AXI Lite Memory

All Signal of Axi Bus :

AXI-Lite has 5 channels:

- AW = write address
- W = write data
- B = write response
- AR = read address
- R = read data

Table B1-1 AXI4-Lite interface signals

Global	Write address channel	Write data channel	Write response channel	Read address channel	Read data channel
ACLK	AWVALID	WVALID	BVALID	ARVALID	RVALID
ARESETn	AWREADY	WREADY	BREADY	ARREADY	RREADY
–	AWADDR	WDATA	BRESP	ARADDR	RDATA
–	AWPROT	WSTRB	–	ARPROT	RRESP

Top module All Signal Details:

```
module axilite_slave #(
parameter ADDR_WIDTH = 32,
parameter DATA_WIDTH = 32,
parameter DEPTH = 128        // number of 32-bit words (addressed by wor
d)
)(
//global signal
input  wire                  s_axi_aclk,
input  wire                  s_axi_aresetn,// Write address channel
```

```
input  wire                  s_axi_awvalid,
output logic                 s_axi_awready,
input  wire [ADDR_WIDTH-1:0]   s_axi_awaddr,

// Write data channel
input  wire                  s_axi_wvalid,
output logic                 s_axi_wready,
input  wire [DATA_WIDTH-1:0]   s_axi_wdata,
input  wire [DATA_WIDTH/8-1:0] s_axi_wstrb,

// Write response channel
output logic                 s_axi_bvalid,
input  wire                  s_axi_bready,
output logic [1:0]           s_axi_bresp,

// Read address channel
input  wire                  s_axi_arvalid,
output logic                 s_axi_arready,
input  wire [ADDR_WIDTH-1:0]   s_axi_araddr,

// Read data channel
output logic                 s_axi_rvalid,
input  wire                  s_axi_rready,
output logic [DATA_WIDTH-1:0]  s_axi_rdata,
output logic [1:0]           s_axi_rresp
```

);

Write Strobe : s_axi_wstrb

AXI Lite Support Multisize Reg 8 or 16 bit

Eg: If The width is 32 bit then its width will be 32/8 = 4 bit

32-bit DATA bus (DATA_WIDTH=32)

WSTRB is:

```
WSTRB[3]  WSTRB[2]  WSTRB[1]  WSTRB[0]
  ↑         ↑         ↑         ↑
```

```
byte3   byte2   byte1   byte0   (LSB)
```

- If `WSTRB[0] = 1` → Write lower 8 bits of WDATA (`[7:0]`)
- If `WSTRB[0] = 0` → Do NOT write `[7:0]`

Example (32-bit)

1. Let's say:

```
WSTRB = 4'b0001
WDATA = 32'hAA_BB_CC_DD
```

Meaning:

- byte0 (lower 8 bits) = `DD`
- Only this byte will be written.

2. If `WSTRB = 4'b1111` on a 32-bit AXI-Lite bus:

All **4 bytes** (all **32 bits**) of the word will be written.

Because:

```
WSTRB[3] = 1 → write WDATA[31:24]
WSTRB[2] = 1 → write WDATA[23:16]
WSTRB[1] = 1 → write WDATA[15:8]
WSTRB[0] = 1 → write WDATA[7:0]
```

So essentially:

```
mem[word_address] <= WDATA;
```

Fsm State:

```
typedef enum logic [2:0] {
    ST_IDLE,
    ST_WRITE_ADDR,
    ST_WRITE_DATA,
    ST_WRITE_RESP,
```

```
    ST_READ_ADDR,
    ST_READ_DATA
} state_t;
```

AXI-Lite has

1. ST_IDLE — Waiting for a Transaction

Purpose: The slave is idle and ready to receive either:

- Write address (AW)
- Write data (W)
- Read address (AR)

Behavior:

- `s_axi_awready = 1` (if no AW captured)
- `s_axi_wready = 1` (if no W captured)
- `s_axi_arready = 1` (if no AR captured)

Transition:

- If AW arrives → go to **ST_WRITE_ADDR**
- Else if AR arrives → go to **ST_READ_ADDR**

2. ST_WRITE_ADDR — Address Phase of Write

Purpose: In this Stage The write Address Happened

Behavior:

- `s_axi_awready = 1` until AW handshake happens
- Also allow W to arrive if master sends W early:
    - `s_axi_wready = ~w_captured`

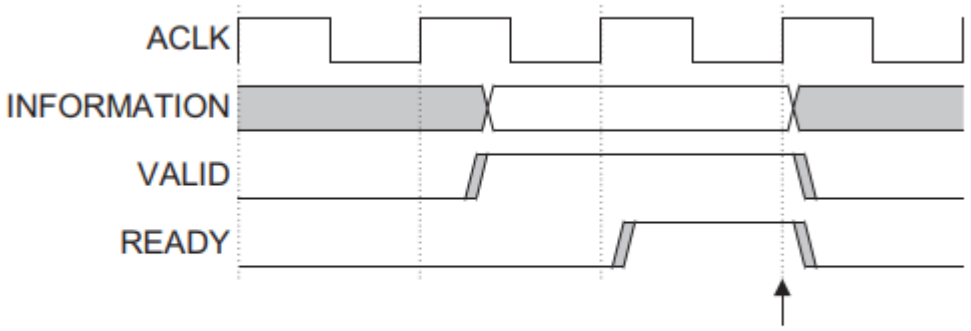

Figure 3-1 VALID before READY handshake

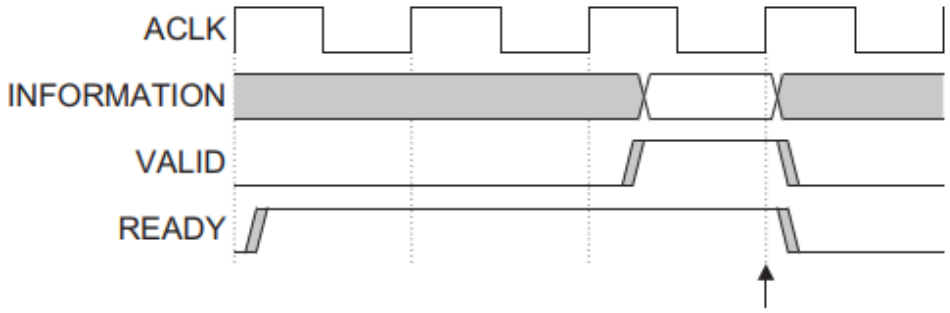

Figure 3-2 READY before VALID handshake

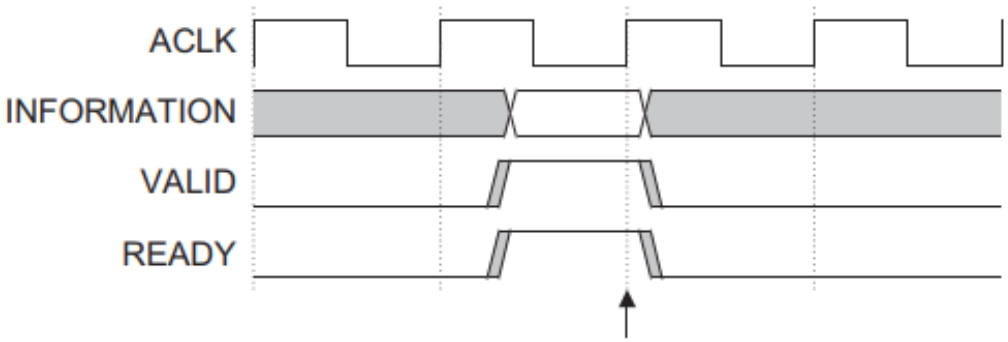

Figure 3-3 VALID with READY handshake

3. ST_WRITE_DATA — Write Data Phase:

Purpose: This is where the **actual memory write happens**

Behavior:

`s_axi_wready = 1` to accept W data if not captured yet

Transition: Immediately go to **ST_WRITE_RESP**

Note : AXI-Lite supports **single-beat writes** only, so no burst → always one cycle.

4. ST_WRITE_RESP — Send Write Response

Purpose:

After writing data to memory, slave must send a response:

```
RESP_OKAY = 2'b00;
RESP_SLVERR = 2'b10;
RESP_DECERR = 2'b11;
```

Behavior:

- `s_axi_bvalid = 1`
- `s_axi_bresp = OKAY`

Note : Slave must hold BVALID high until master asserts BREADY (handshake).

5. ST_READ_ADDR — Read Address Phase

Purpose: This stage captures the read address

Behavior:

- `s_axi_arready = 1` until AR handshake happens

Slave stores the ARADDR in `araddr_reg`.

Transition: Immediately go to **ST_READ_DATA**

6. ST_READ_DATA — Read Data Phase

Purpose: Send read data from memory back to master on the R channel.

Behavior:

- `s_axi_rvalid = 1`
- `s_axi_rdata = mem[araddr_reg >> offset]`
- `s_axi_rresp = OKAY`

Slave waits until master accepts data:

- Master must assert `RREADY`

Transition:

- When `rvalid && rready` → back to **ST_IDLE**

AXI-Lite Internal Addressing & Register Documentation

1.Internal Reg

1. AXI Write Address Register : Stores **AWADDR** when a write address handshake occurs.

```
logic [ADDR_WIDTH-1:0] awaddr_reg;
```

```
if (s_axi_awvalid && s_axi_awready)
awaddr_reg <= s_axi_awaddr;
```

2. AXI Read Address Register : Stores **ARADDR** when a read address handshake occurs.

```
if (s_axi_arvalid && s_axi_arready)
araddr_reg <= s_axi_araddr;
```

3. Captured Write Data Register : This register stores WDATA when WVALID/WREADY handshake happens.

```
if (s_axi_wvalid && s_axi_wready)
wdata_reg <= s_axi_wdata;
```

4. Captured Write Strobe Bits (WSTRB) : Each bit enables a byte write:

Since W may arrive before AW, WSTRB must be stored:

```
logic [(DATA_WIDTH/8)-1:0] wstrb_reg;
```

```
if (s_axi_wvalid && s_axi_wready)
wstrb_reg <= s_axi_wstrb;
```

5. Buffered Read Data Output : Temporarily stores the data read from memory before sending it on the R channel.

3. Internal Memory Array

Simple word-addressed local memory that acts as storage for the AXI-Lite slave.

```
logic [DATA_WIDTH-1:0] mem [0:DEPTH-1];
```

- `DEPTH` = number of words
- Each word = `DATA_WIDTH` bits

4. Capture Flags

These flags track whether a channel handshake has completed.

---

i. **aw_captured** : Set when AW channel handshake is complete. Indicates the write address is stored.

`logic aw_captured;`

```
if (s_axi_awvalid && s_axi_awready)
    aw_captured <= 1'b1;
```

ii. **w_captured :** Set when W channel handshake is complete. Indicates the write data + strobe are stored.

`logic w_captured;`

```
if (s_axi_wvalid && s_axi_wready)
    w_captured <= 1'b1;
```

iii. **ar_captured** : **Set when AR channel handshake is complete.** Indicates read address is stored.

`logic ar_captured;`

```
if (s_axi_arvalid && s_axi_arready)
    ar_captured <= 1'b1;
```

3. Address Decoding Parameters

i. **ADDR_WORD_OFFSET:** Number of **byte offset bits** in an AXI address for the given data bus width.
For 32-bit bus → 4 bytes → `clog2(4) = 2`

Purpose:

AXI addresses are byte-addressed. Memory in RTL is **word-addressed**.

So we drop the lower offset bits:

So we drop the lower offset bits:

```
word_index = AXI_address >> ADDR_WORD_OFFSET
```

`localparam ADDR_WORD_OFFSET = $clog2(DATA_WIDTH/8);`

Meaning:

Number of **byte offset bits** in an AXI address for the given data bus width.

- For 32-bit bus → 4 bytes → `clog2(4) = 2`
- For 64-bit bus → 8 bytes → `clog2(8) = 3`
- For 16-bit bus → 2 bytes → `clog2(2) = 1`

Purpose:

AXI addresses are byte-addressed. Memory in RTL is **word-addressed**.

So we drop the lower offset bits:

```
word_index = AXI_address >> ADDR_WORD_OFFSET
```

ii. Address word width **ADDR_WORD_WIDTH** :Number of bits required to index the internal memory depth.

`localparam ADDR_WORD_WIDTH = $clog2(DEPTH);`

Meaning:

Number of bits required to index the internal memory depth.

If `DEPTH = 128`,

`ADDR_WORD_WIDTH = 7`.

Purpose: Allows clean and safe indexing of memory array `mem[]`.

Memory Write Module:

1.Detect When a Valid Write Transaction Is Ready

A write happens **only when both AW and W are available**

```
if ( (aw_captured || (s_axi_awvalid && s_axi_awready) ) && ( w_captured || (s_axi_wvalid && s_axi_wready)))
```

Your two sources of address/data are:

Condition	Meaning
`aw_captured`	Address was stored earlier
`s_axi_awvalid && s_axi_awready`	Address is arriving now
`w_captured`	WDATA was stored earlier
`s_axi_wvalid && s_axi_wready`	WDATA is arriving now

3. Compute Write Address → Memory Index

```
logic [ADDR_WORD_WIDTH-1:0] windex;

windex = ( (aw_captured ? awaddr_reg : s_axi_awaddr)
        >> ADDR_WORD_OFFSET );
```

This means:

- If AW was captured earlier → use `awaddr_reg`

- Else → use incoming `s_axi_awaddr`

Then you shift right by `ADDR_WORD_OFFSET`.

Example for 32-bit data:

```
ADDR_WORD_OFFSET = 2     // 4 bytes per word
```

So address 0x08 means:

```
0x08 >> 2 = 2 → word index 2 in mem[]
```

4. Check Bounds

```
if (windex < DEPTH) begin
```

Validates that write is inside memory.

5. Iterate Over Each Byte in the Data Word

```
for (int byte = 0; byte < (DATA_WIDTH/8); byte++) begin
```

Example for 32-bit DATA:

```
bytes = 4 → byte=0,1,2,3
```

AXI-Lite uses **byte-level write enables** via WSTRB.

6. The MOST Important Part — Byte-Enable (WSTRB) Handling