ALLIANCE & CORIOLIS Checker Toolkit

Contents

Toolkit Purpose

This toolkit has been created to allow developpers to share through `git` a set of benchmarks to validate their changes in ALLIANCE & CORIOLIS before commiting and pushing them in their central repositories. A change will be considered as validated when all the developpers can run successfully all the benchs in their respective environments.

As a consequence, this repository is likely to be *very* unstable and the commits not well documenteds as they will be quick corrections made by the developpers.

Toolkit Contents

The toolkit provides:

- Six benchmark designs:

Design	Technology	Cell Libraries
`adder`	MOSIS	`msxlib`, `mpxlib`, `msplib`
AM2901 (standard cells)	ALLIANCE dummy	`sxlib`, `pxlib`
AM2901 (datapath)	ALLIANCE dummy	`sxlib`, `dp_sxlib`, `pxlib`
AM2901	ALLIANCE dummy	`sxlib`, `pxlib`
`alliance-run` (AM2901)	ALLIANCE dummy	`sxlib`, `dp_sxlib`, `padlib`
SNX	MOSIS	`msxlib`, `mpxlib`, `msplib`

- Three cell libraries.

  All thoses libraries are for use with the MOSIS technology. We provides them as part of the toolkit as we are still in the process of validating that technology, and we may have to perform quick fixes on them. The design are configured to use them instead of those supplied by the ALLIANCE installation.

  - `msxlib` : Standard Cell library.
  - `mpxlib` : Pad library, compliant with CORIOLIS.
  - `msplib` : Pad library, compliant with ALLIANCE / `ring`. Cells in this library are *wrappers* around their counterpart in `mpxlib`, they provides an outer layout shell that is usable by `ring`.

- The RDS file for the MOSIS technology `scn6m_deep_09.rds`, for the same reason as the cell libraries.
- Miscellenous helper scripts.

Benchmark Makefiles

The main body of the `Makefile` has been put into `benchs/etc/rules.mk`.

The `Makefile` in the various bench directories provides some or all this targets, according to the fact they can be run with CORIOLIS, ALLIANCE or both.

ALLIANCE	`layout-alc`	The complete layout of the design (P&R).
	`druc-alc`	Symbolic layout checking
	`lvx-alc`	Perform LVS.
	`graal`	Launch `graal` in the `Makefile` 's environement
	`dreal`	Launch `dreal` in the `Makefile` 's environement, and load the `gds` file of the design.
CORIOLIS	`layout`	The complete layout of the design (P&R).
	`druc`	Symbolic layout checking
	`lvx`	Perform LVS.
	`view`	Launch `cgt` and load the design (chip)
	`cgt`	Launch `cgt` in the `Makefile` 's environement

A top `Makefile` in a bench directory must looks like:

```
                    CORE = adder
                    CHIP = chip
                  MARGIN = 2
         GENERATE_CORE_VST = Yes
             USE_CLOCKTREE = No
                 USE_MOSIS = Yes
                 USE_DEBUG = No

include ../etc/rules.mk

export          MBK_IN_LO = vst
export         MBK_OUT_LO = vst
export             RDS_IN = gds
export            RDS_OUT = gds

check:      lvx

layout:     chip_crl_kite.ap
lvx:        lvx-chip_crl_kite
druc:       druc-chip_crl_kite
gds:        chip_crl_kite.gds
view:       cgt-view-chip_crl_kite

lvx-alc:    lvx-chip_alc
druc-alc:   druc-chip_alc
```

Where variables have the following meaning:

Variable	Usage
CORE	The name of the *core* model
CHIP	The stem of the *chip* model. It is declined in two versions, one for ALLIANCE (suffix _alc) and one for CORIOLIS (suffix _crl). This is needed because the two core uses different sets of pads.
GENERATE_CORE_VST	Tells if the rules to generate the core has to be included. If set to No, then the core *must* be present and will be considered as a primary file.
USE_CLOCKTREE	Adds a clock-tree to the design (CORIOLIS).
USE_MOSIS	Tells whether or not use the MOSIS technology.
USE_DEBUG	Activate debug support on cgt.

CORIOLIS Configuration Files

Unlike ALLIANCE which is entirely configured through environement variables or system-wide configuration file, CORIOLIS uses configuration files in the current directory. They are present for each bench:

- <cwd>/.coriolis2/techno.py : Select which symbolic and real technology to use.
- <cwd>/.coriolis2/settings.py : Override for any system configuration, except for the technology.

CORIOLIS and Clock Tree Generation

When CORIOLIS is used, it create a clock tree which modificate the original netlist. The new netlist, with a clock tree, has a postfix of _clocked.

Note

Trans-hierarchical Clock-Tree. As CORIOLIS do not flatten the designs it creates, not only the top-level netlist is modificated. All the sub-blocks connected to the master clock are also duplicateds, whith the relevant part of the clock-tree included.

RHEL6 and Clones

Under RHEL6 the developpement version of CORIOLIS needs the devtoolset-2. rules.mk tries, based on uname to switch it on or off.

Benchmarks Special Notes

alliance-run

This benchmark comes mostly with it's own rules and do not uses the ones supplieds by rules.mk. It uses only the top-level configuration variables.

It a sligtly modified copy of the alliance-run found in the ALLIANCE package (modification are all in the Makefile). It build an AM2901, but it is splitted in a control and an operative part (data-path). This is to also check the data-path features of ALLIANCE.

And lastly, it provides a check for the CORIOLIS encapsulation of ALLIANCE through PYTHON wrappers. The support is still incomplete and should be used only by very experienced users. See the demo* rules.

Tools & Scripts

One script to run them all: `go.sh`

To call all the bench's `Makefile` sequentially and execute one or more rules on each, the small script utility `go.sh` is available. Here are some examples:

```
dummy@lepka:bench$ ./bin/go.sh clean
dummy@lepka:bench$ ./bin/go.sh lvx
```

Command Line `cgt`: `doChip.py`

As a alternative to `cgt`, the small helper script `doChip.py` allows to perform all the P&R tasks, on an stand-alone block or a whole chip.

Pad Layout Converter `px2mpx.py`

The `px2mpx.py` script convert pad layout from the `pxlib` (ALLIANCE dummy technology) into `mpxlib` (MOSIS compliant symbolic technology).

Basically it multiplies all the coordinate by two as the source technology is 1µ type and the target one a 2µ. In addition it performs some adjustement on the wire extension and minimal width and the blockage sizes.

As it is a one time script, it is heavily hardwired, so before using it do not forget to edit it to suit your needs.

The whole conversion process is quite tricky as we are cheating with the normal use of the software. The steps are as follow:

1. Using the ALLIANCE dummy technology and in an empty directory, run the script. The layouts of the converted pads (`*_mpx.ap`) will be created.
2. In a second directory, this time configured for the MOSIS technology (see `.coriolis2_techno.conf`) copy the converted layouts. In addition to the layouts, this directory **must also contain** the behavioral description of the pads (`.vbe`). Otherwise, you will not be able to see the proper layout.
3. When you are satisfied with the new layout of the pads, you can copy them back in the official pad cell library.

Note

How Coriolis Load Cells. Unlike in ALLIANCE, CORIOLIS maintain a much tighter relationship between physical and logical (structural or behavioral) views. The loading process of a cell try *first* to load the logical view, and if found, keep tab of the directory it was in. *Second* it tries to load the physical view from the same directory the logical view was in. If no logical view is found, only the physical is loaded.
Conversely, when saving a cell, the directory it was loaded from is kept, so that the cell will be overwritten, and not duplicated in the working directory as it was in ALLIANCE.
This explains why the behavioral view of the pad is needed in the directory the layouts are put into. Otherwise you would only see the pads of the system library (if any).

CADENCE Support

To perform comparisons with CADENCE EDI tools (i.e. `encounter` NANOROUTE), some benchmarks have a sub-directory `encounter` holding all the necessary files. Here is an example for the design named `<fpga>`.

encounter directory	
File Name	**Contents**
fpga_export.lef	Technology & standard cells for the design
fpga_export.def	The design itself, flattened to the standard cells.
fpga_nano.def	The placed and routed result.
fpga.tcl	The TCL script to be run by encounter

The LEF/DEF file exported or imported by Coriolis are *not* true physical files. They are pseudo-real, in the sense that all the dimensions are directly taken from the symbolic with the simple rule `1 lambda = 1 micron`.

Note

LEF/DEF files: Coriolis is able to import/export in those formats only if it has compiled against the SI2 relevant libraries that are subjects to specific license agreements. So in case we don't have access to thoses we supplies the generated LEF/DEF files.

The `encounter` directory contains the LEF/DEF files and the TCL script to be run by `encounter`:

```
ego@home:encounter> . ../../etc/EDI1324.sh
ego@home:encounter> encounter -init ./fpga.tcl
```

Example of TCL script for `encounter`:

```
set_global _enable_mmmc_by_default_flow      $CTE::mmmc_default
suppressMessage ENCEXT-2799
win
loadLefFile fpga_export.lef
loadDefFile fpga_export.def
floorPlan -site core -r 0.998676319592 0.95 0.0 0.0 0.0 0.0
getIoFlowFlag
fit
setDrawView place
setPlaceMode -fp false
placeDesign
generateTracks
generateVias
setNanoRouteMode -quiet -drouteFixAntenna 0
setNanoRouteMode -quiet -drouteStartIteration 0
setNanoRouteMode -quiet -routeTopRoutingLayer 5
setNanoRouteMode -quiet -routeBottomRoutingLayer 2
setNanoRouteMode -quiet -drouteEndIteration 0
setNanoRouteMode -quiet -routeWithTimingDriven false
setNanoRouteMode -quiet -routeWithSiDriven false
routeDesign -globalDetail
global dbgLefDefOutVersion
set dbgLefDefOutVersion 5.7
defOut -floorplan -netlist -routing fpga_nano.def
```