Final Report

ECE 437

Howard Lee, Max Sprague

Section 4

Jimmy Jin

12/14/2025

Overview

The purpose of this report is to provide comparison for the following designs:

1. Pipeline processor without cache
2. Pipeline processor with cache hierarchy
3. Pipeline processor with cache hierarchy integrated into multicore system

This report's primary intent is to measure how pipeline depth, memory hierarchy, and multicore parallelism helps to improve metrics such as CPI, execution time, latency/instruction, and synthesis frequency. Each design provides optimization to the system, for example, with pipelining the processor, the amount of throughput increases thus decreasing the execution time per instruction. Caching the pipeline processor furthermore helps with reducing the memory access penalty, and multicore splits up the total wordload through parallelism and coherence control.

The performance metrics will be generated through using mergesort.asm as a benchmark (dual-merge for dual core). Results will be validated through metrics such as estimated synthesis frequency, average CPI, total execution time, ideal latency per instruction, and speed up of the parallel implementation. The remaining sections of this report detail the full architectural design, including pipeline, cache, and coherence block/state diagrams, followed by measured and estimated results tables, resource utilization summaries, and performance comparisons across all latency configurations.

Design

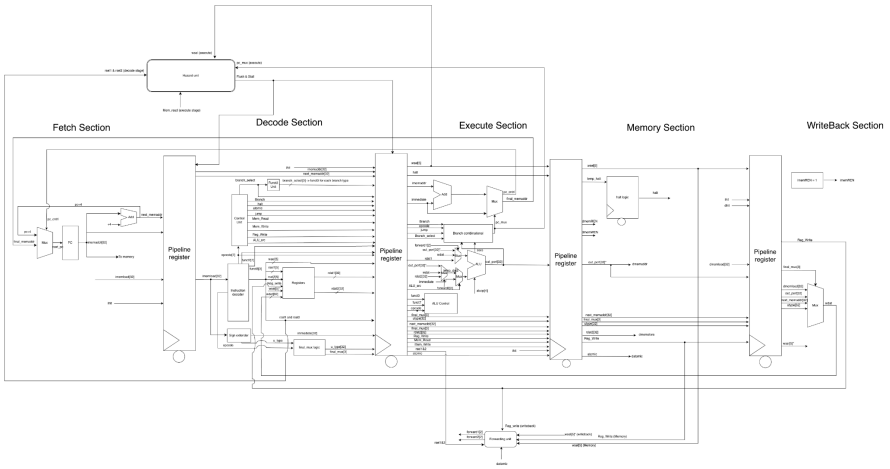

Figure 1. Pipeline Processor

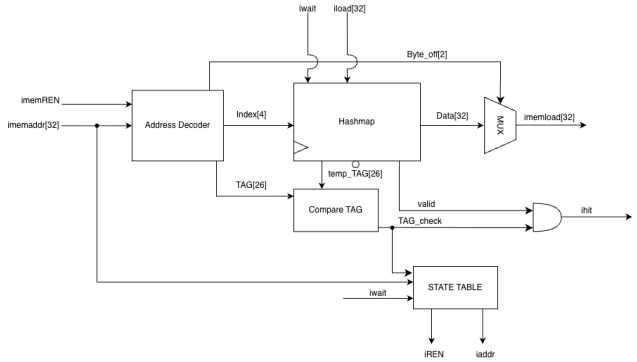

Figure 2. ICache

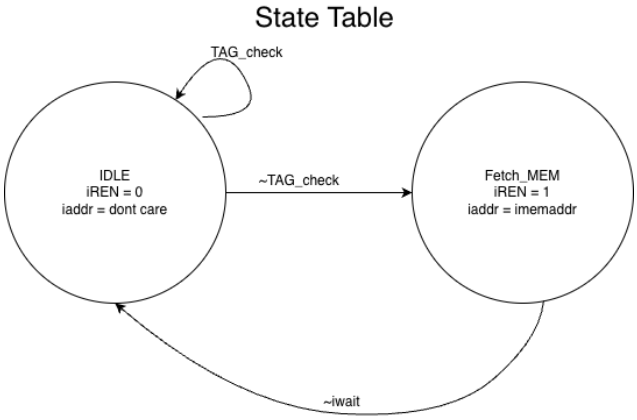

Figure 3. ICache State Transition Diagram

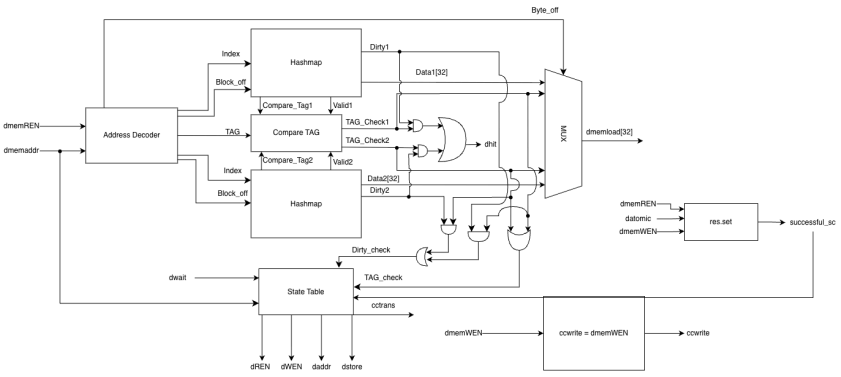

Figure 4. DCache

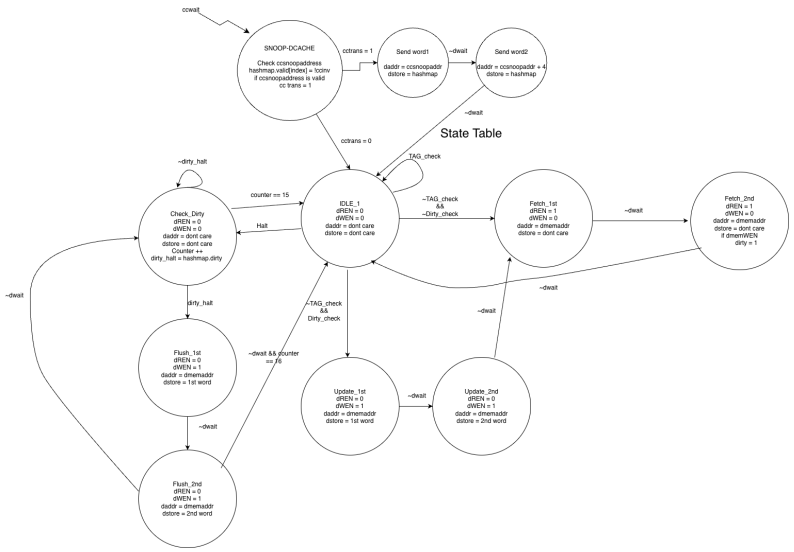

Figure 5. DCache State Transition Diagram

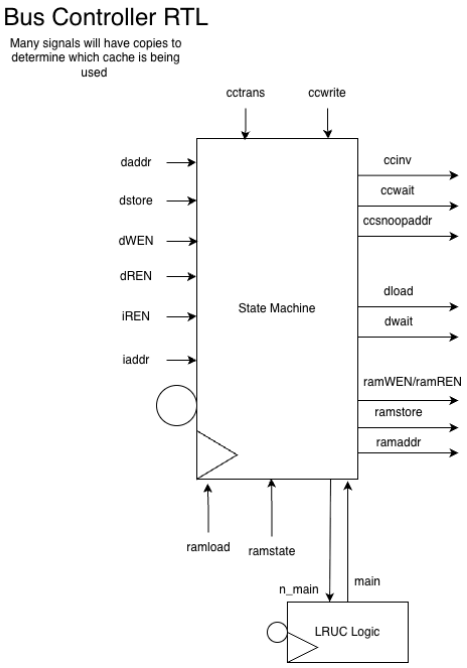

Figure 6. Bus Controller RTL

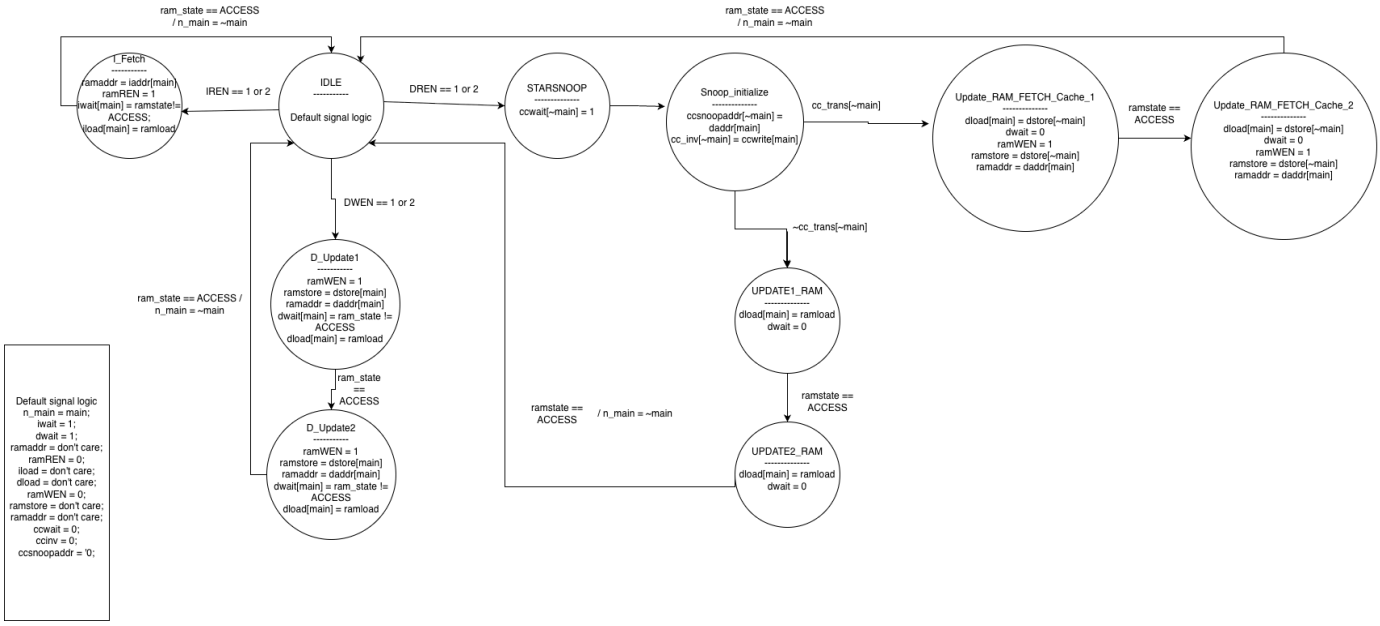

Figure 7. Bus Controller State Transition Diagram

Results

Design Freq. = Min(MAIN/2,CPUCLK), obtain MAIN and CPUCLK under the Timing Summary within the sweep report, labeled as Fmax

Average CLK/Inst. = Cycles/Inst. count, Cycles can be obtained under CLK cycles in the sweep report, Inst. count is given after running sim-t with mergesort.asm

Ideal latency = 1/Target Freq., Target Freq. is given under the Timing Summary within the sweep report, it is the CPUCLK target frequency

Total Execution Time = Inst. count * Avg CLK/Inst. * Ideal latency

Speed up = Total execution time of single cycle / Total execution time of pipeline

Inst. count = 5409

Program used: Merge Sort (Dual Merge sort for Dual Core)

Single Cycle Sweep Report (Main CLK)

Latency	Design Freq.(MHz)	Avg clk/inst.	Ideal latency (Sec)	Total execution time (Sec)
Lat = 0	48.39	1.28	1e-8	6.92e-5
Lat = 2	47.81	2.55	1e-8	1.38e-4
Lat = 6	47.81	5.11	1e-8	2.76e-4
Lat = 10	47.81	7.66	1e-8	4.14e-4

Pipeline without Cache Sweep Report (Main CLK)

Latency	Design Freq.(MHz)	Avg clk/inst.	Ideal latency (Sec)	Total execution time (Sec)
Lat = 0	74.49	1.88	1E-8	1.01E-4
Lat = 2	74.79	3.75	1E-8	2.03E-4
Lat = 6	74.79	7.18	1E-8	3.88E-4
Lat = 10	74.79	10.62	1E-8	5.74E-4

Speed-ups

(How much faster Pipelined is than Single Cycle)

Latency	Speed-up
Lat = 0	-31.5 %
Lat = 2	-32 %
Lat = 6	-29 %
Lat = 10	-28 %

Pipeline with Cache Sweep Report (Main CLK)

Latency	Design Freq.(MHz)	Avg clk/inst.	Ideal latency (Sec)	Total execution time (Sec)
Lat = 0	81.97	1.57	1E-8	8.49E-5
Lat = 2	76.46	2.61	1E-8	1.41E-4
Lat = 6	76.46	3.65	1E-8	1.95E-4
Lat = 10	76.46	4.88	1E-8	2.64E-4

Speed-ups

(How much faster Pipelined w/cache than without cache)

Latency	Speed-up
Lat = 0	18.90 %
Lat = 2	44.40 %
Lat = 6	98.97 %
Lat = 10	117.40 %

Instruction Count: 5429

Dual Core processor w/cache (Main CLK)

Latency	**Design Freq.(MHz)**	**Avg clk/inst.**	**Ideal latency (Sec)**	**Total execution time (Sec)**
Lat = 0	70.49	1.42	1E-8	7.71E-5
Lat = 2	66.61	1.59	1E-8	8.63E-5
Lat = 6	66.61	1.95	1E-8	1.06E-4
Lat = 10	66.61	2.33	1E-8	1.27E-4

Speed-ups

(How much faster Dual core is than single core)

Latency	**Speed-up**
Lat = 0	10.1 %
Lat = 2	63.3 %
Lat = 6	84.0 %
Lat = 10	107.9 %

FPGA Resources for pipeline without cache

Ref Name	Used
FDCE	1759
LUT6	1103
LUT4	481
LUT5	325
LUT2	285
MUXF7	256
LUT3	113
IBUF	83
OBUF	65
MUXF8	64
CARRY4	52
RAMB36E1	16
LUT1	2
FDPE	2
BUFG	2

FPGA Resource for dual core Design

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	11026	0	0	32600	33.82
LUT as Logic	11002	0	0	32600	33.75
LUT as Memory	24	0	0	9600	0.25
LUT as Distributed RAM	24	0			
LUT as Shift Register	0	0			
Slice Registers	9978	0	0	65200	15.30
Register as Flip Flop	9914	0	0	65200	15.21
Register as Latch	64	0	0	65200	0.10
F7 Muxes	1591	0	0	16300	9.76
F8 Muxes	205	0	0	8150	2.52

Contributions

Howard Lee, Senior, Electrical Engineering, May ‘26

- Design and testing of Hazard Unit
- Design and testing of Pipeline Latches
- Design and testing of ICache and DCache
- LR-SC implementation in DCache
- Overall testing and debugging

Max Sprague, Senior, Electrical Engineering, May’26

- Design and testing of Forwarding Unit
- Reorganization/restructuring of Datapath flow
- Design and implementation of Bus Controller
- LR-SC implementation in Datapath
- Overall testing and debugging

Conclusion

Our results show that the pipelined processor without caches performs worse than the single-cycle design when running our benchmark (mergesort). The measured speed-ups are negative across all memory latencies, with the pipelined design being 28–32% slower than the single-cycle version. For example, with zero memory latency the pipelined processor is 31.5% slower, and even at higher latencies (e.g., Lat=10) it remains about 28% slower. The primary reason for this slowdown is mergesort's branching behavior. Mergesort contains nested loops and recursive-style control flow, which generate frequent conditional branches. Because our pipeline assumes all branches are not taken, the processor incurs a 3 cycle misprediction penalty for every taken branch. These misprediction penalties accumulate rapidly in a loop heavy program.

With caches enabled, the pipelined processor shows a clear improvement compared to the same pipeline without caches. Across all tested memory latencies, the speed-ups range from 18–117.4%. This behavior is expected because caches reduce the effective memory access time, and mergesort repeated accesses to similar or identical memory locations. The exploitation of spatial and temporal locality present in mergesort allows the processor to avoid long stall due to memory access. As a result, the pipeline is able to maintain higher throughput, which helps decrease some of the performance penalties previously caused by branch mispredictions.

With dual core, now we see performance significantly improved by a great amount. Across all latencies, the speed-ups range from 10-107%. The reason why we are seeing improvements going from single core to dual core is simple - you split the workload by half. Furthermore, in single core, whenever we have any long-latency memory operations, it completely stalls execution. For dual cores, one core could be stalled and another one could be running other operations without being affected. Another thing to note is the increase in speed ups due to higher latencies, this happens because in higher latencies, the miss penalty increases, but with cache implemented, that effectively cuts the miss rate going from 100% down to a lower number, which can be seen in later design speed ups with caches.