**Extra Credit**

In addition to the baseline design, our team implemented no-write allocation for extra credit. No-write allocation is a write-miss policy where data bypasses the cache and writes directly into main memory. Real world caches often pair this with a corresponding write through policy, where write hits cause data to be written to both main memory and the cache. Since the baseline design already implements the write-through policy, only a small modification was needed to add no-write allocation.

No new states were needed within the finite state machine of the control module to accommodate the new write-miss policy. Instead, the next state values under certain conditions were changed. The FSM has both a WriteMiss and a WriteHit state, both of which are identical in the baseline implementation. The WriteMiss state was modified to bypass the WriteData state, which writes data into the cache, and instead directly passes the data to main memory by going to the WriteMem state. This satisfies the requirements for no-write allocation.

The team also attempted to implement the write-back no-write policy for additional credit, but did not finish in time. Write-back refers to a write-hit policy where data is written into the cache, but does not get written into main memory until it is replaced by more data. To mark weather a block has been modified while it is in the cache, a ‘dirty bit’ is associated with each block. Our incomplete implementation had added this dirty bit as a simple 1-bit array in the tag ram, but the control logic to handle it was never finished.

**Testing Procedure**

For the baseline design, a test procedure was provided with the skeleton code. This procedure consisted of a sample input program which writes a certain value to memory, and code within the testbench module that checks those locations in memory for correctness. This code even printed the results of the simulation to the terminal, so no waveform analysis was necessary to confirm correct operation.

Extra verification was needed to test the functionality of the no-write allocation version. Because no-write allocation is a write-miss policy, a program was written to intentionally trigger a write-miss. This program essentially populates two registers with two different values, then tries to store each register to the same location in memory. This causes a write-miss in the cache. Correct operation is then verified by checking the waveform to ensure that a write-miss was detected and that it was handled correctly. Because the processor implementation was incomplete, the program used simple data processing operations to populate the registers as opposed to using MOV with immediate inputs.

**Evaluation**

The baseline design successfully passed the provided test procedure. Additionally, waveform analysis reveals that the finite state machine implemented for the control section functioned as expected. The extra credit-design project also passed its own corresponding test. The test for this was not automated, but an analysis of the waveforms in Modelsim reveals that the control logic detects the write-miss and that the cache block as a whole handles the data correctly.

In the context of other labs, this cache implementation is not necessarily much faster than the normal memory implementation in the previous labs. This is because the RTL Verilog codebase does not take into account the longer read time of main memory versus the cache: both are stored as the same data type in code. Restructuring the code to account for these would not have made an apparent difference right away either, as the simple input programs used to test the previous labs would not benefit much from the cache.

Overall, this lab was an effective conclusion to the lab portion of this course. Memory hierarchy is utilized in essentially every commercially available processor, so an understanding of these modules and their operations allows for a more productive use of them in the future.