IMPLEMENTATION OF MESI PROTOCOL USING VERILOG

A PROJECT REPORT

Submitted by

THARUNIKA K	710021106037
ANUSRI S	710021106318
SRI HARIHARAN B	710021106305

In partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

ANNA UNIVERSITY REGIONAL CAMPUS, COIMBATORE

ANNA UNIVERSITY: CHENNAI – 600 025

ANNA UNIVERSITY, CHENNAI

BONAFIDE CERTIFICATE

Project report titled "**Implementation of MESI Protocol Using Verilog**" is the bonafide work of "THARUNIKA K (710021106037), ANUSRI S (710021106318), SRIHARIHARAN B (710021106305)" who carried out the project work under my supervision

Dr. V.R.Vijaykumar M.E, Ph.D.,
Head of the Department,
Associate Professor,
Department of ECE,
Anna University Regional Campus,
Coimbatore – 641 046.

Ms . S. Manju M.E.,
Faculty of the department & mentor
Department of ECE,
Anna University Regional Campus,
Coimbatore – 641 046.

ACKNOWLEDGEMENT

First and foremost, we place this project work on the feet of **GOD ALMIGHTY** who is the power of strength in each step of progress towards the successful completion of the project.

We owe wholehearted sense of reverence and gratitude to our Guide

Ms. S. MANJU M.E., and Associate Professor and Head of the Department of ECE **Dr. V.R. VIJAYKUMAR M.E, Ph.D.,** for their efficient and excellent guidance, inspiring discussions insightful and timely encouragements for the successful completion of the project.

I thank **Dr. M. SARAVANAKUMAR MBA., Ph.D.,** Dean Regional Campus, Anna University Regional Campus, Coimbatore for his great support with blessings. I also extend my heartfelt thanks to all Faculties and staff of ECE department who have rendered their valuable help in making this project successful.

THARUNIKA K	**710021106037**
ANUSRI S	**710021106318**
SRI HARIHARAN B	**710021106305**

ABSTRACT

Multiprocessor system is a system which contains two or more processors working simultaneously and sharing the same memory. Nowadays multiprocessors are being widely used due to their high throughput and reliability. It is important to maintain data consistency in multiprocessor system as different processors may communicate and share the data with each other. In multiprocessor systems caching plays a very important role.

Cache coherence is a major issue in multiprocessor systems. In this paper we have designed three direct-mapped caches and to maintain cache coherence and data consistency among the processors we have used the MESI protocol. The MESI protocol is invalidation-based cache coherence protocol. In this protocol each cache block can be in one of four states i.e., Modified, Exclusive, Shared and Invalid.

In this protocol, whenever a processor writes into the local cache, all copies of it in other processors are invalidated in order to maintain data consistency and cache coherence. The cache design is simulated and synthesized using Xilinx ISE 14.7 Simulator and XST Synthesizer.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

1.	PRRD	PROCESSOR READ
2.	PRWR	PROCESSOR WRITE
3.	BUSRD	BUS READ
4.	BUSUPGR	BUS UPGRADE
5.	MEM_RD	MEMORY READ
6.	MEM_WR	MEMORY WRITE
7.	MEM_CS	MEMORY CHIP SELECT

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION

In recent years multiprocessors are gaining more importance as they have better performance and reliability than single processor systems. Multiprocessors with shared memory are being used in the today's computers and researches . Using the single address space the processors can communicate among themselves because address space is shared among the processors in multiprocessor systems. So, same cache entry exists in other processors as the address is being shared. The shared memory multiprocessor system architecture is shown in the Fig.1.Sharing of data among the processors is not a problem during reading operation but it is a serious problem during write operation. When one processor writes a value to a location that is being shared, the changed value has to be updated to all caches otherwise the processors hold different data for the same location which is called as cache coherence problem. Consider three processors named P1, P2, P3 sharing a same memory . The processor P1 wants to read a value at location X. It reads the value from main memory and caches its value into P1. The processor P2 wants to read a value at location X.

1.2 OVERVIEW

In multiprocessor system cache coherence problem may occur when there is no proper coordination between the caches of each Processor. data inconsistency may occur when there is no proper synchronization between caches. This leads to cache coherence problem and this is tackled by cache coherence protocols. To maintain the data consistency a set of rules are implemented in the system called as cache coherence protocols.

1.3 BLOCK DIAGRAM

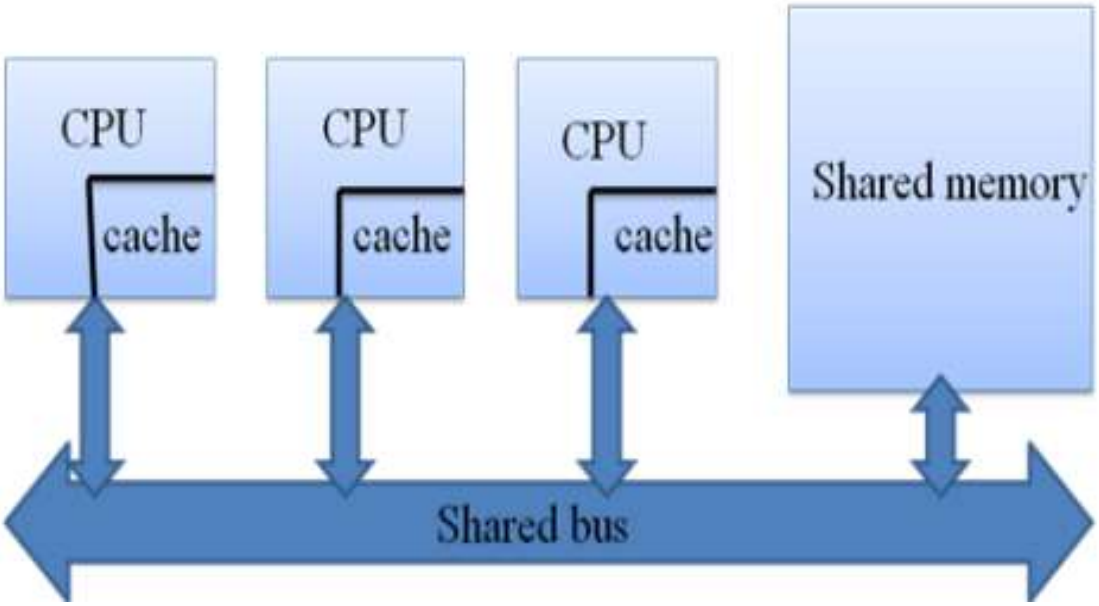

FIG 1.SHARED MEMORY MULTIPROCESSOR SYSTEM

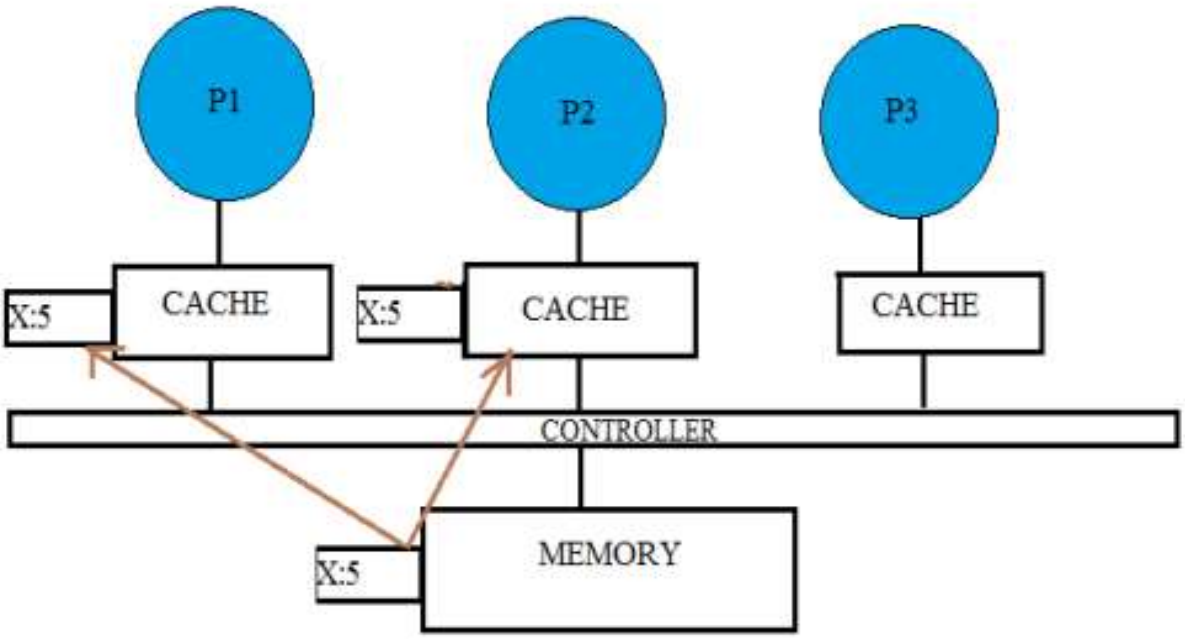

FIG 2.P1, P2 READING VALUE FROM MEMORY

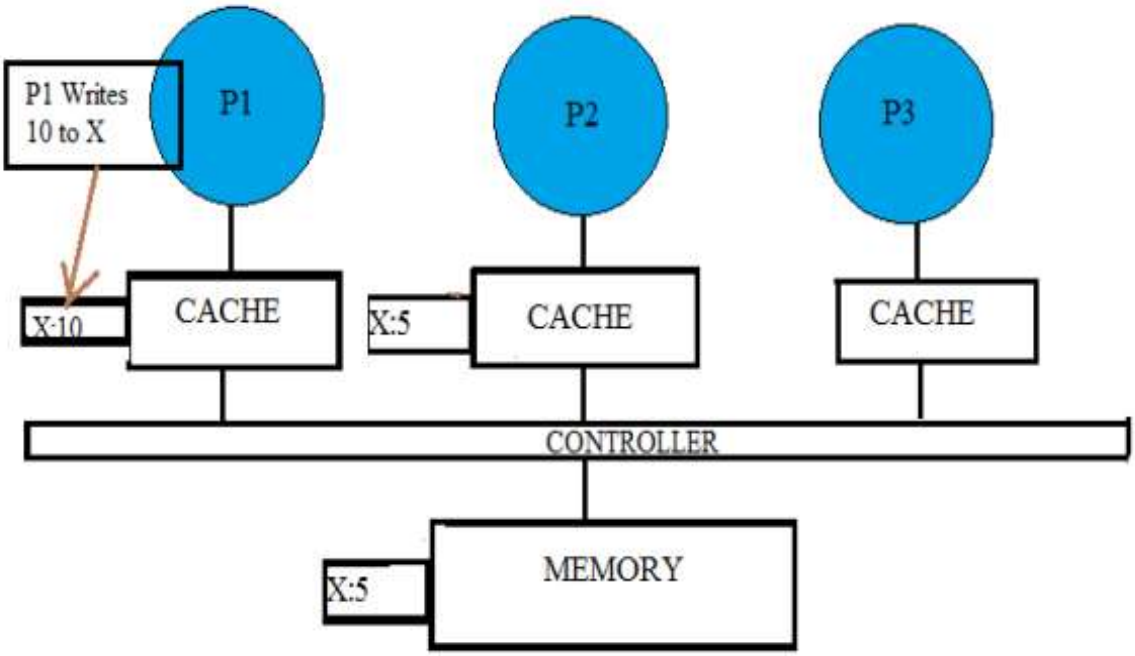

FIG 3.P1 PERFORMING WRITE OPERATION IN LOCATION X

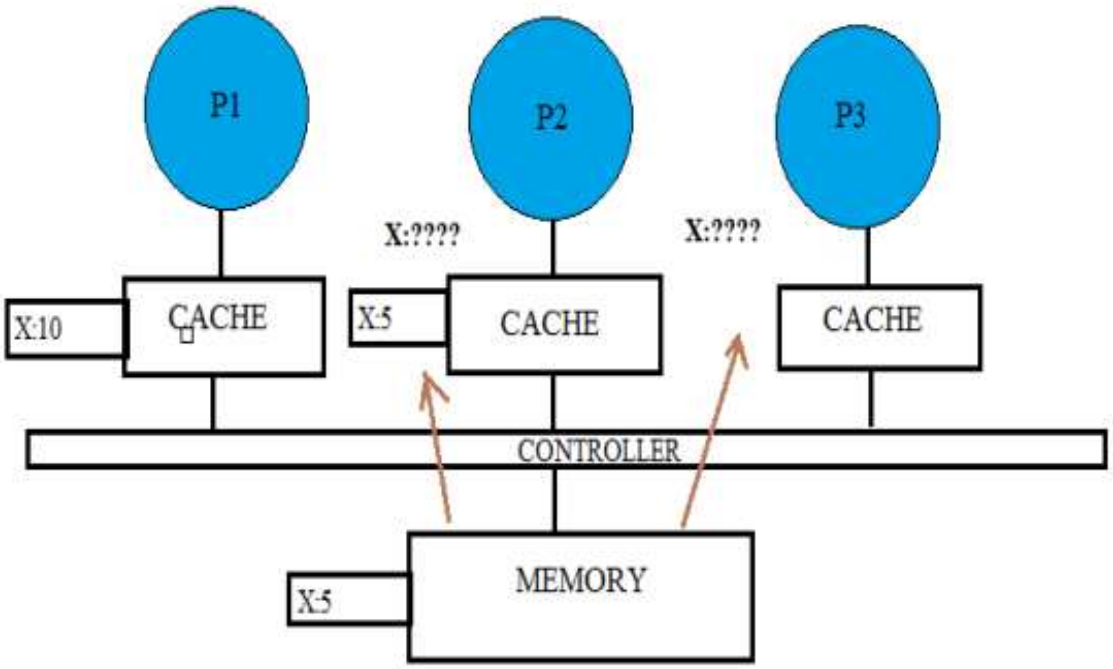

FIG 4.DATA INCONSISTENCY IN MULTIPROCESSOR SYSTEM

1.4 DESCRIPTION

Consider three processors named P1, P2, P3 sharing a same memory [3]. The processor P1 wants to read a value at location X. It reads the value from main memory and caches its value into P1. The processor P2 wants to read a value at location X.

It reads the value from main memory and caches its value into P2as shown in Fig.2. The processor P1 wants to write a value to location X. This is shown in Fig.3. Now if processor P3 performs read operation on location X. In p1 at location X, value stored is 10 and in p2 at location X, the data stored is 5, therefore data inconsistency arises when we perform write operation to a shared address location. This is shown in Fig.4.

The main objective is to overcome the data inconsistency and cache non-coherence in multiprocessor systems. To obtain data consistency and cache coherence in multiprocessor systems, MESI Protocol [4-6] is implemented in this thesis work. In MESI protocol, whenever a write operation to a shared location is performed, an invalidate signal is issued by the cache controller to all those caches containing same address location as shown in the Fig.5, so that no stale data is read by the processors

CHAPTER 2
LITERATURE SURVEY

2.1 INTRODUCTION

In multiprocessor system cache coherence problem may occur when there is no proper coordination between the caches of each Processor. data inconsistency may occur when there is no proper synchronization between caches. This leads to cache coherence problem and this is tackled by cache coherence protocols. To maintain the data consistency a set of rules are implemented in the system called as cache coherence protocols. In this paper one of the three snoopy based cache coherence protocol MESI is implemented by referring various literatures and research papers.

2.2 PREVIOUS WORK

Zainab, Michael [2] describes only two snoopy based cache coherence Protocols. i.e., MSI, MESI. Kaushik Roy, Pavan Kumar S.R[3] have done comparative studies of cache coherence protocols but they have not implemented cache coherence protocols. Neethu, Geeta[4] have studied simulation based performance study of cache coherence protocols in Gem-5 Simulator. Sravanthi, Rajashekara[5] have implemented MESI Protocol. Daman, Sulochana[6] implemented cache coherence protocols in Multiprocessor systems. Liu, Hao [7] proposed a new type of L2 cache in order to combat the high power usage of the conventional L2 cache. Suamher[8] described the snoopy and directory based cache coherence protocols. Amit D.Joshi[9]acknowledged snooping Cache Coherence Protocols are suitable for multiprocessor architectures. Mays K. Faeq, Safa [10] have implemented only MSI Protocol in the multiprocessor system. Danko, Sinisa[11] have done Time domain performance evaluation of adaptive hybrid cache Coherence Protocol. Li, Sizhao [12] work explored the cache access patterns of a GPU by running various benchmarks on both NVIDIA and AMD architectures, and concluded GPU handles cache coherence much better than CPUs. Ibrahim A.Amory, Ahmed H.Ahmed [13] have implemented only MESI Protocol. Bijo, Shiji [18] Discussed parallel execution on multicore architectures with multilevel caches. This Literature survey motivated me to design of snoopy busbased cache coherence protocol for multiprocessor architectures.

2.3 PROPOSED WORK

In multiprocessor systems, each processor has its own local cache. These caches can have different values for the same address in main memory depending on when the data was read from memory. For example, if two processors load the same value from memory and both modify it before storing it back to the memory, one of the values will be overwritten and the other value is lost. This problem is known as cache incoherence. There are multiple different protocols that attempt to keep coherency between the caches so when one processor modifies a value, the change is propagated through the rest of the system without wasting time for accessing the main memory.

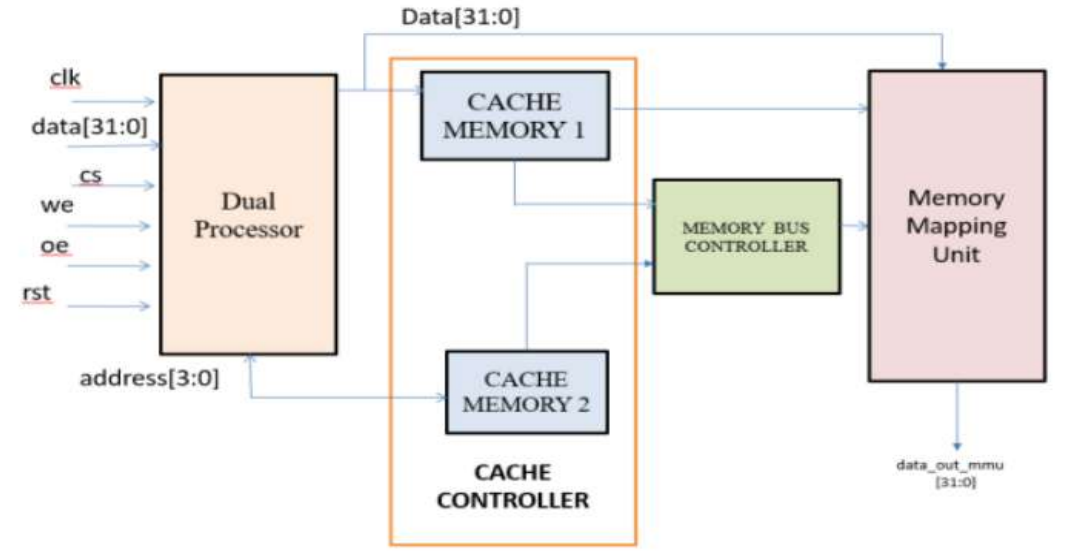

FIG5.PROPOSED SYSTEM FOR CACHE COHERENCE IN MULTIPROCESSOR SYSTEM

In figure 5 the chip selects(cs), write enable(we), clk, rst, output enable (oe), data, address are the input signals applied to the processor. When the write enable is high the data and address is written to cache memory 1 and cache memory2 respectively. Cache memory1 is specifically designed for the purpose of storing the data. Sometimes, processor needs to perform operation immediately. So, in that particular situation, if the processor looks for the data and task address in main memory, it will take more time for performing the

operation. To solve this problem, the data output from the processor will be applied as input to cache memory. It will store the immediate version of data inputs, so the processor can perform the task very immediately. Thus, in order to perform the parallel operation, all the address inputs given by the user must be stored. For this purpose, the input addresses are going to store into the Cache memory2 for faster retrieval of data. Generally, the results synchronization issues across the processor, to perform the Task based on the Cache-1 data output to the cache 2 address output. Thus, to maintain the synchronization between the two cache memories, the memory bus controller mechanism will be useful. The direct cache memory 1 output and memory bus controller output will be applied as the input to the memory mapping unit. Generally, the memory mapping unit is consisting of the real time processor, which consisting of multi-level memory to perform the various tasks. Thus, here the memory mapping unit is used to perform the selection of cache memory 1 output. If any interrupt generated due to the cache coherence protocol, then according to the interrupts, it will select the data and results the outputs as the final data out.

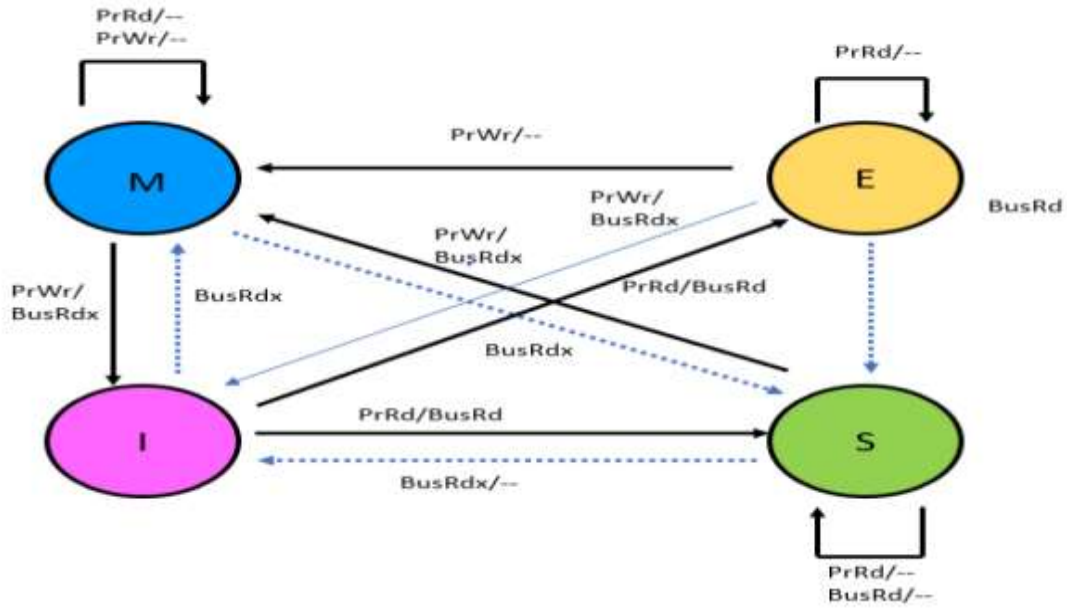

FIG 6 . MESI STATE TRANSITION DIAGRAM

The advantage of MESI Protocol is the extra state added called as exclusive state which reduces the bus transitions when the read miss occurs in the cache. MESI protocol is faster when compared to MSI particularly when working on the serial application by the processor. No cost change will be there since MSI and MESI states are encoded with 2 bits

2.3.1 DESIGN OF PROTOCOLS

Cache coherence protocols are designed in Verilog HDL and simulated in Xilinx. The various input signals are Processor read (PrRd), Processor Write(PrWr), Bus Read(BusRd), Bus upgrade(BusUpgr) determines the Cache states and also Output signals Memory Read(mem_rd), Memory write(mem_wr), Memory chip select(mem_cs) determines the possible operations that can be performed on the cache states.

2.3.2 IMPLEMENTATION OF PROTOCOLS

MESI Cache Coherence Protocols are Implemented in the Proposed System as shown. The modules of Proposed System are designed in Xilinx.

Test benches are designed to verify the functionality of the Protocol in the Proposed System of Cache Coherence Multiprocessor System. Each Module is instantiated in the top-level module by calling each Module in the top-level module according to their functionality. Here in the Proposed System clk, address, data, cs (chip select), we (write enable), oe(output enable) are the input signals, remaining all are output signals. Data out processor should be monitored with respect to the address input. Initially, when we=1 then all the data input will be stored into RAM. Then memory-based error checking operation performs using rollbacks. Finally, ram_oe will be activated. Then Data out of processor signal will generates. Data out of cache should be monitored with respect to the address out of cache. After error correction, cam1_oe becomes 1, then data out will be generated. The valid_f flag becomes active high, whenever error occurred, it will be auto corrected by using the proposed system.

CHAPTER 3

SOFTWARE DESCRIPTION

3.1 INTRODUCTION

Xilinx ISE (Integrated Synthesis Environment) is a discontinued software tool from Xilinx for synthesis and analysis of HDL designs, which primarily targets development of embedded firmware for Xilinx FPGA and CPLD integrated circuit (IC) product families. It was succeeded by Xilinx Vivado. Use of the last released edition from October 2013 continues for in-system programming of legacy hardware designs containing older FPGAs and CPLDs otherwise orphaned by the replacement design tool, Vivado Design Suite.

3.2 XILINX ISE

Xilinx integrated synthesis environment or ISE is a software tool got from Xilinx meant for the synthesis, as well as analysis of the HDL designs, which are known to majorly target embedded firmware development for Xilinx CPLD and FPGA integrated circuit product families.

Also, Xilinx Vivado succeeds it. The use of the edition released last from 2013 goes on for the in-system programming of the design of legacy hardware that contains older CPLDs and FPGA, which the Vivado Design Suite orphaned, which is a replacement design tool.With ISE, the developer will be able to synthesize their timing analysis, designs, examine the RTL diagrams, which stimulate the reaction of a design to many different stimuli, as well as configure this target using the programmer. Also, as practiced in the automation sector, the Xilinx ISE is well coupled to the own chips of the Xilinx (which are the internals that are very proprietary, thus cannot be utilized with the FPGA products gotten from the other vendors.

Due to the very proprietary nature that the product lines of Xilinx offer, it is very rare to make use of open-source options to tooling directly offered from Xilinx.

ISE’s main user interface is known as the project navigator. This includes the sources (also known as the hierarchy of the design), workplace (editor of the source code), processes (processes tree), and Transcript (output console).

This design hierarchy is made up of some modules or design files, with ISE interpreting their dependencies. It also appears as a kind of tree structure. For

the single-chip designs, there could be one major module, with this module including other modules, which are just like the main subroutine present in the C++ programs. Also specified in the modules are mapping and pin configuration.

Also, the hierarchy of the processes explains how the ISE operations will perform or function on the active module. This hierarchy includes the compilation functions, dependency functions, as well as other utilities. Also, the window denotes errors or issues which occur with every function.

For the transcript window, it gives the status of the running operations. It also informs the engineers on possible design issues. These issues could be filtered in order to show errors, warnings, or both.

Testing at the system level might be performed with the help of the ModelSim logic or ISIM simulator. These types of test programs have to be written in the HDL languages. Also, the programs of the test bench might include monitors or waveforms of simulated input signals that verify as well as observe the outputs of that device under the test.

Also, the ISIM or ModelSIM might be used in performing these simulations below:

- Behavioral verification: This helps in verifying the timing and logical issues
- Logical verification: This ensures the expected results of the module practices
- Route and post-place simulation: This helps in verifying the behavior after the module has been placed within the FPGA's reconfigurable logic.

3.3 CONCLUSION

Here's all we can say about the Xilinx ISE design suite. This is a software tool for the synthesis, as well as analysis of the HDL designs, which are known to majorly target embedded firmware development for Xilinx CPLD and FPGA integrated circuit product families.

CHAPTER 4
SOURCE CODE

4.1 TESTBENCH CODE

```
// Code your testbench here
// or browse Examples
typedef enum bit {CACHE,MEM}type_t;
module cache_tb;

  reg clk;
  reg reset;

  bit value;
  logic [2:0] current_state;
  logic [2:0] state_rand;

  // Processor interface
  reg [31:0] addr;
  reg read;
  reg write;
  wire [31:0] data;
  wire hit;

  // Memory interface
  wire [31:0] mem_addr;
  wire mem_read;
```

```
logic mem_write;
reg [31:0] mem_data;
type_t typ;

cache cache_inst (
  .clk(clk),
  .reset(reset),
  .addr(addr),
  .read(read),
  .write(write),
  .data(data),
  .hit(hit),
  .mem_addr(mem_addr),
  .mem_read(mem_read),
  .mem_write(mem_write),
  .current_state(current_state),
  .state_rand(state_rand),
  .mem_data(mem_data)
);

initial begin
  clk = 0;
  reset = 1;
  #10
  reset = 0;
  repeat(1)begin
  //value = $random;
```

```
   value = 0;
 // state_rand = $random;
//typ = value;
//void'(packet.randomize(typ));
case(typ==value)
CACHE: begin
  $display ("Starting CACHE function");
  // Write to cache
addr = 32'h1000;
write = 0;
mem_data = 32'hdeadbeef;
#100
write = 0;

// Read from cache
addr = 32'h1000;
read = 1;
#10
read = 0;
end

MEM: begin
$display ("Starting MEM function");
// Write to memory
addr = 32'h2000;
write = 0;
mem_data = 32'hcafebabe;
```

```
  #100
  write = 0;

  // Read from memory
  addr = 32'h2000;
  read = 1;
  #10
  read = 0;
  end
  endcase
 #50;
  end
     // Invalidate cache line
  //addr <= 32'h1000;
  //mem_write <= 1;
 // #10
  //mem_write <= 0;
// Read from cache (should miss)
  //addr = 32'h1000;
  //read = 1;
  //#10
  //read = 0;
    end
 initial begin
   $dumpfile("dump.vcd");
    $dumpvars;
      #2000
```

```
 $finish;
  end
 always #5 clk <= ~clk;
endmodule
```

4.2 DESIGN CODE

```
typedef enum bit[2:0] {INVALID = 3'b000,
                       SHARED = 3'b001,
                       EXCLUSIVE = 3'b010,
                       Cache_Fill = 3'b100,
                      MODIFIED = 3'b011}state_t;
module cache (
  input clk,
  input reset,
  input [2:0] state_rand,

  // Processor interface
  input [31:0] addr,
  input read,
  input write,
  output reg[31:0] data,
  output hit,

  // Memory interface
  output [31:0] mem_addr,
```

```
  output mem_read,
  output mem_write,
  output reg [2:0] current_state,

  input [31:0] mem_data
);
  state_t state;
   // Cache tags and data
  reg [31:0] tags [1023:0];
  //reg [31:0] c_data [1023:0];
  //reg [2:0] state;
  // Cache controller
  reg [2:0] next_state;

  always @(posedge clk) begin
    if (reset) begin
      next_state <= INVALID;
    end else begin
      current_state = next_state;
      //state = state_rand;
      $display ("execute = %h",state[addr]);
      //case (!state)
      case (state== next_state)
      //case (state== state_rand)
        INVALID: begin
          $display ("executing invalid block");
          if (read) begin
```

```
    next_state <= Cache_Fill;
  end else if (write) begin
    $display ("moving to next block");
    next_state <= MODIFIED;
      end
end
SHARED: begin
  $display ("executing shared block");
  if (read) begin
    next_state <= SHARED;
  end else if (write) begin
    next_state <= MODIFIED;
    // Invalidate other caches
  end
end
EXCLUSIVE: begin
  $display ("executing exclusive block");
  if (read) begin
    next_state <= EXCLUSIVE;
  end else if (write) begin
    next_state <= MODIFIED;
  end
end
MODIFIED: begin
  $display ("executing modified block");
  if (read) begin
    next_state <= MODIFIED;
```

```
        end else if (write) begin
          next_state <= MODIFIED;
        end
      end
    endcase
  end
 end
// Cache hit
 assign hit = (state[addr] != INVALID);

 // Memory operations
 assign mem_addr = addr;
 assign mem_read = (next_state == Cache_Fill);
 assign mem_write = (state[addr] == MODIFIED) && (next_state != Cache_Fill);
// Cache writeback
 always @(posedge clk) begin
  if (mem_write) begin
    data[addr] <= mem_data;
  end
 end
// Snooping
 reg [31:0] snoop_addr;
 reg snoop_read;
 reg snoop_write;
 always @(posedge clk) begin
  snoop_addr <= addr;
  snoop_read <= read;
```

```
    snoop_write <= write;
  end
 // Invalidate other caches
  always @(posedge clk) begin
    if (snoop_write && (state[snoop_addr] != INVALID)) begin
      state[snoop_addr] <= INVALID;
    end
  end
endmodule
```

CHAPTER 5

EXPERIMENTAL RESULTS AND ANALYSIS

5.1 EXPERIMENTAL RESULTS

FIG 7.SIMULATION WAVEFORM OF STATE TRANSITION OF MESI PROTOCOL

FIG 8.IMPLEMENTATION OF MESI PROTOCOL

CHAPTER 6

CONCLUSION & FUTURE WORK

6.1 CONCLUSION

In recent years, multiprocessor systems are being widely used. Multiprocessor systems are highly reliable and economic when compared to uniprocessor systems. Sharing of data among the processors is not a problem during the read operation but it is a serious problem during write operation. When one processor writes a value to a location that is being shared, the changed value has to be updated to all caches otherwise, the processors hold different data for the same address. The major issue in multiprocessor system is data inconsistency and cache non-coherence. To overcome these drawbacks a MESI cache coherence protocol is implemented in this paper. The MESI is a four state, invalidation based protocol. We have designed three caches and have implemented MESI Protocol, to achieve cache coherence and data consistency.

6.2 FUTURE WORK

The following are few of the future scopes

• As I have tested the MESI protocol, MOESI and MOESIF protocols can also be tested by adding the additional owned and forward cases. A comparison can be done amongst the 3 different protocols.

• The trace files only consisted of commands and address values. Changes can be made to add actual data values and then check how it effect the hit ratio, when data values of different sizes are included.

• LRU was used in this project, MRU and random replacement policies can be used and the hit ratio can be compared.

• For this project a 32 bit processor was considered, changes can be made to test for a 64 bit processor and change the cache size and sets accordingly

REFERENCES

[1] Mays K. Faeq, Safaa S.omran. “ MSI Protocol for Multicore Processors Based on FPGA.” International Journal of Engineering Science Invention (IJESI) 2020

[2]. Agarwal, Sukarn, and Hemangees K. Kapoor. "LiNoVo: Longevity Enhancement of Non-Volatile Last Level Caches in Chip Multiprocessors." 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). IEEE, 2020.

[3] Kabadi, Dr, and G. Mohan. "A Comprehensive Study on Design Consideration of Multi Core Processors." (2020).

[4] R. Rodrigues, I. Koren, and S. Kundu, “A mechanism to verify cache coherence transactions in multicore systems,” in 2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Oct 2012.

[5] Kaushik, M. Hassan and H. Patel “Designing Predictable Cache Coherence Protocols for Multi-Core Real-Time Systems.” IEEE Transactions on Computers, 2020.

[6] Bijo, Shiji, et al. "A formal model of parallel execution on multicore architectures with multilevel caches." International Conference on Formal Aspects of Component Software. Springer, Cham, 2017.