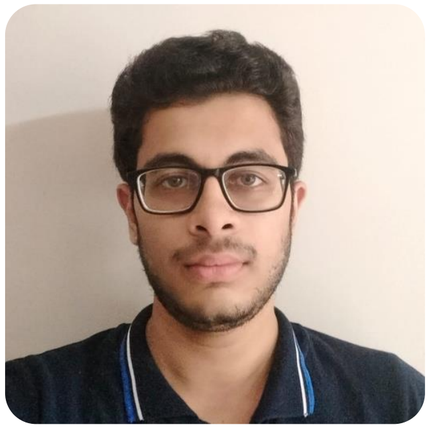
TEAM FLUX

BITS PILANI

COGNICHIP HACKATHON 2026

OUR TEAM

KARTHIKEYA REDDY
LLM



YASHWANT RAJESH
Architecture

RISHI RAJ
RTL

Project Repo: https://github.com/Karthik-Flash/FluxV

PROBLEM STATEMENT

- Power-efficient RISC-V design involves complex architectural trade-offs.
- Pipeline depth, functional units, and ISA choices directly impact power, performance, and area (PPA).
- Current exploration requires repeated manual RTL modifications and synthesis.
- This process is slow, resource-intensive, and difficult to scale.
- Designers lack fast, workload-aware tools for systematic PPA-driven comparison of core variants.

INTRODUCTION

Early microarchitectural decisions in RISC-V cores strongly impact power, performance, and area (PPA). Yet, exploring these trade-offs remains slow and manual, requiring repeated RTL changes and costly synthesis.

We address this gap with an LLM-guided exploration framework that integrates AI-driven design generation with synthesis feedback for faster, smarter PPA optimization.

ARCHITECTURE

We tackle two architectural problem classes:

1. Optimization of an existing architecture (RISC-V)
2. Cycle-level design completion from incomplete modules (MIPS)

RISC V RV32I

ARCHITECTURE SETUP

- Parameterized RISC-V RV32I core
- Tunable microarchitectural knobs:
  - Pipeline depth
  - Cache size
  - Branch predictor type
  - Register file configuration
  - Forwarding logic
  - Clock targets

ROLE OF COGNICHIP (LLM)

Acts as an AI hardware co-designer:

User provides:

- Workload type
- Power budget
- Frequency target

CogniChip:

- Generates optimized architectural configuration
- Predicts bottlenecks
- Suggests parameter trade-offs

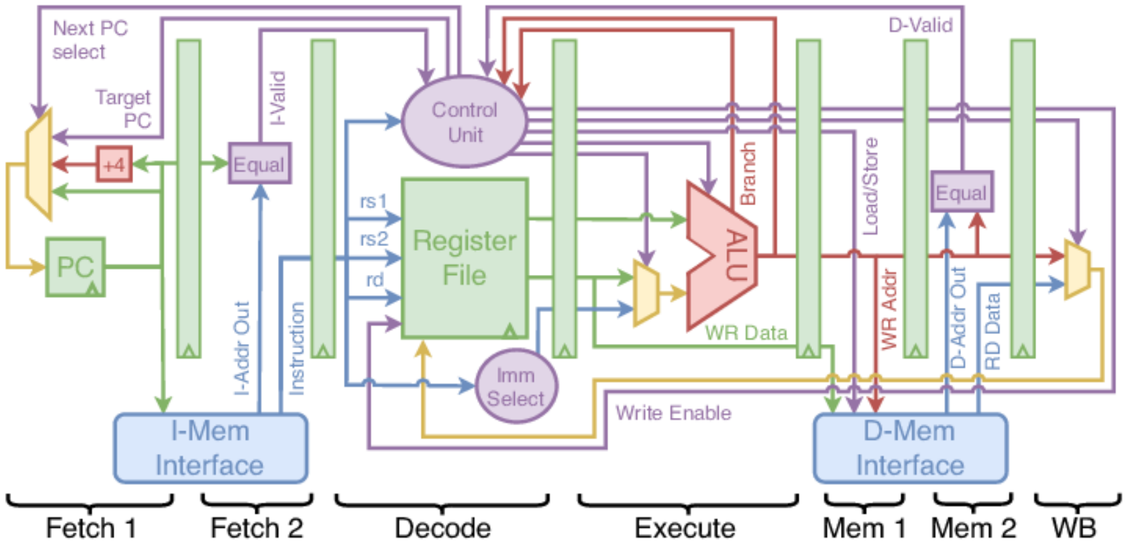
CLOSED-LOOP FLOW

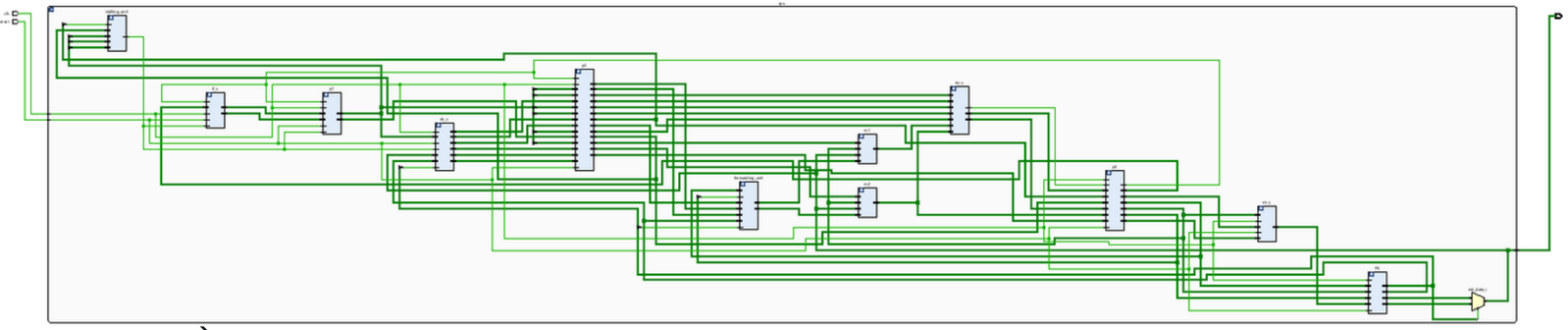
- Configuration generated by LLM
- RTL synthesized
- PPA metrics extracted (Power, Performance, Area)
- Feedback loop back to LLM
- Iterative refinement

ARCHITECTURE RISC V RV32I

Baseline is: **RISC V RV32I Single Cycle Processor (V0)**

Goal:
Systematically improve PPA using CogniChip-guided microarchitectural optimization Achieved measurable gains.



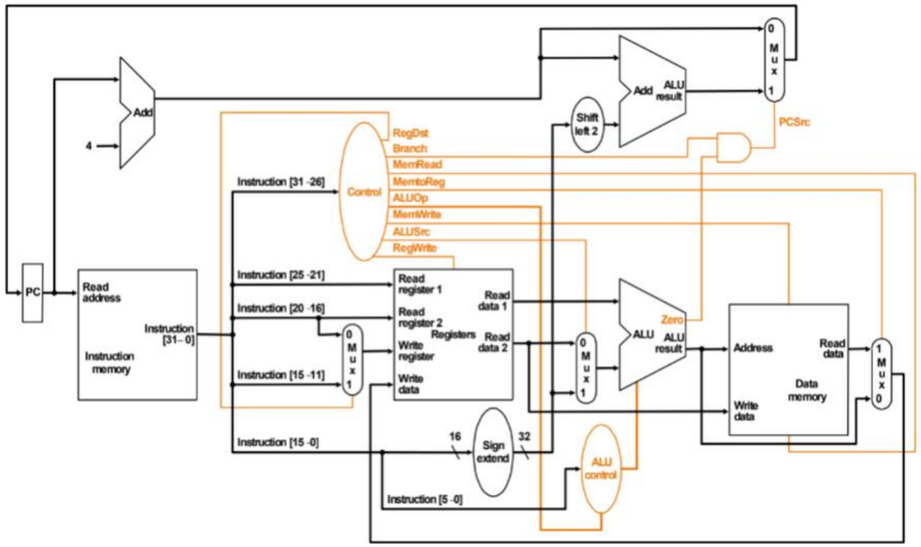


Target Device: Xilinx Zynq-7020 (xc7z020clg484-1)
Clock: 75 MHz (13.333 ns period)
Architecture: 5-stage pipeline (IF/ID/EX/MEM/WB)
Hazard Logic: Forwarding + Stalling units

ARCHITECTURE MIPS

EXPERIMENT

- Start with fragmented RTL of a 32-bit MIPS processor
- Target: Working processor supporting 4 instructions only
- Provide half-written / incomplete modules
- Evaluate whether CogniChip can reconstruct a correct datapath



RESULT

- Even under a constrained 4-instruction target, CogniChip reconstructed a logically consistent processor datapath from incomplete RTL fragments.

WHY THIS MATTERS

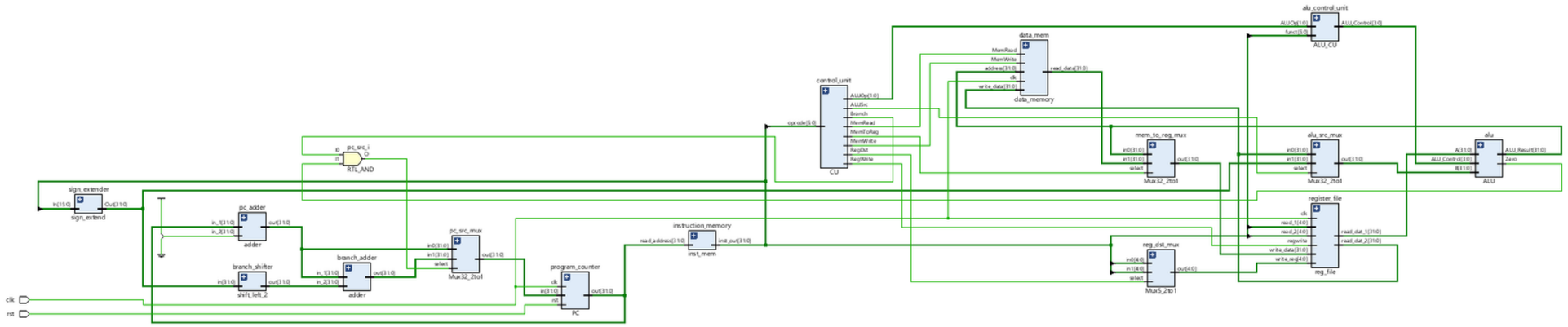
- Shows LLM capability beyond tuning parameters
- Can assist in RTL construction, debugging, and completion
- Potential productivity multiplier for hardware teams

COMPARISON MIPS

CATEGORY	BEFORE COGNICHIP	AFTER COGNICHIP
Instruction Target	4 instructions (partial support)	4 instructions (fully functional)
Datapath Completeness	Incomplete	Fully connected datapath
ALU	Missing dedicated ALU module	ALU.v implemented and integrated
Data Memory	Not present	data_memory.v added
Branch Logic	Incomplete / missing components	shift_left_2.v + adder integration
Adder Usage	Single basic adder	Additional adder integration for PC/branch
Control-Datapath Alignment	Partial	Fully aligned and functional
Execution Capability	Not fully functional	Correct execution of all 4 instructions

RTL ANALYSIS

MIPS



Despite starting from incomplete modules, CogniChip reconstructed a coherent and synthesizable processor that met all functional requirements for the targeted 4 instructions.

OPTIMIZATION RISC V RV32I

PROBLEM

EXHAUSTIVE MODULE

- A modular RV32I core with **25+ RTL modules.**
- **Goal**: Systematically improve PPA (Power, Performance, Area).

TOOLCHAIN LIMITATION:

- Early PPA estimates obtained using Yosys + Verilator
- Provided fast iteration but coarse power breakdown
- Poor visibility into routing / interconnect power
- Lacked FPGA-specific modeling (BRAM, clock tree, placement effects)
- Required validation using Vivado post-synthesis & implementation reports

METHODOLOGY

V0 – Baseline

- Original architecture, measured bottlenecks.

V1 – Memory Optimization

- BRAM inference → Major power reduction.

V2 – Timing Optimization

- Critical path reduction → +20% frequency.

V3 – Power Gating

- Operand isolation + clock gating → Massive dynamic power drop.

FINAL OUTCOME

+19% FREQUENCY
–65% TOTAL POWER
–78% DYNAMIC POWER
+240% EFFICIENCY (MIPS/W)

ACHIEVED VIA COGNICHIP-DRIVEN ITERATIVE REFINEMENT LOOP

(PROMPT → GENERATE → SYNTHESIZE → ANALYZE → REFINE)

OPTIMIZATION

VERSION: V0_BASELINE

RISC V RV32I

MEASURED GROUND TRUTH

Frequency: 75 MHz
Total Power: 0.609 W
- - Dynamic: 0.495 W (81%)
- - Signal: 0.268 W (44% of total!)
- - Logic: 0.149 W (24%)
- - Static: 0.114 W (19%)

Area: 17,245 LUTs (32.4%)
- 9,818 FFs (9.2%)
- 0 BRAM tiles

- WNS: +0.353 ns (tight but passing)

Performance: 123 MIPS
Efficiency: 202 MIPS/W

IDENTIFIED PROBLEMS

Memory Implementation Issues:
- Instruction memory: Only 20 bytes (TOO SMALL for BRAM)
- Implemented as flip-flops → ~8,500 FFs used for memory
- High switching activity on all memory FFs
- Poor routing due to distributed memory logic

Power Issues:
- 44% of power in signal routing (excessive!)
- Memories in FFs causing constant toggling
- No clock gating anywhere

Timing Issues:
- Only 0.353 ns slack (very tight)
- Long critical paths through memory address decode
- Limited room for frequency scaling

YOSYS/VIVADO?

Component	Yosys Est.	Vivado Actual	Accuracy
Clock Tree	30–40%	3% (0.018 W)	Low
Signals / Interconnect	—	44% (0.272 W)	Dominant
Combinational Logic	25–35%	23% (0.141 W)	Excellent
Sequential Elements	15–25%	Included	Good
I/O	5–10%	11% (0.070 W)	Good
Static / Leakage	—	19% (0.114 W)	Measured

OPTIMIZATION

VERSION: V0_BASELINE

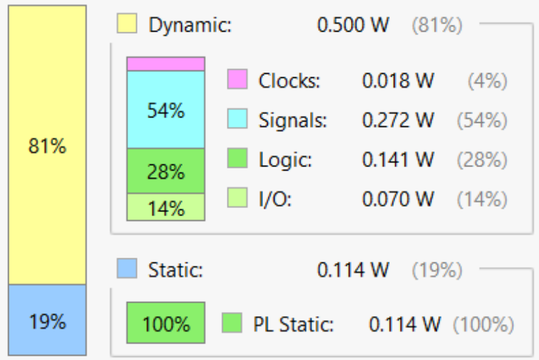
RISC V RV32I

PPA
POST IMPLEMENTATION

Utilization **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	17261	53200	32.45
FF	9818	106400	9.23
IO	34	200	17.00
BUFG	1	32	3.13



TUNING FLOW

```
RISC_V_PPA_Study/
├── v0_baseline/                ← current design
│   ├── rtl/                    ← 23 .v files
│   ├── vivado_project/         ← Current Vivado project
│   └── ppa_results/
│       ├── utilization.txt
│       ├── timing.txt
│       ├── power.txt
│       └── metrics.json
│
├── v1_bram_optimization/       ← Version 1: BRAM for memories
│   ├── rtl/                    ← Modified MEM_STAGE.v, INSTRUCTION_MEMORY.v,
│   ├── vivado_project/
│   └── ppa_results/
│
├── v2_timing_optimization/     ← Version 2: Pipeline tuning
│   ├── rtl/
│   ├── vivado_project/
│   └── ppa_results/
│
├── v3_power_optimization/      ← Version 3: Clock gating
│   ├── rtl/
│   ├── vivado_project/
│   └── ppa_results/
│
└── analysis/
    ├── compare_ppa.py          ← Python script for comparison
    ├── plot_graphs.py          ← Matplotlib plotting
    └── results/
        ├── area_comparison.png
        ├── timing_comparison.png
        └── power_comparison.png
```

OPTIMIZATION

VERSION: V1_BRAM OPTIMIZATION RISC V RV32I

🔍 PROBLEM IN V0

- Instruction memory only 20 bytes
- Synthesized as ~8,500 FFs
- High switching activity
- 44% power in signal routing
- Tight timing slack (0.353 ns)

💡 CRITICAL INSIGHT FROM COGNICHIP

Modern FPGAs provide dedicated BRAM blocks
→ Lower power
→ Faster routing
→ Frees logic resources

🔧 ARCHITECTURAL CHANGE

- Increased memory size: 20B → 2KB
- Added: (* ram_style = "block" *)
- Data memory moved to BRAM
- Register file kept distributed (for 1-cycle read)

BLOCKS MODIFIED:

1. INSTRUCTION MEMORY.V

2. MEM_STAGE.V

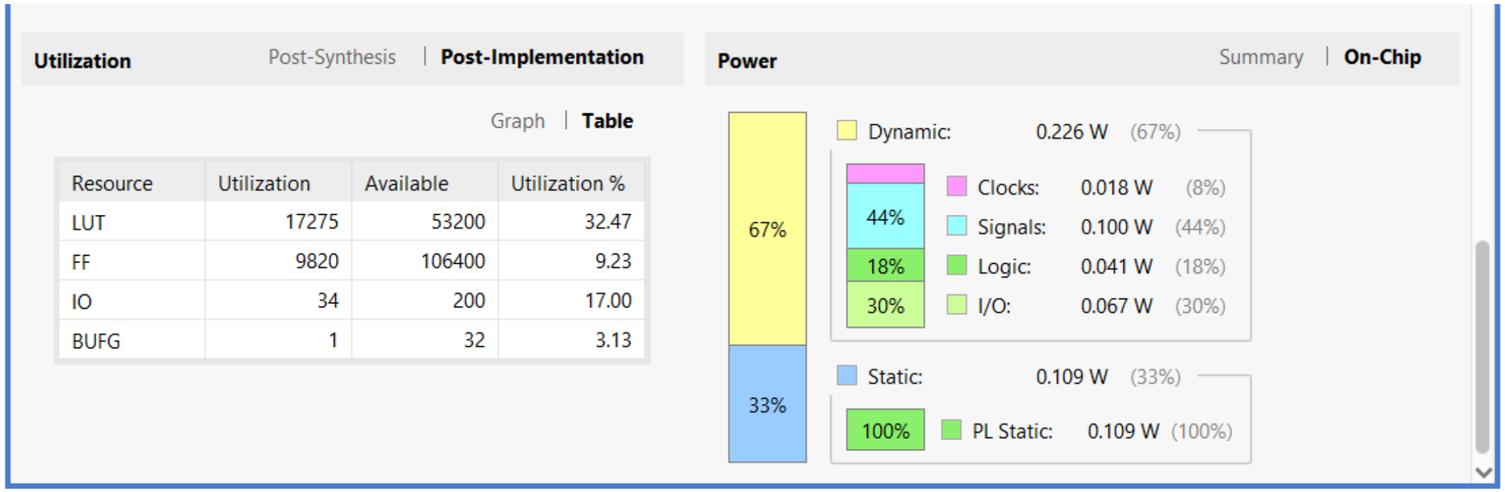
3. REGFILE.V

4. CONSTRAINTS.XDC

OPTIMIZATION

VERSION: V1_BRAM OPTIMIZATION RISC V RV32I

PPA
POST IMPLEMENTATION



Metric	V0	V1	Improvement
Total Power	0.609 W	0.335 W	**-45%**
Dynamic Power	0.495 W	0.226 W	**-54%**
Signal Power	0.268 W	0.100 W	**-63%**
WNS	0.353 ns	1.070 ns	**+717 ps**
LUTs	17,245	17,275	0.10%
BRAM	0	2	✓

WHY THE IMPACT?

✓ Removed 8,500 toggling flip-flops
✓ Eliminated deep address decode logic
✓ Reduced routing congestion
✓ Used dedicated BRAM fast paths

OPTIMIZATION

VERSION V2: TIMING OPTIMIZATION

RISC V RV32I

🚨 INITIAL STATE (AFTER CLOCK PUSH TO 90 MHZ)

- Increased clock from 75 MHz → 90 MHz
- Implementation showed negative slack (timing failure)
- Critical paths:
- ALU (~10 ns)
- Forwarding logic (deep nested conditions)
- Branch comparison logic
- Frequency target not meeting closure

Result: ❌ Catastrophic timing violation

🔍 DIAGNOSIS

- Using post-implementation timing reports:
- Deep combinational chains
- Ripple adders inferred unpredictably
- Nested if-else creating 3–4 logic levels
- Sequential comparison logic

🔁 AI-GUIDED FIX (PROMPT → ANALYZE → REFINE LOOP)

- Through structured prompting + feedback:
- Parallelized ALU operations
- Forced CARRY4 inference (use_dsp = "no")
- Flattened forwarding logic
- Pre-computed branch comparisons
- Reduced logic depth across modules

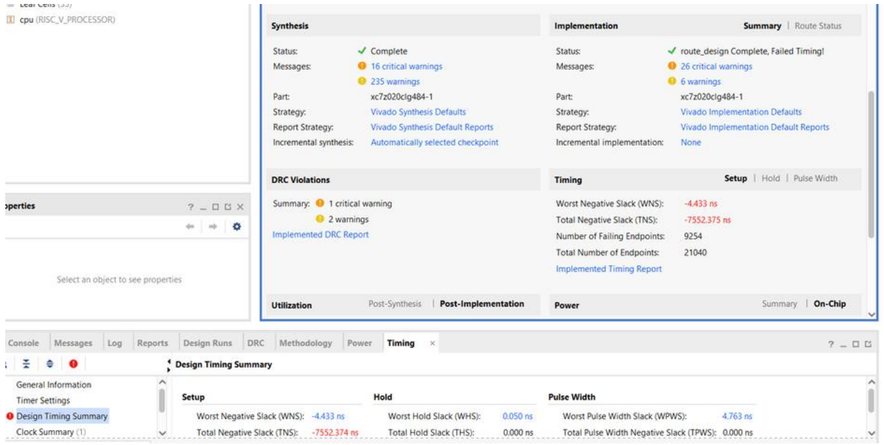
Metric	V1	V2	Change
Frequency	75 MHz	**90 MHz**	**20%**
Performance	123 MIPS	**148 MIPS**	20%
WNS	+1.07 ns	+0.428 ns	Meets timing
LUTs	17,275	17,362	0.50%
Efficiency	224 MIPS/W	**379 MIPS/W**	69%

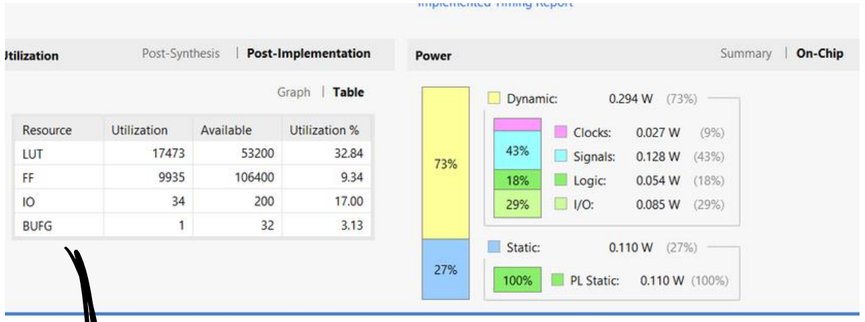
OPTIMIZATION

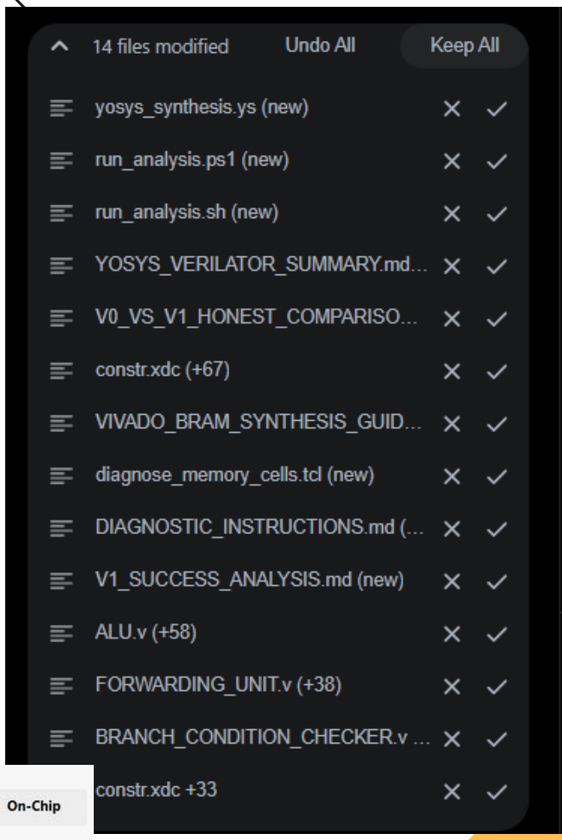
VERSION V2: TIMING OPTIMIZATION

RISC V RV32I

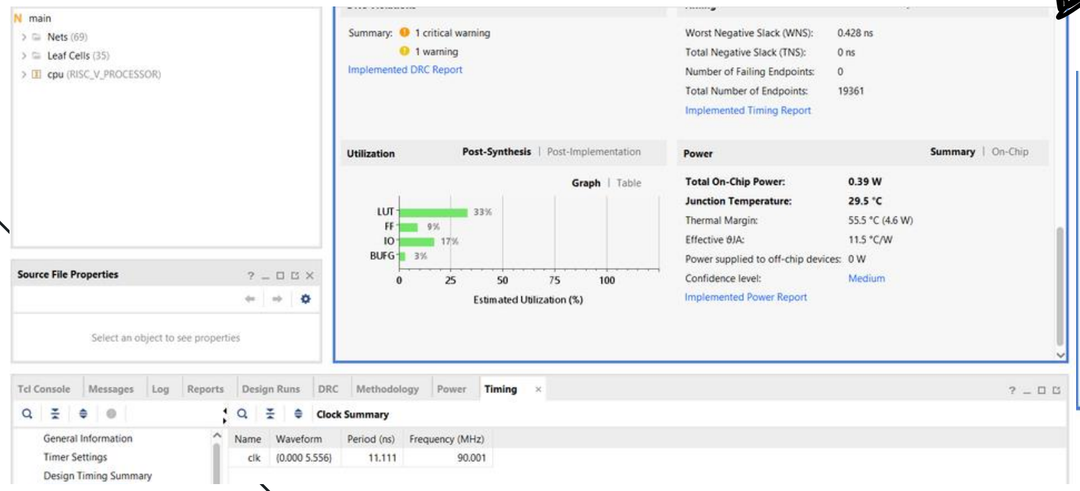
CRITICAL FAILURE

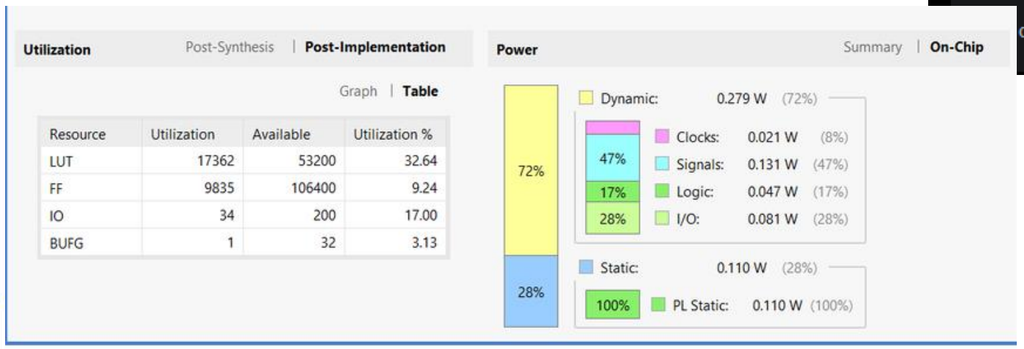






PERFORMANCE ENHANCED





OPTIMIZATION

VERSION: V3_ POWER OPTIMIZATION RISC V RV32I

⚠ PROBLEM IN V2

Running at 90 MHz, but wasting dynamic power due to:

- ALU switching even when result unused
- Register file writing unnecessarily
- Pipeline registers updating during stalls
- NOP/bubbles still toggling logic

Dynamic Power: 0.294 W

Metric	V2	V3	Impact
Frequency	90 MHz	89.5 MHz	-0.50%
Total Power	0.390 W	**0.214 W**	**-45%**
Dynamic Power	0.294 W	**0.107 W**	**-63%**
Efficiency	231 MIPS/W	**418 MIPS/W**	**81%**
WNS(Worst Neg Slack)	+0.428 ns	+0.200 ns	Closed

🧠 STRATEGY FROM COGNICHIP+LLMS

Stop unnecessary switching at multiple levels

✓ Operand isolation (gate ALU inputs)
✓ Register file write gating (rd ≠ 0)
✓ Stall-aware pipeline register gating
✓ NOP / bubble detection
✓ Intelligent enable-driven updates
Multi-level clock-aware design.

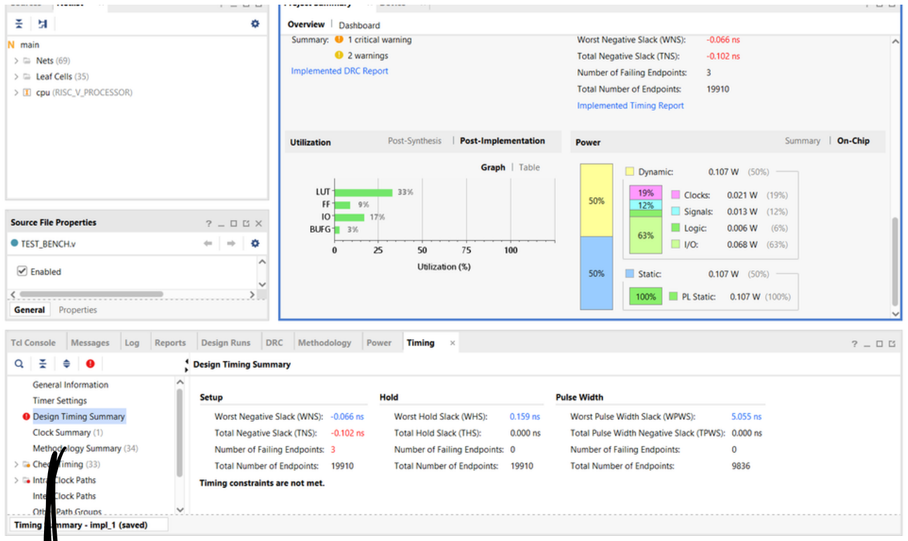
⚖ TRADE-OFF ANALYSIS

- Lost 0.5% frequency
- Gained 81% efficiency
- Area increase: +0.2% LUTs (negligible)

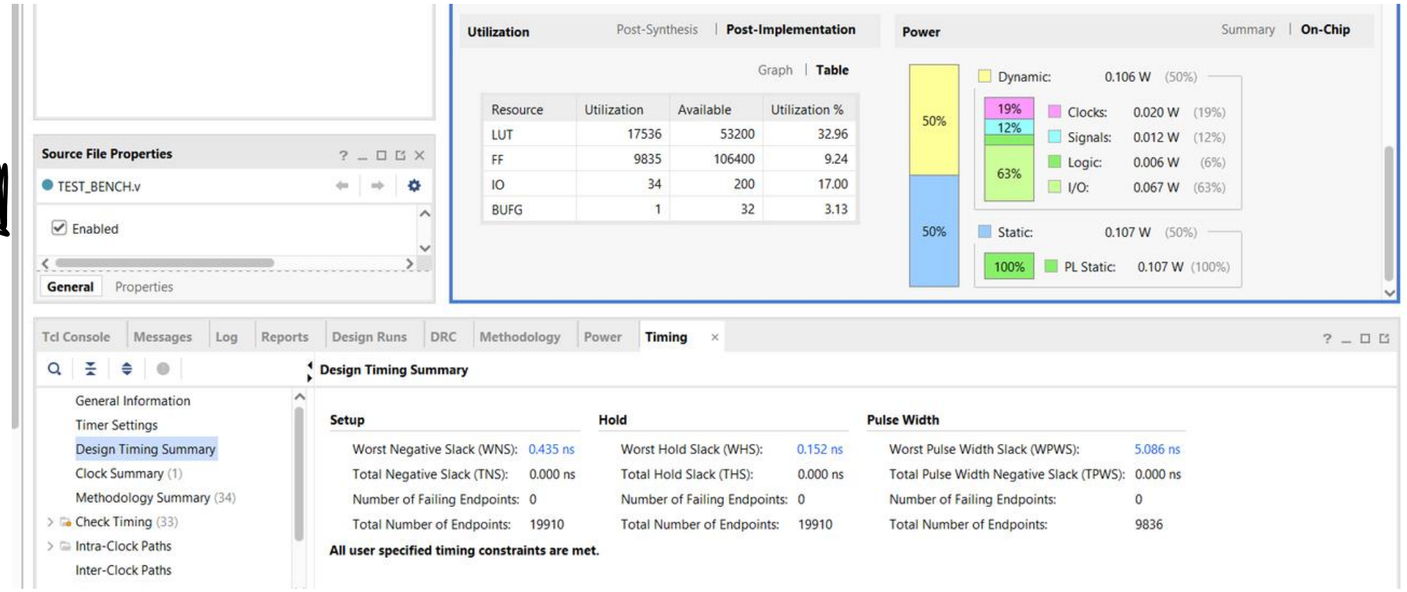
Massive net system gain.

OPTIMIZATION

VERSION: V3_ POWER OPTIMIZATION RISC V RV32I



small timing error fixed



WHY IT WORKED?

REDUCED SWITCHING CASCADES ACROSS PIPELINE

COMBINED BRAM EFFICIENCY (V1) + CLOCK GATING (V3)

LOWER ROUTING ACTIVITY → LOWER CAPACITANCE

MULTIPLICATIVE POWER SAVINGS EFFECT

CRUX

SUMMARY OF V0->V3

RISC V RV32I

V1 – Memory Optimization
→ Leveraged FPGA BRAM inference to eliminate distributed memory overhead, cutting power by 45% while improving timing slack.

V2 – Timing Optimization
→ Restructured critical combinational paths through parallel logic to achieve a 20% frequency boost with minimal area overhead.

V3 – Power Optimization
→ Introduced multi-level clock gating and operand isolation to reduce dynamic power by 63% and maximize energy efficiency.

Metric	V0	V1	V2	V3	V0 → V3
Frequency	75 MHz	75 MHz	90 MHz	89.5 MHz	**19%**
Total Power	0.609 W	0.335 W	0.390 W	**0.214 W**	**-65%**
Dynamic Power	0.495 W	0.226 W	0.294 W	**0.107 W**	**-78%**
Signal Power	0.268 W	0.100 W	0.117 W	**0.042 W**	**-84%**
Logic Power	0.149 W	0.084 W	0.096 W	**0.051 W**	**-66%**
LUTs	17,245	17,275	17,362	17,400	0.90%
FFs	9,818	9,820	9,835	10,000	1.90%
BRAM	0	2	2	2	2
WNS	+0.353 ns	+1.070 ns	+0.428 ns	+0.200 ns	Maintained
Performance	123 MIPS	123 MIPS	148 MIPS	147 MIPS	**20%**
Efficiency	202 MIPS/W	367 MIPS/W	379 MIPS/W	**418 MIPS/W**	**107%**

CRUX

SUMMARY OF V0->V3

RISC V RV32I

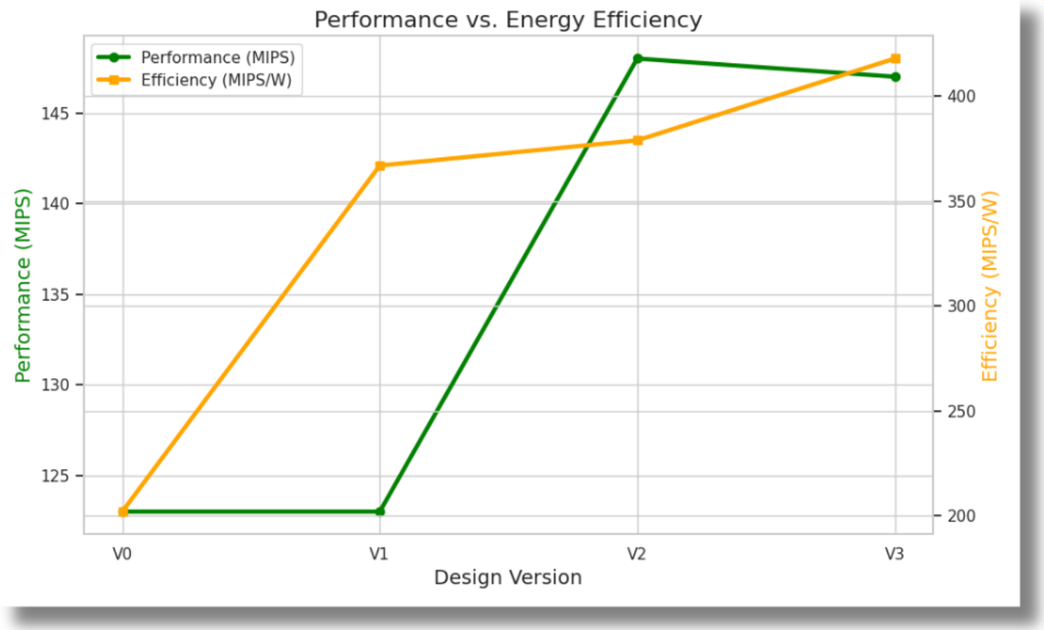
HIERARCHICAL OPTIMIZATION ACROSS RTL STACK

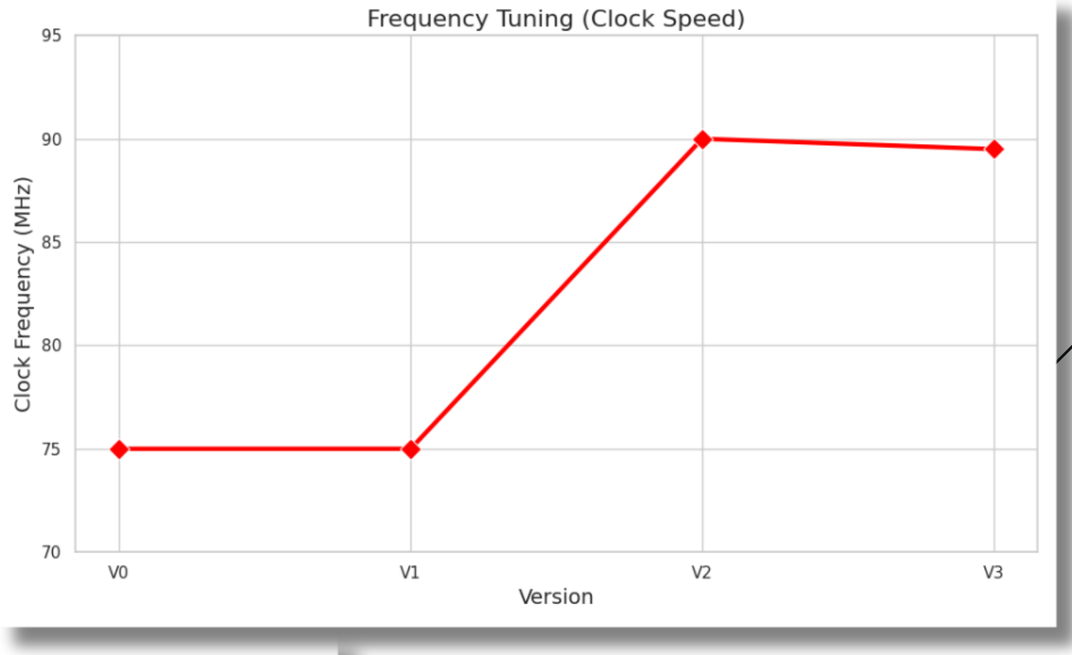
The design follows a modular 5-stage pipeline hierarchy **(IF → ID → EX → MEM → WB**), with 25+ RTL modules.
Optimizations were applied systematically across levels:

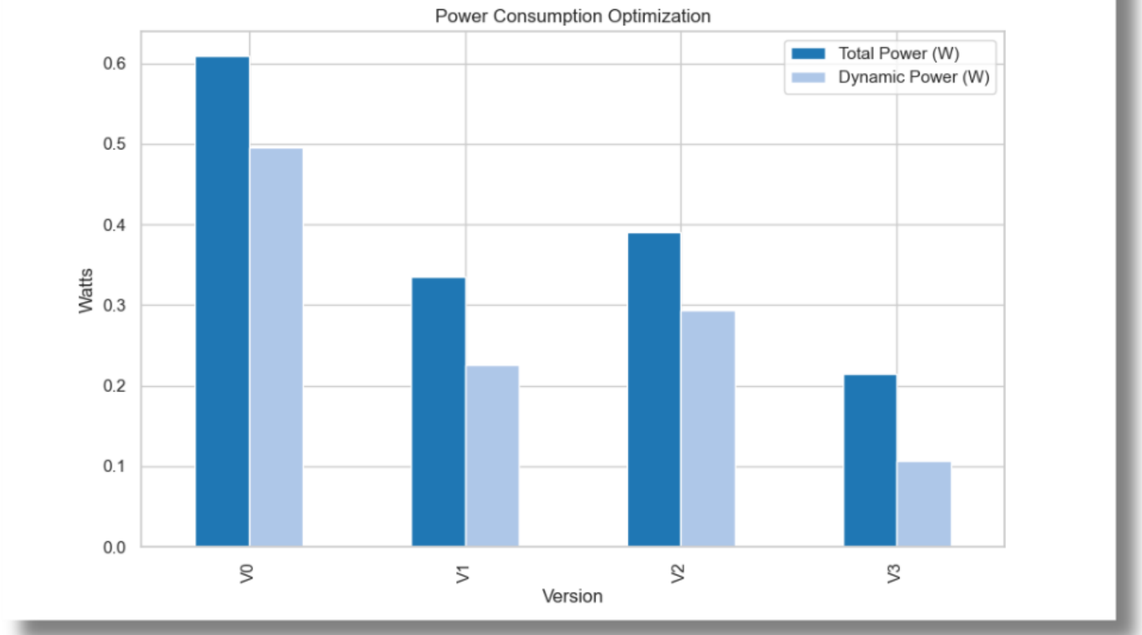
- Memory hierarchy (V1) → **BRAM migration**
- Critical execution path (V2) → **ALU, Forwarding, Branch logic**
- Pipeline control & activity gating (V3) → **Execute, ID/EX, EX/MEM, MEM/WB**

File	V0 → V1	V1 → V2
INSTRUCTION_MEMORY.v	Size 20B → 2KB, BRAM attribute	–
MEM_STAGE.v	BRAM attribute	–
REGFILE.v	Distributed RAM attribute	–
ALU.v	–	Parallel ops, CARRY4
FORWARDING_UNIT.v	–	Flattened logic
BRANCH_CONDITION_CHECKER.v	–	Pre-compute compa
EXECUTE_STAGE.v	–	–
ID_EX.v	–	–
EX_MEM.v	–	–
MEM_WB.v	–	–
constr.xdc	BRAM hints	75 → 90 MHz, pins

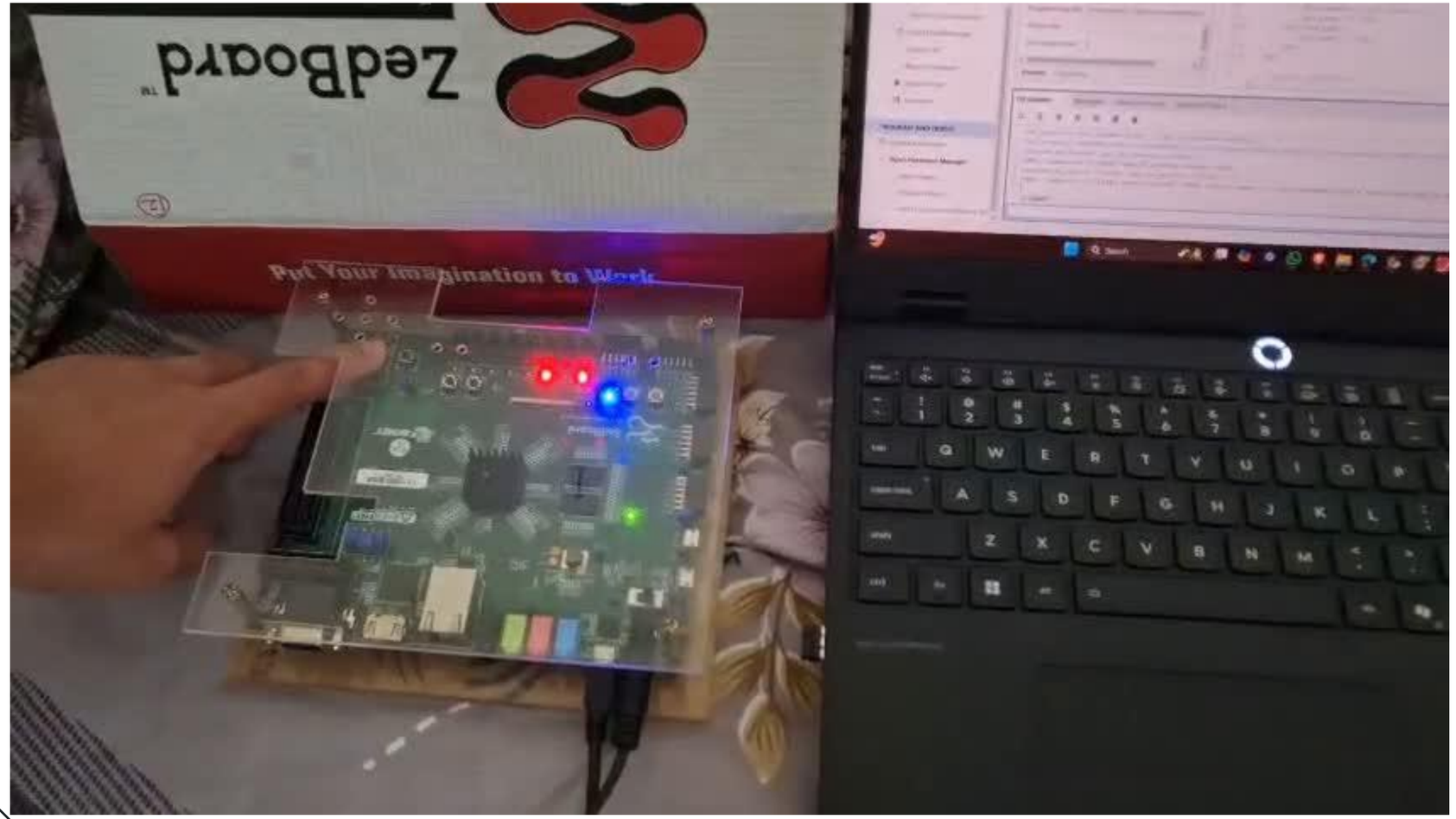
RESULTS


Performance vs. Energy Efficiency
Performance (MIPS)
Efficiency (MIPS/W)
145
140
135
130
125
400
350
300
250
200
V0
V1
V2
V3
Design Version


Frequency Tuning (Clock Speed)
95
90
85
80
75
70
Clock Frequency (MHz)
V0
V1
V2
V3
Version

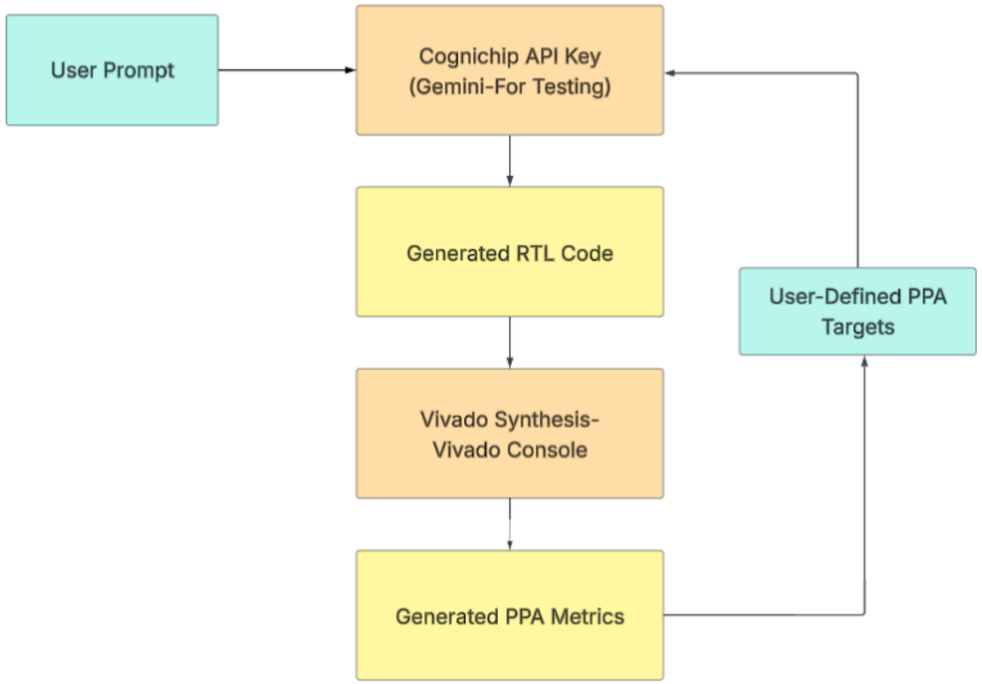

Power Consumption Optimization
Total Power (W)
Dynamic Power (W)
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Watts
V0
V1
V2
V3
Version

SIMULATION ON HARDWARE



FUTURE WORK

For our future work, we plan to build a closed-loop feedback system in anticipation of the Cognichip API release. A Python script will generate RTL from user specifications, run Vivado synthesis via TCL, extract PPA metrics, and feed them back to the API for iterative optimization toward target constraints. We are currently testing this flow with a Gemini API key, but it can be directly transitioned to the Cognichip API once available.



CONCLUSION

INTELLIGENT ARCHITECTURAL OPTIMIZATION

- Optimized a baseline RV32I in-order core
- LLM-guided exploration with synthesis feedback
- Achieved measurable PPA improvements
- Eliminated brute-force architectural search

RTL COMPLETION & ARCHITECTURAL REASONING

- Transformed fragmented RTL into a fully working 4-instruction MIPS processor
- Reconstructed missing datapath and control logic
- Delivered a synthesizable, functionally correct design

WHAT THIS PROVES

- LLMs can assist in early-stage architectural decisions
- LLMs can reason across datapath, control, and integration
- AI can act as a hardware co-designer, not just a code generator

COGNICHIP - IMPROVEMENTS?

ACCESS TO OTHER CHAT HISTORY

UPLOADING IMAGES IN CHAT

IMPROVING COGNICHIP TERMINAL

API KEYS

THANK you