AXI4-Lite - Methodology

**Industry-Standard Practices for Your AXI4-Lite Slave Project**

**1. Reset & Clock Conventions**

* **Reset**: Active-low, synchronous release but asynchronous assertion (ARESETn).  
  → This ensures your design comes up clean in silicon.
* **Clock**: Always a single ACLK. No combinational logic in the clock path.  
  → Use posedge ACLK everywhere (reset exception aside).

**2. Modularity & Hierarchy**

Instead of one giant file, **separate into logical submodules**:

* **Top AXI4-Lite Slave Wrapper** (ports = AXI interface only).
* **Write Address Channel Handler**.
* **Write Data Channel Handler**.
* **Write Response Channel Handler**.
* **Read Address Channel Handler**.
* **Read Data Channel Handler**.
* **Register File / FIFO / Memory backend**.

👉 Why:  
This mirrors *industry IPs* (Xilinx/ARM deliver AXI IPs like this). It lets verification teams swap/register different backends (register block, RAM, FIFO, etc.) without touching the protocol handler.

**3. State Machine Style**

* Use **enumerated FSMs** (typedef enum logic [x:0] {IDLE, WAIT, RESP}) instead of nested if-else.
* **One FSM per channel** (instead of one giant FSM for all).  
  👉 Why: Easier to debug, waveform is readable, code is scalable.

**4. Registering Signals (No Comb Feedback)**

* AXI requires that VALID/READY are **driven from registers**, not combinational loops.

**5. Handshake Discipline**

* Always follow VALID && READY → transfer rule.
* Never deassert VALID until handshake is complete.
* Always register incoming data on handshake.  
  👉 This makes your block fully **AXI-compliant**.

**6. Scalability Hooks**

Even if you’re only targeting a simple register map now:

* Parameterize **address width, data width**.
* Keep backend (FIFO/register block) **pluggable**.
* Document your address map (e.g., reg0 = control, reg1 = status).

👉 Why: Later you can swap your “8-bit register” with your **FIFO project** without rewriting the whole slave.

**7. Coding Style & Readability**

* Use **prefixes**: S\_AXI\_AWVALID, S\_AXI\_AWREADY, … (industry standard).
* One process per channel (always @(posedge clk)).
* Don’t mix protocol + backend logic.
* Comment the *AXI spec clause* near tricky logic.

**8. Verification Practices**

* Write a **self-checking testbench** with a simple AXI master BFM (bus functional model).
* Drive different transaction orders (W first, AW first).
* Randomize spacing between AW/W/R to ensure your FSM works in all cases.

👉 In industry, your block would *fail review* if not self-verified.

**9. Synthesis Awareness**

* No latches.
* No uninitialized regs.
* Reset all flops explicitly.
* Make sure your FIFO integrates cleanly (no combinational feedback across clock domains).

**10. Documentation**

* Write a **README or spec doc**:
  + Ports list + description.
  + Address map.
  + Example transaction.  
    👉 Hiring managers love seeing **spec + RTL + testbench** in GitHub — it screams *“industry-ready”*.

**📑 AXI4-Lite Slave Design Guide (Architecture Document)**

**1. Overview**

AXI4-Lite is a lightweight subset of the AXI4 protocol, supporting only **single-beat** read/write transactions. It is used for **control/status registers (CSRs)** rather than high-bandwidth data transfer.  
Our goal is to design a **configurable AXI4-Lite slave** with a small register file, which later connects to a FIFO, encryption block, or any peripheral.

**2. Key Characteristics of AXI4-Lite**

* **Single data beat only** (no bursts).
* **Five channels**, each with its own handshake:
  1. Write Address (AW)
  2. Write Data (W)
  3. Write Response (B)
  4. Read Address (AR)
  5. Read Data (R)
* **Decoupled channels** → Address & data phases can arrive independently.
* **Handshake mechanism** → VALID and READY must both be high for a transfer.

**3. AXI4-Lite Slave High-Level Block Diagram**

+----------------------------+

| AXI Slave |

| |

AWADDR -->| |--> BRESP

AWVALID ->| |--> BVALID

AWREADY <-|Write Address Channel |<- BREADY

| |

WDATA -->| |

WSTRB -->| |

WVALID -> | Write Data Channel |

WREADY <-| |

| |

ARADDR -->| |--> RDATA

ARVALID ->| Read Address Channel |--> RRESP

ARREADY <-| |--> RVALID

| |<- RREADY

| |

| Register File |

+----------------------------+

**4. Functional Units to Implement**

1. **Register File**
   * Acts as the backend memory.
   * Typically implemented as a small array of 32-bit registers (size configurable).
   * Example: reg [31:0] regfile[0:15];
2. **Write Path**
   * Capture **AWADDR** when AWVALID && AWREADY.
   * Capture **WDATA** when WVALID && WREADY.
   * Update the register file at the decoded address.
   * Send **BRESP = OKAY** once write is committed.
3. **Read Path**
   * Capture **ARADDR** when ARVALID && ARREADY.
   * Fetch data from register file.
   * Drive **RDATA** and **RRESP = OKAY**.
4. **Handshake FSMs (per channel)**
   * Each channel needs small FSM or simple combinational handshake logic.
   * Example:
     + AW: Idle → Capture → Done.
     + W: Idle → Capture → Done.
     + B: Wait for both AW & W complete → Send Response.
     + AR: Idle → Capture → Done.
     + R: Data valid until RREADY handshake.

**5. Control Signals**

* **AWADDR / ARADDR**: Address to access. Must be aligned to word boundaries.
* **WDATA / WSTRB**: Data + byte strobes for partial writes.
* **BRESP / RRESP**: Always OKAY (2’b00) for AXI4-Lite unless error injection is added.
* **VALID / READY**: Classic AXI handshake.

**6. Registers and Mapping**

You should define the register map **early**. Example:

| **Address Offset** | **Register Name** | **Access** | **Description** |
| --- | --- | --- | --- |
| 0x00 | CONTROL | R/W | Control bits for peripheral |
| 0x04 | STATUS | R | Status flags |
| 0x08 | DATA\_IN | W | Data to FIFO / peripheral |
| 0x0C | DATA\_OUT | R | Data from FIFO / peripheral |

**7. Design Considerations**

* **Scalability**: Support multiple registers (not hardwired to 1).
* **Decoupling**: Handle AW and W channels arriving in different cycles.
* **Reset Behavior**: All registers should reset to 0 or known state.
* **Compliance**: Follow AXI handshake rules strictly (VALID/READY handshake).
* **Performance**: AXI4-Lite is not for throughput — keep FSM simple and reliable.

**8. Verification Strategy**

* **Testbench Stimuli**:
  + Single read/write.
  + Back-to-back writes.
  + Back-to-back reads.
  + AW and W arriving in different cycles.
  + Randomized valid/ready delays (to stress-test handshake).
* **Checkpoints**:
  + Register file updates correctly.
  + BRESP and RRESP are always OKAY.
  + No deadlock when channels stall.

**9. Future Extensions**

* Connect to FIFO (instead of raw registers).
* Add interrupt support (IRQ line when STATUS changes).
* Add error injection (SLVERR response).
* Connect multiple slaves using an AXI interconnect.

**📝 Commit Message Conventions (Git best-practice)**

Industry teams often use the **Conventional Commits** style:

* feat: → New feature (e.g., feat: added AW channel FSM)
* fix: → Bug fix (e.g., fix: corrected reset polarity in W channel)
* refactor: → Logic restructuring, no behavior change
* test: → Adding/changing testbench or verification files
* docs: → Documentation changes
* style: → Formatting, renaming signals, cleanup

This will make your repo **readable like a story** of how the AXI block grew.