EECS151 ASIC Project Report

Yung-Chun (Andrew) Chen 3039819175

Chu-Wen (Owen) Chen 3039820410

May 8, 2024

Problem 1: Team Information

Please list the full name of the individuals. If you worked only, then write your name down as *Project Partner 1* and leave *Project Partner 2* blank.

Project Partner 1: Yung-Chun (Andrew) Chen

Project Partner 2: Chu-Wen (Owen) Chen

Problem 2: Design

2.1 Summary:

Core Functional? (Yes/No): Yes

Number of Pipeline Stages: 4

Forwarding Implemented? (Yes/No): Yes

Branch Prediction Implemented? (Yes/No): Always predict not taken

Cache Implemented? (Yes/No): Yes

Cache Functional? (Yes/No): Yes

Cache Size: 4KB

Cache Associativity: Direct-mapped

Cache Read Miss Latency: 6 cycles without eviction, 14 cycles with eviction

Cache Read Hit Latency: 1 cycle (synchronous read)

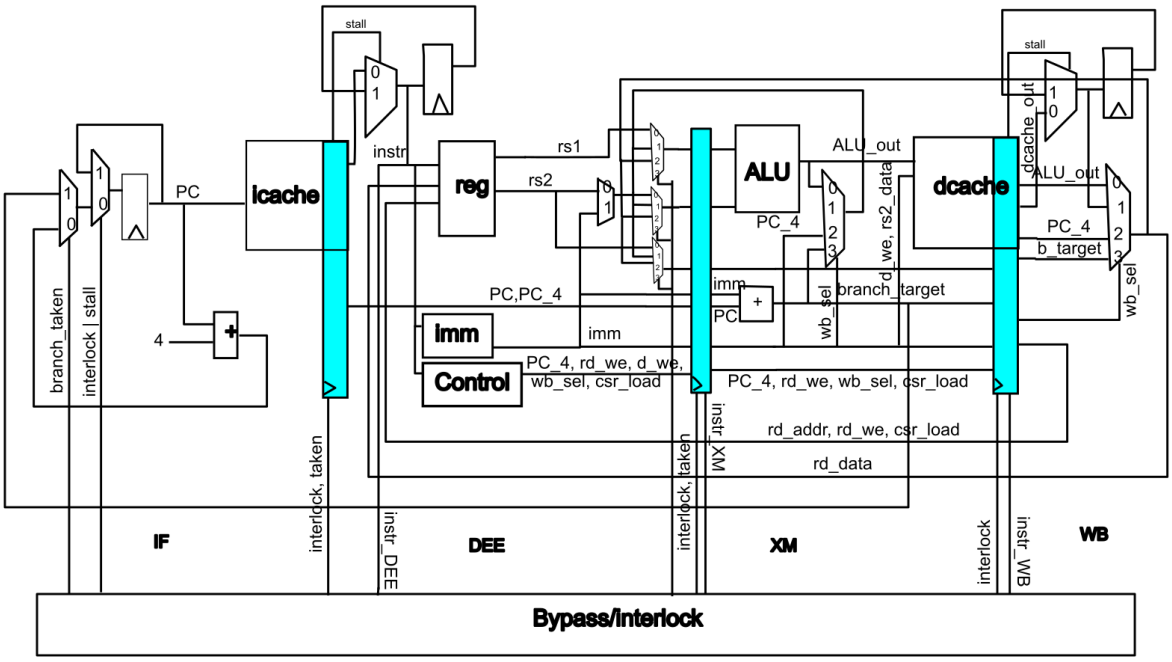
Cache Write Miss Latency: 6 cycles without eviction, 14 cycles with eviction

Cache Write Hit Latency: 1 cycle (synchronous write)

Other Optimization (Yes/No; please describe in later section):

2.2 Pipeline Design

[*Please insert diagram of pipeline diagram here*]



Explain your pipeline at high level below. Be sure to state what occurs in each stage, how hazards are handled, and what forwarding, if any, is implemented. Include any other information you believe TAs should know about, especially if your core is not fully functionally.

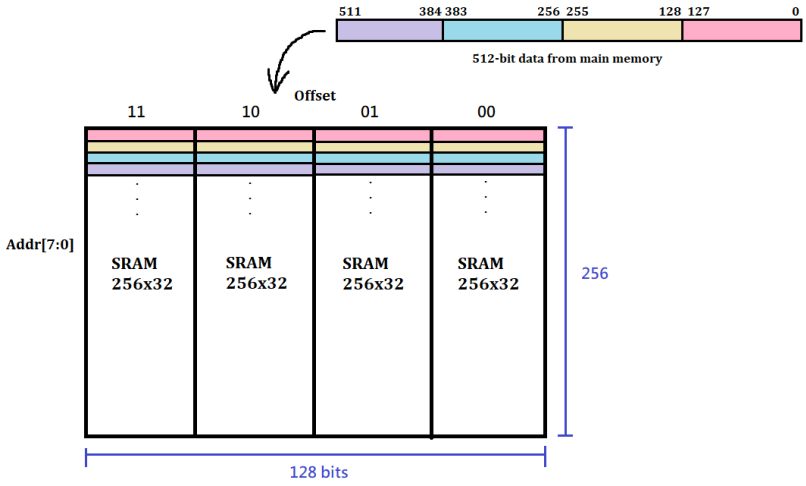
Our CPU is a 4-stage pipeline diagram which compose with IF, DEE, XM, WB. It's similar with classic RISCV 5-stage pipeline, but we merge EX and MEM with one stage and bypass the value to the end of decode stage. PC used as icache address in IF stage and we get instruction in DEE stage. We decode control signal we need for each instruction and generate immediate. In XM stage, we resolve branch and jump instruction and do arithmetic. We pass the address for dcache if needed and read/write the value. In WB, we select the data we want to write back or write to csr.

Most of the data hazards could be resolved by our bypass path from XM stage and WB stage. For those cannot resolve, e.g., load data will always need to wait until WB stage to bypass, we use interlock unit to detect and interlock the subsequent instruction if needed. The control hazard is resolved by flushing the preceding instruction. We always predict not taken and if branch taken or jump, we flush the instruction by the registers between each stage. There's no structural hazard.

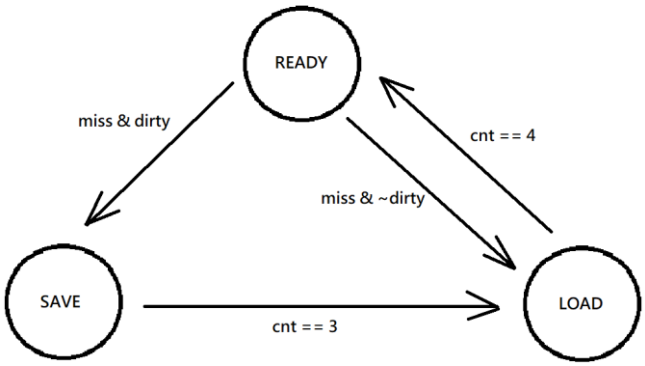
2.3 Cache Design

[*Please insert diagram of your cache structure here. Please include a state transition diagram of your controller.*]

Explain your cache at a high level. Be sure to state which SRAM macros you used, what metadata is used, and how your cache controller works.



The structure of cache is composed of four 256x32 SRAMs (sram22_256x32m4w8). By arranging them in this way, we could store a 512-bit data from main memory to four consecutive addresses in cache.



There are three states in the FSM of cache. First, the default state is READY, waiting for the CPU to request data. When it's a cache hit, the state won't change since it could send the corresponding data and receive the next address immediately. When it's a cache miss, the next state depends on whether the data is dirty. If it's dirty, then we have to perform eviction in SAVE state; on the other hand, if the data is clean, then we could simply load the new data from the main memory, which performs in LOAD state. The counter cnt keeps track of the status in each state, and controls how the state updates.

If you have implemented any optimizations beyond those already mentioned, then please discuss them below:

Instead of using SRAMs to store metadata, we use regfile to store metadata. Since regfile can perform asynchronous read, which could immediately output tag, valid, and dirty bits correspond to the input address. With this method, we could save up one cycle for every cache miss and reduce extra, complicated control logic for cache.

Problem 3: Simulation

3.1 Behavioral

Did you pass all the assembly tests (*asm* tests)? If not, please list the tests you failed and why (if known). Include a screenshot of the passing ASM tests.

Yes, we passed all assembly tests.

```
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/addi.out     () after 297 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/add.out      () after 598 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/andi.out     () after 235 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/and.out      () after 612 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/auipc.out    () after 52 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/beq.out      () after 392 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/bge.out      () after 440 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/bgeu.out     () after 465 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/blt.out      () after 392 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/bltu.out     () after 423 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/bne.out      () after 396 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/jal.out      () after 48 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/jalr.out     () after 146 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/lb.out       () after 302 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/lbu.out      () after 302 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/lh.out       () after 320 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/lhu.out      () after 327 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/lui.out      () after 60 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/lw.out       () after 330 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/ori.out      () after 248 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/or.out       () after 615 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/sb.out       () after 539 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/sh.out       () after 604 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/simple.out   () after 22 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/slli.out     () after 296 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/sll.out      () after 632 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/slti.out     () after 292 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/sltiu.out    () after 292 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/slt.out      () after 586 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/sltu.out     () after 586 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/srai.out     () after 317 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/sra.out      () after 663 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/srli.out     () after 311 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/srl.out      () after 651 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/sub.out      () after 584 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/sw.out       () after 611 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/xori.out     () after 250 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/asm_output/xor.out      () after 614 simulation cycles
```

Did you pass all the benchmark tests (*bmark* tests)? If not, please list the tests you failed and why (if known). Include a screenshot of the passing bmark tests.

Yes, we passed all bmark tests.

```
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/bmark_output/cachetest.out      () after 2925926 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/bmark_output/final.out  () after 5735 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/bmark_output/fib.out    () after 5188 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/bmark_output/sum.out    () after 18173843 simulation cycles
[ PASSED ] /home/tmp/eecs151-ado/asic-project-andrew-owen/skel/bmark_output/replace.out        () after 18135402 simulation cycles
```

List the reported cycles for each test (both assembly and benchmark). Leave the corresponding cell blank if you failed a test.

addi	297	*lb*	302	*slti*	292
add	598	*lbu*	302	*sltiu*	292
andi	235	*lh*	320	*slt*	586
and	612	*lhu*	327	*sltu*	586
auipc	52	*lui*	60	*srai*	317
beq	392	*lw*	330	*sra*	663
bge	440	*ori*	248	*srli*	311
bgeu	465	*or*	615	*srl*	651
blt	392	*sb*	539	*sub*	584
bltu	423	*sh*	604	*sw*	611
bne	396	*simple*	22	*xori*	250
jal	48	*slli*	296	*xor*	614
jalr	146	*sll*	632		

Table 1: Reported cycles for assembly tests

cachetest	2925926
final	5735
fib	5188
sum	18173843
replace	18135402

Table 2: Reported cycles for benchmark tests

3.2 Post-Synthesis DO NOT FILL IN FOR SPRING 2024

Did you pass all the assembly tests (*asm* tests)? If not, please list the tests you failed and why (if known). Include a screenshot of the passing ASM tests.

Did you pass all the benchmark tests (*bmark* tests)? If not, please list the tests you failed and why (if known). Include a screenshot of the passing bmark tests.

List the reported cycles for each test (both assembly and benchmark). Leave the corresponding cell blank if you failed a test.

addi		*lb*		*slti*	

add		*lbu*		*sltiu*	
andi		*lh*		*slt*	
and		*lhu*		*sltu*	
auipc		*lui*		*srai*	
beq		*lw*		*sra*	
bge		*ori*		*srli*	
bgeu		*or*		*srl*	
blt		*sb*		*sub*	
bltu		*sh*		*sw*	
bne		*simple*		*xori*	
jal		*slli*		*xor*	
jalr		*sll*			

Table 3: Reported cycles for assembly tests

cachetest	
final	
fib	
sum	
replace	

Table 4: Reported cycles for benchmark tests

Problem 4: Implementation: Synthesis

4.1 Timing

Does your design pass timing? (Yes/No): Yes

Maximum frequency for which the design passes timing (MHz): T = 24ns; f = 41.6MHz

Slack on Critical Path (ps): 1650ps

Critical Path Location (can list start and end point):

Start point: cpu/DEEXM/R0/q_reg[0]/CLK

End point: cpu/DEEXM/R14/q_reg[11]/D

Explain why the critical path is located where it is given your implementation. If the start and end points of your critical path do not inform the function in your design, then please provide context. This answer should be implementation specific (ex. our forwarding path the ALU, removing a register reduce miss latency).

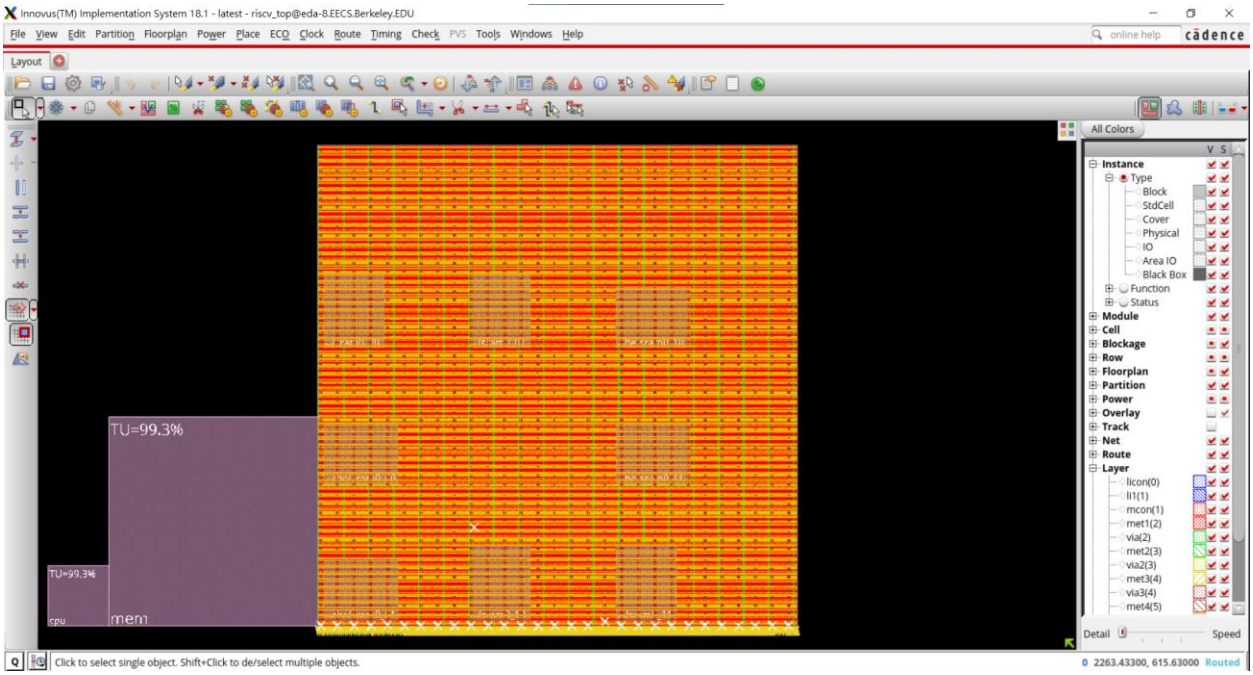
This path starts from the register between DEE and XM stage. The datapath goes through ALU, then becomes the address of dcache. Since we implement asynchronous read for metadata, which the stall signal depends on, the datapath comes back to CPU with stall signal. In the end, stall signal controls the instruction in XM stage, so the critical path ends at the same stage of register.

The reason we keep so much slack in this step is because the wire after PAR will have actual wire delay based on its location and length. SRAM macros are large compared to the synthesized logic of CPU, so the wires between CPU and SRAMs are long and thus cause significant wire delay. Therefore, we have to ensure enough slack for PAR step.

Problem 5: Implementation: Place-and-Route

5.1 Floorplan

[*Please insert a screenshot of your floorplan here*] Explain your floorplanning below.



To balance the length of wire for each data path, we placed SRAMs as a block and let the pins orientation face the middle part.

Therefore, CPU will be automatically placed in the middle part and have similar wire length to each SRAM.

5.2 Timing

Does your design pass timing? (Yes/No): Yes

Maximum frequency for which the design passes timing (MHz): T = 24ns; f = 41.6MHz

Slack on Critical Path (ps): 111ps

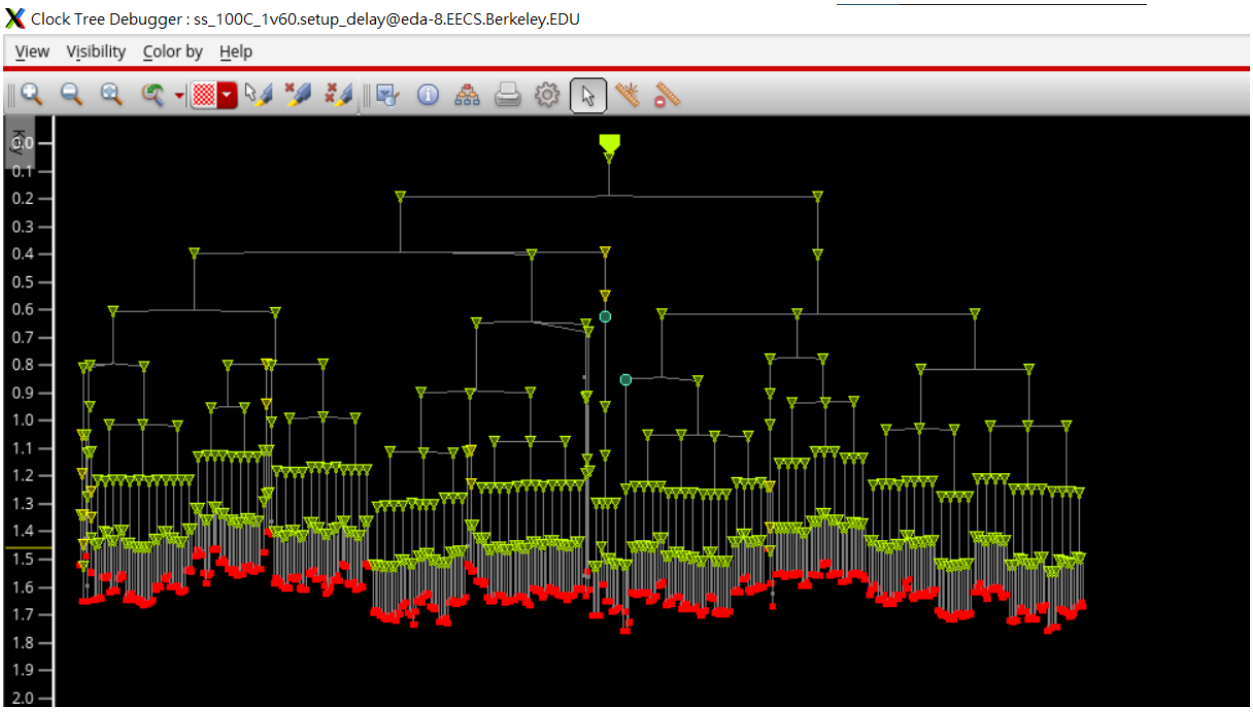
Critical Path Location (can list start and end point):

Start point: cpu/DEEXM/R17/q_reg[1]/CLK

End point: mem/dcache/REG0_43/q_reg[17]/D

[*Please insert a screenshot of your clock tree*]

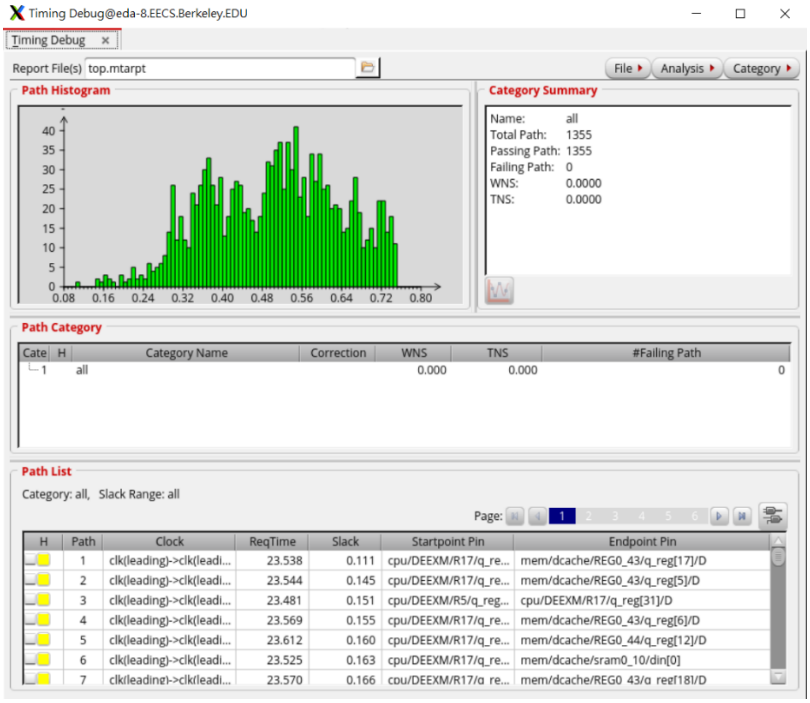
Discuss the quality of your clock tree below.



The clock skew for this clock tree is around 0.35ns.

[*Please insert screenshots of your setup and hold slack critical paths from the Innovus timing reports*]

Explain why this is the critical path given your implementation. Make sure to state whether it is the same as the critical path from synthesis.



The critical path shown here is not the same path from synthesis; however, it's still a path that crosses from CPU to SRAM. The wire delay from CPU to SRAM is large, which acts as bottleneck of the optimization on frequency.

5.3 Area

[*Please insert a screenshot of the output from the report area command in Innovus*]

What module consumes the most area (the area should be seen in the screenshot above). Explain why below.

```
@innovus 10> report_area
Depth  Name                                          #Inst  Area (um^2)
------------------------------------------------------------------------
0      riscv_top                                     22155  1078391.508
1      mem                                           13526  990627.3352
1      cpu                                           8373   84094.4032
2      cpu/WB                                        156    1218.6688
2      cpu/IF                                        233    2257.1648
2      mem/arbiter                                   92     369.104
2      cpu/Control                                   23     140.1344
2      cpu/IFDE                                      184    2197.1072
2      cpu/XM                                        1448   10072.16
2      mem/icache                                    6620   492269.498
2      cpu/MEMWB                                     366    4631.9424
2      mem/dcache                                    6780   497694.7012
2      cpu/DEE                                       5159   55338.0736
2      cpu/SU                                        39     253.9936
2      cpu/DEEXM                                     655    7234.4384
2      cpu/BU                                        72     455.4368
3      mem/icache/REG0_56                            44     578.0544
3      cpu/DEEXM/R7                                  2      26.2752
3      cpu/DEE/A0                                    37     235.2256
3      mem/icache/REG0_33                            44     578.0544
3      mem/dcache/REG0_53                            44     578.0544
3      mem/icache/REG0_25                            44     578.0544
3      mem/icache/REG0_2                             44     578.0544
3      mem/dcache/REG0_22                            44     578.0544
3      mem/icache/REG0_58                            44     578.0544
3      mem/icache/REG0_16                            44     578.0544
```

Memory module consumes a lot of area, for we use regfile for metadata. For CPU part, DEE module consumes most of the CPU area, since most of the arithmetic logic is in the module.

Problem 6: Known Bugs

If you have any known bugs in the core (not including the cache) list them and what you believe is the root cause.

There once was a bug for branch resolve, which will lead to unsuccessful flush of wrong instruction, but we solve it. The CPU functions normally.

If you have any known bugs in the cache list them and what you believe is the root cause.

There once was a bug for loading the data from external memory to cache, caused by the error in control signal counter cnt. However, I've fixed it and all the other appeared bugs. The cache now functions normally.

Problem 7: Optimizations (Optional)

Explain any optimizations you made for either performance or area.

If you would like to be included in the design competition, include your performance-area score for the *sum* benchmark:

$$Score = ClockPeriod * Cycles * Area^{\frac{1}{2}}$$

The units for *ClockPeriod* and *Area* should be *ns* and *μm*. Please refer to Checkpoint 4 for more specifics and an example. List the values for each variable.

ClockPeriod (ns)	24.0
Cycles	18173843
Area (μm)	84094.4032

Score for *sum* benchmark:

Problem 8: Extra Information

Your final grade will be a reflection of your design complexity, functionality, and any other information the TAs have about your submission. If there is anything the TAs should know about while grading your design (not team dynamics or please discuss them below.

We have tried different kinds of cache and different stages of pipeline. The original design is 5-stage pipeline, and we forward the operand we need in the same stage of execution. We find out that forwarding to the same stage will cause the critical path to become extraordinarily long, so we change it to before the registers. We used 4-stage of pipeline eventually because we find out 5-stage pipeline have higher cycles but no significant timing decrease. Because of the longer path due to using register file to store megadata, our path for icache address have designed to be short, only assign from PC.

For cache, we have designed a 8KB 2-way set associative cache to minimize the miss rate for running sum benchmark. But after PAR flow, we found out that the wire delay caused by connecting all SRAMs to CPU is non-

negligible, and greatly affects the maximum frequency. On the other hand, the miss rate for 8KB 2-way set associative cache is only slightly lower than direct-mapped cache. The final FOM of 2-way set associative version is even bigger than that of direct-mapped version (the smaller the better). After some tradeoffs, we decided to implement 4KB direct-mapped cache to lower the quantity of SRAMs, and hence we could design a better placement for SRAMs and CPU to lower the wire delay between them.

Problem 9: Feedback

If you have any feedback on how to improve the project please tell us below. For example, do you feel the labs prepared you to complete the project.

The lab prepares us to complete the project, but does not include a lot of information for optimizing the critical path, especially after PAR. The path after PAR is confusing for us and it's hard for us to optimize from there. The biggest barrier keeping us from increasing the maximum frequency is the wire delay after PAR. I think we could learn more about how to set the parameters for PAR tool to acquire the result we want.