ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Integrierte Systeme
Integrated Systems Laboratory

Department of Information Technology and Electrical Engineering

VLSI II: Design of Very Large Scale Integration Circuits

227-0147-00

Exercise 5

Padring and Floorplanning

Prof. Dr. H. Kaeslin
Dr. N. Felber

SVN Rev.: 1779
Last Changed: 2016-10-18

1 Overview

In this exercise, we will start the back-end design process that will bring the chip design from the synthesized netlist all the way to the fabrication data ready to be sent to the factory. The software tool that we will mainly use for the back-end design is CADENCE INNOVUS . We will learn:

- What type of input data and files are needed to run CADENCE INNOVUS .
- How to add I/O drivers and power connections to our design.
- How to do the floor planing, including how to determine the chip area, place macro blocks, and make power connections.

This exercise covers the first part of the back-end design process, and subsequent exercises will cover further steps as we go along. The starting point of this exercise is a synthesized Verilog netlist which is the result of the front-end design covered in VLSI I. We will make use of what we have learned in Exercise 3, where we selected and configured I/O drivers, and in Exercise 4, where we saw how timing constraints are defined in CADENCE INNOVUS . Now, we will add I/O drivers to our design and use the developed timing constraint files. Furthermore, we will see how power routing is done, however without looking at how the parameters of the routing (such as the number of required connections and the width of the wires) are determined exactly. This will be subject of Exercise 6 and Exercise 7, while the rest of the back-end design flow will be covered in Exercise 8, where we will continue from where we stoped at the end of this exercise.

1.1 About the Style

We will use different styles to identify different types of actions.

Student Task: Parts of the text that have a gray background, like the current paragraph, indicate steps required to complete the exercise.

Actions that require you to select a specific menu will be shown in the following way:
`menu→sub-menu→sub-sub-menu`

To indicate an option or a tab in the current view/menu we will use a BUTTON.

Throughout the exercise you will be asked to enter commands using the commandline. [1] The following is an example of the linux command line.

```
sh> some_linux_command
```

Some of the commands will be entered on the command line of a specific tool:

```
enc> some_innovus_command
```

[1] Some functionality can not be accessed through GUI commands, and in some cases, using the commandline will be much faster. Most importantly, things you enter on the commandline can be converted into a script and executed repeatedly

2 Getting Started

You will need a terminal program to type in commands throughout this exercise. In the computers in the ETZ D61.2 you can open a terminal either by using the keyboard shortcut ALT + F2 (for "Run Applications") and typing "gnome-terminal", or by accessing `Activities` →`Show Applications`, scrolling down, and clicking the Terminal icon.

Student Task 1:

- Change to your home directory and install the training files with the script provided:

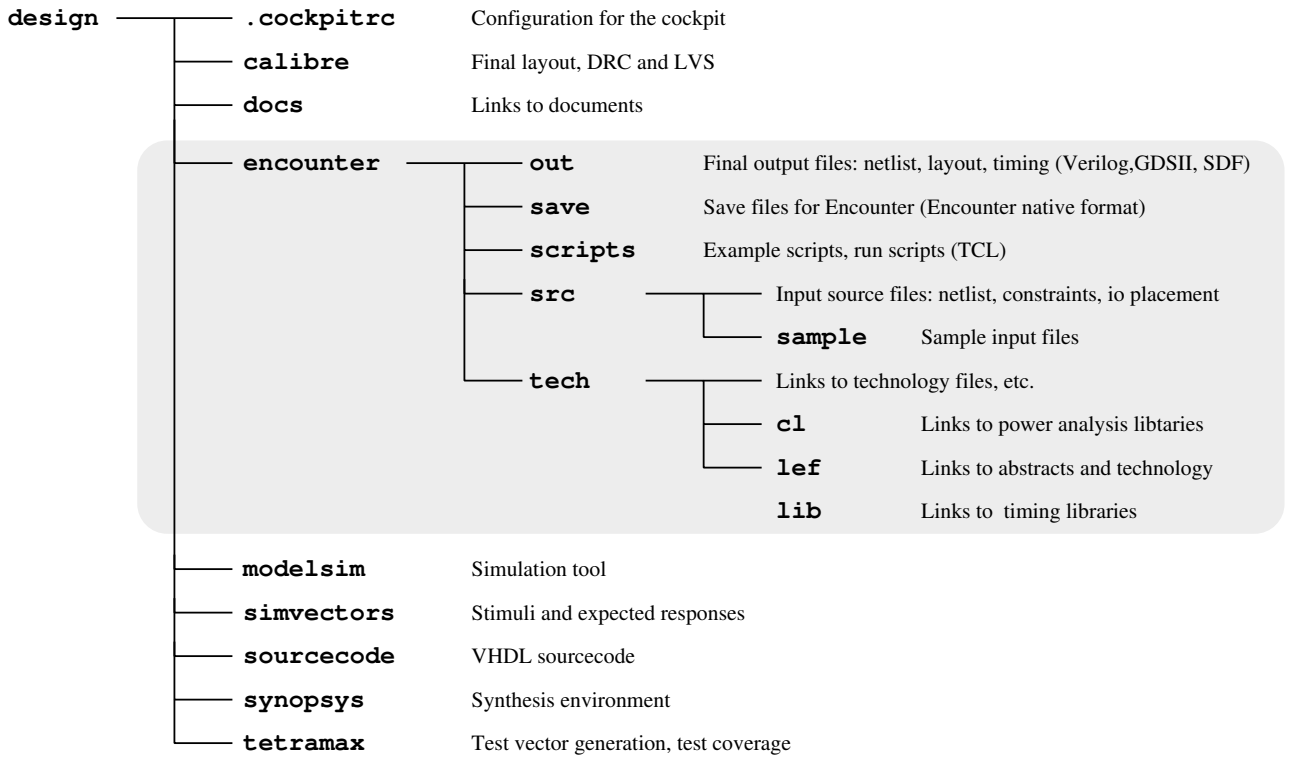
```
sh> cd ~
sh> /home/vlsi2/ex05/install_ex05
```

- Change to the design directory

```
sh> cd ex05
```

2.1 Directory Structure

The copied files and folders are organized in a particular structure created by the cockpit[2] tool developed by the Design Zentrum (DZ) of ETH Zurich. The following figure shows the detailed directory structure:



[2] See `http://eda.ee.ethz.ch/index.php#Cockpit` for more information

There are five sub directories for CADENCE INNOVUS . The tool changed its name many times during the last couple of years. Until recently it was known as CADENCE EDI ENCOUNTER, which is why the corresponding directories are still called "encounter". It is strongly recommended to use the subdirectories in the following way:

src All user input files should be placed here. These include the I/O placement file, timing constraints file and clock tree definition file (all will be explained later in section 3).

scripts Contains TCL scripts. By default several example scripts for common tasks are provided. It is highly recommended to develop a run script that contains all the commands used for your design.

save Put all CADENCE INNOVUS save files, i.e., files in native CADENCE INNOVUS format, in this directory.

tech Holds links to technology specific files. Cockpit manages this directory automatically.

out Place all final data to be exported from CADENCE INNOVUS in this directory. This includes the final netlist , layout and delay files that will be used for post-layout simulation and/or physical verification and chip finishing. A sample script that generates all these files is provided (scripts/exportall.tcl).

3 Preparing Input Files

The input files required for back-end design with CADENCE INNOVUS can be divided into three categories:

- **Source files**
  These contain our design, mainly the netlists generated by SYNOPSYS DESIGN COMPILER.
- **Constraints**
  These files tell CADENCE INNOVUS what our requirements are. Separate files hold the timing describtion, the placement of the I/O pins, and the clock tree definition (not covered in this exercise).
- **Technology information**
  There are specific files containing the information about the technology to be used by CADENCE INNOVUS in the back-end design. The information include e.g. the number of metal layers and their thicknesses, design rules (such as the minimum allowed distance between metal wires), expected parasitic capacitance and resistance of the metal layers etc. In addition, information on the standard cells is needed; e.g., the size of the standard cells, where their connections are located and the timing characteristics.

We will start with a synthesized netlist *./synopsys/netlists/filter_top.v* that contains our design[3]. The cockpit environment provides some templates for most required files under *./encounter/src/sample*. You should copy these samples one directory up to *./encounter/src* before adapting them according to your needs[4]. You will have to provide a new Verilog file that contains the I/O driver instances that will connect your chip to the environment as you have seen in Exercise 3.

[3] This is the dummy design from Exercises 3 and 4 using slightly different parameters to reduce its complexity in order to keep the run-time of the exercise acceptable.

[4] The cockpit environment is designed to keep certain files up to date. This is why the sample files are in a different directory. If there is an update, the samples under the *encounter/src/sample* directory will get updated, but the ones you have modified one directory above not be affected.

3.1 Preparing the pad frame

Microchips require specialized circuitry for electrostatic discharge (ESD) protection and strong buffers to drive large off-chip loads. I/O drivers do include all these circuits and also include bonding pads that enable physical connection between the microchip and the package. I/O drivers are basically specialized buffers, much larger than regular standard cells, and are designed to be placed around the core[5].

At this point in the design process we should already have a synthesized netlist (usually placed under *./synopsys/netlists*). This netlist describes the core circuitry and doesn't include the I/O cells. The most practical way to add I/Os is to add another level of hierarchy which instantiates both the original design and the I/O cells. We call this hierarchy the *pad frame*. In addition to the core design and the I/O drivers, the pad frame will also contain pads for power and ground supply for both the core and the I/O drivers, and special corner cells to connect the edges of the pad frame.

There are many different ways in which we can obtain a netlist describing the pad frame. In this exercise, we will explain one particular way that we think is practical.

3.1.1 Why are netlists in Verilog format?

Although we use VHDL for most of our exercises you may have noticed that the netlists generated by the tools are almost always in Verilog format. There are two main reasons for this:

- A netlist contains only structural HDL code (a hierarchy of interconnected modules) and no behavioral HDL code, and the syntax for Verilog is more compact than that of VHDL resulting in smaller file sizes.
- The Verilog language was initiated and supported by Cadence which was an early leader in placement and routing tools. All Cadence tools like CADENCE INNOVUS accept only Verilog as input. Tools made by competitors also followed this example, and practically all back-end design tools accept netlists only in Verilog format.

It is not difficult to convert a VHDL netlist into a Verilog netlist, we can use SYNOPSYS DESIGN COMPILER to read it in one format and write it out in another one for example. However, since it is not difficult to write a small Verilog file that only contains some I/O drivers, we will write the pad frame directly in Verilog in this exercise.

3.1.2 Structural Verilog code in 10 minutes

In Verilog, a design hierarchy is called a **`module`** instead of **`entity`** in VHDL. The **`module`** declaration contains the name of the module, and the input and output signal definitions (equivalent to **`port map`** in VHDL). To illustrate the similarities and differences of structural code in VHDL and Verilog consider the following two examples. Both codes describe the same circuit containing an AND gate and an inverter.

[5] An internal buffer may drive loads in the order of a few fF, whereas external loads can easily be in the pF range and above.

```
library IEEE;
use IEEE.std_logic_1164.all;

library uk65lscllmvbbl;
use uk65lscllmvbbl.VCOMPONENTS.all;

entity myNand is
 port
 ( A_DI, B_DI : in std_logic;
   P_DO, Q_DO : out std_logic
 );
end myNand;

architecture structural of myNand is
  signal T_D: std_logic;

begin

 i_and: AN2M2W port map (A => A_DI, B =>B_DI, Z=> T_D);
 i_not: INVM4W port map (A=> T_D, Z=> Q_DO);

 P_DO <= T_D;

end structural;
```

In order to instantiate a cell we need to have its component definition. The standard cell library contains a package with all component definitions. A standard cell library may contain several hundreds of cells. To avoid having to add the component definitions of all these cells into each netlist we generate, we simply include the package in the header of our code. In this example, we use the standard cell library called *uk65lscllmvbbl*. The I/O cells are defined in a separate library of their own.

Now let us look at the same netlist, written in Verilog format:

```
module myNand ( A_DI, B_DI, P_DO, Q_DO );
  input A_DI, B_DI;
  output P_DO, Q_DO;

  wire T_D;

  AN2M2W i_and ( .A(A_DI), .B(B_DI), .Z(T_D) );
  INVM4W i_not ( .A(P_DO), .Z(Q_DO) );

  assign P_DO = T_D;

endmodule
```

As mentioned earlier, the Verilog description of the same netlist is much shorter. We also notice the following:

- In Verilog no **`library`** definitions are needed. It is assumed that the correct libraries are specified externally[6].
- There is also no need to have a **`component`** description. Verilog assumes that a module that matches the instantiation will be available[7].
- Internal connections are defined using the **`wire`** statements.
- The instantiation starts with the name of the cell followed by the instance name, which is the opposite of how it is done in VHDL.
- The port mapping starts with a dot followed by the name of the port on the instantiated cell. The signal connected to the port is written in parenthesis. This format of mapping the signals to the ports of a module is called ”connection by name”. If this format is used, the ports do not have to be listed in any particular order, same as in VHDL. An alternative mapping format is called ”connection by order”, where the port name does not appear in the instantiation but just the list of connecting signals. The order in which the signal are listed has to be the same in which the corresponding ports are defined in the module declaration. ”Connection by order” is naturally more error prone, which is why we stick to port mapping using the ”connection by name” format.

3.1.3 Creating the pad frame in Verilog

We now have to create a file that contains the pad frame in Verilog. It requires the following:

- The module declaration, i.e., the module name and the declaration of the I/O ports. Since the pad frame will contain exactly the same ports as the top level of your design, they will have a very similar module declaration.
- The instantiation of the top level module of your netlist.
- The instantiation of the I/O pads.
- The instantiation of the power and ground pads
- The instantiation of the four corner cells. These cells do not contain any active logic, but they help us to route the power connections at the corners of the chip.

Student Task 2:

- Create a new Verilog file under *encounter/src/filter_chip.v* and open it for editing.

  This file will contain our pad frame and will instantiate the top level module. The netlist of the top level module is found under *synopsys/netlists/filter_top.v*.
- Open this file as well and find the declaration of the top level module. Use this information for the next steps.
- Make the module declaration for a Verilog **`module`** called *filter_chip*. Since the I/O cells that

[6] For example, if you are simulating using MENTOR GRAPHICS MODELSIM , you will have to add the libraries on the command line using -L when using Verilog. This is not needed when you simulate VHDL files as the library information can be found within the VHDL files.

[7] If you compile a VHDL file you will immediately know if there is a mismatch between the component and the instantiation. In Verilog you will have to wait until all modules are compiled.

we will use only have bi-directional connections to the outside (see Exercise 3), all ports of the pad frame have to be declared as **`inout`**.

- Create the **`wire`** declarations for the connections between the top level module and the I/O instances that we will place later on.
- Instantiate the top level module of the netlist in your file, by connecting its ports to the internal signals defined in the previous step.

Currently, the pad frame contains only the top-level module. Next, we add all the I/O drivers between the top-level module and the ports of the newly created chip-level module: The names you use for instantiation will be important in the next steps, therefore give them sensible names.

Student Task 3: Instantiate all the I/O cells. In this exercise, we will use the *IUMB* cell. For its configuration consult the documentation under *./docs* and select reasonable values for the configuration pins `PIN1, PIN2, SMT, PU, PD, SR, IDDQ`. Refer to your Exercise 3 notes if you need more information on how to determine the I/O configuration.

Note that we will use the same cell both for inputs and output connections, but we will use a different configuration in each case.

For busses you could use the **`for`** statement to save some time. Note that, if you do so, you will have to use SYNOPSYS DESIGN COMPILER to convert this file with **`for`** statement into a proper netlist that can be used in CADENCE INNOVUS . Consider that, if it is just a handfull of pads, you are faster writing all the instantiations manually.

Student Task 4: Complete the pad frame by instantiating the power pads. Instantiate two *IVDD* and two *IVSS* cells to supply power and ground to the core, as well as two *IVDDIO* and two *IVSSIO* cells to supply power and ground to the I/O drivers. These cells do not have any connections so the port map will remain empty (i.e., `IVDD pad_vdd_c1 ( );`).

Similarly, instantiate the four corner cells *ICORNER*.

You can consult the *I/O application note* in the *./docs* directory for more information on how the I/O cells are organized.

3.1.4 Adding a reset synchronizer

As explained in the lecture[8], a reset synchronizer has many advantages. However, adding such a block introduces several complications to the design flow. In this exercise, we will *not* add a reset synchronizer, but for your own designs you should consider the costs and benefits of introducing a reset synchronizer to your chip. ICs that will be only used on a tester normally do not need a reset synchronizer[9]. If you need to add a reset synchronizer, the *chip−level* file where you instantiated all the pads would be a good place to include it. At this level you only have structural code, and it would be (possible but) impractical to add behavioral code at this place. Therefore, you will need to find a suitable flip-flop and instantiate the cell directly.

[8] For more information, see *Section 6.4.3* of *Hubert Kaeslin, "Top-Down Digital VLSI Design - From Architectures to Gate-Level Circuits and FPGAs"*

[9] IC testers can do many things, but it is almost impossible to generate asynchronous signals with them. When working on a tester you will be able to determine the exact moment when the reset is applied and released, so you will not really see any benefits of a synchronizer, as the signal will be perfectly synchronized.

3.2 The I/O file

Our sources do now contain the pad frame with all necessary pads. However, we also need to provide CADENCE INNOVUS with the information about where on the die each pad should be placed. The pad placement is very important as it directly determines the PCB layout[10]. In our case, there are some additional constraints: We have an arrangement with our MPW provider[11], where the exact dimensions of the dies are fixed[12]. The standard pad frame consists of 36 I/O pads, two core supply pads, two core ground pads, two pad supply pads, two pad ground pads, as well as four corner pads. We want these pads to be located at precise locations in the chip layout, so that we can use the exact same bonding diagram regardless of the design. We also want all designs to share common power and ground pad locations so that a single test board can be used on our ASIC tester[13].

CADENCE INNOVUS allows you to precisely specify the location of the I/O pins in a separate file called *IO file*. The cockpit will copy a sample IO file (*chip.io−template* automatically to the *./encounter/src\ /sample* directory. All lines starting with '`#`' are comments. The file consists of two main sections: `globals` and `iopad`.

```
(globals
  [global definitions]
)
(iopad
   (topleft
      [pads that are on the top left]
   )
   (left
      [pads that are on the left side]
   )

   [definitions for other sides]
)
```

For us, the relevant part is the `iopad` section. This part contains eight subsections that define the names of the pad instances, and their locations in the four sides and four corners. We do not have to touch the corner specifications[14] as they will be the same for all designs. We have to distribute the pads among the four sides of the chip `top`, `right`, `bottom`, `left`. If you look at the sample file you will see that for each pad there is a single line entry in the following form:

```
   (inst name="NAME_OF_PAD"  offset=OFFSET_VALUE ) # pin no: PIN_NUMBER
```

The last part following `#` is a comment, it is there just for your information. The `PIN_NUMBER` is just a reminder to show which particular location is being defined. The location is specified using the `OFFSET_VALUE`. CADENCE INNOVUS uses a coordinate system that bases the coordinate (0,0) on the bottomleft corner as shown in the figure below:

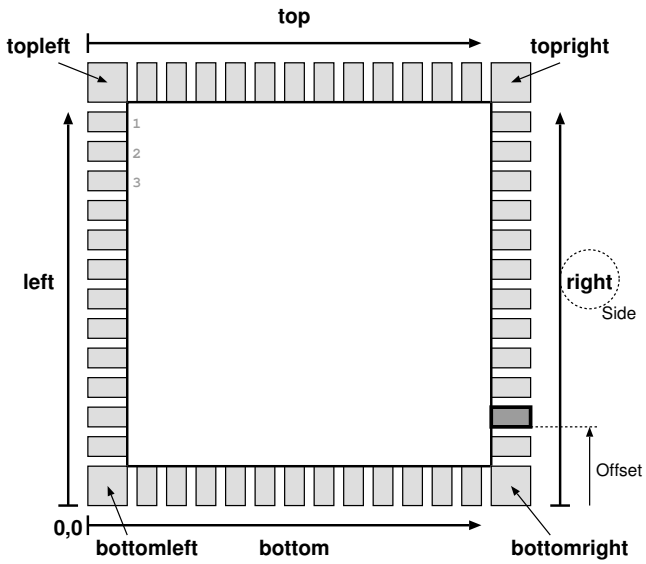
[10] A good pinout can simplify the routing on the PCB, allows you to use fewer layers and results in less parasitics.

[11] Europractice IC service located in Leuven, Belgium offers academic institutions MPW runs for several providers including UMC 65nm in Taiwan that we will be using for this exercise.

[12] Depending on a project, we can also use a custom size, however using the 'standard' simplifies things considerably.

[13] All I/O connections on a tester are connected to programmable channels that are flexible. Power and ground connections are connected on the tester board and are routed to specialized power supply channels of the tester, so these two types of pins are treated differently.

[14] `topleft`, `topright`, `bottomleft`, `bottomright`



On the `left` and `right` side the pads will be ordered from bottom-to-top, and on the `top` and `bottom` side the pads will be ordered from left-to-right. This ordering can be quite confusing, as it is neither clockwise, nor counterclockwise. Therefore the aforementioned comments showing the actual pin numbers will be very useful.

The `OFFSET_VALUE`s given in the template represent fixed locations for the given pad. It is very important that you do not change these values, as the chip-finishing part will rely on the pads being located exactly at these locations.

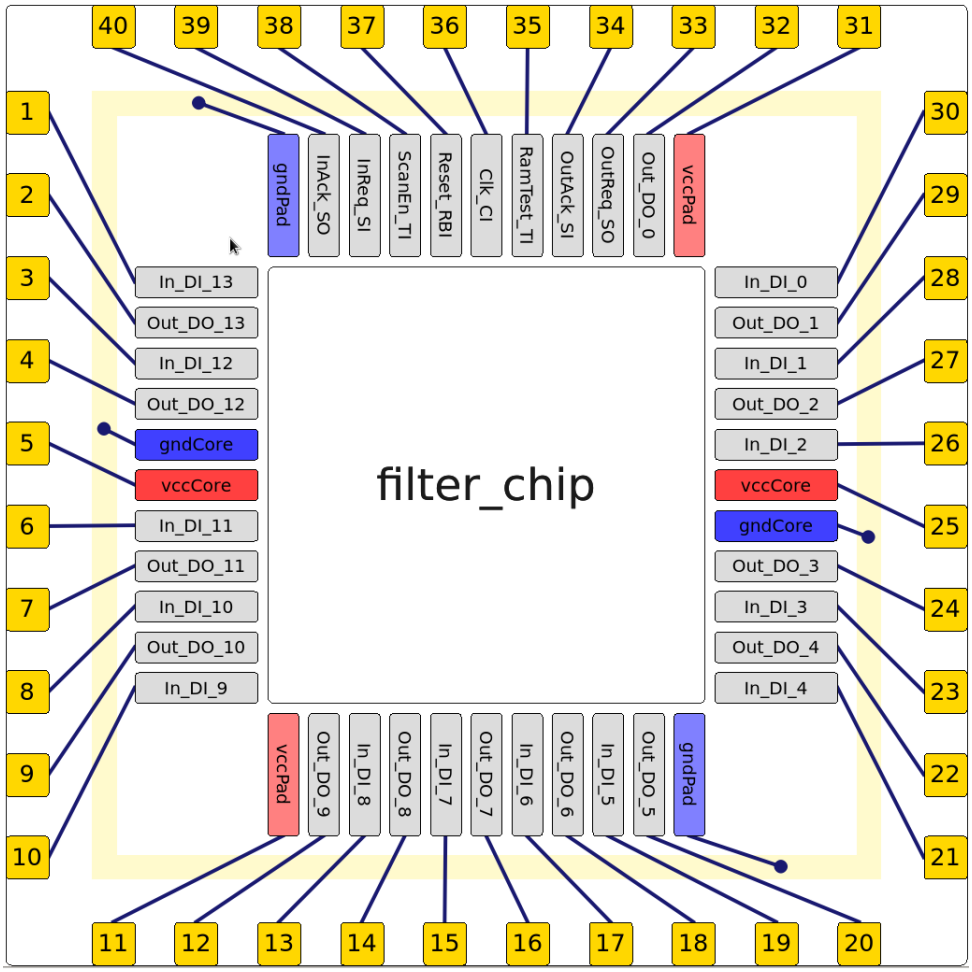
You can assign your pads by writing the name of each pad into the corresponding `NAME_OF_PAD`. The name of the pad should match the name of the instance in the Verilog file. For example, assume your clock signal is assigned to a pad named `pad_Clk_CI`. In your Verilog file you would have the following entry for this pad:

```
IUMB pad_Clk_CI ( .PAD(Clk_CI),  [other pin definitions] )
```

If you now want to place this pad on pin number 36 of your package, you will find the subsection `top` in the IO file and edit the line for pin 36:

```
...
(iopad
  ...
  (top
    ...
    (inst name="pad_Clk_CI"  offset= 570.00 ) # pin no: 36
   ...
  )
  ...
)
```

Be careful, **do not modify** the offset value while you are editing the IO file. Since we use a fixed bonding scheme for the power and ground pins, all we need to do is extract the instance names for all our signal pads and insert them into the `inst name=""` statement corresponding to the desired location. It is also recommended to put the clock pin (if possible) to pin number 36. All new test boards will make sure that the pin 36 has the best signal quality.



Student Task 5: Prepare an I/O constraint file that places the pins as seen in the figure above.

- Copy the template file *./encounter/src/sample/chip.io−template* into the *./encounter/src* directory and open it for editing.
- Place the pads in the IO file as seen in the figure above. Make sure that the name of the I/O pad instance matches the name in the Verilog file you prepared earlier.
- Complete the IO file and save it as *./encounter/src/filter_chip.io*.

Now you should be ready for starting CADENCE INNOVUS and reading in your files. If you made any mistakes in these files, you will receive error messages as you are initializing your design in CADENCE INNOVUS .

We have developed a set of customized commands that will be able to do most of the tasks described here using a GUI. To use this, you can change to the *./encounter/src* directory and execute the following (or equivalent) command:

```
sh>  ./prepare_chip.pl ../../synopsys/netlists/filter_top.v
```

This will parse the Verilog netlist, determine the I/O names and start the tool *Ordinator* in command-line mode. Ordinator will allow you to assign the pins to the desired location, by first clicking on the signal name, and then on the pin location. Once you quit the program by saving your results (click

the floppy disk icon), the required files will be automatically generated. This setup requires a lot of things to go right, and is still in active development.

3.3 Timing Constraints

Please refer to Exercise 4, if you have trouble with timing constraints.

Student Task 6:

- Copy and adapt the example MMMC constraint file(s) based on the templates under *./encounter/src/sample/mmmc**, such that the following constraints are set
  - Define a 200 MHz clock for normal operational mode
  - Specify 1.5 ns input delay for all inputs
  - Specify 1.0 ns output delay for all outputs except for `InAck_SO` and `OutReq_SO` which should have an output delay of 1.5 ns
  - Specify an input transition time of 0.4 ns at all inputs
  - Specify a 8 pF output load for all outputs

3.4 Technology Files

All technology files will be copied and maintained by cockpit and are located under *./encounter/tech* directory. You will find the following files here:

- **Technology files** (umcL65)

  lef/u65ll_8m1t0f1u.lef Base technology description, defines metal layers, vias, spacing rules, routing

  qrc* Tables used to extract parasitic capacitances and resistances for signal and power wires.

  streamout_noObs.map Layer mapping table used when exporting the final layout in GDSII format.

- **Library files** (standard cells, pads, macro-cells)

  lef/*.lef Physical description, shape and allowed orientation of cells, layer and shape of pins, blockages, antenna information, etc.

  lib/*.lib Functional description, timing and power information, maximum load/fanout or transition-time allowed, etc.

  cl/*.cl Power analysis libraries that contain information about how current flows within the cells of the library.

For the power planning later in this exercise it is relevant to know which parasitic resistances have to be expected on which metal layer.

Student Task 7: Find the resistance per square of all metal layers in the technology files. On which layer do you suggest to place the power lines?

3.4.1 Macro-cells

The macro-cells for the umcL65 process are created using dedicated memory compilers. The specific memory compiler we have access to is able to create five different types of macro-cells with various capacities:

- `S`**`H`**`KA65_` : single-port static RAM
- `S`**`J`**`KA65_` : dual-port[15] static RAM
- `S`**`Y`**`KA65_` : single-port register-file[16]
- `S`**`Z`**`KA65_` : two-port[17] register-file
- `S`**`P`**`KA65_` : via programmable ROM

The following parameters are used for the macro-cells:

- **words**
  Number of words in the memory
- **sub-word size**
  Number of bits within a sub-word of the memory. The sub-word is the smallest unit used for data access in the macro-cell.[18]
- **number of sub-words per data word**
  This parameter allows creating multiple sub-words. Each sub-word can be written to separately. For example, a 32-bit RAM can be configured as having a single 32-bit sub-word, or two 16-bit sub-words, or four 8-bit sub-words and so on.
- **column or block multiplexer**
  This parameter affects the internal structure of the macro-block, which can have a significant influence on the geometry, area, power consumption and timing of the macro. There is no general rule to determine this parameter, but once the memory requirements are known, all possible configurations can be considered to select the most suitable variant.

There are several available macro cells. Their datasheets can be found under:

```
/usr/pack/umc-65-kgf/faraday/ll/memaker/201301.1.1/datasheet.dz
```

If none of the available macro-cells suits your needs, more can be easily generated on demand. Please contact the Microelectronics Design Center for this purpose.

Our example design uses a single-port RAM named `SHKA65_8192X32X1CM16`. This RAM consists of 8192 words of 32 bit (single sub-word) and a block multiplexer of 16. All necessary preparations to work with this macro-cell have already been completed, so you do not need to do anything additional for this exercise.

[15] Dual-port memories have two completely independent access ports. At the same time, two separate memory addresses can be read or written. It's similar two independent single-port memories.

[16] Although the name suggests that the memory is made out of individual registers, it is very similar in design to SRAM.

[17] Two-port memories have two separate but dependent access ports to allow you to read and write at the same time with certain restrictions; e.g., it is not possible to write to two addresses at the same time. There are timing constraints for reads and writes to the same address, please refer to the memory compiler manual for details.

[18] In many places, sub-word is referred to as 'byte'. This might be slightly confusing, since a byte is commonly accepted to be an information unit consisting of 8 bits.

4 Importing the Design

Student Task 8:

- Start CADENCE INNOVUS either from your design directory by using cockpit

```
sh> cd ~/ex05
sh> icdesign umcL65 &
```

- or from the *encounter* directory by issuing the command

```
sh> cd ~/ex05/encounter
sh> cds_innovus-16.10.000 innovus
```

We will now import our design.

CADENCE INNOVUS uses a large configuration file that defines the design and technology files to be loaded as well as some global settings to be applied.

Cockpit does automatically generate an appropriate sample configuration file *src/sample/chip.globals*\ that should be used to start with, but needs to be modified:

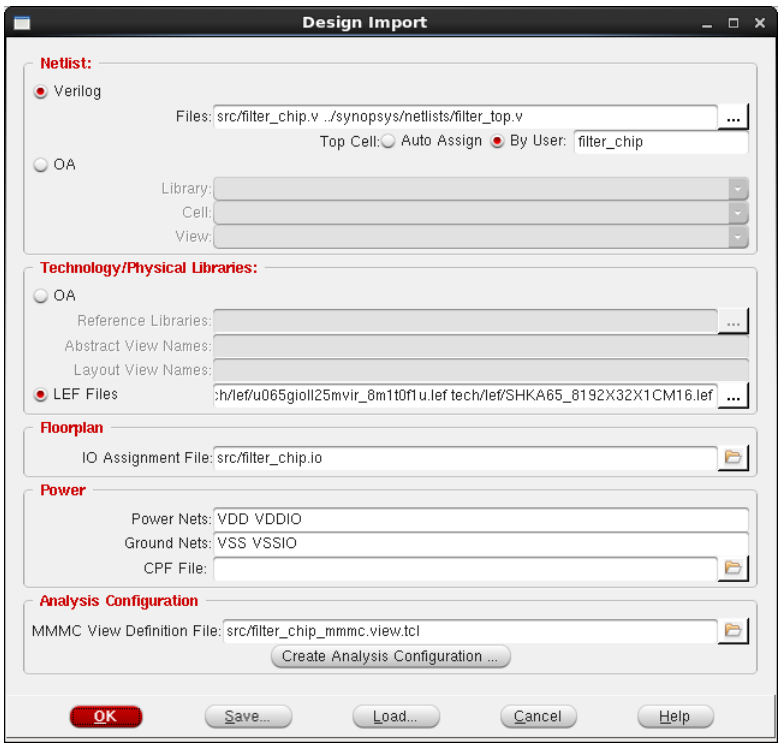
Student Task 9:

- Copy the sample file into the *src* directory

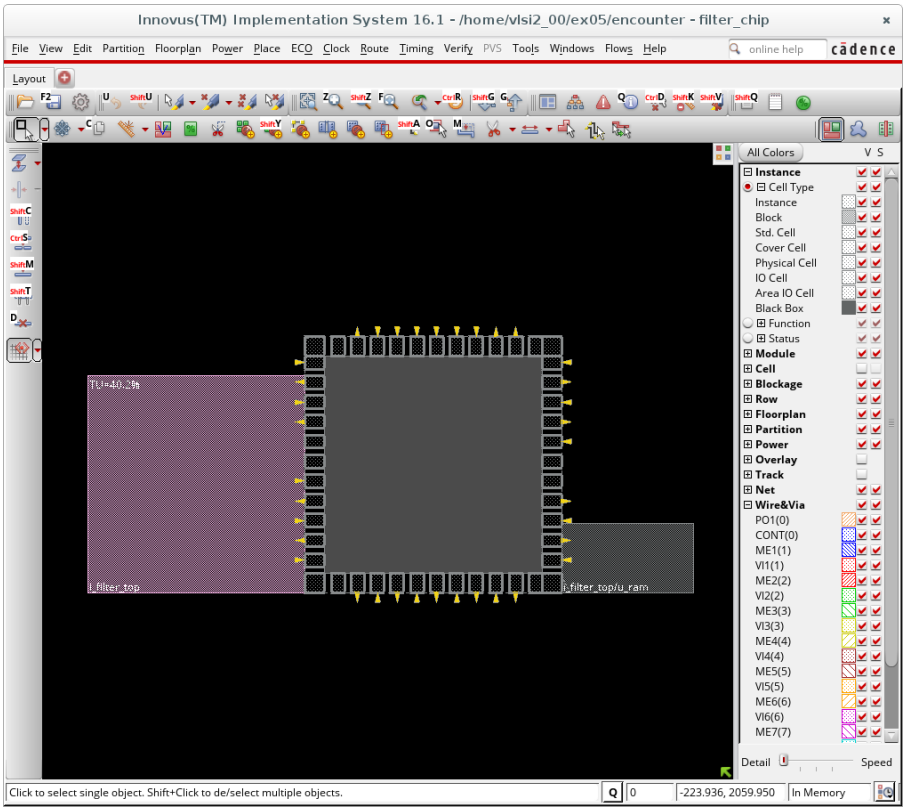
```
sh> cd ~/ex05/encounter
sh> cp src/sample/chip.globals src/filter_chip.globals
```

- Select `File→Import Design ...` to open the design import form. This form contains fields for all configuration options. At the bottom of this window, there are buttons to load and save the configuration from/to a file. Use the LOAD … button to load the configuration file we have just copied to the *src* directory.
- Make sure that NETLIST: VERILOG, IO ASSIGNMENT FILE: and MMMC VIEW DEFINITION FILE: match your design. The LEF FILES: should already be correct.
- Once you are happy with the configuration don't forget to save your changes to the configuration file.
- Click OK to import your design. Monitor the messages on the console for errors[a].
- Pay attention to the messages where the timing constraint files is loaded ("*Info: initialize multi-corner CTS") to see if everything was accepted! If there are errors, you need to fix them!

[a] You can ignore warning (EMS-42)



We are now in the floorplan view of CADENCE INNOVUS which displays an empty floorplan with only the pads placed. All top level module(s) of the netlist are shown as a pink/purple square to the left and all macro-cells to the right. Note that all standard cells are inside the module(s).



5 Floorplanning

Now we will have to decide how cells and macro-cells will be placed on our chip. This process is called floorplanning. For a standard design, our main concern would be to find a floorplan that will result in the smallest possible area, while fulfilling all performance and reliability requirements. This is purely driven by economical reasons, since chip costs are mainly determined by the area. In some cases there are additional geometrical constraints. The manufacturing company may impose certain limits to the aspect ratio of the final layout,[19] or even dictate the maximum height or width of the layout.

Back-end design is not only used for complete chips. Macro-cells that will be part of a larger system-on-chip design can also be designed in this way. In such cases there might be even more restrictions. For example, certain metal layers might be reserved for the system level.

So the question is, "*How small can my layout be so that I am still able to fulfill all specifications?*". As a lower bound, you will need enough area to place all your I/O pads and standard cells. Ideally, in terms of area you will want to place standard cells without leaving extra space in between, completely filling out the core area. This is hardly ever possible because:

- The number of interconnections that can pass through a certain area is limited by the number of metal layers available,[20] wire width and minimum spacing requirements. Depending on the interconnection overhead, the area above the cells may not be sufficient for routing.[21]
- Timing is greatly affected by the placement of your cells. Placing them next to each other with no space in between does not leave the tool any flexibility in placing cells. This in turn reduces the optimization options of the tool, like the ability to cluster cells that are closely interconnected.
- All designs require power routing for operation. Some wires of the power connection limit where the cells can be placed, or restrict signal routing which in turn increases the area requirement.
- The majority of designs require a clock tree to function. This clock tree is added during the back-end design. This requires additional area for the buffers used in the clock tree. Furthermore, the clock tree synthesis algorithm can produce better results if it has more freedom to place its buffers.
- Macro-cells, like the RAM in our example, usually require some extra space along the edges so that they can be properly integrated next to the standard cells.
- Designs that have a high switching activity require a lot of current for a short time which is called a surge. The power distribution network may need additional decoupling capacitors to store some charge that can provide some of the current of the standard cells during such a surge. Additional space for these *decoupling* cells may be required during placement.

As a consequence, the standard cell rows can not be filled completely with standard cells. There needs to remain some free space between the cells. Utilization indicates to what percentage the standard cell rows are filled. [22] Usually, it is not possible to predict whether all requirements can be fulfilled with a certain utilization. Both placement and routing are NP complete problems. Without

[19] Especially in MPW runs, a lot of silicon area is wasted if all designs have wildly different dimensions.
[20] For our technology there are 8 metal layers.
[21] Cells in our technology use only the lowest metal layer ME1 for internal connections, all other layers are free for routing.
[22] At 100% utilization all cells are abutted and there is no extra space, whereas an utilization of 50% means that half of the core area is empty.

going through a complete routing and placement, it's not possible to know. You will have to try and find out, which is the main reason why back-end design is an iterative process. [23]

5.1 Semester Projects

The MPW provider used for the semester projects offers modules called Mini ASIC (mini@sic) with a size of 1200.00 µm × 1200.00 µm. Therefore, the chip size for the semester project ASICs is fixed.

Please refer to the following web page to learn the details.

`http://www.eda.ee.ethz.ch/index.php/UmcL65#Mini.40sic`

As a consequence, we only have to make sure that our design fits on this area, and there is no need to find the smallest possible layout. We may however need to constrain the core area to make it smaller if the utilization is too low, since a spread out design has longer interconnections that may adversely affect timing.

5.2 Sketching a Floorplan

Before we go on with CADENCE INNOVUS we need to make some planning and understand some key concepts. The figure on the following page is an example floorplan (not an ideal one) that shows the important concepts.

In CADENCE INNOVUS **die area** corresponds to the total silicon area available to place pads and core cells. All pads (I/O, power and corner) are placed in the **padframe**. The remaining area can be used for the **core** of the chip.

For the semester projects, the die area is strictly limited to 1200.00 µm × 1200.00 µm, whereas the maximum for core area is 1022.40 µm × 1022.40 µm = 1.05 mm^2.

In the figure, we notice that the core area is surrounded by a **core power ring**. In its simplest form this consists of two wide metal lines (one for `VDD`, one for `VSS`) that evenly distribute the power all around the chip. [24] In order to leave room for the power ring, we need to leave a certain **I/O to core spacing**.

The **standard cells** are designed in such a way that, when placed next to each other their `VDD` and `VSS` pins can be connected with one horizontal metal line each. These power connections are relatively narrow (0.3 µm in the technology that we use) and run over the entire width of the core area to connect to the core power rings. This may create problems for designs that consume much current, since the cells towards the middle would not have a good power connection. [25] To improve this, the two uppermost metal layers are used to place a **power grid** over the core area. The vertical and horizontal **power stripes** of that grid connect down through all metal layers to the narrow standard cell power connections.

[23] Obviously, technology plays an important role, and it is possible to give certain guidelines for a technology. However, back-end design is always highly dependent on the design itself. You will usually see in a few iterations what is possible and what is not.

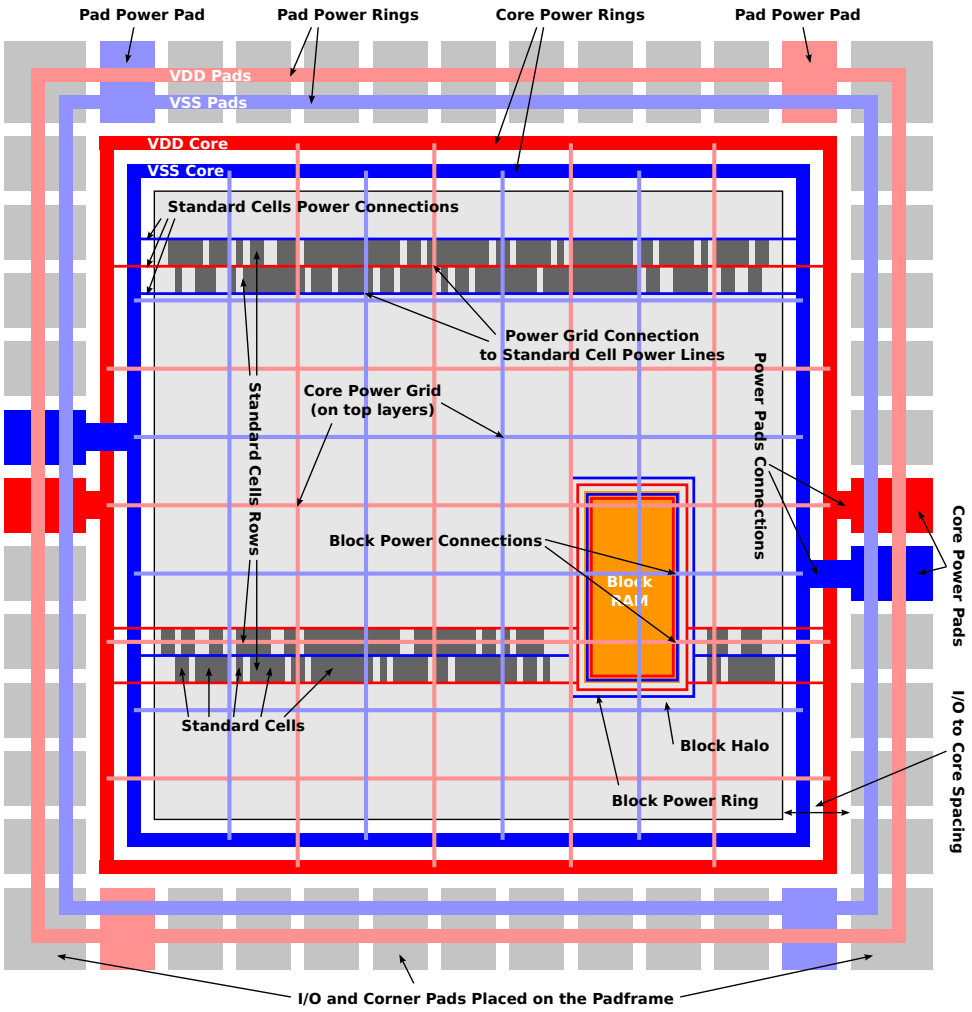
[24] The width of the metal line depends on the amount of current drawn from the line, you will be able to decide on the exact width after Exercise 6 and 7 which is dedicated to estimating the power consumption. We will use a width of 12 µm for now, since this is the widest metal that can be manufactured without slotting (wider metal lines require slots/holes which break up the metal shape).

[25] The problem is that if much current is drawn, there will be a significant IR drop along the power lines. The cells in the middle will be supplied with a lower VDD than the ones on the sides. This could dramatically affect the performance of the system.

The core area is filled with **standard cell rows** on which all standard cells will be placed. In the same area, we will also need to reserve some room for our macro-cells. Most macro-cells need some *free space* around to make signal connections, to add a **block power ring**, or simply to prevent standard cells from being placed too close to the macro. We will define a **block halo** to specify this free space.

When placing a macro-cell, you should also take into account where the power and signal pins of the block are located and what metal layer they are on. Often signal connections are only on two edges and you want them to face the core and not the I/O pads.

Now, when we consider all the above, the core area that remains for core cell placement is much smaller than the 1.05 mm^2 that we started with. However, our example design has a total cell area (including RAM) of 0.39 mm^2 and should therefore comfortably fit into the available area.



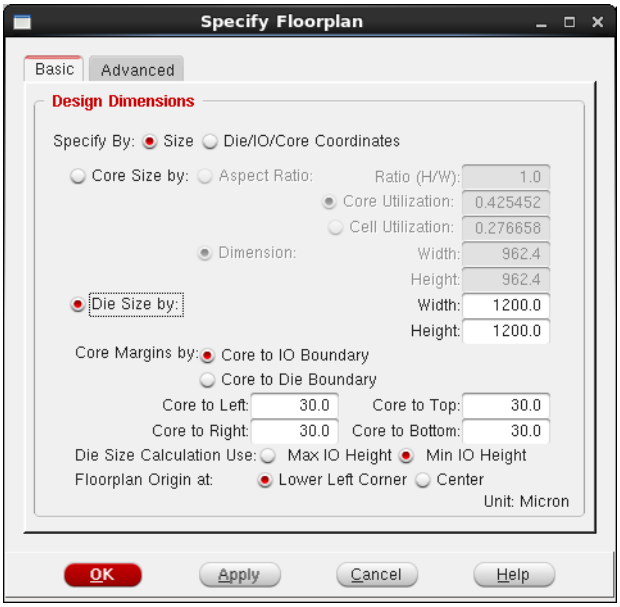
5.3 Initialize Floorplan

We are now ready to proceed with CADENCE INNOVUS .

Student Task 10:

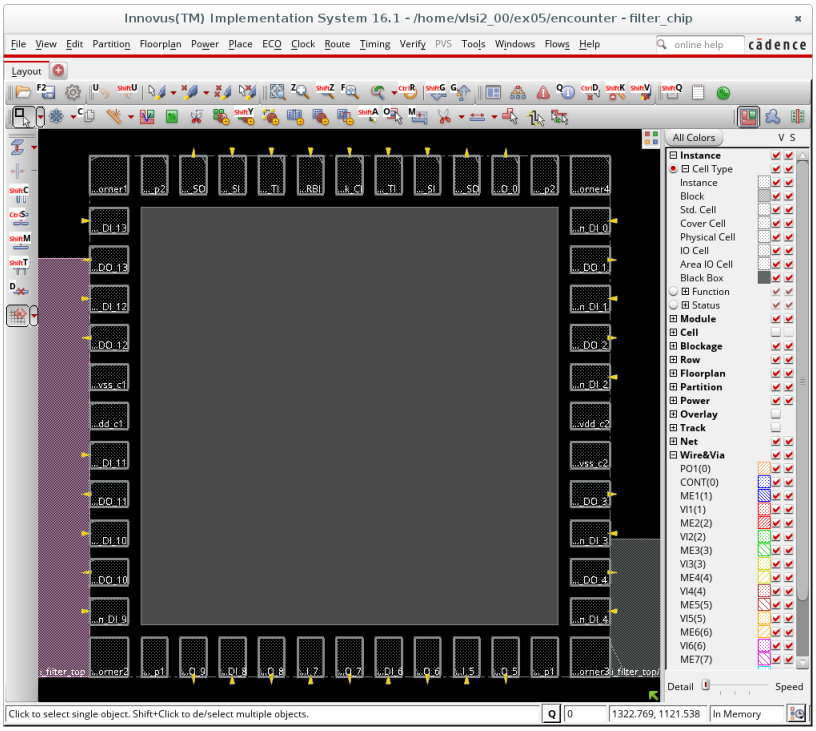
- From the menu select `Floorplan→Specify Floorplan...` A large window will open.
- Select the DIE SIZE BY: option and make sure that width and height are 1200 µm.
- Specify the core to I/O spacing by filling-in the four values under the CORE MARGINS BY: entry. Check that the CORE TO IO BOUNDARY option is selected. There must be sufficient room for the power ring around the core area.

  In this exercise, we use one VSS and one VDD line of maximum width 12 µm. There needs to be some extra space between the lines, say, 30 µm on all sides.
- Click OK to apply the floorplan specifications and to close the window.



The floorplan should now look like in the screenshot below. Note that the pads are all placed at their proper locations as specified in the IO file.

If the die size is not predefined, one can alternatively specify a desired core utilization (e.g., 80%) and let CADENCE INNOVUS calculate the corresponding die size.



The pad cells are designed to form a contiuous power supply ring around the core when placed next to each other. You can make the ring segments within the cells visible by activating "Cell→Pin Shapes" in the Layer Control of CADENCE INNOVUS . You may notice that the I/O pads are placed with some distance between them. [26] To complete the power rings, those gaps between the pad cells need to be filled.

Student Task 11:

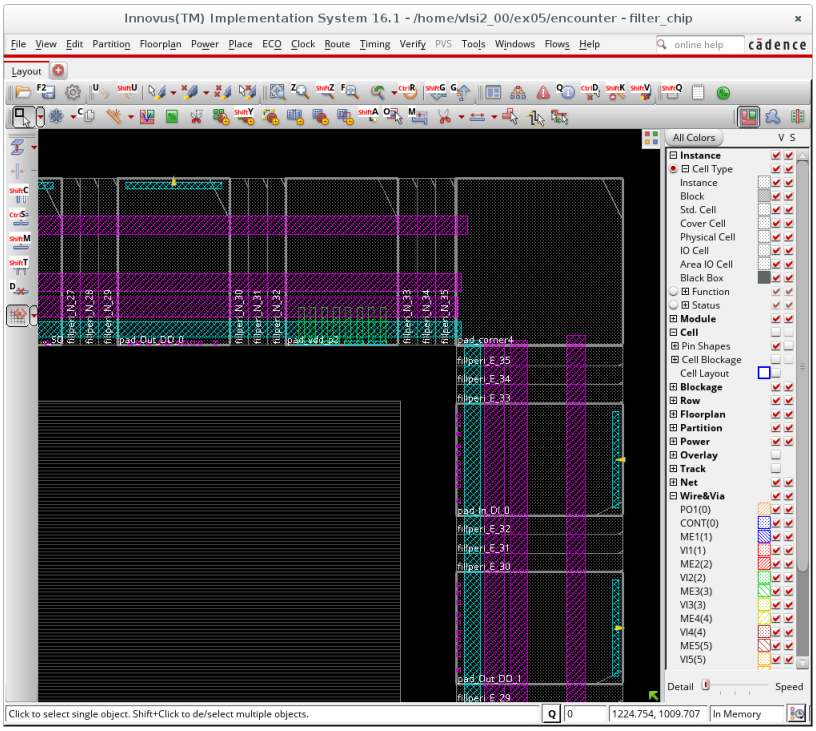
- Instead of using wires, we will place so called *filler cells* that completely fill the gaps and establish the required connectivity.

  There is a script that will automatically insert matching filler cells. Type the following in the CADENCE INNOVUS console window

```
enc>  source scripts/fillperi-insert.tcl
```

If you zoom-in a bit closer you can see that the pad supply rings are now continuously connected.

[26] This is due to the constraints set by the company that bonds the chips. The distance between two adjacent pads must be at least 90 µm, whereas the width of core-limited pad cells in this technology are only roughly 60 µm.



Next, we place the RAM macro-cell:

Student Task 12:

- Change the cursor mode to MOVE/RESIZE/RESHAPE by selecting the appropriate icon (next to the ruler icon) or use the keyboard shortcut 'SHIFT-R'. Now you can select the RAM macro-cell and drag it to any location you like. The blue lines displayed are so called flightiness that show where the signal connections to the block are.

You can change the orientation of the RAM with the drop-down menu that appears when you right-click on the RAM block (or press 'r').

Note that the RAM macro will completely block the lower four metal layers (ME1 - ME4). The remaining metal layers will be available for routing over the RAM macro-cell. [27]

Since an exact placing of the RAM block is hard to achieve using the mouse, you can use the Relative Floorplan tool for placement, which is accessible from the drop-down menu appearing upon a right-click on the RAM block. Select the RELATIVE TO OBJECT option and choose the desired core boundary in order to place the memory at the border. Click APPLY to place the RAM block.

Alternatively, you can place the RAM with the command

```
enc> relativePlace i_filter_top/u_ram CORE -orientation R0 -relation \
       T -alignedBy L -xOffset 0.0000 -yOffset 0.0000
```

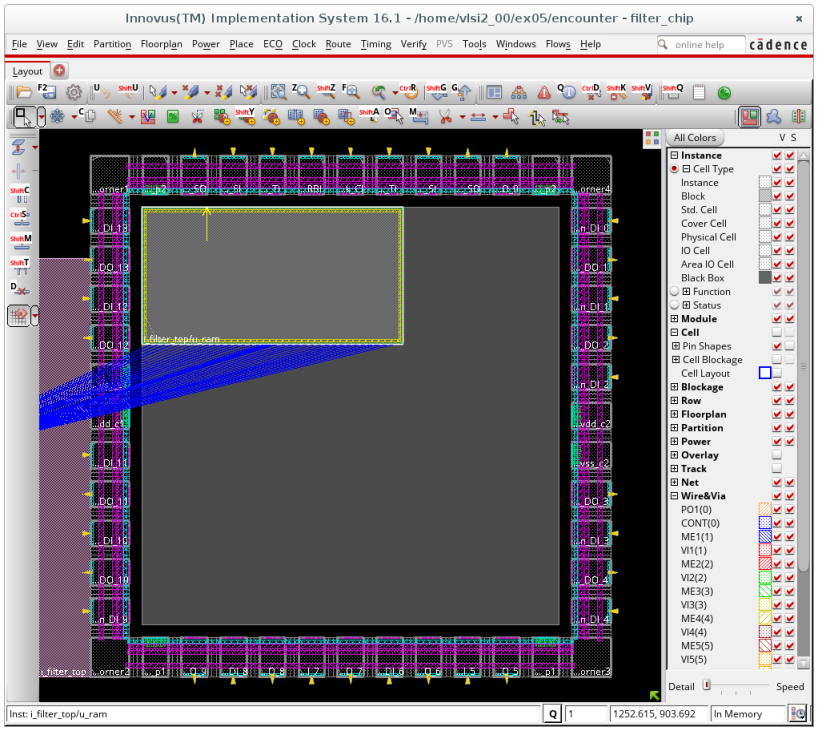
for more information how to use this command enter

```
enc> man relativePlace
```

[27] By default, the internal structures within a cell or block are not being displayed. You need to make "Cell Blockage" visible to see the so called *blockages* within a cell.

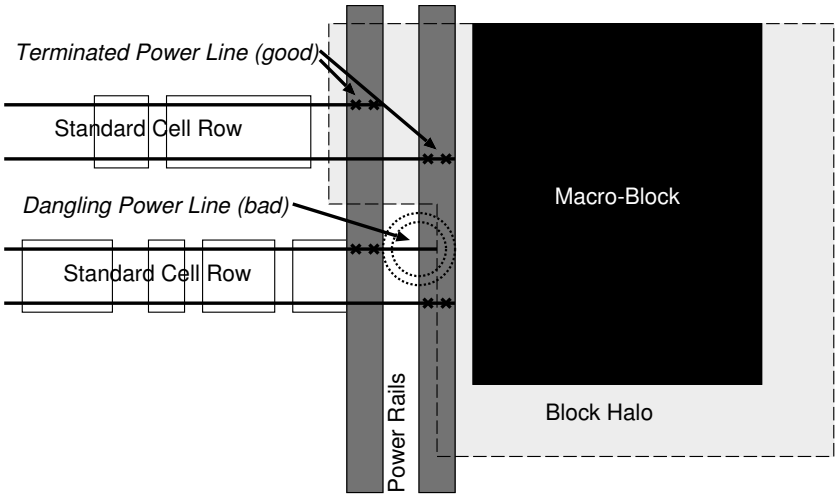
Actually, all commands issued with the graphical user interface can be also invoked with a command from the console. This allows you to write scripts to avoid the tedious clicking thought the command windows. And since the back-end design is an iterative process, scripts really help you to work more efficiently.

We suggest placing the RAM in a corner of the core area such that the connectors are facing inwards. The next screenshot shows how the floorplan will lock with such a placement. However, you are free to place the memory at any other location.



To make sure that standard cells are not placed too close to the RAM, and to avoid any problems with routing the power lines of the standard cell rows, we define a *block halo* for the RAM macro-cell. A halo is a placement blockage and/or routing blockage.

Care must be taken to define the geometry of the block halo in the correct way. The figure below illustrates a possible problem that maybe caused by a wrong definition.



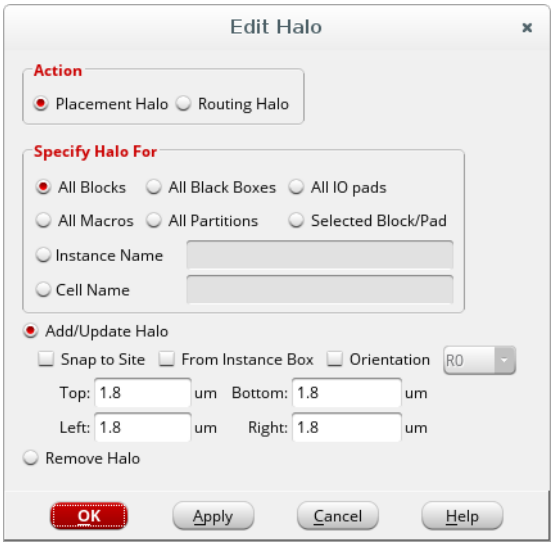
This figure shows a common situation in which standard cells are supposed to be placed next to a macro block. The supply lines of the standard cell rows need to connect to the corresponding power rails running along the side of the macro block. The block halo near the first standard cell row extends far enough to cover the two power lines, which is how it is supposed to be. In constrast, near the second row, the block halo does not cover the power rails, and when making the power connections CADENCE INNOVUS will try to extend the power connection past the power rails as shown in the figure. This leaves a dangling power line known as *geometry antenna*, which will not render your chip useless, but should be avoided.[28]

Student Task 13:

- From the menu select `Floorplan→Edit Floorplan→Edit Halo...`. A window will appear, where you can specify a keep-out zone for routing and/or placement around the macro-cell.

  Usually, we only need a *Placement Halo*. The size will depend on your power routing/floorplan.

- Add a placement halo of 1.8 µ around the RAM block. You can specify the halo for all blocks since we only have one RAM block.



At any point if you wish to delete part of the floorplan you can:

- use the UNDO feature by simply pressing '`u`'
- select and remove objects of a specific class (press '`d`')
- use the menu option `Floorplan →Clear Floorplan...`
- select an object and hit the '`Del`' key on the keyboard

[28] Please note that it is not possible to draw a block halo that has this L-shape (we show this just for the illustration of the problem), but defining the block halo with a wrong dimension will result in the exact same problem.

Student Task 14:

- Also, you can save or load (restore) your floorplan at any time using the menu `File` →`Save` →`Floorplan ...` and `File` →`Load` →`Floorplan ...`, respectively.
- Save your floorplan to the *save* directory.

5.4 Power Planning

The next step is to create the power distribution network.

The Verilog netlist that we started with does not contain any power connections, therefore we need to create this connectivity now. We have to connect the power/ground pins of all instances to the respective global power/ground net that was specified on the DESIGN IMPORT form (category POWER)

The example design (and probably your semester project design) has only one power domain where all the core instances are connected to the same power and ground nets. However, it is possible to have different power domains in a chip (e.g., if certain parts of an integrated system need to be completely shut off dynamically to save power). CADENCE INNOVUS can handle different power domains and the signal crossing from one domain to another, if they are properly specified in what is known as the *Power Intent*. The power intent is written either in the *Unified Power Format* (UPF) or in the *Common Power Format* (CPF).[29]

Student Task 15: Prepare the power intent file that defines the power domains and their connections.

- Copy the template file *./encounter/src/sample/chip.cpf* into the *./encounter/src* directory and open it for editing.
- Have a look at the file. The template file defines a single power domain for the chip. The I/O drivers have actually a higher supply voltage as the core instances, but since they have the function of level shifters they are at the boundary to another power domain which is external to the chip. Thus, the definition of a single domain with two supplies and ground net pairs works in this case.
- Since the definition works for our example chip, all you have to do is adapt the design name and save it as *./encounter/src/filter_chip.cpf*.
- Execute the following two commands to load the power intent into CADENCE INNOVUS

```
enc> read_power_intent -cpf src/filter_chip.cpf
```

```
enc> commit_power_intent
```

Next, we will add the core power rings that distribute power all around the core.

In case something went wrong with the Power Planning you can remove all power lines with

```
enc> deleteAllPowerPreroutes
```

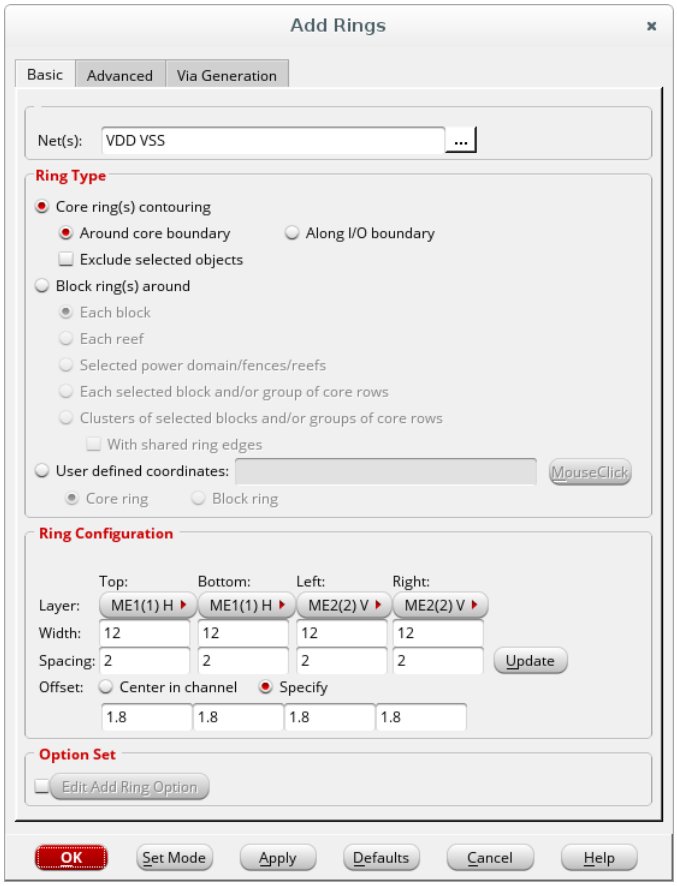
[29] There was a long coexistance of these two formats, until recently when the UPF became an IEEE standard. In this exercise we use CPF, for the time being.

Student Task 16:

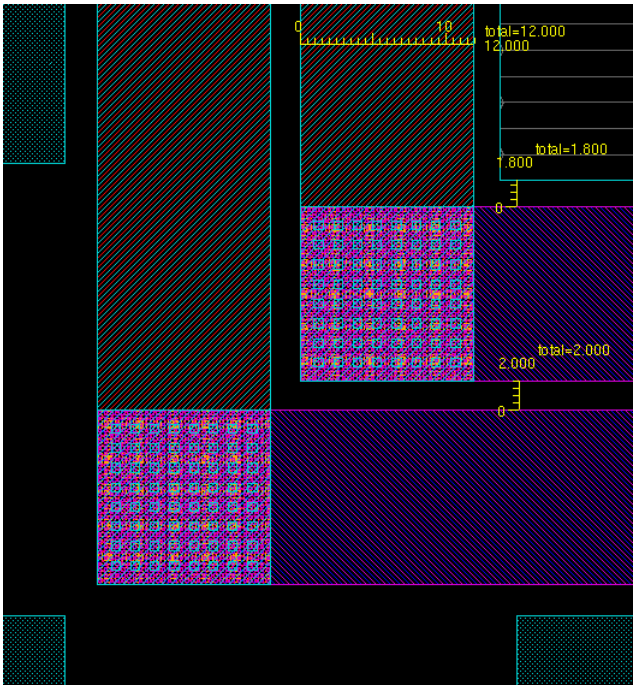
- Select the menu `Power` →`Power Planning` →`Add Ring...`. A large window appears.
- The NET(S) field on the top defines for which nets rings will be created. To create both power rings for power `VDD` and ground `VSS` select the two nets.
- The RING TYPE section specifies which type of rings are to be added. Check that the CORE RING(S) CONTOURING: AROUND CORE BOUNDARY option is selected, since we want to add the rings around the core.
- In the RING CONFIGURATION section you can specify on what layers the ring segments will be created. Select `ME1 H` for TOP and BOTTOM and `ME2 V` for LEFT and RIGHT. Specify WIDTH as 12 µm, SPACING as 2 µm and OFFSET as 1.8 µm.
- Click APPLY to execute the command.

  Now, we have created power rings on the two lower-most metal layers.
- Examine the pins of a `VDD` pad and find out on which layers they are located.

  You will see that they are present on all eight layers. For better power connectivity we place power rings on all layers:
- Add power rings on all the other metal layers by changing the layers in the RING CONFIGURATION. Make sure that the top and bottom part of the ring are placed on the odd metal layers and the left and the right part on the even metal layers.



SPACING determines the distance between the two nets and OFFSET determines the distance between the core area and the innermost ring.



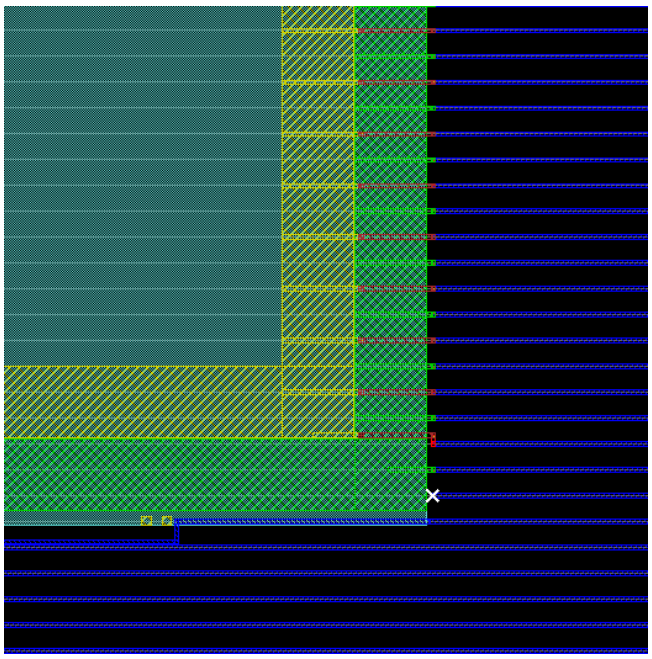
There are many alternative power distribution schemes that can be used. The one that we have chosen here is a very simple one. We have placed a power ring on all metal layers around the core. Note that there is still space to route signal from the core through the power ring to the I/O pads, since the horizontal lines of the rings are placed on the odd layers and the vertical lines are placed on the even layers.

For your own designs, you should perform a power analysis (topic of Exercise 6) to find out the *best* power distribution solution that matches your design.

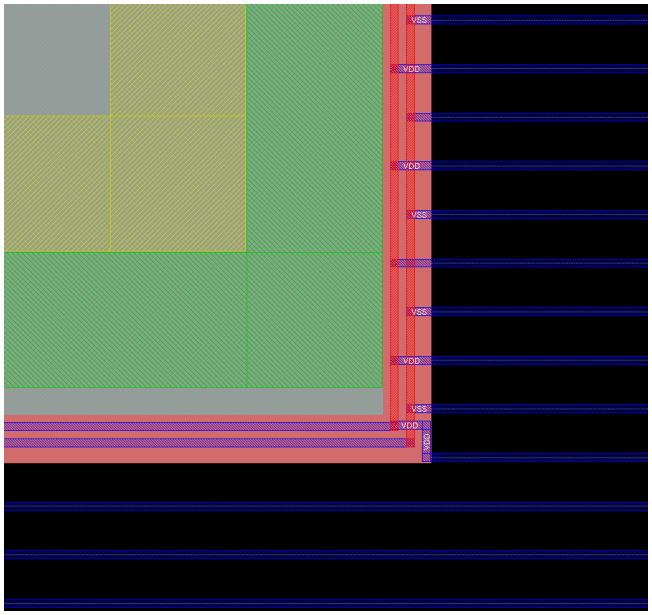
The width of 12 µm was chosen for convenience. In principle, the wider the power connections the better. However, in this technology, metal lines wider than 12 µm need to be slotted ('stress relief slots') which requires extra effort. As an alternative to slotting it is also possible to create several smaller parallel rings, e.g. two `VDD` and two `VSS` rings.

We also need a power ring around the RAM macro-cell. This ring is not to supply the RAM block with power but to provide a save ending point for the routing of the standard cell power connections.

The routing of the standard cell power connections is done in a later step. If you do not place a power ring around the RAM block, the routing will fail and create unwanted dangling wires (see figure below).



The power ring together with the block halo results in a much better routing free of dangling wires:



Add the power rings around the RAM block:

Student Task 17:

- Select the menu `Power→Power Planning →Add Ring...` just like before.
- In the RING TYPE box, select BLOCK RING(S) AROUND. You can keep EACH BLOCK selected since we have only one block anyway.
- CADENCE INNOVUS should create wires only on the block edges where there are no power lines yet, i.e. it will not create new wires on top of existing core rings and block rings.

However, under certain conditions this fails, in which case you can specify the ring sides and extension directions explicitely in the ADVANCED tab.

- Select `ME1 H` for TOP and BOTTOM and `ME2 V` for LEFT and RIGHT. Specify WIDTH, SPACING and OFFSET as 0.3 µm.
- Click OK to close the window and to execute the command.

In the next step we connect:

- The core rings to the core supply pads (`IVDD` and `IVSS`).
- All standard cells to the `VDD` and `VSS` lines.

Since the following routing steps cannot simply be undone without deleting all power lines, save your floorplan.

Student Task 18:

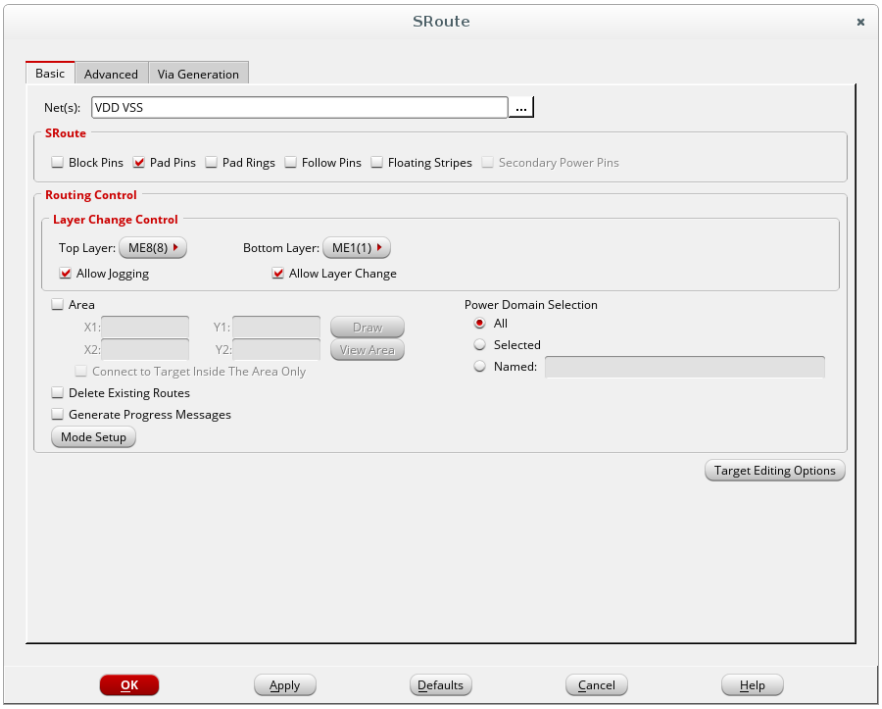
- Save your floorplan to the *save* directory.

Student Task 19:

- Select `Route →Special Route ...` from the menu. *SRoute* is the special net router, and is only used to make power connections.
- First, specify the NET(S) to route. These are `VDD` and `VSS`.
- Next, select the different connection types to establish.
  1. BLOCK PINS are macro-cell power connections. You can uncheck those since we will create a power grid later to connect our RAM block.
  2. PAD PINS are the connections from the core supply pads to the core ring, which is what we want to do in this step.
  3. We will not need the PAD RINGS option since we have already used filler cells to complete the pad power rings.
  4. FOLLOW PINS option will add power lines to the standard cell rows. We also want to generate those.
  5. If you have power stripes in your design that are not connected to the rails yet (not very likely) you can use the FLOATING STRIPES option.
- While it is possible to route all connections at the same time, it is strongly recommended to do it one by one:
  1. Start with PAD PINS. To make sure that all power pins on all layers of the power pads are being connected, go to the ADVANCED tab, select PAD PINS, and activate the option NUMBER OF CONNECTIONS TO MULTIPLE GEOMETRIES: ON THE PREFERRED ROUTING DIRECTION.
  2. Route the FOLLOW PINS. This should create many horizontal ME1 lines that connect to the rings.
- Check the power routing:

1. Check that the power pins of the power pads are connected to the core power ring on multiple layers.
2. Check that the boundary of the block RAM for dangling wires and other routing problems. Compare your routing with the screenshot shown before.
3. Check that the standard cell power lines connect to the core power ring.

- Sketch the metal-layer cross-section of the right inner core power ring (seen from the side) and label the nets. Show it to an assistant. Also explain how the core ring is connected to the power pads.



At this point the power to the standard cells arrives only from the sides, which can be a problem in fast designs as the standard cells in the middle of the standard cell row might not receive sufficient power. Furthermore, the block RAM is not yet connected to the power supply.

In the following, we will place a dense power grid over the entire core area. We have selected the upper metal layers `ME7` and `ME8` for the grid, because in this technology `ME8` is thicker and consequently has less parasitic resistance which is desirable for power distribution.

Student Task 20:

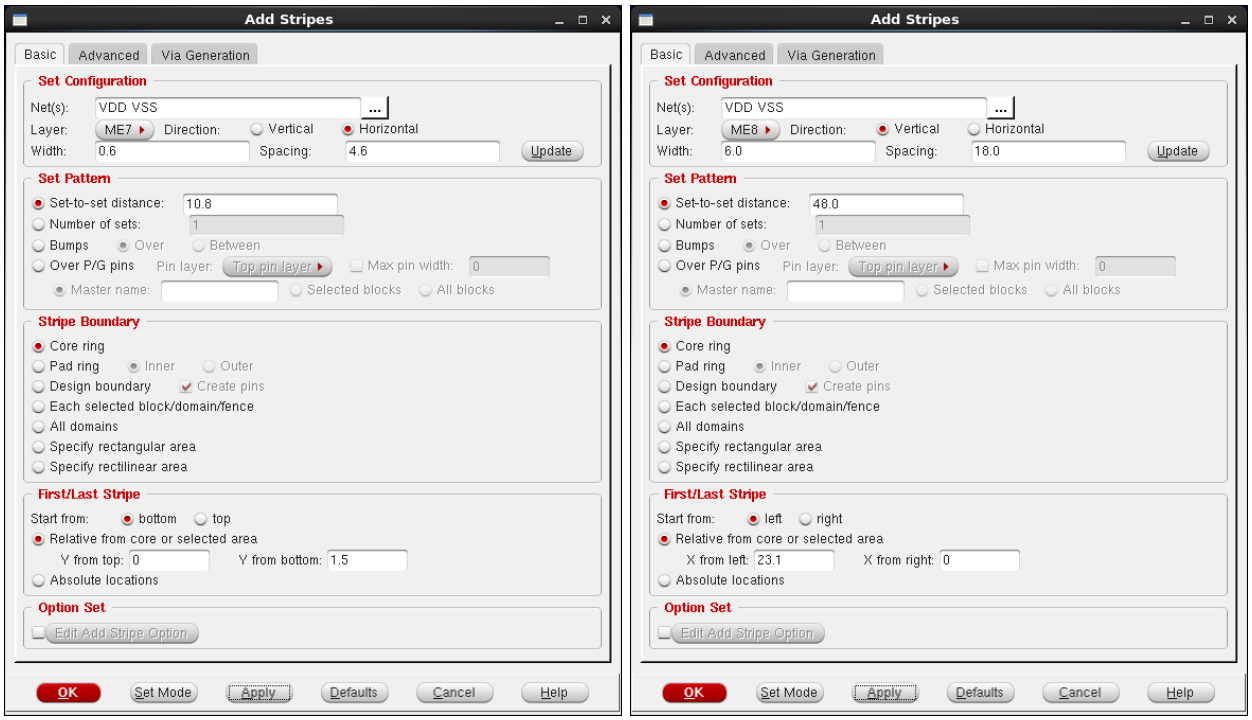
- Select `Power` →`Power Planning` →`Add Stripe ...`.

  The SET CONFIGURATION part of the window defines the properties of one stripe set.

  The SET PATTERN part defines how many stripes will be added. We can either choose to insert a fixed number of sets or only specify the distance between two sets SET-TO-SET DISTANCE:.

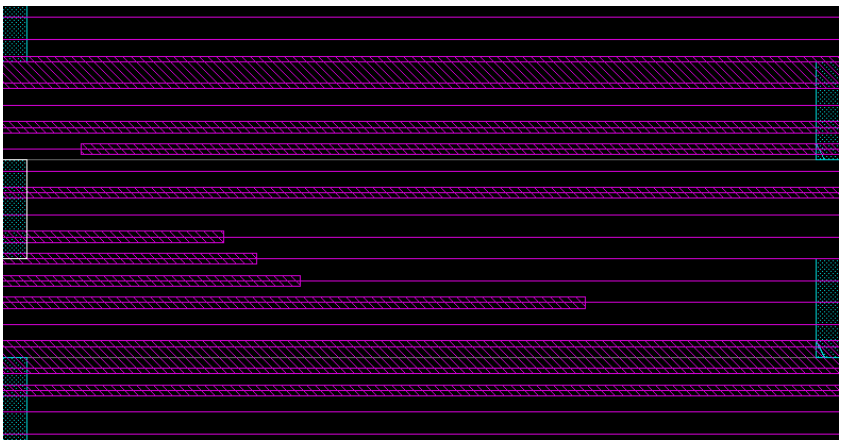
- Specify the NET(S) of the stripes, i.e., `VDD` and `VSS`, first.

- Add fine horizontal stripes on metal layer 7. See the left screenshots below for the exact settings.
- Add coarse vertical stripes on metal layer 8. See the right screenshots below for the exact settings.
- Check that the power grid is properly connected:
  1. Check that the vertical stripes connect to the standard cell power lines and to the core power ring.
  2. Check that the horizontal stripes connect to the vertical stripes and the core power ring.
  3. Check that the power stripes connect to the power pads of the block RAM.



The distances and widths for the stripe placement were chosen such that the stripes are aligned on the signal routing grid. [30] This minimizes interference of the stripes with the signal routing and results in an area-efficient routing:

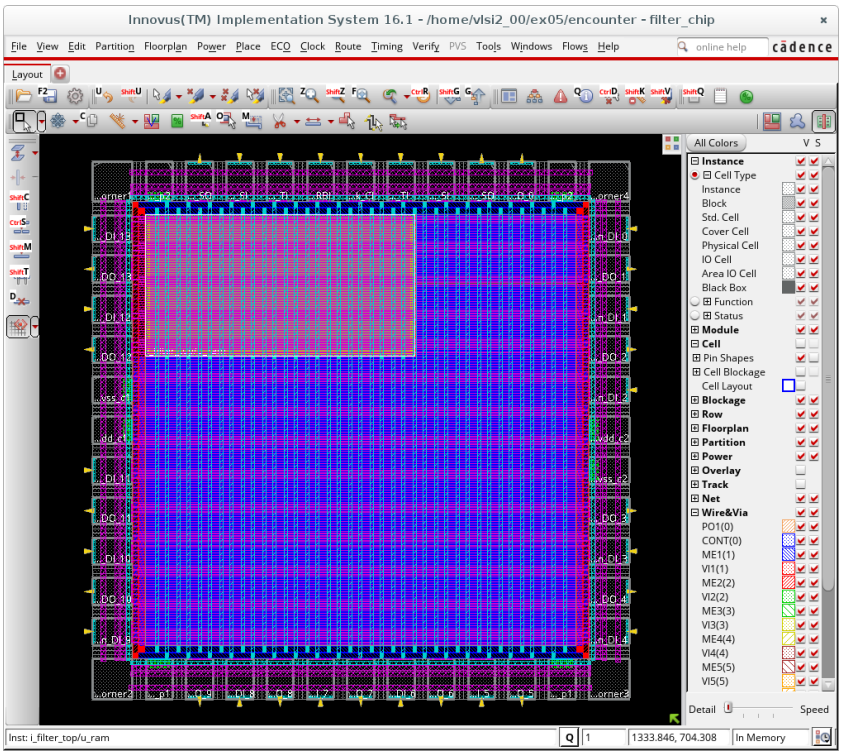
[30] Activate “Track→Pref Track” in the Layer Control to see the routing grid.



There is a script *scripts/power_grid.tcl* that performs the entire power planning done in this exercise. The script also calculates the optimal distances for the stripe placement. Have a look at the script. You should be able to match the commands to the steps you have done with the GUI. The script can be run with:

```
enc> source scripts/power_grid.tcl
```

We are now finished with floorplanning. Your floorplan should look similar to the following screenshot.



6 Next steps

Before we proceed any further, we should save the design. Go to `File→Save Design`. It is important to change the DATA TYPE to INNOVUS. This will change the appearance of the rest of the dialog box. Now, click on the icon on the far right hand side of the FILE NAME, and save your design under the *./encounter/save* directory.

Note: Actually up until this point, we did not do any operation that takes up significant computation time, and it would be just as easy to run a simple script that executes all the commands up until this point. Using such a script has other advantages. You can see exactly what you have done, and if necessary you can make simple additions or corrections easily. Get used to working with scripts as soon as possible.

6.1 What next?

We will continue the placement and routing flow in Exercise 8. The next two exercises will deal with power consumption and distribution. You will see that the parameters of the power routing strongly depend on how much current flows through the circuit. Once you have covered these exercises, you will be able to make more informed decisions on how to plan and dimension the power routing.