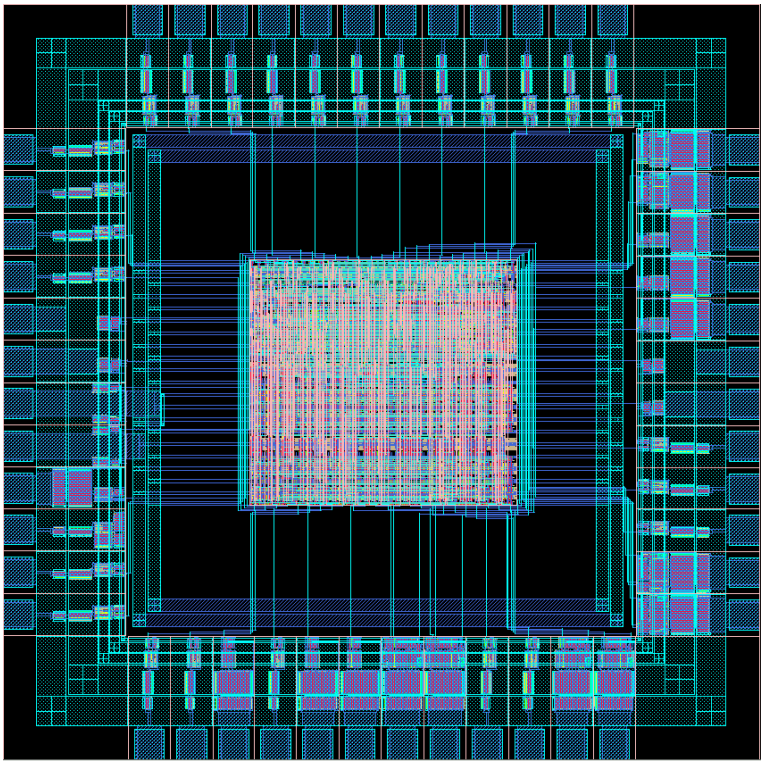
0pt0.6pt

ALLIANCE TUTORIAL

Pierre & Marie Curie University
2001 - 2004

PART 3
place and route

Frederic AK Kai-shing LAM
Modified by LJ



Contents

PART 3 : Place and route

All the files used in this part are located under `/tutorial/place_and_route/src` directory.
This directory contents three subdirectories and one Makefile :

- Makefile
- inv
  - Makefile
  - inv.vbe : behavioral description
  - inv_x1.ap : inverter cell design using GRAAL
- buffer
  - Makefile
  - buffer.vbe : behavioral description
  - buf_x2.ap : buffer cell design using GRAAL
- amd2901
  - Makefile
  - amd2901_ctl.vbe : behavioral description of control part
  - amd2901_dpt.vbe : behavioral description of data-path
  - amd2901_ctl.c : file .c of control part
  - amd2901_dpt.c : file .c of data-path
  - amd2901_core.c : file .c of heart
  - amd2901_chip.c : file .c of the circuit with their pads
  - pattern.pat : tests file

1 Introduction

The goal of this tutorial is to present some ALLIANCE tools :

- **GRAAL** Graphic layout editor ;
- **DRUC** Design rule checker ;
- **COUGAR** Symbolic layout extractor ;
- **PROOF** Formal proof between two behavioral descriptions ;
- **OCP, OCR, NERO, RING** place and route tools .

The beginning of this tutorial will relate to the drawing under **GRAAL** of a inverter cell and a buffer. The predefined cells concepts, model and hierarchy will be introduced .
Then this tutorial contain the methodology used in Alliance to produce the amd2901 physical layout that you conceived in Alliance Tutorial PART 2 "Synthesis" (All the documents used will be provided to you).

2 Inverter and buffer drawing under GRAAL

2.1 Introduction

The library can be enriched by new cells with **GRAAL** editor .
GRAAL is an editor of symbolic *layout* integrating the drawing rules checker **DRUC** and also a net extractor. The first part here aims to draw an inverter cell inv_x1 in the shape of a predefined cell of sxlib complyiant with provided drawing rules.

2.1.1 Technological environment

Some tools of Alliance use a particular technological environment. It is indicated by the environment variable **RDS_TECHNO_NAME** which must be set to **/alliance/etc/cmos.rds**

2.1.2 GRAAL

The *layout* editor handles six different objects types which we can create with the menu **CREATE** :

- The ”instance” (physical cells importation)
- The abutment boxes which define the cell limits
- Segments: DiffN, DiffP, Poly, Alu1, Alu2... CAluX is used to specify a possible rectangle area for the connectors.
- VIAs or contacts: ContDiffN, ContDiffP, ContPoly and ViaMetal1/Metal2.
- Big VIAs
- Transistors: NMOS or PMOS

GRAAL uses the environment variable **GRAAL_TECHNO_NAME**. It must be set to **/alliance/etc/cmos.graal**.
Steps to follow to create a sxlib cell by respecting the sxlib gauge : (cf 2.4 Sxlib gauge)

- place the supply Vdd and Vss using the menu CREATE->Segment
- place the VIAs using the menu CREATE->VIA
- place the transistors PMOS and NMOS using the menu CREATE->Transistor
- place the NWell body using the menu CREATE->Segment
- place the input/output connectors using the menu CREATE->VIA
- link the transistor P and the transistor N with the Poly segment using the menu CREATE->Segment
- supply each transistor by linking them with Ndiff and Pdiff segments and VIAs contacts
- define the cell limit with an abutment box using the menu CREATE->Abutment Box

2.1.3 COUGAR

The tool **COUGAR** is able to extract the *netlist* from a circuit to the format **al** or **spi** given a layout description with the format **ap** . To extract a netlist at transistor level, use the following command :

```
> cougar -t file1 file2
```

COUGAR uses the environment variables **MBK_IN_PH** and **MBK_OUT_LO** according to the input and output formats. For example to generate a SPICE netlist (with the format **.spi**) starting from a

layout description **.ap** it is necessary to set the following environment variables:

```
> MBK_IN_PH = ap
> export MBK_IN_PH
> MBK_OUT_LO = spi
> export MBK_OUT_LO
```

```
> cougar -t circuit circuit
```

The resulting spice netlist can be then simulated using a SPICE simulator and a given model card for a dedicated technology.

The schematic of the transistor neltlist can also be displayed using **XSCH** :

```
> xsch -I spi -l circuit
```

2.2 inverter Diagram

The theoretical inverter diagram is presented at the following figure:

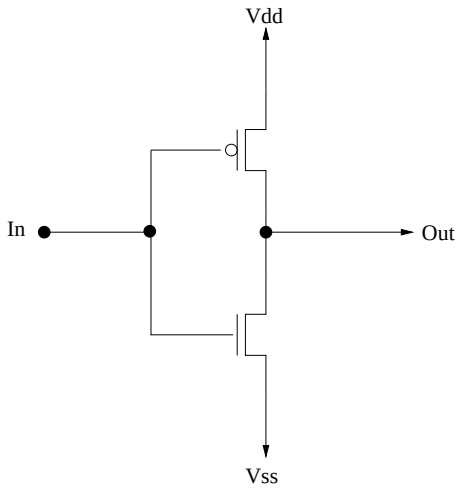


Figure 1: transistors diagram of a C-MOS inverter

2.3 Buffer diagram

The theoretical buffer diagram is presented at the following figure:

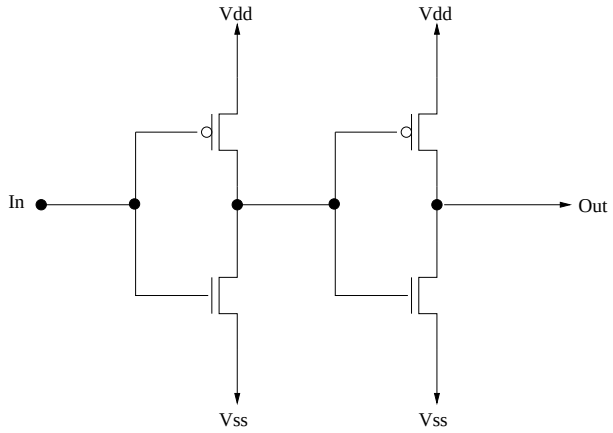


Figure 2: transistors diagram of a C-mos buffer

It uses two inverter according to the hierarchy:

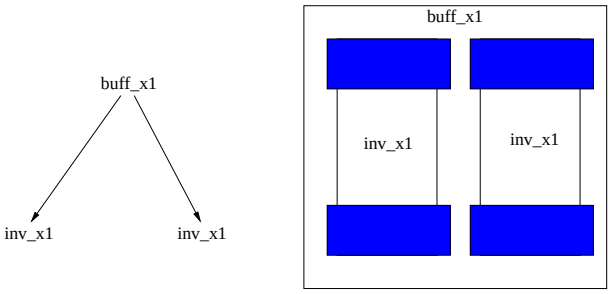


Figure 3: C-mos buffer hierarchy

‘

2.4 sxlib gauge

- The sxlib cells have whole 50 lambdas height and a multiple of 5 lambdas width.
- The supply Vdd and Vss are carried out in Calu1; they have 6 lambdas width and are horizontally placed in top and bottom of the cell.
- The transistors P are placed close to the Vdd while transistors N are placed close to the Vss.
- Box N must have 24 lambdas height .
- The special segments CAluX (CAlu1, Calu2, CAlu3...) form the cell interface (PORT_MAP) and play the role of ”flat” connectors. They must be placed on a 5x5 grid and can be anywhere in the cell.
- The special segments TAlux (TAlu1, TAlu2...) are used to indicate the obstacles for the router. When you want to protect AluX segment, it is necessary to cover them or surround them by corresponding TAlux (same layer). TAluX are placed on a grid with 5 lambdas steps (figure 5).
- The minimal width of CAlu1 is 2 lambda, plus 1 lambda for the extension (figure 6).
- The boxes N and P must be polarized. **It should be respectively connected to Vdd and Vss** .

You will find a summary of these constraints on the diagram 4:

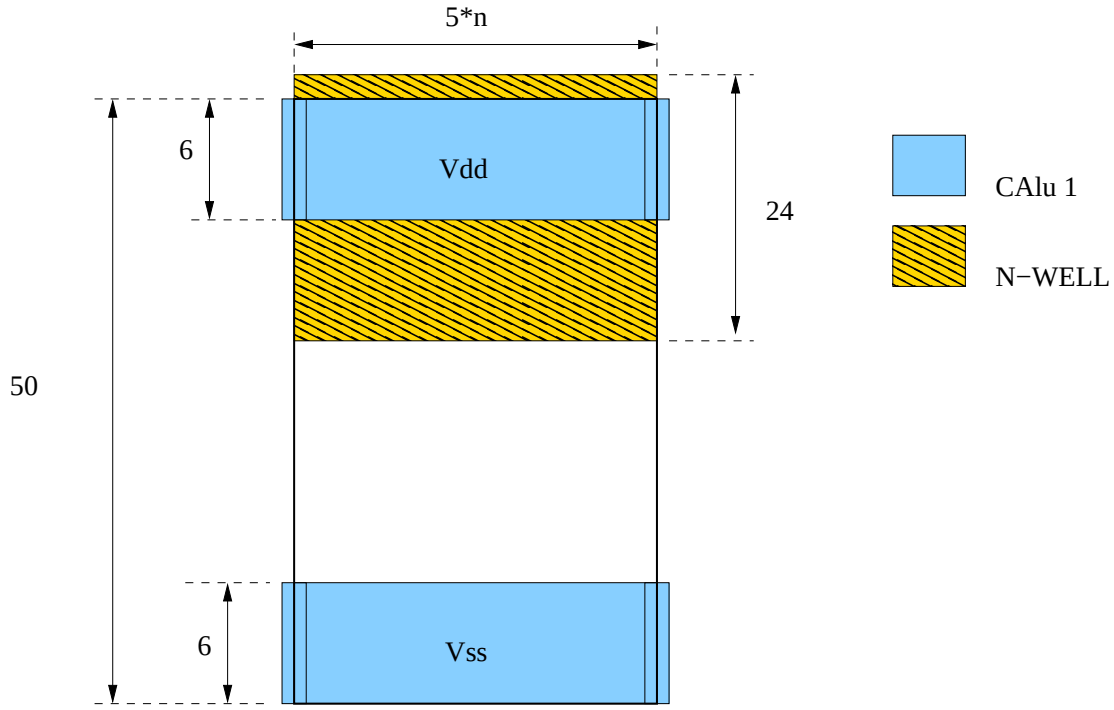


Figure 4: a cell model of the sxlib library

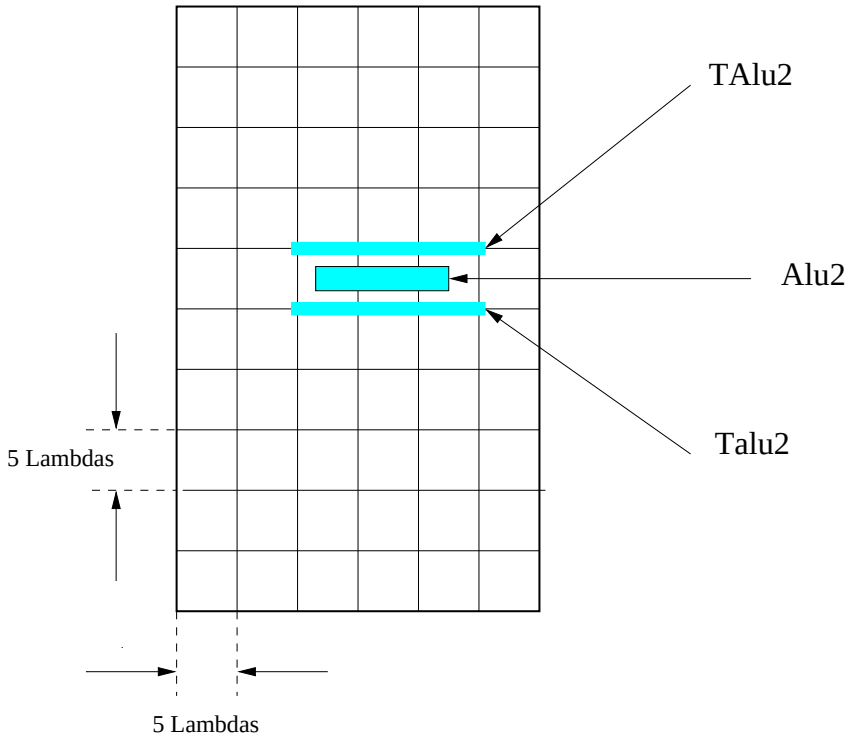


Figure 5: Use the layer TAluX like protection

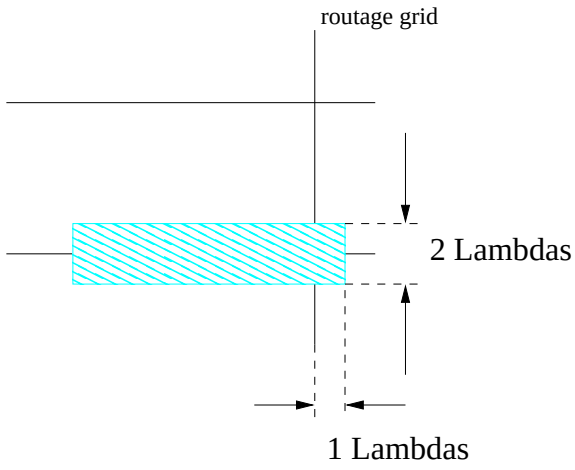


Figure 6: Low size of CAlu1

2.5 steps to follow

2.5.1 Create an inverter

- describe the cell inverter behavior in a file **.vbe** .
- draw the inverter "stick-diagram" inv_x1 whose transistors diagram is represented on the figure 1.

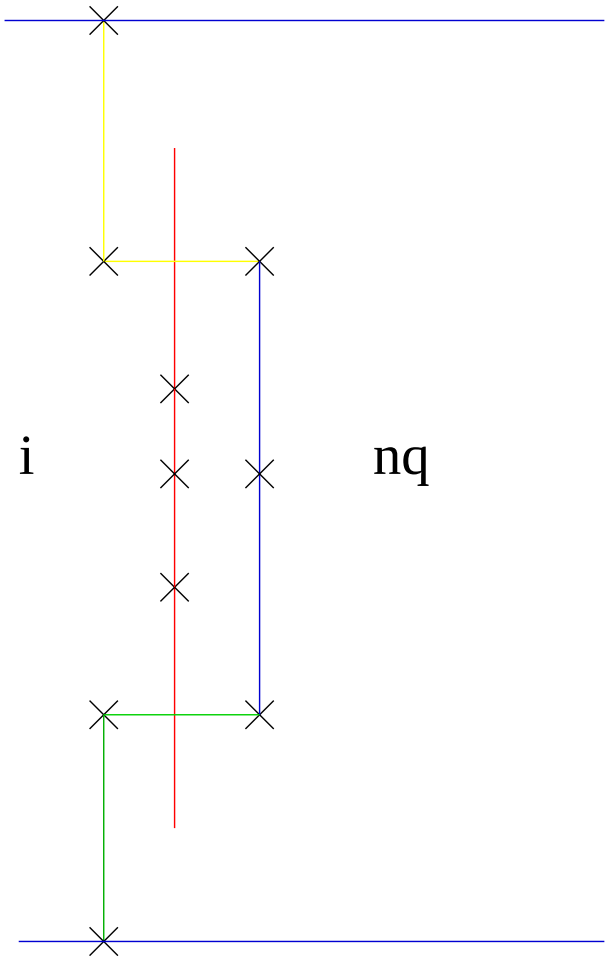


Figure 7: stick diagram

- draw the cell under **GRAAL** by respecting the gauge specified on the figure 4.
- validate the symbolic drawing rules by launching **DRUC** under **GRAAL**.
- extract the *netlist* from the inverter to the format **al** with **COUGAR**.

2.5.2 Create a buffer

The buffer is produced under **GRAAL** starting from the instanciated of two inverters. The hierarchy thus created is represented on the figure 3. The transistors diagram is represented on the figure 2.

- describe the cell buffer behavior in a file **.vbe** .
- draw the cell under **GRAAL** by respecting the gauge specified on the figure 4. You will use for that the instanciated function of **GRAAL** . The cell with instanciate is of course the inverter, which you will connect (will routing) manually.
- validate the symbolic drawing rules by launching **DRUC** under **GRAAL**.
- extract the *netlist* from the buffer to the format **al** with **COUGAR**.

Do not forget that *man* pages exist... We provide you the cells behaviour description inv.vbe and buffer.vbe; and the cells inverter and buffer drawn under **GRAAL** .

3 Place and Route

3.1 Amd2901 architecture

Am2901 breaks up into 2 blocks: the part controls which gathers the logic “ glu ” (random logic) and the operative part (data-path).

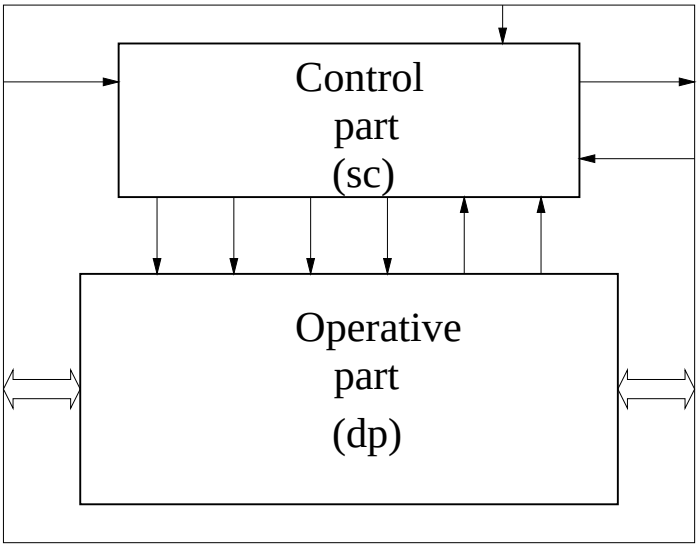


Figure 8: Amd decomposition in functional units

- The data-path contains the regular parts of Amd2901, the registers and the arithmetic logic unit.
- The control part contains irregular logic, the instructions decoding and the “ flags ” computation.

The Hierarchy of descriptions is as follows:

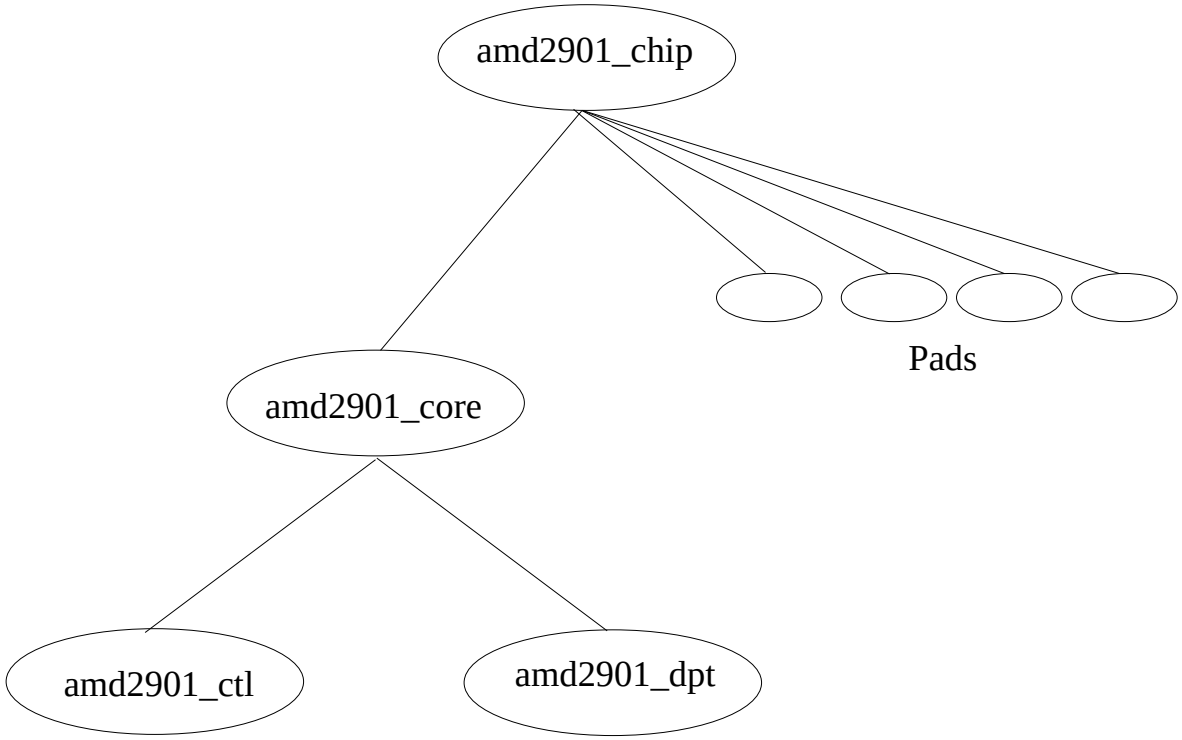


Figure 9: Hierarchy used

3.2 Tools used

You will use place and route tools **ocp** and **nero** , thus all tools for checking seen in the first part of this Tutorial .
ocp is the placer, **nero** allows routing over the cell. The data-path and the control part will be placed and routed together and not separately.
You will use also **lvx**, the netlists comparator. When the system is too complex it is difficult to use **proof**, the formal comparator (calculations too long). A netlists comparison then is used. Test the two methods (**proof** and **lvx**).

3.3 Technological environment

3.4 Beware of file naming

Generally, the file describing a netlist must have the same name as the one describing its physical layout (but of course the file extention is not the same). The file amd2901_dpt.vst (LOFIG) must correspond to the file amd2901_dpt.ap (PHFIG). The same applies to the file amd2901_core. Be carefull not to overwrite a file by mistake !

3.5 Data-path predefined placement

For the moment, your file amd2901_dpt.c describes only the netlist. eg you have a C file that contains the following lines:

```
GENLIB_DEF_LOFIG()
...
GENLIB_SAVE_LOFIG()
```

This permits to generate a structural description in a **VST** file. At the same time, **genlib** will generate physical descriptions of each column in **AP** files. It is up to you to place these columns explicitly. Edit again the file amd2901_dpt.c and include the lines :

```
GENLIB_DEF_PHFIG()
/* add here you placement directives !! */
GENLIB_SAVE_PHFIG()
```

For this placement task, you have the following **GENLIB** functions :

- GENLIB_PLACE()
- GENLIB_PLACE_RIGHT()

- GENLIB_PLACE_TOP()
- GENLIB_PLACE_LEFT()
- GENLIB_PLACE_BOTTOM()
- GENLIB_PLACE_ON()
- GENLIB_DEF_AB()
- ...

Use **GENLIB** manual. The placement of the data-path columns should not be done randomly. The routing feasibility and the quality of the resulting layout depends on it !

Use genlib to generate all:

```
>genlib  amd2901_dpt
```

The figure 10 summarizes the followed process:

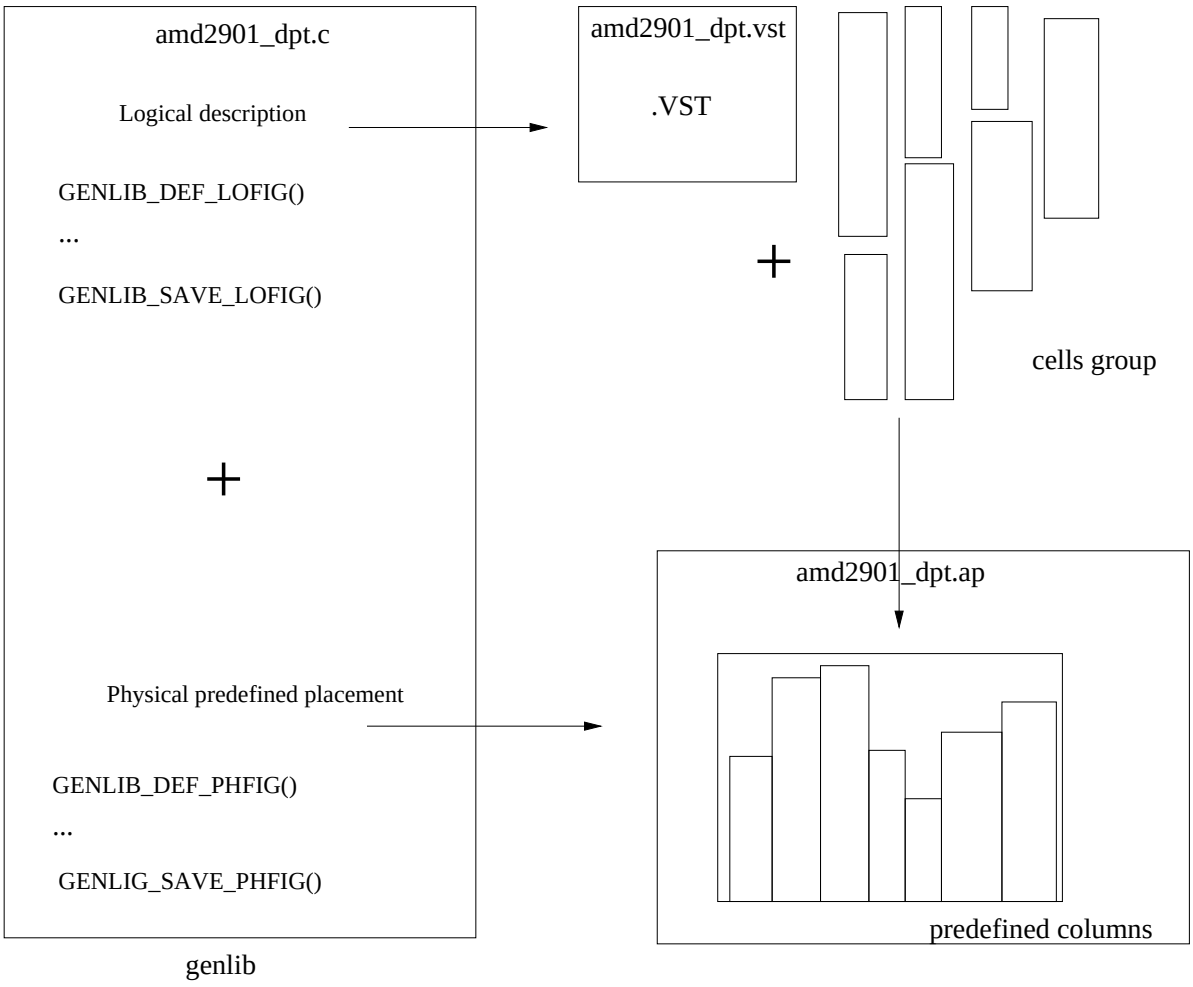


Figure 10: predefined placement

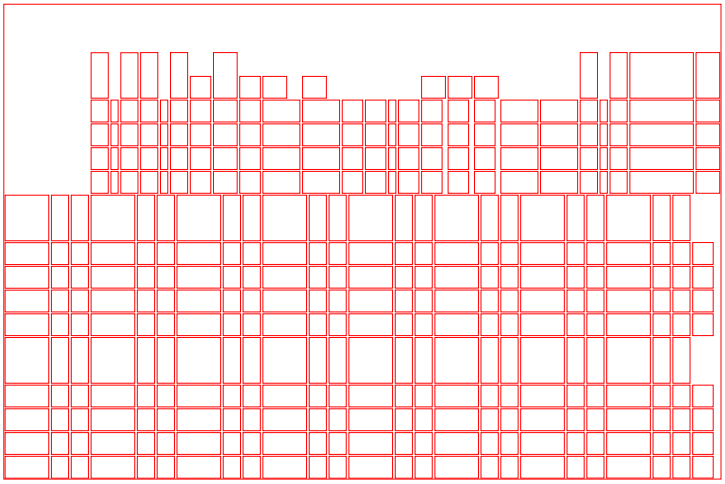


Figure 11: predefined Columns before placement of the part controls

Do not forget to include a abutment box!

3.6 heart Placement

In the same manner, edit agin the file amd2901_core.c and insert data-path explicitly. You should not place the part controls. This one exists only in the form of a structural description. It is the placer **ocp** that will undertake some (during the placement of the heart **ocp** detects which are the cells not placed and supplements the placement). Nevertheless you should reserve enough space for the cells placement **to the top** of the data-path.

Include the lines:

```
GENLIB_DEF_PHFIG()
/* add here placement directives for your data-path */
GENLIB_SAVE_PHFIG()
```

Space necessary to the placer to place the cells of the control part will be determined by successive approximations. You will have to adjust dimensions of the heart abutment box (GENLIB_DEF_AB()). Use the command:

```
> genlib  amd2901_core
```

and

```
> ocp -partial amd2901_core -ioc amd2901_core amd2901_core amd2901_core_p
```

The option **– partial** indicates that you give a partial placement of the data-path. The option **– ioc** permits to specify a placement for external connectors described in a .ioc file. This file, amd2901_core.ioc is provided to you (Modify it according to your predefined placement. The connectors must be in the north and in the south of your circuit).

The third argument is the netlist heart filename, the fourth is the name of the **.ap** resulting file.

The figure 12 summarize the followed process:

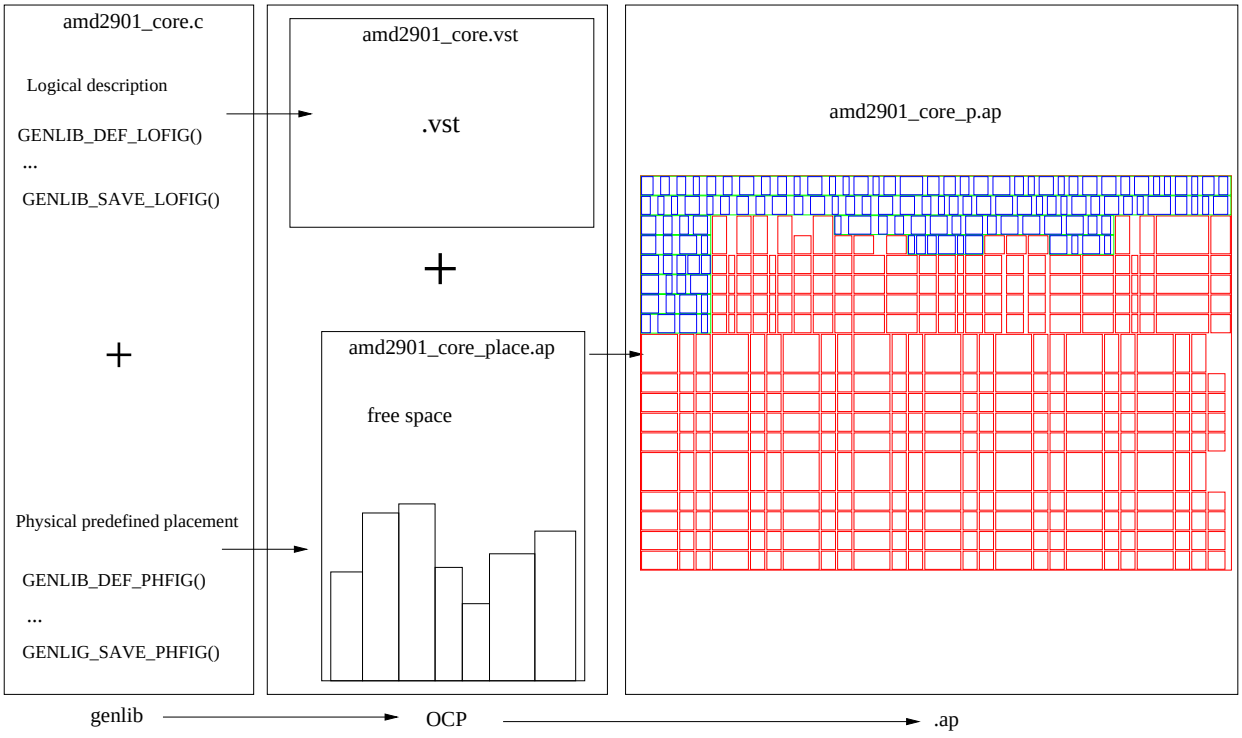


Figure 12: Placement

3.7 Route the heart

Routing the heart by using **NERO** in the following way:

```
> nero -v -3 -p amd2901_core_p amd2901_core amd2901_core
```

3.8 pads placement

The core of the AMD2001 is completed. We focus now on the chip with pads description, placement and routing. Those pads allow the connection of the inputs/outputs of the core with the external nets of the chip.

The tool **ring** instanciates pads that has been specified in a **vst** netlist, place them using a file **.rin** that specified a relative placement of those pads. It then routes those pads with the core according to the input netlist.

This syntax of the **.rin** file:

Where pi1, pi0... are the name of instance pads. Name it " amd2902_chip.rin " and apply the command

```
> ring amd2901_chip amd2901_chip
```

We will validate the work of **ring** with the tools **druc** , **lynx** and **lvx** .

Validate the physical design rules:

```
> druc amd2901_chip
```

Extract the netlist up to leave cells:

```
> MBK_OUT_LO = al
> export MBK_OUT_LO
```

```
> cougar -f amd2901_chip
```

Compare two netlists :

```
> lvx vst al amd2901_chip amd2901_chip -f
```

```
> MBK_OUT_LO = vst
> export MBK_OUT_LO
```

Simulated the extracted netlist with **asimut** . Pay attention to the file **CATAL** !
To know the number of transistors, we carry out an extraction of the circuit on the level transistor:

```
> cougar -t amd2901_chip amd2901_chip
```

If you want to see the amd2901 control part :

```
> make view_ctl_logic
```

If you want to see the data-path physical layout:

```
> make view_dpt_physic
```

note: you can see in red the critical path.
If you want to see the chip physical layout:

```
> make view_chip_physic
```

If you want to see the different propagation times:

```
> make view_chip_simulation
```

4 Annexes

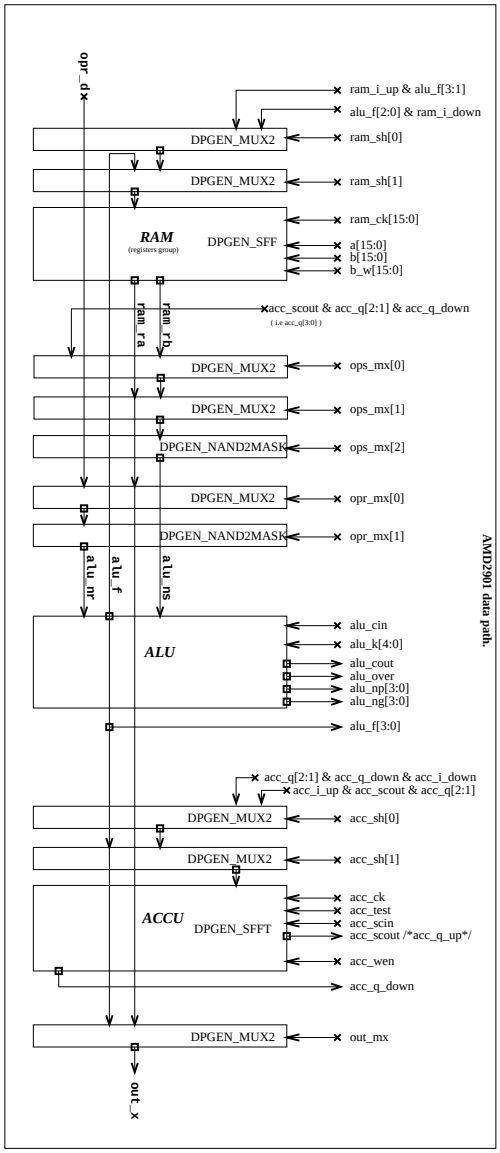


Figure 13: data-path general view