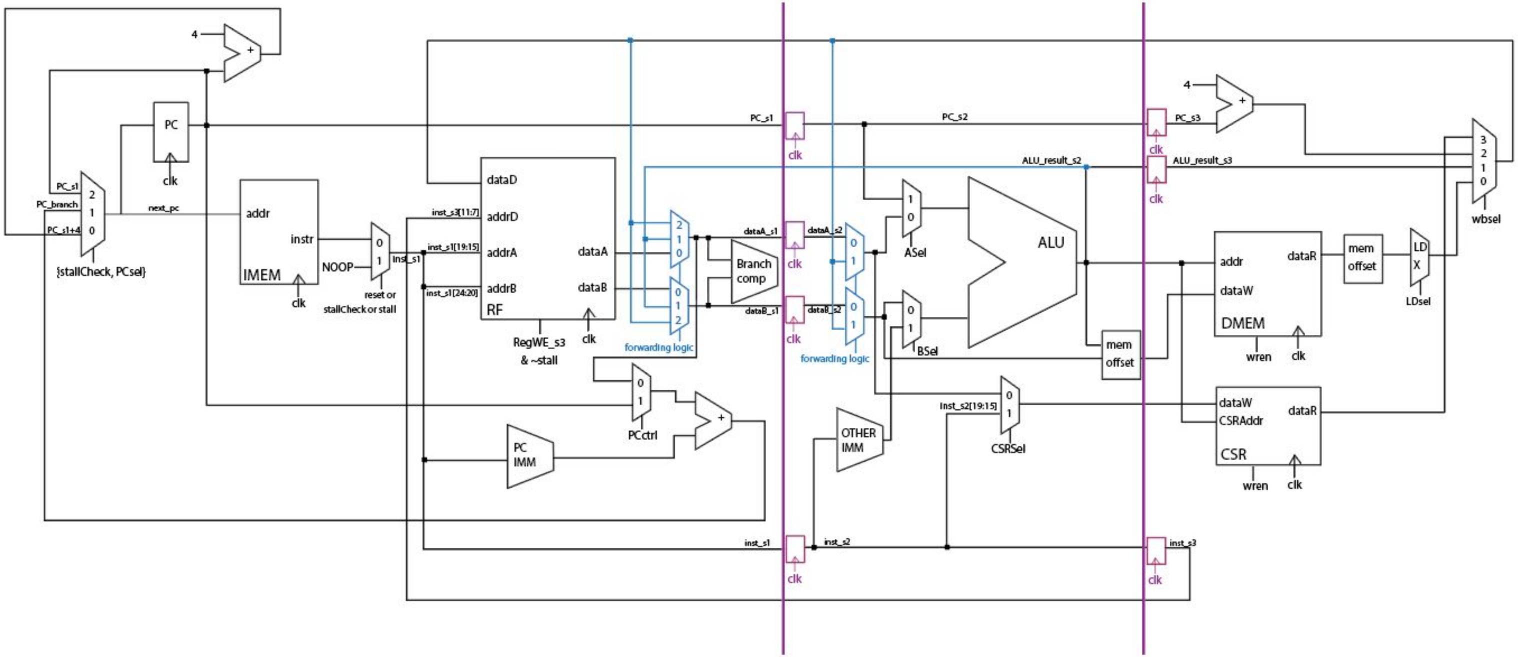
EECS 251A: Team 6

Eric Wu & Lisa Qing

Implementation

CPU & Cache & Baseline Results

3-Stage Pipeline Diagram



control logic

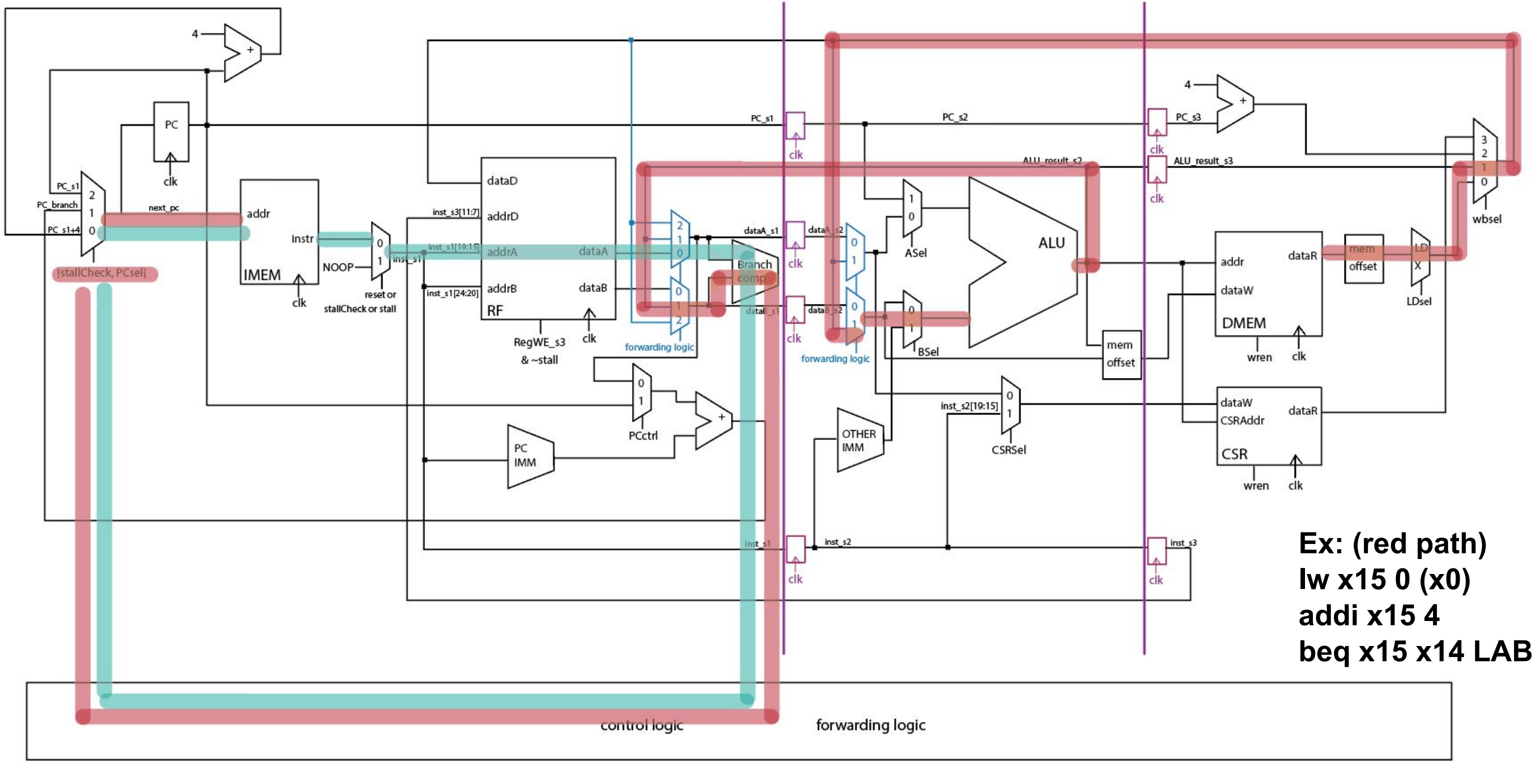
forwarding logic

3-Stage Pipeline Diagram

Design Choices

1. Stages: IF & D | EX | MEM & WB
2. Branch comparator at Decode stage
    a. Pro: less control hazards
    b. Con: longer critical path, extra forwarding muxes for branch comparator, and requires 1 stall for a load then branch/jump instruction

Post-Synthesis Critical Paths



Ex: (red path)
lw x15 0 (x0)
addi x15 4
beq x15 x14 LABEL

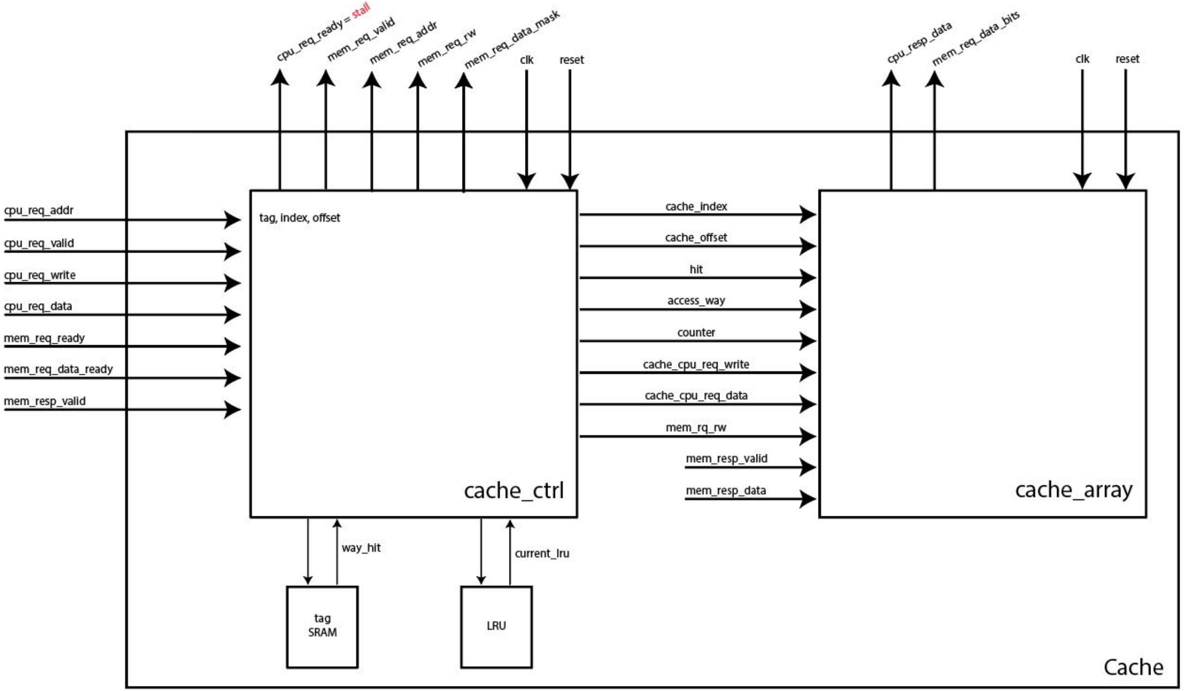
Cache Block Diagram

Two modules:

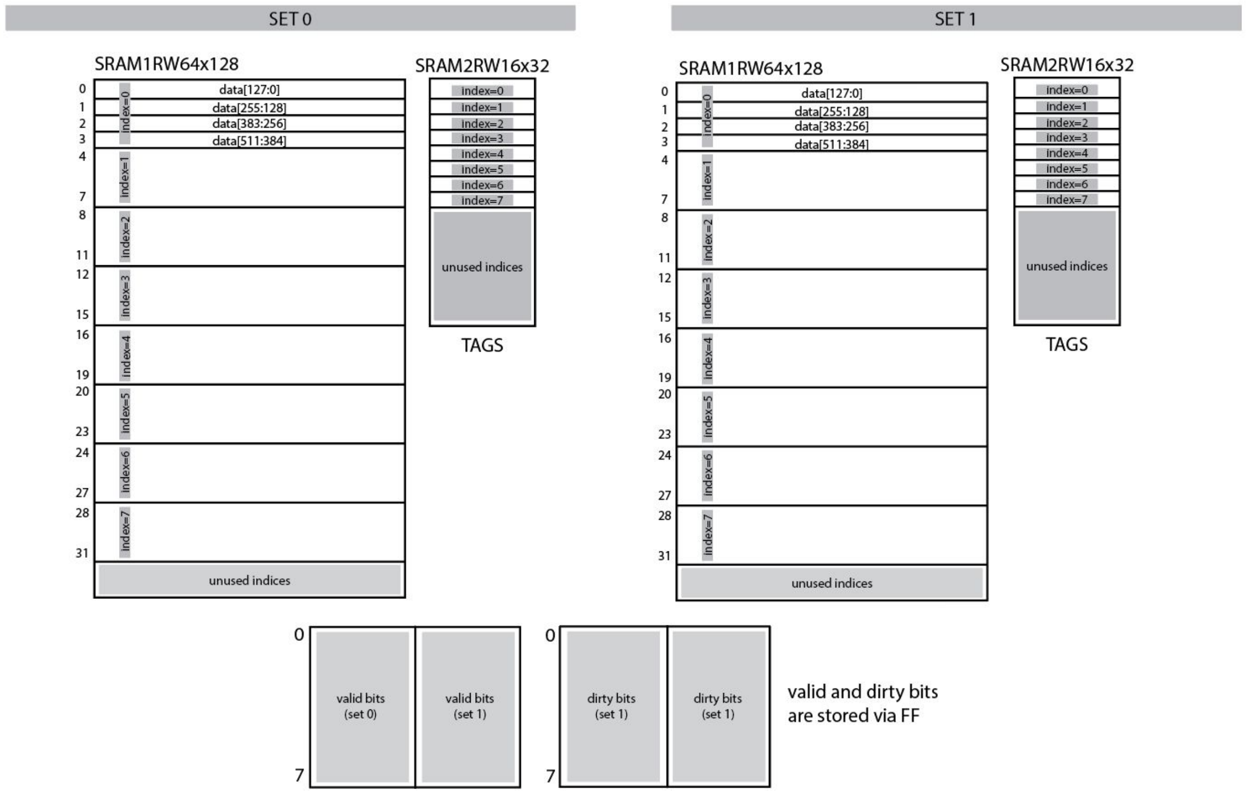
- Cache controller (FSM logic)
- Cache array

Additional Note:

- Accessing cache will stall all stages in the CPU
- LRU Replacement Policy



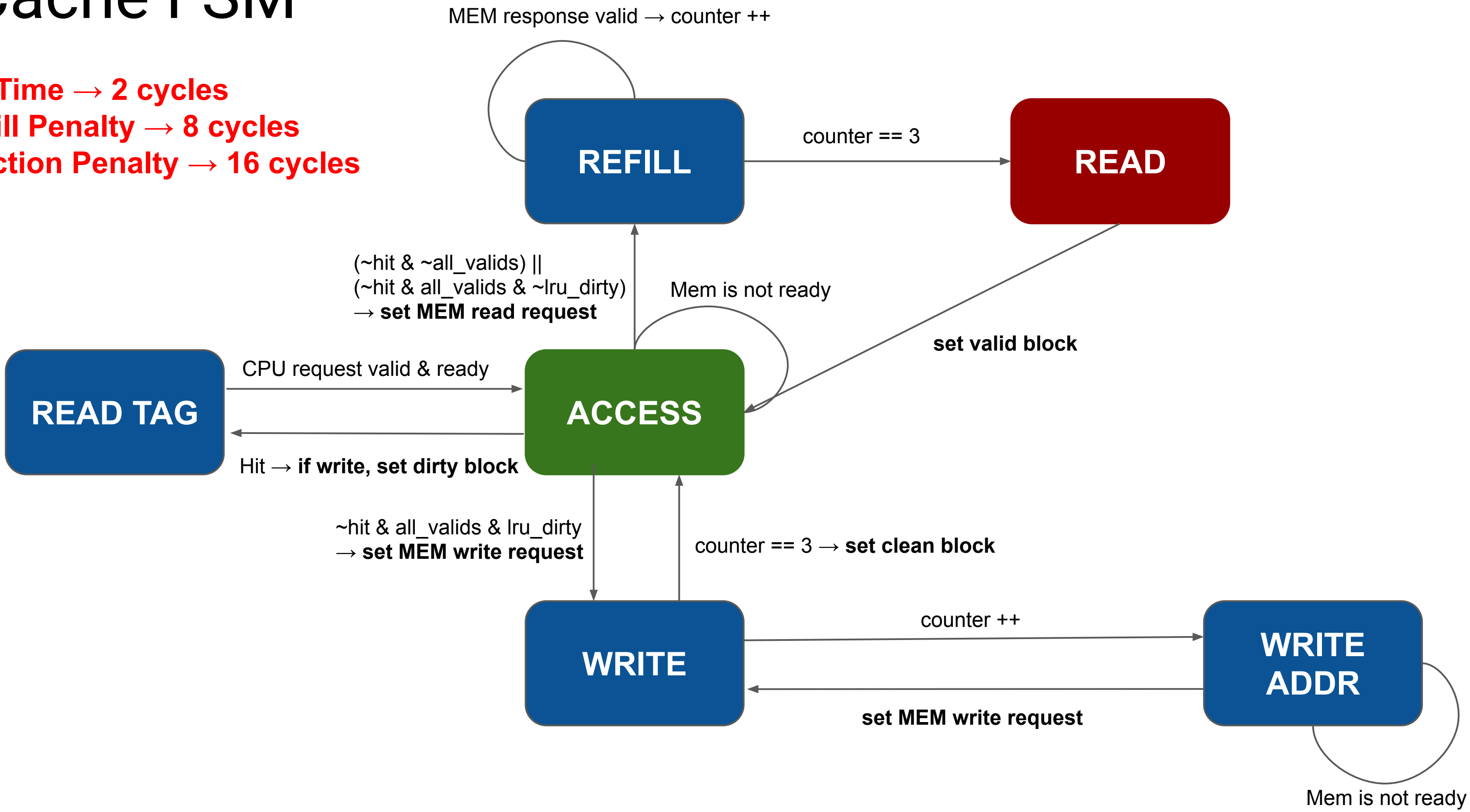
Cache Design (2-way 1kB example)



valid and dirty bits
are stored via FF

Cache FSM

Hit Time → 2 cycles
Refill Penalty → 8 cycles
Eviction Penalty → 16 cycles



MEM response valid → counter ++

REFILL

counter == 3

READ

(~hit & ~all_valids) ||
(~hit & all_valids & ~lru_dirty)
→ **set MEM read request**

Mem is not ready

set valid block

READ TAG

CPU request valid & ready

ACCESS

Hit → **if write, set dirty block**

~hit & all_valids & lru_dirty
→ **set MEM write request**

counter == 3 → **set clean block**

WRITE

counter ++

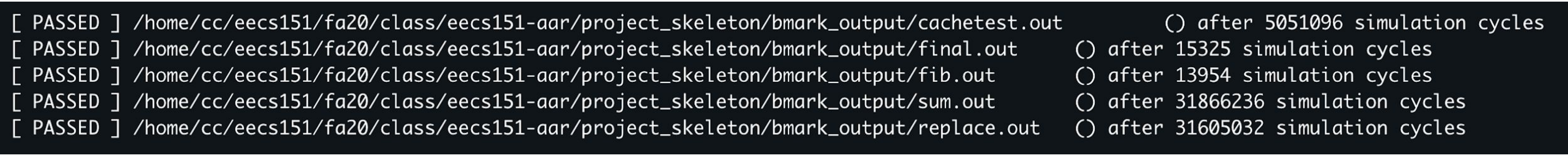
WRITE ADDR

set MEM write request

Mem is not ready

Baseline Results (RTL-SIM)

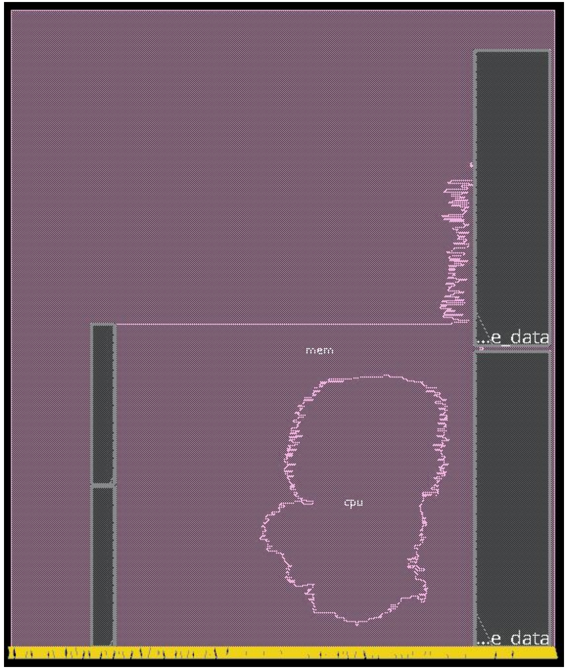
1kB Direct-mapped cache with (1RW tag ram & 1RW data ram w/o bytemask)	
cachetest	5,051,096
final	15,325
fib	13,954
sum	**31,866,236**
replace	31,605,032

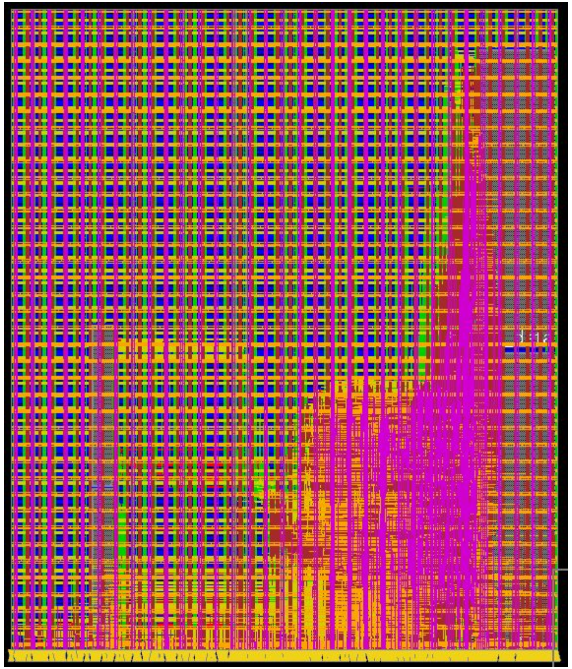


```
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/bmark_output/cachetest.out     () after 5051096 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/bmark_output/final.out     () after 15325 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/bmark_output/fib.out       () after 13954 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/bmark_output/sum.out       () after 31866236 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/bmark_output/replace.out   () after 31605032 simulation cycles
```

Floorplan (1kB DM) → 470 x 550 μm^2

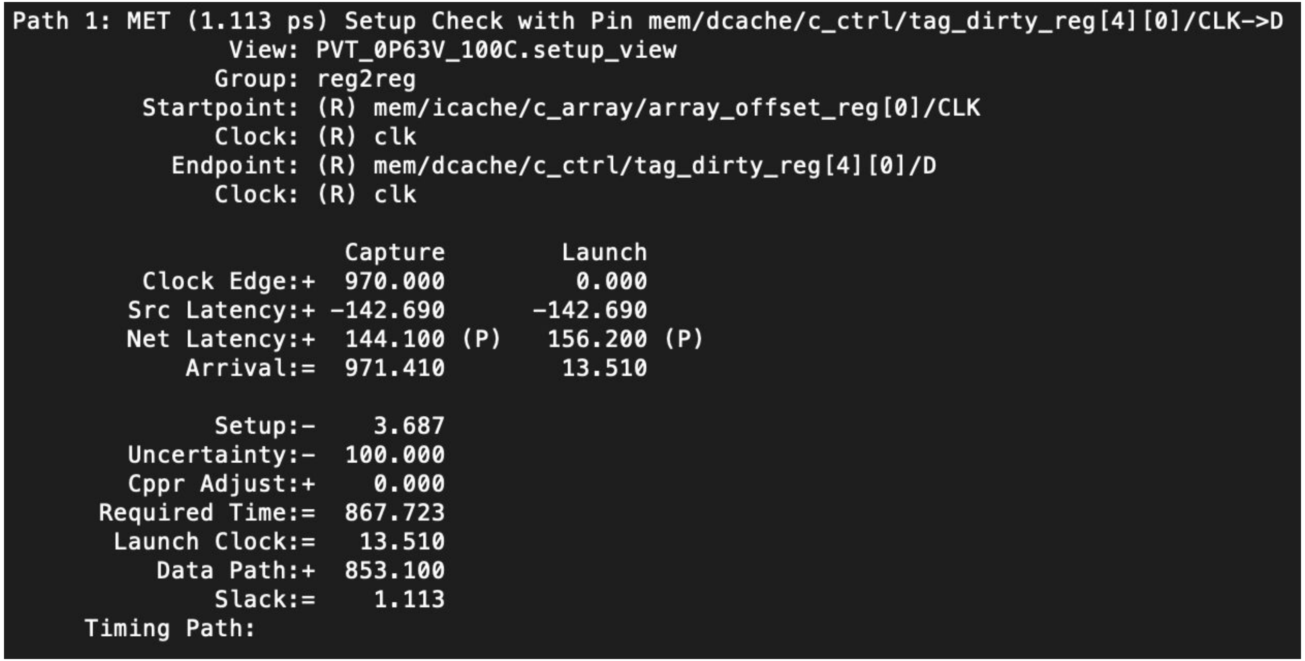
- Design density (pre-filler) : 13.9%





Timing Report (1kB DM)

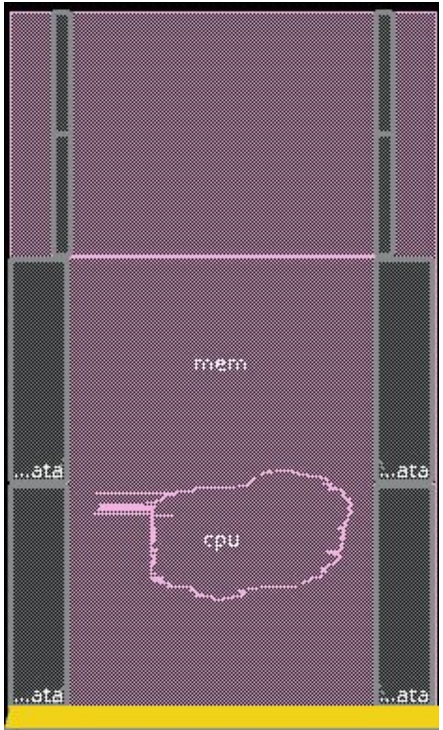
- Min cycle time: 0.97ns

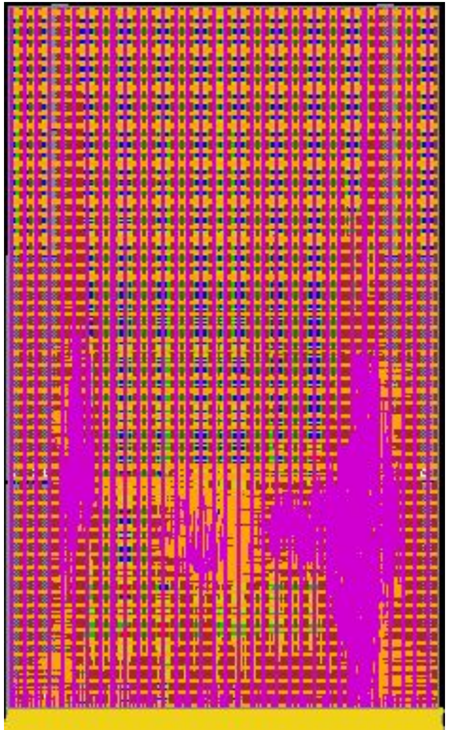


```
Path 1: MET (1.113 ps) Setup Check with Pin mem/dcache/c_ctrl/tag_dirty_reg[4][0]/CLK->D
               View: PVT_0P63V_100C.setup_view
              Group: reg2reg
         Startpoint: (R) mem/icache/c_array/array_offset_reg[0]/CLK
              Clock: (R) clk
           Endpoint: (R) mem/dcache/c_ctrl/tag_dirty_reg[4][0]/D
              Clock: (R) clk

                       Capture        Launch
         Clock Edge:+  970.000         0.000
        Src Latency:+ -142.690      -142.690
        Net Latency:+  144.100 (P)   156.200 (P)
            Arrival:=  971.410        13.510

              Setup:-    3.687
        Uncertainty:-  100.000
        Cppr Adjust:+    0.000
      Required Time:=  867.723
       Launch Clock:=   13.510
          Data Path:+  853.100
              Slack:=    1.113
     Timing Path:
```

Floorplan (1kB 2-way) → 490 x 800 μm^2

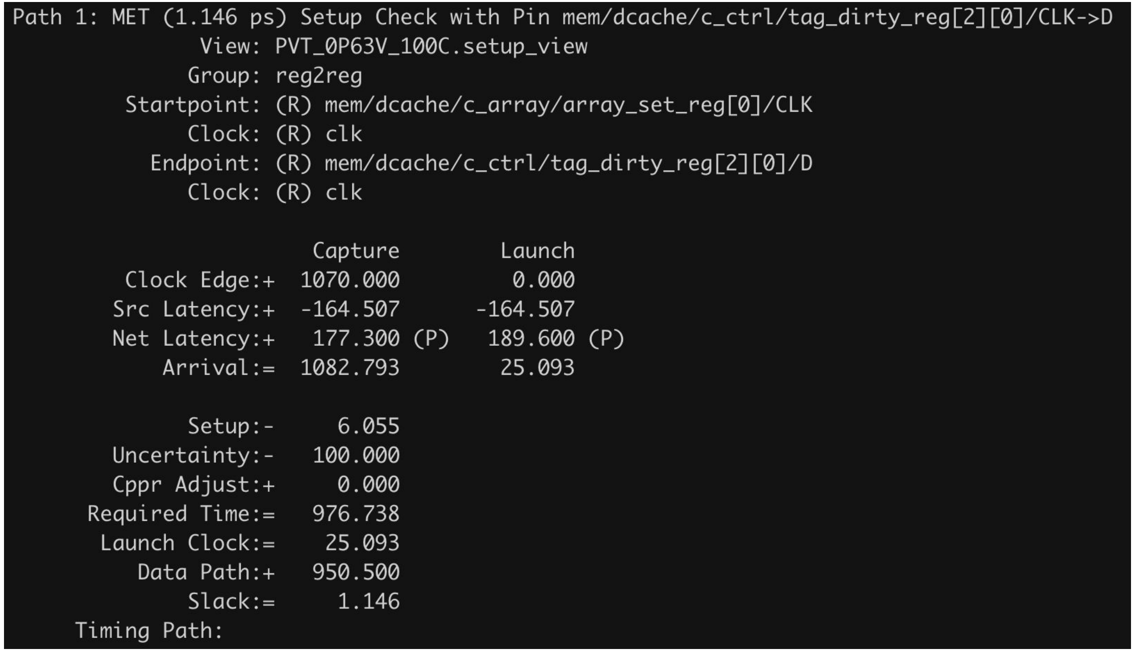
- Design density (pre-filler) : 6%





Timing Report (1kB 2-way)

- Min cycle time: 1.07 ns



```
Path 1: MET (1.146 ps) Setup Check with Pin mem/dcache/c_ctrl/tag_dirty_reg[2][0]/CLK->D
               View: PVT_0P63V_100C.setup_view
              Group: reg2reg
         Startpoint: (R) mem/dcache/c_array/array_set_reg[0]/CLK
              Clock: (R) clk
           Endpoint: (R) mem/dcache/c_ctrl/tag_dirty_reg[2][0]/D
              Clock: (R) clk

                        Capture        Launch
         Clock Edge:+  1070.000         0.000
        Src Latency:+  -164.507      -164.507
        Net Latency:+   177.300 (P)   189.600 (P)
            Arrival:=  1082.793        25.093

              Setup:-     6.055
        Uncertainty:-   100.000
        Cppr Adjust:+     0.000
      Required Time:=   976.738
       Launch Clock:=    25.093
          Data Path:+   950.500
              Slack:=     1.146
     Timing Path:
```

Optimization

Cycle Counts & Critical Path

Optimization

Optimizing for number of cycles

1. Using 2RW tag ram instead of 1RW tag ram → less access cycles
2. Byte mask data ram → less penalty in cache miss by removing READ state
3. Different Cache Configuration → less miss rate

Optimizing for frequency (Critical path: lw → add → beq with data forwarding)

1. Change SRAM macros positions in the floorplan
2. Set false path from mem_req_ready (large delay) → muxwb
3. Move branch comparator to EXE stage and add branch predictor

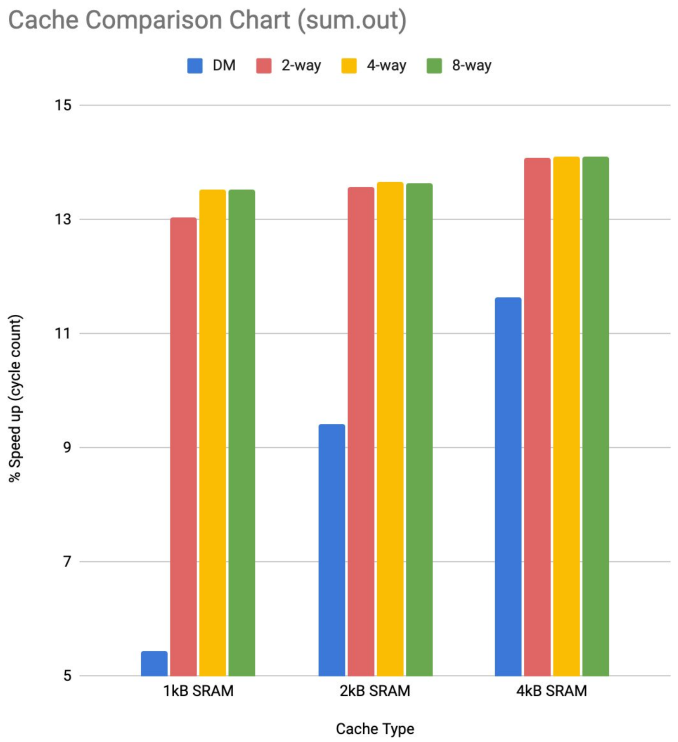
Cache Configuration Comparison (RTL-SIM)

Chart illustrates the % speed up (cycle count optimization) for different cache sizes & set associativity for **2RW Tag Ram w/ Byte Mask Data Ram**

Baseline (1kB DM Cache w/ 1RW Tag Ram w/o Byte Mask): **31,866,236**

Best Result (4kB 4-way Cache w/ 2RW Tag Ram w/ Byte Mask): **27,371,328**

14.1 % speed up



Final Results (cycle counts * min cycle time)

Execution time = Cycle counts * min cycle time

Cache Configuration (baseline): 1kB DM w/ **1RW tag ram w/o mask**

- Exec. Time = 31,866,236 * 0.97 = **30.910 ms**

Cache Configuration (optimized): 1kB 2-way w/ **2RW tag ram w/o mask**

- Exec. Time = 27,713,413 * 1.07 = **29.653 ms**
- Optimized cycle count trade off with frequency
    - Larger memory may require longer access time

Overall 4.07% speed up

Obstacles

Bugs & How we resolve them

Major Bugs

- Load to branch data hazard
  - Insert noop if these instructions are found and stall PC
    - However, this instruction stall that inserts noops is different from the memory stall, which needs to keep the current instructions in the pipeline
- Reading garbage initial data of valid and dirty bits from SRAM
  - SRAM doesn't have a "reset" → use flip-flops instead (separate from tag SRAM)
- CPU write miss to cache
  - At most we only write 32 bits to cache, but each entry in the SRAM is 128 bits
    - Must refill then read to get the present data then mask and write to SRAM
    - Add a READ state after REFILL state
- Area issue
  - Initial design used 16 SRAM2RW16x32 per way → takes a lot of area (38 SRAMS for 2-way)
  - New design to use SRAM1RW64x128 per way (8 SRAMS for 2-way)

Debug Schemes

1. Monitor the signals via testbench (log file)
2. Verify the content in the register files to see which parts of the assembly fails
3. Breakdown the C program into the minimum loop cycles to find the errors
4. Dve to get the current content of the register file and cache

Q&A

Thank you!