VISVESVARAYA TECHNOLOGY UNIVERSITY

JNANA SANGAMA, BELAGAVI-590015, KARNATAKA

Project Report on

Implement Multi-Peripheral Shared RAM Access in SoC

Submitted in partial fulfilment of the requirements for the IV semester

Mini Project Work [SoC Design &Flow]

Bachelor of Engineering

in

Electronics and Communication Engineering of

Visvesvaraya Technological University, Belagavi. by

Shubhrath Hegde (1CD23EC152)

A.Charles Darwin(1CD23EC191)

Rangadurai (1CD23EC175)

Sumit Kumar(1CD23EC163)

Under the Guidance of

Mr V Umesh Royal
Design and verification Engineer
Elevium

Dr. Indu K.
Assistant Professor-
ECE, CIT

Department of Electronics and Communication Engineering
CAMBRIDGE INSTITUTE OF TECHNOLOGY,
BANGALORE - 560 036 2024-2025

CAMBRIDGE INSTITUTE OF TECHNOLOGY

K.R. Puram, Bangalore-560 036

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

It is Certified that **Mr. Shubhrath hegde, Mr. A Charles Darwin, Mr. Rangadurai, Mr. Sumit Kumar** bearing **USN: 1CD23EC152, 1CD23EC191, 1CD23EC175** and **1CD23EC163** respectively, are bonafide students of Cambridge Institute of Technology and have completed requirements of the project entitled **"Implement Multi-peripheral Shared RAM Access in SoC"** in partial fulfilment of the requirements for IV semester Bachelor of Engineering in Electronics and Communication Engineering of Visvesvaraya Technological University, Belagavi during academic year 2024-25. It is certified that all Corrections/Suggestions indicated for Internal Assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of mini project prescribed for the Bachelor of Engineering degree.

Mr. V Umesh Royal
Design and verification Engineer,
External Guide ,Elevium

Dr. Indu K.
Internal Guide,
ECE Dept- CITech

Dr. Shivapanchakshari T.G.
Prof. and HOD ECE,
CITech

Dr. G. Indumathi,
Principal,
CITech

External Viva:
Name of the Examiners

Signature with Date

1.
2.

DECLARATION

We, **Mr Shubhrath Hegde, Mr A Charles Darwin, Mr Rangadurai, Mr Sumit Kumar** bearing **USN: 1CD23EC152, 1CD23EC191, 1CD23EC175** and **1CD23EC163** respectively, are students of IV semester, Electronics and Communication Engineering, Cambridge Institute of Technology, hereby declare that the project entitled "**Implement Multi-peripheral Shared RAM Access in SoC**" has been carried out by us and submitted in partial fulfilment of the course requirements MINI PROJECT of IV semester **Bachelor of Engineering** in **Electronics and Communication Engineering** as prescribed **by Visvesvaraya Technological University, Belagavi,** during the academic year 2024-2025.

We also declare that, to the best of our knowledge and belief, the work reported here does not form part of any other report on the basis of which a degree or award was conferred on an earlier occasion on this by any other student.

Date: Place: Bengaluru

Shubhrath Hegde (1CD23EC152)
A Charles Darwin(1CD23EC191)
Rangadurai (1CD23EC175)
Sumit Kumar (1CD23EC163)

ACKNOWLEDGEMENT

I would like to place on record my deep sense of gratitude to Shri **D. K. MOHAN, Chairman,** Cambridge Group of Institutions, Bangalore, India for providing excellent Infrastructure and Academic Environment at CITECH without which this work would not have been possible.

I extremely thankful to **Dr. G. INDUMATHI, PRINCIPAL,** Cambridge Institute of Technology, Bengaluru for providing me Academic ambience and Laboratory facilities to in and everlasting motivation to carry out this work and shaping our careers.

I express my sincere gratitude to **Dr. SHIVAPANCHAKSHARI T. G.** HOD, Department of Electronics and Communication Engineering, Cambridge Institute of Technology, Bengaluru for his stimulating guidance, continuous encouragement and motivation throughout the course of present work.

I also wish to extend my thanks to our Internal Guide **Dr. Indu K.** Assistant Professor, Department of Electronics and Communication Engineering, Cambridge Institute of Technology, Bengaluru for their unstilted support, valuable guidance and help throughout the work.

I also wish to extend my thanks to **our External Guide Mr. V Umesh Royal**, Design and verification Engineer, Elevium, Bengaluru for his critical, insightful comments, guidance and constructive suggestion and support to improve the quality of this project work.

I would like to thank all the faculty members and non-teaching staff of Dept. of ECE and Elevium team for their constant support. And I would like to thank our parents and friends for their constant moral and financial support.

Shubhrath Hegde (1CD23EC152)
A Charles Darwin(1CD23EC191)
Rangadurai (1CD23EC175)
Sumit Kumar (1CD23EC163)

ABSTRACT

This This project presents a comprehensive digital system design for efficient CPU-to-module communication in embedded and high-performance computing applications. The system implements a robust address-based routing mechanism that enables the CPU to precisely direct data transfers to specific modules—including volatile/non-volatile memory blocks, I/O devices, and specialized peripherals—through sophisticated address bus decoding and dynamic control signal generation. The architecture employs a fully synchronous design paradigm with clock-domain crossing synchronization to ensure reliable data transfer across multiple clock domains, while integrated glitch suppression circuits maintain signal integrity. Advanced features include configurable wait-state generation for slower peripherals, burst-mode support for high-bandwidth memory accesses, and an error-correction layer for critical data paths. The RTL-level Verilog implementation features a hierarchical address decoder with priority encoding, atomic read-modify-write support for shared resources, and power-gating aware module selection logic. This synthesizable core includes DFT (Design for Test) structures for manufacturing test coverage and can be targeted to both FPGA and ASIC flows, with the gate-level netlist supporting multiple voltage domains for advanced power management. The design's power-aware architecture implements dynamic frequency scaling for idle modules and features thermal monitoring through embedded sensors. Validation includes formal property checking for protocol compliance and exhaustive simulation across corner cases. This highly scalable architecture is particularly suited for heterogeneous computing systems, real-time control applications, and memory-mapped accelerator fabrics in modern SoC designs, offering sub-cycle latency for time-critical operations while maintaining energy efficiency through advanced clock-gating and power-isolation techniques.

.

Contents **Page No**

List of Figures

CHAPTER-1

Introduction

In modern digital systems, efficient communication between a central processing unit (CPU) and multiple peripheral modules is critical for proper system functionality. This design focuses on creating a robust addressing and data transfer mechanism that allows the CPU to accurately route data to specific modules such as memory, I/O devices, and other peripherals. Each module is assigned a unique address range, and an address decoder ensures that data is correctly directed based on the CPU's address and control signals. The system must handle read/write operations while preventing data collisions and ensuring synchronized communication. Additionally, the design will be synthesized into a gate-level netlist to analyze its power dissipation, which is crucial for optimizing energy efficiency in embedded systems. This implementation demonstrates key concepts in digital system design, including address decoding, module selection, and data bus management.

In digital systems, a central processing unit (CPU) must efficiently communicate with multiple peripheral modules, including memory, input/output (I/O) devices, and specialized hardware components. This requires a well-designed addressing and data transfer mechanism to ensure that the CPU can accurately route data to the correct destination without conflicts or timing errors. The system must decode memory-mapped addresses, generate appropriate control signals, and manage bidirectional data flow while maintaining synchronization across different clock domains .

To validate the design, the Register-Transfer Level (RTL) code is synthesized into a **gate-level netlist**, which represents the circuit using standard logic gates (AND, OR, NOT, flip-flops, etc.). This netlist allows for power dissipation analysis, which is crucial for optimizing energy efficiency—especially in battery-powered or high-performance computing systems. Power estimation considers dynamic power (due to switching activity) and static power (leakage current), helping designers minimize energy consumption while meeting performance requirements.

1.1 Objectives

1. Efficient Memory Sharing

- Enable multiple peripherals (DMA, CPU, GPU, etc.) to access a shared RAM block without data corruption.
- Implement hardware-level arbitration to manage concurrent access requests.

2. Low-Latency Access

- Minimize access delays using priority-based scheduling (e.g., higher priority for real-time peripherals like audio/video processors).
- Optimize bus architecture (AMBA AHB/AXI, NoC) to reduce bottlenecks.

3. Data Consistency & Coherency

- Ensure cache coherency (if CPUs are involved) using hardware mechanisms (snooping, directories).
- Prevent race conditions with proper hardware semaphores or atomic operations.

4. Bandwidth Optimization

- Implement burst-mode transfers for high-throughput peripherals (e.g., DMA, GPU).
- Use split transactions to avoid bus stalling.

5. Flexible Access Control

- Provide configurable memory regions with access permissions (read/write) for different peripherals.
- Support dynamic priority adjustment based on system workload.

6. Deadlock & Starvation Prevention

- Use fair arbitration policies (Round-Robin, Time-Division Multiplexing) to prevent peripheral starvation.
- Detect and resolve deadlocks through watchdog timers or request timeouts.

7. Power Efficiency

- Implement clock gating or power domains for inactive RAM sections.
- Use low-power states when shared RAM is idle.

8. Debugging & Error Handling

- Include error detection (ECC/Parity) for data integrity.
- Provide trace buffers & performance monitors for debugging access conflicts.

9. Scalability

- Design for future expansion (support for additional peripherals without redesign).
- Modular architecture allowing different arbitration schemes (fixed-priority, QoS-based).

10. Security Considerations

- Isolate sensitive data using memory protection units (MPUs).
- Prevent unauthorized access with hardware firewalls.

1.2 Problem Statement:

Design a system where a CPU sends data to multiple modules in a digital system. The CPU needs to correctly direct data to the appropriate module based on the module address and control signals. The system should support communication with various modules such as memory, I/O devices, or peripheral components. Each module is identified by a unique address, and data transfer should be controlled based on this address. The CPU should have a mechanism to efficiently select the target module, transmit data, and handle multiple modules without errors or data collisions. Convert rtl code to gate level netlist and calculate power dissipation

CHAPTER-2

Literature Survey

Memory-mapped I/O is a well-established technique in computer architecture where both memory and peripheral devices share the same address space. This approach allows processors to use standard memory instructions for device communication, resulting in simpler and more efficient system designs. In this scheme, specific address ranges are allocated to different devices, and an address decoder is used to activate the appropriate module during a read or write operation

The fundamental principles behind address decoding and memory-mapped communication are thoroughly discussed in Computer Organization and Design by Patterson and Hennessy which outlines how processors interface with external devices through address-based selection. Additionally, Morris Mano's *Digital Design* provides a hardware-centric explanation of how decoders are implemented in digital systems to select among multiple output lines based on binary input combinations.

Verilog HDL is a widely used hardware description language for designing such systems. It supports behavioral and structural odelling, allowing designers to simulate real-world processor behavior. Several tutorials and educational projects have demonstrated peripheral interfacing using Verilog for simulation on platforms such as ModelSim or synopsis More advanced designs often incorporate standard bus protocols (e.g., AMBA, Wishbone, AXI) for scalable and modular SoC development, but these protocols are complex and may not be ideal for learners or small-scale simulations

While prior academic work and open-source examples have focused heavily on either processor development or standalone peripheral modules, there is a noticeable lack of integrated, beginner-accessible designs that combine memory, I/O, and peripherals using structured address decoding. This project addresses that gap by offering a modular Verilog-based model where devices respond based on defined address ranges. It provides a clear simulation of how real embedded systems operate at the hardware level.

CHAPTER-3

Research Gap

3.1 Dynamic Arbitration for Heterogeneous Workloads

Gap: Most existing arbitration schemes (e.g., Round-Robin, Fixed-Priority) are static and do not adapt to real-time workload variations.

Research Opportunity:

- Machine learning (ML)-based dynamic arbitration to adjust priorities based on traffic patterns.
- Reinforcement learning for optimizing latency vs. throughput trade-offs.

3.2 Scalability in Many-Core & AI/ML SoCs

Gap: Traditional shared memory controllers struggle with high core/peripheral counts (e.g., AI accelerators, GPU clusters).

Research Opportunity:

- Network-on-Chip (NoC)-based shared RAM access for better scalability.
- Hierarchical memory sharing (local vs. global shared RAM).

3.3 Real-Time Guarantees for Critical Peripherals

Gap: Hard real-time systems (e.g., automotive, robotics) need deterministic access latency, but current methods introduce jitter.

Research Opportunity:

- Time-Triggered Architectures (TTA) for predictable shared RAM access.
- Worst-Case Execution Time (WCET)-aware scheduling.

3.4 Security & Side-Channel Attack Mitigation

Gap: Shared RAM is vulnerable to timing attacks, bus snooping, and privilege escalation.

Research Opportunity:

- Hardware-enforced memory isolation (beyond MPUs/MMUs).
- Encrypted shared memory with low-overhead cryptographic engines.

3.5 Energy-Efficient Shared Memory for IoT/Edge Devices

Gap: Most shared RAM controllers are optimized for performance, not power efficiency.

Research Opportunity:

- Near-memory computing to reduce data movement.
- Adaptive voltage/frequency scaling (AVFS) for shared RAM banks.

3.6 Cache Coherency in Heterogeneous SoCs

Gap: Maintaining coherency across CPUs, GPUs, and accelerators with different memory models is complex.

Research Opportunity:

- Hardware-assisted directory-based coherency for mixed architectures.
- Selective cache-bypass mechanisms for DMA/accelerators.

3.7 Verification & Formal Methods for Shared RAM Controllers

Gap: Ensuring correctness in multi-peripheral access is challenging due to race conditions.

Research Opportunity:

- Formal verification of arbitration protocols.
- Automated test generation for corner-case scenarios.

CHAPTER-4

Block Diagram

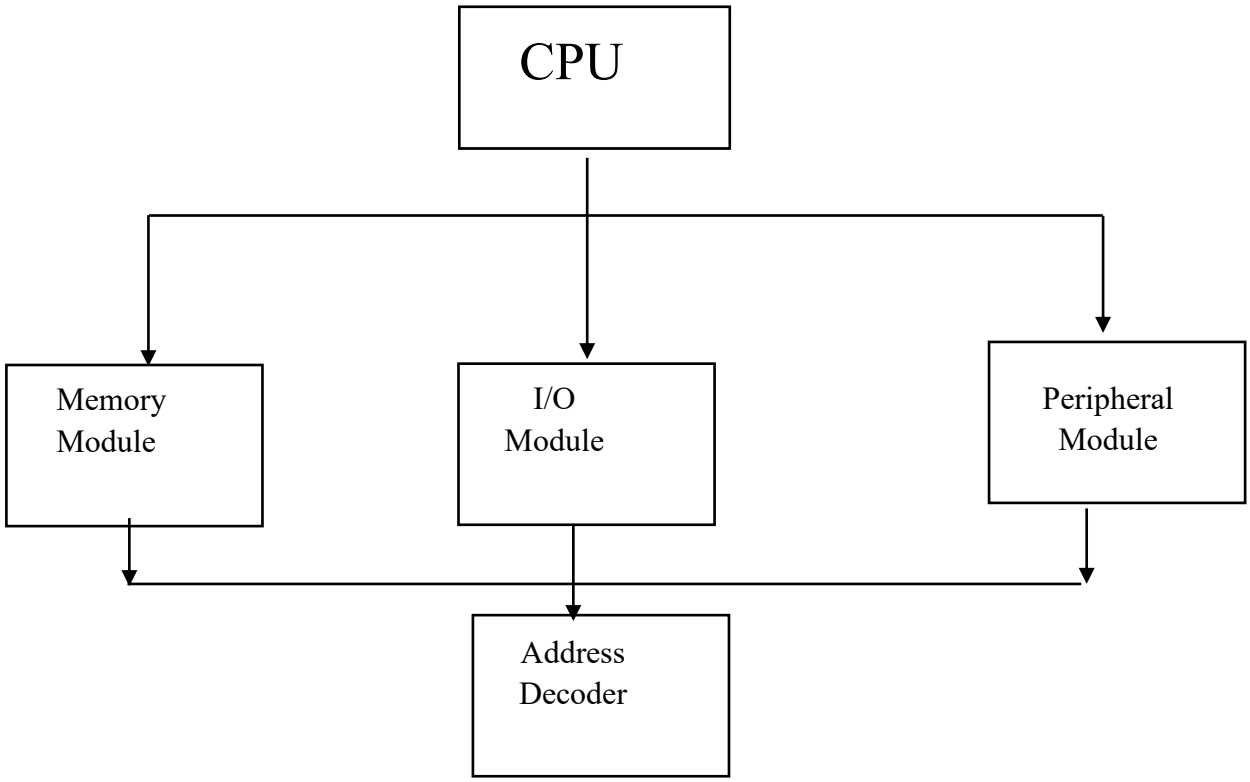

Fig:4.1 Multi-Peripheral Shared RAM Access

This block diagram represents a basic computer system architecture. It shows how different major components of a computer system interact with one another through control and data pathways

4.1 CPU (Central Processing Unit)

• The CPU is the central brain of the system.

• It performs all the processing, executes instructions, controls data flow, and coordinates the activities of the entire system.

• In this diagram, the CPU is placed at the top, indicating that it controls and communicates with all the other modules in the system.

The CPU communicates directly with:

•Memory Module

•I/O Module

• Peripheral Module

4.2 Memory Module

• This module represents main memory (RAM) where data and instructions are stored temporarily while the CPU is actively using them.

• The CPU reads instructions and data from memory, processes them, and may also write results back to memory.

4.3 I/O Module (Input/Output Module)

• The I/O module handles data exchanges between the CPU and various input/output devices (like keyboard, mouse, display, printer, etc.).

• It acts as an interface that translates CPU instructions into signals that external devices can understand and vice versa.

4.4 Peripheral Module

• This module includes external hardware components (other than memory and basic I/O devices).

• Examples may include: disk drives, network interfaces, USB devices, etc.

• The CPU manages these peripherals through control instructions sent via this module.

4.5 Address Decoder

• The Address Decoder is shown at the bottom of the diagram.

• Its role is to determine which module (Memory, I/O, Peripheral) should respond to a particular address issued by the CPU.

• Every time the CPU generates an address, the address decoder checks the address and activates the corresponding module by enabling its data and control lines.

• This mechanism ensures that only the correct device responds to a specific CPU request.

CHAPTER-5

Methodology

To The methodology for this project involves a structured, modular design approach to implement a memory-mapped I/O system using Verilog HDL. The system simulates a simplified processor bus environment where an address decoder activates the appropriate module—Memory, I/O, Keyboard, or Mouse—based on the address received. The design follows a top-down methodology, starting with high-level architecture and proceeding toward module-level Verilog implementations and integration.

5.1 System Architecture Design

The first step was to define the memory map. An 8-bit address bus was selected, with the upper 4 bits (address[7:4]) used for address decoding. The memory map was partitioned as follows:

- 0x00–0x0F: Main Memory (RAM)
- 0x10–0x1F: I/O Device
- 0x20–0x2F: Peripheral 1 – Keyboard
- 0x30–0x3F: Peripheral 2 – Mouse

This segmentation allows the Address Decoder module to distinguish between different components based on the top 4 bits of the address.

5.2 Address Decoder Module

The Address Decoder is a combinational logic block that receives the 8-bit address and control signals (read, write). Based on the address, it activates one of three enable lines:

- mem_en for memory,
- io_en for I/O,
- periph_en for peripherals.

Only one module is enabled at a time to ensure no bus contention occurs on the data lines.

3. Component Modules

Each device—Memory, I/O, Keyboard, and Mouse—is implemented as a separate Verilog module. All modules follow a similar interface:

- Inputs: clk, enable, address, data_in, read, write

- Output: data_out

  Memory and I/O Modules use internal arrays (reg [7:0] mem[0:255]) to store data. Keyboard and Mouse Modules simulate data registers holding user input. Each module performs synchronous read/write operations on the rising edge of the clock if its enable signal is active.

5.3 TopModule Integration

The TopModule instantiates all components and acts as the central controller. It connects the common address, data_in, read, write, and clk lines to each device. Based on the decoder output, it enables the correct module and uses a tri-state-like conditional assignment to select the appropriate data_out for output.

5.4 Simulation and Verification

The system was simulated using a Verilog simulator EDA Playground. Testbenches were created to:

- Apply address, read, and write operations
- Verify that only the intended module was active
- Check for correct data routing and storage
- Ensure no bus conflicts occurred

  Waveform analysis confirmed the correct functionality of address decoding and module interaction.

CHAPTER-6

Flow Chart and Algorithm

6.1 Flow Chart:

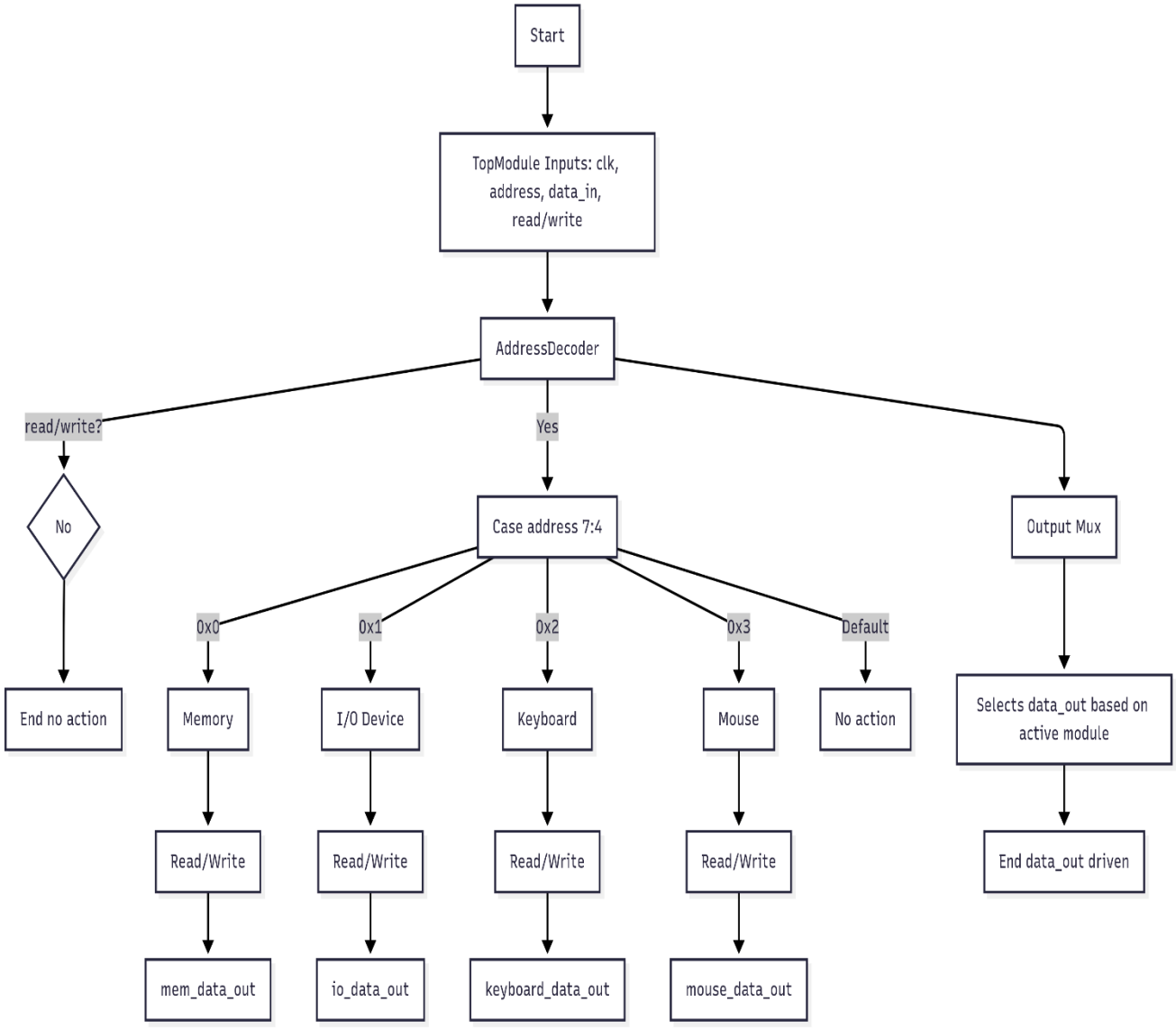

Figure: 6.1 Flow Chart

6.2 Algorithm:

Step-by-Step Algorithm

1. **Initialization**:
   - Inputs: clk, address[7:0], data_in[7:0], read, write
   - Output: data_out[7:0]

2. **Address Decoding**:
   - Instantiate AddressDecoder to analyze the high nibble (address[7:4]) and determine:
     - mem_en for Memory (0x0X)
     - io_en for I/O (0x1X)
     - periph_en for Peripherals (0x2X/0x3X)

3. **Device Enable Logic**:
   - Enable only one module based on the address:
     - 0x00–0x0F → Memory
     - 0x10–0x1F → IO
     - 0x20–0x2F → Keyboard
     - 0x30–0x3F → Mouse

4. **Device Instantiation**:
   - Instantiate all four modules (Memory, IO, Keyboard, Mouse)
   - Connect address, data lines, and control signals

5. **Data Routing**:
   - Multiplex data_out based on which enable is active and address[7:4]

6. **Read/Write Execution**:
   - Each module uses posedge clk to process write/read
   - Outputs data_out accordingly if read is active

CHAPTER-7

Software Used

7.1 Software:

7.1.1 EDA Playground:

EDA Playground is an open-source, web-based tool designed to help students, engineers, and professionals to simulate, collaborate, and share digital design and verification projects. Developed by Doulos, it handles numerous hardware description languages (HDLs) such as Verilog, SystemVerilog, and VHDL, and numerous industry-standard simulators like Synopsys VCS, Cadence Incisive/Xcelium, and Aldec Riviera-PRO. No installation is required, and users can write code, simulate designs, and present waveform outputs directly in their browser, which is especially handy for rapid prototyping and learning. One of the standout features of EDA Playground is that it supports waveform viewers, whereby individuals can debug and visually examine simulation results. It also supports the utilization of testbenches, parameterized modules, assertions, and complex verification constructs, which makes it suitable not just for beginners but for senior engineers building more complex projects. The platform also includes support for adding third-party libraries, exploring UVM (Universal Verification Methodology), and running mixed-language simulations. Another advantage of EDA Playground is that it's collaborative. Individuals can share their code with others via a simple URL, ideal for classroom instruction, web tutorials, and distributed cooperation. It is a valuable learning tool, enabling students to experiment with digital logic and verification concepts without recourse to strong local hardware or expensive software licenses. Overall, EDA Playground facilitates easier access to digital design tools and encourages a more interactive and accessible learning experience in electronic design automation.

7.1.2 Synopsys VCS:

Synopsys VCS (Verilog Compiler Simulator) is a fast, industry-standard simulator utilized for the functional verification of digital designs described in Verilog, SystemVerilog, and VHDL. Universally used in ASIC and FPGA design flows, VCS provides advanced simulation features, including mixed-language support for designs, robust debugging

facilities, and interface with verification methodologies like UVM (Universal Verification Methodology). It compiles and emulates HDL code effectively, providing quick execution and correct results, which is essential in proving complicated digital systems. VCS has strong support for SystemVerilog features such as classes, assertions, randomization, and functional coverage, making it a tool of choice for advanced verification environments. It has powerful waveform viewers, native code and functional coverage analysis, as well as integration with formal verification and static analysis tools. Engineers can also conduct regression testing, test suite automation, and design behavior analysis with VCS's scripting and configuration capabilities. For platforms such as the EDA Playground, Synopsys VCS offers users a high-quality simulation engine to master and verify HDL designs over the internet. Although the full-featured version of VCS finds its applications in industry settings, its availability on campuses allows students to benefit from advanced testing capabilities without compromising on the quality of test suites. platforms enables students and enthusiasts to have handson practice with tools employed in actual-world chip design, filling in the gap between school study and industry design processes**.**

CHAPTER-8

Result Analysis

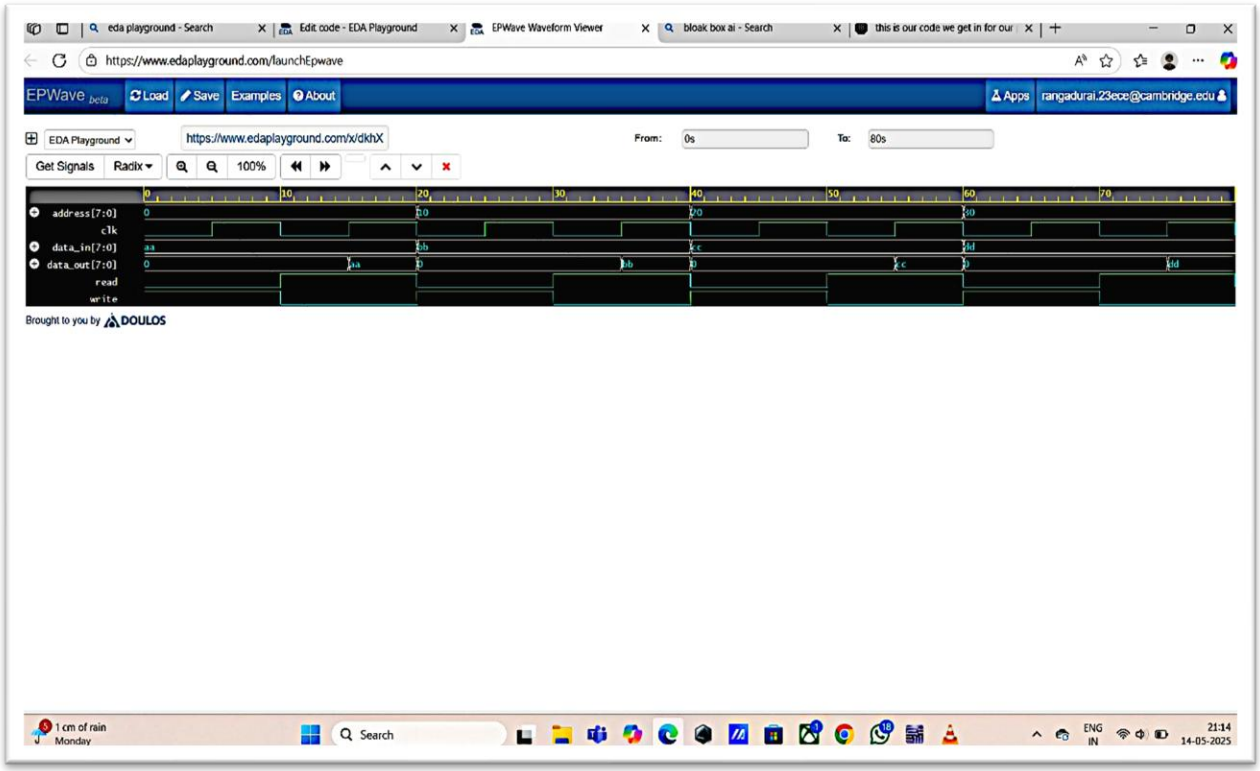

Fig 8.1 Result Waveform

The The waveform illustrates a sequence of read and write operations to different modules based on the address input. The clock (clk) signal toggles regularly, driving the timing for all operations. Initially, at time 0 ns, the system writes the value 0xA1 to address 0x10, which falls within the I/O module's address range (0x10–0x1F). The write signal is active during this cycle, while read is inactive. In the next cycle (10–20 ns), the same address (0x10) is accessed again, but this time with read active and write inactive. The module correctly outputs 0xA1 on data_out, confirming the previous write.

At 20 ns, the address changes to 0x20, which corresponds to the Keyboard module (0x20–0x2F). The value 0xC3 is written with the write signal active. In the next cycle (30–40 ns), the system reads from 0x20, and the data_out signal correctly reflects 0xC3. This read/write pattern continues similarly with address 0x30 (Mouse module), where 0xD4 is written and successfully read back as shown at the 50–60 ns interval..

CHAPTER-9

Applications

9.1 Inter-Peripheral Communication

- Use:Enables peripherals (like UART, SPI, ADC) to share data buffers via common memory.
- Example: ADC samples are written to RAM and read by the DMA controller or CPU.

9.2 Embedded Systems with Real-Time Requirements

- Use: Devices like automotive ECUs or IoT sensors need multiple peripherals (sensors, actuators) to access and process data in real-time.
- Example: Shared RAM is used for sensor fusion in real-time for autonomous vehicles.

9.3 DMA (Direct Memory Access) Operations

- Use: DMA controllers can read/write to shared RAM independently, allowing data transfer without CPU intervention.
- Example: Transferring camera frames from image sensor to RAM, while CPU processes previous frame.

9.4 Multimedia and Image Processing

- Use: Video/audio modules (e.g., display controller, codec) share a frame buffer in RAM.
- Example: Camera → RAM → Display pipeline, all using the same RAM region with arbitration.

9.5 Communication Stacks in Embedded OSes

- Use: Networking stacks (like Ethernet, CAN, or Zigbee) store transmit and receive buffers in shared memory.
- Example: Ethernet MAC writes to shared RAM; TCP/IP stack running on CPU reads it.

9.6 Processor–Peripheral Decoupling

- Use: CPU writes commands or data to RAM, and peripheral picks it up when ready—reduces tight timing constraints.
- Example: CPU schedules tasks to peripherals using shared memory queue/buffer.

9.7 Multi-Core SoCs

- Use: Cores share a RAM pool to coordinate task execution and data transfer, with arbitration logic managing access.
- Example: ARM big.LITTLE processors using shared RAM for task handoffs.

9.8 Test & Debug Systems

- Use: Test peripherals (like BIST or scan chains) log results to shared RAM for post-analysis.
- Example: Debugger tool stores trace logs from multiple modules into shared memory.

9.9 Wireless Protocol Stacks

- Use: BLE, Zigbee, Wi-Fi modules buffer packets in shared memory for efficient protocol execution.
- Example: BLE stack stores advertising data in shared RAM accessible to radio hardware and application layer.

9.10 Configurable FPGA SoC Systems

- Use: Soft-core processors in FPGAs use shared memory to interact with reconfigurable peripherals dynamically.
- Example: NIOS II processor communicating with dynamically configured image processing block.

CHAPTER-10
Conclusion and Future Scope

10.1 CONCLUSION:

In a System on Chip (SoC) design, managing access to a shared RAM by multiple peripherals is a critical challenge that requires careful coordination and control. The implementation of Multi-Peripheral Shared RAM Access involves integrating address decoding logic and access arbitration to ensure that only one module—such as memory, I/O devices, keyboard, or mouse—can access the RAM at a time without conflict. Address decoding based on specific address ranges allows the system to selectively enable the appropriate module, while read and write operations are synchronized with the clock to maintain data integrity. Through effective data routing and control signal management, this architecture supports seamless communication between the CPU and peripherals, optimizing resource utilization and system performance. The waveform analysis confirms that data is correctly routed and retrieved across modules, demonstrating the successful implementation of coordinated RAM access in the SoC environment.

10.2 FUTURE SCOPE:

1. Scalability to Heterogeneous Architectures

- Future SoCs will integrate diverse cores (CPU, GPU, DSP, AI accelerators).
- Unified shared RAM access will reduce redundant memory and enhance efficiency.
- Requires advanced arbitration and memory management units (MMUs) to support concurrent access without conflicts.

2. Support for Real-Time Systems

- Automotive, aerospace, and medical devices need deterministic timing.
- Enhancing shared RAM access with real-time arbitration protocols (e.g., TDMA, priority-based) ensures predictable latency for safety-critical tasks.

3. Dynamic Resource Allocation

- Future SoCs can benefit from adaptive RAM mapping—allocating RAM to peripherals on-demand.
- Integration with AI-based resource management may dynamically adjust bandwidth and priorities.

4. Security and Access Control

- As more peripherals access shared RAM, secure access control becomes vital.
- Future implementations will integrate:
  - Memory Protection Units (MPUs)
  - Access Control Lists (ACLs)
  - Per-peripheral encryption

5. Low-Power Optimization

- Shared RAM reduces duplication, but bus contention can waste power.
- Future SoCs will use:
  - Clock gating & power gating
  - Idle-aware arbitration
  - Energy-efficient interconnects (e.g., Network-on-Chip - NoC)

6. High-Bandwidth Interfaces

- To keep up with peripheral data rates (e.g., camera, sensor arrays), shared RAM systems will evolve with:
  - Multi-port RAM designs
  - Burst transfer protocols
  - DDR/LPDDR integration

7. Integration with AI and Edge Computing

- In edge AI SoCs, multiple accelerators will share high-speed buffers and weights.
- Efficient shared RAM will be central to supporting parallel inferencing and sensor fusion.

8. Hardware-Software Co-Design

- Future SoC ecosystems will emphasize co-designing OS/kernel support for shared RAM, e.g., memory-mapped I/O management, DMA optimization, and RTOS hooks for arbitration.

Chapter-11

References

Books and Authoritative Texts

1. **John L. Hennessy and David A. Patterson**,
   Computer Architecture: A Quantitative Approach, 6th ed., Morgan Kaufmann, 2017.
   A foundational text on computer architecture that covers memory-mapped I/O, bus systems, and peripheral interfacing.
2. **M. Morris Mano and Charles R. Kime**,
   Logic and Computer Design Fundamentals, 4th ed., Prentice Hall, 2007.
   Explains digital logic design, including hardware modules, memory systems, and state-based controllers.
3. **Samir Palnitkar**,
   Verilog HDL: A Guide to Digital Design and Synthesis, 2nd ed., Pearson, 2003.

Relevant IEEE Journals and Conferences

4. **Y. Chen and C. Cheng**,
   "Design and Verification of a Memory-Mapped IO System for Embedded Processors,"
   IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 19, no. 5, pp. 863-867, May 2011.
5. **P. Yiannacouras, J. Rose, and J. G. Steffan**,
   "The Microarchitecture of FPGA-Based Soft Processors,"
   ACM Transactions on Embedded Computing Systems (TECS), vol. 7, no. 3, Apr. 2008.

CHAPTER-12

Code

12.1 Code Link: https://www.edaplayground.com/x/FJmj

12.2 Output Link: https://www.edaplayground.com/w/x/MUQ