Top-Down Digital Design Flow

Version 6.0, October 2011

Alain Vachoux
Microelectronic Systems Lab
STI-IEL-LSM
alain.vachoux@epfl.ch

Abstract
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, running in a Linux environment, are considered in this document:

• Modelsim from Mentor Graphics.
• Design Compiler from Synopsys.
• Encounter and Virtuoso from Cadence Design Systems.

The design kit used is the UMC 90nm CMOS process with the Faraday standard cell library.
# Document history

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2003</td>
<td>Initial version.</td>
</tr>
<tr>
<td>2.0.1</td>
<td>23 apr 04</td>
<td>Updated for new EDA tool releases.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>First design example (adder-subtractor) only.</td>
</tr>
<tr>
<td>2.0.2</td>
<td>26 may 04</td>
<td>Minor changes.</td>
</tr>
<tr>
<td>2.0.3</td>
<td>3 jun 04</td>
<td>Minor changes.</td>
</tr>
<tr>
<td>3.0.1</td>
<td>11 oct 05</td>
<td>Use Cadence Encounter as P&amp;R tool.</td>
</tr>
<tr>
<td>3.0.2</td>
<td>6 dec 05</td>
<td>Minor issues fixed.</td>
</tr>
<tr>
<td>3.1</td>
<td>Nov. 2006</td>
<td>PaR chapter with SoC Encounter rewritten.</td>
</tr>
<tr>
<td>3.2</td>
<td>Oct. 2007</td>
<td>Design example with two inferred arithmetic units.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New synthesis and PaR scripts.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Editorial changes.</td>
</tr>
<tr>
<td>3.3</td>
<td>Nov. 2007</td>
<td>Editorial changes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New design example including an IP (register file).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New project structure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New project and design kit installation scripts.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chapter 2: Modelsim projects.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chapter 3: improved synthesis script.</td>
</tr>
<tr>
<td>4.0b</td>
<td>Oct. 2009</td>
<td>Minor changes in chapters 1 to 3.</td>
</tr>
<tr>
<td>4.0c</td>
<td>Oct. 2009</td>
<td>Chapter 4 completely rewritten.</td>
</tr>
<tr>
<td>4.0d</td>
<td>Oct. 2009</td>
<td>Fixed typos in Chapter 4.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 2.4 completed with information on Verilog library and Wave window.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chapter 4: add filler cells after the routing.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Annex C: explain files.</td>
</tr>
<tr>
<td>5.0</td>
<td>Oct. 2010</td>
<td>Chapter 1: updated explanation of post-synthesis simulation; simplified tutorial installation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chapter 2: rewritten use of Modelsim project.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chapter 3: updated synthesis script.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chapter 4: no more FILLERE cells used; updated p+r script.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No more annexes.</td>
</tr>
<tr>
<td>5.1</td>
<td>Nov. 2010</td>
<td>Fixed incorrect table of contents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chapter 1: Fixed incorrect installation archive path.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chapter 2: Added regfile_pkg.vhd again.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chapter 3: Changed ser_en_reg to split_en_reg in elaboration report.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chapter 4 adapted to Encounter Digital Implementation 9.x.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TCL scripts streamlined.</td>
</tr>
</tbody>
</table>
# Table of Contents

Chapter 1: Introduction ................................................................. 1
  1.1 Top-Down Design Flow ......................................................... 1
  1.2 Tutorial Installation ........................................................... 3
  1.3 Tutorial Roadmap ................................................................. 7

Chapter 2: VHDL/Verilog Simulation ................................................. 9
  2.1 Getting Started ................................................................. 9
  2.2 RTL Logic Simulation .......................................................... 10
      2.2.1 Creating a RTL Simulation Project .................................... 10
      2.2.2 Editing VHDL Files ....................................................... 11
      2.2.3 Compiling the VHDL Source Files ..................................... 11
      2.2.4 Defining a RTL Simulation Configuration ......................... 12
      2.2.5 Simulating the RTL Model .............................................. 13
  2.3 Post-Synthesis Gate-Level Simulation ...................................... 17
      2.3.1 Creating a Post-Synthesis Simulation project ...................... 17
      2.3.2 Compiling the VHDL Source Files ..................................... 17
      2.3.3 Defining a Post-Synthesis Simulation Configuration ............ 18
      2.3.4 Simulating the Post-Synthesis Model ............................... 19
  2.4 Post P+R Gate-Level Simulation ............................................ 20
      2.4.1 Creating a P+R Simulation project ................................... 20
      2.4.2 Compiling the VHDL and Verilog Source Files ..................... 21
      2.4.3 Defining a P+R Simulation Configuration ......................... 21
      2.4.4 Simulating the P+R Model ............................................. 22
  2.5 Opening an Existing Project ............................................... 22

Chapter 3: RTL/Logic Synthesis ...................................................... 23
  3.1 Getting Started ............................................................... 23
  3.2 Analyzing RTL VHDL Model(s) .............................................. 24
  3.3 Elaborating Design ............................................................ 24
  3.4 Linking Design ................................................................. 27
  3.5 Defining the Design Environment ......................................... 28
  3.6 Defining Constraints ........................................................ 29
  3.7 Optimizing and Mapping the Design ...................................... 30
  3.8 Generating Reports .......................................................... 32
      3.8.1 Report on All Violated Constraints ................................ 32
      3.8.2 Area Report ............................................................... 33
      3.8.3 Critical Path Report .................................................... 34
      3.8.4 Resource Usage Report ............................................... 37
  3.9 Generating Verilog Gate-Level Netlist and SDF Timing File .......... 39
  3.10 Generating Design Constraints for Place+Route ......................... 40
  3.11 Using Scripts ................................................................. 40
  3.12 Design Optimization Using Tighter Constraints ....................... 41

Chapter 4: Standard Cell Place and Route ...................................... 43
  4.1 Getting Started ............................................................... 43
  4.2 Importing the Design ........................................................ 45
      4.2.1 Creating the Configuration File ..................................... 45
      4.2.2 Saving the Configuration File ....................................... 47
      4.2.3 Importing and Restoring the Design ................................. 47
      4.2.4 Equivalent Text Commands ........................................... 48
  4.3 Floorplanning the Design .................................................. 48

Version 6.0 - October 2011
4.3.1 Refining the Floorplan .............................................. 49  
4.3.2 Placing the Macro Block ......................................... 49  
4.3.3 Equivalent Text Commands ....................................... 50  

4.4 Power Planning ......................................................... 51  
4.4.1 Connecting Global Nets .......................................... 51  
4.4.2 Adding Power Rings ............................................... 52  
4.4.3 Adding Power Stripes (optional) ............................... 53  
4.4.4 Routing Power Nets ............................................... 54  
4.4.5 Equivalent Text Commands ....................................... 55  

4.5 Placing the Standard Cells ........................................... 55  
4.5.1 Pre-CTS Timing Optimization ................................... 56  
4.5.2 Equivalent Text Commands ....................................... 57  

4.6 Synthesizing a Clock Tree ............................................ 58  
4.6.1 Post-CTS Timing Optimization .................................. 60  
4.6.2 Equivalent Text Commands ....................................... 60  

4.7 Routing the Design .................................................... 61  
4.7.1 Post-Route Timing Optimization ................................ 61  
4.7.2 Equivalent Text Commands ....................................... 62  

4.8 Design Finishing ........................................................ 63  
4.8.1 Adding Filler Cells ................................................. 63  

4.9 Checking the Design .................................................... 64  
4.9.1 Verify connectivity ............................................... 64  
4.9.2 Verify Geometry .................................................... 65  

4.10 Generating Reports .................................................... 66  
4.10.1 Netlist statistics .................................................. 66  
4.10.2 Gate count .......................................................... 66  
4.10.3 Design summary ................................................... 67  
4.10.4 Timing ............................................................... 67  

4.11 Design Export .......................................................... 68  
4.11.1 Generating the SDF Timing File ............................... 68  
4.11.2 Generating the P+R Verilog Netlist ......................... 68  
4.11.3 Generating the GDS2 File ........................................ 69  

4.12 Importing in Virtuoso .................................................. 70  
4.13 Using Scripts ........................................................... 71  
4.14 Place+Route Using Tighter Constraints ........................... 71  

Annex A: VHDL Files .......................................................... 73  
A.1 alu32_pkg.vhd ......................................................... 73  
A.2 alu32dp_rtl.vhd ....................................................... 73  
A.3 alu32ctrl_rtl.vhd ...................................................... 74  
A.4 alu32split_rtl.vhd ..................................................... 75  
A.5 regfile_pkg.vhd ....................................................... 76  
A.6 alu32top_rtl.vhd ...................................................... 76  
A.7 alu32top_tb.vhd ....................................................... 78  
A.8 alu32top_tb_rtl_conf.vhd ......................................... 80  

Annex B: Synopsys DC Script ............................................... 81  

Annex C: Cadence Encounter Files ....................................... 83  
C.1 alu32top_clk10ns.conf .............................................. 83  
C.2 alu32top.io ............................................................ 84  
C.3 alu32top.mplan ....................................................... 85  
C.4 Place+Route Script ................................................... 86
Chapter 1: Introduction

This chapter presents the top-down design flow and its steps. It also discusses a way to organize the numerous design files involved in the flow in a consistent directory structure, and gives instructions to install a proper design environment[1].

1.1 Top-Down Design Flow

Figure 1.1 illustrates the top-down design flow and its specific steps[2].

VHDL RTL model

The first step is the creation of synthesizable VHDL RTL (Register Transfer Level) models. Such models describe the structure and the behavior of the design at a relatively high level of abstraction and

[1] The instructions are only valid when working on EPFL Linux machines. They assume a specific infrastructure that is not detailed in this document.
[2] Iterations are not mentioned for the sake of simplicity.
provide 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. The tools used at this step can range from simple text editors to dedicated graphical environments that generate VHDL code automatically.

**Pre-synthesis logic simulation**

VHDL RTL models can be then verified using logic simulation. VHDL testbench models define a relevant set of stimulus to be applied to the design under test and verification procedures on the output signals of the design under test. Logic simulation uses abstract logic signals and event-driven behaviors to achieve fast simulation times. The simulation of VHDL RTL models essentially checks the design functionality. No timings are considered yet at that stage.

**RTL/logic synthesis**

The RTL (or logic) synthesis step 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, a.k.a. standard cells, 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 gates (e.g., inverter, NAND or MUX gates) and sequential logic gates (e.g., flip-flops, latches). Each logic function is implemented in several gates to accommodate several fanout 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 Verilog gate-level netlist and a SDF (Standard Delay Format) description. The Verilog netlist can used for post-synthesis simulation, as well as the input to the place+route step. The SDF description includes delay information for simulation. Note that considered delays at this step are correct for the gates but only estimated for the interconnections.

**Post-synthesis logic simulation**

The post-synthesis gate-level simulation uses the same testbench models as the ones developed for the verification of RTL models and the same logic simulator. The simulation of the Verilog netlist requires models of the standard cells that are usually provided in the design kit. These models can be back-annotated with timing delays from the SDF file generated in synthesis. The post-synthesis simulation does not provide yet a completely accurate verification as only the timing delays of physical cells are taken into account. The interconnect delays are however either ignored or at best estimated.

**Standard cell placement and routing**

The place+route (P+R) step infers a geometric realisation of the gate-level netlist, so-called a layout. The standard cell design style places logic gates in rows of equal heights. As a consequence, all standard cells from 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 (routing) between gates are done over the cells since current processes allow several metal layers (i.e. up to 9 metal layers for the UMC 90nm CMOS process). As a consequence, the rows may be abutted and flipped so power and ground rails are shared between successive rows. Placement and routing can consider timing constraints, usually the same as the ones defined for the RTL synthesis step. Special nets such as power/ground wires and clocks are usually routed separately to meet specific constraints such as, respectively, voltage drop and electromigration, or clock skew.

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 both cell and accurate interconnect delays. The Verilog netlist may be different from the one read as input as the P+R step may
make further timing optimisations during placement, clock tree generation and routing (e.g. buffer insertion).

**Post place+route logic simulation**

The P+R Verilog gate-level netlist can be finally simulated using the existing VHDL testbenches and the more accurate SDF data extracted from the layout. The P+R Verilog netlist usually differs from the one used as input as the P+R step may have change some cell strengths or added new cells for the clock tree.

**Note**

The steps described above may need to be repeated if constraints are not met. The RTL synthesis and P+R steps use sophisticated tools that provide many commands and settings for achieving different quality of results, but only a basic set of those is presented in this document. The RTL models may also need to be modified at a last resort.

### 1.2 Tutorial Installation

**Creating the working environment**

The following command has to be executed once to create the working environment for the tutorial ($HOME indicates your home directory):[1]

```
$HOME> gtar xvf /softs/classroom/topdown/alu32.tar
```

Given the number of EDA tools and files used in the flow, the working environment must be organised in a proper way. To that end, the tutorial installation creates a project structure rooted at the directory named `ALU32`[2] (Figure 1.2). The necessary files for using the UMC 90nm CMOS design kit for full-custom and semi-custom design are installed in appropriate project subdirectories.

The `edadk.conf` file contains information on the tools and the design kit used in the project (providers, names, versions). Each tool subdirectory has a link to the file in the project root directory to allow each tool to be run in its respective directory.[3]

The roles of the main project subdirectories are as follows:

- The `CDS_SOCE` directory contains all files required for doing the standard cell place+route using Cadence SoC Encounter.
- The `CDS_VISO` directory contains all files required for editing transistor-level schematics and layouts using Cadence Virtuoso.
- The `DOC` directory may contain any relevant documentation about the design.
- The `HDL` directory contains all VHDL and Verilog source files created by the user or generated by tools.
- The `IP` directory contains intellectual property files such as the standard cell library provided by the design kit or externally generated blocks such as RAM/ROM memories or register files.

---

[1] The working environment can be actually installed in any suitable location in your home directory.

[2] The v option used with the gtar command causes the list of all directory and file names that are created to be displayed.

[3] Each “tool” directory contains important files such as setup/configuration files and libraries that are required for properly running the tool. Running the tool outside its directory may fail or may make the tool session unusable.
• The **MGC_MSIM** directory contains all files required to perform logic simulation using Mentor Modelsim.

• The **SNPS_DC** directory contains all files required to perform RTL synthesis using Synopsys Design Compiler.

More details about the use of subdirectories will be given in the next chapters.

---

**Figure 1.2:** Design project structure.

'/' denotes a directory, '->' denotes a symbolic link.

```plaintext
ALU32/
edadk.conf
CDS_SOCE/
edadk.conf -> ../edadk.conf
BIN/
CONF/
CTS/
DB/
DEX/
HDL -> ../HDL
LOG/
RPT/
SDC/
TEC/
TIM/
CDS_VISO/
cds.lib
edadk.conf -> ../edadk.conf
DLIB/
DOC/
HDL/
RTL/
GATE/
TBENCH/
IP/
MGC_MSIM/
edadk.conf -> ../edadk.conf
modelsim.ini
BIN/
DLIB/
HDL -> ../HDL
OUT/
PROJ/
SNPS_DC/
synopsys_dc.setup
.edadk.conf -> ../edadk.conf
BIN/
DB/
DLIB/
HDL -> ../HDL
RPT/
SDC/
TIM/
```

Tools & design kit configuration
Cadence SoC Encounter workspace
to be able to run tool in workspace
commands, scripts
configuration files
clock tree synthesis
design database
design exchange files
access to source files from workspace
Log files
report files
system design constraint files
technology files
timing files
Cadence Virtuoso workspace
design library mappings
to be able to run tool in workspace
design libraries
design libraries
VHDL/Verilog source files
RT-level models
gate-level models
testbench models
IP (e.g., std cells, memories)
Mentor ModelSim workspace
setup file
design libraries
access to source files from workspace
simulation outputs (e.g., waveform files)
projects
Synopsys Design Compiler workspace
setup file
to be able to run tool in workspace
commands, scripts
design database
design library
access to source files from workspace
report files
system design constraint files
timing files
Design example

Figure 1.3 gives the RTL design example that will be used throughout this document for illustrating the steps of the top-down design flow. The design includes a 32-bit ALU, some control logic and a 16x32 bit register file.

Figure 1.3: 32-bit ALU design example.

The `alu32dp` component executes arithmetic operations (addition, subtraction, multiplication). The operation to execute is defined by the primary input `cmd`.

The result of an ALU operation is stored in the `rf16x32` register file component. The register file has 16 32-bit registers and is provided as a hard macro. It won’t be synthesized, but its influence on the design will be properly taken into account, thanks to a number of files that are created by the Faraday memory compiler[1]. Here we are going to use a synchronous single-port register file[2].

The `alu32ctrl` component controls the operation of the whole system. When the signal `start` is asserted to ‘1’, the operands `op1` and `op2` are stored in registers `op1_reg` and `op2_reg`. Then, the requested operation is performed and the result is both directly available at the primary output `res` and used by the `alu32split` component to split the 64-bit input into two 32-bit words that will be stored in the register file. The upper 32 bits of the result of an ALU operation are stored first in the register file. The storage starts as a base address (default base address is 0, but it can be changed in the `alu32ctrl` component) and uses two consecutive registers in the register file. When the 16 registers are used, the storage starts again at the base address.

<table>
<thead>
<tr>
<th>cmd</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>NOOP</td>
</tr>
<tr>
<td>01</td>
<td>ADD</td>
</tr>
<tr>
<td>10</td>
<td>SUB</td>
</tr>
<tr>
<td>11</td>
<td>MULT</td>
</tr>
</tbody>
</table>

[1] A memory compiler takes a memory or register file specification as input (e.g., word size and number of words) and creates the required files for RTL/gate simulation, logic synthesis and place+route tasks.
[2] The documentation for this register file is provided separately.
The following VHDL files are installed in the HDL/RTL and HDL/TBENCH directories:\[1]\[2]:

HDL/
- alu32_pkg.vhd  -- package defining ALU operations
- alu32ctrl_rtl.vhd  -- control component
- alu32dp_rtl.vhd  -- datapath component
- alu32split_rtl.vhd  -- 64 bit to 32 bit word splitter
- alu32top_rtl.vhd  -- RTL model of top-level ALU component
- regfile_pkg.vhd  -- package defining register file component

RTL/
- alu32top_tb.vhd  -- testbench for top-level ALU\[3]
- alu32top_tb_rtl_conf.vhd  -- testbench configuration for RTL model
- alu32top_tb_mapped_conf.vhd  -- testbench configuration for mapped model
- alu32top_tb_par_conf.vhd  -- testbench configuration for P+R model

TBENCH/
- alu32top_tb.vhd  -- testbench for top-level ALU\[3]
- alu32top_tb_rtl_conf.vhd  -- testbench configuration for RTL model
- alu32top_tb_mapped_conf.vhd  -- testbench configuration for mapped model
- alu32top_tb_par_conf.vhd  -- testbench configuration for P+R model

The roles of the VHDL files are as follows:

- The top-level file for simulation is a configuration declaration:
  - alu32top_tb_rtl_conf.vhd for RTL simulation,
  - alu32top_tb_mapped_conf.vhd for post-synthesis gate-level simulation, and
  - alu32top_tb_par_conf.vhd for post place+route logic simulation.

- The alu32top_tb.vhd testbench file is unique for all kinds of simulations. The configurations are used to select which unit under test (e.g., RTL, gate-level/mapped or P+R) must be considered.

- The alu32top_rtl.vhd file defines the RTL model of the design.

- The alu32ctrl_rtl.vhd, alu32dp_rtl.vhd and alu32split_rtl.vhd files are the RTL models for the alu32ctrl, alu32dp and alu32split components, respectively (Figure 1.3).

- The SYAA90_16X32X1CM2.vhd (SYAA90_16X32X1CM2.v) file is the VHDL VITAL (resp. Verilog) simulation model of the register file. It won’t be synthesized.

- The regfile_pkg.vhd file includes a package with the component declaration for the register file.

- The alu32_pkg.vhd file includes a package that defines the encoding of ALU commands.

The files related to the register file hard macro are installed in the IP/RF16X32 directory:\[4]:

IP/
- RF16X32/  -- 16 x 32-bit register file
  - SYAA90_16X32X1CM2.dss  -- data sheet (text file)
  - SYAA90_16X32X1CM2.gds  -- register file layout (GDS2 file)
  - SYAA90_16X32X1CM2.lef  -- abstract layout view for P+R
  - SYAA90_16X32X1CM2.v  -- Verilog simulation model
  - SYAA90_16X32X1CM2.vhd  -- VHDL VITAL simulation model
  - SYAA90_16X32X1CM2_BC.lib  -- .lib files include area and timing
  - SYAA90_16X32X1CM2_FF1P1V125C.lib  -- data; they are used for RTL
  - SYAA90_16X32X1CM2_FF1P1VM40C.lib  -- synthesis and P+R
  - SYAA90_16X32X1CM2_SS0P9V125C.lib
  - SYAA90_16X32X1CM2_TC.lib
  - SYAA90_16X32X1CM2_TT1V25C.lib
  - SYAA90_16X32X1CM2_WC.lib

\[1\] The VHDL sources are given in Annex A.
\[2\] The tree Unix command offers a nice way to display directory trees. Run man tree to get more details.
\[3\] A single VHDL testbench is defined for all cases (RTL model, mapped model and P+R model). Three configuration declarations are then provided to specify which model to simulate.
\[4\] Remember that they have been generated by the Faraday memory compiler.
A number of files are finally installed in the **SNPS_DC** and **CDS_SOCE** directories:

<table>
<thead>
<tr>
<th>Directory</th>
<th>File Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNPS_DC/</td>
<td>alu32top_syn.tcl</td>
<td>-- Tcl synthesis script</td>
</tr>
<tr>
<td>DB/</td>
<td>SYAA90_16X32X1CM2_BC.db</td>
<td>-- best-case</td>
</tr>
<tr>
<td></td>
<td>SYAA90_16X32X1CM2_TC.db</td>
<td>-- typical</td>
</tr>
<tr>
<td></td>
<td>SYAA90_16X32X1CM2_WC.db</td>
<td>-- worst-case</td>
</tr>
<tr>
<td>CDS_SOCE/</td>
<td>alu32top_par.tcl</td>
<td>-- Tcl place+route script</td>
</tr>
<tr>
<td>CONF/</td>
<td>alu32top.io</td>
<td>-- definition of IO pins placement</td>
</tr>
<tr>
<td></td>
<td>alu32top.mplan</td>
<td>-- master plan for register file floorplanning</td>
</tr>
</tbody>
</table>

**Text file editing**

The alias `edt` alias calls the `nedit` text editor. It is also possible to use any other convenient editor such as `vi`, `vim`, `emacs`, `gedit`, or `kate`. All editors and the text editor in the Modelsim graphical environment also support VHDL syntax highlighting.

**1.3 Tutorial Roadmap**

Here are the main steps of the top-down design flow you are going to follow with references to the sections in the document.

Step 1: *VHDL model editing* (tool: text editor).

Step 2: *Pre-synthesis RTL VHDL simulation* (tool: Mentor Modelsim) [Chapter 2, Section 2.2]

Step 3: *RTL synthesis* (tool: Synopsys Design Compiler) [Chapter 3]

Step 4: *Post-synthesis VHDL simulation* (tool: Mentor Modelsim) [Chapter 2, Section 2.3]

Step 5: *Place+route of synthesized design* (tool: Cadence SoC Encounter) [Chapter 4]

Step 6: *Post-P+R mixed VHDL/Verilog simulation* (tool: Mentor Modelsim) [Chapter 2, Section 2.4].
Chapter 2: VHDL/Verilog Simulation

This chapter presents the main steps to perform the logic simulation of VHDL or Verilog models with the Modelsim tool from Mentor Graphics.

2.1 Getting Started

To start the Modelsim environment, go the MGC_MSIM directory and run the vsim command:

```
%ALU32> cd MGC_MSIM
%ALU32/MGC_MSIM> vsim &a*b
```

a. The & character after the command makes the tool running in the background, so the shell is still available for running other commands.
b. The command must be run from a place where the modelsim.ini file is visible. This is a text file that defines all the settings for the Modelsim tool and in particular the mappings of VHDL logical design libraries to actual physical directory locations.

The tool opens in the NoDesign layout view:

One pane lists the available VHDL/Verilog design libraries (from the modelsim.ini file). Another pane includes a command line and displays the output of executed commands. Note that the Help menu on the top right allows one to access the complete documentation of the tool.

**Note:** Since the tool keeps the history of previous sessions, the working directory currently considered may not be the one from where the vsim command has been executed. In that case, from the main menu select File > Change Directory... to point to the proper location.
2.2 RTL Logic Simulation

The task of a RTL logic simulation is to validate the functionality of the synthezisable RTL VHDL model of the system under design.

2.2.1 Creating a RTL Simulation Project

Modelsim allows organizing files and simulation configurations into *simulation projects*. It may take some time to properly set up such projects, but it definitively saves a lot of time when redoing a simulation after having brought changes to models and/or simulation conditions.

To create a new simulation project:

Change the working directory
Select **File > Change Directory...**
and then select the **PROJ** directory.

Click on **File > New > Project...**
and define:
- the project name: **ALU32-RTL**
- the project location: directory path ...
- the default library name: **ALU32-RTL**
- the file **modelsim.ini** to use as a reference to the existing file in the **MGC_MSIM** directory ...

Clicking **OK** creates a new **Project** tab in the upper pane (the library view is now in a **Library** tab). It also creates the **ALU32-RTL.mpf** file in the **PROJ** directory.

Then, existing VHDL files can be added to the project by clicking on **Add Existing File** and selecting the following files:

```
ALU32/HDL/RTL/
  alu32_pkg.vhd
  alu32ctrl_rtl.vhd
  alu32dp_rtl.vhd
  alu32split_rtl.vhd
  alu32top_rtl.vhd
  regfile_pkg.vhd

ALU32/HDL/TBENCH/
  alu32top_tb.vhd
  alu32top_tb_rtl_conf.vhd

IP/RF16X32/SYAA90_16X32X1CM2.vhd
```

It is not necessary to copy the VHDL files in the simulation project directory **PROJ**. They can be only referenced from their current locations.

---

[1] The logical library **WORK** will be automatically bound to the **ALU32-RTL** library.
[2] It is safer to always specify “../modelsim.ini”.
[3] Select the .mpf file when opening an existing project (see Section 2.5).
When the Project tab is active (in front), the main menu includes a Project menu. Selecting Project > Project Settings... allows for getting compiler outputs in the console. Click OK to close the dialog box.

### 2.2.2 Editing VHDL Files

In addition to the editor tools mentioned in “Tutorial Installation” on page 3, VHDL files can be edited in the Modelsim environment by double left-clicking on a file name in the Project tab. A new text editing pane then opens. The text editing pane can be undocked by clicking on the undock icon.

### 2.2.3 Compiling the VHDL Source Files

The Project tab now includes all added files in the order in which they have been added. It is now important to define the correct compilation order, so the tool can only recompile the necessary files when modifications are brought to VHDL sources. To do that, right-click in the Project tab and select Compile > Compile Order... Select the files and click on the up or down arrows to define the order (from top to bottom)[1]. Click OK to store the new order that now appears in the Project tab.

All VHDL files can be compiled in the right order by selecting Compile > Compile All from the main menu (or, alternatively, by right clicking in the Project tab and selecting the same command).

---

[1] To determine the correct order, refer to the descriptions of the files in Chapter 1, Section 1.2. Alternatively, clicking on the Auto Generate button will automatically define the correct order.
If there are no compilation errors, each file status becomes OK (✓). If not, the displayed status is ❌. Compilation error messages are displayed in red in the console pane. Double-clicking on the error message line opens the editor pane and highlights the offending line.

It is possible to only recompile the necessary source files by right clicking in the **Project** tab and selecting **Compile > Compile Out-of-Date**. It is also possible to compile one or more selected files (SHIFT-left click on file names) by right clicking in the **Project** tab and selecting **Compile > Compile Selected**.

### 2.2.4 Defining a RTL Simulation Configuration

A simulation configuration defines the necessary settings for starting a simulation only once. Then, the configuration can be executed as many times as required by left-clicking twice on its icon in the **Project** tab.

To create a new RTL simulation configuration, select from the main menu (or, alternatively, right-click in the **Project** tab and select the same command)

**Project > Add to Project > Simulation Configuration...** and define:

- **The simulation configuration name**: **RTL**.
- The design entity to load: here select the `alu32top_tb_rtl_conf` configuration declaration in the **ALU32-RTL** library\(^1\).
- **The simulation resolution**: **ns**\(^2\).

Do not check the **Enable Optimization** box if you want to access all signals and variables in the design for displaying them in simulation.

Then click on the **Others** tab and fill the **Other Vsim Options** field with:

The `-do` option introduces the path to the file that contains simulator commands to be executed each time the simulation is launched (also called *do file*). The file will actually include commands to set up the waveform window with all required signals\(^3\).

The `-wlf` option introduces the path to the file that will store the simulation waveforms\(^4\). This way it will be possible to reload the waveforms without resimulation.

Click on **Save** to save the simulation configuration in the project. The icon **RTL** is the added in the **Project** tab\(^5\).

---

1. The **WORK** library has been bound to the **ALU32-RTL** library at the project creation.
2. The simulation resolution is here mainly depending on the delays used in the testbench.
3. You'll create the do file after you start the simulation configuration for the first time.
4. The wlf file will be actually created after a first simulation has been run.
2.2.5 Simulating the RTL Model

The simulation of the RTL model is started by double left-clicking on the RTL simulation configuration in the Project tab. The Modelsim environment now displays the Simulate layout view, which includes a sim tab with the simulation hierarchy and an Objects pane with all objects (signals) visible in the current hierarchy level (alu32top_tb as selected in the sim tab).

The console pane has an “invalid command name” error as the do file ../BIN/alu32top_rtl.do does not yet exist. The file will be created in the following step, so this error message won’t be issued anymore when relaunching the RTL simulation.

The next step is to define the signals to display in the Wave window during simulation. In the Objects pane, right-click and select Add > To Wave > Signals in Region. This opens the new Wave pane that includes all signals visible in the current hierarchy. Panes can be undocked by clicking on the undock icon . Note that selecting Add > To Wave > Signals in design includes all signals from the top to the bottom of the hierarchy.

By default, the multi-bit signals op1, op2, res and rf_do are displayed as bit words. It is possible to use a decimal radix by selecting all the signals, right-clicking in the Wave pane, and selecting Radix > Decimal. Other radixes are available (e.g., octal, hexadecimal, unsigned).

[5] It is possible to modify an existing simulation configuration by selecting it in the Project tab, right-clicking, and then selecting Properties...
The waveform settings can now be saved by selecting **File > Save Format** and specifying the path to the file `../BIN/alu32top_rtl.do`. This save command must be executed each time a modification in the Wave pane is intended to be valid for subsequent RTL simulations.

It is often interesting to display signals in the hierarchy. For instance, the **state** signal in the `i_ctrl` instance (`alu32ctrl` component) can be added by selecting the instance in the `sim` tab and the **state** signal in the **Objects** pane, then right-clicking in the **Objects** pane and selecting **Add > To Wave > Selected Signals**.

**VHDL variables** can be also displayed in the **Wave** pane. Let’s do this for the array variable that represents the register file content. The variable is called **memorycore** and is defined in the instance `i_rf` (`rf16x32` component), but we need to open the **Locals** pane (select **View > Locals** in the main menu) to be able to select it and add it to the **Wave** pane.

Select a decimal radix for displaying the values of the **memorycore** variable.
As the signal/variable list in the **Wave** pane can become large, it is convenient to insert dividers to group signals/variables by component instance or any other relevant sorting criterion. Dividers can be inserted by right-clicking in the signals/variables list and selecting **Insert Divider**. Dividers can have names. Signals, variables and dividers can be freely moved by left-clicking on the names and moving the selection by keeping the left mouse button clicked.

Add signals to the **Wave** pane as given on the right. The given full paths are here to tell which instance to select in the `sim` tab so the objects visible at that level are listed in the **Objects** pane.

To display full paths or only leaf names, do the following:
- If the **Wave** pane is *docked*, select
  **Wave > Wave Preferences**... in the main menu.
- If the **Wave** pane is *undocked*, select
  **Tools > Window Preferences**... in the **Wave** pane menu.
Then specify a **Display Signal Path** of 0 for full paths (or 1 for leaf names only). Click **OK** to confirm the change.

Do not forget to save the final state of the **Wave** pane with the **File > Save Format** command.

Now, it is possible to run a simulation. To start the simulation, it is either possible to enter run commands in the simulation console such as:

```
VSIM 7> run 10 ns
```

or to click on the **Run** icon in the main menu or in the **Wave** pane.

Another useful run command is the **Run -All** command that runs the simulation until there is no more pending event.

This last run command can however lead to never ending simulation when the model has a continuously switching signal such as a clock signal\(^1\). It is anyway possible to stop the current simulation by clicking the **Break** icon in the main menu or in the **Wave** pane.

\(^1\) The provided testbench does have a continuously switching clock signal, but it also freezes the clock signal when no more stimulus is applied. This way, the **Run -All** command can be safely used in that particular case.
pane (alternatively, by either selecting Simulate > Break or Simulate > End Simulation in the main menu).

The full simulation of the provided testbench executes a few addition, subtraction and multiplication operations.

You can use the zoom commands and the cursor commands for more details.

If you need to make any changes in the VHDL source files, do not stop the simulation. Make the changes, then recompile all required files as described in Section 2.2.3, and finally restart the simulation by clicking on the Restart icon or with the following command in the console pane:

```
VSIM 8> restart -f
```

Once the RTL model has been successfully validated, it is possible to do the RTL synthesis as described in Chapter 3.
2.3 Post-Synthesis Gate-Level Simulation

This step must be only done when the VHDL RTL model has been synthesized to gates (Chapter 3). At that point, the synthesis step has generated a Verilog gate-level netlist and a post-synthesis SDF timing file.

2.3.1 Creating a Post-Synthesis Simulation project

The creation of the project follows the same steps as for the RTL simulation project (Section 2.2.1).

Change the working directory (if needed). Select File > Change Directory... and then select the PROJ directory.

Click on File > New > Project... and define:
- the project name: ALU32-MAPPED
- the project location: directory path ALU32/MGC_MSIM/PROJ
- the default library name: ALU32-MAPPED
- the file modelsim.ini to use as a reference to the existing file in the MGC_MSIM directory: ../modelsim.ini.

Clicking OK creates a new Project tab in the upper pane (the library view is now in a Library tab). It also creates the file ALU32-MAPPED.mpf in the PROJ directory.

Then, existing VHDL and Verilog files[1] can be added to the project by clicking on Add Existing File and selecting the following files:

```
ALU32/HDL/RTL/
    alu32_pkg.vhd
    regfile_pkg.vhd

ALU32/HDL/GATE/
    alu32top_clk10ns_mapped.v[2]

ALU32/HDL/TBENCH/
    alu32top_tb.vhd
    alu32top_tb_mapped_conf.vhd

IP/RF16X32/SYAA90_16X32X1CM2.vhd[3]
```

When the Project tab is active (in front), the main menu includes a Project menu. Selecting Project > Project Settings... allows for setting the display of compiler output in the console.

Click OK to close the dialog box.

2.3.2 Compiling the VHDL Source Files

Use Compile > Compile Order... to define the correct order of files for compilation.

Use Compile > Compile All for compiling all sources in the given order.

---

[1] The simulation will use a mix of VHDL models (packages, testbench and configuration) and Verilog models (synthesized netlist and IP block).

[2] The file name has been defined when generating the synthesized gate-level Verilog model (Chapter 3, Section 3.9). The file name includes an indication of the considered case (here a 10ns clock).

[3] The Verilog model of the register file could be used as well.
2.3.3 Defining a Post-Synthesis Simulation Configuration

Select Project > Add to Project > Simulation Configuration... and define:
- The simulation configuration name: MAPPED.
- The design entity to load: here select the configuration declaration alu32top_tb_mapped_conf in library ALU32-MAPPED[1].
- The simulation resolution: ps[2].

Click on the Others tab and fill the Other Vsim Options field with:
-do ../BIN/alu32top_mapped.do -wlf ../OUT/alu32top_clk10ns_mapped.wlf

Click on the SDF tab and define:
- The SDF timing file for backannotation. This file has been generated during RTL synthesis (Chapter 3, Section 3.9) and should be located in directory ../SNPS_DC/TIM under the file name alu32top_clk10ns_mapped_vlog.sdf[3]. Click the Add... button to specify the SDF file path. The region to get backannotated is the label name of the component instance in the VHDL testbench, namely: uut.
- The delay type of typ (typical) can be used here. Other types of min or max are possible, but their actual use depends on how far the SDF file specifies different timing values for minimum, typical, and maximum corner cases[4].
- Check the box “Reduce SDF errors to warnings”. This is required to avoid the simulation to stop prematurely due to errors such as “Failed to find port ‘...’”. 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[5]). SDF warnings issued in simulation should be anyway checked. They may sometimes indicate that the used SDF file is not consistent with the backannotated region[6].

[1] The WORK library has been bound to the ALU32-MAPPED library at the project creation.
[2] A smaller time resolution is required to properly take cell delays into accounts in simulation.
[3] The file name can be different when considering a different clock period.
[4] All timing values in SDF files are specified as min:typ:max value sets.
[5] Estimated interconnect delays have been included in cell delays when generating the SDF file.
[6] This is one reason why using a consistent file naming scheme is important.
Click on the **Library** tab and add the path to the pre-compiled library of Verilog models of the cells.

![Image of Library tab with selected library]

Click **Save** to save the simulation configuration.

### 2.3.4 Simulating the Post-Synthesis Model

This step follows the same steps as for the RTL simulation project (Section 2.2.5).

The preparation of the **Wave** pane may need additional work as some interesting local signals such as the state of the control part or internal registers are distributed over several bit signals. It would then be convenient to combine all individual bits into single vectors. Let’s discuss how to do that for the **state** signal in the control part

Select the instance `alu32top/uut/i_ctrl` in the **sim** pane.

In the **Objects** pane, select the three bit signals `state_0_port`, `state_1_port`, and `state_2_port`.

Right-click in the **Objects** pane and select **Add > To Wave > Selected Signals**.

In the **Wave** pane, select the three recently added signals. Select **Combine Signals...** in the main menu **Wave** (or **Tools > Combine Signals...** in the **Wave** pane if it is

---

[1] The state bus signal is sometimes already available, so the combining is not necessary.
[2] The index in the middle is the bit position in the 3-bit word.
Click OK. The 3-bit signal \textbf{state} is now added to the Wave pane. Do not forget to save the new format (File > Save Format) in the file path ../BIN/alu32top_mapped.do.

The simulation of the mapped design should show the effects of timing delays. Two kinds of timing delays are expected: the clock-to-Q delays of flip-flops and the propagation delays in combinational paths. In the Wave window, ModelSim denotes VHDL signals or variables with dark blue icons and Verilog variables as light blue icons.

\section*{2.4 Post P+R Gate-Level Simulation}

This step must be only done when the synthesized netlist has been placed and routed (Chapter 4).

\subsection*{2.4.1 Creating a P+R Simulation project}

The creation of a P+R (place and route) simulation project follows the same steps as for the gate-level simulation project (Section 2.3.1):

- Change the working directory (if needed). Select File > Change Directory... and then select the directory PROJ.

- Click on File > New > Project... and define:
  - the project name: ALU32-PARED
  - the project location: directory path ../ALU32/MGC_MSIM/PROJ
- the default library name: **ALU32-PARED**
- the file `modelsim.ini` to use as a *reference* to the existing file in the `MGC_MSIM` directory:
  ```
  ../modelsim.ini
  ```

Clicking **OK** creates a new **Project** tab in the upper pane (the library view is now in a **Library** tab). It also creates the **ALU32-PARED.mpf** file in the **PROJ** directory.

Then, existing VHDL and Verilog files can be added to the project by clicking on **Add Existing File** and selecting the following files:

- **ALU32/HDL/RTL/**
  - `alu32_pkg.vhd`
  - `regfile_pkg.vhd`
- **ALU32/HDL/GATE/**
  - `alu32top_clk10ns_pared.v`[^1]
- **ALU32/HDL/TBENCH/**
  - `alu32top_tb.vhd`
  - `alu32top_tb_pared_conf.vhd`
- **IP/RF16X32/**
  - `SYAA90_16X32X1CM2.v`[^2]

When the **Project** tab is active (in front), the main menu includes a **Project** menu. Selecting **Project > Project Settings...** allows for setting the display of compiler output in the console. Click **OK** to close the dialog box.

### 2.4.2 Compiling the VHDL and Verilog Source Files

Use **Compile > Compile Order...** to define the correct order of files for compilation.

Use **Compile > Compile All** for compiling all sources in the given order.

### 2.4.3 Defining a P+R Simulation Configuration

Select **Project > Add to Project > Simulation Configuration...** and define:
- The simulation configuration name: **PARED**.
- The design entity to load: here select the `alu32top_tb_pared_conf` configuration declaration in **ALU32-PARED** library[^3].
- The simulation resolution: **100fs**.

Click on the **Others** tab and fill the **Other Vsim Options** field with:
```bash
-wlf ../OUT/alu32top_clk10ns_pared.wlf -do ../BIN/alu32top_pared.do
```

Click on the **SDF** tab and define:
- The SDF timing file for backannotation. This file has been generated during the place+route phase (Chapter 4, Section 4.11.1) and should be located in the **ALU32/CDS_SOCE/TIM** directory under the file name `alu32top_clk10ns_pared_vlog.sdf`[^4]. Also select max delays.
- Check the box “Reduce SDF errors to warnings”.

Click on the **Library** tab and add the path to the pre-compiled library of Verilog models of the cells.

Click **Save** to save the simulation configuration.

[^1]: The file name has been defined when generating the place+route Verilog netlist (Chapter 4, Section 4.11.2).
[^2]: The VHDL model of the register file could be used as well.
[^3]: The **WORK** library has been bound to the **ALU32-PARED** library at the project creation.
[^4]: The file name can be different when considering a different clock period.
2.4.4 Simulating the P+R Model

This step follows the same steps as for the gate-level simulation project (Section 2.3.4). The results should be also similar, with possible (small) differences in timing delays as now the interconnect delays are properly taken into account.

Note that some signal names and signal locations in the model hierarchy may be different from the post-synthesis simulation. This is because we are now using Verilog models for the cells and the $I_{RF}$ component. Verilog models are case-sensitive. Also the memory array in the $I_{RF}$ component is now called Memory..

2.5 Opening an Existing Project

To open an existing project, select File > Open... from the main menu (or click on the icon ).

Select the files of type .mpf and select the project.

Click Open.

If another project was already open, you are invited to close it.
Chapter 3: RTL/Logic Synthesis

This chapter presents the main steps to perform the RTL/logic synthesis of the VHDL RTL model with the Design Vision and Design Compiler tools from Synopsys.

3.1 Getting Started

To start the Design Vision environment, go the \texttt{SNPS\_DC} directory and run the \texttt{design\_vision} command:

\begin{verbatim}
%ALU32> cd SNPS\_DC
%ALU32/\texttt{SNPS\_DC}> design\_vision
\end{verbatim}

\begin{itemize}
  \item For this command to execute properly, a file named \texttt{.synopsys\_dc\_setup} must exist in the working directory. This file is read when the tool starts. It contains the definitions of the target library (i.e., the library that defines the area/timing/power characteristics of the physical logic gates), the symbol library (i.e., the library of the graphical symbols of the logic gates), and the link libraries (i.e., the libraries that define the area/timing/power characteristics of other components such as IP components). The \texttt{.synopsys\_dc\_setup} file also defines the mapping of the VHDL library WORK that will store all analyzed VHDL design units.
\end{itemize}

The effect of the command is to open two windows: one console terminal and the Design Vision GUI\textsuperscript{[1]}.

\begin{itemize}
  \item The console terminal is useful for entering Design Compiler (DC) commands, to source Tcl scripts, or to run some Unix commands. The tutorial will show GUI commands, but not all commands are available.
\end{itemize}

\textsuperscript{[1]} This is specific to the EPFL setup. When exiting Design Vision (File > Exit), the console terminal must be closed separately (click on the upper right X window box).
able in the GUI. For complex and or repetitive synthesis tasks, it is anyway highly recommended to use scripts as discussed in Section 3.11. All commands issued in a session are recorded in the \texttt{command.log} file in the working directory.

To get help on the Synopsys tool suite, run the \texttt{snps_fev_doc} command in a Unix shell. The Design Vision GUI has a Help menu leading to man pages of DC variables and commands and help on Design Vision GUI. It also possible to run the command \texttt{man <DC command or variable>} in the GUI command line.

### 3.2 Analyzing RTL VHDL Model(s)

The analysis phase compiles the VHDL models and checks that the VHDL codes are synthesizable\textsuperscript{[1]}. Select File $\rightarrow$ Analyze... in the main menu. Use the Add... button to select and add all the VHDL files you need to analyze\textsuperscript{[2]}.

Use the up/down arrow buttons to move the files in their correct analyze order (the top one will be analyzed first).

Click OK. The analyzed design units are then stored in the VHDL design library WORK as mapped in the \texttt{.synopsys_dc.setup} file.

The console terminal and the log view contain the trace of the execution of the command. The History tab contain the history of all executed commands in the session.

It is also possible to run the analyze command from the console terminal\textsuperscript{[3]}:

```
.../HDL/RTL/alu32_pkg.vhd
... .../HDL/RTL/alu32top_rtl.vhd }
```

### 3.3 Elaborating Design

The elaboration phase performs a generic pre-synthesis of the analyzed models. It essentially identifies the registers (flip-flops and latches) that will be inferred.

Select File $\rightarrow$ Elaborate... in the main menu.

Select the ALU32TOP(RTL) design entity.

The DEFAULT library is identical to the WORK library.

\textsuperscript{[1]} VHDL code that successfully compiled for RTL simulation may not do here if there are statements that do not have any meaning for synthesis.

\textsuperscript{[2]} The VHDL model of the IP block (register file) is not synthesizable. Another model of this IP that can be used for synthesis will be loaded later (Section 3.4).

\textsuperscript{[3]} Commands in grey boxes can be also executed from the Design Vision command line.
If the “Reanalyze out-of-date libraries” check box is checked, VHDL sources that have been modified are automatically reanalyzed before elaboration.

Click OK[1].

It is also possible to run the elaborate command from the console terminal:

```
design_vision> elaborate ALU32TOP -architecture RTL -library DEFAULT -update
```

The console terminal now displays the inferred registers for each component and their respective kinds of set/reset (AR/AS: asynchronous reset/set, SR/SS: synchronous reset/set).

**Inferred devices in process**

```
in routine alu32top line 41 in file '.../ALU32/HDL/RTL/alu32top_rtl.vhd'.
```

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Type</th>
<th>Width</th>
<th>Bus</th>
<th>MB</th>
<th>AR</th>
<th>AS</th>
<th>SR</th>
<th>SS</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>op2_reg_reg</td>
<td>Flip-flop</td>
<td>32</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>op3_reg_reg</td>
<td>Flip-flop</td>
<td>32</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

**Inferred memory devices in process**

```
in routine alu32split line 24 in file '.../ALU32/HDL/RTL/alu32split_rtl.vhd'.
```

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Type</th>
<th>Width</th>
<th>Bus</th>
<th>MB</th>
<th>AR</th>
<th>AS</th>
<th>SR</th>
<th>SS</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>bysel_reg</td>
<td>Flip-flop</td>
<td>1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>din_reg_reg</td>
<td>Flip-flop</td>
<td>64</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

**Inferred memory devices in process**

```
in routine alu32ctrl line 32 in file '.../ALU32/HDL/RTL/alu32ctrl_rtl.vhd'.
```

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Type</th>
<th>Width</th>
<th>Bus</th>
<th>MB</th>
<th>AR</th>
<th>AS</th>
<th>SR</th>
<th>SS</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>rf_addr_reg</td>
<td>Flip-flop</td>
<td>4</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>rf_we_b_reg</td>
<td>Flip-flop</td>
<td>1</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>split_en_reg</td>
<td>Flip-flop</td>
<td>1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>load_reg</td>
<td>Flip-flop</td>
<td>1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>state_reg</td>
<td>Flip-flop</td>
<td>3</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>addr_reg</td>
<td>Flip-flop</td>
<td>4</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>rf_cs_b_reg</td>
<td>Flip-flop</td>
<td>1</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>done_reg</td>
<td>Flip-flop</td>
<td>1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Check that the report does not mention unwanted latches in the Type column. Note that latches may be legal in some specific cases such as large registers or FIFOs.

---

[1] If the elaborated design entity had generic parameters, they would be listed in the Parameters list. You’d then need to specify actual values for all of the listed parameters.
It is possible to display the elaborated schematics by selecting a component in the hierarchy view\(^1\) and then clicking on the Create Design Schematic icon. Let’s do it for the I_DP instance. The cell list clearly shows that the cells are still generic and not yet bound to real gates.

The displayed schematic for the I_DP instance shows generic components. In particular, the add\_28, sub\_31 and mult\_34 instances are still empty. They correspond to "+", "-" and "\*" operators in the VHDL source. They will be later mapped to a real adder, subtractor and multiplier.

It is good practice to save the elaborated design at that step. This way, it will be possible to reload the elaborated design and apply new synthesis constraints without the need to (re)analyze all VHDL sources.

Select the design unit alu32top in the hierarchy view.

Select File > Save As... in the main menu.

Store the saved design in the SNPS_DC/DB directory under the file name alu32top_elab.ddc.

---

\(^1\) The I_RF instance is not displayed in the hierarchy view as it does not contain any gate.
Select the **DDC** format.

Check the box "Save all designs in hierarchy".

Click **Save**.

It is also possible to run the **write** command from the console terminal:

```
  design_vision> write -hierarchy -format ddc -output DB/alu32top_elab.ddc
```

To read back a ddc file, select **File > Read...** in the main menu and select the file to read. Alternatively, use one of the following commands in the console terminal:

```
  design_vision> read_file -format ddc DB/alu32top_elab.ddc

  design_vision> read_ddc DB/alu32top_elab.ddc
```

### 3.4 Linking Design

After elaboration, the console also displayed warning messages related to the register file component for which no VHDL design entity has been (deliberately) defined:

**Information:** Building the design 'SYAA90_16X32X1CM2'. (HDL-193)

**Warning:** Can't find the design 'SYAA90_16X32X1CM2'
in the library 'WORK'. (LBR-1)

**Warning:** Design 'alu32top' has '1' unresolved references. For more detailedinformation, use the "link" command. (UID-341)

The link operation locates all of the designs and library components referenced in the current design and connects (links) them to the current design.

Select **File > Link Design...** in the main menu.

Add the specification `DB/SYAA90_16X32X1CM2_WC.db`\(^1\) to the link library list.

Click **OK**.

It is also possible to run the **link** command from the console terminal:

```
  design_vision> set link_library "$link_library DB/SYAA90_16X32X1CM2_WC.db"
  design_vision> link
```

---

\(^1\) We are considering a conservative approach using worst-case (WC) timings. Also note that the name is case-sensitive.
Now the **IP_RF** instance is properly linked to the library component that describes the characteristics of the used register file (area, timings, power consumption). You can check that by running the command `report_hierarchy` in the console terminal:

Information: This design contains unmapped logic. (RPT-7)

```
alu32top
 GTECH_NOT  gtech
 SYAA90_16X32X1CM2  SYAA90_16X32X1CM2_WC
alu32ctrl1
 GTECH_AND2  gtech
 GTECH_BUF  gtech
 GTECH_NOT  gtech
 GTECH_OR2  gtech
 GTECH_OR3  gtech
 GTECH_OR5  gtech
alu32dp
 GTECH_AND2  gtech
 GTECH_BUF  gtech
 GTECH_NOT  gtech
 GTECH_OR2  gtech
alu32split
 GTECH_BUF  gtech
 GTECH_NOT  gtech
```

### 3.5 Defining the Design Environment

Before a design can be mapped and optimized to real gates, the environment in which the design is expected to operate must be properly defined. The design environment includes *operating conditions*, *wire load models*, and *system interface characteristics*. You can use the `report_design` command to display the current operating conditions and wire load model used for the design.

Operating conditions relate to temperature, voltage, and process variations. These information are defined in synthesis libraries (.lib/.db files) from the standard cell library provider. Two scenarios are possible: 1) A single library file with cells characterized for typical conditions (temperature 25°C, typical voltage, typical process values) and derating factors for worst-case and best-case conditions, and 2) One library file per operating condition, no derating factors.

We are here in the second case. The `.synopsys_dc_setup` file in the **SNPS_DC** directory defines both a *maximum library* (called `slow.db/fsd0a_a_generic_core_ss0p9v125c`[^1]: 0.9V, 125°C, slow-slow[^2] process) and a *minimum library* (called `fast.db/fsd0a_a_generic_core_ff1p1vm40c`: 1.1V, -40°C, fast-fast process). The maximum library will be used for verifying minimum delay constraints (e.g. set-up times). The minimum library will be used for verifying maximum delay constraints (e.g. hold times).

Wire load models define estimations of the effects of wire length and fanout on resistance, capacitance, and area of nets. This data is used to compute estimated wire delays that can be added to cell delays to give timing performance reports. In our case, several wire load models are defined in the cell library coming from the foundry[^3]. The one to be used in a particular synthesis run is automatically selected according to the (estimated) design area.

System interface characteristics define the direct environment in which the design will work, namely: drive (resistance) characteristics on input ports, capacitance on input and output ports and fanout loads on output ports. These won’t be considered in this document.

[^1]: The second name is the full library name as provided by Faraday. The first name is locally defined.
[^2]: This corresponds to NMOS-PMOS transistors.
[^3]: For time critical designs, it may be better to create more accurate wire load models from the system under design. This can be done in the P+R tool, but this is not considered in this document.
3.6 Defining Constraints

Typical constraints for synthesis are constraints on area and timings. Since the design is synchronous, constraints on the clock signal must be defined. Further, a constraint on the area will require to get the smallest possible area.

To define clock attributes (its source, period/frequency and duty cycle), select the `alu32top` entity in the hierarchy view and then click on the Create Symbol View icon in the main menu.

In the symbol view, select the `clk` pin and then select Attributes > Specify Clock... in the main menu.

Define a clock period of 10 (a unit of ns is implied) and a duty cycle of 50%. Time unit is not explicitly specified here. It is defined in the cell library and is the ns in our case.

Click OK to confirm the clock definition.

It is also possible to run the `create_clock` command from the console terminal:

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

The area constraint is usually that we want a minimum area. Design Compiler always considers timing constraints as to be met first, then it will try to meet area constraints, if any.

Defining an area constraint (maximum area) of 0 is not realistic, but it is the way to ask DC to minimize area. To do that, select the `alu32top` entity in the hierarchy view and then select Attributes > Optimization Constraints > Design Constraints... in the main menu. Specify a maximum area of 0.

Click OK.

Alternatively, the `set_max_area` command can be executed from the console terminal:

```
  design_vision> set_max_area 0
```
3.7 Optimizing and Mapping the Design

The design mapping and optimization phase, also called here *compilation* phase, is technology dependent. It performs the assignment of logic gates from the standard cell library to the generic gates in the elaborated design in such a way that the defined constraints are met.

Select the `alu32top` entity in the hierarchy view and then select Design > Compile Design... in the main menu.

For a first run, there is no need to change the default settings (except the "Exact map" box which is unchecked.

The map effort specifies the relative amount of CPU time spent selecting the proper gates from the cell library. Possible values are *medium* or *high*. A high map effort will certainly consume (much) more CPU time without any guarantee that the mapping result will be (much) better. The area effort is the relative amount of CPU time spent during the area recovery phase. It attempts to meet the defined area constraint without breaking the defined timing constraints. Here possible values are *low*, *medium* or *high*. Since we imposed a maximum area of zero, which is pretty unrealistic, a low or medium area effort is fine.

Click OK to start the compilation process. The console terminal and the log view now display the progress of the work. One of the first messages in the console terminal is an information saying that there are potential problems in the design and that invites you to run the command `check_design` (or select Design > Check Design... from the main menu) for more information.

Running the `check_design` command after the compilation has finished gives the following output on the console terminal:

```
Warning: In design 'alu32dp_dw01_sub_0', port 'CI' is not connected to any nets. (LINT-28)
Warning: In design 'alu32dp_dw01_sub_0', port 'CO' is not connected to any nets. (LINT-28)
Warning: In design 'alu32dp_dw01_add_0', port 'CI' is not connected to any nets. (LINT-28)
Warning: In design 'alu32dp_dw01_add_0', port 'CO' is not connected to any nets. (LINT-28)
```

The messages above come from the inference of arithmetic components from the use of arithmetic operators in the RTL VHDL sources. The predefined components have a full set of ports to support all possible cases, but the carry-in (CI) and carry-out (CO) ports are not used in our case.

---

[1] Note that also checking the "Ungroup all" box will remove all hierarchy. By hierarchy it is meant all VHDL components and all inferred resource operators.

[2] The "Exact map" box affects how sequential elements (flip-flops, latches) are mapped in the final design. This works with other synthesis directives which are not discussed here.
Alternatively, it is possible to run the `compile` command from the console terminal:

```
> compile -map_effort medium -area_effort medium
```

It is now good to save the mapped design. Select `File > Save As...` in the main menu. Save the mapped design in the DDC format in the path `DB/alu32top_clk10ns_mapped.ddc`[1]. Or, alternatively from the console terminal:

```
> write -hierarchy -format ddc -output DB/alu32top_clk10ns_mapped.ddc
```

It is possible to display the mapped schematics by selecting a component in the hierarchy view[2] and then clicking on the `Create Design Schematic` icon.[3] Let’s do it for the `I_DP` instance. The mapped design now includes real cells from the cell library.

The three generic instances `add_29`, `sub_32` and `mult_35` that were inferred during elaboration[3] from the VHDL source `alu32dp_rtl.vhd` are now mapped to a real adder, subtractor, and multiplier, respectively. Displaying the schematic of the `add_29` component shows that a ripple-carry architecture is used (the red rectangle cells are 1-bit full adders). This is the slowest possible implementation as the timing constraints (clock period) are not too tight (see also Section 3.8.4).

[1] It is good practice to indicate the main constraint(s) (here the 10ns clock) in the file name. Generally speaking, it is important to be able to identify the results between synthesis runs using different constraints.  
[2] The `I_RF` instance is still not displayed in the hierarchy view as it does not contain any cell.  
[3] The actual numbers may be different.
3.8 Generating Reports

It is possible to get many reports on various synthesis results. Here we consider only a few significant reports. Run the following help command in the console terminal to get a list of all report commands:

```
$design_vision> help report*
```

3.8.1 Report on All Violated Constraints

Select Design > Report Constraints... in the main menu. Check "Show all violators" and write the report to the RPT/alu32top_clk10ns_mapped_allviol.rpt file path.

Click OK. The console terminal then shows the following:

```
max_area

Design   Required Area   Actual Area         Slack
alu32top  0.00            32848.34       -32848.34 (VIOLATED)
```

This is not surprising as the area constraint is not realistic.

Alternatively, it is possible to run the `report_constraint` command from the console terminal:

```
$design_vision> report_constraint -nosplit -all_violators  
> RPT/alu32top_clk10ns_mapped_allviol.rpt
```
3.8.2 Area Report

To get a report of the area used by the mapped design, select Design > Report Area.... in the main menu. Save the report in the path RPT/alu32top_clk10ns_mapped_area.rpt as well as in the report viewer.

Click OK.

Alternatively, it is possible to run the report_area command from the console terminal:

```
$ design_vision> report_area > RPT/alu32top_clk10ns_mapped_area.rpt
```

A new tab window in the GUI and the console terminal display the report:

**Library(s) Used:**

- fsd0a_a_generic_core_ss0p9v125c (File: .../ALU32/SNPS_DC/DB/slow.db)
- SYAA90_16X32X1CM2_WC (File: .../ALU32/SNPS_DC/DB/SYAA90_16X32X1CM2_WC.db)

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of ports</td>
<td>166</td>
</tr>
<tr>
<td>Number of nets</td>
<td>272</td>
</tr>
<tr>
<td>Number of cells</td>
<td>70</td>
</tr>
<tr>
<td>Number of references</td>
<td>9</td>
</tr>
<tr>
<td>Combinational area</td>
<td>21438.480032</td>
</tr>
<tr>
<td>Noncombinational area</td>
<td>11349.493600</td>
</tr>
<tr>
<td>Net Interconnect area</td>
<td>undefined</td>
</tr>
<tr>
<td>Total cell area</td>
<td>32787.973632</td>
</tr>
<tr>
<td>Total area</td>
<td>undefined</td>
</tr>
</tbody>
</table>

The area unit depends on the cell library. The most common case, as here, is to consider square microns. The net interconnect area is, for this particular cell library, not defined as the supplied wire load models do not define any associated area. The actual net area will be known after the place+route step.
3.8.3 Critical Path Report

The critical path in a design is the path of (combinational) logic for which signals take the longest time to propagate. The default timing report gives the most critical path in the design and considers setup time constraints.

To get the report on the most critical path, select Timing > Report Timing Path... in the main menu.

Save the report in the path RPT/alu32top_clk10ns_mapped_timing.rpt.

Click OK.

The console terminal then displays the following report (times are in ns):

```
****************************************
Report: timing
-path full
-delay max
-max_paths 1
Design: alu32top
Version: B-2008.09-SP3
Date: Fri Oct  23 10:43:07 2009
****************************************

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

Startpoint: op2_reg_reg[1]
(rising edge-triggered flip-flop clocked by clk)
Endpoint: I_SPLIT/din_reg_reg[63]
(rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: max
```
Alternatively, it is possible to run the `report_timing` command from the console terminal:

```
design_vision> report_timing > RPT/alu32top_clk10ns_mapped_timing.rpt
```

The starting point of the most critical path is the clock signal of the flip-flop storing the bit 11 of the `op1` operand. The path then goes through the multiplier component in the `I_DP` instance (datapath) and the `I_SPLIT` instance. The end point of the path, at which the `data arrival time` is computed, is the input of the flip-flop storing the bit 63 in the `I_SPLIT` instance. The `data required time` is computed as the clock period minus the setup time of the flip-flop at the end of the path. The `slack` is the difference between the required time and the arrival time. The slack is the timing margin. A negative slack value indicates a violation.
At this point, a few remarks are in order:

• The timing computations only consider approximate values for interconnect delays. Accurate values will be only known after the place+route step.

• The synthesis tool does not necessarily determine a gate-level netlist that maximizes the slack. A small positive slack (say around 10% of the clock period) is fine.

• A slack of 0.0 or a small negative slack (say around 10% of the clock period) may be recovered later during the place+route step as the latter can also perform further timing optimizations.

• In the present case, the slack of 0.0 should not be taken too literally. It means that, *given the inferred netlist*, the real design would not work properly *if the critical path is actually used*. Design Compiler offers many options to improve the slack, but this goes beyond the context of this document. Another possible way to improve the slack is to (reasonably) overconstrain the clock period (see Section 3.12).

To highlight the critical path on the schematic, select the `alu32top` component in the hierarchy view, display the schematic (select `Schematic > New Design Schematic View` in the main menu), then select `Timing > Path Slack...` in the main menu. Then click `OK` in the dialog window that appeared. The GUI now displays the critical slacks in the design as an histogram:

![Critical Path Slack Histogram](image)

Select the first bar on the left to get the list of signal paths that have the most critical slacks. Then select the first line in the `HistList` tab\(^1\). Go to the `Schematic` tab to see the corresponding critical path highlighted (in red here; see next page).

Double-clicking on a component symbol let you descend into it. You need to re-select the first line in the `HistList` tab to highlight the critical path at that level.

---

\(^1\) Note that the slack value given here is more accurate than in the timing report.
3.8.4 Resource Usage Report

A resource is an arithmetic or comparison operator read in as a part of the VHDL RTL model. Resource sharing can be detected during the execution of the compile command.

Select Design > Report Design Resources... in the main menu.

Check "Show resources for subdesigns".

Write the report the the file path RPT/alu32top_clk10ns_mapped_resources.rpt.

Click OK.
The following extract of the report shows useful information:

```
***************************************
Design: alu32ctrl
***************************************
Resource Sharing Report for design alu32ctrl in file ./HDL/RTL/alu32ctrl_rtl.vhd

<table>
<thead>
<tr>
<th>Resource</th>
<th>Module</th>
<th>Parameters</th>
<th>Contained Resources</th>
<th>Contained Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>r313</td>
<td>DW01_inc</td>
<td>width=4</td>
<td></td>
<td>add_69 add_74</td>
</tr>
</tbody>
</table>

No implementations to report
No multiplexors to report...

***************************************
Design: alu32dp
***************************************
Resource Sharing Report for design alu32dp in file ./HDL/RTL/alu32dp_rtl.vhd

<table>
<thead>
<tr>
<th>Resource</th>
<th>Module</th>
<th>Parameters</th>
<th>Contained Resources</th>
<th>Contained Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>r318</td>
<td>DW01_add</td>
<td>width=32</td>
<td></td>
<td>add_29</td>
</tr>
<tr>
<td>r320</td>
<td>DW01_sub</td>
<td>width=32</td>
<td></td>
<td>sub_32</td>
</tr>
<tr>
<td>r940</td>
<td>DW_mult_tc</td>
<td>a_width=32</td>
<td></td>
<td>mult_35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b_width=32</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Implementation Report

<table>
<thead>
<tr>
<th>Cell</th>
<th>Module</th>
<th>Current Implementation</th>
<th>Set Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_29</td>
<td>DW01_add</td>
<td>rpl</td>
<td></td>
</tr>
<tr>
<td>mult_35</td>
<td>DW_mult_tc</td>
<td>apparch</td>
<td></td>
</tr>
<tr>
<td>sub_32</td>
<td>DW01_sub</td>
<td>rpl</td>
<td></td>
</tr>
</tbody>
</table>

No multiplexors to report

The "Contained Operations" columns indicate the operators that are inferred from the VHDL RTL model and the line number in the VHDL file where the operators have been detected. The sharing reports indicate if resource sharing is possible (for example, the resource r313, an incrementor, is shared between the add_69 and add_74 operators in the alu32ctrl component; this is possible since the operators are used in different clock cycles). The "Current Implementation" column indicate which one of the available architectures has been selected (for example, the resource r318, an adder, is implemented as a ripple carry architecture (rpl), while the resource r940, a multiplier, is implemented as a parallel architecture (apparch)). The architectures are provided by the Synopsys DesignWare (DW) libraries. They are selected to meet the design constraints.

Alternatively, it is possible to run the `report_resources` command from the console terminal:

```
design_vision> report_resources -nosplit -hierarchy
> RPT/alu32top_clk10ns_mapped_resources.rpt
```

[1] The actual line numbers may be different.
3.9 Generating Verilog Gate-Level Netlist and SDF Timing File

This step creates the Verilog model of the mapped netlist. The model can be used for post-synthesis simulation (Section 2.3) as well as the input to the place+route tool[1]. This step also generates SDF (Standard Delay Format) files that include the gate delays and an approximation of net delays. Care should be taken to use the right naming scheme when generating the SDF file, otherwise the backannotation of the delays into the Verilog netlist for simulation will fail.

Before generating the Verilog netlist, it is required to check that appropriate Verilog naming rules are met in the synthesized design. If not, names have to be changed. This is done by running the `change_names` command in the console terminal (there is no equivalent menu command; be sure that the `alu32top` entity is selected in the hierarchy window):

```
design_vision> change_names -rules verilog -hierarchy -verbose
```

The `-verbose` option has the effect of displaying the changes in the console.

Now, the Verilog model of the synthesized gate-level netlist can be generated.

Select `File > Save As...` in the main menu.

Generate the Verilog file in the path `HDL/GATE/alu32top_clk10ns_mapped.v`.

Select the `VERILOG` format and check the option "Save all designs in hierarchy".

Click `Save`.

Alternatively, it is possible to run the `write` command from the console terminal:

```
design_vision> write -format verilog -hierarchy -output HDL/GATE/alu32top_clk10ns_mapped.v
```

To generate a SDF file with timings to be used with the VHDL model, run the `write_sdf` command from the console terminal[2] to generate the file in the path `TIM/alu32top_clk10ns_mapped_vlog.sdf`:

```
design_vision> write_sdf -version 2.1 TIM/alu32top_clk10ns_mapped_vlog.sdf
```

When generating a SDF file, the console displays the following message:

```
Information: Annotated 'cell' delays are assumed to include load delay.
```

This means that the estimated net delay values have been added to the existing cell delays[3].

[1] It is also possible to generate a VITAL VHDL model.
[2] There is no equivalent menu item in the graphical interface.
[3] More specifically, the INTERCONNECT specifications in the generated SDF file have zero values.
3.10 Generating Design Constraints for Place+Route

Both design environment and design constraint definitions may be stored in a format that can be read later by other EDA tools such as Synopsys PrimeTime or Cadence Encounter. The `write_sdc` command creates a new file that includes the design constraints that have been defined for synthesis in Tcl format:

```
design_vision> write_sdc -nosplit SDC/alu32top_clk10ns_mapped.sdc
```

In our case, the file includes the following information:

```
set sdc_version 1.8
set_units -time ns -resistance kOhm -capacitance pF -voltage V -current mA
set_max_area 0
create_clock [get_ports clk] -period 10 -waveform {0 5}
```

It is important to do that step after the Verilog naming rules have been applied to the mapped design (see Section 3.9), otherwise there could be discrepancies on port/signal names between the netlist and the constraint file.

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.

All the commands executed in this chapter can be grouped in a script file. The tutorial installation created the file named `alu32top_syn.tcl` in the `BIN` directory whose content is given in Annex B.

To run a Tcl script, execute the `source` command in the console terminal as follows:

```
design_vision> source BIN/alu32top_syn.tcl
```

The source command may specify two options:

- `echo` Echoes each command in the script as it is executed
- `verbose` Displays the result of the execution of each command in the script

It is also possible to execute a DC Tcl script from the Linux shell as follows:

```
%ALU32/SNPS_DC> dc_shell -f BIN/alu32top_syn.tcl
```
3.12 Design Optimization Using Tighter Constraints

It is possible to let the synthesizer infer faster architectures for the arithmetic operators, e.g., carry look-ahead architectures, 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 slower clock.

1. Read the elaborated design. It is not necessary to re-analyze the VHDL sources.
2. Specify a clock with a 5ns period (or less).
3. Map and optimize the design.
4. Save the mapped design in the file DB/alu32top_clk5nsMapped.ddc (use a different name for a different clock constraint).
5. Get the new violators, area, timing and resources reports. Compare with the reports you got for the 10 ns clock constraint.
6. Generate the Verilog gate-level netlist in HDL/GATE/alu32top_clk5nsMapped.v and the associated SDF timing data file in TIM/alu32top_clk5nsMapped_vlog.sdf.
7. Save the design constraints for placement and routing in the file SDC/alu32top_clk5nsMapped.sdc.
8. Do a post-synthesis logic simulation.
Chapter 4: Standard Cell Place and Route

This chapter presents the main steps to perform the placement and the routing of the synthesized gate-level netlist using the Encounter tool from Cadence Design Systems.

4.1 Getting Started

To start the Encounter environment, go to the CDS_SOCE directory and run the encounter command:

```
%ALU32> cd CDS_SOCE
%ALU32/CDS_SOCE> encounter
```

The effect of the command is to open two windows: one console terminal and the Encounter GUI.[1]

[1] This is specific to the EPFL setup. When exiting Encounter (Design > Exit), the console terminal must be closed separately (click on the upper right X window box).
The console terminal is useful for entering Encounter commands, to source Tcl scripts, or to run some Unix commands. The tutorial will show GUI commands, but not all commands are available in the GUI. For complex and or repetitive place+route tasks, it is anyway highly recommended to use scripts as discussed in Section 4.13.

The Unix shell from which the tool is started is called the Encounter console. The console displays the `encounter>` prompt. This is where you can enter all Encounter text commands and where the tool displays messages. If you use the console for other actions, e.g., Linux 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.

When the AutoQuery box is enabled, the properties of the object below the cursor are automatically displayed.

To get help on the Encounter tool, you can run the `cdnshelp` command in the Linux shell or click on the Help menu in the main window.

All commands issued in a session are recorded in the `encounter.cmdX` file in the working directory, where `X` is an integer increasing from 0. The `encounter.logX` file contains the complete log of a session (actually, what has been displayed in the Encounter console).
4.2 Importing the Design

4.2.1 Creating the Configuration File

Importing the design involves the specification of a number of information that will be stored in a configuration file. Once a configuration file exists, the import resumes to loading that file.

The configuration information includes:
- **Design libraries and files.** This includes information on the technological process, such as metal and via layers and via generate rules, and the cell library in the LEF (Layout Exchange Format) format. The LEF format provides a way to define an abstract view of the full layout as the place+route steps does not need the full layouts of the standard cells and IP blocks. They also provide the minimum abstract information on cell layouts for placement and routing such as geometrical aspect, I/O pin placements and optionally blockages restricting over-the-cell routing. This information is provided by the standard cell provider.
- **Gate-level netlist.** This relates to the (Verilog) netlist to be placed and routed. The file has been generated during synthesis (Chapter 3, Section 3.9).
- **Timing libraries.** This includes information on the cell timings (delays, setup/hold times, etc.). A max (min) library is used for validating setup (hold) time constraints. Timing libraries are those used by the Synopsys Design Compiler tool, but in their textual Liberty (.lib) format. They are provided by the standard cell provider.
- **Timing constraints.** This relates to the constraints used for RTL synthesis. The file has been generated during synthesis (Chapter 3, Section 3.10). Actually, only the clock period is considered for place+route.
- **Power information.** This relates to the power nets to use in the layout. Power net names are those defined in the LEF technology file.

To create the configuration file for the design, select **File > Import Design.** in the main menu.

Then, click on the **Load** button and load the **CONF/1ms90_std.conf** file. This file defines a basic configuration for the UMC 90nm process.

This information still needs to be completed with data specific to the design.

**Verilog Netlist**

The first information to add to the configuration is the Verilog netlist to place and route. The netlist has been generated by the RTL synthesis tool (Chapter 3).

Click on the **...** button in the **Verilog Netlist area.** Open the **Netlist Selection** area by clicking on the open folder icon on the top right. Select the
file `HDL/GATE/alu32top_clk10ns_mapped.v` and click the Add button to add it to the Netlist Files area. Finally, click on the Close button.

The Verilog Netlist area in the Design Import window now specifies the gate-level netlist to import. Select the Auto_Assign box to let the tool extract the top module name from the Verilog file[1].

Register File Data

The register file will be considered as a macro block during the place+route phase. We need to specify its LEF definition for allowing placing and routing it properly. We also need to specify its timing characteristics as the place+route phase will be able to further optimize the timings with respect to the given timing constraints.

Click on the ... button on the right of the LEF Files field. Add the `../IP/RF16X32/SYAA90_16X32X1CM2.lef` path. Click on the ... button on the right of the Max Timing Libraries field. Add the `../IP/RF16X32/SYAA90_16X32X1CM2_WC.lib` (worst-case timing values) path. Click on the ... button on the right of the Min Timing Libraries field. Add the `../IP/RF16X32/SYAA90_16X32X1CM2_BC.lib` (best-case timing values) path.

Timing Constraints

Click on the ... button on the right of the Timing Constraint File field. Select the file `../SNPS_DC/SDC/alu32top_clk10ns_mapped.sdc` (see Chapter 3, Section 3.10)[2]. The dialog window is similar to the one used for specifying the Verilog netlist file. Click on the Close button to come back to the Design Import window.

I/O Pin Placement

Since the alu32top design is intended to be a block in a larger design, there is no need to place I/O pads. However, it is good to control the locations of I/O pins[3]. The alu32top.io file in the CONF directory is provided as an example of the definition of I/O pin locations.

Click on the open folder icon on the right of the IO Assignment File field. Select the CONF/alu32top.io file and click Open.

---

[1] If the file contains more than one design, i.e., more than one top module name, you need to specify the name of the top module explicitly by checking the By_User box and giving the module name.
[2] It is possible to use a modified, slightly overconstrained, version of the SDC file for place+route.
[3] The tool can anyway define the I/O pin locations automatically.
4.2.2 Saving the Configuration File

The Design Import window should now have the following form:

Click **Save** and save the configuration in the `CONF/alu32top_clk10ns.conf` file. In next Encounter sessions, it will be possible to start with loading this file directly.

Click **OK**. The console terminal displays the progress of the import. Examine the console for errors. Warnings here can be safely ignored.

4.2.3 Importing and Restoring the Design

Note that once a design has been imported as above, it is no more possible to re-import it, or import another design, the same way. You’d need to quit Encounter and start a new session. However, it is always possible to restore a state of a design any time in the same session. It is then good to save a first (import) state of the design as follows.

Select **File > Save Design...** in the main menu.

Select the **Encounter** Data Type. Save the current state as the `alu32top_clk10ns-import.enc` file in the `DB` directory.

To restore a design, select **File > Restore Design...** in the main menu and select the proper file with the `.enc` extension in the directory `DB`. 
4.2.4 Equivalent Text Commands

Once the complete configuration file exists, it is possible to run the following commands in the console terminal:

```
loadConfig CONF/alu32top_clk10ns.conf
setDrawView fplan
fit
saveDesign DB/alu32top_clk10ns-import.enc
```

4.3 Floorplanning the Design

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), the area for power rings, the (pre)placement of blocks/macros, and the location of the corner cells (if required). After the design import, an initial floorplan is displayed in the display area.

The objects on the left of the core area are unplaced modules that can be moved and reshaped. The objects on the right of the core area are unplaced blocks that can only be moved. Too small modules or blocks, such as the I_CTRL module, are not displayed. Modules display a target utilization (TU) value that represents their physical design size. Left-clicking on a module or block displays the pins and connection flight lines, that is the connections and number of connections between the selected module or block to any other modules and blocks.
4.3.1 Refining the Floorplan

Select **Floorplan > Specify Floorplan...** in the main menu. In the **Basic** tab:

- Specify an aspect ratio of 1 (square form) for the core.
- Specify a core utilization of **80%**. This means that 20% of the core area will be free for possible power stripes (Section 4.4.3), buffer insertion or cell replacement (up/down sizing)[1].
- Specify the core margin as the core to IO boundary and specify a distance of 6 microns. The power rings around the core will be a pair of VCC/GND rings of 2 microns each ands separated by a distance of 0.6 microns.

Click **OK**. The console terminal shows that some core to IO boundary distances have been adjusted to fit the grid. The design display area shows the updated floorplan.

4.3.2 Placing the Macro Block

The **I_RF** block is going to be placed in its final position in the core.

Where to place the block and its orientation in the core area depends on the power and IO pin locations in the block and around the core area. The former have been defined by the memory compiler used to generate the block. The latter have been defined in the **alu32top.io** file that is used in the design configuration. The goal is to minimize the wire lengths and still allow proper power connections to the core power rings.

---

[1] Small designs need a much smaller core utilization of about 60% as the cell areas become large relatively to the core area.
Select Floorplan > Automatic Floorplan > Plan design... in the main menu. In the planDesign tab, check the Constraints box and specify the CONF/alu32top.mplan file[1]. The file defines the orientation (270° rotation) and the location (bottom right) of the block in the core area (Annex C, Section C.3).

In the setPlanDesignMode tab, check the Fix Placed Macros box. This will set the block as fixed after the floorplaning step is finished. Forgetting to do that may cause the block to move during the cell placement (Section 4.5).

Click OK. The I_RF block is now placed as planned.

It remains however to finish the floorplan by defining a placement halo around the block. This will prevent placing cells too closely to the block.

Select Floorplan > Automatic Floorplan > Finish Floorplan... in the main menu. Check only the Halo and the Add Macro Halo boxes. Click OK. This adds a fixed-width placement halo wherever needed around the I_RF block according to the pin density.

Select File > Save Design... in the main menu and save the current state as the alu32top_clk10ns-fplan.enc file in the DB directory.

4.3.3 Equivalent Text Commands

```
floorPlan -r 1 0.80 6 6 6 6
setPlanDesignMode -fixPlacedMacros true
planDesign -constraints CONF/alu32top.mplan
finishFloorplan -autoHalo -autoBlockage
saveDesign DB/alu32top_clk10ns-fplan.enc
```

[1] The alu32top.mplan file has been created during the tutorial installation.
4.4 Power Planning

This step defines the power structure of the design, namely power and ground rings and stripes around the core and or blocks. It can also add a pad ring[1] and route all power nets.

4.4.1 Connecting Global Nets

This step assigns pins or nets to global power and ground nets. The imported Verilog netlist does not include 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 PPower > Connect Global Nets... in the main menu.

![Global Net Connections]  

For each **VCC** and **GND** global net:
- Specify the **VCC** or **GND**[2] pin name in the **Connect** area.
- Specify the same pin name in the **To Global Net** field.
- Check the **Verbose Output** box.
- Click the **Add to List** button. The **Connection List** area now includes the global connections to perform.

Click **Apply** and then **Close**. The console terminal now includes a report on the number of global connections made[3]:

The power planner created 2 wires

2264 new pwr-pin connections were made to global net 'VCC'.
2264 new gnd-pin connections were made to global net 'GND'.

[1] Inserting a pad ring is not addressed in this document. It is assumed that the considered design is a block to be included in a larger design.
[2] The ‘VCC’ and ‘GND’ names are those used in the loaded LEF files (Section 4.2.1).
[3] Your actual numbers may be different.
4.4.2 Adding Power Rings

This step generates the VCC and ground power rings around the core.

Select Power > Power Planning > Add Ring... in the main menu.

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

In the Ring Configuration area, define ring widths of 2 micron spaced by 0.6 micron. By default, horizontal power wires will be in metal1 and vertical power wires will be in metal2.

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

In the Advanced tab, define extensions of ring segments up to the I/O pin boundary by clicking on the shown segments. Then, click Apply to leave the tab and come back to the Basic tab.

Click OK. The power rings now appear in the design display area.

---

[1] Specifying “VCC GND VCC” would create three power rings with the GND ring in between two VCC rings.

[2] Click the Update button to get minimum spacing values according to the technology.
4.4.3 Adding Power Stripes (optional)

This step optionally adds a number of vertical and/or horizontal power stripes across the core. Stripes ensure a proper power distribution in large cores.

To add stripes, select **Power > Power Planning > Add Stripe...** in the main menu.

In the **Basic** tab, the **Net(s)** field defines the pattern from left to right (vertical stripes). Here a single pattern will be generated (select **Number of sets** and specify 1). Each stripe will be in metal2, 2 micron wide and the space between them will be 0.6 micron\(^1\).

Check the **Relative from core or selected area** and enter the value 100 (micron) in the **X from left** field. The two stripes will be vertically placed near the center of the core.

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

Click **OK** to generate the power stripes.

If the stripes are badly placed, you can select them and remove them by hitting the delete key. Then you can define another value for the **X from left** field.

Additionally, selecting **Floorplan > Clear Floorplan...** in the main menu allows to delete all or parts of the floorplan objects.

---

\(^1\) Click the **Update** button to get minimum spacing values according to the technology.
4.4.4 Routing Power Nets

Now, it is possible to route the power structures. Select Route > Special Route... in the main menu.

In the Advanced tab, check the Pin facing box in the Pin selection area.

Then check the Bottom side and the Right side boxes.

This will connect the power rings inside the I_RF block to the power rings of the core, but only on the bottom and right sides of the block (which are close to the core rings).

Click OK to do the routing.

The design display area now shows the result of the power routing. Note that the power connections of I_RF block have been forced to be done on the side of the power ring.
Select File > Save Design... in the main menu and save the current state as the alu32top_clk10ns-power.enc file in the DB directory.

### 4.4.5 Equivalent Text Commands

```plaintext
addRing \n- around core \n- nets { GND VCC } \n- center 1 \n- width_bottom 2 -width_right 2 -width_top 2 -width_left 2 \n- spacing_bottom 0.6 -spacing_right 0.6 -spacing_top 0.6 -spacing_left 0.6 \n- layer_bottom ME1 -layer_right ME2 -layer_top ME1 -layer_left ME2 \n- bl 1 -br 1 -rb 0 -rt 0 -tl 0 -lt 1 -lb 1
addStripe \n- nets { GND VCC } \n- layer ME2 \n- width 2 -spacing 0.6 \n- number_of_sets 1 \n- xleft_offset 10.0
sroute \n- connect { blockPin corePin floatingStripe } \n- blockPin { onBoundary bottomBoundary rightBoundary } \n- allowJogging 1
saveDesign DB/alu32top_clk10ns-power.enc
```

### 4.5 Placing the Standard Cells

This step places the standard cells in the rows, according to the imported Verilog netlist.

Select Place > Place Standard Cells... in the main menu.

Then click the Mode button.

Check the Run Timing Driven Placement box[1].

Click OK to return to the Place window.

Click OK in the Place window to run the placement. It may take some time to complete.

To see the result of the placement in the design display area, click the Physical view icon .

[1] This is only effective if a SDC constraint file and timing files have been loaded in the configuration.
There are still a number of holes in the rows since we specified a core utilization of 80%.

Select **Place > Check Placement...** in the main menu to get information on the current placement:

*info: Placed = 2210
*info: Unplaced = 0
Placement Density: 80.29% (23844/29698)

### 4.5.1 Pre-CTS Timing Optimization

It is now possible to do a further timing optimization on the placed design (a.k.a. pre-CTS, pre Clock Tree Synthesis, stage).

Select **Optimize > Optimize Design ...** in the main menu.

Check the **Pre-CTS** box. The other default selection of boxes asks to correct setup, max capacitance and max transitions violations.

Click **OK** to run the optimization. A first trial routing is done.

---

[1] Uncheck the **Check Placement report to** button unless you may want to keep the report in a file (the recommended location is then in the RPT/alu32top_clock10ns directory).
After some time, the console terminal gives a status on the timing performances so far\[^{[1]}\]:

*** Timing Is met
*** Check timing (0:00:00.1)
Reported timing to dir RPT/alu32top_clk10ns
**optDesign ... cpu = 0:00:14, real = 0:00:15, mem = 464.3M **

---

**optDesign Final 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>0.087</td>
<td>0.087</td>
<td>8.601</td>
<td>N/A</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>N/A</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>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>All Paths:</td>
<td>394</td>
<td>188</td>
<td>276</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>DRVs</th>
<th>Real</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nr nets(terms)</td>
<td>Worst Vio</td>
</tr>
<tr>
<td>max_cap</td>
<td>0 (0)</td>
<td>0.000</td>
</tr>
<tr>
<td>max_tran</td>
<td>0 (0)</td>
<td>0.000</td>
</tr>
<tr>
<td>max_fanout</td>
<td>0 (0)</td>
<td>0</td>
</tr>
</tbody>
</table>

Density: 79.353%
Routing Overflow: 0.00% H and 0.00% V

The worst negative slack (WNS) gives the slack of the critical path in the design. Since it as a positive value here, the timing constraints are (already) met. The core density increased a bit due to changes in some cell strengths\[^{[2]}\].

Select **File > Save Design...** in the main menu and save the current state as the file `alu32top_clk10ns-placed.enc` in the directory `DB`.

### 4.5.2 Equivalent Text Commands

```plaintext
setPlaceMode -timingDriven true
placeDesign -prePlaceOpt
setDrawView place
checkPlace
optDesign -preCTS -outDir RPT/alu32top_clk10ns
saveDesign DB/alu32top_clk10ns-placed.enc
```

\[^{[1]}\] The GUI version of the timing optimize command does not allow to specify a directory for the reports and it creates the directory `timingReports` in the working directory. The file names have the `_preCTS` prefix.

\[^{[2]}\] Standard cells are usually derived in several driving capabilities (strengths) for each logic function.
4.6 Synthesizing a Clock Tree

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 have from the previously placed design is shown below. To get this view, select **Tools > Design Browser...** in the main menu. Then select the net **clk** and highlight it. To only see the net in the core area, click the right part of the select bar to get the Physical View visibility and selectability toggles and deselect the V box[1].

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 specify design specific data.

Select **Clock > Synthesis Clock Tree...** in the main menu. Then, in the Basic tab, click the **Gen Spec...** button.

Add all cells named **BUFCK...** and **INVCK...** to the list of selected cells. These are special buffer and inverter cells for the clock tree. Specify the path of the specification file as **CTS/alu32top_clk10ns_spec.cts**. Click OK.

In the Basic tab, define the Results Directory as **RPT/alu32top_clk10ns**. The directory will be created if it does not yet exist. Click OK in the Synthesize Clock Tree window to synthesize the clock tree.

[1] Re-check the V box to see again all items in the Physical View.
Select **Clock > Display > Display Clock Tree...** to see the result in the design display area. Check the **Clock Route Only** and the **Display Clock Tree** boxes. Click **Apply**. Uncheck the V box in the Physical View to only see the clock tree in the design display area.

Check the **Display Clock Phase Delay** box to see the phase delay of the clock signal. Colors at the blue (red) end of the spectrum indicate the smallest (longest) phase delay.

Click **OK** or **Cancel** to close the **Display Clock Tree** window.

The file **RPT/alu32top_clk10ns/clock.report** gives details on the generated clock tree. It shows that three **INVCK** cells have been added at the source of the clock net. The buffered clock net is then distributed to all flip-flops in the design.
It is possible to see this in the GUI by selecting **Clock > Browse Clock Tree...** in the main menu. Select the clock net **clk** and check the **Clock Route Only** box. Click **Apply** to see the tree.

### 4.6.1 Post-CTS Timing Optimization

It is now possible to do a further timing optimization on the placed design with clock tree (a.k.a. post-CTS, post Clock Tree Synthesis, stage).

Select **Optimize > Optimize Design...** in the main menu. Check the **Post-CTS** box.

After some time, the console terminal gives a status on the timing performances so far. The WNS has been now reduced to 0.077 ns.[1]

Select **File > Save Design...** in the main menu and save the current state as the file **alu32top_clk10ns-cts.enc** in the directory **DB**.

### 4.6.2 Equivalent Text Commands

```
clockDesign -specFile CTS/alu32top_clk10ns_spec.cts -outDir RPT/alu32top_clk10ns
optDesign -postCTS -outDir RPT/alu32top_clk10ns
saveDesign DB/alu32top_clk10ns-cts.enc

createClockTreeSpec -output CTS/alu32top_clk10ns_spec.cts -bufferList BUFCKX1 BUFCKX12 BUFCKX16 BUFCKX1P BUFCKX20 BUFCKX3 
BUFCKX4 BUFCKX6 BUFCKX8 INVCKX1 INVCKX12 INVCKX16 INVCKX1P INVCKX2 INVCKX20 INVCKX3 INVCKX4 INVCKX6 INVCKX8 INVCKXLP
```

[1] Same remark concerning the default report directory as in Section 4.5.1. The file names have now the _postCTS prefix.
4.7 Routing the Design

This step generates all the wires that are required to connect the cells as defined in the imported Verilog netlist.

Select Route > NanoRoute > Route... in the main menu. Check the Timing Driven box. Keep the effort to the default value of 5. A higher value increases the effort toward meeting the timing constraints and decreases the effort toward relieving congestion. Click OK to start the routing.

4.7.1 Post-Route Timing Optimization

It is now possible to do a further timing optimization on the routed design.

Select Timing > Optimize... in the main menu. Check the Post-Route box.

After some time, the console terminal gives a status on the timing performances so far. The WNS has been now reduced to 0.033 ns[1].

Select File > Save Design... in the main menu and save the current state as the file alu32top_clk10ns-routed.enc in the directory DB.

[1] Same remark concerning the default report directory as in Section 4.5.1. The file names have now the _postRoute prefix.
The design display area now shows the routed design:

4.7.2 Equivalent Text Commands

```bash
setNanoRouteMode -routeWithTimingDriven true -routeTdrEffort 5
routeDesign
optDesign -postRoute -outDir RPT/alu32top_clk10ns
saveDesign DB/alu32top_clk10ns-routed.enc
```
4.8 Design Finishing

4.8.1 Adding Filler Cells

Filler cells will fill remaining holes in the rows and ensure the continuity of power/ground rails and N+/P+ wells.

Select Place > Physical Cell > Add Filler... in the main menu. Click the Select button to select the available filler cells in the cells list and add them to the selectable cells list. Click Close to return to the Add Filler window. Uncheck the Mark Fixed box[1]. Click OK to start adding the filler cells.

The following information is displayed in the console:

*INFO: Added 20 filler insts (cell FILLERD64 / prefix FILLER).
*INFO: Added 18 filler insts (cell FILLERD32 / prefix FILLER).
*INFO: Added 95 filler insts (cell FILLERD16 / prefix FILLER).
*INFO: Added 167 filler insts (cell FILLERD8 / prefix FILLER).
*INFO: Added 369 filler insts (cell FILLERD4 / prefix FILLER).
*INFO: Added 260 filler insts (cell FILLERD3 / prefix FILLER).
*INFO: Added 609 filler insts (cell FILLERD1 / prefix FILLER).
*INFO: Total 1907 filler insts added - prefix FILLER (CPU: 0:00:00.1).
1907 new pwr-pin connections were made to global net 'VCC'.
1907 new gnd-pin connections were made to global net 'GND'.

Select File > Save Design... in the main menu and save the current state as the file alu32top_clk10ns-filled.enc in the directory DB.

The equivalent text commands are:

```
addFiller \
  -cell { FILLERD8 FILLERD64 FILLERD4 FILLERD32 FILLERD3 FILLERD2 \
         FILLERD16 FILLERD1 } \
  -prefix FILLER
setDrawView place
saveDesign DB/alu32top_clk10ns-filled.enc
```

[1] This will allow some further changes such as fixing antenna violations.
4.9 Checking the Design

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

4.9.1 Verify connectivity

Select Verify > Verify Connectivity... in the main menu.
Specify the report file path as RPT/alu32top_clk10ns/connectivity.rpt.
Click OK.

The console terminal displays the result of the connectivity check:

Design Name: alu32top
Database Units: 1000
Design Boundary: (0.0000, 0.0000) (210.7800, 208.1600)
Error Limit = 1000; Warning Limit = 50
Check all nets
Net clk: Found a geometry with bounding box (-0.07,0.00) (0.07,0.14) outside design boundary.
Violations for such geometries will be reported.
Net VCC: unconnected terminal, special open.
Net GND: unconnected terminal, special open.

Begin Summary
19 Problem(s) [ 96]: Terminal(s) are not connected.
2 Problem(s) [200]: Special Wires: Pieces of the net are not connected together.
21 total info(s) created.
End Summary

The 19 problems reporting not connected terminals relate to unconnected VCC/GND pins in the I_RF block. Remember it has been a decision to only connect power pins on the right and bottom sides of the block (Section 4.4.4). The internal power rings in the block should be well connected anyway.
The 2 other problems relate to the power rings around the core and to the previous 19 unconnected power pins. You can safely clear the violations by selecting Tools > Violation Browser in the main menu and then click on Clear Violation.
4.9.2 Verify Geometry

Now select Verify > Verify Geometry... in the main menu.
In the Advanced tab, Specify the report file path as RPT/alu32top_clk10ns/geometry.rpt.
Click OK.

The console terminal displays the result of the connectivity check:

VERIFY GEOMETRY ...... Starting Verification
VERIFY GEOMETRY ...... Initializing
VERIFY GEOMETRY ...... Deleting Existing Violations
VERIFY GEOMETRY ...... Creating Sub-Areas
...... bin size: 1920
VERIFY GEOMETRY ...... SubArea : 1 of 1
VERIFY GEOMETRY ...... Cells : 0 Viols.
VERIFY GEOMETRY ...... SameNet : 0 Viols.
VERIFY GEOMETRY ...... Wiring : 0 Viols.
VERIFY GEOMETRY ...... Antenna : 0 Viols.
VERIFY GEOMETRY ...... SubArea: 1 complete 0 Viols. 0 Wrngs.
VG: elapsed time: 2.00
Begin Summary ...
  Cells : 0
  SameNet : 0
  Wiring : 0
  Antenna : 0
  Short : 0
  Overlap : 0
End Summary

The equivalent text commands are:

```shell
verifyConnectivity -type all -report RPT/alu32top_clk10ns/connectivity.rpt
verifyGeometry -report RPT/alu32top_clk10ns/geometry.rpt
```
4.10 Generating Reports

A number of reports have been already generated in the previous steps. They should be located in the directory `RPT/alu32top_clk10ns`. The File > Report menu offers the way to generate additional reports.

4.10.1 Netlist statistics

File > Report> Netlist Statistics gives the following output in the console:

```plaintext
*** Statistics for net list alu32top ***
Number of cells = 4165
Number of nets = 2847
Number of tri-nets = 0
Number of degen nets = 0
Number of pins = 9939
Number of i/os = 166

Number of nets with 2 terms = 2356 (82.8%)
Number of nets with 3 terms = 188 (6.6%)
Number of nets with 4 terms = 85 (3.0%)
Number of nets with 5 terms = 32 (1.1%)
Number of nets with 6 terms = 3 (0.1%)
Number of nets with 7 terms = 2 (0.1%)
Number of nets with 8 terms = 12 (0.4%)
Number of nets with 9 terms = 31 (1.1%)
Number of nets with >=10 terms = 138 (4.8%)

*** 81 Primitives used:
Primitive SYAA90_16X32X1CM2 (1 insts)
Primitive XOR4X1P (1 insts)
Primitive XOR3X1 (2 insts)
Primitive XOR2X1P (6 insts)
Primitive XOR2X1 (8 insts)
...```

4.10.2 Gate count

File > Report> Gate Count... gives the following output in the console and in the file `RPT/alu32top_clk10ns/gateCount.rpt`:

Gate area 2.3520 um²

```
[0] alu32top Gates=13953 Cells=2212 Area=32819.7 um²
[1] I_RF Gates=3804 Cells=0 Area=8948.4 um²
[1] I_DP Gates=8693 Cells=1865 Area=20446.7 um²
[2] I_DP/sub_37 Gates=374 Cells=66 Area=880.4 um²
[2] I_DP/add_34 Gates=341 Cells=33 Area=803.6 um²
[2] I_DP/mult_40 Gates=7716 Cells=1681 Area=18148.8 um²
[1] I_SPLIT Gates=727 Cells=228 Area=1709.9 um²
[1] I_CTRL Gates=171 Cells=49 Area=402.2 um²
```
4.10.3 Design summary

File > Report > Summary... allows to generate a text and/or an html report.

If the Text Only box is selected, specify the file path RPT/alu32top_clk10ns/summaryReport.rpt.

If the HTML and Text box is selected, specify the output directory path RPT/alu32top_clk10ns/html.

In the latter case, open the file RPT/alu32top_clk10ns/html/alu32top.main.htm in a browser[1].

4.10.4 Timing

Run the report_timing command in the console terminal to get a report on the critical path:

Path 1: SET Setup Check with Pin I_SPLIT/din_reg_reg_63_/CK
Endpoint: I_SPLIT/din_reg_reg_63_/D (v) checked with leading edge of 'clk'
Beginpoint: op1_reg_reg_8_/Q (^) triggered by leading edge of 'clk'

| Other End Arrival Time | 0.096 |
| - Setup | 0.141 |
| + Phase Shift | 10.000 |
| = Required Time | 9.955 |
| - Arrival Time | 9.871 |
| = Slack Time | 0.084 |

Clock Rise Edge: 0.000
+ Clock Network Latency (Prop): 0.099
= Beginpoint Arrival Time: 0.099

<table>
<thead>
<tr>
<th>Instance</th>
<th>Arc</th>
<th>Cell</th>
<th>Delay</th>
<th>Arrival Time</th>
<th>Required Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>op1_reg_reg_8__</td>
<td>CK ^</td>
<td>QDFERBX1</td>
<td>0.359</td>
<td>0.457</td>
<td>0.542</td>
</tr>
<tr>
<td>I_DP/mult_40/U2013</td>
<td>I ^ -&gt; O v</td>
<td>INVX1</td>
<td>0.144</td>
<td>0.601</td>
<td>0.685</td>
</tr>
<tr>
<td>I_DP/mult_40/U2081</td>
<td>I2 v -&gt; O ^</td>
<td>XOR2X1</td>
<td>0.108</td>
<td>0.709</td>
<td>0.793</td>
</tr>
<tr>
<td>I_DP/mult_40/U2936</td>
<td>I1 ^ -&gt; O v</td>
<td>ND3X1</td>
<td>0.521</td>
<td>1.230</td>
<td>1.314</td>
</tr>
<tr>
<td>I_DP/mult_40/U2169</td>
<td>CI v -&gt; CO v</td>
<td>FA1X2</td>
<td>0.123</td>
<td>9.298</td>
<td>9.382</td>
</tr>
<tr>
<td>I_DP/mult_40/U2613</td>
<td>CI v -&gt; CO v</td>
<td>FA1X1</td>
<td>0.113</td>
<td>9.411</td>
<td>9.495</td>
</tr>
<tr>
<td>I_DP/mult_40/U2646</td>
<td>CI v -&gt; CO v</td>
<td>XORD1X1</td>
<td>0.224</td>
<td>9.635</td>
<td>9.719</td>
</tr>
<tr>
<td>I_DP/U7</td>
<td>B1 v -&gt; O v</td>
<td>AOX12X1</td>
<td>0.146</td>
<td>9.780</td>
<td>9.864</td>
</tr>
<tr>
<td>I_SPLIT/U2</td>
<td>B1 v -&gt; O v</td>
<td>MAO11X1</td>
<td>0.091</td>
<td>9.871</td>
<td>9.955</td>
</tr>
<tr>
<td>I_SPLIT/din_reg_reg_63__</td>
<td>D v</td>
<td>QDFFRBX1</td>
<td>0.000</td>
<td>9.871</td>
<td>9.955</td>
</tr>
</tbody>
</table>

The file RPT/alu32top_clk10ns/alu32top_postRoute_all.tarpt includes the same report. The Encounter tool offers also graphical representations of the timing report.

The equivalent text commands are:

```
reportNetStat
reportGateCount -outfile RPT/${DESIGN_NAME}/gateCount.rpt
summaryReport -outdir RPT/${DESIGN_NAME}/summary
```

[1] The Display HTML option does not seem to work all the times.
4.11 Design Export

4.11.1 Generating the SDF Timing File

Before generating the place+route SDF timing file, the parasitics must be extracted.

Select **Timing > Write_SDF...** from the main menu.

Uncheck **Ideal Clock**.

Specify the SDF output file path as **TIM/alu32top_clk10ns_pared.sdf**.

Click **OK**.

The equivalent text command is:

```
write_sdf TIM/alu32top_clk10ns_pared.sdf
```

4.11.2 Generating the P+R Verilog Netlist

The P+R netlist may be different from the imported netlist as cells may have been added (e.g. buffers) or replaced (e.g. resized) during clock tree synthesis and various timing optimization phases.

Select **File > Save > Netlist...** in the main menu.

Uncheck the **Include Leaf Cell Definition** box.

Specify the netlist file path as **HDL/GATE/alu32top_clk10ns_pared.v**.

Click **OK**.

The equivalent text command is:

```
saveNetlist -excludeLeafCell HDL/GATE/alu32top_clk10ns_pared.v
```

[1] The SDF file is written using the default version 3.0 of the format. The equivalent text command can specify the version 2.1 as well. It can also define the number of digits to use (default is 3).
4.11.3 Generating the GDS2 File

The placed and routed design can be exported in different formats for further processing outside the Encounter tool. The GDS2 binary format is a standard format for integrating the block in the top-level layout, doing DRC/LVS checkings, or delivering the layout to the foundry.

To export the design in the GDS2 format, select **File > Save > GDS/OASIS...** in the main menu.

![GDS Export Interface](image)

Save the GDS2 file in the path `DEX/alu32top_clk10ns_pared.gds`.

The GDS map file has been installed in the directory `DEX`. The library name `alu32top_clk10ns` will be the Virtuoso design library name.

It is also important to merge the GDS2 file of the register file macro. This file has been generated by the Faraday memory compiler and is located in the path: `ALU32/IP/RF16X32/IP/RF16X32/SYAA90_16X32X1CM2.gds`.

Click **OK**.

The console terminal reports on the GDS2 file generation. A few warnings are issued regarding missing master cells in the merged file. These master cells are all standard cells and their full layouts will be included when importing the design in Virtuoso.

The equivalent text command is:

```bash
streamOut DEX/alu32top_clk10ns_pared.gds
  -mapFile DEX/gds2out.map
  -libName alu32top_clk10ns
  -merge ../IP/RF16X32/SYAA90_16X32X1CM2.gds
```
4.12 Importing in Virtuoso

To import the P+R design in the Virtuoso environment, go to the ALU32/CDS_VISO directory and run the icfb command[1]. If it does not yet exist, create a new library called alu32top_clk10ns and attach the technology file umc90nm.

In the icfb log window, select File > Import > Stream... Click the Options button and check the Retain Reference Library (No Merge) option. Since the cell layouts are available in the FDAY_90NM_1D0V_CORE library, it is possible to display the full layout of the block.

The log file alu32top_clk10ns_gdsin.log will be created in the CDS_VISO directory.

The full imported layout is then as follows:

[1] All the required files in the CDS_VISO directory have been created at the tutorial installation.
4.13 Using Scripts

As for the RTL synthesis, 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. Cadence Encounter supports the Tcl language for building scripts.

All the commands executed in this chapter can be grouped in a script file. The tutorial installation created the file named `alu32top_par.tcl` in the BIN directory whose content is given in Annex C, Section C.4.

To run a Tcl script, execute the `source` command in the console terminal as follows:

```
encounter> source BIN/alu32top_par.tcl
```

It is also possible to execute a DC Tcl script from the Unix shell as follows:

```
%ALU32/CDS_SOCE> encounter -init BIN/alu32top_par.tcl -win
```

The `-win` option starts the Encounter GUI.

4.14 Place+Route Using Tighter Constraints

Redo the steps in this chapter to perform the place and route of the design with tighter constraints as synthesized in Chapter 3, Section 3.12.

Create a new configuration file `CONF/alu32top_clk5ns.conf`. The 10ns config file can be copied and edited to use the new proper files, or the full design import step can be done.
Annex A: VHDL Files

This chapter gives the RTL VHDL sources used as the starting point for the steps presented in this document. The VHDL source of the register file is not given here for confidentiality reasons.

A.1 alu32_pkg.vhd

```vhdl
-- alu32_pkg.vhd
--
-- Utility package for 32-bit ALU model
-- Definitions of ALU commands
--

library IEEE;
use IEEE.std_logic_1164.all;

package alu32_pkg is
  constant CMD_SIZE : natural := 2;
  constant ALU_NOOP : std_logic_vector := "00";
  constant ALU_ADD : std_logic_vector := "01";
  constant ALU_SUB : std_logic_vector := "10";
  constant ALU_MULT : std_logic_vector := "11";
end package alu32_pkg;
```

A.2 alu32dp_rtl.vhd

```vhdl
-- alu32dp_rtl.vhd
--
-- 32-bit ALU datapath
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.alu32_pkg.all;

entity alu32dp is
  port (  
    signal cmd : in std_logic_vector(CMD_SIZE-1 downto 0);
    signal op1, op2 : in std_logic_vector(31 downto 0);
    signal res : out std_logic_vector(63 downto 0)
  );
end entity alu32dp;

architecture rtl of alu32dp is
begin
  process (cmd, op1, op2)
  variable tres : signed(31 downto 0);
  begin
    res <= (others => '0');
    case cmd is
      when ALU_ADD =>
        tres := signed(op1) + signed(op2);
        res <= std_logic_vector(resize(tres,res'length));
      when ALU_SUB =>
        tres := signed(op1) - signed(op2);
        res <= std_logic_vector(resize(tres,res'length));
      when ALU_MULT =>
        res <= std_logic_vector(signed(op1) * signed(op2));
    end case;
  end process;
end rtl;
```
when others =>
  null;
end case;
end process;
end architecture rtl;

A.3 alu32ctrl_rtl.vhd

-- alu32ctrl_rtl.vhd
--
-- 32-bit ALU control
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity alu32ctrl is
  port (    signal clk : in std_logic;
           signal rst_b : in std_logic;
           signal start : in std_logic;
           signal load : out std_logic;
           signal ser_en : out std_logic;
           signal rf_cs_b : out std_logic;
           signal rf_we_b : out std_logic;
           signal rf_addr : out std_logic_vector(3 downto 0);
           signal done : out std_logic    );
end entity alu32ctrl;

architecture rtl of alu32ctrl is
  type fsm_states is (    ST_IDLE, ST_START, ST_OP, ST_WR0, ST_WR1, ST_WR2 );
  signal state : fsm_states;
begin
  process (rst_b, clk)
    constant BASE_ADDR : unsigned(rf_addr'range) := (others => '0');
    variable addr : unsigned(rf_addr'range);
  begin
    if rst_b = '0' then
      addr := BASE_ADDR;
      rf_cs_b <= '1';
      rf_we_b <= '1';
      load <= '0';
      ser_en <= '0';
      done <= '0';
      state <= ST_IDLE;
    elsif clk'event and clk = '1' then
      case state is
        when ST_IDLE =>
          rf_cs_b <= '1';
          rf_we_b <= '1';
          load <= '0';
          ser_en <= '0';
          done <= '0';
        when ST_START =>
          if start = '1' then
            load <= '1';
            state <= ST_START;
          end if;
          when ST_START =>
          begin
            case state is
              when ST_IDLE =>
                if start = '1' then
                  load <= '1';
                  state <= ST_OP;
                end if;
              when ST_START =>
                -- load operand registers
                state <= ST_OP;
              when ST_OP =>
                -- 1 clock cycle for operation
                ser_en <= '1';
            end case;
          end when;
      end case;
    end if;
  end process;
end architecture rtl;
state <= ST_SER;
when ST_SER =>
    -- serialize 64-bit datapath result
rf_cs_b <= '0';
rf_we_b <= '0';
rf_addr <= std_logic_vector(addr);
addr := addr + 1;
state <= ST_WR0;
when ST_WR0 =>
rf_addr <= std_logic_vector(addr);
addr := (addr + 1) mod 2**addr'length;
state <= ST_WR1;
when ST_WR1 =>
rf_cs_b <= '1';
state <= ST_WR2;
when ST_WR2 =>
der_en <= '0';
done <= '1';
state <= ST_IDLE;
end case;
en if;
end process;
end architecture rtl;

A.4 alu32split_rtl.vhd

-- alu32ser_rtl.vhd
--
-- 32-bit ALU 64 to 32 bit splitter
--
library IEEE;
use IEEE.std_logic_1164.all;

entity alu32split is
port (   signal clk,
         rst_b,
        enbl : in std_logic;
         signal din : in std_logic_vector(63 downto 0);
         signal dout : out std_logic_vector(31 downto 0)
        );
end entity alu32split;

architecture rtl of alu32split is
signal din_reg : std_logic_vector(63 downto 0);
signal bysel : std_logic;
beginn    process (clk, rst_b)
beginn      if rst_b = '0' then
         din_reg <= (others => '0');
bysel <= '0';
      elsif clk'event and clk = '1' then
         if enbl = '1' then
            din_reg <= din;
bysel <= not bysel;
         end if;
      end if;
end process;
dout <= din_reg(63 downto 32) when bysel = '1' else
      din_reg(31 downto 0);
end architecture rtl;
A.5 regfile_pkg.vhd

-- regfile_pkg.vhd
--
-- Component declaration for register file
--
library IEEE;
use IEEE.std_logic_1164.all;

package regfile_pkg is
  component SYAA90_16X32X1CM2 is
    port (
      DO : OUT std_logic_vector (31 downto 0);
      A : IN std_logic_vector (3 downto 0);
      DI : IN std_logic_vector (31 downto 0);
      WEB : IN std_logic;
      CK : IN std_logic;
      CSB : IN std_logic
    );
  end component;
end package regfile_pkg;

A.6 alu32top_rtl.vhd

-- alu32top_rtl.vhd
--
-- 32-bit ALU top level
--
library IEEE;
use IEEE.std_logic_1164.all;
use work.regfile_pkg.all;
use work.alu32_pkg.all;

entity alu32top is
  port (    
    signal clk : in std_logic;
    signal rst_b : in std_logic;
    signal start : in std_logic;
    signal cmd : in std_logic_vector(CMD_SIZE-1 downto 0);
    signal op1,  
      op2 : in std_logic_vector(31 downto 0);
    signal res : out std_logic_vector(63 downto 0);
    signal rf_do : out std_logic_vector(31 downto 0);
    signal done : out std_logic
  );
end entity alu32top;

architecture rtl of alu32top is
begin
  OP_REGS : process (clk, rst_b)
  begin
    if rst_b = '0' then

op1_reg <= (others => '0');
op2_reg <= (others => '0');
elsif clk'event and clk = '1' then
  if load = '1' then
    op1_reg <= op1;
op2_reg <= op2;
  end if;
end if;
end process;

I_DP : entity work.alu32dp(rtl)
  port map (
    cmd => cmd,
op1 => op1_reg,
op2 => op2_reg,
res => dp_res
  );
res <= dp_res;

I_SPLIT : entity work.alu32split(rtl)
  port map (
    clk => clk,
rst_b => rst_b,
enbl => ser_en,
din => dp_res,
dout => rf_di
  );

I_CTRL : entity work.alu32ctrl(rtl)
  port map (
    clk => clk,
rst_b => rst_b,
start => start,
load => load,
ser_en => ser_en,
rf_cs_b => rf_cs_b,
rf_we_b => rf_we_b,
rf_addr => rf_addr,
done => done
  );

I_RF : component syaa90_16x32x1cm2
  port map (
    DO => rf_do,
A => rf_addr,
DI => rf_di,
WEB => rf_we_b,
CK => clk,
CSB => rf_cs_b
  );

end architecture rtl;
A.7 alu32top_tb.vhd

-- alu32top_tb.vhd
--
-- Testbench for 32-bit ALU top level
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.alu32_pkg.all;
use STD.textio.all;

entity alu32top_tb is
end;

architecture bench of alu32top_tb is

constant CLK_PER : time := 10 ns;

signal clk : std_logic := '0';
signal rst_b,
    start,
    done : std_logic;

signal finished : boolean := FALSE;

signal cmd : std_logic_vector(CMD_SIZE-1 downto 0);
signal op1,
    op2 : std_logic_vector(31 downto 0);

signal res : std_logic_vector(63 downto 0);
signal xpect : signed(63 downto 0);
signal rf_do : std_logic_vector(31 downto 0);

component alu32top is
    port (  
        signal clk : in std_logic;
        signal rst_b : in std_logic;
        signal start : in std_logic;
        signal cmd : in std_logic_vector(CMD_SIZE-1 downto 0);
        signal op1,
            op2 : in std_logic_vector(31 downto 0);
        signal res : out std_logic_vector(63 downto 0);
        signal rf_do : out std_logic_vector(31 downto 0);
        signal done : out std_logic
    );
end component;

begin

UUT : component alu32top
    port map (  
        clk => clk,
        rst_b => rst_b,
        start => start,
        cmd => cmd,
        op1 => op1,
        op2 => op2,
        res => res,
        rf_do => rf_do,
        done => done
    );

-- clock generator
process (clk, finished)
begin
    -- stopping the clock once all possible input values have been scanned
    -- allows for executing a "run -all" command
if not finished then
    clk <= not clk after CLK_PER/2;
end if;
end process;

-- initial asynchronous reset
rst_b <= '1', '0' after CLK_PER/4, '1' after 3*CLK_PER/4;

-- stimulus generator

procedure init is
begin
    start <= '0';
    cmd <= ALU_NOOP;
    op1 <= (others => '0');
    op2 <= (others => '0');
end procedure init;

procedure do_cmd (cmd_p : in std_logic_vector(CMD_SIZE-1 downto 0);
    op1_p, op2_p : in signed(31 downto 0);
    exp_res : in signed(63 downto 0)) is
begin
    wait until clk = '0';
    cmd <= cmd_p;
    op1 <= std_logic_vector(op1_p);
    op2 <= std_logic_vector(op2_p);
    start <= '1';
    wait until clk = '0';
    start <= '0';
    xpect <= exp_res;
    wait until done = '1';
    assert (signed(res) = exp_res);
end procedure do_cmd;

begin
    init;
    do_cmd(ALU_ADD,
        to_signed(5, 32),
        to_signed(555, 32),
        to_signed(5 + 555, 64));
    do_cmd(ALU_ADD,
        to_signed(integer'high/2, 32),
        to_signed(integer'high/2, 32),
        to_signed(integer'high/2 + integer'high/2, 64));
    do_cmd(ALU_NOOP,
        to_signed(0, 32),
        to_signed(0, 32),
        to_signed(0, 64));
    do_cmd(ALU_ADD,
        to_signed(integer'high/2, 32),
        to_signed(-integer'high, 32),
        to_signed(integer'high/2 - integer'high, 64));
    do_cmd(ALU_ADD,
        to_signed(-12, 32),
        to_signed(24, 32),
        to_signed(-12 + 24, 64));
    do_cmd(ALU_MULT,
        to_signed(256, 32),
        to_signed(-256, 32),
        to_signed(256 * (-256), 64));
    do_cmd(ALU_MULT,
        to_signed(123, 32),
        to_signed(999, 32),
        to_signed(123 * 999, 64));
    do_cmd(ALU_SUB,
        to_signed(123456789, 32),
to_signed(123456789, 32),
to_signed(123456789 - 123456789, 64));
do_cmd(ALU_MULT,
to_signed(-12345, 32),
to_signed(999, 32),
to_signed(-12345 * 999, 64));
do_cmd(ALU_ADD,
to_signed(-5, 32),
to_signed(555, 32),
to_signed(-5 + 555, 64));
wait until clk = '0';
wait until clk = '0';
finished <= TRUE;
wait;
end process;
end architecture bench;

A.8  alu32top_tb_rtl_conf.vhd

--  alu32top_tb_rtl_conf.vhd
--
--  Configuration for testbench of 32-bit ALU top level
--

configuration alu32top_tb_rtl_conf of alu32top_tb is
  for bench
    for all : alu32top use entity work.alu32top(rtl); end for;
  end for;
end configuration alu32top_tb_rtl_conf;
Annex B: Synopsys DC Script

This annex includes a simple Tcl script for Synopsys Design Compiler that executes the steps as described in Chapter 3. The beginning of the script defines variables that can be changed for processing a different design or changing constraints.

```tcl
# File : alu32top_syn.tcl
# Description : Simple synthesis script for ALU32 top-level model.
# Notes : The script can be executed in DC shell by running
# source [-echo] [-verbose] path-to-tcl-script
# -echo echoes each command as it is executed
# -verbose displays the result of each command executed
# (error messages are displayed regardless)
# The script can also be run from a Linux shell as follows:
# dc_shell -f path-to-tcl-script
# echo echoes each command as it is executed
# -verbose displays the result of each command executed
# The script can also be run from a Linux shell as follows:
# tools [dc_shell -f path-to-tcl-script]
# Author : Alain Vachoux, EPFL STI IEL LSM, alain.vachoux@epfl.ch
# for EDATP labs.
# Tools : Synopsys DC 2010.12
# design related definitions
# set ENTITY_NAME alu32top
set ARCH_NAME rtl
set CLK_NAME clk
set CLK_PERIOD 10 ;# ns
# set DESIGN_ENTITY "${ENTITY_NAME}_${ARCH_NAME}"
set DESIGN "${ENTITY_NAME}_clk${CLK_PERIOD}ns"
# start from fresh state
# remove design -all
# analyze VHDL sources
# puts "-i- Analyze VHDL sources"
analyze -format vhdl { 
    HDL/RTL/alu32_pkg.vhd 
    HDL/RTL/regfile_pkg.vhd 
    HDL/RTL/alu32dp_rtl.vhd 
    HDL/RTL/alu32ctrl_rtl.vhd 
    HDL/RTL/alu32split_rtl.vhd 
    HDL/RTL/alu32top_rtl.vhd 
}
# elaborate design
# puts "-i- Elaborate design"
elaborate ${ENTITY_NAME} -architecture ${ARCH_NAME} \ 
    -library DEFAULT -update
# save elaborated design
# puts "-i- Save elaborated design"
write -hierarchy -format ddc -output DB/${DESIGN}_elab.ddc
# link design
# puts "-i- Link design"
set link_library "$link_library DB/SYAA90_16X32X1CM2_WC.db"
link
# define constraints
```
# Define constraints
puts "-i- Define constraints"
puts "-i-  set_max_area 0"
puts "-i-  create_clock $CLK_NAME -period $CLK_PERIOD"
puts "-i-  set_fix_multiple_port_nets all"
set_max_area 0
create_clock $CLK_NAME -period $CLK_PERIOD
set_fix_multiple_port_nets all

# Map and optimize design
#
puts "-i- Map and optimize design"
compile -map_effort medium -area_effort medium

# save mapped design
#
puts "-i- Save mapped design"
write -hierarchy -format ddc -output DB/${DESIGN}_mapped.ddc

# generate reports
#
puts "-i- Generate reports"
report_constraint -nosplit -all_violators > RPT/${DESIGN}_allviol.rpt
report_area > RPT/${DESIGN}_area.rpt
report_timing > RPT/${DESIGN}_timing.rpt
report_resources -nosplit -hierarchy > RPT/${DESIGN}_resources.rpt
report_reference -nosplit -hierarchy > RPT/${DESIGN}_references.rpt

# generate Verilog netlist
#
puts "-i- Generate Verilog netlist"
change_names -rules verilog -hierarchy
write -format verilog -hierarchy -output HDL/GATE/${DESIGN}_mapped.v

# generate SDF timing file for Verilog
#
puts "-i- Generate SDF file for Verilog netlist"
write_sdf -version 2.1 TIM/${DESIGN}_mapped_vlog.sdf

# generate design constraint file
#
puts "-i- Generate SDC design constraint file"
write_sdc -nosplit SDC/${DESIGN}_mapped.sdc

puts "-i- Finished"
Annex C: Cadence Encounter Files

This annex includes configuration files for Cadence Encounter used in the tutorial.

C.1 alu32top_clk10ns.conf

This file is generated during the design import (Chapter 4, Section 4.2).

```
# Generated by: Cadence Encounter 09.14-s273_1
# OS: Linux 1686(Host ID lsmpc36.epfl.ch)
# Generated on: Thu Oct 13 16:25:31 2011

global rda_input
set cwd /.../alu32_DEV/CDS_SOCE
set rda_input(import_mode) "-treatUndefinedCellAsBox=0 -keepEmptyModule=1"
set rda_input(ui_netlist) "../HDL/GATE/alu32top_clk10ns_mapped.v"
set rda_input(ui_rttlist) ""
set rda_input(ui_1mlist) ""
set rda_input(ui_1lmfile) ""
set rda_input(ui_settop) "{}
set rda_input(ui_topcell) "none"
set rda_input(ui_celllib) ""
set rda_input(ui_iolib) ""
set rda_input(ui_areaiolib) ""
set rda_input(ui_blklib) ""
set rda_input(ui_kboxlib) ""
set rda_input(ui_gds_file) ""
set rda_input(ui_oa_oa2lefversion) ""
set rda_input(ui_view_definition_file) ""
set rda_input(ui_timelib,max) "TIM/fsd0a_a_generic_core_ss0p9v125c.lib ../IP/RF16X32/SYAA90_16X32X1CM2_WC.lib"
set rda_input(ui_timelib,min) "TIM/fsd0a_a_generic_core_ff1p1vm40c.lib ../IP/RF16X32/SYAA90_16X32X1CM2_BC.lib"
set rda_input(ui_smodDef) ""
set rda_input(ui_smodData) ""
set rda_input(ui_locvlib) ""
set rda_input(ui_dpath) ""
set rda_input(ui_tech_file) ""
set rda_input(ui_io_file) "CONF/alu32top.io"
set rda_input(ui_timingcon_file) ""
set rda_input(ui_latency_file) ""
set rda_input(ui_scheduling_file) ""
set rda_input(ui_buf footprint) ""
set rda_input(ui_delay footprint) ""
set rda_input(ui_inv footprint) ""
set rda_input(ui_leffile) "TEC/header9m126_V55.1ef TEC/fsd0a_a_generic_core.1ef ../IP/RF16X32/SYAA90_16X32X1CM2.1ef"
set rda_input(ui_cts_cell footprint) {BUFCKX1 INVCKX1 }
set rda_input(ui_cts_cell_list) ""
set rda_input(ui_core cnt) {aspect}
set rda_input(ui_aspect ratio) {1.0}
set rda_input(ui_core util) {0.800}
set rda_input(ui_core height) ""
set rda_input(ui_core width) ""
set rda_input(ui_core to left) {50}
set rda_input(ui_core to right) {50}
```
C.2 alu32top.io

This file is referenced in the configuration file created during design import (Chapter 4, Section ). It defines the locations of the IO pins around the periphery of the core.

# Syntax:
# Pin: <pin-name> <location> [ <metal layer> ]
# where
# <location> may be either one of:
# n north (top)
# e east (right)
# s south (bottom)
# w west (left)
# <metal layer> may be 1 to 9 (default 3)
# The default I/O order for a vertical edge is from the bottom to the top,
# and for a horizontal edge, it is from the left to the right.
#
Version: 2

# North (outputs) # East (outputs)
Pin: res[0] n Pin: rf_do[0] e

# West (inputs) # South (control)
# Pin: op1[31] w Pin: clk s
Pin: op1[30] w Pin: rst_b s
... Pin: op1[2] w Pin: start s
Pin: op1[0] w Pin: cmd[0] s
Pin: op2[0] w

C.3 alu32top.mplan

This file is used during automatic floorplanning for placing the macro block (Chapter 4, Section 4.3.2).

###########################################################
# Syntax Convention: #
# [] means optional #
# <> means filling with real value or name in your design #
# () indicates the unit name for your value #
# | means OR #
# ... means more similar items #
###########################################################

BEGIN SEED
I_RF
END SEED

###########################################################
# Hard Macro Section (optional) #
# cellOrientation <cellName> [R0] [MX] [MY] [R180] [MX90] [R90] [R270] [MY90] #
# instOrientation <instName> [R0] [MX] [MY] [R180] [MX90] [R90] [R270] [MY90] #
# cellSpacing <cellName> <xMin(um)> <yMin(um)> #
# instSpacing <instName> <xMin(um)> <yMin(um)> #
# fenceSpacing <fenceName> <xMin(um)> <yMin(um)> #
###########################################################
BEGIN MACRO
C.4 Place+Route Script

# File: alu32top_par.tcl
# Description: Simple place+route script for ALU32 top-level model.
# Notes: The script can be executed in Encounter console by running
# source path-to-tcl-script
# The script can also be run from a Linux shell as follows:
# encounter -init path-to-tcl-script -win
# Author: Alain Vachoux, EPFL STI IEL LSM, alain.vachoux@epfl.ch
# for EDATP labs.
# Tools: Encounter Digital Implementation 9.14

# Utility procedure
# proc put_banner TITLE {
#    puts ""
#    puts "---"n--------------------"n
#    puts "$TITLE"n
#    puts "--------------------"n
#}

# Design name
# set DESIGN_NAME alu32top_clk10ns

# Load configuration
# put_banner "LOAD CONFIGURATION"
loadConfig CONF/${DESIGN_NAME}.conf
setDrawView fplan
fit
saveDesign DB/${DESIGN_NAME}-import.enc

# Specify the floorplan
# put_banner "SPECIFY FLOORPLAN"

set ASPECT_RATIO 1.0 ;# rectangle with height = 1.0*width
set ROW_DENSITY 0.80 ;# 0.1..1.0
set CORE_TO_LEFT 6.0 ;# micron
set CORE_TO_BOTTOM 6.0 ;# micron
set CORE_TO_RIGHT 6.0 ;# micron
set CORE_TO_TOP 6.0 ;# micron

floorPlan \
    -r $ASPECT_RATIO \n    $ROW_DENSITY $CORE_TO_LEFT $CORE_TO_BOTTOM $CORE_TO_RIGHT $CORE_TO_TOP
# Process IP macro block
#
put_banner "PROCESS IP MACRO"

setPlanDesignMode -fixPlacedMacros true
planDesign -constraints CONF/alu32top.mplan
finishFloorplan -autoHalo -autoBlockage
saveDesign DB/${DESIGN_NAME}-fplan.enc

# Connect global nets
#
put_banner "CONNECT GLOBAL NETS"

set_PWR_NET VCC ;# power net name as in LEF file
set_GND_NET GND ;# ground net name as in LEF file

clearGlobalNets
globalNetConnect $PWR_NET -type pgpin -pin $PWR_NET -inst * -module {} -verbose
globalNetConnect $GND_NET -type pgpin -pin $GND_NET -inst * -module {} -verbose

# Add power rings
#
put_banner "ADD POWER RINGS"

set_CENTER_RING 1 ;# center rings between I/O and core
set_WIDTH_BOT 2.0 ;# width of bottom ring segments
set_WIDTH_RIGHT 2.0 ;# width of right ring segments
set_WIDTH_TOP 2.0 ;# width of top ring segments
set_WIDTH_LEFT 2.0 ;# width of left ring segments
set_SPACING_BOT 0.6 ;# spacing of bottom ring segments
set_SPACING_RIGHT 0.6 ;# spacing of right ring segments
set_SPACING_TOP 0.6 ;# spacing of top ring segments
set_SPACING_LEFT 0.6 ;# spacing of left ring segments
set_LAYER_BOT ME1 ;# LEF metal layer of bottom ring segments
set_LAYER_RIGHT ME2 ;# LEF metal layer of right ring segments
set_LAYER_TOP ME1 ;# LEF metal layer of top ring segments
set_LAYER_LEFT ME2 ;# LEF metal layer of left ring segments
set_EXT_BL 1 ;# 0 1 - extend bottom segment to left
set_EXT_BR 1 ;# 0 1 - extend bottom segment to right
set_EXT_RB 1 ;# 0 1 - extend right segment to bottom
set_EXT_RT 0 ;# 0 1 - extend right segment to top
set_EXT_TR 0 ;# 0 1 - extend top segment to right
set_EXT_TL 0 ;# 0 1 - extend top segment to left
set_EXT_LT 0 ;# 0 1 - extend left segment to top
set_EXT_LB 1 ;# 0 1 - extend left segment to bottom

addRing
  -around core
  -nets "$GND_NET $PWR_NET"
  -center $CENTER_RING
  -width_bottom $WIDTH_BOT -width_right $WIDTH_RIGHT
  -width_top $WIDTH_TOP -width_left $WIDTH_LEFT
  -spacing_bottom $SPACING_BOT -spacing_right $SPACING_RIGHT
  -spacing_top $SPACING_TOP -spacing_left $SPACING_LEFT
  -layer_bottom $LAYER_BOT -layer_right $LAYER_RIGHT
  -layer_top $LAYER_TOP -layer_left $LAYER_LEFT
  -bl $EXT_BL -br $EXT_BR -rb $EXT_RB -rt $EXT_RT
  -tr $EXT_TR -tl $EXT_TL -lt $EXT_LT -lb $EXT_LB

# Add power stripes
#
put_banner "ADD POWER STRIPES"

set_STRP_LAYER ME2 ;# LEF metal layer of stripe segments
set_STRP_WIDTH 2.0 ;# width of stripe segments
set_STRP_SPACING 0.6 ;# spacing of stripe segments
set_STRP_NUM_SETS 1 ;# number of stripe sets
set_STRP_XLEFT_OFFS 100.0 ;# start point of stripes from core boundary
addStripe
-net " $GND_NET $PWR_NET "
-layer $STRP_LAYER 
-width $STRP_WIDTH -spacing $STRP_SPACING 
-number_of_sets $STRP_NUM_SETS 
-xleft_offset $STRP_XLEFT_OFFS

# Route power nets
# put_banner "ROUTE POWER NETS"
sroute
-connector { blockPin corePin floatingStripe } 
-blockPin { onBoundary bottomBoundary rightBoundary } 
-AllowJogging $ALLOW_JOG

saveDesign DB/${DESIGN_NAME}-power.enc

# Place design
# put_banner "PLACE STD CELLS"

set PROCESS 90 ;# process technology value [micron]

setDesignMode -process $PROCESS
setPlaceMode -timingDriven true
placeDesign -prePlaceOpt -inPlaceOpt
setDrawView place
checkPlace

put_banner "PRE-CTS OPTIMIZATION"

optDesign -preCTS -outDir RPT/${DESIGN_NAME}

saveDesign DB/${DESIGN_NAME}-placed.enc

# Clock tree synthesis
# put_banner "CREATE CLOCK TREE"

set CLOCK_NAME clk ;# clock net name

createClockTreeSpec -output CTS/${DESIGN_NAME}_spec.cts 
-bufferList BUFX1 BUFX12 BUFX16 BUFX1P BUFX2 BUFX20 BUFX3 
BUFX4 BUFX6 BUFX8 INVX1 INVX12 INVX16 INVX1P 
INVX2 INVX20 INVX3 INVX4 INVX6 INVX8 INVXXP

clockDesign -specFile CTS/${DESIGN_NAME}_spec.cts -outDir RPT/${DESIGN_NAME}
ckECO -postCTS -useSpecFileCellsOnly -report RPT/${DESIGN_NAME}/cts.rpt

put_banner "POST-CTS OPTIMIZATION"

optDesign -postCTS -outDir RPT/${DESIGN_NAME}

saveDesign DB/${DESIGN_NAME}-cts.enc

# Route design
# put_banner "ROUTE DESIGN"

set ROUTE_TIMING true ;# true | false - timing driven routing
set ROUTE_TDR_EFFORT 5 ;# 0..10 - 0: opt. congestion; 1: opt. timing

setNanoRouteMode 
-routeWithTimingDriven $ROUTE_TIMING 
-routeTdREffort $ROUTE_TDR_EFFORT
routeDesign
put_banner "POST-ROUTE OPTIMIZATION"

optDesign -postRoute -outDir RPT/${DESIGN_NAME}

saveDesign DB/${DESIGN_NAME}-routed.enc

# Add filler cells
# put_banner "ADD FILLER CELLS"

set FILLER_CELLS \
   "FILLERD8 FILLERD64 FILLERD4 FILLERD32 FILLERD2 FILLERD16 FILLERD1"

set PREFIX FILLER

addFiller \
   -cell "$FILLER_CELLS" \
   -prefix "$PREFIX"

setDrawView place

saveDesign DB/${DESIGN_NAME}-filled.enc

# Verify design
# put_banner "VERIFY DESIGN"

verifyConnectivity -type all -report RPT/${DESIGN_NAME}/connectivity.rpt
verifyGeometry -report RPT/${DESIGN_NAME}/geometry.rpt

# Generate reports
# put_banner "GENERATE REPORTS"

reportNetStat
reportGateCount -outfile RPT/${DESIGN_NAME}/gateCount.rpt
summaryReport -outdir RPT/${DESIGN_NAME}/summary

# Generate SDF timing file
# put_banner "GENERATE SDF TIMING FILE"

set SDF_VERSION 2.1 ;# 2.1 | 3.0
set SDF_PREC 4 ;# number of digits for delay values

write_sdf -version $SDF_VERSION -precision $SDF_PREC \ 
   TIM/${DESIGN_NAME}_pared_v${SDF_VERSION}_prec${SDF_PREC}.sdf

# Generate Verilog netlist
# put_banner "GENERATE VERILOG NETLIST"

saveNetlist -excludeLeafCell HDL/GATE/${DESIGN_NAME}_pared.v

# Generate GDS2 file
# put_banner "GENERATE GDS2 FILE"

streamOut DEX/${DESIGN_NAME}_pared.gds \
   -mapFile DEX/gds2out.map \
   -libName ${DESIGN_NAME} \
   -merge ../IP/RF16X32/SYAA90_16X32X1CM2.gds