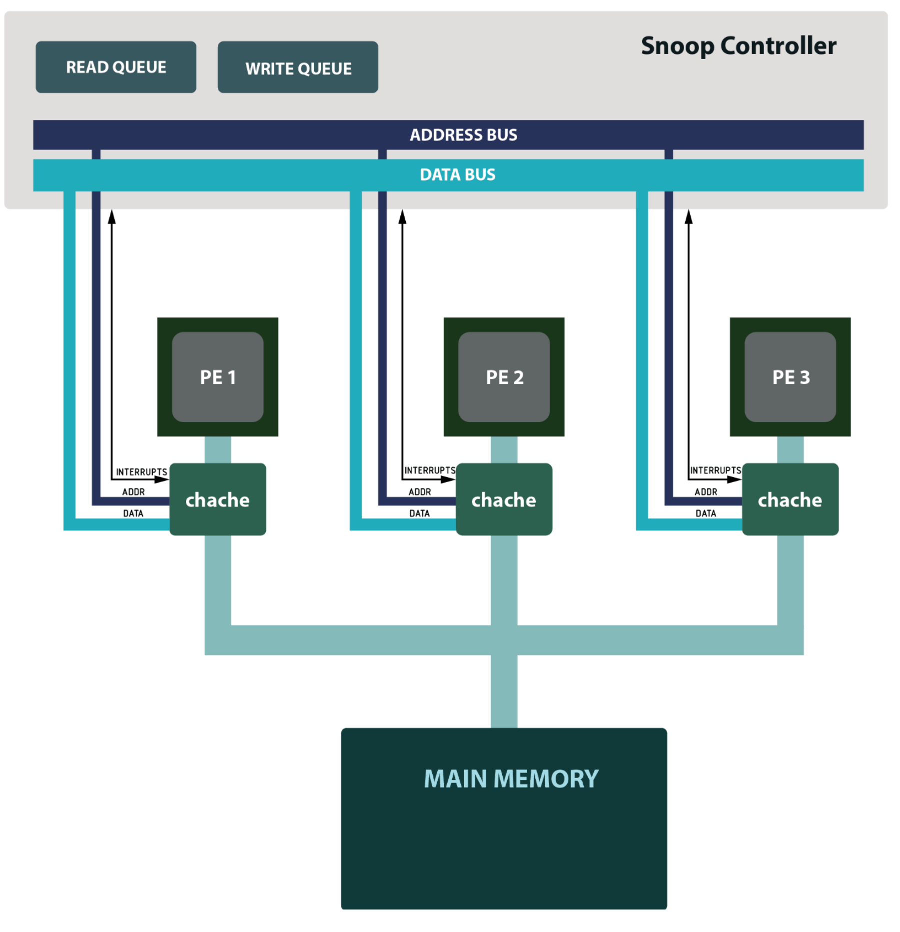
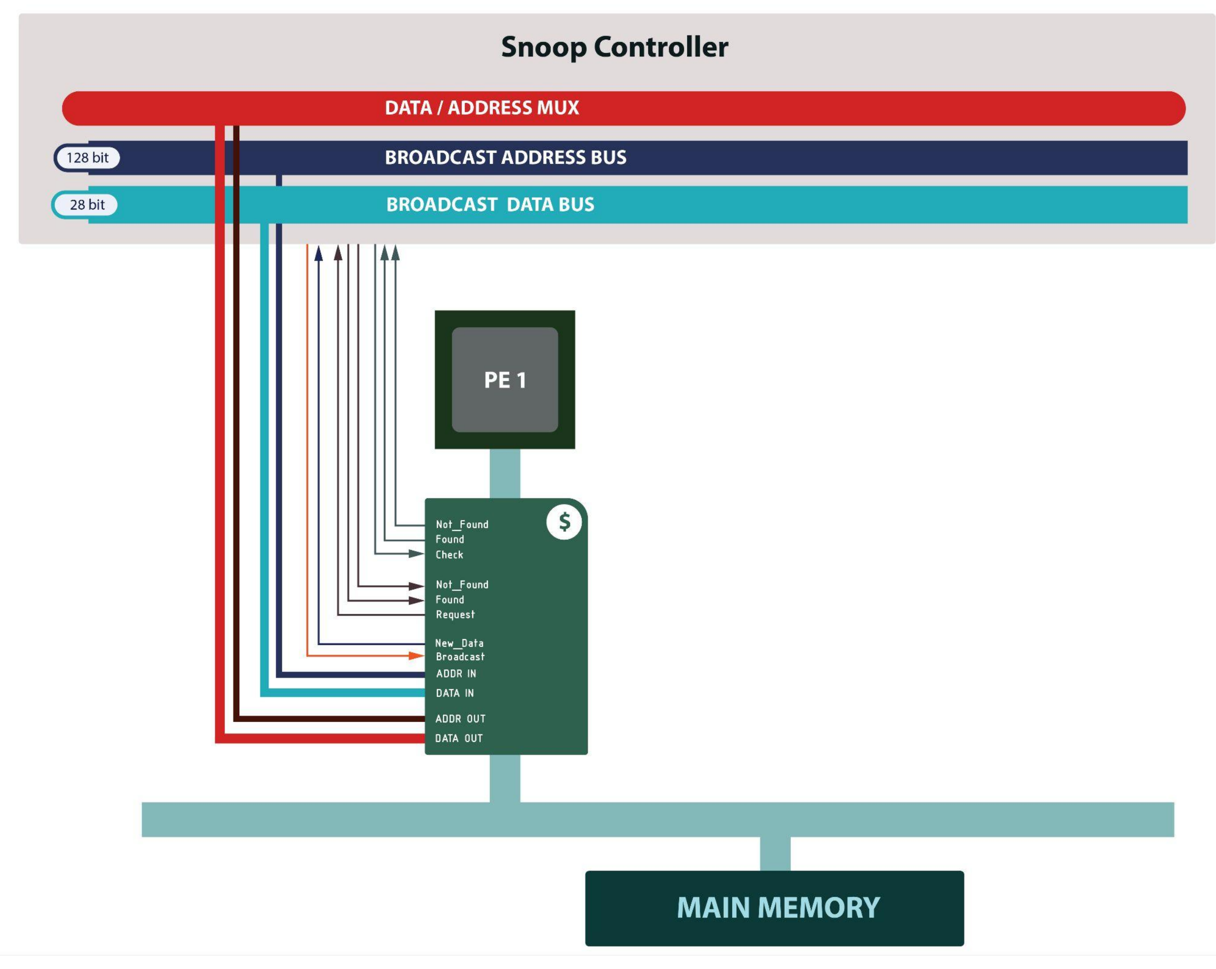
Introduction

In this phase our task is to build a MultiProcessor system on a chip with shared memory. In this system each processor element has separate caches. And all of them are connected to the snoop controller this snoop controller will handle all the sharing scenarios throughout the multi processor system. The local caches are directly connected to the main memory as well. If the requested block is not available inside any of the local caches they are fetched from the main memory. The caching protocol is mainly based on the MESI protocol. The following diagram shows how the processing elements and the snoop controller are connected. To reduce the complexity and bus conflicts we have designed a centralized system. The snoop controller is coordinating all the bus activities. The system is an interrupt based one so the system works in an asynchronous manner.





Cache

There are separate caches for each processing element. The cache is the unit responsible for requesting data blocks from the snoop controller when required and notifying the snoop controller when there is an update in a cache block which is shared between each cache. To track the state of each block it has to maintain a state for each cache block. This state is used to decide what action should be taken in a state change. We have decided to use a state machine which follows MESI protocol, in order to update those states. In this state machine there are 4 states as follows

Invalid - the data block is Invalid. A data block block will be marked as invalid, if a snoop invalidation hits for a specific shared block. An invalid block will be fetched form and marked as shares. If it was fetched from the memory, it will be marked as Exclusive.

Shared - this state means that this block is shared between several caches.

Exclusive - this state means that this block is the only copy cached from the memory. An exclusive block will be changed to a shared in a case of snoop hit.

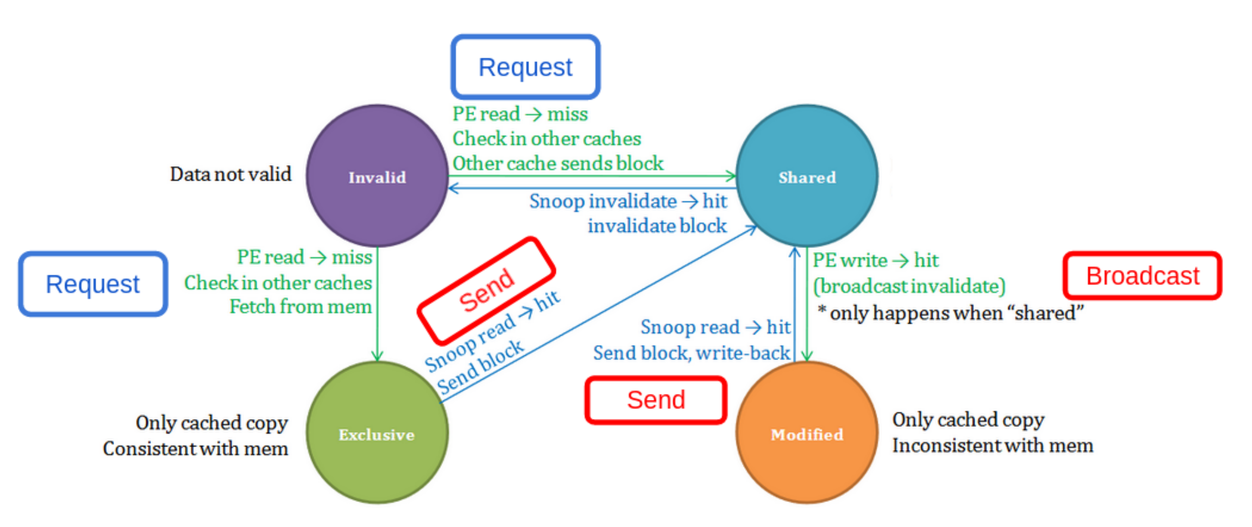
Modified - this means that this block is the only cached copy that is inconsistent with the Memory

For any given pair of caches, the permitted states of a given cache line are as follows

	M	E	S	I
M	✗	✗	✗	✓
E	✗	✗	✗	✓
S	✗	✗	✓	✓
I	✓	✓	✓	✓

When the block is marked M (modified) or E (exclusive), the copies of the block in other Caches are marked as I (Invalid).

The state diagram of this state machine as follows



From the state transitions above we have found that there are 3 processes that will interact between the snoop controller and the cache. Those processes are as follows.

Request

This is the process that is required to execute when the cache hits a miss and seeks for that specific cache block in other caches in the system. This request could end up in two results. Either the snoop controller found or not found that specific block in another cache.

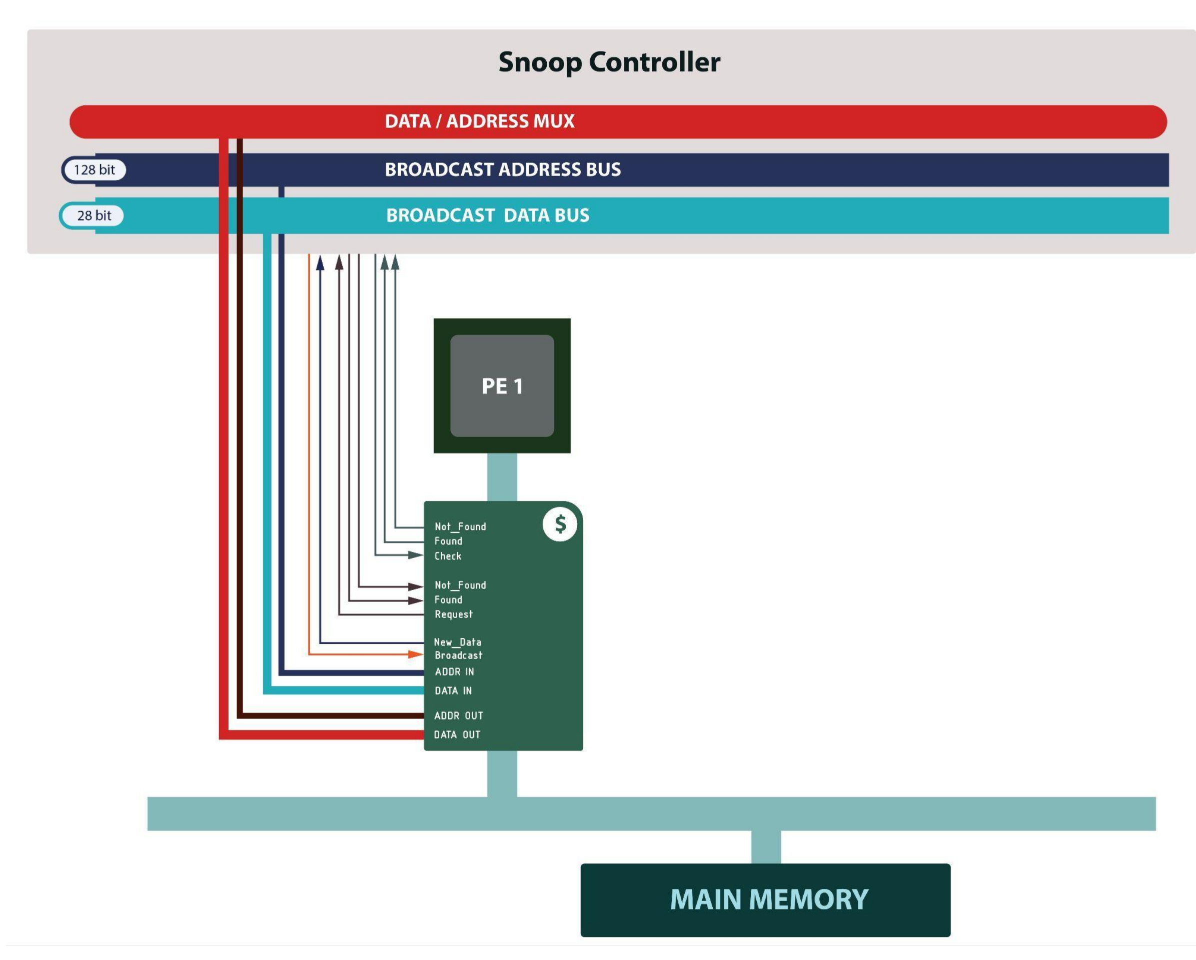
Send

In a case that the cache is required to send a specific block to the snoop controller, it is required to perform a send operation. This happens on two occasions. One is if an exclusive block had a snoop hit, and the other is if a modified block got a snoop hit.

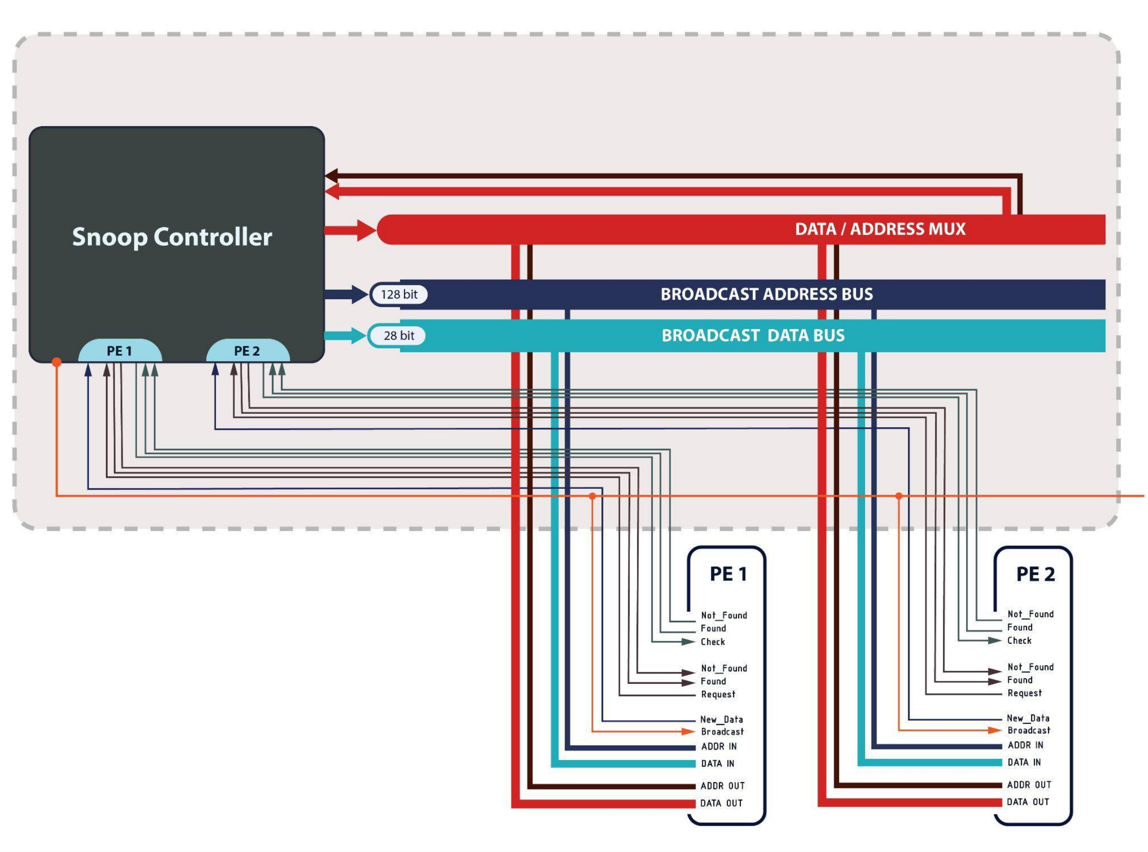
Broadcast

This operation is required to notify the snoop controller that a shared block has a PE hit and is modified. This broadcast will be broadcasted through the snoop controller in order to invalidate that specific block in other caches.

Data Lines, Control Lines and Busses



The snoop controller uses multiplexers to choose the data and address from each processing element.



Signals for two processing elements

```
BROADCAST_ADDR_BUS,
BROADCAST_DATA_BUS,
BROADCAST_INTERRUPT,
CPU_1_INPUT_ADDR_BUS,
CPU_1_INPUT_DATA_BUS,
CPU_1_INPUT_NEW_DATA_INTERUPT,
CPU_1_REQUEST_DATA_INTERUPT,
CPU_1_INPUT_REQUEST_DATA_INTERUPT,
CPU_1_INPUT_DATA_NOT_AVAILABLE,
CPU_1_INPUT_DATA_AVAILABLE,
CPU_1_OUTPUT_DATA_FOUND,
CPU_1_OUTPUT_DATA_NOT_FOUND,

CPU_2_INPUT_ADDR_BUS,
CPU_2_INPUT_DATA_BUS,
CPU_2_INPUT_NEW_DATA_INTERUPT,
CPU_2_REQUEST_DATA_INTERUPT,
CPU_2_INPUT_REQUEST_DATA_INTERUPT,
CPU_2_INPUT_DATA_NOT_AVAILABLE,
CPU_2_INPUT_DATA_AVAILABLE,
CPU_2_OUTPUT_DATA_FOUND,
CPU_2_OUTPUT_DATA_NOT_FOUND
```

Procedures

1. ***On Update Local Cache of Processor Element - Send***
   a. Put the relevant **data** and the **block address** to the bus.
   b. Pulse **NEW_DATA_INTERRUPT** signal.
   c. Snoop controller put the data into a que.
   d. Snoop controller put the data and address into broadcast_data and broadcast_address.
   e. Snoop Controller Pulse **BROADCAST_INTERRUPT** signal

2. ***Requesting a Memory Block from Snoop Controller - Request***
   a. Processor element puts the address of the data block into **ADDR_BUS.**
   b. Then the processor element Pulse **REQUEST_DATA_INTTERUPT.**
   c. Then the Snoop Controller put the request into a que.
   d. Then the Snoop Controller requests data from all other Processor Elements.
   e. Snoop Controller puts the address into **BROADCAST_ADDRESS_BUS.**
   f. And Pulse **INPUT_REQUEST_DATA_INTTERUPT.**
   g. Then if the cache controller of that PE has the data block it pulse **INPUT_DATA_AVAILABLE** signal.
   h. Otherwise it pulses **INPUT_DATA_NOT_AVAILABLE** signal.
   i. If Snoop Controller receives a positive response from any of the PEs. It pulses the OUTPUT_DATA_FOUND signal. Otherwise it pulses **OUTPUT_DATA_NOT_FOUND**

3. ***If the buffer full in snoop controller***
   a. Cache waits until the snoop controllers signal whether data is available or not.
   b. If the local cache does not receive a response from the snoop controller for a certain amount of time. The local cache sends the request to the snoop controller again.

Problems Encountered

1. How to handle the bus traffic and collections in data and address busses?
   - We have moved to a centralized system with Snoop Controller as the master and he is the one who coordinated the **DATA** and **ADDRESS** Busses.
2. Are we going for a synchronous system or asynchronous one?
   - We finally decided to move to an Interrupt based asynchronous system.

References

[1] https://en.wikipedia.org/wiki/MESI_protocol
[2] https://people.cs.pitt.edu/~melhem/courses/2410p/ch5-4.pdf