Lecture 22
Design Compiler in Depth

Xuan ‘Silvia’ Zhang
Washington University in St. Louis

http://classes.engineering.wustl.edu/ese461/

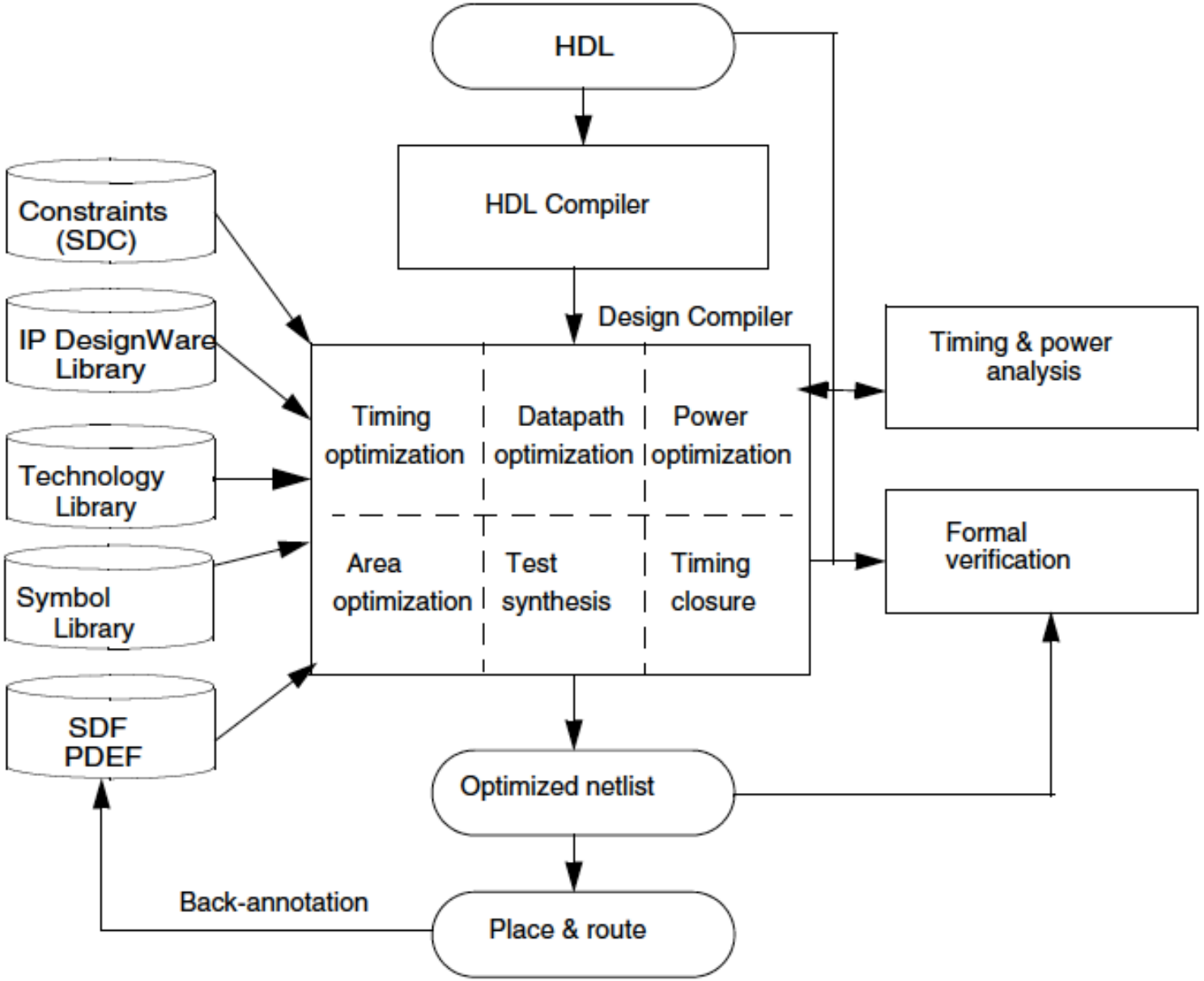
Timeline

- Class project tasks
  - logic synthesis
  - design optimization and iteration
  - place and route
  - final report
- Project milestones
  - 12/7: in-class presentation
  - 12/12: final report due
  - hard deadlines, not extendable
- Lecture plans
  - 11/30: 30min team meeting
  - 12/5: last lecture, Encounter tips, conclusion

Feedback from Mid-Project Reports

- Synthesizable Verilog code style
  - If-else, case: complete default branch
  - DO NOT mix blocking and non-blocking statements
  - proper FSM implementation
  - clocked Always block, reset and sensitivity list
- Testbench setup
  - instantiate memory
  - simulation termination
  - endianess (Bitcoin)
- To-do list
  - logic synthesis
  - place and route

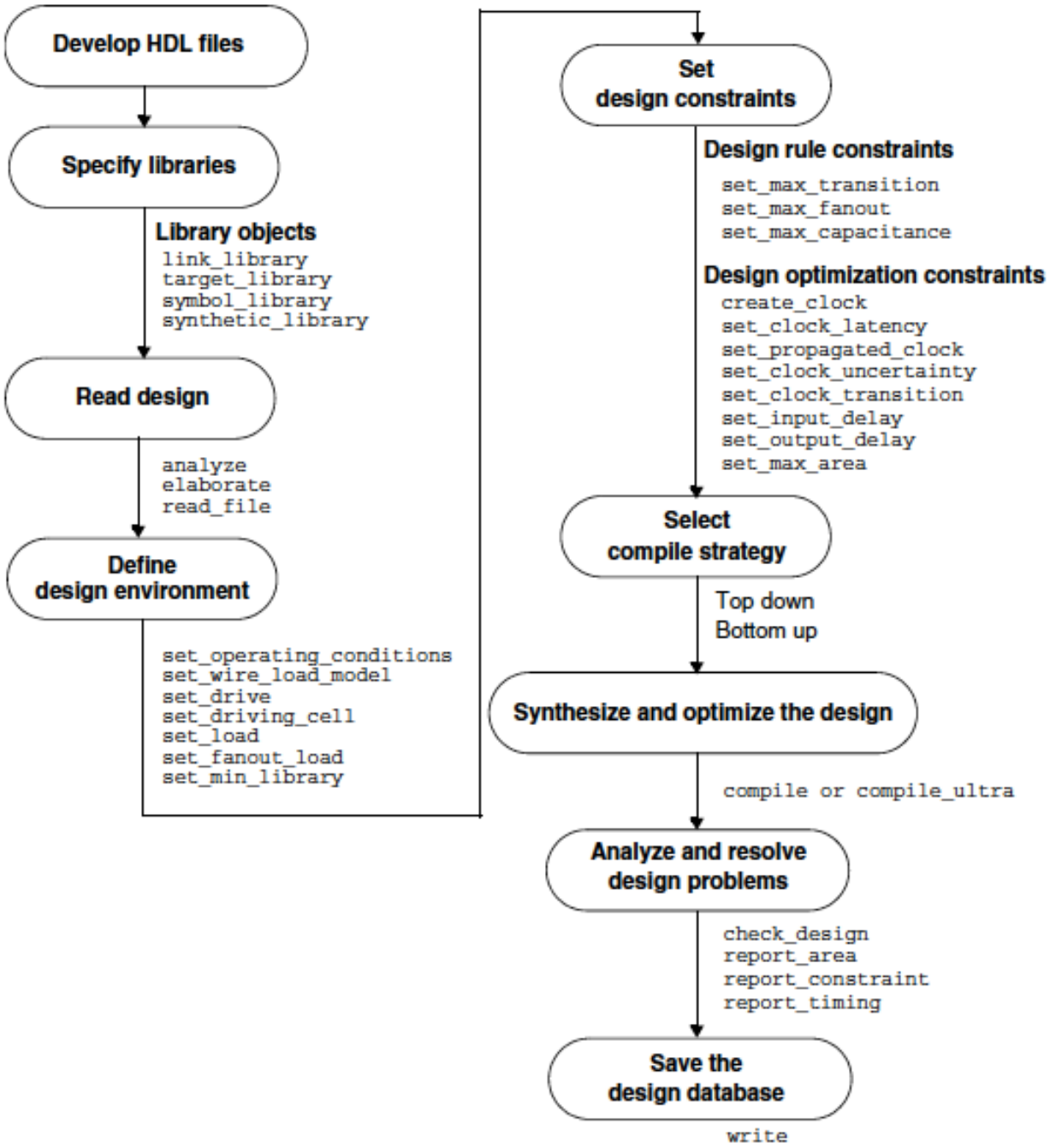
Figure 1-1 Design Compiler and the Design Flow



Design Compiler User Guide

- Develop HDL files
  - Chapter 3, “Preparing design files for synthesis”
- Specify libraries
  - Chapter 4, “Working with libraries”
- Read design
  - Chapter 5, “Working with designs in memory”
- Define design environment
  - Chapter 6, “Defining the design environment”
- Set design constraints
  - Chapter 7, “Defining design constraints”
- Select compile strategy
- Synthesize and optimize the design
  - Chapter 8, “Optimizing the design”

Figure 2-2 Basic Synthesis Flow



Organize the Design Data

Figure 3-1 Top-Down Compile Directory Structure

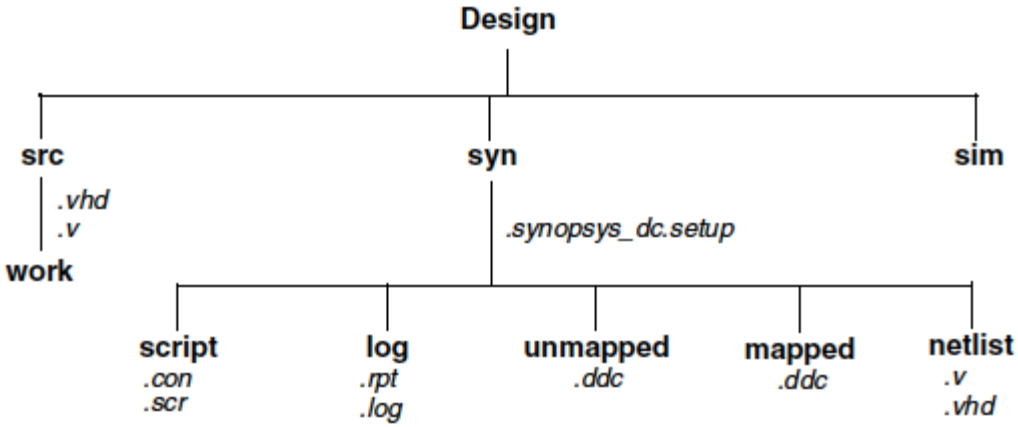
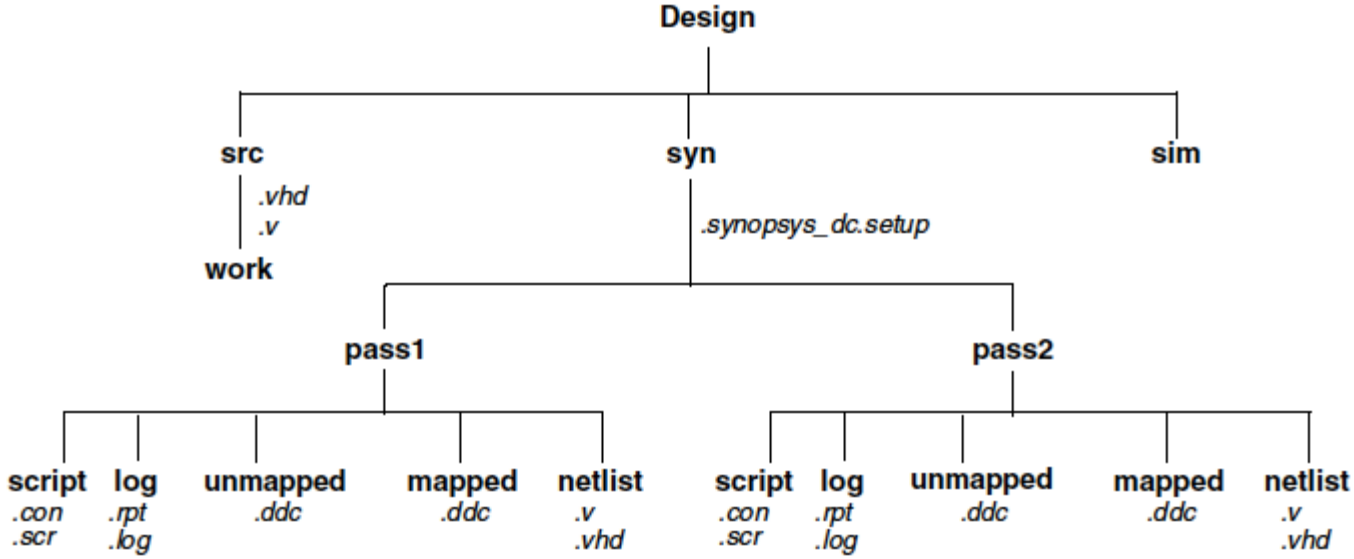


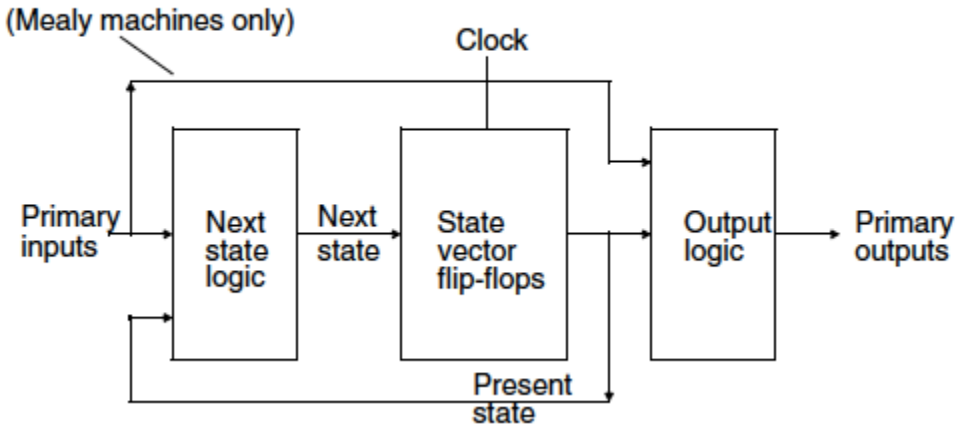
Figure 3-2 Bottom-Up Compile Directory Structure



HDL Coding for Synthesis

- FSM

Figure 3-10 Finite State Machine Architecture



- Sensitivity list
- Incomplete control statement

Example 3-3 Incorrect if Statement (Verilog)

```
if ((a == 1) && (b == 1))
   z = 1;
```

HDL Coding for Synthesis

- Value assignments
  - use nonblocking assignments within sequential always
  - use blocking assignments within combinational always
- Constant definition

Example 3-7 Using Macros and Parameters (Verilog)

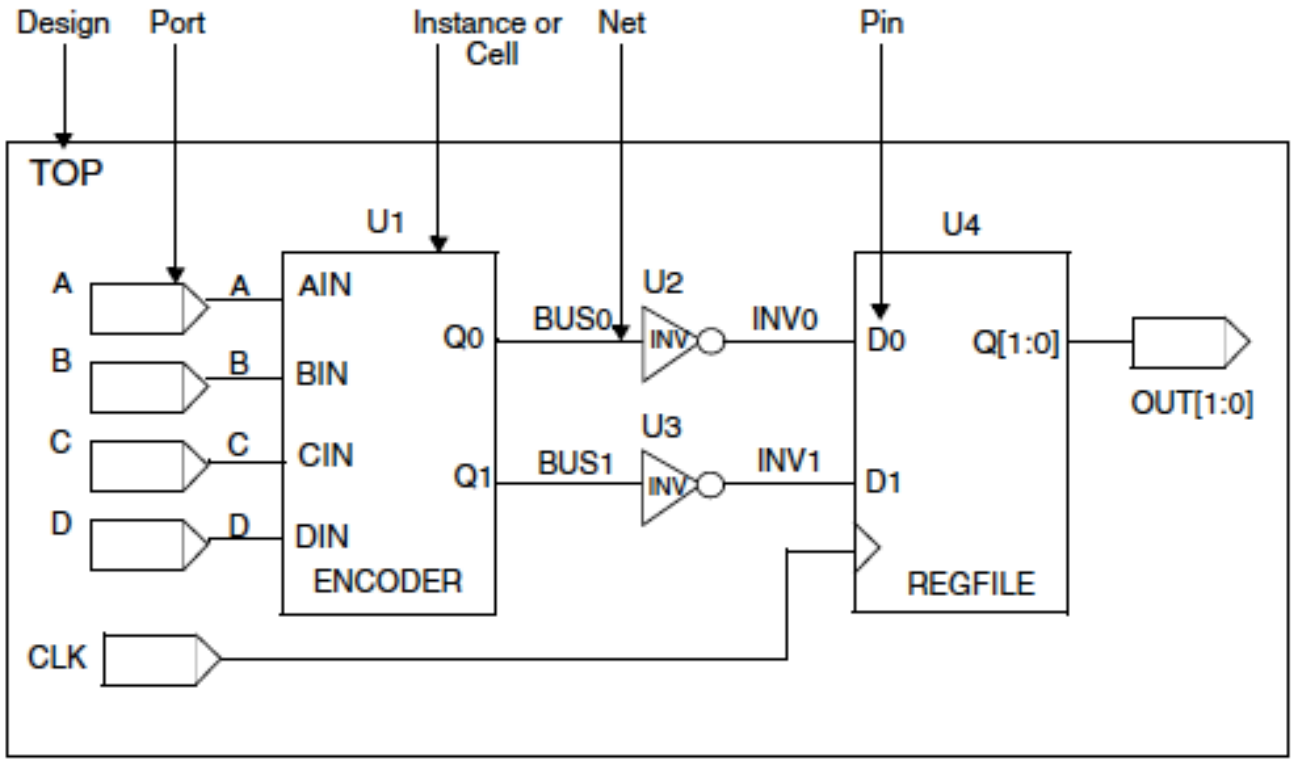
```
// Define global constant in def_macro.v
`define WIDTH 128

// Use global constant in reg128.v
reg regfile[WIDTH-1:0];

// Define and use local constant in module foo
module foo (a, b, c);
   parameter WIDTH=128;
   input [WIDTH-1:0] a, b;
   output [WIDTH-1:0] c;
```

- Guidelines for identifiers, expressions, and functions

Design Terminology and Objects

- Flat vs Hierarchical Designs

Figure 5-1 Design Objects



Design: {TOP, ENCODER, REGFILE}

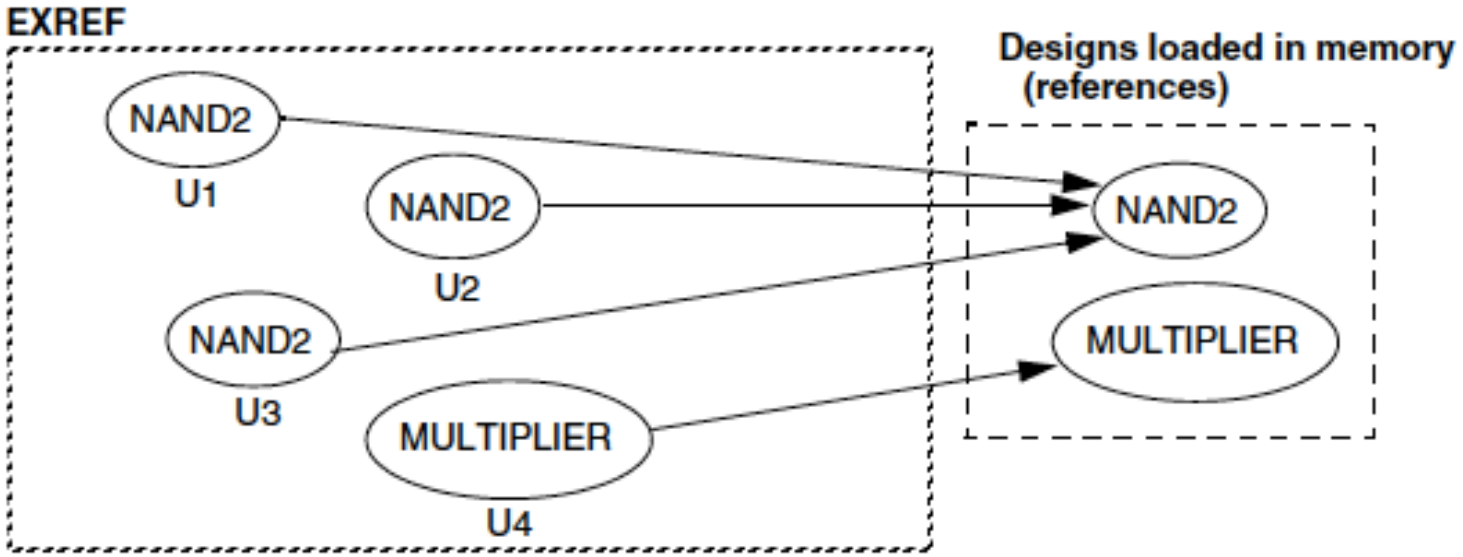
Reference: {ENCODER, REGFILE, INV}

Instance: {U1, U2, U3, U4}

Designs, Instances, and References

- analyze
- elaborate
- read_file
- link: link_library, search_path

Figure 5-2 Instances and References



Ungroup Hierarchies Automatically

- compile_ultra
  - by default, perform delay-based auto-ungrouping

Table 5-11 Preserving Hierarchical Pin Timing Constraints

Compile flow	Effect on hierarchical pin timing constraints
Ungrouping a hierarchy before optimization by using `ungroup`	Timing constraints placed on hierarchical pins are preserved. In previous releases, timing attributes placed on the hierarchical pins of a cell were not preserved when that cell was ungrouped. If you want your current optimization results to be compatible with previous results, set the `ungroup_preserve_constraints` variable to `false`. The default for this variable is `true`, which specifies that timing constraints will be preserved.
Ungrouping a hierarchy during optimization by using `compile -ungroup_all` or `set_ungroup` followed by `compile`	Timing constraints placed on hierarchical pins are not preserved. To preserve timing constraints, set the `auto_ungroup_preserve_constraints` variable to `true`.
Automatically ungrouping a hierarchy during optimization, that is, by using the `compile_ultra` or `compile -auto_ungroup area|delay` command	Design Compiler does not ungroup the hierarchy. To make Design Compiler ungroup the hierarchy and preserve timing constraints, set the `auto_ungroup_preserve_constraints` variable to `true`.

Edit Designs

Table 5-12 Design Editing Tasks and Commands

Object	Task	Command
Cells	Create a cell	`create_cell`
	Delete a cell	`remove_cell`
Nets	Create a net	`create_net`
	Connect a net	`connect_net`
	Disconnect a net	`disconnect_net`
	Delete a net	`remove_net`
Ports	Create a port	`create_port`
	Delete a port	`remove_port`
		`remove_unconnected_port`
Pins	Connect pins	`connect_pin`
Buses	Create a bus	`create_bus`
	Delete a bus	`remove_bus`

Edit Designs

```
dc_shell> get_pins U8/*
{"U8/A", "U8/Z"}
dc_shell> all_connected U8/A
{"n66"}
dc_shell> all_connected U8/Z
{"OUTBUS[10]"}
dc_shell> remove_cell U8
Removing cell 'U8' in design 'top'.
1
dc_shell> create_cell U8 IVP
Creating cell 'U8' in design 'top'.
1
dc_shell> connect_net n66 [get_pins U8/A]
Connecting net 'n66' to pin 'U8/A'.
1
dc_shell> connect_net OUTBUS[10] [get_pins U8/Z]
Connecting net 'OUTBUS[10]' to pin 'U8/Z'.
1
```

Define the Design Environment

Figure 6-1 Commands Used to Define the Design Environment

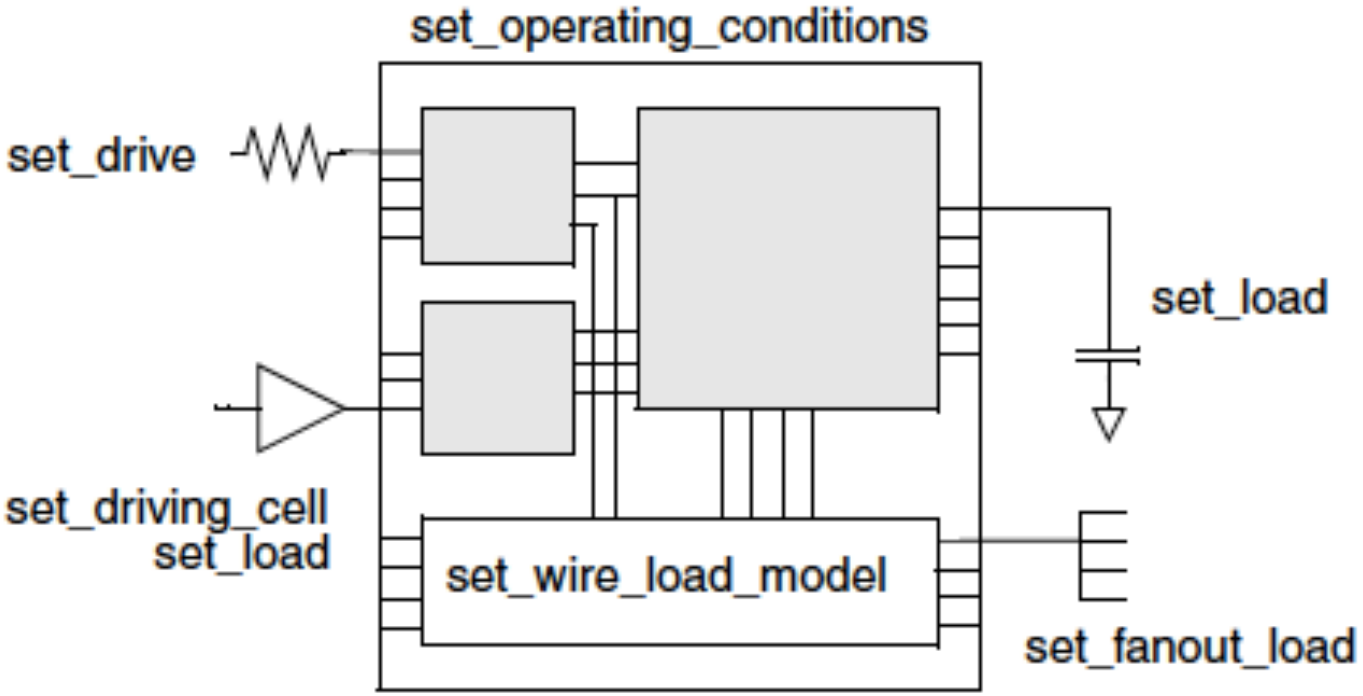
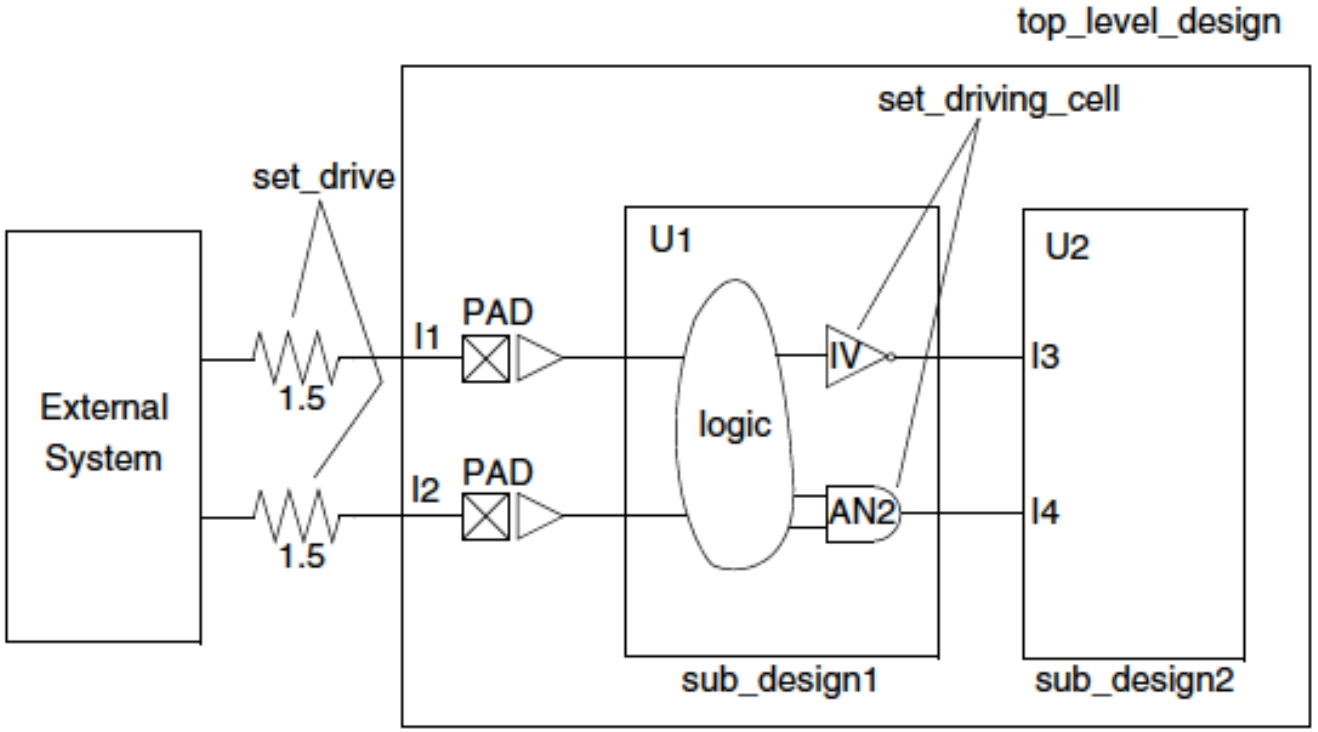


Figure 6-3 Drive Characteristics



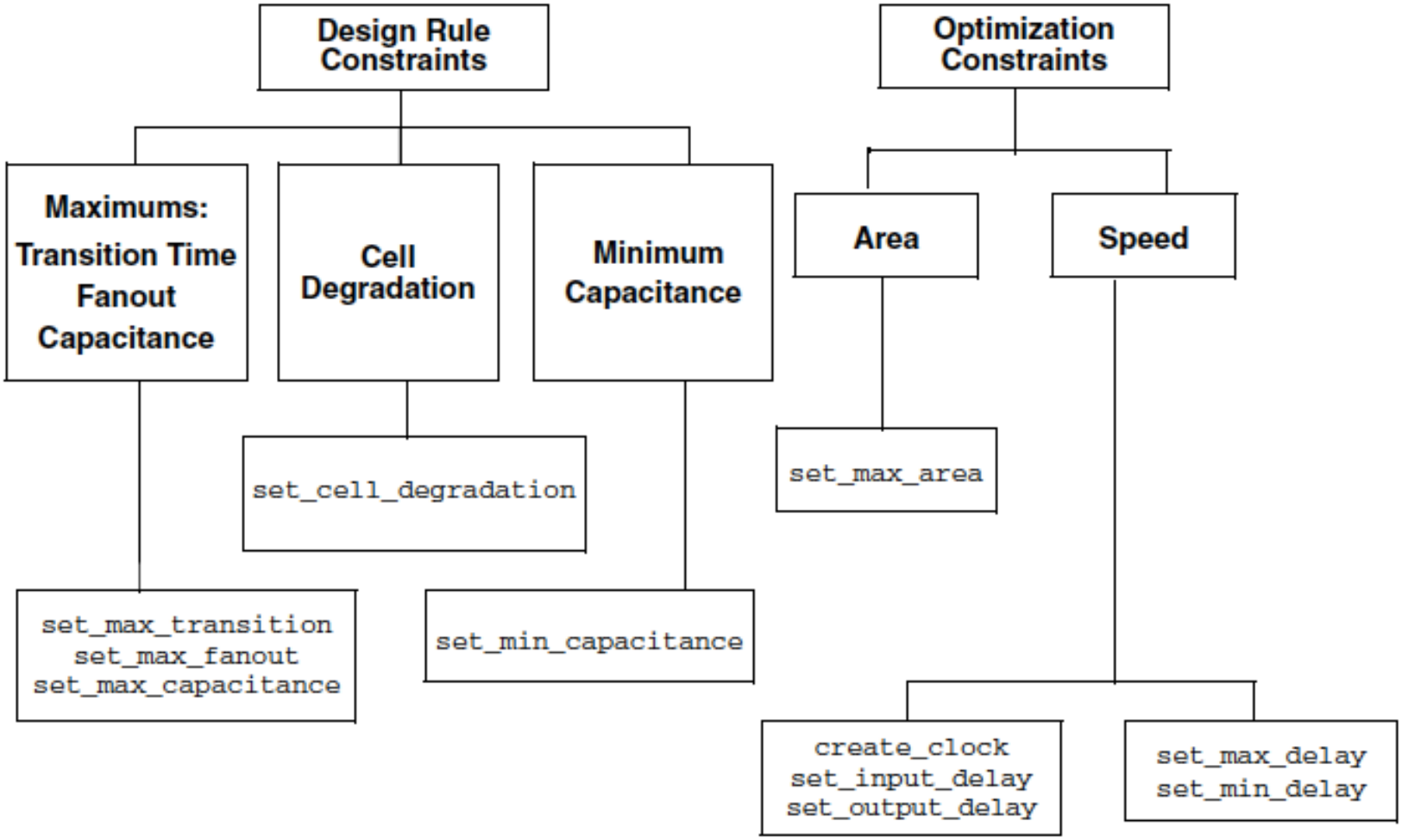

```
dc_shell> current_design top_level_design
dc_shell> set_drive 1.5 {I1 I2}

dc_shell> current_design sub_design2

dc_shell> set_driving_cell -lib_cell IV {I3}

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

Model the System Interface

- Define drive characteristics for input ports
- Define load on input and output ports
- Define fanout loads on output ports
- (Similarly) Set logic constraints on ports
  - define ports as logically equivalent
  - define logically opposite input ports
  - allow assignment of any signal to an input
  - always one or zero
  - unconnected

Define Design Constraints

Figure 7-1 Major Design Compiler Constraints



Summary of Design Rule Commands

Table 7-1 Design Rule Command and Object Summary

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

Optimization Constraints

- Timing constraints (performance and speed)
  - input and output delays (synchronous paths)
  - minimum and maximum delay (asynchronous paths)
  - note: set_fix_hold

- Maximum area
  - number of gates

Cost Function Calculation

- Design rule cost function

Figure 7-2 Design Rule Cost Equation

$$\sum_{i=1}^{m} max(d_i,\ 0) \times w_i$$

i = Index
d = Delta Constraint
m = Total Number of Constraints
w = Constraint Weight

- Max delay cost function

Figure 7-5 Cost Calculation for Maximum Delay

$$\sum_{i=1}^{m} v_i \times w_i$$

i = Index
v = worst violation
m = number of path groups
w = weight

- Min delay cost function

Figure 7-6 Cost Calculation for Minimum Delay

$$\sum_{i=1}^{m} v_i$$

i = index
m = number of paths affected by set_min_delay or set_fix_hold
v = minimum delay violation max(0,required_path_delay - actual_path_delay)

Optimize the Design

- Architecture optimization
  - sharing common subexpressions
  - sharing resources
  - reordering operators
  - selecting DesignWare implementations, etc.
- Logic-level optimization
  - structuring
  - flattening
- Gate-level optimization
  - mapping
  - delay optimization
  - design rule fixing
  - area optimization

Optimization Example

Figure 8-1 Design to Illustrate Compile Strategies

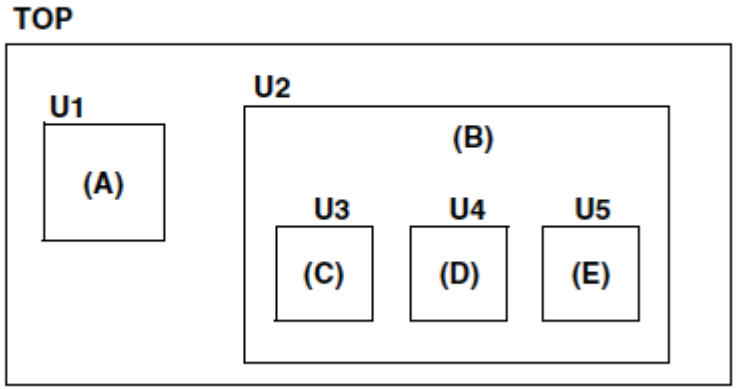


Table 8-1 Design Specifications for Design TOP

Specification type	Value
Operating condition	WCCOM
Wire load model	"20x20"
Clock frequency	40 MHz
Input delay time	3 ns
Output delay time	2 ns
Input drive strength	drive_of (IV)
Output load	1.5 pF

Top-Down Optimization

Example 8-1 Constraints File for Design TOP (defaults.con)

```
set_operating_conditions WCCOM
set_wire_load_model "20x20"
create_clock -period 25 clk
set_input_delay 3 -clock clk \
   [remove_from_collection [all_inputs] [get_ports clk]]
set_output_delay 2 -clock clk [all_outputs]
set_load 1.5 [all_outputs]
set_driving_cell -lib_cell IV [all_inputs]
set_drive 0 clk
```

Example 8-2 Top-Down Compile Script

```
/* 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
```

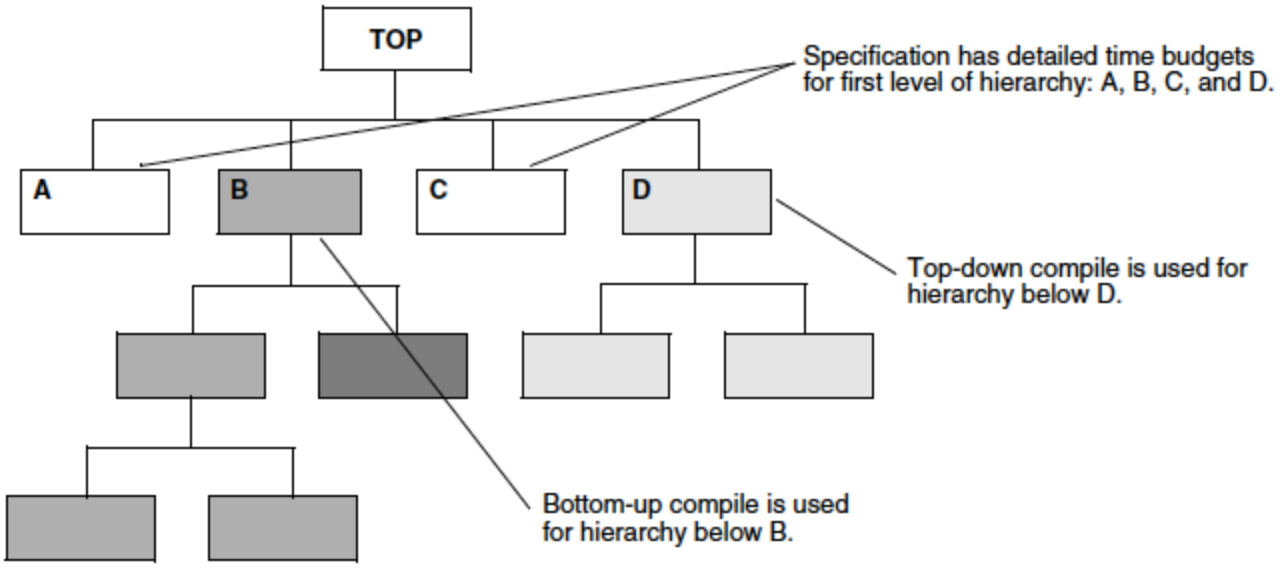
Bottom-Up Optimization

Example 8-3 Bottom-Up Compile Script

```
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.

# 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 Optimization

Figure 8-2 Mixing Compilation Strategies



Suggested Reading

- Partitioning for Synthesis
  - Chapter 3 (3-4)
- Working with Attributes
  - Chapter 5 (5-40)
- Define Design Constraints
  - Chapter 6
  - reporting constraints
  - characterize subdesigns
- Optimize the Design
  - Chapter 7

Questions?

Comments?

Discussion?