EECS 251A Fall 2020 | Professor Sophia Shao

ASIC Final Project Report

Lisa Qing & Eric Wu

I. RISC-V Processor Design

CPU Design

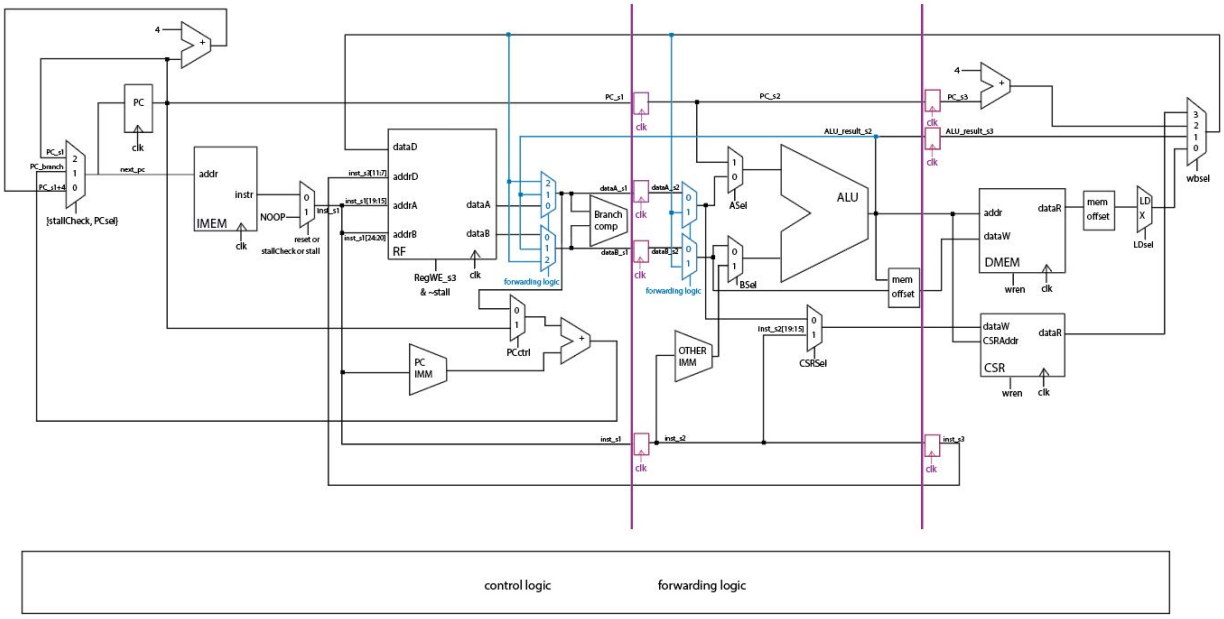


Fig. 1. Final 3-stage pipeline diagram

The pipeline diagram in Fig. 1 illustrates how we divided the CPU into 3 stages (indicated with the purple line and registers). Of the typical 5 stages (IF/D/EX/MEM/WB), we decided to group instruction fetch and register decode (and branch comparator) into one stage, and the memory and write back were grouped into one stage. The other major design choice we made regarding the CPU was placing the branch comparator in stage 1. This enables our design to immediately determine whether the branch is taken or not and designate the correct next PC value. The benefits and detriments of implementing this design are described more in *Section II. Post-Synthesis Analysis* as this decision plays a major role in our CPU's critical path.

Cache Design

At the high level, we divided our cache into two modules: cache controller and cache array. The controller handles the cache's FSM logic, which covers all the events that the cache needs to handle for both hits and misses. The controller module also includes the interaction with the tag

SRAMs to determine which addresses are currently stored in the data cache. The cache array is the module that actually interacts with the data cache by reading and writing to the SRAMs. Since we implemented a configurable cache that can be direct-mapped and N-way set associative, we implemented an LRU replacement policy. The LRU we used the exact LRU approach (mentioned in HW 10), where the age bits come from the combination of each way. For example, for a 4-way set, the bits represent (0<1), (0<2), (0<3), (1<2), (1<3), and (2<3). The number of bits needed is the combination of (NUM_WAYS, 2) ($= \frac{NUM\_WAYS!}{(NUM\_WAYS-2)!\,2!}$), which is equivalent to `NUM_WAYS*(NUM_WAYS-1)>>1`. The open-source RISC processor's LRU was used as reference. For cache and memory accesses, the CPU will stall all stages such that the current instructions in each stage will remain until this stall is no longer asserted. It is also important to note that this kind of stall is different from the instruction stall that is used to prevent data hazards between instructions (discussed for the load to branch hazard we have in *Section II*). Fig. 2 illustrates the top-level cache module.

Fig. 3 depicts the way in which we organize our cache within SRAMs for a 1kB 2-way cache. We chose to use SRAM1RW64x128 for our data storage and SRAM2RW16x32 for our tag storage. Essentially, every 4 indices in the SRAM corresponds to an index in the cache because each cache line is 512 bits. Then to prevent reading potential garbage data in the tag SRAM, we use 2D arrays for valid bits and dirty bits and store them inside flip flops. As shown at the bottom of the diagram, the rows indicate the cache index and the column indicates the way.

Fig. 4 shows the cache FSM in detail. The cache FSM has 6 total states: {READ_TAG, ACCESS, REFILL, READ, WRITE, WRITE_ADDR}. We have a 2-cycle hit time as we need to read from the tag SRAMs and then read data from the data SRAMs. We have an 8-cycle penalty for misses because we need to first identify the miss, refill the cache (4 cycles for 512 bits), read, and potentially write to the cache. Lastly, there is a 16-cycle penalty for evictions as we need to write to memory (if there is a dirty bit) and refill the cache as well.

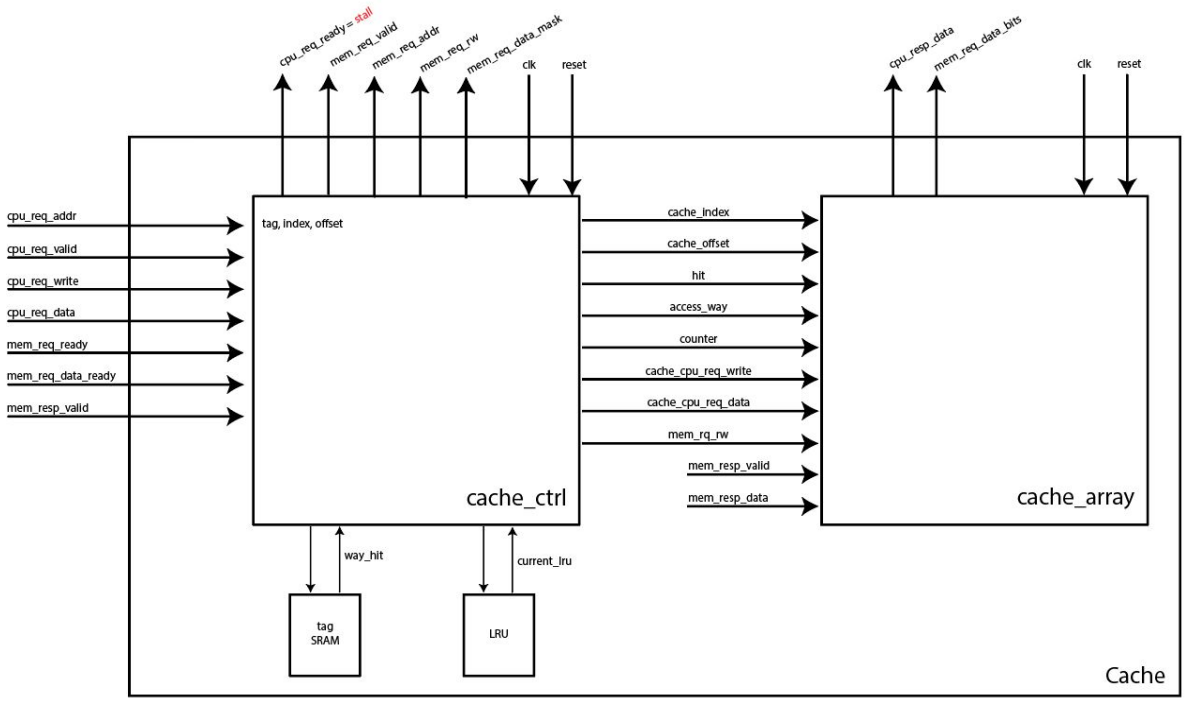


Fig. 2. Top-level cache block diagram

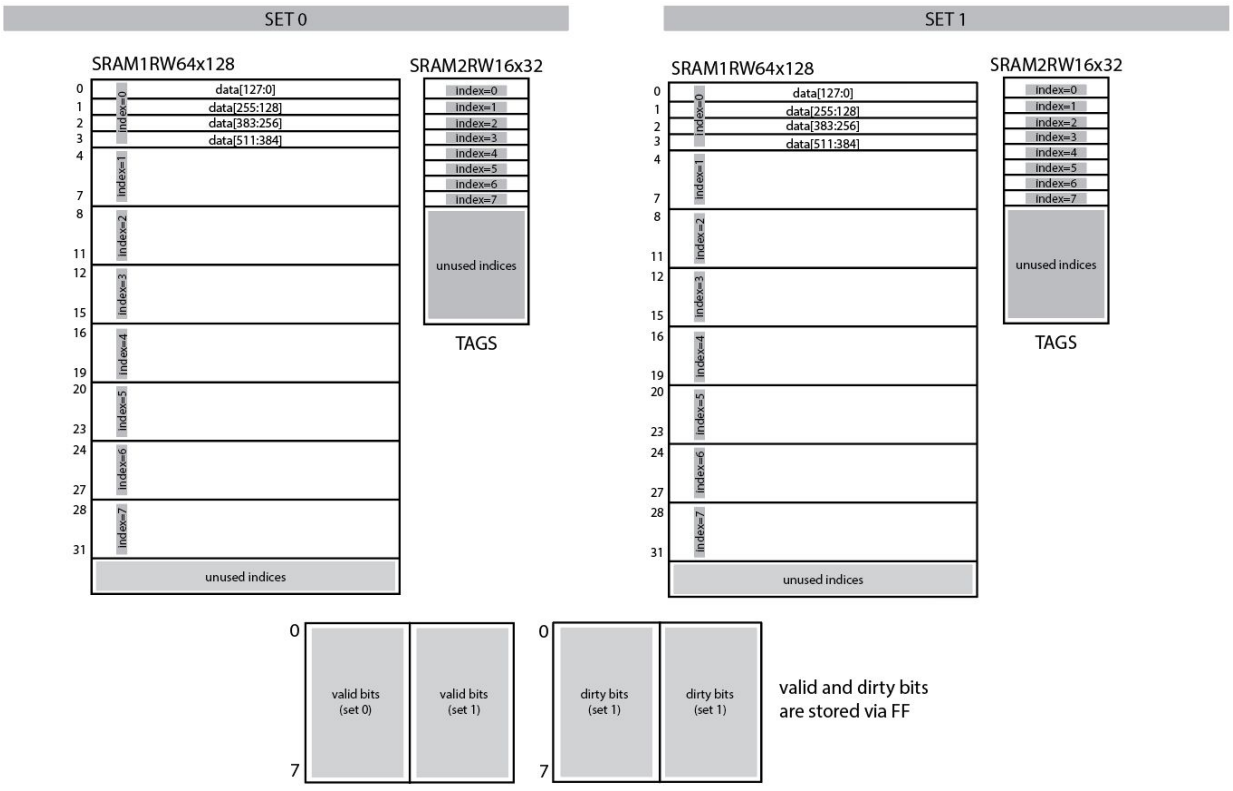


Fig. 3. Example 1kB cache design for 2-way set associative

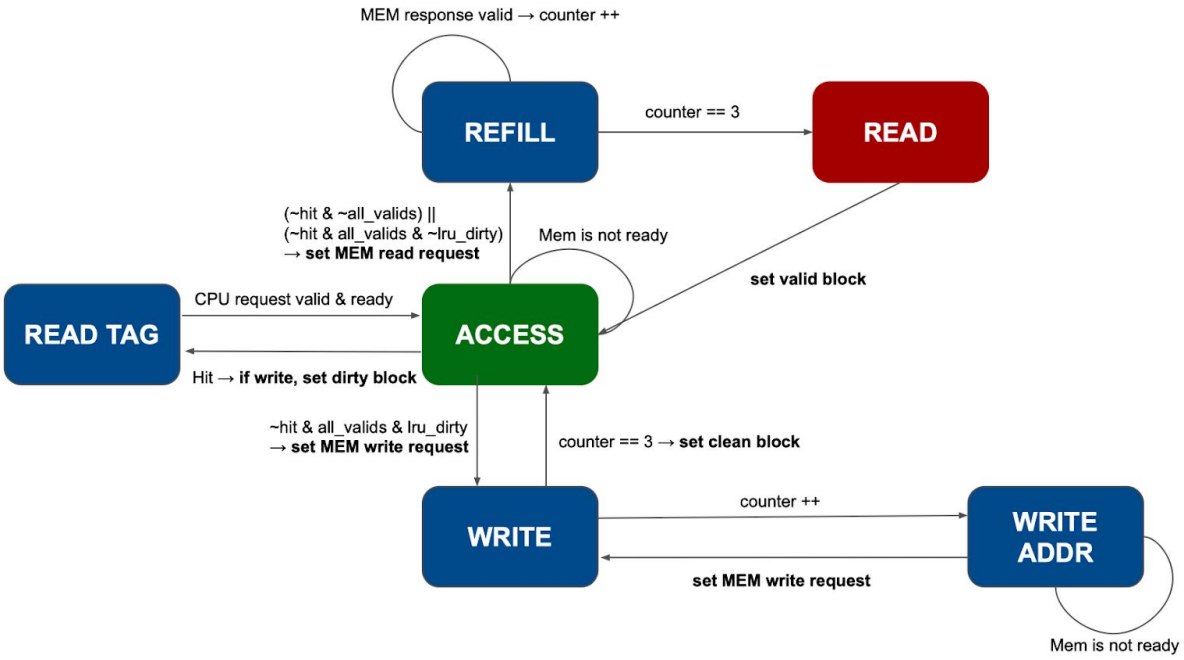


Fig. 4. Cache FSM in detail

II. Post-Synthesis Analysis

One of the major design choices was to place the branch comparator in stage 1 with the decoding process. The purpose of this was to reduce control hazards and potential extra cycles resulting from branch misprediction. However, this design choice came at the cost of a longer critical path, extra forwarding muxes to the branch comparator, and an unavoidable instruction stall for load to branch instructions where we must insert a noop in-between these two instructions (because load will not receive its value until stage 3 but branch occurs in stage 1). Fig. 5 highlights the path on the pipeline diagram and Fig. 6 shows the critical path that was found in the post-synthesis report. The critical path is the worst-case forwarding scenario in our pipeline design. The signal gets forwarded from mem (stage 3) to execution (stage 2) then to branch comparator (stage 1) to determine the next PC. Below is a set of example assembly instructions that would cause this behavior:

```
lw x15 0(x0)
addi x15 4
beq x15 x14 LABEL
```

In other words, our assumption that branching would cause a bottleneck in our CPU frequency was confirmed in these reports. A total of 63 standard cells are in this path.

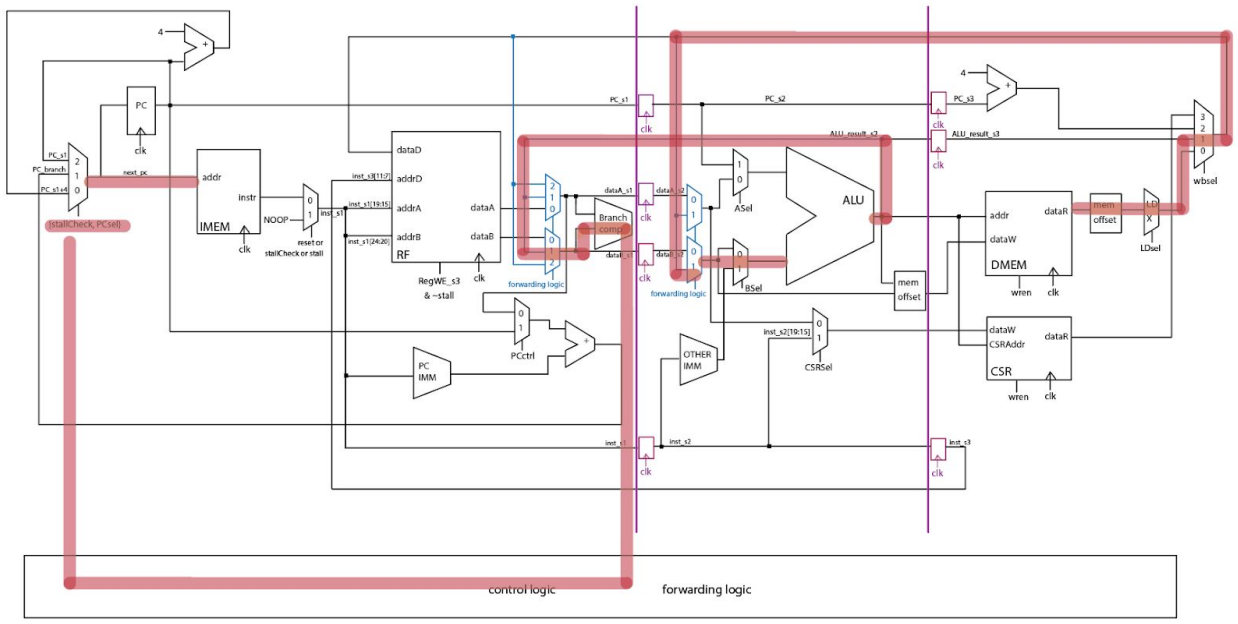


Fig. 5. Critical path highlighted in pipeline diagram

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

                  Capture       Launch
     Clock Edge:+     890            0
     Src Latency:+      0            0
     Net Latency:+      0 (I)        0 (I)
         Arrival:=    890            0

           Setup:-      3
     Uncertainty:-    100
    Required Time:=   787
     Launch Clock:-     0
        Data Path:-    787
            Slack:=      0
```

Timing Point	Flags	Arc	Edge	Cell	Fanout	Load (fF)	Trans (ps)	Delay (ps)	Arrival (ps)	Instance Location
mem/dcache/c_array/array_offset_reg[1]/CLK	-	-	R	(arrival)	195	-	0	-	0	(-,-)
mem/dcache/c_array/array_offset_reg[1]/Q	-	CLK->Q	R	DFFHQx4_ASAP7_75t_SL	6	5.2	17	46	46	(-,-)
mem/dcache/c_array/g14200/Y	-	A->Y	F	INVx5_ASAP7_75t_SL	5	3.2	10	6	52	(-,-)
mem/dcache/c_array/g14165/Y	-	B->Y	F	AND3x2_ASAP7_75t_SL	3	2.2	11	14	66	(-,-)
mem/dcache/c_array/g14160/Y	-	A->Y	F	BUFx6f_ASAP7_75t_SL	5	4.9	10	14	80	(-,-)
mem/dcache/c_array/g14158/Y	-	A->Y	F	BUFx12f_ASAP7_75t_SL	4	4.0	7	13	93	(-,-)
mem/dcache/c_array/g14157/Y	-	A->Y	F	BUFx6f_ASAP7_75t_SL	6	6.5	11	14	107	(-,-)
mem/dcache/c_array/g14033/Y	-	B2->Y	R	AOI22x1_ASAP7_75t_SL	1	1.3	20	12	120	(-,-)
mem/dcache/c_array/g13975/Y	-	A->Y	F	NAND4xp75_ASAP7_75t_SL	2	1.2	24	10	130	(-,-)
cpu/stage3/g1281/Y	-	A->Y	F	BUFx2_ASAP7_75t_SL	2	1.8	11	18	148	(-,-)
cpu/stage3/g1242/Y	-	B->Y	R	NAND2x1p5_ASAP7_75t_SL	1	1.3	17	8	156	(-,-)
cpu/stage3/g1205/Y	-	A->Y	F	NAND2x1p5_ASAP7_75t_SL	1	1.3	18	6	161	(-,-)
cpu/stage3/ldx/g881/Y	-	B->Y	R	NAND2x1p5_ASAP7_75t_SL	1	0.7	15	8	169	(-,-)
cpu/stage3/ldx/g866/Y	-	B->Y	F	NAND2xp5_ASAP7_75t_SL	1	0.7	14	8	178	(-,-)
cpu/stage3/muxwb/g2209/Y	-	A2->Y	F	AO21x2_ASAP7_75t_SL	3	1.6	14	18	196	(-,-)
cpu/stage2/Aforwarding/g601/Y	-	A2->Y	F	AO21x2_ASAP7_75t_SL	2	1.8	12	19	214	(-,-)
cpu/stage2/ALUAsel/g623/Y	-	A1->Y	R	AOI21x1_ASAP7_75t_SL	1	1.5	29	11	226	(-,-)
cpu/stage2/ALUAsel/g611/Y	-	A->Y	F	INVx3_ASAP7_75t_SL	6	6.0	21	13	239	(-,-)
cpu/stage2/alu/g2778/Y	-	A->Y	F	BUFx6f_ASAP7_75t_SL	8	6.0	11	17	256	(-,-)
cpu/stage2/alu/sll_29_31_g2913/Y	-	B2->Y	R	AOI22x1_ASAP7_75t_SL	2	1.4	24	13	269	(-,-)
cpu/stage2/alu/sll_29_31_g2828/Y	-	A2->Y	R	OA22x2_ASAP7_75t_SL	2	2.0	15	21	289	(-,-)
cpu/stage2/alu/sll_29_31_g2790/Y	-	A1->Y	F	OAI22x1_ASAP7_75t_SL	2	1.3	18	10	300	(-,-)
cpu/stage2/alu/sll_29_31_g2741/Y	-	A1->Y	R	AOI21xp5_ASAP7_75t_SL	1	0.7	24	12	311	(-,-)
cpu/stage2/alu/sll_29_31_g2720/Y	-	B->Y	F	OAI21xp5_ASAP7_75t_SL	1	0.7	35	10	321	(-,-)
cpu/stage2/alu/g8015/Y	-	A2->Y	F	AO21x1_ASAP7_75t_SL	1	0.7	12	18	339	(-,-)
cpu/stage2/alu/g7980/Y	-	B->Y	R	AOI21xp5_ASAP7_75t_SL	1	0.7	29	12	351	(-,-)
cpu/stage2/alu/g7958/Y	-	B->Y	F	NAND2xp5_ASAP7_75t_SL	1	1.1	22	12	363	(-,-)
cpu/stage2/alu/g7925/Y	-	B->Y	R	AOI21x1_ASAP7_75t_SL	1	1.3	41	13	376	(-,-)
cpu/stage2/alu/g7913/Y	-	B->Y	F	NAND2x1p5_ASAP7_75t_SL	3	1.6	20	8	385	(-,-)
cpu/stage1/muxB/g1821__7410/Y	-	A1->Y	F	AO21x2_ASAP7_75t_SL	5	3.1	17	24	408	(-,-)
cpu/stage1/bc/g2934/Y	-	A->Y	F	BUFx2_ASAP7_75t_SL	2	1.9	11	16	425	(-,-)
cpu/stage1/bc/lt_38_23_g804__5526/Y	-	A->Y	R	NAND2x1_ASAP7_75t_SL	2	1.3	18	11	436	(-,-)
cpu/stage1/bc/lt_38_23_g761__9315/Y	-	A1->Y	F	AOI21xp5_ASAP7_75t_SL	1	1.1	21	12	448	(-,-)
cpu/stage1/bc/lt_38_23_g740__5477/Y	-	A2->Y	R	OAI21x1_ASAP7_75t_SL	1	1.1	20	12	460	(-,-)
cpu/stage1/bc/lt_38_23_g730__7482/Y	-	B->Y	F	AOI21x1_ASAP7_75t_SL	1	1.2	17	9	470	(-,-)
cpu/stage1/bc/lt_38_23_g726__7098/Y	-	A1->Y	R	OAI21x1_ASAP7_75t_SL	1	1.1	20	12	482	(-,-)
cpu/stage1/bc/lt_38_23_g725__8246/Y	-	B->Y	F	AOI21x1_ASAP7_75t_SL	1	1.3	16	9	491	(-,-)
cpu/stage1/bc/g4513__5115/Y	-	B->Y	R	NOR2x1p5_ASAP7_75t_SL	2	1.5	15	9	500	(-,-)
cpu/stage1/bc/g4441__7410/Y	-	B2->Y	F	AOI221x1_ASAP7_75t_SL	1	1.2	35	15	514	(-,-)
cpu/stage1/bc/g4440__1666/Y	-	B->Y	R	OAI21x1_ASAP7_75t_SL	2	2.6	36	14	529	(-,-)
cpu/stage1/muxPC/fopt1849/Y	-	A->Y	F	INVx3_ASAP7_75t_SL	2	2.2	15	8	537	(-,-)
cpu/stage1/muxPC/fopt1848/Y	-	A->Y	R	INVx2_ASAP7_75t_SL	3	2.4	15	10	546	(-,-)
cpu/stage1/muxPC/g1835__6161/Y	-	B->Y	F	NOR2x1p5_ASAP7_75t_SL	2	1.7	48	8	555	(-,-)
cpu/stage1/muxPC/g2__9315/Y	-	A2->Y	R	AOI21x1_ASAP7_75t_SL	1	1.3	29	16	571	(-,-)
cpu/stage1/muxPC/g1799__6417/Y	-	B->Y	F	NAND2x1p5_ASAP7_75t_SL	3	1.8	17	8	579	(-,-)
mem/icache/c_ctrl/g11317/Y	-	A2->Y	F	AO21x2_ASAP7_75t_SL	3	2.2	15	20	599	(-,-)
mem/icache/c_ctrl/g11295/Y	-	A->Y	R	INVx2_ASAP7_75t_SL	3	2.4	15	10	608	(-,-)
mem/icache/c_ctrl/g11268/Y	-	B->Y	F	NAND2x1_ASAP7_75t_SL	1	1.1	31	7	615	(-,-)
mem/icache/c_ctrl/g11265/Y	-	B->Y	R	AOI21x1_ASAP7_75t_SL	1	1.2	23	14	630	(-,-)
mem/icache/c_ctrl/g11261/Y	-	B->Y	F	OAI21x1_ASAP7_75t_SL	2	1.7	19	12	641	(-,-)
mem/icache/c_ctrl/g11256/Y	-	B->Y	R	NAND2x1p5_ASAP7_75t_SL	2	1.7	18	11	652	(-,-)
mem/icache/c_ctrl/g11253/Y	-	A2->Y	F	OAI21x1_ASAP7_75t_SL	1	1.3	18	9	661	(-,-)
mem/icache/c_ctrl/g11251/Y	-	B->Y	R	NAND2x1p5_ASAP7_75t_SL	1	1.1	14	9	670	(-,-)
mem/icache/c_ctrl/g8950/Y	-	A->Y	F	INVx2_ASAP7_75t_SL	4	2.7	13	8	678	(-,-)
mem/arbiter/g269/Y	-	B->Y	R	NOR2x1p5_ASAP7_75t_SL	3	2.6	21	12	690	(-,-)
mem/dcache/c_ctrl/fopt12538/Y	-	A->Y	F	INVx3_ASAP7_75t_SL	3	1.7	10	6	696	(-,-)
mem/dcache/c_ctrl/g12259/Y	-	B2->Y	R	AOI22xp5_ASAP7_75t_SL	1	1.1	36	15	711	(-,-)
mem/dcache/c_ctrl/g12231/Y	-	A2->Y	F	OAI21x1_ASAP7_75t_SL	2	1.2	23	9	720	(-,-)
mem/dcache/c_ctrl/g12219/Y	-	A2->Y	F	AO21x2_ASAP7_75t_SL	2	3.2	17	23	744	(-,-)
mem/dcache/c_ctrl/g8305/Y	-	A->Y	R	INVx6_ASAP7_75t_SL	8	9.8	20	12	756	(-,-)
mem/dcache/c_ctrl/g8202/Y	-	B->Y	F	NOR2x1p5_ASAP7_75t_SL	3	1.8	19	9	765	(-,-)
mem/dcache/c_ctrl/g8031/Y	-	A->Y	R	NAND2xp5_ASAP7_75t_SL	1	0.7	17	12	777	(-,-)
mem/dcache/c_ctrl/g8014/Y	-	B->Y	F	O2A1O1Ixp33_ASAP7_75t_SL	1	0.7	18	10	787	(-,-)
mem/dcache/c_ctrl/tag_dirty_reg[1][0]/D	-	-	F	DFFHQx4_ASAP7_75t_SL	1	-	-	0	787	(-,-)

Fig. 6. Post-synthesis critical path report at max frequency (clock period = 0.89 ns)

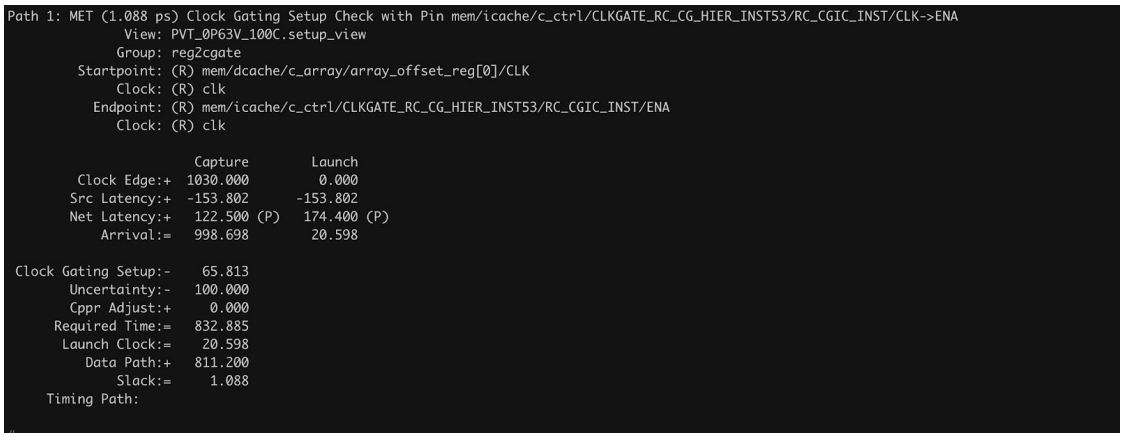
III. Post-Route Analysis

Timing Analysis

We placed-and-routed our design at the max frequency (clock period = 1.03 ns). The cycle time is slightly longer than the synthesized design, which is an expected result. One of the major reasons is that the innovus tool had to take into account the geometry of the SRAM macros to not interfere with the standard cells. This worsened the routability of the design, resulting in higher cycle time. Fig. 7 shows the post-route critical path. Note that it passes through a similar path to the post-synthesis design which is shown in Fig. 5 and Fig. 6, where it traverses each stage through all forwarding muxes to find the next PC value. A total of 54 standard cells and 55 nets are in this path. The number of standard cells is different from the post-synthesis since the path's ending point is different from the post-synthesis. The post synthesis critical path goes through the arbiter to the dcache control at the end of the path; however, in post-route, the path ends at the icache's clock gate. This is most likely due to the optimization by tool in PAR.

Floorplan

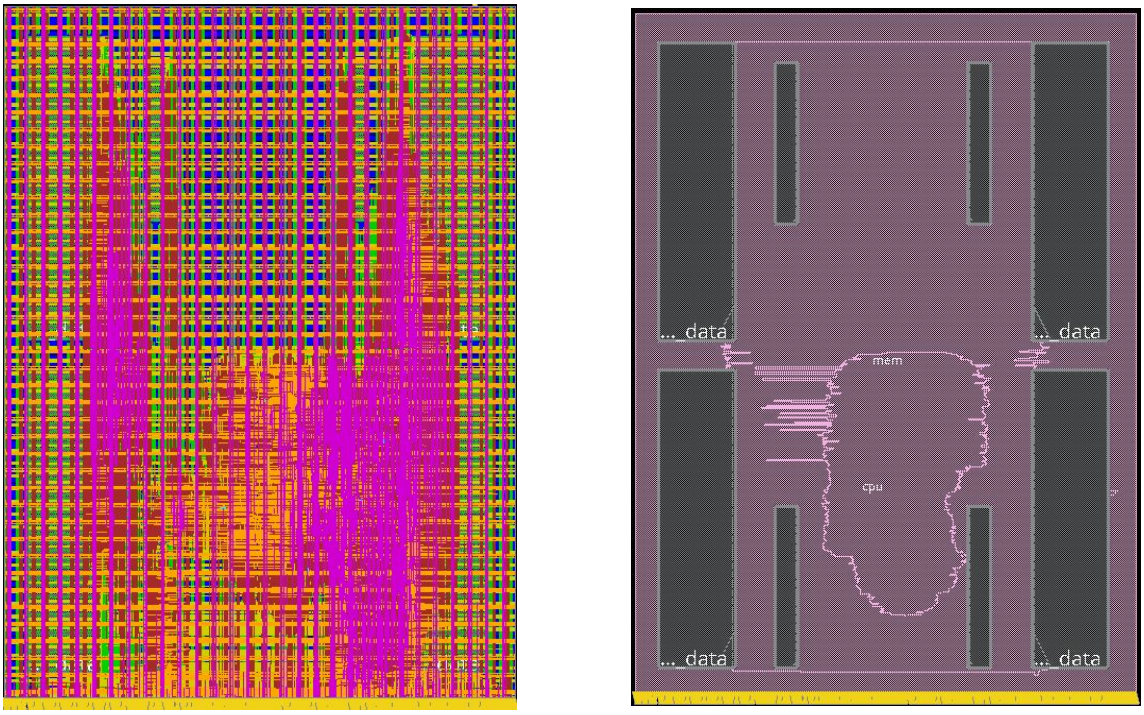
Fig. 8 shows the floorplan of our design, a 32-bit RISC-V CPU with 1kB 2-way set associative instruction/data cache. The dimension is 430x580 μm^2 with the design density 11.4%. Our strategy to place the SRAM macros is to put the data SRAM at each edge of the design (left and right) and the tag SRAM is enclosed inside the data SRAM. This helped the CPU access the tag and data more easily compared to previous designs where we put the tag SRAM at the top of the data SRAM. Also, we left some spacing between each SRAM macros and the edge of the canvas. This gave the tool more flexibility to do routing around the SRAM macros and eliminate unnecessary DRC errors. With this strategy, the critical path improved from 1.1 ns to 1.03 ns, and the design density increased from 6.9% to 11.4%.



Timing Point	Flags	Arc	Edge	Cell	Fanout	Trans (ps)	Delay (ps)	Arrival (ps)
clk	-	clk	R	(arrival)	2	4.100	16.200	-137.602
clk	-	-	R	(net)	2	-	-	-
CTS_ccl_a_buf_00128/Y	-	A->Y	R	BUFx6f_ASAP7_75t_SRAM	3	38.200	36.500	-101.102
CTS_7	-	-	R	(net)	3	-	-	-
CTS_ccl_a_buf_00125/Y	-	A->Y	R	BUFx6f_ASAP7_75t_SRAM	4	30.700	34.700	-66.402
CTS_6	-	-	R	(net)	4	-	-	-
CTS_ccl_a_buf_00122/Y	-	A->Y	R	BUFx12f_ASAP7_75t_SRAM	6	23.100	30.100	-36.302
CTS_1	-	-	R	(net)	6	-	-	-
CTS_ccl_a_buf_00099/Y	-	A->Y	R	BUFx10_ASAP7_75t_SRAM	53	23.300	56.900	20.598
CTS_3	-	-	R	(net)	53	-	-	-
mem/dcache/c_array/array_offset_reg[0]/Q	-	CLK->Q	F	DFFHQx4_ASAP7_75t_SL	4	64.800	53.300	73.898
mem/dcache/c_array/array_offset[0]	-	-	F	(net)	4	-	-	-
mem/dcache/c_array/FE_OFC1870_array_offset_0/Y	-	A->Y	R	INVx5_ASAP7_75t_SL	2	16.900	6.600	80.498
mem/dcache/c_array/FE_DBTN21_array_offset_0	-	-	R	(net)	2	-	-	-
mem/dcache/c_array/g12378/Y	-	B->Y	F	NAND2x1p5_ASAP7_75t_SL	2	9.500	6.900	87.398
mem/dcache/c_array/n_124	-	-	F	(net)	2	-	-	-
mem/dcache/c_array/g12361/Y	-	A->Y	F	OR2x6_ASAP7_75t_SL	6	11.200	24.700	112.098
mem/dcache/c_array/n_140	-	-	F	(net)	6	-	-	-
mem/dcache/c_array/FE_OFC3357_n_140/Y	-	A->Y	R	INVx8_ASAP7_75t_SL	15	19.900	15.000	127.098
mem/dcache/c_array/FE_DBTN20_n_140	-	-	R	(net)	15	-	-	-
mem/dcache/c_array/g12233/Y	-	A2->Y	R	AO22x2_ASAP7_75t_SL	1	25.100	19.200	146.298
mem/dcache/c_array/n_261	-	-	R	(net)	1	-	-	-
mem/dcache/c_array/g12212/Y	-	A->Y	F	NOR2x1p5_ASAP7_75t_SL	1	13.200	9.200	155.498
mem/dcache/c_array/n_282	-	-	F	(net)	1	-	-	-
mem/dcache/c_array/g12174/Y	-	B->Y	R	NAND2x1p5_ASAP7_75t_SL	3	13.200	20.500	175.998
dcache_dout[26]	-	-	R	(net)	3	-	-	-
cpu/stage3/g1376/Y	-	B2->Y	F	AOI22x1_ASAP7_75t_SL	2	39.600	15.500	191.498
cpu/stage3/n_18	-	-	F	(net)	2	-	-	-
cpu/stage3/g1335/Y	-	A2->Y	R	OAI21x1_ASAP7_75t_SL	1	28.000	21.400	212.898
cpu/stage3/masked_mem_data[2]	-	-	R	(net)	1	-	-	-
cpu/stage3/muxwb/g1471/Y	-	A2->Y	R	AO21x1_ASAP7_75t_SL	3	38.900	19.200	232.098
cpu/dataD_s3[2]	-	-	R	(net)	3	-	-	-
cpu/stage2/Bforwarding/g586/Y	-	B2->Y	R	AO22x2_ASAP7_75t_SL	2	20.500	18.000	250.098
cpu/DMEM_dataW_s2[2]	-	-	R	(net)	2	-	-	-
cpu/stage2/ALUBsel/g589/Y	-	A2->Y	R	AO22x2_ASAP7_75t_SL	2	13.000	23.900	273.998
cpu/stage2/B[2]	-	-	R	(net)	2	-	-	-
cpu/stage2/alu/FE_OFC3053_B_2/Y	-	A->Y	F	INVx3_ASAP7_75t_SL	6	24.600	10.800	284.798
cpu/stage2/alu/FE_OFN2426_B_2	-	-	F	(net)	6	-	-	-
cpu/stage2/alu/FE_OFC3415_B_2/Y	-	A->Y	F	BUFx2_ASAP7_75t_SL	5	16.800	19.300	304.098
cpu/stage2/alu/FE_OFN2599_n	-	-	F	(net)	5	-	-	-
cpu/stage2/alu/FE_OFC3059_B_2/Y	-	A->Y	R	INVx2_ASAP7_75t_SL	13	15.300	27.500	331.598
cpu/stage2/alu/FE_OFN2463_B_2	-	-	R	(net)	13	-	-	-
cpu/stage2/alu/FE_OFC2881_B_2/Y	-	A->Y	R	BUFx2_ASAP7_75t_SL	13	51.400	34.300	365.898
cpu/stage2/alu/FE_OFN2402_n	-	-	R	(net)	13	-	-	-
cpu/stage2/alu/srl_31_31_g2843/Y	-	B->Y	F	NOR2xp33_ASAP7_75t_SL	3	45.800	24.900	390.798
cpu/stage2/alu/srl_31_31_n_9	-	-	F	(net)	3	-	-	-
cpu/stage2/alu/srl_31_31_g2765/Y	-	A->Y	R	NAND2xp5_ASAP7_75t_SL	2	44.500	20.700	411.498
cpu/stage2/alu/srl_31_31_n_87	-	-	R	(net)	2	-	-	-
cpu/stage2/alu/srl_31_31_g2735/Y	-	B->Y	F	NOR2xp33_ASAP7_75t_SL	1	31.200	17.800	429.298
cpu/stage2/alu/n_348	-	-	F	(net)	1	-	-	-
cpu/stage2/alu/g4535/Y	-	B1->Y	F	AO222x2_ASAP7_75t_SL	1	29.300	19.600	448.898
cpu/stage2/alu/n_156	-	-	F	(net)	1	-	-	-
cpu/stage2/alu/g4500/Y	-	C->Y	F	AOI221x1_ASAP7_75t_SL	1	12.600	23.500	472.398
cpu/stage2/alu/n_190	-	-	F	(net)	1	-	-	-
cpu/stage2/alu/g4469/Y	-	B->Y	R	AOI21x1_ASAP7_75t_SL	1	18.100	12.100	484.498
cpu/stage2/alu/n_212	-	-	R	(net)	1	-	-	-
cpu/stage2/alu/g4451/Y	-	B->Y	F	NAND2x1_ASAP7_75t_SL	4	50.800	22.000	506.498
dcache_addr[28]	-	-	F	(net)	4	-	-	-
cpu/stage1/muxA/g1319__9315/Y	-	A1->Y	F	AO21x2_ASAP7_75t_SL	6	40.500	29.200	535.698
cpu/stage1/FE_OFN2068_dataA_s1_28	-	-	F	(net)	6	-	-	-
cpu/stage1/bc/lt_38_23_g784__5477/Y	-	A->Y	R	NAND2xp5_ASAP7_75t_SL	1	19.400	11.000	546.698
cpu/stage1/bc/lt_38_23_n_43	-	-	R	(net)	1	-	-	-
cpu/stage1/bc/lt_38_23_g732__2802/Y	-	A1->Y	F	OAI22xp5_ASAP7_75t_SL	1	16.200	12.800	559.498
cpu/stage1/bc/lt_38_23_n_99	-	-	F	(net)	1	-	-	-
cpu/stage1/bc/lt_38_23_g719__1666/Y	-	A2->Y	R	AOI21x1_ASAP7_75t_SL	1	23.200	11.600	571.098
cpu/stage1/bc/lt_38_23_n_115	-	-	R	(net)	1	-	-	-
cpu/stage1/bc/lt_38_23_g715__9315/Y	-	B->Y	F	OAI21x1_ASAP7_75t_SL	1	19.100	9.000	580.098
cpu/stage1/bc/lt_38_23_n_119	-	-	F	(net)	1	-	-	-
cpu/stage1/bc/lt_38_23_g712__7482/Y	-	B->Y	R	AOI21xp5_ASAP7_75t_SL	1	13.900	12.600	592.698

cpu/stage1/bc/lt_38_23_n_122	-	-	R	(net)	1	-	-	-
cpu/stage1/bc/lt_38_23_g711__5115/Y	-	B->Y	R	OA21x2_ASAP7_75t_SL	1	21.800	18.700	611.398
cpu/stage1/bc/n_65	-	-	R	(net)	1	-	-	-
cpu/stage1/bc/FE_RC_7_0/Y	-	A2->Y	F	AOI22x1_ASAP7_75t_SL	2	14.800	12.600	623.998
cpu/stage1/bc/n_40	-	-	F	(net)	2	-	-	-
cpu/stage1/bc/FE_OFC2491_n_40/Y	-	A->Y	R	INVx2_ASAP7_75t_SL	1	27.000	7.300	631.298
cpu/stage1/bc/n_41	-	-	R	(net)	1	-	-	-
cpu/stage1/bc/g2106__9945/Y	-	A2->Y	F	O2A1O1Ixp5_ASAP7_75t_SL	1	11.500	11.100	642.398
cpu/stage1/bc/n_59	-	-	F	(net)	1	-	-	-
cpu/stage1/bc/FE_RC_8_0/Y	-	A2->Y	R	OAI21x1_ASAP7_75t_SL	1	25.800	12.200	654.598
cpu/stage1/bc/n_60	-	-	R	(net)	1	-	-	-
cpu/stage1/bc/g2104__6161/Y	-	A->Y	F	NAND2x1p5_ASAP7_75t_SL	2	22.400	8.700	663.298
cpu/stage1/PCsel_real	-	-	F	(net)	2	-	-	-
cpu/stage1/muxPC/g1079__4319/Y	-	B->Y	F	AND2x4_ASAP7_75t_SL	4	22.300	16.100	679.398
cpu/stage1/muxPC/n_6	-	-	F	(net)	4	-	-	-
cpu/stage1/muxPC/g1066__7410/Y	-	A1->Y	R	AOI22x1_ASAP7_75t_SL	1	8.800	14.100	693.498
cpu/stage1/muxPC/n_15	-	-	R	(net)	1	-	-	-
cpu/stage1/muxPC/g1056__1881/Y	-	B->Y	F	NAND2x2_ASAP7_75t_SL	3	50.300	8.200	701.698
icache_addr[8]	-	-	F	(net)	3	-	-	-
mem/icache/c_ctrl/FE_RC_31_0/Y	-	A2->Y	R	OAI21x1_ASAP7_75t_SL	5	21.800	22.800	724.498
mem/icache/c_ctrl/n_136	-	-	R	(net)	5	-	-	-
mem/icache/c_ctrl/FE_OFC1587_n_136/Y	-	A->Y	F	INVx3_ASAP7_75t_SL	7	44.100	8.900	733.398
mem/icache/c_ctrl/FE_OFN233_cache_index_2	-	-	F	(net)	7	-	-	-
mem/icache/c_ctrl/g8413/Y	-	A2->Y	R	OAI22xp5_ASAP7_75t_SL	1	18.500	12.300	745.698
mem/icache/c_ctrl/n_239	-	-	R	(net)	1	-	-	-
mem/icache/c_ctrl/FE_RC_20_0/Y	-	B2->Y	F	AOI22xp5_ASAP7_75t_SL	1	28.000	11.900	757.598
mem/icache/c_ctrl/n_250	-	-	F	(net)	1	-	-	-
mem/icache/c_ctrl/g8399/Y	-	B->Y	R	NOR2x1_ASAP7_75t_SL	1	22.700	9.900	767.498
mem/icache/c_ctrl/n_253	-	-	R	(net)	1	-	-	-
mem/icache/c_ctrl/g8395/Y	-	A1->Y	F	OAI21x1_ASAP7_75t_SL	2	17.300	8.800	776.298
mem/icache/c_ctrl/n_257	-	-	F	(net)	2	-	-	-
mem/icache/c_ctrl/g8392/Y	-	A2->Y	R	AOI21x1_ASAP7_75t_SL	2	39.800	14.300	790.598
mem/icache/c_ctrl/n_258	-	-	R	(net)	2	-	-	-
mem/icache/c_ctrl/g8390/Y	-	A2->Y	R	AO21x2_ASAP7_75t_SL	2	25.200	22.300	812.898
mem/icache/c_ctrl/FE_OFN2618_ic_mem_req_valid	-	-	R	(net)	2	-	-	-
mem/icache/c_ctrl/g7001/Y	-	A2->Y	F	OAI21x1_ASAP7_75t_SL	2	20.700	7.500	820.398
mem/icache/c_ctrl/n_288	-	-	F	(net)	2	-	-	-
mem/icache/c_ctrl/g6982/Y	-	B->Y	R	NAND2xp5_ASAP7_75t_SL	1	36.900	11.400	831.798
mem/icache/c_ctrl/n_289	-	-	R	(net)	1	-	-	-
mem/icache/c_ctrl/CLKGATE_RC_CG_HIER_INST53/RC_CGIC_INST/ENA	-	ENA	R	ICGx1_ASAP7_75t_SRAM	1	29.300	0.100	831.798

Fig. 7. Post-route critical path report at max frequency (clock period = 1.03 ns)



(a) (b)

Fig. 8. (a) Final 2-way floorplan (b) Final 2-way floor plan in amoeba view

IV. Optimization

Optimizing for number of cycles

We identified three methods to improve the cycle count for benchmark execution. The first technique is using 2RW tag SRAM instead of 1RW tag SRAM. This reduced the cycles that the FSM needs to stay in the ACCESS state since while the tag SRAM is updated, we could read the up-to-date result from another port. The second technique is using the byte mask data SRAM to only write to specific, designated bits in the SRAM. Recall that in the cache refilling process (Fig. 4), the FSM went to the READ state after all 512 bits were brought up. The READ state was necessary because without it, we would be reading the previous data or garbage. With the byte mask data SRAM, this additional state can be eliminated, reducing another 1 cycle for the cache misses. The last technique is to configure the cache differently. We increased the set associativity and the cache size to reduce the miss rate. However, this method comes with the trade off of some frequency. After using all three techniques, we yielded a 14.1% improvement in cycle count between the baseline cache, 1kB direct-mapped cache with 1RW tag SRAM and data SRAM without byte mask, and the best configuration, 4kB 4-way cache with 2RW tag SRAM and data SRAM with byte mask.

The result with different cache configuration is shown in Fig. 9. We observed that just with the first two techniques, the cache is improved by 5.45% according to the first bar. The best result is the 4kB, 4-way cache. The amount of change between different N-way set associative caches are less drastic than with the DM cycle counts.

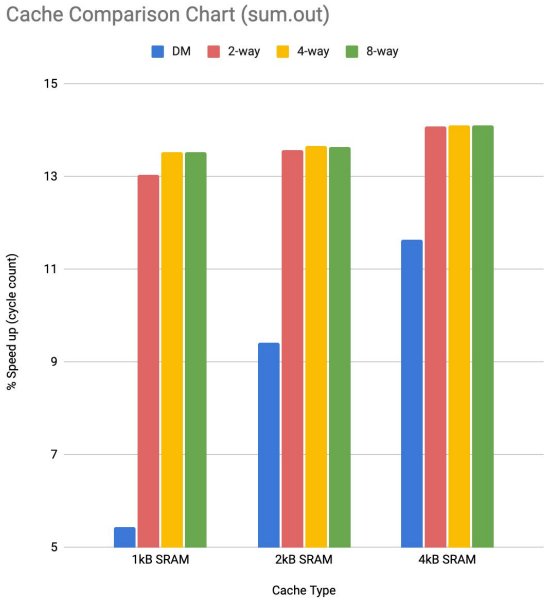


Fig. 9. Cache comparison chart with optimization techniques.
We measured the result using the sum.out benchmark.

Optimizing for frequency

The SRAM macros' positions are important to the length of the critical path, so we tried out many different placement strategies. The strategy mentioned in *Section III* yields the best result. During our simulations, we found that our critical path would wait for the mem_req_ready signal even if the address was a hit in our cache, but with our design, this signal is not needed for hits. Therefore, we also set false paths from memory request ready (mem_req_ready, with large input delay) to the CPU data path by adding the below option in *syn.yml*. These paths would not change the CPU states since mem_req_ready is for cache refill/eviction process.

```
vlsi.inputs.custom_sdc_constraints: [
     "set_false_path -from riscv_top/mem_req_ready -through
[get_db insts -if {.name == */stage1*}]"
]
```

The last idea is to fix the RTL design. Recalling the critical path in Fig. 5, we can move the branch comparator to the EXE stage and add a branch predictor scheme. This strategy removes the extra forwarding muxes that were used for stage 1 and makes the pipeline stages more balanced. We would also need a method to flush out any instructions that occur during a mispredict. However, since this could heavily change our design, we left it as future work.

V. Statistics

Area Utilization

The design has a total area of 0.105 mm^2. The memory occupied 83% of the area which is very significant. Also, the register file in the CPU occupied 50% of the CPU logics. In other words, the actual CPU logics only occupied 8.5% of the overall chip area. This tells us that we can actually design a more complicated CPU architecture to improve the performance. What's interesting is that the LRU unit only accounts for 4.6 μm^2 per cache. This is because our cache is only a 2-way cache, and the complication of the LRU can become significant when the set associativity increases.

Depth	Name	#Inst	Area (um^2)
0	riscv_top	13189	105597.61056
1	cpu	8109	16355.2608
1	mem	4844	87835.90464
2	mem/arbiter	38	73.4832
2	cpu/CLKGATE_RC_CG_HIER_INST8	1	4.19904
2	cpu/fwd	61	104.50944
2	cpu/stage2	1841	2678.52096
2	cpu/CLKGATE_RC_CG_HIER_INST3	1	4.6656
2	cpu/stage1	4364	9959.18976
2	cpu/ctrl	63	72.08352
2	cpu/CLKGATE_RC_CG_HIER_INST1	1	4.19904
2	cpu/CLKGATE_RC_CG_HIER_INST6	1	4.6656
2	mem/dcache	2684	44546.64192
2	cpu/CLKGATE_RC_CG_HIER_INST9	1	4.19904
2	cpu/CLKGATE_RC_CG_HIER_INST4	1	4.6656
2	cpu/CLKGATE_RC_CG_HIER_INST2	1	4.6656
2	cpu/stage3	622	1154.50272
2	cpu/CLKGATE_RC_CG_HIER_INST7	1	4.19904
2	cpu/muxStall	45	72.3168
2	mem/icache	2110	43192.6848
2	cpu/CLKGATE_RC_CG_HIER_INST10	1	4.19904
2	cpu/CLKGATE_RC_CG_HIER_INST0	1	4.6656
2	cpu/CLKGATE_RC_CG_HIER_INST5	1	4.6656
3	cpu/stage2/ALUAsel	58	101.01024
3	cpu/stage1/muxA	80	141.83424
3	cpu/stage2/Bforwarding	72	127.83744
3	cpu/stage1/pcgen	32	44.78976
3	cpu/stage3/csr_reg	281	589.49856
3	cpu/stage1/muxPCsrc	43	76.28256
3	mem/icache/c_array	1235	36076.584
3	cpu/stage3/ldx	46	60.6528
3	cpu/stage2/ALUBsel	48	87.01344
3	cpu/stage1/muxB	63	122.93856
3	cpu/stage2/CSRdata	56	78.38208
3	cpu/stage1/bc	280	395.4096
3	cpu/stage1/pc	80	202.9536
3	mem/icache/c_ctrl	873	7113.768
3	cpu/ctrl/ALUdecode	24	28.92672
3	mem/dcache/c_ctrl	1288	7970.13888
3	mem/dcache/c_array	1395	36575.10336
3	cpu/stage1/regfile	3226	8227.7856
3	cpu/stage3/muxwb	129	235.6128
3	cpu/stage2/Aforwarding	75	129.23712
3	cpu/stage1/muxPC	59	119.43936
3	cpu/stage2/alu	1461	2066.62752
4	cpu/stage3/csr_reg/CLKGATE_RC_CG_HIER_INST43	2	6.29856
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST25	1	4.19904
4	mem/icache/c_ctrl/CLKGATE_RC_CG_HIER_INST49	1	4.19904

4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST21	1	4.19904
4	mem/dcache/c_ctrl/CLKGATE_RC_CG_HIER_INST47	1	4.19904
4	mem/dcache/c_ctrl/CLKGATE_RC_CG_HIER_INST44	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST12	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST33	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST38	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST29	2	5.59872
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST32	1	4.19904
4	mem/icache/c_ctrl/CLKGATE_RC_CG_HIER_INST53	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST19	1	4.19904
4	mem/dcache/c_ctrl/CLKGATE_RC_CG_HIER_INST45	1	4.6656
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST24	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST40	2	5.832
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST15	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST36	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST27	1	4.19904
4	mem/dcache/c_ctrl/genblk4.u_lru	2	3.03264
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST17	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST26	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST22	1	4.19904
4	mem/dcache/c_ctrl/CLKGATE_RC_CG_HIER_INST48	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST13	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST16	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST34	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST39	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST30	1	4.19904
4	cpu/stage3/csr_reg/CLKGATE_RC_CG_HIER_INST42	2	6.29856
4	mem/icache/c_ctrl/genblk4.u_lru	1	1.63296
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST20	1	4.19904
4	mem/dcache/c_ctrl/CLKGATE_RC_CG_HIER_INST46	2	5.59872
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST11	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST41	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST37	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST28	1	4.43232
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST31	1	4.19904
4	mem/icache/c_ctrl/CLKGATE_RC_CG_HIER_INST51	1	4.43232
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST18	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST23	1	4.19904
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST14	2	5.59872
4	cpu/stage1/regfile/CLKGATE_RC_CG_HIER_INST35	1	4.19904

Fig. 10. Area utilization report from innovus

Power Analysis

The total static power consumption is 13.42 mW, with switching power being the major source of power consumption (69.1%). Combinational logics contribute to 85.32% of the power consumption, with sequential elements 11.35%. Fig. 11 shows the static power report. We also did dynamic power analysis using final.out benchmark, shown in Fig. 12. The total dynamic power consumption is 13.97 mW, with switching power accounting for 70.5%. Combinational logics contribute to 85.15% of the power consumption, with sequential elements 10.99%. We can see that this is very similar to static power consumption. We also can find that the dynamic power consumption is larger than the static power consumption (13.97 mW vs. 13.42 mW), with a larger portion of switching power (9.86 mW vs. 9.28 mW). This is reasonable since final.out benchmark is a computationally heavy benchmark. With simulation results, we can more accurately calculate the actual switching activity. Interestingly, the two analyses have the same leakage power (1.394 mW).

```
Total Power
-----------------------------------------------------------------------------------------
Total Internal Power:        2.75144391            20.5000%
Total Switching Power:       9.27535631            69.1073%
Total Leakage Power:         1.39487765            10.3927%
Total Power:                13.42167787
-----------------------------------------------------------------------------------------


Group                           Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
Sequential                         1.237     0.06968      0.2168       1.523       11.35
Macro                                  0     0.04108           0     0.04108      0.3061
IO                                     0           0   7.936e-07   7.936e-07   5.913e-06
Combinational                      1.402        8.88       1.169       11.45       85.32
Clock (Combinational)            0.05216      0.2362   8.507e-05      0.2885       2.149
Clock (Sequential)               0.06032     0.04848    0.008773      0.1176       0.876
-----------------------------------------------------------------------------------------
Total                              2.751       9.275       1.395       13.42         100
-----------------------------------------------------------------------------------------


Rail                  Voltage   Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
VDD                      0.63      2.751       9.275       1.395       13.42         100


Clock                           Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
clk                              0.1125      0.2847    0.008858       0.406       3.025
-----------------------------------------------------------------------------------------
Total                            0.1125      0.2847    0.008858       0.406       3.025
-----------------------------------------------------------------------------------------
```

```
Clock: clk
Clock Period: 0.001030 usec
Clock Toggle Rate:  1941.7475 Mhz
Clock Static Probability:  0.5000



--------------------------------------------------------------------------------
*       Power Distribution Summary:
*              Highest Average Power:  FE_OFC3660_mem_req_valid (BUFx24_ASAP7_75t_SL):         0.08193
*              Highest Leakage Power: mem/icache/c_array/FE_OFC5203_FE_OFN488_cache_din_87 (BUFx24_ASAP7_75t_SL):        0.001023
*                Total Cap:      2.44961e-10 F
*                Total instances in design: 13189
*                Total instances in design with no power:     0
*                Total instances in design with no activty:     0

*                Total Fillers and Decap:     0
--------------------------------------------------------------------------------
```

Fig. 11. Static power report from innovus

```
Total Power
-----------------------------------------------------------------------------------------------
Total Internal Power:        2.71798103        19.4525%
Total Switching Power:       9.85952029        70.5644%
Total Leakage Power:         1.39487800         9.9831%
Total Power:                13.97237930
-----------------------------------------------------------------------------------------------


Group                            Internal   Switching     Leakage       Total  Percentage
                                 Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------------
Sequential                          1.227     0.09243      0.2168       1.536       10.99
Macro                                   0           0           0           0           0
IO                                      0           0   7.936e-07   7.936e-07    5.68e-06
Combinational                       1.378       9.349       1.169        11.9       85.15
Clock (Combinational)             0.05309      0.3536   8.546e-05      0.4068       2.911
Clock (Sequential)                0.05963     0.06417    0.008773      0.1326      0.9489
-----------------------------------------------------------------------------------------------
Total                               2.718        9.86       1.395       13.97         100
-----------------------------------------------------------------------------------------------


Clock                            Internal   Switching     Leakage       Total  Percentage
                                 Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------------
clk                               0.1127      0.4178    0.008858      0.5393        3.86
-----------------------------------------------------------------------------------------------
Total (excluding duplicates)      0.1127      0.4178    0.008858      0.5393        3.86
-----------------------------------------------------------------------------------------------
```

```
Clock: clk
Clock Period:       0.001030000008 usec
Clock Toggle Rate:  1941.7476 MHz
Clock Static Probability:  0.5000


Rail                Voltage   Internal   Switching     Leakage       Total  Percentage
                              Power      Power         Power         Power  (%)
--------------------------------------------------------------------------------
VDD                    0.63      2.718        9.86       1.395       13.97        100


--------------------------------------------------------------------------------
*   Power Distribution Summary:
*              Highest Average Power:  FE_OFC3660_mem_req_valid (BUFx24_ASAP7_75t_SL):         0.08189
*              Highest Leakage Power: mem/icache/c_array/FE_OFC5203_FE_OFN488_cache_din_87 (BUFx24_ASAP7_75t_SL):        0.001023
*         Total Cap:      2.59248e-10 F
*         Total instances in design: 13189
*         Total instances in design with no power:     0
*         Total instances in design with no activity:     0
*         Total Fillers and Decap:     0
--------------------------------------------------------------------------------
```

Fig. 12. Dynamic power report from Voltus using final.out benchmark

Performance

The design and reports we submitted is a RISC-V processor with 1kB 2-way instruction/data cache. We recorded the cycle counts from the RTL simulation results and used the minimum cycle time we obtained from the PAR report (1.03 ns). The performance results of our CPU is shown at Table 1. Fig. 13 shows that we passed the RTL simulation on the provided benchmarks. We also verified that we passed the post-synthesis/post-route simulation on assembly tests in Fig. 14 and Fig. 15 with 1.03 ns cycle time.

Benchmarks	Cycle counts	Execution time (ms)
cachetest.out	4,645,485	4.785
final.out	13,379	0.0138
fib.out	13,089	0.0135
sum.out	27,713,431	28.545
replace.out	27,715,699	28.547

Table 1. 1kB 2-way cache benchmarking results

```
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/bmark_output/cachetest.out         () after 4645485 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/bmark_output/final.out     () after 13379 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/bmark_output/fib.out       () after 13089 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/bmark_output/sum.out       () after 27713413 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/bmark_output/replace.out   () after 27715699 simulation cycles
```

Fig. 13. Passed benchmarks screenshot

```
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/addi.out        () after 5352 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/add.out         () after 5858 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/andi.out        () after 5246 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/and.out         () after 5892 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/auipc.out       () after 4932 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/beq.out         () after 5456 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/bge.out         () after 5504 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/bgeu.out        () after 5554 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/blt.out         () after 5456 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/bltu.out        () after 5512 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/bne.out         () after 5456 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/jal.out         () after 4924 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/jalr.out        () after 5062 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/lb.out  () after 5358 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/lbu.out         () after 5358 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/lh.out  () after 5388 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/lhu.out         () after 5402 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/lui.out         () after 4950 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/lw.out  () after 5408 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/ori.out         () after 5260 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/or.out  () after 5898 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/sb.out  () after 5770 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/sh.out  () after 5888 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/simple.out      () after 4890 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/slli.out        () after 5350 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/sll.out         () after 5920 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/slti.out        () after 5342 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/sltiu.out       () after 5342 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/slt.out         () after 5840 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/sltu.out        () after 5840 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/srai.out        () after 5386 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/sra.out         () after 5964 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/srli.out        () after 5374 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/srl.out         () after 5952 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/sub.out         () after 5836 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/sw.out  () after 5902 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/xori.out        () after 5270 simulation cycles
[ PASSED ] /home/cc/eecs151/fa20/class/eecs151-aar/project_skeleton/asm_output/xor.out         () after 5896 simulation cycles
```

Fig. 14. Post-synthesis passed assembly tests at 1.03 ns

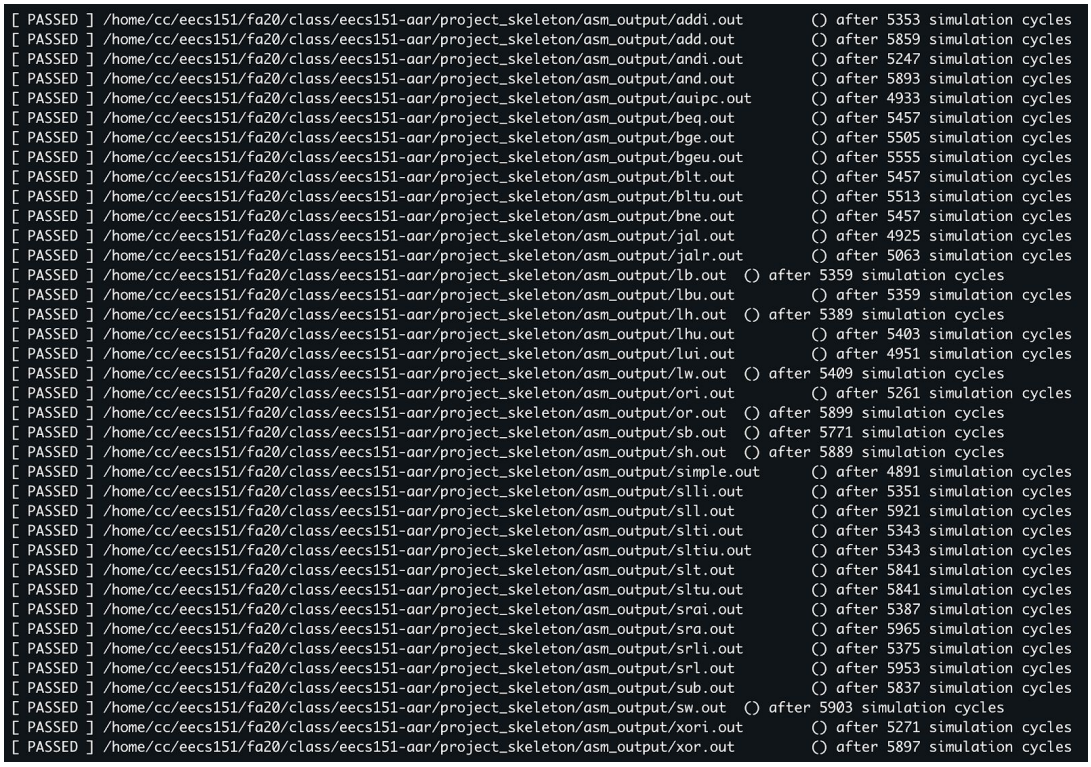


Fig. 15. Post-route passed assembly tests at 1.03 ns

Table 2. shows the most optimized results in terms of cycle counts, with 4kB 4-way cache. We didn't push this cache configuration through the place-and-route flow because its cycle time may heavily increase due to the number of SRAM macros (double of 2-way macros). We show the results for the purpose of showing the optimization using larger and more complicated caches.

Benchmarks	Cycle counts	Execution time (ms)
cachetest.out	4,644,913	N/A
final.out	13,236	N/A
fib.out	12,957	N/A
sum.out	27,371,328	N/A
replace.out	27,378,214	N/A

Table 2. 4kB 4-way cache benchmarking results

VI. Summary

We created a 3-stage pipelined CPU with the following stages: (1) IF & D, (2) EX, and (3) MEM & WB. We incorporated the branch comparator into the decode stage, allowing us to correctly determine the next PC on branches immediately. However, by doing so, we increase our CPU's critical path, add a forwarding mux to stage 1, and have to insert noops for a specific instruction combination. The CPU's critical path (post-syn and post-route) were the worst case forwarding scenarios (MEM to ALU then ALU to branch comparator) to branching. Post-syn involved 63 standard cells, and post-route involved 54 standard cells. The min synthesis clock period that our CPU can operate at was 0.89 ns. The min PAR clock period that our CPU can operate at was 1.03 ns.

For PAR, we used the 1kB 2-way set associative cache configuration, and the PAR frequency (1.03 ns) was used to calculate execution time for bmark tests. The area of the full design was 0.105 mm^2, and the memory occupied a majority of it at 0.87 mm^2 (83%). Our design's static power consumption was 13.42 mW, and the switching power was the major source of it at 9.28 mW (69.1%). We did dynamic power analysis using final.out as well, which showed a total of 13.97 mW for power consumption and switching power of 9.86 mW (70.5%). We passed post-synthesis and post-route simulation on assembly test-suite at the max PAR frequency (1.03 ns). In terms of performance, our cache passed all of the tests in the test-suite: cache test (cycle count = 4,645,485 cycles, execution time = 4.785 ms), final (13,379 cycles, 0.0138 ms), fib (13,089 cycles, 0.0135 ms), sum (27,713,431 cycles, 28.545 ms), and replace (27,715,699 cycles, 28.547 ms). We also compared the cycle counts between the most baseline cache configuration (1 kB DM cache with 1RW tag SRAM) and various cache sizes and configurations. We found that the most optimal cache configuration (in regards to cycle count) was the 4 kB 4-way cache with a 2RW tag SRAM and byte mask data SRAM, with a 14.1% improvement. We optimized our frequency through floorplan placement and designating false paths. As future work, we can better our critical path by moving the branch comparator to the execution stage.

VII. Submissions

All verilog files for implementation are in `src/` directory

All reports from our final synthesis and PAR can be found in `submission/` directory

- `submission/syn_max_0.89ns_reports/*`
  - Reports from max synthesis frequency (0.89 ns) from syn-rundir
- `submission/par_max_1.03ns_timingReports/*`
  - Timing reports from max PAR frequency (1.03 ns) for 1kB 2-way cache from par-rundir
- `submission/innovus.log*`
  - Innovus log from max PAR frequency (1.03 ns) for 1kB 2-way cache from par-rundir

VIII. Acknowledgement

In this project, we learned a lot about how to design an ASIC from scratch. We did design specifications first, and did implementation from RTL coding to PAR. From this project, we learned how to develop caches with the CPU and what the difficulties of implementing the cache mechanism are. The experience to work with the SRAM macros were beneficial as well. Finally, we were able to play around with our design and optimize it as best we could given the timing constraints. We are very happy that we made it through this semester under this difficult time. All of the GSIs were very supportive and helpful for our learning. Especially thankful for Harrison's help all the time on the piazza, you are an awesome TA. Also, we are thankful for Jingyi's insightful recommendation in the checkpoint sessions. This helped us recognize our current obstacles and push us to our potentials. In summary, we learned a lot and had fun during this semester. Thank you once again to the entire teaching staff!