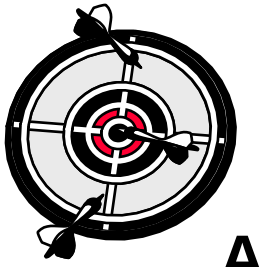
RTL Synthesis using Design Compiler

Dr Basel Halak

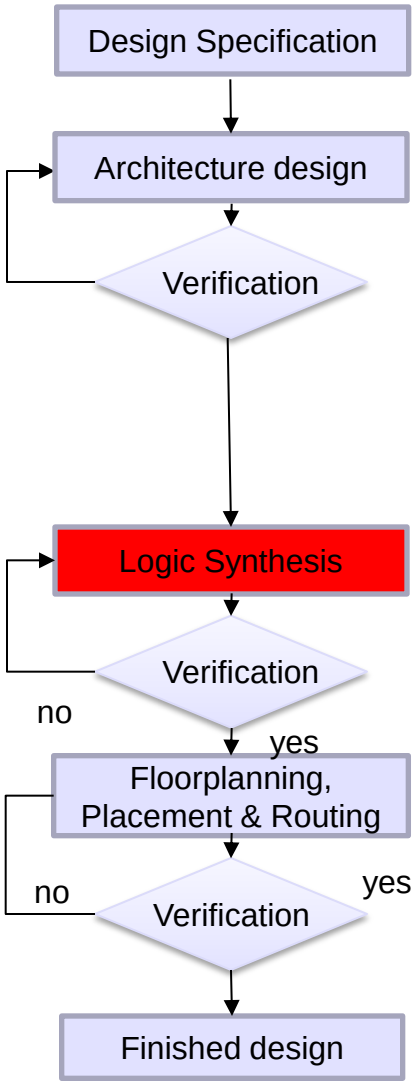
Learning Outcomes:



After completing this unit, you should be able to:

1. **Set up the DC RTL Synthesis Software and run synthesis tasks**
2. **Synthesize a simple RTL design**
3. **Create your own scripts**
4. **Carry out basic timing and power analysis based on the results**

Digital Design Flow

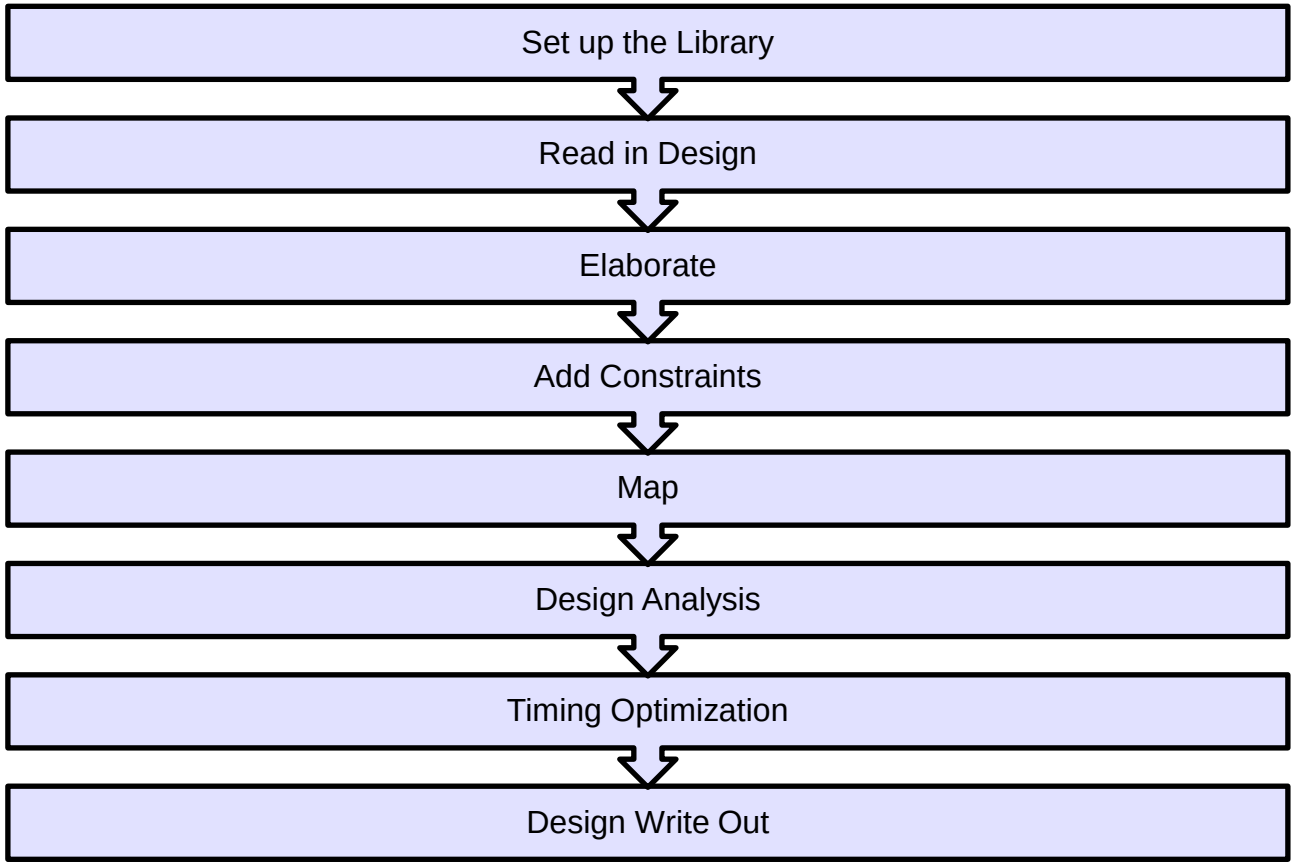


Design Compiler Lab Instructions

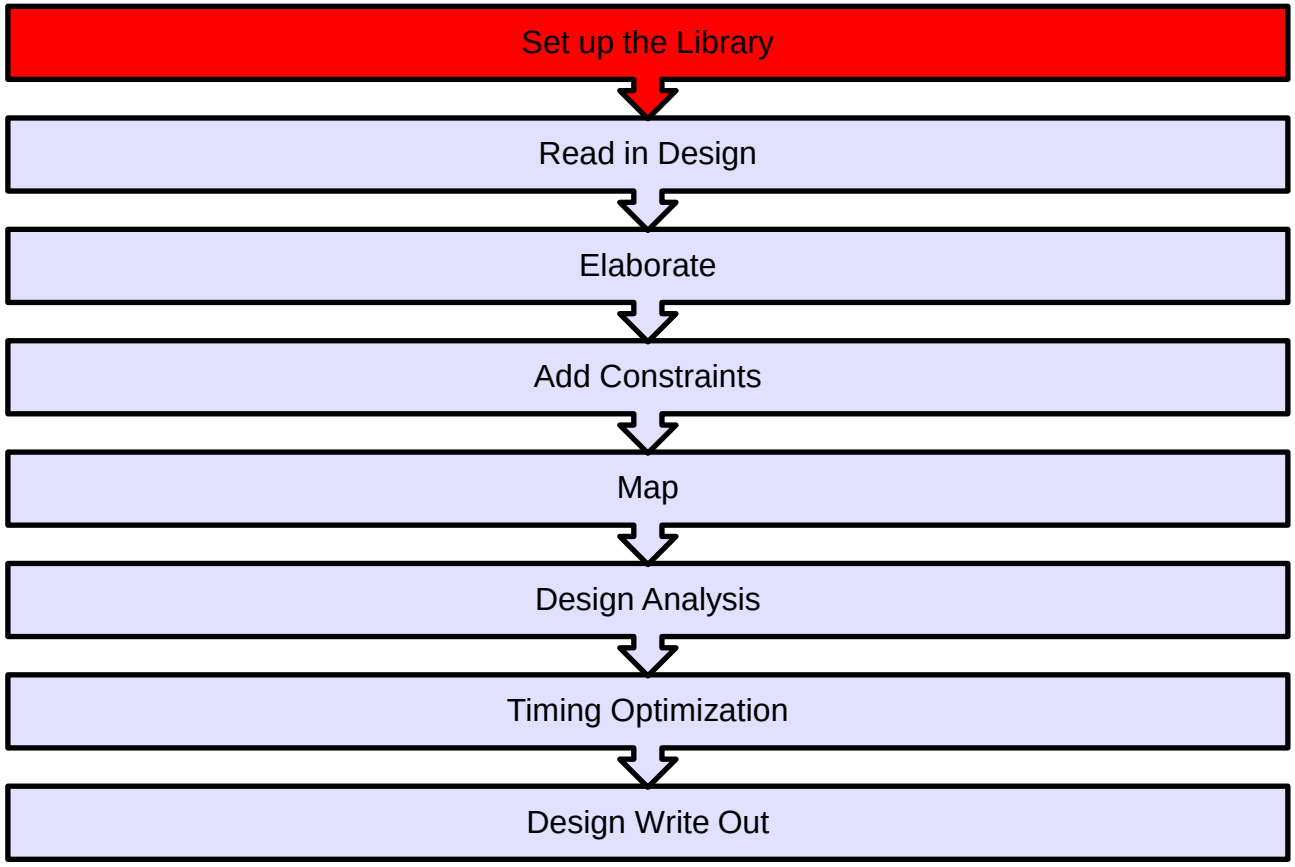
For this lab you will need:

1. A synthesizable Verilog of the design (this is provided)
2. Tool Setup File (DC_Setup.sh) (this is provided)
3. Library Setup File (this is provided)

Synthesis Process

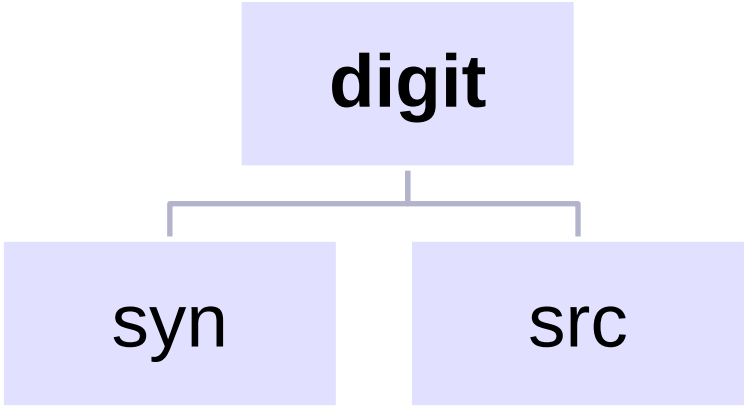


Synthesis Process



Design Directory Management

1. Create a directory for your digital design project (e.g. digit)
2. Inside this directory create a sub-directory called src and copy your hdl design files into it.
3. Also create a sub-directory for your synthesis files and call it syn and copy your hdl design files into it. You also need to save the tool setup file “DC_Setup.sh” in this folder.



How to setup Linux Environment to run DC

1. **Move into syn directory and type the following commands:**

source DC_Setup.sh

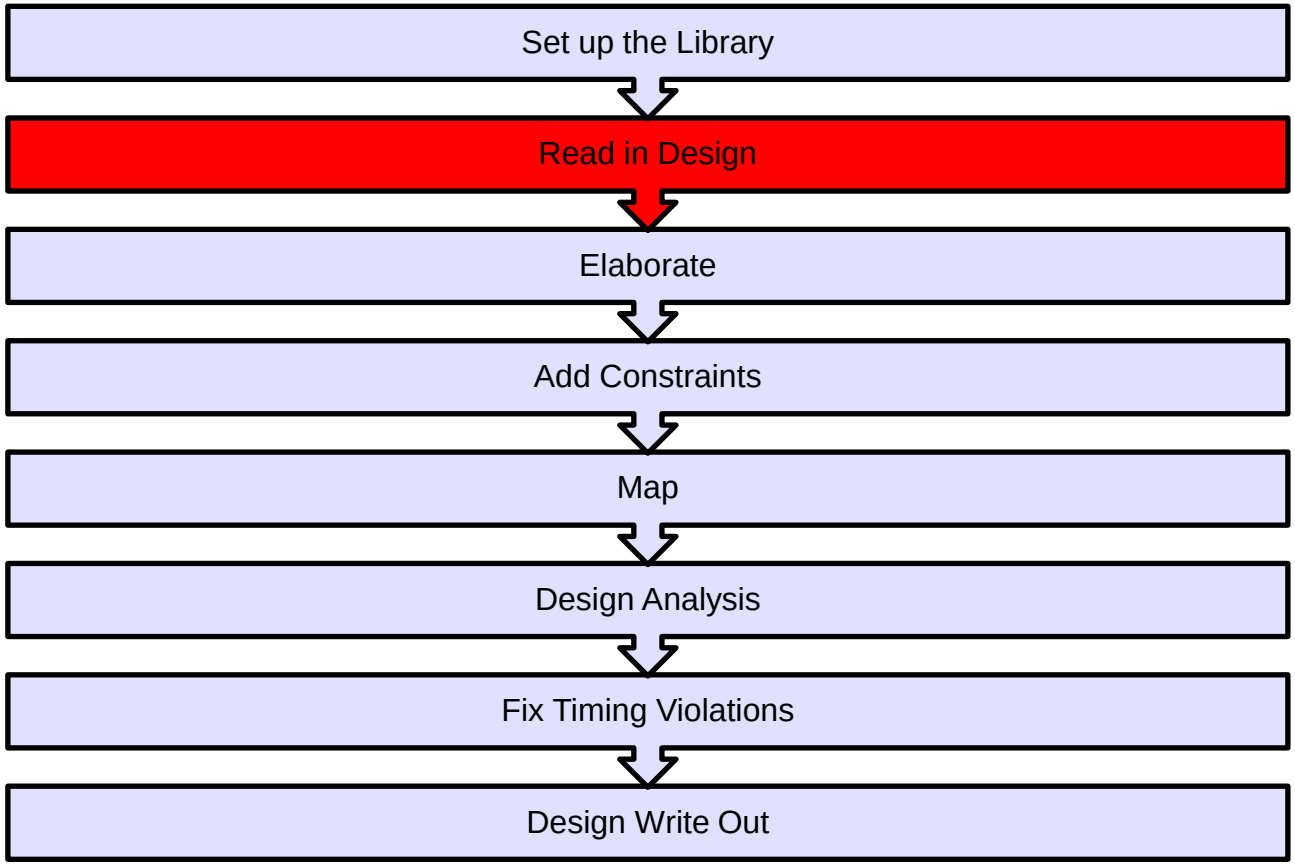
2. **If you have done this part correctly, you should be able to run dc-shell without any error massages, by typing**

dc_shell -gui

3. Setup the tool to use 0.35 um AMS technology library by typing the following command in the dc_shell command line:

source library_Setup

Synthesis Process



2. Read in the HDL

- The design example is in verilog so you need to type

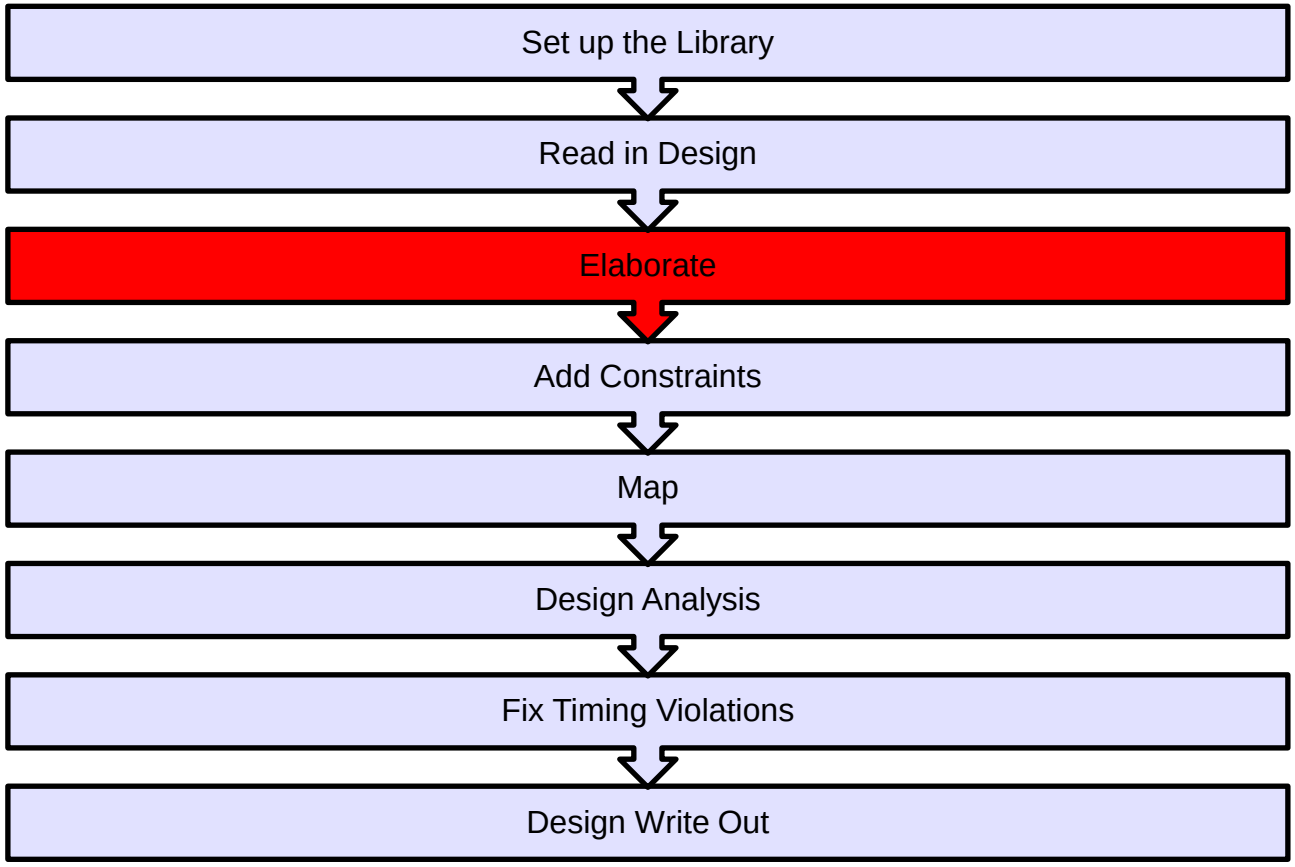
analyze -format verilog {./qmults.v}

You should get the following message :

Presto compilation completed successfully.

1

Synthesis Process



3. Elaboration

- Elaboration is the stage where the design is translated into a series of graphs, which can then be mapped onto an optimal structure
- The basic command is very simple:

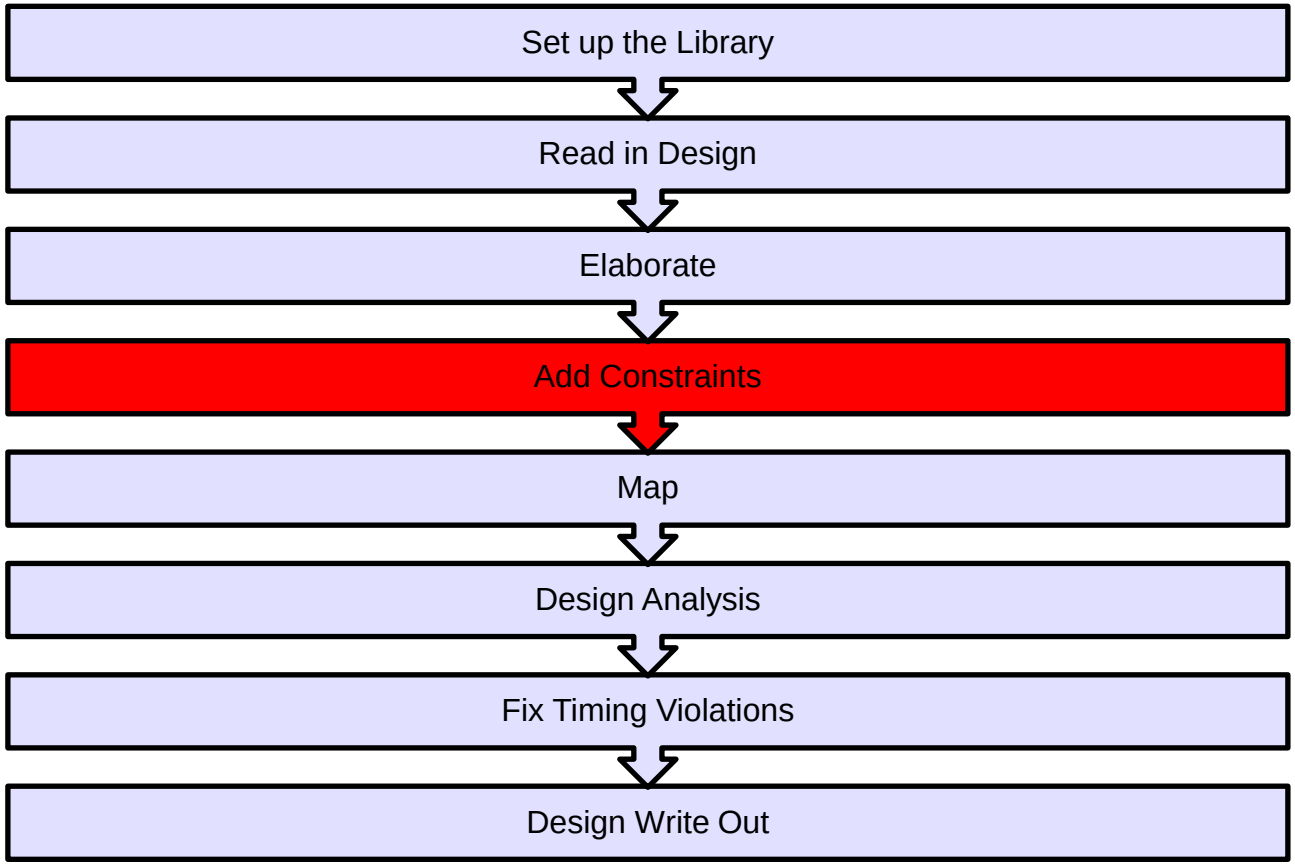
  elaborate MODULENAME

- In this case you need to type:

  elaborate qmults -architecture verilog -library DEFAULT

- Click on the following link to see how the schematic should look like:
  - **Schematic.pdf**

Synthesis Process



Add Constraints

- You can use constraints to control the action of the compiler.
- Constraints describe the surrounding environment of the circuit, such as loads and drives of IO, and clock characteristics.
- By default, Design Compiler will not constrain any paths. If you issue the command:

check_timing

You will get the following output

Warning: The following end-points are not constrained for maximum delay.

End point
----
o_complete
o_overflow
o_result_out[0]
o_result_out[1]
o_result_out[2]
……

Add Constraints: Defining the clock

- Most of the paths can be constrained by defining a clock.
- Example: Create a clock called “i_clock", applied to the input port "Clock", with a period of 8 ns with the following command:

```
create_clock i_clk -name i_clk  -period 8
```

Add Constraints: Optimize Area

- **set_max_area**: This constraint specifies the maximum area a particular design should have. The value is specified in units used to describe the gate-level macro cells in the technology library.
- Example: to minimise the design area, we issue the commands

set_max_area

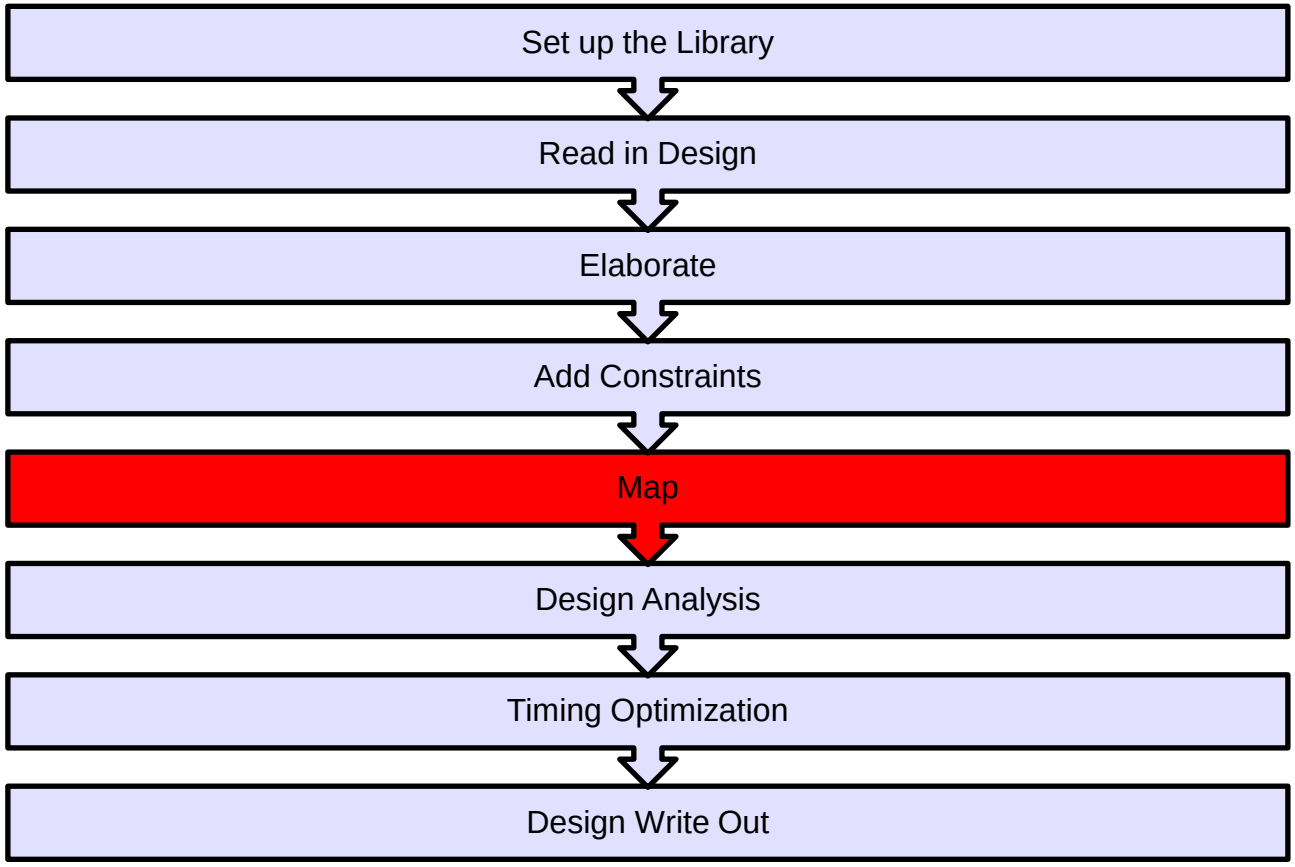
Add Constraints: Optimize Area

- **set_max_area**: This constraint specifies the maximum area a particular design should have. The value is specified in units used to describe the gate-level macro cells in the technology library.
- Example: to minimise the design area, we issue the commands

set_max_area 0

This will instruct design compiler to use as less area as possible.

Synthesis Process



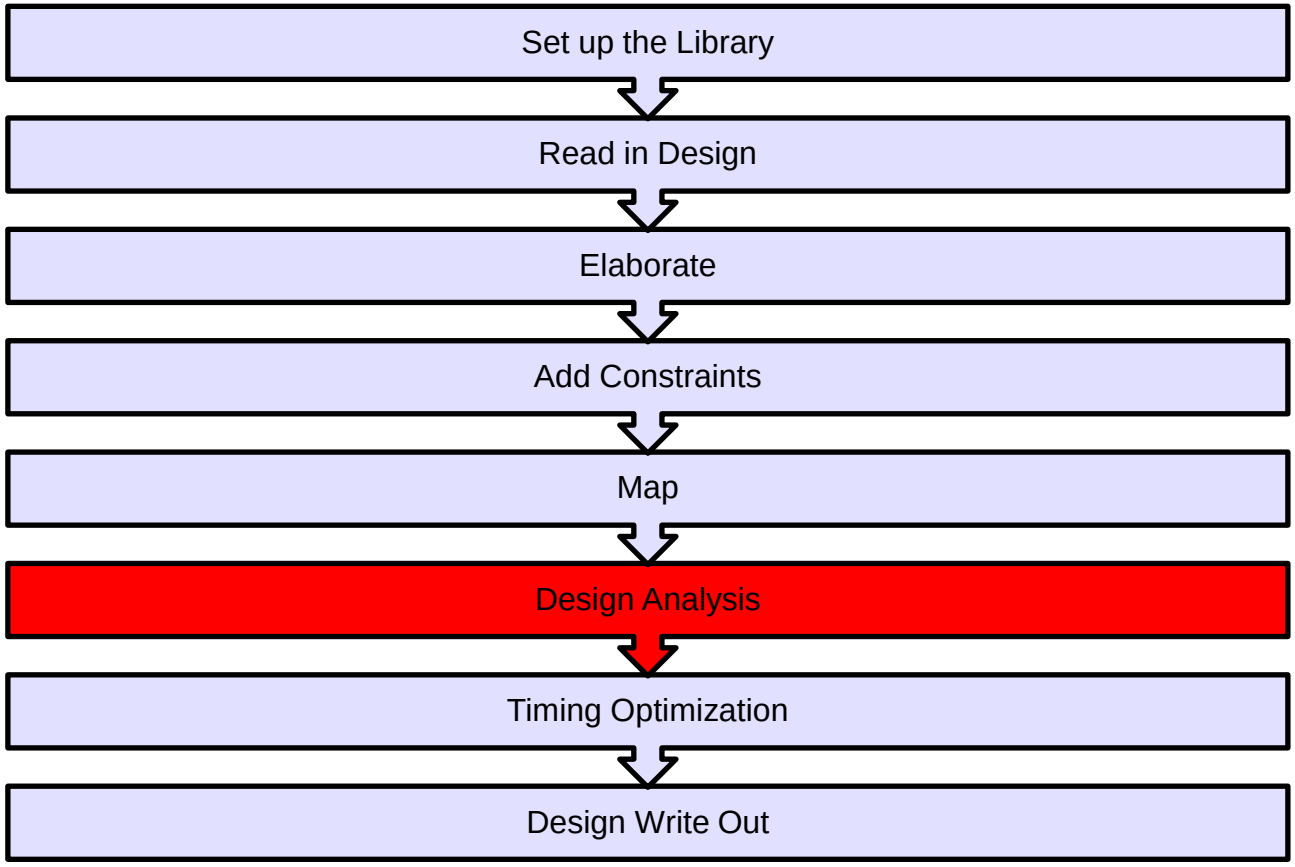
4. Synthesis

- In this step the logic functionality of the design will be implemented in terms of cell instances from the process specific library.
- This is done using the command:

compile

This command may take few minutes depending on the size of you design

Synthesis Process



Design Analysis

- We can apply a range of analysis capabilities to our design to check that it meets our requirements,
- To get a quick summary of the design performance and area statues we can use the command

report_qor

This will provide information about the timing information, critical path slack, critical path clock period, total design area and information about the CPU statistics

```
****************************************
Report : qor
Design : qmults
Version: C-2009.06-SP4
Date   : Mon Jul 21 17:19:31 2014
****************************************
  Timing Path Group 'i_clk'
  -----------------------------------
  Levels of Logic:           20.00
  Critical Path Length:       5.96
  Critical Path Slack:        0.04
  Critical Path Clk Period:   6.00
  Total Negative Slack:       0.00
  No. of Violating Paths:     0.00
  Worst Hold Violation:       0.00
  Total Hold Violation:       0.00
  No. of Hold Violations:     0.00

.....................


  Area
  -----------------------------------
  Combinational Area:   78696.799866
  Noncombinational Area: 54345.200165
  Net Area:           27702.000000
  -----------------------------------
  Cell Area:          133042.000031
  Design Area:        160744.000031

  Design Rules
  -----------------------------------
  Total Number of Nets:       1557
  Nets With Violations:       0
  -----------------------------------

.....................
```

Design Analysis: Power Report

- To get a quick summary of the design power statues we can use the command

report_power

This will provide information about dynamic and leakage power.

Design Analysis: Power Report

```
****************************************
****************************************
Report : power
       -analysis_effort low
Design : qmults
Version: C-2009.06-SP4
Date   : Mon Jul 21 17:22:44 2014
****************************************
Library(s) Used:
    c35_CORELIB (File:
/opt/esdcad/designkits/ams/v370/synopsys/c35_3.3V/c35_COR
ELIB.db)
operating Conditions: nom_pvt   Library: c35_CORELIB


……................................


……..........................................................
```

Continued:

```
Global Operating Voltage = 3.3
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  =  12.1502 mW   (77%)
  Net Switching Power  =   3.7034 mW   (23%)
                         ---------
Total Dynamic Power    =  15.8535 mW  (100%)


Cell Leakage Power     = 109.5970 nW
```

Design Analysis: Timing

- To report the timing of the design. we can use the command

report_timing

By default, the report_timing command displays information on the critical path or the timing path with the maximum delay.

Design Analysis: Timing

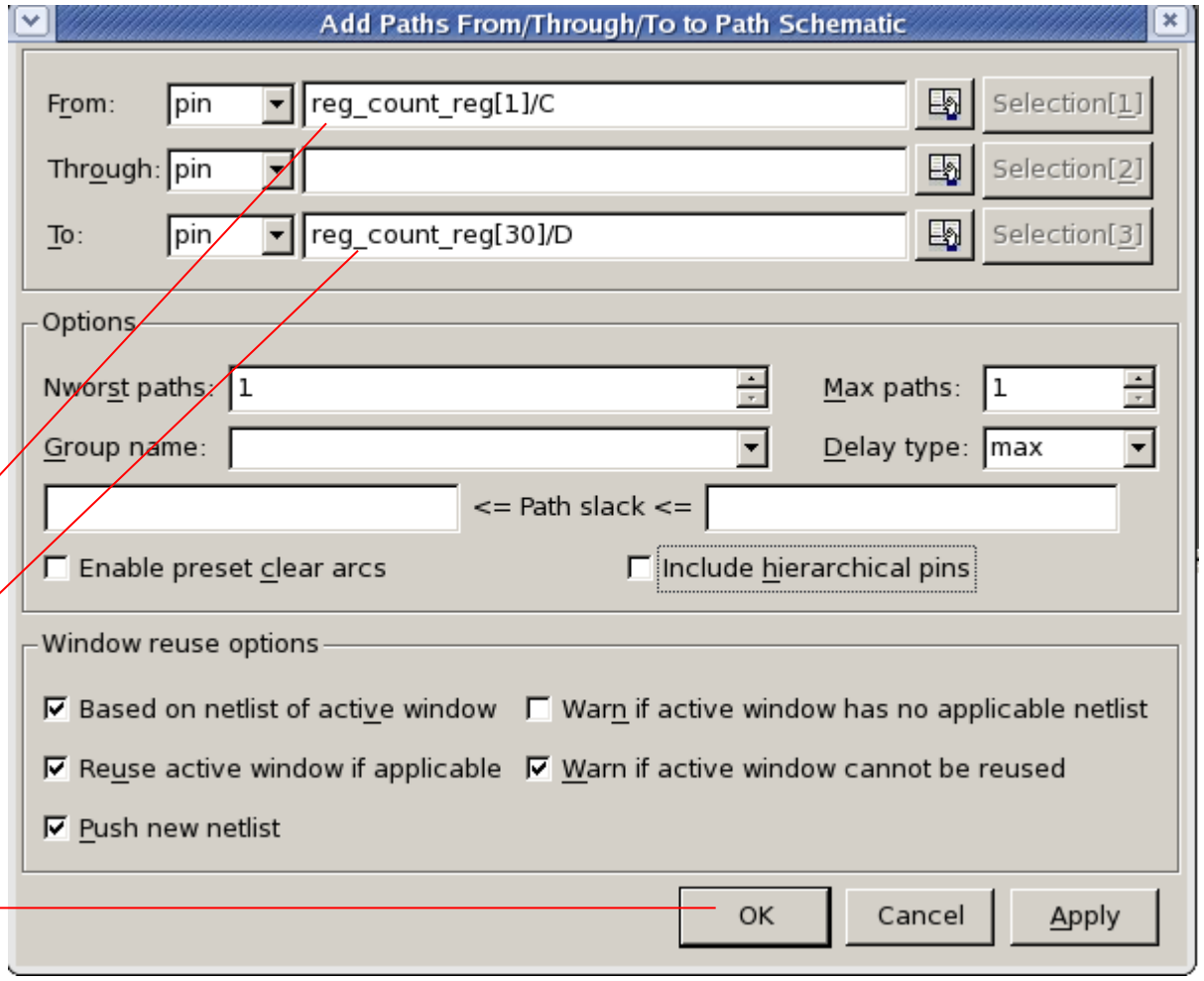
```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : qmults
Version: C-2009.06-SP4
Date   : Mon Jul 21 17:28:07 2014
****************************************

Operating Conditions: nom_pvt   Library: c35_CORELIB
Wire Load Model Mode: enclosed

  Startpoint: reg_count_reg[1]
              (rising edge-triggered flip-flop clocked by i_clk)
  Endpoint: reg_count_reg[30]
            (rising edge-triggered flip-flop clocked by i_clk)
  Path Group: i_clk
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  qmults             10k                   c35_CORELIB
  qmults_DW01_inc_1  10k                   c35_CORELIB

  Point                                    Incr       Path
  ------------------------------------------------------------
  clock i_clk (rise edge)                  0.00       0.00
  clock network delay (ideal)              0.00       0.00
  reg_count_reg[1]/C (DF3)                 0.00       0.00 r
  reg_count_reg[1]/Q (DF3)                 0.88       0.88 f
```

Continued:

```
  add_84/A[1] (qmults_DW01_inc_1)          0.00       0.88 f
  add_84/U62/Q (INV3)                      0.15       1.04 r
  add_84/U61/Q (NOR21)                     0.14       1.17 f
  add_84/U60/Q (NAND41)                    0.38       1.55 r
  add_84/U6/Q (BUF6)                       0.25       1.80 r
  add_84/U42/Q (NOR31)                     0.33       2.14 f
  add_84/U41/Q (INV3)                      0.21       2.35 r
  add_84/U46/Q (NOR31)                     0.35       2.70 f
  add_84/U45/Q (INV3)                      0.21       2.91 r
  add_84/U53/Q (NOR31)                     0.35       3.26 f
  add_84/U52/Q (INV3)                      0.21       3.48 r
  add_84/U49/Q (NOR31)                     0.37       3.85 f
  add_84/U5/Q (CLKIN6)                     0.15       4.00 r
  add_84/U2/Q (NOR31)                      0.34       4.34 f
  add_84/U10/Q (NAND22)                    0.37       4.71 r
  add_84/U13/Q (NOR31)                     0.28       4.99 f
  add_84/U12/Q (INV3)                      0.18       5.17 r
  add_84/U15/Q (NOR21)                     0.18       5.35 f
  add_84/U63/Q (XNR21)                     0.29       5.64 r
  add_84/SUM[30] (qmults_DW01_inc_1)       0.00       5.64 r
  U397/Q (NAND22)                          0.07       5.71 f
  U395/Q (NAND22)                          0.25       5.96 r
  reg_count_reg[30]/D (DF3)                0.00       5.96 r
  data arrival time                                   5.96

  clock i_clk (rise edge)                  6.00       6.00
  clock network delay (ideal)              0.00       6.00
  reg_count_reg[30]/C (DF3)                0.00       6.00 r
  library setup time                       0.00       5.99
  data required time                                  5.99
  ------------------------------------------------------------
  data required time                                  5.99
  data arrival time                                  -5.96
  ------------------------------------------------------------
  slack (MET)                                         0.04
```

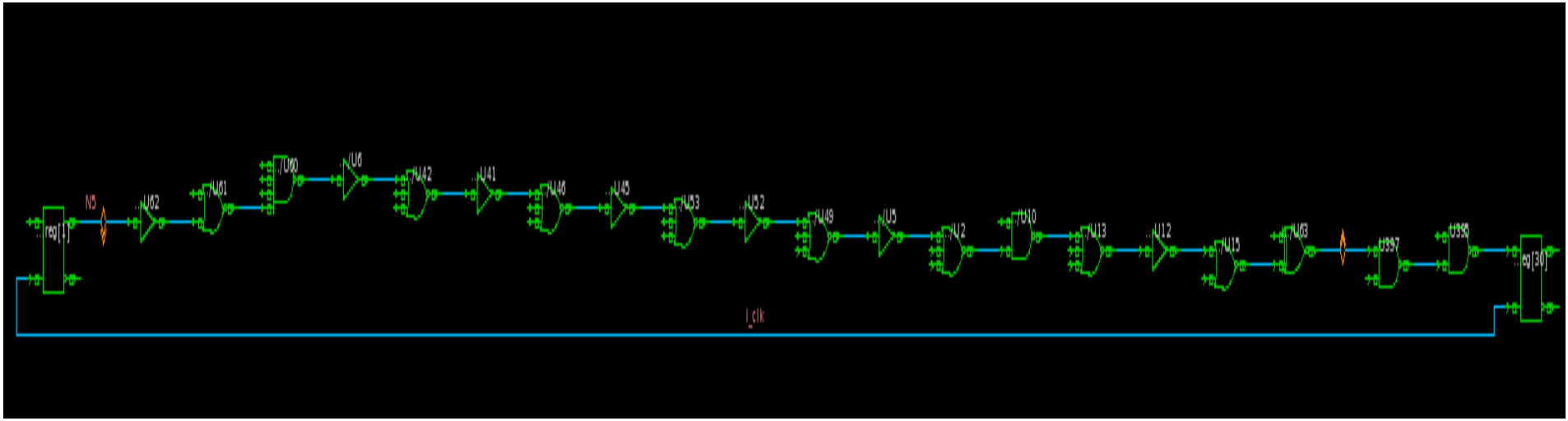
Design Analysis: Timing

- To examine the critical path graphically
- In design vision: go to the **Schematic** menu and choose the option **Add Paths From/To**
- Fill in the starting point and end point of the critical path from the previous timing report
- Click ok
- Click ok aging for the pop up window



Design Analysis: Timing

- **This will generate a graphical view of your critical path:**

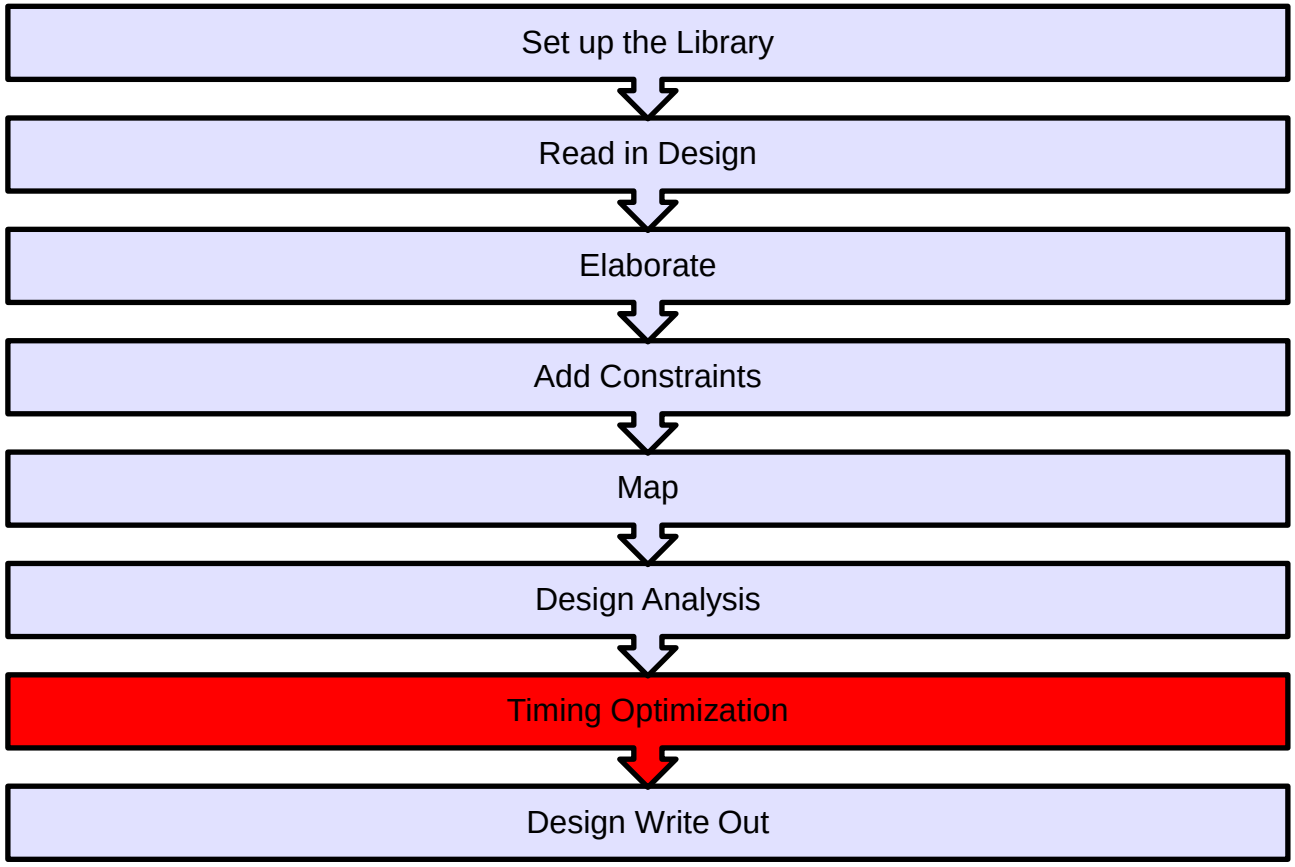


Design Analysis: Save Reports

- You can output the report to a file using:

```
report_area > synth_area.rpt
report_power > synth_power.rpt
report_timing > synth_timing.rpt
```

Synthesis Process



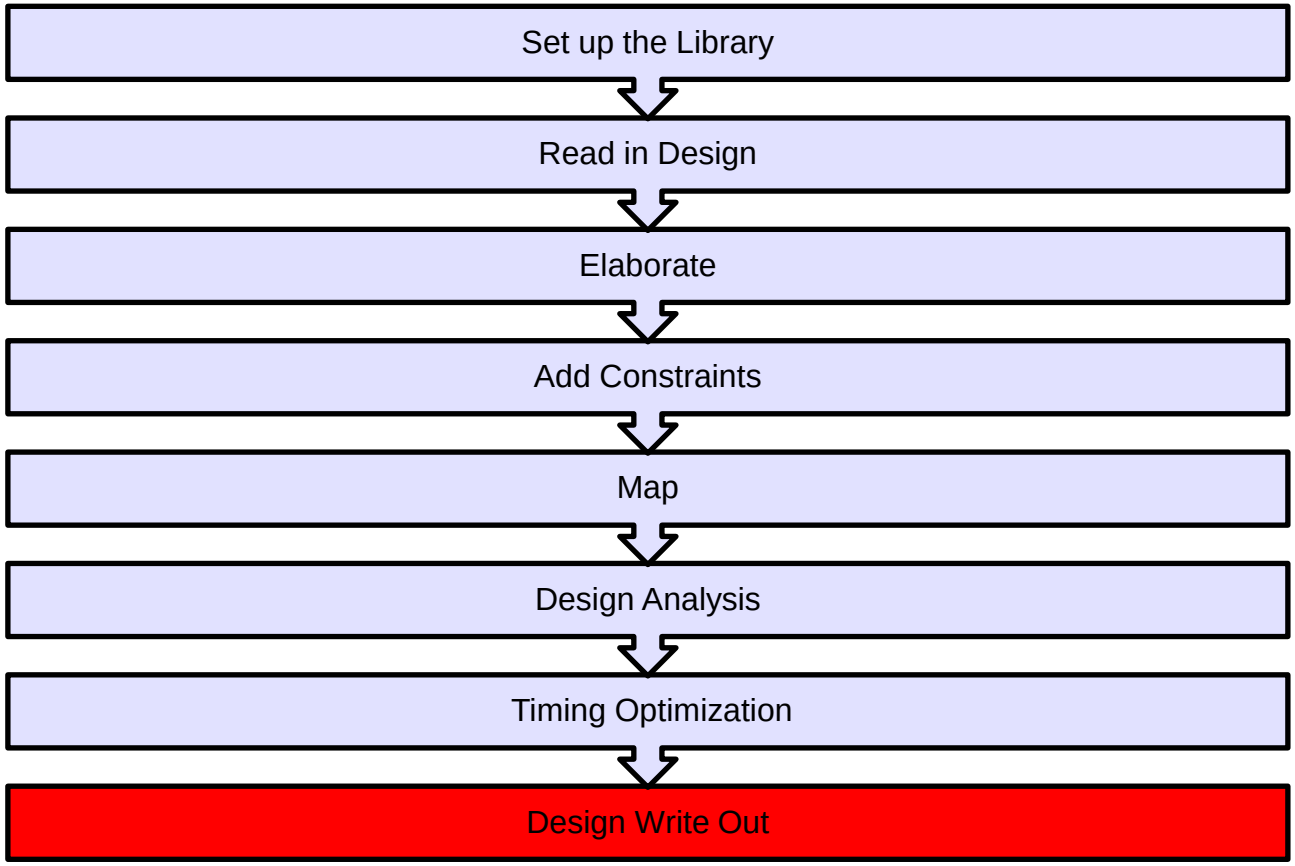
Timing Optimization

- **Optimise the performance of the design as follows:**

1. Change timing constraints to target higher clock frequencies
2. Use the techniques provided in appendix 1 to meet your target performance
3. Repeat the above process several times until you reach the maximum achievable frequency
4. Estimate area and power of the design for each target frequency

- **Make sure your design is free from Hold time violation (refer to appendix 1)**

Synthesis Process



Fix Naming

it is important to ensure that the naming styles of variable in the design are appropriatefor the target output language we are using (Verilog in this case). We can firstly see what names need changing:

report_names -rules verilog

If we are happy with the proposed new names we can perform the name changing process:

change_names -rules verilog -hierarchy -verbose

Save Out the Design:

This is the final step in the synthesis flow, it allows the designer to transfer the synthesised circuit to the next stage of the design flow. This can be done as follows:

1. **For Place and Route Stage: You need to save the following files:**

   1.1. Save the hierarchical Verilog:

   write -f verilog -hierarchy -output “qmults_syn.v"

   1.2. Save the timing constraints (sdc file)

   write_sdc design.sdc

2. For post synthesis simulation: You need to save the following files:

   1.1. Save the hierarchical Verilog:

   write -f verilog -hierarchy -output “qmults_syn.v"

   1.2. Save the timing information (sdf file)

   write_sdf design.sdf

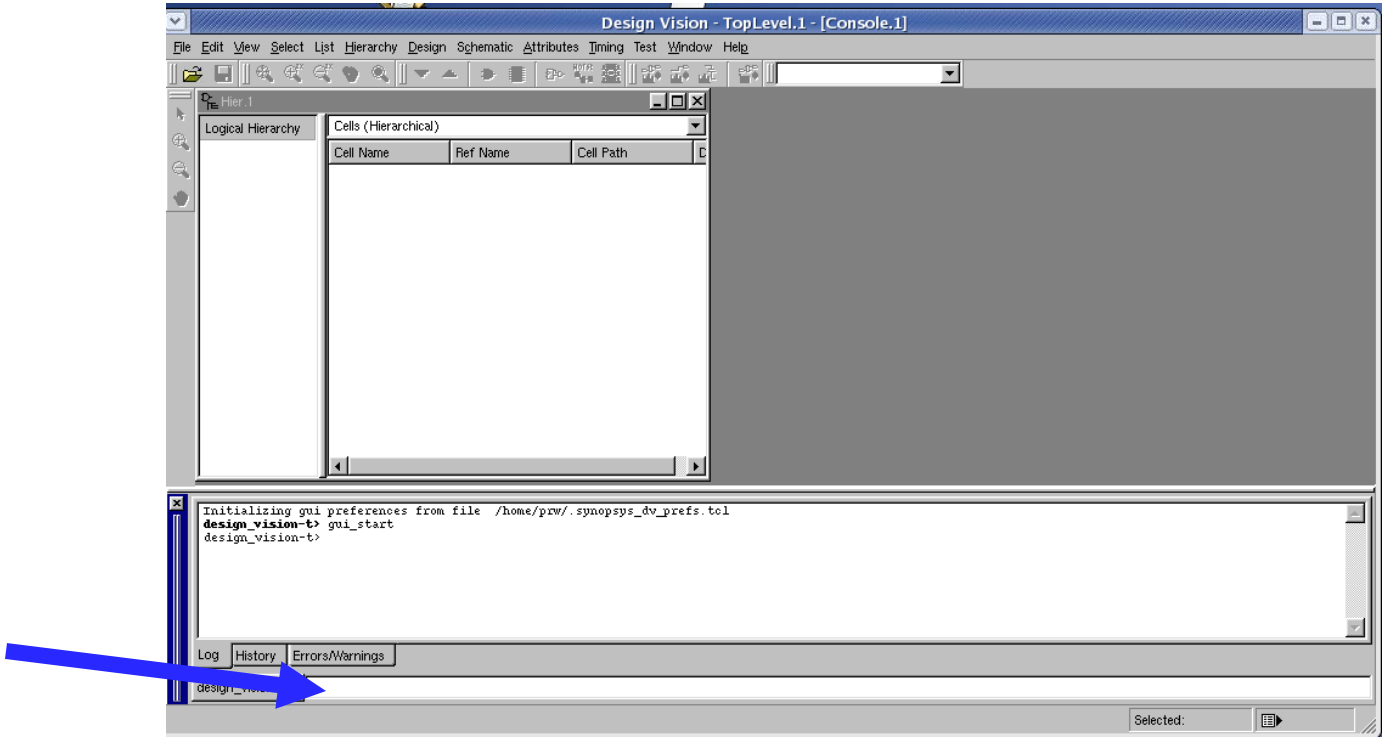
The easy way….

- **Rather than type all these commands in each time, they can be stored in a simple script text file and run with the appropriate design name in each case.**

- **Simply click on the non-graphical window with the "dc" prompt and type in:**
  - source *scriptname*.tcl

Using Design Vision

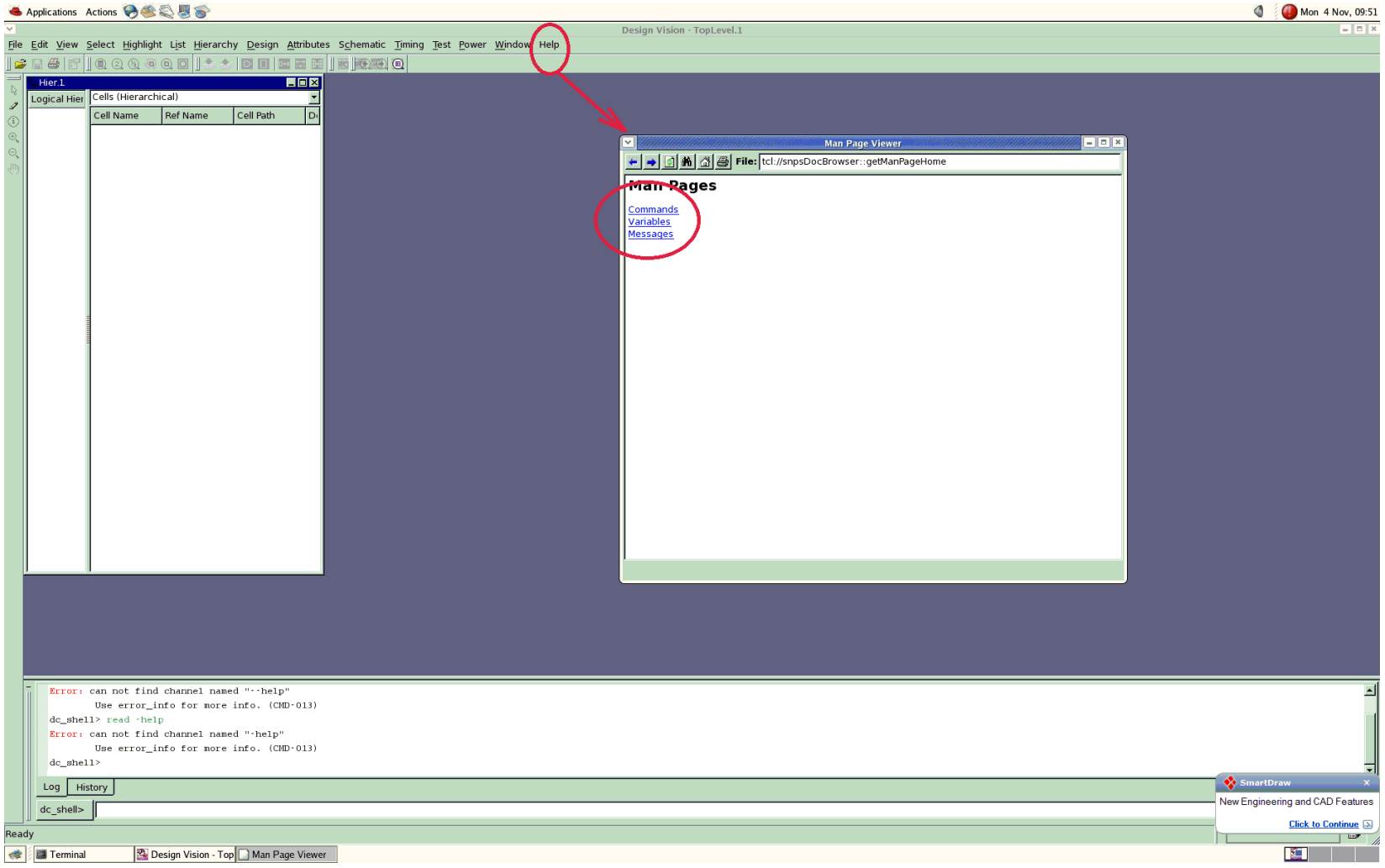
- **Using the GUI you can also simply load the script described previously into the command line of the GUI**
  - source script.tcl



Simple Script

```
analyze -format verilog  {./qmults.v}

elaborate qmults -architecture verilog -library DEFAULT

create_clock i_clk -name i_clk  -period 6
compile
 report_area > synth_area.rpt
report_power > synth_power.rpt
change_names -rules verilog -hierarchy -verbose

write -f verilog -hierarchy -output “qmults_syn.v"

write_sdc design.sdc
write_sdf design.sdf

exit
```

How to get Help:



Discussion Points

1. What is the maximum frequency you can achieve? Can you optimise the design further? Explain how
2. How does optimizing design performance affect its area overhead?
3. How does optimizing design performance affects its energy dissipation?
4. How can you resolve hold time violations?

Appendix 1: Optimization using synthesis tool

There are many ways in which a designer can tweak the design at the synthesis stage to obtain the target performance :

- Compilation with map_effort high option
- Register balancing
- Removing Hierarchy
- Choosing High-Speed Implementation for High-level Functional Module

Compilation with map_effort high option

- Using a map_effore high option during the first synthesis run is not advisable as the run-time for a map_effort high option is significantly longer than that for a map_effort medium.

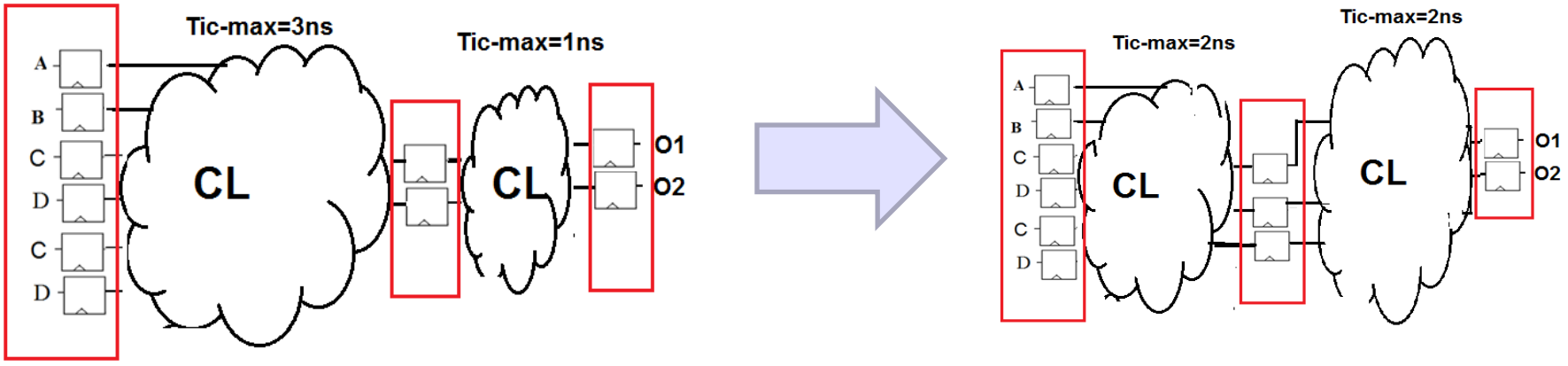
- Generally, during synthesis, it is advisable for the designer to run a quick synthesis on the design using a map_effort medium option when employing design constraints. This would allow the designer to have a feel for the timing violations if any exist.
- Example:

compile -map_effort high

Register Balancing

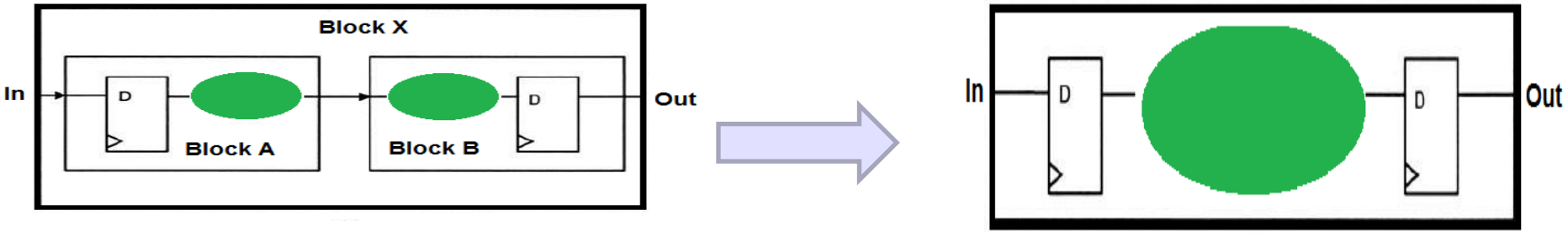
- Register balancing is a very useful command when it comes to optimizing designs that are made up of pipelines.
- The concept here is to allow *Design Compiler* to move logic from one stage of the pipeline to another. This would allow *Design Compiler* the flexibility to move logic away from pipeline stages that are overly constrained to pipeline stages that have additional timing.
- You can balance the register after you compile by typing:

balance_registers



Removing Hierarchy

- The original hierarchy of the design form a logical boundary, which prevents synthesis tools from optimizing across this boundary. By default, DC maintains the original hierarchy of the design.
- Having needless hierarchy in the design limits DC to optimize within that boundary without optimizing across the hierarchy.



Removing Hierarchy

- How remove hierarchy:

Current design = modulename

ungroup -all -flatten

compile -map_effort high -incremental_mapping

- **However**, this option is not suitable for usage if the hierarchical design is large. Too huge a design will take up considerable computing resources (for example, a long time to compile).

Choosing High-Speed Implementation for High-level Functional Module

- The designer can manually change the implementation selection specified synthetic library cell instances by setting the variable set_implementation as follows:

Set_implementation <implementation_type> <cell_list>

- If A1 is an instance of the DW01_ADD cell in the current design, a cla carry-lookahead implementation can be specified to implement A1

set_implementation cla A1

Implementation type	*Description*
rpl	Ripple carry
cla	Carry look ahead
clf	Fast carry look ahead
sim	Simulation model

Choosing High-Speed Implementation for High-level Functional Module

- The set_implementation command does not function on cell instances that are not defined in a synthetic library.
- To list the current implementation of all synthetic library instances, and indicates whether the implementation

report_resources

- To remove any effects of set_implementation from all synthetic cells of the current design:

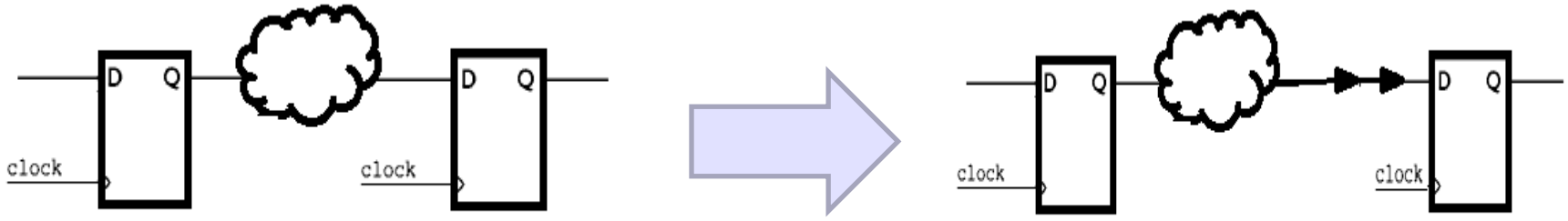
remove_attribute [get_cells *] implementation

How to Fix Hold Time Violation

$$T_{Hold} \leq T_D + T_{IC-min}$$

How Fix Hold Time Violation

- **Any path that has hold time violations can be fixed by adding buffers in that path. These buffers will add delay to the path and ultimately slow it down.**



How Fix Hold Time Violation

- **During synthesis** design, the designer can set the attribute set_fix_hold to have Design Compiler fix the hold violations using the following commands:

set_fix_hold <clock_name>

compile –map_effort high -incremental_mapping

- Example: the following command sets a fix_hold attribute on clock "clk1".

set_fix_hold clk1

- **To remove** the fix_hold attribute from clock "clk1“:

remove_attribute [get_clocks clk1] fix_hold