PROJECT: ALLIANCE DESIGN FLOW

Ain Shams University

PART 3
Physical Synthesis

Frederic AK Kai-shing LAM

Modified by LJ
Pierre & Marie Curie University

Adapted by M. Dessouky

Ain Shams University

2017

Contents

1 Introduction

In this project, you'll build a simple frame decoder chip. There are four parts of the project:

1. High-Level Design.
2. Low-Level Synthesis.
3. Design for test.
4. Placement and Routing (this document)

This document describes part (4) of the project; Physical Synthesis.
The **ALLIANCE** tools used are:

- **ocp:** Standard Cell Placer.
- **nero:** Over-Cell Router.
- **cougar:** Symbolic Netlist Extractor.
- **lvx:** Netlist comparator.
- **graal:** Symbolic layout editor.
- **druc:** Symbolic Design-rule checker.
- **Ring:** Pad ring router.
- **s2r :** Symbolic-to-Real layout converter.
- **dreal :** Real layout editor.

During physical synthesis, we are going to use the following technology parameter files (Both can be found in the supplied project_techno.tar archive):

- **techno/techno-symb.rds :** the Symbolic technology parameters file.
- **techno/techno-035.rds :** A generic 0.35 micron technology parameters file.

2 Steps to follow

In the previous part of the project, we obtained several gate-level netlists (.vst) corresponding to different state encodings using the tool **SYF**. After netlist optimization, select the best netlist to continue with in the placement and routing flow below.

Generally, the file describing a netlist must have the same name as the one describing its physical layout (but of course the file extension is not the same). The netlist file *file_name*.vst must correspond to the layout file *file_name*.ap. Be careful not to overwrite a file by mistake!

2.1 Floorplanning

Before starting physical synthesis, you must first construct the chip floorplanning. A sample floorplanning is provided in the lecture.

2.2 Placement

To launch placement, use the **ocp** tool;

```
>ocp -v -ring -ioc <connectors> <input\_netlist> <output\_layout>
```

The **–ioc** option permits to specify a placement for external connectors (pins) described in a .ioc file. Please refer to the lecture for an example of such file. The connectors must be in the north and in the south of your circuit. The –ring option permits to automatically place the connectors for the ring pad placement tool. The <input_netlist> is the optimized gate-level netlist filename without extension. Note that we never use file extensions with Alliance tools. Instead file types, and hence extensions, are specified using environment variables. In this case, the **MBK_IN_LO** variable should be set to **vst**. The last argument is the name of the **.ap** resulting symbolic layout file. The **MBK_OUT_PH** variable should be set to **ap**. It is recommended to name the output layout differently from the netlist.

In order to review the resulting placement, the symbolic layout editor **graal** can be used. Before launching graal, you should set the technology file environment variable **RDS_TECHNO_NAME** to the symbolic technology parameter file **techno/techno-symb.rds**. Graal alows to flatten the design to inspect the layout details of the standard cells.

2.3 Routing

Routing the placed cells is done by using **nero** in the following way:

```
> nero -V -p <placement_layout> <netlist> <layout>
```

The **–p** option sets the name of the input placement layout file. Otherwise, the placement file is expected to have the same name as the netlist. **The final layout must have the same file name as the netlist**, but with different extensions. This will be required by the **ring** pad placement and routing tool, see below.

The layout editor tool **graal** can be then used to inspect the output layout. It should be the same as the placement file with over-the-cell routing added.

2.4 Post-Layout Verification

Post-layout verification is composed of two steps: electrical and physical. Electrical verification checks that the extracted netlist is exactly the same as the gate-level netlist (LVS), while physical verification checks if there are design-rule errors in the generated layout (DRC).

2.4.1 Layout-vs-Schematics (LVS)

Netlist comparison is done on two steps; first the netlist is extracted using the **cougar** tool. The original netlist is in **vst** format. We'll have the extracted netlist in **al** format, in order not to overwrite files. This can be achieved by setting the **MBK_OUT_LO** to **al**. Also, you should set the technology file environment variable **RDS_TECHNO_NAME** to the real technology parameter file **techno/techno-035.rds**.

Netlist extraction is then performed using the command:

```
> cougar -v <input\_layout> <extracted\_netlist>
```

This is followed by netlist comparison using the command:

```
> lvx vst al <input\_netlist> <extracted\_netlist> -f
```

2.4.2 Design-Rule Checking (DRC)

Design-rule checking can be done using two methods:

- Inside the layout editor tool **graal**, open the generated layout. Select from the menus Tools->Druc, then select the whole layout using the mouse.
- Using the **druc** design rule checker. In order to check symbolic rules, the technology file environment variable **RDS_TECHNO_NAME** should be set to the symbolic technology parameter file **techno/techno-symb.rds**.

```
> druc <layout>
```

2.5 Chip Assembly

In order to complete the chip, IO pads must be placed around the core in consistence with the designed floorplan. The first step in chip assembly is to create the chip netlist, which instantiates the core netlist in addition to the needed pads (.vst).

However, this structural netlist description (.vst) of the whole chip must be written by hand, since a top-level netlist including the pads does not exist. Such netlist instantiates core gate-level vst resulting from synthesis, in addition to the pads and makes necessary connections between the core pins and corresponding pad pins.

In order to know pad cell pins, open the pad (.vbe) in padlib: (/usr/lib/alliance/cells/padlib).

Used pads are:

- pck_sp (ck pad, only one pad),
- pi_sp (one for each input pad),
- po_sp (one for each output pad),
- pvddi_sp (vdd, only one pad),
- pvssick_sp (supplies both vss and ck to the core, only one pad),
- pvdde_sp (vdd for pads, only one pad),
- pvsse_sp (pvss for pads, only one pad).

In such pad cells, pin names are described below:

- vddi: core vdd (should be connected to all pads + core)
- vssi: core vss (should be connected to all pads + core)
- vdde: pad vdd (should be connected to all pads only)
- vsse: pad vss (should be connected to all pads only)
- ck: pad ck (should be connected to all pads only. Not connected to core)
- cko: clock, must be connected to core (found only in pvssick_sp)

In addition, the placement of pads around the core must be decided in a **.rin** file. This file must have the same name as the chip netlist above but with a different extension. Refer to the lecture for the .rin file syntax:

- VDD and VSS must be opposite to each other near core center, otherwise errors would appear.
- Must place PADS facing the corresponding core PIN on the same side.
- Use pad instance names used while creating the chip vst netlist.

Please refer to the lecture for more details. Finally, chip placement and routing is accomplished using the command:

```
> ring <chip\_netlist> <chip\_layout>
```

where <chip_netlist> defines the name of both the input netlist and the (.rin) file, while <chip_layout> defines the name of the output chip layout with the pads inserted and routed.

It should be noted that the core netlist (.vst) instantiated in the chip netlist (.vst) and corresponding core routed layout by the nero tool (.ap) should have the same name.

2.6 Chip Verification

The generated chip layout must undergo both electrical and physical verification outlined in section 2.4, both LVS and DRC at the chip level.

2.7 Symbolic-to-Real Conversion

Finally, the generated symbolic layout should be converted to a real technology. We'll use a cif output format for the real layout. The **RDS_OUT** variable must be set to **cif**. Technology is specified by setting the **RDS_TECHNO_NAME** to the real technology parameter file **techno/techno-035.rds**. Conversion is then done using the command:

```
> s2r -v -r <chip\_layout>
```

You should obtain a <chip_layout>.cif file ready for fabrication.

You can view this file using the **dreal** real layout viewer.