Publisher
Vandana Publications
www.vandanapublications.com

International Journal of Engineering and Management Research

2025 Volume 15 Number 2 April

E-ISSN:2250-0758
P-ISSN:2394-6962

Research Article

Verilog HDL

Design and Implementation of AHB to APB Bridge using Verilog

Bellerimath PS[1*], Shirakol S[2], Gunari LV[3], N Rachana[4], Mahat SP[5], Arali C[6]

DOI:10.5281/zenodo.15478857

[1*] Preeti S Bellerimath, Assistant Professor, Department of Electronics and Communication Engineering, SDM College of Engineering and Technology, Dharwad, Karnataka, India.

[2] Shrikanth Shirakol, Assistant Professor, Department of Electronics and Communication Engineering, SDM College of Engineering and Technology, Dharwad, Karnataka, India.

[3] Laxmi Vadiraj Gunari, Student, Department of Electronics and Communication Engineering, SDM College of Engineering and Technology, Dharwad, Karnataka, India.

[4] N Rachana, Student, Department of Electronics and Communication Engineering, SDM College of Engineering and Technology, Dharwad, Karnataka, India.

[5] Sahil P Mahat, Student, Department of Electronics and Communication Engineering, SDM College of Engineering and Technology, Dharwad, Karnataka, India.

[6] Chirag Arali, Student, Department of Electronics and Communication Engineering, SDM College of Engineering and Technology, Dharwad, Karnataka, India.

Efficient on-chip communication is vital in modern embedded and System-on-Chip (SoC) architectures. This project presents the design and implementation of an AHB (Advanced High-performance Bus) to APB (Advanced Peripheral Bus) bridge using Verilog Hardware Description Language (HDL). The bridge serves as an interface between high-speed AHB masters and low-speed APB peripherals, with a Finite State Machine (FSM) governing the control flow—including address decoding, data transfer, and handshake signal management. The Verilog-based design emphasizes modularity, timing accuracy, and hardware compatibility, making it well-suited for both FPGA prototyping and ASIC integration. Functional simulation and synthesis validate the bridge's correctness, performance, and efficient resource utilization. The FSM-based control ensures predictable and reliable communication across the AHB and APB domains, fulfilling the structured connectivity demands of AMBA-based SoC designs [1].

Keywords: Verilog HDL, FSM, AHB to APB Bridge, SoC Communication, AMBA Protocol, Embedded Systems, Bus Interface, FPGA, Data Transfer

Corresponding Author

Preeti S Bellerimath, Assistant Professor, Department of Electronics and Communication Engineering, SDM College of Engineering and Technology, Dharwad, Karnataka, India.
Email: prtbellerimath69@gmail.com

How to Cite this Article

Bellerimath PS, Shirakol S, Gunari LV, N Rachana, Mahat SP, Arali C, Design and Implementation of AHB to APB Bridge using Verilog. Int J Engg Mgmt Res. 2025;15(2):199-203.
Available From
https://ijemr.vandanapublications.com/index.php/j/article/view/1749

To Browse

Manuscript Received	Review Round 1	Review Round 2	Review Round 3	Accepted
2025-03-13	2025-04-01			2025-04-22

Conflict of Interest	Funding	Ethical Approval	Plagiarism X-checker	Note
None	Nil	Yes	4.93	

1. Introduction

In System-on-Chip (SoC) architectures, efficient communication between high-speed cores and low-speed peripherals is crucial. The ARM AMBA (Advanced Microcontroller Bus Architecture) specification addresses this need with a range of bus protocols, among which AHB (Advanced High-performance Bus) and APB (Advanced Peripheral Bus) are commonly used [1],[6]. While AHB supports high-throughput operations suitable for processors and memory, APB is designed for simple, low-power peripheral communication. To enable seamless interaction between these two domains, a bridge is required. This project focuses on the design and implementation of an AHB to APB bridge using Verilog HDL. The bridge translates high-speed AHB transactions into APB-compatible signals, ensuring proper timing and control through a Finite State Machine (FSM). The design adheres to the AMBA protocol, aiming for accuracy, modularity, and synthesis efficiency. The Fig 1 represents the AMBA architecture.

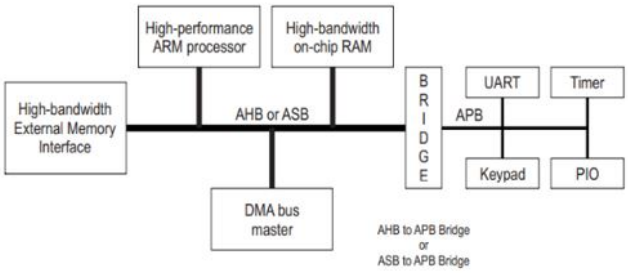

Figure 1: Block diagram of AMBA Architecture [1]

2. AHB Master

The AHB master initiates all bus transactions by generating address, control, and data signals. It controls the direction and timing of data transfers using handshake signals such as HREADY and HRESP, which indicate the readiness of the slave to respond. The master supports both read and write operations and drives the bus protocol's pipeline behavior. In the bridge design, the AHB master acts as the transaction initiator, enabling high-throughput communication suitable for cores and memory components.

3. AHB Slave

The AHB slave responds to commands issued by the master. During a transaction, it decodes the address to determine whether it should respond, and either accepts data (in write operations) or returns data (in read operations). It also generates HREADY and HRESP to indicate the status of each transaction. In this design, the AHB slave interface receives pipelined control signals and forwards them to the FSM controller. Verilog HDL modeling ensures precise timing simulation and synthesis compatibility.

4. FSM Controller

In the AHB to APB bridge design, a Finite State Machine (FSM) is used to manage the control flow of transactions between the high-speed AHB interface and the low-power APB interface. The FSM ensures that data transfers are properly sequenced, and it enforces timing and handshaking requirements specific to each bus [2], [7]. It operates based on key control signals such as Valid, HWRITE, and a temporary write flag (HwriteReg), transitioning through various states that define the phases of read and write transactions.

The FSM consists of eight distinct states, each responsible for a specific operation or synchronization step. The overall behavior and transitions are illustrated in the FSM state diagram (Fig. 2), and are described as follows:

- ST_IDLE: This is the initial or idle state, entered after reset (HRESETn = 0) or when no valid AHB transaction is occurring (Valid = 0). The FSM remains here until a transaction is detected.
- ST_READ: Triggered when Valid = 1 and HWRITE = 0, this state captures the address and control information for a read operation. The FSM transitions from ST_IDLE to ST_READ when a read is initiated.
- ST_RENABLE: Follows ST_READ to initiate the APB read phase. It enables the APB bus and waits for the data to become available. The FSM then returns to ST_IDLE once the read is complete.
- ST_WRITE: Entered when Valid = 1 and HWRITE = 1, signalling the beginning of a write transfer. This state prepares the system for data transmission by latching the address and data inputs.
- ST_WWAIT (Write Wait): This intermediate state is activated when a write is initiated but HwriteReg = 0, indicating that the write enable timing is not yet met. It acts as a synchronization buffer before proceeding.

- ST_WRITEP (Write Pipeline): Represents the state where a pipelined write is handled (Valid = 1, HwriteReg = 1). The FSM uses this state to process overlapping write commands, improving throughput in successive transfers.
- ST_WENABLE: When Valid = 0 and HwriteReg = 1, the FSM enters this state to assert APB control signals and finalize the write operation. It ensures proper APB handshake signalling.
- ST_WENABLEP (Write Enable Pipeline): Similar to ST_WENABLE but used for pipelined transfers (Valid = 1, HwriteReg = 1). It allows seamless continuation of back-to-back write operations.

The FSM transitions are governed by control signals such as Valid, *HWRITE*, and HwriteReg. From *ST_IDLE*, it moves to *ST_READ* or *ST_WRITE* based on the transaction type. Read operations proceed to *ST_RENABLE*, while writes may go through *ST_WWAIT*, *ST_WRITEP*, and then *ST_WENABLE* or *ST_WENABLEP*. All sequences eventually return to *ST_IDLE*.

This approach ensures correct timing, supports pipelining, and maintains AMBA protocol compliance [1][2][7].

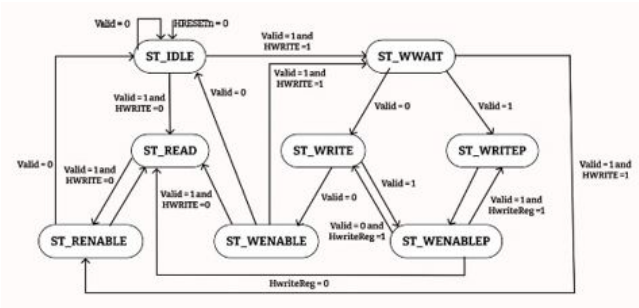

Figure 2: State machine of AHB to APB [1]

5. APB Bus

The Advanced Peripheral Bus (APB) is optimized for simple, low-power peripheral communication. Unlike the pipelined AHB, APB uses a straightforward two-phase protocol: setup and enable. This makes it ideal for connecting low-speed devices such as UARTs, timers, and GPIOs [5], [8]. In the proposed design:

- Setup Phase: The FSM controller issues control signals such as PADDR, PWRITE, and PSELx, preparing the APB slave for a transaction.
- Enable Phase: PENABLE is asserted, and the actual read or write operation occurs. The APB slave acknowledges the transaction via the PREADY signal.

The minimalistic nature of APB makes it highly suitable for connecting UARTs, timers, GPIOs, and other memory-mapped peripheral devices. The APB slave used in this work follows AMBA-defined timing and handshake behavior [1][7].

6. AHB to APB Architecture

The AHB to APB bridge enables communication between the high-speed AHB and low-power APB protocols. As shown in Figure 3, the system consists of an AHB Slave, a Bridge FSM, and an APB Interface. The AHB Slave receives address, data, and control signals from the AHB master and forwards pipelined signals to the FSM. The FSM generates intermediate APB control signals like paddr_temp, pwrite_temp, and penable_temp, which are then passed to the APB Interface to drive the final APB signals. This architecture ensures synchronized and efficient data exchange between AHB and APB domains [3], [7], [8].

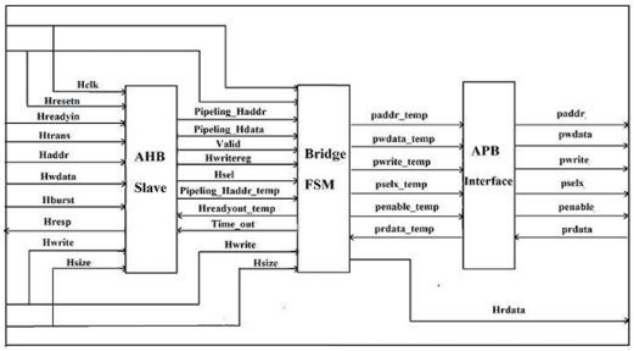

Figure 3: Architecture of AHB to APB Bridge [1]

7. Results

The simulation confirms the correct functionality of the AHB to APB bridge. Both write and read operations are executed with proper handshaking, and the FSM ensures correct signal sequencing. Waveform results validate the design for SoC integration.

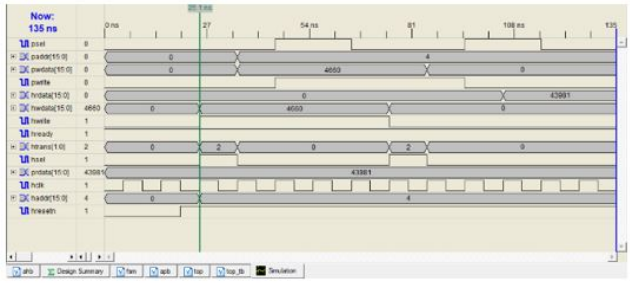

Figure 4: Simulation results from write operation

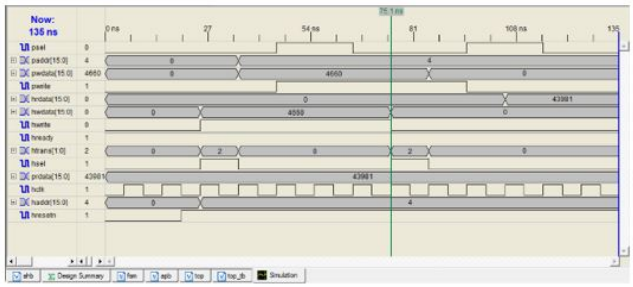

Figure 5: Simulation results from read operation

The write operation in Figure 4 starts at 27 ns with valid address, data, and control signals (hwrite=1, htrans=2), and completes by 81 ns after the APB handshake (psel, penable). The read operation in Figure 5 begins at 81 ns with hwrite=0 and completes by 135 ns when data appears on hrdata, confirming proper timing and synchronization between AHB and APB.

Area Report:

Logic Utilization	Used	Available	Utilization
Number of Slices	31	768	4%
Number of Slice Flip Flops	6	1536	0%
Number of 4 input LUTs	5	1536	0%
Number of bonded IOBs	105	124	84%
Number of GCLKs	1	8	12%

Timing Report:

Minimum period	3.746ns (Maximum Frequency: 266.923MHz)
Minimum input arrival time before clock	4.279ns
Maximum output required time after clock	6.306ns

8. Conclusion

This paper presents a Verilog HDL-based implementation of an AHB to APB bridge using an FSM controller to efficiently manage communication between a high-performance AHB master and a low-power APB peripheral. The synthesis results confirm excellent area efficiency, with only 4% of slices used, less than 1% utilization of flip-flops and LUTs, and 84% utilization of bonded IOBs. The timing report shows a minimum clock period of 3.746 ns, corresponding to a maximum frequency of 266.92 MHz. This low delay significantly enhances the operational speed of the design, making it well-suited for high-performance embedded systems. Furthermore, the design meets all input and output timing constraints, ensuring stable and reliable performance.

The modular FSM-based control structure also provides flexibility for future enhancements such as multi-peripheral support, pipelined AHB transfers, and AXI interface compatibility, paving the way for scalable and adaptable SoC integration.

References

[1] Prarthi Bhatt, & Prof. Devang Shah. (2018). Design Amba based AHB to APB bridge using verilog HDL. *Journal of Emerging Technologies and Innovative Research (JETIR)*.

[2] Prof. Ravi Mohan Sairam Prof. Sumit Sharma, & Miss. Geeta Pal. (2013). FSM & handshaking based AHB to APB bridge for high speed systems. *International Journal of Engineering Research & Technology (IJERT), 2*.

[3] N. Venkateswara Rao, PV Chandrika, Abhishek Kumar, & Sowmya Reddy. (2020). Design of AMBA based AHB2APB protocol for efficient utilization of AHB and APB. *International Research Journal of Engineering and Technology (IRJET)*.

[4] Soumya Rai, & Virendra Pratap Yadav. (2021). Comparison between ARM AMBA Protocols and Verification of APB Protocol Using System Verilog & UVM. *International Research Journal of Engineering and Technology (IRJET)*.

[5] Shankar, Dipti Girdhar, & Neeraj Kr. Shukla. (2014). Design and verification of AMBA APB protocol. *International Journal of Computer Applications, 95*(21).

[6] Deeksha L, & Shivakumar B.R. (2019). Effective design and implementation of AMBA AHB bus protocol using Verilog. *International Conference on Intelligent Sustainable Systems, IEEE Xplore Part Number: CFP19M19-ART*. ISBN: 978-1-5386-7799-5.

[7] Abhijeet Paunikar, Rohan Gavankar, Nachiket Umarikar, & K Sivasankaran. (2014). Design and implementation of area efficient, low power AMBA-APB Bridge for SoC. *International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE)*, pp. 1-6.

[8] G Prathibha, & Ambika Sekhar. (2013). APB bridge based on AMBA 4.0. *International Journal of Engineering Research & Technology (IJERT), 2*(9).

[9] Jaehoon Song, et al. (2009). An efficient SOC test technique by reusing on/off chip bus bridge. *IEEE Transactions on Circuits and Systems-I: Regular Papers, 56*(3).

[10] Krishna Sekar, et al. (2008). Dynamically configurable bus topologies for high performance on-chip communication. *IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, 16*(10).

[11] Vani.R.M, & Merope. (2010). Design of AMBA based AHB2APB bridge. *IJCSNS, 10*(11).

[12] Samir Palnitkar. (2008). *Verilog HDL: A guide to digital design and synthesis*. (2nd ed.). Pearson.

[13] Shanthi V A. (2023). AHB2APB bridge verification. *Maven Silicon*.

[14] Niraj Kumar Mishra, Dr. R. P. Singh, & Dr. Jaikarn Singh. (2016). AMBA-APB RTL implementation using efficient power and constructs through Verilog. *IJSRSET, 2*(3).

[15] A. Angadi, S. Umarani, T. Sajjanar, R. Karnam, K. Marali, & S. Shirakol. (2023). Architectural design of built in self-test for VLSI circuits using LFSR. *International Conference on Applied Intelligence and Sustainable Computing (ICAISC)*, pp. 1-7. DOI: 10.1109/ICAISC58445.2023.10200604.