EESM518/ ELEC516 Design Automation Tutorial

Teaching Assistant: Michael Ling (EESM518) , Toby Qian (ELEC516)

23/02/2010

Objective

After this tutorial, you should gain essential understandings about ASIC design flow. Two popular tools for logic synthesis and auto placement/route would be introduced. A lab material is also supplied to help you understand the flow.

Background on ASIC design flow

Front End and Back End design are the most popular division between ASIC flows. Front end is more concerned with design logic while Back end is more related to the physical aspect of the design. The whole process is shown in Fig. 1.

Front End Design

Each design process begins with a specification. The function of a system is specified and partitioned into submodels or systems to improve the efficiency and robustness. After specification. each module is designed to meet its own function. In digital domain, this design is usually modeled in a hardware description language (i.e., VHDL, verilog). Behavior simulation is devised to check the design function.

After the desired behavior is verified, we move to a stage called logic synthesis. In a fully custom design, this stage may be completed by human beings. Here, we only refer standard-cell based design (a semi-custom design strategy) where this step is accomplished by computer tools. Several key transformations are made here:

1. The design logic function is extracted from the HDL for each input and output. The result logic is then presented in basic Boolean equation;
2. All the extracted logic function is simplified to achieve better area/ power/ timing;
3. All the logic is then mapping into specific cells provided in the standard-cell library. This library is usually provided by foundry;
4. The timing/area/power is estimated and optimized to meet certain design constraints;

The output from logic synthesis is a netlist describing the interconnection of library gates. You will see it soon. Then you can utilize this netlist and the library gate model to device a pre-layout simulation which would include the cell delay into consideration and make the result more realistic. If this simulation passes, we hand over the netlist to the back end guys.

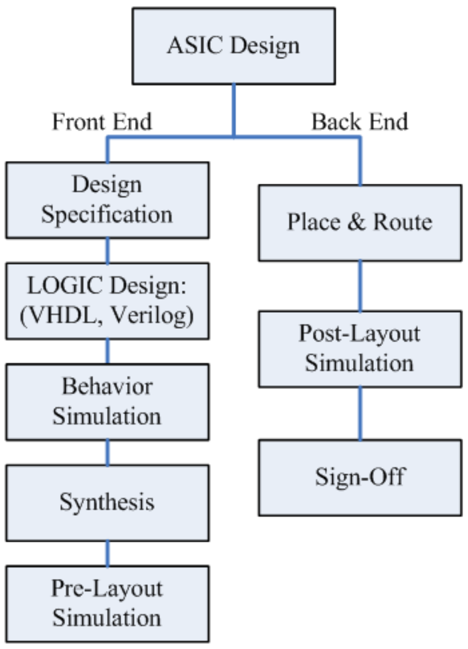


Figure 1: ASIC Design Flow

Back End Design

The back end process focuses on turning the netlist into a physical layout. This involves cell placement and interconnection routing (for signals). Good cell placement and interconnection routing tools would result in a significant better design in terms of area/power/timing. So several iterations of optimization would be performed here to achieve this goal. Usually back end design requires an understanding of tools and real-life experiences.

Note on Design Presentations

Probability a better way to see this process is to take a look at your design presentations:

1. You write a code in VHDL/Verilog;
2. Simulate your design in behavior simulation; We call it behavior simulation in a sense that at this stage, we do not have any information about the physical implementation of your design. Only the logic function is verified. There's no timing information in it.

3. Synthesis your design and get your design netlist; This stage your design has been mapped into structure connections with standard-cells. Standard-cell library provides a forest of logic cells for you to implement all possible logic functions (actually, a 2-input nand gate can implement all possible logic); This netlist file is usually in Verilog format;
4. With the netlist file, you can do Pre-Layout simulation with a standard library verilog file. The standard library file is only to describe cell functions and their delay information. This time, the simulation result is more realistic in the sense that all the cell/gate delays in your design is included. However, since we still haven't actually layout the cell, we do not know the wire delay (we do not know the distance between to cells, so their connecting wire lengths);
5. Placement & Route your design. This time, you get a physical design. You can extract the wire delay information and another netlist (since during this step, some logic might be revised for better timing);
6. With the wire delay information and the final netlist, you can redo your simulation (Post-Layout simulation). If it passes, then congratulations! You probability have a functional design (But function is only one concern, power/area are also important).

Tutorial Setup

In this tutorial, we would use a simple 16-bit multiplier design to demonstrate the ASIC flow. A template folder "elec516_lab" (or eesm518_lab) contains all the necessary file and setup information for you to practice and use for your project. (See "Before Getting Started" section to download this folder)

The sub-folder "encounter" is used when you to Back-End place and route. There are two files in it.

- The "LEF" file contains the placement information for each standard cell (i.e., size, dimensions, input/output locations);
- The "lib" file is the timing library containing the cell and wire delay information;

You are encouraged to open these two files with a text editor to get a sense.

The "synopsys" folder contains synthesis and simulation setup.

- "db" folder is for you to store your synthesis results. it is short for your design library database;
- "libs" folder contains standard cell library ("db" file) and symbol library ("sdb" file) for synthesis use;
- "sim" folder is for your behavior simulation;
- "syn-sim" folder is for your gate-level (pre-layout/post-layout) simulation. There're verilog files of the standard cell library inside. Since after synthesis, your design is present in standard cells. So this verilog file contains the cell function and delays for you to simulate your design in a more practical situation;
- "syn" folder for you to synthesize your design. You must do it in this folder since it contains a setup file ".synopsys_dc_setup" (hidden, use ls -a to see it);
- "verilog" folder for you to put your source code;

Before Getting Start

Download the tempelate folder :

>cp ~qianzl/public/elec516-lab . (copy the folder to your current working directory)

In order to set up the environment for the tools (so that the system can recognize the licenses and versions for sysnopsys and cadence tools) , in “elec516_lab” (or “ eesm518_lab”) folder,we need to source “.cshrc_user” file first .

```
> source .cshrc_user
```

After this step , you can type commands like

```
> which design_vision
```

to check the current version of the tools which we’ll use.

Note: in many cases , if the system tells you “command not found” , it’s most likely you haven’t “source .cshrc_user” yet or the previous source action is no longer valid after reboot. You can source .cshrc_user file again to fix this problem.

Behavior Simulation (For VHDL input)

1) Go to the “elec516_lab/synopsys/sim” folder .

2) In “elec516_lab/synopsys/sim” directory execute the following commands to analyze VHDL source code . If multiple VHDL file need to be analyzed, the bottom level module file is expected to be analyzed before top module file:

```
> vhdlan -nc ../vhdl/ARITH_module.vhd
> vhdlan -nc ../vhdl/tb_multiplier.vhd
>vhdlan -nc ../vhdl/cfg_array_mult.vhd
```

please check and fix any error according to the the information given after execute this command

3) Execute the command below to compile and elaborate the design

```
>scs -nc cfg_array_mult
```

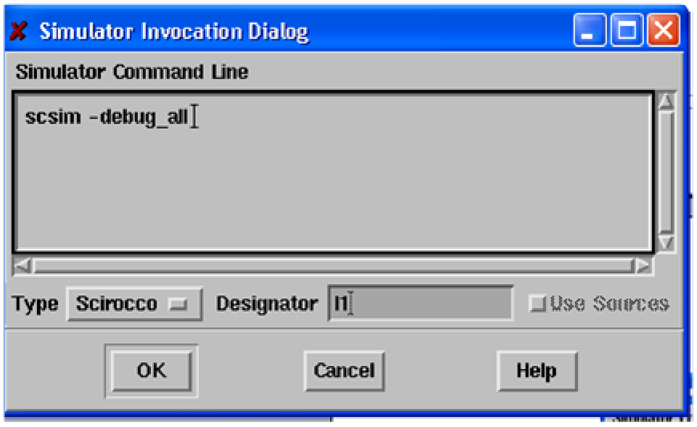
4) simulate the design

Two kinds of method can be used to simulate the design : File I/O– read stimulus from file and write results into file. (In our example, we generate a vector file named “expect_vector” , each cycle, the testbench read one row of “expector_vector” and pass the values to the hardware multiplier as input. Then compare the hardware output with the expected value in “expect_vector”, If any mismatch happens, write this cycle simulation information to a file named “error_vector” for debugging use) .

The second method is through observing the wave form of the simulation (following the commands below):

```
> scirocco &
```

Type in ’scsim -debug_all’ to open the simulation in debug mode. (As seen in figure)



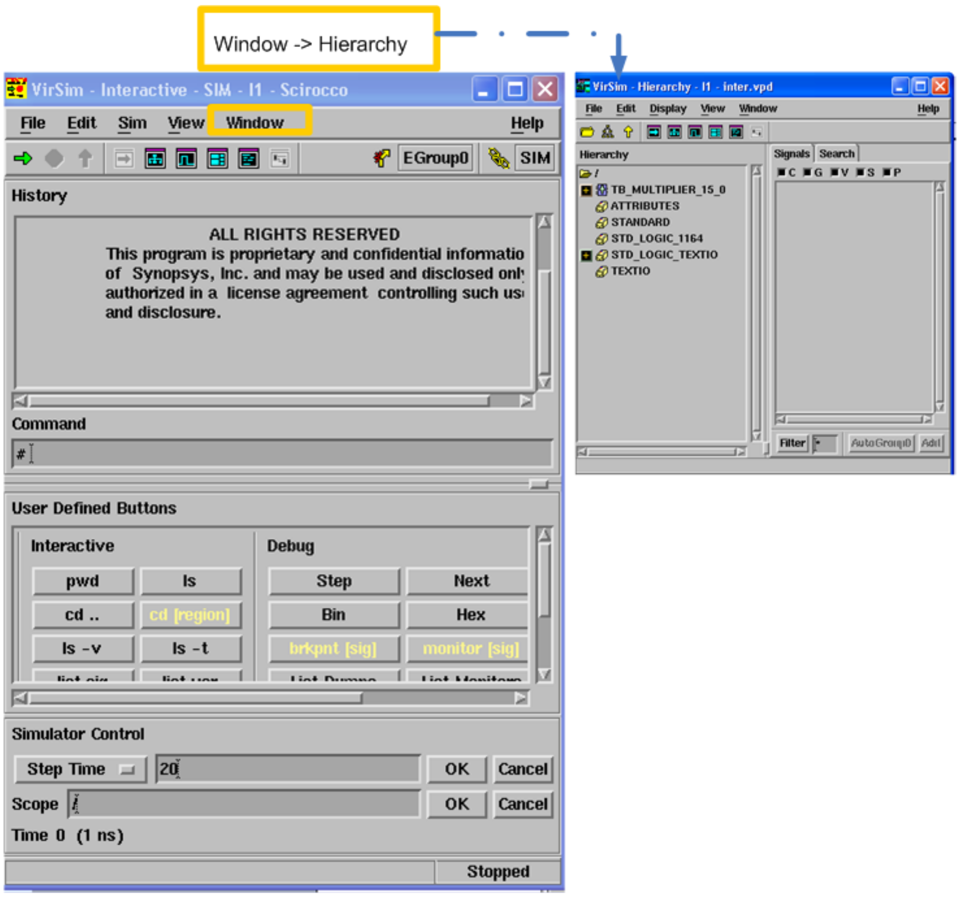
Simulator Command Line

```
scsim -debug_all
```

Type: Scirocco   Designator: l1

Figure 2: scsim window

In the VirSim window, click window -> Hierarchy



History

```
ALL RIGHTS RESERVED
This program is proprietary and confidential informatio
of Synopsys, Inc. and may be used and disclosed onl
authorized in a license agreement controlling such us
and disclosure.
```

Command: #

Simulator Control

Step Time: 20

Scope: /

Time 0 (1 ns)

Stopped

Figure 3: Hierarchy Window of Scsim

In the VirSim-Interactive window, click Window -> Waveform , in order to add signals into the waveform , you need to select these signals in the Hierarchy window.For our tutorial design, in Hierarchy window, choose “PT(31 downto 0) “ and press the middle button of your mouse, drag this signal to the waveform region. Similarly, add signals “XT(15 downto 0)” and “YT(15 downto 0)” to the waveform window.

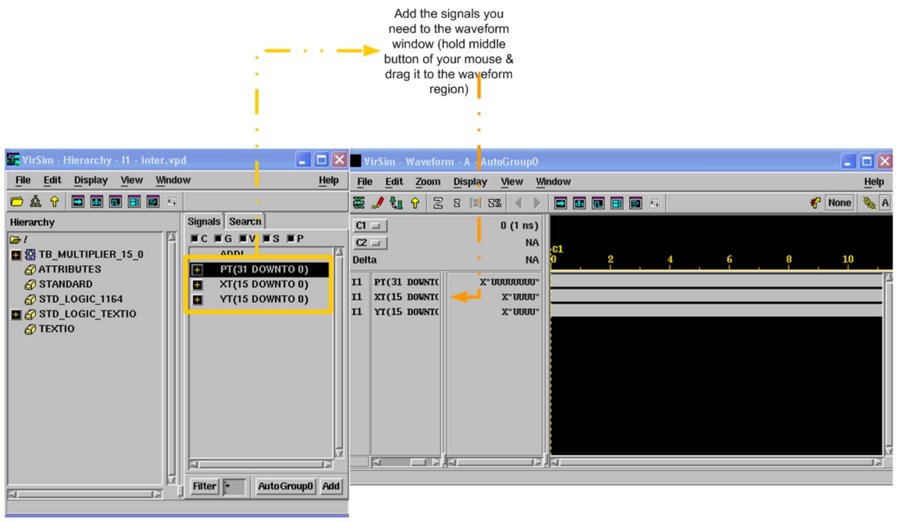
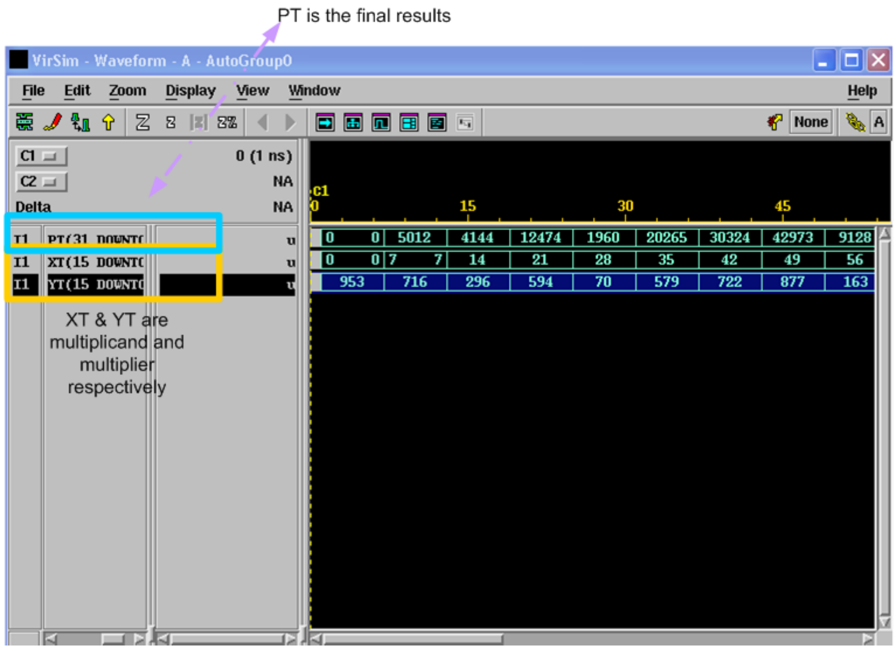

Add the signals you need to the waveform window (hold middle button of your mouse & drag it to the waveform region)
VirSim - Hierarchy - I1 - inter.vpd
VirSim - Waveform - A - AutoGroup0
PT(31 DOWNTO 0)
XT(15 DOWNTO 0)
YT(15 DOWNTO 0)
TB_MULTIPLIER_15_0
ATTRIBUTES
STANDARD
STD_LOGIC_1164
STD_LOGIC_TEXTIO
TEXTIO
X"UUUUUUUU"
X"UUUU"
X"UUUU"

Figure 4: Choose signals to be observed

In the VirSim-Interactive window, in “Simulator Control” area, you can set the Step Time (we leave 20ns as default), and click “OK” . The Simulation begins , and you can observe the resulting waveform for your debugging.

For behavior simulation, the major purpose is to verify the correctness of function. And from the wave form below, we indeed observe that $PT = XT \times YT$; thus verify the behavioral design.

Also, we can check the generated output file “error_vector” for behavioral verification.

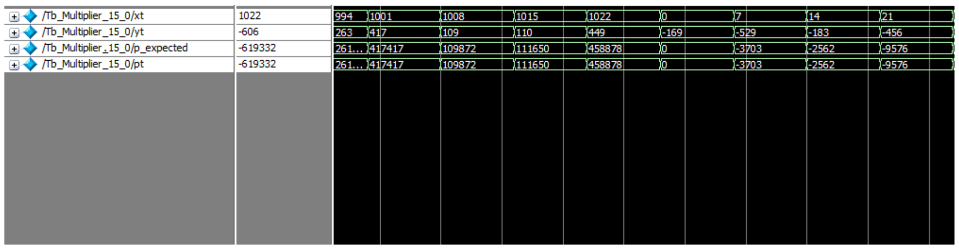


```
I1  PT(31 DOWNTO    u   0 | 0 | 5012 | 4144 | 12474 | 1960 | 20265 | 30324 | 42973 | 9128
I1  XT(15 DOWNTO    u   0 | 0 | 7 | 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56
I1  YT(15 DOWNTO    u   953 | 716 | 296 | 594 | 70 | 579 | 722 | 877 | 163
```

Figure 5: Simulation results

Behavior Simulation (For Verilog input)

You are encouraged to use a third-party simulator during your code implementation stage. It is more convenient. Modelsim, ActiveHDL for examples. Also you can try it on the workstation with synopsys tools – VCS.



```
/Tb_Multiplier_15_0/xt          1022     994  1001    1008    1015    1022    0    7     14    21
/Tb_Multiplier_15_0/yt          -606     263  417     109     110     449     -169 -529  -183  -456
/Tb_Multiplier_15_0/p_expected  -619332  261… 417417  109872  111650  458878  0    -3703 -2562 -9576
/Tb_Multiplier_15_0/pt          -619332  261… 417417  109872  111650  458878  0    -3703 -2562 -9576
```

Figure 6: A behavioral simulation by Modelsim

In “synopsys/sim” directory, following the commands as follows.

1) analyze the verilog file and testbench

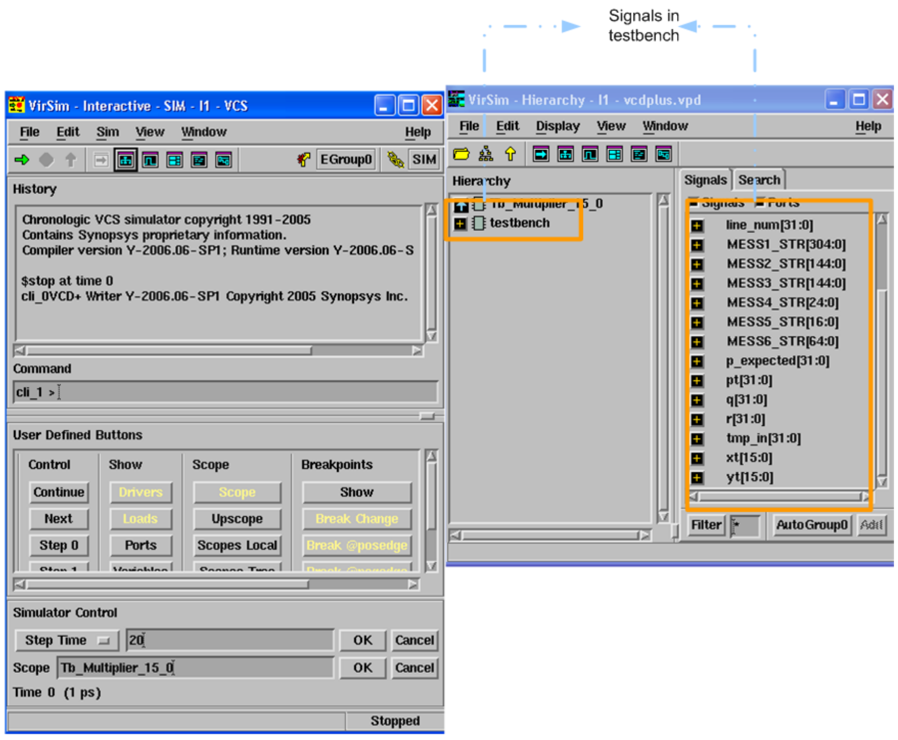
> vlogan ../verilog/ARITH_module.v

>vlogan +v2k ../verilog/Tb_multiplier.v

(note: our testbench is written in verilog IEEE2000 syntax, without “+v2k” ,vlogan command may fail due to compatible issue)

2) compile/simulate the design

- vcs +v2k ../verilog/Tb_multiplier.v ../verilog/ARITH_module.v (this compiles a simv file)
- vcs +v2k -RI ../verilog/Tb_multiplier.v ../verilog/ARITH_module.v (this opens a GUI interface)
- In VirSim-Interactive window ; choose Window -> Hierarchy to get a hierarchy view of our design. Choose Window -> Waveform to generate the waveform window.


Signals in testbench
VirSim - Interactive - SIM - I1 - VCS
File Edit Sim View Window Help
EGroup0 SIM
History
Command
User Defined Buttons
Control Show Scope Breakpoints
Continue Drivers Scope Show
Next Loads Upscope Break Change
Step 0 Ports Scopes Local Break @posedge
Simulator Control
Step Time
OK Cancel
Stopped
VirSim - Hierarchy - I1 - vcdplus.vpd
File Edit Display View Window Help
Hierarchy
Signals Search
Filter AutoGroup0 Add

```
Chronologic VCS simulator copyright 1991-2005
Contains Synopsys proprietary information.
Compiler version Y-2006.06-SP1; Runtime version Y-2006.06-S

$stop at time 0
cli_0VCD+ Writer Y-2006.06-SP1 Copyright 2005 Synopsys Inc.
```

```
cli_1 >
```

Step Time: 20
Scope: Tb_Multiplier_15_0
Time 0 (1 ps)

```
Tb_Multiplier_15_0
testbench
```

```
line_num[31:0]
MESS1_STR[304:0]
MESS2_STR[144:0]
MESS3_STR[144:0]
MESS4_STR[24:0]
MESS5_STR[16:0]
MESS6_STR[64:0]
p_expected[31:0]
pt[31:0]
q[31:0]
r[31:0]
tmp_in[31:0]
xt[15:0]
yt[15:0]
```

Figure 7: VirSim hierarchy window

- Choose the signals we want to observe in the Hierarchy window (pt[31:0], xt[15:0], yt[15:0] etc), hold the middle button of the mouse , drag them into the waveform window
- In “VirSim- Interactive window” , in “Simulation Control” area, type in your desired step time, and press OK. Then, you can observe the output waveforms

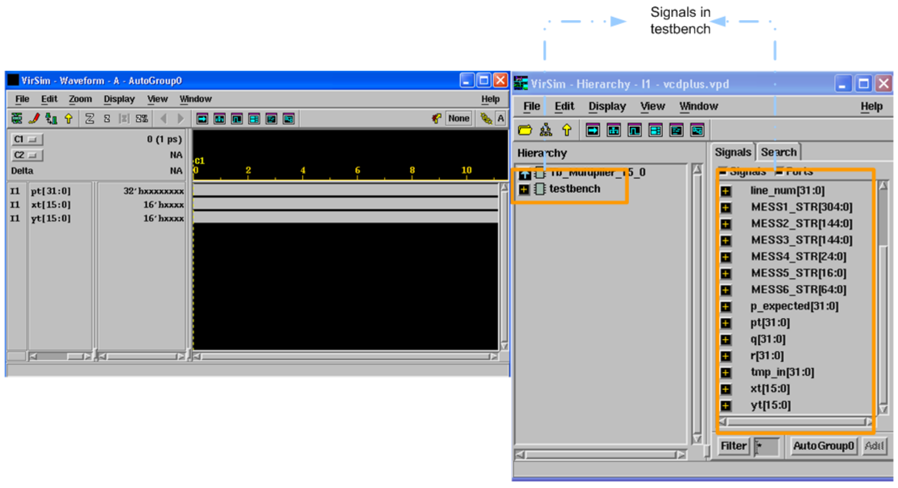
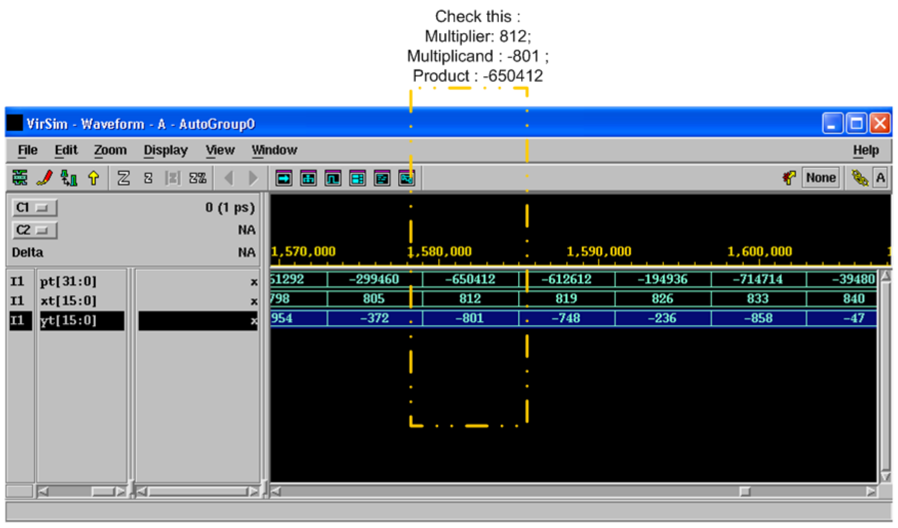


Figure 8: Waveform window to be observed



```
1,570,000        1,580,000        1,590,000        1,600,000
51292   -299460  -650412  -612612  -194936  -714714  -39480
798     805      812      819      826      833      840
954     -372     -801     -748     -236     -858     -47
```

Figure 9: Out put Waveforms

Synthesis with Synopsys Design Vision

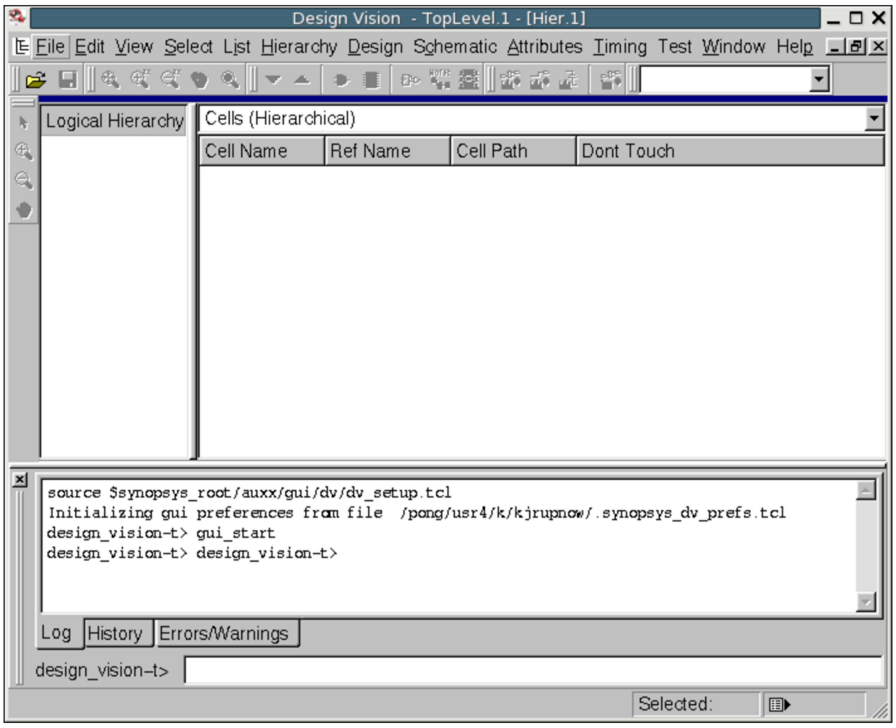
Input Your Design

This section would guide you step by step to synthesize a 16-bit multiplier example. We would use Synopsys Design Vision for this purpose.

- go to the syn library: cd ~/elec516-lab/synopsys/syn
- check out the there is a setup file ".synopsys_dc_setup" : ls -a

- In terminal : source ../../.cshrc_user (source the design vision)
- start design vision: design_vision &

For Linux workstations, the design_vision can be loaded correctly without any error or warning as figure below:


Design Vision - TopLevel.1 - [Hier.1]
File Edit View Select List Hierarchy Design Schematic Attributes Timing Test Window Help
Logical Hierarchy
Cells (Hierarchical)
Cell Name
Ref Name
Cell Path
Dont Touch
Log
History
Errors/Warnings
design_vision-t>
Selected:

```
source $synopsys_root/auxx/gui/dv/dv_setup.tcl
Initializing gui preferences from file  /pong/usr4/k/kjrupnow/.synopsys_dv_prefs.tcl
design_vision-t> gui_start
design_vision-t> design_vision-t>
```

Figure 10: Design Vision Main Window

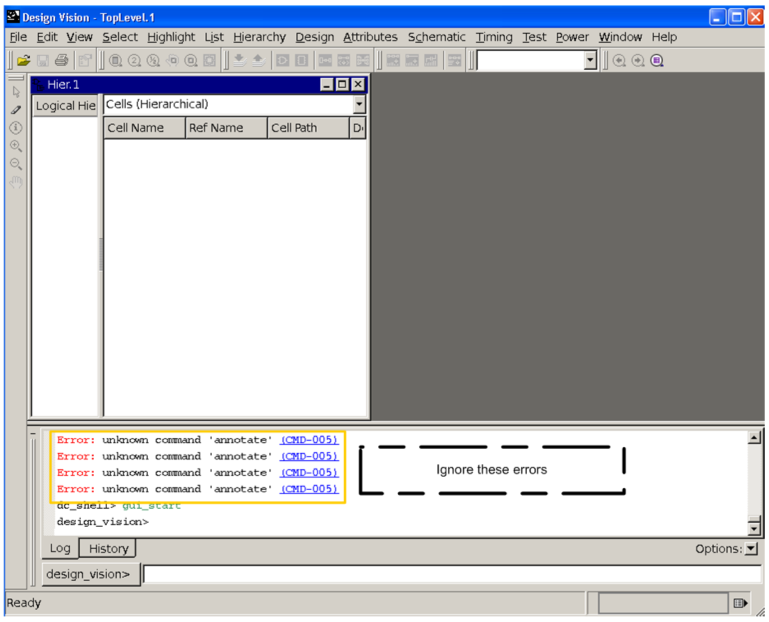
For UNIX SUN workstations, there may display some error messages of X-display as figures below, normally, it won't affect our synthesis process, we can ignore this compatible issue.

We may look at the main window of Design Vision to see a few of the program's features. In the top panel, there are two panes. The left pane is a full Hierarchy pane; it will show the entire hierarchy of the current design (as selected from the drop-down box in the upper control panel). The right panel is a context based panel which will display contents based upon the selection in the drop down box at the top of the panel.

The bottom panel has three different tabs: log, history and errors/warnings. The important thing to note about this panel is that every command you perform will appear in this panel, allowing you to learn the commands and create scripts of commands for future use. It is also the panel that you must monitor to determine the source of errors and warnings, allowing you to fix the code or correctly determine if a warning is expected.

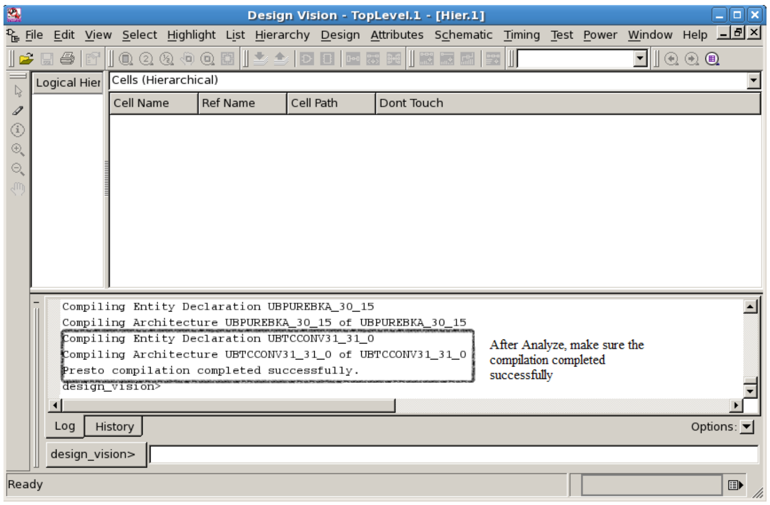
- analyze the file. Select File->Analyze and add the file you want to analyze, in our design , choose "elec516-lab/synopsys/vhdl/ARITH_module.vhd".

After analyzing the file, please check whether the compilation is sucessful.


Design Vision - TopLevel.1
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power Window Help
Hier.1
Logical Hie
Cells (Hierarchical)
Cell Name
Ref Name
Cell Path
Ignore these errors
Log
History
Options:
Ready

```
Error: unknown command 'annotate' (CMD-005)
Error: unknown command 'annotate' (CMD-005)
Error: unknown command 'annotate' (CMD-005)
Error: unknown command 'annotate' (CMD-005)
dc_shell> gui_start
design_vision>
```

Figure 11: Errors due to compatible issue in Sun(eesu) workstation


Design Vision - TopLevel.1 - [Hier.1]
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power Window Help
Logical Hier
Cells (Hierarchical)
Cell Name
Ref Name
Cell Path
Dont Touch
After Analyze, make sure the compilation completed successfully
Log
History
Options:
Ready

```
Compiling Entity Declaration UBPUREBKA_30_15
Compiling Architecture UBPUREBKA_30_15 of UBPUREBKA_30_15
Compiling Entity Declaration UBTCCONV31_31_0
Compiling Architecture UBTCCONV31_31_0 of UBTCCONV31_31_0
Presto compilation completed successfully.
design_vision>
```

Figure 13: After Analyze

Analyze is similar to compilation. It will check the syntax of each of the files to verify correct use of the language and that all code used is synthesizable.Note that for VHDL code, you must analyze in a specific order because of some of the ordering requirements of VHDL. No particular order will be required when using Verilog code with Design Vision. I recommend for the first time analyze your code, analyze one by one to see the messages, after that, you can add it together[1].

[1]Be careful not to include the testbench file , this is not part of the design and can not be synthesized

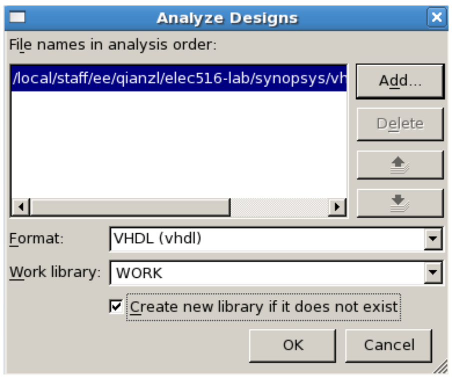

Analyze Designs
File names in analysis order:
/local/staff/ee/qianzl/elec516-lab/synopsys/vh
Add...
Delete
Format:
VHDL (vhdl)
Work library:
WORK
Create new library if it does not exist
OK
Cancel

Figure 12: Analyze VHDL Code

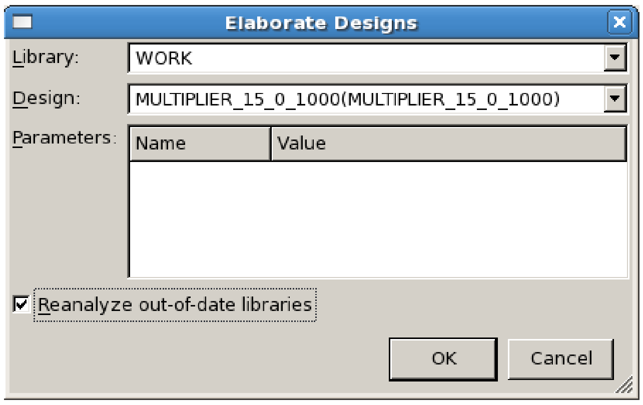
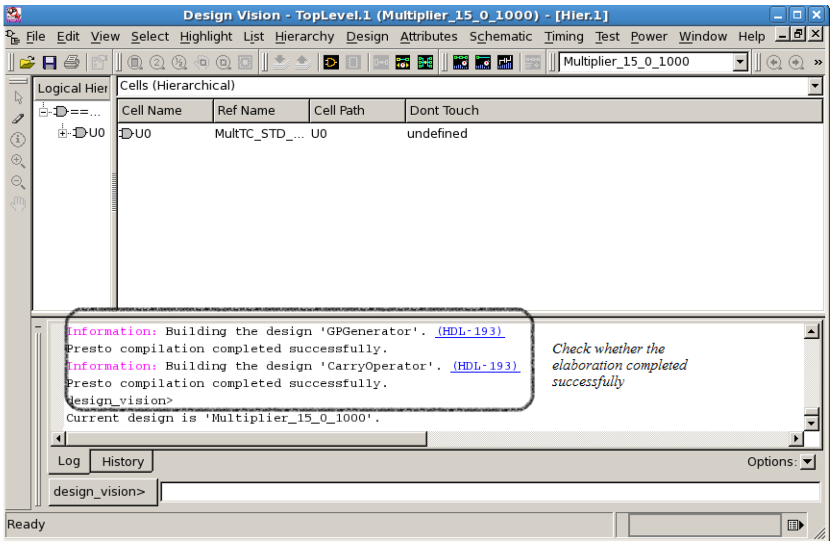

Elaborate Designs
Library:
WORK
Design:
MULTIPLIER_15_0_1000(MULTIPLIER_15_0_1000)
Parameters:
Name
Value
Reanalyze out-of-date libraries
OK
Cancel

Figure 14: Elaborate Design

- elaborate the analyzed code:
  - File-> Elaborate;
  - Change Library to “Work”; This is where you defined to store your intermediate results;
  - Change Design to “Multiplier_15_0_1000”. This is your top module;
  - Click “OK”;

The elaboration step may take several minutes. This step is similar to loading the design in Modelsim. The design is checked to make sure that the code is synthesizable, the subdesigns connect correctly and that there are no major errors in the implied circuit. (At the end of this step you may need to fix your code to remove them by checking the output message: THIS IS A MUST). Alternatively, you can use File-> Read to perform the function of analyze and elaborate.


Design Vision - TopLevel.1 (Multiplier_15_0_1000) - [Hier.1]
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power Window Help
Multiplier_15_0_1000
Logical Hier
Cells (Hierarchical)
Check whether the elaboration completed successfully
Log
History
Options:
Ready

Cell Name	Ref Name	Cell Path	Dont Touch
U0	MultTC_STD_...	U0	undefined

```
Information: Building the design 'GPGenerator'. (HDL-193)
Presto compilation completed successfully.
Information: Building the design 'CarryOperator'. (HDL-193)
Presto compilation completed successfully.
design_vision>
Current design is 'Multiplier_15_0_1000'.
```

```
design_vision>
```

Figure 15: Message After Elaboration

Now that we have elaborated the design, we see some basic structure of the design in the main window of Design Vision. See Figure.15. The left pane of the main window shows the full hierarchy of the design starting from the design that is specified as the top-level (from the drop-down box in the top tool panel). In the right pane we can select what we see to help perform later tasks. By right clicking on a module, we may also choose to see a schematic view from the right click menu. The schematic view may be useful, but requires a good understanding of synthesis, as the module names may seem cryptic at first.

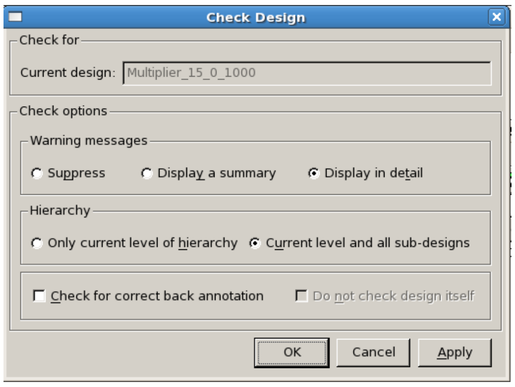
Design Check & Set Constraints

After you input all the design files, it is sometimes necessary for you to link the design (In File->Link Design). For sometime tools might no be able to correctly establish the interconnection between your different modules. When you finish all these, you check your design for possible errors (Design->Check Design). Example check results have warnings due to output connected to input ports directly (We can ignore this kind of warnings since we dedicately connect them in our design) .

- Select Multiplier_15_0_1000 as your current design (this is the top design module) in the module selection in the selection box right upper corner;
- Select Design->Check Design, and set the choice as shown in Figure.16;
- Check out the output messages;

You should perform frequency design check after each major operations which would affect the design logic.

Now you are going to set your design constraints to guide the mapping process. Common constraints are: clock periods, timing/area/power constraints, input delay, output delay, false path, wire load, operating conditions, fanout, etc.. (It's highly depends on the specification of the design system)



Check Design

Check for

Current design: Multiplier_15_0_1000

Check options

Warning messages

Suppress · Display a summary · [✓ Display in detail]

Hierarchy

Only current level of hierarchy · [✓ Current level and all sub-designs]

☐ Check for correct back annotation · ☐ Do not check design itself

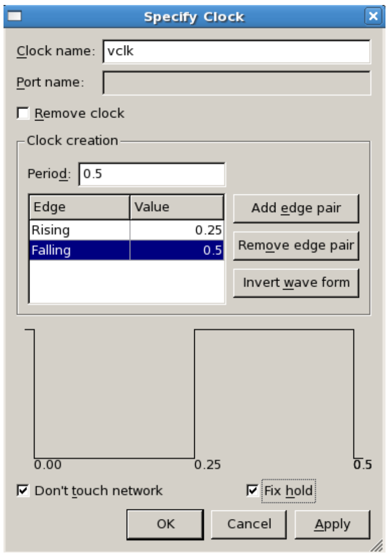
OK · Cancel · Apply

Figure 16: Check Design Box

Set Clock

Specify clock tells design vision the clock rate at which the design should be able to operate. This goal will tell design vision to make sure to organize the modules such that calculations can all be performed within the clock period. Various techniques such as logic duplication are used to achieve the goals. It is important to set realistic goals for the clock rate so that design vision does not perform too much logic duplication to attempt to reach the goal.

- Set Clock: If you have the clock pin in your design, then select this pin in port view, since in our multiplier design,we don't have any clock pins, we need't select any pin ;
- Select Attributes->Specify Clock (see. Figure. 17);
- We create a clock name : vclk which means this is a virtual clock; Period is 0.5 here; feel free to try out different values to see how fast the system can run;



Specify Clock

Clock name: vclk

Port name:

☐ Remove clock

Clock creation

Period: 0.5

Edge	Value
Rising	0.25
Falling	0.5

Add edge pair · Remove edge pair · Invert wave form

☑ Don't touch network · ☑ Fix hold

OK · Cancel · Apply

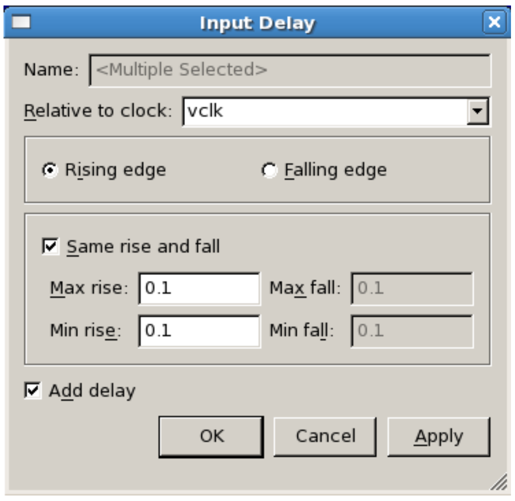
Figure 17: Specify Clock

You will know if you have correctly selected Clk based upon if Clk shows up in the grayed out Port name box. If that box is empty, be sure to close this dialog and make sure Clk is selected before opening this dialog. Since a clock is a signal we don't want to optimize (prevents errors of the synthesizer disconnecting the signal), we want to set the clock as a "don't touch" network. Select Don't touch network before clicking OK.

Set Input/Output Delay

Input delay tells design vision that a signal will always arrive at a certain time relative to the clock (after the rising edge of the clock). When specifying delay, the delay should always be specified relative to a clock so that design vision may calculate delays correctly.

- Change the drop down box to Pins/Ports.
- Select->Ports/Pins->Input Ports to select all input pins
- Attributes->Operating Environment->Input Delay (If you did not select any signals, either the Input Delay option in the menu would be grayed out, or there would be no entries in the Name field).
- Select vclk in the Relative to clock dropdown Specify 0.1 as the Minimum and Maximum delay This means that input signals should be modeled so that they arrive 0.1nS after the edge of clk. Click OK
- Do the same for the output delay (This specifies how long a signal takes to reach the chips output after leaving your modules output after the clock edge.)


Input Delay

Name: <Multiple Selected>

Relative to clock: vclk

[✓ Rising edge] Falling edge

[✓] Same rise and fall

Max rise: 0.1 Max fall: 0.1

Min rise: 0.1 Min fall: 0.1

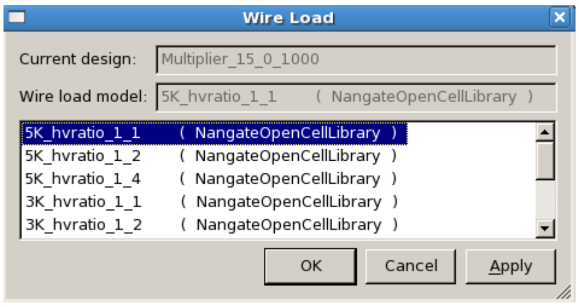
[✓] Add delay

Figure 18: Set Input Delay

Wire Load

We must specify a wire load model so that design vision can estimate the delay that wires in the design have. Each model is based upon a different amount of resistance and capacitance for a certain amount of wire. Design vision will use that amount to estimate how much delay is added to the circuit based on the length of the wires and distance between the wires.

- Attributes->Operating Environment->Wire Load;
- Select-> 5K_hvratio_1_1;



Current design: Multiplier_15_0_1000

Wire load model: 5K_hvratio_1_1 (NangateOpenCellLibrary)

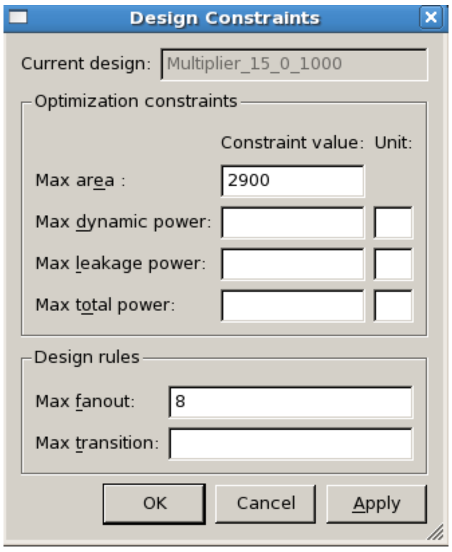
```
5K_hvratio_1_1   ( NangateOpenCellLibrary )
5K_hvratio_1_2   ( NangateOpenCellLibrary )
5K_hvratio_1_4   ( NangateOpenCellLibrary )
3K_hvratio_1_1   ( NangateOpenCellLibrary )
3K_hvratio_1_2   ( NangateOpenCellLibrary )
```

Figure 19: Wire Load

Design Constraints

This level of constraints allows us to set overall objectives of the design for design vision to attempt to reach. We can set maximum values for area, power, fanout, and transition. Normally for this course we will only set values for area and fanout, allowing power and transition to vary as design vision likes. This will speed our synthesis time and will allow us to concentrate on the operation of the circuit rather than worrying about power and other effects that require attention after full correctness is guaranteed.

In "Attributes > Optimization Constraints > Design constraints" we can set the optimization goal



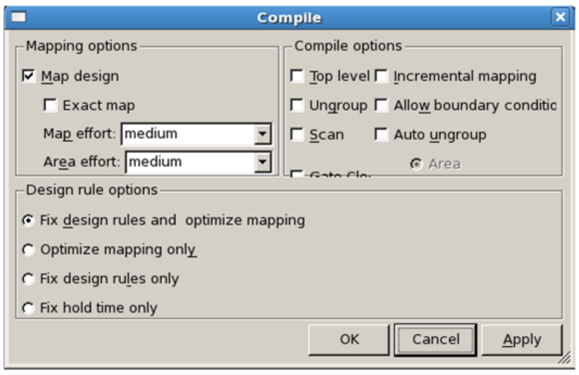
Current design: Multiplier_15_0_1000

	Constraint value:	Unit:
Max area :	2900	
Max dynamic power:		
Max leakage power:		
Max total power:		

Max fanout: 8

Max transition:

Figure 20: Design Constraints



Compile

Mapping options
- [✓] Map design
- [ ] Exact map
- Map effort: medium
- Area effort: medium

Compile options
- [ ] Top level
- [ ] Incremental mapping
- [ ] Ungroup
- [ ] Allow boundary conditic
- [ ] Scan
- [ ] Auto ungroup
- Area

Design rule options
- (●) Fix design rules and optimize mapping
- ( ) Optimize mapping only
- ( ) Fix design rules only
- ( ) Fix hold time only

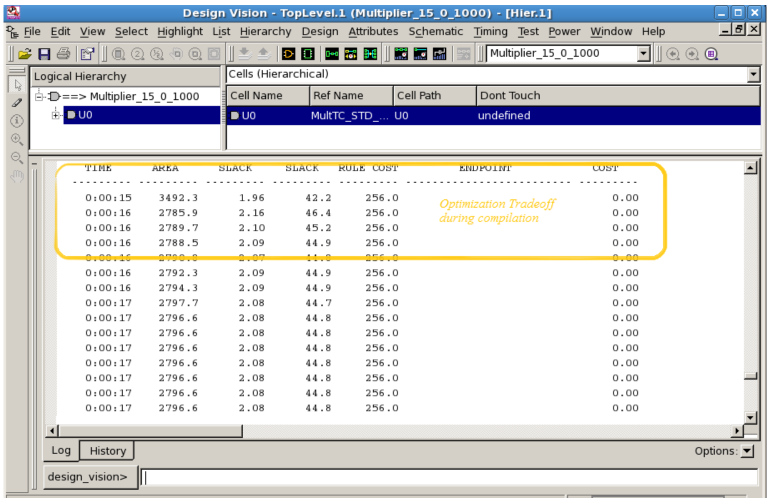
Figure 21: Compile Design

Start Compile

Now we've setup all the constraints. We will start to compile the design (logic simplification and technology mapping) and hope that Design Vision would get a satisfactory design for all our constraints.

- Design->Compile Design
- Uncheck the exact map option (this would leads no logic optimization). See Figure. 21

After finish, check out the output message. There're each optimization step output to tell you the current trade-offs between area and timings. Check if the design is satisfactory. Perform check design if necessary.



Design Vision - TopLevel.1 (Multiplier_15_0_1000) - [Hier.1]

File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power Window Help
Multiplier_15_0_1000
Logical Hierarchy
==> Multiplier_15_0_1000
U0
Cells (Hierarchical)

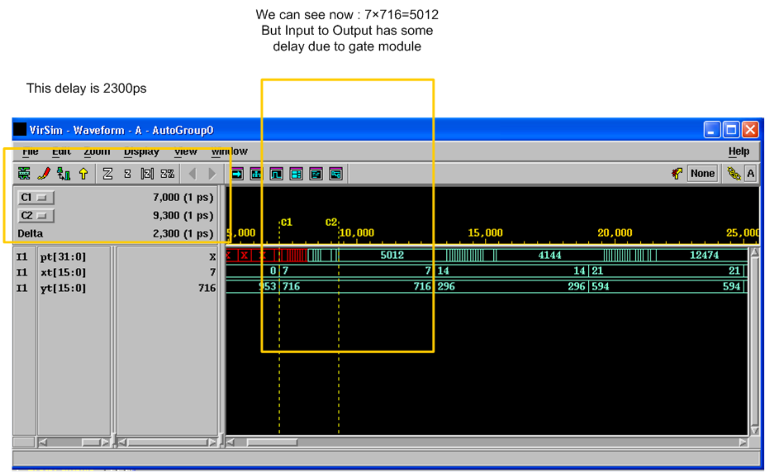
Cell Name	Ref Name	Cell Path	Dont Touch
U0	MultTC_STD_...	U0	undefined

```
 TIME       AREA       SLACK      SLACK     RULE  COST            ENDPOINT             COST
--------- --------- --------- --------- --------- ------------------------- ---------
  0:00:15    3492.3      1.96      42.2     256.0                                0.00
  0:00:16    2785.9      2.16      46.4     256.0                                0.00
  0:00:16    2789.7      2.10      45.2     256.0                                0.00
  0:00:16    2788.5      2.09      44.9     256.0                                0.00
  0:00:16    2790.0      2.07      44.8     256.0                                0.00
  0:00:16    2792.3      2.09      44.9     256.0                                0.00
  0:00:16    2794.3      2.09      44.9     256.0                                0.00
  0:00:17    2797.7      2.08      44.7     256.0                                0.00
  0:00:17    2796.6      2.08      44.8     256.0                                0.00
  0:00:17    2796.6      2.08      44.8     256.0                                0.00
  0:00:17    2796.6      2.08      44.8     256.0                                0.00
  0:00:17    2796.6      2.08      44.8     256.0                                0.00
  0:00:17    2796.6      2.08      44.8     256.0                                0.00
  0:00:17    2796.6      2.08      44.8     256.0                                0.00
  0:00:17    2796.6      2.08      44.8     256.0                                0.00
```

Figure 22: After compilation of the design

Then, you can save the mapped design. All the designs would be saved in ddc format. Later you can read in the design where you left off. Here, i want emphasis two things you need to save for use in back-end design.

- A verilog netlist for place & route and pre-layout simulation: In the hierarchical window, select the top module, and then File->Save as, choose verilog format, name it as syn_netlist.v


We can see now : 7×716=5012
But Input to Output has some delay due to gate module

This delay is 2300ps

Figure 23: pre-layout simulation results

- A constraint file for the constraints you set for synthesis. it would be used for back-end too. we name it syn_netlist.sdc .

Report File

There're various report method you can use to check your design. The most important one is timing report (like "Design > Report Design").

Pre-Layout Simulation

Till now, you are required to perform Pre-Layout simulation to verify your function. The file you needed is syn_netlist.v and a library verilog file called typical.v under "syn_sim" folder.

Using the similar method introduced in the " behavioral simulation for verilog " to do the pre-layout simulation

> cd elec516-lab/synopsys/syn_sim (where library file typical.v and syn_netlist.v locates in)

>cp ../verilog/Tb_multiplier.v . (copy testbench to syn_sim folder)

>source /usr/eelocal/synopsys/vcs_mx-vy2006.06-sp1/.cshrc

>vlogan syn_netlist.v

>vlogan +v2k Tb_multiplier.v

>vcs -RI Tb_multiplier.v +v2k syn_netlist.v -v typical.v

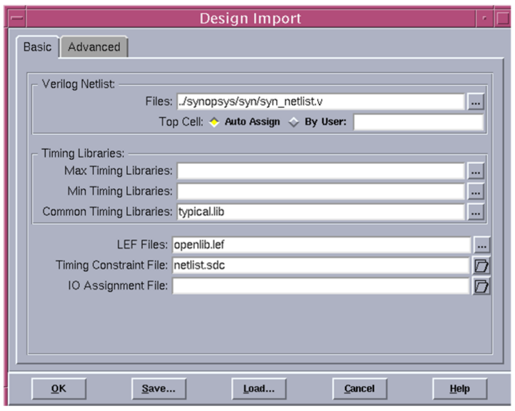
Then,we enter into the GUI of the simulator. For how to choose signals for observation and the explanations of the interface, you may refer to previous " behavioral simulation for verilog " section.

Place & Route with Cadence Encounter

setup and start encounter

copy your generated netlist file and constraint file into encounter for back end use:

- cp syn_netlist.v ../../encounter
- source ../.cshrc_user
- check there're library files under the "encounter" folder: ".lib" and ".lef" respectively
- start encounter: encounter (Make sure DO NOT type " & " after encounter)

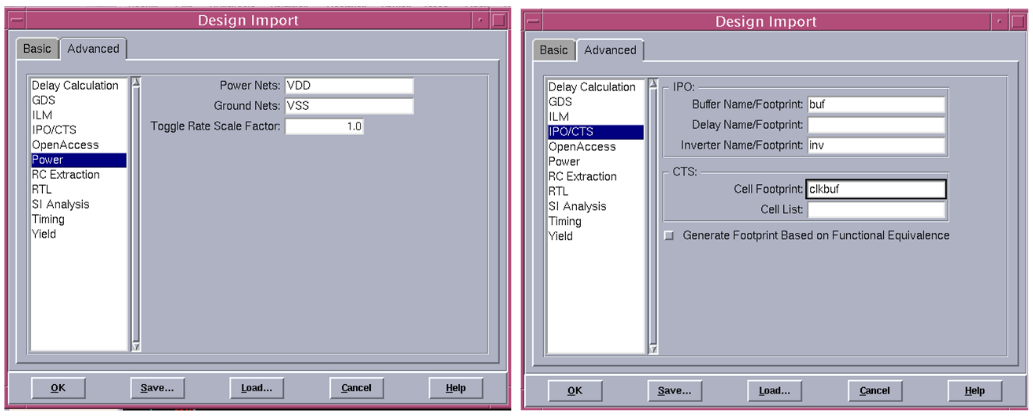


Verilog Netlist:
Files: ./synopsys/syn/syn_netlist.v
Top Cell: [✓ Auto Assign] By User:

Timing Libraries:
Max Timing Libraries:
Min Timing Libraries:
Common Timing Libraries: typical.lib

LEF Files: openlib.lef
Timing Constraint File: netlist.sdc
IO Assignment File:

(a) Basic Setting



Power Nets: VDD
Ground Nets: VSS
Toggle Rate Scale Factor: 1.0

IPO:
Buffer Name/Footprint: buf
Delay Name/Footprint:
Inverter Name/Footprint: inv

CTS:
Cell Footprint: clkbuf
Cell List:

☐ Generate Footprint Based on Functional Equivalence

(b) Power Setting & IPO Setting

Figure 24: Design Import

Load Design

- Design-> Design Import, set the Basic and Advanced Option as Figure. 24

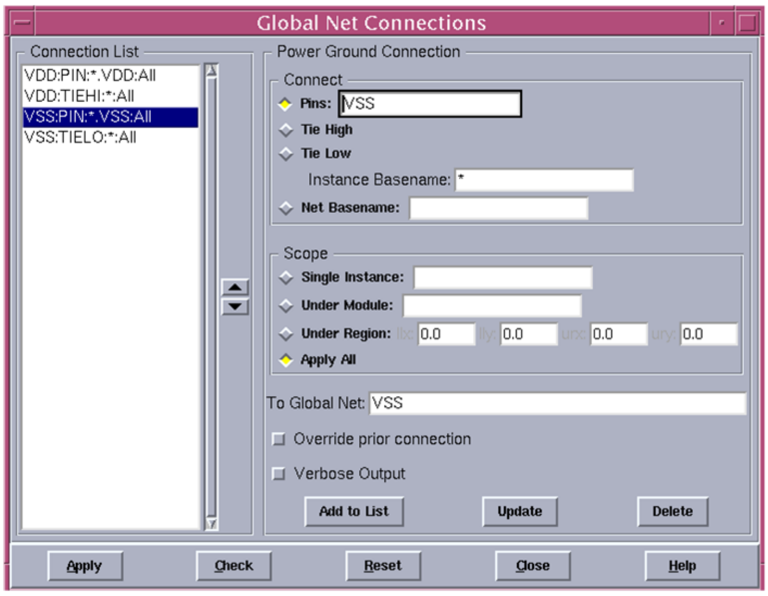
- For future use, after you set up all the required field, you can save a config file and next time directly load it is ok

Connect Global Nets

- Click Floorplan->Connect Global Nets
- Following the setting listed in the table below and add the connects to the list
- Click Apply, and then Check. The Check command check for unconnected pins in your design. Verify and fix if any warnings or errors appear in the encounter console.

Setting	Setting1	Setting2	Setting3	Setting4
Connect Region	Pins: VDD	Tie High	Pins: VSS	Tie Low
Scope Region	Apply All	Apply All	Apply All	Apply All
To Global Net	VDD	VDD	VSS	VSS

Table 1: Global Nets Connections


Global Net Connections

Connection List
VDD:PIN:*.VDD:All
VDD:TIEHI:*:All
VSS:PIN:*.VSS:All
VSS:TIELO:*:All

Power Ground Connection
Connect
[✓ Pins:] VSS
Tie High
Tie Low
Instance Basename: *
Net Basename:

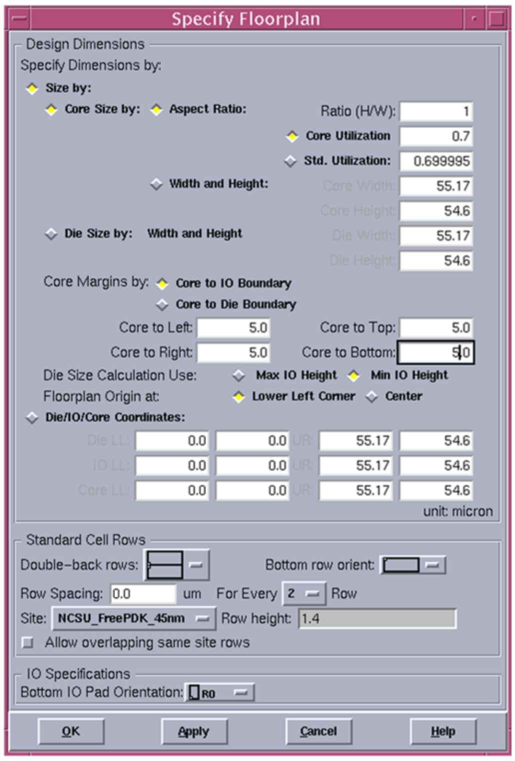
Scope
Single Instance:
Under Module:
Under Region: llx 0.0 lly 0.0 urx 0.0 ury 0.0
[✓ Apply All]

To Global Net: VSS
☐ Override prior connection
☐ Verbose Output

Figure 25: Connect Global Nets

Floorplan

- Floorplan-> Specify Floorplan. See Figure.26 for the setting;
- Figure.27 shows the result floorplan;



Specify Floorplan

Design Dimensions

Specify Dimensions by:

[✓ Size by:]

[✓ Core Size by:] [✓ Aspect Ratio:] Ratio (H/W): 1

[✓ Core Utilization] 0.7

Std. Utilization: 0.699995

Width and Height: Core Width 55.17

Core Height 54.6

Die Size by: Width and Height Die Width 55.17

Die Height 54.6

Core Margins by: [✓ Core to IO Boundary]

Core to Die Boundary

Core to Left: 5.0 Core to Top: 5.0

Core to Right: 5.0 Core to Bottom: 5.0

Die Size Calculation Use: Max IO Height [✓ Min IO Height]

Floorplan Origin at: [✓ Lower Left Corner] Center

Die/IO/Core Coordinates:

Die LL:	0.0	0.0	UR:	55.17	54.6
IO LL:	0.0	0.0	UR:	55.17	54.6
Core LL:	0.0	0.0	UR:	55.17	54.6

unit: micron

Standard Cell Rows

Double-back rows: Bottom row orient:

Row Spacing: 0.0 um For Every 2 Row

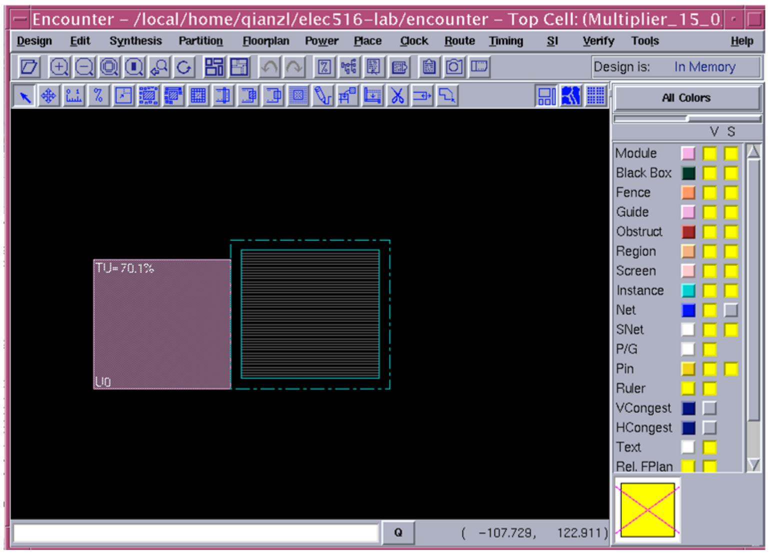
Site: NCSU_FreePDK_45nm Row height: 1.4

Allow overlapping same site rows

IO Specifications

Bottom IO Pad Orientation: R0

Figure 26: Floorplan Setting

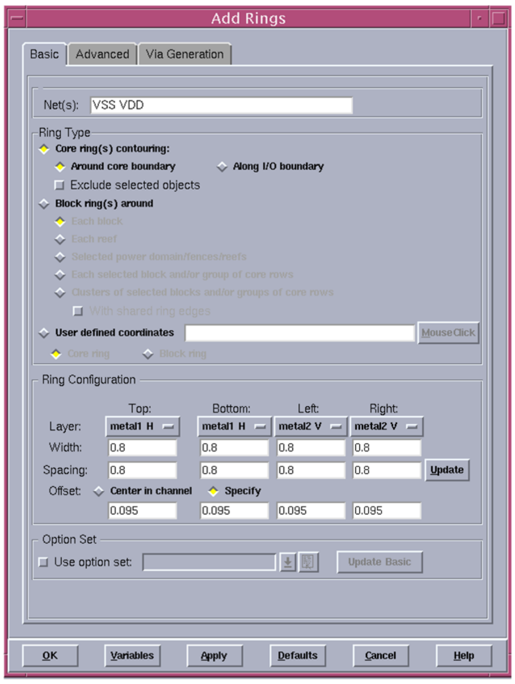


Encounter – /local/home/qianzl/elec516-lab/encounter – Top Cell: (Multiplier_15_0

Figure 27: View After Floorplan

Add Power Ring and Stripe

- Power->Power Planning->Add Rings (Figure. 28)



Add Rings

Basic | Advanced | Via Generation

Net(s): VSS VDD

Ring Type

[✓ Core ring(s) contouring:]

[✓ Around core boundary] Along I/O boundary

☐ Exclude selected objects

Block ring(s) around

[✓ Each block]
Each reef
Selected power domain/fences/reefs
Each selected block and/or group of core rows
Clusters of selected blocks and/or groups of core rows
☐ With shared ring edges

User defined coordinates [MouseClick]

[✓ Core ring] Block ring

Ring Configuration

	Top:	Bottom:	Left:	Right:
Layer:	metal1 H	metal1 H	metal2 V	metal2 V
Width:	0.8	0.8	0.8	0.8
Spacing:	0.8	0.8	0.8	0.8
Offset:	0.095	0.095	0.095	0.095

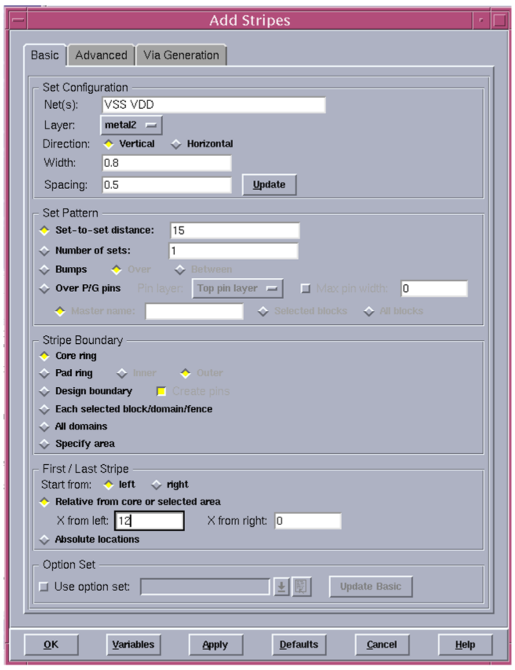
Offset: Center in channel [✓ Specify]

Option Set

☐ Use option set:

Figure 28: Add Ring Config

- Power->Power Planning->Add Stripe (Figure. 29)



Add Stripes

Basic | Advanced | Via Generation

Set Configuration
Net(s): VSS VDD
Layer: metal2
Direction: [✓ Vertical] Horizontal
Width: 0.8
Spacing: 0.5 Update

Set Pattern
[✓ Set-to-set distance:] 15
Number of sets: 1
Bumps Over Between
Over P/G pins Pin layer: Top pin layer Max pin width: 0
Master name: Selected blocks All blocks

Stripe Boundary
[✓ Core ring]
Pad ring Inner Outer
Design boundary Create pins
Each selected block/domain/fence
All domains
Specify area

First / Last Stripe
Start from: [✓ left] right
[✓ Relative from core or selected area]
X from left: 12 X from right: 0
Absolute locations

Option Set
Use option set: Update Basic

OK Variables Apply Defaults Cancel Help

Figure 29: Add Stripe Config

After you add power ring and stripes, you can see now we have a primitive view in Figure.30 .

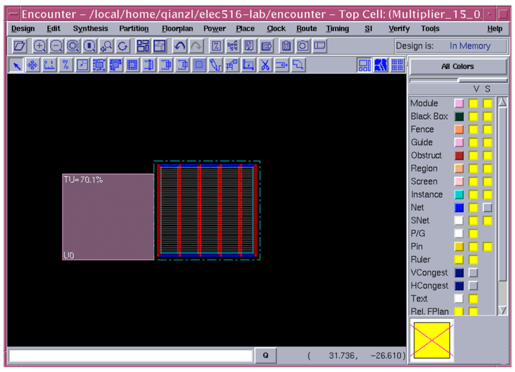

Encounter – /local/home/qianzl/elec516-lab/encounter – Top Cell: (Multiplier_15_0
Design Edit Synthesis Partition Floorplan Power Place Clock Route Timing SI Verify Tools Help
Design is: In Memory
All Colors
V S
Module
Black Box
Fence
Guide
Obstruct
Region
Screen
Instance
Net
SNet
P/G
Pin
Ruler
VCongest
HCongest
Text
Rel. FPlan
TU=70.1%
U0
(31.736, -26.610)

Figure 30: View after add power ring and stripes

Special Routing

- Route->Special Route; leave everything in default and ok
- Special Route would help you do the VDD/VSS connection.

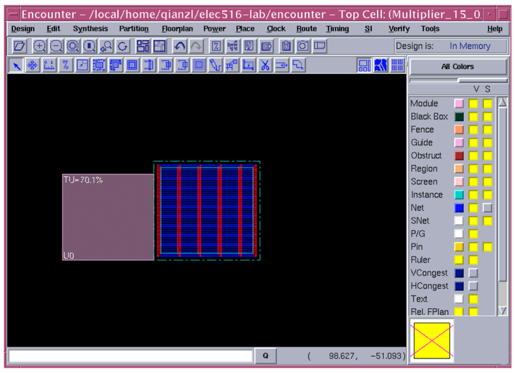
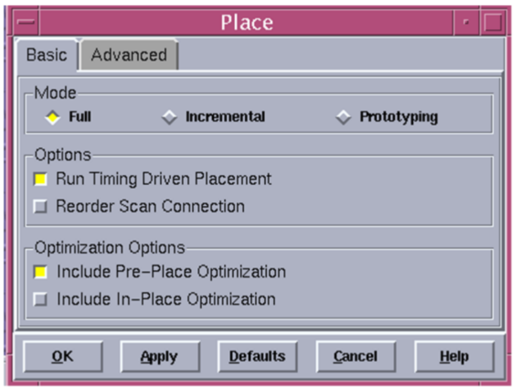

Encounter – /local/home/qianzl/elec516-lab/encounter – Top Cell: (Multiplier_15_0
Design Edit Synthesis Partition Floorplan Power Place Clock Route Timing SI Verify Tools Help
Design is: In Memory
All Colors
V S
Module
Black Box
Fence
Guide
Obstruct
Region
Screen
Instance
Net
SNet
P/G
Pin
Ruler
VCongest
HCongest
Text
Rel. FPlan
TU=70.1%
U0
(98.627, -51.093)

Figure 31: View after Special Routing

Place the Standard Cell

This step is to place the cell in your design into the floorplan. The quality of placement would affect your timing a lot.

- Place->Standard Cell and Blocks
- Un-check the Reorder Scan Connection



Place

Basic | Advanced

Mode

[✓ Full] Incremental Prototyping

Options

[✓ Run Timing Driven Placement]

Reorder Scan Connection

Optimization Options

[✓ Include Pre–Place Optimization]

Include In–Place Optimization

OK Apply Defaults Cancel Help

Figure 32: Placement Setting

After placement, switch your view into physical view. You can see how your cell is placed.

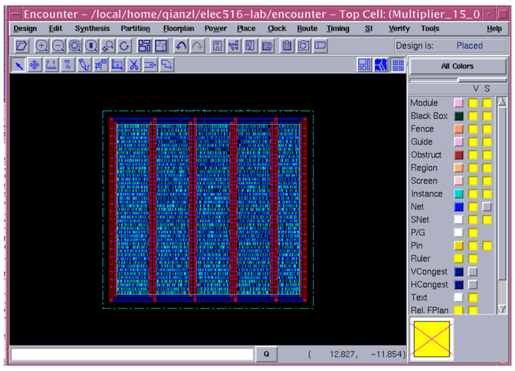

Encounter – /local/home/qianzl/elec516-lab/encounter – Top Cell: (Multiplier_15_0
Design Edit Synthesis Partition Floorplan Power Place Clock Route Timing SI Verify Tools Help
Design is: Placed
All Colors
Module
Black Box
Fence
Guide
Obstruct
Region
Screen
Instance
Net
SNet
P/G
Pin
Ruler
VCongest
HCongest
Text
Rel. FPlan

Figure 33: Physical View after placement

Route the connection

- Route-> Nano Route->Route, leave everything is default

Here is what you get after routing:

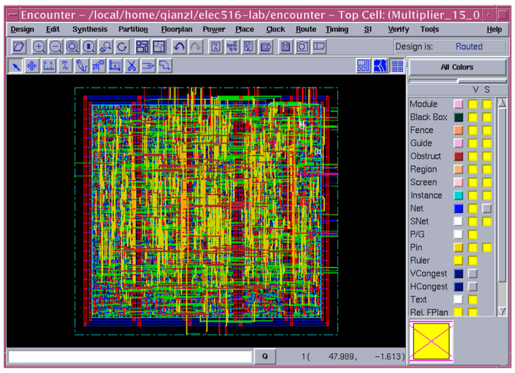
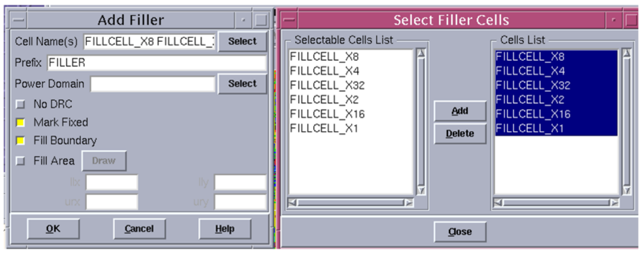

Encounter – /local/home/qianzl/elec516-lab/encounter – Top Cell: (Multiplier_15_0
Design Edit Synthesis Partition Floorplan Power Place Clock Route Timing SI Verify Tools Help
Design is: Routed
All Colors
Module
Black Box
Fence
Guide
Obstruct
Region
Screen
Instance
Net
SNet
P/G
Pin
Ruler
VCongest
HCongest
Text
Rel. FPlan

Figure 34: View After Routing

Add Filler Cell

Filler Cell has no function and is added to fill all the blank area without cells. The aim is to increase physical stress.

- Place->Filler->Add...
- Select Filler Name and Add all available fillers (shown in Figure.)
- OK



Cell Name(s) FILLCELL_X8 FILLCELL_:

Prefix FILLER

Power Domain

No DRC ☐

Mark Fixed ✓

Fill Boundary ✓

Fill Area ☐

Selectable Cells List: FILLCELL_X8, FILLCELL_X4, FILLCELL_X32, FILLCELL_X2, FILLCELL_X16, FILLCELL_X1

Cells List: FILLCELL_X8, FILLCELL_X4, FILLCELL_X32, FILLCELL_X2, FILLCELL_X16, FILLCELL_X1

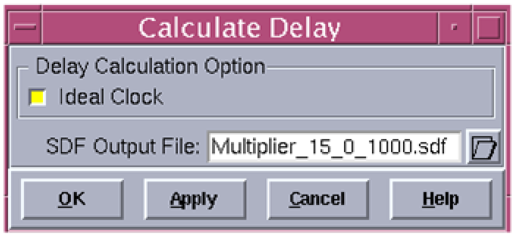
Figure 35: Add Filler

Calculate Delay:

> Timing -> Extract RC... (leave everything as default and click OK)

> Timing -> Calculate Delay

The delay is extracted from the wire loading and save into an delay file called "sdf " file. We can use it to do the Post-Layout simulation



Delay Calculation Option

Ideal Clock ✓

SDF Output File: Multiplier_15_0_1000.sdf

Figure 36: Calculate Delay

Save the Final Netlist and layout file

This netlist might be different from the one you saved after synthesis. During P&R, some logic might be optimized again for better timing. So you should use this netlist and sdf from last step to perform your Post-Layout Simulation.

- Design->Save->Netlist, name the file as your wish
- Design->Save Design to save the layout in enc file format (the actual file for fabrication is in gds format you can also save it)

Post-simulation

The simulation can be done with the netlist from P&R, sdf file from P&R which we have saved in the previous step and standard cell library verilog file (the same typical.v file in pre-layout simulation). The sdf file is annotated to take account for the interconnection delay. you can refer to the appendix. for a reference to add it in the testbench. ("$sdf_annotate" in the testbench) ; after adding sdf annotation in test bench, the post layout simulation is similar to pre-layout simulation. (Note : Because we use the testbench which contains sdf (timing information of layout) ,the results are more accurate).

Appendix A

```
--------------------------------------------------------------------------------
// sample verilog testbench
‘timescale 1ns / 1ps
module MyDesign_tb;
reg clk;
MyDesign uut ( // design ports );
always #50

    clk = !clk;

initial begin

    $sdf_annotate("./MyDesign.sdf", uut);
    // uut is the name you instantiate your design in the testbench
    // assume sdf in within the same folder of your simulation folder

end
initial begin

    # 0 // Have your test bench here
    # 10000000000
    $finish;

end
endmodule
---------------------------------------------------------------------------------
```