ECE 429 - Tutorial III: Hierarchical Design and Formal Verification

Created by Jia Wang, Aug. 2011, Revised by Ken Choi, September 2015.

Overview

This tutorial introduces you to hierarchical design and formal verification techniques that are essential to build complex circuits. We will build a 2-input AND gate from a NAND gate and an inverter using the Cadence Virtuoso platform. Then we will verify its functionality formally using the Synopsys Formality ESP equivalence checker.

Since network failure may interrupt your operation, please save your data often.

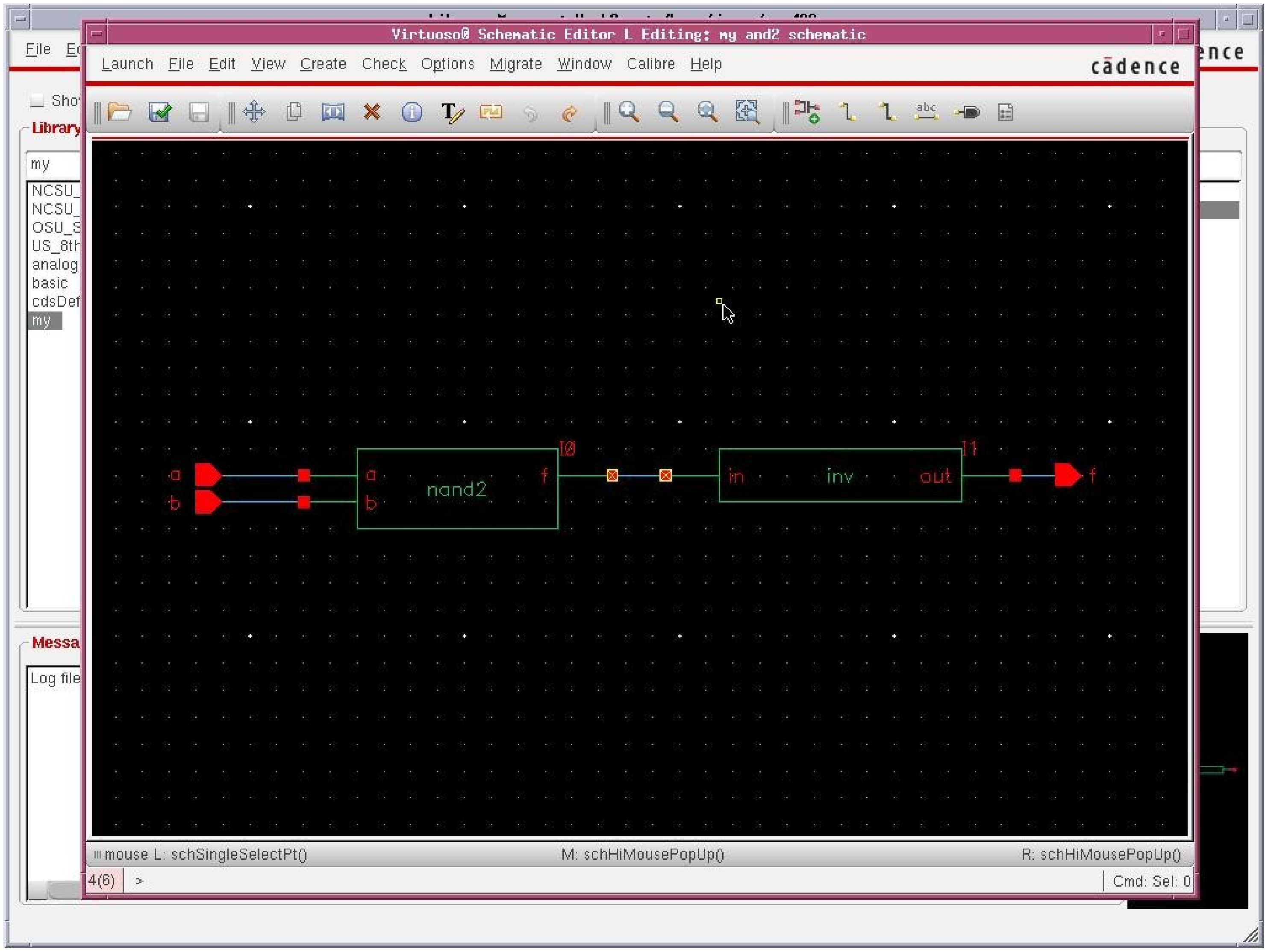
Table of Contents

1. Hierarchical Schematic Design using Virtuoso

I would assume you have completed Tutorial I, Tutorial II, and Lab 4 in the working directory 'ece429'. Start Virtuoso:

```
source /import/scripts/ece429.cshrc
source /import/scripts/hspice.cshrc
source /import/scripts/synopsys2012.cshrc
cd ece429
virtuoso
```

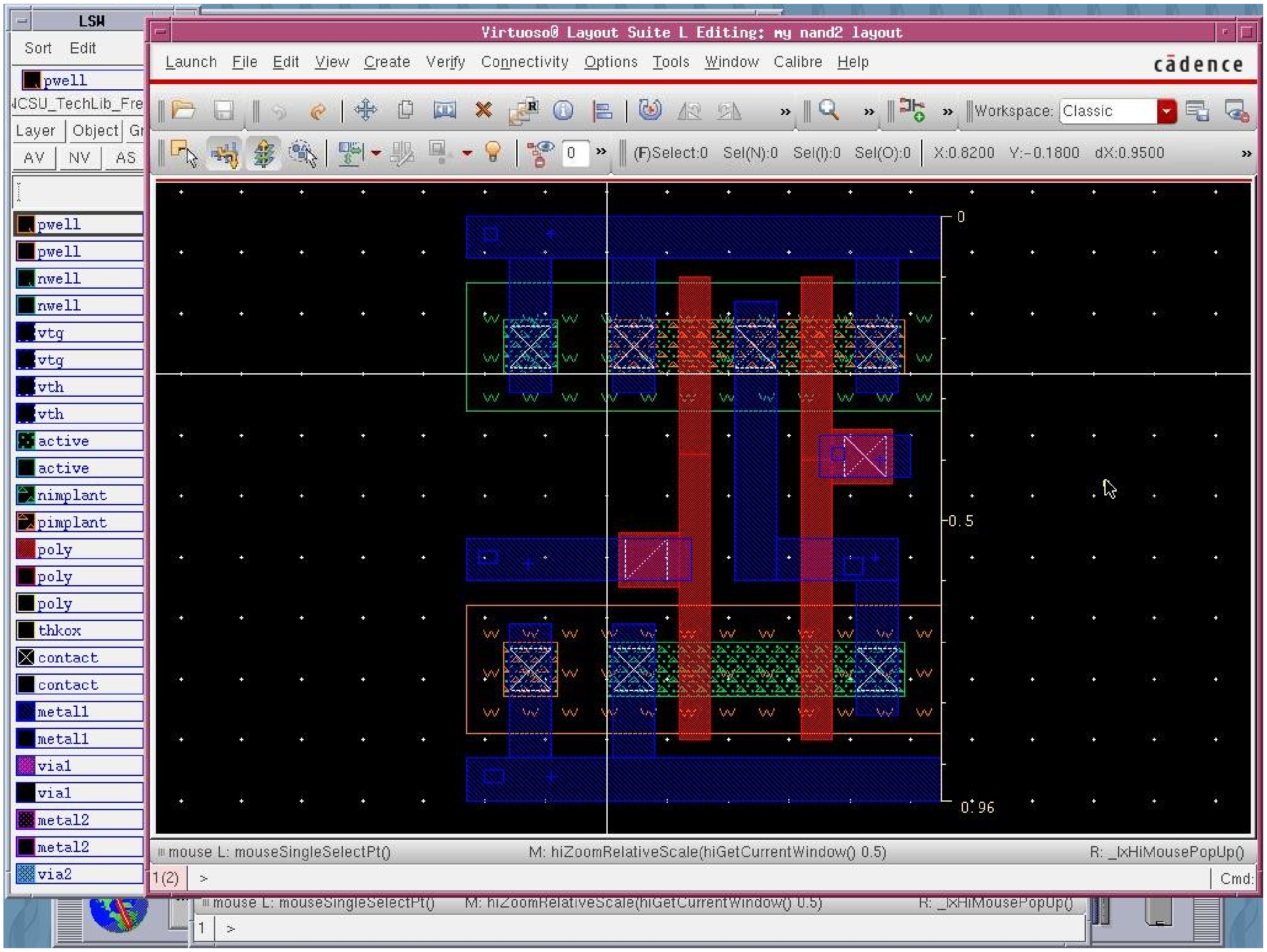
Create a cell 'and2' and its 'schematic' view for our 2-input AND gate design. Assume your 2-input NAND gate design from Lab 4 is a cell 'nand2' in the library 'my'. Similar to Tutorial I, create an instance of the inverter from the 'symbol' view of the cell 'inv', and an instance of the NAND gate from the 'symbol' view of the cell 'nand2'. Introduce two input pins 'a' and 'b', and one output pin 'f'. Wiring them accordingly to form a 2-input AND gate design.


Virtuoso® Schematic Editor L Editing: my and2 schematic
Launch File Edit View Create Check Options Migrate Window Calibre Help
cādence
a
b
I0
a
b
nand2
f
I1
in
inv
out
f
mouse L: schSingleSelectPt()
M: schHiMousePopUp()
R: schHiMousePopUp()
4(6)
>
Cmd: Sel: 0

2. Hierarchical Layout Design using Virtuoso

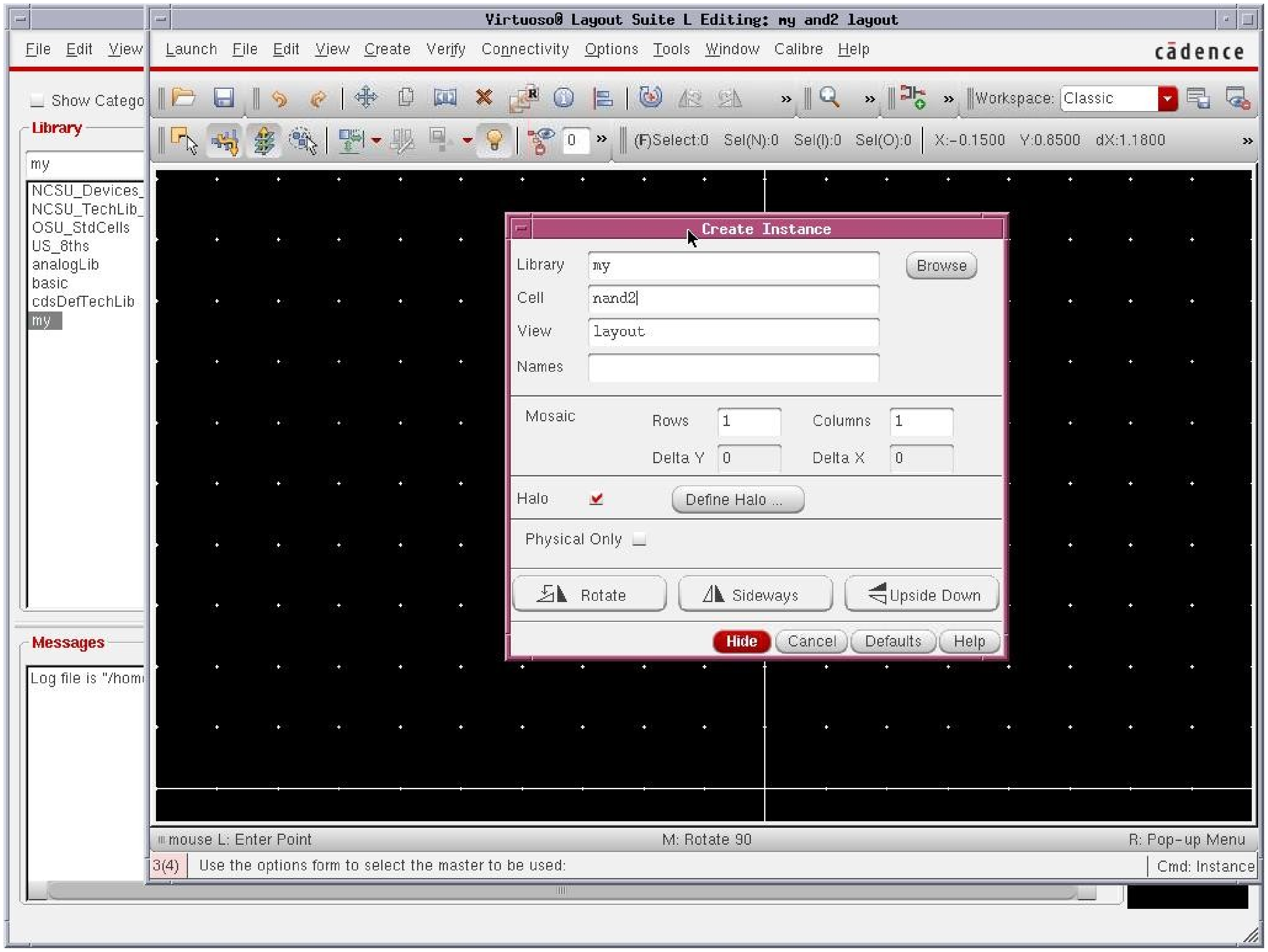
a) Layout of 2-Input NAND Gate

Create the 'layout' view of the cell 'nand2'. Draw the layout of a 2-input NAND gate. Run DRC/LVS and correct all errors.


LSW
Sort Edit
NCSU_TechLib_Fre
Layer Object Gr
AV NV AS
pwell
nwell
vtg
vth
active
nimplant
pimplant
poly
thkox
contact
metal1
via1
metal2
via2
Virtuoso® Layout Suite L Editing: my nand2 layout
Launch File Edit View Create Verify Connectivity Options Tools Window Calibre Help
cadence
Workspace: Classic
(F)Select:0 Sel(N):0 Sel(I):0 Sel(O):0 X:0.8200 Y:-0.1800 dX:0.9500
0
0.5
0.96
mouse L: mouseSingleSelectPt() M: hiZoomRelativeScale(hiGetCurrentWindow() 0.5) R: _lxHiMousePopUp()
1(2) >
Cmd:

b) Hierarchical Layout Design

Create the 'layout' view of the cell 'and2'. Within the layout editor, create an instance of the NAND gate layout from the 'layout' view of the cell 'nand2'.


Virtuoso® Layout Suite L Editing: my and2 layout

Create Instance

Library: my
Cell: nand2
View: layout
Names:

Mosaic — Rows: 1, Columns: 1, Delta Y: 0, Delta X: 0

Halo ✓

Physical Only ☐

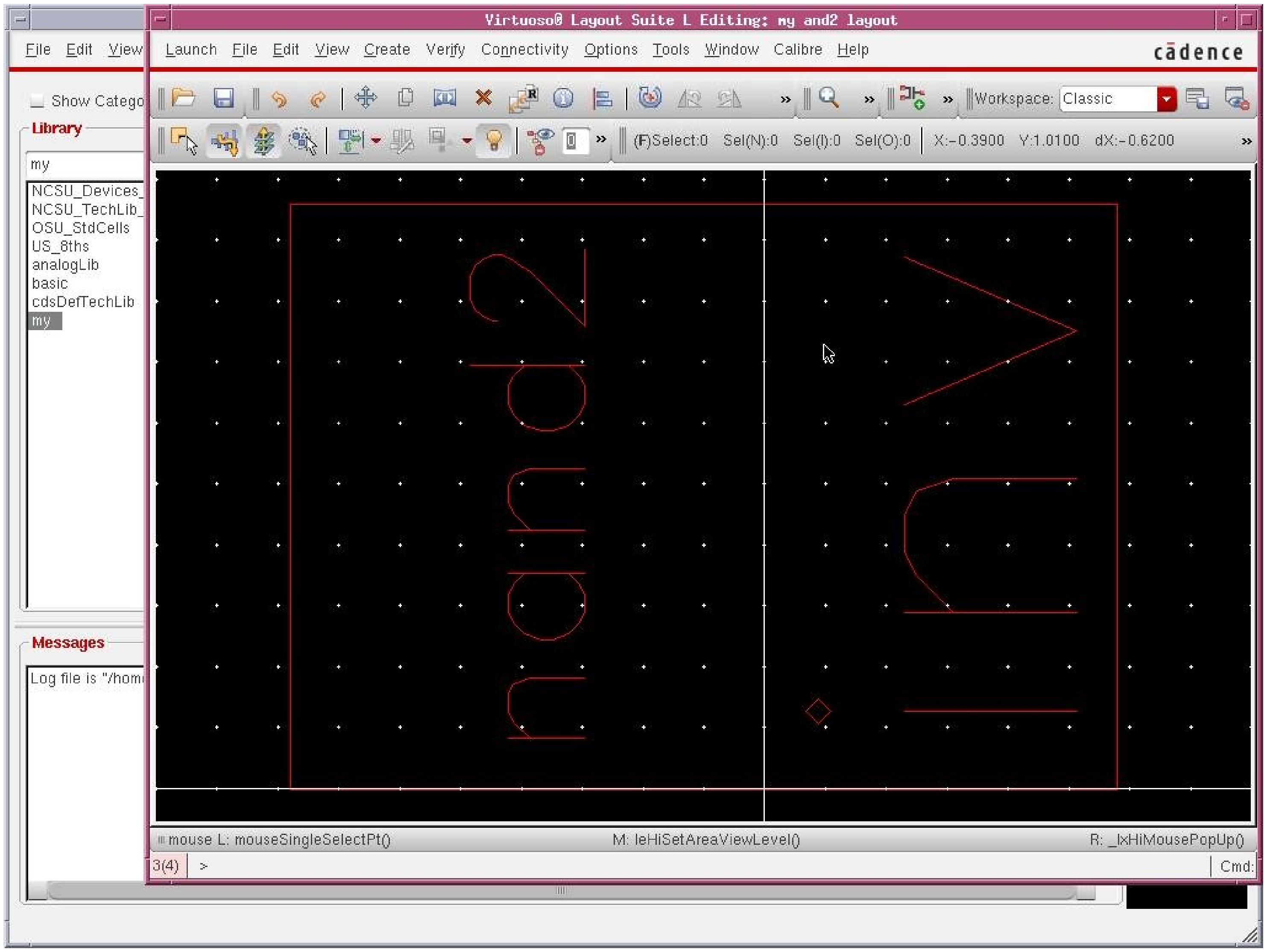
(F)Select:0 Sel(N):0 Sel(I):0 Sel(O):0 X:-0.1500 Y:0.8500 dX:1.1800

mouse L: Enter Point   M: Rotate 90   R: Pop-up Menu

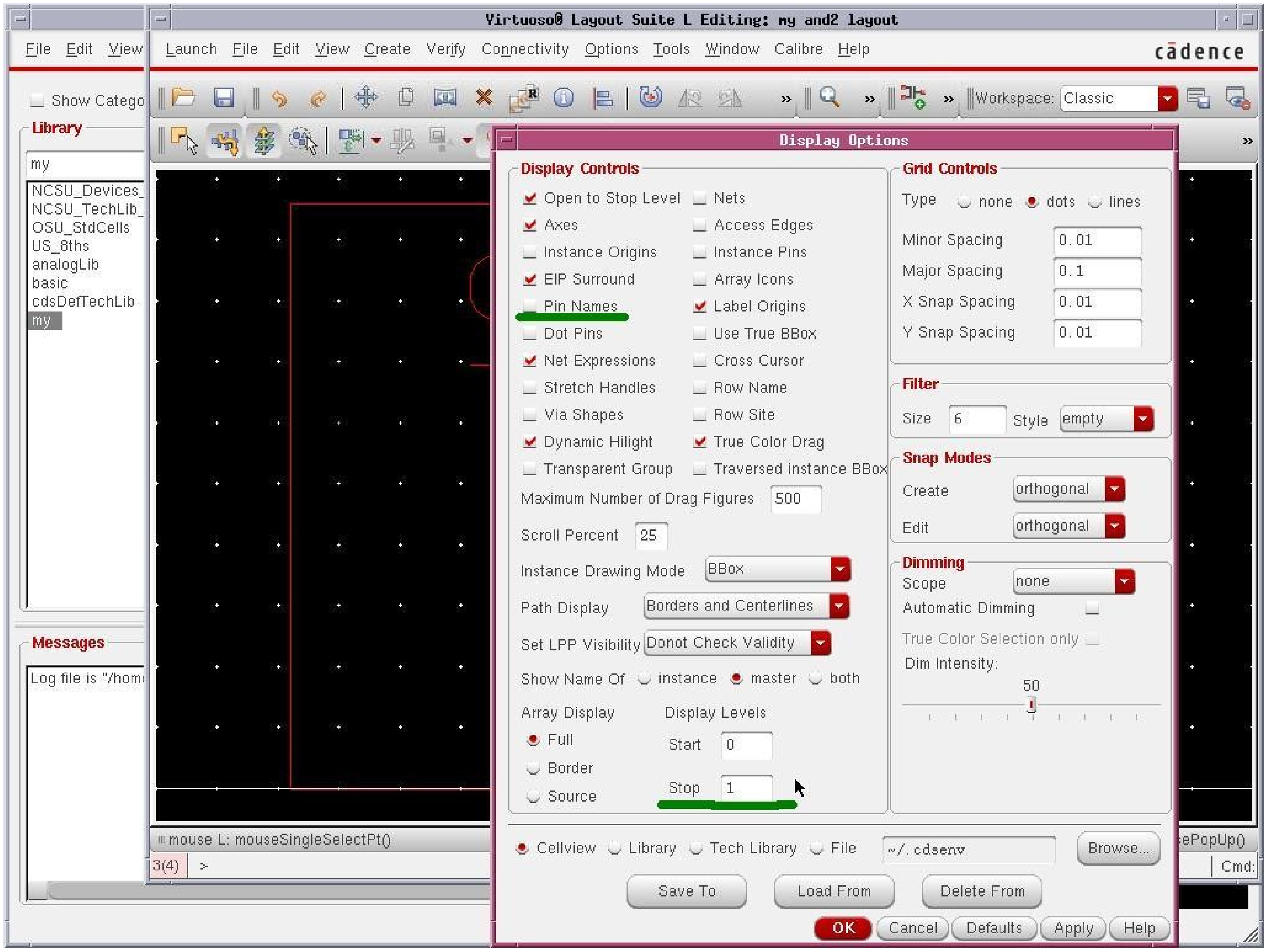
3(4) Use the options form to select the master to be used: Cmd: Instance

Log file is "/hom

Place the instance of the NAND gate layout and then create and place an instance of the inverter layout from the 'layout' view of the cell 'inv'. For simplicity, make the layouts abut.


Virtuoso® Layout Suite L Editing: my and2 layout
File Edit View
Launch File Edit View Create Verify Connectivity Options Tools Window Calibre Help
cādence
Show Catego
Workspace: Classic
(F)Select:0 Sel(N):0 Sel(I):0 Sel(O):0 X:-0.3900 Y:1.0100 dX:-0.6200
Library
my
NCSU_Devices_
NCSU_TechLib_
OSU_StdCells
US_8ths
analogLib
basic
cdsDefTechLib
my
Messages
Log file is "/hom
nand2
in
mouse L: mouseSingleSelectPt()
M: leHiSetAreaViewLevel()
R: _lxHiMousePopUp()
3(4)
Cmd:

Bring out the Display Options dialog. Uncheck 'Pin Names' in 'Display Controls'. Then change 'Stop' under 'Display Levels' to 1.



Display Options

Display Controls

[✓ Open to Stop Level] Nets
[✓ Axes] Access Edges
Instance Origins Instance Pins
[✓ EIP Surround] Array Icons
Pin Names [✓ Label Origins]
Dot Pins Use True BBox
[✓ Net Expressions] Cross Cursor
Stretch Handles Row Name
Via Shapes Row Site
[✓ Dynamic Hilight] [✓ True Color Drag]
Transparent Group Traversed instance BBox

Maximum Number of Drag Figures 500
Scroll Percent 25
Instance Drawing Mode BBox
Path Display Borders and Centerlines
Set LPP Visibility Donot Check Validity
Show Name Of instance [✓ master] both
Array Display: [✓ Full] Border Source
Display Levels: Start 0, Stop 1

Grid Controls

Type none [✓ dots] lines
Minor Spacing 0.01
Major Spacing 0.1
X Snap Spacing 0.01
Y Snap Spacing 0.01

Filter

Size 6 Style empty

Snap Modes

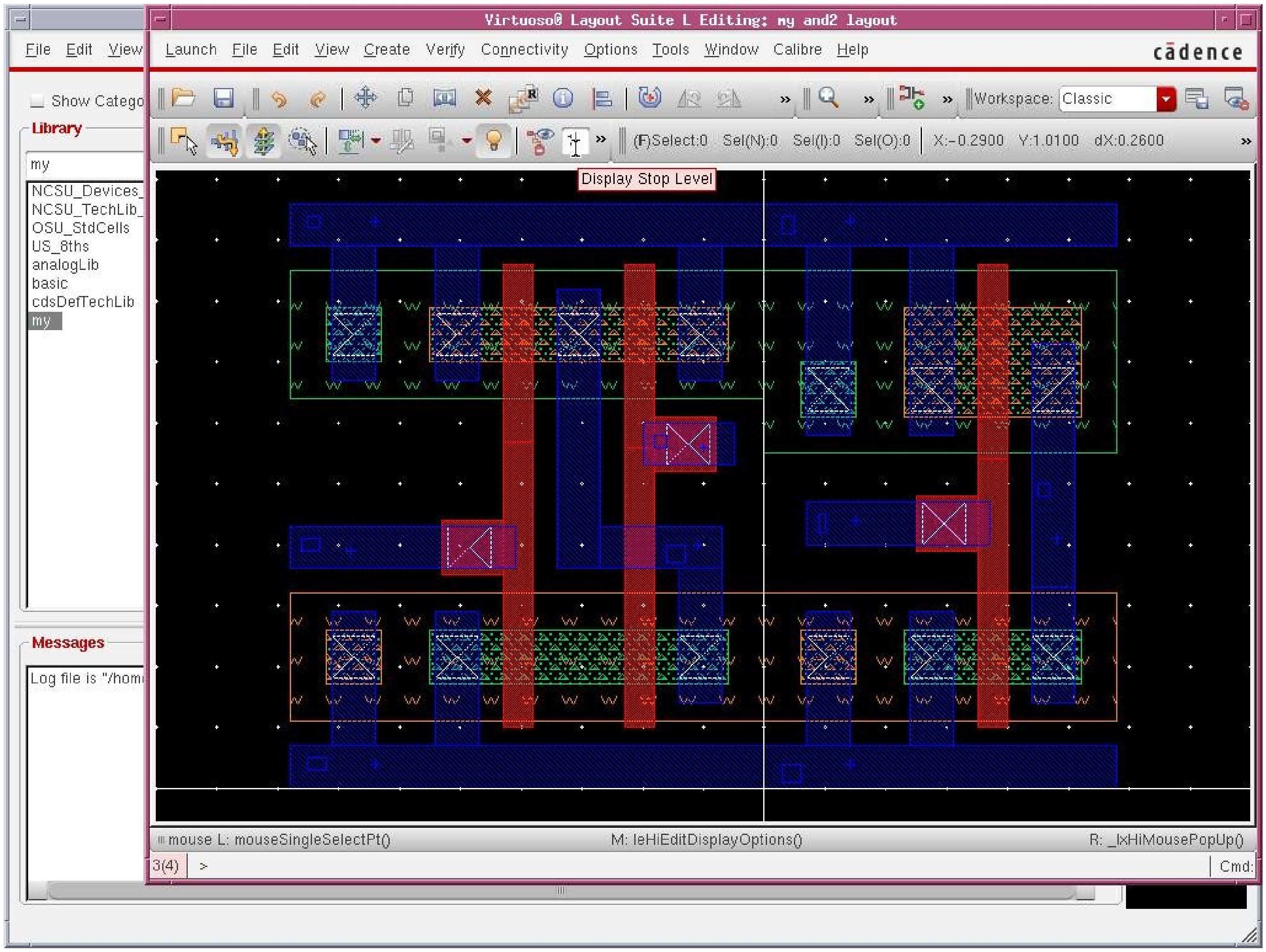
Create orthogonal
Edit orthogonal

Dimming

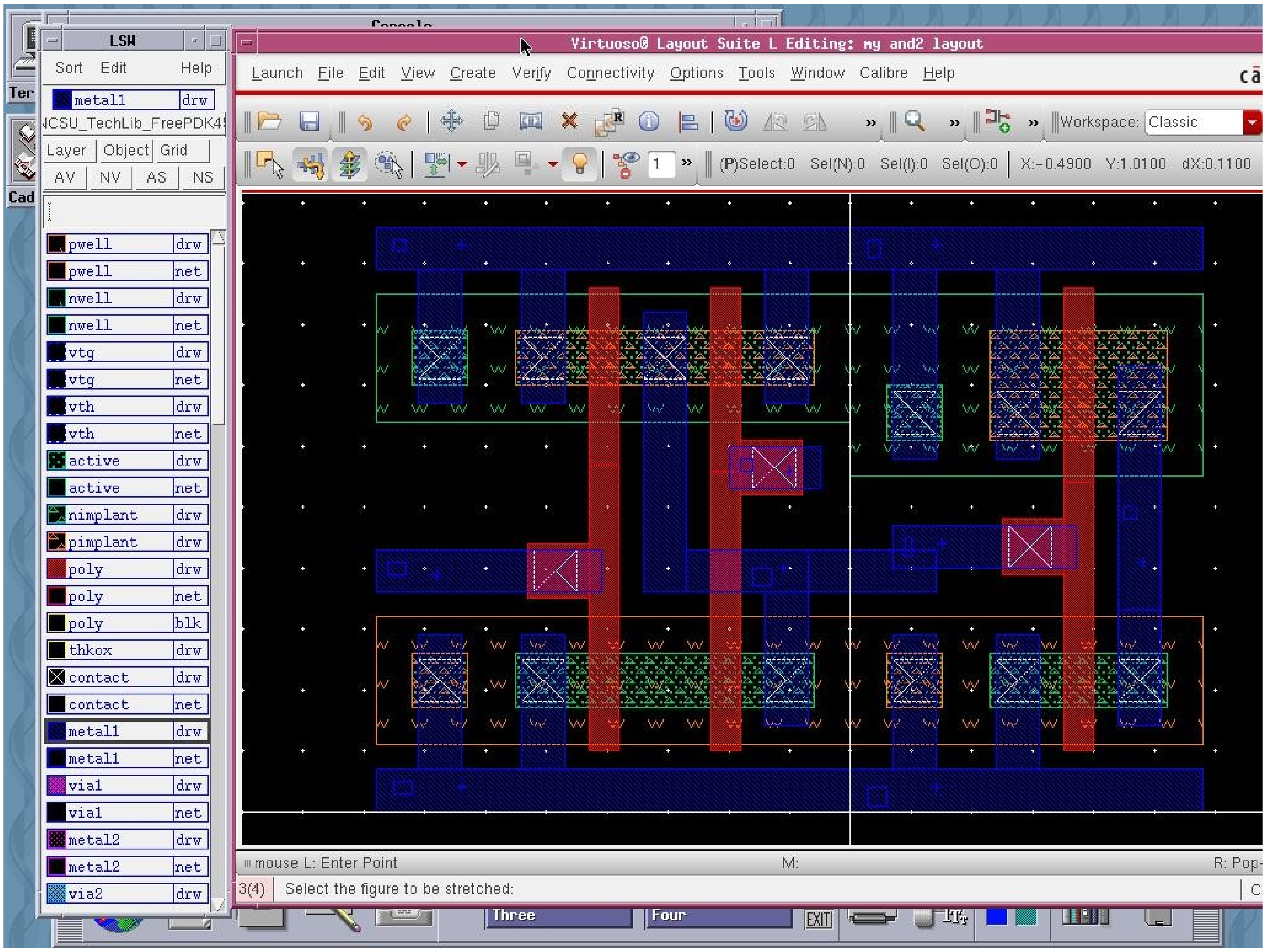
Scope none
Automatic Dimming
True Color Selection only
Dim Intensity: 50

[✓ Cellview] Library Tech Library File ~/.cdsenv

Click 'OK' and you should be able to see 'through' the NAND gate and the inverter for their layouts. Also note that you can change the 'Display Stop Level' directly on the toolbar.

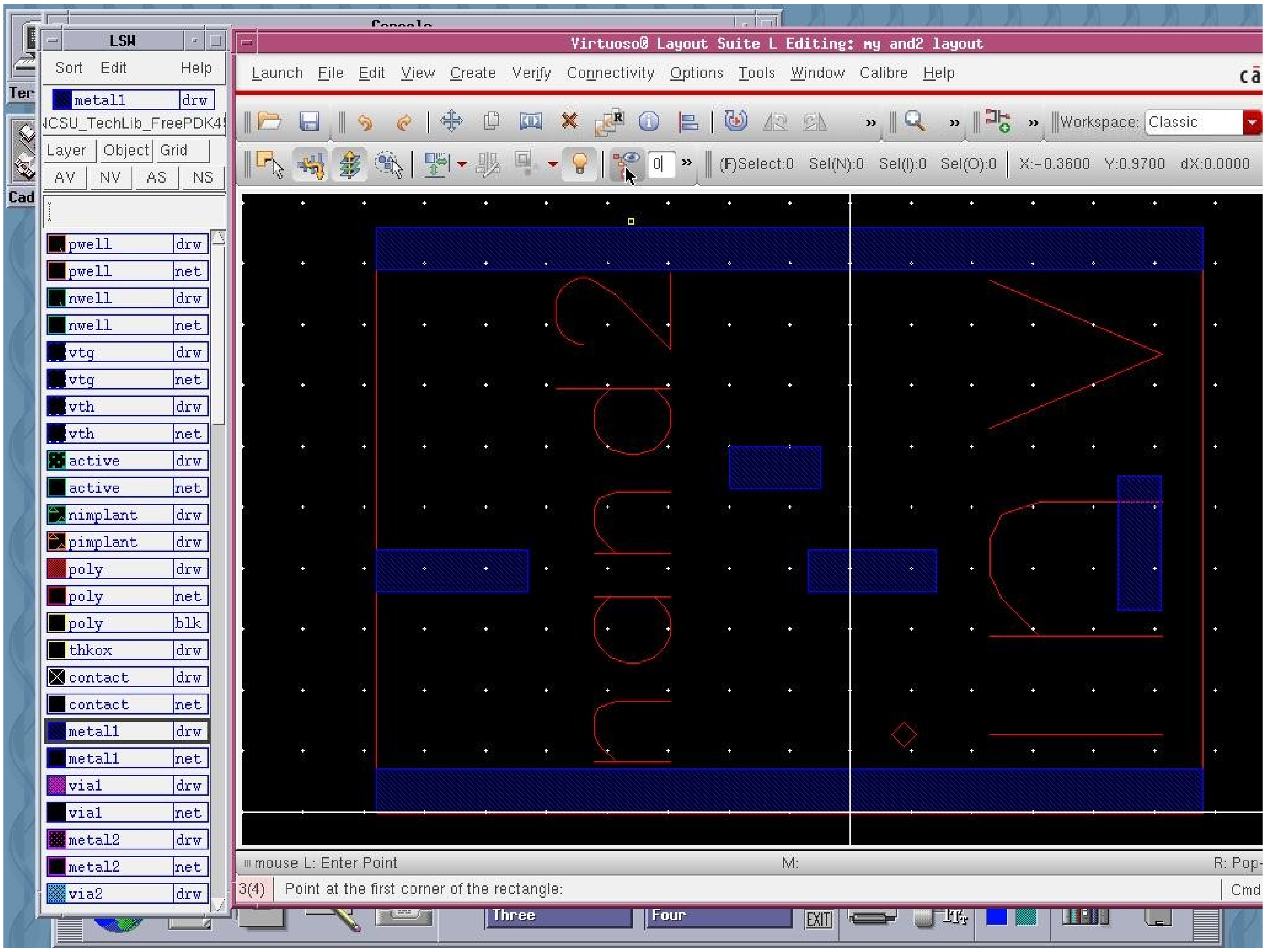

Virtuoso® Layout Suite L Editing: my and2 layout
Launch File Edit View Create Verify Connectivity Options Tools Window Calibre Help
cadence
Workspace: Classic
(F)Select:0 Sel(N):0 Sel(I):0 Sel(O):0 X:-0.2900 Y:1.0100 dX:0.2600
Display Stop Level
File Edit View
Show Catego
Library
my
NCSU_Devices_
NCSU_TechLib_
OSU_StdCells
US_8ths
analogLib
basic
cdsDefTechLib
my
Messages
Log file is "/hom
mouse L: mouseSingleSelectPt()
M: leHiEditDisplayOptions()
R: _lxHiMousePopUp()
3(4) >
Cmd:

Use a 'metal1|drw' rectangle to connect the 'nand2' output to the 'inv' input. Note that although logically there exists a hierarchy, physically rectangles within different cells but on the same layer belong to a single layer. Therefore you can connect a metal1 rectangle in 'nand2' to a metal1 rectangle in 'inv' using a metal1 rectangle at the top-most level.

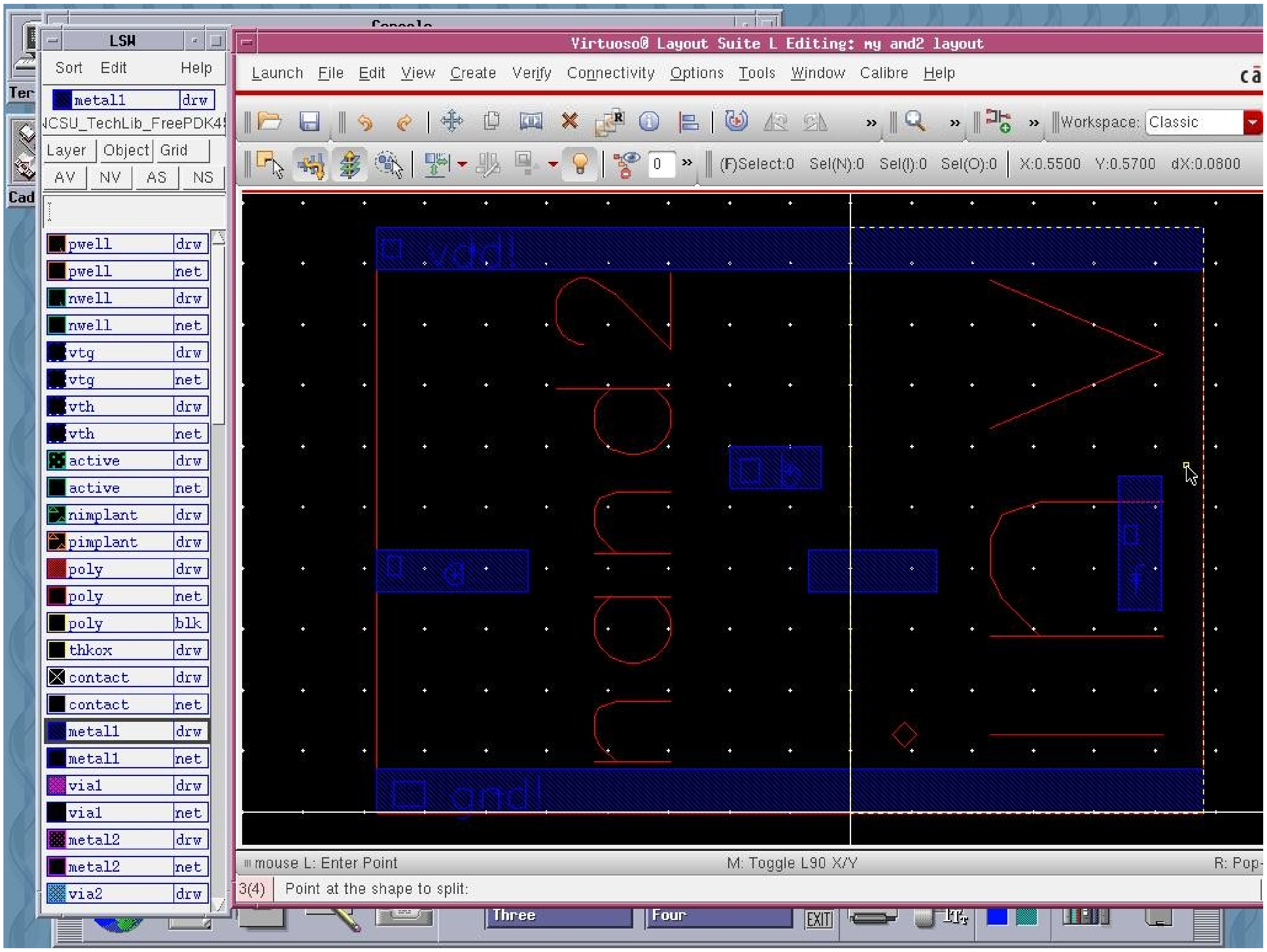


Note that since we design the layouts of 'nand2' and 'inv' to have the same height, the VDD and the GND rails in the layouts are connected automatically. Run DRC and there should be no error.

Now we need to introduce pins to the layout so we can run LVS. To make this task easy, draw a few 'metal1|drw' rectangles to overlap the inputs and the output of the 2-input AND gate. Don't forget the VDD and the GND rail. Bring out the Display Options dialog again. Check 'Pin Names' in 'Display Controls' and change 'Stop' under 'Display Levels' to 0. You should see only the rectangles at the top-most level.



Add the pins to the layout. Run LVS and there should be no error.



3. Equivalence Checking using Formality ESP

While you can always validate the functionality of your schematic/layout designs by SPICE or other simulation techniques, it will become prohibitive for complex VLSI designs -- technically, you need to double the number of the input patterns for every additional bit of input. Formal verification techniques provide an alternative to verify the functionality of your designs using mathematics, and become a must-have in today's VLSI design flow because of the complexity in the designs.

One of the most mature formal verification techniques is called equivalence checking that determines whether two designs have the same functionality. Usually one of the two designs, called the reference, is easier to make error-free and the other, called the implementation, is prone to errors. Equivalence checking can effectively detect errors in the implementation as the discrepancies between the two.

In some sense, LVS is one example of equivalence checking: any error in the layout can be detected using the schematic. However, as the complexity of the circuit increases, it will become more difficult to make the schematic designs correct. Therefore, we need to introduce designs at higher abstraction levels and use tools supporting equivalence checking at such levels to verify the schematics. In this tutorial, we will introduce Formality ESP that supports to verify a schematic as a SPICE netlist against a Verilog model.

a) Prepare a Verilog file for AND Gate

Create the Verilog description of a 2-input AND gate in the file 'and2.v' as follows.

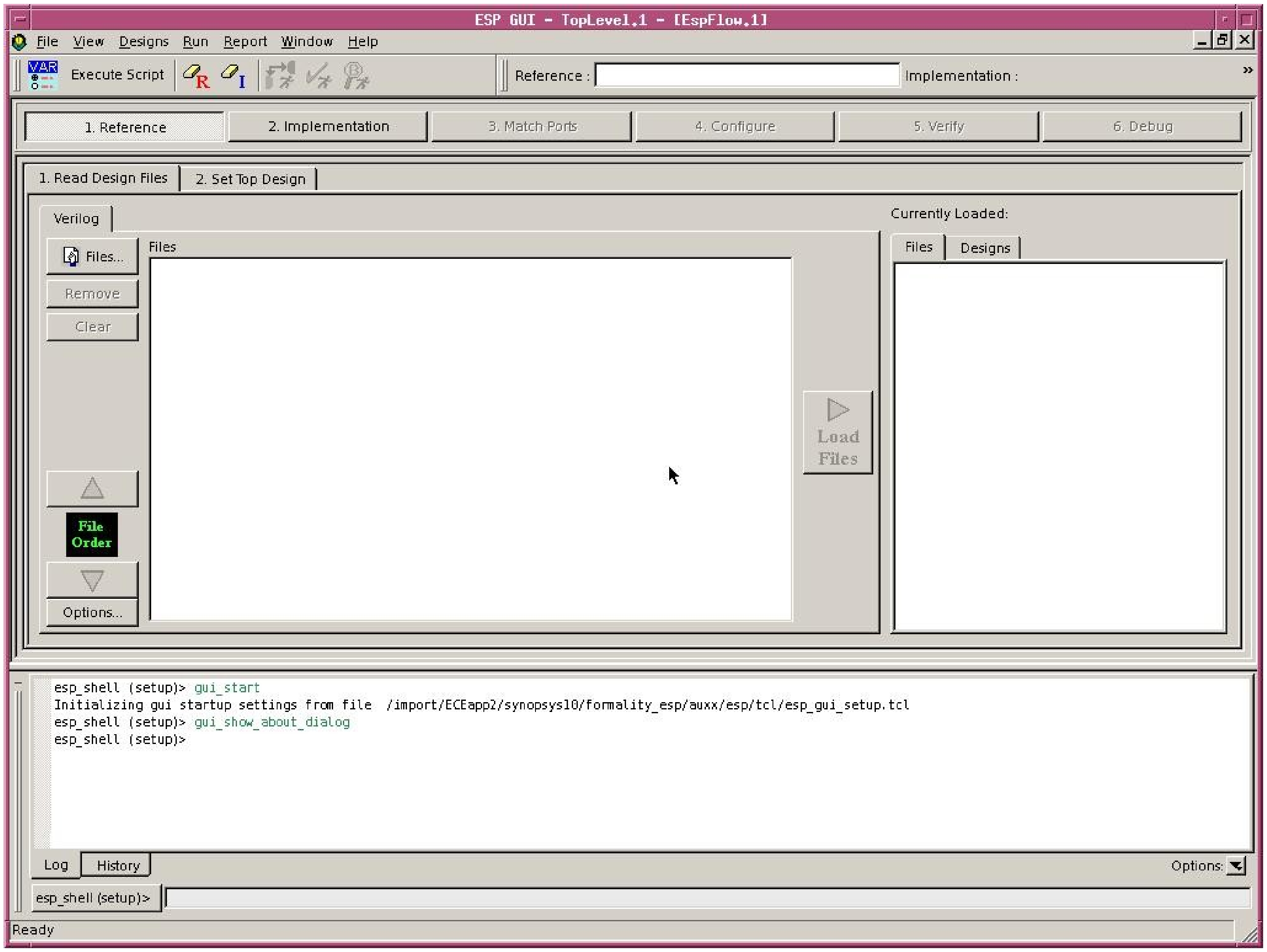
```
module and2(a,b,f);
   input a,b;
   output f;
   assign f=a&b;
endmodule
```

In case you don't have any previous experience with Verilog, the above code defines a module 'and2', i.e. something represents a gate in Verilog, with two inputs 'a' and 'b' and one output 'f'. The line starts with 'assign' defines the relationship between the inputs and the output -- an AND gate. Note that I match the names for the module, the inputs, and the output with our 'and2' schematic in order to make the equivalence checking easier. We will stop here on Verilog since that is all you need to know to complete this tutorial.

b) Equivalence Checking of Verilog and Schematic

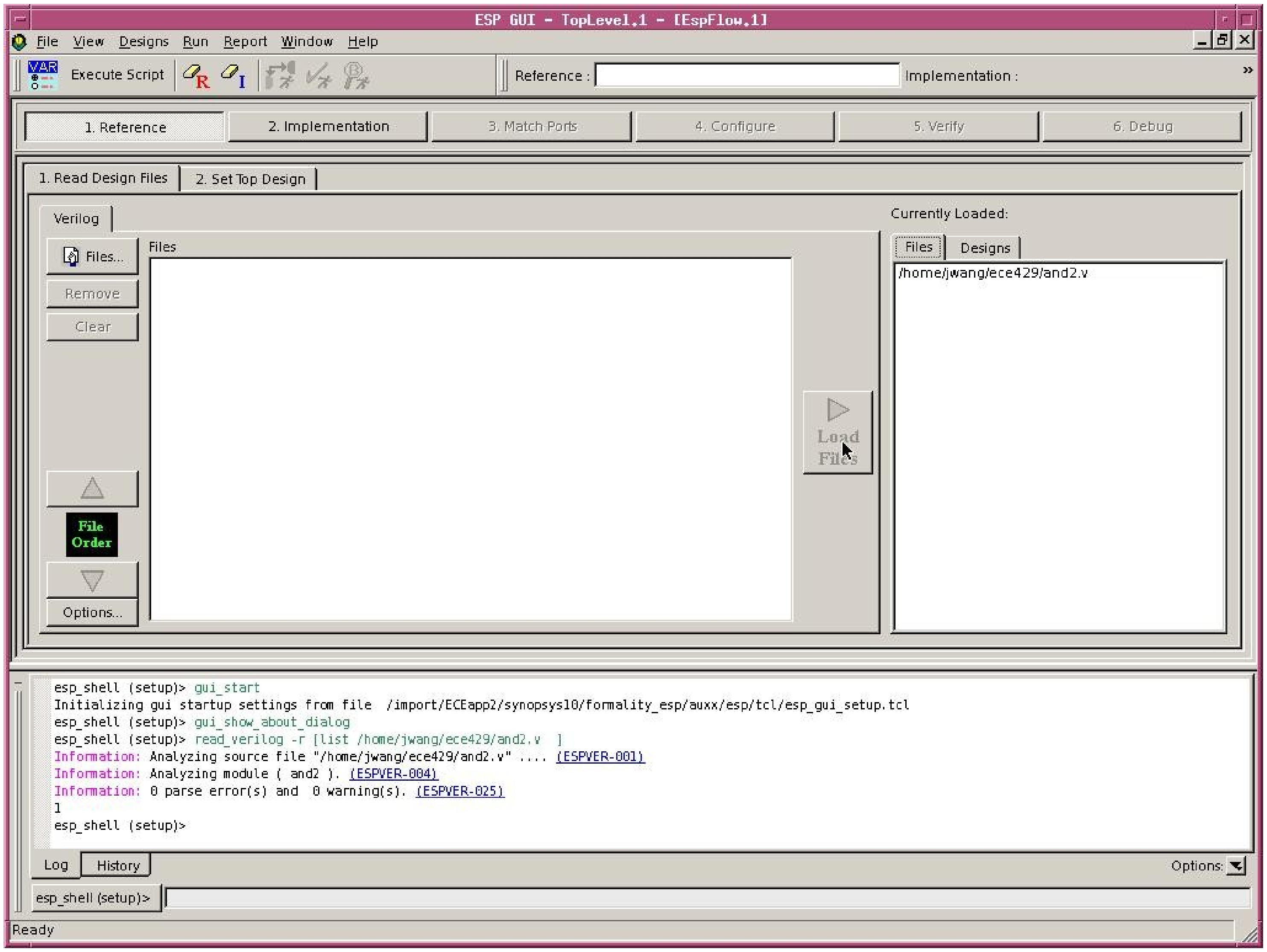
Start Formality ESP:
esp -gui

You will see a guided interface to perform equivalence checking: the steps are numbered from 1 to 6 and you won't be able to proceed to the next step without completing the current step correctly.



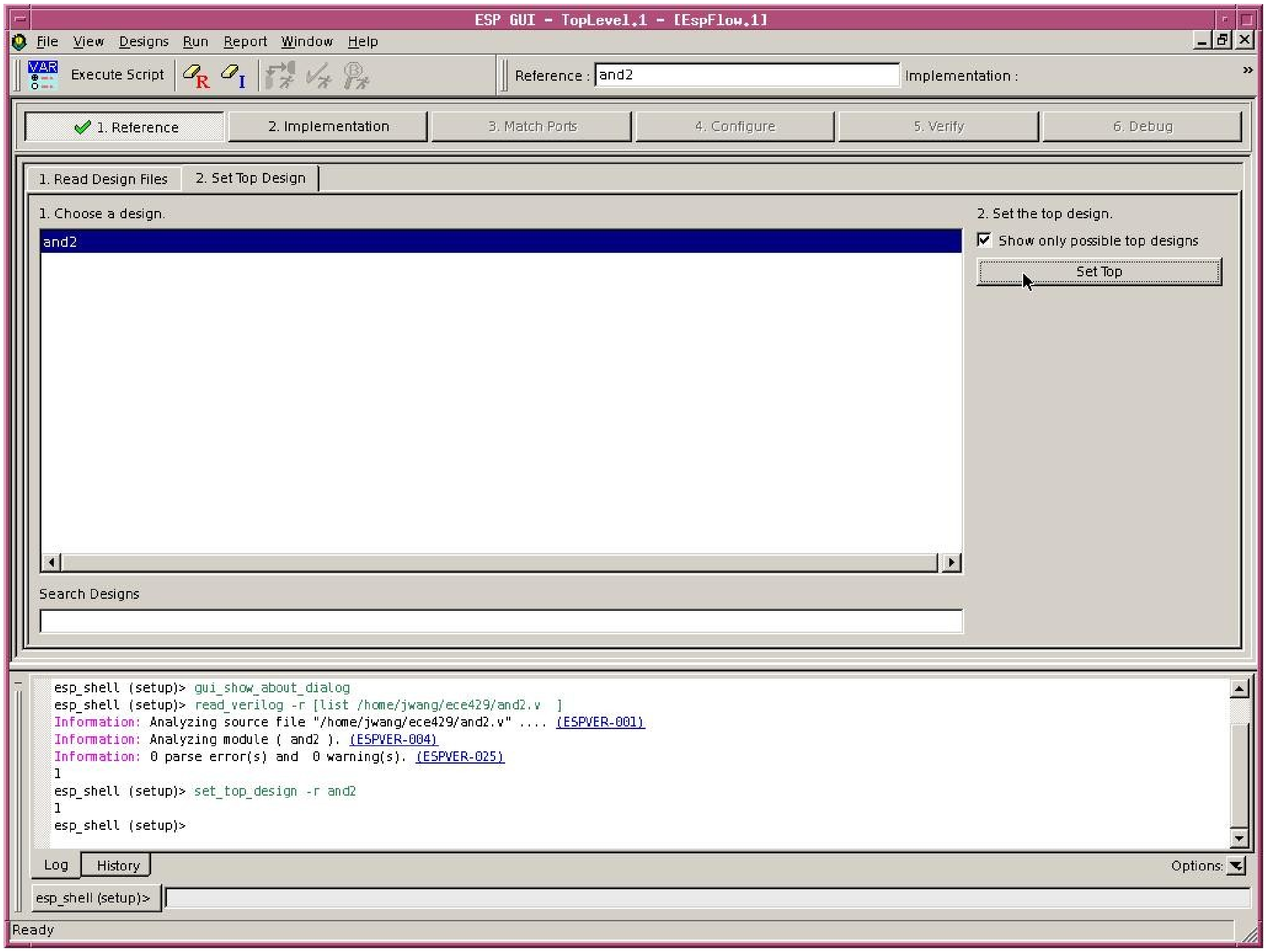
```
esp_shell (setup)> gui_start
Initializing gui startup settings from file  /import/ECEapp2/synopsys10/formality_esp/auxx/esp/tcl/esp_gui_setup.tcl
esp_shell (setup)> gui_show_about_dialog
esp_shell (setup)>
```

To complete the first step '1. Reference', we need to first appoint a reference design in Verilog. Click the 'Files...' button to choose 'and2.v'. The 'Load Files' button in the middle will then be enabled. Click it to load 'and2.v' into 'Currently loaded'. There should be no error in the Log window at the bottom.



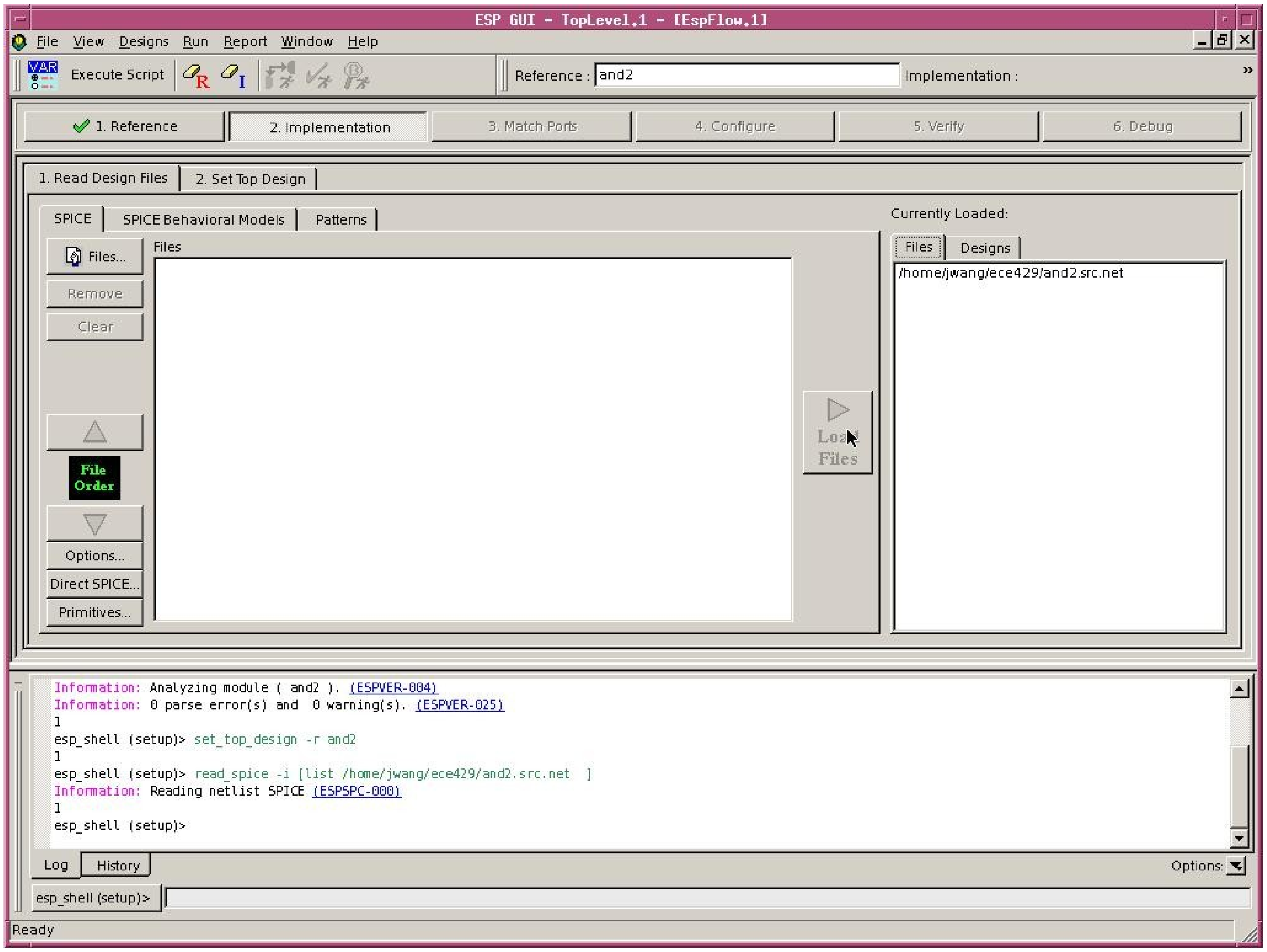
```
esp_shell (setup)> gui_start
Initializing gui startup settings from file  /import/ECEapp2/synopsys10/formality_esp/auxx/esp/tcl/esp_gui_setup.tcl
esp_shell (setup)> gui_show_about_dialog
esp_shell (setup)> read_verilog -r [list /home/jwang/ece429/and2.v  ]
Information: Analyzing source file "/home/jwang/ece429/and2.v" .... (ESPVER-001)
Information: Analyzing module ( and2 ). (ESPVER-004)
Information: 0 parse error(s) and  0 warning(s). (ESPVER-025)
1
esp_shell (setup)>
```

The second part of the first step is to 'Set Top Design', which tells the tool which module to check as there may exist many modules in a Verilog file. Click '2. Set Top Design', choose 'and2', and then click 'Set Top' on the right. The green check to the side of '1. Reference' appears to indicate the step is completed correctly.



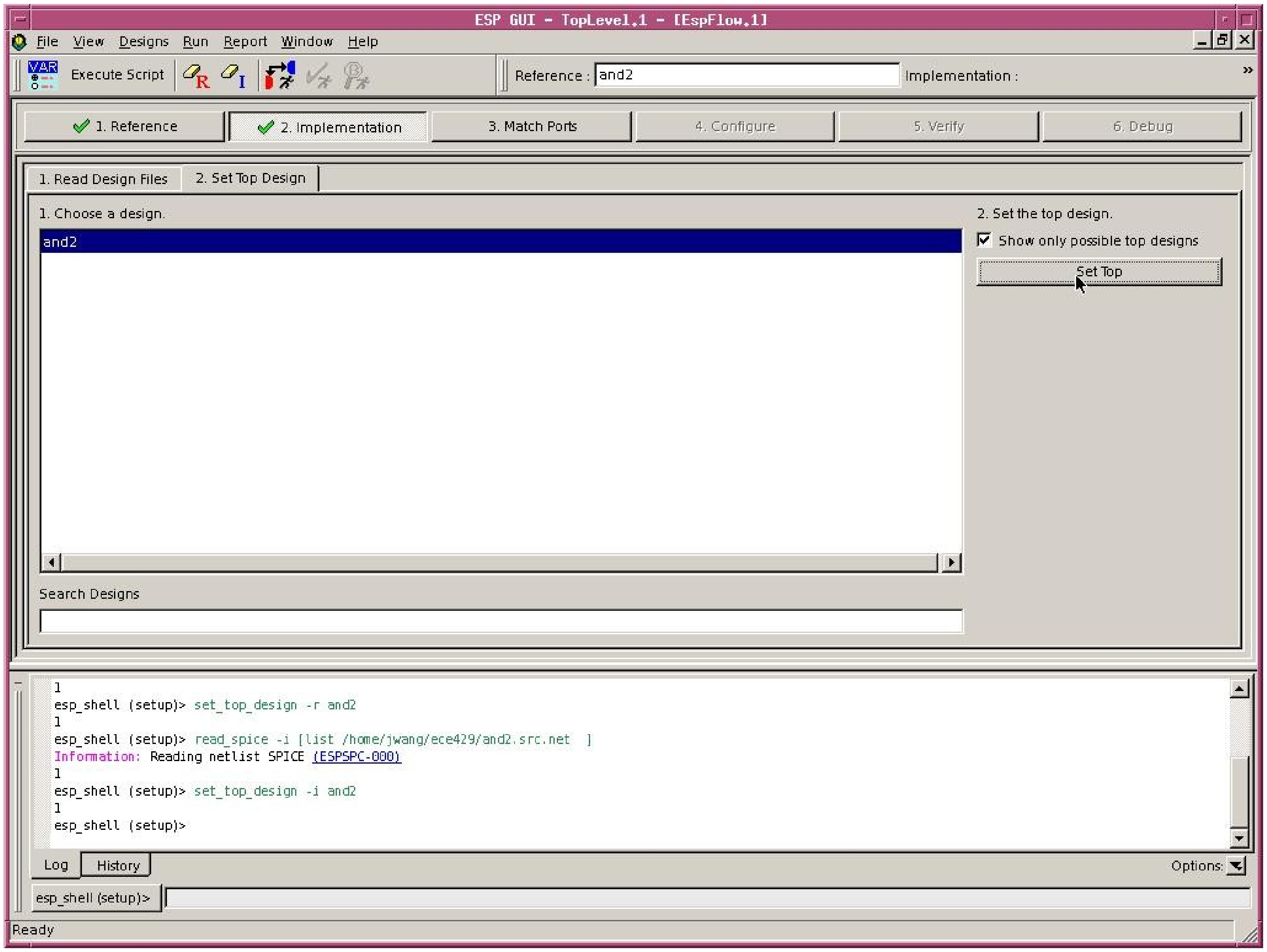
```
esp_shell (setup)> gui_show_about_dialog
esp_shell (setup)> read_verilog -r [list /home/jwang/ece429/and2.v  ]
Information: Analyzing source file "/home/jwang/ece429/and2.v" .... (ESPVER-001)
Information: Analyzing module ( and2 ). (ESPVER-004)
Information: 0 parse error(s) and  0 warning(s). (ESPVER-025)
1
esp_shell (setup)> set_top_design -r and2
1
esp_shell (setup)>
```

Move to the step '2. Implementation'. Choose and load the SPICE netlist 'and2.src.net', which should be generated by LVS   before.



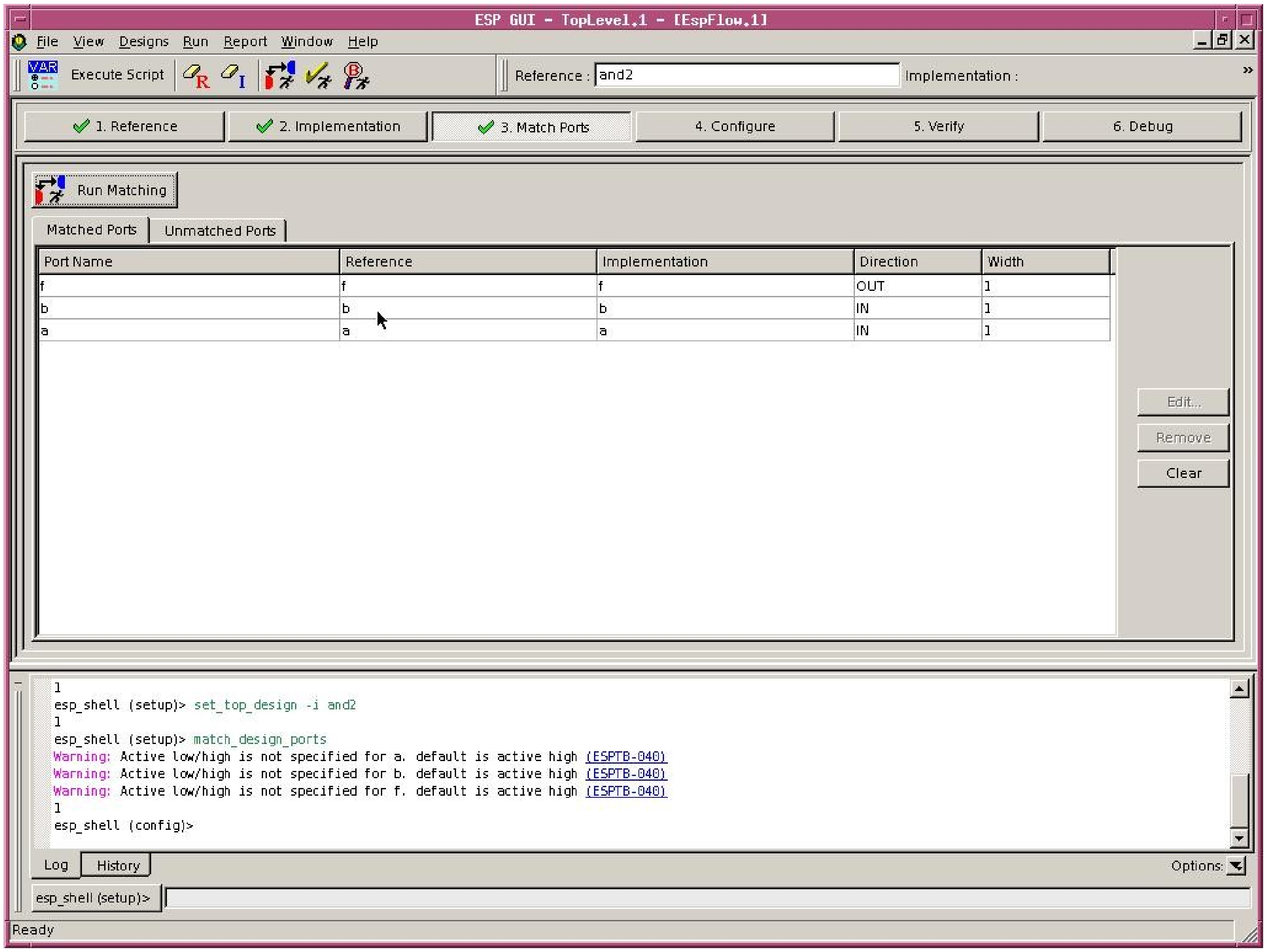
```
Information: Analyzing module ( and2 ). (ESPVER-004)
Information: 0 parse error(s) and  0 warning(s). (ESPVER-025)
1
esp_shell (setup)> set_top_design -r and2
1
esp_shell (setup)> read_spice -i [list /home/jwang/ece429/and2.src.net  ]
Information: Reading netlist SPICE (ESPSPC-000)
1
esp_shell (setup)>
```

Switch to '2. Set Top Design', choose 'and2' and then 'Set Top'. You are done with the second step.



```
1
esp_shell (setup)> set_top_design -r and2
1
esp_shell (setup)> read_spice -i [list /home/jwang/ece429/and2.src.net  ]
Information: Reading netlist SPICE (ESPSPC-000)
1
esp_shell (setup)> set_top_design -i and2
1
esp_shell (setup)>
```

Move to the step '3. Match Ports'. Click 'Run Matching' to match the inputs/outputs of the two designs. Since we use the same name for inputs/outputs in the schematic and the Verilog file, three pairs of matched ports will be identified automatically.

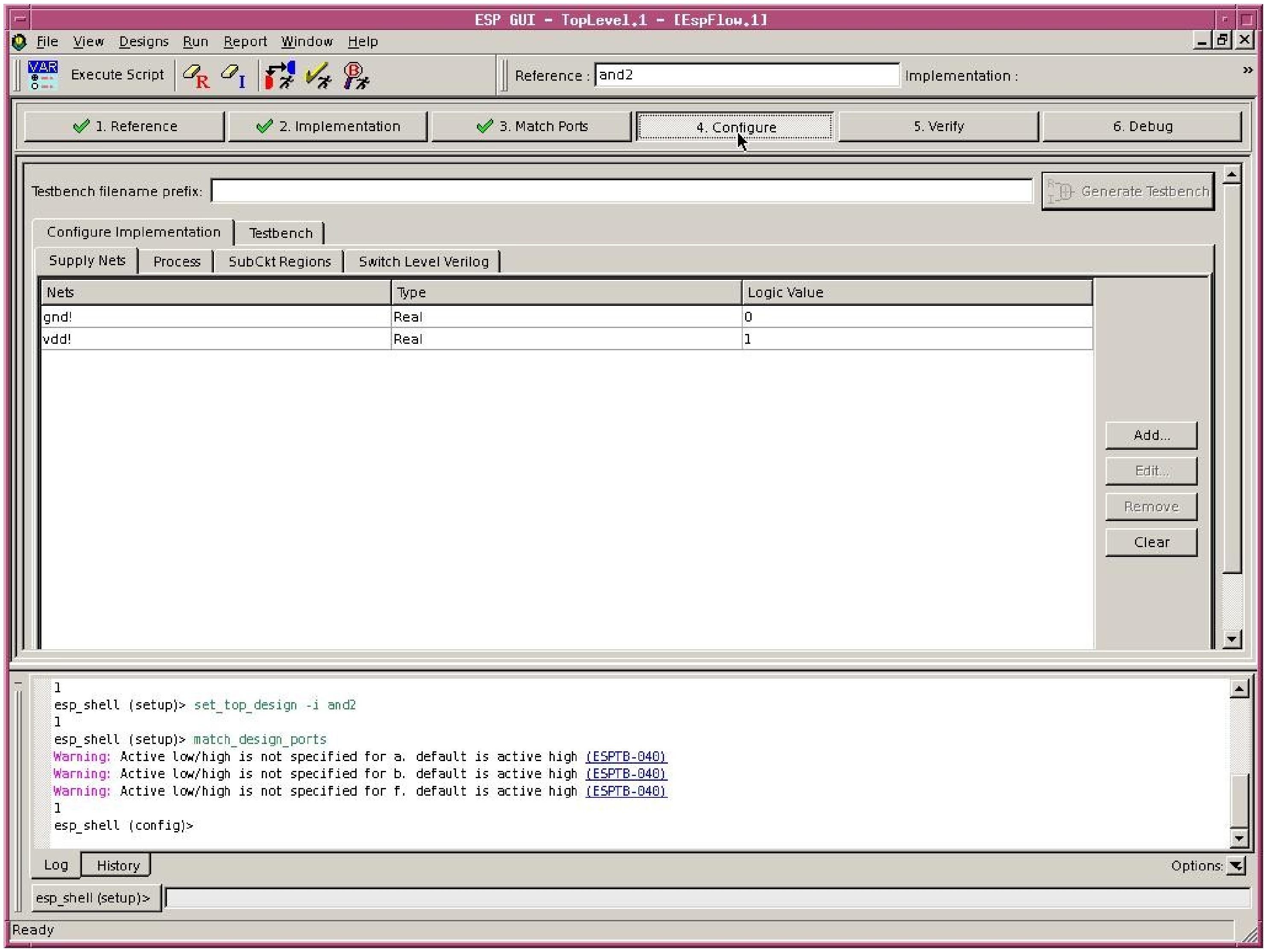


Port Name	Reference	Implementation	Direction	Width
f	f	f	OUT	1
b	b	b	IN	1
a	a	a	IN	1

```
1
esp_shell (setup)> set_top_design -i and2
1
esp_shell (setup)> match_design_ports
Warning: Active low/high is not specified for a. default is active high (ESPTB-040)
Warning: Active low/high is not specified for b. default is active high (ESPTB-040)
Warning: Active low/high is not specified for f. default is active high (ESPTB-040)
1
esp_shell (config)>
```

Note that the Log window shows three warnings. The tool must know which voltage level in the SPICE netlist corresponds to a logical '0' or '1' in the Verilog file. By default, it assumes active high, which means VDD represents '1' and GND represents '0'.

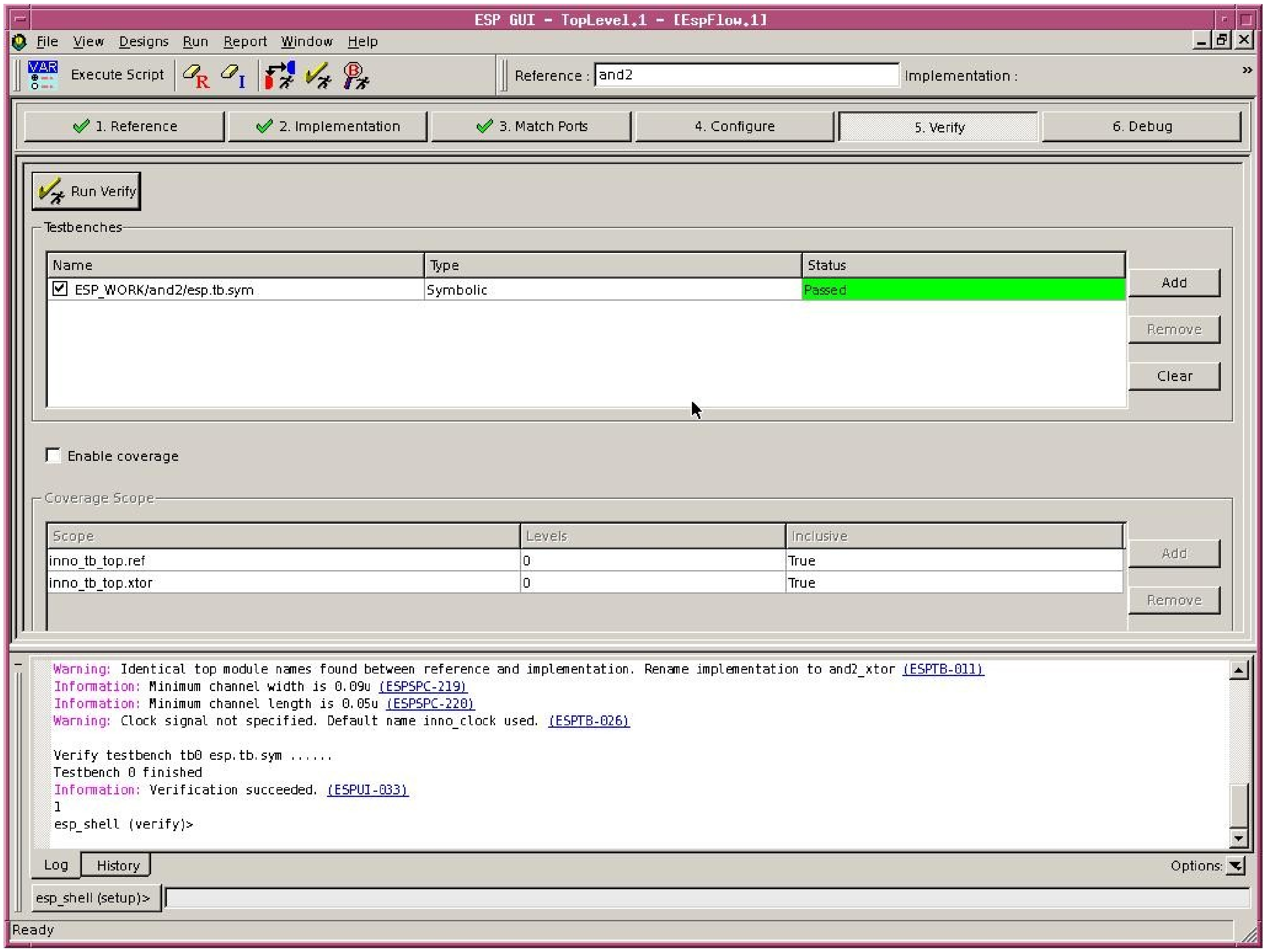
Move to the step '4. Configure'. Make sure 'vdd!' and 'gnd!' are listed under 'Supply Nets' and their corresponding logic values are correct.



Nets	Type	Logic Value
gnd!	Real	0
vdd!	Real	1

```
1
esp_shell (setup)> set_top_design -i and2
1
esp_shell (setup)> match_design_ports
Warning: Active low/high is not specified for a. default is active high (ESPTB-040)
Warning: Active low/high is not specified for b. default is active high (ESPTB-040)
Warning: Active low/high is not specified for f. default is active high (ESPTB-040)
1
esp_shell (config)>
```

Move to the step '5. Verify'. Click 'Run Verify'. You should see the green 'Passed' for the 'Symbolic' testbench. This indicate the two designs -- the Verilog file and the SPICE netlist, are equivalent. So your 'and2' schematic design is correct!



Name	Type	Status
☑ ESP_WORK/and2/esp.tb.sym	Symbolic	Passed

Scope	Levels	Inclusive
inno_tb_top.ref	0	True
inno_tb_top.xtor	0	True

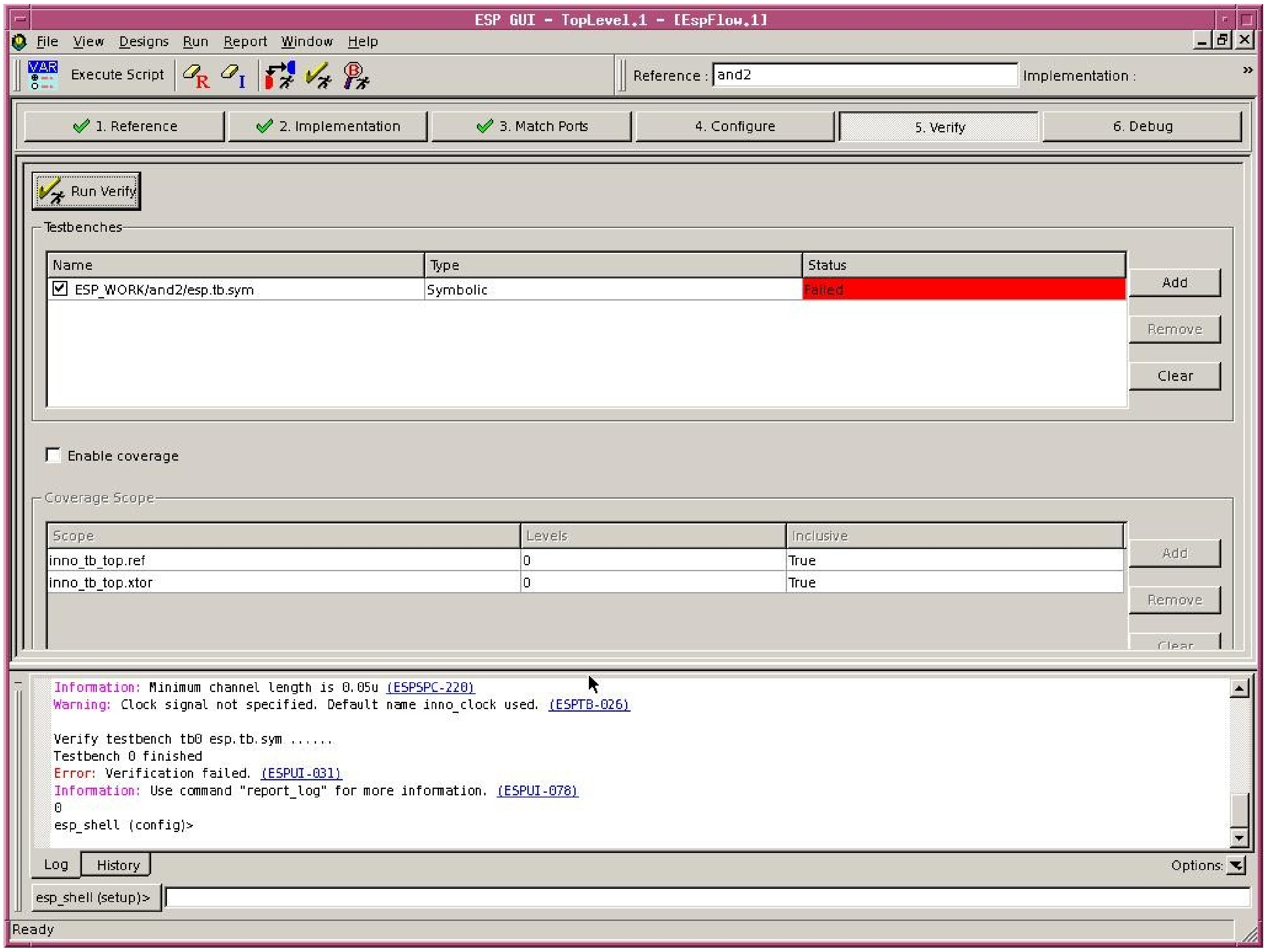
```
Warning: Identical top module names found between reference and implementation. Rename implementation to and2_xtor (ESPTB-011)
Information: Minimum channel width is 0.09u (ESPSPC-219)
Information: Minimum channel length is 0.05u (ESPSPC-220)
Warning: Clock signal not specified. Default name inno_clock used. (ESPTB-026)

Verify testbench tb0 esp.tb.sym ......
Testbench 0 finished
Information: Verification succeeded. (ESPUI-033)
1
esp_shell (verify)>
```

On the other hand, if the two designs are not equivalent, say you have replaced

```
assign  f=a&b;
```
with

```
assign  f=a|b;
```
in your 'and2.v'. Then you will see the red 'Failed'.



Name	Type	Status
☑ ESP_WORK/and2/esp.tb.sym	Symbolic	Failed

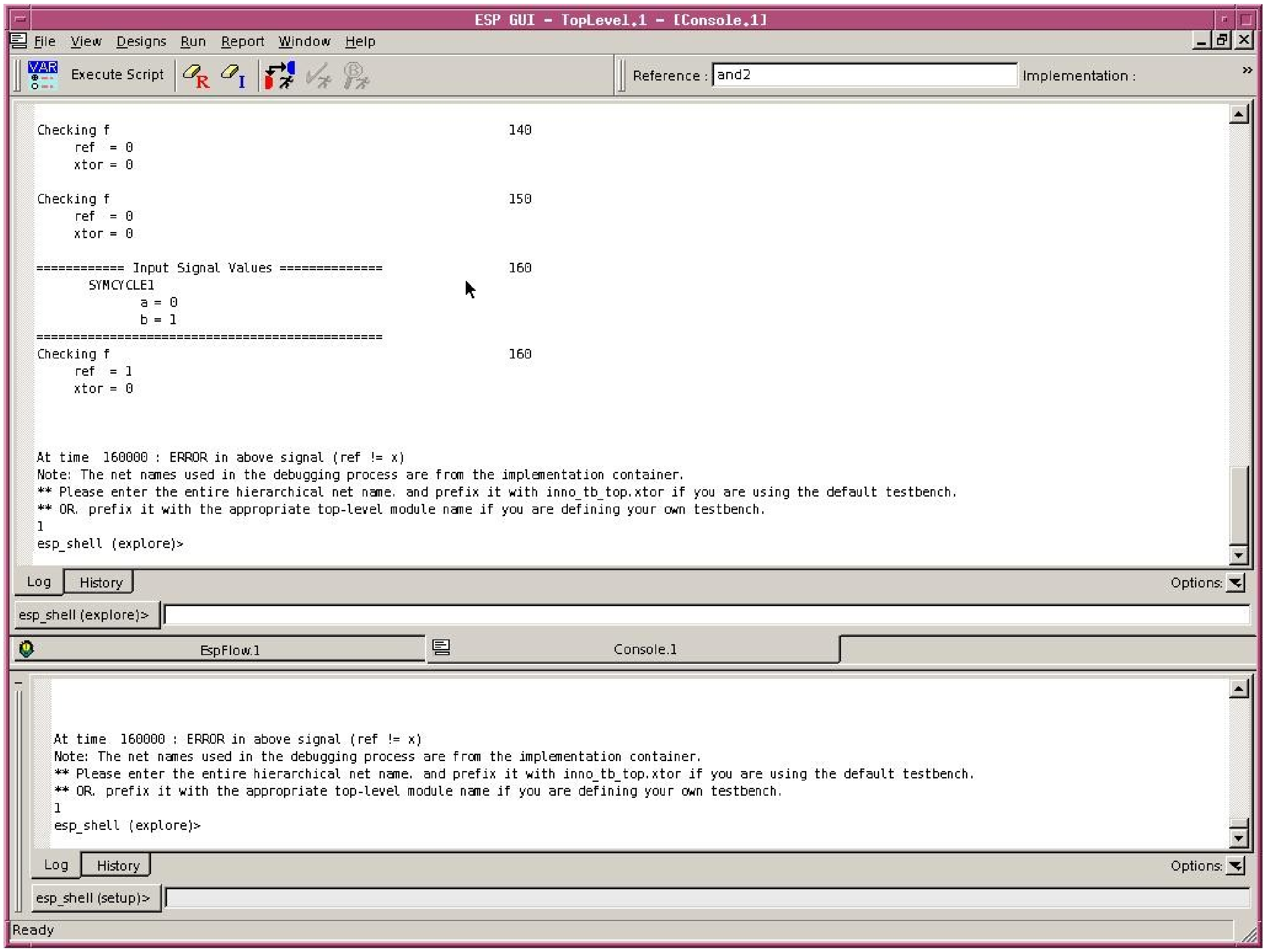
Scope	Levels	Inclusive
inno_tb_top.ref	0	True
inno_tb_top.xtor	0	True

```
Information: Minimum channel length is 0.05u (ESPSPC-220)
Warning: Clock signal not specified. Default name inno_clock used. (ESPTB-026)

Verify testbench tb0 esp.tb.sym ......
Testbench 0 finished
Error: Verification failed. (ESPUI-031)
Information: Use command "report_log" for more information. (ESPUI-078)
0
esp_shell (config)>
```

```
esp_shell (setup)>
```

In such case, you can proceed to '6. Debug' to see the problem. The tool will generate a set of inputs that make the two designs to output different outputs, which serve as both an evidence for the failed equivalence checking and a start-point for you to locate the issue in your designs.



```
Checking f                                          140
     ref  = 0
     xtor = 0

Checking f                                          150
     ref  = 0
     xtor = 0

============ Input Signal Values ==============      160
       SYMCYCLE1
              a = 0
              b = 1
==============================================
Checking f                                          160
     ref  = 1
     xtor = 0



At time  160000 : ERROR in above signal (ref != x)
Note: The net names used in the debugging process are from the implementation container.
** Please enter the entire hierarchical net name. and prefix it with inno_tb_top.xtor if you are using the default testbench.
** OR. prefix it with the appropriate top-level module name if you are defining your own testbench.
1
esp_shell (explore)>
```

```
At time  160000 : ERROR in above signal (ref != x)
Note: The net names used in the debugging process are from the implementation container.
** Please enter the entire hierarchical net name. and prefix it with inno_tb_top.xtor if you are using the default testbench.
** OR. prefix it with the appropriate top-level module name if you are defining your own testbench.
1
esp_shell (explore)>
```