Yihang Bi

EECS 570

Apr. 8, 2025

# Cache Coherency Protocol Implementation and Verification Report

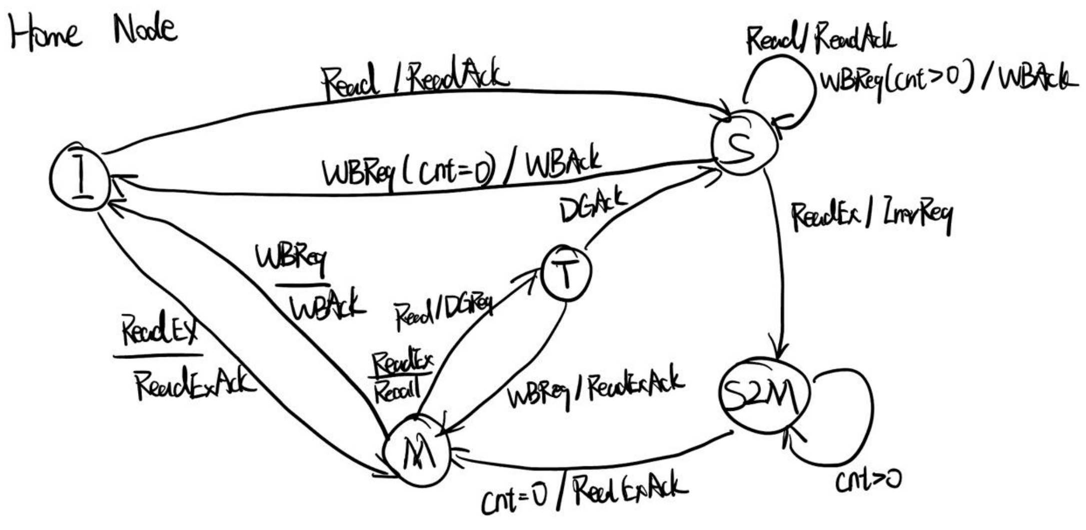
In this programming assignment, I implemented the MSI protocol for the baseline protocol and verified the feasibility and integrity of it. I then optimized the baseline protocol by adding the self-downgrade and E-state to the protocol. All my work is done on top of the example given by the starter files, and I reused some of the code from *twostate.m*.

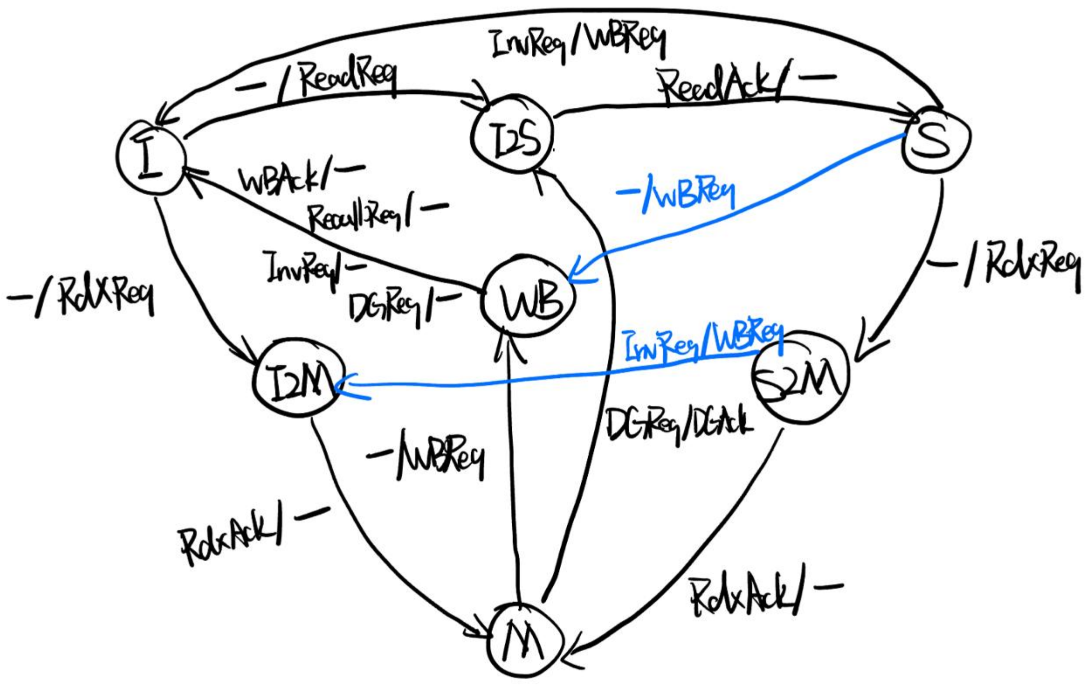
Besides the two new states, I added many transition pending states to the state machine. The main purposes of those new pending states are for:

1. Protect the integrity and coherency of caches across processors and home node.
2. Prevent message overflow on the ports.

As a result, I added 1 pending state to the home node and 4 pending states to the processors. I believe that some of the new pending states can be merged (such as I2M and S2M) into one because they have similar transition conditions, but I decided to separate them for debugging purposes.

Below is the state machine of both home node and processors with MSI protocol.





I added as many assertions and invariants as I could before running the code. I failed some of the assertions and invariants at the first few attempts, but they turned out to be mistakes I made during programming. After fixing those the program ended without errors found, proving the MSI protocol's effectiveness.

I added the auto-downgrade feature on the base of MSI protocol. The processor will spontaneously downgrade itself from modified to shared state. I added another pending state solely for this transition. The self-downgrading processor will send AutoDowngradeReq to the home node and the home node will change the state accordingly. One thing worth noticing is I added another channel to the entire network. In theory the third VC is unnecessary because the message in the inbox will not exceed 2. However, in practice, due to the out-of-order message processing rule, some message will always block the channel and cause deadlock. It might be solved by dynamically allocating the channel while sending the message. However, when every message has to determine its channels during the compilation, I must add another channel to prevent conflicts.

So far when I am writing this report, I am still working on the MESI protocol. I made significant progress on it but there's unexpected behavior. If I can deliver by deadline I will include those optimizations to my final submission.

If I have more time on this assignment, besides exploring more optimization on the protocol, I will improve my implementation so that it will be more efficient. As I mentioned, I would be able to reduce the number of pending states and VCs, which will improve the performance of the cache communication.