ECE 437L Final Report

Brian Rieder (mg267)

Pooja Kale (mg273)

TA: Nick Pfister

9 December 2016

1 Executive Summary

The purpose of this report is to provide a comparison of the pipeline processor with and without a cache heirarchy and the pipeline processor with cache and a multicore processor. We will be using 4 different points of comparison for this report. To compare the instruction processing speed, three different CPU designs, we will look at the estimated synthesis frequency of each design, the number of instructions executed per clock cycle, and the latency of one instruction, the amount of time it take to complete one unstruction on the multicore. The comparisons will be further made by looking at the FPGA resources for required for our design using Total Utilization Combinational Functions on each design, the utilized dedicated logic registers and the speedup from Sequential to Parallel.

The pipeline without caches is design with five stages: instruction fetch, instruction decode, execute, memory, and write-back. This allows the design to process five instructions simultaneously within different stages of their execution. The pipeline design does not have a cache system, forcing the design to always access RAM memory. The pipeline design with caches has a level 1 cache hierarchy consisting of a direct mapped instruction cache with a two-way associative data cache. This allows the fast access to addresses one level above the memory. The final multicore design includes a coherence control unit. The purpose of this unit is to maintain a coherent address space for all processors, for any address all processors see the same data. The coherency logic will be in two places, the memory control unit and the data cache. This allows multiple instructions to be processed at the same time.

The three designs will be tested using the merge-sort assembly file and the dualmergesort file. The benefit of the merge-sort files is that it includes all possible assembly instructions that all three processors can decode and execute. The merge-sort file also includes multiple branches to test branch logic in all three designs. The merge-sort files are also more complex giving more accurate comparisons between the three designs.

2 Processor Design

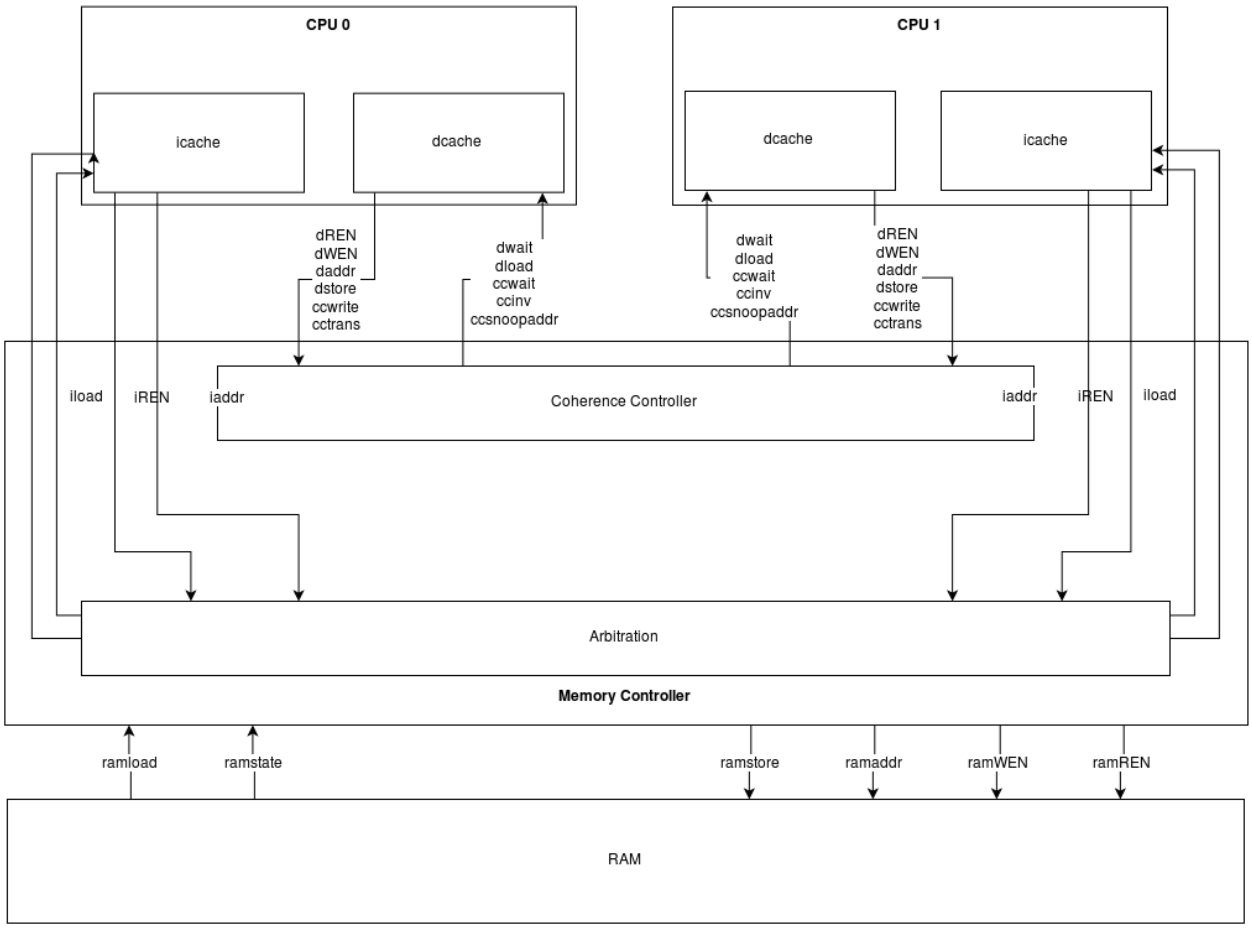

Figure 1: Multicore Block Diagram

Text for your multicore design

Text for your pipeline design

Text for your cache design

3 Processor Debug

4 Results

Text for your results

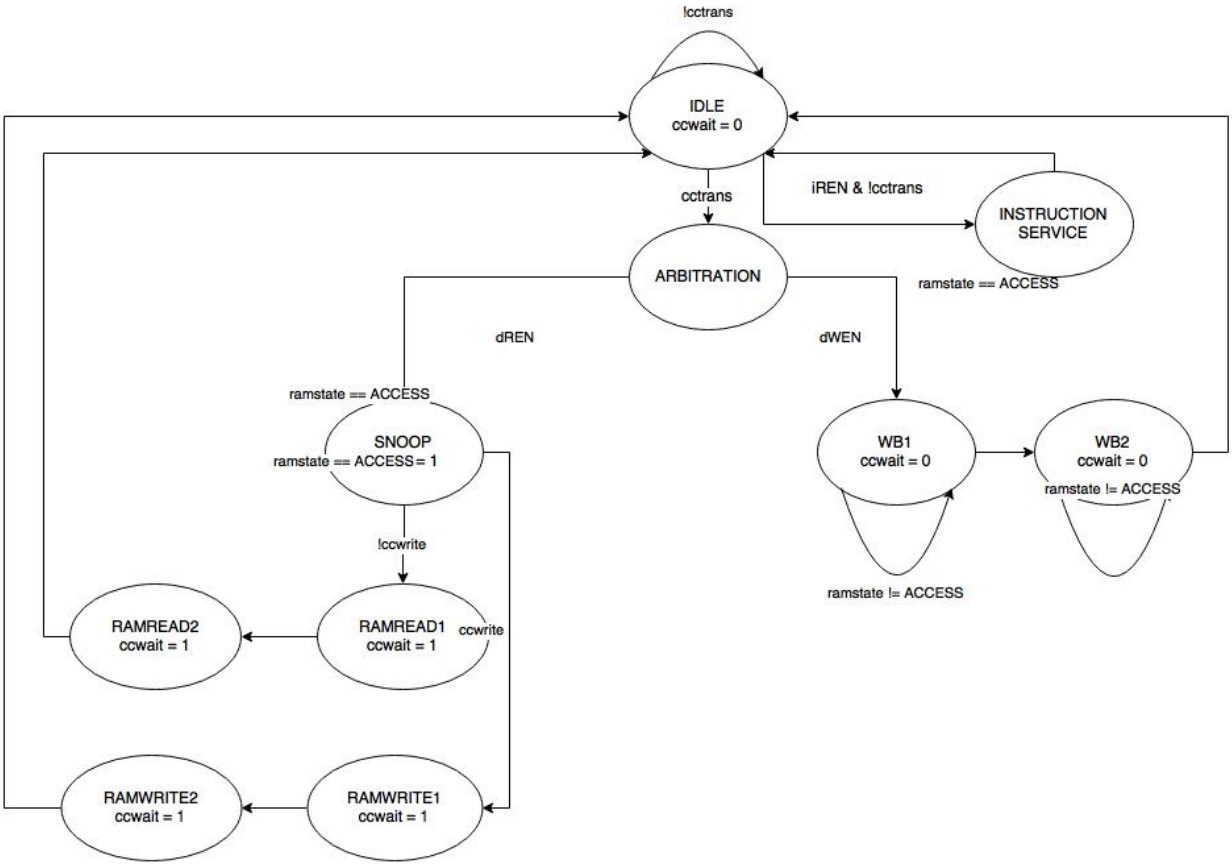

Figure 2: Cache Coherency Unit Block Diagram

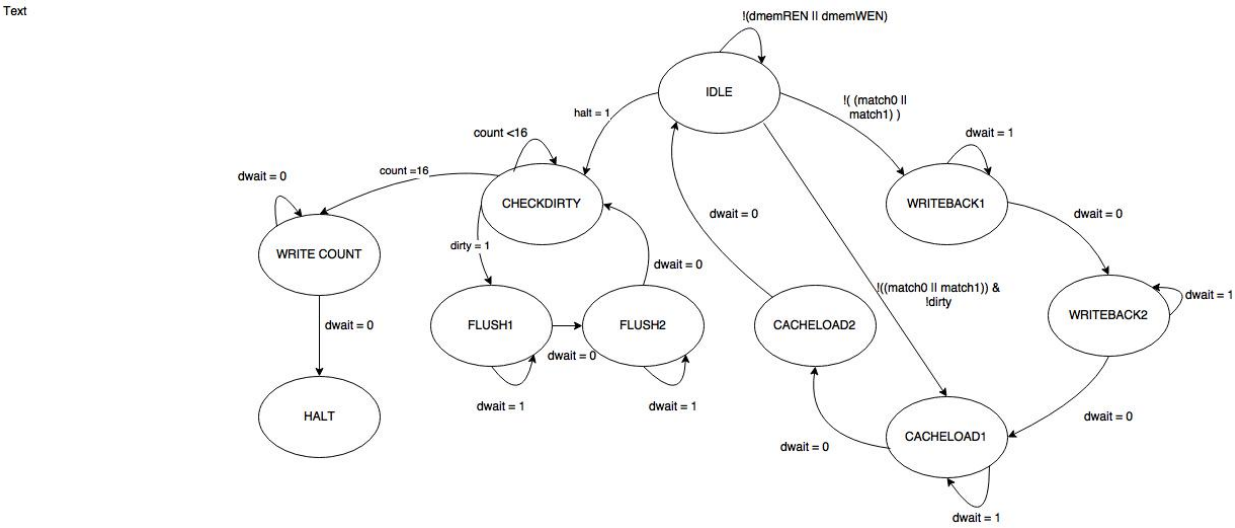

Figure 3: Cache State Diagram

5 Conclusion

Text for your conclusion

6 Contributions and Collaboration Strategies

Placeholder for contribution section intro

6.1 Individual Contributions

Placeholder for individual contributions

6.2 Collaboration Strategies

Account information was shared between team members at the beginning of the collaboration stages and SSH keys were set up between the two accounts for ease of user-switching, but each team member did their development within their respective accounts and contributed

	Pipeline without Cache	**Pipeline with Cache**	**Multicore**
Estimated Synthesis Frequency	??		
Instructions per Cycle (IPC)	??ns		
Single Instruction Latency			
Total Utilized Combinational Functions			
Utilized Dedicated Logic Registers			
Speedup from Sequential to Parallel			

Table 1: Synthesis Results

to GitHub using their own account. Because of this Git usage and presence on GitHub, all contributions are documented in a quantitative manner within the repository.

6.2.1 Git VCS Usage

Just as in the midterm, in order to compartmentalize development of features and to keep both contributors on the same page, an industry strategy of progressive-stability branching was employed and all development was done within respective branches and "work silos" - a master branch was maintained with the most recent functioning design, "singlecycle", "pipeline", "caches", and "multicore" branches were maintained, and each feature were developed on an independent branch for that feature (some examples being "branchpredict" and "forwarding" in the earlier stages of development). While this required more overhead in terms of planning and maintenance, the preservation of states and releases within the development process enabled ease of rollbacks, identification of changes, and the ability to develop without worrying about destruction of a prior working state.