UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Elpis

From simulation to fabrication

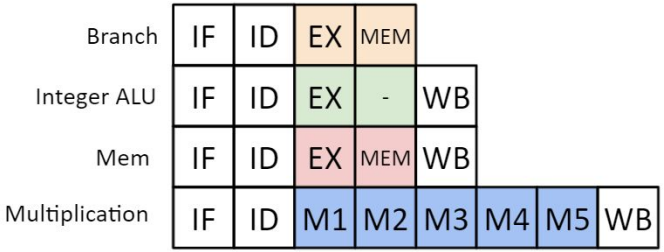
Rodrigo Huerta Gañán (rodrigo@ac.upc.edu)
Aurora Tomás Berjaga (aurora@ac.upc.edu)

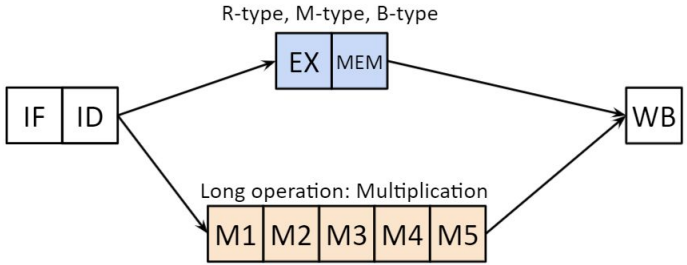
ARCO - 23 November 2021

Table of contents

1. Processor Architecture (PA) work

1. Baseline





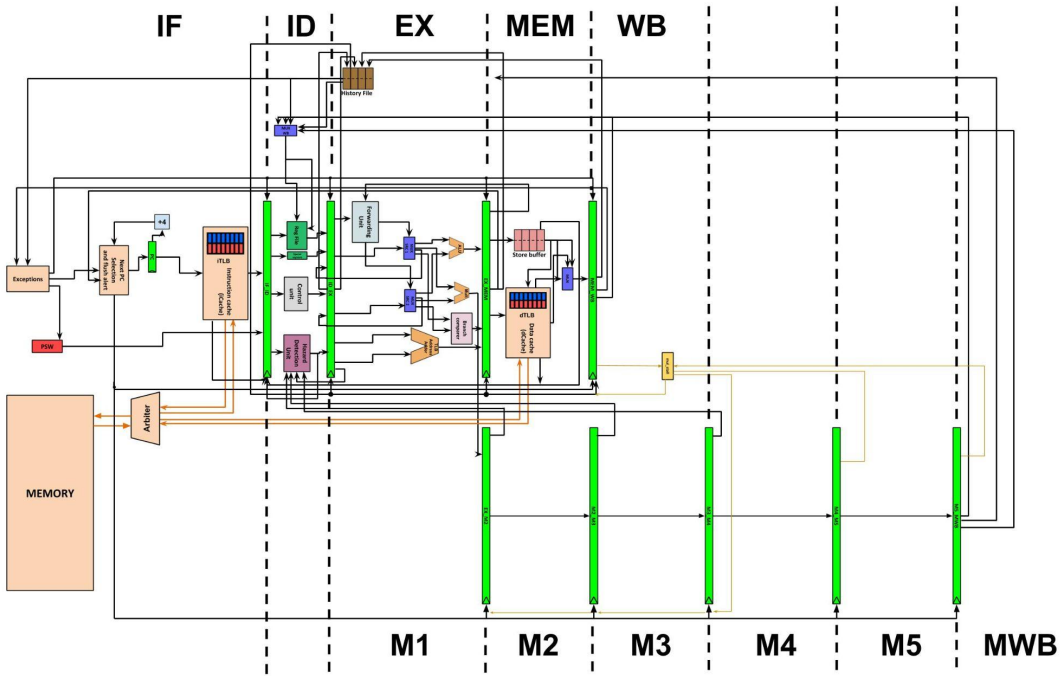
Elpis core

- ✓ Total custom core implemented from scratch in Verilog
- ✓ Based on RISC-V mixed with some MIPS ideas, and customized instructions
- ✓ 5-stage (IF, ID, EX, MEM, WB) pipelined, in-order and multicycle processor
- ✓ Support for different type of instructions and pipeline lengths

1. Baseline

Microarchitectural features:

- Full set of bypasses
- Main memory (5 cycles delay)
- L1 iCache and L1 dCache
- Memory arbiter
- iTLB and dTLB
  - **Virtual memory support**
- History File
  - **In-order exceptions support**
- Store Buffer
  - Solve RAW dependencies
  - **OoO writes**



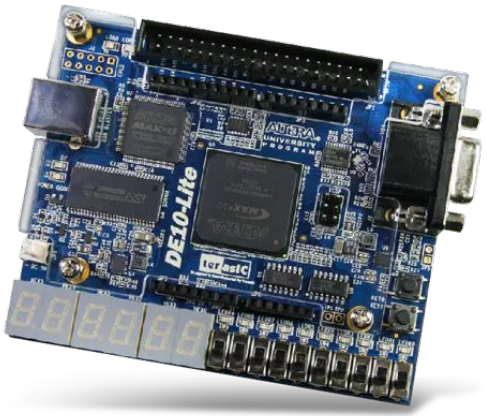
Basic instructions: `ADD, SUB, MUL, LDB, LDW, STB, STW, BEQ, JUMP`

2. Multiprocessors Architecture (MA) work

2. FPGA Synthesis process

Main issues:

- Assignation to regs from different `always` blocks
- Mix of blocking and non-blocking assignations
- Use of '`x` values
- Non-desired inferred latches
- Non inferred RAM blocks
  - Use of SRAM bits for RAM and caches instead of Logic Elements
    - Complete re-coding of RAM and TLB
    - Big modifications to caches
- No timing requirements accomplished
  - Use of a PLL to downclock the original FPGA clock



Altera De10-Lite

3. Core extensions

```
aurora@LAPTOP-0EIUAJ1E:/mnt/c/git/Elpis-Compiler$ python3 ecc.py print.asm
00002b83        //      0x0     lw x23, 0(x0)
00402083        //      0x4     lw x1, 4(x0)
fff08093        //      0x8     addi x1, x1, -1
00000033        //      0xc     add x0, x0, x0
fe009ce3        //      0x10    bne x1, x0, -8
00100193        //      0x14    addi x3, x0, 1
000ba103        //      0x18    lw x2, 0(x23)
```

Elpis compiler

- ✓ **Correction** of detected **bugs**
- ✓ **Enabling I/O**
  - Interaction with switches and hexadecimal displays
- ✓ **More complete ISA**
  - Incorporation of new instructions: `addi, subi, or, ori, and, andi, xor, xori, sll, srl, sra, slli, srli, srai, bge, blt, bne, movr*, ecall*, read*, print`
- ✓ Creation of a **compiler** in Python
  - Translates from assembly language to machine code
- ✓ Incorporation and use of **Unit Testing**

```
==== Summary ========================================
pass lib.tb_bypass2.all              (5.1 seconds)
pass lib.tb_bypass1.all              (5.1 seconds)
pass lib.tb_bypass3.all              (5.1 seconds)
pass lib.tb_io_print.all             (2.0 seconds)
pass lib.tb_io_read.all              (1.7 seconds)
pass lib.tb_io_readPrint.all         (1.7 seconds)
pass lib.tb_memAcc_bytes1.all        (1.7 seconds)
pass lib.tb_memAcc_bytes2.all        (1.7 seconds)
pass lib.tb_memAcc_bytes3.all        (1.7 seconds)
pass lib.tb_memAcc_word.all          (2.0 seconds)
pass lib.tb_mul1.all                 (2.0 seconds)
pass lib.tb_mul2.all                 (2.0 seconds)
pass lib.tb_mul3.all                 (2.2 seconds)
pass lib.tb_mul4.all                 (1.9 seconds)
pass lib.tb_mul5.all                 (2.1 seconds)
pass lib.tb_mul6.all                 (2.2 seconds)
pass lib.tb_bench2.all               (17.7 seconds)
pass lib.tb_mul7.all                 (2.3 seconds)
pass lib.tb_mul8.all                 (2.3 seconds)
pass lib.tb_storeBuffer_sb1.all      (2.3 seconds)
pass lib.tb_storeBuffer_sb2.all      (2.4 seconds)
pass lib.tb_storeBuffer_sb3.all      (2.4 seconds)
pass lib.tb_storeBuffer_sb4.all      (2.2 seconds)
pass lib.tb_multi_shared.all         (1.9 seconds)
pass lib.tb_tlb.all                  (2.5 seconds)
pass lib.tb_multi_bench1_and_2.all   (31.7 seconds)
pass lib.tb_bench3.all               (185.4 seconds)
=====================================================
pass 39 of 39
=====================================================
Total time was 319.7 seconds
Elapsed time was 185.5 seconds
=====================================================
All passed!
```

Unit Testing

3. Core extensions

Once Elpis was synthesized, the compilation report obtained was:

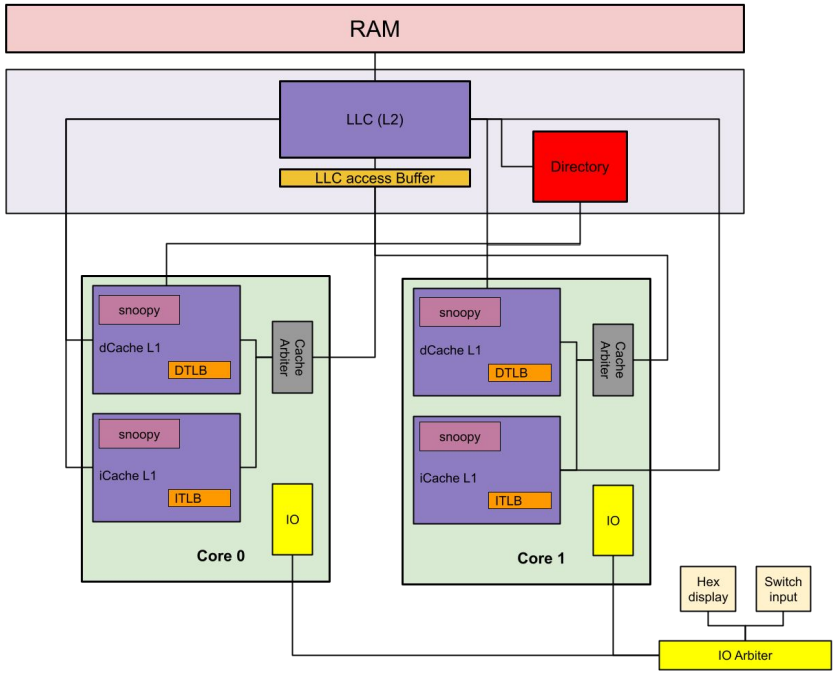
- Fmax of 43.94 MHz, fixed with PLL to run at 40 MHz.
- Low resource usage:
  - ➢ 27% of use of logic elements
  - ➢ 16% of use of memory bits

Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	Elpis
Top-level Entity Name	Elpis
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	13,481 / 49,760 (27 %)
Total registers	7057
Total pins	126 / 360 (35 %)
Total virtual pins	0
Total memory bits	262,144 / 1,677,312 (16 %)
Embedded Multiplier 9-bit elements	6 / 288 (2 %)
Total PLLs	1 / 4 (25 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

	Fmax	Restricted Fmax	Clock Name	Note
1	43.94 MHz	43.94 MHz	cpuPll|altpll_component|auto_generated|pll1|clk[0]	

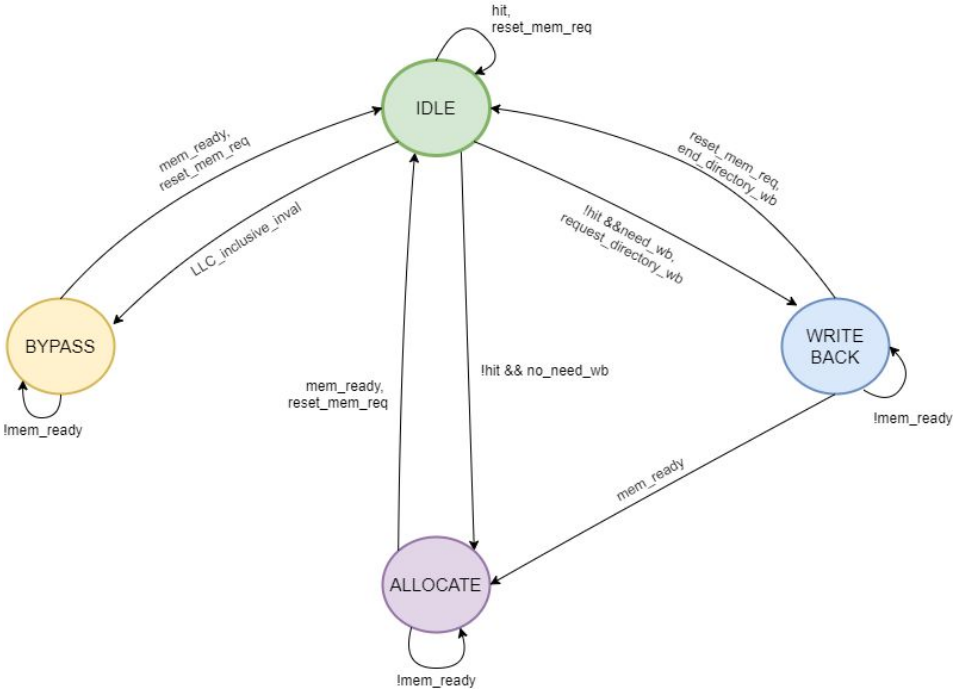
4. Road to dual core: Horus diagram

- Multiprocessor composed of 2 Elpis cores
- New incorporations:
  - ➢ Shared Last Level Cache (LLC)
  - ➢ Directory-based cache coherence
  - ➢ LLC access buffer as a serialization point
  - ➢ IO arbiter
- Core modifications:
  - ➢ L1 caches → 4'Cs
  - ➢ Larger RAM size
  - ➢ Point to point communication to L1 caches from LLC and directory
  - ➢ Caches arbiter



4. Road to dual core: L1 caches

- Direct-mapped with 4 lines of 128 bits each
- Added support for directory invalidations and requests
  - ➢ Coherence support with valid and dirty bits, instead of a FSM for each line based on MSI for directory
- Added support for invalidations due to LLC inclusivity
  - ➢ Added a new state: ByPass



4. Road to dual core: Directory

- Centralized directory with inclusivity LLC
- Write-invalidate protocol
- Stable states and transient states

Structure for each LLC cache block:

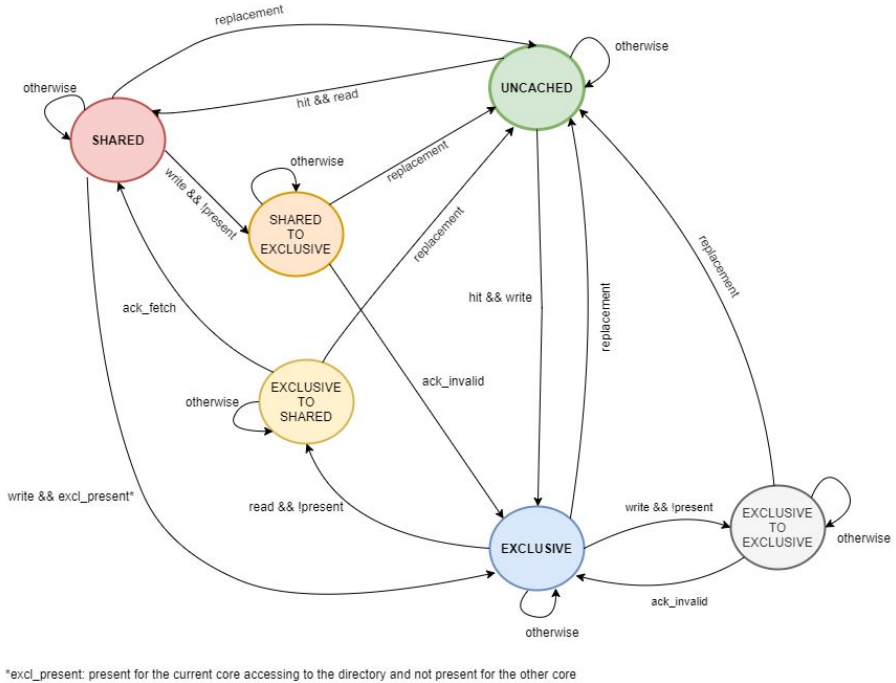
➢ Presence bit for each core
➢ Dirty bits
➢ FSM

Inputs:

✓ Petitions (read, write)
✓ ACKs fetch and invalid
✓ Way and set of addresses in LLC

Outputs:

✓ Petition permission
✓ Requests fetch and invalid



*excl_present: present for the current core accessing to the directory and not present for the other core

4. Road to dual core: LLC

- Addition of 2 extra states: Petition and ByPass
- First approach was direct-mapping with 64 blocks of 128 bits
  - ➢ Rejected due to inclusivity
- Final approach → N-way associative:
  - ➢ 8 ways
  - ➢ 8 sets each way
  - ➢ 128 bits each block
  - ➢ pLRU
- Consistency model based on Relaxed consistency

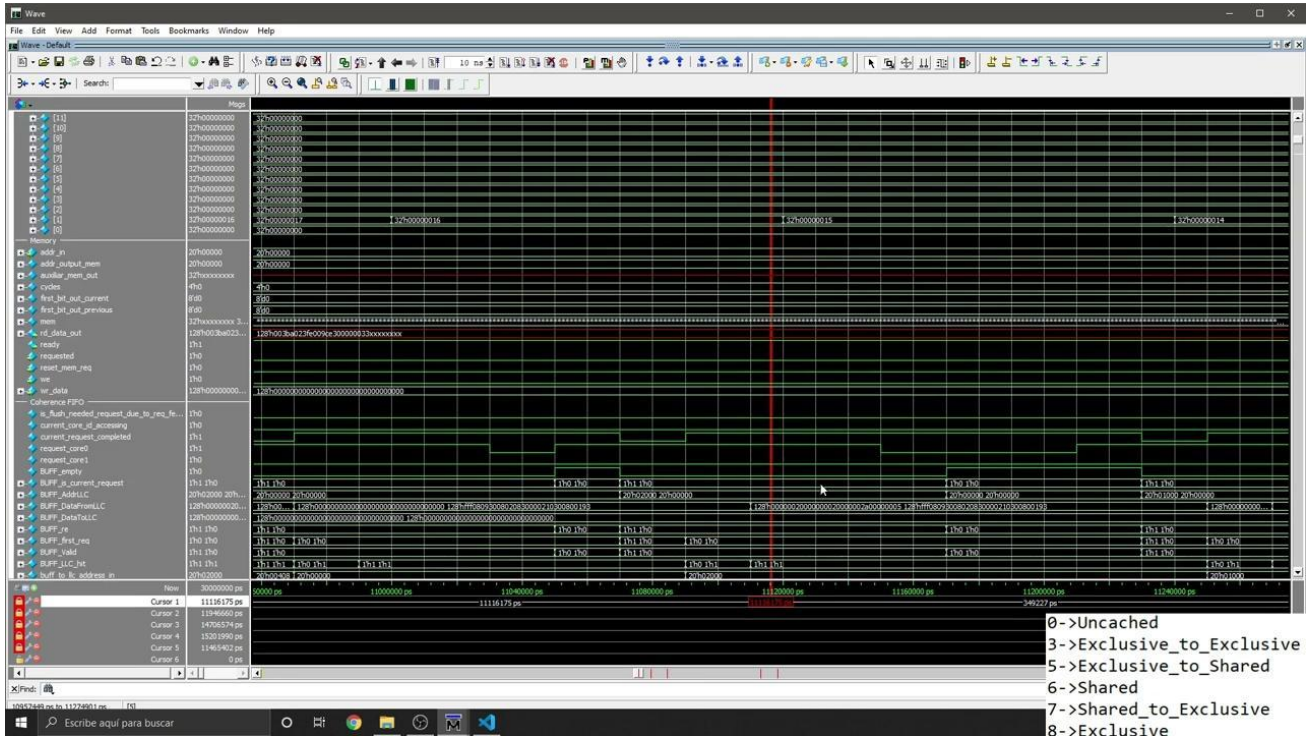
4. Road to dual core: Issues & overview

Once Horus was synthesized, the compilation report obtained was:

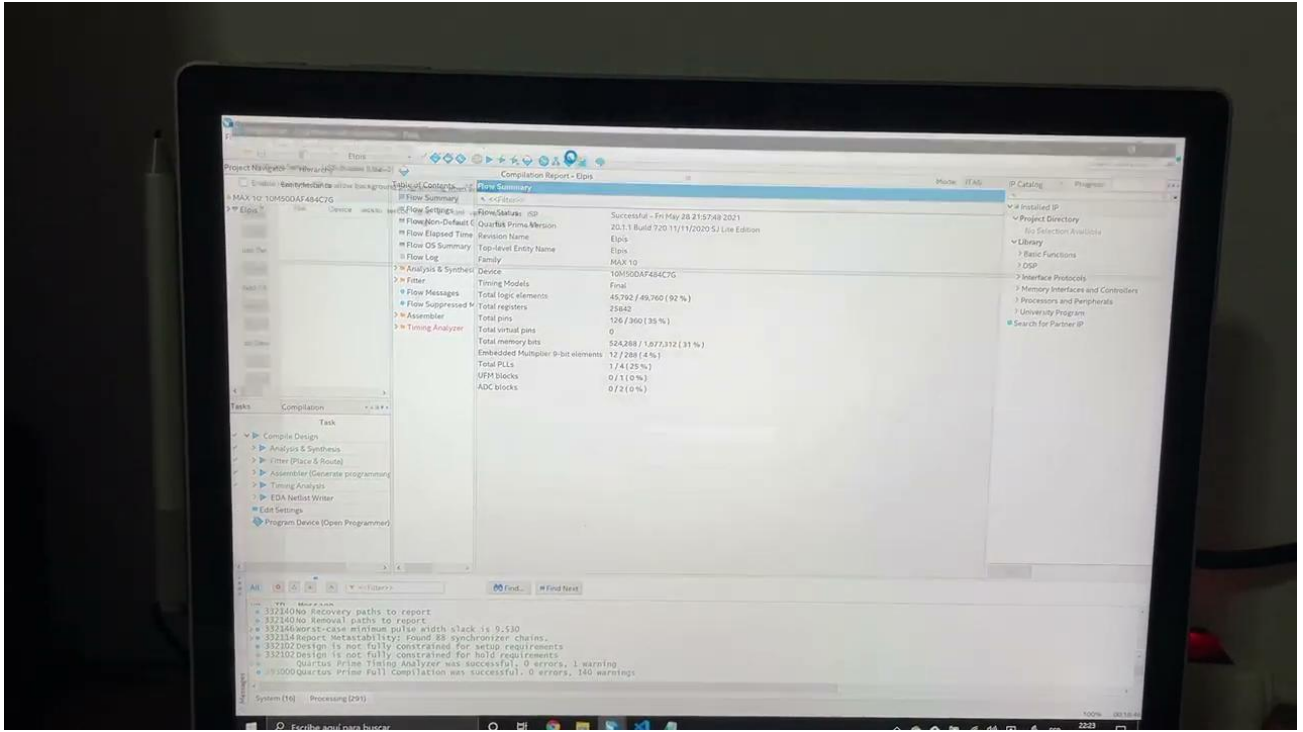
- High resource usage:
  - ➢ 92% of usage of LEs due to LLC 8-way associative cache → With direct-mapping LLC the usage was about 60%
- Fmax of 28MHz:
  - ➢ Use of a PLL to fix the clock to run at 25MHz
  - ➢ With 1 core we can reach up to 40MHz. The decrease of the frequency is because we have use some atomicity of operations.

Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	Elpis
Top-level Entity Name	Elpis
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	45,792 / 49,760 (92 %)
Total registers	25842
Total pins	126 / 360 (35 %)
Total virtual pins	0
Total memory bits	524,288 / 1,677,312 (31 %)
Embedded Multiplier 9-bit elements	12 / 288 (4 %)
Total PLLs	1 / 4 (25 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

5. Demo Coherence



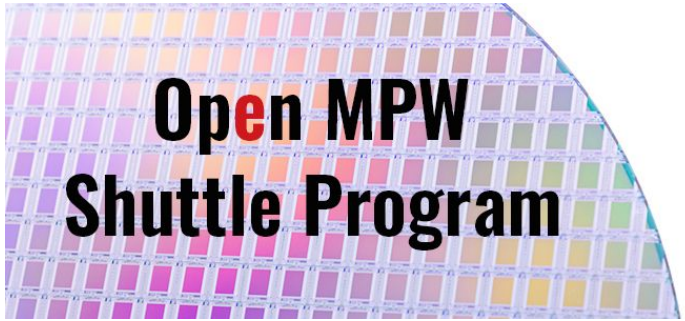
5. Demo FPGA


Compilation Report - Elpis
Flow Summary
Flow Status
Quartus Prime Version
Revision Name
Top-level Entity Name
Family
Device
Timing Models
Total logic elements
Total registers
Total pins
Total virtual pins
Total memory bits
Embedded Multiplier 9-bit elements
Total PLLs
UFM blocks
ADC blocks
Successful - Fri May 28 21:57:49 2021
20.1.1 Build 720 11/11/2020 SJ Lite Edition
Elpis
Elpis
MAX 10
10M50DAF484C7G
Final
45,792 / 49,760 (92 %)
25842
126 / 360 (35 %)
0
524,288 / 1,677,312 (31 %)
12 / 288 (4 %)
1 / 4 (25 %)
0 / 1 (0 %)
0 / 2 (0 %)
Quartus Prime Full Compilation was successful. 0 errors, 140 warnings

3. Processor Design (PD) work

6. Open MPW Shuttle Program (MPW-3 edition)

- MPW-3 is the third Open MPW Shuttle **providing fabrication for fully open-source projects using the SkyWater 130 nm Open Source PDK** announced by Google and SkyWater.



- **Efabless** is the company in charge of the fabrication and Google pays the tapeout for 40 of different projects

efabless.com

Sponsored by Google

- **Our target:** Deliver the Elpis project to be manufactured

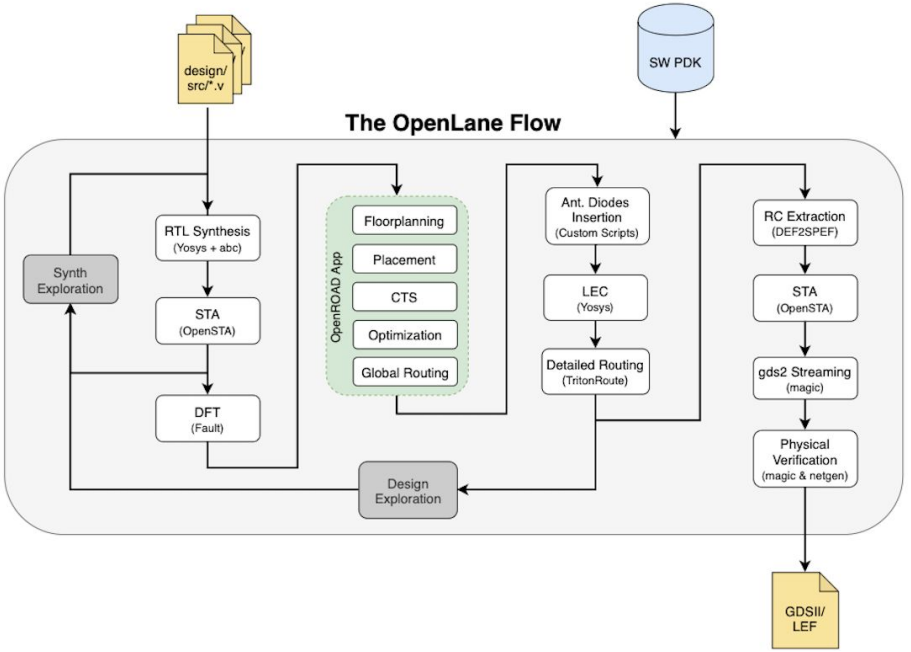
7. What is needed

- **Skywater PDK** → Set of files that defines each cell (“logic gate”) in the foundry
- Caravel user project wrapper
  - ➢ **PicoRISC-V** as main processor that is used to program the user design
  - ➢ Fully integration with OpenLane, Sky130nm and Caravel (PicoRISC-V and related minimum HW)
- **OpenLane** is an automated RTL to GDSII flow based on several components and scripts for design exploration and optimization. The flow performs full ASIC implementation steps from RTL all the way down to GDSII.
  - ➢ **GDSII** → File that contains the layout of the chip and is sent to the vendor who makes the mask for the chip to manufactured in the fab.

8. OpenLane workflow

Flow that consists of several stages, where each stage may consist of multiple sub-stages.

- Synthesis
- Floorplan and PDN (Power Distribution Network)
- Placement
- Clock Tree Synthesis (CTS)
- Routing
- GDSII Generation
- Checks



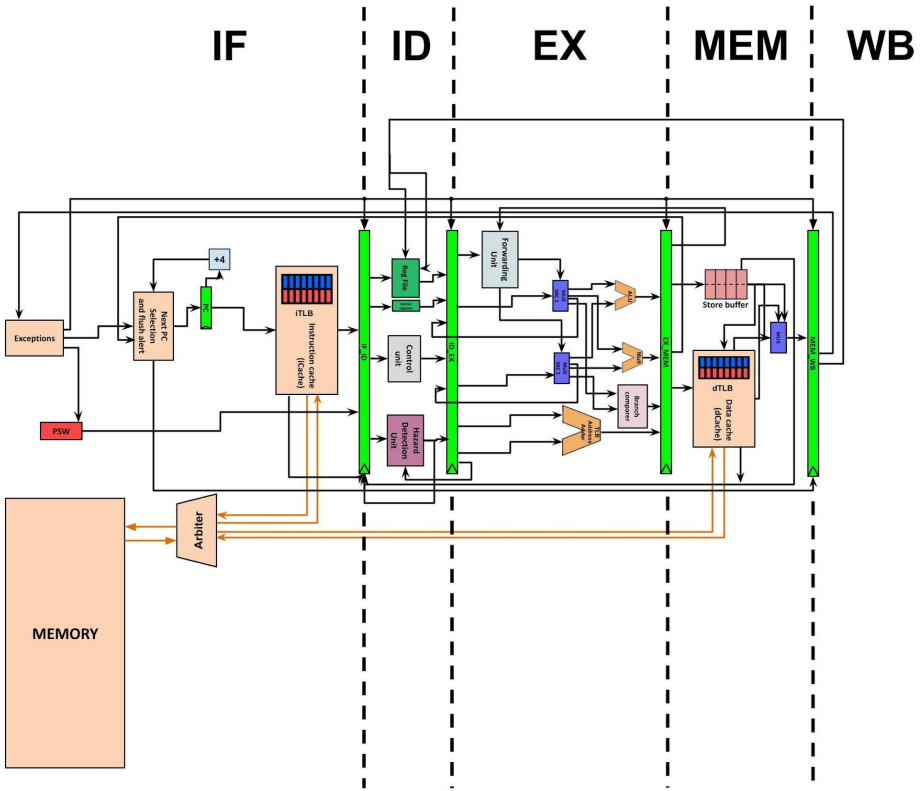
9. First set of problems

- Very difficult to install all the needed tools and dependencies
- Each workflow is highly time consuming
- Documentation of caravel user project and openlane is unclear, disperse and incomplete (and sometimes unavailable 😀)
- IVerilog compiles different to Quartus, so we had to perform some changes in our Verilog code
- The .c compilation program the user project and connect it to caravel has no support of the c standard library
- We had to move to doing macros for different modules after playing a bit with OpenLane as the flow was failing due to different issues such us overlap, density, etc. Even that we played with different variables like DIE_AREA or PL_TARGET_DENSITY among others
- As the area was a critical restriction, we decided to present a lighter version of our Elpis core

10. Elpis (light version)

In our simpler version of Elpis core we decided to get rid off:

- ✘ Multiplication pipeline
- ✘ History File
- ✘ TLBs

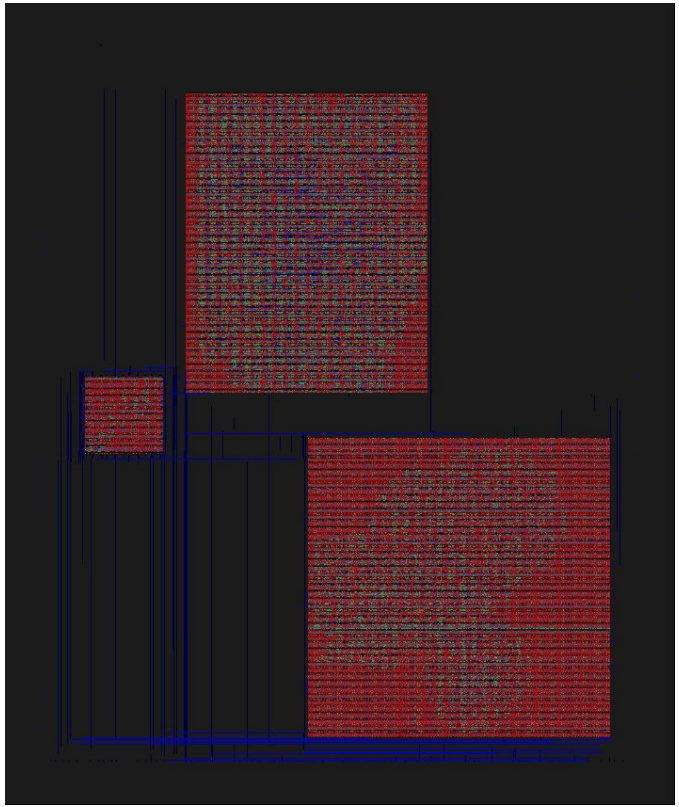


11. SRAM

- The SkyWater PDK has no support for inferring SRAMs
- **OpenRAM** is an external tool that its purpose is to give support of creating SRAM for the Sky130 PDK, among others
- However, it has many problems:
  - ➢ The SRAM needs to be on the top design due to power layer restrictions → Caches not viable
  - ➢ The DRC (Design Rule Checking) checking fails. It can be fixed changing some PDK files, but then it will fail for the rest of the design
- So, it was discarded to be used after some effort trying to solve each problem and we created our custom RAM (with very reduced RAM size) just letting the tool to infer the needed logic for a simple Verilog definition

12. Using macros

- We decided to create different macros in order to take advantage of different density configurations
- First, we tried to have different macros: chip_controller, i_arbiter, o_arbiter, custom_sram and core
- As some macros where tiny, we had problems in the connection of the power pins and we decided to join i_arbier, o_arbiter and chip_controller in a unique macro. So, we have 3 different macros in the end



13. Second set of problems

- Solve **setup violations** (the setup time constraint is the amount of time required for the input to a flip-flop to be stable before a clock edge)
  - ➢ Solved it increasing the clock period
- Solve **hold violations** (the hold time constraint is the minimum amount of time needed for the input to a flip-flop to be stable after a clock edge)
  - ➢ Solve it adding extra variables for the hold violations
- Warnings for **slew violations** that we were impossible to fix (time needed to change from 0 to 1 or vice versa in a flip flop)
  - ➢ They do not avoid finishing the design

14. Second set of problems

- Run 2 additional checkings in the Efabless platform: Precheck and Tapeout precheck → Later one raised a strange DRC error related to metal density
  - ➢ It was supposed to be an official fix before the final date (it wasn't) → we ended up using a workaround with a modified OpenLane docker image provided by another user, but ...
  - ➢ … this workaround creates huge slew violations in the wrapper (top design that joins the macros)
  - ➢ Final decision: User image for macros + official image for wrapper
    - ✓ This mix eliminates the new slew violations and solves the DRC error
- When we reached this point, we were run out of time and we only had time to test the whole design in RTL simulation and a very similar design in a Intel FPGA. So, after delivering the design we tried to run a GL simulation with the generated netlist and we have found out that it is not working :(

15. Conclusions

- The compilation of different RTL tools is different (Quartus, modelsim, Iverilog, yosys)
  - ➢ GL tests are essential even having the design working in a FPGA, e.g.

- OpenLane and Sky130 have potential, but they need more development, better documentation support and maturity (they have been around for 1 year and a half)

- It’s hard to do a tapeout in 1 month and a half with a (yet immature) tool that is used for first time for us even having a previous design for another workflow working (FPGA)

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Elpis

From simulation to fabrication

Rodrigo Huerta Gañán (rodrigo@ac.upc.edu)
Aurora Tomás Berjaga (aurora@ac.upc.edu)

ARCO - 23 November 2021