Top-down digital design flow

EDA tools:
Modelsim, Synopsys Design Compiler, Cadence Encounter

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

version 3.0.2 / 6 Dec 2005
## 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. 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>
</tbody>
</table>
# Table of Contents

## Chapter 1: Introduction
1. Top-down design flow .............................................. 1
2. EDA tools and design kit configuration .................... 3
3. Design project organisation ................................. 3
4. Installation of the AMS design kit ......................... 4
5. VHDL example: Adder-subtractor ............................ 6
6. Text editing .................................................. 7
7. Design flow steps .............................................. 8

## Chapter 2: VHDL and Verilog simulation
1. Starting the Modelsim graphical environment .......... 9
2. Simulation of (pre-synthesis) RTL VHDL models .... 10
3. Simulation of the post-synthesis VHDL model with timing data 12
4. Simulation of the post-PaR Verilog model with timing data 14

## Chapter 3: Logic synthesis
1. Starting the Design Vision graphical environment ...... 17
2. RTL VHDL model analysis ...................................... 18
3. Design elaboration ........................................... 18
4. Design environment definition ............................. 19
5. Design constraint definitions ............................... 20
6. Design mapping and optimization ......................... 21
7. Report generation ............................................ 23
8. VHDL/Verilog gate-level netlist generation and post-synthesis timing data (SDF) extraction ............... 26
9. Design constraints generation for placement and routing .... 28
10. Design optimization with tighter constraints ........... 28
11. Using scripts ................................................ 28

## Chapter 4: Standard cell placement and routing
1. Starting the Encounter graphical environment .......... 33
2. Design import ................................................ 34
3. Global net connections ...................................... 36
4. Operating conditions definition ......................... 37
5. Floorplan Specification ..................................... 37
6. CAP cell placement ........................................ 38
7. Power ring/strip creation and routing .......... 39
8. Core cell placement ..................................... 41
9. Timing analysis ............................................ 42
10. Clock tree synthesis (optional) ......................... 43
11. Design routing .......................................... 44
12. Timing analysis .......................................... 45
13. Design checks ............................................ 45
14. Report generation ......................................... 46
15. Post-route timing data extraction .................. 47
16. Post-route netlist generation ......................... 47
17. GDS2 file generation .................................. 47
18. Design import in Virtuoso ................................ 48
19. Using scripts ............................................. 49
Chapter 1: Introduction

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

The following tools are considered in this document:
- Modelsim v6.1b or higher, from Mentor Graphics.
- Design Compiler and Design Vision 2005.09 or higher from Synopsys.
- Encounter 4.1, and IC 5.0.33 or higher from Cadence Design Systems.

The design kit used is the Hit-Kit 3.70 from AMS. The process is the 0.35 micron 4-metal CMOS called C35B4.

Next chapters in this document are each addressing a specific set of tasks. Chapter 2 is about VHDL and Verilog simulation, chapter 3 is about logic synthesis and chapter 4 is about place and route. Steps in these chapters are not necessarily to be done in the given sequence. Go to “1.7 Design flow steps” to get a typical sequence of steps.

1.1 Top-down design flow

Figure 1.1 illustrates the flow whose main steps are the following:

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

- **RTL simulation**
  The VHDL RTL models are validated through simulation by means of a number of testbenches also written in VHDL. The tool used at this step is a logic simulator (in our case, Modelsim).

- **RTL synthesis**
  The synthesis process infers a possible gate-level realisation 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 targetted logic gates belong to a library that is provided by a foundry or an IP company as part of a so-called design kit. Typical gate libraries include a few hundreds of combinational and sequential logic gates. Each logic function is implemented in several gates to accomodate 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 gate-level VHDL netlist, a Verilog gate-level netlist, and a SDF description. The first netlist is typically used for post-synthesis simulation, while the second netlist is better suited as input to the place&route step. The SDF description includes delay information for simulation. Note that considered delays are at this step correct for the gates but only estimated for the interconnections. The tool used at this step is Design Compiler.

• **Post-synthesis gate-level simulation**

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

• **Standard cell place and route**

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

  The interconnections between gates are today usually done over the cells since current processes allow several metal layers (i.e. 4 metal layers for the AMS C35 process). As a consequence, the rows may be abutted and flipped so power and ground rails are shared between successive rows.

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

- **Post-layout gate-level simulation**
  The Verilog gate-level netlist can be simulated by using the existing VHDL testbenches and the more accurate SDF data extracted from the layout. The tool used at this step is Modelsim.

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

### 1.2 EDA tools and design kit configuration

The ~/.cshrc file must include the following lines in the given order:

```
source /softs/site/pre_cshrc
-------- start user specific section here
# set EDA tools
set amshk = v3.70   # AMS design kit
set cadence_soce = 4.1  # Cadence Encounter
set modelsim = v6.1b # Modelsim VHD/Verilog simulator
set synopsys_dc = 2005.09 # Synopsys Design Compiler tool suite
-------- end user specific section here
source /softs/site/post_cshrc
```

These settings are valid on Sun Solaris machines in the DIA003 room, on their related servers (desun1, immmsun3, immmsun4) and on LSM Sun Solaris and Linux machines. You’ll need then to start a new shell or source the .cshrc file to put the settings in effect.

### 1.3 Design project organisation

Given the number of EDA tools and files used in the flow, it is strongly recommended to organise the working environment in a proper way. To that end, the create_project script can be used to create a directory structure in which design files are stored (provided the settings in 1.2 are in place). The use of the script is as follows:

```
create_project [ -v | -h ]
create_project [ -f | -n ] [ -b <tool> ] <name>
```

where:

- `-b <tool>` specify the backend tool:
  - `se` Silicon Ensemble (default)
  - `enc` Encounter
- `-f` force the creation of missing subdirectories
- `-h` (or no arguments) display this help
- `-n` display what the command would do
- `-v` display script version
- `<name>` project directory name

For example, to create the project directory called ADDSUB that will be used to do the steps presented in the rest of this document:
The ADDSUB top-level directory hosts the configuration files for logic simulation (Modelsim), logic synthesis (Synopsys DC) and standard cell place and route (Cadence Encounter). As a consequence, it is required that the tools are always started from that point. One exception is full-custom layout tools (Cadence IC) that must be started from the subdirectory LAY, which hosts different configuration files.

Figure 1.2 gives the proposed directory structure and the role of each subdirectory (directories have a “/” at the end of their names). Text starting with “#” is a comment. The actual use of the subdirectories and files will be explained while going through the tutorial example later in this document.

### Figure 1.2: Design project structure.

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

### 1.4 Installation of the AMS design kit

To install the files required by the AMS design kit, first go down into the top-level project directory:

```
vachoux@lsmsun1-educ> cd ADDSUB
```

and execute the `ams_setup` script as follows:

```
ams_setup [ -h ]
ams_setup -i
ams_setup [ -f ] [ -s ] -p <process> -t <toolset>
```
where:

- no arguments display this help
- `-f` force replacement of existing files
- `-i` display available processes and tools
- `-p` select process
- `-s` silent mode
- `-t` select toolset

To install the 0.35 micron CMOS called C35B4 for logic simulation and synthesis, use:

```
[28] vachoux@lsmsun1-ADDSUB> ams_setup -p c35b4 -t synopsys_dc
```

The command produces the following output¹:

```
IMPORTANT NOTICE

All AMS documentation and design files made available with this command are subject to non-disclosure agreements between Europractice, AMS and EPFL, and hence must be considered as strictly confidential.
For more information: Alain Vachoux, STI-LSM, alain.vachoux@epfl.ch.

-- Configuration parameters:
AMS HIT-Kit version v3.70
Process: c35b4
Tool : synopsys_dc
-- Configuration log:
File "./.synopsys_dc.setup" created
File "./.modelsim.ini" created
Link "LIB/C35_CORELIB_VLOG" to "/softs/dkits/ams/site/lib/modelsim/v6.1b/c35_corelib_vlog" created
Link "LIB/C35_IOLIB_3M_VLOG" to "/softs/dkits/ams/site/lib/modelsim/v6.1b/c35_iolib_3m_vlog" created
Link "LIB/C35_IOLIB_4M_VLOG" to "/softs/dkits/ams/site/lib/modelsim/v6.1b/c35_iolib_4m_vlog" created
Directory "./.LIB/SNPS" created
Directory "./.LIB/MSIM" created
Modelsim library WORK mapped to ./LIB/MSIM
Synopsys library WORK mapped to ./LIB/SNPS
```

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

```
[29] vachoux@lsmsun1-ADDSUB> ams_setup -p c35b4 -t cadence_enc
```

The command produces the following output:

```
IMPORTANT NOTICE

All AMS documentation and design files made available with this command are subject to non-disclosure agreements between Europractice, AMS and EPFL, and hence must be considered as strictly confidential.
For more information: Alain Vachoux, STI-LSM, alain.vachoux@epfl.ch.

-- Configuration parameters:
AMS HIT-Kit version v3.70
Process: c35b4
Tool : cadence_enc
-- Configuration log:
Run ams_encounter in a temporary directory...
Move files from temporary directory to proper location...
../PAR/BIN/gemma.tcl
../PAR/BIN/fillcore.tcl
../PAR/BIN/fillperi.tcl
../PAR/CONF/c35b4_std.conf
```

1. The output is slightly different if the project directory structure has not been created first.
For information on the AMS design kits, execute the amshk_doc command. It will start Netscape and display the home page of the local AMS web site. Note that some information such as design rules document is password protected.

1.5 VHDL example: Adder-subtractor

Code 1.1 gives the VHDL model of a generic N bit adder-subtractor. The model is deliberately quite abstract to show how the synthesizer will be able to infer different adder architectures (ripple carry or carry look-ahead architectures) depending on the given constraints.

![Code 1.1](http://example.com) RTL synthesisable model of a N bit adder-subtractor.

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

entity addsub is
  generic (NBITS: natural := 4);
  port (a, b: in  std_logic;
         add: in  std_logic;
         clk, rst, z: out unsigned(NBITS-1 downto 0));
end entity addsub;

architecture dfl of addsub is
  signal a_reg, b_reg, z_reg: unsigned(NBITS-1 downto 0);
begin
  process (rst, clk)
  begin
    if rst = '1' then
      a_reg <= (others => '0');
      b_reg <= (others => '0');
      z <= (others => '0');
    elsif clk'event and clk = '1' then
      a_reg <= a;
      b_reg <= b;
      z <= z_reg;
    end if;
  end process;
  z_reg <= a_reg + b_reg when add = '1' else
            a_reg - b_reg;
end dfl;
```

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

```
[30]vachoux@lsmsun1-ADDSUB> install_topdown addsub_enc
```

The following files are copied:

```
HDL/RTL/addsub_dfl.vhd
HDL/TBENCH/tb_addsub_mapped.vhd
HDL/TBENCH/tb_addsub_par.vhd
HDL/TBENCH/tb_addsub_rtl.vhd
SYN/BIN/addsub_syn.tcl
PAR/BIN/addsub_par.tcl
```
Code 1.2 gives the testbench for the RTL model with N=8 (file tb_addsub_rtl.vhd). It is by far not complete as it only checks one addition and one subtraction. The testbenches for the mapped design (file tb_addsub_mapped.vhd) and for the simulation of the placed and routed Verilog netlist (file tb_addsub_par.vhd) only differs by the instantiation of the component under test.

Code 1.2. VHDL testbench for the RTL model with N=8.

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

entity tb_addsub_rtl is end;

architecture bench of tb_addsub_rtl is

constant CLK_PER: time    := 20 ns;
constant NBITS  : natural := 8;

signal clk     : std_logic := '0';
signal rst, add: std_logic;
signal a, b, z : unsigned(NBITS-1 downto 0)

begin

UUT: entity work.addsub(dfl)
    generic map (NBITS)
    port map (clk, rst, add, a, b, z);

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

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

end architecture bench;
```

The file addsub.tcl in directory SYN/BIN is a Tcl script that performs synthesis of the VHDL model in batch mode.

1.6 Text editing

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

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

Step 1) VHDL modeling (tool: text editor)

Step 2) Pre-synthesis VHDL simulation (tool: Modelsim) [2.2]
   2.1) Compilation of the RTL VHDL model and related testbench
   2.2) Simulation of the RTL VHDL model

Step 3) Logic synthesis (tool: Synopsys Design Compiler)
   3.1) RTL VHDL model analysis [3.2]
   3.2) Design elaboration (generic synthesis) [3.3]
   3.3) Design environment definition (operating conditions, wire load model) [3.4]
   3.4) Design constraint definitions (area, clock, timings) [3.5]
   3.5) Design mapping and optimization (mapping to gates) [3.6]
   3.6) Report generation [3.7]
   3.7) VHDL gate-level netlist generation [3.8]
   3.8) Post-synthesis timing data (SDF) generation for the VHDL netlist [3.8]
   3.9) Verilog gate-level netlist generation [3.8]
   3.10) Design constraints generation for placement and routing [3.9]

Step 4) Post-synthesis VHDL simulation (tool: Modelsim) [2.3]
   4.1) Compilation of the VHDL/Verilog netlist and related testbench
   4.2) Simulation of the post-synthesis gate-level netlist with timing data

Step 5) Placement and routing (tool: Cadence Encounter)
   5.1) Design import (technological data + Verilog netlist) [4.2]
   5.2) Global net connections definition [4.3]
   5.3) Operating conditions definition [4.4]
   5.4) Floorplan specification [4.5]
   5.5) CAP cell placement [4.6]
   5.6) Power ring/stripe creation and routing [4.7]
   5.7) Core cell placement [4.8]
   5.8) First timing analysis [4.9]
   5.9) Clock tree synthesis (optional) [4.10]
   5.10) Design routing [4.11]
   5.11) Design checks [4.13]
   5.12) Report generation [4.14]
   5.13) Post-PaR timing data (SDF) extraction [4.15]
   5.14) Post-PAR Verilog netlist generation [4.16]
   5.15) GDS2 file generation [4.17]

Step 6) Post-layout VHDL/Verilog simulation (tool: Modelsim) [2.4]
   6.1) Compilation of the Verilog netlist and related testbench
   6.2) Simulation of the post-synthesis or post-PaR gate-level netlist with PaR timing data
Chapter 2: VHDL and Verilog simulation

This chapter presents the main steps to perform the logic simulation of VHDL and Verilog models with the Modelsim tool. Execute the Unix command “alias | grep "ms_“ to get all aliases related to Modelsim documentation (aliases that have the ms_ prefix).

2.1 Starting the Modelsim graphical environment

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

```
[vachoux@lsmsun1-ADDSUB> vsim &
```

The modelsim.ini file actually defines the mapping between logical design libraries and their physical locations. Note that the Help menu on the top right allows one to access the complete documentation of the tool.
2.2 Simulation of (pre-synthesis) RTL VHDL models

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

```
ModelSim> vcom HDL/RTL/addsub_dfl.vhd
  # Model Technology ModelSim SE vcom 6.1b Compiler 2005.09 Sep 8 2005
  # -- Loading package standard
  # -- Loading package std_logic_1164
  # -- Loading package numeric_std
  # -- Compiling entity addsub
  # -- Compiling architecture dfl of addsub

ModelSim> vcom HDL/TBENCH/tb_addsub_rtl.vhd
  # Model Technology ModelSim SE vcom 6.1b Compiler 2005.09 Sep 8 2005
  # -- Loading package standard
  # -- Loading package std_logic_1164
  # -- Loading package numeric_std
  # -- Compiling entity tb_addsub_rtl
  # -- Compiling architecture bench of tb_addsub_rtl
  # -- Loading entity addsub
```

The other way is to left-click on the Compile icon, to select the files to compile in the ./HDL/RTL and ./HDL/TBENCH directories, click on Compile and finally close the window (click Done).

The compiled modules are stored in the logical library WORK which is mapped to the physical location ./LIB/MSIM.

Once VHDL (or Verilog) models have been successfully compiled in the design library, it is possible to create a make file that can be used to recompile only the required files. The `vmake` command can only be run from a Unix shell and creates the make file:

```
[vachoux@lsmsun1-ADDSUB> vmake > Makefile
```

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

To simulate the RTL model, select the main menu item Simulate -> Start simulation... to get the simulation dialog window.

Select the architecture of the testbench and a resolution of ns. Then click OK.

The main window now changes a bit to show the simulation hierarchy, the list of signals in the testbench and the simulation console (with now
the VSIM #> prompt). Left clicking on an instance in the simulation hierarchy pane displays the corresponding VHDL source in the right pane.

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

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

The selected signals are displayed in a new window called wave. The wave pane is by default located on the top right (as a new tab on the source windows). You can click on the Undock icon to make the wave pane separate.

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

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

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

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

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

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

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

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

```
ModelSim> vcom HDL/GATE/addsub_dfl_nbits8_mapped.vhdl
...
ModelSim> vcom HDL/TBENCH/tb_addsub_mapped.vhd
...
```

To simulate the RTL model, select the main menu item **Simulate -> Start simulation**... to get the simulation dialog window. Select the architecture of the testbench and a resolution of 100ps. Then click the **SDF** tab.

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

Note that the box **Reduce SDF errors to warnings** must be checked. This is required to avoid the simulation to stop prematurely due to errors such as “Failed to find port ‘a(7)’”. These are not really errors here as they are related to interconnect delay data in the SDF file that are not used in the simulation (they are actually all set to zero).
Then click **OK** in the remaining **Start Simulation** dialog box to load the mapped netlist. Clock to output delays of the order of 500ps to 600ps should be visible in the wave window.
2.4 Simulation of the post-PaR Verilog model with timing data

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

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

```
[120]vachoux@lsmsun1-ADDSUB> vdir -r
Library Vendor : Model Technology
Maximum Unnamed Designs : 3
ENTITY addsub
ARCH df1
ENTITY addsub_nbits8
  ARCH syn_df1
ENTITY addsub_nbits8_dw01_addsub_0
  ARCH syn_rpl
ENTITY tb_addsub_mapped
  ARCH bench
ENTITY tb_addsub_par
  ARCH bench
ENTITY tb_addsub_rtl
  ARCH bench
```

The `ENTITY` and `ARCHITECTURE` keywords denote VHDL design units. The problem is that the modules in the Verilog netlist have the same names as in the VHDL post-synthesis models (actually they only differ in the case of some name parts). To allow the VHDL testbench `HDL/tb_addsub_par.vhd` to instantiate a Verilog model, the direct instantiation statement cannot include the architecture name. Therefore, a default mapping is considered and the design library must only contain the required compiled units. As a consequence, some existing units must be deleted from the library with the `vdel` command:

```
[121]vachoux@lsmsun1-ADDSUB> vdel addsub_nbits8_dw01_addsub_0
[122]vachoux@lsmsun1-ADDSUB> vdel addsub_nbits8
```

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

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

```
[123]vachoux@lsmsun1-ADDSUB> vlog HDL/GATE/addsub_nbits8-routed.v
[124]vachoux@lsmsun1-ADDSUB> vcom HDL/TBENCH/tb_addsub_par.vhd
```

To simulate the placed and routed netlist with timing data, select the item `Simulate -> Start simulation...` in the main menu to get the simulation dialog window. Select the architecture of the testbench and a resolution of 100ps. Then click the `Libraries` tab to add the Verilog library of the gates ./LIB/C35_CORELIB_VLOG, and load the SDF timing file `PAR/TIM/addsub_nbits8-routed.sdf`.

Note that the box `Reduce SDF errors to warnings` must be checked. This is required to avoid the simulation to stop prematurely due to errors such as "Failed to find matching specify timing constraint". These are not really errors here as they are related to removal (asynchronous) timing constraints generated by Encounter that are not supported in the Verilog models of the gates.
Then click **OK** in the remaining **Start Simulation** dialog box to load the mapped netlist. Clock to output delays of the order of 800ps to 1000ps should be visible in the wave window.

![Waveform Diagram](image)

More accurate values for the delays can be obtained using a smaller time unit in simulation (e.g., 10ps or less).
Chapter 3: Logic synthesis

This chapter presents the main steps to perform the logic synthesis of the VHDL RTL model with the Synopsys Design Vision and Design Compiler tools. The `sold` alias displays the complete Synopsys documentation set.

3.1 Starting the Design Vision graphical environment

To start the Synopsys Design Vision environment, enter the `design_vision` command in the Unix shell:

```
[65]vachoux@lsmsun1-ADDSUB> design_vision -db_mode
```

The `-db_mode` switch is required for version 2005.09 and later to use the old database format. The command line is also echoed in the terminal shell from which the tool has been started, so it is possible to enter DC commands from there as well (`design_vision-t> prompt`). In addition it is still possible to execute Unix commands from here.
3.2 RTL VHDL model analysis

The analysis phase compiles the VHDL model and checks that the VHDL code is synthesizable.

Select `File -> Analyze...` in the main menu.

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

In the case you have more than one VHDL file to analyze, be careful to list the files in the correct analysis order. Another way is to select all the files and click on the button `Automatic Ordering` to let the tool find the right dependency order. This button is only active when several VHDL files are listed.

Click `OK`. The console now includes a warning message saying that only generic parameters of type integer (and not subtype, such as natural or positive) are supported in synthesis:

```
Warning: Type of the generic is assumed to be 'Integer' in synthesis on line 6 (VHDL-2023)
```

Each warning or error message is identified by a code, such as VHDL-2023. To get more details on the message, you can execute the command “man VHDL-2023” in the command line.

3.3 Design elaboration

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

Select `File -> Elaborate...` in the main menu.

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

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

```
Inferred memory devices in process
in routine addsub_NBITS8 line 17 in file
'/home/edpeo/tesr/ADSBH/ADH/RTL/addsub_4FL.vhd'
```

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

### 3.4 Design environment definition

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

To define the operating conditions, select the main menu item **Attributes -> Operating Environment -> Operating Conditions...**

Select the WORST-IND condition, which defines a temperature of 85°C, a voltage of 3V (the C35 process is a 3.3V process), and a slow process. Each cell library defines its own set of operating conditions and may use different names for each set.

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

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

```
report_lib c35_CORELIB
```

The information on the operating conditions and wire load models are given at the beginning of the display.

### 3.5 Design constraint definitions

Many kinds of constraints may be defined on the design. Here only constraints on the area and the clock will be defined. To define the clock attributes, i.e. its period and duty cycle, select the entity `addsub_NBIT8` in the hierarchy window and then click the **Create Symbol View** icon.

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

Define a clock period of 10 ns with 50% duty cycle. Time unit is not specified here. It is defined in the cell library and is usually ns.

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

```
create_clock -name "clk" -period 10 -waveform { 0 5 } { clk }
```
To define an area constraint, select in the main menu the item Attributes -> Optimization Constraints -> Design Constraints....

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

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

```
set_max_area 0
```

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

Select the entity addsub_NBITS8 in the hierarchy window and then the main menu item File -> Save As....

Save the elaborated design under the name addsub_dfl_nbits8_elab.db in the ./SYN/DB directory.

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

The console includes the equivalent command line:

```
write -hierarchy -format db
-output .../ADDSUB/SYN/DB/addsub_dfl_nbits8_elab.db
```

To read back an elaborated design, select the main menu item File -> Read... and then select the file to read.

### 3.6 Design mapping and optimization

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

Select the main menu item Design -> Compile Design....

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

Click OK. The console and the Unix shell now include the progress of the work.
The equivalent command line is:

```bash
compile -map_effort medium -area_effort medium
```

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

The inferred 8 bit adder has a ripple-carry architecture and uses the available ADD32 standard cell that implements a 1 bit full adder.

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

Note that the default resource allocation and implementation for operative parts is based on timing constraints. That is why two separate adder and subtractor have been inferred. To force the synthesizer to share operators into a single adder-subtractor component, and then save area further, read back the elab-
orated design, reapply the design environment and design constraints definitions and set the following variables before compile:

```
design_vision-t> set_resource_allocation area_only
design_vision-t> set_resource_implementation area_only
```

The mapped design can now be saved. Select the entity `addsub_NBITS8` in the hierarchy window and then the main menu item `File -> Save As...`. Save the mapped design under the name `addsub_dfl_nbits8_mapped.db` in the directory `/SYN/DB`.

### 3.7 Report generation

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

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

Save the report in the file `/SYN/RPT/addsub_dfl_nbits8_mapped_area.rpt` as well as in the report viewer.

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

```
****************************************
Report : area
Design : addsub_NBITS8
Version: X-2005.09
Date   : Thu Nov 24 18:39:25 2005
****************************************

Library(s) Used:
c35_CORELIB (File: /softs/dkits/ams/v3.70/synopsys/c35_3.3V/c35_CORELIB.db)

Number of ports: 27
Number of nets: 78
Number of cells: 45
Number of references: 6

Combinational area: 5514.599121
Noncombinational area: 7425.598633
Net Interconnect area: 1638.000000

Total cell area: 12940.200195
Total area: 14578.197266
```

The area unit depends on the standard cell library. Here all area figures are in square microns. The net interconnect area is estimated with the use of a wire load model that has been automatically selected from the design area.
To get a report on the most critical timing path in the mapped design, select the main menu item **Timing -> Report Timing...**

Save the report in the file `.SYN/RPT/addsub_dff_nbits8_mapped_timing.rpt` as well as in the report viewer.

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

```
To get a report on the most critical timing path in the mapped design, select the main menu item **Timing -> Report Timing...**

Save the report in the file `.SYN/RPT/addsub_dff_nbits8_mapped_timing.rpt` as well as in the report viewer.

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

```
```
All times are expressed in ns (the time unit is actually defined in the cell library). The slack defines the time margin from the clock period. A positive slack means that the latest arriving signal in the path still arrives before the end of the clock period. A negative slack means that the timing constraint imposed by the clock is violated. The timing delays that are accounted for are the internal gate delays (from the cell library) and the estimated interconnect delays (from the cell library and the wire load model in use).

To highlight the critical path on the schematic, select the addsub_NBITS8 entity in the hierarchy window and then the main menu item View -> Highlight -> Critical Path (CTRL-H). You can see that the path goes through the sub31_minus component so you can also select the component and do the same to visualize the critical path inside the component.

Another useful report is the list of resources used. A resource is an arithmetic or comparison operator read in as part of an HDL design. Resources can be shared during execution of the compile command.
To get a report on the resources used, select the main menu item **Design -> Report Design Resources...**

Save the report in the file `.SYN/RPT/addsub_dfl_nbits8_mapped_resources.rpt` as well as in the report viewer.

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

```
**********************************
Report : resources
Design : addsub_NBITS8
Version: X-2005.09
Date   : Fri Nov 25 09:57:30 2005
**********************************
```

```
Resource Sharing Report for design addsub_NBITS8 in file
/home/edatp0/test/ADDSUB/HDL/RTL/addsub_dfl.vhd
```

```
<p>|          |              |            | Contained     |                      |</p>
<table>
<thead>
<tr>
<th>Resource</th>
<th>Module</th>
<th>Parameters</th>
<th>Resources</th>
<th>Contained Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>r244</td>
<td>DW01_sub</td>
<td>width=8</td>
<td></td>
<td>sub_31/minus</td>
</tr>
<tr>
<td>r246</td>
<td>DW01_add</td>
<td>width=8</td>
<td></td>
<td>add_30/plus</td>
</tr>
</tbody>
</table>
```

```
Implementation Report
```

```
|                  |                  | Current          | Set              |
| Cell             | Module           | Implementation   | Implementation   |
| add_30/plus      | DW01_add         | rpl              |                  |
| sub_31/minus     | DW01_sub         | rpl              |                  |
```

No multiplexors to report

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

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

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

Before generating the VHDL netlist, it is required to apply some VHDL naming rules to the design. This is done by entering the following command in the console (be sure that the entity `addsub_NBITS8` is selected in the hierarchy window):

```
change_names -hierarchy -rules vhdl -verbose
```
Save the mapped design in the file `addsub_dfl_nbits8_mapped.vhdl` in the directory `./HDL/GATE`.

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

Click **Save**. The console now echoes the equivalent command line:

```
write -hierarchy -format vhdl
-output .../ADDSUB/HDL/GATE/addsub_dfl_nbits8_mapped.vhdl
```

The console issues many warnings about the creation of dummy nets. This is because QN ports of inferred flip-flops are not used and the VHDL generator does not want to have open ports.

To generate the SDF file, select the main menu item **File -> Save info -> Design Timing...**.

Save the timing file in the file `./SYN/TIM/addsub_dfl_nbits8_mapped.sdf`

Click **OK**. The console now includes the equivalent command line and some informational message:

```
write_sdf -version 2.1 SYN/TIM/addsub_dfl_nbits8_mapped.sdf
Information: Annotated 'cell' delays are assumed to include load delay.
```

The informational message says that the estimated interconnect delays are actually included in the SDF file as part of the cell delays. The generated SDF file actually includes a list of interconnect delays of zero values.

Before generating the Verilog netlist, it is better to reload the database and apply specific Verilog naming rules to the design. This is done by selecting **File -> Remove All Designs** from the main menu, then reading the database file `./SYN/DB/addsub_dfl_nbits8_mapped.db`, and entering the following command in the console (be sure that the entity `addsub_NBITS8` is selected in the hierarchy window):

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

Save the mapped design in the file `addsub_dfl_nbits8_mapped.v` in the directory `./HDL/GATE`.

Click **Save**. The console now echoes the equivalent command line:

```
write -hierarchy -format verilog
-output .../ADDSUB/HDL/GATE/addsub_dfl_nbits8_mapped.v
```
3.9 Design constraints generation for placement and routing

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

```
write_sdc -nosplit SYN/SDC/addsub_dfl_nbits8_mapped.sdc
```

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

3.10 Design optimization with tighter constraints

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

1. Read the elaborated design. It is not necessary to re-analyze the VHDL sources.
2. Specify the clock with a 5 ns period.
3. Save the new elaborated entity in the file ./SYN/DB/addsub_dfl_nbits8_5ns_elab.db.
4. Map and optimize the design.
5. Save the mapped design in the file ./SYN/DB/addsub_dfl_nbits8_5ns_mapped.db.
6. Get the new area, timing and resources reports. Compare with the reports you got for the 10 ns clock period.
7. Generate the VHDL gate-level netlist in ./HDL/GATE/addsub_dfl_nbits8_5ns_mapped.vhd and the associated SDF timing data file in ./SYN/TIM/addsub_dfl_nbits8_5ns_mappedsdf.
8. Do a post-synthesis simulation.
9. Generate the Verilog gate-level netlist in ./HDL/GATE/dfl_nbits8_5ns_mapped.v.
10. Save the design constraints for placement and routing in the file ./SYN/SDC/addsub_dfl_nbits8_5ns_mapped.sdc

3.11 Using scripts

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

An example of such a script for the synthesis of the adder-subtractor design has been installed in the SYN/BIN directory (see “1.5 VHDL example: Adder-subtractor”). The script must be run from the project top directory and it assumed a directory organization as described in “1.3 Design project organisation”.

To run the Tcl script, execute the following command in a Unix shell:

```
[80]...-ADDSUB> dc_shell -tcl_mode -db_mode -f SYN/BIN/addsub_syn.tcl
```

The script is given below. It may be modified to define design information and constraints and to control the flow to some extent.
# Synopsys DC shell script for adder-subtractor.
# Process: AMS 0.35u CMOS (C35), Hit-Kit 3.70
#
# It is assumed that a project directory structure has already been
# created using 'create_project' and that this synthesis script is
# executed from the project root directory $PROJECT_DIR
#---------------------------------------------------------------
set PROJECT_DIR [pwd]
#---------------------------------------------------------------
# Design related information (can be changed)
#---------------------------------------------------------------
set VHDL_ENTITY addsub
set VHDL_ARCH dfl
set NBITS 8
set CLK_NAME clk
set CLK_PERIOD 10;   # ns
set INPUT_DELAY 2;   # ns
set OUTPUT_DELAY 2;  # ns
set OPERATING_COND WORST-IND
#---------------------------------------------------------------
# Flags that drive the script behavior (can be changed)
#---------------------------------------------------------------
# RESOURCE_AREA (0 | 1)
#   if 1, force the tool to share operators as much as possible
# COMPILER_SIMPLE (0 | 1)
#   if 1, only do a single compile with default arguments
#   if 0, do a two-step compilation with ungrouping in between
# OPT (string)
#   can be used to have different mapped file names
#---------------------------------------------------------------
set RESOURCE_AREA 1
set COMPILER_SIMPLE 1
set OPT "_10ns"        ;# to denote the 10ns clock period case
#---------------------------------------------------------------
# File names
#---------------------------------------------------------------
set SOURCE_FILE_NAME          ${VHDL_ENTITY}_${VHDL_ARCH}
set ROOT_FILE_NAME            ${VHDL_ENTITY}_${VHDL_ARCH}_nbits${NBITS}
set VHDL_SOURCE_FILE_NAME     ${SOURCE_FILE_NAME}.vhd
set ELAB_FILE_NAME            ${ROOT_FILE_NAME}_elab
set MAPPED_FILE_NAME          ${ROOT_FILE_NAME}${OPT}_mapped
set DB_ELAB_FILE_NAME         ${ELAB_FILE_NAME}.db
set DB_MAPPED_FILE_NAME       ${MAPPED_FILE_NAME}.db
set VHDL_NETLIST_FILE_NAME    ${MAPPED_FILE_NAME}.vhd
set VLOG_NETLIST_FILE_NAME    ${MAPPED_FILE_NAME}.v
set SDF_FILE_NAME             ${MAPPED_FILE_NAME}.sdf
set SDC_FILE_NAME             ${MAPPED_FILE_NAME}.sdc
set RPT_AREA_FILE_NAME        ${MAPPED_FILE_NAME}_area.rpt
set RPT_TIMING_FILE_NAME      ${MAPPED_FILE_NAME}_timing.rpt
set RPT_REFERENCES_FILE_NAME  ${MAPPED_FILE_NAME}_references.rpt
set RPT_CELLS_FILE_NAME       ${MAPPED_FILE_NAME}_cells.rpt
#---------------------------------------------------------------
# Absolute paths
#---------------------------------------------------------------
set VHDL_SOURCE_FILE          ${PROJECT_DIR}/HDL/RTL/${VHDL_SOURCE_FILE_NAME}
set VHDL_NETLIST_FILE         ${PROJECT_DIR}/HDL/GATE/${VLOG_NETLIST_FILE_NAME}
set DB_ELAB_FILE              ${PROJECT_DIR}/SYN/DB/${DB_ELAB_FILE_NAME}
set DB_MAPPED_FILE            ${PROJECT_DIR}/SYN/DB/${DB_MAPPED_FILE_NAME}
set SDF_FILE               ${PROJECT_DIR}/SYN/TIM/${SDF_FILE_NAME}
set SDC_FILE               ${PROJECT_DIR}/SYN/SDC/${SDC_FILE_NAME}
set RPT_AREA_FILE          ${PROJECT_DIR}/SYN/RPT/${RPT_AREA_FILE_NAME}
set RPT_TIMING_FILE        ${PROJECT_DIR}/SYN/RPT/${RPT_TIMING_FILE_NAME}
set RPT_RESOURCES_FILE     ${PROJECT_DIR}/SYN/RPT/${RPT_RESOURCES_FILE_NAME}
set RPT_REFERENCES_FILE    ${PROJECT_DIR}/SYN/RPT/${RPT_REFERENCES_FILE_NAME}
set RPT_CELLS_FILE         ${PROJECT_DIR}/SYN/RPT/${RPT_CELLS_FILE_NAME}

#   Analyze RTL source
analyze -format vhdl -lib WORK $VHDL_SOURCE_FILE

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

#   Define environment
set_operating_conditions -library c35_CORELIB $OPERATING_COND

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

# Use only plain DFF cells
set_dont_use [list c35_CORELIB.db:c35_CORELIB/DFE* \\   c35_CORELIB.db:c35_CORELIB/DFS* \\   c35_CORELIB.db:c35_CORELIB/TF* \\   c35_CORELIB.db:c35_CORELIB/JK*]
set_fix_multiple_port_nets -all

#   Set resource allocation and implementation
if { $RESOURCE_AREA } {
    set_resource_allocation area_only
    set_resource_implementation area_only
}

#   Save elaborated design and constraints
write -hierarchy -format db -output $DB_ELAB_FILE

#   Map design to gates
if { $COMPILE_SIMPLE } {
    compile
} else {
    compile -map_effort medium -area_effort medium
    ungroup -all -flatten
    compile -incremental -map_effort high
# Save mapped design
write -hierarchy -format db -output $DB_MAPPED_FILE

# Generate reports
report_area -nosplit > $RPT_AREA_FILE
report_timing -path full \
    -delay max \ 
    -nworst 1 \ 
    -max_paths 1 \ 
    -significant_digits 2 \ 
    -nosplit \ 
    -sort_by group \ 
    > $RPT_TIMING_FILE
report_resources -nosplit -hierarchy > $RPT_RESOURCES_FILE
report_reference -nosplit > $RPT_REFERENCES_FILE
report_cell -nosplit > $RPT_CELLS_FILE

# Generate VHDL netlist
change_names -rule vhdl -hierarchy -verbose
write -format vhdl -hierarchy -output $VHDL_NETLIST_FILE

# Generate SDF data
write_sdf -version 2.1 $SDF_FILE

# Generate Verilog netlist
remove_design -all
read_file -format db $DB_MAPPED_FILE
change_names -rule verilog -hierarchy -verbose
write -format verilog -hierarchy -output $VLOG_NETLIST_FILE

# Save system constraints
write_sdc -nosplit $SDC_FILE
Chapter 4: Standard cell placement and routing

This chapter presents the main steps to perform the placement and the routing of the synthesized gate-level netlist using standard cells from the AMS design kit. The tool used here is Cadence Encounter. The `cdsdoc` command gives access to the Cadence documentation.

4.1 Starting the Encounter graphical environment

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

```
[85]vachoux@lsmsun1-ADDSUB> encounter -log PAR/LOG/encounter -overwrite
```

The `PAR/LOG` directory will contain the log file of the session and a file that includes all commands entered in the session. If the `-overwrite` switch is not used, both log and command files are incremented at each new session.

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

The main window includes three different design views that you can toggle during a session: the Floorplan view, the Amoeba view, and the Physical view.

The Floorplan view displays the hierarchical module and block guides, connection flight lines, and floorplan objects, including block placement, and power/ground nets. The Amoeba view displays the outline of the modules and sub-modules after placement, showing physical locality of the module. The Physical view displays the detailed placements of the module’s blocks, standard cells, nets, and interconnects.

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

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

4.2 Design import

Importing the design into Encounter involves specifying the following set-up information:

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

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

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

- **Power information.** This relates to the power nets to use in the layout.

To start the design import, select **Design -> Design Import...** in the main menu. Then, click on the **Load...** button and load the file ./PAR/CONF/c35b4_std.conf.
This file defines a basic import configuration. There is a number of additions and changes to bring to the initial configuration. The new configuration will then be saved for future uses.

The first information to add is the netlist. Click on the ... button on the right of the Verilog Files field. You get a new dialog window with only one pane. Click on the top-right icon to get the full window.

Remove the VERILOG/none line in the left pane.

Select the Verilog netlist file ./HDL/GATE/addsub_dfl_nbits8_mapped.v, add it to the left pane and close the window.

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

In the “LEF files” and “Common Timing Libraries” fields, you should remove the reference to the IO pad library if you don’t intend to include pads, namely:

```
/softs/dkits/ams/v3.70/artist/HK_C35/LEF/c35b4/IOLIB_4M.lef
/softs/dkits/ams/v3.70/liberty/c35_3.3V/c35_IOLIB_4M.lib
```

In the Power tag, you can keep only the vdd! and gnd! power nets if you won’t add periphery cells.

In the Timing tag, you can specify the timing constraints that you used for synthesis.

Select the file ./SYN/SDC/addsub_dfl_nbits8_mapped.sdc

Not all the constraints in the file are actually supported in Encounter. In our case, only the clock definition is actually imported.
The rest of the settings can be left as is. You can now save the updated configuration in the file 
./PAR/CONF/addsub_nbits8.conf by clicking on the Save... button.

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

In the case of the AMS design kit, a number of similar errors are reported when the LEF file is loaded. The error message is like

**ERROR: Macro '...' obs coordinate y value ... isn't on manufacturing grid. 
It's likely result in placement/routing that can't be manufactured

The errors can be safely ignored. The violating coordinates belong to obstructions in the cell LEF file that describe 45 degree shapes. These shapes are not on the manufacturing grid, but this is no problem for the place & route flow. Finally the cells are replaced by the complete layouts anyhow.

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

### 4.3 Global net connections

This step assigns pins or nets to global power and ground nets.

Select Floorplan -> Global Net Connections... in the main menu.

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

Click on Apply.
4.4 Operating conditions definition

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

Select the Timing -> Specify Analysis Conditions -> Specify Operating Condition/PVT... in the main menu.

In the max tab, select the WORST-IND operating condition. In the min tab, select the BEST-IND operating condition.

Click OK.

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

4.5 Floorplan Specification

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

Select Floorplan -> Specify Floorplan... in the main menu.

Define an aspect ratio of 1.

The default core utilization is 85%, which means that 15% of the core area will be free for possible future buffer insertions or cell replacements. The other utilization ratio (Std. Utilization) refers to the density of the standard cells.

Define I/O to core distances of 16 microns. This is enough as there won’t be any periphery cells (I/O pads) in the layout.

The cell rows will be abbutted and flipped each two row to share VDD and ground power rails.
Click **OK**. The display design area pane now shows the defined floorplan with the required number of rows.

It is a good idea to save the design at that stage to allow restarting here quickly without needing to redo all the previous steps. Select **Design -> Save Design...** in the main menu and save the current state in the file `./PAR/DB/addsub_nbits8-fplan.enc`. The data are actually saved in the directory `./PAR/DB/addsub_nbits8-fplan.enc.dat`.

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

### 4.6 CAP cell placement

The AMS design kit requires that physical-only termination cells must be placed at each row ends. The so-called CAP cells are used to properly bias the P+ and N+ substrates.

To do this, select **Place -> Filler -> Add End Cap...** in the main menu. Then, select the `ENDACPL` cell as the cell to place at the beginning of each row and the `ENDCAPR` cell as the cell to place at the end of each row.

Click **OK**. The added cells are now visible in the Physical view.
4.7 Power ring/stripe creation and routing

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

Select Floorplan -> Power Planning -> Add Rings... in the main menu.

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

The ring configuration defines ring widths of 4 micron spaced by 0.6 micron. The rings will be placed in the center of the channel between the core and the chip boundary (or the IO pads, if any).

It is possible to extend the ring segments to reach the core boundary. Click on the Advanced tab and click on the segments you’d like to extend. Other power and ground side trunks can be defined by selecting only horizontal or vertical segments.

Click OK to generate the rings.
To add power stripes, select **Floorplan -> Power Planning -> Add Stripes...** in the main menu.

The Net(s) field defines the pattern. Here a single pattern will be generated (select Number of sets and insert 1). Each stripe will be 2 micron wide and the space between them will be 1 micron.

Check Relative from core or selected area and enter the value 60 (micron) in the X from left field. The two stripes will be vertically placed near the center of the core. It is possible to measure sizes by using the ruler (press ESC to remove all rulers).

Click **OK** to generate the stripes.

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

It is recommended to save the new stage of the design. Select **Design -> Save Design...** in the main menu and save the current state in the file /PAR/DB/addsub_nbits8-pring.enc.
4.8 Core cell placement

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

Select **Place -> Place...** in the main menu.

Select the **Timing Driven** option. This is only valid if timing constraints have been properly defined on the design. This option will optimize the placement of the cells that are on the critical path.

Click **OK** to do the placement. It may take some time to complete, especially when the placement is timing driven.

If you do not intend to synthesize a clock tree (see "4.10 Clock tree synthesis (optional)"), you can now complete the placement with filler cells. Filler cells will fill remaining holes in the rows and ensure the continuity of power/ground rails and N+P+ wells in the row.

To fill the holes with filler cells, select **Place -> Filler -> Add Filler...** in the main menu. Select the cells FILL1, FILL2, FILL5, FILL10, and FILL25 and click **OK** to place the filler cells.

Do not add filler cells now if you intend to synthesize a clock tree. Clock tree synthesis may need some free space to insert clock buffers to balance the clock paths.

The placement should then look like below:

It is recommended to save the new stage of the design. Select **Design -> Save Design...** in the main menu and save the current state in the file ./PAR/DB/addsub_nbits8-placed.enc (if no filler cell inserted) or ./PAR/DB/addsub_nbits8-placed_filled.enc (if filler cells inserted).
4.9 Timing analysis

It is now possible to get a first report on the design timings with the current placement.

Select Timing -> Timing Analysis... in the main menu. Define the path for the slack report file. Click OK.

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

*** Dump slack report (0:00:00.0)
**Info: no slack violation path.

Slack File : PAR/RPT/addsub_nbits8.slk
targetSlack : 0.000 ns
---------------------------------- ------- ------- -------
Slack Range (ns) Count Sum
---------------------------------- ------- ------- -------
( 1.116 ~ 1.000] 1 1
( 1.000 ~ 1.500] 0 1
---------------------------------- ------- ------- -------
Total negative slacks (TNS)=0
Worst negative slacks (WNS)=1.116

The design is not critical as the slack is positive (1.116ns). The slack report file includes more detailed information.

Another useful view is the slack histogram which displays the distribution of the computed slacks.
Select Timing -> Timing Debug -> End Point Slack Histogram... in the main menu. The input file is the slack report file generated previously. Check the Report non Violating box to see all the paths in the design. Click Apply to generate the histogram

Clicking in a histogram bar displays the related information in the slack report file.
4.10 Clock tree synthesis (optional)

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

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

Select Clock -> Create Clock Tree Spec... in the main menu.

Select the BUF2 and INV0 cells for footprints and save the specification in the file .PAR/CTS/addsub_nbits8-spec.cts.

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

The next step is to load the clock tree specification file. Select Clock -> Specify Clock Tree ... in the main menu and select the file .PAR/CTS/addsub_nbits8-spec.cts that has been previously created (and possibly edited).

To create the clock tree, select Clock -> Synthesize Clock Tree ... in the main menu. Define the result directory as .PAR/CTS and the base file name as addsub_nbits8.cts. Click OK to create the clock tree.

Now it is possible to run timing optimization based on the generated clock tree.

Select Timing -> Optimization... in the main menu and check PostCTS. Click OK to do the optimization.
The console now displays the results of the timing optimization:

```
optDesign
------------------------------------------------------------
Setup mode
  Worst Slack: 1.120ns
  TNS: 0.000ns  Violating Paths: 0
Pathgroup Slacks
  reg2reg: 2.900ns
  in2reg: 1.120ns
  reg2out: 6.965ns
  in2out: 0.000ns
Density: 106.579%
Routing Overflow: 0.00% H and 0.00% V
Real DRV (fanout, cap, tran): (0, 0, 0)
Total DRV (fanout, cap, tran): (0, 0, 0)
------------------------------------------------------------
```

The slack did not improve from the preCTS timing analysis as the design is pretty small.

You can now add the filler cells as described in "4.8 Core cell placement".

It is recommended to save the new stage of the design. Select Design -> Save Design... in the main menu and save the current state in the file ./PAR/DB/addsub_nbits8-cts.enc.

### 4.11 Design routing

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

To route the design, select Route -> Nanoroute... in the main menu, check the Timing Driven box and click OK to do the routing. You now get the routed design:

![Routed Design](image)

It is recommended to save the new stage of the design. Select Design -> Save Design... in the main menu and save the current state in the file ./PAR/DB/addsub_nbits8-routed.enc.
4.12 Timing analysis

Following "4.9 Timing analysis", it is now possible to get a new timing analysis report.

4.13 Design checks

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

Select Verify -> Verify Connectivity... in the main menu. Define the report file as PAR/RPT/addsub_nbits8-conn.rpt. Click OK. The console displays the results:

```
*********** Start: VERIFY CONNECTIVITY ***********
Start Time: Thu Dec 1 18:52:00 2005
Design Name: addsub_NBITS8
Database Units: 1000
Design Boundary: (0.0000, 0.0000) (153.0500, 136.9000)
Error Limit = 1000; Warning Limit = 50
Check all nets
Begin Summary
  Found no problems or warnings.
End Summary

End Time: Thu Dec 1 18:52:00 2005
*********** End: VERIFY CONNECTIVITY ***********
Verification Complete : 0 Viols. 0 Wrngs.
```

Select Verify -> Verify Geometry... in the main menu. In the Advanced tab, define the report file as PAR/RPT/addsub_nbits8-geom.rpt. Click OK. The console displays the results:

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

Verification Complete : 0 Viols. 0 Wrngs.

***********End: VERIFY GEOMETRY***********
### 4.14 Report generation

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

**Tools -> Netlist Stats** gives the following output in the console:

```
*** Statistics for net list addsub_NBITS8 ***
Number of cells = 83
Number of nets = 70
Number of tri-nets = 0
Number of degen nets = 1
Number of pins = 192
Number of I/Os = 27

Number of nets with 1 terms = 1 (1.4%)
Number of nets with 2 terms = 65 (92.9%)
Number of nets with 3 terms = 1 (1.4%)
Number of nets with 9 terms = 1 (1.4%)
Number of nets with >=10 terms = 2 (2.9%)

*** 11 Primitives used:
Primitive ENDCAPR (8 insts)
Primitive ENDCAPL (8 insts)
Primitive TIE0 (1 insts)
Primitive FILL1 (6 insts)
Primitive FILL2 (9 insts)
Primitive FILL5 (8 insts)
Primitive XNR21 (1 insts)
Primitive XNR20 (7 insts)
Primitive DFC3 (24 insts)
Primitive ADD32 (8 insts)
Primitive INV3 (3 insts)
************
```

**Tools -> Gate Count Report...** gives the following output in the console:

```
Gate area 54.6000 um^2
```

Finally, **Tools -> Summary Report...** displays the following window:
4.15 Post-route timing data extraction

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

The parasitics must be first extracted.
Select **Timing -> Extract RC...** in the main menu.

The generated Cap file includes the wired capacitance, pin capacitance, total capacitance, net length, wire cap per unit length and the fanout of each net in the design.

The generated SPEF (Standard Parasitics Exchange Format) file includes RC values in a SPICE-like format.

The SDF file may be then generated by selecting **Timing -> Calculate Delay...** in the main menu.

The checked **Ideal Clock** switch means that flip-flops are considered as having 0ps rising and falling transition times.

4.16 Post-route netlist generation

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

Select **Design -> Save -> Netlist...** in the main menu.

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

4.17 GDS2 file generation

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

To export the design in the GDS2 format, select **Design -> Save -> GDS...** in the main menu.

The GDS map file has been copied bay the AMS setup script in the **PAR/DEX** directory. The generated GDS2 file is written in the same directory.
4.18 Design import in Virtuoso

The GDS2 file can be imported in Virtuoso for further processing. To do that, it is recommended to start the Cadence IC environment in the LAY directory.

Start Cadence IC:

```
ams_cds -t c35b4 -m fb
```

Create a new library called ADDSUB with the attached technology file TECH_C35B4.

In the DFII shell, select `File -> Import -> Stream...`

![Import dialog box](image)

In the Options tab, check the `Retain Reference Library (No Merge)` option. Since the cell layouts are available in the CORELIB library, it is possible to display the full layout of the block.
4.19 Using scripts

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

An example of such a script for placement and routing of the adder-subtractor design has been installed in the PAR/BIN directory (see “1.5 VHDL example: Adder-subtractor”). The script must be run from the project top directory and it assumes a directory organization as described in “1.3 Design project organisation”.

To run the Tcl script, execute the following command in a Unix shell:

```
[vachoux@lsmsun1-ADDSUB> encounter PAR/BIN/addsub_par.tcl
```

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

```
# Cadence Encounter Tcl script for adder-subtractor.
#
# Process: AMS 0.35u CMOS (C35), Hit-Kit 3.70
#
#
# It is assumed that a project directory structure has already been
# created using 'create_project' and that this synthesis script is
# executed from the project root directory $PROJECT_DIR
#
#
# Design related information (can be changed)
#
set DESIGN addsub_nbits8
set TIM_LIBRARY C35_CORELIB
set TIM_OC_MAX WORST ;# TYPICAL | WORST | WORST-IND
set TIM_OC_MIN BEST ;# TYPICAL | BEST | BEST-IND

# Floorplan settings
#
set FP_ASPECT_RATIO 1
set FP_ROW_DENSITY 0.85 ;# percent
set FP_CORE2IO 16 ;# micron

# Power ring and settings
#
set PR_WIDTH 4 ;# micron
set PR_SPACING 1 ;# micron
set PR_LAYER_TB MET1 ;# top and bottom layer
set PR_LAYER_LR MET2 ;# left and right layer

# Power stripe settings
#
set ST_NUM_SETS 1 ;# number of sets
set ST_SPACING 1 ;# micron
set ST_LAYER_V $PR_LAYER_LR
set ST_WIDTH 2 ;# micron
set ST_XOFS_R 60 ;# micron
set ST_XOFS_L 60 ;# micron

# Placement settings
#
set PL_EFFORT -high ;# -low | -medium | -high
```
# Clock tree synthesis settings
#
set CTS_BUFFER    BUF2
set CTS_INV       INV0

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

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

#-------------------------------------------------------------
# Absolute paths
#-------------------------------------------------------------
set CONF_FILE              ${PROJECT_DIR}/PAR/CONF/${CONF_FILE_NAME}
set SAVE_DESIGN_FP_FILE    ${PROJECT_DIR}/PAR/DB/${SAVE_DESIGN_FP_NAME}
set SAVE_DESIGN_PR_FILE    ${PROJECT_DIR}/PAR/DB/${SAVE_DESIGN_PR_NAME}
set SAVE_DESIGN_PL_FILE    ${PROJECT_DIR}/PAR/DB/${SAVE_DESIGN_PL_NAME}
set SAVE_DESIGN_PF_FILE    ${PROJECT_DIR}/PAR/DB/${SAVE_DESIGN_PF_NAME}
set SAVE_DESIGN_CT_FILE    ${PROJECT_DIR}/PAR/DB/${SAVE_DESIGN_CT_NAME}
set SAVE_DESIGN_RO_FILE    ${PROJECT_DIR}/PAR/DB/${SAVE_DESIGN_RO_NAME}
set SDF_FILE               ${PROJECT_DIR}/PAR/TIM/${SDF_FILE_NAME}
set SPEF_FILE              ${PROJECT_DIR}/PAR/TIM/${SPEF_FILE_NAME}
set TIM_RCDB_FILE          ${PROJECT_DIR}/PAR/TIM/${TIM_RCDB_NAME}
set RPT_CHECK_TA_FILE      ${PROJECT_DIR}/PAR/RPT/${RPT_CHECK_TA_NAME}
set RPT_REPORT_TA_FILE     ${PROJECT_DIR}/PAR/RPT/${RPT_REPORT_TA_NAME}
set RPT_SLACK_FILE         ${PROJECT_DIR}/PAR/RPT/${RPT_SLACK_NAME}
set RPT_GATE_COUNT_FILE    ${PROJECT_DIR}/PAR/RPT/${RPT_GATE_COUNT_NAME}
set RPT_NOTCH_FILE         ${PROJECT_DIR}/PAR/RPT/${RPT_NOTCH_NAME}
set RPT_GEOM_FILE          ${PROJECT_DIR}/PAR/RPT/${RPT_GEOM_NAME}
set RPT_DENSITY_FILE       ${PROJECT_DIR}/PAR/RPT/${RPT_DENSITY_NAME}
set VLOG_NETLIST_SIM_FILE  ${PROJECT_DIR}/HDL/GATE/${VLOG_NETLIST_SIM_NAME}
set VLOG_NETLIST_LVS_FILE  ${PROJECT_DIR}/HDL/GATE/${VLOG_NETLIST_LVS_NAME}
set CTS_SPEC_FILE          ${PROJECT_DIR}/PAR/CTS/${CTS_SPEC_NAME}
set CTS_RGUIDE_FILE        ${PROJECT_DIR}/PAR/CTS/${CTS_RGUIDE_NAME}
set CTS_RPT_FILE           ${PROJECT_DIR}/PAR/RPT/${CTS_RPT_NAME}
set GDS_FILE               ${PROJECT_DIR}/PAR/DEX/${GDS_FILE_NAME}
set GDS_MAP_FILE           ${PROJECT_DIR}/PAR/DEX/gds2.map

#-----------------------------------------------------
# Procedures
#-----------------------------------------------------

# make_clock_tree
#
proc make_clock_tree create_spec {
    global CTS_BUFFER CTS_INV CTS_SPEC_FILE CTS_RGUIDE_FILE CTS_RPT_FILE
    if { $create_spec || ![file exists $CTS_SPEC_FILE] } {
        createClockTreeSpec \ 
            -bufFootprint $CTS_BUFFER \ 
            -invFootprint $CTS_INV \ 
            -output $CTS_SPEC_FILE
    }

    specifyClockTree -clkfile $CTS_SPEC_FILE
    cksynthesis \ 
        -rguide $CTS_RGUIDE_FILE \ 
        -report $CTS_RPT_FILE
    optDesign -postCTS -setup -drv -outDir PAR/RPT
}

#-----------------------------------------------------
# Load configuration file
#-----------------------------------------------------
loadConfig $CONF_FILE 0
commitConfig

#-----------------------------------------------------
# Set operating conditions
#-----------------------------------------------------
setOpCond \ 
    -maxLibrary $TIM_LIBRARY -max $TIM_OC_MAX \ 
    -minLibrary $TIM_LIBRARY -min $TIM_OC_MIN

#-----------------------------------------------------
# Set user grids
#-----------------------------------------------------
setPreference ConstraintUserXGrid 0.1
setPreference ConstraintUserXOffset 0.1
setPreference ConstraintUserYGrid 0.1
setPreference ConstraintUserYOffset 0.1
setPreference SnapAllCorners 1
setPreference BlockSnapRule 2

#-----------------------------------------------------
# Define global Power nets - make global connections
#-----------------------------------------------------
clearGlobalNets
globalNetConnect vdd! -type pgpin -pin vdd! -inst * -module {} -verbose
globalNetConnect gnd! -type pgpin -pin gnd! -inst * -module {} -verbose
#globalNetConnect vdd3o! -type pgpin -pin vdd3o! -inst * -module {} -verbose
#globalNetConnect vdd3r1! -type pgpin -pin vdd3r1! -inst * -module {} -verbose
#globalNetConnect vdd3r2! -type pgpin -pin vdd3r2! -inst * -module {} -verbose
#globalNetConnect gnd3o! -type pgpin -pin gnd3o! -inst * -module {} -verbose
#globalNetConnect gnd3r! -type pgpin -pin gnd3r! -inst * -module {} -verbose

# Initialize floorplan
floorPlan -r $FP_ASPECT_RATIO \ $FP_ROW_DENSITY \ $FP_CORE2IO $FP_CORE2IO $FP_CORE2IO $FP_CORE2IO
fit
saveDesign $SAVE_DESIGN_FP_FILE

# Add CAP cells
addEndCap -preCap ENDCAPL -postCap ENDCAPR -prefix ENDCAP

# Create and route power rings and power stripes
addRing \ -nets { gnd! vdd! } \ -width_bottom $PR_WIDTH -width_top $PR_WIDTH \ -width_left $PR_WIDTH -width_right $PR_WIDTH \ -spacing_bottom $PR_SPACING -spacing_top $PR_SPACING \ -spacing_left $PR_SPACING -spacing_right $PR_SPACING \ -layer_bottom $PR_LAYER_TB -layer_top $PR_LAYER_TB \ -layer_left $PR_LAYER_LR -layer_right $PR_LAYER_LR \ -center 1 \ -tl 1 -tr 1 -bl 1 -br 1 -lt 1 -lb 1 -rt 1 -rb 1 \ -stacked_via_bottom_layer MET1 -stacked_via_top_layer MET4 \ -threshold 0.7
if { $ADD_STRIPES } {
  addStripe \ -nets { gnd! vdd! } \ -number_of_sets $ST_NUM_SETS \ -spacing $ST_SPACING \ -layer $ST_LAYER_V \ -width $ST_WIDTH \ -xleft_offset $ST_XOFS_L
}sroute \ -jogControl { preferWithChanges differentLayer } \ -nets { gnd! vdd! }
saveDesign $SAVE_DESIGN_PR_FILE

# Core cell placement
if { $PLACE_TIMING } {
  amoebaPlace $PL_EFFORT -timingdriven
} else {
  amoebaPlace $PL_EFFORT
} setDrawMode place
saveDesign $SAVE_DESIGN_PL_FILE

# Create clock tree (optional)
if { $CLOCK_TREE } {

Top-down digital design flow / Chapter 4: Standard cell placement and routing

```bash
make_clock_tree $CTS_CREATE_SPEC
saveDesign $SAVE_DESIGN_CT_FILE
#
# Add filler cells
#---------------------------------------------------------------
addFiller -cell FILL25 FILL10 FILL5 FILL2 FILL1 -prefix FILLER
saveDesign $SAVE_DESIGN_PF_FILE
#
# Route design (Nanoroute)
#---------------------------------------------------------------
if { $ROUTE_TIMING } {
    setNanoRouteMode -quiet -timingEngine CTE
    setNanoRouteMode -quiet -routeWithTimingDriven true
    setNanoRouteMode -quiet -routeTdrEffort 0
}
globalDetailRoute
optDesign -postRoute -setup -drv -outDir PAR/RPT
saveDesign $SAVE_DESIGN_RO_FILE
setDrawMode place
#
# Verifications
#---------------------------------------------------------------
fillNotch -report $RPT_NOTCH_FILE
verifyConnectivity \
    -type all \n    -error 1000 \n    -warning 50 \n    -report $RPT_CONN_FILE
verifyGeometry \n    -allowSameCellViols \n    -allowRoutingBlkgPinOverlap \n    -allowRoutingCellBlkgOverlap \n    -report $RPT_GEOM_FILE
verifyMetalDensity \n    -detailed \n    -report $RPT_DENSITY_FILE
#
# Extract parasitics
#---------------------------------------------------------------
setExtractRCMode \n    -detail \n    -rcdb $TIM_RCDB_FILE \n    -relative_c_t 0.01 \n    -total_c_t 5.0 \n    -reduce 5
extractRC
#
# Generate RC and timing files
#---------------------------------------------------------------
rcOut -spef $SPEF_FILE
delayCal -sdf $SDF_FILE
#
# Generate reports
#---------------------------------------------------------------
reportGateCount -outfile $RPT_GATE_COUNT_FILE
#
# timings
#---------------------------------------------------------------
setCteReport
setAnalysisMode -setup -async -skew -noClockTree -sequentialConstProp
reportAnalysisMode
```
buildTimingGraph
checkTA -verbose > $RPT_CHECK_TA_FILE
reportTA \n  -format { hpin arc cell delay arrival required slew fanout load } \n  -late \n  -max_points 10 \n  -net \n  > $RPT_REPORT_TA_FILE

#------------------------------------------------------------------------------
#  Save netlist
#------------------------------------------------------------------------------
saveNetlist -excludeLeafCell $VLOG_NETLIST_SIM_FILE
saveNetlist -physical $VLOG_NETLIST_LVS_FILE

#------------------------------------------------------------------------------
#  Save GDS2
#------------------------------------------------------------------------------
streamOut $GDS_FILE \n  -mapFile $GDS_MAP_FILE \n  -libName ADDSUB \n  -structureName $DESIGN_NAME \n  -stripes $ST_NUM_SETS \n  -units 1000 \n  -mode ALL