



Birzeit University

Department of Electrical & Computer Engineering

First Semester 2024/2025

HW Design Lab

Project#1 – Physical Design

Name : Anas Naji

ID : 1200231

Section: #2

Instructor : Dr Abdullatif abuissa

Teaching assistant: Eng. Rania Shahwan

## Contents

Objective .....	4
Initial Analysis and Planning.....	5
Design Specification .....	5
Modification and Expansion of Code .....	6
Testing and Debugging .....	6
Simulation Results.....	7
Physical Design Steps .....	7
Synthesis .....	7
Floorplanning and Power Planning.....	8
Placement.....	9
Clock Tree Synthesis (CTS).....	12
Routing .....	15
The Final Design .....	17
Verification and Validation .....	18
References.....	19

Figure 1 : Synthesis codes .....	7
Figure 2 : Shown the circui .....	7
Figure 3 : Floorplanning and Power Planning codes .....	8
Figure 4 : Design layers .....	8
Figure 5 : Ports zoomed in .....	9
Figure 6 codes for Placement.....	9
Figure 7 : Our design .....	10
Figure 8 : : The Site Row & Cell Site on the design.....	10
Figure 9 : final stage of Placement.....	11
Figure 10 :check_legality.....	11
Figure 11 : report_scenarios and report_modes .....	12
Figure 12: report clocks .....	12
Figure 13 : codes for Clock Tree Synthesis (CTS) .....	13
Figure 14 : our design.....	13
Figure 15 : Design.....	14
Figure 16 : Report timing .....	14
Figure 17:Design layout.....	15
Figure 18: After disenable power pin & ground pin.....	16
Figure 19 : Report timing .....	16
Figure 20 : Using report_global_timing & report_clock_qor.....	17
Figure 21: Design.....	17
Figure 22 : cheacking legality .....	18

## Objective

The goal of this project was to gain practical experience in digital and physical design by modifying an existing 8-bit processor design to support 32-bit operations. The project involved updating the data paths, control units, and memory interfaces, as well as performing synthesis, floorplanning, placement, routing, and clock tree synthesis (CTS) to meet timing, power, and area constraints.

## Initial Analysis and Planning

The selected section outlines three main components of the provided 8-bit processor design :

1. ALU (Arithmetic Logic Unit)
  - Performs the core arithmetic and logical operations.
  - Supported operations include:
    1. **Addition (add)**: Adds two numbers.
    2. **Subtraction (sub)**: Subtracts one number from another.
    3. **Logical AND (and)**: Performs a bitwise AND operation.
    4. **Logical OR (or)**: Performs a bitwise OR operation.
    5. **Move (mov)**: Transfers a value from one register to another.
    6. **Load Immediate (loadi)**: Loads a constant value directly into a register.
2. Register File
  - A set of eight 8-bit registers used for storing intermediate values and results of computations.
  - These registers provide inputs to and store outputs from the ALU.
3. Instruction Set
  - Defines the operations the processor can execute, such as:
    1. Arithmetic operations: add, sub.
    2. Logical operations: and, or.
    3. Data transfer operations: mov, loadi.
    4. Control flow instructions: jump (j), branch if equal (beq).
  - Instructions follow a structured format, ensuring consistency in decoding and execution.

## Design Specification

Key changes for the 32-bit processor involved increasing the data width from 8-bit to 32-bit, adjusting the instruction format to accommodate 32-bit operations, and expanding the control unit to decode 32-bit instructions while managing larger registers and buses.

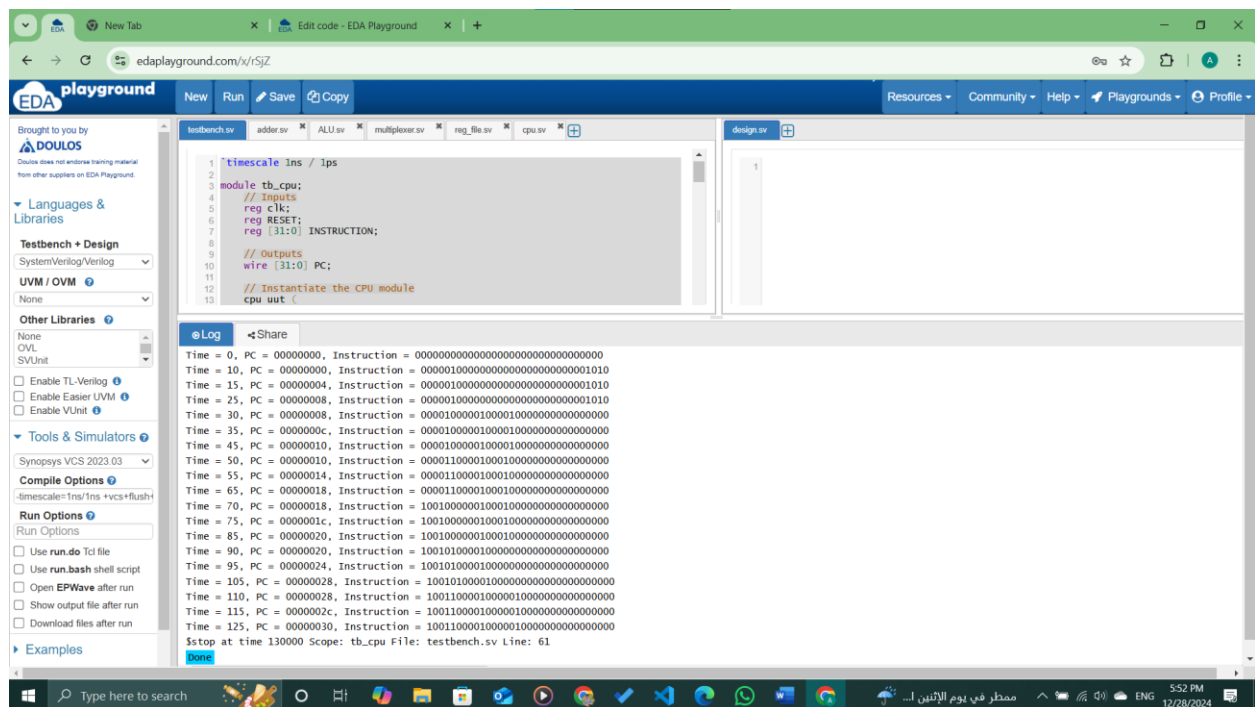
## Modification and Expansion of Code

The data path was expanded by updating all data signals, registers, and buses to 32 bits, allowing for operations on larger data widths. The ALU was adjusted to handle 32-bit inputs effectively, and the instruction decoder was expanded to support the 32-bit instruction format. Additionally, new opcodes were implemented to facilitate 32-bit operations, and control signals were modified to ensure compatibility with the wider data paths and updated instruction set.

## Testing and Debugging

A testbench was written to validate the design's functionality using simulation. The testbench applied a series of 32-bit instructions, including `loadi`, `add`, `and`, `or`, `mov`, and `sub`. Each instruction was executed and verified against expected outcomes. Key highlights from the simulation results include:

- The program counter (PC) updated correctly after each instruction.
- The ALU performed the specified operations, including addition, bitwise AND/OR, and subtraction, without errors.
- The zero flag was set correctly for ALU operations that resulted in a zero output.



```
1 timescale 1ns / 1ps
2
3 module tb_cpu;
4     // Inputs
5     reg clk;
6     reg RESET;
7     reg [31:0] INSTRUCTION;
8
9     // Outputs
10    wire [31:0] PC;
11
12    // Instantiate the CPU module
13    CPU uut (
        .clk(clk),
        .reset(RESET),
        .instruction(INSTRUCTION),
        .pc(PC)
    );
endmodule
```

Simulation Log:

```
Time = 0, PC = 00000000, Instruction = 00000000000000000000000000000000
Time = 10, PC = 00000000, Instruction = 00000100000000000000000000000010
Time = 15, PC = 00000004, Instruction = 00000100000000000000000000000010
Time = 25, PC = 00000008, Instruction = 00000100000000000000000000000010
Time = 30, PC = 00000008, Instruction = 00001000001000010000000000000000
Time = 35, PC = 0000000c, Instruction = 00001000001000010000000000000000
Time = 45, PC = 00000010, Instruction = 00001000001000010000000000000000
Time = 50, PC = 00000010, Instruction = 00001100001000010000000000000000
Time = 55, PC = 00000014, Instruction = 00001100001000010000000000000000
Time = 65, PC = 00000018, Instruction = 00001100001000010000000000000000
Time = 70, PC = 00000018, Instruction = 10010000001000100000000000000000
Time = 75, PC = 0000001c, Instruction = 10010000001000100000000000000000
Time = 85, PC = 00000020, Instruction = 10010000001000100000000000000000
Time = 90, PC = 00000020, Instruction = 10010100001000000000000000000000
Time = 95, PC = 00000024, Instruction = 10010100001000000000000000000000
Time = 105, PC = 00000028, Instruction = 10010100001000000000000000000000
Time = 110, PC = 00000028, Instruction = 10011000010000010000000000000000
Time = 115, PC = 0000002c, Instruction = 10011000010000010000000000000000
Time = 125, PC = 00000030, Instruction = 10011000010000010000000000000000
Stop at time 130000 Scope: tb_cpu File: testbench.v Line: 61
```

## Simulation Results

- Instructions executed correctly, as observed in the EDA Playground logs.
- All tested instructions produced the expected results, confirming the correctness of the design.

## Physical Design Steps

### Synthesis

To do this part we used these commands :

 \*commands.txt - Notepad

File Edit Format View Help

```
1. mkdir -p ./1200231/EXP2
```

```
2. cd ./1200231/EXP2
```

```
3. cp -R /home/iccta/all_labs/dc_EXP3_env/* .
```

-> Then we use this command : "gedit MY\_DESIGN.v" to enter to the file and put our code

```
4. dcnxt_shell
```

```
5. source rm_dc_scripts/dc.tcl
```

```
6. read_ddc results/MY_DESIGN.elab.ddc
```

```
7.start_gui
```

Figure 1 : Synthesis codes

And the result observed is that :

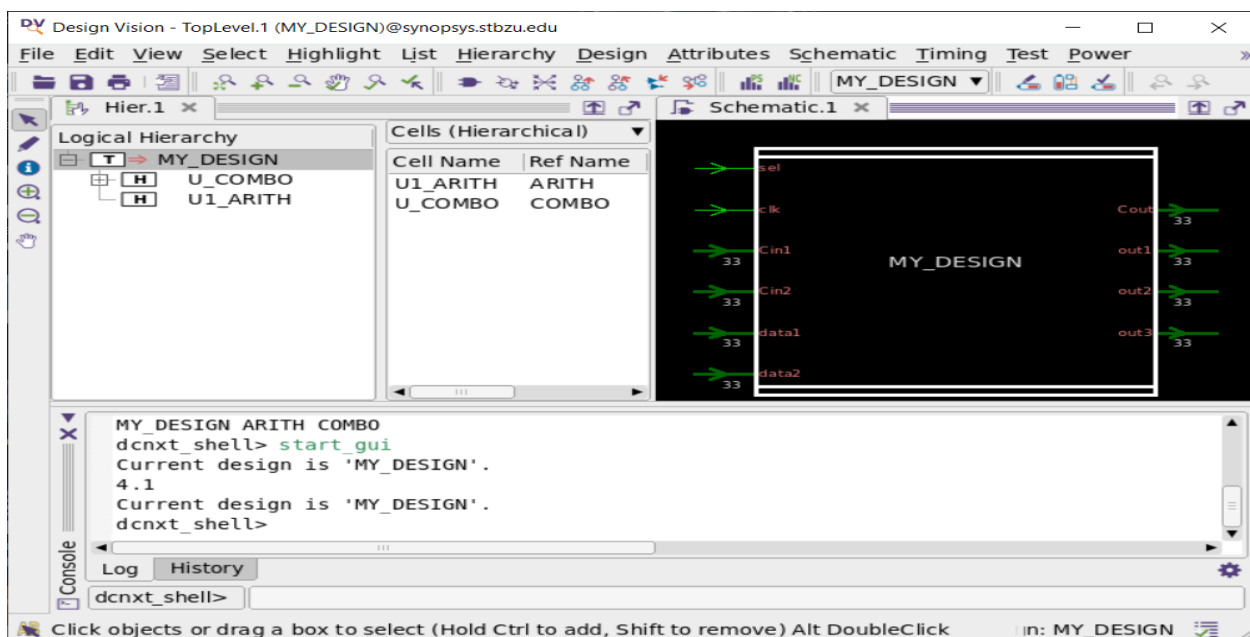


Figure 2 : Shown the circui

## Floorplanning and Power Planning

To do this part we used these commands :

```
commands.txt - Notepad
File Edit Format View Help
1. mkdir -p./1200231/EXP3
2. cd ./1200231/EXP3
3. cp -R /home/iccta/all_labs/dc_EXP4_env/* .
4. ln -s /home/BZU6/1200231/EXP2/results/MY_DESIGN.mapped.v inputs/
5. make init_design
6. icc2_shell
7. open_lib MY_DESIGN.nlib
8. list_blocks
9. open_block MY_DESIGN/init_design.design
10. start_gui
-> then after we worked in the gui window we enter this command : save_block ->
->to Save the block with the power connections
```

Figure 3 : Floorplanning and Power Planning codes

And this is the result that observed after we finish:

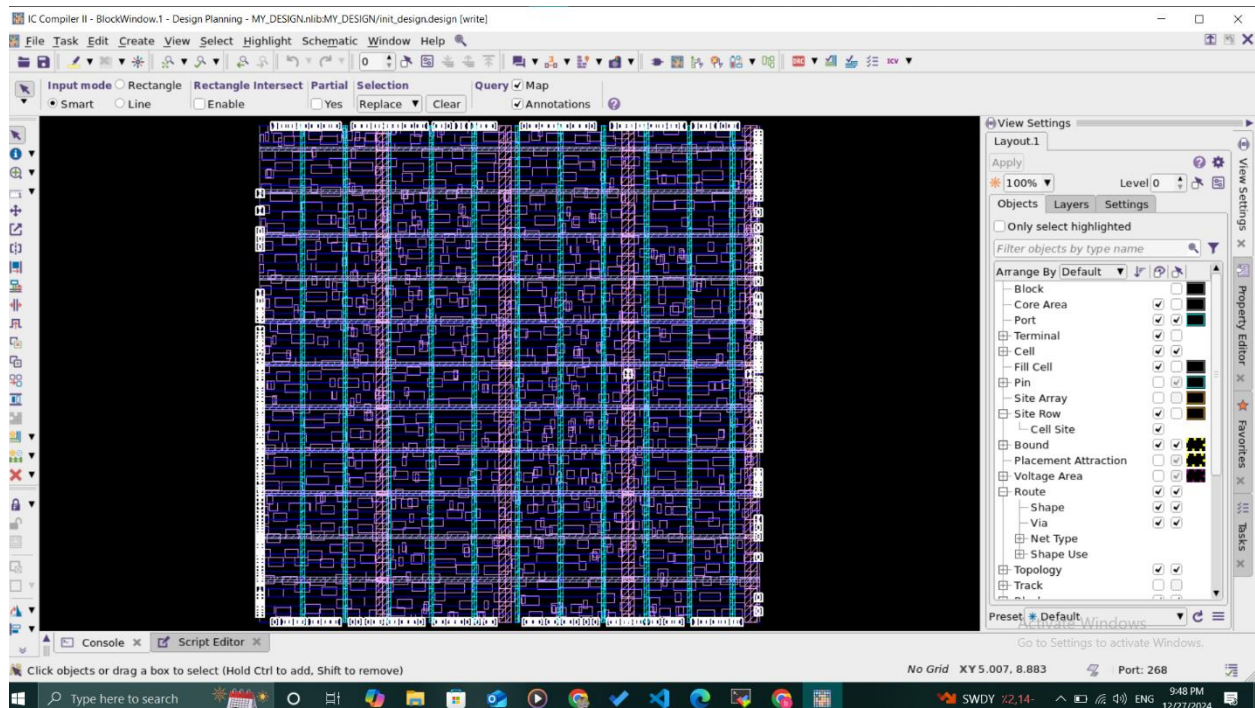


Figure 4 : Design layers



And here is assigned ports zoomed in :

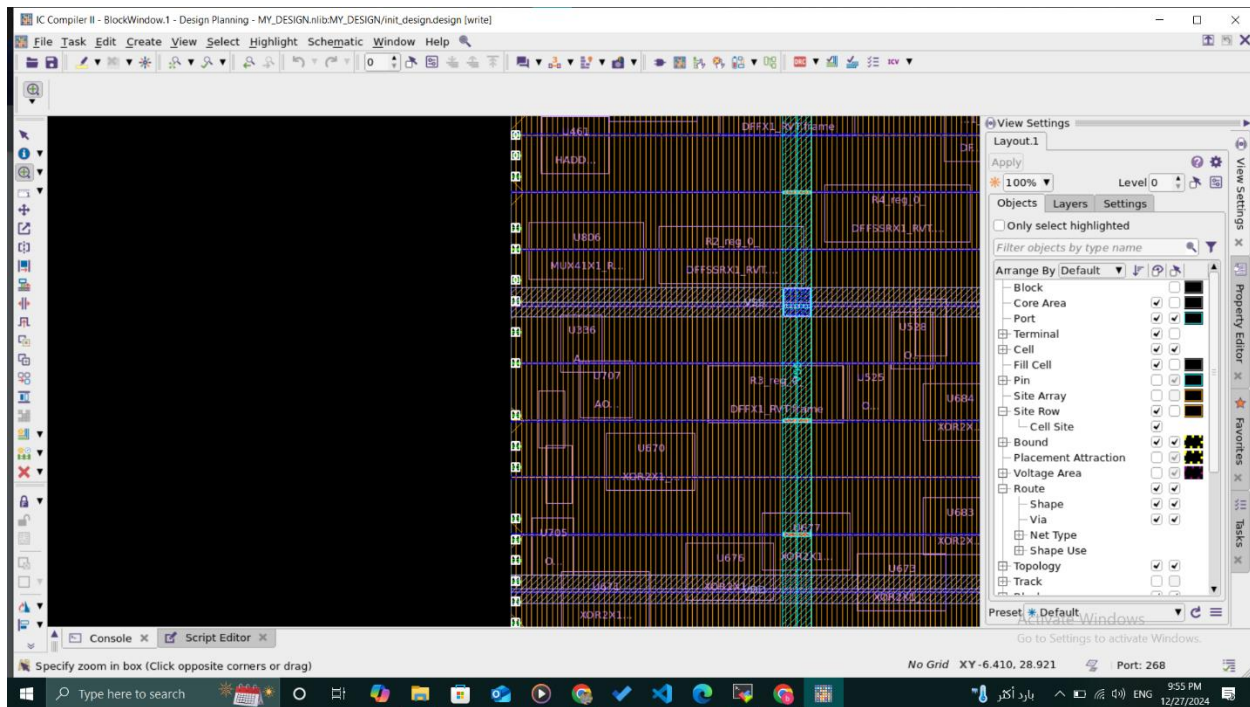



Figure 5 : Ports zoomed in

## Placement

To do this part we used these commands :

 \*commands.txt - Notepad

File Edit Format View Help

1. mkdir -p ./1200231/EXP4
2. cd ./1200231/EXP4
3. cp -R /home/iccta/all\_labs/dc\_EXP5\_env/\* .
4. ln -s /home/BZU6/1200231/EXP2/results/MY\_DESIGN.mapped.v inputs/
5. cp -RPi /home/BZU6/1200231/EXP3/MY\_DESIGN.nlib .
6. make place\_opt
7. icc2\_shell
8. open\_lib MY\_DESIGN.nlib
9. list\_blocks
10. open\_block MY\_DESIGN/init\_design.design
11. start gui

Figure 6 codes for Placement

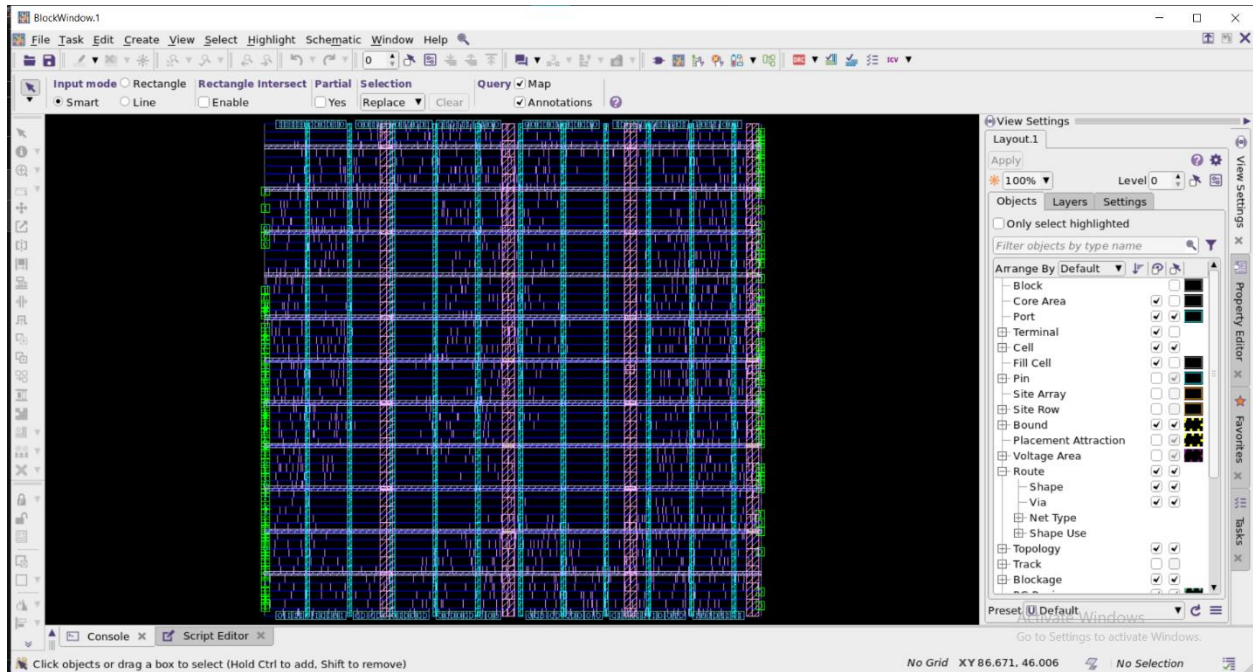


Figure 7 : Our design

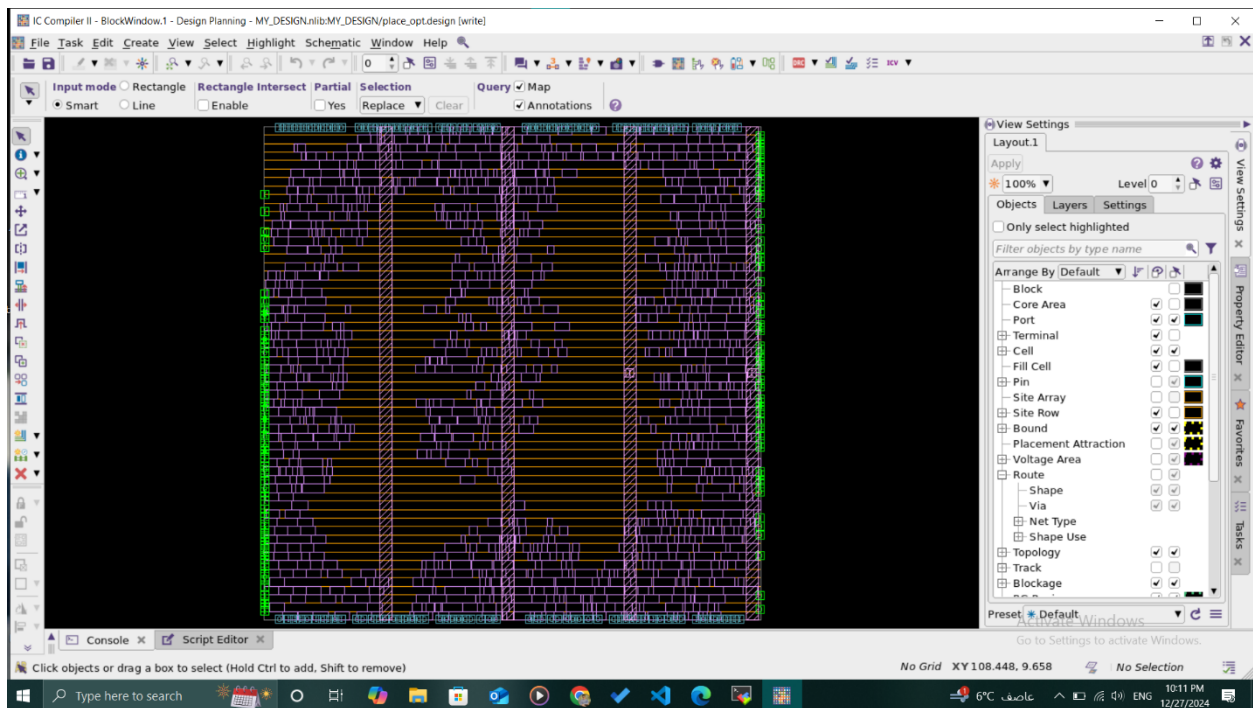


Figure 8 : : The Site Row & Cell Site on the design

And this is the result that observed after we finish:

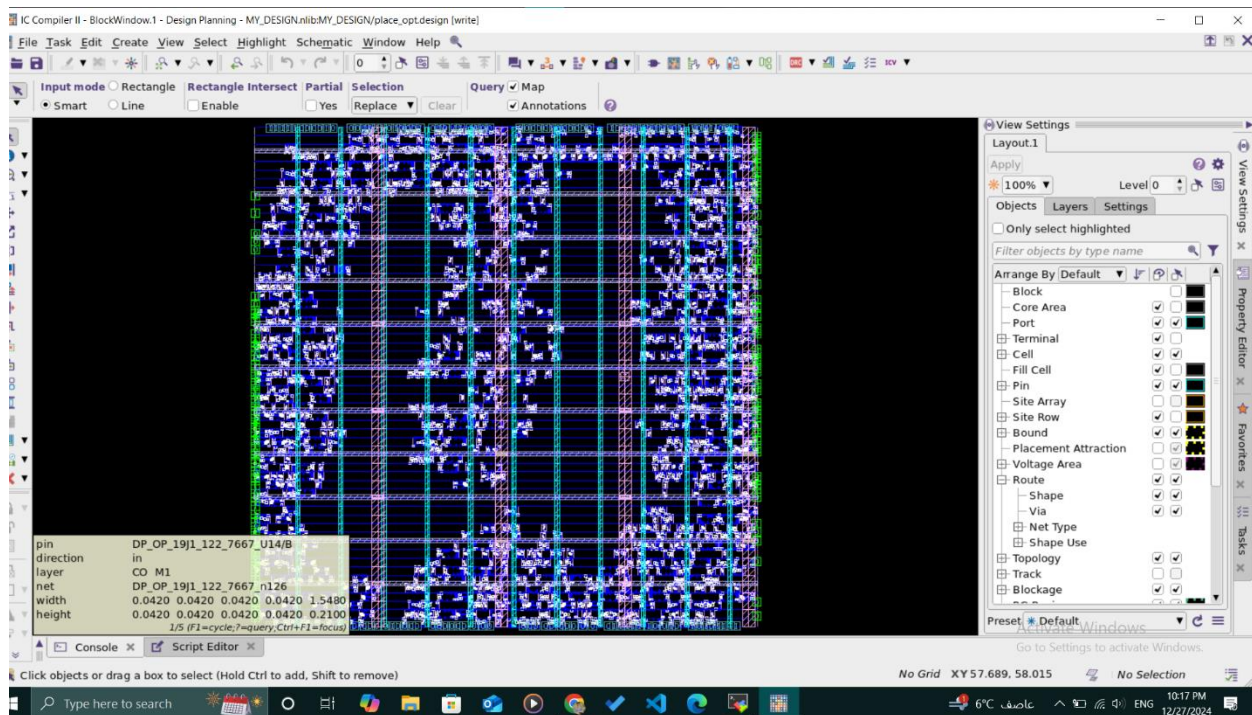


Figure 9 : final stage of Placement

confirm all the settings are read in correctly:

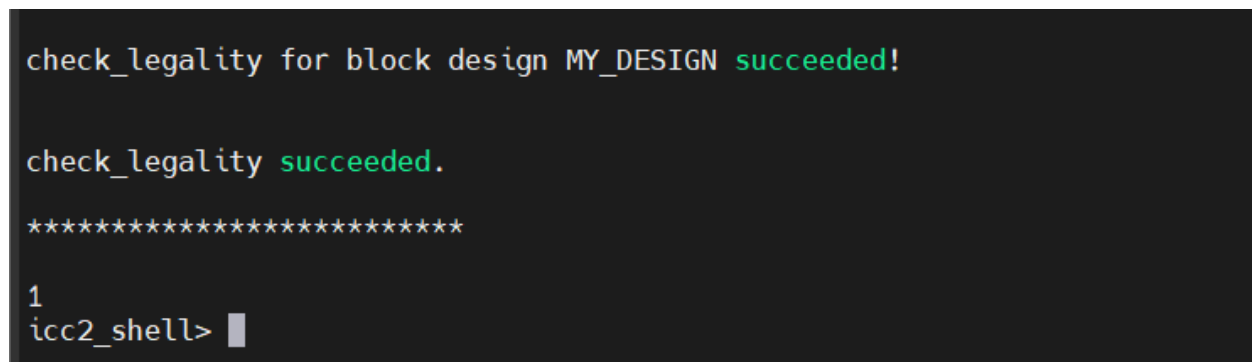


Figure 10 :check\_legality

```

icc2_shell> report_scenarios
*****
Report : scenario
Design : MY_DESIGN
Version: P-2019.03-SP2
Date   : Fri Dec 27 23:05:06 2024
*****

Name      Mode      Corner      Active  Setup  Hold  Leakage Power  Dynamic Power  Max_tran  Max_cap  Min_cap  Cell EM  Signal EM
-----
func::sspg  func      sspg      true    true   false true         true          true       true      true     false   false   false

1
icc2_shell> report_modes
*****
Report : mode
Design : MY_DESIGN
Version: P-2019.03-SP2
Date   : Fri Dec 27 23:05:17 2024
*****

Mode func

Current: true Default: false Empty: false

Scenarios associated with this mode:

Scenario      Corner      Active  Setup  Hold  Leakage Power  Dynamic Power  Max_tran  Max_cap  Min_cap  Cell_em  Signal_em
-----
func::sspg      sspg      true    true   false true         true          true       true      false   false   false

1
icc2_shell>

```

Figure 11 : report\_scenarios and report\_modes

```

icc2_shell> report_clocks
*****
Report : clock
Design : MY_DESIGN
Mode   : func
Version: P-2019.03-SP2
Date   : Fri Dec 27 23:06:20 2024
*****

Attributes:
  p - Propagated clock
  G - Generated clock
  U - Unexpanded generated clock

Clock      Period  Waveform      Attrs      Sources
-----
main_clk    1.00    {0 0.5}      {clk}

1
icc2_shell>


```

Figure 12: report clocks

## Clock Tree Synthesis (CTS)



To do this part we used these commands :

 \*commands.txt - Notepad

File Edit Format View Help

1. mkdir -p ./1200231/EXP5
2. cd ./1200231/EXP5
3. cp -R /home/iccta/all\_labs/dc\_EXP6\_env/\* .
4. ln -s /home/BZU6/1200231/EXP2/results/MY\_DESIGN.mapped.v inputs/
5. cp -Rpi /home/BZU6/1200231/EXP4/MY\_DESIGN.nlib .
6. make clock\_opt\_opto
7. icc2\_shell
8. open\_lib MY\_DESIGN.nlib
9. list\_blocks
10. open\_block MY\_DESIGN/init\_design.design
11. start\_gui

Figure 13 : codes for Clock Tree Synthesis (CTS)

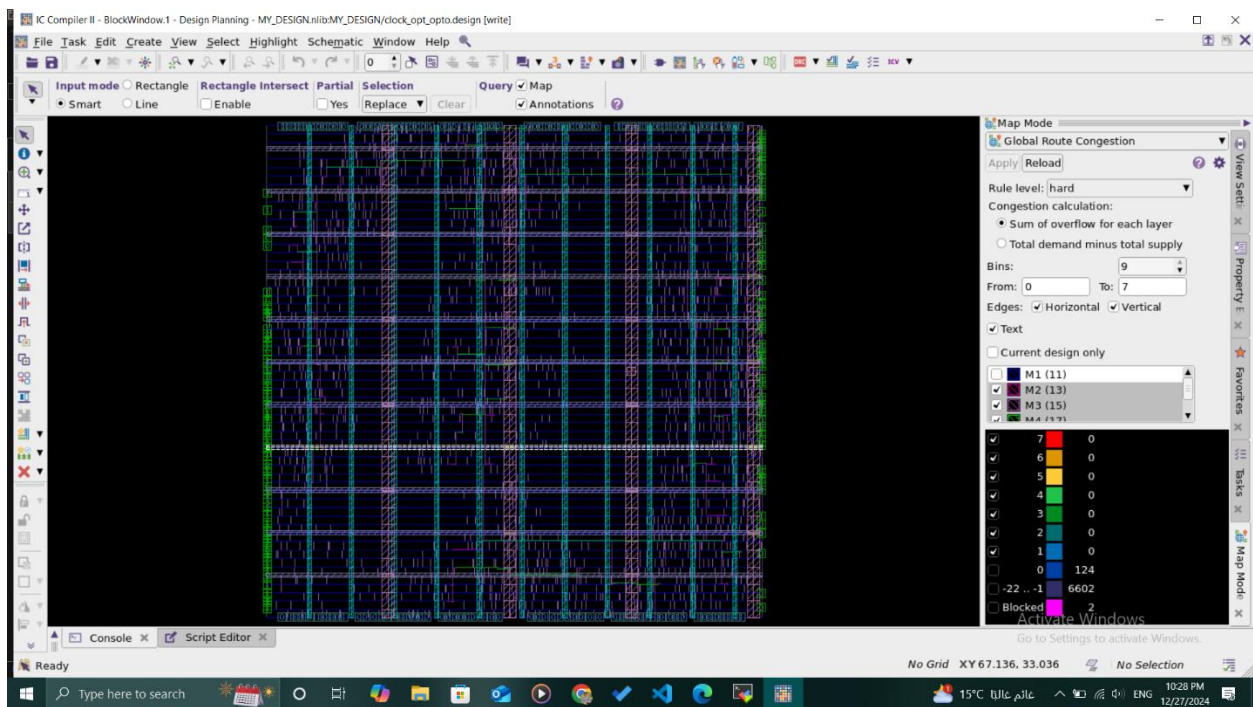


Figure 14 : our design

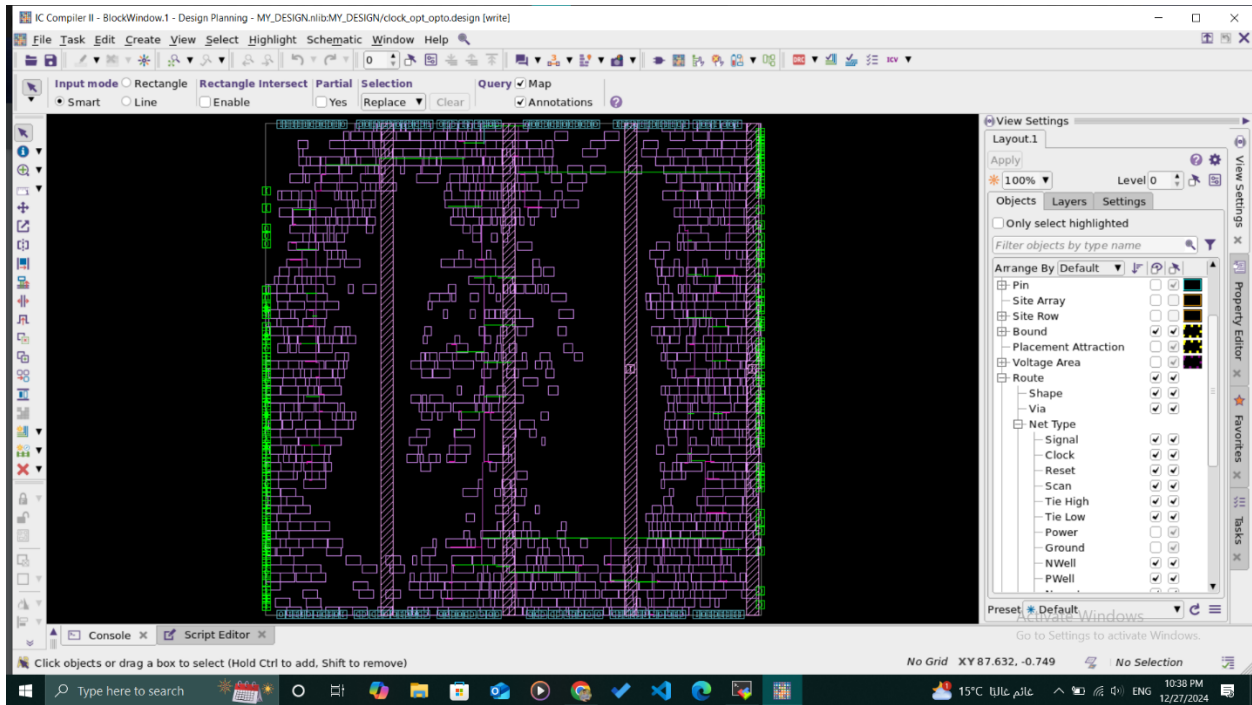


Figure 15 : Design

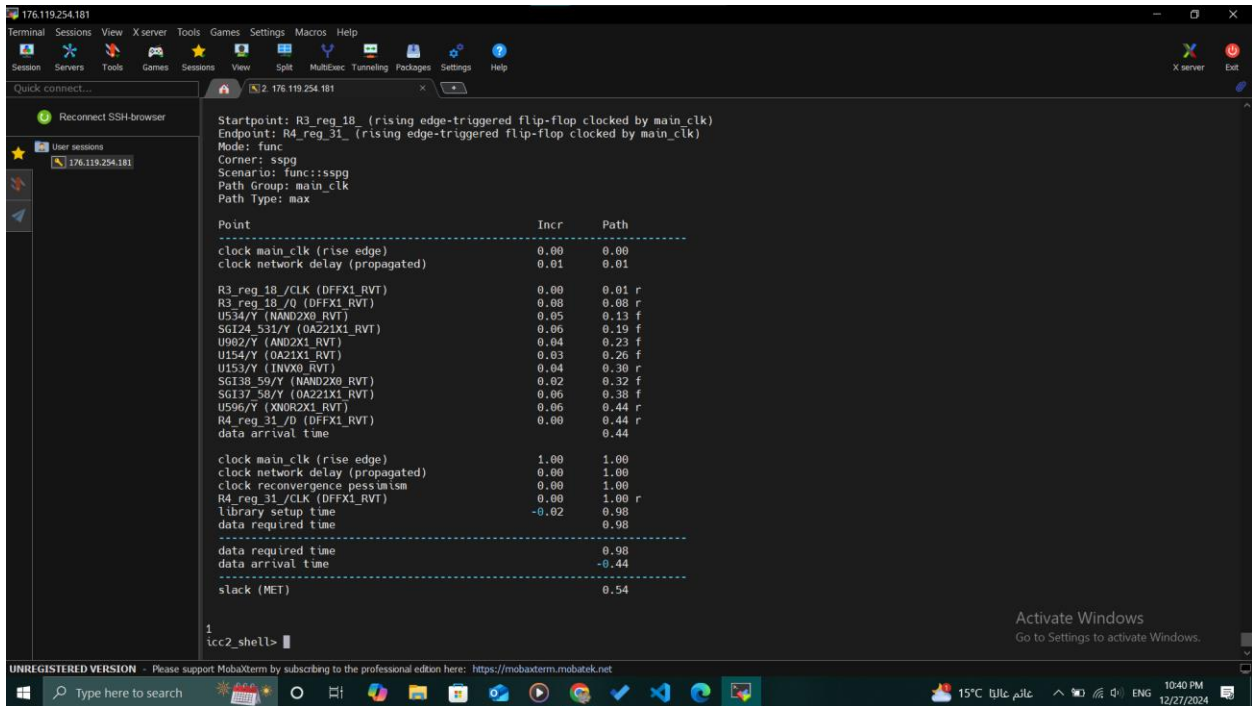


Figure 16 : Report timing

## Routing

To do this part we used these commands :



commands.txt - Notepad

File Edit Format View Help

```
1. mkdir -p./1200231/EXP6
2. cd ./1200231/EXP6
3. cp -R /home/iccta/all_labs/dc_EXP7_env/* .
4. ln -s /home/BZU6/1200231/EXP2/results/MY_DESIGN.mapped.v inputs/
5. cp -RPi /home/BZU6/1200231/EXP5/MY_DESIGN.nlib .
6. make route_opt
7. icc2_shell
8. open_lib MY_DESIGN.nlib
9. list_blocks
10. open_block MY_DESIGN/init_design.design
11. start_gui
```

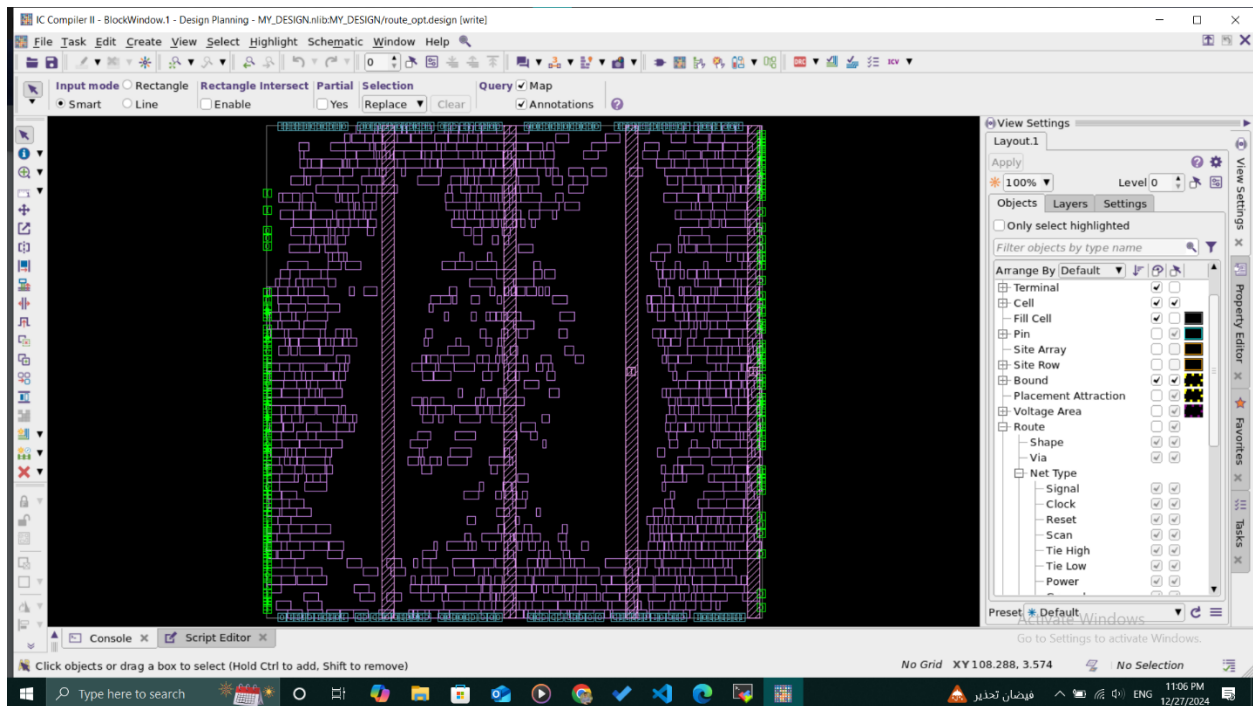


Figure 17: Design layout

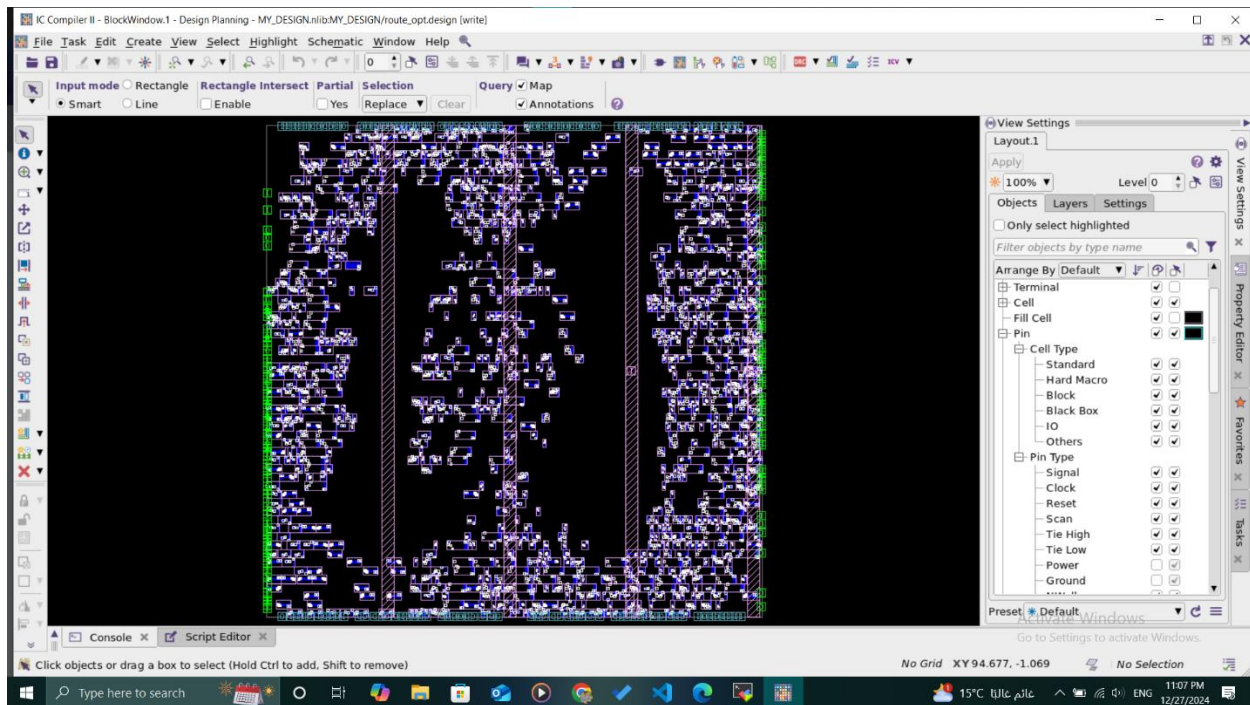


Figure 18: After disable power pin & ground pin

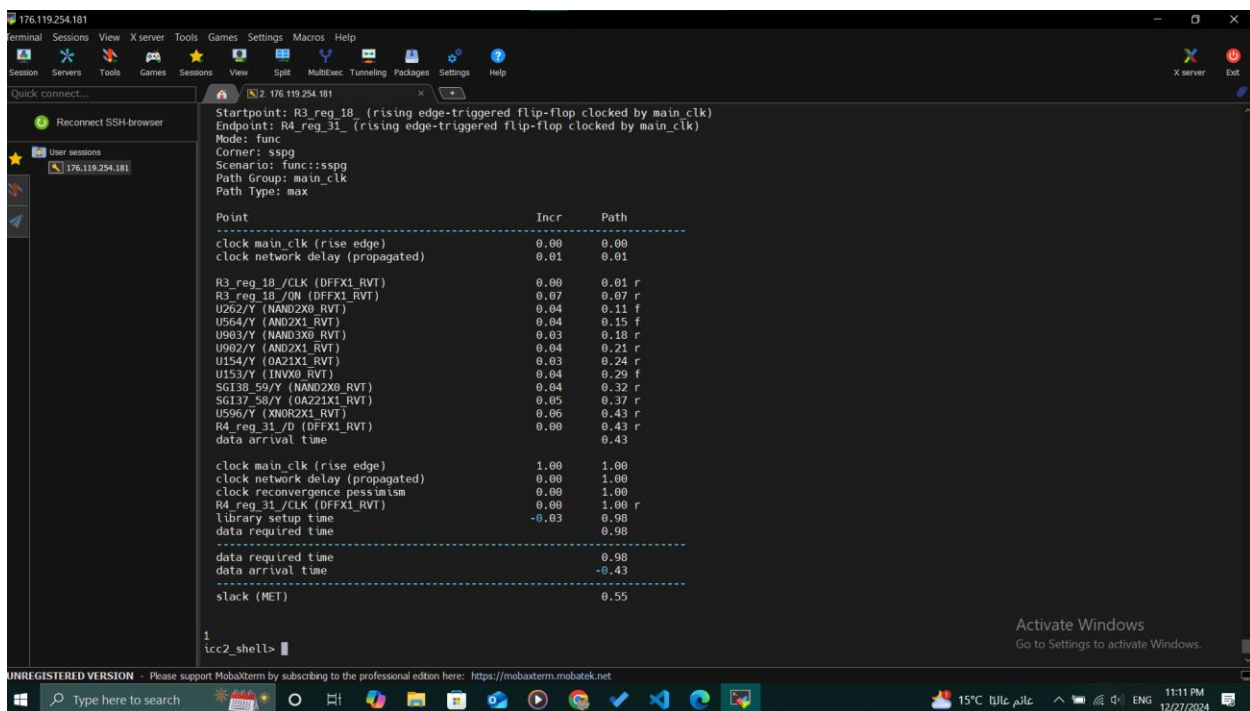


Figure 19 : Report timing



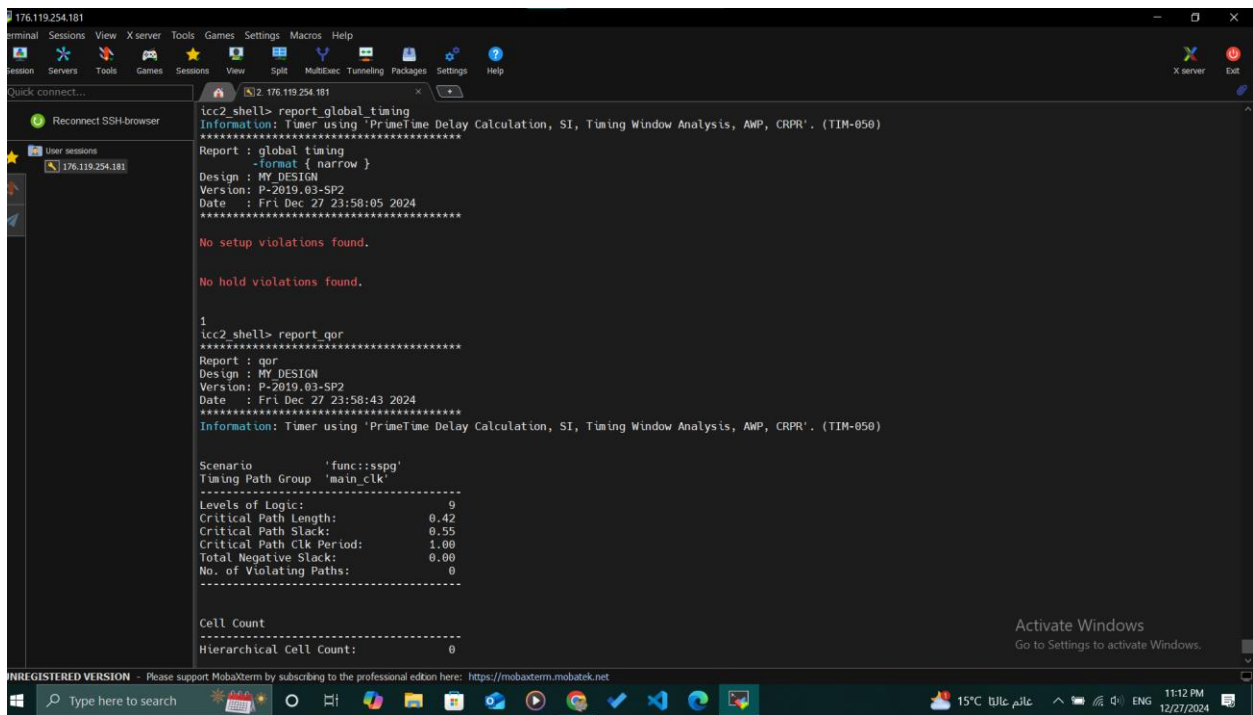


Figure 20 : Using report\_global\_timing & report\_clock\_qor

## The Final Design

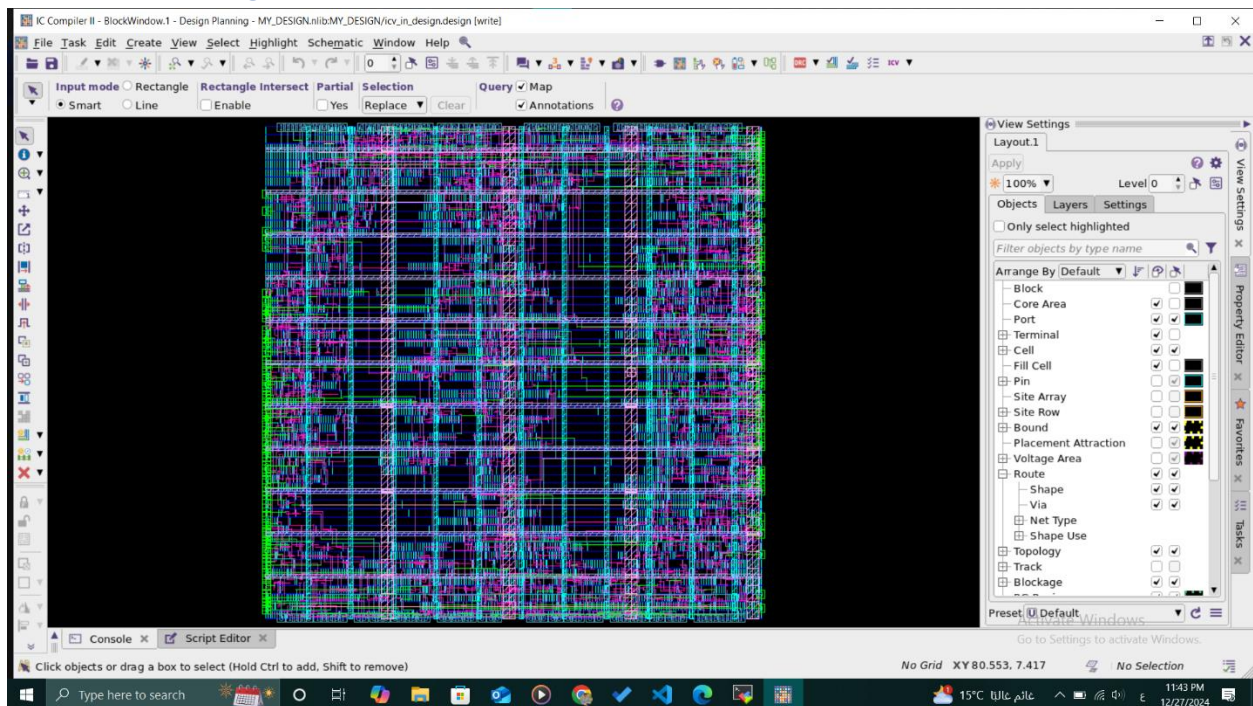


Figure 21: Design

```
176.119.254.181
terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect...
Reconnect SSH-browser
User sessions
176.119.254.181
0 0 0 A cell has the wrong variant for its location.
0 0 0 A cell is not aligned with a site.
0 0 0 A cell is not aligned with the base site.
0 0 0 A cell is not aligned with an overlaid site.
0 0 0 A cell has an illegal orientation.
0 0 0 A cell spacing rule is violated.
0 0 0 A spacing rule is violated in a row.
0 0 0 A spacing rule is violated between adjacent rows.
0 0 0 A cell violates vertical abutment rule.
0 0 0 A cell violates metal spacing rule.
0 0 0 A layer rule is violated.
0 0 0 A layer VTH rule is violated.
0 0 0 A layer OD rule is violated.
0 0 0 A layer OD max-width rule is violated.
0 0 0 A layer ALL_OD corner rule is violated.
0 0 0 A layer max-vertical-length rule is violated.
0 0 0 A layer TPO rule is violated.
0 0 0 Filler cell insertion cannot satisfy layer rules.
0 0 0 A cell is in the wrong region.
0 0 0 A cell is outside its hard bound.
0 0 0 A cell is in the wrong voltage area.
0 0 0 A cell violates an exclusive movebound.
0 0 0 Two cells violate cts margins.
0 0 0 Two cells violate coloring.
check_legality for block design MY_DESIGN succeeded!
check_legality succeeded.
*****
1
icc2_shell>
```

Figure 22 : checking legality

## Verification and Validation

The verification and validation phase of the project confirmed that the design met all timing, power, and layout constraints. Static Timing Analysis (STA) showed no timing violations, ensuring the design operated correctly at the target frequency. Power analysis revealed that both leakage and dynamic power consumption were within acceptable limits, with optimizations reducing unnecessary switching activity. Design Rule Checks (DRC) and Layout Versus Schematic (LVS) validation confirmed that the layout adhered to fabrication rules and matched the intended schematic. Post-layout simulations further validated the functionality and timing of the design, with all test cases producing the expected results, demonstrating the overall correctness and robustness of the processor.

## References

- [1]. [Synopsys | EDA Tools, Semiconductor IP and Application Security Solutions](#)
- [2] [Placement | Physical Design | VLSI Back-End Adventure \(vlsi-backendadventure.com\)](#)
- [3] <https://vlsidesignbasic.com/floorplan-basic/>
- [4] <https://myvlsibasics.blogspot.com/2019/10/floorplan-vlsi.html>
- [5] <https://siliconvlsi.com/power-planning/>
- [6] [Introduction to Digital VLSI Design | by Yu | LazyYu's Blog | Medium](#)
- [7] [PPT - Parallel Algorithms for VLSI Routing PowerPoint Presentation, free download - ID:1691807 \(slideserve.com\)](#)
- [8] <https://zhuanlan.zhihu.com/p/164919161>