cadence®

Logic Equivalence Checking with Conformal

Version 5.0

December 19, 2004

Blank Page

Using Conformal for Logic Equivalence Check

December 19, 2004

Starting and Exiting Conformal LEC

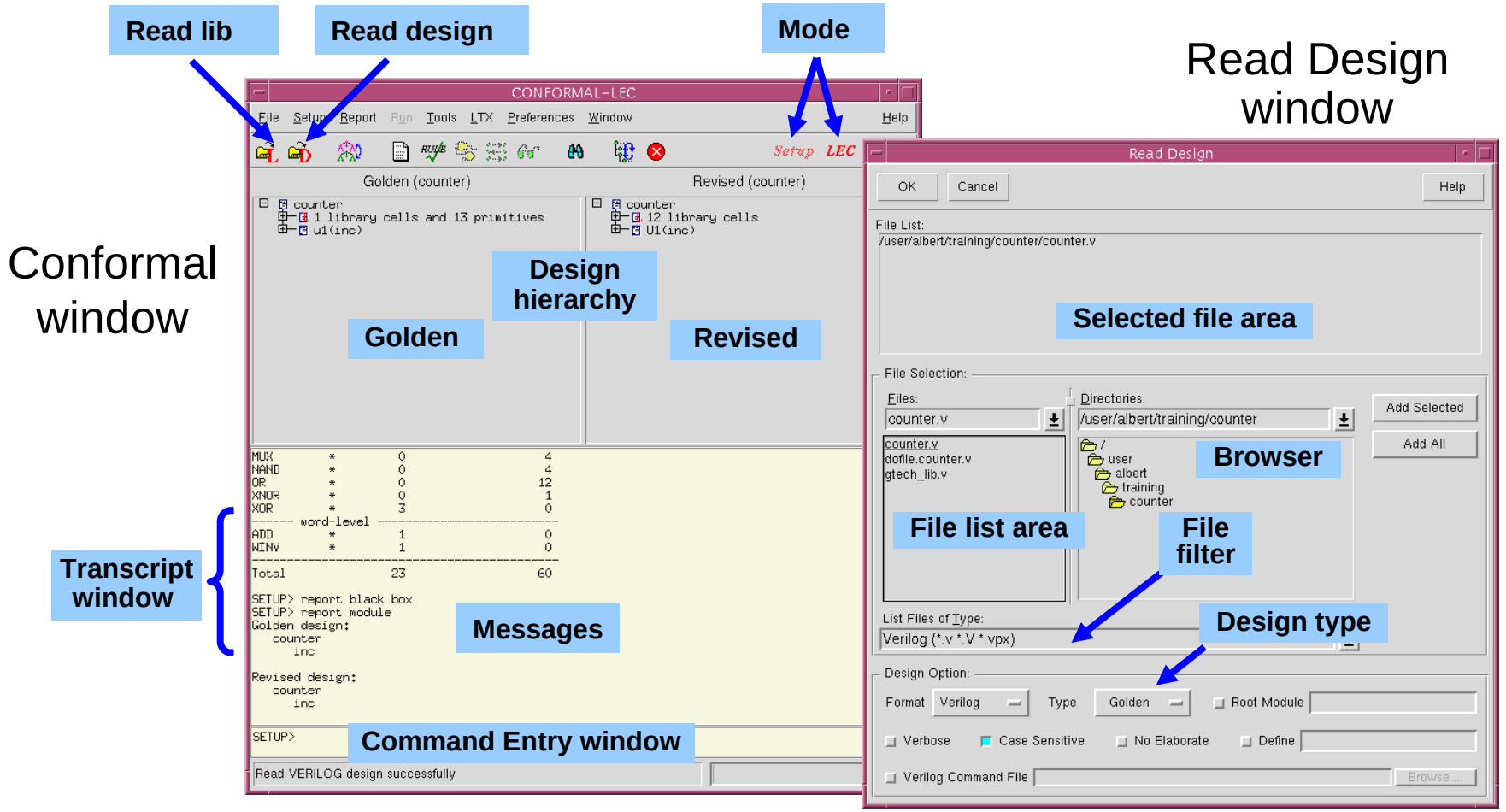
Starting Conformal LEC

- GUI mode: UNIX% `lec`
- Non-GUI mode: UNIX% `lec -nogui`
- Switch GUI & non-GUI modes
  - LEC> `set gui [on | off]`
- Batch mode:
  - UNIX % `lec -dofile <batch_file> -nogui`
  - LEC> `dofile <batch_file>`

Exiting Conformal LEC

- LEC> `exit -force`
- LEC automatically closes all windows upon exit

Conformal GUI Environment



Command: 3-2-1 Convention

- You can abbreviate most three-word commands

  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 the 3-2-1 convention does not produce a unique command, you can add more letters

Command: 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 the previous command

- Up and down arrow keys help avoid retyping previous commands

Alias

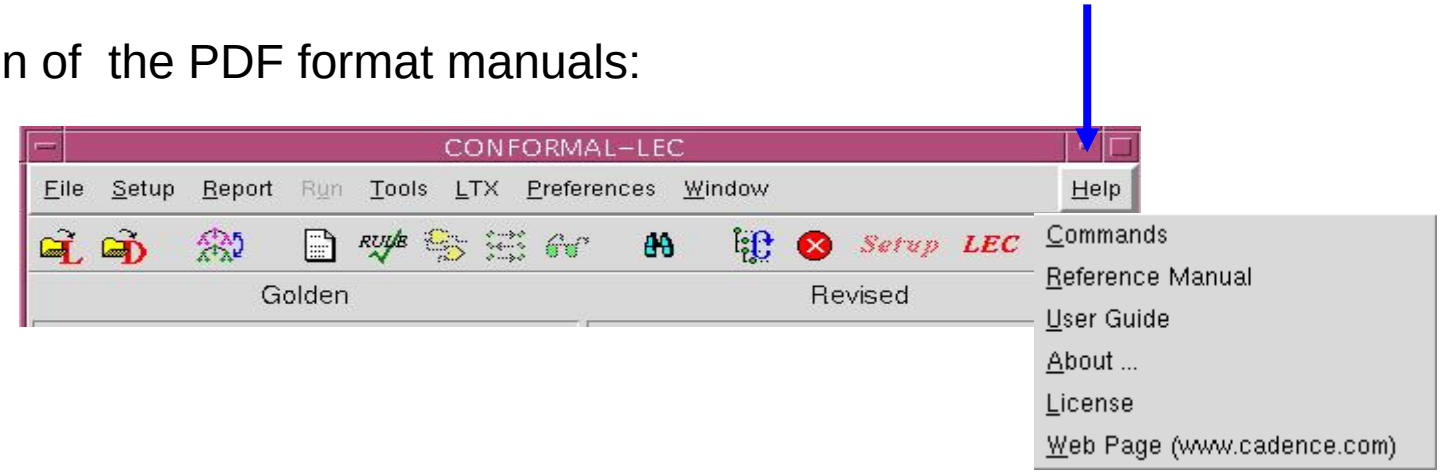
- Alias command usage: `add alias <name> <string>`
- Example alias in *.conformal_lec* file:

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

- Initial command files, *.conformal_lec,* executed in the following order:
  1. Verplex install directory: $VERPLEX_HOME/lib/.conformal_lec
  2. Home directory: ~/.conformal_lec
  3. 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

Notes

Running Conformal with Flat Compare Method

December 19, 2004

Conformal LEC Modes of Operation

SETUP mode

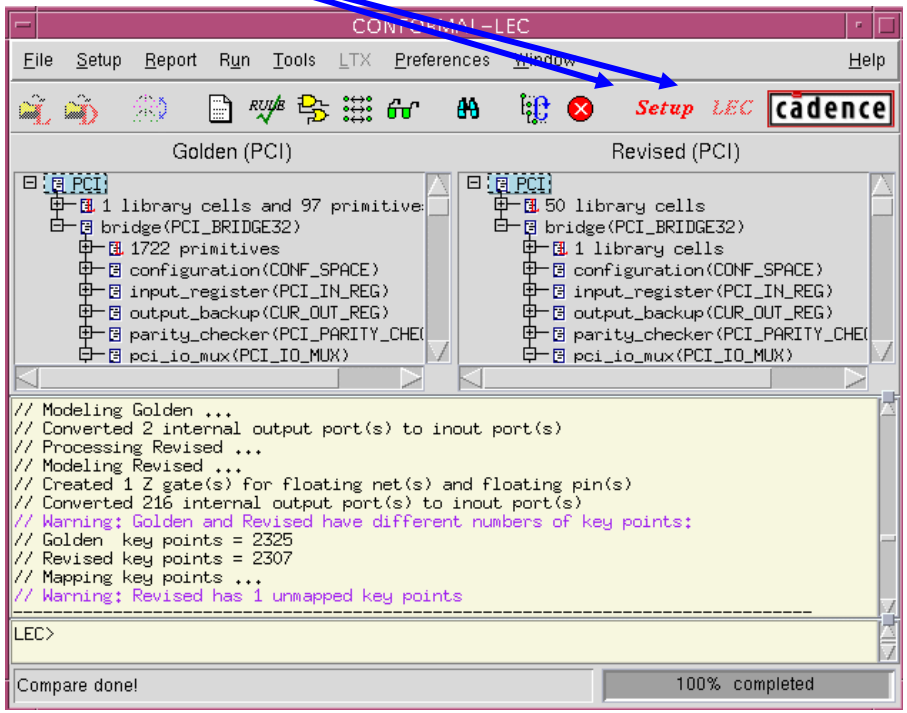
- Saving Conformal transcript to a log file
- Specifying black boxes
- Reading libraries and designs
- Specifying design constraints
- Specifying modeling directives

LEC mode

- Mapping process
- Comparing process
- Reporting run statistics

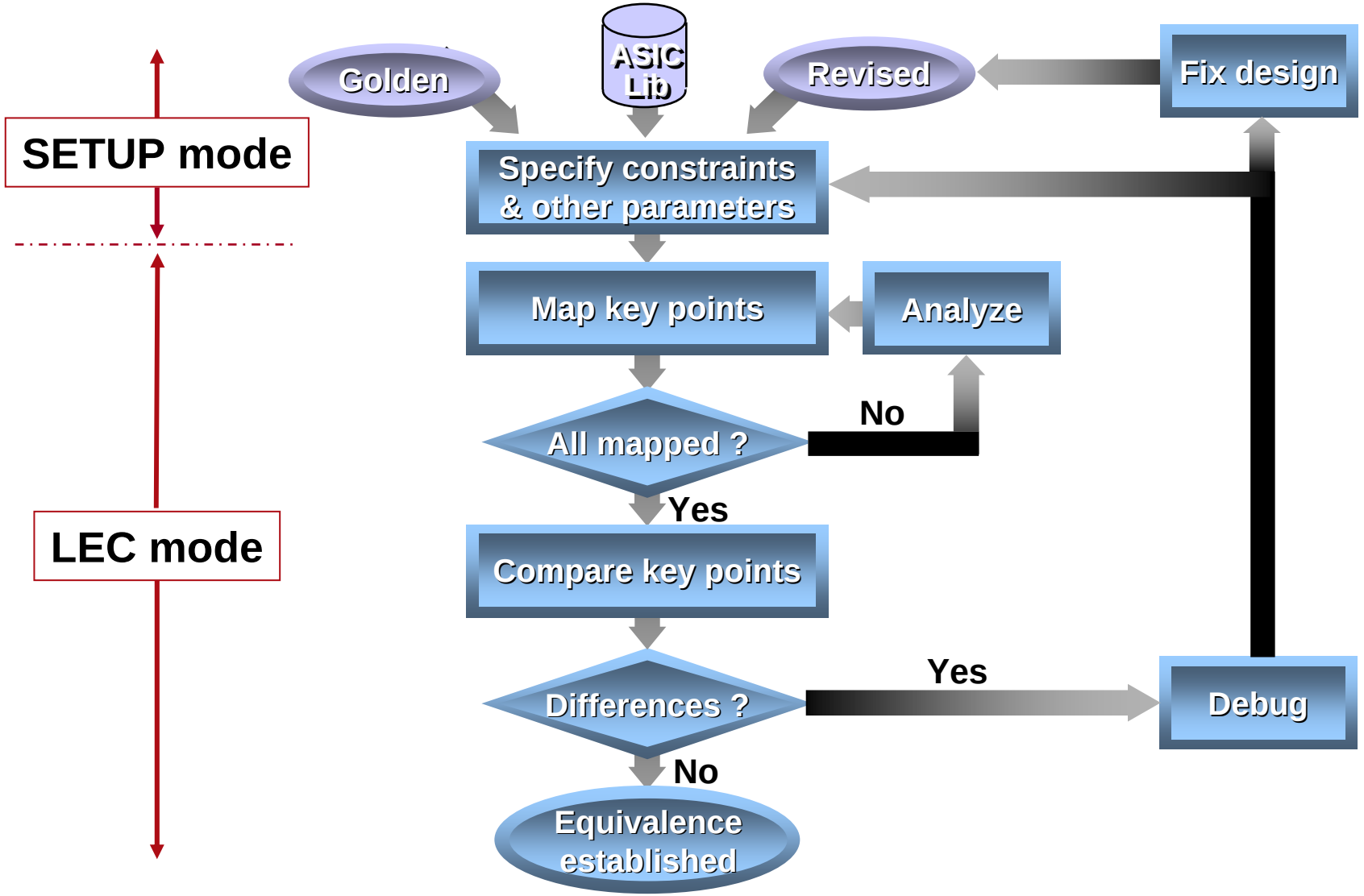
Switching Modes of Operation

- GUI: click



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

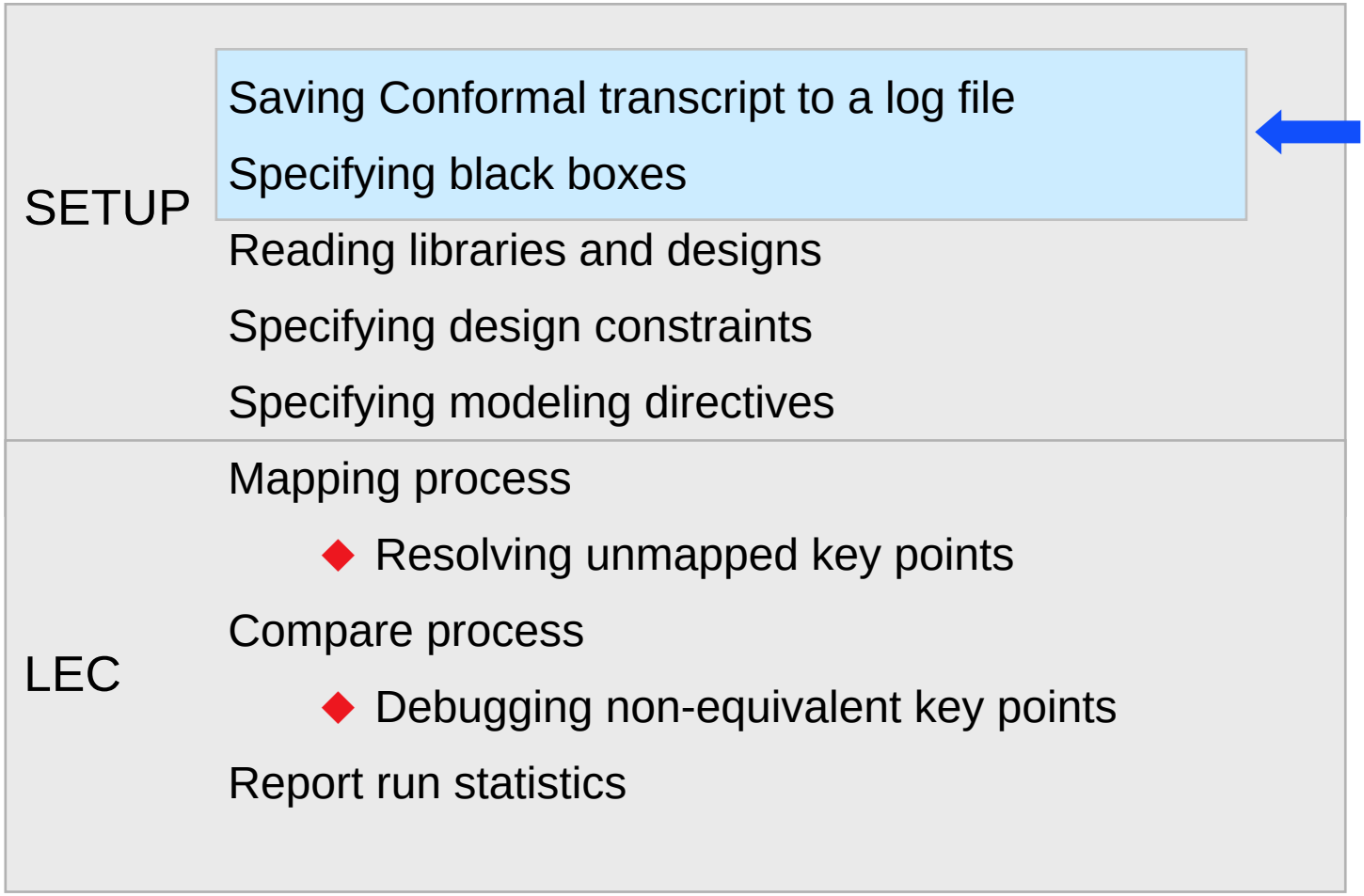
Conformal LEC Flow



Flat Compare Method:
Set Up Before Reading Library and Design Files

December 19, 2004

A Typical Session (Flat Compare Method)



Saving Conformal Transcript to a Log File

Specify the name of the log file:

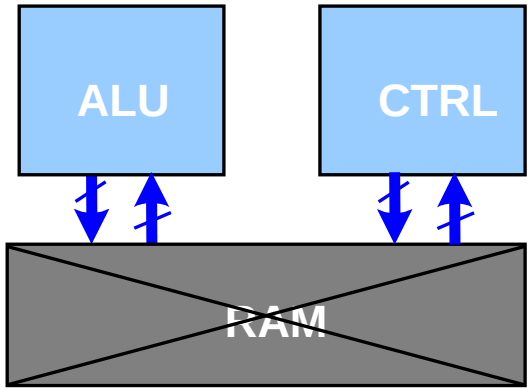
```
set log file logfile -replace
...
```

Or

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

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 –library -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`

Transcript window

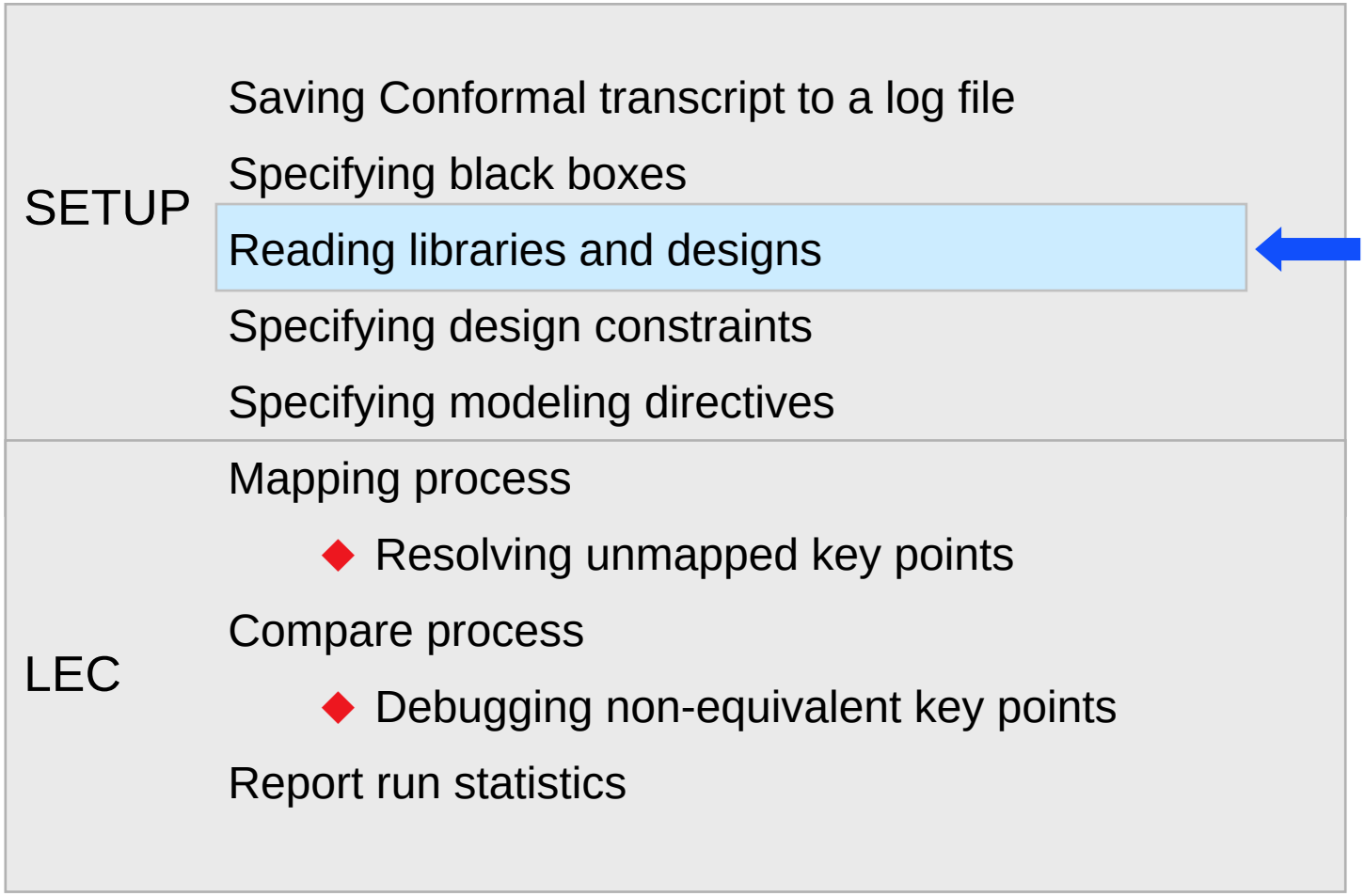
```
SETUP> report black box
SYSTEM: (G R) ram8x256
SYSTEM: (G R) hstm
```

- Check for unbalanced black boxes in the Golden and Revised designs

Flat Compare Method:
Reading Library and Design Files

December 19, 2004

A Typical Session (Flat Compare Method)



Supported Formats

Library

- Verilog (standard simulation libraries)
- SystemVerilog subset
- Liberty™

Design (Synthesizable RTL, gate)

- Verilog
- SystemVerilog subset
- VHDL
- SPICE
  - SPICE is supported in Conformal Ultra and Conformal Custom

Reading Verilog Library & Design: Syntax

Read library:

```
READ LIbrary <filename …>
          [-VErilog | -VERILOG2K | SYStemverilog | -Liberty]
          [-REPlace | -APPend]
          [-Both | -Golden | -Revised]
```

Read design:

```
READ DEsign <filename …>
          [-file <verilog_command_file>]
          [-VErilog | -VERILOG2K | -SYStemverilog]
          [-Golden | -Revised]
```

Reading Verilog Library & Design: Method 1

- Read files by using the 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 search path specification

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

Reading Verilog Library & Design: Method 2

- Read files by using the Verilog command file
- Conformal reads only the 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-languages: 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-languages: 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

- Violation message

Transcript window

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

- To see what a message means:

  SETUP> `help <message_name>`

  Example:

  SETUP> `help RTL2.3`

HDL Rule Checks (continued)

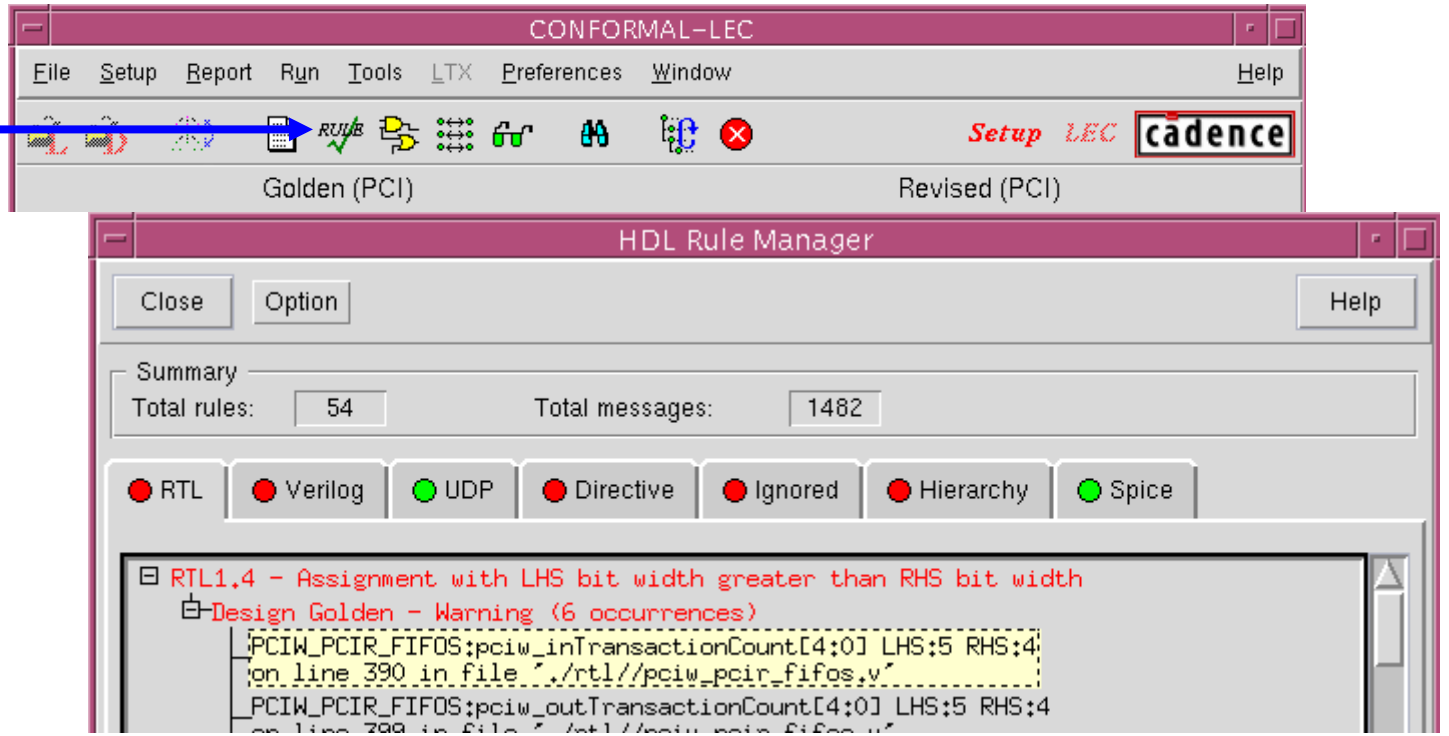
Locate the source that causes a message.

- Non-GUI mode

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

- GUI mode

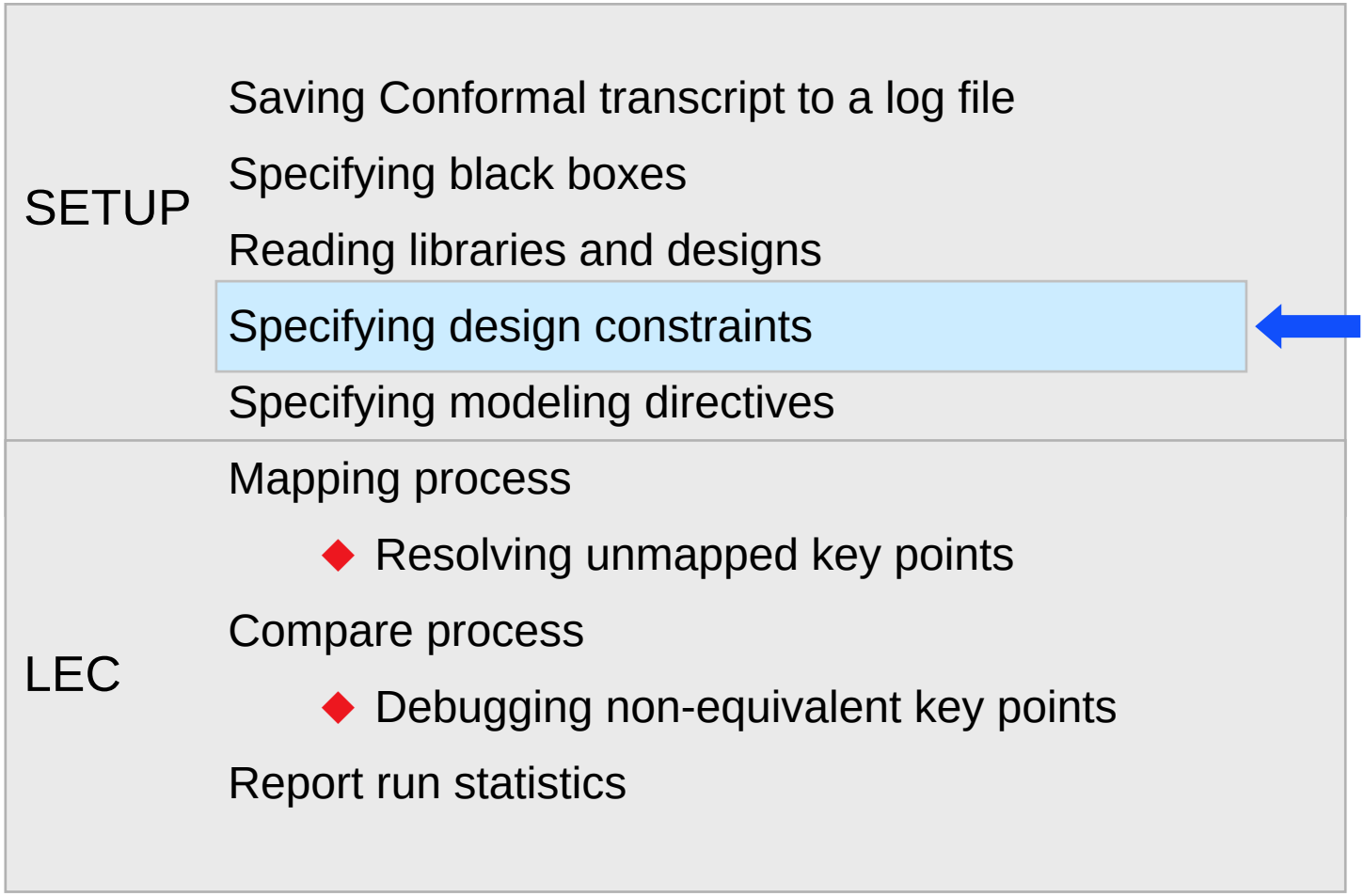
Click to open the HDL Rule Manager



Flat Compare Method:
Specify Design Constraints

December 19, 2004

A Typical Session (Flat Compare Method)



Design Constraints

What are design constraints?

- User's inputs to control part of a design's logic

Purpose of constraints

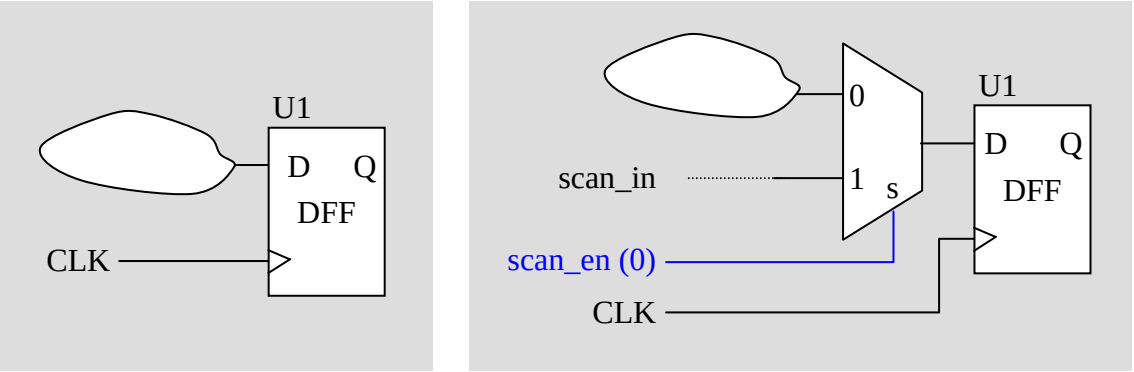
- To disable test logic (for example; scan and JTAG)
- To specify one-hot or one-cold conditions
- To specify relationships between pins
- To constrain undriven signals

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).

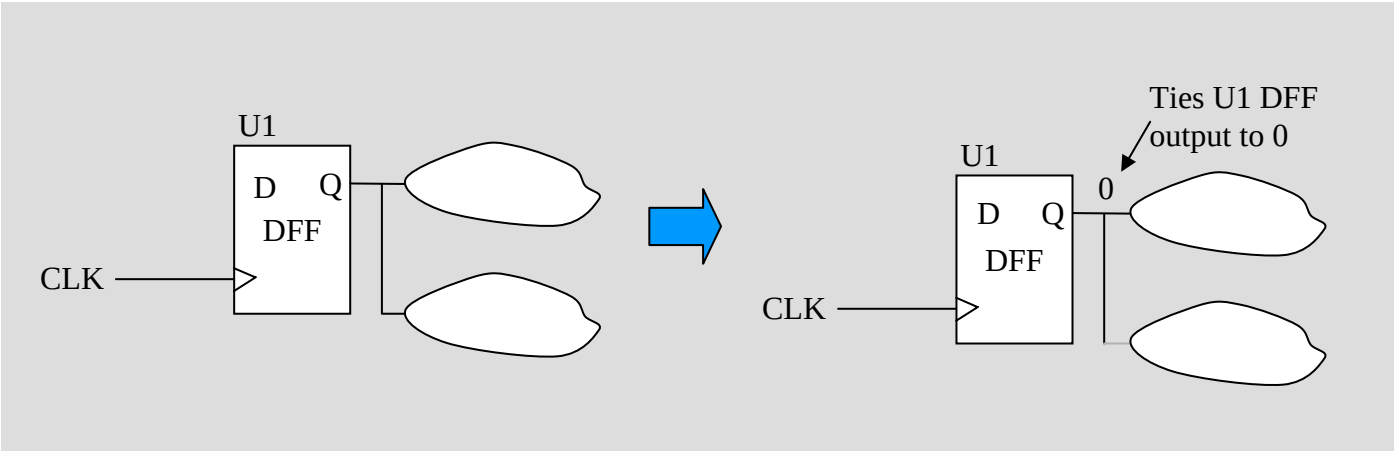


Golden Revised

```
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

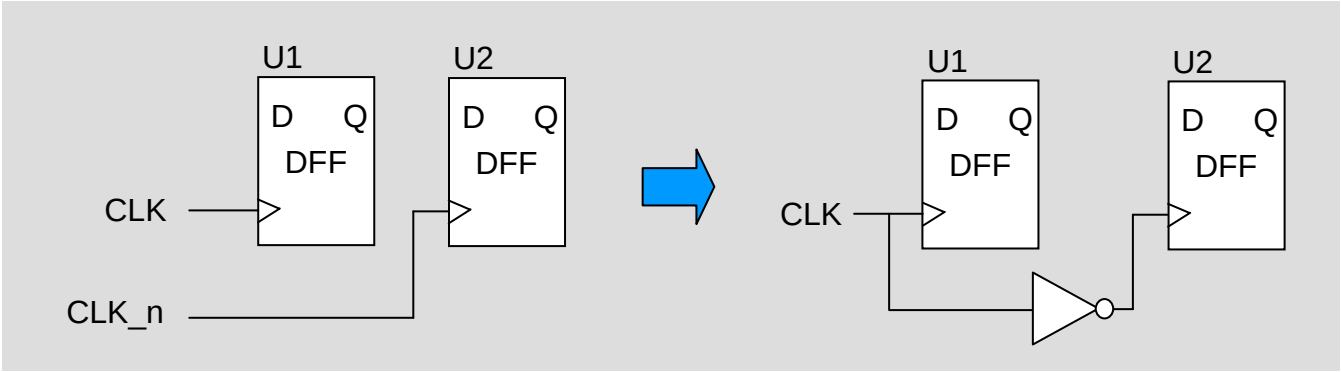
Apply to any internal DFF or D-latch output to logic-0 or logic-1 (for example; JTAG registers).




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

Pin Equivalence

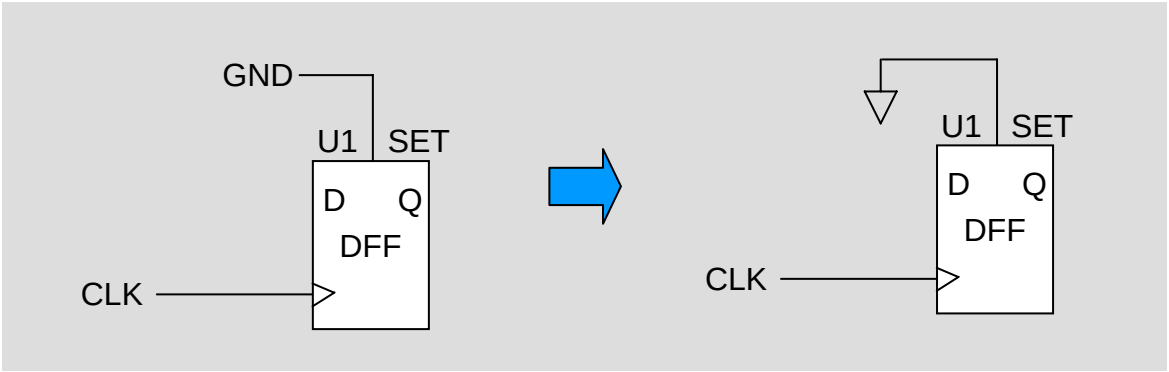
Specify the relationship (equivalent or inverted equivalent) between two or more primary input pins.




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

Tied Signal

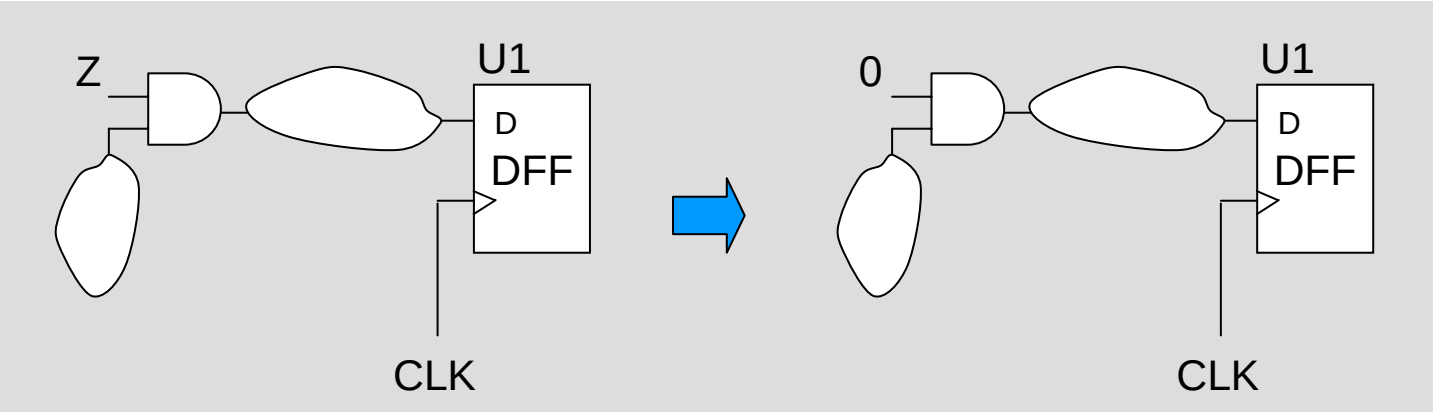
Specify floating nets or pins to be tied to Logic-0 or Logic-1 (for example; equate GND to 0 or VDD to 1).




```
...
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).

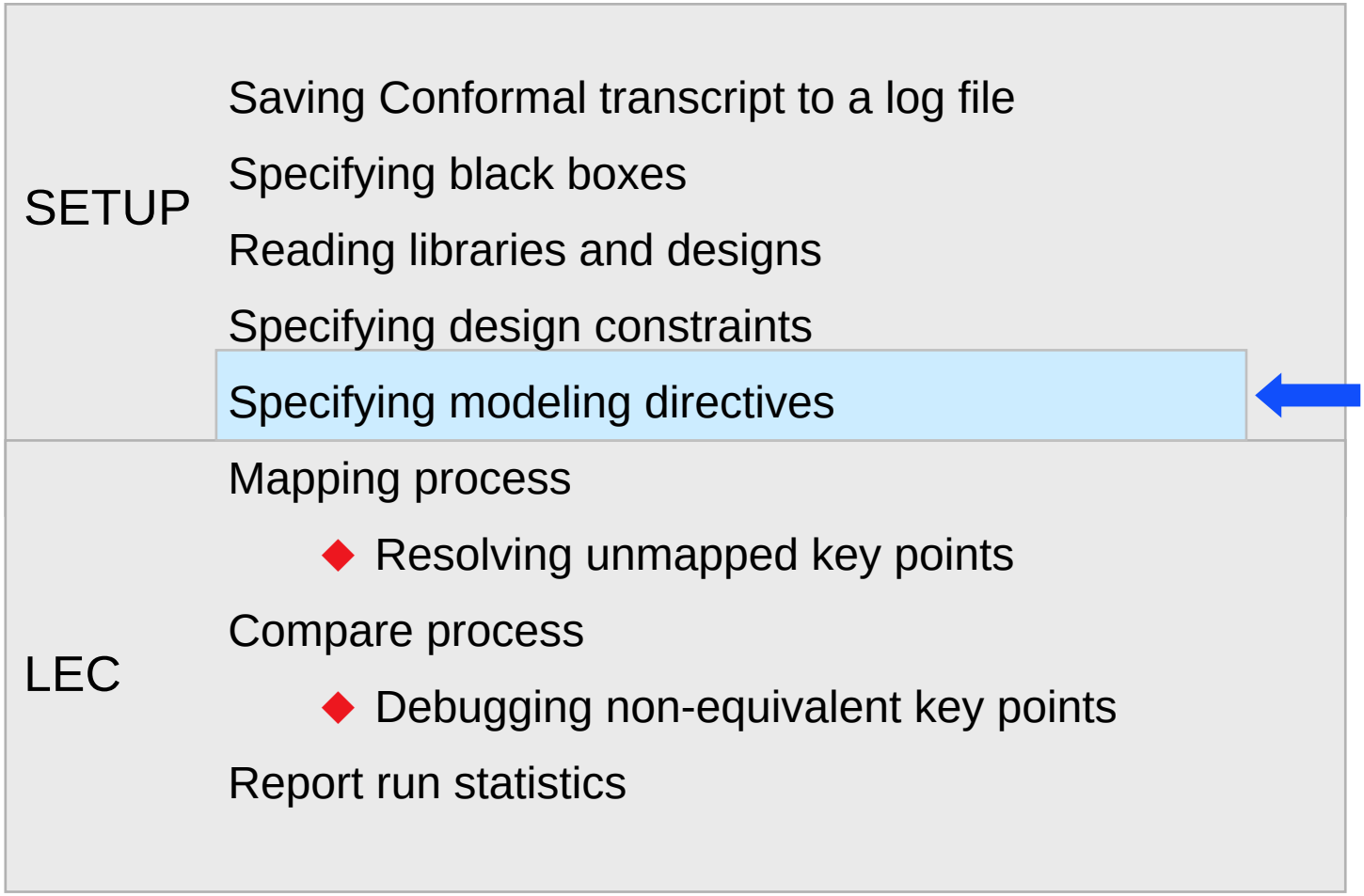



```
...
set undriven signal 0 -revised
...
```

Flat Compare Method:
Specifying Modeling Directives

December 19, 2004

A Typical Session (Flat Compare Method)

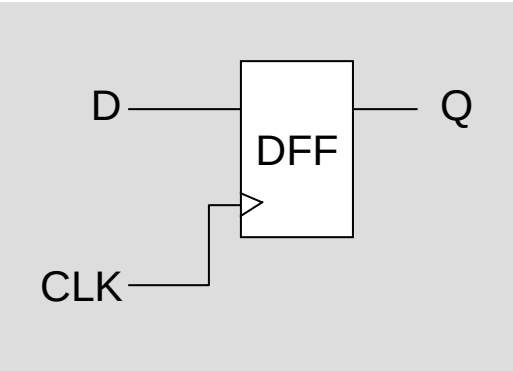


Modeling Directives

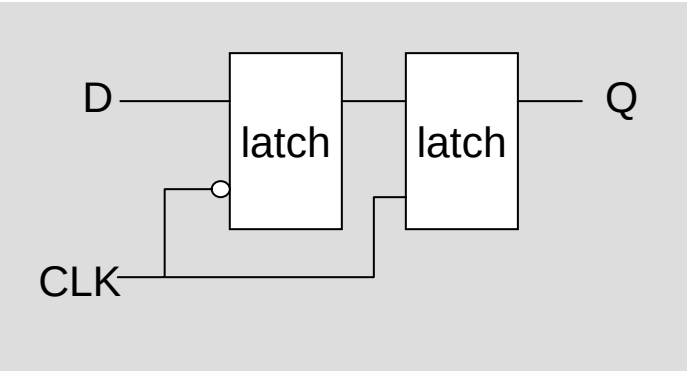
- In SETUP mode, you can specify directives to influence the way Conformal LEC models the design.
- Modeling directives are needed to handle modeling styles specific to vendors' libraries or synthesis tools.
- Examples of modeling options:
  - Latch folding
  - Gated-clock
  - Sequential redundancy
  - Sequential constant
  - Latch transparent
  - Sequential merge

Latch Folding: Problem

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



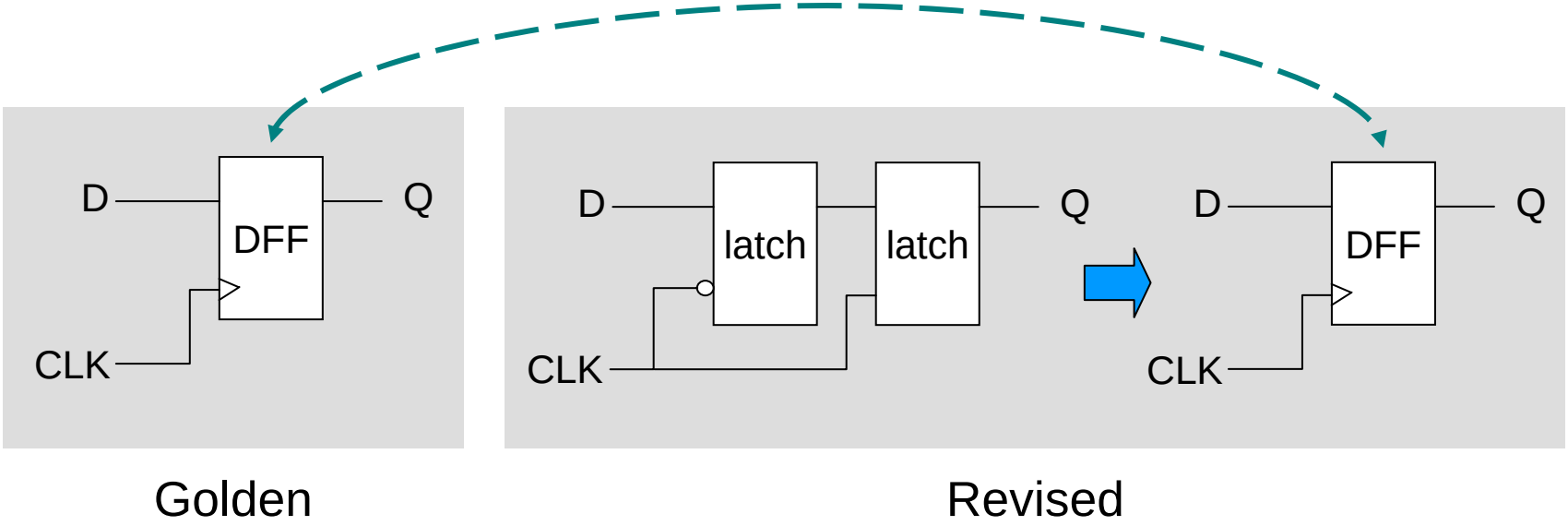
Golden



Revised

Latch Folding: Solution

Fold the 2 D-latches into a DFF.



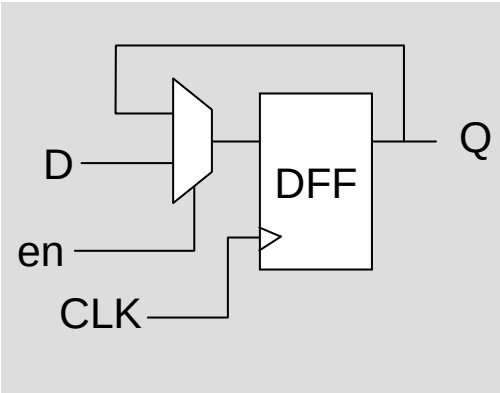
Golden

Revised

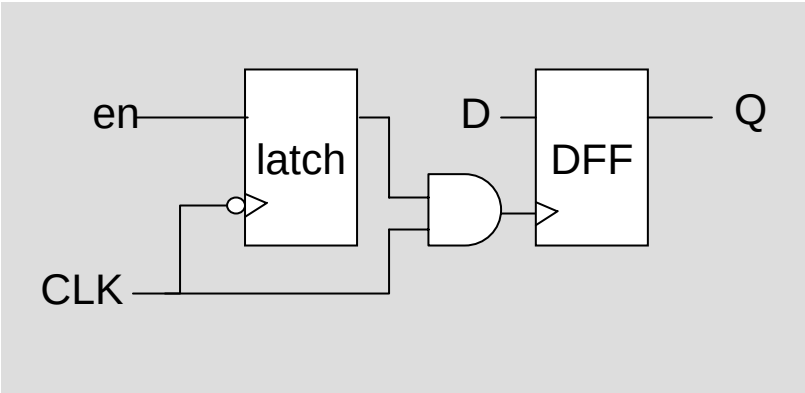
```
...
set flatten model -latch_fold
...
```

Gated-Clock: Problem

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



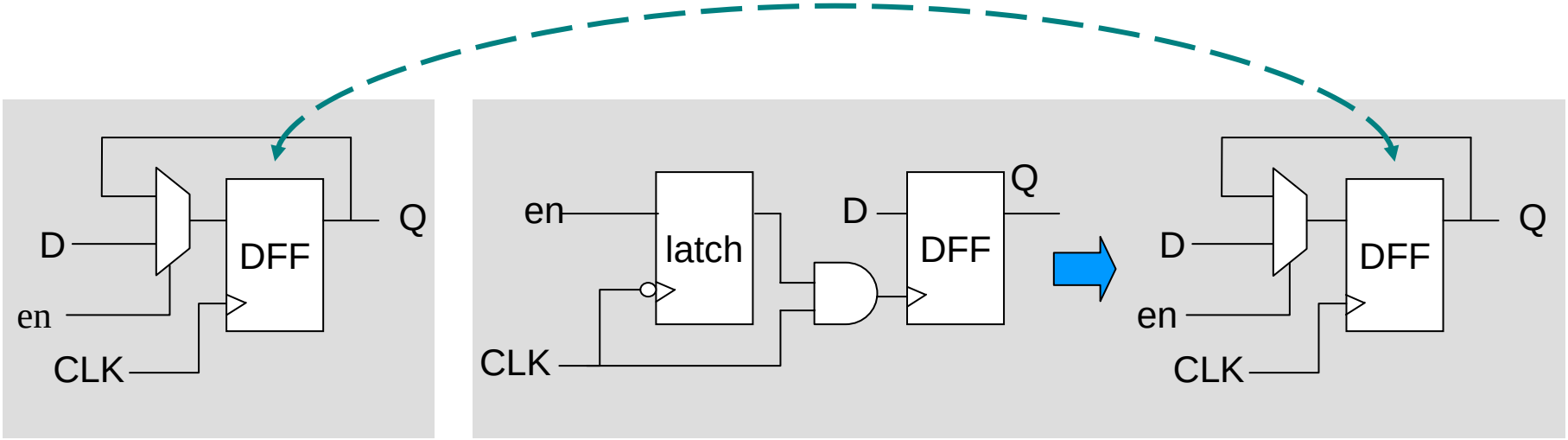
Golden



Revised

Gated-Clock: Solution

Enable gated-clock learning.



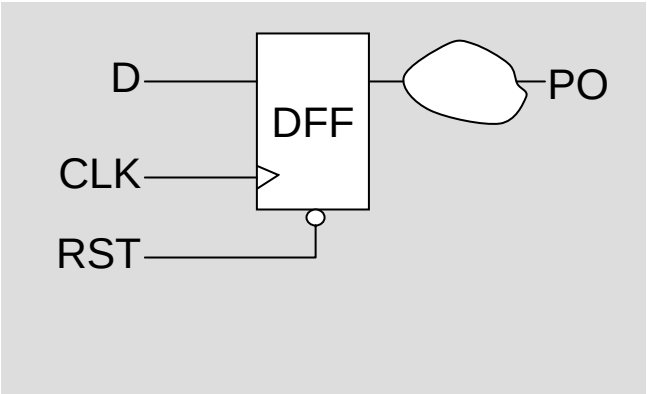
Golden

Revised

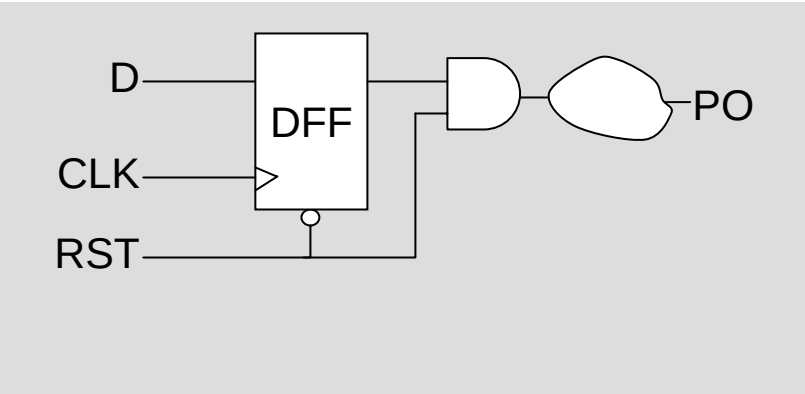
```
...
set flatten model –gated_clock
...
```

Sequential Redundancy: Problem

- A redundant AND gate is introduced so that the reset signal takes effect right away.
- Causes comparing problem!



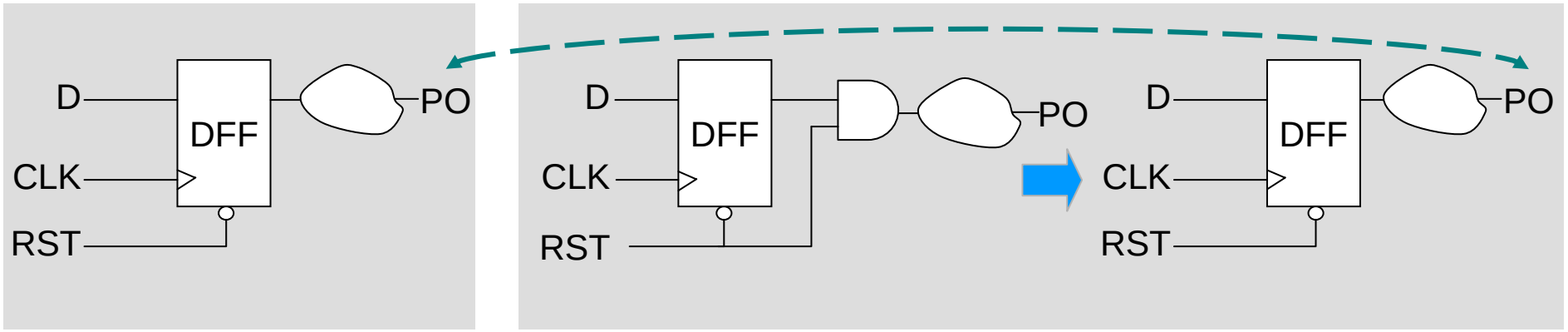
Golden



Revised

Sequential Redundancy: Solution

Remove sequential redundancies.



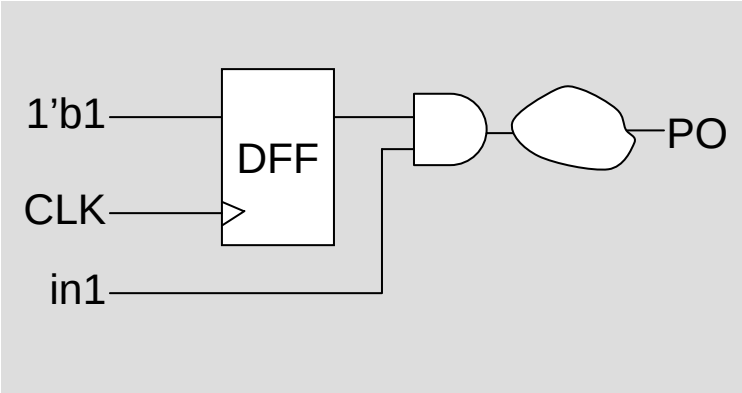
Golden

Revised

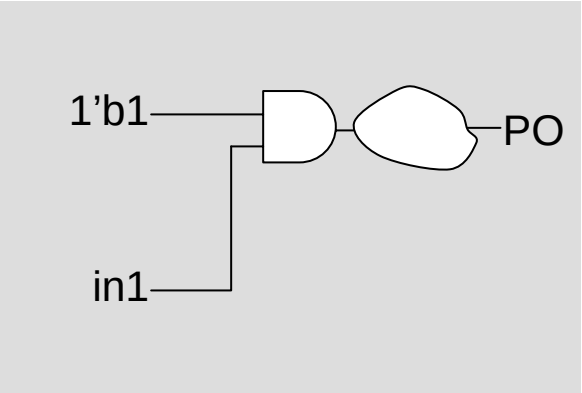
```
...
set flatten model –seq_redundant
...
```

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!



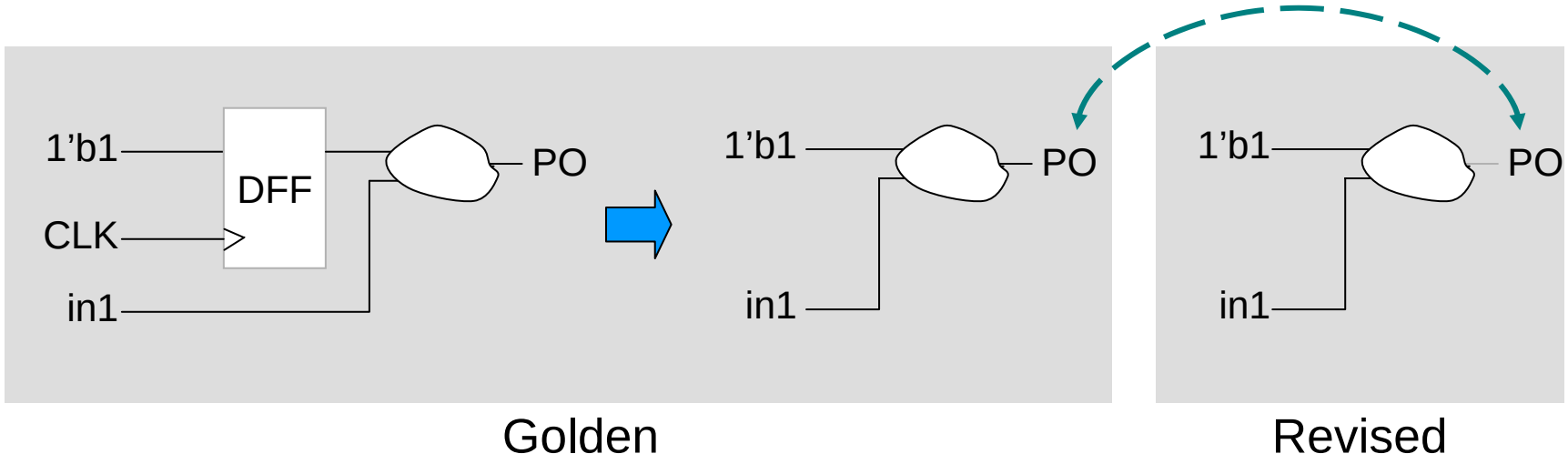
Golden



Revised

Sequential Constant: Solution

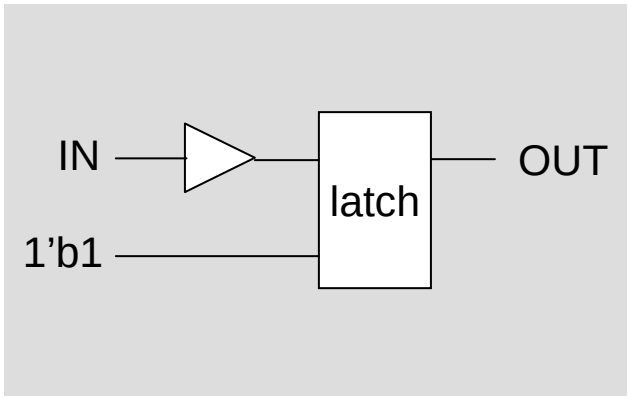
Convert a DFF or a D-latch to a ZERO/ONE gate if the data port is set to 0/1.



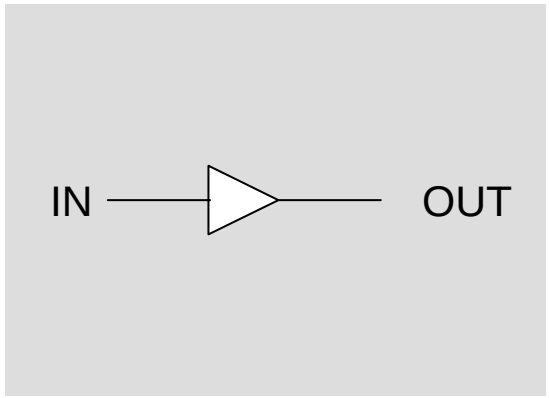

```
...
set flatten model –seq_constant
...
```

Latch Transparent: Problem

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



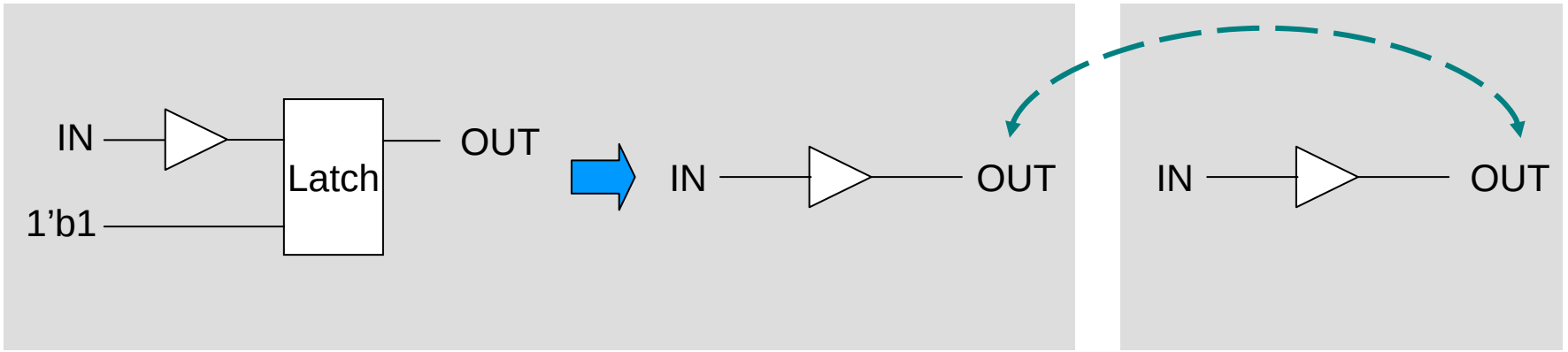
Golden



Revised

Latch Transparent: Solution

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

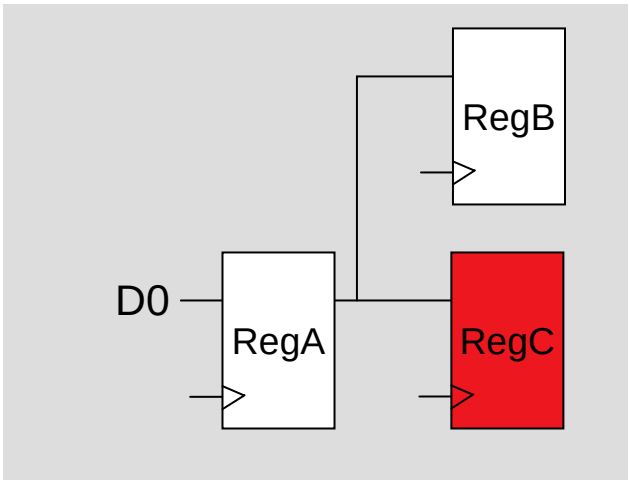


Golden Revised

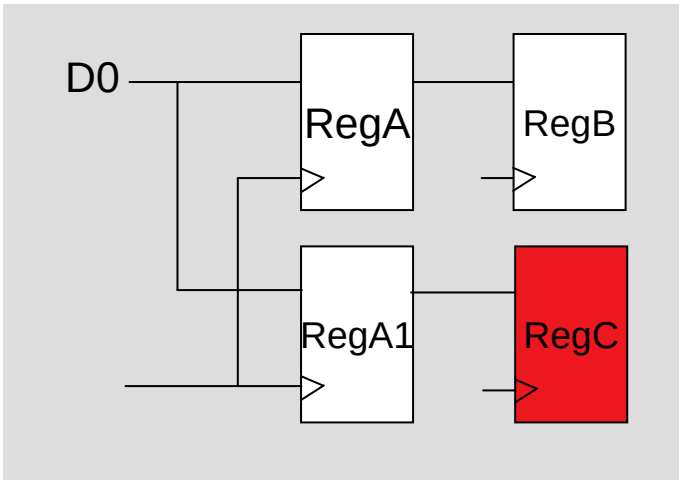
```
...
set flatten model –latch_transparent
...
```

Sequential Merge: Problem

- Duplicated register in the clock and/or data logic cone
- Causes comparing problem!
  - Different supporting key points for RegC (RegA vs RegA1)



Golden

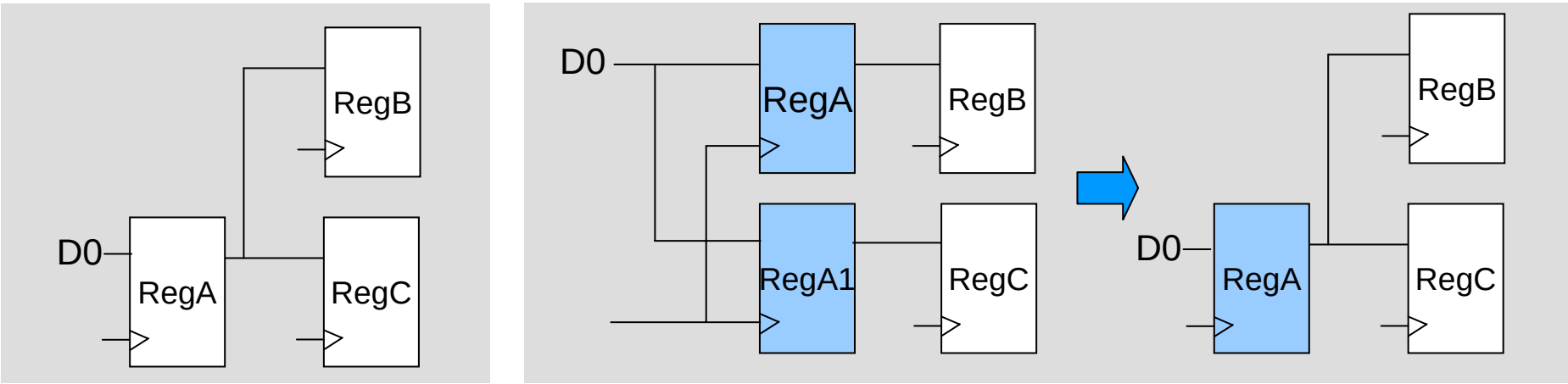


Revised

Sequential Merge: Solution

Merge equivalent sequential elements into one.

- Merge RegA1 into RegA in the Revised



Golden Revised

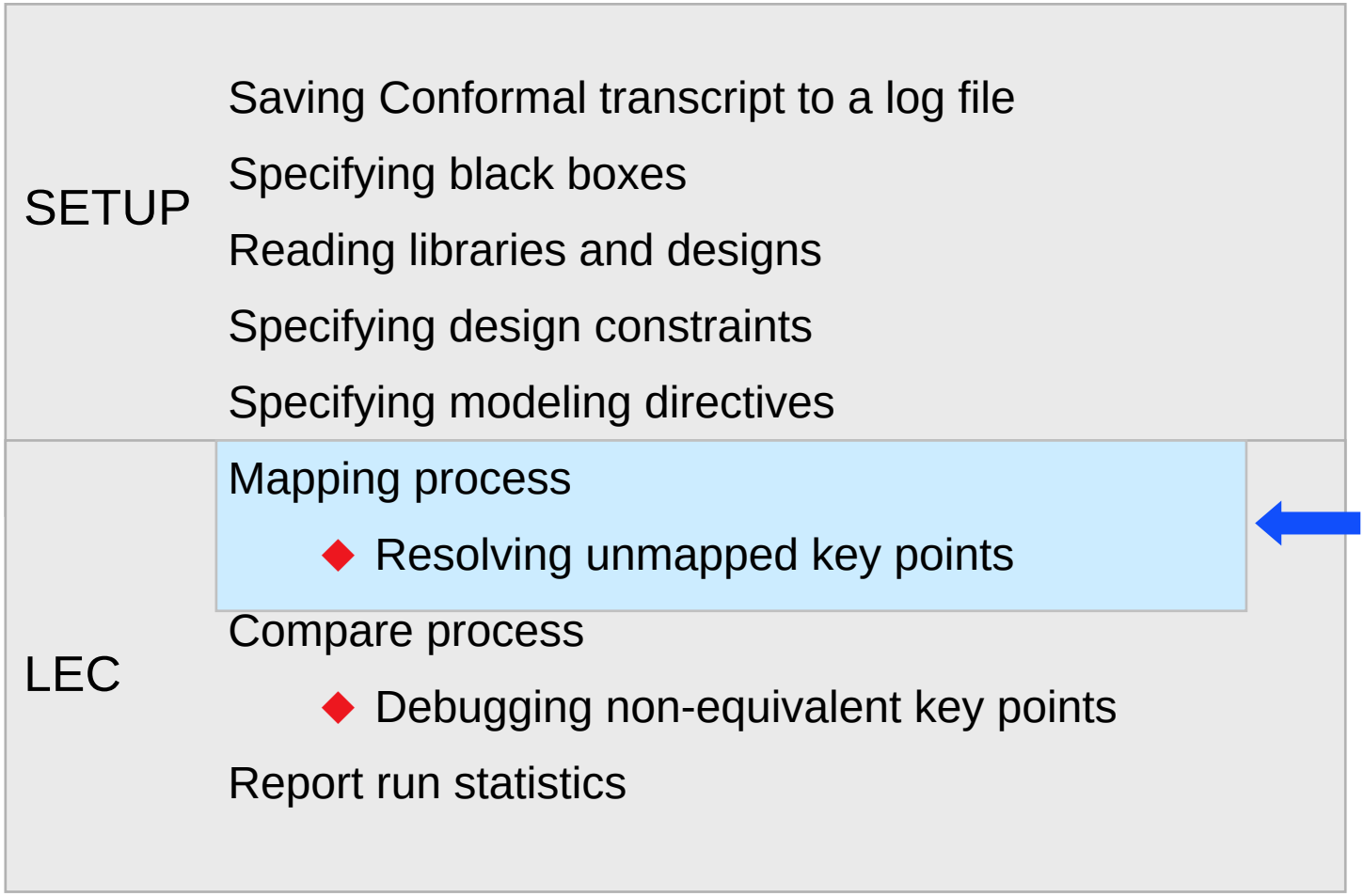
```
...
set flatten model –all_seq_merge
...
```

Notes

Flat Compare Method:
Mapping Process and Using Renaming Rules

December 19, 2004

A Typical Session (Flat Compare Method)



Switching to LEC Mode

When change mode from SETUP to LEC

- Golden and Revised designs are flattened
- Circuit modeling is performed
- Automatic key point mapping take places after circuit modeling
- All of the steps above happen through one command

```
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

- Messages

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 report modeling messages

LEC> `report messages -model`

Modeling Messages (continued)

- Example of reported messages

Transcript window

```
LEC> report messages
// Command: report messages
Report modeling message for Golden
F34: converted X assignment(s) as don't care(s) (Occurrence: 78)
Report modeling message for Revised
F5: Folded DLAT(s) into DFF(s) (Occurrence: 340)
```

- To see what a message means

  LEC> `help <message_name>`

  Example: LEC> `help F5`

  Note: Modeling messages are documented in the User Guide

- To view instances affected by a particular message

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

Mapping Process - Outline

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

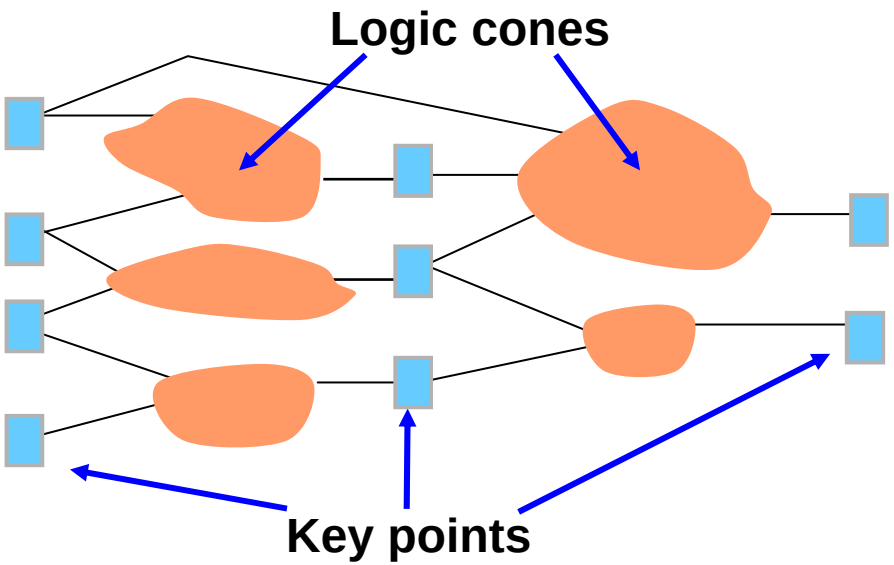
- Mapping takes too long
- Incomplete mapping

What Are Key Points?

Key points are defined as:

- Primary inputs(PI)
- Primary outputs(PO)
- D Flip-Flop
- D Latches
- Black boxes (BBOX)
- TIE-Z gates
- TIE-E gates
- CUT gates

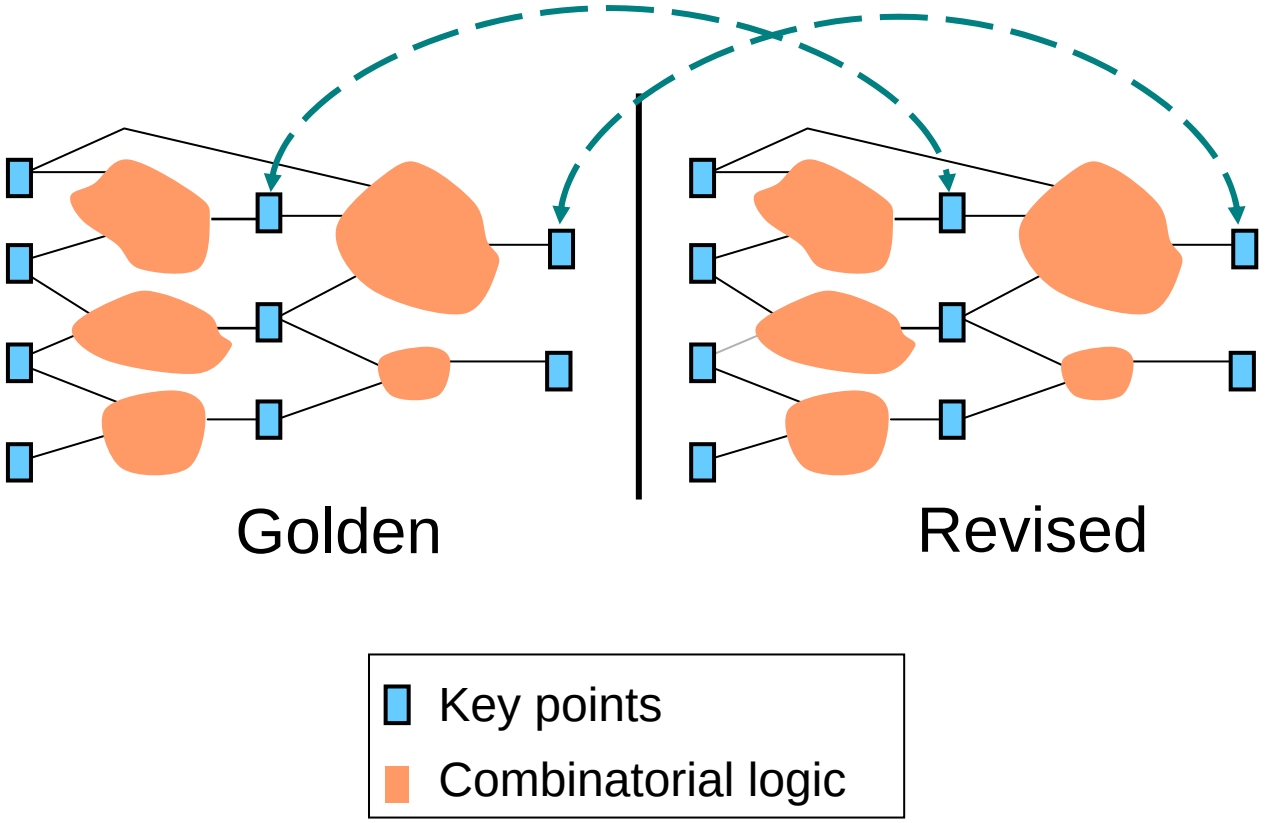
The design consists of combinational logic cones bounded by key points.



What Is Key Point Mapping?

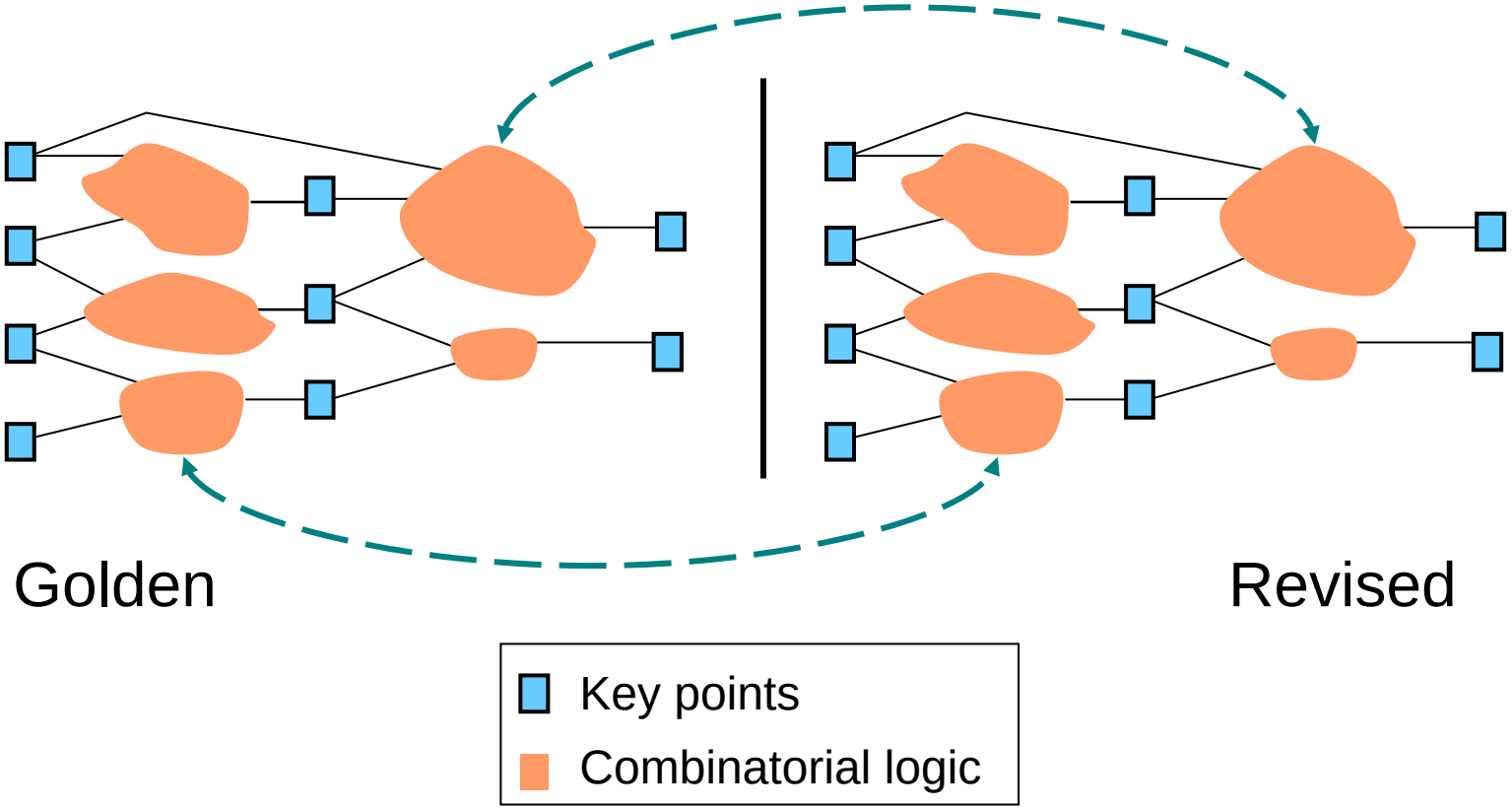
Pairing corresponding Golden/Revised key points:

G	R
PI	PI
PO	PO
DFF	DFF
DLAT	DLAT
BBOX	BBOX
CUT	CUT
Z	Z



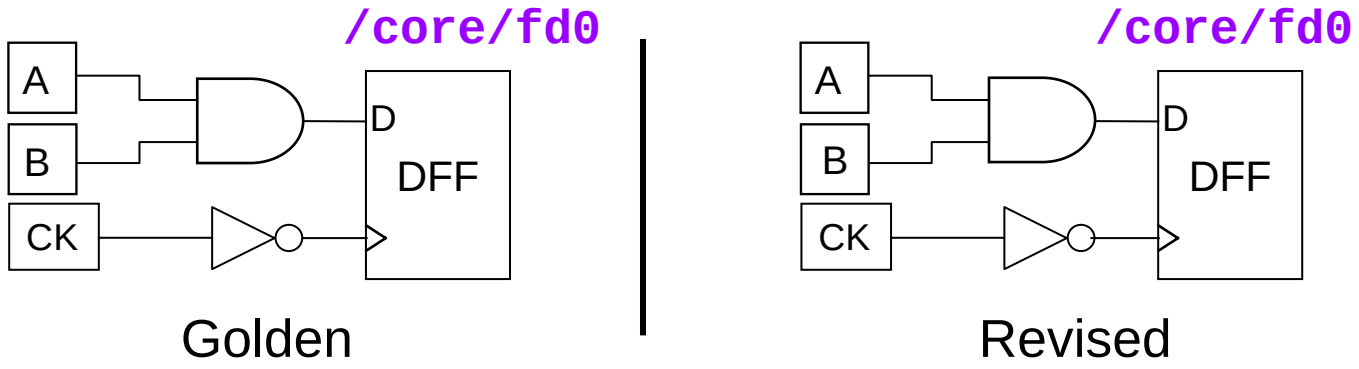
Why Is Mapping Necessary?

So that corresponding combinational logic cones are correctly paired.

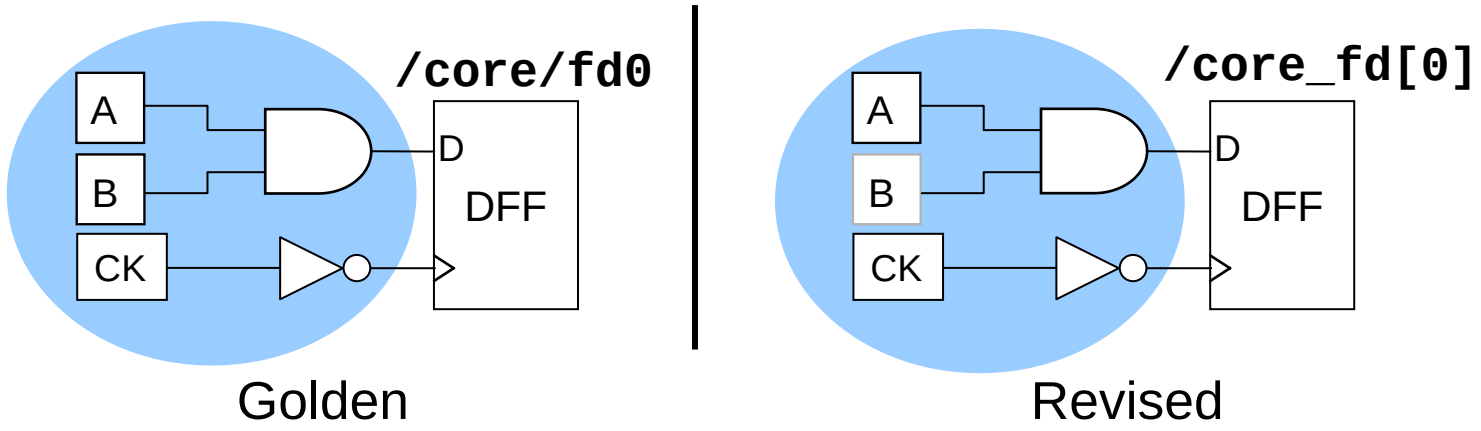


How Is Mapping Done?

Name-based



Function-based



Mapping Options

Command for 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`

- Message 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 Conformal 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

Message 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

- Message after mapping

Transcript window

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

- Unmapped points must be resolved

Solution: Add renaming rule

- Determine renaming rules from the Mapping Manager

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

Special characters in search_string

- % . * ^ $ | ( ) [ ] / \
- Needs to be preceded by the escape character, "\"

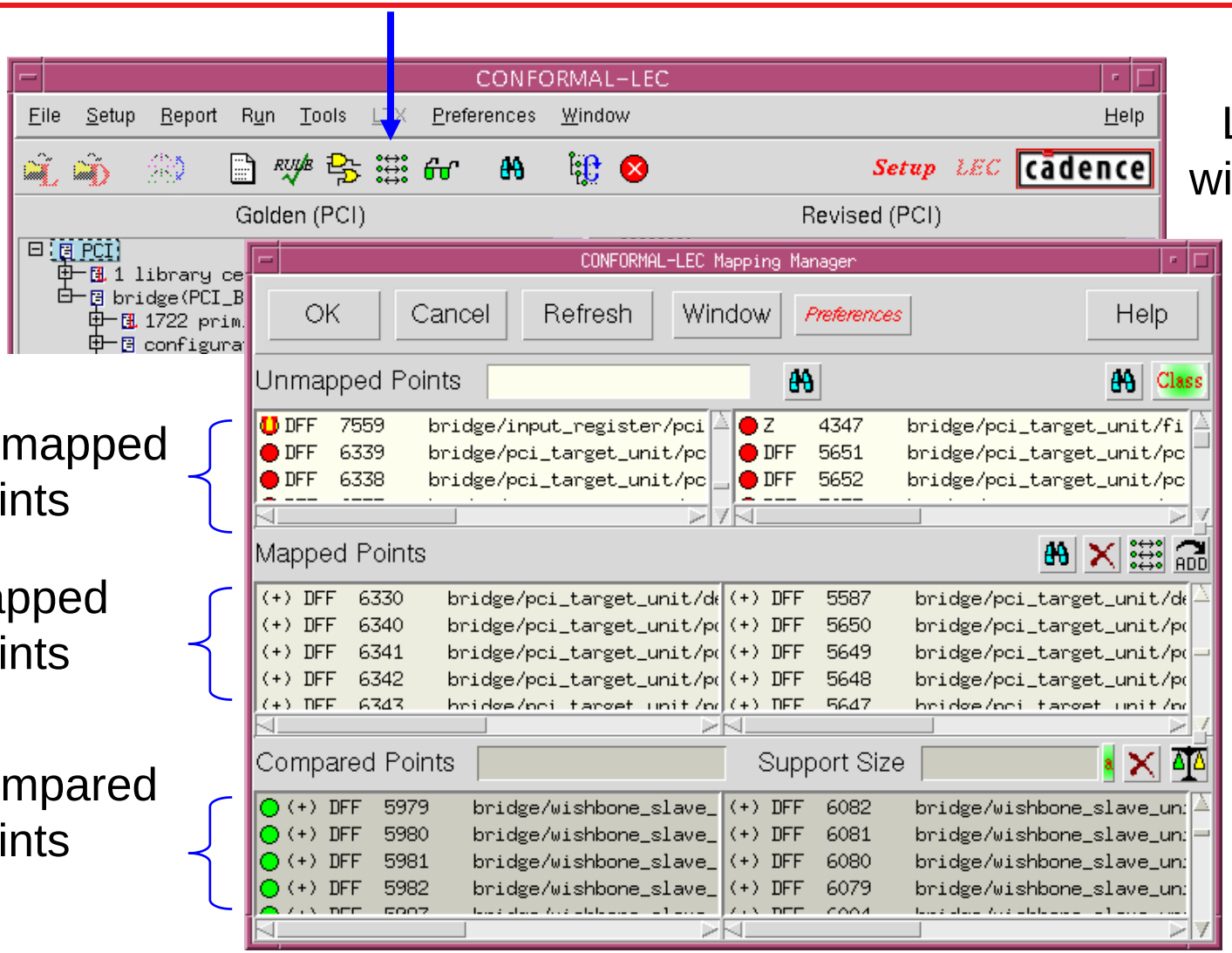
Add Renaming Rule: Example

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



LEC window

Unmapped Points

Mapped Points

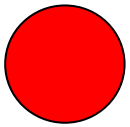
Compared Points

Mapping Manager

Categories of Unmapped Points

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*

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

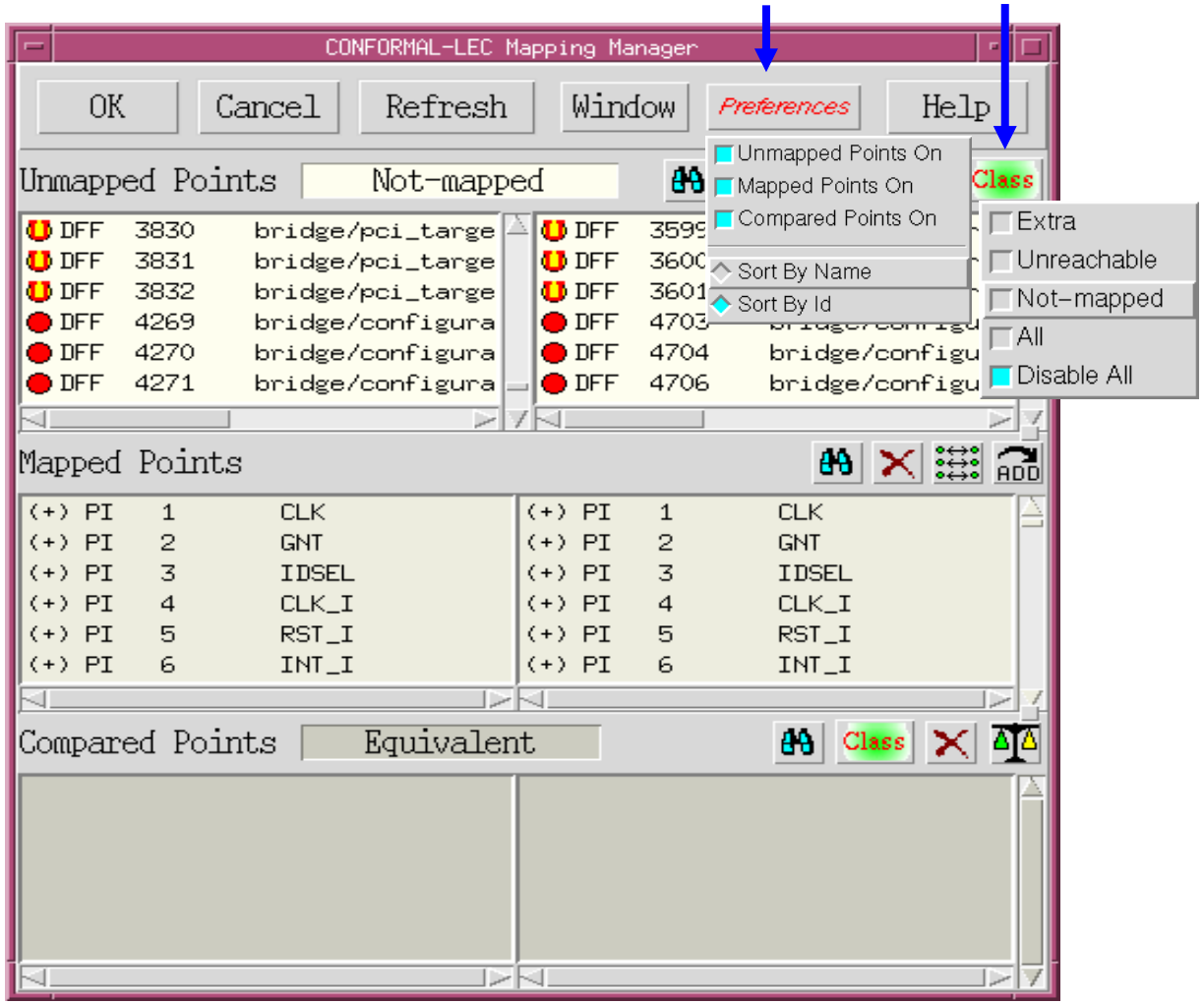


Not-mapped: Key point that has no correspondence on the other side. May be resolved with renaming rules (red-filled circle)

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

LEC> `report unmapped points`

Resolving Incomplete Mapping



Using Mapping Manager to create renaming rules.

Tips

- Preference → Sort by name
- Class → Disable All → Not-Mapped

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
...
```

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 (continued)

- Test all added renaming rules.
  - 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
========================================================================
```

- Rules are order dependant.

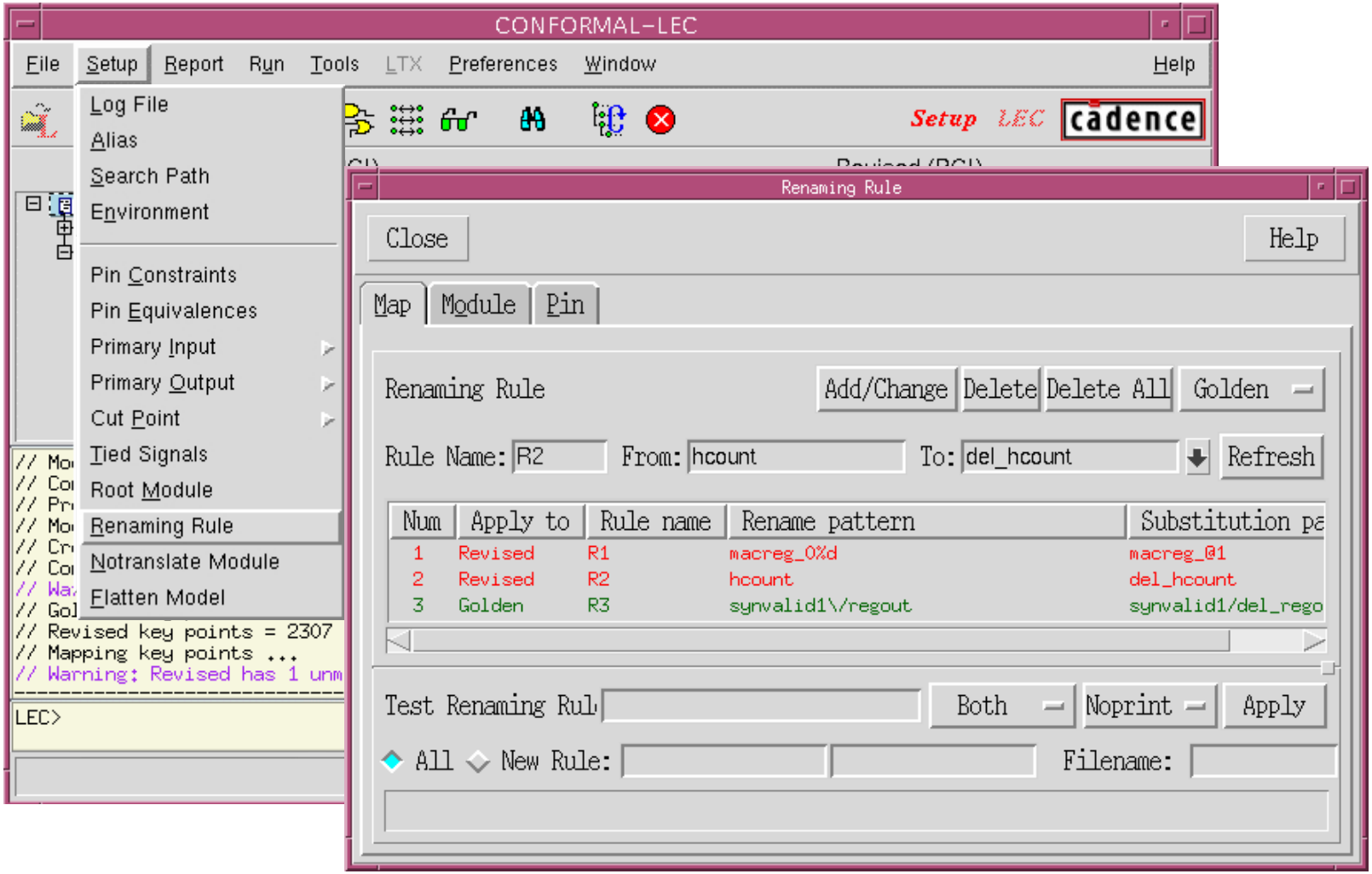
Built-in Renaming Rules

Built-in renaming rules 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.



Exercise

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

1	Golden `/u0/u1/c_reg[5]`	Revised `/u0/c_reg(5)/I1/U$1`
2	*answer:* `add renaming rule r1 "u1\/" "" -golden` Note: Trailing characters for library cells are truncated automatically (`/I1/U$1`)	
3	Golden `/u0/u1/c_reg[1]`	Revised `/u0_u1_c_.ram1_reg`
	answer: `add renaming rule` _____	
	Golden `/u0/b_reg[24]`	Revised `/u0/b_2_reg[22]`
	answer: `add renaming rule`	

For solutions, please see next page.

Exercise Answers

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]`

Summary

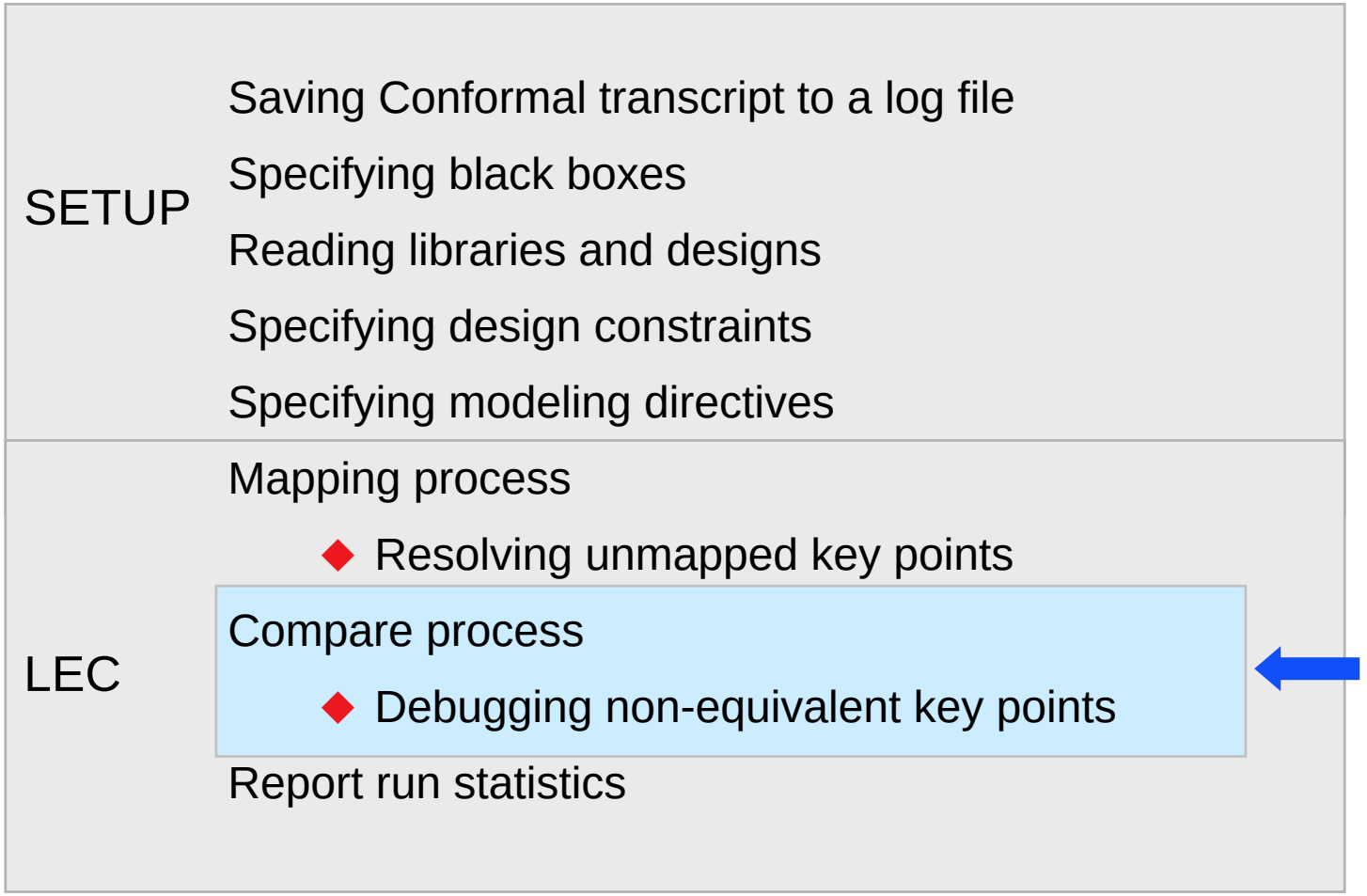
Once you are satisfied with the mapping results, you can incorporate all renaming rules into your dofile.

```
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 -revised
set system mode lec
...
```

Flat Compare Method: Compare Process

December 19, 2004

A Typical Session (Flat Compare Method)



Compare Process - Outline

Understanding the compare process:

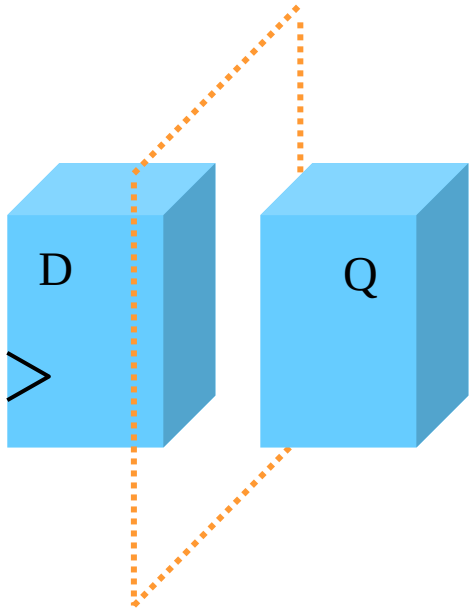
- How is the state element handled?
- What are the compare points?
- What is being compared?

Debugging non-equivalent key points

How Is the State Element Handled?

No input and output connection

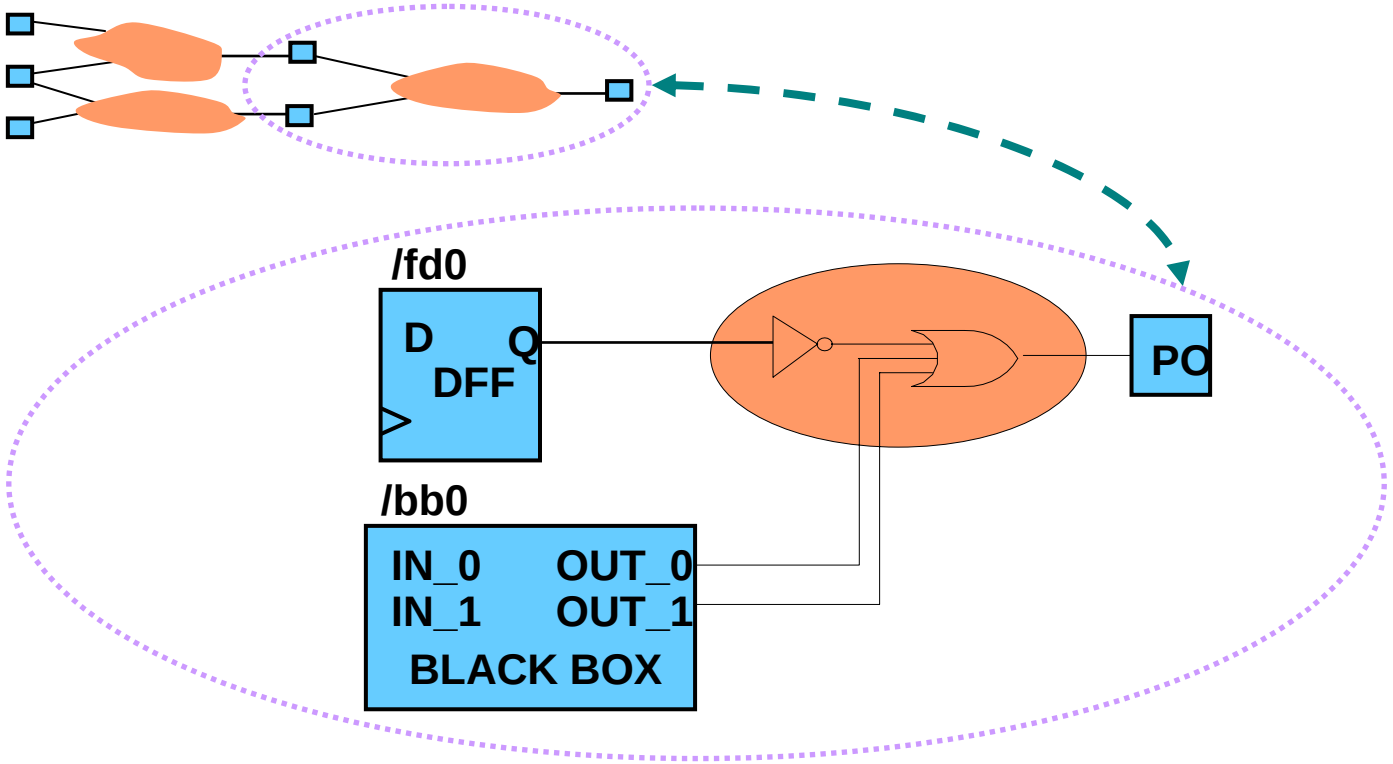
- Think of a D-latch or a DFF being cut in halves
- Input and output are verified separately



What Are Compare Points?

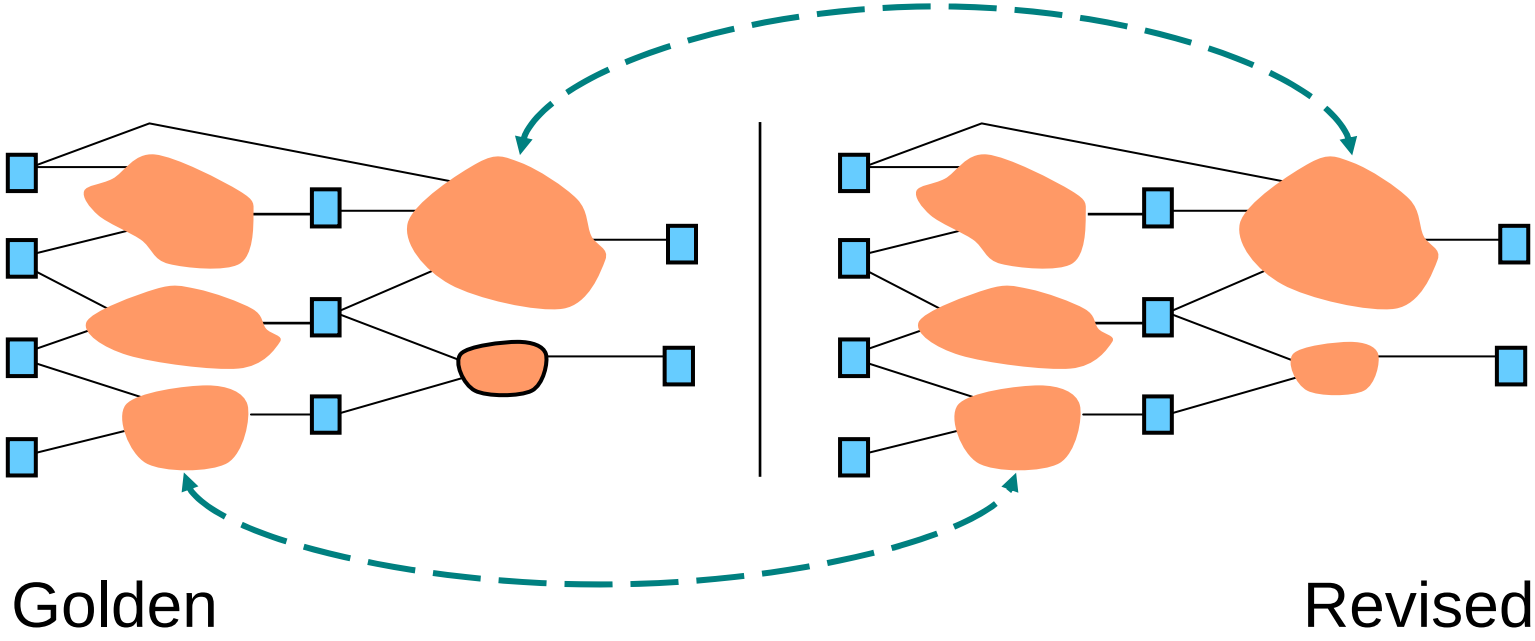
Compare points are:

- 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

Comparison

- Only mapped points can be compared
- Comparison is an iterative process
  - Conformal remembers points already compared equiv/non-equiv
  - Can interrupt with "Ctrl-c"
  - Enter "`compare`" to continue comparing

```
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

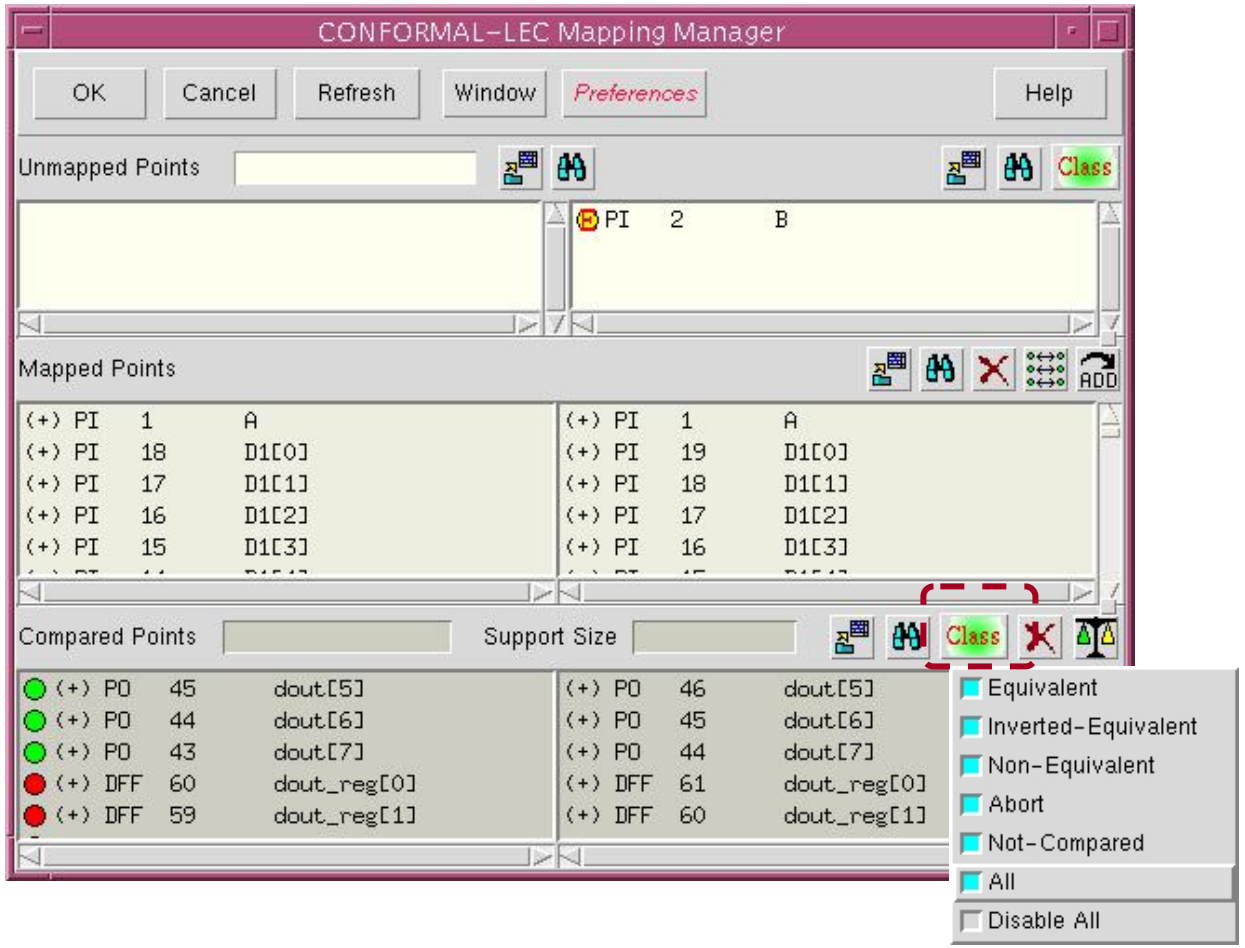
Message after compare:

Transcript window

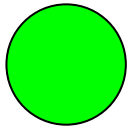
```
...
// Command: compare
================================================================================
Compared points     PO     DFF    DLAT   BBOX      Total
--------------------------------------------------------------------------------
Equivalent          2      146     2       1        151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

Comparison Results in GUI

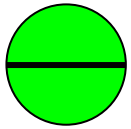
Filtering comparison results on the Mapping Manager



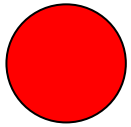
Categories of Comparison Results



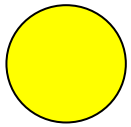
Equivalent: Key points proven to be equivalent (green-filled circle)



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)

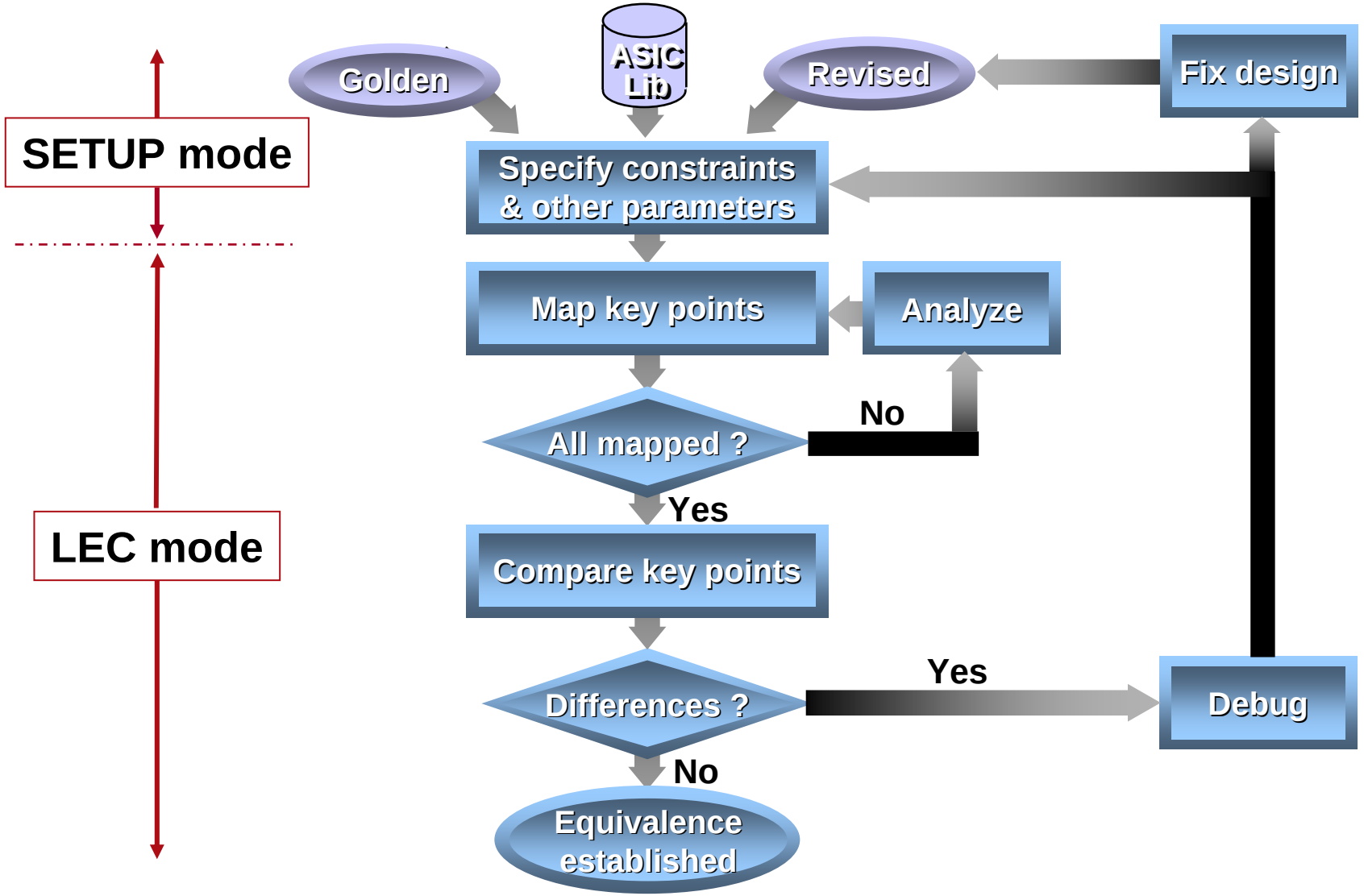
Not-Compared: Key points not yet compared

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

A Typical Debugging Session

December 19, 2004

Conformal LEC Flat Compare Flow



Flat Compare – Sample dofiles

```
set log file logfile.$LEC_VERSION –replace
setenv LIB /user1/lib/verilog/
read design –file cpu_rtl.vc -verilog –golden
read design -file verilog.vc -verilog –revised
add pin constraint 0 scan_en –revised
set flatten model –seq_constant –gated_clock
set system mode lec
add compare point –all
compare
```

Reading Mixed Languages

Mixed-languages: 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-languages: 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

- Violation messages after reading libraries/designs

Transcript window

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

- To see what a message means:

  SETUP> `help <message_name>`

  Example:

  SETUP> `help RTL2.3`

HDL Rule Checks (cont'd)

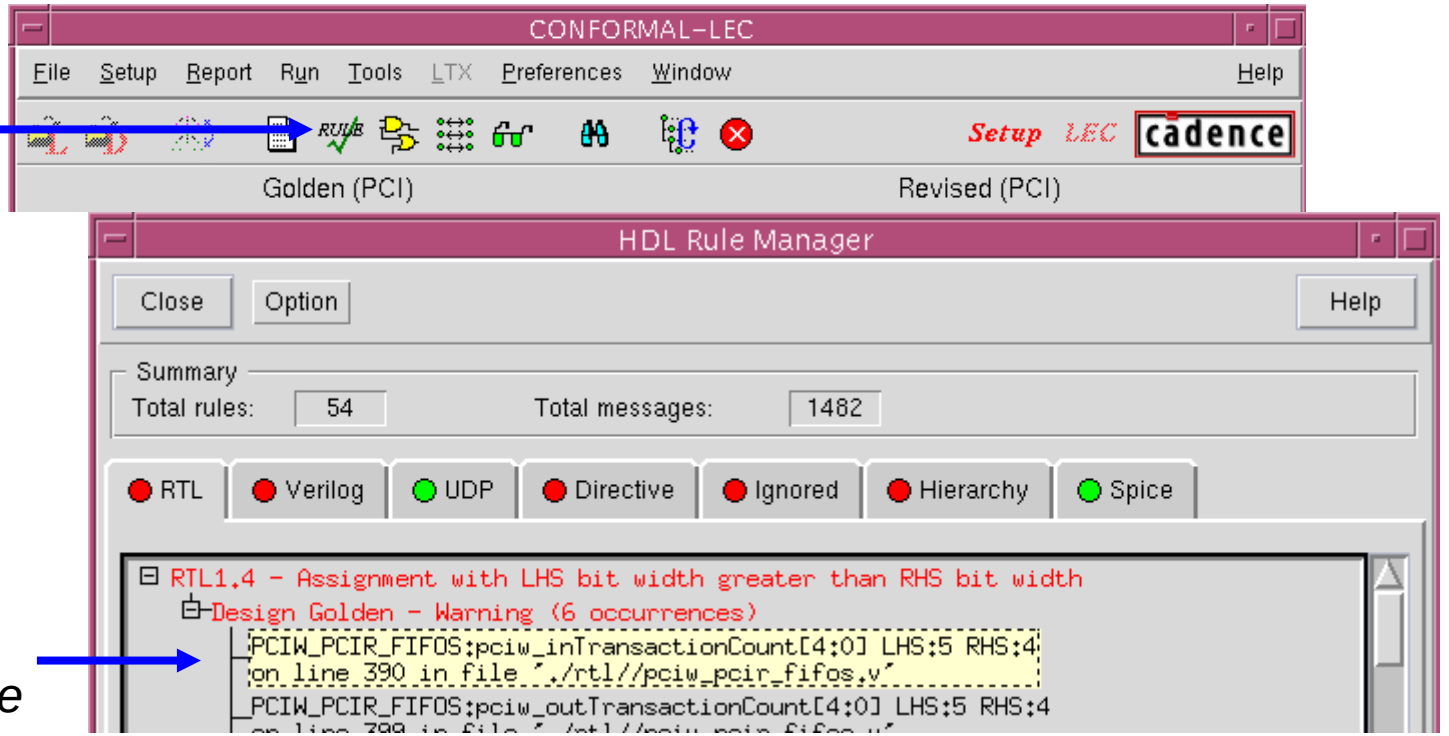
Locating the source that causes a message

- Non-GUI mode

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

- GUI mode

Click to open the HDL Rule Manager



Right click and select *Source Code*

HDL Rule Check Categories

You can specify the severity level for each category: "Error", "Warning", "Note", or "Ignore" with the **SET RUle Handling** command

- **Directives**: Synthesis and Verplex directives
- **Hierarchy**: Rules apply to hierarchical design
- **Ignored**: Redundant constructs or statements are not supported and are ignored by LEC
- **RTL**: RTL-level Verilog or VHDL rules
- **UDP**: User-defined primitives
- **Verilog**: Designs written in Verilog language

Modeling Messages

- Modeling messages are generated after you change mode to LEC.

Transcript window

```
// Command: set system mode lec
// Processing golden ...
// Modeling golden ...
// Processing revised ...
// Modeling revised ...
// Remodeled 1048 gated-clock DFF/DLAT(s)
// Converted 116 DFF/DLAT(s) to ZERO/ONE
// Converted 13 DFF/DLAT(s) to ZERO/ONE
// Warning: Golden and Revised have different numbers of key points:
// Golden  key points = 51704
// Revised key points = 51319
```

Report Modeling Messages

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

  LEC> `report message –model` *Transcript window*

```
// Command: report message –model
// Report modeling message for Golden
// F18: Converted DFF/DLAT(s) to ZERO/ONE (Occurrence: 116)
// Report modeling message for Revised
// F14: Remodeled gated-clock DFF(s) or DLAT(s) to mux-feedback
   (Occurrence: 1048)
// F18: Converted DFF/DLAT(s) to ZERO/ONE (Occurrence: 13)
```

- To see what a message means use:

  LEC> `help <message_name>`

- To display the detail of a message, i.e message **F18:**

  LEC> `report message -model -rule F18 -verbose`

Mapping Messages

- Mapping key points is automatic after modeling is done
- Example message for a successful mapping (no “Not-mapped”):

Transcript window

```
// Command: set system mode lec
... (modeling messages)
// Mapping key points ...
================================================================================
Mapped points: SYSTEM class
--------------------------------------------------------------------------------
Mapped points     PI     PO     DFF    Z      BBOX      Total
--------------------------------------------------------------------------------
Golden            221    196    50741  150    10        51318
--------------------------------------------------------------------------------
Revised           221    196    50741  150    10        51318
================================================================================
Unmapped points:
================================================================================
Golden:
--------------------------------------------------------------------------------
Unmapped points   DFF       Total
--------------------------------------------------------------------------------
Unreachable       386       386
================================================================================
Revised:
--------------------------------------------------------------------------------
Unmapped points   DFF       Total
--------------------------------------------------------------------------------
Unreachable        5         5
================================================================================
```

Mapping Messages

- Example of unsuccessful mapping:

Transcript window

```
// Command: set system mode lec
... (modeling messages)
// Mapping key points …
================================================================================
Mapped points: SYSTEM class
--------------------------------------------------------------------------------
Mapped points     PI     PO     DFF    Z      BBOX      Total
--------------------------------------------------------------------------------
Golden            221    190    50741  150    10        51312
--------------------------------------------------------------------------------
Revised           221    190    50741  150    10        51312
================================================================================
Unmapped points:
================================================================================
Golden:
--------------------------------------------------------------------------------
Unmapped points           PO     DFF       Total
--------------------------------------------------------------------------------
Unreachable               0      386       386
Not-mapped                6       0         6

================================================================================
Revised:
--------------------------------------------------------------------------------
Unmapped points           PO     DFF       Total
--------------------------------------------------------------------------------
Unreachable               5       5         5
Not-mapped                6       0         6
================================================================================
// Warning: Key point mapping is incomplete
```

Resolving Unsuccessful Mapping

- Not-mapped key points are not compared and cause NEQ (not unreachable and extra unmapped types)
- Add renaming rules to map by name

```
add renaming rule <rule_name> <string> <string> [-Golden | -Revised]
```

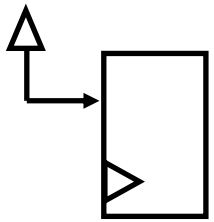
- Sequential Constant modeling
- Black box merging

Sequential Constant Options

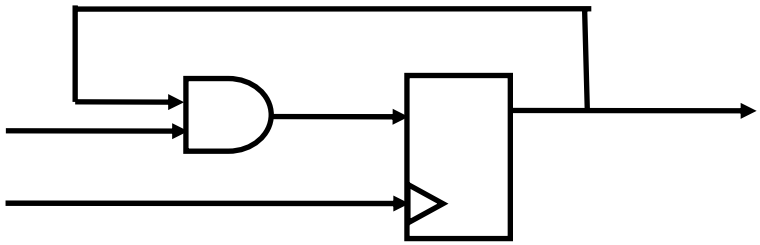
Model sequential constant registers to constant

Sequential constant options :

- -seq_constant



- -seq_constant_feedback

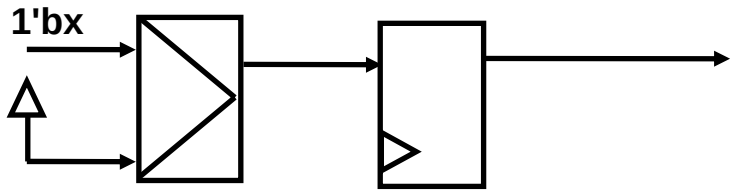


Once register is 0, it will always be 0

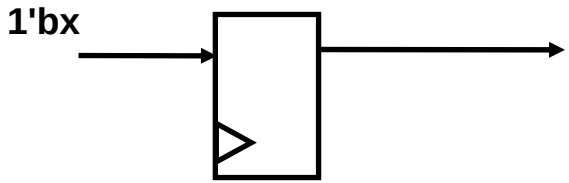
Sequential Constant Options(cont)

Sequential constant options :

- -seq_const_dc



- -seq_constant_x_to [0|1]



Remodel

- Designed to resolve mis-compares due to keypoint issues
- Performed in LEC mode
- Applies to unmapped points only
- Can takes a set of key points and attempts to remodel them
- Only works if number of unmapped points is less than the MAX_UNMAP limit
- Default is 5,000 unmapped points
  - Use remodel –max_unmap <N> to change limit
  - Useful to skip remodel until mapping is fairly complete

Sequential constant optimization flow

Example Script1

```
set flatten model –seq_constant –seq_constant_x_to 0
set system mode lec
remodel –seq_constant –seq_constant_feedback –repeat
map key point
```

Example Script2

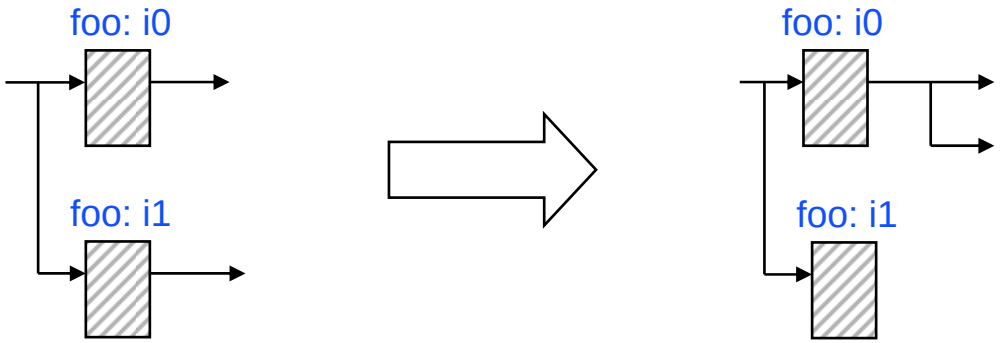
```
set flatten model–seq_constant_x_to 0
set mapping method –name only
set system mode lec
remodel –seq_constant –seq_constant_feedback –repeat
set mapping method –name first
map key point
```

Requirement :

- Good name mapping
  - Might need to use "add renaming rule"

Black Box Merging

set flatten model -bbox_merge



- Same black box module
  - Can use module renaming rules
- Same inputs (simple buffering, constants allowed)
- Equivalent inputs not handled yet
- Modeling message F51 printed

Compare Results

A successful compare result

Transcript window

```
// Command: compare
================================================================
Compared points     PO     DFF    Z      BBOX      Total
----------------------------------------------------------------
Equivalent          196    50741  150    10        51097
----------------------------------------------------------------
Inversed-Equivalent   0       41   0      0        41
================================================================
```

A compare result with NEQ failure

Transcript window

```
// Command: compare
================================================================
Compared points     PO     DFF    Z      BBOX      Total
----------------------------------------------------------------
Equivalent          190    50741  150    0         51081
----------------------------------------------------------------
Non-equivalent       6      0      0     10         16

================================================================
```

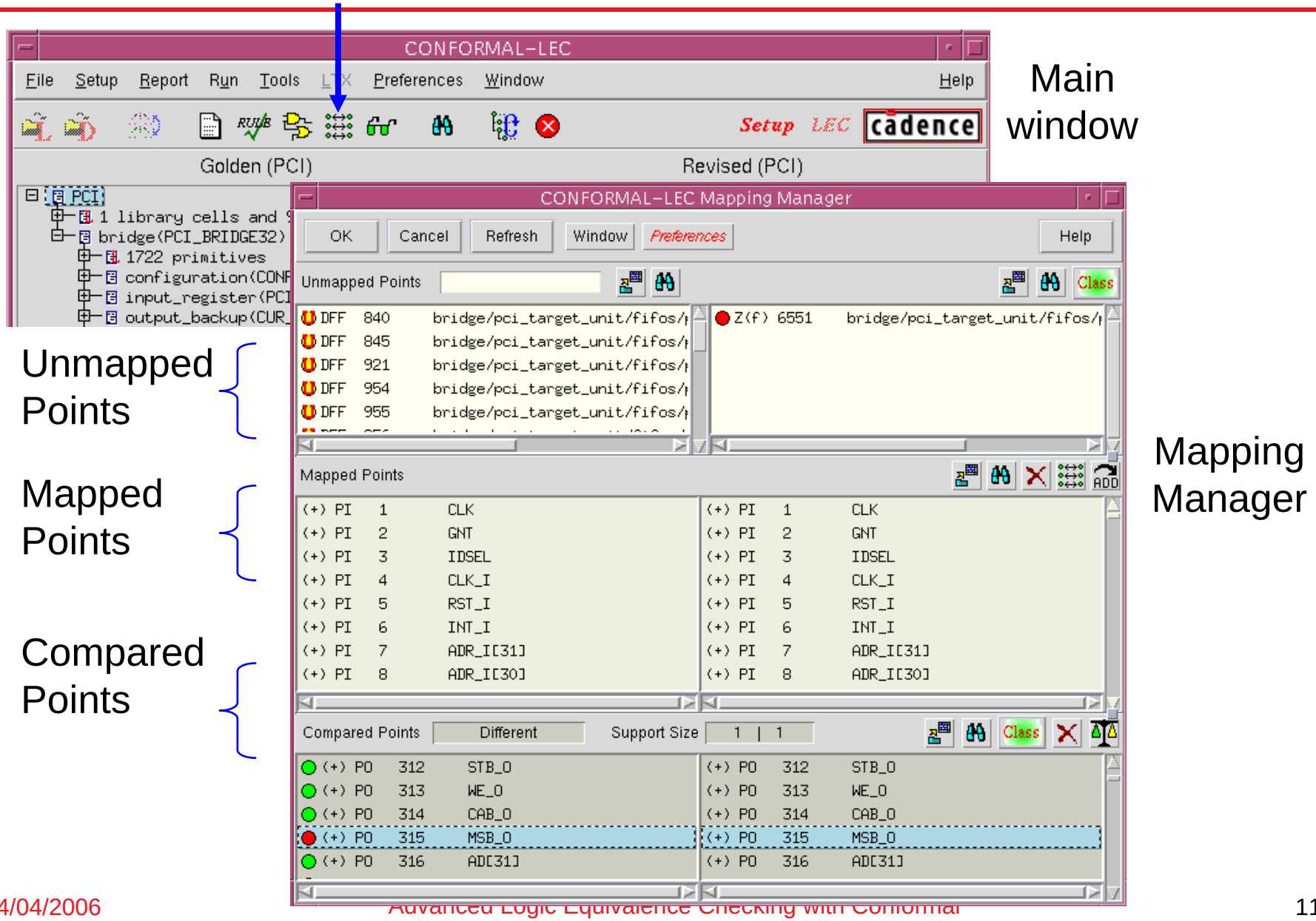
How Do I Debug NEQs?

Use the diagnosis managers below to debug:

- Mapping Managers
- Diagnosis Manager
- Schematic Viewer
- Hierarchical Browser
- Source Code Manager
- Gate Manager

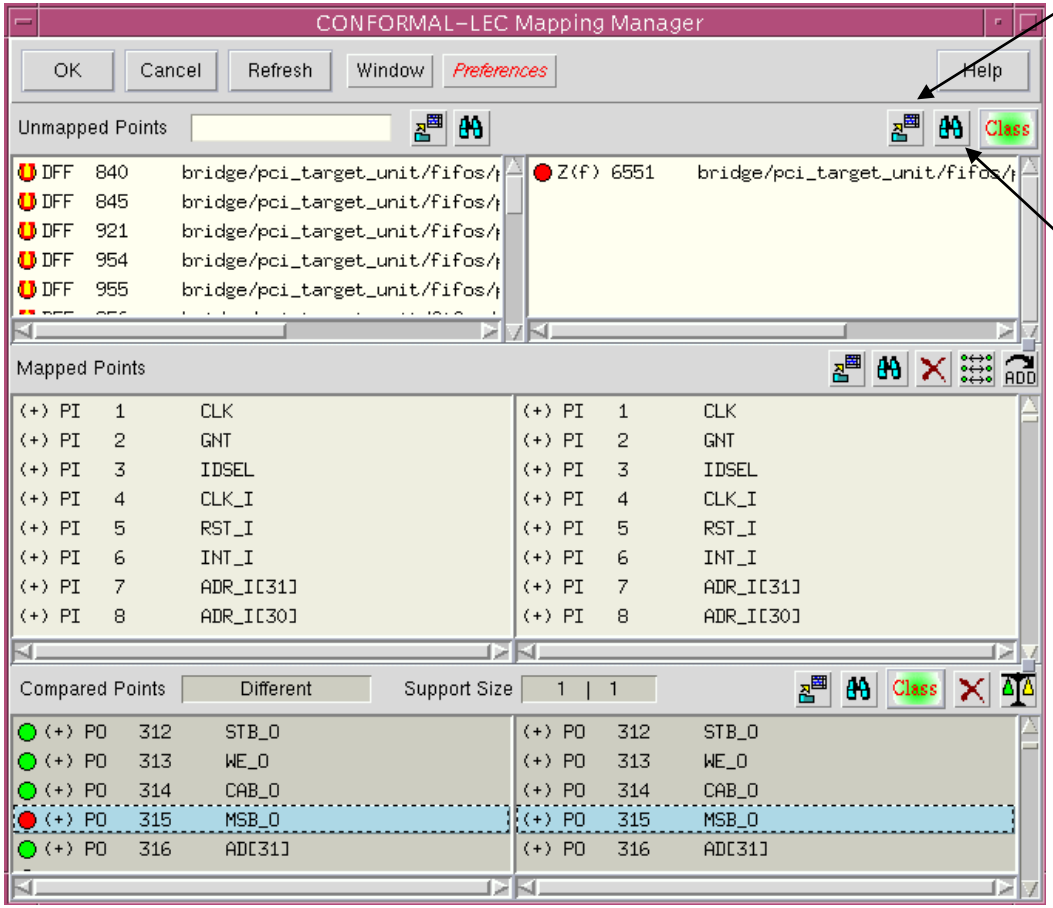
Diagnosis managers are integrated.

Mapping Manager



Mapping Manager (continued)

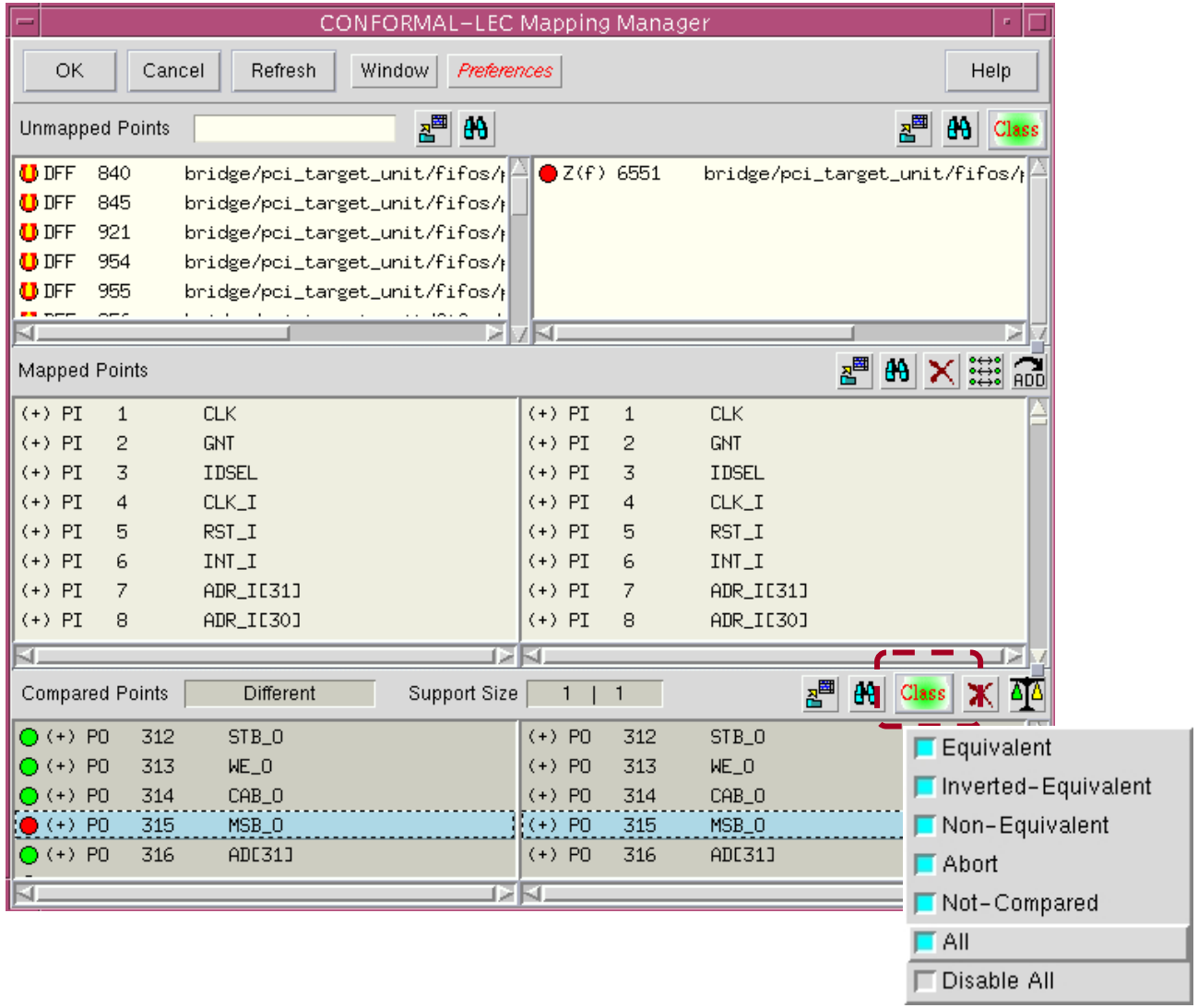
Find/filter key points



- Filter key point.
  - Example: *dst_eg*
- Find key points with partial name or ID

Mapping Manager (continued)

Display only non-equivalent results

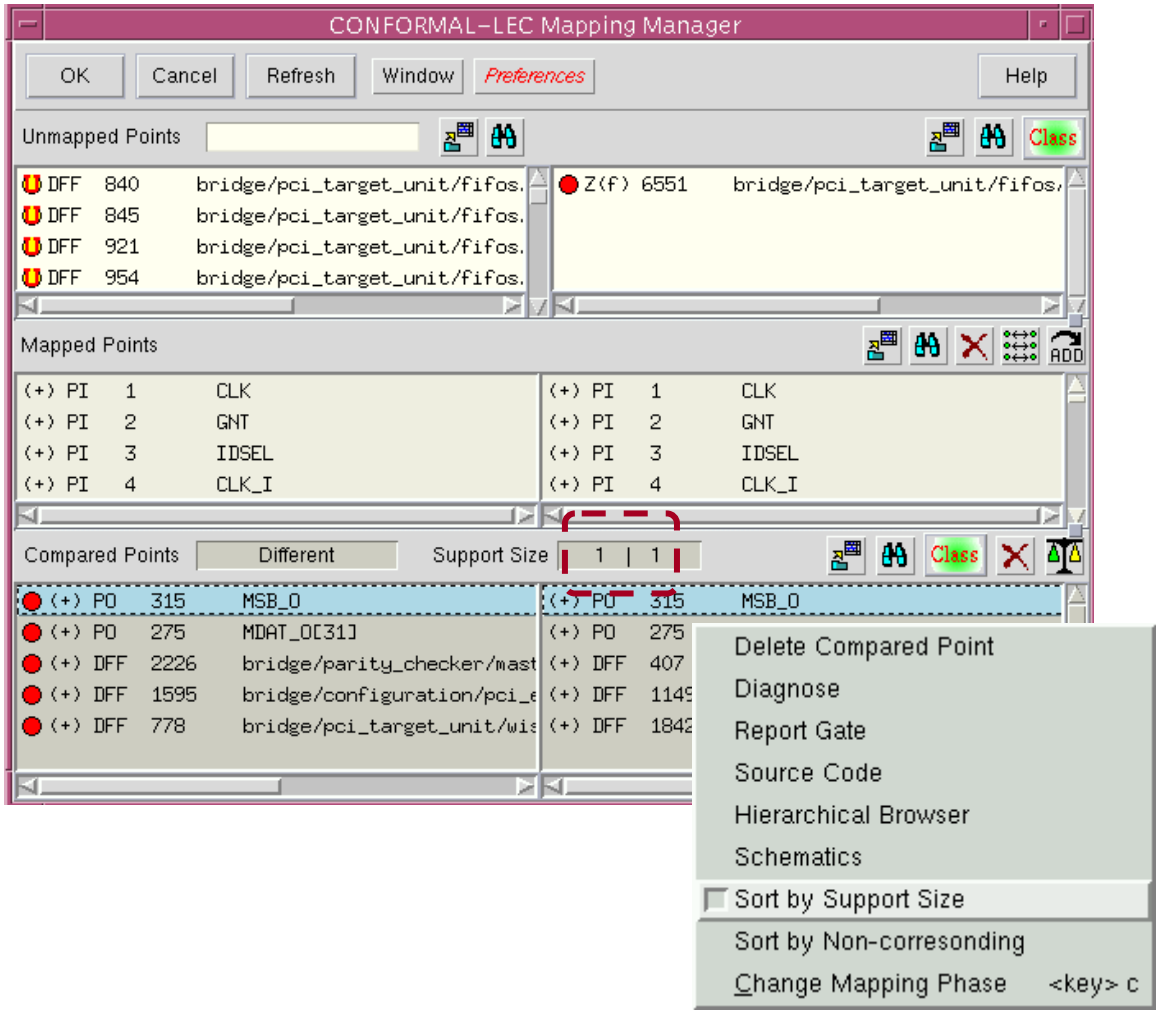


Select:

1. Disable All

2. Non-equivalent

Mapping Manager (continued)

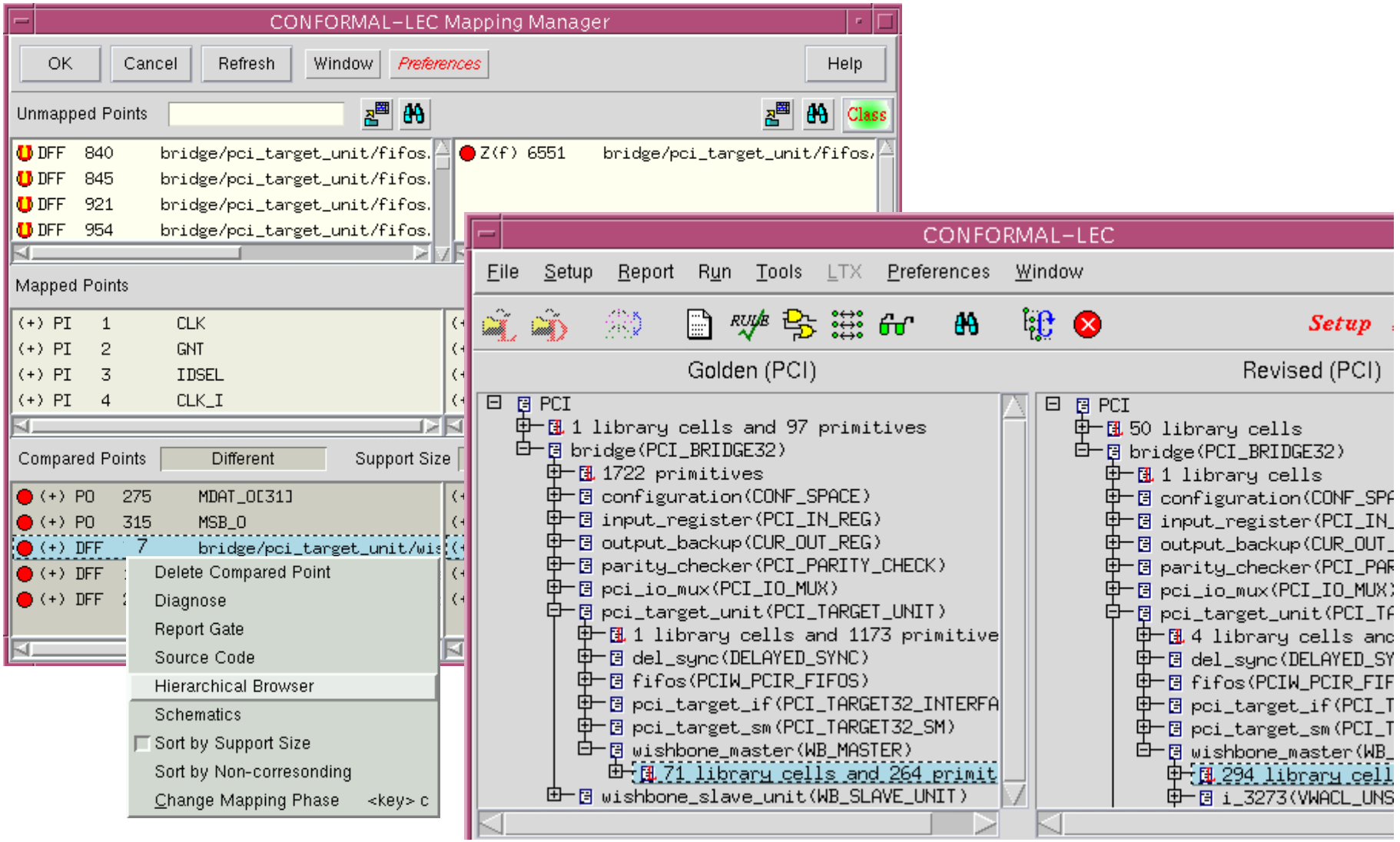


- Sort key points by size to diagnose smaller point first
- Smaller size on top of the list

1. **Left click on a compared point**
2. **Right click**
3. **Sort by …**

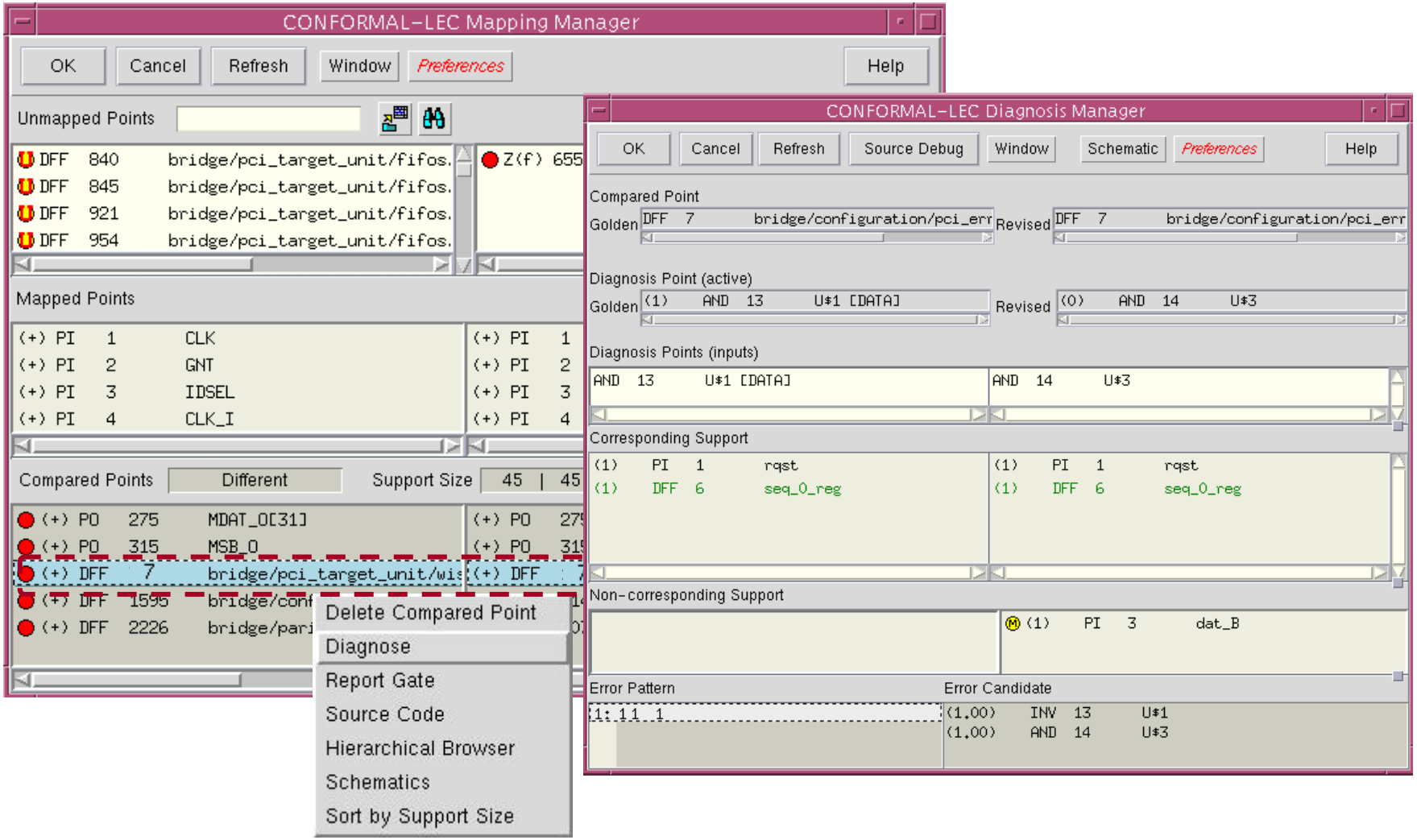
Mapping Manager

Link to the Hierarchical Browser

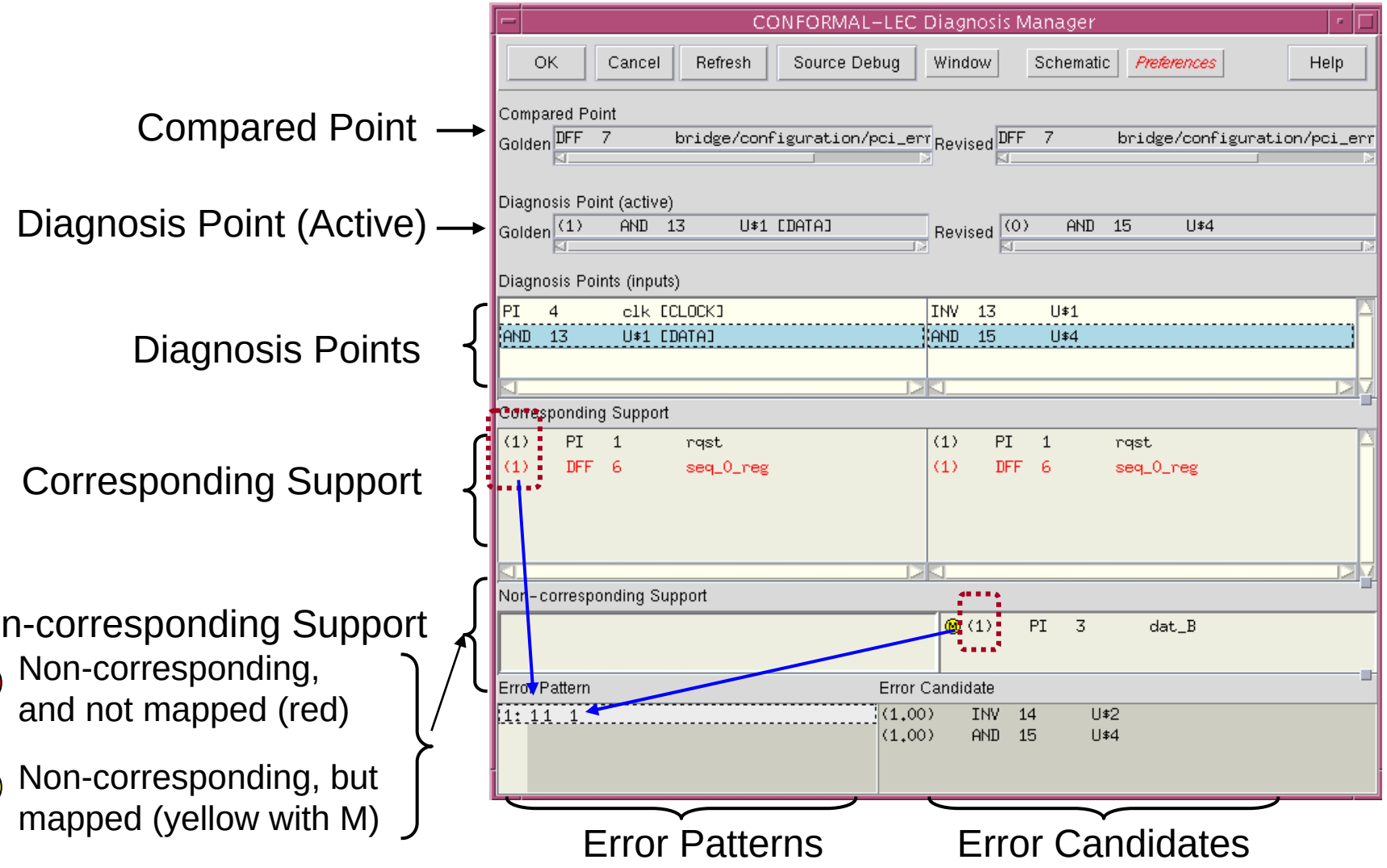


Mapping Manager

Invoke the Diagnosis Manager to debug



Diagnosis Manager



Diagnosis Information

Compared Point: Non-equivalent compared point

Diagnosis Point (Active): Point at which diagnosis failed. Simulation value is shown in parenthesis ( )

Diagnosis Point (Inputs): Lists all the fanin diagnosis points for the compare point

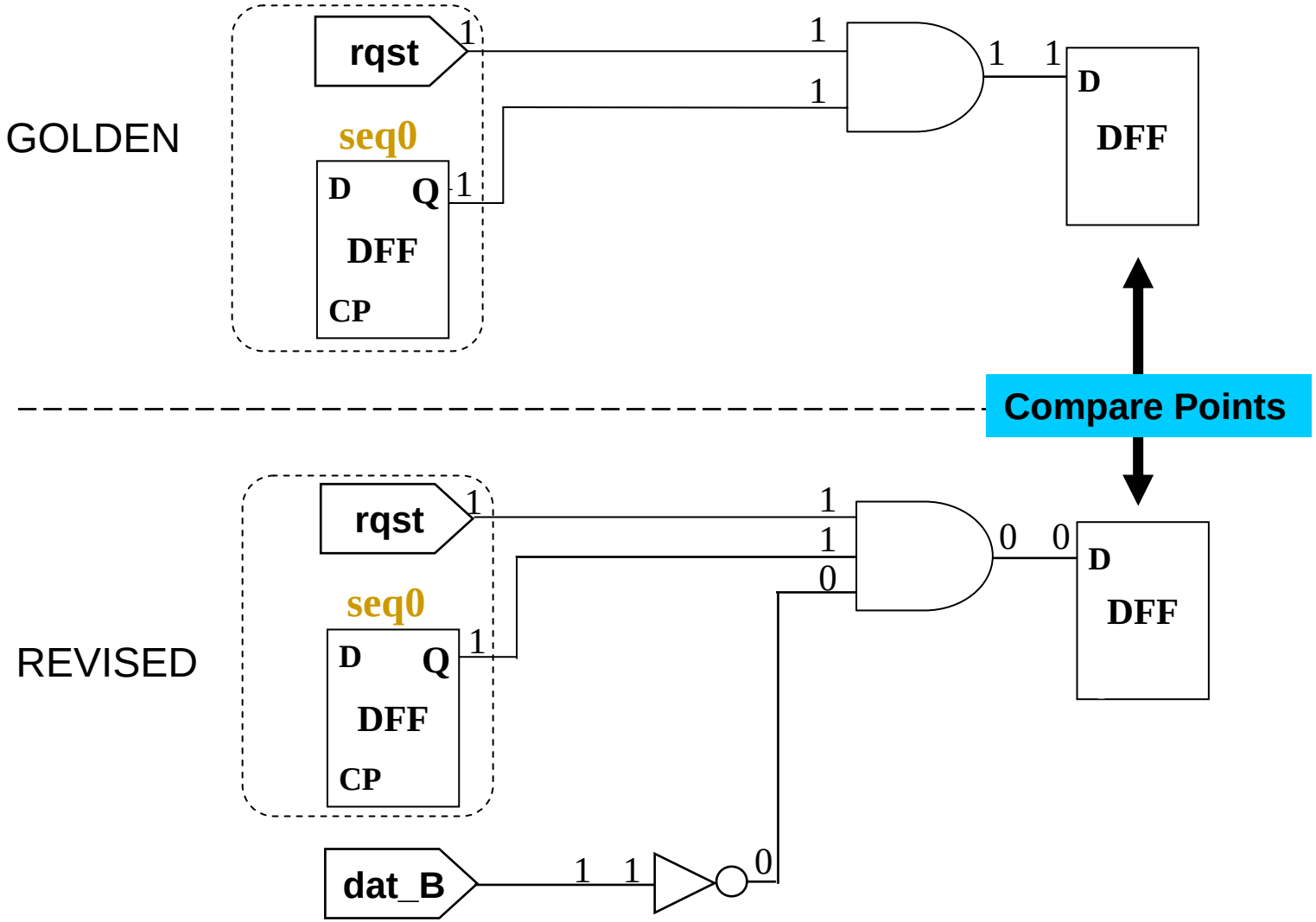
Corresponding Support: Displays the mapped points that are in the fanin cone of the diagnosis point of both the Golden and Revised designs. Simulation values of the corresponding support points are shown along with the key point names

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

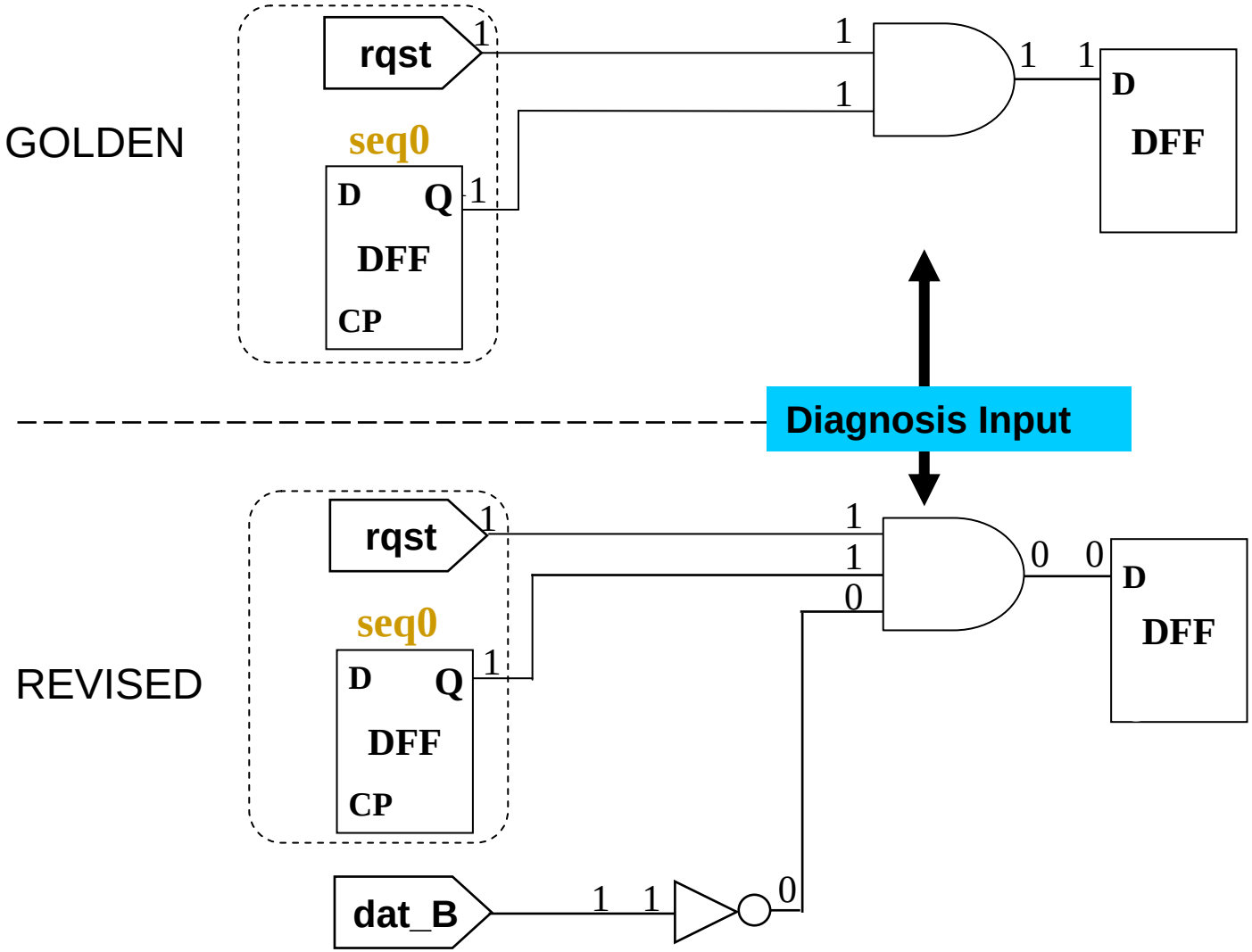
Error Pattern: Test vector proving the diagnosis point to be non-equivalent

Error Candidate: Gates in the Revised designs with highest probability of causing non-equivalence

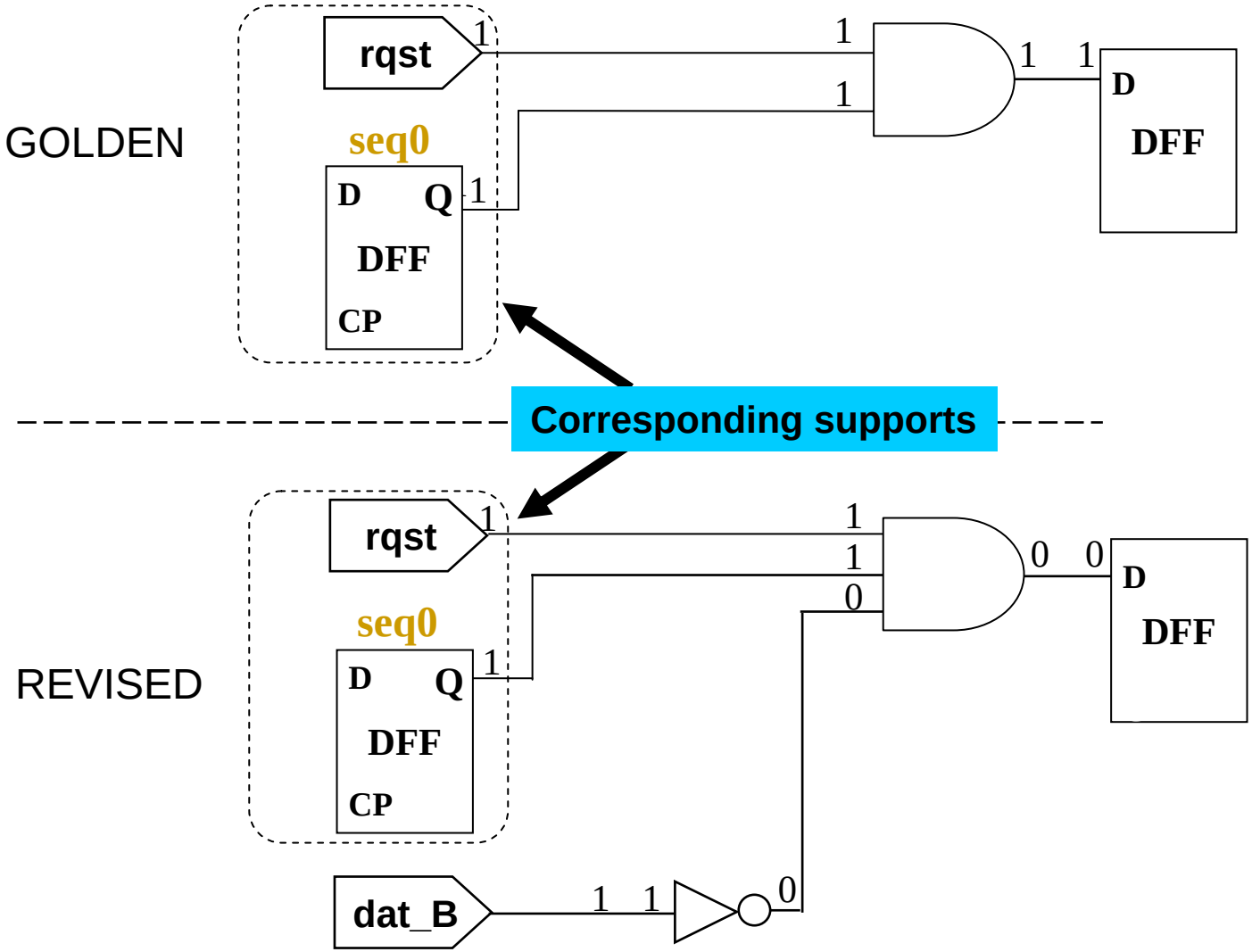
Diagnosis Information



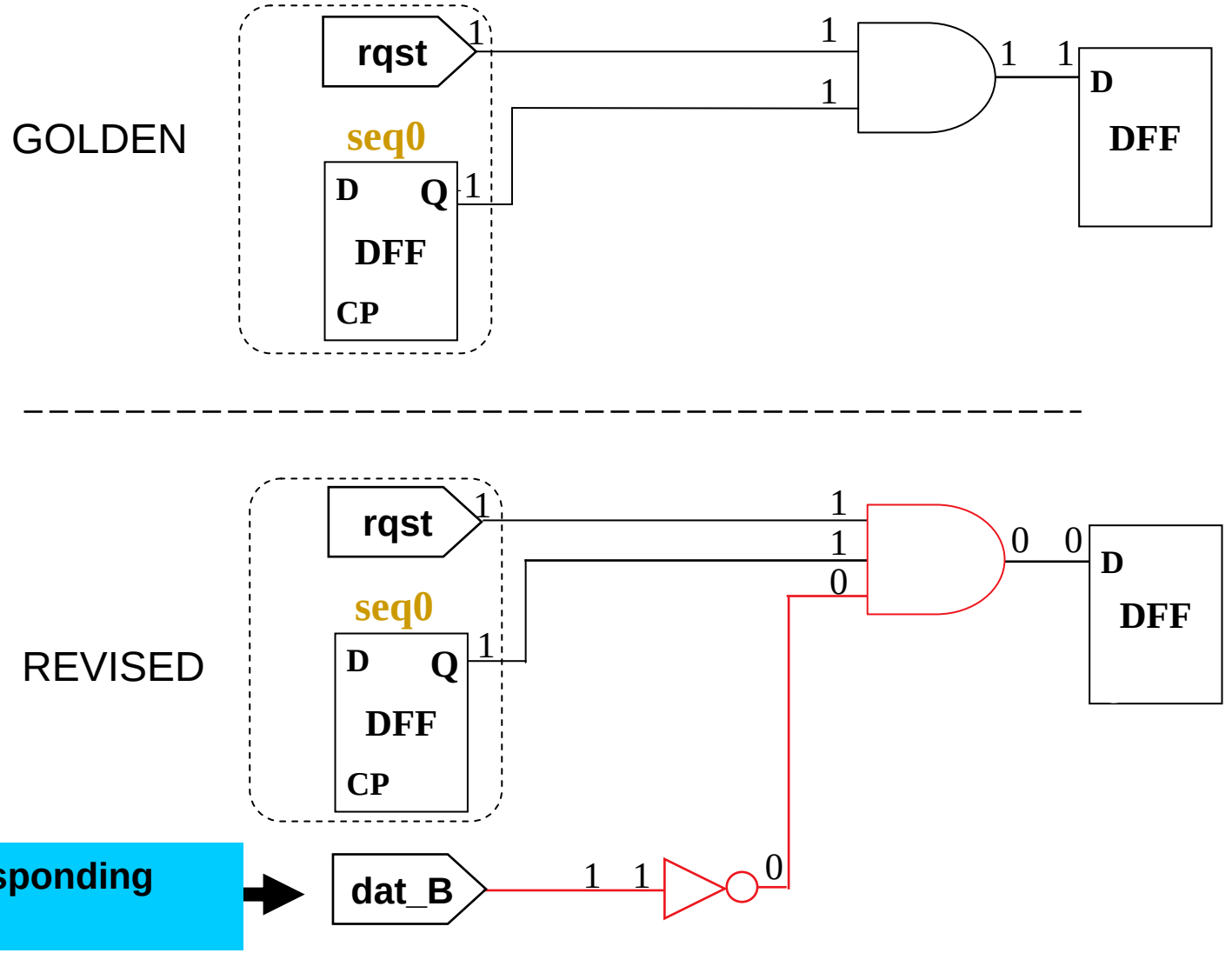
Diagnosis Information



Diagnosis Information



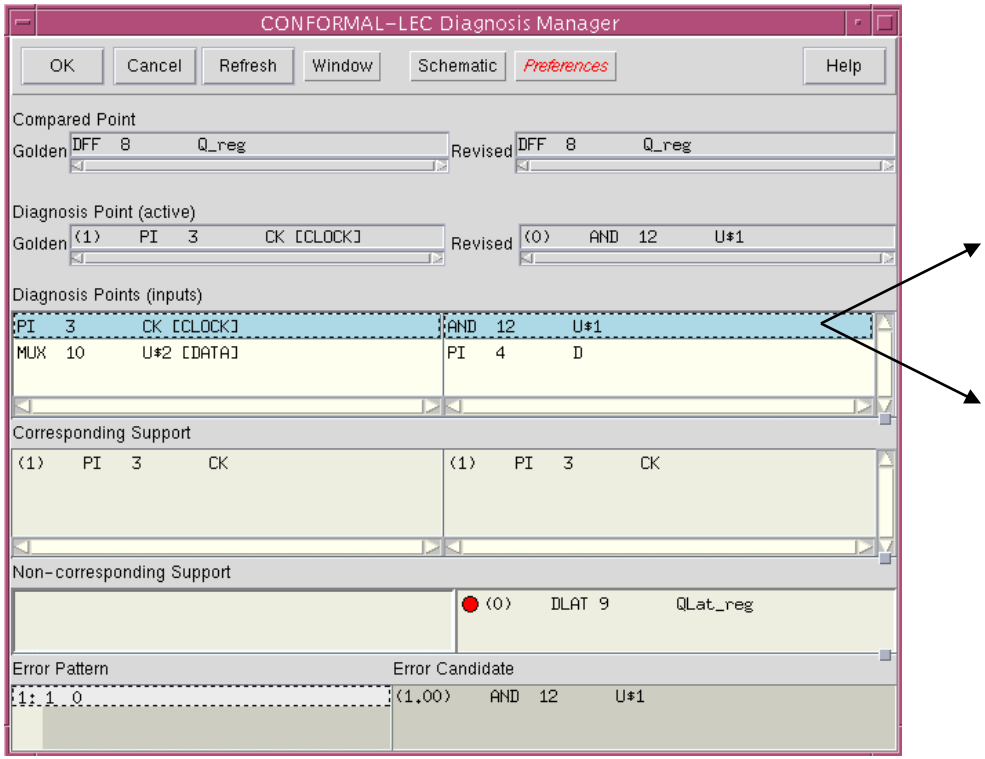
Diagnosis Information

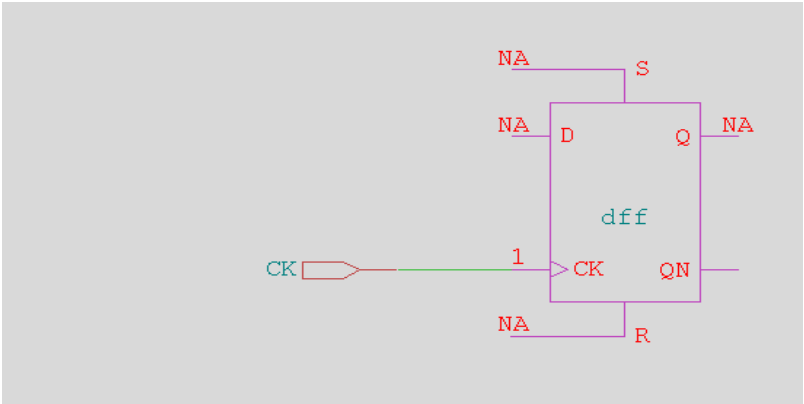


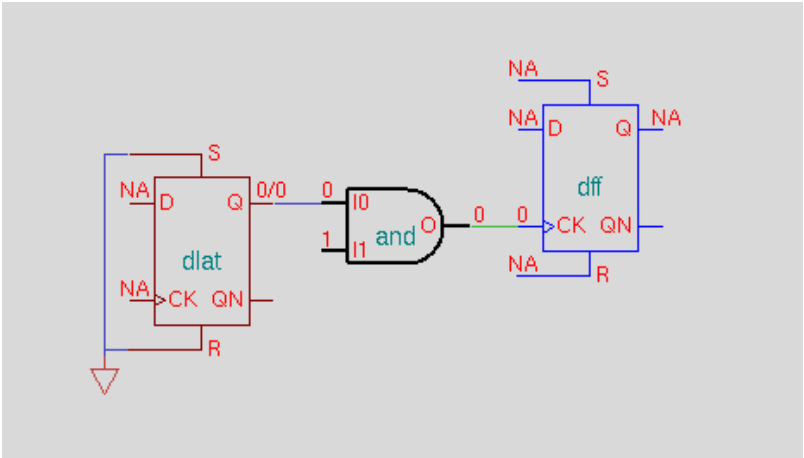
Diagnosis Inputs

A key point might have more than one input points being NEQ

Example: clock cone

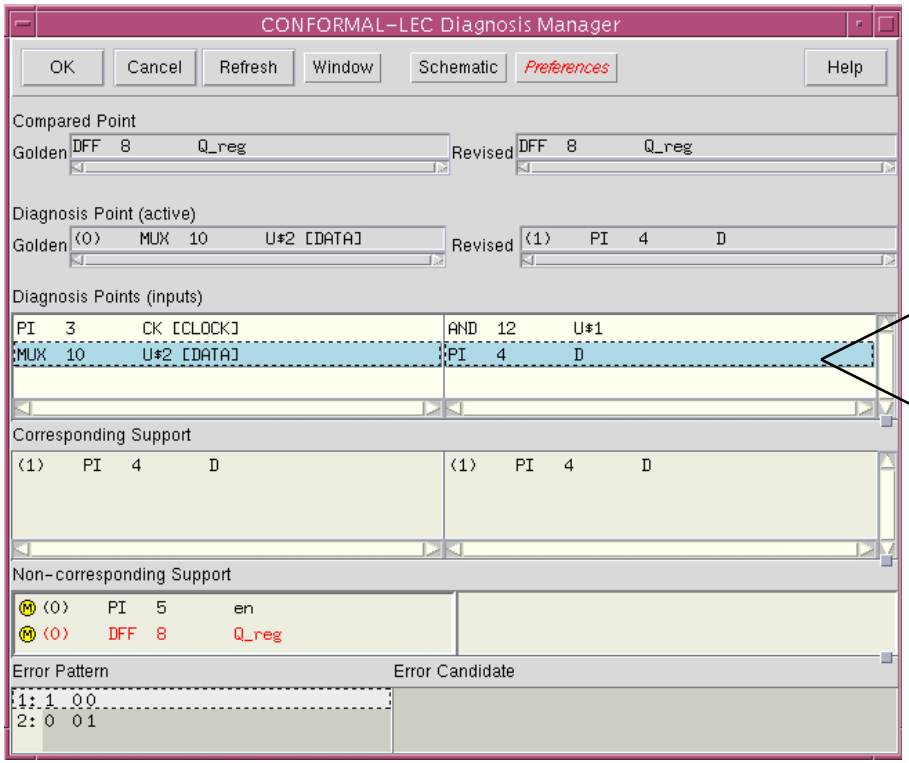


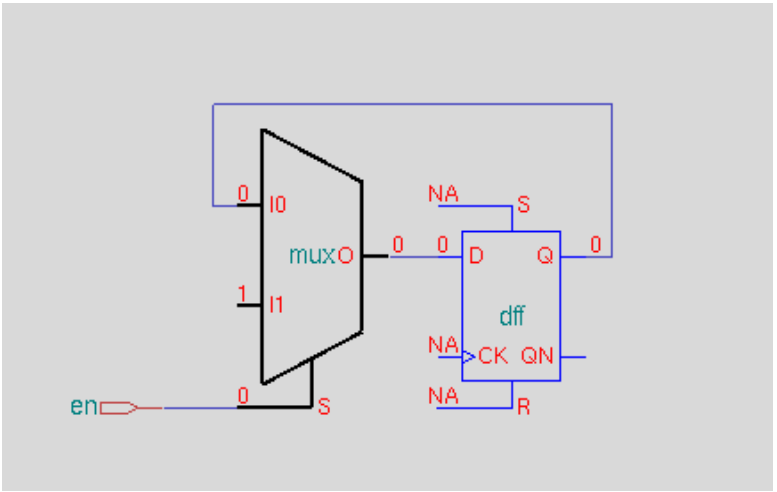


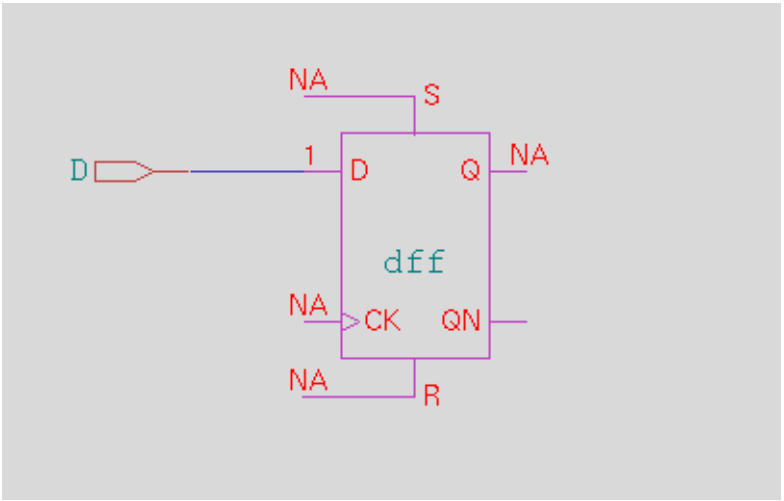


Diagnosis Inputs

Example: data cone

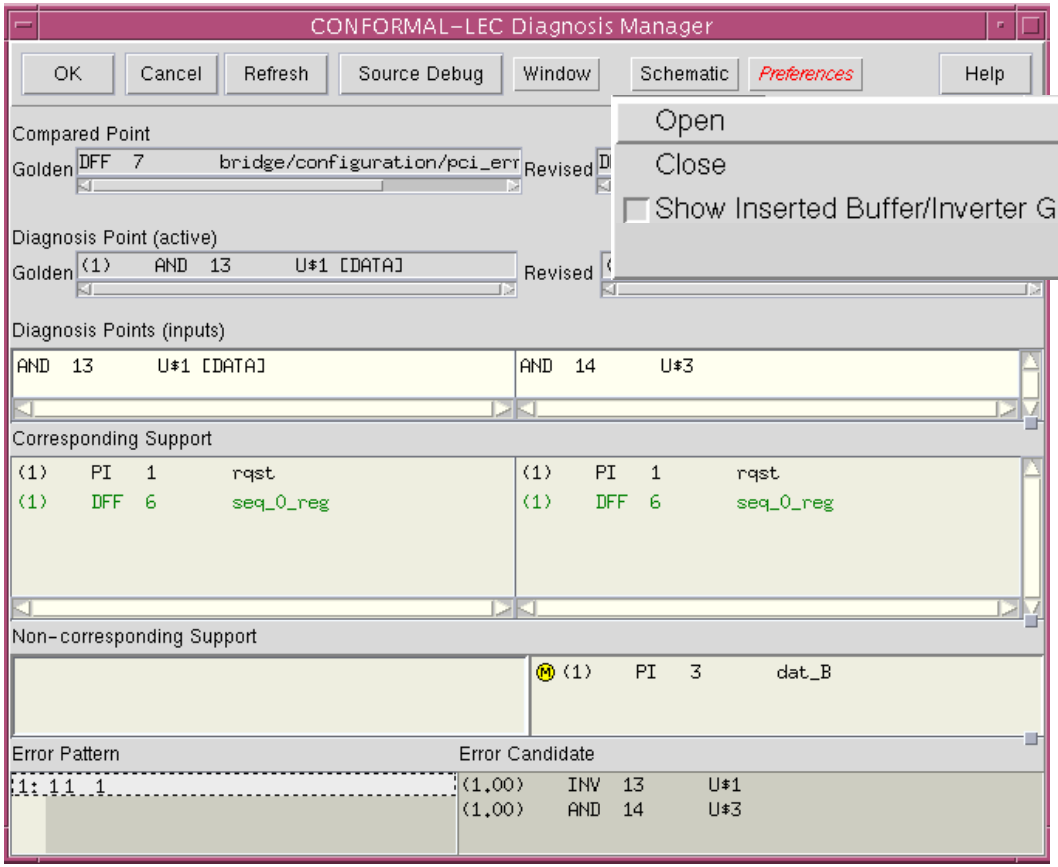


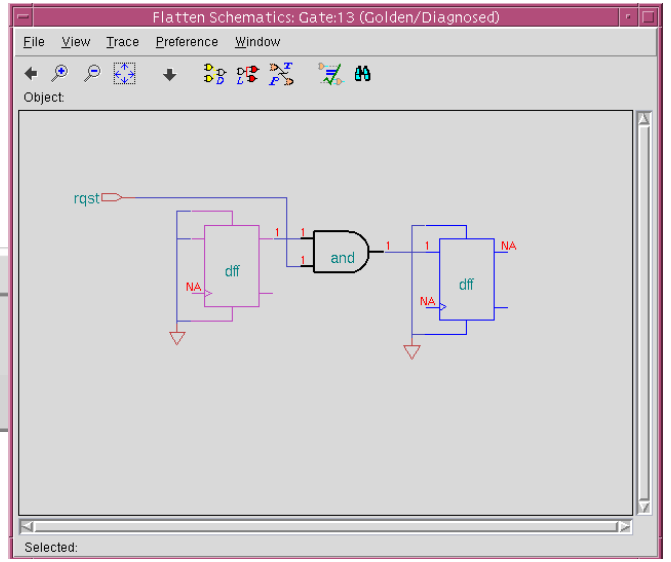


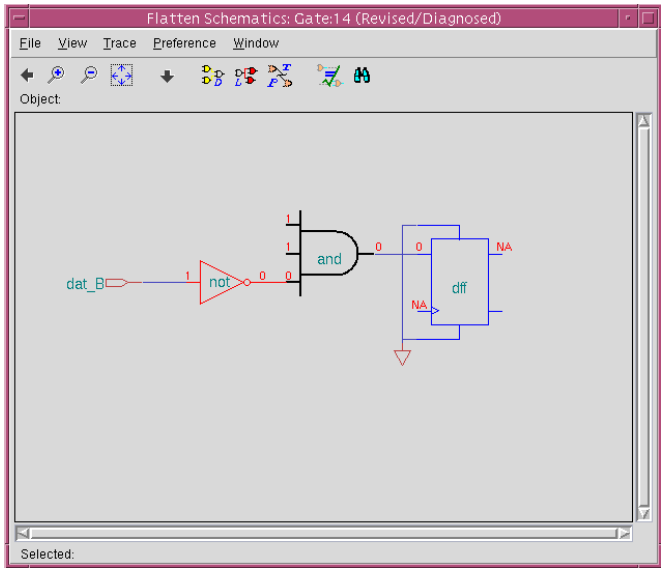


Schematic Viewer

Invoke from Diagnosis Manager

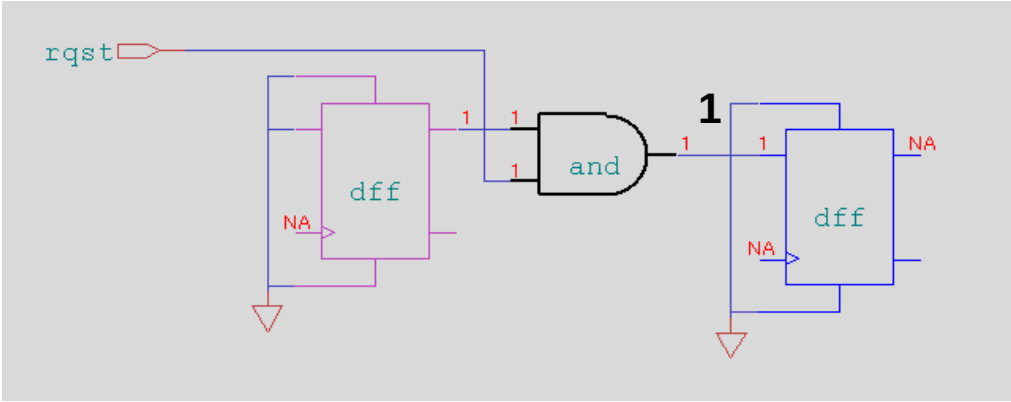






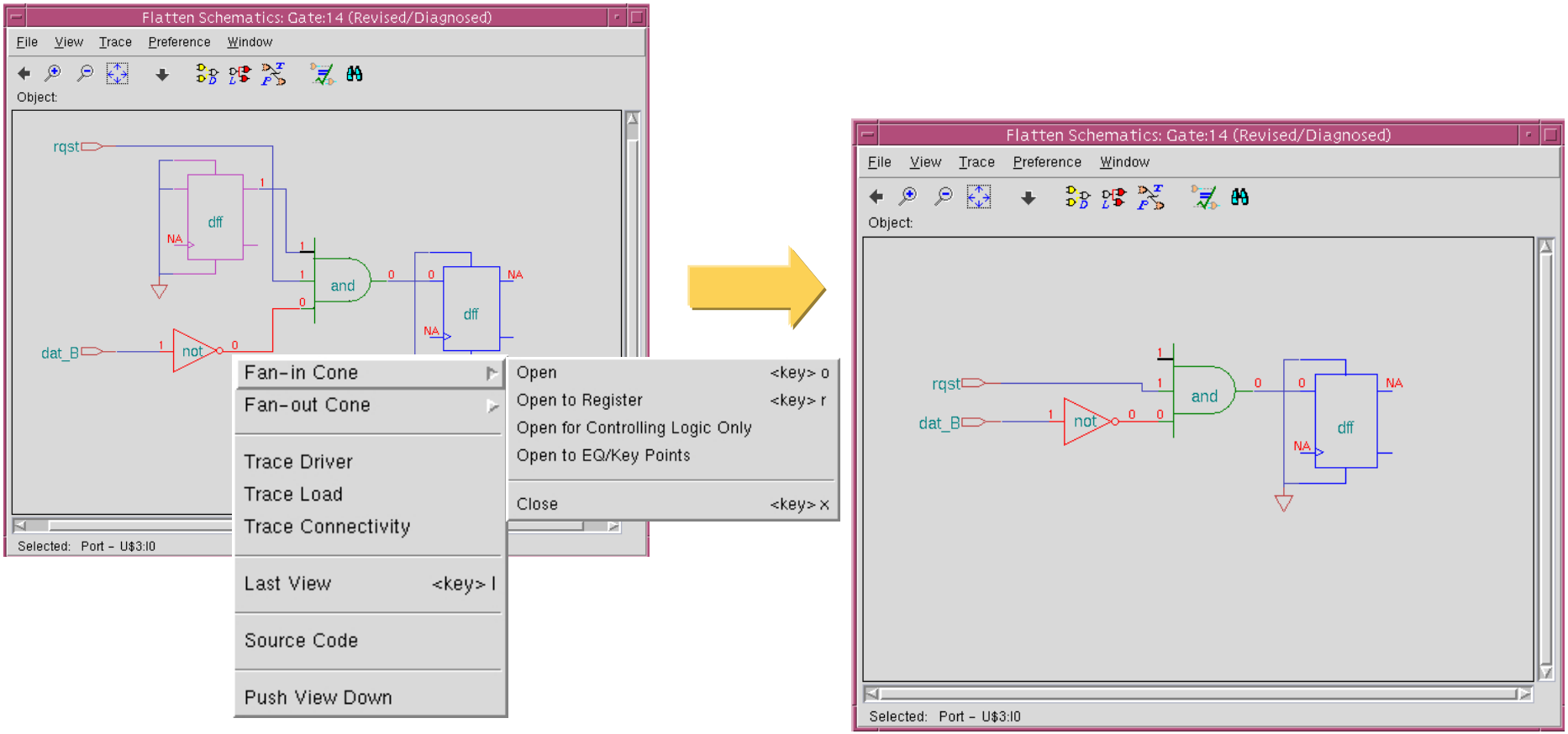
Schematic Viewer

By default, cones are trimmed to show non-equivalent (simulation mismatch at diagnosis input)



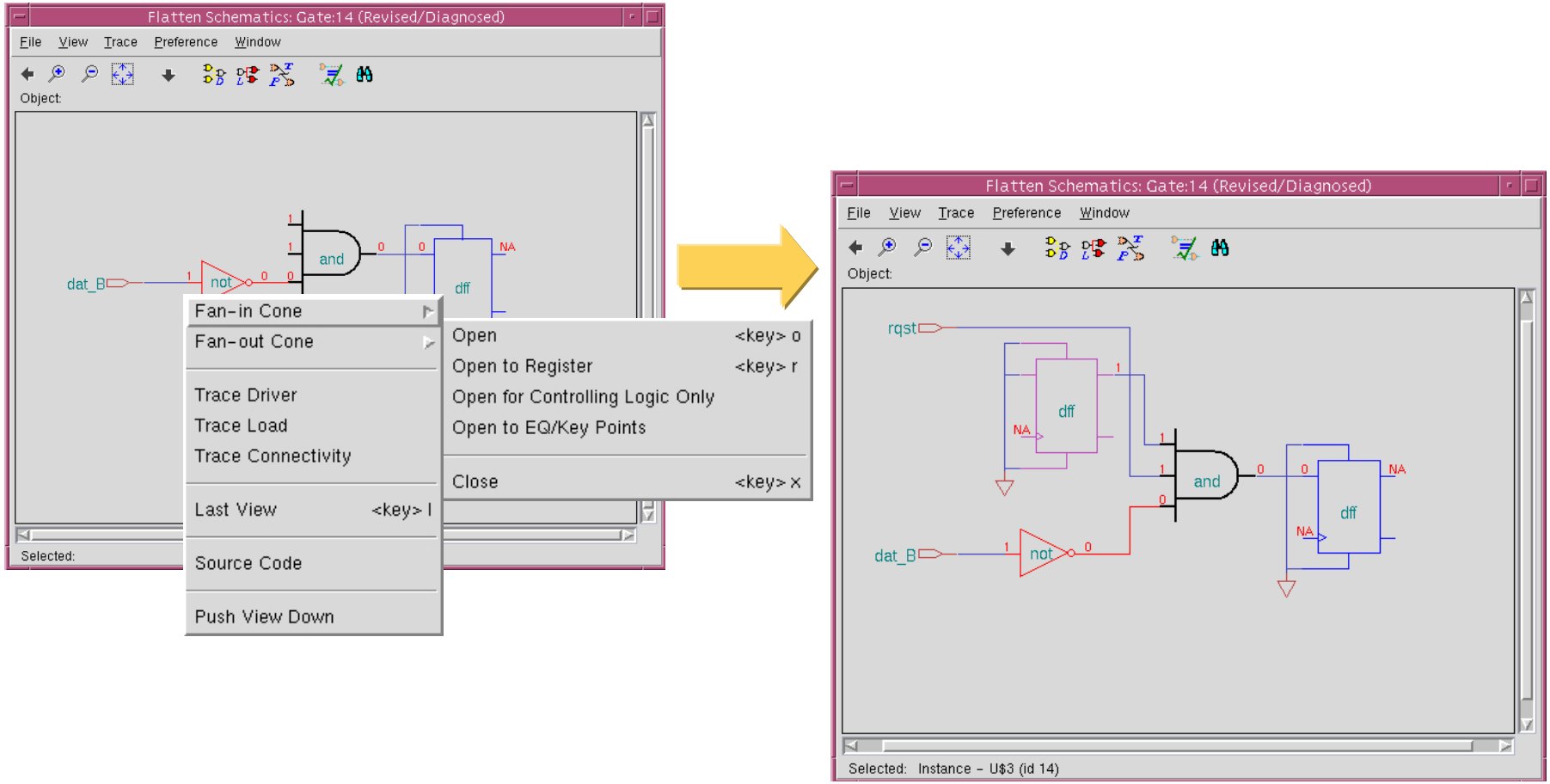
Schematic Viewer: Trimming Fanin Gates

- Select gate to trim all (or a net) fanin
- Short cut = ‘x’ on keyboard

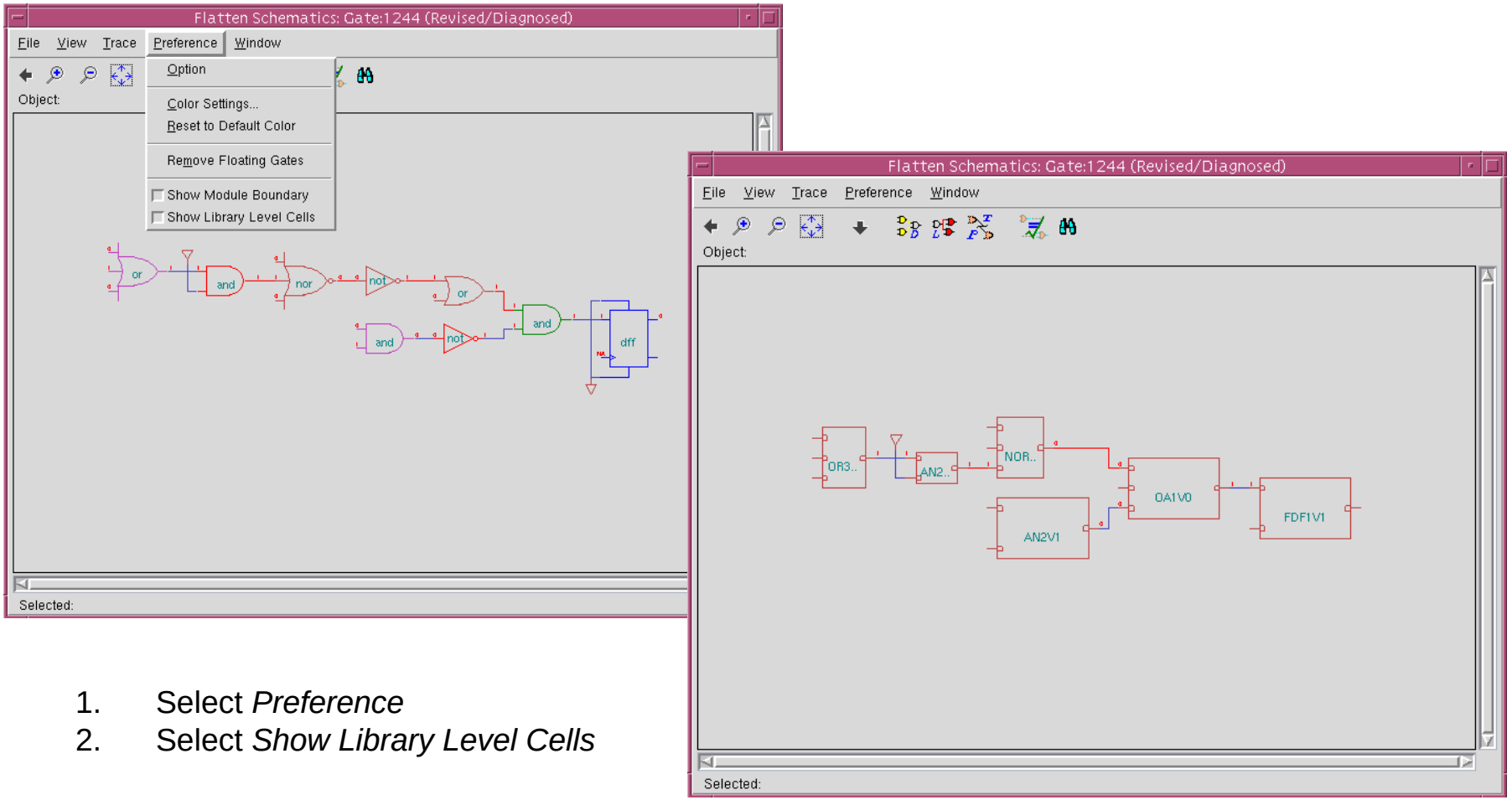


Schematic Viewer: Expanding Fanin Gates

- Select gate to expand all (or a net) fanin
- Short cut = ‘o’ or ‘r’ on keyboard



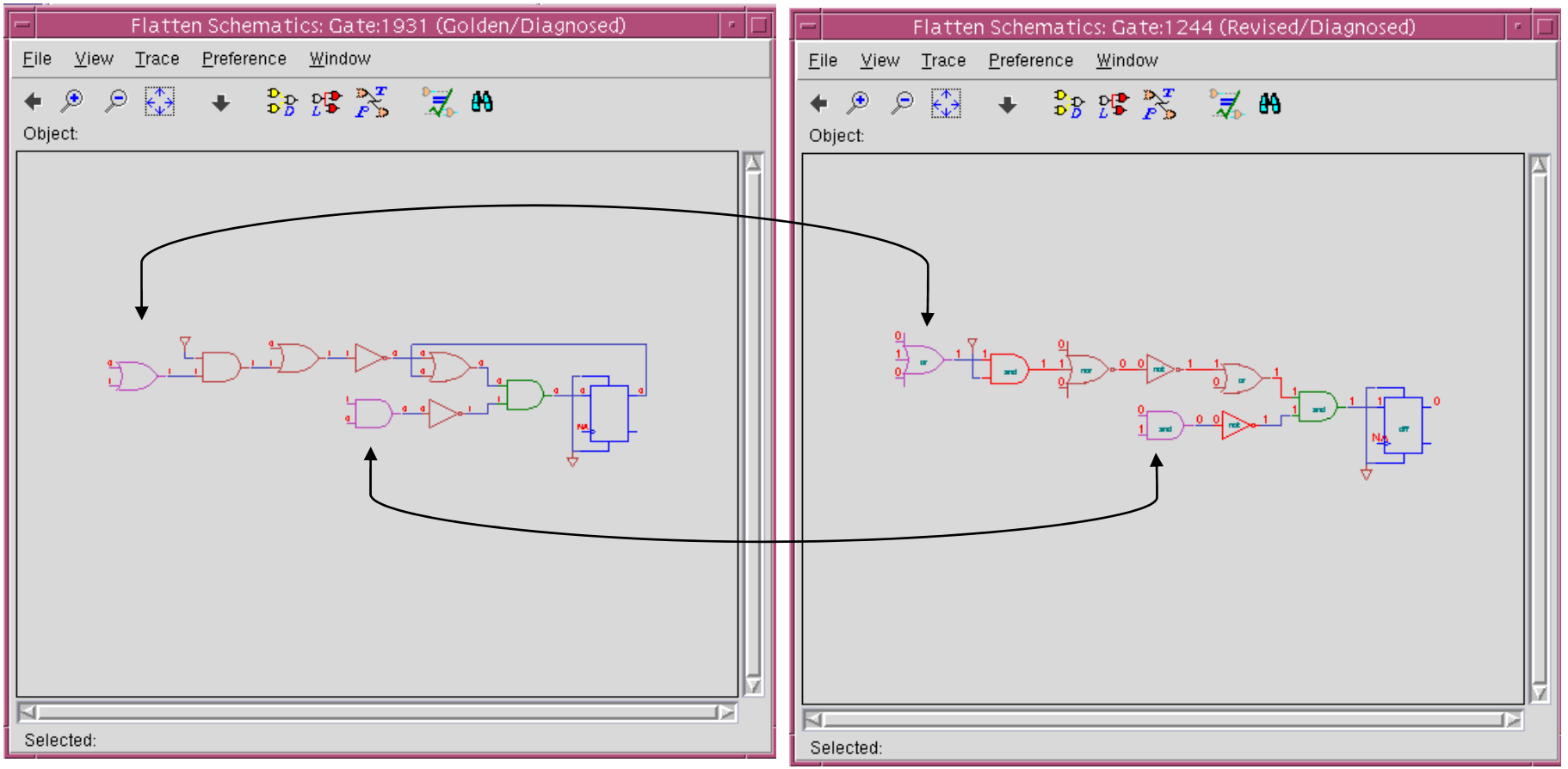
Show Library Boundary



1. Select *Preference*
2. Select *Show Library Level Cells*

Purple Gates

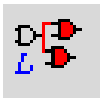
Purple gates are intermediate equivalent points



Schematics Tool Bar

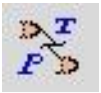
Last View: Return to the previous schematic view

Trace Driver: Highlight the drivers in yellow



Trace Load: Highlight the loads in red

Zoom to Full: Display all the contents of the schematic



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

Schematic Viewer

Previous slides showed

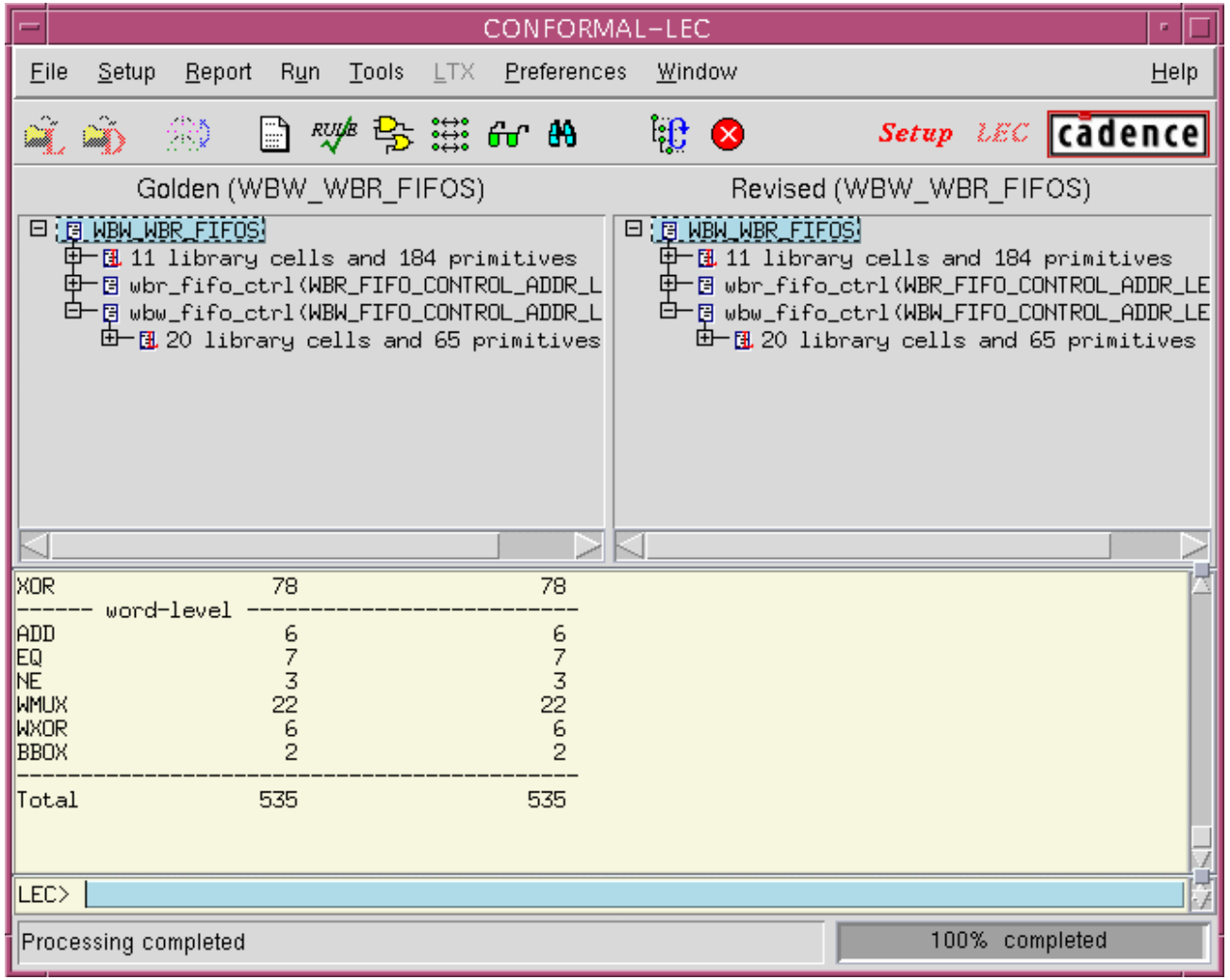
- Flattened schematic
  - Schematic after flattening/modeling applied
  - Opened from Mapping/Diagnosis/Gate Managers

Next …

- Hierarchical schematic
  - Before flattening/modeling applied
  - Opened from Hierarchical Browser

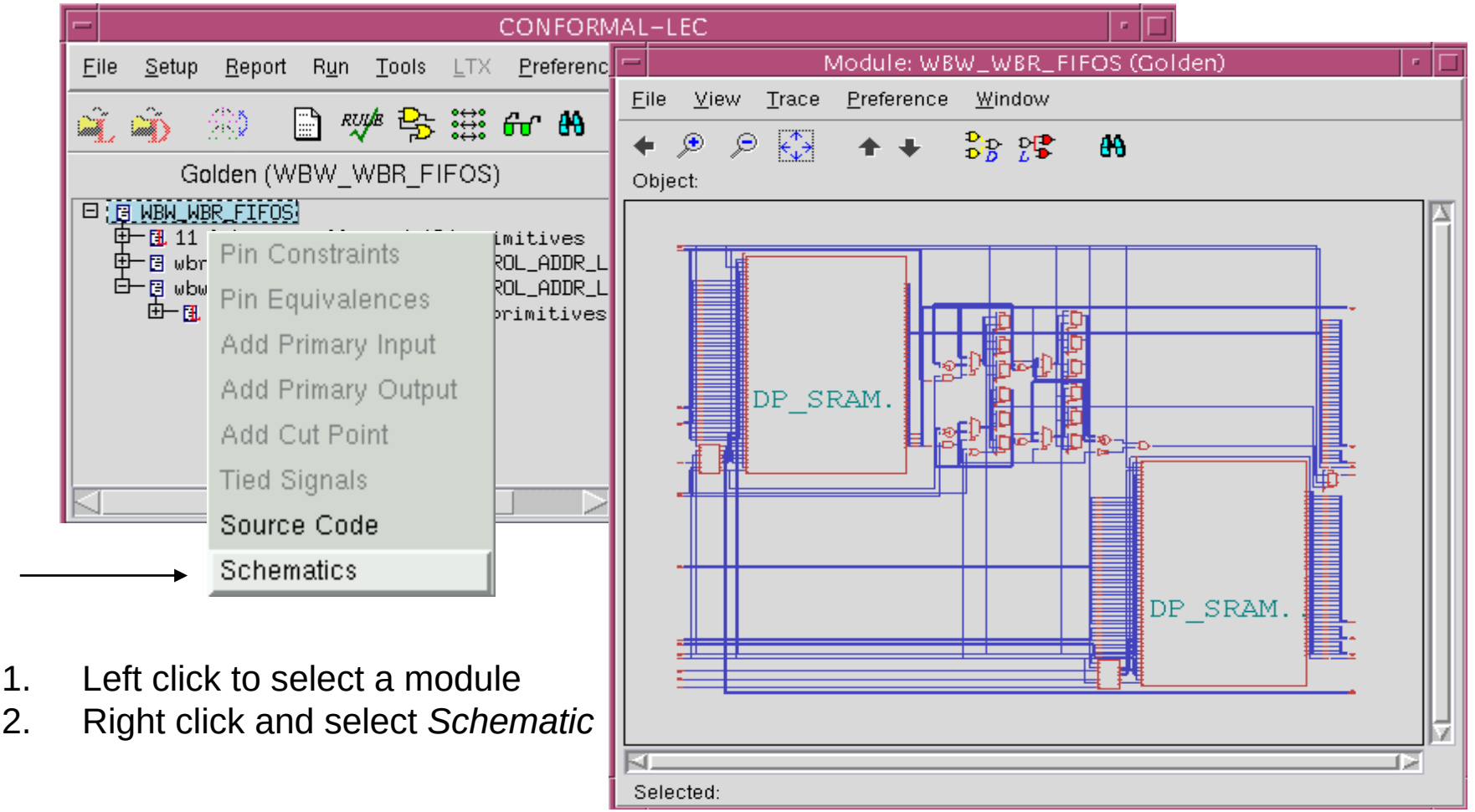
Hierarchical Browser

Show design hierarchy and allow access to any module



Hierarchical Schematic

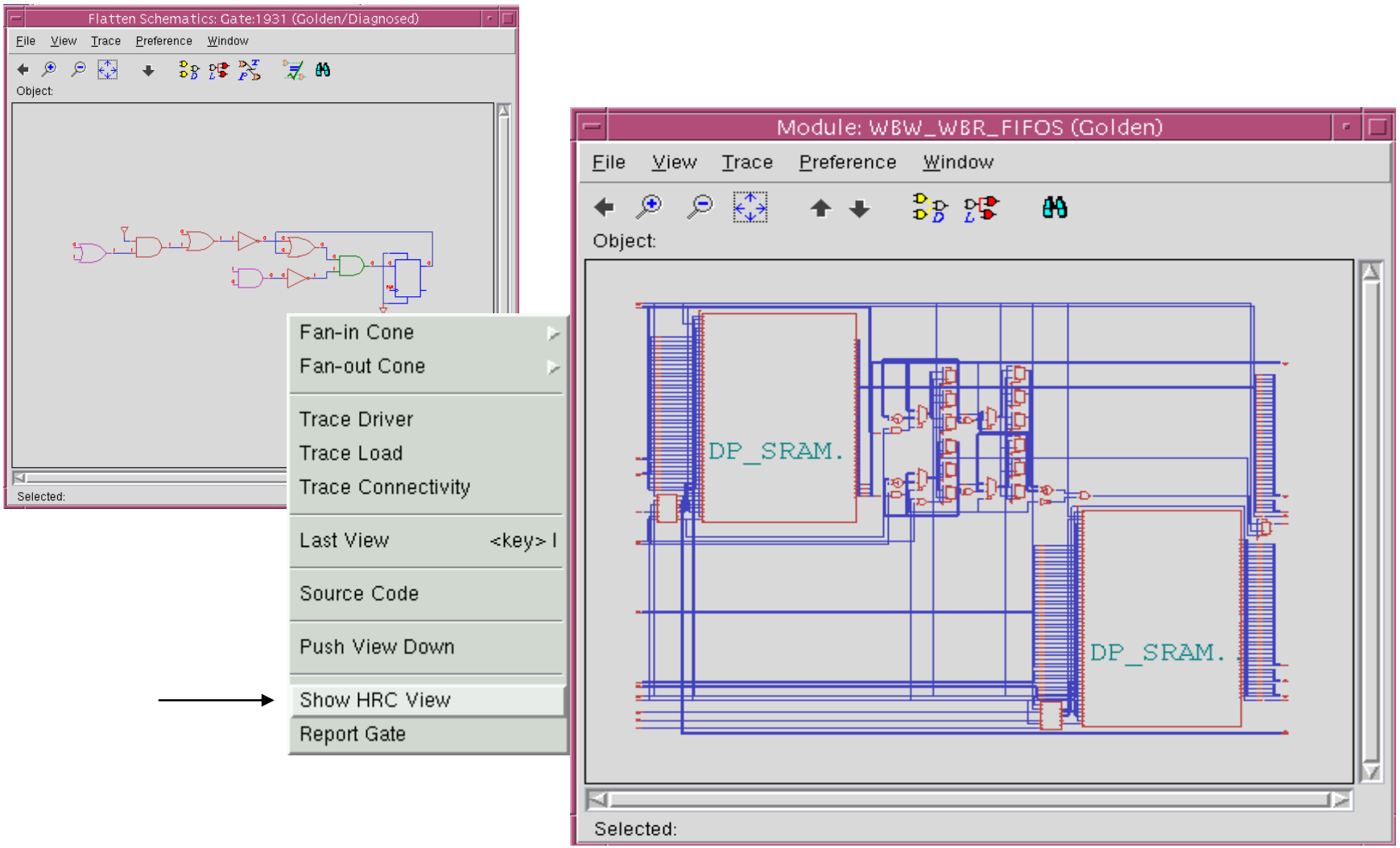
- Can be open from Hierarchical Browser
- Open schematic for any module



1. Left click to select a module
2. Right click and select *Schematic*

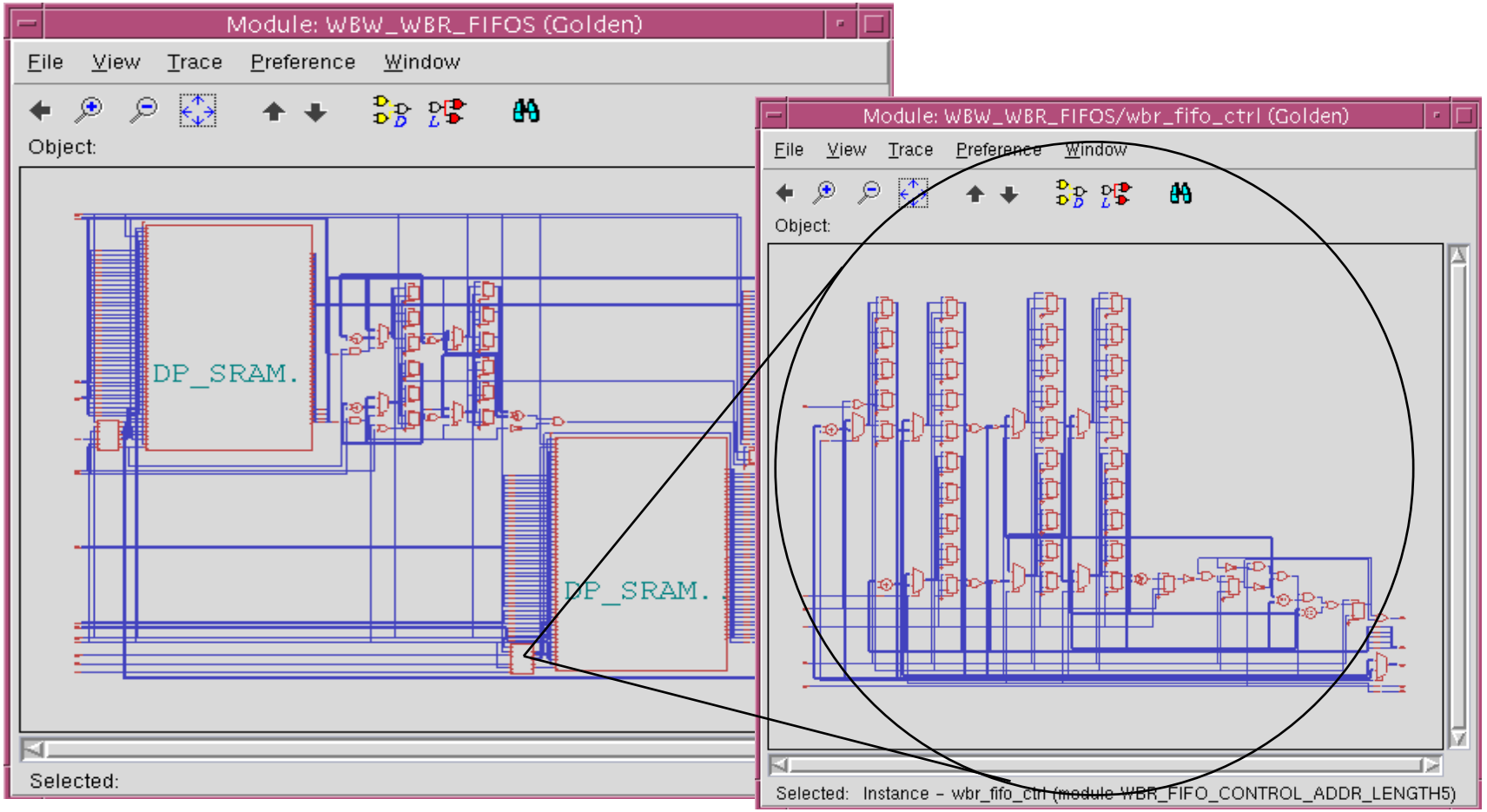
Hierarchical Schematic

- Can also be opened from flatten schematic



Hierarchical Schematic

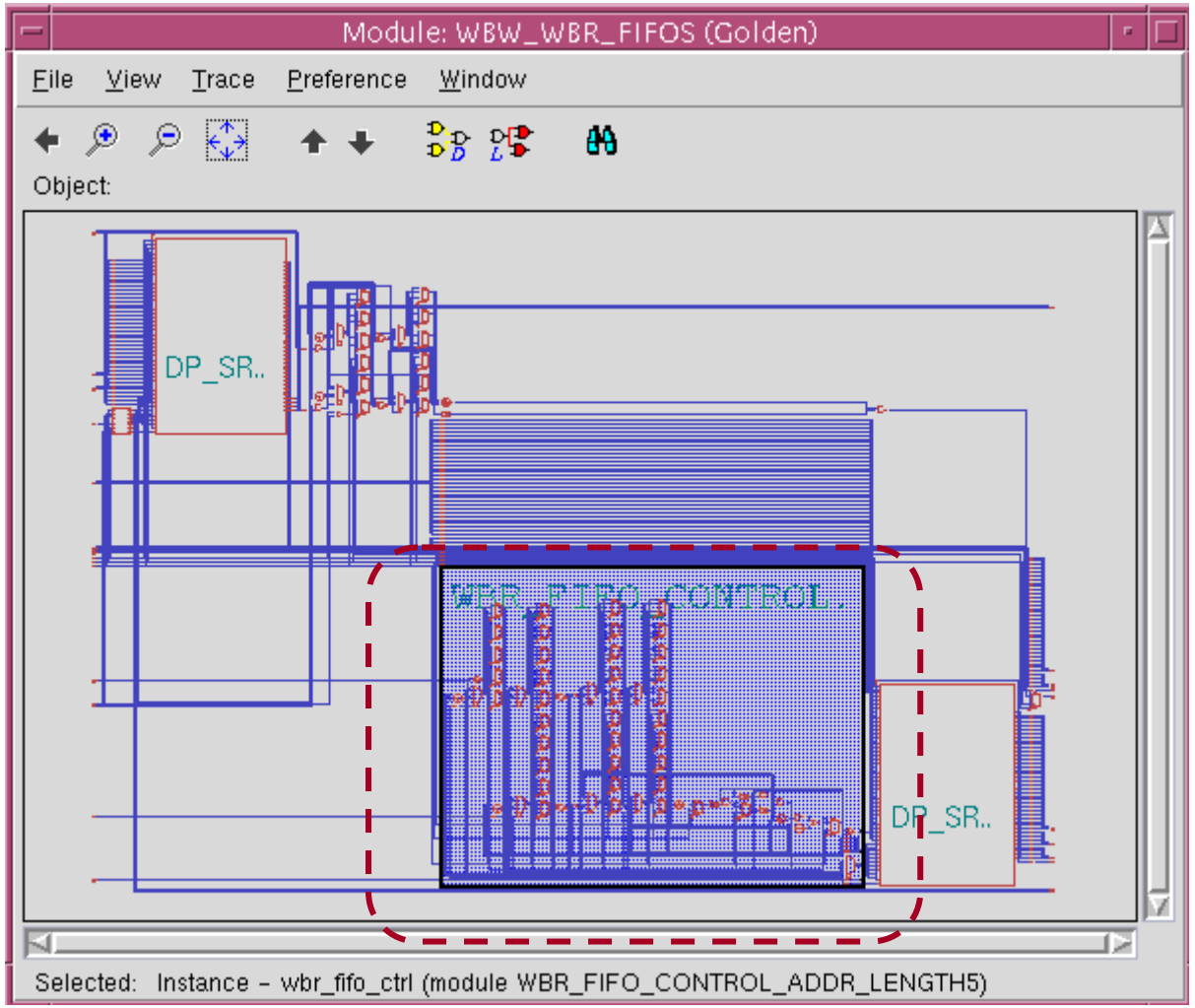
Double click an instance to view its content



Hierarchical Schematic

Instance content can also be displayed using the *Next Level Instance View* feature:

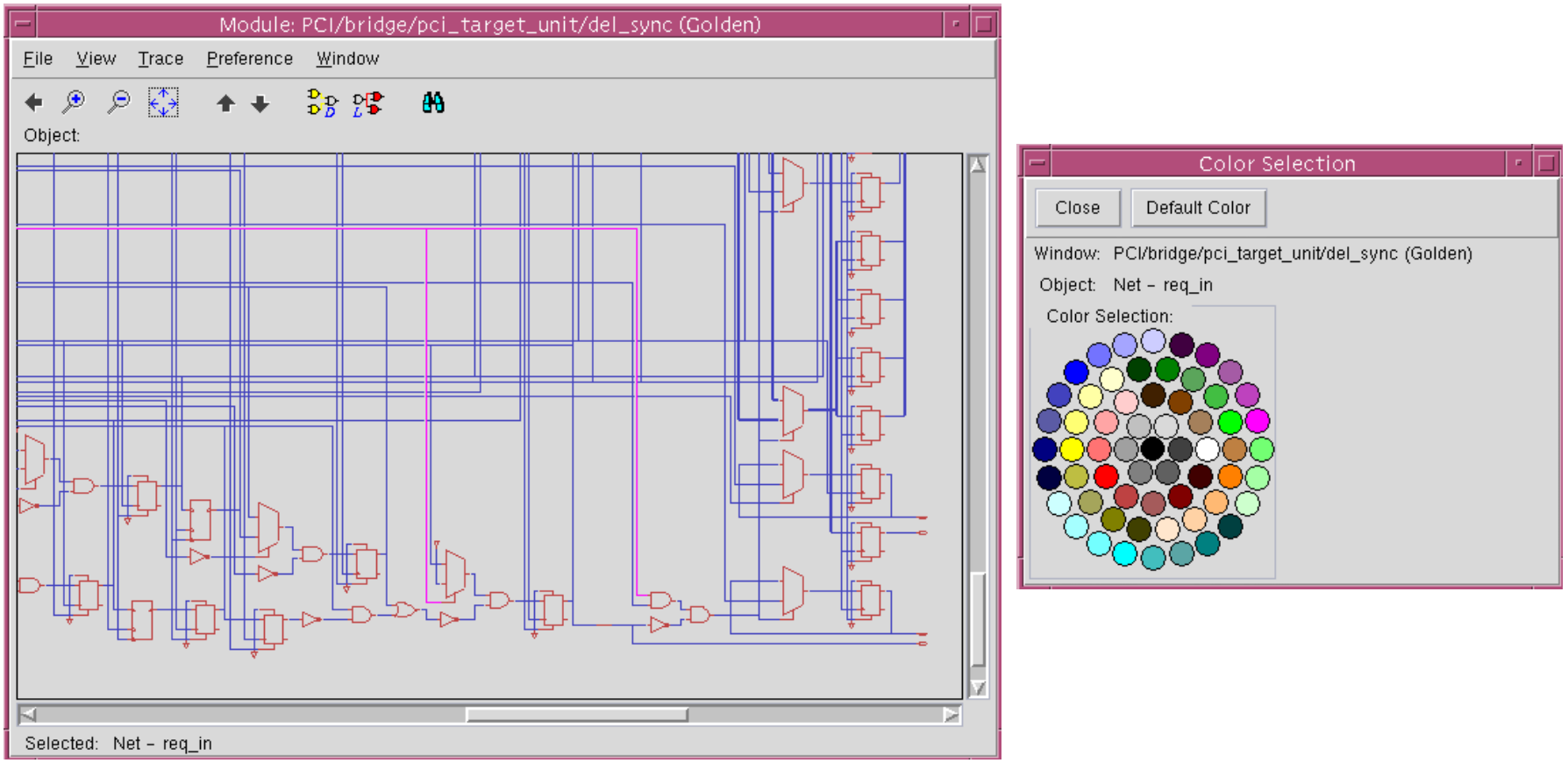
1. Click on an instance to select it
2. Right-click and select *Open Next Level Instance View*



Hierarchical Schematic

To color code a net

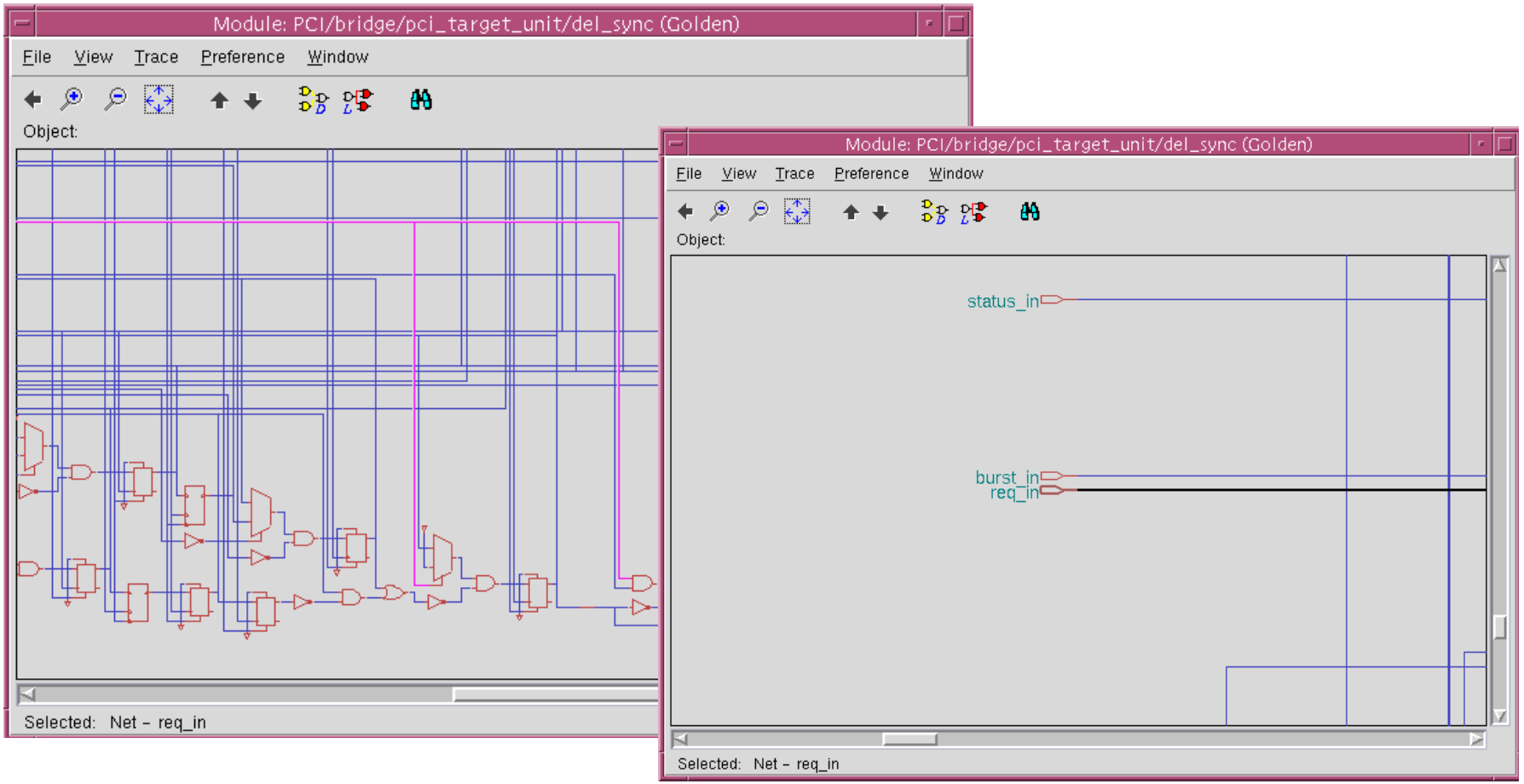
1. Click to highlight the net
2. Press ‘c’ on keyboard, then choose a color from pop-up color wheel



Hierarchical Schematic

To trace a driver:

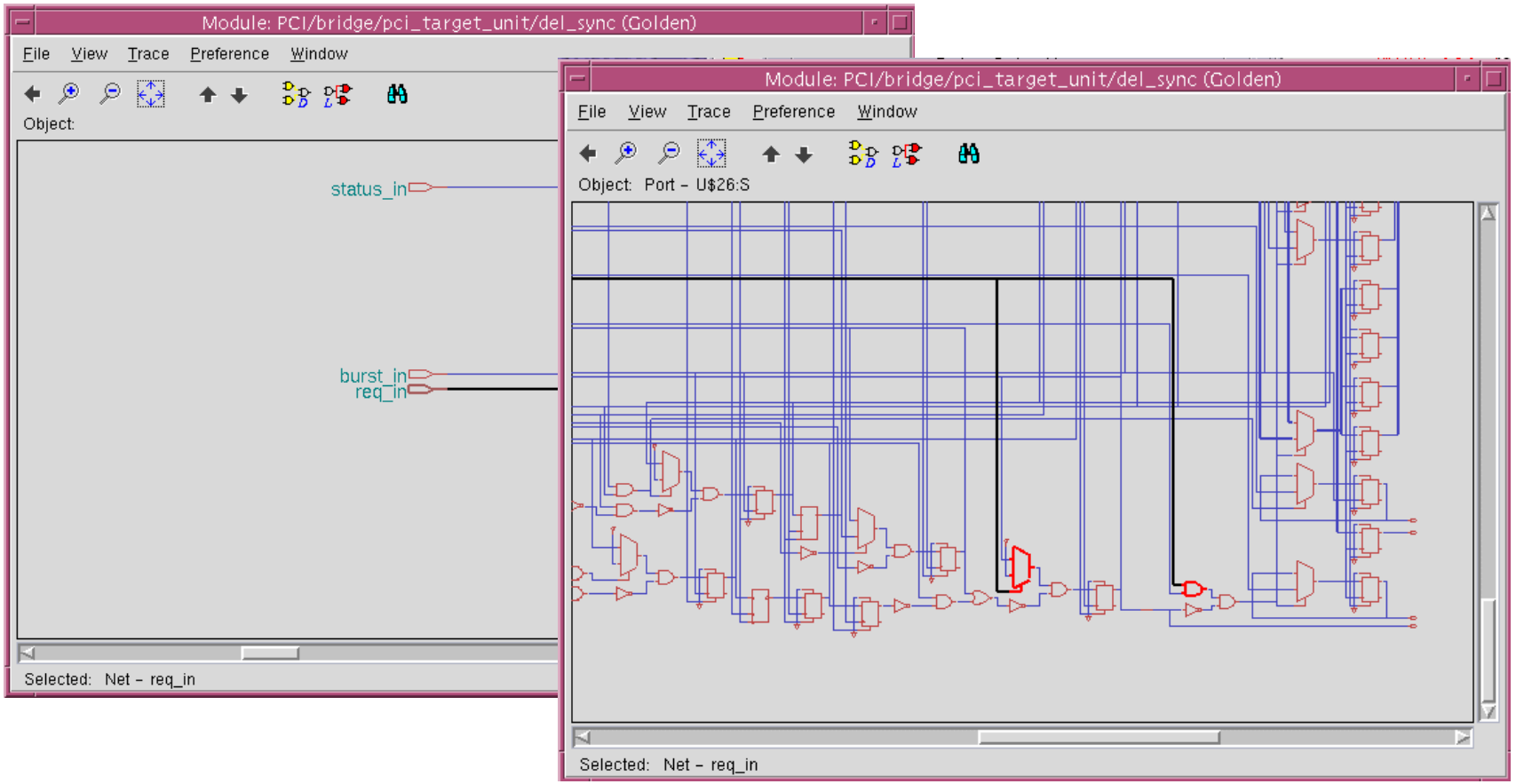
1. Click to select the net.
2. Click the **Driver** icon. 



Hierarchical Schematic

To trace load:

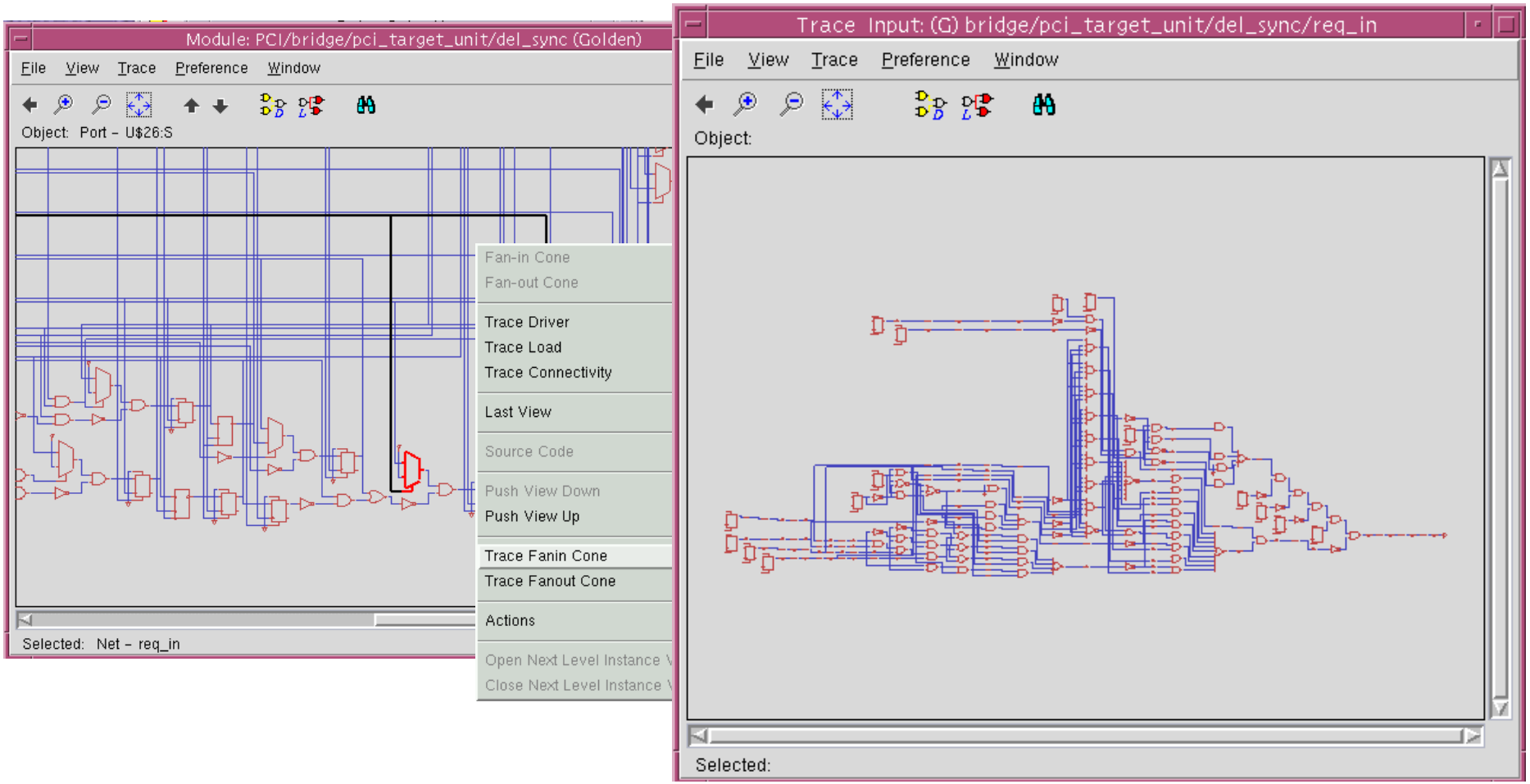
1. Click to select the net.
2. Click the **Load** icon. 



Hierarchical Schematic

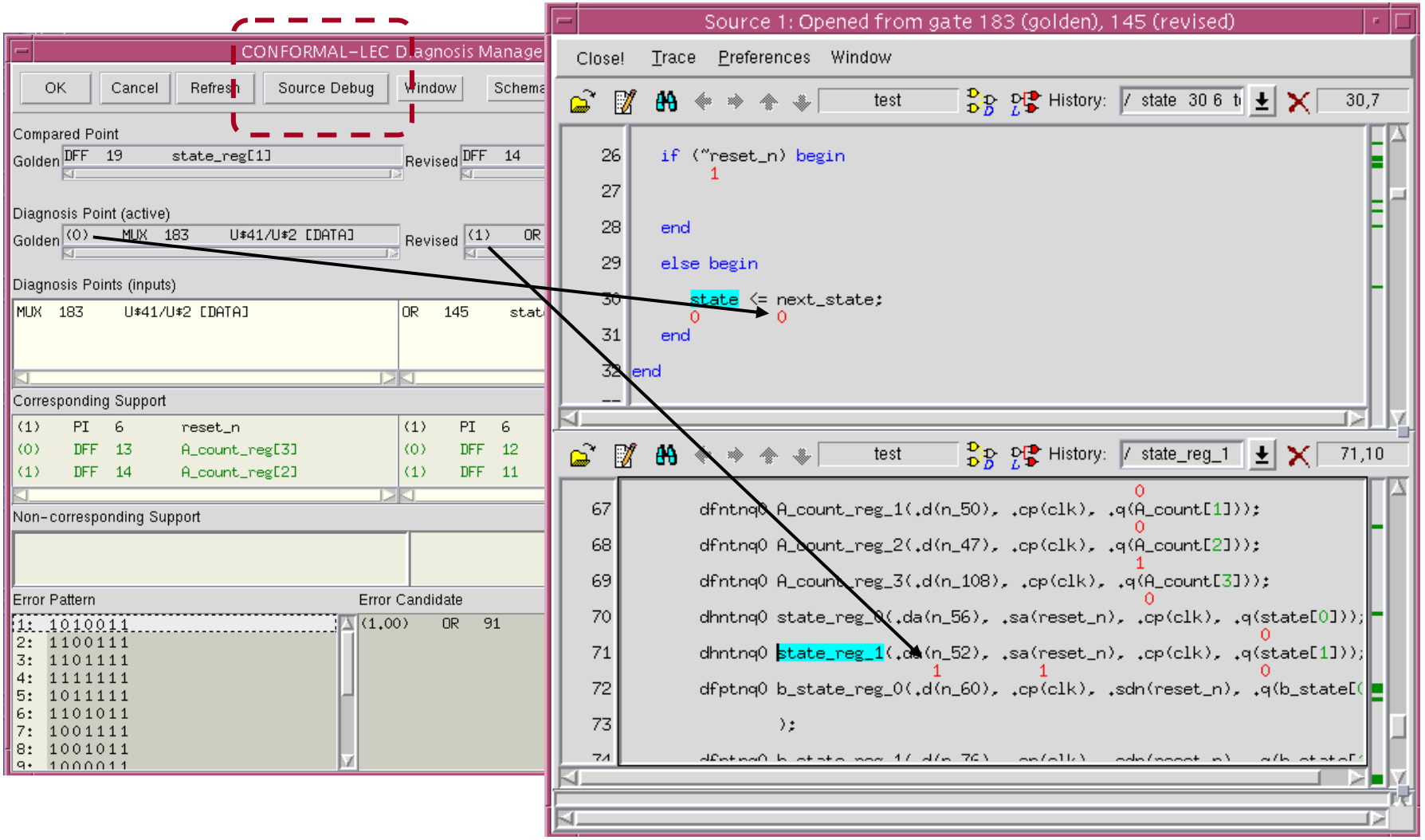
To open a net fanin cone

1. Click to select the net
2. Right click to select *Trace Fanin Cone*



Source Code Manager

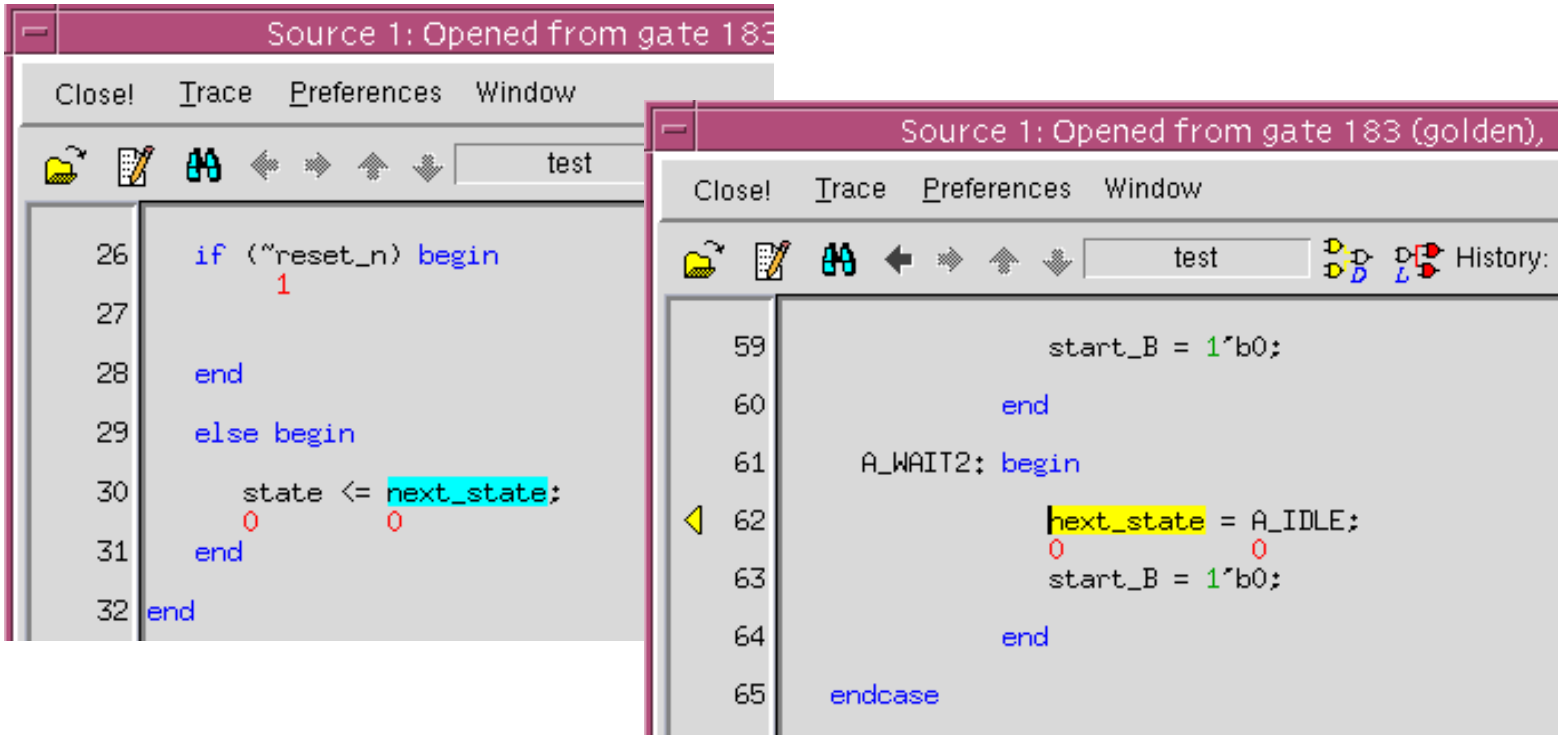
Select *Source Debug* from the Diagnosis Manager (annotated source code)



Source Code Manager

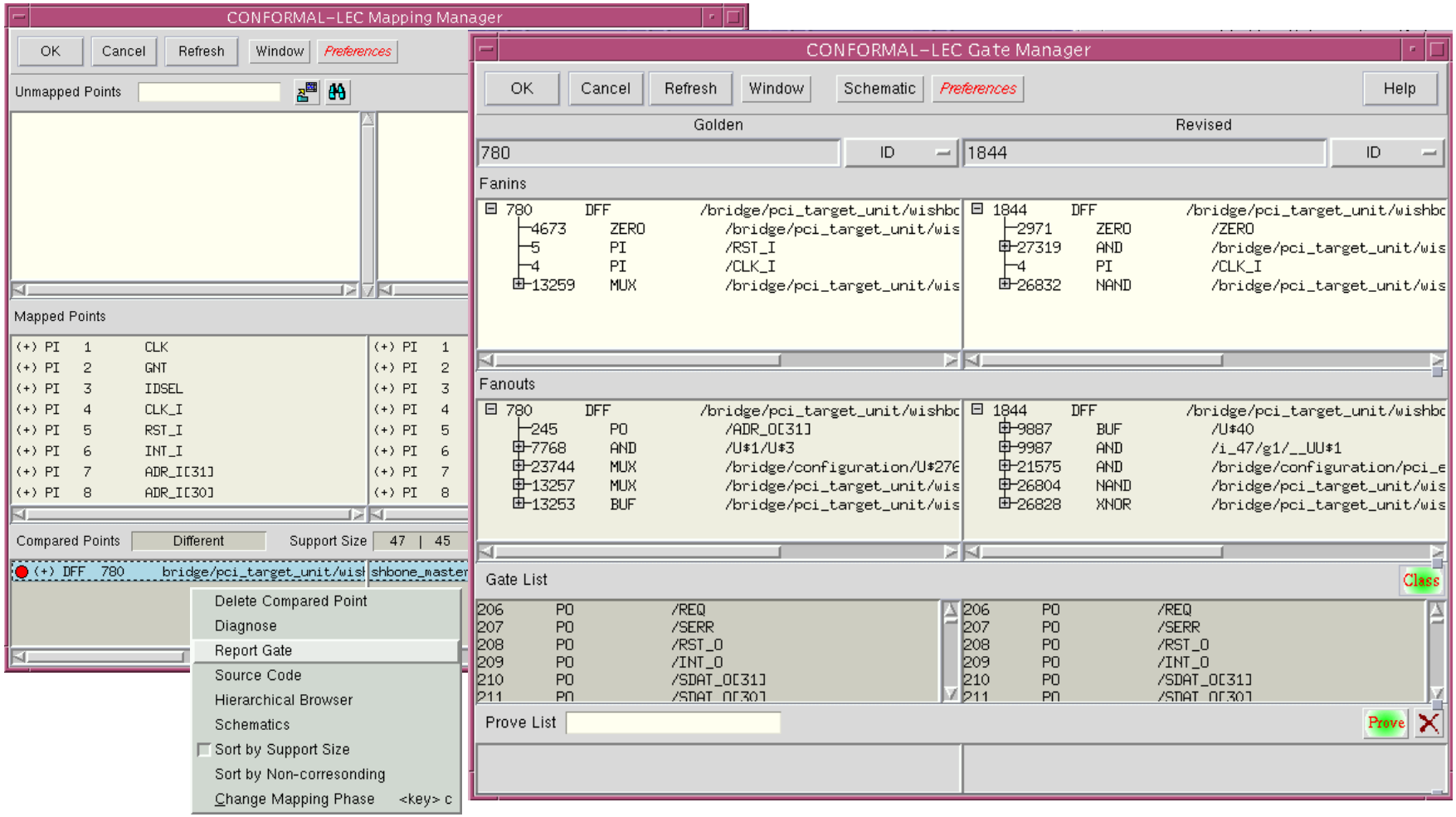
To trace a signal driver:

1. Double click on the signal.
2. Click the **Driver** icon. 



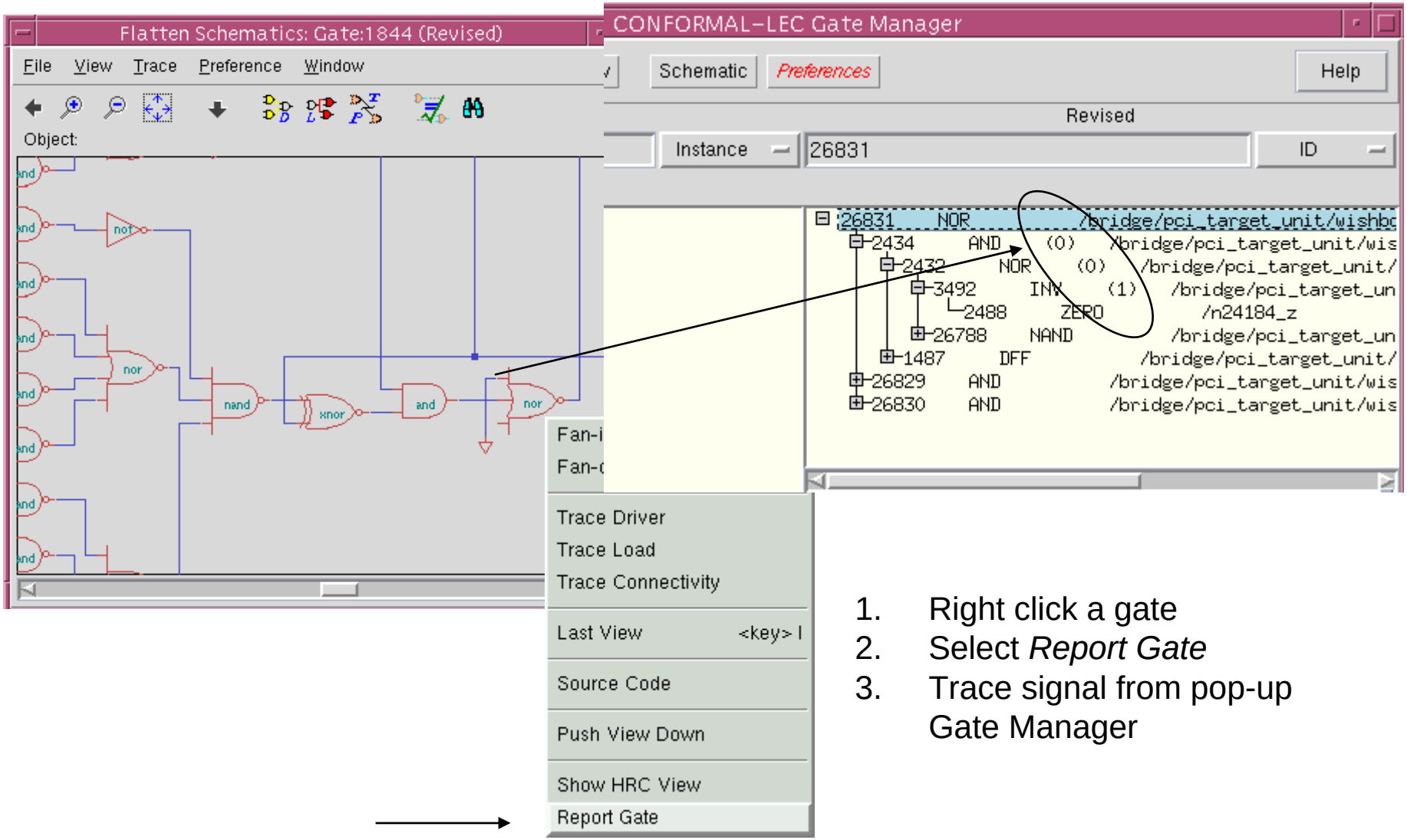
Gate Manager

- Trace signal connectivity
- Can be invoked from Mapping Manager



Gate Manager - Example

Trace the source of ‘0’ from the NOR gate



1. Right click a gate
2. Select *Report Gate*
3. Trace signal from pop-up Gate Manager

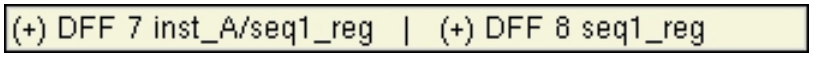
Drag and Drop Feature

Drag and Drop feature available for

- Key Points from Mapping/Diagnosis Manager → Schematic/Source Code Manager
- Gates from Schematic → Mapping/Source Code Manager

Click and hold the middle mouse button to drag a key point/gate to destination

- It will appear in a light yellow box

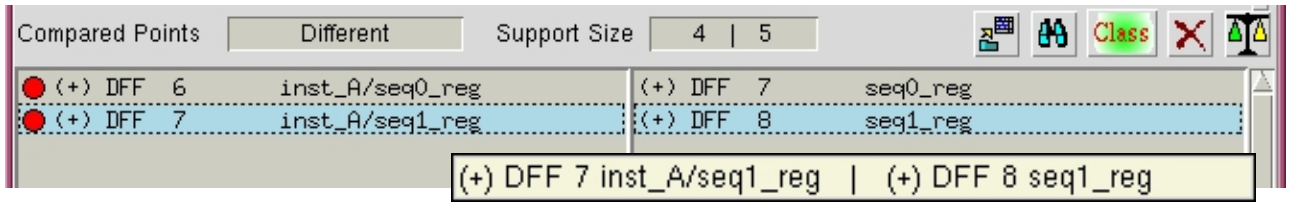


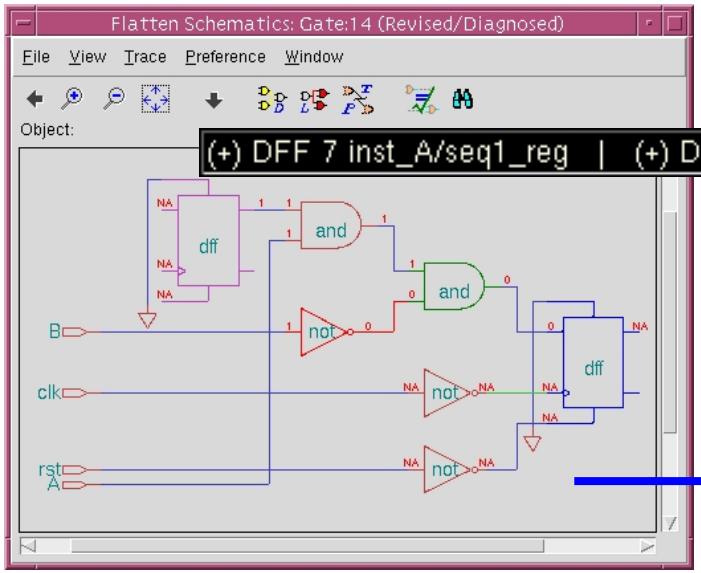
Release the middle mouse button to drop

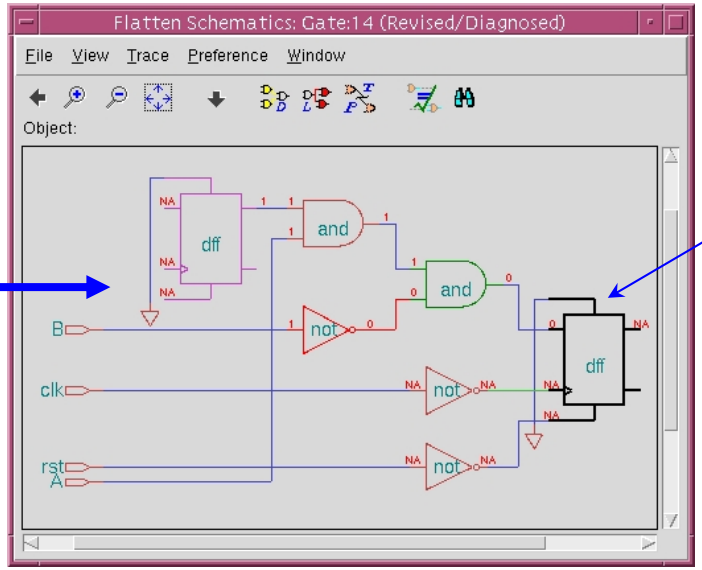
- Yellow box will change to black in a droppable area

Drag and Drop: To Schematic

Key points can be dragged from Mapping Manager and dropped into the Schematic.



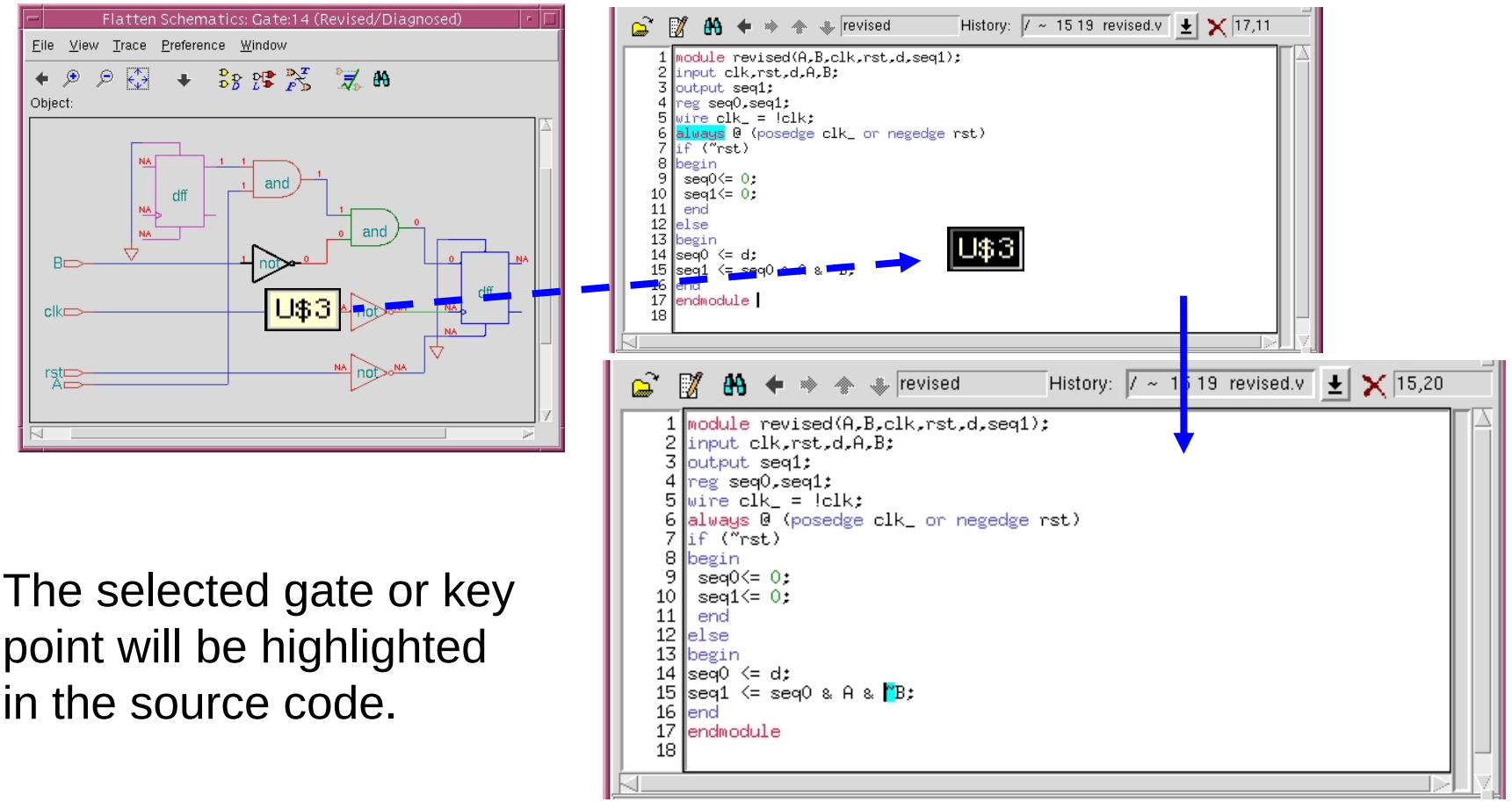




The selected key point will be highlighted

Drag and Drop: To Source Code

Gates or key points can be dragged from the Schematic and dropped into the Source Code Manager.



The selected gate or key point will be highlighted in the source code.

Typical Causes of Non-Equivalence

Module 2

December 19, 2004

Module Objectives

In this module, you will

- Learn the typical causes of non-equivalence
- Debug typical non-equivalent cases using the diagnosis tools

What Can Cause Non-Equivalences (NEQ) ?

NEQ key points can be caused by the following:

- Unqualified library
- Incorrect version of RTL and Gate netlist
- Incomplete design constraints
- Unspecified modeling options
- Incomplete & incorrect mapping
- Unbalanced black-boxes
- ECO/Design bug/Synthesis bug

Incorrect Version of RTL and Gate Netlist

- Be sure to use the correction design version
- Helpful Conformal commands for checking :
  - `report design data`
  - `report primary input`
  - `report primary output`
  - `report module`

NEQ: Unqualified Library

Problem:

Verilog/VHDL simulation library vs synthesis Liberty format library

- Conformal recommends using a simulation library for logical equivalence checking
- Design verification sign-off on Verilog/VHDL simulation library (.v)
- Synthesis tool uses synthesis Liberty library (.lib, .db)

Solution:

Qualify library to ensure correctness between synthesis library and simulation through Conformal LEC

- Conformal LEC commands :

```
read design slow.lib –liberty -golden
read design slow.v –verilog -revised
write hierarchical dofile validate_lib.do –all –replace
dofile validate_lib.do
```

Design Constraints

What are design constraints?

- User 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

Examples of constraints:

- Pin constraint
- Instance constraint
- Pin equivalence
- Tied signal
- Undriven signal

Example of Design Constraints - Test logic

Problem:

- NEQ can be caused by one design having test logic while the other design doesn’t

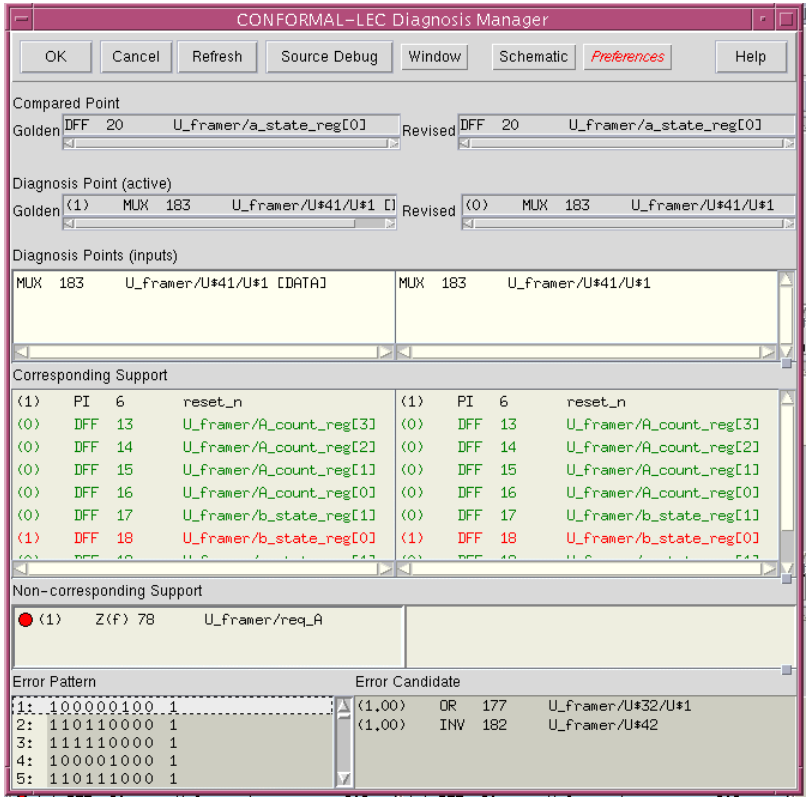
Solution:

- Constrain away test logic to avoid false NEQ
  - Scan
  - JTAG

Another Example of Design Constraint

NEQ due to floating signal

Golden non-corresponding support ‘Z’ gate (Revised netlist optimized away Z)



Command “`set undriven signal`” should be carefully used. Generally recommended to tie off floating pins in RTL

Example of Constraining Commands

```
add pin constraint
add instance constraints
add net constraints
add output stuck_at
add tied signals
add instance equivalences
add pin equivalences
set undriven signal
```

Modeling Options

Problem:

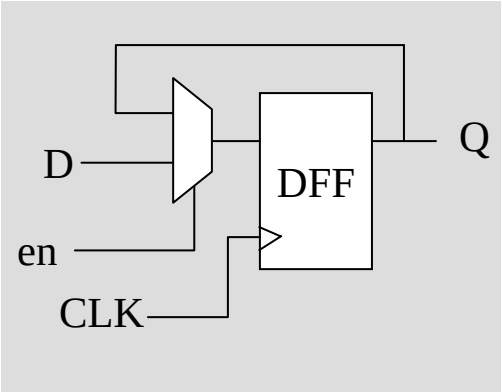
- Modeling style and optimization can lead to false NEQ unless handled properly
- Examples
  - Gated clock
  - Sequential optimization
  - Sequential replicate/merge

Solution:

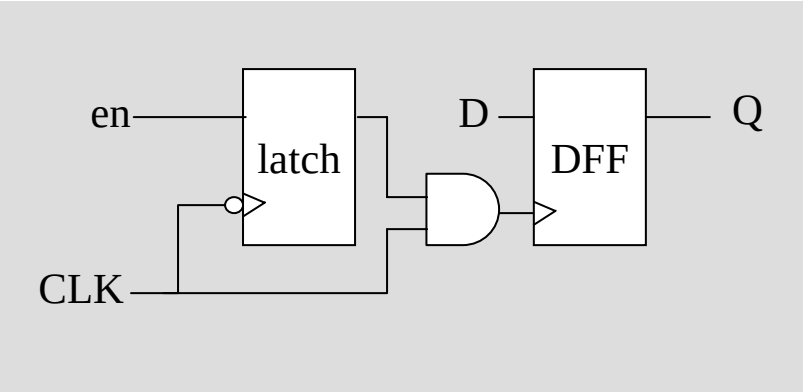
- Use the "`set flatten model …`" command to choose modeling style and enable optimization.

Example: Gated Clock

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



Golden

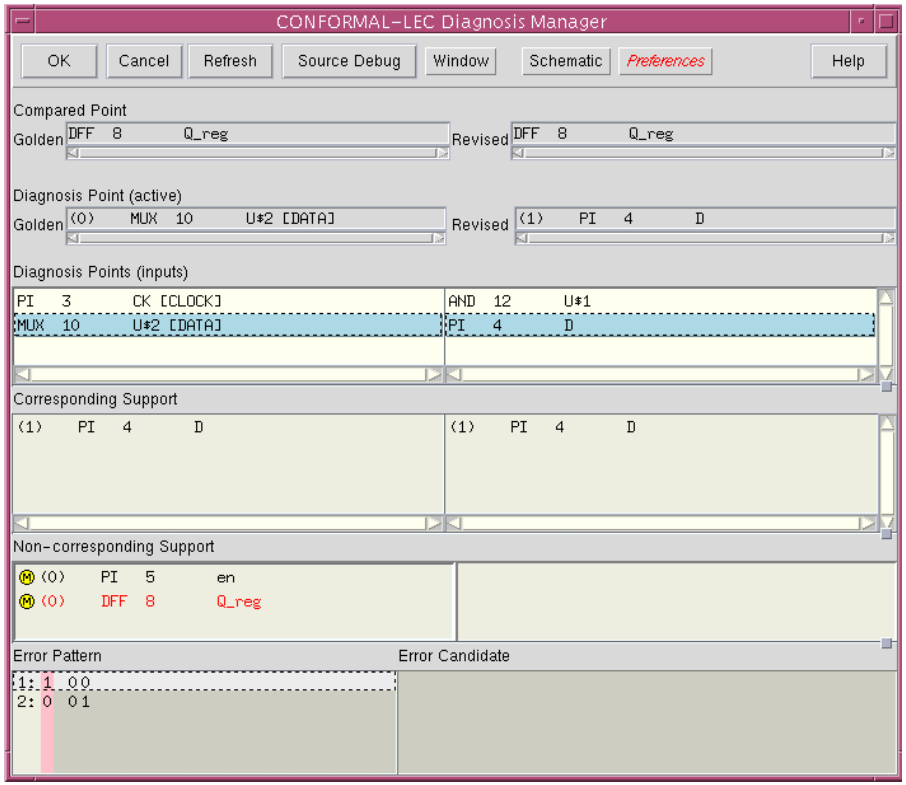


Revised

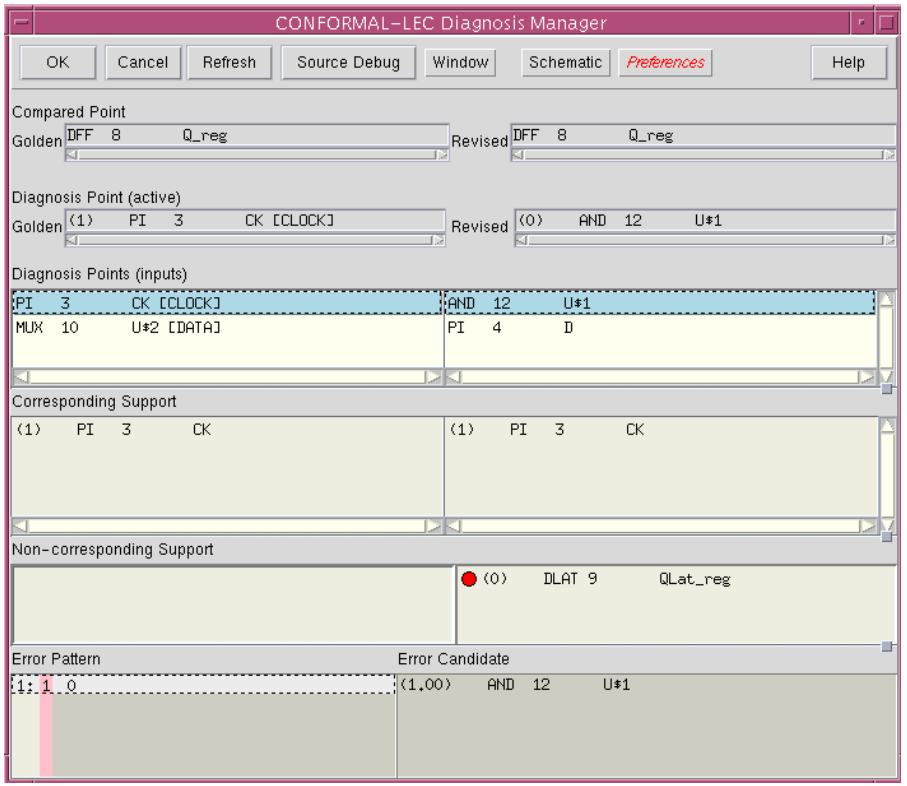
Example: Gated Clock

Information on the Diagnosis Manager

Diagnose NEQ on data cone



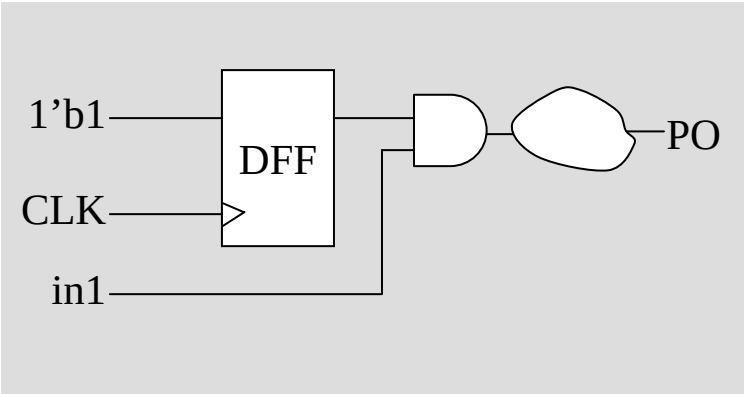
Diagnose NEQ on clock cone



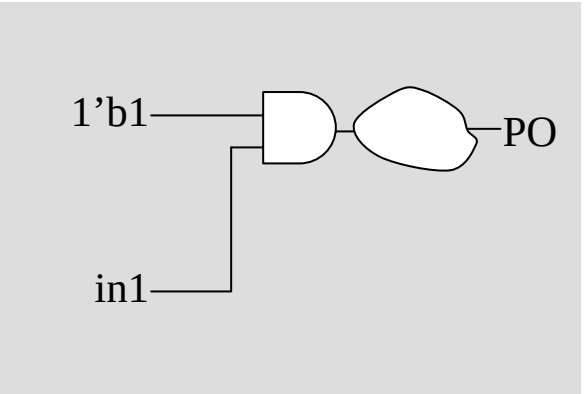
Use “`set flatten model -gated_clock`” to resolve

Example: Sequential Constant

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



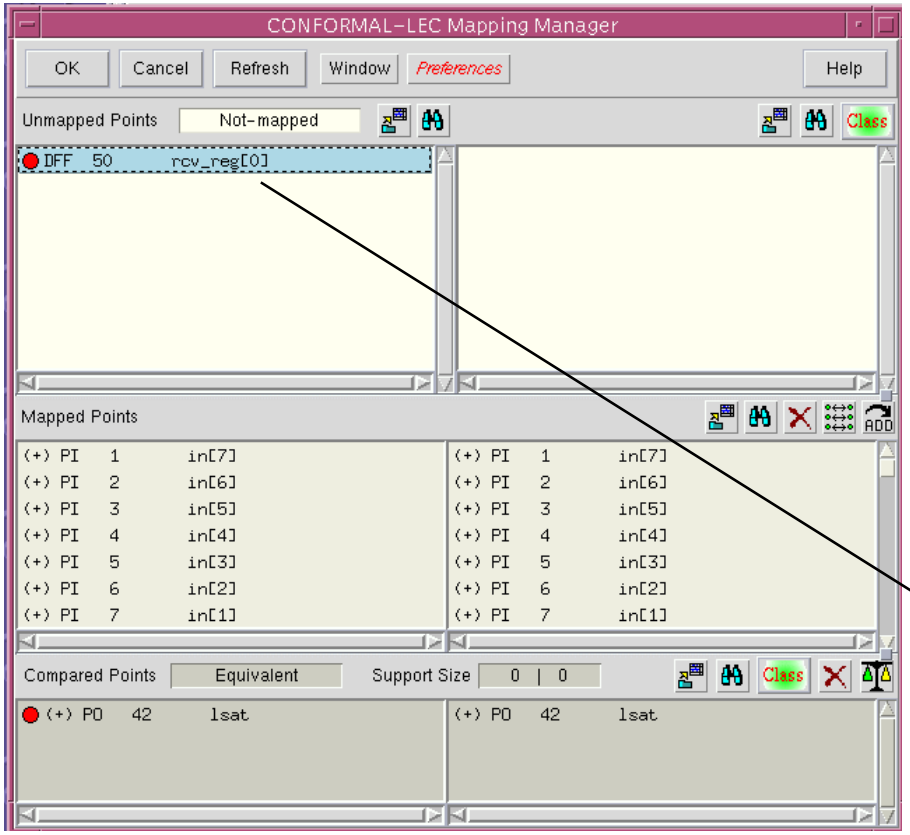
Golden



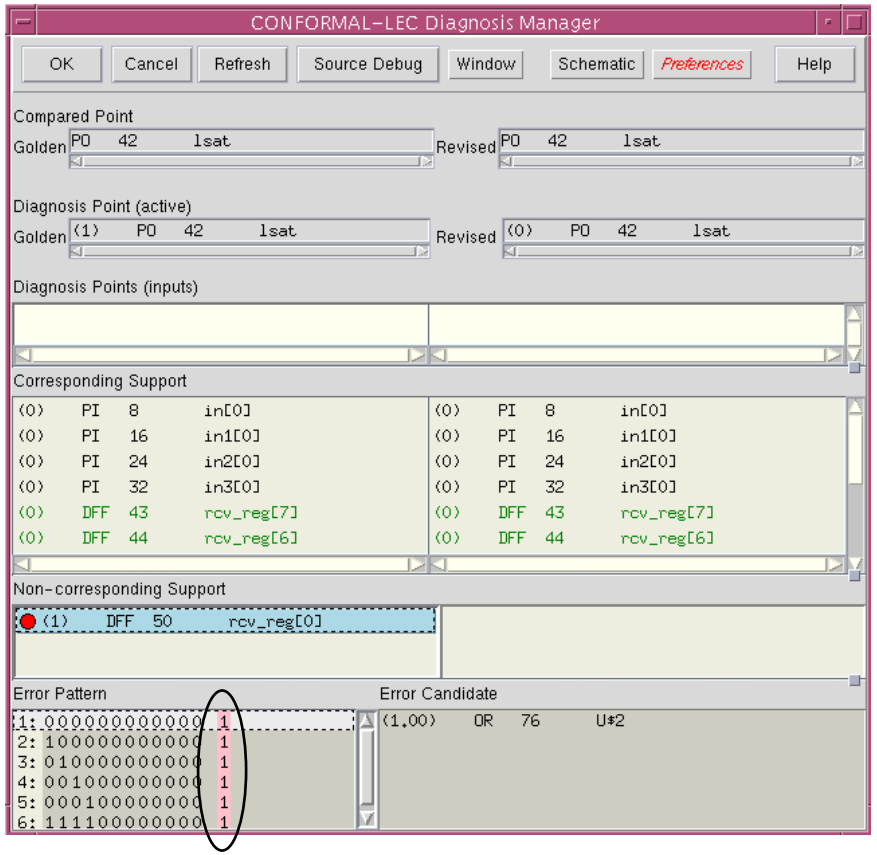
Revised

Example: Sequential Constant

Mapping Manager



Diagnosis Manager



Use “`set flatten model -seq_constant`” to optimize registers

Sequential Constant (cont'd)

"`remodel –seq_constant`" can also be used to solve a sequential constant problem

- A command in LEC mode
- Can be used in conjunction with `set flatten model –seq_constant`
- Works best after mapping is done

Transcript window

```
...
// Command: set system mode lec
// Command: remodel –seq_constant
// Command: map key point
// Command: add compare point –all
// Command: compare
...
```

Modeling Command

```
SET FLatten Model
            [-NOLATCH_Fold | -LATCH_Fold]
            [-NOLATCH_Transparent | -LATCH_Transparent]
            [-NOALL_SEQ_Merge | -ALL_SEQ_Merge]
            [-NOALL_INV_SEQ_Merge | -ALL_INV_SEQ_Merge]
            [-NOSEQ_Redundant | -SEQ_Redundant]
            [-NOLIB_SEQ_Redundant | -LIB_SEQ_Redundant]
            [-NOSEQ_Constant | -SEQ_Constant]
            [-SEQ_FEEDBACK_CONSTant | -NOSEQ_FEEDBACK_CONSTant]
            [-DFF_TO_DLAT_FEEDBACK | -NODFF_TO_DLAT_FEEDBACK]>
            [-NOLOOP_AS_DLAT | -LOOP_AS_DLAT]
            [-AUTO_MODELING | -NOAUTO_MODELING]
            …
```

Mapping

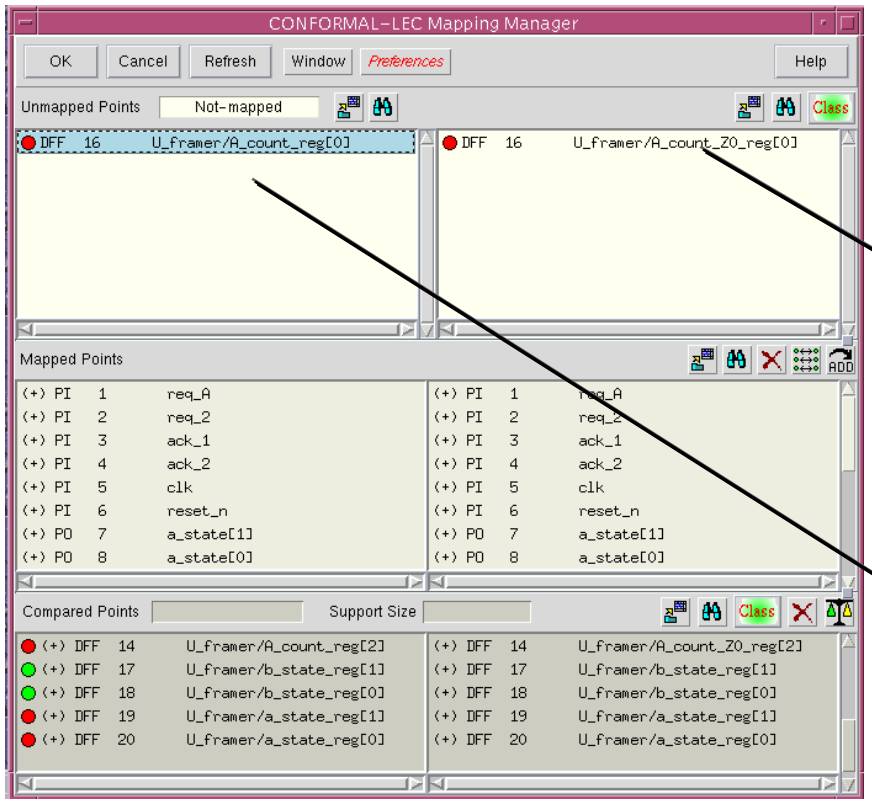
NEQ can be caused by

- Incomplete mapping of key points
- Incomplete mapping of black box pins
- Incorrect mapping
  - Incorrect pairing
  - Phase map

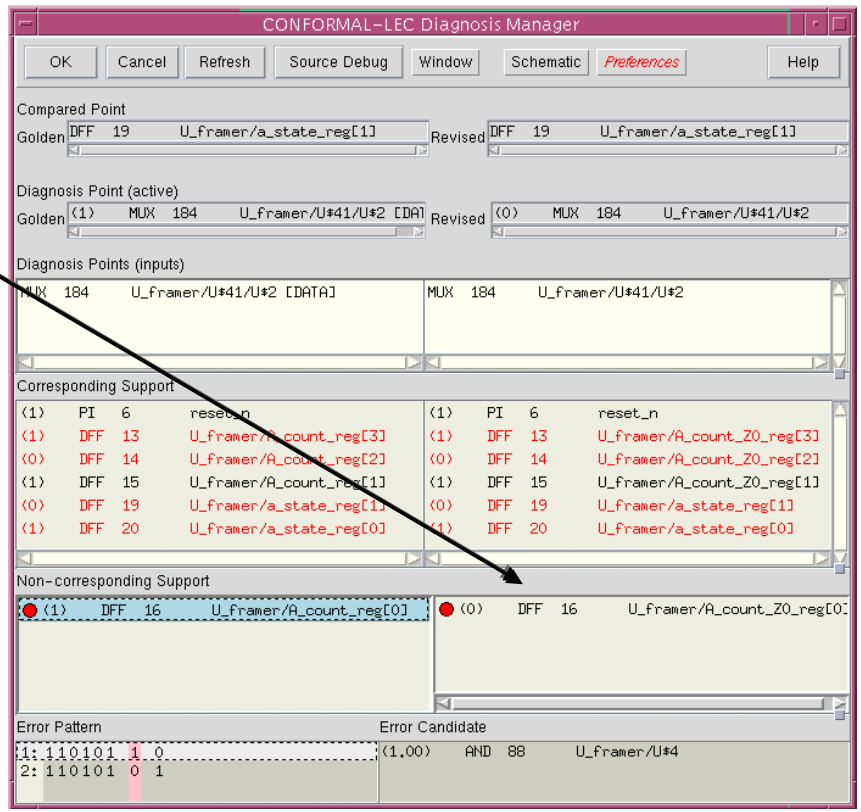
Mapping Problem: Incomplete Mapping

Incomplete mapping causes NEQ (except Unreachable & Extra)

Mapping Manager



Diagnosis Manager



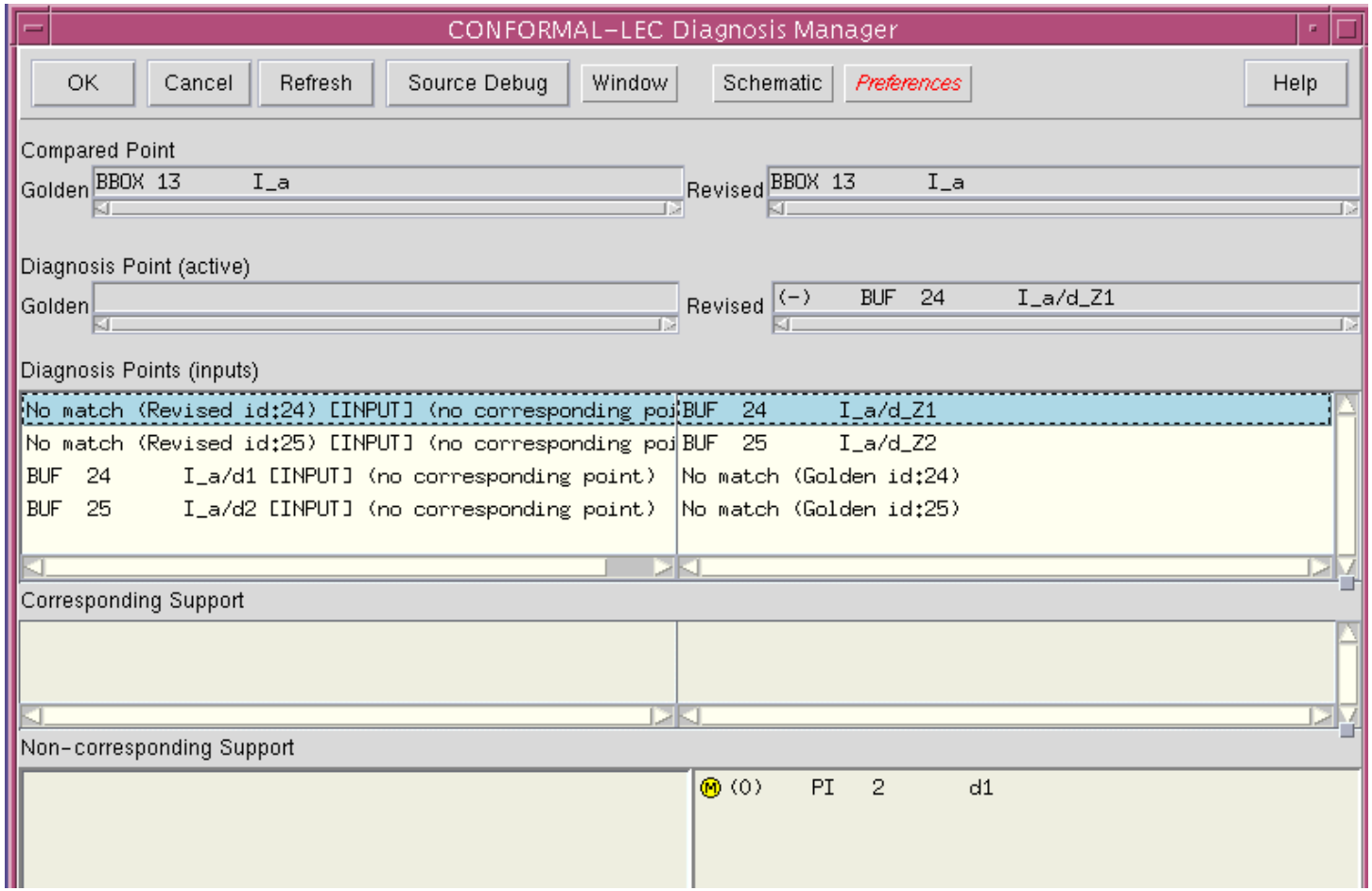
Resolving Incomplete Mapping

Use the command `“add renaming rule”` to complete mapping

Mapping Problem: Incomplete Pin Mapping

- Black box pins not mapped due to name differences
- Diagnose BBOX NEQ

NEQ pins



Reporting Black Box Pin Mapping

- Check to make sure black box pins are mapped (unless not to be compared)

  LEC> `report mapped point <BBOX> -input`

Transcript window

```
LEC> report mapped point I_a -input
// Command:  report mapped point I_a -input
Mapped point for
  (G) + 13  BBOX /I_a
is
  (R) + 13  BBOX /I_a
---------- input mapping ----------
[INPUT] pair
  (G) + 23  BUF  /I_a/ck
  (R) + 23  BUF  /I_a/ck
[INPUT] (no correspondence)
  (G)   24  BUF  /I_a/d1
[INPUT] (no correspondence)
  (G)   25  BUF  /I_a/d2
[INPUT] (no correspondence)
  (R)   24  BUF  /I_a/d_Z1
[INPUT] (no correspondence)
  (R)   25  BUF  /I_a/d_Z2
```

Pins not mapped

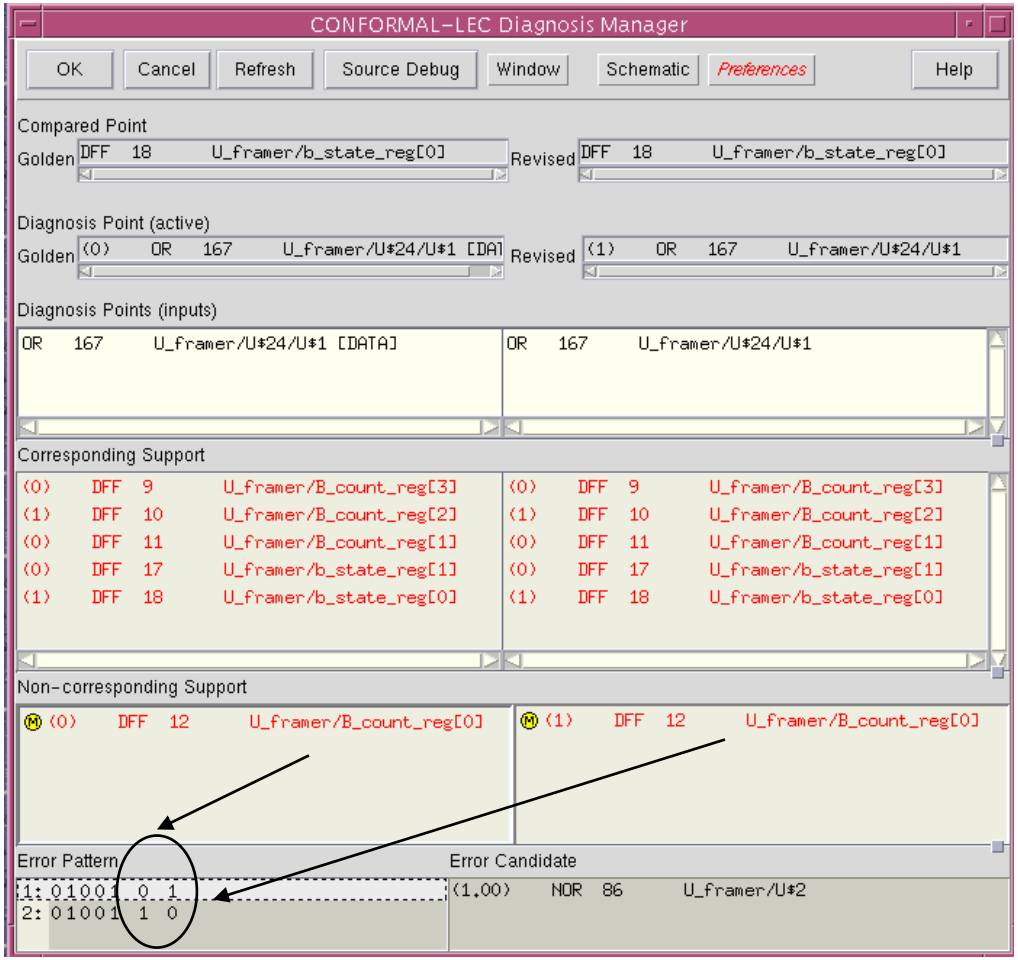
Resolving Incomplete Pin Mapping

Apply the renaming rule command to map the pins, as follows:

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

Mapping Problem: Incorrect Mapping

Symptom of incorrect pairing of key points

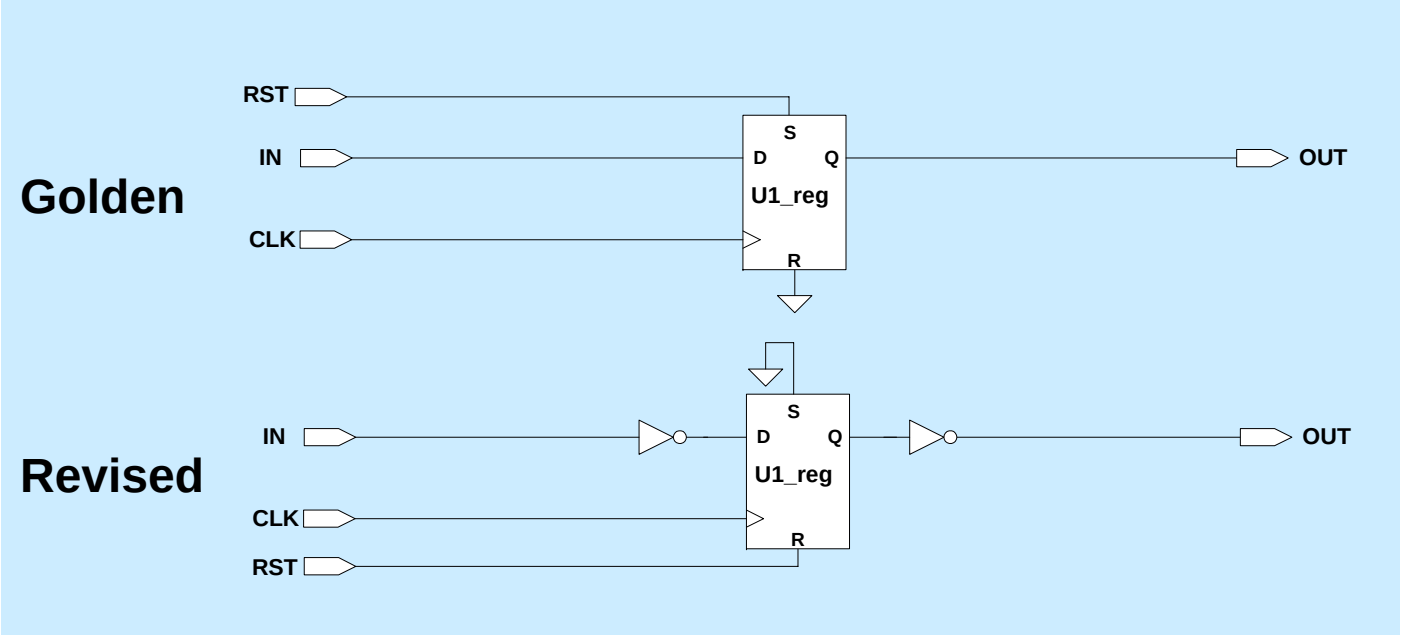


Resolving Incorrect Mapping

- Remap the incorrectly mapped pair with `"delete mapped point …"` and `"add map point …"`
- Or use `"add renaming rule"` to control how key points should be mapped

Mapping Problem: Phase Inverted Mapping

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

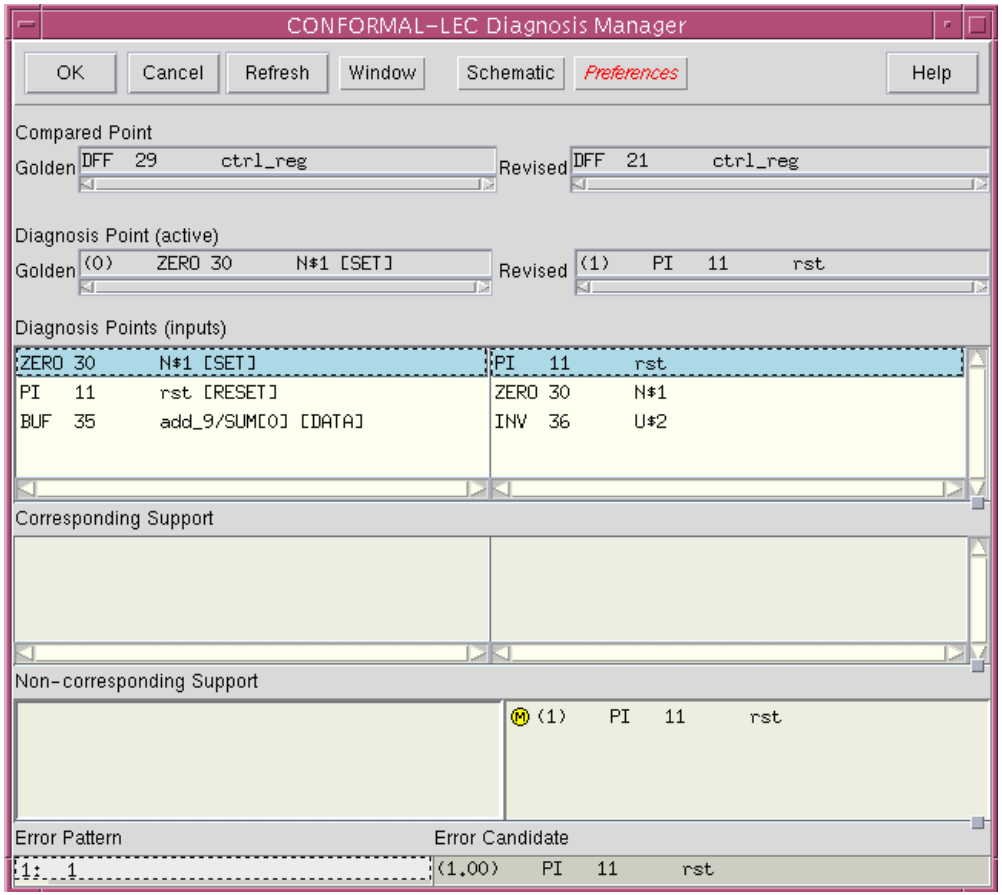


- Instance U1_reg must be phase-mapped

Mapping Problem: Phase Inverted Mapping

Symptom of phase mapping problem

Combination of set/reset/data cones non-equivalence



Resolving Phase Inverted Mapping

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 to 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
...
```

On-the-fly Method to Resolve Mapping Problem

During diagnosis, try the method below if you think some key points might be incorrectly mapped, or need to be phase-mapped

```
set log file logfile.$LEC_VERSION –replace
setenv LIB /user1/lib/verilog/
read design –file cpu_rtl.vc -verilog –golden
read design -file verilog.vc -verilog –revised
add pin constraint 0 scan_en –revised
set flatten model –seq_constant –gated_clock
set system mode lec
add compare point –all
compare
//compare result has non-equivalences
delete mapped point –noneq
set mapping method –phase
map key point
add compare –all
compare
```

NEQ Due to Black Boxes

Problem:

Not evenly black boxing both designs

Solution:

- Check for unbalanced black boxes in the Golden and Revised , then black box both designs evenly
- SETUP> `report black box`

LEC window

Find out why the Revised module ‘astm’ is not black boxed

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

ECO/Design bug/Synthesis bug

- Non-equivalences can be a real bug.
- Use the diagnosis tools to debug.

Blank Page

Resolving Aborts

Module 3

December 19, 2004

Module Objectives

In this module, you will

- Learn what causes aborts
- Learn how to resolve abort key points

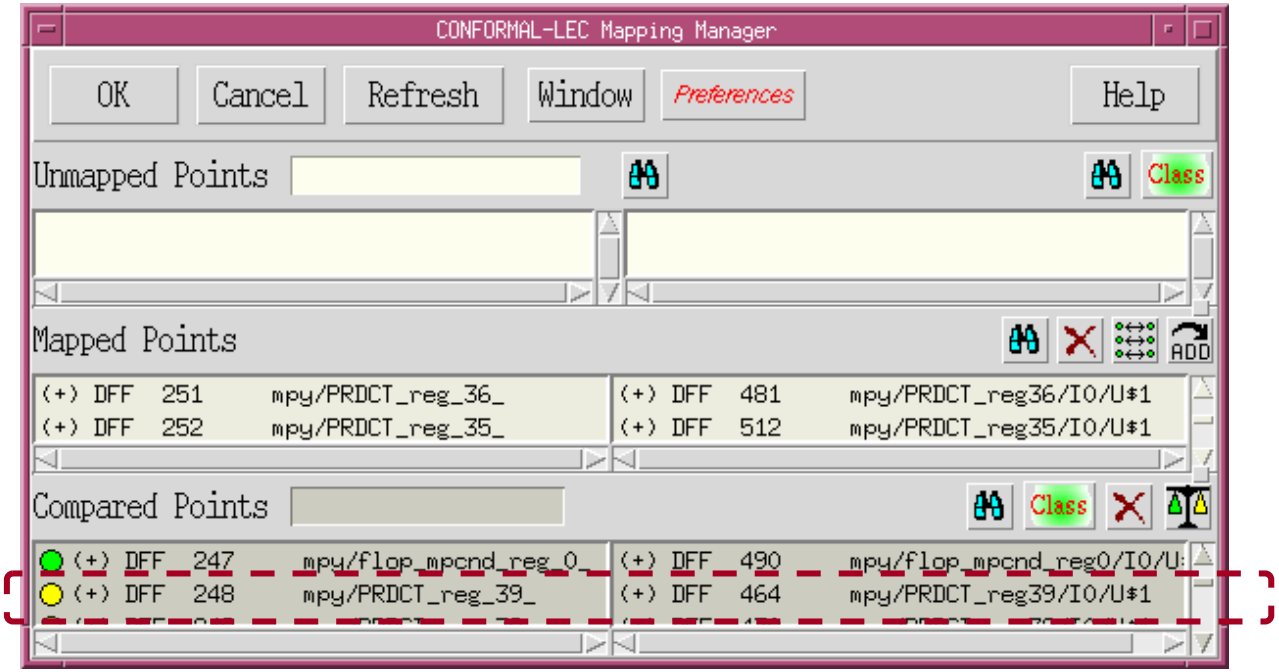
Module Overview

- What are abort points?
- Why aborts?
- What to do if compare aborts:
  - Ensure that the dofile is set up properly
  - Increase compare effort
  - Perform hierarchical compare

What are Abort Key Points?

- Inconclusive comparison result. You need to resolve the aborts
- GUI: yellow-filled circle on Mapping Manager

Abort key points are shown with yellow-filled circle →



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

Why Abort?

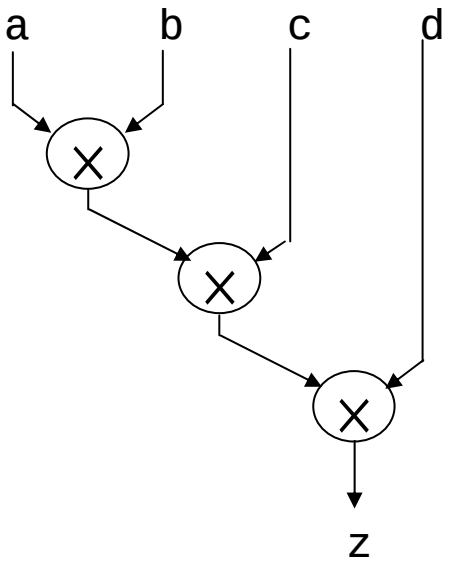
- Complex datapath
- Large number of don’t cares (X)
- Large logic cone size

Complex Data Path: Operators Tree

- Different logical structures

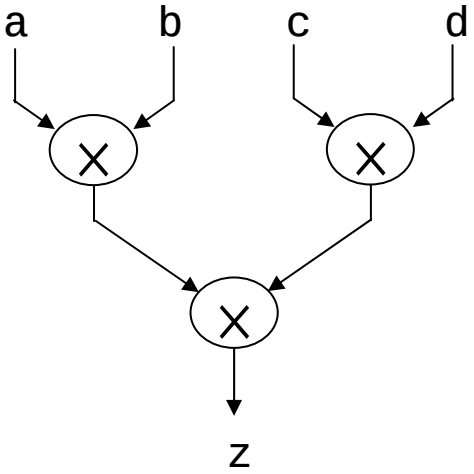
RTL ordering

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



Netlist ordering

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

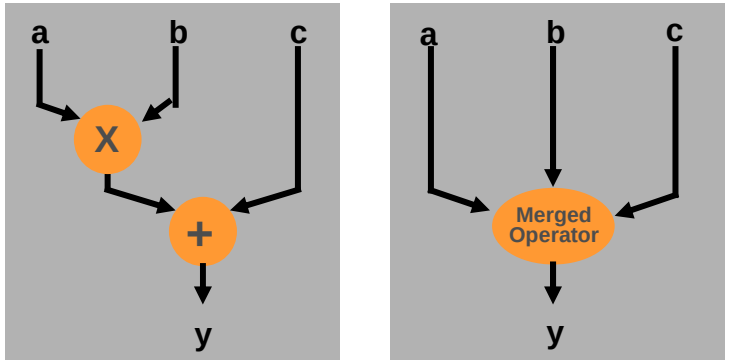


Complex Data Path: Operators Merging

Combine addition-based operators into carry-save tree

- +, -, *, <, >, <=, >=, ==, !=

Example: y = a * b + c



Intermediate value not computed (a*b)

Use Conformal Ultra command “`analyze datapath –merge`”

Don’t Cares

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

    “report map point <abort_point> -property”

  - Full case/parallel case/x-assignments
  - Re-coding RTL to get rid of don’t cares might help
  - Hierarchical compare might help too. Please refer to hierarchical compare (Module 5)

Handling Aborts

- Ensure that the dofile is set up properly
- Increase compare effort
- Perform hierarchical compare
- Adding CUT points
- Use Partition Keypoint flow

Resolving Aborts: Set Up the Dofile Properly

- Check if RTL has multipliers
  - LEC> `report design data`

Transcript window

```
...
------ word-level --------------------------
ADD         *              55                  0
MULT        *               3                  0

BBOX                        2                  2
-----------------------------------------------
Total                    1058               1058
```

- Please refer to Multiplier Handling (Module 4)
- Design with complex datapath
  - Use "`analyze datapath`" command in Conformal Ultra

Resolving Aborts: Increase Compare Effort

- Default effort: low
- Change effort to high or super
  - For any abort cases, higher compare effort can certainly help, but not guaranteed to solve all abort cases.
  - 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 super
compare
...
```

Resolving Aborts: Hierarchical Compare

- Narrows down abort modules
- Reduces logic cone size
  - Generally helps the tool to compare designs easier
- Please refer to the Hierarchical Compare section

Resolving Aborts: Adding CUT Points

Adding CUT points is an effective method to resolve aborts

- At multiplexor select nets
- To partition datapath operators

Adding CUT points can help in diagnosis and error isolation

Difficulties in adding CUT points

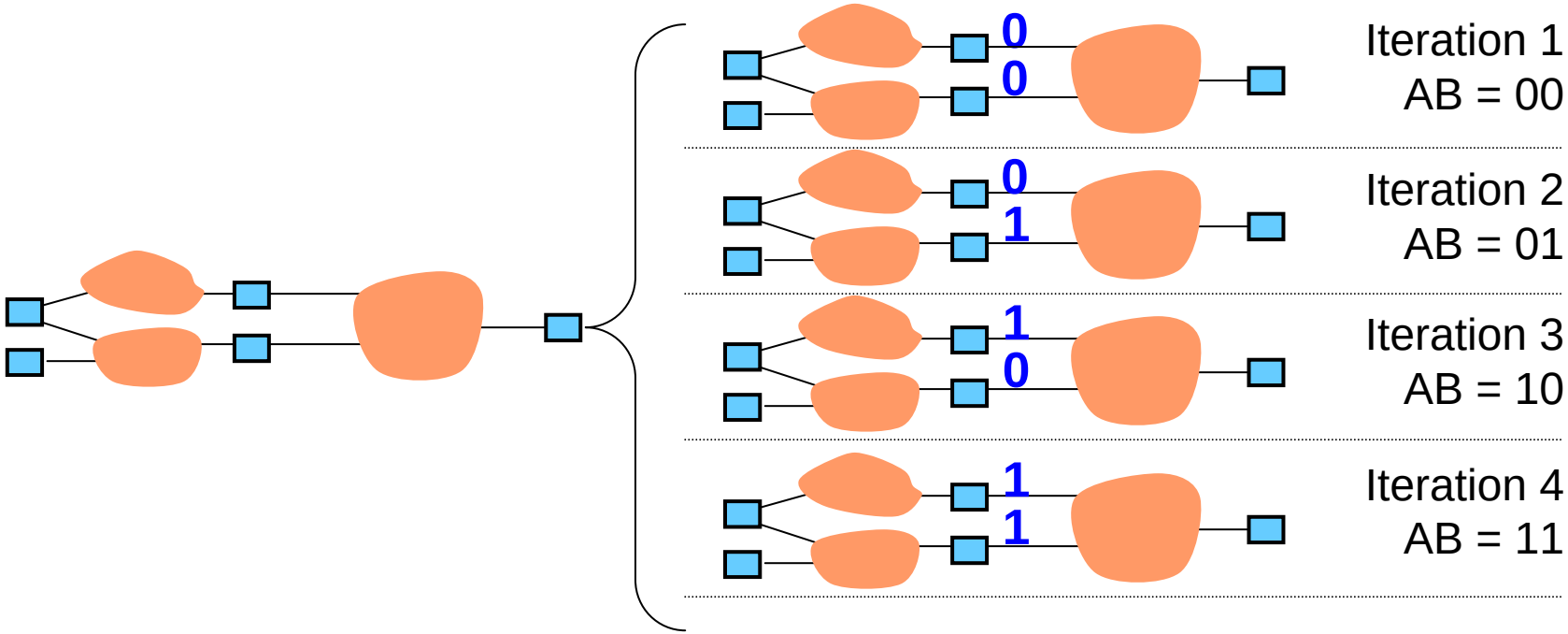
- Need to find corresponding points in both designs
- Need to switch between LEC and SETUP mode
- Need to find and specify the correct net
- Need to resolve false-negatives
- Specification in SETUP mode is sometimes not accurate

Approach for Adding CUT Points

```
// Identify golden cut points
// Identify revised cut points
SETUP> add cut point ...
SETUP> set system mode LEC
LEC> add compare points –all ; compare
// Investigate NEQ or need for more cut points
LEC> set system mode SETUP
SETUP> Delete cut point ...
SETUP> Add cut point …
SETUP> set system mode LEC
// Check mapping of CUT points
LEC> add compare point –all; compare
. . . .
```

Key Point Partitioning

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



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

Key Point Partitioning

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
```

Handling Multipliers

Module 4

December 19, 2004

Module Objectives

In this module, you will

- Learn how to compare designs with multipliers

Multipliers

- Multiplier logic is complex and hard to compare
- Can lead to long runtime or abort

DC Synthesized Multipliers

DesignCompiler uses DesignWare 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

Report architectures from DC command

- dc_shell>report_resources

Instantiated Multiplier or Divider

Multiplier or divider instantiated in RTL

- DW02_mult #(Awidth, Bwidth) U1 (.A(a), B(b), .TC(1'b1),
  .PRODUCT(z);
- DW02_divide #(Awidth, Bwidth, TCmode) U2(.A(a), .B(b), .TC(tc),
  .DIVIDE_BY_0(div_by_0),
  .QUOTIENT(result));

Conformal 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

```
...
------ word-level --------------------------
ADD         *              55                    0
MULT        *               3                    0
BBOX                        2                    2
-------------------------------------------------
Total                    1058                 1058
```

- Unmatched multiplier architectures can lead to long runtime or abort

Static Multipliers

CSA, NBW, WALL, RCA, and BKA architectures

- CSA, NBW, and WALL for DC
- BKA for BuildGates (but not BGX)

Conformal ASIC capability

- Automatically analyze and resolve unmatched architectures
- No user input required to handle static multipliers

Dynamic Multipliers

Dynamic multipliers with module boundary

- Conformal ASIC can analyze and resolve unmatched architectures
  - Use command "`analyze multiplier`" in LEC mode

Transcript window

```
...
// Command: set system mode lec
// Command: analyze multiplier
...
```

  - Or use "`set multiplier option –auto`" in SETUP mode

Transcript window

```
...
// Command: set multiplier option -auto
// Command: set system mode lec
...
```

Dynamic Multipliers (continued)

Ungrouped (flattened) dynamic multipliers

- Conformal Ultra is required
- Analyze and resolve unmatched architectures with command "`analyze datapath`"

Transcript window

```
...
// Command: set system mode lec
// Command: analyze datapath
...
```

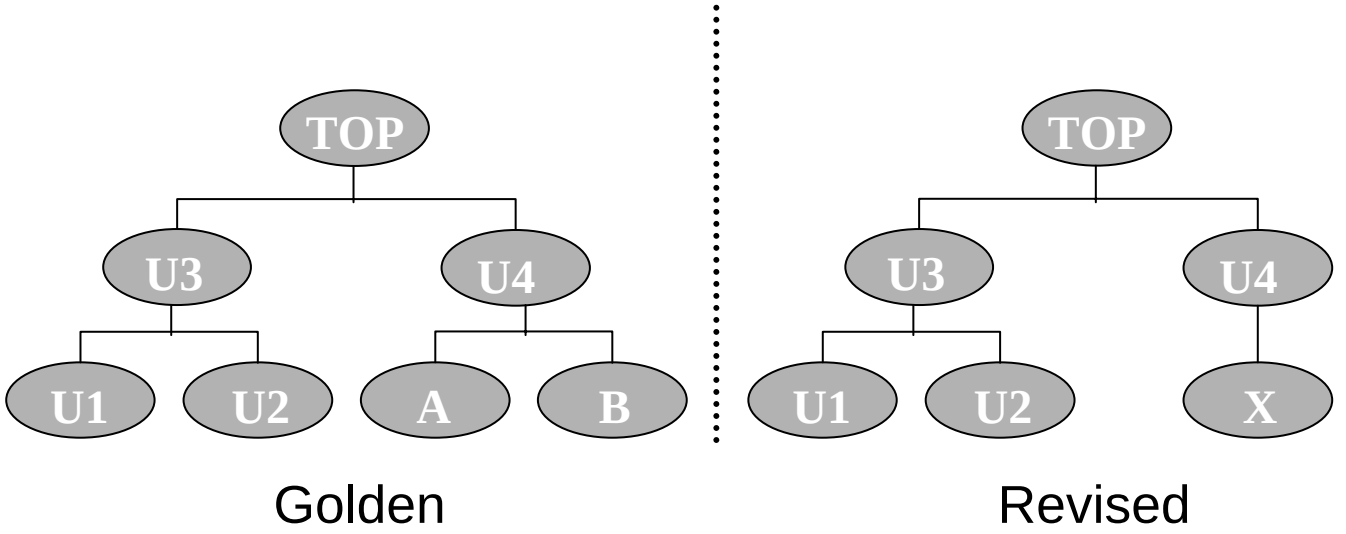
Still Long Runtime Because of Multipliers?

- 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

Hierarchical Compare

December 19, 2004

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.

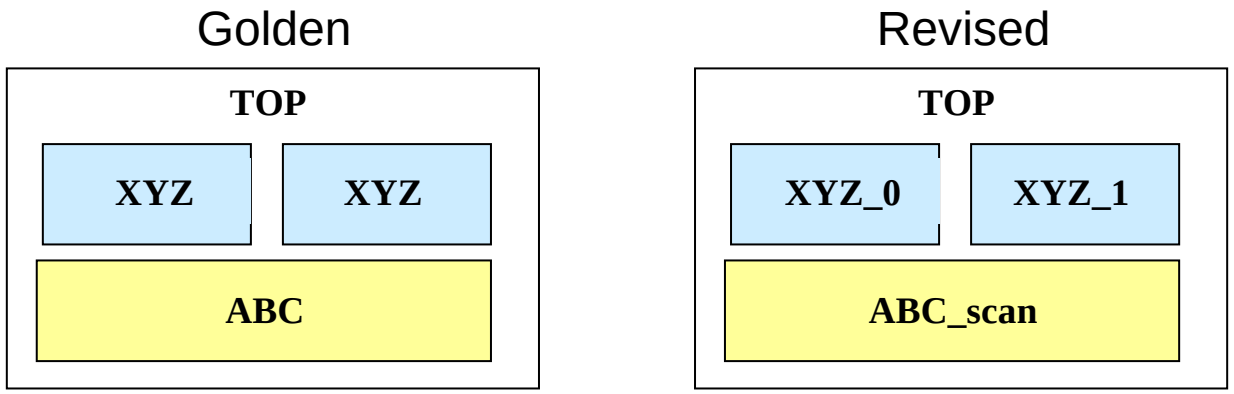
Hier Compare Command and Options

```
write hier dofile hier.do -prepend_string strings \
-append_string strings -threshold <value> -module \
<golden_module revised_module> -usage -replace \
-noexact_pin_match -constraint
```

- `-prepend_string`: append commands to the hier dofile script before key point comparison of each module
- `-append_string`: append commands to hier dofile script after key point comparison of each module
- `-threshold`: minimum number of primitives within a module that will be written to hier dofile script. Default value is 50
- `-module`: create dofile for this module and below, only
- `-noexact_pin_match`: attempt to write out modules even if pin names are not exact
- `-usage`: append the `usage` command after each module comparison
- `-replace`: overwrite the existing hier.do file
- `-constraint`: propagate top-level constraints to lower level modules
- For more options, please refer to the Reference manual

Module Name Mismatch

Module names change due to uniquification during synthesis or back-end flow



Make module names match to include them in hierarchical compare

- Use command "`uniquify –all –nolibrary -golden`" to have Conformal uniquify the Golden modules so that module names match with the Revised.

Hierarchical Compare Result

Transcript window

```
// Command: report hier_compare result

Total Equivalent modules       = 3
Total Non_equivalent modules   = 1

Hierarchical compare : Non-equivalent
// Command: report hier_compare result -Non_equivalent
Status          Golden                              Revised
-------------------------------------------------------------------------
Non_equivalent: arbr4                               arbr4
Total Non_equivalent modules   = 1
// Command: report hier_compare result -Abort
// Command: report hier_compare result -Uncompared
```

Debug NEQ Modules

Transcript window

```
…
// Command: report hier_compare result

Total Equivalent modules       = 3
Total Non_equivalent modules   = 1

Hierarchical compare : Non-equivalent
// Command: report hier_compare result -Non_equivalent
Status          Golden                                Revised
-------------------------------------------------------------------------
Non_equivalent: arbr4                                 arbr4
Total Non_equivalent modules   = 1
// Command: report hier_compare result -Abort
// Command: report hier_compare result –Uncompared
SETUP> set root module arb4 –both
SETUP> set system mode lec
LEC>   add compare point –all
LEC>   compare
```

What can go wrong?

Manual efforts needed to resolve false NEQ

- Diagnose the modules causing non-equivalence
- Manually add NOBLACK BOXes
- Regenerate dofile and rerun comparison

Want *more* module comparisons *without* false NEQ

Difficult to handle aborted modules

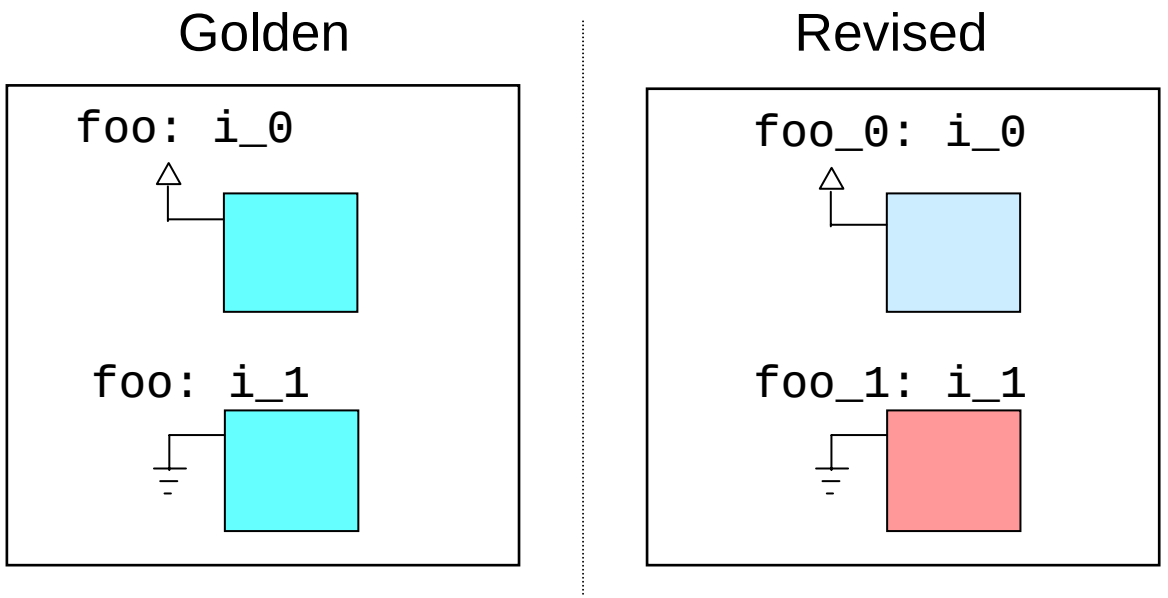
- Regenerate the dofile script for each of the aborted modules
- Modify dofile with different compare options to resolve abort

Need better control of the run

- Cannot interrupt and continue hierarchical comparison
- Hierarchically compare retimed modules only
- Stop at the first non-equivalent or abort result

Non-Compatible Instantiation: Problem

Conflicting instantiation: will skip module foo



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

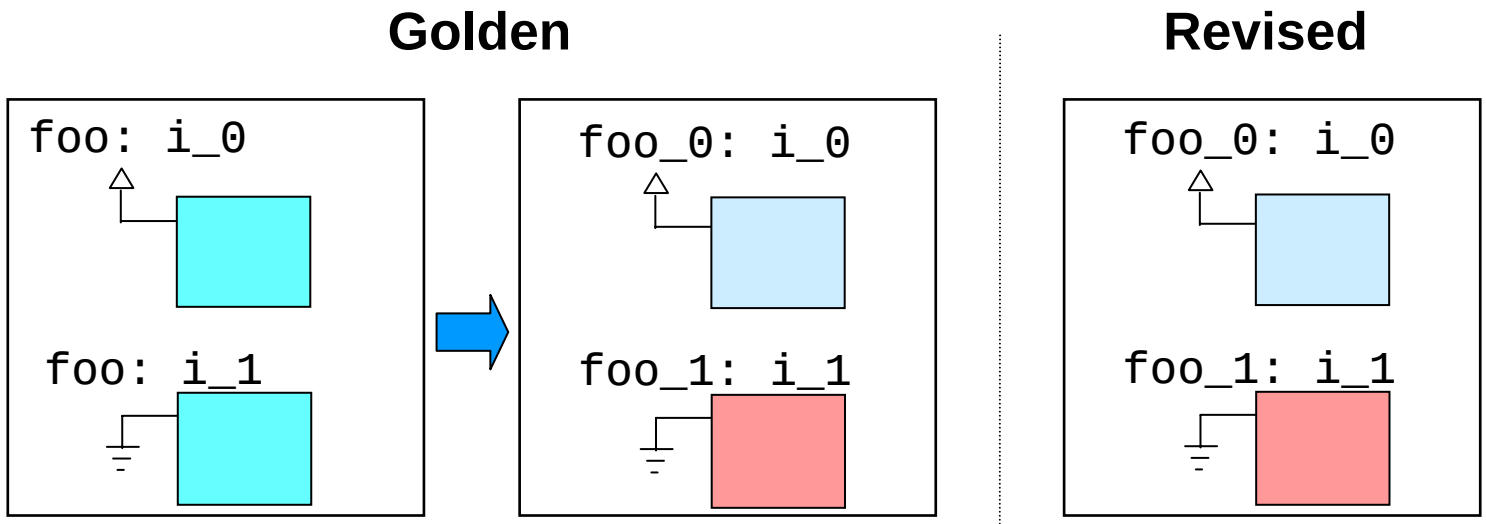
Transcript Window

```
...
// 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 the "uniquify" command.

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

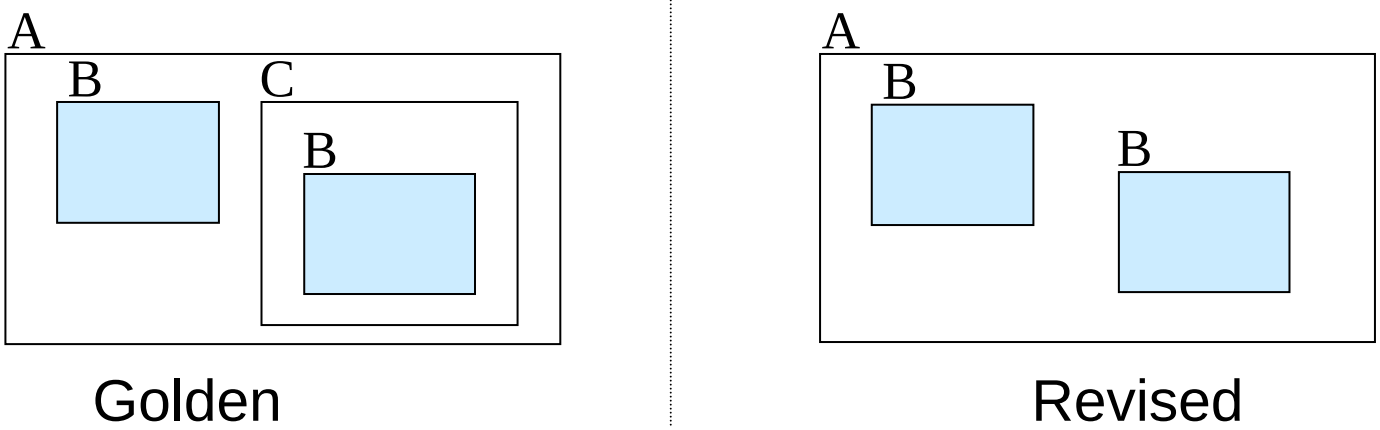


Transcript Window

```
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
```

Unbalanced Instantiation: Problem

Hierarchical script will skip module B



Golden

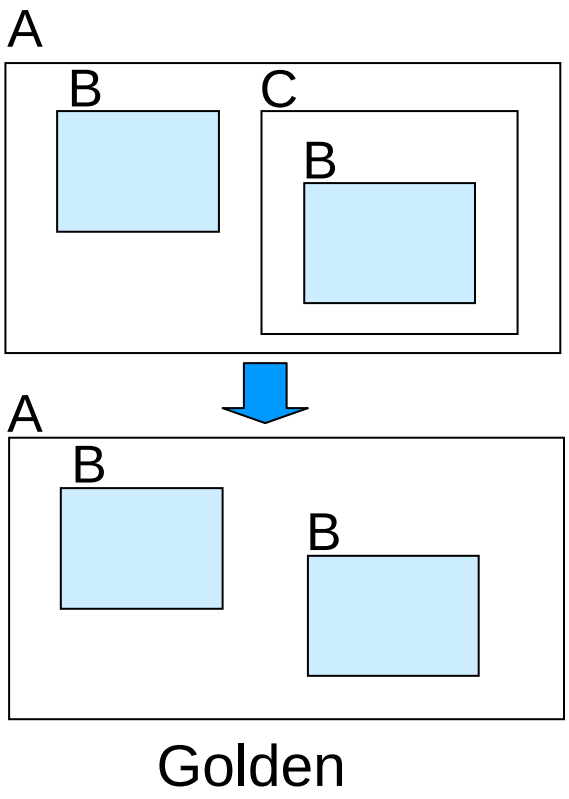
Revised

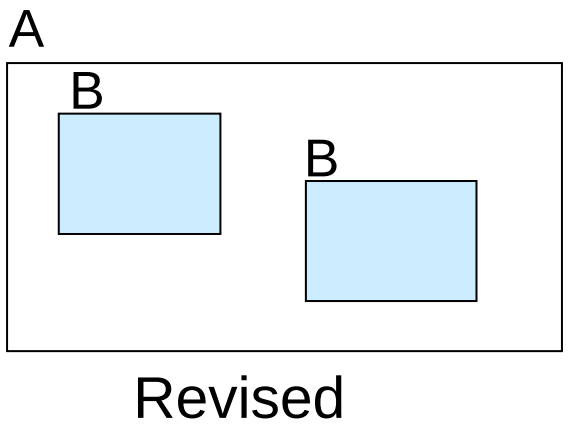
SETUP> `write hier dofile hier.dofile -constraint -noexact`

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
...
```

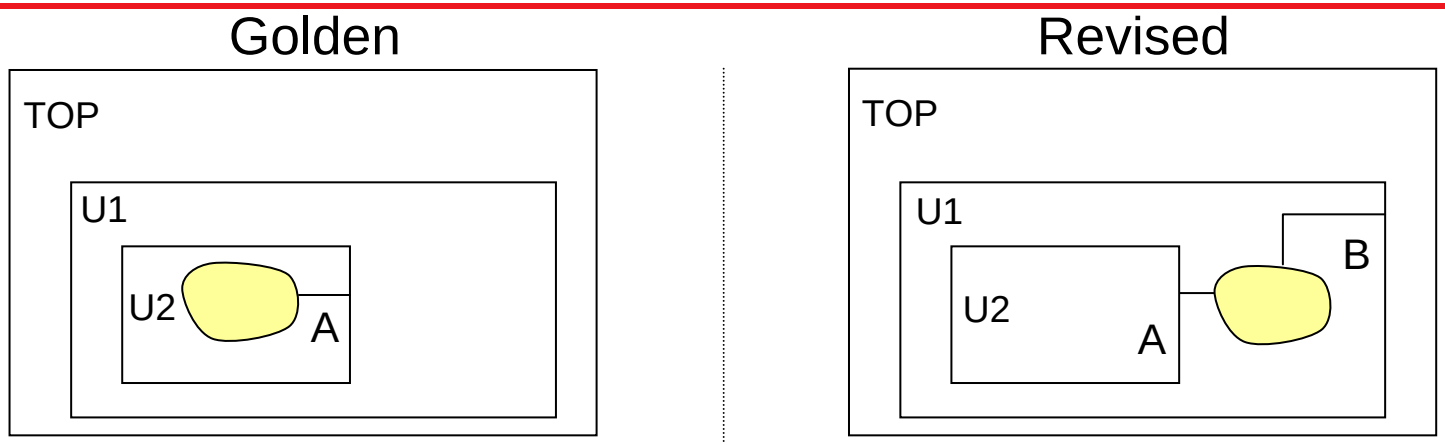
Unbalanced Instantiation: Solution





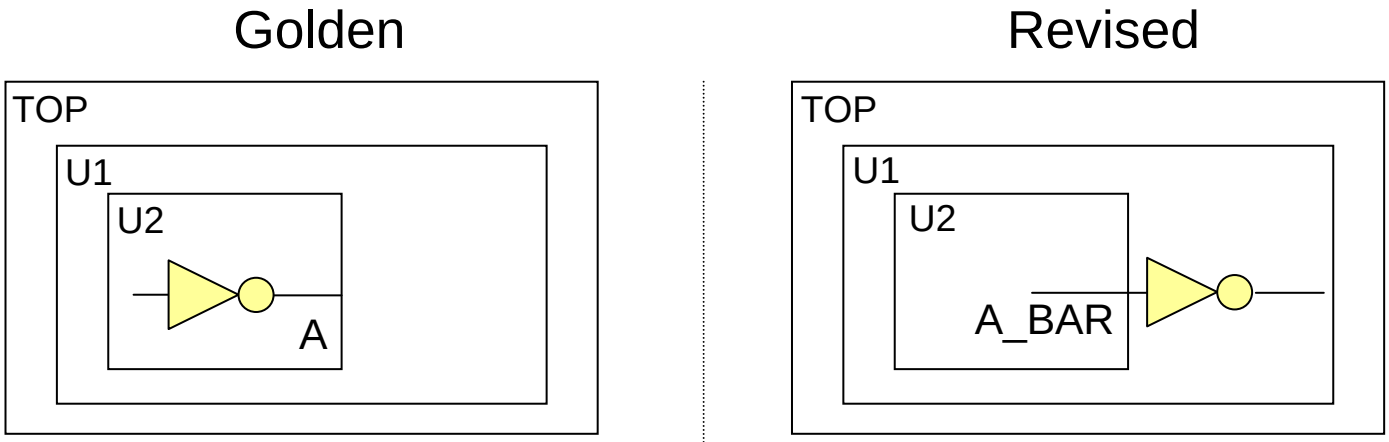

```
read design rtl.v -verilog -golden
read design -file verilog.vc gate.v -ver -rev
add pin constraint 0 scan_en -revised
resolve C -golden
write hier dofile hier.dofile -constraint -replace
dofile hier.dofile
```

Cross-Boundary Optimization



- Logic moved across module boundary
- False negative during hierarchical compare
  - Miscompare U2 at point A and U1 at point B
- Two forms of logic rearrangement:
  - Simple inverter
  - All other cases

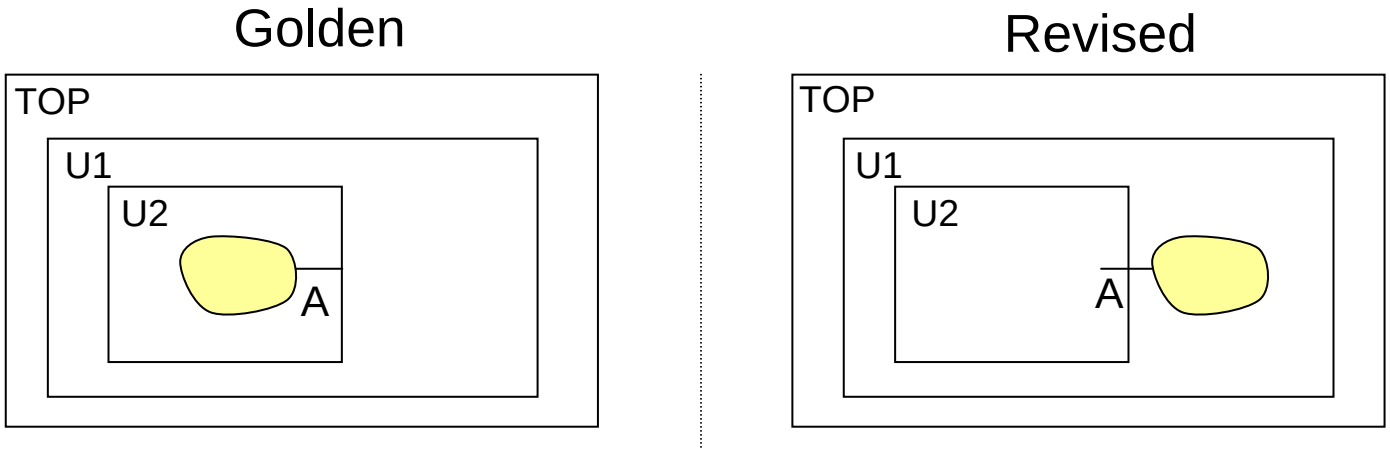
Logic Rearrangement - Case 1: Simple Inverter



- Inverters moved across module boundary
- Must set naming rule before read library/design
- Conformal 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: Random 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
```

Dynamic Hierarchical Comparison

A new method for performing hierarchical comparison

New command:

```
SETUP> RUN HIER_compare <dofile>
```

Available in LEC version 5.2

Requires Ultra license

Provides advanced capabilities and is easy to use

- Generate dofile only once
- Automatically add noblack box
- Easy to compare any sub-module
- Can interrupt with *Control-C* and continue
- Stop at the first non-equivalent or abort result
- Integrated with analyze abort

New Hierarchical Comparison Flow

Old Flow

```
write hier_compare dofile hrc.do
dofile hrc.do
// Diagnose cause of NEQ
add noblack box bad_module
write hier_compare dofile hrc.do -rep
dofile hrc.do
// Check for aborted modules
// Edit hrc.do or generate new dofile
dofile hrc.do.abort
```

New flow

```
write hier_compare dofile hrc.do
run hier_compare hrc.do
run hier_compare hrc.do <more options>
```

Command Syntax

```
SETUP> RUN HIER_COMPARE <dofile>
   [-DYNamic_hierarchy | -NODYNamic_hierarchy]
   [-ROOT_module <golden module> <revised module>]
   [-ANALYZE_abort] [-RETIMED_modules]
   [-NOREStart | -REStart]
   [-BREAK_NONEQ ] [-BREAK_ABORT]
```

<dofile> is generated using the command:

```
WRITE HIER_COMPARE dofile
```

Blank Page

cadence®

Using Conformal for Complex Datapath Verification

December 19, 2004

Difficulties in Datapath Design Verification

Compare Issues – abort or long runtimes

- Multipliers
- Expressions - Operator Merging

Advanced EC Issues – Non-equivalent in static EC

- Pipeline retiming
  - Designs have different number of state elements
  - Logic cones of pipelined registers are different

Multipliers

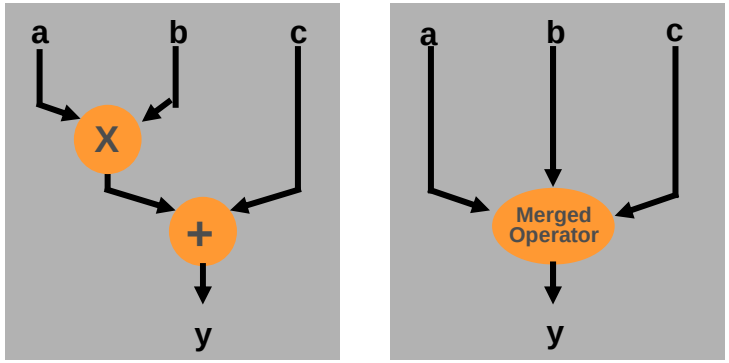
- Unmatched multiplier architectures can lead to abort
- Static multipliers
  - CSA, RCA, NBW, WALL, and PKA
  - Can be handled by Conformal ASIC
- Dynamic multipliers
  - Multiplier with module boundary can be handled by Conformal ASIC
  - Flattened multipliers require Conformal Ultra

Expressions – Operator Merging

Combine addition-based operators into a carry-save tree

- +, -, *, <, >, <=, >=, ==, !=

Example: y = a * b + c



Intermediate a*b result not required

Expression – Operator Merging (continued)

Use the Conformal Ultra datapath analysis command:

"`analyze datapath –merge`"

Transcript window

```
...
// Command: set system mode lec
// Command: analyze datapath –merge
...
```

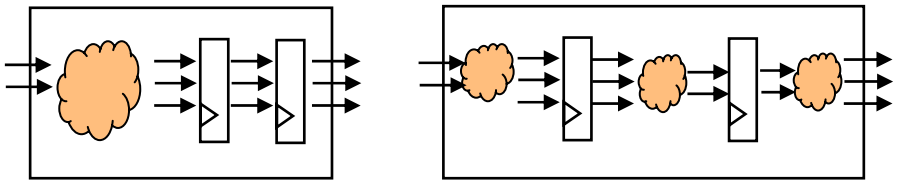
Pipeline Retiming

- Pipeline retiming is non-equivalent in static EC
  - Different number of state points
- Simple pipeline retiming
  - Can be handled with Conformal ASIC
- Advanced pipeline retiming
  - Required Conformal Ultra

Simple Pipeline Retiming

Guideline for simple pipeline retime circuit

- All data must move from register stage to register stage
- All paths must have the same number of stages
- All pipeline registers must have the same clock
- Sequential loops or latches are prohibited
- Asynchronous set and reset should be disabled
- Stalls should be disabled
- Compatible with DC's pipeline_design command



Golden Revised

Handling Simple Pipeline Retiming Circuit

- Pipeline retimed module must be root module
- Can use Conformal ASIC command

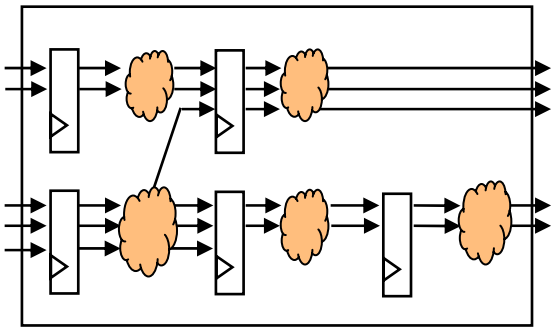
```
add module attribute <module> –pipeline_retime [-
dff2buffer] [-golden | -revised]
```

Transcript window

```
// ...
// Command: set root module ocd_cs_r2c –both
// Command: add pin constraint 0 stall -both
// Command: add module attribute ocd_cs_r2c –pipeline_retime -golden
// Command: set system mode lec
// ...
// Modeling Golden ...
// Pipeline-retimed 50 DFF(s) as 50 DFF(s) in 3 stages
// Modeling Revised ...
// Pipeline-retimed 362 DFF(s) as 50 DFF(s) in 3 stages
```

Advanced Pipeline Retiming

- Conformal Ultra license required
- Supports
  - Each part of the module has different number of stages



  - Some forms of set and reset (1 async signal, set or reset)
  - Sequential feedback
  - MUX enable, but not MUX stall
  - Latches that are not retimed

Example dofile (Flat Compare)

```
// Read libraries and designs
read library …
read design golden.v  –golden
read design revised.v –revised

// Specify design constraints, pipeline retime
module
add pin constraint 0 stall –both
add module attribute ocd_cs_r2c –pipeline_retime

// To automate modeling and mapping process
set system mode lec

// To specify datapath analysis
analyze datapath –merge

// To invoke key points comparison
add compare point –all
compare
```

Hierarchical Compare

Use

```
set datapath option –auto -merge
```

instead of `analyze datapath -merge`

- `analyze datapath` is used in flat compare in LEC mode
- -auto: Call `analyze datapath` during `set system mode lec`
- -merge: Use merge option for `analyze datapath`

Example dofile (Hierarchical Compare)

```
// Read libraries and designs
read library …
read design golden.v  –golden
read design revised.v –revised

// Specify design constraints, pipeline retime module
add pin constraint 1 hold –both
add module attribute ocd_cs_r2c –pipeline_retime

// To specify datapath analysis
set datapath option -auto
write hier dofile hier.do –noexac_pin_match –constraint –replace
dofile hier.do
```

cadence®

RTL Coding Guidelines for Easy Equivalence Checking

December 19, 2004

Module Objectives

In this module, you will

- Be introduced to coding guidelines that make equivalence checking easier

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 a case statement.
- Use more assign statements.
- Avoid combinatorial feedback loops.

If you follow the above guidelines it will reduce problems during the EC process.

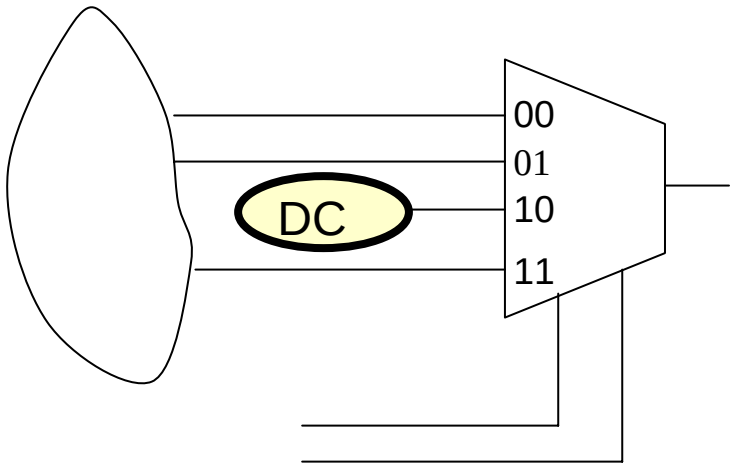
Keep Difficult Arithmetic Blocks Separate

- 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: Problem

- Applying synthesis full_case directive or X assignments on incompletely specified case statements creates "don't care" (DC) conditions in Conformal LEC
- Conformal LEC creates DC circuitry to ignore comparison of unspecified states
  - Starting from version 4.3.0.a, Conformal creates E gate for X in Revised design
- DC circuitry grows with greater numbers of unspecified states
- Bigger DC circuitry leads to longer Conformal 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
```

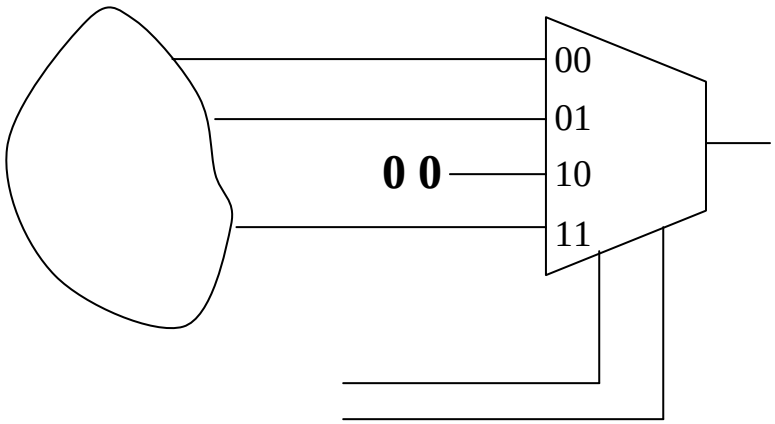


Assign Known Value to All Cases: Solution

- Use "default" with known value to catch all unspecified conditions
- If you use default with the known value it 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.*

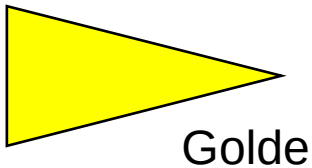
```
Module xyz (sm, out);
   input [1:0] sm;
   output out;
   reg out;
   always @(sm)
     case (sm)
         2'b00: out = 2'b01;
         2'b01: out = 2'b10;
         2'b11: out = 2'b00;
         default = 2'b00;
     endcase
endmodule
```



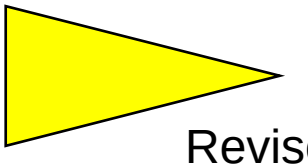
Using More assign Statements

- 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);
```



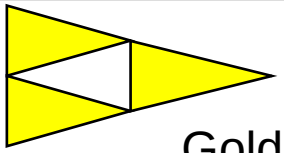
Golden



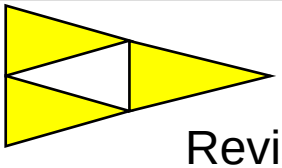
Revised

Before

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



Golden



Revised

After

Avoid Combinatorial Feedback

- Combination feedback loops can hinder verification process
- Conformal 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, you must specify where to insert cut gate


cadence