# Microarchitectural Timing Optimization of the CV32E40P RISC-V Core on FPGA

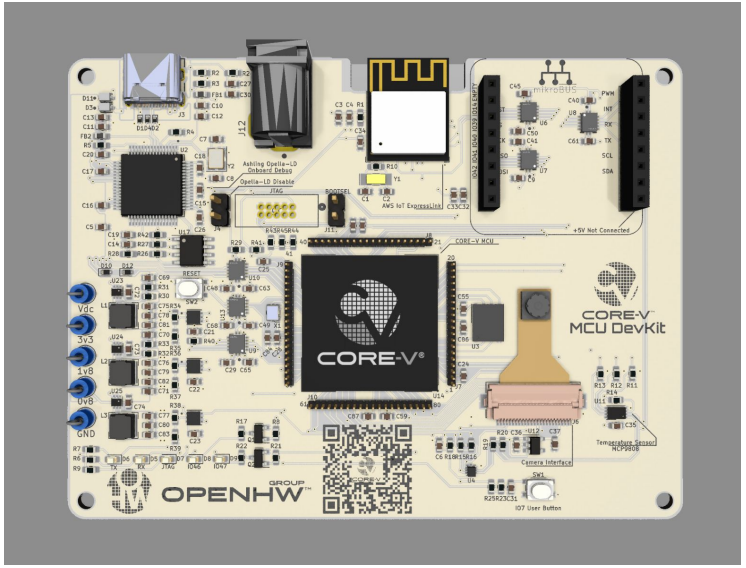



ECE 494 - Microprocessor Design

Tsiantos Dimitrios | Tsogkas Panagiotis Nikolaos | Nikas Ioannis Iason

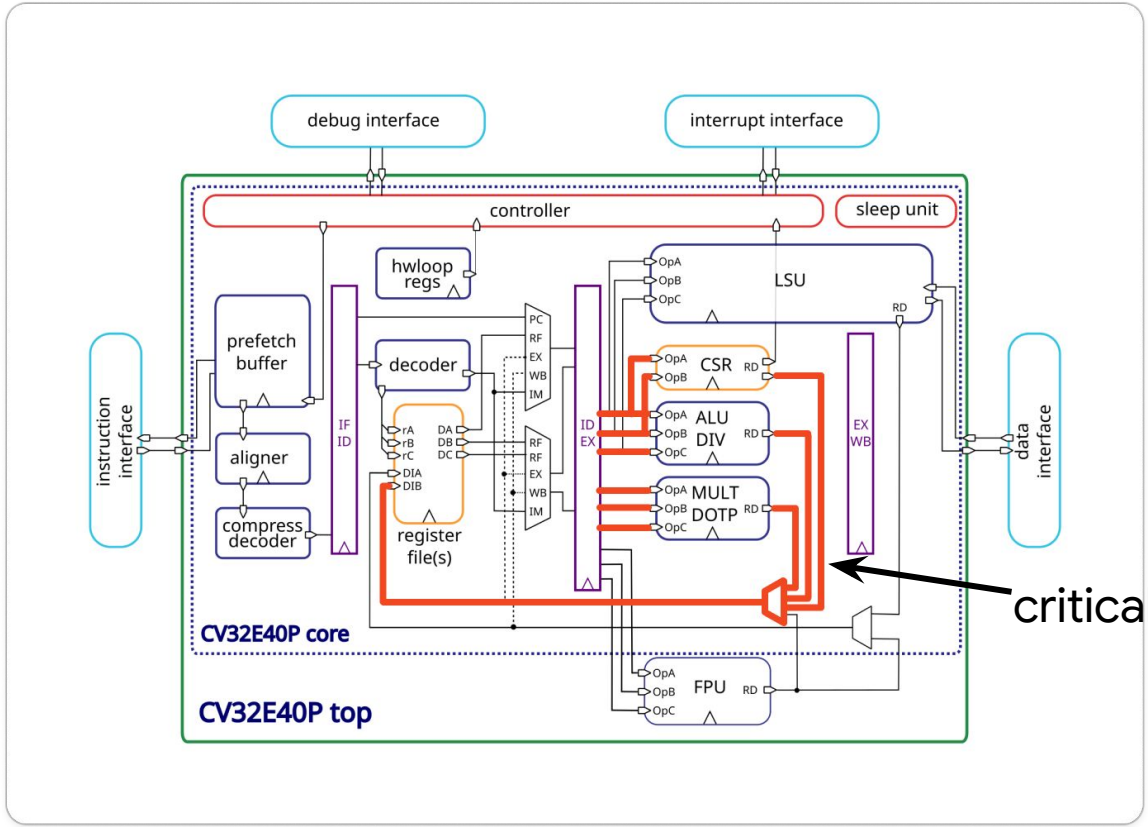
# CV32E40P Core

- **32-bit**
- **RISC-V** ISA
- **4-stage** pipeline
- **In-order** execution




CORE-V MCU DevKit featuring the cv32e40p core

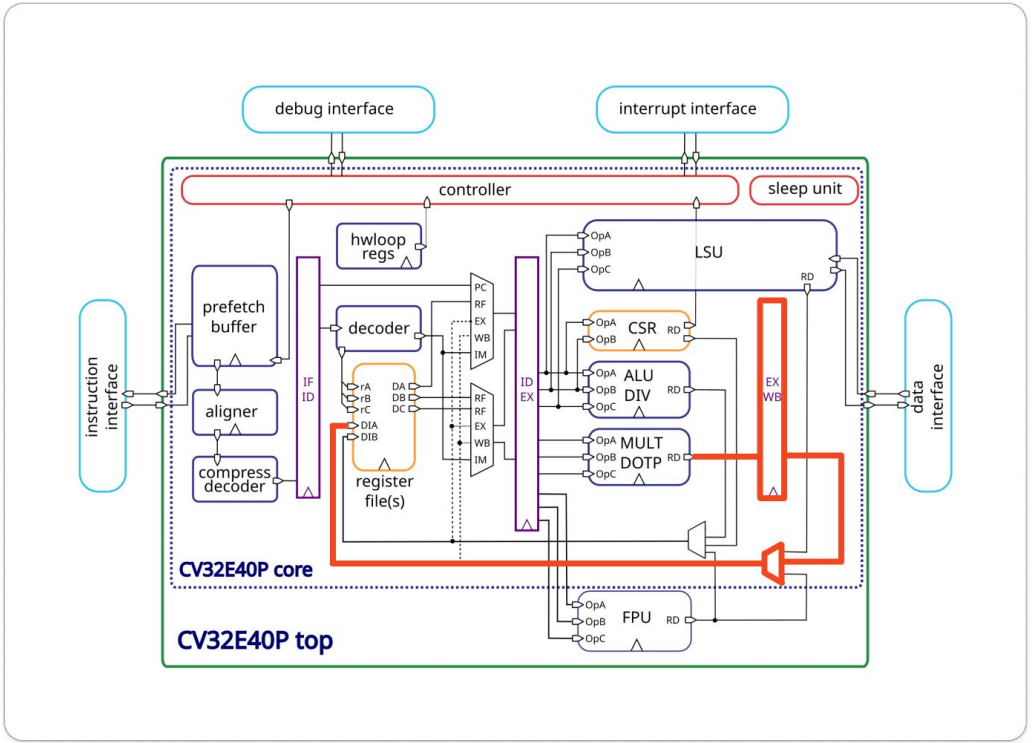
# EX→ID Forwarding



# Proposed Design

- Pass **MULT result** to **WB**
- Fix **hazard unit**

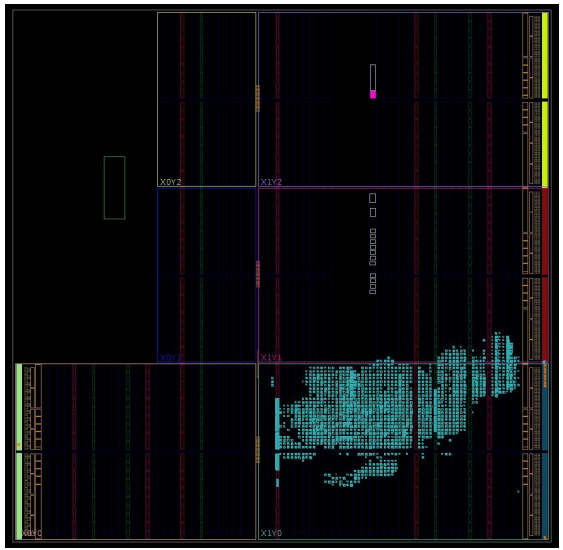
\+ Shorter combinational path
\+ Lower fanout
\- Mult stalls one cycle*

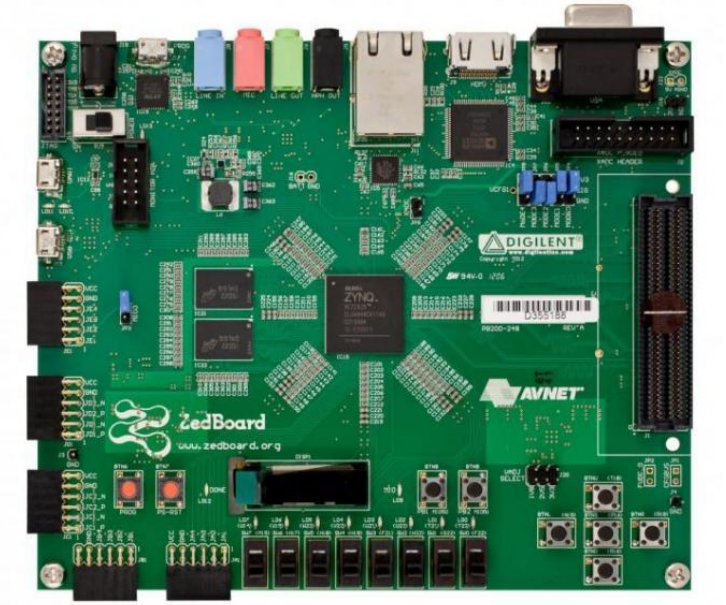



*only when dependency exists

# Methodologies

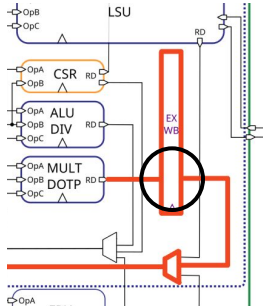
- **Implementation** on Zedboard (xc7z020clg484-1) using Vivado





# Implementation

- Add to the **EX/WB pipeline**



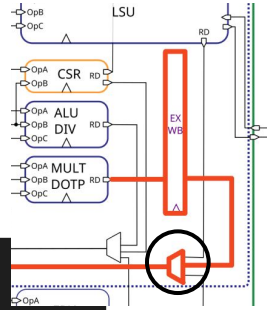

```
logic        regfile_we_mul_wb;    // whether a MULT writes back
logic [5:0]  regfile_waddr_mul_wb; // destination register rd
logic [31:0] regfile_wdata_mul_wb; // MULT result data
```

```
if (mult_en_i & regfile_alu_we_i) begin
  regfile_waddr_mul_wb <= regfile_alu_waddr_i;
  regfile_wdata_mul_wb <= mult_result;
end

// ...
regfile_we_mul_wb <= mult_en_i & regfile_alu_we_i;
```

# Implementation

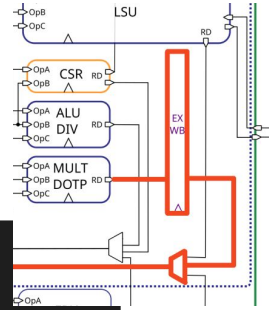
- Add to **write-back MUX**




```
// ...
else if (regfile_we_mul_wb) begin
  regfile_we_wb_o       = 1'b1;
  regfile_we_wb_power_o = 1'b1;
  regfile_waddr_wb_o    = regfile_waddr_mul_wb;
  regfile_wdata_wb_o    = regfile_wdata_mul_wb;
end
```

# Implementation

- **Stall** when a dependency exists



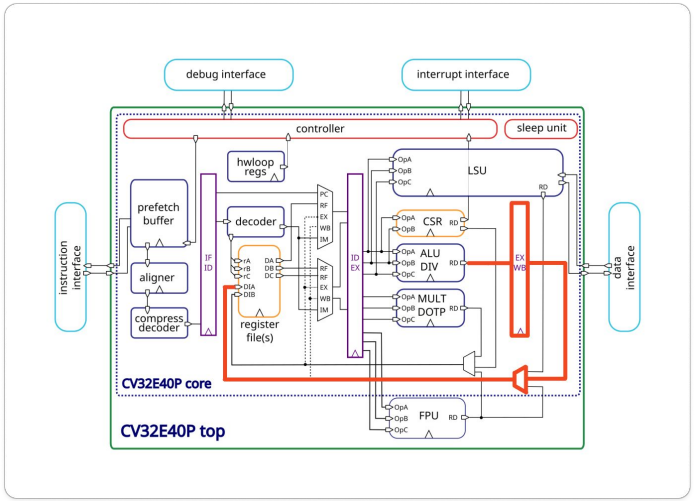

```
// Stall one cycle for consumers of a MUL result. MUL results are registered
// into the EX/WB stage and are only available for forwarding from WB.
if ((mult_en_ex_i == 1'b1) && (regfile_alu_we_ex_i == 1'b1) &&
    ((reg_d_alu_is_reg_a_i == 1'b1) ||
     (reg_d_alu_is_reg_b_i == 1'b1) ||
     (reg_d_alu_is_reg_c_i == 1'b1)))
begin
  deassert_we_o   = 1'b1;
  load_stall_o    = 1'b1;
end
```

is mul

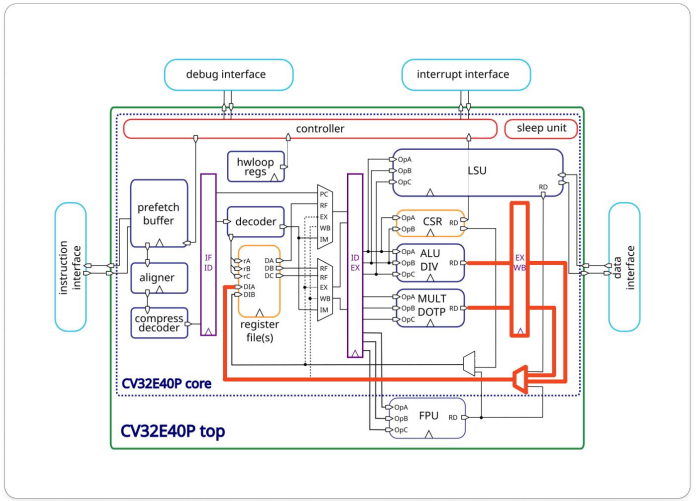
EX writes to regfile

ID reads rd
(mul rd, rs, rt)

# Additional Designs



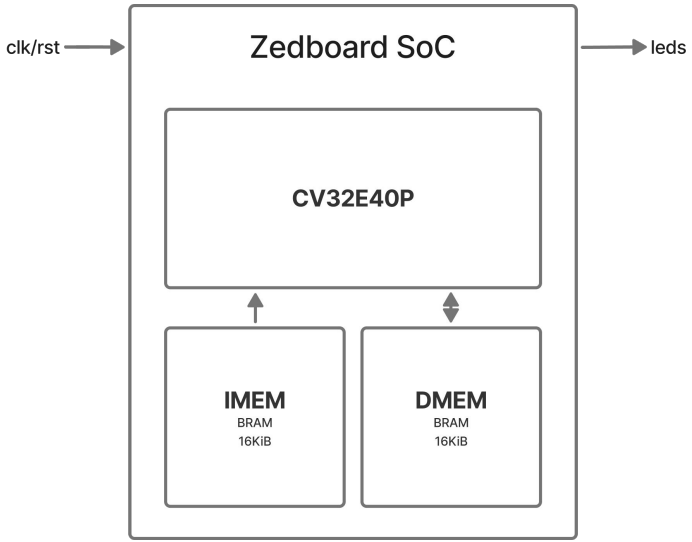

2) No ALU fwd



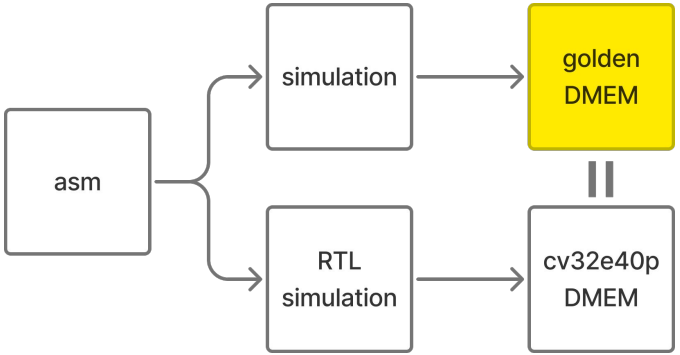

3) No MUL & ALU fwd

# Setup & Testing

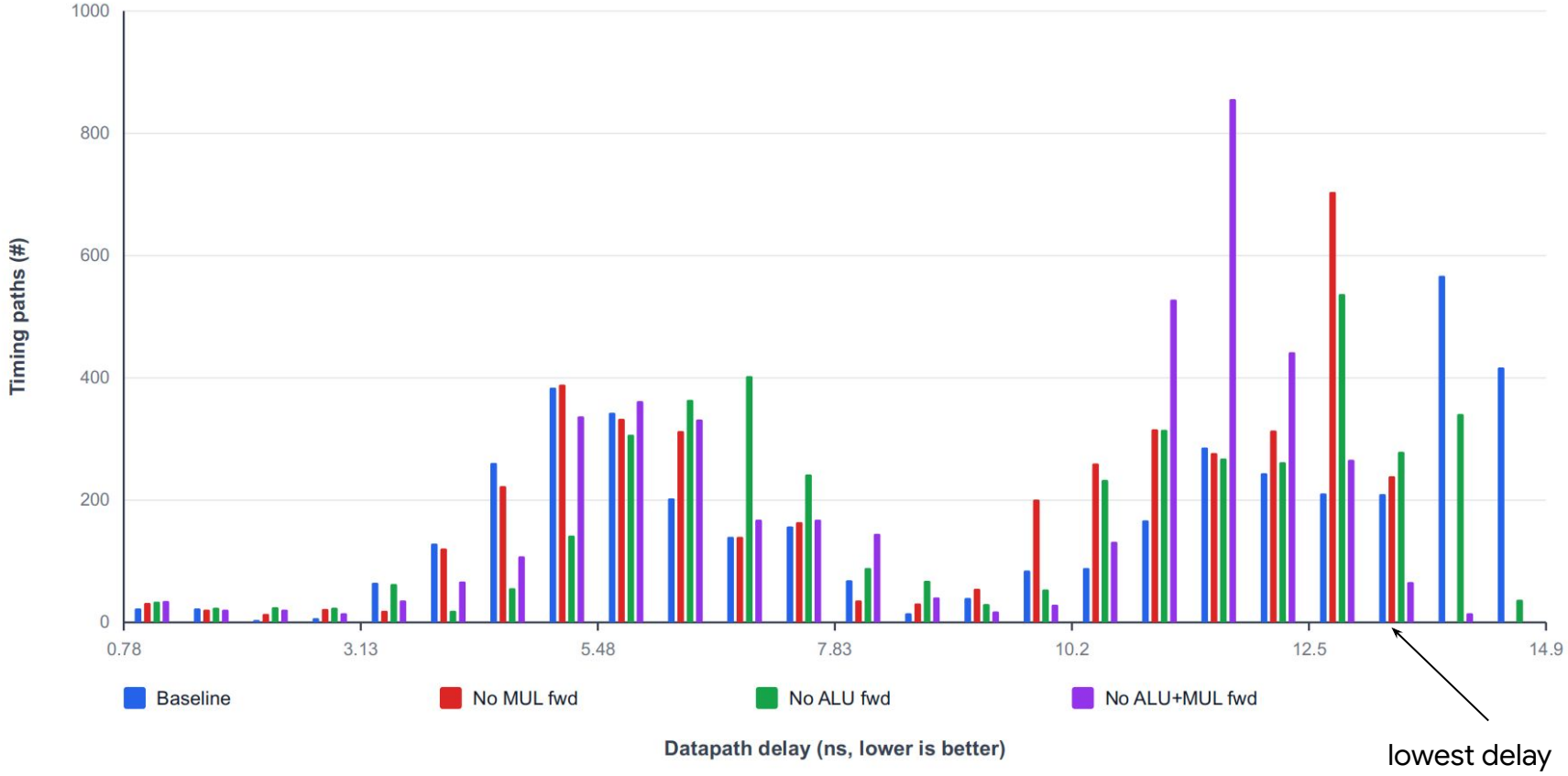
- Minimal runnable **FPGA SoC** (ZedBoard)




- RTL **testbench** and golden **simulation** (in python)

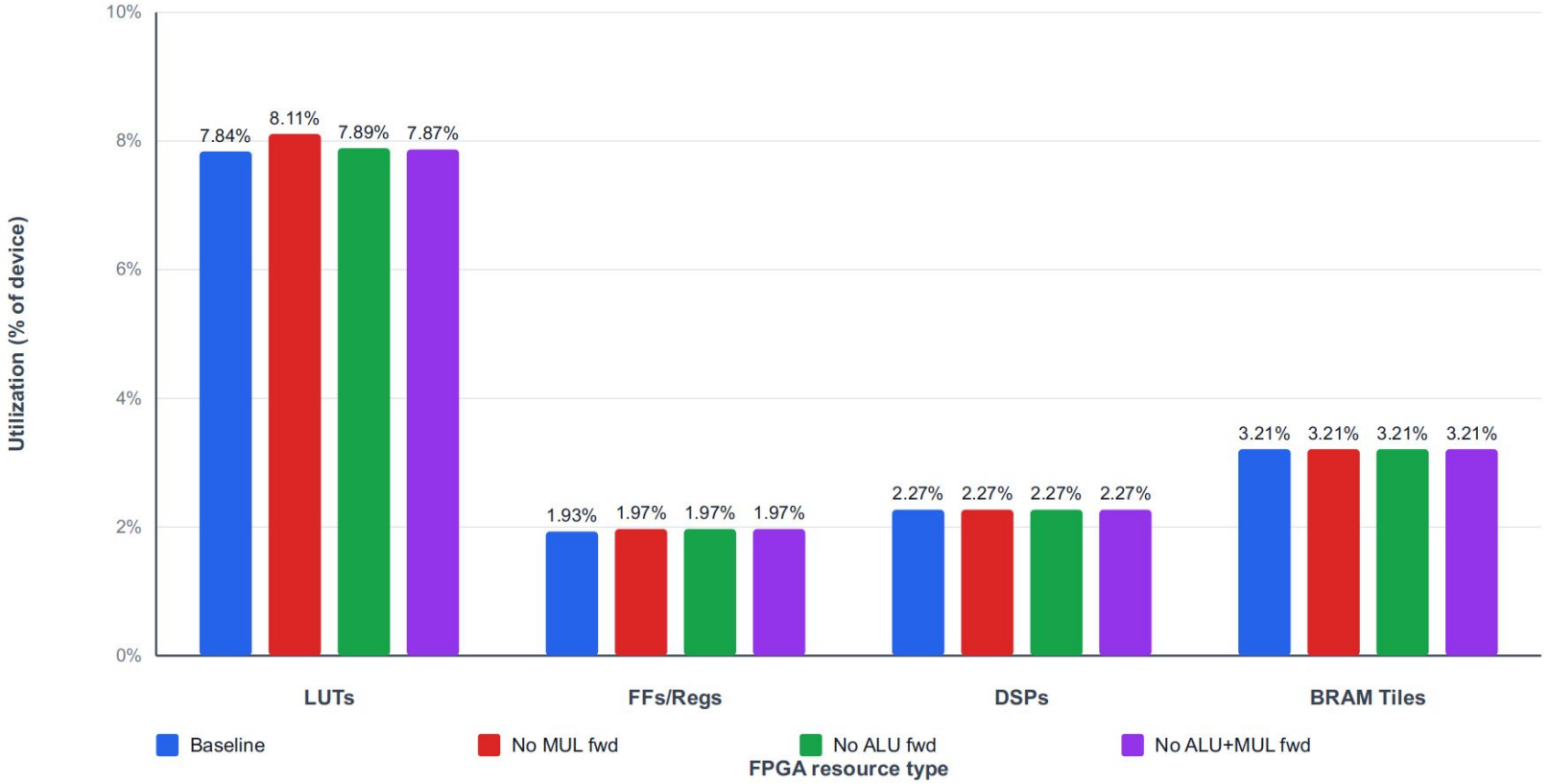


# Datapath Delay Histogram

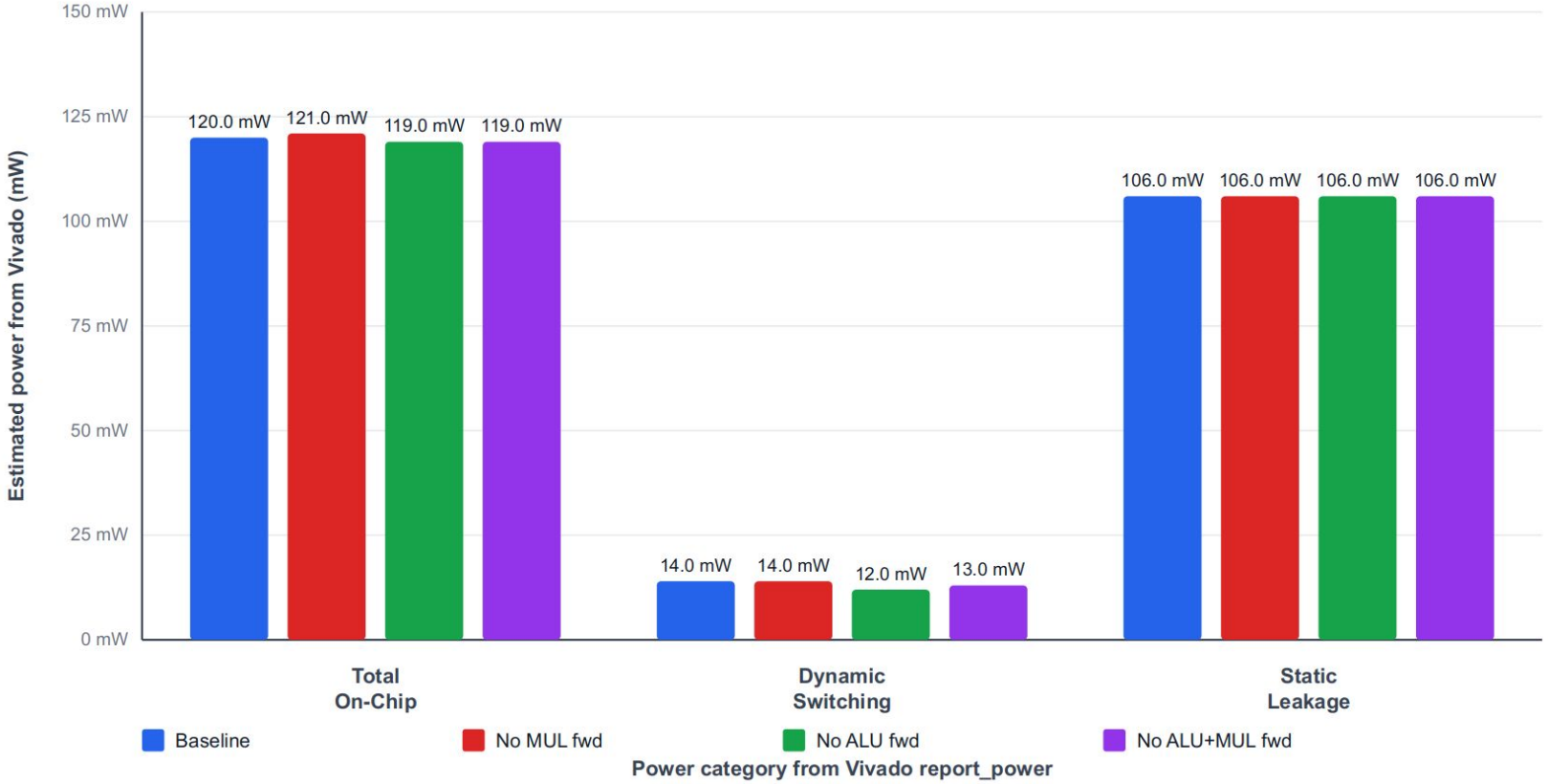



*first 1000 paths

# Utilization



# Power


150 mW
125 mW
100 mW
75 mW
50 mW
25 mW
0 mW
Estimated power from Vivado (mW)
120.0 mW
121.0 mW
119.0 mW
119.0 mW
14.0 mW
14.0 mW
12.0 mW
13.0 mW
106.0 mW
106.0 mW
106.0 mW
106.0 mW
Total On-Chip
Dynamic Switching
Static Leakage
Baseline
No MUL fwd
No ALU fwd
No ALU+MUL fwd
Power category from Vivado report_power

# Design Comparison

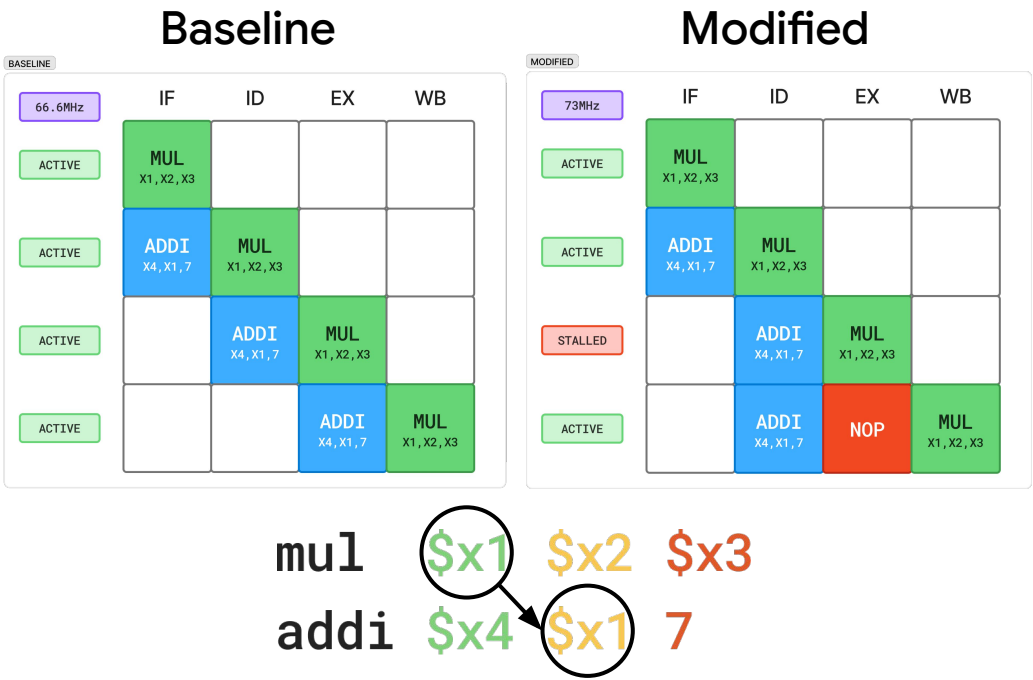
| | Baseline | Improved | Delta |
|---|---|---|---|
| Target frequency | 66.667 MHz | **71.429 MHz** | **+4.762 MHz (+7.1%)** |
| Estimated Fmax | 66.8 MHz | **73.201 MHz** | **+6.401 MHz (+9.6%)** |
| WNS | 0.03 ns | **0.339 ns** | **+0.309 ns** |
| Critical path delay | 14.871 | **13.44 ns** | **-1.431 ns (-9.6%)** |

With minimal area increase (+0.27% LUTs)

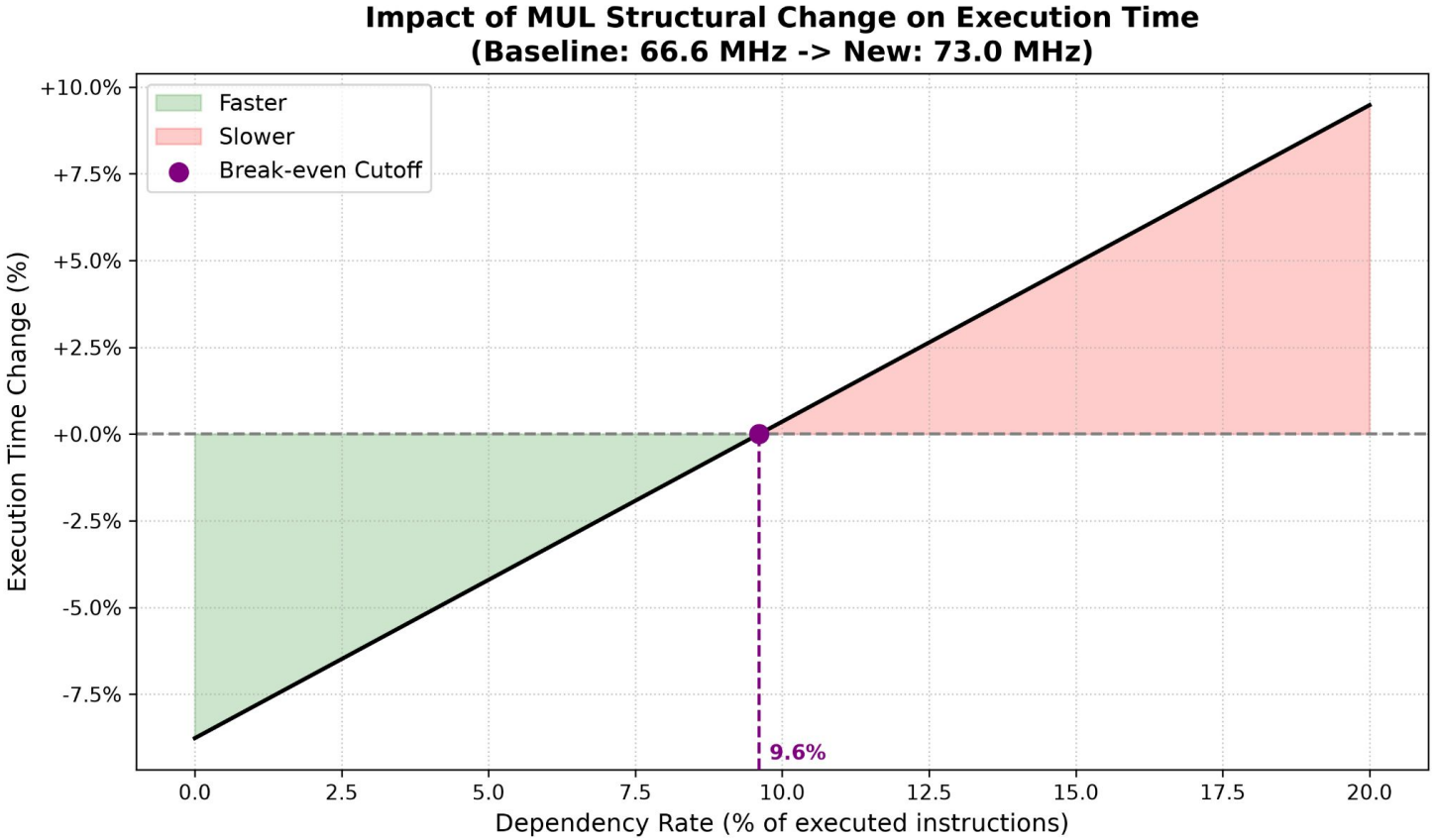
# Pipeline Side-by-Side

Pipeline comparison:

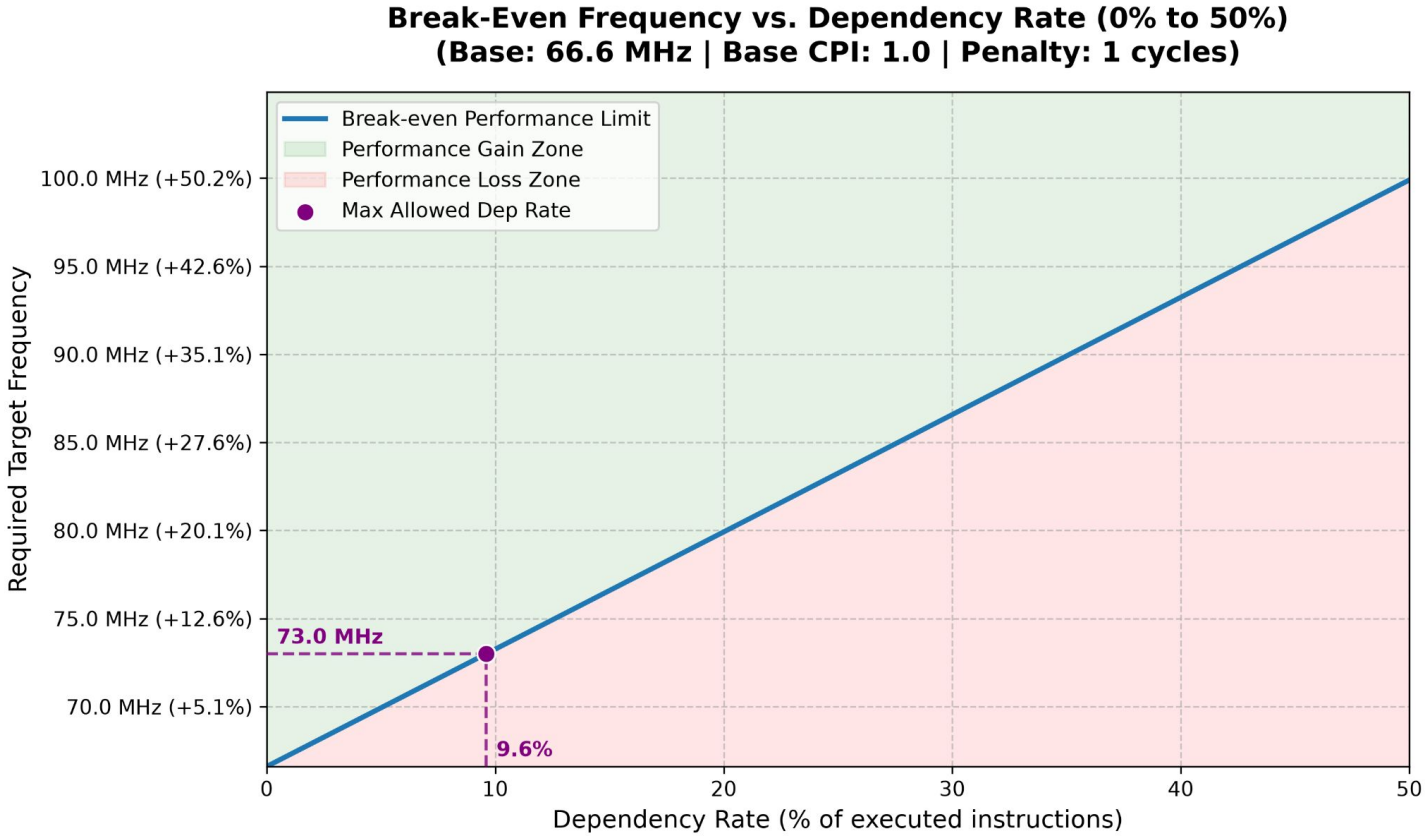
- **Faster** but **simpler** core philosophy
- Better for specific use cases



# Performance Metrics



# Target Frequency Estimation



# Conclusion: Examples of Ideal Use Cases

- Control-Dominated Workloads (IoT)
  - These tasks rely heavily on basic ALU operations, branches, and memory access.
- Cryptography (Hashing & Encryption)
  - Algorithms like **AES** and **SHA-256** rely on bit-shifting, XORs, and additions.
- Network Packet Processing
  - Managing network traffic involves parsing headers, moving payload data, and computing checksums.

# Questions?




Tsiantos Dimitrios | Tsogkas Panagiotis Nikolaos | Nikas Ioannis Iason