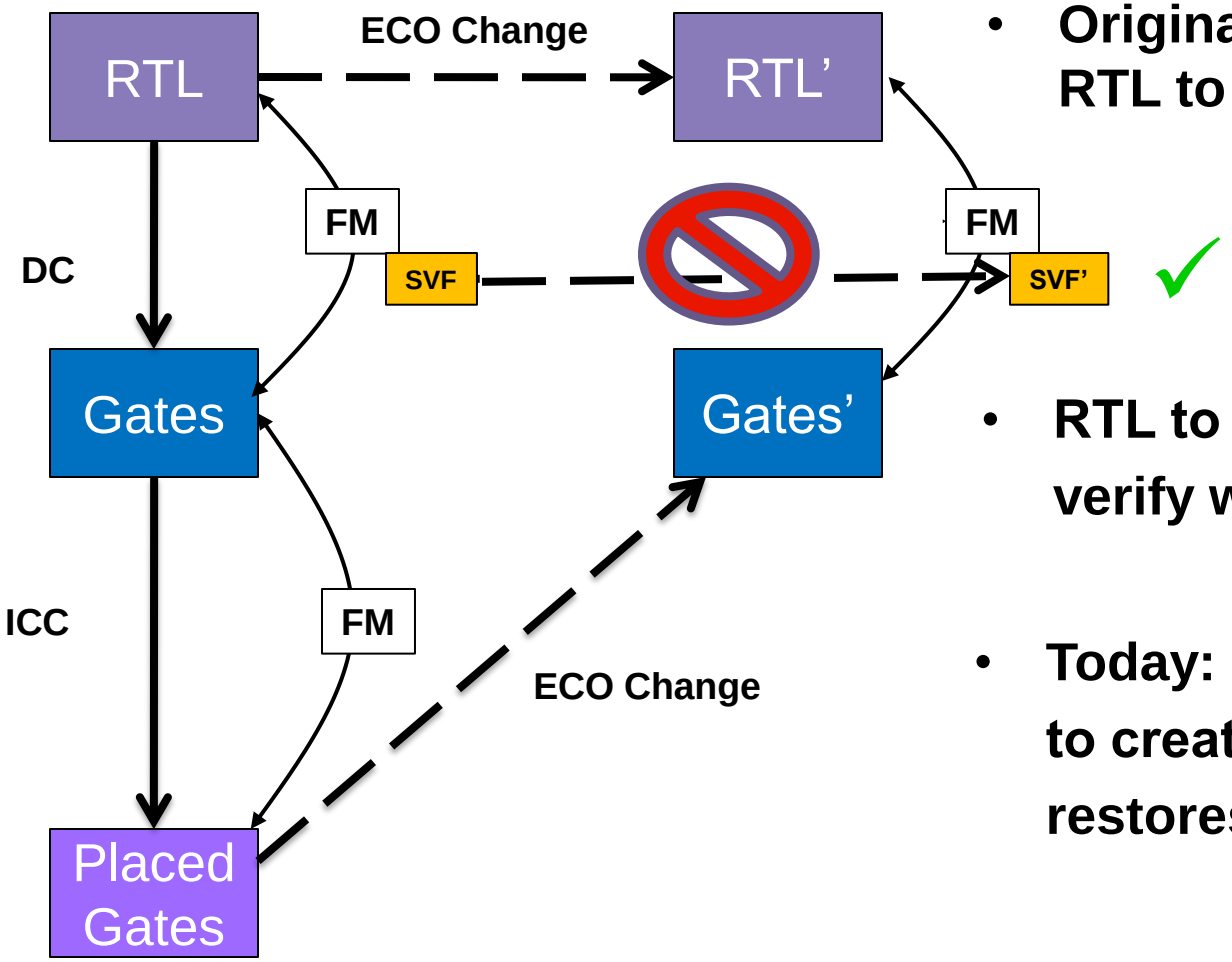
Reusing the Automated Setup File (SVF)

VERIFYING DESIGNS FOR ECO

Verifying Designs for ECO



- **Original verifies: RTL to gates**
- **RTL to Gates' will *not* verify with SVF**
- **Today: Hand-modify SVF to create SVF'; restores verifiability**

Verifying Designs for ECO

Original RTL

```
1: module test ( A,B,Y );
2: input [7:0] A, B ;
3: output [15:0] Y ;
4: reg[15:0] Y ;
5: always @( A or B ) begin
6: Y = A * B ;
7: end
8: endmodule
```

ECO RTL

```
1: module test ( A,B,Y );
2: input [7:0] A, B ;
3: output [15:0] Y ;
4: reg[15:0] Y ;
5: // insert comment
6: always @( A or B ) begin
7: Y = A * B ;
8: end
9: endmodule
```

Original SVF

```
guide_change_names
design { test }
{ { cell mult_6 mult_x_5_1 } }

guide_multiplier
-design { test }
-instance { mult_x_5_1 }
-arch { apparch}
-body { test_DW_mult_uns_J1_0 }
```

Using original SVF with ECO RTL will cause SVF rejection of mult_6

Verifying Designs for ECO

- Automate the line changes or operator/positional changes that are made to the SVF to create SVF'.

Original RTL file orig/foo.v

```
1 module bar(A,B,Z1,Z2);
2       input [1:0] A,B;
3       output [2:0] Z1,Z2;
4       assign Z1 = A + B;
5       assign Z2 = A - B;
6       // Blank
7       // Blank
8 endmodule
```

ECO modified RTL file ECO/foo.v

```
    // This is the ECO version
module bar(A,B,Z1,Z2);
      input [1:0] A,B;
      output [2:0] Z1,Z2;
      assign Z1 = A + B + 1;
      assign Z2 = A - B;
endmodule
```

Verifying Designs for ECO

- To generate SVF' to account for ECO changes:

1. Generate ECO change information
   Line number changes in SVF format

2. Generate operator mapping information
   Datapath operator changes in SVF format

3. Verify ECO RTL to ECO implementation netlist
   Using original SVF and SVF' generated by Steps #1 and #2 above.

Verifying Designs for ECO

Step 1- ECO change information

- Generates SVF highlighting RTL source changes between original RTL sources and ECO RTL sources.

- Script resides in the Formality distribution directory

  `$SYNOPSYS/<PLATFORM>/fm/bin/fm_eco_to_svf`

- Script automatically produces ECO change SVF syntax

Verifying Designs for ECO

Step 1- ECO change information

- Usage

```
fm_eco_to_svf {orig RTL} {ECO RTL} > SVFfile
```

- For a single RTL file

```
fm_eco_to_svf ./original/foo.v ./eco/foo.v > eco_change.svf
fm_eco_to_svf ./original/bar.v  ./eco/bar.v  >> eco_change.svf
```

- For all the RTL files in a directory

```
fm_eco_to_svf ./original ./eco >   eco_change.svf
```

- Creates eco_change.svf

Verifying Designs for ECO

Step 1- ECO change information

- New SVF command (guide_eco_change)

```
guide_eco_change -file { foo.v } -type { insert } -original { 4 } -eco { 4 5 }
guide_eco_change -file { foo.v } -type { delete } -original { 7 } -eco { 8 }
guide_eco_change -file { foo.v } -type { replace } -original { 12 14 } -eco { 13
    14 }
```

`-file` : Name of source file
`-type` : Inserting lines, deleting lines, or replacing lines.
`-original` : {line region}
`-eco` : {line region}

- Line region
  - Single lines are represented by a single line number
  - Multiple lines are represented by two line numbers

Verifying Designs for ECO

- Mapping the datapath operator

Original RTL file orig/foo.v

```
1 module bar(A,B,Z1,Z2);
2   input [1:0] A,B;
3   output [2:0] Z1,Z2;
4   assign Z1 = A + B;
5   assign Z2 = A - B;
6 // Blank
7 // Blank
8 endmodule
```

ECO modified RTL file ECO/foo.v

```
// This is the ECO version
module bar(A,B,Z1,Z2);
  input [1:0] A,B;
  output [2:0] Z1,Z2;
  assign Z1 = A + B + 1;
  assign Z2 = A - B;
endmodule
```

Verifying Designs for ECO

Step 2: ECO Operator mapping information

- Generate SVF' for datapath operator changes
- Uses ECO change SVF information from Step 1
- New Formality command - `generate_eco_map_file`
  - Generate the new SVF mapping of datapath blocks that have been changed by ECO
  - Usage:

```
generate_eco_map_file [-replace] SVFfile
```

- May require user intervention
- Generates a file named eco_map.svf

Verifying Designs for ECO

Step 2: ECO Operator mapping information

Example FM script

```
# Read original and line change SVF files
   set_svf original.svf eco_change.svf

# Read original RTL source or container read_container -r
   design_original_RTL.fsc

# Read ECO RTL source or container read_container -i
   design_eco_RTL.fsc

# Create operator mapping file
   generate_eco_map_file -replace eco_map.svf
```

Verifying Designs for ECO

Step 2: ECO Operator mapping information

```
###
### ECO REGION 1 ###
###

# Cells in original design 'bar':
#   add_4

# Cells in ECO design 'bar':
#   add_5
#   add_5_2

### Cell add_4 (original name)
guide_eco_map -from {add_4} -to {add_5} -design {bar}
```

Verifying Designs for ECO

Step 2: ECO Operator mapping information

- Sometimes user intervention is required for operator mapping.
- RTL changes may be too complicated to be definite.
- Output of the `generate_eco_map_file` command provides the possible `guide_eco_map` commands (commented).
- You must inspect and chose which ones to keep.

Verifying Designs for ECO

Step 2: ECO Operator mapping information

```
guide
### IMPORTANT: Inspect and change the following guide_eco_map commands.
### Each "from" operator can be matched to at most one "to" operator,
### and vice versa.
### Uncomment the correct matches.
### INSPECT AND CHANGE THESE LINES
# guide_eco_map -design { foo } -from { add_5 } -to { add_6 }
# guide_eco_map -design { foo } -from { add_5 } -to { add_6_2 }
# guide_eco_map -design { foo } -from { add_5 } -to { add_6_3 }
# guide_eco_map -design { foo } -from { add_5_2 } -to { add_6 }
# guide_eco_map -design { foo } -from { add_5_2 } -to { add_6_2 }
# guide_eco_map -design { foo } -from { add_5_2 } -to { add_6_3 }
setup
```

Verifying Designs for ECO

Step 3: ECO RTL to ECO Gate verification

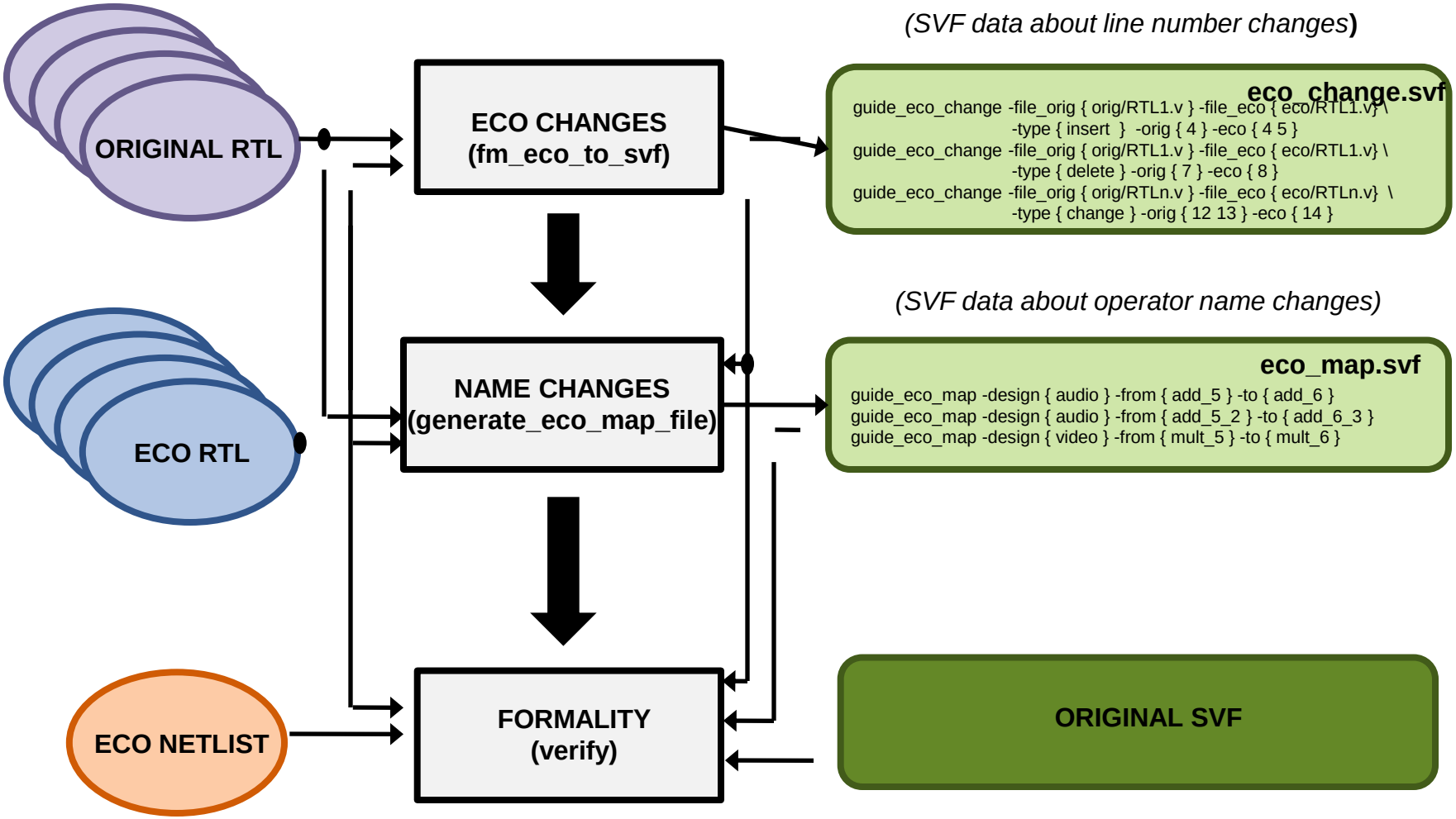
- Verify ECO RTL to ECO gate using original SVF and SVF' create from Steps 1 and 2

```
set_svf eco_change.svf eco_map.svf original.svf
read ECO RTL source files
read ECO netlist
verify
```

Verifying Designs for ECO

- Summary
  - Greatly reduced SVF rejection rate vs. original SVF alone
  - Faster verification times for ECO RTL to gate verifications

Verifying Designs for ECO



ENHANCEMENTS TO THE AUTO-SETUP MODE

Enhancements to the Auto-Setup Mode

- Set the don’t verify attribute on macro test inputs, by default
  - `guide_dont_verify_scan_input`
- Selective control of auto-setup
  - Control the effects of the auto-setup mode

Enhancements to the Auto-Setup Mode

- Macros might have test pins that are connected by `insert_dft` command
- Macros might be treated as black boxes by Formality
  - Might cause verification failures unless they are disabled
  - The new `guide_dont_verify_scan_input` SVF guide command is inserted by the `insert_dft` command
  - Tells Formality not to verify the scan input pin or port

Enhancements to the Auto-Setup Mode

- New command in SVF

```
guide_dont_verify_scan_input
   -ports { design/port … }
   -pins { <instance_name_of_pin> … }
```

- If `synopsys_auto_setup` is set to `true`, Formality issues a `set_dont_verify_point` for each referenced black box pin or port.
- The tool reports a warning message about this action in the auto-setup summary at the end of verification.

Enhancements to the Auto-Setup Mode

- Selective control of auto-setup
  - Overrides the effects of `synopsys_auto_setup`
  - Controlled by the new `synopsys_auto_setup_filter` Formality variable
  - Any variable or category contained in this new list will not have its default value(s) changed during auto-setup
  - Define the filter list before setting the `synopsys_auto_setup` variable

Enhancements to the Auto-Setup Mode

- The following variable list can be controlled
  - `hdlin_error_on_mismatch_message`
  - `hdlin_ignore_embedded_configuration`
  - `hdlin_ignore_full_case`
  - `hdlin_ignore_parallel_case`
  - `signature_analysis_allow_subset_match`
  - `svf_ignore_unqualified_fsm_information`
  - `hdlin_dyn_array_bnd_check`
  - `verification_set_undriven_signals`
  - `verification_verify_directly_undriven_output`
  - `scan_input` (auto-setup ignores all SVF `guide_scan_input` and `guide_dont_verify_scan_input` commands)
  - `clock_gating` (auto-setup ignores the following SVF `guide_environment` commands: `clock_gating_latch_and`, `clock_gating_latch_or`, `clock_gating_and`, `clock_gating_or`)

Enhancements to the Auto-Setup Mode

- Example 1: Allow all default options of auto setup mode except for scan insertion setup

```
set synopsys_auto_setup_filter {scan_input}
set synopsys_auto_setup true
```

- Example 2: Allow all default options of auto setup mode except for synthesis interpretation of full_case and parallel_case directives

```
set synopsys_auto_setup_filter { hdlin_ignore_full_case
  hdlin_ignore_parallel_case }
set synopsys_auto_setup true
```

UPDATES TO FORMALITY COMMANDS AND VARIABLES

Updates to Formality Commands and Variables

- Name Matching of Black Box Pins for Functionally-matched Black Boxes

In earlier versions:

- Black Box pins are also matched using functional matching (even if there was a name match)
- Could cause bad matches

Starting with version F-2011.09

- Matching of black box pins is now both name based and functionality based
- Improves the quality of matching

Updates to Formality Commands and Variables

- Allow `set_user_match` when `verification_blackbox_match_mode` is set to `identity`

In earlier versions

- Formality matches black boxes that have the same design name and come from the same library
- `set_user_match` could not be used to match black boxes with non-identical names

Starting with version F-2011.09

- `set_user_match` allowed to match black boxes with non-identical names

NEW FORMALITY COMMANDS AND VARIABLES

New Formality Commands and Variables

- Tech cell “interface_only”
  - Separate “interface_only” variable exclusively for tech cells
    - Avoids override of `hdlin_interface_only`
    - Restricted to tech cells
  - New command
    `set library_interface_only “cell_names”`
  - Independent of `hdlin_interface_only`

Updates to Formality Commands and Variables

  - Can be used for tech cells that always need to be black boxed
  - Example
    - An internal CAD group that supplies libraries can define the set of tech cells to the end user using `library_interface_only`
    - You can still use `hdlin_interface_only` to black-box other cells or designs
- Usage in earlier versions:
  - `set hdlin_interface_only "lib/Cell_A lib/Cell_B"`
  - `set hdlin_interface_only "module_CCC"`

  Cell that will have "interface_only": "module_CCC"

- Usage in versions starting with F-2011.09:
  - `set library_interface_only "lib/Cell_A lib/Cell_B"`
  - `set hdlin_interface_only "module_CCC"`

  Cells that will have "interface_only: "lib/Cell_A lib/Cell_B module_CCC"

Updates to Formality Commands and Variables

- report_constant_sources
  - Reports the origin of constants on a specified net
  - New command

```
report_constant_sources [ netID ... ]
```

  - Traces through the fanin cone of net until it reaches
    - Constant LOGIC0 or LOGIC1 source
    - Net with a user-specified constant

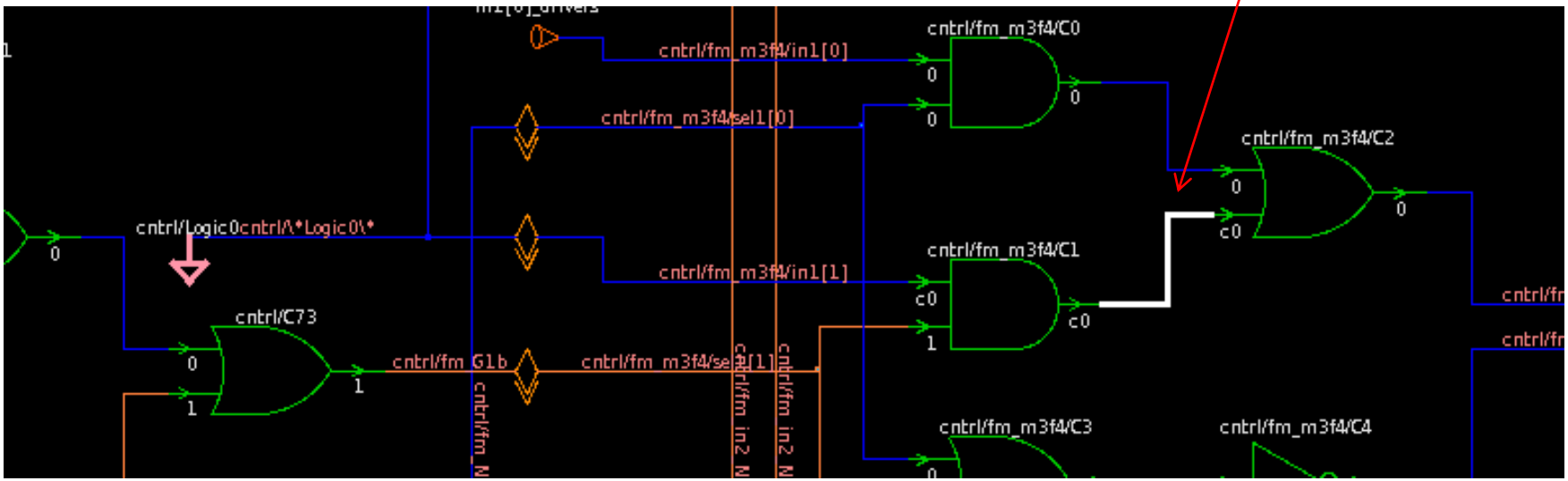
Updates to Formality Commands and Variables

– Example

```
Formality (verify)> report_constant_source r:/WORK/mR4000/cntrl/fm_m3f4/and_1out

Constant     Object
 Value        Type      Object Name
--------     ------     -----------
   0          Cell      r:/WORK/mR4000/cntrl/Logic0
1 Net with a user-specified constant
```

Selected net



Updates to Formality Commands and Variables

- `report_multidriven_nets` (E-2010.12-SP1)
  - Reports information about multidriven nets
    - Report available after match phase
    - Report each multiply driven net and its list of drivers
  - Command

```
report_multidriven_nets
      [-substring substring]
      [-reference]
      [-implementation]
```

`-substring` Reports only the nets containing the specified substring.

`-reference` Reports only the nets in the reference design.

`-implementation` Reports only the nets in the implementation design

Updates to Formality Commands and Variables

- Example

```
fm_shell> report_multidriven_nets
          2 Multiply driven nets:

          Net     r:/WORK/top/Z
          Drivers (3)
                  r:/WORK/and_multi/C1/OUT
                  r:/WORK/top/mid/out       (inv)  (TECH_WORK)
                  r:/WORK/and_multi/C0/OUT

          Net     i:/WORK/top/Z
          Drivers (3)
                  i:/WORK/and_multi/C1/OUT
                  i:/WORK/top/mid/out       (inv)  (TECH_WORK)
                  i:/WORK/and_multi/C0/OUT
```

Updates to Formality Commands and Variables

- `report_undriven_nets (E-2010.12-SP1)`
  - Reports information about undriven nets
    - Report available after match phase
  - Command

```
report_undriven_nets
        [-substring substring]
        [-reference]
        [-implementation]
```

`-substring` Reports only the nets containing the specified substring.

`-reference` Reports only the nets in the reference design.

`-implementation` Reports only the nets in the implementation design.

Updates to Formality Commands and Variables

– Example

```
fm_shell> report_undriven_nets

          4 Undriven nets:

          r:/WORK/top/out_or
          r:/WORK/top/or_one.a
          i:/WORK/top/and_zero.b
          i:/WORK/top/out_or
```

Predictable Success