TUTORIAL : AMD 2901

Welcome to the **ALLIANCE** CAD system.
The goal of this tutorial is to present a typical design-flow of a simple 4 bits processor (AMD2901), using the **ALLIANCE** tools.
The tools used here are :

- **asimut** : **VHDL** compiler and simulator.
- **genlib** : Net-List capture.
- **scr** : Standard Cell placer and router.
- **ring** : Core to pads router.
- **lynx** : Symbolic layout extractor.
- **lvx** : Net-List comparator.
- **druc** : Design rule checker.
- **graal** : Graphic layout editor.
- **yagle** : Functional abstractor.
- **proof** : Formal proof between two behavioral descriptions.
- **s2r** : Symbolic to real layout converter.
- **tas** : Timing Analysis static.

At any time you can get information on any **ALLIANCE** tool using the command :

```
> man <tool name>
```

All the available documentation is described in the **README** file. This tutorial doesn't contain a description of the **AMD2901**, but the methodology used in **ALLIANCE** to produce it. Nevertheless, you will find in the file **AMD2901_doc.ps.tar**, the original data-sheet of the circuit. To uncompress the sheets, use the **UNIX** command :

```
>tar xvf AMD2901_doc.ps.tar
>uncompress AMD2901*.Z
```

A GIF format of the data-sheet is also available in the file **AMD2901_doc.gif.tar**.
The design-flow used in this tutorial is composed of five steps :

- **Step 1** : Behavioural description (**VHDL** model) and simulation.
- **Step 2** : Generation and validation of the structural description (gate net-list).

- **Step 3** : Physical design layout (Place and route).
- **Step 4** : Extraction and verification.
- **Step 5** : Timing analysis.
- **Step 6** : Chip finishing.

In order to build the chip in about two hours, all the source files are provided with this tutorial :

- **amd.vbe** : **VHDL** behavioural model.
- **pattern.pat** : Simulation Patterns.
- **chip.c** : Chip structural description.
- **chip.rin** : Pads placement (for **ring**).
- **heart.scr** : Connectors placement (for **scr**).
- **chip.inf** : Registers renaming (for **desb**).

You will use the **ALLIANCE** tools to validate these input files and to build the physical layout. The output is a **CIF** layout file ready for the foundry.
To build the chip you can :

- - run the **UNIX** commands in the order indicated by this tutorial.
- - build automatically the entire **AMD2901 CHIP** using the command :

  ```
  > make
  ```

  If you want to start again this example from the begining, you just have to type :

  ```
  > make clean
  > make
  ```

All **ALLIANCE** tools use a set of **UNIX** environment variables. These variables are set by the **UNIX** command "export".

```
> MBK_WORK_LIB=.
> MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/cells/rin
> export MBK_WORK_LIB MBK_CATA_LIB
```

- **MBK_CATA_LIB :** This variable defines all paths to the directories containing the **ALLIANCE** predefined cell libraries.
- **MBK_WORK_LIB :** This variable defines the working directory : all user files will be written in this directory.

Some of the path names have to be modified in order to correspond to your particular installation of Alliance, in this tutorial we will assume that the directory structure of the sources files has not been altered In this tutorial, the commands which are inside [ ] are preset. If you stay in the same shell during the making of the **AMD2901**, you don't have to set again these environment variables.

1 Behavioural specification

1.1 Behavioural model

The circuit behaviour is described in the amd.vbe file using the **ALLIANCE VHDL** subset (see *man vhdl* and *man vbe*). You must run the **VHDL** compiler to validate the **VHDL** file syntax.

```
[> MBK_WORK_LIB=.]
[> MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/cells/ri
[> export MBK_CATA_LIB MBK_WORK_LIB]
> asimut -b -c amd
```

- **amd** is the filename of the behavioural description (**amd.vbe**).
- **-b** means that the file is a pure behavioural description.
- **-c** stands for compilation only (compilability is checked, no simulation is performed).

1.2 Simulation patterns

As you have specified a formal specification, you have to define a set of simulation patterns. The same patterns will be used to validate each design step from specification to physical layout.
The file **pattern.pat** is given as an example.

1.3 Simulation

You have now a logical description of your circuit, and a list of patterns. You can run the zero-delay VHDL simulator **asimut**. You have to set up some new environment variables specific to **asimut**.

```
> VH_MAXERR=10
> export VH_MAXERR
```

- **VH_MAXERR** : The maximum number of errors accepted before **asimut** stops simulation.

```
> VH_PATSFX=pat]
> export VH_PATSFX]
```

- **VH_PATSFX** the extension of simulation patterns file.

```
[> VH_MAXERR=10]
[> VH_PATSFX=pat]
[> MBK_WORK_LIB=.]
[> MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/cells
[> export MBK_CATA_LIB VH_MAXERR VH_PATSFX MBK_WORK_LIB]
> asimut -b amd pattern result_beh
```

- **amd** : is the filename of the description (**amd.vbe**).
- **pattern** : is the filename for the input patterns (**pattern.pat**).
- **result_beh** : is the filename for the resulting patterns (**result_beh.pat**).
- **-b** : means that the **amd** file is a pure behavioural description.

The input pattern file can provide the expected outputs. Any difference between predicted outputs in **pattern.pat** and simulation results will be reported as commentary on the screen and in **result_beh.pat**. This output file may be used again for an another simulation, since it has the same format as the input file.

2 Structural Description

2.1 Structural Design

The aim of this step is to build the gate level schematic corresponding to the behavioural specification.

We must instantiate and connect logical gates and pads, supplied in the standard cell library **scr** and the pad library **ring**. To do this, **ALLIANCE** doesn't use a schematic editor but the procedural language **genlib** which is a set of C functions (see *man genlib*).

The structural description of the **AMD2901** follows a hierarchical approach.

- First, each functionnal block is described as a separate gate net-list. There are five blocks : **accu, alu, ram, muxe, muxout**.
- The next hierarchical level is the **heart** that interconnects these five blocks among each others.
- The final hierarchical level is the **chip** that connects the **heart** to the **pads**.

The **chip.c** file uses the **genlib** language to describe these 7 hierarchical blocs.

The **ALLIANCE** system accepts several external file formats for net-list (EDIF, VHDL, SPICE, ALLIANCE, VLSI). Environment variables allow to set the formats. The **.vst** extension corresponds to the structural **VHDL**.

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

All net-list files created and used by **genlib** will now be structural **VHDL**.

```
[ > MBK_IN_LO=vst ]
[ > MBK_OUT_LO=vst ]
[ > MBK_WORK_LIB=. ]
[ > MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/cells/ri
[ > export MBK_CATA_LIB MBK_IN_LO MBK_OUT_LO MBK_WORK_LIB]
> genlib -v chip
```

The 7 files **alu.vst accu.vst ram.vst muxe.vst muxout.vst heart.vst chip.vst** have been created.

2.2 Structural Model Simulation

The various net-list files we have created represent the structural description that should be validated by simulation, using the same patterns as in step 1.

```
[ > MBK_IN_LO=vst ]
[ > VH_PATSFX=pat ]
[ > VH_MAXERR=10 ]
[ > MBK_WORK_LIB=. ]
[ > MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/cells/ri
[ > export MBK_CATA_LIB MBK_IN_LO VH_PATSFX
    VH_MAXERR MBK_WORK_LIB ]
> asimut chip pattern result_str
```

- **chip** is the structural input file (**chip.vst**).
- **pattern** : is the filename for the input patterns (**pattern.pat**).
- **result_str** : is filename for the resulting patterns (**result_str.pat**).

Note the **-b** option does not appear any more, since we now have a structural model.

Any new error appearing will be reported in the **result_str.pat** file. You can try to introduce a schematic error by modifying the **chip.c** file, running **genlib** and **asimut** to see the errors.

3 Physical Design

Building the heart, and routing the heart to the pads are quite different jobs. So we use different tools.

3.1 Routing the heart

Scr is a standard cell placing and routing tool.

Here again **ALLIANCE** accepts several external file formats for the symbolic layout. Environment variables allow to set the formats.

```
> MBK_IN_PH=ap
> MBK_OUT_PH=ap
[ > export MBK_IN_PH MBK_OUT_PH
```

The hierarchical net-list will be flattenned before routing.

```
[ > MBK_IN_PH=ap ]
[ > MBK_OUT_PH=ap ]
[ > MBK_IN_LO=vst ]
[ > MBK_WORK_LIB=. ]
[ > MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/cells/ri
[ > export MBK_CATA_LIB MBK_IN_PH MBK_OUT_PH
    MBK_IN_LO MBK_WORK_LIB]
> scr -p -r heart
```

- **heart** is the input net-list (heart.vst) and the connector placement (heart.scr). The output layout will be heart.ap.
- **-p** automatic placement.
- **-r** routing required.

The symbolic layout file **heart.ap** has been created.

3.2 Heart verification

- DRC : Druc checks the symbolic layout rules (see *man druc*)

  ```
  [ > MBK_IN_PH=ap ]
  [ > MBK_WORK_LIB=. ]
  [ > MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/ce
  > RDS_TECHNO_NAME=$ALLIANCE_TOP/etc/cmos_7.rds
  > RDS_IN=cif
  > RDS_OUT=cif
  [ > export MBK_CATA_LIB MBK_IN_PH MBK_WORK_LIB
      RDS_TECHNO_NAME RDS_IN RDS_OUT]
  > druc heart
  ```

  If necessary, **druc** generates an error file (**core.err**).

- Net-list extraction.

  Lynx is a hierarchical extractor that provides a gate net-list.
  In order to avoid name collision, we use another file format for extracted net-list (**.al** format). This new format is used to support additional data

: extracted parasitic capacitances.

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

Then we run the **lynx extractor** :

```
[ > MBK_OUT_LO=al ]
[ > MBK_IN_PH=ap ]
[ > MBK_WORK_LIB=. ]
[ > MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/ce
[ > export MBK_CATA_LIB MBK_OUT_LO MBK_IN_PH MBK_WORK_LIB]
> lynx heart
```

The extracted net-list file **heart.al** is created at this level.

- Net-compare.

  The next tool is the net-compare **lvx** that performs a netlist comparison between the input net-list **heart.vst** and the extracted net-list **heart.al**, after flattening to the gate level.

```
[ > MBK_WORK_LIB=. ]
[ > MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/ce
[ > export MBK_CATA_LIB MBK_WORK_LIB]
> lvx vst al heart heart -f
```

3.3 Routing the chip

The pad placement depends on external constraints. The file **chip.rin** defines the pads placement constraints (see *man ring*).

```
[ > MBK_IN_LO=vst ]
[ > MBK_IN_PH=ap ]
[ > MBK_OUT_PH=ap ]
[ > MBK_WORK_LIB=. ]
[ > MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/cells/ri
[ > export MBK_CATA_LIB MBK_IN_LO
    MBK_IN_PH MBK_OUT_PH MBK_WORK_LIB]
> ring chip chip
```

- **chip** The input files chip.vst and chip.rin (the same name **chip** must be used for the two files).
- **chip** The output physical file chip.ap.

The symbolic layout file **chip.ap** has been created. At this point, you can display the chip with the layout editor **graal**.
You can see the chip and the instantiated heart, using the **graal** commands to go through the hierarchy levels.

```
[ > MBK_WORK_LIB=. ]
[ > MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/cells/ri
[ > export MBK_CATA_LIB MBK_WORK_LIB]
> graal
```

4 Physical Validation

The **ALLIANCE** verification tools allow both flat and hierarchical verification.
We will use a hierarchical approach.

4.1 Chip verification

The same procedure used for the heart applies for the entire chip.

```
[ > MBK_IN_PH=ap ]
[ > MBK_WORK_LIB=. ]
[ > MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/cells/ri
[ > RDS_TECHNO_NAME=$ALLIANCE_TOP/etc/cmos_7.rds]
[ > RDS_IN=cif]
[ > RDS_OUT=cif]
[ > export MBK_CATA_LIB MBK_IN_PH MBK_WORK_LIB
    RDS_TECHNO_NAME RDS_IN RDS_OUT]
> druc chip
```

Eventually, **druc** generates an error file (**chip.err**).

```
[ > MBK_OUT_LO=al ]
[ > MBK_IN_PH=ap ]
[ > MBK_WORK_LIB=. ]
[ > export MBK_CATA_LIB MBK_OUT_LO MBK_IN_PH]
> lynx chip
```

The extracted net-list file **chip.al** is created by this step (This net-list instantiates the heart).

```
[ > MBK_WORK_LIB=. ]
[ > MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/cells/ri
[ > export MBK_CATA_LIB MBK_WORK_LIB]
> lvx vst al chip chip
```

Both net-lists **chip.vst** and **chip.al** are flattened to the gate level by **lvx** before comparison (see *man catal*).

4.2 Chip simulation

Finally you can check globally the extracted net-list by applying the original patterns to the extracted net-list. Thanks to the multiformat approach the simulator **asimut** accepts **.al** format as structural input description.

```
> MBK_IN_LO=al
> export MBK_IN_LO
```

Then you have to run **asimut**.

```
[ > MBK_IN_LO=al]
[ > VH_PATSFX=pat]
[ > VH_MAXERR=10]
[ > MBK_WORK_LIB=. ]
[ > MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/cells/ri
[ > export MBK_CATA_LIB MBK_IN_LO
    VH_PATSFX VH_MAXERR MBK_WORK_LIB]
> asimut chip pattern res_pattern
```

To complete the validation of our chip, the functional abstraction followed by the proof must be done.

4.3 Functionnal abstraction and formal proof

The functional abstraction of the transitor net-list is realized by the tool **yagle** (see *man yagle*). **yagle** flattens the chip (**chip.al**) to obtain a transistor level description, and abstract then a behavioral description (**chip.vbe**). To keep the coherence of register names between the behavioral description given by **yagle** (**chip.vbe**) and the initial behavioral description (**amd.vbe**), a file describing the name's transformation is required. This file is **chip.inf**.

```
[ > MBK_IN_LO=al ]
[ > MBK_WORK_LIB=. ]
[ > MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/cells/ri
[ > export MBK_CATA_LIB MBK_IN_LO MBK_WORK_LIB]
> yagle chip -i -v
```

- **chip** is the input layout (**chip.al**).
- **-i** read the **chip.inf** file.
- **-v** to vectorize the interface of behavioral description.

The VHDL DATA FLOW description as been generated : **chip.vbe**

Now you have the abstracted behavioral description (**chip.vbe**) and the initial behavioral description (**amd.vbe**). With **proof** (see *man proof*), you can check the formal equivalence between the two descriptions.

```
[ > MBK_WORK_LIB=. ]
[ > export MBK_WORK_LIB]
> proof -d amd chip
```

- **chip** is the abstracted file (**chip.vbe**).
- **amd** is the initial behavioral description (**amd.vbe**).
- **-d** displays errors.

If you want to see an error you can change one line in the file **amd.vbe**. For example, change the line 301 :
scout <= NOT accu(3) AND test_mode ;
and try **proof** again.

5 Timing analysis

With the extracted net list chip.al TAs analyses each path of the circuit taking account of the various capacitances. These results are summarize in a file .ttv In this file we can see the critical path and the correspnding time. So it is possible to know the period of the chip

You will enter :

```
[ > MBK_IN_PH=ap ]
[ > MBK_IN_LO=al ]
[ > MBK_WORK_LIB=. ]
[ > MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/cells/ri
[ > export MBK_IN_PH MBK_WORK_LIB MBK_CATA_LIB ]
> tas chip
```

6 Chip finishing

Until now we used symbolic layout (all coordinates in **lambda**-units). We should now convert them to real dimensions, creating the physical layout (two output formats are supported : CIF and GDSII). This last step is done by **s2r**. **s2r** performs symbolic to real expansion, gap filling, denotching and instantiates preexisting physical cells (this is necessary for the pads).

You must define 3 new environment variables :

```
> RDS_TECHNO_NAME=/$ALLIANCE_TOP/etc/prol10.rds
> export RDS_TECHNO_NAME
```

The **prol10.rds** is the technology file that contains the parameters corresponding to the target process. (In this case the target process is 1 micron.)

```
> RDS_OUT= cif
> export RDS_OUT
```

Defines the output format.

```
> RDS_IN=cif
> export RDS_IN
```

Defines the input format for the preexisting layout cells (pads).

You will enter :

```
[ > MBK_IN_PH=ap ]
[ > MBK_WORK_LIB=. ]
[ > MBK_CATA_LIB=.:/$ALLIANCE_TOP/cells/sclib/prol10:/$ALLIANCE_TOP/cells/ri
[ > RDS_TECHNO_NAME=/$ALLIANCE_TOP/etc/prol10.rds ]
[ > RDS_IN=cif ]
[ > RDS_OUT=cif]
[ > export RDS_OUT MBK_IN_PH MBK_WORK_LIB
    MBK_CATA_LIB RDS_TECHNO_NAME RDS_IN]
> s2r -c chip amd2901
```

- **-c** deletes connectors at the highest level of hierarchy.

You did it...
The file produced is **amd2901.cif**. It is ready for the foundry.