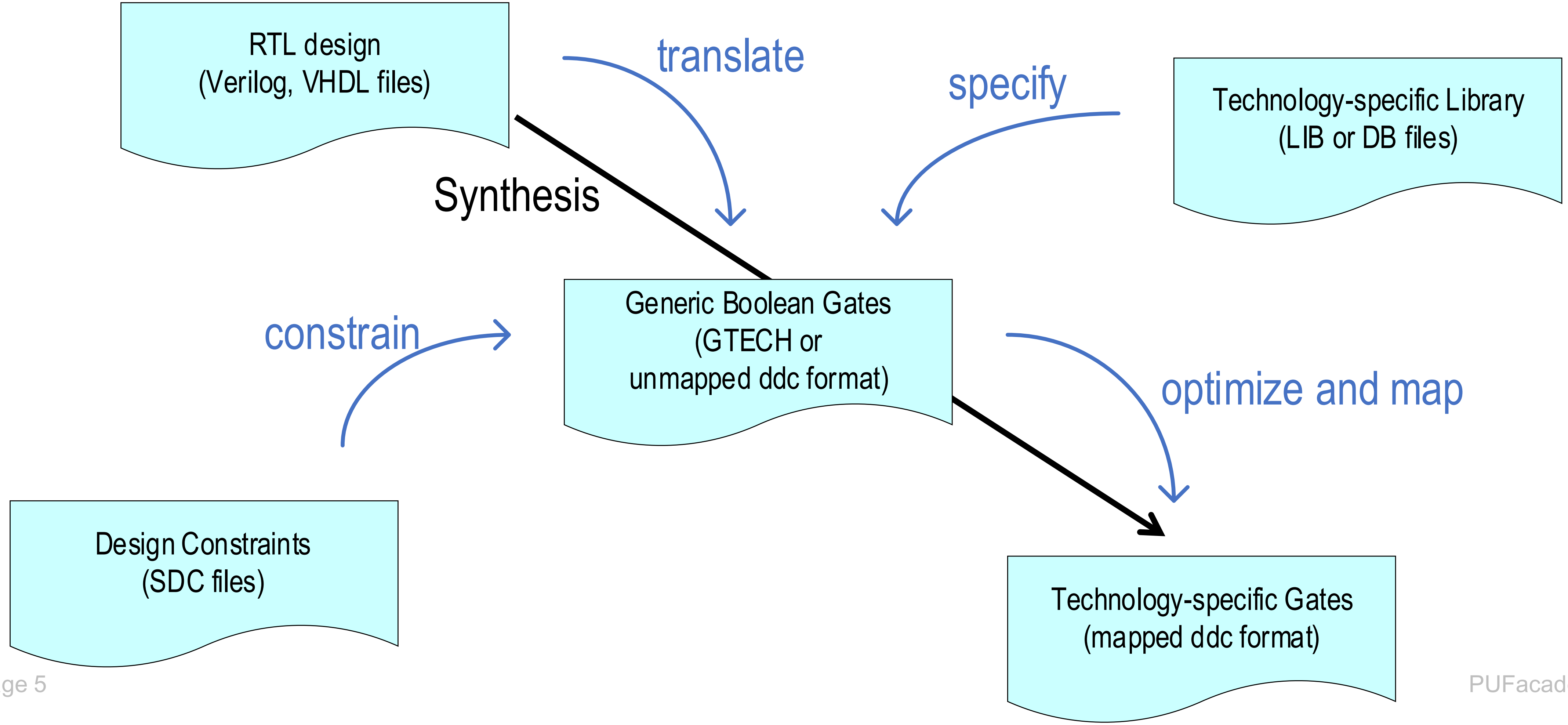
Timing & Power Analysis

Physical Verification

Tape out

# Concept of Synthesis

- Synthesis = Translation + Logic Optimization + Technology-specific Gate Mapping



RTL design
(Verilog, VHDL files)

Technology-specific Library
(LIB or DB files)

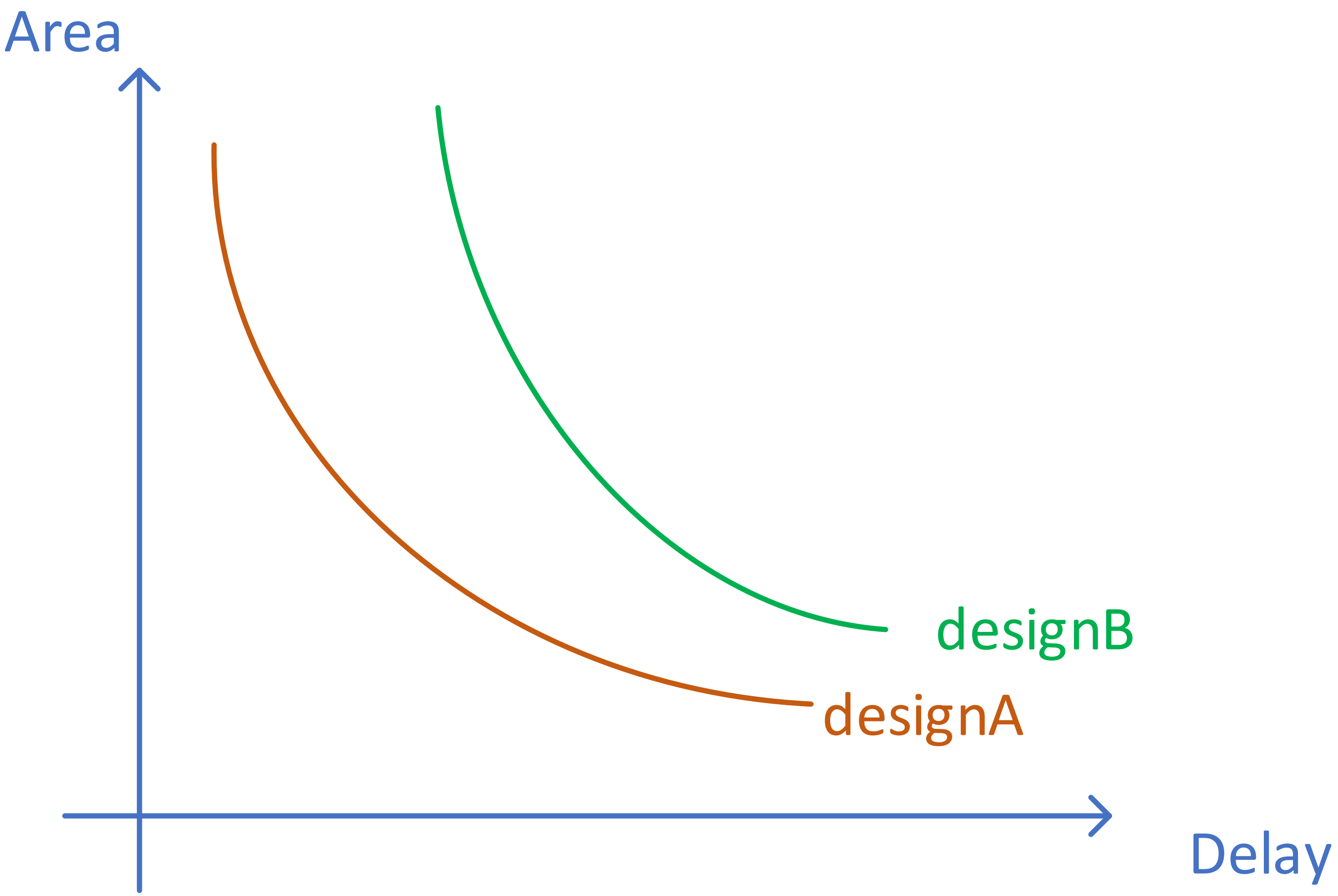
Generic Boolean Gates
(GTECH or
unmapped ddc format)

Design Constraints
(SDC files)

Technology-specific Gates
(mapped ddc format)

# Design Space Curve

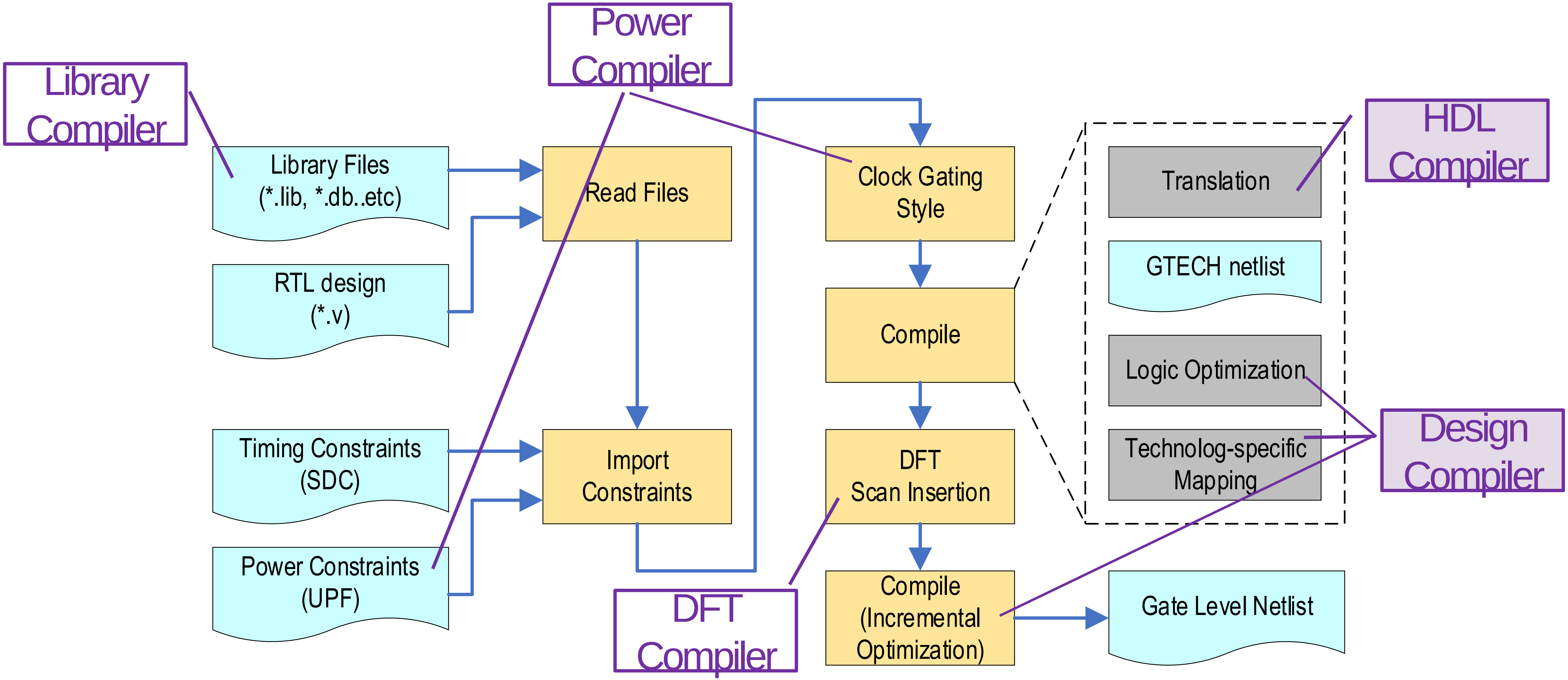
- The shape of the curve demonstrates the tradeoff between area-efficient and speed-efficient circuits



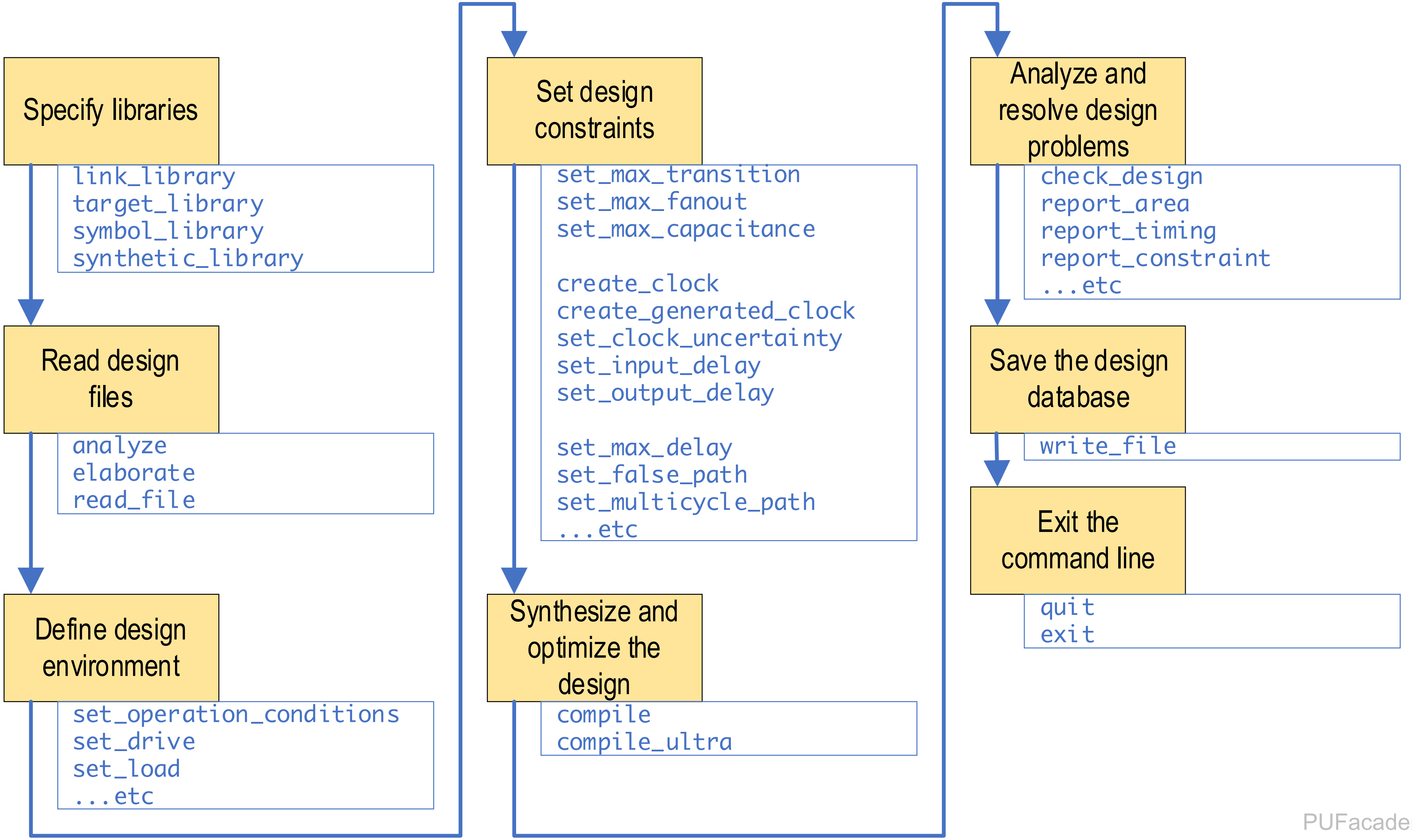
# Agenda

1. Synthesis Flow
2. Design Compiler

# Synopsys Tools for Synthesis



# Commands for Design Compiler



Specify libraries

```
link_library
target_library
symbol_library
synthetic_library
```

Read design files

```
analyze
elaborate
read_file
```

Define design environment

```
set_operation_conditions
set_drive
set_load
...etc
```

Set design constraints

```
set_max_transition
set_max_fanout
set_max_capacitance

create_clock
create_generated_clock
set_clock_uncertainty
set_input_delay
set_output_delay

set_max_delay
set_false_path
set_multicycle_path
...etc
```

Synthesize and optimize the design

```
compile
compile_ultra
```

Analyze and resolve design problems

```
check_design
report_area
report_timing
report_constraint
...etc
```

Save the design database

```
write_file
```

Exit the command line

```
quit
exit
```

# Using Design Compiler Shell

- Design Compiler Shell is based on tool command language (Tcl)
  - Design compiler shell is similar to UNIX command shells
  - Use dcnxt_shell or dc_shell commands to enter the design compiler shell

```
[xxx@wsxx syn]$ dc_shell
```

  - Use dcnxt_shell or dc_shell commands with argument –f to run the script file

```
[xxx@wsxx syn]$ dc_shell -f syn_design.scr
```

  - Enter individual commands interactively at the dcnxt_shell or dc_shell prompt

```
dc_shell> report_timing
```

  - Use source to run Tcl command script at the dcnxt_shell or dc_shell prompt

```
dc_shell> source syn_design.scr
```

# Getting Help on command line

- Use help command
  - Displays the quick help for command

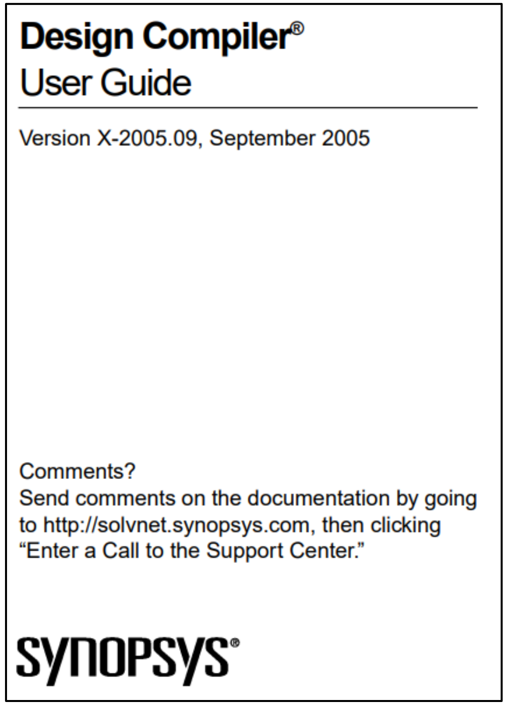
```
dc_shell> help *report*
```

- Use help argument
  - Gets the information about the arguments of a command

```
dc_shell> report_timing -help
```

- Use man command
  - Displays the man page on the command line

```
dc_shell> man report_timing
```



**Design Compiler®**
User Guide

Version X-2005.09, September 2005

Comments?
Send comments on the documentation by going to http://solvnet.synopsys.com, then clicking "Enter a Call to the Support Center."

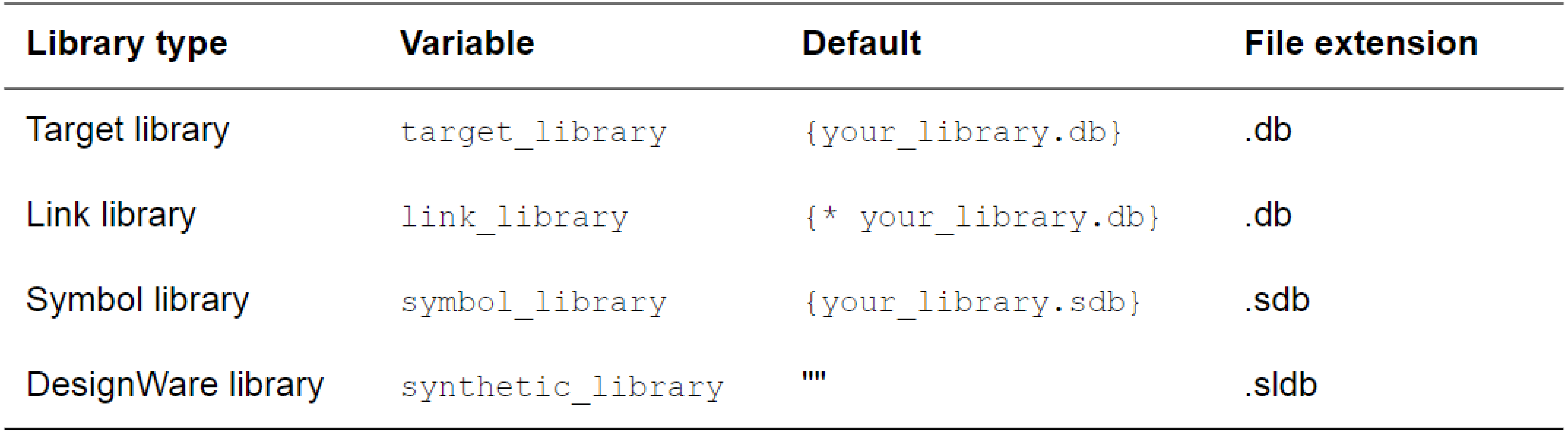
SYNOPSYS®

# Specify Libraries

# Library Requirements

- **Search Path**
  - Library search path
- **Target Libraries**
  - The technology library to which Design Compiler maps during optimization.
  - Contain the cells used to generate the netlist
  - Definitions for the operating conditions and wire load models
- **Symbol Libraries**
  - Contains the schematic symbols of the logic cells
- **DesignWare Libraries**
  - Synopsys provided reusable design components
  - Selected for the optimization during synthesis
- **Link Library**
  - Contain target library, DesignWare and your design files.
  - Design Compiler uses the link library to resolve design references

# Specify Libraries



| 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 |

```
set_app_var search_path “$search_path ./mylib”

set_app_var target_library “vendor_xxx.db”
set_app_var symbol_library “vendor_xxx.sdb”
set_app_var synthetic_library “dw_foundation.sldb”
set_app_var link_library “* $target_library $synthetic_library”
```

# Read Design Files

# Read Design Files

- Use **read_file** command
  - read all formats of files using argument

```
read_file –format verilog my_design.v
read_file –format ddc my_design.ddc
```

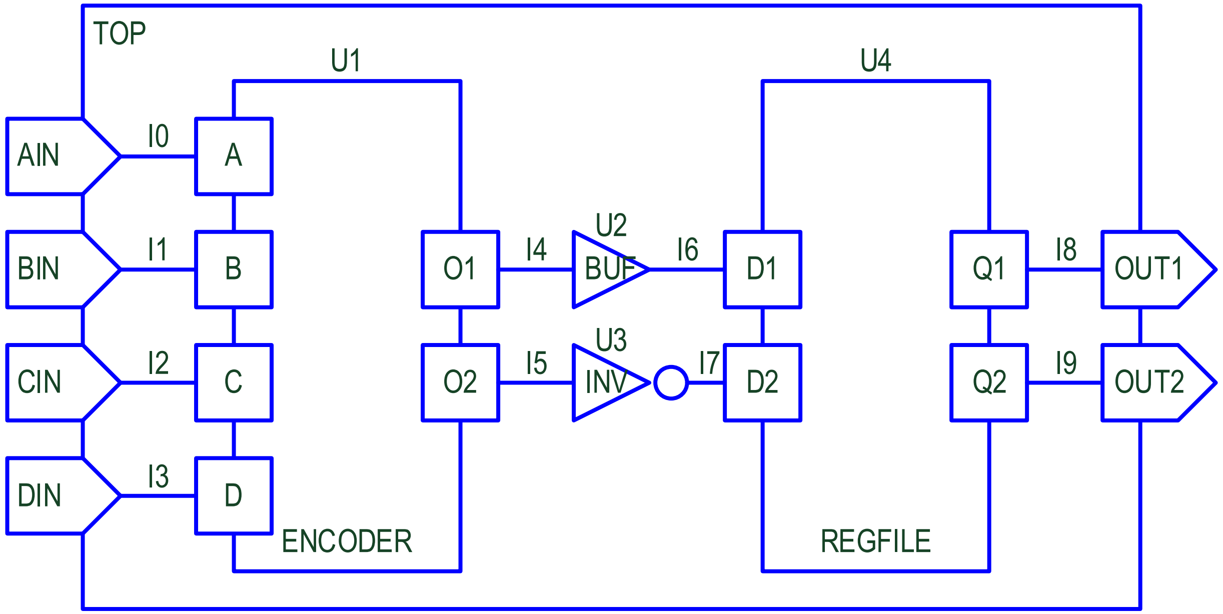
- Use **read_xxx** commands (read_verilog, read_vhdl, read_ddc …)
  - read a specific format  of file

```
read_verilog my_design.v
read_ddc my_design.ddc
```

- Use **analyze** and **elaborate** commands
  - use analyze and elaborate to build the parameterized designs

```
analyze –format verilog my_design.v
analyze –format verilog my_subbk1.v
...
elaborate
```

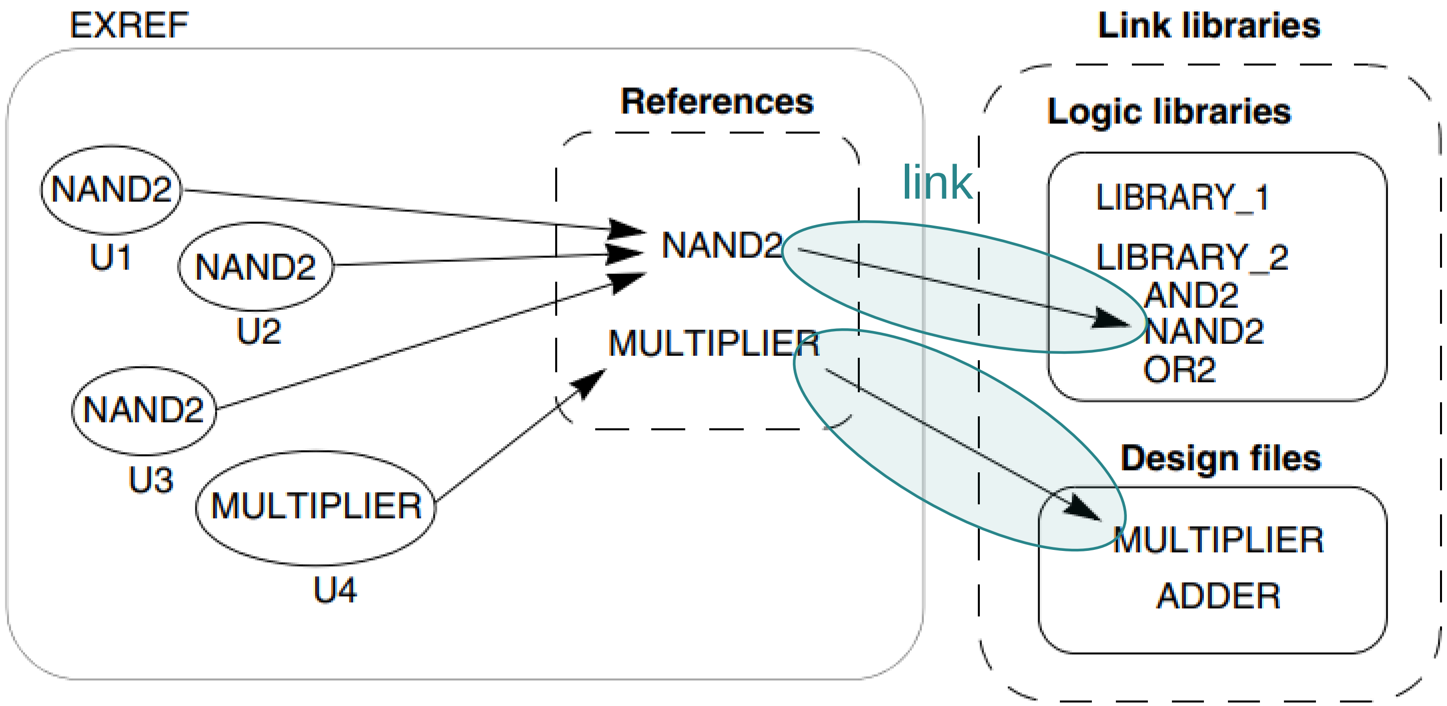
# Design Objects



| Design Objects | |
|---|---|
| Designs | TOP, ENCODER, REGFILE |
| References | ENCODER, REGFILE, BUF, INV |
| Cells | U1, U2, U3, U4 |
| Ports | AIN, BIN, CIN, DIN, OUT1, OUT2 |
| Pins | A, B, C, D, O1, O2, D1, D2, Q1, Q2 |
| Nets | I0, I1, I2, …, I9 |

# Linking Designs

- A design must connect to all the library components it references
- Use link command with **link_library** and **search_path** to resolve design references



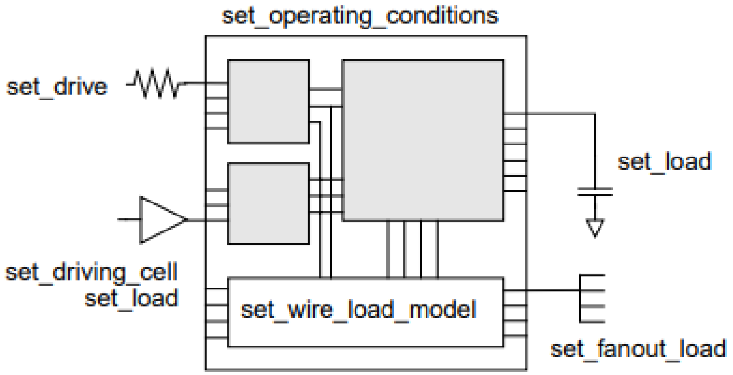
# Define Design Environment

# Design Environment

- Environment in which the design is expected to operate.

Modern designs may operate under multiple conditions. The multimode-multicorner (MMMC) optimization is another scenario.

*Figure 6-1 Commands Used to Define the Design Environment*



| Commands | Description |
|---|---|
| set_operating_conditions | Set operating conditions for the current design |
| set_wire_load_model | Set wire load model for the current design |
| set_wire_load_mode | Set wire load mode for the current design |
| set_driving_cell | Set drive characteristics on ports |
| set_drive | Set drive resistance value on ports |
| set_load | Set load capacitance value on ports |
| set_fanout_load | Set external fanout values on output ports |

# Operating Condition

- set_operating_conditions
  - To specify the process, voltage, and temperature (PVT) conditions under which the design should be synthesized and analyzed.
  - The condition name is written in technology .lib file

```
operating_conditions(WCCOM) {
   temperature : 25 ;
   voltage : 0.9 ;
 }
```

  - Set operating condition to design compiler

```
set_operating_conditions WCCOM
```

# Wire Load Model

- **Wire Load Model**
  - An estimate of a net’s RC parasitics based on the net’s fanout
  - Used to calculate the delay of nets in the absence of placement and routing information.

  - The model name is written in technology .lib file

```
wire_load(“50x50") {
   resistance : 0.004714;
   capacitance : 0.000218;
   area : 1e-40;
   slope : 6.12;
   fanout_length (1, 9.18);
 }
```

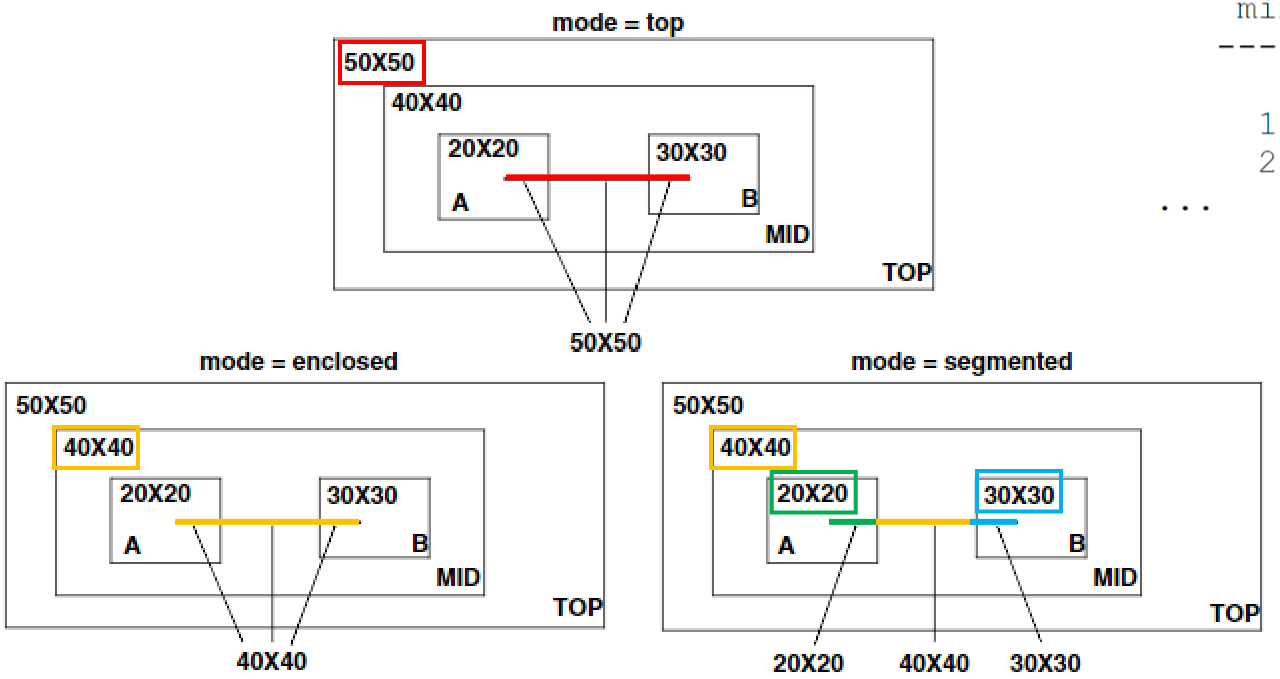
  - Set wire load model and mode to design compiler

```
set_wire_load_model -name 50x50
set_wire_load_mode segmented
```

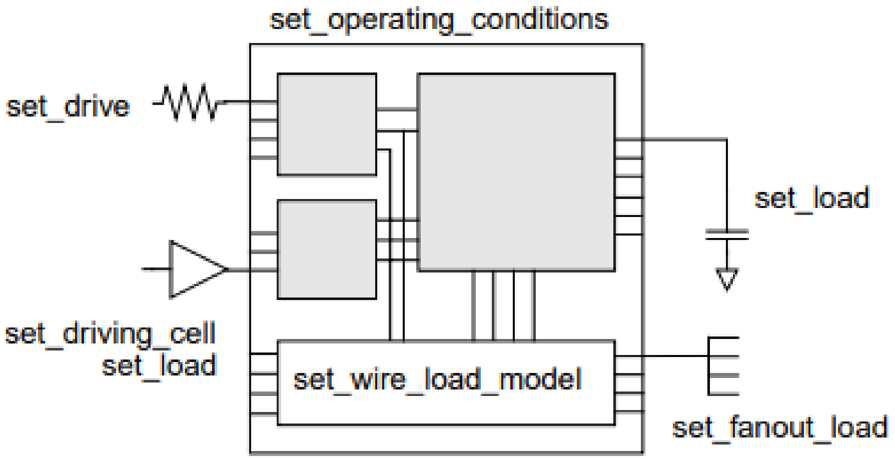
# Wire Load Mode

- **Wire Load Mode**
  - top
  - enclosed
  - segmented

```
Wire Loading Model Selection Group:

   Name              : my_lib

        Selection                   Wire load name
    min area    max area
   --------------------------------------------
        0.00     1000.00                05x05
     1000.00     2000.00                10x10
     2000.00     3000.00                20x20
...
```



# Design Environment example



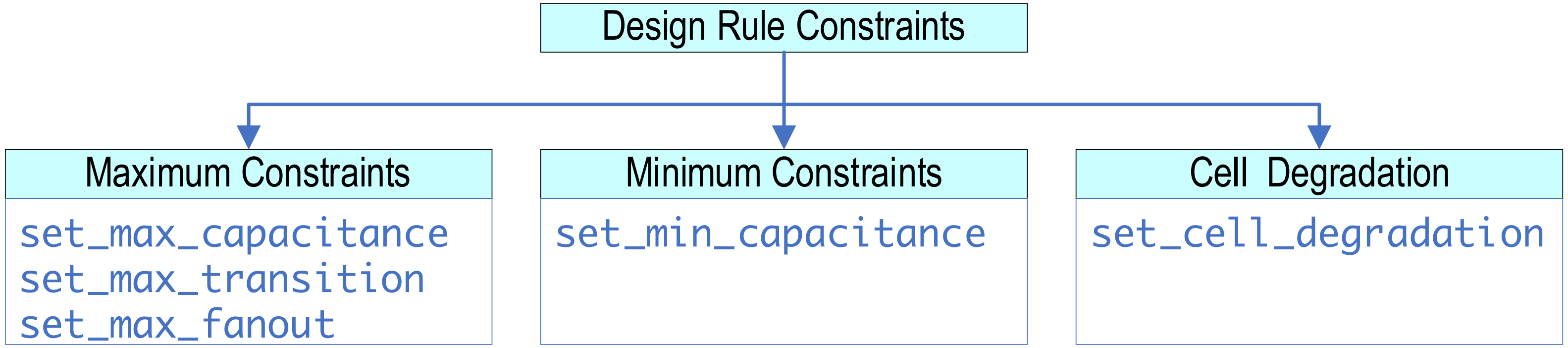
```
set_operating_conditions WCCOM
set_wire_load_model -name 50x50
set_wire_load_mode segmented
set_driving_cell -lib_cell BUF {IN1}
set_drive 1.5 {IN2, IN3}
set_load 2.2 {OUT1, OUT2}
set_fanout_load 3 {OUT3}
...
```

The names are different between vendors library

# Set Design Constraint

# Design Constraints

- Design rule constraints
  - Design rules constraints restrict transition times, fanout loads, and capacitances



Design Rule Constraints

Maximum Constraints

set_max_capacitance
set_max_transition
set_max_fanout

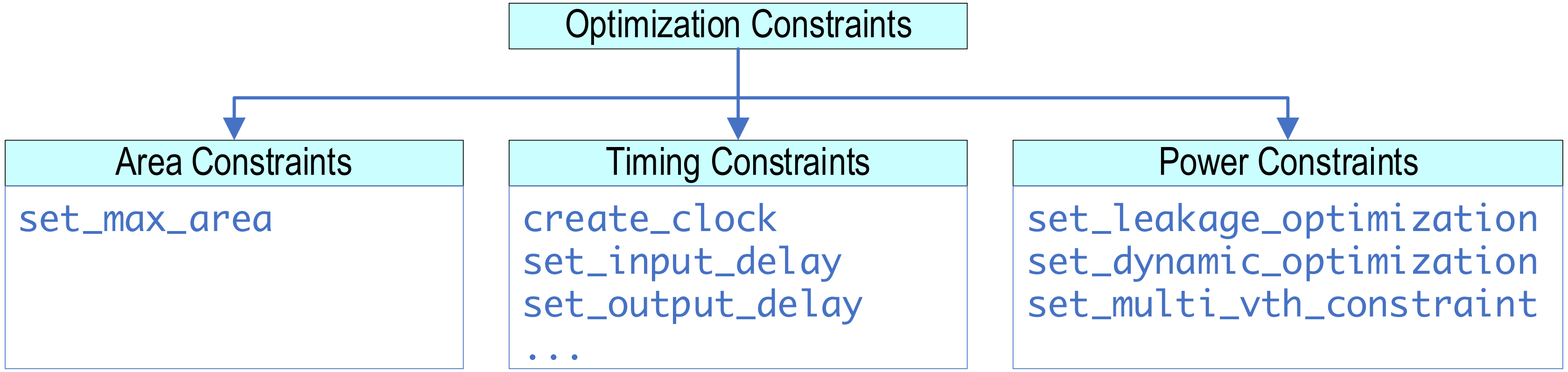
Minimum Constraints

set_min_capacitance

Cell Degradation

set_cell_degradation

- Design optimization constraints (goals)
  - Design optimization constraints restrict speed, area and power design goals



Optimization Constraints

Area Constraints

set_max_area

Timing Constraints

create_clock
set_input_delay
set_output_delay
...

Power Constraints

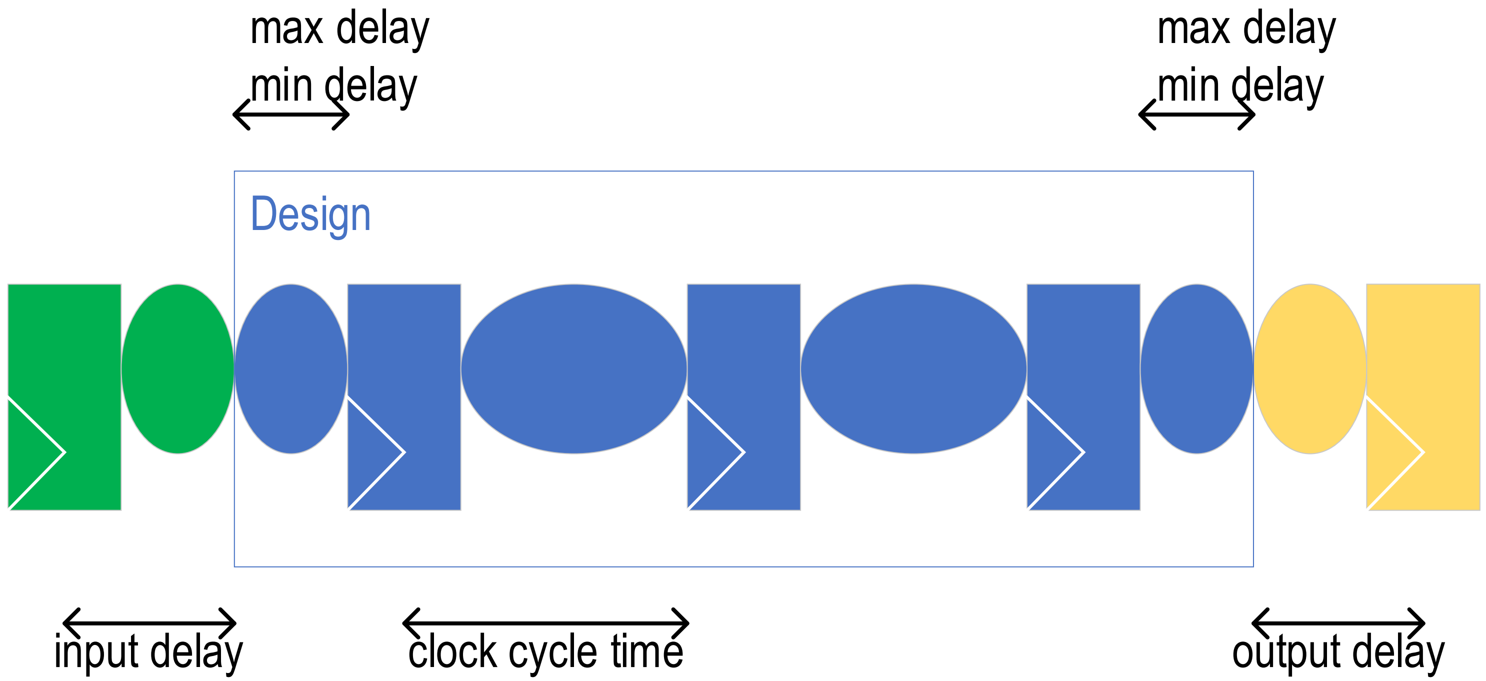
set_leakage_optimization
set_dynamic_optimization
set_multi_vth_constraint

# Timing Constraints

| Commands | Description |
|---|---|
| create_clock | Creates a clock object and defines its waveform |
| set_input_delay | Sets input delay on ports relative to a clock signal |
| set_output_delay | Sets output delay on ports relative to a clock signal |
| set_max_delay | Specifies a maximum delay for paths |
| set_min_delay | Specifies a minimum delay for paths |
| …etc | |

There are many other timing constraints used for the more complicated designs.

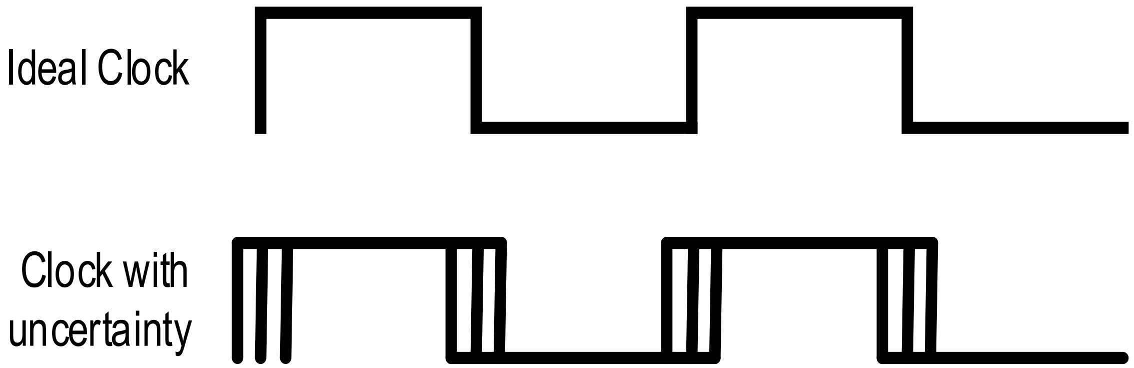
# Timing Constraints (cont.)



```
//direct assign a delay constraint
set_max_delay 3 –from [get_cells FF1] –to [get_ports OUT1]

//or constraint related to a clock
create_clock –name clk –period 10 [get_ports clk]
set_input_delay 5 -clock clk [all_inputs]
set_output_delay 5 –clock clk [all_outputs]
```

# Clock Uncertainty

- Clock uncertainty
  - Clock uncertainty is the possible time variation between any pair of clock edges
  - Clock uncertainty is caused by the clock jitter or skew
    - Jitter: caused by clock source
    - Skew: cause by different clock path



- Set clock uncertainty command to describe this behavior

```
set_clock_uncertainty –setup 0.5 clk
set_clock_uncertainty –hold  0.2 clk
```

# Synthesis and Optimize

# Compile and Save Design

- Use **current_design** commands
  - Set the design you are working on

```
current_design TOP
```

- Use **compile** or **compile_ultra** commands
  - Synthesize RTL designs to optimized, technology dependent, gate level design
  - compile_ultra has additional features: DesignWare components …etc

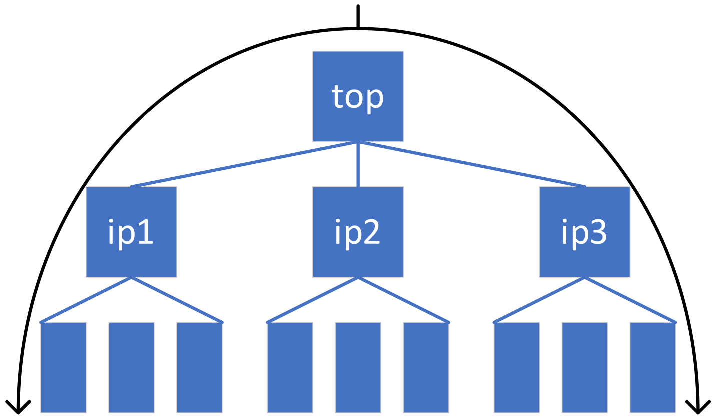
```
compile_ultra
```

- Use **write_file** command
  - ddc is the synopsys internal database format using for the design flows
  - verilog is standard format using for pre-layout simulation (netlist)

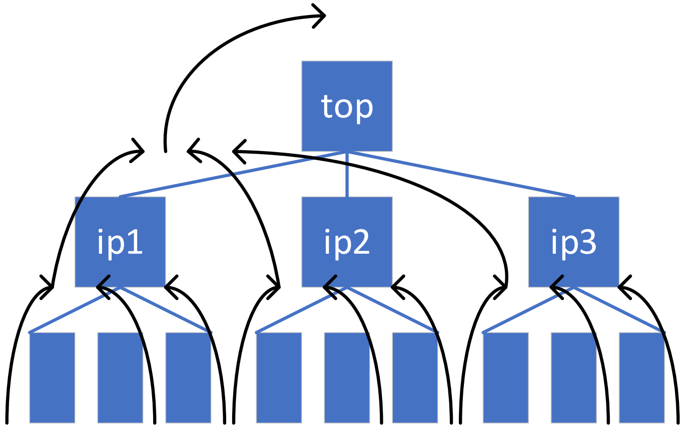
```
write_file -hierarchy -format ddc -output initial_compile.ddc
write_file -hierarchy -format verilog -output my_design.v
```

# Compile Strategy

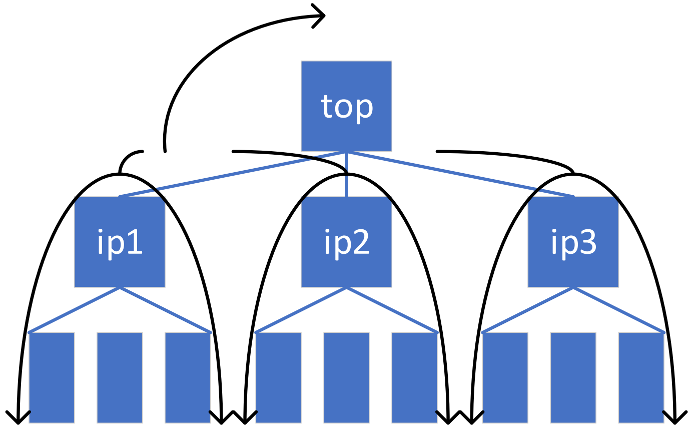
- Top-Down
  - The top-level and all its sub-designs are compiled together
- Bottom-Up
  - The individual sub-designs are compiled separately, starting from the bottom of the hierarch and proceeding up through the levels of the hierarchy until the top-level design is compiled.
- Mixed
  - Mix the Top-Down and Bottom-Up strategy



Top-Down Strategy



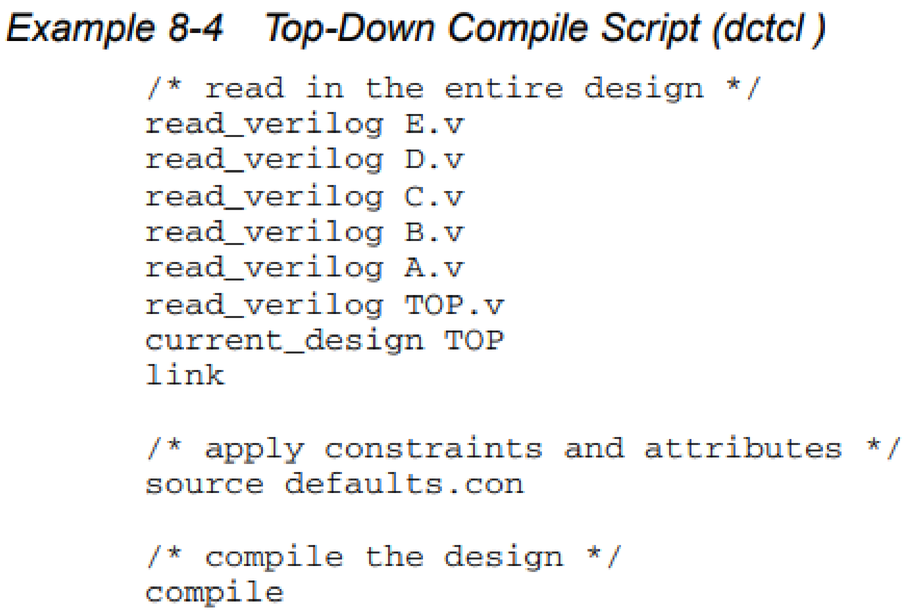
Bottom-Up Strategy



Mixed Strategy

# Top-Down Compile Strategy

- Advantage
  - Provides a push-button approach
  - Takes care of interblock dependencies automatically
  - Recommend: design area < 100K gates

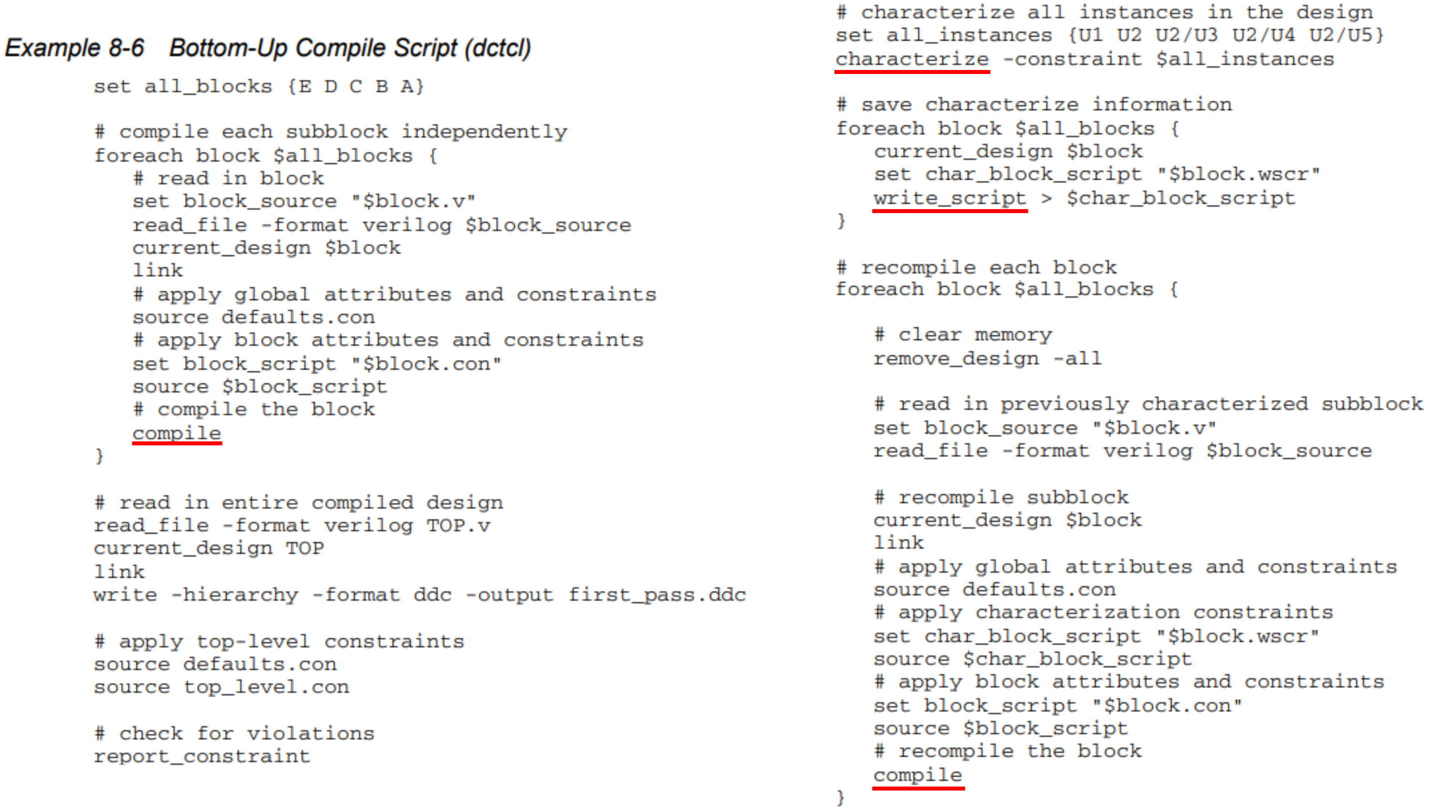


*Example 8-4 Top-Down Compile Script (dctcl )*

```
/* read in the entire design */
read_verilog E.v
read_verilog D.v
read_verilog C.v
read_verilog B.v
read_verilog A.v
read_verilog TOP.v
current_design TOP
link

/* apply constraints and attributes */
source defaults.con

/* compile the design */
compile
```

# Bottom-Up Compile Strategy

- Also known as the compile-characterize-write_script-recompile method.
  - divide-and-conquer approach
  - Requires less memory than top-down compile



*Example 8-6 Bottom-Up Compile Script (dctcl)*

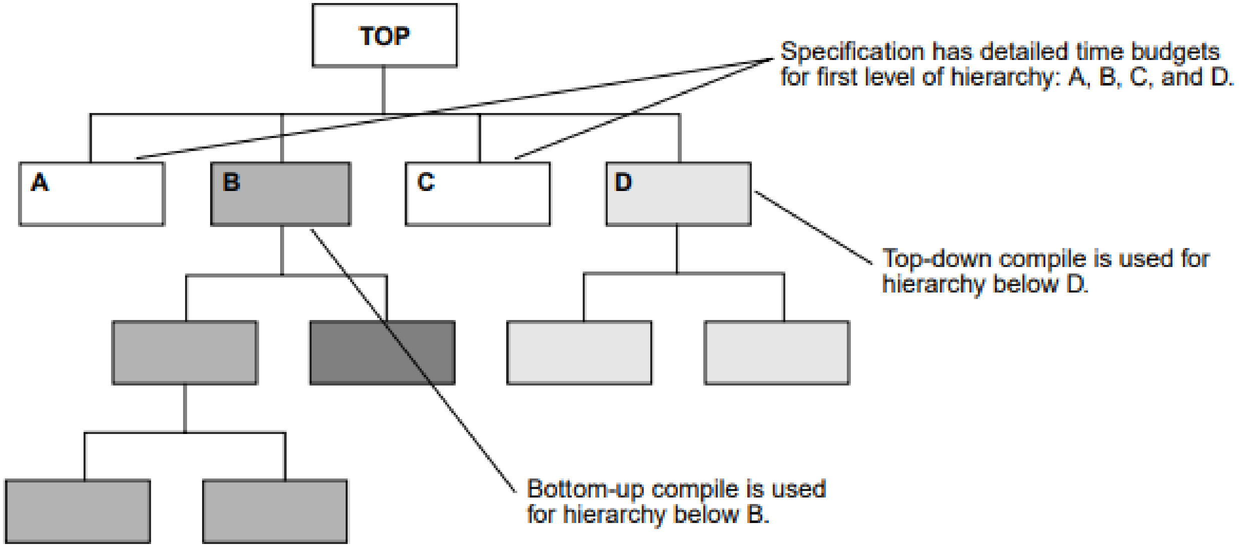
```
set all_blocks {E D C B A}

# compile each subblock independently
foreach block $all_blocks {
   # read in block
   set block_source "$block.v"
   read_file -format verilog $block_source
   current_design $block
   link
   # apply global attributes and constraints
   source defaults.con
   # apply block attributes and constraints
   set block_script "$block.con"
   source $block_script
   # compile the block
   compile
}

# read in entire compiled design
read_file -format verilog TOP.v
current_design TOP
link
write -hierarchy -format ddc -output first_pass.ddc

# apply top-level constraints
source defaults.con
source top_level.con

# check for violations
report_constraint
```

```
# characterize all instances in the design
set all_instances {U1 U2 U2/U3 U2/U4 U2/U5}
characterize -constraint $all_instances

# save characterize information
foreach block $all_blocks {
   current_design $block
   set char_block_script "$block.wscr"
   write_script > $char_block_script
}

# recompile each block
foreach block $all_blocks {

   # clear memory
   remove_design -all

   # read in previously characterized subblock
   set block_source "$block.v"
   read_file -format verilog $block_source

   # recompile subblock
   current_design $block
   link
   # apply global attributes and constraints
   source defaults.con
   # apply characterization constraints
   set char_block_script "$block.wscr"
   source $char_block_script
   # apply block attributes and constraints
   set block_script "$block.con"
   source $block_script
   # recompile the block
   compile
}
```

# Mixed Compile Strategy

- take advantage of the benefits of both the top-down and the bottom-up compile strategies
  - top-down for small hierarchies of blocks.
  - bottom-up to tie small hierarchies together into larger blocks.

*Figure 8-2 Mixing Compilation Strategies*

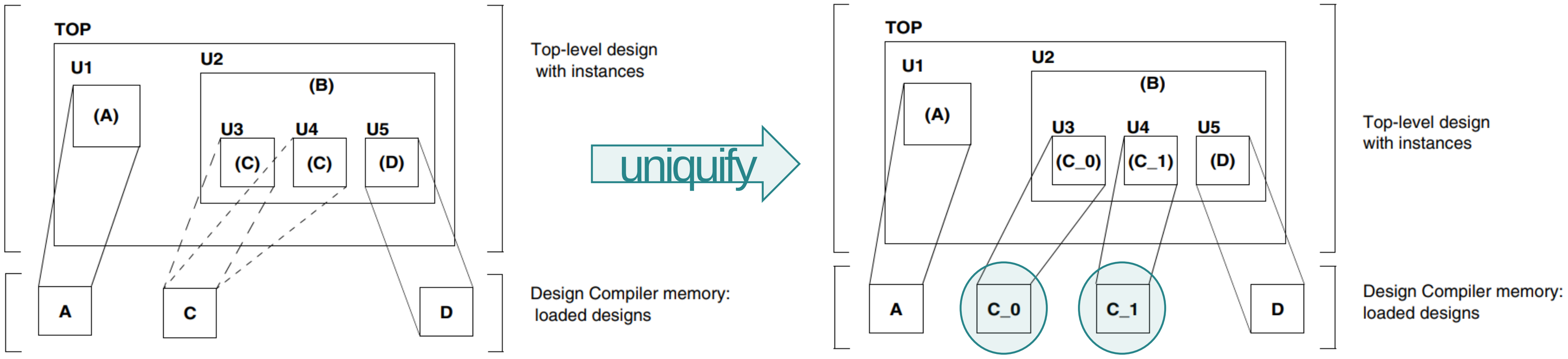

TOP
A
B
C
D
Specification has detailed time budgets for first level of hierarchy: A, B, C, and D.
Top-down compile is used for hierarchy below D.
Bottom-up compile is used for hierarchy below B.

# Resolving Multiple Instances

- Uniquify Method
  - Use **uniquify** command, the tool automatically uniquifies sub-designs
- Compile-Once-Don’t-Touch Method
  - Use **set_dont_touch** command to preserve the compiled sub-design
- Ungroup Method
  - Use **ungroup** command to remove the hierarchy

| Method | Advantage | Disadvantage |
|---|---|---|
| uniquify | • Better local characterization | • More working memory<br>• Longer compile time |
| set_dont_touch | • Less working memory<br>• Less compile time | • Bad local characterization |
| ungroup | • Best synthesis result | • More working memory<br>• Longer compile time<br>• No user defined hierarchy |

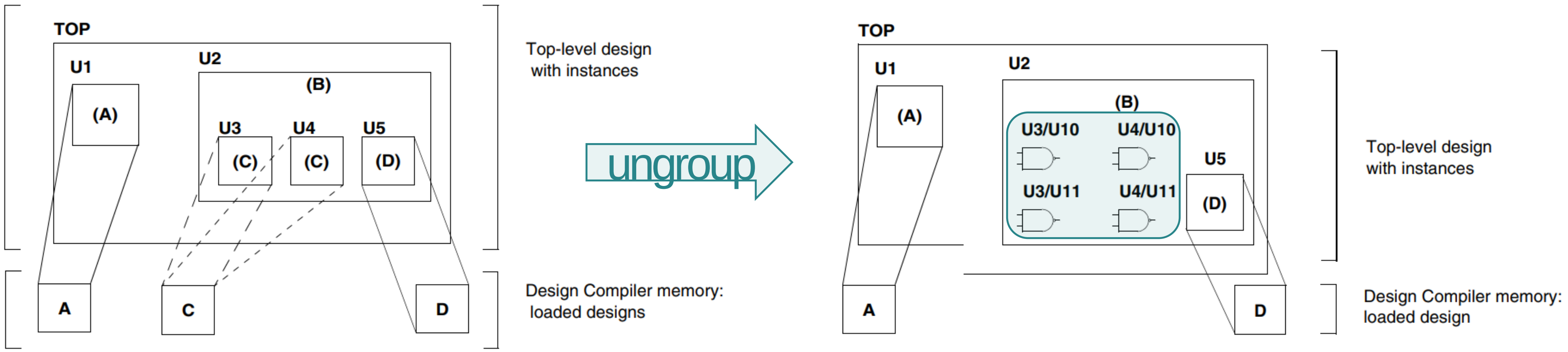
# Uniquify Method


TOP
U1
(A)
U2
(B)
U3
(C)
U4
(C)
U5
(D)
A
C
D
Top-level design with instances
Design Compiler memory: loaded designs
uniquify
TOP
U1
(A)
U2
(B)
U3
(C_0)
U4
(C_1)
U5
(D)
A
C_0
C_1
D
Top-level design with instances
Design Compiler memory: loaded designs

```
...
current_design TOP
uniquify
compile_ultra
...
```

- Often used in top-down compile strategy

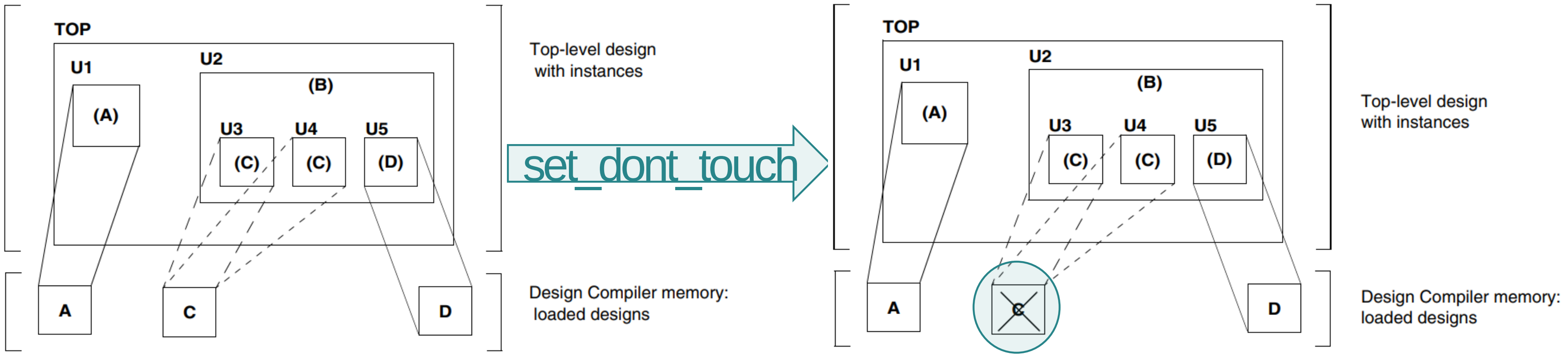
# Ungroup Method



```
...
current_design B
ungroup {U3 U4}
current_design TOP
compile_ultra
...
```

- Often used in top-down compile strategy

# Compile-Once-Don’t-Touch Method


TOP
U1
U2
(B)
(A)
U3
U4
U5
(C)
(C)
(D)
A
C
D
Top-level design with instances
Design Compiler memory: loaded designs
set_dont_touch
TOP
U1
U2
(B)
(A)
U3
U4
U5
(C)
(C)
(D)
A
C
D
Top-level design with instances
Design Compiler memory: loaded designs

```
...
current_design TOP
characterize U2/U3
current_design C
compile_ultra
current_design TOP
set_dont_touch {U2/U3 U2/U4}
compile_ultra
...
```

- Often used in bottom-up or mixed compile strategy

# Analyze and Resolve Problems

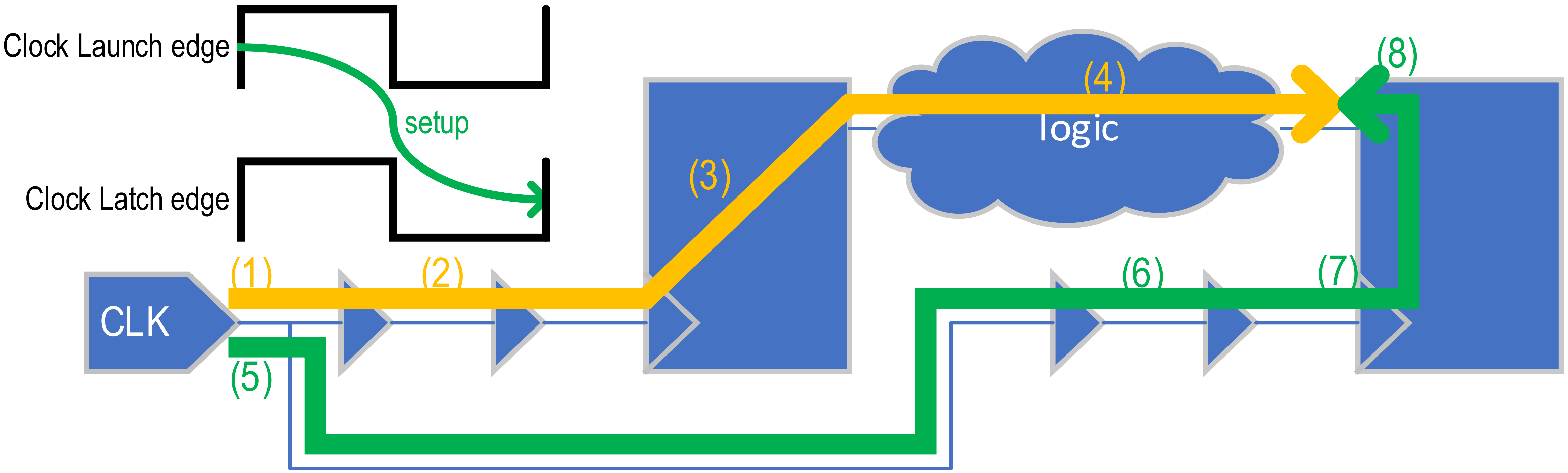
# Analyze Design

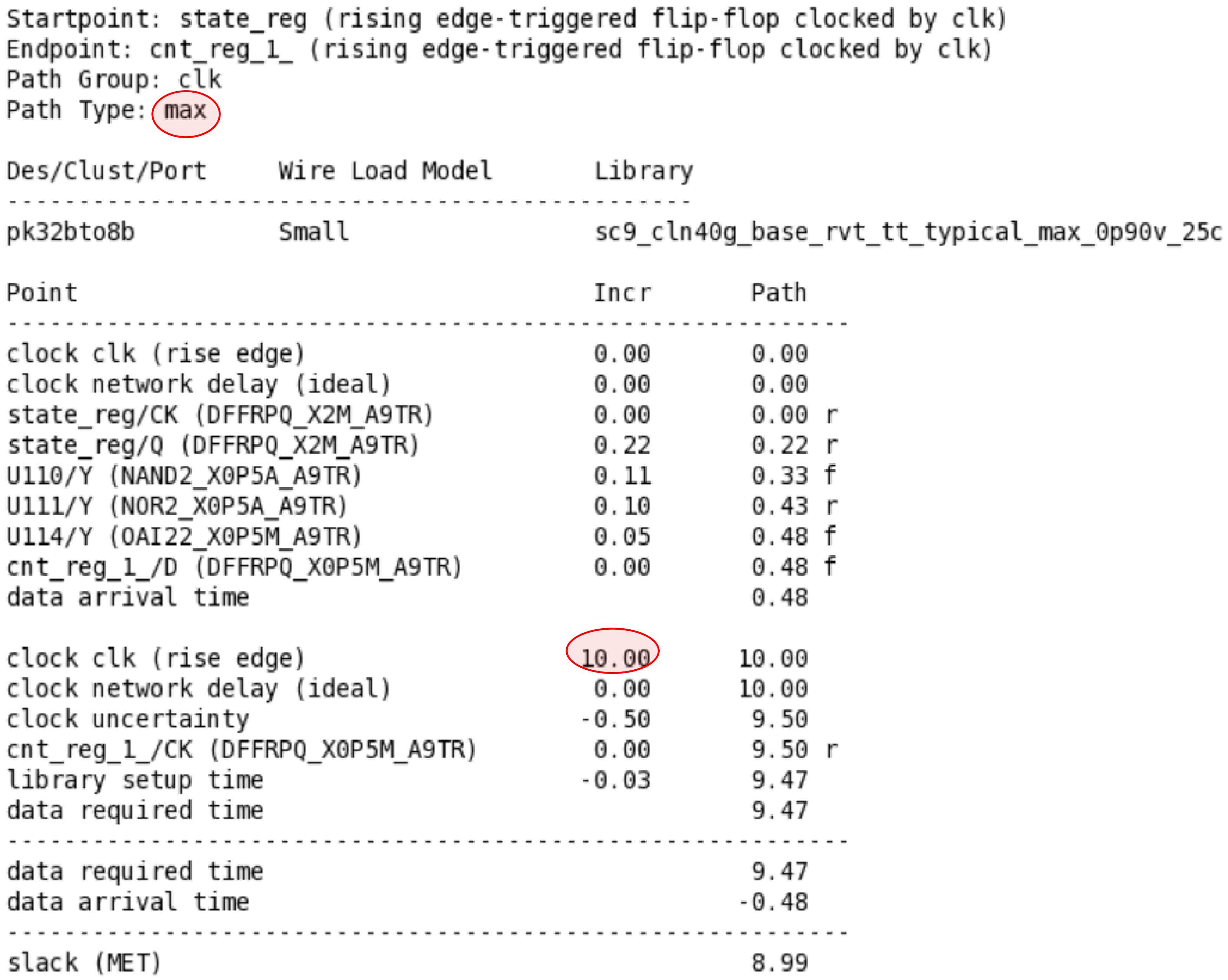
| Commands | Description |
|---|---|
| report_constraint | Displays constraint-related information for the current design |
| report_area | Displays area information for the current design |
| report_timing | Displays timing information for the current design |

```
report_constraint –verbose > design_cons.rpt
report_area –hierarchy –designware > design_area.rpt
report_timing > design_time.rpt
```

# Setup Time Slack

- Setup time slack = data required time – data arrived time
  - data arrive time = clock launch time(1) + clock network delay(2) + clock to q delay(3) + max logic delay(4)
  - data required time = clock latch time(5) + clock network delay(6) – clock uncertainty(7) – setup time(8)

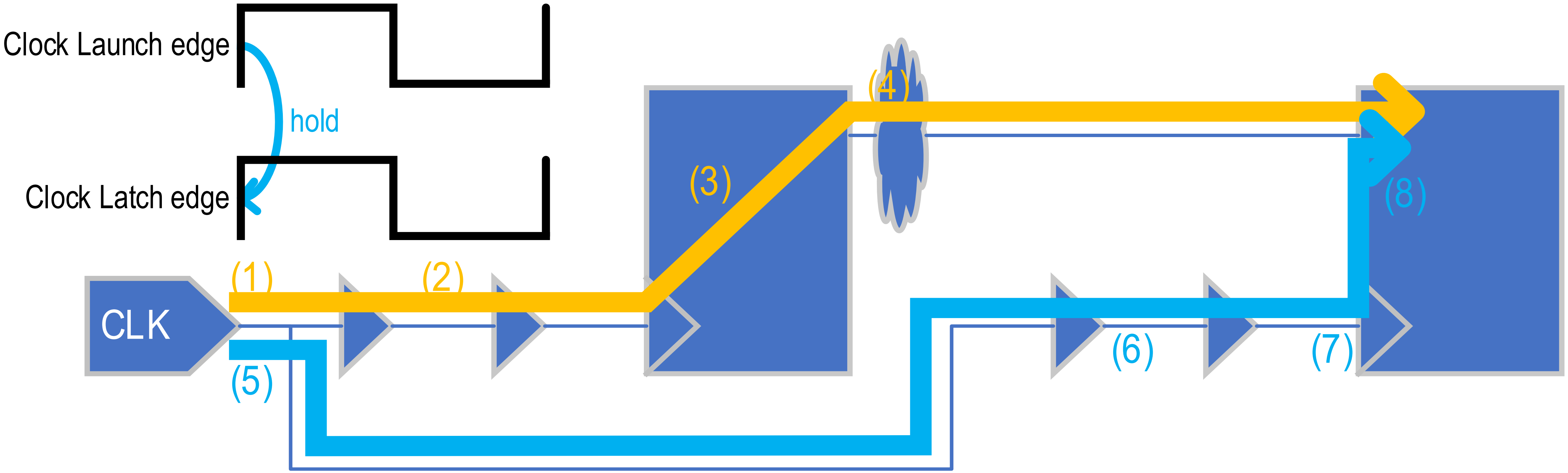


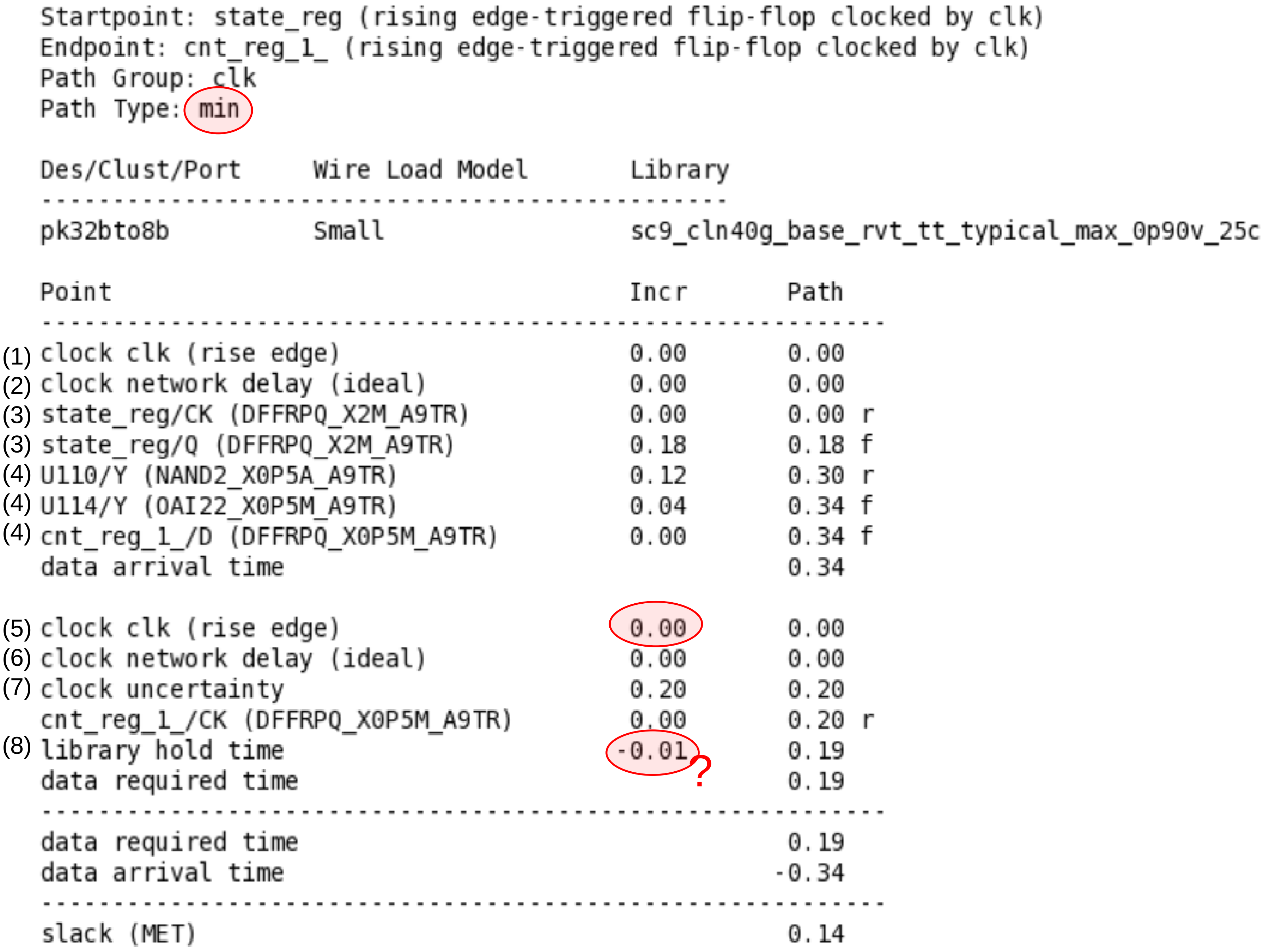


```
    Startpoint: state_reg (rising edge-triggered flip-flop clocked by clk)
    Endpoint: cnt_reg_1_ (rising edge-triggered flip-flop clocked by clk)
    Path Group: clk
    Path Type: max

    Des/Clust/Port     Wire Load Model      Library
    ------------------------------------------------
    pk32bto8b          Small                sc9_cln40g_base_rvt_tt_typical_max_0p90v_25c

    Point                                   Incr       Path
    -----------------------------------------------------------
(1) clock clk (rise edge)                   0.00       0.00
(2) clock network delay (ideal)             0.00       0.00
(3) state_reg/CK (DFFRPQ_X2M_A9TR)          0.00       0.00 r
(3) state_reg/Q (DFFRPQ_X2M_A9TR)           0.22       0.22 r
(4) U110/Y (NAND2_X0P5A_A9TR)               0.11       0.33 f
(4) U111/Y (NOR2_X0P5A_A9TR)                0.10       0.43 r
(4) U114/Y (OAI22_X0P5M_A9TR)               0.05       0.48 f
(4) cnt_reg_1_/D (DFFRPQ_X0P5M_A9TR)        0.00       0.48 f
    data arrival time                                  0.48

(5) clock clk (rise edge)                  10.00      10.00
(6) clock network delay (ideal)             0.00      10.00
(7) clock uncertainty                      -0.50       9.50
    cnt_reg_1_/CK (DFFRPQ_X0P5M_A9TR)       0.00       9.50 r
(8) library setup time                     -0.03       9.47
    data required time                                 9.47
    -----------------------------------------------------------
    data required time                                 9.47
    data arrival time                                 -0.48
    -----------------------------------------------------------
    slack (MET)                                        8.99
```

# Hold Time Slack

- Hold time slack = data arrived time – data required time
  - data arrive time = clock launch time(1) + clock network delay(2) + clock to q delay(3) + min logic delay(4)
  - data required time = clock latch time(5) + clock network delay(6) + clock uncertainty(7) + hold time(8)





```
    Startpoint: state_reg (rising edge-triggered flip-flop clocked by clk)
    Endpoint: cnt_reg_1_ (rising edge-triggered flip-flop clocked by clk)
    Path Group: clk
    Path Type: min

    Des/Clust/Port     Wire Load Model       Library
    ------------------------------------------------
    pk32bto8b          Small                 sc9_cln40g_base_rvt_tt_typical_max_0p90v_25c

    Point                                    Incr       Path
    -----------------------------------------------------------
(1) clock clk (rise edge)                    0.00       0.00
(2) clock network delay (ideal)              0.00       0.00
(3) state_reg/CK (DFFRPQ_X2M_A9TR)           0.00       0.00 r
(3) state_reg/Q (DFFRPQ_X2M_A9TR)            0.18       0.18 f
(4) U110/Y (NAND2_X0P5A_A9TR)                0.12       0.30 r
(4) U114/Y (OAI22_X0P5M_A9TR)                0.04       0.34 f
(4) cnt_reg_1_/D (DFFRPQ_X0P5M_A9TR)         0.00       0.34 f
    data arrival time                                   0.34

(5) clock clk (rise edge)                    0.00       0.00
(6) clock network delay (ideal)              0.00       0.00
(7) clock uncertainty                        0.20       0.20
    cnt_reg_1_/CK (DFFRPQ_X0P5M_A9TR)        0.00       0.20 r
(8) library hold time                       -0.01 ?     0.19
    data required time                                  0.19
    -----------------------------------------------------------
    data required time                                  0.19
    data arrival time                                  -0.34
    -----------------------------------------------------------
    slack (MET)                                         0.14
```

# Reference

- Design Compiler User Guide
- Using Tcl with Synopsys Tools

# Feedback to us

https://forms.office.com/r/DYDu8vLaWN

# Thank you!

PUFacademy

A PUFsecurity Alliance

Visit our website: pufacademy.com

Contact us: PUFacademy@pufsecurity.com