Integrated Systems Architectures

Simulation with Siemens QuestaSim

In this course we are going to simulate HDL based designs with QuestaSim. To use QuestaSim first we have to source the initialization script as follows:
`source /eda/scripts/init_questa_core_prime`.

Then we can start building our project. There are two main strategies to work with QuestaSim. The first is creating a project, the second is issuing commands. In the following we will detail the second one. Assuming that we have an `src` directory containing the design and a `tb` directory containing the test-bench, we first create a `sim` directory where we will perform the simulation. In the `sim` directory, we have to create a working directory, named `work`, where QuestaSim will write intermediate files. This is obtained by issuing:
`vlib work`

Then we can start compiling the HDL files. For each VHDL source file in the `src` directory we write
`vcom -work ./work ../src/<VHDL_file>.vhd`

A similar command line is required for VHDL files in the `tb` directory. On the other hand, a SystemVerilog file in the `tb` directory is compiled as
`vlog -sv -work ./work ../tb/<verilog_file>.sv`
Note that `vlog` compiles Verilog files as well by removing the option `-sv`.

Finally, if the test-bench module is named `tb_fir` we start the simulation as
`vsim work.tb_fir -voptargs=+acc`.
This command opens the simulator graphical interface and allows for adding signals to the waveform viewer and running simulations. As an example, to run a 100 ns simulation issue
`run 100 ns`.

Suggestion: It is **highly recommended** to perform the simulation using a script. To achieve this, create a text file with the extension *.do* (for example, *script.do*) containing all the commands that would normally be entered in the QuestaSim shell. An example of such a script is shown below:

```
vcom -work ./work ../src/<VHDL_file>.vhd
...
vlog -sv -work ./work ../tb/<SystemVerilog_file>.sv
vsim work.tb_fir -voptargs=+acc
add wave *
run 100 ns
```

Once QuestaSim is open, the script can be executed by entering in the command line:
`do <script_name>.do`

Integrated Systems Architectures

Logic synthesis with Synopsys Design Compiler

To perform the logic synthesis with Synopsys Design Compiler two steps are required before starting.

1. source the initialization script as follows
   `source /eda/scripts/init_design_vision`
2. prepare a setup file for Design Compiler

We assume that the project is structured in directories as suggested in the text of the lab experiences. To ease the design flow go to the `syn` directory and prepare a file named `.synopsys_dc.setup`. **Please note that the first character of the name is '.'.** This file contains the names of the technology libraries used to perform the synthesis and the path to find them in the file-system (see the following description). A copy of the file is available on *Portale della didattica*.

Follows an explanation of the content of .synopsys_dc.setup. First, we define the name of the place used by Design Compiler to write its working files. This place is referred to as WORK. However, we have to bind this name to a real directory. Let `work` be the name of the real directory. The binding is performed as follows:

```
define_design_lib WORK -path ./work
```

Then, we define the path to the two directories (one for Design compiler and one for the technology) where libraries are stored as follows:

```
set search_path [list .  /eda/synopsys/2021-22/RHELx86/SYN_2021.06-SP4/libraries/syn
/eda/dk/nangate45/synopsys ]
```

Finally, we have to set the names of the libraries:

```
set link_library [list "*" "NangateOpenCellLibrary_typical_ecsm.db"
"dw_foundation.sldb" ]
set target_library [list "NangateOpenCellLibrary_typical_ecsm.db" ]
set synthetic_library [list "dw_foundation.sldb" ]
```

The setup proposed for this lab makes use of the DesignWare library. Thanks to this library the synthesizer detects known basic blocks (e.g. multipliers, adders, ...). For each of these blocks it can choose different implementations to meet the constraints imposed by the user.

Please note that you have to create the `work` directory as

```
mkdir work
```

At this point we can start design compiler and issue the commands to perform the logical synthesis. Design Compiler can be started in two modes: graphical mode and shell mode. The graphical mode can be launched as `design_vision`, whereas, the the shell mode is launched as `dc_shell-xg-t`. Note that in both cases when you issue a command Design Compiler shows you the output of your command on *standard output*. However, you can redirect the output on a file; thus, you have the possibility to carefully read all the messages produced by the synthesizer. Redirection is obtained with the standard operator >.

The logical synthesis process can be divided into the following steps:

- reading source files;
- applying constraints;
- start the synthesis;
- save the results.

Reading source files The first command to import source files in Design Compiler is `analyze`. We have to provide the format of the sources (in our case SystemVerilog) with the option `-f sverilog`. Please note that Design Compiler knows Verilog and VHDL as well (`-f verilog` and `-f vhdl`). Then, we specify where to put the working files `-lib WORK`. Finally, we write the name of the source file we want to analyze. So for each source file we give the following command:
`analyze -f sverilog -lib WORK ../src/<file name>.sv`
If we have only one file named `myfir.sv` placed in the `src` directory, we only issue
`analyze -f sverilog -lib WORK ../src/myfir.sv`

Before completing the reading of source, we set one parameter to preserve rtl names in the netlist to ease the procedure for power consumption estimation.
`set power_preserve_rtl_hier_names true`

Then, we can issue the `elaborate` command. It requires some parameters, namely the name of the top entity in the design you synthesize. In our case, we assume the name is `myfir`. Since in VHDL the same entity can have different architectures, Design Compiler permits to specify the name of the architecture. Let us assume that the architecture of `myfir` is named `beh`. Finally, we specify the library where Design Compiler will put the working files, that is WORK. So the command to issue is:
`elaborate myfir -arch beh -lib WORK`

If the design contains multiple instances of the same block, you can issue the `uniquify` command. Finally, issue the `link` command.

Applying constraints In Design Compiler there are a lot of possibilities to apply constraint to a design. One possibility is the following. First, create a symbolic clock signal (e.g. MY_CLK) with a certain clock period and bind it to a real clock pin in your design. Assume you want a 100 MHz clock frequency (period 10 ns) and your clock pin is named `CLK`, you have to issue the following command
`create_clock -name MY_CLK -period 10.0 CLK`.
Then, since the clock is a "special" signal in the design, we set the *dont_touch* property:
`set_dont_touch_network MY_CLK`.
Since the clock could be affected by jitter we can set the *uncertainty* of the clock signal as
`set_clock_uncertainty 0.07 [get_clocks MY_CLK]`
where, if we do not have further information, we can set the uncertainty as a very small value with respect to the clock period. Moreover, each input signal could arrive with a certain delay with respect to the clock. Assuming that all input signals have the same maximum input delay, we can set their input delay as
`set_input_delay 0.5 -max -clock MY_CLK [remove_from_collection [all_inputs] CLK]`.
In general the input delay must be lower than the clock period to avoid slow input paths. Similarly, we can set the maximum delay of output ports:
`set_output_delay 0.5 -max -clock MY_CLK [all_outputs]`. Finally, we can set the load of each output in our design. For the sake of simplicity we assume that the load of each output is the input capacitance of a buffer. Among the buffers available in this technology we choose the *BUF_X4*, which input port is named *A*, so
`set OLOAD [load_of NangateOpenCellLibrary/BUF_X4/A]`
`set_load $OLOAD [all_outputs]`
where `NangateOpenCellLibrary` is the name of the target technology library. You can check the cells available in the technology library by reading one of the available files, such as the liberty file (.lib). For the `NangateOpenCellLibrary` you can check:
`NangateOpenCellLibrary_typical_ecsm.lib`
in `/eda/dk/nangate45/liberty/`.

Start the synthesis One simple command to start the synthesis process is:
`compile`

This command has several options (try `man compile` for details). In particular, if you equip your registers with an enable signal, you can ask the synthesizer to apply *clock gating*:
`compile -gate_clock`

Save the results At this point we can simply collect the results of the synthesis. Some results are useful to understand the result of the synthesis process, others are required to complete the design flow and to estimate the power consumption. Among all the data available after the synthesis process we will observe i) timing verification and ii) area. To verify the timing of the design we simply run the following command:
`report_timing`

This command shows the longest path in the design and if the timing requirement imposed with the `create_clock` statement is met or not. In both cases we see the difference between the timing required and the one achieved (slack), so we can increase or decrease the clock period constraint accordingly. Please note that when the report states "Met" with a low slack for a given clock constraint, it does not mean the synthesizer is not able to do better. You may discover that increasing the clock frequency the synthesizer changes the circuit to meet the new constraint. As an example usually the synthesizer uses ripple-carry adders, but to achieve higher frequency it may use fast architectures as the carry-look-ahead ones.

To obtain the area of the design, we issue the command
`report_area`

Relevant information is grouped in the last lines where the breakdown is given in terms of combinational, buffers/inverters, non-combinational, macro and interconnect areas together with the total cell area and the total area. In our design we have combinational, buffers/inverters and non-combinational areas. We do not have any macro (e.g. RAMs) or interconnect area estimation. Results are expressed in μm^2.

Finally, we can save the data required to complete the design and to perform switching-activity-based power estimation. In the following we assume that the design name is `myfir`. First, we ungroup the cells to flatten the hierarchy as follows:
`ungroup -all -flatten`.

Then, we have to export the netlist in verilog. So that we impose verilog rules for the names of the internal signals. This is obtained with
`change_names -hierarchy -rules verilog`.
We also save a file describing the delay of the netlist:
`write_sdf ../netlist/myfir.sdf`.
We can now save the netlist in verilog:
`write -f verilog -hierarchy -output ../netlist/myfir.v`
and the constraints to the input and output ports in a standard format:
`write_sdc ../netlist/myfir.sdc`.
To close Design Compiler simply type
`quit`.

Some notes The logic synthesizer can use/reuse results of a synthesis. Thus, if you run a synthesis then you change the constraints and run again the synthesis, you obtain a result which might depend on the first synthesis. To make the flow deterministic, so results can be reproduced, it is better to run one synthesis, get the results, and then close Design Compiler. If you need to run a different synthesis, clean the work first, then start Design Compiler and

run the new synthesis.

If you have to find the maximum clock frequency of your design, you can force the period to 0 when applying the constraints. Then run the synthesis and get the timing result, which gives you the minimum period ($T_{\min}$). You can check your result by running a new synthesis forcing the clock period to $T_{\min}$.

If you want to save in a file the output of a command (e.g. `elaborate`) you can redirect it as follows: `elaborate myfir -arch beh -lib WORK > ./elaborate.txt`
where `elaborate.txt` is the file where you will find the output of your command.

Suggestion: It is **highly recommended** to perform the simulation using a script. The script should be contained in a text file with the extension *.scr* (for example, *script.scr*). Similarly to the script created for QuestaSim, the script should contain all the commands that would normally be entered in the Design Compiler shell.
If you are using Design Compiler in graphical mode, you can launch the script using the prompt:
`source <script_name>.scr`
Instead, if you are using Design Compiler in shell mode, you can execute the script and log the output with the command:
`dc_shell-xg-t -f <script_name>.scr > mylogfile.log`

For further details, you can check the Design Compiler User Guide (dcug.pdf) in:
`/eda/synopsys/2021-22/RHELx86/SYN_2021.06-SP4/doc/syn/`

Integrated Systems Architectures

Switching-activity-based power consumption estimation

To perform switching-activity-based power consumption estimation we have to jointly use Questasim and Synopsys Design Compiler. The whole procedure requires several steps, now we will concentrate on each of them.

1. launch Questasim with proper options and record switching activity;
2. convert the file with the switching activity from *vcd* to *saif*;
3. perform power consumption estimation with Synopsys Design Compiler.

Before starting go to the directory that contains your project (e.g. `lab1`) and create one directory named `vcd` and one directory named `saif`.

Launch Questasim with proper option and record switching activity To obtain accurate power consumption estimation we need the switching activity of the nodes into the design. This can be done by simulating the netlist. We still have to specify when the monitoring will finish. If you provided you test-bench with a signal that identifies the end of the simulation, we can simply perform a sufficiently long simulation such that when the simulation is complete there will be no other switching activity. The idea is that we have a signal named `END_SIM_i` that goes to '1' when the simulation is finished. This signal is used to stop the clock and so to stop the activity of the whole circuit.

In the following we will assume that the test-bench file is `tb_fir.sv` and the test-bench module is named `tb_fir`. Similarly, we assume that the design to be simulated is named `myfir` and the corresponding instance in the test-bench is named `UUT`. To simulate we have to compile the verilog netlist produced by Design Compiler and all the test-bench files. Go to the `sim` directory and compile the vhdl files (if any) you are using as stimulus for your test-bench. As an example:

```
vcom -work ./work ../src/clk_gen.vhd
vcom -work ./work ../src/data_maker_new.vhd
vcom -work ./work ../src/data_sink.vhd
```

to compile the Verilog files:

```
vlog -work ./work ../netlist/myfir.v
```

and to compile the SystemVerilog files:

```
vlog -sv -work ./work ../tb/tb_fir.sv
```

Then we include in the project the functional model of the cells in the technology library. Two possible approaches can be used to include cell models: i) include the verilog source of the cells in your project and compile it; ii) compile the verilog source once and link the compiled library. The second option is often preferred even if it requires to compile the verilog source of the cells with a version of Questasim compatible with the one employed by the user who is running the simulation.

Please note that the following description is incremental. Read and understand before pressing any key on the keyboard.

If the compiled library of the cells is located at `/eda/dk/nangate45/verilog/qsim2020.4` we can link it to Questasim by issuing

```
vsim -L /eda/dk/nangate45/verilog/qsim2020.4 work.tb_fir
```

To obtain an accurate switching activity report, we have to link also the delay file (*.sdf* file) generated by Synopsys Design Compiler so

```
vsim -L /eda/dk/nangate45/verilog/qsim2020.4
-sdftyp /tb_fir/UUT=../netlist/myfir.sdf work.tb_fir
```

When Questasim opens you can see some warnings. Some warnings are related to the cells in the technology library and state that there are missing connections. As an example `Too`

`few port connections for '...', Expected 5, found 4.` and `Missing connection for port 'QN'.` means that Design Compiler placed a flip-flop in your design and it is using only one output (Q) whereas the other output (QN) is unused and so not connected. Questasim is telling the line in the verilog file where this happens, so you can double check it. This type of warning can be neglected.

Other warnings are related to negative timing check limits in the sdf file. Questasim forces these values to zero. Usually, this is not a problem, as it is often related to the `RECOVERY` field, which represents the limit of the time between the release of an asynchronous control signal (the asynchronous reset) from the active state and the next active clock edge.

In this design flow we will use value-change-dump (*vcd*) files. Questasim can manage vcd files, so before starting the simulation we open a vcd file where switching information will be written:

```
vcd file ../vcd/myfir_syn.vcd
```

Then, we specify that we want to monitor all the signals inside our unit-under-test (the FIR filter):

```
vcd add /tb_fir/UUT/*
```

Now we can run the simulation. As an example a 2 μs simulation is run as:

```
run 2 us
```

As stated before, in this case the `END_SIM_i` signal is particularly useful as it stops all the switching activity in our design when the simulation is over. Thus, even if the simulation time we specified with the `run` command is longer than required, the vcd file (`myfir_syn.vcd` file in the `vcd` directory) will contain only the correct information.

Convert the file with the switching activity from vcd to saif From the shell where you run Design Compiler you can execute a script named vcd2saif, which makes the conversion. As an example, you can run:

```
vcd2saif -input ../vcd/myfir_syn.vcd -output ../saif/myfir_syn.saif
```

Note that `vcd2saif` script is part of the Synopsys Design Compiler tool suite, so it is available only if you sourced the initialization script to setup Design Compiler.

Power consumption estimation with Synopsys Design Compiler The last step requires to go back to the `syn` directory and to launch Synopsys Design Compiler. First, we have to read back the netlist:

```
read_verilog -netlist ../netlist/myfir.v
```

Note that, if you activate the clock gating during the synthesis, then the netlist might contain two modules: the design and the clock gating module. Thus, you have to specify which is the module you apply the switching activity to. This can be obtained as:

```
current_design myfir
```

given that *myfir* is the name of the module you will apply the switching activity to.
Then, we read the saif file generated by QuestaSim simulation:

```
read_saif -input ../saif/myfir_syn.saif -instance tb_fir/UUT -unit ns -scale 1.
```

We have shown to Design Compiler that there is a clock signal in the design with a certain period (e.g. 10 ns) and so we issue:

```
create_clock -name MY_CLK -period 10.0 CLK.
```

Finally, we can issue the `report_power` command to obtain the power consumption of the design and finally we can quit Design Compiler. Note that the `report_power` command has several options that give you details about the annotation and the internal power consumption breakdown.

Integrated Systems Architectures

Place and route with Cadence Innovus

To perform the place and route operations with Cadence Innovus first we assume you are in a directory containing your project (e.g. `lab1`). Create a directory named `innovus`. Move to the `innovus` directory and source the initialization script:
`source /eda/scripts/init_cadence_2020-21`
Let's go to the tool: it can be launched as `innovus`. To complete the design flow, the following steps are required.

1. Importing the design;
2. Floorplanning;
3. Power planning and routing;
4. Cell placing and clock tree synthesis;
5. Signal routing;
6. Timing and design analysis;

Note that each time you run innovus, it produces two files named `innovus.cmd` and `innovus.log`. The former contains the sequence of commands corresponding to the steps you performed on the gui. The second contains all the messages produced by innovus command after command. Thus, you can carefully check the flow by reading the `innovus.log` file. Moreover, you can modify the `innovus.cmd` file to obtain a script of commands to run innovus in batch-mode by adding at the beginning of the command list the line `source design.globals` and modifying the extension from `.cmd` to `.tcl`. **Before starting dowload from *Portale della didattica* the design.globals and mmm_design.tcl files.**

Importing the design First, you need to prepare a file with setup information. In the following we will refer to this file as `design.globals`. As you can see this file contains two sections: the first has to be customized for your design. The second is already done. Edit the file and setup the first section. First check the input directory is the correct one:
`set IN_DIR "../netlist"`.
Then, set the top of the hierarchy:
`set TopLevelDesign "myfir"`.
Target technology libraries are already set:

```
set LIB_DIR /eda/dk/nangate45/liberty
set MyTimingLib ${LIB_DIR}/NangateOpenCellLibrary_typical_ecsm.lib
set LEF_DIR /eda/dk/nangate45/lef
set LEF_list [list ${LEF_DIR}/NangateOpenCellLibrary.lef]
```

From the top innovus menu import the `design.globals` file: **File** → **Import Design**; a new window opens: select in the menu below **Load** and select from the browse menu the file `design.globals`. Click **Open** in the browse window. Then click on the folder icon of the *Analysis Configuration* menu. Here we specify the setup for the timing analysis: select the mmm_design.tcl file and click **Open**. Finally, click **OK** in the Design Import window: the design is imported with the correct cell reference and descriptions.

Structuring the floorplan At this step Innovus sets the area to be assigned to the cell ensemble. This area usually is the center of the silicon die. On the other hand, the area where the power supply will be routed is made of rings placed around the cell area.

From the Main window top menu select: **Floorplan** → **Specify Floorplan**. An option window opens, allowing you to define the Core aspect ratio (set to 1.0) and utilization (set to

0.6). In the section *Core Margins by* select **Core to die boundary**, and force 5 (μm) from the four sides of the core (unity is μm here). Click **OK** and see what happens. The cells height is already known at this point, as you can note by the gray horizontal lines. All the standards cells have the same height by construction. The width changes coherently with the transistors and interconnections area. At this point innovus knows how many rows will be needed for the design.

Inserting Power Rings In this step the channel defined before will be filled by two metal rings for power (VDD) and ground (VSS) respectively. These metal stripes will be connected with the VDD and VSS pads (if this is the final chip a wirebonding package is supposed here, if not, more probable, these rings will be connected to other rings of other blocks) and will distribute hierarchically the power and ground signal to the whole chip.

From the Main window top menu select: **Power** → **Power Planning** → **Add Rings**. An option window opens. At the top of the window you can fill the *Net(s)* item. Click on the folder icon, a small windows opens. You have to select VDD and VSS from the *Possible Nets* and add them to the *Chosen Nets* using the **Add** >> button. Then, click **OK**.

In the *Add Rings* window go to the *Ring Configuration* section and choose for the *Offset* the option **Center in channel**. Set the width and the spacing to 0.8. Then, click **OK**. Now two rings have been designed: one for VDD and one for VSS (the connection to the electrical generator will be done later). Note that in the corners there are vias for the connection between metals. If you click on one of the stripes you will receive information in the bottom window on the metal layer, on the stripe geometry and on its coordinates. **Note: for medium to large designs one can improve the power supply distribution by inserting stripes, this is not our case.**

Standard cell power routing This operation allows to place horizontal wires preparing the VDD and VSS wires for the standard cells. Such wires will be connected to the ring and (in case) to the vertical stripes as well. From the Main window top menu select: **Power** → **Connect Global Nets**. A window opens. In the section *Connect* of the *Power Ground Connection* side you have to specify the *Pin Name(s)* field. Write **VDD**. Then, go the the field named *To Global Net* and specify **VDD**. Click on **Add to List**. An item will appear on the left side of the window in the *Connection List*. Do the same for **VSS** in order to have both connections in the *Connection List*. Click on **Apply** and then on **Cancel**. Finally, in the Main window top menu select: **Route** → **Special Route**. A window opens. You have to fill the *Net(s)* field. Click on "...". A small window opens where you can select **VDD** and **VSS** from the *Possible nets* and add them to the *Chosen Nets* by pressing **Add** >>. Now on the *SRoute* window click **OK** with the default values.

Placement Now the cells will be placed. Up to this point the only thing known is the total area of the circuit and the number of rows to be used. After this point the layout of each cell will have a unique position in one of the predefined rows. Before starting the placement phase select **Place** → **Specify** → **Placement Blockage** and select layers from 1 to 8. Now click on **Place** → **Place Standard Cell** Then, click on **Mode ...**. A new window opens where you can specify several options. For our purpose we simply add in the *Placement* tab of the *Placement mode* the flag **Place IO Pins**. Then, click on **OK**. The window closes and we are back to the small *Place* window, where we have to click on **OK**.

At this point you can select cells and read their name or zoom in the view for recognizing the input/output pin names. If you click on a pin you will see its connectivity (virtual) to the other pins. In the die border you can see input and output pins (to be connected to the pads) and if you click on one of them you can see its connection to the cell pins. What you see is only an abstract view of each cell, in which only its sizing, its orientation and the position and connectivity of its pins which will be used during the routing phase for connecting the cells among them.

Pre Clock Tree Synthesis optimization This step is optional, but it can help in closing the design respecting all the timing constraints. Till this point Innovus assumes for RC parasitics and timing related issues that the technology is a 90nm one. Thus, it before starting the optimization it is better to set the process to 45nm. This can be obtained with:

```
setDesignMode -process 45
```

Note that for design processes above 32nm parasitics extraction can be performed either with captables or with Quantus techfiles. Since for the Nangate 45 we only have the captables, we will proceed with simple RC extraction not using Quantus. To proceed with the optimization: from the Main window select **ECO** → **Optimize Design**. In the *Design Stage* section choose **Pre-CTS** and in *Optimization type* check **Setup**. Then, click **OK**.

Clock Tree Synthesis (CTS) This phase generates the clock tree. First, we specify the maximum transition time of the clock and the target skew, e.g. 0.08 ns and 0.5 ns. So from the Innovus shell we have to issue the following commands:

```
set_ccopt_property target_max_trans 0.08
set_ccopt_property target_skew 0.5
```

Finally, we can create the clock tree where the *-source* option is used to specify the name of the clock pin, in this example *CLK*:

```
create_ccopt_clock_tree -name MY_CLK -source CLK
ccopt_design
```

You can get information about the clock tree and the skew with the following commands:

```
report_ccopt_clock_trees
report_ccopt_skew_groups
```

Post CTS optimization This step is optional but (as the previous one) it can help in closing the design respecting all the timing constraints. To proceed with the optimization: from the Main window select **ECO** → **Optimize Design**. In the *Design Stage* section choose **Post-CTS** and in *Optimization type* check both **Setup** and **Hold**. Then, click **OK**.

Routing This phase is the last one: the connection among the cells will be performed using the available metal layers: **Route** → **NanoRoute** → **Route** and **OK**. Note in the innovus shell a few messages on the routing iteration steps for this optimization, together with some interesting data on the routing performed: the number of wires used for each layer, the total length routed in each layer, the number of via used.

Post routing optimization Note that at this point the design is complete (even if we don't have used real PADS all around the design). During this (optional) last step we can try to optimize our design to achieve the required timing constraints. Before starting the last optimization you have to issue on innovus command line (the shell where you launched innovus from) the command `setAnalysisMode -analysisType onChipVariation`. Then, in the Main window select **ECO** → **Optimize Design**. In the *Design Stage* section choose **Post-Route** and in *Optimization type* check both **Setup** and **Hold**. Then, click **OK**.

Place filler It is of help for technological reasons to complete the placement with filler cells. These will fill the holes to ensure continuity in n+ and p+ wells in each row. From the top menu select **Place** → **Physical Cell** → **Add Filler**, a new window opens. To fill the *Cell Name(s)* Click **Select** and choose all the fill cells available in the *Cells List*. Then, click on **Add**. You will see the fillers you selected appearing the *Selectable Cells List* column. Click on **Close**. These are possible fill cells among which the placer will choose for filling the placement gaps. Now click **OK** in the *Add Filler* window and see what happens.

Saving the design Before going on, save your design: **File** → **Save Design**. Check that selected *Data Type* is **Innovus** and name the file with a meaningful name. Hereinafter you will be able to import your design at the routed level. You can plot the image by the main menu: from **Tools** you can use either *Snapshot* or *Screen Capture*.

Parasitics extraction For analyzing the time behavior Innovus uses the resistance and capacitance parasitic values for each metal wire. The extraction of such parasitics is the task of this step. The engine is able to compute the resistance and capacitance associated to each rectangle using its properties (technology and geometry information). From the main menu select **Timing** → **ExtractRC**, then click **OK**.

Timing analysis From the main menu select: **Timing** → **Report Timing**, a new window opens. In *Design Stage* choose **Post-Route** and in *Analysis Type* choose **Setup**. Click **OK**. Repeat the same step choosing **Hold** in the *Analysis Type* section. In the directory named *timingReports* you find all the results produced by the timing analysis. Files .slk and .tarpt contain general and detailed informations on the timing paths and violations. Remember that *slack* means "what is remaining between what you asked to have and what you actually have". So, if the slack is positive you are ok (you spared some time), if it is negative you are violating the constraint you decided.

Design analysis and verification Before ending the place and route phase we have to verify the connectivity and the design rules: **Verify** → **Verify Connectivity** and **OK**. Check the message produced by Innovus and verify there are no violations. Usually violations are caused by floating wires. To verify the design rules: **Verify** → **Verify DRC** and **OK**. Check the message shown by Innovus and verify there are no violations. Usually violations are caused by wrong constraints on the geometric feature during the place and route design flow. As an example if we require a narrow spacing between VDD and VSS layers when creating the power supply rings we may violate the design rules imposed by the technology we are employing. We can save area and gate count data as: **File** → **Report** → **Gate Count** and **OK**. Finally, we save i) the post place and route verilog netlist as **File** → **Save** → **Netlist** and **OK**; ii) the file with delay annotation (.sdf) as **Timing** → **Write SDF** and **OK**. You will notice in the shell and in the .log file that Innovus suggest to relaunch the command with the *-recompute_delay_calc* option. Thus, from the Innovus shell you can issue:

```
write_sdf -ideal_clock_network -recompute_delay_calc myfir.sdf
```

Documentation and scripting You can access the on-line documentation by sourcing the initialization script and then typing the command `help_cds_innovus`. Innovus accepts tcl scripting so you can setup the starting point of the design flow by loading the *design.globals* and *mmm_design.tcl* files. The following commands can be used inside the Innovus shell:

```
source design.globals
set init_mmmc_file mmm_design.tcl
init_design
```

Integrated Systems Architectures

Post place and route simulation and switching-activity-based power consumption estimation

To perform post place and route simualtion and switching-activity-based power consumption estimation we have to jointly use Questasim and Cadence Innovus. The whole procedure requires some steps, now we will concentrate on each of them.

1. Questasim simulation and switching activity recording;
2. power consumption estimation with Cadence Innovus.

Questasim simulation and switching activity recording In the following we will assume that the test-bench file is `tb_fir.v` and the test-bench module is named `tb_fir`. Similarly, we assume that the design to be simulated is named `myfir` and the corresponding instance in the test-bench is named `UUT`. Before starting check that your project (e.g. `lab1`) contains a directory named `vcd`, if not, create it. Then, to simulate we have to compile the verilog netlist produced by Innovus and all the test-bench files. Go to the `sim` directory and compile the vhdl files (if any) you are using as stimulus for your test-bench. As an example:

```
vcom -work ./work ../tb/clk_gen.vhd
vcom -work ./work ../tb/data_maker.vhd
vcom -work ./work ../tb/data_sink.vhd
```

Then, to compile the Verilog type:

```
vlog -work ./work ../innovus/myfir.v
```

and the SystemVerilog type: `vlog -sv -work ./work ../tb/tb_fir.sv`
Then we include in the project the functional model of the cells in the technology library. We use the same approach employed in post synthesis simulation:

```
vsim -L /eda/dk/nangate45/verilog/qsim2020.4 work.tb_fir
```

To obtain an accurate switching activity report we have to link also the delay file (*.sdf* file) generated by Cadence Innovus. Note that the *.sdf* file contains only min and max delays. The typical value is not provided, so the command line is

```
vsim -L /eda/dk/nangate45/verilog/qsim2020.4
-sdfmax /tb_fir/UUT=../innovus/myfir.sdf work.tb_fir
```

Now the simulator is ready.

In this design flow we will use value-change-dump (*vcd*) files. For this part of the flow, the approach is the same as the one we used for post-synthesis power estimation. So before runnig the simulation we open a vcd file where switching information will be written:

```
vcd file ../vcd/design.vcd
```

Then, we specify that we want to monitor all the signals inside our unit-under-test (the FIR filter):

```
vcd add /tb_fir/UUT/*
```

Now we can run the simulation. As an example a 2 μs simulation is run as: `run 2 us`
Note that also in this case the `END_SIM_i` signal is particularly useful as it stops all the switching activity in our design when the simulation is over. Thus, even if the simulation time we specified with the `run` command is longer than required, the vcd file will contain only the correct information. Now we can run the simulation and quit, the result will be the switching activity information stored in the `design.vcd` file.

Power consumption estimation with Cadence Innovus The last step requires to go back to the `innovus` directory and to launch Cadence Innovus. First we have to read back the design: **File** $\rightarrow$ **Restore Design**, choose *Innovus* as *Data Type* and then select the design file you want to restore (`myfir` or the design you saved during the place and route) and **OK**. Once the design is re-loaded we have to recover the parasitics: **Timing** $\rightarrow$ **ExtractRC**. Then, we

will read the vcd file produced by Questasim to obtain post place and route power estimation: **Power** → **Power Analysis** → **Setup** and click **OK**. Then, **Power** → **Power Analysis** → **Run** in the *Basic* tab select *Activity File*, specify the *VCD* file `../vcd/design.vcd` and the *Scope*, which is the name of the test-bench module with the name of the filter instance, such as `/tb_fir/UUT`, and click on **Add**. The file will appear in a box with its scope. Finally, click on **OK** to start the power analysis and get the results. From **Power** → **Report** → **Power** you can also generate a report.