cadence®

Conformal® LEC

Logic Equivalence Checker

Basic Training Manual

Getting Help

- You can get help with Cadence software from the following sources:
  - Education Services training materials
  - SourceLink service on the Internet: *sourcelink.cadence.com*
    - OR go to the Cadence home page (*http://www.cadence.com*).
      - Select the Customer Support link.
      - You must have a maintenance contract to log in (authorization number).
  - Customer Response Center (CRC) or send your request to support@cadence.com

Agenda of Conformal LEC Basic Training

- Overview of Verplex formal verification tools
- Conformal® LEC usage
  - Introduction to LEC
  - A typical session (flat compare)
  - Hierarchical compare

Overview of Verplex formal verification tools

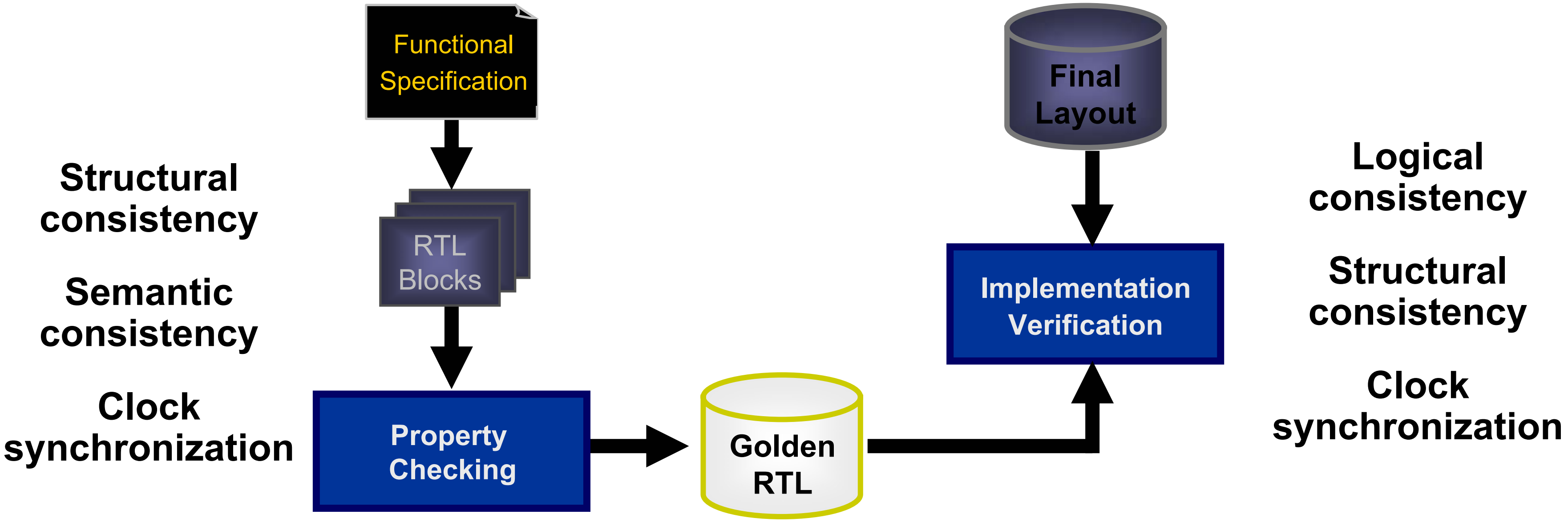
Formal verification

- A systematic method that uses mathematical proof to check the design’s properties
- Advantages of formal verification
  - Enable "White box" verification
  - No test vectors required
  - Exhaustive verification
  - Find bugs earlier
  - Speed

Functional Closure

Verplex enables you to obtain the Golden RTL that meets the functional specifications

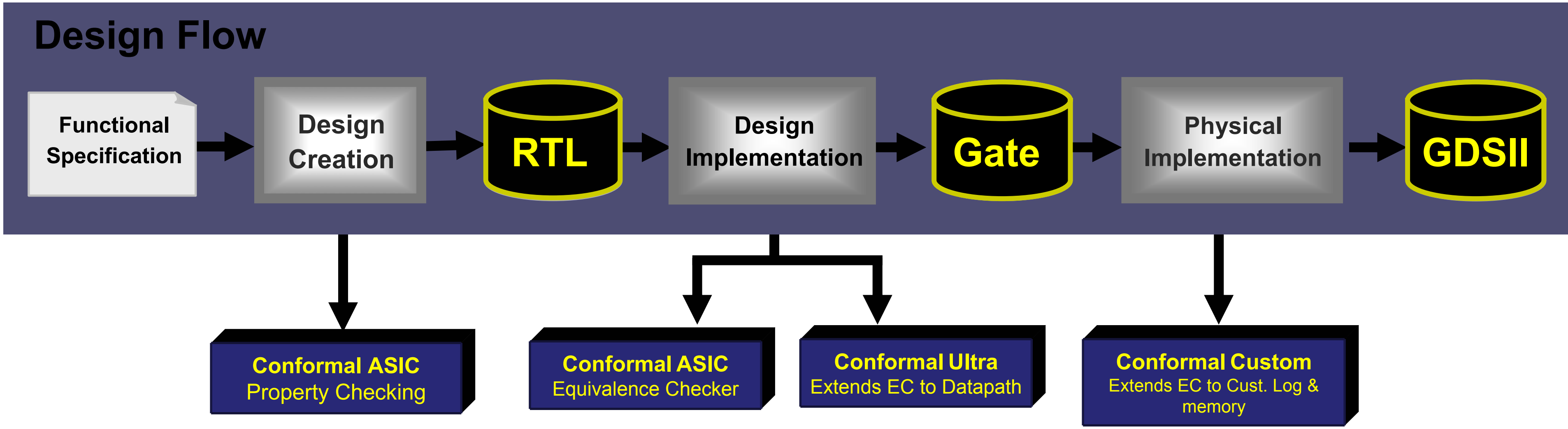
Verplex ensures that the functionality of the final layout matches that of the Golden RTL



Our formal technology enables you to catch bugs

- ✧ **Earlier** Target a class of bugs typically found with simulation late in the design cycle
- ✧ **Faster** Formal is magnitudes faster than simulation
- ✧ **Easier** No test vectors required, easy debugging

VERPLEX Conformal Solution Overview



Design Flow

Functional Specification → Design Creation → RTL → Design Implementation → Gate → Physical Implementation → GDSII

Conformal ASIC
Property Checking

Conformal ASIC
Equivalence Checker

Conformal Ultra
Extends EC to Datapath

Conformal Custom
Extends EC to Cust. Log & memory

Verifies
- Clock domain crossings
- Semantic consistency
- Structural consistency

RTL-gate Equivalence Checking (EC)

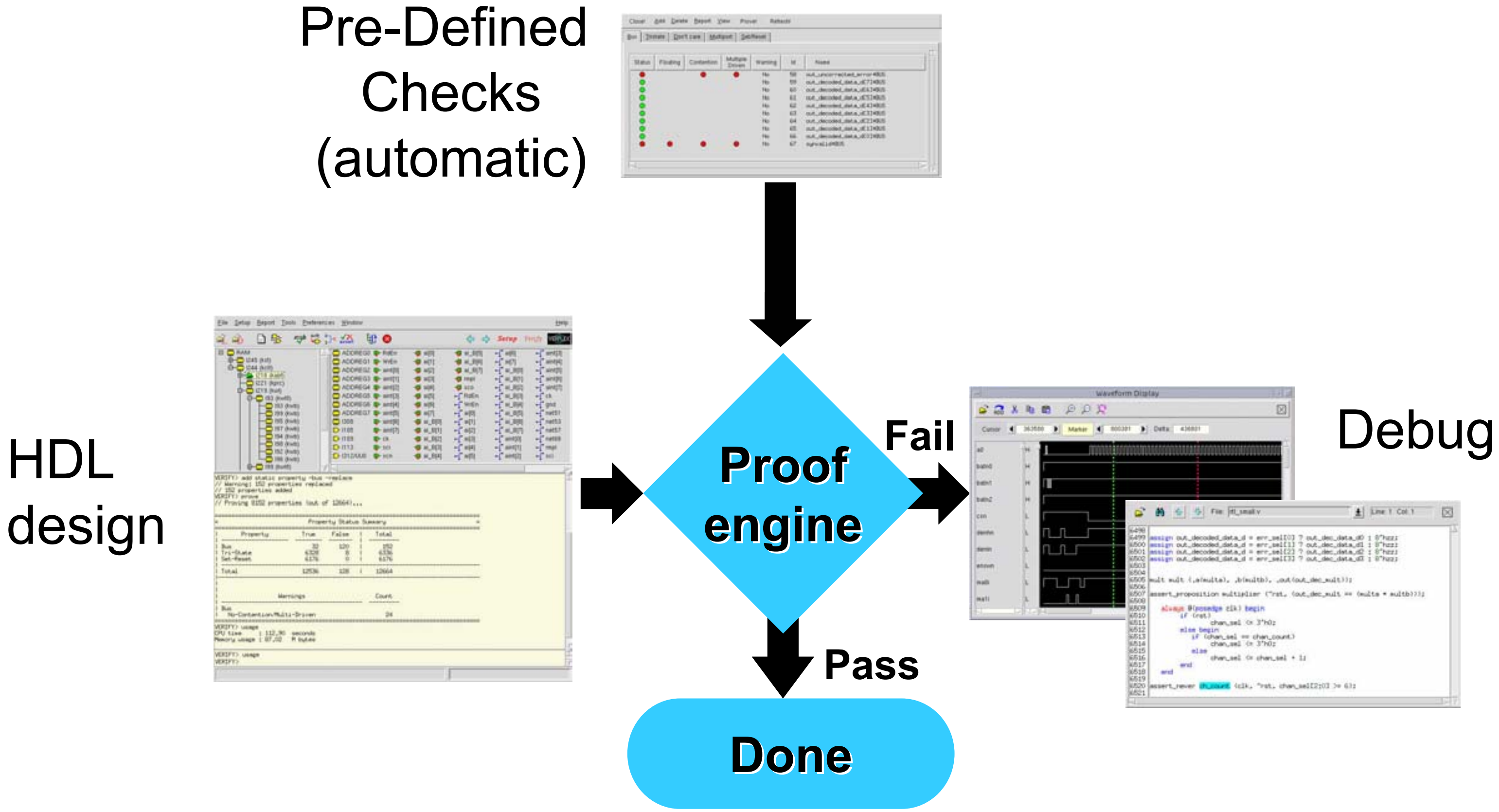
Verifies
- Clock domain crossings
- Structural consistency

RTL-gate Equivalence Checking (EC)

Extends EC to custom logic and memory

Design it golden - keep it golden.^{TM}

Assertion Based Property Checker



Pre-Defined Checks (automatic)

HDL design

Proof engine

Fail

Debug

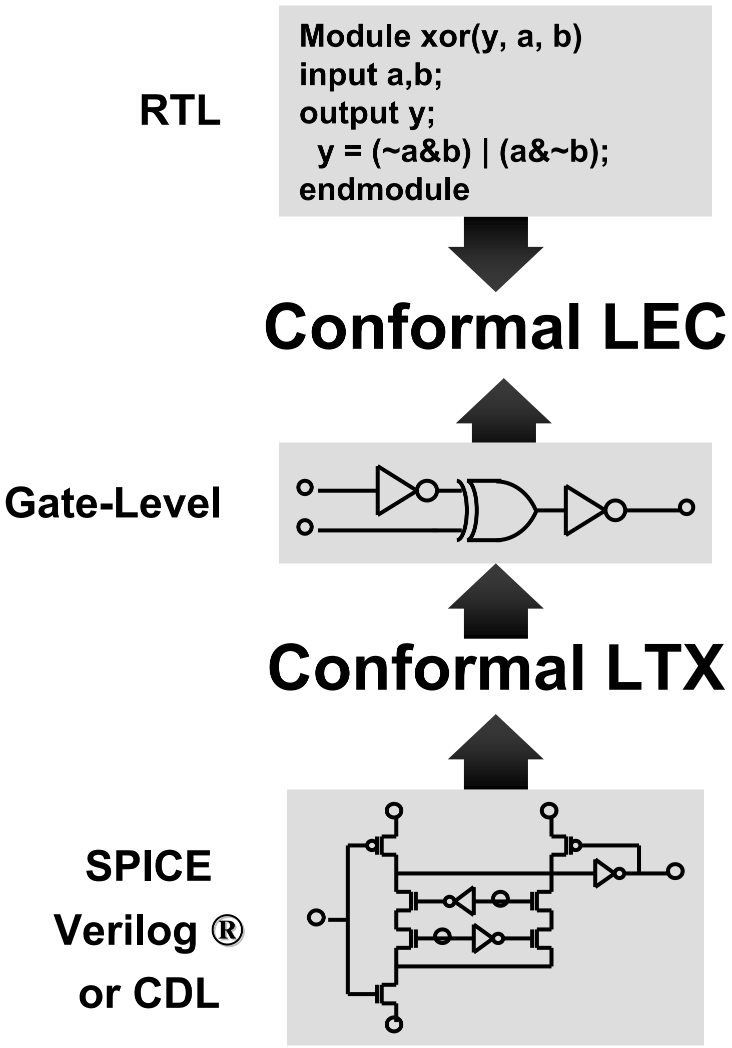
Pass

Done

Assertion Based Functional Checker

- Pre-Defined Checks (PDC)
  - Automatically checks for
    - Asynchronous clock domain crossing
      - Clock domain analysis
      - Structural checks for metastability prevention
      - Functional checks for data stability
    - Semantic inconsistency
      - full_case/parallel_case
      - range-overflow conditions
      - X-assignment
    - Structural inconsistency
      - Tri-state enable stuck-at conditions
      - Simultaneous set-reset conflict
      - Conflicting values on multi-port latches
      - Bus contention and bus floating

Conformal LTX Logic Transistor Extraction



RTL

```
Module xor(y, a, b)
input a,b;
output y;
  y = (~a&b) | (a&~b);
endmodule
```

Conformal LEC

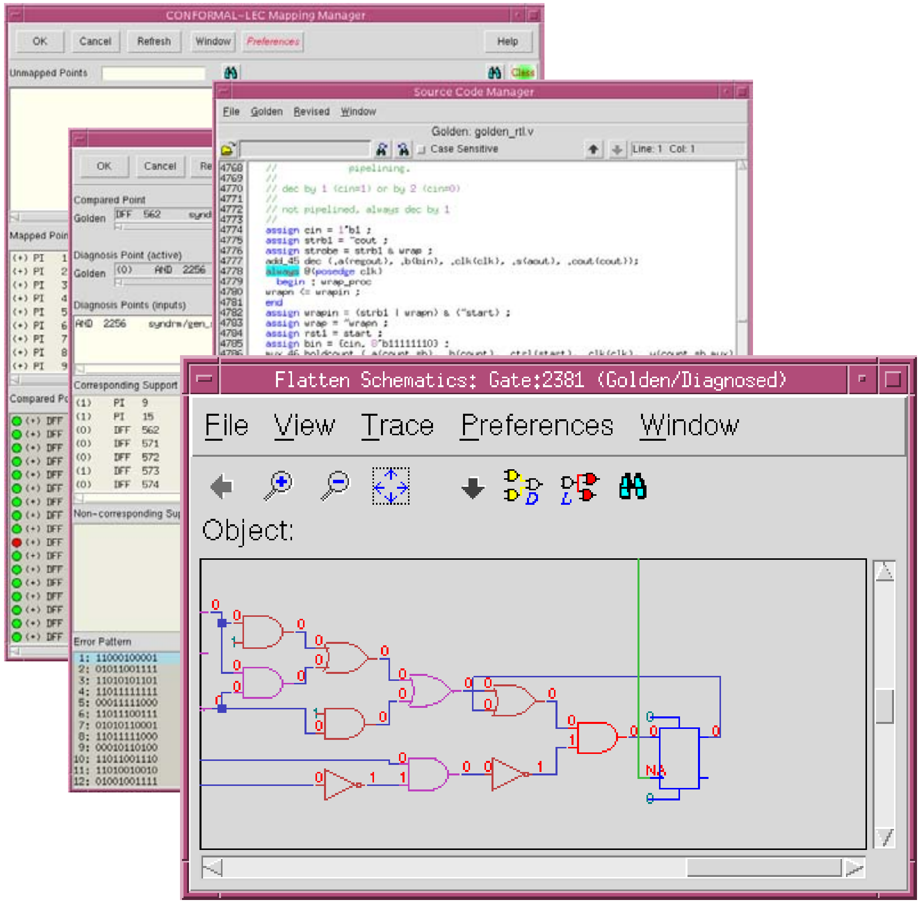
Gate-Level

Conformal LTX

SPICE
Verilog ®
or CDL

- Abstract transistor-level circuitry into gate-level
- Enable equivalence checking

Conformal LEC Logic Equivalence Checker



```
//                pipelining.
//
// dec by 1 (cin=1) or by 2 (cin=0)
//
// not pipelined, always dec by 1
//
assign cin = 1'b1 ;
assign strb1 = ~cout ;
assign strobe = strb1 & wrap ;
add_45 dec (.a(regout), .b(bin), .clk(clk), .s(aout), .cout(cout));
always @(posedge clk)
  begin : wrap_proc
wrapn <= wrapin ;
end
assign wrapin = (strb1 | wrapn) & (~start) ;
assign wrap = ~wrapn ;
assign rst1 = start ;
assign bin = {cin, 8'b11111110} ;
```

```
(+) PI 1
(+) PI 2
(+) PI 3
(+) PI 4
(+) PI 5
(+) PI 6
(+) PI 7
(+) PI 8
(+) PI 9
```

Compared Point
Golden: DFF 562 synd

Diagnosis Point (active)
Golden: (0) AND 2256

Diagnosis Points (inputs)
AND 2256 syndrm/gen_

Corresponding Support

```
(1)  PI   9
(1)  PI   15
(0)  DFF  562
(0)  DFF  571
(0)  DFF  572
(1)  DFF  573
(0)  DFF  574
```

Non-corresponding Support

Error Pattern

```
 1: 11000100001
 2: 01011001111
 3: 11010101101
 4: 11011111111
 5: 00011111000
 6: 11011100111
 7: 01010110001
 8: 11011111000
 9: 00010110100
10: 11011001110
11: 11010010010
12: 01001001111
```

- Largest designs with the highest completion rate
  - Handles even complex logic
  - Verifies any step back to original RTL
  - Accommodates any combination: RTL and gate, flat or hierarchical
- Easiest to use
  - Annotated error patterns
  - RTL-gate cross highlighting
  - Logic cone schematic display
- Highest performance
  - Flat R2G compare: 1hr / 1M gates
  - Flat G2G compare: 20min / 1M gates

Conformal LEC Usage

LEC usage

- Introduction to LEC
  - Getting familiar with LEC
  - LEC modes of operation
  - LEC flow
- A typical session (flat compare)
- Hierarchical compare

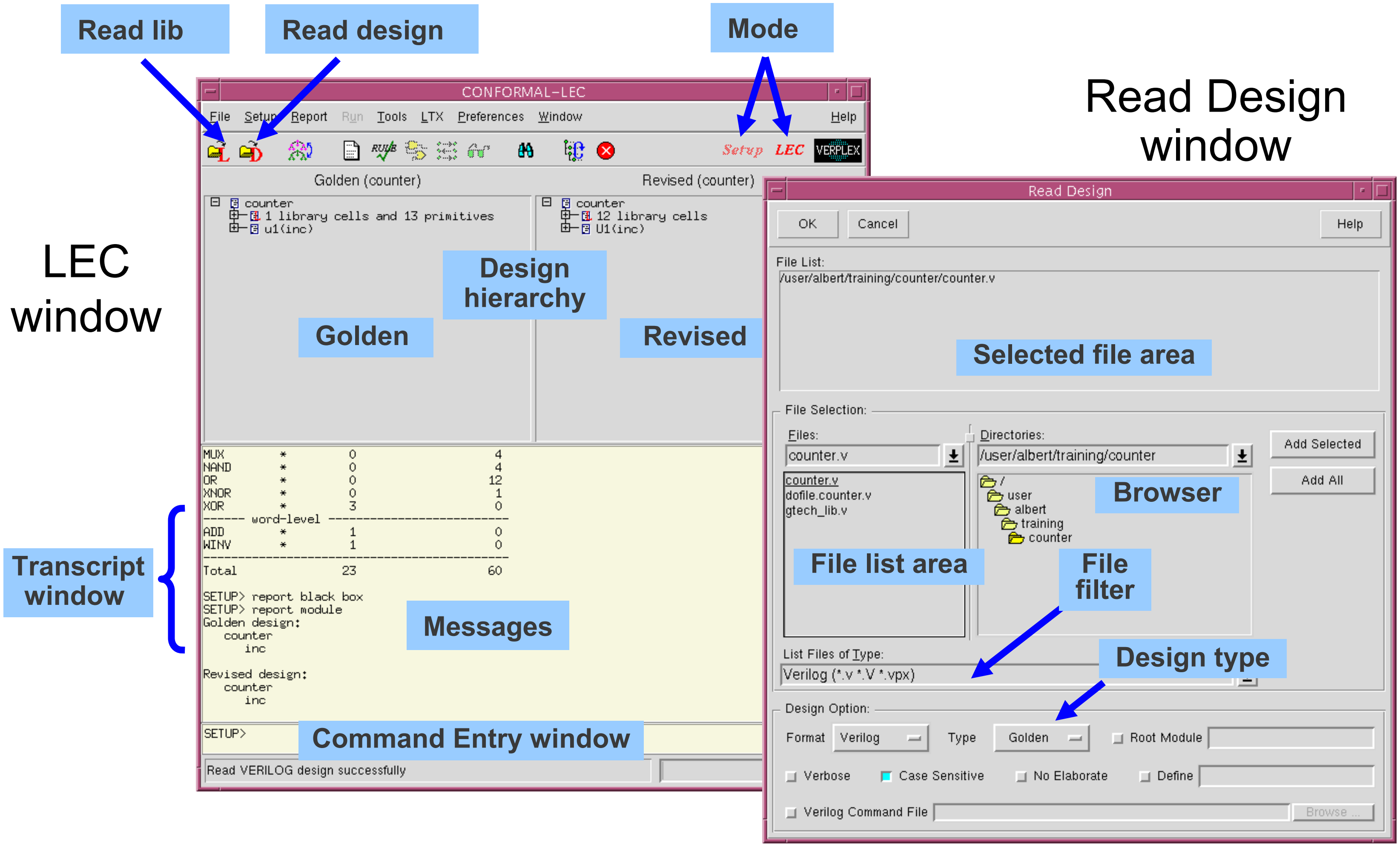
Getting familiar with LEC

- Starting and exiting LEC
- LEC GUI environment
- LEC command conventions
  - 3-2-1
  - Add/Delete/Report
- Convenient features
- Online help and documentation

Starting and exiting LEC

- Starting LEC
  - GUI mode: UNIX% `lec`
  - Non-GUI mode: UNIX% `lec -nogui`
  - Switch GUI & non-GUI modes (anytime during LEC session)
    - LEC> `set gui [on | off]`
  - Batch mode:
    - UNIX % `lec -dofile <batch_file> -nogui`
    - LEC> `dofile <batch_file>`
- Exiting LEC
  - LEC> `exit -force`
  - LEC automatically closes all windows upon exit

GUI environment



Read lib

Read design

Mode

LEC window

Transcript window

Design hierarchy

Golden

Revised

Messages

Command Entry window

Read Design window

Selected file area

Browser

File list area

File filter

Design type

```
⊟ counter
  ⊞─ 1 library cells and 13 primitives
  ⊞─ u1(inc)
```

```
⊟ counter
  ⊞─ 12 library cells
  ⊞─ U1(inc)
```

```
MUX        *       0            4
NAND       *       0            4
OR         *       0           12
XNOR       *       0            1
XOR        *       3            0
------ word-level -------------------------
ADD        *       1            0
WINV       *       1            0
-------------------------------------------
Total             23           60

SETUP> report black box
SETUP> report module
Golden design:
   counter
      inc

Revised design:
   counter
      inc
```

```
SETUP>
```

Read VERILOG design successfully

File List:
/user/albert/training/counter/counter.v

File Selection:

Files: counter.v

counter.v
dofile.counter.v
gtech_lib.v

Directories: /user/albert/training/counter

```
/
  user
    albert
      training
        counter
```

List Files of Type: Verilog (*.v *.V *.vpx)

Design Option:

Format: Verilog   Type: Golden   □ Root Module

□ Verbose   ☑ Case Sensitive   □ No Elaborate   □ Define

□ Verilog Command File

Command: 3-2-1 convention

- Most three-word commands can be abbreviated

Example:

```
SETUP> add pin constraint 0 scan_en
```

can be written as:

```
SETUP> add pi c 0 scan_en
```

Commands	3-2-1 convention
SET LOg File	set lo f
ADD PIn Constraint	add pi c
ADD INstance Constraint	add in c

- If 3-2-1 convention does not produce unique command, more letters may be added

Add/Delete/Report convention

For every ADD command, there is a corresponding REPORT and DELETE command

For example, if you specify:

```
SETUP> ADD PIn Constraint 0 scan_en
```

To report all pin constraints:

```
SETUP> REPort PIn Constraint
```

To remove the added constraint:

```
SETUP> DELete PIn Constraint scan_en
```

Convenient features

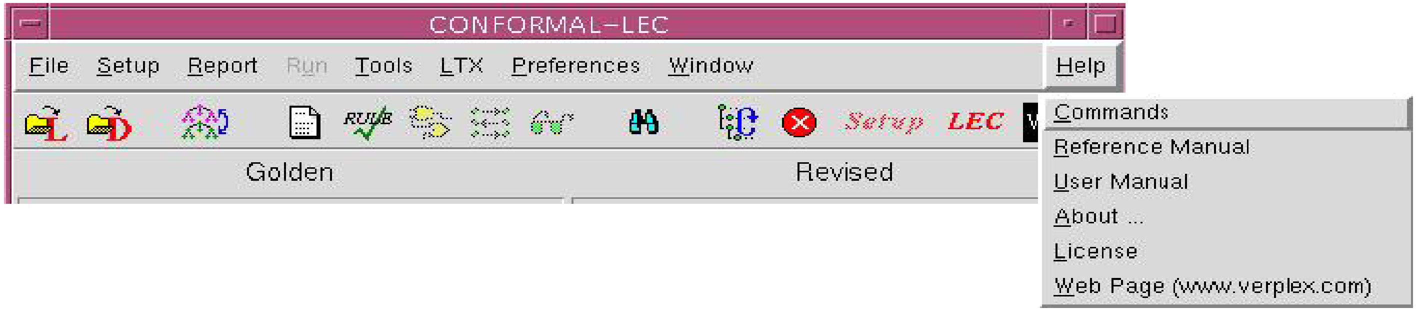
- Rerun previous command
  - Up and down arrow keys help avoid retyping previous commands
- Alias
  - Alias command usage: `add alias <name> <string>`
  - Example alias in *.conformal_lec* file:

    ```
    add alias setup set system mode setup
    ```

  - Initial command file, *.conformal_lec,* executed in the following order (3.4.0.a or later):
    - a. Verplex install directory: $VERPLEX_HOME/lib/.conformal_lec
    - b. Home directory: ~/.conformal_lec
    - c. Current working directory: ./.conformal_lec

Online help and documentation

- Access location of the PDF format manuals:



  - **Commands**: List of all command options, usage, examples, and related commands
  - **Reference Manual**: PDF format file of command reference with detailed command usage and definitions
  - **User Manual**: PDF format file with information related to the product (for example; installation, process flow, and GUI)

- Directory location of the PDF format manuals:
  - $VERPLEX_HOME/doc

LEC usage

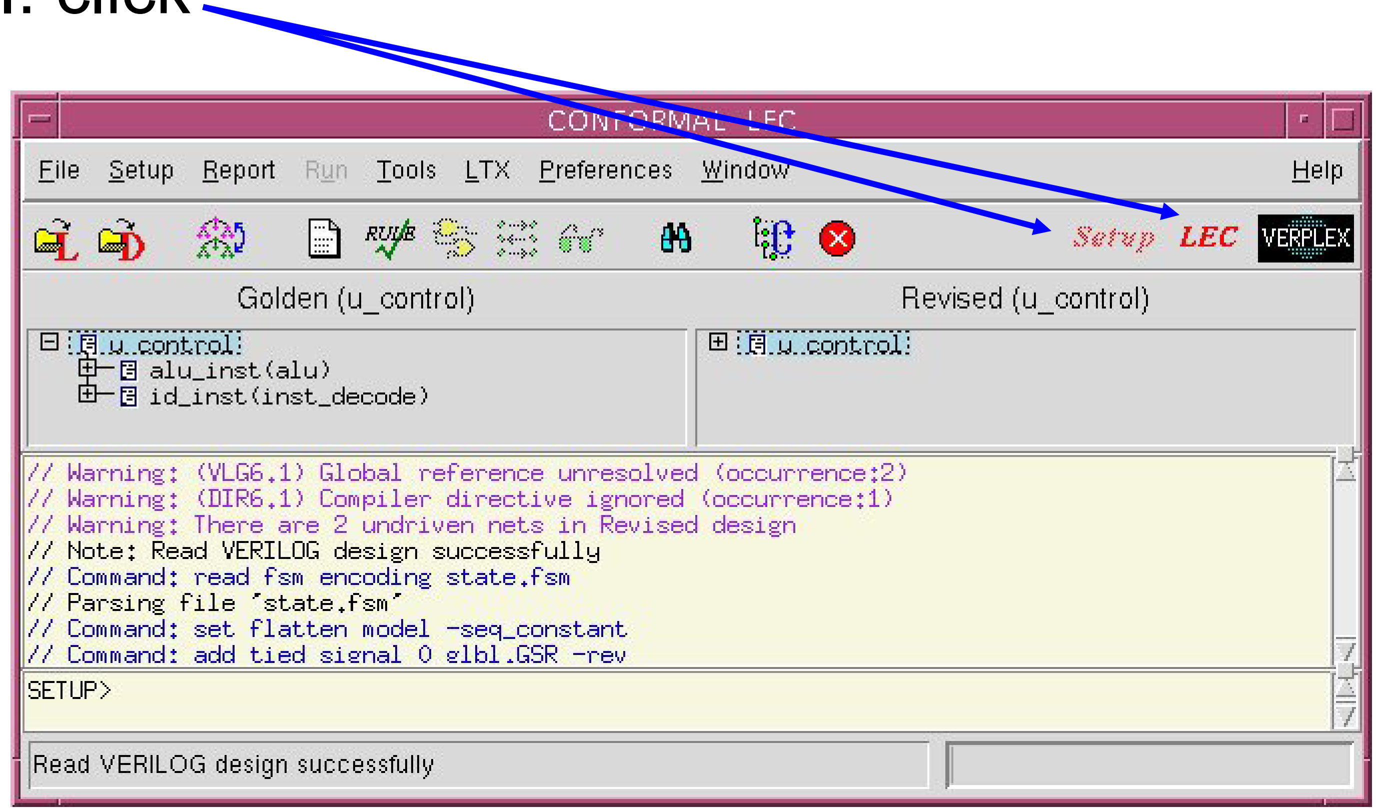
- Introduction to LEC
  - Getting familiar with LEC
  - LEC modes of operation ←
  - LEC flow
- A typical session (flat compare)
- Hierarchical compare

Modes of operation

- SETUP mode
  - Saving LEC transcript to a log file
  - Specifying black boxes
  - Reading libraries and designs
  - Specifying design constraints
  - Specifying modeling directives
- LEC mode
  - Mapping process
    - Resolving unmapped key points
  - Compare process
    - Debugging non-equivalent key points
  - Report run statistics

Switching modes of operation

- GUI: click



```
// Warning: (VLG6.1) Global reference unresolved (occurrence:2)
// Warning: (DIR6.1) Compiler directive ignored (occurrence:1)
// Warning: There are 2 undriven nets in Revised design
// Note: Read VERILOG design successfully
// Command: read fsm encoding state.fsm
// Parsing file 'state.fsm'
// Command: set flatten model -seq_constant
// Command: add tied signal 0 glbl.GSR -rev
SETUP>
```

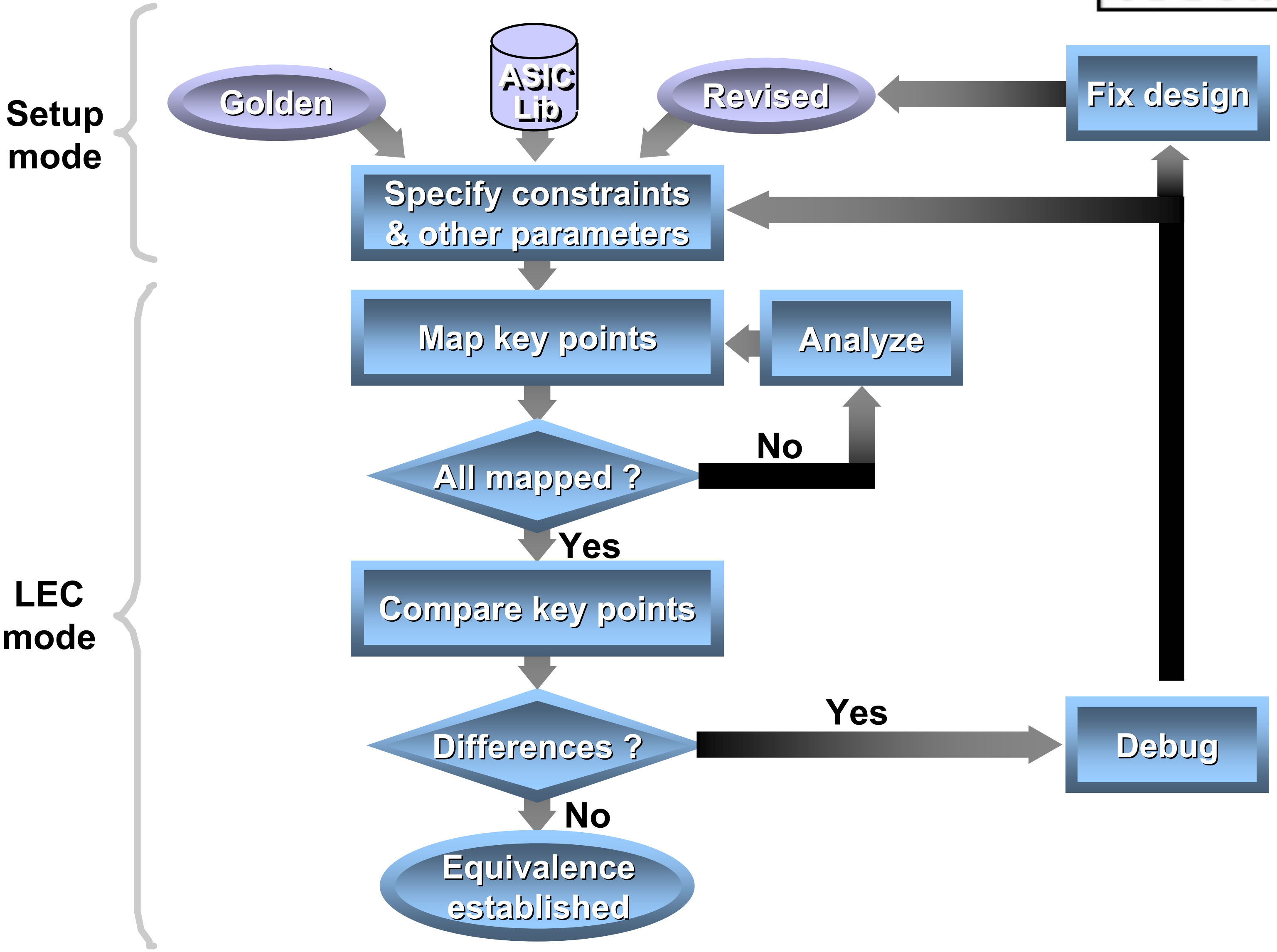
Read VERILOG design successfully

- Command: `set system mode [lec | setup]`

LEC usage

- Introduction to LEC
  - Getting familiar with LEC
  - LEC modes of operation
  - LEC flow
- A typical session (flat compare)
- Hierarchical compare

LEC flow


Setup mode
Golden
ASIC Lib
Revised
Fix design
Specify constraints & other parameters
LEC mode
Map key points
Analyze
All mapped ?
No
Yes
Compare key points
Differences ?
Yes
Debug
No
Equivalence established

LEC usage

- Introduction to LEC
  - Getting familiar with LEC
  - LEC modes of operation
  - LEC flow
- A typical session (flat compare)
- Hierarchical compare

A typical session (flat compare)

SETUP

- Saving LEC transcript to a log file
- Specifying black boxes
- Reading libraries and designs
- Specifying design constraints
- Specifying modeling directives
- Switching to LEC mode

LEC

- Mapping process
  - Resolving unmapped key points
- Compare process
  - Debugging non-equivalent key points
- Report run statistics

Saving LEC transcript to a log file

Specify the name of the log file where LEC session transcript is to be written

Example 1:

```
set log file logfile -replace
...
```

Example 2:

```
set log file logfile.$LEC_VERSION -replace
...
```

Note: $LEC_VERSION is automatically set to the current LEC version

A typical session (flat compare)

SETUP

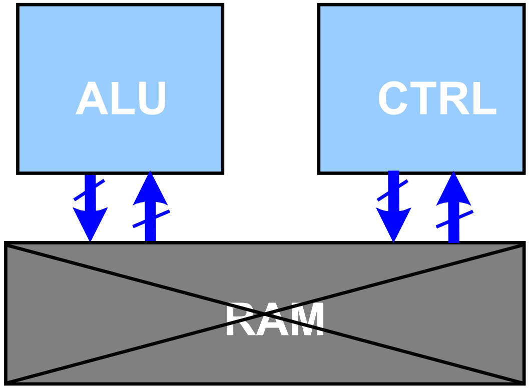
- Saving LEC transcript to a log file
- Specifying black boxes
- Reading libraries and designs
- Specifying design constraints
- Specifying modeling directives
- Switching to LEC mode

LEC

- Mapping process
  - Resolving unmapped key points
- Compare process
  - Debugging non-equivalent key points
- Report run statistics

Black box

- Module types
  - RAM, ROM, analog, behavioral, or any module you don’t want to verify
- Allows wildcard(*) specification
- Black box connections are verified



Specifying black boxes

- Execute black boxing command before module is read in

```
set log file logfile.$LEC_VERSION -replace
setenv LIB /user1/lib/verilog/
add notranslate module *ram* hstm -full -both
read library $LIB/verilog/*.v -verilog -both
...
```

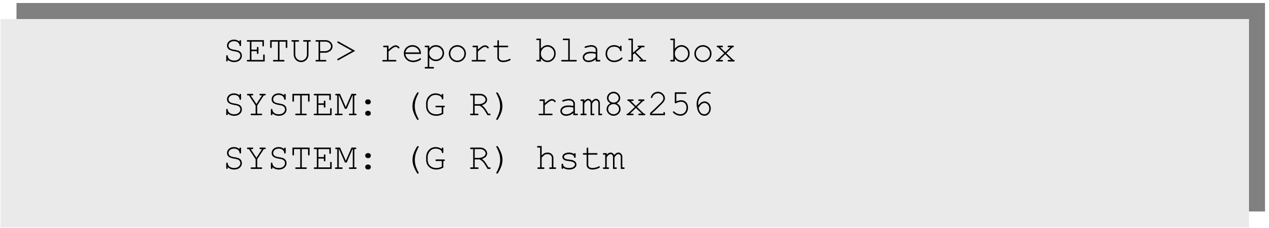
- For missing module, create one with just input/output declaration

```
module sram1(in1, in2, out2, out2);
input in1, in2;
output out1, out2;
      //empty
endmodule
```

Reporting black boxes

- SETUP> report black box

LEC window



```
SETUP> report black box
SYSTEM: (G R) ram8x256
SYSTEM: (G R) hstm
```

- Check for unbalanced black boxes in the Golden and Revised

A typical session (flat compare)

SETUP

- Saving LEC transcript to a log file
- Specifying black boxes
- Reading libraries and designs
- Specifying design constraints
- Specifying modeling directives
- Switching to LEC mode

LEC

- Mapping process
  - Resolving unmapped key points
- Compare process
  - Debugging non-equivalent key points
- Report run statistics

Supported formats

- Library
  - Verilog (standard simulation libraries)
  - Liberty™
- Design (Synthesizable RTL, gate, transistor)
  - Verilog
  - VHDL
  - SPICE

Reading Verilog library & design: Syntax

- Read library:

```
read library <filename …>
         [-verilog | -liberty]
         [-replace | -append]
         [-both | -golden | -revised]
```

- Read design:

```
read design <filename …>
         [-file <verilog_command_file>]
         [-verilog | -verilog2k]
         [-golden | -revised]
```

Reading Verilog library & design: Method 1

- Reading files via command line
- Allows wildcard (*) specification

```
set log file logfile.$LEC_VERSION -replace
setenv LIB /user1/lib/verilog/
read design cpu_rtl.v -verilog -golden
read library $LIB/*.v -verilog -revised
read design cpu_gate.v -verilog -revised
...
```

- Allows path search specification

```
add search path /user1/lib/verilog/ -lib -revised
read library *.v -verilog -revised
```

Reading Verilog library & design: Method 2

- Reading files via Verilog command file
- LEC reads only library cells being instantiated

```
set log file logfile.$LEC_VERSION -replace
setenv LIB /user1/lib/verilog/
read design cpu_rtl.v -verilog -golden
read design -file verilog.vc -verilog -revised
...
```

Content:

```
-y $LIB
cpu_gate.v
```

Syntax:

<File>...	Design Data File
-v <File>...	List of Library Files
-y <Directory>...	Library Directory(s)
+incdir+<dir_name>...	Include Directories
+libext+<extension>...	File Extension e.g ".v"
-yd <Directory>...	Design Directory(s)

Reading VHDL design: Syntax

```
read design -vhdl <filename …>
        [-map <library_name> <library_path>]
        [-mapfile <library_name> <filename …>]
        [-golden | -revised]
```

- `-map:` read in library files from the specified library_path for the specified library_name
- `-mapfile:` read in the specified library files for the specified library_name
- Multiple -map and -mapfile options are allowed

Reading VHDL design: Example

Content of mb_lock.vhd:

```
LIBRARY MB_LIB;
LIBRARY DP_LIB;
USE MB_LIB.ATTRIBUTES.ALL;
USE DP_LIB.MISC_PKG.ALL;
...
```

```
read design -vhdl RTL/mb_lock.vhd -golden \
            -map MB_LIB /user1/lib/vhdl/ \
            -mapfile DP_LIB ./vhdl_lib/misc_pkg.vhd
...
```

Reading mixed languages

- Mixed-language: Libraries

```
set log file logfile.$LEC_VERSION -replace
setenv LIB /user1/lib/
read library $LIB/verilog/*.v -verilog -golden
read library $LIB/vhdl/*.lib -liberty -append -golden
...
```

- Mixed-language: Designs

```
set log file logfile.$LEC_VERSION -replace
setenv LIB /user1/lib/
read library $LIB/verilog/*.v -verilog -golden
read design RTL/core.vhd -vhdl -noelab -golden
read design RTL/top.v -verilog -golden
...
```

HDL rule checks

- LEC displays HDL rule violations after parsing design

Transcript window

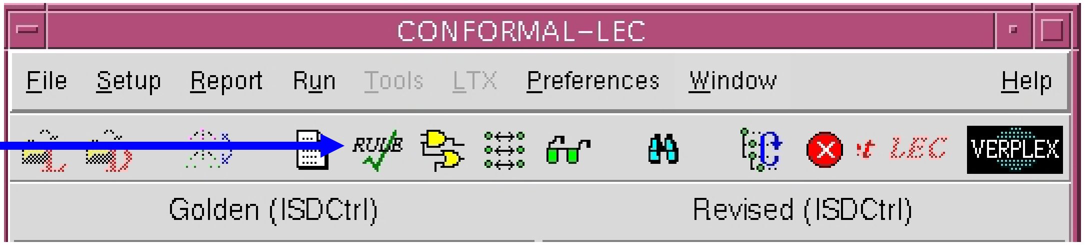
```
// Command: read design src/idsctrl.v -gold
// Check out vlg 3.0 license
// Parsing file src/idsctrl.v ...
// Golden root module is set to ‘idsctrl’
// Warning: (RTL2.3) externally defined signal reference not supported
// Warning: (RTL5.1) overlapped case items in parallel case statement
...
```

- Non-GUI mode

  SETUP> `report rule check <rule_name> -verbose`

- GUI mode

  Click to open



HDL rule check categories

- You can specify the severity level for each category: "Error", "Warning", "Note", or "Ignore" with command "SET RUle Handling"
  - **Directives**: Synthesis and Verplex directives
  - **Hierarchy**: Rules apply to hierarchical design
  - **Ignored**: Redundant constructs or statement not supported, and thus ignored by LEC
  - **RTL**: RTL-level Verilog or VHDL rules
  - **UDP**: User-defined primitives
  - **Verilog**: Designs written in Verilog language

A typical session (flat compare)

SETUP

- Saving LEC transcript to a log file
- Specifying black boxes
- Reading libraries and designs
- Specifying design constraints
- Specifying modeling directives
- Switching to LEC mode

LEC

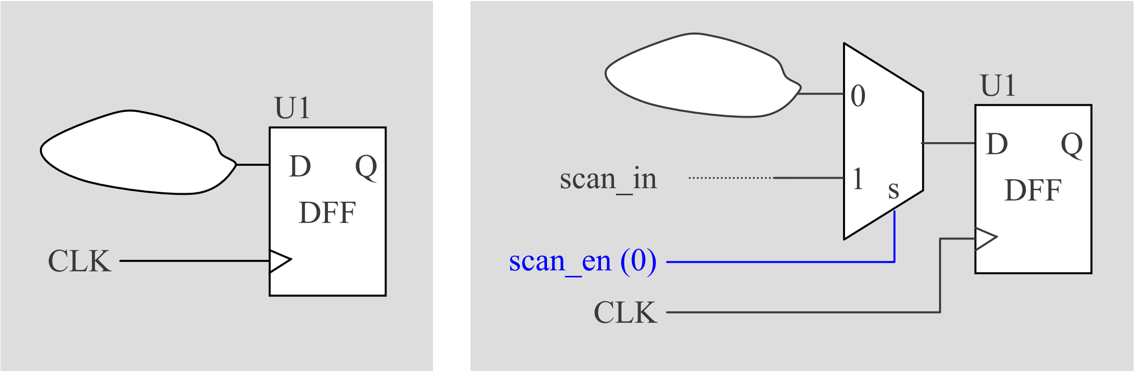
- Mapping process
  - Resolving unmapped key points
- Compare process
  - Debugging non-equivalent key points
- Report run statistics

Design constraints

- What are design constraints?
  - User’s inputs to control part of a design’s logic
- Purpose of constraints
  - To disable test logic (for example; scan and JTAG)
  - To specify one-hot or one-cold conditions
  - To specify relationships between pins
  - To constrain undriven signals
- Example of constraints
  - Pin constraint
  - Instance constraint
  - Pin equivalence
  - Tied signal
  - Undriven signal

Pin constraint

Specify the mode of circuit operation under which comparison will take place (for example; functional vs. scan operation)



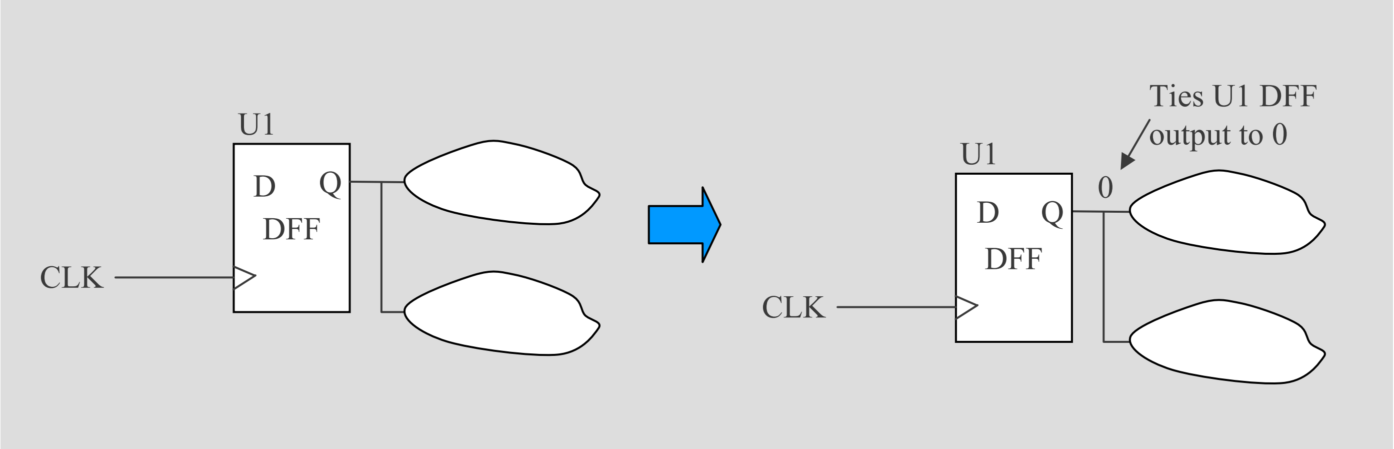
Golden

Revised

```
set log file logfile.$LEC_VERSION -replace
add notranslate module *sram* -library -both
read design cpu_rtl.v -verilog -golden
read design -file verilog.vc -verilog -revised
add pin constraint 0 scan_en -revised
...
```

Instance constraint

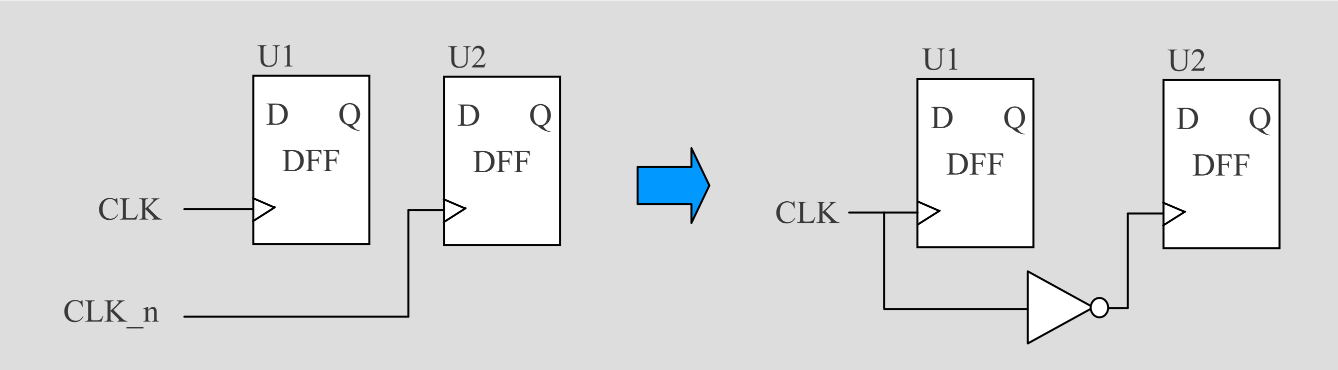
Apply to any internal DFF or D-latch output to Logic-0 or logic-1 (for example; JTAG registers)



```
...
add instance constraint 0 U1 -revised
...
```

Pin equivalence

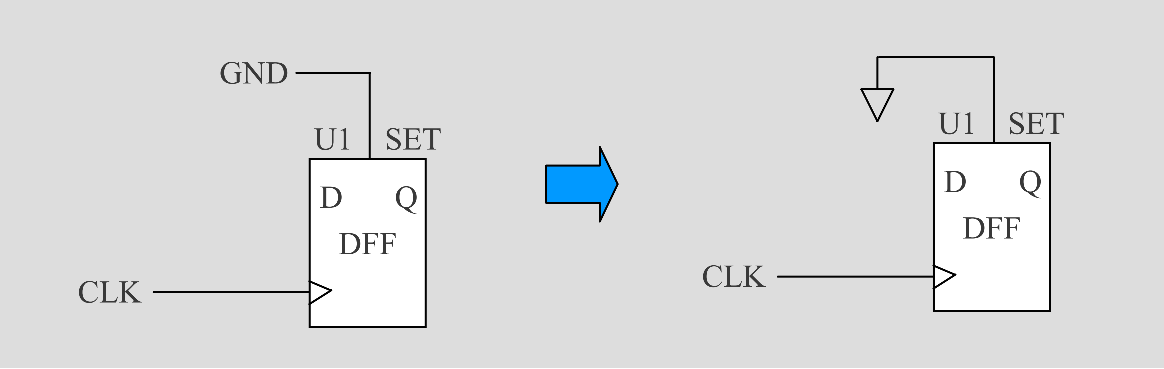
Specify the relationship (equivalent or inverted equivalent) between two or more primary input pins



```
...
add pin equivalence CLK –invert CLK_n -revised
...
```

Tied signal

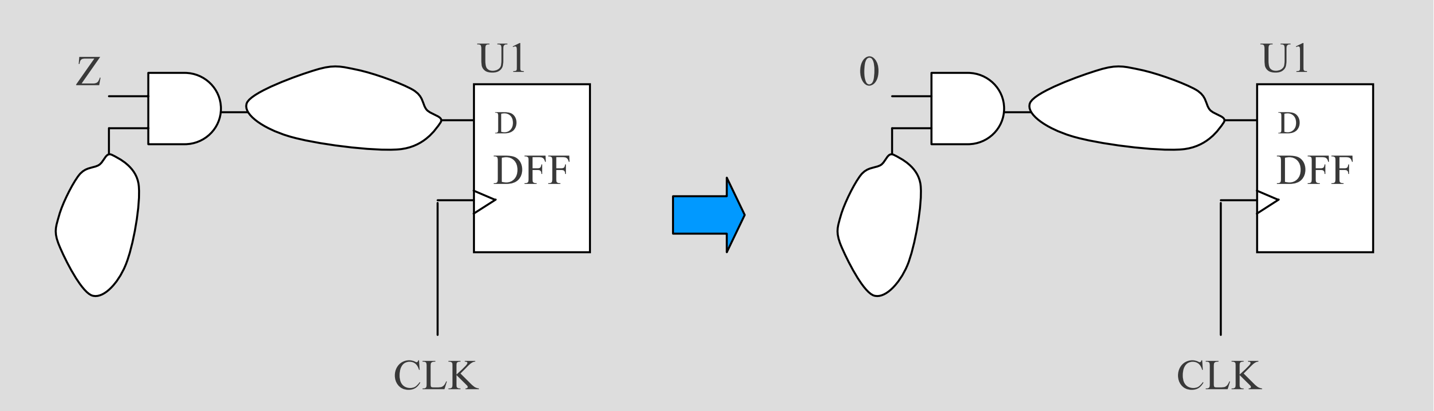
Specify floating nets or pins to be tied to Logic-0 or Logic-1 (for example; equate GND to 0 or VDD to 1)



```
. . .
add tied signal 0 GND -net -revised
. . .
```

Undriven signals

Specify the global behavior of floating signals in the designs (for example; ties all floating signals to a constant)



```
...
set undriven signal 0 -revised
...
```

A typical session (flat compare)

SETUP

- Saving LEC transcript to a log file
- Specifying black boxes
- Reading libraries and designs
- Specifying design constraints
- **Specifying modeling directives**
- Switching to LEC mode

LEC

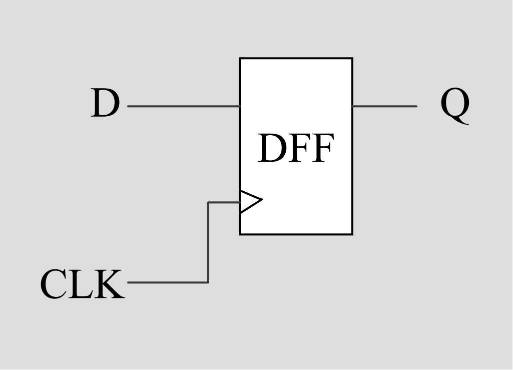
- Mapping process
  - Resolving unmapped key points
- Compare process
  - Debugging non-equivalent key points
- Report run statistics

Modeling directives

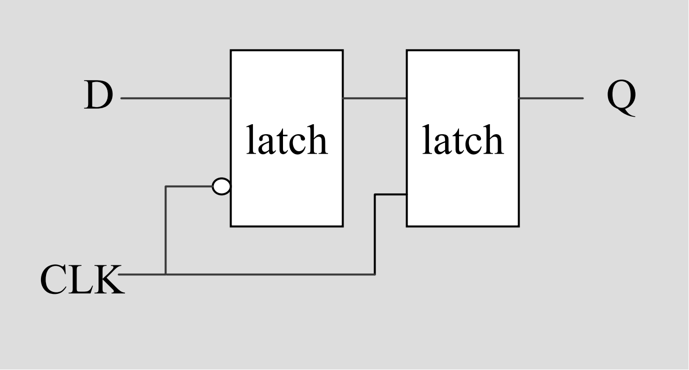
- In SETUP mode, user can specify directives to influence the way LEC models the design
- Modeling directives are needed to handle modeling styles specific to vendors' libraries or synthesis tools
- Modeling options:
  - Latch folding
  - Gated-clock
  - Sequential redundancy
  - Sequential constant
  - Latch transparent
  - Sequential merge (clock and data logic cones)
  - Grounded clock
  - DFF direct feedback

Latch folding: Problem

- Library cell uses 2 D-latches (master/slave) to model a DFF
- Causes mapping problem!



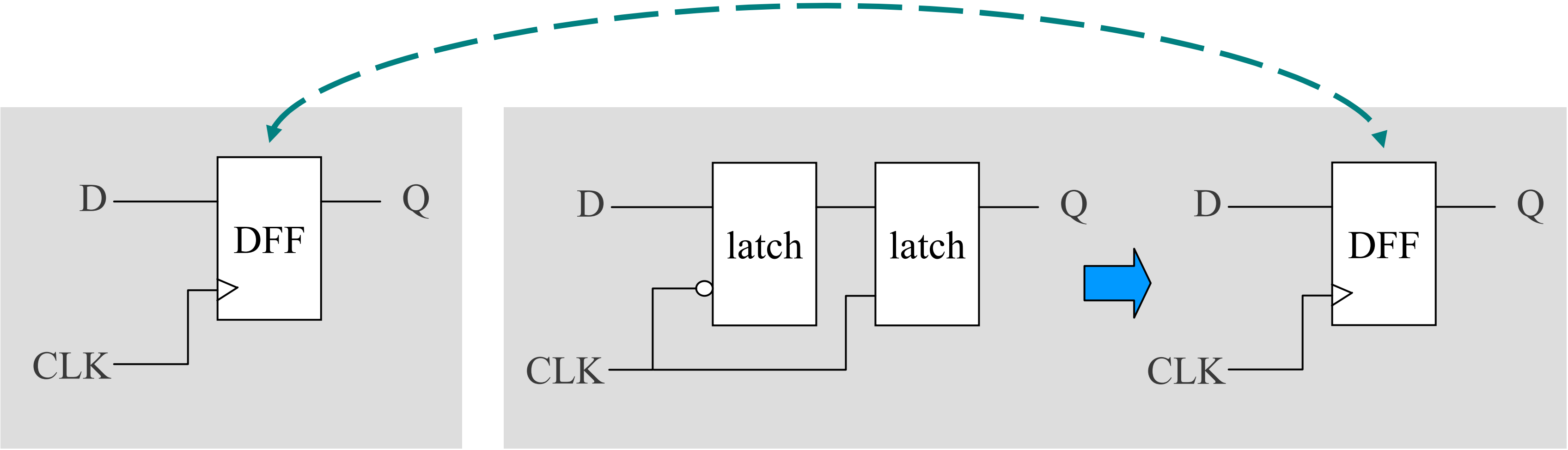
Golden



Revised

Latch folding: Solution

Fold the 2 D-latches into a DFF:



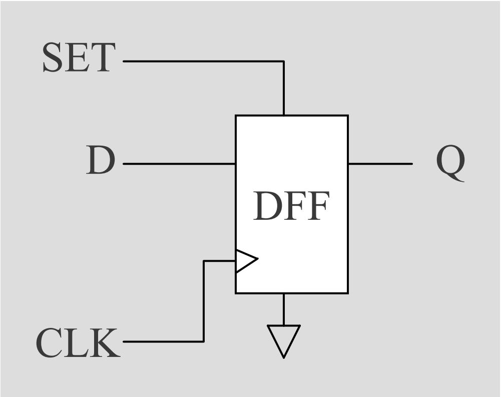
Golden

Revised

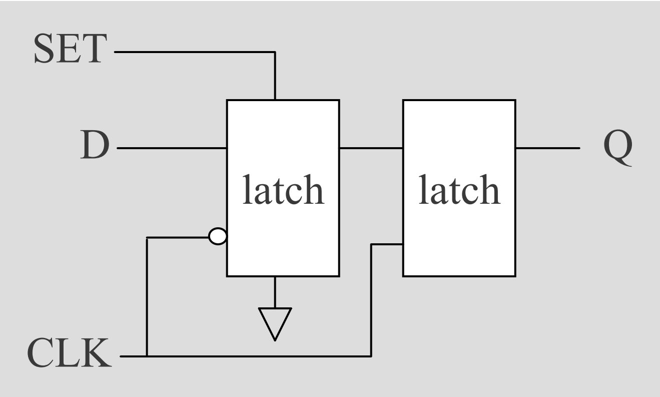
```
. . .
set flatten model -latch_fold
. . .
```

Latch folding (master): Problem

- Library cell uses two D-latches (master/slave), with only set/reset to the master D-latch
- Causes mapping problem!



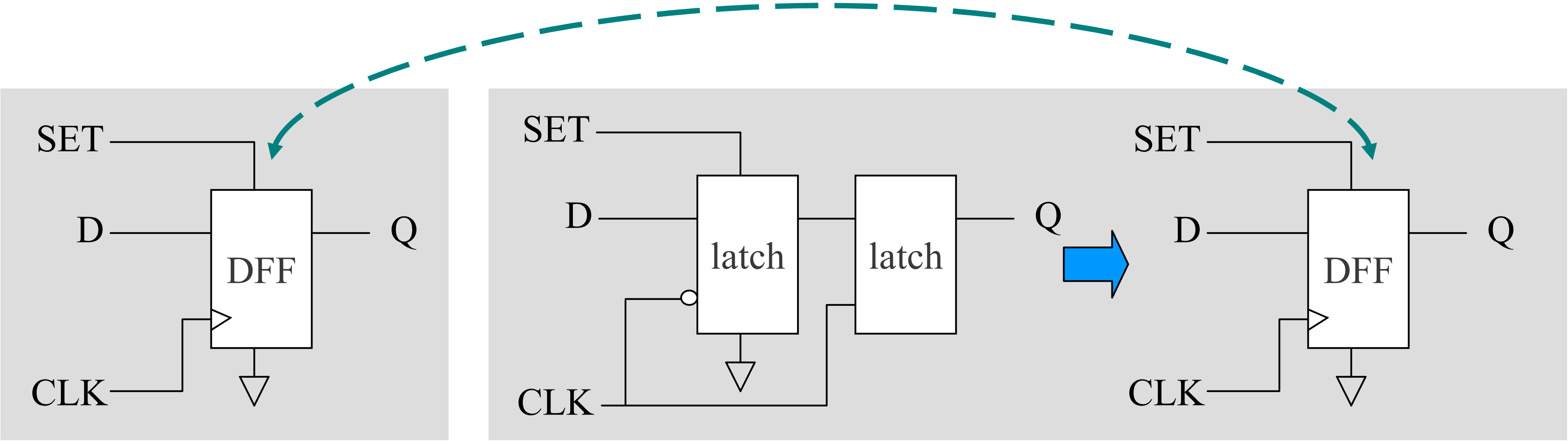
Golden



Revised

Latch folding (master): Solution

Fold the two D-latches into a DFF:



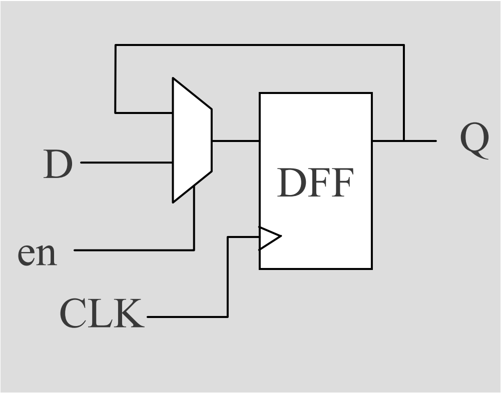
Golden

Revised

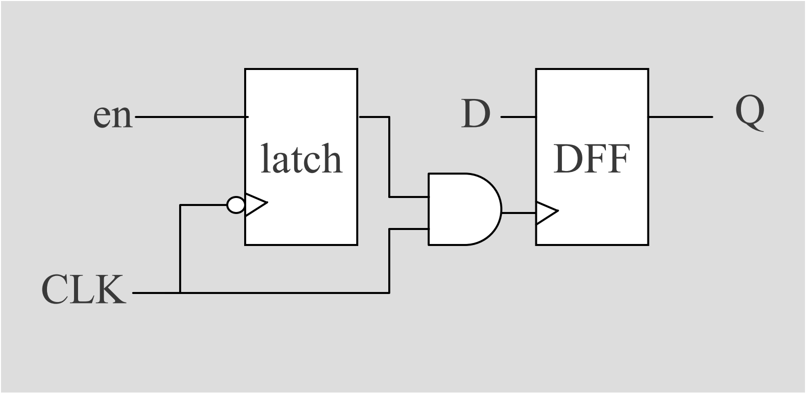
```
...
set flatten model -latch_fold_master
set flatten model -latch_fold
...
```

Gated-clock: Problem

- Power optimization tools create latch-based gated clock circuit
- Causes compare problem!



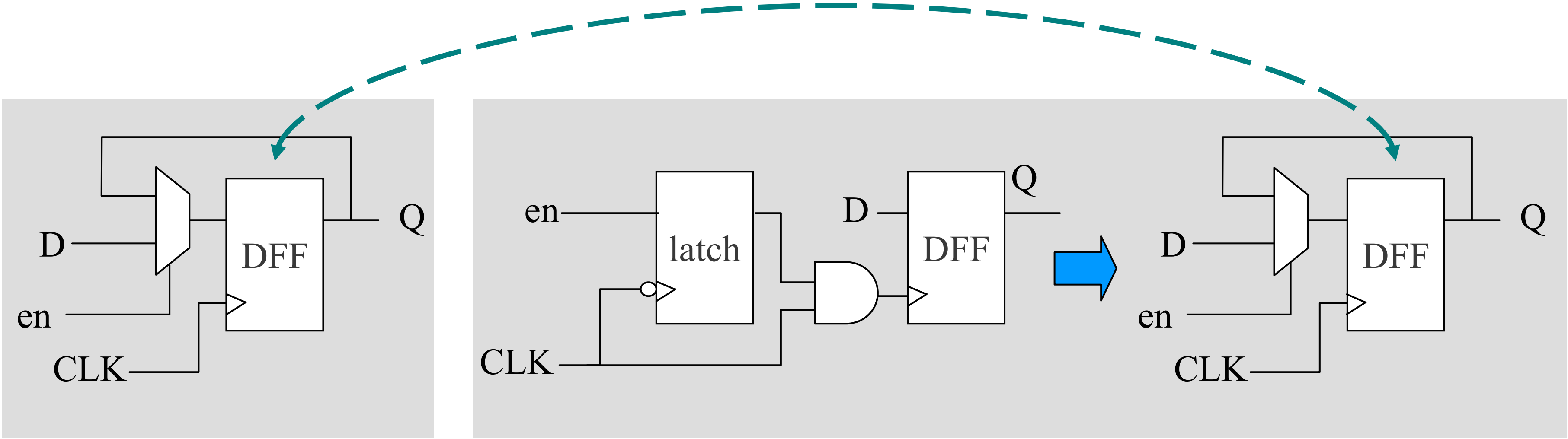
Golden



Revised

Gated-clock: Solution

Enable gated-clock learning:



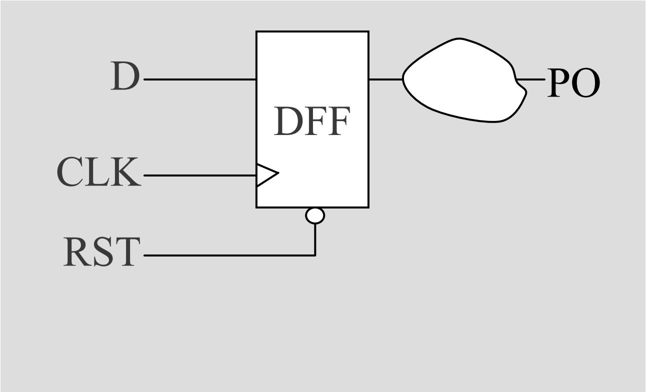
Golden

Revised

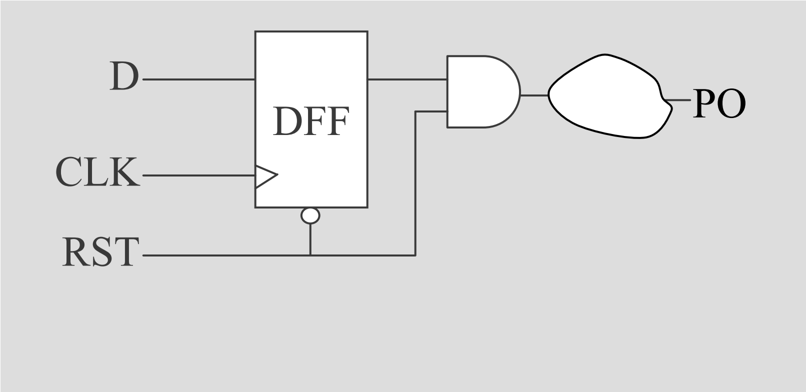
```
...
set flatten model -gated_clock
...
```

Sequential redundancy: Problem

- Redundant **AND** gate is introduced so that the reset signal takes effect right away
- Causes comparing problem!



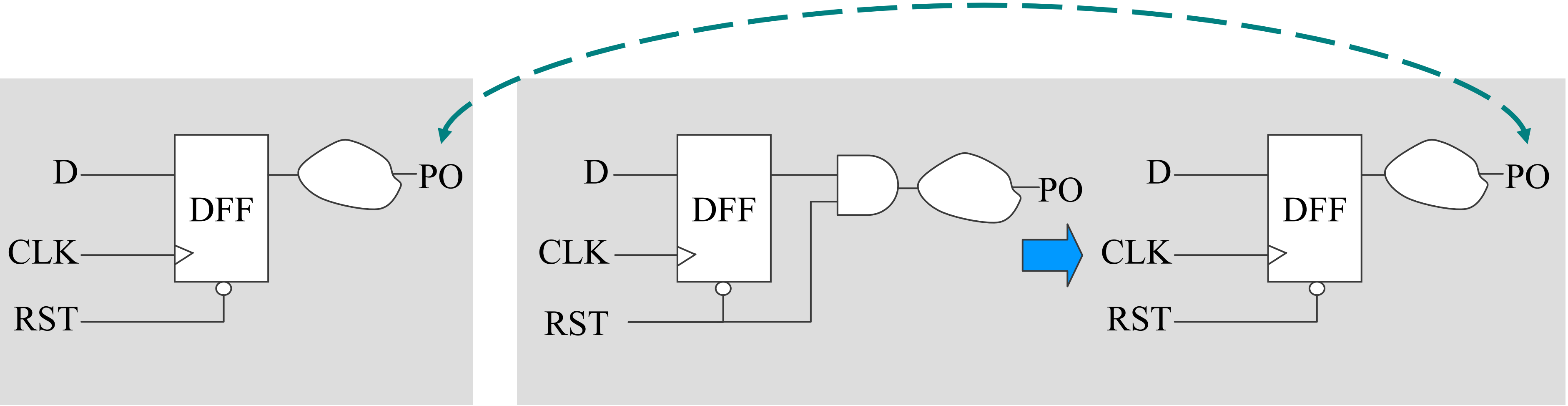
Golden



Revised

Sequential redundancy: Solution

Remove sequential redundancies:



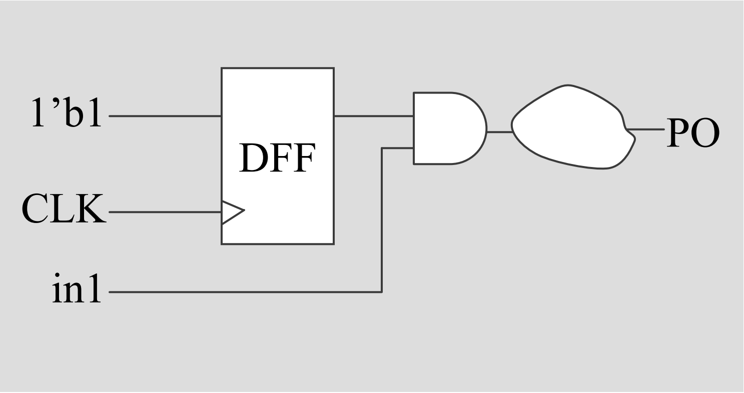
Golden

Revised

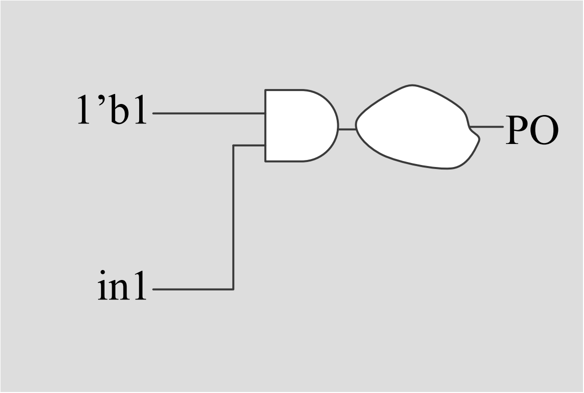
```
...
set flatten model –seq_redundant
...
```

Sequential constant: Problem

- Occurs due to the way the circuit is designed or a designer's preference to constrain the data port
- Causes comparing problem!



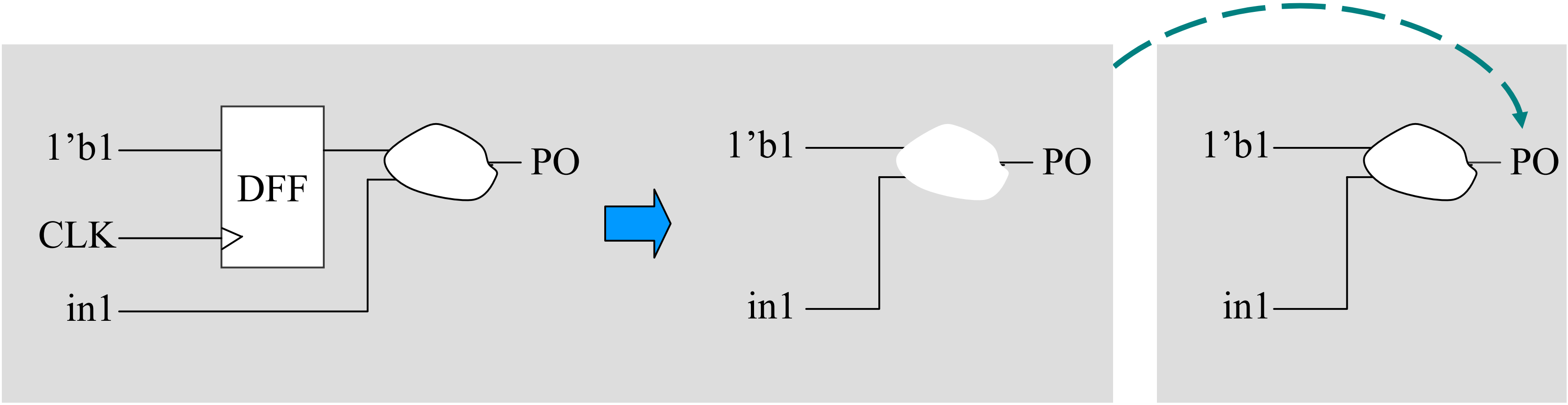
Golden



Revised

Sequential constant: Solution

Convert a DFF or a D-latch to a ZERO/ONE gate if the data port is set to 0/1:



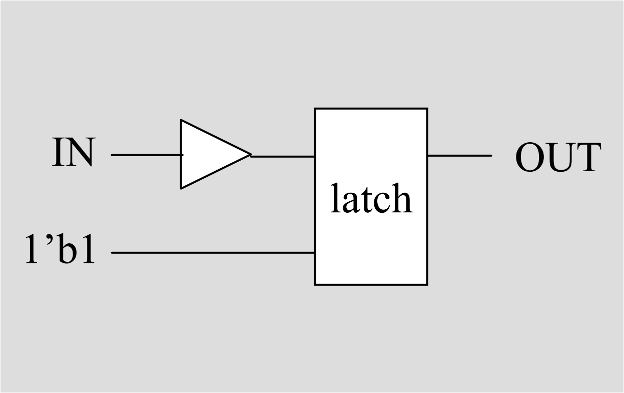
Golden

Revised

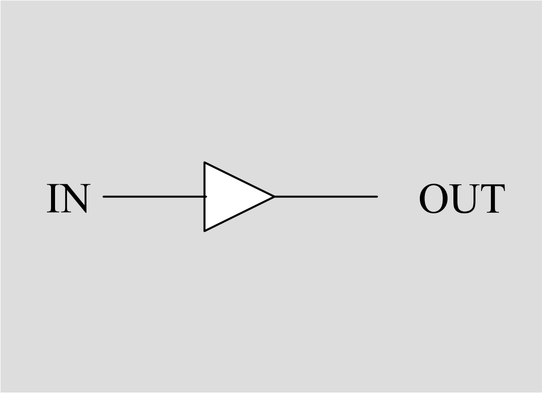
```
...
set flatten model -seq_constant
...
```

Latch transparent: Problem

- Designer’s choice to have a D-latch with its clock always enabled as a buffer (transparent latch)
- Causes comparing problem!



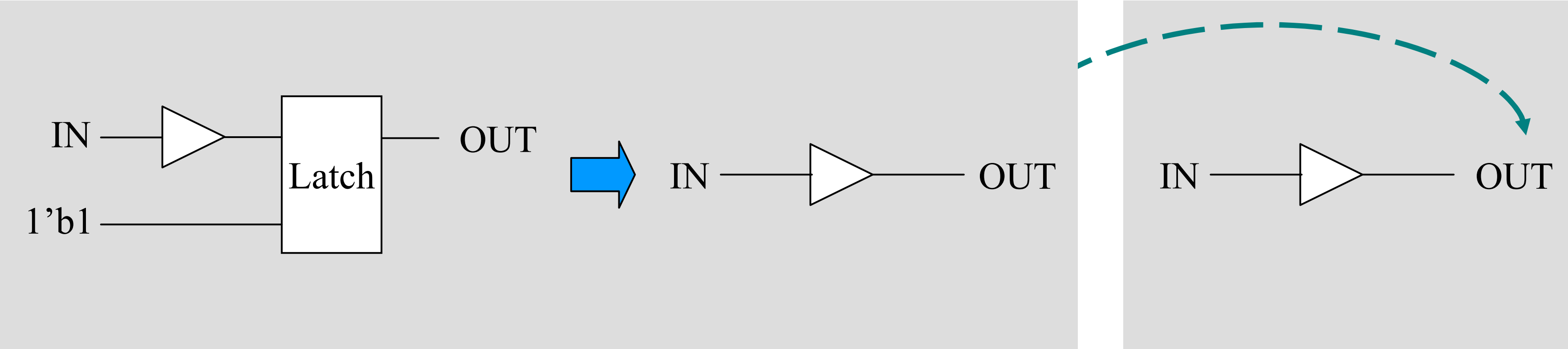
Golden



Revised

Latch transparent: Solution

Remodel D-latches whose clock ports are always enabled into buffers (transparent latches):



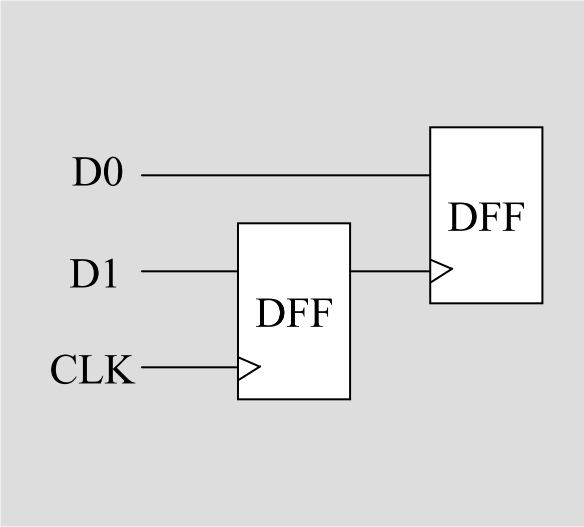
Golden

Revised

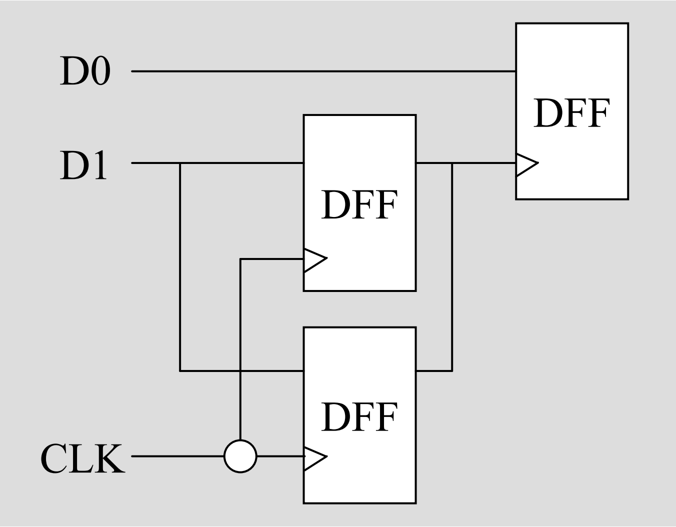
```
...
set flatten model -latch_transparent
...
```

Sequential merge (clock cone): Problem

- Duplicated register in the clock logic cone
- Causes mapping/comparing problem!



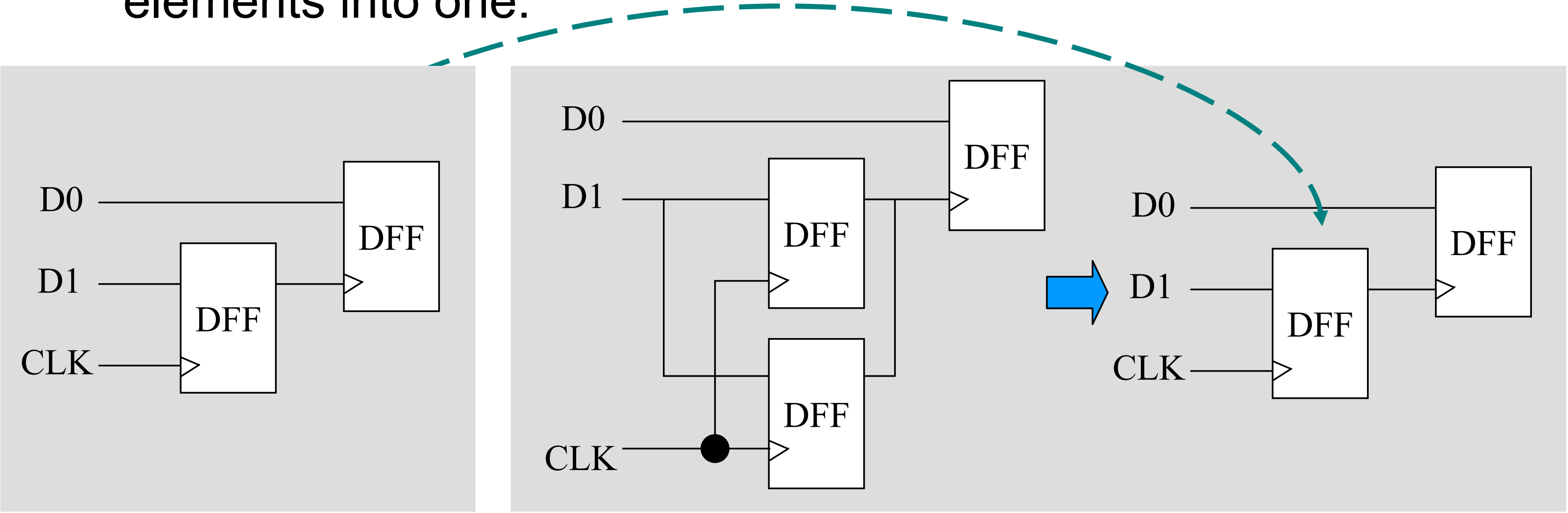
Golden



Revised

Sequential merge (clock cone): Solution

Merge a group of functionally equivalent sequential elements into one:



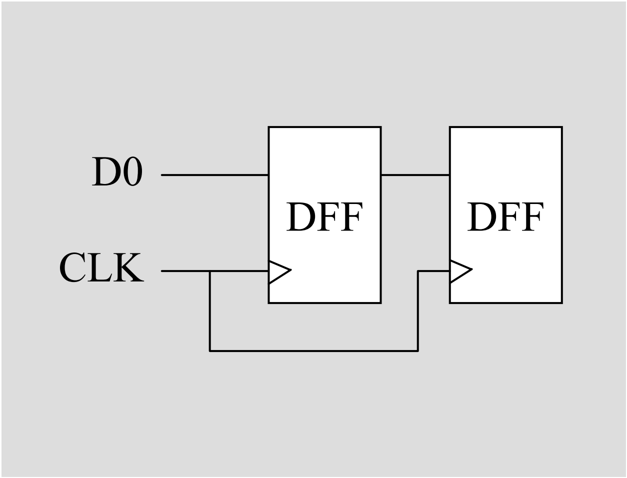
Golden

Revised

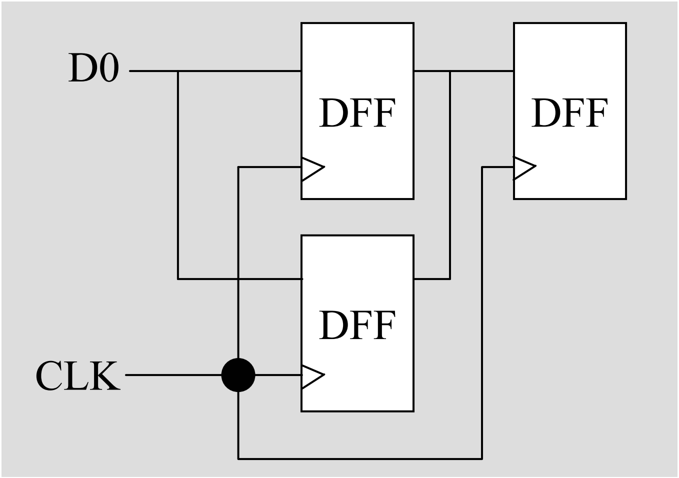
```
...
set flatten model -seq_merge
...
```

Sequential merge: Problem

- Duplicated register in the clock and/or data logic cone
- Causes comparing problem!



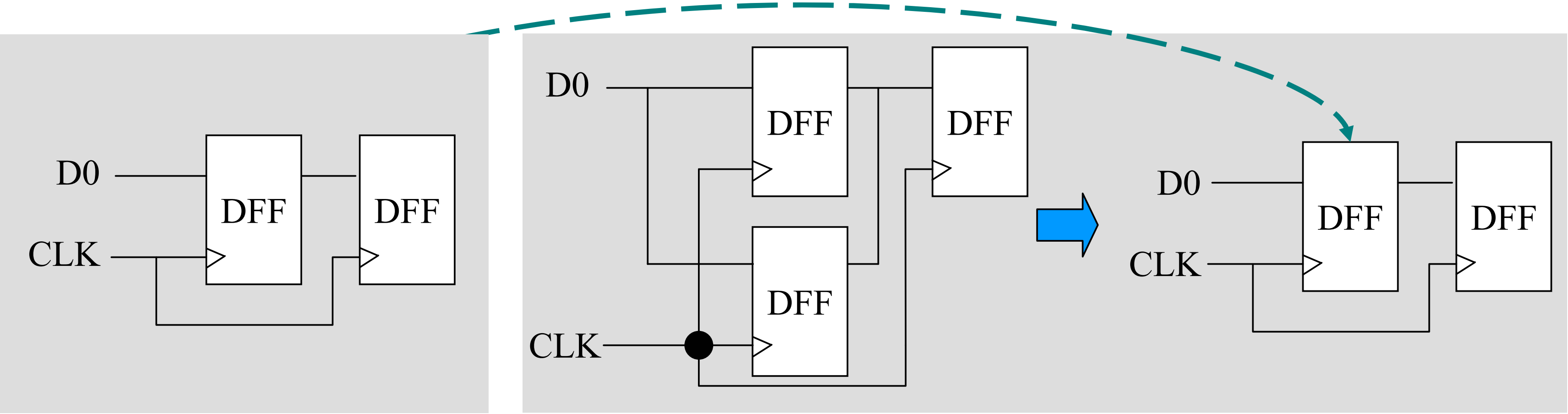
Golden



Revised

Sequential merge: Solution

- Merge a group of functionally equivalent sequential elements into one
- This directive is more "expensive" than -seq_merge



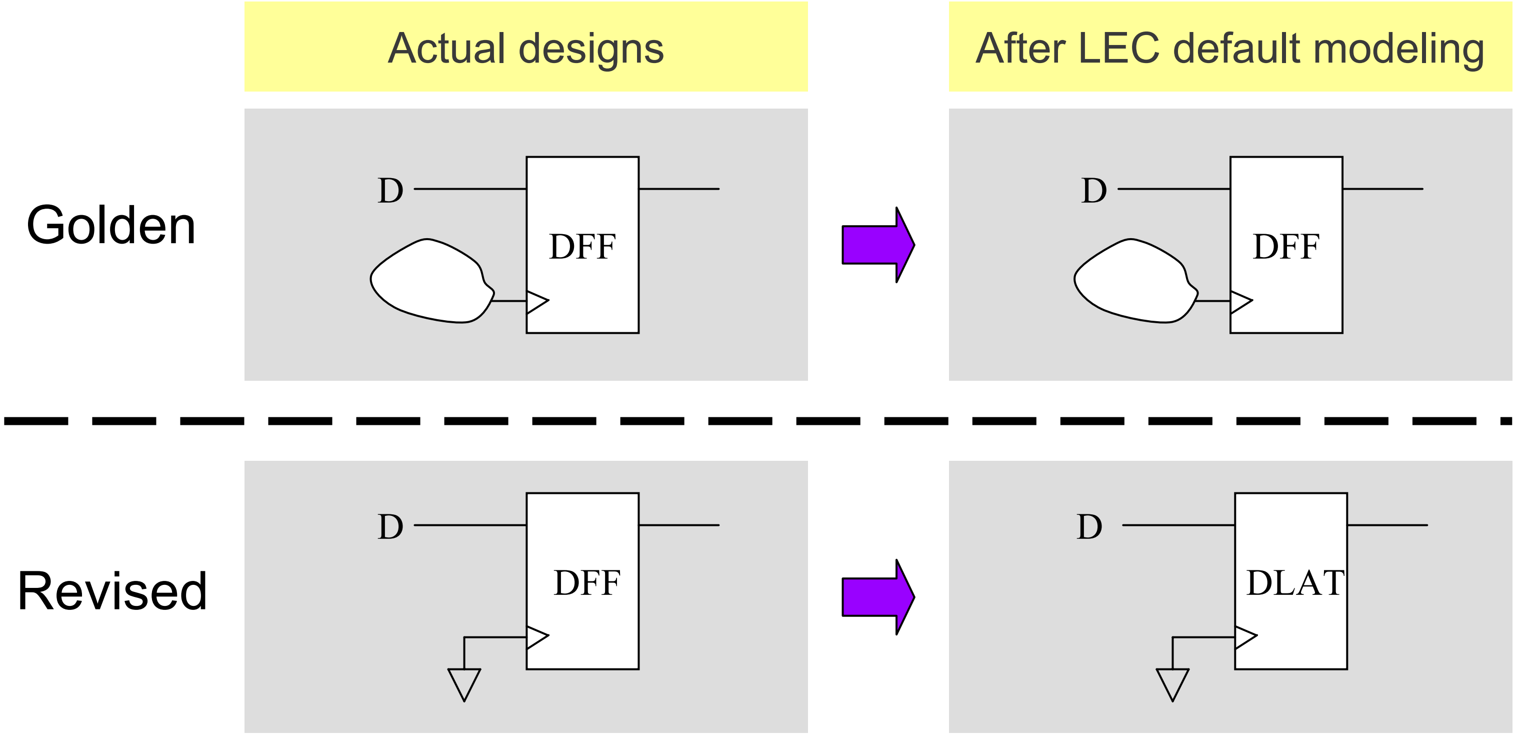
Golden

Revised

```
...
set flatten model -all_seq_merge
...
```

Grounded clock: Problem

- By default, LEC converts a DFF with clock signal disabled to a D-latch
- Might cause mapping/comparing problem!



Actual designs

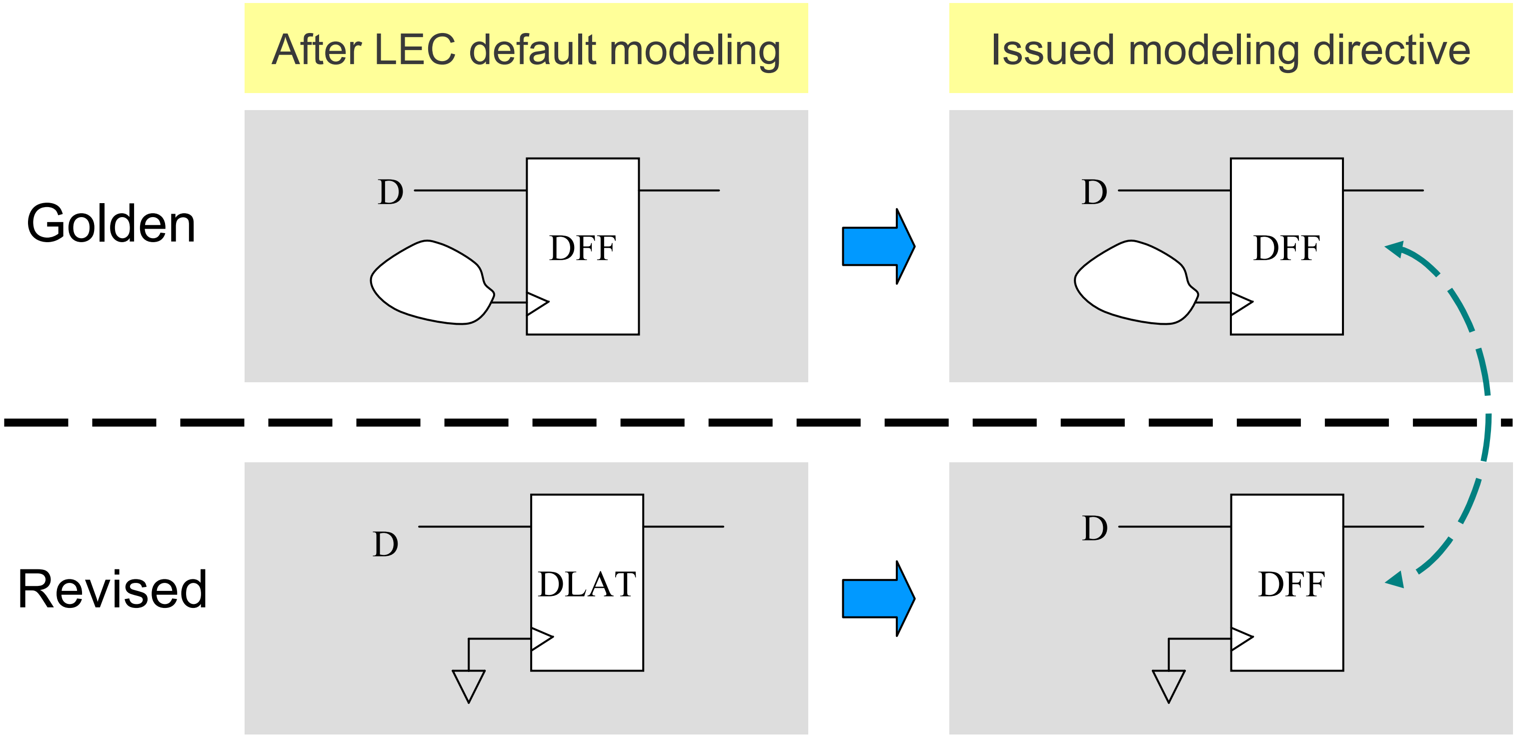
After LEC default modeling

Golden

Revised

Grounded clock: Solution

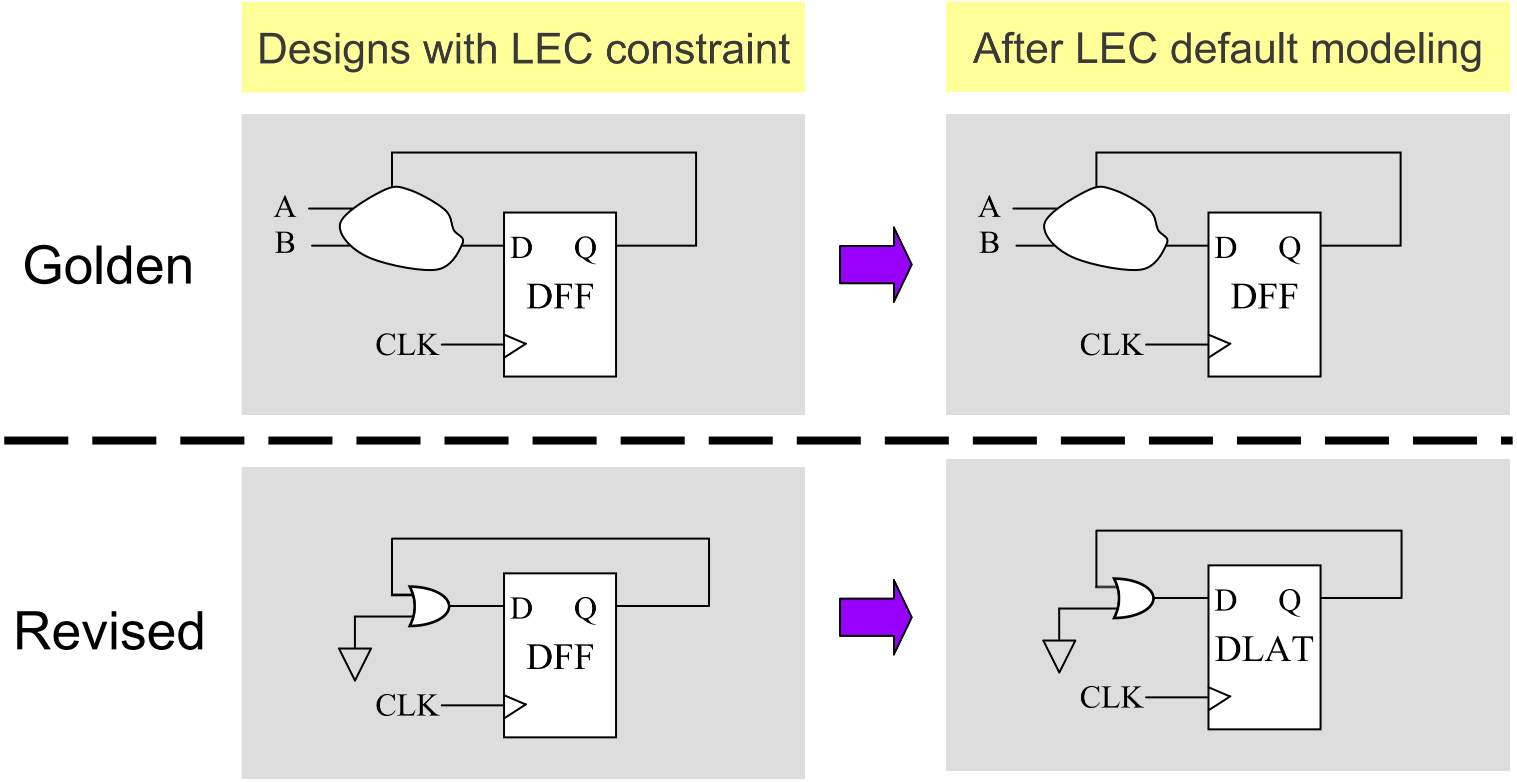
Instruct LEC not to turn DFF with disabled clock into a D-latch:



```
...
set flatten model -nodff_to_dlat_zero
...
```

DFF direct feedback: Problem

- By default, LEC converts a DFF to a D-latch when there is a direct feedback from the Q port to the D port
- Causes mapping/comparing problem!



Designs with LEC constraint

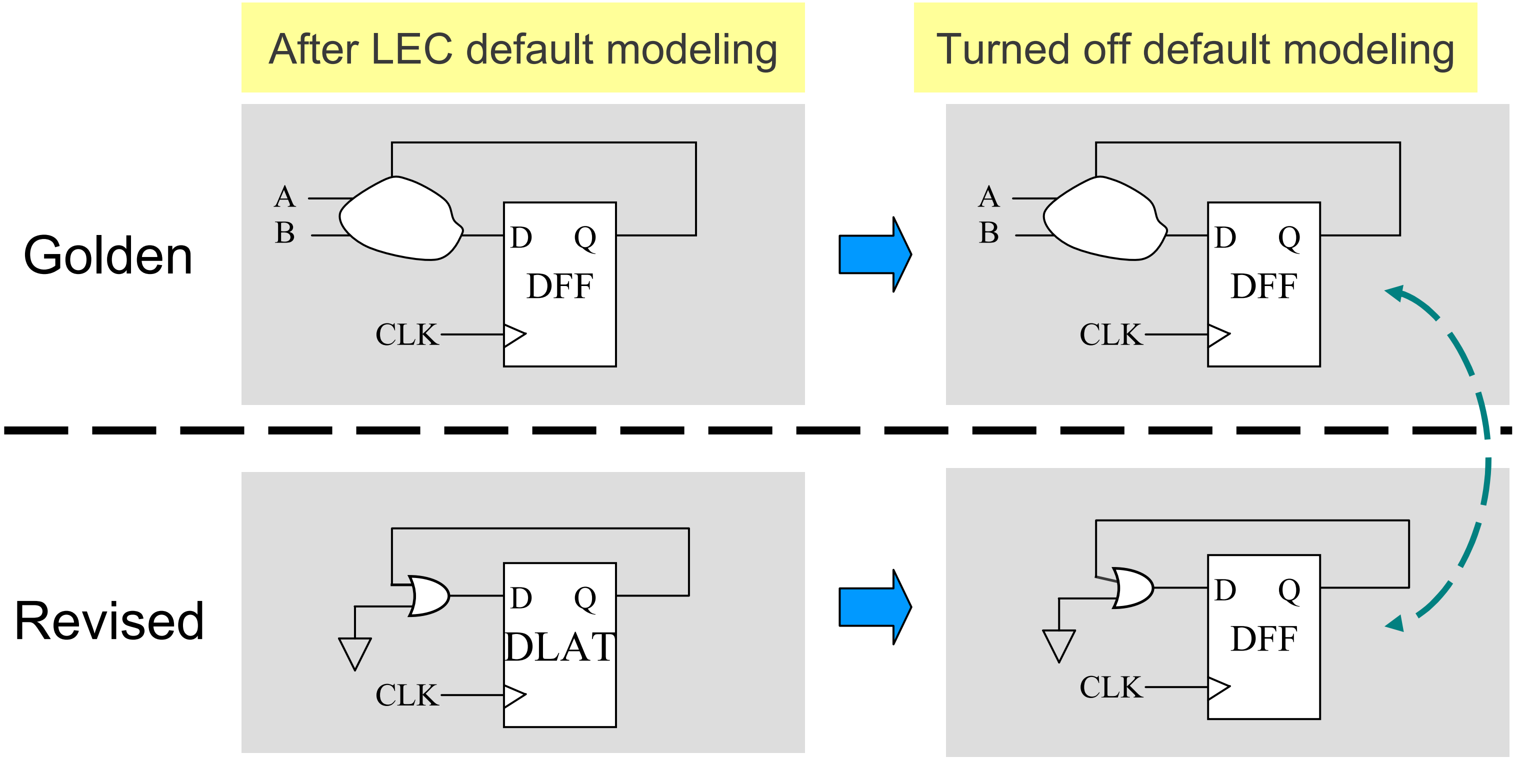
After LEC default modeling

Golden

Revised

DFF direct feedback: Solution

Instruct LEC not to model a DFF with direct feedback into a D-latch:



After LEC default modeling

Turned off default modeling

Golden

Revised

```
. . .
set flatten model –nodff_to_dlat_feedback
. . .
```

A typical session (flat compare)

SETUP

- Saving LEC transcript to a log file
- Specifying black boxes
- Reading libraries and designs
- Specifying design constraints
- Specifying modeling directives
- Switching to LEC mode

LEC

- Mapping process
  - Resolving unmapped key points
- Compare process
  - Debugging non-equivalent key points
- Report run statistics

Switching to LEC mode

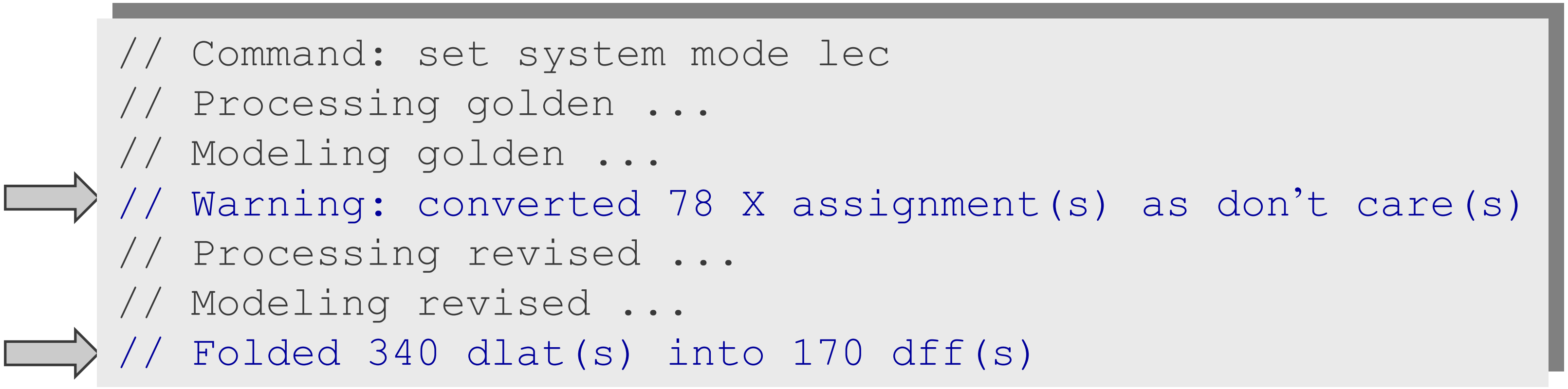
- Flatten golden and revised designs
- Perform circuit modeling
- Map key points (automatic by default)
  - Can turn off automatic mapping to read file containing mapped key points

```
set log file logfile.$LEC_VERSION -replace
add notranslate module *sram* -library -both
read design cpu_rtl.v -verilog -golden
read design -file verilog.vc -verilog -revised
add pin constraint 0 scan_en -revised
set flatten model -latch_fold
set system mode lec
...
```

Modeling messages

- LEC generates modeling messages after you change mode to LEC

Transcript window



```
// Command: set system mode lec
// Processing golden ...
// Modeling golden ...
// Warning: converted 78 X assignment(s) as don't care(s)
// Processing revised ...
// Modeling revised ...
// Folded 340 dlat(s) into 170 dff(s)
```

- To get a list of modeling messages and their rule number

  LEC> `report message -model`

- To display a particular message with detail

  LEC> `report message -model -rule <rule#> -verbose`

A typical session (flat compare)

SETUP

- Saving LEC transcript to a log file
- Specifying black boxes
- Reading libraries and designs
- Specifying design constraints
- Specifying modeling directives
- Switching to LEC mode

LEC

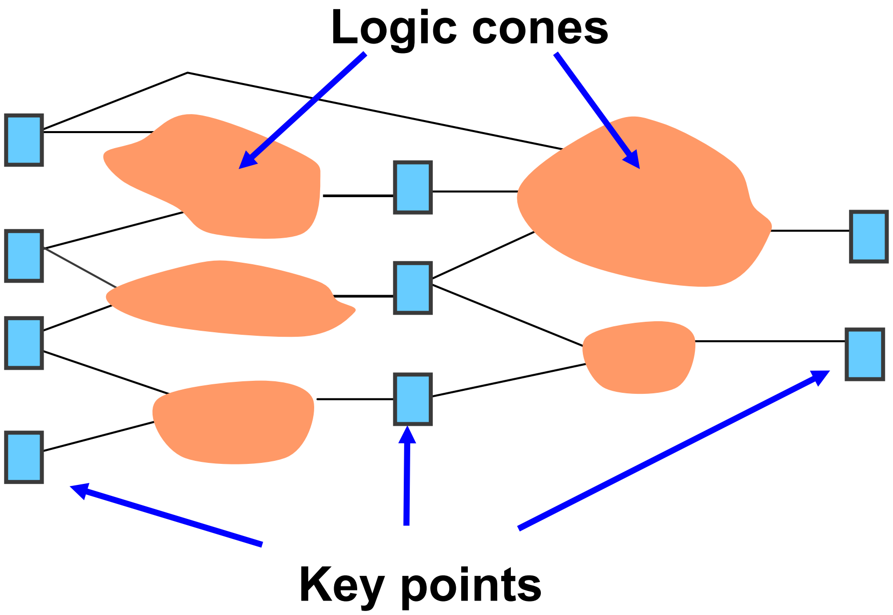
- Mapping process
  - Resolving unmapped key points
- Compare process
  - Debugging non-equivalent key points
- Report run statistics

Mapping process

- Understanding mapping process
  - What are key points?
  - What is key point mapping?
  - Why is key point mapping necessary?
  - How is key point mapping done?
- Resolve mapping issues
  - Mapping takes too long
  - Incomplete mapping

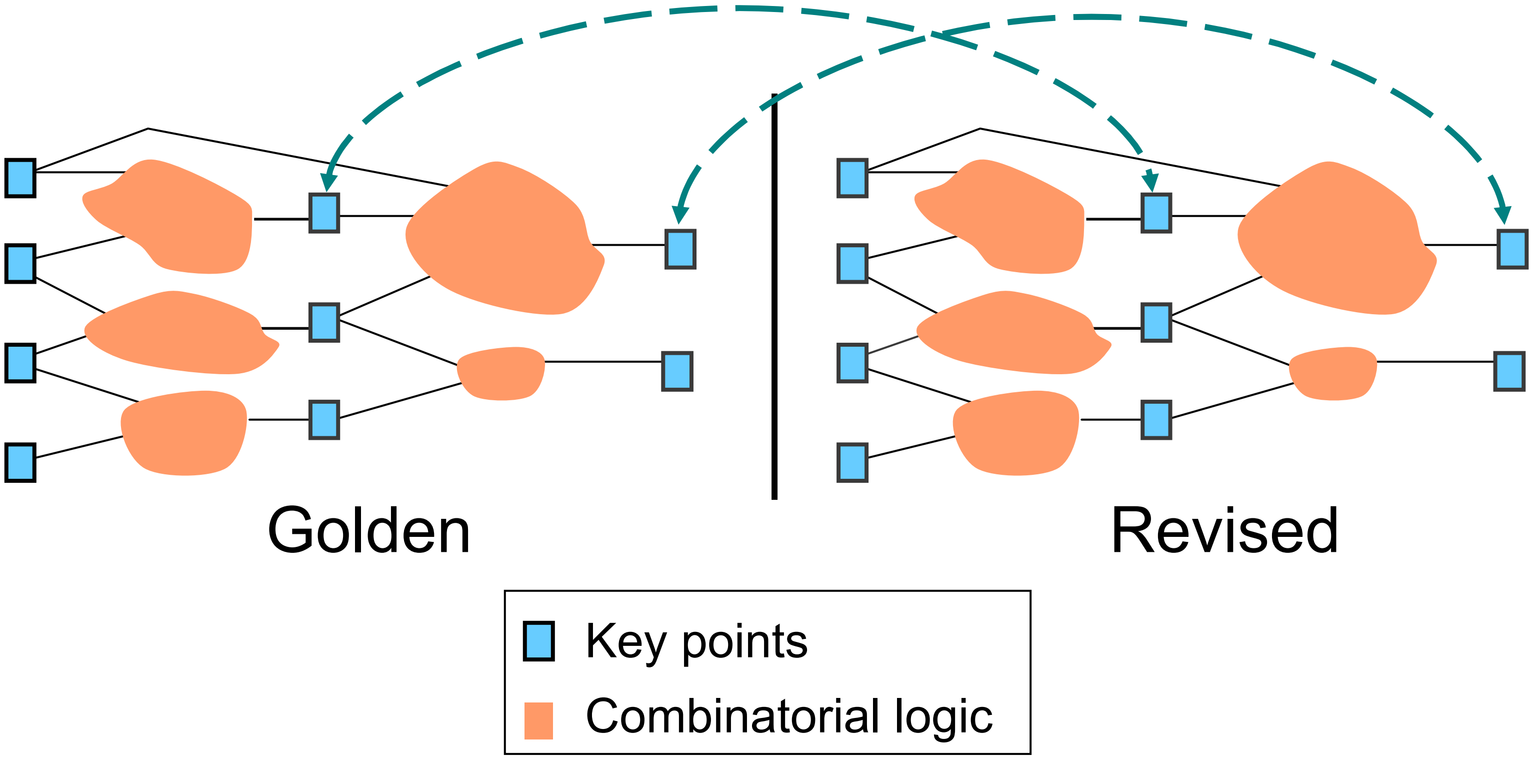
What are key points?

- Key Points: primary inputs(PI), primary outputs(PO), DFFs, D-latches, black boxes, Z gates, cut gates
- Design consists of combinational logic cones bounded by key points



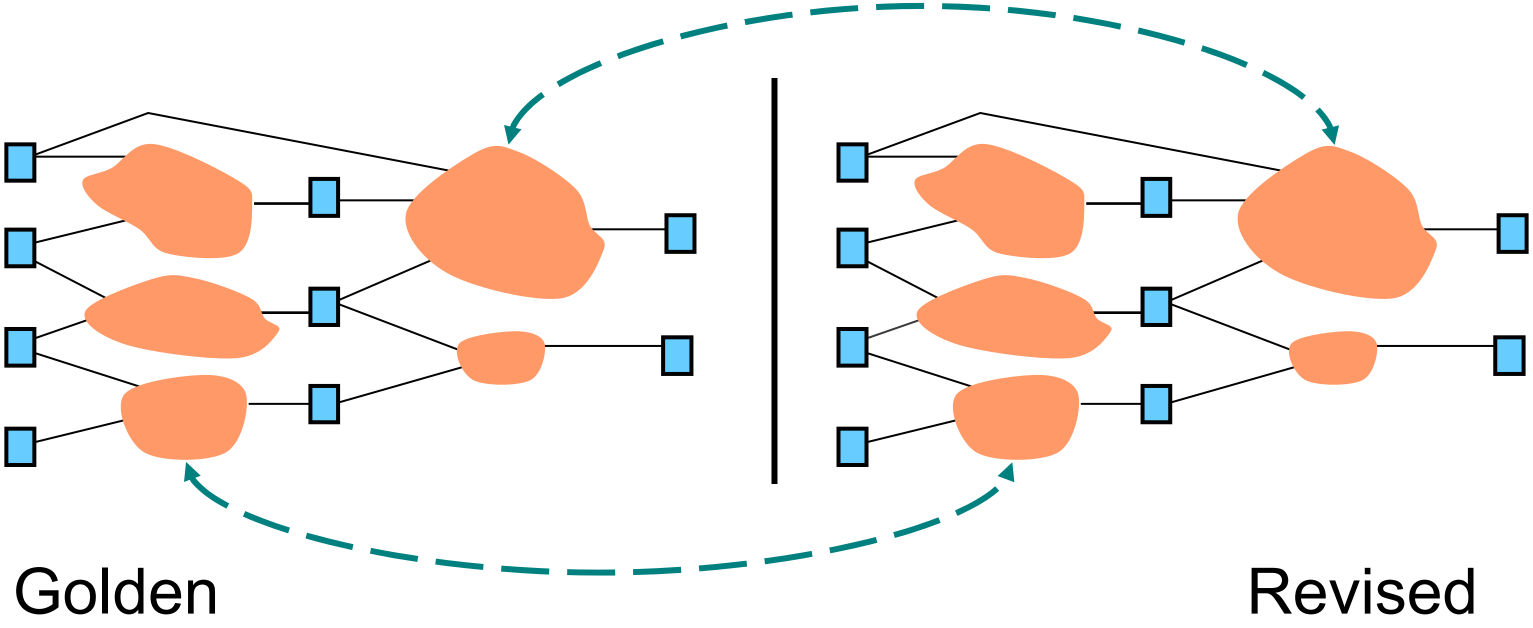
What is key point mapping?

Pairing corresponding key points: PI’s, PO’s, DFFs, D-latches, black boxes, Z gates, cut gates



Why is mapping necessary?

So that corresponding combinational logic cones are correctly paired

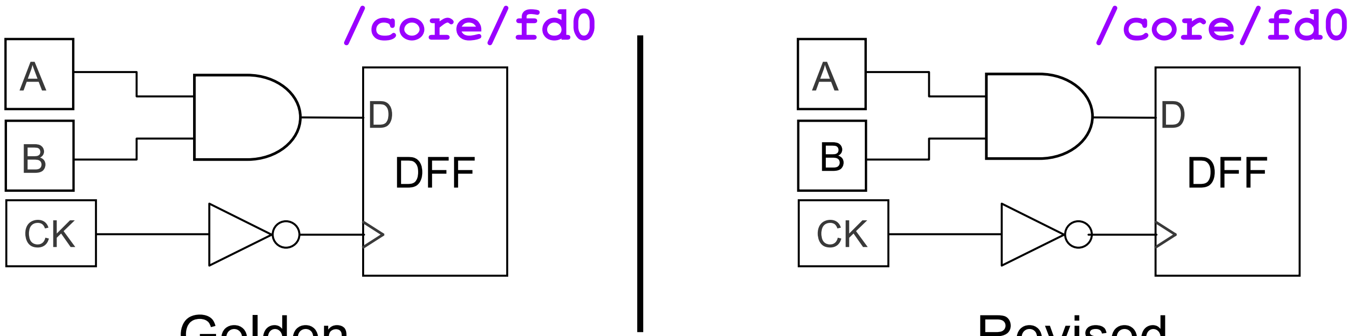


🟦 Key points

🟧 Combinatorial logic

How is mapping done?

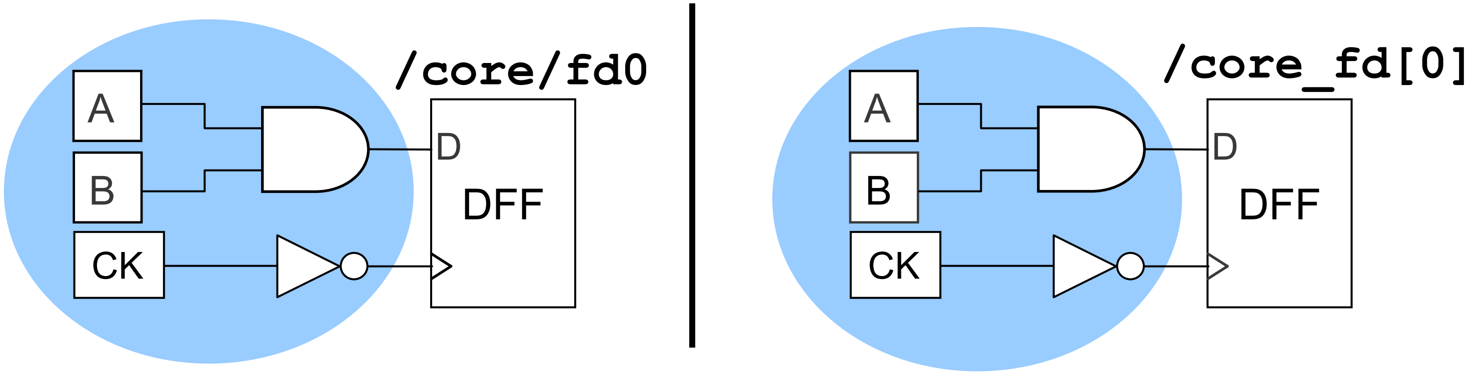
- **Name-based**: Mapping based strictly on instance pathnames (or net names) of key points



Golden

Revised

- **Function-based**: Mapping based on function and structure of logic cones



Golden

Revised

Mapping options

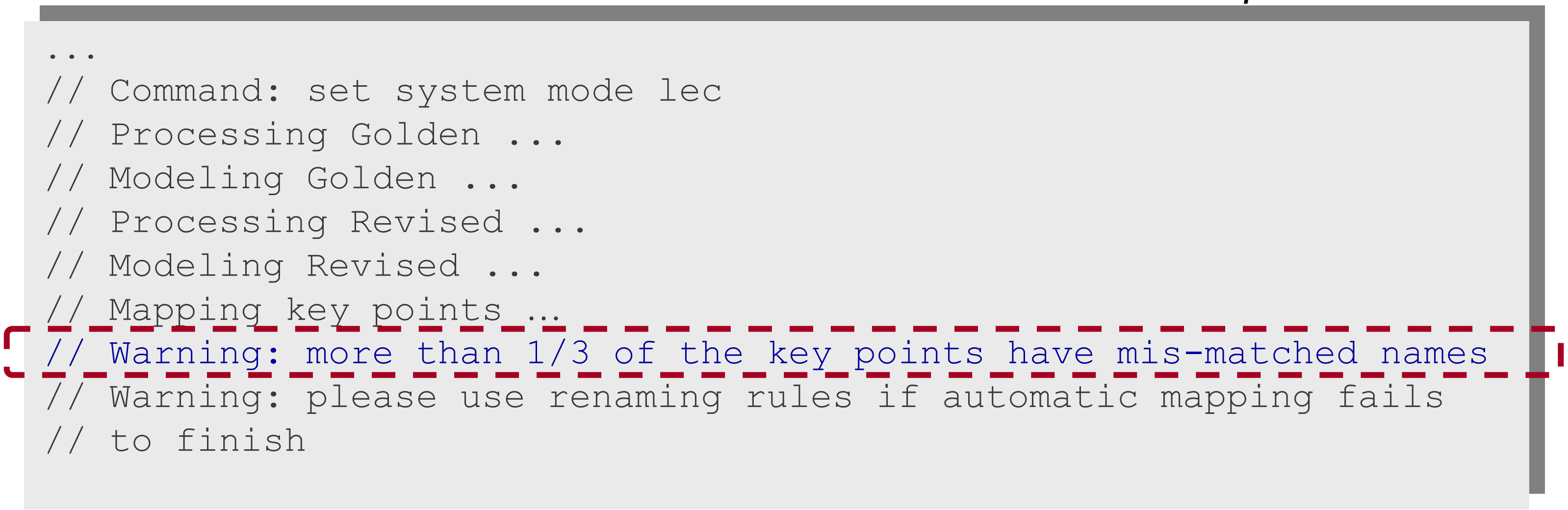
- LEC command to specify mapping options:
  - Syntax: `set mapping method <mapping_option>`
  - Can be applied in both SETUP and LEC modes
  - Available mapping options:

Mapping options	Execute mapping method	Followed by
-name first (*default*)	name-based	function-based
-name only	name-based	
-name guide	function-based	name-based
-noname	function-based	

Mapping messages

- SETUP> `set system mode lec`
  - The following message is displayed during mapping:

Transcript window

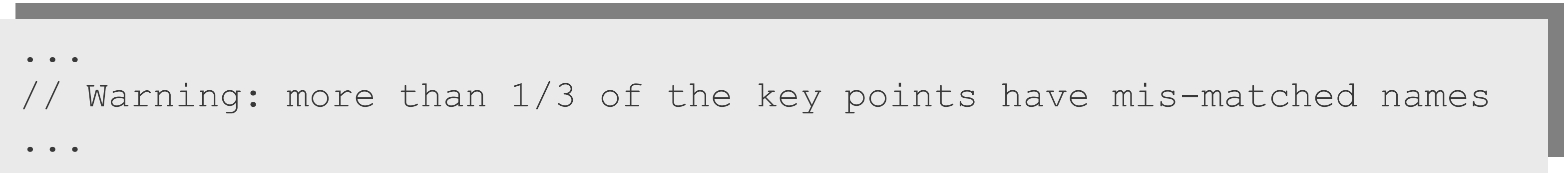


```
...
// Command: set system mode lec
// Processing Golden ...
// Modeling Golden ...
// Processing Revised ...
// Modeling Revised ...
// Mapping key points …
// Warning: more than 1/3 of the key points have mis-matched names
// Warning: please use renaming rules if automatic mapping fails
// to finish
```

Analyze mapping messages: Mapping takes too long

- Problem: Mapping takes too long
  - Mapping seems to hang and LEC displays this message

Transcript window



```
...
// Warning: more than 1/3 of the key points have mis-matched names
...
```

  - Default mapping
    - Name-based followed by function-based mapping
    - Function-based mapping is taking a long time

Analyze mapping messages: Mapping takes too long

- Solution:
  - Interrupt the mapping process: Ctrl-C on UNIX terminal
  - Remap using the "name only“ method:

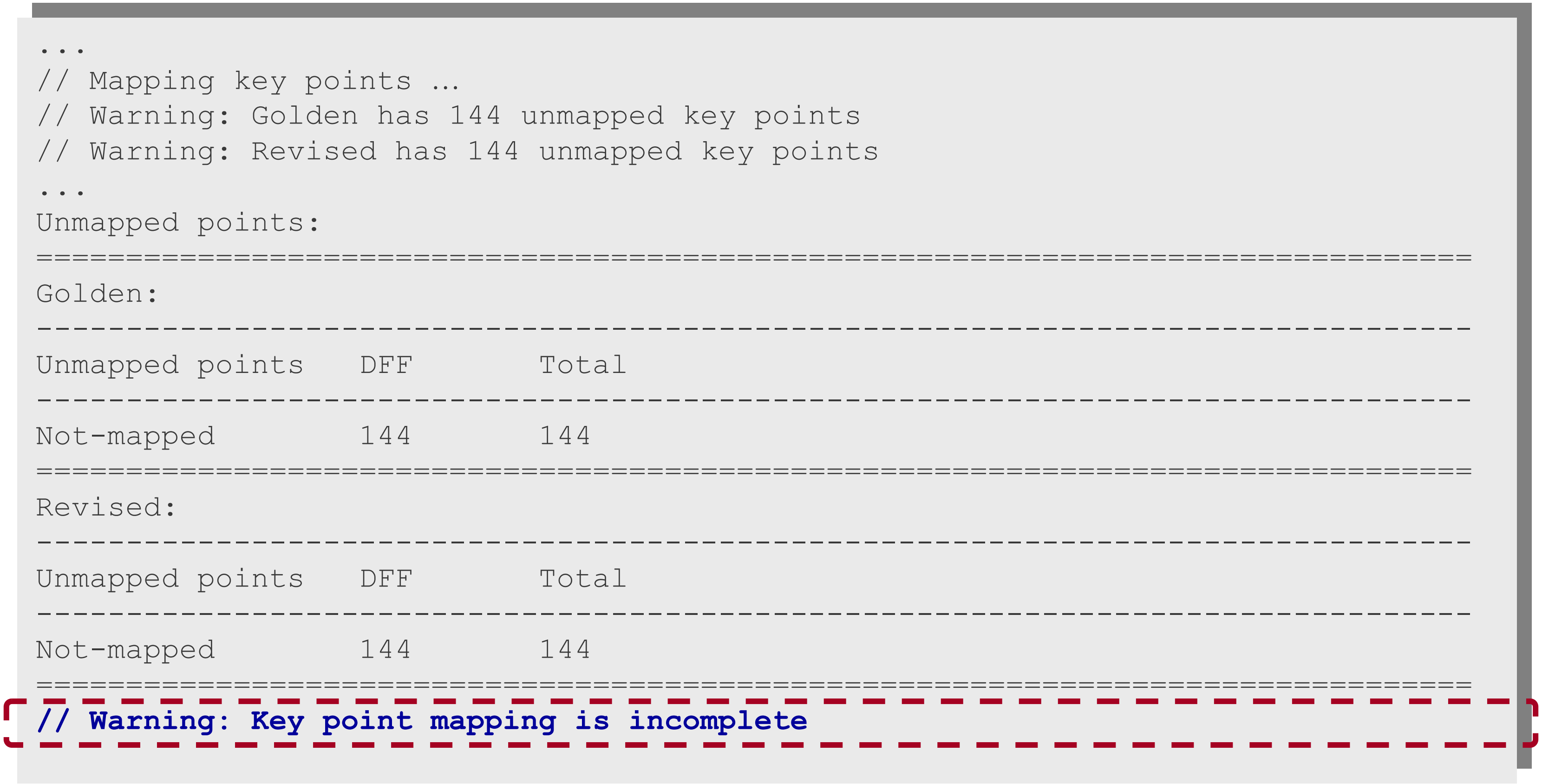
```
LEC> delete mapped points -all

LEC> set mapping method -name only
LEC> map key points
```

Mapping messages

The following message is displayed after mapping:

Transcript window



```
...
// Mapping key points ...
// Warning: Golden has 144 unmapped key points
// Warning: Revised has 144 unmapped key points
...
Unmapped points:
================================================================================
Golden:
--------------------------------------------------------------------------------
Unmapped points   DFF        Total
--------------------------------------------------------------------------------
Not-mapped        144        144
================================================================================
Revised:
--------------------------------------------------------------------------------
Unmapped points   DFF        Total
--------------------------------------------------------------------------------
Not-mapped        144        144
================================================================================
// Warning: Key point mapping is incomplete
```

Analyze mapping messages: Incomplete mapping

- Problem: Incomplete mapping
  - Mapping process ends with the message

Transcript window

```
...
// Warning: Key point mapping is incomplete
...
```

  - Unmapped points must be resolved
- Solution: Add renaming rule
  - Technique 1: Determine renaming rules from the Mapping Manager
  - Technique 2: Write out unmapped points into files then determine renaming rules from the files

Add renaming rule

- Command usage:

```
add renaming rule <rule_id>  <search_string> \
<replacement_string> [-golden | -revised]
```

- Renaming rule structures
  - %d - matches 1 or more digits (0-9)
  - #(expr) - expr evaluates to a constant integer
  - %s - matches 1 or more alpha-numeric characters
  - Refer to the Reference Manual for more renaming structures
- The escape character " \ "
  - You must prefix the following special characters with "\" if they are used in the search_string as a normal character

    % . * ^ $ | ( ) [ ] / \

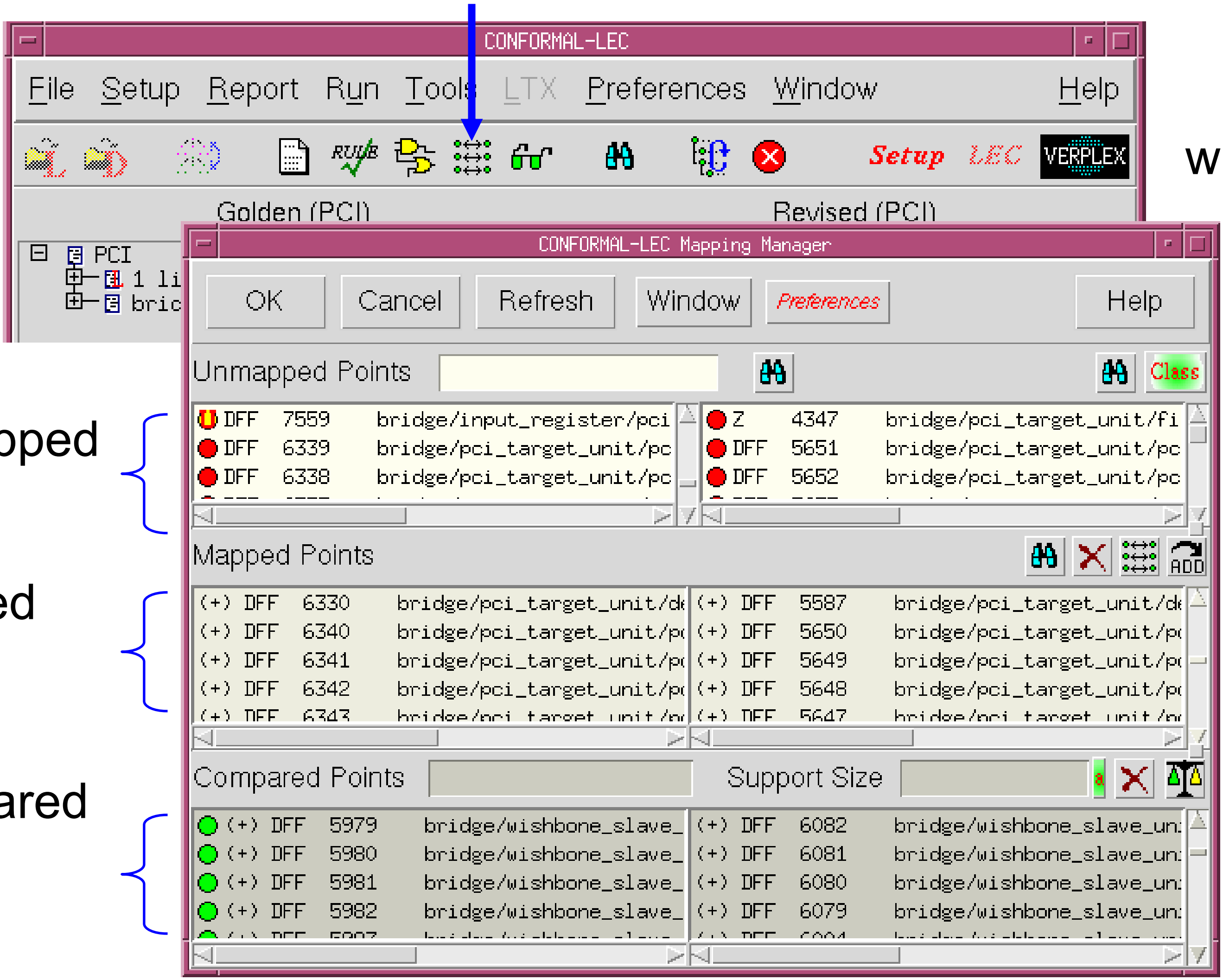
Add renaming rule: Example

Unmapped points in Golden	Unmapped points in Revised
/fifo_reg[0][0]	/fifo_0__reg[0]
/fifo_reg[0][1]	/fifo_0__reg[1]
/fifo_reg[0][2]	/fifo_0__reg[2]

Matching the Revised key points to the Golden:

```
add renaming rule R1 "%d__reg\[%d\]" "reg[@1][@2]" \
-revised
```

Mapping Manager



Unmapped Points

```
DFF  7559   bridge/input_register/pci     Z    4347   bridge/pci_target_unit/fi
DFF  6339   bridge/pci_target_unit/pc     DFF  5651   bridge/pci_target_unit/pc
DFF  6338   bridge/pci_target_unit/pc     DFF  5652   bridge/pci_target_unit/pc
```

Mapped Points

```
(+) DFF  6330   bridge/pci_target_unit/de   (+) DFF  5587   bridge/pci_target_unit/de
(+) DFF  6340   bridge/pci_target_unit/pc   (+) DFF  5650   bridge/pci_target_unit/pc
(+) DFF  6341   bridge/pci_target_unit/pc   (+) DFF  5649   bridge/pci_target_unit/pc
(+) DFF  6342   bridge/pci_target_unit/pc   (+) DFF  5648   bridge/pci_target_unit/pc
(+) DFF  6343   bridge/pci_target_unit/pc   (+) DFF  5647   bridge/pci_target_unit/pc
```

Compared Points

```
(+) DFF  5979   bridge/wishbone_slave_   (+) DFF  6082   bridge/wishbone_slave_un
(+) DFF  5980   bridge/wishbone_slave_   (+) DFF  6081   bridge/wishbone_slave_un
(+) DFF  5981   bridge/wishbone_slave_   (+) DFF  6080   bridge/wishbone_slave_un
(+) DFF  5982   bridge/wishbone_slave_   (+) DFF  6079   bridge/wishbone_slave_un
```

LEC window

Unmapped Points

Mapped Points

Compared Points

Mapping Manager

Categories of unmapped points

E **Extra**: Key point that exists in only the Golden design or in only the Revised design, but does not affect the circuit functionality. Example: *scan_in, scan_out*

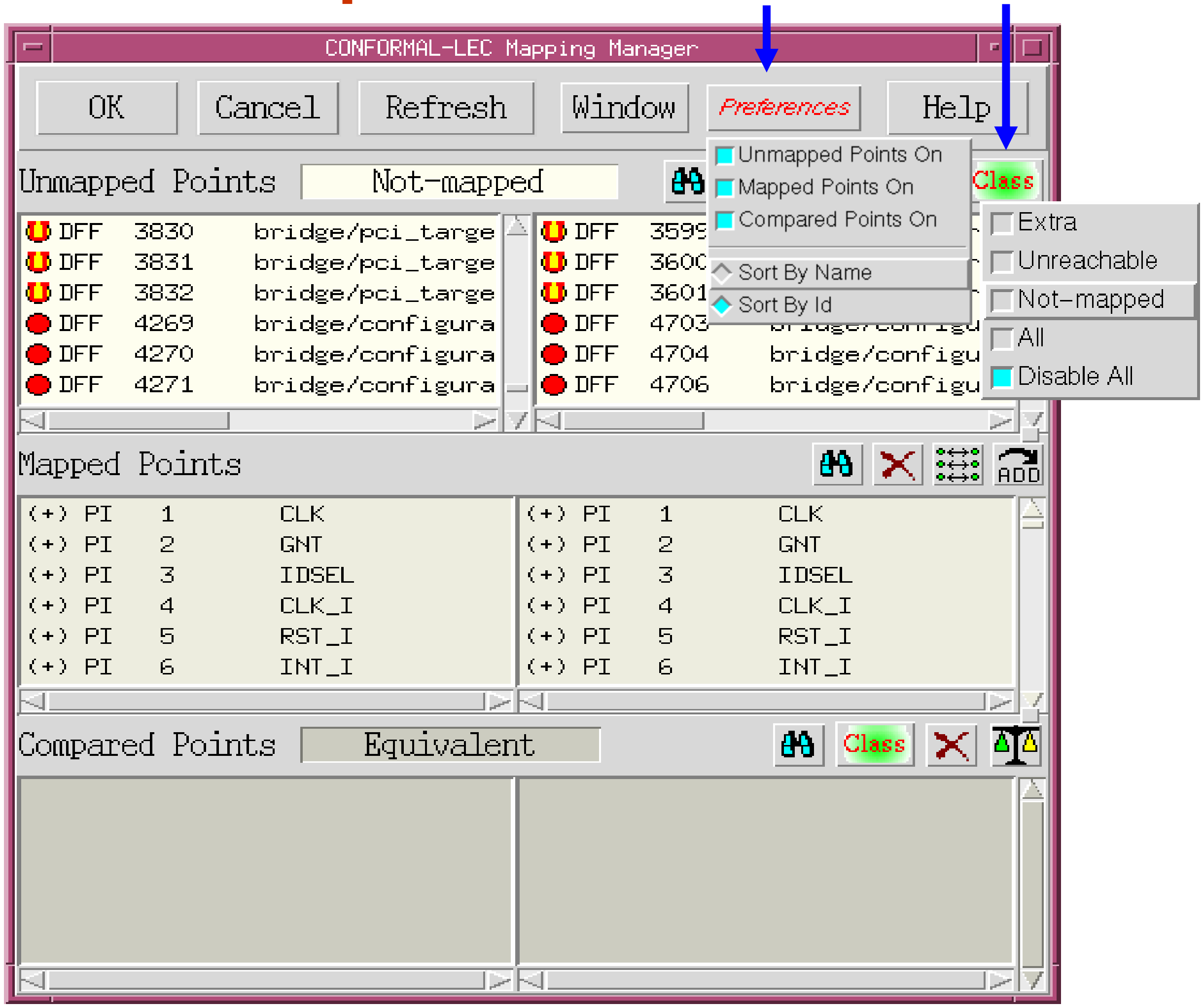
U **Unreachable**: Key point that is not propagated to any observable point. Example: *spare flops*

🔴 **Not-mapped**: Key point that has no correspondence on the other side. May be resolved with renaming rules (red-filled circle)

Note: To view a list of unmapped points, enter:

```
LEC> report unmapped points
```

Resolving incomplete mapping: Technique 1



Unmapped Points:

```
DFF   3830   bridge/pci_targe        DFF   3599
DFF   3831   bridge/pci_targe        DFF   3600
DFF   3832   bridge/pci_targe        DFF   3601
DFF   4269   bridge/configura        DFF   4703
DFF   4270   bridge/configura        DFF   4704   bridge/configu
DFF   4271   bridge/configura        DFF   4706   bridge/configu
```

Mapped Points:

```
(+) PI   1   CLK          (+) PI   1   CLK
(+) PI   2   GNT          (+) PI   2   GNT
(+) PI   3   IDSEL        (+) PI   3   IDSEL
(+) PI   4   CLK_I        (+) PI   4   CLK_I
(+) PI   5   RST_I        (+) PI   5   RST_I
(+) PI   6   INT_I        (+) PI   6   INT_I
```

- Using Mapping Manager to create renaming rules
- Tips

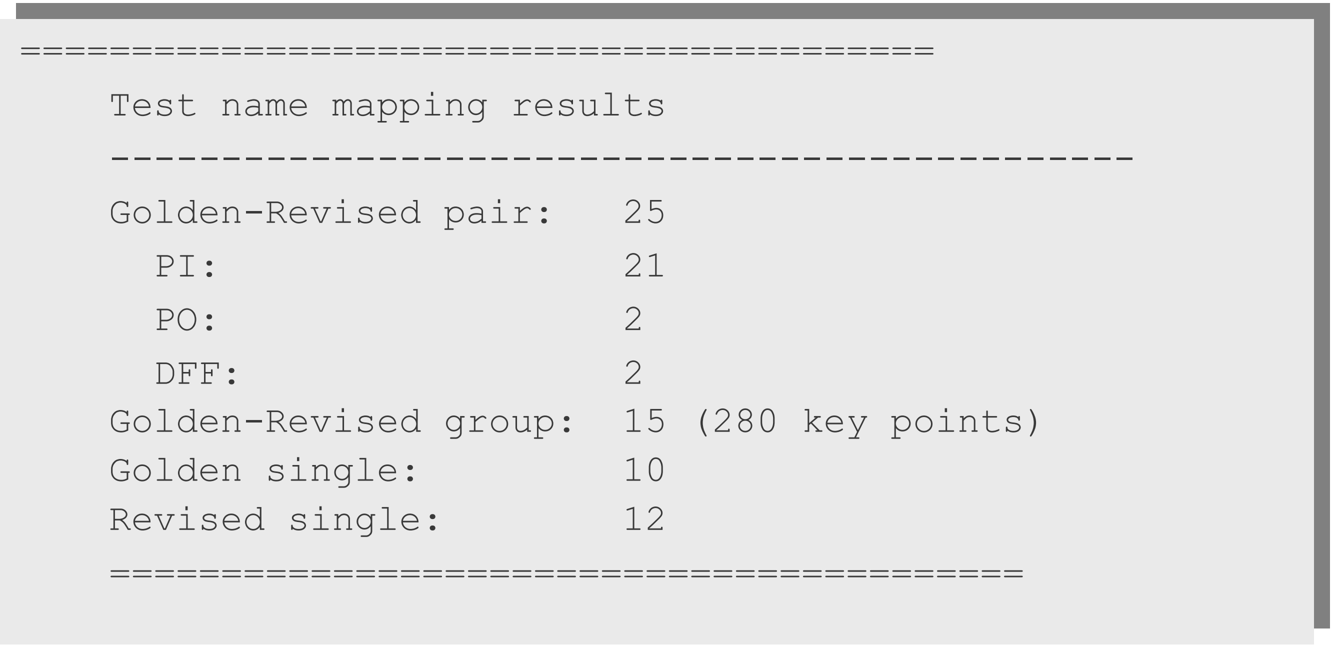
Preference → Sort by name

Class → Disable All → Not-Mapped

Resolving incomplete mapping: Technique 2

- Write out unmapped key points to files, then define renaming rules based on the output files
- Write out all unmapped key points as seen by LEC
  - LEC> `test renaming rule -map`

Transcript window



```
=========================================
    Test name mapping results
    ----------------------------------------------
    Golden-Revised pair:   25
      PI:                  21
      PO:                  2
      DFF:                 2
    Golden-Revised group:  15 (280 key points)
    Golden single:         10
    Revised single:        12
    =========================================
```

Resolving incomplete mapping: Technique 2 (cont'd)

```
LEC> test renaming rule -print group -file \
     group.list -replace
```

Content of group.list:

```
Key point groups: 15
   (G)    437   DFF   /syndrm/macreg/q_reg[4]
   (G)    439   DFF   /syndrm/macreg/q_reg[2]
   (R)    383   DFF   /syndrm/macreg/q_reg(4)/g2/i0
   (R)    385   DFF   /syndrm/macreg/q_reg(2)/g2/i0

   (G)    445   DFF   /syndrm/macreg_1/q_reg[4]
   (G)    447   DFF   /syndrm/macreg_1/q_reg[2]
   (R)    463   DFF   /syndrm/macreg_1/q_reg(4)/g2/i0
   (R)    465   DFF   /syndrm/macreg_1/q_reg(2)/g2/i0
...
```

Resolving incomplete mapping: Technique 2 (cont'd)

```
LEC> test renaming rule -print single -file single.list \
     -replace
```

Content of single.list:

```
Golden un-grouped single key point: 10
   (G)    2554   DFF   /rtn_data/xramBO_reg_2_/U$1
   (G)    2555   DFF   /rtn_data/xramBO_reg_1_/U$1
   (G)    2556   DFF   /rtn_data/xramBR_reg_31_/U$1
   (G)    2557   DFF   /rtn_data/xramBR_reg_30_/U$1
...
Revised un-grouped single key point: 12
   (R)    6993   DFF   /array_byte_reg_7_/Vpx/U$1/i0
   (R)    6994   DFF   /base_addr1_reg_9_/Vpx/U$1/i0
   (R)    6995   DFF   /array_byte_reg_4_/Vpx/U$1/i0
   (R)    6996   DFF   /request_reg_16_/Vpx8/U$1/i0
...
```

Based on "single.list" and "group.list", new renaming rules can be created to make the mapping more name-based.

Mapping is an iterative process

- Iterative process
  - Add rules incrementally
  - Continue the mapping process with command:

    ```
    LEC> map key point
    ```

    Note: LEC will skip the existing mapped points
  - Example:

    ```
    LEC> add renaming rule rule1 ...
    LEC> map key points
     ...
    LEC> add renaming rule rule3 ...
    LEC> add renaming rule rule4 ...
    LEC> map key points
    ...
    ```

Notes on renaming rules

- Rules are order dependent
- To view rules:
  - `report renaming rule`
- To delete a rule:
  - `delete renaming rule <rule_id>`
- "`test renaming rule`" can test:
  - A rule before adding
  - All rules added
- Built-in renaming rules
- Renaming Rule GUI window can be used to edit rule(s)

Test renaming rule

- Test a renaming rule before adding it

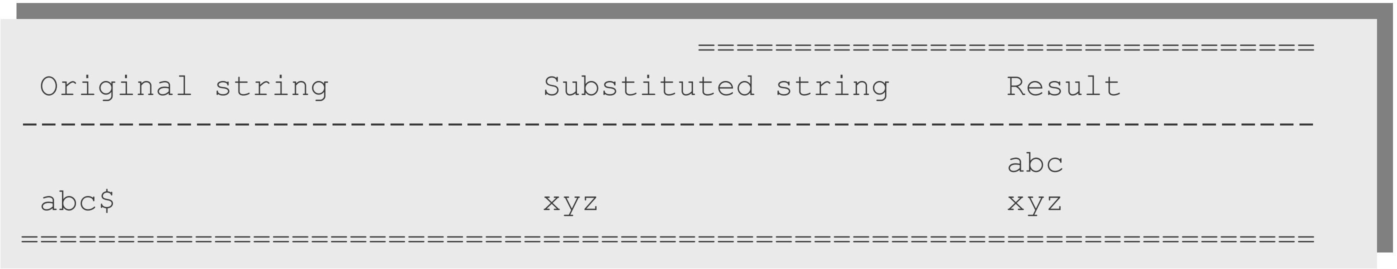
  Syntax:

```
test renaming rule <test_string | -gate_id <gate_id>> \
-new_rule <search_string> <replace_string>
```

  Example:

```
LEC> test re r abc -new "abc$" "xyz"
```

Transcript window



```
                                ====================================
 Original string                Substituted string       Result
------------------------------------------------------------------------------
                                                         abc
 abc$                           xyz                      xyz
==============================================================================
```

  - Add the rule when the intended result is achieved

```
LEC> add re r rule0 "abc$" "xyz" -revised
```

Test renaming rule (cont'd)

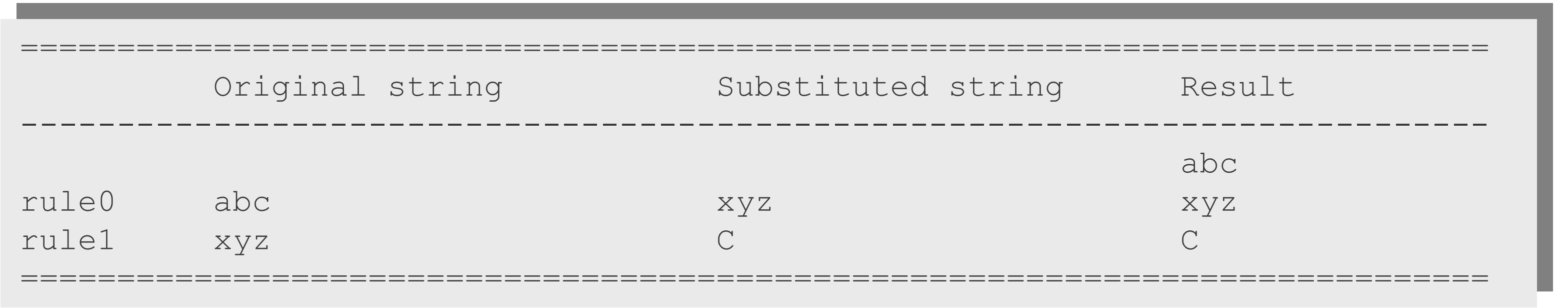
- Test all added renaming rules
  - Renaming rules are order dependent
  - Syntax:

```
test renaming rule <test_string | -gate_id <gate_id>>
```

  - Example:

```
LEC> add re r rule1 "xyz$" "C" -map -revised
LEC> test re r abc  -revised
```

Transcript window



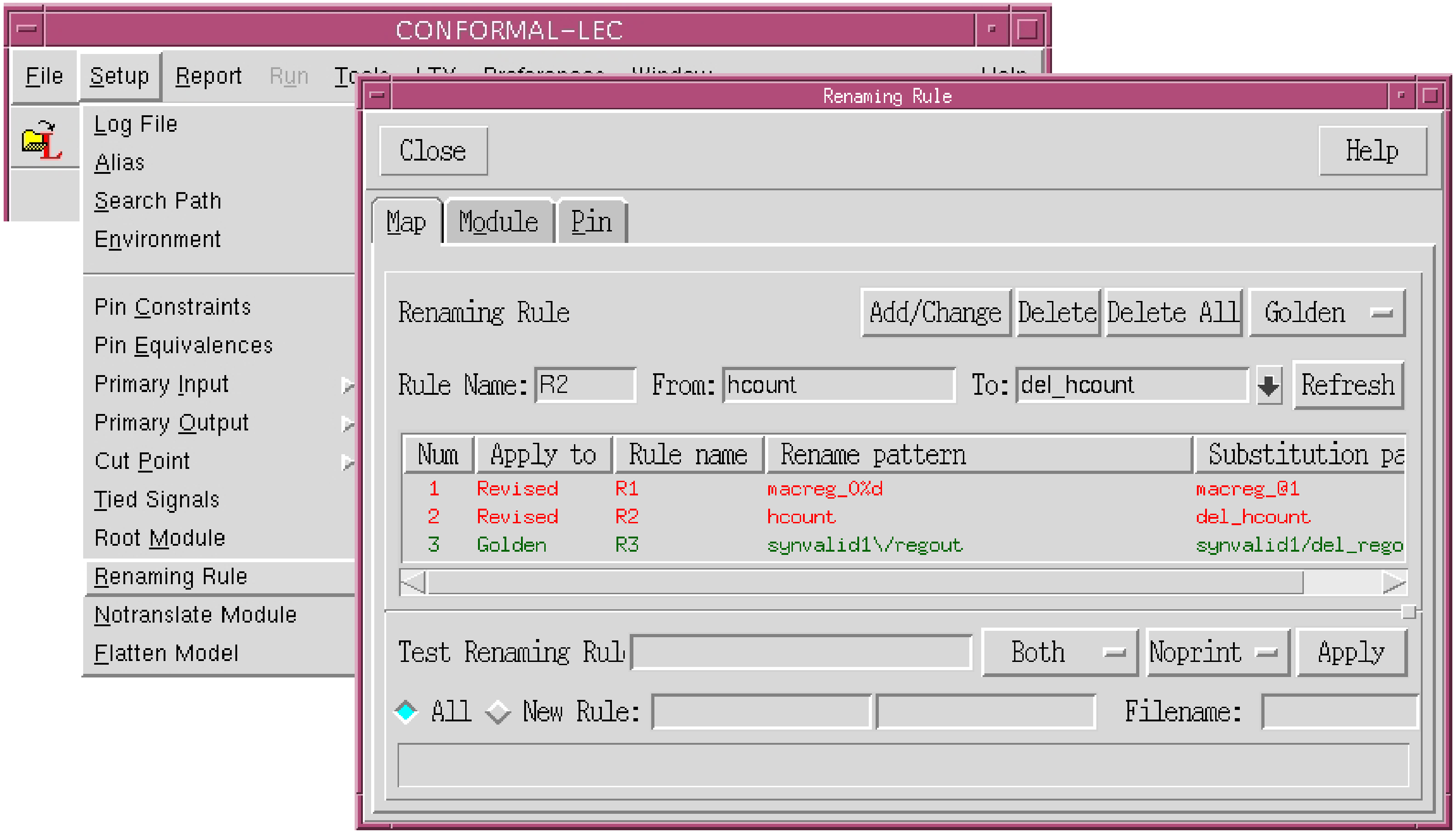
```
=====================================================================
         Original string         Substituted string     Result
---------------------------------------------------------------------
                                                        abc
rule0    abc                     xyz                    xyz
rule1    xyz                     C                      C
=====================================================================
```

Built-in renaming rules

- Built-in renaming rules (version 3.4.0.a or later) resolve the following renaming rule issues:
  - Array delimiters: [ ], _ _, < >, ( )
  - Hierarchical separator: _, /
  - Extraneous digits: reg%d_%d
    - Multiple always blocks

Renaming Rule GUI window

You can add, test, and edit rules in the Renaming Rule GUI window



Rule Name: R2 From: hcount To: del_hcount

Num	Apply to	Rule name	Rename pattern	Substitution pa
1	Revised	R1	macreg_0%d	macreg_@1
2	Revised	R2	hcount	del_hcount
3	Golden	R3	synvalid1\/regout	synvalid1/del_rego

◆ All ◇ New Rule:

Exercise

Define renaming rules to match the following Golden/Revised key points pairs

<table>
<tr><td>1</td><td>Golden
/u0/u1/c_reg[5]</td><td>Revised
/u0/c_reg(5)/I1/U$1</td></tr>
<tr><td></td><td colspan="2"><i>answer:</i> <code>add renaming rule r1 "u1\/" "" -golden</code>
Note: Trailing characters for library cells are tripped automatically (<code>/I1/U$1</code>)</td></tr>
<tr><td>2</td><td>Golden
/u0/u1/c_reg[1]</td><td>Revised
/u0_u1_c_.ram1_reg</td></tr>
<tr><td></td><td colspan="2"><i>answer:</i> <code>add renaming rule</code> ____</td></tr>
<tr><td>3</td><td>Golden
/u0/b_reg[24]</td><td>Revised
/u0/b_2_reg[22]</td></tr>
<tr><td></td><td colspan="2"><i>answer:</i> <code>add renaming rule</code></td></tr>
</table>

Solutions are in the back of this manual

Add renaming rule: Module pins

- Problem: Black box pins not mapped due to name differences

- Solution: Apply renaming rule to the pin
  - **Syntax**: `add renaming rule name_name \`
    `<search_string> <replace_string>` *`-pin -bbox`* `\`
    `<module_name>` `[-golden|-revised]`

Summary

Once you are satisfied with the mapping results, you can incorporate all renaming rules into your dofile

```
set log file logfile.$LEC_VERSION -replace
add notranslate module *sram* -library -both
read design cpu_rtl.v -verilog -golden
read design -file verilog.vc -verilog -revised
add pin constraint 0 scan_en -revised
set flatten model -latch_fold
add renaming rule rule0 "abc" "xyz" -map -revised
add renaming rule rule1 "xyz" "C" -map -golden
set system mode lec
...
```

A typical session (flat compare)

SETUP

- Saving LEC transcript to a log file
- Specifying black boxes
- Reading libraries and designs
- Specifying design constraints
- Specifying modeling directives
- Switching to LEC mode

LEC

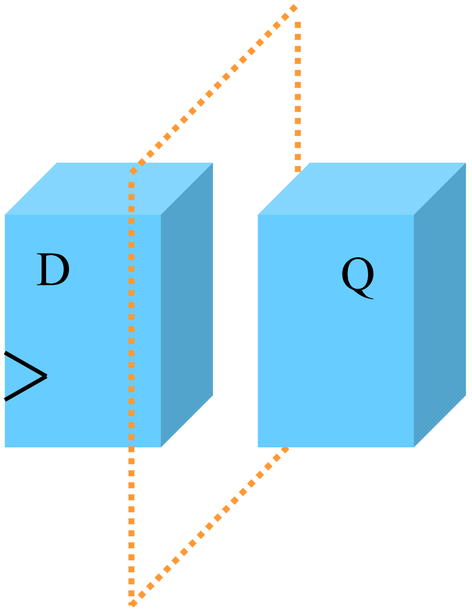
- Mapping process
  - Resolving unmapped key points
- Compare process
  - Debugging non-equivalent key points
- Report run statistics

Compare process

- Understanding compare process
  - How is state element handled?
  - What are the compare points?
  - What is being compared?
- Debugging non-equivalent key points

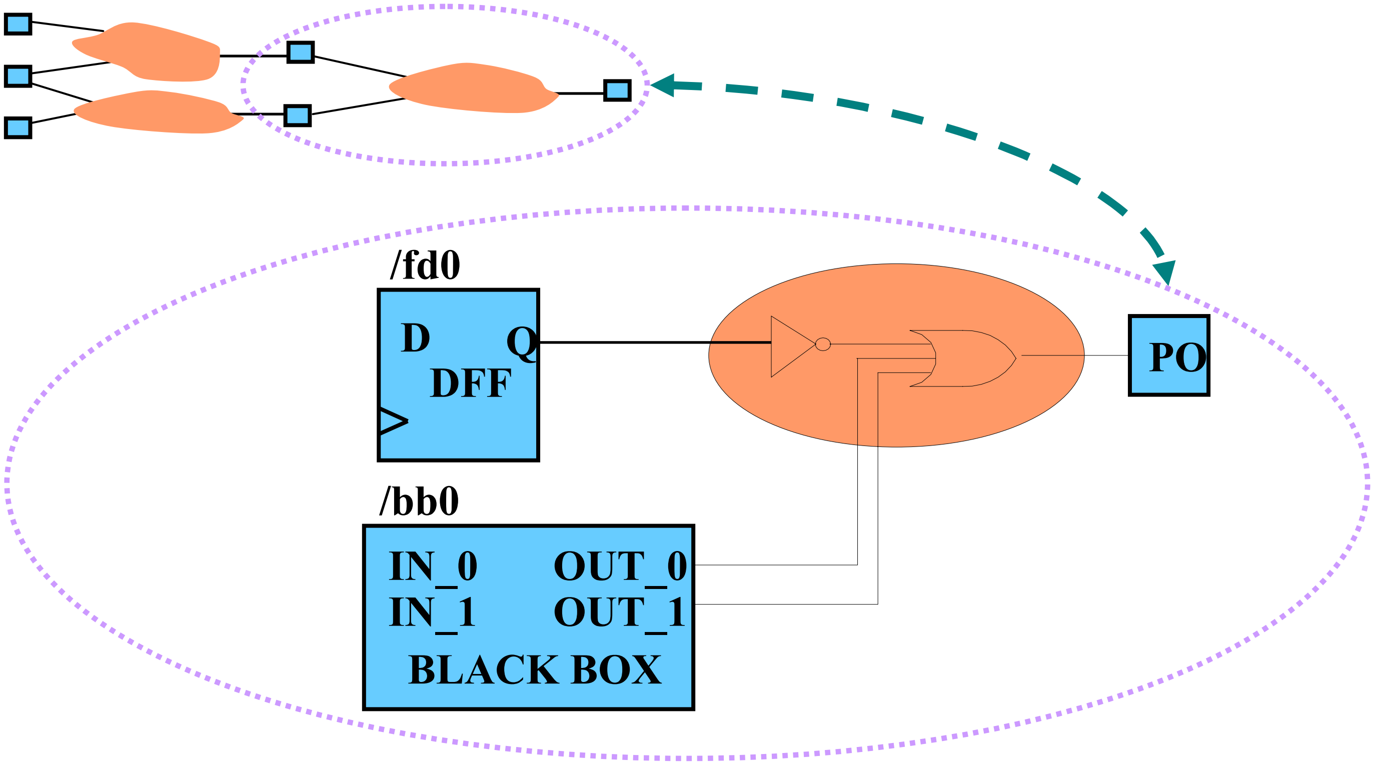
How is state element handled?

- No connection between input and output
  - Think of a D-latch or a DFF being cut in halves
  - Input and output are verified separately



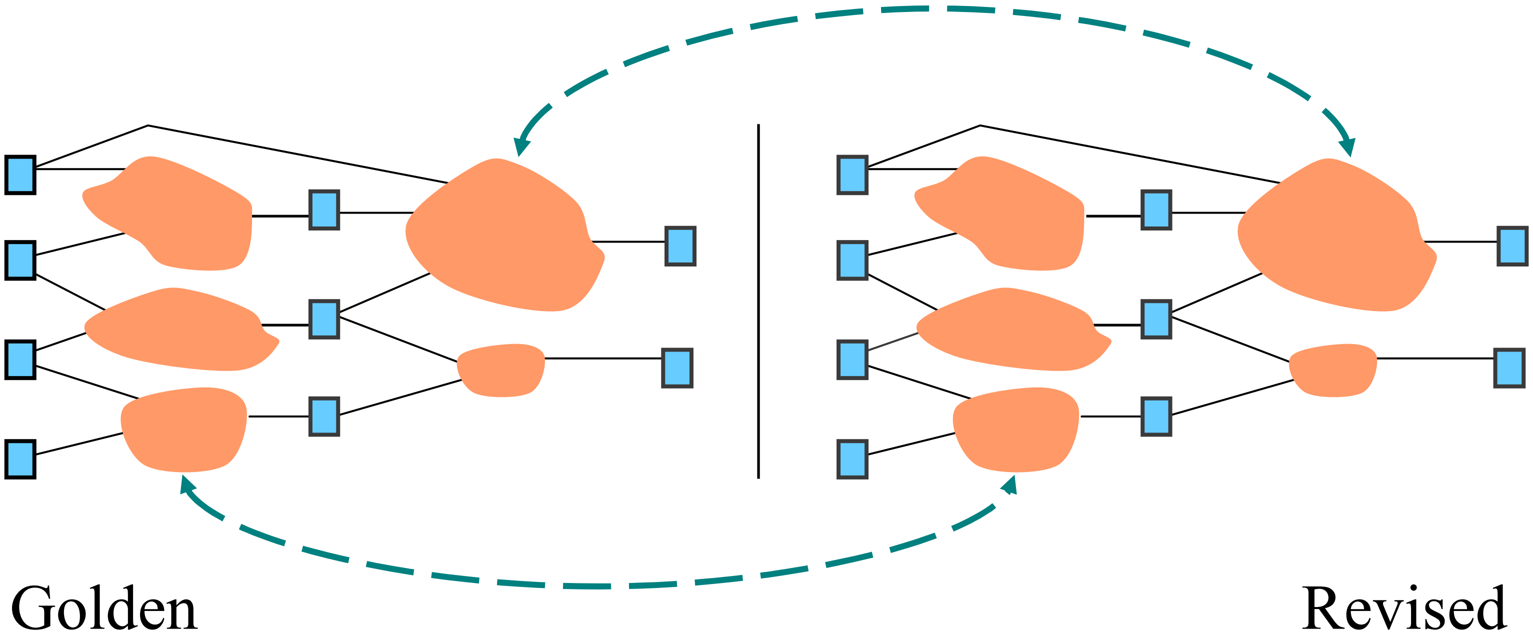
What are compare points?

- Compare points are:
  - Sink points of logic cones
  - Primary outputs, cut gates, DFFs, D-latches, and black boxes



What is being compared?

- Corresponding *combinational* logic cones



- Two designs are equivalent when *all* corresponding cones are equivalent

Comparing

- Consider all successfully mapped points for comparison
- Comparison is an iterative process
  - LEC remembers points already compared equiv/non-equiv
  - Can interrupt with "Ctrl-C" to change compare options

    To continue the compare process, enter "`compare`"

```
set log file logfile.$LEC_VERSION -replace
add notranslate module *sram* -library -both
read design cpu_rtl.v -verilog -golden
read design -file verilog.vc -verilog -revised
add pin constraint 0 scan_en -revised
set flatten model -latch_fold
add renaming rule rule0 "abc" "xyz" -map -revised
add renaming rule rule1 "xyz" "C" -map -golden
set system mode lec
add compare points -all
compare
...
```

Comparison results

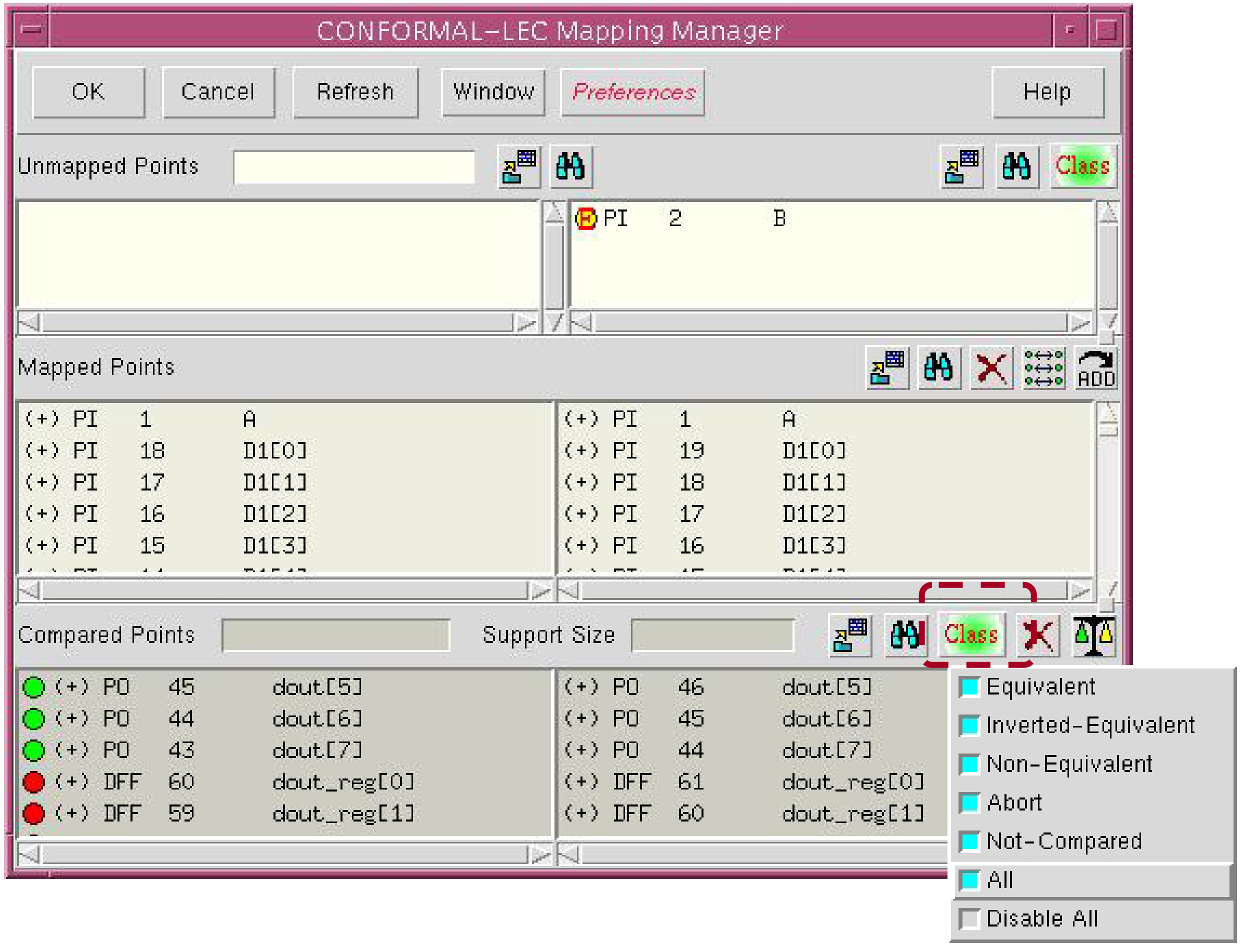
- The following message is displayed after comparing:

Transcript window

```
...
// Command: compare
================================================================================
Compared points     PO     DFF    DLAT   BBOX      Total
--------------------------------------------------------------------------------
Equivalent          2      146     2       1        151
--------------------------------------------------------------------------------
Non-equivalent      0        2     0       0          2
================================================================================
```

Comparison results in GUI

Filtering comparison results on the Mapping Manager



Unmapped Points:

```
PI    2        B
```

Mapped Points:

```
(+) PI   1      A          (+) PI   1      A
(+) PI   18     D1[0]      (+) PI   19     D1[0]
(+) PI   17     D1[1]      (+) PI   18     D1[1]
(+) PI   16     D1[2]      (+) PI   17     D1[2]
(+) PI   15     D1[3]      (+) PI   16     D1[3]
```

Compared Points:

```
● (+) PO   45    dout[5]         (+) PO   46    dout[5]
● (+) PO   44    dout[6]         (+) PO   45    dout[6]
● (+) PO   43    dout[7]         (+) PO   44    dout[7]
● (+) DFF  60    dout_reg[0]     (+) DFF  61    dout_reg[0]
● (+) DFF  59    dout_reg[1]     (+) DFF  60    dout_reg[1]
```

Categories of comparison results

🟢 **Equivalent:** Key points proven to be equivalent (green-filled circle)

⊖ **Inverted-Equivalent:** Key points proven to be complementary (divided green-filled circle)

🔴 **Non-Equivalent:** Key points proven to be different (red-filled circle)

🟡 **Abort:** Key points not yet proven equivalent or non-equivalent due to timeout or other system parameters (yellow-filled circle)

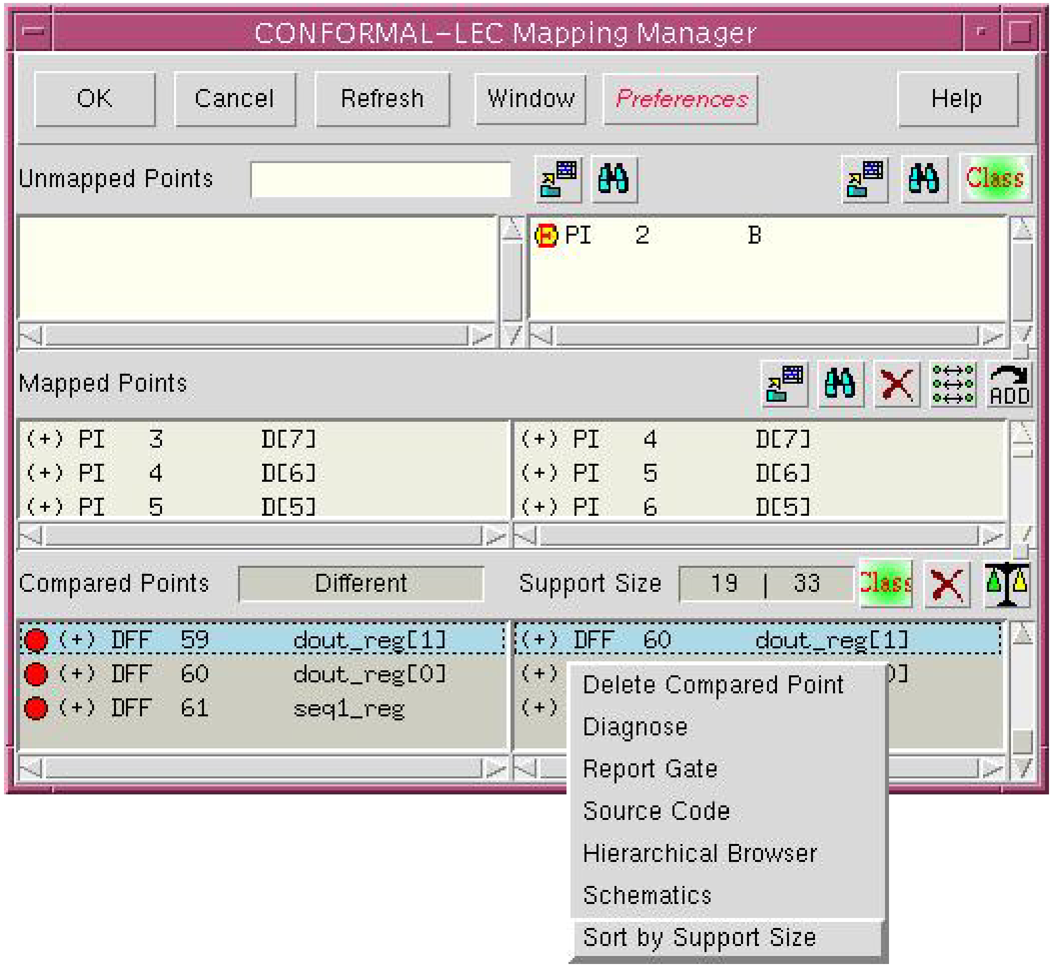
? **Not-Compared:** Key points not yet compared

```
LEC> report compare data -class [...]
```

Debugging non-equivalent key points

- Diagnose smaller cone first
  - Using the Mapping Manager to sort key points by supporting size
- Concentrate on one logic cone at a time
- Debugging tools
  - Diagnosis Manager
  - Schematic Viewer
  - Source Code Manager
  - Gate Manager

Sort key point by supporting size



```
(+) PI   2      B
```

```
(+) PI   3      D[7]        (+) PI   4      D[7]
(+) PI   4      D[6]        (+) PI   5      D[6]
(+) PI   5      D[5]        (+) PI   6      D[5]
```

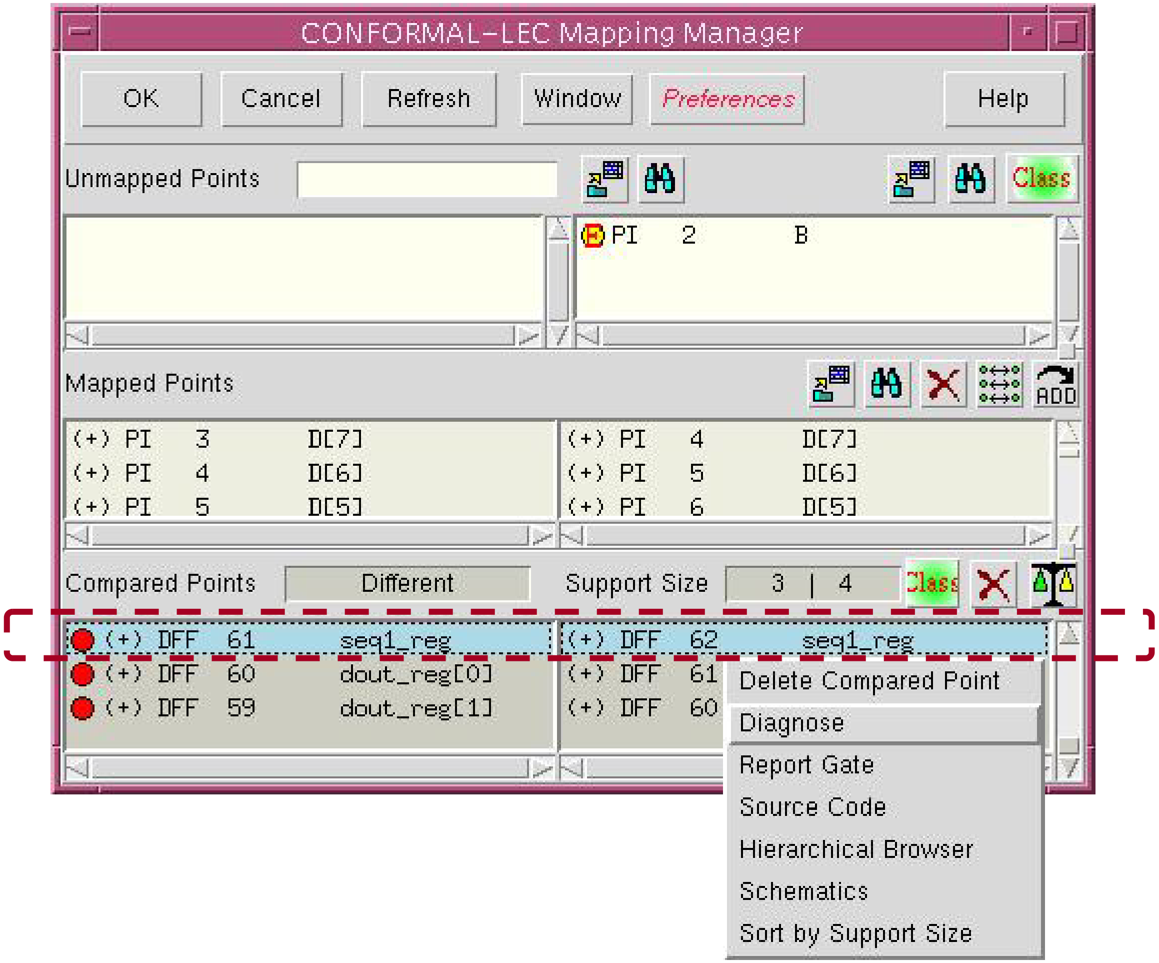
Support Size 19 | 33

```
● (+) DFF  59   dout_reg[1]     (+) DFF  60   dout_reg[1]
● (+) DFF  60   dout_reg[0]     (+)                 0]
● (+) DFF  61   seq1_reg        (+)
```

Left-click to select a compared point.

Then right-click to select "Sort by Support Size“ from pull-down menu

Invoking the Diagnosis Manager



```
PI   2        B
```

```
(+) PI   3         D[7]        (+) PI   4         D[7]
(+) PI   4         D[6]        (+) PI   5         D[6]
(+) PI   5         D[5]        (+) PI   6         D[5]
```

```
● (+) DFF  61      seq1_reg        (+) DFF  62      seq1_reg
● (+) DFF  60      dout_reg[0]     (+) DFF  61
● (+) DFF  59      dout_reg[1]     (+) DFF  60
```

Left-click to select a non-equivalent point (red-filled circle).

Then right-click, choose "Diagnose"

Diagnosis Manager

Compared Point

Diagnosis Point (Active)

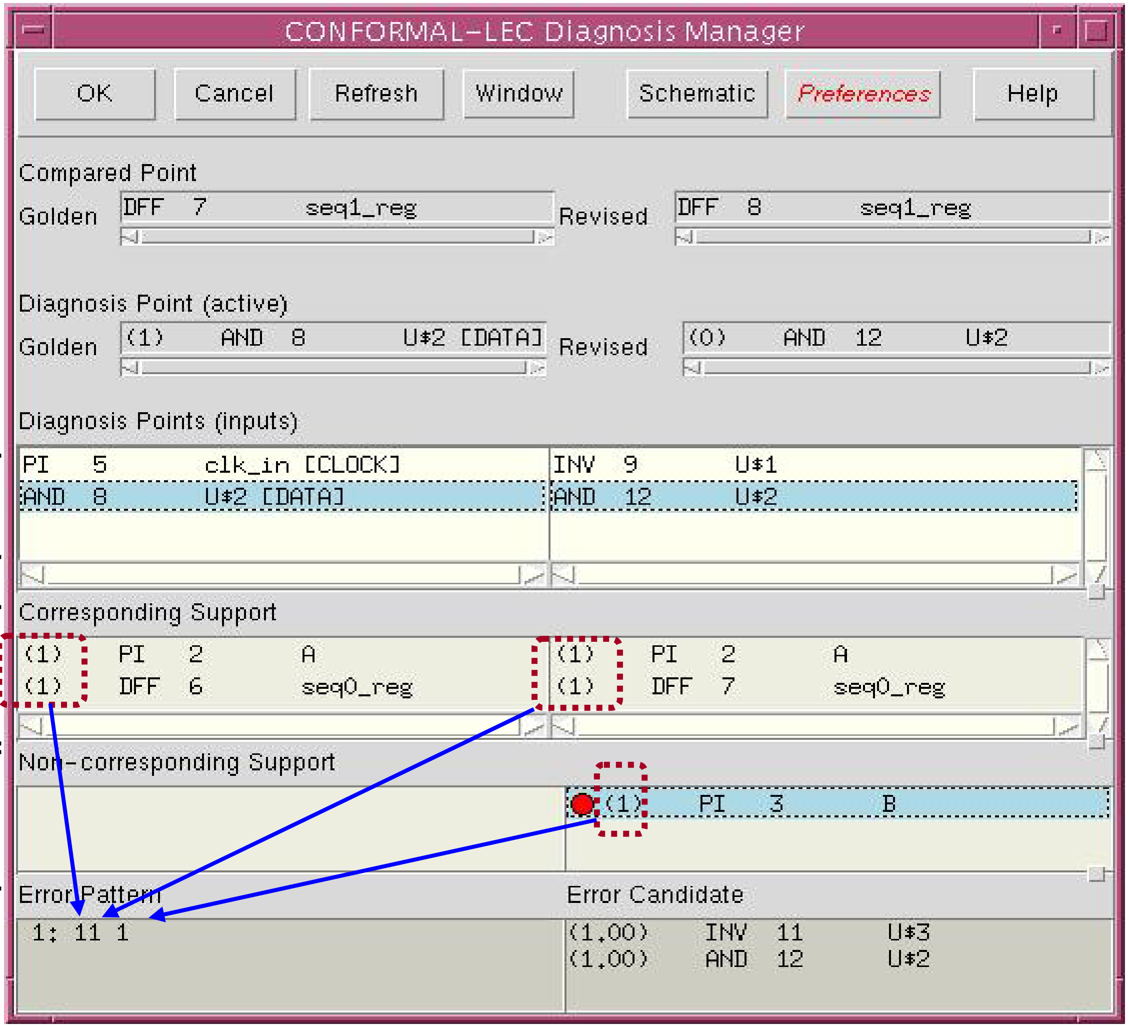
Diagnosis Points

Corresponding Support

Non-corresponding Support

● Non-corresponding, and not mapped (red)

Ⓜ Non-corresponding, but mapped (yellow with M)



Compared Point

Golden: DFF 7 seq1_reg | Revised: DFF 8 seq1_reg

Diagnosis Point (active)

Golden: (1) AND 8 U$2 [DATA] | Revised: (0) AND 12 U$2

Diagnosis Points (inputs)

PI 5 clk_in [CLOCK]	INV 9 U$1
AND 8 U$2 [DATA]	AND 12 U$2

Corresponding Support

(1) PI 2 A	(1) PI 2 A
(1) DFF 6 seq0_reg	(1) DFF 7 seq0_reg

Non-corresponding Support

● (1) PI 3 B

Error Pattern

1: 11 1

Error Candidate

(1.00) INV 11 U$3
(1.00) AND 12 U$2

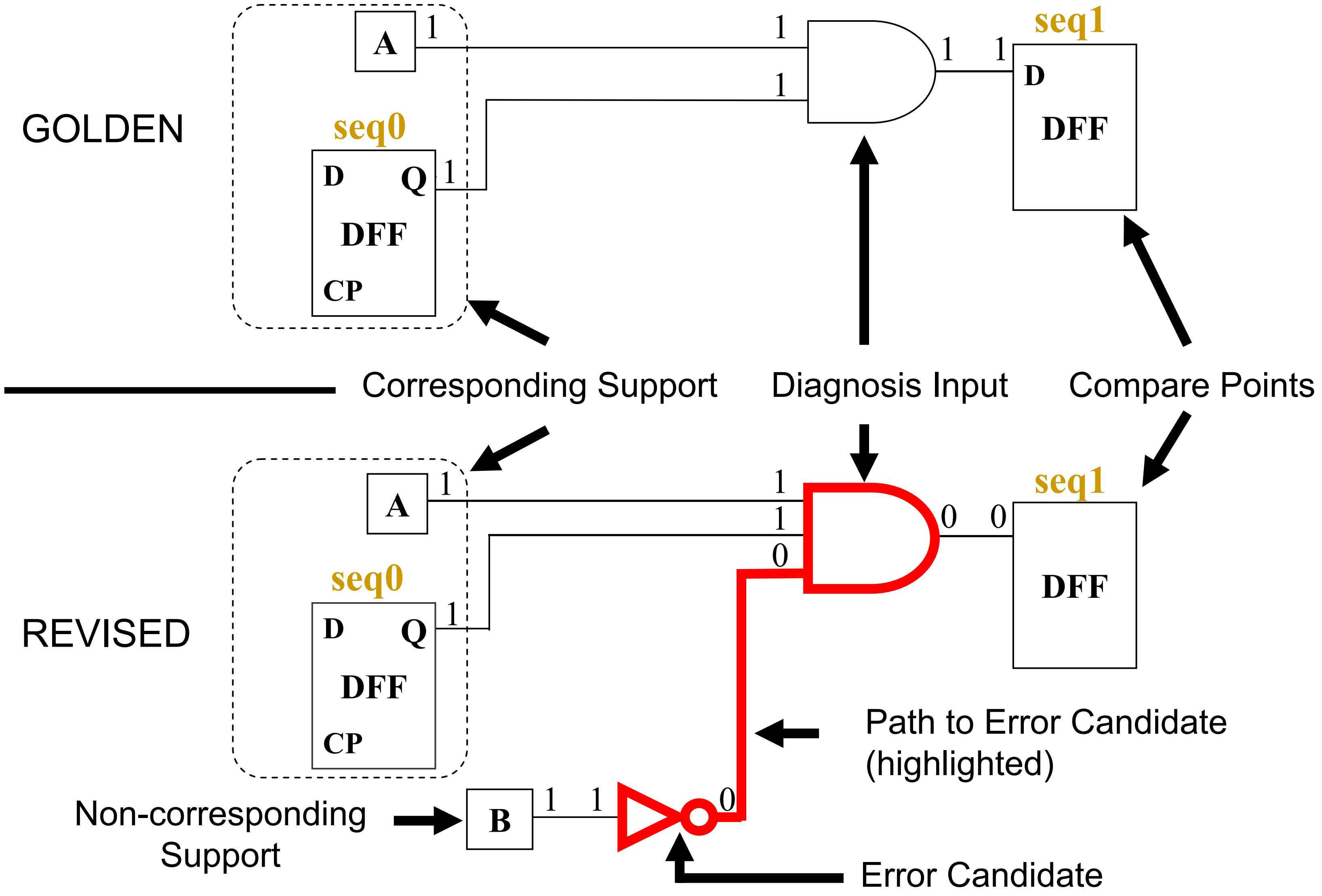
Error Patterns

Error Candidates

Diagnosis information

- **Compared Point**: Non-equivalent compared point
- **Diagnosis Point (Active)**: Point at which diagnosis failed. Simulation value shown in parenthesis ( )
- **Diagnosis Point (Inputs)**: Lists all the fanin diagnosis points for the compare point
- **Corresponding Support**: Displays the mapped points that are in the fanin cone of the diagnosis point of both the Golden and Revised designs. Simulation values of the corresponding support points are shown along with the key point names
- **Non-corresponding Support**: Displays the mapped or unmapped points that are in the fanin cone of the diagnosis point for either the Golden or Revised designs. Simulation values of the non-corresponding support points are shown along with the key point names
- **Error Pattern**: Test vector proving the diagnosis point to be non-equivalent
- **Error Candidate**: Gates in the Revised with highest probability of causing non-equivalence

Diagnosis information



GOLDEN

REVISED

Corresponding Support

Diagnosis Input

Compare Points

Path to Error Candidate (highlighted)

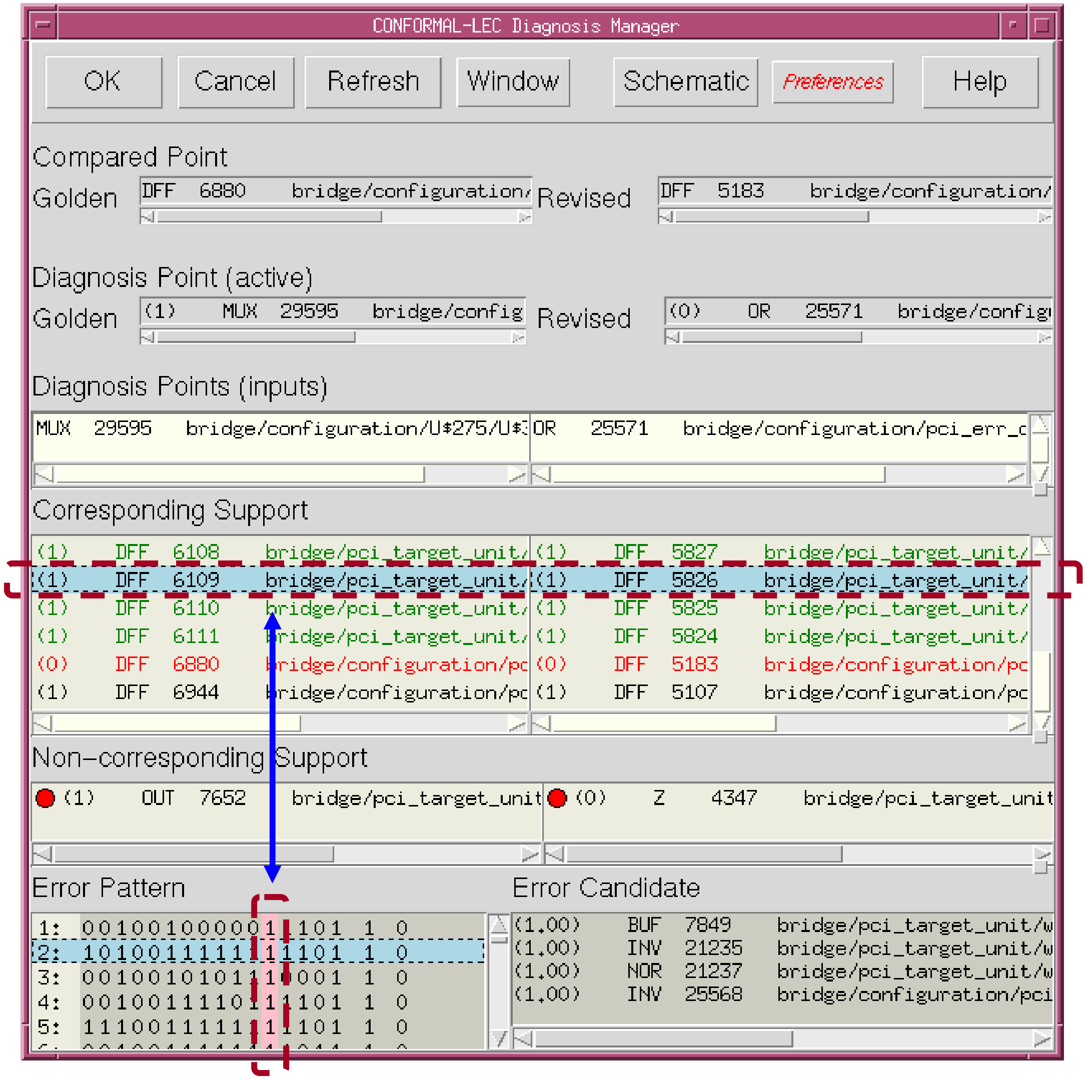
Non-corresponding Support

Error Candidate

Diagnosis Manager: New features

- Color coding for corresponding support points
- Cross highlighting of support key point and error pattern

Note: These features are only available to LEC 3.3.1.a or later



Compared Point

```
Golden   DFF  6880    bridge/configuration/   Revised   DFF  5183    bridge/configuration/
```

Diagnosis Point (active)

```
Golden   (1)    MUX  29595   bridge/config   Revised   (0)    OR   25571   bridge/config
```

Diagnosis Points (inputs)

```
MUX  29595   bridge/configuration/U$275/U$3   OR   25571   bridge/configuration/pci_err_c
```

Corresponding Support

```
(1)   DFF  6108   bridge/pci_target_unit/  (1)   DFF  5827   bridge/pci_target_unit/
(1)   DFF  6109   bridge/pci_target_unit/  (1)   DFF  5826   bridge/pci_target_unit/
(1)   DFF  6110   bridge/pci_target_unit/  (1)   DFF  5825   bridge/pci_target_unit/
(1)   DFF  6111   bridge/pci_target_unit/  (1)   DFF  5824   bridge/pci_target_unit/
(0)   DFF  6880   bridge/configuration/pc  (0)   DFF  5183   bridge/configuration/pc
(1)   DFF  6944   bridge/configuration/pc  (1)   DFF  5107   bridge/configuration/pc
```

Non-corresponding Support

```
● (1)   OUT  7652   bridge/pci_target_unit   ● (0)   Z   4347   bridge/pci_target_unit
```

Error Pattern

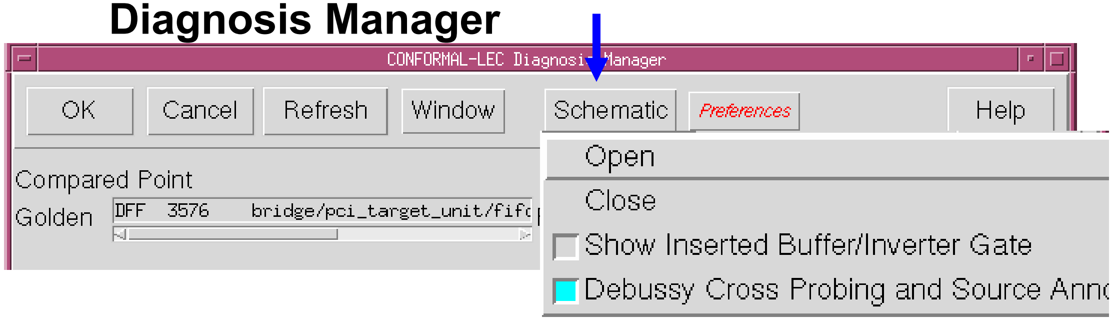
```
1:  0 0 1 0 0 1 0 0 0 0 0 1 1 1 0 1  1  0
2:  1 0 1 0 0 1 1 1 1 1 1 1 1 1 0 1  1  0
3:  0 0 1 0 0 1 0 1 0 1 1 1 0 0 0 1  1  0
4:  0 0 1 0 0 1 1 1 1 0 1 1 1 1 0 1  1  0
5:  1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1  1  0
```

Error Candidate

```
(1.00)   BUF   7849    bridge/pci_target_unit/w
(1.00)   INV   21235   bridge/pci_target_unit/w
(1.00)   NOR   21237   bridge/pci_target_unit/w
(1.00)   INV   25568   bridge/configuration/pci
```

Schematic viewer

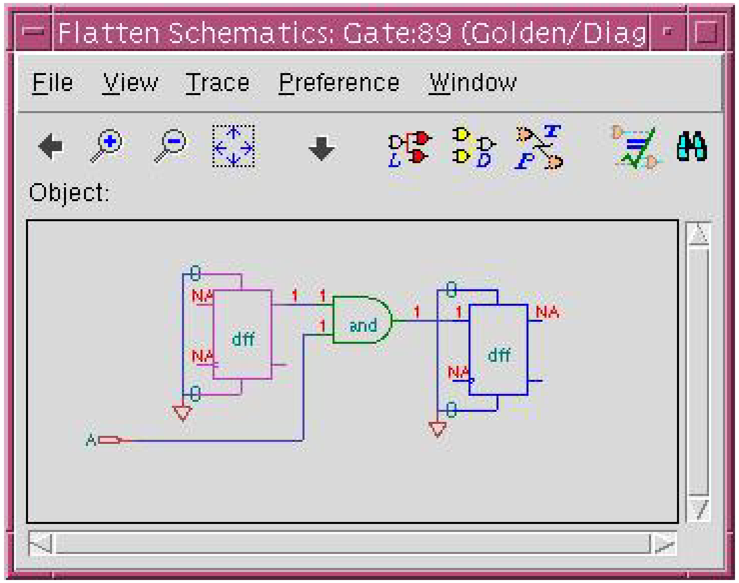
Diagnosis Manager

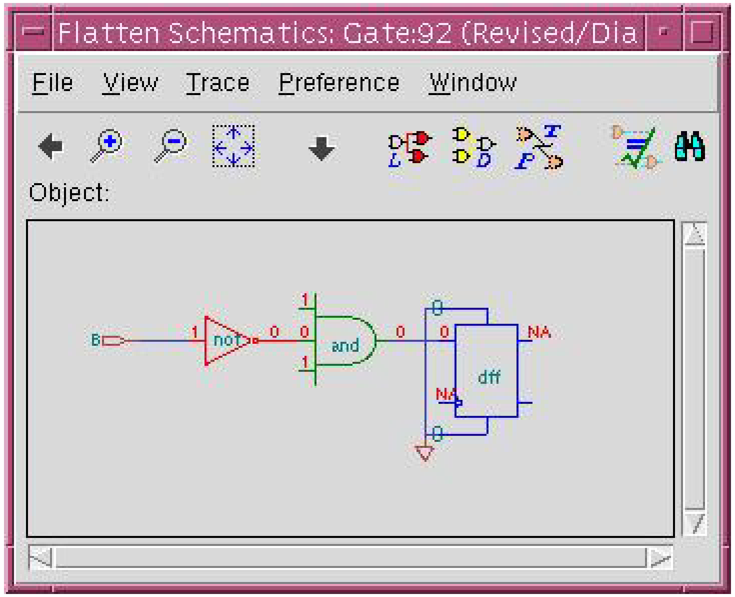

CONFORMAL-LEC Diagnosis Manager
OK
Cancel
Refresh
Window
Schematic
Preferences
Help
Open
Close
□ Show Inserted Buffer/Inverter Gate
■ Debussy Cross Probing and Source Annc

Compared Point

Golden DFF 3576 bridge/pci_target_unit/fifc

Can invoke the Source Code Manager by double clicking on any gate





Schematic Tool Bar

Last View : Return to the previous schematic view

Zoom to Full : Display all the contents of the schematic

Push View Up : Displays the parent level hierarchy

Push View Down : Displays the lower level of hierarchy

Trace Load : Highlight the load in red

Trace Driver : Highlight the driver in yellow

Trace 2 Points : Trace path between 2 points in the schematic

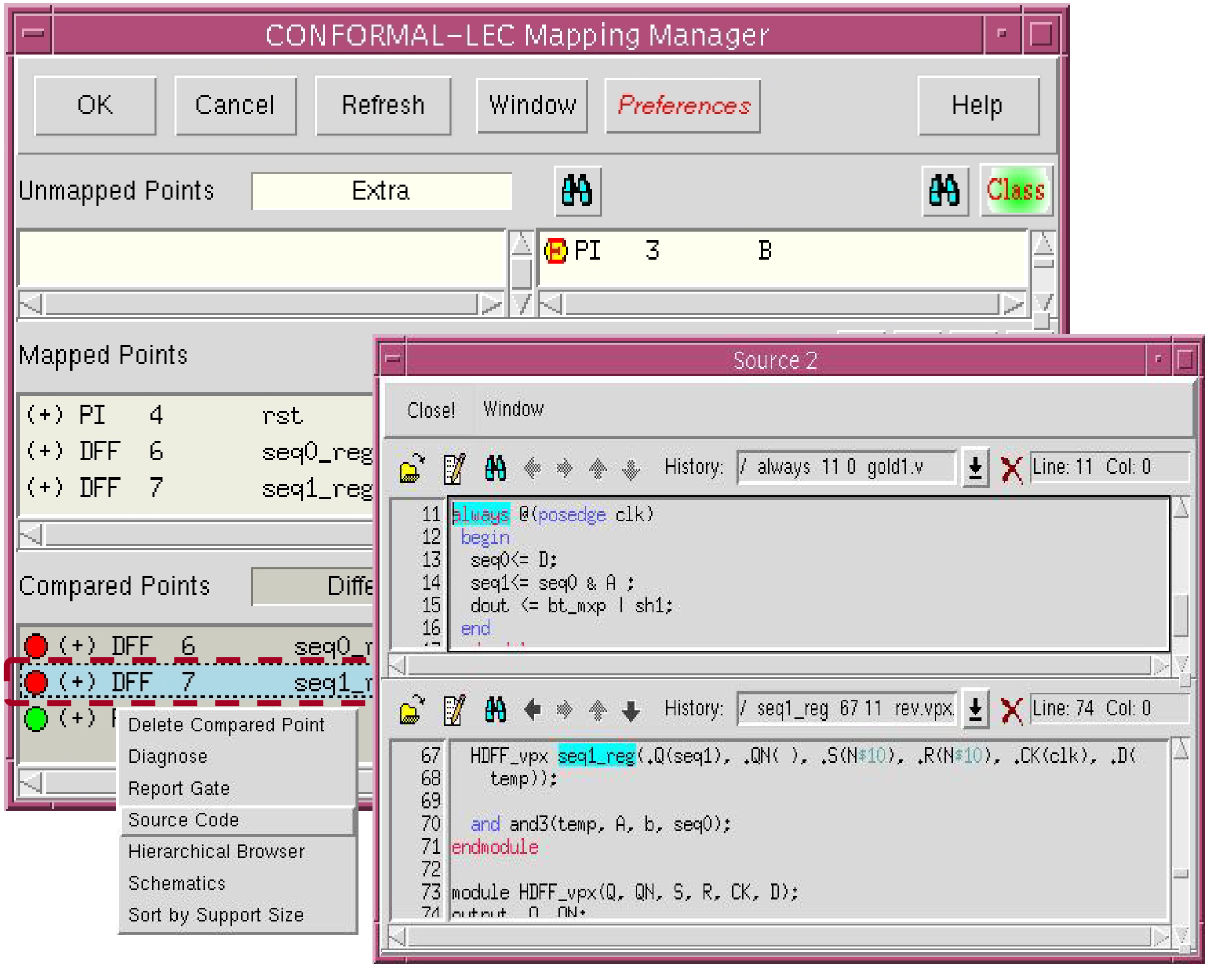
Prove : Prove equivalency between 2 points in the schematic

Find : Find and highlight the specified object

Note: Trace 2 Points and Prove are only available in the Flatten Schematics.

Source Code Manager

- Corresponding gates on Golden and Revised are highlighted
- Signal can be traced across modules



```
E PI    3         B
```

```
(+) PI    4         rst
(+) DFF   6         seq0_reg
(+) DFF   7         seq1_reg
```

```
● (+) DFF   6         seq0_r
● (+) DFF   7         seq1_r
● (+) I
```

History: / always 11 0 gold1.v

```
always @(posedge clk)
 begin
  seq0<= D;
  seq1<= seq0 & A ;
  dout <= bt_mxp | sh1;
 end
```

History: / seq1_reg 67 11 rev.vpx

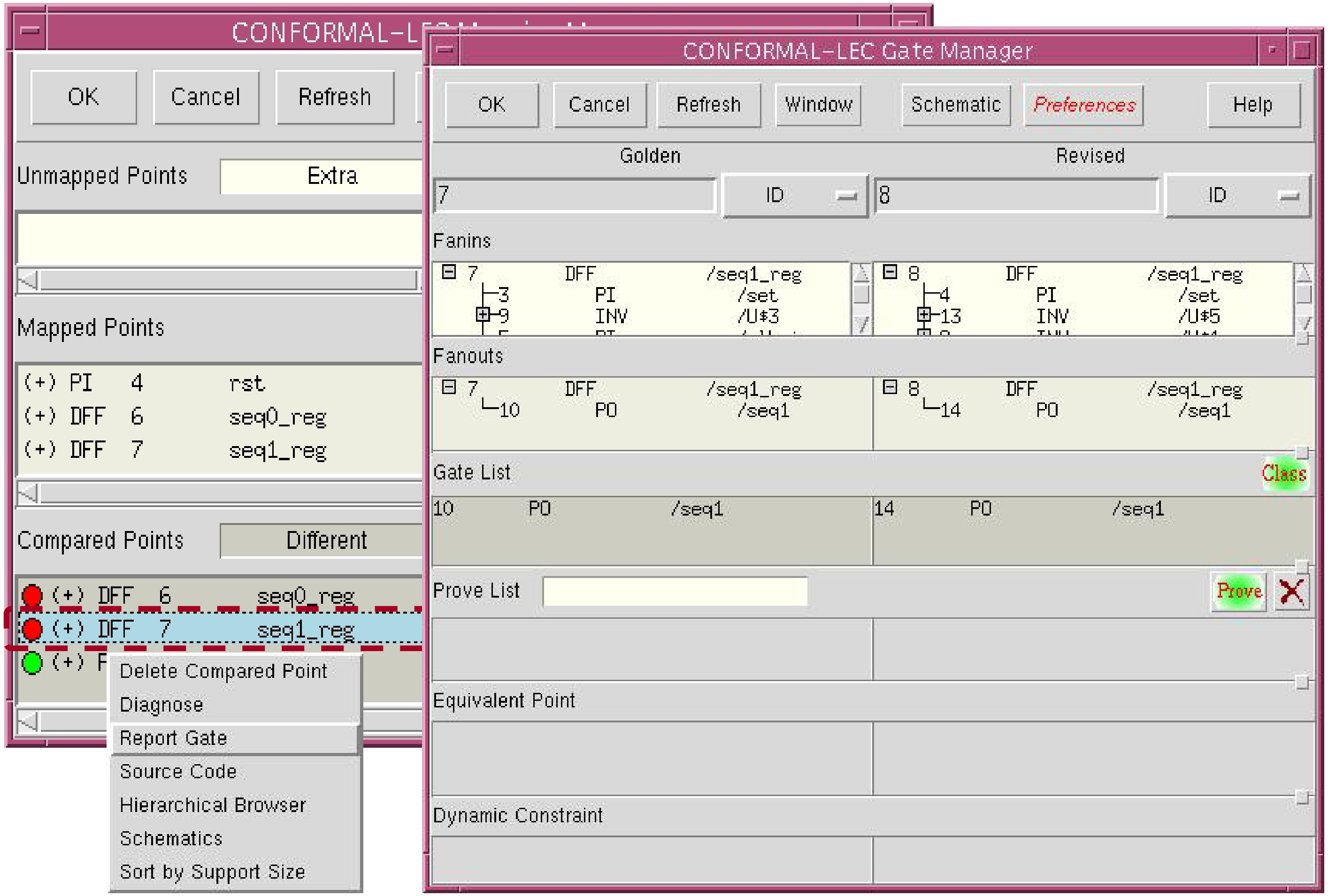
```
  HDFF_vpx seq1_reg(.Q(seq1), .QN( ), .S(N$10), .R(N$10), .CK(clk), .D(
    temp));

  and and3(temp, A, b, seq0);
endmodule

module HDFF_vpx(Q, QN, S, R, CK, D);
output Q, QN;
```

Gate Manager

Can be used to browse design connectivity



Mapped Points

```
(+) PI   4    rst
(+) DFF  6    seq0_reg
(+) DFF  7    seq1_reg
```

Compared Points

```
● (+) DFF  6    seq0_reg
● (+) DFF  7    seq1_reg
● (+) F
```

Golden: 7 ID — Revised: 8 ID

Fanins

```
⊟ 7      DFF    /seq1_reg        ⊟ 8      DFF    /seq1_reg
  ├3      PI     /set              ├4      PI     /set
  ⊞9      INV    /U$3              ⊞13     INV    /U$5
```

Fanouts

```
⊟ 7      DFF    /seq1_reg        ⊟ 8      DFF    /seq1_reg
  └10     PO     /seq1             └14     PO     /seq1
```

Gate List

```
10     PO     /seq1              14     PO     /seq1
```

A typical session (flat compare)

SETUP

- Saving LEC transcript to a log file
- Specifying black boxes
- Reading libraries and designs
- Specifying design constraints
- Specifying modeling directives
- Switching to LEC mode

LEC

- Mapping process
  - Resolving unmapped key points
- Compare process
  - Debugging non-equivalent key points
- Report run statistics

LEC run statistics

- Report:
  - Total CPU runtime
  - Memory usage

```
set log file logfile.$LEC_VERSION -replace
add notranslate module *sram* -library -both
read design cpu_rtl.v -verilog -golden
read design -file verilog.vc -verilog -revised
add pin constraint 0 scan_en -revised
set flatten model -latch_fold
add renaming rule rule0 "abc "xyz" -map -revised
add renaming rule rule1 "xyz" "C" -map -golden
set system mode lec
add compare points -all
compare
usage
...
```

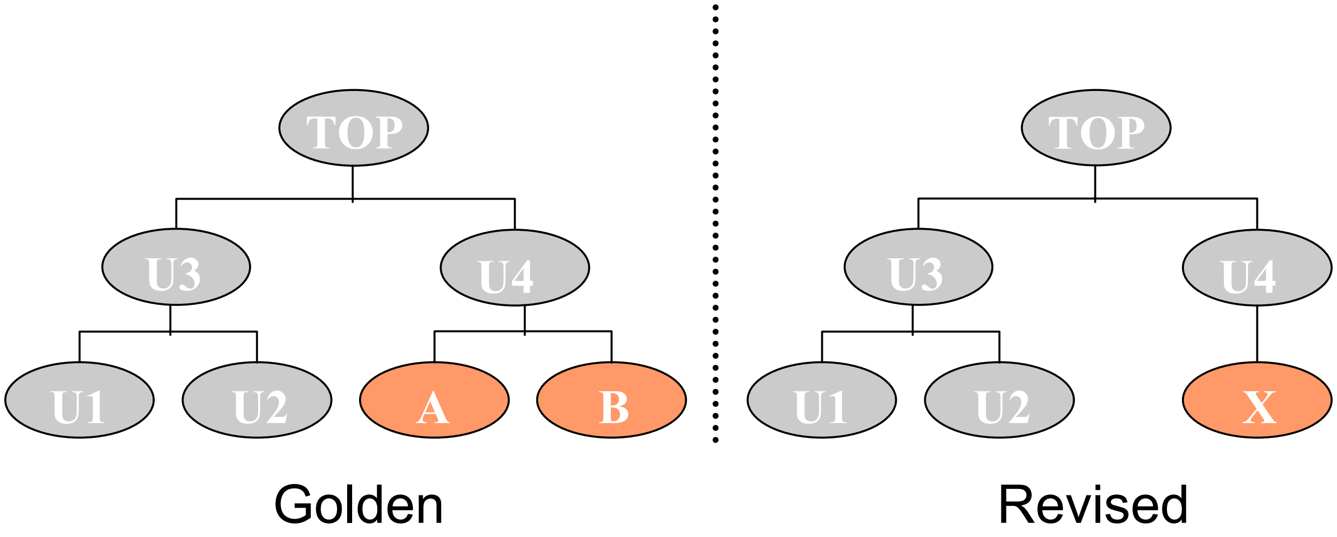
LEC usage

- Introduction to LEC
  - Getting familiar with LEC
  - LEC modes of operation
  - LEC flow
- A typical session (flat compare)
- Hierarchical compare

Introduction

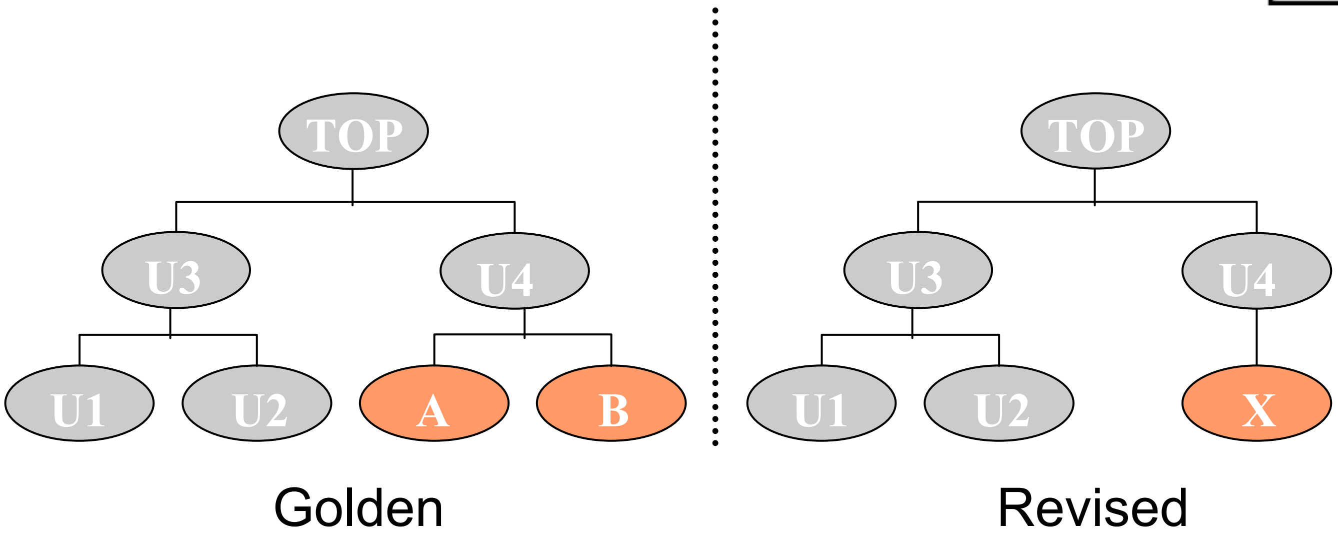
- What is Hierarchical Compare?
- Flow of Hierarchical Compare
- Setup for Hierarchical Compare
- Skipped modules in Hierarchical Compare
- Cross-boundary optimization

What is Hierarchical Compare?



- A bottom-up, module-by-module comparison
- Requirement: Designs must contain some hierarchy
- Benefits: Shorter runtime, easier to debug
- Automatic: Tool-generated script or dofile

Flow



1) Set root module to U1 → Compare U1 → Save U1 Result → Black Box U1

2) Set root module to U2 → Compare U2 → Save U2 Result → Black Box U2

3) Set root module to U3 → Compare U3 → Save U3 Result → Black Box U3

4) Set root module to U4 → Compare U4 → Save U4 Result → Black Box U4

5) Set root module to TOP → Compare TOP → Save TOP Result

Note: Modules U4 and their sub-modules will be compared flat

Setup for Hier Compare:
A typical dofile

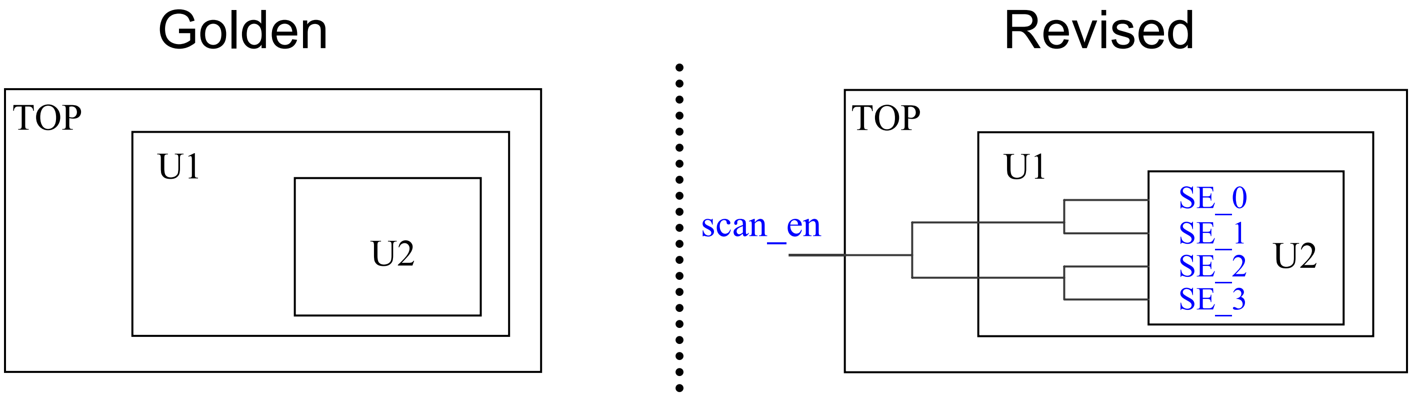
```
set log file hier.log -replace
read design rtl.v -verilog -gold
read design -file verilog.vc gate.v -rev
add renaming rule rule0 "abc" "xyz" -map -rev
set flatten model -latch_fold
//
// --- running in flat comparison mode
//
// set system mode lec
// add compare points -all
// compare
// usage
//
// --- running in hierarchical mode ---
//

write hier dofile hier.dofile -replace
dofile hier.dofile
```

Tool-generated file: *hier.dofile*

```
...
// Comparing module 'U1'
set root module U1 -golden
set root module U1 -revised
report black box -NOHidden
set system mode lec
add compare points -all
compare
save hier_compare result
set system mode setup
add black box U1 -module -golden
add black box U1 -module -revised
// Comparing module 'U2'
set root module U2 -golden
set root module U2 -revised
report black box -NOHidden
set system mode lec
add compare points -all
compare
save hier_compare result
set system mode setup
add black box U2 -module -golden
add black box U2 -module -revised
...
```

Constraint propagation



Automatic propagation to all lower-level modules

- Constraint values
- Pin equivalence relationships

```
set log file hier.log -replace
read design rtl.v -verilog -golden
read design -file verilog.vc gate.v -revised
add pin constraint 0 scan_en -revised
write hier dofile hier.dofile -constraint -noexact -replace
dofile hier.dofile
```

Resulting *hier.dofile*

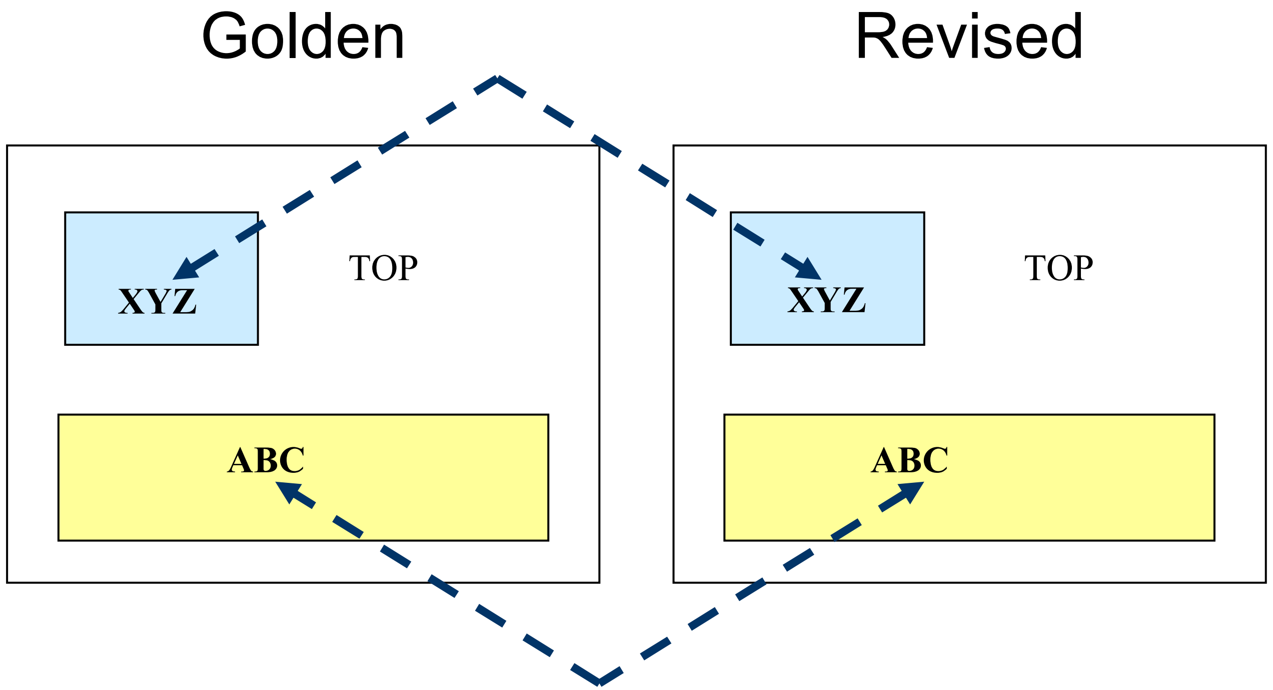
```
...
set system mode setup
//
// Comparing module ’U2’
//
set root module U2 -golden
set root module U2 -revised
add pin equivalences SE_0 SE_1 -revised
add pin equivalences SE_0 SE_2 -revised
add pin equivalences SE_0 SE_3 -revised
add pin constraint 0 SE_0 -revised
report black box -NOHidden
set system mode lec
add compare points -all
compare
save hier_compare lec result
add black box U2 -module -golden
add black box U2 -module -revised
...
```

Skipped modules

- Hierarchical script generation may skip modules due to:
  - Module name mismatch
  - Extra ports
  - Non-compatible instantiation
  - Unbalanced instantiation
- Attempt to maximize hierarchical comparison by reducing skipped modules to:
  - Minimize runtime
  - Isolate abort modules
  - Isolate non-equivalent points

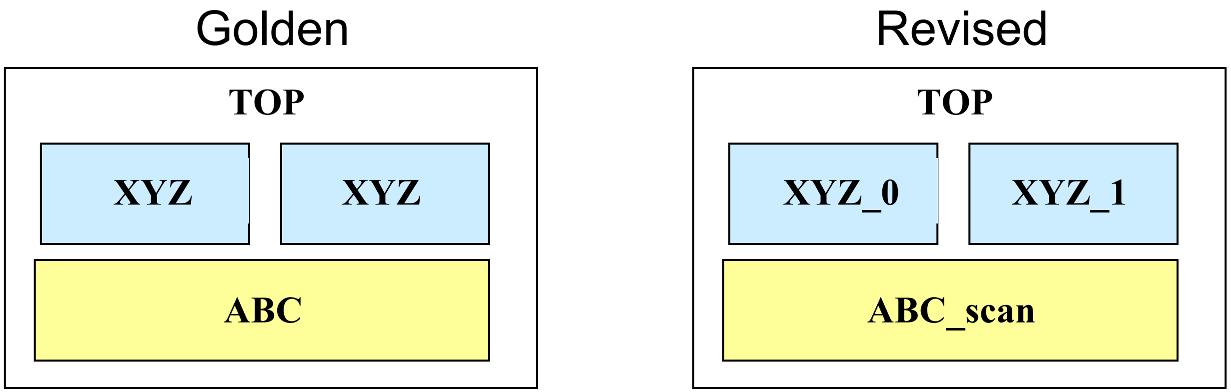
Module mapping

Pairing modules with the same *module names*



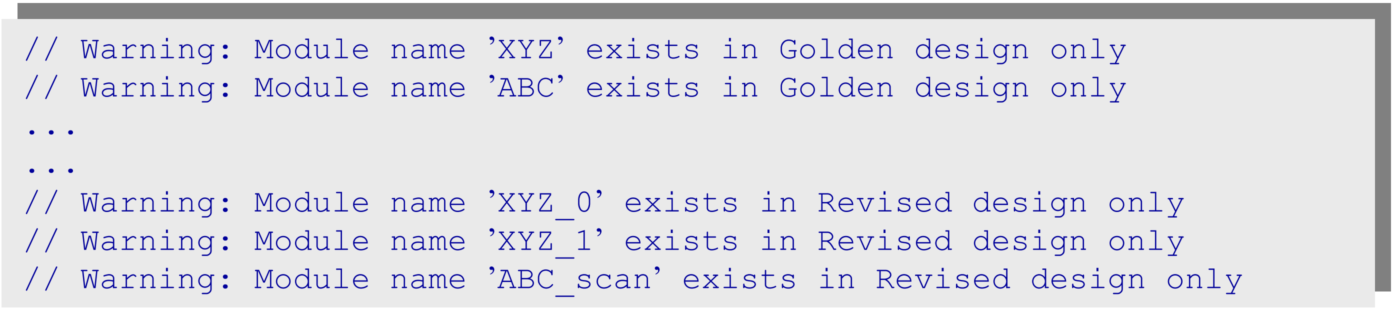
Module name mismatch: Problem

- Module names change due to uniquification during synthesis or backend flow



SETUP> `write hier dofile hier.dofile -constraint -noexact`

Transcript Window



```
// Warning: Module name 'XYZ' exists in Golden design only
// Warning: Module name 'ABC' exists in Golden design only
...
...
// Warning: Module name 'XYZ_0' exists in Revised design only
// Warning: Module name 'XYZ_1' exists in Revised design only
// Warning: Module name 'ABC_scan' exists in Revised design only
```

Module name mismatch: Solution

- Method 1:
  - Use module renaming rule to fix module name differences

```
set log file hier.log -replace
read design rtl.v -verilog -golden
read design -file verilog.vc gate.v -revised
add pin constraint 0 scan_en -revised
add renaming rule m1 "%s_%d$" "@1" -module -revised
add renaming rule m2 "%s_scan$" "@1" -module -revised
write hier dofile hier.dofile -constraint -noexact -replace
dofile hier.dofile
```

- Method 2:
  - Use `uniquify` command (explained later)

    SETUP> `uniquify XYZ -golden`

Extra ports

- If you use the constraint option and this removes extra ports, module may be compared
  - Most common examples are scan ports and CT insertion
- Bit Blasting - Array ports changed to single ports
  - `d_in[7:0] => d_in_7, d_in_6, ...`

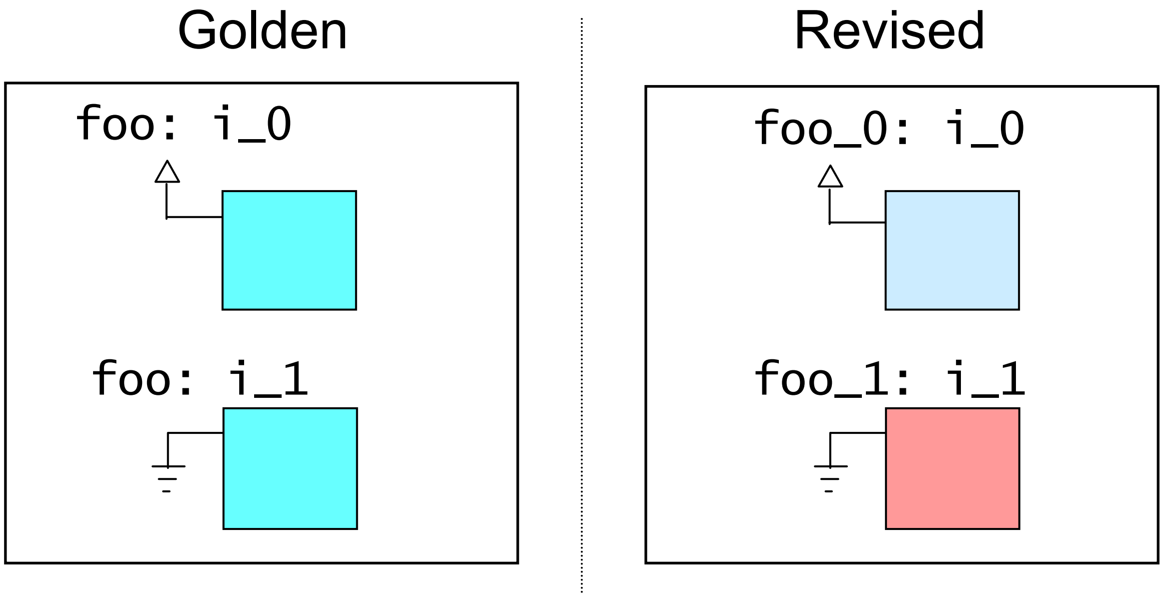
```
set log file hier.log -replace
read design rtl.v -verilog -golden
read design -file verilog.vc gate.v -ver -rev
add pin constraint 0 scan_en -revised
add renaming rule m1 %s_%d$ @1 -module -revised
add renaming rule m2 %s_scan$ @1 -module -revised

add renaming rule r1 "d_in\[%d\]" "d_in_@1" -pin -bbox <module_name>

write hier dofile hier.dofile -constraint -replace
dofile hier.dofile
```

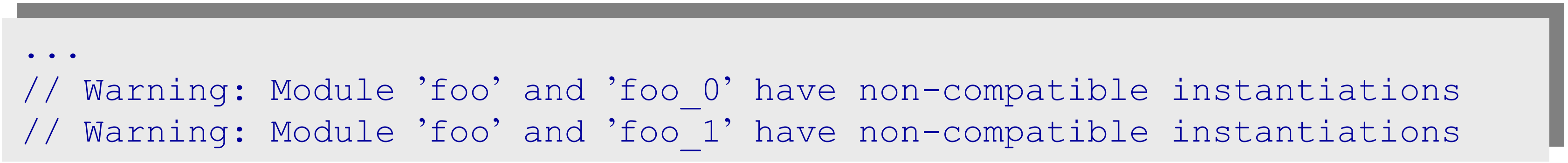
Non-compatible instantiation: Problem

- Conflicting instantiation: will skip module `foo`



```
SETUP> write hier dofile hier.dofile -constraint -noexact
```

Transcript Window

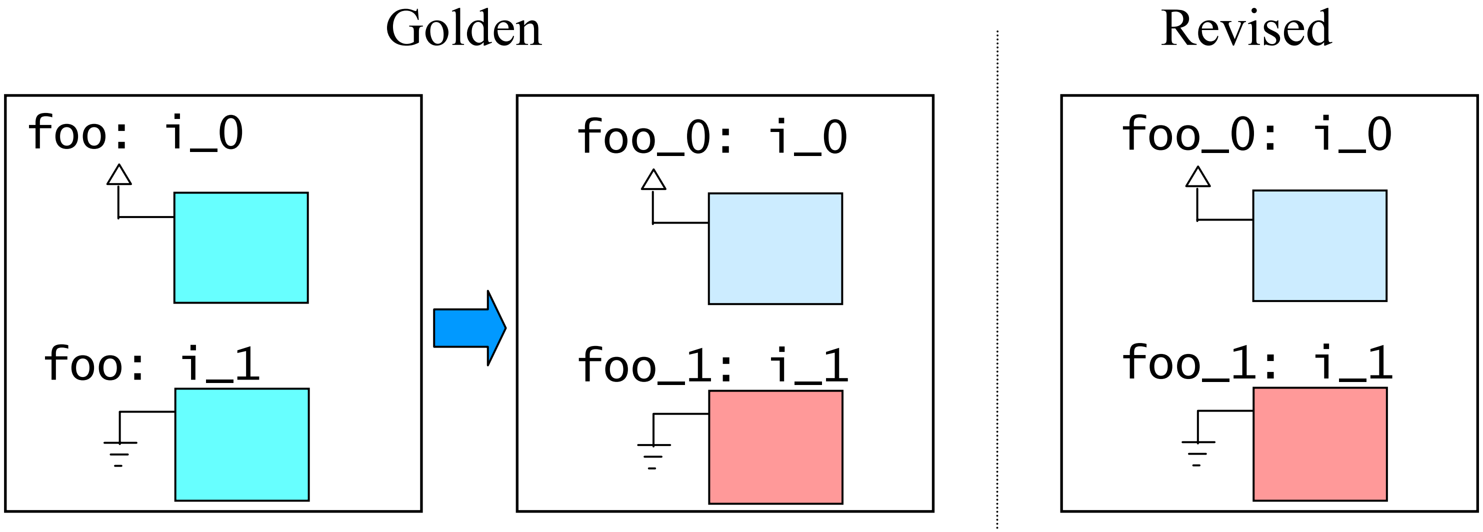


```
...
// Warning: Module 'foo' and 'foo_0' have non-compatible instantiations
// Warning: Module 'foo' and 'foo_1' have non-compatible instantiations
```

Non-compatible instantiation: Solution

Use `uniquify` command

- ✧ Module foo is replaced by `foo_0` and `foo_1`
- ✧ Determine names by looking at the other design

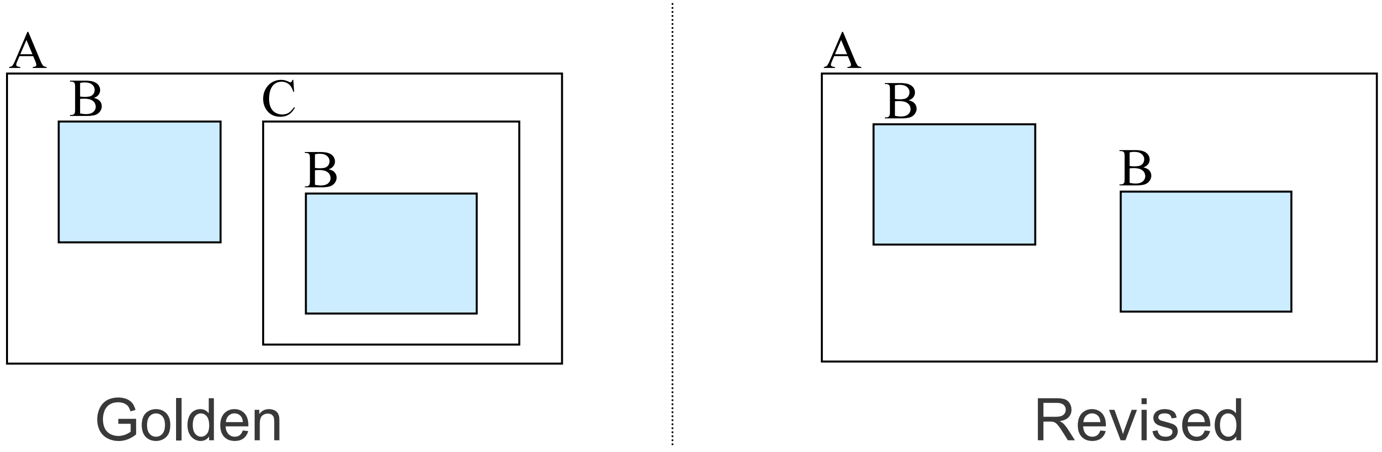


```
read design rtl.v -verilog -golden
read design -file verilog.vc gate.v -ver -rev
add pin constraint 0 scan_en -revised
uniquify foo -golden
write hier dofile hier.dofile -constraint -noexact -replace
dofile hier.dofile
```

Transcript Window

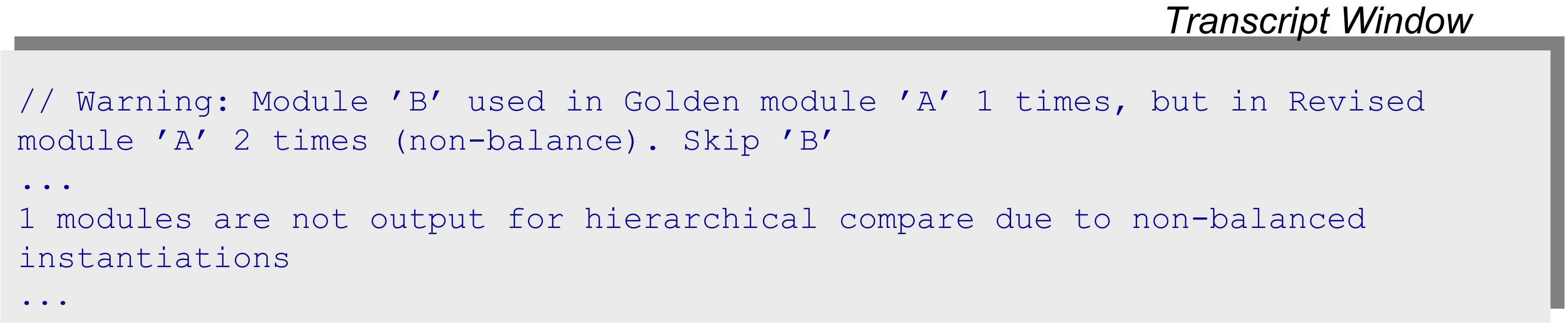
Unbalanced instantiation: Problem

Hierarchical script will skip module B



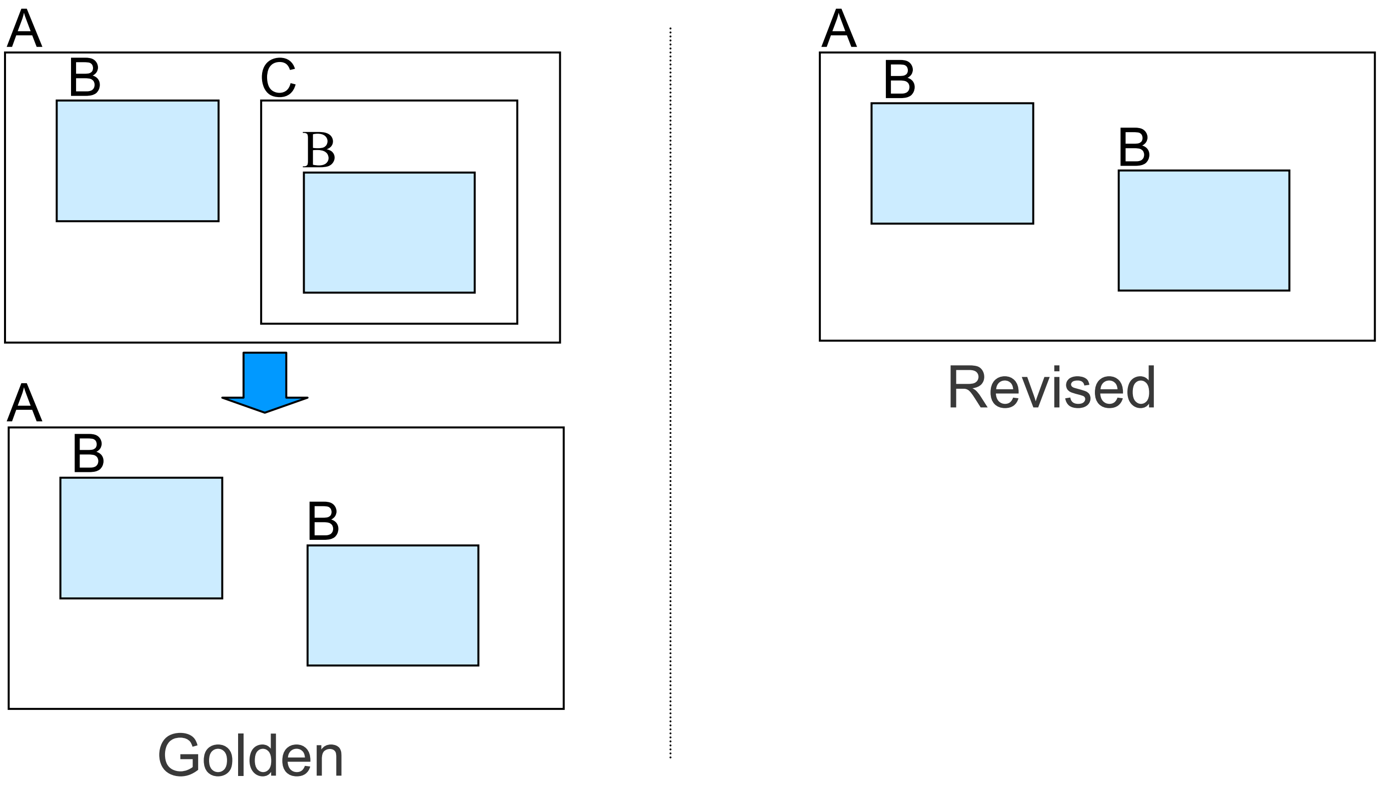
SETUP> `write hier dofile hier.dofile -constraint -noexact`

Transcript Window



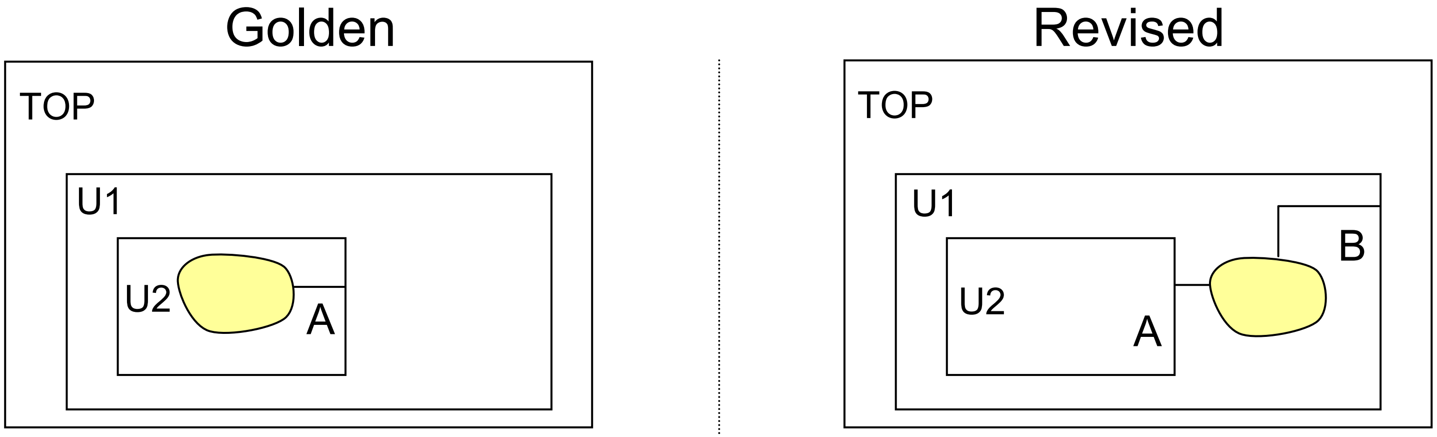
```
// Warning: Module 'B' used in Golden module 'A' 1 times, but in Revised
module 'A' 2 times (non-balance). Skip 'B'
...
1 modules are not output for hierarchical compare due to non-balanced
instantiations
...
```

Unbalanced instantiation: Solution



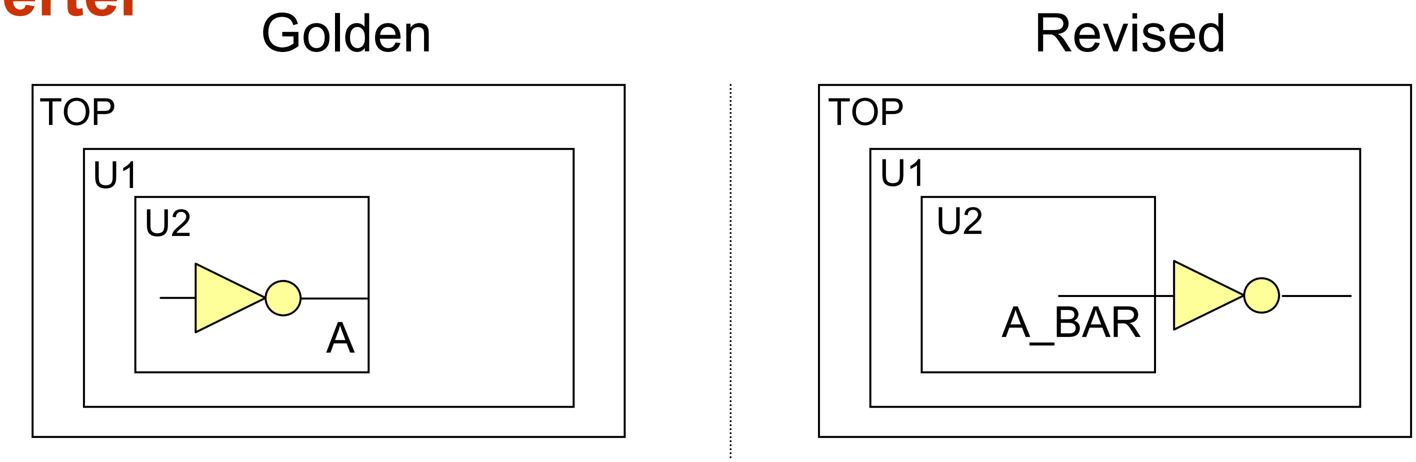
```
read design rtl.v -verilog -golden
read design -file verilog.vc gate.v -ver -rev
add pin constraint 0 scan_en -revised
resolve C -golden
write hier dofile hier.dofile -constraint -replace
dofile hier.dofile
```

Cross-boundary optimization



- User might allow logic to move across module boundary during synthesis optimization
- False negative during LEC hierarchical compare
  - Miscompare `U2` at point `A` and `U1` at point `B`
- Two forms of logic rearrangement:
  - Simple inverter
  - All other cases

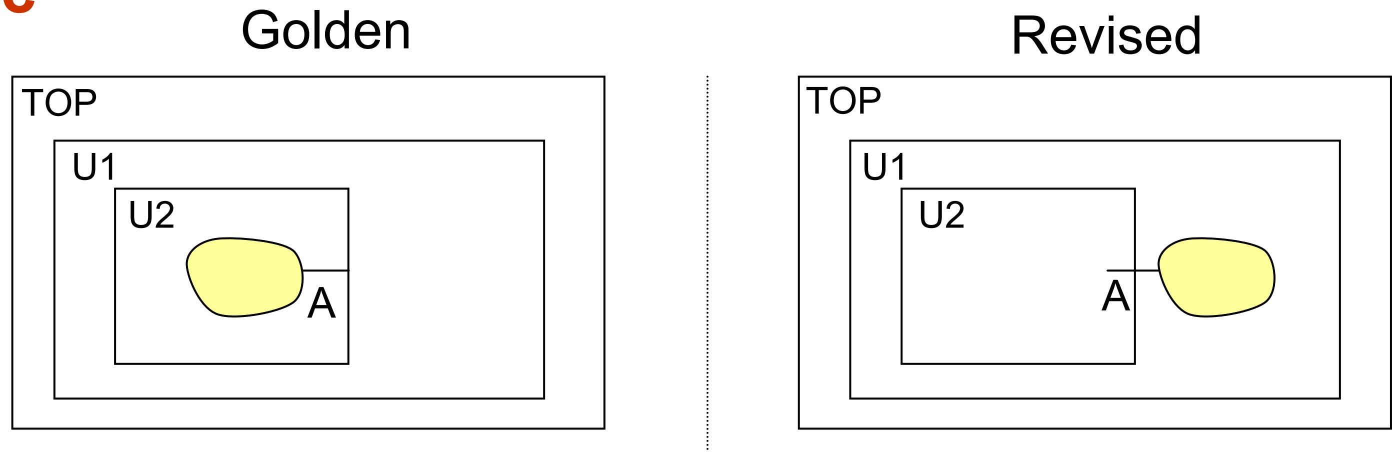
Logic rearrangement - Case 1: Simple inverter



- Inverters moved across module boundary
- Must `set naming rule` before read library/design
- LEC maintains inversion relationship

```
set log file hier.log -replace
set naming rule _BAR -inverted_pin -golden
read design rtl.v -verilog -golden
read design -file verilog.vc gate.v -revised
add pin constraint 0 scan_en -revised
write hier dofile hier.dofile -constraint -noexact -replace
dofile hier.dofile
```

Logic rearrangement - Case 2: General logic



- Block of logic moved across module boundary
- User specifies no compare at `U2` level
- Logic of `U2` is considered at `U1` level

```
set log file hier.log -replace
read design rtl.v -verilog -golden
read design -file verilog.vc gate.v -revised
add pin constraint 0 scan_en -revised
add noblack box U2 –both
write hier dofile hier.dofile -constraint -noexact -replace
dofile hier.dofile
```

Exercise answer

Renaming rules	Final names
1. Add re r r1 "u1\/" "" -gold	/u0/c_reg[5]
2. Add re r r2 "c_\.ram%d_reg" "c_reg[@1]" -rev	/u0_u1_c_reg[1]
3. Add re r r3 "b_%d_reg\[%d\]" "b_reg[#(@1+@2)]" -rev	/u0/b_reg[24]

Lab 1

Objective: Read VHDL and Verilog designs, and libraries

1. UNIX % `cd ~/lec_labs/setup/single_language`
2. Type "`lec`" to start GUI LEC
3. Read VHDL design (./RTL/mult_sign_oper.vhd) along with its pkg libraries all in one command. *mult_sign_oper.vhd* uses DPAK_LIB, FCTN_LIB, and IEEE libraries.

   Use the –mapfile option to map the file ./vhdl_lib/dpak_unit/minimum_pkg.vhd for DPAK_LIB library

   Use the –map option to map the whole directory ./vhdl_lib/common/ for FCTN_LIB library.
4. Read the library ./verilog_lib/lib.v for the revised design
5. Read the revised design ./synthesis/multiplier_oper.gate.v
6. Click **LEC** icon to switch to LEC mode
7. Compare design from **Run** Menu
8. Use **File** Menu to save the commands you entered into a dofile. Exit LEC from **File** Menu

   Note: Solution is in the back of the manual and current working directory

Lab 2

Objective: Black box library modules; read libraries and design using Verilog command file

1. UNIX% `cd ~/lec_labs/setup/vcfile_bbox`
2. Create Verilog command files to:

   a. Read in Golden design (./RTL/golden.v) and its library elements in ./lib/std/ and ./lib/mem/ directories

   b. Read in Revised design (./GATES/revised.v) and its library elements in ./lib/std/ and ./lib/mem/ directories
3. Start LEC
4. Issue commands to black box library modules that have "rom" and "ram" in their names (for both Golden and Revised)
5. Read Golden design and libraries via Verilog command file format
6. Read Revised design and libraries via Verilog command file format
7. Exit LEC

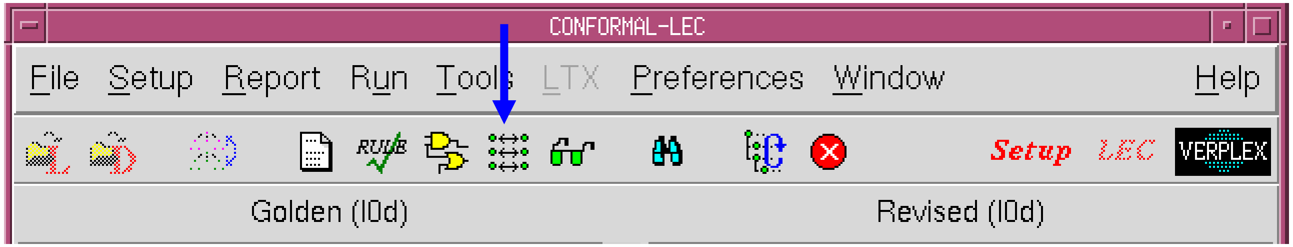
Note: Solution is in the back of the manual and current working directory

Lab 3

OBJECTIVE: Add renaming rule(s) to complete mapping key points

1. Unix % `cd ~/lec_labs/mapping/renaming1`, start LEC
2. Execute the dofile *init.dofile*
4. Open the Mapping Manager to examine the unmapped points.

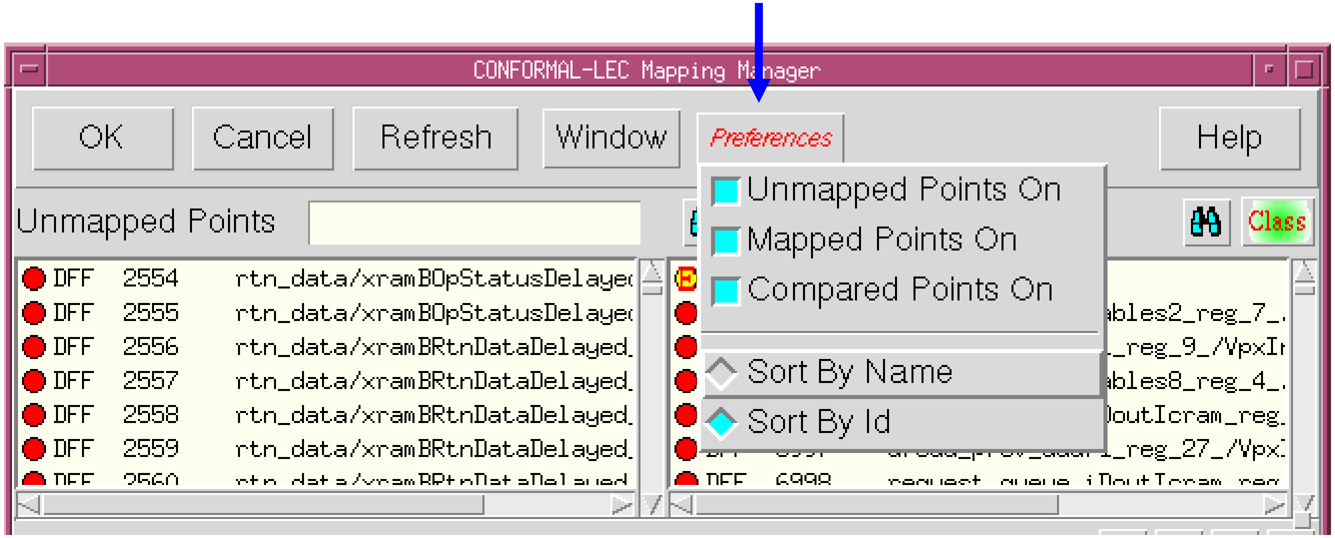
Main GUI window



Lab 3 (cont'd)

Sort key points by name to help determine the renaming rules (Preference → Sort by name)

Mapping Manager



```
● DFF   2554   rtn_data/xramBOpStatusDelaye
● DFF   2555   rtn_data/xramBOpStatusDelaye
● DFF   2556   rtn_data/xramBRtnDataDelayed_
● DFF   2557   rtn_data/xramBRtnDataDelayed_
● DFF   2558   rtn_data/xramBRtnDataDelayed_
● DFF   2559   rtn_data/xramBRtnDataDelayed_
● DFF   2560   rtn_data/xramBRtnDataDelayed
```

5. Create renaming rules to resolve unmapped key points. After rule(s) are specified, continue mapping with the command "`map key point`"

6. Make sure all key points are mapped

7. Exit LEC

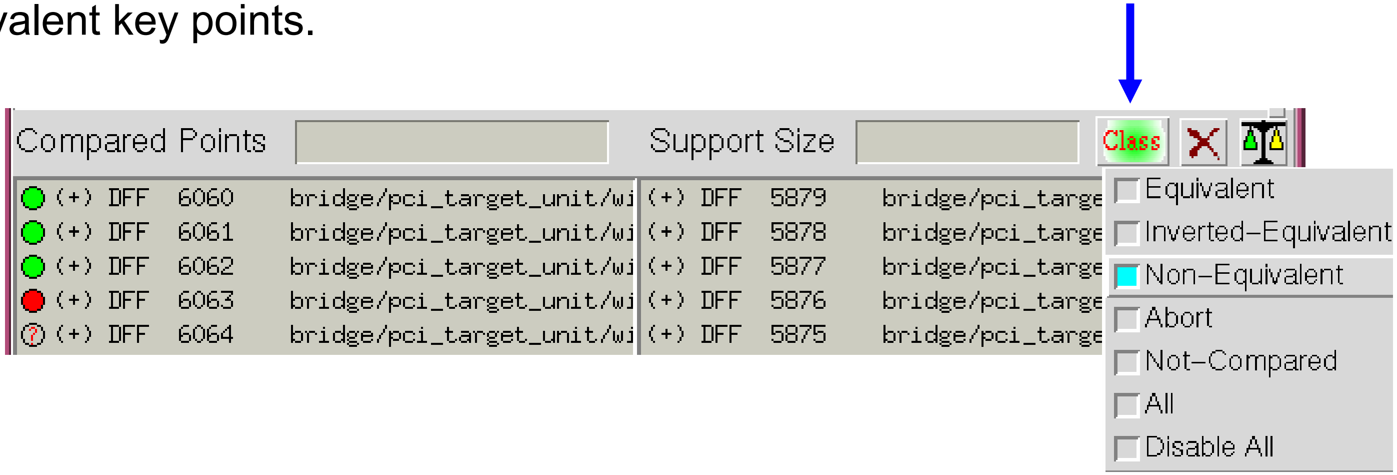
   Note: Solution is in the back of the manual and current working directory

Lab 4

OBJECTIVE: Debugging non-equivalent key points

1. Unix % `cd ~/lec_labs/compare/compare1/`, start LEC.
2. Execute the dofile *init.dofile*.
3. Invoke the **Mapping Manager.**
4. On the Mapping Manager, use "Class" pull-down menu to display only the non-equivalent key points.

Mapping Manager

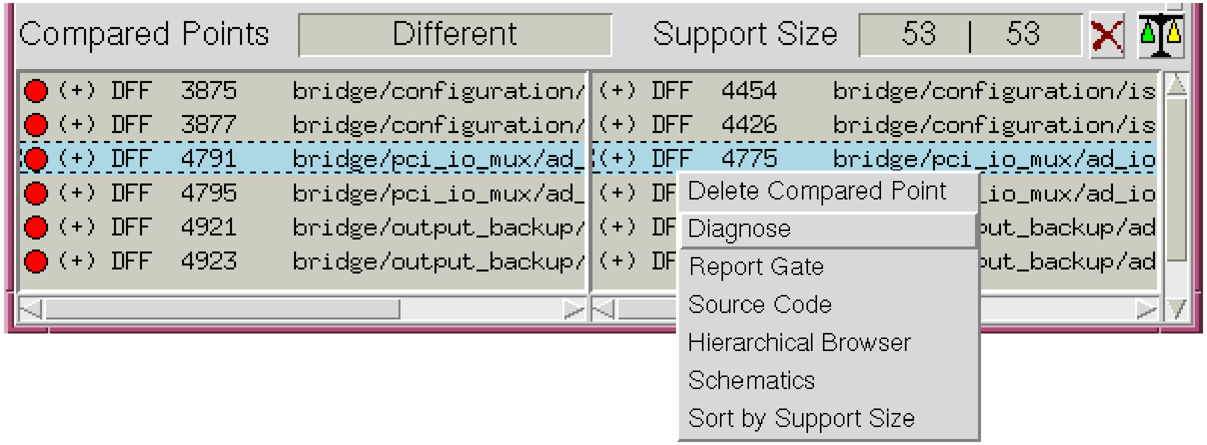


```
🟢 (+) DFF   6060    bridge/pci_target_unit/wi (+) DFF   5879    bridge/pci_targe
🟢 (+) DFF   6061    bridge/pci_target_unit/wi (+) DFF   5878    bridge/pci_targe
🟢 (+) DFF   6062    bridge/pci_target_unit/wi (+) DFF   5877    bridge/pci_targe
🔴 (+) DFF   6063    bridge/pci_target_unit/wi (+) DFF   5876    bridge/pci_targe
?  (+) DFF   6064    bridge/pci_target_unit/wi (+) DFF   5875    bridge/pci_targe
```

Lab 4 (cont'd)

5. Invoke the **Diagnosis Manager** for a non-equivalent key point.

Mapping Manager

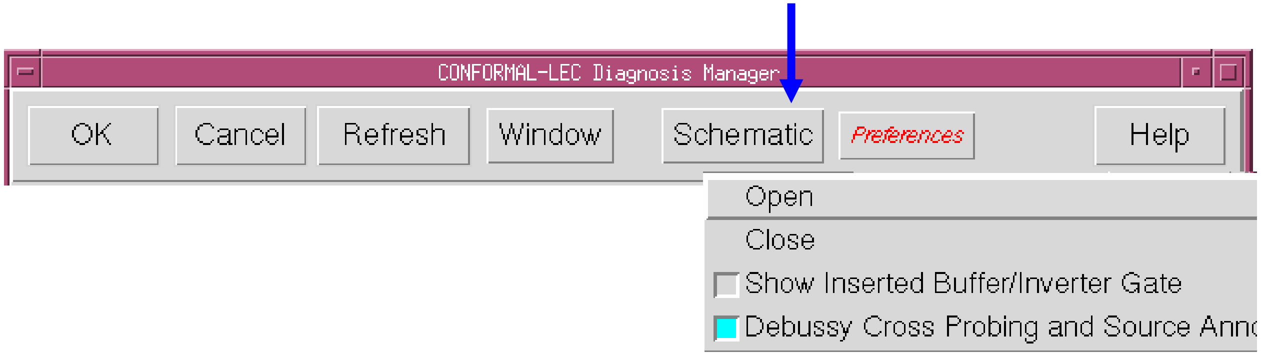


Compared Points | Different | Support Size | 53 | 53

```
● (+) DFF 3875   bridge/configuration/  (+) DFF 4454   bridge/configuration/is
● (+) DFF 3877   bridge/configuration/  (+) DFF 4426   bridge/configuration/is
● (+) DFF 4791   bridge/pci_io_mux/ad_  (+) DFF 4775   bridge/pci_io_mux/ad_io
● (+) DFF 4795   bridge/pci_io_mux/ad_  (+) DF                  _io_mux/ad_io
● (+) DFF 4921   bridge/output_backup/  (+) DF                  put_backup/ad
● (+) DFF 4923   bridge/output_backup/  (+) DF                  put_backup/ad
```

6. Invoke the Schematic Viewer from the **Diagnosis Manager**.

Diagnosis Manager



CONFORMAL-LEC Diagnosis Manager
OK
Cancel
Refresh
Window
Schematic
Preferences
Help
Open
Close
□ Show Inserted Buffer/Inverter Gate
■ Debussy Cross Probing and Source Annc

Lab 4 (cont'd)

7. On the schematic you should see a simulation mismatch for the compare point. Compare point is located at the right hand side of the schematics. Examine the schematics to find the gate that causes the mismatch. That gate causing the the mismatch is the three-input NOR gate on the Revised schematic.
8. Double click the NOR gate in the Revised design to go back to netlist.
6. Do the same to the Golden design on any gate.
7. Exit LEC and close all views.

Note: Solution is in the back of the manual and current working directory.

Lab 5

OBJECTIVE: Write dofile for hierarchical compare

1. Unix % `cd ~/lec_labs/hier_compare/hier1`, start LEC.
2. Read in *golden.v* for the Golden design.
3. Read in *revised.v* for the Revised design. The library *lib.v* is needed for the Revised design.
4. Generate hierarchical compare script.
5. Execute hierarchical compare script.

   Note: Solution is in the back of the manual and current working directory.

Lab 6

OBJECTIVE: Module Mapping

1. Unix% `cd ~/lec_labs/hier_compare/hier2`, start LEC.
2. Read in *golden.v* for the Golden design.
3. Read *revised.v* for the Revised design. The library *lib.v* is needed for the Revised design.
4. Generate hierarchical compare script.
5. Fix module mapping problem with `add renaming rule`.
6. Re-generate hierarchical compare script.

Note: Make sure 5 modules are included in the generated script.

Note: Solution is in the back of the manual and current working directory.

Lab solutions

Lab1

```
read design -vhdl ./RTL/mult_sign_oper.vhd -gold  -map FCTN_LIB \
      ./vhdl_lib/common -mapfile DPAK_LIB \
      ./vhdl_lib/dpak_unit/minimum_pkg.vhd
read library verilog_lib/lib.v -revised
read design synthesis/multiplier_oper.gate.v -verilog -revise
set system mode lec
add compare point –all
compare
```

Lab 2

```
add notranslate modules *rom* -library –both
add notranslate modules *ram* -library –both
read design golden.vc –f golden.vc –verilog -golden
read design revised.vc –f revised.vc –verilog -revised
set system mode lec
add compare point –all
compare
```

Lab solutions

Lab3

```
read design golden_rtl.v –verilog –golden
read library lib.v –verilog
read design revised_gate.v –verilog –revised
set mapping method –name only
set system mode lec
add renaming rule R1 macreg_0%d macreg_@1 -revised
add renaming rule R2 hcount del_hcount -revised
add renaming rule R3 synvalid1\/regout synvalid1/del_regout -revised
map key points
```

Lab4

Non-equivalent occurs in revised_gate.v when an "OR" gate is purposely replaced by a "NOR" gate.

Lab solutions

Lab5

```
set log file hier.log -replace
read design -golden golden.v -verilog
read library lib.v -verilog
read design -revised revised.v -verilog
write hier dofile hr.do -replace
dofile hr.do
```

Lab6

```
set log file hier.log -replace
read design -golden golden.v -verilog
read library lib.v -verilog
read design -revised revised.v -verilog
// These renaming rules will fix module mapping problems
//add renaming rule rule1 %s_scan$ @1 -module -revised
//add renaming rule rule2 %s_%d$ @1 -module -revised
write hier dofile hr.do -replace
dofile hr.do
```

Sample flat compare dofile

```
Set log file logfile.$LEC_VERSION -replace
Add notranslate module *sram* -library -both
Read design cpu_rtl.v -verilog -golden
Read design -file verilog.vc -verilog –revised
// Note: The read design above can be replaced by
// Any of the examples explained in this manual
Add pin constraint 0 scan_en -revised
Set flatten model -latch_fold
Set flatten model –seq_redundant
Report floating signals -undriven -full –both
Set mapping method -name only
//The above commands are the same for both flat and hier dofiles
Add renaming rule rule0 "abc" "xyz" -revised
Add renaming rule rule1 "xyz" "C" –golden
// Note: The renaming rules above are for a particular case only
Set system mode lec
// Test renaming rule -design -all
// Test renaming rule -print group -file group.list
// Test renaming rule -print single -file single.list
Report mapped points > mapped.log
Report unmapped points -notmapped > unmapped.log
Add compare points -all
Compare
Report compare data –noneq > noneq.log
Usage
```

Sample Hier Compare dofile

```
Set log file logfile.$LEC_VERSION -replace
Add notranslate module *sram* -library -both
Read design cpu_rtl.v -verilog -golden
Read design -file verilog.vc -verilog –revised
//Note: The read design above can be replaced by
//Any of the examples explained in this manual
Add pin constraint 0 scan_en -revised
Set flatten model -latch_fold
Set flatten model –seq_redundant
Report floating signals -undriven -full –both
Set mapping method -name only
//The above commands are the same for both flat and hier dofiles
Add black box U2 -both
Write hier dofile hier.dofile –constraint –noexact -replace
dofile hier.dofile
Usage
```

Command overview

Add / Delete / Report Commands

```
Add black box
Add compared points
Add instance constraints
Add noblack box
Add notranslate modules
Add pin constraints
Add pin equivalences
Add instance equivalences
Add renaming rule
Add tied signals
```

```
Delete black box
Delete compared points
Delete instance constraints
Delete mapped points
Delete noblack box
Delete notranslate modules
Delete pin equivalences
Delete pin constraints
Delete renaming rule
Delete tied signals
```

```
Report black box
Report compare data
Report compared points
Report design data
Report environment
Report floating signals
Report instance constraints
Report instance equivalences
Report library data
Report messages
Report modules
Report notranslate modules
Report noblack box
Report pin constraints
Report pin equivalences
Report rule check
Report renaming rule
Report statistics
Report tied signals
Report unmapped points
```

General Commands

```
Compare
Dofile
Exit
Help
Map key points
Read design
Read library
Set flatten model
Set gui
Set log file
Set mapping method
Set system mode
Set undefined cell
Set undriven signal
Uniquify
Version
Test renaming rule
Usage
```

Conformal® LEC

Logic Equivalence Checker

Advanced Training Manual

Agenda

1. LEC-1 review
2. Mapping Problems
3. Debugging & Diagnosis Tips
4. Multipliers
5. Resolving abort key points
6. RTL coding guidelines
7. Labs

1. LEC-1 review

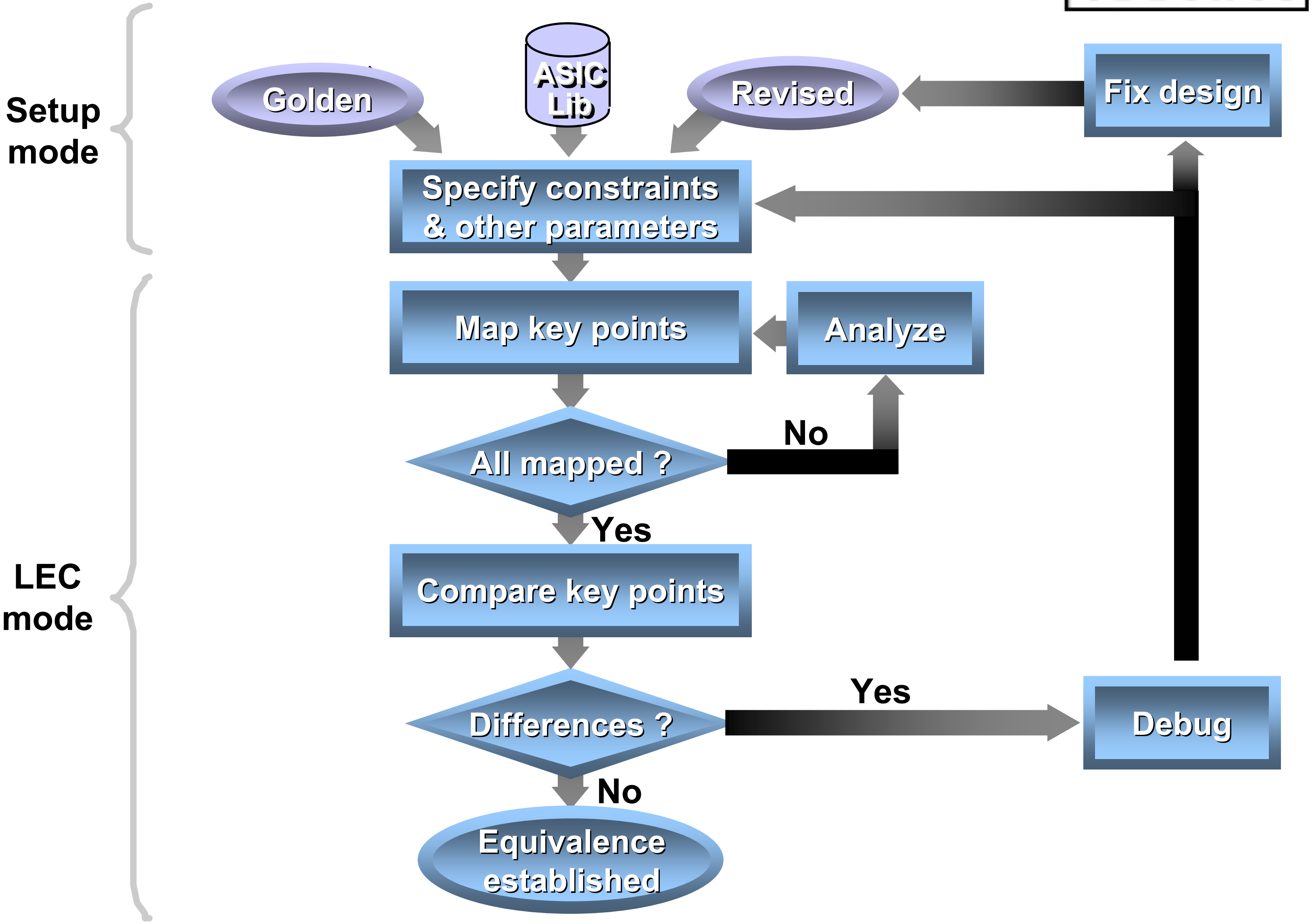
LEC-1 review

- LEC modes of operation
- LEC flow
- A typical LEC session (flat comparison)
- Hierarchical comparison session

LEC modes of operation

- SETUP mode
  - Specifying black boxes
  - Reading libraries and designs
  - Specifying design constraints
  - Specifying modeling directives
- LEC mode
  - Mapping process
    - Resolving unmapped key points
  - Compare process
    - Debugging non-equivalent key points

LEC flow


Setup mode
LEC mode
Golden
ASIC Lib
Revised
Fix design
Specify constraints & other parameters
Map key points
Analyze
All mapped ?
No
Yes
Compare key points
Differences ?
Yes
Debug
No
Equivalence established

A typical LEC session (flat compare)

SETUP

- Saving LEC transcript to a log file
- Specifying black boxes
- Reading libraries and designs
- Specifying design constraints
- Specifying modeling directives
- Switching to LEC mode

LEC

- Executing mapping process
  - Resolving unmapped key points
- Executing comparison process
  - Debugging non-equivalent key points
- Reporting run statistics

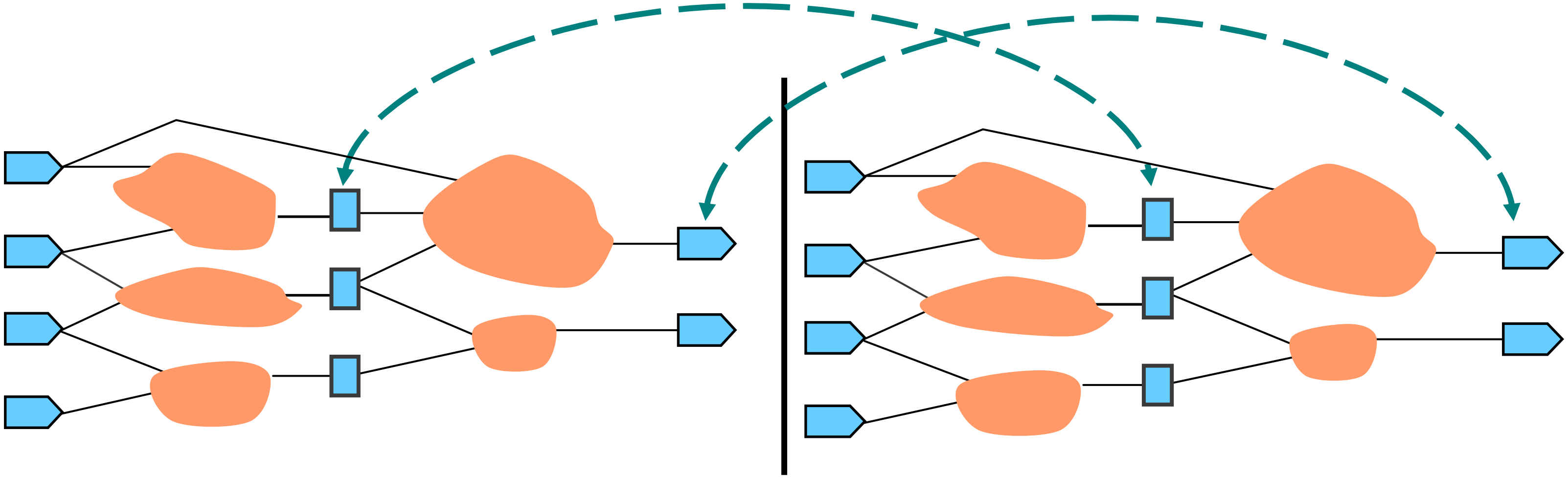
Hierarchical Comparison session

SETUP	❖ Saving LEC transcript to a log file ❖ Specifying black boxes ❖ Reading libraries and designs ❖ Specifying design constraints ❖ Specifying modeling directives ❖ Generating a hierarchical compare script ❖ Executing the hierarchical compare script
LEC	❖ Analyzing comparison result ❖ Reporting run statistics

2. Problem of Mapping key points

What is key point mapping?

Pairing corresponding key points: PI’s, PO’s, DFFs, D-latches, black boxes, Z gates, cut gates



🟦 Key points

🟧 Combinatorial logic

Mapping problem: Special cases

- Black boxes
- Module pins
- Phase inversion

Mapping black boxes

- Problem:
  - Module names are different (e.g, parameterized modules)
  - By default, LEC doesn’t map black boxes of different module names, even though instance names are the same
- Solutions:
  - Manual mapping

    ```
    Add mapping point <golden_bbox> <rev_bbox>
    ```
  - Turn off default mode:

    ```
    set mapping method -nobbox_name_match
    ```
  - Match module names with renaming rule

    ```
    add renaming rule <rule_id> <…> <…> -module
    ```

Mapping module pins

- Problem:

  Black box pins not mapped due to name differences

- To see how the pins are mapped:

```
report map point <black_box> -input -output
```

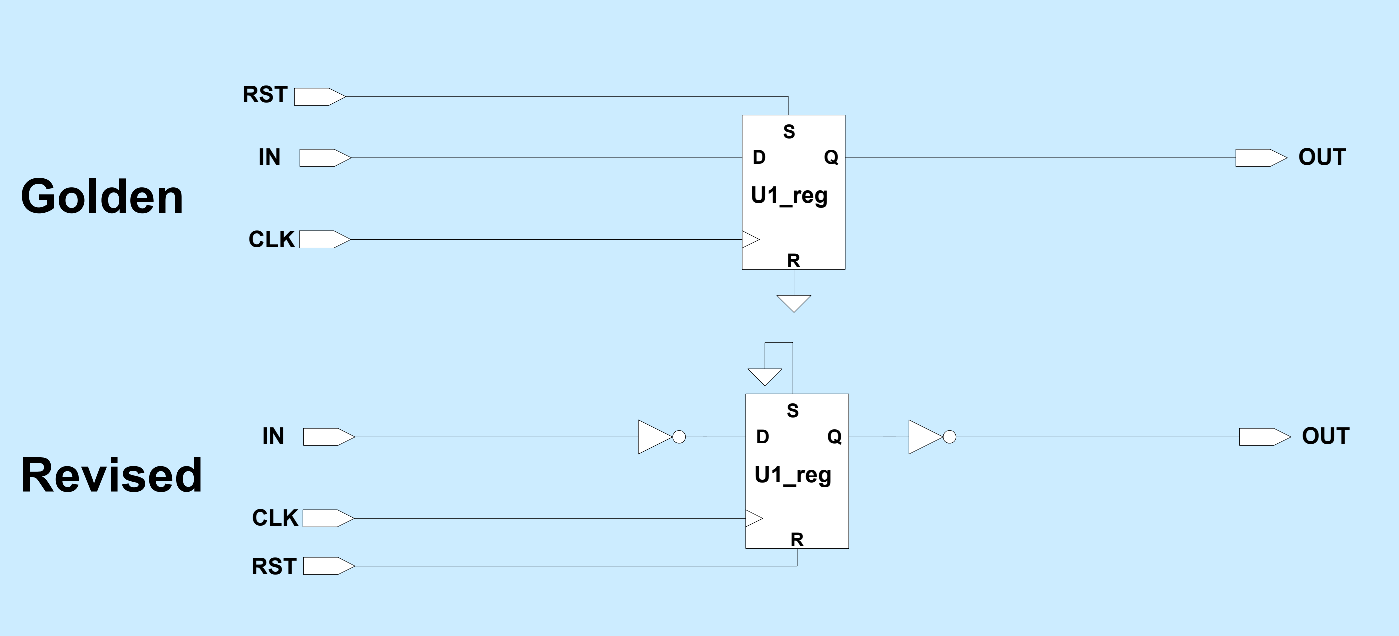
- Solution:

  Apply renaming rule to the pin

  - Syntax:
    ```
    add renaming rule name_name \
    <search_string> <replace_string> -pin -bbox \
    <module_name> [-golden|-revised]
    ```

Phase inverted mapping: Problem

- DFF/DLAT implemented with phase inverted library cells (inverted data, set/reset swapping)



- U1_reg must be phase-mapped

Phase inverted mapping: Solution

- Enable phase map method

```
read design cpu_rtl.v -verilog -golden
read design cpu_netlist.vg -verilog -revised
set mapping method -phase
set system mode lec
...
```

- Faster if phase map only applies on phase-inverted cells

```
read design cpu_rtl.v -verilog -golden
read design cpu_netlist.vg -verilog -revised
add mapping model <cell_name* …> -invert -revised

set mapping method -phase
set system mode lec

...
```

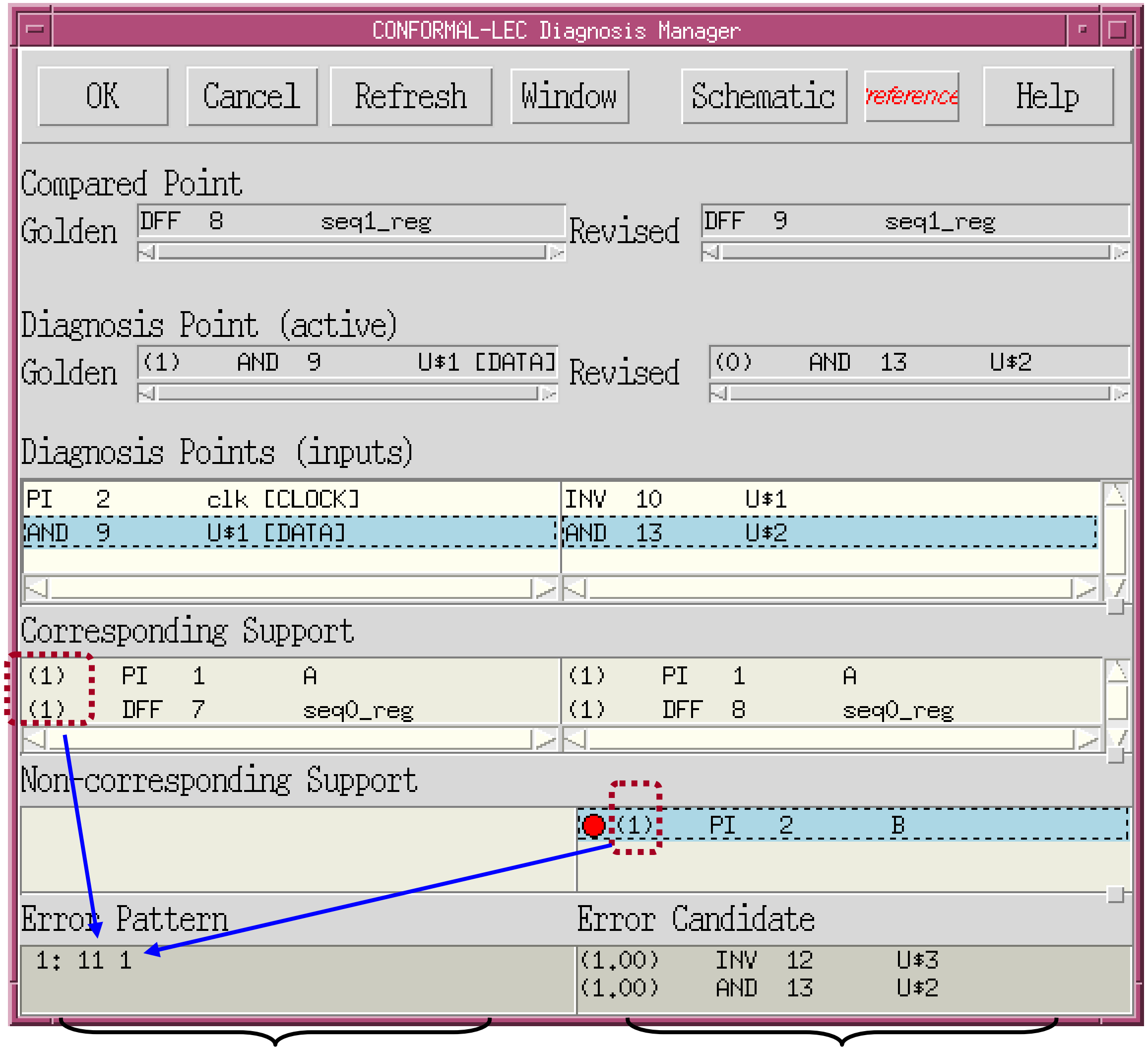
Note: `add mapping model` command is not documented

3. Debugging & Diagnosis

Debugging non-equivalent key points

- Diagnosis tips
  - Diagnose smaller cone first
    - Using the Mapping Manager to sort key points by supporting size
  - Concentrate on one logic cone at a time
- Diagnosis tools
  - Diagnosis Manager
  - Schematic Viewer
  - Source Code Manager
  - Gate Manager

Diagnosis Manager



Compared Point

Golden DFF 8 seq1_reg

Revised DFF 9 seq1_reg

Diagnosis Point (active)

Golden (1) AND 9 U$1 [DATA]

Revised (0) AND 13 U$2

Diagnosis Points (inputs)

```
PI   2        clk [CLOCK]       INV  10       U$1
AND  9        U$1 [DATA]        AND  13       U$2
```

Corresponding Support

```
(1)   PI   1        A           (1)   PI   1        A
(1)   DFF  7        seq0_reg    (1)   DFF  8        seq0_reg
```

Non-corresponding Support

```
● (1)   PI   2        B
```

Error Pattern

```
1: 11 1
```

Error Candidate

```
(1.00)   INV  12      U$3
(1.00)   AND  13      U$2
```

Compared Point →

Diagnosis Point (Active) →

Diagnosis Points

Corresponding Support

Non-corresponding Support

● Non-corresponding, and not mapped (red)

(M) Non-corresponding, but mapped (yellow with M)

Error Patterns

Error Candidates

How do I debug from the Diagnosis Manager?

- *Non-corresponding* support points hint at setup or mapping problems
  - Shows unconstrained signals (e.g., scan_en, undriven signals)
  - shows incomplete mapping, possible sequential optimization, merging, and replication
  - Invoke the Schematic Viewer to view logic cones
- Examine the simulation patterns in the *corresponding supports*
  - Look for common values in all reported test vectors
- Examine non-equivalence support points (red color code)
  - need phase mapping?
  - incorrect mapping?
- Invoke Schematic Viewer & Source Code Viewer to debug

Typical false non-equivalence: Setup problems

- Missing constraints
  - Need to disable scan-chain and JTAG test logic when verifying normal mode of operation
  - Tie off undriven signals
- Missing modeling directives
  - Modeling options are specific to synthesis tools and ASIC library vendors. User must turn on appropriate modeling directives
  - Example: Folding D-latch that models a DFF

```
set flatten model -latch_fold
```

More typical causes of false non-equivalence

- Unbalanced black-boxes
  - Use LEC commands :
    - `report design data`
    - `report black box`
- Port order inverted for black-boxes (undefined cell)
  - `set undef cell black -noascend`
- Not phase mapped
  - Phase map method is off by default for optimal runtime
  - Symptom shows non-equivalence on Data, Set, and Reset cones
  - Can try "`prove <golden_ID> <revised_ID> -invert`"

Debug non-equivalence: Hierarchical compare

- Isolate problem module
- Reduce cone size for easier debugging
- Refer to Hierarchical Comparison section for how to set up

4. Multipliers

Introduction

- Multipliers can take a long time to compare due to
  - Unmatched architectures
  - Operand swapping
- Solution:
  - Match multiplier architectures
  - Swap operands

DC synthesized multipliers

- DesignCompiler$^{\text{TM}}$ uses DesignWare$^{\text{TM}}$ components to implement arithmetic operators, comparators, etc.

  -
    ```
    assign z = a * b;
    ```

  -
    ```
    <mod>_DW02_mult U1 (.A(a), .B(b), .TC(1'b0),
    .PRODUCT(z));
    ```
- Three DW multiplier architectures
  - Carry Save Adder (CSA): DW Basic
  - Wallace Tree (WALL): DW Foundation required
  - Non-Booth Wallace Tree (NBW): DW Foundation required (new in DC 99.05)
- Report architectures from DC command

  -
    ```
    dc_shell>report_resources
    ```

Explicit control of DC implementation

```
module foo (in1, in2, out);
      input   [9:0] in1, in2;
      output [19:0] out;
      // synopsys dc_script_begin
      // set_implementation wall
      // synopsys dc_script_end
      assign out = in1 * in2;
endmodule
```

- LEC understands this embedded script and will create specified architecture
- LEC also understands analogous VHDL syntax

Instantiated multiplier or divider

- Multiplier or divider instantiated in RTL

  -
    ```
    DW02_mult #(A_width, B_width) U1 (.A(a),
                .B(b), .TC(1’b1), .PRODUCT(z);
    ```

  -
    ```
    DW02_divide #(A_width, B_width, TC_mode)U2
                (.A(a), .B(b), .TC(tc),
                 .DIVIDE_BY_0(div_by_0),
                 .QUOTIENT(result));
    ```
- LEC has a built-in representation for DW02_mult and DW02_divide
  - Do not read in the simulation models (for example: DW02_mult.v)

Long runtime due to multipliers?

- Check to see if there are any multipliers in the design
  - LEC> `report design data`

Transcript window

```
                  1058                 1058
```

- Matched architecture will be compared faster

Matching multiplier architecture

- Method 1: LEC commands
  - Automatic (recommended)
    - `set multiplier option`
  - Manual
    - `set multiplier implementation`
- Method 2: Verplex directive
  - Hard-coded in RTL

Matching multiplier architecture: Automatic

- LEC automatically
  - chooses best matching architecture
  - swaps the operand order if necessary
- Requirement:
  - specify command: `set multiplier option –auto`
  - version 4.0.1.a or later
  - multiplier module must have boundary
- Advantage:
  - User doesn’t need to know the revised architecture
  - Push-button solution
- Disadvantage:
  - Doesn’t work on multiplier with no module boundary

Matching multiplier architecture: Manual

- User manually specifies
  - Which type of architecture to set
  - Operand swapping
- Requirement:
  - Specify command:

```
set multiplier implementation <auto | CSA |
WALL | NBW | RCA | BKA>
```

    - `auto` – default, choose NBW if combined input width < 52, WALL otherwise
    - `CSA`, `WALL`, & `NBW` – correspond to DC architectures
    - `RCA` – Ripple Carry Adder
    - `BKA` – BuildGate™ architecture
  - Command must be executed before the "`read design`" command

Matching multiplier architecture: Manual (cont’d)

- Advantage
  - Doesn’t require module boundary for multiplier modules
- Disadvantage
  - User’s knowledge of the Revised architecture

Matching multiplier architecture: Verplex directive

```
module foo (a, b, c, d, out1, out2);
      input  [7:0] a, b, c, d;
      output [7:0] out1, out2;
      // verplex multiplier wall
      assign out1 = a * b;
      assign out2 = c * d;
endmodule
```

- Only the next statement is affected by directive
  - Implement `out1` with WALL architecture
- `out2` will use the default implementation

Operand swapping

- Multiplier structures are not symmetric
  - Comparing multipliers with different operand order can also take a long time
  - Synthesis tools might swap the operand order (usually happens in VHDL)
- `set multiplier implementation [-noswap | -swap | -autoswap]`
  - `noswap` - do not swap operands
  - `swap` - swap operands
  - `autoswap` - first operand is the largest
- Don’t have to specify swapping if “`set multiplier option –auto`” command is used
- May have to re-code RTL to control operand ordering

Isolating multiplier modules

- Multiplier logic is complex and hard to compare
- Isolating to compare multiplier modules and compare them separately can help the compare process
- Two ways to isolate multiplier modules
  - Flat compare: Black box multiplier modules and compare separately
  - Hierarchical compare: Please refer to the Hierarchical Compare section

5. Resolving abort key points

Introduction

- What are abort points?
- Investigate RTL causing abort
  - Don't Cares
  - Data Path
- What to do if compare aborts:
  - Increase compare effort (first resort)
  - Partition large logic cones
    - Hierarchical compare
    - Key point partition -automatic
    - Key point partition -manual

What are abort points?

- Key points that have neither been proven equivalent nor non-equivalent, based on the current compare effort algorithms
- Abort points can be attributed to large logic cone sizes with large numbers of support points
- Abort points can also be attributed to different structures with don’t care conditions (RTL-gate)

Investigating RTL don’t cares

- Please see RTL coding guideline section
- Don’t cares
  - Use LEC to report don’t cares in logic cone:

    ```
    report map point <abort_point> -property
    ```

    Note: `-property` option is not documented
  - full case/parallel case/x-assignments are common source of don’t cares
  - re-coding RTL to get rid of don’t cares might help

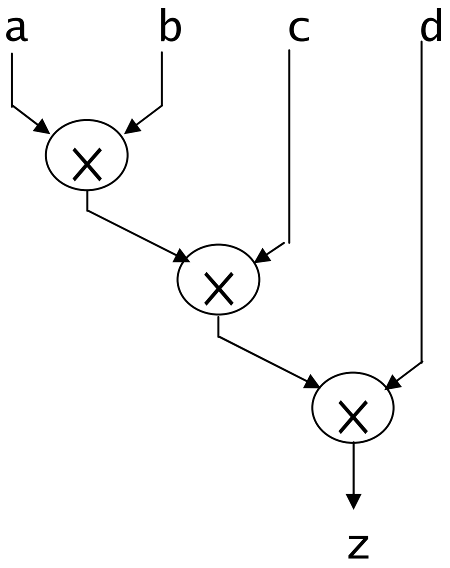
Investigating RTL data path

- Operator ordering
- Multiplier - refer to the Multiplier section

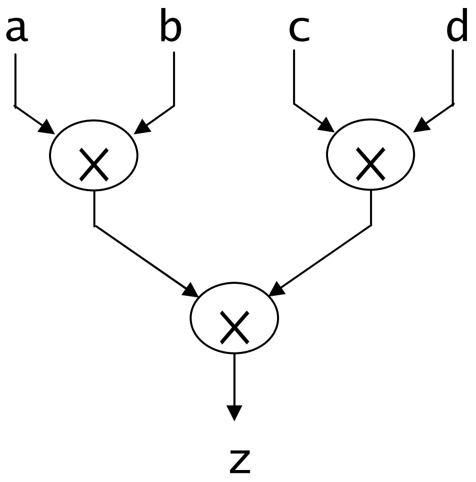
Operator ordering

- LEC operator tree implementation

```
z = a * b * c * d;
```



```
z = (a * b) * (c * d);
```

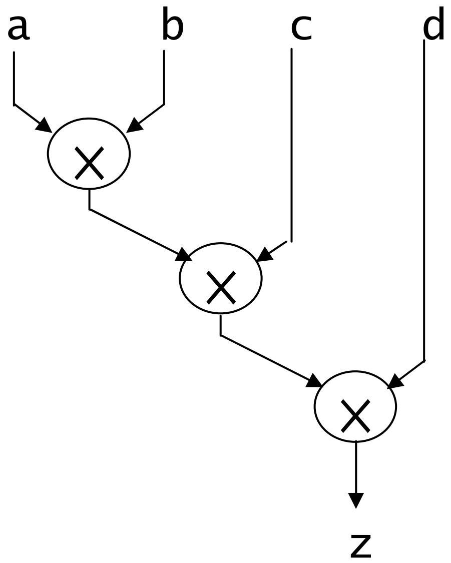


Operator ordering

- Different logical structures can exacerbate situation

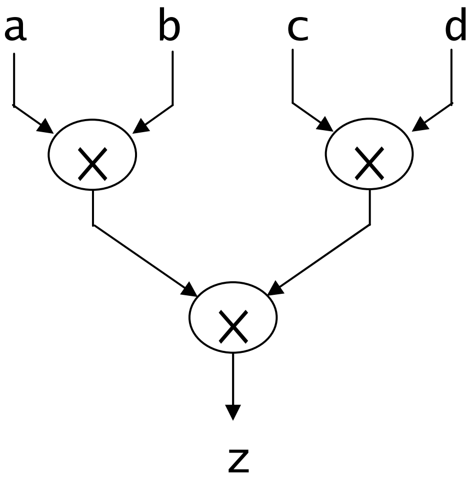
RTL ordering

`z = a * b * c * d;`



Netlist ordering

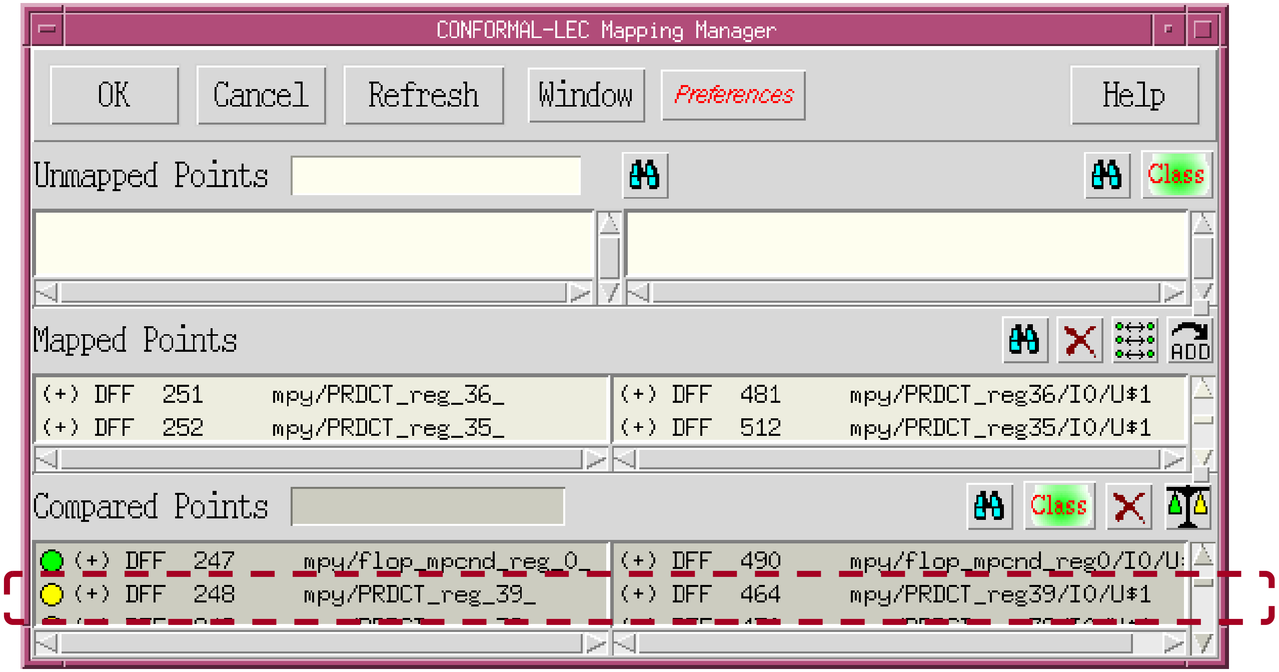
`(a * b) * (c * d);`



- Changing RTL to match the netlist ordering can make comparison faster

Identifying abort points in LEC

- GUI: yellow-filled circle on Mapping Manager



```
(+) DFF   251      mpy/PRDCT_reg_36_        (+) DFF   481      mpy/PRDCT_reg36/IO/U$1
(+) DFF   252      mpy/PRDCT_reg_35_        (+) DFF   512      mpy/PRDCT_reg35/IO/U$1
```

```
● (+) DFF   247      mpy/flop_mpcnd_reg_0_    (+) DFF   490      mpy/flop_mpcnd_reg0/IO/U:
● (+) DFF   248      mpy/PRDCT_reg_39_        (+) DFF   464      mpy/PRDCT_reg39/IO/U$1
```

Abort key points are shown with yellow-filled circle

- Non-GUI: LEC> `rep compare data -class abort`

Resolving abort points: Increase compare effort

- Default compare effort: low
- Can be changed to high or super
- Should be used after low-effort run is completed *(to avoid "expensive" methods used on easy compare points)*

```
set log file logfile.$LEC_VERSION -replace
read design cpu_rtl.v -verilog -golden
read design -f verilog.vc -verilog -revised
set system mode lec
add compare points -all

compare
//”set compare effort auto” will automatically switch to super compare
                           effort to aborted key points
report compare data -abort
set compare effort high
compare
...
```

Resolving abort points: Partition large logic cone

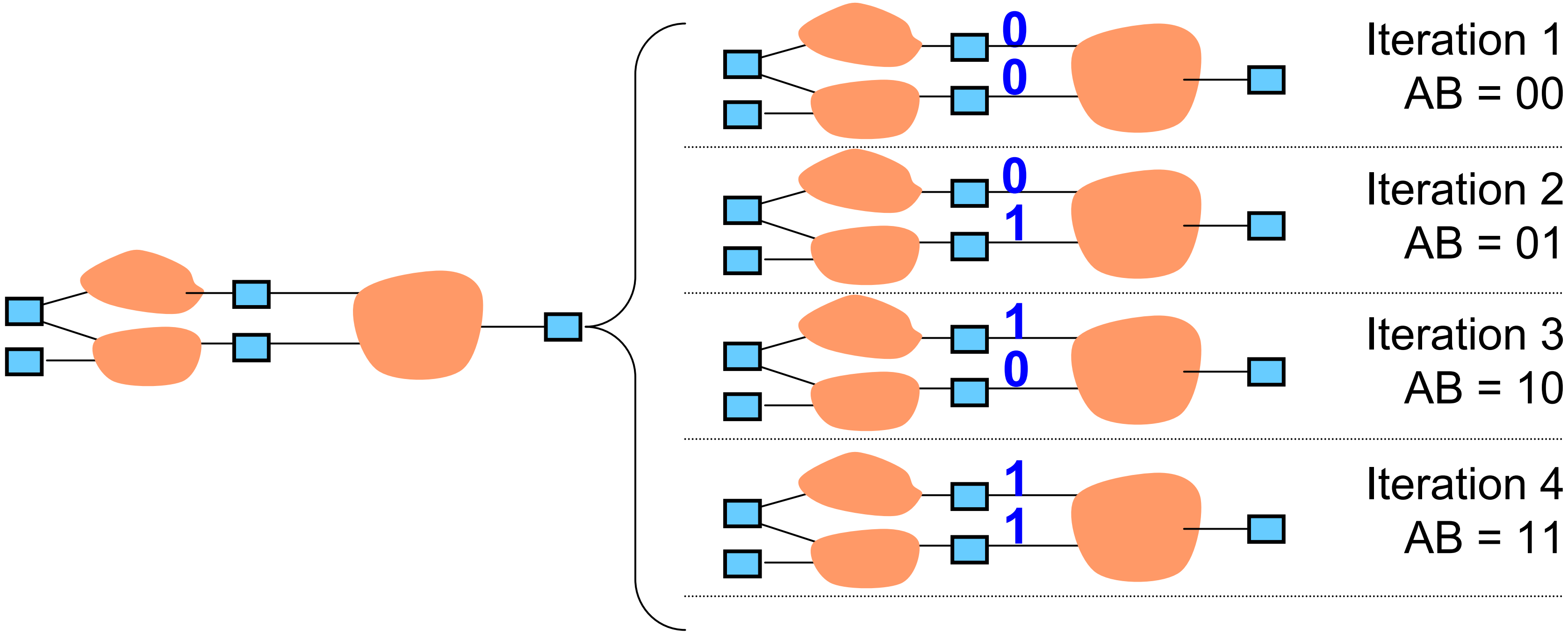
- Hierarchical compare
- Key point partition - Automatic
- Key point partition - Manual

Hierarchical compare

- Narrows down abort modules
- Reduces logic cone size
- Please refer to the Hierarchical Compare section

Key point partitioning

- Divide-and-conquer approach to verifying large logic cones



- Key points are selected by LEC or users
- LEC automatically generates a dofile that executes the above iterations

Key point partitioning: Automatic

- LEC can select partition key points automatically
  - `set compare option -partition <flow | key>`
    - `flow` allows LEC to select points inside the cloud of logic
    - `key` will limit partition points to inputs to logic cone
    - **Note**: This command is *not* documented

```
read design cpu_rtl.v -verilog -golden
read design -f verilog.vc -verilog -revised
set system mode lec
add compare points -all
compare
report compare data -abort

set compare effort high
compare
report compare data -abort > aborts.high
set compare option -partition key    //This command is not documented
compare
...
```

Key point partitioning: Manual

- When should I *manually* partition the key points?
  - When automatic selection does not work to your satisfaction
- Criteria in selecting key points:
  - Support points of aborted cones
  - Key points that are already proven equivalent

```
read design cpu_rtl.v -verilog -golden
read design -f verilog.vc -verilog -revised
set system mode lec
add compare points -all
compare
report compare data -abort

set system mode setup
add partition key_point -pin a b
write partition dofile partition.dofile -replace
dofile partition.dofile

...
```

Contents of partition.dofile

```
// Iteration 1
//
delete pin constraint a -golden
delete pin constraint a -revised
delete pin constraint b -golden
delete pin constraint b -revised
add pin constraint 0 a -golden
add pin constraint 0 a -revised
add pin constraint 0 b -golden
add pin constraint 0 b -revised
set system mode lec
add compare points -all
compare
set system mode setup
//
// Iteration 2
//
delete pin constraint a -golden
delete pin constraint a -revised
delete pin constraint b -golden
delete pin constraint b -revised
add pin constraint 0 a -golden
add pin constraint 0 a -revised
add pin constraint 1 b -golden
add pin constraint 1 b -revised
set system mode lec
add compare points -all
compare
set system mode setup
```

```
// Iteration 3
//
delete pin constraint a -golden
delete pin constraint a -revised
delete pin constraint b -golden
delete pin constraint b -revised
add pin constraint 1 a -golden
add pin constraint 1 a -revised
add pin constraint 0 b -golden
add pin constraint 0 b -revised
set system mode lec
add compare points -all
compare
set system mode setup
//
// Iteration 4
//
delete pin constraint a -golden
delete pin constraint a -revised
delete pin constraint b -golden
delete pin constraint b -revised
add pin constraint 1 a -golden
add pin constraint 1 a -revised
add pin constraint 1 b -golden
add pin constraint 1 b -revised
set system mode lec
add compare points -all
compare
```

6. RTL coding guidelines to optimize equivalence checking

Introduction

- The following RTL coding guidelines can be used to improve the performance of the equivalence checking process
  - Keep difficult arithmetic blocks separate
  - Assign known values to all cases in case statement
  - Use more assign statements
  - Avoid combinatorial feedback loops
- Following the guidelines helps reduce problems described in sections:
  - Multipliers
  - Comparison: Debugging non-equivalent compare points
  - Resolving abort key points

Keep difficult arithmetic blocks separate

- Arithmetic blocks such as multipliers tend to produce longer runtime, more difficult mapping, abort points (more than random logic blocks)

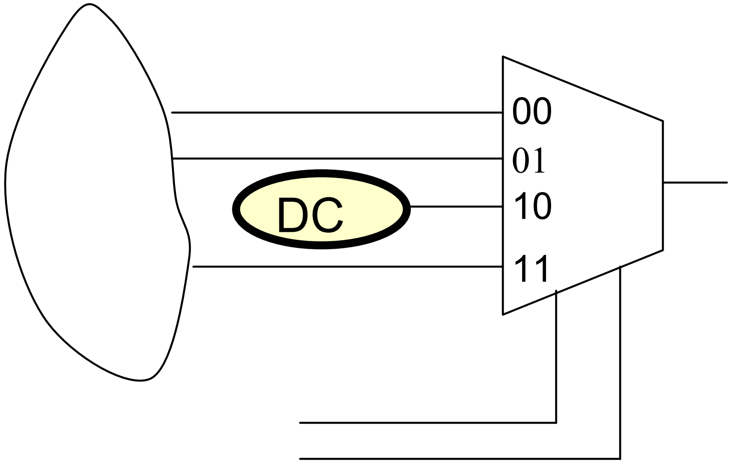
- Complexity dramatically increases when these multipliers are buried within a flattened design

- When kept in their own hierarchy, verification stands a much better chance of completion through module-based (hierarchical) comparison mode

Assign known value to all cases in `case` statement: Problem

- Applying synthesis full_case directive or X assignments on incompletely specified case statements creates "don’t care" (DC) conditions in LEC
- LEC creates DC circuitry to ignore comparison of unspecified states
- DC circuitry grows with greater numbers of unspecified states
- Bigger DC circuitry leads to longer LEC runtime

```
Module xyz (sm, out);
   input [1:0] sm;
   output out;
   reg out;
   always @(sm)
     case (sm) //synopsys full_case
         2’b00: out = 2’b01;
         2’b01: out = 2’b10;
         2’b11: out = 2’b00;
     endcase
endmodule
```



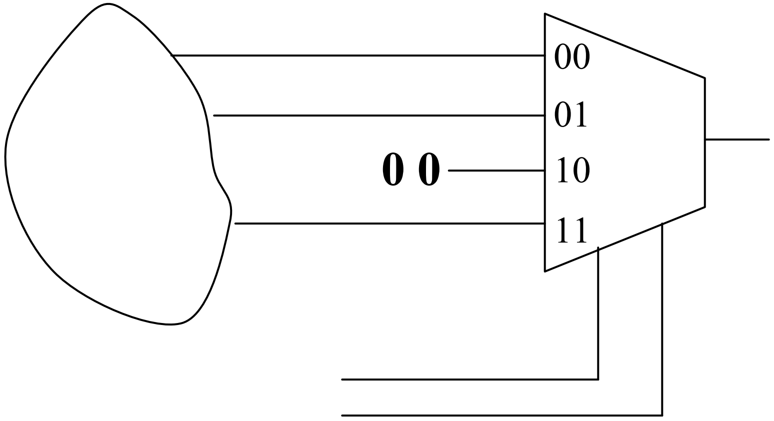
Assign known value to all cases in `case` statement: Solution

- Use "default" with known value to catch all unspecified conditions

- Using default with known value does not necessarily impact synthesis performance in area/speed optimization

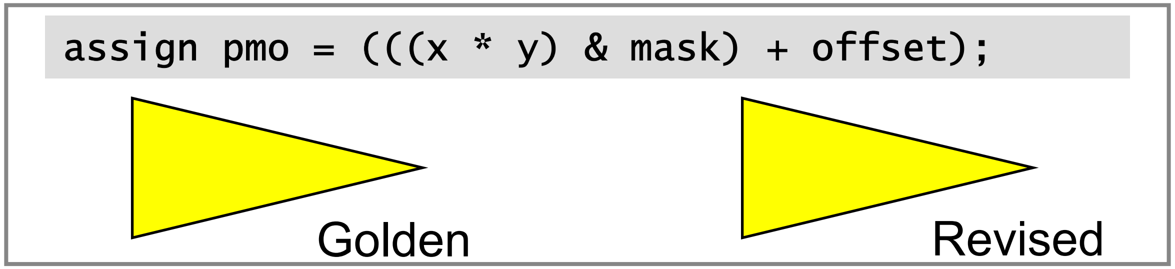
__Note__: Refer to an Esnug paper on "full_case parallel_case", the Evil Twins of Verilog Synthesis by Cliff Cummings, Sunburst Design, Inc.

```
Module xyz (sm, out);
   input [1:0] sm;
   output out;
   reg out;
   always @(sm)
     case (sm)
         2’b00: out = 2’b01;
         2’b01: out = 2’b10;
         2’b11: out = 2’b00;
         default = 2’b00;
     endcase
endmodule
```



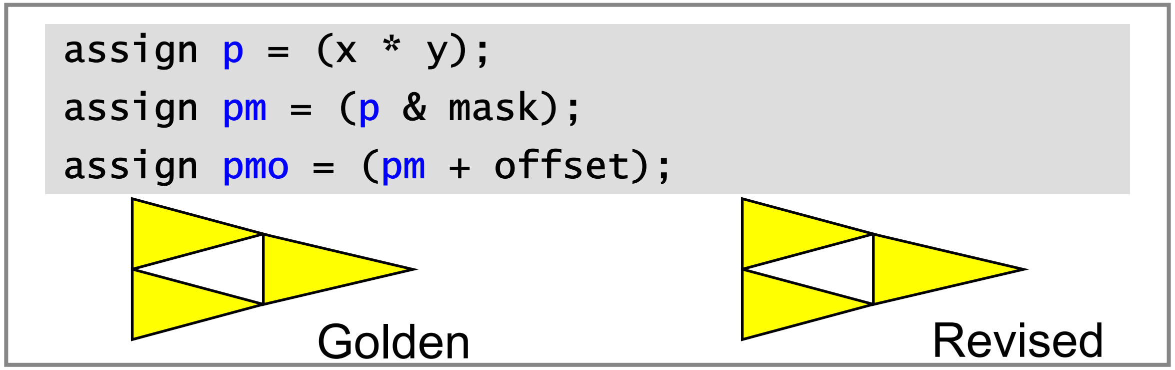
Using more `assign` statements

- Break down the size of logic cones
- Improve compare performance with smaller intermediate cones
- Easier to diagnose with more intermediate similarity points



```
assign pmo = (((x * y) & mask) + offset);
```

Before



```
assign p = (x * y);
assign pm = (p & mask);
assign pmo = (pm + offset);
```

After

Avoid combinatorial feedback

- Combination feedback loops can hinder verification process
- LEC must insert cut gate to break the loops
- Hard to match point of cut in the Golden and the Revised loops
- For difficult cases, user has to specify where to insert cut gate

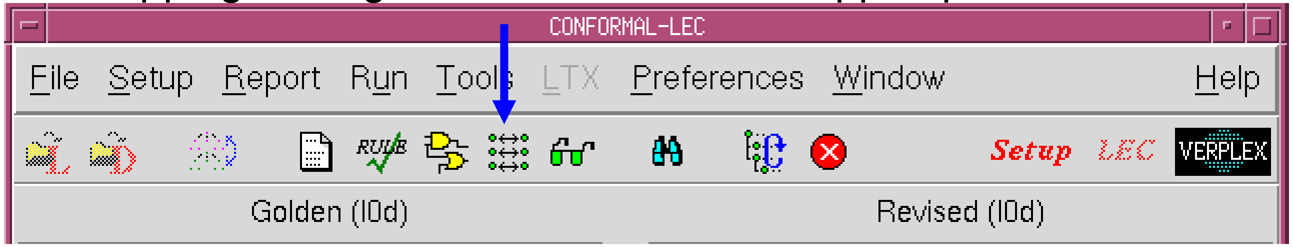
7. Labs

Lab 1

- **OBJECTIVE**: Add renaming rule(s) to complete mapping key points

- 1. Unix % `cd ~/lec2_labs/mapping/key_point_mapping`, start LEC.
- 2. Execute the dofile *init.do.*
- 4. Open the Mapping Manager to examine the unmapped points.

Main GUI window

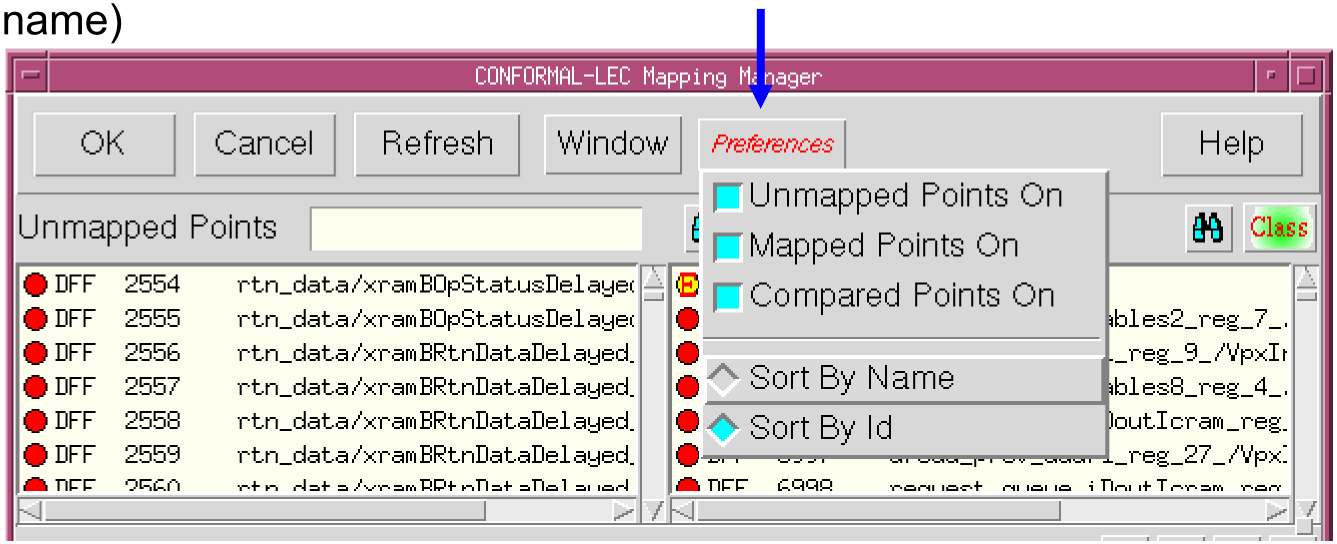


-
-

Lab 1 (cont'd)

- Sort key points by name to help determine the renaming rules (Preference → Sort by name)

Mapping Manager



```
DFF  2554  rtn_data/xramBOpStatusDelaye
DFF  2555  rtn_data/xramBOpStatusDelaye
DFF  2556  rtn_data/xramBRtnDataDelayed
DFF  2557  rtn_data/xramBRtnDataDelayed
DFF  2558  rtn_data/xramBRtnDataDelayed
DFF  2559  rtn_data/xramBRtnDataDelayed
DFF  2560  rtn_data/xramBRtnDataDelayed
```

```
ables2_reg_7_
_reg_9_/VpxI
ables8_reg_4_
DoutIcram_reg
_reg_27_/Vpx
DFF  6998  request_queue_iDoutIcram_reg
```

- 5. Create renaming rules to resolve unmapped key points. After you specify rules, continue mapping with the command `map key point`
- 6. Make sure all key points are mapped
- 7. Exit LEC
- Note: Solution is at the end of this section and current working directory

Lab 2

➢ **OBJECTIVE**: Mapping black box pins

- ➢ 1. Unix % `cd ~/lec2_labs/mapping/pin_mapping`, start LEC.
- ❖ 2. Execute the dofile *init.do.*
- ❖ 4. Specify `rep compare data -class nonequivalent` and check the report to see that the black box `bridge/pci_target_unit/fifos/pcir_ram` is non-equivalent due to non-corresponding inputs (input pins not being mapped).
- ❖ 5. Report black box pin mapping with the command `rep mapped point <black_box> -input -output`. Input pins that have no correspondence need to be mapped
- ❖ 6. Delete the mapped black box with the command `delete mapped point <black_box>`. Pins are only mapped while the black box is being mapped, so we need to re-map the black box so that the unmapped pins can be mapped.
- ❖ 7. Specify renaming rules to map the pins that weren’t mapped, and then re-map the black box with the command `map key point`.

Lab 2 (cont'd)

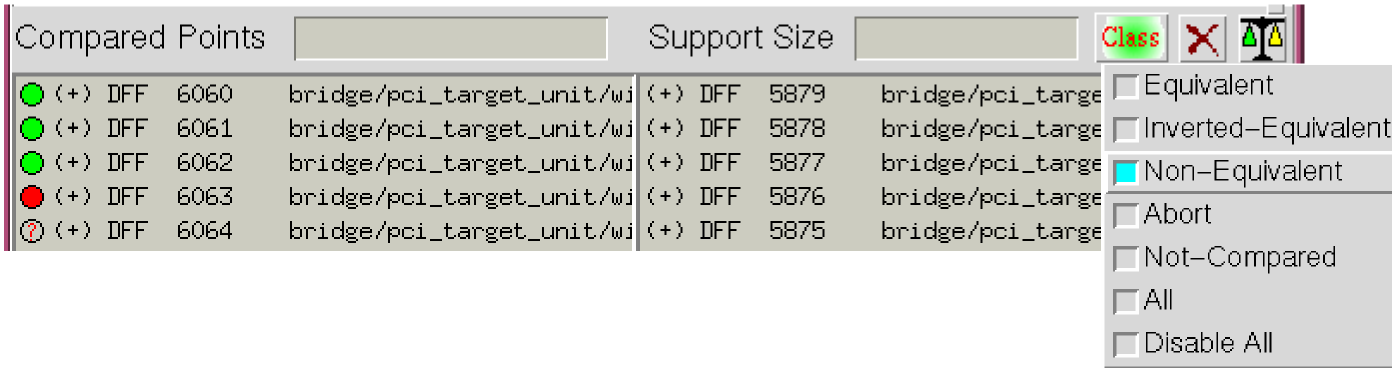
- 8. Specify `rep mapped point <black_box> -input -output` again to verify that the pins indeed mapped (paired) after you specified renaming rules.
- 9. Compare to verify the problem is fixed:
- a) LEC> `add compare point -all`
- b) LEC> `compare`
- c) LEC> `report compare data -class nonequivalent`
- 10. Exit LEC.
- Note: Solution is at the end of this section and current working directory.

Lab 3

- **OBJECTIVE:** Debug non-equivalent key points through the Diagnosis
- Manager

- 1. Unix % `cd ~/lec2_labs/debug_neq/neq_mapping`, start LEC.
- 2. Execute the dofile *init.do.*
- *3.* Invoke the Mapping Manager.
- 4. On the Mapping Manager, use "Class" pull-down menu to display only the non-equivalent key points.

Mapping Manager

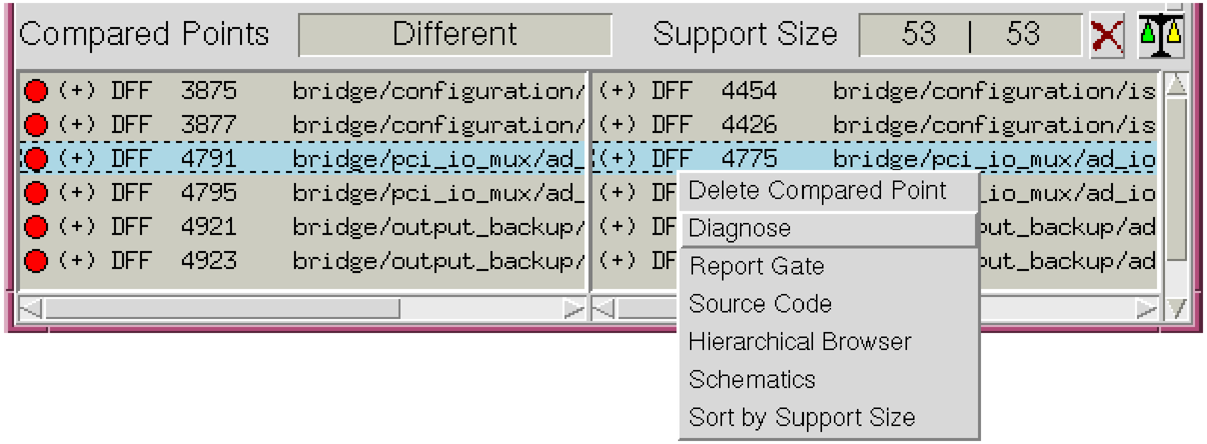


```
Compared Points                              Support Size
● (+) DFF  6060  bridge/pci_target_unit/wi | (+) DFF  5879  bridge/pci_targe
● (+) DFF  6061  bridge/pci_target_unit/wi | (+) DFF  5878  bridge/pci_targe
● (+) DFF  6062  bridge/pci_target_unit/wi | (+) DFF  5877  bridge/pci_targe
● (+) DFF  6063  bridge/pci_target_unit/wi | (+) DFF  5876  bridge/pci_targe
? (+) DFF  6064  bridge/pci_target_unit/wi | (+) DFF  5875  bridge/pci_targe
```

Lab 3 (cont'd)

- 5. Invoke the Diagnosis Manager for a non-equivalent key point and try to figure out what the problem is. (Hint: Non-corresponding points . . . possible mapping problem?)

Mapping Manager



Compared Points | Different | Support Size | 53 | 53

```
● (+) DFF  3875   bridge/configuration/  (+) DFF  4454   bridge/configuration/is
● (+) DFF  3877   bridge/configuration/  (+) DFF  4426   bridge/configuration/is
● (+) DFF  4791   bridge/pci_io_mux/ad_  (+) DFF  4775   bridge/pci_io_mux/ad_io
● (+) DFF  4795   bridge/pci_io_mux/ad_  (+) DF                  _io_mux/ad_io
● (+) DFF  4921   bridge/output_backup/  (+) DF                  put_backup/ad
● (+) DFF  4923   bridge/output_backup/  (+) DF                  put_backup/ad
```

- 6. Fix the problem:
- a) Delete mapped points with the command `delete mapped points`.
- b) Specify renaming rules.
- c) Run the compare process again. Verify that the design is 100% equivalent.
- 7. Exit LEC.
- Note: Solution is at the end of this section and current working directory.

Lab 4

➢ **OBJECTIVE**: Debug non-equivalent key points

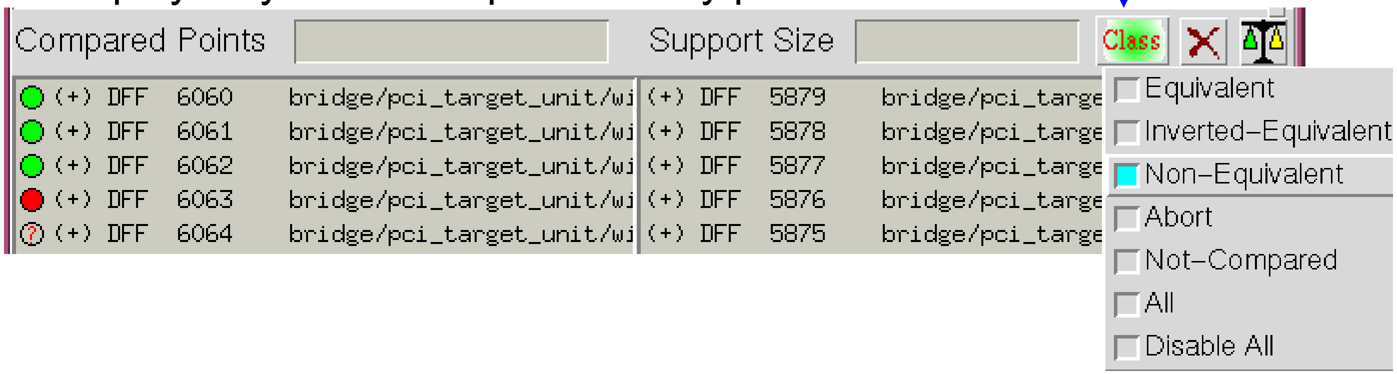
➢ 1. Unix % `cd ~/lec2_labs/debug_neq/cone_partition`, start LEC.

❖ 2. Execute the dofile *init.do.*

❖ 3. Invoke the Mapping Manager.

❖ 4. On the Mapping Manager, use "Class" pull-down menu on the Mapping Manager to display only the non-equivalent key points.

Mapping Manager



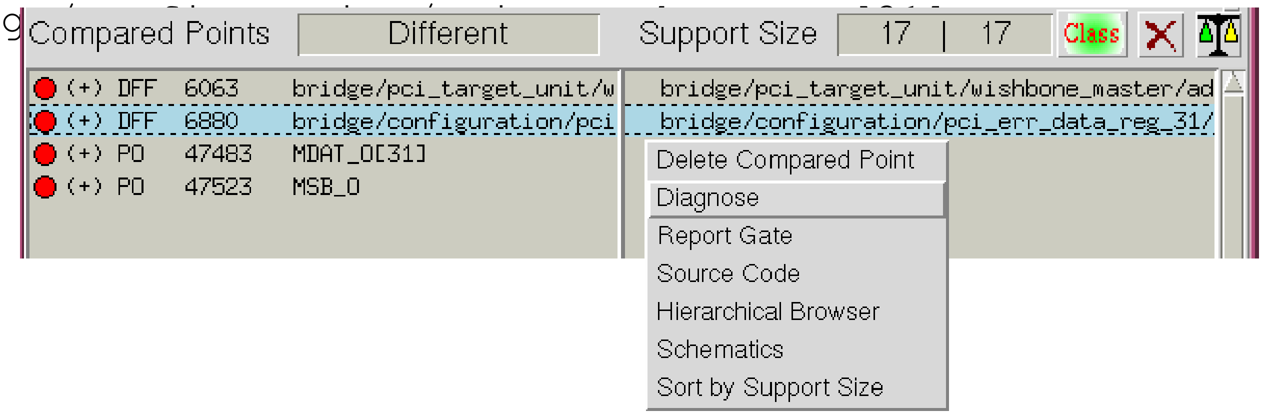
```
Compared Points [          ]          Support Size [          ]  Class
● (+) DFF  6060  bridge/pci_target_unit/wi  (+) DFF  5879  bridge/pci_targe
● (+) DFF  6061  bridge/pci_target_unit/wi  (+) DFF  5878  bridge/pci_targe
● (+) DFF  6062  bridge/pci_target_unit/wi  (+) DFF  5877  bridge/pci_targe
● (+) DFF  6063  bridge/pci_target_unit/wi  (+) DFF  5876  bridge/pci_targe
? (+) DFF  6064  bridge/pci_target_unit/wi  (+) DFF  5875  bridge/pci_targe
```

□ Equivalent
□ Inverted–Equivalent
☑ Non–Equivalent
□ Abort
□ Not–Compared
□ All
□ Disable All

Lab 4 (cont'd)

- 5. a) Invoke the Diagnosis Manager to diagnose the non-equivalent key point:
- bridg

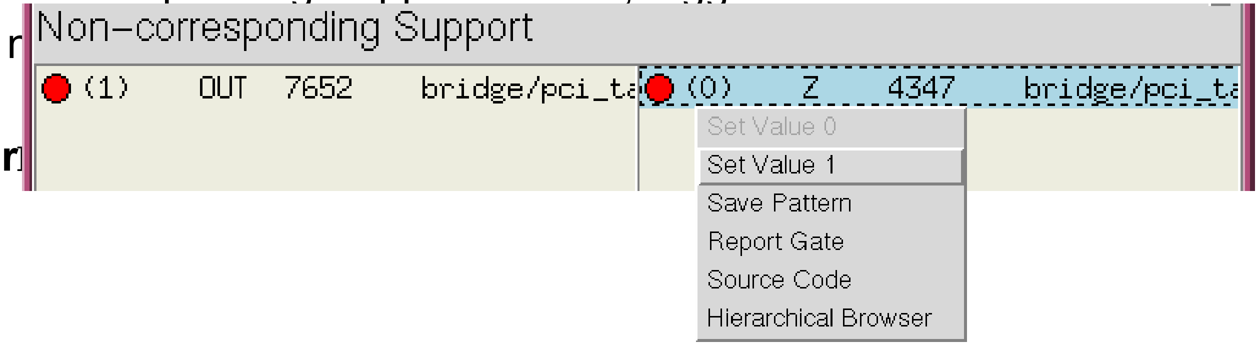
Mapping Manager



```
Compared Points   Different   Support Size   17 | 17
(+) DFF   6063    bridge/pci_target_unit/w      bridge/pci_target_unit/wishbone_master/ad
(+) DFF   6880    bridge/configuration/pci      bridge/configuration/pci_err_data_reg_31/
(+) PO    47483   MDAT_O[31]
(+) PO    47523   MSB_O
```

- b) In the Non-corresponding Support section, toggle the simulation value for
- each r
- pin

Diagnosis Manager



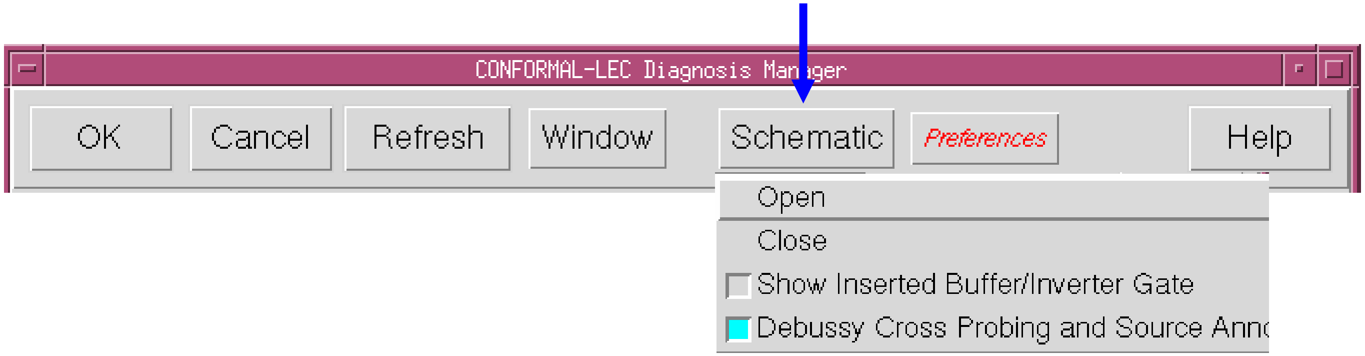
```
Non-corresponding Support
(1)   OUT   7652   bridge/pci_ta   (0)   Z   4347   bridge/pci_ta
```

- Look for any simulation value changes at the
- Diagnosis Point (active) after toggling.

Lab 4 (cont'd)

- 6. Invoke logic cones schematic from the Diagnosis Manager.

Diagnosis Manager


CONFORMAL-LEC Diagnosis Manager
OK
Cancel
Refresh
Window
Schematic
Preferences
Help
Open
Close
□ Show Inserted Buffer/Inverter Gate
■ Debussy Cross Probing and Source Ann

- 7. Starting from the compare point (dark blue DFF), trace the fan-in logic to find the source of the logic difference. LEC (3.4.0.a or later) trims the similar logic for easier debugging, but it might be hard to see the logic connection at first. To have LEC open more gates to get a better picture of the circuit, click to highlight the compare point then right click → Fan-in Cone → Close → Right click again → Fan-in Cone → Open to EQ/Key Points. Do this for both the Golden and the Revised schematics.

Lab 4 (cont'd)

- 8. Exit LEC when done debugging.
- Note: Solution is at the end of this section and current working directory

Lab 5

- **OBJECTIVE**: Map modules with renaming rules

- 1. Start LEC: Unix% `cd ~/lec2_labs/hier_compare/mod_map_renaming`
- 2. Read *golden.v* for the Golden design.
- 3. Read the Revised library *lib.v*.
- 4. Read *revised.v* as the Revised design.
- 5. Generate hierarchical compare script.
- 6. Use `add renaming rule` to map all modules (5 modules).
- 7. Regenerate the hierarchical compare script.
- Verify that the generated script includes 5 modules for hierarchical compare.

- Note: The solution is at the end of this section and current working directory.

Lab 6

- **OBJECTIVE**: Map modules with `uniquify` command

- 1. Start LEC :
- Unix% `cd ~/lec2_labs/hier_compare/mod_map_uniquify`
- 2. Read *golden.v* for the Golden design.
- 3. Read the Revised library *lib.vpx.v*.
- 4. Read *revised.v* as the Revised design
- 5. Generate hierarchical compare script.
- 6. Instead of using `add renaming rule,` use `uniquify` command to fix mapping problem for the modules being instantiated more than once (7 modules).
- 7. Re-generate hierarchical compare script.
- Verify that the generated script includes 7 modules for hierarchical compare.
-
- Note: The solution is at the end of this section and current working directory.

Lab 7

- **OBJECTIVE**: Debug non-equivalent key points using
- hierarchical compare technique

- 1. Unix % `cd ~/lec2_labs/debug_neq/debug_via_hier`, start LEC.
- 2. Execute the dofile *init.do.*
- 3. Try debugging to pinpoint the error gate.
- 4. If you haven't pinpointed the error gate, try narrowing down the non-equivalent module using a hierarchical compare.
- a) Switch to SETUP mode.
- b) `set gui off`
- c) Generate a hierarchical compare script, and then execute the script. Note that one module will be reported non-equivalent.
- 5. Set root module to the non-equivalent module, and then start comparing this module flat.

Lab 7 (cont'd)

- Sort the non-equivalent key points by size with the command:
- `diagnose -summary -sort size`.
- From the report, find the key points with smaller cone size to diagnose. Note these key points by their ID number, and then invoke the Mapping Manager to start the diagnosis process. From the schematic window, you should be able to see that the Revised design has an extra inverter.
- 7. Exit LEC.
- Note: Solution is at the end of this section and current working directory.

Lab solutions

Lab1: Add renaming rules to complete mapping key points

```
add notranslate module regfile_9x30 regfile_9x64 -lib
read library lib/baselib.v lib/lib.v lib/regfile_9x30.v lib/regfile_9x64.v
read design golden.v
read design revised.v -rev
set undriven signal 0
add pin constraint 0 scan_en -rev
set mapping method -name only
set system mode lec
add renaming rule R1 \/ _ -gold
add renaming rule R2 \/ _ -rev
map key point
```

Lab solutions

Lab 2: Mapping black box pins

```
add notranslate module DP_SRAM*
add search path ./rtl/ -gold
read library lib/*.v -gold
read design -f rtl/filelist -root PCI -gold
read library lib/lib.vpx.v -rev
read design gate/PCI.gv -root PCI -rev
set flatten model -nodff_to_dlat_feedback
set undriven signal 0
add renaming rule R1 reg_%d reg[@1] -rev //this rule speeds up the mapping process
add renaming rule R2 in_I in -pin -bbox DP_SRAM_ADDR_LENGTH5_SIZE32_1 -rev
add renaming rule R3 inA in -pin -bbox DP_SRAM_ADDR_LENGTH5_SIZE32_1 -rev
set mapping method -nobbox_name_match
set sys mode lec
add compare point -all
compare
```

Lab solutions

Lab 3: Debug non-equivalent key points through the Diagnosis Manager

```
add notranslate module DP_SRAM*
add search path ./rtl/ -gold
read library lib/*.v -gold
read design -f rtl/filelist -root PCI -gold
read library lib/lib.vpx.v -rev
read design gate/PCI.gv -root PCI -rev
set flatten model -nodff_to_dlat_feedback
add renaming rule R1 reg_%d reg[@1] -rev //this rule speeds up the mapping process
add renaming rule R2 isr_bit3_0_reg%d_%d isr_bit3_0_reg[@2] -rev
set mapping method -nobbox_name_match
set sys mode lec
add compare point -all
compare
```

Lab solutions

Lab 4: Debug non-equivalent key points

```
The output data_out[31]the BBOX bridge/pci_target_unit_fifos/pciw_ram is disconneted
on the revised design.
```

Lab solutions

Lab5: Mapping modules with renaming rules

```
read design golden.v
read library lib.v -rev
read design revised.v -rev
add renaming rule R1 %s_scan$ @1 -module -rev
add renaming rule R2 %s_%d$ @1 -module -rev
write hier dofile hier.do -noexact -constraint -replace
//dofile hier.do
```

Lab 6: Mapping modules with uniquify command

```
read design golden.v
read library lib.vpx.v -rev
read design revised.v -rev
uniquify lrp -gold
uniquify burp -gold
write hier dofile hier.do -noexact -constraint -replace
//dofile hier.do
```

Lab solutions

Lab 7: Debug non-equivalent key points using hierarchical compare technique

```
add notranslate module DP_SRAM*
add search path ./rtl/ -gold
read library lib/*.v -gold
read design -f rtl/filelist -root PCI -gold
read library lib/lib.vpx.v -rev
read design gate/PCI.gv -root PCI -rev
set flatten model -nodff_to_dlat_feedback
add renaming rule R1 reg_%d reg[@1] -rev //this rule speeds up the mapping process
set mapping method -nobbox_name_match
write hier dofile hier.do -noexact -constraint -replace
dofile hier.do
set root module PCI_TARGET32_SM -both  //non-equivalent module needs to be diagnosed
set sys mode lec
add compare point -all
compare
diagnose -summary -sort size

//diagnose the NEQ key point bc0_out will show very obviousely that there is
//an extra inverter in the revised logic cone, which causes NEQ
```