


The Andrew and Erna Viterbi Faculty of
ELECTRICAL & COMPUTER
ENGINEERING

# Customization of RISCV Code and Backend Design of a RISCV Processor Project report

**Students:**

Muhammad Biadsy

Omar Sharafy

**Supervised By:**

Amnon Stanislavsky

**Winter Semesters 2024/25**

# Contents

# Abstract

Our primary objective in this project is to modify and prepare the pipelined RISC-V processor, as taught in the “Digital Systems & Computer Structure” course, for a VLSI backend flow and potential chip fabrication while improving some of its functionally and adding debug capabilities.

The processor design provided in the course was intended for simulation and educational purposes only and is not synthesizable in its original form. To prepare it for fabrication, we made several structural and functional changes.

These include replacing the non-synthesizable memory initialization method with an external memory interface, switching to area-efficient compiled memories, and later improving the memory loading method and functionally of the module we worked on. and adding debugging capabilities that allow access to internal registers after fabrication.

As part of this effort, we also improved the Verilog implementation to support higher clock frequencies and better performance in the physical design flow.

On the path to achieving our goal, we modified and verified the RTL, followed by a backend flow including synthesis, scan insertion, logical equivalence checking, floor planning, power grid design, placement, clock tree synthesis, routing, and timing analysis.

# 1. Introduction

## 1.1 Project Goals and Requirements

Transit our current pipelined RISC-V module for implementation using Tower's 18nm RAM technology, reaching layout stage, with the following design goals:

- Maximumzing area & power efficiency.
- Optimizing chip memory.
- Create memory loading method.
- Debug capabilities for accessing internal registers post-fabrication.
- Optimizing chip Frequency.

## 1.2 Alternative Solutions

The RISC-V processor we worked with can utilize FPGA-based memories and using the control unit as a debug tool to monitor internal signals. It also leverages the initial lines of code to load specific content into the memory. While this approach enables functional testing and debugging during development, it presents several limitations.

FPGA memories are not efficient in terms of area and power consumption, and the Verilog code used to integrate them is not optimal.

Additionally, the original design imposes restrictions on accessible memory addresses, which limits flexibility and scalability in more advanced use cases.

## 1.3 Selected Solution

Using SRAM memories (Designed by Tower), which are both area & power-efficient, allowing simultaneous read and write operations to multiple addresses, and making the chip synthesizable.

A memory loading method will be added, to allow memory initialization and testing post-fabrication.

Additionally, creating a dedicated debugging unit, separate from the control unit, to provide additional debugging capabilities & accessing internal registers after fabrication, one register at a time, while halting the processor.

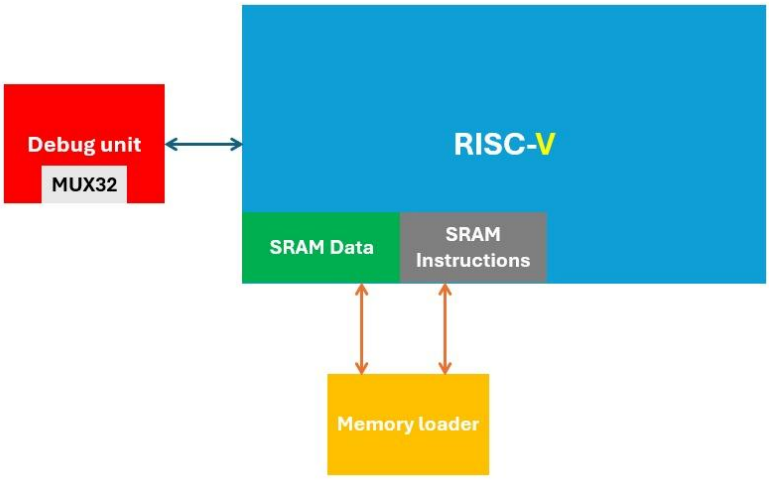



Figure 1: top-level description of the solution

# 2. Verilog Implementation's

## 2.1 SRAM & memory loading method Integration into RISCV

Both the instruction & data memories in the original RISC-V design had to be replaced.

However, the selected SRAMs do not support direct memory initialization during simulation. This limitation prevented the loading of test programs and data, effectively blocking the ability to verify and test the RISC-V core in a simulated environment.

To overcome this, a custom memory loading mechanism was implemented,
Allowing the loading of both instruction and data memories at runtime via an external interface or testbench control, by enabling pre-execution memory initialization.

this solution ensures full support for simulation-based testing, preserving the functional validation and debugging capabilities of the RISC-V processor.

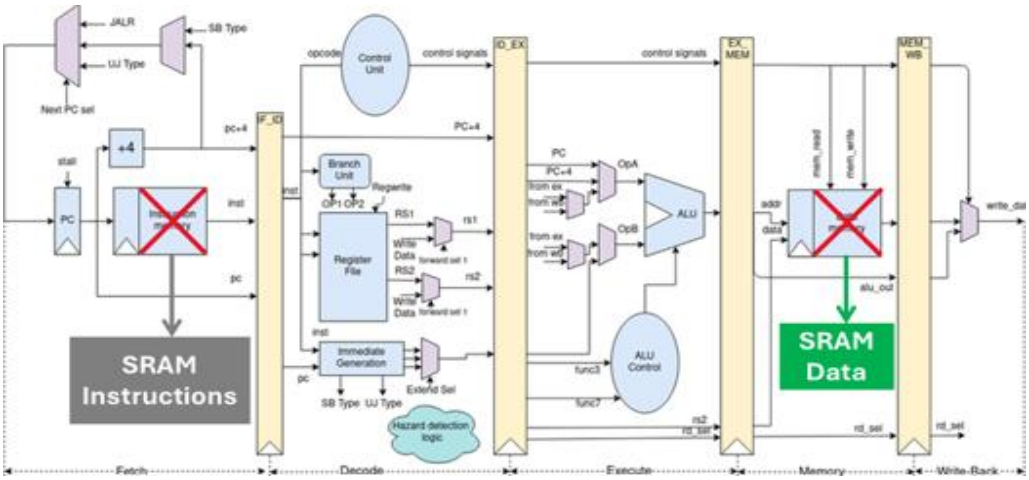



Figure 2: SRAM changes

### 2.1.1 Instruction memory changes

In the used RISC-V, an instruction is a vector of 32 bits.

4 Fpga RAMS on chip (65536*8) were replaced by one tower RAM (2048*32), thus reducing wires and simplifying logic (less wires & eliminating the need of slicing of current instruction address & joining the sliced instruction parts from all Fpgas) in the processor.

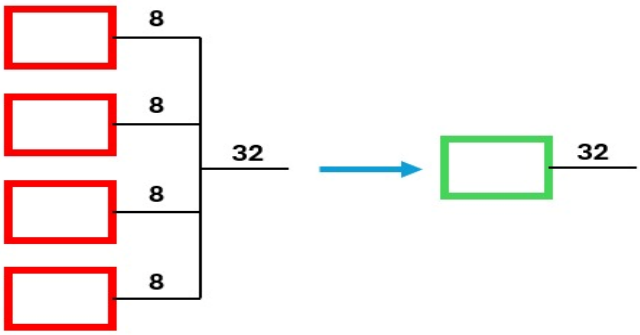



Figure 3: Simplified Diagram of Instruction Memory Modifications

### 2.1.2 Data memory changes

In the implemented RISC-V design, load and store operations for bytes, half-words, and full words are fully supported.

However, it is important to note that correct and expected processor behaviour is guaranteed only when memory addresses used for these operations are word-aligned—that is, the address must be divisible by 4 (address % 4 == 0).

Misaligned accesses may lead to undefined behaviour or require additional hardware handling, which was not implemented in this design.

The four original FPGA memory blocks (each 65,536 × 8) were replaced with four Tower 18nm SRAM blocks (each 4,096 × 8), ensuring continued support for the previously mentioned memory operations.

To obtain full functionality Verilog logic was implemented, this logic extends the processor's capabilities to correctly handle load and store operations for all addressable memory locations—including those that are not word-aligned (i.e., addresses not divisible by 4). As a result, the processor now fully supports byte-, half-word-, and word-level memory access across the entire memory space.

The added logic **Reorders memory block addresses and data** accordingly, dynamically mapping each byte to the correct SRAM address and block, the least significant two bits of the address determine the alignment.

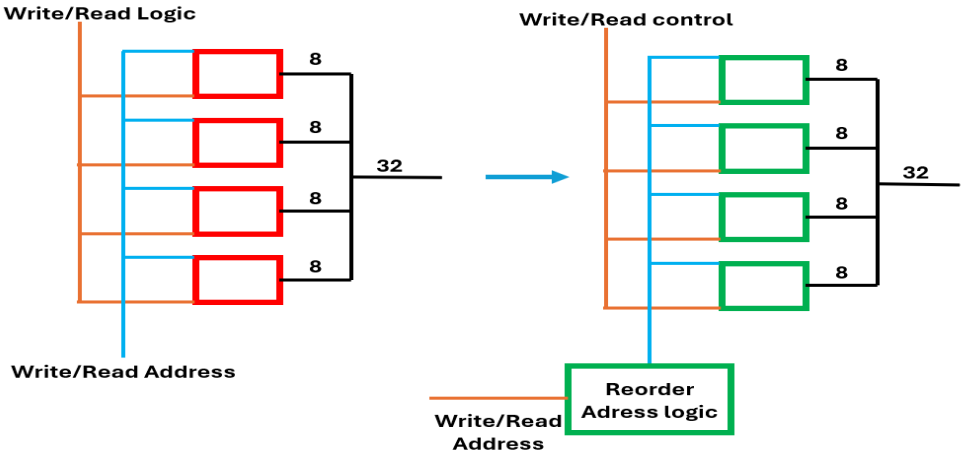



Figure 4: Simplified Diagram of data Memory Modifications

## 2.1.3 Memory Loading Mechanism

As explained previously, Tower SRAMs do not support direct memory initialization during simulation, additionally, in the real world, post-fabrication, it is desirable to be able to load memory onto the chip to run benchmarks and tests.

A memory loading mechanism was implemented, utilizing three 32-bit vectors to specify the data and instructions to be loaded—two vectors for data and one for instructions. Additionally, three vectors were used to designate the addresses where the data and instructions would be loaded, along with two control wires to manage the loading process.

When one of the control wires is set, the processor halts and enters a “loading” state, where it loads the provided data to the specified addresses every cycle.

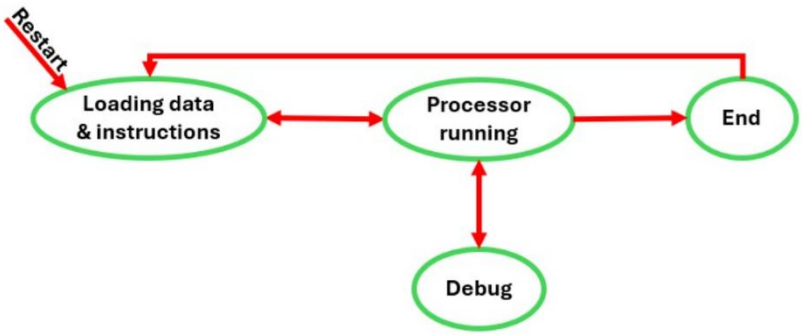



Figure 5: Simplified Diagram of processor states

## 2.2 Debug unit Integration into RISCV

In real-world chip design, post-fabrication testing and debugging are critical phases that require dedicated hardware support.

To address this need, a **debug unit logic** was integrated into the processor architecture, providing visibility into the internal operation of the system.

In this design, **29 crucial observation points** were selected across different pipeline stages and control/data paths, these points were connected to a **multiplexer-based system (32-way MUX)** controlled externally, allowing a user to dynamically select and monitor specific signals, enabling the inspection of key Datapath & memory elements such as the ALU result, fetched opcodes, register values, memory addresses, and data being read or written.

These points include the program counter (PC_debug), control and data signals from all five pipeline stages (Fetch, Decode, Execute, Memory, and Write-back), memory interfaces, register file accesses, and the result from the write-back stage. The unit also tracks instruction decoding fields such as opcode, Funct3, and Funct7, propagating them through the pipeline via registered outputs.

The unit incorporates a **five-stage pipeline register tracing mechanism** for the opcode and function fields, allowing real-time observation of how an instruction moves through the pipeline. The outputs are exposed externally to provide full visibility into the instruction flow and control signal evolution, which is invaluable for debugging complex behaviours like hazards, control flow errors, or incorrect memory operations.

The enable debug signal allows an external user to activate debug mode and halt the processor, freezing its state for inspection. This is especially useful for post-silicon benchmarking and fault diagnosis. Combined with the memory loading mechanism, the debug unit provides a reliable post-fabrication interface for effective testing and validation.

The full logic implementation can be found in Design/DebugUnit.sv file.

### 2.2.1 Debug connections and signals in the RISC-V

The following table maps each DebugSel value to a specific internal signal from various stages of the RISC-V pipeline. It highlights the critical points in the processor that are exposed to the outside world, enabling internal observation and effective debugging.

Unspecified DebugSel values are assigned a zero-vector padded to 32 bits, included to allow simplifying the multiplexer design (32-way mux) in the debug unit.

| DebugSel | Output Source Signal | Description / Notes |
|---|---|---|
| **0** | {25'b0, opcodeFetch} | Opcode from Fetch stage (padded to 32 bits) |
| **1** | {25'b0, opcodeDecode} | Opcode form the operation in Decode stage (padded to 32 bits) |
| **2** | {25'b0, opcodeExecute} | Opcode form the operation in Execute stage (padded to 32 bits) |
| **3** | {25'b0, opcodeMem} | Opcode form the operation in Memory stage (padded to 32 bits) |
| **4** | {25'b0, opcodeWb} | Opcode form the operation in Writeback stage (padded to 32 bits) |
| **5** | {25'b0, Funct7Decode_Dout} | Funct7 form the operation in Decode stage (padded to 32 bits) |
| **6** | {29'b0, Funct3Decode_Dout} | Funct3 form the operation in Decode stage (padded to 32 bits) |
| **7** | {25'b0, Funct7Execute_Dout} | Funct7 form the operation in Execute stage (padded to 32 bits) |
| **8** | {29'b0, Funct3Execute_Dout} | Funct3 form the operation in Execute stage (padded to 32 bits) |
| **9** | {25'b0, Funct7Mem_Dout} | Funct7 form the operation in Memory stage (padded to 32 bits) |
| **10** | {29'b0, Funct3Mem_Dout} | Funct3 form the operation in Memory stage (padded to 32 bits) |
| **11** | {25'b0, Funct7Wb_Dout} | Funct7 form the operation in Writeback stage (padded to 32 bits) |
| **12** | {29'b0, Funct3Wb_Dout} | Funct3 form the operation in Writeback stage (padded to 32 bits) |
| **13** | {23'b0, PC_debug} | Program Counter (padded to 32 bits) |
| **14** | FAmux_Result_debug | Forwarding A MUX result |
| **15** | SrcB_debug | Source B debug |
| **16** | {31'b0, PcSel_debug} | PC select signal (padded to 32 bits) |
| **17** | {23'b0, BrPC_debug} | Branch PC (padded to 32 bits) |
| **18** | ALUResult_debug | ALU result |
| **19** | {28'b0, Operation} | ALU operation code (padded to 32 bits) |
| **20** | {23'b0, addr} | Memory address (padded to 32 bits) |
| **21** | wr_data | Data written to memory |
| **22** | rd_data | Data read from memory |
| **23** | {31'b0, wr} | Register written to number (padded to 32 bits) |
| **24** | {31'b0, rd} | Register read from number (padded to 32 bits) |
| **25** | {27'b0, reg_num} | Register number (padded to 32 bits) |
| **26** | reg_data | Data read from register |
| **27** | {31'b0, reg_write_sig} | Register writes enable signal (padded to 32 bits) |
| **28** | WB_Data | Write-back data |

# 3. Simulation results

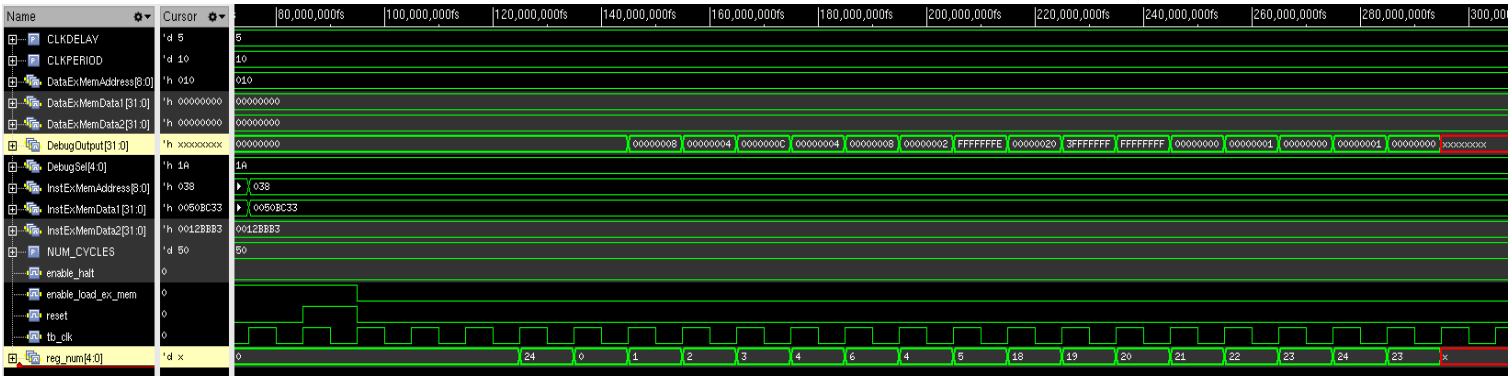
To ensure that the new design behaved correctly, multiple tests were done using simulations.

## 3.1 ALU Simulation

This test focused on the functionality of the ALU in the design, ensuring it produces the correct mathematical results and writes them back correctly to the registers and memories.

The same instructions from the original design git repository were run ( check references – first link).

The results can be seen in the attached photo, showing the result of the simulation with the current design:




We can see that each reg gets its correct data result (via the debug output & reg num).

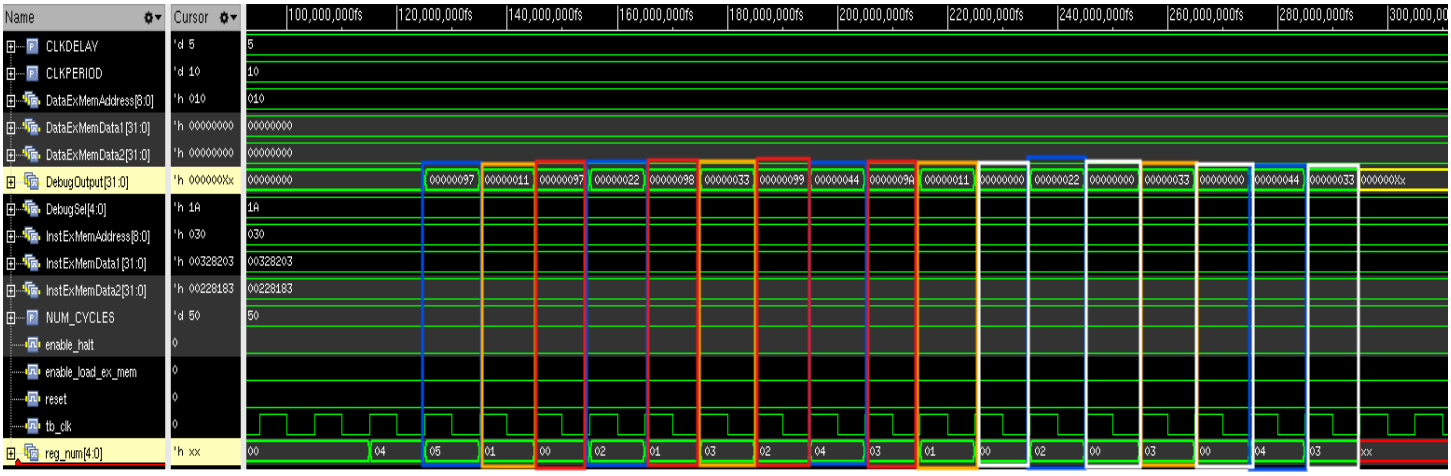
## 3.2 Write & read from random address with all cases

In this test, to ensure our memory changes can now work with all the memory space (in opposite to the original design, in which you could ensure the right result only when using an address dividable by 4).

These instructions were run on the new design:

```
# --- Setup Memory Base Address ---
addi x5, x0, 151    # x5 = 0x97 (Hexadecimal for 151. This is our base memory address)

# --- Store Bytes into Memory ---
addi x1, x0, 17     # x1 = 0x11 (Hexadecimal for 17. Value to store)
sb x1, 0(x5)        # Store 0x11 from x1 at Memory[0x97] (x5 + 0)

addi x2, x0, 34     # x2 = 0x22 (Hexadecimal for 34)
sb x2, 1(x5)        # Store 0x22 from x2 at Memory[0x98] (x5 + 1)

addi x3, x0, 51     # x3 = 0x33 (Hexadecimal for 51)
sb x3, 2(x5)        # Store 0x33 from x3 at Memory[0x99] (x5 + 2)

addi x4, x0, 68     # x4 = 0x44 (Hexadecimal for 68)
sb x4, 3(x5)        # Store 0x44 from x4 at Memory[0x9A] (x5 + 3)

# --- Load Bytes from Memory ---
lb x1, 0(x5)        # Load byte from Memory[0x97] into x1 (x1 becomes 0x11)
lb x2, 1(x5)        # Load byte from Memory[0x98] into x2 (x2 becomes 0x22)
lb x3, 2(x5)        # Load byte from Memory[0x99] into x3 (x3 becomes 0x33)
lb x4, 3(x5)        # Load byte from Memory[0x9A] into x4 (x4 becomes 0x44)

# --- Final State ---
# Registers: x1=0x11, x2=0x22, x3=0x33, x4=0x44, x5=0x97
# Memory: Mem[0x97]=0x11, Mem[0x98]=0x22, Mem[0x99]=0x33, Mem[0x9A]=0x44
```

And it can be shown that we got the correct result via the attached simulation.



# 4. Synthesis and Layout

## 4.1 Used Technology

To pressed with simplified backend flow for the design, the following tools and technologies were used:

**Synthesis:**

- **Tool:** Synopsys Design Vision
- **Technology:** TSMC TSL 108 Tower Design Kit
  This setup was used to convert the RTL design into a gate-level netlist that matches the target manufacturing process.

**Layout:**

- **Tool:** Cadence Innovus.
- **Technology:** TSMC TSL 108 Tower Design Kit
  This was used to create the physical layout of the chip, including placement, routing, and checking for design rule violations.

Using these tools ensured that the design could be correctly prepared for fabrication using the TSMC TSL 108 process.

## 4.2 Synthesis Flow

The **synthesis process** was performed using **Synopsys Design Vision** along with the **TSMC TSL 108 Tower Design Kit**.

- Firstly, the design files were setup, by both reading the relevant system Verilog files, analysing & elaborating parametrized designs, and creating the Hierarchy.
- Secondly, we define constraints: by creating a primary clock, maximum delays between signals were specified using commands like `set_max_delay` and `set_input_delay`.
- The design was synthesized using the compile function in Design Vision. This step mapped the RTL code to gate-level components supported by the technology library (TSL 108), optimizing for area, speed, and power.
- After synthesis, the design was checked for warnings and errors using Design → Check Design. Reports on area, timing, and power were generated to verify synthesis quality.

## 4.3 Layout Flow

The **Layout process** was performed using **Cadence Innovus 20.1** along with the **TSMC TSL 018 (Tower 0.18µm) Design Kit.**

**1) Preparation of the Synthesized File:**
The synthesized Verilog file was updated using the gentop.pl script, which adds pad definitions and creates two files: top.v (updated design) and top.io (pad locations), replacing the general technology gates with specific gates.

**2) Floorplan Definition:**
The layout environment was initialized in Innovus. The floorplan was defined, including die size, core area, and spacing between the core and IO pads.

**3) Power Planning:**
Power and ground nets (VDD/VSS) were connected using rings and stripes. These ensure stable power distribution across the chip.

**4) Cell Placement:**
Standard cells were placed automatically within the defined floorplan, including Tower SRAMs considering timing and congestion. Placement was optimized to minimize delay and area.

**5) Clock Tree Synthesis (CTS):**
A balanced clock network was created to distribute the clock signal across the chip, ensuring minimal skew (1.3 ns in our case) and proper synchronization.

**6) Routing Preparation and Execution:**
After placement and CTS, power routing was finalized. Then, full signal routing (global and detailed) was performed using the NanoRoute engine to connect all nets according to design rules while keeping the frequency & clock tree we defined.

**7) STA (Static Timing Analysis) check:**

Using the **PrimeTime** tool, we checked for timing violations in the design.

The tool analyses all possible paths in the gate-level netlist under **worst-case conditions & assumption**.

By using slightly modified scripts, PrimeTime verifies **setup and hold constraints**, **path delays**, and **critical corner cases**.

This process enables us to **debug timing issues** and adjust the **operating frequency** to find the **maximum achievable frequency** that still satisfies all timing requirements.

# 5. Summary and Conclusions

## 5.1 Results

In this project, we optimized the original RISC-V design by replacing the FPGA-based memories with Tower's efficient SRAM blocks, which improved area and power consumption. Additionally, we extended support for the full memory address space, enabling proper handling of all load and store operations.

- A Clock Tree with 1.3ns slack (Figure 6)
- Critical path length: Several clock periods were tested to determine the maximum operating frequency without setup or hold violations.
    - At a **4 ns clock period**, the critical path length was **4.34 ns** (or **6.05 ns** when including clock uncertainty/slack), which resulted in timing violations.
    - At a **6 ns clock period**, the critical path length was **5.93 ns**, leaving a **slack of 0.07 ns (on setup)**, which meets the timing requirements.
    - Based on this analysis, a **6 ns clock period** (equivalent to **166.67 MHz**) was selected as the optimal operating frequency.
- Design area: refer section 8 (appendices).

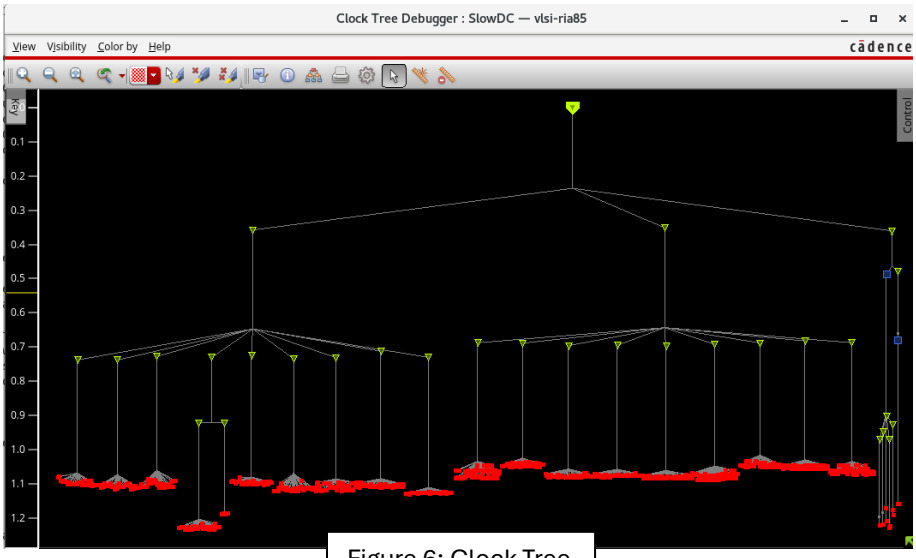



Figure 6: Clock Tree

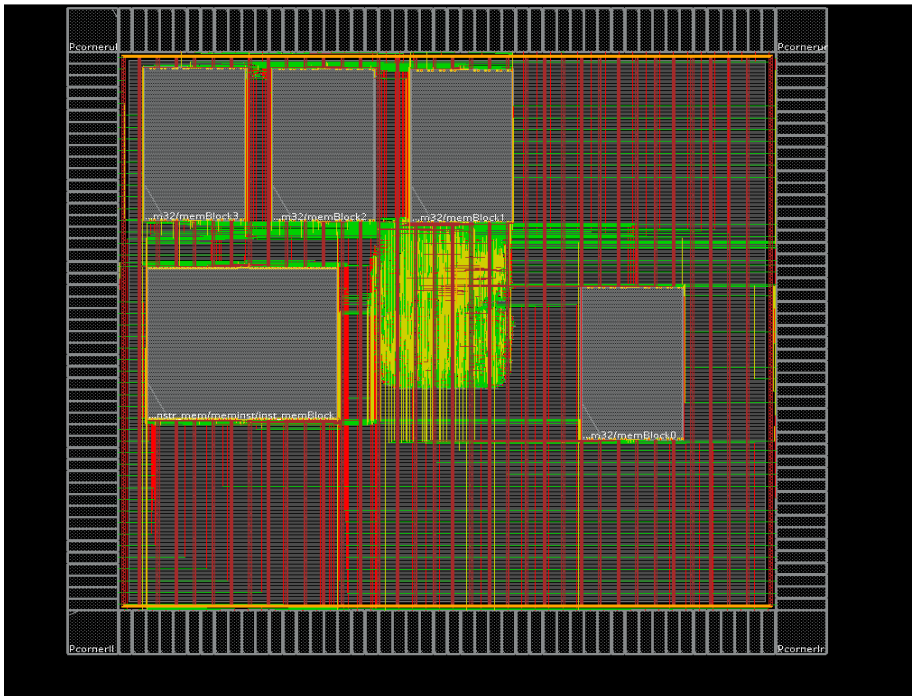



Figure 7: Post CTS layout

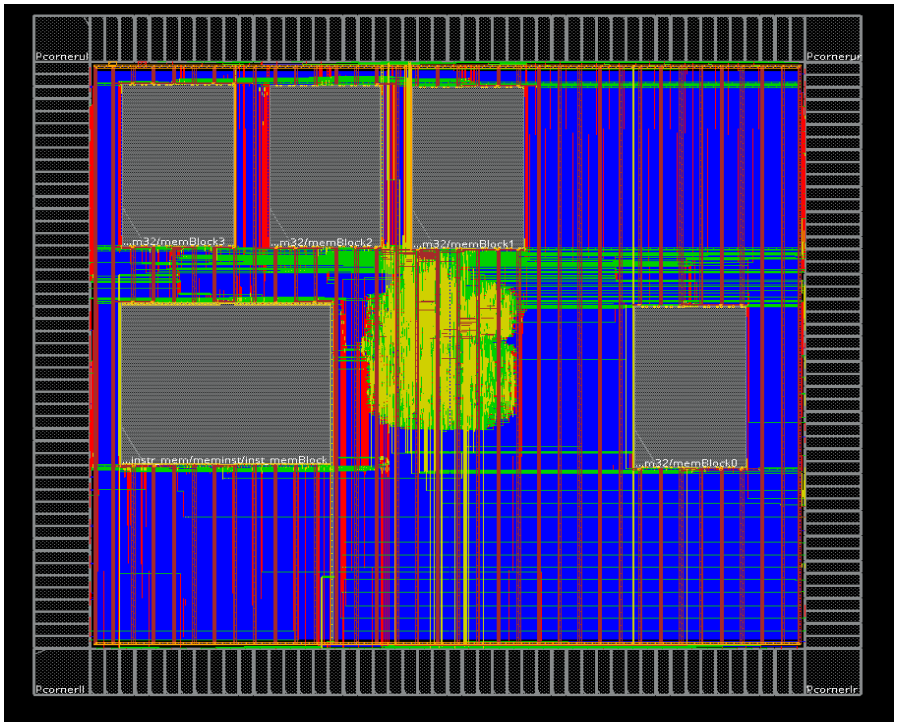



Figure 8: Post Route layout

## 5.2 summary

In this project, we implemented code changes and Tower SRAM memory integration to a RISC-V-based processor with added support for post-fabrication testing, debugging, and memory loading.

We integrated Tower SRAMs, adding support for all memory space in memory operations, and using area & power effecting memories

We developed a custom debug unit, allowing external users to monitor internal signals and halt execution when needed. A memory loading mechanism was also integrated to simplify testing and benchmarking.

To test it, we modified a script to allow creating complex test benchmark for simulations and performed mutable tests.

For backend flow, we performed synthesis using Synopsys Design Vision and the TSL 108 Tower Design Kit and used Cadence Innovus for physical layout.

## 5.3 Problems, Challenges & Solution

There were several challenges we faced; most notable is the size of our design and toolchain incompatibility between VLSI Backend Manuals.

### 5.3.1 Design Size

At first added **Debug unit** was meant to expose to the user all the 29 critical points we determined as important, this led to many inputs and outputs, this increasing the size of our design exponentially.

This meant our design was huge and contained multiple large wires that we passed thro the debug unit from the RISCV logic, creating a high-density & complex design, which was also power consuming.

The large size and complex big wires also led to a high clock skew, resulting in low work frequency for the RISC-V.

To counterpart this problem, we used a technique that is largely known in chip DFT industry, the Debug unit outputs were all connected to a 32-way mux, thus reducing the 29 outputs with ~600 bits to a 32bit output wire, thus reducing design size ( by about ~%60) and complexity ( skew decreased by ~%52, from  2.7 ns to 1.2).

### 5.3.2 Toolchain Incompatibility Between VLSI Backend Manuals

During the backend flow, we encountered a significant challenge stemming from the coexistence of two backend manuals: a comprehensive (long) manual and a condensed (short) manual – which we used, both designed for physical implementation using Tower 180nm technology.

While both manuals aim to generate a valid layout for fabrication, they use different standard cell libraries and tool setups.

As a result, their scripts, environments, and verification flows are incompatible, making it difficult to combine methods or reuse tools across manuals.

This limitation was shown during our attempt to use PrimeTime (as guided in the long manual) within a project originally structured using the short manual.

Due to differences in library versions and environment variables, we were unable to run the tool directly. Instead, we had to construct a new environment manually, identifying and importing only the minimal required set of libraries and configuration files necessary to enable timing verification. This process required extensive troubleshooting and consultation with lab staff, delaying our progress.

# 6. Future Work

During the semester (winter 2024/25), the same design was used in another project, which included creating a branching unit.

We would recommend merging the projects in the future, thus creating a powerful RISC-V with a branching unit, Full memory space support in load/store operations, an efficient SRAMS and a debug unit.

We would also like to integrate formal verification tools, especially jasper (which was added to the lab in 2024/25) to the project, since simulations are not powerful enough, especially in projects with small code & logic changes that have large impact.

Lastly, a full back-end layout prosses would increase even our future design and the maximum working frequency of the design.

# 7. References


- Original GitHub project repository - pipelined RISC-V: https://github.com/estufa-cin-ufpe/RISC-V-Pipeline/tree/master .

- FPJA virtual memory used to simulate the original project - altsyncram. v file: https://github.com/dlitz/openmsp430/blob/master/fpga/altera_de1_board/bench/verilog/altsyncram.v .

- **Synopsys (2022/23) VHDL/Verilog Sim and Synthesis Tower 0.18u** - VLSI Lab, Technion.

- **Cadence NCSIM System Verilog VHDL Manual (2023/24)** Manual, VLSI Lab Technion.

- SRAM memories Manual, VLSI Lab Technion - **Hardware Accelerator for Machine Learning in System Verilog (98)**.

- **Innovus20 with Tower_0.18u** Manual, VLSI Lab - Technion.

# 8. Appendices

## 8.1 Area report

| Hinst Name | Module Name | Inst Count | Total Area |
|---|---|---|---|
| top | | 9121 | 6170781.233 |
| I0 | riscv | 8926 | 2817031.233 |
| I0/ac | ALUController | 31 | 435.904 |
| I0/c | Controller | 26 | 354.368 |
| I0/dp | Datapath | 8278 | 2802640.129 |
| I0/dp/Ext_Imm | imm_Gen | 63 | 1078.784 |
| I0/dp/FAmux | mux4_0 | 106 | 1721.664 |
| I0/dp/FBmux | mux4_2 | 105 | 1727.936 |
| I0/dp/alu_module | alu | 1501 | 26587.008 |
| I0/dp/alu_module/add_45 | alu_DW01_add_1 | 197 | 3230.080 |
| I0/dp/alu_module/r384 | alu_DW01_cmp6_1 | 131 | 2420.992 |
| I0/dp/alu_module/sub_53 | alu_DW01_sub_1 | 200 | 3261.440 |
| I0/dp/brunit | BranchUnit | 135 | 2496.256 |
| I0/dp/brunit/add_41 | BranchUnit_DW01_add_2 | 101 | 1894.144 |
| I0/dp/data_mem | datamemory | 653 | 1772271.056 |
| I0/dp/data_mem/mem32 | Memoria32Data | 557 | 1769840.656 |
| I0/dp/data_mem/mem32/add_51_2 | Memoria32Data_DW01_inc_0 | 12 | 395.136 |
| I0/dp/data_mem/mem32/r419 | Memoria32Data_DW01_inc_1 | 11 | 360.640 |
| I0/dp/detect | HazardDetection | 15 | 398.272 |
| I0/dp/forunit | ForwardingUnit | 46 | 969.024 |
| I0/dp/instr_mem | instructionmemory | 98 | 821589.873 |
| I0/dp/instr_mem/meminst | Memoria32 | 15 | 820228.849 |
| I0/dp/pcadd | adder | 8 | 470.400 |
| I0/dp/pcadd/add_8 | adder_DW01_add_0_DW01_add_9 | 8 | 470.400 |
| I0/dp/pcmux | mux2_WIDTH9 | 9 | 225.792 |
| I0/dp/pcreg | flopr | 19 | 831.040 |
| I0/dp/resmux | mux2_WIDTH32_1 | 32 | 802.816 |
| I0/dp/rf | RegFile | 3697 | 120472.576 |
| I0/dp/srcbmux | mux2_WIDTH32_0 | 33 | 840.448 |
| I0/dp/wrsmux | mux4_1 | 100 | 1680.896 |
| I0/du | DebugUnit | 586 | 13381.312 |
| I0/du/DebugMux | mux32 | 419 | 7197.120 |

## 8.2 Power report

```
Total Power
-----------------------------------------------------------------------------------------
Total Internal Power:        38.89284587         88.4050%
Total Switching Power:        4.74633040         10.7886%
Total Leakage Power:          0.35478280          0.8064%
Total Power:                 43.99395908
-----------------------------------------------------------------------------------------


Group                              Internal    Switching      Leakage        Total  Percentage
                                   Power       Power          Power          Power  (%)
-----------------------------------------------------------------------------------------
Sequential                            5.779       0.4538     0.005797        6.239       14.18
Macro                                22.99        0.1838        0.27        23.44       53.29
IO                                    7.995       0.4939     0.07256         8.562       19.46
Combinational                         1.899       2.563      0.006268        4.469       10.16
Clock (Combinational)                 0.2305      1.051      0.0001698       1.282        2.914
Clock (Sequential)                          0           0              0           0           0
-----------------------------------------------------------------------------------------
Total                                38.89        4.746        0.3548       43.99         100
-----------------------------------------------------------------------------------------


Rail                    Voltage    Internal    Switching      Leakage        Total  Percentage
                                   Power       Power          Power          Power  (%)
-----------------------------------------------------------------------------------------
VDD                        1.62       38.89        4.746        0.3548       43.99         100


Clock                              Internal    Switching      Leakage        Total  Percentage
                                   Power       Power          Power          Power  (%)
-----------------------------------------------------------------------------------------
clk                                   0.6541      1.054      0.000559        1.708        3.883
-----------------------------------------------------------------------------------------
Total                                 0.6541      1.054      0.000559        1.708        3.883
-----------------------------------------------------------------------------------------
Clock: clk
Clock Period: 0.022000 usec
Clock Toggle Rate:    90.9091 Mhz
Clock Static Probability:  0.5000


-----------------------------------------------------------------------------------------
*    Power Distribution Summary:
*               Highest Average Power: I0/dp/instr_mem/meminst/inst_memBlock (dpram2048x32_CB):            8.412
*               Highest Leakage Power: I0/dp/instr_mem/meminst/inst_memBlock (dpram2048x32_CB):          0.05525
*                 Total Cap:      2.93828e-10 F
*                 Total instances in design:  9121
*                 Total instances in design with no power:     8
*                 Total instances in design with no activty:     8

*                 Total Fillers and Decap:     4
-----------------------------------------------------------------------------------------
```

## 8.3 Timing report

```
Path 1: MET Setup Check with Pin I0/dp/data_mem/mem32/memBlock3/CEB1
Endpoint:   I0/dp/data_mem/mem32/memBlock3/I1[1] (^) checked with trailing edge
of 'clk'
Beginpoint: I0/dp/C_reg_Alu_Result__0_/Q         (v) triggered by  leading edge
of 'clk'
Path Groups: {clk}
Analysis View: SlowView
Other End Arrival Time          11.143
- Setup                          0.027
+ Phase Shift                    0.000
= Required Time                 11.116
- Arrival Time                   5.806
= Slack Time                     5.310
     Clock Rise Edge                      0.000
     + Clock Network Latency (Prop)       0.020
     = Beginpoint Arrival Time            0.020
     +-----------------------------------------------------------------------------------------------+
     |              Instance                |     Arc      |    Cell     | Delay | Arrival | Required |
     |                                      |              |             |       |  Time   |   Time   |
     |--------------------------------------+--------------+-------------+-------+---------+----------|
     | I0/dp/C_reg_Alu_Result__0_           | CP ^         |             |       |   0.020 |    5.330 |
     | I0/dp/C_reg_Alu_Result__0_           | CP ^ -> Q v  | dfnrq1      | 0.699 |   0.719 |    6.029 |
     | I0/dp/U1151                          | I0 v -> Z v  | mx02d1      | 0.674 |   1.393 |    6.703 |
     | I0/dp/data_mem/mem32/U23             | I v -> ZN ^  | inv0d0      | 0.411 |   1.804 |    7.114 |
     | I0/dp/data_mem/mem32/U30             | A2 ^ -> ZN v | nd02d1      | 1.042 |   2.847 |    8.157 |
     | I0/dp/data_mem/mem32/U7              | A1 v -> ZN v | nd12d1      | 0.493 |   3.339 |    8.649 |
     | I0/dp/data_mem/mem32/FE_OFC50_n156   | I v -> Z v   | buffd3      | 0.597 |   3.936 |    9.246 |
     | I0/dp/data_mem/mem32/U148            | C1 v -> ZN ^ | oai221d1    | 1.078 |   5.014 |   10.324 |
     | I0/dp/data_mem/mem32/FE_OFC52_inS3_1 | I ^ -> Z ^   | buffd1      | 0.768 |   5.782 |   11.092 |
     | I0/dp/data_mem/mem32/memBlock3       | I1[1] ^      | dpram4096x8 | 0.024 |   5.806 |   11.116 |
     +-----------------------------------------------------------------------------------------------+
```