ECE 429 - Tutorial IV: Standard Cell Based ASIC Design Flow

updated by Jia Wang, Aug. 2011, revised by Ken Choi, September 2015.

Overview

This tutorial introduces you to the standard cell based ASIC design flow using tools and libraries from various vendors. We will use the OSU standard cell library from FreePDK45 to implement an 8-bit accumulator design. We will first synthesize the design using the Synopsys Design Compiler and then perform place and route using the Cadence Encounter Digital Implementation System. The final layout will be painted in the Cadence Virtuoso platform and the final design will be verified by the Synopsys Formality equivalence checker.

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

Table of Contents

1. RTL Simulation

Create and initialize a directory for the project.

```
source /import/scripts/ece429_enc.cshrc
source /import/scripts/synopsys2012.cshrc
mkdir accu
cd accu
ece429-init-dir
```

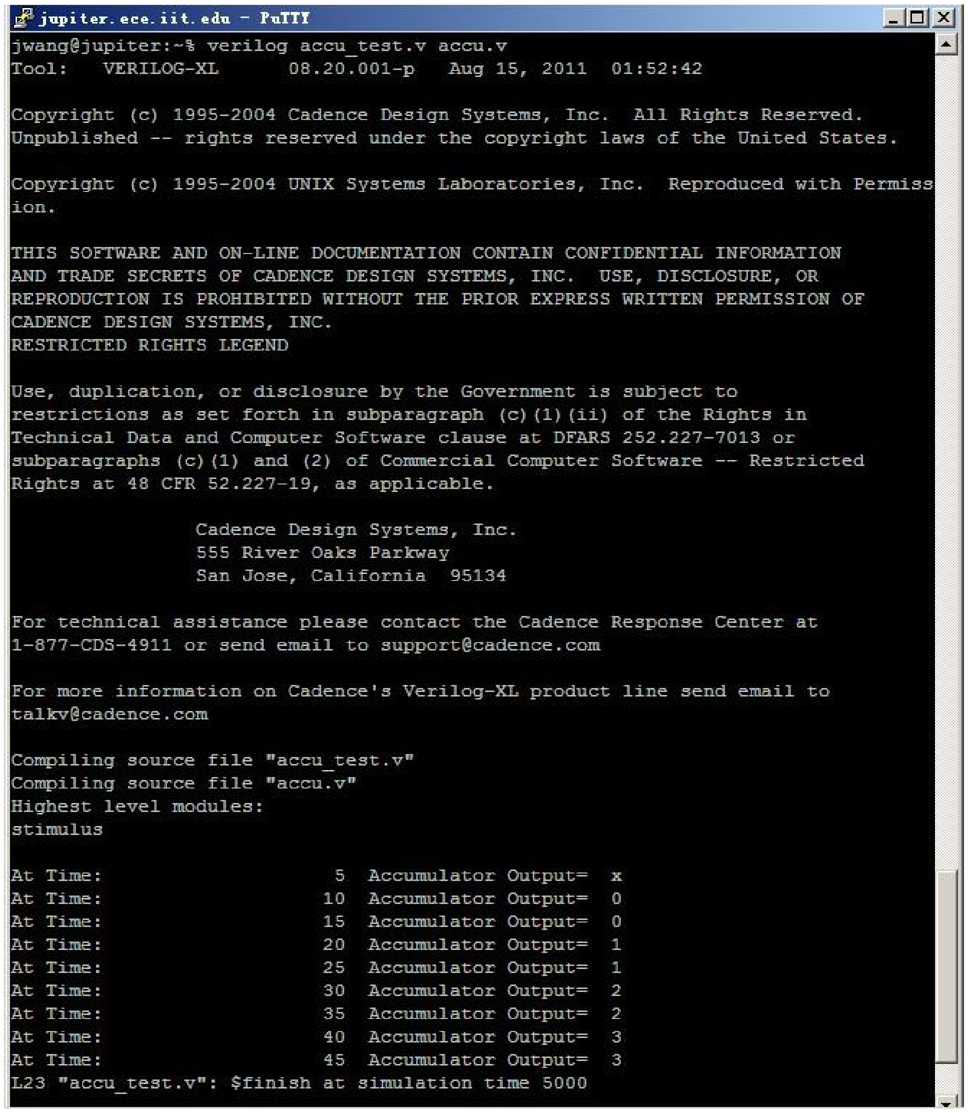
Typically you enter code in Verilog on the Register-Transfer Level (RTL), where you model your design using clocked registers, datapath elements and control elements. You will use Cadence Verilog-XL to simulate your design. You will also need to create a Verilog testbench for your circuit. In this tutorial there are 2 files as follows.

- accu.v: Verilog RTL code for an 8-bit accumulator
- accu_test.v: Verilog testbench for accu.v

In order to simulate Verilog code, use this command:

```
verilog accu_test.v accu.v
```

This testbench provides results directly on the screen and also in a waveform database. From the screen we can see that the design behaves as expected as follows.

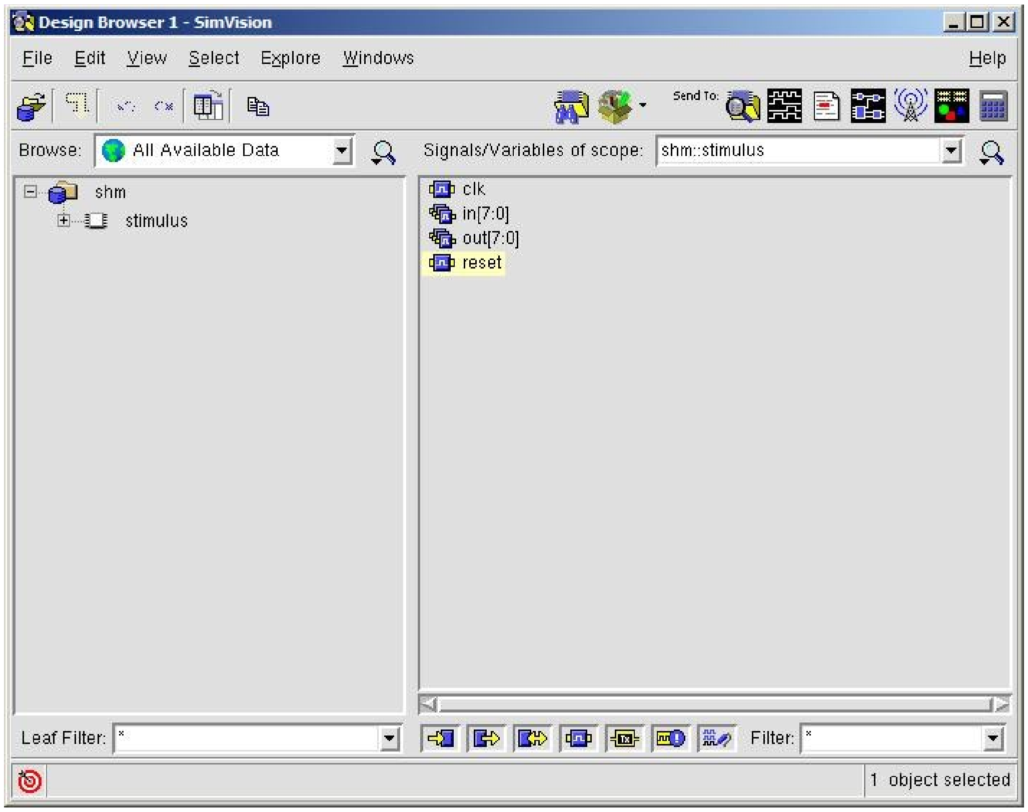


```
jwang@jupiter:~% verilog accu_test.v accu.v
Tool:   VERILOG-XL      08.20.001-p   Aug 15, 2011  01:52:42

Copyright (c) 1995-2004 Cadence Design Systems, Inc.  All Rights Reserved.
Unpublished -- rights reserved under the copyright laws of the United States.

Copyright (c) 1995-2004 UNIX Systems Laboratories, Inc.  Reproduced with Permiss
ion.

THIS SOFTWARE AND ON-LINE DOCUMENTATION CONTAIN CONFIDENTIAL INFORMATION
AND TRADE SECRETS OF CADENCE DESIGN SYSTEMS, INC.  USE, DISCLOSURE, OR
REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF
CADENCE DESIGN SYSTEMS, INC.
RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to
restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in
Technical Data and Computer Software clause at DFARS 252.227-7013 or
subparagraphs (c)(1) and (2) of Commercial Computer Software -- Restricted
Rights at 48 CFR 52.227-19, as applicable.

                Cadence Design Systems, Inc.
                555 River Oaks Parkway
                San Jose, California  95134

For technical assistance please contact the Cadence Response Center at
1-877-CDS-4911 or send email to support@cadence.com

For more information on Cadence's Verilog-XL product line send email to
talkv@cadence.com

Compiling source file "accu_test.v"
Compiling source file "accu.v"
Highest level modules:
stimulus

At Time:                    5  Accumulator Output=  x
At Time:                   10  Accumulator Output=  0
At Time:                   15  Accumulator Output=  0
At Time:                   20  Accumulator Output=  1
At Time:                   25  Accumulator Output=  1
At Time:                   30  Accumulator Output=  2
At Time:                   35  Accumulator Output=  2
At Time:                   40  Accumulator Output=  3
At Time:                   45  Accumulator Output=  3
L23 "accu_test.v": $finish at simulation time 5000
```

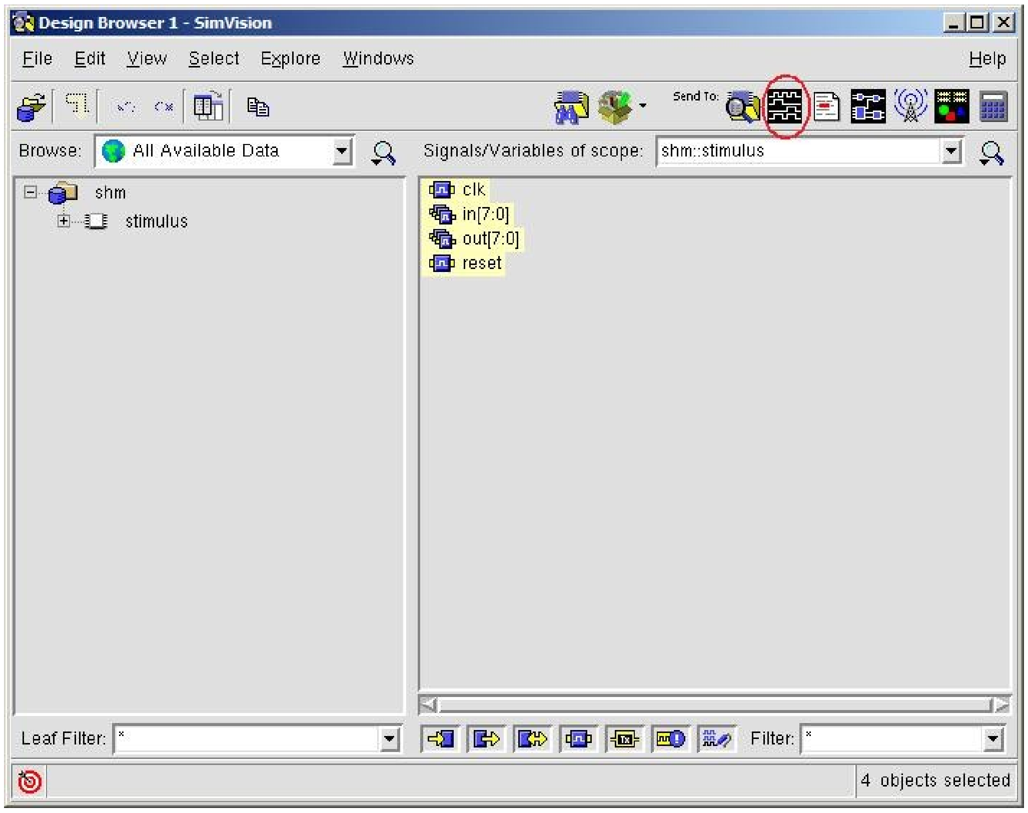
That is, every 10ns we add 1 to the accumulator. This is expected since in the testbench a clock of 10ns is specified and the input 'in' is connected to a constant 1.

We use the program Cadence SimVision to look at the waveform database that was created by Verilog-XL. Type the following command:
simvision

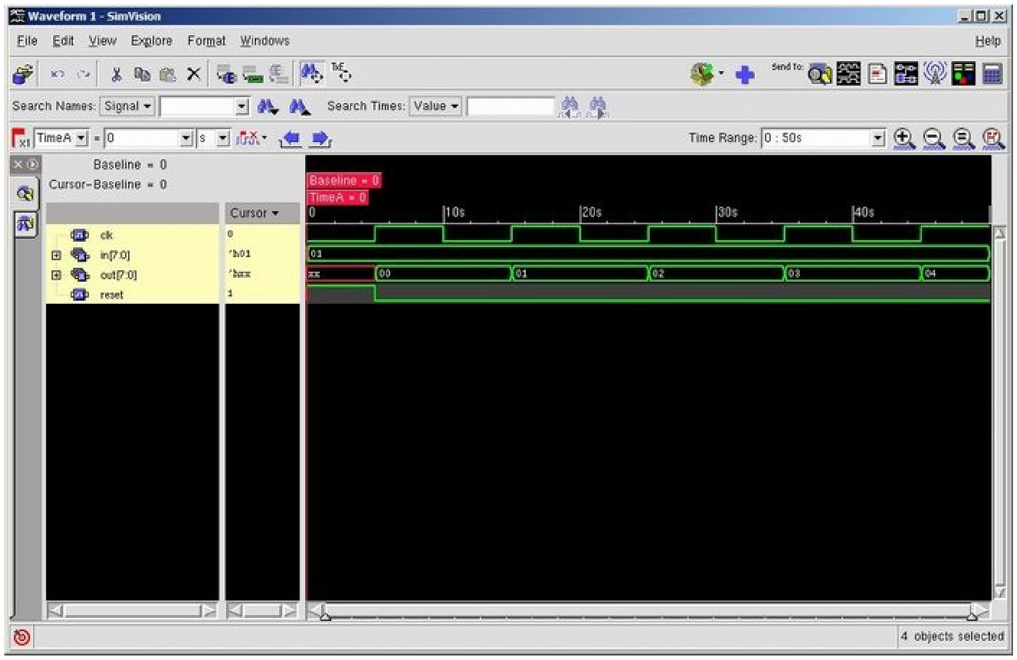
Now we need to open the Waveform database. Click on the "Open" symbol, choose the directory "shm.db", which is where the file is located, and double-click on the file "shm.trn" to open it. To see the contents of the waveform database, click on "stimulus":


Design Browser 1 - SimVision
File Edit View Select Explore Windows Help
Send To:
Browse: All Available Data
Signals/Variables of scope: shm::stimulus
shm
stimulus
clk
in[7:0]
out[7:0]
reset
Leaf Filter: *
Filter: *
1 object selected

Now we want to plot our waveforms. We need to select which signals we are interested in. In this case lets look at all waveforms. Select all 4 waveforms on the right and hit the picture-button with the square waves.


Design Browser 1 - SimVision
File Edit View Select Explore Windows Help
Send To:
Browse: All Available Data
Signals/Variables of scope: shm::stimulus
shm
stimulus
clk
in[7:0]
out[7:0]
reset
Leaf Filter: *
Filter: *
4 objects selected

We see that the circuit works fine. At every rising clock edge the output changes.



2. Logic Synthesis using Design Compiler

Once you have verified that your Verilog RTL code is working correctly you can synthesize it into standard cells. The result will be a gate-level netlist that only contains interconnected standard cells.

There are template files for all the following steps already prepared for you. We will now copy those templates into our project.
cp /import/scripts/FreePDK45_2011/FreePDK45/osu_soc/flow_ece429/* .

We will use the Synopsys Design Compiler for logic synthesis. Since a hardware design requires not only the Verilog descriptions but also the specifications, we will use a script file to automate the synthesis task. The template file is provided as 'compile_dc.tcl'. Note that dc stands for Design Compiler (DC).

Please open 'compile_dc.tcl' in a text editor. Although you don't need to modify this file for this tutorial, you will need to modify it for the final project so please read the following part carefully. To make it easier to modify the file, all key values are defined in the beginning of the file.

The first few lines will show

```
#/* All verilog files, separated by  spaces          */
set my_verilog_files [list  accu.v]
```

This specifies what Verilog files should be included for our design. It is not necessary to include the testbench as it is not part of the chip.

Then you will see the name of the module that represents the whole design.

```
#/* Top-level  Module                              */
set my_toplevel  accu
```

The name of the clock signal is specified next. It's usually 'clk'.

```
#/* The name of the clock pin. If no  clock-pin     */
#/* exists, pick  anything                          */
set my_clock_pin  clk
```

Finally, you specify the desired clock frequency of the circuit.

```
#/* Target frequency in MHz for  optimization       */
set my_clk_freq_MHz  1000
```

Once you have the script file ready, you can go ahead to synthesize the circuit:
dc_shell -f compile_dc.tcl

Design Compiler will run for a short time and create substantial amounts of output. When it is finished it will return to the command line. If there is an error it will specify the exact source of the error and the line number in the script that was responsible for the error. Typically there will be no errors. We can verify the initial estimations of area, timing, and power by reading into the following three files 'cell.rep', 'timing.rep', and 'power.rep'.

We can also look at the output of DC. As we said above, it is a gate-level Verilog netlist that only contains interconnected standard cells. The netlist is called 'accu.vh' and you can use any text editor to check its content. Note that the top-level module still has the name 'accu' and the names of the inputs and outputs have not changed. From the outside it is exactly the same circuit as you coded on the RTL level. But on the inside all functionality is now expressed only in terms of standard cells. A post-synthesis simulation can be performed by including Verilog models of the standard cells available from 'gscl45nm.v'. The

command line is
verilog gscl45nm.v accu_test.v accu.vh
Note how we re-used the original testbench from the RTL level simulation. That is an excellent way to ensure that the gate-level representation matches the RTL level. The simulation results should look similar to before.

---

3. Place and Route using Encounter

Since we have layouts for all standard cells, we can now place and route them for the final layout. We first use Encounter to place the standard cells and to route the interconnects. The output will be the placement of the cells and the metal interconnects. Further processings are required to obtain the final layout for fabrication.

Similar to logic synthesis using DC, we will use a set of script files to automate the placement and route task. All the commands for placement and route are in the template file 'encounter.tcl'. It includes commands to input the gate-level netlist, floorplan the chip, place the cells, route the cells, and to verify and output the result in GDSII format, which is the most popular format for IC layouts. The circuit to be placed and routed is specified in the file 'encounter.conf'.

Similar to 'compile_dc.tcl', although you don't need to modify 'encounter.conf' for this tutorial, you will need to modify it for the final project.

The only place you need to change are the lines

```
# Specify the name of your toplevel module
set my_toplevel  accu
```
This variable 'my_toplevel' will be used to deduct various file names. So it should be the same as in 'compile_dc.tcl'.

Now we are ready to run Encounter. The command line is
encounter -init encounter.tcl

Encounter will run for a while and create substantial amounts of output. We can obtain more accurate timing by reading into the file 'timing.rep.5.final'.

The outputs of Encounter include a GDSII stream 'final.gds2' and an equivalent Verilog model 'final.v' of the placed circuit. Similar to 'accu.vh', the top-level module in 'final.v' still has the name 'accu' and the names of the inputs and the outputs are not changed. From the outside it is exactly the same circuit as you coded on the RTL level. But on the inside additional buffers are introduced to reduce interconnect delays and enhance clocking robustness. A post-P&R simulation can be performed with:
verilog gscl45nm.v accu_test.v final.v
The simulation results should look similar to before.

Please type "**reportGateCount -limit 0**" in the encounter shell to get the area report of your design. For example, we can get the following output for the accu design:

Gate area 2.8158 um^2
[0] accu Gates=56 Cells=30 Area=158.6 um^2

"Gate area 2.8158 um^2" states the area of the reference gate.
"Gates=56" is the equivalent gate count, computed by round(158.6/2.8158).
"Cells=30" is the actual number of gates and cells, e.g. inverters, NAND gates, flip-flops, etc.
"Area=158.6 um^2" is the area.

To get the power report, please type "**report_power**" in the encounter shell. (The unit is mW.)”

---

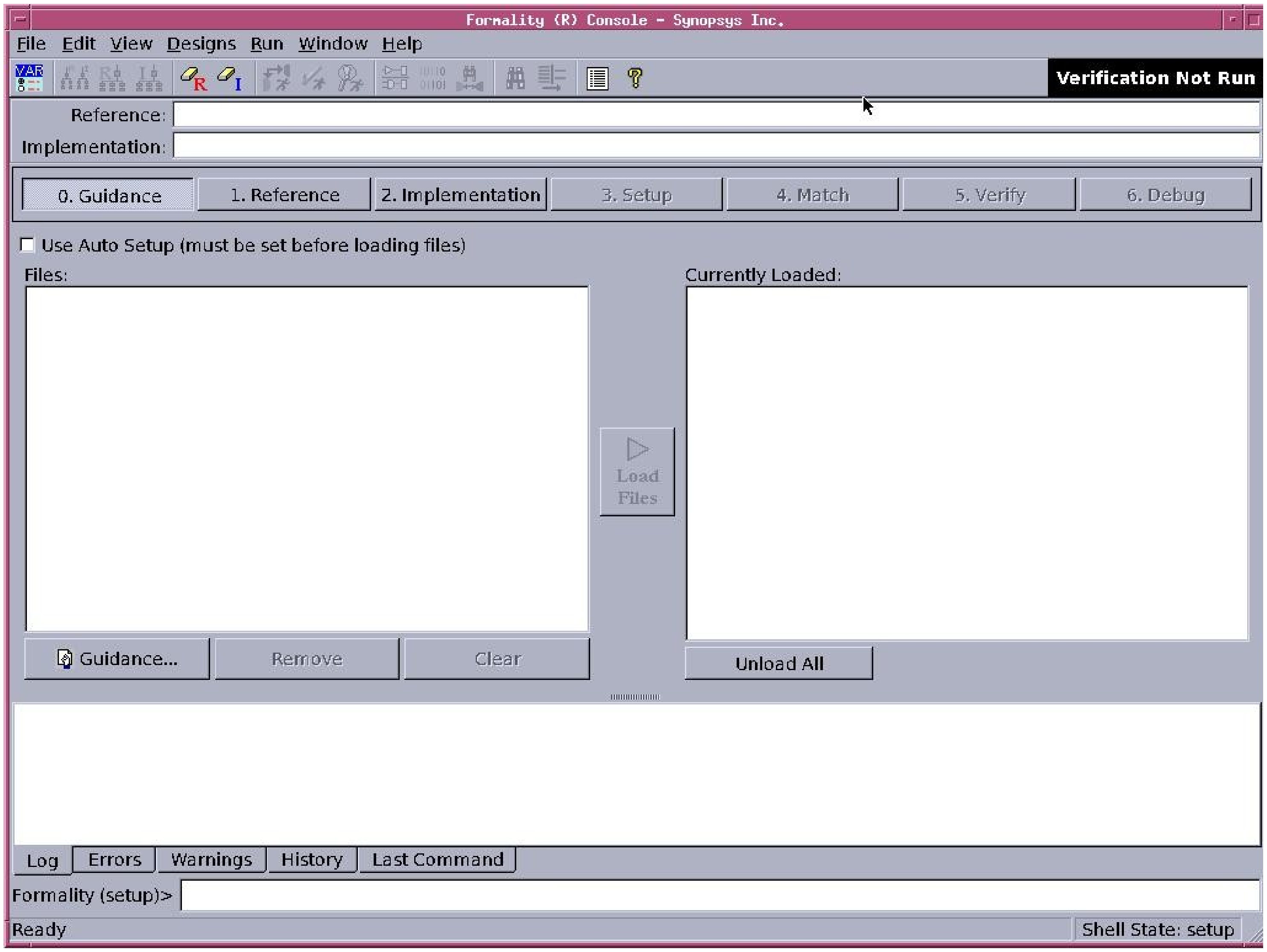
4. Equivalence Checking using Formality

While post-synthesis and post-P&R simulations may help you detect errors during logic synthesis and place and route due to either bugs in the EDA tools or careless manual modifications, they cannot guarantee to find all errors if there is any. We will again rely on equivalence checking to formally prove the correctness of logic synthesis and place and route steps.

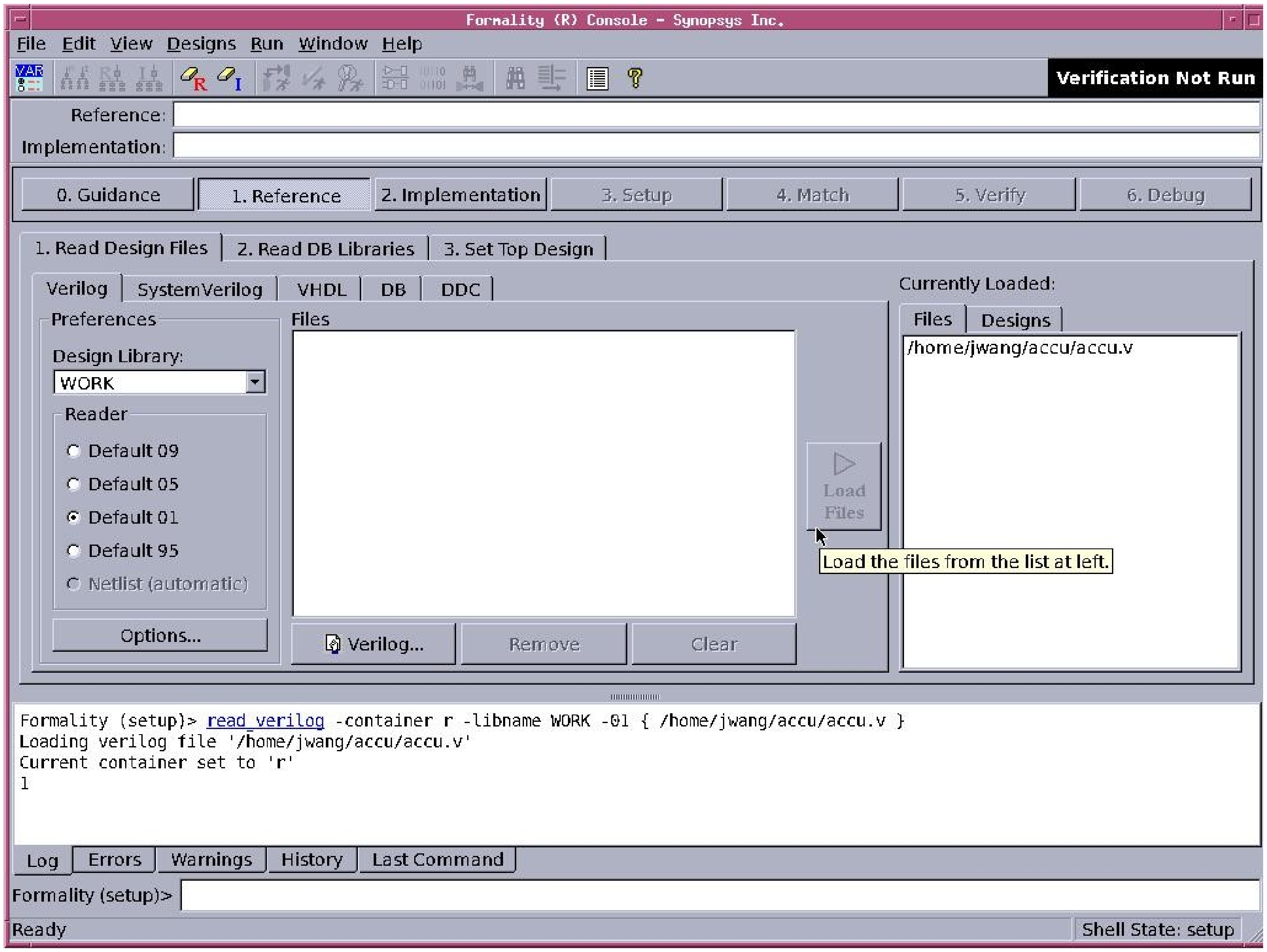
Since both logic synthesis and place and route output circuits in Verilog, we cannot use Formality ESP as introduced in Tutorial III since it validates a SPICE netlist against a Verilog model. Actually, Formality ESP is an extension to Synopsys Formality that validates two Verilog models. Therefore, we will use Formality directly for this tutorial. Note that although we have three Verilog models of the design -- 'accu.v', 'accu.vh', and 'final.v', we only need to validate 'accu.v' against 'final.v' since 'accu.vh' can be treated as an intermediate result.

Start Formality:
fm_shell -gui

You will see a guided interface to perform equivalence checking that is very similar to Formality ESP.



Skip the step '0. Guidance'. For the step '1. Reference', click 'Verilog...' in '1. Read Design Files' to choose 'accu.v' and then 'Load Files' to load it.



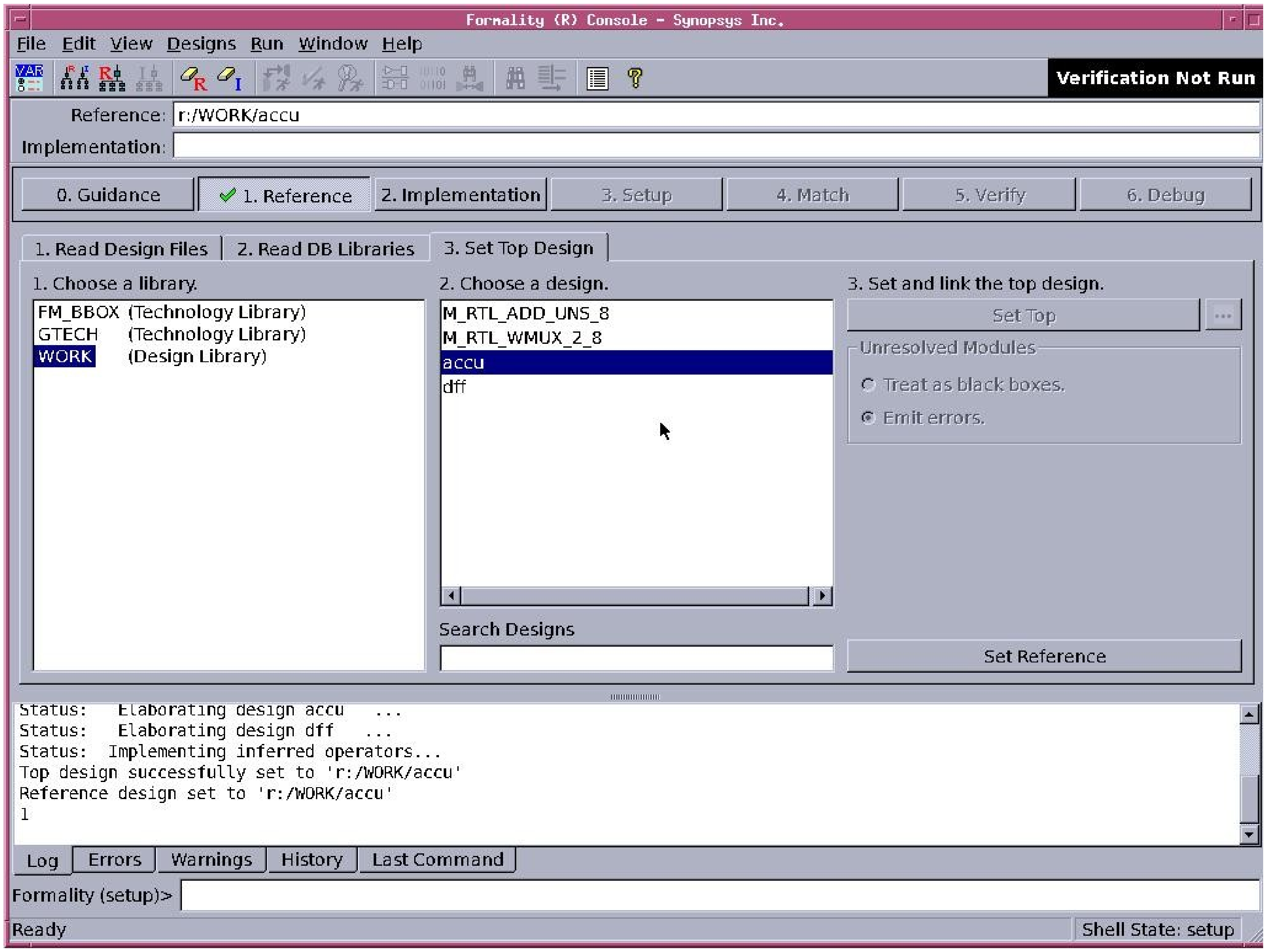
Reader: Default 09 [✓ Default 01] Default 05 Default 95 Netlist (automatic)

/home/jwang/accu/accu.v

```
Formality (setup)> read_verilog -container r -libname WORK -01 { /home/jwang/accu/accu.v }
Loading verilog file '/home/jwang/accu/accu.v'
Current container set to 'r'
1
```

Formality (setup)>

Click '3. Set Top Design', choose 'accu', and then click 'Set Top'. You will see the green check to the side of '1. Reference' indicating the step is completed correctly.



Reference: r:/WORK/accu
Implementation:

1. Choose a library.

FM_BBOX	(Technology Library)
GTECH	(Technology Library)
WORK	(Design Library)

2. Choose a design.

- M_RTL_ADD_UNS_8
- M_RTL_WMUX_2_8
- accu
- dff

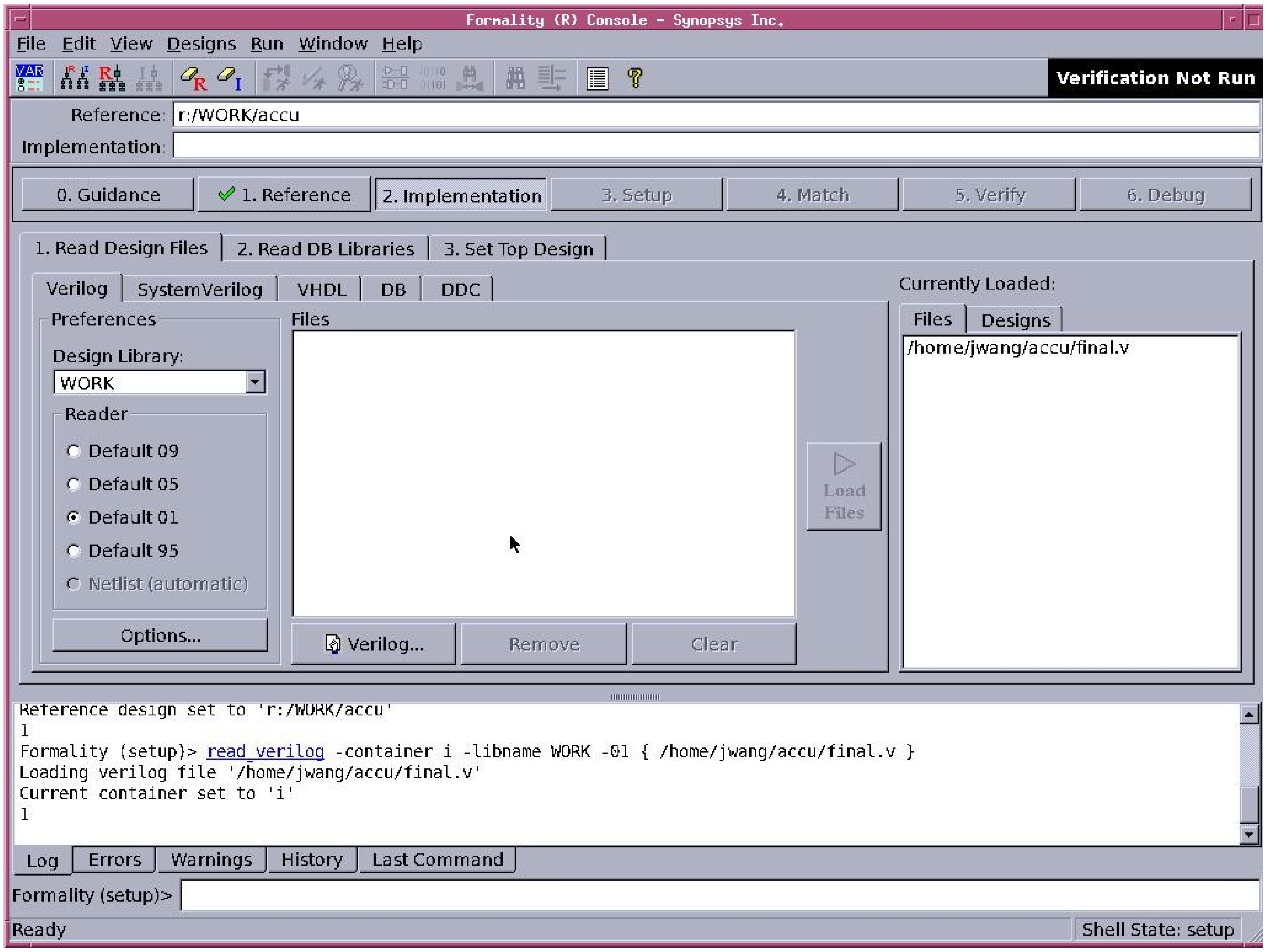
3. Set and link the top design.

Unresolved Modules: Treat as black boxes. [✓ Emit errors.]

```
Status:   Elaborating design accu   ...
Status:   Elaborating design dff   ...
Status:  Implementing inferred operators...
Top design successfully set to 'r:/WORK/accu'
Reference design set to 'r:/WORK/accu'
1
```

Formality (setup)>

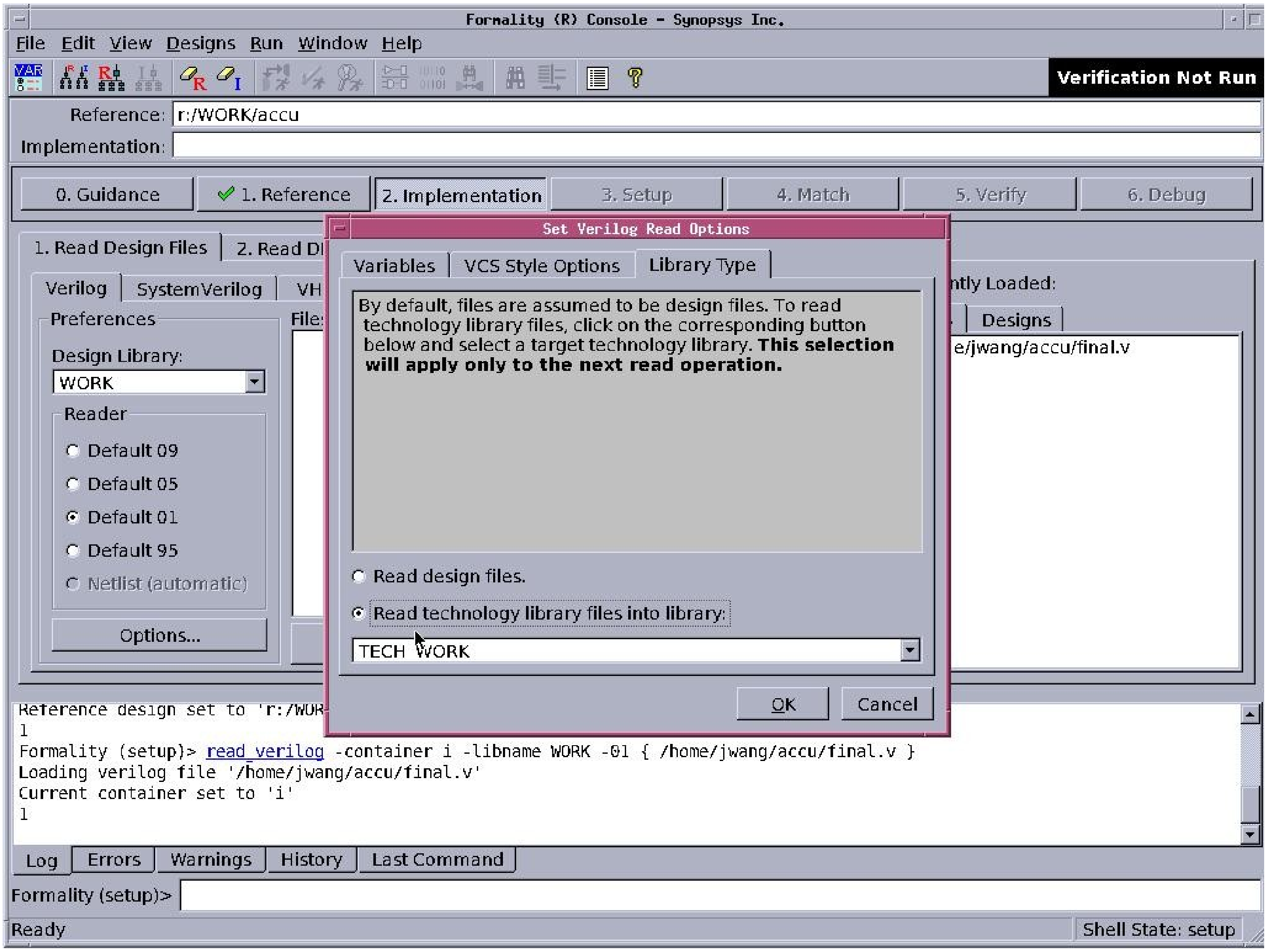
Move to the step '2. Implementation'. Choose and load 'final.v'.



Reader: Default 09 [✓ Default 01] Default 05 Default 95 Netlist (automatic)

```
Reference design set to 'r:/WORK/accu'
1
Formality (setup)> read_verilog -container i -libname WORK -01 { /home/jwang/accu/final.v }
Loading verilog file '/home/jwang/accu/final.v'
Current container set to 'i'
1
```

Now, since 'final.v' uses the cells from the standard cell library, we need to let Formality know the functionality of the cells. This is achieved by load 'gscl45nm.v' as a technology library. Click the 'Options' button to bring up the Set Verilog Read Options dialog box. Switch to the 'Library Type' tab and choose 'Read technology library files into library'.



Reference: r:/WORK/accu

Implementation:

Design Library: WORK

Reader: Default 09 [✓ Default 01] Default 05 Default 95 Netlist (automatic)

By default, files are assumed to be design files. To read technology library files, click on the corresponding button below and select a target technology library. **This selection will apply only to the next read operation.**

Read design files. [✓ Read technology library files into library:]

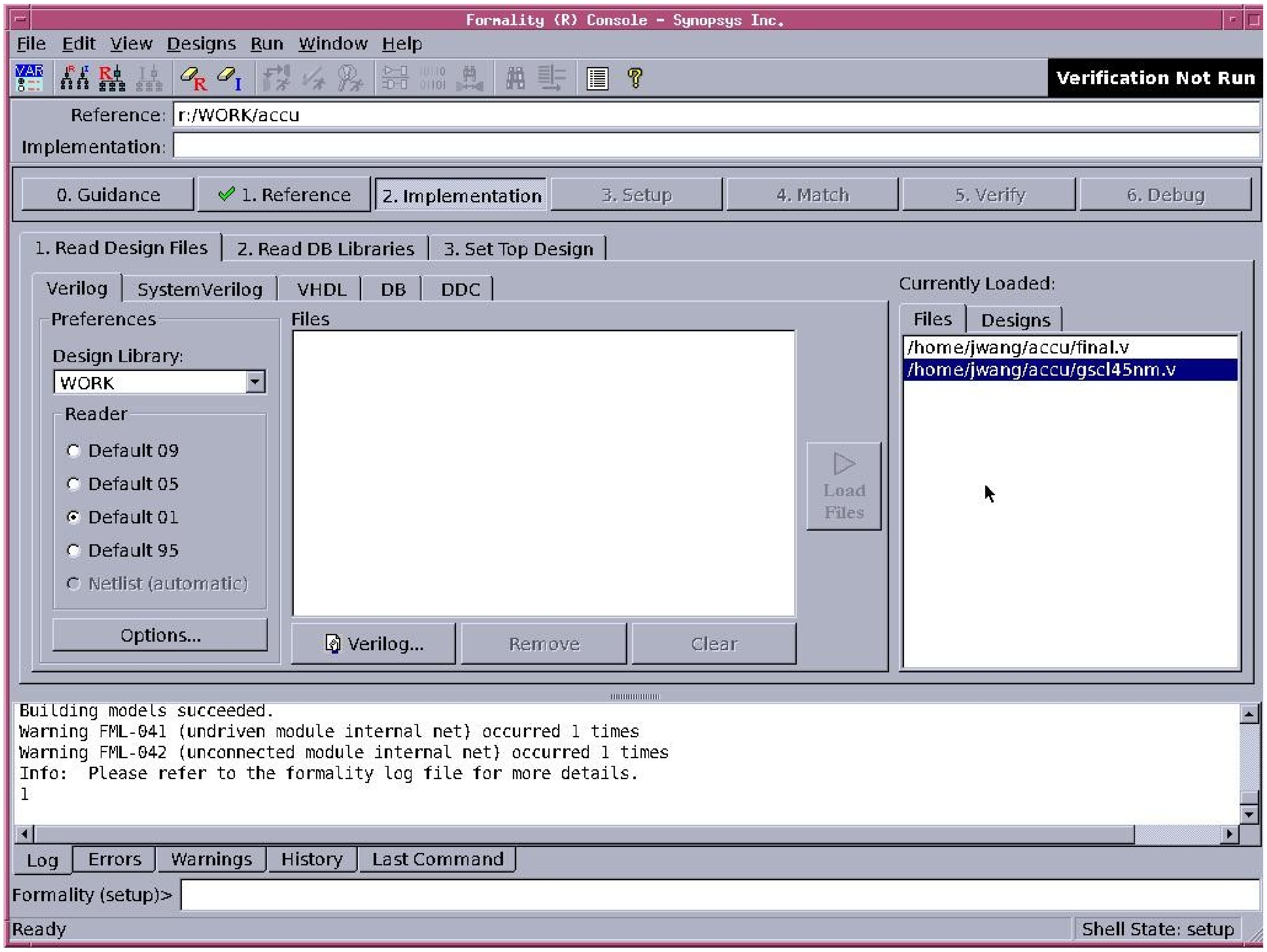
TECH WORK

e/jwang/accu/final.v

```
Reference design set to 'r:/WOR
1
Formality (setup)> read_verilog -container i -libname WORK -01 { /home/jwang/accu/final.v }
Loading verilog file '/home/jwang/accu/final.v'
Current container set to 'i'
1
```

Formality (setup)>

Click 'OK'. Choose and load 'gscl45nm.v'.



Reference: r:/WORK/accu

Implementation:

Design Library: WORK

Reader: ( ) Default 09 ( ) Default 05 [✓ Default 01] ( ) Default 95 ( ) Netlist (automatic)

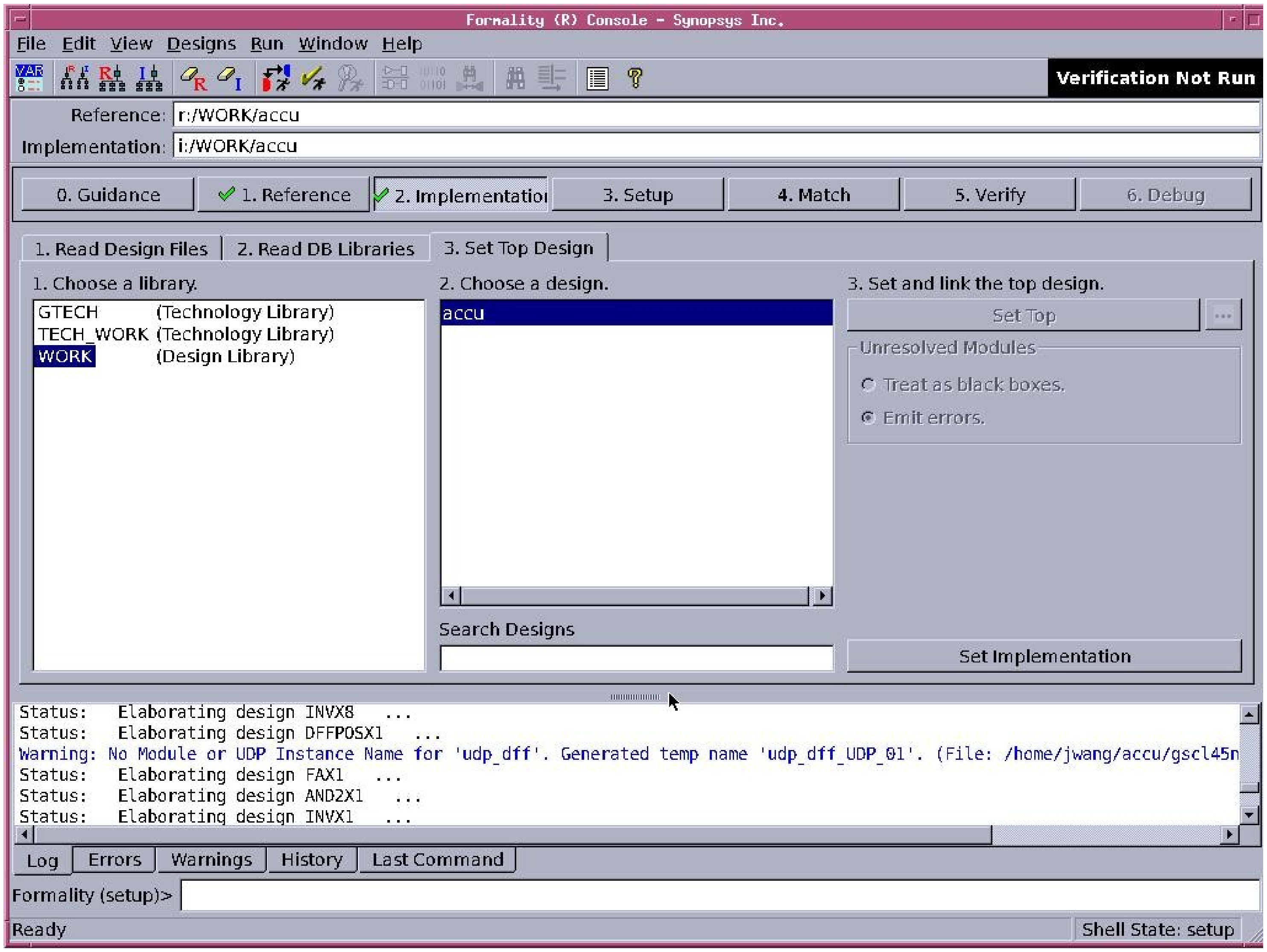
/home/jwang/accu/final.v
/home/jwang/accu/gscl45nm.v

```
Building models succeeded.
Warning FML-041 (undriven module internal net) occurred 1 times
Warning FML-042 (unconnected module internal net) occurred 1 times
Info:  Please refer to the formality log file for more details.
1
```

Formality (setup)>

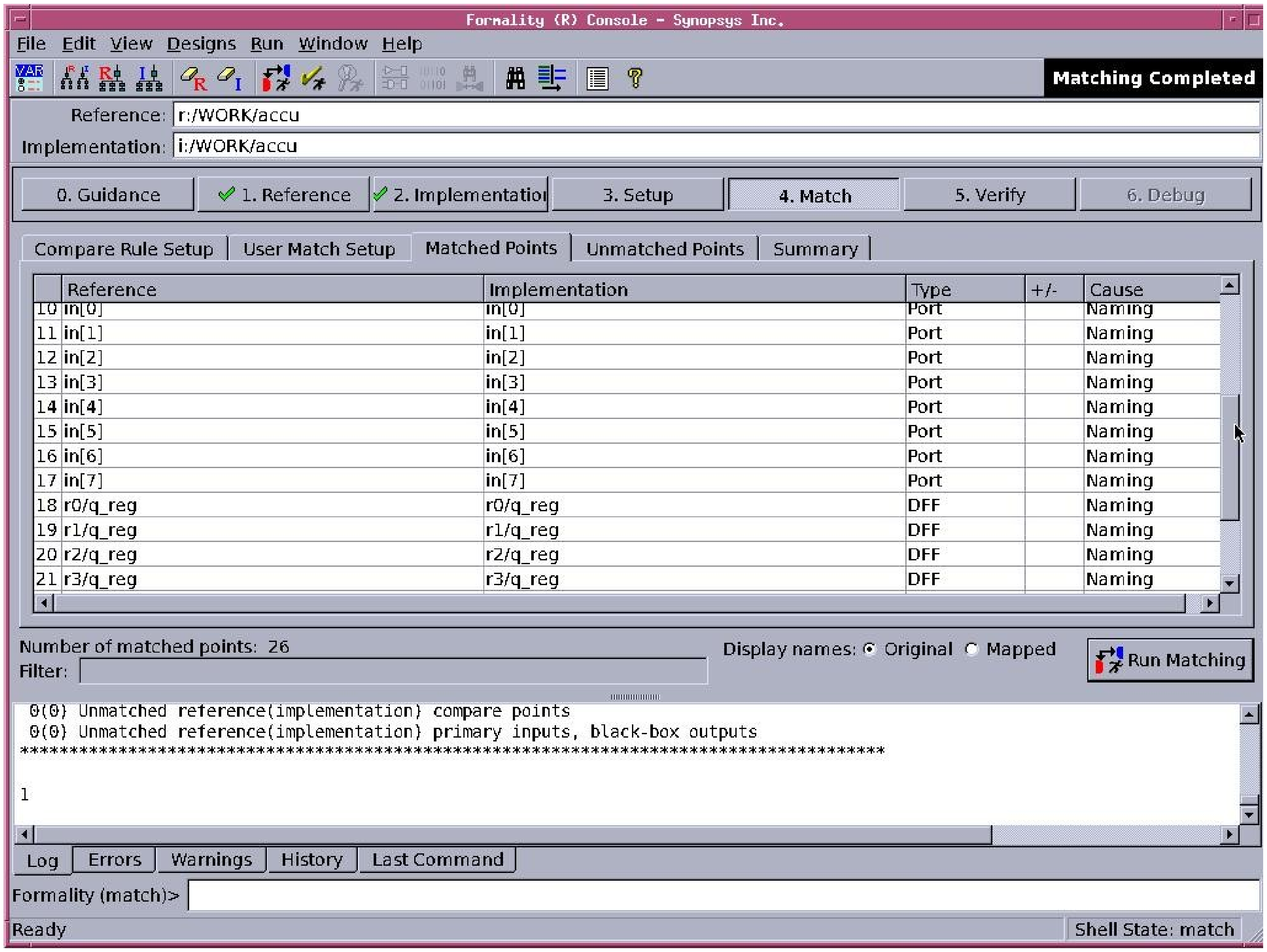
Ready   Shell State: setup

Click '3. Set Top Design', choose 'accu', and then click 'Set Top'.



```
Status:   Elaborating design INVX8   ...
Status:   Elaborating design DFFPOSX1   ...
Warning: No Module or UDP Instance Name for 'udp_dff'. Generated temp name 'udp_dff_UDP_01'. (File: /home/jwang/accu/gscl45n
Status:   Elaborating design FAX1   ...
Status:   Elaborating design AND2X1   ...
Status:   Elaborating design INVX1   ...
```

Skip '3. Setup'. For '4. Match', . Click 'Run Matching' to match the inputs/outputs/DFFs of the two designs. All of them should be matched by their names.



Reference: r:/WORK/accu
Implementation: i:/WORK/accu

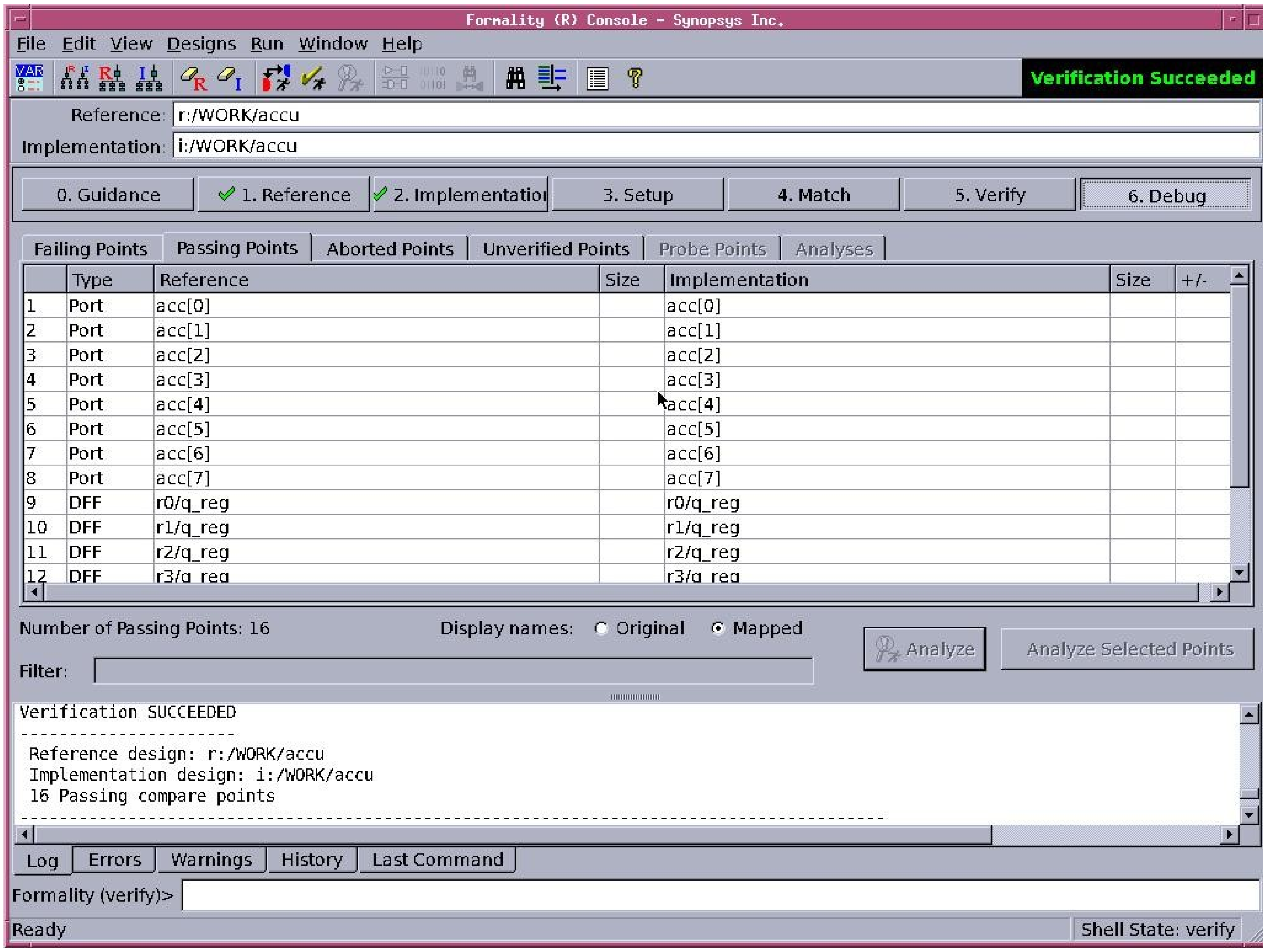
	Reference	Implementation	Type	+/-	Cause
10	in[0]	in[0]	Port		Naming
11	in[1]	in[1]	Port		Naming
12	in[2]	in[2]	Port		Naming
13	in[3]	in[3]	Port		Naming
14	in[4]	in[4]	Port		Naming
15	in[5]	in[5]	Port		Naming
16	in[6]	in[6]	Port		Naming
17	in[7]	in[7]	Port		Naming
18	r0/q_reg	r0/q_reg	DFF		Naming
19	r1/q_reg	r1/q_reg	DFF		Naming
20	r2/q_reg	r2/q_reg	DFF		Naming
21	r3/q_reg	r3/q_reg	DFF		Naming

Number of matched points: 26
Filter:
Display names: [✓ Original] Mapped

```
 0(0) Unmatched reference(implementation) compare points
 0(0) Unmatched reference(implementation) primary inputs, black-box outputs
*******************************************************************************************

1
```

Formality (match)>
Ready
Shell State: match

Move to the step '5. Verify'. Click 'Verify'. If the two designs are the same, you will see a dialog box showing 'Verification succeeded!'. Then '6. Debug' will show all output ports and DFFs as 'Passing Points'.



Reference: r:/WORK/accu

Implementation: i:/WORK/accu

	Type	Reference	Size	Implementation	Size	+/-
1	Port	acc[0]		acc[0]		
2	Port	acc[1]		acc[1]		
3	Port	acc[2]		acc[2]		
4	Port	acc[3]		acc[3]		
5	Port	acc[4]		acc[4]		
6	Port	acc[5]		acc[5]		
7	Port	acc[6]		acc[6]		
8	Port	acc[7]		acc[7]		
9	DFF	r0/q_reg		r0/q_reg		
10	DFF	r1/q_reg		r1/q_reg		
11	DFF	r2/q_reg		r2/q_reg		
12	DFF	r3/q_reg		r3/q_reg		

Number of Passing Points: 16

Display names: Original [✓ Mapped]

Filter:

```
Verification SUCCEEDED
----------------------
 Reference design: r:/WORK/accu
 Implementation design: i:/WORK/accu
 16 Passing compare points
-----------------------------------------------------------------------------------------
```

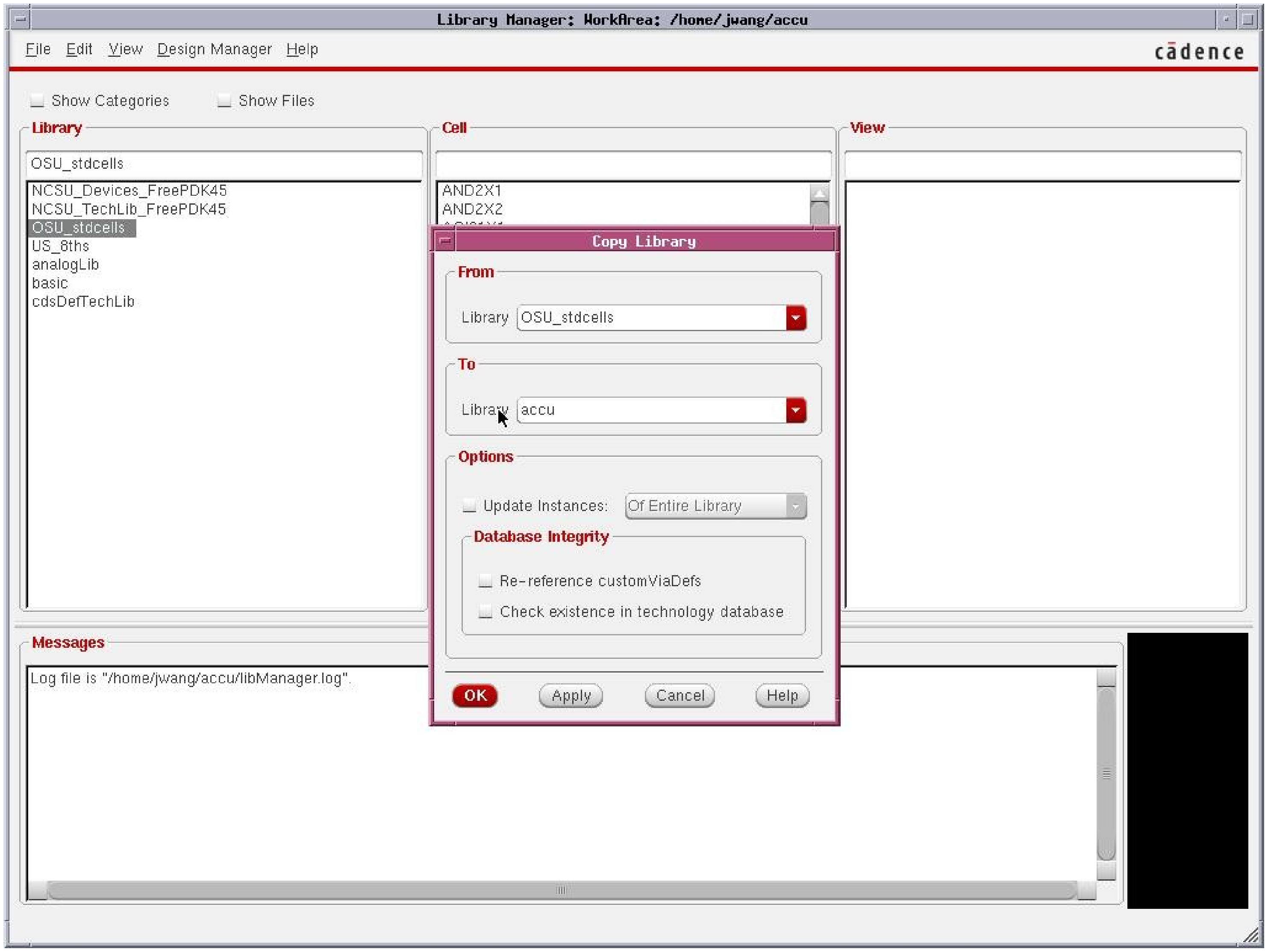
Formality (verify)>

Have a try to modify the functionality of 'accu.v' and see how Formality can detect the discrepancies.

5. Layout Printing using Virtuoso

The GDSII stream 'final.gds2' contains only the placement of the cells and the metal interconnects. It doesn't contain any layout details inside the cells, e.g. wells and poly. Therefore, we need to combine this stream with cell layouts to obtain the final layout for fabrication.

We will use the Virtuoso platform for this task. Start Virtuoso in the directory 'accu'. Select the library 'OSU_stdcells' in the library manager. It contains all the cell layouts we are going to use. Click **Edit→Copy** to copy the cells in 'OSU_stdcells' to a new library 'accu'.


Library Manager: WorkArea: /home/jwang/accu

Library: OSU_stdcells

Library list: NCSU_Devices_FreePDK45, NCSU_TechLib_FreePDK45, OSU_stdcells, US_8ths, analogLib, basic, cdsDefTechLib

Cell list: AND2X1, AND2X2

Copy Library

From — Library: OSU_stdcells

To — Library: accu

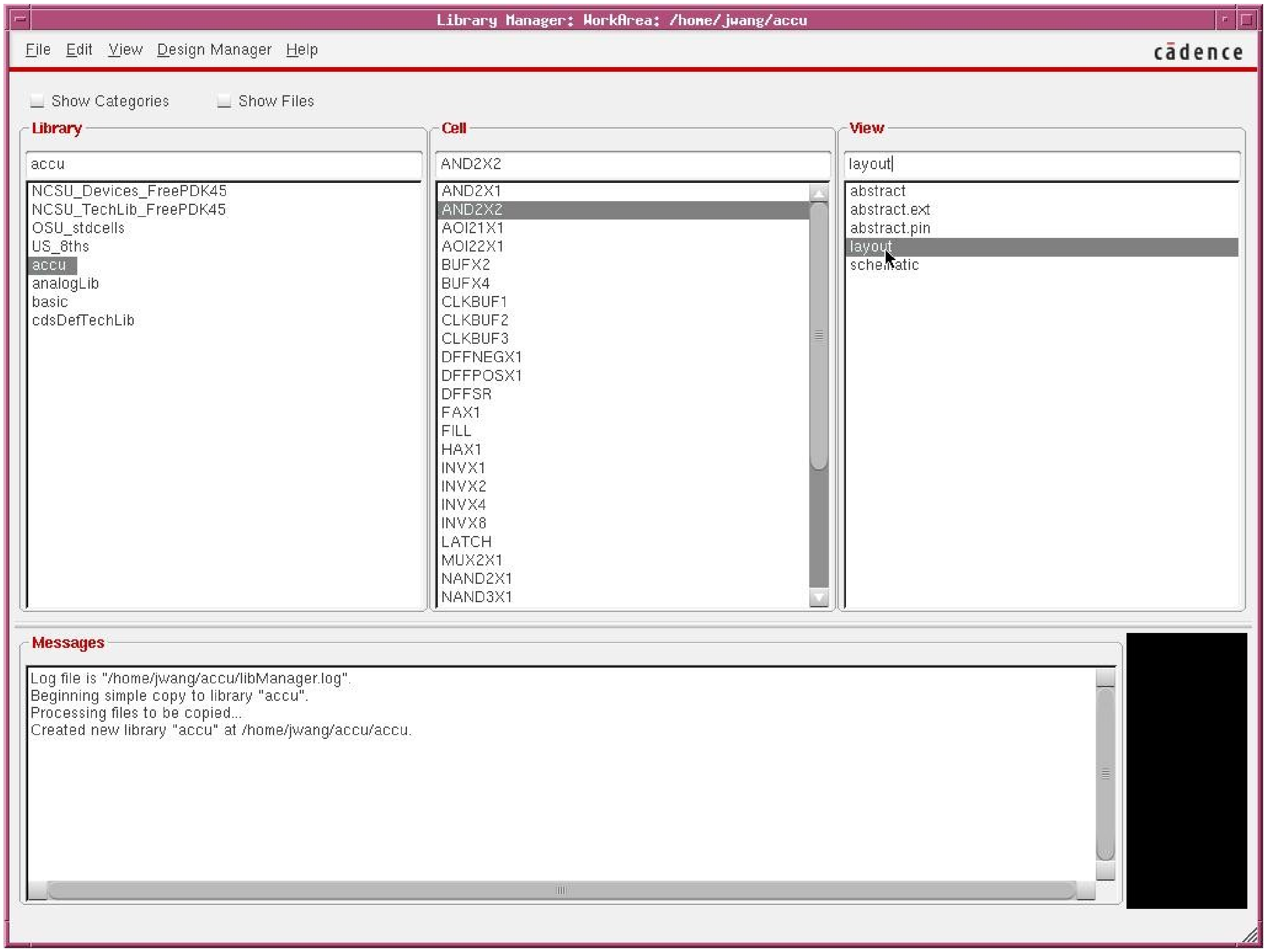
Options — Update Instances: Of Entire Library

Database Integrity — Re-reference customViaDefs; Check existence in technology database

Messages:

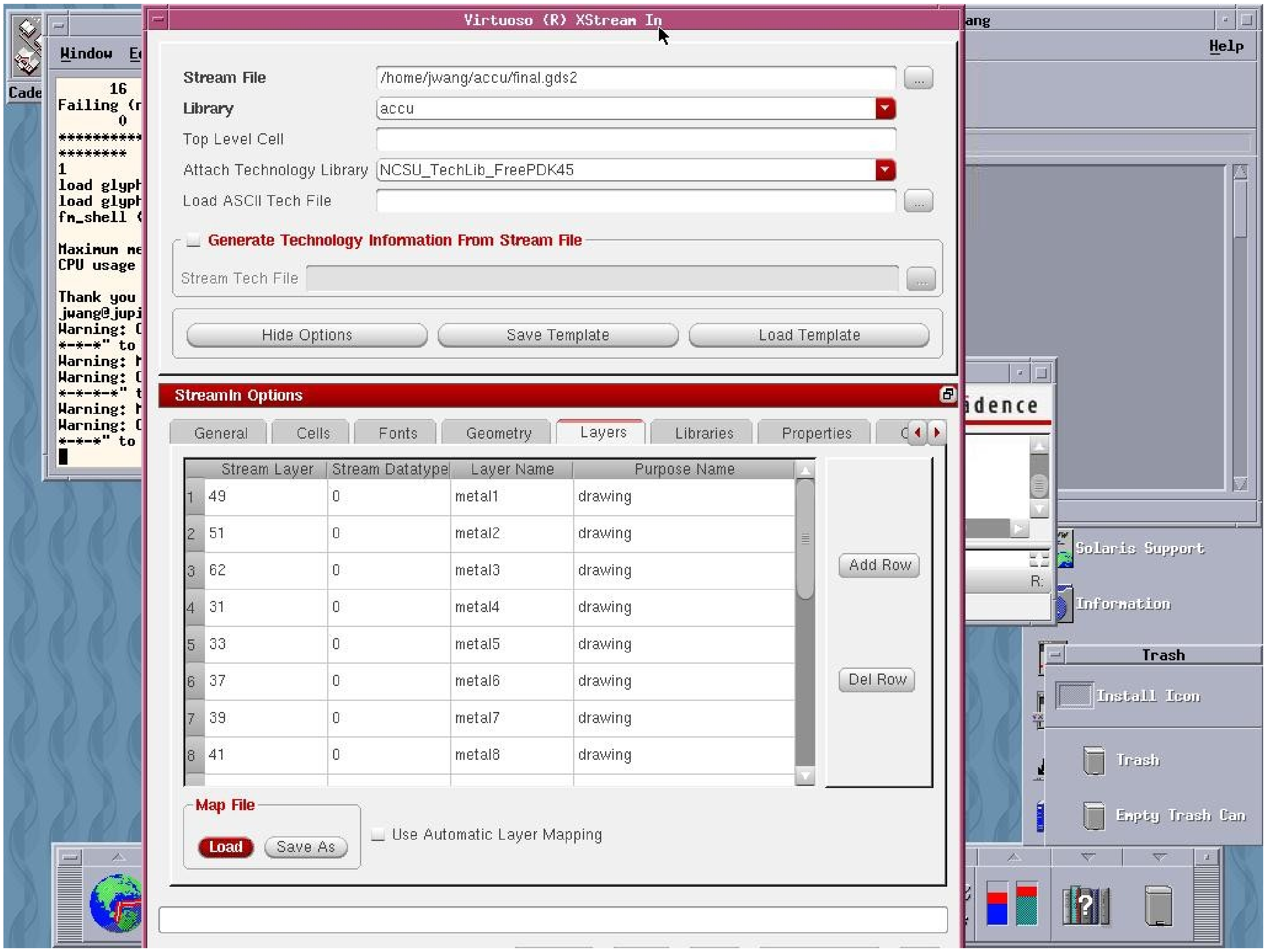
```
Log file is "/home/jwang/accu/libManager.log".
```

Click 'OK' when asked to confirm the location of the library 'accu'. Then you should be able to see the cells are copied to 'accu' in the library manager.



```
Log file is "/home/jwang/accu/libManager.log".
Beginning simple copy to library "accu".
Processing files to be copied...
Created new library "accu" at /home/jwang/accu/accu.
```

Now, bring up the Virtuoso Log window and click **File→Import→Stream**. You should see the Virtuoso XStream In dialog box. Click 'Show Options' to show 'StreamIn Options'. At the top section, select 'final.gds2' for 'Stream File', 'accu' for 'Library', and 'NCSU_TechLib_FreePDK45' for 'Attach Technology Library'. For 'StreamIn Options', switch to the 'Layers' tab. Load 'gds2_virtuoso.map' as the 'Map File', which translates GDSII layers into a human readable format.


Virtuoso (R) XStream In

Stream File	/home/jwang/accu/final.gds2
Library	accu
Top Level Cell	
Attach Technology Library	NCSU_TechLib_FreePDK45
Load ASCII Tech File	

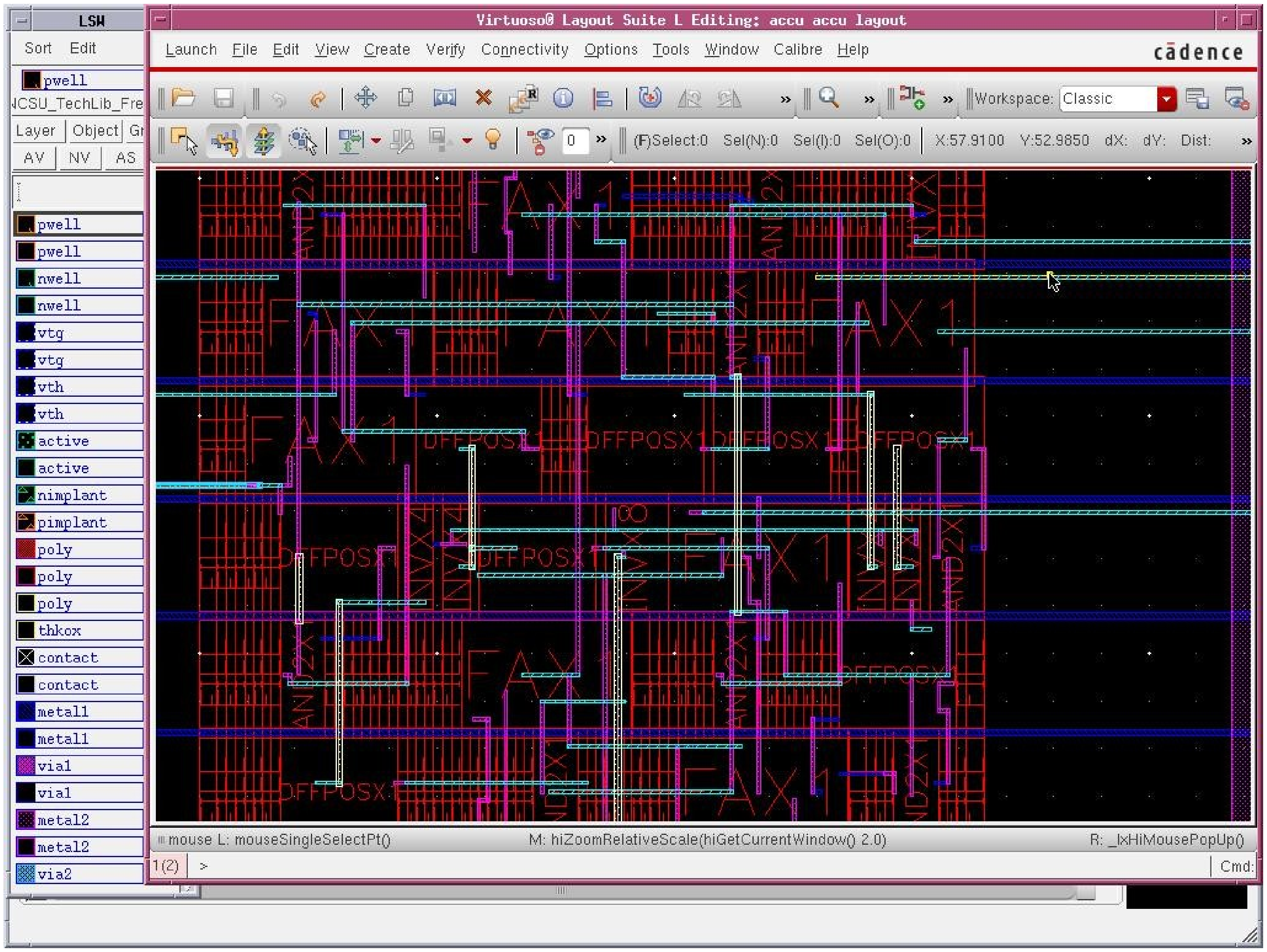
Generate Technology Information From Stream File

Stream Tech File

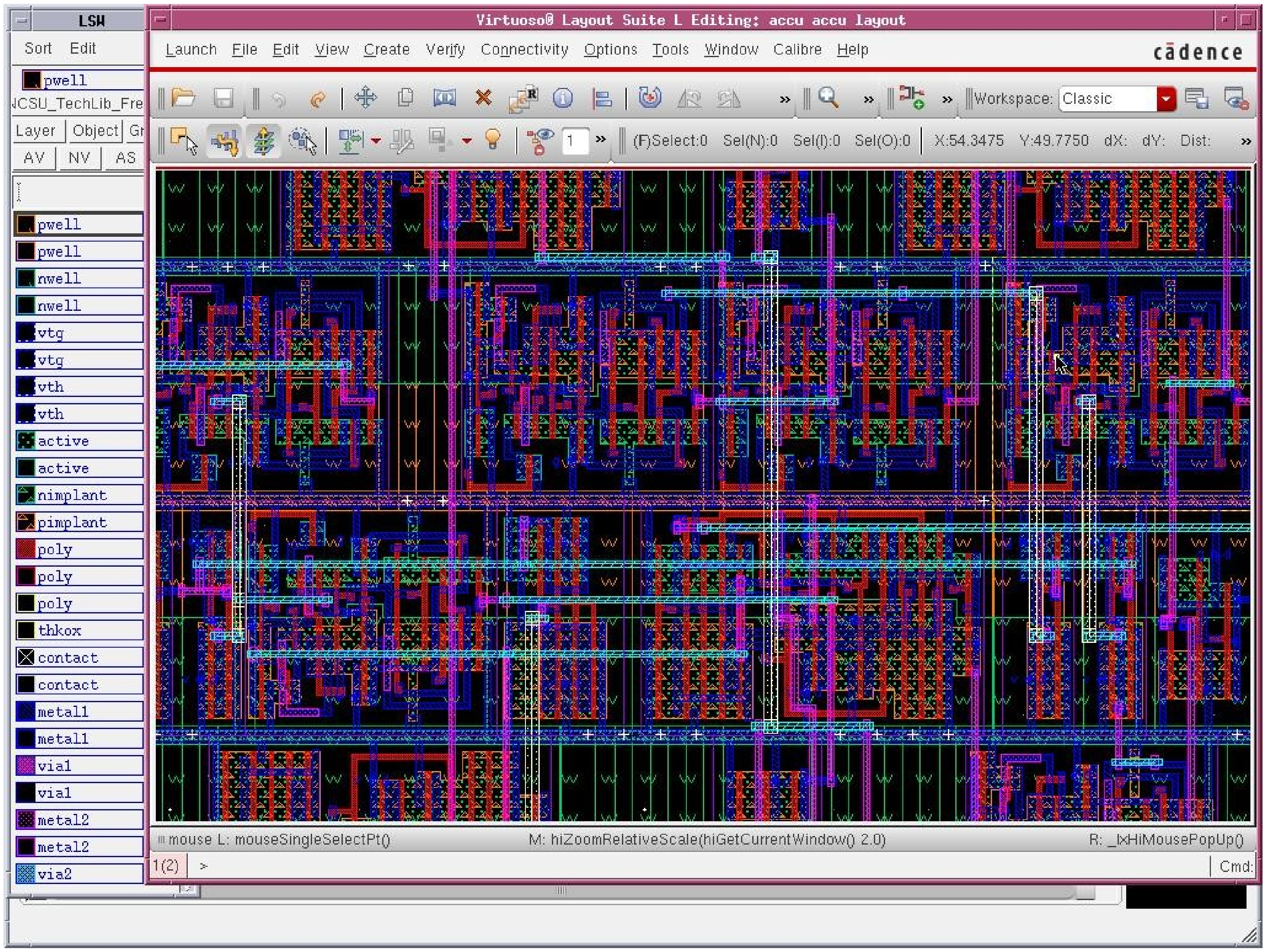
StreamIn Options — Layers

	Stream Layer	Stream Datatype	Layer Name	Purpose Name
1	49	0	metal1	drawing
2	51	0	metal2	drawing
3	62	0	metal3	drawing
4	31	0	metal4	drawing
5	33	0	metal5	drawing
6	37	0	metal6	drawing
7	39	0	metal7	drawing
8	41	0	metal8	drawing

'Hide Options' and then click 'Translate' to import the stream. When it finishes, there should be no error. In the library manager, you should see a new cell 'accu' in the library 'accu' with a single 'layout' view. Open it and zoom in. You will see the placed cells and the metal interconnects in the design.


LSW
Sort Edit
pwell
NCSU_TechLib_Fre
Layer Object Gr
AV NV AS
pwell
pwell
nwell
nwell
vtg
vtg
vth
vth
active
active
nimplant
pimplant
poly
poly
poly
thkox
contact
contact
metal1
metal1
via1
via1
metal2
metal2
via2
Virtuoso® Layout Suite L Editing: accu accu layout
Launch File Edit View Create Verify Connectivity Options Tools Window Calibre Help
cādence
Workspace: Classic
(F)Select:0 Sel(N):0 Sel(I):0 Sel(O):0 X:57.9100 Y:52.9850 dX: dY: Dist:
FAX1
DFFPOSX1
AND2X1
INVX4
mouse L: mouseSingleSelectPt() M: hiZoomRelativeScale(hiGetCurrentWindow() 2.0) R: _lxHiMousePopUp()
1(2) > Cmd:

Change the 'Display Stop Level' to 1. The final layout is shown as follows.


LSW
Sort Edit
pwell
NCSU_TechLib_Fre
Layer Object Gr
AV NV AS
pwell
pwell
nwell
nwell
vtg
vtg
vth
vth
active
active
nimplant
pimplant
poly
poly
poly
thkox
contact
contact
metal1
metal1
via1
via1
metal2
metal2
via2
Virtuoso® Layout Suite L Editing: accu accu layout
Launch File Edit View Create Verify Connectivity Options Tools Window Calibre Help
cādence
Workspace: Classic
(F)Select:0 Sel(N):0 Sel(I):0 Sel(O):0 X:54.3475 Y:49.7750 dX: dY: Dist:
mouse L: mouseSingleSelectPt()
M: hiZoomRelativeScale(hiGetCurrentWindow() 2.0)
R: _lxHiMousePopUp()
1(2) >
Cmd: