Lab 10: AXI4-Lite Interface Design

1. Objective

The objective of this lab is to design, implement, and test an AXI4-Lite Slave Interface that provides access to a 16 x 32-bit register bank. The design must support read and write transactions with proper address decoding, handshake signaling, and error response generation. The slave will later be integrated with modules from previous labs such as the UART controller, timer/counter, and FIFO.

2. AXI4-Lite Protocol Overview

AXI4-Lite is a simplified subset of the AXI4 protocol, mainly used for register-mapped peripherals.

Key Characteristics:

- 32-bit address and data buses
- Separate read and write address channels
- Write response channel for acknowledging write transactions
- No burst support (single transfers only)
- Simple VALID/READY handshake protocol

Handshake Protocol:

Write Transaction:

- Master provides AWADDR with AWVALID
- Slave responds with AWREADY
- Master provides WDATA with WVALID
- Slave responds with WREADY
- Slave issues BRESP and BVALID
- Master acknowledges with BREADY

Read Transaction:

- Master provides ARADDR with ARVALID
- Slave responds with ARREADY
- Slave sends RDATA with RVALID
- Master acknowledges with RREADY

3. Design Specification

- Register Bank: 16 registers × 32-bit wide
- Access: Read/Write support on all registers
- Address Decode: Word-aligned using bits [5:2] (16 valid locations)

- Error Handling: SLVERR response for invalid addresses
- State Machines:
  - Write channel FSM (W_IDLE, W_ADDR, W_DATA, W_RESP)
  - Read channel FSM (R_IDLE, R_ADDR, R_DATA)

4. Design Methodology

Step 1: AXI4-Lite Interface
A SystemVerilog interface was created (axi4_lite_if) to encapsulate all AXI4-Lite signals and define modports for master and slave.

Step 2: AXI4-Lite Slave
- Implemented a 16×32-bit register bank (register_bank[0:15]).
- Address decode logic maps awaddr[5:2] and araddr[5:2] to register indices.
- Two FSMs were designed:
  - Write FSM: Handles address acceptance, write data, and response.
  - Read FSM: Handles read address, data return, and response.

Step 3: Response Logic
- Valid address → BRESP = 2'b00 (OKAY), RRESP = 2'b00 (OKAY)
- Invalid address → BRESP = 2'b10 (SLVERR), RRESP = 2'b10 (SLVERR)

5. Implementation

Write Channel FSM:
- W_IDLE: Wait for AWVALID
- W_ADDR: Capture address, check validity
- W_DATA: Accept data, apply WSTRB for selective byte writes
- W_RESP: Send BRESP

Read Channel FSM:
- R_IDLE: Wait for ARVALID
- R_ADDR: Capture address, prepare data
- R_DATA: Send RDATA and RRESP

6. Testbench

A SystemVerilog testbench was written to verify the slave design.

Stimulus:
1. Write 0xABCDEFCC into register[0]
2. Write 0x00560078 into register[1] (partial write using WSTRB)
3. Read back register[0]

4. Read back register[1]

Expected Results:
- Register[0] returns 0xABCDEFCC
- Register[1] returns 0x00560078
- Invalid addresses return 0xDEADBEEF with SLVERR

7. Simulation Results

- Waveform confirmed correct VALID/READY handshakes.
- Writes updated the register bank correctly.
- Reads returned the correct values with expected responses.
- Error handling worked for invalid addresses.

8. Integration with Previous Labs

This AXI4-Lite slave interface can be extended to connect with:
- UART Controller (AXI registers for TX/RX buffers, status flags)
- Timer/Counter Module (registers for count values, control, interrupt flags)
- FIFO Buffers (status and control registers exposed over AXI4-Lite)
- System Control (configuration and global status registers)

This provides a unified memory-mapped interface for CPU/SoC integration.

9. Conclusion

Successfully implemented an AXI4-Lite slave with a register bank.
- Designed FSMs for handling read and write transactions.
- Verified design using a testbench with multiple read/write operations.
- Extended design concept for integration with UART, timer, and FIFO modules.

This lab provided hands-on experience with the AXI4-Lite protocol, state machine design, and register-mapped peripheral implementation, which are fundamental concepts in SoC and FPGA-based system design.