Lab 5 Report

Section1. Answers to the Pre-lab questions

1. **Why are transient states necessary?**

Transient states are necessary because the set of actions required in MSI directory protocol is not atomic. If it is not atomic, if Core A reads a block, it goes into S state and stays there until the actions are completed, Core B comes along and do a write and finishes before Core A's actions finishes, Core A will be in S while Core B will be in M. This violate single writer/multiple readers invariant. So transient states solve this issue by facilitate the transitions between two stable states.

2. **Why does a coherence protocol use stalls?**

Coherence protocol uses stalls to ensure each set of actions is atomic. If you want to do something and see someone is in the transient state, you must stall until the other one is out of the transient state before proceeding. This ensures each set of actions is atomic.

3. **What is deadlock and how can we avoid it?**

Deadlock happens when A is waiting for B and B is waiting for A. Since they are both waiting for each other, none can proceed. This is a deadlock. In coherence cache. we avoid deadlock using virtual network rather than single message buffer. Virtual networks haves its own queues, buffer and message tags thus can bypass the message travelling on other virtual networks.

4. **What is the functionality of Put-Ack (i.e., WB-Ack) messages? They are used as a response to which message types?**

The functionality of Put-Ack messages is to acknowledge the Cache controller-initiated request message type. They acknowledge writeback requests of the caches.

5. **What determines who is the sender of a data reply to the requesting core? Which are the possible options?**

The states of each processor determine the sender of a data reply.
If there is a Core's cache block is in M state, the data came from that Core.
If there is no cache block in M state, data came from memory.

6. **What is the difference between a Put-Ack and an Inv-Ack message?**

Put-Ack message acknowledges writeback requests while Inv-Ack acknowledges invalidate requests.

Section2. Answers to all questions from Section 6

1. **How does the FSM know it has received the last Inv Ack reply for a TBE entry?**

FSM has an ack counter per TBE entry. When the counter is 1 and it receives an Inv Ack, FSM knows that the Inv Ack that was just received was the last Inv Ack reply for a TBE entry.

2. **How is it possible to receive a PUTM request from a Nonowner core? Please provide set of events that will lead to this scenario.**

This is possible due to propagation of signal.

Scenario: Core A has cache block A in modified state => Core B wants to write to cache block A => Directory gets Core B's request, sends Core A a GETM request for cache block A and removes Core A as the owner of the cache block A => When Core A gets the GETM signal, it will send a PUTM signal along with the required data. Core A isn't the owner but still send the PUTM signal.

3. **Why do we need to differentiate between PUTS and a PUTS-Last request?**

PUTS is a signal that tells the directory, the cache block is transitioning from S -> I. PUTS-Last does the same thing but also tells the directory that the cache block that send this info is the last sharer of the cache block. After receiving PUTS-Last the directory's FSM can transition from shared to invalid but if the FSM received PUTS, the state doesn't change as not all the sharers have replied.

4. **How is it possible to receive a PUTS Last request for a block in modified state in the directory? Please provide set of events that will lead to this scenario.**

Core A,B,C are in shared state for cache block A => Core A wants to write to cache block A => Directory gets the request, sends Core B,C invalidated request and place Core A in modified state=> Core B gets the invalid request and sends PUTS to directory => Directory gets the signal Core B => Core C gets invalid request and sends PUTS Last to directory as it is the last sharer => Directory now gets a PUTS Last request for a block in modified state.

5. **Why is it not possible to get an invalidation request for a cache block in a Modified state? Please explain.**

This is because the data in a cache block in modified state cannot be invalidated since it modified the copy of the cache and other require that copy for coherency. Thus, if the cache block is kicked out of modified state, the signal it will receive is GETM since the other caches required the must up to date copy of the cache.

6. **Why is it not possible for the cache controller to get a Fwd-GetS request for a block in SI_A state? Please explain.**

This is because Fwd-GetS request is meant for the cache block that is in modified state to forward its copy to the sharers since all copies of the cache block should be the same. SI_A is a transition state from shared to invalid or MI_A to invalid. In the shared state, the data in the cache block is the same as other copies of the same cache block thus it makes no sense to get a Fwd-GetS request. In MI_A case, MI_A got Fwd-GetS and transition to SI_A. This means it has already forwarded the data already so it will not do it again in SI_A.

7. **Was your verification testing exhaustive? How can you ensure that the random tester has exercised all possible transitions?**

Yes. We ensure that random tester has exercised all possible transition by limiting the number of cache blocks. With a few cache blocks, there is a much higher chance of blocks being evicted and changing states. With more changing states, we can presume that there is a higher probability that all states were visited sometime during the testing process.

Section3. Modifications with respect to Tables 8.1 and 8.2

We have one more event with respect to Table 8.1, the modifications we made are:

	Data_from_Dir_Ack_Cnt_Last
IM_AD	Write Data to Cache/ M
SM_AD	Write Data to Cache/ M

The modification on Table 8.2 is huge. Thus, we made a new Table 8.2:

	GetS	GetM	PutM-Owner	PutM-NotOwner	PutS-NotLast	PutS-Last	Memory-Data (MD)	Memory-Ack	Data (D)
I	Add Req to sharers/ IS_MD	Add Req to owner/ IM_MD		Send Put-Ack to Req	Send Put-Ack to Req	Send Put-Ack to Req			
S	Send data to Req, add Req to sharers	Send data to Req, send Inv to sharers, clear sharers, set Owner to Req		Send Put-Ack to Req	Remove Req from sharers, send Put-Ack to Req	Remove Req from sharers, send Put-Ack to Req/ I			
M	Send Fwd-GetS to Owner, add Req and Owner to Sharers/ MS_D	Send Fwd-GetM to Owner, set Owner to Req	Write Data into Memory/ MI_MA	Send Put-Ack to Req	Send Put-Ack to Req	Send Put-Ack to Req			
IS_MD	stall	stall	stall	stall	stall	stall	Send Data to sharer/S	stall	stall
IM_MD	stall	stall	stall	stall	stall	stall	Send Data to owner/M	stall	stall
MS_D	stall	stall		Remove Req from sharers, send Put-Ack to Req	Remove Req from sharers, send Put-Ack to Req	Remove Req from sharers, send Put-Ack to Req			Write Data to Memory/MS_MA
MS_MA	stall	stall		Remove Req from sharers, send Put-Ack to Req	Remove Req from sharers, send Put-Ack to Req	Remove Req from sharers, send Put-Ack to Req		Go to S state	
MI_MA	stall	stall		Remove Req from sharers, send Put-Ack to Req	Remove Req from sharers, send Put-Ack to Req	Remove Req from sharers, send Put-Ack to Req	Send Put-Ack to Req	Clear Owner, Send Put-Ack to Req	

Section4. Statement of Work:

Everything is evenly separated between two team members