Design Vision – User manual for Beginners

Welcome to the world of ASIC Design. You must be familiar with the synthesis process by now, having worked on tools like Xilinx ISE or Quartus from Altera. This document has been prepared with the intention of making you comfortable with the synthesis process in ASIC design using Design compiler tool from Synopsys as well as throw some light on how to read in the files, analyze the reports and saving them. The manual consists of two parts:

1.) The first making the user familiar with the Graphical user interface and loading the design
2.) The second explaining the synthesize process and constraining the design, generating and analyzing the reports.

Basic Logic Synthesis Process
Figure 1.01 shows the design flow you will follow in this tutorial.

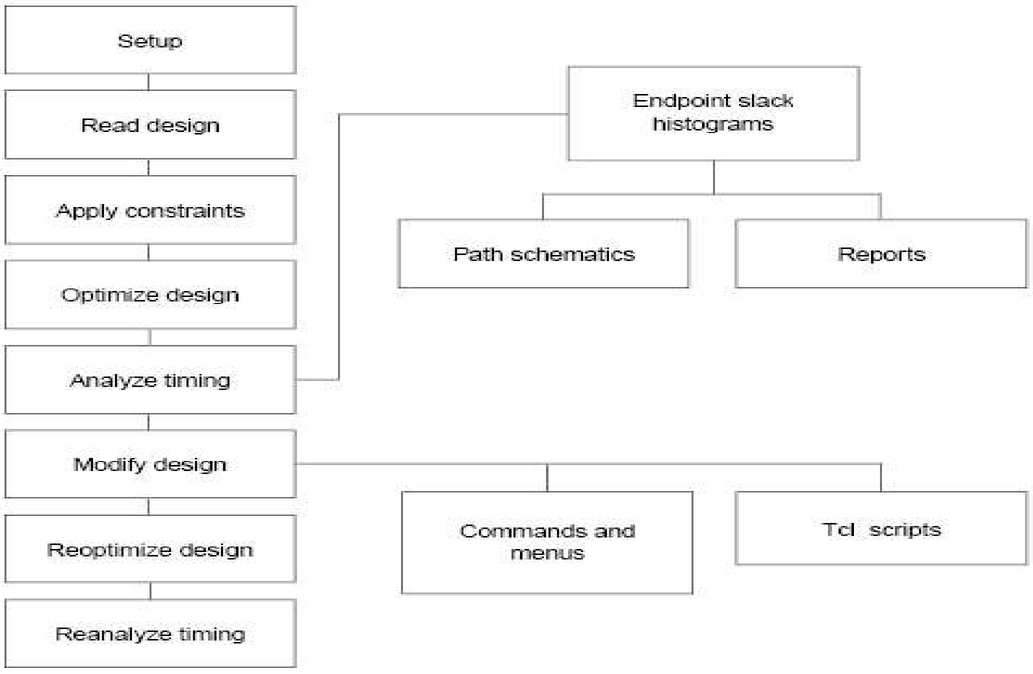


Figure 1.01: Design Flow

The tutorial directories and files are installed under the lab_exercises directory in the desktop of user login. The directory consists of

- udc.v
- cb13fs120_tsmc_max.db
- cb13fs120_icon.sdb
- .synopsys_dc.setup
- run.scr

where udc.v is a Verilog code for a loadable updown counter, cb13fs120_tsmc_max.db is the logic library, cb13fs120_icon.sdb is the symbol library, run.scr is the script file, and .synopsys_dc.setup is the setup file, more of which shall be discussed additionally as we proceed further.

Chapter 1

1.1 Starting and Exiting Design Compiler From Design Vision

Design Vision is the tool which allows you to access the Design compiler synthesis tool through the Graphical User Interface(GUI). You invoke Design Vision from the Linux shell by first right clicking on the desktop and then by clicking on ***open terminal*** (as shown in figure 1.02). Then navigate to the appropriate directory using the "***cd***" command. These directories have their own ".synopsys_dc.setup" file that must be used. Therefore, before invoking Design Vision for a particular exercise, make sure you change to the correct directory. Ensure that you are in the appropriate directory using the "***pwd***" command as in figure 1.03.

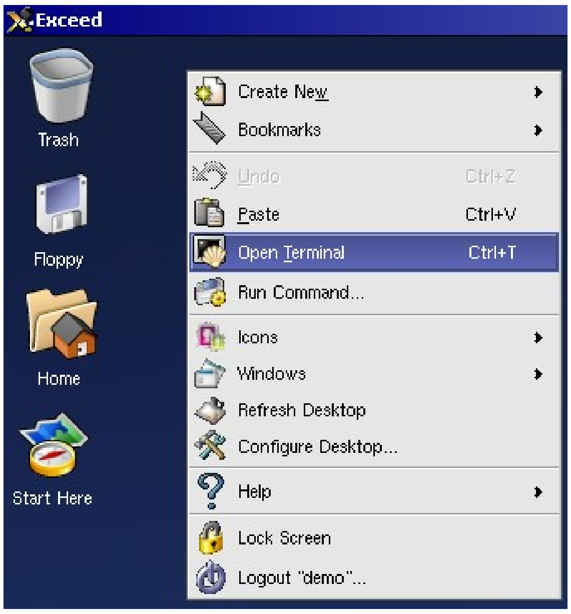
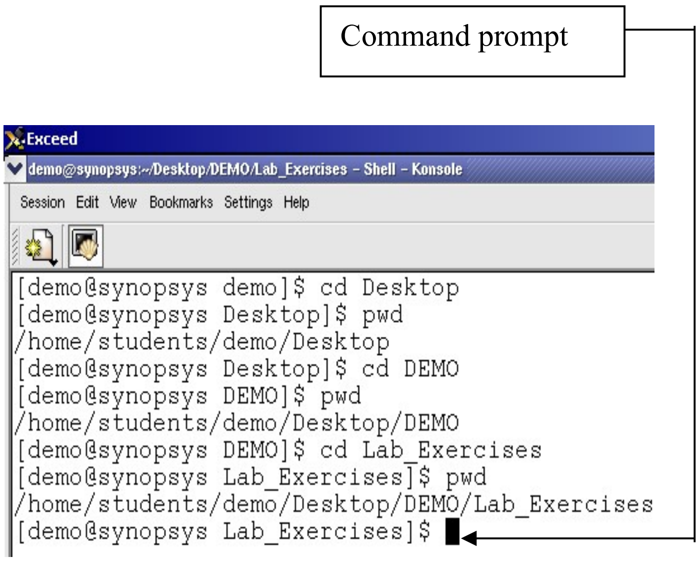


Figure 1.02 : Open Linux Shell here


Command prompt
Exceed
demo@synopsys:~/Desktop/DEMO/Lab_Exercises - Shell - Konsole
Session Edit View Bookmarks Settings Help

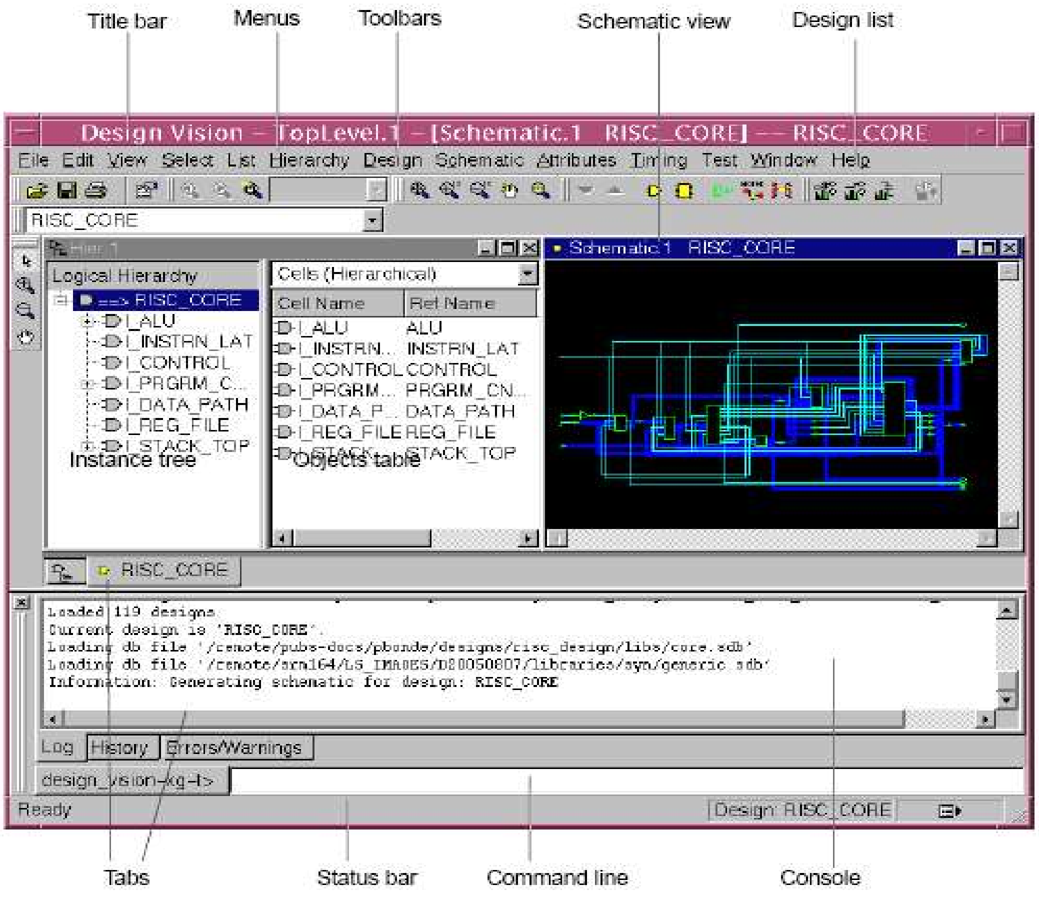
```
[demo@synopsys demo]$ cd Desktop
[demo@synopsys Desktop]$ pwd
/home/students/demo/Desktop
[demo@synopsys Desktop]$ cd DEMO
[demo@synopsys DEMO]$ pwd
/home/students/demo/Desktop/DEMO
[demo@synopsys DEMO]$ cd Lab_Exercises
[demo@synopsys Lab_Exercises]$ pwd
/home/students/demo/Desktop/DEMO/Lab_Exercises
[demo@synopsys Lab_Exercises]$ █
```

Figure 1.03 : Navigate to the proper directory

To start Design Vision, enter
% ***design_vision*** at the command prompt.

The Design Vision window, shown in Figure 1.04, is displayed. In addition, the design_vision-t> prompt appears in the shell where you started Design Vision and you are ready to begin the tutorial exercise. In this tutorial, all scripts are Tcl scripts. You use the dctcl command language to interact with Design Vision. You can exit Design Compiler from Design Vision at any time. Choose File > Exit from the menu bar and then click OK in the warning message box, or enter the exit command on the command line.

Note:
Design Compiler does not automatically save the designs that you work on. To save your designs, use the File > Save or File > Save As menu commands, or enter the write –format -db command on the command line.



Cell Name	Ref Name
I_ALU	ALU
I_INSTRN...	INSTRN_LAT
I_CONTROL	CONTROL
I_PRGRM...	PRGRM_CN...
I_DATA_P...	DATA_PATH
I_REG_FILE	REG_FILE
I_STACK...	STACK_TOP

```
Loaded 119 designs
Current design is 'RISC_CORE'.
Loading db file '/remote/pubs-docs/pbonde/designs/risc_design/libs/core.sdb'
Loading db file '/remote/srm164/LS_IMAGES/D20050807/libraries/syn/generic.sdb'
Information: Generating schematic for design: RISC_CORE
```

Figure 1.04: Design Vision GUI

1.2 Introduction to the Design Vision Interface

The window consists of a title bar, a menu bar, and several toolbars at the top of the window and a status bar at the bottom of the window. You use the workspace between the toolbars and the status bar to display view windows containing graphical and textual design information. When you start a Design Vision session, the following view windows appear in the workspace:

• Logical hierarchy view

The Logical hierarchy view helps you navigate your design and gather information. The view is divided into the following two panes:

- Instance tree, on the left
- Objects table, on the right

The instance tree lets you quickly navigate the design hierarchy and see the relationships among its levels. Each instance is an occurrence of a design loaded in Design Compiler memory. An instance is also known as a cell. If you select a hierarchical instance (an instance that contains other instances), information about that instance appears in the objects table. You can Shift-click or Control-click instance names to select combinations of instances.

By default, the objects table displays information about instances belonging to the selected instance in the instance tree. To display information about other types of objects, select the object types in the list above the objects table. You can display information about hierarchical cells, all cells, pins and ports, pins of child cells, and nets.

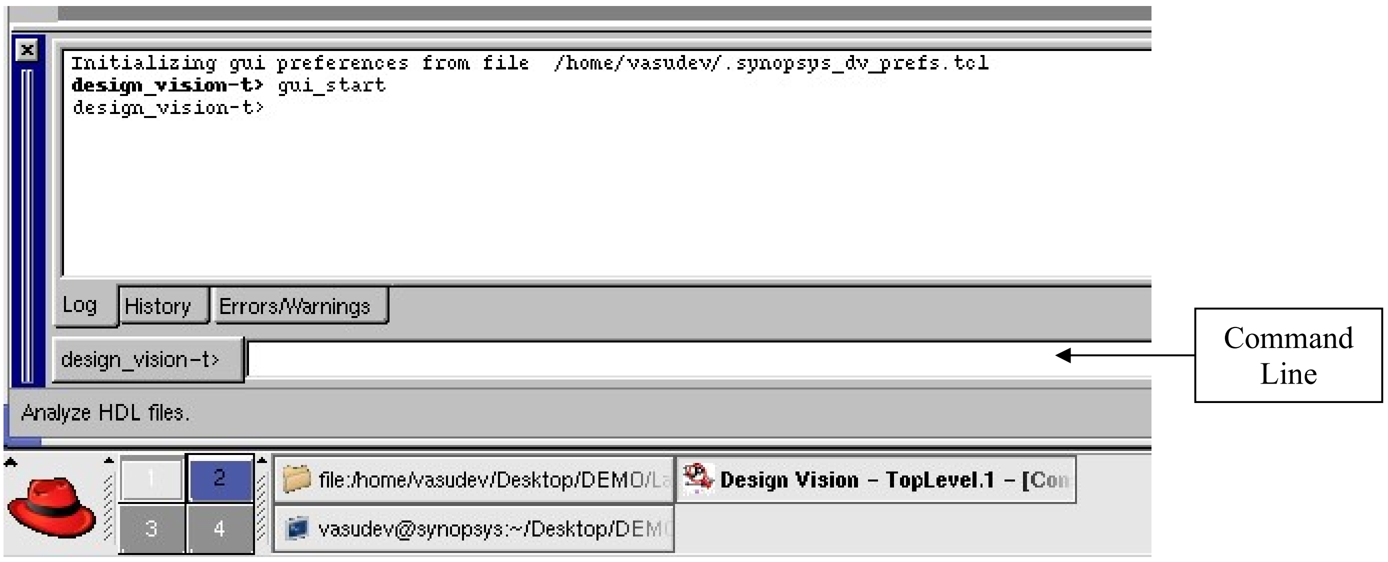
1.2.1 Exploring the Design Vision Interface

If some of the text in a column is missing because the column is too narrow, you can hold the pointer over it to display the information in an InfoTip. You can also adjust the width of a column by dragging its right edge left or right.

• Console

The console provides a command-line interface and displays information about the commands you use during the session in the following three views:

- Log view
- History view
- Errors and warnings view



```
Initializing gui preferences from file  /home/vasudev/.synopsys_dv_prefs.tcl
design_vision-t> gui_start
design_vision-t>
```

Log
History
Errors/Warnings
design_vision-t>
Command Line
Analyze HDL files.
file:/home/vasudev/Desktop/DEMO/L
Design Vision – TopLevel.1 – [Co
vasudev@synopsys:~/Desktop/DEM

Figure 1.05: Design vision Console

You can enter dctcl commands on the command line at the bottom of the console, shown in figure 1.03 above. Enter these commands just as you would enter them at the dctcl prompt in a standard UNIX or Linux shell. Design Vision echoes the dctcl command output (including processing messages and any warnings or error messages) in the console log view.

The log view provides the session transcript. The history view provides a list of the commands that you have used during the session. The errors and warnings view displays error and warning messages.

To select a view, click the tab at the bottom of the console. The log view is displayed by default when you start Design Vision.

Always keep an eye on the Errors/warnings view as it proves to be very useful for debugging purposes in case you end up getting stuck at some place.

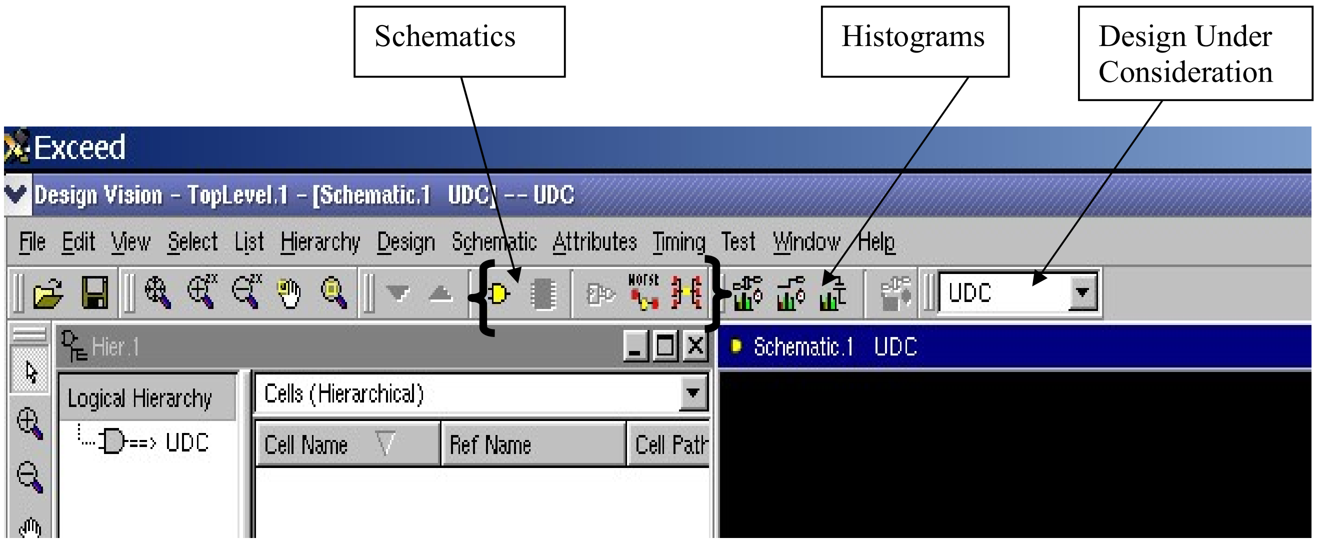

Schematics
Histograms
Design Under Consideration
Exceed
Design Vision - TopLevel.1 - [Schematic.1 UDC] -- UDC
File Edit View Select List Hierarchy Design Schematic Attributes Timing Test Window Help
UDC
Hier.1
Schematic.1 UDC
Logical Hierarchy
UDC
Cells (Hierarchical)
Cell Name
Ref Name
Cell Path

Figure 1.06: Histogram & Schematic options

In addition to the logical hierarchy view and the console, you can display design information in the following views:

- Schematic views (design schematics, path schematics, and symbol views)
- Histogram views (endpoint slack, path slack, and net capacitance)
- Report views
- List views (cells, ports, pins, nets, and designs)

Design Vision displays a tab at the bottom of the View Area for each undocked view you open. You can adjust the sizes of view windows for viewing purposes, and you can move them to different locations within the Design Vision window. When you click anywhere within a view window, Design Vision highlights its title bar to indicate that it has the focus (that is, it is the active view) and can receive keyboard and mouse input.

1.2.2 Accessing Manual Entries (man Pages)

You can get help in Design Vision from the man pages. These help systems provide a level of detailed information beyond that found in the tutorial exercises.As you carry out the tutorial exercises, use the man pages whenever you want more information about the interface and how to use it, or more information about Design Compiler commands.

Man Page Help

You can get man page help for any dc_shell command by choosing Help > Man Pages on the menu bar or entering either of the following commands on the console command line:

- man *command_name*
- *command_name* -help

1.3 Reading the Design

In this section, you will learn how to read in the counter design Verilog file, checking the Design, Examining the Design's Structure.

1.3.1 About the Design

This is a design of a four bit counter which can count either up or down and has a reset as well as option for the user to load data onto the counter. Invariably it has a clock too! In all it has 4 output ports and 8 input ports including the clock port.

1.3.2 Starting the Tutorial Exercise

You start the main tutorial exercise by invoking Design Vision from the lab_exercises directory.
To start the main exercise,

1. Navigate to the Desktop->lab_exercises directory.
2. Open a UNIX shell on your terminal from the directory.
3. At the shell prompt, enter

```
% design_vision
```

The Design Vision window appears. In addition, the design_vision-t> prompt (in dctcl command language) appears in the shell where you started Design Vision and you are ready to begin the tutorial exercise.

1.3.3 Reading the Counter Designs

To read in the HDL designs you can use the analyze and elaborate commands, or you can alternatively use the read_file command.
The analyze command checks the HDL designs for proper syntax and synthesizable logic, translates the design files into an intermediate format, and stores the intermediate files in the directory you specify (that is, the WORK directory).
The elaborate command first checks the intermediate format files before building the design. Then the command determines whether it has the necessary synthetic operators to replace the HDL operators. It also determines correct bus size.
In this tutorial, you use the Analyze and Elaborate menu commands to read in the Verilog designs from the source directory.

The other alternative to the analyze-elaborate command is the read command. This not only checks whether the code is free from syntax errors but also converts the code from its HDL format to a format, which is understandable to the tool. It prepares the code for synthesis. The difference being read command can be used to read in netlists as well as codes and meanwhile this command does not create any intermediate files whereas the analyze command does create some intermediate files such as .mr, .st, .syn files.

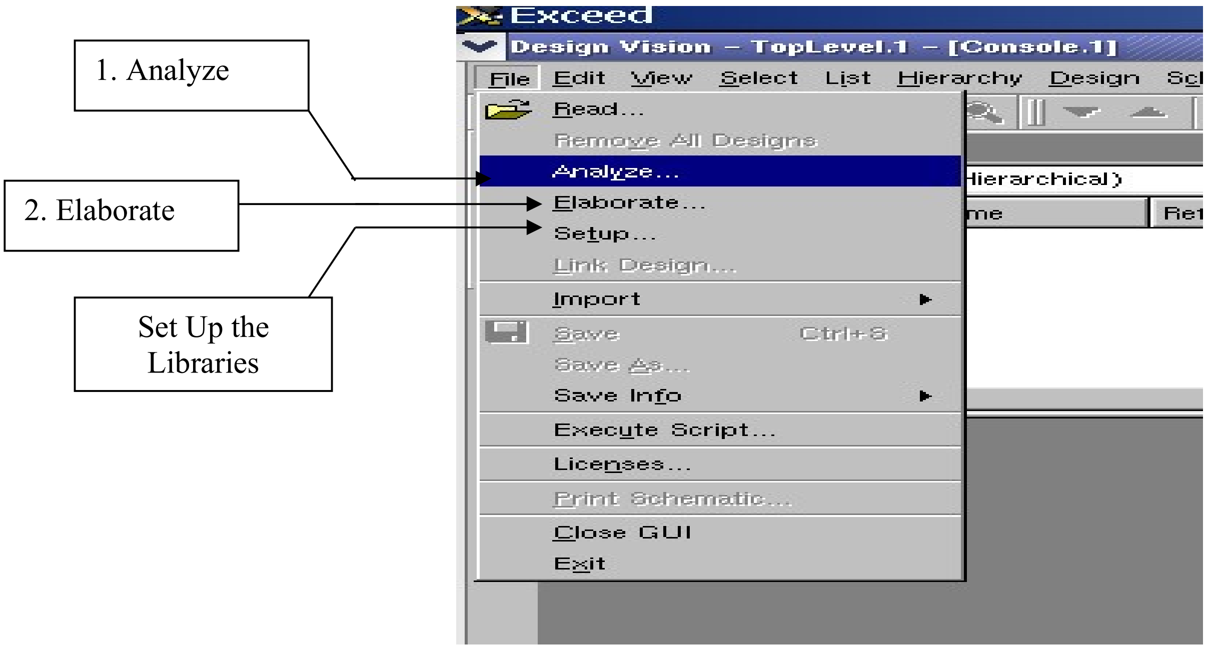


Figure 1.07 : Reading in the code and setting the libraries

Checking the design for Syntax errors and synthesis rules.

To ensure the syntax of the Verilog code

1. Choose File > Analyze.
   The Analyze Designs dialog box appears as shown in Figure 1.08 below.
2. Select Verilog in the Format list.
3. Click the Add button in the dialog box to open the Analyze Designs file browser.
4. Double click the source directory to open it.
5. Select udc.v from the file name list and click the Select button. The file browser closes.
6. Make sure the WORK library is selected in the Analyze Designs dialog box. (Scroll through the Work library list if necessary.)
7. Click OK.

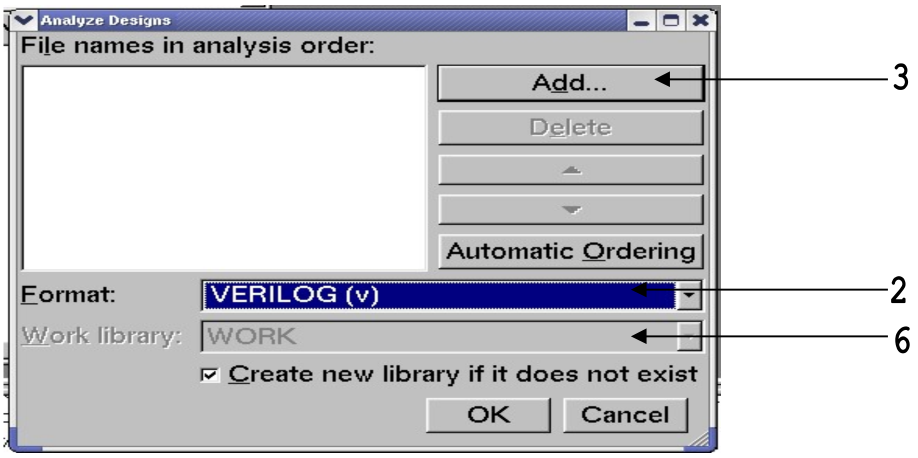


Figure 1.08 : Analyze Designs Dialog Box

Here the code is checked for syntax errors and converted into intermediate format for the next process i.e. "**elaborate**".

The console lists the files that were read into memory, including the libraries standard.sldb, gtech.db and the intermediate files are now stored in the WORK library.

Equivalent dctcl Commands

analyze -format verilog -lib WORK ./udc.v

Translating the Top-Level Design

After the design is analyzed, you elaborate the design

1. To elaborate the design, Choose File > Elaborate.
   The Elaborate Designs dialog box appears as shown in Figure 1.09.
2. Select the WORK library.
   The designs in the WORK library appear in the Design list.
3. Select the UDC design.
4. Make sure the Re-Analyze Out-of-Date Libraries option is selected.
5. Click OK.

The Elaborate menu command creates generic technology (GTECH) designs from the intermediate format files produced by the Analyze menu command. The console log view lists the design and inferred components.

When the elaborate process is done, the current design is the UDC design. This design name appears in the instance tree and in the Design list on the toolbar.

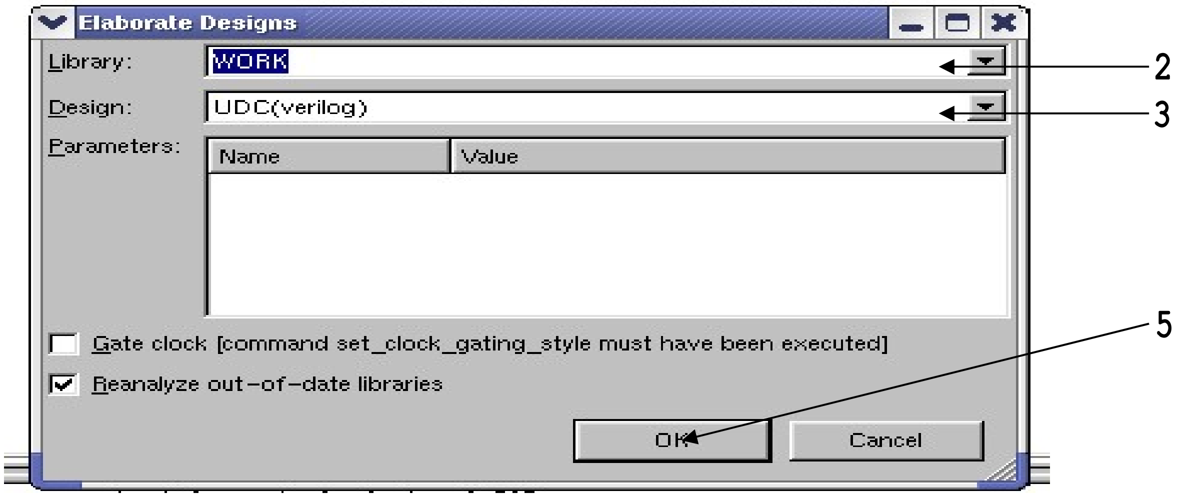
The current design is the active design (the design being worked on). Most commands are specific to the current design, that is, they operate within the context of the current design.

Note:

As part of the elaborate process, all designs are *autolinked*. You do not have to manually link the designs.

Equivalent dctcl Command

elaborate UDC -lib WORK -update



Library: WORK

Design: UDC(verilog)

Parameters:

[ ] Gate clock [command set_clock_gating_style must have been executed]

[✓] Reanalyze out-of-date libraries

Figure 1.09: To elaborate the design

Synthesis is a three step process namely Translate, Map & Optimize. The **elaborate** command does the translate part of the synthesis process where the code is translated to a format understandable by the tool, here using generic libraries. The schematic of the translated code can be viewed by clicking on the *"yellow and-gate"* symbol in the tool bar.

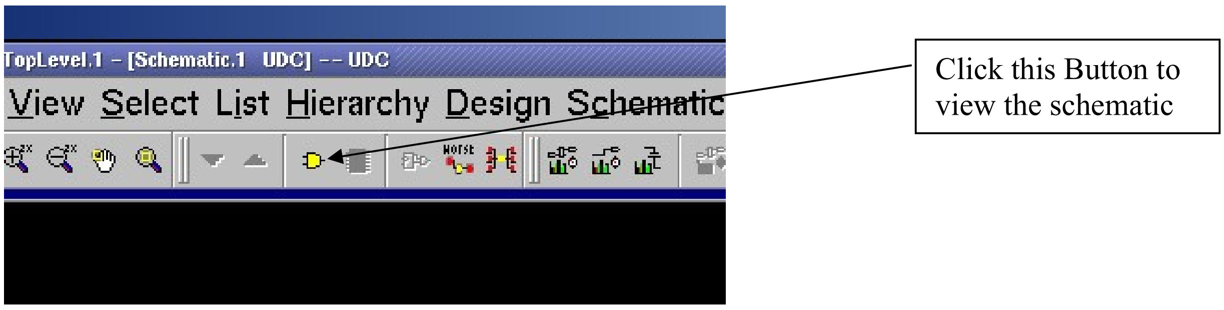

TopLevel.1 – [Schematic.1 UDC] -- UDC
View Select List Hierarchy Design Schematic
Click this Button to view the schematic

Figure 1.10: Tool Bar

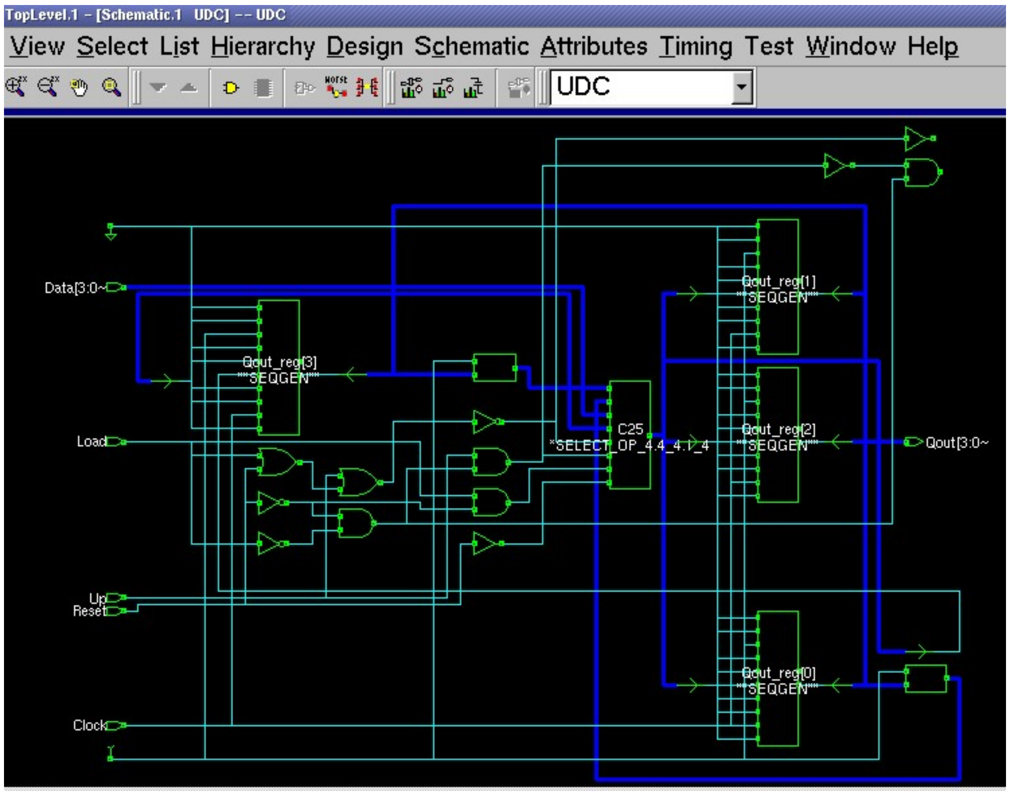


Figure 1.11: Technology Independent Schematic

The schematic after the elaborate process consists of components taken from the generic library which is specific to Synopsys and is technology independent. These cells have no area or timing attributes associated with them. On close observation you can see in the Figure 1.11 above some components named as "SEQGEN" which are basically SEQuential components from the GENeric library.

1.4 Setting the Libraries

Before proceeding further it is worthwhile to spend some time on Libraries. Now, we know that Design Compiler is used for generating a netlist from any given HDL Code, i.e some English language is converted to some gates, how does this happen? Where do these gates exist? The answer to this question lies in Libraries. These are files, which contain the information of the gates and their properties so before you convert the code to netlist you need to ensure that the proper libraries are loaded into the tool. You can load them by two ways one is manually by clicking on the "setup" button, shown in figure 1.05, and then navigating to the location where the libraries are and then selecting them or alternatively you need to have a setup file, which automatically loads the libraries for you when you invoke the tool. A typical setup file which should be in your working directory contains the following information

.synopsys_dc.setup file

```
      set search_path "/home/Vasudev/Desktop/DEMO/Lab_Exercises"

      set link_library   "/home/Master_Files/db/cb13fs120_tsmc_max.db"
      set target_library "/home/Master_Files/db/cb13fs120_tsmc_max.db"
      set symbol_library "/home/Master_Files/db/cb13fs120_icon.sdb"

      define_design_lib WORK -path ./WORK
```

Here search_path specifies the path where your designs exist. When you invoke the tool from the lab_exercises directory it searches for the .synopsy_dc.setup file and then looks for the libraries that have been defined in the link_library and target_library variable and automatically loads these into the memory. The "cb13fs120_tsmc_max.db" library has been set as the target library as well as the link library. And the related "cb13fs120_tsmc_icon.sdb" has been set as the symbol library from which the tool obtains the symbols during generation of schematic.

Note: Here the search path is pointing to the folder in my login, the path will be different for you, you need to ensure that your paths are set properly as per your working logins and directories.

1.5 Constraining the Design

Design constraints define physical conditions that restrict and guide the optimization process. They are applied as attributes with specific values to the designs in memory.

Constraints can be broadly classified into two categories that are optimization constraints and Design Rule Constraints, where the Design rule constraints are those specified by the vendor and to which a higher priority is associated, while the optimization constraints are the ones which the designer chooses so as to achieve the specific specifications of the design. Even without the designer explicitly specifying them design rule constraints (default values) are applied to the design by the libraries, the designer has the option of tightening the constraints but these cannot be relaxed. Alternatively optimization constraints need to be overtly applied on the circuit to extract better performance from it. There also exists a class of constraints, which form the environmental constraints, which are again specified by the fabrication house, and the designer has a choice of choosing a particular set amongst those that are provided by the foundry.

The constraints can be set by using the Attribute menu, entering constraint commands directly on the console command line, executing a script file of constraint commands, or using the three methods in any combination.

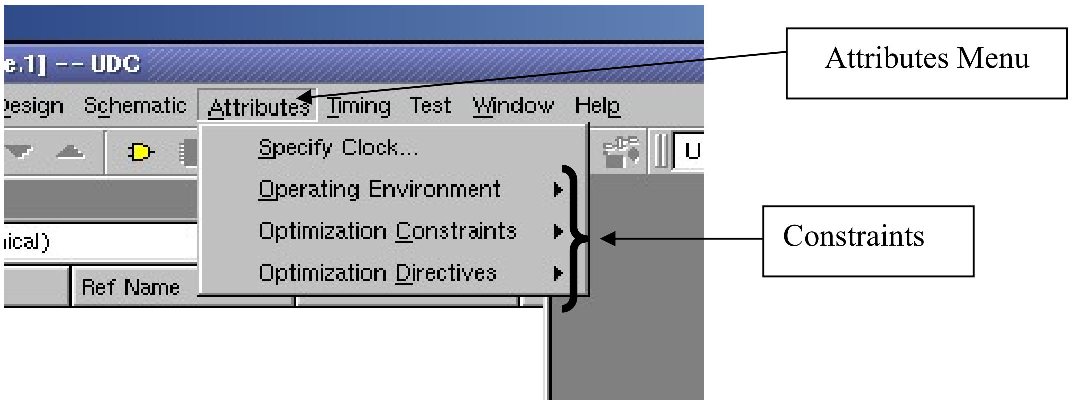
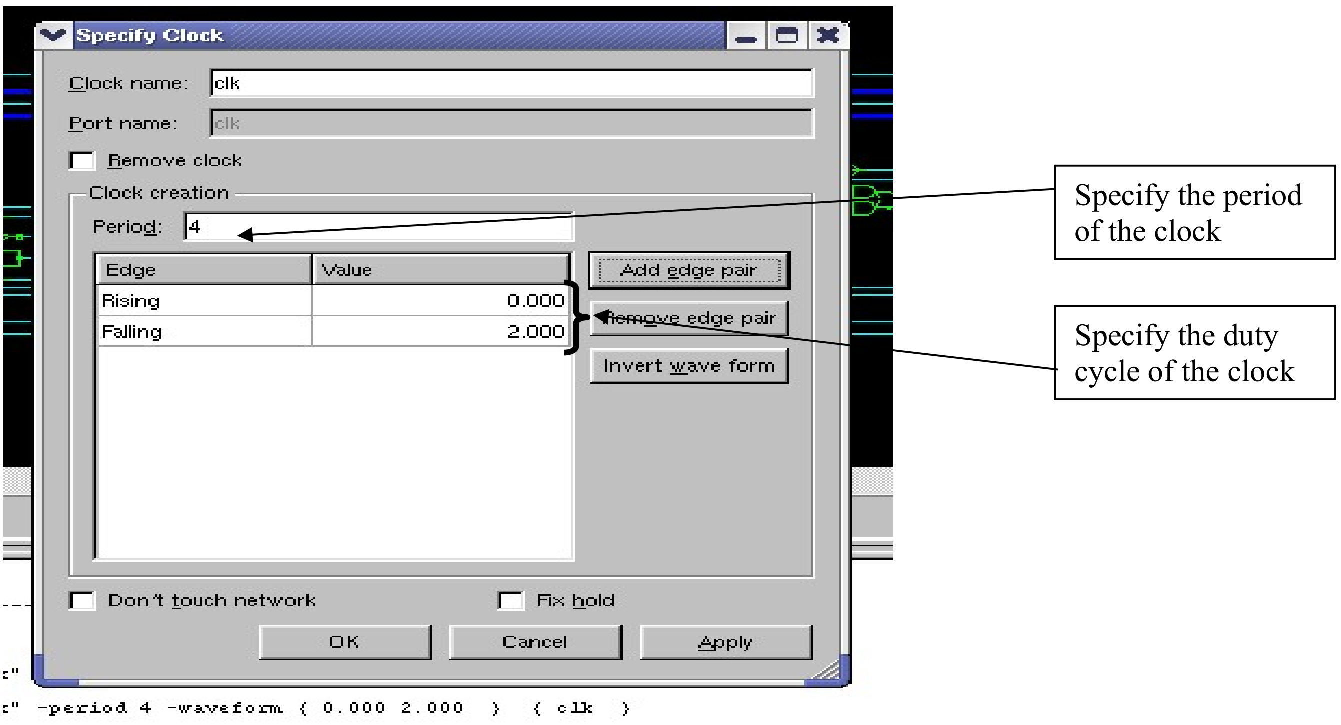


Figure 1.12: Attributes Menu

Figure 1.12 shows the attributes menu using which you can set some of the constraints, such as clock, maximum_area, maximum_power, input_delay, output_delay etc. (To be discussed in detail in Chapter 2)



Clock name: clk

Port name: clk

Period: 4

Edge	Value
Rising	0.000
Falling	2.000

```
:" -period 4 -waveform { 0.000 2.000 } { clk }
```

Figure 1.13: Defining the clock

You can specify the clock through GUI by first selecting the clock port of your design from the schematic, then click on "specify clock" in the attributes menu. Enter the period of the clock in the space provided for it, similarly you can also specify the rise time and fall time of the clock. After entering the appropriate values click on OK button.

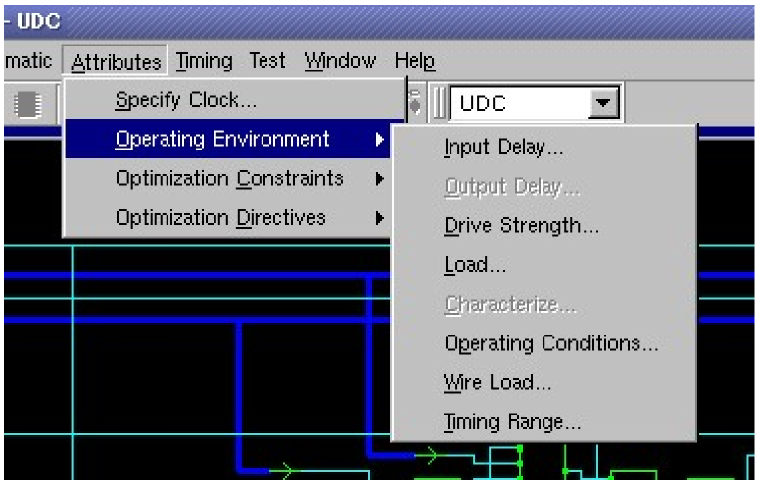


Figure 1.14 Applying environment constraints

Figure 1.14 shows the various environment constraints that are available in the menu list, but some of the other constraints such as Input Delay, Output delay, Load will not be highlighted until you select a port or pin in the schematic view. While the operating conditions, Wire Load can be selected even without any object selected in the schematic.

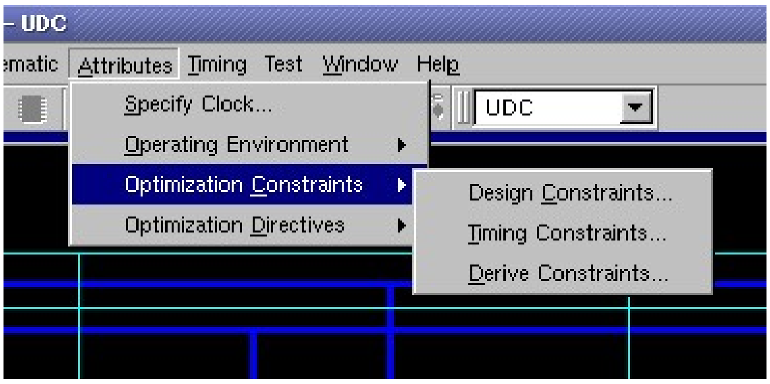


Figure 1.15: Applying optimization constraints.

The optimization constraints can also be applied on the design through the GUI, some of the constraints available in the design constraints are Maximum power, Maximum Fanout, maximum transition, etc.
Similarly you can also control the way the synthesis is performed by setting some optimization directives, such as don't touch etc.

So far in Chapter 1 you have learnt how to read in the code as well as the libraries, and a part of constraints was also discussed. Currently in the designing phase we have completed the translation of the code. Now the design is translated from its HDL format and it's Boolean equation representation is now mapped on to the generic library and the design is now ready to be targeted to the technology library.

Chapter 2

2.1 Synthesizing the design and generating reports

The process of converting a design from its technology independent representation to the technology dependent form is known as map - a fundamental step of the synthesis process. Only after the design is mapped a clear picture of the capabilities of the design is arrived at and the proper optimization constraints can be identified. In this chapter the various constraints and constraint methodology will be discussed.

2.1.1 The Map Process

Here the design refers the target library and chooses suitable components to generate a gate level representation of the design.
The command here is ***compile***.

The compile command targets the design from its present generic format to the technology library specified. The command compile is alone sufficient although it has a number of switches/options associated with it. Eg: In case the application requires more area optimization as compared to speed, during compilation more emphasis can be given for area optimization. As with almost all of the commands in DC even the map process can be done through GUI by clicking on the compile design option under the "Design" menu in the tool bar as shown in figure 2.01 below.

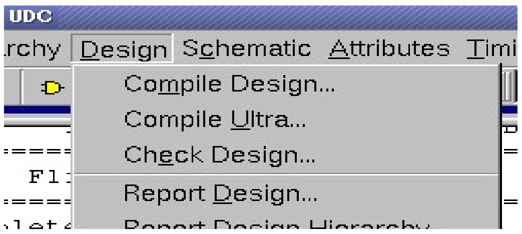
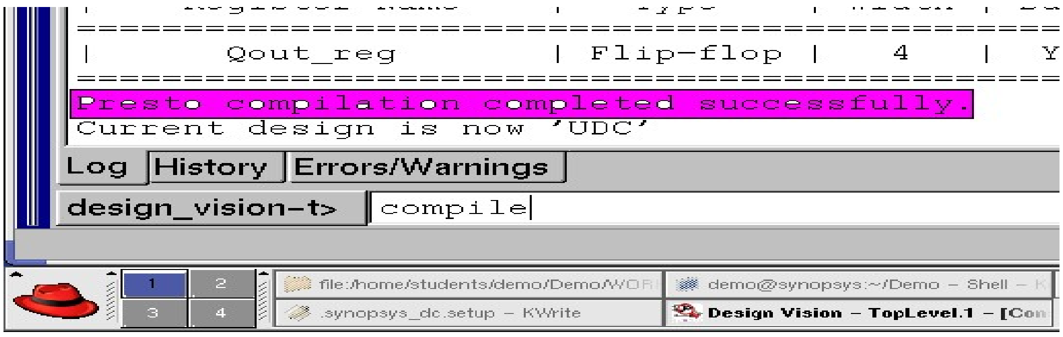


Figure 2.01

Along with the simple compile there is another option of compile Ultra which is a more optimized way of synthesizing a design but it would be too early to discuss it at this stage of our process, so let us leave it for later discussions.

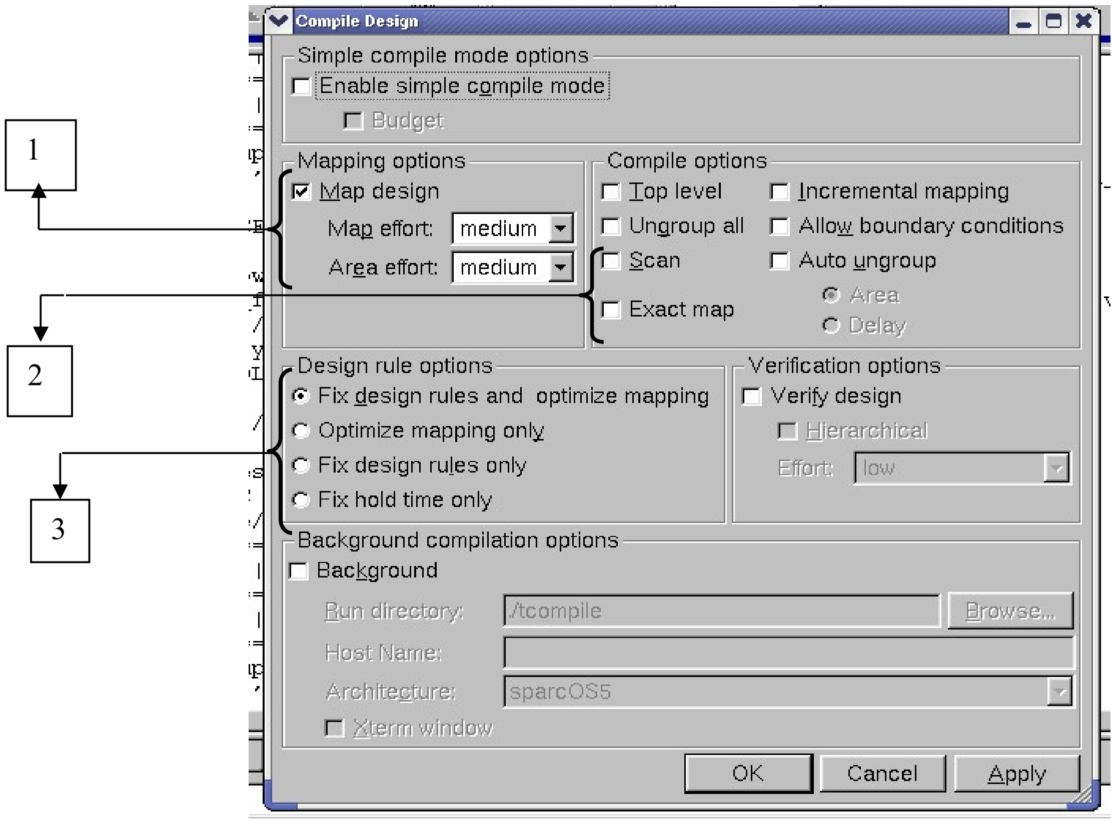


```
==================================================
|        Qout_reg        | Flip-flop |    4    |   Y
==================================================
Presto compilation completed successfully.
Current design is now 'UDC'
```

Log
History
Errors/Warnings
design_vision-t>
compile
file:/home/students/demo/Demo/WOR
demo@synopsys:~/Demo - Shell
.synopsys_dc.setup - KWrite
Design Vision - TopLevel.1 - [Con

Figure 2.02 : Map the design

When the compile command is given through the command line as shown in figure 2.02 above the default options associated with compile is effected on the design. As you can observe in the figure prior to the Map process the design compilation should have completed successfully (highlighted in pink) and ensure that the current design is the one that you intend to compile.



Compile Design

Simple compile mode options
- [ ] Enable simple compile mode
- [ ] Budget

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

Compile options
- [ ] Top level
- [ ] Incremental mapping
- [ ] Ungroup all
- [ ] Allow boundary conditions
- [ ] Scan
- [ ] Auto ungroup
- [✓ Area] Delay
- [ ] Exact map

Design rule options
- [✓ Fix design rules and optimize mapping]
- Optimize mapping only
- Fix design rules only
- Fix hold time only

Verification options
- [ ] Verify design
- [ ] Hierarchical
- Effort: low

Background compilation options
- [ ] Background
- Run directory: ./tcompile
- Host Name:
- Architecture: sparcOS5
- [ ] Xterm window

OK Cancel Apply

Figure 2.03: Some more options for compile

In figure 2.03 above you can see some of the options that are available during synthesis.

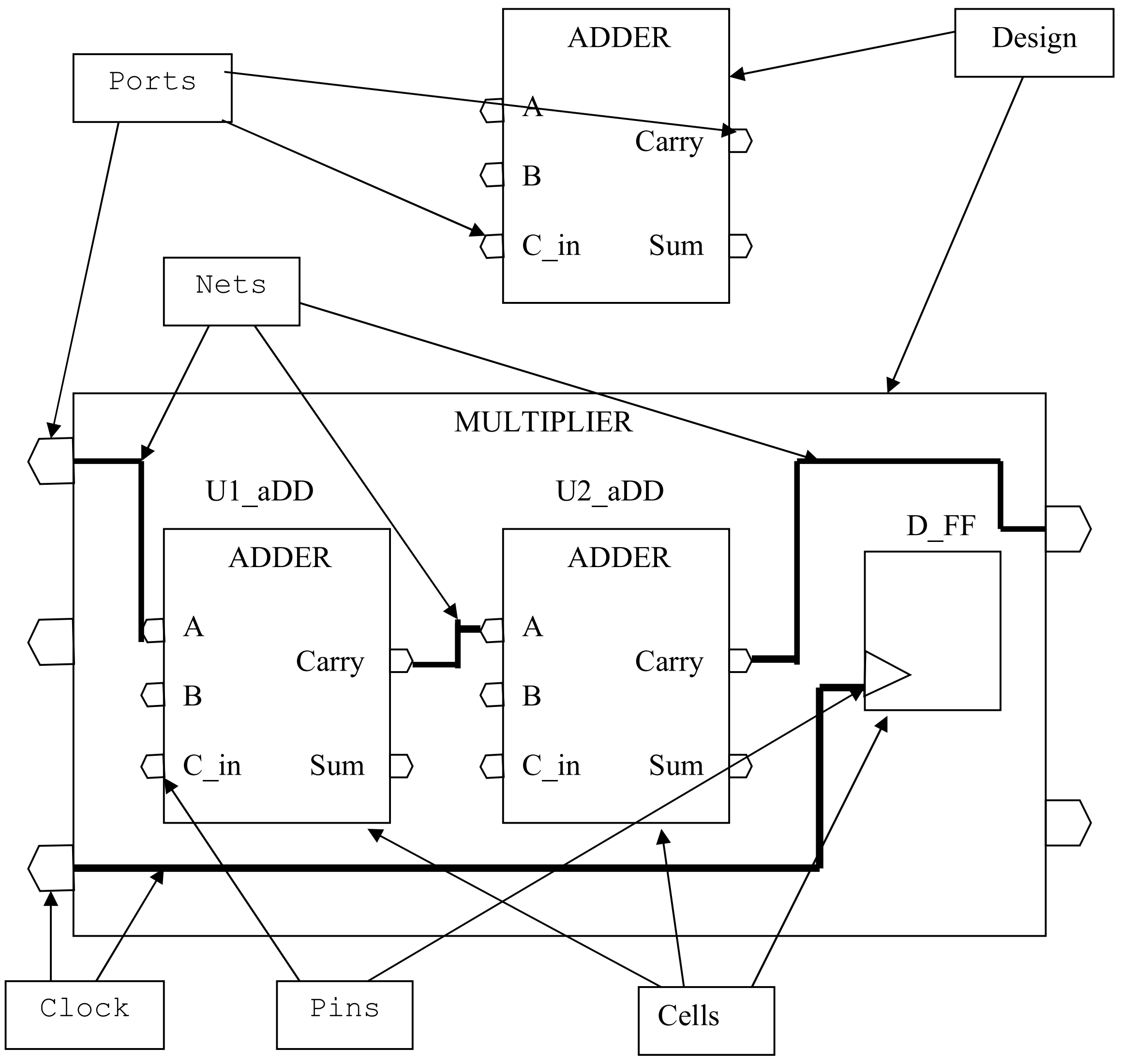
1. The design can be optimized either for area or for speed, by default the values for these are medium but depending on the requirement, either of them can be set to low, medium or high.
2. During the synthesis process the tool can be instructed to do the scan insertion for DFT purposes, and the tool can also be instructed to carry out a synthesis, which results in the sequential elements of the gate level design to be exactly similar to the way they are coded.
3. During optimization whether only design rules need to be fixed or even optimization needs to be carried out can be controlled using the design rule options.

The Map process fundamentally is again carried out in sections where first the code is directly mapped onto the target library, then area optimization is done, after which timing optimization is done, following which, an area recovery process is done. Depending on whether any power related constraints and DRC related constraints are applied, map process executes them also.

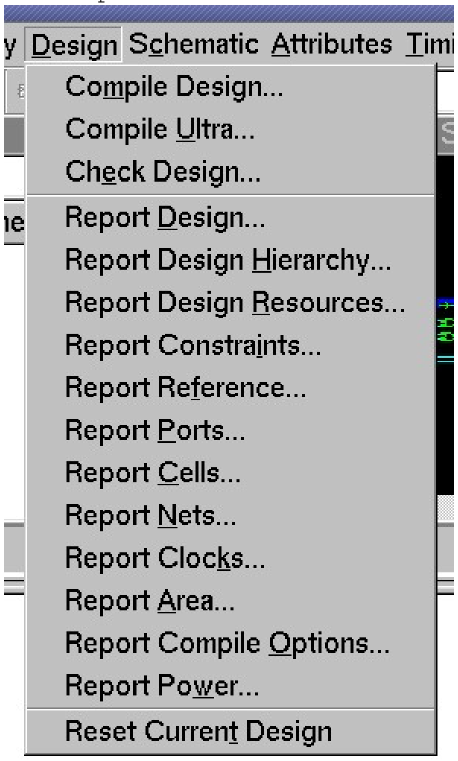
The command compile essentially completes the synthesis process by mapping the design from its technology independent format to the technology dependent format. Now we have a netlist in hand which basically speaks of the capability of the design, from which we can get a fair idea of the speed of the design, its area and the power required. From this we can now zero in to get the exact or nearly accurate constraints, which will further help in realizing a more optimized circuit, which is nearer to meeting our goals.

For synthesis any design is considered as a collection of a number of objects. There are 7 types of objects namely

1.) Design : As the name suggests this is the design under consideration
2.) Ports : All the external interfaces of a design are known as ports
3.) Pins : All the interfaces of individual cells are known as pins (or instances of a design in other design – as you can see below the ports of Adder in context of multiplier are known as pins).
4.) Nets : The interconnects in the design are known as nets.
5.) Clock : The clock port, clock net and clock pin of sequential cells are all classified as design object Clock
6.) Cell : The sub-components of a design are known as cells, Eg: In this case the D flip-flop as well as the two adders U1_ aDD and U2_aDD are cells.
7.) Reference: The original object to which the cell refers is the reference, here the ADDER design is the reference to which the cells U1_aDD and U2_aDD are referring to.



Once the design has been mapped onto the technology library now the performance of the design before constraining it can be gauged through the numerous "report" commands that are available.

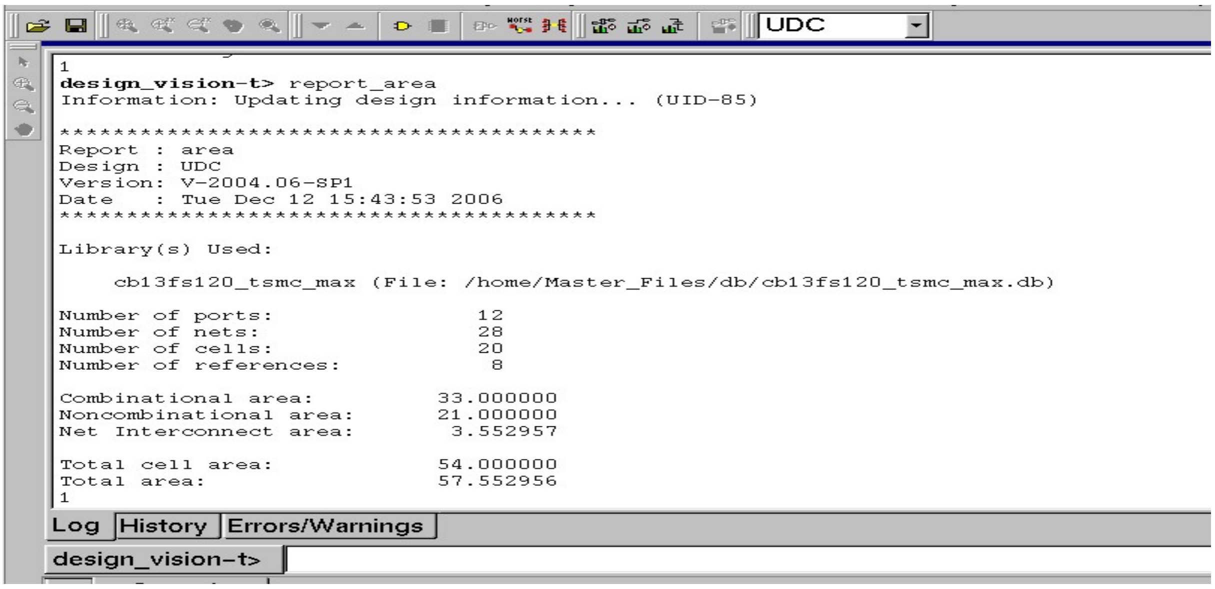


These reports return the various characteristics of the design such as its area, the power consumed, the number of cells, ports, pins I the Design. The design hierarchy and he resources it calls for. With this information in mind a fair estimate of the appropriate constraints can be arrived at.

But some of the reports such as constraints report and clock report can be obtained only after they have been previously set on the design.

Figure 2.04: Various reports that can be obtained.

Through the report_area command the total area occupied by the cells can be obtained, the report gives a break up of area occupied by sequential cells, purely combinational cells as well as interconnects as shown in figure below. The interconnect area is obtained from the wire load model information.



```
1
design_vision-t> report_area
Information: Updating design information... (UID-85)

****************************************
Report : area
Design : UDC
Version: V-2004.06-SP1
Date   : Tue Dec 12 15:43:53 2006
****************************************

Library(s) Used:

    cb13fs120_tsmc_max (File: /home/Master_Files/db/cb13fs120_tsmc_max.db)

Number of ports:                12
Number of nets:                 28
Number of cells:                20
Number of references:            8

Combinational area:          33.000000
Noncombinational area:       21.000000
Net Interconnect area:        3.552957

Total cell area:             54.000000
Total area:                  57.552956
1
```

Figure 2.05 : Area Report from GUI

The report area command returns the total number of cells. To know more about the kind of cells that the design has utilized the report_cells command is used.

Similar to the ‘report’ series of commands there are the “get” category of commands that are used to fetch a collection E.g. get_cells returns all the cells in the design. Our design being a 4-bit counter design you can view four flops and some other gates. Remember that any gate ending with the name reg is a sequential element and almost always a flip-flop.

```
design_vision-t> get_cells
{"Qout_reg[1]", "Qout_reg[2]", "Qout_reg[0]", "Qout_reg[3]", "U13", "U14", "U15",
"U16", "U17", "U18", "U19", "U20", "U21", "U22", "U23", "U24", "U25", "U26",
"U27", "U28"}
```

```
design_vision-t> get_pins
{"Qout_reg[1]/CP", "Qout_reg[1]/D", "Qout_reg[1]/Q", "Qout_reg[2]/CP",
"Qout_reg[2]/D", "Qout_reg[2]/Q", "Qout_reg[0]/CP", "Qout_reg[0]/D",
"Qout_reg[0]/Q", "Qout_reg[3]/CP", "Qout_reg[3]/D", "Qout_reg[3]/Q", "U13/B2",
"U13/B1", "U13/A2", "U13/A1", "U13/Z", "U14/A1", "U14/A2", "U14/A3", "U14/Z",
"U15/A1", "U15/A2", "U15/B1", "U15/B2", "U15/Z", "U16/A1", "U16/A2", "U16/ZN",
"U17/I", "U17/ZN", "U18/B2", "U18/B1", "U18/A2", "U18/A1", "U18/Z", "U19/A1",
"U19/A2", "U19/A3", "U19/Z", "U20/A", "U20/B", "U20/CI", "U20/CO", "U21/I",
"U21/ZN", "U22/B2", "U22/B1", "U22/A2", "U22/A1", "U22/Z", "U23/A1", "U23/A2",
"U23/A3", "U23/Z", "U24/I", "U24/ZN", "U25/B2", "U25/B1", "U25/A2", "U25/A1",
"U25/Z", "U26/A1", "U26/A2", "U26/Z", "U27/I", "U27/ZN", "U28/A1", "U28/A2",
"U28/ZN"}
```

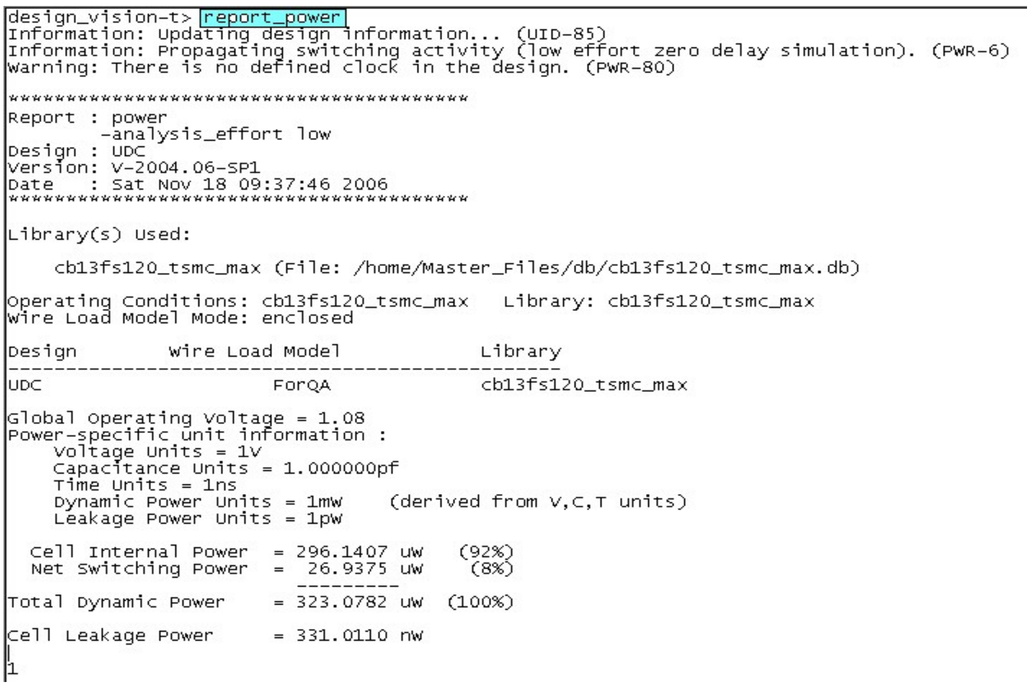
```
design_vision-t> get_ports
{"Qout[3]", "Qout[2]", "Qout[1]", "Qout[0]", "Data[3]", "Data[2]", "Data[1]",
"Data[0]", "Up", "Load", "Reset", "Clock"}
```

The power consumption of the design can be checked with the report_power command, which gives the dynamic as well as static power of the circuit. But for carrying out power optimization power compiler license is required, which is not bundled with the Design Compiler Licensing.

For calculating the power activity in the interconnects the information is gathered from the wire load models. It needs to be remembered here that for an precise calculation of power the switching activity of the design needs to be informed to the tool, failing which it makes only an rough estimate of the switching power depending on some default values. And also for sequential designs it is essential that the clock frequency be defined before calculation of the power without which the default timing unit from the library is taken.

The power report of our design is as seen in figure 2.06 below. As the clock has not yet been defined you can observe a warning mentioning that clock has not yet been defined.

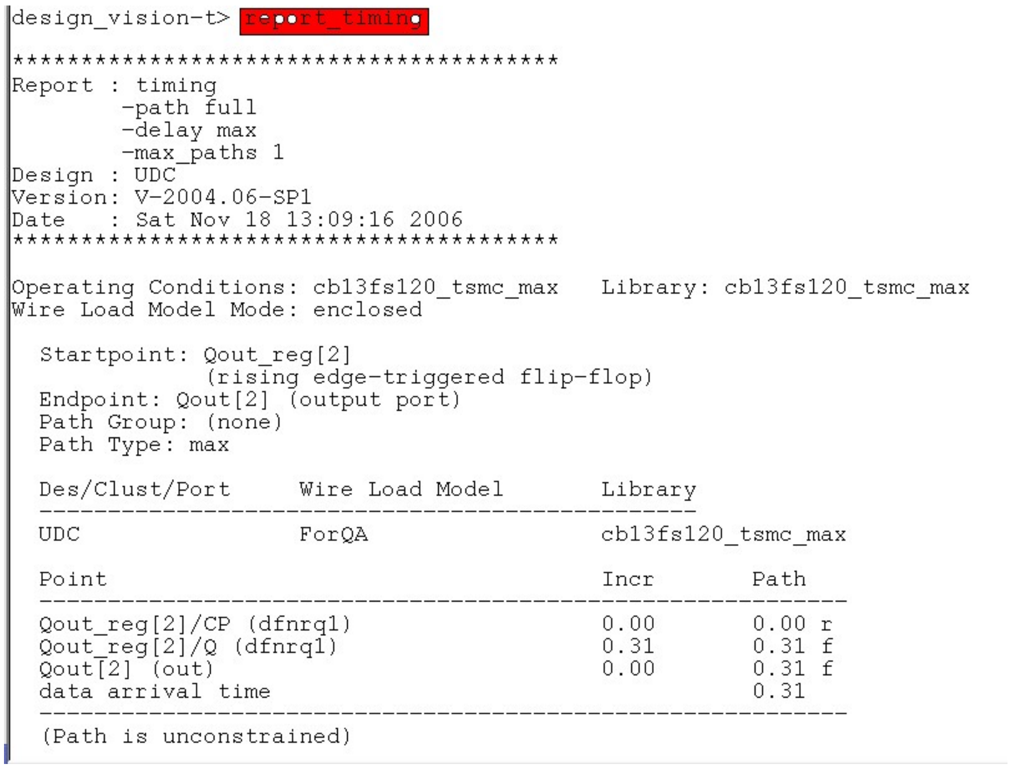
As this is not yet the optimized design, we need to optimize this design and the reasonable power constraints can now be arrived at.



```
design_vision-t> report_power
Information: Updating design information... (UID-85)
Information: Propagating switching activity (low effort zero delay simulation). (PWR-6)
Warning: There is no defined clock in the design. (PWR-80)

****************************************
Report : power
        -analysis_effort low
Design : UDC
Version: V-2004.06-SP1
Date   : Sat Nov 18 09:37:46 2006
****************************************

Library(s) Used:

    cb13fs120_tsmc_max (File: /home/Master_Files/db/cb13fs120_tsmc_max.db)

Operating Conditions: cb13fs120_tsmc_max   Library: cb13fs120_tsmc_max
Wire Load Model Mode: enclosed

Design        Wire Load Model            Library
------------------------------------------------
UDC                    ForQA             cb13fs120_tsmc_max

Global Operating Voltage = 1.08
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW

  Cell Internal Power  = 296.1407 uW   (92%)
  Net Switching Power  =  26.9375 uW    (8%)
                         ---------
Total Dynamic Power    = 323.0782 uW  (100%)

Cell Leakage Power     = 331.0110 nW

1
```

Figure 2.06: Power Report from Shell window

The most important information that we look for in any design is its timing capabilities, and this is obtained by the command ***report_timing.***



```
design_vision-t> report_timing

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : UDC
Version: V-2004.06-SP1
Date   : Sat Nov 18 13:09:16 2006
****************************************

Operating Conditions: cb13fs120_tsmc_max   Library: cb13fs120_tsmc_max
Wire Load Model Mode: enclosed

  Startpoint: Qout_reg[2]
              (rising edge-triggered flip-flop)
  Endpoint: Qout[2] (output port)
  Path Group: (none)
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  UDC                ForQA                 cb13fs120_tsmc_max

  Point                                    Incr       Path
  -----------------------------------------------------------
  Qout_reg[2]/CP (dfnrq1)                  0.00       0.00 r
  Qout_reg[2]/Q (dfnrq1)                   0.31       0.31 f
  Qout[2] (out)                            0.00       0.31 f
  data arrival time                                   0.31
  -----------------------------------------------------------
  (Path is unconstrained)
```

Figure 2.07: Timing Report from Shell window

Now timing being the most significant aspect of synthesis there are lots of switches available with the ***report_timing*** command. In its basic format report_timing returns the timing information of the path with most delay. It returns only one path per path group. And it gives out only the information of maximum delay. If the path is constrained it reports both the required time as well as arrival time, but for an unconstrained design it reports only the actual delay taken by the path and hence in figure 2.07 above "path is unconstrained" is displayed at the end, of the timing report.

Before we delve more into constraints it is important to understand the fact that synthesis is a timing driven process where the most important criteria that every synthesis tool tries to meet is timing related. So almost all the constraints that are applied implicitly or explicitly affect the timing of the circuit. Hence before looking into constraints it is imperative to know about the basic concepts of timing.

For any timing analysis there are few concepts that need to be understood which are:

1.)	Timing path	: This is the path taken by data to travel from one point to another in the design.
2.)	Start point	: The point at which any data can originate E.g.: Input ports or Clock pins of sequential elements.
3.)	End point	: That location in a design where a data is supposed to stop. E.g. Output ports or Data pins of sequential elements.
4.)	Arrival time	: The time taken by data to traverse through a timing path, due to the actual delays associated with the elements of the path.
5.)	Required time	: The time at which data is supposed to arrive at the end point. This requirement exists on the path due to the constraints applied by the designer.
6.)	Slack	: The difference between arrival time and required time.
7.)	Critical path	: The path with the maximum delay.

It is advisable to take some time and understand these concepts clearly and only when you are clear with them you can proceed confidently into synthesis and optimization. Further for any timing analysis the design can be considered as a combination of various timing paths which are as follows

1.)	Input to register	: All paths of sequential designs starting from the input port and ending at data pins of sequential elements.
2.)	Input to Output	: This type of path represents paths with purely combinational elements.
3.)	Register to register	: Purely sequential paths.
4.)	Register to Output	: All paths to the output ports from sequential elements.

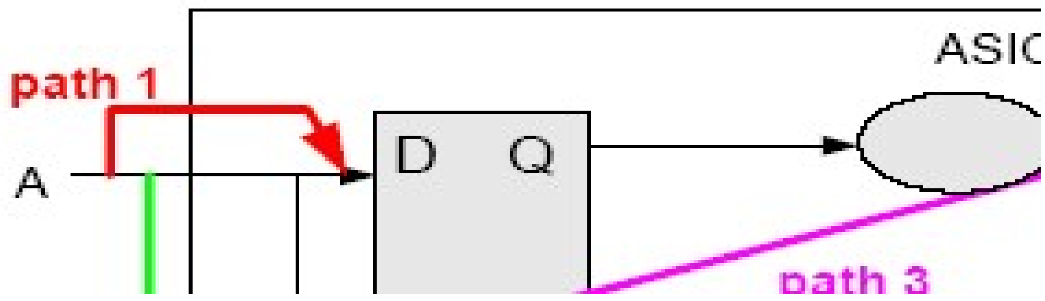


Figure 2.08 : Different timing paths.

These different timing paths are constrained in various methods

1.) The sequential path [3] is mainly constrained by the clock of the design which implicitly sets a timing constraint on the path.(Defining the period of the clock was discussed in section 1.5 before - where the figure 1.13 explained as to how the clock could be defined using the GUI.) The same is done through the command line by the command "***create_clock***",
2.) The purely combinational path [2] is constrained explicitly for timing. This is done using commands such as "***set_max_delay***" and "***set_min_delay***".
3.) The input to register path [1] is also implicitly constrained by the clock of the design.
4.) The register to output path [4] is generally unconstrained and can not be constrained by the clock unless there exists a specific requirement for the data to arrive at the output w.r.t the clock.

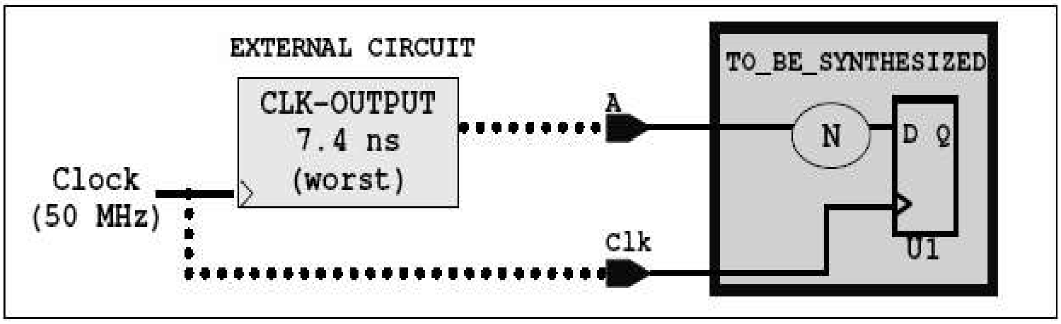


Figure 2.09 : A delay before the input port.

In figure 2.09 above there is some logic through which data has to travel before arriving at the design to be synthesized, due to the delay of this external logic, data may arrive at U1 later than when it is expected hence if information of this external delay is supplied before synthesis the logic N will be optimized so that data at U1 arrives in time, this is done using the command "***set_input_delay***".

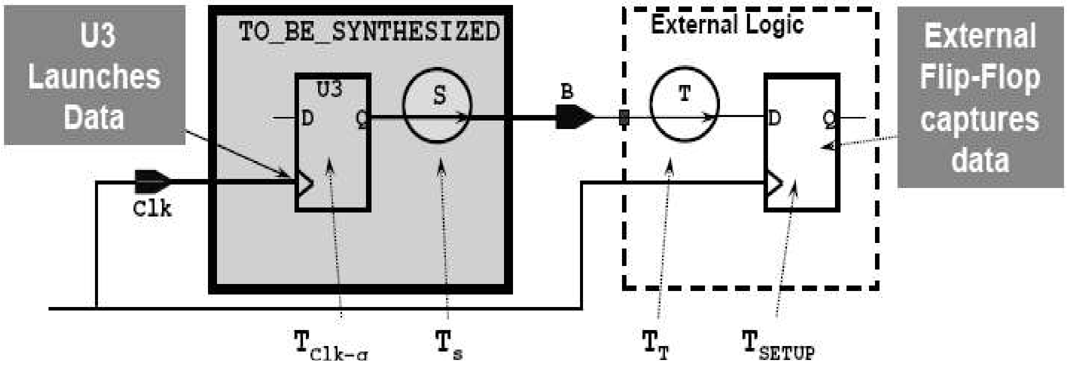


Figure 2.10 : An external delay at the output port.

In figure 2.10 above there is some logic through which data has to travel after exiting from the synthesized design, due to the delay of this external logic, data may arrive at external flop later than when it is expected hence if information of this external delay is supplied before synthesis the logic S will be optimized so that data at external flop arrives in time, this is done using the command "***set_output_delay***".

The create_clock command specifies the frequency of the clock as well as its waveform but it models only an ideal clock, whereas the real clock may arrive early or later than the ideal clock (clock skew). These conditions need to be replicated during designing so that there is no major variation in performance from the obtained design after the synthesis stage and the Place and route stage.

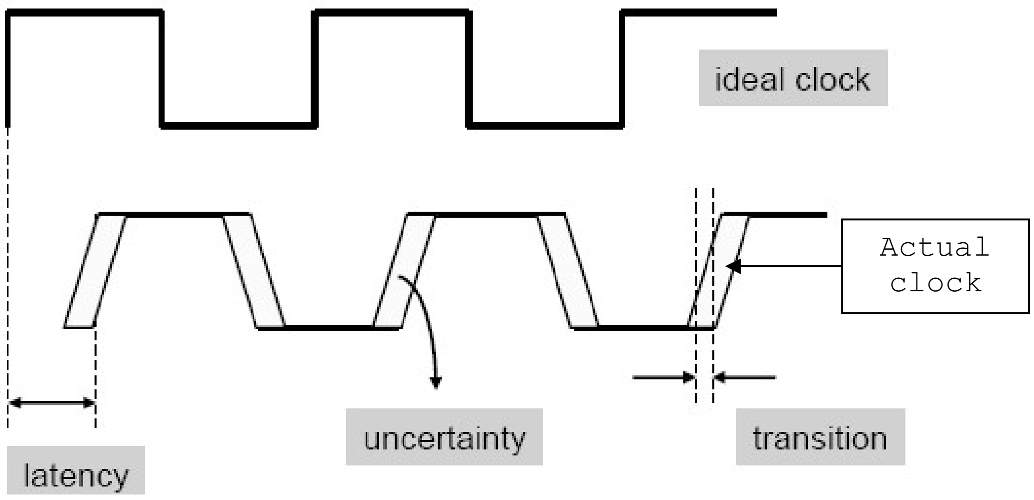
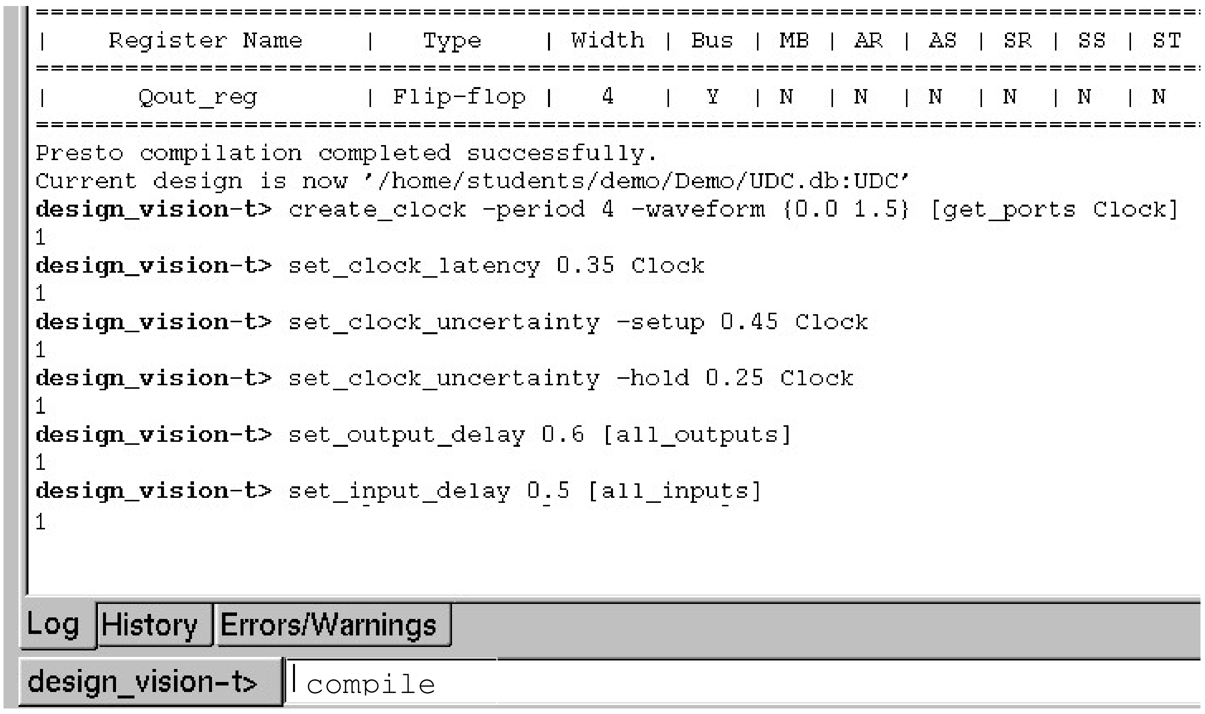


Figure 2.11 : Clock skew and latency

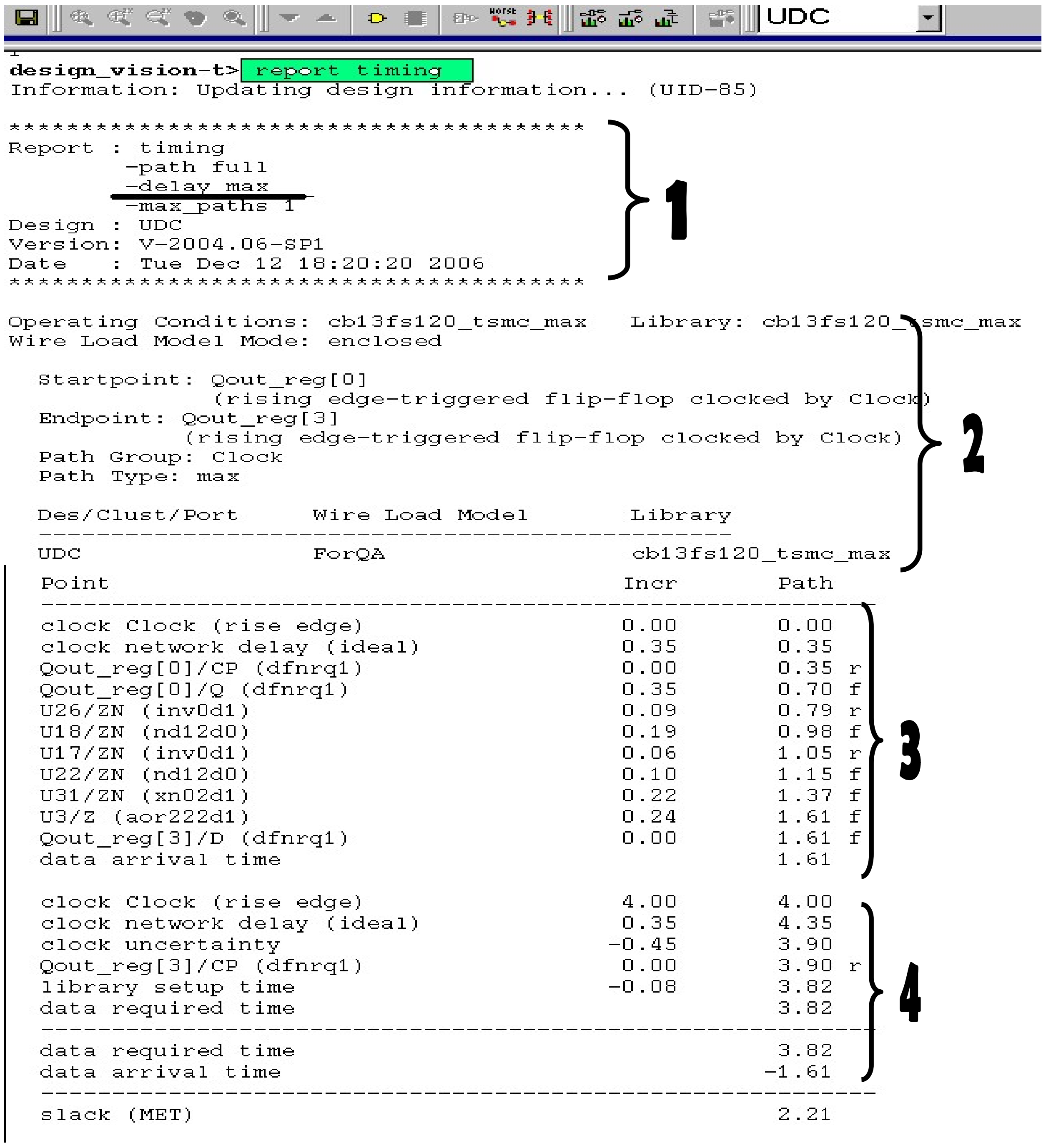
The clock related information is given using the set_clock_latency and set_clock_uncertainty as well as set_clock_transition commands.



```
===========================================================================
|    Register Name     |    Type    | Width | Bus | MB | AR | AS | SR | SS | ST
===========================================================================
|      Qout_reg        | Flip-flop |   4   |  Y  | N  | N  | N  | N  | N  | N
===========================================================================
Presto compilation completed successfully.
Current design is now '/home/students/demo/Demo/UDC.db:UDC'
design_vision-t> create_clock -period 4 -waveform {0.0 1.5} [get_ports Clock]
1
design_vision-t> set_clock_latency 0.35 Clock
1
design_vision-t> set_clock_uncertainty -setup 0.45 Clock
1
design_vision-t> set_clock_uncertainty -hold 0.25 Clock
1
design_vision-t> set_output_delay 0.6 [all_outputs]
1
design_vision-t> set_input_delay 0.5 [all_inputs]
1
```

Figure 2.12 : Constraining the design

Figure 2.12 highlights some of the constraints that can be set on the design.



```
design_vision-t> report timing
Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : UDC
Version: V-2004.06-SP1
Date   : Tue Dec 12 18:20:20 2006
****************************************

Operating Conditions: cb13fs120_tsmc_max   Library: cb13fs120_tsmc_max
Wire Load Model Mode: enclosed

  Startpoint: Qout_reg[0]
              (rising edge-triggered flip-flop clocked by Clock)
  Endpoint: Qout_reg[3]
            (rising edge-triggered flip-flop clocked by Clock)
  Path Group: Clock
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  UDC                ForQA                 cb13fs120_tsmc_max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock Clock (rise edge)                  0.00       0.00
  clock network delay (ideal)              0.35       0.35
  Qout_reg[0]/CP (dfnrq1)                  0.00       0.35 r
  Qout_reg[0]/Q (dfnrq1)                   0.35       0.70 f
  U26/ZN (inv0d1)                          0.09       0.79 r
  U18/ZN (nd12d0)                          0.19       0.98 f
  U17/ZN (inv0d1)                          0.06       1.05 r
  U22/ZN (nd12d0)                          0.10       1.15 f
  U31/ZN (xn02d1)                          0.22       1.37 f
  U3/Z (aor222d1)                          0.24       1.61 f
  Qout_reg[3]/D (dfnrq1)                   0.00       1.61 f
  data arrival time                                   1.61

  clock Clock (rise edge)                  4.00       4.00
  clock network delay (ideal)              0.35       4.35
  clock uncertainty                       -0.45       3.90
  Qout_reg[3]/CP (dfnrq1)                  0.00       3.90 r
  library setup time                      -0.08       3.82
  data required time                                  3.82
  -----------------------------------------------------------
  data required time                                  3.82
  data arrival time                                  -1.61
  -----------------------------------------------------------
  slack (MET)                                         2.21
```

Figure 2.13: Timing Report

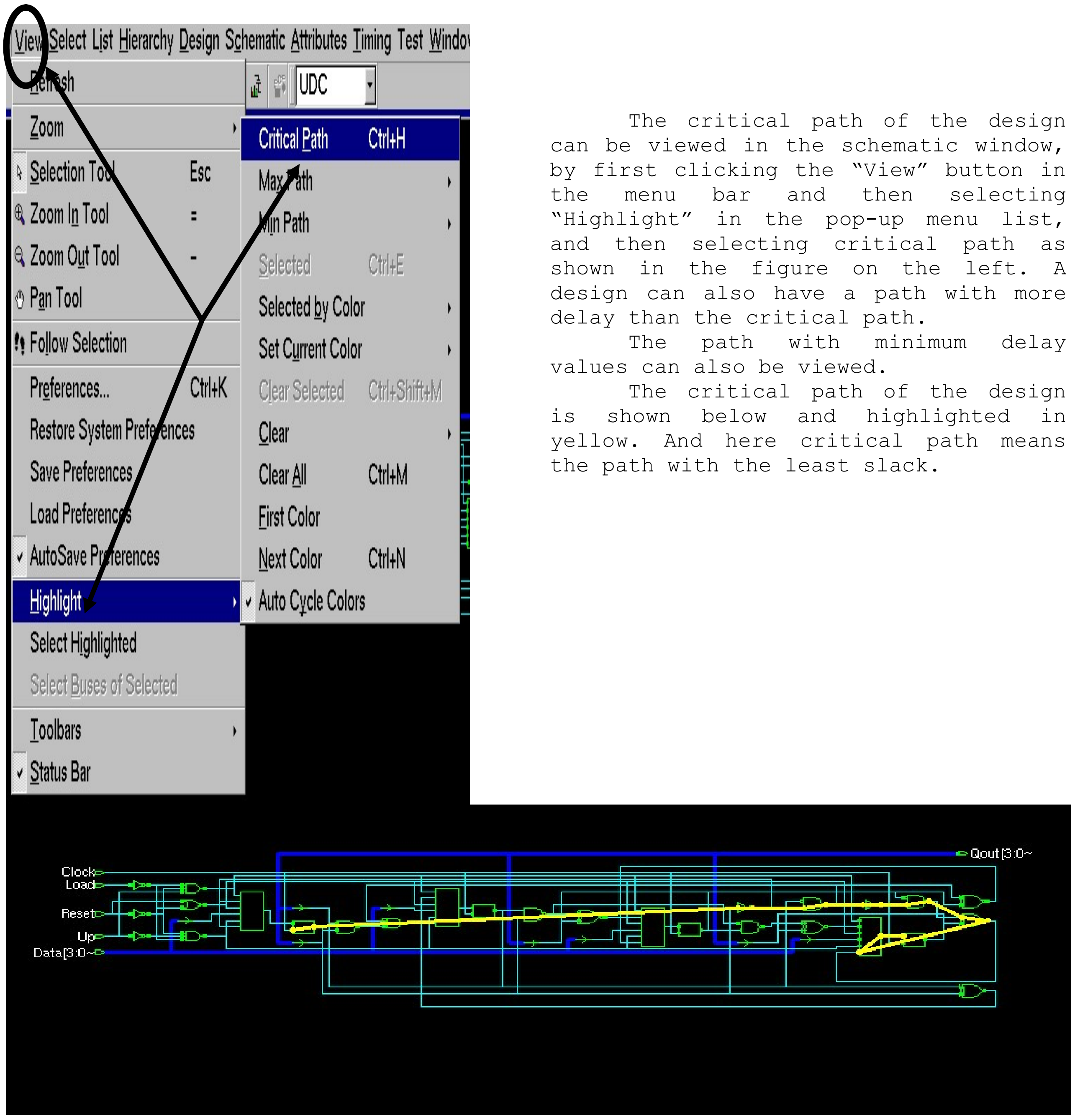
Above given is the timing report of the design, it can be considered to be consisting of 4 parts.

1.) Type of timing information reported: By default report_timing reports the timing of the critical path. It reports only the maximum delay associated with that path. And it also reports only one path per path group.
2.) Operating environment chosen: Here the operating condition and wire load model selected for the timing analysis is reported (here the default values are used), the start point and end point of the timing path is also mentioned.

]

3.) Arrival time : The actual time taken due to the delays of the various components in the specified timing paths as well as delays due to nets are reported here, it can be noticed that the clock latency adds up to the arrival time and is mentioned as "clock network delay" here.
4.) Required time : Here the requirement of the design is reported as specified by the designer. The clock uncertainty (setup) or rather negative skew is subtracted from the clock period, and latency is added to the clock period because latency pushes the capture clock as well.

And then finally the arrival time is subtracted from the required time and slack value is arrived at (which always needs to be positive).



The critical path of the design can be viewed in the schematic window, by first clicking the "View" button in the menu bar and then selecting "Highlight" in the pop-up menu list, and then selecting critical path as shown in the figure on the left. A design can also have a path with more delay than the critical path.

The path with minimum delay values can also be viewed.

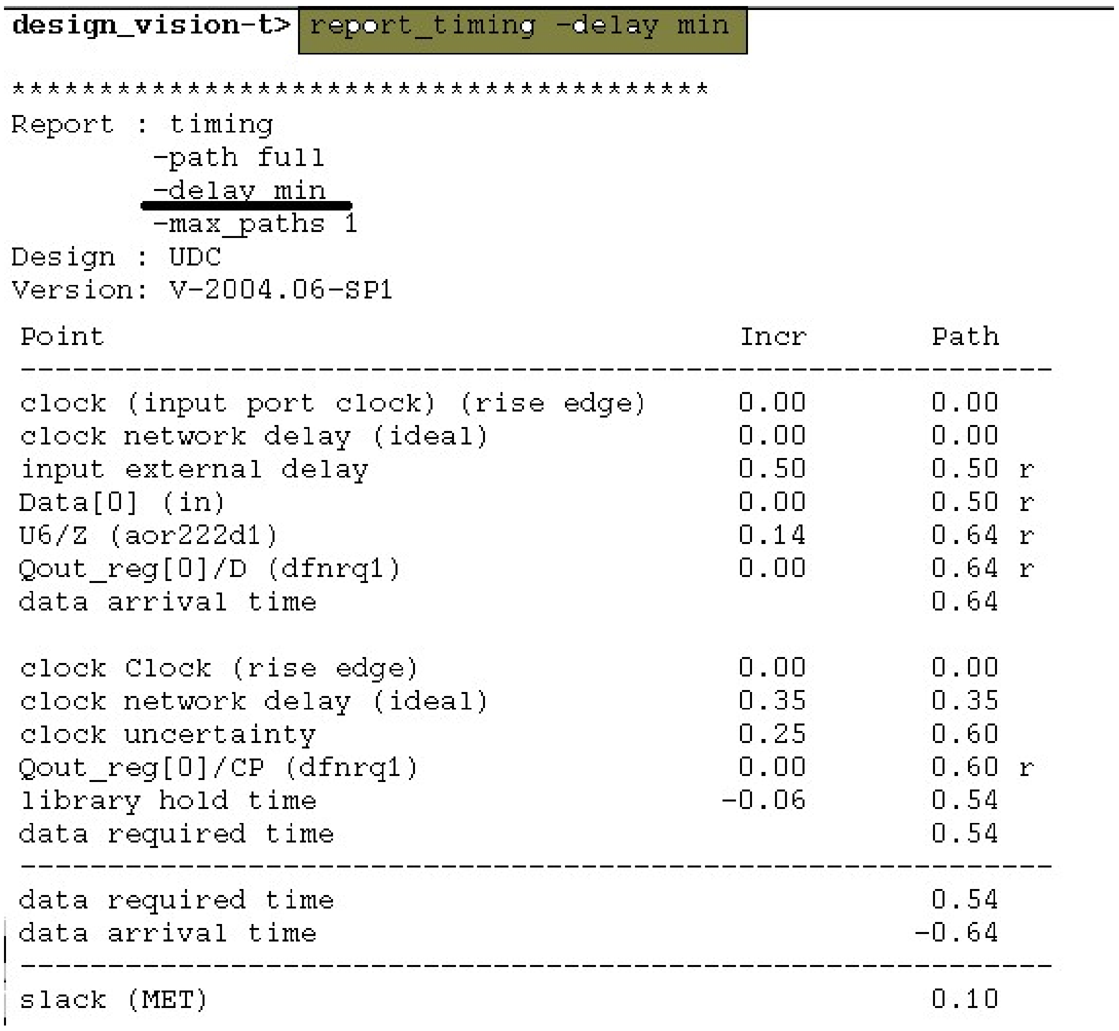
The critical path of the design is shown below and highlighted in yellow. And here critical path means the path with the least slack.

Figure 2.14 : The critical path highlighted

As already discussed the timing analysis is done only with respect to maximum delays and synthesis almost always strives to take care that setup violations do not occur and not much effort is put on hold requirements unless the designer specifically demands so. In order to view the minimum delays and to get a report of the paths with minimum delay as well as to verify whether there is any hold violation the report_timing command is used along with the switch **"-delay"** and the option **"min"**, as shown in Figure 2.15 below.

Here some parts of the report have been edited out and only the relevant information has been included. It can be observed that the report is for "-delay min" (as compared to -delay max in the timing report in figure 2.13). The clock uncertainty (hold) or rather positive skew is added to the required time - since positive skew pushes the clock and data has to be stable for that much more amount of time.

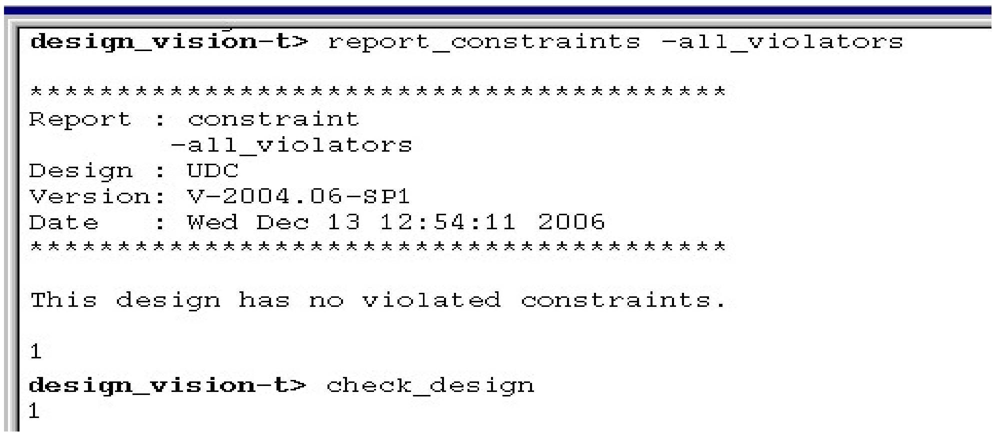
During analysis of minimum delay the slack is calculated by subtracting the required time from the arrival time.



```
design_vision-t> report_timing -delay min

****************************************
Report : timing
        -path full
        -delay min
        -max_paths 1
Design : UDC
Version: V-2004.06-SP1

 Point                                     Incr       Path
 -----------------------------------------------------------
 clock (input port clock) (rise edge)      0.00       0.00
 clock network delay (ideal)               0.00       0.00
 input external delay                      0.50       0.50 r
 Data[0] (in)                              0.00       0.50 r
 U6/Z (aor222d1)                           0.14       0.64 r
 Qout_reg[0]/D (dfnrq1)                    0.00       0.64 r
 data arrival time                                    0.64

 clock Clock (rise edge)                   0.00       0.00
 clock network delay (ideal)               0.35       0.35
 clock uncertainty                         0.25       0.60
 Qout_reg[0]/CP (dfnrq1)                   0.00       0.60 r
 library hold time                        -0.06       0.54
 data required time                                   0.54
 -----------------------------------------------------------
 data required time                                   0.54
 data arrival time                                   -0.64
 -----------------------------------------------------------
 slack (MET)                                          0.10
```

Figure 2.15: Timing report for minimum delays.

After checking the reports it needs to be ensured that the design has not violated any constraints, this is done through the "**report_constraints**" command, as shown in figure 2.16 below. If any of the constraints have been violated, they are reported along with the margins of violations. Also the design needs to be checked for design rule violations, which can be done through "**check_design**".



```
design_vision-t> report_constraints -all_violators

****************************************
Report : constraint
         -all_violators
Design : UDC
Version: V-2004.06-SP1
Date   : Wed Dec 13 12:54:11 2006
****************************************

This design has no violated constraints.

1
design_vision-t> check_design
1
```

Figure 2.16: Checking the design for violations

Once the design is constrained, compiled and the reports generated if the performance of the design is satisfactory it needs to be saved, this again can be done either through the GUI or from the command line.

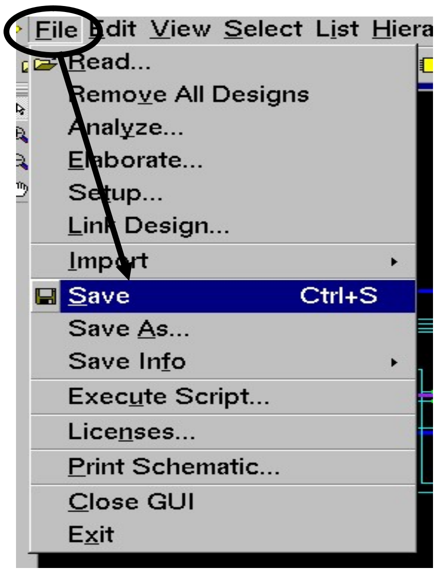


Figure 2.17 : Saving the design

When the design is saved from the GUI the netlist by default takes the name of the module name (or entity name in case of VHDL design) and the netlist is saved in the DB (database) format (specific to Synopsys). If the designer chooses the netlist can be saved with any other name and any of the formats E.g.: .db, .v, .vhd etc.

As shown in figure 2.17 click the "File" option in the menu bar and then click on "Save" to save the netlist in .db format with module name.

Alternatively clicking "Save As…" in the same pop-up menu bar leads to the save-as-design window as shown below (Figure 2.18) where one can choose any name and any of the available formats.

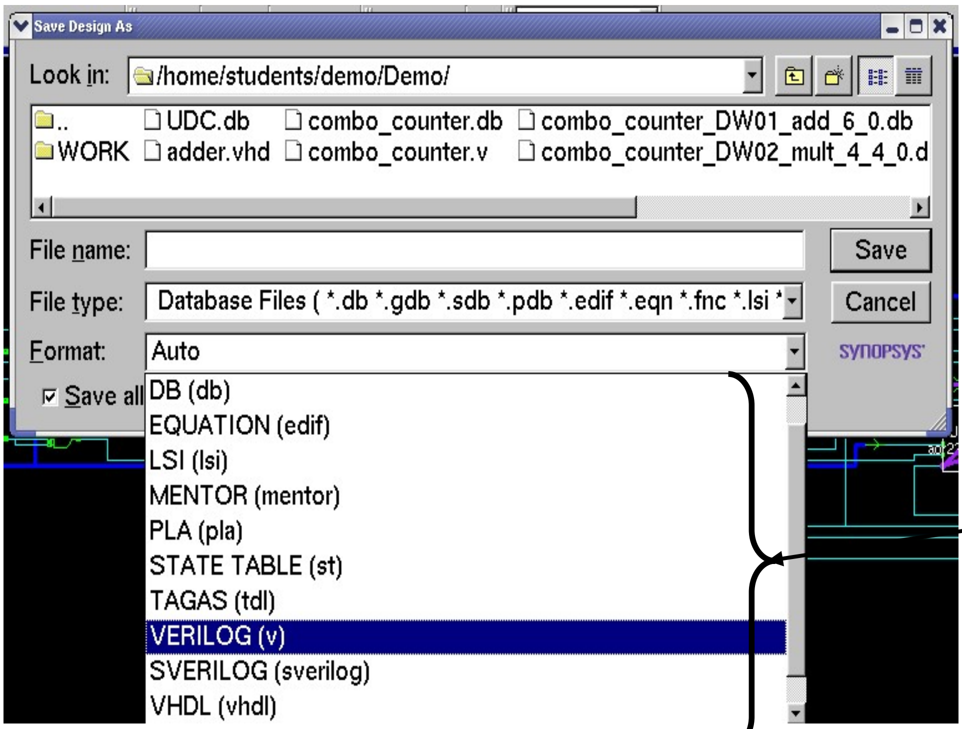


Figure 2.18 : Saving the design with different name and user defined format

The netlist can be saved in any of the formats but also depends on what is next performed on it. If the netlist is to be used for static timing analysis using Primetime from Synopsys then its is preferable to have it in "**.db**" format whereas if it is to be taken to the P&R flow using Astro (Synopsys) then it is preferable to have it in "**.v**" format. The equivalent commands for saving the files are as shown below in figure 2.19.

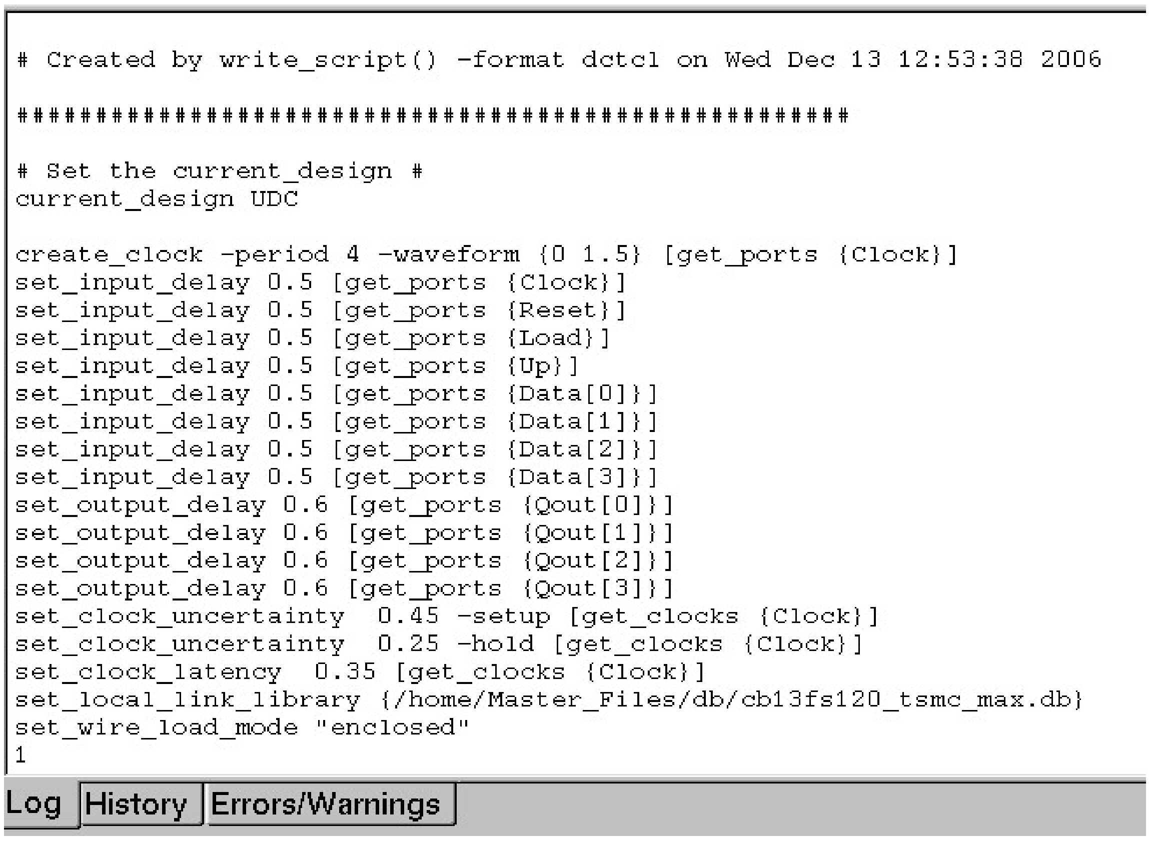
```
design_vision-t> write -hierarchy
Writing to file /home/students/demo/Demo/UDC.db
1
design_vision-t> write -hierarchy -format verilog -output /home/students/demo/Demo/UpDC.v
```

Figure 2.19 : Saving the design from the command line

When the netlist is saved in "**.db**" format the constraints (used while synthesizing the design) are saved explicitly and can then be extracted at a later stage. In case a netlist has been given the constraints used during the synthesis process can be known by first loading the netlist (using the "**read**" command) and then using the "**write_script**" command. This command puts out the constraints onto the console, whereas if the constraints need to be stored in a file the switch "-output" should be used followed by the name of text file.

```
design_vision-t> write_script -output constraints.scr
1
```

The constraints file can be saved in the .txt format, .sdc format or the .scr format. The .scr or .sdc files can be used to again constrain the design by making some changes in the files.



```
# Created by write_script() -format dctcl on Wed Dec 13 12:53:38 2006

#####################################################

# Set the current_design #
current_design UDC

create_clock -period 4 -waveform {0 1.5} [get_ports {Clock}]
set_input_delay 0.5 [get_ports {Clock}]
set_input_delay 0.5 [get_ports {Reset}]
set_input_delay 0.5 [get_ports {Load}]
set_input_delay 0.5 [get_ports {Up}]
set_input_delay 0.5 [get_ports {Data[0]}]
set_input_delay 0.5 [get_ports {Data[1]}]
set_input_delay 0.5 [get_ports {Data[2]}]
set_input_delay 0.5 [get_ports {Data[3]}]
set_output_delay 0.6 [get_ports {Qout[0]}]
set_output_delay 0.6 [get_ports {Qout[1]}]
set_output_delay 0.6 [get_ports {Qout[2]}]
set_output_delay 0.6 [get_ports {Qout[3]}]
set_clock_uncertainty  0.45 -setup [get_clocks {Clock}]
set_clock_uncertainty  0.25 -hold [get_clocks {Clock}]
set_clock_latency  0.35 [get_clocks {Clock}]
set_local_link_library {/home/Master_Files/db/cb13fs120_tsmc_max.db}
set_wire_load_mode "enclosed"
1
```

Figure 2.20: The constraints extracted from the netlist.

Above shown in figure 2.20 is the output of the write_script command when applied without the "-output" switch. All the constraints, which were applied on the design, have been listed out and also the default constraint that is used while synthesizing the design is also listed. Eg: Here wire load mode is selected as enclosed which is by default defined in the technology library and also the library used to synthesize the design is also listed. The same can also be put out to a script file – with extension .scr, a Synopsys Design Constraint file – with extension .sdc or a normal text file with the .txt file format.

Thus far we have discussed some basic constraints and the method to compile the design, various reports that can be generated has also been discussed. Some of the various options available with the tool has been mentioned, this is only a gist of the capability of the Design Compiler tool and gets you ready to have a hands on experience on it. With a little inquisitiveness and some hard work it won't be long before you can master the tool.