Computer Architectures Session 1

Single-cycle RISC-V Processor

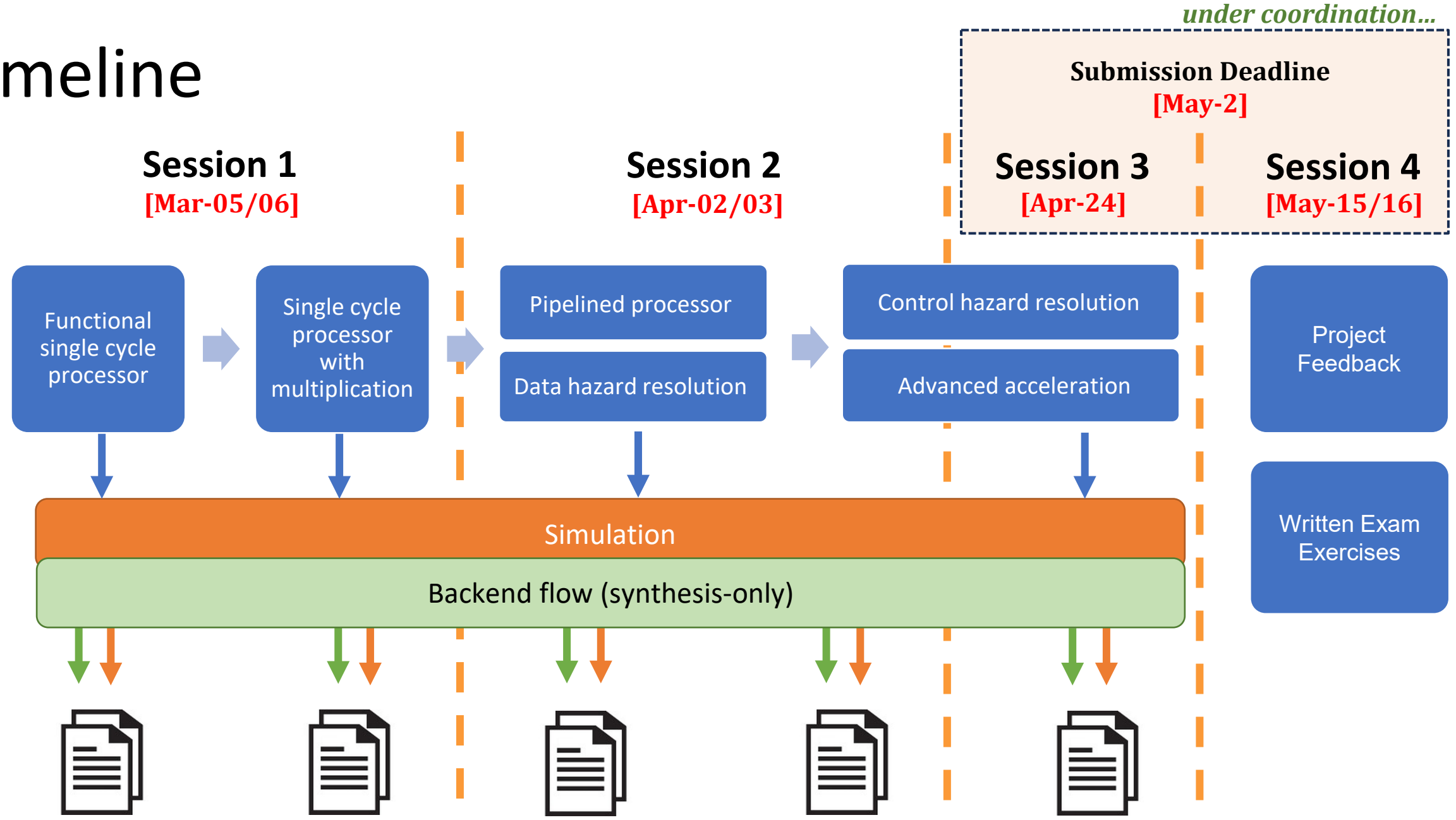
TAs :
Jun Yin (jun.yin@kuleuven.be)
Yuanyang Guo (yuanyang.guo@imec.be)
Xiaoling Yi (xiaoling.yi@kuleuven.be)
Yunzhu Chen (yunzhu.chen@imec.be)

Objective

- RTL design of a RISC-V microprocessor
  - Simple Implementation
  - Pipelined implementation
  - Data Hazard resolution
  - Advanced Acceleration
    - Branch prediction
    - SIMD
    - …

Chapter 4

Timeline


under coordination…
Submission Deadline
[May-2]
Session 1
[Mar-05/06]
Session 2
[Apr-02/03]
Session 3
[Apr-24]
Session 4
[May-15/16]
Functional single cycle processor
Single cycle processor with multiplication
Pipelined processor
Data hazard resolution
Control hazard resolution
Advanced acceleration
Project Feedback
Written Exam Exercises
Simulation
Backend flow (synthesis-only)

Session requirements and criteria:

- The assignments must be completed in groups of **≤2 people**.

- After completing each one of the scenarios:

1. You have to copy-paste your RTL source code into the corresponding SOLUTION folder.
2. You have to complete part of a small report to record performance readings and answer several related questions (see report.docx) .

- This project counts for **4 points** in the final course grade.

1. The performance check in each design scenario.
2. The report.

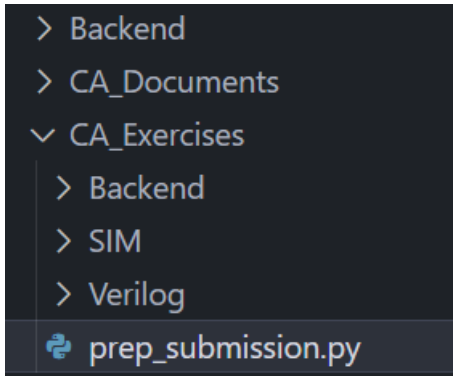
Grading

ITEM	Points
Functional pipelined MULT2	0.5
Functional pipelined MULT3	0.5
Functional MULT4	0.4
Functional MULT4 #cycles ≤ baseline impl. (1636 cc)	0.8
Advanced MULT4 #cycles ≤ advanced impl. (828 cc)	0.8
Report	1.0
Total	**4.0**

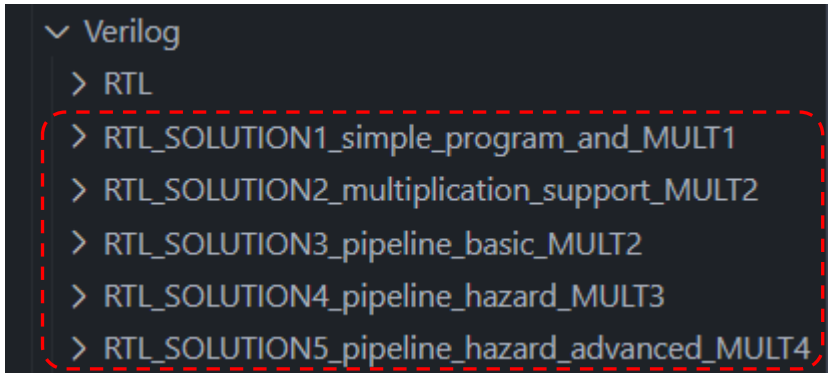
Project handover

Deadline: **May 2nd**

Session requirements and criteria:



- We release the same grading script as final.
  - Command: `python3 prep_submission.py`
  - This script works in standalone folders. But it is always wise to backup your ongoing **/RTL** codes before execution.
- Do not forget to perform at least one dry-run before the final submission.
  - **Make sure you have put the right version of source code into the right folder!**



```
Checking simpleprogram->simple_program ...
Checking mult1->MULT1 ...
Checking mult2->multiplication_support_MULT2 ...
Checking mult2->pipeline_basic_MULT2 ...
Checking mult3->pipeline_hazard_MULT3 ...
Checking mult4->pipeline_hazard_advanced_MULT4 ...
Done
Scoreboard:

        Funtional pipelined MULT2:      0.5 pts (9 cc, 2300 cc, 40 cc, 40 cc)

        Funtional pipelined MULT3:      0.5 pts (25 cc)

        Funtional pipelined MULT4:      1.2 pts (1636 cc)

        ----------------------------------------
        Total Score:    2.2 pts

Do not forget to complete the GROUP_X folder before your submission!
```

Reference outputs when grading our baseline (2.2/3 pts).

prep_submission.py

- Evaluate your RTL_SOLUTION(s)
- Basic debug tracing if the program fails
- Create the structure for submission

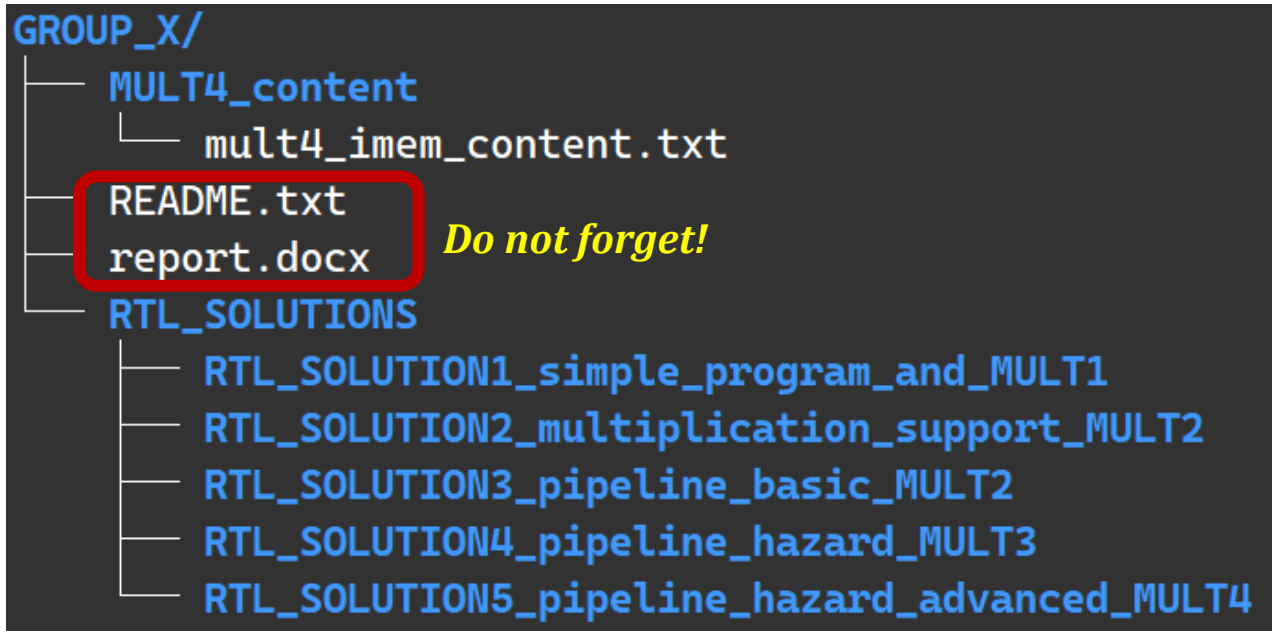
```
GROUP_X/
├── MULT4_content
│   └── mult4_imem_content.txt
└── RTL_SOLUTIONS
    ├── RTL_SOLUTION1_simple_program_and_MULT1
    ├── RTL_SOLUTION2_multiplication_support_MULT2
    ├── RTL_SOLUTION3_pipeline_basic_MULT2
    ├── RTL_SOLUTION4_pipeline_hazard_MULT3
    └── RTL_SOLUTION5_pipeline_hazard_advanced_MULT4
```

```
Checking simpleprogram->simple_program_and_MULT1 ...
Checking mult1->simple_program_and_MULT1 ...
Checking mult2->pipeline_basic_MULT2 ...
Verilog src folder not found. Please check file structures.
Checking mult3->pipeline_hazard_MULT3 ...
        rm -rf xcelium.d xrun.* xmverilog.* *.vcd *.shm
        clear;
xrun +sv -f files_verilog_grading.f -64bit -timescale 1ns/10ps -access +rwc -allowredefinition

        TOOL:   xrun(64)        23.03-s002: Started on Jan 31, 2024 at 11:10:57 CET
        xrun(64): 23.03-s002: (c) Copyright 1995-2023 Cadence Design Systems, Inc.
        file: ../Verilog/cpu_tb.v
                module worklib.cpu_tb:v
                        errors: 0, warnings: 0
        file: ../Verilog/sky130_sram_2rw.v
                module worklib.sky130_sram_2rw_32x128_32:v
                        errors: 0, warnings: 0
                module worklib.sky130_sram_2rw_64x128_64:v
                        errors: 0, warnings: 0
                        Caching library 'worklib' ....... Done
                Elaborating the design hierarchy:
                        Caching library 'worklib' ....... Done
        cpu dut(
                |
        xmelab: *E,CUVMUR (../Verilog/cpu_tb.v,67|6): instance 'cpu_tb.dut' of design unit 'cp
u' is unresolved in 'worklib.cpu_tb:v'.
        xrun: *E,ELBERR: Error (*E) or soft error (*SE) occurred during elaboration (status 1)
, exiting.
        TOOL:   xrun(64)        23.03-s002: Exiting on Jan 31, 2024 at 11:10:57 CET  (total: 0
0:00:00)
make: *** [sim_grading] Error 1


Errors occurred when simulating. Please check your source code structure.
Checking mult4->pipeline_hazard_advanced_MULT4 ...
Done
Scoreboard:

        Funtional pipelined MULT2:      0.0 pts (9 cc, 2300 cc, False cc)

        Funtional pipelined MULT3:      0.0 pts (False cc)

        Funtional pipelined MULT4:      1.0 pts (844 cc)

        ----------------------------------------
        Total Score:    1.0 pts
```

Debug tracing example.

Session requirements and criteria:

- The **report** and your **final processor design** need to be handed in through Toledo, following the below file structure:



Do not forget!

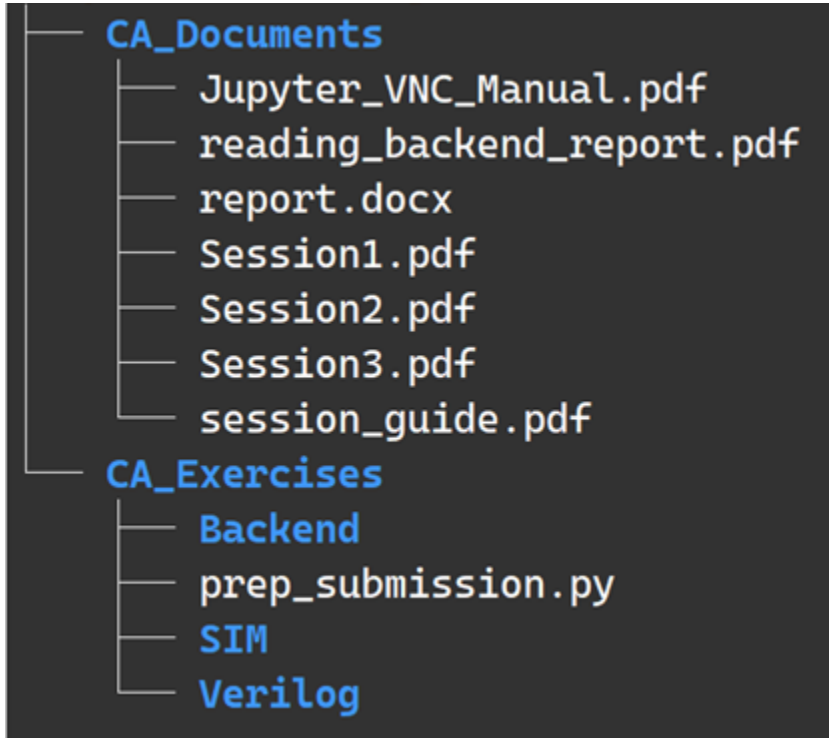
1) Name your project as **Group_X** (X is your group number);
2) Put all the students' name and students' R-number of the group in README.txt;
3) You can also put anything that you think we need to know before running your project in the README.txt.
4) Pack your group folder into a zip file and submit.

- **Mult4 session (session3) is special so that additional files are allowed.**

- The **deadline** of the project is **May. 2nd, 2025**.

Course material

- Project resources



- Extra resources
  - Computer organization and design. RISC-V edition.

    See Toledo
  - RISC-V specification

    https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf
  - RISC-V green card

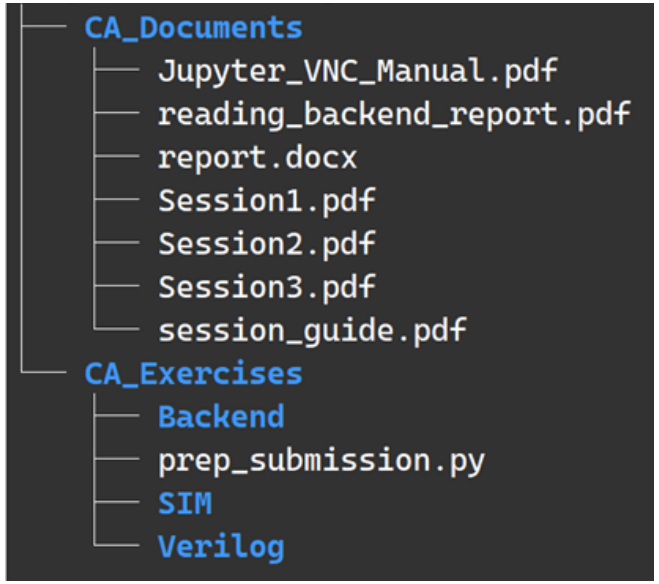
    https://dejazzer.com/coen2710/lectures/RISC-V-Reference-Data-Green-Card.pdf
  - Verilog tutorial

    https://www.asic-world.com/verilog/veritut.html
  - RISC-V machine code generator *

    https://github.com/Kritagya-Agarwal/Assembly-To-Machine-Code-RISC-V
  - LLMs are also good at explaining these open-sourced “facts” now. Feedbacks are welcomed if you have tried out!

* assembles RV64 but only simulates RV32, **only use it for assembly to machine code translation**

Project resources



Please refer to the CA_Documents for more information:

- **session_guide**
- **Jupyter_VNC_Manual**
- **reading_backend_report**

- **Verilog**
  - RTL
    RTL source code
  - sky130_sram_2rw.v
    Memory behaviour model
  - cpu_tb.v
    Testbench ***(modification not allowed)***
- **Backend**
  - CA_RISCV.ipynb
    Python file to run backend flow
  - Figs
  - OpenRAM_output
  - Setup.sh
- **prep_submission.py**
  Python script to dry-run the grading before submission

- **SIM**
  Simulation related resources
  - data
    Instruction and data memory sources
    - testcode_m
      Imem and dmem contents used for different scenarios
    - dmem_content.txt
      Data memory to be loaded in testbench
    - imem_content.txt
      Instruction memory to be loaded in testbench
  - files_verilog.f
    RTL simulation filelist ***(file structure needs sync)***
  - Makefile
    simulation-related commands
  - xcelium_23.03.rc
    simulator

Remarks: Keep your project safe!

Please keep all 5 versions of your processor (RTL), we will grade based on them!

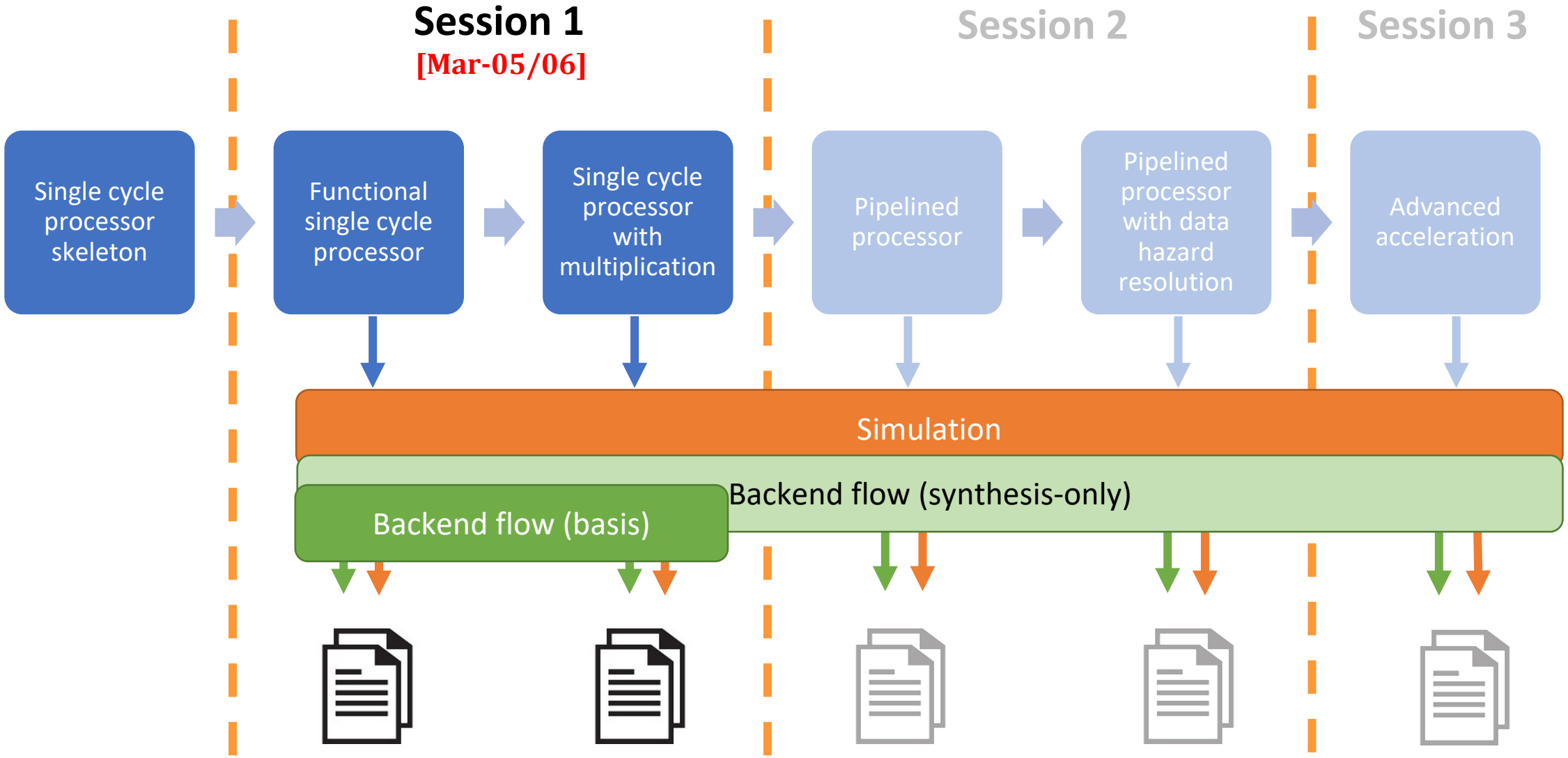
We highly suggest you to use ***git*** as a versioning tool.

It helps you to:

- keep track of modifications
- automatically sync the project between group members
- allow TAs to support you more easily

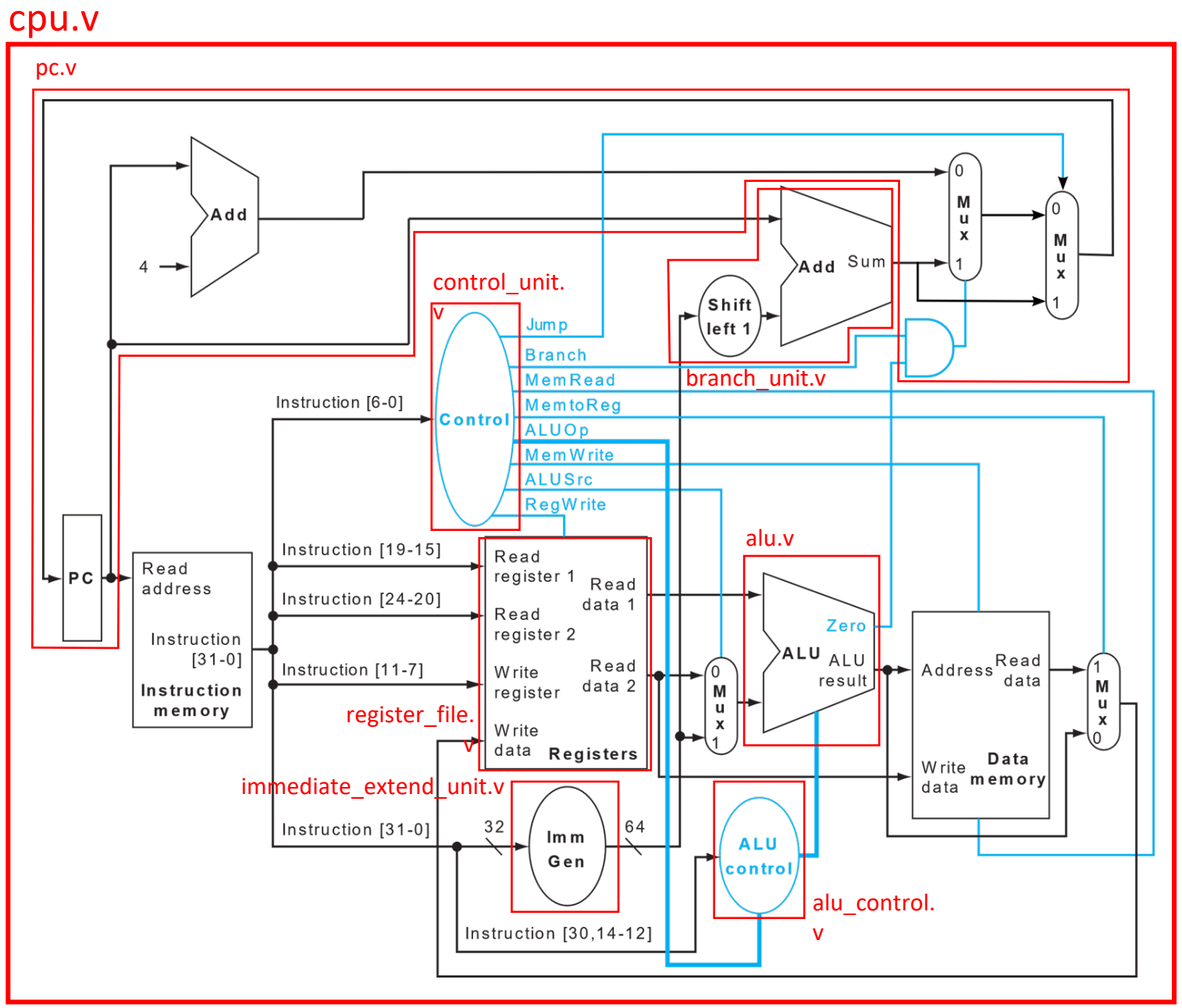
Please have a look at the dedicated guide.

Today's session: Single-cycle processor



Basic architecture

- Testbench
  - cpu_tb.v
- RTL
  - alu_control.v
  - alu.v
  - branch_unit.v
  - control_unit.v
  - cpu.v
  - immediate_extend_unit.v
  - mux_2.v
  - pc.v
  - register_file.v
  - sram.v
- IP Library
  - sky130_sram_2rw.v



Patterson-Hennessy Section 4.1 to 4.4

Simulation Tool

- Simulation (Cadence Xcelium)
  - Tool to simulate the behaviour of the RTL
- In the `./SIM` folder, set up the environment

```
source xcelium_23.03.rc
```

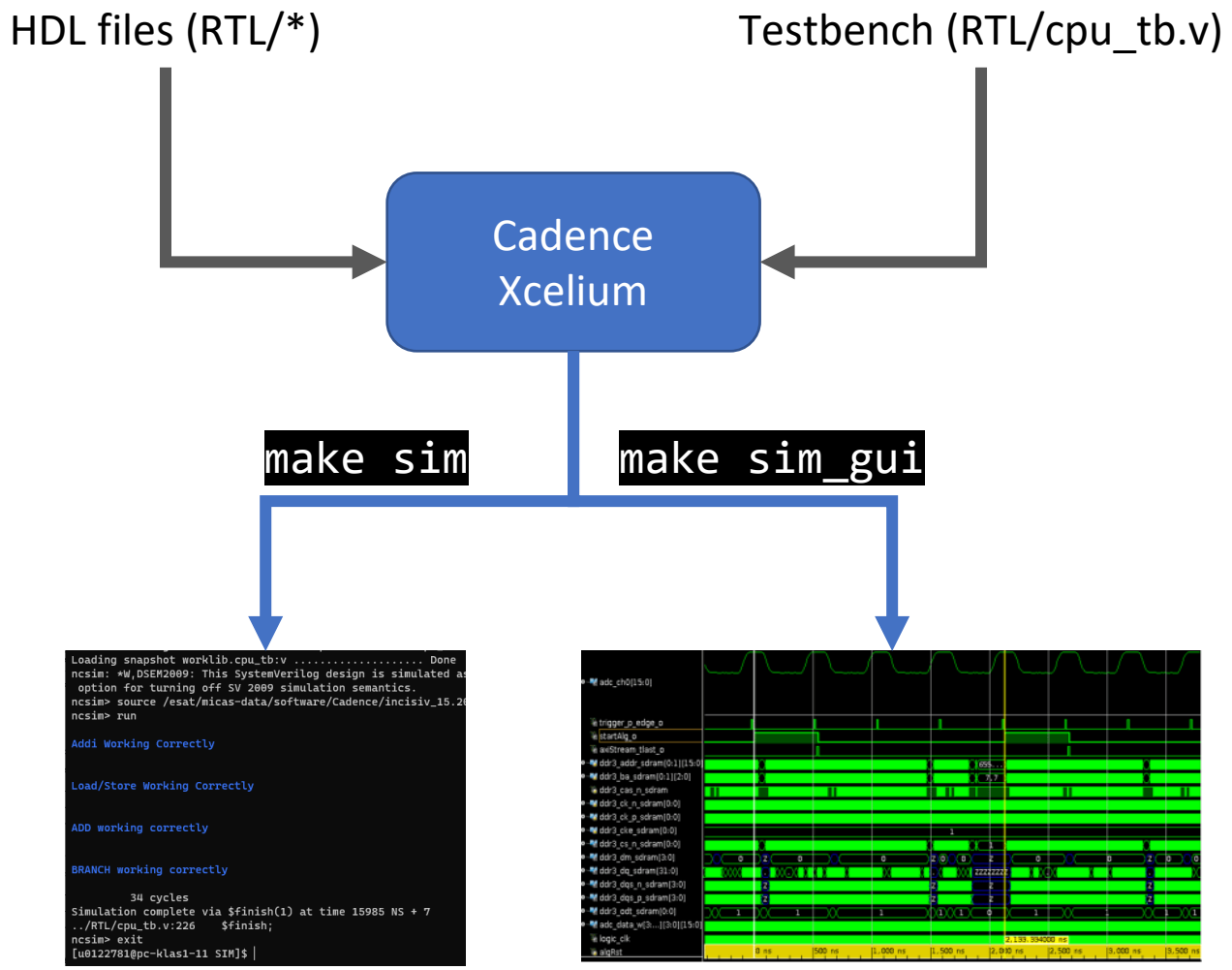
- Make simulation

```
make sim
```

uses `cpu_tb.v`, and the content of `SIM/data/imem_content` and `SIM/data/dmem_content` to verify that the current RTL is able to run the program.

```
make sim_gui
```

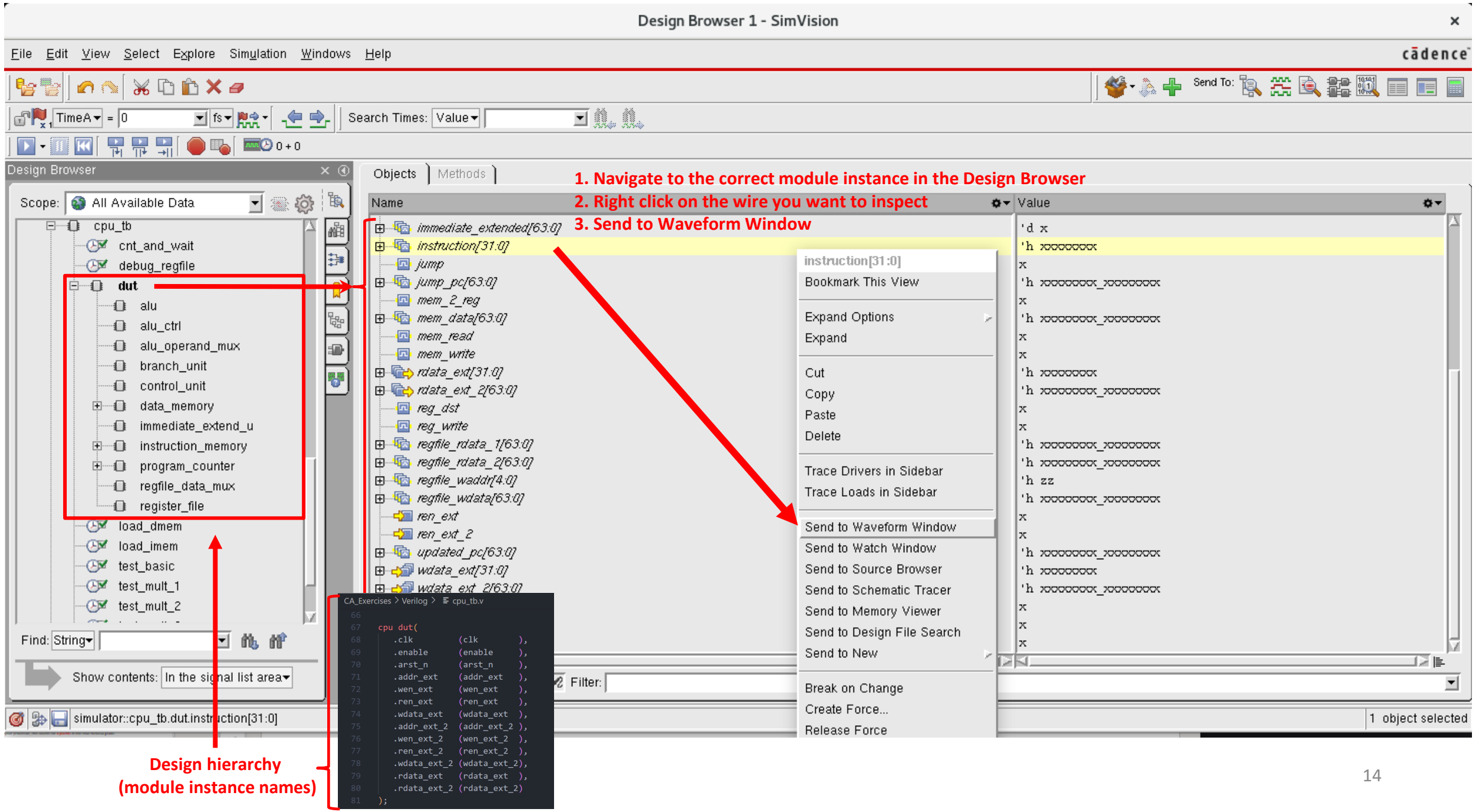
same but invokes waveform for debugging (interactive)



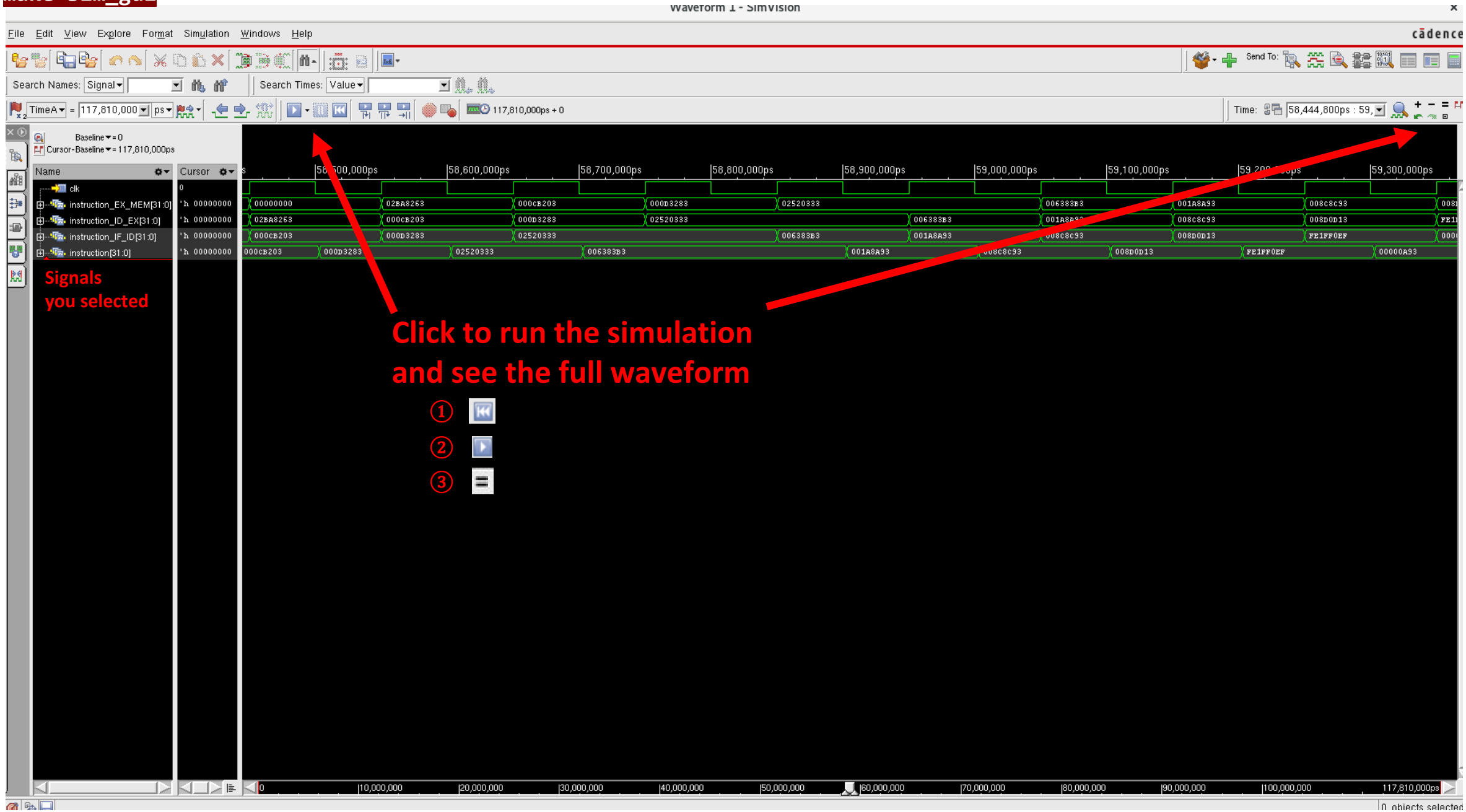
Test if program runs properly

Provide waveform for debugging

```
make sim_gui
```

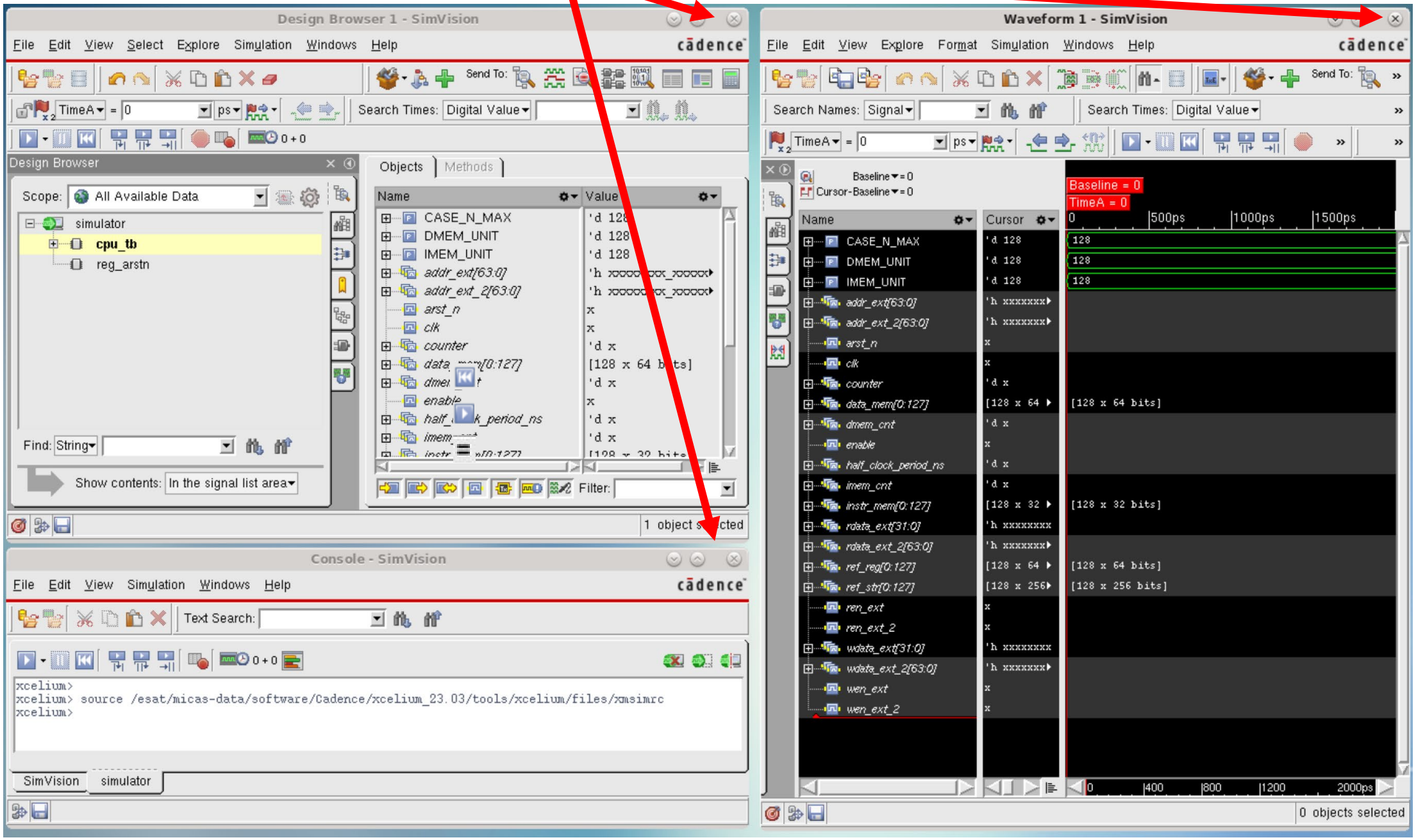

Design Browser 1 - SimVision
File Edit View Select Explore Simulation Windows Help
cadence
Design Browser
Scope: All Available Data
cpu_tb
cnt_and_wait
debug_regfile
dut
alu
alu_ctrl
alu_operand_mux
branch_unit
control_unit
data_memory
immediate_extend_u
instruction_memory
program_counter
regfile_data_mux
register_file
load_dmem
load_imem
test_basic
test_mult_1
test_mult_2
Find: String
Show contents: In the signal list area
Objects
Methods
Name
Value
immediate_extended[63:0]
instruction[31:0]
jump
jump_pc[63:0]
mem_2_reg
mem_data[63:0]
mem_read
mem_write
rdata_ext[31:0]
rdata_ext_2[63:0]
reg_dst
reg_write
regfile_rdata_1[63:0]
regfile_rdata_2[63:0]
regfile_waddr[4:0]
regfile_wdata[63:0]
ren_ext
ren_ext_2
updated_pc[63:0]
wdata_ext[31:0]
wdata_ext_2[63:0]
1. Navigate to the correct module instance in the Design Browser
2. Right click on the wire you want to inspect
3. Send to Waveform Window
instruction[31:0]
Bookmark This View
Expand Options
Expand
Cut
Copy
Paste
Delete
Trace Drivers in Sidebar
Trace Loads in Sidebar
Send to Waveform Window
Send to Watch Window
Send to Source Browser
Send to Schematic Tracer
Send to Memory Viewer
Send to Design File Search
Send to New
Break on Change
Create Force...
Release Force
Filter:
simulator::cpu_tb.dut.instruction[31:0]
1 object selected
CA_Exercises > Verilog > cpu_tb.v
cpu dut(
.clk (clk),
.enable (enable),
.arst_n (arst_n),
.addr_ext (addr_ext),
.wen_ext (wen_ext),
.ren_ext (ren_ext),
.wdata_ext (wdata_ext),
.addr_ext_2 (addr_ext_2),
.wen_ext_2 (wen_ext_2),
.ren_ext_2 (ren_ext_2),
.wdata_ext_2 (wdata_ext_2),
.rdata_ext (rdata_ext),
.rdata_ext_2 (rdata_ext_2)
);
Design hierarchy
(module instance names)

```
make sim_gui
```

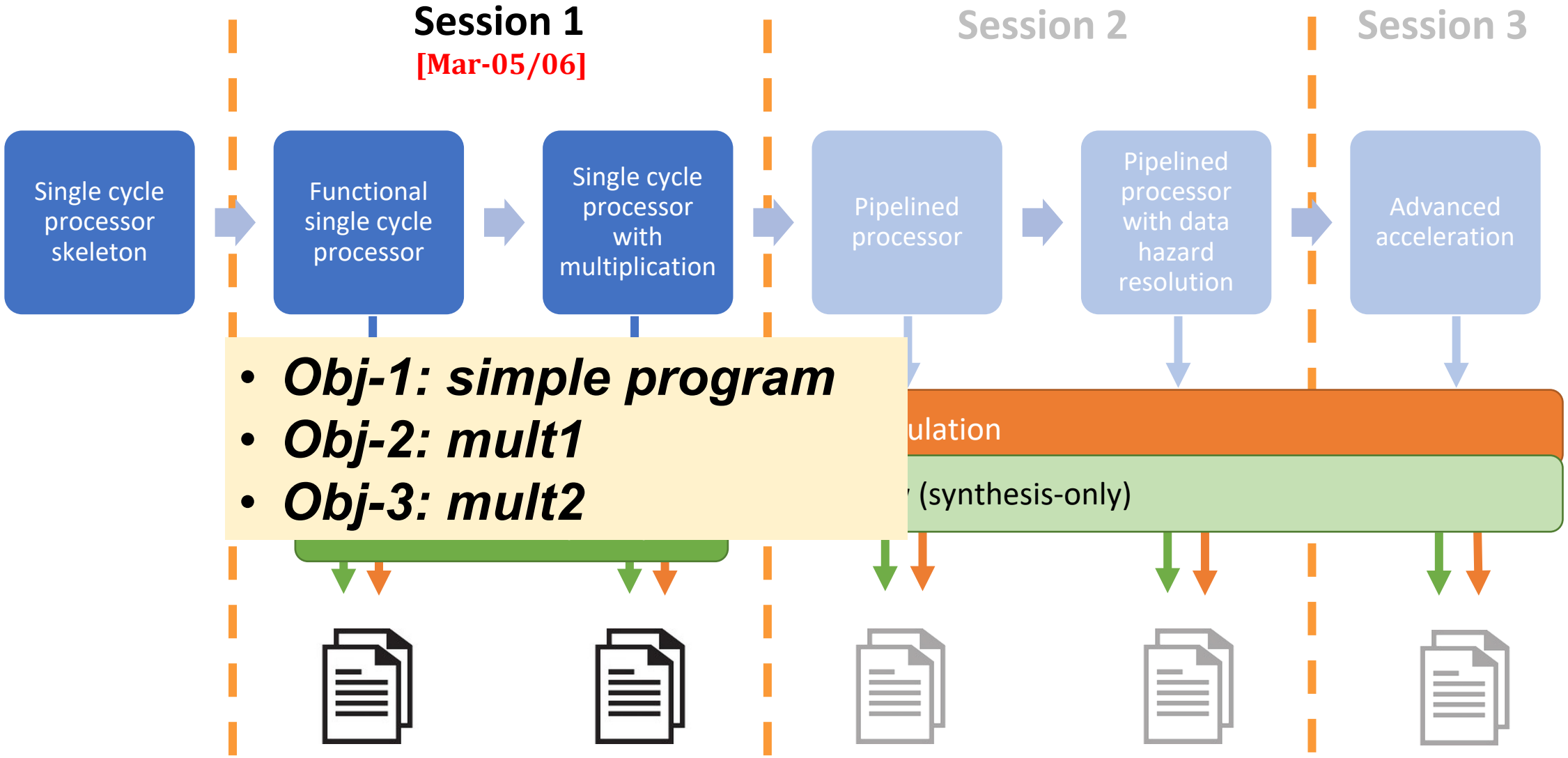

Waveform 1 - SimVision
File Edit View Explore Format Simulation Windows Help
cadence
Search Names: Signal
Search Times: Value
Send To:
TimeA = 117,810,000 ps
117,810,000ps + 0
Time: 58,444,800ps : 59,
Baseline = 0
Cursor-Baseline = 117,810,000ps
Name
Cursor
clk
instruction_EX_MEM[31:0]
instruction_ID_EX[31:0]
instruction_IF_ID[31:0]
instruction[31:0]
'h 00000000
58,500,000ps
58,600,000ps
58,700,000ps
58,800,000ps
58,900,000ps
59,000,000ps
59,100,000ps
59,300,000ps
00000000
02BA8263
000CB203
000D3283
02520333
006383B3
001A8A93
008C8C93
008D0D13
FE1FF0EF
00000A93
Signals
you selected
Click to run the simulation
and see the full waveform
①
②
③
0
10,000,000
20,000,000
30,000,000
40,000,000
50,000,000
60,000,000
70,000,000
80,000,000
90,000,000
100,000,000
117,810,000ps

```
make sim_gui
```

Terminate: close all three GUI windows



Today's session: Single-cycle processor



Obj-1: simple program

Single cycle processor skeleton → Functional single cycle processor

- Functional single-cycle processor
  - **Complete** the **`RTL/control_unit.v`** to support BEQ, JUMP, LD, SD, ADDI and R-type ALU instructions (ALU_R)
  - Follow the RUN CYCLE-ACCURATE SIMULATION in session_guide.pdf
  - Run **`SIMPLE_PROGRAM`**
    - Get to folder **`/SIM`**
    - Overwrite the instruction & data memory for the testbench to imem and dmem

```
cat data/testcode_m/simpleprogram_imem_content.txt > data/imem_content.txt
cat data/testcode_m/simpleprogram_dmem_content.txt > data/dmem_content.txt
```

    - Run simulation with `make sim` or `make all`

- Simple Program

```
addi x8, x0, 7
addi x9, x8, 2
sd x9, 0(x0)
ld x17, 0(x0)
ld x18, 8(x0)
add x19, x17, x18
beq x9, x17, FINAL
add x20, x8, x9
FINAL: add x20, x18, x19
sll x21, x18, x8
STOP
```

STOP is a "fake" instruction used for the cpu_tb to recognize the end of the program (Check line 305 of cpu_tb.v)

Obj-2: mult1

- Multiplication realized by addition loops

Algorithm 1:

1. Acc = 0
2. N = operand1
3. Traverse each bit of operand2 (LSB -> MSB)
   a. If the bit of operand2 is 1, accumulate. Acc = Acc + N.
   b. If the bit is 0, Acc = Acc.
   c. Left shift N by 1 position.

		1	1	0
		1	0	1
		1	1	0
	0	0	0	
1	1	0		
1	1	1	1	0

- MULT1
  - Multiply adjacent operand in pairs (5 multiplications) using Algorithm 1
  - Sums results together
- Run **MULT1** with your processor

Single cycle processor skeleton → Functional single cycle processor

```
# init
addi x16, x0, 80
addi x8, x0, 0
addi x9, x0, 0
addi x10, x0, 1

# Looping over operands
M1:ld x17, 0(x8)
ld x18, 8(x8)
addi x23, x0, 64
addi x19, x0, 1
addi x20, x0, 0
add x22, x0, x17

# Multiplication
LOOP_0: and x21, x18, x19
beq x21, x0, SHIFTING_0
add x20, x20, x22
SHIFTING_0: sll x22, x22, x10
sll x19, x19, x10
addi x23, x23, -1
beq x23, x0, M2
jal LOOP_0

# Summing results
M2: add x9, x9, x20
addi x16, x16, -16
addi x8, x8, 16
beq x16, x0, FINISH
jal M1
FINISH:
```

x16: Total size of data memory
x8: Data Memory Pointer
x23: Iteration over the number of bits
x20: Accumulated result of each multiplication
x9: Final result

Obj-3: mult2

- Single cycle processor with multiplication support
  - **Check if you need to modify** the **`alu_control.v`**, **`alu.v`** and **`cpu.v`** to support the ***mul*** instruction
  - Run **`MULT2`** with your processor

Functional single cycle processor → Single cycle processor with multiplication

- MULT2

```
# load operands
ld x8, 0(x0)
ld x9, 8(x0)
ld x10, 16(x0)
ld x11, 24(x0)
ld x12, 32(x0)
ld x13, 40(x0)
ld x14, 48(x0)
ld x15, 56(x0)
ld x16, 64(x0)
ld x17, 72(x0)

# multiplication
mul x18, x8, x9
mul x19, x10, x11
mul x20, x12, x13
mul x21, x14, x15
mul x22, x16, x17

#sums
addi x23, x0, 0
nop
nop
nop
add x23, x23, x18
nop
nop
nop
add x23, x23, x19
nop
nop
nop
add x23, x23, x20
nop
nop
nop
add x23, x23, x21
nop
nop
nop
add x23, x23, x22
nop
nop
nop
nop
```

The MUL instruction in RISC-V.

0000001	XXXXX	XXXXX	000	XXXXX	0110011
[31:25] funct7	[24:20] rs2	[19:15] rs1	[14:12] funct3	[11:7] rd	[6:0] opcode

Obj-3: hints

Single cycle processor skeleton → Functional single cycle processor

0000001	XXXXX	XXXXX	000	XXXXX	0110011
[31:25] funct7	[24:20] rs2	[19:15] rs1	[14:12] funct3	[11:7] rd	[6:0] opcode

- RISC-V **MUL** operation needs more control bits than provided in alu_control.v
- Refer to the RISC-V ISA and start from the module IO definition.

```
4  module alu_control(
5+     input wire        func7_5,
6      input wire [2:0] func3,
7      input wire [1:0] alu_op,
8      output reg [3:0] alu_control
9  );
```

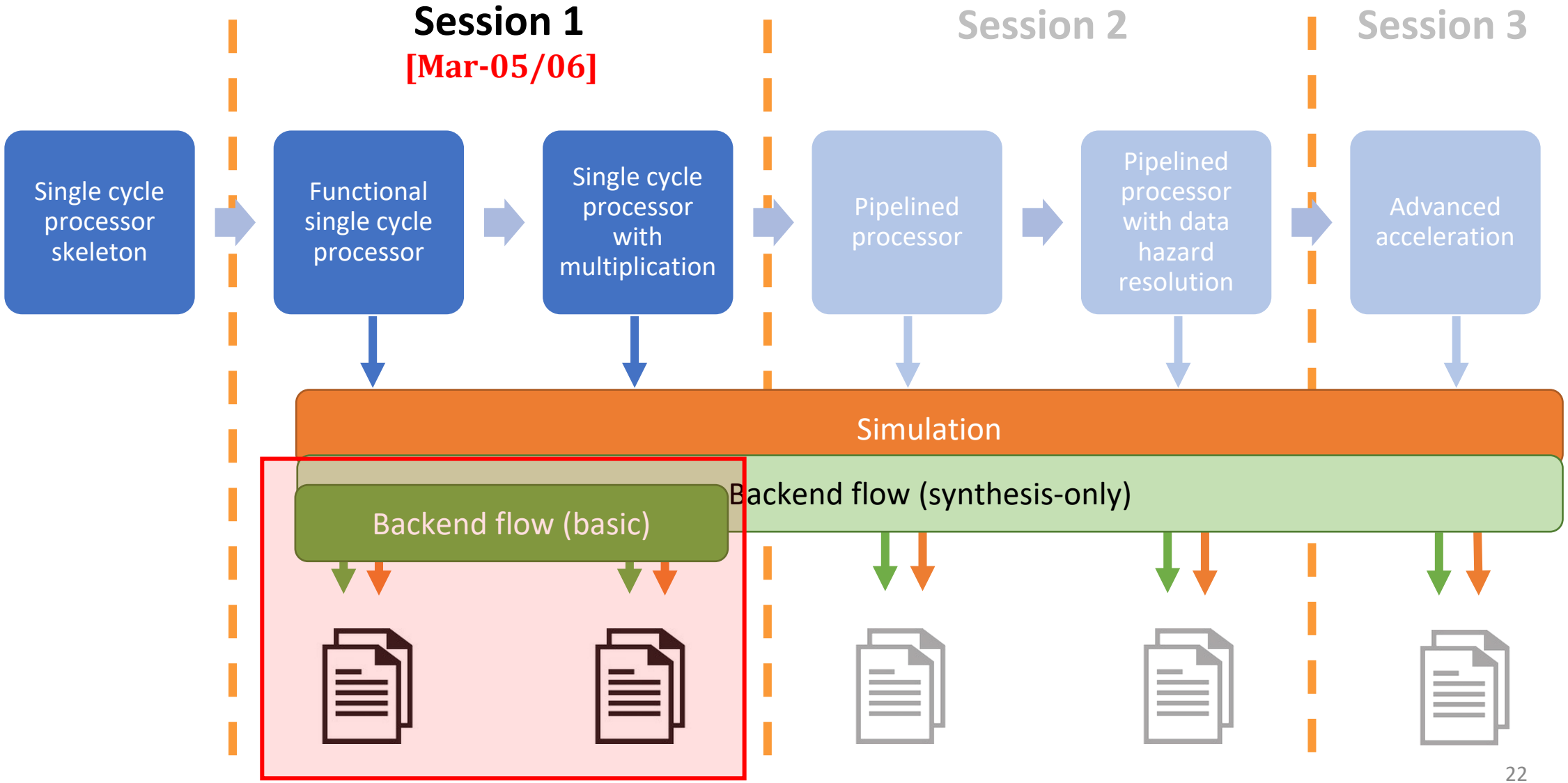
Module Definition

```
alu_control alu_ctrl(
    .func7_5        (instruction[30]   ),
    .func3          (instruction[14:12]),
    .alu_op         (alu_op            ),
    .alu_control    (alu_control       )
);
```

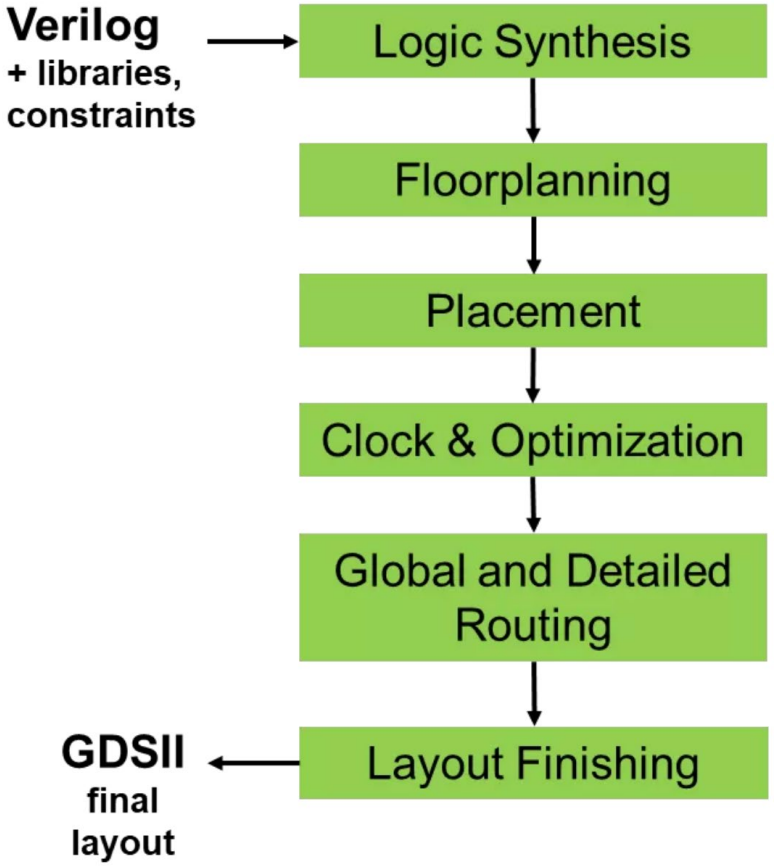
Module Instance

- Do not forget to update the module instance (in cpu.v) after any modification on IO definitions.

Today's session: Single-cycle processor



Digital Backend Flow



The RTL design is synthesized into a gate-level netlist.

Decide on the chip area and set memory macros location in the chip.

The synthesized gates are placed in the chip according to the floorplan.

A clock tree is generated to distribute the clock signal throughout the chip, while minimizing skew and ensuring timing requirements are met.

The global routing involves determining the overall routing paths between the blocks and macros; The detailed routing creates the actual metal interconnects between the gates.

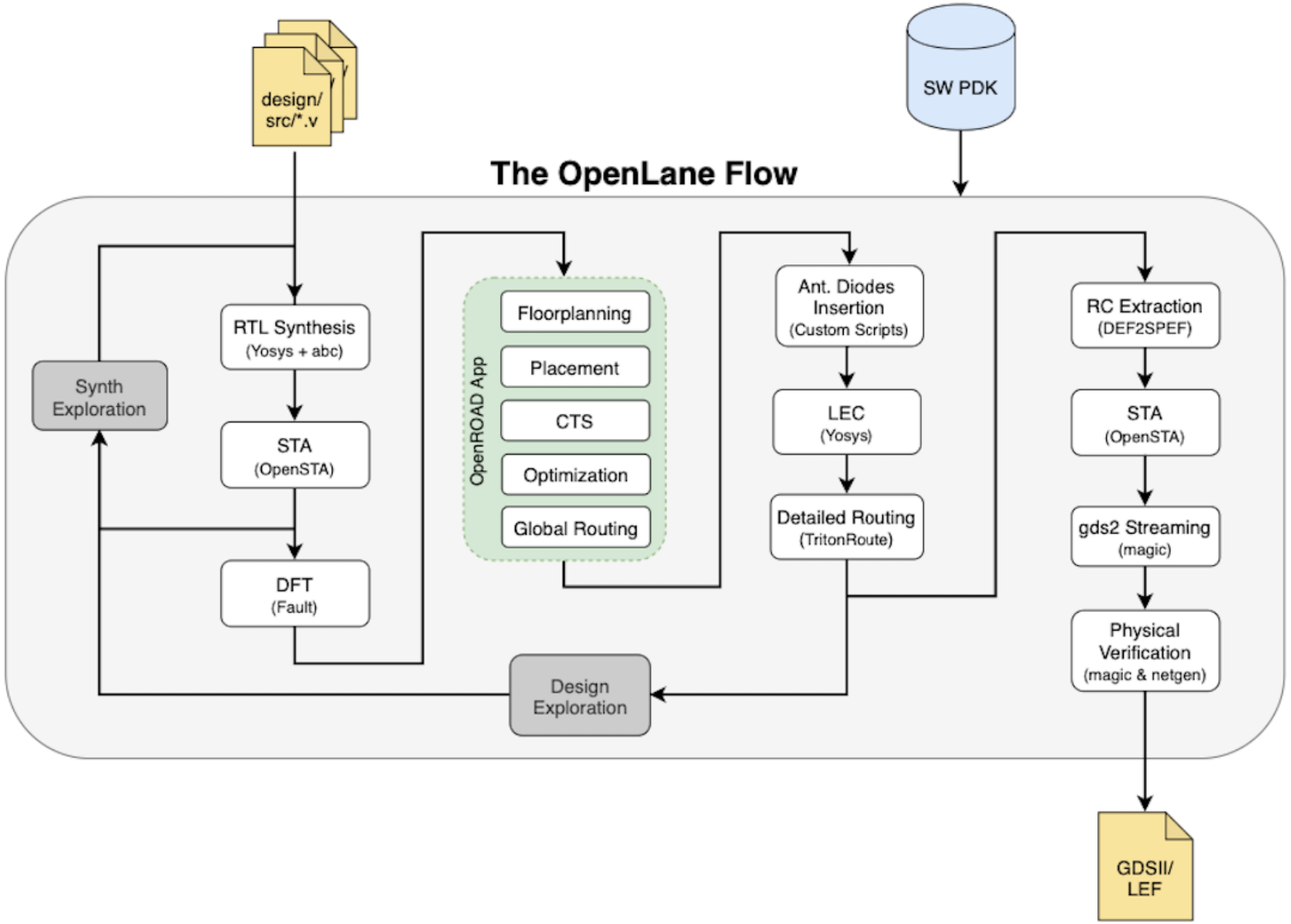
Design rule checking: Once the routing is complete, the design is checked against a set of design rules to ensure that it is manufacturable.

Backend Flow in this exercise

Three key components:

1. Flow tool: OpenLane
2. PDK library: SKY130
3. Memory macros: OpenRAM

1. Automated Backend Flow: OpenLane



OpenLane is an **automated RTL to GDSII flow** based on several components.

More details on: https://openlane.readthedocs.io/en/latest/flow_overview.html

2. Related: open-source PDK -- SKY130

PDK: **P**rocess **D**esign **K**it - information about the process technology

PDK includes

- Process documentation
- Device models
- Layout design rules
- Design libraries: Pre-designed circuit blocks, such as standard cells, memories, and IO cells
- …

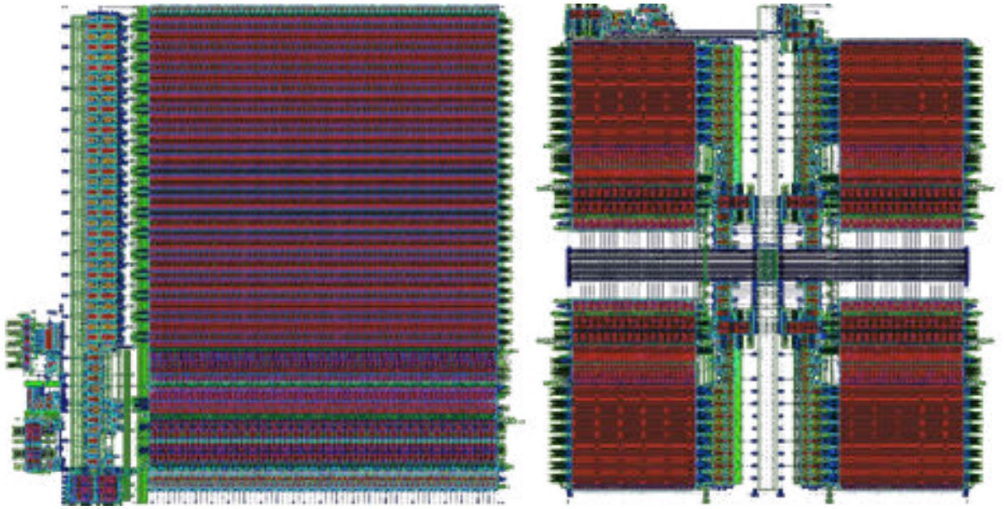
More details on: https://skywater-pdk.readthedocs.io/en/main/index.html

3. Related: open-source SRAM Compiler: OpenRAM

OpenRAM

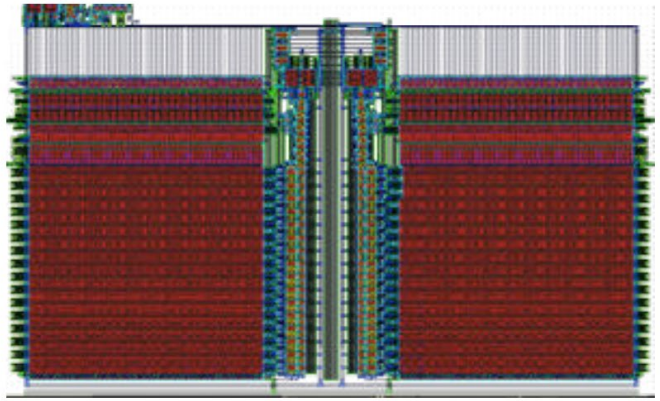
Create SRAM for ASIC design:

- **Layout**
- **Netlists**
- **Timing and power models**
- **Placement and routing models**



2 Kb (1 bank x 32 words x 64 bits)

4 Kb (4 banks x 32 words x 32 bits)



16 Kb (2 banks x 128 words x 64 bits)

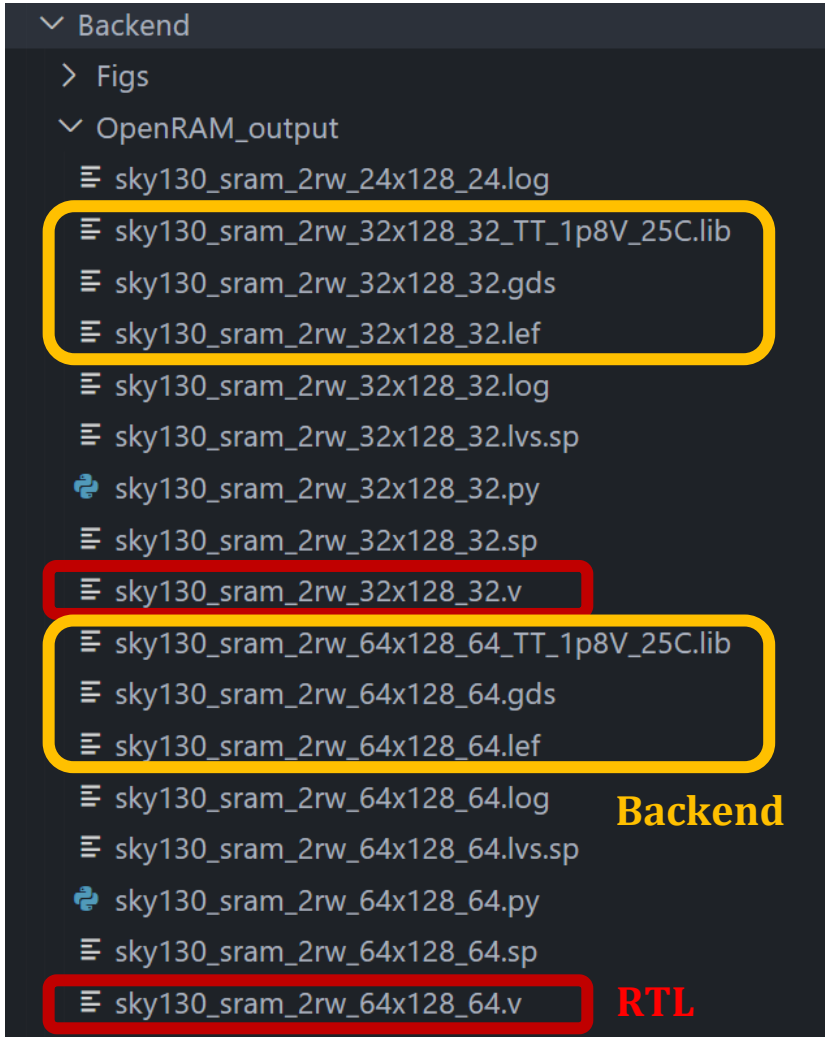
More details on: https://openram.org/

3. Related: open-source SRAM Compiler: OpenRAM

Create SRAM for ASIC design:

- **Layout**
- **Netlists**
- **Timing and power models**
- **Placement and routing models**

In the exercise, the memory macros used by instruction memory and data memory are ***already generated for you****.*



More details on: https://openram.org/

Environment settings + Backend tools installation

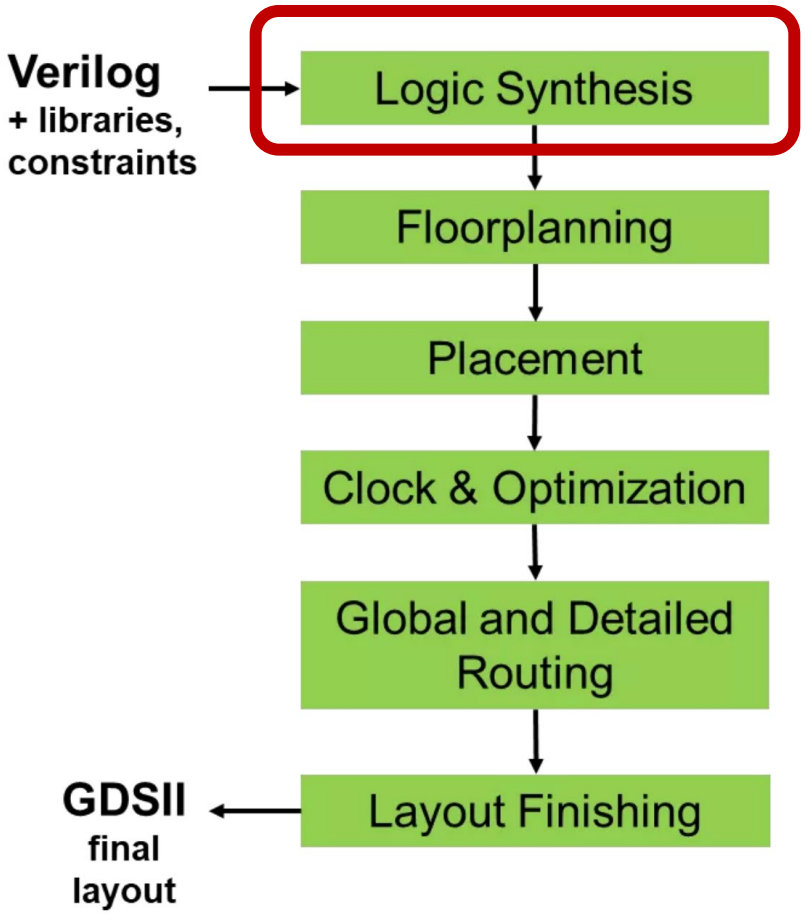
Backend environment setup:

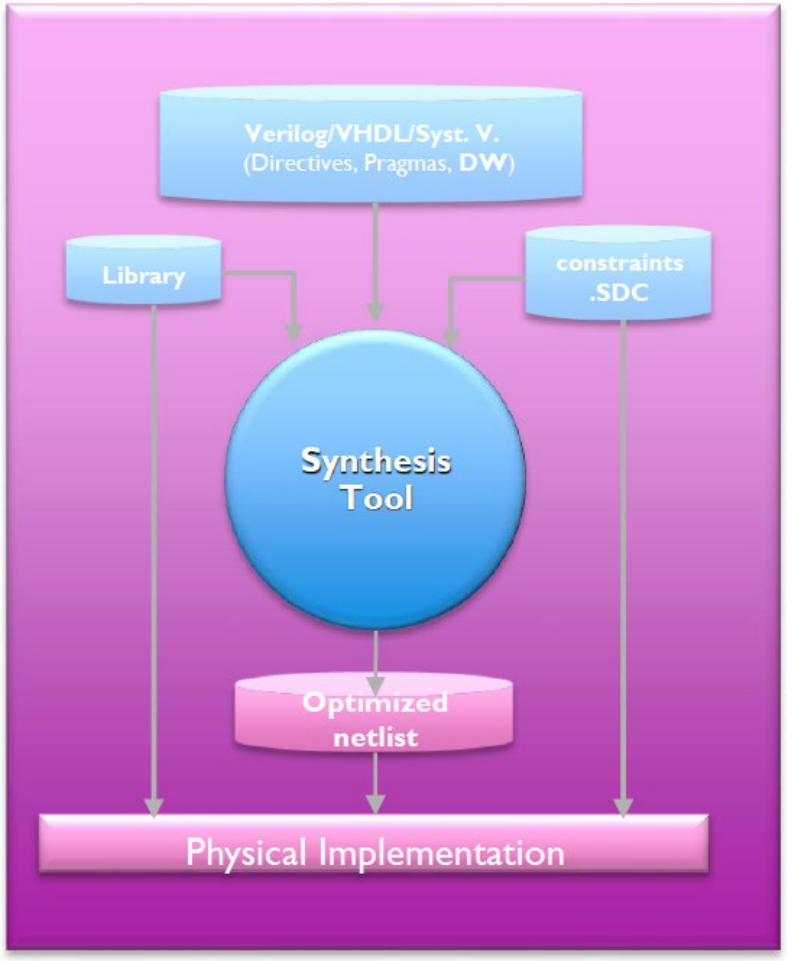
- **Enter the folder: Backend/**
- **In the folder, run the command: source setup.sh. This should setup the backend tools and jupyter.**
- **In the terminal, type the command jupyter-lab. This should introduce you to the website of jupyter notebook.**

PS1: Every time you want to use the backend tools and jupyter, please run the command source setup.sh to check the integrity of the toolchain.

PS2: If you want to use jupyter-lab at home, see this manual on how to open *jupyter notebook remotely. (CA_Documents/Jupyter_VNC_Manual.pdf)*

Synthesis: a critical step in the backend flow





- Pass the RUN CYCLE-ACCURATE SIMULATION and prepare your source code in RTL_SOLUTION* folders.
- Follow the RUN BACKEND FLOW in **session_guide.pdf** and the instructions in **CA_RISCV.ipynb**
- In **Session 1** we only do the **synthesis stage**.

Today's session: task summary

With **session_guide.pdf**

- Study the RUN CYCLE-ACCURATE SIMULATION and RUN BACKEND FLOW
- Follow the TASKS TO BE DONE and fill in the **report.docx**

Copy-paste your finished **/RTL/*.v** into the SOLUTION folders.

- **Obj-1&2** → **RTL_SOLUTION1_simple_program_and_MULT1**
- **Obj-3** → **RTL_SOLUTION2_multiplication_support_MULT2**

- **Note:**

1. We use universal test patterns for fair grading.
2. **Do not modify cpu_tb.v & sky130_sram_2rw.v**
3. **Do not modify *mem_content.txt**