Portland State UNIVERSITY

High-Level Design Specification (HLDS)

for

Multi-Processor System with Memory-Controller and Shared Bus

ECE-593:
Fundamentals of Pre-Silicon Validation

Prepared by,
Viraj Pashte
Omkar Jadhav
Zoheb Mir
Vamsi Krishna Masetty
Tanmay Mhetre

31st Jan, 2024

Version 1.0

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

The primary purpose of this high-level specification document is to establish a clear, comprehensive framework for the project. It serves to clarify the project's objectives and align them with broader organizational or research goals. This document functions as a crucial communication tool, ensuring all stakeholders, including designers, developers, testers, and end-users, share a common understanding of the project's goals and methodologies. It guides the design and development process, setting clear guidelines and expectations for the system's architecture, including its shared-memory architecture and processor interconnectivity. Additionally, the document is instrumental in risk management, helping identify potential challenges early on and developing strategies to mitigate them. It also sets the stage for the verification and validation process by defining expected performance metrics and operational criteria. Lastly, this document serves as a foundational reference for future enhancements, providing detailed information on the system's design and capabilities for ongoing development and upgrades.

1.2 Document Conventions

A High-Level Design Specification (HLDS) for a Multi-Processor System with Memory-Controller and Shared Bus system typically follows specific document conventions to ensure clarity, consistency, and effective communication of the system's design.

Here are some document conventions commonly used in HLDS for Multi-Processor System:
1. Provided information on the document version by nomenclature R0,R1,R2 etc and include a revision history table at last that outlines changes made in each version.
2. Include a comprehensive table of contents to facilitate easy navigation.
3.Provide a high-level architectural overview of the Multi-Processor system, including key modules, components, and their relationships. Use diagrams, such as block diagrams, to visually represent the system architecture.
4. Included data flow diagrams to illustrate how data moves through the system. Show how information enters the system, is processed, and exits according to the Multi-Processor principle.
5. Detail the interfaces between different modules or components of the FIFO system.
6. Specified documentation standards or conventions followed in the HLDS.

- Italicized naming for variables and parameters that are used in code.
- A code sample with the name underlined is provided.
- Red ink is used to deliver error notices.
- Different operation flags are created with the use of highlighted color marks.
- Different blocks employ different font sizes and types.

1.3 Intended Audience and Reading Suggestions

This document is made for professionals who design, build, and test digital systems. It's important for this group of readers to carefully follow the document because it's structured to lead them step by step through the process of understanding. In this approach, readers get a complete picture of how the Multi-Processor system design should be implemented and integrated into larger systems by starting with the basics and moving on to more complex details.

1.4 Product Scope

The product scope of our multi-processor system, encompassing a memory-controller and a shared bus, is designed to forge a high-performance computing architecture, ideal for integration into larger systems like System on Chips (SoCs), CPUs, and network chips. This design focuses on interconnecting multiple processors through a shared bus, facilitating streamlined data communication and processing, complemented by a centralized memory-controller to efficiently manage shared memory access. Such an arrangement promises significant benefits like improved data throughput and reduced latency, essential in multi-processing environments. The primary objective is to craft a scalable and adaptable architecture, enhancing the capabilities of SoCs in handling complex tasks, boosting processing power and speed in CPU designs, and elevating data handling in network chips for improved network efficiency. This multi-processor system is not just about achieving speed and efficiency; it's about creating a versatile backbone for advanced computational and networking devices, responding to the increasing demands for rapid data processing and effective memory management across various technology sectors.

1.5 References

- http://csg.csail.mit.edu/6.884/projects/group6-presentation.pdf
- https://www.sciencedirect.com/science/article/abs/pii/B9780124201583000022
- https://www.geeksforgeeks.org/multiport-memory-multiprocessor-system/
- A. Zitouni, M. Abid and R. Tourki, "Arbitration schemes synthesis approach for multiprocessor systems," 1998 IEEE Workshop on Signal Processing Systems. SIPS 98. Design and Implementation (Cat. No.98TH8374), Cambridge, MA, USA, 1998, pp. 499-508, doi: 10.1109/SIPS.1998.715812. keywords: {Multiprocessing systems;Hardware;Application specific integrated circuits;Field programmable gate arrays;Costs}
- S. M. Mahmud and M. Showkat-Ul-Alam, "A new arbitration circuit for synchronous multiple bus multiprocessor systems," Systems Integration '90. Proceedings of the First International Conference on Systems Integration, Morristown, NJ, USA, 1990, pp. 57-62, doi: 10.1109/ICSI.1990.138662. keywords: {Multiprocessing system.

2. Overall Description

2.1 Product Perspective

The Multi-Processor System is conceived as a key component in advanced computing architectures, notably in Systems on Chips (SoCs), CPUs, and network chips. Its design, featuring a shared bus and a centralized memory-controller, is tailored to significantly elevate processing efficiency and data handling capabilities in these larger systems. The shared bus architecture ensures seamless and rapid communication between multiple processors, while the memory-controller optimizes memory access and utilization. This system is not just a standalone unit but a strategic enhancement to existing and future computing technologies, aimed at addressing the increasing demands for high-speed data processing and efficient memory management. Its adaptability and scalability make it a versatile solution, capable of being integrated into various complex computing environments, thereby promising a substantial improvement in overall system performance and reliability.

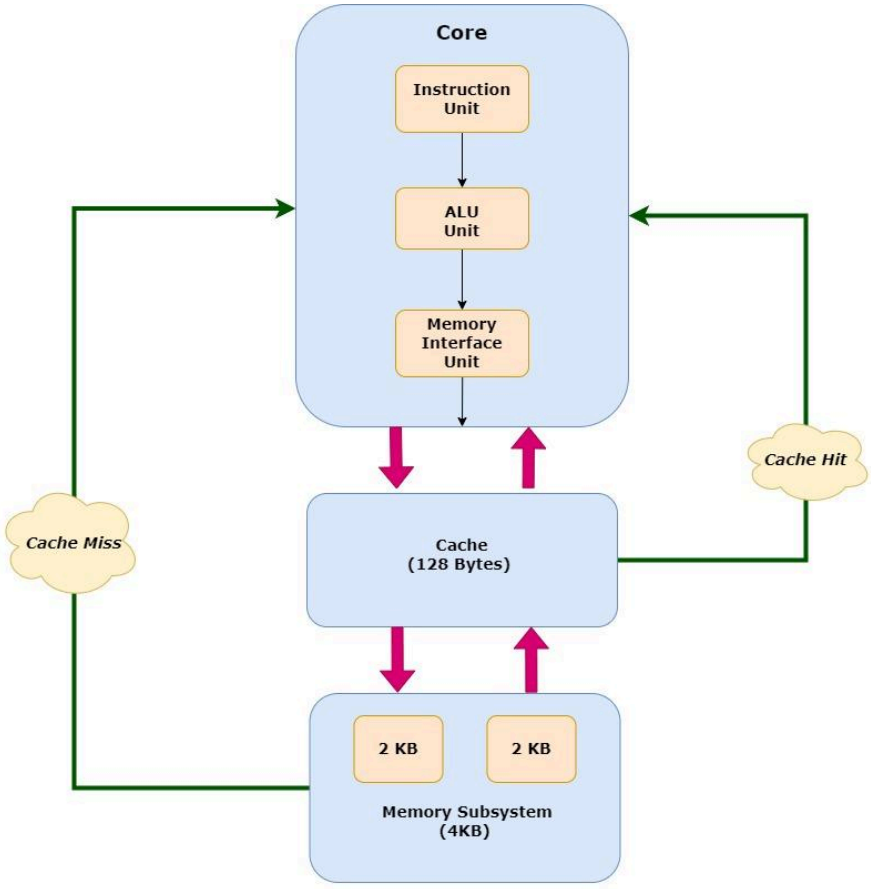


Fig: Data Flow Without Arbitor Diagram

2.2 Product Functions

The major functions of this Multi-Processor System include:

- ➔ Executing multiple instructions simultaneously across different processors, enhancing overall computational speed and efficiency.
- ➔ Utilizing a centralized memory-controller to efficiently manage memory access for all processors, ensuring optimal data availability and reducing memory access conflicts.
- ➔ Facilitating rapid and seamless data exchange between processors through the shared bus system, crucial for maintaining high performance in multi-processing tasks.
- ➔ Designed to be integrated into larger computing systems such as SoCs, CPUs, and network chips, providing scalability and adaptability to various computing needs and environments.
- ➔ Distributing processing tasks among the multiple processors to optimize resource utilization and prevent any single processor from becoming a bottleneck.
- ➔ Maintaining consistency of data in the cache memory across multiple processors, which is critical for ensuring data reliability and system integrity.
- ➔ Identifying and managing errors within the multi-processor environment, enhancing system reliability and robustness.
- ➔ Efficiently managing the power usage of the processors, which is crucial for reducing overall energy consumption, especially in complex systems like SoCs and CPUs.

2.3 User Classes and Characteristics

The anticipated users of this Multi-Processor System product are

High-Performance Computing Engineers: Professionals focusing on supercomputing and complex simulations
Telecommunications Sector: Engineers designing infrastructure for telecom networks can leverage the system for its high data throughput and efficient memory.
Automotive Industry Professionals: Especially those in advanced driver-assistance systems (ADAS) and autonomous vehicle development, where real-time data processing is critical.
Consumer Electronics Developers: For designing high-end gaming consoles and VR systems where speed and efficient data handling are key.
Academic and Research Institutions: Researchers in fields like quantum computing, artificial intelligence, and big data analytics can utilize the system's advanced processing capabilities for experimental and theoretical work.

Integrate is possible in terms of,

Modular Design: Allowing easy integration into diverse system architectures, enhancing customizability.
Robust Cache Coherence Protocol: Ensuring data accuracy and consistency across multiple processors, vital for complex calculations and simulations.
Advanced Thermal Management: Key for maintaining system integrity and performance in environments with high computational loads.
State-of-the-Art Power Efficiency: Reducing operational costs and enabling the use in energy-sensitive environments.
High-Speed Interconnects: Facilitating rapid data exchange, crucial for real-time applications.

2.4 Tools and Software

The QuestaSim tool is used to design the product in System Verilog. Questa Sim is available for Windows as well as Linux operating systems. The design is synthesizable and hence the design files can be used with various software like Xilinx Vivado tool for synthesis and implementation.

2.5 Design and Implementation Constraints

Design Constraints:

1. **Memory Configuration:** The system's memory configuration, with one partitioned memory array out of a 2KB memory setup, may introduce constraints in data handling efficiency. This configuration could impact the system's capability in managing large or complex data set
2. **Processor Limitations:** With three processors, there are inherent limits to parallel processing, impacting the handling of highly complex tasks.
3. **Cache Size:** A cache size of 128 bytes may not be sufficient for scenarios requiring extensive data caching for rapid processor access.
4. **Shared Bus Bottleneck:** The shared bus, essential for inter-processor communication, could become a bottleneck
5. **Memory Width and Depth:** The memory width of 1 byte and depth of 2K may limit the type and size of data that can be efficiently processed and stored, affecting the system's ability to handle wide or deep data sets.
6. **Fixed Opcode Operations:** The system is designed to handle a specific set of operations, this limitation might restrict the system's versatility in handling a broader range of computational tasks.

Implementation Constraints:

1. Reset and initialization of the Multi-Processor System triggers Cache Memory reset.
2. Memory write entry to cache will encompasses read then write operation.
3. Memory read entry when cache is full should evict the cache line and read from memory to write to cache line.

2.6 Assumptions and Dependencies

Design Assumptions:

1. Reset: Reliable and synchronous reset for all cores and shared resources to ensure a consistent start state.
2. Operations are not halted when the error signal is raised.
3. Instruction Handling: Presume that each core receives and processes instructions correctly one after another without any parallel operation.
4. Instruction Size: 28 Bit wide
5. Pipelining: For instruction processing no pipeline is used. Next instruction fetches only after the first is completed.
6. Cache Type : Direct Mapped Cache
7. Planning to design RTL to handle some error scenarios.
8. Different flags signal to know the status or give warnings to the user.

Design Dependencies:

1. Clock Dependency: The design and verification environment is dependent on a clock signal for synchronization
2. Control Signal Dependencies: Various control signals, such as trigger,write request, read request to create dependencies.
3. Error Handling Dependencies: Dependencies on error handling mechanisms are crucial. The design relies on error detection and correction strategies to ensure the integrity of data processing.

3. External Interface Requirements

3.1 Hardware Interfaces

1. **Processor Modules:** Three processors are designed to execute a set of predefined operations. Each processor's logical characteristics involve handling specific instructions like arithmetic operations, load/store, and shifts. The processors also interact with shared resources, necessitating efficient synchronization and communication protocols.

2. **Memory Subsystem:** The 2KB memory, with one array partitioned, serves as the primary storage. The memory module's logic involves managing read/write operations from the processors, necessitating a robust interface for efficient data exchange.

3. **Shared Bus:** This module is responsible for interconnecting the processors and the memory subsystem. The shared bus logic includes data transmission protocols and arbitration mechanisms, particularly utilizing a round-robin scheme for equal access opportunity.

4. **Cache Memory:** A 128-byte cache is integrated for faster data access. Its logic includes cache coherence protocols to maintain data consistency across processors.

5. **Counters and Control Logic:** Counters may be used for timing and synchronization purposes, ensuring coordinated operations among the processors and memory.

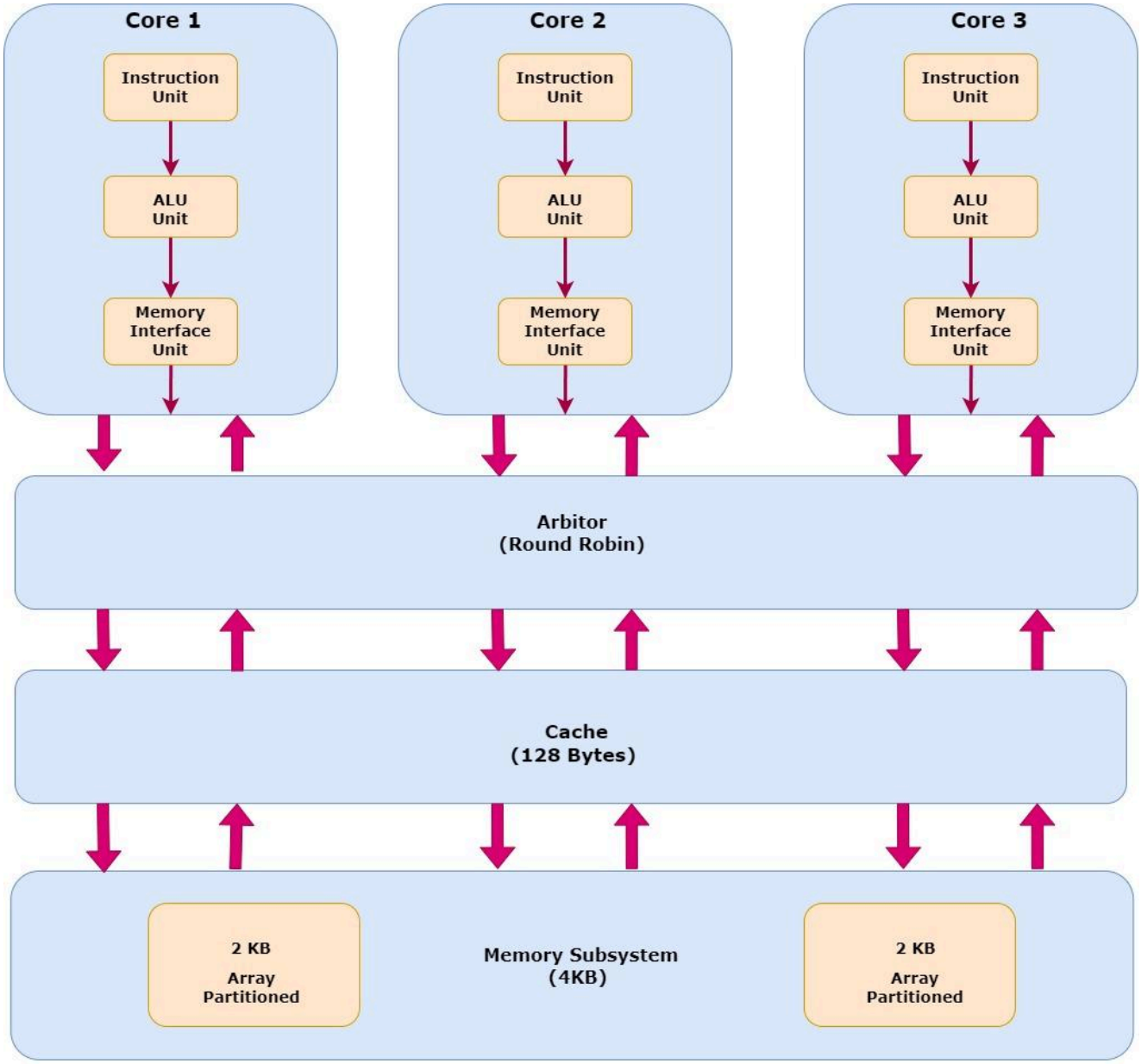
3.2 Software Interfaces

1. **EDA Tools:** Tools like Cadence, Synopsys, and Mentor Graphics for design, simulation, and verification. Versions would depend on the latest or most stable releases compatible with the design.

2. **Operating Systems:** OS versions such as Windows 10, Linux distributions like Ubuntu, or macOS.

3. **Hardware Description Languages (HDLs):** SystemVerilog or VHDL for writing the design
code.

4. **Simulation Libraries:** Libraries that come with the EDA tools, which might include standard cell libraries, delay calculation libraries, Assertions, UVM, Constraints & Randomization.



5. **Firmware/Drivers**: In cases where the Multi-Processor is part of a larger system, firmware or drivers for interfacing with the hardware might be developed using C/C++, with versions depending on the compiler or development environment used.

4. Product Features

4.1 Memory Subsystem

This module includes a 2KB memory array, with one partitioned section, and operates at a width of 1 Byte and a depth of 2K. The memory subsystem is critical for storing and providing data to the processors.

4.2 Processors

There are three processors designed to execute a set of operations including arithmetic, load/store, and shifts. The processors are the core computational units of the system, processing instructions and interacting with memory.

4.3 Shared Bus

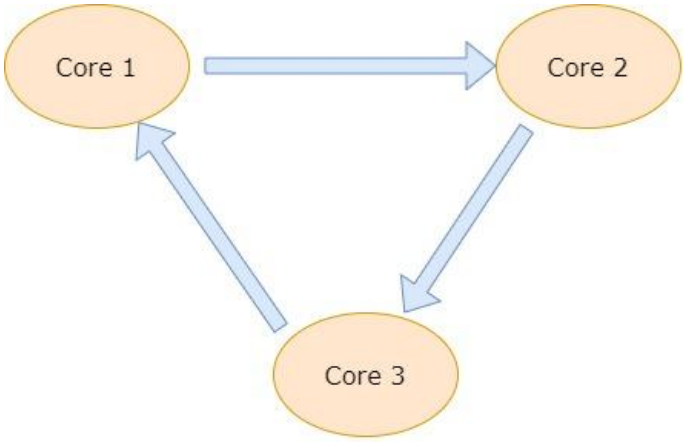
This bus serves as the communication backbone, connecting the processors to the memory subsystem. It operates under a round-robin arbitration scheme to manage access to memory.

4.4 Cache Memory

A 128-byte cache is included for each processor to optimize data access speed. The cache plays a crucial role in reducing data access latency.

4.5 Arbiter Mechanism

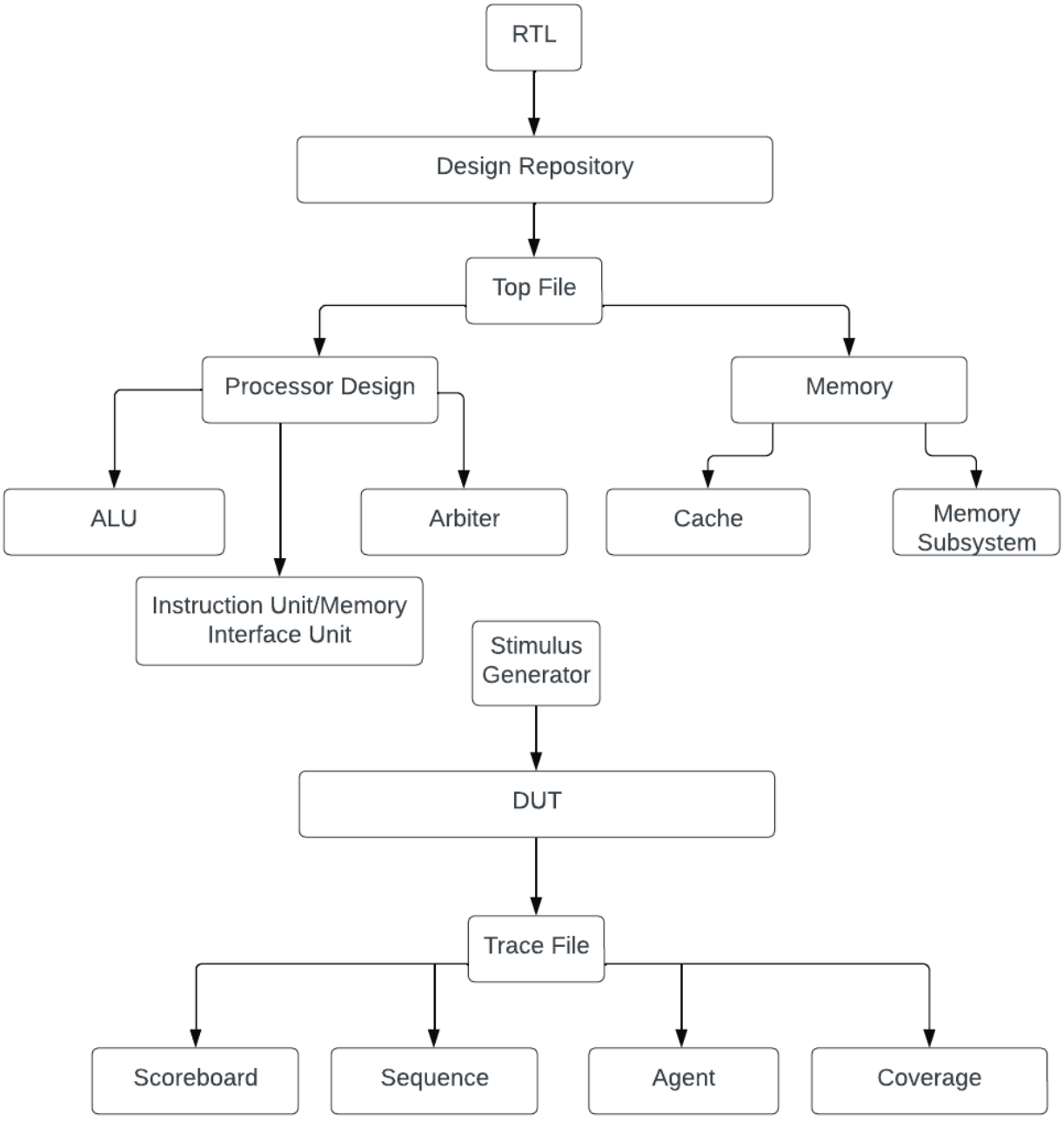
Utilizing a round-robin arbiter, this mechanism ensures equal opportunity for each processor to access the shared bus, crucial for managing processor-memory interactions.



Arbiter : Round Robin

5. Logic Design

5.1 Team15_ECE593W24



https://github.com/BLANK-1908/ECE593_Pre_Silicon

5.2 Design modules

<what will be your hardware coding style, how many modules, how many, how they will be connected to each other, how will they interact with each other and to the outer world>

- **Top Module**
  XXXXX
- **ALU Module**
  Contains internal modules to identify single cycle operations, multi cycle operations, shift operations and special functions
- **Instruction Unit**
  It is responsible for fetching and decoding the instruction needed to perform each cycle. Only one instruction can be performed per cycle, and it ensures this by waiting for the control signals.
- **Cache**
  Implementing a direct mapped cache with dirty and valid bits support to detect cache line fill and cache line modified condition for replacement
- **Main Memory**
  It implements all the control signals and manages the total memory size excluding cache. It receives the address from the arbiter and either returns the requested data or stores the requested data into memory.
- **Round robin arbiter**
  With 3 requests and 1 grant system

The data transmission happens in the form of packets

5.3 SystemVerilog abstraction Features

Parameterization: Make your designs more flexible and reusable by parameterizing various aspects of your design.
Abstraction : process of hiding implementation details while exposing only the essential features
Randomization : Generating random values for variables based on specified constraints.
Test: Represents a single test scenario or case.
Testbench: Contains the environment, test sequences, and other testbench components.
Environment: Represents the top-level verification environment and includes agents, scoreboard, configuration, and other components.
Agent: Interfaces with the DUT (Design Under Test) and contains drivers, monitors, and sequencers.
Driver: Drives stimulus to the Design Under Test.
Monitor: Observes and captures signals from the DUT.
Sequencer: Generates stimulus sequences for the driver.
Scoreboard: Compares expected and actual results and reports errors.

5.4 Simulation, Tools, Directory Structure

1. Simulation:

- Simulation using tools like ModelSim, VCS, QuestaSim
- UVM (Universal Verification Methodology) is often used as a verification framework
- Using SystemVerilog Assertions (SVA)

2. Tools:

- Debugging tools, waveform viewers, transcripts and performance analysis tools are also for effective verification.

3. Directory Structure:

- Planning to use well-organized directory structure helps manage the verification environment

```
/src          # Source code (RTL, testbench, sequences, etc.)
/sim          # Simulation scripts and setup files
/scripts      # Utility scripts for automation and analysis
/log          # Log files generated during simulation
/results      # Verification results and reports
/Makefile     # (run.do)
```

4. Documentation:

- Proper documentation is for maintaining and understanding the verification environment.

6. Verification

6.1 Testbench Style

For the verification of your multi-core processor project, the test bench is going to adopt a UVM-based style, enabling the independent and combined testing of each processor core, cache, and memory subsystem. It emphasizes reusability, ensuring components are applicable across various scenarios and future projects. Randomization will play a role in testing system responses to unpredictable conditions, while coverage-driven verification will ensure comprehensive testing of all functionalities. Additionally, the test bench will be self-checking, automatically verifying and reporting the system's behavior, streamlining the verification process for efficiency and thoroughness

6.2 Testing Strategies

The testing strategies for your multi-core processor project will encompass constrained random testing to cover a wide range of scenarios, directed testing for critical areas like core-cache interactions, a coverage-driven approach for comprehensive validation, and performance testing to evaluate metrics like latency, ensuring a robust and efficient system under various operational conditions.

6.3 Test case scenarios

Test case scenarios will include core-to-cache interaction tests to verify data handling and cache coherence, memory access tests under various load conditions, and arbiter functionality tests focusing on the round-robin mechanism. Error handling tests will be conducted to evaluate system resilience and recovery capabilities. These scenarios aim to ensure robustness, efficiency, and reliability across a spectrum of operational situations.

6.4 Others

Verification plans should include careful selection of simulation and synthesis tools, a structured debugging methodology for efficient error identification and resolution, and comprehensive documentation and reporting for traceability and future reference, ensuring a robust and effective verification process.

Summary

The verification plan for your multi-core processor project involves ensuring the seamless interaction of three processor cores with a shared cache and memory, using a round-robin arbiter for core request selection. This plan focuses on validating the efficient coordination and data management among the cores and the shared resources.