AXI to APB Bridge Documentation

Ali Hamza
Muhammad Attaullah Khan
Shahzad Akhter

Nust Chip Design Centre

December 17, 2024

Contents

Chapter 1

Introduction

Modern embedded systems often combine high-speed components with low-power peripherals, which requires efficient communication between different bus protocols. The Advanced eXtensible Interface (AXI) protocol is a high-performance, pipelined, burst-oriented interface commonly used in processors and memory subsystems. In contrast, the Advanced Peripheral Bus (APB) protocol is designed for low-power, simple, and latency-insensitive communication with peripherals.

Integrating AXI-based devices with APB-based peripherals presents a challenge due to significant differences in their protocol characteristics. An AXI-to-APB bridge addresses this challenge by acting as a protocol converter, translating AXI transactions into APB-compatible operations while maintaining compliance with both protocols.

The objective of this project is to design and implement an AXI to APB bridge that supports AXI burst modes (FIXED, INCREMENTING, and WRAP), ensuring seamless and efficient communication between AXI and APB domains.

This document provides a detailed description of the functional specifications, architectural design, implementation, and verification of the bridge. The Results and Analysis section highlights the performance of the bridge, followed by conclusions and potential future improvements.

Chapter 2

Background

The Advanced Microcontroller Bus Architecture (AMBA) protocols, developed by ARM, have become the standard for interconnect solutions in modern System-on-Chip (SoC) designs. Among these protocols, the Advanced eXtensible Interface (AXI) Advanandced Peripheral Bus (APB) play critical roles in facilitating communication between components with different performance and power requirements.

- **AXI (Advanced eXtensible Interface)**: The AXI protocol supports high-performance, high-frequency system designs for communication between Manager and Subordinate components. The AXI protocol features are:
  - **AXI (Advanced eXtensible Interface)**: The AXI protocol supports high-performance, high-frequency system designs for communication between Manager and Subordinate components. The AXI protocol features are:
    - Suitable for high-bandwidth and low-latency designs.
    - High-frequency operation is provided without using complex bridges.
    - The protocol meets the interface requirements of a wide range of components.
    - Suitable for memory controllers with high initial access latency.
    - Flexibility in the implementation of interconnect architectures is provided.
    - Backward-compatible with AHB and APB interfaces.
  - **The key features of the AXI protocol are:**
    - Separate address/control and data phases.
    - Support for unaligned data transfers using byte strobes.
    - Uses burst-based transactions with only the start address issued.
    - Separate write and read data channels that can provide low-cost Direct Memory Access (DMA).
    - Support for issuing multiple outstanding addresses.
    - Support for out-of-order transaction completion.
    - Permits easy addition of register stages to provide timing closure.
  - **APB (Advanced Peripheral Bus)**: The APB protocol is a low-cost interface, optimized for minimal power consumption and reduced interface complexity. The APB protocol features are:
    - Low-cost interface designed for minimal power consumption.
    - Reduced interface complexity compared to other protocols.
    - Non-pipelined, simple, and synchronous protocol.
    - Each transfer requires at least two cycles to complete.
    - Designed for accessing programmable control registers of peripheral devices.
    - APB peripherals are typically connected to the main memory system using an APB bridge.
  - **APB transfer characteristics:**

  - Transfers are initiated by an APB bridge (Requester).
  - Peripheral interfaces respond to requests (Completers).
  - APB bridges connect APB peripherals to higher-bandwidth systems like AXI.
  - Example: A bridge from AXI to APB connects APB peripherals to an AXI memory system.

- **AXI (Advanced eXtensible Interface)**: The AXI protocol supports high-performance, high-frequency system designs for communication between Manager and Subordinate components. The AXI protocol features are:
  - Suitable for high-bandwidth and low-latency designs.
  - High-frequency operation is provided without using complex bridges.
  - The protocol meets the interface requirements of a wide range of components.
  - Suitable for memory controllers with high initial access latency.
  - Flexibility in the implementation of interconnect architectures is provided.
  - Backward-compatible with AHB and APB interfaces.
- The key features of the AXI protocol are:
  - Separate address/control and data phases.
  - Support for unaligned data transfers using byte strobes.
  - Uses burst-based transactions with only the start address issued.
  - Separate write and read data channels that can provide low-cost Direct Memory Access (DMA).
  - Support for issuing multiple outstanding addresses.
  - Support for out-of-order transaction completion.
  - Permits easy addition of register stages to provide timing closure.
- **APB (Advanced Peripheral Bus)**: The APB protocol is a low-cost interface, optimized for minimal power consumption and reduced interface complexity. The APB protocol features are:
  - Low-cost interface designed for minimal power consumption.
  - Reduced interface complexity compared to other protocols.
  - Non-pipelined, simple, and synchronous protocol.
  - Each transfer requires at least two cycles to complete.
  - Designed for accessing programmable control registers of peripheral devices.
  - APB peripherals are typically connected to the main memory system using an APB bridge.
- APB transfer characteristics:
  - Transfers are initiated by an APB bridge (Requester).
  - Peripheral interfaces respond to requests (Completers).
  - APB bridges connect APB peripherals to higher-bandwidth systems like AXI.
  - Example: A bridge from AXI to APB connects APB peripherals to an AXI memory system.

Integration of AXI and APB: Despite their individual advantages, the integration of AXI-based devices with APB-based peripherals presents challenges due to their differing protocol characteristics. The AXI to APB bridge addresses this gap by converting AXI transactions into APB-compatible operations. This conversion is essential for ensuring seamless communication in embedded systems and SoC designs.

This project focuses on designing an efficient AXI to APB bridge to support such integrations, ensuring compatibility while maintaining high performance and low power consumption.

Chapter 3

Functional Overview

The AXI to APB Bridge facilitates communication between AXI-based masters and APB-based peripherals. The bridge translates AXI protocol signals to the corresponding APB protocol signals, enabling interaction between these two protocols. Below are the functional specifications for the bridge.

3.1 General Overview

The AXI to APB Bridge translates the AXI protocol, commonly used for high-performance communication, to the simpler APB protocol, which is used for peripheral devices that do not require high-bandwidth access. The bridge handles both read and write operations, converting them between AXI and APB formats.

3.2 Signal Descriptions

3.2.1 AXI Interface Signals (Inputs to the Bridge)

- **Write Address Channel (AW)**
  - **AWADDR**: AXI write address for the transaction.
  - **AWBURST**: Burst type (INCR, WRAP, FIXED) indicating the type of burst for the transaction.
  - **AWLEN**: Burst length, indicating the number of beats in the burst.
  - **AWVALID**: AXI signal indicating that the write address is valid.
- **Read Address Channel (AR)**
  - **ARADDR**: AXI read address for the transaction.
  - **ARBURST**: Burst type (INCR, WRAP, FIXED) for the read transaction.
  - **ARLEN**: Burst length, indicating the number of beats in the read burst.
  - **ARVALID**: AXI signal indicating that the read address is valid.
- **Write Data Channel (WD)**
  - **WDATA**: Data to be written to the APB slave.
  - **WVALID**: AXI signal indicating that the write data is valid.
  - **WLAST**: AXI signal indicating that the last write beat has been sent.
- **Read Data Channel (RD)**
  - **RREADY**: AXI signal indicating that the master is ready to accept the read data.
- **Write Response Channel (B)**
  - **BREADY**: AXI signal indicating that the master is ready to accept the write response.

3.2.2 APB Interface Signals (Outputs from the Bridge)

– **Control and Timing Signals**

- **PADDR**: APB address to be accessed by the slave.
- **PWRITE**: APB write control signal, indicating whether the operation is a write (1) or read (0).
- **PSEL1, PSEL2, PSEL3, PSEL4**: APB select signals, used to select the target APB slave.
- **PENABLE**: APB enable signal, indicating that the APB transaction is valid.
- **PWDATA**: Data to be written to the APB slave.
- **PRDATA**: Data read from the APB slave.
- **PREADY**: APB ready signal, indicating that the APB slave is ready to respond.

3.3 Write Address Channel

In the Write Address Channel of the AXI protocol, the AXI master initiates a write transaction by asserting the `AWVALID` signal to indicate that the address is valid. The bridge responds by asserting `AWREADY`, completing the handshake between the master and the bridge. The `AWADDR` signal specifies the address to which the data will be written, and the `AWBURST` and `AWLEN` signals describe the burst characteristics.

This section describes the handshake mechanism involved in the Write Address Channel.

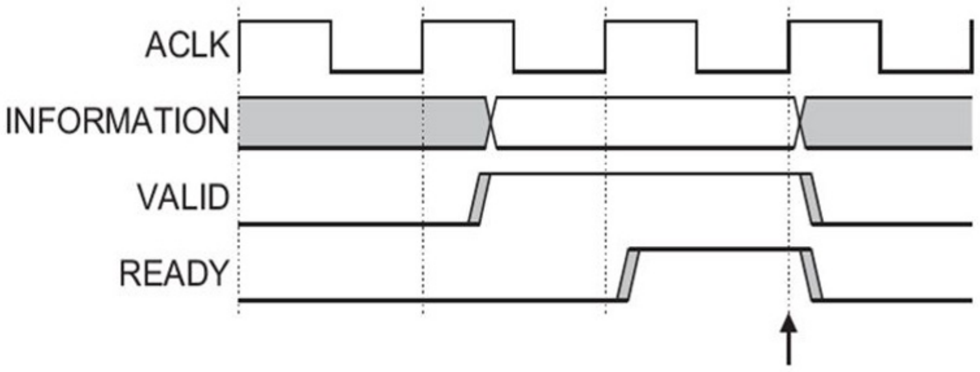


Figure 3.1: Write Address Channel Handshake Mechanism (Valid before Ready)

When the `AWVALID` signal is asserted by the AXI master, the bridge waits for the `AWREADY` signal to be asserted before proceeding with the transaction. The following signals are involved:

- `AWADDR`: Specifies the address for the write operation.
- `AWBURST`: Specifies the burst type (e.g., INCR, WRAP, FIXED).
- `AWLEN`: Specifies the burst length, indicating how many data beats will be transferred.
- `AWVALID`: Indicates that the write address is valid and ready to be processed.
- `AWREADY`: Indicates that the bridge is ready to accept the address from the AXI master.

Once the handshake is completed (when both `AWVALID` and `AWREADY` are asserted), the AXI master can proceed with the data transfer.

3.4 Write Data Channel (WD)

The Write Data Channel (WD) in the AXI protocol is responsible for transferring data from the AXI master to the APB slave. The key signals involved in the Write Data Channel are:

- `WVALID`: Asserted by the AXI master to indicate that valid write data (`WDATA`) is being sent.
- `WDATA`: The write data being sent by the AXI master.
- `WDREADY`: Asserted by the bridge to signal that it is ready to receive the write data.
- `WLAST`: Asserted by the AXI master when the last packet in the burst is being transmitted, indicating the end of the write operation.

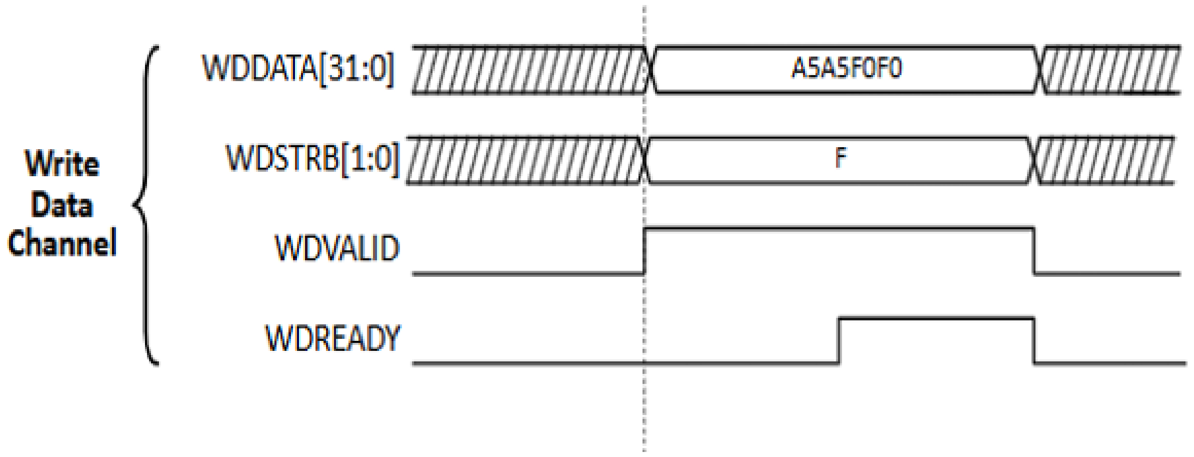


Figure 3.2: Write Data Channel (WD) Handshake

3.5 Write Response Channel (B)

The Write Response Channel (B) in the AXI protocol signals the completion and status of a write transaction. The key signals involved are:

- `BRESP`: Indicates the status of the write transaction.
  - ∗ 00 for `OKAY`: Transaction completed successfully.
  - ∗ 10 for `SLVERR`: Slave error, indicating an error in the write operation.
- `BVALID`: Asserted by the bridge to signal that the write response is valid and ready for the AXI master.
- `BREADY`: Driven by the AXI master to acknowledge the response.

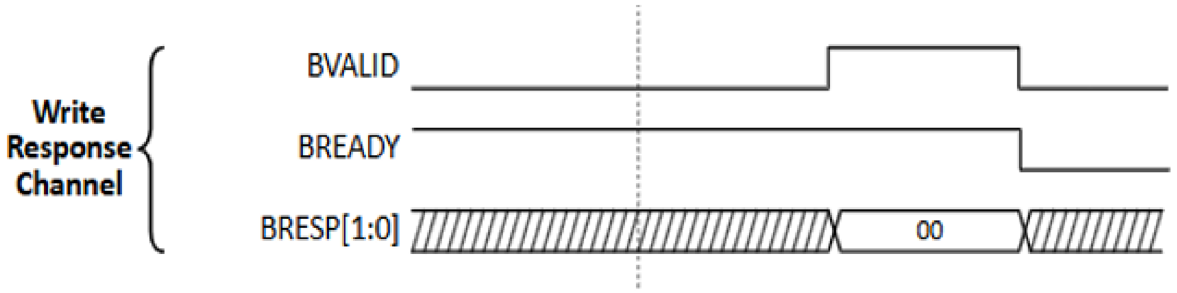


Figure 3.3: Write Response Channel (B) Handshake

3.6 Read Address Channel (AR)

The Read Address Channel (AR) in the AXI protocol is used to send the address for a read transaction. The key signals involved in the Read Address Channel are:

- `ARVALID`: Asserted by the AXI master to indicate that a valid read address is being sent.
- `ARADDR`: The read address sent by the AXI master.
- `ARLEN`: Specifies the number of beats (packets) in the read burst.
- `ARBURST`: Specifies the burst type (INCR, WRAP, FIXED) for the read operation.
- `ARREADY`: Asserted by the bridge one clock cycle after `ARVALID` is asserted, signaling that the read address is accepted.

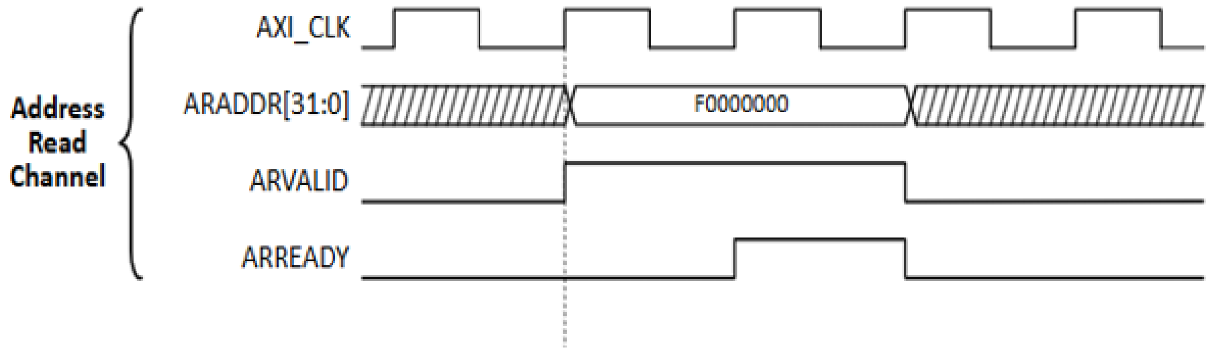


Figure 3.4: Read Address Channel (AR) Handshake

3.7 Read Data Channel (RD)

The Read Data Channel (RD) is used for transferring the read data from the bridge to the AXI master. The key signals involved in the Read Data Channel are:

- `RREADY`: Input signal from the AXI master indicating that it is ready to accept read data.
- `RVALID`: Output signal from the bridge indicating that valid read data is available.
- `RRRESP`: Output signal providing the response status for the read operation (e.g., OKAY, SLVERR).
- `RDATA`: Output signal carrying the read data from the bridge to the AXI master.
- `RLAST`: Output signal asserted by the bridge during the last beat of the read burst, indicating the end of the read transaction.

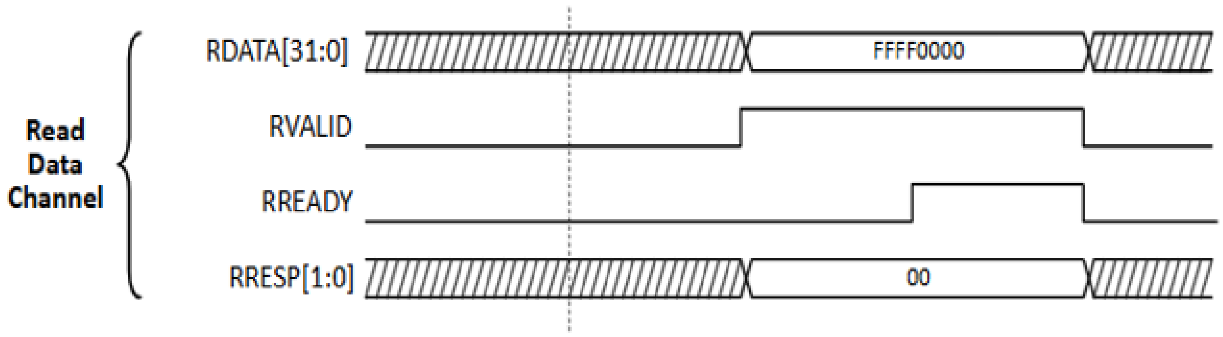


Figure 3.5: Read Data Channel (RD) Handshake

Chapter 4

Design Architecture

The AXI to APB Bridge is designed to mediate data transfers between the AXI master and the APB slave, ensuring compatibility and efficiency. The bridge architecture is based on modular components, featuring separate finite state machines for write and read transactions, as well as a round-robin arbiter for balanced operation. This chapter explains the key design aspects and is supported by illustrative diagrams for clarity.

4.1 Key Components

4.1.1 AXI Slave Interface

The AXI slave interface is responsible for receiving write and read requests from the AXI master and interacting with the FIFOs:

– **Write Requests:** Accepts data, stores it in the Write FIFO, and manages burst addresses.

– **Read Requests:** Retrieves data from the Read FIFO and sends it to the AXI master.

4.1.2 APB Master Interface

The APB master interface handles write and read operations with the APB peripherals:

- **Write Transactions:** Fetches data from the Write FIFO and sends it to the APB slave.
- **Read Transactions:** Retrieves data from the APB slave and places it in the Read FIFO.

4.1.3 Round-Robin Arbiter

The round-robin arbiter alternates priority between the Write FSM and Read FSM, ensuring fair allocation of resources.

4.1.4 FIFOs

- **Write FIFO:** Buffers data from the AXI slave until the APB master can process it.
- **Read FIFO:** Buffers data from the APB slave for delivery to the AXI master.

4.2 Block Diagram

The following block diagram provides a high-level overview of the AXI to APB Bridge, including its main components and their interconnections:

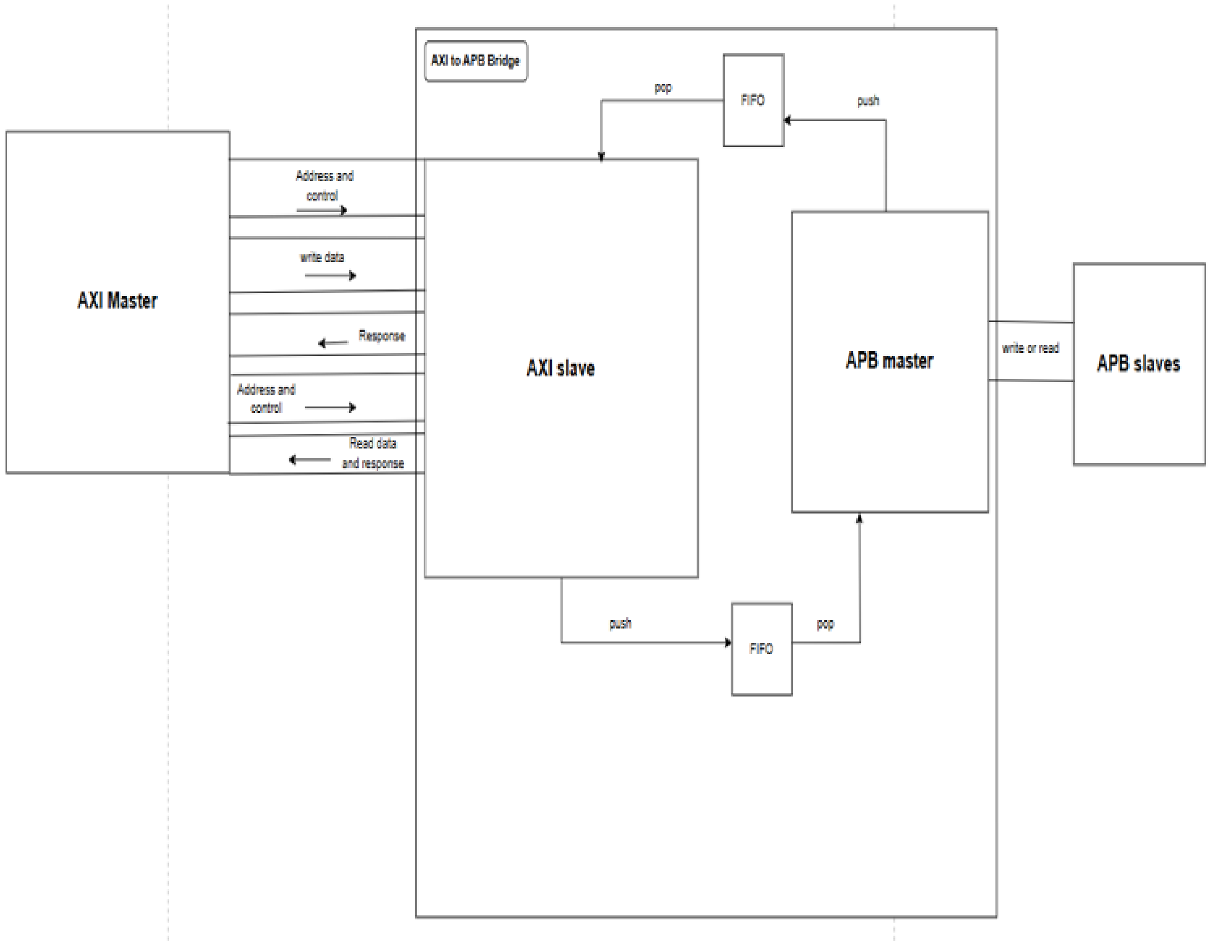


Figure 4.1: Block Diagram of the AXI to APB Bridge

4.3 Finite State Machines (FSMs)

The bridge uses two separate FSMs: one for handling write transactions and the other for read transactions.

4.3.1 Write FSM Overview

The Write FSM manages the state transitions during a write transaction, primarily handling the AXI write flow. Here are the key states and their conditions:

1. **IDLE_WRITE**:
   - This is the initial idle state where the system waits for a valid write request. It transitions to the next state when AWVALID is asserted (1) and the current arbiter state is "WRITE PRIORITY".
2. **WSETUP_M**:

   - In this state, the system prepares for the write transaction by setting up necessary parameters. The state transitions to WPREACCESS_M when AWVALID is still asserted (1).

3. **WPREACCESS_M**:
   - This state is reached when the write address has been accepted, but the data has not yet been processed. The system waits until PREADY is asserted (1), indicating the slave is ready to receive data. Then, it moves to WACCESS_M.
4. **WACCESS_S**:
   - Here, the system accesses the slave for the actual write transaction. It checks whether lenM is greater than 0 and lenS is non-zero, then proceeds to WSETUP_S for data transfer.
5. **WSETUP_S**:
   - The system prepares the data for the slave, asserting DWREQ = 2'b11 as part of the setup. It then transitions to WTERMINATE after the data setup.
6. **WTERMINATE**:
   - This is the final state where the write transaction is complete. The system transitions back to the idle state once BREADY is asserted (1) to signify that the transaction has ended.

4.3.2 FSM Transitions

- **From IDLE to Write SETUP M:** Triggered when `AWVALID = 1` and the current state is not `Write PRIORITY`.
- **From Write SETUP M to Write PREACCESS M:** Occurs when `AWVALID = 1`.
- **From Write ACCESS M to Write SETUP S:** Activated when `lenM != 1  WLAST != 1`.
- **From Write ACCESS M to Write TERMINATE :** Happens when `lenM != 0  WLAST ==1`.
- **From Write SETUP S to Write ACCESS S:** Occurs without any condition.
- **From write Terminate to Write SETUP S :** Occurs when `BREADY = 1`.

4.3.3 Write FSM State Diagram

The following diagram illustrates the state transitions for the Write FSM:

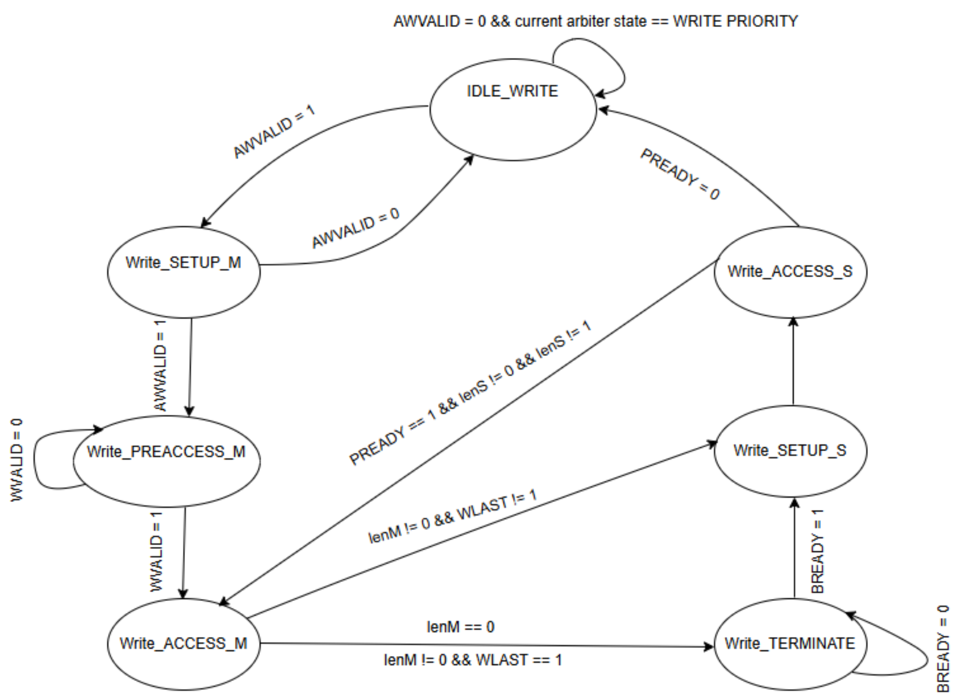


Figure 4.2: Write FSM State Diagram

4.4 Read FSM Overview

The Read FSM manages the state transitions during a read transaction, primarily handling the AXI read flow. Below are the key states and their conditions:

1. **IDLE_READ**:
   - This is the initial idle state where the system is waiting for a valid read request. It transitions to SETUP_M when ARVALID is asserted (1) and the arbiter grants access.
2. **SETUP_M**:
   - In this state, the system prepares for the read transaction. It transitions to ACCESS_S when the setup is complete.
3. **ACCESS_S**:
   - Here, the system accesses the requested data. It checks whether lenM is non-zero and proceeds to PREACCESS_S.
4. **PREACCESS_S**:
   - This state is a preparatory step before the actual data access. It transitions back to IDLE_READ when lenS becomes zero (indicating no more data to read).

4.4.1 FSM Transitions

- **From IDLE to READ SETUP M:** Triggered when `ARVALID = 1` and the current state is not `READ PRIORITY`.
- **From READ SETUP M to READ SETUP S:** Occurs when `ARVALID = 1`.

- **From READ SETUP S to READ ACCESS S:** Activated when `PREADY = 1`.
- **From READ ACCESS S to READ ACCESS M:** Happens when `lenS == 0`.
- **From READ ACCESS S to READ PREACCESS M:** Occurs when `lenS != 0`.
- **From READ PREACCESS M to READ ACCESS M :** Occurs when `PREADY = 1`.

4.4.2 Key Signals

- **ARVALID**: Indicates a valid read address from the AXI interface.
- **RREADY**: Indicates readiness to receive data.
- **lenM**: Represents the length of data to be read.
- **lenS**: Represents the length of data that has been successfully read.

4.4.3 Read FSM State Diagram

The following diagram illustrates the state transitions for the Read FSM:

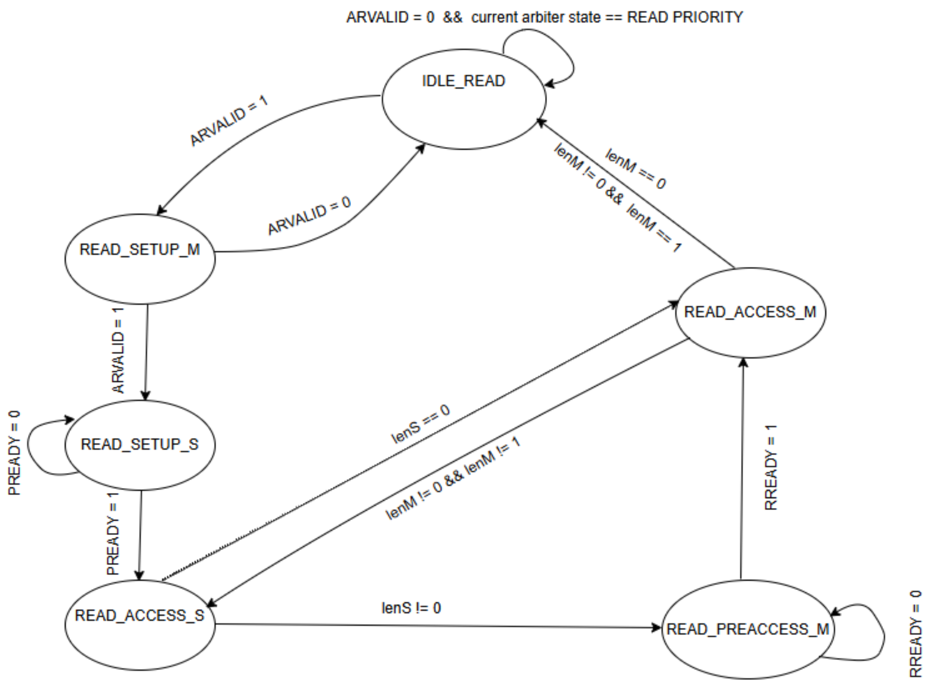


Figure 4.3: Read FSM State Diagram

4.5 Write Transaction Flow

1. **AXI Master to AXI Slave (Write Address Channel):** The AXI master sends the write address and burst information to the AXI slave through the Write Address Channel, including signals like AWADDR, AWBURST, AWLEN, and AWVALID.

2. **AXI Slave (Write Data Channel):** The AXI slave checks the WDVALID signal to determine if the data is valid. It then stores the write data (WDATA) in the Write FIFO, while also monitoring WLAST and WDREADY.

3. **FIFO Buffer:** The Write FIFO temporarily holds the data until the APB master is ready to read it.

4. **APB Master to APB Slave (Write Address and Data):** The APB master retrieves the data from the FIFO and sends it to the APB slave, including PADDR, PWRITE, PENABLE, and PWDATA. The APB slave asserts PREADY when transaction complete.

5. **Write Response Channel (AXI Slave):** The AXI slave receives the BREADY signal and responds with BRES and BVALID to indicate the completion of the write operation.

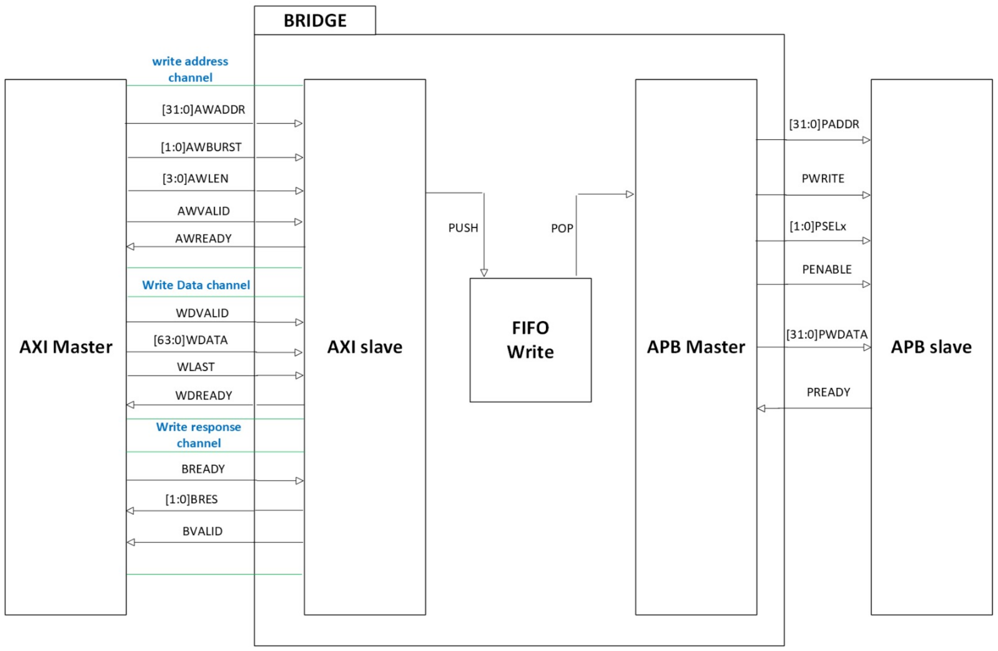


Figure 4.4: Write Transaction Flow

4.6 Read Transaction Flow

1. **AXI Master to AXI Slave (Read Address Channel):** The AXI master sends the read address and burst information to the AXI slave through the Read Address Channel, including signals like ARADDR, ARBURST, ARLEN, and ARVALID.

2. **AXI Slave (Read Data Channel):** The AXI slave checks the RVALID signal to determine if the data is valid. It retrieves the requested data and sends it back to the AXI master via the Read Data Channel, including RDATA, RRESP, and RLAST.

3. **FIFO Buffer:** The FIFO Read temporarily holds the data retrieved by the APB master until the AXI slave is ready to read it.

4. **APB Master to APB Slave (Read Address and Data):** The AXI slave retrieves the data from the APB master via the FIFO buffer and sends them to the AXI master, using the appropriate signals. The APB slave completes the transaction by asserting RREADY for each read operation, indicating that the data have been successfully transferred.

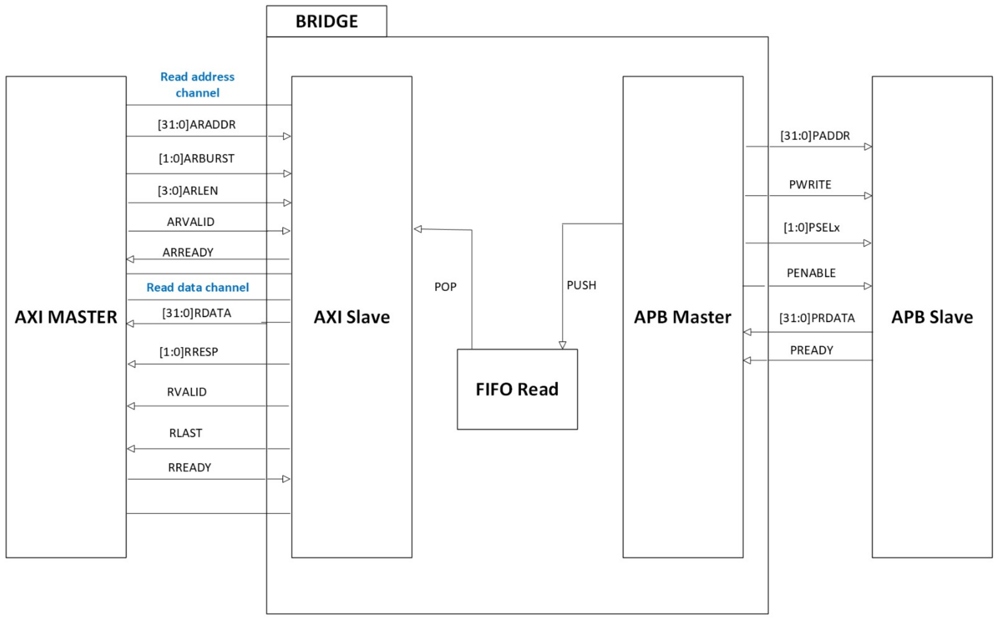


Figure 4.5: Read Transaction Flow

4.7 Round-Robin Arbitration

The round-robin arbiter ensure the equal distribution of access between the Write and Read transactions.

- **Equal Weightage**: Both FSMs have equal priority, alternating control after each completed transaction.
- **Idle Handling**: At reset, the write FSM will get the priority. The arbiter FSM active one FSM and make idle the second FSM.

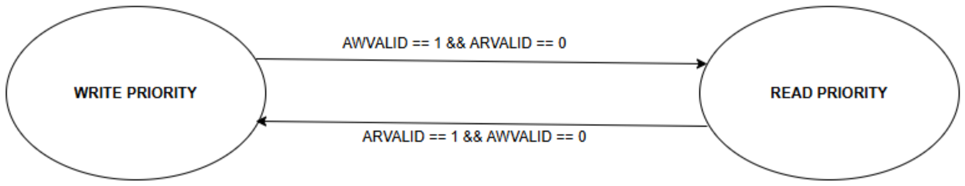


Figure 4.6: Round robin arbiter

In conclusion, this chapter provides a detailed understanding of the AXI write transaction flow and the associated finite- state machine (FSM). We explored the sequential states of the write FSM, highlighting the role of key signals such as AWVALID, WVALID, PREADY, and BREADY to ensure the smooth execution of write operations. The diagrams and FSM transitions presented in this chapter serve as a comprehensive guide to understanding how data flows from the AXI master to the APB slave, emphasizing the importance of synchronization between different components in the system. This foundation will prove valuable in designing, optimizing, and troubleshooting AXI to APB based communication protocols in future hardware designs.

Chapter 5

Simulation

5.1 Simulation Overview

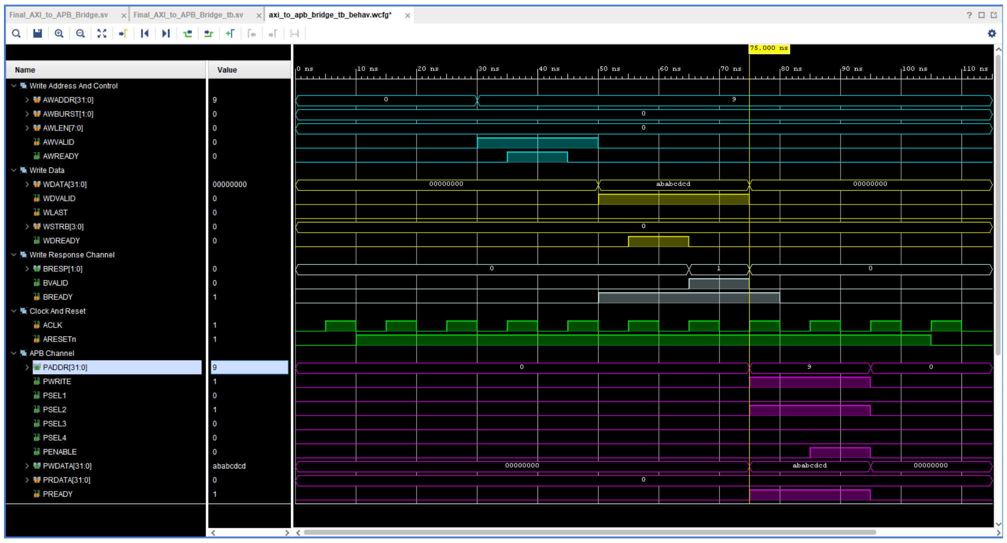
The design of the AXI to APB bridge was simulated using Vivado to verify its functionality and performance. The simulation environment was set up to test various scenarios to ensure the bridge operates as expected under different conditions. The goal of these simulations was to verify the correctness of the bridge's response to both write and read transactions, as well as to ensure that it can handle different AXI burst types (e.g., FIXED, INCREMENTING, WRAP) and simultaneous read and write operations to/from a slave.

5.2 Test Scenarios

The following test scenarios were executed during the simulation:

5.2.1 Single Write Transaction

In a Single Write Transaction, the AXI master asserts the AWVALID signal and provides the write address (AWADDR) along with control parameters. After the AWVALID signal, the AXI slave asserts AWREADY, indicating it is ready to accept the address. Subsequently, the AXI master asserts the WVALID signal with the write data (WDATA) to be sent, and the AXI slave receives these data, storing them in the Write FIFO. The AXI slave then asserts WDREADY, confirming that the data has been writtten to the FIFO. Finally, the APB master retrieves the data from the FIFO and sends it to the APB slave, completing the transaction.


Final_AXI_to_APB_Bridge.sv
Final_AXI_to_APB_Bridge_tb.sv
axi_to_apb_bridge_tb_behav.wcfg*
75.000 ns
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns
110 ns
abcdcdcd

Name	Value
Write Address And Control	
AWADDR[31:0]	9
AWBURST[1:0]	0
AWLEN[7:0]	0
AWVALID	0
AWREADY	0
Write Data	
WDATA[31:0]	00000000
WDVALID	0
WLAST	0
WSTRB[3:0]	0
WDREADY	0
Write Response Channel	
BRESP[1:0]	0
BVALID	0
BREADY	1
Clock And Reset	
ACLK	1
ARESETn	1
APB Channel	
PADDR[31:0]	9
PWRITE	1
PSEL1	0
PSEL2	1
PSEL3	0
PSEL4	0
PENABLE	0
PWDATA[31:0]	ababcdcd
PRDATA[31:0]	0
PREADY	1

Figure 5.1: Single Write Transaction Simulation

5.2.2 Single Read Transaction

In a Single Read Transaction, the AXI master asserts the ARVALID signal along with the read address (ARADDR) and the control parameters. After the ARVALID signal, the AXI slave asserts ARREADY, indicating that it is ready to accept the read address. Once the address is accepted, the AXI master waits for the RVALID signal from the AXI slave, which indicates that the requested data are ready. The AXI slave retrieves the requested data and sends them back to the AXI master via the RDATA signal, while also asserting the RRESP signal to indicate the response status. The AXI master then asserts RREADY to acknowledge receipt of the data, completing the read transaction.

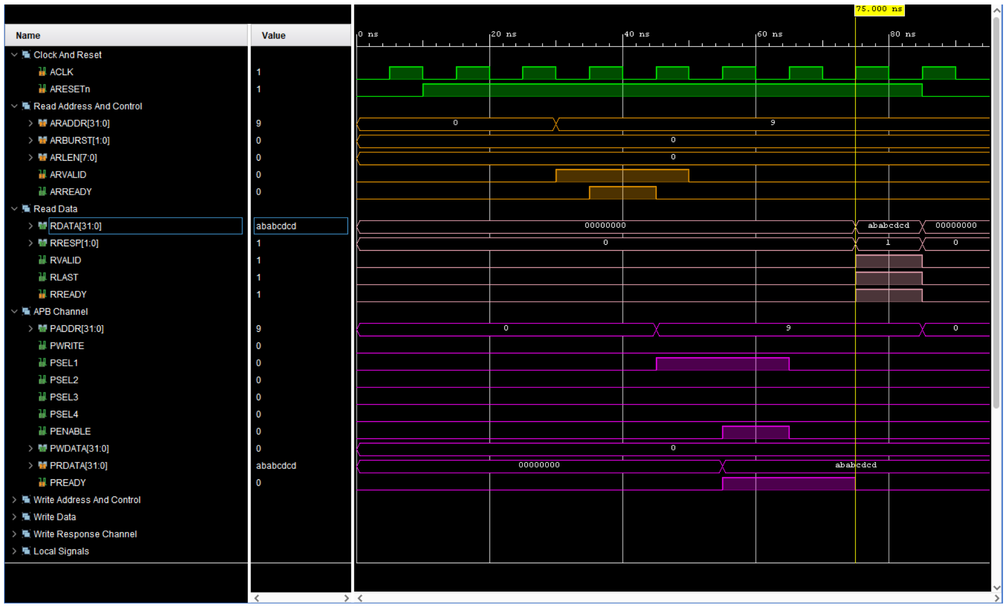


Figure 5.2: Single Read Transaction Simulation

5.2.3 Fixed Burst Write Transaction

In a Fixed Burst Transaction, the AXI master initiates the transaction by asserting AWVALID along with the control parameters burst address (AWADDR), burst type (AWBURST) and burst length (AWLEN). The AWREADY signal is then asserted by the AXI slave after the AWVALID signal to acknowledge the received address. After this, the AXI master asserts the WVALID signal with the data (WDATA) to be written. In the subsequent clock cycle, the WDREADY signal is asserted by the AXI slave to indicate that the data are accepted. This continues for the specified burst length, with each transfer of data corresponding to the same address, and the same address is used for each beat in the burst. The data is passed from the AXI master to the APB slave via the AXI to APB bridge, which retrieves the data from the FIFO and sends these to the APB slave with the appropriate signals (PADDR, PWDATA, PWRITE, PENABLE). The APB master responds with PREADY to complete each transaction. This process continues until all the burst data are transferred.

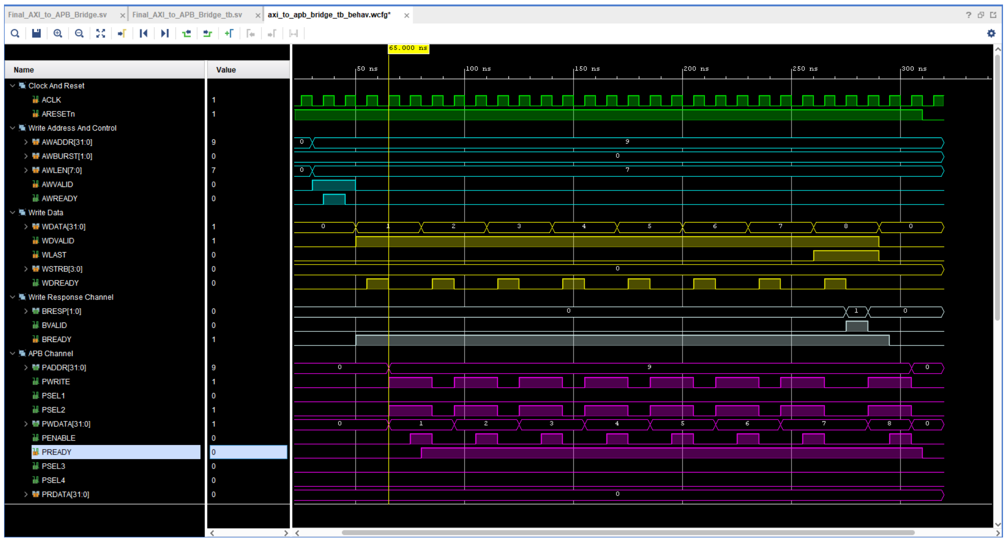


Figure 5.3: Fixed Burst Write Transaction Simulation

5.2.4 Fixed Burst Read Transaction

In a Fixed Burst Read Transaction, the AXI master initiates the read transaction by asserting ARVALID with the read address (ARADDR), burst type (ARBURST), and burst length (ARLEN). The AXI slave responds by asserting ARREADY after the ARVALID signal to acknowledge the received address. Afterward, the AXI slave retrieves the requested data and asserts RVALID with the data (RDATA) on the Read Data Channel. The AXI master, in turn, acknowledges the valid data by asserting RREADY. This process continues for the specified burst length, with the same address used for each data read, ensuring that the data are read sequentially from the AXI slave for the specified burst length. The AXI slave retrieves the data from the APB master via the FIFO buffer and sends them to the AXI master, using the appropriate signals. The AXI slave completes the transaction by asserting RREADY for each read operation, indicating that the data have been successfully transferred.

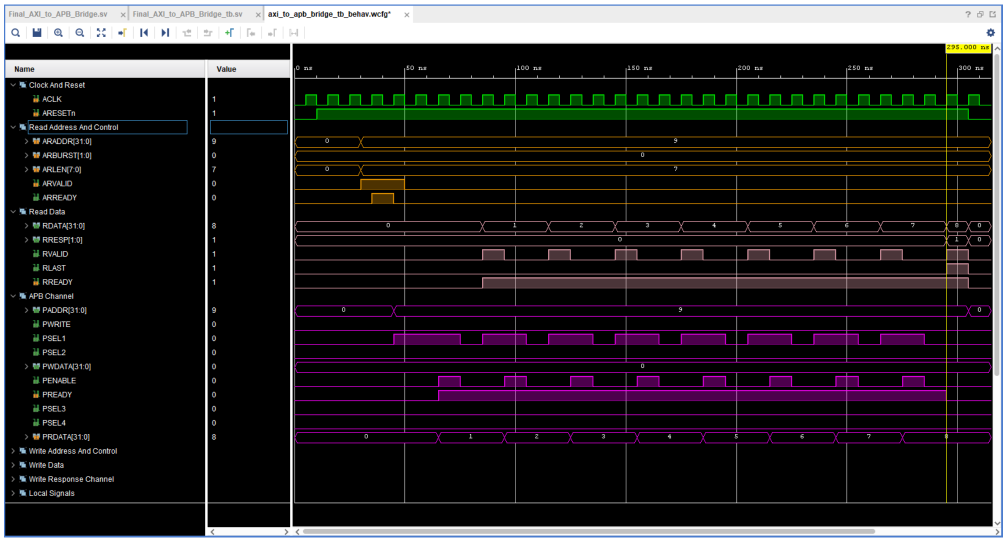


Figure 5.4: Fixed Burst Read Transaction Simulation

5.2.5 Incrementing Burst Write Transaction

In an Incrementing Burst Write Transaction, the AXI master starts by asserting AWVALID with the write address (AWADDR), burst type (AWBURST), burst length (AWLEN), and the address to which the data should be written. The AXI slave acknowledges the address by asserting AWREADY after the AWVALID signal. Then, the AXI master asserts WVALID with the data (WDATA) it wants to write to the AXI slave. The AXI slave responds by asserting WDREADY to indicate it is ready to accept the data. After this, the AXI master sends data to the AXI slave while incrementing the address (AWADDR) for each subsequent data transfer, based on the burst length (AWLEN). This continues for the full length of the burst. The APB master retrieves the data from the AXI slave via the Write FIFO and sends it to the APB slave, using the appropriate signals (PADDR, PWRITE, PENABLE, PWDATA). The APB master signals completion by asserting PREADY for each transfer, completing the write transaction.

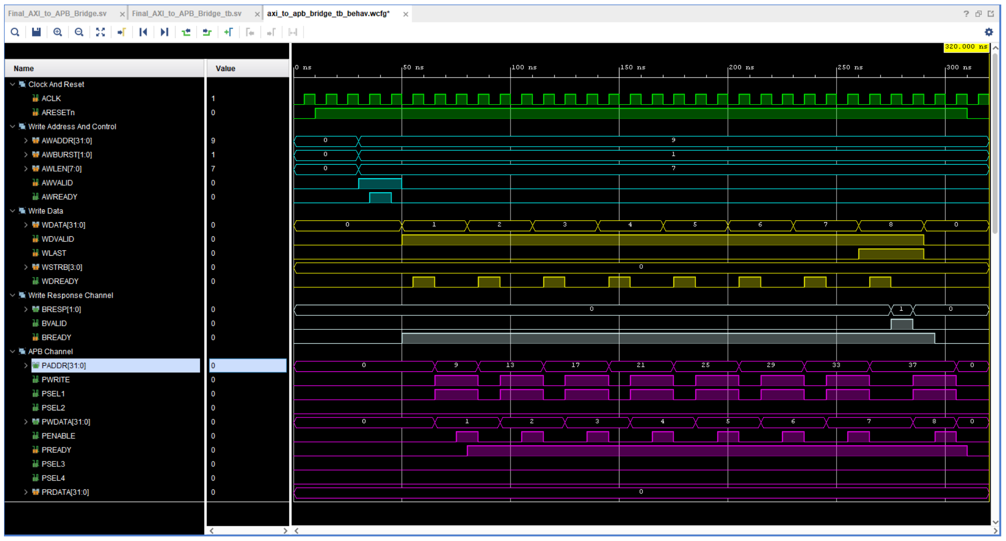


Figure 5.5: Incrementing Burst Write Transaction Simulation

5.2.6 Incrementing Burst Read Transaction

In an Incrementing Burst Read Transaction, the AXI master first asserts ARVALID along with the read address (ARADDR), burst type (ARBURST), and burst length (ARLEN) to initiate the read transaction. The AXI slave responds by asserting ARREADY after one clock cycle to indicate that it is ready to accept the address. After accepting the address, the AXI master waits for RVALID to be asserted by the AXI slave. Once RVALID is asserted, the AXI slave sends the requested data (RDATA) back to the AXI master. For each subsequent data transfer in the burst, the address (ARADDR) is incremented according to the burst length (ARLEN), and the AXI master continues to receive data from the AXI slave. This process repeats for the full length of the burst. The AXI slave retrieves the data from the APB master via the FIFO buffer and sends them to the AXI master, using the appropriate signals. The APB slave completes the transaction by asserting RREADY for each read operation, indicating that the data have been successfully transferred.

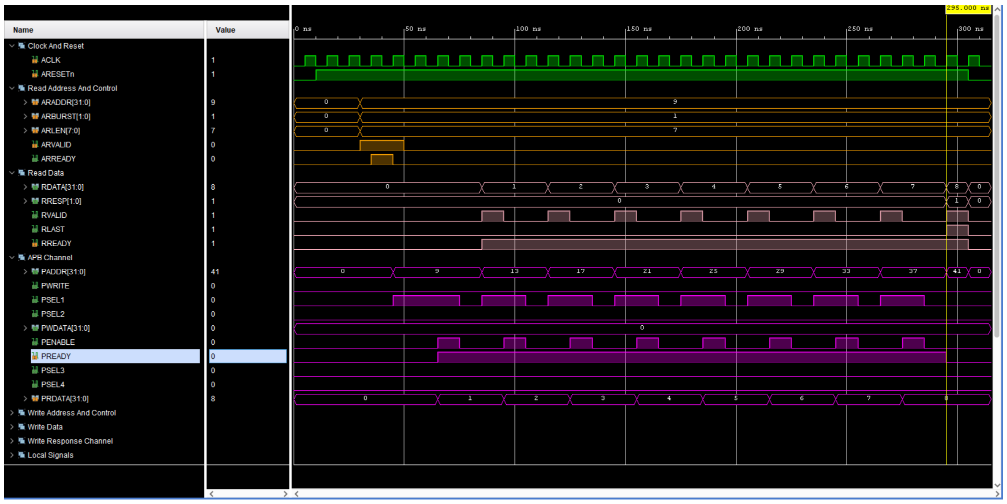


Figure 5.6: Incrementing Burst Read Transaction Simulation

5.2.7 Wrap-Around Burst Write Transaction

In a wraparound burst write transaction, the AXI master first asserts AWVALID along with the write address (AWADDR), burst type (AWBURST), burst length (AWLEN) and the address to which the data is to be written. After the AWVALID signal, the AXI slave asserts AWREADY to indicate that the address has been accepted. In the next clock cycle, the AXI master asserts WVALID along with the write data (WDATA) and the byte enables (WSTRB) to specify which bytes in the data should be written. The AXI slave responds by asserting WDREADY when it is ready to accept the data. The AXI master continues to send write data for the entire burst length.

In the case of a wrap-around burst, once the end of the address range is reached, the address wraps around to the beginning of the burst address, continuing from the base address and writing to the same memory locations. The AWADDR for the next data transfer will be incremented to the base address again after each burst. The APB master retrieves the data from the AXI slave via the Write FIFO and sends it to the APB slave using the appropriate signals (PADDR, PWRITE, PENABLE, PWDATA). The APB slave signals asserted PREADY after each transaction, completing the wrap-around burst write operation.

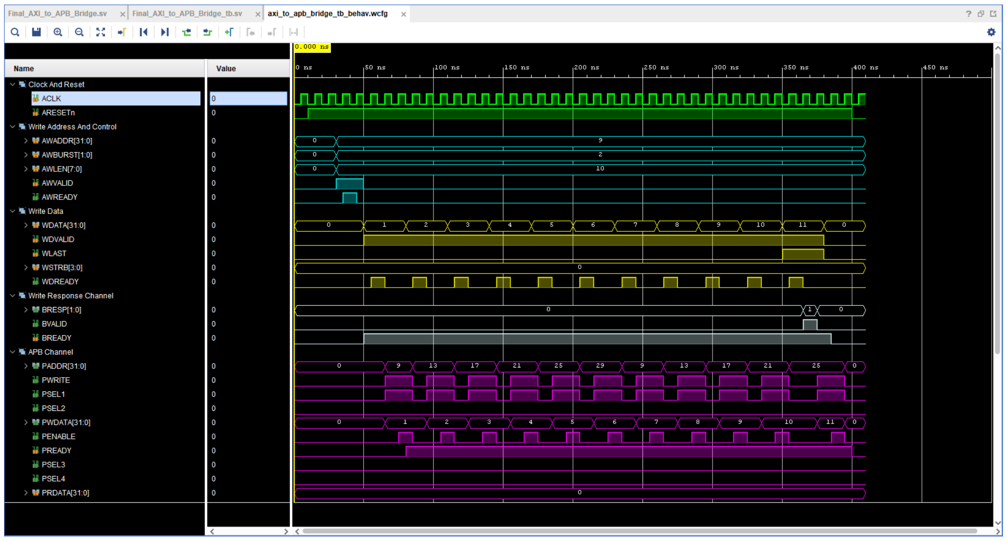


Figure 5.7: Wrap-Around Burst Write Transaction Simulation

5.2.8 Wrap-Around Burst Read Transaction

In a wrap-around burst read transaction, the AXI master first asserts the ARVALID signal, along with the starting address and burst type information (ARADDR, ARBURST, ARLEN). The AXI slave responds by asserting ARREADY after ARVALID signal, indicating that the read address is accepted. The AXI master then asserts the RREADY signal to indicate that it is ready to receive data. The AXI slave retrieves the requested data and sends it to the master through the RDATA signal. For a wrap-around burst, the address for each subsequent read is wrapped around to the start address after reaching the specified burst length, based on the wrap-around burst type. This process continues until the full burst length is complete and the AXI slave asserts RVALID to indicate that the data are valid. The data is then transferred through the Read Data Channel, and the APB master retrieves it from the FIFO before sending it to the APB slave.

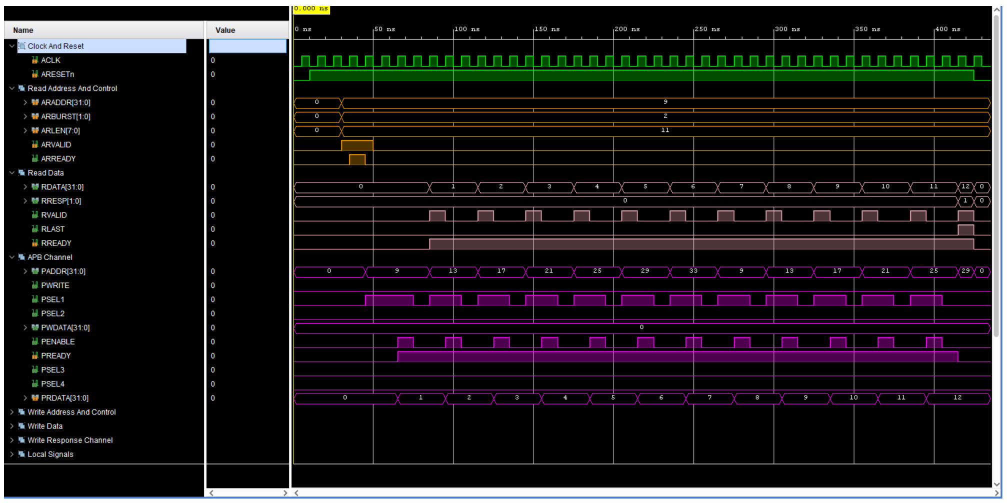


Figure 5.8: Wrap-Around Burst Read Transaction Simulation

5.2.9 Simultaneous Write and Read Transaction from a Slave

In a simultaneous write and read transaction, the round-robin arbiter in the AXI-to-APB bridge ensures fair access by alternating between write and read operations. Initially, the arbiter grants control to the write transaction, allowing the AXI master to send the write address and data. Once the write operation is completed and acknowledged by the APB slave, the arbiter shifts control to the read transaction. The AXI master sends the read address, and the corresponding data is retrieved from the APB slave and sent back to the AXI master. After completing the read operation, the arbiter alternates back to a write transaction. This process continues, ensuring balanced handling of simultaneous write and read requests without giving undue priority to either.

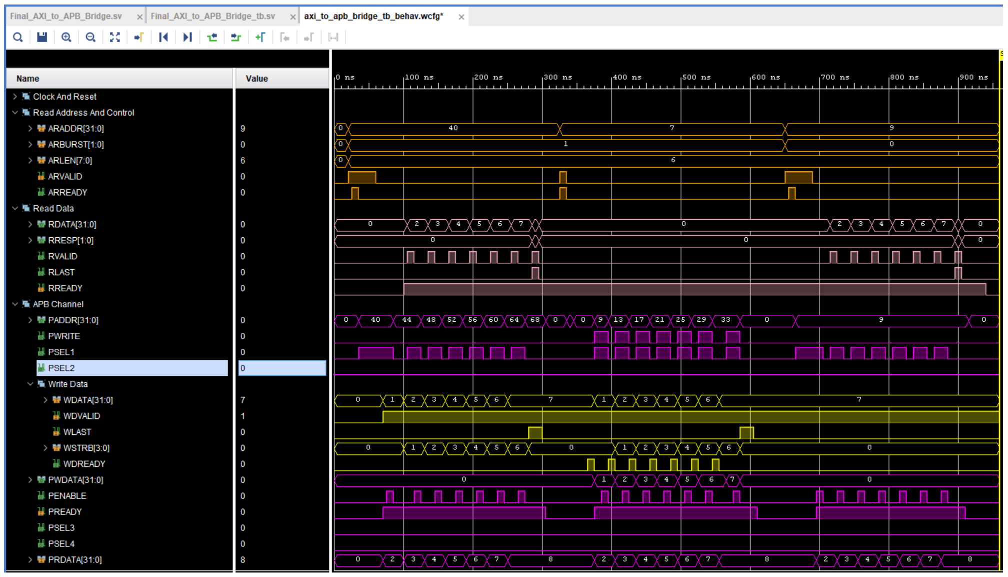


Figure 5.9: Simultaneous Write and Read Transaction Simulation

5.2.10 Independent Write and Read Transaction from Different Slaves

In independent read and write transactions, the AXI master performs write and read operations simultaneously but targets different slaves. The write operation begins by asserting the **AWVALID** signal and specifying the write address, while the read operation asserts the **ARVALID** signal at the same time with a separate address for the read transaction. Since the addresses correspond to different slaves, the write data can be sent to one slave while the read data is retrieved from another slave concurrently. This parallel operation is efficiently handled by the bridge, ensuring no contention between the transactions.

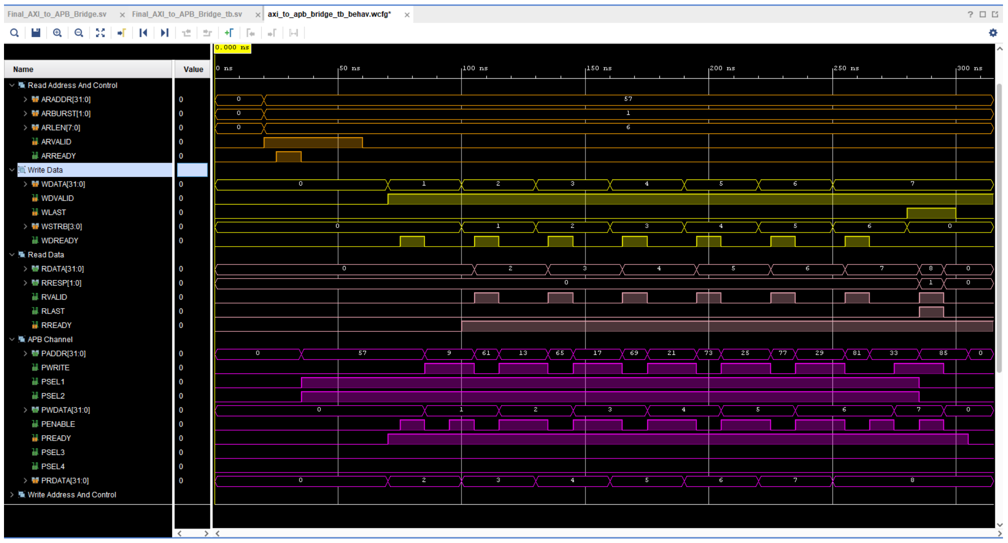


Figure 5.10: Independent Write and Read Transactions from Different Slaves

5.3 Simulation Results

The simulation results were captured in waveforms and output logs. Each test scenario was evaluated for correctness, and the results were compared against the expected outcomes. The bridge correctly handled all the different transaction types and scenarios without errors, indicating that the design is functioning as intended.

5.4 Conclusion

The simulations successfully verified the functionality of the AXI to APB bridge across different transaction scenarios. The design handled single transactions as well as burst transactions, including FIXED, INCREMENTING, and WRAP burst types. Additionally, the bridge was tested for simultaneous read and write operations. These results confirm the bridge's reliability and performance, making it ready for further validation and deployment in real-world applications.

Chapter 6

Reports

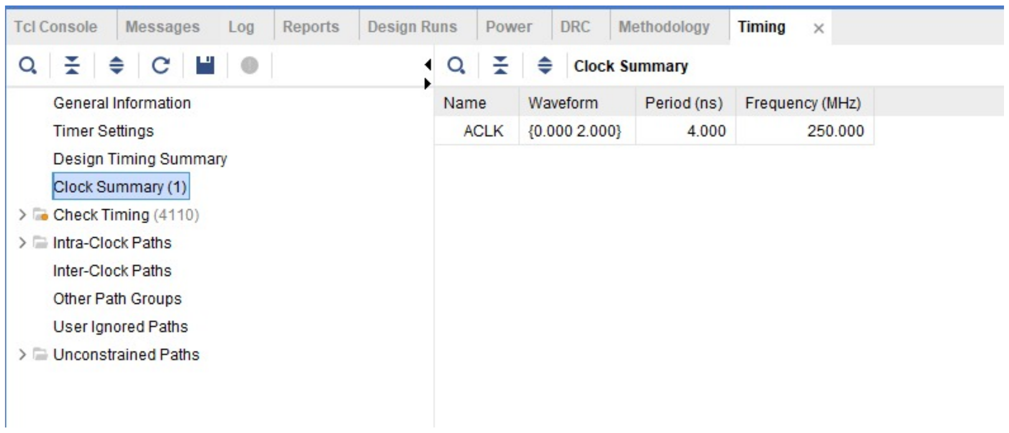
This chapter presents the timing analysis and power consumption results obtained after synthesizing and implementing the AXI to APB bridge design. The reports demonstrate the performance, constraints satisfaction, and resource utilization of the design. The tools used for analysis were Vivado's timing and power reports.

6.1 Timing Analysis

Timing analysis ensures that all timing constraints such as setup time, hold time, and clock-to-output delays are met. The analysis focuses on the critical paths that determine the maximum achievable clock frequency for the design.

6.1.1 Clock Summary

The clock summary provides the details of the clock period and frequency used in the design. The waveform and clock characteristics are shown below.



Name	Waveform	Period (ns)	Frequency (MHz)
ACLK	{0.000 2.000}	4.000	250.000

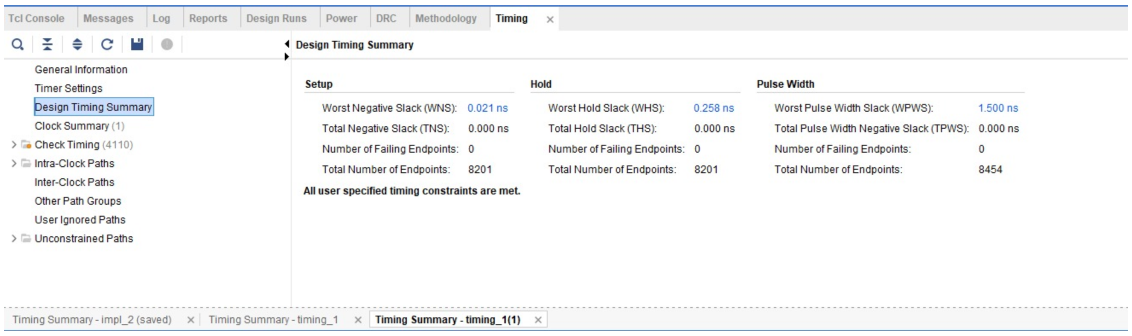
Figure 6.1: Clock Summary Report

In this design:

- Clock Name: ACLK
- Clock Period: 4 ns
- Clock Frequency: 250 MHz

6.1.2 Design Timing Summary

The Design Timing Summary provides an overview of the timing constraints, including setup, hold, and pulse width slacks.



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.021 ns	Worst Hold Slack (WHS):	0.258 ns	Worst Pulse Width Slack (WPWS):	1.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	8201	Total Number of Endpoints:	8201	Total Number of Endpoints:	8454

All user specified timing constraints are met.

Figure 6.2: Design Timing Summary Report

Key timing metrics include:

- Setup Time:
  - Worst Negative Slack (WNS): 0.021 ns
  - Total Negative Slack (TNS): 0.000 ns
  - Number of Failing Endpoints: 0
- Hold Time:
  - Worst Hold Slack (WHS): 0.258 ns
  - Total Hold Slack (THS): 0.000 ns
  - Number of Failing Endpoints: 0
- Pulse Width:
  - Worst Pulse Width Slack (WPWS): 1.500 ns
  - Total Pulse Width Negative Slack (TPWS): 0.000 ns
  - Number of Failing Endpoints: 0

The results demonstrate that:

- All user-specified timing constraints are met.
- There are no failing endpoints for setup, hold, or pulse width timing checks.

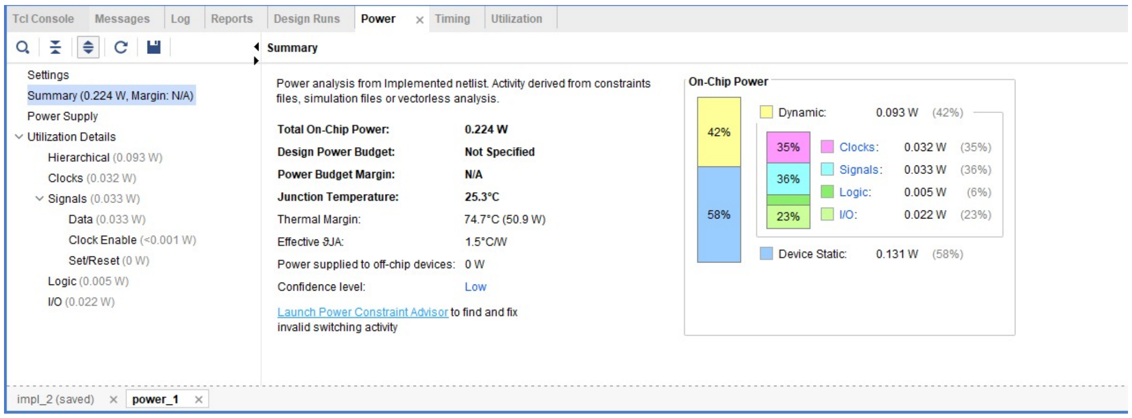
6.2 Power Analysis

Power analysis provides insights into the dynamic and static power consumption of the design, ensuring the power constraints are met and identifying opportunities for optimization.

6.2.1 Power Summary

The power report summarizes the total on-chip power, dynamic power, and static power, along with their distribution across various components.
Key power metrics from the analysis include:

- Total On-Chip Power: 0.224 W



Summary

Settings
Summary (0.224 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (0.093 W)
Clocks (0.032 W)
Signals (0.033 W)
Data (0.033 W)
Clock Enable (<0.001 W)
Set/Reset (0 W)
Logic (0.005 W)
I/O (0.022 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.224 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.3°C**
Thermal Margin:	74.7°C (50.9 W)
Effective ϑJA:	1.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	0.093 W	(42%)
Clocks:	0.032 W	(35%)
Signals:	0.033 W	(36%)
Logic:	0.005 W	(6%)
I/O:	0.022 W	(23%)
Device Static:	0.131 W	(58%)

Figure 6.3: Power Report Summary

- Dynamic Power: 0.093 W (42%)
- Static Power (Device): 0.131 W (58%)
- Junction Temperature: 25.3°C
- Thermal Margin: 74.7°C

6.2.2 Power Breakdown

The power consumption is further broken down into the following components:

- Clocks: 0.032 W (35%)
- Signals: 0.033 W (36%)
- Logic: 0.005 W (6%)
- I/O: 0.022 W (23%)

The report highlights that:

- Dynamic power accounts for 42% of the total power.
- Static power (device leakage) contributes 58% of the overall power consumption.
- Signals and clocks consume the majority of the dynamic power.

Optimizations can focus on:

- Reducing unnecessary switching activity to minimize dynamic power.
- Optimizing clock gating to reduce clock power.
- Implementing low-leakage techniques to manage static power.

6.3 Utilization Overview

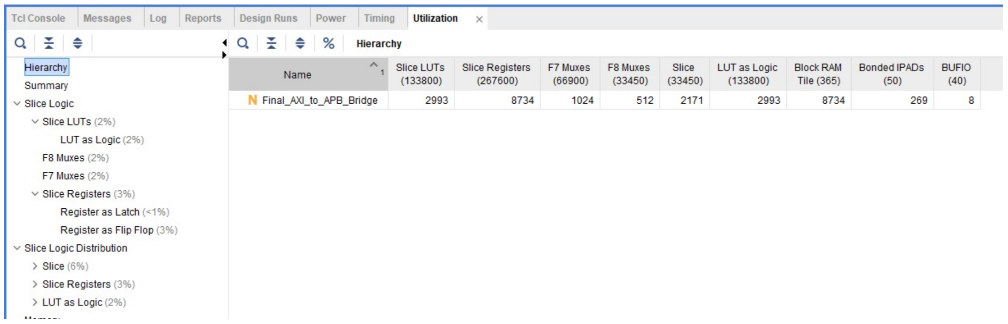
The following table summarizes the utilization metrics for the design:

Metric	**Value**
Slice LUTs	2,993
FF Muxes	8,734
F7 Muxes	2,154
Slice LUTs as Logic	2,171
Block RAM Tile (BRAM)	0
Bonded PLDs	48
BUFDs	0

Table 6.1: Utilization Metrics

6.4 Screenshot of Utilization Report

Below is a screenshot of the utilization report:



- Hierarchy
- Summary
- Slice Logic
  - Slice LUTs (2%)
    - LUT as Logic (2%)
  - F8 Muxes (2%)
  - F7 Muxes (2%)
  - Slice Registers (3%)
    - Register as Latch (<1%)
    - Register as Flip Flop (3%)
- Slice Logic Distribution
  - Slice (6%)
  - Slice Registers (3%)
  - LUT as Logic (2%)

Name	Slice LUTs (133800)	Slice Registers (267600)	F7 Muxes (66900)	F8 Muxes (33450)	Slice (33450)	LUT as Logic (133800)	Block RAM Tile (365)	Bonded IPADs (50)	BUFIO (40)
Final_AXI_to_APB_Bridge	2993	8734	1024	512	2171	2993	8734	269	8

Figure 6.4: Utilization Report

6.5 Key Observations

- The design utilizes a total of 2,993 Slice LUTs.
- There are 8,734 FF Muxes and 2,154 F7 Muxes, indicating a significant use of multiplexing.
- Slice LUTs used as logic are relatively close to the total Slice LUTs.
- The design has no Block RAM Tile usage.
- There are 48 Bonded PLDs, with no BUFDs utilized.

6.6 Summary

In this chapter, the timing and power analyses of the design are discussed in detail. The **Timing Analysis** ensures that the design meets the specified constraints, and the results show that all timing constraints have been satisfied. The worst negative slack (WNS) is 0.021 ns, indicating that there are no significant timing violations. The design's setup and hold checks also meet the required criteria, with no failing endpoints. Pulse width constraints are satisfied with a worst pulse width slack (WPWS) of 1.500 ns, confirming robust timing performance.

On the other hand, the **Power Analysis** shows that the total on-chip power consumption is 0.224 W, with dynamic power (42%) and static power (58%) contributions. The dynamic power consumption is dominated by signals and clocks, while the static power, mainly due to device leakage, accounts for the remaining portion. The analysis provides insights into the distribution of power across various components such as logic, I/O, and clocks, and highlights areas for potential power optimization. Reducing switching activity and applying low-power design techniques are suggested to improve power efficiency without compromising timing performance.

Together, these analyses confirm that the design meets both timing and power requirements, while also identifying areas for further optimization to improve performance and reduce energy consumption.

Chapter 7

Conclusion

The AXI to APB Bridge was successfully designed, implemented, and tested to ensure seamless communication between AXI and APB protocols. The key aspects of this project include efficient handling of single and burst transactions, both for read and write operations, as well as simultaneous read-write operations through a round-robin arbitration mechanism.

7.1 Key Achievements

- **Design Implementation:** The bridge supports various AXI burst modes, including single, fixed burst, incrementing burst, and wrap-around burst transactions.
- **Round-Robin Arbitration:** A fair arbitration mechanism was implemented to balance simultaneous write and read operations, ensuring equal priority.
- **Simulation and Validation:** The design was simulated in Vivado, and multiple scenarios were tested successfully. These include:
  - Single Write Transaction
  - Single Read Transaction
  - Fixed Burst Write and Read
  - Incrementing Burst Write and Read
  - Wrap-Around Burst Write and Read
  - Simultaneous Write and Read Transactions

7.2 Summary of Results

Each test scenario was analyzed in detail, and simulation results demonstrated that the bridge operates correctly under all specified conditions. Write and read transactions were executed efficiently, and data integrity was maintained throughout the process.

7.3 Future Enhancements

The following improvements can be considered for future work:

- **Integration with Advanced Verification Environments:** Integrate advanced verification methods for comprehensive testing.
- **Interrupt Support for APB Peripherals:** Add interrupt signaling from APB slaves back to the AXI master.

7.4 Conclusion

The AXI to APB Bridge successfully facilitates communication between AXI masters and APB slaves, ensuring compatibility between two widely used protocols. The design adheres to AXI and APB specifications and provides a robust, reliable, and efficient solution for bridging these interfaces. The successful simulation and validation of multiple test cases confirm the correctness and effectiveness of the design.

References

- **AXI Protocol Documentation:** Available at: https://www.arm.com/why-arm/technologies/axi.
- **APB Protocol Documentation:** Available at: https://www.arm.com/why-arm/technologies/apb.
- **Github Repo:** Available at: https://github.com/AXI-To-APB-Bridge.