RISC-V design using FOSS

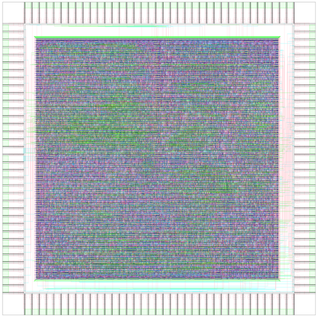
Jean-Paul CHAPUT
LIP6, Sorbonne Université
CIAN Team

Marie-Minerve LOUËRAT, Roselyne CHOTIN, Jean-Paul CHAPUT, Adrian SATIN
Jean-Paul.Chaput@lip6.fr

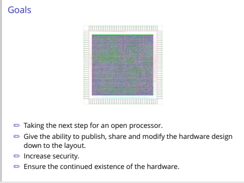
Paris, October 2nd, 2019

Plan

Goals



- ✏ Taking the next step for an open processor.
- ✏ Give the ability to publish, share and modify the hardware design down to the layout.
- ✏ Increase security.
- ✏ Ensure the continued existence of the hardware.



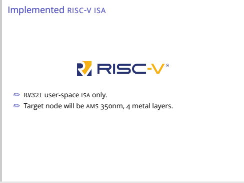
Goals

- Taking the next step for an open processor.
- Give the ability to publish, share and modify the hardware design down to the layout.
- Increase security.
- Ensure the continued existence of the hardware.

- It seems only natural for a free and open processor to be built using fre tools.
- By checking the layout, we can better detect hardware trojan and ensur the chip is exactly what it is.
- We expect FOSS to have the same effect of community building.
- NASA was forced to scavenge 8086 on eBay for the space shuttle arounc 2002.

Implemented RISC-V ISA

- `RV32I` user-space ISA only.
- Target node will be AMS 350nm, 4 metal layers.

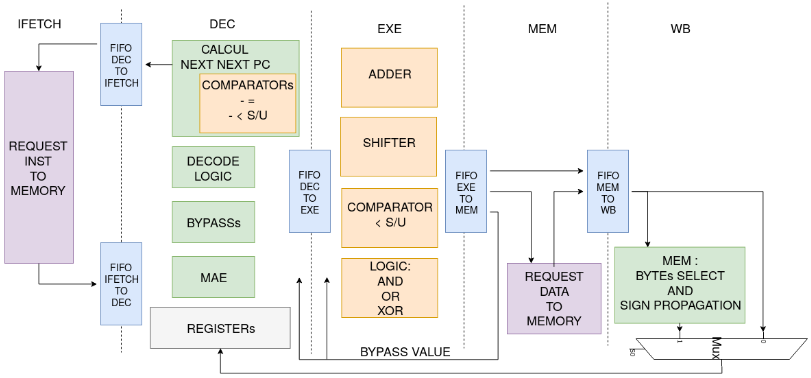


Implemented RISC-V ISA

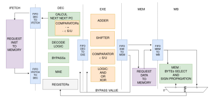
- RV32I user-space ISA only.
- Target node will be AMS 350nm, 4 metal layers.

- We choose to start with as small possible a component. Always better for debugging...
- Use of a *mature node* so not too expensive and not too much features t
implement in the tools.

Architecture of our RISC-V

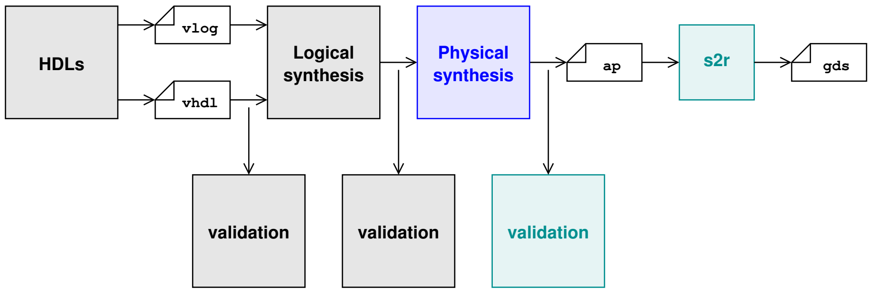


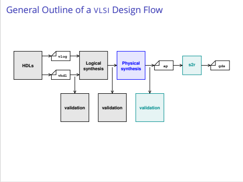
Architecture of our RISC-V



- A simple five stage pipeline.
- Based on our experience over the design of the MIPS R3000.
- I'm not the architect, so I couldn't answer tricky design questions...

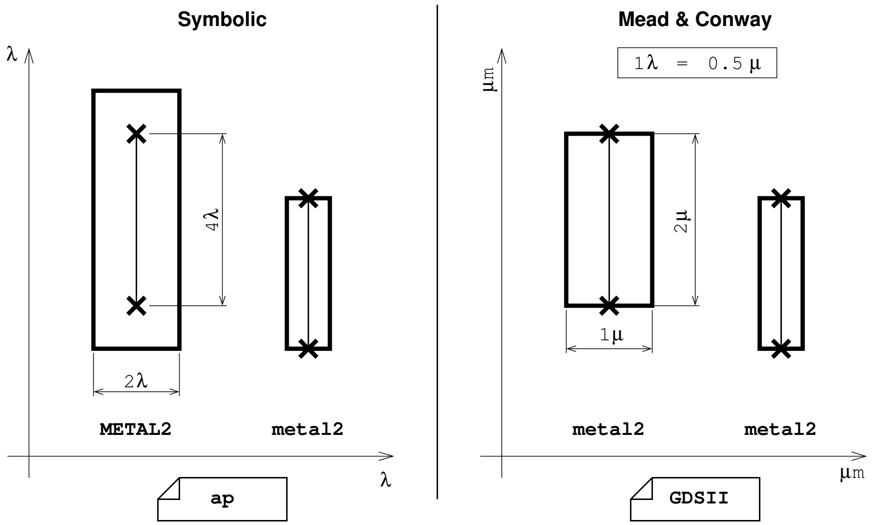
General Outline of a VLSI Design Flow





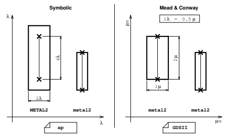
- A word about HDL languages, for now we did it the old way in VHDL. CHISEL and SPINALHDL have a logic more suited for programmers than computer scientists or electronic people. MIGEN is better and written in Python but do not generate VHDL (yet ?). All of them are difficult to extend if an unsupported feature occurs.
- The LIP6 contribution to the flow is mostly focused on the physical design stage.
- More tools exists for the stage before because they can also target FPGA
- The equal size of the boxes do not reflect on the hardness of each stage...
- The last step, with S2R will be explained shortly thereafter.

A brief history of *symbolic layout* (1/2)



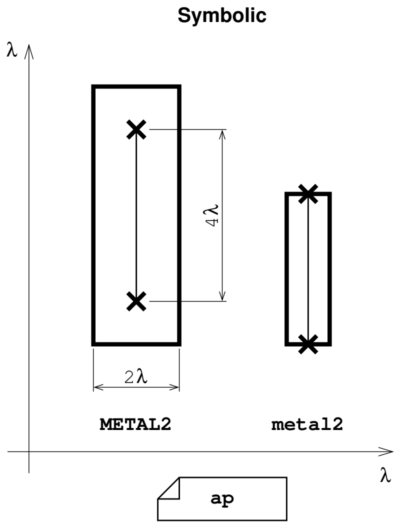
└A brief history of *symbolic layout* (1/2)

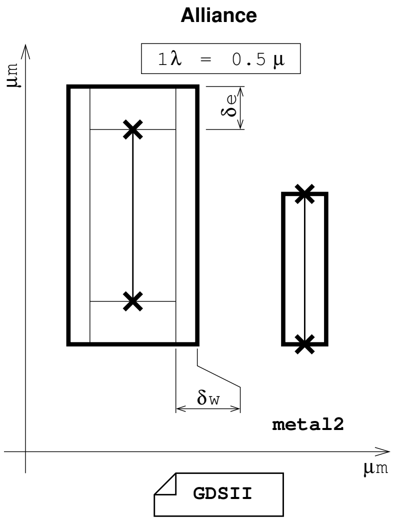
A brief history of *symbolic layout* (1/2)



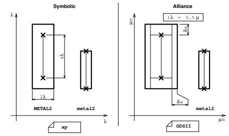
- Invented in 1980 by MEAD & CONWAY. Draw your layout using a special dimension unit, the λ. Then scale to the target node. Assume that the shrink rate is almost the same for all layers.
- Designed to cross the boundaries of foundries and nodes.
- Allows a drastic reduction in the number of design rules.
- Main drawback : the area loss, about 10%.
- At the origin of MOSIS.
- Simple shrink finally proven a little bit too rigid.

A brief history of *symbolic layout* (2/2)



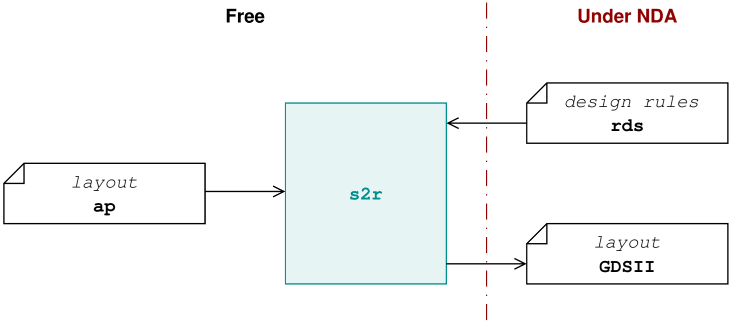


A brief history of *symbolic layout* (2/2)

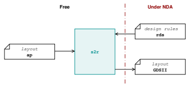


- Refined by BULL, to give ALLIANCE symbolic. Add cap and width extensions to give more slack in the transformation process.
- BULL is fabless and did not want to be tied to one foundry but didn’t want to develop twice it’s designs.
- Has a big advantage unforeseen at the time, it is NDA free but still very close the the real layout.
- So, layout **is publishable** and can be **verified** against what comes back from the foundry. Can be critical for security.

symbolic vs. real layout

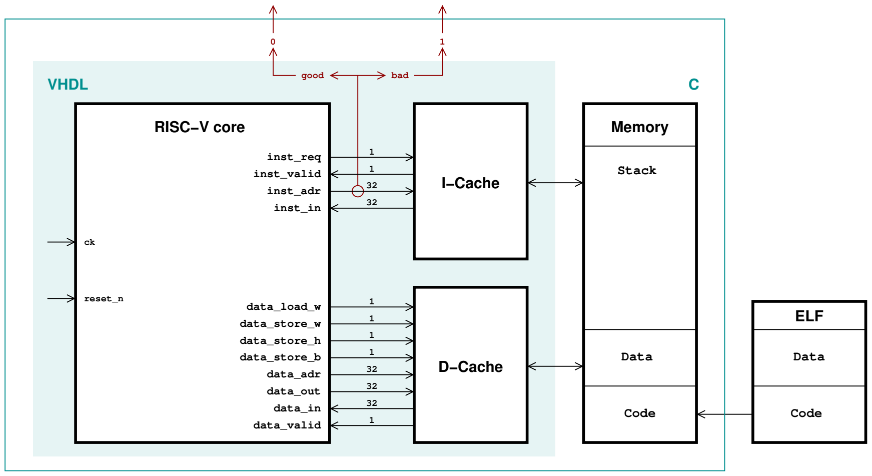


symbolic vs. real layout

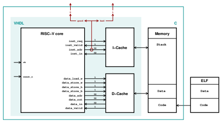


- The translation from symbolic to real layout is ensured by the S2R program. It needs a parametrisation for the target node.
- We are working for a way to provide this file (and some more) to other users through the MYCMP service.
- We keep as much as possible of the toolchain on the «left side»...
- This is more difficult when it comes down to timing informations and extraction.
- The other way around NDA has been taken by FREEPDK, which develop fake but realistic design kits. Still they are made mainly for commercial tools.

Simulation Plateform with GHDL

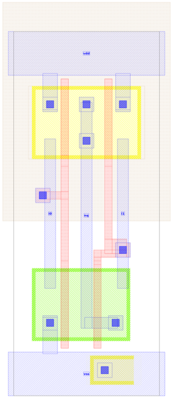


Simulation Plateform with GHDL

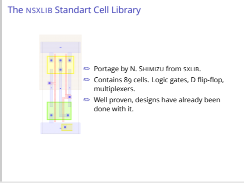


- We provide the RISC-V VHDL model with a simple access to data and instructions. `D-Cache` and `I-Cache` are a misnomer, they are just VHDL proxies for code and datas stored in the C part of the platform.
- Code and datas are loaded at runtime from an ELF file generated from the tests provided by the RISC-V fondation.
- The whole platform is compiled (both VHDL and C) as a binary.
- It is also used to validate the description after synthesis and place and route.

The NSXLIB Standart Cell Library



- Portage by N. SHIMIZU from SXLIB.
- Contains 89 cells. Logic gates, D flip-flop, multiplexers.
- Well proven, designs have already been done with it.

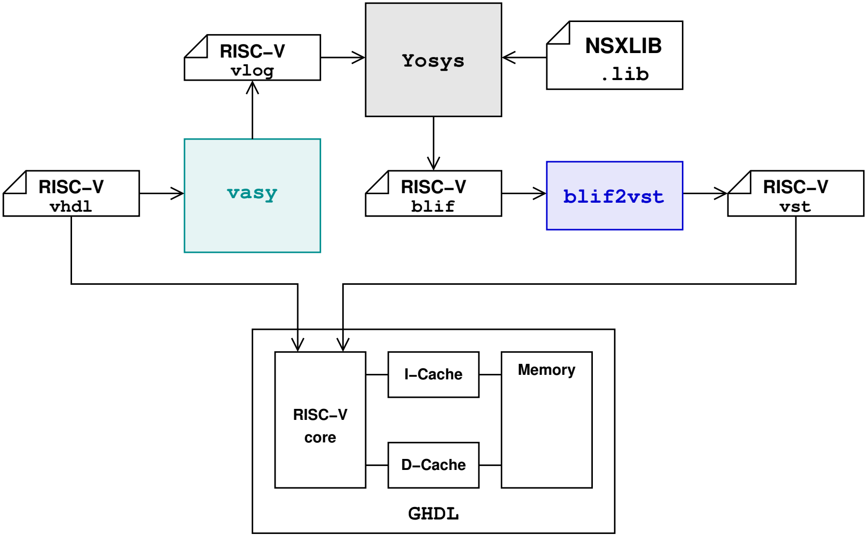


The NSXLIB Standart Cell Library

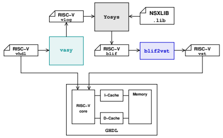
- Portage by N. SHIMIZU from SXLIB.
- Contains 89 cells. Logic gates, D flip-flop, multiplexers.
- Well proven, designs have already been done with it.

- The symbolic rules needed to be adapted to better fit the deep submicron technologies (45nm and below).
- Automatic characterization procedure with a fake technology to generate a liberty file `.lib` (SYNOPSYS).

Logical Synthesis

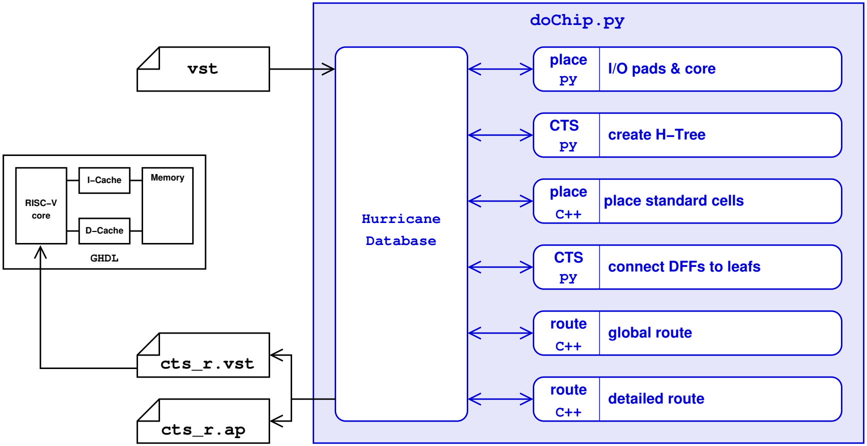


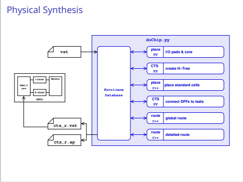
Logical Synthesis



- We use YOSYS to perform the logical synthesis.
- This is a nightmare of format translation between, VHDL, Verilog, `blif` and `vst`... (give some details about those formats).
- Color code: cian for ALLIANCE, blue for CORIOLIS.
- The GHDL platform is used to check that the generated netlist (RTL) is consistent with the behavioral description.

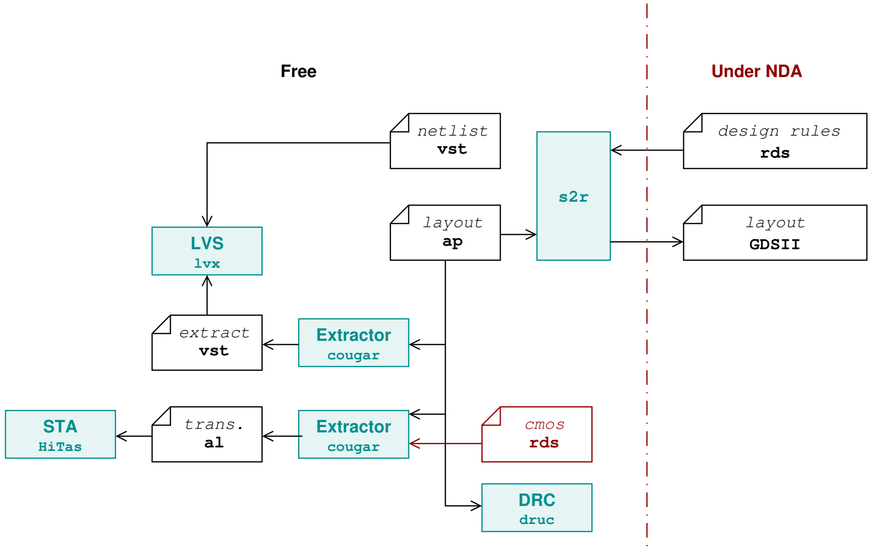
Physical Synthesis




Physical Synthesis

- Scan-path is inserted in the netlist just after the logical synthesis. It chains the FIFOS, the DECOD & and IFECTH state. Not the register file.
- All tools works *in memory* using the HURRICANE database, this allow a tight integration between the tools. In particular, notice the interleaving between CTS and standard cell placement, this way we can easily connect the DFFs to the nearest clock-tree leaf.
- Another feature of importance is seamless integration between Python and C++ parts. Here again, we can mix them almost any way we want.
- This is so true that there isn't event a CORIOLIS binary. It is only Python scripts that we can tailor to whatever we want.
- As we modify the netlist when inserting the clock tree, we check again with the GHDL platform.
- The placer is analytical, based on SIMPL.

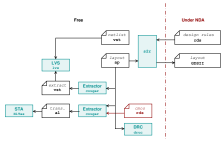
Validation


Free
Under NDA
netlist vst
design rules rds
s2r
layout ap
layout GDSII
LVS lvx
extract vst
Extractor cougar
STA HiTas
trans. al
Extractor cougar
cmos rds
DRC druc

2019-09-29

RISC-V design using FOSS
└─Description of the Design Flow
└─Validation

Validation



- The STA `HiTas` is an old but industry proven tool.
- For the `cougar` extractor to work it needs technological informations. W
  supply a fake technology.

Design Flow Automation

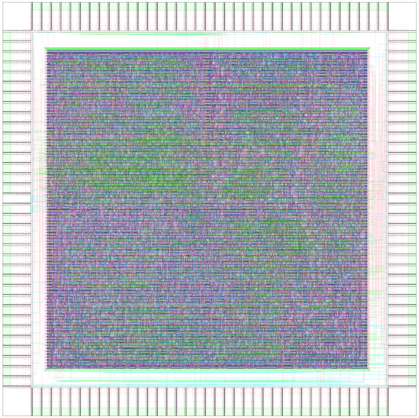
- ✏ Alliance Check Toolkit provides a set of GNU `Makefiles` to fully automate the flow.
- ✏ Only one top level `Makefile` is needed to build a design.
- ✏ A regression suite for the tools.
- ✏ Various example designs of blocks or whole chips.

Design Flow Automation

- Alliance Check Toolkit provides a set of GNU `Makefiles` to fully automate the flow.
- Only one top level `Makefile` is needed to build a design.
- A regression suite for the tools.
- Various example designs of blocks or whole chips.

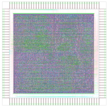
- Alliance Check Toolkit contains a lot of paraphernalia.
- We are at the limit to what can be done with even evolved GNU `Makefiles`. In the future we may write an integrated Python script.

The Chip



- 11K gates.
- 144 I/O pad.
- Chip is core limited.
- Size is 5×5 mm ($25 mm^2$)

The Chip



- 11K gates.
- 144 I/O pad.
- Chip is core limited.
- Size is 5×5 mm (25mm^2)

- Precisely 11348 gates with YOSYS 0.7.
- As we are not finished yet, those results will slightly evolve still.
- The core is symbolic but we use the I/O pad supplied by AMS. This is a part that we cannot make symbolic due to foundry constraints.

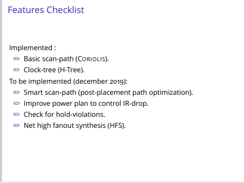
Features Checklist

Implemented :

- ✏ Basic scan-path (Coriolis).
- ✏ Clock-tree (H-Tree).

To be implemented (december 2019):

- ✏ Smart scan-path (post-placement path optimization).
- ✏ Improve power plan to control IR-drop.
- ✏ Check for hold-violations.
- ✏ Net high fanout synthesis (HFS).



Features Checklist

Implemented :

- Basic scan-path (Coriolis).
- Clock-tree (H-Tree).

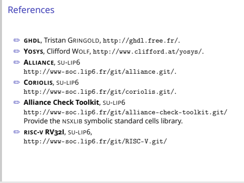
To be implemented (december 2019):

- Smart scan-path (post-placement path optimization).
- Improve power plan to control IR-drop.
- Check for hold-violations.
- Net high fanout synthesis (HFS).

- Of courses, we are talking about features besides classic place and rout
- Our middle term goal is to implement features allowing us to use smaller and smaller nodes.

References

- **GHDL**, Tristan GRINGOLD, http://ghdl.free.fr/.
- **YOSYS**, Clifford WOLF, http://www.clifford.at/yosys/.
- **ALLIANCE**, SU-LIP6
  http://www-soc.lip6.fr/git/alliance.git/.
- **CORIOLIS**, SU-LIP6
  http://www-soc.lip6.fr/git/coriolis.git/.
- **Alliance Check Toolkit**, SU-LIP6
  http://www-soc.lip6.fr/git/alliance-check-toolkit.git/
  Provide the NSXLIB symbolic standard cells library.
- **RISC-V RV32I**, SU-LIP6,
  http://www-soc.lip6.fr/git/RISC-V.git/



References

- **GHDL**, Tristan GRINGOLD, `http://ghdl.free.fr/`.
- **YOSYS**, Clifford WOLF, `http://www.clifford.at/yosys/`.
- **ALLIANCE**, SU-LIP6 `http://www-soc.lip6.fr/git/alliance.git/`.
- **CORIOLIS**, SU-LIP6 `http://www-soc.lip6.fr/git/coriolis.git/`.
- **Alliance Check Toolkit**, SU-LIP6 `http://www-soc.lip6.fr/git/alliance-check-toolkit.git/` Provide the NSXLIB symbolic standard cells library.
- **RISC-V RV32I**, SU-LIP6, `http://www-soc.lip6.fr/git/RISC-V.git/`

Pas de notes pour ce transparent.

And now, let’s have a demo...

And now, let's have a demo...

- Run the logical synthesis stage on the console, then switch to graphical mode for the physical synthesis.
- Commands:

```
ego@home:RISC-V> cd PlaceAndRoute
ego@home:PlaceAndRoute> make scan
ego@home:PlaceAndRoute> cgt -V -cell=riscv_core_scan
```

  Generate the chip from the pads using AMS pads.
  Place chip (with clock tree).
  Do not forget to go down the corona.
  Global then detail route the corona / core.
- Comment on Etesian analytical placer (force directed).