Formality Jumpstart Training

Copyright

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 may be used only as permitted under such an agreement.

LEGAL NOTICE

Information contained in this presentation reflects Synopsys plans as of the date of this presentation. 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.

Formality Mission Statement

If Design Compiler reads it, optimizes it, or writes it, Formality must verify it.

- Primary Goal – Highest design QoR in a verifiable flow
  - Verification of all Design Compiler Ultra default optimizations

    No need to turn off optimizations to pass equivalence checking
  - Lockstep language support (SystemVerilog, VHDL, Verilog)
- Secondary Goal – Time to results
  - Auto-setup environment for Design Compiler Ultra

    Less manual setup
  - Continuous improvement in performance and capacity

Agenda

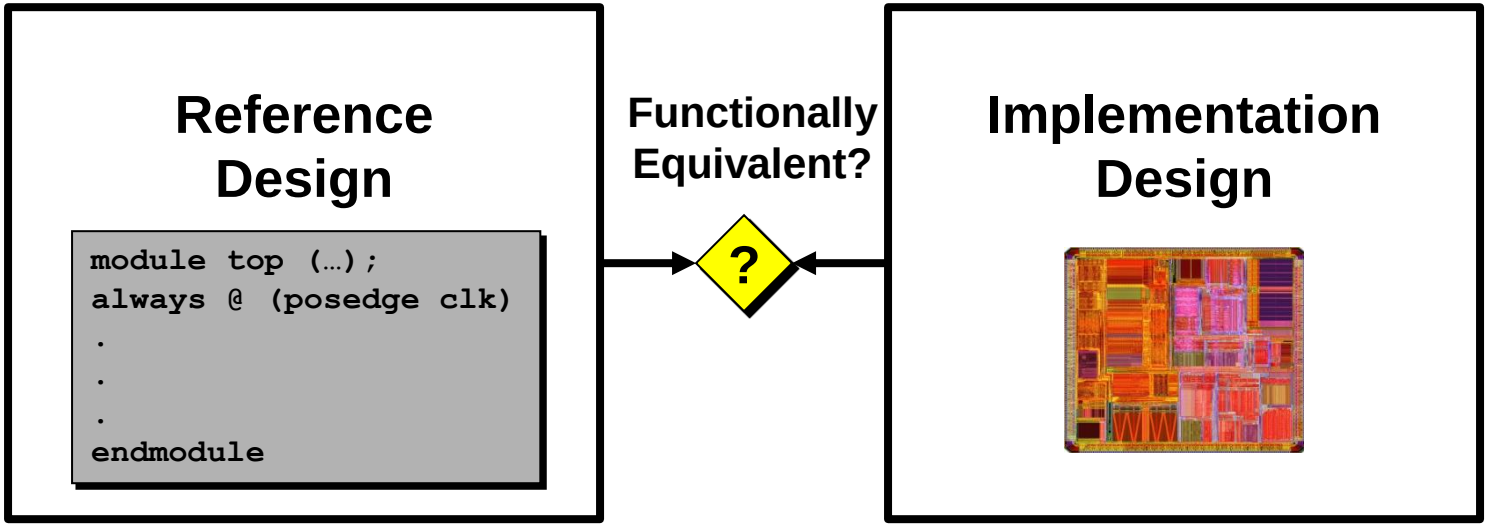
- Introduction to Equivalence Checking
- Using Formality
  - Flow Overview
  - Guidance
  - Read
  - Setup
  - Match
  - Verify
  - Debug
- Documentation and Help
- Lab Exercises

Glossary

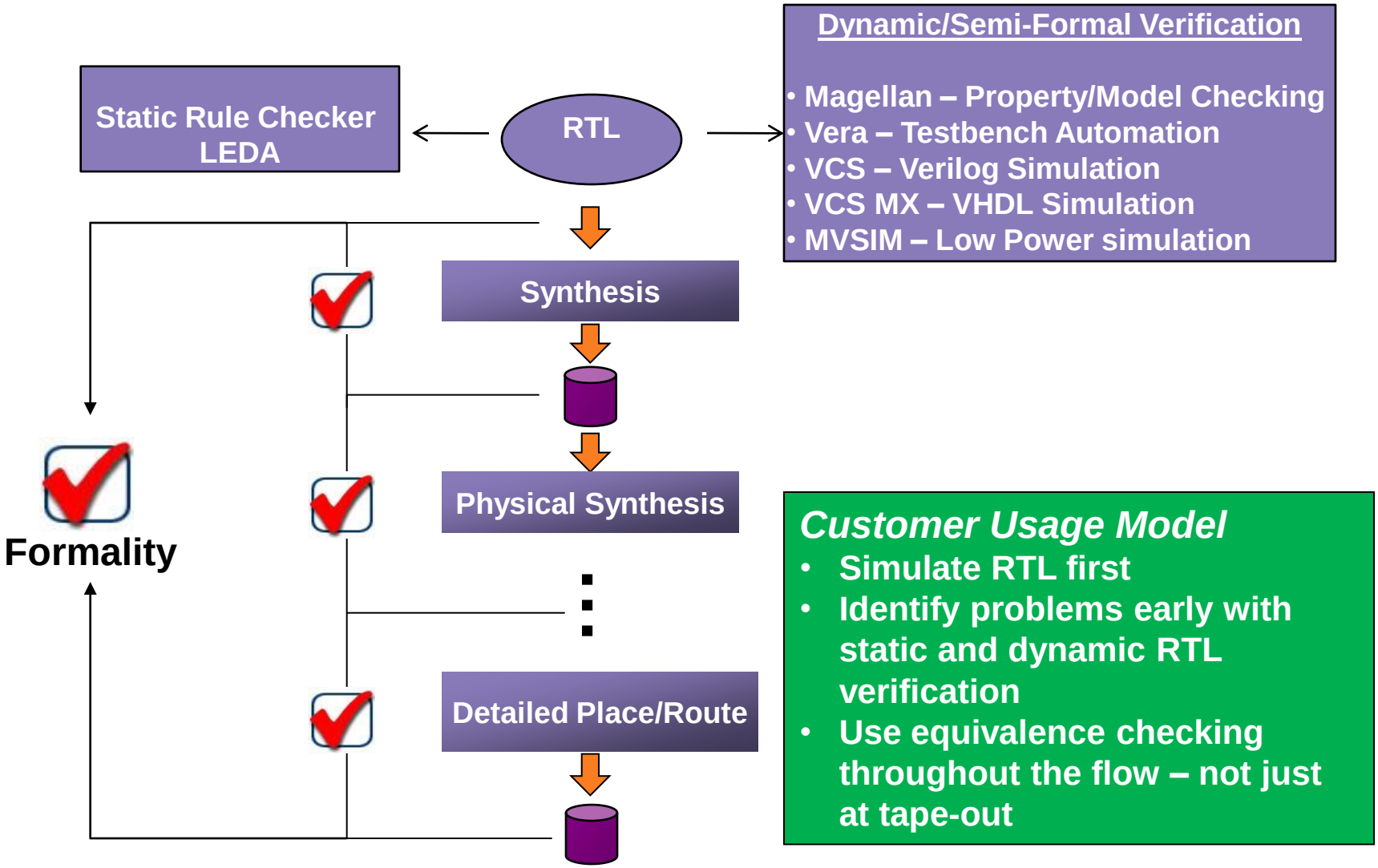
- Reference Design
  - The *golden* design against which a design is tested
  - Usually the RTL (Verilog, SystemVerilog, VHDL)
  - Simulated and known to be good
- Implementation Design
  - The modified design that is checked against the reference design
- Containers
  - A Formality database consisting of designs and libraries
  - The default reference container is named *r*
  - The default implementation container is named *i*
  - Can be saved and read using any version of Formality

Equivalence Checking

- Assumes that the reference design is functionally correct
- Determines if the implementation design is functionally equivalent to the reference
  - Provides counter-examples if designs are functionally different
- Is mathematically exhaustive with no missing corner cases
- Does not require test vectors



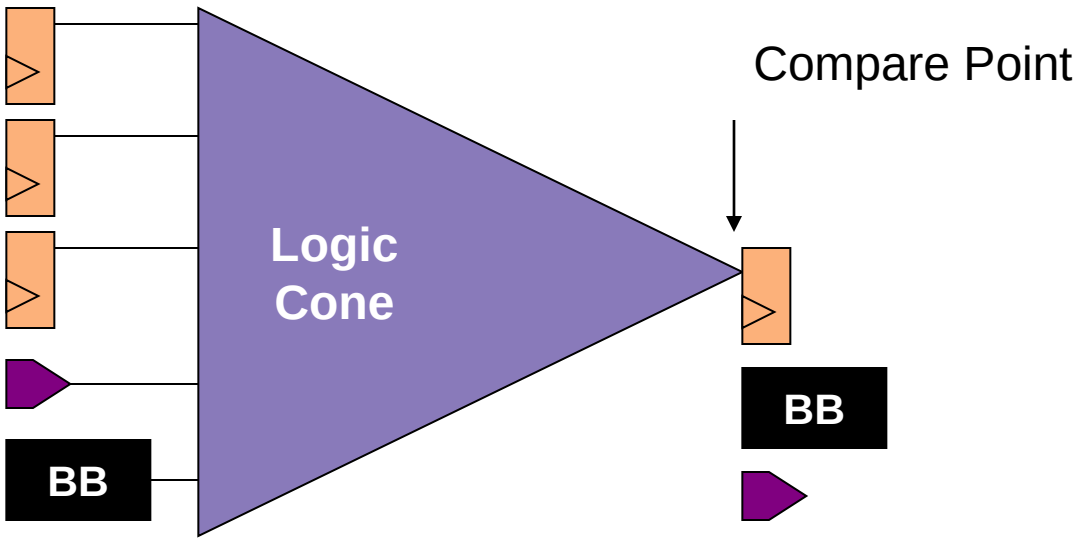
Equivalence Checking in Your Flow



Key Equivalence Checking Concepts

- *Logic Cones and Compare Points*
  - Common Compare Points
    - Primary output
    - Register or latch
    - Input of a black box
  - Less Common Compare Points
    - Multiply-driven net
    - Loop
    - Cutpoint
  - Logic Cone
    - A block of combinational logic that drives a compare point

Logic Cone



Inputs to a logic cone

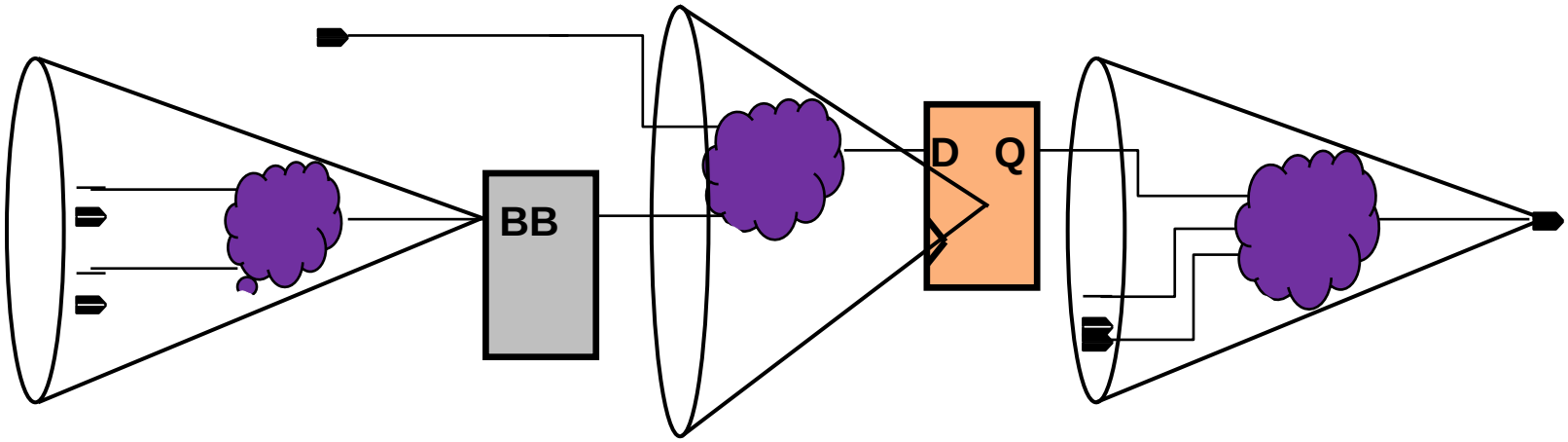
- Register Output Pins
- Primary Input Ports
- Black Box Output Pins

Compare Points

- Registers
- Primary Output Ports
- Black Box Input Pins

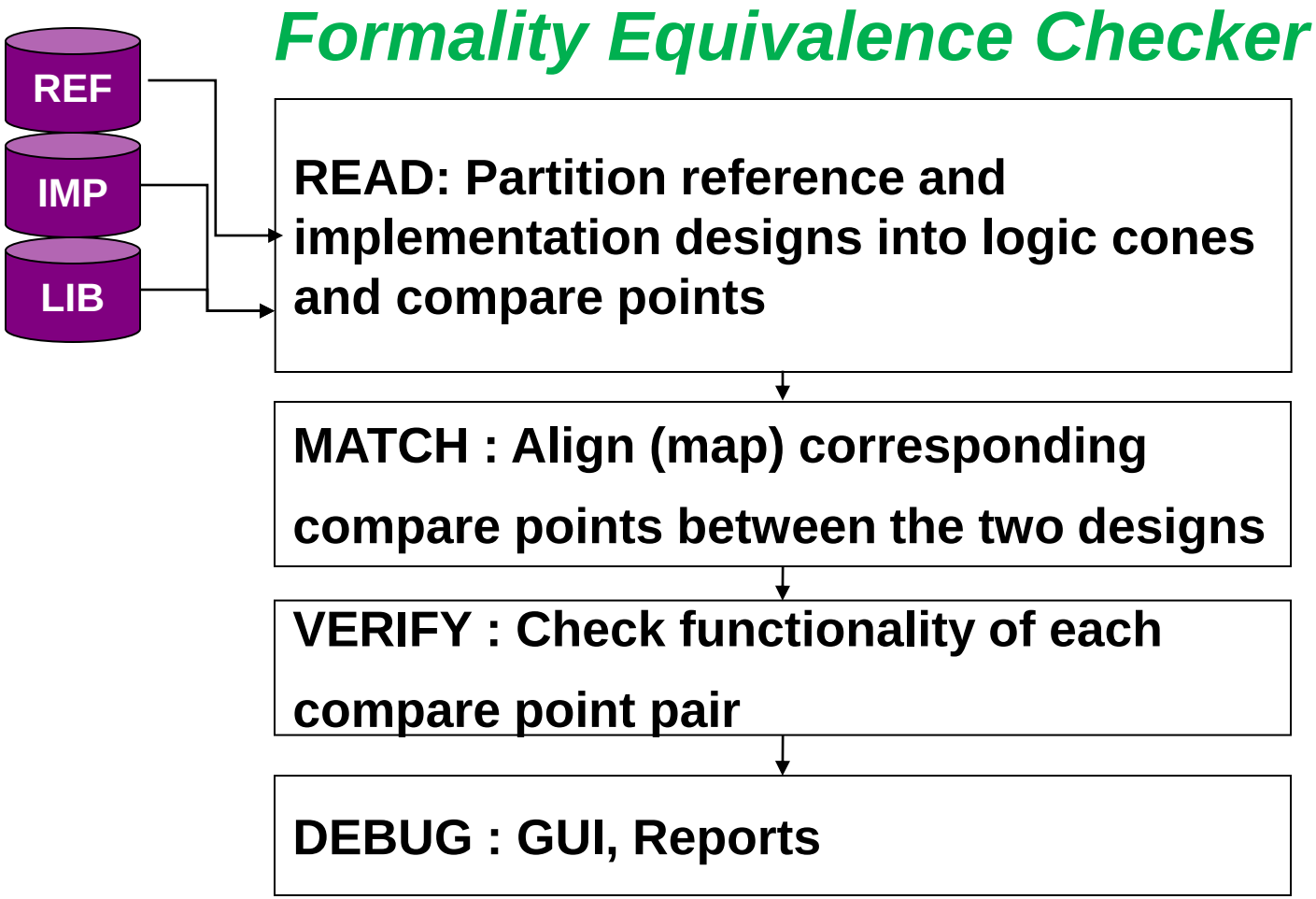
Logic Cones and Compare Points

- Formality breaks the reference and implementation designs up into compare points, each with its own logic cone



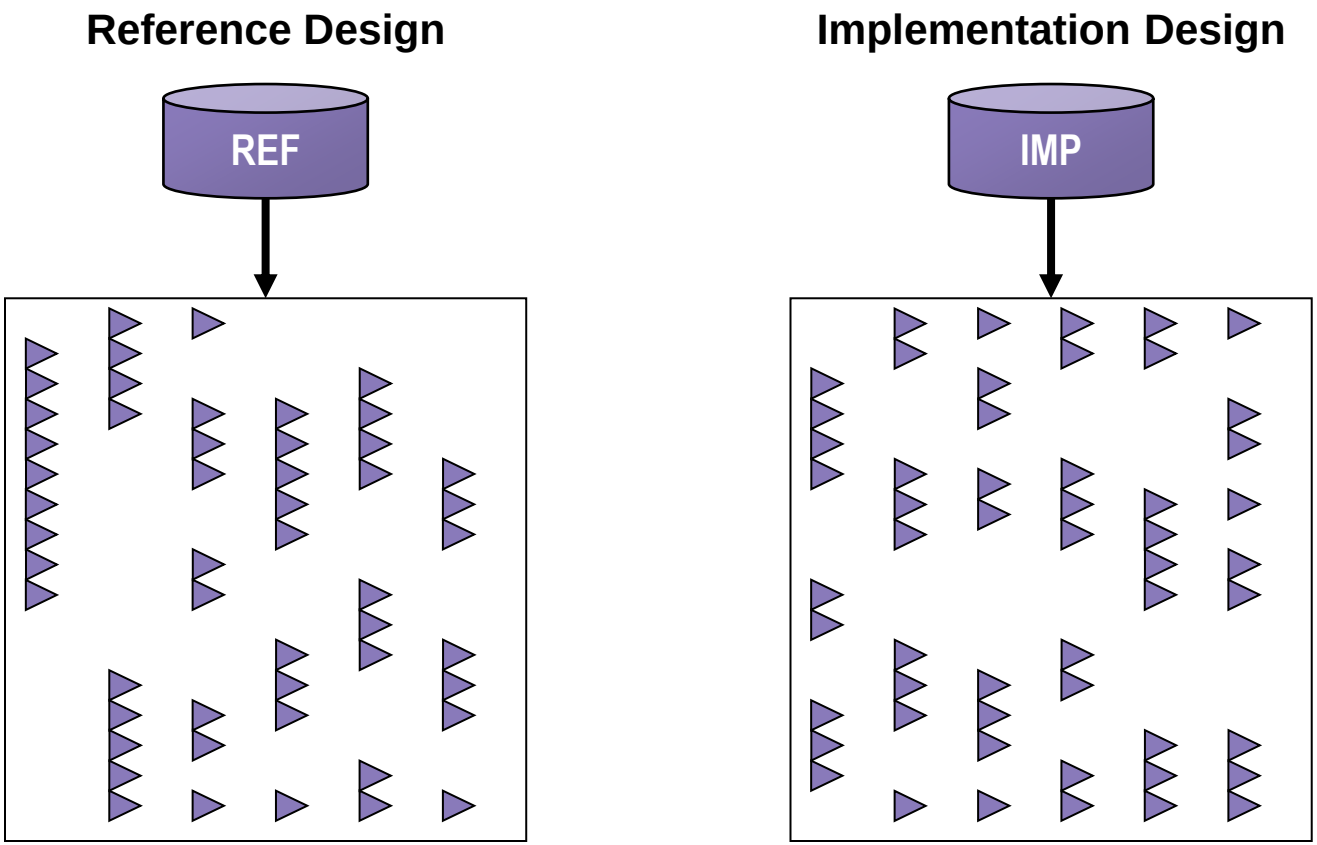
Determining Compare Points

Formality Flow Overview



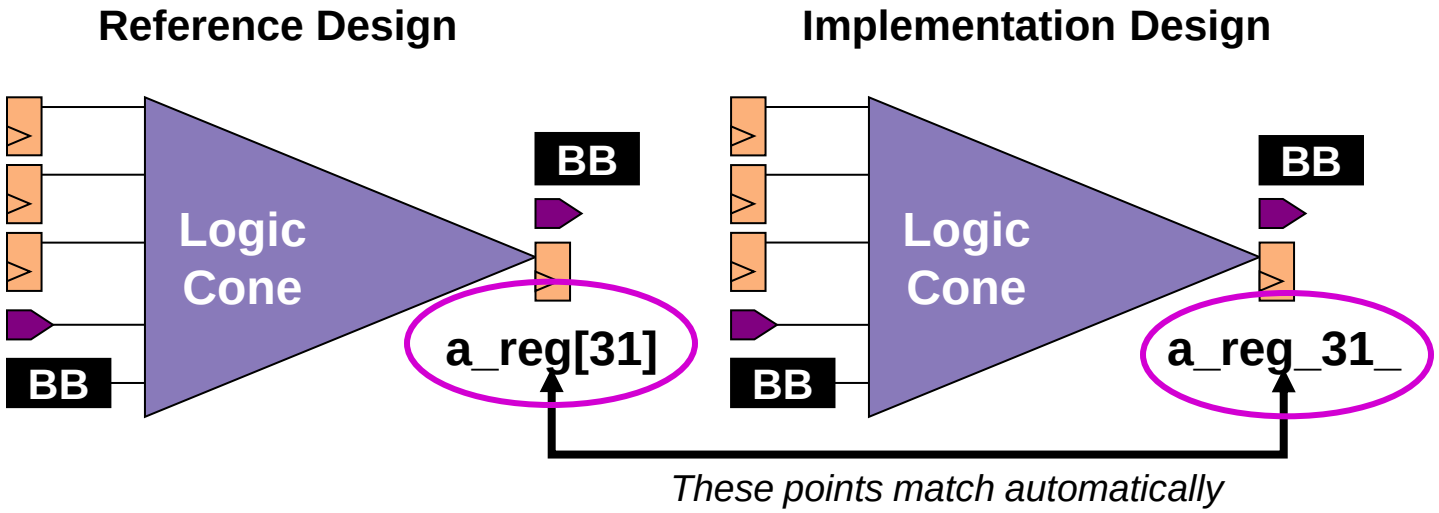
The Design Read Cycle

- Breaks Designs into Logic Cones



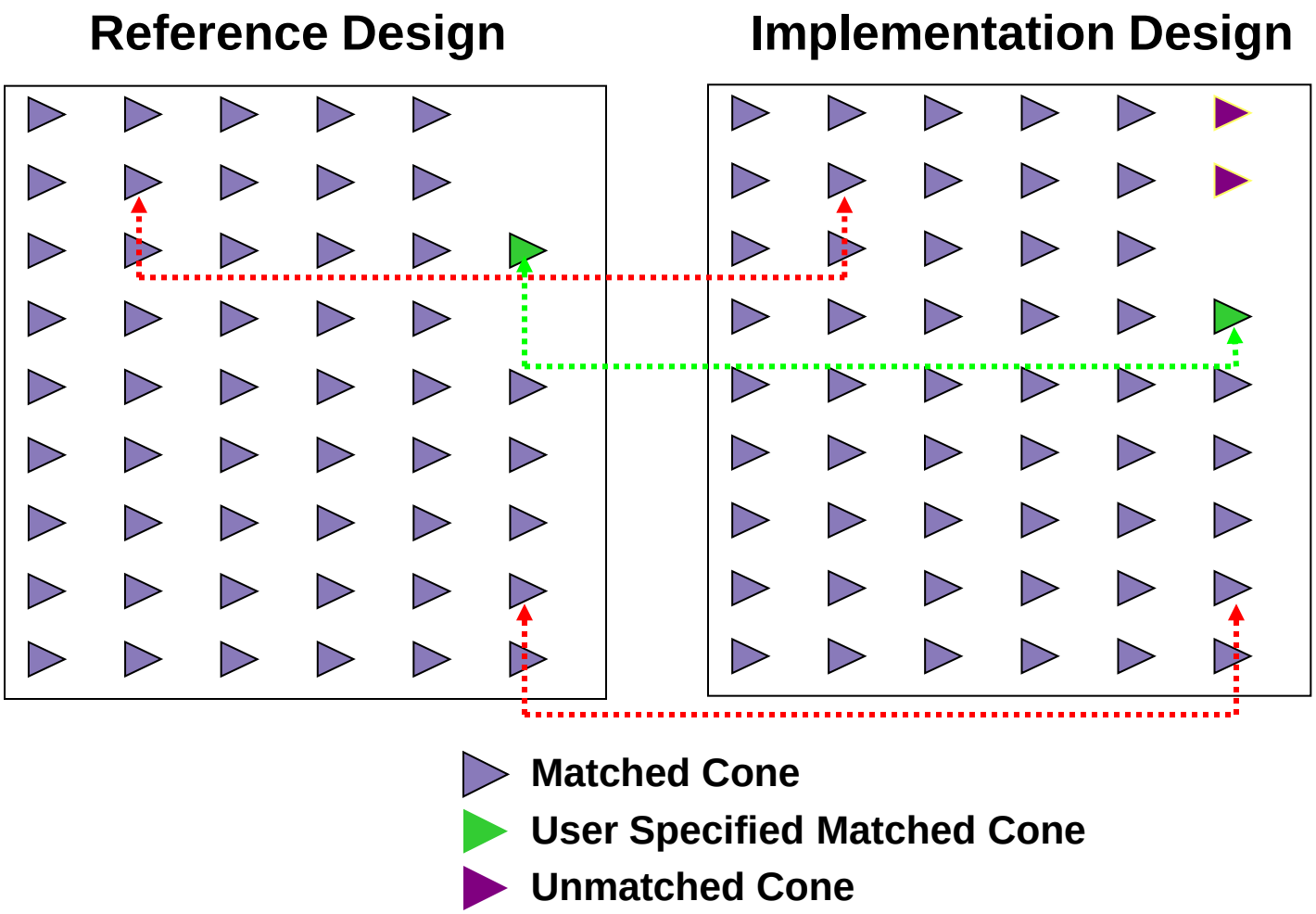
The Matching Cycle

- Matches corresponding points between designs



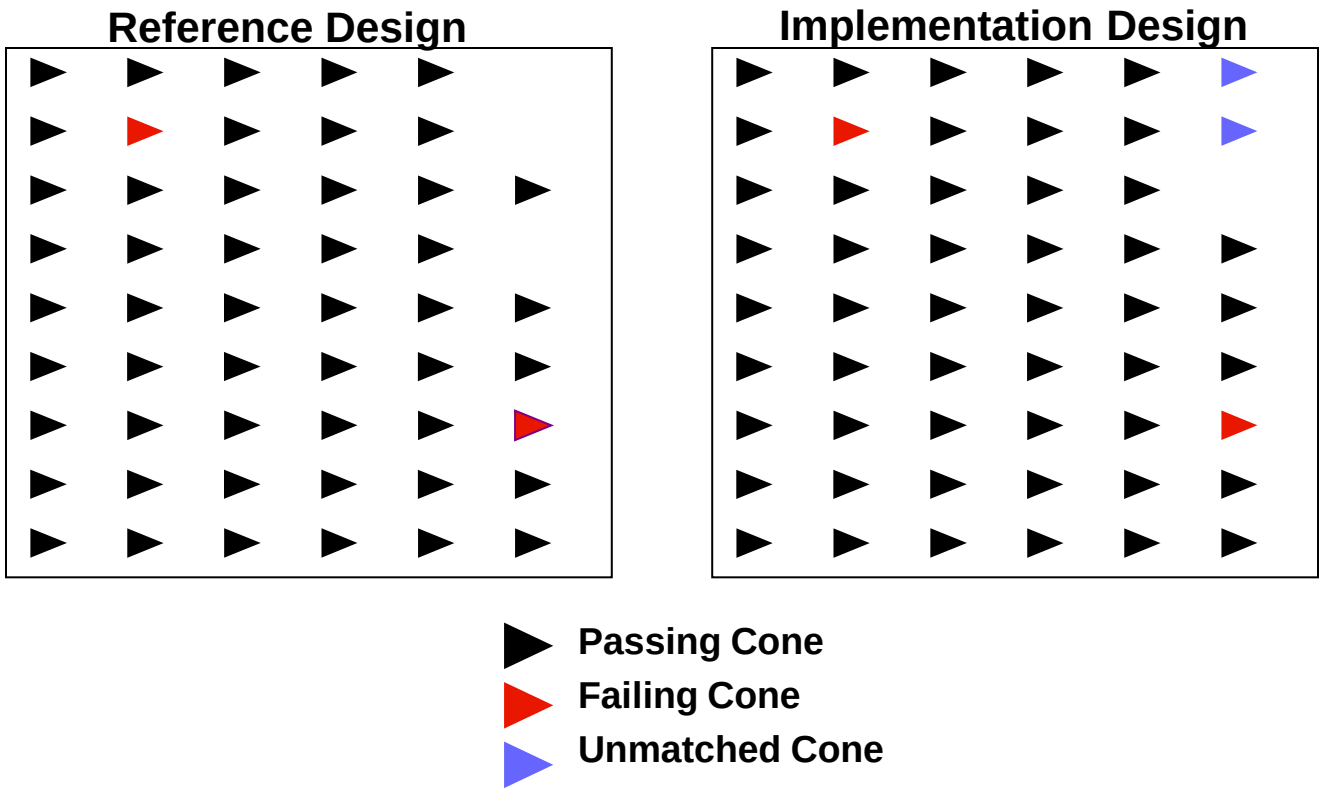
Most compare points match by name. Those that don't need guidance information, manual matching, or compare rules.

The Matching Cycle



The Verification Cycle

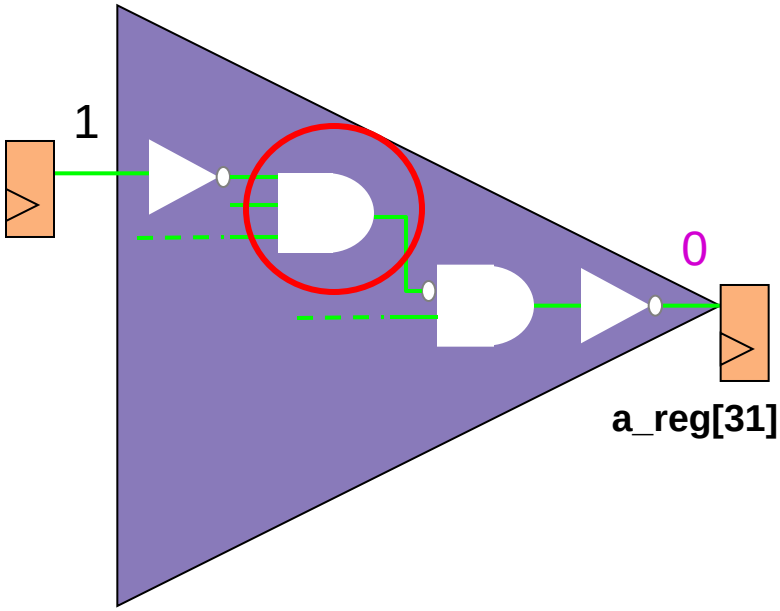
- Verifies logical equivalence for each logic cone



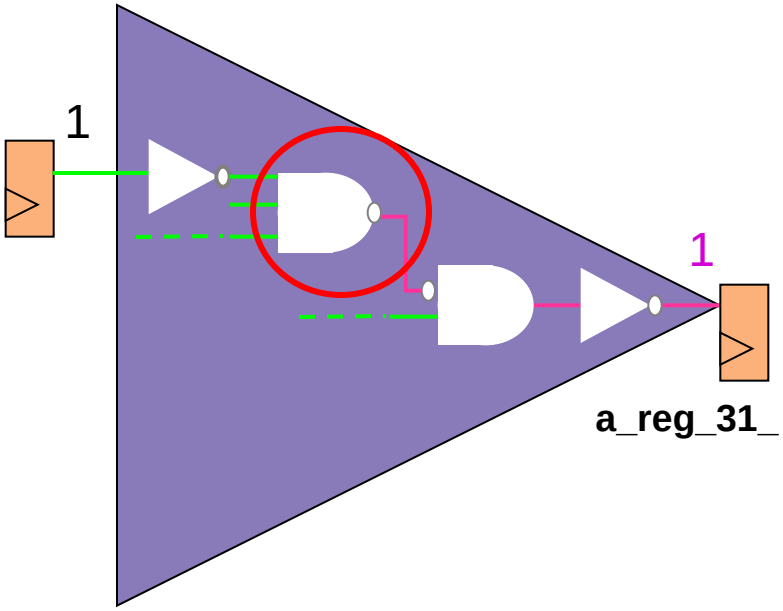
The Debug Cycle

- Isolation of implementation errors

Reference Design Cone



Implementation Design Cone



Agenda

- Introduction to Equivalence Checking
- Using Formality
  - Flow Overview
  - Guidance
  - Read
  - Setup
  - Match
  - Verify
  - Debug
- Documentation and Help
- Lab Exercises

Invoking Formality

- To run a typical Formality Tcl script

```
% fm_shell -f runme.fms |tee runme.log
```

- To start the GUI from UNIX

```
% formality
```

or

```
% fm_shell -gui -f runme.fms |tee runme.log
```

- To start the GUI within a batch session

```
fm_shell (setup)> start_gui
```

- To view other invocation options

```
% fm_shell -help
```

Files that Formality Generates

- Record of commands issued
  - fm_shell_command.log
- Log file that stores informational messages
  - formality.log
- Working files
  - FM_WORK directory
  - fm_shell_command.lck and formality.lck
  - Formality automatically deletes all working files when you exit the tool (gracefully)

Formality Setup File

- Formality reads the `.synopsys_fm.setup` file when invoked.
- A typical setup file contains commands such as:

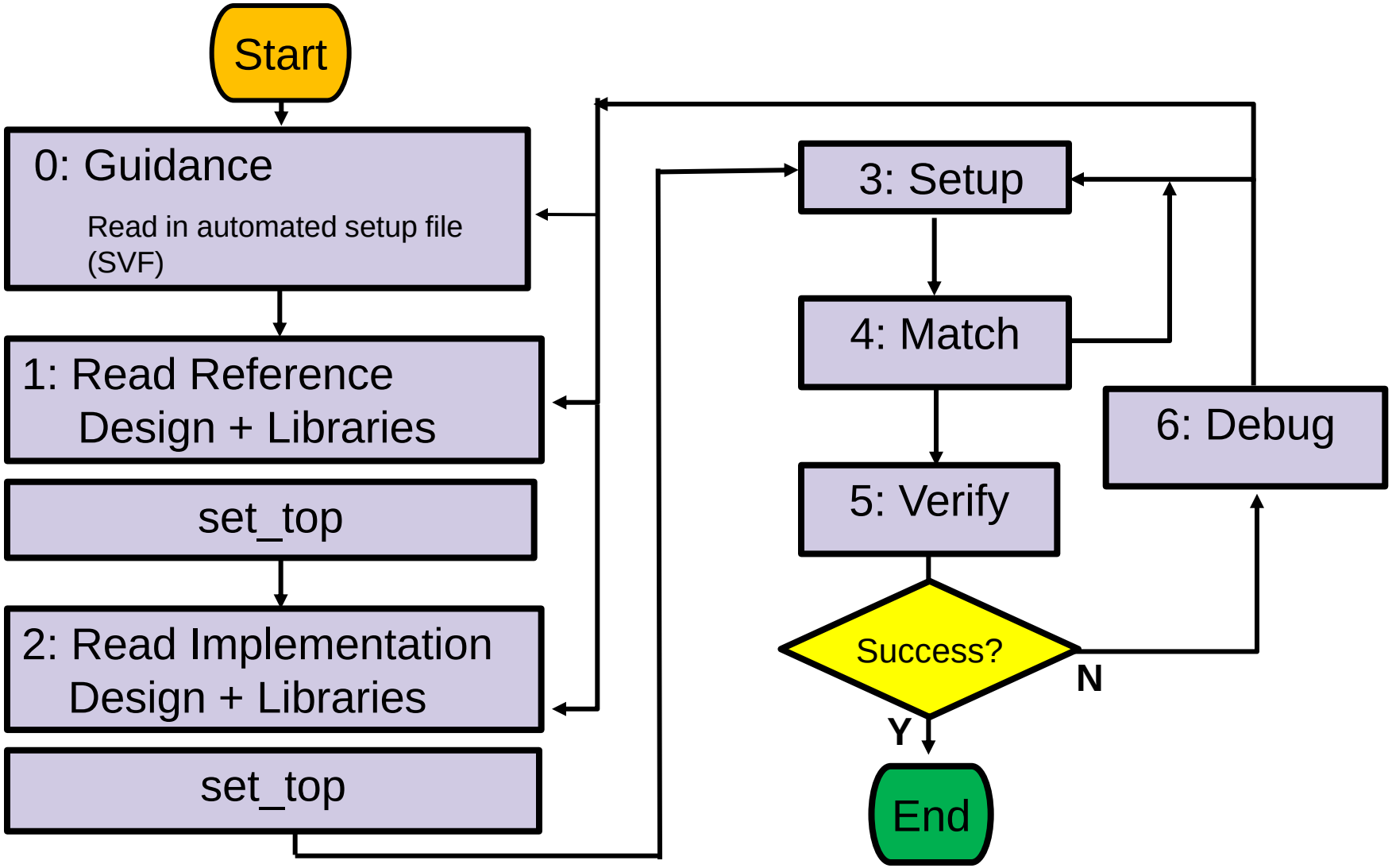
```
set search_path ". ./lib ./netlists ./rtl"
alias h history
```

- Formality reads this file from the following locations:
  1. The Formality installation directory: *`formality_root`*`/admin/setup/.synopsys_fm.setup`
  2. Your home directory
  3. The current working directory
- The setup is a cumulative effect of all three files.

Agenda

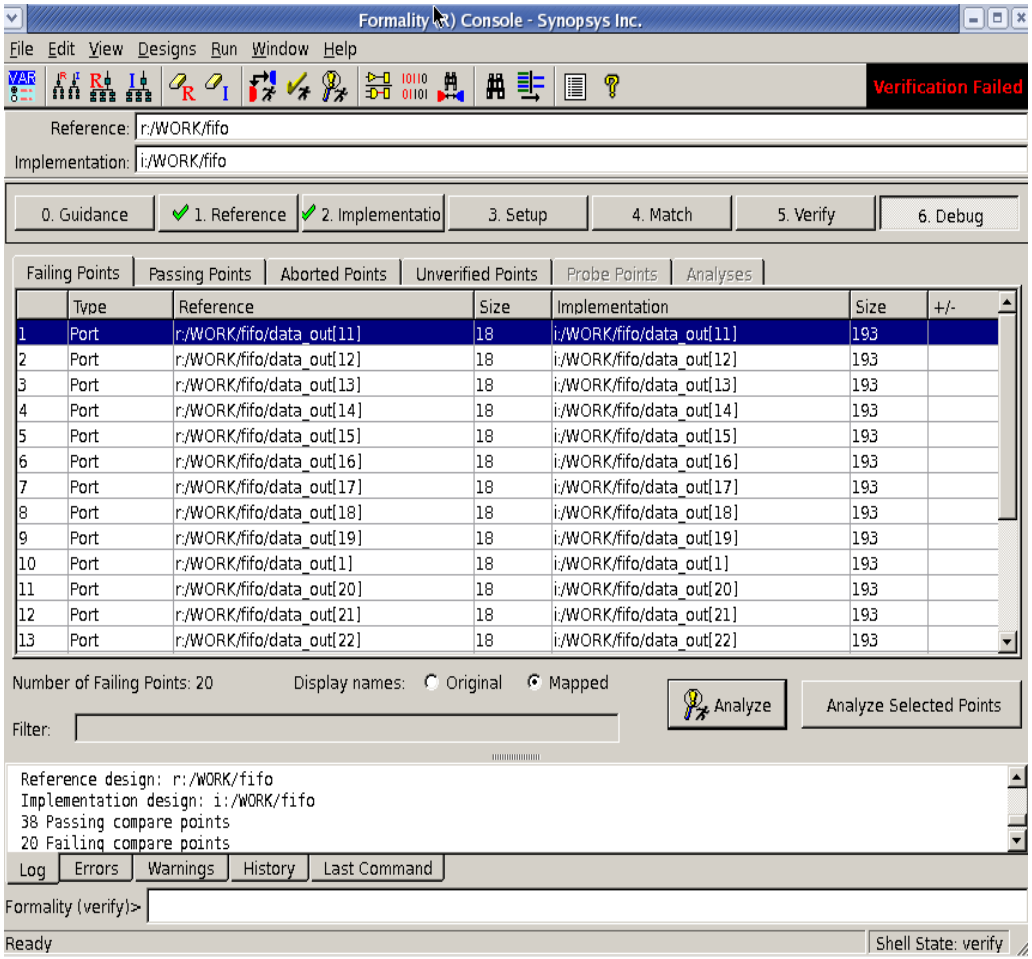
- Introduction to Equivalence Checking
- Using Formality
  - Flow Overview
  - Guidance
  - Read
  - Setup
  - Match
  - Verify
  - Debug
- Documentation and Help
- Lab Exercises

Formality Flow Overview



The Formality GUI

- Recommended for new users
- Guides you through the flow
- Has context-sensitive help
- Tabs for each step of the flow
- You do not have to remember the Tcl syntax
- Displays the corresponding Tcl commands
- GUI preferences are stored in the `~/.synopsys_fmg` folder



A Basic Formality Script

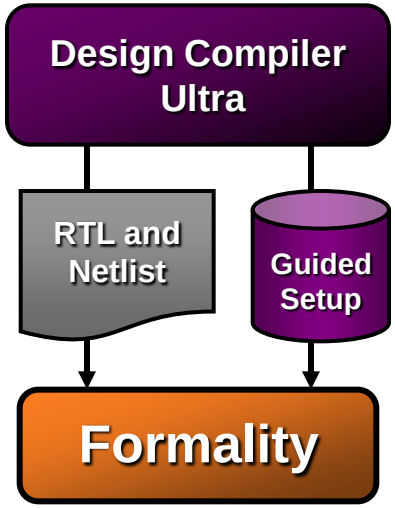
```
#Step 0: Guidance
set_svf default.svf

#Step 1: Read Reference Design
read_verilog -r alu.v
set_top alu

#Step 2: Read Implementation Design
read_db -i lsi_10k.db
read_verilog -i alu.fast.vg
set_top -auto

#Step 3: Setup
#No setup required here

#Steps 4 & 5: Match and Verify
verify
```

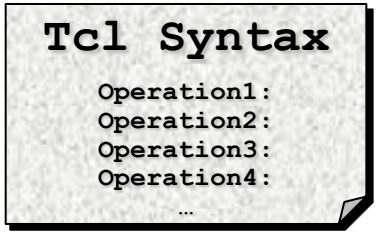
Agenda

- Introduction to Equivalence Checking
- Using Formality
  - Flow Overview
  - Guidance
  - Read
  - Setup
  - Match
  - Verify
  - Debug
- Documentation and Help
- Lab Exercises

What is Guidance?

- SVF – Automated Guidance Setup file
- Hints passed from Design Compiler to Formality
  - Automatically generated by Design Compiler
  - Contains both setup and guidance information
  - Reduces user setup effort and errors
  - Removes unnecessary verification iterations
- SVF data is implicitly or explicitly proven in Formality, or it is not used
- Using the SVF flow is recommended
  - Required when verifying a netlist containing retiming, register merging, or register inversions





Guidance File Contents

- SVF contains the following information:
  - Object name changes
  - Constant register optimizations
  - Duplicate and merged registers
  - Multiplier and divider architecture types
  - Datapath transformations
  - Finite State Machine (FSM) re-encoding
  - Retiming
  - Register phase inversion

Using the Automated Setup File

- By default, Design Compiler names the automated setup file *default.svf*
  - To specify the name, use the `set_svf file.svf` command in Design Compiler
- Formality uses the same command to read automated setup files: `set_svf file.svf`
  - Specify one file, multiple files, or a directory
  - SVF guidance is specified by the design name
  - Automatically determines multiple SVF file processing order
  - Places the *formality_svf* in the current working directory
    - Creates ASCII text version “svf.txt”

Using Feature: Auto Setup Mode

- Variable `set synopsys_auto_setup true`
  - Assumptions made in Design Compiler are also made in Formality
  - Increases out-of-the-box (OOTB) verification success rate
  - Set the auto setup variable before running the `set_svf file.svf` command
- Works with or without the SVF, does more with SVF
  - Handles undriven signals like synthesis
  - RTL interpretation like synthesis
  - Auto-enable clock-gating and auto-disable scan (requires SVF)
- You can overwrite the SVF passed variables and commands
  - Transcript summary shows variable settings
  - Variables take the last value that was set

What Auto Setup Mode Does

- Runs the following commands by default (and more):

```
set hdlin_error_on_mismatch_message false
set hdlin_ignore_parallel_case false
set hdlin_ignore_full_case false
set svf_ignore_unqualified_fsm_information false
set verification_set_undriven_signals synthesis
set verification_verify_directly_undriven_output false
```

- Design Compiler places additional setup information in the SVF
  - Clock-gating notification
  - Scan mode disable information

Benefits of Auto Setup Mode

- Formality is easier to use
- Reduces the need for debugging
  - A large percentage of failing verifications are “false failures” caused by incorrect or missing setup in Formality
- Improves productivity
  - Dramatically reduces manual setup
- Simplifies overall verification effort

Formality Script Generator

- The **`fm_mk_script`** command automatically generates Formality Tcl script using information in the SVF
- Syntax:

  `fm_mk_script <svf…> [-o[utput] <file>]`

  where

  `<svf…>` : List of SVF files or directory

  `-o[utput]` : Specifies the output script to be created. Default name is fm_mk_script.tcl

- Examples:

  The **`fm_mk_script default.svf`** command creates the *fm_mk_script.tcl* script.

  The **`fm_mk_script default.svf -output fm.tcl`** command creates the *fm.tcl* script

Agenda

- Introduction to Equivalence Checking
- Using Formality
  - Flow Overview
  - Guidance
  - Read
  - Setup
  - Match
  - Verify
  - Debug
- Documentation and Help
- Lab Exercises

Read Commands

- Formality input formats:
  - Verilog (synthesizable subset) - `read_verilog`
  - VHDL (synthesizable subset) - `read_vhdl`
  - SystemVerilog (synthesizable subset) - `read_sverilog`
  - Synopsys Milkyway - `read_milkyway`
  - Synopsys binary files - `read_db`, `read_ddc`
- Designs are read into containers

  `-r` # default reference container

  `-i` # default implementation container

  `-container containerID` # Other container name
- Link top-level of design by using the `set_top` command
  - Load all required designs and libraries before running the `set_top` command
  - Elaborate each container before loading subsequent containers

Reading in Libraries

- Verilog Simulation Libraries
  - Use the `-vcs` option of the `read_verilog` command

    Example: `read_verilog -i top.vg -vcs "-y ./lib +libext+.v"`
- Design Libraries
  - Use `read_verilog -tech design.v`

    or `read_vhdl -tech design.vhd`
  - Subsequent containers will have access to this library
  - Use the `-r` or `-i` options to place library only within the specified containers

Reading in Libraries

- Synopsys binary libraries (.db file format libraries)
  - Use the `read_db` command

    Example: `read_db lsi_10k.db`
  - Shared technology libraries
  - Subsequent containers will have access to this library
  - Use the `-r` or `-i` options to place the library in the reference or implementation containers
- Instantiated DesignWare components
  - Set the `hdlin_dwroot` variable to the top-level of Design Compiler software tree.
- Note that pure RTL does not require any component library

Linking and Referencing Designs

- After reading in the source files, use the **`set_top`** command to elaborate or link the design and designate the top-level module.
  - Using default containers (“r” and “i”) the **`set_top`** command automatically designates which design is reference or implementation.
  - Using non-default container names, specify which container is reference or implementation.
    - **`set_reference_design`**
    - **`set_implementation_design`**
- After **`set_top`** has been completed,
  - The **`$ref`** Tcl variable specifies the reference design
  - The **`$impl`** Tcl variable specifies the implementation design
- The syntax of **`$ref`** and **`$impl`** is:
  - ContainerName:/Library/Design
  - Examples:
    r:/DESIGN/chip
    i:/WORK/alu_0

Reference Design

```
fm_shell (setup)> read_verilog -r alu.v
fm_shell (setup)> set_top -auto
```

- The `read_verilog` command loads design into a container
  - The `-r` option signifies the (default) reference container
- This script does not load a technology library into the reference container
  - The file alu.v is pure RTL (no mapped logic)
- The `set_top -auto` command finds and links the top-level module
  - The `set_top` command uses the current container ("r")
  - The top-level module found by Formality is "alu"
  - Since the current container is "r", Formality automatically sets the `set_reference_design` variable (`$ref`) to r:/WORK/alu
    - WORK is the default library name

Reading and Linking

Example

```
read_ver -r { controller.v  multiplier.v  top.v }
…
set_top  r:/WORK/top
Setting top design to 'r:/WORK/top'
Status:   Elaborating design top   ...
Warning: Cannot link cell '/WORK/top/add1' to its reference design 'adder'. (FE-LINK-2)
Warning: Cannot link cell '/WORK/top/add2' to its reference design 'adder'. (FE-LINK-2)
Status:   Elaborating design controller   ...
Error: Unresolved references detected during link. (FM-234)
Error: Failed to set top design to 'r:/WORK/top' (FM-156)
0
read_ver -r adder.v
No target library specified, default is WORK
Loading verilog file '/…/rtl/adder.v'
1
set_top  r:/WORK/top
Setting top design to 'r:/WORK/top'
Status:   Elaborating design adder   ...
Status:  Implementing inferred operators...
Top design successfully set to 'r:/WORK/top'
Reference design set to 'r:/WORK/top'
```

Implementation Design

```
fm_shell (setup)> read_verilog -i alu.vg
fm_shell (setup)> read_db -i class.db
fm_shell (setup)> set_top alu_0
```

- The **read_verilog** command loads the implementation design.
  - The **-i** option specifies the (default) implementation container.
- The **read_db** command loads the technology library ***class.db***
  - Because the **-i** option is specified, this library is visible only in the implementation container
- The **set_top** command links top-level module ***alu_0***
  - The script reads both design and technology library before **set_top**
  - The **set_top** command uses the current container ("i")
  - Since the current design is "i", Formality automatically sets the implementation design variable (**$impl**) to i:/WORK/alu_0
    - WORK is the default library name
    - The script specifies that the top level module is ***alu_0***

Simulation-Style Verilog Read

- The **read_verilog** supports VCS style switches
- **read_verilog -r top.v -vcs "switches"**

  where **"switches"** include:

  -y <directory_name>

  Search <directory_name> for unresolved modules

  -v <file_name>

  Search <file_name> for unresolved modules

  +libext+<extension>

  Look at files with this extension, typically "*.v*" or "*.h*"

  +define+ : Define values for Verilog parameters

  +incdir <dirname> : Directory containing `include files

  -f <file_name> **:** VCS option file supported
- Use the **-vcs** option only <u>once</u> for each container.

Reading and Writing Containers

- Command: **`write_container`**
  - Saves all design information in the current elaborated state, including libraries, to a file
  - Recommended: Run the **`set_top`** command before saving the container
    - Can save without running the **`set_top`** command using the **`-pre_set_top`** option
- Command: **`read_container`**
  - Restores a design
- Recommended to save containers before running **`match`**
  - SVF processing can change the contents of the container
- Complete containers can be used with any version of Formality

```
fm_shell> write_container -replace -r ref.fsc
fm_shell> read_container -r ref.fsc
```

Save and Restore Session

- Use after verification to save the current state of Formality.
- Commonly used to debug failing verification in a separate Formality run.
- Saved sessions are not portable across Formality releases.

```
fm_shell> save_session -replace mysession_file
fm_shell> restore_session mysession_file.fss
```

Agenda

- Introduction to Equivalence Checking
- Using Formality
  - Flow Overview
  - Guidance
  - Read
  - Setup
  - Match
  - Verify
  - Debug
- Documentation and Help
- Lab Exercises

Setup Needed for Verification

- Guidance might be needed for matching and verification
  - Recommended: Use the automated setup file (SVF)
    - Essential for retiming, register merging, or register inversion
- Design transformations that may need setup:
  - Internal Scan
  - Boundary Scan
  - Clock-gating
  - Clock Tree Buffering
  - Finite State Machine (FSM) Re-encoding
  - Black-boxes
- Auto Setup Mode handles most setup automatically

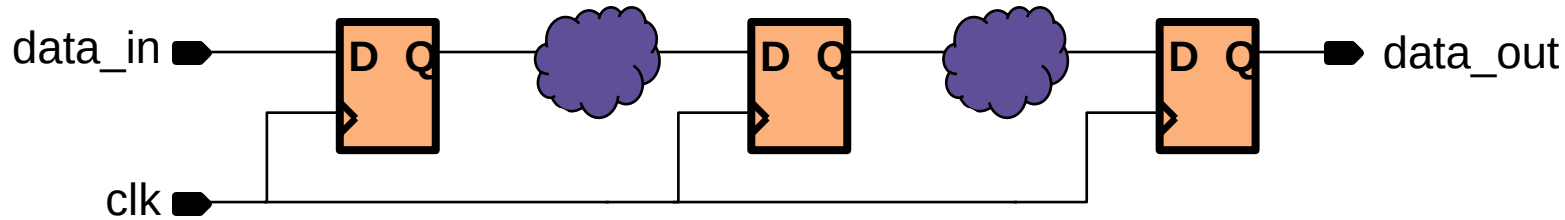
```
set synopsys_auto_setup true
```

Internal Scan: What Is It?

- Implemented by DFT Compiler
  - Replaces flip-flops with scan flip-flops
  - Connects scan flip-flops into shift registers or “scan chains”
- The scan chains make it easier to set and observe the state of registers internal to a design for manufacturing test

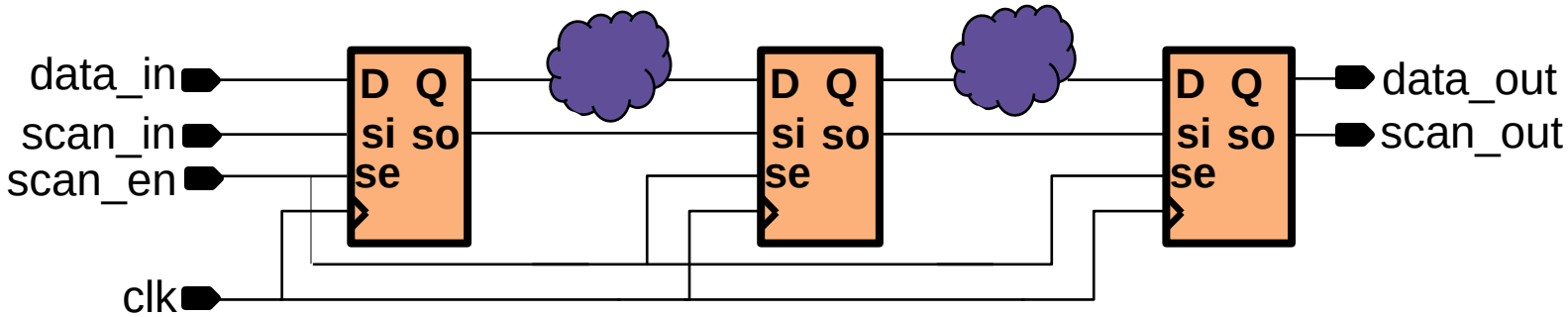
Internal Scan: Why it Requires Attention

- The additional logic added during scan insertion changes the combinational function



Pre-Scan

Post-Scan



Internal Scan: How to Deal With It

- Determine which ports disable the scan circuitry
  - Default for DFT Compiler is test_se
- Set those ports to the inactive state using the **set_constant** command

```
fm_shell (setup) > set_constant i:/WORK/TOP/test_se 0
```

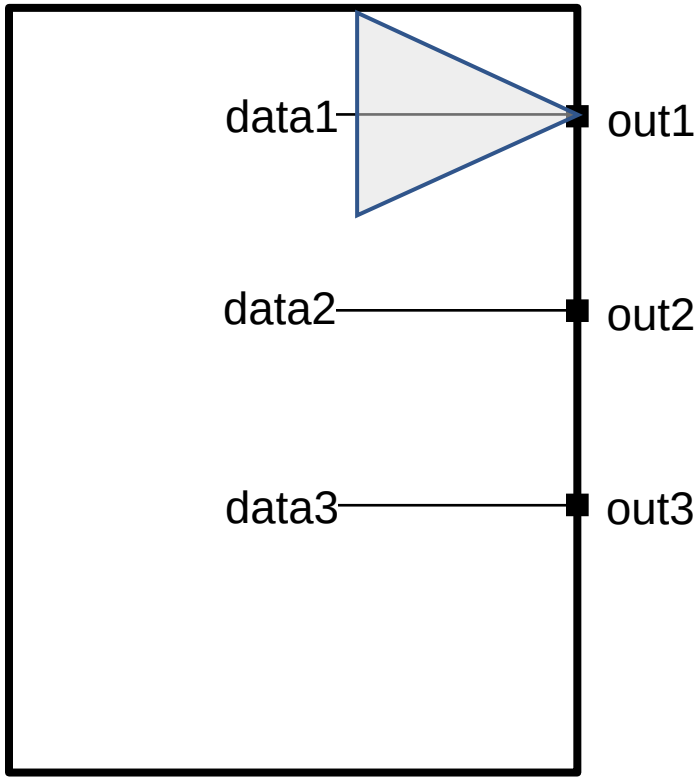
Boundary Scan: What Is It?

- Boundary scan involves the addition of logic to a design:
  - The added logic makes it possible to set and or observe the logic values at the primary inputs and outputs (the boundaries) of a chip
  - Used in manufacturing test at board and system level
  - Added by BSD Compiler
- Boundary scan is also referred to as
  - The IEEE 1149.1 specification
  - JTAG

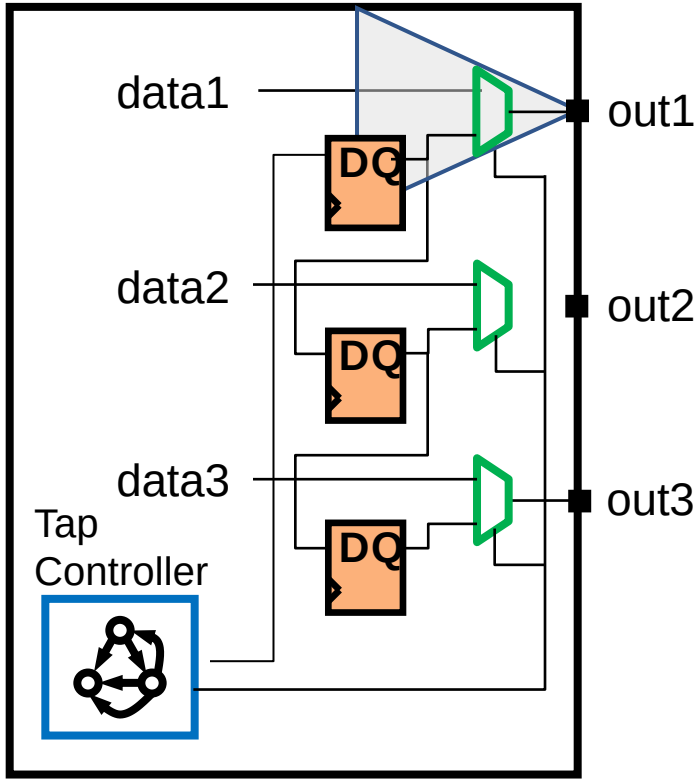
Boundary Scan: Why it Requires Attention

- The logic cones at the primary outputs are different
- The logic cones driven by primary inputs are different
- The design has extra state holding elements

Pre-Boundary Scan



Post-Boundary Scan



Boundary Scan: How to Deal With it

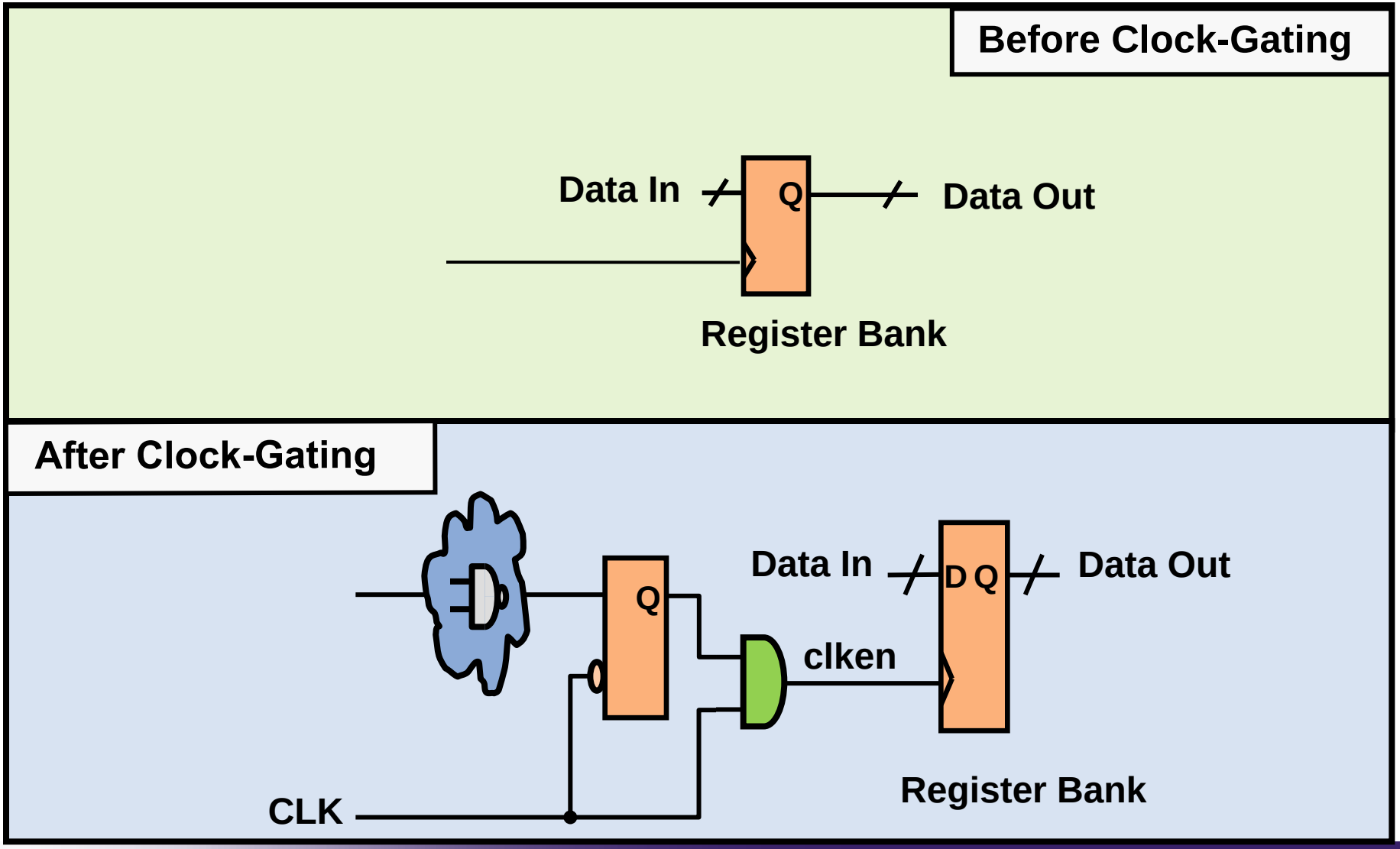
- Disable the Boundary scan:
  - If the design has an optional asynchronous TAP reset pin (such as TRSTZ or TRSTN), use **set_constant** on the pin to disable the scan cells
  - If the design has only the 4 mandatory TAP inputs (TMS, TCK, TDI and TDO), then force an internal net of the design using the **set_constant** command

```
fm_shell (setup) > set_constant i:/WORK/TOP/TRSTZ 0
fm_shell (setup) > set_constant i:/WORK/alu/somenet 0
```

Clock-Gating: What is it?

- Added by Power Compiler
- Addition of logic in a register's clock path, which disables the clock when the register output is not changing
- Saves power by not clocking register cells unnecessarily

Clock-Gating



Clock-Gating: Why is it an Issue?

- Without intervention, compare points will fail verification
  - A compare point is created for each clock-gating latch
    - This compare point does not have a matching point in the other design and will fail
  - The logic feeding the clock input of the register bank has changed
    - The register bank compare points will fail

Clock-Gating: How to Deal With it

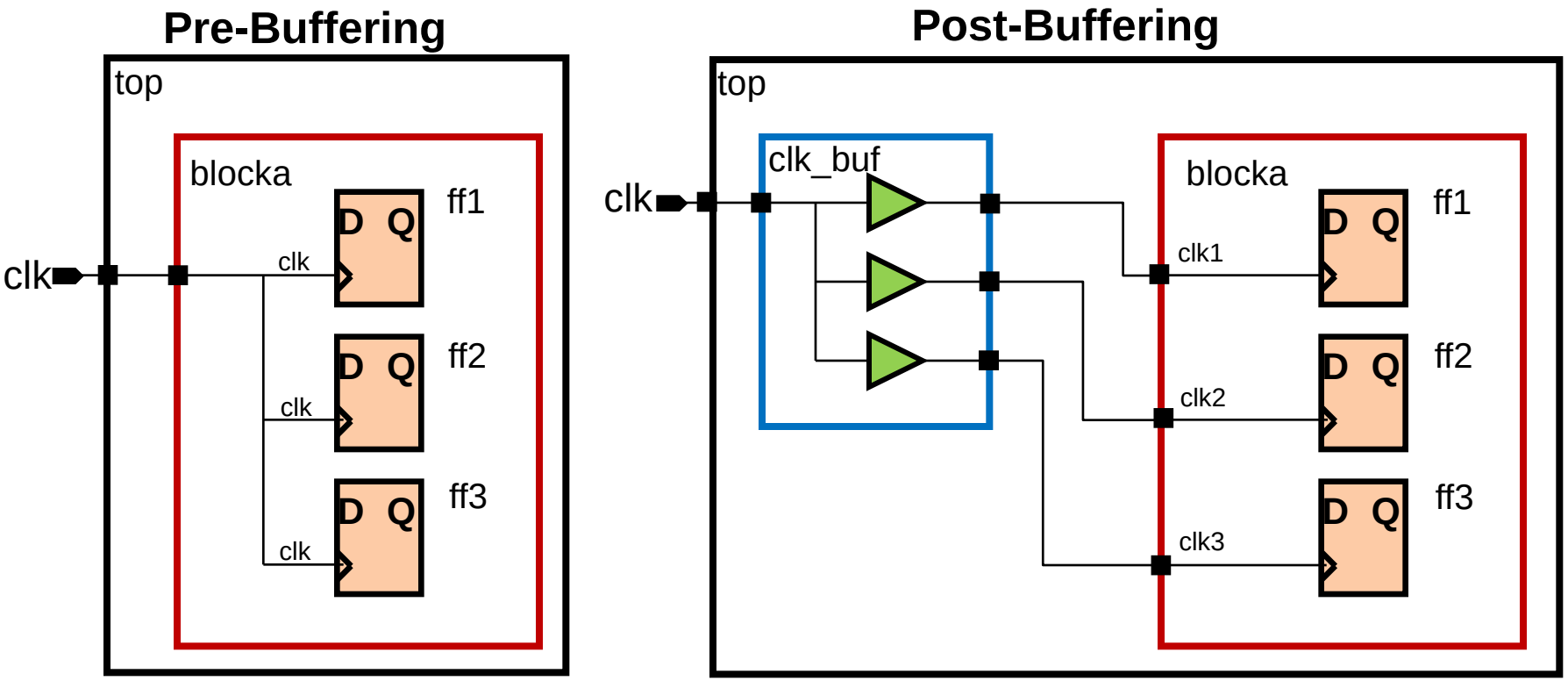
- Use: **set verification_clock_gate_hold_mode low**
  - Use option **low** or **any** if the clock-gating net drives the clock pin of positive edge-triggered DFF
  - If the clock-gating net also drives primary outputs or black-box inputs use, the **collapse_all_cg_cells** option
  - Use the **set_clock** command to identify the primary input clock net if clock-gating cells do not drive any clock pin of a DFF
- Auto setup mode enables clock-gating by default
- Use the following variable only if clock-gating verification issues continue:

  set verification_clock_gate_edge_analysis true

```
fm_shell (setup)> set verification_clock_gate_hold_mode low
```

Clock Tree Buffering

- Clock tree buffering is the addition of buffers in the clock path to allow the clock signal to drive large loads.



Clock Tree Buffering: How to Deal With It

- Verification at the top level requires no setup
- When verifying at "blocka" sub-block level use the **set_user_match** command to show that the buffered clock pins are equivalent

```
fm_shell (setup)> set_reference_design r:/WORK/blocka
fm_shell (setup)> set_implementation_design i:/WORK/blocka
fm_shell (setup)> set_user_match r:/WORK/blocka/clk \
i:/WORK/blocka/clk1 \
i:/WORK/blocka/clk2 \
i:/WORK/blocka/clk3
fm_shell (setup)> verify
```

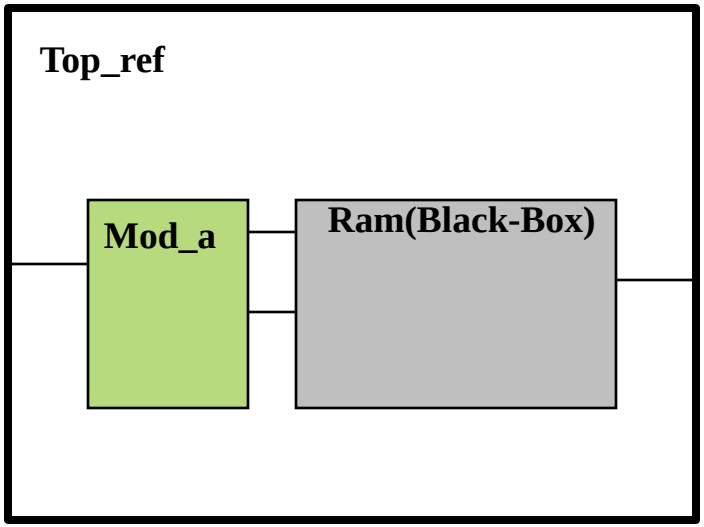
Finite State Machine Re-encoding

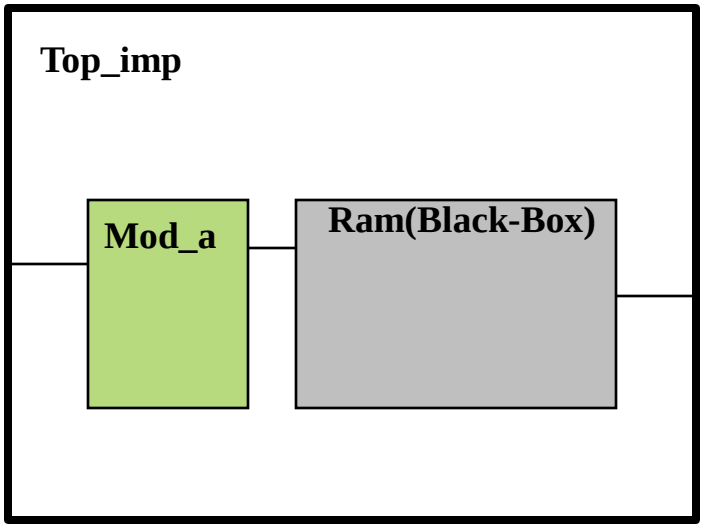
- Verify that the re-encoding in the automated setup file (SVF) is correct
  - View the ASCII file: ./formality_svf/svf.txt
- Enable the use of this setup information in Formality
- Auto setup mode will enable use of this FSM information by default

```
fm_shell> set svf_ignore_unqualified_fsm_information false
```

Black Boxes

- A black box is a module or entity which contains no logic
  - These are modules that are not verified
    - Analog circuitry
    - Memory devices
  - Match black boxes between reference and implementation





How Are Black Boxes Created?

- RTL modules that have only I/O port declarations are read in
- Library .db cells with port and timing information only
  - Typically a memory
- Missing a piece of design and are using this variable:

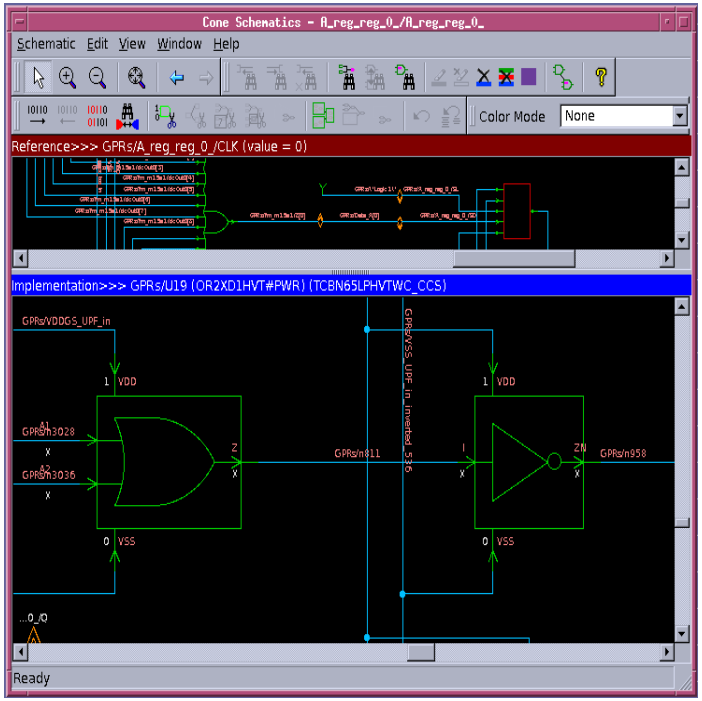
  `set hdlin_unresolved_modules black_box`
- Usage of other variable when reading in designs:

  `set hdlin_interface_only "SRAM* dram16x8"`
  - Any module beginning with SRAM and the dram16x8 module will become a black-box
- Declare a sub-design as a black-box

  `set_black_box designID`
- Command `report_black_boxes` shows list of black-boxes

Formality Low Power Capabilities

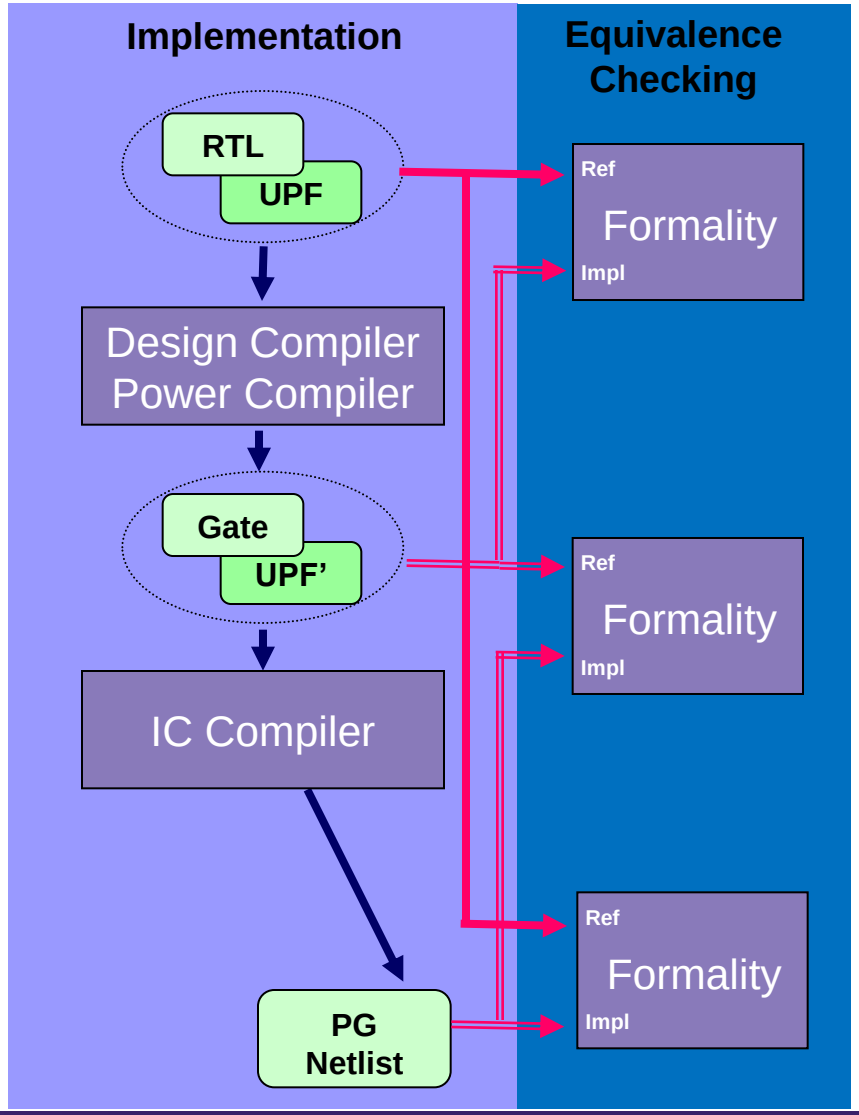
- Complete low-power static verification solution
- Adheres to IEEE 1801 (UPF)
- Comprehensive low-power checks
  - Verifies all legal power states as defined in the power state table
    - Including power-up and power-down states
  - Supports advanced low-power design techniques
  - Supports special low-power cells
- Complemented by MVRC for static low-power rule checking



SYNOPSYS® Predictable Success

Low Power Verification Flow

- Data Requirements
  - RTL and UPF should be simulated with Synopsys tool MVSIM
  - Design Compiler netlist can be either Verilog or DDC
  - UPF must be from the Design Compiler command `save_upf`
  - Technology libraries
    - Power aware cells must have power pins and power down functions
    - Formality can create power pins for standard logic function cells



Formality's Support of Low Power Designs

- Formality checks for functional equivalence
  - Functional comparison of post layout PG netlist against original low power design specification (RTL+UPF)
- Formality supports special low power cells
  - isolation cells, level shifters, always on cells, retention registers, power gates
- Synopsys tool MVRC checks for adherence to multi-voltage rules and power intent specification
  - Level shifter or isolation cell missing, level shifter or isolation cell interchange, illegal routing and placement, polarity of isolation signal
  - Checks power up and down sequence, control signal networks

Formality Tcl Command `load_upf`

- Use the `load_upf` command after running the `set_top` command
- Example script for RTL and UPF versus Post-DC-netlist+UPF

```
read_db {low_power_library.db special_lp_cells.db}
read_verilog -r { top.v block1.v block2.v block3.v }
set_top r:/WORK/top
load_upf -r top.upf
read_verilog -i { post_dc_netlist.v }
set_top i:/WORK/top
load_upf -i top_post_dc.upf
```

- Formality modifies the reference or implementation design to meet the specification implied by the UPF commands
- UPF commands cannot be issued interactively

Agenda

- Introduction to Equivalence Checking
- Using Formality
  - Flow Overview
  - Guidance
  - Read
  - Setup
  - Match
  - Verify
  - Debug
- Documentation and Help
- Lab Exercises

Matching Compare Points

```
fm_shell (setup)> match
```

- The `match` command is optional
  - The `verify` command will also run matching
  - Recommendation:
    - For interactive work use the match command for feedback
    - Omit the match command from scripts to reduce runtime
- Name matching algorithms are used first
- Remaining unmatched points matched by signature analysis
  - Includes structural techniques
  - Signature analysis may be turned off (but not recommended)
- Any remaining unmatched points are then reported
  - User can specify compare rules or can manually set matches
- Use the SVF flow improves name matching performance and completion
  - Matches points by name without user intervention

GUI Unmatched Point Report

Formality (R) Console – Synopsys Inc.

File Edit View Designs Run Window Help

Verification Failed

Reference: r:/WORK/tv80s

Implementation: i:/WORK/tv80s

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

Compare Rule Setup | User Match Setup | Matched Points | Unmatched Points | Summary

	Reference Object	Type
1	i_tv80_core/ACC_reg[0]	DFF
2	i_tv80_core/ACC_reg[1]	DFF
3	i_tv80_core/ACC_reg[2]	DFF
4		DFF
5		DFF
6	i_tv80_core/ACC_reg[5]	DFF
7	i_tv80_core/ACC_reg[6]	DFF
8	i_tv80_core/ACC_reg[7]	DFF
9	i_tv80_core/ALU_Op_r_reg[0]	DFF
10	i_tv80_core/ALU_Op_r_reg[1]	DFF

Set User Match + − ?

	Implementation Object	Type
1	ACC_reg_0_	DFF
2		
3		
4		
5	ACC_reg_4_	DFF
6	ACC_reg_5_	DFF
7	ACC_reg_6_	DFF
8	ACC_reg_7_	DFF
9	ALU_Op_r_reg_0_	DFF
10	ALU_Op_r_reg_1_	DFF

Create a compare rule

Select two points and set match

Number of unmatched reference points: 314 Number of unmatched implementation points: 351 Display names: Original Mapped

Filter on reference list: Filter on implementation list: Run Matching

Compare Rules

- When names change in predictable ways, write a compare rule.
- Use SED syntax to translate names in one design to the corresponding names in the other design:

```
fm_shell (match)> set_compare_rule $ref    \
                  -from {i_tv80_core} -to {}
fm_shell (match)> match
```

Agenda

- Introduction to Equivalence Checking
- Using Formality
  - Flow Overview
  - Guidance
  - Read
  - Setup
  - Match
  - Verify
  - Debug
- Documentation and Help
- Lab Exercises

Verify Implementation Design

- Runs Formality's verification algorithms on compare points
  - Formality deploys many different solvers
  - Each solver uses a different algorithm to prove equivalence or non-equivalence
- Four possible results:
  - Succeeded: implementation is equivalent to the reference
  - Failed: implementation is not equivalent to the reference
    - True logic difference, or setup problem
  - Inconclusive: no points failed, but analysis is incomplete
    - Might be due to timeout or complexity
  - Not run: a problem earlier in the flow prevented verification from running at all

Verify Implementation Design (cont)

- For each matched pair of compare points, Formality
  - Confirms same functionality of logic cones
  - Marks point as “passed”

  Or,
  - Determines that functionality is different between logic cones
  - Finds one or more “counter examples” that shows different response at compare point
  - Marks the compare point as “failed”
- All valid compare points are verified
  - Constant registers are not verified
  - “Unread” compare points are not verified by default
    - Unread points do not affect other compare points or primary outputs

Verify Example

```
fm_shell (match)> verify
```

- Verification is incremental
  - Verification can continue again after being stopped
  - You may match additional compare points manually and continue with verification
  - To force verification of entire design: `verify -restart`
- Options:
  - Verification of single compare point
  - Verification against a constant: `verify $ref/cp -constant0`
  - Use `set_dont_verify` to exclude points from verification

Controlling Verification Runtimes

- `set verification_timeout_limit` ***`hrs:min:sec`***
  - Halts entire verification after a specified time
  - 36 cpu hours is default limit (0:0:0 means no timeout)
  - Remaining unverified compare points are not attempted

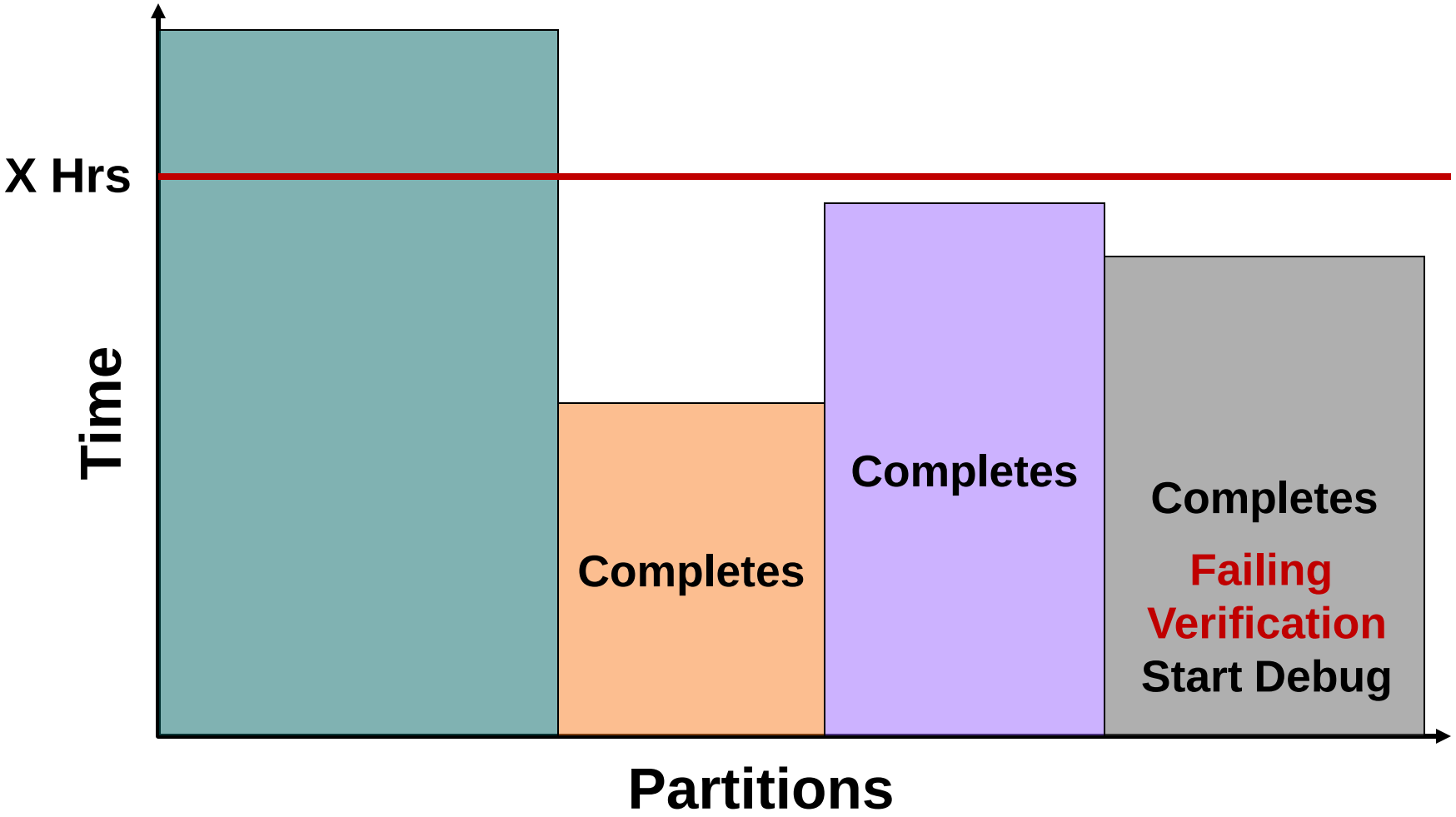
- `set verification_failing_point_limit` ***`number`***
  - Halts verification after specified number of compare points fail (default is 20 failing compare points)
  - Allows you to correct for any missing setup
  - Allows you to begin debugging failing compare points

Controlling Verification Runtimes

- `set verification_effort_level` *`[super_low | low | medium | high]`*
  - Specifies amount of effort spent solving a compare point (default level is high)
  - Using super_low finds failing compare points quickly but will also produce several aborted compare points

- `set verification_partition_timeout_limit` *`hrs:min:sec`*
  - A partition is a collection of similar logic cones
  - Verification of a partition stops after allotted time and verification moves onto next partition

Partition Time Out Limit
Find Errors Faster



Hierarchical Verification

- Command `write_hierarchical_verification_script`
  - Formality generates Tcl script that performs hierarchical verification on current reference and implementation designs
  - Helpful for debugging large and hard-to-verify designs
  - Usage:

```
…
set_top i:/WORK/top
set_constant $impl/test_se  0
write_hier -replace -level 3 myhierscript
source myhierscript.tcl
quit
```

    - View results in file fm_myhierscript.log
  - Formality will create one session file, by default, if verification fails on a sub-design

Distributed Verification

- Time-to-results advantage for long runs
  - Use on designs greater than 250K gates and
  - Taking over two hours to verify
- How it works:
  - Formality divides the design into partitions (groupings of compare points) and distributes the verification workload
- Does not require extra licenses
  - Up to four distributed verifications allowed along with master
  - No additional purchase of licenses
- Example:

```
add_distributed_processors frodo bilbo gandalf*2
```

- Support for LSF and GRD

Agenda

- Introduction to Equivalence Checking
- Using Formality
  - Flow Overview
  - Guidance
  - Read
  - Setup
  - Match
  - Verify
  - Debug
- Documentation and Help
- Lab Exercises

Example of Typical Formality Script

```
set search_path ". ./rtl ./lib ./netlist"
set synopsys_auto_setup true
set hdlin_dwroot  /tools/syn/E-2010.12

set_svf  default.svf

read_verilog -r "fifo.v  gray_counter.v  \
       pop_ctrl.v  push_ctrl.v  rs_flop.v"
set_top fifo

read_db -i tcb013ghpwc.db
read_verilog -i fifo.vg
set_top fifo

# set_constant $impl/test_se 0

verify
```

Debugging: Problem 1

Find the problem in this script:

```
read_verilog -r alu.v
set_top alu

read_verilog -i alu.fast.vg
set_top alu
read_db -i class.db

verify
```

Debugging: Problem 2

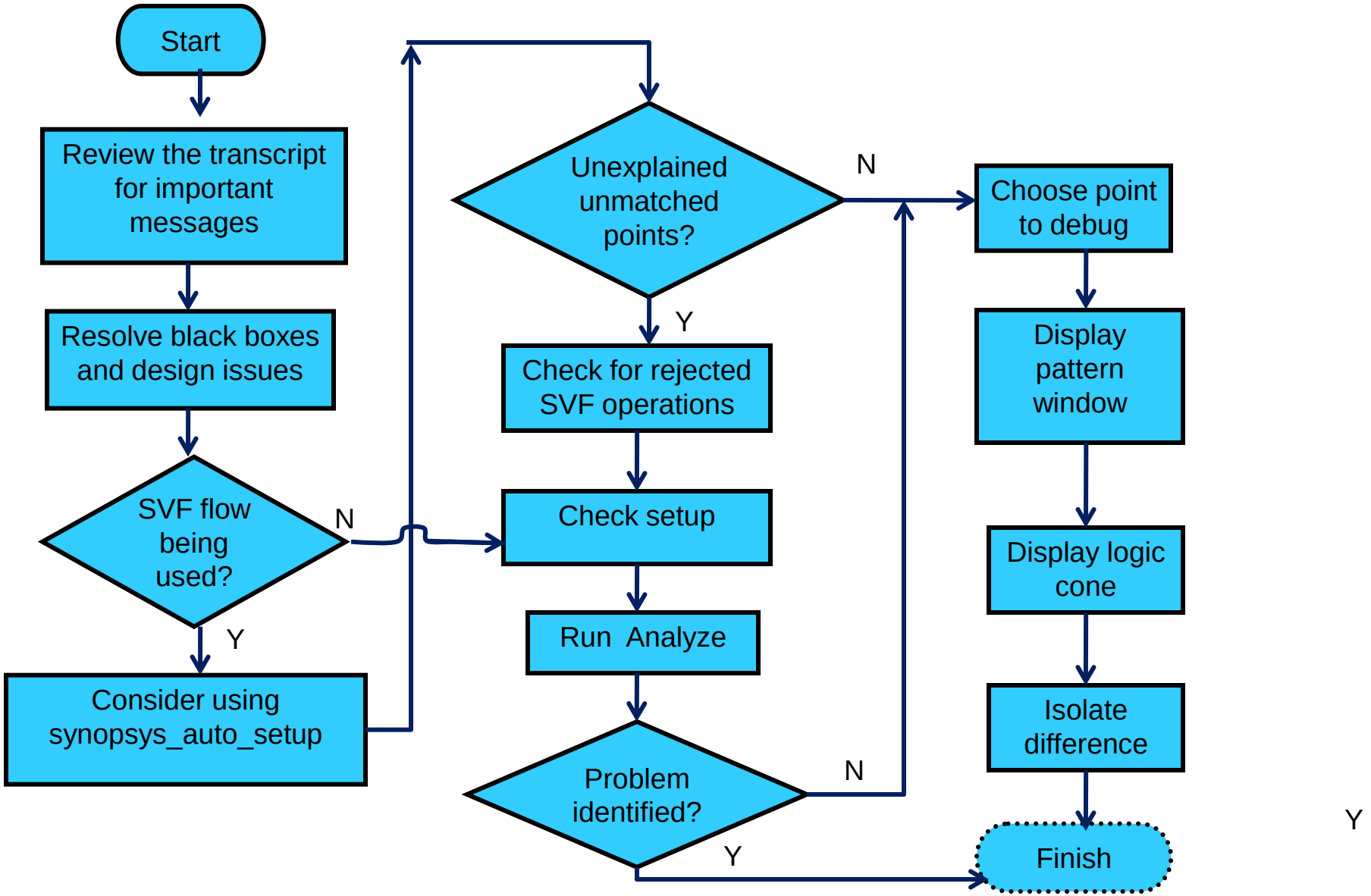
Find the problem in this script:

```
read_verilog -r alu.v

read_db -i class.db
read_verilog -i alu.fast.vg

set_top r:/WORK/alu
set_top i:/WORK/alu

verify
```

Debugging Flow

- Step 1: Look at the transcript for clues
- Step 2: Use debugging tools and commands
- Step 3: Identify and resolve problem areas
- Step 4: Try the verification again
- Step 5: Ask for help

Debugging Flow Chart



Steps of Debugging
Check for Warning Signs

- Check for simulation or synthesis mismatch errors
- Check RTL interpretation messages in transcript
- Were full_case and parallel_cases pragmas interpreted?
- Check for black-box warnings in the transcript
- Check for rejected SVF guidance commands
- Check for unmatched compare points
  - Unmatched compare points only in implementation?
  - Clock-gating latches?
- Is there a setup problem? Did you disable scan?
- Try using Auto Setup Mode
  - `set synopsys_auto_setup true`

Debugging Tools: analyze_points

- Provides debugging guidance for failing or hard to verify compare points
- Command `analyze_points`
  - Options for failing verifications: `-failing, -all`
  - Options for hard verifications: `-aborted, -unverified, -no_operator_svp, -all`
  - Takes a single or list of compare points as an argument
- Report command `report_analysis_results`
  - Options: `-summary`
- Variable `verification_run_analyze_points`
  - Default value is false
  - When enabled runs `analyze_points -all`
- For hard-to-verify compare points, the `analyze_points` command looks at datapath specific SVF operations involved with the logic cone
  - Produces DC TCL script command: `set_verification_priority`
    - Targets specific block(s), instances, or arithmetic operators
    - Turns off specific optimizations
    - Improves verification success
    - Minimizes QoR impact

Debugging Tools: analyze_points

Guidance for Failing Verifications

```
fm_shell (verify)> analyze_points -failing

********************************** Analysis Results **********************************
Found 1 Unconstrained Implementation Input
--------------------------------
Unmatched input ports in the implementation typically result
from test logic insertion. Constraining the unmatched ports
to a constant value may correct the failures.
--------------------------------
i:/WORK/aes_cipher_top/test_se
    Unmatched in the implementation cones for 20 compare point(s):
        i:/WORK/aes_cipher_top/text_in_r_reg_112_
        i:/WORK/aes_cipher_top/us22/sbox2/dreg_reg_2_
        i:/WORK/aes_cipher_top/us22/sbox2/dreg_reg_4_
        i:/WORK/aes_cipher_top/us22/sbox2/dreg_reg_7_
        {…}

    Try adding this command before verify:
        set_constant i:/WORK/aes_cipher_top/test_se 0
-----------
*******************************************************************************
Analysis Completed
```

Debugging Tools: analyze_points

Guidance for Hard Verifications

```
********************************** 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/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 } ]

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

Debugging Tools: analyze_points

Formality (R) Console - Synopsys Inc.

File Edit View Designs Run Window Help

Verification Failed

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

	Type	Reference	Size	Implementation	Size	+/-
1	DFF	/Fp_reg[1]		Fp_reg_1_		
2	DFF	/Fp_reg[4]		Fp_reg_4_		

Number of Failing Points: 2 Display names: Original Mapped

Analyze | Analyze Selected Points

Filter:

Debugging Tools: analyze_points

Formality (R) Console - Synopsys Inc.

File Edit View Designs Run Window Help

Verification Failed

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

- Unconstrained Implementation Input (1)
  - i:/WORK/top/test_se
- Rejected Guidance Type (0)
- Undriven Reference Signal (0)
- Unretimed DesignWare Component (0)
- Missing Retention Register (0)
- Modules With Rejected Datapath Guidance (0)
- Modules With Datapath Components (0)

Description - Unconstrained Implementation Input:
Unmatched input ports in the implementation typically result from test logic insertion. Constraining the unmatched ports to a constant value may correct the failures.

Recommendations:
i:/WORK/top/test_se
Unmatched in the implementation cones for 2 compare point(s):

- i:/WORK/top/Fp_reg_1_
- i:/WORK/top/Fp_reg_4_

Try adding this command before verify:
set_constant i:/WORK/top/test_se 0

Debugging Tools: Pattern Viewer

- Formality automatically creates sets of vectors to illustrate failure at the compare point
  - These counter-examples are failing patterns
  - Failing patterns are applied on the inputs of each logic cone
  - Proof of non-equivalence performed mathematically
  - No failing patterns exist for passing or hard to verify compare points
- Viewing the logic cone inputs and failing patterns are extremely helpful in debugging

Debugging Tools: Pattern Viewer

Formality (R) Console – Synopsys Inc.

File Edit View Designs Run Window Help

Reference: r:/WORK/mR4000

Implementation: i:/WORK/mR4000

0. Guidance | 1. Reference | 2. Implementation | 3. Setup | 4. Match

Failing Points | Passing Points | Aborted Points | Unverified Points | Errors

	Type	Reference	Size	Implementation
28	DFF	Instruction_reg_8_	3	Instruction_reg_8_
29	DFF	Instruction_reg_9_	3	Instruction_reg_9_
30	DFF	Instruction_reg_7_	3	Instruction_reg_7_
31	DFF	Instruction_reg_0_	3	Instruction_reg_0_
32	DFF	Instruction_reg_3_	3	Instruction_reg_3_
33	DFF	ALUOp_reg_1_	195	ALUOp_reg_1_
34	DFF	ALUSelB_reg_0_	195	ALUSelB_reg_0_
35	DFF	IRWrite_reg	195	IRWrite_reg
36	DFF	ALUSelB_reg_1_	195	ALUSelB_reg_1_

Number of Failing Points: 252

Display names: Original Mapped

Filter:

Diagnose | Diagnose Selected Points

Implementation design: i:/WORK/mR4000
98 Passing compare points
262 Failing compare points
0 Aborted compare points

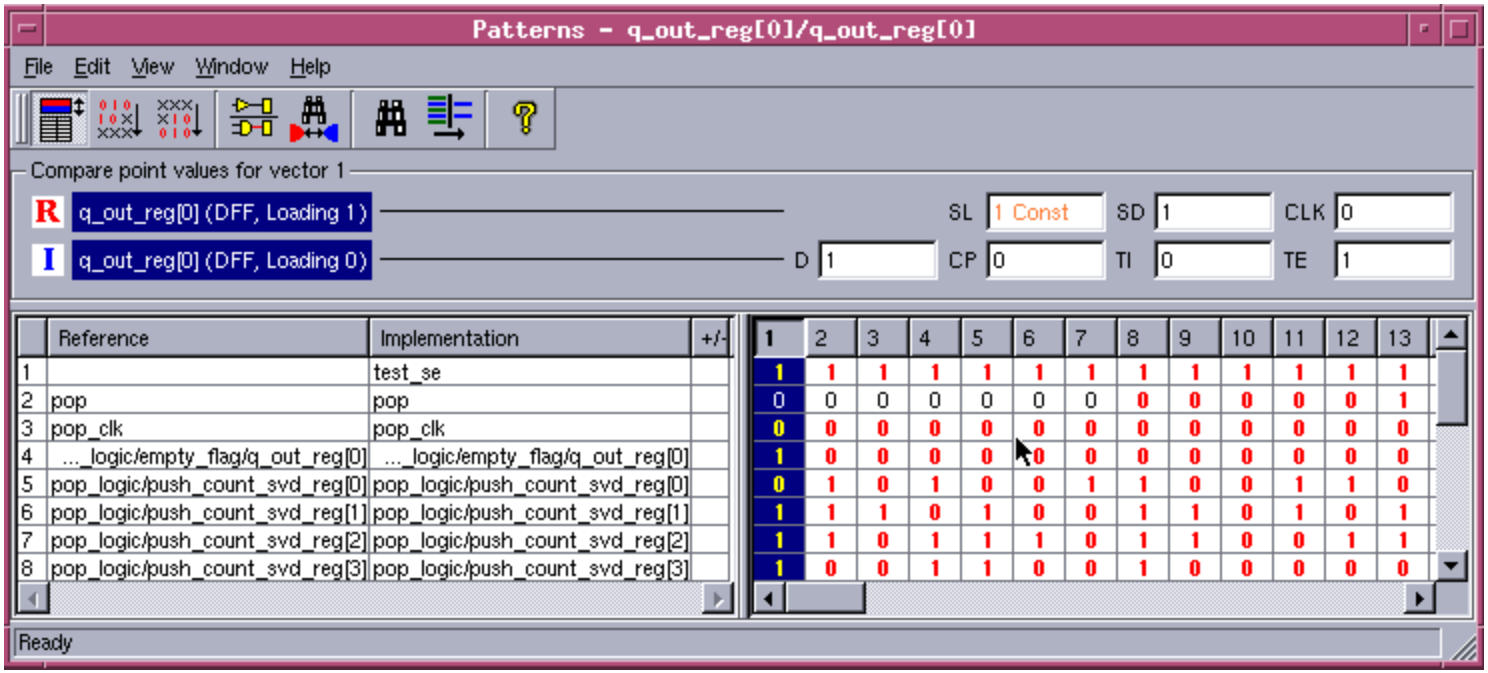
Log | Errors | Warnings | History | Last Command

Formality (verify)>

Ready | Shell State: verify

- Show Logic Cones
- Show Selected Cone Sizes
- Show All Cone Sizes
- Show Patterns
- Show Matching Tool
- View Reference Object
- View Implementation Object
- View Reference Source
- View Implementation Source
- Set Don't Verify
- Analyze
- Analyze Selected
- Diagnose
- Diagnose Selected Points
- Copy

Debugging Tools: Pattern Viewer



- Allows quick identification of issues with setup and matching
  - For this example, note failure when scan enable “test_se” has “1” value
  - Try using `set_constant $impl/test_se 0` to get a successful verification

Debugging Tools: Pattern Viewer

Failing Compare Point values annotated

Vector annotated in schematic (logic cone view)

Patterns - q_out_reg[0]/q_out_reg[0]

File Edit View Window Help

Compare point values for vector 1

R q_out_reg[0] (DFF, Loading 1)

I q_out_reg[0] (DFF, Loading 0)

SL 1 Const SD 1 CLK 0

D 1 CP 0 TI 0 TE 1

	Reference	Implementation	+/-
1		test_se	
2	pop	pop	
3	pop_clk	pop_clk	
4	..._logic/empty_flag/q_out_reg[0]	..._logic/empty_flag/q_out_reg[0]	
5	pop_logic/push_count_svd_reg[0]	pop_logic/push_count_svd_reg[0]	
6	pop_logic/push_count_svd_reg[1]	pop_logic/push_count_svd_reg[1]	
7	pop_logic/push_count_svd_reg[2]	pop_logic/push_count_svd_reg[2]	
8	pop_logic/push_count_svd_reg[3]	pop_logic/push_count_svd_reg[3]	

1	2	3	4	5	6	7	8	9	10	11	12	13
1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0		0	1
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0		0
0	1	0	1	0	0	1	1	0	0	1		0
1	1	1	0	1	0	0	1	1	0	1	0	
1	1	0	1	1	1	0	1	1	0	0	1	
1	0	0	1	1	0	0	1	0	0	0	0	0

Ready

Cone Schematics - q_out_reg[0]/q_out_reg[0]

Schematic Edit View Window Help

Color Mode Error Candidates

Reference>>> pop_logic/empty_flag/q_out_reg[0]

pop

pop_logic/pop

pop_logic/empty_flag/q_out_reg[0]

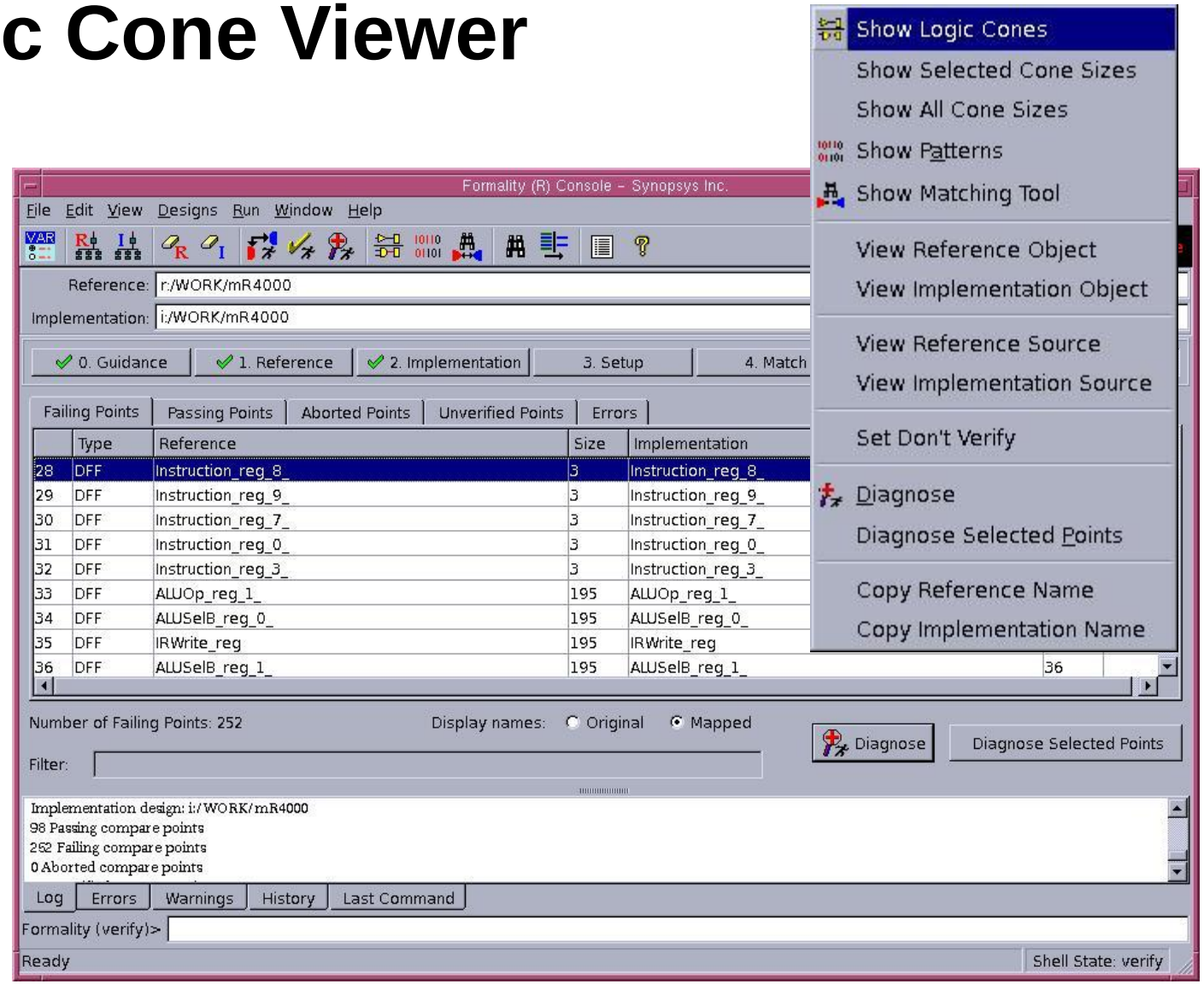
SL SD CLK

pop_logic/empty_flag/q_out[0]

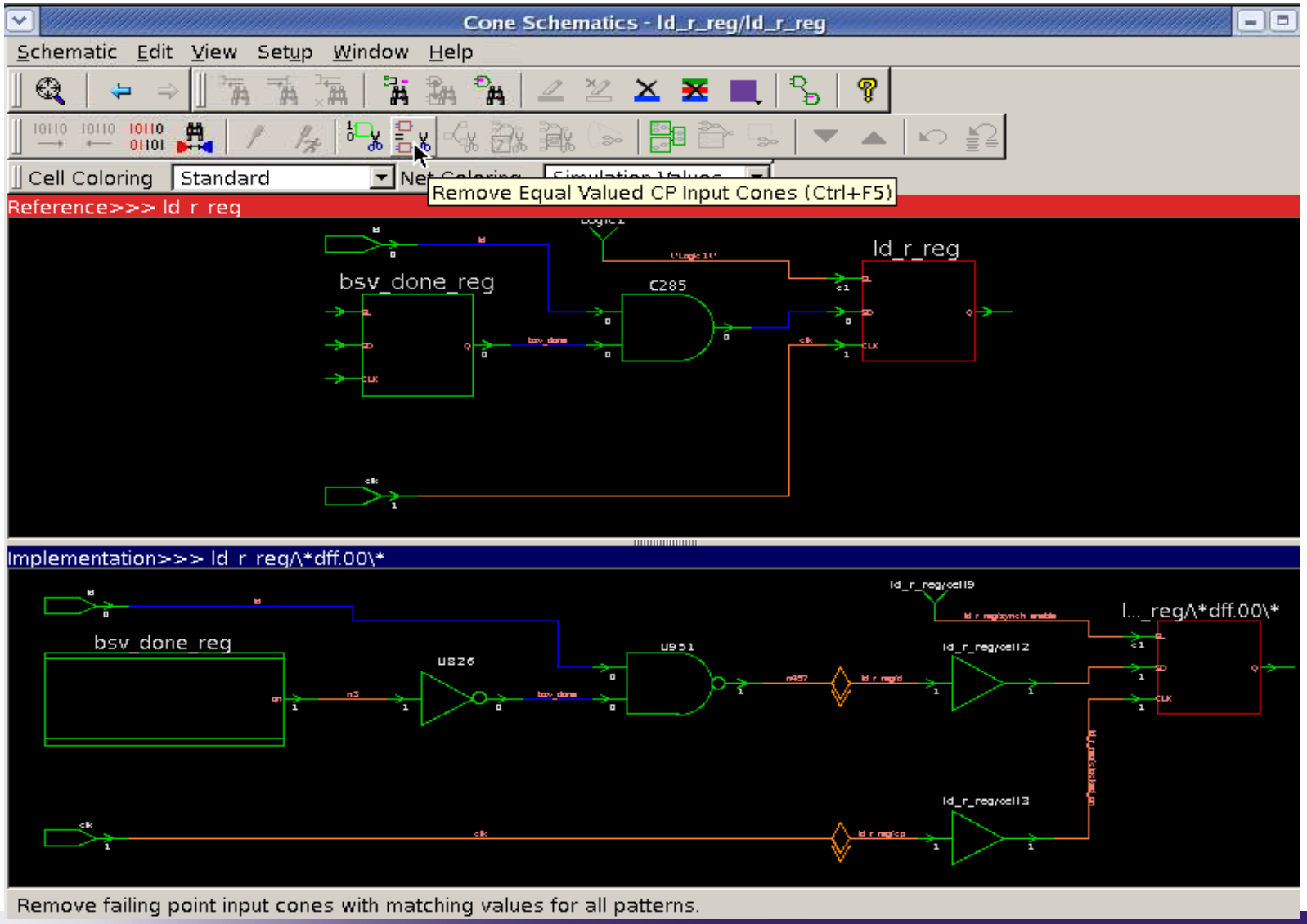
pop_logic/empty_flag/q_out_reg[0]
(SEQ)
Loading 1
SL = synchronous load (enable)
SD = synchronous data

Ready

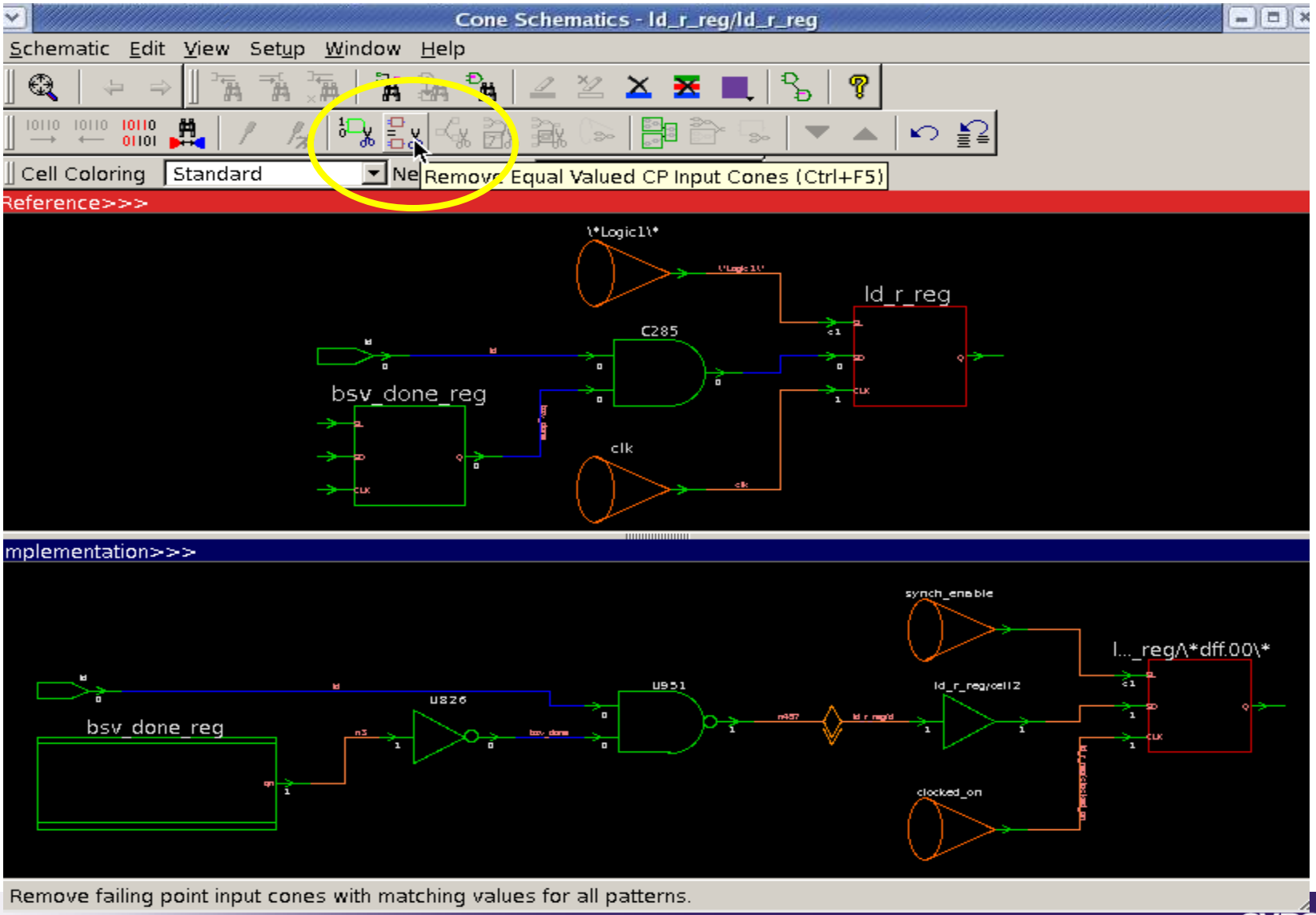
Debugging Tools: Logic Cone Viewer



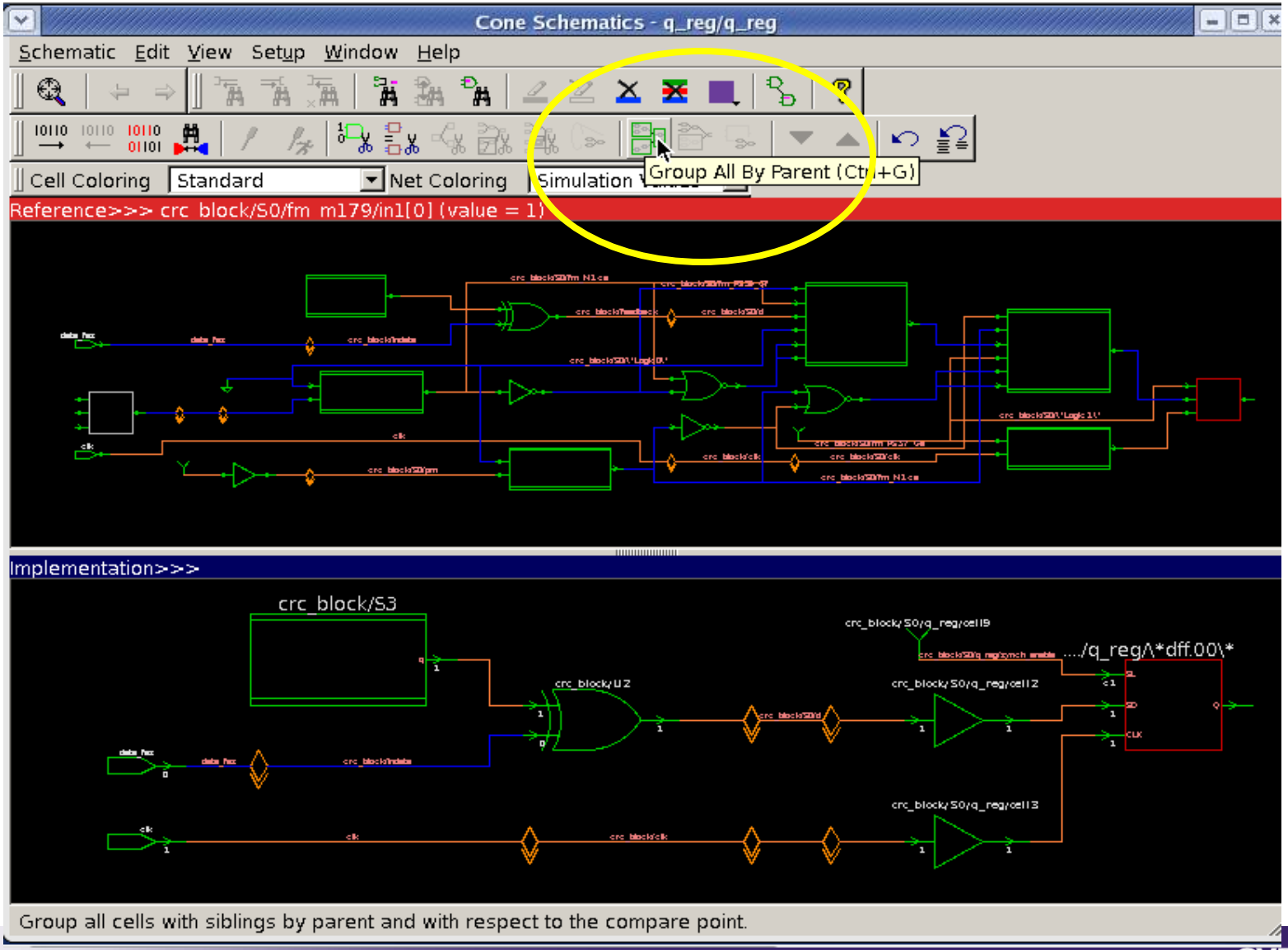
Debugging Tools: Logic Cone Viewer



Debugging Tools: Logic Cone Viewer

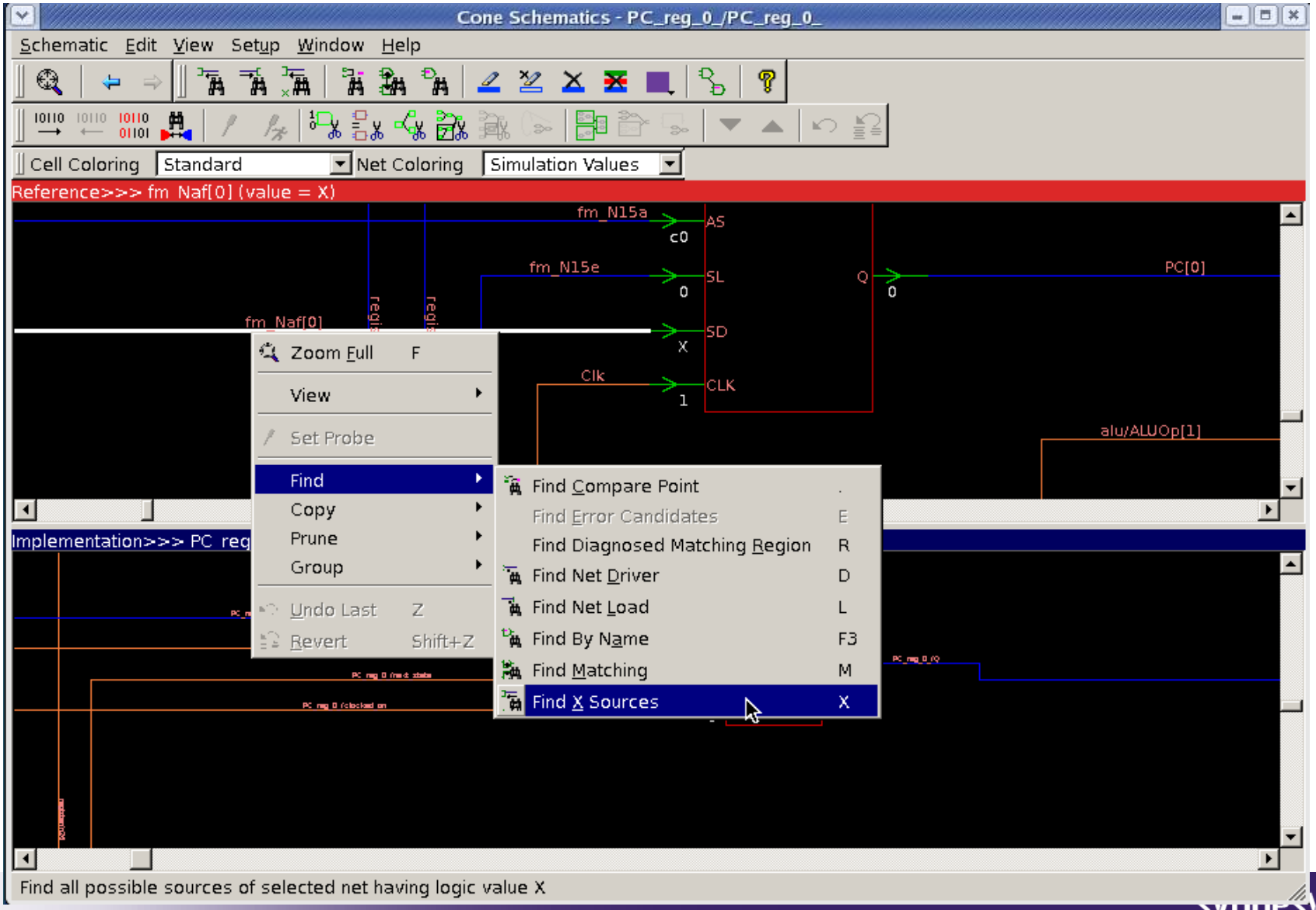


Debugging Tools: Logic Cone Viewer



Predictable Success

Debugging Tools: Logic Cone Viewer



Predictable Success

Debugging Tools: Logic Cone Viewer

Cone Schematics - q_reg/q_reg

Schematic Edit View Setup Window Help

Cell Coloring Standard Net Coloring Simulation Values

Reference>>> crc_block/S0/eq_17/in1[0] (value = 1)

Implementation>>> crc_block/S0/q_reg/CP (value = 1

Zoom Full F
View
Set Probe
Find
Copy
Prune
Group
Undo Last Z
Revert Shift+Z

Find Compare Point .
Find Error Candidates E
Find Diagnosed Matching Region R
Find Net Driver D
Find Net Load L
Find By Name F3
Find Matching M
Find X Sources X

Find driver of current net

Debugging Tools: Logic Cone Viewer

Cone Schematics - q_reg/q_reg

Schematic Edit View Setup Window Help

Cell Coloring Standard Net Coloring Simulation Values

Reference>>> crc_block/S0/*Logic1* (value = Const 1)

Zoom Full F
View
Set Probe
Find
Copy
Prune
Group
Undo Last Z
Revert Shift+Z

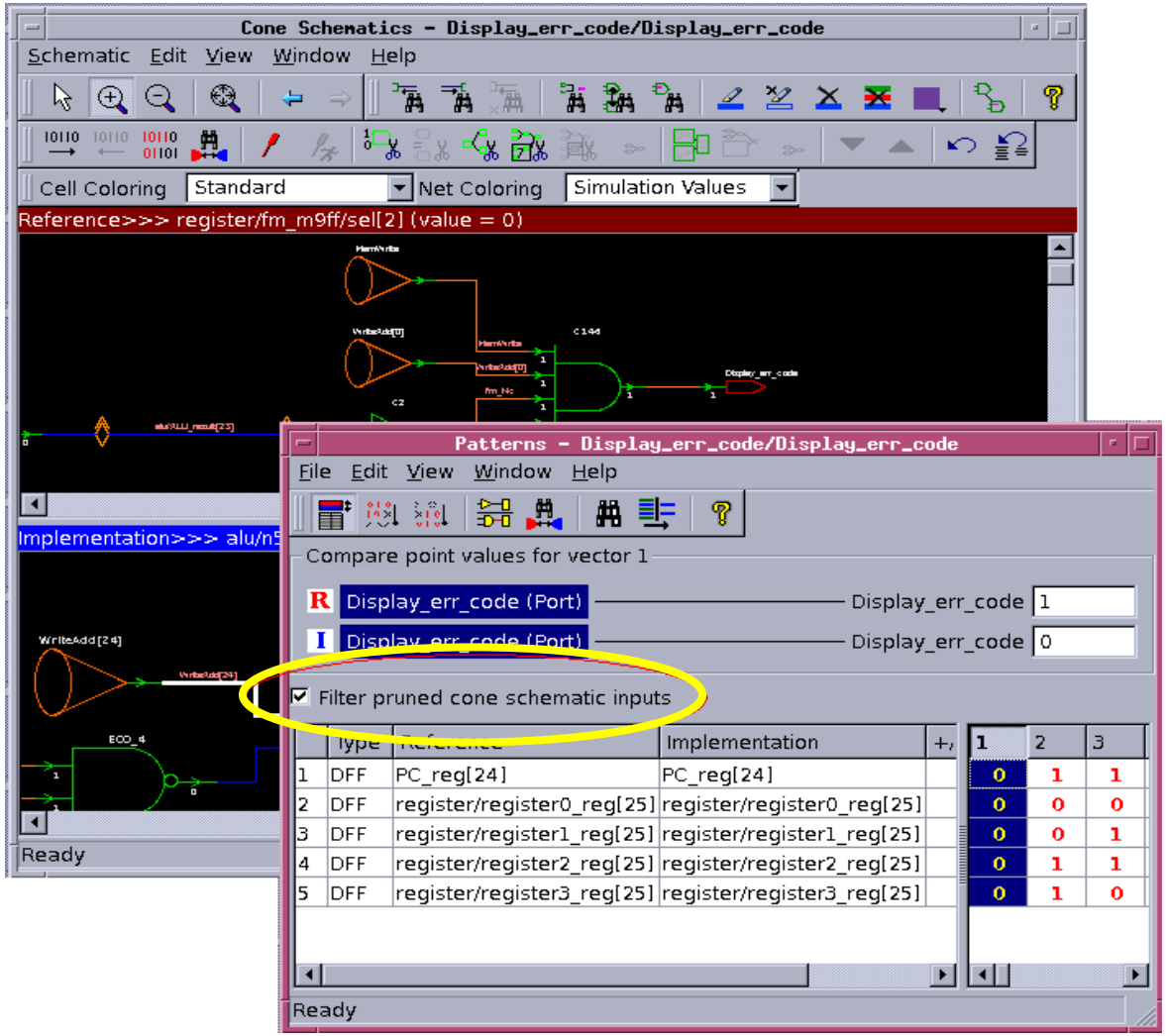
Remove Non-Controlling F5
Remove Equal Valued CP Inputs Ctrl+F5
Remove Subcone F6
Isolate Subcone F7
Isolate Error Candidates F8
Return Subcone Ctrl+F6

Implementation>>> crc_block/S0/q_reg/synch_enable (value = Const 1)

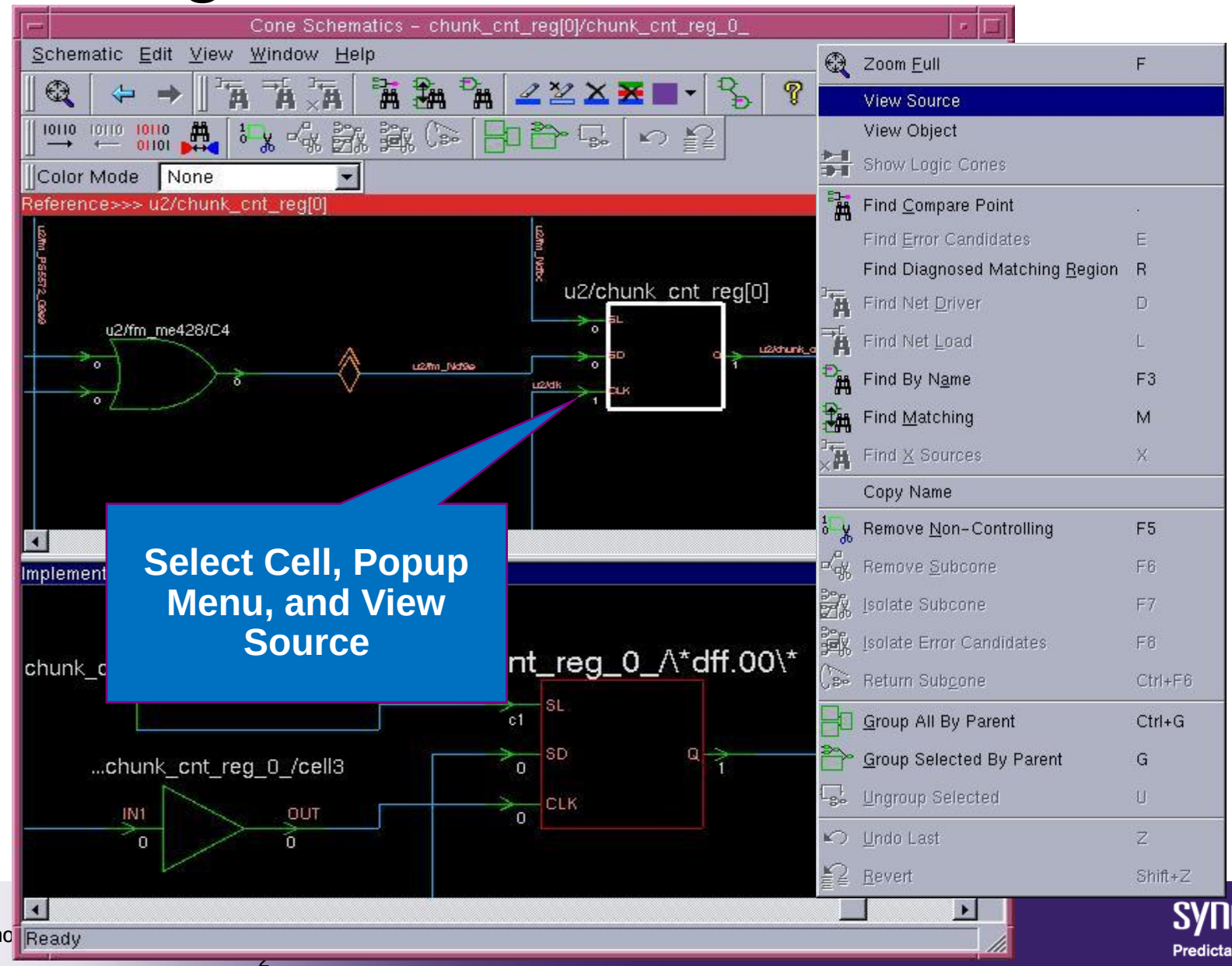
crc_block/S3/q_reg
crc_block/U2
crc_block/S0/q_reg/cell2
crc_block/S0/q_reg/cell3

Remove subcones of selected nets.

Pruning Correlation From Logic Cone to Pattern Viewer



Viewing RTL Source From Schematics



Select Cell, Popup Menu, and View Source

Source Code Browser

Reference Source Browser

File Edit View Window

/remote/fmcae4/users/ebhatch/dc_fest/FM_debugging_labs/lab5/rtl/...

```
197                 LD_DESC4        = 11'b000_1000_0000,
198                 LD_DESC5        = 11'b001_0000_0000,
199                 WB              = 11'b010_0000_0000,
200                 PAUSE           = 11'b100_0000_0000;
201
202 reg     [10:0] /* synopsys enum state */ state, next_state;
203 // synopsys state_vector state
204
205 reg     [31:0] mast0_adr, mast1_adr;
206
207 reg     [29:0] adr0_cnt, adr1_cnt;
208 wire    [29:0] adr0_cnt_next, adr1_cnt_next;
209 wire    [29:0] adr0_cnt_next1, adr1_cnt_next1;
210 reg            adr0_inc, adr1_inc;
211
212 reg     [8:0]  chunk_cnt;
213 reg            chunk_dec;
214
215 reg     [11:0] tsz_cnt;
216 reg            tsz_dec;
217
218 reg            de_txsz_we;
219 reg            de_csr_we;
220 reg            de_adr0_we;
221 reg            de_adr1_we;
```

Ready

Gate and line number highlighted

Queued Setup Commands

Cone Schematics - text_in_r_reg_112_/text_in_r_reg_112_

Schematic Edit View Setup Window Help

Set Constant 0
Set Constant 1
Set Black Box
Set Cutpoint
Set Clock
Remove Black Box
Remove Constant
Remove Cutpoint
Remove Clock

Cell Coloring Standa ng Simulation Values

Reference>>>

Command Queue

set constant -type port i:/WORK/aes_cipher_top/test_se 0

Clear Queue | Delete Selected | Execute Queue | Cancel

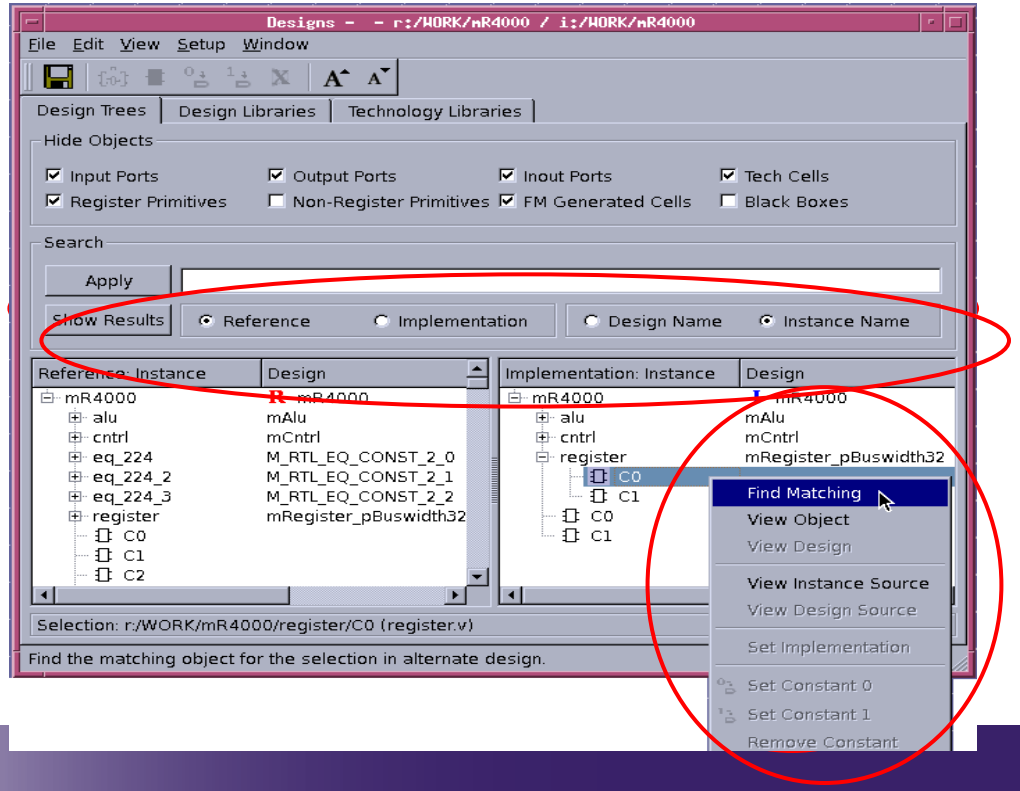
Implementation>>> test_se

test_se
1
test_se
CLK
1
U2681
CN
0
n2174
CLK
0
U2678
CN
1
n2171
bsv_done_reg

Ready

Debugging Tools: Dual Design Browser

- Reference and implementation browser now integrated together
- Search feature
- “Find Matching” feature
  - Select an object and find corresponding object in other container



Agenda

- Introduction to Equivalence Checking
- Using Formality
  - Flow Overview
  - Guidance
  - Read
  - Setup
  - Match
  - Verify
  - Debug
- Documentation and Help
- Lab Exercises

Formality Online Help

- Click on a hyperlink in the transcript, or use the **`man`** command

```
Status:   Elaborating design top   ...
Warning: Cannot link cell '/WORK/top/add1' to its reference design 'adder'. (FE-LINK-2)
Warning: Cannot link cell '/WORK/top/add2' to its reference design 'adder'. (FE-LINK-2)
Status:   Elaborating design multiplier   ...
Status:   Elaborating design multiplier2   ...
Status:   Elaborating design controller   ...
Error: Unresolved references detected during link. (FM-234)
Error: Failed to set top design to 'r:/WORK/top' (FM-156)
Formality (setup)> man sh_man_browser_mode
Info:  Opening help page 'file:///u/fmmgr/release/e_fm_main_1/e_fm_main_1_IMAGE/D20110224/
```

Log | Errors | Warnings | History | Last Command

Formality (setup)>

Ready

- Variable **`sh_man_browser_mode`** controls the GUI opening the browser for **`man`** command

Formality Online Help

Web browser window

Formality Online Help F-2011.09 – Mozilla Firefox

File Edit View Go Bookmarks Tools Help

file:///u/fmmgr/release/f_fm_main_1/f_fm_mai Go

WebMail Calendar Radio People Yellow Pages Download Customize...

TOC Quick search SYNOPSYS

- Formality Online Help
- Formality Concepts
- Formality Environment
- Formality Use Model
- Tcl Syntax as Applied to Formalit
- Commands
  - Commands by Group
  - Commands List
- Variables
- Error and Warning Messages

TOC
Search
Favorites

```
find_designs
    [object_list]
    [-passing]
    [-failing]
    [-aborted]
    [-not_verified]
```

Arguments

object_list

Lists one or more objectIDs in the current workspace. By default, this option is off.

-passing

Filters the design list to include designs that have passed the most recent verification. By default, this option is off.

-failing

Filters the design list to include designs that have failed the most recent verification. By default, this option is off.

Done

Help For Commands and Variables

- Three important commands for getting help:

  `printvar`

  - Displays the value of a Tcl variable
  - Accepts wildcards

  `help`

  - Displays brief description of a Formality command
  - Accepts wildcards

  `man`

  - Displays detailed information about a Formality command, Tcl variable, warning, or error message
  - Does not accept wildcards

Help Examples

```
fm_shell (setup)> help report_con*
report_constants                        # Report user specified constants
report_constraint                     # Reports on the defined constraints
```

```
fm_shell (setup)> read_verilog -r r400.v
Error: Can't open file r400.v (FM-016)
0

fm_shell (setup)> man FM-016
messages                          N.  Messages                  Command
   Reference

NAME
       FM-016 (error) Can't open file %s.

DESCRIPTION
       The specified file does not exist or cannot be created.

WHAT NEXT
       Verify  that  you specified the correct filename and that you
   have permission to open and create files.
```

Command Editing and Completion

- The Tcl shell supports powerful command editing and completion capabilities
  - Command completion with “Tab”
  - Use up and down arrow keys for moving through command stack

```
fm_shell (setup)> read_v
 read_verilog read_vhdl
fm_shell (setup)> read_verilog
```

Hit Tab key

Enter “e” and hit Tab key

Sources for Information

- SolvNet Website: https://solvnet.synopsys.com/
  - Formality release notes and user guides
  - Online training
  - Articles
  - Reference Methodology Guides
    - https://solvnet.synopsys.com/rmgen/
    - Design Compiler and Formality TCL scripts
    - IC Compiler and Formality TCL script
- Synopsys Website: http://www.synopsys.com/Tools/Verification/FormalEquivalence/Pages/Formality.aspx

Verilog RTL Interpretation

- Paper: "full_case parallel_case", the Evil Twins of Verilog Synthesis, by Cliff Cummings

http://www.sunburst-design.com/papers/CummingsSNUG1999Boston_FullParallelCase.pdf

  - Important information about using synthesis pragmas in your Verilog RTL for CASE statements

Command Summary – 1

fm_shell [-gui]	Launch Formality from Unix command line
formality	Launch Formality GUI from Unix command line
start_gui	Launch Formality GUI from Formality shell
printvar	Display current value of a Tcl variable
set	Set value of a Tcl variable
help [-verbose]	Get brief help on a Formality command
man	Get detailed help on a Formality command, variable, or error message.

Command Summary - 2

set_svf	Specify guidance file
read_verilog	Read Verilog source files
read_sverilog	Read SystemVerilog source files
read_vhdl	Read VHDL source files
read_db, read_ddc, read_mdb	Read Synopsys binary formats
set_top	Link the design
write_container	Save current container
read_container	Reads container file created by write_container

Command Summary - 3

match	Match compare points
verify	Check designs for equivalence
save_session	Save “snapshot” of current work
restore_session	Restore session file created by save_session
analyze_points	Provides debugging guidance for failing or hard compare points

Agenda

- Introduction to Equivalence Checking
- Using Formality
  - Flow Overview
  - Guidance
  - Read
  - Setup
  - Match
  - Verify
  - Debug
- Documentation and Help
- Lab Exercises

SYNOPSYS®

Predictable Success