



Circuits and Systems

Mekelweg 4,
2628 CD Delft
The Netherlands
<http://ens.ewi.tudelft.nl/>

ET4351-2011-01

Tutorial

Digital Design Flow

For EDA tools:

Mentor Modelsim

Synopsys Design Compiler

Cadence SoC Encounter

Ir. A.C. de Graaf
Ing. H.J. Lincklaen-Ariens
Dr. ir. T.G.R van Leuken

Preface

This document describes the top-down design flow of the implementation a SoC design. Starting from a example HDL description the designer is guided through all the design steps to tapeout GDS2 layout description.

This tutorial is derived from "Top-Down digital design flow" version 3.1 (November 2006) by Alain Vachoux, Microelectronic Systems Lab EPFL, Lausanne, Switzerland.

Ir. A.C. de Graaf
Ing. H.J. Lincklaen-Ariens
Dr. ir. T.G.R van Leuken
Delft, The Netherlands
My Graduation Date

Contents

Preface	iii
1 Introduction	1
1.1 Top-down design flow	1
1.2 Design project organisation	3
1.3 EDA tools and design kit configuration	4
1.4 Installation of the FARADAY design kit	5
1.5 VHDL example: Adder-subtractor	7
1.6 Text editing	10
1.7 Design flow steps	11
2 VHDL and Verilog simulation	13
2.1 Starting the Modelsim graphical environment	13
2.2 Simulation of (pre-synthesis) RTL VHDL models	14
2.3 Simulation of the post-synthesis VHDL model with timing data	17
2.4 Simulation of the post-route Verilog model with timing data	18
3 Logic synthesis	23
3.1 Starting the Design Vision graphical environment	23
3.2 RTL VHDL model analysis	25
3.3 Design elaboration	25
3.4 Design environment definition	26
3.5 Design constraint definitions	27
3.6 Design mapping and optimization	29
3.7 Report generation	31
3.8 VHDL/Verilog gate-level netlist generation and post-synthesis timing data (SDF) extraction	38
3.9 Design constraints generation for placement and routing	39
3.10 Design optimization with tighter constraints	41
3.11 Using scripts	41
4 Standard cell placement and routing	47
4.1 Starting the Encounter graphical environment	47
4.2 Design import	49
4.3 Floorplan Specification	51
4.4 Power ring/stripe creation and routing	52
4.5 Global net connections	55
4.6 Operating conditions definition	56
4.7 Core cell placement	56
4.8 Post-placement timing analysis	58
4.9 Clock tree synthesis (optional)	60
4.10 Design routing	63
4.11 Post-routing timing optimization and analysis	65

4.12 Filler cell placement	66
4.13 Design checks	67
4.14 Report generation	69
4.15 Post-route timing data extraction	72
4.16 Post-route netlist generation	73
4.17 GDS2 file generation	73
4.18 Using scripts	74
5 Appendix A: Design Metrics	85

List of Figures

1.1	Top-down design flow	1
1.2	Design project structure.	4
2.1	Modelsim console window	13

List of Tables

Introduction

This document details the typical steps of a top-down digital VHDL/Verilog design flow with the help of one simple design example.

The following tools are considered in this document:

- Modelsim v6.5e or higher, from Mentor Graphics.
- Design Compiler and Design Vision D-2010.03 or higher from Synopsys.
- Encounter 8.1 or higher from Cadence Design Systems.

The design kit used is from Faraday. The process is the 90 nm 9-metal CMOS called L90.SP. Each of the next chapters in this document is addressing a specific set of tasks. Chapter 2 is about VHDL and Verilog simulation, chapter 3 is about logic synthesis and chapter 4 is about place and route. Steps in these chapters are not necessarily to be done in the given sequence. Go to “1.7 Design flow steps” to get a typical sequence of steps supporting a top-down approach.

1.1 Top-down design flow

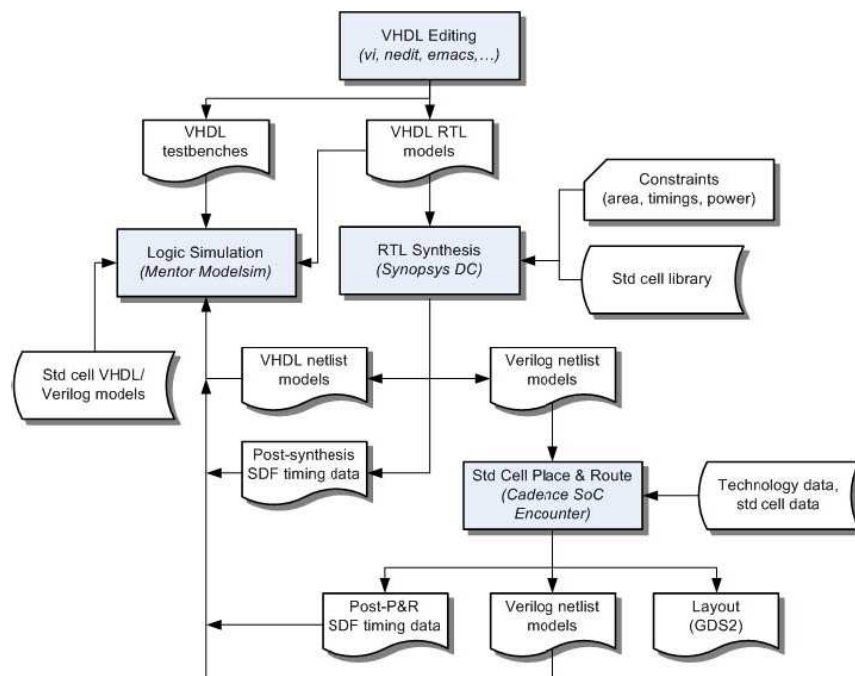


Figure 1.1: Top-down design flow

Figure 1.1 illustrates the top-down flow that includes the following steps:

VHDL RTL model creation

The goal here is to develop synthesizable VHDL models at the RTL level (RTL means Register-Transfer Level). Such models usually define a clear separation between control parts (e.g. finite state machines-FSM) and operative parts (e.g. arithmetic and logic units). Registers are used to store small size data between clock cycles. RAM/ROM memories are used to store large amounts of data or program code. Blocks such as FSMs, ALUs, registers are usually described as behavioral models that do not imply any particular gate-level implementation. Tools used at this step can range from simple text editors to dedicated graphical environments that generate VHDL code automatically.

RTL simulation

The VHDL RTL models are validated through simulation by means of a number of testbenches also written in VHDL.

RTL synthesis

The synthesis process infers a possible gate-level realization of the input RTL description that meets user-defined constraints such as area, timings or power consumption. The design constraints are defined outside the VHDL models by means of tool-specific commands. The targeted logic gates belong to a library that is provided by a foundry or an IP company as part of a so-called design kit. Typical gate libraries include a few hundreds of combinational and sequential logic gates. Each logic function is implemented in several gates to accommodate several fan-out capabilities or drive strengths. The gate library is described in a tool-specific format that defines, for each gate, its function, its area, its timing and power characteristics and its environmental constraints.

The synthesis step generates several outputs: a gate-level VHDL netlist, a Verilog gate-level netlist, and a SDF description. The first netlist is typically used for post-synthesis simulation, while the second netlist is better suited as input to the place&route step. The SDF description includes delay information for simulation. Note that considered delays are at this step correct for the gates but only estimated for the interconnections.

Post-synthesis gate-level simulation

The testbenches used for RTL model validation can be reused (with possibly some modifications to use the VHL gate-level netlists). The gate-level simulation makes use of VHDL models for the logic gates that are provided in the design kit. These VHDL models follow the VITAL modeling standard to ensure proper back-annotation of delays through the SDF files generated by the synthesis or the place&route step.

Standard cell place and route

The place&route (P&R) step infers a geometric realization of the gate-level netlist so-called a layout. The standard cell design style puts logic cells in rows of equal heights. As a consequence, all logic gates in the library have the same height, but may have different widths. Each cell has a power rail at its top and a ground rail at its bottom.

The interconnections between gates are today usually done over the cells since current processes allow several metal layers (i.e. 9 metal layers for the Faraday L90_SP process). As a consequence, the rows may be abutted and flipped so power and ground rails are shared between successive rows.

The P&R step generates several outputs: a geometric description (layout) in GDS2 format, a SDF description and a Verilog gate-level netlist. The SDF description now includes interconnect delay. The Verilog netlist may be different from the one read as input as the P&R step may make further timing optimizations during placement, clock tree generation and routing (e.g. buffer insertion).

Post-layout gate-level simulation

The Verilog gate-level netlist can be simulated by using the existing VHDL testbenches and the more accurate SDF data extracted from the layout.

System-level integration

The layout description is then integrated as a block in the designed system. This step is not covered in this document.

1.2 Design project organisation

First we have to setup a proper work environment. The toolscripts need the CSH shell as command shell while the standard shell after login is the Bash shell. So first we take care that the CSH shell is our default command shell by:

```
1 cp /opt/eds/Designlab/bin/dot.bashrc ~/.bashrc
```

Now close the current terminal window and start a new terminal. Then add the DesignLab script directory to our PATH by:

```
source /opt/eds/Designlab/bin/dlab.csh
```

Given the number of EDA tools and files used in the flow, it is strongly recommended to organize the working environment in a proper way. To that end, the `create_eda_project` script can be used to create a directory structure in which design files will be stored.

The use of the script is as follows:

```
create_eda_project <project-name>
```

where `[project-name]` is the name of the top-level directory that will host all design files for the projects.

For example, to create the project directory called ADDSUB that will be used to do the tasks presented in the rest of this document, execute the following command:

```
student@tango> create_eda_project ADDSUB
student@tango> cd ADDSUB
```

The ADDSUB top-level directory hosts the configuration files for logic simulation (Modelsim), logic synthesis (Synopsys DC) and standard cell place and route (Cadence SoC Encounter). As a consequence, it is required that the tools are always started from that point. One exception is full-custom layout tools (Cadence IC) that must be started from the subdirectory LAY, which hosts different configuration files.

Figure 1.2 gives the proposed directory structure and the role of each subdirectory. The actual use of the subdirectories and files will be explained while going throughout the tutorial in this document.

```

<project-name>          # project directory home
|-- .synopsys_dc.setup   # setup file for Synopsys tools
|-- modelsim.ini         # setup file for Modelsim tool
|-- DOC                  # documentation (pdf, text, etc.)
|-- HDL                  # VHDL/Verilog source files
|   |-- GATE             # gate-level netlists
|   |-- RTL              # RTL descriptions
|   '-- TBENCH           # testbenches
|-- IP                   # external blocks (e.g., memories)
|-- LAY                  # full-custom layout files
|-- LIB                  # design libraries
|   |-- MSIM             # Modelsim library (VHDL, Verilog)
|   '-- SNPS             # Synopsys library (VHDL, Verilog)
|-- PAR                  # place & route files
|   |-- BIN              # commands, scripts
|   |-- CONF             # configuration files
|   |-- CTS              # clock tree synthesis files
|   |-- DB               # database files
|   |-- DEX              # design exchange files
|   |-- LOG              # log files
|   |-- RPT              # report files
|   |-- SDC              # system design constraint files
|   |-- TEC              # technology files
|   '-- TIM              # timing files
|-- SIM                  # simulation files
|   |-- BIN              # commands, scripts
|   '-- OUT              # output files (e.g., waveforms)
'-- SYN                  # synthesis files
    |-- BIN              # commands, scripts
    |-- DB               # database files
    |-- LOG              # log files
    |-- RPT              # report files
    |-- SDC              # system design constraint files
    '-- TIM              # timing files

```

Figure 1.2: Design project structure.

1.3 EDA tools and design kit configuration

In order to use the EDA tools and the design kit, a file called `edatk.conf` must exist in the directory from which the tools are launched (the top-level project directory).

The `edatk.conf` file must include the following lines (order is not important):

```

mgc msim 6.5e
snps syn D-2010.03
cds soc 8.1

```

dk Faraday L90_SP

1.4 Installation of the FARADAY design kit

To install the files required by the FARADAY design kit, execute the `tech_setup` script in the top-level project directory as follows:

```
tech_setup -t <toolset>
```

where: -t select toolset

Running the command without arguments displays a short help.

To install the 90 nm CMOS called L90_SP for logic simulation and synthesis, use:

```
student@tango-ADDSUB> tech_setup -t synopsys_dc
```

The command produces the following output:

IMPORTANT NOTICE

```

    All FARADAY documentation and design files made available with this
    command
4    are subject to non-disclosure agreements between Europractice, FARADAY
    and TU Delft, and hence must be considered as strictly confidential.
    For more information:
        Alexander de Graaf, EEMCS-ME-CAS, a.c.degraaf@tudelft.nl.

9    — Configuration parameters:
    FARADAY Design Kit
    Process: L90_SP
    Tool:     synopsys_dc
    — Configuration log:
14   File "../synopsys_dc.setup" created
    File "../modelsim.ini" created
    Directory "../LIB/SNPS" created
    Synopsys library link "../LIB/SNPS/techmap" created
    Synopsys library link "../LIB/SNPS/alib-52" created
19   Directory "../LIB/MSIM" created
    Modelsim library LIB/MSIM/work created
    Modifying modelsim.ini
    Modelsim library WORK mapped to ../LIB/MSIM/work
```

The command creates configuration files for Modelsim and Synopsys as well as the required design libraries for hosting compiled VHDL/Verilog models and for simulation.

To install the same design kit for standard cell place and route, use:

```
student@tango-ADDSUB> tech_setup -t cadence_soc
```

The command produces the following output:

IMPORTANT NOTICE

```

    All FARADAY documentation and design files made available with this
    command
4    are subject to non-disclosure agreements between Europractice, FARADAY
    and TU Delft, and hence must be considered as strictly confidential.
    For more information:
        Alexander de Graaf, EEMCS-ME-CAS, a.c.degraaf@tudelft.nl.
```

```

9      — Configuration parameters:
      FARADAY Design Kit
      Process: L90_SP
      Tool:     cadence_socce
14     — Configuration log:
      File "PAR/BIN/fillcore.tcl" created
      File "PAR/BIN/fillperi.tcl" created
      File "PAR/CONF/L90_SP_std.conf" created
      File "PAR/CONF/pads.io" created
      File "PAR/DEX/gds2.map" created

```

The command creates files in specific locations in the PAR (place and route) subdirectory.

The tech_setup command also creates a script file with the necessary PATHs to the tools.

Add these PATHs by executing:

```
student@tango-ADDSUB> source edadk.csh
```

For information on the FARADAY design kits, find the FARADAY documentation in `/opt/eds/DesignLab/tech/Faraday/L90_SP/docs`.

1.5 VHDL example: Adder-subtractor

The adder-subtractor example will be used as the reference design throughout the topdown flow. Listing 1.1 gives the VHDL model of a generic N bit adder-subtractor. The model is deliberately quite abstract to show how the synthesiser will be able to infer different adder architectures (ripple carry or carry look-ahead architectures) depending on the given constraints. We will call this model the *Core*.

Listing 1.1: RTL synthesisable model of a N bit adder-subtractor.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
4
entity addsub is
    generic (NBITS: natural := 4);
    port (
        clk, rst, add: in std_logic;
9        a, b: in unsigned(NBITS-1 downto 0);
        z: out unsigned(NBITS-1 downto 0));
    end entity addsub;
    architecture dfl of addsub is
        signal a_reg, b_reg, z_reg: unsigned(NBITS-1 downto 0);
14    begin
        process (rst, clk)
        begin
            if rst = '1' then
                a_reg <= (others => '0');
19                b_reg <= (others => '0');
                z <= (others => '0');
            elsif clk'event and clk = '1' then
                a_reg <= a;
                b_reg <= b;
24                z <= z_reg;
            end if;
        end process;

        z_reg <= a_reg + b_reg when add = '1' else
29        a_reg - b_reg;
    end dfl;
```

However, to design a real chip we have to add IO and PAD cells around the *Core* to create a Die design. For this purpose we will make use of technology independent generic components for IO and Memory cells defined in the library **techmap**.

Listing 1.2 gives the code for the toplevel cell **addsub_top**.

Listing 1.2: RTL top model of a N bit adder-subtractor.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
4
library techmap;
use techmap.gencomp.all;
use techmap.fod0a_b25_t25_generic_io_dummy_pkg.all;

9 entity addsub_top is
    generic (NBITS: natural := 8;
            padtech : integer := faraday );
    port (
14     clk, rst, add: in std_logic;
        a, b: in std_logic_vector(NBITS-1 downto 0);
        z: out std_logic_vector(NBITS-1 downto 0));
    end entity addsub_top;

architecture farl90_sp of addsub_top is
19
    component addsub is
        generic (NBITS: natural := 4);
        port (
24         clk, rst, add: in std_logic;
            a, b: in unsigned(NBITS-1 downto 0);
            z: out unsigned(NBITS-1 downto 0));
        end component addsub;

    signal clk_s, rst_s, add_s: std_logic;
29    signal a_s, b_s: std_logic_vector(NBITS-1 downto 0);
    signal z_s: unsigned(NBITS-1 downto 0);

begin
    -----
34    -- IO + Core P/G
    -----
    io_pg: entity work.addsub_top_pg(farl90_sp);
    -----
    -- IO Signal Cells
    -----
39
    io_clk : inpad generic map (tech => padtech, limit => core_limited)
        port map (clk, clk_s);
    io_rst : inpad generic map (tech => padtech, limit => core_limited)
        port map (rst, rst_s);
44    io_add : inpad generic map (tech => padtech, limit => core_limited)
        port map (add, add_s);

```

```

io_a : inpadv generic map (width => NBITS, tech => padtech, limit =>
    core_limited)
    port map (a, a_s);
io_b : inpadv generic map (width => NBITS, tech => padtech, limit =>
    core_limited)
49    port map (b, b_s);
io_z : outpadv generic map (width => NBITS, tech => padtech, limit =>
    core_limited)
    port map (z, std_logic_vector(z_s));

-----
54  -- Core Instance
-----

addsub_1: entity work.addsub(dfl)
    generic map (
        NBITS => NBITS
59    )
    port map (
        clk => clk_s,
        rst => rst_s,
        add => add_s,
64    a => unsigned(a_s),
        b => unsigned(b_s),
        z => z_s
    );
end farl90_sp;

```

To install the VHDL model and its associated testbenches in the project directory, enter the following command in the top-level project directory ADDSUB:

```
student@tango-ADDSUB> install_design addsub
```

The following files are copied:

```

HDL/RTL/addsub-dfl.vhd created
HDL/RTL/addsub_top_pg.vhd created
HDL/RTL/addsub_top.vhd created
4 HDL/TBENCH/tb_addsub_top_mapped.vhd created
HDL/TBENCH/tb_addsub_top_par.vhd created
HDL/TBENCH/tb_addsub_top_rtl.vhd created
SYN/BIN/addsub_syn.tcl created
PAR/BIN/addsub_par.tcl created
9 PAR/CONF/addsub_nbits8.conf created
PAR/CONF/addsub_nbits8.io created
PAR/CONF/addsub_nbits8.fp created

```

Listing 1.3 gives the testbench for the RTL model with N=8 (file tb_addsub_top_rtl.vhd). It is by far not complete as it only checks one addition and one subtraction. The testbenches for the mapped design (file tb_addsub_top_mapped.vhd) and for the simulation of the placed and routed Verilog netlist (file tb_addsub_top_par.vhd) only differs by the instantiation of the component under test. The addsub_nbits8.io file defines the positions of the IO pins for place and route.

Listing 1.3: VHDL testbench for the RTL model with N=8.

```

library ieee;
use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;

entity tb_addsub_top_rtl is end;

architecture bench of tb_addsub_top_rtl is
8   constant CLK_PER: time := 20 ns;
   constant NBITS : natural := 8;
   signal clk : std_logic := '0';
   signal rst, add: std_logic;
   signal a, b, z : std_logic_vector(NBITS-1 downto 0);
13 begin
    UUT: entity work.addsub_top(far190_sp)
        generic map (NBITS=>NBITS)
        port map (clk, rst, add, a, b, z);

18   clk <= not clk after CLK_PER/2;
   rst <= '0', '1' after CLK_PER/4, '0' after 3*CLK_PER/4;
   add <= '1', '0' after 11*CLK_PER/4;

   stimulus: process
23   begin
       wait for 3*CLK_PER/4;
       a <= std_logic_vector(to_unsigned(31, a'length));
       b <= std_logic_vector(to_unsigned(12, b'length));
       wait;
28   end process stimulus;
end architecture bench;

```

The file addsub_syn.tcl in directory SYN/BIN is a Tcl script that performs synthesis of the VHDL model in batch mode. The file addsub_par.tcl in directory PAR/BIN is a Tcl script that performs the placement and routing of the synthesized Verilog netlist in batch mode.

1.6 Text editing

The edt alias calls the nedit text editor which provides syntax highlighting. It is also possible to use any other convenient editor such as vi, vim, emacs, xemacs, etc. The Modelsim graphical environment has its own text editor with syntax highlighting.

1.7 Design flow steps

Here are the main steps of the top-down design flow with references to the sections in the document that give more details.

Step 1) Pre-synthesis VHDL simulation (tool: Modelsim)

- 2.1 Compilation of the RTL VHDL model and related testbench [2.2]
- 2.2 Simulation of the RTL VHDL model

Step 2) Logic synthesis (tool: Synopsys Design Compiler)

- 3.1 RTL VHDL model analysis [3.2]
- 3.2 Design elaboration (generic synthesis) [3.3]
- 3.3 Design environment definition (operating conditions, wire load model) [3.4]
- 3.4 Design constraint definitions (area, clock, timings) [3.5]
- 3.5 Design mapping and optimization (mapping to gates) [3.6]
- 3.6 Report generation [3.7]
- 3.7 VHDL gate-level netlist generation [3.8]
- 3.8 Post-synthesis timing data (SDF) generation for the VHDL netlist [3.8]
- 3.9 Verilog gate-level netlist generation [3.8]
- 3.10 Design constraints generation for placement and routing [3.9]

Step 3) Post-synthesis VHDL simulation (tool: Modelsim) [2.3]

- 4.1 Compilation of the VHDL/Verilog netlist and related testbench
- 4.2 Simulation of the post-synthesis gate-level netlist with timing data

Step 4) Placement and routing (tool: Cadence Encounter)

- 5.1 Design import (technological data + Verilog netlist) [4.2]
- 5.2 Floorplan specification [4.3]
- 5.3 Power ring/stripe creation and routing [4.4]
- 5.4 Global net connections definition [4.5]
- 5.5 Operating conditions definition [4.6]
- 5.6 Core cell placement [4.7]
- 5.7 Post-placement timing analysis [4.8]
- 5.8 Clock tree synthesis (optional) [4.9]
- 5.9 Design routing [4.10]
- 5.10 Post-route timing optimization and analysis [4.11]
- 5.11 Filler cell placement [4.12]
- 5.12 Design checks [4.13]
- 5.13 Report generation [4.14]

5.14 Post-route timing data extraction [4.15]

5.15 Post-route netlist generation [4.16]

5.16 GDS2 file generation [4.17]

Step 5) Post-layout VHDL/Verilog simulation (tool: Modelsim) [2.4]

6.1 Compilation of the Verilog netlist and related testbench

6.2 Simulation of the post-synthesis or post-PaR gate-level netlist with PaR timing data

VHDL and Verilog simulation

This chapter presents the main steps to perform the logic simulation of VHDL and Verilog models with the Modelsim tool.

2.1 Starting the Modelsim graphical environment

To start the Modelsim environment, enter in the **vsim** command in the Unix shell:

```
student@tango-ADDSUB> vsim &
```

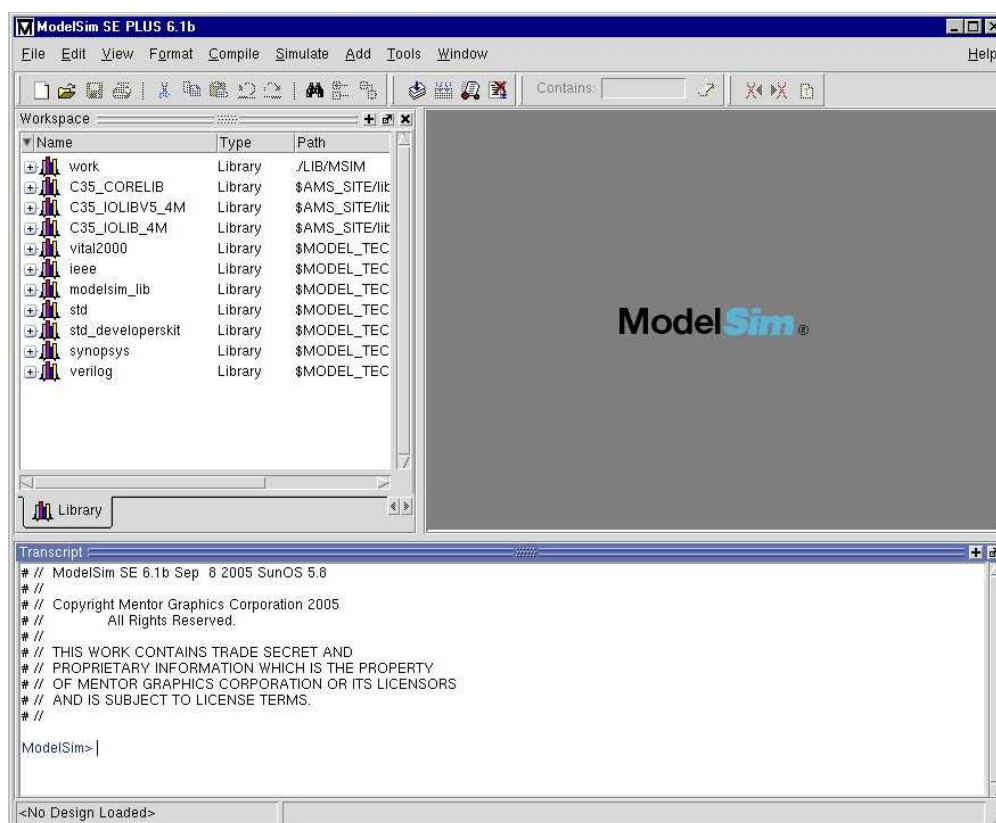



Figure 2.1: Modelsim console window

The **modelsim.ini** file actually defines the mapping between logical design libraries and their physical locations. Note that the Help menu on the top right allows one to access the complete documentation of the tool.

2.2 Simulation of (pre-synthesis) RTL VHDL models

The task here is to validate the functionality of the VHDL model that will be synthesized. The first step is to compile the VHDL model and its associated testbench. There are two ways to compile VHDL models. One way is to execute the `vcom` command from the command line of the Modelsim window:

```
ModelSim> vcom HDL/RTL/addsub_dfl.vhd
ModelSim> vcom HDL/RTL/addsub_top.vhd
ModelSim> vcom HDL/TBENCH/tb_addsub_top_rtl.vhd
```

The other way is to left-click on the **Compile** icon , to select the files to compile in the

HDL/RTL and HDL/TBENCH directories, click on Compile and finally close the window (click Done).

The compiled modules are stored in the logical library WORK which is mapped to the physical location LIB/MSIM. Once VHDL (or Verilog) models have been successfully compiled in the design library, it is possible to create a make file that can be used to recompile only the required files. The `vmake` command can only be run from a Unix shell and creates the make file:

```
student@tango-ADDSUB> vmake > Makefile
```

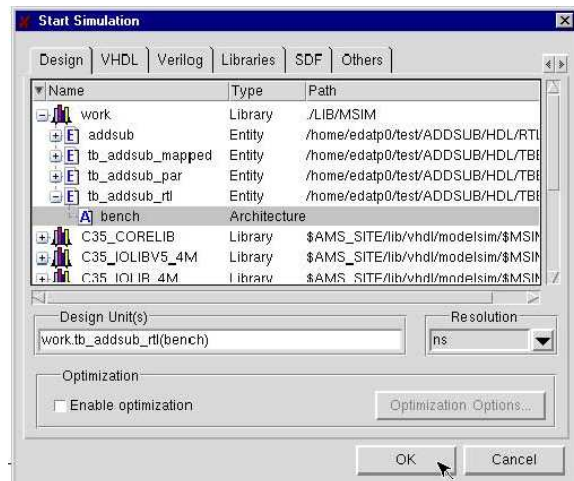
The created file Makefile now defines the design unit dependencies and the compilation commands to recompile only those source files that have been modified or that depend on modified files. To rebuild the library, run the **make** command in the Unix shell



To simulate the RTL model, select the main menu item

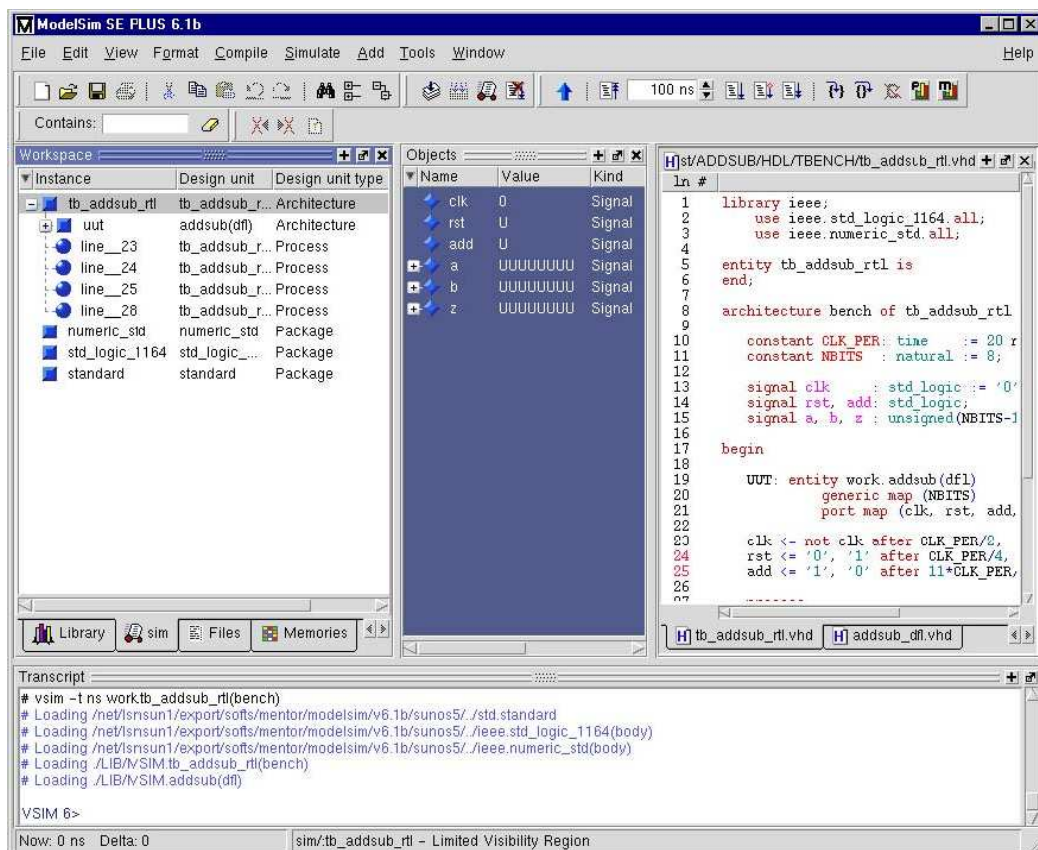
Simulate ⇒ **Start simulation...**

to get the simulation dialog window. Select the architecture of the testbench and a resolution of ns. Then click **OK**.




The main window now changes a bit to show the simulation hierarchy, the list of signals in the testbench and the simulation console (with now the VSIM number; prompt).

Left clicking twice on an instance in the simulation hierarchy pane displays the corresponding VHDL source in the right pane.




The next step is to select the signals to display in simulation. Right click in the Objects (top center) pane, then select **Add to Wave** ⇒ **Signal in Region**.

Note that the appropriate hierarchy level is selected in the simulation hierarchy window. Selecting another level, e.g. uut, will display all the signals visible in this scope. You may want to add selected signals from inner levels (local signals).

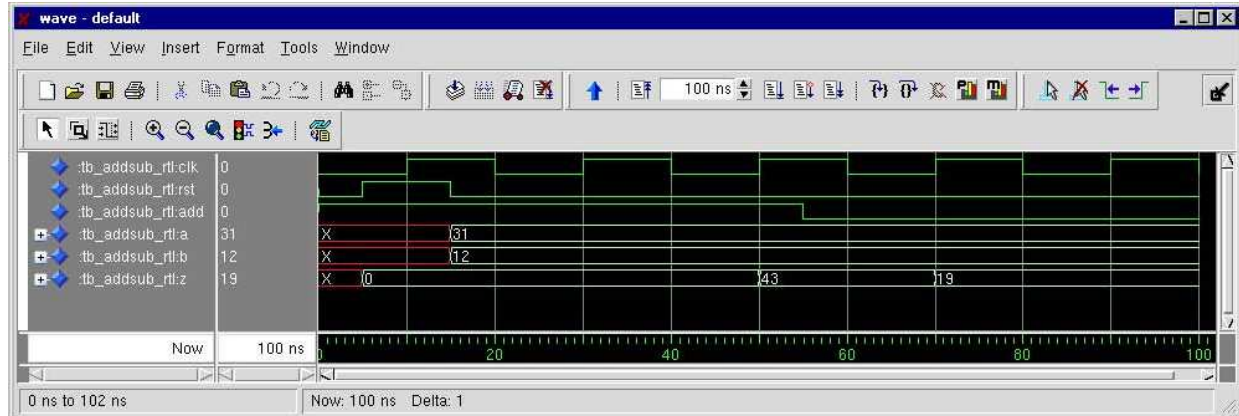
The selected signals are displayed in a new window called wave. The wave pane is by default located on the top right (as a new tab on the source windows). You can click on the Undock icon  to make the wave pane separate. To start the simulation, it is either possible to enter run commands in the simulation console such as:




```
VSIM 7> run 100 ns
```

or to click on the Run icon  in the main window or in the wave window.

The signal waveforms are then visible in the wave window. To change the radix of the displayed signals, select the signals (press shift left-click for multiple selection), then select the wave menu item **Format** ⇒ **Radix** ⇒ **Unsigned**.



Note that the command run -all runs the simulation until there is no more pending event in the simulation queue. This could lead to never ending simulation when the model, like the testbench loaded here, has a continuously switching signal such as the clock signal clk. It is however possible to stop the current simulation by clocking the Break icon  in the main window or in the wave window.

If you make any modification to the VHDL source, you need to recompile the sources (manually or using the vmake command described earlier in this section), and then restart the simulation in the same environment (e.g., the same displayed waveforms or the same simulation breakpoints) with the restart -f command.

2.3 Simulation of the post-synthesis VHDL model with timing data

This step occurs after the RTL model has been synthesized into a gate-level netlist. The timing information about the design is stored in a SDF file. See -3.8 VHDL/Verilog gate-level netlist generation and post-synthesis timing data (SDF) extraction.

Compile the VHDL gate-level netlist generated by the logic synthesis and its testbench:

```
ModelSim> vcom HDL/GATE/addsub_top_nbits8_mapped.vhdl
```

```
...
```

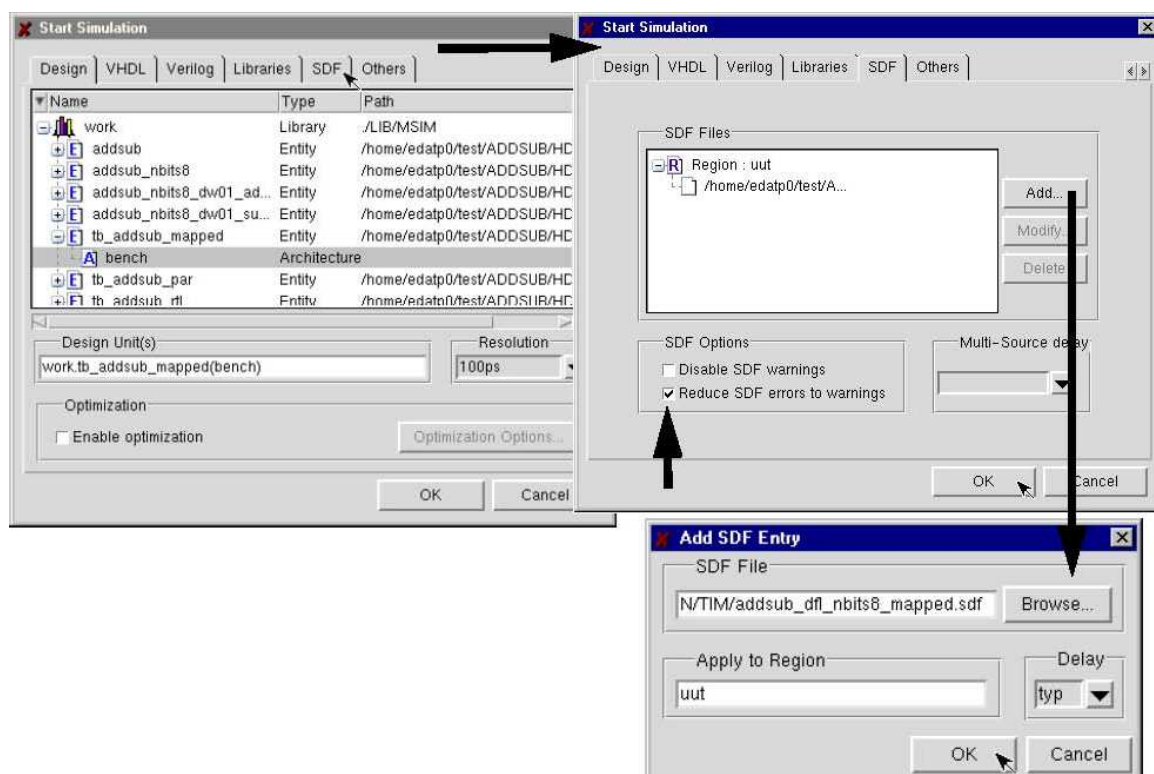
```
ModelSim> vcom HDL/TBENCH/tb_addsub_top_mapped.vhd
```

```
4 ...
```

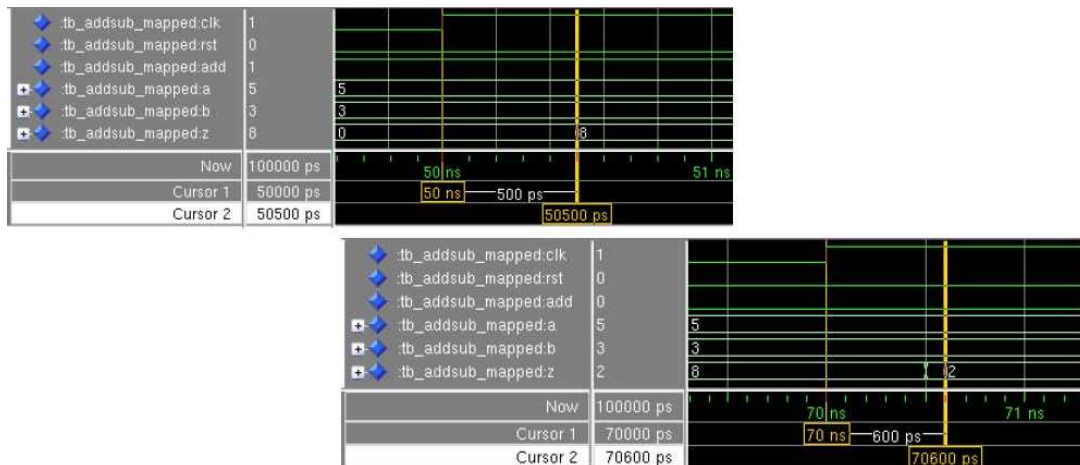
To simulate the RTL model, select the main menu item **Simulate** ⇒ **Start simulation...** to get the simulation dialog window. Select the architecture of the testbench and a resolution of 100ps. Click the **Libraries** tab to add the gate libraries fsd0a_a-generic_core and fod0a.b25_t25-generic_io, then click the **SDF** tab.

In the SDF dialog window, add the file SYN/TIM/addsub_top_nbits8_mapped.sdf and specify the region UUT, which is the label of the instance in the testbench that will be annotated with timing data.

Note that the **Reduce SDF errors to warnings** box must be checked. This is required to avoid the simulation to stop prematurely due to errors such as `Failed to find port 'a(7)'`. These are not really errors here as they are related to interconnect delay data in the SDF file that are not used in the simulation (they are actually all set to zero).



Then click **OK** in the remaining Start Simulation dialog box to load the mapped netlist. Clock to output delays of the order of 500ps to 1ns should be visible in the wave window.



2.4 Simulation of the post-route Verilog model with timing data

This step occurs after the design has been placed and routed. See "4.15 Post-route timing data extraction" and "4.16 Post-route netlist generation". This steps involves the simulation of a Verilog gate-level netlist with a VHDL testbench.

Before compiling the Verilog netlist and the associated VHDL testbench, it is necessary to prepare the design library. The **vdir -r** command gives the content of the design library:

```
1 student@tango-ADDSUB> vdir -r
```

```
Library Vendor : Model Technology
Maximum Unnamed Designs : 3
OPTIMIZED DESIGN _opt
4 OPTIMIZED DESIGN _opt1
OPTIMIZED DESIGN _opt2
ENTITY addsub
  ARCH dfl
ENTITY addsub_nbits8
9  ARCH syn_dfl
ENTITY addsub_top
  ARCH farl90_sp
ENTITY addsub_top_nbits8
  ARCH syn_farl90_sp
14 ENTITY addsub_top_pg
  ARCH farl90_sp
PACKAGE conv_pack_addsub_top_nbits8
...
ENTITY tb_addsub_top_mapped
19  ARCH bench
ENTITY tb_addsub_top_rtl
  ARCH bench
```

The ENTITY and ARCHITECTURE keywords denote VHDL design units. The problem is that the modules in the Verilog netlist have the same names as in the VHDL post-synthesis models (actually they only differ in the case of some name parts). To allow the VHDL testbench HDL/tb.addsub_par.vhd to instantiate a Verilog model, the direct instantiation statement cannot include the architecture name. Therefore, a default mapping is considered

and the design library must only contain the required compiled units. As a consequence, some existing units must be deleted from the library with the **vdel** command:

```
student@tango-ADDSUB> vdel addsub_nbbits8_dw01_addsub_0
```

```
student@tango-ADDSUB> vdel addsub_top_nbbits8
```

Note that this could also be done from within the Modelsim GUI.

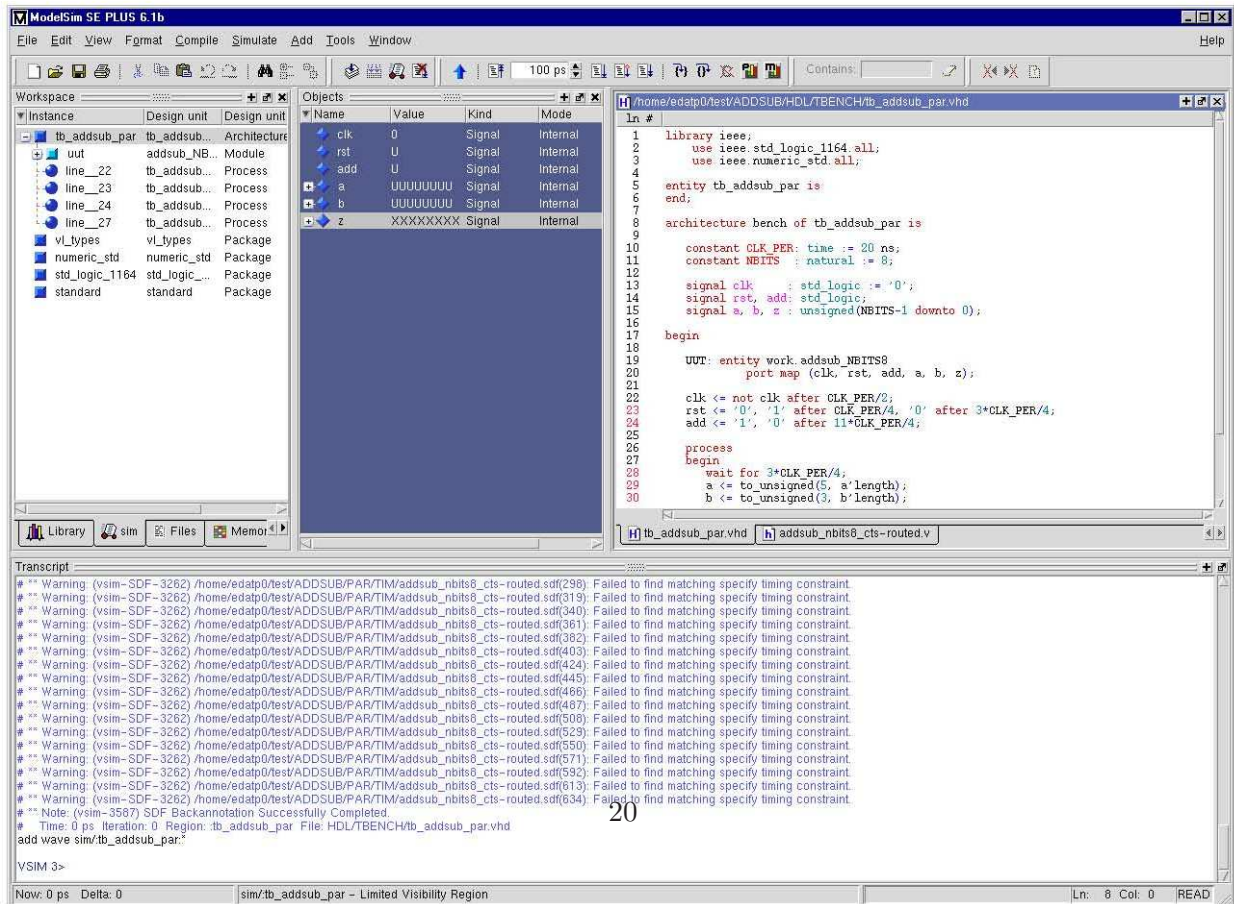
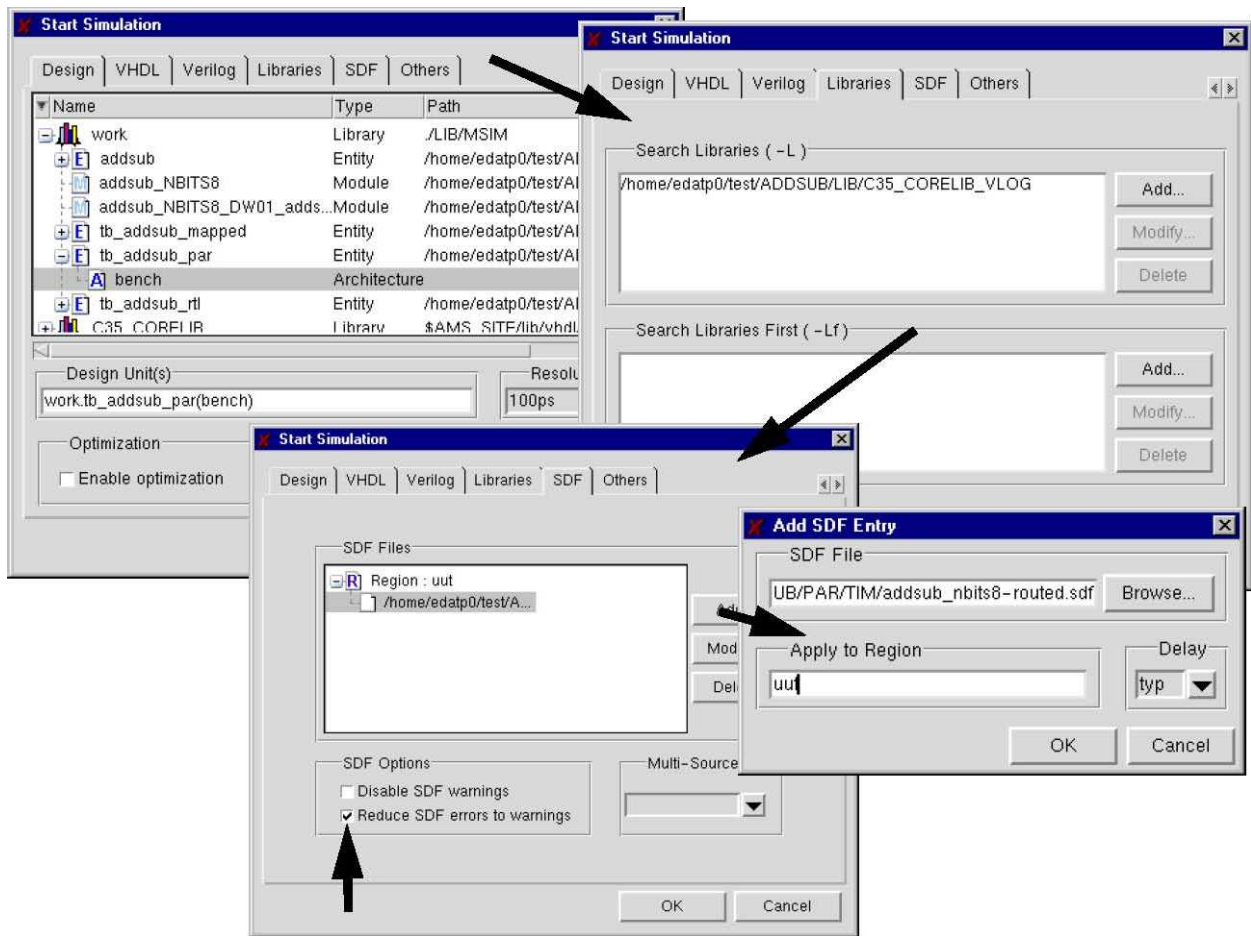
Now you can compile the Verilog netlist and the VHDL testbench:

```
student@tango-ADDSUB> vlog HDL/GATE/addsub_top_nbbits8-routed.v
```

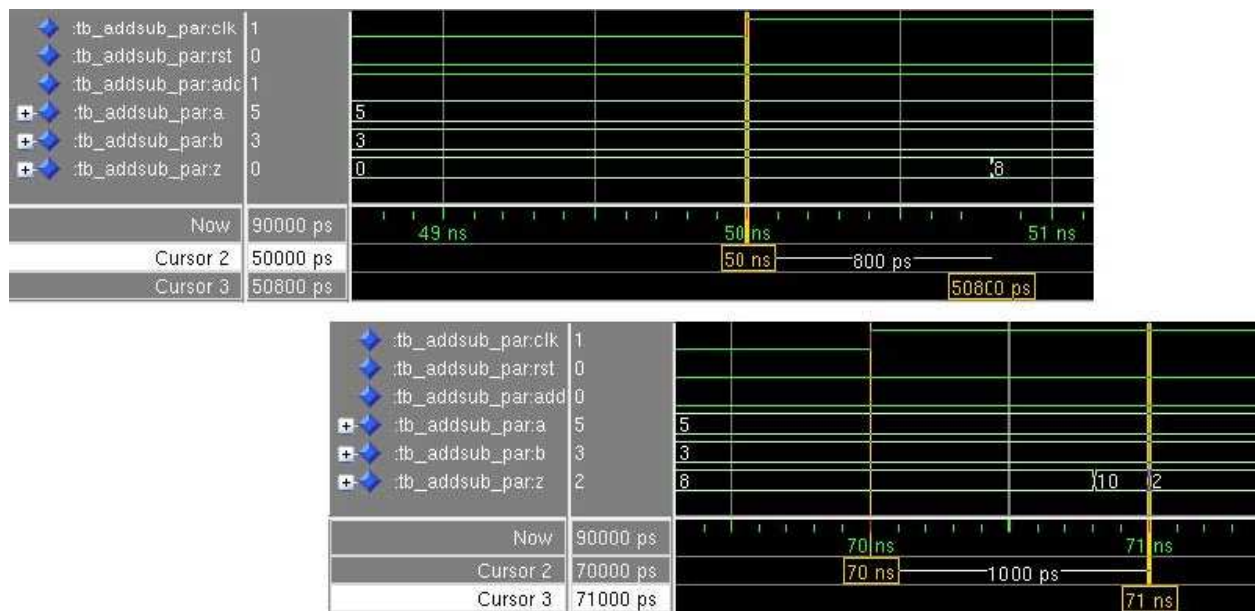
```
student@tango-ADDSUB> vcom HDL/TBENCH/tb_addsub_top_par.vhd
```

To simulate the placed and routed netlist with timing data, select the item **Simulate** ⇒ **Start simulation...** in the main menu to get the simulation dialog window. Select the architecture of the testbench and a resolution of 100ps. Then click the **Libraries** tab to add the gate libraries fsd0a.a_generic_core and fod0a.b25_t25_generic_io, and load the SDF timing file PAR/TIM/addsub_nbbits8-routed.sdf.

Note that the Reduce SDF errors to warnings box must be checked. This is required to avoid the simulation to stop prematurely due to errors such as “Failed to find matching specify timing constraint”. These are not really errors here as they are related to removal (asynchronous) timing constraints generated by Encounter that are not supported in the Verilog models of the gates.



Then click **OK** in the remaining Start Simulation dialog box to load the mapped netlist. Clock to output delays of the order of 800ps to 1000ps should be visible in the wave window.



More accurate values for the delays can be obtained using a smaller time unit in simulation (e.g., 10ps or less).

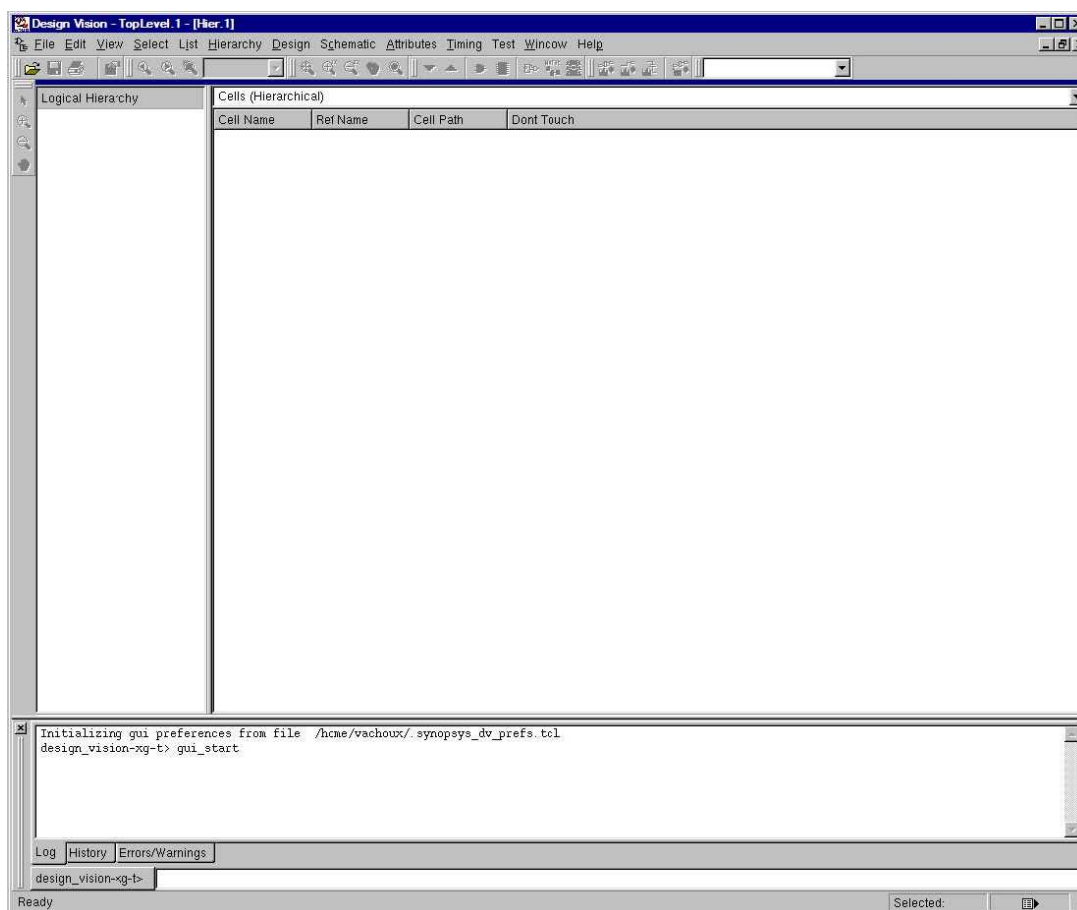
Logic synthesis

This chapter presents the main steps to perform the logic synthesis of the VHDL RTL model with the Synopsys Design Vision and Design Compiler tools. The sold alias displays the complete Synopsys documentation set. Manual pages are available by executing the command “snps man command” (e.g., snps man design_vision).

3.1 Starting the Design Vision graphical environment

To start the Synopsys Design Vision environment, enter the design_vision command in a new shell:

```
student@tango-ADDSUB> design_vision
```



The command line is also echoed in the terminal shell from which the tool has been started, so it is possible to enter DC commands from there as well

(the shell has the `design_vision_` prompt). It is still possible to execute some Unix commands from here.

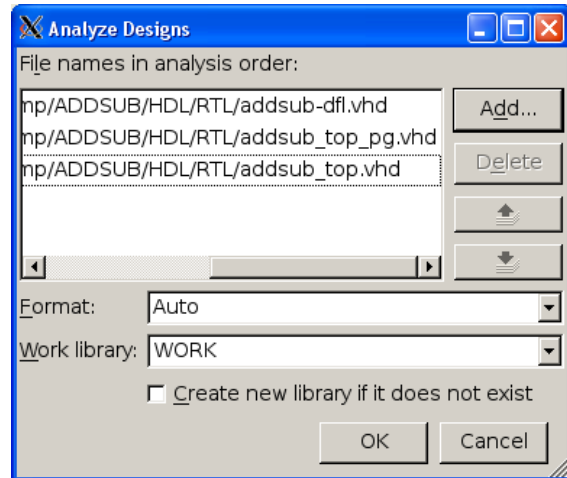
3.2 RTL VHDL model analysis

The analysis phase compiles the VHDL model and checks that the VHDL code is synthesizable. Select **File** ⇒ **Analyze...** in the main menu.

Use the **Add...** button to add all the VHDL sources you need to analyze.

In the case you have more than one VHDL file to analyze, be careful to list the files in the correct analysis order.

Click **OK**.



3.3 Design elaboration

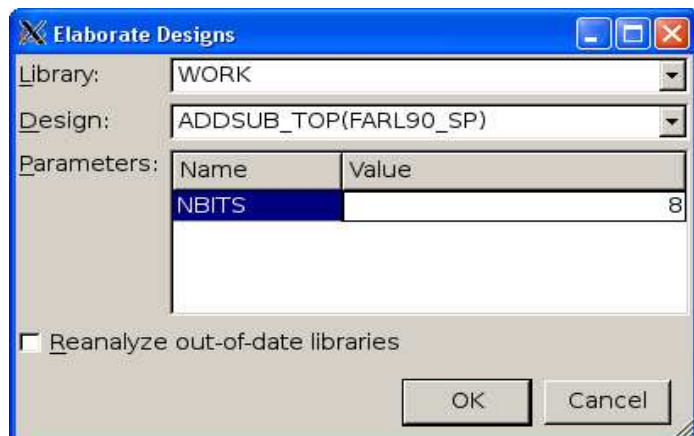
The elaboration phase performs a generic pre-synthesis of the analyzed model. It essentially identifies the registers that will be inferred.

Select **File** ⇒ **Elaborate...** in the main menu.

The DEFAULT library is identical to the WORK library. Specify the value for the NBITS generic parameter to 8.

Click **OK**.

The console now displays the inferred registers and the kind of reset (here asynchronous reset - AR: Y).



```
design_vision-t> elaborate addsub -architecture dfl -library WORK -parameters "NBITS = 8"
```


Inferred memory devices in process
in routine addsub_NBITS8 line 17 in file
'/home/edatp0/test/ADDSUB/HDL/RTL/addsub_dfl.vhd'

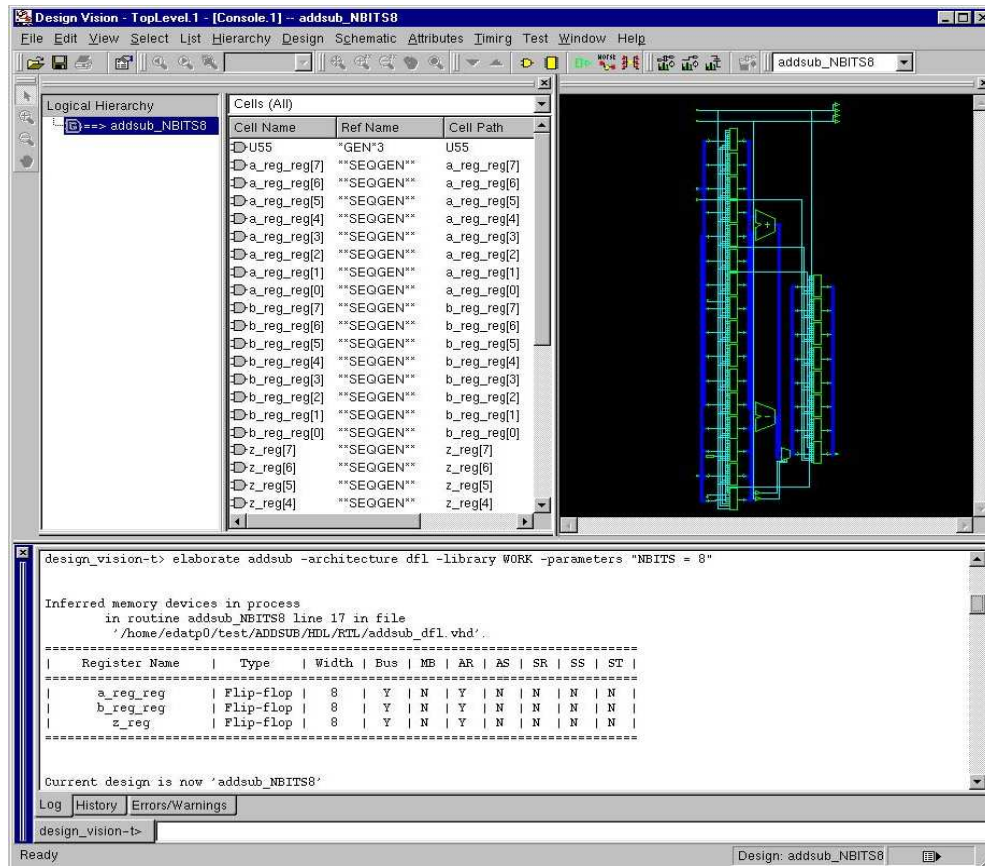
Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
a_reg_reg	Flip-flop	8	Y	N	Y	N	N	N	N
b_reg_reg	Flip-flop	8	Y	N	Y	N	N	N	N
z_reg	Flip-flop	8	Y	N	Y	N	N	N	N

Current design is now 'addsub_NBITS8'

Log History Errors/Warnings

Note the name addsub_top_NBITS8 given to the elaborated entity.

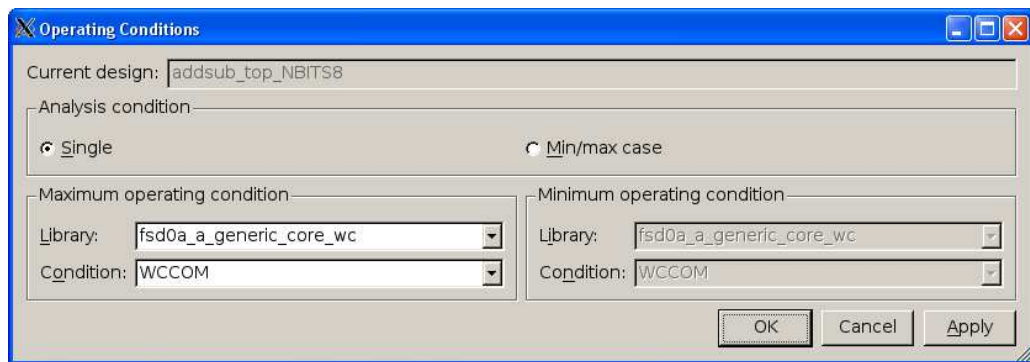
It is possible to display the elaborated schematic by selecting the entity `addsub_NBITS8` in the hierarchy window and then clicking the **Create Design Schematic** icon . Note that the symbols merely indicate generic components that do not yet represent any real logic gate.



3.4 Design environment definition

Before a design can be optimized, you must define the environment in which the design is expected to operate. You define the environment by specifying operating conditions, wire load models, and system interface characteristics. Operating conditions include temperature, voltage, and process variations. Wire load models estimate the effect of wire length on design performance. System interface characteristics include input drives, input and output loads, and fanout loads. The environment model directly affects design synthesis results. Here we will only deal with operating conditions and wire load models.

To define the operating conditions, select the main menu item **Attributes**



⇒ **Operating Environment** ⇒ **Operating Conditions...**

Select the WCCOM condition, which defines a temperature of 85 °C, a voltage of 1V (the L90_SP process is a 1V process), and a slow process. Each cell library defines its own set of operating conditions and may use different names for each set. Click **OK**.

Wire load models allow the tool to estimate the effect of wire length and fanout on the resistance, capacitance, and area of nets. The FARADAY design kit defines a number of wire load models. It also defines an automatic selection of the wire load model to use according to the design area, which is actually considered here.

To get the definitions of the available operating conditions (and on the cell library), execute the report.lib command in the tool command line:

```
report_lib fsd0a_a_generic_core_wc
```

The report_design command summarizes the definitions of the design environment.

3.5 Design constraint definitions

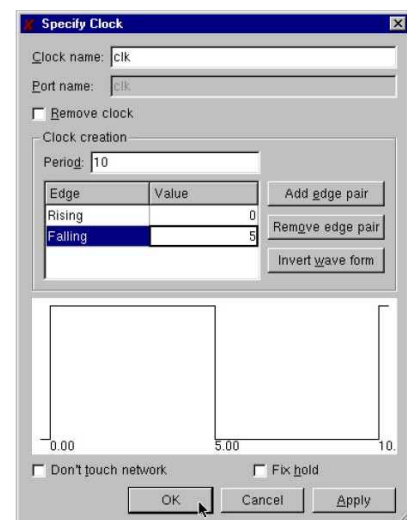
Many kinds of constraints may be defined on the design. Here only constraints on the area and the clock will be defined. To define the clock attributes, i.e. its period and duty cycle, select the entity addsub_NBITS8 in the hierarchy window and then click the Create Symbol

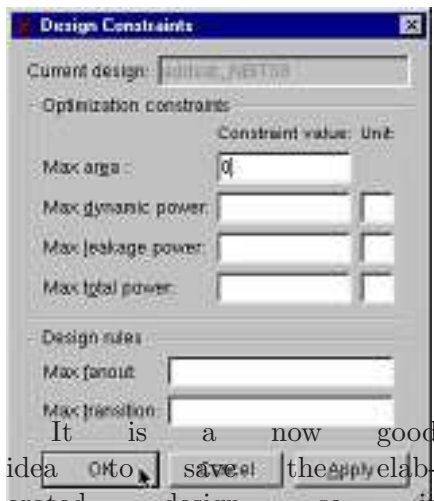
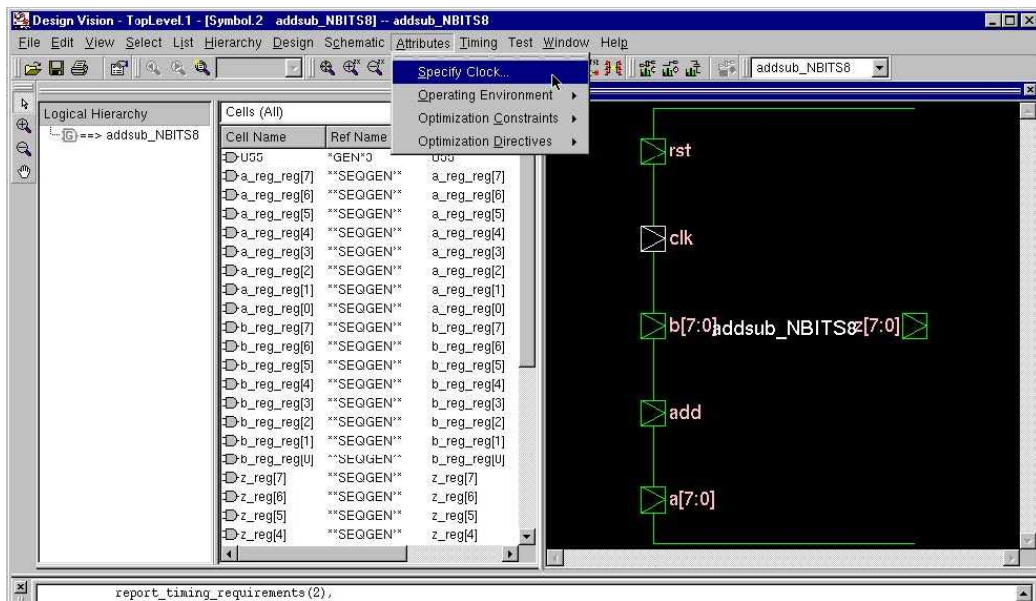
View icon. 

In the symbol view, select the clk pin and then select the main menu item **Attributes** ⇒ **Specify Clock...** Define a clock period of 10 ns with 50% duty cycle. Time unit is not specified here. It is defined in the cell library and is usually ns.

Click **OK**. The console now includes the command line equivalent of the clock definition:

```
create_clock -name "clk" -period 10
-waveform { 0 5 } { clk }
```





It is a now good idea to save the elaborated design so it will be possible to run several optimization steps from that point.

Select the entity `addsub_top_NBITS` in the hierarchy window and then the main menu item **File** \Rightarrow **Save As....**

Save the elaborated design under the name `addsub_top_nbts8_elab.ddc` in the SYN/DB directory.

The selection of the option *Save all design in hierarchy* is relevant for hierarchical designs.

The console includes the equivalent command line:

write-hierarchy -format ddc -output ../ADDSUB/SYN/DB/addsub_top_nbts8_elab.ddc

To read back an elaborated design, select the main menu item **File** \Rightarrow **Read...** and then select the file to read.

To define an area constraint, select in the main menu the item

Attributes \Rightarrow Optimization Constraints \Rightarrow Design Constraints....

A max area constraint set to zero is not realistic but it will force the synthesizer to target a minimum area.

Click **OK**. The console now includes the command line equivalent of the constraint definition:

`set_max_area 0`



3.6 Design mapping and optimization

The optimization phase, also called here compilation phase, is technology dependent. It performs the assignment of logic gates from the standard cell library to the elaborated design in such a way the defined constraints are met. Select the main menu item

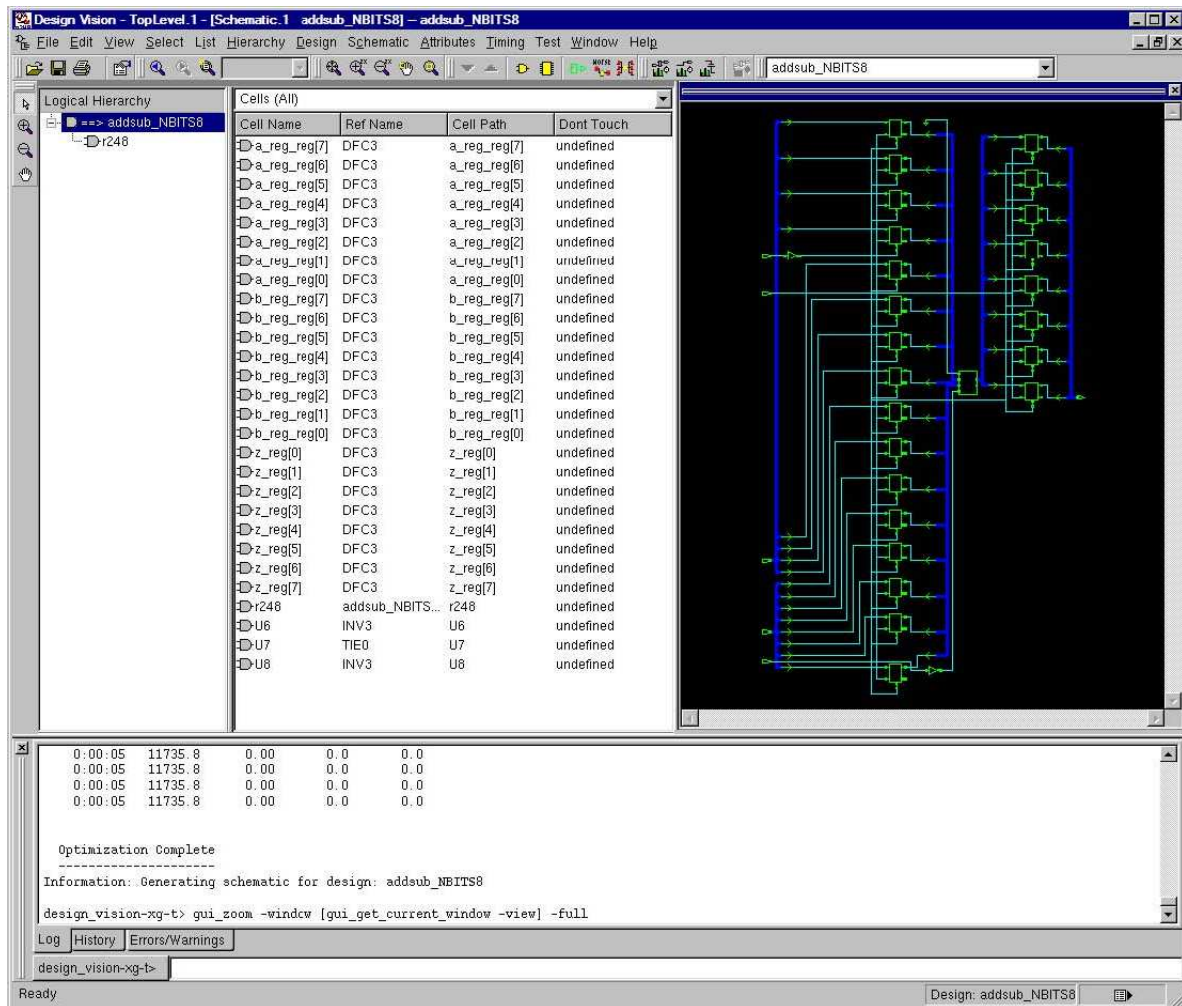
Design ⇒ Compile Ultra....

For a first run there is no need to change the default settings.

Click **OK**. The console and the Unix shell now include the progress of the work.



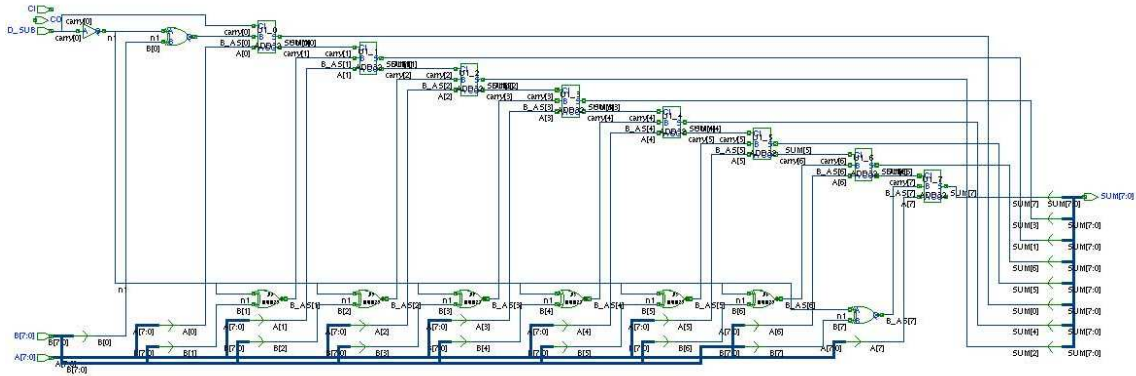
The equivalent command line is: **compile_ultra -map_effort medium -area_effort medium**



The mapped design schematic is now hierarchical as it includes one instance of a 8 bit adder and one instance of a 8 bit subtractor. Also, the cells are now real gates from the cell library.

The inferred 8 bit adder has a ripple-carry architecture and uses the available FA1X standard cell that implements a 1 bit full adder.

The inferred 8 bit subtractor also uses a ripple-carry architecture.



Note that the default resource allocation and implementation for operative parts is based on timing constraints.

This means that resource sharing is used so that timing constraints are met or not worsened. In our case, a single adder has been inferred for both adder and subtractor operations.

The mapped design can now be saved. Select the entity `addsub_top_NBITS8` in the hierarchy window and then the main menu item **File** ⇒ **Save As....** Save the mapped design under the name `addsub_top_nbits8_mapped.ddc` in the directory `SYN/DB`.

3.7 Report generation

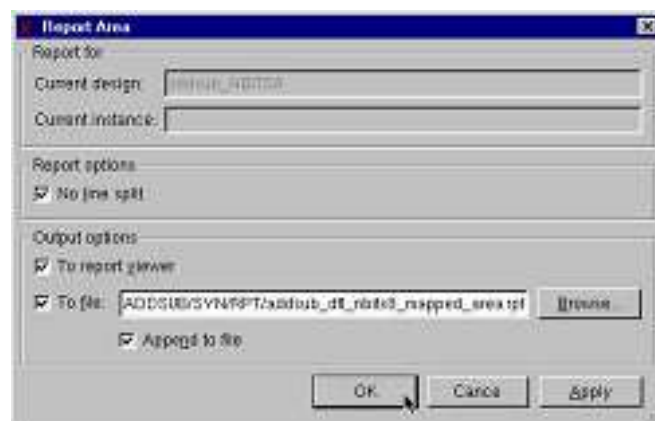
It is possible to get many reports on various synthesis results. Here only reports on the area used, critical path timing and the resources used will be generated.

To get a report of the area used by the mapped design, select the main menu item **Design** ; **Report Area....**

Save the report in the file

`SYN/RPT/addsub_top_nbits8_mapped_area.rpt` as well as in the report viewer. Click **OK**.

A new window and the console now display the report:



Report : area

4 Design : addsub_top_NBITS8

```
Version: B-2008.09-SP1-1
Date   : Tue Apr 28 08:57:15 2009
*****
```

9 Library(s) Used:

```
fsd0a_a_generic_core_wc (File: /opt/eds/DesignKits/Faraday/L90_SP/Core/
fsd0a_a/2007Q1v1.7/GENERIC_CORE_1D0V/FrontEnd/synopsys/
fsd0a_a_generic_core_wc.db)
fod0a_b25_t25_generic_io_ss0p9v125c (File: /opt/eds/DesignKits/Faraday/
L90_SP/IO/fod0a_b25/2007Q3v1.3/T25_GENERIC_IO/FrontEnd/synopsys/
fod0a_b25_t25_generic_io_ss0p9v125c.db)
```

```
14 Number of ports:          27
    Number of nets:          54
    Number of cells:         7
    Number of references:     7

19 Combinational area:      209763.000000
    Noncombinational area:   88560.000000
    Net Interconnect area:   undefined (Wire load has zero net area)

    Total cell area:         298323.000000
24 Total area:              undefined
```

The area unit depends on the standard cell library. Here all area figures are in square microns. The net interconnect area is estimated with the use of a wire load model that has been automatically selected from the design area.

To get a report on the most critical timing path in the mapped design, select the main menu item **Timing ⇒ Report Timing Path**

Save the report in the file

SYN/RPT/addsub_top_nbits8_mapped_timing.rpt as well as in the report viewer.

Click **OK**. A new window and the console now display the report:

```
1 *****
Report : timing
      -path full
      -delay max
6      -max_paths 1
      -sort_by group
Design : addsub_top_NBITS8
Version: B-2008.09-SP1-1
Date   : Tue Apr 28 09:07:40 2009
11 *****

Operating Conditions: WCCOM   Library: fsd0a_a_generic_core_wc
Wire Load Model Mode: enclosed

16 Startpoint: add (input port)
    Endpoint: addsub_1/z_reg[7]
           (rising edge-triggered flip-flop clocked by clk)
    Path Group: clk
    Path Type: max
21
    Des/Clust/Port      Wire Load Model      Library
```

Report Timing Paths

From: pin Through: pin To: pin

Report options:

Worst paths per aspoint: 1 Maximum path delay: Delay type: max Path type: full Sign by: group Significant digits: 2

True path reporting:

☐ Report timing loops
☐ Justify paths with input vector
☐ Find true path

☒ No fan split
☐ Show nets in combinational path
☐ Show input pins in combinational path
☐ Show dont_touch, sigs_only attributes for nets and cells

☐ Enable asynchronous decs
☐ Show net transition time
☐ Show net capacitance

Output options:

☒ To report viewer
☒ To file: c:\projects\ADCSUB\SynAPPT\adcsu8_all_etc8_mapped_timing Browse
☐ Append to file

OK Cancel Apply

```

addsub_top_NBITS8    enG100K    fsd0a_a_generic_core_wc
addsub_NBITS8        enG5K      fsd0a_a_generic_core_wc

```

26

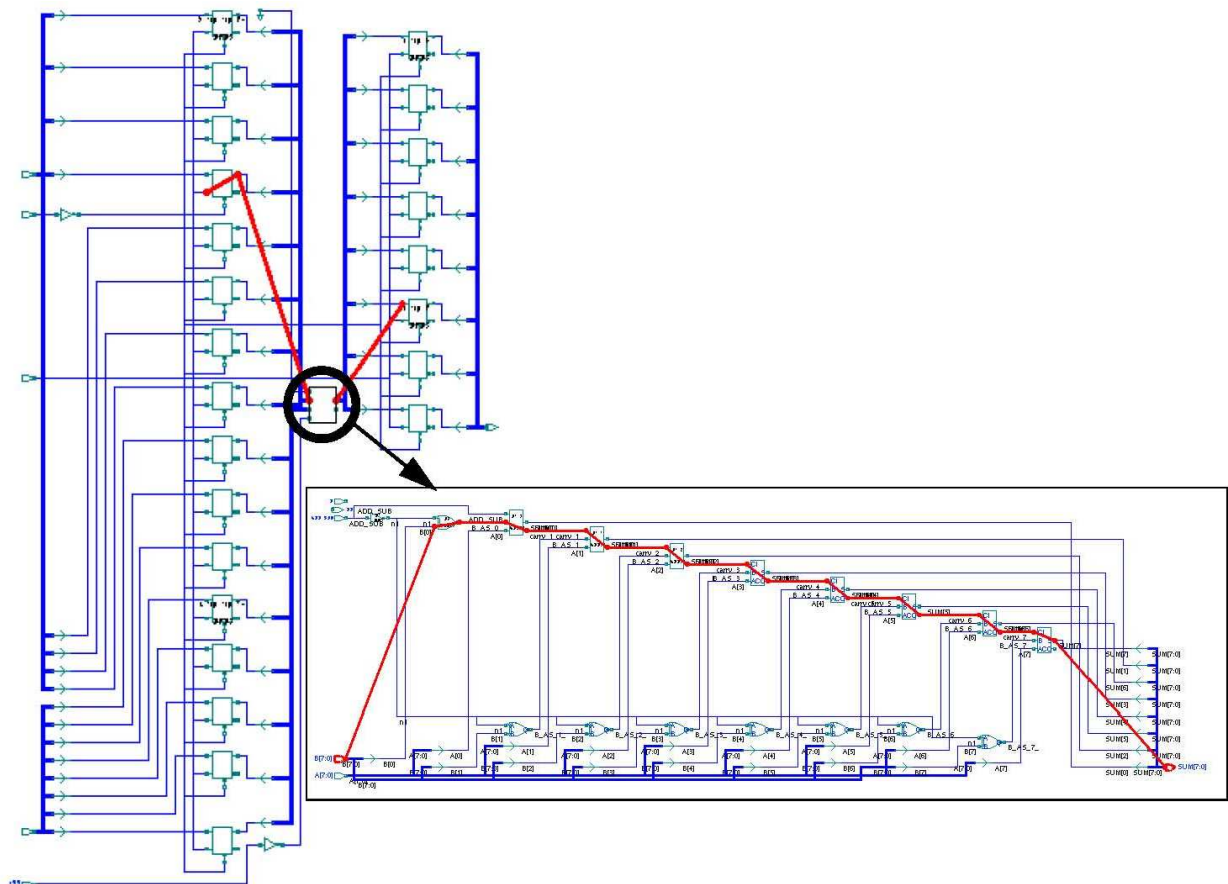
Point	Incr	Path
clock (input port clock) (rise edge)	0.00	0.00
input external delay	0.00	0.00 r
31 add (in)	0.00	0.00 r
io_add/pad (inpad_25_0_3_0_8_1_17)	0.00	0.00 r
io_add/x0/ip/0 (UYNGB)	0.64	0.64 r
io_add/o (inpad_25_0_3_0_8_1_17)	0.00	0.64 r
addsub_1/add (addsub_NBITS8)	0.00	0.64 r
36 addsub_1/U35/0 (INVX1)	0.12	0.76 f
addsub_1/U33/0 (MUX2X1)	0.18	0.93 f
addsub_1/U26/0 (MA0222X1)	0.16	1.09 f
addsub_1/U19/0 (MA0222X1)	0.16	1.26 f
addsub_1/U12/0 (MA0222X1)	0.16	1.42 f
41 addsub_1/U5/0 (MA0222X1)	0.15	1.57 f
addsub_1/DP_OP_11_296_6182/U4/CO (FA1X1)	0.14	1.70 f
addsub_1/DP_OP_11_296_6182/U3/CO (FA1X1)	0.12	1.82 f
addsub_1/DP_OP_11_296_6182/U1/0 (XOR2X1)	0.10	1.92 f
addsub_1/z_reg[7]/D (QDFFRBX1)	0.00	1.92 f
46 data arrival time		1.92
clock clk (rise edge)	10.00	10.00
clock network delay (ideal)	0.00	10.00
addsub_1/z_reg[7]/CK (QDFFRBX1)	0.00	10.00 r
51 library setup time	-0.14	9.86
data required time		9.86

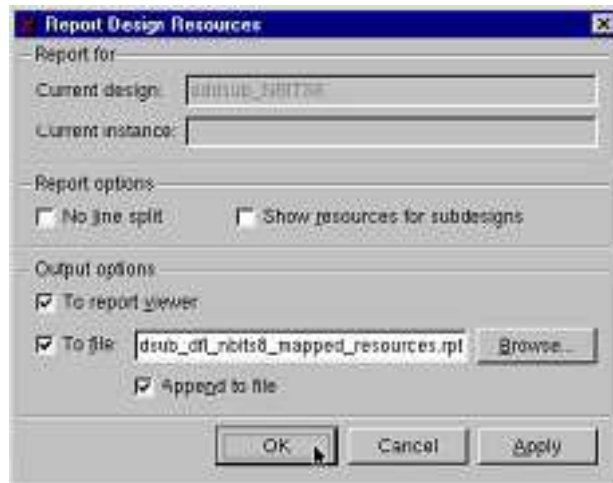
	data required time	9.86
	data arrival time	-1.92
56	<hr/>	
	slack (MET)	7.94

All times are expressed in ns (the time unit is defined in the cell library). The slack defines the time margin from the clock period. A positive slack means that the latest arriving signal in the path still arrives before the end of the clock period. A negative slack means that the timing constraint imposed by the clock is violated.

The timing delays that are accounted for are the internal gate delays (from the cell library) and the estimated interconnect delays (from the cell library and the wire load model in use).

To highlight the critical path on the schematic, select the `addsub_top_NBITS8` entity in the hierarchy window and then the **Select** menu item **Paths From/Through/To...**. You can see that the path goes through the `addsub_1` component so you can also select the component and do the same to visualize the critical path inside the component.





Another useful report is the list of resources used. A resource is an arithmetic or comparison operator read in as part of an HDL design. Resources can be shared during execution of the compile command.

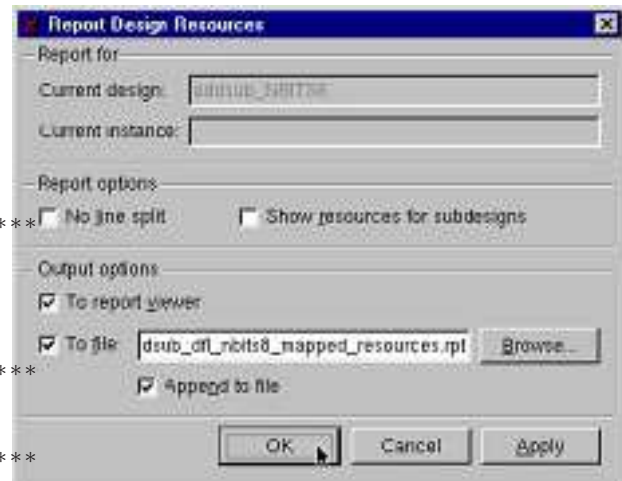
To get a report on the resources used, select the main menu item **Design** \Rightarrow **Report Design Resources...**

Save the report in the file

SYN/RPT/addsub_top_nbits8_mapped_resources.rpt as well as in the report viewer.

Click **OK**.

A new window and the console now display the report:



```

1 *****
Report : resources
Design : addsub_top_NBITS8
Version: B-2008.09-SP1-1
6 Date   : Tue Apr 28 09:40:06 2009
*****

*****
11 Design : addsub_NBITS8
*****

Resource Report for this hierarchy in file
    /mnt/tango/md2/users/sander/tmp/ADDSUB/HDL/RTL/addsub-df1.vhd
16
+-----+-----+-----+-----+
| Cell           | Module           | Parameters | Contained Operations |
+-----+-----+-----+-----+
| DP_OP_11_296_6182 |                  |            |                       |
|                  | DP_OP_11_296_6182 |            |                       |
21 +-----+-----+-----+-----+

Datapath Report for DP_OP_11_296_6182
+-----+-----+
| Cell           | Contained Operations |
+-----+-----+
26 | DP_OP_11_296_6182 | add_29 sub_29        |
+-----+-----+

+-----+-----+-----+-----+
31 | Var  | Type | Data  | Width | Expression |
+-----+-----+-----+-----+
| I1   | PI   | Signed | 8     |            |
| I2   | PI   | Signed | 8     |            |
36 | I3   | PI   | Unsigned | 1    |            |
| O1   | PO   | Signed | 8     | addsub(I1,I2,I3) |
+-----+-----+-----+-----+

41 No implementations to report

No multiplexors to report

```

You can see that the inferred arithmetic component is implemented as a ripple-carry (rpl) architecture. The tool uses the so-called DesignWare library which contains a number of predefined HDL models of blocks (arithmetic, etc.) with several possible architectures for each block. The best architecture is selected to meet the design constraints.

3.8 VHDL/Verilog gate-level netlist generation and post-synthesis timing data (SDF) extraction

This step generates a VHDL model of the mapped design for simulation and a Verilog model of the same design to be used as input to the placement and routing tool. It also generates a SDF (Standard Delay Format) file that includes the gate delays. Care should be taken to use the right naming scheme when generating the SDF file, otherwise the back-annotation of the delays onto the VHDL or Verilog netlists for simulation will fail. Here we only consider the back-annotation of VHDL netlists.

Before generating the VHDL netlist, it is required to apply some VHDL naming rules to the design. This is done by entering the following command in the console (be sure that the entity `addsub_top_NBITS8` is selected in the hierarchy window):

```
change_names -hierarchy -rules vhdl -verbose
```

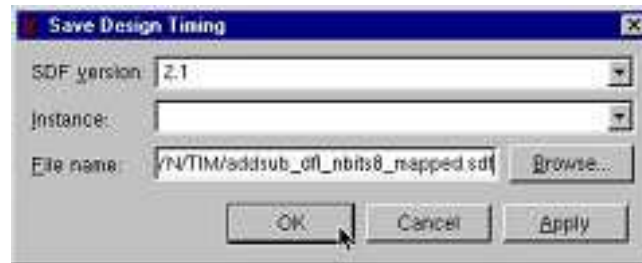


Save the mapped design in the file

`addsub_top_nbits8_mapped.vhdl` in the directory `HDL/GATE`.

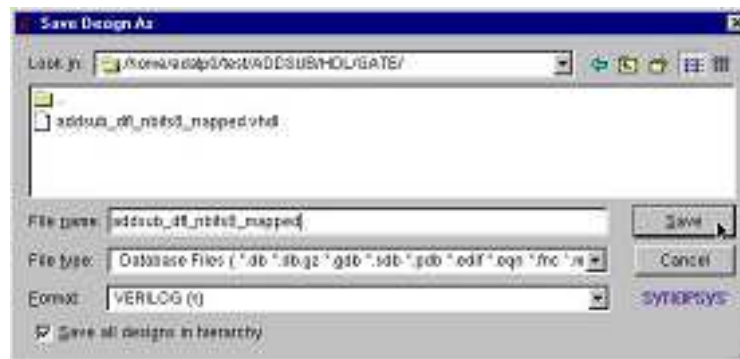
Note: the dialog window creates HDL files with the `.vhdl` extension rather than `.vhd` as used so far.

Click **Save**. The console now echoes the equivalent command line:
`write -hierarchy -format vhdl -output ../ADDSUB/HDL/-
GATE/addsub_top_nbits8_mapped.vhdl`



To generate the SDF file, enter the following command in the console
`write_sdf -version 3 SYN/TIM/addsub_top_nbits8_mapped.sdf`
Information: Annotated 'cell' delays are assumed to include load delay.
The informational message says that the estimated interconnect delays are actually included in the SDF file as part of the cell delays. The generated SDF file actually includes a list of interconnect delays of zero values.

Before generating the Verilog netlist, it is better to reload the database and apply specific Verilog naming rules to the design. This is done by selecting **File** ⇒ **Remove All Designs** from the main menu, then reading the database file `./SYN/D-B/addsub_top_nbits8_mapped.ddc`, and entering the following command in the console (be sure that the entity `addsub_top_NBITS8` is selected in the hierarchy window):
`change_names -hierarchy -rules verilog -verbose`



Save the mapped design in the file `addsub_top_nbits8_mapped.v` in the directory `HDL/-GATE`.

Click **Save**. The console now echoes the equivalent command line:
`write -hierarchy -format verilog
-output ../ADDSUB/HDL/GATE/addsub_dfl_nbits8_mapped.v`

3.9 Design constraints generation for placement and routing

Both design environment and design constraint definitions may be stored in a format that can be read by other Synopsys tools such as PrimeTime or other EDA tool such as Cadence Soc Encounter.

The following command creates a new file that includes the design constraints that have been defined for synthesis in Tcl format:

```
write_sdc -nosplit SYN/SDC/addsub_top_nbits8_mapped.sdc
```

It is important to do that step after the Verilog naming rules have been applied to the mapped design (see 3.8), otherwise there could be discrepancies on port/signal names between the netlist and the constraint file.

3.10 Design optimization with tighter constraints

It is possible to let the synthesizer infer another faster adder architecture, e.g., a carry look-ahead architecture, by shortening the clock period. The goal here is to redo some steps in this chapter and to compare the results with the ones obtained with the initially slow clock.

1. Read the elaborated design. It is not necessary to re-analyze the VHDL sources.
2. Specify the clock with a 5 ns period.
3. Save the new elaborated entity in the file SYN/DB/addsub_top_nbits8_5ns_elab.db.
4. Map and optimize the design.
5. Save the mapped design in the file SYN/DB/addsub_top_nbits8_5ns_mapped.db.
6. Get the new area, timing and resources reports. Compare with the reports you got for the 10 ns clock period.
7. Generate the VHDL gate-level netlist in
HDL/GATE/addsub_top_nbits8_5ns_mapped.vhdl
and the associated SDF timing data file in
SYN/TIM/addsub_top_nbits8_5ns_mapped.sdf.
8. Do a post-synthesis simulation.
9. Generate the Verilog gate-level netlist in HDL/GATE/addsub_top_nbits8_5ns_mapped.v.
10. Save the design constraints for placement and routing in the file
SYN/SDC/addsub_top_nbits8_5ns_mapped.sdc

3.11 Using scripts

It is much more convenient to use scripts and to run the synthesis tool in batch mode when the design complexity increases. Scripts also conveniently capture the synthesis flow and make it reusable. Synopsys Design Compiler supports the Tcl language for building scripts.

An example of such a script for the synthesis of the adder-subtractor design has been installed in the SYN/BIN directory (see “1.5 VHDL example: Adder-subtractor”). The script must be run from the project top directory and it assumed a directory organization as described in “1.2 Design project organisation”. To run the Tcl script, execute the following command in a Unix shell:

```
student@tango-ADDSUB> dc_shell -f SYN/BIN/addsub_syn.tcl
```

When the script finishes executing, the dc_shell environment is still active so you can enter other dc_shell commands. Enter quit or exit to return to the Unix shell.

The script is given below. It may be modified to define design information and constraints and to control the flow to some extent.

```

#
# Synopsys DC shell script for adder-subtractor.
#
4 # Process: Faraday 90 nm CMOS (L90_SP)
#
#
-----

# It is assumed that a project directory structure has already been
# created using 'create_project' and that this synthesis script is
9 # executed from the project root directory $PROJECT_DIR
#
-----

set PROJECT_DIR [pwd]
#
-----

# Design related information (can be changed)
14 #
-----

set VHDL_ENTITY addsub_top
set VHDL_ARCH farl90_sp
set NBITS 8
set CLK_NAME clk
19 # all time values are in ns
set CLK_PERIOD 10;
set INPUT_DELAY 2;
set OUTPUT_DELAY 2;
set OPERATING_COND WCCOM
24 #
-----

# Flags that drive the script behavior (can be changed)
#
# DB_FORMAT (db | ddc)
# if db, use the old DB format to store design information
29 # if ddc, use the new XG format to store design information (recommended)
# SHARE_RESOURCES (0 | 1)
# if 1, force the tool to share resources as much as possible
# if 0, no resource sharing
# COMPILE_SIMPLE (0 | 1)
34 # if 1, only do a single compile with default arguments
# if 0, do a two-step compilation with ungrouping in between
# OPT (string)
# can be used to have different mapped file names
#
-----

39 set DB_MODE ddc
set SHARE_RESOURCES 1
set COMPILE_SIMPLE 1

```

```

set OPT "" ;# to denote the 10ns clock period case
#
-----

44 # File names
#
-----

set SOURCE_FILE_NAME ${VHDL_ENTITY}
set ROOT_FILE_NAME ${VHDL_ENTITY}_nbits${NBITS}
set VHDL_SOURCE_FILE_NAME ${SOURCE_FILE_NAME}.vhd
49 set ELAB_FILE_NAME ${ROOT_FILE_NAME}_elab
set MAPPED_FILE_NAME ${ROOT_FILE_NAME}${OPT}_mapped
set DB_ELAB_FILE_NAME ${ELAB_FILE_NAME}.${DB_MODE}
set DB_MAPPED_FILE_NAME ${MAPPED_FILE_NAME}.${DB_MODE}
set VHDL_NETLIST_FILE_NAME ${MAPPED_FILE_NAME}.vhd
54 set VLOG_NETLIST_FILE_NAME ${MAPPED_FILE_NAME}.v
set SDF_FILE_NAME ${MAPPED_FILE_NAME}.sdf
set SDC_FILE_NAME ${MAPPED_FILE_NAME}.sdc
set RPT_AREA_FILE_NAME ${MAPPED_FILE_NAME}_area.rpt
set RPT_TIMING_FILE_NAME ${MAPPED_FILE_NAME}_timing.rpt
59 set RPT_RESOURCES_FILE_NAME ${MAPPED_FILE_NAME}_resources.rpt
set RPT_REFERENCES_FILE_NAME ${MAPPED_FILE_NAME}_references.rpt
set RPT_CELLS_FILE_NAME ${MAPPED_FILE_NAME}_cells.rpt
set RPT_POWER_FILE_NAME ${MAPPED_FILE_NAME}_power.rpt
#
-----

64 # Absolute paths
#
-----

set VHDL_SOURCE_DIR ${PROJECT_DIR}/HDL/RTL
set VHDL_SOURCE_FILE ${VHDL_SOURCE_DIR}/${VHDL_SOURCE_FILE_NAME}
set VHDL_NETLIST_FILE ${PROJECT_DIR}/HDL/GATE/${VHDL_NETLIST_FILE_NAME}
69 set VLOG_NETLIST_FILE ${PROJECT_DIR}/HDL/GATE/${VLOG_NETLIST_FILE_NAME}
set DB_ELAB_FILE ${PROJECT_DIR}/SYN/DB/${DB_ELAB_FILE_NAME}
set DB_MAPPED_FILE ${PROJECT_DIR}/SYN/DB/${DB_MAPPED_FILE_NAME}
set SDF_FILE ${PROJECT_DIR}/SYN/TIM/${SDF_FILE_NAME}
set SDC_FILE ${PROJECT_DIR}/SYN/SDC/${SDC_FILE_NAME}
74 set RPT_AREA_FILE ${PROJECT_DIR}/SYN/RPT/${RPT_AREA_FILE_NAME}
set RPT_TIMING_FILE ${PROJECT_DIR}/SYN/RPT/${RPT_TIMING_FILE_NAME}
set RPT_RESOURCES_FILE ${PROJECT_DIR}/SYN/RPT/${RPT_RESOURCES_FILE_NAME}
set RPT_REFERENCES_FILE ${PROJECT_DIR}/SYN/RPT/${RPT_REFERENCES_FILE_NAME}
}
set RPT_CELLS_FILE ${PROJECT_DIR}/SYN/RPT/${RPT_CELLS_FILE_NAME}
79 set RPT_POWER_FILE ${PROJECT_DIR}/SYN/RPT/${RPT_POWER_FILE_NAME}
#
-----

# Analyze RTL source
#
-----

```

```

analyze -format vhdl -lib WORK ${VHDL_SOURCE_DIR}/addsub-dfl.vhd
84 analyze -format vhdl -lib WORK ${VHDL_SOURCE_DIR}/addsub_top-pg.vhd
analyze -format vhdl -lib WORK $VHDL_SOURCE_FILE
current_design ${VHDL_ENTITY}
#
-----

# Elaborate design
89 #
-----

elaborate $VHDL_ENTITY \
  -arch $VHDL_ARCH \
  -lib DEFAULT -update \
  -param [set NBITS $NBITS]
94 #
-----

# Define environment
#
-----

set_dont_touch [list addsub_far_pads addsub_far_pg]
#set_dont_touch [list VCCKGB GNDKGB VCC2IOGB GND2IOGB CORNERGB PAD9M126G]
99 #set_operating_conditions -library fsd0a_a_generic_core_wc
  $OPERATING_COND
#
-----

# Define constraints
#
-----

create_clock -name $CLK_NAME -period $CLK_PERIOD [get_ports $CLK_NAME]
104 set_input_delay $INPUT_DELAY -clock $CLK_NAME [list [all_inputs]]
set_output_delay $OUTPUT_DELAY -clock $CLK_NAME [list [all_outputs]]
set_max_area 0
set_fix_multiple_port_nets -all
#
-----

109 # Set resource allocation and implementation
#
-----

set_resource_implementation use_fastest
if { $SHARE_RESOURCES } {
  set_resource_allocation area_only
114 } else {
  set_resource_allocation none
}
#

```

```

-----
# Save elaborated design and constraints
119 #
-----

write -hierarchy -format $DB_MODE -output $DB_ELAB_FILE
#
-----

# Map design to gates
#
-----

124 if { $COMPILE_SIMPLE } {
    compile_ultra
} else {
    compile_ultra -map_effort medium -area_effort medium
    ungroup -all -flatten
129 compile_ultra -incremental -map_effort high
}
#
-----

# Save mapped design
#
-----

134 write -hierarchy -format $DB_MODE -output $DB_MAPPED_FILE
#
-----

# Generate reports
#
-----

report_area -nosplit -hierarchy > $RPT_AREA_FILE
139 report_power -nosplit -hier -hier_level 2 > $RPT_POWER_FILE
report_timing -path full \
    -delay max \
    -nworst 1 \
    -max_paths 1 \
144 -significant_digits 2 \
    -nosplit \
    -sort_by group \
    > $RPT_TIMING_FILE
report_resources -nosplit -hierarchy > $RPT_RESOURCES_FILE
149 report_reference -nosplit > $RPT_REFERENCES_FILE
report_cell -nosplit > $RPT_CELLS_FILE
#
-----

# Generate VHDL netlist

```

```

#
-----

154 change_names -rule vhdl -hierarchy -verbose
    write -format vhdl -hierarchy -output $VHDL_NETLIST_FILE
#
-----

# Generate SDF data
#
-----

159 write_sdf -version 3 $SDF_FILE
#
-----

# Generate Verilog netlist
#
# The design is reloaded from scratch to avoid potential naming problems
164 # when using the netlist for placement and routing
#
-----

remove_design -all
read_file -format $DB_MODE $DB_MAPPED_FILE
change_names -rule verilog -hierarchy -verbose
169 write -format verilog -hierarchy -output $VLOG_NETLIST_FILE
#
-----

# Save system constraints
#
-----

write_sdc -nosplit $SDC_FILE

```


Standard cell placement and routing

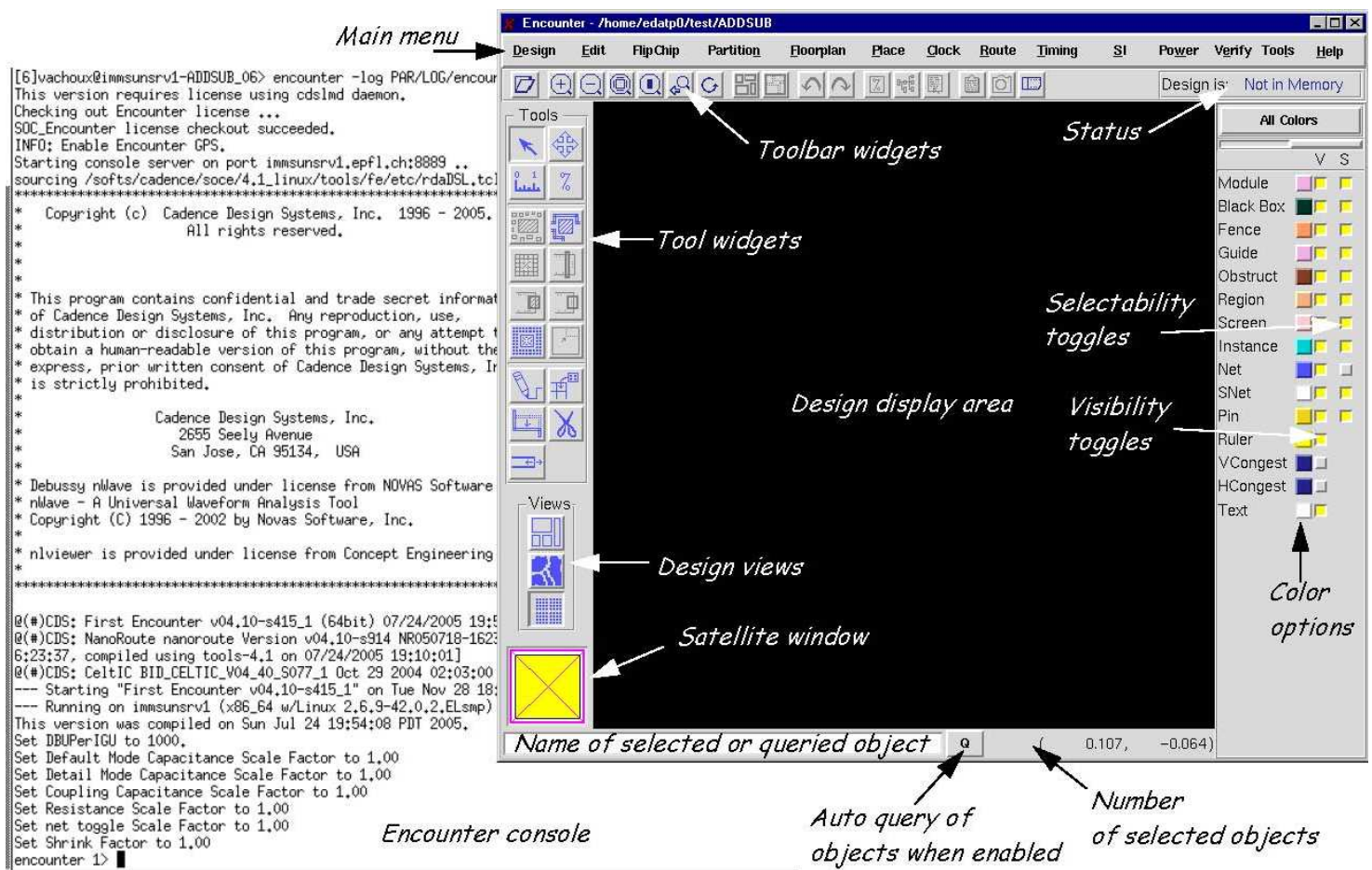
4

This chapter presents the main steps to perform the placement and the routing of the synthesized gate-level netlist using standard cells from the FARADAY design kit. The tool used here is Cadence Encounter. The cdsdoc on-line command gives access to the Cadence documentation. The tool also has a Help menu.

4.1 Starting the Encounter graphical environment

To start the Encounter environment, enter the encounter command in a new Unix shell:

```
student@tango-ADDSUB> encounter -log PAR/LOG/encounter -overwrite
```



that includes all commands entered in the session. If the `-overwrite` switch is not used, both log and command files are incremented at each new session. The Unix shell from which the tool is started is called the Encounter console. The console displays the encounter prompt. This is where you can enter all Encounter text commands and where the tool displays messages. If you use the console for other actions, e.g., Unix commands, the Encounter session suspends until you finish the action.

The main window includes three different design views that you can toggle during a session: the Floorplan view, the Amoeba view, and the Physical view. The Floorplan view displays the hierarchical module and block guides, connection flight lines, and floorplan objects, including block placement, and power/ground nets. The Amoeba view displays the outline of the modules and sub-modules after placement, showing physical locality of the module. The Physical view displays the detailed placements of the module's blocks, standard cells, nets, and interconnects.



The main window includes a satellite window, which identifies the location of the current view in the design display area, relative to the entire design. The chip area is identified by a yellow box, the satellite view is identified by the pink crossbox. When you display an entire chip in the design display area, the satellite crossbox encompasses the chip area yellow box.

When you zoom and pan through the chip in the design display area, the satellite crossbox identifies where you are relative to the entire chip.

- To move to an area in the design display area, click and drag on the satellite crossbox.
- To select a new area in the design display area, click and drag on the satellite crossbox.
- To resize an area in the satellite window, click with the Shift key and drag a corner of the crossbox.
- To define a chip area in the satellite window, right-click and drag on an area.

There are a number of binding keys available (hit the key when the Encounter GUI is active):

b display the list of binding keys

d (de)select or delete objects

f zoom the display to fit the core area

k create a ruler

K remove last ruler displayed

q display the object attribute editor form for the selected object; click the left-button mouse to select an object, Shift-click to select or deselect an object

u undo last command

U redo last command

z zoom-in 2x

Z zoom-out 2x

Arrows pan the display.

Hit CTRL-R to refresh the display.

4.2 Design import

Importing the design into Encounter involves specifying the following setup information:

Design libraries and files. This includes information on the technological process and the cell library in the LEF (Layout Exchange Format) format. LEF files provides information such as metal and via layers and via generate rules which is used for routing tasks. They also provide the minimum information on cell layouts for placement and routing.

Gate-level netlist. This relates to the (Verilog) netlist to be placed and routed.

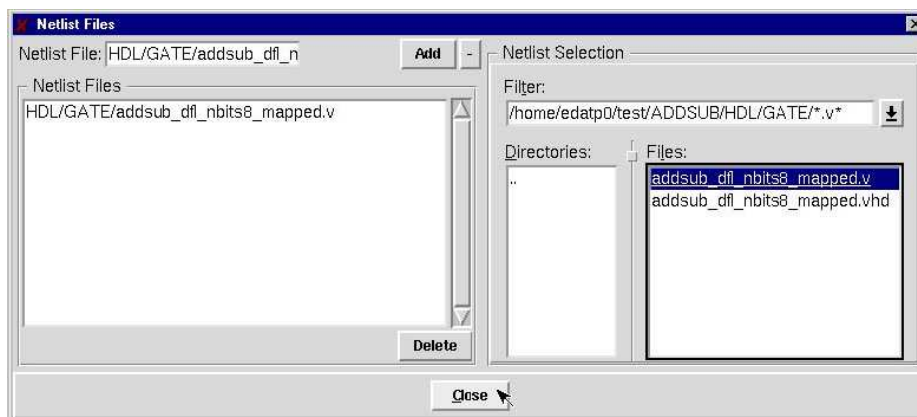
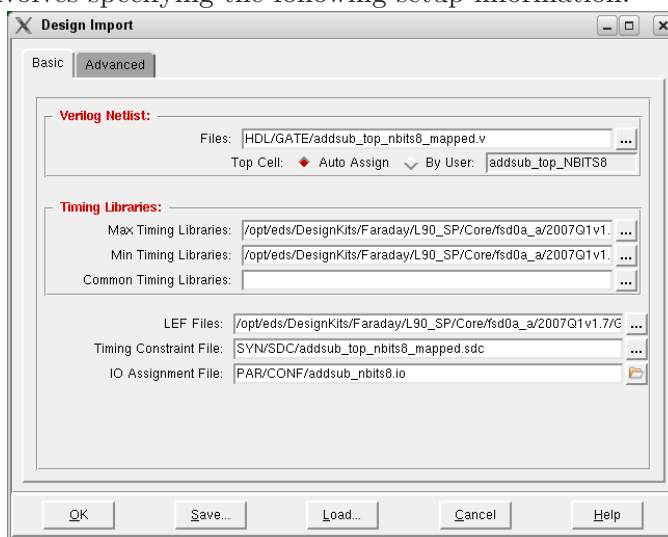
Timing libraries. This includes information on the cell timings (delays, setup/hold times, etc.).

To start the design import, select **Design** ⇒ **Design Import...** in the main menu. Then, click on the **Load...** button and load the file PAR/CONF/L90_SP_std.conf

This file defines a basic import configuration.

There is a number of additions and changes to bring to the initial configuration. The new configuration will then be saved for future uses.

The first information to add is the netlist. Click on the ... button on the right of the Verilog Files field. You get a new dialog window with only one pane. Click on the top-right icon to get the full window. Remove the VERILOG/none line in the left pane.



Select the Verilog netlist file HDL/GATE/addsub_top_nbits8_mapped.v (or the Verilog netlist you want to place and route), add it to the left pane and close the window. It is assumed here that the imported netlist is the one generated for the 10 ns clock period.

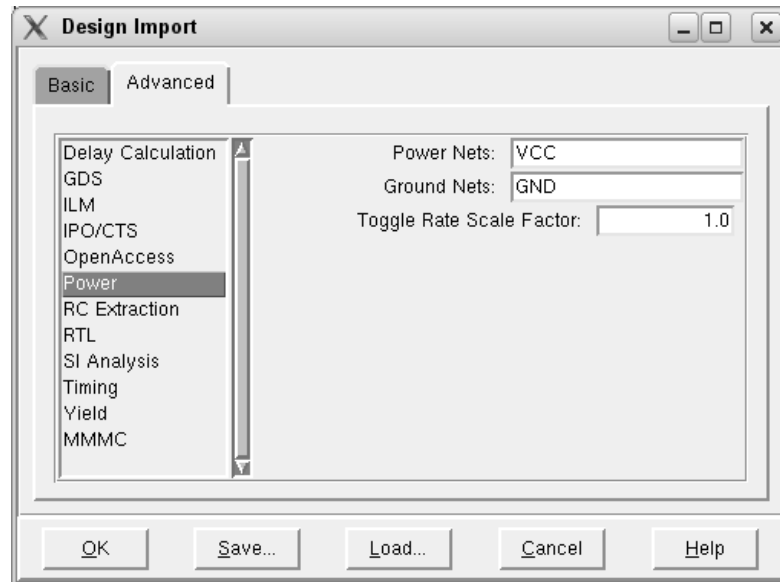
In the Design Import window, select the **Auto Assign** box to let the tool extract the top cell name from the file. If the Verilog file includes more than one design (more than one top module name), you need to give the name of the top module to use explicitly.

In the Timing Constraint File Field:

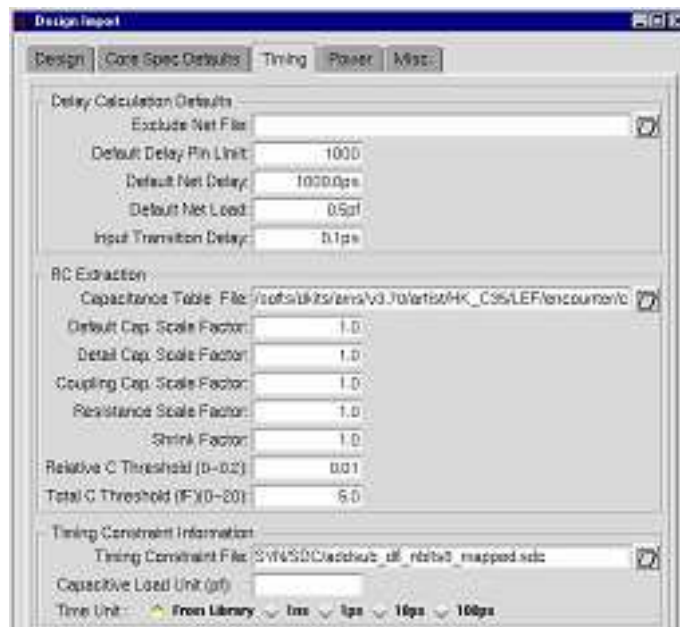
Select the file that has been generated during logic synthesis (3.6 Design mapping and optimization): **SYN/SDC/addsub_top_nbits8_mapped.sdc** Only timing information in the constraint file is actually used by Encounter.

In the IO Assignment File Field:

Select the PAR/CONF/addsub_nbits8.io file



In the **Advanced** tab select in the left pane the **Power** tag, you can keep only the VCC and GND power nets. The names of power and ground nets must be the same as the ones used in the LEF file that describes the standard cells.



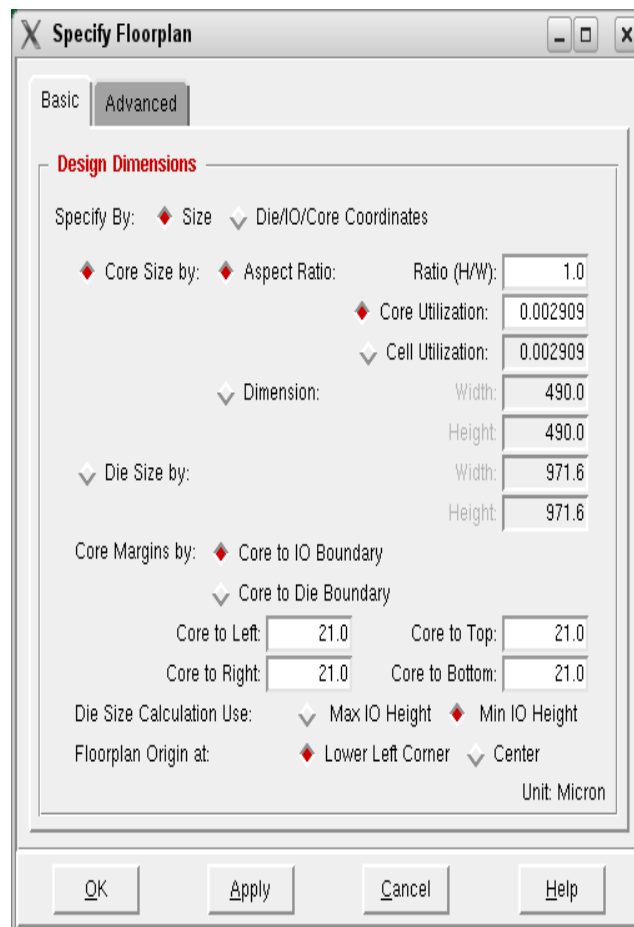
You can now save the updated configuration in the file PAR/CONF/addsub_nbits8.conf by clicking on the **Save...** button.

Finally, click on **OK**. The configuration is then read in.

To reload a configuration, select **Design** \Rightarrow **Design Import...** in the main menu. Then, click on the **Load...** button and load the configuration file from the PAR/CONF directory.

4.3 Floorplan Specification

The floorplan defines the actual form, or aspect ratio, the layout will take, the global and detailed routing grids, the rows to host the core cells and the I/O pad cells (if required), and the location of the corner cells (if required).



Select **Floorplan** \Rightarrow **Specify Floorplan...** in the main menu.

In order to be able to add PAD cells and IO cells we need to specify exact locations for Die/IO/Core so click this button and fill in the following coordinates:

Die LL: 0.0 0.0 UR: 1133.72 1133.72

IO LL: 219.8 219.8 UR: 913.92 913.92

Core LL: 303.8 303.8 UR: 829.92 829.92

Click **OK**.

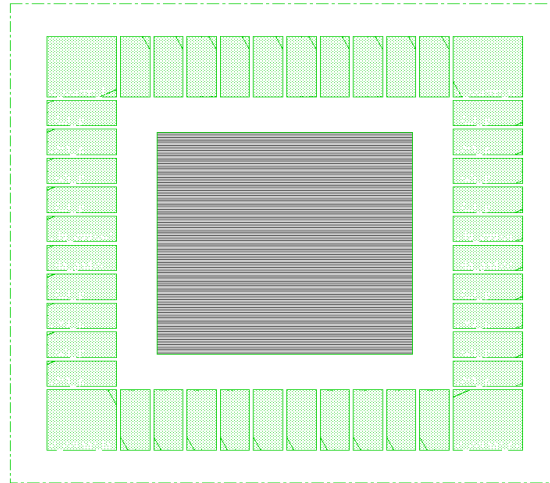
If you want to change the coordinates you will need the metrics of the core, IO and pad cells. These metrics are described in Appendix A.

Now the gaps between the IO cells must be filled with IO-filler cells.

Execute the fillperi.tcl script by the following command on the command line:

encounter;
source PAR/BIN/fillperi.tcl

The display design area pane now shows the defined floorplan with the required number of rows.



It is a good idea to save the design at that stage to allow restarting here quickly without needing to redo all the previous steps.

Select **Design** \Rightarrow **Save Design...** in the main menu and save the current state in the file PAR/DB/addsub_top_nbits8-fplan.enc.

The data are actually saved in the directory PAR/DB/addsub_top_nbits8-fplan.enc.dat.

To restore design data, select **Design** \Rightarrow **Restore Design...** in the main menu and select the .enc file to read in the PAR/DB directory.

4.4 Power ring/stripe creation and routing

This step generates the VCC and ground power rings around the core and optionally adds a number of vertical and/or horizontal power stripes across the core. Stripes ensure a proper power distribution in large cores. They are not strictly required here as the design is small.

Select **Floorplan** \Rightarrow **Power Planning** \Rightarrow **Add Rings...** in the main menu.

The Net(s) field defines the number and the kinds of rings from the core. In our case, there will be first a ground ring around the core and a VCC ring around the ground ring. The net names should be consistent with the power net names in the cell LEF file.

The ring configuration should define ring widths of 2.8 micron spaced by 1.12 micron.

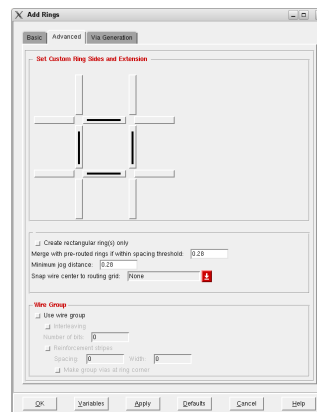
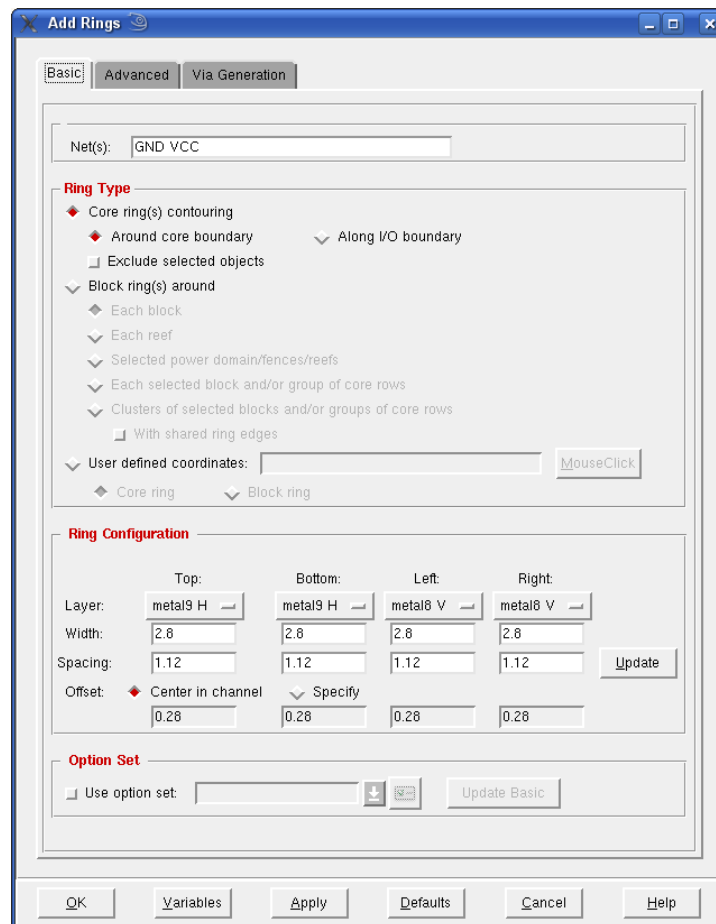
The rings will be placed in the center of the channel between the core and the chip boundary (or the IO pads, if any). Check the Center in channel box in the Ring Configuration part.

It is possible to extend the ring segments to reach the core boundary.

Click on the **Advanced** tab and click on the segments you'd like to extend.

Other power and ground side trunks can be defined by selecting only horizontal or vertical segments.

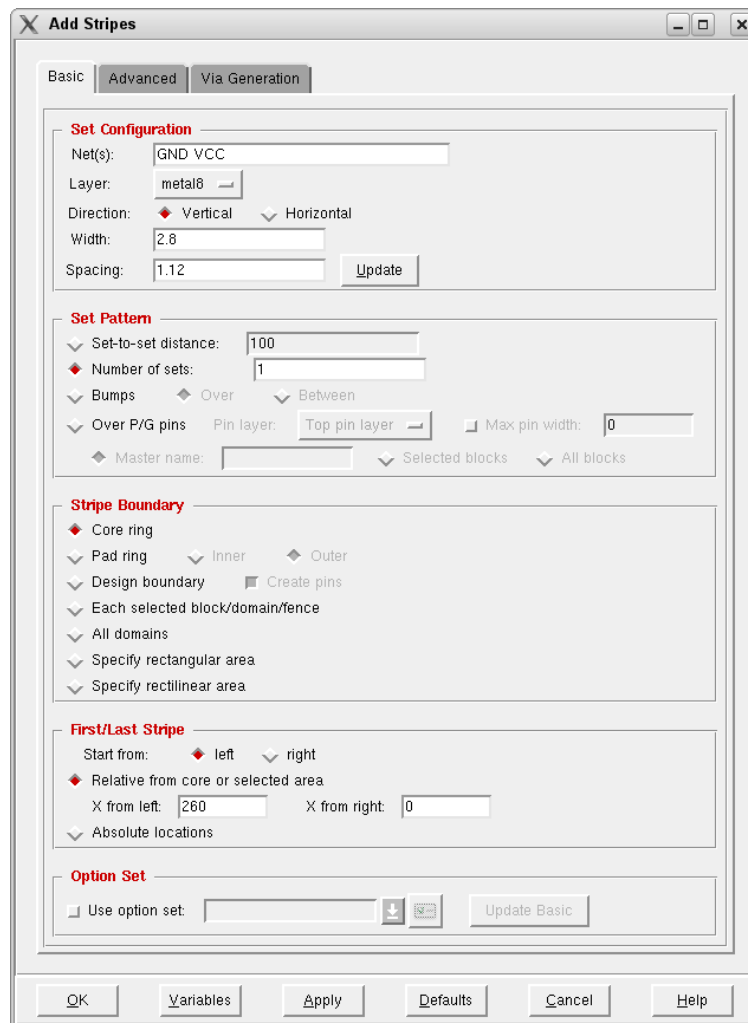
Click **OK** to generate the rings.



To add power stripes, select **Floorplan** ⇒ **Power Planning** ⇒ **Add Stripes...** in the main menu.

The Net(s) field defines the pattern. Here a single pattern will be generated (select Number of sets and insert 1). Each stripe will be 2.8 micron wide and the space between them will be 1.12 micron.

Check Relative from core or selected area and enter the value 260 (micron) in the X from left field. The two stripes will be vertically placed near the center of the core.



It is possible to measure sizes by using the ruler



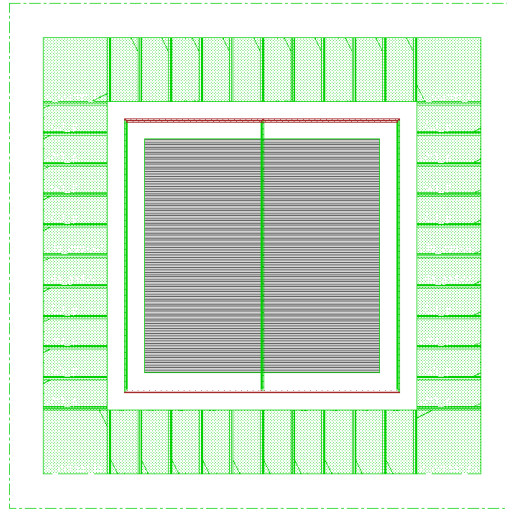
(or hit the **k** binding key). Hit **K** to remove the last ruler or press ESC to remove all rulers.

Click **OK** to generate the stripes. It is possible to remove the stripes by selecting them and hitting the Delete key.

Additionally, the command **Floorplan** \Rightarrow **Clear Floorplan...** allows you to delete all or parts of the floorplan objects.

Now, it is possible to route the power grid. Select **Route** \Rightarrow **SRoute...** in the main menu. All default values are fine. Click **OK** to do the routing. The design now looks like below:

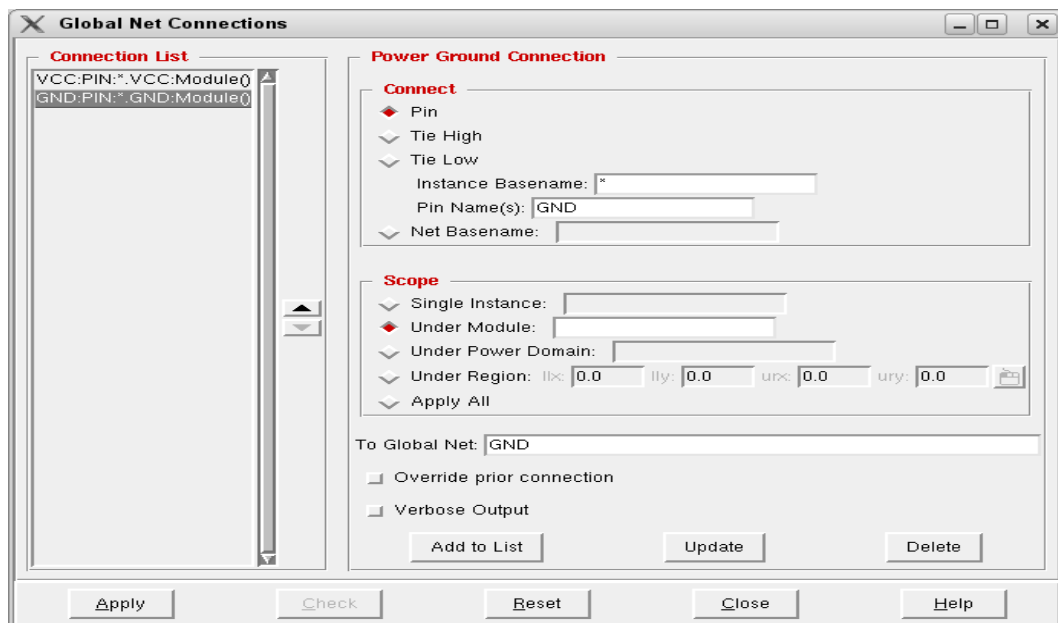
It is recommended to save the new stage of the design. Select **Design** \Rightarrow **Save Design...** in the main menu and save the current state in the file PAR/DB/addsub_top_nbits8-pring.enc.



4.5 Global net connections

This step assigns pins or nets to global power and ground nets. The imported Verilog netlist does not mention any power and ground connections. However, the cells that will be placed do have power/ground pins that will need to be routed to the global power/ground nets defined for the block.

Select **Floorplan** ⇒ **Connect Global Nets...** in the main menu.

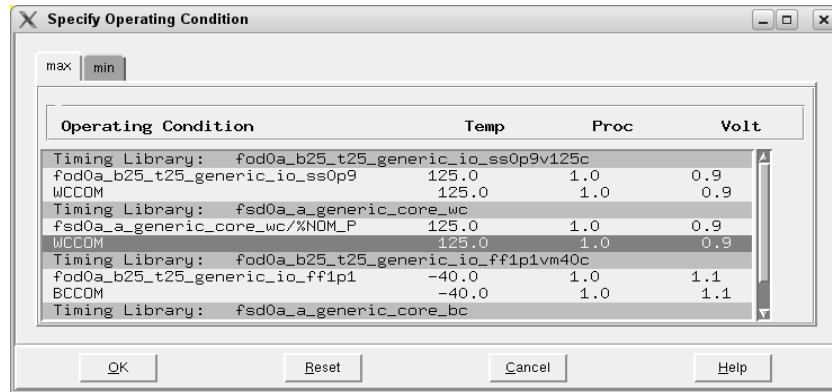


The left pane (Connection List) is initially empty. For each VCC and ground net: Check the Pins field and enter the pin name (VCC or GND). Fill the To Global Net field with either VCC or GND. Click on Add to List. The left pane now includes the related global net connection. Click on Apply and then on Close.

4.6 Operating conditions definition

The operating conditions define the temperature, process and voltage conditions for the design. They impact the timing calculations and optimizations.

Select the **Timing** ⇒ **Analysis Condition** ⇒ **Specify Operating Condition/PVT...** in the main menu.



In the max tab, select the WCCOM operating condition. In the min tab, select the BCCOM operating condition.

Click **OK**.

The max operating conditions will be used to meet setup timing constraints, while the min operating conditions will be used to meet hold timing constraints.

The Encounter console summarizes the settings:

```
Set Min operating condition for timing library group 'min' to 'BCCOM'
defined in Timing Library 'fsd0a_a_generic_core_bc'
Process: 1.00 Temperature: -40.000 Voltage: 1.100
4 Set Max operating condition to 'WCCOM' defined in Timing Library '
fsd0a_a_generic_core_wc'
Process: 1.00 Temperature: 125.000 Voltage: 0.900
*** Calculating scaling factor for min libraries using operating
condition:
Name: BCCOM Process: 1.00 Temperature: -40.000 Voltage: 1.100
*** Calculating scaling factor for max libraries using operating
condition:
9 Name: WCCOM Process: 1.00 Temperature: 125.000 Voltage: 0.900
```

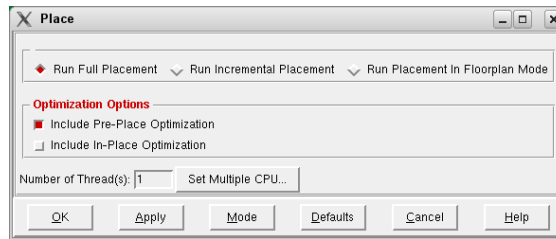
Running the following command in the Encounter console gives the active operating conditions:

```
1 encounter 1> getOpCond -v
min: BCCOM proc: 1.0000 volt: 1.1000 temp: -40.0000
max: WCCOM proc: 1.0000 volt: 0.9000 temp: 125.0000
```

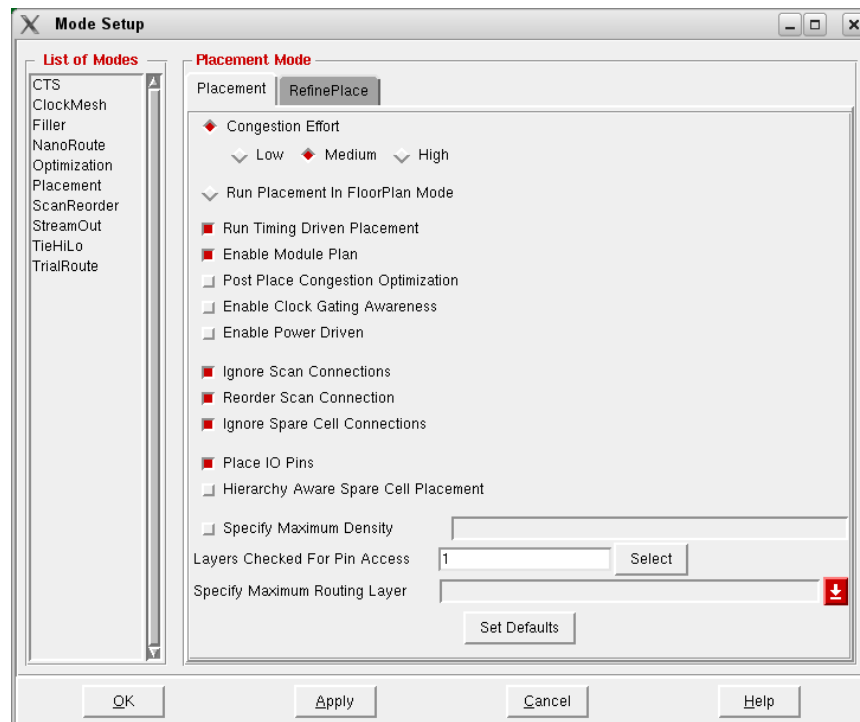
4.7 Core cell placement

This step places the cells of the imported Verilog netlist in the rows.

Select **Place** ⇒ **Standard Cells...** in the main menu.



By clicking the **Mode** button one can specify placement options. By default it will run in **Timing Driven Placement Mode**. Stick to the default options and click **OK**.

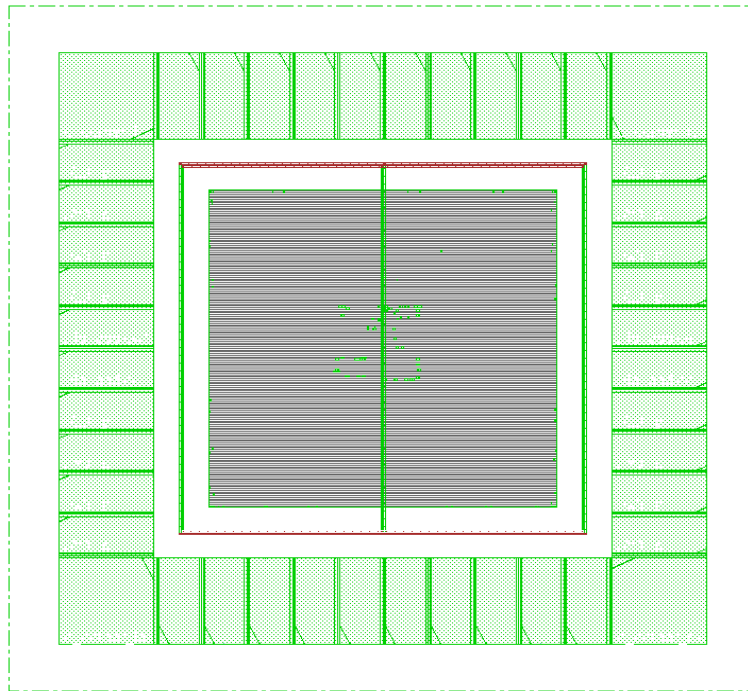


The **Timing Driven Placement Mode** option will optimize the placement of the cells that are on the critical path. Some cell instances may be replaced with cells having lower driving capabilities (downsizing) or stronger driving capabilities (upsizing). Buffers may be also added or deleted. The Encounter console notifies such changes.

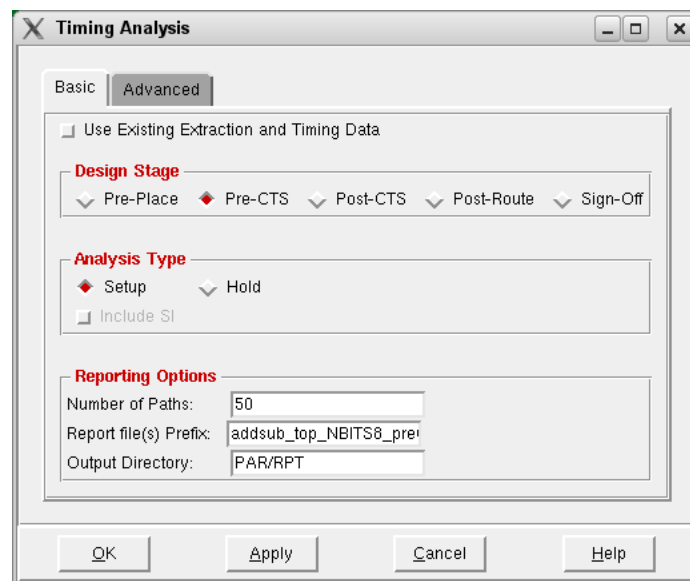
Click **OK** to do the placement. It may take some time to complete, especially when the placement is timing driven and a high effort level is used .

The placement should then look like below:

It is recommended to save the new stage of the design. Select **Design** ⇒ **Save Design...** in the main menu and save the current state in the file PAR/DB/addsub_top_nbbits8-placed.enc.



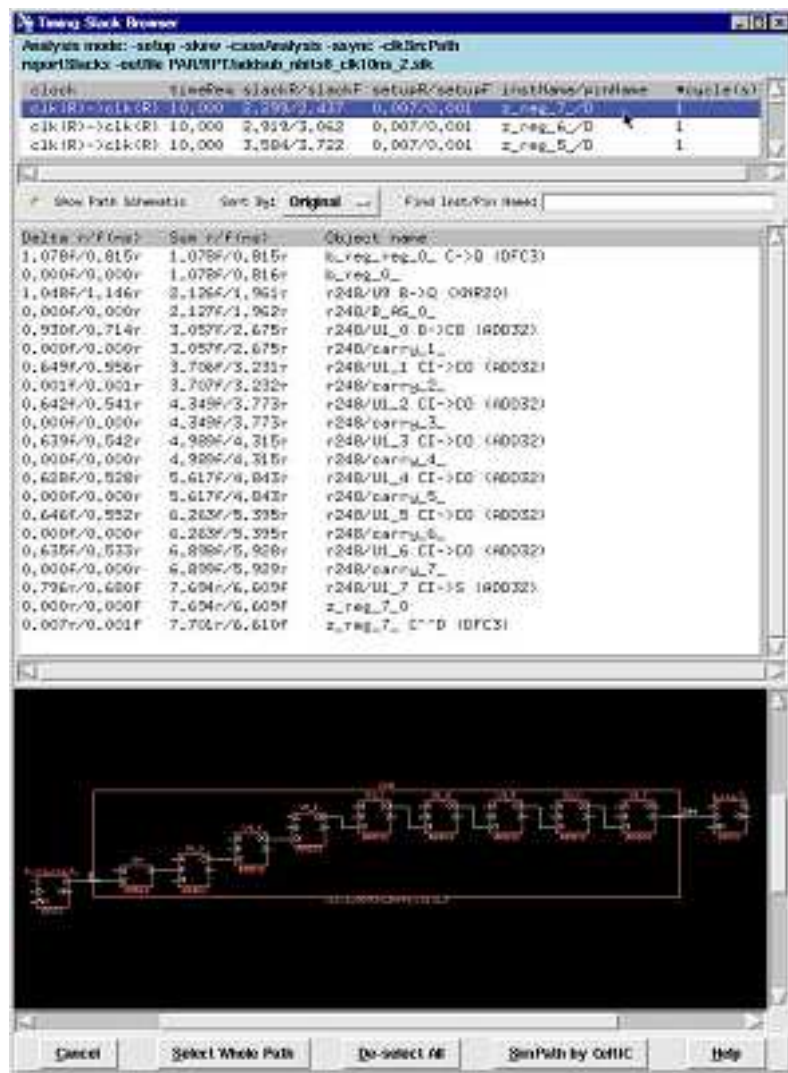
4.8 Post-placement timing analysis



The timing analysis engine in Encounter can now be run to get a relatively good idea of the timing performances of the design. It actually performs a trial routing and a parasitic extraction based on the current cell placement.

Select **Timing** ⇒ **Analyze Timing...** in the main menu. Define the path for the slack report file. Click **OK**.

In the Encounter console window you get a summary of the timing analysis:



```
# generated on Thu May 7 11:21:43
2 # Top Cell: addsub_top_NBITS8
```

timeDesign Summary							

7							
	Setup mode	all	reg2reg	in2reg	reg2out	in2out	clkgate
	WNS (ns):	4.978	8.455	5.906	4.978	N/A	N/A
12	TNS (ns):	0.000	0.000	0.000	0.000	N/A	N/A
	Violating Paths:	0	0	0	0	N/A	N/A
	All Paths:	56	8	47	8	N/A	N/A

17							
	DRVs	Real		Total			
		Nr	Worst Vio	Nr			

22	max_cap	0	0.000	0			

max_tran	0	0.000	0
max_fanout	0	0	0

27 Density: 0.254%
Routing Overflow: 0.00% H and 0.00% V

The design is not critical as the slack is positive (4.978 ns).

To get more details on the critical path, select

Timing ⇒ Debug Timing... in the main menu.

Another way to do a timing analysis is to execute the following commands in the Encounter console:

```
encounter 16; reportTA
```

The following report is then displayed in the console:

```

1 Path 1: MET Late External Delay Assertion
  Endpoint: z[7] (^) checked with leading edge of 'clk'
  Beginpoint: addsub_1/z_reg_7/Q (^) triggered by leading edge of 'clk'
  Other End Arrival Time 0.000
  - External Delay 2.000
6 + Phase Shift 10.000
  = Required Time 8.000
  - Arrival Time 3.914
  = Slack Time 4.086
  Clock Rise Edge 0.000
11 = Beginpoint Arrival Time 0.000

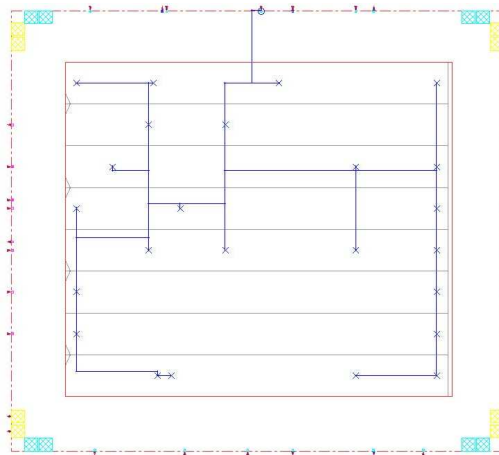
```

Instance	Arc	Cell	Delay	Arrival Time	Required Time
io_clk/x0_ip	clk ^	UYNGB	0.696	0.000	4.086
addsub_1/z_reg_7_	I ^ -> 0 ^	QDFFRBX1	0.635	0.696	4.782
io_z/x0_7/x0_op	CK ^ -> Q ^	VYA4GSGB	2.583	1.331	5.417
	I ^ -> 0 ^		0.000	3.914	8.000
	z[7] ^			0.000	8.000

4.9 Clock tree synthesis (optional)

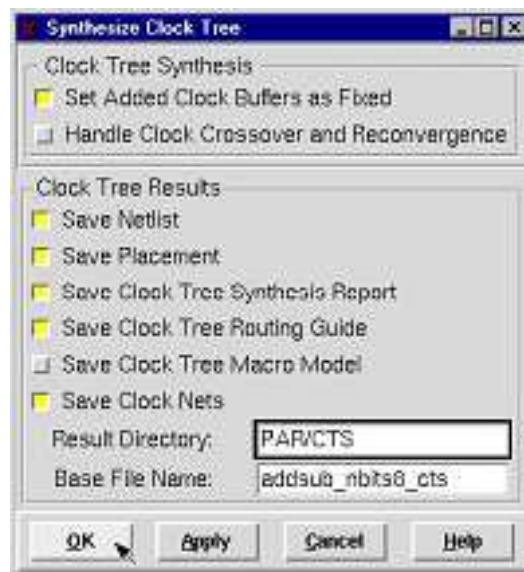
As the paths that will propagate the clock signal in the design are not necessarily balanced, some registers may receive the active clock edge later than others (clock skew) and may therefore violate the assumed synchronous design operation. For example, the original clock tree we can get from the previously placed design is shown below.





To create a balanced clock tree, you have first to create a clock tree specification file. Encounter can create a first draft version of the file you can then edit to include design specific data.

Select **Clock** \Rightarrow **Design Clock...** in the main menu. In the **Synthesize Clock Tree** popup menu change the Results Directory to PAR/CTS and click on the **Gen Spec...** button to get the **Generate Clock Spec** menu. Select the BUFCKX* and INVCK* cells for footprints and save the specification in the file PAR/CTS/addsub_top_nbits8-spec.cts.



Click **OK** to generate the specification file. You can then edit the file to change timing values such as the maximum allowed clock skew (300ps by default).

The next step is to load the clock tree specification file.

Select **Clock** \Rightarrow **Specify Clock Tree ...** in the main menu and select the file PAR/CTS/addsub_nbits8-spec.cts that has been previously created (and possibly edited).

To create the clock tree, select

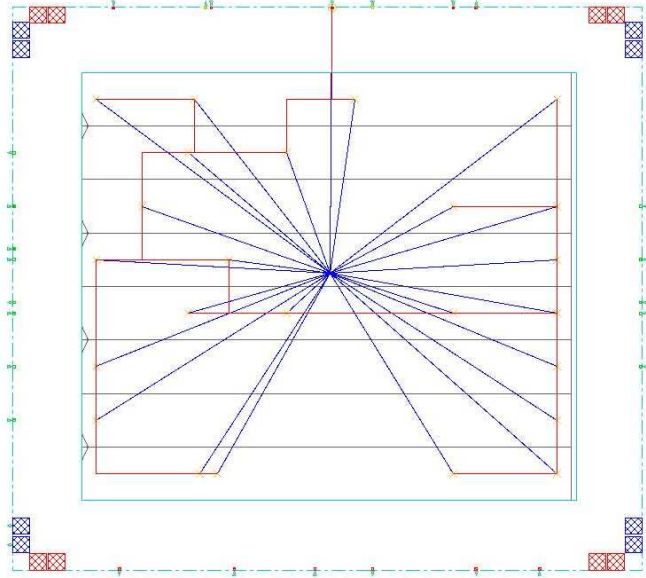
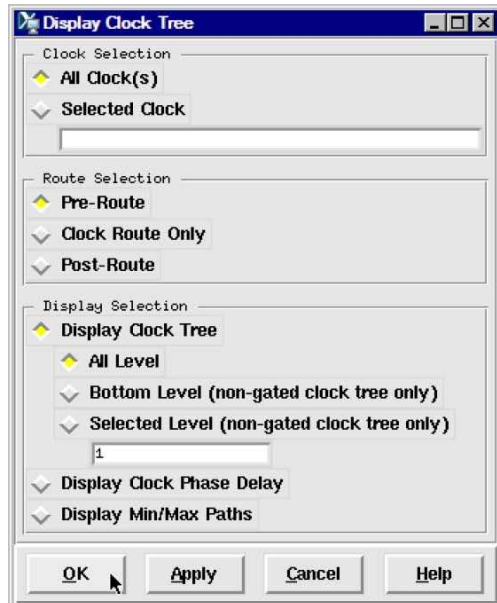
Clock \Rightarrow **Synthesize Clock Tree ...** in the main menu.

Define the result directory as PAR/CTS and the base file name as addsub_nbits8_cts.

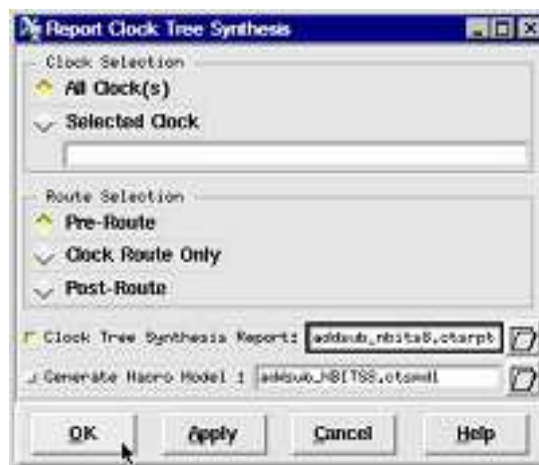
Click **OK** to create the clock tree.

To display the generated clock tree, select

Clock ⇒ Display ⇒ Display Clock Tree ... in the main menu.



The clock paths have been balanced according to the clock tree specifications.



To get a report on the clock tree synthesis, enter the following command at the encounter prompt.

```
encounter 1> reportClockTree -report PAR/RPT/addsub_top_nbts8.ctsrpt
```

The following report is also displayed in the Encounter console:

```
Redoing specifyClockTree -clkfile PAR/CTS/addsub_nbts8-spec.cts ...
```

```
4 reportClockTree Option :  
*** Look For Reconvergent Clock Component ***
```


The `clock` tree `clk` has no reconvergent cell.

```
#
9 # Mode : Setup
# Library Name : fod0a_b25_t25_generic_io_ss0p9v125c
# Operating Condition : fod0a_b25_t25_generic_io_ss0p9v125c/%NOM_PVT
# Process : 1
# Voltage : 0.9
14 # Temperature : 125
#
***** Clock clk Pre-Route Timing Analysis *****
Nr. of Subtrees : 2
Nr. of Sinks : 24
19 Nr. of Buffer : 0
Nr. of Level (including gates) : 1
Root Rise Input Tran : 0.1(ps)
Root Fall Input Tran : 0.1(ps)
Max trig. edge delay at sink(R): addsub_1/z_reg_0_/CK 770.4(ps)
24 Min trig. edge delay at sink(R): addsub_1/a_reg_reg_2_/CK 768(ps)

(Rise Phase Delay) (Actual) (Required)
29 Rise Phase Delay : 768~770.4(ps) 0~10000(ps)
Fall Phase Delay : 564.4~566.8(ps) 0~10000(ps)
Trig. Edge Skew : 2.4(ps) 300(ps)
Rise Skew : 2.4(ps)
Fall Skew : 2.4(ps)
Max. Rise Buffer Tran. : 31(ps) 400(ps)
34 Max. Fall Buffer Tran. : 31(ps) 400(ps)
Max. Rise Sink Tran. : 101.1(ps) 400(ps)
Max. Fall Sink Tran. : 85.6(ps) 400(ps)
Min. Rise Buffer Tran. : 31(ps) 0(ps)
Min. Fall Buffer Tran. : 31(ps) 0(ps)
39 Min. Rise Sink Tran. : 101(ps) 0(ps)
Min. Fall Sink Tran. : 85.6(ps) 0(ps)

Generating Clock Analysis Report addsub_top_NBITS8.ctrpt ....
44 Clock Analysis (CPU Time 0:00:00.0)
```

```
*** End reportClockTree (cpu=0:00:00.0, real=0:00:00.0, mem=347.5M) ***
```

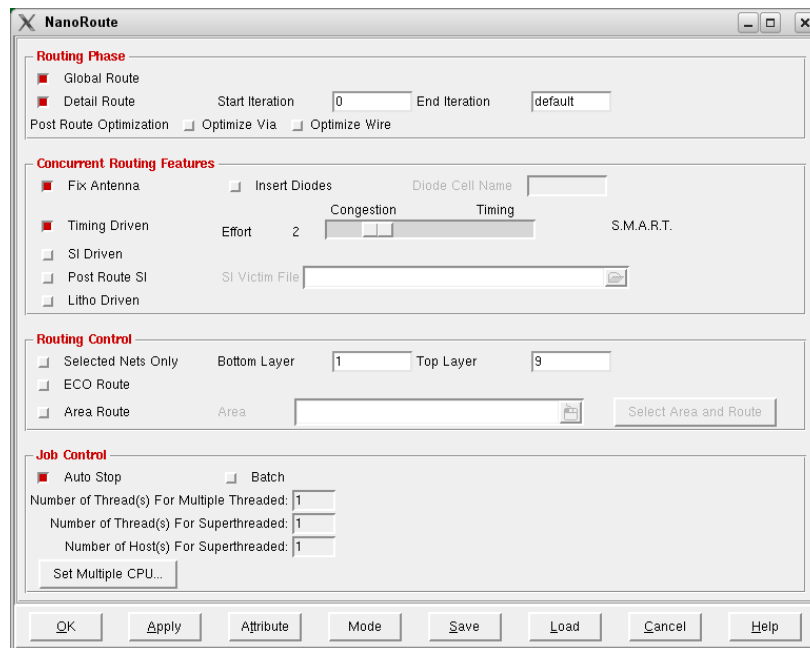
Several clock report files are also available in the PAR/CTS directory.

It is recommended to save the new stage of the design. Select **Design** ⇒ **Save Design...** in the main menu and save the current state in the file PAR/DB/addsub_top_nbbits8-cts.enc.

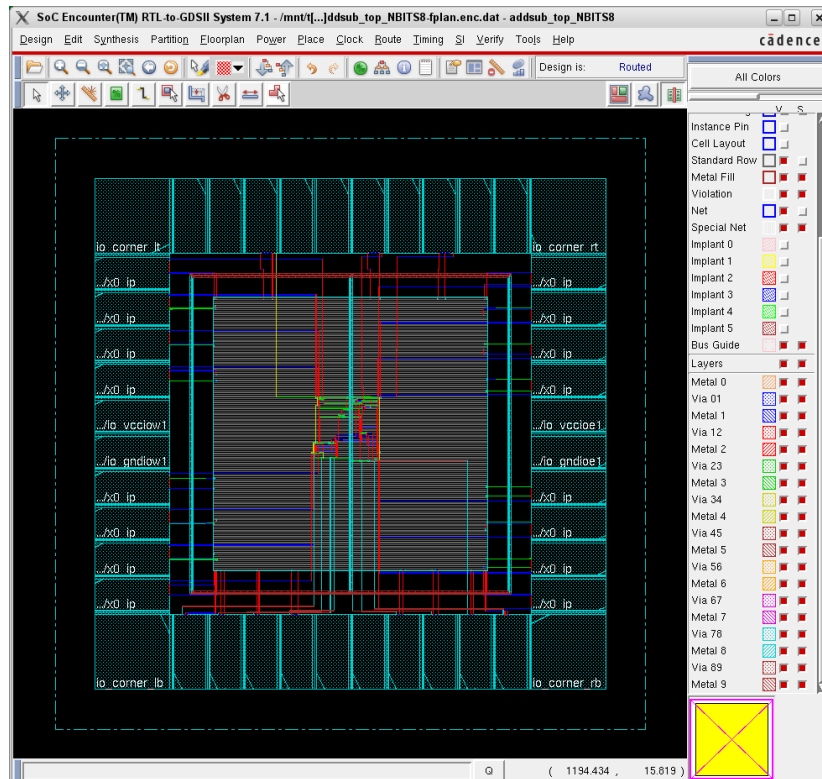
4.10 Design routing

This steps generates all the wires required to connect the cells according to the imported gate-level netlist.

To route the design, select



Route \Rightarrow Nanoroute... in the main menu, check the Timing Driven box and a maximum effort.
Click **OK** to do the routing.
You now get the routed design:



It is recommended to save the new stage of the design. Select **Design** \Rightarrow **Save Design...** in the main menu and save the current state in the file
 PAR/DB/addsub_top_nbits8-routed.enc.

4.11 Post-routing timing optimization and analysis



A final timing optimization may be done on the routed design. Select **Timing** \Rightarrow **Optimize...** in the main menu. Select the postRoute box. Click **OK**.

The results of the optimization is displayed in the Encounter console:

Initial Summary

7	Setup mode	all
	WNS (ns):	4.086
	TNS (ns):	0.000
	Violating Paths:	0
12	All Paths:	56

17	DRVs	Real		Total
		Nr	Worst Vio	Nr
	max_cap	0	0.000	0
	max_tran	0	0.000	0
22	max_fanout	0	0	0

Density: 0.254%

```

**optDesign ... cpu = 0:00:00, real = 0:00:00, mem = 349.6M **
27 *** Timing Is met
*** Check timing (0:00:00.0)
*** Setup timing is met (target slack 0.0ns)
Reported timing to dir ./timingReports
**optDesign ... cpu = 0:00:00, real = 0:00:00, mem = 349.6M **
32
-----
      optDesign Final Summary
-----

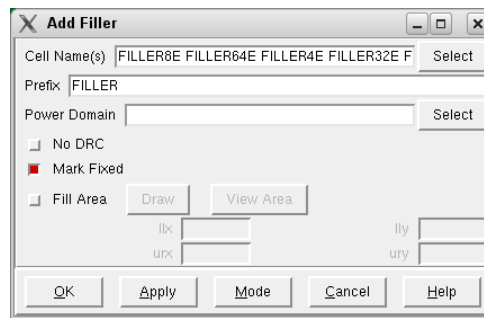
37 +-----+-----+-----+-----+-----+-----+
|      +-----+
|      Setup mode      |      all      |      reg2reg   |      in2reg    |      reg2out   |      in2out    |
|      clkgate |
+-----+-----+-----+-----+-----+-----+
|      +-----+
|      WNS (ns):|      4.086   |      8.426   |      6.669   |      4.086   |      N/A      |
|      N/A      |
|      TNS (ns):|      0.000   |      0.000   |      0.000   |      0.000   |      N/A      |
|      N/A      |
42 |      Violating Paths:|      0      |      0      |      0      |      0      |      N/A      |
|      N/A      |
|      All Paths:|      56     |      8      |      47     |      8      |      N/A      |
|      N/A      |
+-----+-----+-----+-----+-----+-----+
|      +-----+
|
+-----+-----+-----+-----+-----+-----+
47 |      +-----+-----+-----+-----+
|      DRV's      |      +-----+-----+-----+
|      |          |      Nr      |      Worst Vio      |      Nr      |
|      +-----+-----+-----+-----+
52 |      max_cap    |      0      |      0.000          |      0      |
|      max_tran    |      0      |      0.000          |      0      |
|      max_fanout  |      0      |      0              |      0      |
|      +-----+-----+-----+-----+
|
Density: 0.254%
57 -----
**optDesign ... cpu = 0:00:01, real = 0:00:00, mem = 349.6M **
-core {} # string, default=""
*** Finished optDesign ***

```

4.12 Filler cell placement

Filler cells will fill remaining holes in the rows and ensure the continuity of power/ground rails and N+/P+ wells in the rows. To fill the holes with filler cells, select **Place** ⇒ **Filler** ⇒ **Add Filler...** in the main menu.

Select the cells FILLER64E, FILLER32E, FILLER16E, FILLER8E, FILLER4E, FILLER3, FILLER2 and FILLER1 and click **OK** to place the filler cells.



4.13 Design checks

The Verify menu has a number of items to check that the design has been properly placed and routed.



Select **Verify** ⇒ **Verify Connectivity...** in the main menu. Define the report file as PAR/RPT/addsub.top_nbits8_conn.rpt.

Click **OK**.

The console displays the results:

```
***** Start: VERIFY CONNECTIVITY *****
```

```
Start Time: Thu Dec 1 18:52:00 2005
```

```
4 Design Name: addsub{\_}NBITS8
```

```
Database Units: 1000
```

```
Design Boundary: (0.0000, 0.0000) (153.0500, 136.9000)
```

```
Error Limit = 1000; Warning Limit = 50
```

```
Check all nets
```

```
9 Begin Summary
```

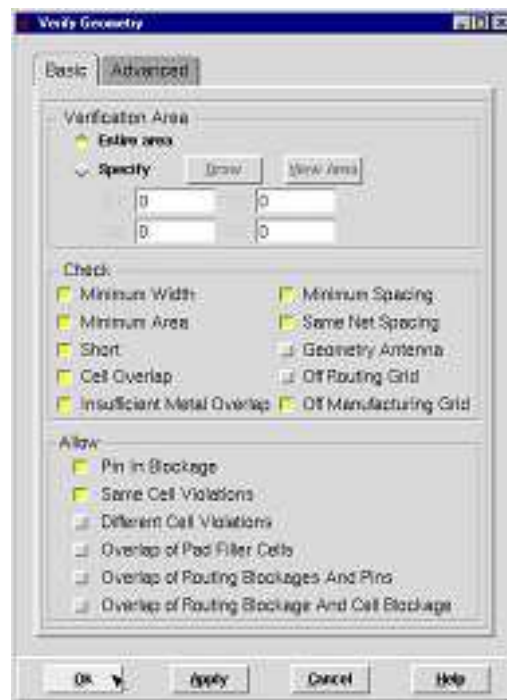
```
Found no problems or warnings.
```

```
End Summary
```

```
End Time: Thu Dec 1 18:52:00 2005
```

```
14 ***** End: VERIFY CONNECTIVITY *****
```

Verification Complete : 0 Viols. 0 Wrngs.



Select **Verify** ⇒ **Verify Geometry...** in the main menu. In the **Advanced** tab, define the report file as
PAR/RPT/addsub_nbits8-geom.rpt.
Click **OK**.

The console displays the results:

```
*** Starting Verify Geometry (MEM: 222.2)
VERIFY GEOMETRY ..... Starting Verification
VERIFY GEOMETRY ..... Initializing
4  VERIFY GEOMETRY ..... Deleting Existing Violations
VERIFY GEOMETRY ..... Creating Sub-Areas
VERIFY GEOMETRY ..... SubArea : 1 of 1
VERIFY GEOMETRY ..... Cells : 0 Viols.
VERIFY GEOMETRY ..... SameNet : 0 Viols.
9  VERIFY GEOMETRY ..... Wiring : 0 Viols.
VERIFY GEOMETRY ..... Antenna : 0 Viols.
VERIFY GEOMETRY ..... Sub-Area: 1
complete 0 Viols. 0 Wrngs.
Begin Summary ...
14 Cells : 0
SameNet : 0
Wiring : 0
Antenna : 0
Short : 0
19 Overlap : 0
End Summary

Verification Complete : 0 Viols. 0 Wrngs.
```

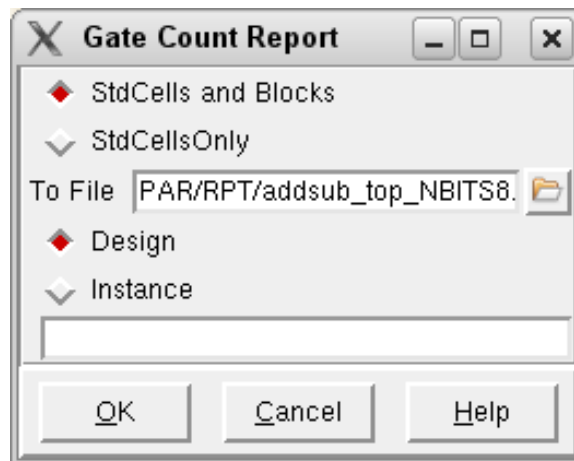
4.14 Report generation

A number of reports have been already generated in the previous steps. They should be located in the PAR/RPT directory. The Tools menu includes some additional reports:

Design ⇒ Report ⇒ Netlist Statistics gives the following output in the console:

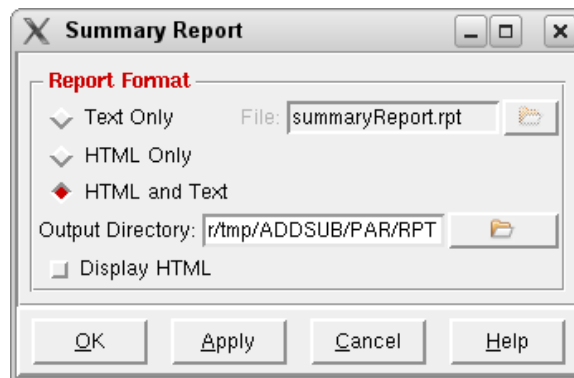
```
encounter 2> *** Statistics for net list addsub_top_NBITS8 ***
Number of cells      = 338
3  Number of nets     = 166
Number of tri-nets   = 8
Number of degen nets = 0
Number of pins       = 466
Number of i/os       = 27
8
Number of nets with 2 terms = 114 (68.7%)
Number of nets with 3 terms = 21 (12.7%)
Number of nets with 4 terms = 28 (16.9%)
Number of nets with >=10 terms = 3 (1.8%)
13
*** 21 Primitives used:
Primitive VCCKGB (1 insts)
Primitive GNDKGB (1 insts)
Primitive VCC2IOGB (5 insts)
18 Primitive GND2IOGB (6 insts)
Primitive EMPTY1GB (44 insts)
Primitive CORNERGB (4 insts)
Primitive EMPTY16GB (44 insts)
```

```
Primitive EMPTY8GB (44 insts)
23 Primitive EMPTY4GB (44 insts)
Primitive VYA4GSGB (8 insts)
Primitive UYNGB (19 insts)
Primitive QDFFRBX1 (24 insts)
Primitive XOR2X1 (5 insts)
28 Primitive TIE1X1 (27 insts)
Primitive TIEOX1 (27 insts)
Primitive MXL2XLP (13 insts)
Primitive MUX2X1 (1 insts)
Primitive MAO222X1 (4 insts)
33 Primitive INVX1 (2 insts)
Primitive INVCKX1 (13 insts)
Primitive FA1X1 (2 insts)
*****
```

Design ⇒ **Report** ⇒ **Gate Count...** gives the following output in the console:

```
Gate area 2.3520 um^2
[0] addsub_top_NBITS8 Gates=297 Cells=118 Area=698.5 um^2
```

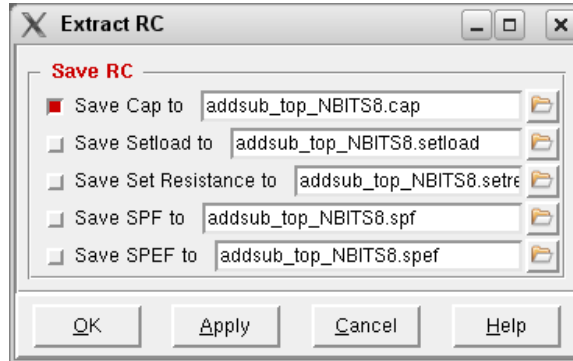


Finally, **Design** ⇒ **Report** ⇒ **Summary...** displays the following window:

```
summaryReport -outdir summaryReport
Creating directory summaryReport.
3 Start to collect the design information.
Build netlist information for Cell addsub_top_NBITS8.
Finish to collect the design information.
Generating standard cells used in the design report.
Generating IO cells used in the design report.
8 Analyze library ...
Analyze netlist ...
Analyze timing ...
Analyze floorplan/placement ...
Analysis Routing ...
13 Report saved in file summaryReport/addsub_top_NBITS8.main.htm.ascii.
```

4.15 Post-route timing data extraction

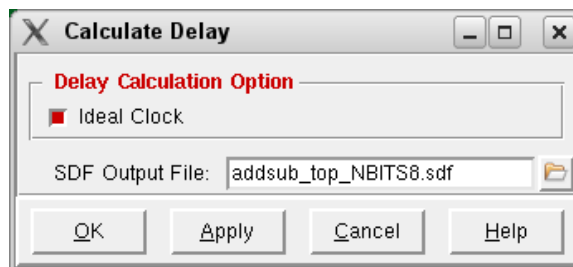
This step generates the post-route SDF file that includes both the actual interconnect and cell timing delays.



The parasitics must be first extracted.

Select **Timing** ⇒ **Extract RC...** in the main menu.

The generated Cap file includes the wired capacitance, pin capacitance, total capacitance, net length, wire cap per unit length and the fanout of each net in the design. The generated SPEF (Standard Parasitics Exchange Format) file includes RC values in a SPICE-like format.

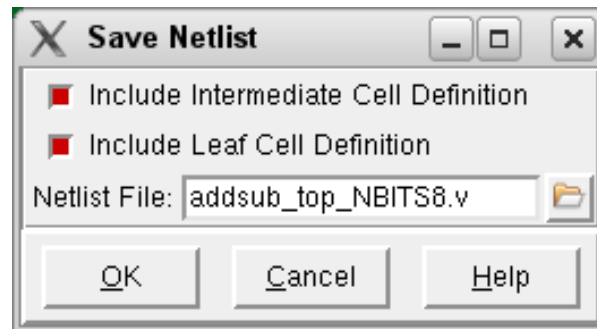


The SDF file may be then generated by selecting

Timing ⇒ **Calculate Delay...** in the main menu. The checked Ideal Clock switch means that flip-flops are considered as having 0ps rising and falling transition times.

4.16 Post-route netlist generation

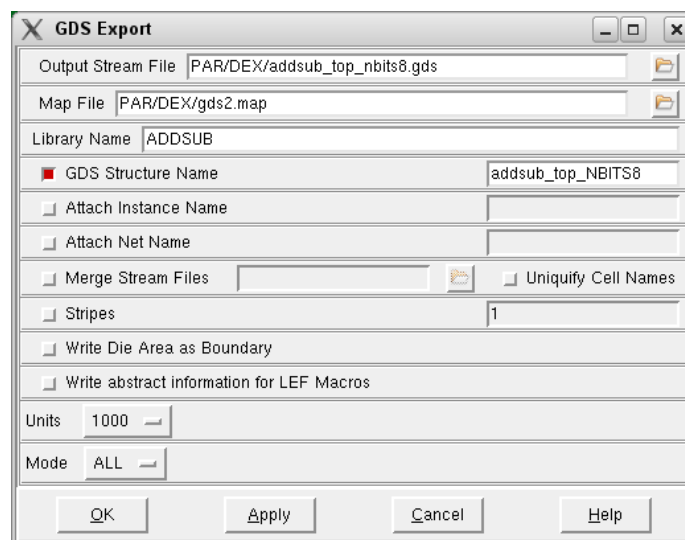
This step generates a Verilog netlist of the routed design. The netlist may be different from the imported netlist as cells may have been added or replaced during clock tree synthesis and timing-driven optimizations.



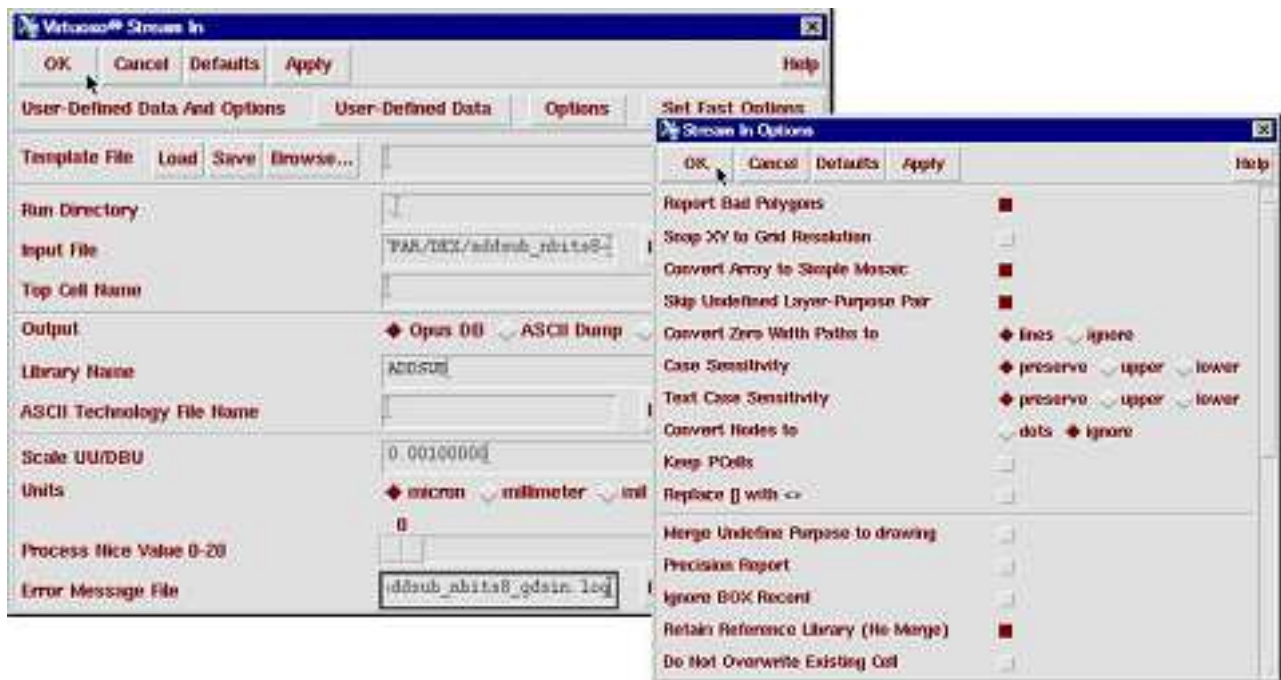
Select **Design** ⇒ **Save** ⇒ **Netlist...** in the main menu. Do not select Include Leaf Cell Definition as they are provided in a separate library.

The generated file should go into the HDL/GATE directory.

4.17 GDS2 file generation



The placed and routed design can be exported in different formats for further processing outside the Encounter tool. The GDS2 binary format is a standard format for integrating the block in the top-level layout, doing DRC/LVS checkings, or delivering the layout to the foundry. To export the design in the GDS2 format, select **Design** ⇒ **Save** ⇒ **GDS...** in the main menu. The GDS map file has been copied by the tech.setup script in the PAR/DEX directory. The generated GDS2 file is written in the same directory.



4.18 Using scripts

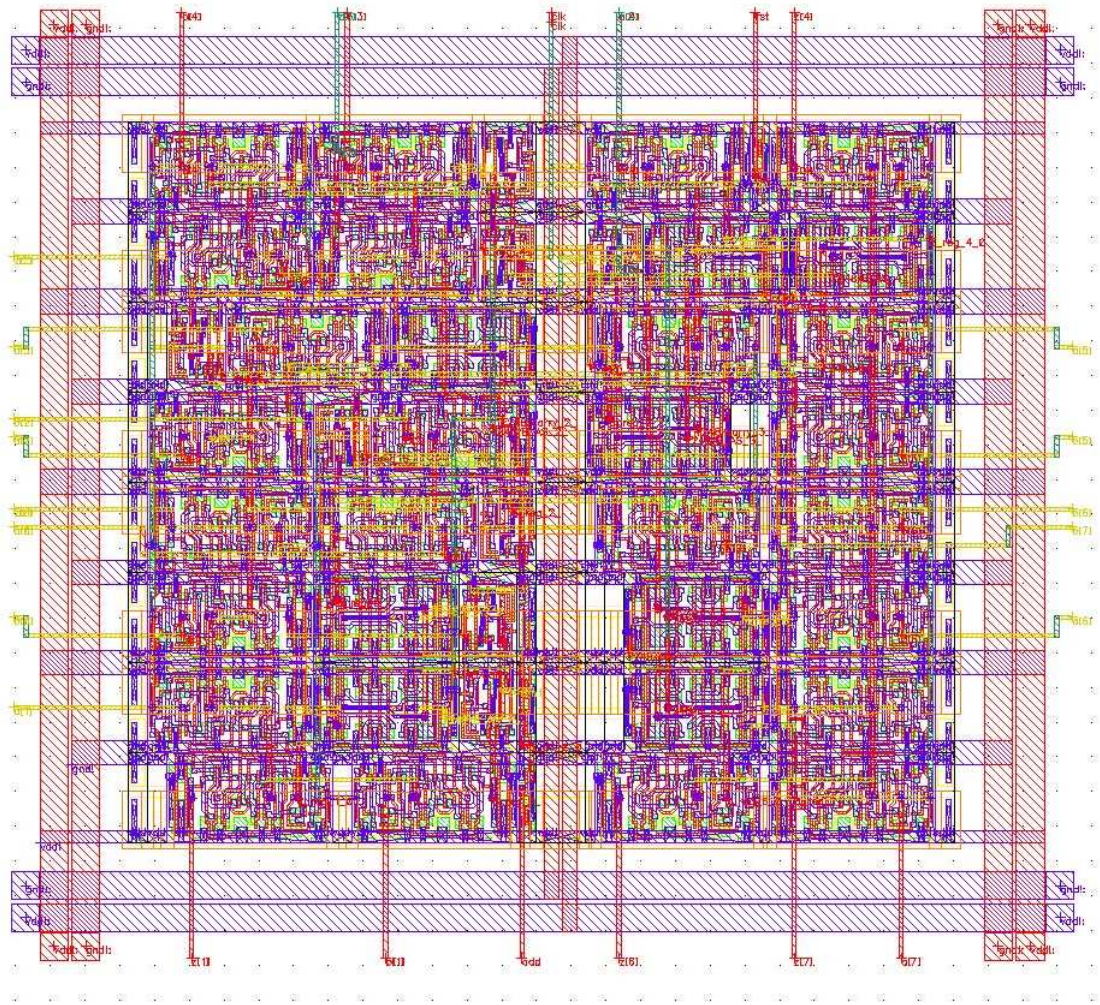
As for the synthesis step, it is much more convenient to capture the placement and routing flow in a script. Cadence Encounter also support sthe Tcl language for building scripts.

An example of such a script for placement and routing of the adder-subtractor design has been installed in the PAR/BIN directory (see “1.5 VHDL example: Adder-subtractor”). The script must be run from the project top directory and it assumes a directory organization as described in “1.2 Design project organisation”. To run the Tcl script, execute the following command in a Unix shell:

```
encounter -log PAR/LOG/encounter -overwrite -init PAR/BIN/addsub_par.
tcl -win
```

The script is given below. It may be modified to define design information and to control the flow to some extent. Note that a configuration file must exist before running the script. The configuration file name is in the PAR/CONF directory and its name is defined in the script.

The script does a bit more than the steps described earlier. For example, it uses an I/O pin placement definition as provided by a PAR/CONF/*.io file.



```

#
# Cadence Encounter Tcl script for adder-subtractor.
#
4 # Process: AMS 0.35u CMOS (C35), Hit-Kit 3.70
#
#
-----

# It is assumed that a project directory structure has already been
# created using 'create_project' and that this synthesis script is
9 # executed from the project root directory $PROJECT_DIR
#
-----

set PROJECT_DIR [pwd]
#
-----

# Design related information (can be changed)

```

```

14 #
-----

set DESIGN addsub_nbits8
set TIM_LIBRARY fsd0a_a_generic_core
set TIM_OC_MAX WCCOM ;# TYPICAL | WORST | WORST-IND
set TIM_OC_MIN BCCOM ;# TYPICAL | BEST | BEST-IND
19 # Floorplan settings
#
set FP_ASPECT_RATIO 1
set FP_ROW_DENSITY 0.85 ;# percent
set FP_CORE2IO 21 ;# micron
24 # Power ring and settings
#
set PR_WIDTH 2.8 ;# micron
set PR_SPACING 1.68 ;# micron
set PR_LAYER_TB METAL9 ;# top and bottom layer
29 set PR_LAYER_LR METAL8 ;# left and right layer
# Power stripe settings
#
set ST_NUM_SETS 1 ;# number of sets
set ST_SPACING 1 ;# micron
34 set ST_LAYER_V $PR_LAYER_LR
set ST_WIDTH 1.4 ;# micron
set ST_XOFS_R 12 ;# micron
set ST_XOFS_L 0 ;# micron
# Placement settings
39 #
set PL_EFFORT -highEffort ;# -low | -medium | -high
# Clock tree synthesis settings
#
set CTS_BUFFER BUFCKX1 BUFCKX2 BUFCKX1P BUFCKX12 BUFCKX16 BUFCKX20
44 set CTS_INV INVCKX1 INVCKX2 INVCKX1P INVCKX12 INVCKX16 INVCKX20 INVCKX3
    INVCKX4 INVCKX6 INVCKX8
#
-----

# Flags that drive the script behavior (can be changed)
#
# ADD_STRIPE (0 | 1)
49 # if 1, add stripes
# PLACE_TIMING (0 | 1)
# if 1, do a timing driven placement
# CLOCK_TREE (0 | 1)
# if 1, create a clock tree
54 # CTS_CREATE_SPEC (0 | 1)
# if 1, create a clock tree specification file with default values
# ROUTE_TIMING (0 | 1)
# if 1, do a timing driven routing
# OPT (string)
59 # can be used to have different generated file names
#
-----

```

```

set ADD_STRIPEs 1
set PLACE_TIMING 1
set CLOCK_TREE 1
64 set CTS_CREATE_SPEC 0
set ROUTE_TIMING 1
set OPT "_cts"
#
-----

# File names
69 #
-----

set CONF_FILE_NAME ${DESIGN}.conf
set IO_FILE_NAME ${DESIGN}.io
set DESIGN_NAME ${DESIGN}${OPT}
set SAVE_DESIGN_FP_NAME ${DESIGN_NAME}-fplan.enc
74 set SAVE_DESIGN_PR_NAME ${DESIGN_NAME}-pring.enc
set SAVE_DESIGN_PL_NAME ${DESIGN_NAME}-placed.enc
set SAVE_DESIGN_PF_NAME ${DESIGN_NAME}-placed_filled.enc
set SAVE_DESIGN_CT_NAME ${DESIGN_NAME}-cts.enc
set SAVE_DESIGN_RO_NAME ${DESIGN_NAME}-routed.enc
79 set TIM_RCDB_NAME ${DESIGN_NAME}.rcdb
set SDF_FILE_NAME ${DESIGN_NAME}-routed.sdf
set SPEF_FILE_NAME ${DESIGN_NAME}-routed.spef
set RPT_CHECK_TA_NAME ${DESIGN_NAME}-checkta.rpt
set RPT_REPORT_TA_NAME ${DESIGN_NAME}-ta.rpt
84 set RPT_SLACK_NAME ${DESIGN_NAME}-slack.rpt
set RPT_GATE_COUNT_NAME ${DESIGN_NAME}-gate_count.rpt
set RPT_NOTCH_NAME ${DESIGN_NAME}-notch.rpt
set RPT_CONN_NAME ${DESIGN_NAME}-conn.rpt
set RPT_GEOM_NAME ${DESIGN_NAME}-geom.rpt
89 set RPT_DENSITY_NAME ${DESIGN_NAME}-density.rpt
set VLOG_NETLIST_SIM_NAME ${DESIGN_NAME}-routed.v
set VLOG_NETLIST_LVS_NAME ${DESIGN_NAME}-routed_lvs.v
set CTS_SPEC_NAME ${DESIGN_NAME}-spec.cts
set CTS_RGUIDE_NAME ${DESIGN_NAME}-guide.cts
94 set CTS_RPT_NAME ${DESIGN_NAME}-cts.rpt
set GDS_FILE_NAME ${DESIGN_NAME}.gds
#
-----

# Absolute paths
#
-----

99 set CONF_FILE ${PROJECT_DIR}/PAR/CONF/${CONF_FILE_NAME}
set IO_FILE ${PROJECT_DIR}/PAR/CONF/${IO_FILE_NAME}
set SAVE_DESIGN_FP_FILE ${PROJECT_DIR}/PAR/DB/${SAVE_DESIGN_FP_NAME}
set SAVE_DESIGN_PR_FILE ${PROJECT_DIR}/PAR/DB/${SAVE_DESIGN_PR_NAME}
set SAVE_DESIGN_PL_FILE ${PROJECT_DIR}/PAR/DB/${SAVE_DESIGN_PL_NAME}
104 set SAVE_DESIGN_PF_FILE ${PROJECT_DIR}/PAR/DB/${SAVE_DESIGN_PF_NAME}

```

```

set SAVE_DESIGN_CT_FILE ${PROJECT_DIR}/PAR/DB/${SAVE_DESIGN_CT_NAME}
set SAVE_DESIGN_RO_FILE ${PROJECT_DIR}/PAR/DB/${SAVE_DESIGN_RO_NAME}
set SDF_FILE ${PROJECT_DIR}/PAR/TIM/${SDF_FILE_NAME}
set SPEF_FILE ${PROJECT_DIR}/PAR/TIM/${SPEF_FILE_NAME}
109 set TIM_RCDB_FILE ${PROJECT_DIR}/PAR/TIM/${TIM_RCDB_NAME}
set RPT_CHECK_TA_FILE ${PROJECT_DIR}/PAR/RPT/${RPT_CHECK_TA_NAME}
set RPT_REPORT_TA_FILE ${PROJECT_DIR}/PAR/RPT/${RPT_REPORT_TA_NAME}
set RPT_SLACK_FILE ${PROJECT_DIR}/PAR/RPT/${RPT_SLACK_NAME}
set RPT_GATE_COUNT_FILE ${PROJECT_DIR}/PAR/RPT/${RPT_GATE_COUNT_NAME}
114 set RPT_NOTCH_FILE ${PROJECT_DIR}/PAR/RPT/${RPT_NOTCH_NAME}
set RPT_CONN_FILE ${PROJECT_DIR}/PAR/RPT/${RPT_CONN_NAME}
set RPT_GEOM_FILE ${PROJECT_DIR}/PAR/RPT/${RPT_GEOM_NAME}
set RPT_DENSITY_FILE ${PROJECT_DIR}/PAR/RPT/${RPT_DENSITY_NAME}
set VLOG_NETLIST_SIM_FILE ${PROJECT_DIR}/HDL/GATE/${VLOG_NETLIST_SIM_NAME}
}
119 set VLOG_NETLIST_LVS_FILE ${PROJECT_DIR}/HDL/GATE/${VLOG_NETLIST_LVS_NAME}
}
set CTS_SPEC_FILE ${PROJECT_DIR}/PAR/CTS/${CTS_SPEC_NAME}
set CTS_RGUIDE_FILE ${PROJECT_DIR}/PAR/CTS/${CTS_RGUIDE_NAME}
set CTS_RPT_FILE ${PROJECT_DIR}/PAR/RPT/${CTS_RPT_NAME}
set GDS_FILE ${PROJECT_DIR}/PAR/DEX/${GDS_FILE_NAME}
124 set GDS_MAP_FILE ${PROJECT_DIR}/PAR/DEX/gds2.map
#

```

```

# Procedures
#

```

```

# make_clock_tree
129 #
proc make_clock_tree create_spec {
global CTS_BUFFER CTS_INV CTS_SPEC_FILE CTS_RGUIDE_FILE CTS_RPT_FILE
if { $create_spec || ![file exists $CTS_SPEC_FILE] } {
createClockTreeSpec \
134 -bufFootprint $CTS_BUFFER \
-invfFootprint $CTS_INV \
-output $CTS_SPEC_FILE
}
specifyClockTree -clkfile $CTS_SPEC_FILE
139 ckSynthesis \
-rguide $CTS_RGUIDE_FILE \
-report $CTS_RPT_FILE
optDesign -postCTS -setup -drv -outDir PAR/RPT
} ;# make_clock_tree
144 #

```

```

# Load configuration file
#

```

```

loadConfig $CONF_FILE 0

```



```

commitConfig
149 #
-----

# Load IO file
#
-----

loadIoFile $IO_FILE
#
-----

154 # Set operating conditions
#
-----

setOpCond \
-maxLibrary $TIM_LIBRARY -max $TIM_OC_MAX \
-minLibrary $TIM_LIBRARY -min $TIM_OC_MIN
159 #
-----

# Set user grids
#
-----

setPreference ConstraintUserXGrid 0.1
setPreference ConstraintUserXOffset 0.1
164 setPreference ConstraintUserYGrid 0.1
setPreference ConstraintUserYOffset 0.1
setPreference SnapAllCorners 1
setPreference BlockSnapRule 2
#
-----

169 # Define global Power nets - make global connections
#
-----

clearGlobalNets
globalNetConnect VCC -type pgpin -pin VCC -inst * -module {} -verbose
globalNetConnect GND -type pgpin -pin GND -inst * -module {} -verbose
174 #
-----

# Initialize floorplan
#
-----

floorPlan -r $FP_ASPECT_RATIO \
$FP_ROW_DENSITY \
179 $FP_CORE2IO $FP_CORE2IO $FP_CORE2IO $FP_CORE2IO
fit

```

```

saveDesign $SAVE_DESIGN_FP_FILE
#
-----

# Add CAP cells
184 #
-----

#addEndCap -preCap ENDCAPL -postCap ENDCAPR -prefix ENDCAP
#
-----

# Create and route power rings and power stripes
#
-----

189 addRing \
    -around core \
    -nets { GND VCC } \
    -width_bottom $PR_WIDTH -width_top $PR_WIDTH \
    -width_left $PR_WIDTH -width_right $PR_WIDTH \
194 -spacing_bottom $PR_SPACING -spacing_top $PR_SPACING \
    -spacing_left $PR_SPACING -spacing_right $PR_SPACING \
    -layer_bottom $PR_LAYER_TB -layer_top $PR_LAYER_TB \
    -layer_left $PR_LAYER_LR -layer_right $PR_LAYER_LR \
    -center 1 \
199 -tl 1 -tr 1 -bl 1 -br 1 -lt 1 -lb 1 -rt 1 -rb 1 \
    -stacked_via_bottom_layer MET1 -stacked_via_top_layer MET9 \
    -threshold 0.7
    if { $ADD_STRIPE } {
        addStripe \
204 -nets { GND VCC } \
        -number_of_sets $ST_NUM_SETS \
        -spacing $ST_SPACING \
        -layer $ST_LAYER_V \
        -width $ST_WIDTH \
209 -xleft_offset $ST_XOFS_L
    }
    sroute \
    -jogControl { preferWithChanges differentLayer } \
    -nets { GND VCC }
214 saveDesign $SAVE_DESIGN_PR_FILE
#
-----

# Core cell placement
#
-----

if { $PLACE_TIMING } {
219 setPlaceMode $PL_EFFORT -timingdriven -doCongOpt -ignoreScan -ignoreSpare
    placeDesign
} else {

```

```

    setPlaceMode $PL_EFFORT -doCongOpt -ignoreScan -ignoreSpare
    placeDesign
224 }
    setDrawMode place
    saveDesign $SAVE_DESIGN_PL_FILE
    #
    -----

# Create clock tree (optional)
229 #
    -----

if { $CLOCK_TREE } {
    make_clock_tree $CTS_CREATE_SPEC
    saveDesign $SAVE_DESIGN_CT_FILE
}
234 #
    -----

# Add filler cells
#
    -----

addFiller -cell FILL25 FILL10 FILL5 FILL2 FILL1 -prefix FILLER
saveDesign $SAVE_DESIGN_PF_FILE
239 #
    -----

# Route design (Nanoroute)
#
    -----

if { $ROUTE_TIMING } {
    setNanoRouteMode -quiet -timingEngine CTE
244 setNanoRouteMode -quiet -routeWithTimingDriven true
    setNanoRouteMode -quiet -routeTdrEffort 0
}
    globalDetailRoute
    optDesign -postRoute -setup -drv -outDir PAR/RPT
249 saveDesign $SAVE_DESIGN_RO_FILE
    setDrawMode place
    #
    -----

# Verifications
#
    -----

254 fillNotch -report $RPT_NOTCH_FILE
    verifyConnectivity \
    -type all \
    -error 1000 \
    -warning 50 \

```

```

259 -report $RPT_CONN_FILE
    verifyGeometry \
    -allowSameCellViols \
    -allowRoutingBlkgPinOverlap \
    -allowRoutingCellBlkgOverlap \
264 -report $RPT_GEOM_FILE
    verifyMetalDensity \
    -detailed \
    -report $RPT_DENSITY_FILE
    #
    -----

269 # Extract parasitics
    #
    -----

    setExtractRCMode \
    -detail \
    -rcdb $TIM_RCDB_FILE \
274 -relative_c_t 0.01 \
    -total_c_t 5.0 \
    -reduce 5
    extractRC
    #
    -----

279 # Generate RC and timing files
    #
    -----

    rcOut -spef $SPEF_FILE
    delayCal -sdf $SDF_FILE
    #
    -----

284 # Generate reports
    #
    -----

    reportGateCount -outfile $RPT_GATE_COUNT_FILE
    # Timings
    #
289 setCteReport
    setAnalysisMode -setup -async -skew -noClockTree -sequentialConstProp
    reportAnalysisMode
    buildTimingGraph
    checkTA -verbose > $RPT_CHECK_TA_FILE
294 reportTA \
    -format { hpin arc cell delay arrival required slew fanout load } \
    -late \
    -max_points 10 \
    -net \
299 > $RPT_REPORT_TA_FILE

```

```

#
-----

# Save netlist
#
-----

saveNetlist -excludeLeafCell $VLOG_NETLIST_SIM_FILE
304 saveNetlist -physical $VLOG_NETLIST_LVS_FILE
#
-----

# Save GDS2
#
-----

streamOut $GDS_FILE \
309 -mapFile $GDS_MAP_FILE \
    -libName ADDSUB \
    -structureName $DESIGN_NAME \
    -stripes $ST_NUM_SETS \
    -units 1000 \
314 -mode ALL

```


Appendix A: Design Metrics

A design begins with the planning of the die. The mini@sic program from EuroPractice is a multiproject wafer where a minimum area of 1875 x 1875 μm^2 is required. This area constraint is the startpoint for planning the whole chip. Another important issue is the cell library metrics. The metrics of Faraday L90_SP library are:

Description	Width	Height
Grid	0.28 μm	0.28 μm
Core Cell	x * 0.28 μm	2.8 μm
IO Cell	60.48 μm	142.8 μm
Corner Cell	142.8 μm	142.8 μm
Pad Cell	64 μm	77 μm