ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN-ĐIỆN TỬ
BỘ MÔN KỸ THUẬT ĐIỆN TỬ

ASIC CHIP AND IP CORE DESIGN

Chapter 3: Architecture Design – Part 2

3.1 Design Entry
3.2 Design Constraints
3.3 Optimizing the design
3.4 Hardware Description Languages

3.3 Optimizing The Design

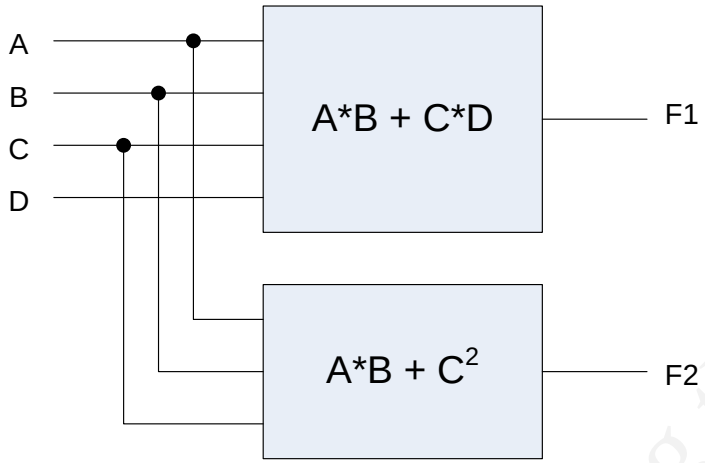
- **Objectives:** maps the design to an optimal combination of specific target library cells, based on the design's functional, speed, and area requirements
  1. The Optimization Process
  2. Selecting and Using a Compile Strategy
  3. Resolving Multiple Instances of a Design Reference
  4. Preserving Subdesigns
  5. Understanding the Compile Cost Function
  6. Performing Design Exploration
  7. Performing Design Implementation
  8. Using Target Library Subsets
  9. Using Library Subsets for Sequential Cells

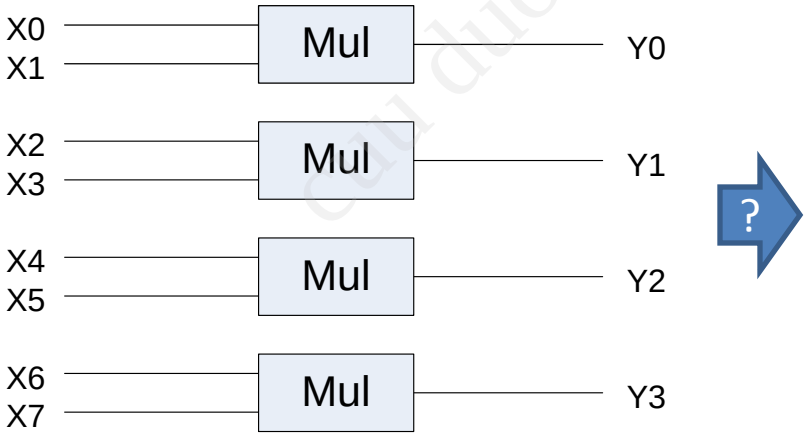
3.3.1 The Optimization Process

- Design Compiler performs the following three levels of optimization:
  - Architectural optimization
  - Logic-level optimization
  - Gate-level optimization
- Architectural Optimization
  - Sharing common subexpressions
  - Sharing resources
  - Selecting DesignWare implementations
  - Reordering operators
  - Identifying arithmetic expressions for data-path synthesis (DC Ultra only).

3.3.1 The Optimization Process

- Optimize the following designs



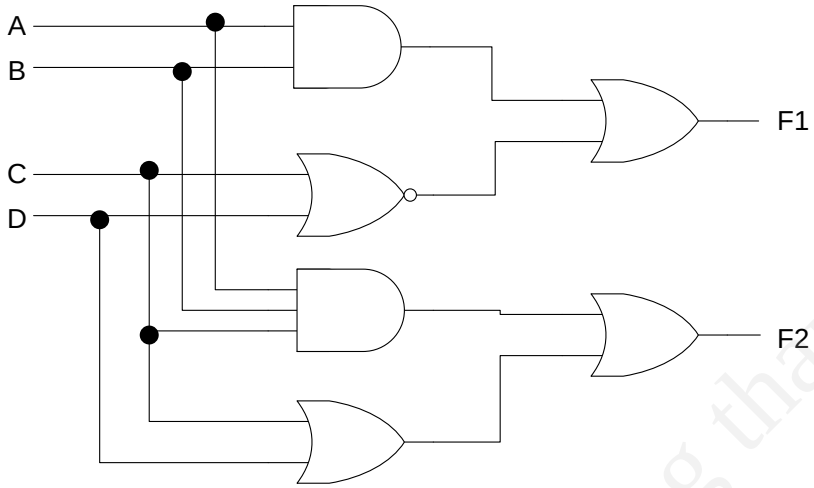


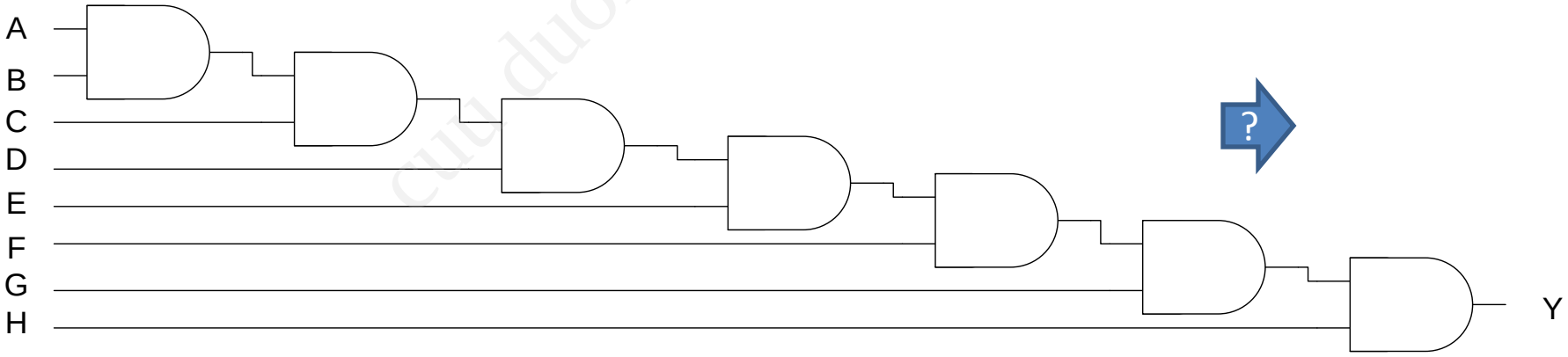
3.3.1 The Optimization Process

- Logic-Level Optimization
  - Structuring
    - adds intermediate variables and logic structure to a design
    - Design Compiler searches for subfunctions that can be factored out and evaluates these factors
  - Flattening
    - convert combinational logic paths of the design to a two-level, sum-of-products representation
    - Design Compiler removes all intermediate variables, and therefore all its associated logic structure, from a design.

3.3.1 The Optimization Process

- Optimize the following circuits





3.3.1 The Optimization Process

- Gate-Level Optimization: works on the generic **netlist** created by logic synthesis to produce a technology-specific netlist.
  - Mapping
    - This process 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. Use the compile_ultra command or the compile command to control the mapping algorithms used by Design Compiler.
  - Delay optimization
    - The process goal is to fix delay violations introduced in the mapping phase. Delay optimization does not fix design rule violations or meet area constraints.
  - Design rule fixing
    - The process goal is to correct design rule violations by inserting buffers or resizing existing cells. Design Compiler tries to fix these violations without affecting timing and area results, but if necessary, it does violate the optimization constraints.
  - Area optimization
    - The process goal is to meet area constraints after the mapping, delay optimization, and design rule fixing phases are completed. However, Design Compiler does not allow area recovery to introduce design rule or delay constraint violations as a means of meeting the area constraints.

3.3.2 Selecting and Using a Compile Strategy

- **Top-down compile:** the top-level design and all its subdesigns are compiled together
- **Bottom-up compile:** 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:** the top-down or bottom-up strategy, whichever is most appropriate, is applied to the individual subdesigns

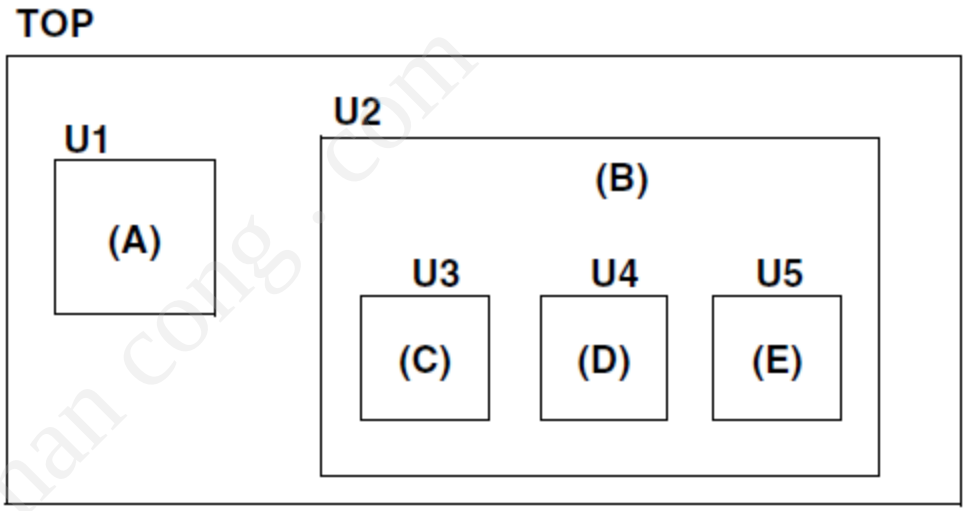
Top-Down Compile

- **Advantages**
  - Only top level constraints are needed.
  - Takes care of interblock dependencies automatically
  - Better results due to optimization across entire design.
- **Disadvantages**
  - Long compile time
  - Incremental changes to the sub-blocks require complete re-synthesis.
  - Does not perform well, if design contains multiple clocks or generated clocks.
- **Carry out the following steps**
  - Read in the entire design
  - Apply attributes and constraints to the top level
  - Compile the design

Selecting and Using a Compile Strategy

Example: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 Compile

- Advantages:
  - Compiles large designs by using the divide-and-conquer approach
  - Requires less memory than top-down compile
  - Allows time budgeting
- Disadvantages:
  - Complex constraints
  - Reduce compile time
- Requirements
  - Iterating until the interfaces are stable
  - Manual revision control

Bottom-Up Compile

Example: 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.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
```

Bottom-Up Compile

Example: Bottom-Up Compile Script

```
# 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
```

Bottom-Up Compile

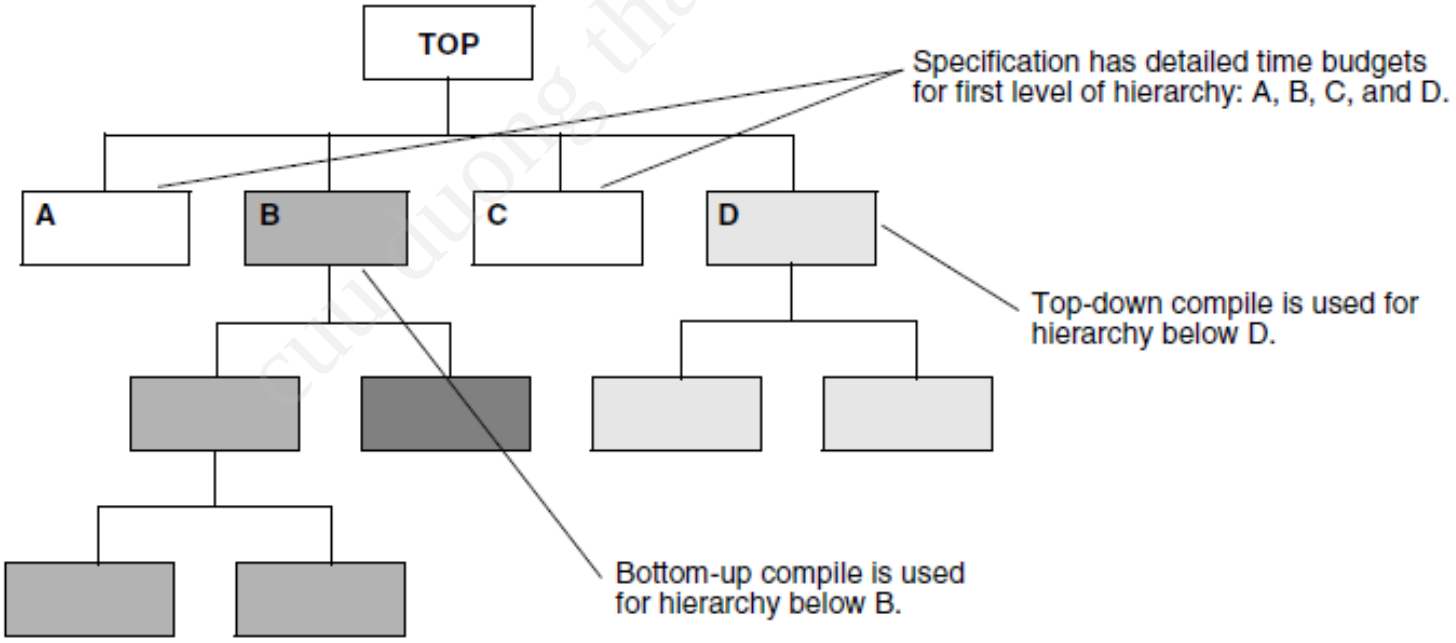
Compile steps:

1. Develop both a default constraint file and subdesign-specific constraint files.
2. Compile the subdesigns independently.
3. Read in the top-level design and any compiled subdesigns not already in memory.
4. Set the current design to the top-level design, link the design, and apply the top-level constraints. If the design meets its constraints, you are finished. Otherwise, continue with the following steps.
5. Apply the characterize command to the cell instance with the worst violations.
6. Use write_script to save the characterized information for the cell.
7. Use remove_design -all to remove all designs from memory.
8. Read in the RTL design of the previously characterized cell.
9. Set current_design to the characterized cell's subdesign and recompile, using the saved script of characterization data.
10. Read in all other compiled subdesigns.
11. Link the current subdesign.
12. Choose another subdesign, and repeat steps 3 through 9 until you have recompiled all subdesigns, using their actual environments.

Mixed Compile Strategy

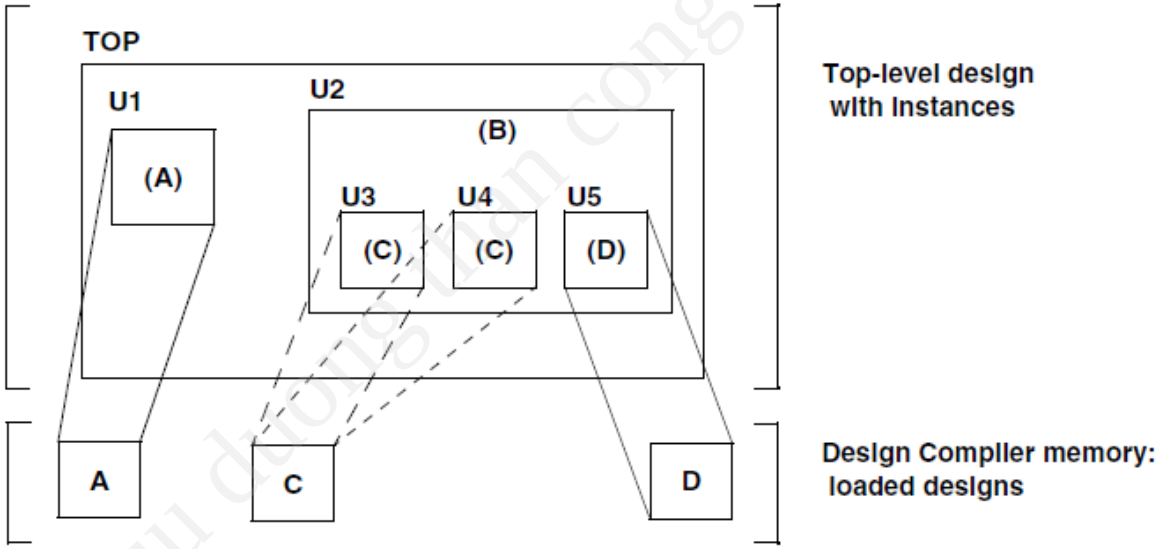
- Take advantage of the benefits of both the top-down and the bottom-up compile strategies by using both.
  - Use the top-down compile strategy for small hierarchies of blocks.
  - Use the bottom-up compile strategy to tie small hierarchies together into larger blocks.

Example of the mixed compilation strategy



3.3.3 Resolving Multiple Instances of a Design Reference

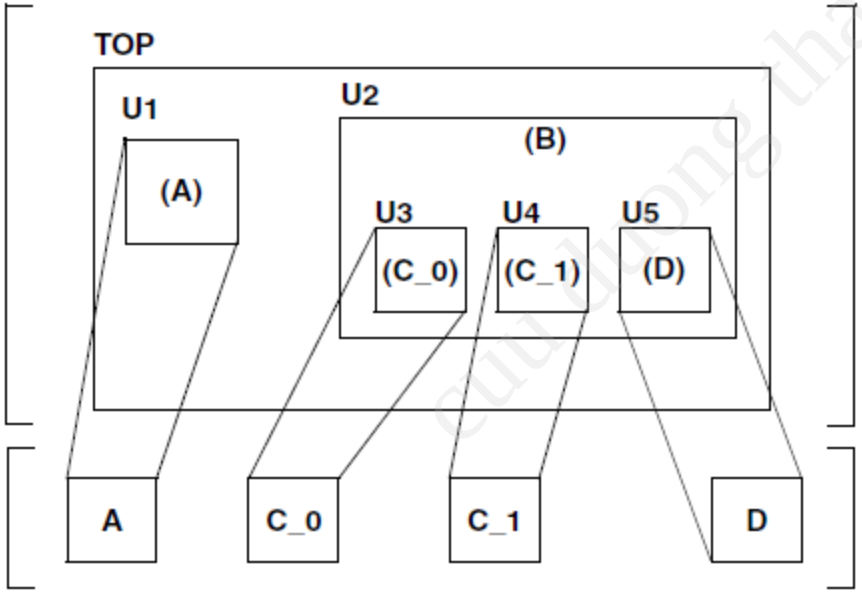
- In a hierarchical design, subdesigns are often referenced by more than one cell instance, that is, multiple references of the design can occur.



- Three methods to solve the problem:
  - The uniquify method
  - Compile-once-don’t-touch method
  - Ungroup method

The Uniquify Method

- **Uniquify** command: create a uniquely named copy of the design for each instance.
- From the version V-2004.06, the tool automatically uniquifies design as part of the compile process
  - dc_shell> **uniquify -cell *mid1/bot1***



Top-level Design With Instances

dc_shell> **current_design top**
dc_shell> **compile**

Design Compiler Memory: Loaded Designs

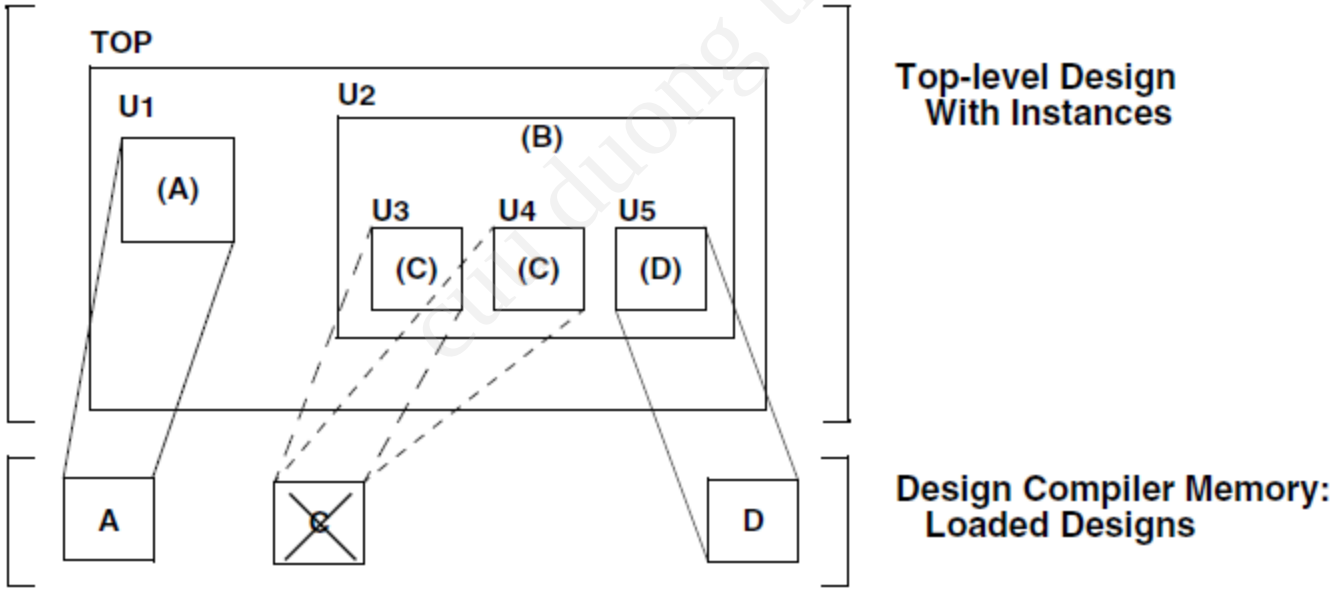
Compile-Once-Don't-Touch Method

- Use:
  - **check_design -multiple_design:** report information messages realted to multiply-instantiated designs
  - **set_dont_touch** : preserve the compiled sybdesign while the remaining design are compiled
- Steps:
  1. Characterize the subdesign's instance that has the worst-case environment.
  2. Compile the referenced subdesign.
  3. Use the set_dont_touch command to set the dont_touch attribute on all instances that reference the compiled subdesign.
  4. Compile the entire design.

Compile-Once-Don't-Touch Method

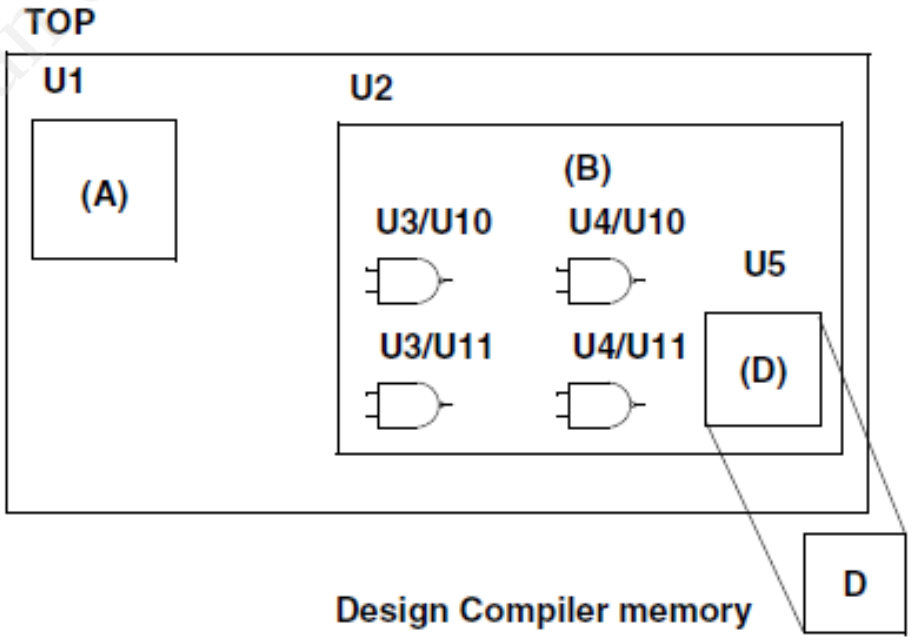
- dc_shell> **current_design top**
- dc_shell> **characterize U2/U3**
- dc_shell> **current_design C**
- dc_shell> **compile**
- dc_shell> **current_design top**
- dc_shell> **set_dont_touch {U2/U3 U2/U4}**
- dc_shell> **compile**

The X drawn over the C design, which has already been compiled, indicates that the dont_touch attribute has been set. This design is not modified when the top-level design is compiled.



Ungroup Method

- Use:
  - Ungroup command: remove the hierarchy to produce a flattened netlist
- Example: After ungrouping the instances of a subdesign, recompile the top-level design.
  - dc_shell> **current_design B**
  - dc_shell> **ungroup {U3 U4}**
  - dc_shell> **current_design top**
  - dc_shell> **compile**



Compare three methods

No.	Method	Characteristics
1	Uniquify	•Requires more memory •Takes longer to compile
2	Compile-once-don't-touch	•Compiles the reference design once •Requires less memory than the uniquify method •Takes less time to compile than the uniquify method
3	Ungroup	•Requires more memory and takes longer to compile than the compile-once-don't-touch method •Provides the best synthesis results

3.3.4 Preserving Subdesigns

- The **set_dont_touch** command preserves a subdesign during optimization.
- 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.
- Example:
  - dc_shell> **get_nets *my_net***
  - dc_shell> **set enable_keep_signal_dt_net true**
  - dc_shell> **set_dont_touch [get_nets *my_net*] true**
  - dc_shell> **compile_ultra**
  - dc_shell> **get_nets *my_net***
- **To remove don't_touch**
  - dc_shell> **set_dont_touch [get_nets *my_net*] false**

3.3.5 Understanding the Compile Cost Function

- The compile cost function consists of design rule costs and optimization costs.
  a) Design rule costs
  b) Connection class
  c) Multiple port nets
  d) Maximum transition time
  e) Maximum fanout
  f) Maximum capacitance
  g) Cell degradation
- Optimization costs
  a) Maximum delay
  b) Minimum delay
  c) Maximum power
  d) Maximum area

3.3.5 Understanding the Compile Cost Function

- **set_cost_priority**: change the priority of the maximum design rule costs and the delay costs
- **compile -no_design_rule** : disable evaluation of the design rule cost function
- **compile -only_design_rule** : disable evaluation of the optimization cost function

3.3.6 Perform Design Exploration

- To invoke the default synthesis algorithm, use the compile command with no options:
  - dc_shell> **compile**
- Use options:
  - map_effort medium
  - area_effort

3.3.7 Performing Design Implementation

- There are 4 techniques:
  a) Optimizing High-Performance Designs
  b) Optimizing for Maximum Performance
  c) Optimizing for Minimum Area
  d) Optimizing Datapaths

a) Optimizing High-Performance Designs

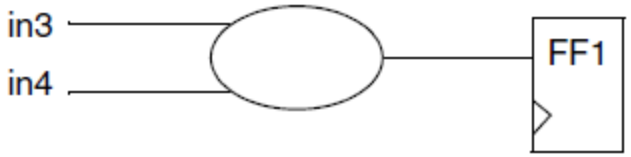
- Use command
  - Compile_ultra
- Apply the best possible set of timing-centric variables or commands during compile for critical delay optimization as well as improvement in area quality of results (QoR)
- Need DC Ultra license and DesignWare Foundation license

b) Optimizing for Maximum Performance

- Apply the following methods:
  - Create path groups: using **group_path** command
  - Fix heavily loaded nets
  - Auto-ungroup hierarchies on the critical path
  - Perform a high-effort incremental compile
  - Perform a high-effort compile

dc_shell> **group_path -name group3 -from in3 -to FF1/D -weight 2.5**

Path Group Example

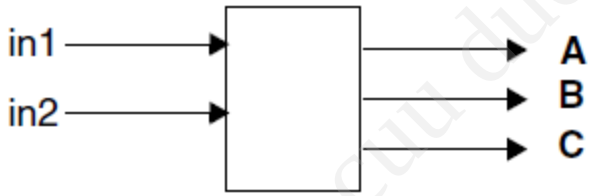


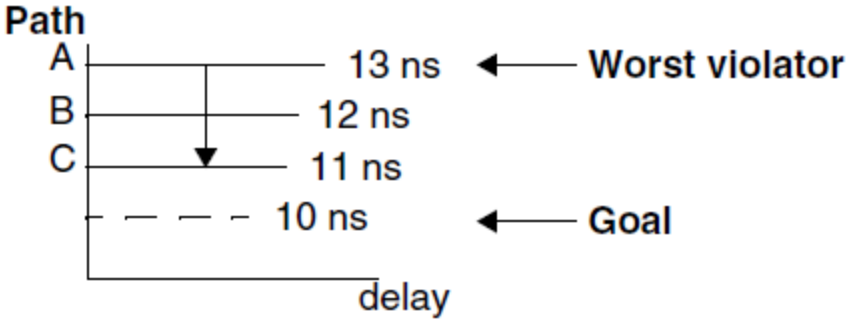
b) Optimizing for Maximum Performance

- Optimize Near-Critical Paths
  - Use the **-critical_range** option of the **group_path** command.
  - Use the **set_critical_range** command.

Example:

```
create_clock -period 20 clk
set_critical_range 3.0 $current_design
set_max_delay 10 {A B C}
group_path -name group1 -to {A B C}
```





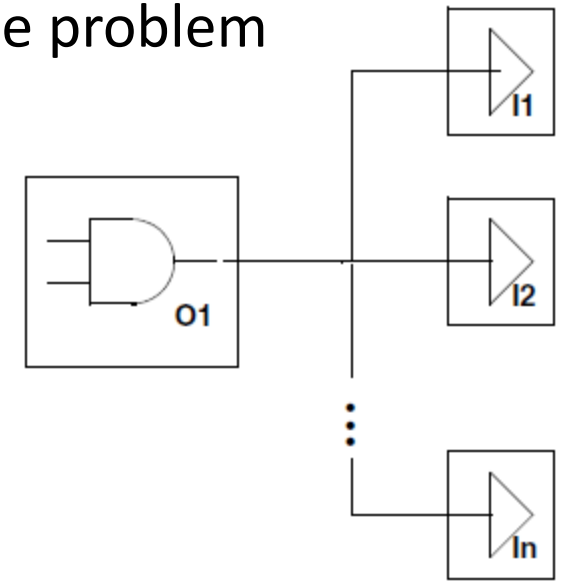
b) Optimizing for Maximum Performance

- Optimizing All Paths
  - Script to create a path group for each endpoint.

```
set endpoints \
    [add_to_collection [all_outputs] \
    [all_registers -data_pins]]
foreach_in_collection endpt $endpoints {
    set pin [get_object_name $endpt]
    group_path -name $pin -to $pin
}
```

b) Optimizing for Maximum Performance

- Fixing Heavily Loaded Nets
  - If the large load resides in a single module and the module contains no hierarchy
    - source constraints.con
    - compile_ultra
    - balance_buffer -from [get_pins buf1/Z]
  - If the large loads reside across the hierarchy from several modules, apply design rules to fix the problem
    - source constraints.con
    - compile_ultra
    - set_max_capacitance 3.0
    - compile_ultra -only_design_rule



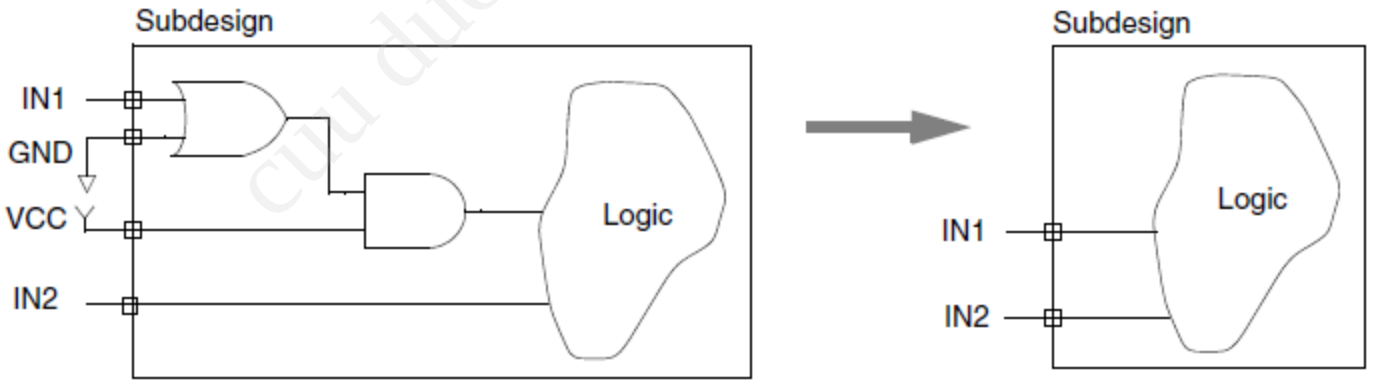
b) Optimizing for Maximum Performance

- Automatically Ungrouping Hierarchies on the Critical Path
  - compile_ultra –auto_ungroup
- Performing a High-Effort Compile
  - dc_shell> **elaborate my_design**
  - dc_shell> **compile -map_effort high**
- Performing a High-Effort Incremental Compile
  - dc_shell> **dont_touch *noncritical_blocks***
  - dc_shell> **compile -map_effort high -incremental_mapping**
- To reduce runtime, place a **dont_touch** attribute on all blocks that already meet timing constraints

c) Optimizing for Minimum Area

- Try the following methods :
  - **Disabling Total Negative Slack Optimization**
    - dc_shell> **set_max_area -ignore_tns *max_area***
  - **Boundary Optimization**
    - dc_shell> **compile -boundary_optimization**

      or
    - dc_shell> **set_boundary_optimization subdesign**



d) Optimizing Datapaths

- Datapath extraction transforms arithmetic operators, such as addition, subtraction, and multiplication, into datapath blocks to be implemented by a datapath generator.
- This transformation improves the QoR by utilizing the carry-save arithmetic technique.
- DC Ultra enables datapath extraction and explores various datapath and resource-sharing options during compile
  - Use: compile_ultra
- Benefits:
  - Shares datapath operators
  - Extracts the datapath
  - Explores better solutions that might involve a different resource-sharing configuration
  - Allows the tool to make better tradeoffs between resource sharing and datapath optimization

3.3.8 Using Target Library Subsets

- Design Compiler allows you to restrict optimization of a design block using a subset of the target library
- To restrict the library cells to be used in a particular block:
  - set_target_library_subset -object_list *design_cells -top {library_list}* -dont_use *lib_cells -only_here lib_cells*
    - -object_list lists the hierarchical cells
    - -top specifies to use the set_target_library_subset command at the top level.
    - library_list is a space-separated list of target library file names
    - -dont_use lib_cells specifies library cells that cannot be used for optimization
    - -only_here lib_cells specifies library cells that can only be used for optimization
- To remove a target library subset constraint from the design:
  - remove_target_library_subset
  - Reset_design

3.3.8 Using Target Library Subsets

- Checking target library subsets
  - **check_mv_design -target_library_subset :** searches the design for any cells that do not follow the rules for the subset.
  - **check_target_library_subset:** check for the following conditions
    - Conflicts between target library subsets and the global target_library variable
    - Conflicts between operating condition and target library subset
    - Conflicts between the library cell of a mapped cell and target library subset
- Example:
  - dc_shell-topo> set target_library "lib1.db lib2.db"
  - dc_shell-topo> set_target_library_subset "lib2.db" -object_list \[get_cells u1]

3.3.9 Using Library Subsets for Sequential Cells

- **define_libcell_subset** : restrict the mapping and optimization of sequential cells in a design to a user-specified subset of library cells in a target library.
  - dc_shell-topo> define_libcell_subset -libcell_list {SDFLOP1 SDFLOP2} \
  - -family_name specialflops
  - dc_shell-topo> set_libcell_subset -object_list [get_cells {reg01 reg02}] \
  - -family_name specialflops
- **report_libcell_subset**: report information about the library cell subset
  - dc_shell-topo> report_libcell_subset -object_list [get_cells reg01]

```
****************************************
Report : library cell subset
Design : chip
Version: G-2012.06
Date : Wed Mar 21 16:16:58 2012
****************************************
Libcell Family Libcells
--------------------------------------------------------------------------------
specialflops sff0 sff1 sff2
--------------------------------------------------------------------------------
Object Reference Libcell Family
--------------------------------------------------------------------------------
reg01 sff0 specialflops
```

3.3.9 Using Library Subsets for Sequential Cells

- Removing Library Cell Subsets
  - **remove_libcell_subset** command removes a library cell subset constraint from specified sequential cells or removes a user-defined library cell subset.

- Ex1: the library subset constraint is removed from the reg01 and reg02 instances:
  - dc_shell-topo> remove_libcell_subset -object_list [get_cells reg01 reg02]

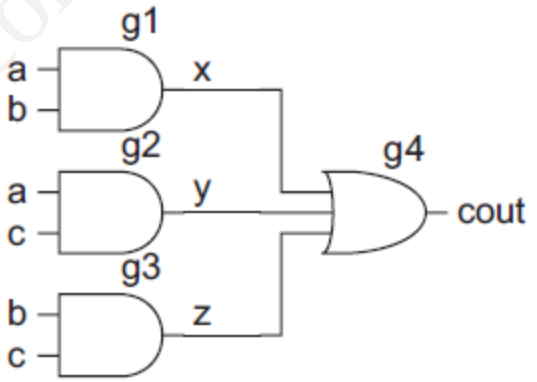
- Ex2: the user-defined specialflops library cell subset is removed:
  - dc_shell-topo> remove_libcell_subset -family_name specialflops

3.4 Hardware Description Language

- The two most popular HDL
  - Verilog
    - Developed by Advanced Integrated Design Systems
  - VHDL (Very High Speed Integrated Circuits HDL)
    - was developed by committee under government sponsorship

Circuit Design by Verilog

```
module carry( input logic a, b, c,
                    output logic cout);
  logic x, y, z;
  and  g1(x, a, b);
  and  g2(y, a, c);
  and  g3(z, b, c);
  or   g4(cout, x, y, z);
endmodule
```



(a)