OpenCores.Org

MESI_ISC Specification Draft

Author: Yair Amitay

yair.amitay@yahoo.com

www.linkedin.com/in/yairamitay

Rev. 0.10

January 2013

This page has been intentionally left blank.

Revision History

Rev.	Date	Author	Description
0.10	1/3/2013	Yair Amitay	Draft, Under Construction

Contents

1 Introduction

The MESI InterSection Controller (ISC) is a coherence system controller. It supports the MESI coherence protocol. It synchronizes the memory requests of the system masters. It enables to keep the consistency of the data in the memory and in the local caches.

This project provides a synthesizable controller core and it defines the requirements of the system masters.

2

Coherency Systems

A coherency system is a system in which all the different copies of the same memory address are consistency. It means that if a master writes a certain data to a memory address then any other master that access this address read the update data. One of the cases that an inconsistency can occur is when the system masters have memory caches. If a data copy in the cache is not update according to the least written data the master that own this cache may read the wrong data.

A system is coherent if it obeys the all following rules:

Rule 1

Time	Master	Address	Write Data	Read Data
T1	M1	A1	D1	
T2	M1	A1		D1

T2 > T1

Rule 2

Time	Master	Address	Write Data	Read Data
T1	M1	A1	D1	
T2	M2	A1		D1

T2 > T1

Rule 3

Time	Master	Address	Write Data	Read Data	Comment
T1	M1	A1	D1		
T2	M2	A1	D2		
T3	M3	A1		D2	
T4	M3	A1		D2	It is not allowed to read D1 at this point

T4 > T3 > T2 > T1

Figure 1 describes an example of a basic system. It contains a main memory and three masters. The masters are connected to the main memory and all of them can access the memory through the arbiter. A write action of a master has three stages. First the master performs a read access to the main memory and the requested memory data is copied to the master`s cache. Then the local copy of the data in the cache is updated with the write data. Later, the master may evict the cache line that contains the update data and write it back to the memory. Data inconsistent can occur in several cases. The following scenario describes an example of data inconsistent:

1. Master 1 write data D1 to address A1
    a. Master 1 performs a read access to A1 in the main memory.
    b. The data of A1 in the main memory, D0, is copied to the local cache of master 1.
    c. The data of A1 in the local cache is overwritten with the written data, D1.
2. Master 2 read from address A1
    a. Master 2 performs a read access to A1 in the main memory.
    b. The data of A1 in the main memory, D0, is copied to the local cache of master 2.

The data of A1 in the memory in not update and Master 2 read the wrong data.

In a basic system data inconsistent can be prevented by software synchronization between the masters.

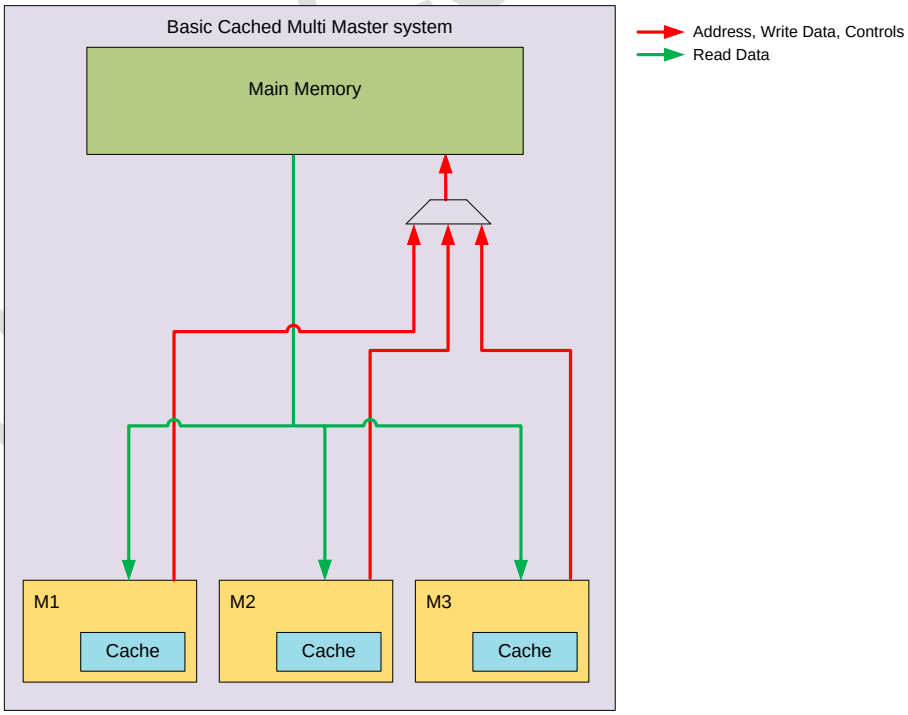

Figure 1: Basic System

Figure 2 describes an example of a hardware coherency system. It is similar to the basic system with some changes. Each master monitors (snoops) the actions of all other master. A master can postpones main memory accesses of the other masters. A master postpones a memory access of other master when it has a data copy of a certain memory address in its cache and the other master tries to access this memory location. It evicts the cache line that contains the certain data and writes it back to the memory. Then it enables the other master to continue and access the memory. The following scenario describes an example which preventing data inconsistent:

1. Master 1 write data D1 to address A1
    a. Master 1 performs a read access to A1 in the main memory.
    b. The data of A1 in the main memory, D0, is copied to the local cache of master 1.
    c. The data of A1 in the local cache is overwritten with the written data, D1.
2. Master 2 read from address A1
    a. Master 2 starts to perform a read access to A1 in the main memory.
    b. Master 1 detects this access and holds it.
    c. Master 1 evicts the cache line that contains the data copy of A1. This line contains data D1.
    d. Data D1 is written to the main memory to address A1.
    e. Master 1 releases the memory access of master 2 and lets it continue.
    f. Master 2 finishes performing the read access from A1 in the main memory and read the data D1.

In a coherency system data inconsistent is prevented by the hardware.

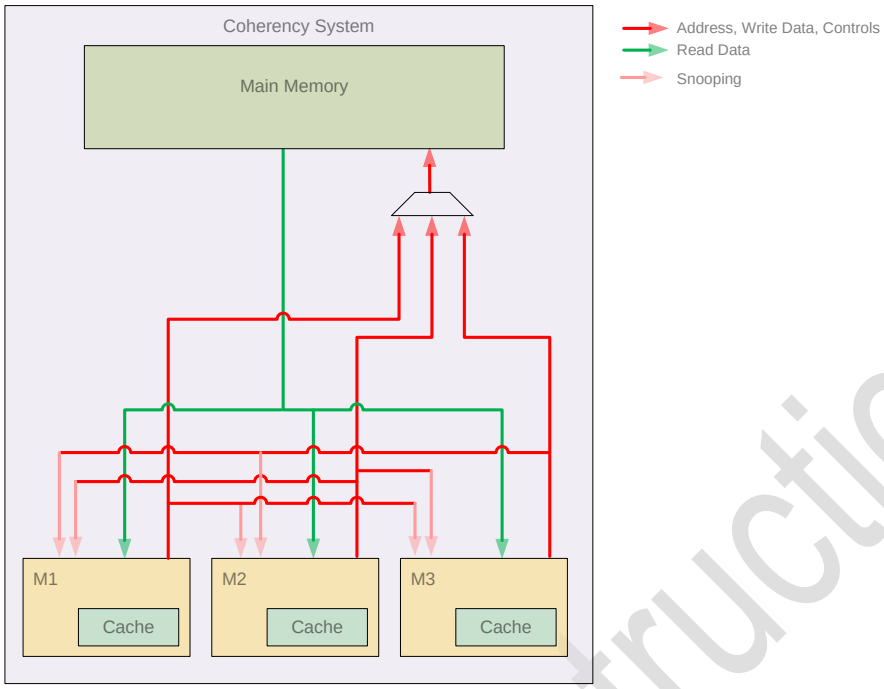

Figure 2: Schematic Coherence System

Coherency protocol (Under Construction)

Snoop protocol

Write back

3

Project Purpose

A coherence system contains several elements that together enable the coherency. The major elements of the coherency mechanism are:

1. Coherency controller
2. Coherency masters
3. Coherency buses

This project provides a synthesizable coherency controller block. This block can be combined to exist system with some the additional changes. In addition this project defines the masters' behavior and requirements that enable a correct coherency operation.

4

MESI_ISC Coherency Concept

MESI Protocol (Under Construction)

The MESI protocol is a protocol of memory and cache coherency. According to the MESI protocol any cache can be in one of four states: **M**odified, **E**xclusive, **S**hared, and **I**nvalid.

The next table describes, for any given pair of caches, the permitted states of a given cache line:

	Cache A			
Cache B	**M**	**E**	**S**	**I**
M	X	X	X	V
E	X	X	X	V
S	X	X	V	V
I	V	V	V	V

Figure 3 describes the MESI state machine.

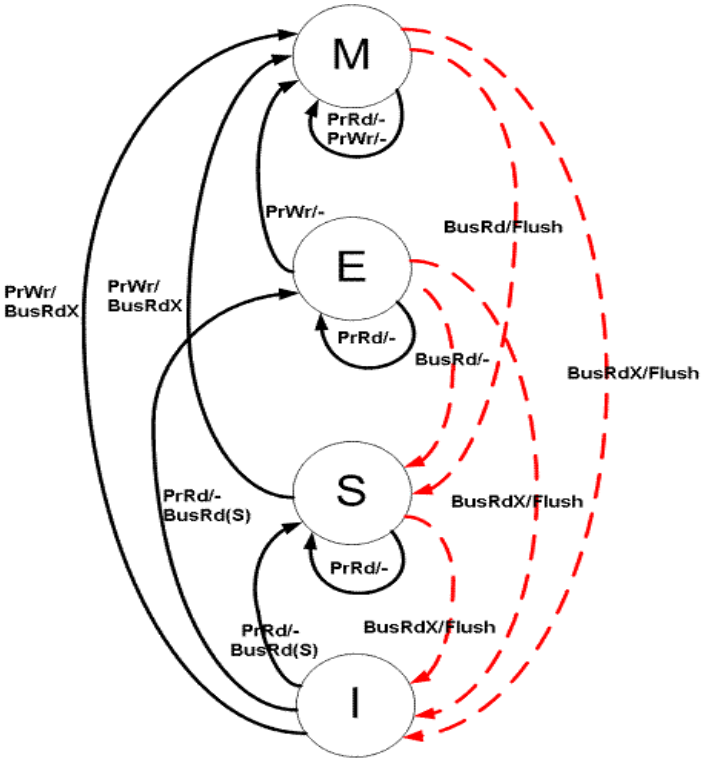

Figure 3: MESI State Machine (Wikipedia)

Coherency Protocol

In a basic system any master has a bus port for performing the memory read accesses and writes accesses. In coherency system the masters have additional port of coherency bus. Both the main bus and the coherency bus are used for the coherence protocol as describes below. The broadcast request enables preventing data conflicts.

The transactions of the main bus are initiated and drive by the masters. They respond by the main memory, the system matrix or the coherency controller. The transactions that are done in the main bus are:

1. Write access – A write access to the memory (legacy bus transaction).
2. Read access – A read access to the memory (legacy bus transaction).
3. Write broadcast – A write broadcast request. Asks for all other master to evict and invalidate data of the requested address. This transaction type is unique for coherency systems.

4. Read broadcast – A read broadcast request. Asks for all other master to evict modified data of the requested address. This transaction type is unique for coherency systems.

The coherency bus is unique for coherency systems. Its transactions are initiated and drive by the coherency controller. They respond by the masters. The transactions that are done in the coherency bus are:

1. Write snoop – Another master request to write to a requested memory location.
2. Read snoop – Another master request to read to a requested memory location.
3. Enable write – A respond to a write broadcast (which was performed in the main bus). It means that the write to the requested memory location can be done.
4. Enable read – A respond to a read broadcast (which was performed in the main bus). It means that the read to the requested memory location can be done.

In general, a coherency operation starts when a master (initiator) generates an access the memory. Prior to any memory access the master sends a broadcast request in the main memory. The coherency controller spreads the request to all the other masters and collects the responds. Then it enable the initiator to perform the memory access. All the operation of the coherency controller are done in the coherency bus.

A coherency operation occurs when one of the caches in the system performs a read miss, a write miss, or a write to a Shared cache line. Write hit, read hit, line eviction, and line invalidate do not cause to a coherency operations.

The following tables describe in details all the stages for the coherency operations. In the tables the meanings of some expression are:

Source/destination: Initiator – A master which requests to perform one of the following a memory accesses: (1) A read access to a memory location that is not present in its cache, or (2) a write access to a memory location that is not present in its cache or present and has a shared state.

Source/destination: Coherency Controller – The element that responsibles for the broadcast management. In this project is the MESI_ISC.

Source/destination: Snooper – A master that receives a write or read snoop request.

Bus: Internal – An internal operation.

Coherency operation for a write miss

The following table describes the stages that are done for a write miss.

Stage	Source	Destination	Bus	Operation	Comments
1	Initiator	Coherency Controller	Main	Send write broadcast	
2	Coherency Controller	Initiator	Main	Acknowledge write broadcast request	When it receive the request
3	Coherency Controller	Snooper	Coherency	Write snoop	Done to all masters except the initiator
4	Snooper		Internal	Evicts a dirty line	In case the line is M state
				Cache state: E/S->I	In case the line is E or S states
				Do nothing	In case there is no a valid line
5	Snooper	Memory	Main	Write back line to memory Cache state: M->I	In case of eviction
6	Snooper	Coherency Controller	Coherency	Acknowledge write snoop	
7	Coherency Controller	Initiator	Coherency	Enable write	After all masters acknowledged the write snoop broadcast
8	Initiator	Memory	Main	Read line	Fill line
	Initiator		Internal	Cache state: I->S	For the cache line that contains the read data
9	Initiator		Internal	Write to the cache Cache state: S->M	For the cache line that contains the read data

Coherency operation for a read miss

The following table describes the stages that are done for a read miss.

Stage	Source	Destination	Bus	Operation	Comments
1	Initiator	Coherency Controller	Main	Send read broadcast	
2	Coherency Controller	Initiator	Main	Acknowledge read broadcast request	When it receive the request
3	Coherency Controller	Snooper	Coherency	Read snoop	Done to all masters except the initiator
4	Snooper		Internal	Evicts a dirty line	In case the line is M
				Cache state: E->S	In case the line is E
				Do nothing	In case the line is S or there is no a valid line
5	Snooper		Main	Write back line to memory Cache state: M->I	In case of eviction
6	Snooper	Coherency Controller	Coherency	Acknowledge read snoop	
7	Coherency Controller	Initiator	Coherency	Enable read	After all masters acknowledged the read snoop
8	Initiator	Memory	Main	Read line	Fill line
9	Initiator		Internal	Cache state: I->S	For the cache line that contains the read data

Coherency operation for a write to a Shared line

The following table describes the stages that are done for a write hit to a line in Shared state.

Stage	Source	Destination	Bus	Operation	Comments
1	Initiator	Coherency Controller	Main	Send write broadcast	
2	Coherency Controller	Initiator	Main	Acknowledge write broadcast request	
3	Coherency Controller	Snooper	Coherency	Write snoop	Done to all masters except the initiator
5	Snooper		Internal	Invalidates the valid line: Cache state: S->I	
6	Snooper	Coherency Controller	Coherency	Acknowledge write snoop	
7	Coherency Controller	Initiator	Coherency	Enable write	After all masters acknowledged the write snoop
8	Snooper		Internal	Write to the cache Cache state: S->M	For the cache line that contains the read data

Examples for Coherency Scenarios

Write miss to a an Invalid location

The following diagram describes a write miss of M0 to an address that is invalid in all masters. M0 sends write-broadcast. M1 and M2 receive write-snoop and return immediately acknowledge. M0 receives write-enable and writes to the memory (read A1 from memory and write to the cache).

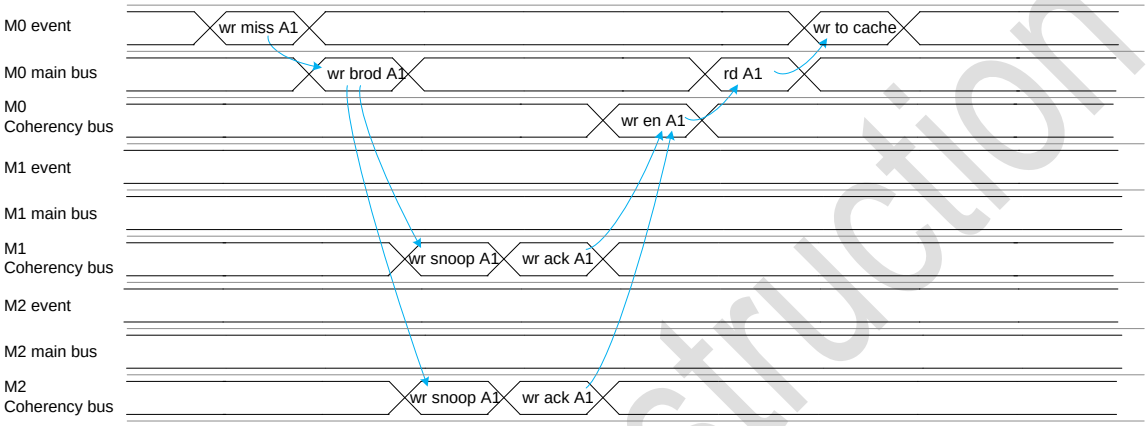

Figure 4: Write miss to a an Invalid location

Write miss to a Modified location in other master

The following diagram describes a write miss of M0 to an address that is modified in M1. M0 sends write-broadcast. M1 and M2 receive write-snoop. M2 returns immediately acknowledge. As a result of the write snoop, M1 evicts A1 and write it back to the memory. Then M1 returns acknowledge. M0 receives write-enable and writes to the memory (read A1 from memory and write to the cache).

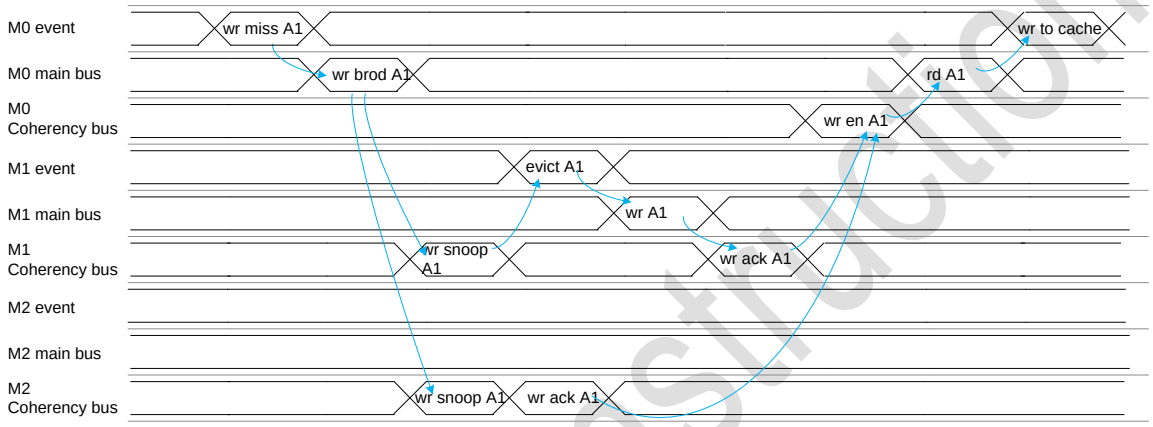

Figure 5: Write miss to a Modified location in other master

Two parallel write misses to an Invalid location

The following diagram describes a write miss of M0 to address A1 in parallel to a write miss of M1 to address A1. A1 is invalid in all caches. M0 and M1 send separately write-broadcasts. MESI_ISC respond first to M0. M1 and M2 receive write-snoop and return immediately acknowledge. M0 receives write-enable and writes to the memory (read A1 from memory and write to the cache). Then MESI_ISC respond the broadcast of M1.

M0 and M2 receive write-snoop. M2 returns immediately acknowledge. As a result of the write snoop, M0 evicts A1 and write it back to the memory. Then M0 returns acknowledge. M1 receives write-enable and writes to the memory (read A1 from memory and write to the cache).

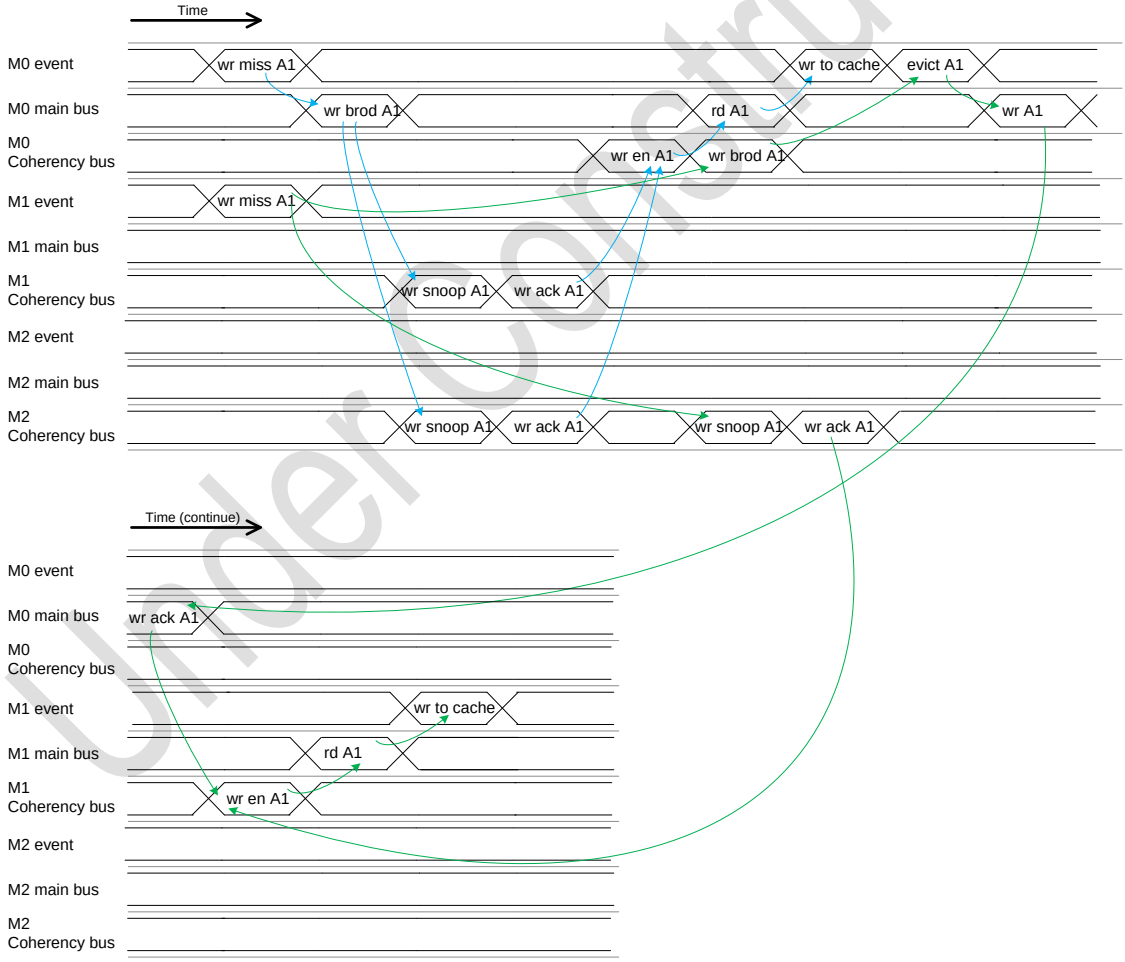

Figure 6: Two parallel write misses to an Invalid location

5 Architecture

Operation (Under Construction)

Figure 3 describe a coherence system with MESI_ISC

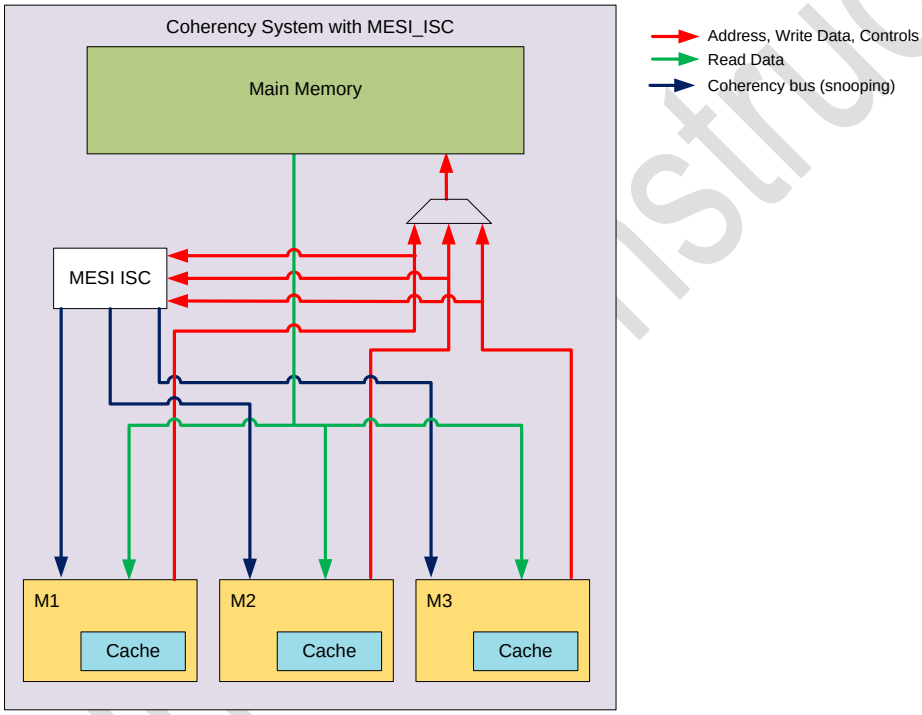

Figure 7: MESI_ISC Architecture and the Masters system

System Performance (Under Construction)

Clock and reset (Under Construction)

Masters definition and requirements (Under Construction)

Integration MESI_ISC to existing systems (Under Construction)

6

Micro Architecture

(Under Construction)

Figure 4 describes the micro-architecture of MESI_ISC

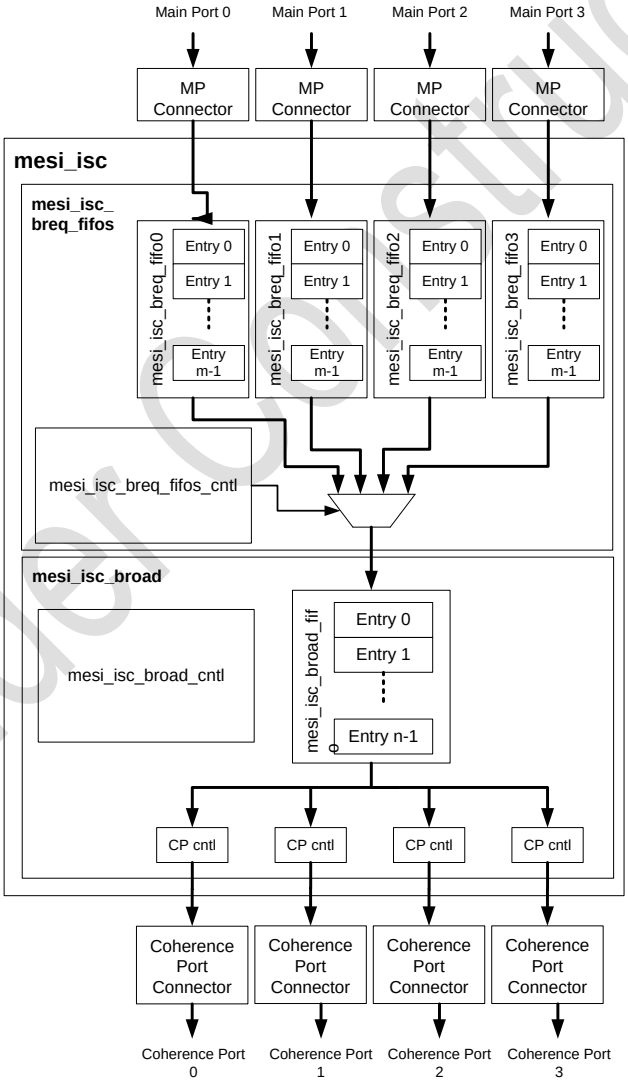

Figure 8: MESI_ISC Micro-Architecture

7

Verification

(Under Construction)

Validation of Data Consistency

Validation of MESI Protocol

Random Stimulus

Verification Environment (Under Construction)

Figure 5 describes the MESI_ISC verification environment.

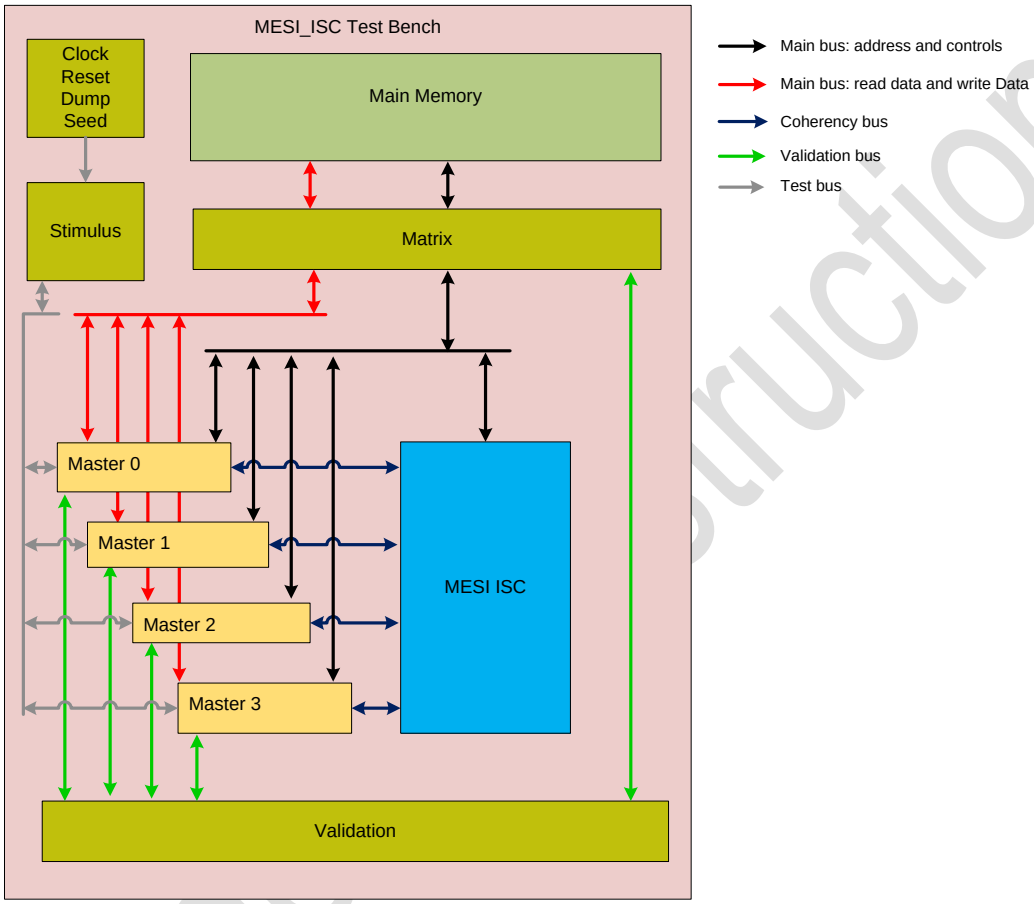

Figure 9: MESU_ISC Verification Environment

8

Timing, Power and Area

(Under Construction)

9

Design Environment

(Under Construction)

Tools

Synthesis

Simulation

Lint

10

IO Ports

(Under Construction)

This section specifies the MESI_ISC IO ports.

Clock and reset

Port	Direction	Description
clk	Input	
rst	Input	

Main bus

Port	Direction	Description
mbus_cmd3_i	Input	
mbus_cmd2_i	Input	
mbus_cmd1_i	Input	
mbus_cmd0_i	Input	
mbus_addr3_i	Input	
mbus_addr2_i	Input	
mbus_addr1_i	Input	
mbus_addr0_i	Input	
mbus_ack3_o	Output	
mbus_ack2_o	Output	
mbus_ack1_o	Output	
mbus_ack0_o	Output	

Coherency Bus

Port	Direction	Description
cbus_ack3_i	Input	
cbus_ack2_i	Input	
cbus_ack1_i	Input	
cbus_ack0_i	Input	
cbus_addr_o	Output	
cbus_cmd3_o	Output	
cbus_cmd2_o	Output	
cbus_cmd1_o	Output	
cbus_cmd0_o	Output	

11 Waveforms

(Under Construction)

12

Open Issues