

ALLIANCE & CORIOLIS Checker Toolkit

Contents

ALLIANCE & CORIOLIS Checker Toolkit	1
Toolkit Purpose	1
Toolkit Contents	1
Benchmark Makefiles	2
CORIOLIS Configuration Files	3
CORIOLIS and Clock Tree Generation	4
RHEL6 and Clones	4
Benchmarks Special Notes	4
alliance-run	4
Libraries Makefiles	4
Tools & Scripts	5
One script to run them all: go.sh	5
Command Line cgt: doChip.py	5
Pad Layout Converter px2mpx.py	5
Yagle Cell Extraction Script extractCell.tcl	6
CADENCE Support	6
Technologies	7

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 committing and pushing them in their central repositories. A change will be considered as validated when all the developpers can run successfully all the benches in their respective environments.

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

Toolkit Contents

The toolkit provides:

- Eleven benchmark designs:

Design	Technology	Cell Libraries
adder	MOSIS	nsxlib, mpplib, msplib
AM2901 (standard cells)	ALLIANCE dummy	sxlib, pxlib
AM2901 (datapath)	ALLIANCE dummy	sxlib, dp_sxlib, pxlib
alliance-run (AM2901)	ALLIANCE dummy	sxlib, dp_sxlib, padlib
CPU	MOSIS	nsxlib, mpplib, msplib
SNX	MOSIS	nsxlib, mpplib, msplib
MIPS (microprogrammed)	ALLIANCE dummy	sxlib, dp_sxlib, rf2lib
MIPS (pipeline)	ALLIANCE dummy	sxlib, dp_sxlib, rf2lib
MIPS (pipeline+chip)	ALLIANCE dummy	sxlib, dp_sxlib, rf2lib, pxlib
FPGA (Moc4x4_L4C12)	ALLIANCE dummy	sxlib
ISPD05 (bigblue1)	Aucune	Genérée à la volée.

- Three cell libraries.

All those 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.

- `nsxlib` : 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 provides some or all of the following targets. If the place and route stage of a bench can be done by both CORIOLIS and ALLIANCE an `alliance/` subdirectory will be present.

CORIOLIS	<code>layout</code>	The complete symbolic layout of the design (P&R).
	<code>gds</code>	Generate the real layout (GDSII)
	<code>druc</code>	Symbolic layout checking
	<code>lvx</code>	Perform LVS.
	<code>graal</code>	Launch <code>graal</code> in the Makefile 's environnement
	<code>dreal</code>	Launch <code>dreal</code> in the Makefile 's environnement, and load the <code>gds</code> file of the design.
	<code>view</code>	Launch <code>cgt</code> and load the design (chip)
	<code>cgt</code>	Launch <code>cgt</code> in the Makefile 's environnement

A top Makefile in a bench directory must looks like:

```

BOOMOPT = -A
BOOGOPT =
LOONOPT =
NSL2VHOPT = -vasy
LIBRARY_FAMILY = nsxlib
USE_SYNTHESIS = Yes
USE_CLOCKTREE = No
USE_DEBUG = No

NETLISTS = cla16      \
           incl6      \
           reg4        \
           type_dec    \
           alu16_model \
           snx_model

include ../etc/rules.mk

lvx:      lvx-chip_kite
druc:     druc-chip_kite
view:     cgt-chip_kite

layout:   chip_kite.ap
gds:      chip_kite.gds

```

Where variables have the following meaning:

Variable	Usage
NETLISTS	The list of <i>netlists</i> that are requireds to perform the place and route stage. The files must we given <i>without</i> extension. According to the value of <code>USE_SYNTHESIS</code> they are user supplied or generated. In the later case, be aware that calling the <code>clean</code> target will remove the generated files.
LIBRARY_FAMILY	Tells which library set to use. Legal values are <code>sxlib</code> (default) and <code>nsxlib</code> (MOSIS technology).
USE_SYNTHESIS	If set to <code>Yes</code> , then the files given in the <code>NETLISTS</code> variables will be synthetised from the reference <code>vhdl</code> or <code>ns1</code> description, if this tool is available. Note that the <code>clean</code> will remove all generated files.
USE_CLOCKTREE	Adds a clock-tree to the design (CORIOLIS).
USE_DEBUG	Activate debug support on <code>cgt</code> .

CORIOLIS Configuration Files

Unlike ALLIANCE which is entirely configured through environnement 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.

CORIORIS and Clock Tree Generation

When CORIORIS 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 CORIORIS do not flatten the designs it creates, not only the top-level netlist is modificado. All the sub-blocks connected to the master clock are also duplicated, with the relevant part of the clock-tree included.

RHEL6 and Clones

Under RHEL6 the developpement version of CORIORIS 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 supplied by `rules.mk`. It uses only the top-level configuration variables.

It a slightly 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 CORIORIS encapsulation of ALLIANCE through PYTHON wrappers. The support is still incomplete and should be used only by very experienced users. See the `demo*` rules.

Libraries Makefiles

The `bench/etc/rules.mk` provides rules to perform the check of a library as a whole or cell by cell. To avoid too much clutter in the library directory, all the intermediate files generated by the verification tools are kept in a `./check/` subdirectory. Once a cell has been validated, a `./check/<cell>.ok` is generated too prevent it to be checked again in subsequent run. If you want to force the recheck of the cell, do not forget to remove this file.

The checking procedure:

- DRC with `druc`.
- Formal proof between the layout and the behavioral description. This is a somewhat long chain of tools:
 1. `cougar`, extract the spice netlist (`.spi`).
 2. `yagle`, rebuild a behavioral description (`.vhd`) from the spice netlist.
 3. `vasy`, convert the `.vhd` into a `.vbe` (Alliance VHDL subset for behavioral descriptions).
 4. `proof`, perform the formal proof between the reference `.vbe` and the extracted one.

Rule or File	Action
check-lib	Validate every cell of the library
clean-lib-tmp	Remove all intermediate files in the <code>./check</code> subdirectory except for the <code>*.ok</code> ones. That is, cells validated will not be rechecked.
clean-lib	Remove all files in <code>./check</code> , including <code>*.ok</code>
<code>./check/<cell>.ok</code>	Use this rule to perform the individual check of <code><cell></code> . If the cell is validated, a file of the same name will be created, preventing the cell to be checked again.

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 ajustement 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).

Yagle Cell Extraction Script `extractCell.tcl`

A small TCL script for `avt_shell` to perform a cell extraction with `yagle`. Cell which name starts by `sff` will be treated as D flip-flop.

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
<code>fpga_export.lef</code>	Technology & standard cells for the design
<code>fpga_export.def</code>	The design itself, flattened to the standard cells.
<code>fpga_nano.def</code>	The placed and routed result.
<code>fpga.tcl</code>	The TCL script to be run by <code>encounter</code>

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 been compiled against the S12 relevant libraries that are subjects to specific license agreements. So in case we don't have access to those 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
```

Technologies

We provides configuration files for the publicly available MOSIS technology SCN6M_DEEP.

- ./bench/etc/scn6m_deep_09.rds, RDS rules for symbolic to real transformation.
- ./bench/etc/scn6m_deep.hsp, transistor spice models for yagle.

References:

- [MOSIS Scalable CMOS \(SCMOS\)](#)
- [MOSIS Wafer Acceptance Tests](#)

Technical informations:

MOSIS WAFER ACCEPTANCE TESTS

RUN: T92Y (MM_NON-EPI_THK-MTL)
 TECHNOLOGY: SCN018

VENDOR: TSMC
 FEATURE SIZE: 0.18 microns

Run type: DED

INTRODUCTION: This report contains the lot average results obtained by MOSIS from measurements of MOSIS test structures on each wafer of this fabrication lot. SPICE parameters obtained from similar measurements on a selected wafer are also attached.

COMMENTS: DSCN6M018_TSMC

TRANSISTOR PARAMETERS	W/L	N-CHANNEL	P-CHANNEL	UNITS
MINIMUM	0.27/0.18			
Vth		0.50	-0.49	volts
SHORT	20.0/0.18			
Idss		572	-276	uA/um
Vth		0.52	-0.49	volts
Vpt		4.7	-5.2	volts
WIDE	20.0/0.18			
Ids0		20.8	-15.2	pA/um
LARGE	50/50			
Vth		0.42	-0.41	volts
Vjbkd		3.7	-4.4	volts
Ijlk		<50.0	<50.0	pA
K' (Uo*Cox/2)		171.0	-37.0	uA/V^2
Low-field Mobility		406.07	87.86	cm^2/V*s

COMMENTS: Poly bias varies with design technology. To account for mask bias use the appropriate value for the parameters XL and XW in your SPICE model card.

Design Technology	XL (um)	XW (um)
-----	-----	-----
SCN6M_DEEP (lambda=0.09)	0.00	-0.01
thick oxide	0.00	-0.01
SCN6M_SUBM (lambda=0.10)	-0.02	0.00
thick oxide	-0.02	0.00

FOX TRANSISTORS	GATE	N+ACTIVE	P+ACTIVE	UNITS
Vth	Poly	>6.6	<-6.6	volts

PROCESS PARAMETERS	N+	P+	POLY	N+BLK	PLY+BLK	M1	M2	UNITS
Sheet Resistance	7.0	8.1	8.3	59.5	306.6	0.08	0.08	ohms/sq
Contact Resistance	8.3	8.8	8.1				4.83	ohms

Gate Oxide Thickness	41							angstrom
PROCESS PARAMETERS	M3	POLY_HRI	M4	M5	M6	N_W	UNITS	
Sheet Resistance	0.08		0.08	0.07	0.01	951	ohms/sq	
Contact Resistance	9.74		15.36	21.50	23.45		ohms	

COMMENTS: BLK is silicide block.

CAPACITANCE PARAMETERS	N+	P+	POLY	M1	M2	M3	M4	M5	M6	R_W	D_N_W	M5P	N_W	UNIT
Area (substrate)	969	1234	101	34	14	9	7	5	4		129		130	aF/u
Area (N+active)			8517	53	20	14	11	9	8					aF/u
Area (P+active)			8275											aF/u
Area (poly)				64	17	10	7	5	4					aF/u
Area (metal1)					35	14	9	6	5					aF/u
Area (metal2)						36	14	9	6					aF/u
Area (metal3)							37	14	9					aF/u
Area (metal4)								36	14					aF/u
Area (metal5)									35			1039		aF/u
Area (r well)	953													aF/u
Area (d well)										562				aF/u
Area (no well)	140													aF/u
Fringe (substrate)	196	229		53	36	29	24	21	19					aF/u
Fringe (poly)				68	38	29	23	19	18					aF/u
Fringe (metal1)					49	34		22	20					aF/u
Fringe (metal2)						45	35	27	23					aF/u
Fringe (metal3)							54	34	30					aF/u
Fringe (metal4)								63	43					aF/u
Fringe (metal5)									66					aF/u

CIRCUIT PARAMETERS		UNITS
Inverters	K	
Vinv	1.0	0.74 volts
Vinv	1.5	0.79 volts
Vol (100 uA)	2.0	0.08 volts
Voh (100 uA)	2.0	1.62 volts
Vinv	2.0	0.83 volts
Gain	2.0	-24.67
Ring Oscillator Freq.		
D1024_THK (31-stg, 3.3V)	302.91	MHz
DIV1024 (31-stg, 1.8V)	377.13	MHz
Ring Oscillator Power		
D1024_THK (31-stg, 3.3V)	0.07	uW/MHz/gate
DIV1024 (31-stg, 1.8V)	0.02	uW/MHz/gate

COMMENTS: DEEP_SUBMICRON