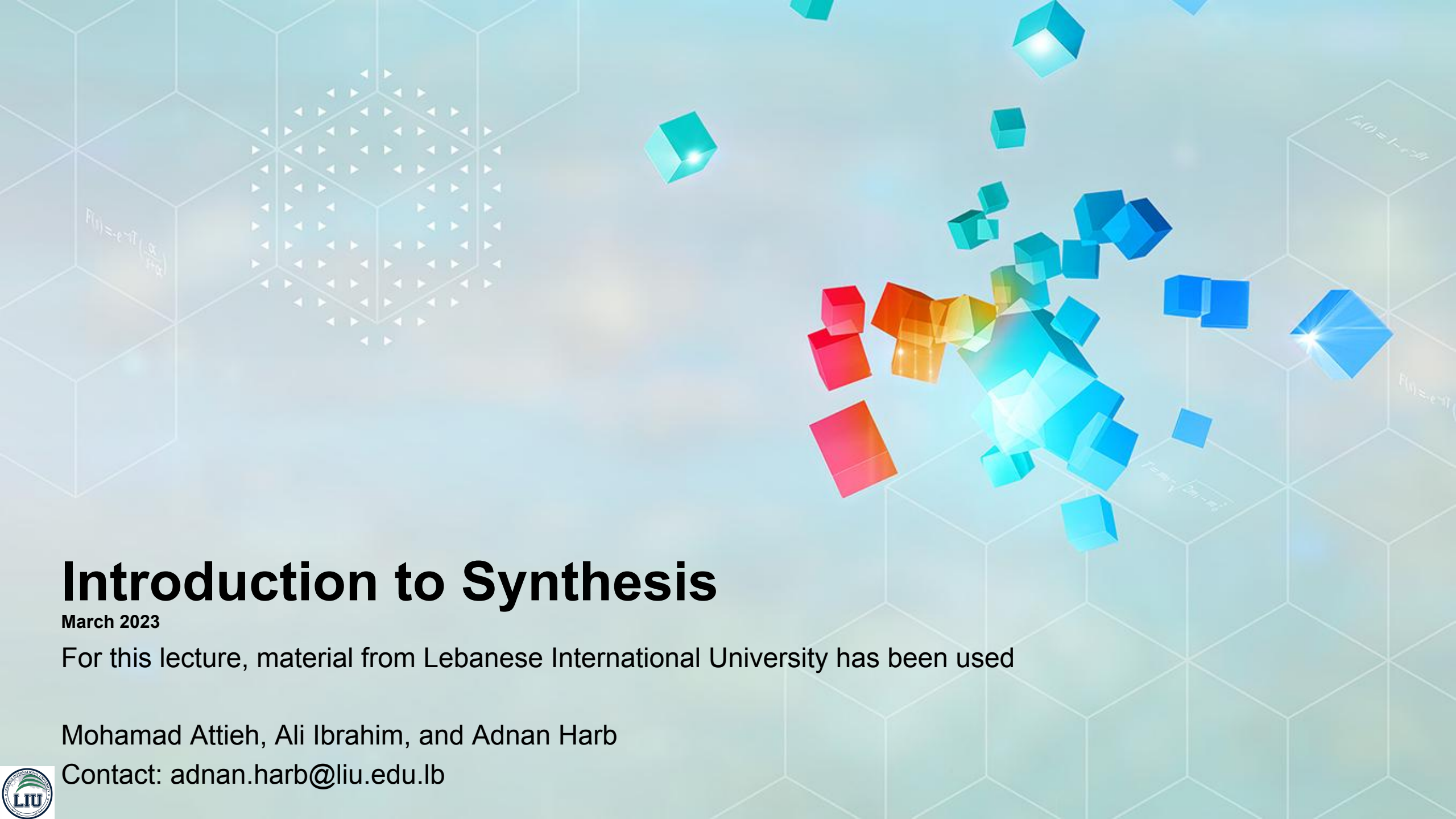


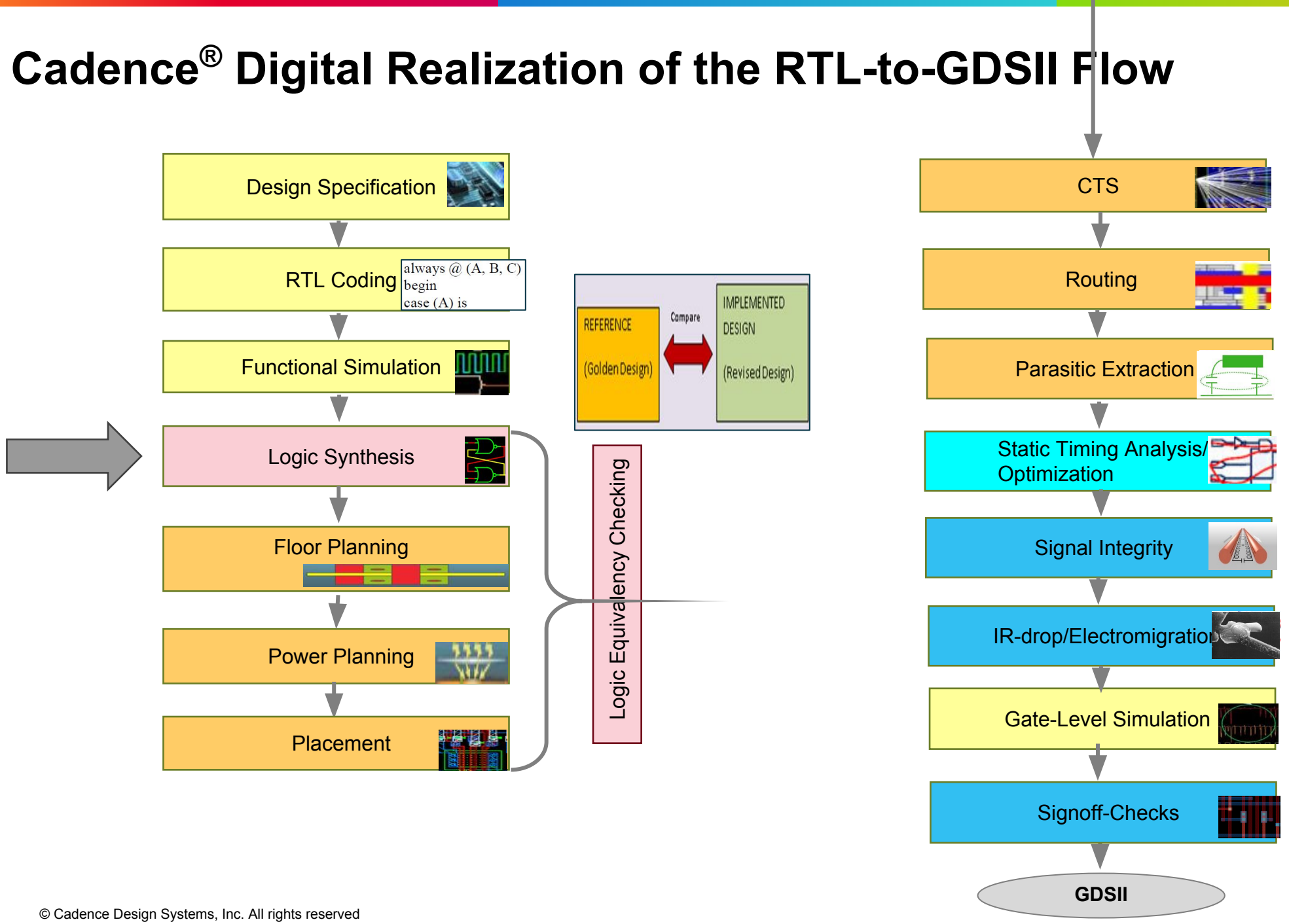
Introduction to Synthesis

March 2023

For this lecture, material from Lebanese International University has been used

Mohamad Attieh, Ali Ibrahim, and Adnan Harb
Contact: adnan.harb@liu.edu.lb

Cadence® Digital Realization of the RTL-to-GDSII Flow



Lecture Outline

This lecture is organized as follows:

Part A: Includes a general explanation of the synthesis theory

Part B: Introduces the design rule constraints

Part C: Provides some tips on script preparation before the synthesis

Part D: Explains the synthesis steps using Genus tool

Part E: Presents the Design for Testability Process

cadence

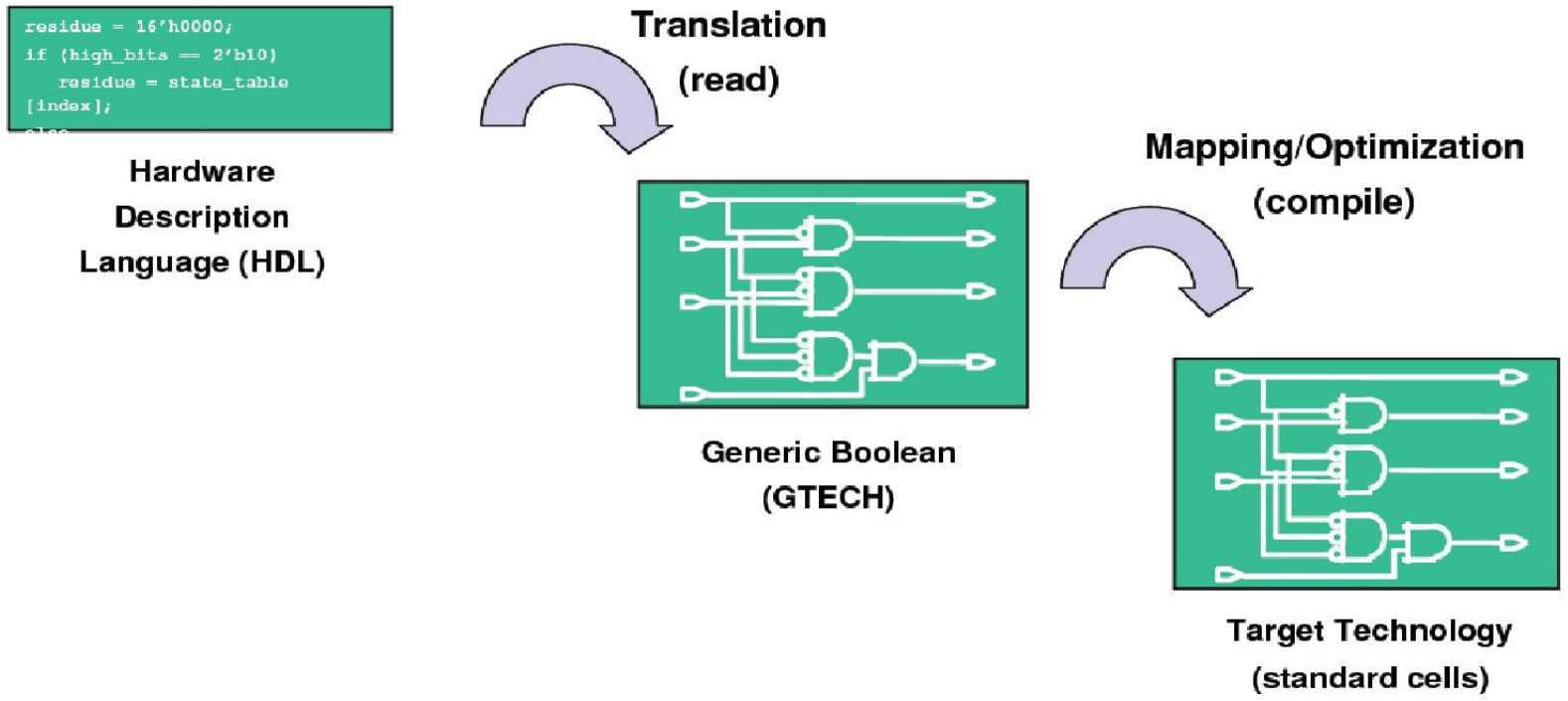
Part A: The Synthesis Theory

The Synthesis Theory

Definition:

Logic Synthesis is the process of transforming **Hardware Description Language (HDL) code** **into a logic circuit based on:**

1. a standard-cell library
2. user-specified optimization/timing constraints.



"Source: Cadence Customer Training Course RTLtoGDSII, Version: 5.0"

The Synthesis Theory

The Synthesis process consists of three main stages which are:

LOGIC SYNTHESIS = TRANSLATION + MAPPING + OPTIMIZATION

TRANSLATION

Translation is the first step of the logical synthesis where the RTL is converted to a general library netlist. During translation, the following operations are performed:

1. HDL syntax checking
2. Optimizes HDL
3. Conversion of HDL into functional Boolean equivalent
4. Mapping of the arithmetic, sequential, and combinational function

MAPPING

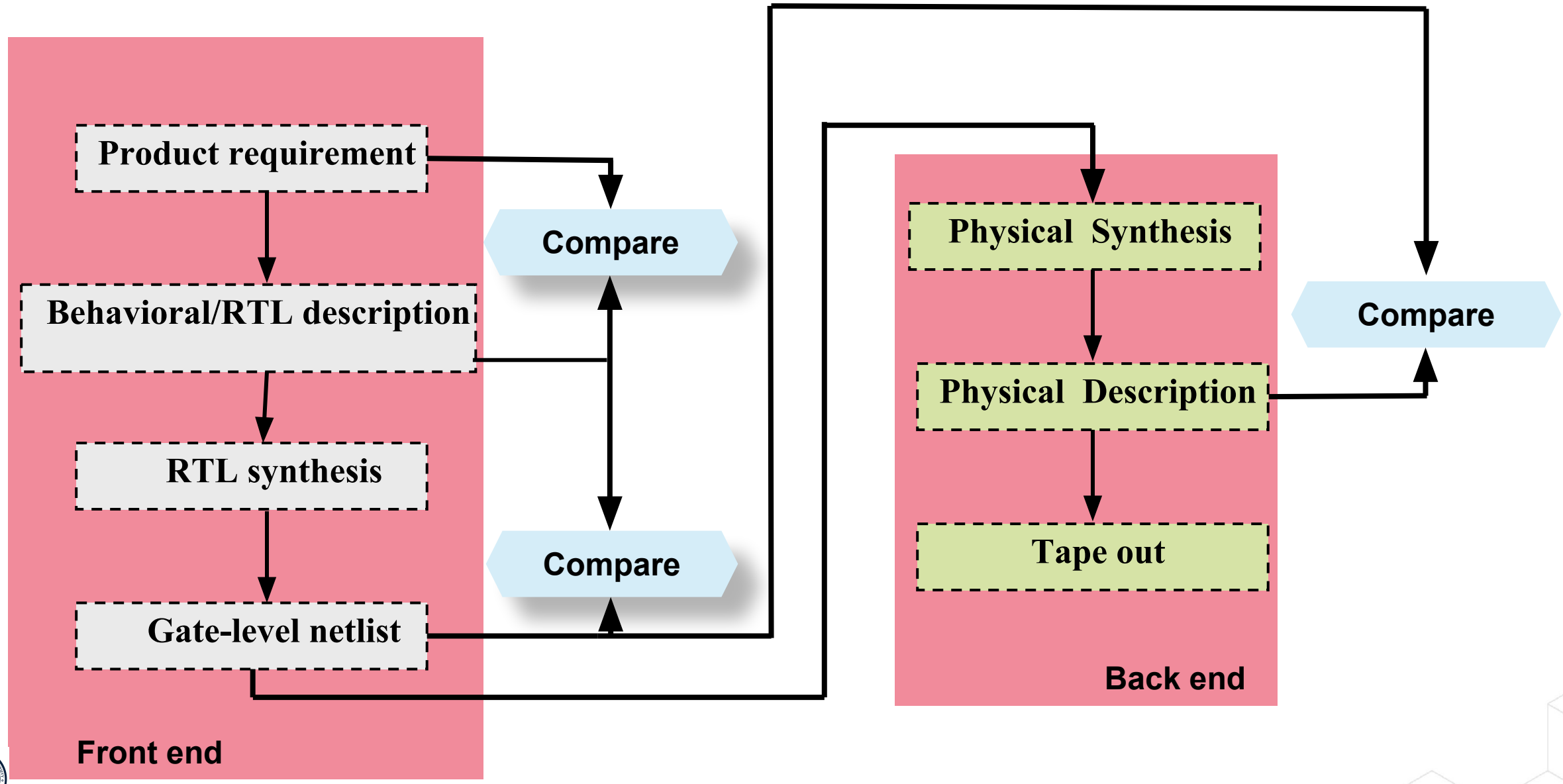
During mapping, the tool maps the generic Boolean netlist generated from the translation step, into the available gates in the standard cell library. The Boolean functions are mapped into technology-specific primitive functions.

OPTIMIZATION

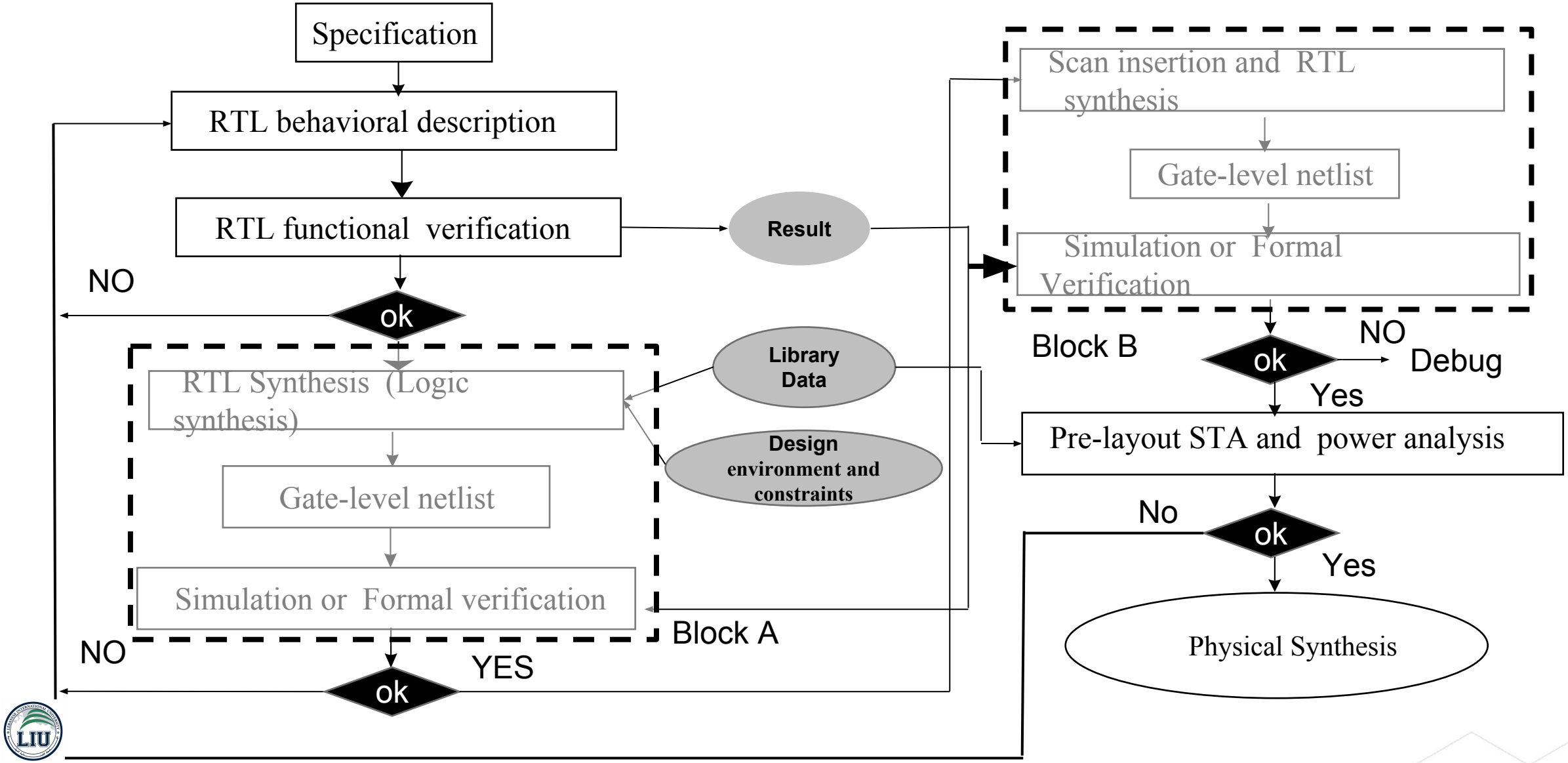
In the optimization step, the mapping are modified in order to meet specific design goals such as speed, area, and power consumption

cadence

The Digital Design Flow



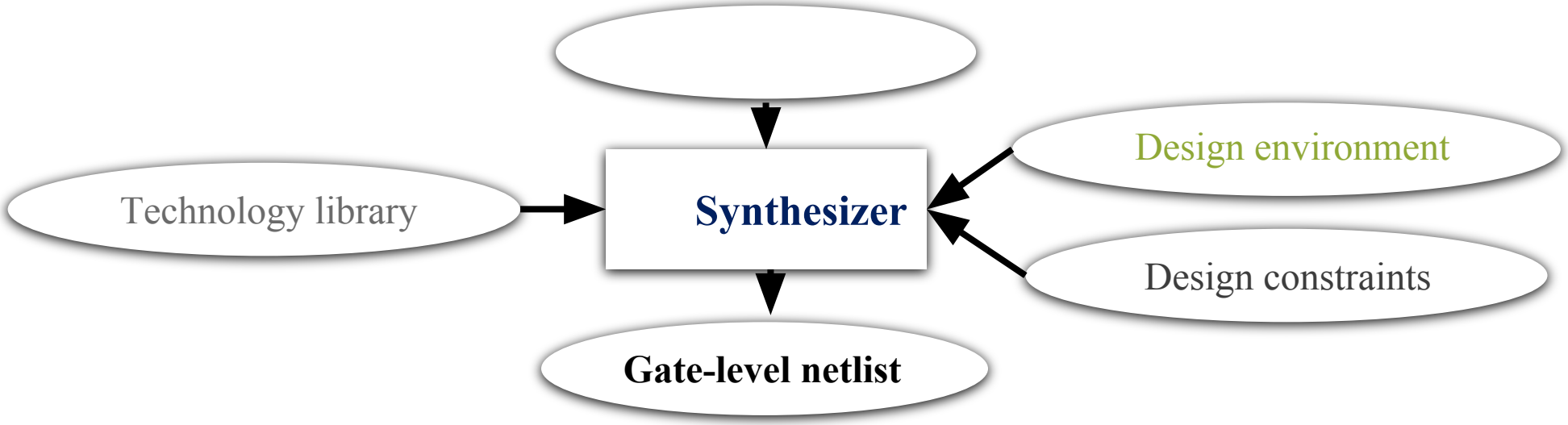
The RTL Synthesis Flow



Logic Synthesis Environment

The following must be provided to synthesis tools:

- **Design environment**
- **Design constraints**
- **Gate-level netlist**
- **Technology library**



Design Environment

Specify those directly influence

- design synthesis and optimization results

External operating conditions (PVT) include

- manufacturing process
  - worst case: setup-time violations
  - best case: hold-time violations
- operating conditions: voltage and temperature

I/O port attributes contain

- drive strength of input port
- capacitive loading of output port
- design rule constraints: fanin, fanout

Statistical wire-load model provides a wire-load model for processing the pre-layout static timing analysis.

Design Constraints

clock signal specification

- period, duty cycle
- transition time, skew

Delay specifications

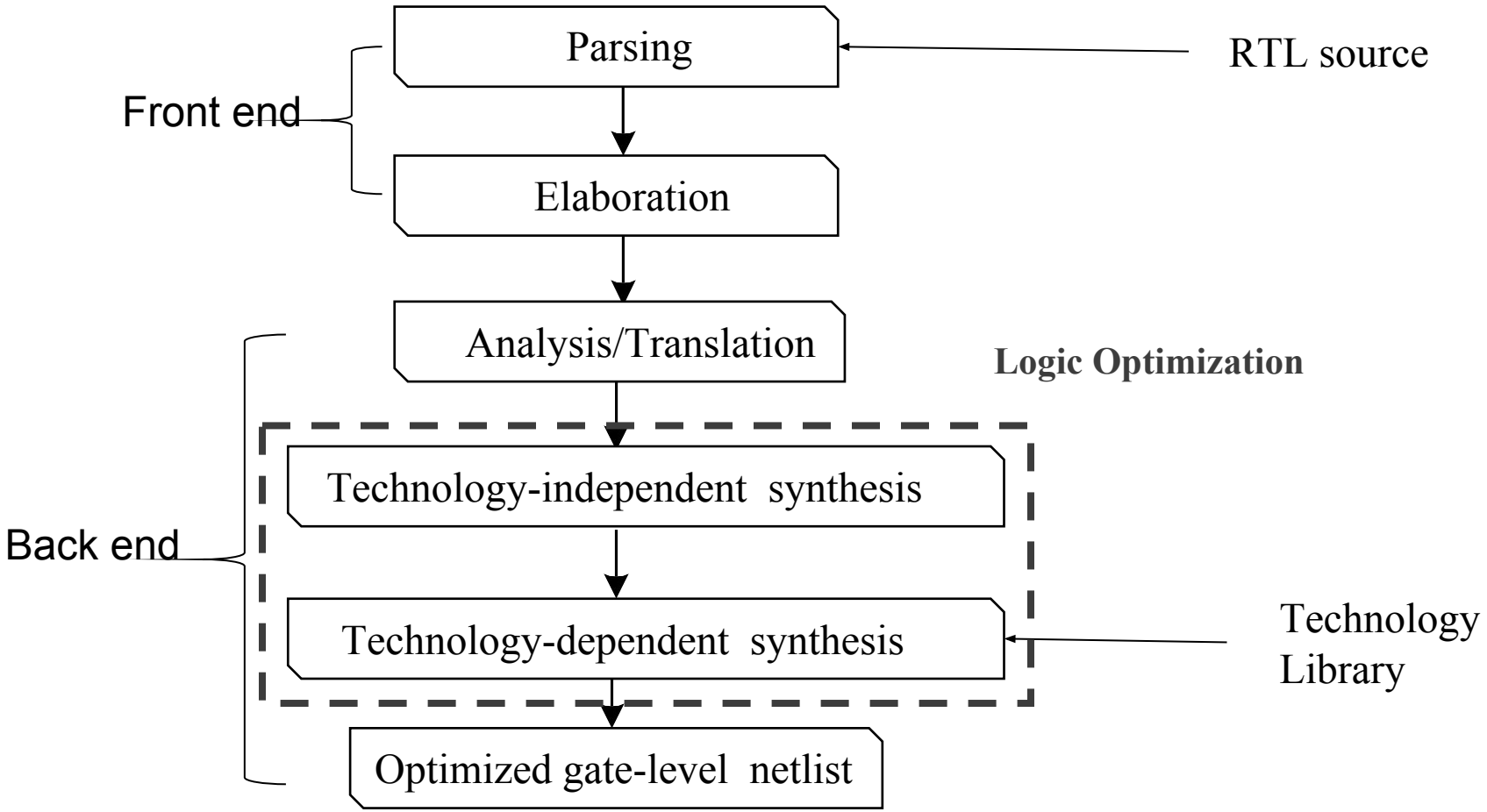
- input delay, output delay
- maximum, minimum delay for combinational circuits

Timing exception

- false path: instruct the synthesis to ignore a particular path for timing optimization
- multicycle path: inform the synthesis tool regarding the number of clock cycles that a particular path requires to reach its endpoint

Path grouping: bundle together critical paths in calculating a cost function

The Architecture of Logic Synthesis Tools



Logic Synthesis Tools: Front end

Parsing phase

- checks the syntax of the source code
- creates internal components

Elaboration phase (to construct a complete description of the input circuit)

- connects the internal components
- unrolls loops
- expands generate-loops
- sets up parameters passing for tasks and functions
- and so on

Logic Synthesis Tools: Back end

Analysis/translation prepares for technology-independent logic synthesis.

- managing the design hierarchy
- extracting finite-state machine (FSM)
- exploring resource sharing and so on.

Logic optimization creates a new gate network that computes the functions specified by a set of Boolean functions, one per primary output.

Netlist generation generates a gate-level netlist.

cadence

Logic Optimization

Major concerns when synthesizing a logic gate network:

- functional metric: such as fan-in, fan-out, and others.
- non-functional metrics: such as area, power, and delay.

Two phases of logic synthesis:

- technology-independent logic optimization
- technology-dependent logic optimization

The process of translating from a technology-independent to a technology-dependent gate network is called **library binding.**

Technology-Independent Logic Optimization

Technology-independent logic synthesis

- Simplification rewrites a single function in the network to the literals of that network.
- Restructuring the network creates new function nodes that can be used as common factors and collapses sections of the network into a single node.
- Restructuring delay changes the factorization of a subnetwork to reduce the number of function nodes through which delay-critical signal must pass.

Technology-Independent Logic Optimization

An example of simplification could be by expressing the network with newly created functions and variables.

For example:

$$S = xyz + ab \quad \text{and} \quad T = xyz + ac$$

the new function is denoted by F = xyz therefore:

$$S = F + ab$$

$$T = F + ac$$

cadence

Technology-Independent Logic Optimization

An example of the restructuring the network is when the divisor of the logical function is replaced by a new literal.

For example:

$S = xyz + xyw$ let u = xy and v = z + w

then we will have:

$S = uv + ab$

Technology-Independent Logic Optimization

An example of the restructuring delay is by factorization which take place following three different steps: 1) generating the common factors, 2) selecting the factors to be substituted in the network, and then 3) reconstructing the network with the new factors.

For example:

$$S = xyz + xyw + ab,$$

Can be factorized into:

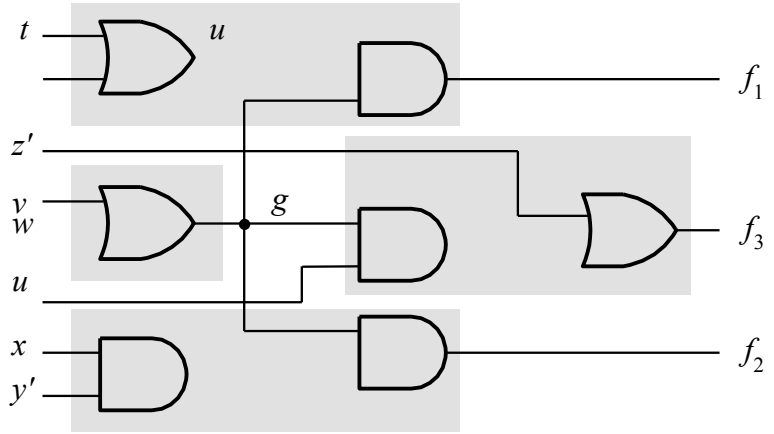
$$S = xy(z+w) + ab$$

cadence

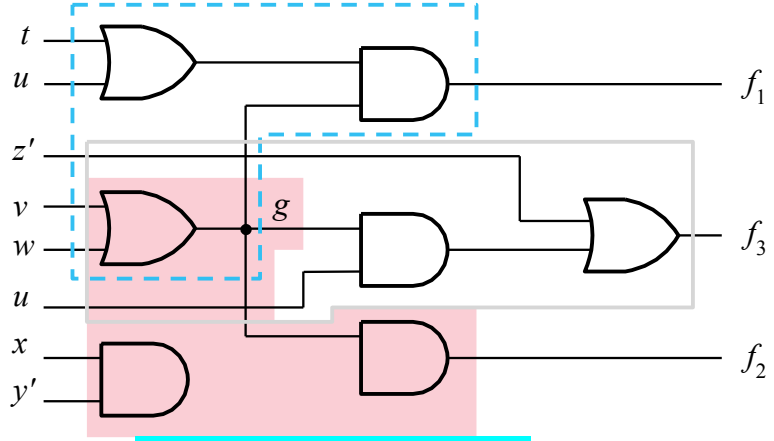
Technology Mapping

A two-step approach

- The network is decomposed into nodes with no nodes more than k inputs, where k is determined by the fan-in of each LUT.
- The number of nodes is reduced by combining some of them taking into account the special features of LUTs.



- Mapping to 4 LUTs



- Mapping to only 3 LUTs

Technology-Dependent Logic Optimization

This task involves selecting the most cost-effective combination of primitive logic elements from a given library to cover a Boolean network.

The optimization process aims to minimize both the area and delay of the system.

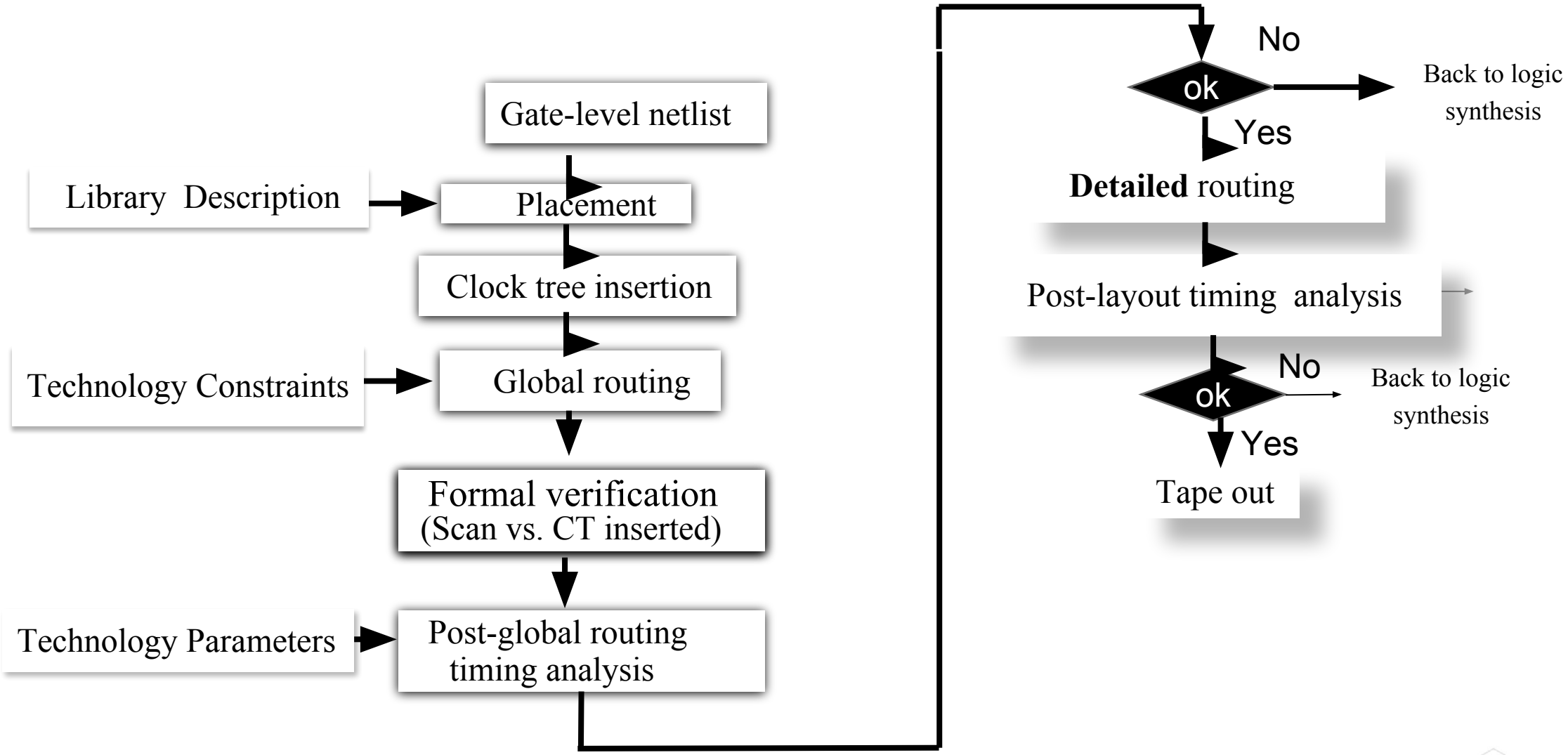
Summary on the Synthesis-Tool Tasks

Synthesis tools at least perform the following critical tasks:

- Detect and eliminate redundant logic
- Detect combinational feedback loops
- Exploit don’t-care conditions
- Detect unused states
- Detect and collapse equivalent states
- Make state assignments
- Synthesize optimal, multilevel logic subject to constraints.

cadence

The Physical Synthesis Flow



Importance of Physical Synthesis

These same challenges ripple back from block-level synthesis to “unit-level” synthesis environments run by individual logic designers, often resulting in almost no correlation between what an RTL designer sees at the unit level and what the physical design team sees in P&R.

cadence

Importance of Physical Synthesis

As correlation from P&R to unit-level synthesis degrades, more and more iterations are required between unit-level synthesis, block-level synthesis, and P&R, adding yet more pressure to achieve a fast turnaround time of each of these tasks by keeping block sizes small. A new generation of synthesis tool is needed to close this SoC "design productivity gap" one that is:

Cadence has developed a next-generation logical and physical synthesis tool, the GenusTM Synthesis Solution, that is architected from the ground up to comprehensively address the design productivity gap.

cadence

Physical Design using Genus™ Synthesis tool

The solution can scale its capacity to well beyond 10 million instances flat. It also delivers tight timing and wire length correlation to within 5% of the place and route.

cadence

Physical Synthesis Flows

Modern synthesis tools like Genus offer three physical-related flows. They provide increasing accuracy in predicting the wire lengths.

The simple PLE flow uses technology information and cell areas from the LEF libraries instead of from the synthesis technology libraries. The PLE flow uses parasitic resistance and capacitance values from the LEF libraries or the capacitance tables (if available) when estimating the wire lengths.

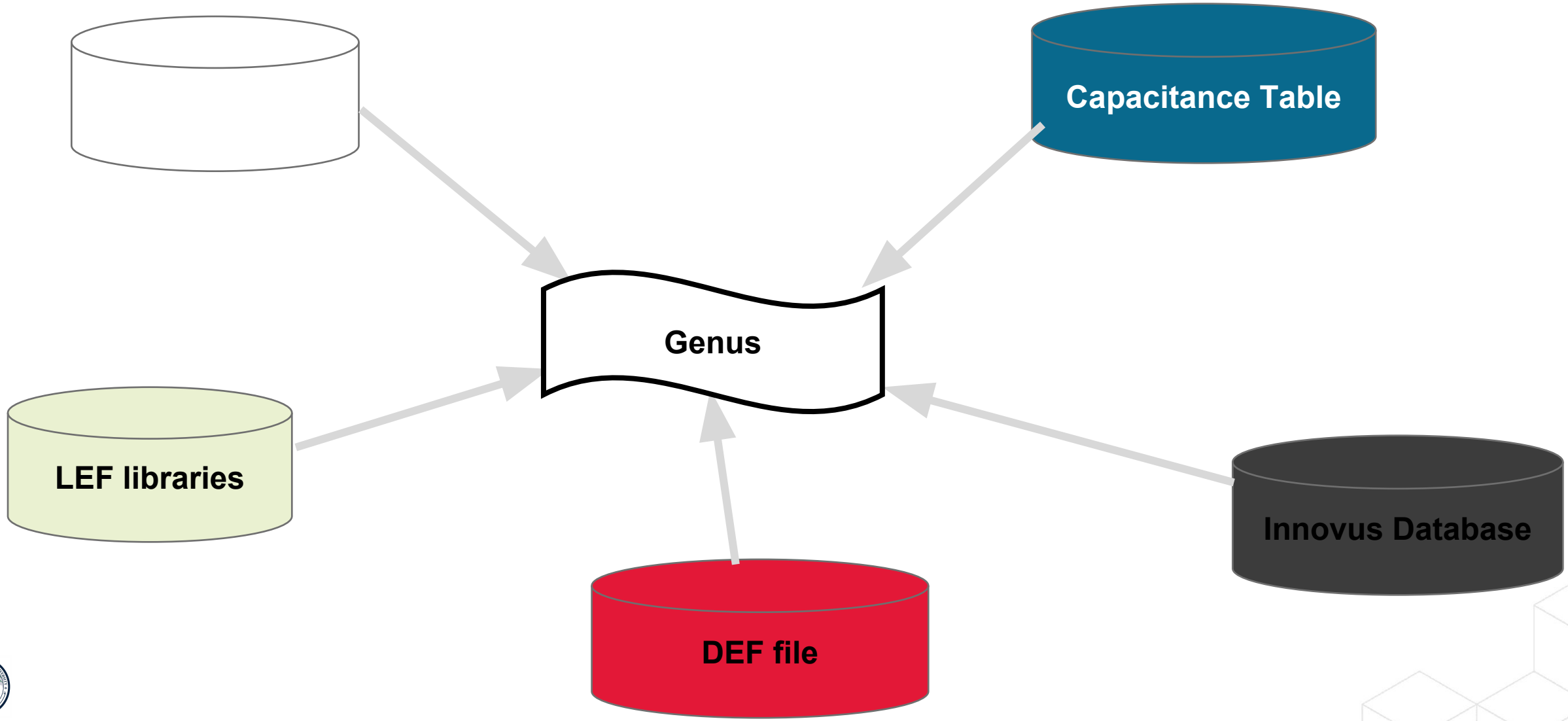
cadence

Physical Synthesis Flows

The Genus-Physical flow uses in addition a complete placement and considers congestion and legal placement as a cost function during the RTL-to-gates phase, to create a better netlist. This flow requires access to the Innovus Place & Route tool

cadence

Physical Information Files



Part B: Design Rule Constraints

Constraint Definition

Performs logic and gate-level synthesis and optimization on the design.

Optimization is controlled by user-specified constraints:

Obtain the smallest possible circuit

Fastest design

Any other design requirement

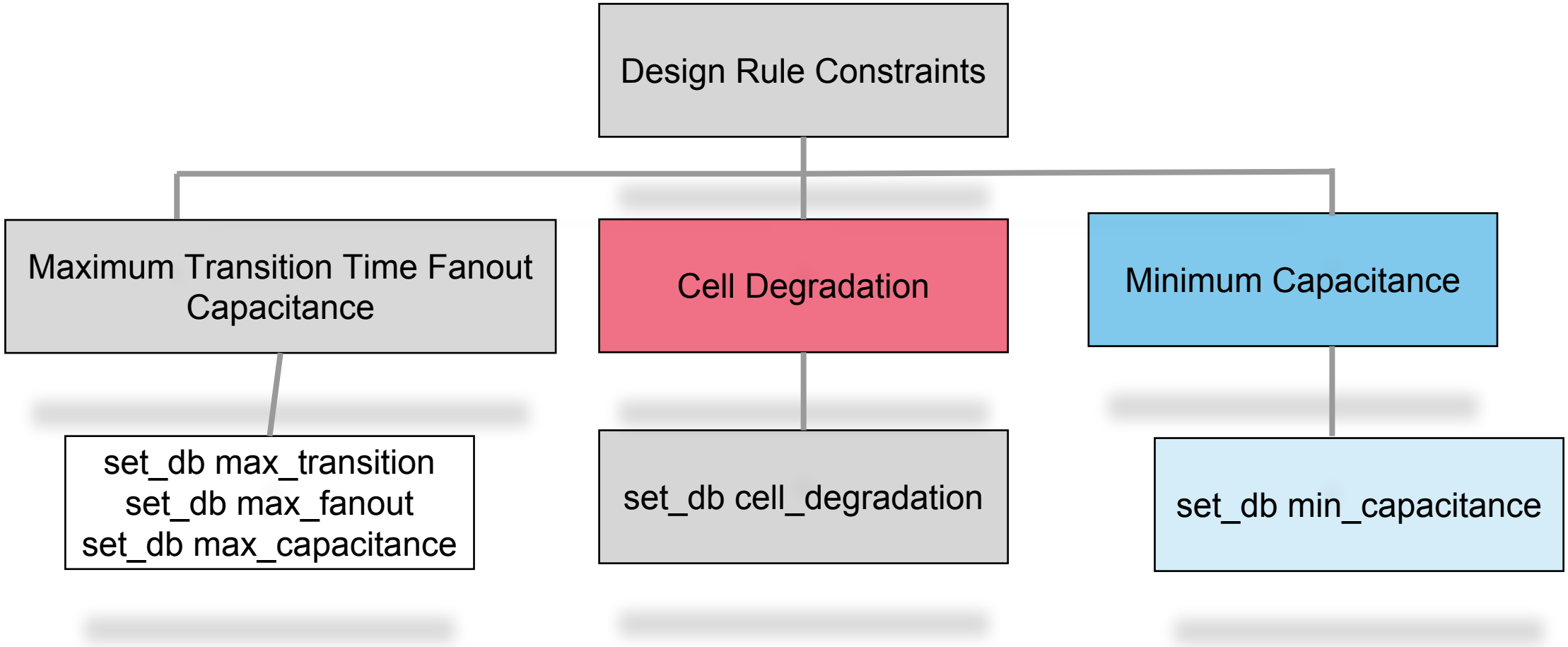
The constraints describe:

Goals for the optimization process

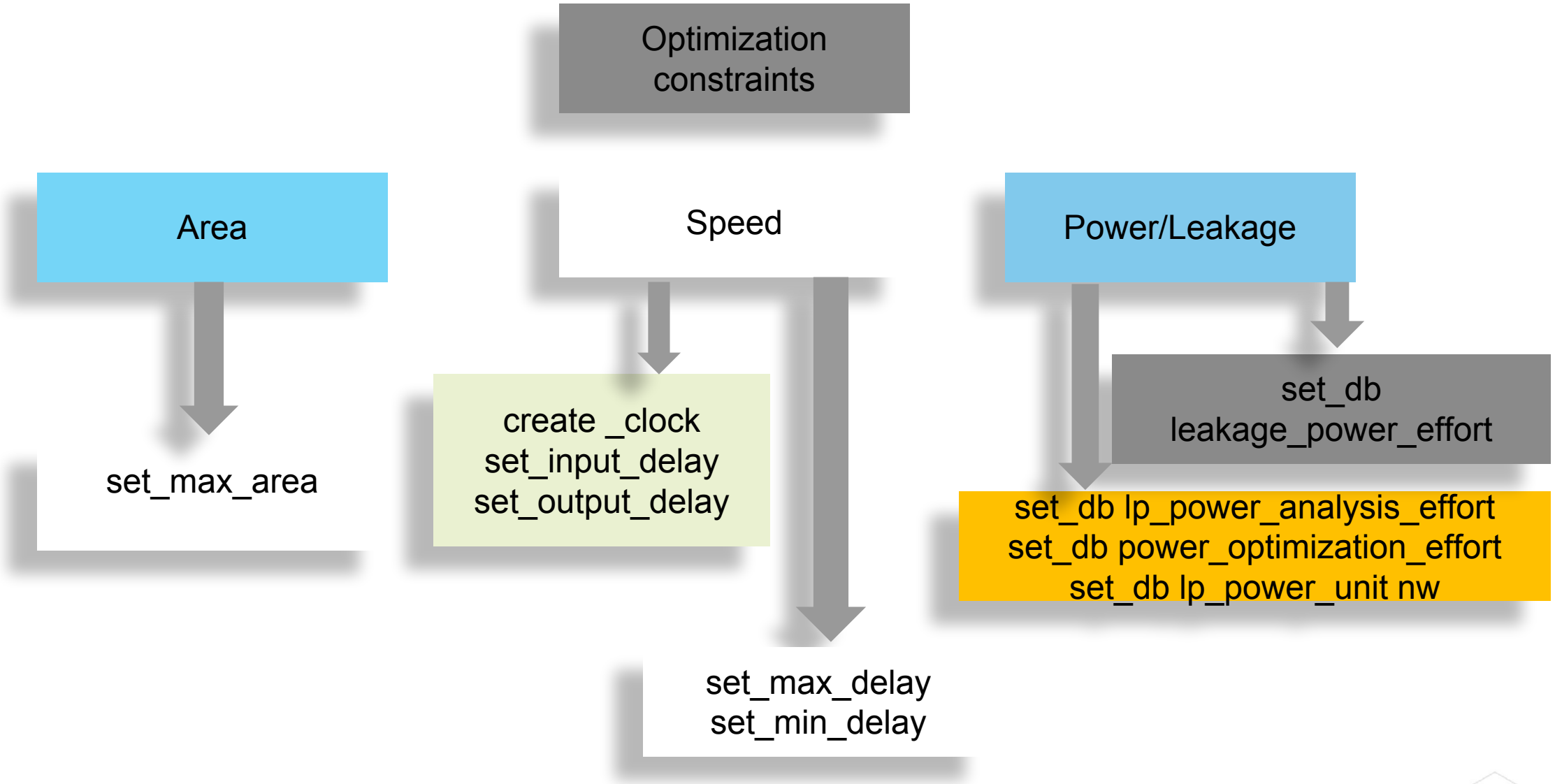
Force specified outputs to meet timing requirements

Value for components area and speed used during synthesis and optimization are obtained from manufacturer libraries.

Major Design Rule Constraints



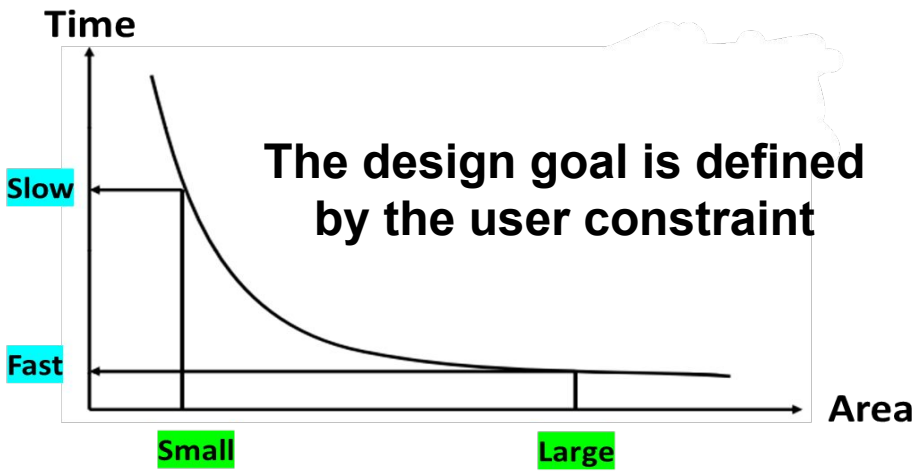
Major Design Optimization Constraints



Power vs Area

Power

- Power optimization is not enabled by default.
- Optimizes power constraints for leakage power and dynamic power set by the attributes
- max_leakage_power & max_dynamic_power.



Area

- smallest design that satisfies the timing constraint by default
- set_max_area
- logic in the non-critical paths is automatically downsized to save area

Part C: Before your Start – Prepare your Script

Scripting

Scripting is the most efficient way of automating the tasks that are performed with any tool.

To support scripting at both a basic and advanced level, Genus uses the standard scripting language, Tool Control Language (TCL).

In most cases, a Genus script consists of a series of Genus commands listed in a file, in the same format that is used interactively.

Script Example

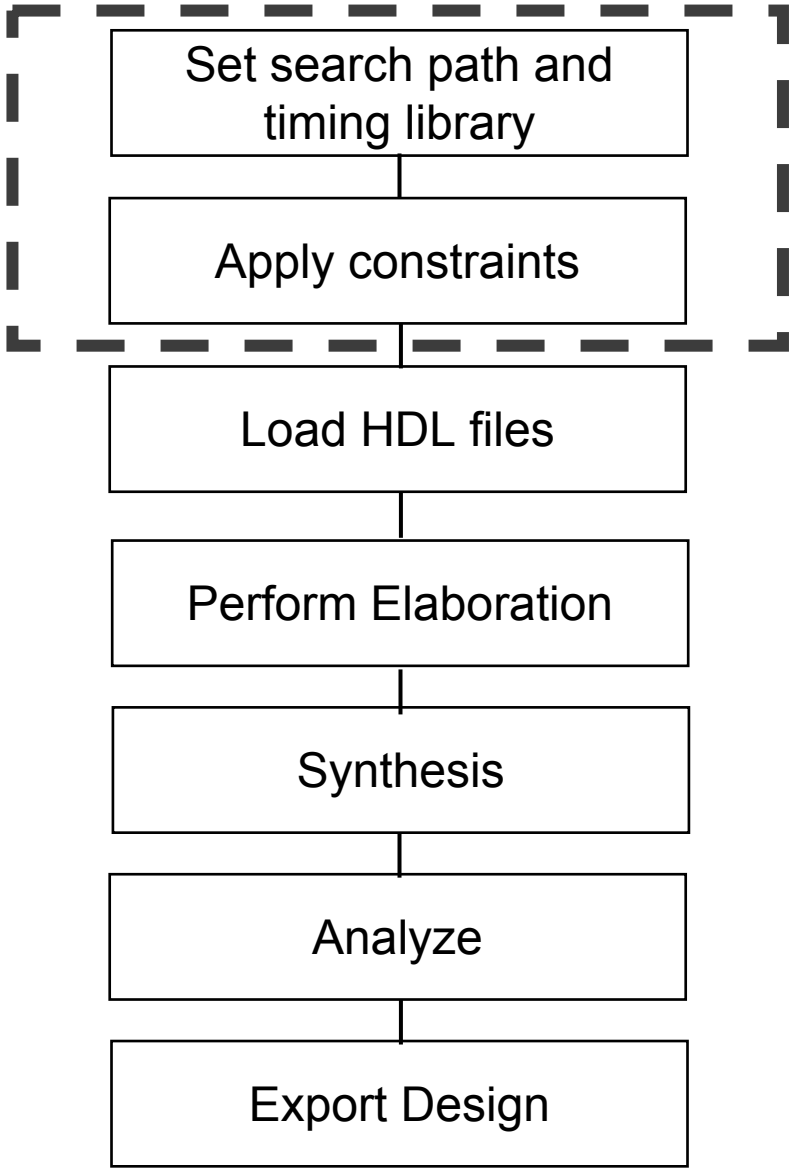
```
#-------------------------------------------------------------------------
# Info and path setup
#-------------------------------------------------------------------------
if {[file exists /proc/cpuinfo]} {
   sh grep "model name" /proc/cpuinfo
   sh grep "cpu MHz"    /proc/cpuinfo
}
puts "Hostname : [info hostname]"
set DESIGN "clock_divider"
set DATE [clock format [clock seconds] -format "%b%d-%T"]
set OUTPUTS_PATH ./results
set REPORTS_PATH ./reports
set_db init_hdl_search_path ../rtl/
set_db lib_search_path ../pdk/
#-------------------------------------------------------------------------
# General Attributes
#-------------------------------------------------------------------------
set_db information_level 9
set_db lp_insert_clock_gating true
set_db hdl_language vhdl
set_db hdl_vhdl_read_version 1993
#-------------------------------------------------------------------------
```

```
#Power root attributes
#-------------------------------------------------------------------------
set_db lp_power_analysis_effort high
set_db lp_power_unit mW
set_db power_optimization_effort medium
set_db leakage_power_effort low
#-------------------------------------------------------------------------
# Library setup
#-------------------------------------------------------------------------
set_db library {
../pdk/xh018/diglibs/D_CELLS_JIHD/v4_1/liberty_LPMOS/v4_1_1/PVT_1_80V_range/D_CELLS_JIHD_LPMOS_slo
w_1_62V_125C.lib \

../pdk/xh018/diglibs/IO_CELLS_F3V/v2_1/liberty_UPF_LPMOS/v2_1_0/PVT_1_80V_3_30V_range/IO_CELLS_F3V_
LPMOS_UPF_slow_1_62V_3_00V_125C.lib }
 set_db lef_library { ../pdk/xh018/cadence/v8_0/techLEF/v8_0_1_1/xh018_xx51_HD_MET5_METMID.lef \
                      ../pdk/xh018/diglibs/D_CELLS_JIHD/v4_1/LEF/v4_1_1/xh018_D_CELLS_JIHD.lef \

../pdk/xh018/diglibs/D_CELLS_JIHD/v4_1/LEF/v4_1_1/xh018_xx51_MET5_METMID_D_CELLS_JIHD_mprobe.lef \

../pdk/xh018/diglibs/IO_CELLS_F3V/v2_1/LEF/v2_1_1/xh018_xx51_MET5_METMID_IO_CELLS_F3V.lef \

../pdk/xh018/spram/XSPRAMLP_128X16_M8P/v4_0_2/xh018_1151_LPMOS_MET5_METMID/LEF/XSPRAMLP_12
8X16_M8P.lef }
set_db cap_table_file { ../pdk/xh018/cadence/v8_0/capTbl/v8_0_1/xh018_xx51_MET5_METMID_typ.capTbl }
#_____________________________________________________________________________
```

```
#clock
#-------------------------------------------------------------------------
create_clock -name clk -period 10 -waveform {0 5} [get_ports "clk"]
set_clock_transition -rise 0.1 [get_clocks "clk"]
set_clock_transition -fall 0.1 [get_clocks "clk"]
set_clock_uncertainty 0.01 [get_ports "clk"]
set_input_delay -max 1.0 [get_ports "reset"] -clock [get_clocks "clk"]
set_output_delay -max 1.0 [get_ports "clock_out"] -clock [get_clocks "clk"]
#-------------------------------------------------------------------------
# Read design
#-------------------------------------------------------------------------
read_hdl -vhdl clock_divider.vhd
elaborate ${DESIGN}
syn_gen
syn_map
syn_opt
write_hdl > results/clock_divider.vhd
write_hdl > clock_divider_netlist.v
write_sdc > clock_divider_sdc.sdc
syn_opt -physical
gui_show
```

Strategy For The Script

- Path and information
- General Attribute
- Power Attribute
- Library attribute
- Clock



Information and Path Setup

- First, give the location for the code as the above figure
  set_db init_hdl_search <destination>
- Then search for the library path
  set_db lib_search_path <destination>

```
# Info and path setup
#-------------------------------------------------------

if {[file exists /proc/cpuinfo]} {
    sh grep "model name" /proc/cpuinfo
    sh grep "cpu MHz"    /proc/cpuinfo
}

puts "Hostname : [info hostname]"

set DESIGN "clock_divider"
set DATE [clock format [clock seconds] -format "%b%d-%T"]
set OUTPUTS_PATH ./results
set REPORTS_PATH ./reports
set_db init_hdl_search_path ../rtl/
set_db lib_search_path ../pdk/
```

General Attribute

```
# General Attributes
#--------------------------------------
set_db information_level 9
set_db lp_insert_clock_gating true
set_db hdl_language vhdl
set_db hdl_vhdl_read_version 1993
```

Define how detailed information you want to see in the terminal or log files (normally you may have enough information by 5 to 7):

set_db information_level 9

Controls whether to generate clock-gating logic by using basic libcells if no usable integrated clock-gating cells are available in the library

set_db lp_insert_clock_gating true

cadence

General Attribute

Specifies the VHDL version to be used when reading VHDL designs

set_db hdl_vhdl_read_version 1993

If your design consists of just one type of HDL file in “Configuring the Environment – native parameters”, set your flow default hdl language:

(hdl_language {v2001 | v1995 | vhdl | sv} Default: v2001)

set_db hdl_language vhdl

Power Attributes

Specify attributes for the power optimization

```
#-------------------------------------------------------
#Power root attributes
#-------------------------------------------------------
set_db lp_power_analysis_effort high
set_db lp_power_unit mW
set_db power_optimization_effort medium
set_db leakage_power_effort low
```

cadence

Library Setup

Define the attributes which specify the various libraries and LEFs for following the flow in the design:

Notice that these libraries are to be used as the specific target library from the technology to which you decide to map

1. set_db library { ../LIB/lib1.lib \ ../LIB/lib2.lib} /
2. set_db lef_library { ../LEF/lef1.lef \ ../LEF/lef2.lef} /

Specify the cap tables:

set_db cap_table_file capfile

```
# Library setup
#--------------------------------------------------------------------------
set_db library { ../pdk/xh018/diglibs/D_CELLS_JIHD/v4_1/liberty_LPMOS/v4_1_1/PVT_1_80V_range/D_CELLS_JIHD_LPMOS_slow_1_62V_125C.lib \
                 ../pdk/xh018/diglibs/IO_CELLS_F3V/v2_1/liberty_UPF_LPMOS/v2_1_0/PVT_1_80V_3_30V_range/
IO_CELLS_F3V_LPMOS_UPF_slow_1_62V_3_00V_125C.lib }


set_db lef_library { ../pdk/xh018/cadence/v8_0/techLEF/v8_0_1_1/xh018_xx51_HD_MET5_METMID.lef \
                     ../pdk/xh018/diglibs/D_CELLS_JIHD/v4_1/LEF/v4_1_1/xh018_D_CELLS_JIHD.lef \
                     ../pdk/xh018/diglibs/D_CELLS_JIHD/v4_1/LEF/v4_1_1/xh018_xx51_MET5_METMID_D_CELLS_JIHD_mprobe.lef \
                     ../pdk/xh018/diglibs/IO_CELLS_F3V/v2_1/LEF/v2_1_1/xh018_xx51_MET5_METMID_IO_CELLS_F3V.lef \
                     ../pdk/xh018/spram/XSPRAMLP_128X16_M8P/v4_0_2/xh018_1151_LPMOS_MET5_METMID/LEF/XSPRAMLP_128X16_M8P.lef }



set_db cap_table_file { ../pdk/xh018/cadence/v8_0/capTbl/v8_0_1/xh018_xx51_MET5_METMID_typ.capTbl }
```

Setting Effort Levels Prior to Synthesis

You can change the default effort levels for synthesis by using attributes.

Generic Synthesis (*syn_generic* command) effort level	`set_db syn_generic_effort <low|medium|high|express>`
The optimization effort for the mapping stage (*syn_map* command)	`set_db syn_map_effort <low|medium|high|express>`
The optimization effort to use for incremental optimization (*syn_opt* command)	`set_db syn_global_effort <none | low | medium | high | express>`
To specify the global effort for all synthesis commands, use	`set_db syn_global_effort <none | low | medium | high | express>`

- Default effort is:
  - *Medium* for generic.
  - *High* for mapping and optimization.
  - *None* for global.
- Use the option *express* to enable express flow for both logical and physical synthesis.
- The express flow enables early feasibility analysis with much faster runtimes and reasonable quality of results.

Timing Constraints

To accurately set up timing constraints, you need to specify the following:

1. Clock
2. I/O Timing requirements
3. Combinational path delay requirements
4. Timing exceptions

Clocks: To define a clock command is used, **and the create_generated_clock** is used command is used to define an internally generated clock. A few specifications of the clock include clock source, clock period, duty cycle, clock name, etc.

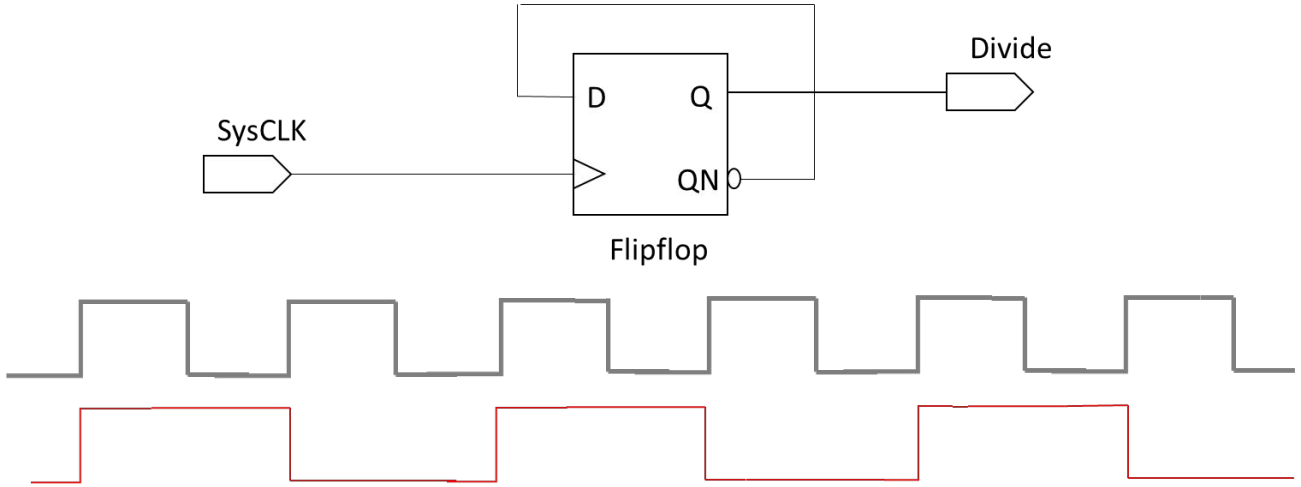
cadence

Building a Clock

Create a clock and divide by 2 generated clock example is given bellow

create_clock –name SYSCLK_NAME –period 2 -waveform {0 1} {get_ports SYSCLK}

create_generated_clock –name DIVIDE -source {get_ports SYSCLK} divide_by 2 {get_pinsFF1/Q}



Clock commands

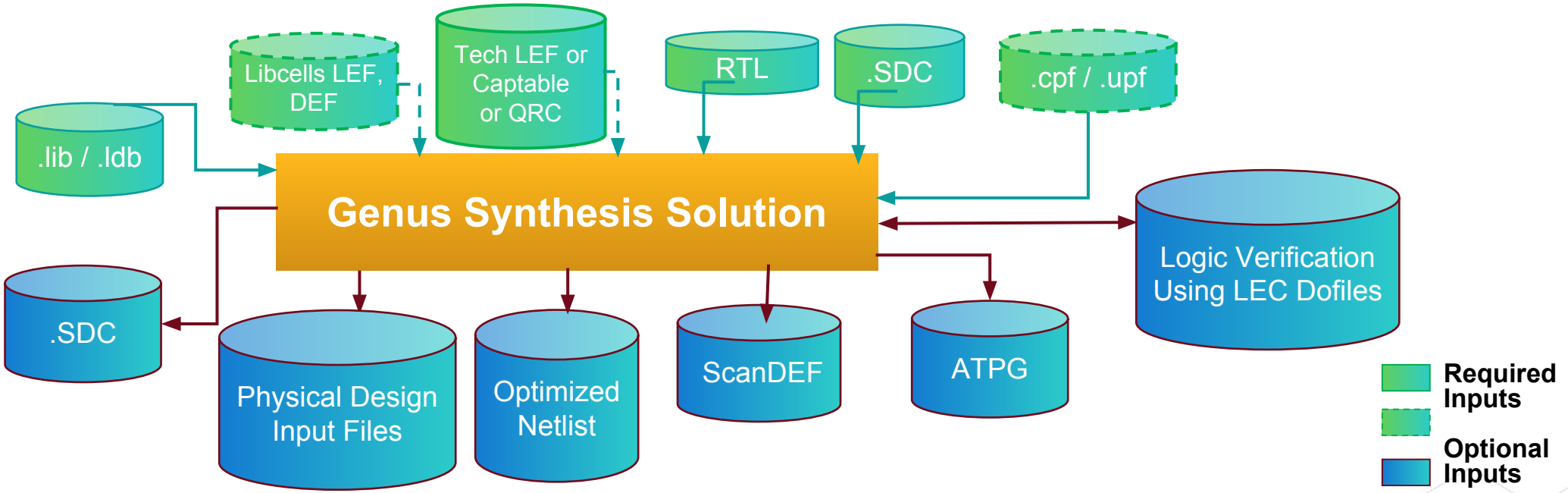
Specify clock signal

```
#clock
#-------------------------------------------------------------------------------
create_clock -name clk -period 10 -waveform {0 5} [get_ports "clk"]

set_clock_transition -rise 0.1 [get_clocks "clk"]
set_clock_transition -fall 0.1 [get_clocks "clk"]
set_clock_uncertainty 0.01 [get_ports "clk"]
set_input_delay -max 1.0 [get_ports "reset"] -clock [get_clocks "clk"]
set_output_delay -max 1.0 [get_ports "clock_out"] -clock [get_clocks "clk"]
#-------------------------------------------------------------------------------
```

Part D: Synthesize Your Design using Genus tool

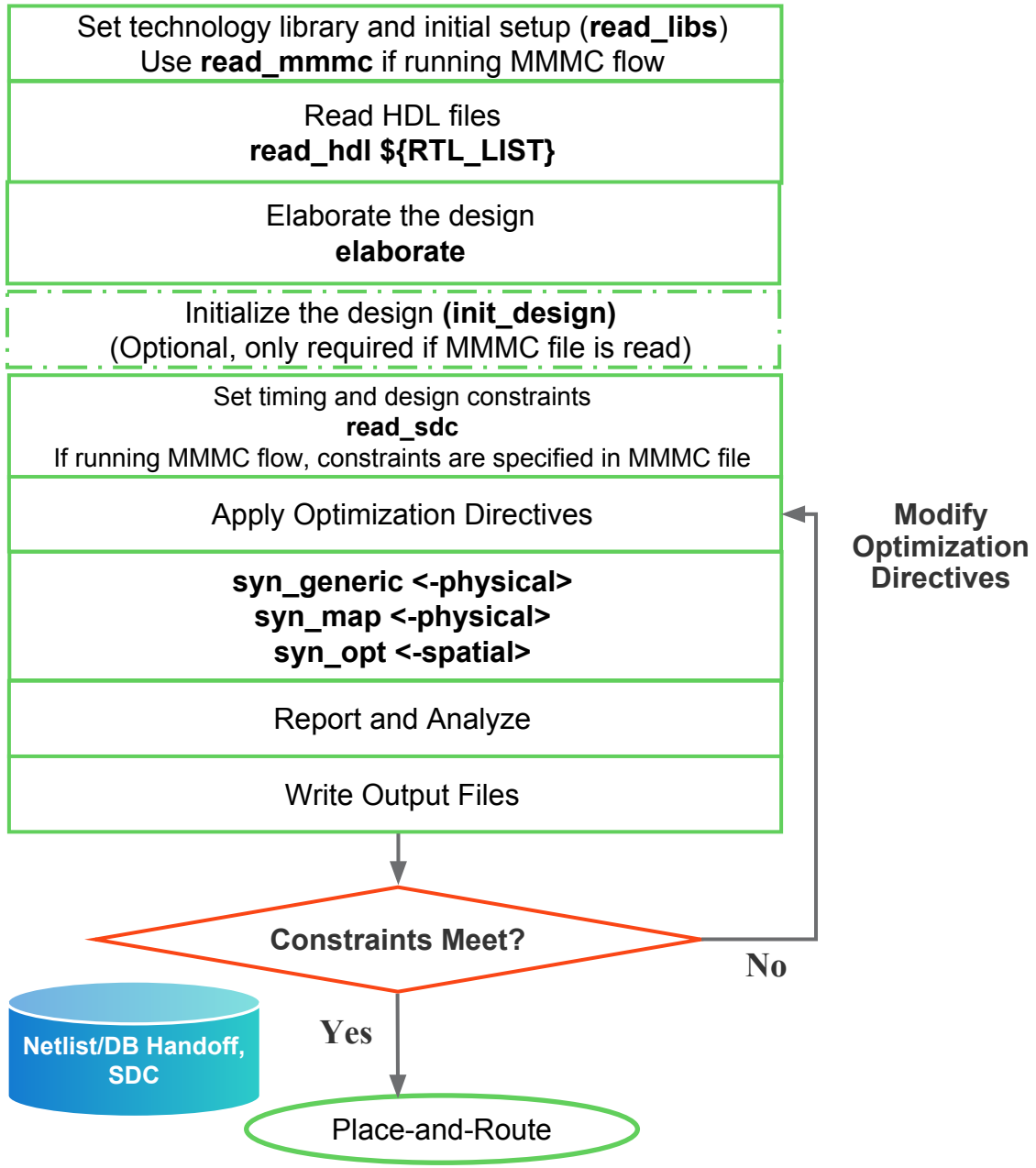
Inputs and Outputs of Genus Synthesis Solution

Inputs	Outputs
RTL: Verilog, VHDL, directives, pragmas, SystemVerilog	Optimized netlist
Constraints: .sdc	LEC dofile
Library: .lib or .ldb	ATPG, ScanDEF, and others
Power Intent: CPF, UPF/IEEE 1801 (Optional)	Constraints: .sdc
Physical: Captable, QRC Technology file, LEF, DEF (Optional)	Physical design input files



"Source: Cadence Customer Training Course RTLtoGDSII, Version: 5.0"

Genus Synthesis Flow



"Source: Cadence Customer Training Course RTLtoGDSII, Version: 5.0"

Part E: Design for Testability

What Is DFT?

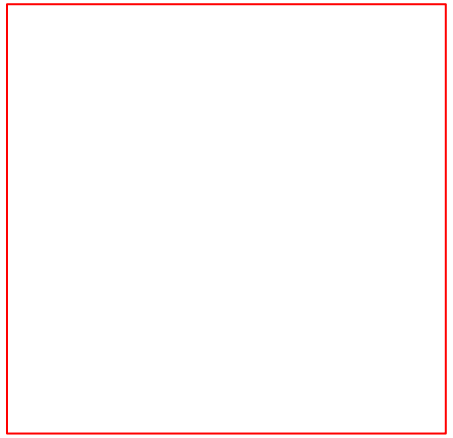
Design for Test (DFT) techniques provide measures to comprehensively test the manufactured device for quality and coverage.

Design for Testability (DFT) makes it possible to:

- Assure the detection of all faults in a circuit.
- Reduce the cost and time associated with test development.
- Reduce the execution time of performing a test on fabricated chips.

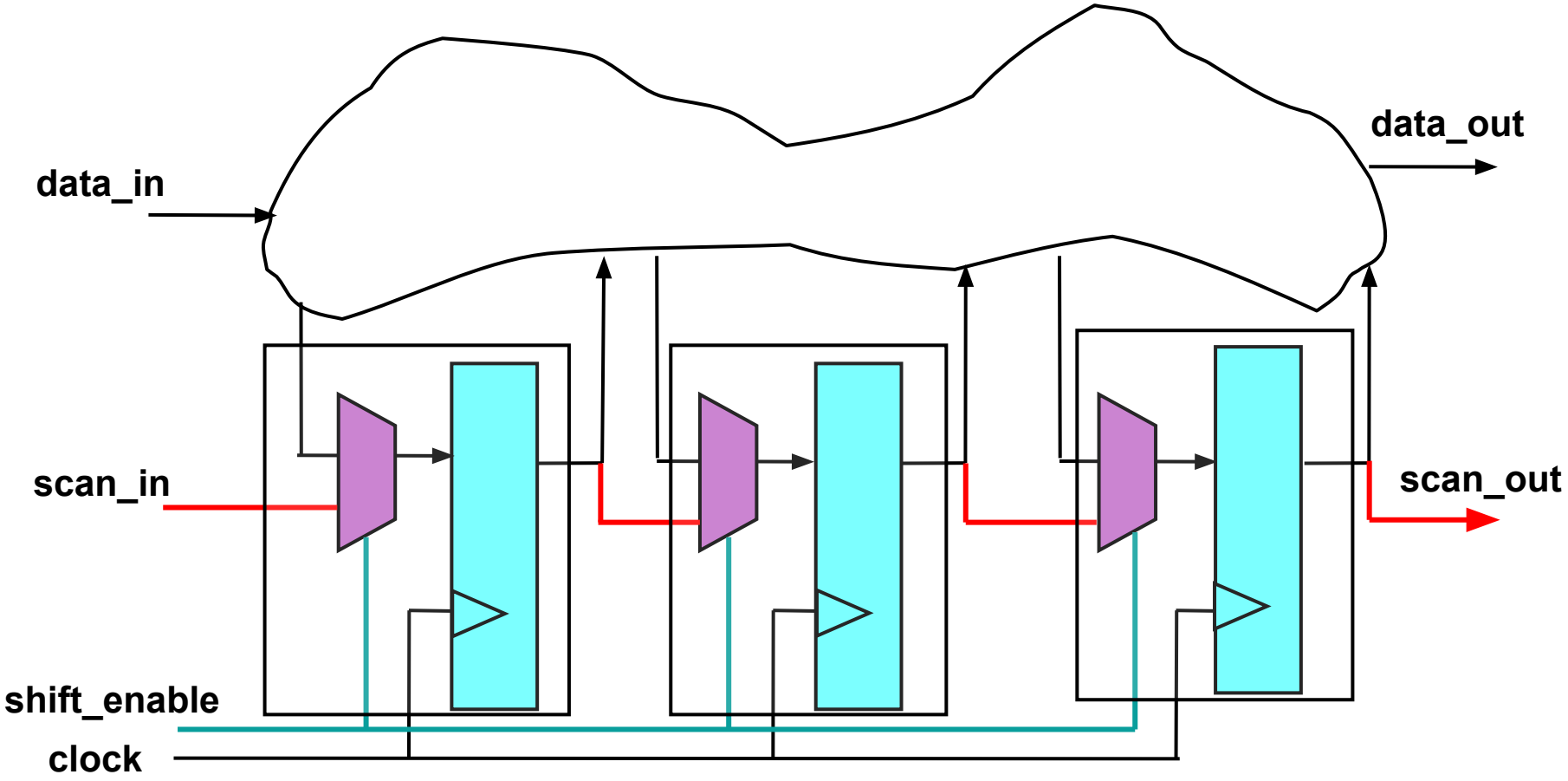
Different faults in DFT:

- Stuck-At faults
- Transition faults
- Stuck open and Shorts
- Bridging faults



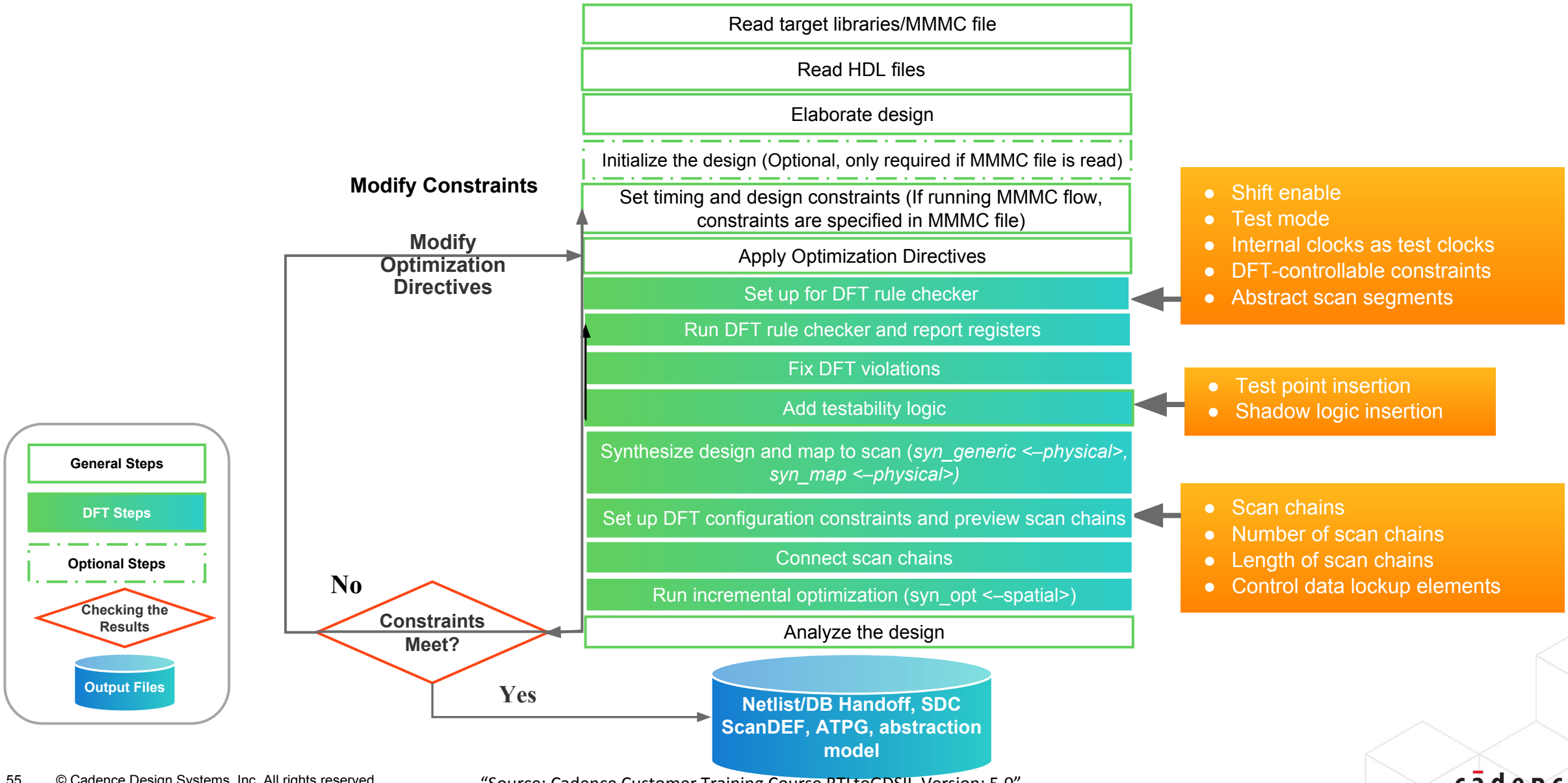
"Source: Cadence Customer Training Course RTLtoGDSII, Version: 5.0"

Design with Test Circuit



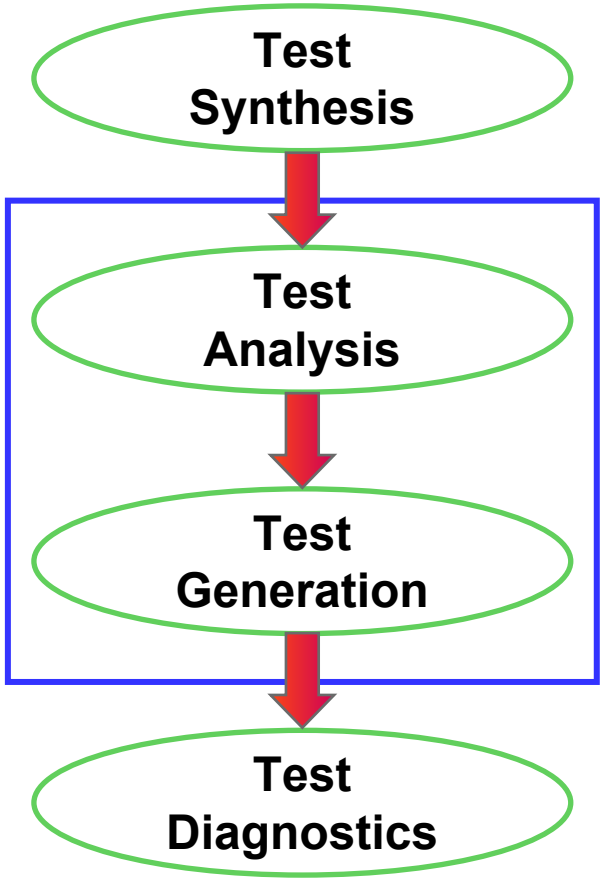
"Source: Cadence Customer Training Course RTLtoGDSII, Version: 5.0"

RTL Top-Down DFT Flow



"Source: Cadence Customer Training Course RTLtoGDSII, Version: 5.0"

Modus Test Disciplines

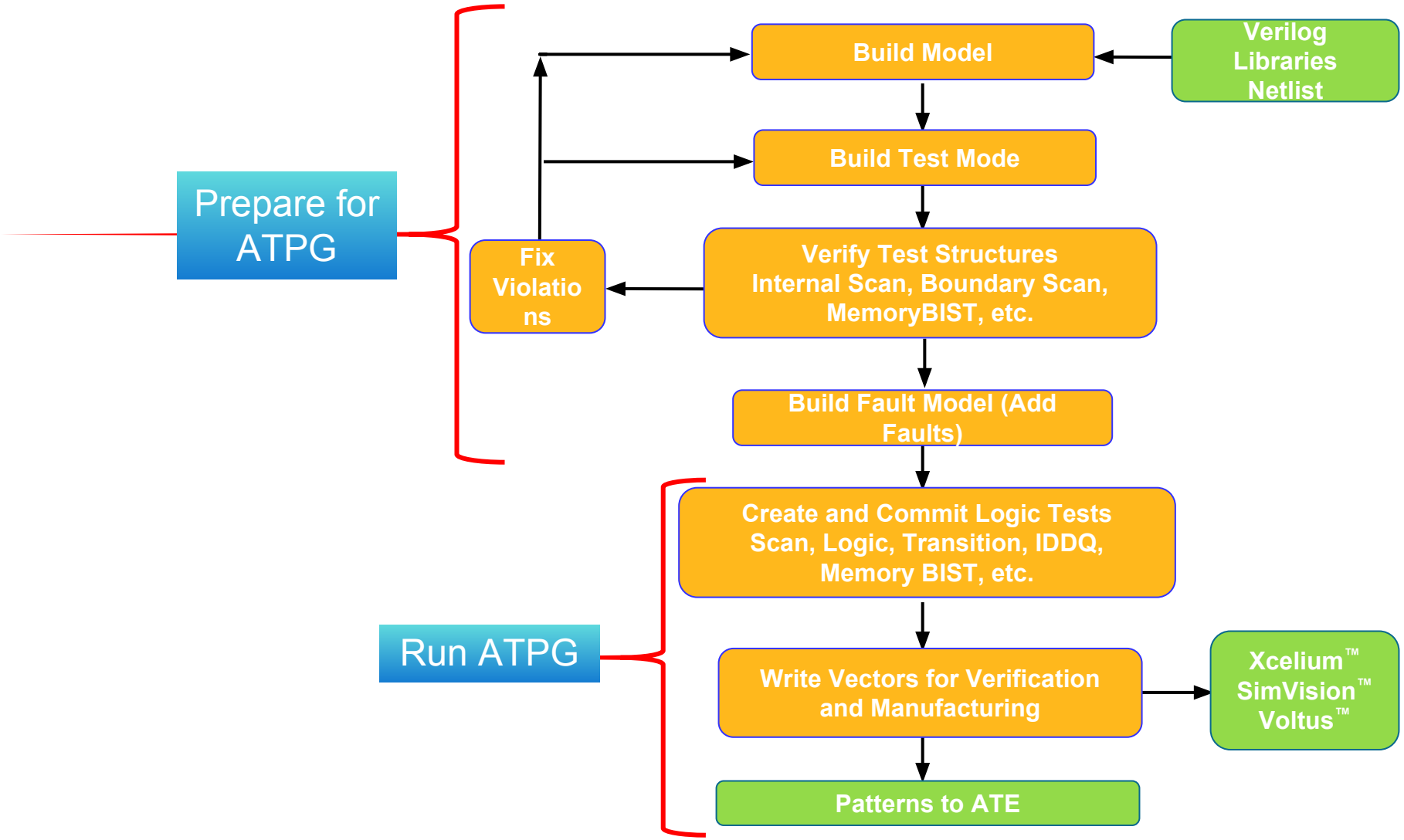


Correct-by-design test structures using Genus™:

- Verify (and optionally enforce) compliance with test design rules.
- Full range of tests for all types of digital testing.
- Diagnostics for fault isolation, failure analysis, and process monitoring.

"Source: Cadence Customer Training Course RTLtoGDSII, Version: 5.0"

Modus Test ATPG Flow



"Source: Cadence Customer Training Course RTLtoGDSII, Version: 5.0"