COE-561 DIGITAL SYSTEM DESIGN AND SYNTHESIS

(Term 051)

Final Report of a tutorial on Advanced ASIC Chip Synthesis using Synopsys

By

S.M.Rehman

# 230419

Submitted to

Dr.Aimane El-Maleh

ABSTRACT

As the number of logic gates fabricated on to a small chip using deep submicron technologies is exponentially increasing, the conventional schematic capture used with the help of CAD tools started declining giving way to a more sophisticated but exact VHDL coding. A conventional ASIC design flow is reported using Synopsys synthesis tool. Synthesis, as referred to in present day IC design, can be broadly divided into **Logic synthesis** and **High level synthesis.** An overview of relevant Synopsys products such as Library Compiler, Design Compiler and Design Vision, Physical Compiler, PrimeTime, DFT Compiler, and Formality is mentioned in this report. A description of RTL and VHDL gate level simulation is also stressed. This report talks in detail how to use Synopsys products for area, delay as well as power optimization.

1. INTRODUCTION

Since the 1980s, when schematic capture was introduced into the world of VLSI design, it has been a widely used design format. In the concept of schematic capture, logic gates that will be used to design a certain circuit are hand drawn using CAD tool. Upon completion of schematics drawing, a database is stored based on the hand drawn schematics. A common format used for the database interface is Electronic Database Interchange Format (EDIF).

Schematic capture failed in 1990s when the number of logic gates involved in a design increased to hundreds of thousands a smaller ‘window market’ is forcing designers to design a product for a much shorter time frame. This is where HDLs enter the scene.

In VHDL (Very High Speed Integrated Circuit Hardware Description Language) design, a designer will code the design in terms of VHDL code as opposed to conventional method of schematic capture.This code can then be synthesized using VHDL synthesis tools. The synthesized ciruit can then be stored in a netlist database. Among the tools commonly used for synthesis are Synopsys’s Design Compiler, Mentor Graphics’s Autologix, Exemplar, Synplicity’s Synplify, Cadence’s Ambit and many others.

In general, a VHDL design can be categorized into 3 different groups. Each has its own distinct characteristics and style of coding:

1) Stuctural VHDL
2) Behavioral VHDL , and
3) Synthesizable VHDL.

Structural VHDL is a data type structure that is best described as a netlist of a design or schematic. It has declarations of all the types of components used in the design and interconnects to connect all the different components.

Behavioral VHDL structure describes the design in a behavioral manner, mimicking its performance and functionality. A design coded behaviorally is just a black box. The code is written in such a way as to generate the specified output signals for a given set of input signals. This form of coding is nonsynthesizable and is normally used only for system testing.

Synthesizable (RTL) coding is the most complicated form of coding as it describes a design in a high level manner through a subset of VHDL syntax. This form of coding is somewhere between structural and behavioral code. It is at a higher level of description compared to structural VHDL but at a lower level of description compared to that of behavioral VHDL. There are many different styles to write RTL code.

In VHDL synthesis, the timing and functionality of a design must always be considered together. In synthesis, VHDL code is mapped into hardware logic gates for a specific technology library. During this phase of synthesis, the designer will also input design contraints into the synthesis tool. This would allow the tool to map more efficiently to logic gates. If the synthesized result meets all timing criteria, the designer can move forward to layout. However if timing is not met, the designer will have to analyse the design to fix the timing violations.

2. ASIC DESIGN FLOW USING SYNTHESIS

The synthesis based ASIC design flow consists of the following steps:

1) Functional specification of the design.
2) HDL coding in VHDL/Verilog RTL
3) RTL/ behavioral or functional simulation of the HDL
4) Logic synthesis
5) Test insertion and ATPG
6) Post synthesis or gate level simulation
7) Floorplanning / place and route.

After the design has been captured in HDL, it is essential to verify that the code matches the required functionality, prior to synthesis. This step is called as pre-synthesis behavioral simulation of the HDL. This can be performed by simply viewing the waveforms in a graphical simulation tool. An alternative way is to write a testbench.

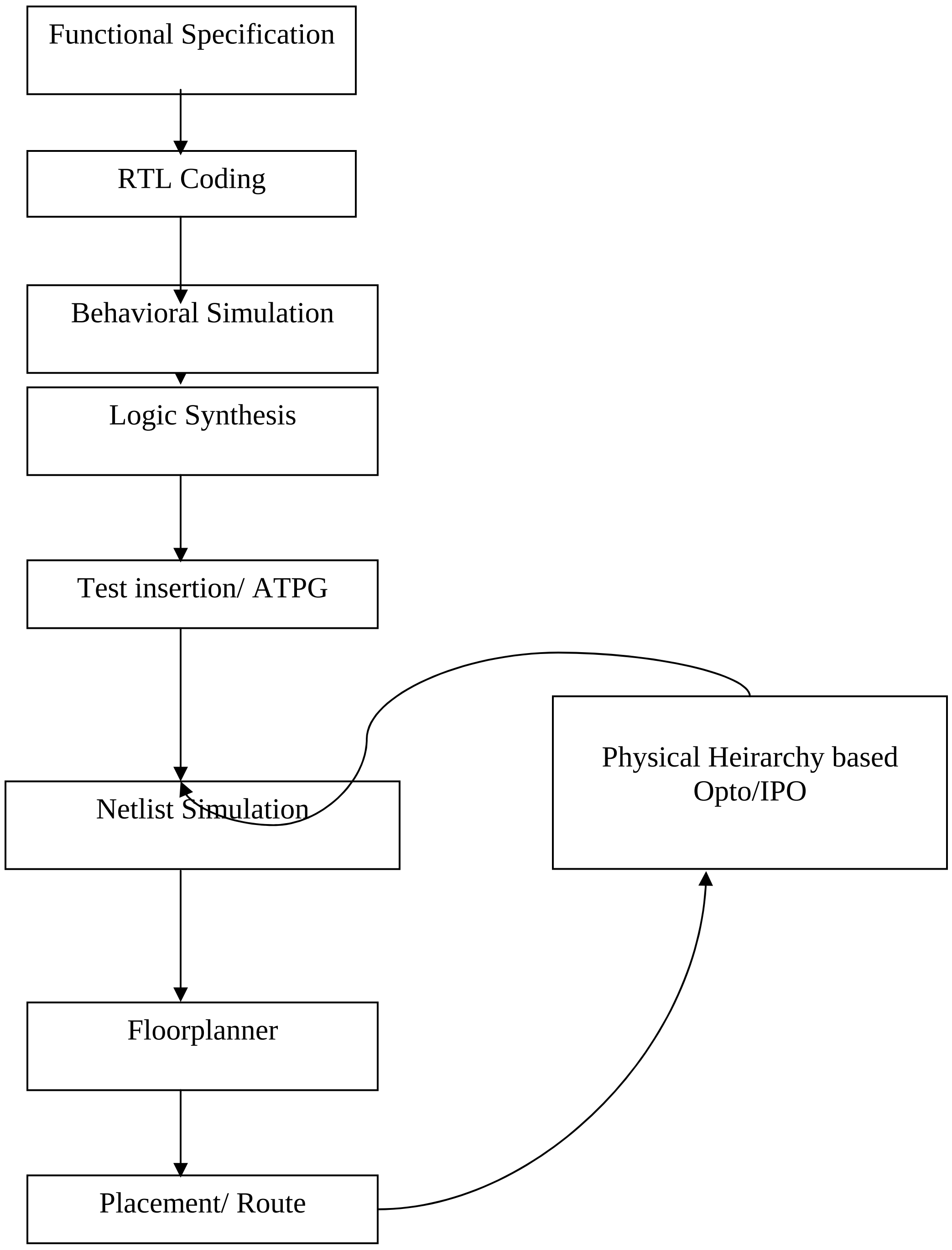
To simulate a synthesized gate level netlist, VHDL simulation models of the technology library cells are required. These can be of 4 types:

1) Unit Delay Structural Model (UDSM)—Combination cells have a rise/fall delay of 1ns, while all sequential cells have a rise/fall delay of 2 ns.

2) Full Timing Structural Model (FTSM)—includes transport wire delays and pin to pin delays on a zero delay functional network.

3) Full Timing Behavioral Model (FTBM) – includes detailed timing verification.

4) Full Timing Optimized Gate level Simulation (FTGS) – used for fast sign off quality timing verification.


Functional Specification
RTL Coding
Behavioral Simulation
Logic Synthesis
Test insertion/ ATPG
Netlist Simulation
Physical Heirarchy based Opto/IPO
Floorplanner
Placement/ Route

Synthesis, as referred to in present day IC design, can be broadly divided into **Logic synthesis** and **High level synthesis.** High level synthesis involves synthesis of logic from behavioral descriptions. Logic synthesis on other hand synthesizes logic from Register Transfer Logic (RTL) descriptions. The logic synthesis process consists of 2 steps— **translation and optimization**. Translation involves transforming a HDL (RTL) description to gates, while optimization involves selecting the optimal combination of ASIC technology library cells to achieve the required functionality.

Logic synthesis provides the best results when the critical path lies in one heirarchical block as opposed to traversing multiple heirarchical blocks. Logic synthesis optimizes area and timing whereas behavioral synthesis adds one more dimension to optimization, namely latency.In general behavioral synthesis is suited to designs with complex data flow or I/O operations, and several memory accesses. It is meant for fully synchronous designs and implies faster simulation. A tight integration is required between Behavioral Synthesis and Logic synthesis tools, since the output of behavioral synthesis is an RTL description which is then synthesized to gates using logic synthesis.

Designs for testability (DFT) techniques in ASIC design have been used in recent years to reduce the cost of testing by defining testability criteria early in the design cycle. The most popular DFT technique is the Scan Design Technique.

With the advent of deep submicron technologies, interconnect or net delays have become a significant component of the overall delays. In other words, while gate delays have decreased, the wire delays have increased due to effects such as lateral capacitance, fringe capacitance and overalap capacitance.

3. SYNTHESIS USING SYNOPSYS TUTORIAL

3.1. SYNOPSYS PRODUCTS

The following are relevant Synopsys products:

1) Library Compiler
2) Design Compiler and Design Vision
3) Physical Compiler
4) PrimeTime
5) DFT Compiler
6) Formality

3.1.1. LIBRARY COMPILER

The core of any ASIC design is the technology library containing a set of logic cells. The library may contain functional description, timing, area and other pertinent information of each cell. Library Compiler (LC) parses this textual information for completeness and correctness, before converting it into a format used globally by all Synopsys applications. LC is invoked by typing lc_shell in UNIX shell.

3.1.2. DESIGN COMPILER (DC) AND DESIGN VISION (DV)

DC and DV comprise a powerful suite of logic synthesis products, designed to provide an optimal level of gate-level synthesized netlist based on the design specifications, and timing constraints. In addition to high level synthesis capabilities, it also incorporates a static timing analysis engine, along with solutions for FPGA synthesis and links to layout (LTL).

DC is a command line interface of Synopsis synthesis tool and is invoked by either typing dc_shell or dc_shell-t in a Unix shell.

DV is the graphical front-end version of DC and is launched by typing design_vision. DV also supports schematic generation, with critical path analysis through point to point highlighting.

3.1.3. PHYSICAL COMPILER (PHYC)

PhyC is a new tool by symopsys that is a superset of DC IN addition to incorporating all the synthesis and optimization capabilities of DC, it also provides the ability to concurrently place cells optimally, based on the timing and/or area constraints of the design.
PhyC is invoked by typing psyn_shell. A separate GUI version is also available, which is launched by typing psyn_gui. Although slow by comparison, psyn_gui provides the users the ability to traverse between the logical and schematic view of the design.
PhyC is the superset of DC.

3.1.4. PRIMETIME (PT)

PT is the synopsis time-off quality, full chip, gate level static timing analysis tool. In addition it also allows for comprehensive modelling capabilities, often required by large designs. PT is faster compared to DCs internal static timing analysis. It also provides enhanced analysis capabilities, both textually and graphically.
PT is a stand-alone device and can be invoked as a command line interface (pt_shell) or graphically (primetime).

3.1.5. DFT COMPILER (DFTC)

The DFTC is the Synopsys test insertion tool that is incorporated within the DC suit of tools. The DFTC is used to insert DFT features like scan insertion and boundary scan, to the design. All DFTC commands are directly invoked from dc_shell or psyn_shell.

3.1.6. FORMALITY

Formality is the Synopsys formal verification or more precisely a logic equivalence checking tool. The tool features enhanced graphical debugging capabilities that include schematic representation of logic under verification, and visual suggestion annotated to the schematic as pointers of possible incorrect logic. It also provides suggestions for possible fixes to the design.

3.2. TOOL INTRODUCTION

The most commonly used synthesis tool in the ASIC industry is Synopsy's Design Compiler.Synopsy's synthesis tool is divided into 2 sections: Design Analyzer (DA) and Design compiler (DC). DA is the graphical front end of the Synopsys's synthesis tool. DC or the dc_shell is the command line interface for the same synthesis tool.
DA allows the following actions:

1) Set system variable values such as the technology library name.
2) Read and write designs in multiple formats such as EDIF, netlist, PLA, Verilog, VHDL, and Equation.
3) Set constraints and attributes graphically on designs, cells, pins, nets, buses, and clocks.
4) Work with heirarchical designs.
5) Synthesize digital circuits.

3.2.1. DESIGN ANALYZER WINDOW

The DA window has a frame (border) around it provided by the Motif Wireless Manager (MWM) for the UNIX system. These borders control the size, shape, and location of an MWM window.

To use DA,

1) The X windows system as well as the Window manager is started on the workstation.
2) DA is started.

When using Synopsys's synthesis tool, a startup file must be present in the currnet working directory from which the synthesis tool has been invoked. This startup file is **.synopsys_dc.setup** file.For PhyC, DC and PT, the default start up file are automatically loaded upon invocation of these tools. These default files do not contain the design dependent data. Their function is to load the Synopsys's technology independent libraries and other parameters. In addition to this system wide file, the user can have a local **.synopsys_dc.setup** file. In the local startup files, the following minimum information must be set before any synthesis is performed:

a) **search_path**: This parameter is used to specify to the synthesis tool all the paths that are to be searched when looking for a synthesis technology library to reference during synthesis.
b) **target_library**: The file pointed to by this parameter is the library that contains all the logic cells for mapping during synthesis.
c) **symbol_library**: This parameter points to the library that contains "visual" information on the logic cells in the synthesis technology library.. All logic cells have symbolic representation and information about the symbols stored in this library.
d) **link_library**: This parameter points to the library that contains information on the logic gates in the synthesis technology library.

3.2.2. DESIGN OBJECTS

There are 8 different types of design objects categorized by DC. These are:

1) Design
2) Cell
3) Reference
4) Port
5) Pin
6) Net
7) Clock
8) Library.

Finding Design Objects

One of the most useful commands provided by DC & PT is the **get_*** commands. A full list of get commands can be found by typing “help get_*” in the dc_shell command line.

3.2.3. VARIABLES

All variables are global and last only during the session. They are not saved along with the design database. A list of all DC variables can be obtained by

```
dc_shell –t > printvar *
```

For a particular type of variable, say test related,

```
dc_shell –t > printvar *test*
```

3.2.4. ATTRIBUTES

Attributes are similar in nature to variables. Both store information. However, attributes store information on a particular design object such as nets, cells or clocks.

3.2.5. SYNOPSYS FORMATS

Most Synopsys products support and share, a common internal structure, called the "db" format. The db files are the binary compiled forms representing the text data, be it the RTL code, the mapped gate level designs or the Synopsys library itself.

A common practice of organizing files is the following:

Script files:	<filename> .scr
RTL Verilog files:	<filename> .v
Synthesized Verilog netlist	<filename> .sv
RTL VHDL file:	<filename> .vhd
Synthesized VHDL netlist	<filename> .svhd
EDIF file:	<filename> .edf
Synopsys database file:	<filename> .db
Reports:	<filename> .rpt
Log files:	<filename> .log

Design Analyzer uses 4 different views of designs. The first view shows all design in memory. The other 3 views represent different aspects of a design.

3.3. DESIGNS VIEW

Shows all designs and subdesigns in memory. A certain type of icon represents each design. Designs view is the initial view. From the designs view, a design is selected for exploration or for setting design attributes and constraints.

3.3.1. HEIRARCHY VIEW

Shows a design as a set of one or more named subdesigns.

3.3.2. SYMBOL VIEW

Shows a design as a black box with input or output ports. From the symbol view, attributes and constraints can be set for a design and its ports.

3.3.3. SCHEMATIC VIEW

Shows a design as a schematic composed of instances, nets, and ports. An instance in a schematic can be a subdesign.

3.4. DESCRIBING THE MENU BAR

The DA provides a help menu at the far right of the menu bar. The others are:

1) Setup menu
2) File menu
3) Edit menu
4) View menu
5) Attributes menu
6) Analysis menu
7) Tools menu.

3.4.1. SETUP MENU

The setup menu can be used to obtain information about

a) Defaults
b) Variables
c) Licenses
d) Execute script
e) Scripts
f) Command window

3.4.2. FILE MENU

The file menu can be used to get information about

a) Read
b) Analyze
c) Elaborate
d) Import
e) Save
f) Save As
g) Save Info
h) Plot
i) Quit.

3.4.3. EDIT MENU

The edit menu can be used to get information about

a) Delete
b) Insert pads
c) Select
d) Unselect All
e) Group
f) Ungroup
g) Uniquify
h) Reset

3.4.4. VIEW MENU

The view menu can be used to get information about

a) Full view
b) Zoom in
c) Zoom out
d) Change sheet
e) Change view
f) Change level
g) Push to reference
h) New View
i) Style
j) Refresh
k) Recreate.

3.4.5. ATTRIBUTES MENU

The attributes menu can be used to get information about the following four sections:

a) Clocks
b) Operating Environment
c) Optimization Constraints
d) Optimization directives.

3.4.6. ANALYSIS MENU

The analysis menu can be used to get information about

a) Link Design
b) Check Design
c) Time Design
d) Show Timing
e) Show Net load
f) Highlight
g) Test report
h) Report

3.4.7. TOOLS MENU

The tools menu can be used to get information about

a) Design optimization
b) Finite State Machines (FSM)
c) FPGA compiler
d) Test Synthesis.

4. PRE AND POST-SYNTHESIS SIMULATION

Simulation is the process of verifying the functionality and timing of a design against its original specifications. In the ASIC design flow, designers perform functional simulation prior to synthesis. After synthesis, gate level simulation is performed on the netlist generated by synthesis.

4.1. RTL SIMULATION

Before Synthesis, the design must be entered in DC in the RTL format (although other formats also exist). The following two methods of design entry can be done:

1) “read” command
2) “analyze/elaborate” command.

VHDL RTL simulation is used to verify that the design coded in VHDL captures the fuctionality required by the design specifications. After the design has been coded in

VHDL, a testebch which applies stimulus to the design must be created. The design to be simulated is refrerred to as the Design Under Test (DUT). The outputs generated by the DUT can then be compared in the testbench or verified interactively during debugging.

4.2. VHDL GATE LEVEL SIMULATION

To perform gate level simulation of a VHDL netlist, one requires the VHDL simulation libraries from the ASIC vendor. The Synopsis ‘liban’ utility can generate the VHDL library models from the synthesis technology library. For the more complex cells, simulation models have to be manully created. The VHDl models generated are encrypted so that the vendor proprietary information is protected.

4.3. TECHNOLOGY LIBRARY

The Synopsys technology libraries can be separated into 2 broad classes:

1) Logic Library
2) Physical Library.

4.3.1. LOGIC LIBRARY

The logic library contains information relevant only to the synthesis process and is used by the DC for the synthesis and optimization of the design. This information may include pin to pin timing, area, pin types and power along with other necessary data needed by DC. The logic library is a text file (“.lib”) which is compiled using the Library Compiler (LC) to generate a binary format with “.db” extension.

4.3.2. PHYSICAL LIBRARY

The physical library contains the physical characteristics of the cell along with other necessary information relevant to Physical Compiler.Such information may contain data relating to the physical dimensions of cell, a corresponding phyical cell should also be present. The physical library is also a text file (“.plib”) and is compiled By LC to generate a binay format with a “.pdb” extension

5. PHYSICAL SYNTHESIS

Time to market is rapidly shrinking while design complexities are increasing. The problem is further aggravated by shrinking geometries, forcing ASIC designers to think about power and cross talk along with timing, much earlier in the design cycle. The exchange of data between layout tools and DA is certianly not efficient. Time wasted during synthesis layout iterations is still a major bottleneck.

The main cause of synthesis layout iterations can be attributed to the traditional synthesis approach of relying on wire-load models to synthesize the design. The wire-load models are just estimates of the final routed design. They may differ considerably from the real extracted delays of the layout surface. Going back and forth from layout to synthesis solves this problem however at the expense of time.

Inorder to alleviate this problem, Synopsys introduced a novel approach of synthesizing the design without the need for wire-load models. This new tool is called Physical Compiler (or PhyC) and it performs synthesis along with concurrent placement, based on the floorplan information. Combining synthesis and placement provides an accurate modelling of actual interconnect delays during synthesis. In addition this tool also minimzes the previous headache of passing data back and forth from the layout tool to the synthesis tool.

6. GUIDELINES FOR LOGIC SYNTHESIS

The following guidelines are not ‘hard and fast’ rules for effective synthesis but are applicable to most cases and exceptions to these guidelines are possible.

1) For better results from the synthesis, accurate point to point delays for asynchronous paths are to be specified.
2) By registering outputs of the different design modules, the designer saves from having to perform painstaking time budgeting.
3) Positive and Negative edge flipflops have to be separated into separate heirarchical blocks. This makes the debug process and timing analysis during synthesis much simpler.
4) FSMs should be grouped & optimized separately.
5) The recommended size of the module for synthesis is in the range of 250-5000.
6) Too many heirarchical blocks should be avoided. On the other hand having a large flat design with no heirarchy is not the solution. One has to develop a feel for the ‘middle of the road’ strategy.
7) Logic in the critical path has to be captured into a seaparate level of heirarchy.
8) Last but not the least, it is always advisable to perform a preliminary synthesis and place and route so as to identify any serious issues which may require rewriting the HDL code.

7. DESIGN VALIDATION AND OPTIMIZATION

This section gives an introduction to the methodology of optimizing an ASIC for parameters like Area, Power and Delay. Various optimizing schemes like derving constraints on area and power, setting limits on timing requirement, Flattening logic, Boolean optimization etc have been introduced here. The process mentioned below covers validation, optimization using synopsys tool at gate level.

Before we start the synthesis the setup files have to be created to store the RTL files

mkdir synthesis
cd synthesis

Type in **synopsys_tools**

The following setup files are to be copied to the login.

cp /usr/cad/course/devine/.synopsys_dc.setup .

cp /usr/cad/course/devine/.synopsys_vss.setup .

When analyzing the RTL files, a **WORK** directory is needed, so a directory WORK is created.

mkdir WORK

Design_analyzer &

Let’s first copy a VHDL or a VERILOG file in to the directory we are working on. We then analyze the file by clicking on File option in the menu and choosing Analyze.

File -----> Analyze

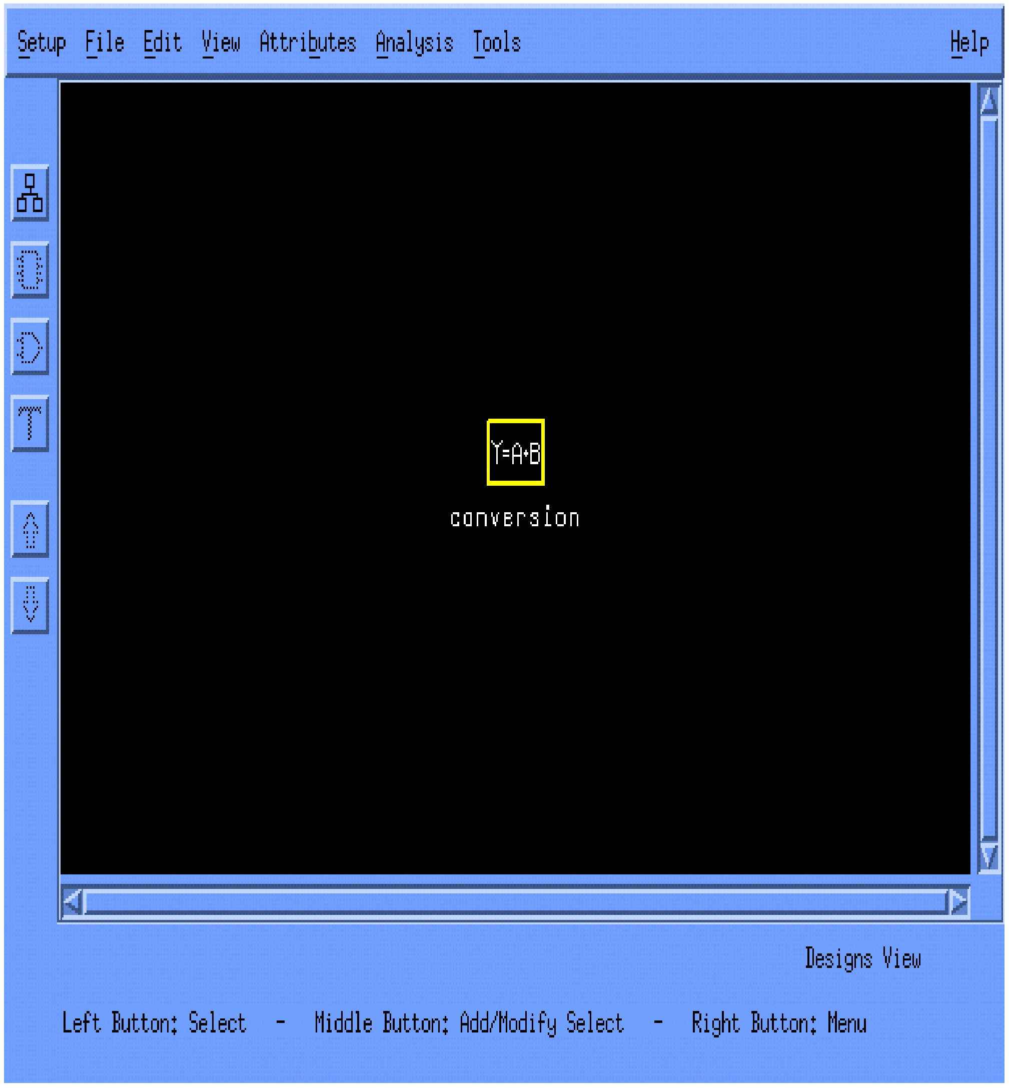
This command analyzes the HDL file and stores the intermediate format for the HDL description in a specified library. In this case the files would be stored in WORK directory. Then elaborate the design by choosing

File ----->Elaborate

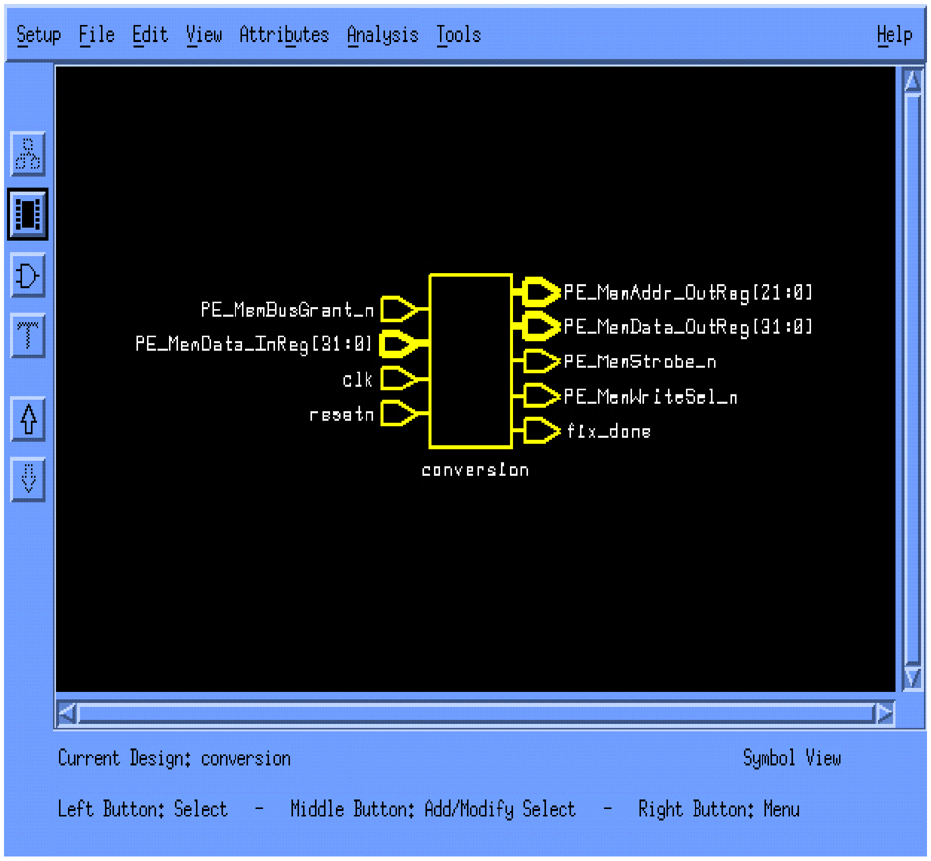
This command builds a design from the intermediate format of VHDL module. Sometimes we may have saved the design as a Synopsys file with an extension **.db** In such cases we may just

File ----> Read

This command reads in all the primitives associated with that particular file.So we should have a schematic like this.


Setup File Edit View Attributes Analysis Tools
Help
Y=A+B
conversion
Designs View
Left Button: Select - Middle Button: Add/Modify Select - Right Button: Menu

This level of the design can be expanded by clicking on the box which says "Y=A+B”. The expanded design level would be as shown below


Setup File Edit View Attributes Analysis Tools Help
PE_MemBusGrant_n
PE_MemData_InReg[31:0]
clk
resetn
PE_MemAddr_OutReg[21:0]
PE_MemData_OutReg[31:0]
PE_MemStrobe_n
PE_MemWriteSel_n
fix_done
conversion
Current Design: conversion
Symbol View
Left Button: Select - Middle Button: Add/Modify Select - Right Button: Menu

By clicking on the schematic view, we can find the gate level configuration of this RTL.

To start the synthesis on this RTL, we need to set up the clock. Get back to the symbol view of the design and click on the input pin **clk** and choose from the menu

Attributes --> Clocks --> specify

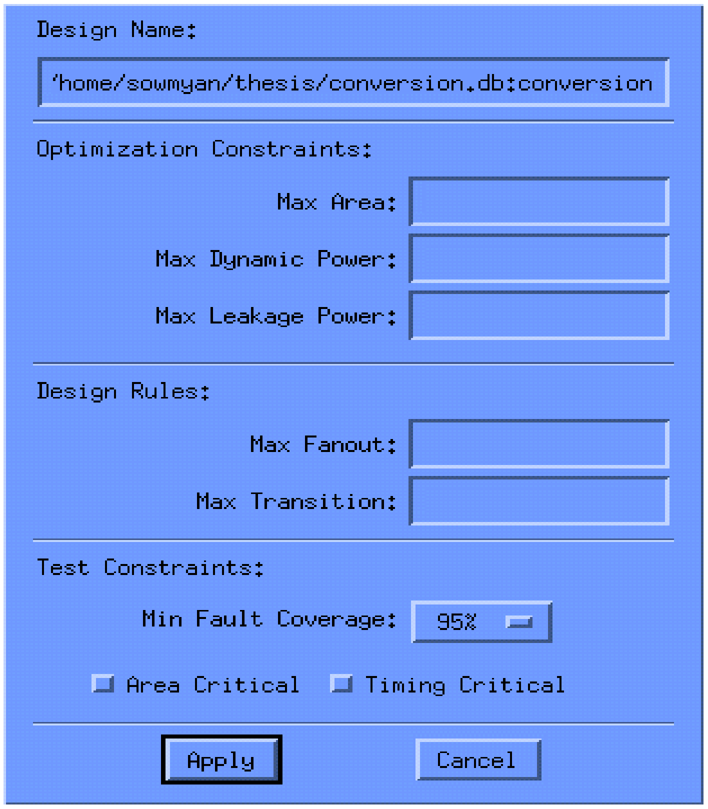
Enter the clock period to be 20000. The units are pico seconds, which means that we have set the frequency to be 50 MHz. Now also enable the option **Dont Touch Network.** This enables the clock is not modified during the synthesis. Also if there is a module which does not have an output, it is better to use the command Dont touch network, to save it during synthesis. Since during the synthesis, the tool may remove it from the netlist. Then to specify the skew click on skew and set it to 1000. This command sets the clock skew values for all flip flops and latches in the transitive fanout of the specified clocks. Select the propagated option. Design Compiler will use faster gates along a path if it is necessary to meet a particular maximum delay requirement, which is affected by the skew and the setup time of the flip-flop.

The next phase is setting up the design constraints for the synthesis.

7.1. AREA OPTIMIZATION

Let’s start with optimizing area first. Choose

Attributes --->Optimization Constraints --->Design constraints



Design Name: ‘home/sowmyan/thesis/conversion.db:conversion

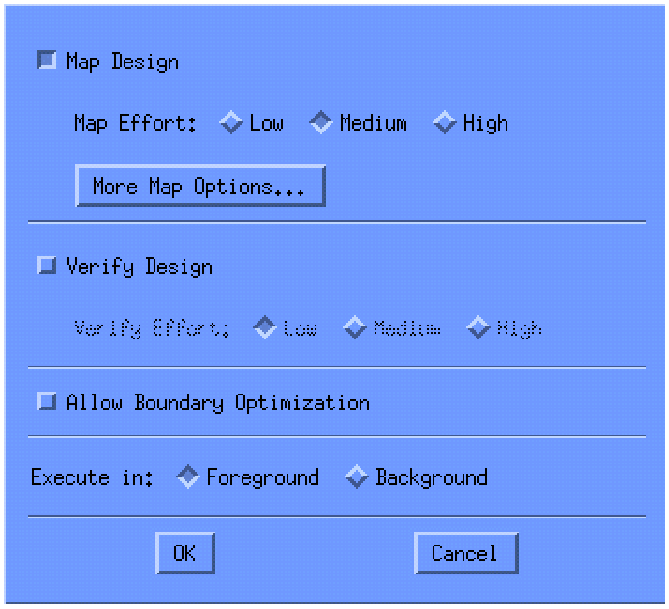
Min Fault Coverage: 95%

Values are not specified for Maximum area, Maximum power.Let optimization be done without being Area and timing critical. When these options are used, the compiler would use the default (minimum) values in the technology library and the synthesis would be done for minimum values of area, power and delay.

The **Max Area** option sets the maximum area attribute for the design. This attribute represents the target area of the design and is used by the compiler to calculate the area cost of the design. The option **Area critical** when enabled tells the compiler that area must be given precedence when inserting scan cells into the design. Scan cells are introduced into the design to improve the testability of the design. In this section, we will not concentrate much on inserting scan cells.

Now that we have the constraint set up, we can go ahead and synthesize the design. So choose from the menu

Tools ---> Design Optimization


Map Design
Map Effort: Low Medium High
More Map Options...
Verify Design
Verify Effort: Low Medium High
Allow Boundary Optimization
Execute in: Foreground Background
OK
Cancel

In this window choose **Verify Design** and Medium level of Map Effort. The synthesis without any design constraints would take couple of minutes. Once the synthesis is done, we need to generate a report on area, power and delay. Click on

Analysis ----> Report

Then enable Area, Power and Timing options. This should bring out a report which gives the area of the design, power consumed, required time, arrival time and slack. The result of this synthesis would be a chip that has smaller area, lesser power and comparitively fast.

When we perform the synthesis on some RTLs, we may notice in the command window **(Setup --> Command Window)** messages like "cell name # does not have active output activity". In such a case, a note is made of such cells, then up in the schematic view we click on them and mark them as " Dont touch network". Marking them as Dont touch network, can be done by choosing

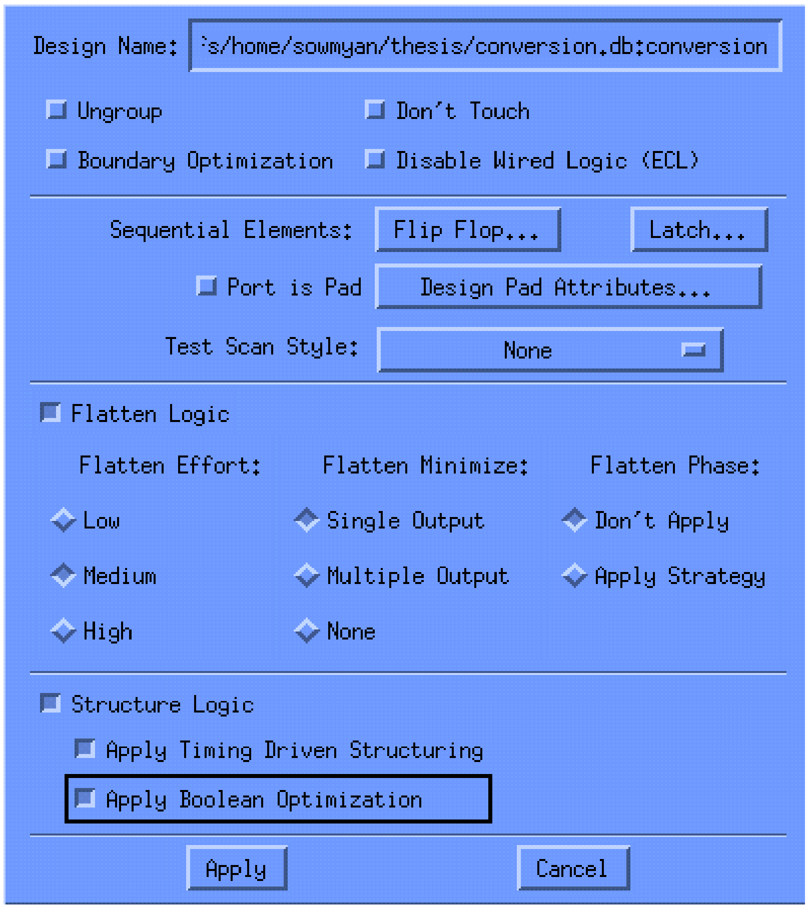
Attributes ---> Optimization Directives ---> Cell

This is done so that the cell is not removed from the netlist during synthesis.

To optimize the design for maximum area, we enter a value larger than the value got from library default synthesis. This would give the compiler some freedom during optimization and we can see in the report a reduction in power with an increase in area. To optimize for minimum area we then choose a value for area that is far less than what was achieved with library defaults, I used "0" for minimum area optimization. The result would be a reduction in area, with an increase in one of the other two design variables.

Options like flattening can also be used to perform area and power optimization. Click on

Attributes ----> Optimization Directives ----> Design


Design Name: `s/home/sowmyan/thesis/conversion.db:conversion

☐ Ungroup ☐ Don't Touch
☐ Boundary Optimization ☐ Disable Wired Logic (ECL)

Sequential Elements: Flip Flop... Latch...
☐ Port is Pad Design Pad Attributes...
Test Scan Style: None

☐ Flatten Logic

Flatten Effort:	Flatten Minimize:	Flatten Phase:
◇ Low	◇ Single Output	◇ Don't Apply
◇ Medium	◇ Multiple Output	◇ Apply Strategy
◇ High	◇ None	

☐ Structure Logic
☐ Apply Timing Driven Structuring
☐ Apply Boolean Optimization

Enable the option "Flatten logic" and "Boolean optimization" and perform the synthesis. Flattening merges modules present in a design into one. So one would expect a reduced area as the result of this synthesis, well that's what would be the answer in most of the cases. But in some cases the result of the synthesis results in an increase in area. It is because this optimization constraint reduces the logical network to a two level sum of products representation and sometimes this may not essentially result in the most efficient design layout.

If we are interested in inserting scan circuits into the design, then "area critical" and "timing critical" options can be used to see the change in area and timing, with/without using these options.

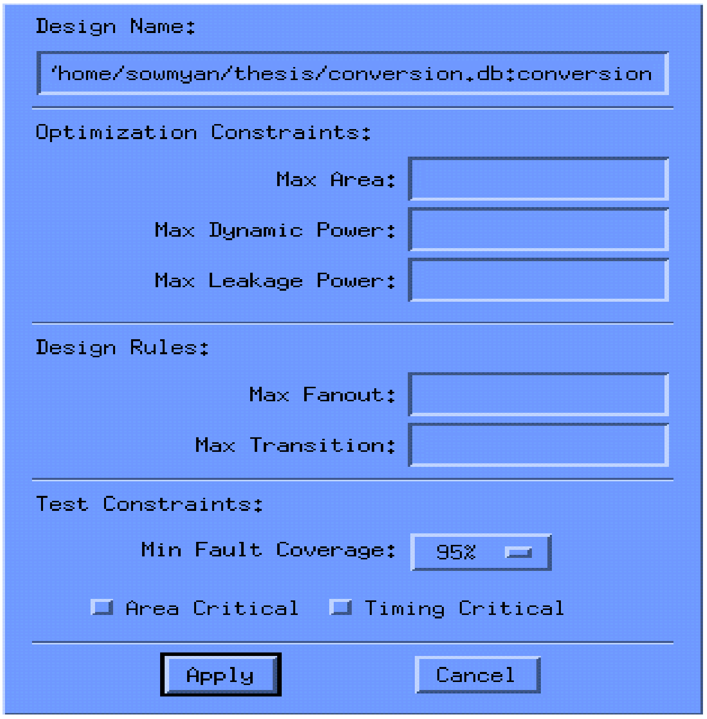
7.2. POWER OPTIMIZATION

For power optimization we choose

Attributes ---> Optimization Constraints ---> Design Constraints

and perform the steps similar to area optimization by entering values for max power. This sets the target value for the tool during power optimization. An important issue with power optimization is that the technology library file used decides on the effectiveness of the synthesis. This is because some tech libraries may not have an equivalent low power module for the module being synthesized, and this may result in the module being removed from the netlist. HP26G is one such library. Hence if we use this library, we may find that the area and power are substantially reduced. The reason can be attributed to the tech library.

The design constraints used in the case of area optimization can also be combined with the constraints used for power optimization to acheive area and power optimization.


Design Name:
/home/sowmyan/thesis/conversion.db:conversion
Optimization Constraints:
Max Area:
Max Dynamic Power:
Max Leakage Power:
Design Rules:
Max Fanout:
Max Transition:
Test Constraints:
Min Fault Coverage: 95%
Area Critical
Timing Critical

Now we move to the tougher part of the synthesis and optimization, the Delay Optimization

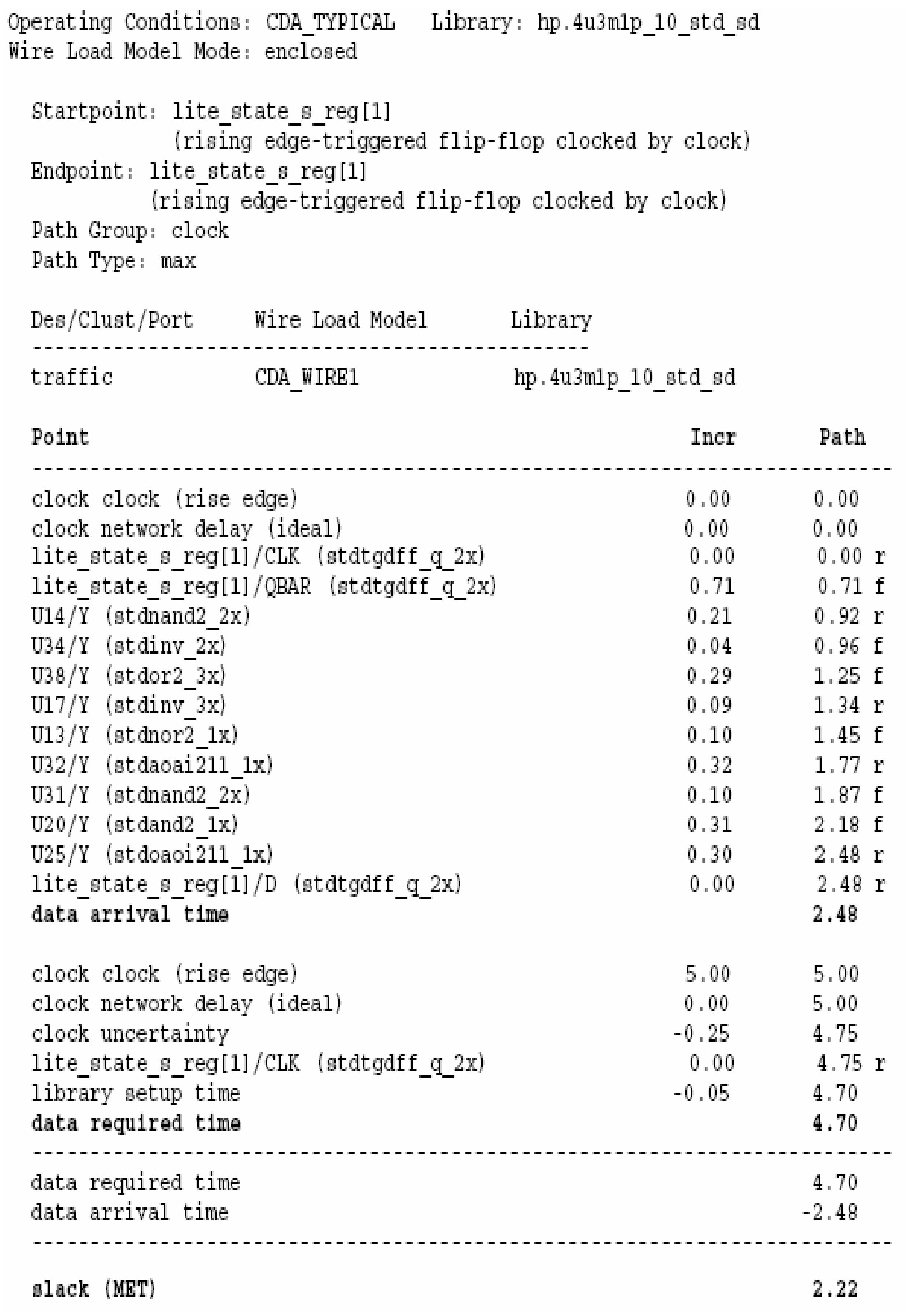
7.3. DELAY OPTIMIZATION

Let's not set any constraints for area and power optimization and just synthesize for the default area and power. In the report that is generated at the end of the synthesis, go to the section which shows the timing results.

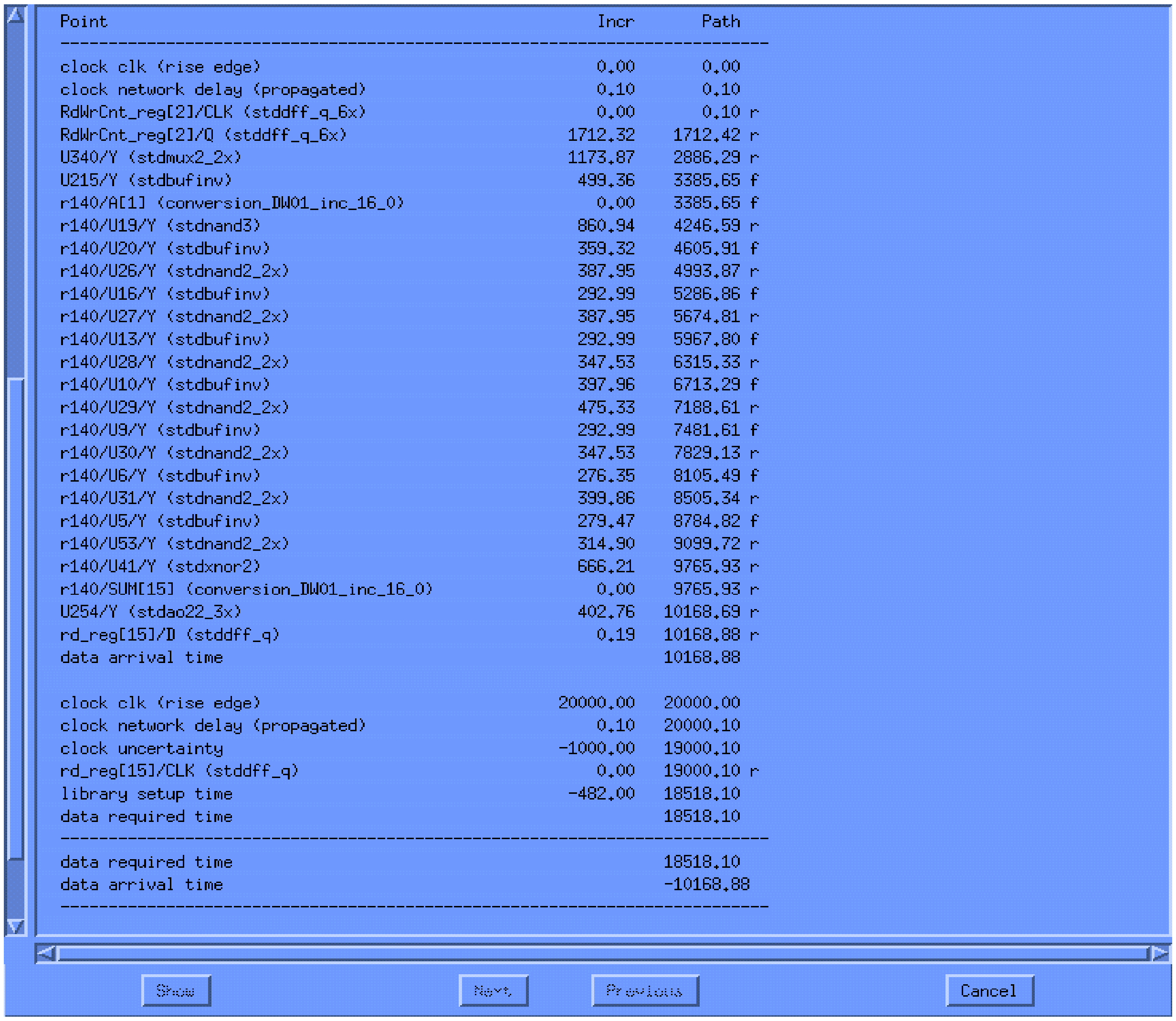
7.3.1. READING THE TIMING REPORTS

The *Timing* and *Point Timing* reports show a path and give timing values in that path. It is critical that we understand how to read what is shown. A sample path is given below:

On the upper left hand side of the timing table is the name *Point*. Below this are listed components in the circuit which a signal travels through. To the right of this is the name *Incr*. Below this are listed the delay associated with the components on the left. To the extreme right is the name *Path*. Below this is the total time the signal has spent in the path at a given point. To the right of the numbers is an *r* if the signal is rising, or *f* if the signal is falling.The *data arrival time* represents the time it takes the signal to traverse the given path. In the *Timing* report, this is the critical path. Below this is timing related to the clock. The *data required time* represents the minimum amount of time found between the rising and falling clock edges. The *slack* is the amount of wasted clocking time. It tells us that we could decrease the clock by the given amount, and the circuit would still operate properly.



```
Operating Conditions: CDA_TYPICAL   Library: hp.4u3m1p_10_std_sd
Wire Load Model Mode: enclosed

  Startpoint: lite_state_s_reg[1]
              (rising edge-triggered flip-flop clocked by clock)
  Endpoint: lite_state_s_reg[1]
            (rising edge-triggered flip-flop clocked by clock)
  Path Group: clock
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  traffic            CDA_WIRE1             hp.4u3m1p_10_std_sd

  Point                                                   Incr       Path
  --------------------------------------------------------------------------
  clock clock (rise edge)                                 0.00       0.00
  clock network delay (ideal)                             0.00       0.00
  lite_state_s_reg[1]/CLK (stdtgdff_q_2x)                 0.00       0.00 r
  lite_state_s_reg[1]/QBAR (stdtgdff_q_2x)                0.71       0.71 f
  U14/Y (stdnand2_2x)                                     0.21       0.92 r
  U34/Y (stdinv_2x)                                       0.04       0.96 f
  U38/Y (stdor2_3x)                                       0.29       1.25 f
  U17/Y (stdinv_3x)                                       0.09       1.34 r
  U13/Y (stdnor2_1x)                                      0.10       1.45 f
  U32/Y (stdaoai211_1x)                                   0.32       1.77 r
  U31/Y (stdnand2_2x)                                     0.10       1.87 f
  U20/Y (stdand2_1x)                                      0.31       2.18 f
  U25/Y (stdoaoi211_1x)                                   0.30       2.48 r
  lite_state_s_reg[1]/D (stdtgdff_q_2x)                   0.00       2.48 r
  data arrival time                                                  2.48

  clock clock (rise edge)                                 5.00       5.00
  clock network delay (ideal)                             0.00       5.00
  clock uncertainty                                      -0.25       4.75
  lite_state_s_reg[1]/CLK (stdtgdff_q_2x)                 0.00       4.75 r
  library setup time                                     -0.05       4.70
  data required time                                                 4.70
  --------------------------------------------------------------------------
  data required time                                                 4.70
  data arrival time                                                 -2.48
  --------------------------------------------------------------------------

  slack (MET)                                                        2.22
```



```
Point                                                   Incr       Path
------------------------------------------------------------------------
clock clk (rise edge)                                   0.00       0.00
clock network delay (propagated)                        0.10       0.10
RdWrCnt_reg[2]/CLK (stddff_q_6x)                        0.00       0.10 r
RdWrCnt_reg[2]/Q (stddff_q_6x)                       1712.32    1712.42 r
U340/Y (stdmux2_2x)                                  1173.87    2886.29 r
U215/Y (stdbufinv)                                    499.36    3385.65 f
r140/A[1] (conversion_DW01_inc_16_0)                    0.00    3385.65 f
r140/U19/Y (stdnand3)                                 860.94    4246.59 r
r140/U20/Y (stdbufinv)                                359.32    4605.91 f
r140/U26/Y (stdnand2_2x)                              387.95    4993.87 r
r140/U16/Y (stdbufinv)                                292.99    5286.86 f
r140/U27/Y (stdnand2_2x)                              387.95    5674.81 r
r140/U13/Y (stdbufinv)                                292.99    5967.80 f
r140/U28/Y (stdnand2_2x)                              347.53    6315.33 r
r140/U10/Y (stdbufinv)                                397.96    6713.29 f
r140/U29/Y (stdnand2_2x)                              475.33    7188.61 r
r140/U9/Y (stdbufinv)                                 292.99    7481.61 f
r140/U30/Y (stdnand2_2x)                              347.53    7829.13 r
r140/U6/Y (stdbufinv)                                 276.35    8105.49 f
r140/U31/Y (stdnand2_2x)                              399.86    8505.34 r
r140/U5/Y (stdbufinv)                                 279.47    8784.82 f
r140/U53/Y (stdnand2_2x)                              314.90    9099.72 r
r140/U41/Y (stdxnor2)                                 666.21    9765.93 r
r140/SUM[15] (conversion_DW01_inc_16_0)                 0.00    9765.93 r
U254/Y (stdao22_3x)                                   402.76   10168.69 r
rd_reg[15]/D (stddff_q)                                 0.19   10168.88 r
data arrival time                                              10168.88

clock clk (rise edge)                               20000.00   20000.00
clock network delay (propagated)                        0.10   20000.10
clock uncertainty                                   -1000.00   19000.10
rd_reg[15]/CLK (stddff_q)                               0.00   19000.10 r
library setup time                                   -482.00   18518.10
data required time                                             18518.10
------------------------------------------------------------------------
data required time                                             18518.10
data arrival time                                             -10168.88
------------------------------------------------------------------------
```

The slack given at the bottom of the report should say "MET". This means that signal is present for enough duration for the process to be done. The term data **Required Time** is the time by which the signal has to settle down into a steady state and the **Arrival Time** is the time by which the signal settles down into a steady state. When the report says “slack met” it means that the required time is larger than the arrival time, hence modules receive the signals just right for the process to be done.

Make a note of the arrival time and the required time. Let’s say that our arrival time is 10000 ps and that the slack was met. Click on

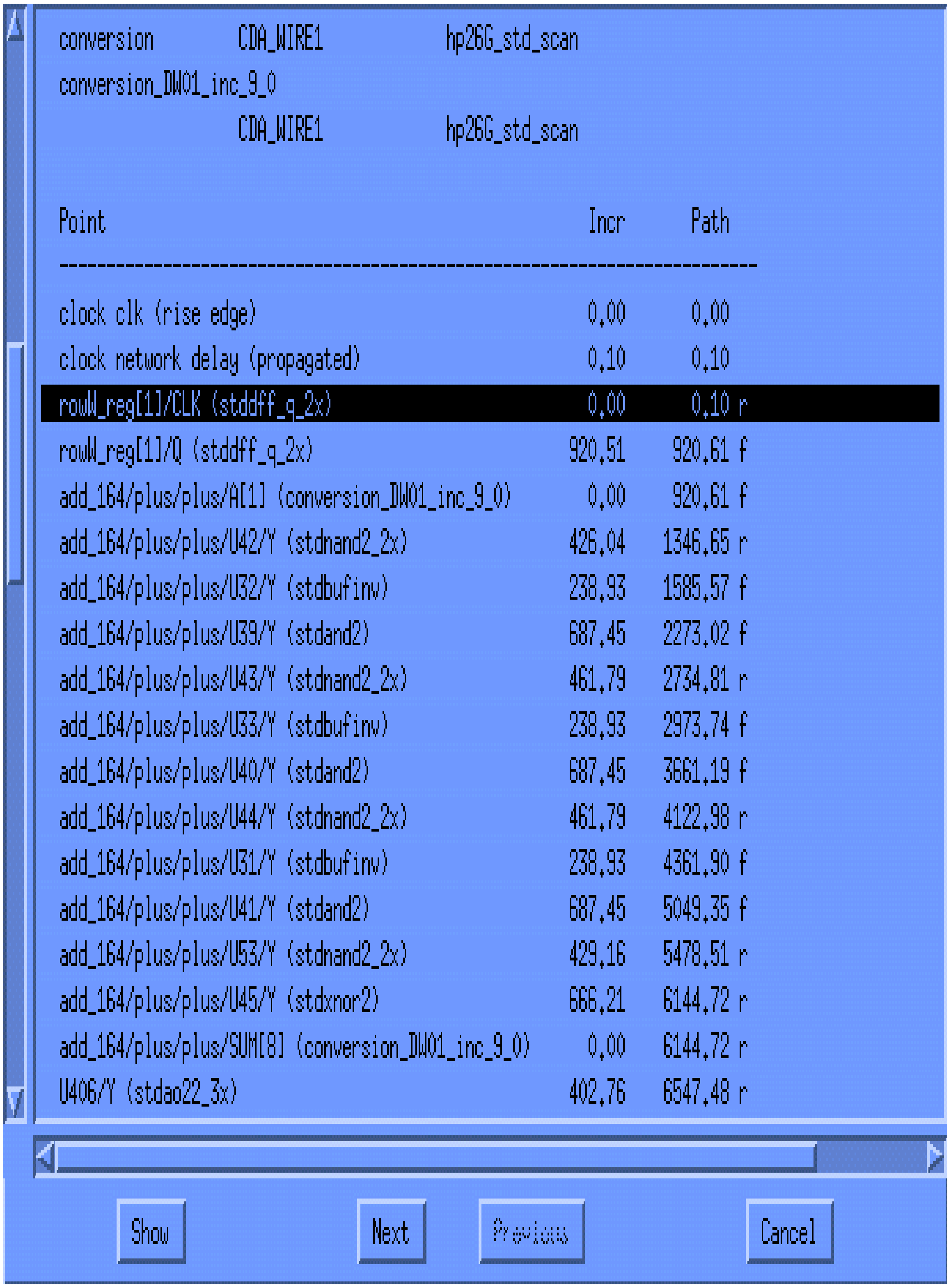
Attributes ----> Optimization Constraints ---> Timing Constraints

Now double click on the row that follows the clock attrributes in the report.

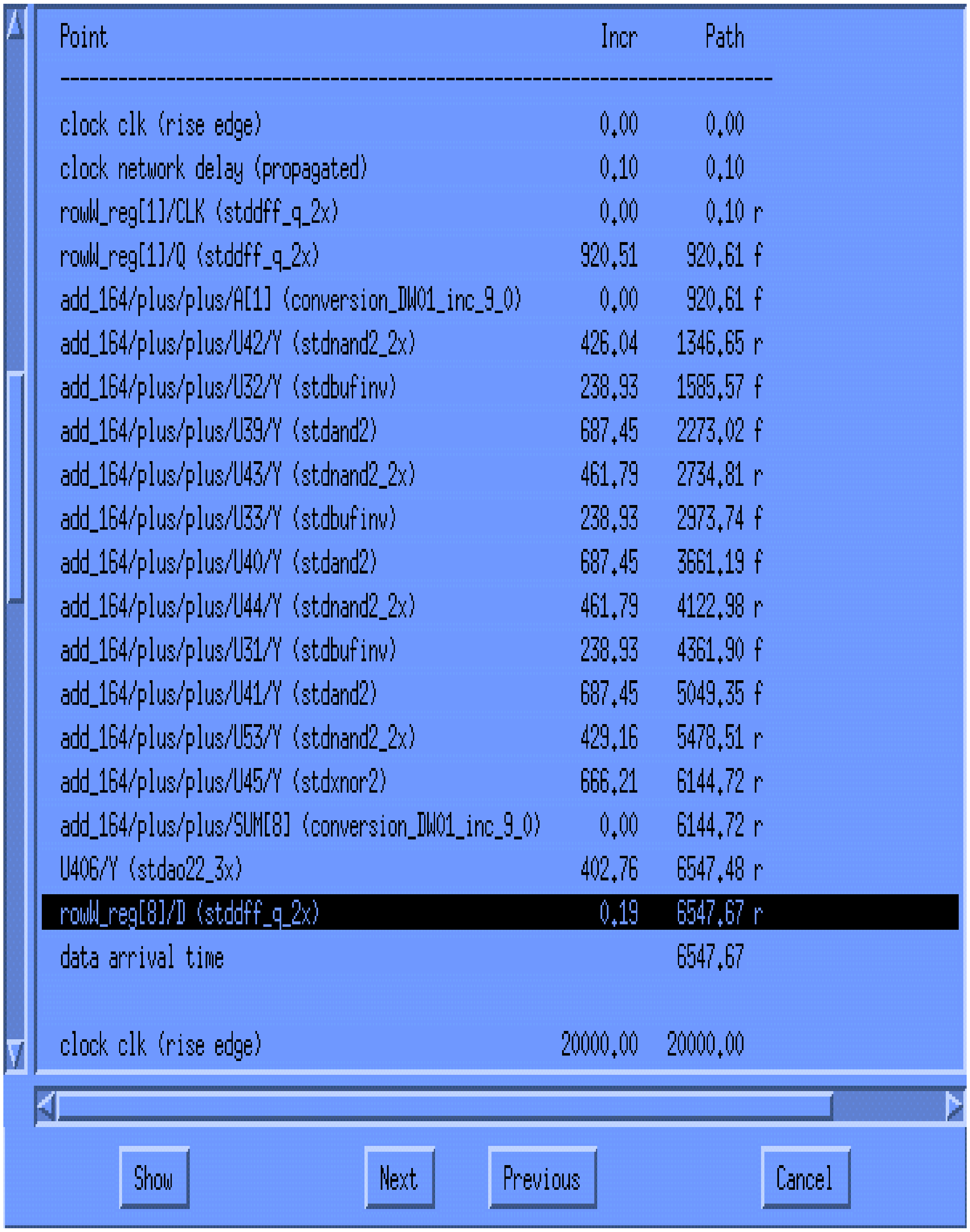
The figure given below should help in figuring out where to click. You would find the row attribute you have clicked on is loaded into the window for timing constraints. Also double click on the row that’s just above the row that says "data arrival time". Once the start and end point have been choosen, we then set the maximum and minimum delay time in the timing constraints window. The delay we set are from the modules present in the "from" box to the modules present in the "To" box in the timing constraint window. Initially we would find no element present in the "from" box, so click on the first element (row) in the "to" box, and click on the up arrow present. This would move that module to the "from" box.

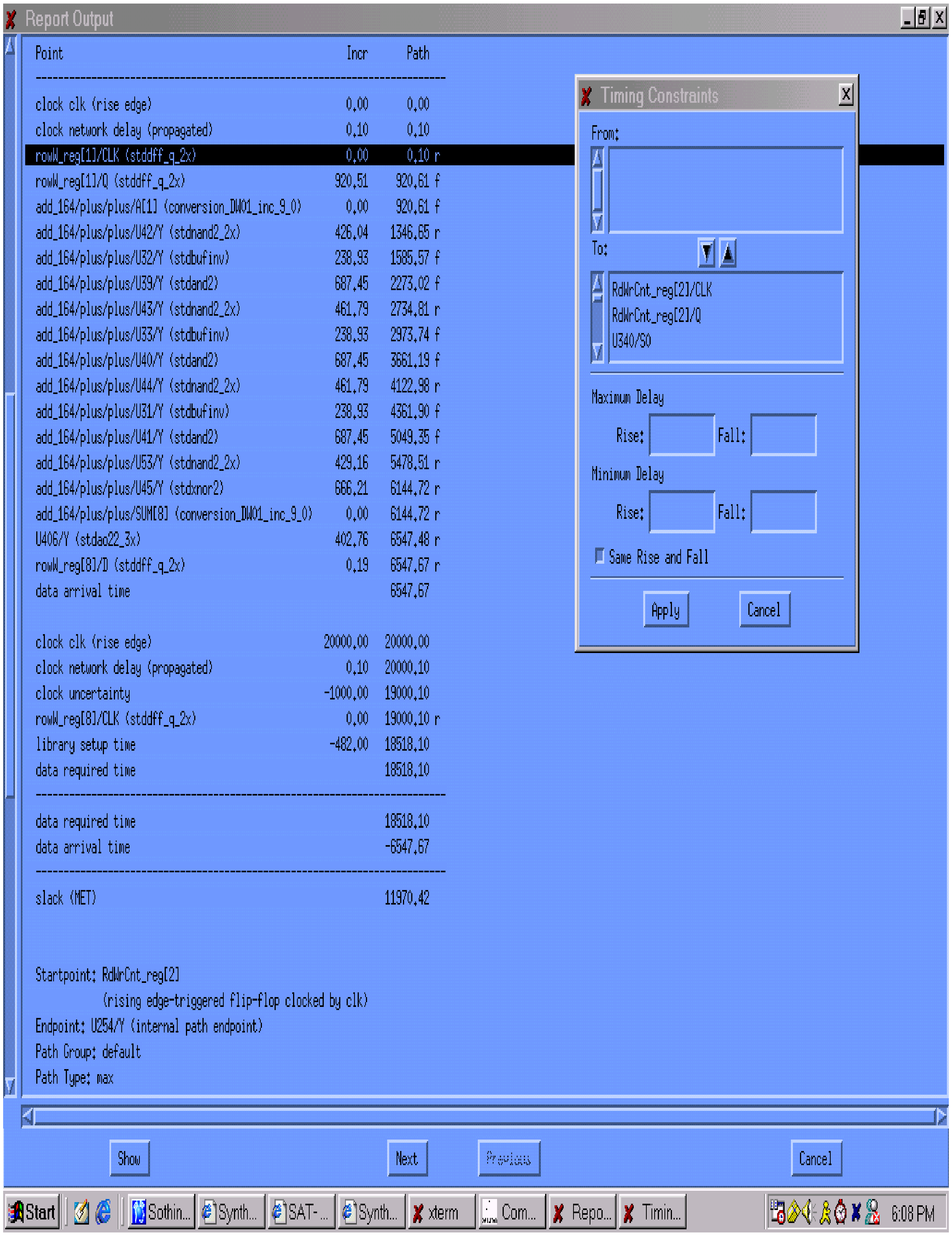
Now go ahead and set the value for the max delay and min delay. As assumed earlier, let’s have the arrival time to be 10000ps (we can find what the arrival time in our case from the report). Then to get a smaller delay netlist, type a value that is smaller than the arrival time previously got.

If you look at the report after optimizing, you would find that the arrival time has reduced. Make sure that the slack is met. If the slack is not met then use a smaller value for "Max Delay".



```
conversion         CDA_WIRE1           hp26G_std_scan
conversion_DW01_inc_9_0
                   CDA_WIRE1           hp26G_std_scan


Point                                          Incr       Path
-------------------------------------------------------------------------
clock clk (rise edge)                          0.00       0.00
clock network delay (propagated)               0.10       0.10
rowW_reg[1]/CLK (stddff_q_2x)                  0.00       0.10 r
rowW_reg[1]/Q (stddff_q_2x)                  920.51     920.61 f
add_164/plus/plus/A[1] (conversion_DW01_inc_9_0)  0.00   920.61 f
add_164/plus/plus/U42/Y (stdnand2_2x)        426.04    1346.65 r
add_164/plus/plus/U32/Y (stdbufinv)          238.93    1585.57 f
add_164/plus/plus/U39/Y (stdand2)            687.45    2273.02 f
add_164/plus/plus/U43/Y (stdnand2_2x)        461.79    2734.81 r
add_164/plus/plus/U33/Y (stdbufinv)          238.93    2973.74 f
add_164/plus/plus/U40/Y (stdand2)            687.45    3661.19 f
add_164/plus/plus/U44/Y (stdnand2_2x)        461.79    4122.98 r
add_164/plus/plus/U31/Y (stdbufinv)          238.93    4361.90 f
add_164/plus/plus/U41/Y (stdand2)            687.45    5049.35 f
add_164/plus/plus/U53/Y (stdnand2_2x)        429.16    5478.51 r
add_164/plus/plus/U45/Y (stdxnor2)           666.21    6144.72 r
add_164/plus/plus/SUM[8] (conversion_DW01_inc_9_0)  0.00  6144.72 r
U406/Y (stdao22_3x)                          402.76    6547.48 r
```


```
Point                                                Incr       Path
--------------------------------------------------------------------------
clock clk (rise edge)                                0.00       0.00
clock network delay (propagated)                     0.10       0.10
rowW_reg[1]/CLK (stddff_q_2x)                        0.00       0.10 r
rowW_reg[1]/Q (stddff_q_2x)                        920.51     920.61 f
add_164/plus/plus/A[1] (conversion_DW01_inc_9_0)     0.00     920.61 f
add_164/plus/plus/U42/Y (stdnand2_2x)              426.04    1346.65 r
add_164/plus/plus/U32/Y (stdbufinv)                238.93    1585.57 f
add_164/plus/plus/U39/Y (stdand2)                  687.45    2273.02 f
add_164/plus/plus/U43/Y (stdnand2_2x)              461.79    2734.81 r
add_164/plus/plus/U33/Y (stdbufinv)                238.93    2973.74 f
add_164/plus/plus/U40/Y (stdand2)                  687.45    3661.19 f
add_164/plus/plus/U44/Y (stdnand2_2x)              461.79    4122.98 r
add_164/plus/plus/U31/Y (stdbufinv)                238.93    4361.90 f
add_164/plus/plus/U41/Y (stdand2)                  687.45    5049.35 f
add_164/plus/plus/U53/Y (stdnand2_2x)              429.16    5478.51 r
add_164/plus/plus/U45/Y (stdxnor2)                 666.21    6144.72 r
add_164/plus/plus/SUM[8] (conversion_DW01_inc_9_0)   0.00    6144.72 r
U406/Y (stdao22_3x)                                402.76    6547.48 r
rowW_reg[8]/D (stddff_q_2x)                          0.19    6547.67 r
data arrival time                                            6547.67

clock clk (rise edge)                             20000.00   20000.00
```



```
Point                                               Incr       Path
-----------------------------------------------------------------------
clock clk (rise edge)                               0.00       0.00
clock network delay (propagated)                    0.10       0.10
rowW_reg[1]/CLK (stddff_q_2x)                       0.00       0.10 r
rowW_reg[1]/Q (stddff_q_2x)                       920.51     920.61 f
add_164/plus/plus/A[1] (conversion_DW01_inc_9_0)    0.00     920.61 f
add_164/plus/plus/U42/Y (stdnand2_2x)             426.04    1346.65 r
add_164/plus/plus/U32/Y (stdbufinv)               238.93    1585.57 f
add_164/plus/plus/U39/Y (stdand2)                 687.45    2273.02 f
add_164/plus/plus/U43/Y (stdnand2_2x)             461.79    2734.81 r
add_164/plus/plus/U33/Y (stdbufinv)               238.93    2973.74 f
add_164/plus/plus/U40/Y (stdand2)                 687.45    3661.19 f
add_164/plus/plus/U44/Y (stdnand2_2x)             461.79    4122.98 r
add_164/plus/plus/U31/Y (stdbufinv)               238.93    4361.90 f
add_164/plus/plus/U41/Y (stdand2)                 687.45    5049.35 f
add_164/plus/plus/U53/Y (stdnand2_2x)             429.16    5478.51 r
add_164/plus/plus/U45/Y (stdxnor2)                666.21    6144.72 r
add_164/plus/plus/SUM[8] (conversion_DW01_inc_9_0)  0.00    6144.72 r
U406/Y (stdao22_3x)                               402.76    6547.48 r
rowW_reg[8]/D (stddff_q_2x)                         0.19    6547.67 r
data arrival time                                           6547.67

clock clk (rise edge)                            20000.00   20000.00
clock network delay (propagated)                    0.10   20000.10
clock uncertainty                               -1000.00   19000.10
rowW_reg[8]/CLK (stddff_q_2x)                       0.00   19000.10 r
library setup time                               -482.00   18518.10
data required time                                         18518.10
-----------------------------------------------------------------------
data required time                                         18518.10
data arrival time                                          -6547.67
-----------------------------------------------------------------------
slack (MET)                                                11970.42


Startpoint: RdWrCnt_reg[2]
            (rising edge-triggered flip-flop clocked by clk)
Endpoint: U254/Y (internal path endpoint)
Path Group: default
Path Type: max
```

```
From:

To:
RdWrCnt_reg[2]/CLK
RdWrCnt_reg[2]/Q
U340/S0

Maximum Delay
  Rise:          Fall:
Minimum Delay
  Rise:          Fall:
☐ Same Rise and Fall
```

When performing the synthesis, we may find that the tool is inserting delay between the modules. This is because the value that we have set up has resulted in some of the modules processing the data faster (because of sizing) and the data is available to the next module even before it is ready to handle them.

Attributes --->Optimization Constraints ---> Derive

option also sets up delay optimization attributes for the tool. The command derives timing constraints from the existing timing of the design. Both sequential and combinational timing constraints are derived from all previously unconstrained timing paths in the design. There options within this command are Minimum delay, Maximum delay and Maximim period. These constraints use a scaling factor, which is multiplied with the timing and placed into the design. In the case of Minimum delay, a scaling value less than one means less restrictive timing than the actual timing of the design. In the case of maximum delay and period, it’s the opposite.

If the use of this command results in "Slack" not being met, then click on

Attributes ---> Operating Environment ---> Input/Output Delay

Here we can induce input/output delays between various points in the design. This is a tedious process.

Commands used with area and power optimization can be used along with the delay optimization commands to acheive a netlist optimized in all three design spaces.

The above mentioned are some of the ways by which you can optimize a netlist for area, power and delay. Options like wire loads, operating conditions, capacitive loads etc can also be used to change the design space parameters of the design (but things get tougher since more the constraints, more care needs to be taken to see that there are no violations).

And finally if we want to save the commands that we had used in a script which can be executed, in the command widow type

history -h > script.scr

This is the way where we do not have to do the repetitive things like analyze etc, instead run the script.

CONCLUSION

This report serves as a good hands-on tutorial for learning Synopsys tool. Due to license limitations, few products such as PrimeTime where timing analysis can better be analysed couldn't be exploited to its maximum. Similarly, the product Formality also lacks license which could be better used as a good place and route tool. This helps in floorplanning. This report gives a tour of Synopsys for VHDL and Logic synthesis with the help of an example.

REFERENCES

1) **VHDL CODING AND LOGIC SYNTHESIS WITH SYNOPSYS** --- Weng Fook Lee.
2) **LOGIC SYNTHESIS USING SYNOPSYS** --- Pran Kurup, Taher Abbasi.
3) **ADVANCED ASIC CHIP SYNTHESIS USING SYNOPSYS DESIGN COMPILER, PHYSICAL COMPILER AND PRIMETIME** --- Himanshu Bhatnagar.
4) http://www.ee.vt.edu/~ha/cadtools/synopsys/da/html
5) http://www.scudc.scu.edu/mentortu/synopsys_tutorial.html
6) **SYNTHESIZING AN ASIC FOR VARIOUS AREA, POWER AND DELAY** --- Sowmyan Rajagopalan
7) **DIGITAL LOGIC SYNTHESIS USING SYNOPSYS- A TUTORIAL** --- Ted Obuchowicz.
8) **DESIGN ANALYSER** --- Reference Manual, Synopsys Online Documentation.