SYNOPSYS®
Predictable Success

Debugging and Resolving Inconclusive Verifications

Uday Dixit

Joe Bosia

Formality CAE

Copyright Notice

CONFIDENTIAL INFORMATION

The following material is being disclosed to you pursuant to a non-disclosure agreement between you or your employer and Synopsys. Information disclosed in this presentation shall be used only as permitted under such an agreement.

LEGAL NOTICE

Information contained in this document reflects Synopsys plans as of the date of this document. Such plans are subject to completion and are subject to change. Products may be offered and purchased only pursuant to an authorized quote and purchase order. Synopsys is not obligated to develop the software with the features and functionality discussed in the materials.

SYNOPSYS® Predictable Success

Debugging Inconclusive Verifications Three Step Approach

Agenda

- Datapaths and Inconclusive Verifications (Quick Overview)
- Inconclusive Debug: 3-Step Approach
  - Find **List** of Hard Points (We want list to be as small as possible)
  - Find **Cause** of the Hard Points
  - Find **Resolution** of Hard Points

SYNOPSYS® Predictable Success

What is a Inconclusive Verification?

- Verification session which Equivalence Checker cannot completely verify all compare points
  - Transcript shows no apparent progress for many hours or days
  - Verification aborts or stops due to design complexity
- Usually involves datapath designs
  - DC Ultra produces complex, highly optimized, datapath blocks
- Sometimes due to non-datapath causes
  - CRC, parity generators, XOR trees
  - Very large complex logic cones

SYNOPSYS® Predictable Success

Example of Inconclusive Verification

Appears to be hung or stuck

```
Status:  Matching...

********************************** Matching Results **********************************
 32 Compare points matched by name
 0 Compare points matched by signature analysis
 0 Compare points matched by topology
 96 Matched primary inputs, black-box outputs
 0(0) Unmatched reference(implementation) compare points
 0(0) Unmatched reference(implementation) primary inputs, black-box outputs
 15(0) Unmatched reference(implementation) unread points
******************************************************************************************

Status:  Verifying...

Status:  Matching hierarchy...

Status:  Pre-verifying recovered datapath block vs reference r:/WORK/test/mult_5 (/WORK/M_RTL_MULT_UNS_16_16).
    Pre-verification of recovered datapath block SUCCEEDED
    Number of unmatched datapath blocks found: 1
    Use 'report_unmatched_points -datapath' to report unmatched datapath blocks.
    Use 'set_user_match' to match unmatched datapath blocks.


Status:  Verifying...
............................ 0F/0A/3P/29U (9%) 02/22/07 23:27 159MB/1747.60sec
............................ 0F/0A/3P/29U (9%) 02/22/07 23:57 200MB/3536.84sec
............................ 0F/0A/3P/29U (9%) 02/23/07 00:33 200MB/5665.90sec
............................ 0F/0A/3P/29U (9%) 02/23/07 01:03 266MB/7379.06sec
............................ 0F/0A/3P/29U (9%) 02/23/07 01:33 266MB/9136.15sec
............................ 0F/0A/3P/29U (9%) 02/23/07 02:03 266MB/10921.82sec
............................ 0F/0A/3P/29U (9%) 02/23/07 02:33 266MB/12664.95sec
............................ 0F/0A/3P/29U (9%) 02/23/07 03:06 266MB/14637.13sec
............................ 0F/0A/3P/29U (9%) 02/23/07 03:36 266MB/16426.55sec
............................ 0F/0A/3P/29U (9%) 02/23/07 04:06 266MB/18138.65sec
............................ 0F/0A/3P/29U (9%) 02/23/07 04:36 266MB/19889.62sec
```

- <u>Unverified</u> compare points if verification is interrupted

SYNOPSYS® Predictable Success

Example of Inconclusive Verification

Aborted due to complexity

```
    Compare point mix[16] is aborted
    Compare point mix[17] is aborted
    Compare point mix[18] is aborted
    Compare point mix[19] is aborted
    Compare point mix[20] is aborted
    Compare point mix[21] is aborted
    Compare point mix[22] is aborted
    Compare point mix[23] is aborted
    Compare point mix[24] is aborted
    Compare point mix[25] is aborted
    Compare point mix[26] is aborted
    Compare point mix[27] is aborted
    Compare point mix[28] is aborted
    Compare point mix[29] is aborted
    Compare point mix[30] is aborted
    Compare point mix[31] is aborted

********************************** Verification Results **********************************
Verification INCONCLUSIVE
(Equivalence checking aborted due to complexity)
-----------------------------------------------
 Reference design: r:/WORK/test
 Implementation design: i:/WORK/test
 3 Passing compare points
 29 Aborted compare points
 0 Unverified compare points
-----------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
-----------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0       3       0       0       3
Failing (not equivalent)       0       0       0       0       0       0       0       0
Aborted
  Hard  (too complex)          0       0       0       0      29       0       0      29
*****************************************************************************************
```

- Verification was attempted on all compare points
- Aborted compare points

SYNOPSYS® Predictable Success

What About Performance?

- Some verifications may have successful results, but take a very long time
- The same debug techniques used for investigating Inconclusive verifications may be used to try to speed up successful verifications

SYNOPSYS® Predictable Success

What Is A Datapath?

- Any circuit that contains arithmetic components
  - Adders/Subtractors
  - Multipliers/Dividers
  - Comparators
- Can contain a limited set of ‘support’ components
  - Selectors
  - Shifters
- Generally register and/or block bounded

SYNOPSYS® Predictable Success

The Datapath Verification Problem

- Why are Datapaths difficult to verify ?
  - Very large cones of logic
  - Reference and implementation designs typically dissimilar
- DC Ultra creates highly optimized datapath
  - Large multipliers and merged operators create large dissimilar cones

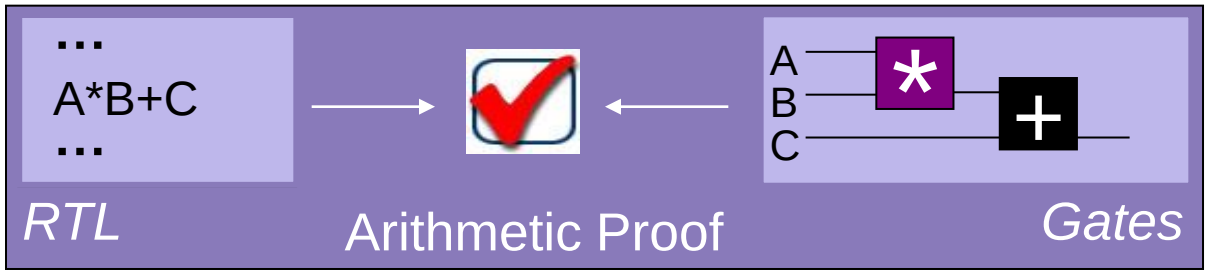
Complex datapath - the most difficult equivalency checking completion issue

SYNOPSYS® Predictable Success

Formality Datapath Verification Strategy

Three Techniques

- Key to Solution -> SVF flow restructures view of RTL to resemble DC gates
  - SVF guidance provides recipe of important optimizations
  - The more similar the design, the easier the verification
- Datapath Solver
  - Relies on finding, matching, and formally proving gates versus RTL datapath expression



  - Verifies a netlist implements a mathematical expression for every possible input and output
- General solver technology

SYNOPSYS® Predictable Success

Datapath Verification Flow

Summary

SVF transforms the FM view of RTL to resemble the operator structure used by DC

- Read transformation from SVF
- Prove that transformation is valid
- Apply transformation to RTL container
  - If invalid, reject transformation
- Continue with all transformations
- Verify design

SYNOPSYS® Predictable Success

Debugging Inconclusive Verifications Three Step Approach

Agenda

- Inconclusive Debug: 3-Step Approach
  - Find **List** of Hard Points (We want list to be as small as possible)
  - Find **Cause** of the Hard Points
  - Find **Resolution** of Hard Points .

SYNOPSYS® Predictable Success

Find List of Hard Points

Techniques to get minimized list of Hard Points quickly

- Hierarchical Script
  - (-path option and other techniques)
- Re-verification
  - Timeout Limits , Effort Levels
- Helpful commands…. (Remember to use them)
  - report_unverified_points
  - report_aborted_points
  - save_session

Note: None of this is new, but all worthy of mention

SYNOPSYS® Predictable Success

Find List of Hard Points

Hierarchical Script

- If Hierarchies are available on ref and impl, take advantage of them to isolate your hard block(s)
- At very least can hone in on smaller list of unverified points, and potentially solve the verification outright
- write_hierarchical_verification_script
  - Use to isolate specific hierarchical block(s)
  - Use –level <#int> OR -path to get to block level of interest
    - `set verification_timeout_limit` <reasonable_limit>
    - Source <script>
    - May have to deal with false differences

SYNOPSYS® Predictable Success

Find List of Hard Points

Hierarchical Script

- -path switch

  Specify a list of instance pathnames to any ref or impl design object (cell,port,net,pin)

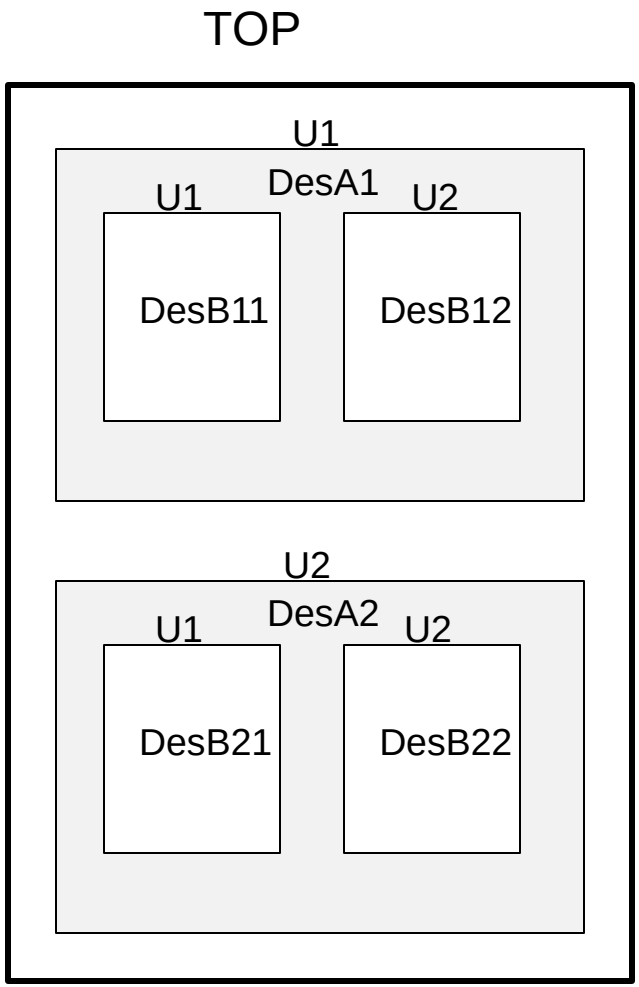
  Allows minimal script to be written out for targeting just the blocks of interest (those which contain your hard points)

- Use `"set_param -flatten"` on blocks with false differences and regenerate hierarchical script
- If you can achieve success with a hierarchical script, but top-down default is incomclusive , please file a star

SYNOPSYS® Predictable Success

Find List of Hard Points

Hierarchical Script



Scenerio: We have a hard point in DesB12, so we want to isolate that level of hierarchy

```
write_hier  hier-path1 -path $ref/u1/u2
```

Verifying block i:/WORK/DesB12 vs r:/WORK/DesB12...
Verifying block i:/WORK/DesA1 vs r:/WORK/DesA1...
Verifying block i:/WORK/top vs r:/WORK/top

Scenerio: We have a hard point in DesB12, above gave us false failures

```
write_hier  hier-path1 -path $ref/u1/
```

Verifying block i:/WORK/DesA1 vs r:/WORK/DesA1...
Verifying block i:/WORK/top vs r:/WORK/top

Goal is Isolation

SYNOPSYS® Predictable Success

Find List of Hard Points

Techniques to get minimized list of Hard Points quickly

- Hierarchical Script
  - (-path option and other techniques)
- Re-verification
  - Timeout Limits , Effort Levels
- Helpful commands…. (Remember to use them)
  - report_unverified_points
  - report_aborted_points
  - save_session

Note: None of this is new, but all worthy of mention

SYNOPSYS® Predictable Success

Find List of Hard Points

Re-verification

How can we make use of this ?

- Given a hard verification with a known quantity of hard points after some long run time OOTB…..
- Our goal is to get to at least that set of hard points (or less) in some reasonable (fixed) amount of time
- Using re-verifications allows for different solver deployments and hopefully we are able to obtain our goal

SYNOPSYS® Predictable Success

Find List of Hard Points

Re-Verification

Any technique that results in a re-verification can have a similar effect, such as

- `verification_time_out_limit`
- `CTRL-C`
- `verification_effort_level`

SYNOPSYS® Predictable Success

Find List of Hard Points

Re-verification

Example Using Effort Levels

```
set verification_effort_level super_low
verify
set verification_effort_level low
verify
set verification_effort_level medium
verify
set verification_effort_level high
verify
```

Or simply:

```
set verification_effort_level medium
verify
set verification_effort_level high
verify
```


SYNOPSYS® Predictable Success

Find List of Hard Points

Re-verification

Example Using Verification Timeout Limits

```
set verification_timeout_limit 8:0:0
verify;verify;verify  -- try to change partition makeup in first 24 hrs.
set verification_timeout_limit 0
verify                -- unlimited verification on the rest
```


SYNOPSYS® Predictable Success

Find List of Hard Points

Re-verification

- There is no guarantee these methods will give desired result, it may even be worse
- Given a fixed amount of time, it is very difficult to know what is the best combination of timeout or effort level along with re-verification to get to the minimum possible number of unverified points

SYNOPSYS® Predictable Success

Find List of Hard Points

Techniques to get minimized list of Hard Points quickly

- Hierarchical Script
  - (-path option and other techniques)
- Re-verification Utilizing Timeouts
  - Timeout Limits, Effort Levels
- Helpful commands…. (Remember to use them)
  - report_unverified_points
  - report_aborted_points
  - save_session

Note: None of this is new, but all worthy of mention

SYNOPSYS® Predictable Success

Find List of Hard Points

Make Use of Reports

- Starting point for debugging HVs is looking at the actual hard points
- Make sure to use the following in your scripts
  - `report_unverfied_points`
  - `report_aborted_points`
  - `save_session`

SYNOPSYS® Predictable Success

Find List of Hard Points

Make Use of Reports

- Use them both, they are different
  - `report_aborted_points > aborts.rpt`
  - `report_unverified_points > unver.rpt`
- Use them often
  - Use in conjunction with any timeouts you have
- Create session file
  - We can then generate the reports ourselves
- Formality now has a default verification timeout limit of 36 hours
  - By default saves session file after each user defined timeout limit reached with a name “formality_timeout_session.fss”

SYNOPSYS® Predictable Success

Find List of Hard Points

Points to remember…

- Techniques shows in previous slides like
  - Hierarchical verification
  - Re-Verification using small timeouts
- Those are tools to debug hard verification and find least number of hard to verify points quickly
- It could slow down the verification if you add those in the default Formality script

SYNOPSYS® Predictable Success

Debugging Inconclusive Verifications Three Step Approach

Agenda

- Inconclusive Debug: 3-Step Plan
  - Find **List** of Hard Points (We want list to be as small as possible)
  - Find **Cause** of the Hard Points
  - Find **Resolution** of Hard Points

SYNOPSYS® Predictable Success

Find Cause of Hard Points

SVF Debug Commands

Techniques to find cause of Hard Points quickly

- SVF debug commands
  - report_guidance –summary
  - report_svf_operation
- analyze_points
- Schematics

SYNOPSYS® Predictable Success

Find Cause of Hard Points

SVF Debug Commands

- Command **`report_guidance -summary`** gives condensed status summary of all SVF guidances

- Why look at this First ?

- To see if there is any “global” issue that should be dealt with. For instance, if you see :

```
***************************** Guidance Summary ******************************
                                          Status
Command                  Accepted   Rejected  Unsupported  Unprocessed  Total
-----------------------------------------------------------------------------
architecture_netlist:          0          1            0            0      1
datapath             :         0         87            0            0     87
```

It is clearly some global setup issue……

SYNOPSYS® Predictable Success

Find Cause of Hard Points

More likely scenario is handful of hard points and a summary as follows

```
***************************** Guidance Summary ******************************
                                          Status
Command                   Accepted   Rejected  Unsupported  Unprocessed  Total
-----------------------------------------------------------------------------
architecture_netlist:          1          0          0           0          1
datapath             :        79         22          0           0        101
scan_input           :         1          0          0           0          1
transformation
   map               :        12          0          0           0         12
   merge             :        87         14          0           0        101
   share             :        23          2          0           0         25
   tree              :        13          0          0           0         23
ungroup              :         4          0          0           0          4
uniquify             :         2          0          0           0          2
```

So, Do we care about ALL 22 rejected Datapaths ??

No !, Only care about the “one” in cone of hard points

SYNOPSYS® Predictable Success

Find Cause of Hard Points

report_svf_operation

- Reports SVF operations based on command name and/or operation status

```
report_svf_operation [ -command
 command_name] [ -status status_name ]
 <object id>
```

- Allows you to debug on a CONE level for a particular hard point

SYNOPSYS® Predictable Success

Find Cause of Hard Points

report_svf_operation

- Use `report_svf_operation` on the ref object ids of your hard points

  Few examples…

  `report_svf_operation ref:/WORK/test/out[2] -summary`
  - Reports all the SVF operations which are in the fanin of the hard point

  `report_svf_operation {ref:/WORK/test/U3 ref:/WORK/test/U1} -status accepted -summary`
  - Reports all SVF operations which are accepted

  `report_svf_operation ref:/WORK/test/U* -status rejected -summary`
  - Reports all SVF operations which are rejected
  - This is what is typically done first
- Useful command for interactive debugging

SYNOPSYS® Predictable Success

Find Cause of Hard Points

report_svf_operation

- Use summary option first ……

```
report_svf_operation
  $ref:/WORK/hard_output[2] -summary
```

Operation	Line	Command	Status
18	207	transformation_map	accepted
19	218	transformation_map	accepted
30	339	transformation_map	accepted
38	420	transformation_merge	rejected
43	474	datapath	rejected

SYNOPSYS® Predictable Success

Find Cause of Hard Points

report_svf_operation

```
fm_shell (verify)> report_svf_operation 38
## SVF Operation 38 (Line: 420) - transformation_merge.  Status: rejected ## Operation Id: 38 guide_merge \
  -design { top } \
  -datapath { add_1506_DP_OP_258_1212_1 } \
  -input { 6 I1 } \
  -input { 14 I2 } \
  -input { 6 I3 } \
  -input { 6 I4 } \
  -input { 6 I5 } \
  -output { 22 O3 } \
  -pre_resource { { 20 } mul_1517 = MULT_TC { { I1 } { I2 } { 0 } } } \
  -pre_resource { { 12 } mul_1510 = MULT_TC { { I3 } { I4 } { 0 } } } \
  -pre_resource { { 21 } add_1506_2 = ADD { { mul_1517 ZERO 21 } { mul_1510 ZERO 21 } } } \
  -pre_resource { { 22 } add_1506 = ADD { { add_1506_2 ZERO 22 } { I5 ZERO 22 } } } \
  -pre_assign { O3 = { add_1506 } }
```


SYNOPSYS® Predictable Success

Find Cause of Hard Points

analyze_points command

Techniques to find cause of Hard Points quickly

- SVF debug commands
  - report_guidance –summary
  - report_svf_operation
- analyze_points
- Schematics

SYNOPSYS® Predictable Success

Find Cause of Hard Points

analyze_points command

- Determines possible causes for failing and inconclusive compare points
- `analyze_points`
  - Options: `-failing, -aborted, -unverified, -all`
  - Can take single or list of compare points as an argument
- Analyzes hard points to look for common causes
  - Modules with complex datapath where SVF has been rejected resulting in a hard verification
- It is a great command for batch jobs where you can have `analyze_points -all` run as part of the script
- Recommends next step

SYNOPSYS® Predictable Success

Find Cause of Hard Points

analyze_points command

- When used on hard points, it looks for datapath specific SVF operations in the fanin of the hard point
- Produces Design Compiler `set_verification_priority` commands that can be inserted into the DC Ultra script
  - targets specific block(s), instances, or arithmetic operators
  - turns off specific optimizations
  - improves verification success
  - minimizes QoR impact
- This gives better verifiability of new netlist

SYNOPSYS® Predictable Success

Find Cause of Hard Points

analyze_points command

GUI snapshot

udixit's X desktop (igcae089:12)

Formality (R) Console - Synopsys Inc.

File Edit View Designs Run Window Help

Verification Inconclusive

Reference: r:/WORK/top

Implementation: i:/WORK/top

0. Guidance | 1. Reference | 2. Implementation | 3. Setup | 4. Match | 5. Verify | 6. Debug

Failing Points | Passing Points | Aborted Points | Unverified Points | Probe Points | Analyses

Possible Failure Causes

- Hard Datapath Component Module (1)
  - r:/WORK/top
- Unconstrained Implementation Input (0)
- Unmatched Cone Input (0)
- Rejected Guidance Command (0)
- Undriven Reference Signal (0)
- Directly Undriven Reference Port (0)
- Unmatched Blackbox Net (0)
- Failing Blackbox Net (0)
- Unretimed DesignWare Component (0)
- Missing Retention Register (0)
- X Propagation to Implementation Compare Point (0)
- Rejected Datapath Guidance Module (0)

Description - Hard Datapath Component Module:
These modules contain arithmetic operators that may be contributing to hard verifications.
Lowering the Design Compiler optimization level for the these modules may permit verification to succeed.

Recommendations:
r:/WORK/top in file /remote/fmcae4/users/udixit/PRS/fm016_blend1/rtl/test.v
Module with datapath cell(s):

- r:/WORK/top/DP_OP_23J1_125_5602

Try adding the following command(s) to your Design Compiler script right before the first compile_ultra command:
current_design top
set_verification_priority [get_cells { add_28 mult_28 sub_28 }]
current_design top

```
current_design top

-----------
```

Log | Errors | Warnings | History | Last Command

Formality (verify)>

Ready | Shell State: verify

Terminal (5) | xterm | tcsh | Formality (R) Co | xterm | xterm | emacs: test.v<2 | Cone Schematic | Wed Mar 2 2:22 PM

SYNOPSYS® Predictable Success

Find Cause of Hard Points

analyze_points command

- Shell Example report

```
********************************** Analysis Results **********************************
Found 1 Hard Datapath Component Module
-------------------------------
These modules contain arithmetic operators that may be
contributing to hard verifications.
Lowering the Design Compiler optimization level for the these
modules may permit verification to succeed.
-------------------------------
r:/WORK/top in file /remote/fmcae4/users/udixit/rtl/test.v
   Module with datapath cell(s):
      r:/WORK/top/DP_OP_23J1_125_5602

    Try adding the following command(s) to your Design Compiler script right before the first compile_ultra command:
        current_design top
        set_verification_priority [ get_cells { add_28 mult_28 sub_28 } ]

        current_design top

-----------
-------------------------------
*******************************************************************************************
Analysis Completed
```


SYNOPSYS® Predictable Success

Find Cause of Hard Points

Schematics

Techniques to find cause of Hard Points quickly

- SVF debug commands
  - report_guidance –summary
  - report_svf_operation
- analyze_points
- Schematics

SYNOPSYS® Predictable Success

Find Cause of Hard Points

Schematics

- Schematics are useful to identify don't care space or selector logic in the fanin of the hard points
- They help to identify candidates for manual factoring
- Most useful when hierarchy grouping is used

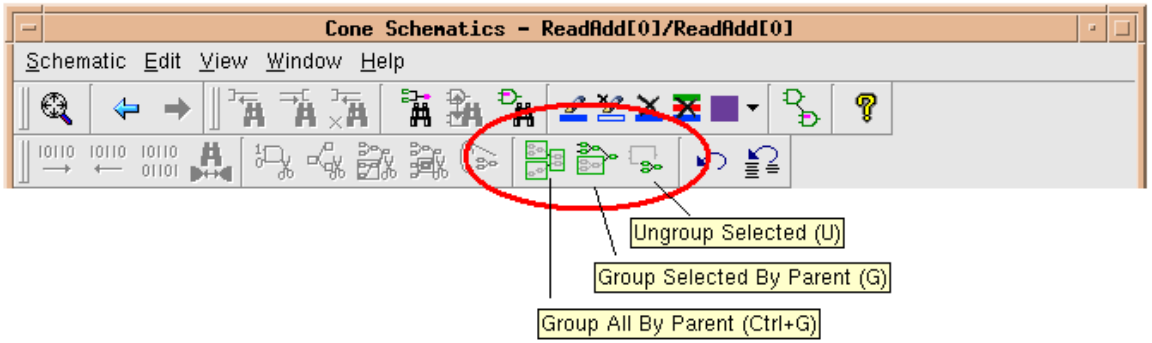
SYNOPSYS®
Predictable Success

Find Cause of Hard Points

Schematics

Hierarchical Grouping

- Perform hierarchical grouping in a logic cone view
- Group all, Group by selected, or Ungroup Selected

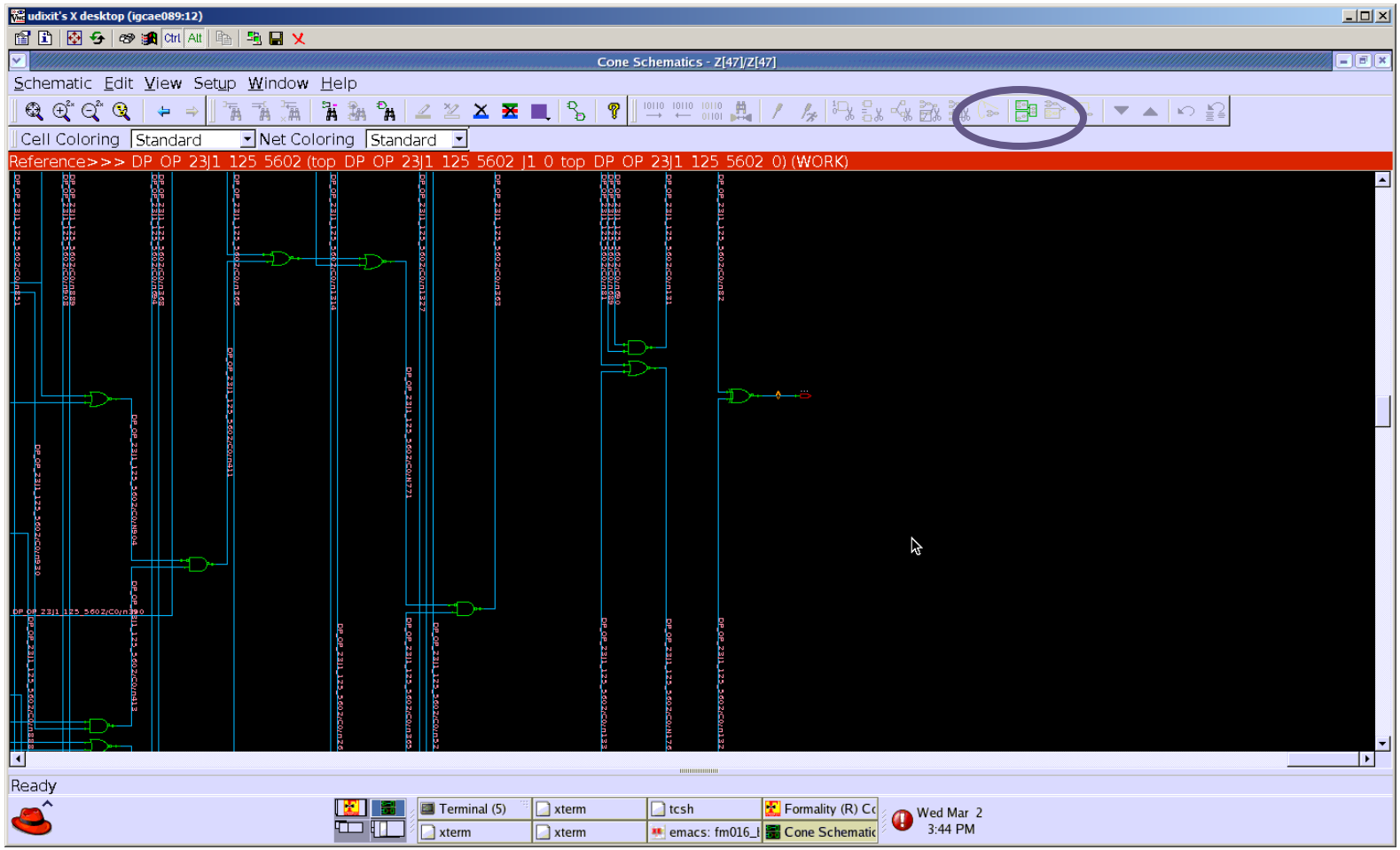


SYNOPSYS® Predictable Success

Find Cause of Hard Points

Schematics

Hierarchy Grouping

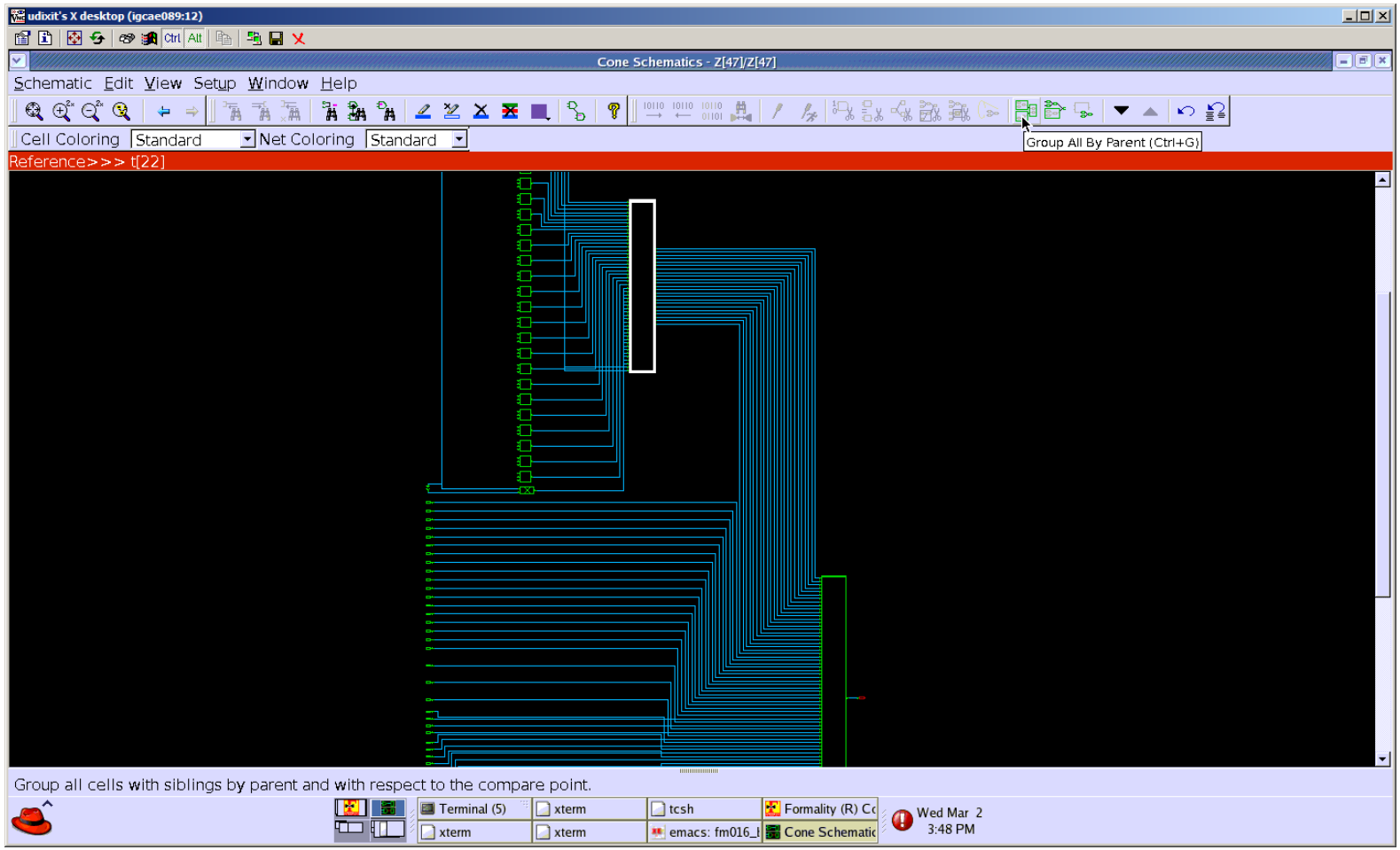


SYNOPSYS®
Predictable Success

Find Cause of Hard Points

Schematics

Hierarchy Grouping



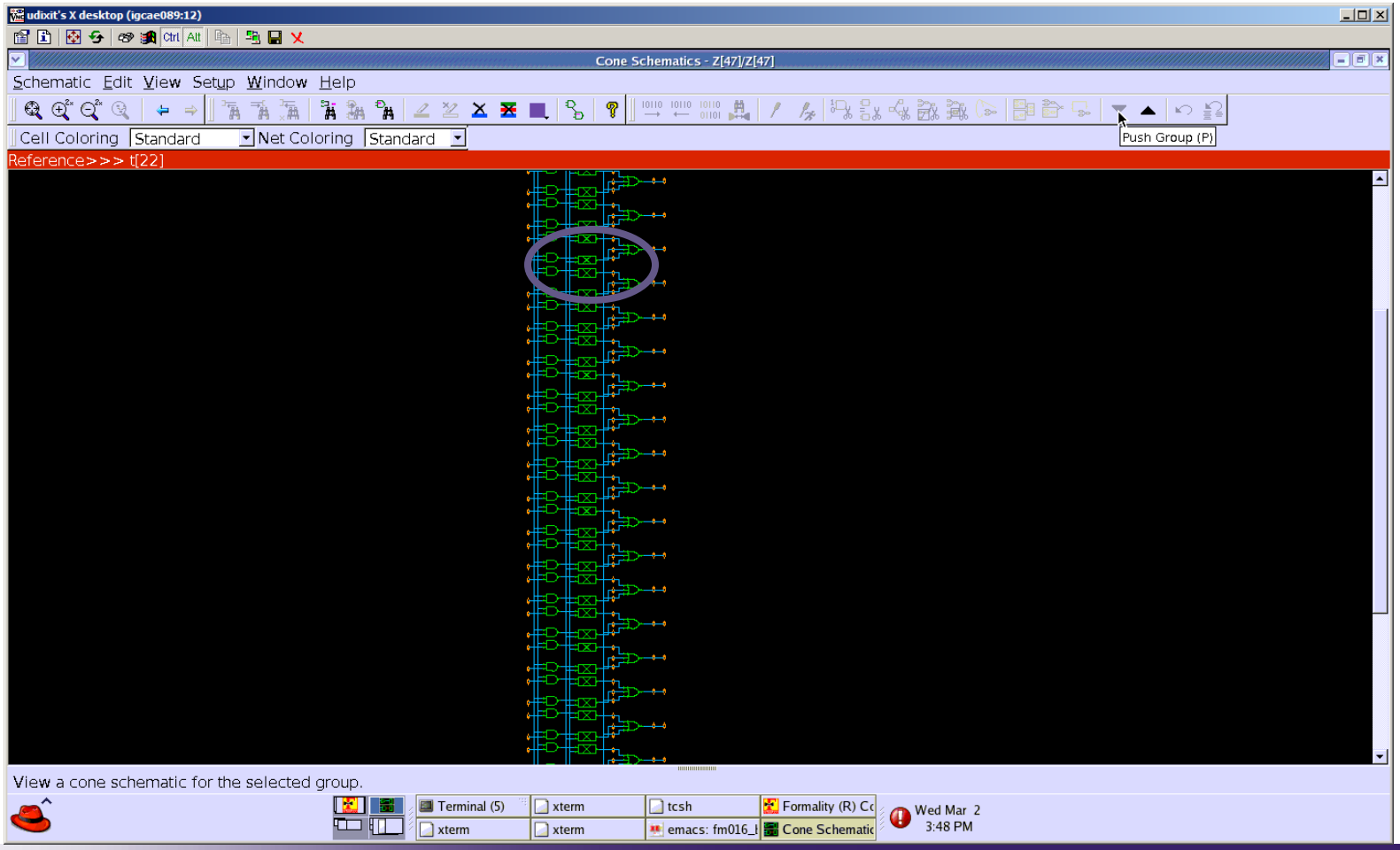
SYNOPSYS®
Predictable Success

Find Cause of Hard Points

Schematics

Hierarchy Grouping

Finding don't care cells in the fanin of the hard point



SYNOPSYS® Predictable Success

Debugging Inconclusive Verifications

Three Step Approach

Agenda

- Inconclusive Debug: 3-Step Plan
  - Find **List** of Hard Points (We want list to be as small as possible)
  - Find **Cause** of the Hard Points
  - Find **Resolution** of Hard Points

SYNOPSYS® Predictable Success

Debugging Inconclusive Verifications

Three Step Approach

Agenda

- Inconclusive Debug: 3-Step Plan
  - Find **List** of Hard Points (We want list to be as small as possible)
  - Find **Cause** of the Hard Points
  - Find **Resolution** of Hard Points

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

- set_verification_priority(SVP)
- Verification of large XOR chains
- FM techniques
  - Manual Factoring
  - Cut points
  - Hierarchical verification
- DC Techniques
  - Disabling Optimizations
  - 2 pass flow
- RTL Coding Tips

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

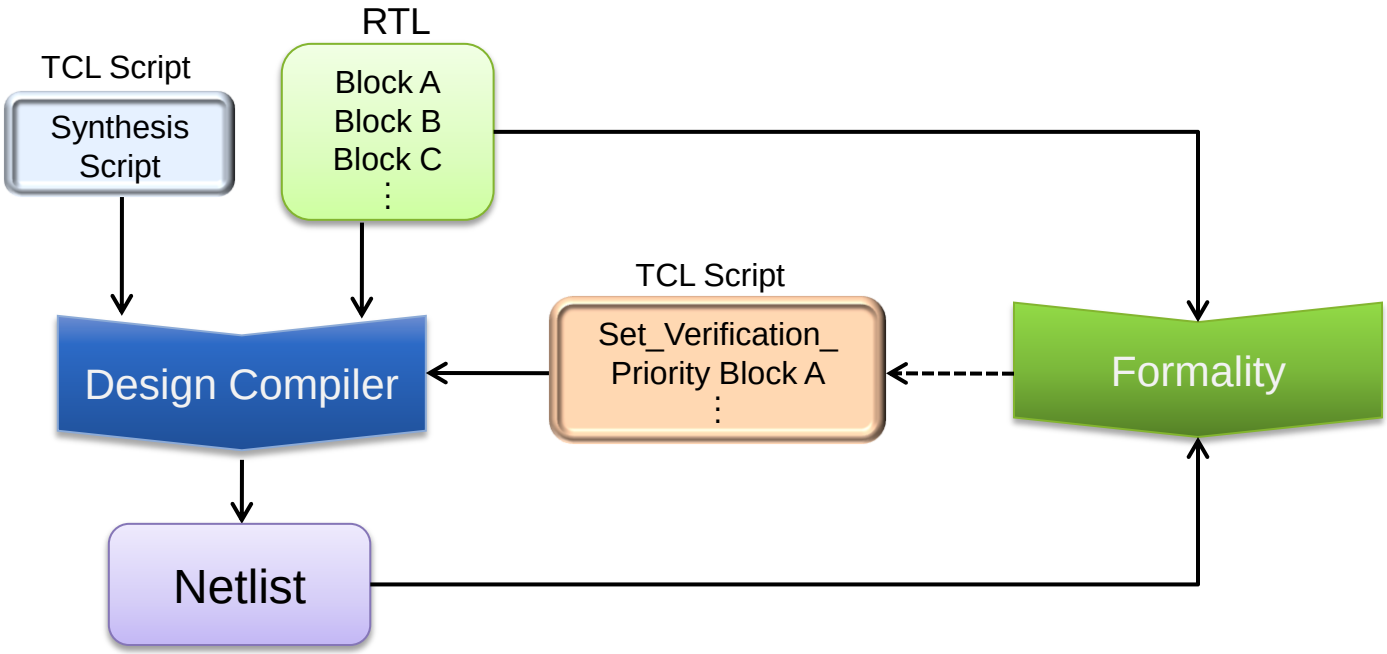
set_verification_priority (SVP)

- Solution that is unique to Formality and DC Ultra
- Formality `analyze_points` command produces Design Compiler `set_verification_priority` commands that can be inserted into the DC Ultra script
  - targets specific block(s), instances, or arithmetic operators
  - turns off specific optimizations
  - improves verification success
  - minimizes QoR impact

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

set_verification_priority(SVP)



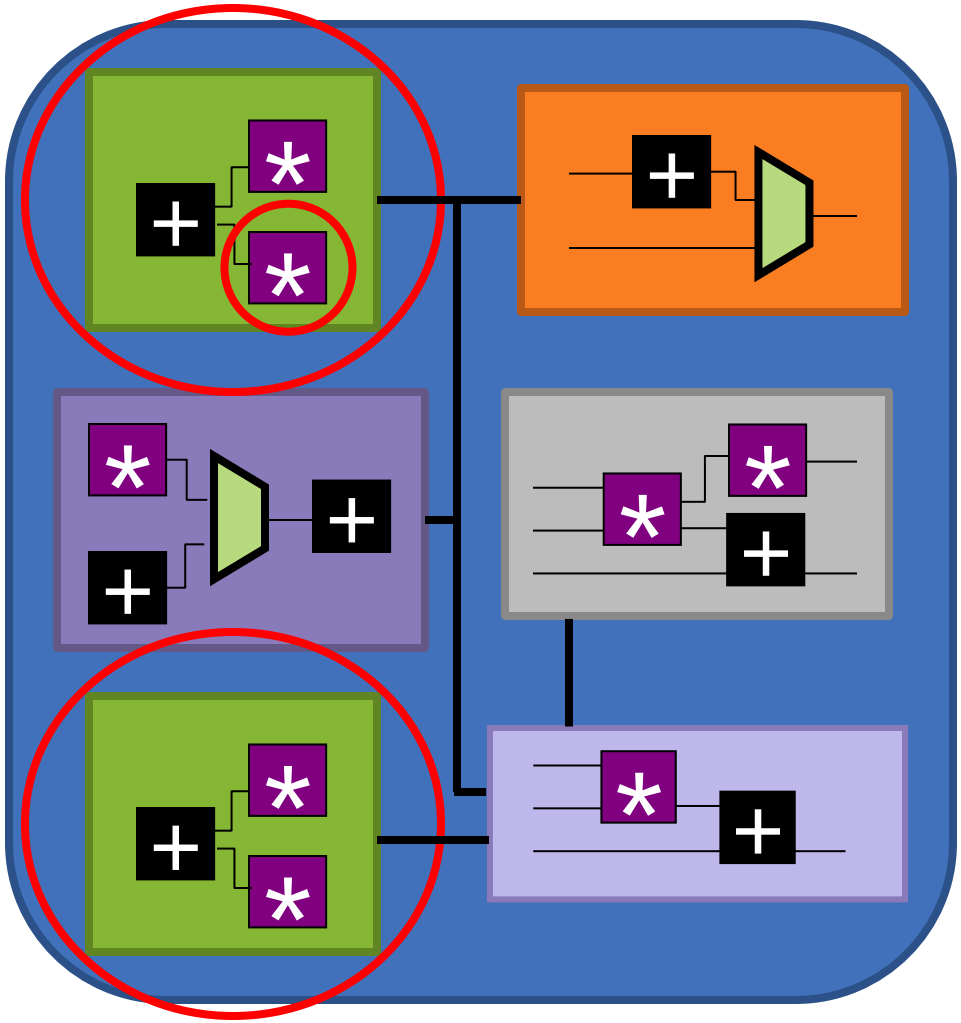
- If Formality run is inconclusive, use analyze_points command to generate `set_verification_priority` commands for DC
- Rerun synthesis on affected block(s)

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

set_verification_priority(SVP)
Reducing QoR impact

- SVP previously targeted the block level
  - DC optimizations altered on all instances
- Since 2010.12-SP1, SVP surgically targets the operator level within the specific instance
  - QoR impact of SVP is negligible for most designs
- Customer feedback has been very positive and many are incorporating the use of SVP into their standard flows.



SYNOPSYS® Predictable Success

Find Resolution of Hard Points

set_verification_priority (SVP)

- Already shown in earlier slide…but worthy to mention again…
- Shell Example report for SVP recommendation through analyze_points command

```
********************************** Analysis Results **********************************
Found 1 Hard Datapath Component Module
--------------------------------
These modules contain arithmetic operators that may be
contributing to hard verifications.
Lowering the Design Compiler optimization level for the these
modules may permit verification to succeed.
--------------------------------
r:/WORK/top in file /remote/fmcae4/users/udixit/rtl/test.v
   Module with datapath cell(s):
      r:/WORK/top/DP_OP_23J1_125_5602

    Try adding the following command(s) to your Design Compiler script right before the first compile_ultra command:
      current_design top
      set_verification_priority [ get_cells { add_28 mult_28 sub_28 } ]

      current_design top

-----------
--------------------------------
*****************************************************************************************
Analysis Completed
```


SYNOPSYS® Predictable Success

Find Resolution of Hard Points

set verification priority (SVP)

- Design Compiler script
  - Before the use of `set_verification_priority` command

```
current_design top
compile_ultra -scan
```

  - After the use of set_verification_priority command

```
current_design top
set_verification_priority [ get_cells {
  add_28 mult_28 sub_28 } ]
compile_ultra -scan
```

- It should always be added before first `compile_ultra` command

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

- set_verification_priority(SVP)
- Verification of large XOR chains
- FM techniques
  - Manual Factoring
  - Cut points
  - Hierarchical verification
- DC Techniques
  - Disabling Optimizations
  - 2 pass flow
- RTL Coding Tips

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

Verification of large XOR chains

- Formality better handles verification of designs that contain large XOR chains, such as CRCs, ECCs, and parity trees
- Design Compiler, through the use of the command `set_verification_priority` (to be set on designs with CRC logic), will preserve the XOR chains into a hierarchy, and issue new SVF guidance (`guide_group_function`) as necessary

```
set_verification_priority [ get_designs { *crc* } ]
```

- Formality uses this guidance to create the appropriate hierarchy which assists verification
- As an alternative to `set_verification_priority` command, you can set following variable to true in DC script before first compile command

```
set compile_isolate_crc_logic true
```


SYNOPSYS® Predictable Success

Find Resolution of Hard Points

- set_verification_priority(SVP)
- Verification of large XOR chains
- FM techniques
  - Manual Factoring
  - Cut points
  - Hierarchical verification
- DC Techniques
  - Disabling Optimizations
  - 2 pass flow
- RTL Coding Tips

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

- set_verification_priority(SVP)
- Verification of large XOR chains
- FM techniques
  - Manual Factoring
  - Cut points
  - Hierarchical verification
- DC Techniques
  - Disabling Optimizations
  - 2 pass flow
- RTL Coding Tips

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Manual Factoring

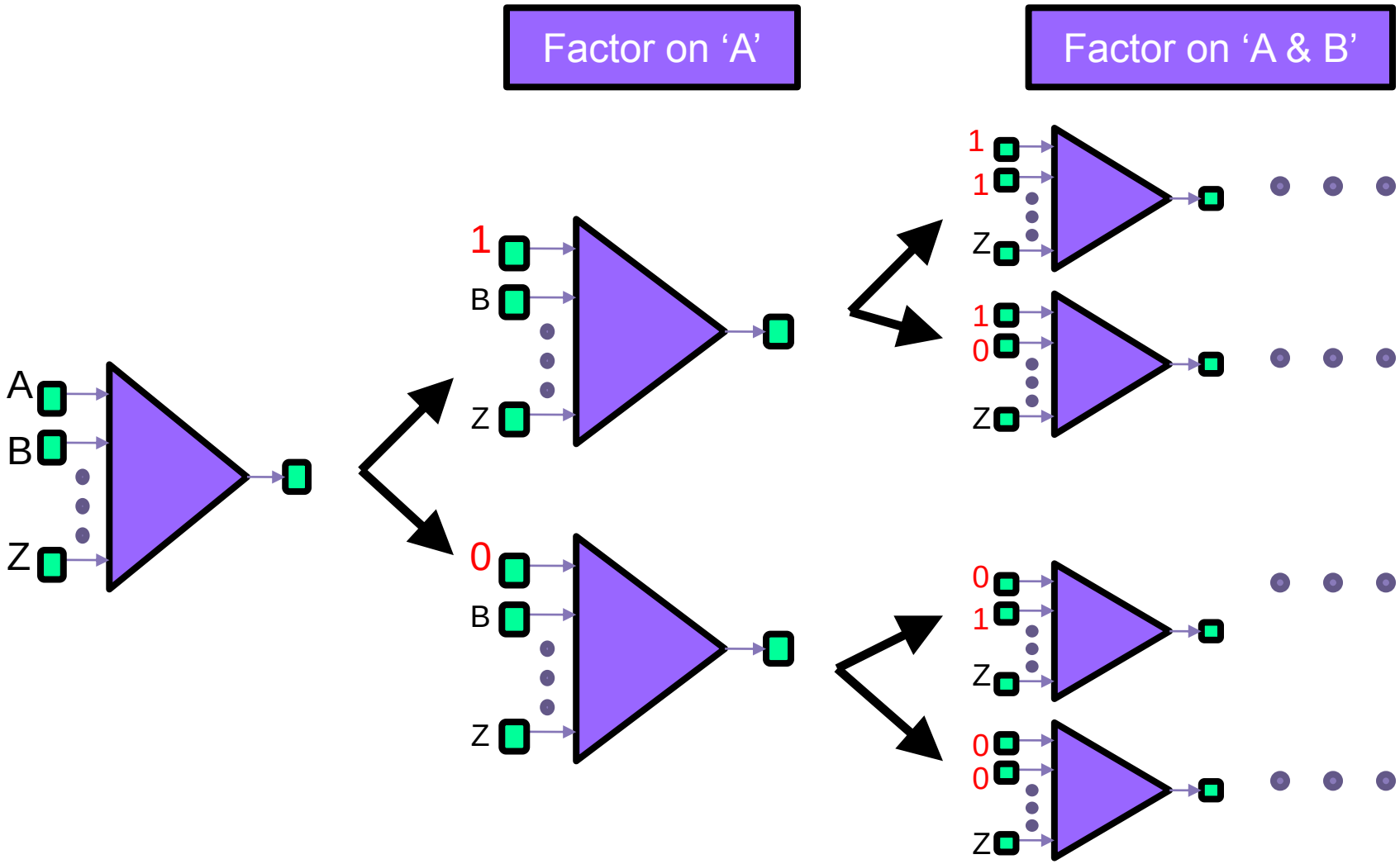
- What is a Factored verification ??
  - Breaking up a problem into sub problems
  - Same Solvers used on sub-problems
  - Success comes when sub-problems are significantly easier than the original problem
- Formality deploys automatic factoring under the hood during verification
  - Holding an input of a logic cone to constant “0”, verifying, then holding the input to constant “1”, verifying
- Works with large cones having selectors, don’t care space
  - Not useful with CRC, parity generators, XOR trees
- Can use manual factoring in unlikely event automatic factoring does not help

```
fm_shell> set_factor_point -type port $ref/data/sel2_*
  Set factoring variable at 'r:/WORK/top/data/sel2_1'
  Set factoring variable at 'r:/WORK/top/data/sel2_2'
  Set factoring variable at 'r:/WORK/top/data/sel2_3'
```


SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Manual Factoring



SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Manual Factoring

- Commands to set, report and remove factor points are
  - `set_factor_point`
  - `report_factor_point`
  - `remove_factor_point`
- `set_factor_point -type <object-type> <list of objectIDs>`
  - Sets the user defined factor points
  - objectIDs
    - Primary input ports
    - Registers
    - Bbox output pins
    - Any previously set hard cutpoint
    - Can use wildcards
  - -type <object type>
    - Needed to distinguish between objects of same type but different name

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Manual Factoring

`report_factor_point`

- Returns any user defined factor points
- Can be run in setup, match, or verify phases

`remove_factor_point -type <object-type> <list of objectIDs> | -all`

- Object ids and –type <object type> works same way as set_factor_point
- -all removes all factor points

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Manual Factoring

Finding Factor Candidates

- Investigate RTL of Cone
  - Look for partially specified Case statements, Enums, etc. where don't cares can be introduced
  - Look for Cases in general
- Look at SVF for significant datapath in Cone
  - Look in Schematics for Selectors to any significant Datapath in Cones

    Find DP blocks from cone

    Look for selectors if any

    Trace back to PI of cone

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Manual Factoring

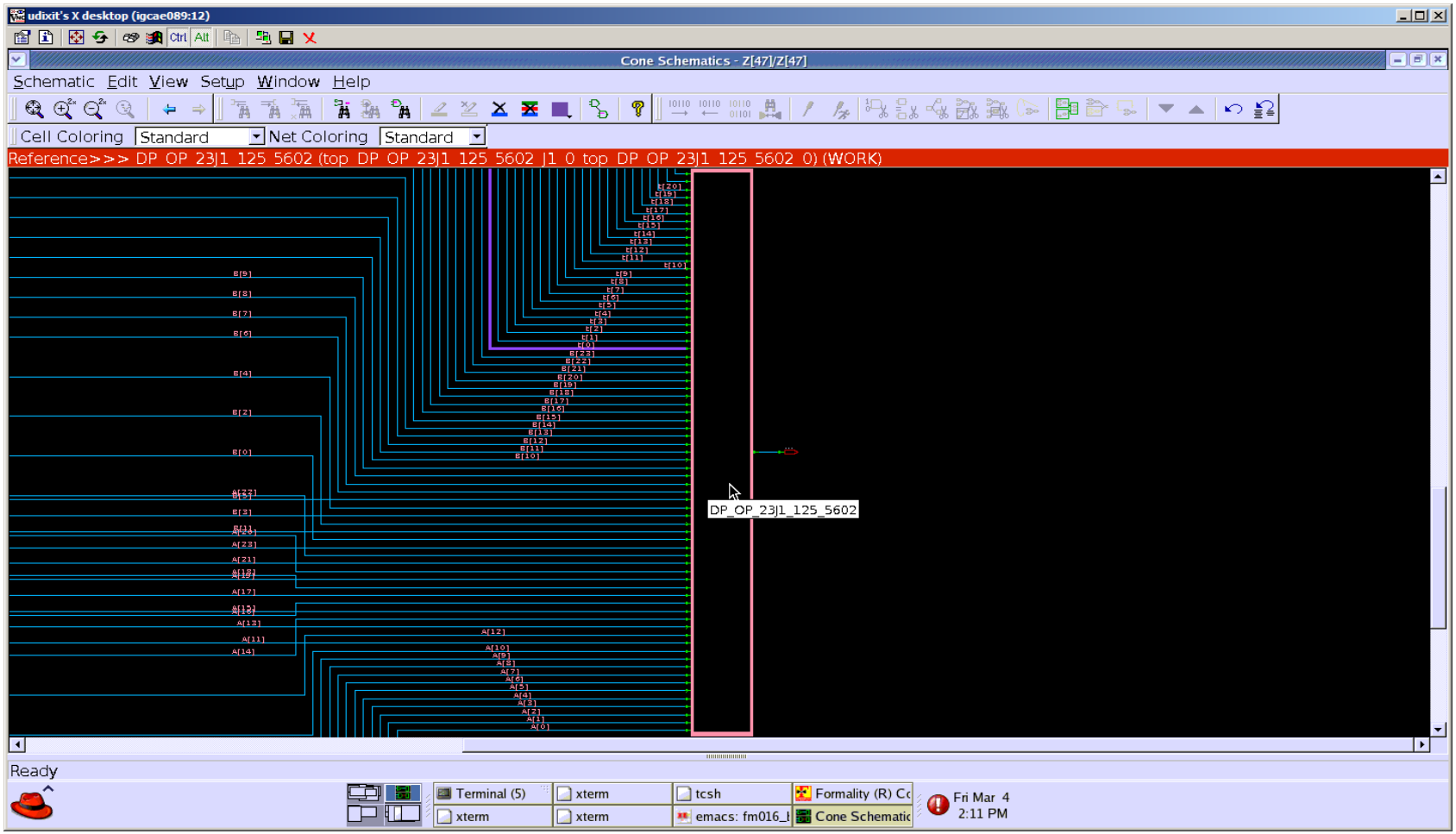
- Examples…
  - Next few slides shows couple of examples to find manual factoring points with the help of schematics
  - One example has don’t care space in the input
  - Other one has big selector

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Manual Factoring

Example 1: don't_care space

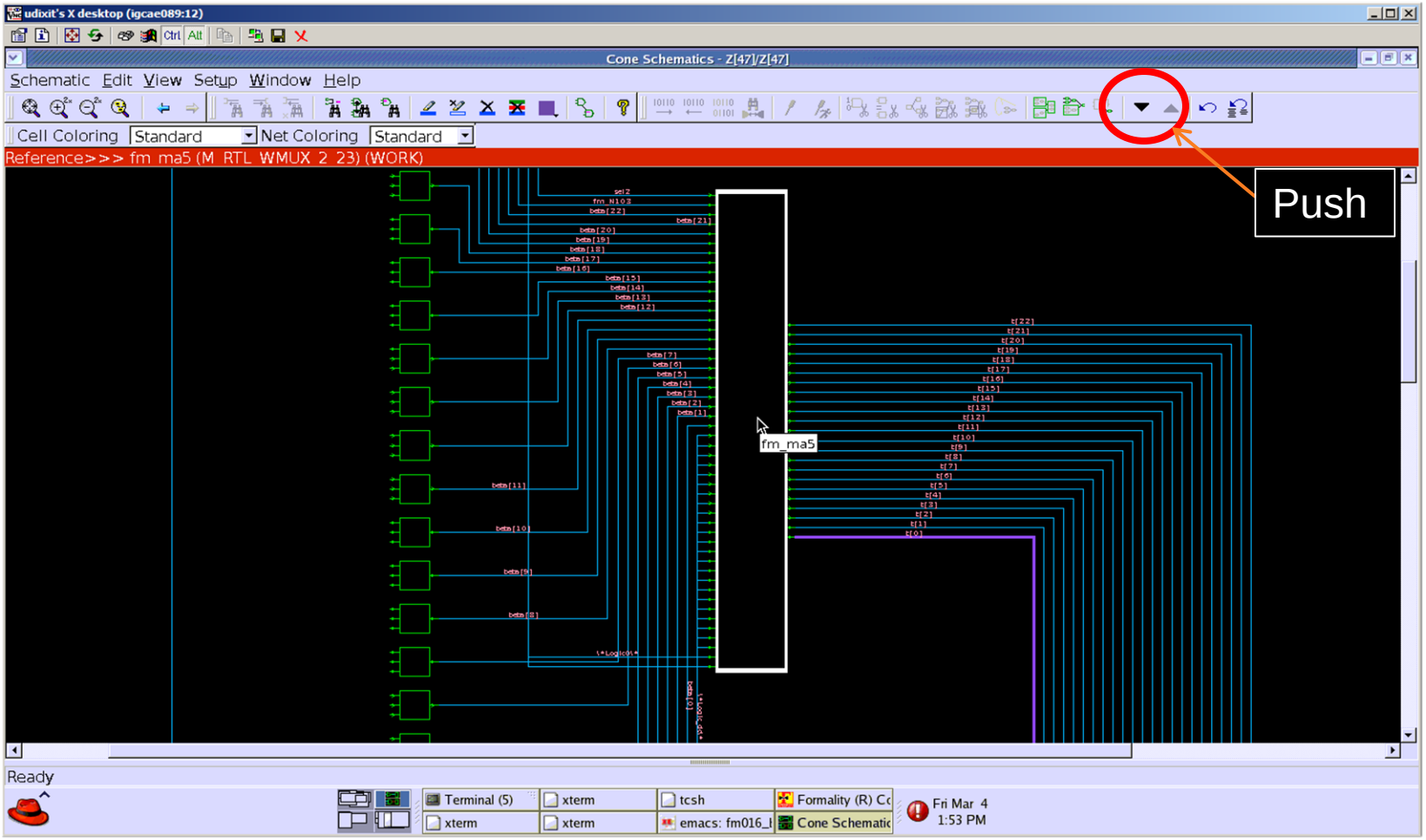


SYNOPSYS®
Predictable Success

Find Resolution of Hard Points

FM Techniques: Manual Factoring

Example 1: don't_care space

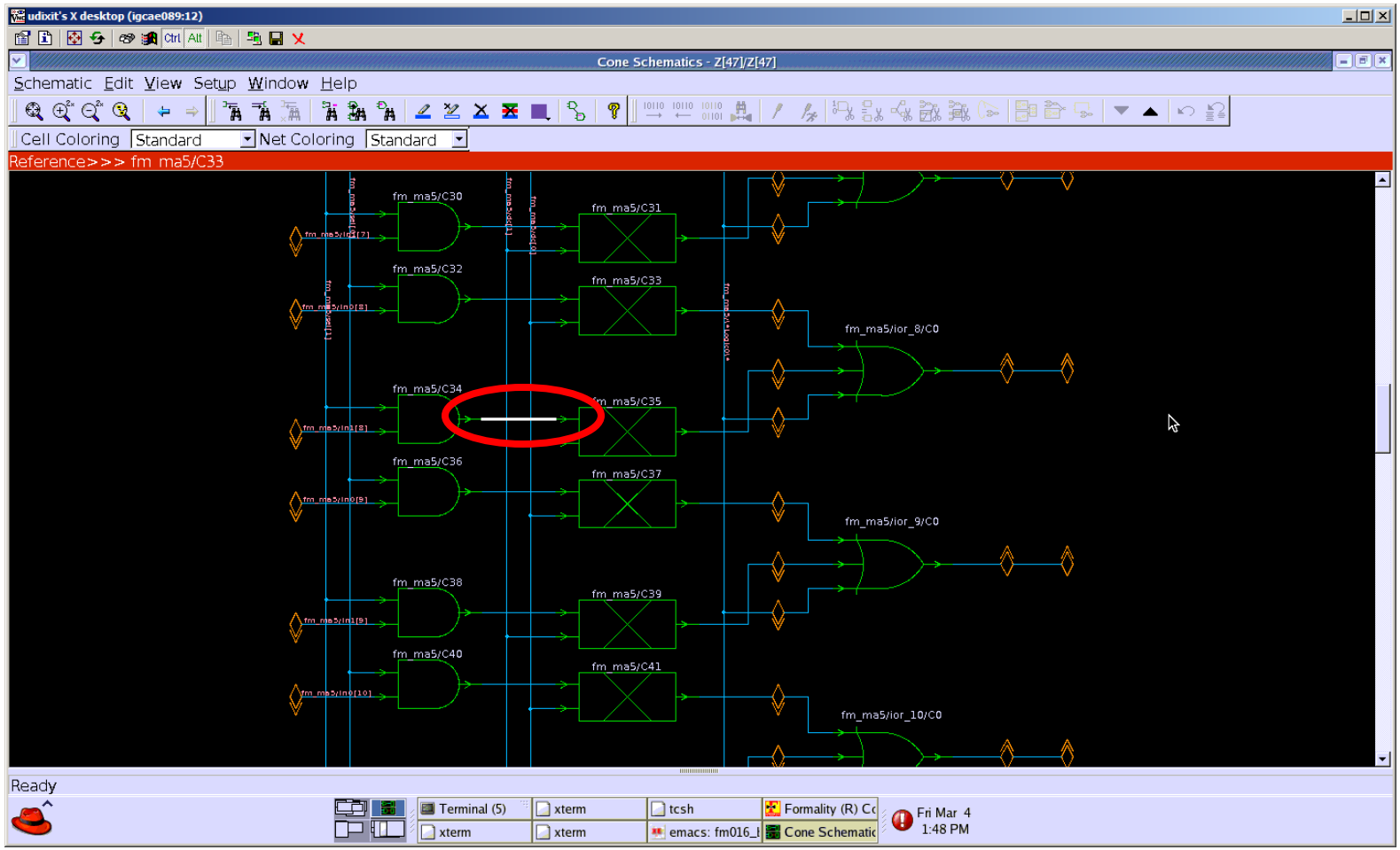


SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Manual Factoring

Example 1: don't_care space



SYNOPSYS®
Predictable Success

Find Resolution of Hard Points

FM Techniques: Manual Factoring

Example 1: don't_care space

- Solution is manual factoring on following points

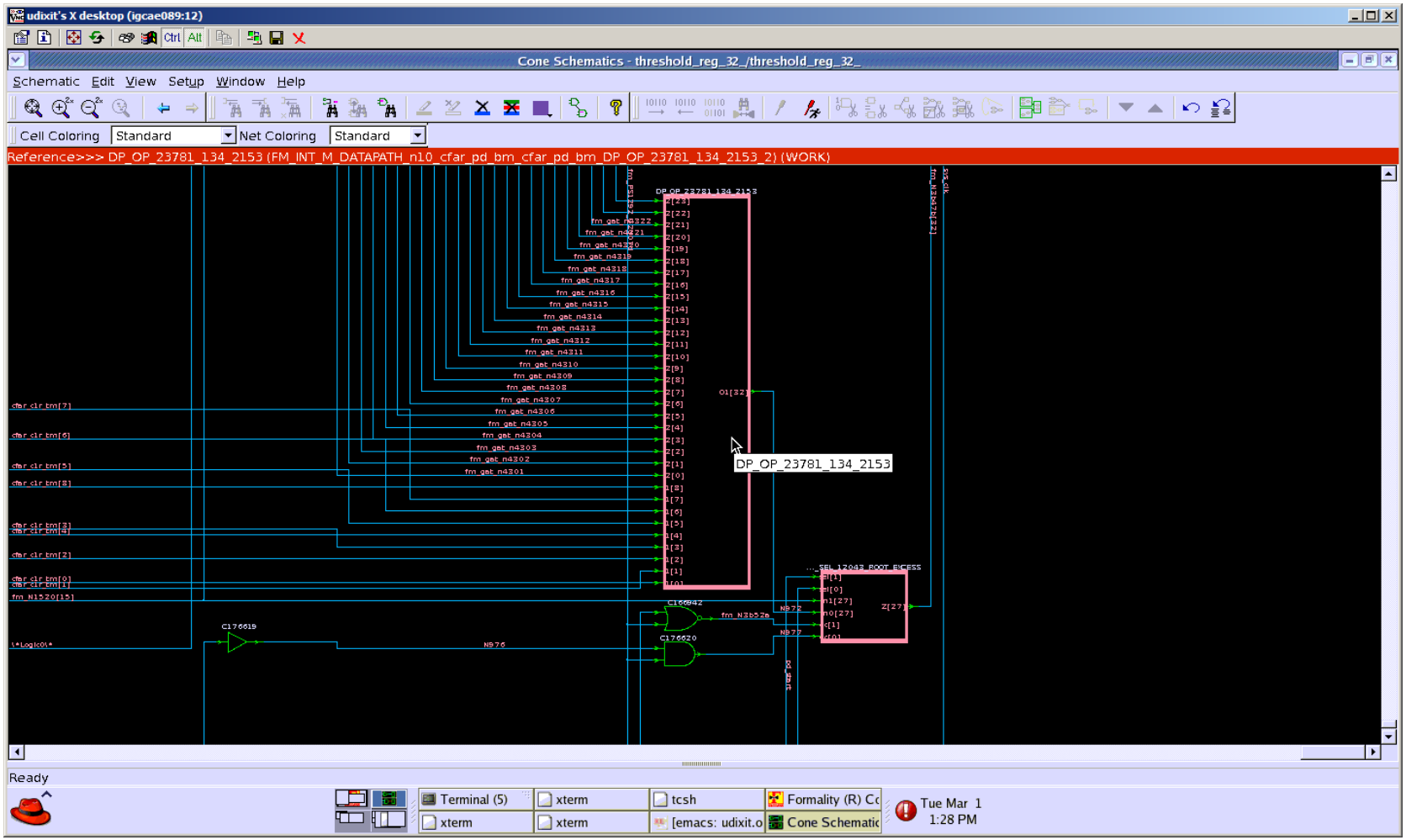
```
set_factor_point $ref/u1/u2/u3/u4/factor_reg[*]
```


SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Manual Factoring

Example 2: selector logic



SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Manual Factoring

Example 2: selector logic

```
if (condition) begin
        …
        …
        if (in1 <= 27)
                out  <= in2* in3[~reg1][7+in1];
        else
                out  <= in2* in3[~reg1][7];
end
```


SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Manual Factoring
Example 2: selector logic

- Solution is manual factoring on following points

```
set_factor_point r:/WORK/top/in1[*]
set_factor_point r:/WORK/top/condition
```


SYNOPSYS® Predictable Success

Find Resolution of Hard Points

- set_verification_priority(SVP)
- Verification of large XOR chains
- FM techniques
  - Manual Factoring
  - Cut points
  - Hierarchical verification
- DC Techniques
  - Disabling Optimizations
  - 2 pass flow
- RTL Coding Tips

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Cutpoints

- Cutpoints in general are used in Formality to divide cones of logic into smaller chunks
  - Inherent problem is false negatives
- Soft Cutpoints
  - Automatically created by Formality
  - Automatically removed them if they fail
    - Cone is re-verified
- Hard Cutpoints
  - User defined with `set_cutpoint` command
  - Never dropped
  - Problem is if failures, is it real or did the cutpoint introduce a false failure ?

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Cutpoints

```
set_cutpoint -type <ObjectType> ObjectID
```

- Can be used wherever you are sure a specific boundary has been preserved
  - Usually true in SVF flow for most DW/DP_OP blocks
- Must be set on matched points (use set_user_match)
- Related Commands
  - `report_cutpoint`
  - `remove_cutpoint -type <ObjectType> ObjectId`
  - `remove_cutpoint -all`

- Idea is for certain hard blocks we can attempt to break the cone up into smaller pieces by setting hard cutpoints at the outputs of DP_OP/DW blocks within the cone

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Cutpoints

Potential strategy:

- Identify DP_OP/DW blocks are in cones of Hard Points
  - Use find/report_svf_operation/analyze_points commands
- Generate matched report of block_pins
  - Use `report_match -point_type block_pin`
  - Insert cutpoints on outputs of DP_OP or DesignWare blocks within hard verification logic cone
  - Must be matched in both reference and implementation cones
- Example:

```
set_cutpoint -type net $ref/PC[1]
set_cutpoint -type net $impl/PC[1]
set_user_match -type net $ref/PC[1] $impl/PC[1]
```

- Re-run verification with hard cutpoints and user_matches set

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Cutpoints

Some closing points:

- This solution is Dynamic !
  - If synthesis/RTL changes, scripts will likely need to be regenerated
- Must account for inverted matches
- If you get failures, it could be due to the hard cutpoints themselves
- Why not do this automatically?
  - DC propagates constraints across hierarchies.
- Resolve any known datapath issues found in cones first

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

- set_verification_priority(SVP)
- Verification of large XOR chains
- FM techniques
  - Manual Factoring
  - Cut points
  - Hierarchical verification
- DC Techniques
  - Disabling Optimizations
  - 2 pass flow
- RTL Coding Tips

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

FM Techniques: Hierarchical Verification

- Using hierarchical script technique
  - we already spoke about how it can help with Hard verifications.
- Not always effective technique as it used to be because of ungrouping and other optimization done by Design Compiler

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

- set_verification_priority(SVP)
- Verification of large XOR chains
- FM techniques
  - Manual Factoring
  - Cut points
  - Hierarchical verification
- DC Techniques
  - Disabling Optimizations
  - 2 pass flow
- RTL Coding Tips

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

- set_verification_priority(SVP)
- Verification of large XOR chains
- FM techniques
  - Manual Factoring
  - Cut points
  - Hierarchical verification
- DC Techniques
  - Disabling Optimizations
  - 2 pass flow
- RTL Coding Tips

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

DC Techniques: Disabling Optimizations

- `set_verification_priority`
  - Dials back some datapath optimizations
  - Increases verifiability at cost of some QOR
  - This should always be the first technique to consider
- `"compile_isolate_crc_logic"` to true in DC script.
  - Increases verifiability by keeping the hierarchy intact of XOR designs
  - SVP can be used to target this as a first preference, use this option only if you suspect widespread use of crc type logic.

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

DC Techniques: Disabling Optimizations

- DC Ultra ungroups designs so logic can move across boundaries
  - Enables more optimization opportunities
  - Increases QoR
  - To disable: `compile_ultra -no_boundary_opts`
  - Maintains user hierarchy and improves cutpoints based verification in Formality
- Hinders Formality's recovering the hierarchical boundaries of datapath elements
- Performs ungrouping of all DesignWare hierarchy
  - To disable: `set compile_ultra_ungroup_dw false`
- Automatically performs area-based and delay-based ungrouping of user hierarchy
  - To disable: `compile_ultra -no_autoungroup`

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

DC Techniques: Disabling Optimizations

`hdlin_verification_priority`

- This off by default Presto variable needs to be set to true before design read

```
set hdlin_verification_priority true
```

- When set to true,
  - Disables some optimizations in Presto and only affects optimizations done during reading and elaboration of RTL files
- It doesn't have block/operator level control and can have QOR impact but improves verifiability
- Variable `hdlin_verification_priority` controls optimization done during design read where as `set_verification_priority` controls optimization done during compile

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

- set_verification_priority(SVP)
- Verification of large XOR chains
- FM techniques
  - Manual Factoring
  - Cut points
  - Hierarchical verification
- DC Techniques
  - Disabling Optimizations
  - 2 pass flow
- RTL Coding Tips

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

DC Techniques: Two Pass Flow

- Suggestion here is to write out intermediate netlist/svf after each incremental compile in Design Compiler
- For example
  - In case of retiming, write out pre retimed netlist/svf and then perform retiming
  - In case of ungrouping/flattening, write out hierarchical netlist/svf before ungrouping
- Do incremental compare in Formality
  - This flow works
  - Customers have seen successful verification with this flow for a cases which were inconclusive otherwise
- This is a workaround, our goal is to verify out of the box DC netlist

SYNOPSYS® Predictable Success

Find Resolution of Hard Points

- set_verification_priority(SVP)
- Verification of large XOR chains
- FM techniques
  - Manual Factoring
  - Cut points
  - Hierarchical verification
- DC Techniques
  - Disabling Optimizations
  - 2 pass flow
- RTL Coding Tips

SYNOPSYS® Predictable Success

RTL Coding Tips

- In general avoid using explicit Don't Care assignments
- Use fully specified CASE statements
  - Assign known values to all branches of CASE to reduce implicit Don't Care conditions
- Often seen cases when customer does RTL ECO, Formality gives inconclusive verification which was conclusive before ECO was done
  - Main reason is SVF rejections
  - SVF flow is very sensitive to RTL line numbers
  - With RTL ECO, line number changes and doesn't match with SVF
  - Formality ends up rejecting SVF guidances and hence inconclusive verification
- Recommendation is to use new Formality ECO flow available in F-2011.09 release

SYNOPSYS® Predictable Success

Debugging Inconclusive Verifications

Summary

- First things:
  - Study the log files for obvious errors
  - Try `analyze_points` for initial help
  - Use latest releases of DC and Formality !
- Find minimized list of hard to verify compare points

  `report_unverified_points and report_aborted_points`
- Find cause of hard points

  `report_svf_operation and analyze_points`
- Find Resolution of the hard points
  - Formality techniques
    - Re-verification with timeouts
    - Hierarchical verification
    - Manual factoring
    - Hard Cutpoints
  - Design Compiler techniques include using the
    - `set_verification_priority,` always try this first
    - DC variables to disable ungrouping, boundary optimization and isolate CRC logic
    - Presto command `hdlin_verification_priority`, try when presto optimization is the cause
  - RTL Coding tips

SYNOPSYS® Predictable Success

What to Provide BU for HV Stars

- Help us to Help you !
- The Usual…..
  - Pre-Match Containers, SVF, UPF
  - Formality log file
  - List of Hard Points
  - Either Report or Session File
  - Allows us to target problem areas immediately
- Bonus:
  - RTL, Libraries and Synthesis scripts
  - Allows us to try DC techniques

SYNOPSYS® Predictable Success

SYNOPSYS®

Predictable Success