Code ET4351

Digital Design Flow

Tutorial for EDA Tools:
  Synopsys Design Compiler
  Mentor Modelsim
  Cadence SOC Encounter

Ir. A.C. de Graaf
Dr. ir. T.G.R. van Leuken
Preface

This document describes the top-down design flow of the implementation a SoC design. Starting from an 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.
# Contents

1 Introduction  
1.1 Top-down design flow ............................................. 1  
1.2 Design project organisation ...................................... 3  
1.3 VHDL example: FIR-Filter ...................................... 5  
1.4 Design flow steps ................................................ 9  

2 VHDL and Verilog simulation  
2.1 Starting the Modelsim graphical environment ................. 11  
2.2 Simulation of (pre-synthesis) RTL VHDL models .......... 12  
2.3 Simulation of the post-synthesis Verilog model with timing data ............................................. 14  
2.4 Simulation of the post-route Verilog model with timing data .................................................. 15  

3 Logic synthesis  
3.1 Starting the Design Vision graphical environment .......... 17  
3.2 RTL VHDL model analysis ...................................... 18  
3.3 Design elaboration .............................................. 18  
3.4 Design environment definition ................................ 19  
3.5 Design constraint definitions .................................. 20  
3.6 Design mapping and optimization .............................. 22  
3.7 Report generation ................................................ 23  
3.8 VHDL/Verilog gate-level netlist generation and post-synthesis timing data (SDF) extraction .......... 28  
3.9 Design constraints generation for placement and routing .............. 29  
3.10 Design optimization with tighter constraints ............. 29  
3.11 Using scripts ................................................... 30  

4 Standard cell placement and routing  
4.1 Starting the Encounter graphical environment ............. 31  
4.2 Generate uniquified netlist ..................................... 33  
4.3 Design import ................................................... 33  
4.4 Floorplan Specification ....................................... 35  
4.5 Global net connections ........................................ 36  
4.6 Power ring/stripe creation and routing ....................... 37  
4.7 Operating conditions definition ............................... 39  
4.8 Core cell placement ......................................... 40  
4.9 Post-placement timing analysis .............................. 41  
4.10 Clock tree synthesis (optional) .............................. 43  
4.11 Filler cell placement ........................................ 45  
4.12 Design routing ............................................ 45
Chapter 1

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 v10.1 or higher, from Mentor Graphics.
- Design Compiler and Design Vision D-2010.03 or higher from Synopsys.
- Encounter 9.12 or higher from Cadence Design Systems.

The design kit used is from Faraday. The process is L90_SP, a 90 nm 9-metal CMOS process. 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 “1.4 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
CHAPTER 1.  INTRODUCTION

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 VHDL 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**
CHAPTER 1. INTRODUCTION

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:

```bash
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:

```bash
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:

```bash
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 FILTER that will be used to do the tasks presented in the rest of this document, execute the following command:

```bash
student@tango> create_eda_project FILTER
student@tango> cd FILTER
```

The FILTER 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. 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.
CHAPTER 1. INTRODUCTION

<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 VHDL example: FIR-Filter

The FIR-Filter example will be used as the reference design throughout the topdown flow. To install the VHDL model and its associated testbenches in the project directory, enter the following command in the top-level project directory FILTER:

```
student@tango-FILTER> install_design filter
```

In order to use the EDA tools and Faraday design kit, a script file called edadk.csh with the necessary PATHs to the tools exists in the directory from which the tools are launched (the top-level project directory).

Add these PATHs to your current environment by executing:

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

For information on the FARADAY design kits, find the FARADAY documentation in

```
/opt/eds/DesignLab/tech/Faraday/L90_SP/Docs
```

Listing 1.1 gives the entity declaration of the VHDL model of a generic 128 tap FIR-Filter. You will find the complete VHDL model in Appendix A.1-3. We will call this model the Core.

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

entity filter_soc is
generic (
    CWIDTH : integer := 16;
    CAW : integer := 7;
    DWIDTH : integer := 16;
    DAW : integer := 7
);
port (
    ClkxCI : in std_logic;
    ResetxRBI : in std_logic;
    DataInxDI : in std_logic_vector (DWIDTH-1 downto 0);
    DataInReqxSI : in std_logic;
    DataInAckxSO : out std_logic;
    DataOutxDO : out std_logic_vector (DWIDTH-1 downto 0);
    DataOutReqxSO : out std_logic;
    DataOutAckxSI : in std_logic
);
end filter_soc;
```
Figure 1.3 shows the hierarchy of the filter design. The chplevel entity named filter_soc consists of components IO-padcells and the toplevel filter design filter_top. Filter top consists of instances of the components coeff, dataRam and filter. The IO-padcells connect the port signals to the outside world. For this purpose we will use a technology independent library with generic components for IO cells defined in the library techmap. See Appendix B for the usage of IO-padcells.
CHAPTER 1. INTRODUCTION

Listing 1.2 gives the testbench for the RTL model (file filter_soc_tb.vhd). The testbench for the mapped netlist (file filter_soc_mapped_tb.vhd) can also be used for the simulation of the placed and routed Verilog netlist. The file filter_top.io defines the positions of the IO pins for place and route.

Listing 1.2: VHDL testbench for the RTL model with N=128.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_signed.ALL;
USE ieee.math_real.ALL;

ENTITY filter_soc_tb IS
  GENERIC(
    clock_delay : time := 16 ns;
    CWIDTH : integer := 16;
    CAW : integer := 7;
    DWIDTH : integer := 16;
    DAW : integer := 7
  );
END filter_soc_tb;

ARCHITECTURE behavioral OF filter_soc_tb IS

  component filter_soc is
    generic(
      CWIDTH : integer := 16;
      CAW : integer := 7;
      DWIDTH : integer := 32;
      DAW : integer := 7
    );
    port (
      ClkxCI : in std_logic;
      ResetxRBI : in std_logic;
      DataInxDI : in std_logic_vector (DWIDTH-1 downto 0);
      DataInReqxSI : in std_logic;
      DataInAckxSO : out std_logic;
      DataOutxDO : out std_logic_vector (DWIDTH-1 downto 0);
      DataOutReqxSO : out std_logic;
      DataOutAckxSI : in std_logic
    );
  end component;

  SIGNAL ClkxCI : std_logic;
  SIGNAL ResetxRBI : std_logic;
  SIGNAL DataInxDI : std_logic_vector (DWIDTH-1 downto 0);
  SIGNAL DataInReqxSI : std_logic;
  SIGNAL DataInAckxSO : std_logic;
  SIGNAL DataOutxDO : std_logic_vector (DWIDTH-1 downto 0);
  SIGNAL DataOutReqxSO : std_logic;
```
CHAPTER 1. INTRODUCTION

SIGNAL DataOutAckxSI : std_logic;
BEGIN
  -- Instantiate device-under-test.
dut: filter_soc
  generic map
    (CWIDTH => CWIDTH,
    CAW => CAW,
    DWIDTH => DWIDTH,
    DAW => DAW)
  port map
    (ClkxCI => ClkxCI,
    ResetxRBI => ResetxRBI,
    DataInxDI => DataInxDI,
    DataInReqxSI => DataInReqxSI,
    DataInAckxSO => DataInAckxSO,
    DataOutxDO => DataOutxDO,
    DataOutReqxSO => DataOutReqxSO,
    DataOutAckxSI => DataOutAckxSI
    );

clock_generation:
  PROCESS
    BEGIN
      -- Generate equal duty-cycle clock.
      ClkxCI <= '0';
      WAIT FOR (clock_delay / 2);
      ClkxCI <= '1';
      WAIT FOR (clock_delay / 2);
    END PROCESS clock_generation;

generate_stimulus:
  PROCESS
    BEGIN
      -- Initialize input signals.
      DataInReqxSI <= '0';
      DataInxDI <= (OTHERS => '0');
      DataOutAckxSI <= '0';

      -- Reset the design and wait for 2 clock cycles.
      ResetxRBI <= '0';
      WAIT FOR clock_delay * 2;
      ResetxRBI <= '1';

      -- Wait for 2 more clock cycles before beginning test.
      WAIT FOR clock_delay * 2;
      DataInxDI <= (0 => '1', OTHERS => '0');

      WAIT FOR clock_delay;
      FOR I IN 127 DOWNTO 0 LOOP
        DataInReqxSI <= '1';
      END LOOP;
      WHILE DataInAckxSO = '0' LOOP
        WAIT FOR clock_delay;
      END LOOP;
    END PROCESS generate_stimulus;
WHILE DataOutReqxSO = '0' LOOP
    WAIT FOR clock_delay;
END LOOP;

DataOutAckxSI <= '1';
WAIT FOR clock_delay;

DataOutAckxSI <= '0';
END LOOP;
--
-- Wait forever.
--
WAIT;
END PROCESS generate_stimulus;
END behavioral;

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

1.4 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
CHAPTER 1. INTRODUCTION

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

5.1 Generate a uniquified netlist (Convert Verilog mapped netlist) [??]
5.2 Design import (technological data + Verilog netlist) [4.3]
5.3 Floorplan specification [4.4]
5.4 Power ring/strip creation and routing [4.6]
5.5 Global net connections definition [4.5]
5.6 Operating conditions definition [4.7]
5.7 Core cell placement [4.8]
5.8 Post-placement timing analysis [4.9]
5.9 Clock tree synthesis (optional) [4.10]
5.10 Design routing [4.12]
5.11 Post-route timing optimization and analysis [4.13]
5.12 Filler cell placement [4.11]
5.13 Design checks [4.14]
5.14 Report generation [4.15]
5.15 Post-route timing data extraction [4.16]
5.16 Post-route netlist generation [4.17]
5.17 GDS2 file generation [4.18]

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-P&R gate-level netlist with PaR timing data
Chapter 2

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, create first a work library in the directory LIB/MSIM then enter in the vsim command in the Unix shell:

    student@tango-FILTER> vlib LIB/MSIM/work &
    student@tango-FILTER> vsim &

![Modelsim console window]

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
CHAPTER 2. VHDL AND VERILOG SIMULATION

the tool. After having started the vsim GUI you have to compile all the VHDL source of your design into the simulation library work. Now you have two options to simulate your model, either interactively or by executing the following convenience scripts (see directory SIM/BIN) on the VSIIM command line:

- `wave.do` To start a wave window with the ports of the top level module.
- `start_far[mapped | routed].do` To start up the simulation of the particular module.
- `run_far[mapped | routed].do` To load the filter coefficients into the coefficient ROM and runs the simulation for 300 us.

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 -quiet HDL/RTL/SYAA90_128X16X1CM2.vhd
ModelSim> vcom -quiet HDL/RTL/SPAA90_S12X16BM1A.vhd
ModelSim> vcom -quiet HDL/RTL/coeffFAR.vhd
ModelSim> vcom -quiet HDL/RTL/dataRamFAR.vhd
ModelSim> vcom -quiet HDL/RTL/filter.vhd
ModelSim> vcom -quiet HDL/RTL/filter_top.vhd
ModelSim> vcom -quiet HDL/RTL/filter_soc.vhd
ModelSim> vcom -quiet 1288,1074,1194 HDL/TBENCH/filter_soc_tb.vhd
```

or alternatively execute the shell script compile_msim_far.sh from the CSH command line:

```
student@tango-FILTER> sh compile_msim_far.sh
```

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/work. 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-FILTER> 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 in the Design Tab from library work 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 ⇒ Signals in Region**.

Note that the appropriate hierarchy level is selected in the simulation hierarchy window. Selecting another level, e.g. dut, 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.

Now first load the content coefficient file into the coefficient memory:

```
VSIM 7> run 35 ns
VSIM 7> do SIM/BIN/load.do
```

Then to start the simulation, it is either possible to enter run commands in the simulation console such as:

```
VSIM 7> run 300 us
```

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 clicking the Break icon in the main window or in the wave window.

Run the simulation interactively as described in the previous section or run the following scripts (See SIM/BIN) from the VSIM command line.

VSIM 7> do SIM/BIN/start_far.do
VSIM 8> do SIM/BIN/wave.do
VSIM 9> do SIM/BIN/run_far.do

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 Verilog 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 which includes the delay of the library cells only is stored in a SDF file. (See -3.8 VHDL/Verilog gate-level netlist generation and post-synthesis timing data (SDF) extraction.)

Compile the Verilog gate-level netlist generated by the logic synthesis and its testbench in a new library called mapped:

vlib LIB/MSIM/mapped
ModelSim> vcom -work LIB/MSIM/mapped HDL/RTL/SYAA90_128X16X1CM2.vhd
ModelSim> vcom -work LIB/MSIM/mapped HDL/RTL/SPAA90_512X16BM1A.vhd
ModelSim> vlog -work LIB/MSIM/mapped HDL/GATE/filter_soc_mapped.v
ModelSim> vcom -work LIB/MSIM/mapped HDL/TBENCH/filter_soc_tb_mapped.vhd

or execute the shell script compile_msim_far_mapped.sh.

To simulate the RTL model,

1. Select the main menu item Simulate ⇒ Start simulation... to get the simulation dialog window.
2. Select from library mapped the architecture of the testbench and a resolution of 100ps.
3. Click the Libraries tab to add the gate libraries fsd0a_a_generic_core and fod0a_b25_t25_generic_io.
4. Then click the **SDF** tab. In the SDF dialog window, add the file SYN/TIM/filter_soc_mapped.sdf and specify the region **dut**, 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 100ps to 1ns should be visible in the wave window.

Run the simulation interactively as described in the previous section or run the following scripts (See **SIM/BIN**) from the VSIM command line.

VSIM 7> do SIM/BIN/start_far_mapped.do  
VSIM 8> do SIM/BIN/wave.do  
VSIM 9> do SIM/BIN/run_far_mapped.do

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

This step occurs after the design has been placed and routed. The Post-route SDF-file contains cell delay and wire delays of the circuit. See "4.16 Post-route timing data extraction" and "4.17 Post-route netlist generation". This step involves the simulation of a Verilog gate-level netlist with a VHDL testbench. Com-
pile the Verilog gate-level netlist generated by the logic synthesis and its testbench in a new library called `routed`:

```bash
vlib LIB/MSIM/routed
ModelSim> vcom -work LIB/MSIM/routed HDL/RTL/SYAA90_128X16X1CM2.vhd
ModelSim> vcom -work LIB/MSIM/routed HDL/RTL/SPAA90_512X16BM1A.vhd
ModelSim> vlog -work LIB/MSIM/routed HDL/GATE/faraday-io.v
ModelSim> vlog -work LIB/MSIM/routed HDL/GATE/filter_soc-routed.v
ModelSim> vcom -work LIB/MSIM/routed HDL/TBENCH/filter_soc_tb_mapped.vhd
```

or execute the shell script `compile_msim_far_routed.sh`.

To simulate the placed and routed netlist with timing data:

1. Select the item `Simulate` ⇒ `Start simulation...` in the main menu to get the simulation dialog window.
2. Select from library `routed` the architecture of the testbench and a resolution of 100ps.
3. Then click the `Libraries` tab to add the gate libraries `fsd0a_a_generic_core` and `fod0a_b25_t25_generic_io`.
4. Load the SDF timing file `PAR/TIM/filter_soc-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.

Run the simulation interactively or run the following scripts (See `SIM/BIN`) from the VSIM command line.

```bash
VSIM 7> do SIM/BIN/start_far_routed.do
VSIM 8> do SIM/BIN/wave.do
VSIM 9> do SIM/BIN/run_far_routed.do
```
Chapter 3

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-FILTER> 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 CWIDTH, DWIDTH generic parameters to 16 and for the CAW, DAW parameters 7. Click OK. The console now displays the inferred registers and the kind of reset (here asynchronous reset - AR: Y).

Listing 3.1: “Elaborate design command”

```
design_vision> elaborate FILTER_SOC -architecture RTL -library WORK -parameters "CWIDTH = 16, CAW = 7, DWIDTH = 16, DAW = 7, CWIDTH = 16"
```

Inferred memory devices in process
in routine filter_DWIDTH16_DAW7 line 158 in file
'/mnt/tango/md2/users/sander/ET4351/asic/designs/filter1/HDL/RTL/filter.vhd'.
CHAPTER 3. LOGIC SYNTHESIS

Note the name filter_soc_CWIDTH16_CAW7_DWIDTH16_DAW7 given to the elaborated entity. It is possible to display the elaborated schematic by selecting the entity i_filter_top 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 WCCOM condition, which defines a temperature of 125˚C, a voltage of 0.9V (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_ss0p9v125c
```

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 `filter_soc_CWIDTH16_CAW7_DWIDTH16_DAW7` 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 }
```

To define an area constraint, select in the main menu the item `Attributes` ⇒ `Optimization Constraints` ⇒ `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
```
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 entity `filter_soc_CWIDTH16_CAW7_DWIDTH16_DAW7` in the hierarchy window and then the main menu item

**File ⇒ Save As....**

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

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 ../FILTER/SYN/DB/filter_soc_elab.ddc
```

To read back an elaborated design, select the main menu item **File ⇒ Read....** and then select the file to read.
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 instances of the coefficient rom, data ram and the filter circuit. Also, the cells are now real gates from the cell library.

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 filter_soc_CWIDTH16_CAW7_DWIDTH16_DAW7 in the hierarchy window and then the main menu item File ⇒ Save As.... Save the mapped design under the name filter_soc_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/filter_soc_mapped_area.rpt as well as in the report viewer. Click OK.

A new window and the console now display the report:

![Report window]

Information: Updating graph... (UID-83)

********************************************************************************
Report : area
Design : filter_soc
Version: D-2010.03-SP5-1
Date : Wed May 30 13:52:10 2012
********************************************************************************

Information: Updating design information... (UID-85)
Library(s) Used:

fsd0a_a_generic_core_ss0p9v125c (File: /opt/eds/DesignLab/tech/Faraday/L90_SP/Core/ fsd0a_a/2010Q4v2.1/GENERIC_CORE/FrontEnd/synopsys/synthesis/
 fsd0a_a_generic_core_ss0p9v125c.db)

fod0a_b25_t25_generic_io_ss0p9v125c (File: /opt/eds/DesignLab/tech/Faraday/L90_SP/IO
 /fod0a_b25/2009Q3v3.0/T25_GENERIC_IO/FrontEnd/synopsys/
fod0a_b25_t25_generic_io_ss0p9v125c.db)
CHAPTER 3. LOGIC SYNTHESIS

SYAA90_128X16X1CM2_BC (File: /opt/eds/DesignLab/tech/Faraday/L90_SP/Memory/ SYAA90_128X16X1CM2_BC.db)
SPAA90_512X16BM1A_BC (File: /opt/eds/DesignLab/tech/Faraday/L90_SP/Memory/ SPAA90_512X16BM1A_BC.db)

Number of ports: 38
Number of nets: 78
Number of cells: 97
Number of references: 24

Combinational area: 176044.062891
Noncombinational area: 190436.095396
Net Interconnect area: undefined (Wire load has zero net area)

Total cell area: 366480.158287
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/filter_soc_mapped_timing.rpt as well as in the report viewer. Click OK. A new window and the console now display the report:
CHAPTER 3. LOGIC SYNTHESIS

************************************************
Report : timing
-path full
-delay max
-max_paths 1
-sort_by group
Design : filter_soc
Version: D-2010.03-SP5-1
Date : Wed May 30 13:52:13 2012
************************************************

Operating Conditions: WCCOM   Library: fsd0a_a_generic_core_ss0p9v125c
Wire Load Model Mode: enclosed

Startpoint: i_filter_top/i_filter/StatexDP_reg[0]
    (rising edge-triggered flip-flop clocked by ClkxCI)
Endpoint: DataInAckxSO
    (output port clocked by ClkxCI)
Path Group: ClkxCI
Path Type: max

Des/Clust/Port       Wire Load Model Library
filter_soc          G5K   fsd0a_a_generic_core_ss0p9v125c
filter_DWIDTH16_DAW7 enG5K fsd0a_a_generic_core_ss0p9v125c

<table>
<thead>
<tr>
<th>Point</th>
<th>Incr</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock ClkxCI (rise edge)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>clock network delay (ideal)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>i_filter_top/i_filter/StatexDP_reg[0]/CK (DFFRBX1)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>i_filter_top/i_filter/StatexDP_reg[0]/QB (DFFRBX1)</td>
<td>0.35</td>
<td>0.35</td>
</tr>
<tr>
<td>i_filter_top/i_filter/U178/O (NR2X1)</td>
<td>0.69</td>
<td>1.03</td>
</tr>
<tr>
<td>i_filter_top/i_filter/DataInAckxSO (filter_DWIDTH16_DAW7)</td>
<td>0.00</td>
<td>1.03</td>
</tr>
<tr>
<td>io_DataInAckxSO/x0/op/O (VYA4GGSGB)</td>
<td>3.03</td>
<td>4.06</td>
</tr>
<tr>
<td>DataInAckxSO (out)</td>
<td>0.00</td>
<td>4.06</td>
</tr>
<tr>
<td>data arrival time</td>
<td></td>
<td>4.06</td>
</tr>
<tr>
<td>clock ClkxCI (rise edge)</td>
<td>8.00</td>
<td>8.00</td>
</tr>
<tr>
<td>clock network delay (ideal)</td>
<td>0.00</td>
<td>8.00</td>
</tr>
<tr>
<td>output external delay</td>
<td>-0.80</td>
<td>7.20</td>
</tr>
<tr>
<td>data required time</td>
<td></td>
<td>7.20</td>
</tr>
<tr>
<td>data required time</td>
<td></td>
<td>7.20</td>
</tr>
<tr>
<td>data arrival time</td>
<td></td>
<td>-4.06</td>
</tr>
<tr>
<td>slack (MET)</td>
<td></td>
<td>3.14</td>
</tr>
</tbody>
</table>

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 filter_soc_CWIDTH16_CAW7_DWIDTH16_DAW7 entity in the hierarchy window and then the Select menu item.
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 ⇒ Report Design Resources. Save the report in the file SYN/RPT/filter_soc_mapped_resources.rpt as well as in the report viewer.
Click OK.
As the report stated no special operator were found in our filter_soc design.
The last useful report is the one on power used. Here a estimate on power consumption is given based on switch activity and the library models used in the circuit.
To get a report on the power used, select the main menu item Design ⇒ Report Power. Save the report in the file SYN/RPT/filter_soc_mapped_power.rpt as well as in the report viewer.
Click OK.
CHAPTER 3. LOGIC SYNTHESIS

Load db file '/opt/eds/DesignLab/tech/Faraday/L90_SP/IO/fod0a_b25/2009Q2v3.0/T25_GENERIC_IO/FrontEnd/synopsys/fod0a_b25_t25_generic_io_ss0p9v125c.db'
Load db file '/opt/eds/DesignLab/tech/Faraday/L90_SP/Core/fsd0a_a/2010Q4v2.1/GENERIC_CORE/FrontEnd/synopsys/synthesis/fsd0a_a_generic_core_ss0p9v125c.db'
Load db file '/opt/eds/DesignLab/tech/Faraday/L90_SP/IO/fod0a_b25/2009Q2v3.0/T25_GENERIC_IO/FrontEnd/synopsys/fod0a_b25_t25_generic_io_ss0p9v125c.db'
Load db file '/opt/eds/DesignLab/tech/Faraday/L90_SP/Memory/SYAA90_512X16BM1A_BC.db'
Load db file '/opt/eds/DesignLab/tech/Faraday/L90_SP/Memory/SPAA90_512X16BM1A_BC.db'
Information: Propagating switching activity (low effort zero delay simulation). (PWR-6)
Warning: Design has unannotated primary inputs. (PWR-414)
Warning: Design has unannotated sequential cell outputs. (PWR-415)
Warning: Design has unannotated black box outputs. (PWR-428)

************************************************************
Report: power
-analysis_effort low
Design: filter_soc
Version: D-2010.03-SP5-1
Date: Wed May 30 13:52:13 2012
************************************************************

Library(s) Used:

- fsd0a_a_generic_core_ss0p9v125c (File: /opt/eds/DesignLab/tech/Faraday/L90_SP/Core/fsd0a_a/2010Q4v2.1/GENERIC_CORE/FrontEnd/synopsys/synthesis/fsd0a_a_generic_core_ss0p9v125c.db)
- fod0a_b25_t25_generic_io_ss0p9v125c (File: /opt/eds/DesignLab/tech/Faraday/L90_SP/IO/fsd0a_b25/2009Q2v3.0/T25_GENERIC_IO/FrontEnd/synopsys/fod0a_b25_t25_generic_io_ss0p9v125c.db)
- SYAA90_128X16X1CM2_BC (File: /opt/eds/DesignLab/tech/Faraday/L90_SP/Memory/SYAA90_128X16X1CM2_BC.db)
- SPAA90_512X16BM1A_BC (File: /opt/eds/DesignLab/tech/Faraday/L90_SP/Memory/SPAA90_512X16BM1A_BC.db)

Operating Conditions: WCCL
Wire Load Model Mode: enclosed

<table>
<thead>
<tr>
<th>Design</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter_top_CWIDTH16_CAW7_DWIDTH16_DAW7</td>
<td>enG30K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
</tr>
<tr>
<td>faraday_vcccopad_limit1 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>faraday_gndcopad_limit1 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>filter_DWIDTH16_DAW7 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>faraday_gndcopad_limit1_6 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>faraday_gndcopad_limit1_5 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>faraday_gndcopad_limit1_4 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>faraday_gndcopad_limit1_3 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>faraday_gndcopad_limit1_2 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>faraday_gndcopad_limit1_1 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>faraday_vcccopad_limit1_0 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>faraday_vcccopad_limit1_1 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>faraday_vcccopad_limit1_2 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>faraday_vcccopad_limit1_3 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>faraday_vcccopad_limit1_4 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>faraday_vcccopad_limit1_5 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>faraday_vcccopad_limit1_6 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>faraday_vcccopad_limit1_7 enG5K</td>
<td>fsd0a_a_generic_core_ss0p9v125c</td>
<td></td>
</tr>
<tr>
<td>SYAA90_128X16X1CM2_BC</td>
<td>SPAA90_512X16BM1A_BC</td>
<td></td>
</tr>
</tbody>
</table>

Global Operating Voltage = 0.9
Power-specific unit information:
Voltage Units = 1V
Capacitance Units = 1.000000 pf
Time Units = 1ns
Dynamic Power Units = mW (derived from V.C.T units)
Leakage Power Units = pW

| Cell Internal Power | 1.4108 mW (95%) |
| Net Switching Power | 79.1487 uW (5%)  |
| Total Dynamic Power | 1.4899 mW (100%) |
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 filter_soc_CWIDTH16_CAW7_DWIDTH16_DAW7 is selected in the hierarchy window):

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

Save the mapped design in the file `filter_soc_mapped.vhd` 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 .../FILTER/HDL/GATE/filter_mapped.vhd
```

To generate the SDF file, enter the following command in the console

```
write_sdf -version 2.1 SYN/TIM/filter_soc_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/DB/filter_soc_mapped.ddc`, and entering the following command in the console (be sure that the entity filter_soc_CWIDTH16_CAW7_DWIDTH16_DAW7 is selected in the hierarchy window):

```
change_names -hierarchy -rules verilog -verbose
```

Save the mapped design in the file `filter_soc_mapped.v` in the directory HDL/GATE. Click Save. The console now echoes the equivalent command line:

```
write -hierarchy -format verilog -output .../FILTER/HDL/GATE/filter_soc_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/filter_soc_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 1 ns period.
3. Save the new elaborated entity in the file SYN/DB/filter_soc_CWIDTH16_CAW7_DWIDTH16_DAW7_1ns_elab.db.
4. Map and optimize the design.
5. Save the mapped design in the file SYN/DB/filter_soc_1ns_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/filter_soc_1ns_mapped.vhdl
   ```
   and the associated SDF timing data file in
   
   ```
   SYN/TIM/filter_soc_1ns_mapped.sdf
   ```
8. Do a post-synthesis simulation.
9. Generate the Verilog gate-level netlist in HDL/GATE/filter_soc_1ns_mapped.v.
10. Save the design constraints for placement and routing in the file
    
    ```
    SYN/SDC/filter_soc_1ns_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 filter_soc design has been installed in the SYN/BIN directory (see “1.3 VHDL example: FIR-Filter”). 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-FILTER> dc_shell -f SYN/BIN/filter_soc_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.
Chapter 4

Standard cell placement and routing

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 (Velocity).

4.1 Starting the Encounter graphical environment

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

```
student@tango-FILTER> velocity -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 velocity> 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 submodules 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.
CHAPTER 4. STANDARD CELL PLACEMENT AND ROUTING

4.2 Generate uniquified netlist

Within your synthesized netlist, there are several instances of the same modules. You might realize that these modules should have unique names and currently they do not. You will fix this using an Encounter command to make the instances unique:

```
student@tango-FILTER> uniquifyNetlist
```

4.3 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 `File ⇒ Import Design...` 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/filter_soc_unique.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 8 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/filter_soc_mapped.sdc`. Only timing information in the constraint file is actually used by Encounter.

In the IO Assignment File Field:

Select the `PAR/CONF/filter_soc.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/filter_soc.conf by clicking on the Save... button.
Finally, click on OK. The configuration is then read in.
To reload a configuration, select File ⇒ Import Design... in the main menu. Then, click on the Load...
button and load the configuration file from the PAR/CONF directory.

4.4 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 ⇒ Specify Floorplan... in the main menu.
In order to be able to add PAD cells and IO cells we need to specify Die Size and the Core to IO Boundary
parameters. Click the button at Die Size by: and fill in 1872.02 µm for both Width and Height.
At "Core Margins by:" click the button Core to IO Boundary and fill in the distance between the I/O
boundary and the core, here 560iµm at Core to Left, Top, Right, Bottom. 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 D.
Select Floorplan ⇒ Automatic Floorplan ⇒ Plan Design... in the main menu to execute the floorplanning. The memory macro’s are placed into the core area. You can move the memory cells if the initial
placement does not suit you.
Additionally, the command Floorplan ⇒ Clear Floorplan... allows you to delete all or parts of the floor-
plan objects.
With Floorplan ⇒ Automatic Floorplan ⇒ Finish Floorplan... halo’s around the macro cell can be
specified to prevent placement of standardcells to close to the macro cells. Specify here a halo width of 28
um and leave everything else default.
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:
velocity> 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 **File ⇒ Save Design...** in the main menu and save the current state in the file PAR/DB/filter_soc-fplan.enc. The data are actually saved in the directory PAR/DB/filter_soc-fplan.enc.dat.

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

### 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).
CHAPTER 4. STANDARD CELL PLACEMENT AND ROUTING

37

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.
Repeat these actions with Tie High checked and finally with Tie Low checked.
Click on Apply and then on Close.

4.6  Power ring/stripes creation and routing

This step generates the VCC and GND 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 Power ⇒ Power Planning ⇒ 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.
In the Ring Type field check Core ring(s) contouring. The Ring Configuration field should define ring widths of 2.8 micron spaced by 1.12 micron.
The rings will be placed either in the center of the channel between the core and the chip boundary (or the IO pads, if any) or at a particular offset from the core. Check the Specify in the Ring Configuration field and specify an offset to the core of 28 µm.
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 block rings around the memory blocks, select Power ⇒ Power Planning ⇒ Add Rings... in the main menu.
In the **Ring Type** field check **Block ring(s) around**. The **Ring Configuration** field should define ring widths of 1.12 micron spaced by 0.56 micron. Check the Specify in the **Ring Configuration** field and specify an offset to the block of 5.6 \( \mu \)m.

Click **OK** to generate the block rings.

It is possible to measure sizes by using the ruler (hit the `k` binding key). Hit `K` to remove the last ruler or press ESC to remove all rulers.

To remove ring segments select them and hit the Delete key.

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

The **Set Configuration** area defines the Net(s) pattern, direction, layer, width and spacing of the stripes. Our example does not need any stripes so click **Cancel**
Now, it is possible to route the power grid. Select **Route ⇒ SRoute...** in the main menu. All default values are fine. Click **OK** to do the routing. The design now looks like below:

![Design Image]

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

### 4.7 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 **Options ⇒ Set Mode ⇒ Specify Operating Condition/PVT...** in the main menu.

In the max tab, select the WCCOM operating condition for Timing Library: fsd0a_a_generic_core_ss0p9v125c. In the min tab, select the BCCOM operating condition for Timing Library: fsd0a_a_generic_core_ff1p1vm40c. 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 to ‘fsd0a_a_generic_core_ff1p1vm40c/%NOM_PVT’ defined in Timing Library ‘fsd0a_a_generic_core_ff1p1vm40c’ Process: 1.00 Temperature: -40.000 Voltage: 1.100
Set Max operating condition to ‘fsd0a_a_generic_core_ss0p9v125c/%NOM_PVT’ defined in Timing Library ‘fsd0a_a_generic_core_ss0p9v125c’ Process: 1.00 Temperature: 125.000 Voltage: 0.900

*** Calculating scaling factor for min libraries using operating condition:
Name: fsd0a_a_generic_core_ff1p1vm40c/%NOM_PVT Process: 1.00 Temperature: -40.000 Voltage: 1.100

*** Calculating scaling factor for max libraries using operating condition:
Name: fsd0a_a_generic_core_ss0p9v125c/%NOM_PVT Process: 1.00 Temperature: 125.000 Voltage: 0.900

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

```
velocity 1> getOpCond -v
min: fsd0a_a_generic_core_ff1p1vm40c/%NOM_PVT proc: 1.0000 volt: 1.1000 temp: -40.0000
max: fsd0a_a_generic_core_ss0p9v125c/%NOM_PVT proc: 1.0000 volt: 0.9000 temp: 125.0000
```

4.8 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 Velocity 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 File ⇒ Save Design... in the main menu and save the current state in the file PAR/DB/filter_soc-placed.enc.
4.9 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 ⇒ Report 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 by: Cadence Encounter 09.12-s139_1
# OS: Linux (Host ID salsa)
# Generated on: Wed May 30 11:14:18 2012
# Command: timedesign -preCTS -idealClock -pathReports -drvReport...
```

```
timedesign Summary

<table>
<thead>
<tr>
<th>Setup mode</th>
<th>all</th>
<th>reg2reg</th>
<th>in2reg</th>
<th>reg2out</th>
<th>in2out</th>
<th>clkgate</th>
</tr>
</thead>
<tbody>
<tr>
<td>WNS (ns):</td>
<td>2.082</td>
<td>3.385</td>
<td>5.841</td>
<td>2.082</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TNS (ns):</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Violating Paths:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>All Paths:</td>
<td>200</td>
<td>102</td>
<td>98</td>
<td>18</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DRVsv</th>
<th>Real</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nr nets (terms)</td>
<td>Worst Vio</td>
<td>Nr nets (terms)</td>
</tr>
</tbody>
</table>
```
The design is not critical as the slack is positive (2.082 ns).

To get more details on the critical path execute the following commands in the Encounter console:

velocity 16> report_timing

The following report is then displayed in the console:

Path 1: MET Late External Delay Assertion

<table>
<thead>
<tr>
<th>Distance</th>
<th>Arc</th>
<th>Cell</th>
<th>Delay</th>
<th>Arrival</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>i_filter_top/i_filter/StateDP_reg_0/QB (v) triggered by leading edge of 'ClkCI'</td>
<td>OK ^ &gt; QB v</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i_filter_top/i_filter/StateDP_reg_0_1</td>
<td>OK ^ &gt; QB v</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>io_DataInAckxSO_x0_op</td>
<td>1</td>
<td>VYA4GSGB</td>
<td>3.405</td>
<td>5.118</td>
<td>7.200</td>
</tr>
</tbody>
</table>

Density: 10.698%
Routing Overflow: 0.00% and 0.00%
If timing requirements are not met optimization is possible by selecting Optimize ⇒ Optimize Design... in the main menu and check pre-CTS and click OK.

4.10 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. Here we will use a ready made specification file PAR/CTS/filter_soc-spec.ctstch Select Clock ⇒ Synthesize Clock Tree... in the main menu. In the popup menu change the Results Directory to PAR/CTS/clock_report at Clock Specification Files: on the ... button to add the clock specification file PAR/CTS/filter_soc-spec.ctstch.

![Clock Tree Specification File](image)

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 velocity prompt.

```
velocity 1> reportClockTree -report PAR/RPT/filter_soc.ctsrpt
```

The following report is also displayed in the Encounter console:

```
reportClockTree Option :
*** Look For Reconvergent Clock Component ***
The clock tree io_ClkxCI_x0_ip/O has no reconvergent cell.
#
# Mode Name  : Setup
# Library Name : fsd0a_a_generic_core_ss0p9v125
********** Clock io_ClkxCI_x0_ip/O Pre-Route Timing Analysis **********
Nr. of Subtrees : 1
Nr. of Sinks : 66
Nr. of Buffer : 1
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): i_filter_top / i_filter / AccuxDP_reg_15_ / CK 385.4(ps)
Min trig. edge delay at sink(R): i_filter_top / i_filter / OutRegxDP_reg_1_ / CK 364.5(ps)
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Actual</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rise Phase Delay</td>
<td>364.3–385.4(ps)</td>
<td>0–8000(ps)</td>
</tr>
<tr>
<td>Fall Phase Delay</td>
<td>302.1–323(ps)</td>
<td>0–8000(ps)</td>
</tr>
<tr>
<td>Trig. Edge Skew</td>
<td>20.9(ps)</td>
<td>300(ps)</td>
</tr>
<tr>
<td>Rise Skew</td>
<td>20.9(ps)</td>
<td></td>
</tr>
<tr>
<td>Fall Skew</td>
<td>20.9(ps)</td>
<td></td>
</tr>
<tr>
<td>Max. Rise Buffer Tran.</td>
<td>182.5(ps)</td>
<td>400(ps)</td>
</tr>
<tr>
<td>Max. Fall Buffer Tran.</td>
<td>156.6(ps)</td>
<td>400(ps)</td>
</tr>
<tr>
<td>Max. Rise Sink Tran.</td>
<td>170.2(ps)</td>
<td>400(ps)</td>
</tr>
<tr>
<td>Max. Fall Sink Tran.</td>
<td>76.4(ps)</td>
<td>400(ps)</td>
</tr>
<tr>
<td>Min. Rise Buffer Tran.</td>
<td>182.5(ps)</td>
<td>0(ps)</td>
</tr>
<tr>
<td>Min. Fall Buffer Tran.</td>
<td>156.6(ps)</td>
<td>0(ps)</td>
</tr>
<tr>
<td>Min. Rise Sink Tran.</td>
<td>170.2(ps)</td>
<td>0(ps)</td>
</tr>
<tr>
<td>Min. Fall Sink Tran.</td>
<td>76.2(ps)</td>
<td>0(ps)</td>
</tr>
</tbody>
</table>

Several clock report files are also available in the PAR/CTS/clock_report directory. It is recommended to save the new stage of the design. Select File ⇒ Save Design... in the main menu and save the current state in the file PAR/DB/filter_soc-cts.enc.
4.11 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 \texttt{Place $\Rightarrow$ Physical Cell $\Rightarrow$ Add Filler...} in the main menu.

Select the cells FILLER64E, FILLER32E, FILLER16E, FILLER8E, FILLER4E, FILLER3, FILLER2 and FILLER1 and click \texttt{OK} to place the filler cells.

Another way to add the filler cells is by executing the tcl script "fillcore.tcl":

\begin{verbatim}
velocity 1> source PAR/BIN/fillcore.tcl
\end{verbatim}

4.12 Design routing

This step generates all the wires required to connect the cells according to the imported gate-level netlist. To route the design, select \texttt{Route $\Rightarrow$ Nanoroute...} in the main menu, check the Timing Driven box and a maximum effort. Click \texttt{OK} to do the routing.

You now get the routed design:
CHAPTER 4. STANDARD CELL PLACEMENT AND ROUTING

4.13 Post-routing timing optimization and analysis

A final timing optimization may be done on the routed design. Select Optimize ⇒ Optimize Design... in the main menu. Select the postRoute box. Click OK.

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

4.14 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/filter_soc-conn.rpt. Click OK.

The console displays the results:

******** Start: VERIFY CONNECTIVITY ********
Design Name: filter_soc
Database Units: 1000
Design Boundary: (0.0000, 0.0000) (1872.0800, 1872.0800)
Error Limit = 1000; Warning Limit = 50
Check all nets
VC Elapsed Time: 0:00:00.0
Begin Summary
  Found no problems or warnings.
End Summary
******** End: VERIFY CONNECTIVITY ********
Verification Complete : 0 Viols. 0 Wrngs.
(CPU Time: 0:00:00.1 MEM: 0.008M)

Select Verify ⇒ Verify Geometry... in the main menu. In the Advanced tab, define the report file as PAR/RPT/filter_soc-geom.rpt.
Click OK.
The console displays the results:

```
VERIFY GEOMETRY ...... Sub-Area : 15 complete 0 Viols. 0 Wrngs.
VERIFY GEOMETRY ...... SubArea : 16 of 16
VERIFY GEOMETRY ...... Cells : 0 Viols.
VERIFY GEOMETRY ...... SameNet : 0 Viols.
VERIFY GEOMETRY ...... Wiring : 0 Viols.
VERIFY GEOMETRY ...... Antenna : 0 Viols.
VERIFY GEOMETRY ...... Sub-Area : 16 complete 0 Viols. 0 Wrngs.
VG: elapsed time: 5.00
Begin Summary ...
Cells : 0
SameNet : 0
Wiring : 0
Antenna : 0
Short : 0
Overlap : 0
End Summary

 Verification Complete : 0 Viols. 0 Wrngs.

**********End: VERIFY GEOMETRY**********
*** verify geometry (CPU: 0:00:04.4 MEM: 3.8M)```

### 4.15 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:

**File ⇒ Report⇒ Netlist Statistics** gives the following output in the console:

```
velocity 15> *** Statistics for net list filter_soc ***
Number of cells = 2949
Number of nets = 689
```
CHAPTER 4. STANDARD CELL PLACEMENT AND ROUTING

Number of tri-nets = 18
Number of degen nets = 0
Number of pins = 2411
Number of I/Os = 38

Number of nets with 2 terms = 419 (60.8%)
Number of nets with 3 terms = 147 (21.3%)
Number of nets with 4 terms = 38 (5.5%)
Number of nets with 5 terms = 18 (2.6%)
Number of nets with 6 terms = 12 (1.7%)
Number of nets with 7 terms = 8 (1.2%)
Number of nets with 8 terms = 4 (0.6%)
Number of nets with 9 terms = 6 (0.9%)
Number of nets with >=10 terms = 37 (5.4%)

*** 69 Primitives used:
Primitive PAD9MI26G (56 insts)
Primitive VCCKGB (1 insts)
Primitive GNDKGB (1 insts)
Primitive VCC2IOGB (8 insts)
Primitive GND2IOGB (8 insts)
Primitive SPAA90_512X16M1A (1 insts)
Primitive UYNGB (20 insts)
Primitive VYAA42GB (18 insts)
Primitive SYAA90_128X16X1CM2 (1 insts)
Primitive EMPTY8GB (32 insts)
Primitive EMPTY4GB (32 insts)
Primitive EMPTY2GB (34 insts)
Primitive EMPTY1GB (40 insts)
Primitive EMPTY16GB (492 insts)
Primitive CORNERGB (4 insts)
Primitive XOR3X1 (4 insts)
Primitive XOR2X1 (8 insts)
Primitive XOR2CX1 (1 insts)
Primitive TIE1X1 (2 insts)
Primitive TIE3X1 (2 insts)
Primitive QDFERBX1 (22 insts)
Primitive OR2X1 (1 insts)
Primitive OA122XLP (1 insts)
Primitive OA112XLP (2 insts)
Primitive OA112X1 (15 insts)
Primitive OA112X1 (7 insts)
Primitive OA112XLP (3 insts)
Primitive OA112X1 (1 insts)
Primitive OA12XLP (2 insts)
CHAPTER 4. STANDARD CELL PLACEMENT AND ROUTING

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

velocity 15> Gate area 2.3520 um^2
[0] filter_soc Gates=16332 Cells=504 Area=38414.3 um^2
[1] i_filter_top Gates=16314 Cells=497 Area=38371.1 um^2
[2] i_filter_top/i_filter Gates=1855 Cells=493 Area=4363.7 um^2
[2] i_filter_top/i_coeff_a9d16 Gates=6995 Cells=0 Area=16453.7 um^2
[2] i_filter_top/i_dataRAM_i_dmem Gates=7461 Cells=0 Area=17549.0 um^2

Finally, File ⇒ Report ⇒ Summary... displays the following window:

*** Statistics for net list filter_soc ***
Number of cells = 2949
Number of nets = 689
Number of tri-nets = 18
Number of degen nets = 0
Number of pins = 2411
Number of i/o's = 38

Number of nets with 2 terms = 419 (60.8%)
Number of nets with 3 terms = 147 (21.3%)
Number of nets with 4 terms = 38 (5.5%)
Number of nets with 5 terms = 18 (2.6%)
Number of nets with 6 terms = 12 (1.7%)
Number of nets with 7 terms = 8 (1.2%)
Number of nets with 8 terms = 4 (0.6%)
Number of nets with 9 terms = 6 (0.9%)
Number of nets with >=10 terms = 37 (5.4%)

*** 69 Primitives used:
Primitive PAD9M126G (56 insts)
Primitive VCCKGB (1 inst)
Primitive GDKGGB (1 inst)
Primitive VCC2IOGB (8 insts)
Primitive GND2IOGB (8 insts)
Primitive SPAA90_512X16BM1A (1 insts)
Primitive UYNGB (20 insts)
Primitive VTAA4GB (18 insts)
Primitive SYAA90_128X16X1CM2 (1 insts)
Primitive EMPTY8GB (32 insts)
Primitive EMPTY4GB (32 insts)
Primitive EMPTY2GB (34 insts)
Primitive EMPTY1GB (40 insts)
Primitive EMPTY16GB (492 insts)
Primitive CORNERGB (4 insts)
Primitive XOR3X1 (4 insts)
Primitive XOR2X1 (8 insts)
Primitive XOR2CKX1 (1 insts)
4.16 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. Therefore set the extraction mode:
Select Options ⇒ Set Mode ⇒ Specify RC Extraction Mode... in the main menu.

Check PostRoute, EffortLevel: Low, Extraction Type: Coupled RC
And extract the netlist:
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 ⇒ Write SDF... in the main menu. The checked Ideal Clock switch means that flip-flops are considered as having 0ps rising and falling transition times.
4.17 Post-route netlist generation

This steps 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 File ⇒ 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.

4.18 GDS2 file generation

The placed and routed design can be exported in different formats for further processing outside the En-
counter 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 **File ⇒ Save ⇒ GDS/OASIS...** in the main menu. The GDS map file has been copied by the tech_setup script into the PAR/DEX directory. The generated GDS2 file is written in the same directory.

### 4.19 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 the Tcl language for building scripts.

An example of such a script for placement and routing of the Fir-Filter design has been installed in the PAR/BIN directory (see "? VHDL example: FIR-Filter"). 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:

```
velocity -log PAR/LOG/encounter -overwrite -init PAR/BIN/top-level.tcl -win
```

The script top-level.tcl given below calls TCL subscripts, one for each design step. During an interactive session of Encounter you can source the scripts in the proper sequence at the velocity command prompt. This allows for checking the result after each Place&Route step. By modifying parameters in a particular design step script you can 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/filter_soc.io file.

```tcl
set PROJECT_DIR [pwd]
set PAR_BIN [${PROJECT_DIR}]/PAR/BIN
# Import the design
source [${PAR_BIN}]/start.tcl
# Create a floorplan
source [${PAR_BIN}]/fplan.tcl
# Create power/ground distribution lines
source [${PAR_BIN}]/pplan.tcl
# Place standard cells into the core
source [${PAR_BIN}]/place.tcl
# Create the Clock Tree
source [${PAR_BIN}]/cts.tcl
# Place Filler Cells and route the design
source [${PAR_BIN}]/route.tcl
# Verify Connectivity and Geometry
source [${PAR_BIN}]/verify.tcl
# Extract the circuit
source [${PAR_BIN}]/results.tcl
```
Appendix A

Appendix A: VHDL Netlists

A.1 File: filter.vhd

Listing A.1: RTL synthesisable model of a 128 tap FIR-Filter.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity filter is
generic (  
  CWIDTH : integer := 16;
  CAW : integer := 7;
  DWIDTH : integer := 32;
  DAW : integer := 7
);
port (  
  ClkxCI : in std_logic;
  ResetxRBI : in std_logic;
  DataInxDI : in std_logic_vector (DWIDTH-1 downto 0);
  DataInReqxSI : in std_logic;
  DataInAckxSO : out std_logic;
  DataOutxDO : out std_logic_vector (DWIDTH-1 downto 0);
  DataOutReqxSO : out std_logic;
  DataOutAckxSI : in std_logic;
  LutAddrxDO : out std_logic_vector (CAW-1 downto 0);
  LutReadxDI : in std_logic_vector (CWIDTH-1 downto 0);
  RamWriteEnxSO : out std_logic;
  RamAddrxDO : out std_logic_vector (DAW-1 downto 0);
  RamReadxDI : in std_logic_vector (DWIDTH-1 downto 0);
  RamWritexDO : out std_logic_vector (DWIDTH-1 downto 0)
);  
end filter;

architecture rtl of filter is
  -- component declarations
  -- component declarations
  type state_type is (idle, new_data, run, data_out);
  signal StatexDP, StatexDN : state_type;
  -- Registers
  signal RamWriteEnxS : std_logic;
```
APPENDIX A. APPENDIX A: VHDL NETLISTS

```vhdl
signal InRegEnxS : std_logic;
signal InRegxDN : std_logic_vector(DWIDTH-1 downto 0);

-- Counters
signal OffsetDecxS : std_logic;
signal OffsetxDP, OffsetxDN : unsigned(DAW-1 downto 0);
signal CounterIncxS : std_logic;
signal CounterxDP, CounterxDN : unsigned(CAW-1 downto 0);

-- Counters
signal RamAddrxD : std_logic_vector(DAW-1 downto 0);
signal LutAddrxD : std_logic_vector(CAW-1 downto 0);

-- ALU signals
signal SumxD : signed((DWIDTH+CWIDTH)-1 downto 0);
signal SumStdxD : std_logic_vector((DWIDTH+CWIDTH)-1 downto 0);
signal RamReadxD : std_logic_vector(DWIDTH-1 downto 0);
signal RamSignedxD : signed(DWIDTH-1 downto 0);
signal LutReadxD : std_logic_vector(CWIDTH-1 downto 0);
signal LutSignedxD : signed(CWIDTH-1 downto 0);
signal MultxD : signed((DWIDTH+CWIDTH)-1 downto 0);

-- Registers
signal AccuClrxS : std_logic;
signal AccuxDP, AccuxDN : signed((DWIDTH+CWIDTH)-1 downto 0);

-- Registers
signal OutRegEnxS : std_logic;
signal OutRegxDN : std_logic_vector(DWIDTH-1 downto 0);
begin
--RamWriteEnxSO <= RamWriteEnxS;
------------------------------------------------------------
-- Input register
------------------------------------------------------------
begin
if ResetxRBI='0' then
  RamWritexDO <= (others => '0');
elsif ClkxCI'event and ClkxCI='1' then
  if InRegEnxS = '1' then
    RamWritexDO <= InRegxDN;
  else
    end if;
end if;
end process p_inreg;

------------------------------------------------------------
-- Counters
------------------------------------------------------------
begin
OffsetxDN <= OffsetxDN;
if OffsetDecxS = '1' then
  OffsetxDN <= OffsetxDN - "0000001";
end if;
end process p_data;

begin
CounterxDN <= CounterxDN;
if CounterIncxS = '1' then
```
APPENDIX A. VHDL NETLISTS

```vhdl
-- Counters
------------------------------------------------------------
-- two similar registers combined in one process
p_adrclk: process (ClkxCI, ResetxRBI)
begin
if ResetxRBI='0' then
  CounterxDP <= (others => '0');
  OffsetxDP <= (others => '0');
elsif ClkxCI'event and ClkxCI='1' then
  CounterxDP <= CounterxDN;
  OffsetxDP <= OffsetxDN;
end if;
end process p_adrclk;

-- add and convert the address
RamAddrxDO <= std_logic_vector ( OffsetxDP + CounterxDP );
LutAddrxDO <= std_logic_vector ( CounterxDP );

------------------------------------------------------
-- ALU
------------------------------------------------------
-- type conversion
RamSignedxD <= signed ( RamReadxDI );
LutSignedxD <= signed ( LutReadxDI );
-- signed operations
MultxD <= RamSignedxD * LutSignedxD;
SumxD <= MultxD + AccuxDP;
-- type conversion
SumStdxD <= std_logic_vector (SumxD);
-- simple truncate
OutRegxDN<= SumStdxD((DWIDTH+CWIDTH)-1 downto CWIDTH);
OutRegxDN <= SumStdxD (DWIDTH -1 downto 0);

------------------------------------------------------
-- ALU
------------------------------------------------------
-- Accumulator next state
p_accu: process (ClkxCI, ResetxRBI)
begin
if ResetxRBI='0' then
  AccuxDP <= (others => '0');
elsif ClkxCI'event and ClkxCI='1' then
  AccuxDP <= AccuxDN;
end if;
end process p_accu;

------------------------------------------------------
-- Output register
p_outreg: process (ClkxCI, ResetxRBI)
begin
if ResetxRBI='0' then
  OutRegxDP<= (others => '0');
elsif ClkxCI'event and ClkxCI='1' then
  if OutRegEnxS = '1' then
```
OutRegxDP <= OutRegxDN;
end if;
end if;
end process p_outreg;

DataOutxD0 <= OutRegxDP;

-------------------------------------------------------
-- Control FSM
-------------------------------------------------------
p_fsm : process(StatexDP, DataInReqxSI,
DataOutAckxSI, CounterxDP)
begin
--defaults
InRegEnxS <= '0';
OutRegEnxS <= '0';
AccuClrxxS <= '0';
OffsetDecxxS <= '0';
CounterIncxS <= '0';
RamWriteEnxSO <= '0';
DataInAckxSO <= '0';
DataOutReqxSO <= '0';
StatexDN <= StateDP;
--case statements
case StateDP is
when idle =>
  if DataInReqxSI = '1' then
    InRegEnxS <= '1';
    StateDN <= new_data;
  end if;
when new_data =>
  AccuClrxxS <= '1';
  RamWriteEnxSO <= '1';
  DataInAckxSO <= '1';
  StateDN <= run;
when run =>
  CounterIncxS <= '1';
  if CounterxDP = "111111" then
    OutRegEnxS <= '1';
    OffsetDecxxS <= '1';
    StateDN <= data_out;
  end if;
when data_out =>
  DataOutReqxSO <= '1';
  if DataOutAckxSI = '1' then
    StateDN <= idle;
  end if;
when others => null;
end case;
end process p_fsm;

p_clk : process (ClkxCI, ResetxRBI)
begin
  if ResetxRBI='0' then
    StatexDP <= idle;
  elsif ClkxCI'event and ClkxCI='1' then
    StatexDP <= StateDN;
  end if;
end process p_clk;
end rtl;
A.2 File: filter_top.vhd

Listing A.2: RTL top model of filter and coefficient ROM.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity filter_top is
  generic (
    CWIDTH : integer := 16;
    CAW : integer := 7;
    DWIDTH : integer := 32;
    DAW : integer := 7
  );
  port (
    ClkxCI : in std_logic;
    ResetxRBI : in std_logic;
    DataInxDI : in std_logic_vector (DWIDTH-1 downto 0);
    DataInReqxSI : in std_logic;
    DataInAckxSO : out std_logic;
    DataOutxDO : out std_logic_vector (DWIDTH-1 downto 0);
    DataOutReqxSO : out std_logic;
    DataOutAckxSI : in std_logic
  );
end filter_top;

architecture rtl of filter_top is
-- component declarations
-------------------------------------------------------------
component coeff
  generic (
    CWIDTH : integer;
    CAW : integer)
  port (
    ClkxCI : in std_logic;
    AddrxDI : in std_logic_vector (CAW-1 downto 0);
    DataxDO : out std_logic_vector (CWIDTH-1 downto 0)
  );
end component;

component dataRAM
  generic (
    DWIDTH : integer;
    DAW : integer)
  port (
    AddrxDI : in std_logic_vector (DAW-1 downto 0);
    WExSI : in std_logic;
    WClkxCI : in std_logic;
    DinxDI : in std_logic_vector (DWIDTH-1 downto 0);
    DoutxDO : out std_logic_vector (DWIDTH-1 downto 0)
  );
end component;

component filter
  generic (
    DWIDTH : integer;
    DAW : integer)
  port (
    ClkxCI : in std_logic;
    ResetxRBI : in std_logic;
    DataInxDI : in std_logic_vector (DWIDTH-1 downto 0);
    DataInReqxSI : in std_logic;
    DataInAckxSO : out std_logic;
    DataOutxDO : out std_logic_vector (DWIDTH-1 downto 0);
    DataOutReqxSO : out std_logic;
    DataOutAckxSI : in std_logic
  );
end component;
```

```
APPENDIX A. APPENDIX A: VHDL NETLISTS

DataInAckxSO : out std_logic;
DataOutDO : out std_logic
(DWIDTH-1 downto 0);
DataOutReqxSO : out std_logic;
DataOutAckxSI : in std_logic;
LutAddrxD : out std_logic
(CAW-1 downto 0);
LutReadxDI : in std_logic
(CWIDTH-1 downto 0);
RamWriteEnxSO : out std_logic;
RamAddrxD : out std_logic
(DAW-1 downto 0);
RamReadxDI : in std_logic
(DWIDTH-1 downto 0);
RamWritexD : out std_logic
(DWIDTH-1 downto 0);
end component;

signal RamWriteEnxS : std_logic;
signal RamAddrxD : std_logic
(DAW-1 downto 0);
signal RamReadxD : std_logic
(DWIDTH-1 downto 0);
signal RamWritexD : std_logic
(DWIDTH-1 downto 0);
signal LutAddrxD : std_logic
(CAW-1 downto 0);
signal LutReadxD : std_logic
(CWIDTH-1 downto 0);

begin
------------------------------------------
-- Component Instantiations
------------------------------------------

i_coeff : coeff
generic map
(CWIDTH => CWIDTH,
CAW => CAW)
port map
(Clk => ClkxCI,
AddrxDI => LutAddrxD,
DataxDO => LutReadxD);

i_dataRAM : dataRAM
generic map
(DWIDTH => DWIDTH,
DAW => DAW)
port map
(AddrxDI => RamAddrxD,
WE => RamWriteEnxS,
WCLKxCI => ClkxCI,
DinxDI => RamWritexD,
DoutxDO => RamReadxD);

i_filter : filter
generic map
(DWIDTH => DWIDTH,
DAW => DAW)
port map
(ClkxCI => ClkxCI,
ResetxRBI => ResetxRBI,
DataInxDI => DataInxDI,
DataInReqxSI => DataInReqxSI,
DataInAckxSO => DataInAckxSO,
DataOutxDO => DataOutxDO,
DataOutReqxSO => DataOutReqxSO,
DataOutAckxSI => DataOutAckxSI,
LutAddrxD => LutAddrxD,
LutReadxDI => LutReadxD,
RamWriteEnxSO => RamWriteEnxS,
RamAddrxD => RamAddrxD,
RamReadxDI => RamReadxD,
A.3 File: filter_soc.vhd

Listing A.3: RTL soc model.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
library techmap;
use techmap.gencomp.all;

entity filter_soc is
generic (  
  CWIDTH : integer := 16;
  CAW : integer := 7;
  DWIDTH : integer := 16;
  DAW : integer := 7
);
port (  
  ClkxCI : in std_logic;
  ResetxRBI : in std_logic;
  DataInxDI : in std_logic_vector (DWIDTH-1 downto 0);
  DataInReqxSI : in std_logic;
  DataInAckxSO : out std_logic;
  DataOutxDO : out std_logic_vector (DWIDTH-1 downto 0);
  DataOutReqxSO : out std_logic;
  DataOutAckxSI : in std_logic
);
end filter_soc;

architecture rtl of filter_soc is
-- component declarations
component filter_top is
generic (  
  CWIDTH : integer := 16;
  CAW : integer := 7;
  DWIDTH : integer := 16;
  DAW : integer := 7
);
port (  
  ClkxCI : in std_logic;
  ResetxRBI : in std_logic;
  DataInxDI : in std_logic_vector (DWIDTH-1 downto 0);
  DataInReqxSI : in std_logic;
  DataInAckxSO : out std_logic;
  DataOutxDO : out std_logic_vector (DWIDTH-1 downto 0);
  DataOutReqxSO : out std_logic;
  DataOutAckxSI : in std_logic
);
end component;
-- signal declarations
```
APPENDIX A. APPENDIX A: VHDL NETLISTS

------------------------------------------------------------
signal ClkxCI_s : std_logic;
signal ResetxRBI_s : std_logic;
signal DataInxDI_s : std_logic_vector(DWIDTH-1 downto 0);
signal DataInReqxSI_s : std_logic;
signal DataOutxDO_s : std_logic_vector(DWIDTH-1 downto 0);
signal DataOutReqxSO_s : std_logic;
signal VCCIO : std_logic_vector(7 downto 0);
signal GNDIO : std_logic_vector(7 downto 0);
signal VCCCO : std_logic;
signal GNDCO : std_logic;
constant padtech : integer := faraday;
constant padlevel : integer := 0;
begin
------------------------------------------------------------
-- Pad Instantiations
------------------------------------------------------------
-- Power Pads
io_VCCIO : vcciopadv generic map (width => 8, tech => padtech, limit => core_limited)
port map (VCCIO);
io_GNDIO : gndiopadv generic map (width => 8, tech => padtech, limit => core_limited)
port map (GNDIO);
io_VCCCO : vcccopad generic map (tech => padtech, limit => core_limited)
port map (VCCCO);
io_GNDCO : gndcopad generic map (tech => padtech, limit => core_limited)
port map (GNDCO);
-- Signal Pads
io_ClkxCI : inpad generic map (tech => padtech, level => padlevel, limit => core_limited)
port map (ClkxCI, ClkxCI_s);
io_ResetxRBI : inpad generic map (tech => padtech, level => padlevel, limit => core_limited)
port map (ResetxRBI, ResetxRBI_s);
io_DataInxDI : inpadv generic map (width => DWIDTH, tech => padtech, level => padlevel, limit => core_limited)
port map (DataInxDI, DataInxDI_s);
io_DataInReqxSI : inpad generic map (tech => padtech, level => limit => core_limited)
port map (DataInReqxSI, DataInReqxSI_s);
io_DataInAckxSO : outpad generic map (tech => padtech, level => padlevel, slew => fast, limit => core_limited)
port map (DataInAckxSO, DataInAckxSO_s);
io_DataOutxDO : outpadv generic map (width => DWIDTH, tech => padtech, level => padlevel, slew => fast, limit => core_limited)
port map (DataOutxDO, DataOutxDO_s);
io_DataOutReqxSO : outpad generic map (tech => padtech, level => padlevel, slew => fast, limit => core_limited)
port map (DataOutReqxSO, DataOutReqxSO_s);
io_DataOutAckxSI : inpad generic map (tech => padtech, level => limit => core_limited)
port map (DataOutAckxSI, DataOutAckxSI_s);
------------------------------------------------------------
-- Component Instantiations
------------------------------------------------------------
i_filter_top : filter_top
generic map (CWIDTH => CWIDTH, CAW => CAW, DWIDTH => DWIDTH,
APPENDIX A. APPENDIX A: VHDL NETLISTS

```
DAW => DAW;
port map (
  ClkxCI => ClkxCI_s,
  ResetxRBI => ResetxRBI_s,
  DataInxDI => DataInxDI_s,
  DataInReqxSI => DataInReqxSI_s,
  DataInAckxSO => DataInAckxSO_s,
  DataOutxDO => DataOutxDO_s,
  DataOutReqxSO => DataOutReqxSO_s,
  DataOutAckxSI => DataOutAckxSI_s
);
end rtl;
```

A.4 File: filter_soc_tb.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_signed.ALL;
USE ieee.math_real.ALL;

ENTITY filter_soc_tb IS
  GENERIC( clock_delay : time := 16 ns;
    CWIDTH : integer := 16;
    CAW : integer := 7;
    DWIDTH : integer := 16;
    DAW : integer := 7
  );
END filter_soc_tb;

ARCHITECTURE behavioral OF filter_soc_tb IS

component filter_soc is
  generic (
    CWIDTH : integer := 16;
    CAW : integer := 7;
    DWIDTH : integer := 32;
    DAW : integer := 7
  );
  port (
    ClkxCI : in std_logic;
    ResetxRBI : in std_logic;
    DataInxDI : in std_logic_vector(DWIDTH-1 downto 0);
    DataInReqxSI : in std_logic;
```
APPENDIX A. APPENDIX A: VHDL NETLISTS

DataInAckxSO : out std_logic;
DataOutxDCO : out std_logic_vector(DWIDTH - 1 downto 0);
DataOutReqxSO : out std_logic;
DataOutAckxSI : in std_logic);
end component;

SIGNAL ClkxCI : std_logic;
SIGNAL ResetxRBI : std_logic;
SIGNAL DataInxDI : std_logic_vector(DWIDTH - 1 downto 0);
SIGNAL DataInReqxSI : std_logic;
SIGNAL DataInAckxSO : std_logic;
SIGNAL DataOutxDCO : std_logic_vector(DWIDTH - 1 downto 0);
SIGNAL DataOutReqxSO : std_logic;
SIGNAL DataOutAckxSI : std_logic;

BEGIN
-- Instantiate device-under-test.
dut: filter_soc
generic map
(CWIDTH => CWIDTH,
CAW => CAW,
DWIDTH => DWIDTH,
DAW => DAW)
port map
(ClkxCI => ClkxCI,
ResetxRBI => ResetxRBI,
DataInxDI => DataInxDI,
DataInReqxSI => DataInReqxSI,
DataInAckxSO => DataInAckxSO,
DataOutxDCO => DataOutxDCO,
DataOutReqxSO => DataOutReqxSO,
DataOutAckxSI => DataOutAckxSI );

clock_generation:
PROCESS
BEGIN
-- Generate equal duty-cycle clock.
ClkxCI <= '0';
WAIT FOR ( clock_delay / 2 );
ClkxCI <= '1';
WAIT FOR ( clock_delay / 2 );
END PROCESS clock_generation;

generate_stimulus:
PROCESS
BEGIN
-- Initialize input signals.
DataInReqxSI <= '0';
DataInxDI <= ( OTHERS =>'0' );
DataOutAckxSI <= '0';

-- Reset the design and wait for 2 clock cycles.
ResetxRBI <= '0';
WAIT FOR clock_delay * 2;
ResetxRBI <= '1';

-- Wait for 2 more clock cycles before beginning test.
WAIT FOR clock_delay * 2;
\begin{verbatim}
APPENDIX A. APPENDIX A: VHDL NETLISTS

DataInxDI <= ( 0 => '1', OTHERS => '0' );
WAIT FOR clock_delay;
FOR I IN 127 DOWNTO 0 LOOP
  DataInReqxSI <= '1';
  WHILE DataInAckxSO = '0' LOOP
    WAIT FOR clock_delay;
  END LOOP;
  DataInReqxSI <= '0';
  DataInxDI <= ( OTHERS => '0' );
  WHILE DataOutReqxSO = '0' LOOP
    WAIT FOR clock_delay;
  END LOOP;
  DataOutAckxSI <= '1';
  WAIT FOR clock_delay;
  DataOutAckxSI <= '0';
END LOOP;
--
-- Wait forever.
--
WAIT;
END PROCESS generate_stimulus;

END behavioral;
\end{verbatim}
Appendix B

Appendix B: Technology independent IO-cell library

The techmap generic IO-cell library consists of configurable input, output and tri-state buffer cells. Figure B shows the architecture of such IO buffer cell. Through specification of particular generics one can configure the IO-cell. Port IO is the IO-pad to the outside world. Output ports of a toplevel cell are connected to I ports of the IO-cells. Input ports of the toplevel cell are connected to O ports of IO-cells. Input and output buffers have an enable port. The strength, slew rate of output buffers are configurable through E4,E8 and SR. PullUp/PullDown resistor and Schmitttrigger behavior of input buffers are configurable through generics PU, PD and SMT. For the sake of ease for this course we use the following generic icell configurations. Table B.1 lists the available types of IO-cells.

![Figure B.1: Architecture](image)

<table>
<thead>
<tr>
<th>Table B.1: Techmap IO cells</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Input</strong></td>
</tr>
<tr>
<td><strong>Output</strong></td>
</tr>
<tr>
<td><strong>IO</strong></td>
</tr>
<tr>
<td><strong>Tri_state</strong></td>
</tr>
<tr>
<td><strong>Single IO</strong></td>
</tr>
<tr>
<td>inpad (pad, o)</td>
</tr>
<tr>
<td>outpad (pad, i)</td>
</tr>
<tr>
<td>iopad (pad, i, e, o)</td>
</tr>
<tr>
<td>toutpad (pad, e, i)</td>
</tr>
<tr>
<td><strong>Vector IO</strong></td>
</tr>
<tr>
<td>inpadv (pad, o)</td>
</tr>
<tr>
<td>outpadv (pad, i)</td>
</tr>
<tr>
<td>iopadv (pad, i, e, o)</td>
</tr>
<tr>
<td>iopadvv (pad, i, e, o)</td>
</tr>
<tr>
<td>toutpadv (pad, e, i)</td>
</tr>
<tr>
<td>toutpadvv (pad, e, i)</td>
</tr>
</tbody>
</table>
The generic *width* is used to configure the width of the connected vectored in/output ports.

```vhdl
io_DataInxDI : inpadv
  generic map (width => DWIDTH, tech => padtech,
                level => padlevel, limit => core_limited)
  port map (DataInxDI, DataInxDI_s);

io_DataOutxD0 : outpadv
  generic map (width => DWIDTH, tech => padtech,
               level => padlevel, slew => fast, limit => core_limited)
  port map (DataOutxD0, DataOutxD0_s);
```
Appendix C

Appendix C: Tool Scripts

C.1 Synopsys: Design Compiler

C.2 Cadence: SOC Encounter

C.2.1 start.tcl

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

set PROJECT_DIR [pwd]

# Design related information (can be changed)
set DESIGN filter_soc
set TIM_MAX_LIBRARY fsd00_a_generic_core_ss0p9v125c
set TIM_MIN_LIBRARY fsd00_a_generic_core_ff1p1vm40c
set TIM_OC_MAX WCCOM ;# TYPICAL | WORST | WORST-IND
set TIM_OC_MIN BCCOM ;# TYPICAL | BEST | BEST-IND

# Floorplan settings
set FP_ASPECT_RATIO 1
set FP_ROW_DENSITY 0.85 ;# percent
set FP_CORE2IO 224 ;# micron

# Power ring and settings
set PR_WIDTH 2.8 ;# micron
set PR_SPACING 1.68 ;# micron
set PR_LAYER_TB ME7 ;# top and bottom layer
set PR_LAYER_LR ME8 ;# left and right layer

# Power stripe settings
set ST_NUM_SETS 1 ;# number of sets
set ST_SPACING 1.12 ;# micron
set ST_LAYER_V $PR_LAYER_LR
set ST_WIDTH 1.4 ;# micron
set ST_XOFS_R 0 ;# micron
set ST_XOFS_L 280 ;# micron

# Placement settings
```
APPENDIX C. TOOL SCRIPTS

# set PL_EFFORT medium;# low | medium | high
# Clock tree synthesis settings
# set CTS_BUFFER BUFCKX1
set CTS_INV INVCKX1
# Flags that drive the script behavior (can be changed)
#
# ADD_STRIPES (0 | 1)
# 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
# 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)
# can be used to have different generated file names
#
set ADD_STRIPES 1
set PLACE_TIMING 1
set CLOCK_TREE 1
set CTS_CREATE_SPEC 0
set ROUTE_TIMING 1
set OPT ""
#
# File names
#
set CONF_FILE_NAME ${DESIGN}.conf
set IO_FILE_NAME ${DESIGN}.io
set DESIGN_NAME ${DESIGN}|${OPT}
set SAVE_DESIGN_PP_NAME ${DESIGN_NAME}|fplan.enc
set SAVE_DESIGN_PR_NAME ${DESIGN_NAME}|pplan.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
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
set RPT_BLACK_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
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
set CTS_RPT_NAME ${DESIGN_NAME}|cts.rpt
set GDS_FILE_NAME ${DESIGN_NAME}.gds
#
# Absolute paths
#
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/$SAVE_DESIGN_FP_NAME \nset SAVE_DESIGN_PR_FILE \$(PROJECT_DIR)/PAR/DB/$SAVE_DESIGN_PR_NAME \nset SAVE_DESIGN_PL_FILE \$(PROJECT_DIR)/PAR/DB/$SAVE_DESIGN_PL_NAME \nset SAVE_DESIGN_PF_FILE \$(PROJECT_DIR)/PAR/DB/$SAVE_DESIGN_PF_NAME \nset SAVE_DESIGN_CT_FILE \$(PROJECT_DIR)/PAR/DB/$SAVE_DESIGN_CT_NAME \nset SAVE_DESIGN_RO_FILE \$(PROJECT_DIR)/PAR/DB/$SAVE_DESIGN_RO_NAME \nset SDF_FILE \$(PROJECT_DIR)/PAR/TIM/$SDF_FILE_NAME \nset SPEF_FILE \$(PROJECT_DIR)/PAR/TIM/$SPEF_FILE_NAME \nset TIM_RCDB_FILE \$(PROJECT_DIR)/PAR/TIM/$TIM_RCDB_NAME \nset RPT_CHECK_TA_FILE \$(PROJECT_DIR)/PAR/RPT/$RPT_CHECK_TA_NAME \nset RPT_REPORT_TA_FILE \$(PROJECT_DIR)/PAR/RPT/$RPT_REPORT_TA_NAME \nset RPT_BLACK_FILE \$(PROJECT_DIR)/PAR/RPT/$RPT_BLACK_NAME \nset RPT_GATE_COUNT_FILE \$(PROJECT_DIR)/PAR/RPT/$RPT_GATE_COUNT_NAME \nset RPT_NOTCH_FILE \$(PROJECT_DIR)/PAR/RPT/$RPT_NOTCH_NAME \nset RPT_CONN_FILE \$(PROJECT_DIR)/PAR/RPT/$RPT_CONN_NAME \nset RPT_GEOM_FILE \$(PROJECT_DIR)/PAR/RPT/$RPT_GEOM_NAME \nset RPT_DENSITY_FILE \$(PROJECT_DIR)/PAR/RPT/$RPT_DENSITY_NAME \nset VLOG_NETLIST_SIM_FILE \$(PROJECT_DIR)/HDL/GATE/$VLOG_NETLIST_SIM_NAME \nset VLOG_NETLIST_LVS_FILE \$(PROJECT_DIR)/HDL/GATE/$VLOG_NETLIST_LVS_NAME \nset CTS_SPEC_FILE \$(PROJECT_DIR)/PAR/CTS/$CTS_SPEC_NAME \nset CTS_RGUIDE_FILE \$(PROJECT_DIR)/PAR/CTS/$CTS_RGUIDE_NAME \nset CTS_RPT_FILE \$(PROJECT_DIR)/PAR/RPT/$CTS_RPT_NAME \nset GDS_FILE \$(PROJECT_DIR)/PAR/DEX/$GDS_FILE_NAME \nset GDS_MAP_FILE \$(PROJECT_DIR)/PAR/DEX/gds2.map

#-----------------------------------------------------------------------------
# Suppress Messages
#-----------------------------------------------------------------------------
suppressMessage NRDB 733
suppressMessage NREX 4 28 30
suppressMessage SOCNET 3032 3080
suppressMessage SOCLF 58 200
suppressMessage SOCOPT 3544
suppressMessage SOCPP 2008
suppressMessage TLEGAL 330
suppressMessage TECHLIB 436

#-----------------------------------------------------------------------------
# Procedures
#-----------------------------------------------------------------------------
proc make_clock_tree create_spec {
  global PROJECT_DIR CTS_BUFFER CTS_INV CTS_SPEC_FILE CTS_RGUIDE_FILE CTS_RPT_FILE
  if { \$create_spec || ![file exists \$CTS_SPEC_FILE] } {
    createClockTreeSpec \n    -bufFootprint \$CTS_BUFFER \n    -invFootprint \$CTS_INV \n    -output \$CTS_SPEC_FILE
    createClockTreeSpec \n    -output \$CTS_SPEC_FILE
    specifyClockTree -file \$CTS_SPEC_FILE \n    ckSynthesis \n    -rguide \$CTS_RGUIDE_FILE \n    -report \$CTS_RPT_FILE
    optDesign -postCTS -drv -outDir \$(PROJECT_DIR)/PAR/RPT
  }
}

# Load configuration file
#
loadConfig \$CONF_FILE

#
C.2.2 fplan.tcl

# Initialize floorplan
#-----------------------------------------------------------------------------------
#floorPlan -b 0.0 0.0 1872.08 1872.08 \ # 219.8 219.8 1092.28 1092.28 \ # 779.8 779.8 1092.28 1092.28
#floorPlan -b 0.0 0.0 1872.08 1872.08 \ # 219.8 219.8 1652.28 1652.28 \ # 261.8 261.8 1610.28 1610.28
#floorPlan -d 1872.08 1872.08 \ # 280.0 280.0 280.0 280.0
floorPlan -d 1872.08 1872.08 \ # 560.0 560.0 560.0 560.0
#floorPlan -d 3874.92 1874.88 \ # 1400.0 560.0 1400.0 560.0
#snapFPlanIO -usergrid
APPENDIX C. TOOL SCRIPTS

# snapFPlan -all

# setPlanDesignMode -useExistingPowerRail true
# setPlanDesignMode -fixPlacedMacros true
# setPlanDesignMode -groupHardMacro true -effort medium -boundaryPlace true
planDesign

#finishFloorplan -autoHalo -autoBlockage -staircase
finishFloorplan -addHalo 14 -autoBlockage

source ${PROJECT_DIR}/PAR/BIN/fillperi.tcl

saveDesign $SAVE_DESIGN_FP_FILE
setDrawView fplan
fit

C.2.3 pplan.tcl

```
## File : pwr.tcl
## Created at : Mon Jun 07 16:33:03 CEST 2010
## Created by : Alexander de Graaf
puts "----------------- Power Planning ------------------------------------------"
set crwidth 2.8
set crspace 2.80
set croffset 28.00
set brwidth 1.12
set brspace 0.56
set broffset 4.20
set vswidth 0.94
set vsspace 1.68
set vsoffset 2.80
set hswidth 0.94
set hsspace 1.68
set hsoffset 2.80
cutRow
clearCutRow
deselectAll

# Generate core rings

puts "----------------- Making Power Rings ---------------------------------------"
setAddRingOption -avoid_short 0 -ignore_rows 0
addRing -nets {GND VCC} -around core \
- layer_top ME5 - layer_bottom ME5 \ 
- layer_left ME6 - layer_right ME6 \ 
- width_top $crwidth - width_bottom $crwidth - width_left $crwidth - width_right $crwidth \ 
- spacing_top $crspace - spacing_bottom $crspace - spacing_left $crspace - spacing_right $crspace \ 
- offset_top $croffset - offset_bottom $croffset - offset_left $croffset - offset_right $croffset \ 
- snap_wire_center_to_grid None
```

# -center 1
deselectAll
#
# #################################################################################################################
#
# Generate block rings
#
# #################################################################################################################
#
# Generate block ring around buffer memory
selectInst i_filter_top / i_dataRAM_i_dmem
setAddRingOption -avoid_short 0 -ignore_rows 0 -extend_over_row 0
addRing -nets {GND VCC} -type block_rings -around selected \\
- layer_top ME5 - layer_bottom ME5 \\
- layer_left ME6 - layer_right ME6 \\
- width_top $brwidth - width_bottom $brwidth - width_left $brwidth - width_right \\
- spacing_top $brspace - spacing_bottom $brspace - spacing_left $brspace - spacing_right \\
- offset_top $broffset - offset_bottom $broffset - offset_left $broffset - offset_right \\
- use_wire_group 1 - use_interleaving_wire_group 0 - use_wire_group_bits 1 \\
- snap_wire_center_to_grid None
deselectAll
#
# Generate block ring around coefficient rom
selectInst i_filter_top / i_coeff_a9d16
setAddRingOption -avoid_short 0 -ignore_rows 0 -extend_over_row 0
addRing -nets {GND VCC} -type block_rings -around selected \\
- layer_top ME5 - layer_bottom ME5 \\
- layer_left ME6 - layer_right ME6 \\
- width_top $brwidth - width_bottom $brwidth - width_left $brwidth - width_right \\
- spacing_top $brspace - spacing_bottom $brspace - spacing_left $brspace - spacing_right \\
- offset_top $broffset - offset_bottom $broffset - offset_left $broffset - offset_right \\
- use_wire_group 1 - use_interleaving_wire_group 0 - use_wire_group_bits 1 \\
- snap_wire_center_to_grid None
deselectAll
#
# Generate stripes
#
# #################################################################################################################
#
# puts "----------------- Making Power Stripes ------------------------------------"
setAddStripeOption -remove_floating_stripe_over_block 1
addStripe -nets {GND VCC} -direction vertical -layer ME4 \ 
- width $vswidth - spacing $vsspace - set_to_set_distance 300.16 \ 
- xleft_offset 200.20 - xright_offset 84.0 \ 
- extend_to none - merge_strips_value auto - break_strips_at_block_rings 1 \\
- snap_wire_center_to_grid None
deselectAll
#
# Route VCC and GND core rings to pad pins of VCC/GND IO-cells
#
# ####################################################################################################################
#
# sroute -noCorePins -noPadRings -noStripes - jogControl { preferWithChanges differentLayer } -nets {GND VCC}
sroute -allowJogging 1 - allowLayerChange 1 - nets {GND VCC}
#
saveDesign $SAVE_DESIGN_PR_FILE
setDrawView fplan
fit
puts "------------------ Power Planning done -------------------------------"
Appendix D

Appendix D: 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 um2 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:

<table>
<thead>
<tr>
<th>Description</th>
<th>Width</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid</td>
<td>0.28 um</td>
<td>0.28 um</td>
</tr>
<tr>
<td>Core Cell</td>
<td>x * 0.28 um</td>
<td>2.8 um</td>
</tr>
<tr>
<td>IO Cell</td>
<td>60.48 um</td>
<td>142.8 um</td>
</tr>
<tr>
<td>Corner Cell</td>
<td>142.8 um</td>
<td>142.8 um</td>
</tr>
<tr>
<td>Pad Cell</td>
<td>64 um</td>
<td>77 um</td>
</tr>
</tbody>
</table>