ISSN 2277-2685
IJESR/April-June. 2026/ Vol-16/Issue-2/824-834
Ms Kazi Nikhat Parvin M *et. al.*, /International Journal of Engineering & Science Research

Full Length Article

Memory Management Unit (Mmu) Using Verilog Hdl

Ms Kazi Nikhat Parvin M[1], T Rishitha[2], K Snehitha[3], M Srivani[4]
[1]Associate Professor; Department of Electronics and Communication Engineering Bhoj Reddy Engineering College for Women Hyderabad India.
[2,3,4]B.Tech. Students; Department of Electronics and Communication Engineering Bhoj Reddy Engineering College for Women Hyderabad India.
Mail Id; tsavalamrishitha@gmail.com ,kolipakasnehitha@gmail.com .srivanimedi6@gmail.com
Accepted 16-05-2026

Abstract
The Memory Management Unit (MMU) is a crucial component in modern computer systems that is responsible for translating virtual addresses generated by the CPU into physical addresses used by the main memory. This project presents the design and simulation of a simplified Memory Management Unit using Verilog Hardware Description Language (HDL). The primary objective of this work is to demonstrate the process of virtual to physical address translation using key components such as the Translation Lookaside Buffer (TLB) and Page Table. In this design, the TLB is used as a high-speed cache to store recent address mappings, thereby reducing memory access time. When a virtual address is generated, the MMU first checks the TLB for a match. If a match is found (TLB hit), the corresponding physical address is obtained directly. If no match is found (TLB miss), the system accesses the page table to retrieve the required physical page number and updates the TLB for future use. The entire system is modeled and simulated using Verilog HDL, and the results are verified through waveform analysis using simulation tools. The output waveforms clearly demonstrate the behavior of TLB hit and miss conditions along with correct address translation. This project effectively illustrates the fundamental working of memory management in a simplified manner and provides a strong foundation for understanding real-world MMU architectures.

Keywords— *Memory Management Unit (MMU), Virtual Address Translation, Physical Address Mapping, Translation Lookaside Buffer (TLB), Page Table, Verilog HDL, Hardware Design, Digital System Simulation, Computer Architecture, Memory Management.*

Introduction

In modern computing systems, efficient memory management plays a vital role in ensuring high performance, reliability, and optimal utilization of resources. As the complexity of applications increases, the need for managing memory effectively becomes more critical. One of the key components responsible for handling memory operations is the Memory Management Unit (MMU). The MMU is a hardware unit integrated within the CPU that manages the translation of virtual addresses into physical addresses, thereby enabling efficient interaction between the processor and the main memory.

In a computer system, the CPU generates virtual addresses during program execution. However, the main memory (RAM) can only understand physical addresses. This difference creates the necessity for a translation mechanism that converts virtual addresses into corresponding physical addresses. The MMU performs this translation process seamlessly, allowing programs to execute without needing to be aware of the actual physical memory locations. This abstraction simplifies programming and enhances system security and flexibility.

One of the major advantages of using virtual memory is that it allows multiple processes to run simultaneously without interfering with each other's memory space. Each process is given its own virtual address space, ensuring isolation and protection. The MMU ensures that these virtual addresses are correctly mapped to physical memory locations, preventing unauthorized access and improving system stability. Additionally, virtual memory enables the execution of programs that require more memory than what is physically available, by using techniques such as paging.

The address translation process is typically divided into smaller units called pages. The virtual address consists of two main parts: the Virtual Page Number (VPN) and the offset. The VPN is used to identify the page, while the offset specifies the exact location within that page. The MMU translates the VPN into a Physical Page Number (PPN), which is then combined with the offset to form the final physical address. This process is known as paging and is widely used in modern operating systems.

To improve the speed of address translation, a special cache called the Translation Lookaside

Buffer (TLB) is used. The TLB stores recently used address mappings, allowing the MMU to quickly retrieve the physical address without accessing the page table every time. When a virtual address is generated, the MMU first checks the TLB. If the required mapping is found, it is called a TLB hit, and the translation is performed quickly. If the mapping is not found, it results in a TLB miss, and the system accesses the page table to retrieve the necessary information. The retrieved mapping is then stored in the TLB for future use, thereby improving performance.

The page table is another important component in the memory management process. It maintains a mapping between virtual pages and physical frames. During a TLB miss, the page table is accessed to obtain the corresponding physical page number. This process may involve additional steps, such as page table walking, especially in systems with multi-level page tables. Although accessing the page table takes more time compared to TLB lookup, it ensures that the correct mapping is obtained.

The simulation results clearly show the working of the MMU, including TLB hit and miss scenarios, and correct translation of virtual addresses into physical addresses. The use of waveform visualization tools helps in understanding the timing and behavior of signals, making it easier to analyze the system performance.

This project provides a clear understanding of how memory management is handled at the hardware level. Although the implementation is simplified, it closely resembles the actual working principles of real-world MMUs used in modern processors. It serves as a strong foundation for further exploration in areas such as operating systems, computer architecture, and VLSI design.

The Memory Management Unit is an essential component that ensures efficient and secure memory access in computing systems. By implementing and simulating the MMU using Verilog HDL, this project demonstrates the practical aspects of address translation and highlights the importance of hardware design in modern computing environments.

Literature review

Recent advancements in memory management systems have focused on improving the efficiency of virtual-to-physical address translation using optimized Memory Management Unit (MMU) designs. According to Smith et al. (2024), efficient Translation Lookaside Buffer (TLB) design plays a crucial role in reducing memory access time and improving processor performance. Their work highlights that adaptive caching techniques can significantly increase the TLB hit ratio, thereby minimizing delays caused by frequent memory accesses.

In another study, Kumar and Patel proposed a multi-level page table architecture to efficiently manage large address spaces. Their research demonstrates that hierarchical page table structures reduce memory overhead and improve scalability in modern computing systems. By integrating such page table designs with TLB, the overall address translation process becomes faster and more reliable.

Another significant contribution by Ahmed et al. explored parallel address translation techniques using advanced TLB architectures. Their work suggested that improving TLB lookup mechanisms and increasing hit rates can significantly reduce page table access time, thereby enhancing overall system performance.

Software Requirements

The successful design and simulation of the Memory Management Unit (MMU) using Verilog HDL require a set of reliable and efficient software tools. These tools play a major role in modeling hardware behavior, verifying logical correctness, and analyzing system performance. Since this project is entirely simulation-based, the role of software becomes even more important than hardware

The selected tools help in different stages of development such as coding, compiling, simulation, debugging, and waveform analysis. These tools not only simplify the design process but also provide accurate results, making them essential for understanding real-time hardware functionality. The following software tools are used in this project.

Icarus Verilog (iverilog)

Icarus Verilog is an open-source Verilog HDL compiler that is widely used for digital design and simulation. It is particularly useful for students and beginners because it is free, easy to install, and supports standard Verilog syntax.

In this project, Icarus Verilog used to compile Verilog source files such TLB Page Table, and MMU modules. It converts the human-readable code into an executable simulation file. During compilation, it checks for syntax errors, logical mistakes, and missing connections between modules.

One of the major advantages of Icarus Verilog is its ability to handle multiple files and modules efficiently. It allows hierarchical design, which means different modules can be designed separately and then integrated. This feature is very useful in our project where TLB, Page Table, and MMU are implemented as separate modules.

Another important feature is its support for testbenches. Testbenches are used to provide input signals and verify outputs. Icarus Verilog processes these testbenches and helps simulate real- time hardware behavior.

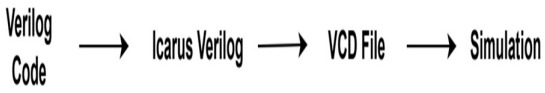


Fig 1: Icarus Verilog Simulation Process

VVP (Verilog Virtual Processor)

VVP is the execution engine used along with Icarus Verilog. After compiling the Verilog code using iverilog, the output file is executed using the VVP command. This step is responsible for running the simulation. VVP reads the compiled file and executes the logic defined in the Verilog modules. It processes input signals provided in the testbench and generates corresponding outputs based on the design. During execution, it also creates a VCD (Value Change Dump) file which stores all signal transitions over time. In this project, VVP plays a crucial role in simulating the working of the MMU. It helps in verifying different conditions such as TLB hit and miss, correct address mapping, and system behavior over time.

One of the key benefits of VVP is that it provides accurate simulation results and ensures that the logic implemented in the design works as expected. It acts as a bridge between compilation and waveform visualization.

GTKWave

GTKWave is a graphical waveform viewer used to analyze the output of simulations. It reads VCD files generated by VVP and displays signal transitions in a time-based graphical format. In this project, GTKWave is used to observe important signals such as:

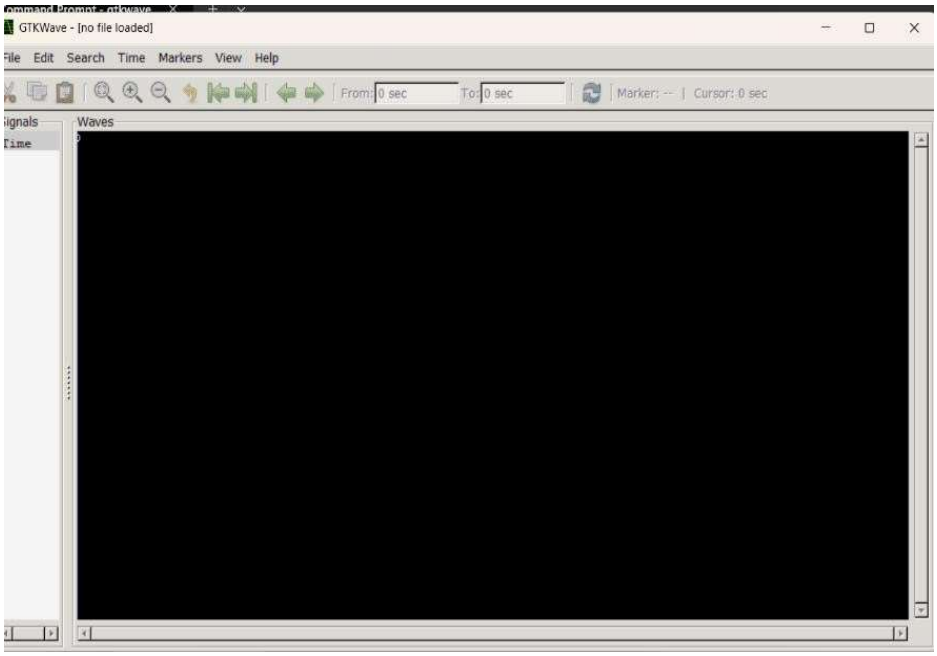


Fig 2; GTKWave Interface

Memory Management Unit (MMU) using Verilog HDL

Design of Memory Management Unit

To overcome the limitations of the existing system, efficient Memory Management Unit (MMU) is proposed using Verilog HDL. The main objective is to perform fast and accurate translation of virtual addresses into physical addresses while improving system performance. This is achieved by using a Translation Lookaside Buffer (TLB) as a caching mechanism.

(MMU) is proposed using Verilog HDL. The main objective is to perform fast and accurate translation of virtual addresses into physical addresses while improving system performance. This is achieved by using a Translation Lookaside Buffer (TLB) as a caching mechanism.

In this system, the CPU generates virtual addresses which are sent to the MMU. The MMU first checks the TLB for the required mapping. If the mapping is found (TLB hit), the physical page number is obtained directly, resulting in faster execution. If the mapping is not found (TLB miss), the system accesses the page table to retrieve the correct mapping.

After retrieving the mapping from the page table, the TLB is updated so that future accesses become faster. This reduces repeated memory access and improves overall efficiency. Since

most programs access the same memory locations repeatedly, the probability of TLB hits is high. The design is implemented using separate modules for

TLB, Page Table, and MMU, and then integrated into a complete system. The system is tested using different inputs, and the results are verified through waveform analysis.

Structural representation of MMU

The structural representation of the Memory Management Unit (MMU) illustrates how different functional blocks are organized and interconnected to perform address translation. It provides a high-level view of the system architecture without focusing on internal decision-making steps. This representation helps in understanding how data flows between various components in the MMU.

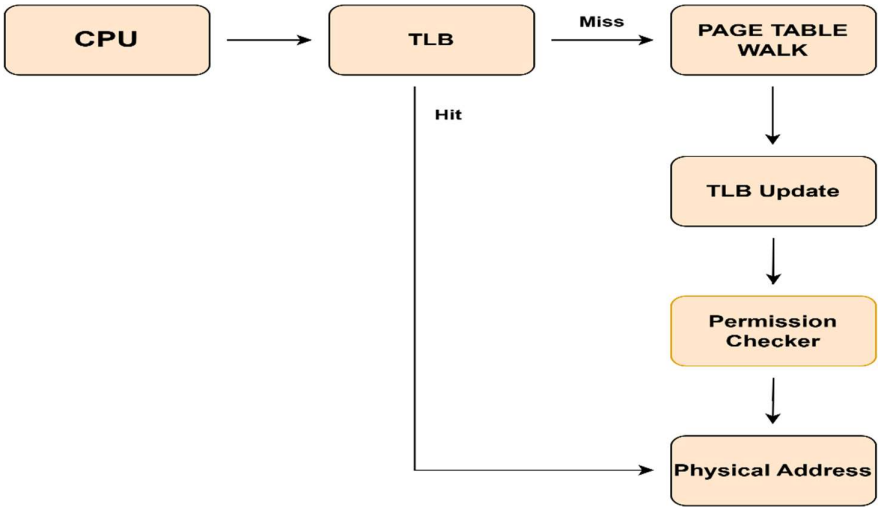


Fig 3; Structural representation of MMU

The process begins with the CPU, which generates virtual addresses during program execution. These addresses are forwarded to the MMU, where the Translation Lookaside Buffer (TLB) is accessed first. The TLB is a high-speed memory that stores recently used virtual-to-physical address mappings.

If the required mapping is available in the TLB, it results in a TLB hit. In this case, the physical address is generated directly without accessing any additional components. This direct path improves system performance by reducing memory access time.

If the mapping is not available in the TLB, it results in a TLB miss. In this situation, the system performs a page table walk to retrieve the correct mapping from the page table. The page table contains complete information about all virtual-to-physical address mappings.

The Page Table Walker (PTW) is activated when a TLB miss occurs, meaning the required address mapping is not available in the Translation Lookaside Buffer. It is responsible for accessing the page table to retrieve the correct mapping. The PTW takes the Virtual Page Number (VPN) as input and searches the page table to find the corresponding Physical Page Number (PPN). Since the page table is stored in main memory, this process takes more time compared to TLB lookup.

After retrieving the mapping, the PTW forwards the Physical Page Number to the next stage of the system. It also updates the TLB with the new entry so that future accesses can be handled quickly After retrieving the mapping from the page table, the TLB is updated with the new entry. This ensures that future accesses to the same address can be handled quickly through a TLB hit.

Once the Physical Page Number is obtained, the system performs a permission check to verify whether the requested memory operation is valid. This step ensures that only authorized access is allowed, thereby maintaining system security.

Another important observation from the structure is the flow of data between components. Each block is connected in a way that ensures smooth transfer of information without conflict. The MMU Controller manages this flow effectively, making sure that the correct path is selected and the required operations are performed in sequence. The inclusion of both update and validation stages indicates that the MMU is not just a translation unit but also a control and optimization unit. The update mechanism improves future performance, while the permission check ensures safe operation. These features together make the system robust and efficient.

Address Translation Process in MMU

The given flow represents the working of a Memory Management Unit (MMU), which converts virtual addresses generated by the CPU into physical addresses used by main memory. This process is

essential for efficient memory utilization and supports multiple programs running

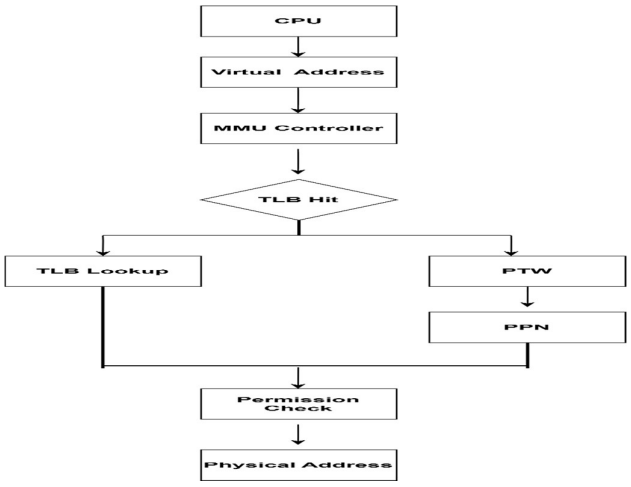


Fig 4: Address Translation Process in MMU

CPU and Virtual Address

The CPU generates virtual addresses during program execution. These addresses are not directly used by memory and must be translated. A virtual address is divided into two parts: Virtual Page Number (VPN) and offset. The VPN identifies the page, while the offset specifies the exact location within that page.

Simulation Results

The simulation of the proposed Memory Management Unit (MMU) was carried out using Verilog HDL to verify its functionality. Different virtual address inputs were applied to the system to observe the behavior of address translation under various conditions.

During simulation, the system first checks the Translation Lookaside Buffer (TLB) for the required mapping. When the mapping is available, a TLB hit occurs, and the physical address is generated quickly. This demonstrates the efficiency of the caching mechanism in reducing memory access time.

In cases where the mapping is not available in the TLB, a TLB miss occurs. The system then accesses the page table to retrieve the correct mapping. Although this introduces a small delay, it ensures accurate address translation. After retrieving the mapping, the TLB is updated for future accesses.

The simulation results clearly show the working of both TLB hit and TLB miss conditions. The generated physical address matches the expected output for all given inputs, confirming the correctness of the design.

TLB Waveform

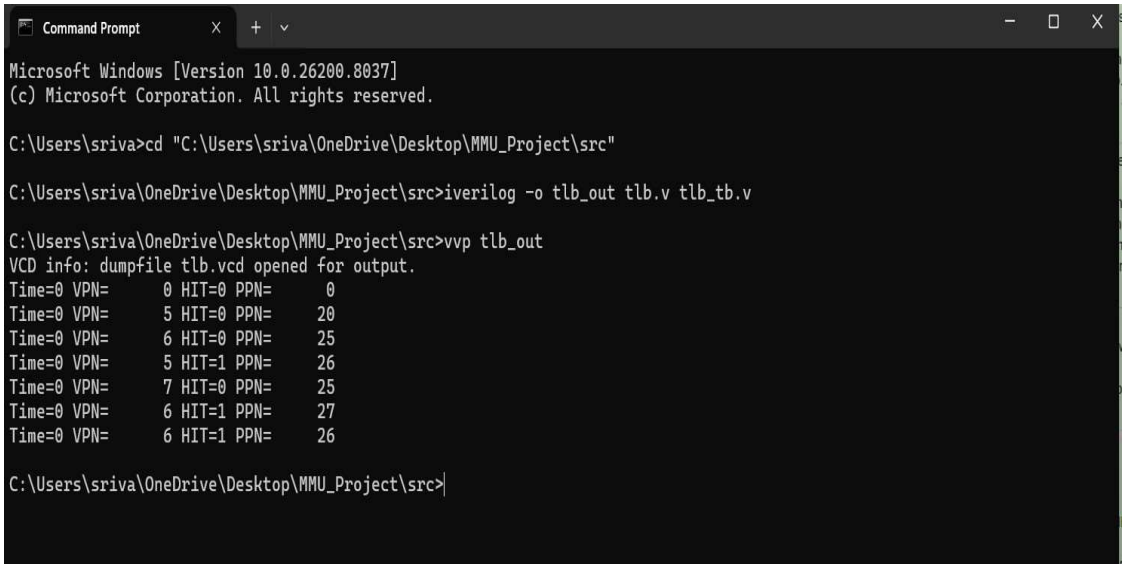


Fig 5 Compilation and Simulation of TLB using Icarus Verilog

The command prompt output shows the simulation results of the TLB module. It displays different Virtual Page Number (VPN) inputs along with the corresponding hit signal and Physical Page Number (PPN) outputs.

Initially, when the VPN value is 0, the hit signal is 0 and the PPN is also 0. This indicates that there is no mapping available in the TLB for this address.

When the VPN changes 5 and 6, the hit signal remains 0, which indicates TLB miss condition conditions. During these cases, the PPN values such as 20 and 25 are observed. These are not valid mappings but represent intermediate or previously stored values in the output value register.

Later, when the VPN value repeats (for example VPN = 5 again), the hit signal becomes 1. This indicates that the mapping is now available in the TLB. The corresponding PPN value (such as 26) is now valid and represents correct address translation. Similarly, for VPN = 6, hit signal becomes 1 later cycles, showing that the TLB successfully stored and reused mapping. This demonstrates caching behavior of the TLB. For VPN values that are not present in the TLB (such as VPN = 7), the hit signal remains 0, indicating miss condition

Fig 6; TLB waveform

The waveform represents the simulation results of the Translation Lookaside Buffer (TLB). It shows how different signals vary with time and how the TLB performs virtual-to-physical address translation based on input conditions.

Clk: The clock (clk) signal is a periodic waveform that controls the timing of the entire system. All operations in the TLB occur in synchronization with the clock edges. The regular toggling of the clock indicates that the simulation is running correctly.

rst_n: The reset (rst_n) signal is active low and is used to initialize the system. At the beginning of the simulation, the reset signal is low, which sets all outputs to their default state. After a few cycles, it becomes high, allowing the system to start normal operation.

The virtual page number (vpn[19:0]) acts as the input to the TLB. It changes over time to represent different memory access requests. Values such as 00000, 00005, 00006, and 00007 can be observed in the waveform, indicating that multiple test cases are applied to verify the functionality of the TLB.

Hit: The hit signal (hit) indicates whether the required mapping is found in the TLB. When the hit signal is high, it represents a TLB hit, meaning the mapping is directly available. When the hit signal is low, it indicates a TLB miss.

The physical page number (ppn[19:0]) represents the output of the TLB. For each input VPN, a corresponding PPN is generated. Values such as 00014, 00019, and 0001A indicate successful address translation.

Virtual Page Number (VPN): The Virtual Page Number (vpn[19:0]) is the input given to the TLB. In the waveform, the VPN values change over time (such as 00000, 00005, 00006), representing different memory access requests.

Each time the VPN changes, the TLB checks for a matching entry. Based on this, the system generates a hit or miss and produces the corresponding output.

Physical Page Number (PPN): The physical page number (ppn[19:0]) represents the output of the TLB. For each input VPN, a corresponding PPN is generated. Values such as 00014, 00019, and 0001A indicate successful address translation. An important observation in the waveform is that the PPN signal may still show values even when the hit signal is low. This occurs because the output register may retain its previous value or display intermediate values. However, these values are not valid unless the hit signal is high. Therefore, the PPN should be considered valid only when the hit signal is equal to 1.

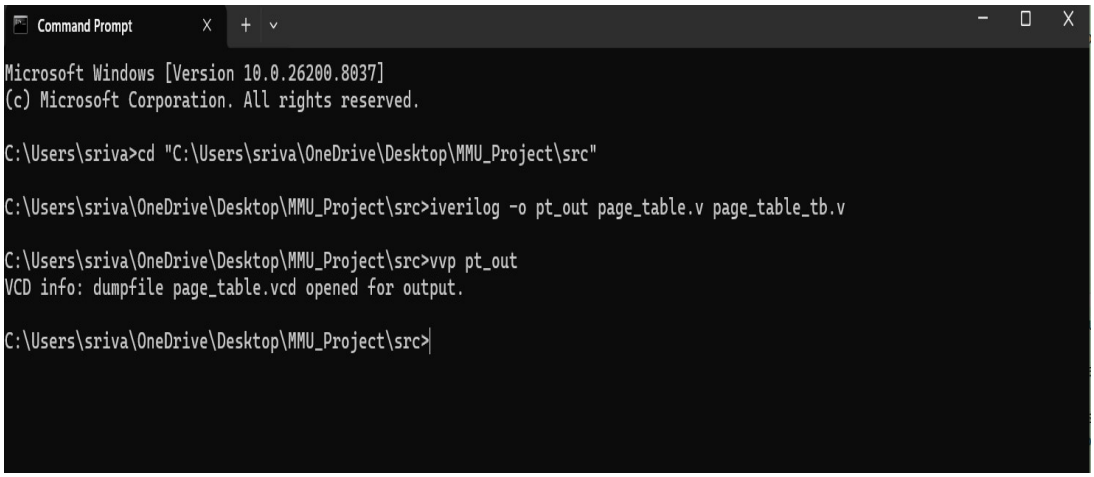


Fig 7; Compilation and Simulation of Page table using Icarus Verilog

The above command prompt output shows the successful execution of the Page Table module using Icarus Verilog. This step is an important part of the simulation process, where the Verilog code is compiled and executed to verify the functionality of the design.

Initially, the iverilog command is used to compile the source files of the Page Table (page_table.v) along with its testbench (page_table_tb.v). This step checks for syntax errors and converts the Verilog code into an executable simulation file.

After successful compilation, the vvp command is used to run the simulation. During execution, the system processes the input values defined in the testbench and performs the required operations based on the logic implemented in the Page Table module.

The message "VCD info: dumpfile page_table.vcd opened for output" indicates that the simulation has started generating a Value Change Dump (VCD) file. This file stores all signal transitions over time and is used for waveform analysis. The absence of errors in the command prompt confirms that the design is correctly implemented and ready for further verification using waveform visualization.

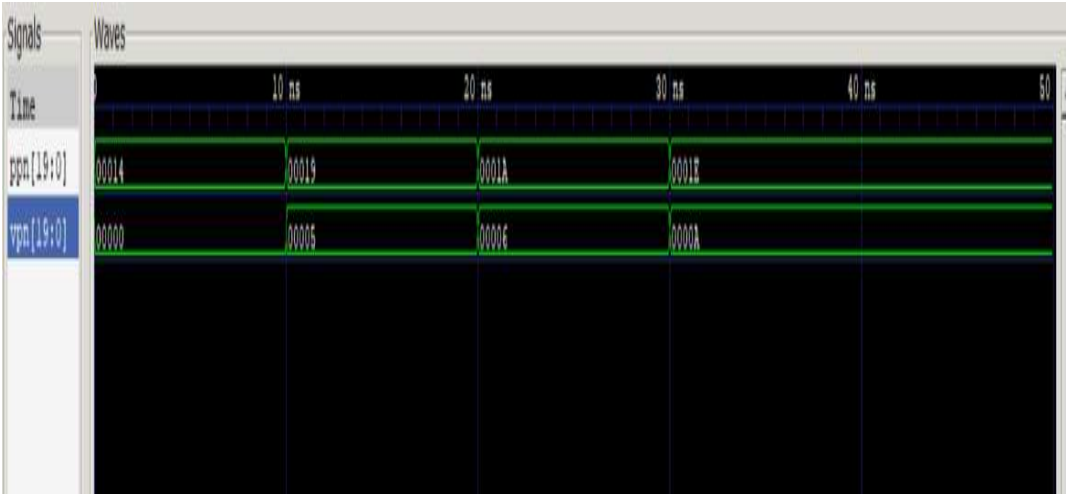
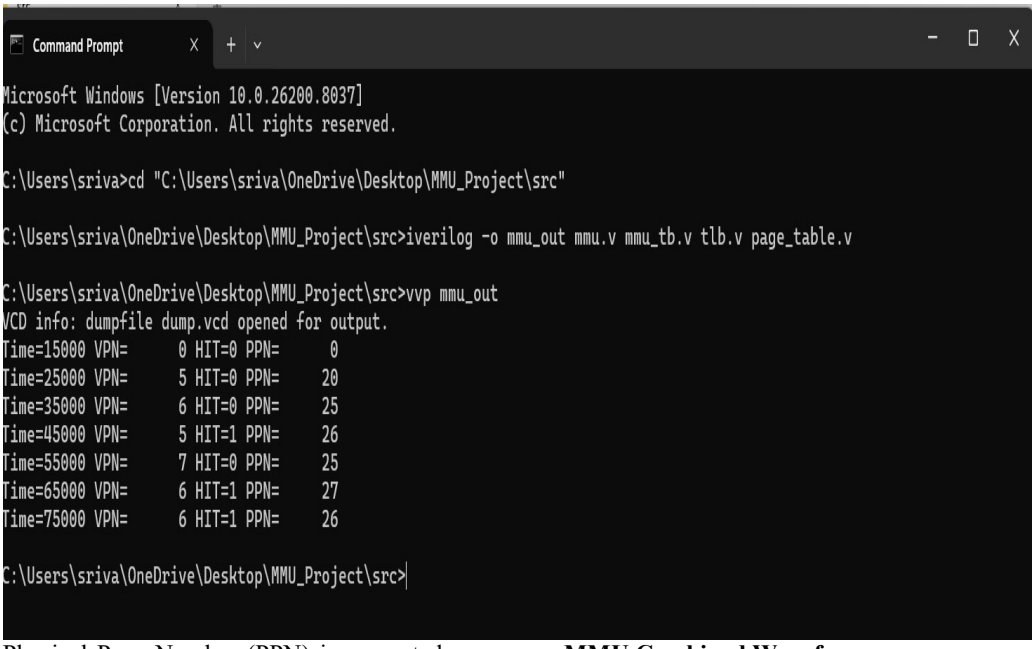


Fig 5.4 Page Table Waveform

The above waveform represents the simulation results of the Page Table module. It shows how the system performs address translation by mapping Virtual Page Numbers (VPN) to Physical Page Numbers (PPN) using the page table. The clock signal controls the timing of the system and ensures that all operations occur in a synchronized manner. The reset signal initializes the system at the beginning of the simulation, after which normal operation starts. The Virtual Page Number (VPN) acts as the input to the Page Table. As the VPN values change over time, the system searches the page table for the corresponding mapping. Each VPN represents a different memory access request. the output of the Page Table. For every valid VPN input, the corresponding PPN is produced based on the mapping stored in the page table. This confirms that the Page Table correctly performs address translation.

Unlike the TLB, the Page Table does not depend on hit or miss conditions. It directly accesses the mapping from memory, which ensures that the correct output is always generated. However, this process takes more time compared to TLB lookup because the page table is stored in main memory. The waveform demonstrates that for each VPN input, a corresponding PPN is generated without any dependency on cache-like behavior.



The Physical Page Number (PPN) is generated as **MMU Combined Waveform**

Fig 5.5 Compilation and Simulation of MMU using Icarus Verilog

The above command prompt output represents the simulation results of the complete Memory Management Unit (MMU). It shows how the system processes different Virtual Page Numbers (VPN) and generates the corresponding Physical Page Numbers (PPN) along with the hit signal. The simulation begins with an initial time delay, after which different VPN values are applied sequentially. At time = 15000, the VPN value is 0, and the hit signal is 0, indicating that no mapping is found in the TLB. The PPN is also 0, representing the default state. At time = 25000 and 35000, the VPN values change to 5 and 6 respectively. In both cases, the hit signal remains 0, which indicates TLB miss conditions. During these miss conditions, the system relies on the Page Table to retrieve the mapping. The PPN values such as 20 and 25 are observed, showing that the Page Table provides the required physical address.

At time = 45000, the VPN value returns to 5. This time, the hit signal becomes 1, indicating a TLB hit. This shows that the mapping has been stored in the TLB after the previous access. The PPN value (26) is now directly obtained from the TLB, resulting in faster access. At time = 55000, the VPN value changes to 7, and the hit signal becomes 0 again, indicating a TLB miss. The system once again uses the Page Table to obtain the mapping.

At time = 65000 and 75000, the VPN value is 6, and the hit signal becomes 1. This indicates that the mapping for VPN = 6 is now available in the TLB. The corresponding PPN values (27 and 26) confirm that the TLB is being updated and reused effectively.Even when the hit signal is 0, the PPN values are still displayed. These values are obtained from the Page Table and are valid in the context of

MMU operation. However, the hit signal indicates whether the mapping was obtained from the TLB or not.

hit = 1 → Mapping from TLB (fast access)
hit = 0 → Mapping from Page Table (slower access)

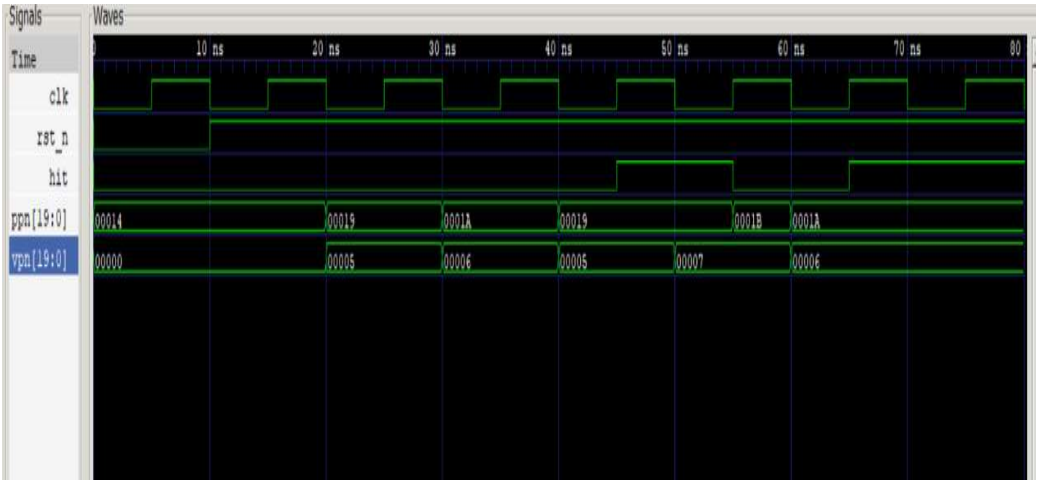


Fig 5.6 Fig MMU Combined Waveform

The above waveform represents the combined simulation of the complete Memory Management Unit (MMU), which integrates both the Translation Lookaside Buffer (TLB) and the Page Table. It shows how the system processes virtual addresses and generates corresponding physical addresses over time. The clock (clk) signal controls the timing of the entire system and ensures that all operations are synchronized. The reset (rst_n) signal initializes the system at the beginning, after which the MMU starts normal operation. The Virtual Page Number (vpn[19:0]) acts as the input to the MMU. In the waveform, the VPN values change sequentially (such as 00000, 00005, 00006, 00007), representing different memory access requests generated by the CPU. The Physical Page Number (ppn[19:0]) is the final output of the MMU. For each VPN input, a corresponding PPN is generated. The values such as 00014, 00019, 0001A indicate successful address translation. The hit signal (hit) plays a key role in understanding the system behavior. When the hit signal is high, it indicates that the mapping is found in the TLB, and the PPN is generated quickly. When the hit signal is low, it indicates a TLB miss, and the system retrieves the mapping from the Page Table.
Initially, for new VPN values, the hit signal is low, indicating that the TLB does not contain the mapping. During this time, the Page Table provides the correct PPN. After this, the TLB gets updated with the new mapping.

Conclusion

The project successfully demonstrates the design and implementation of a Memory Management Unit (MMU) using Verilog Hardware Description Language. The main objective of converting virtual addresses into physical addresses is achieved effectively through the integration of the Translation Lookaside Buffer (TLB) and Page Table.
The system performs address translation accurately under both TLB hit and TLB miss conditions. In the case of a TLB hit, the physical address is generated quickly, which improves system performance. During a TLB miss, the page table is accessed to retrieve the required mapping, ensuring correctness of the translation process.
The simulation results and waveform analysis confirm that the system behaves as expected. All input virtual addresses are correctly translated into corresponding physical addresses. The use of TLB as a caching mechanism significantly reduces memory access time and enhances overall efficiency.
The modular design approach using separate components for TLB, Page Table, and MMU makes the system easy to understand and implement. This project provides a clear understanding of memory management concepts and their practical implementation in hardware design.

Future Scope

The current MMU design provides a basic understanding of address translation and memory management. However, there are several improvements and extensions that can be implemented in the future to enhance the system.
Future enhancements may include the implementation of multi-level page tables to support larger memory systems. Advanced TLB replacement algorithms such as Least Recently Used (LRU) can be added to improve performance further.
The system can also be extended to include memory protection features, such as access control and

privilege levels, to improve security. Additionally, support for larger address spaces and dynamic memory allocation can be incorporated. Integration with cache memory and advanced processor architectures can further improve system performance. The design can also be implemented on FPGA hardware for real-time testing and validation.

References

[1] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 5th ed. Burlington, MA, USA: Morgan Kaufmann, 2014.
[2] D. A. Patterson and J. L. Hennessy, *Computer Architecture: A Quantitative Approach*, 6th ed. San Francisco, CA, USA: Morgan Kaufmann, 2017.
[3] S. Palnitkar, *Verilog HDL: A Guide to Digital Design and Synthesis*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2003.
[4] M. Morris Mano and M. D. Ciletti, *Digital Design*, 5th ed. Upper Saddle River, NJ, USA: Pearson Education, 2013.
[5] W. Stallings, *Computer Organization and Architecture: Designing for Performance*, 10th ed. Boston, MA, USA: Pearson, 2016.
[6] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed. Waltham, MA, USA: Elsevier, 2012.
[7] Xilinx Inc., *Vivado Design Suite User Guide: Logic Simulation*, San Jose, CA, USA, 2023.
[8] Intel Corporation, *Introduction to Memory Management and Address Translation*, Intel Technical Documentation, 2022.