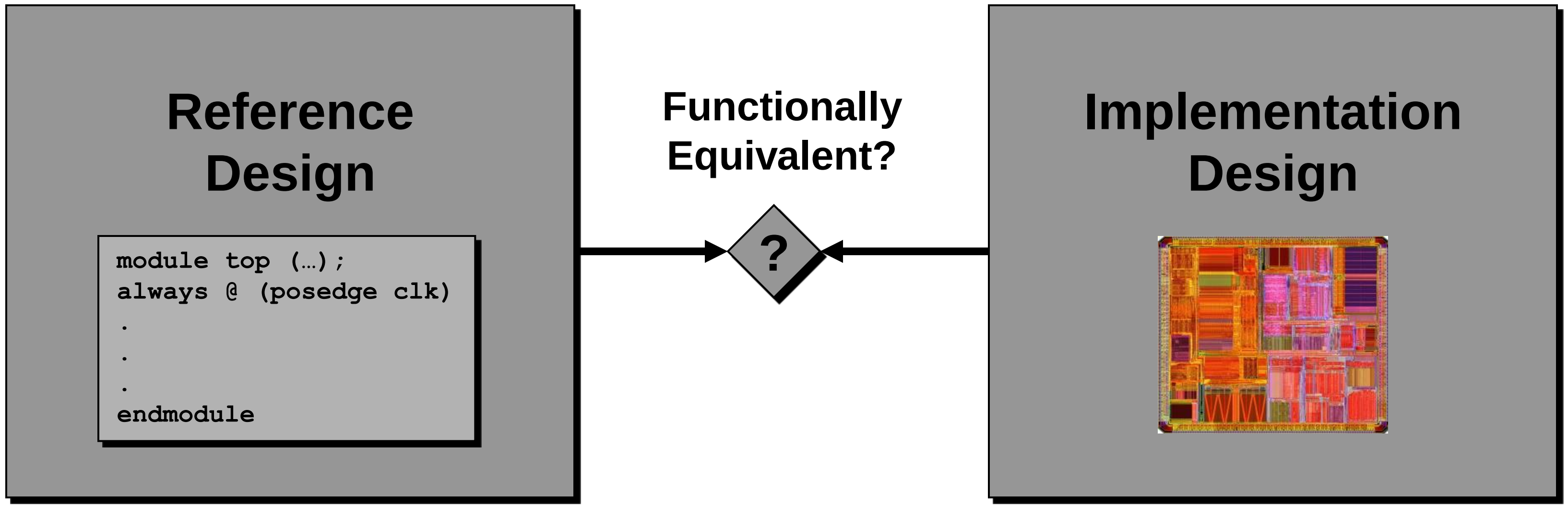
Introduction to Formality

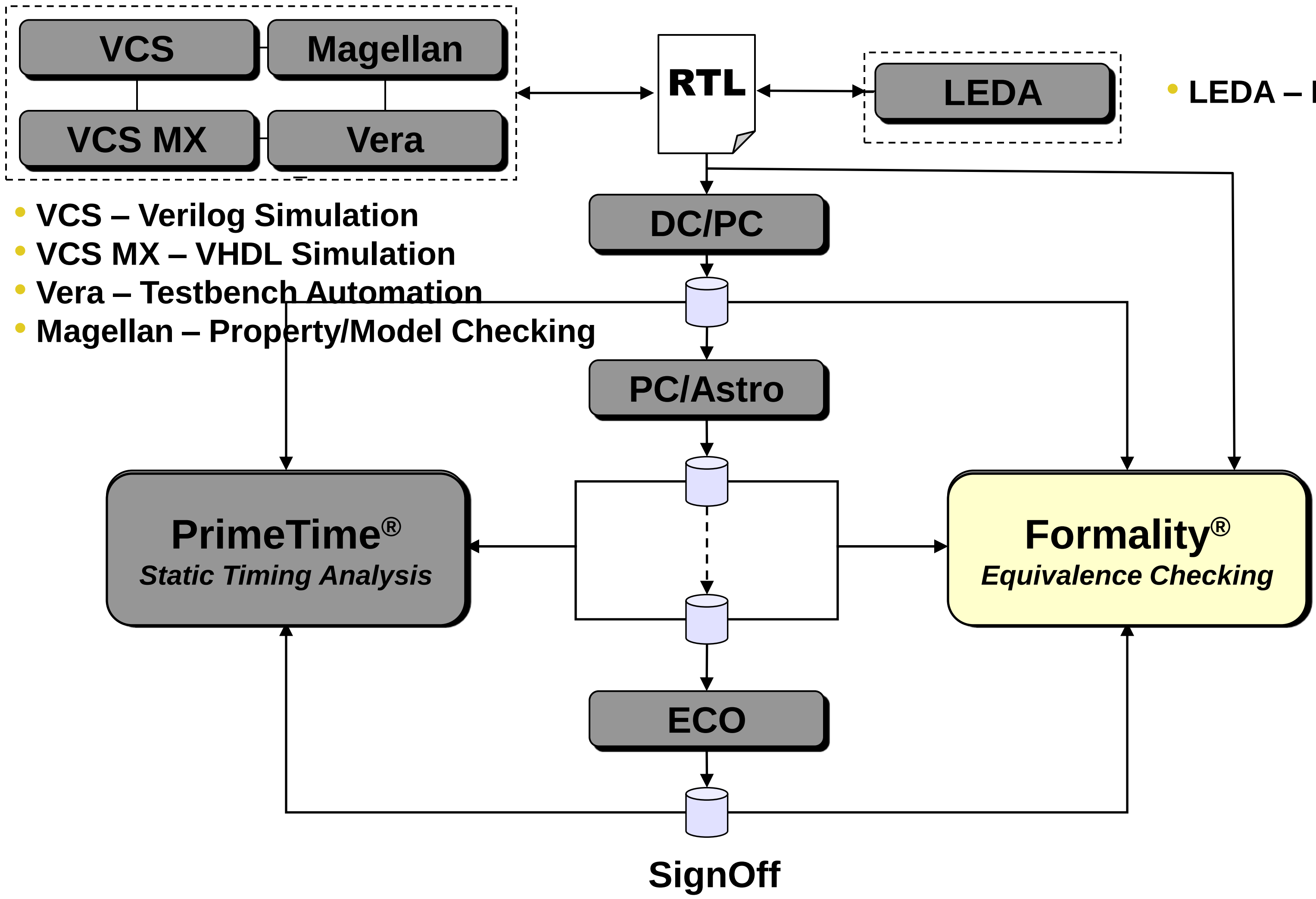
Formality is an Equivalence Checker

- Assumes that the reference design is good
- Determines if the implementation design is functionally equivalent
- Is mathematically exhaustive, no missing corner cases
- Does not require test vectors



```
module top (…);
always @ (posedge clk)
.
.
.
endmodule
```

Synopsys Smart Verification

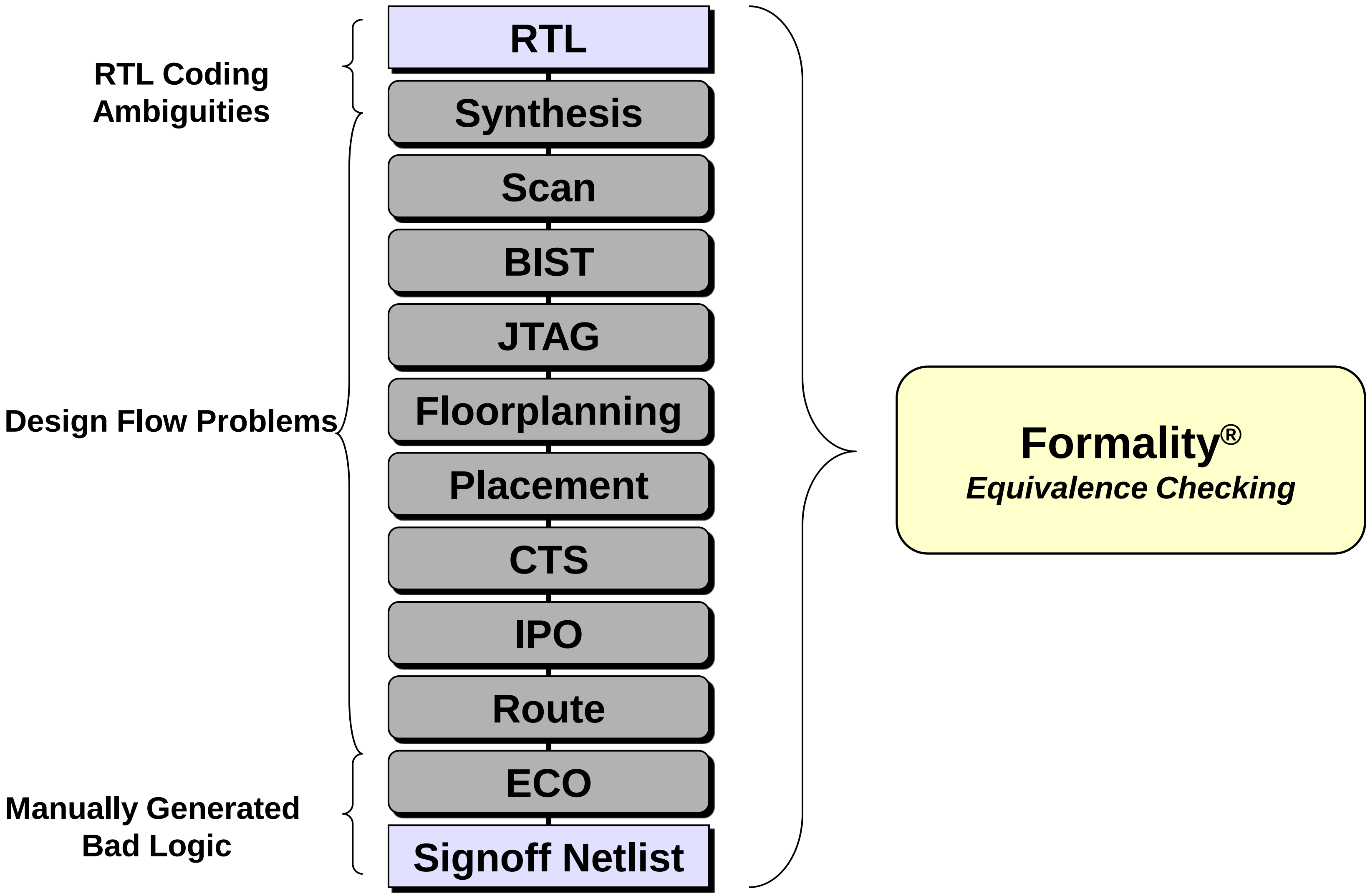


- VCS – Verilog Simulation
- VCS MX – VHDL Simulation
- Vera – Testbench Automation
- Magellan – Property/Model Checking

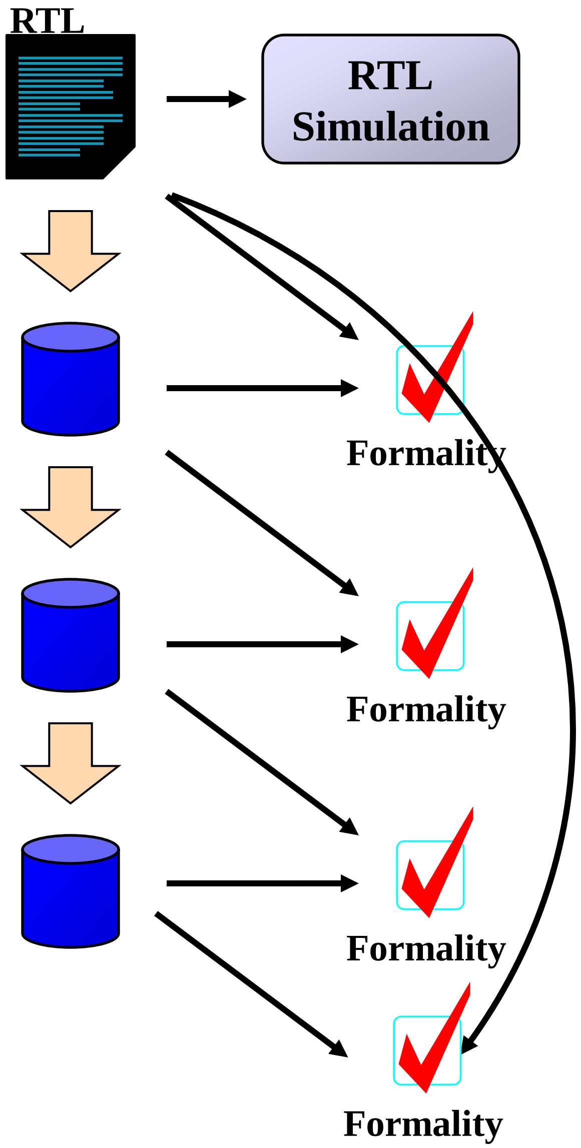
- LEDA – RTL Checker

Formality® Equivalence Checker

Verifies Functionality Throughout Entire Flow



Formality® in Your Verification Flow



RTL to gate (handoff)
- Check chip integration
- Final verification (setup)

Gate to gate
- Most common application
- Faster verification
- Faster debugging
- Isolates errors as they occur

RTL to gate (tape out)
- Final verification
- ECO verification

What Formality® Is Not

- **Formality is <u>not</u> a simulator (like VCS or VCS MX)**
  - It does not validate that your RTL functions correctly
- **Formality is <u>not</u> a testbench generator (like Vera)**
  - It does not generate testbenches
- **Formality is <u>not</u> a model checker (like Magellan)**
  - It does not validate protocols such as ‘a request is always acknowledgd within two clock cycles’
- **Formality is <u>not</u> a timing analyzer (like PrimeTime)**
  - Formality considers “static” responses
  - Formality does not check for timing hazards

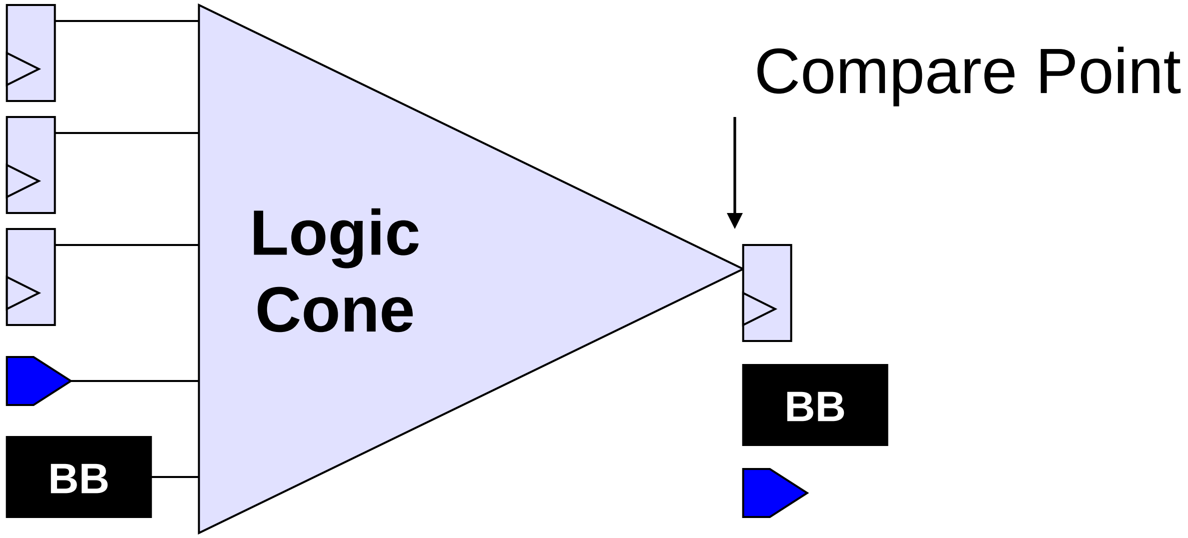
Two Key Concepts

A design contains Logic Cones and Compare Points

- **Compare Point**
  - A primary output of a circuit
  - A registers within a circuit
  - An input to a black boxes within circuit
- **Logic Cone**
  - A block of combinational logic which drives a compare point

Two Key Concepts - schematic

A design contains Logic Cones and Compare Points



<u>Inputs</u>

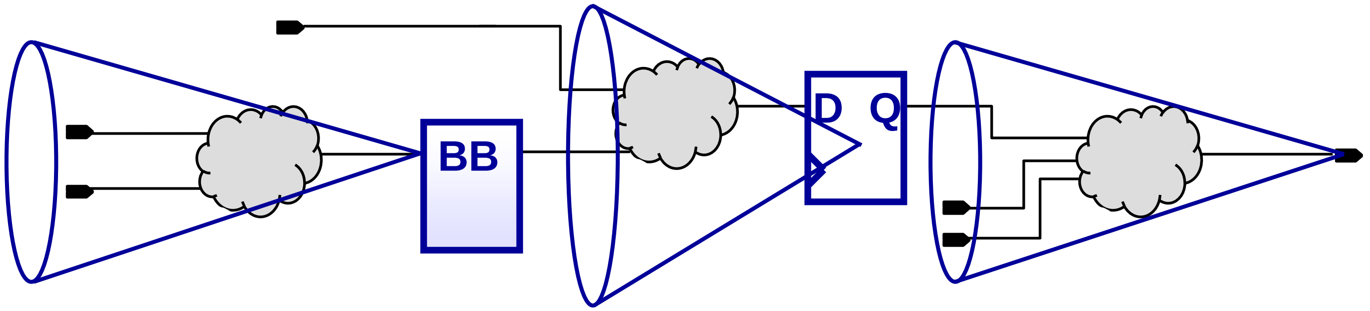
- **Outputs from Registers**
- **Primary Input Ports**
- **Outputs from Black Boxes**

<u>Compare Points</u>

- **Inputs to Registers**
- **Primary Output Ports**
- **Inputs to Black Boxes**

Breaking Design Into Cones and Points

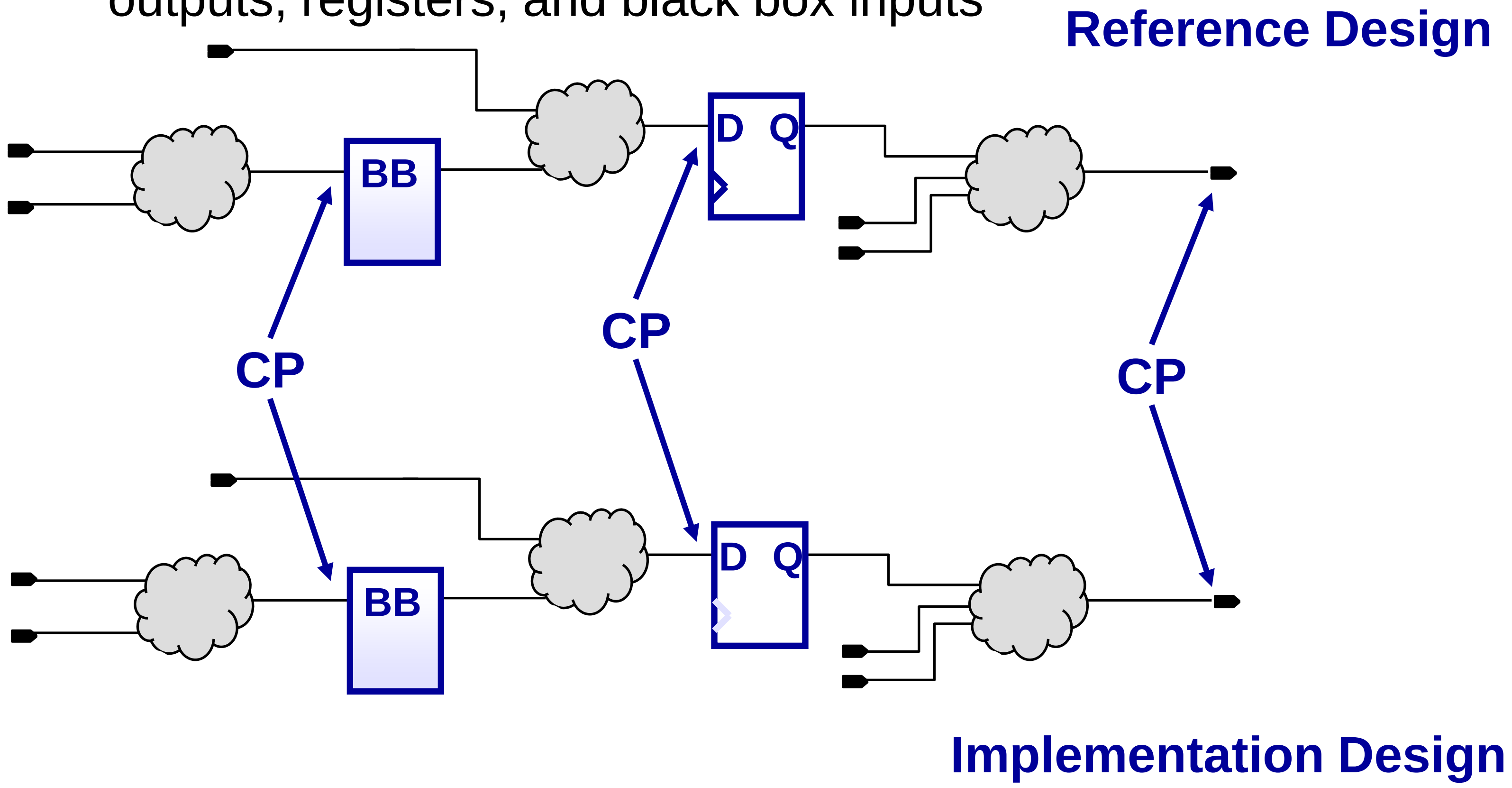
- **Breaks the two logic circuits up into logic cones:**
  - End points (compare points) are primary outputs, registers, and black box inputs



Determining Compare Points

Match Compare Points

- **Compare points are then aligned:**
  - This process is called “compare point matching”
  - End points of logic cones (compare points) are primary outputs, registers, and black box inputs



Verify Design

- **For each matched pair of compare points Formality tries to :**

  Either

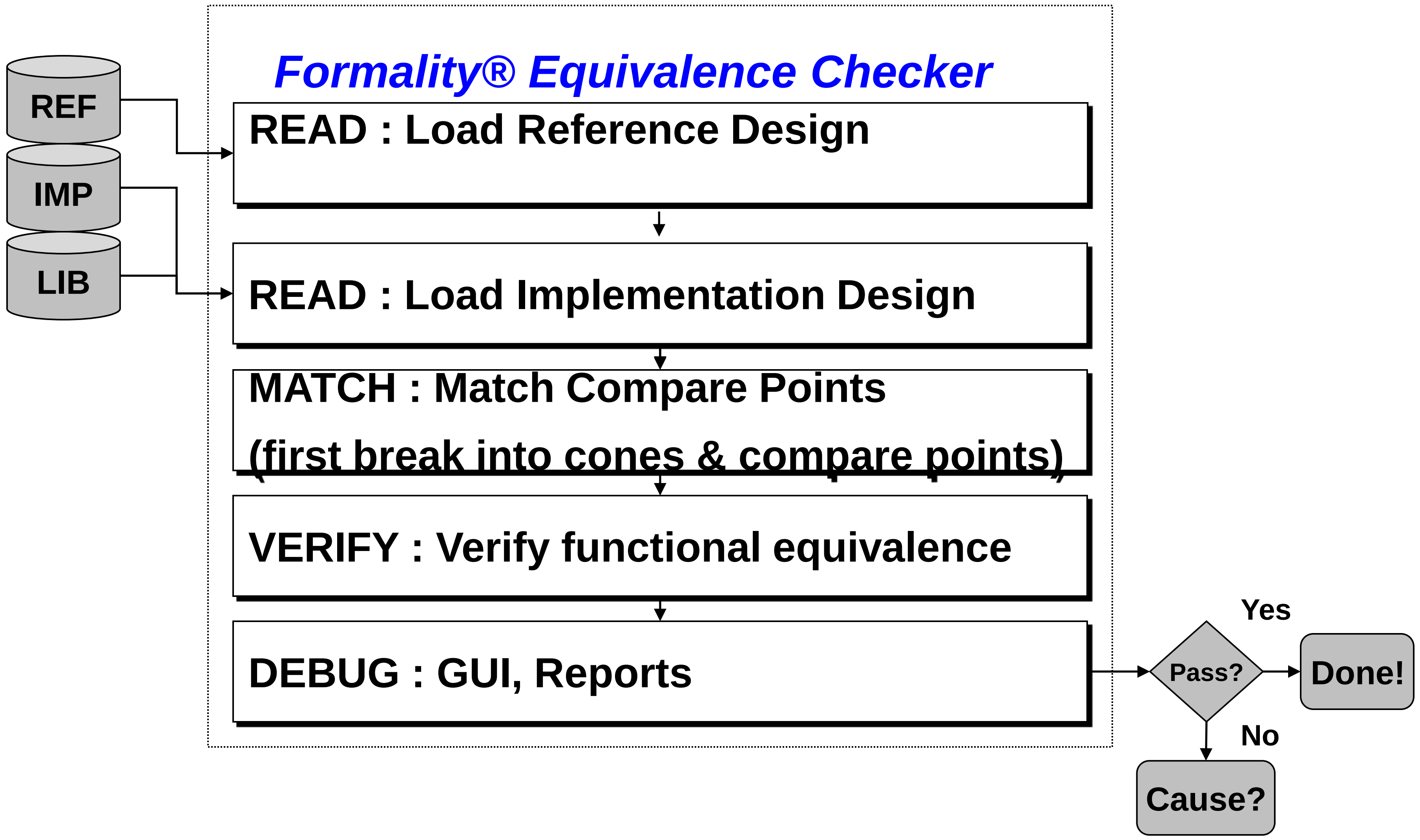
  - Confirms same response for all possible input combinations.
  - Marks point as "passed"

  Or

  - Finds a "counter example" that shows different response
  - Marks point as "failed"

- **This analysis is performed by solvers**
  - Formality deploys many different solvers
  - Each solver uses a different algorithm to prove equivalence / non-equivalence

Formality Flow Overview


Formality® Equivalence Checker
REF
IMP
LIB
READ : Load Reference Design
READ : Load Implementation Design
MATCH : Match Compare Points
(first break into cones & compare points)
VERIFY : Verify functional equivalence
DEBUG : GUI, Reports
Pass?
Yes
Done!
No
Cause?

Formality Terminology - 1

- **Reference Design**
  - The “golden” design under test
- **Implementation Design**
  - The modified design under test that is to be checked against the reference design
- **Containers**
  - Entity which stores all elements of a design to be tested
  - Default reference container is named “r”
  - Default implementation container is named “i”
  - You can create other containers and use as reference or implementation

Formality Terminology - 2

- **Matching**
  - Process of aligning compare points between two designs
- **Verification**
  - Process of proving / disproving that compare points are equivalent (have same functionality)
- **Debug**
  - Designer investigation of a failing verification to identify why compare point(s) have failed verification

Invoking Formality: fm_shell

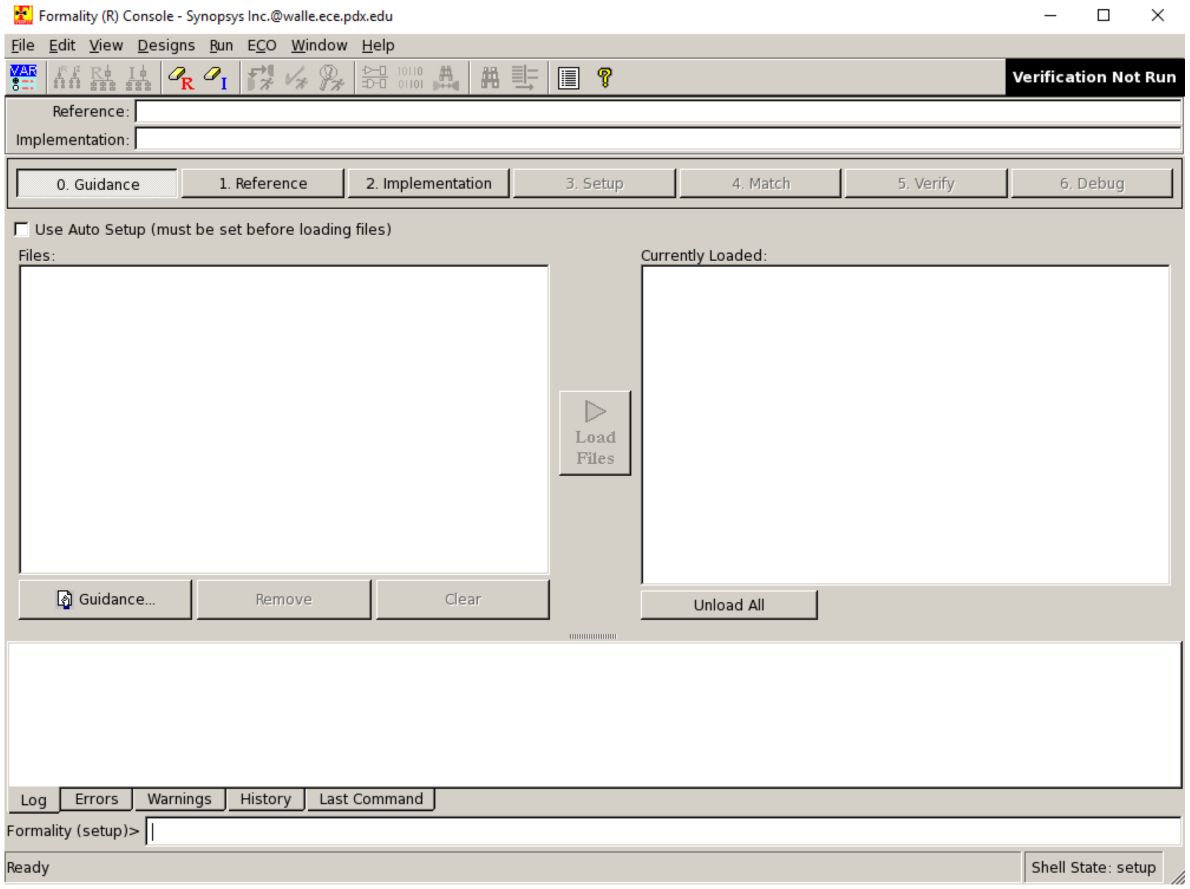
To invoke the Formality shell type fm_shell:

```
                         <3> % fm_shell
                          Formality (R)
                  Version I-2013.12-SP1 -- Jan 14, 2014
               Copyright (c) 1988-2018 by Synopsys, Inc.
                                ALL RIGHTS RESERVED

    This program is proprietary and confidential information of
                         Synopsys, Inc.
   and may be used and disclosed only as authorized in a license
                           agreement
               controlling such use and disclosure.

                ** Highlights of Formality 2013.12 **
        - ECO assistance and advanced debugging features(requires
                    Formality Ultra license)
                - Support for Synopsys Golden UPF Flow
       - Improved UPF debugging using GUI cell/net coloring modes
                 - Improved SVF processing performance
              - Support for System Verilog configurations

                        fm_shell (setup)>
```

Invoking Formality: GUI

To invoke the Formality GUI, type `formality`:

```
<2> % formality
<2> % fm_shell -gui
```



Launching GUI from within fm_shell

To launch the GUI type start_gui:

```
                        <3> % fm_shell

                         Formality (R)
                     Version I-2013.12-SP1 -- Jan 14, 2014
                  Copyright (c) 1988-2018 by Synopsys, Inc.
                                  ALL RIGHTS RESERVED

   This program is proprietary and confidential information of
                        Synopsys, Inc.
  and may be used and disclosed only as authorized in a license
                          agreement
              controlling such use and disclosure.

             ** Highlights of Formality 2013.12 **
      - ECO assistance and advanced debugging features(requires
                   Formality Ultra license)
             - Support for Synopsys Golden UPF Flow
     - Improved UPF debugging using GUI cell/net coloring modes
              - Improved SVF processing performance
            - Support for System Verilog configurations

                  fm_shell (setup)>start_gui
```

.synopsys_fm.setup File

- **When Formality is invoked, it reads a file called .synopsys_fm.setup, containing commands such as:**

```
set search_path “. ./lib ./netlists ./rtl”
history keep 200
alias h history
```

- **It is read from each of the following 3 locations:**

  1st. <formality_root>/admin/setup/.synopsys_fm.setup

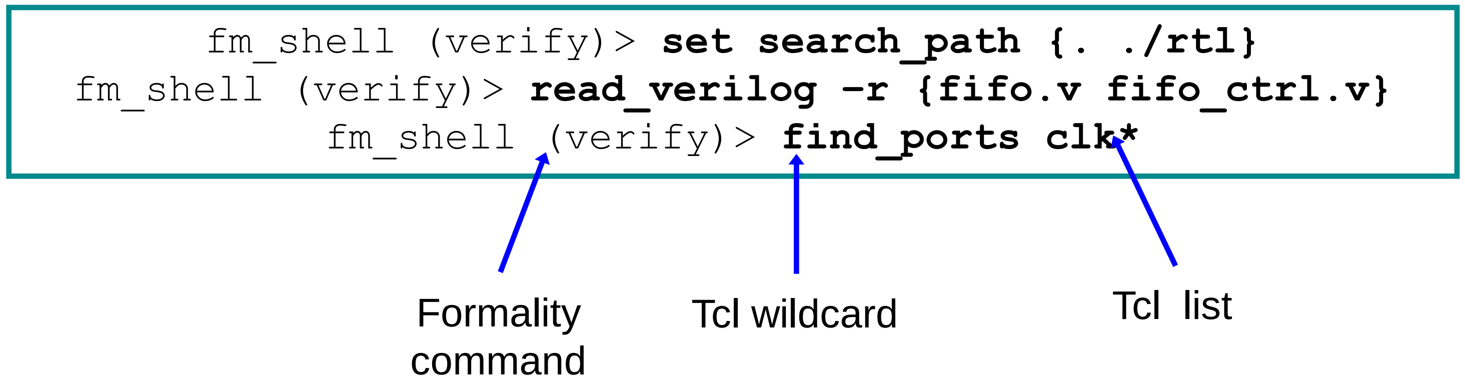
  2nd. Your home directory

  3rd. Current working directory

- **The effect of the setup files is cumulative:**
  - The setup files in all 3 locations are read into Formality, in the order shown above

Formality Command Syntax and Tcl

- **Formality command syntax is based on Tcl:**



```
fm_shell (verify)> set search_path {. ./rtl}
fm_shell (verify)> read_verilog -r {fifo.v fifo_ctrl.v}
fm_shell (verify)> find_ports clk*
```

  - Full support for Tcl constructs such as lists and procedures
  - Most Formality commands support Tcl wildcards

Getting Brief Help On Commands

- **From the Formality prompt:**
  - Type help command_name

```
fm_shell (setup)> help report_con*
report_constants      # Report user specified constants
report_constraint     # Reports on the defined constraints
report_constraint_type # Reports information about constraint types
report_containers     # Report containers in session

fm_shell (setup)>
```

  - help -verbose command_name

```
fm_shell (setup)> help -verbose set_user_match
set_user_match        # Sets the specified design objects as matched points
        [-type type]              (Type of objects:
                                   Values: port, net, cell, pin)
        objectID1                 (First design object of point pair)
        objectID2                 (Second design object of point pair)

fm_shell (setup)>
```

Command Editing and Completion

- **The TCL shell supports powerful command editing and completion capabilities**
  - set the variable sh_enable_line_editing to true in your .synopsys_fm.setup file

- **For example, command completion with “Tab”**

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

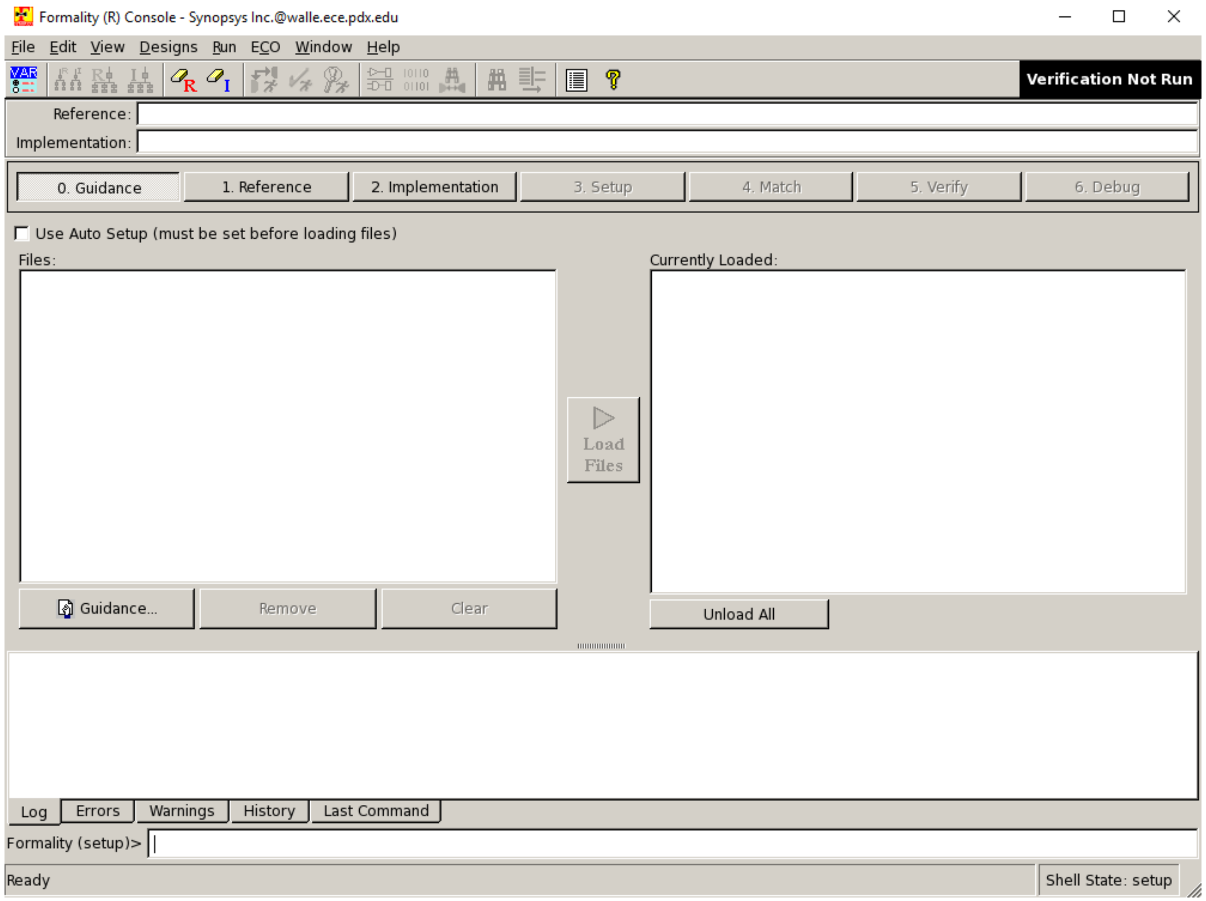
Hit Tab key

Enter “e” and hit Tab key

Running Formality Scripts

To execute a tcl script in formality:

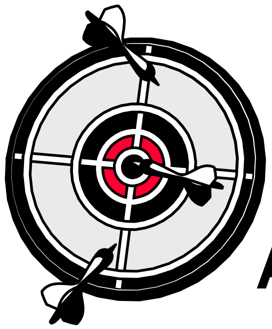
```
<2> % fm_shell -f script.tcl | tee script.log
fm_shell > source script.tcl
Formality GUI: File->Run Script
```



Command Summary

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.

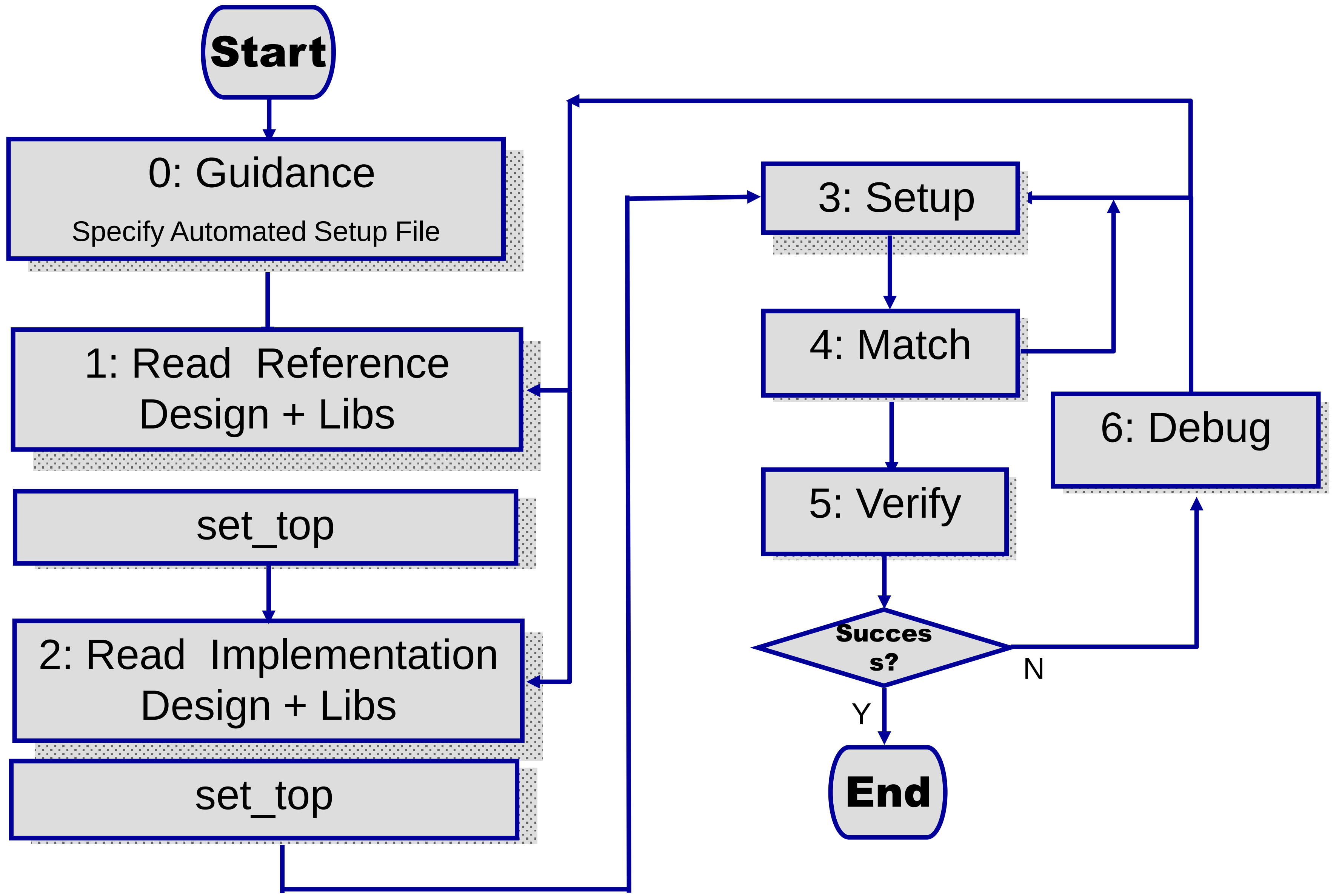
Unit Objectives



After completing this unit, you should be able to:

- Describe the seven-step Formality flow
- Read in libraries and designs
- Interactively compare two designs
- Create and run a script to compare two designs
- Save the results of a verification for later analysis

Formality Flow Overview


Start
0: Guidance
Specify Automated Setup File
1: Read Reference Design + Libs
set_top
2: Read Implementation Design + Libs
set_top
3: Setup
4: Match
5: Verify
6: Debug
Success?
N
Y
End

Basic Formality Script

```
#Step 0: Guidance
#default.svf not require

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

# Step 2: Read Implementation

read_verilog -i alu.fast.v
set_top alu

# Step 3: Setup
# No setup required here

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

Steps 1 and 2: Reading the Designs

- **Formality supports the following input formats:**

• Verilog (synthesizable subset)	- read_verilog
• Verilog (simulation libraries)	- read_verilog -vcs
• VHDL (synthesizable subset)	- read_vhdl
• EDIF	- read_edif
• Synopsys binary files	- read_db, read_ddc, read_mdb (*)

- **Read designs into containers**

• -r	# default reference container
• -i	# default implementation container

- **Link top level of design with set_top**
  - Must load required designs and libraries prior to executing set_top
  - Must execute set_top before loading subsequent containers

Setting Reference and Implementation Design

- **For the default “r” container the set_top command automatically sets the reference design**
  - The read-only variable $ref specifies the current reference design
- **For the default “i” container the set_top command automatically sets the implementation design**
  - The read only variable $impl specifies the current implementation design
- **Format of $ref and $impl is:**
  - Container:/Library/Design
  - Examples:
    - r:/DESIGN/chip
    - i:/WORK/alu_0

Step 1: Read the Reference - Example

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

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

Step 2: Read the Implementation Design - Example

```
fm_shell (setup)> read_verilog -i alu.fast.v
fm_shell (setup)> set_top alu_0
```

- **read_verilog loads the implementation design**
  - The "-i" signifies the (default) implementation container
- **read_db loads the technology library class.db**
  - Since "-i" specified this library is visible only in the container "i"
- **set_top links top-level module "alu_0"**
  - Script reads both design and technology library before set_top
  - Here set_top uses the current container ("i")
  - set_top sets the variable $impl to i:/WORK/alu_0

Step 3: Setup

- **In the setup stage you can enter additional information to guide matching and/or verification.**
- **Setup is most likely required when:**
  - You are not using SVF files
  - There are complex design transformations
  - You are working outside the Synopsys flow

Step 3: Setup - Example

```
fm_shell (setup)> set_constant $impl/test_se 0
```

- **This script disables scan logic in the implementation design**
  - This is a typical setup step
- **The “Basic Script” slide 3-4 did not contain any setup commands**

Step 4: Match

- **Before verification you must match compare points by either:**
  - Explicitly running Formality's matching algorithms
  - Allowing the "verify" command to run matching for you
- **Formality uses powerful algorithms to match points**
  - Can match most points automatically
  - SVF files will help – particularly if your DC script changes object names
- **Manual intervention is possible through**
  - Rules created by user
  - Explicit matches by user
- **Module 5 will discuss matching in detail**

Step 4: Match - Example

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

- **This script uses the explicit match command**
  - An optional command.
  - The verify command will run matching if you do not execute the match command explicitly
- **Recommendation:**
  - In interactive work use the explicit match command
    - Gives you important feedback
  - Omit the match command from scripts
    - Reduces runtime

Step 5: Verify

- **Run Formality's verification algorithms**
  - By default all points are verified
- **Four possible results:**
  - Succeeded: implementation is equivalent to the reference
  - Failed: implementation is not equivalent to the reference
    - Logic difference or setup problem
  - Inconclusive: no points failed, but analysis incomplete
    - Analysis incomplete due to timeout or to complexity
    - Module 9 will discuss techniques for very complex designs
  - Not run: a problem earlier in the flow prevented verification from running at all.
- **Module 6 will discuss Verification in detail**

Step 5: Verify - Example

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

- **By default the verify command verifies all points**
- **However there are options allowing you to:**
  - Run verification on a single point
  - Exclude points from verification

Reference

- Formality® User Guide, Synopsys , 2013