ALLIANCE TUTORIAL

Pierre & Marie Curie University

year 2001 - 2002

PART 3
place and route

Frederic AK Kai-shing LAM

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
- inversor
  - Makefile
  - inversor.vbe : behavioral description
  - inv_x1.ap : inversor cell under GRAAL
- buffer
  - Makefile
  - buffer.vbe : behavioral description
  - buf_x2.ap : buffer cell under 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, RING** place and route tools .

The beginning of this tutorial will relate to the drawing under **GRAAL** of a inversor 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 Inversor 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**. The first part here aims to draw a inversor cell inv_x1 in the shape of a predefined cell sxlib by complying with the 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 positioned with **/asim/alliance/etc/cmos_12.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 indicate a possible portion 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 positioned with **/asim/alliance/etc/cmos_12.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 **vst** starting from a description with the format **ap** . To extract on the level transistor, the command to be used is:

```
> 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 netlist with the format **.al** starting from a description **.ap** it is necessary to write:

```
> MBK_IN_PH = ap
> export MBK_IN_PH
> MBK_OUT_LO = al
> export MBK_OUT_LO
```

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

2.1.4 YAGLE

The tool **YAGLE** is able to extract the behavioral VHDL description of a circuit to the format **.vbe** starting from a *netlist* with the format **.al** *if this one is on the transistor level* . The command to be used is:

```
> MBK_IN_LO = al
> export MBK_IN_LO
> YAGLE_BEH_FORMAT = vbe
> export YAGLE_BEH_FORMAT
```

```
> yagle file1 file2
```

Above all, you must use the command:

```
> source /users/soft5/newlabo/AvtTools/etc/avt_env.csh
```

which allows to set up the environment necessary to use **YAGLE** .
Documentations for this tool are in : **/users/soft5/newlabo/AvtTools/doc**

But the tool **YAGLE** is not part of Alliance anymore. If you want to use it, you have to get the licence from Avertec.

2.1.5 PROOF

When we want to prove the equivalence between two behavioral descriptions of the same circuit with N inputs, we can simulate by asimut 2^n vectors for two descriptions and compare them. This solution quickly becomes expensive in CPU time and it is better to use formal proof tool which carries out the "mathematical" comparison of the two Boolean networks. **PROOF** carries out this operation between descriptions file1.vbe and file2.vbe by the command:

```
> proof file1 file2
```

2.2 inversor Diagram

The theoretical inversor diagram is presented at the following figure:

Figure 1: transistors diagram of a C-MOS inversor

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 inversors 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 obligatorily 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 inversor

- describe the cell inversor behavior in a file **.vbe** .

- draw the inversor "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 inversor to the format **al** with **COUGAR**.
- extract the behavioral VHDL with **YAGLE**
- carry out the formal proof between the file **.vbe** extracts by **YAGLE** and the file **.vbe** from the initial specification.

2.5.2 Create a buffer

The buffer is produced under **GRAAL** starting from the instanciated of two inversors. 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 inversor, 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**.
- extract the behavioral VHDL with **YAGLE**
- carry out the formal proof between the file **.vbe** extracts by **YAGLE** and the file **.vbe** from the initial specification.

Do not forget that the *mans* exist... We provide you the cells behaviour description inversor.vbe and buffer.vbe; and the cells inversor and buffer draws under **GRAAL** .

3 Place and Route

3.1 Amd2901 architecture

Am2901 breaks up into 2 blocks: the part controls which gathers the logical " glu " 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 " calculation.

Hierarchical description used is as follows:

Figure 9: Hierarchy used

3.2 Tools used

You will use place and route tools **ocp** and **ocr** , thus all tools for checking seen in the first part of this Tutorial .
ocp is the placer, **ocr** 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

```
> VH_MAXERR = 10
> export VH_MAXERR
> MBK_WORK_LIB = .
> export MBK_WORK_LIB
> MBK_CATA_LIB = $ALLIANCE_TOP/cells/sxlib
> export MBK_CATA_LIB
> MBK_CATA_LIB = $MBK_CATA_LIB:$ALLIANCE_TOP/cells/dp_sxlib
> export MBK_CATA_LIB
> MBK_CATA_LIB = $MBK_CATA_LIB:$ALLIANCE_TOP/cells/padlib
> export MBK_CATA_LIB
> MBK_CATA_LIB $MBK_CATA_LIB:.
> export MBK_CATA_LIB
> MBK_CATAL_NAME = CATAL
> export MBK_CATAL_NAME
> MBK_IN_LO = vst
> export MBK_IN_LO
> MBK_OUT_LO = vst
> export MBK_OUT_LO
> MBK_IN_PH = ap
> export MBK_IN_PH
> MBK_OUT_PH = ap
> export MBK_OUT_PH
```

3.4 Beware of naming the files

Generally, the files describing a logical netlist must be the same name as the corresponding file describing the physical netlist. the file amd2901_dpt.vst (LOFIG) must correspond to the file amd2901_dpt.ap (PHFIG). The same applies to the file amd2901_core. Check well that you do not overwrite a file!

3.5 Data-path predefined placement

For the moment, your file amd2901_dpt.c contains only one logical description of the netlist. eg you have a file C which contains the lines:

GENLIB_DEF_LOFIG()
...
GENLIB_SAVE_LOFIG()

That enables you to generate a description structural in file **VST** . But at the same time, **genlib** generated physical descriptions of each column in files **AP** . It is about placing these columns explicitly.
Take again the file amd2901_dpt.c and include the lines :

GENLIB_DEF_PHFIG()
...
GENLIB_SAVE_PHFIG()

The suspension points are to be supplemented, they represent your operators placement. You have for, that **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 made randomly. The routing 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

Same manner, take again the file amd2901_core.c and place 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** which will undertake some (during the placement of the heart **ocp** detects which are the cells not placed and supplements the placement). You should nevertheless envisage space for the cells placement **to the top** of the data-path.

Include the lines:

GENLIB_DEF_PHFIG()
...
GENLIB_SAVE_PHFIG()

The suspension points represent the placement of data-path. 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 transmit a partial placement of the data-path. The option **– ioc** request the loading of a file giving the placement of the connectors. This file, amd2901_core.ioc is provided to you (Modify it according to your predefined placement. The connectors must be in north and the south). The third argument is the netlist heart, the fourth is the file **AP** result.

The figure 12 summarize the followed process:

Figure 12: Placement

3.7 Route the heart

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

```
> ocr  -P amd2901_core_p -L amd2901_core -O amd2901_core -l 3 -v -i 30
```

3.8 pads placement

The heart is now completed. The pads still should be added allowing the connection of the inputs/outputs to the case.
The tool **ring** allows to instanciate the pads it has need for signals list describing the relations between the heart and the pads, as well as a file **.rin** specifying the geometrical provision of the crown of pads.

This file uses syntax:

```
> east ( pi1 pi0 )
> west ( pck pi4 )
> north ( pvdd pvss )
> south ( pvdde pvsse )
```

Where pi1, pi0... are the names of the pads ”instances”. 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 drawing rules:

```
> druc amd2901_chip
```

Extract the symbolic layout and flattened it:

```
> 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 file extracts 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