EE-382M: VLSI–II

Tutorial: Logic Synthesis

Mark McDermott

Introduction

- The *Design Compiler* is the core of the Synopsys synthesis software products. It includes tools that synthesis the HDL designs into optimized technology-dependent, gate level designs. It can optimize for speed, area and power.

- *Synthesis*: is the process that generates a gate-level netlist for an IC design that has been defined using a Hardware Description Language (HDL). Synthesis includes reading the HDL source code an optimizing the design from that description.

- *Optimization*: is the step in the synthesis process that attempts to implement a combination of library cells that best meet the functional timing, and area requirements of the design.

Introduction (cont)

- *Compile*: is the Design Compiler command and process that executes the optimization step. After reading in the design performing the necessary tasks, the compile command is invoked to generate a gate-level netlist for the design.

- *Libraries*: link, target and symbol libraries
  - *Link and target libraries*: define the semiconductor vendor's set for cells and related information, such as cell names, cell pin names, delay arcs, pin loading, design rules and operating conditions.
  - *Symbol library*: defines symbols for schematic and viewing the design. (Needed if GUI is to be used).

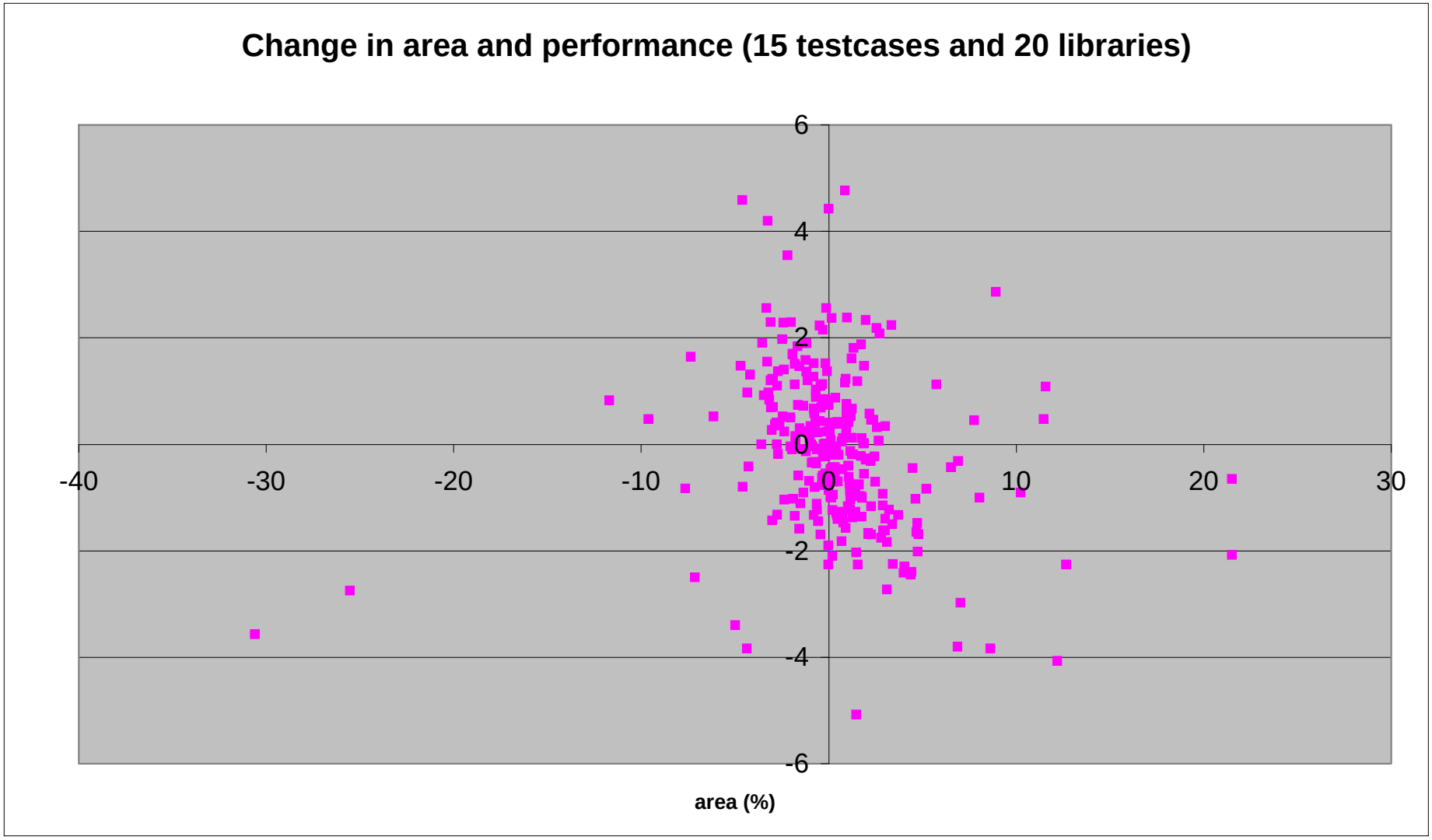
Objective Function for Synthesis

- **Minimize area**
  - in terms of literal count, cell count, register count, etc.
- **Minimize power**
  - in terms of switching activity in individual gates, deactivated circuit blocks, etc.
- **Maximize performance**
  - in terms of maximal clock frequency of synchronous systems, throughput for asynchronous systems
- **Any combination of the above**
  - combined with different weights
  - formulated as a constraint problem
    - "minimize area for a clock speed > 300MHz"
- **More global objectives**
  - feedback from layout
    - actual physical sizes, delays, placement and routing

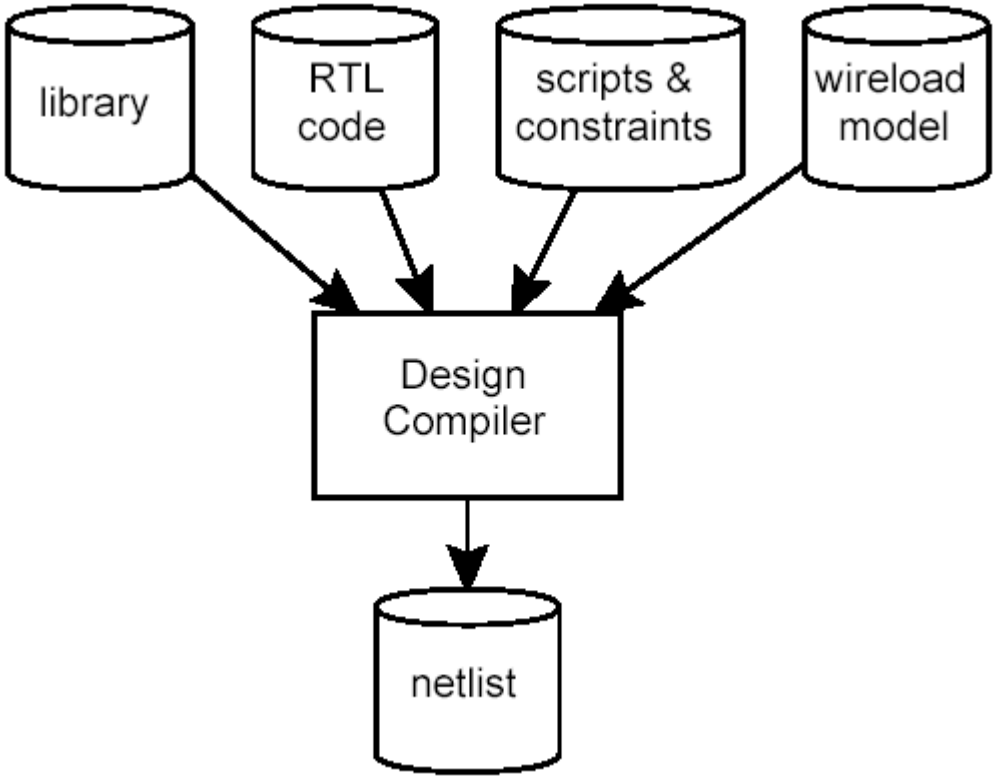
Constraints on Synthesis

- **Given implementation style:**
  - two-level implementation (PLA, CAMs)
  - multi-level logic
  - FPGAs
- **Given performance requirements**
  - minimal clock speed requirement
  - minimal latency, throughput
- **Given cell library**
  - set of cells in standard cell library
  - fan-out constraints (maximum number of gates connected to another gate)
  - cell generators

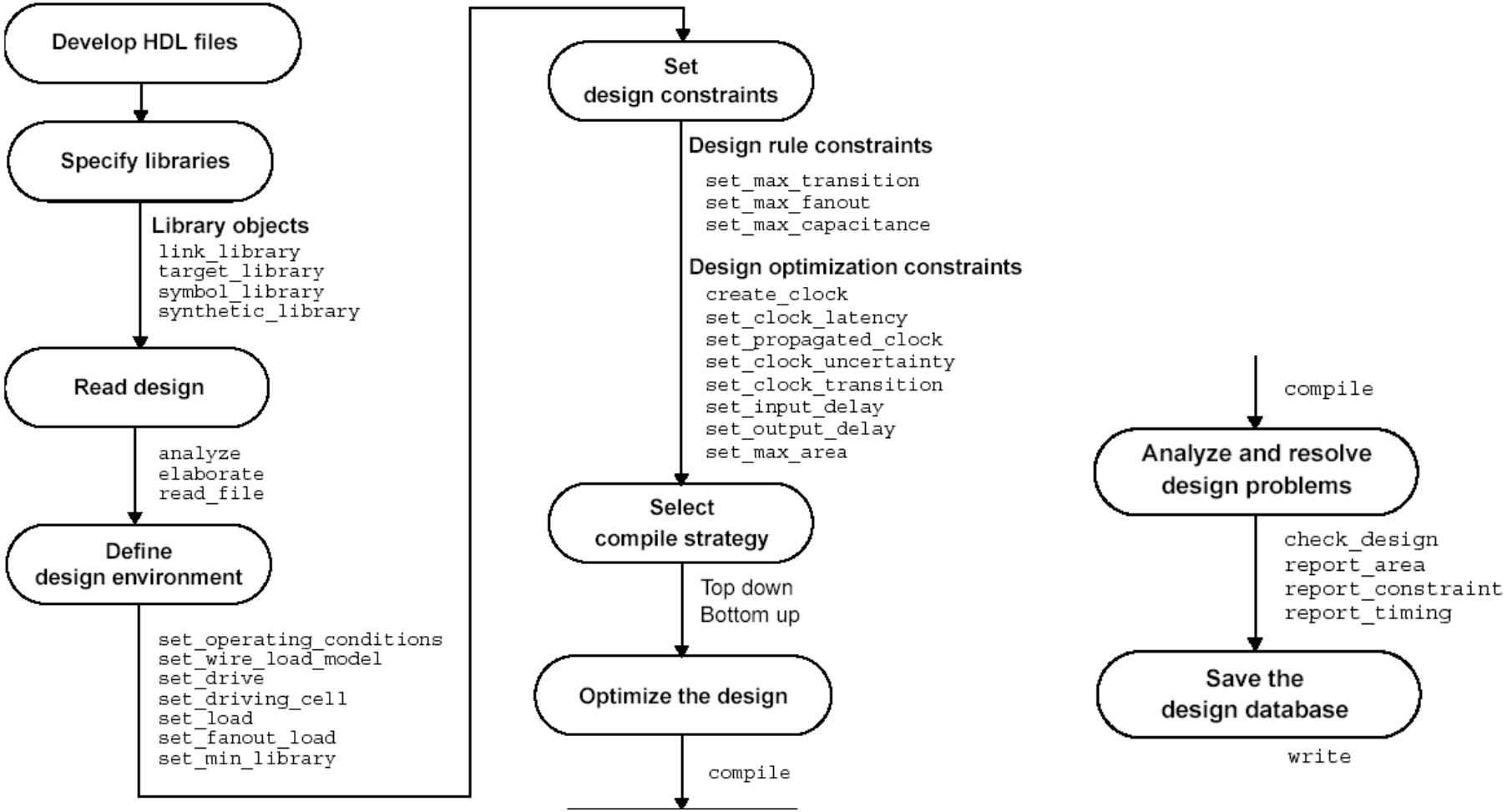
Instability of Logic Synthesis



Synthesis using Design Compiler

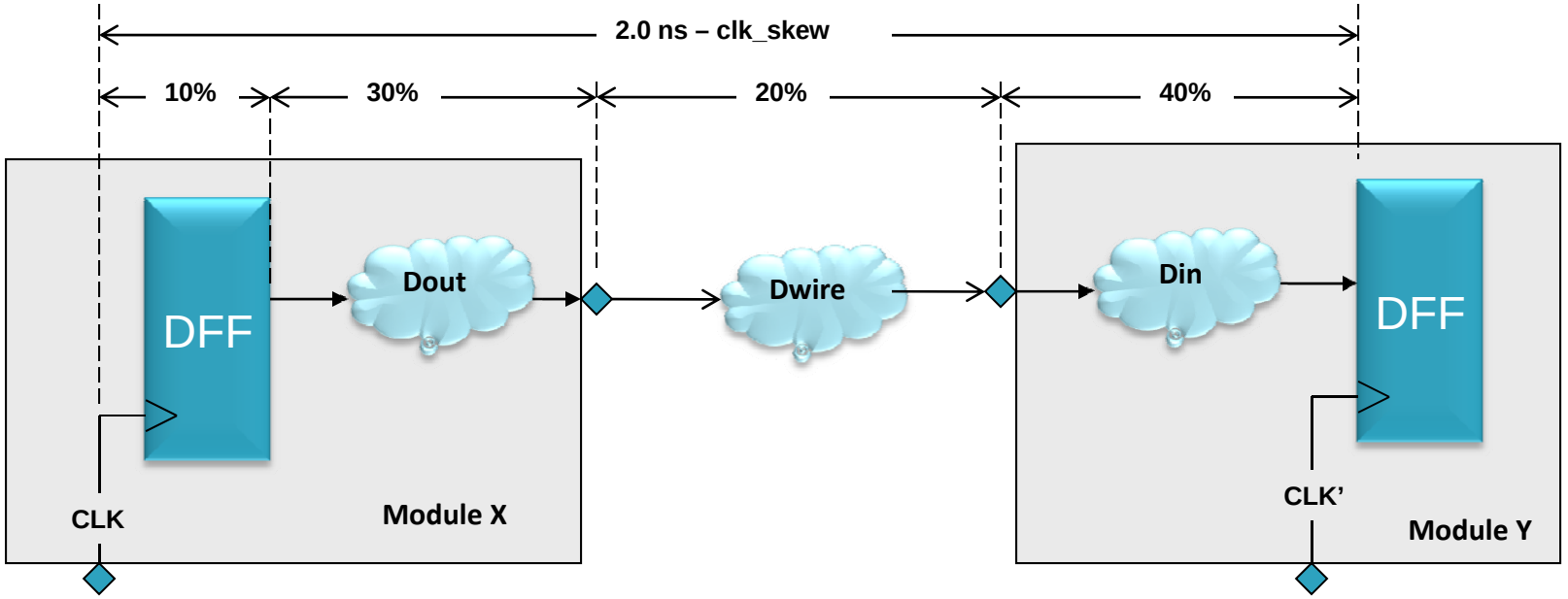


Basic Flow


Develop HDL files
Specify libraries
Library objects
link_library
target_library
symbol_library
synthetic_library
Read design
analyze
elaborate
read_file
Define
design environment
set_operating_conditions
set_wire_load_model
set_drive
set_driving_cell
set_load
set_fanout_load
set_min_library
Set
design constraints
Design rule constraints
set_max_transition
set_max_fanout
set_max_capacitance
Design optimization constraints
create_clock
set_clock_latency
set_propagated_clock
set_clock_uncertainty
set_clock_transition
set_input_delay
set_output_delay
set_max_area
Select
compile strategy
Top down
Bottom up
Optimize the design
compile
compile
Analyze and resolve
design problems
check_design
report_area
report_constraint
report_timing
Save the
design database
write

Step 1: Specifying Targets & Libraries

```
set TARGET_FREQ 0.5
set CLK_SKEW 0.125
set CCT [expr (1 / $TARGET_FREQ) - $CLK_SKEW]
set INPUT_EXT_DELAY  [expr 0.60 * $CCT]
set OUTPUT_EXT_DELAY [expr 0.60 * $CCT]
set TSMC_HOME {/home/projects/circuits/tsmc_180nm/libs_arm/cl018g/cl018g_SC/aci/sc/synopsys};
```




```
set link_library {$TSMC_HOME/typical.db} ;
set target_library {$TSMC_HOME/typical.db} ;
set synthetic_library  {} ;
set command_log_file  "./command.log";
set symbol_library
   "/home/projects/circuits/tsmc_180nm/libs_arm/cl018g/cl018g_SC/aci/sc/symbols/synopsys/tsmc18.sdb"
```

Step 2: Read in Design

```
# Set some potentially useful local variables
set myBlock or1200_if

# Simple routine to collect every source file available

set fileList [list ./verilog_or1200/or1200_defines.v
  ./verilog_or1200/timescale.v ./verilog_or1200/${myBlock}.v ]

#  Read all source files

foreach file $fileList {
    read_file -format verilog $file
}

# Identify the top-level design on which to operate
current_design $myBlock

#  Link the design with the libraries
link

#  Checks the current design for consistency
check_design
```

Step 3: Defining the Design Environment

Design Environment Commands:

create_clock	Defines the period and the waveform for the clock		*set_max_delay* *set_min_delay*	Defines maximum delay for combinational paths
set_clock_latency *set_propagated_clock* *set_clock_uncertainty*	Defines clock delay		*set_false_path*	Specifies false paths
set_input_delay	Defines timing requirements for input ports relative to the clock period		*set_multicycle_path*	Specifies multicycle paths
set_output_delay	Defines timing requirements for output ports relative to the clock period		*report_clock*	Informs about all clock sources in the design

```
# ****************************************************************
#  Create clock(s) apply block attributes and constraints
# ****************************************************************

create_clock -period $CCT [get_ports clk]
```

Step 3: Defining the Design Environment (cont)

```
# ***************************************************
#  Apply these constraints to the top-level entity
# ***************************************************
set passthrough_input_list {}
set passthrough_output_list {}

# RATs
set_input_delay -max  $INPUT_EXT_DELAY  -clock clk [remove_from_collection [all_inputs] {clk
   icpu_dat_i icpu_ack_i flushpipe rfe flushpipe icpu_tag_i if_freeze no_more_dslot icpu_err_i
   $passthrough_input_list}]

set_input_delay -max 0.9 -clock clk flushpipe
set_input_delay -max 0.55 -clock clk {icpu_ack_i no_more_dslot icpu_err_i rfe}
set_input_delay -max 0.6 -clock clk {icpu_tag_i icpu_dat_i}
set_input_delay -max 0.8 -clock clk if_freeze

# ATs
set_output_delay -max 0.35 -clock clk [remove_from_collection [all_outputs]
   $passthrough_output_list]

# Set Pass-Throughs delays
# set_max_delay 0.08 -from icpu_err_i -to if_stall


# Make first input gate a minimum-size inverter
set_isolate_ports [remove_from_collection [all_inputs] {clk flushpipe}] -driver INVERT_A –
   force
```

Step 3: Defining the Design Environment (cont)

```
# ****************************************************************
# Output Load
# ****************************************************************
#  default, driving 1 INVERT_A and 25um of Metal

set_load 0.006647 [remove_from_collection [all_outputs] {if_stall if_pc
  except_ibuserr except_itlbmiss except_immufault}]

#  this is a fanout of 2, one to the default INVERT_A
#  set_load 0.018192 if_stall

set_load 0.011545 {except_ibuserr except_itlbmiss except_immufault if_pc}

# Driving Cell Example
# set_driving_cell -lib_cell INVERT_A [remove_from_collection [all_inputs]
  flushpipe]
```

Step 4: Compile Design and Generate Reports

```
#check_design -no_warnings
# set_dont_touch block

ungroup -all
compile -map_effort high
# compile -map_effort medium
# compile -no_map

report_timing -max 10 > timing_report_if
report_constraint -all_violators > constraint_report_if

write -format verilog –hierarchy -output ${myBlock}.gate.v
```

Step 5: Optimization

- **The types of optimiztion techniques**
  - Architectural Optimization
  - Logic-level Optimization
  - Gate-level Optimization

- **Architectural Optimization: Works on the HDL description. It includes such high-level synthesis tasks as:**
  - Sharing common sub-expressions
  - Sharing resources
  - Re-ordering operators

  The output is a generic technology independent net list.

  High level synthesis tasks are based on the constraints and on the HDL coding style.

Step 5: Optimization (cont)

- **Logic Level Optimization:**
  - Works on the net list produced by Architectural Optimization.
  - Two processes are involved:
    - Structuring: Adds intermediate variables and logic structures to a design, which can result in reduced design area.
    - Flattening: Converts the combinational logic paths of the design to a two level, sum of products representation.
- **Gate Level Optimization:**
  - Works on the generic net list created by logic synthesis to produce a technology-specific net list.
  - Composed of: (1) Mapping, (2) Delay Optimization, (3) Design Rule Fixing and (4) Area optimization.
  - Mapping: Uses gates from the target technology libraries to generate a gate-level implementation of the design whose goal is to meet timing and area goals.

Starting up Synopsys Design Compiler

- To synthesize your designs, you will use Design Compiler(or design_vision). Every time you log on to the server, you have to type this command:

```
module load synopsys/syn/2008
```

- To invoke Design Compiler, type this command:

```
dc_shell-xg-t
```

- To invoke Design vision, type this command:

```
design_vision
```

- To synthesize your design, use the tcl script described in the previous slides. Usage:
  - 1. type "`source dc_script.tcl`" in dc_shell or
  - 2. type "`dc_shell-xg-t -f dc_sc.ript.tcl`" on the Linux command line

Generating AT/RAT file

- The synthesis result will be in: `${myBlock}.gate.v`
- For the cell count and AT/RAT file generation, copy ***`getLibCellCount.tcl`*** and ***`genAtRat.tcl`*** files from the scripts directory and execute:

```
source ./getLibCellCount.tcl > ./getLibCellCount.log
```

  - this will generate "myBlock.cells" which has the # number of cells in your block

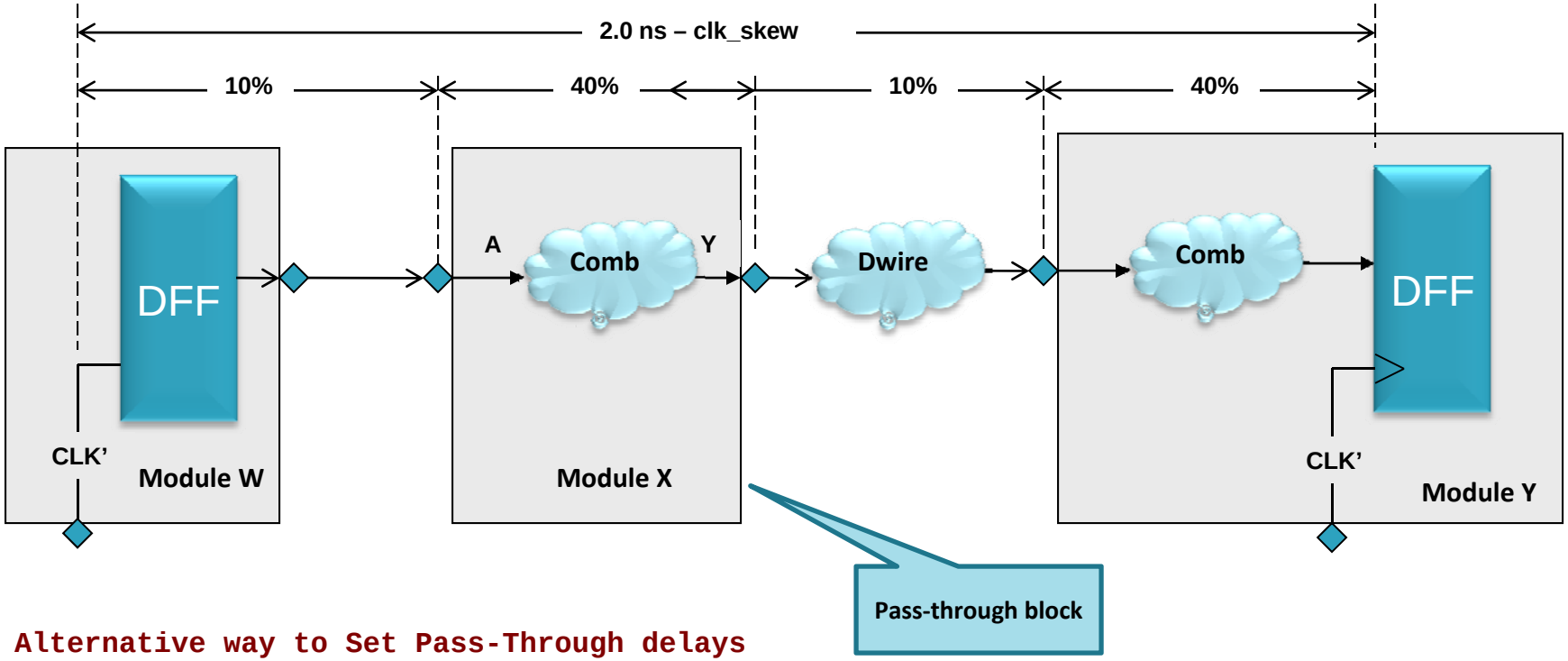
```
source ./genAtRat.tcl
```

  - this will generate "myBlock.atrat", which is atrat file

Backup

Specifying constraints for pass-through blocks

```
set TARGET_FREQ 0.5
set CLK_SKEW 0.125
set CCT [expr (1 / $TARGET_FREQ) - $CLK_SKEW]
set INPUT_EXT_DELAY  [expr 0.1 * $CCT]
set OUTPUT_EXT_DELAY [expr 0.5 * $CCT]
```




```
# Alternative way to Set Pass-Through delays
#set_max_delay 0.8 -from pin_A -to pin_Y
```