AXI4-Compliant Memory-Mapped Slave

Specifications

About AXI4:

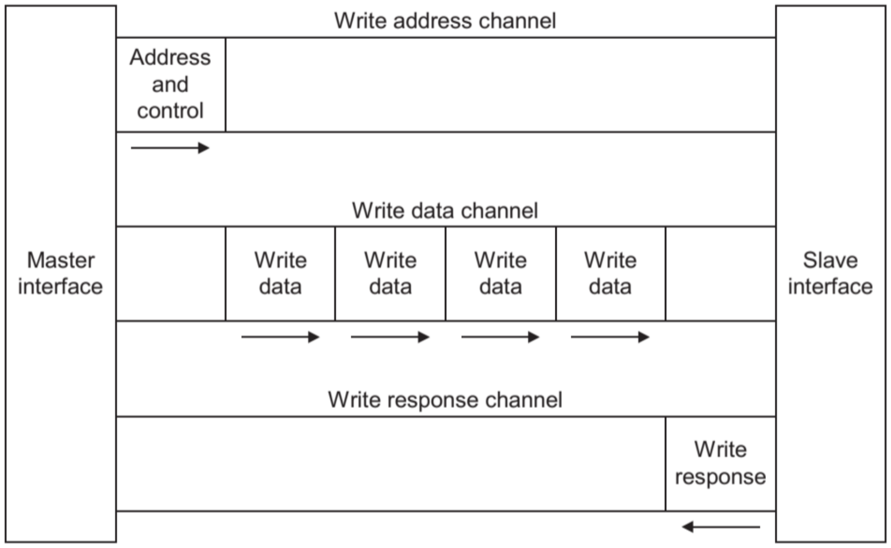
AXI4 stands for **Advanced eXtensible Interface 4**, and it's part of the AMBA (Advanced Microcontroller Bus Architecture) specification developed by ARM. It is a high-frequency system design for communication between Manager and Subordinate components used widely in System-on-Chip (SoC) designs to connect IP blocks. 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.

AXI4 Architecture:

The AXI protocol is burst-based. Every transaction has address and control information on the address channel that describes the nature of the data to be transferred. The data is transferred between master and slave using a write data channel to the slave or a read data channel to the master. In write transactions, in which all the data flows from the master to the slave, the AXI protocol has an additional write response channel to allow the slave to signal to the master the completion of the write transaction. AXI has two channels architecture representing write and read operations.

- Channel Architecture of Write



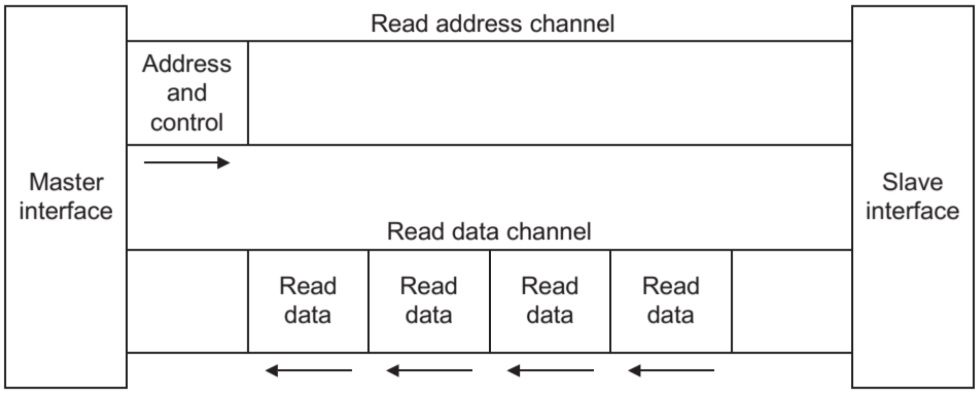
Write address channel The master can assert the AWVALID signal only when it drives valid address and control information. It must remain asserted until the slave accepts the address and control information and asserts the associated AWREADY signal.

Write data channel During a write burst, the master can assert the WVALID signal only when it drives valid write data. WVALID must remain asserted until the slave accepts the write data and asserts the WREADY signal. The default value of WREADY can be HIGH, but only if the slave can always accept write data in a single cycle. The master must assert the WLAST signal when it drives the final write transfer in the burst.

Write response channel The slave can assert the BVALID signal only when it drives a valid write response. BVALID must remain asserted until the master accepts the write response and asserts BREADY.

The default value of BREADY can be HIGH, but only if the master can always accept a write response in a single cycle.

- Channel Architecture of Read



Read address channel

The master can assert the ARVALID signal only when it drives valid address and control information. It must remain asserted until the slave accepts the address and control information and asserts the associated ARREADY signal.
The default value of ARREADY default value is HIGH, although if ARREADY is HIGH then the slave must be able to accept any valid address that is presented to it.

Read data channel

The slave can assert the RVALID signal only when it drives valid read data. RVALID must remain asserted until the master accepts the data and asserts the RREADY signal. Even if a slave has only one source of read data, it must assert the RVALID signal only in response to a request for the data. The master interface uses the RREADY signal to indicate that it accepts the data. The default value of RREADY can be HIGH, but only if the master is able to accept read data immediately, whenever it performs a read transaction. The slave must assert the RLAST signal when it drives the final read transfer in the burst.

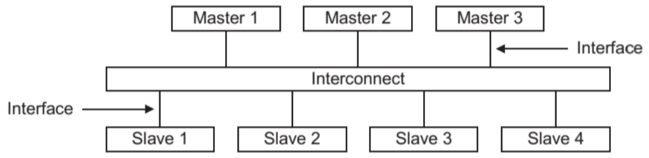
Each of the five independent channels consists of a set of information signals and uses a two-way VALID and READY handshake mechanism.

The information source uses the VALID signal to show when valid data or control information is available on the channel. The destination uses the READY signal to show when it can accept the data. Both the read data channel and the write data channel also include a LAST signal to indicate when the transfer of the final data item within a transaction takes place.

- A typical system consists of a number of master and slave devices connected together through some form of interconnect

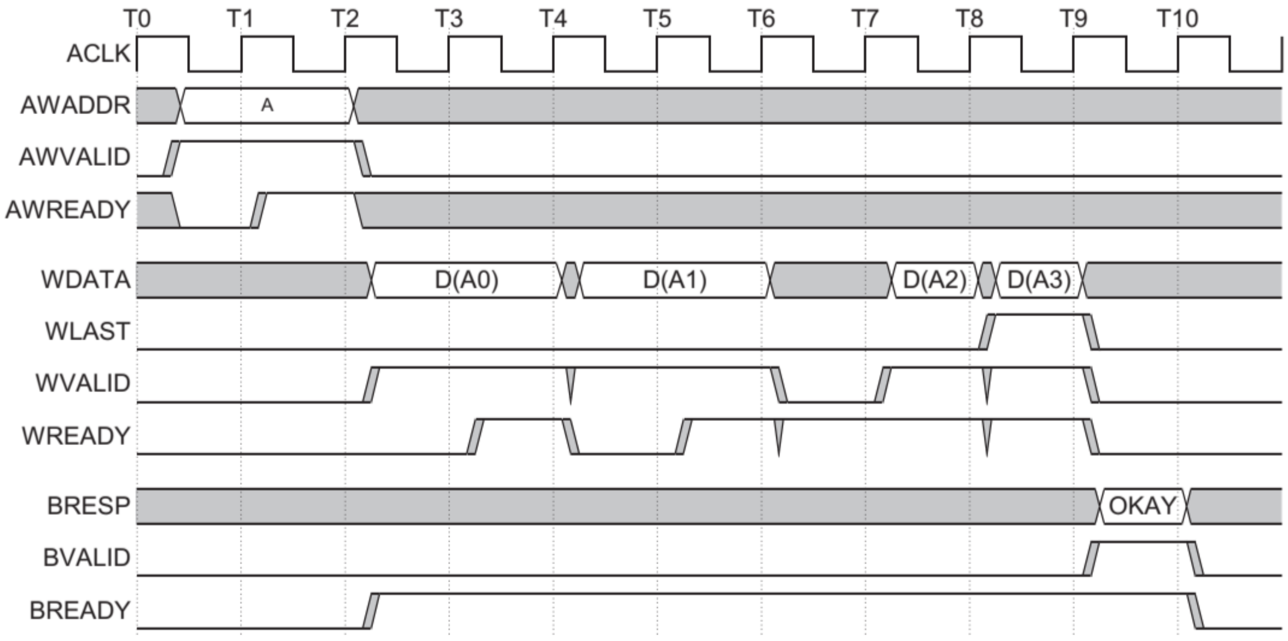
The AXI protocol provides a single interface definition for describing interfaces:

- between a master and the interconnect.
- between a slave and the interconnect.
- between a master and a slave.

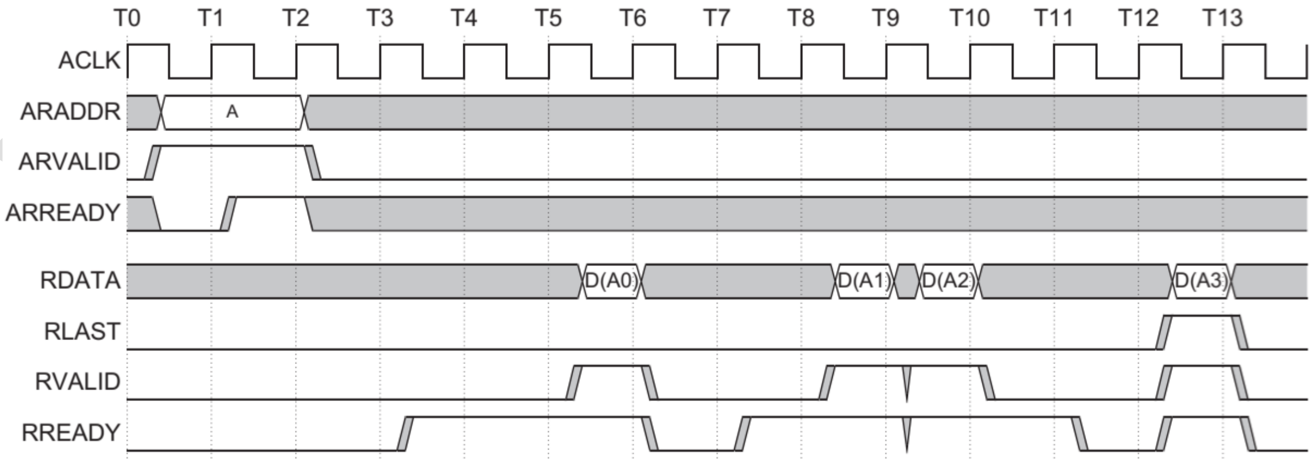


The interface definition enables a variety of different interconnect implementations. The interconnect between devices is equivalent to another device with symmetrical master and slave ports to which real master and slave devices can be connected.

Write burst example



Read burst example

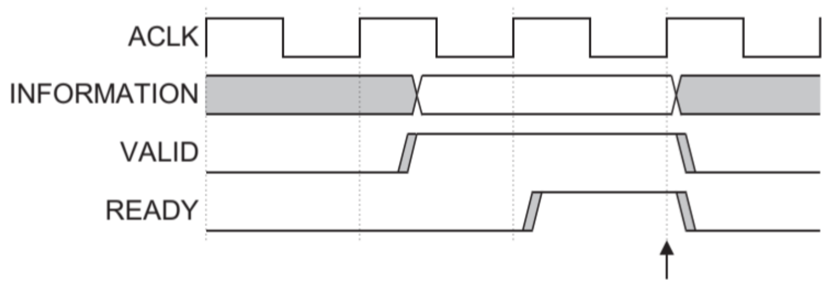


Signal Descriptions

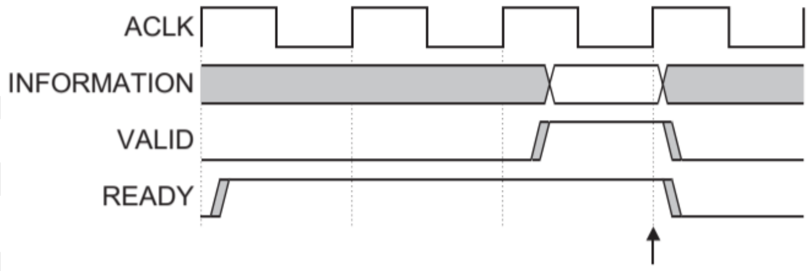
Signal	Port Type	Width	Description
ACLK	**input**	**1**	Global clock signal. All signals are sampled on the rising edge of the global clock.
ARESETn	**input**	**1**	Global reset signal. This signal is active LOW
AWADDR[15:0]	**input**	**16**	Write address. The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst
AWLEN[7:0]	**input**	**8**	Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address
AWSIZE[2:0]	**input**	**3**	Burst size. This signal indicates the size of each transfer in the burst.
AWVALID	**input**	**1**	Write address valid. This signal indicates that valid write address and control information is available: 1 = address and control information available 0 = address and control information not available. The address and control information remain stable until the address acknowledge signal, AWREADY, goes HIGH
AWREADY	**output**	**1**	Write address ready. This signal indicates that the slave is ready to accept an address and associated control signals: 1 = slave ready 0 = slave not ready.
WDATA[31:0]	**input**	**32**	Write data. The write data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide.
WLAST	**input**	**1**	Write last. This signal indicates the last transfer in a write burst.
WVALID	**input**	**1**	Write valid. This signal indicates that valid write data and strobes are available: 1 = write data and strobes available, 0 = write data and strobes not available
WREADY	**output**	**1**	Write ready. This signal indicates that the slave can accept the write data: 1 = slave ready 0 = slave not ready.
BRESP[1:0]	**output**	**2**	Write response. This signal indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
BVALID	**output**	**1**	Write response valid. This signal indicates that a valid write response is available:1 = write response available, 0 = write response not available
BREADY	**input**	**1**	Response ready. This signal indicates that the master can accept the response information. 1 = master ready, 0 = master not ready.
ARADDR[31:0]	**input**	**16**	Read address. The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
ARLEN[7:0]	**input**	**8**	Burst length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address
ARSIZE[2:0]	**input**	**3**	Burst size. This signal indicates the size of each transfer in the burst
ARVALID	**input**	**1**	Read address valid. This signal indicates, when HIGH, that the read address and control information is valid and will remain stable until the address acknowledge signal, ARREADY, is high. 1 = address and control information valid, 0 = address and control information not valid.
ARREADY	**output**	**1**	Read address ready. This signal indicates that the slave is ready to accept an address and associated control signals:1 = slave ready 0 = slave not ready.
RDATA[31:0]	**output**	**32**	Read data. The read data bus can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits
RRESP[1:0]	**output**	**2**	Read response. This signal indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR
RLAST	**output**	**1**	Read last. This signal indicates the last transfer in a read burst.
RVALID	**output**	**1**	Read valid. This signal indicates that the required read data is available and the read transfer can complete:1 = read data available 0 = read data not available
RREADY	**input**	**1**	Read ready. This signal indicates that the master can accept the read data and response information: 1= master ready. 0 = master not ready.

All five channels use the same VALID/READY handshake to transfer data and control information. This two-way flow control mechanism enables both the master and slave to control the rate at which the data and control information moves. The source generates the VALID signal to indicate when the data or control information is available. The destination generates the READY signal to indicate that it accepts the data or control information. Transfer occurs only when both the VALID and READY signals are HIGH.

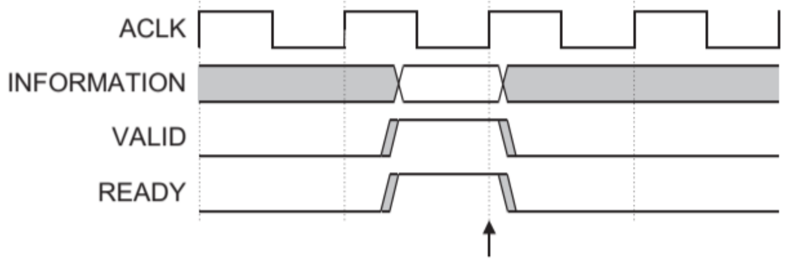
VALID before READY handshake



READY before VALID handshake



VALID with READY handshake



Project Spec

This project implements a simplified AXI4-based system that enables communication between an AXI4 master and an internal memory module. The goal is to model, simulate, and verify the behavior of AXI4 transactions such as **burst reads and writes**, address decoding, and proper data handshaking.

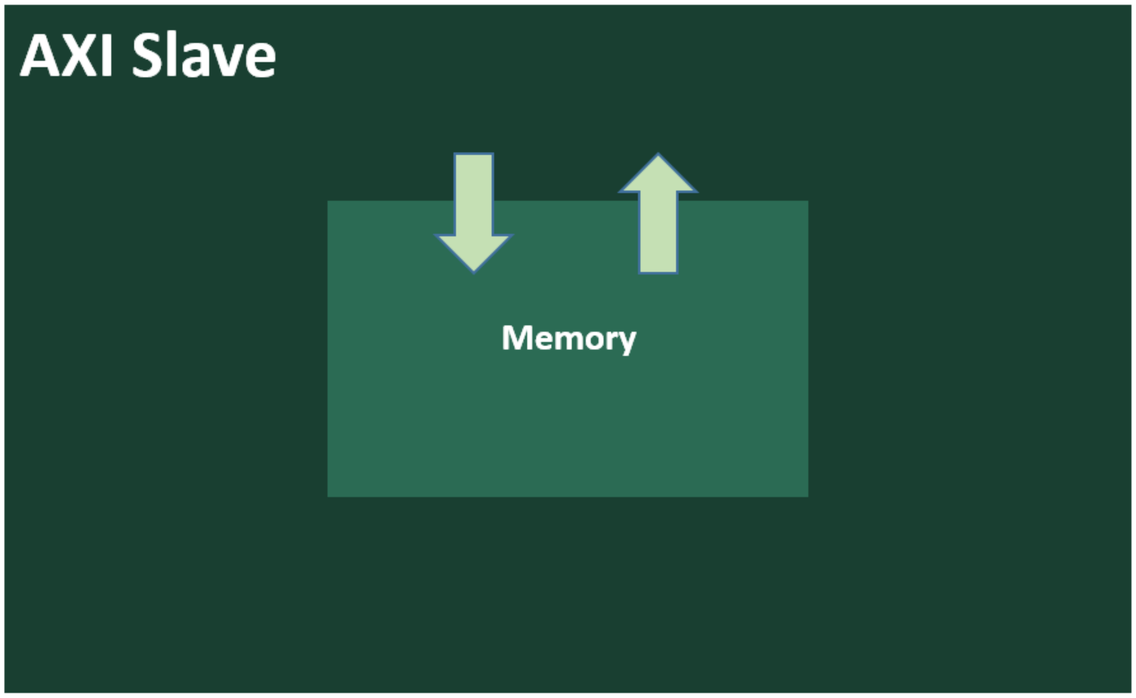
The design includes:

- An AXI4 **slave interface** module that receives commands and data from an AXI4 master (your testbench in that case).
- A **synchronous memory block** that stores 32-bit data in a 4 KB memory space.

This system follows key rules of the AXI4 protocol, such as:

- Burst handling through AWLEN, AWSIZE, ARLEN, and ARSIZE
- Write response generation
- 4KB boundary compliance

The entire design is parameterized for reusability and scalability.



1. axi4_memory

Purpose: Implements a synchronous single-port RAM, used as the destination and source of AXI4 read/write transactions.

Item	Description
Interface	Internal memory bus
Access Type	Word-addressable (32-bit word)
Depth	1024 locations
Word Size	32 bits
Total Size	4 KB (1024 × 4 bytes)

Key Signals:

- **mem_en, mem_we:** Enable and write-enable controls
- **mem_addr, mem_wdata, mem_rdata:** Data/address ports

---

2. axi4_slave

Purpose: Interprets and manages AXI4 transactions including burst reads/writes, ID management, and protocol enforcement. The axi4 slave can execute write and read operations on the connected memory based on the transaction type.

Item	Description
Compliant With	AMBA AXI4 Specification
Supported Features	Burst, ready/valid handshake
Boundary Check	Enforces 4KB boundary for bursts

---

1. Write Operation Specification Context

The **AXI4 write operation** follows the **three-channel handshaking** mechanism as per the AXI4 protocol: **write address (AW), write data (W), and write response (B)**. Below is the lifecycle of a typical write transaction:

Write Operation Flow:

1. **Write Address Phase**:
   - The master sends a valid write address **AWADDR**, burst length **AWLEN**, and burst size (**AWSIZE**) along with **AWVALID**.
   - The slave sets **AWREADY** high when ready to accept the address. In our design it is set at one by default to improve timing efficiency.
   - On **AWVALID** && **AWREADY**, the slave:
     - Captures the write address and burst parameters.
     - Then move to writing the data.
2. **Write Data Phase**:
   - **WVALID** indicates the master is sending write data (**WDATA**), and **WLAST** signals the final data beat in the burst which should be driven with last data sent in the brust.
   - The slave sets **WREADY** after recognizing **WVALID** is high to accept data.
   - For each valid data transfer:
     - If within memory bounds and not crossing a 4KB boundary, the data is written to internal memory (**mem_we = 1**).
     - If the address is invalid or crosses the 4KB boundary, no write occurs, and an error response is prepared.
3. **Write Response Phase**:
   - Once all beats are transferred or **WLAST** is recognized, the slave:
     - Lowers **WREADY**.
     - Sets **BVALID = 1** and provides a response (BRESP = **OKAY or SLVERR**), the design handles those only
   - When the master (your TB) asserts **BREADY**, the response is acknowledged, and the FSM resets to W_IDLE.

---

2. Read Operation Specification Context

The **AXI4 read operation** also follows a handshaking model but uses two channels: **read address (AR) and read data (R)**.

Read Operation Flow:

1. **Read Address Phase**:
   - The master provides a valid address (**ARADDR**), burst length (**ARLEN**), and size (**ARSIZE**) along with **ARVALID**.
   - The slave sets **ARREADY** high when it is ready to accept the address.
   - On **ARVALID** && **ARREADY**, the slave:
     - Stores the address and burst parameters.
     - Moves to reading the data from memory phase.

2. **Read Data Phase**:
   - In **R_ADDR**, the first memory access is initiated if address and boundary are valid.
   - FSM moves to **R_DATA** and:
     - On each clock cycle, if memory access is valid:
       - Reads the data from internal memory.
       - Sets **RVALID = 1**, **RDATA** with read value in same cycle, **RRESP = OKAY.**
       - If it's the final beat in the brust, **RLAST = 1.** If not it continues reading the data from memory until the end of beats un the brust **[**number of beats is the **ARLEN + 1]**
     - If **RREADY && RVALID**, proceeds to next beat or returns to **R_IDLE.**
     - After sampling de-assertion of **RVALID** in your TB de-assert the **RREADY** the next cycle directly.

3. **Error Conditions**:
   - If address is out of range or crosses a 4KB boundary, **RDATA** = 0, **RRESP** = SLVERR, **RLAST** = 1**.**

3. Boundary Checks and Address Alignment Specification Context

AXI4 supports **burst transfers**, which can lead to complex address alignment and boundary crossing issues. This design applies **compliance checks** to ensure proper memory behavior.

Boundary Checks Implemented:

1. **4KB Boundary Check**:

   - AXI4 protocol mandates that bursts must not cross 4KB boundaries. Which means the AWADDR or ARADDR must not exceed the 4KB boundaries to work normally without errors.
   - If it exceeds, the burst is **invalid**, and a `SLVERR` response is generated.

2. **Memory Range Check:**

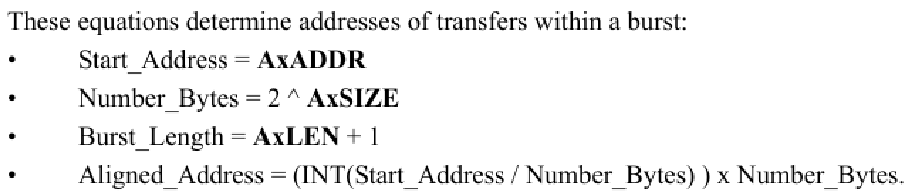
   - Internal memory is word-addressable.
   - The design shifts addresses right by 2 (>> 2) to convert from byte address to word address.
   - A range check confirms: (addr >> 2) < MEMORY_DEPTH

Note that

- AWLEN/ARLEN determines the number of beats in the brust, such that: no of beats = AWLEN + 1 ,for write operation and for read operation: no of beats = ARLEN + 1.

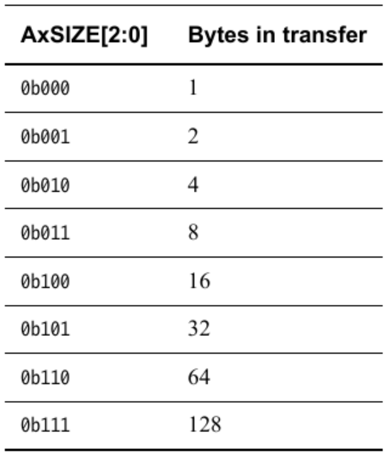
- AWSIZE/ARSIZE determines the number of bytes per beats and as in our design we are working only with data width 32bit it needs to be fixed at what?

Example: AWSIZE = 2 ⇒ 4 bytes (word)
The internal memory expects word-aligned accesses: Hence, addresses are divided by 4 (>> 2) to access word locations. Any unaligned address or transfer size mismatch may lead to invalid memory operations or ignored transactions.



These equations determine addresses of transfers within a burst:

- Start_Address = **AxADDR**
- Number_Bytes = 2 ^ **AxSIZE**
- Burst_Length = **AxLEN** + 1
- Aligned_Address = (INT(Start_Address / Number_Bytes)) x Number_Bytes.



AxSIZE[2:0]	Bytes in transfer
0b000	1
0b001	2
0b010	4
0b011	8
0b100	16
0b101	32
0b110	64
0b111	128

Project Goals:

You are required to **build a complete SystemVerilog testbench environment** to verify the AXI4 design with the integrated memory. This environment should serve as a practical application of **Coverage-Driven Verification (CDV)** principles using key features of SystemVerilog..

Testbench Must Include:

- **SystemVerilog Classes**:
  - To encapsulate transaction-level representations (e.g., `axi4_packet`).
  - Stimulus generation, drivers, and checkers.
- **Constrained Randomization**:
  - Randomize AXI signals (`AWADDR`, `ARLEN`, `WDATA`, etc.) under valid protocol rules
  - Apply constraints to avoid illegal scenarios, and explore corner cases.
- **Interfaces**:
  - Define clean and reusable interfaces for the AXI4 protocol.
  - Use modports to manage signal direction and access in various components.
- **Functional Coverage**:
  - Cover all important protocol features: burst lengths, data sizes, address regions, and write/read behaviors.
  - Use covergroups and coverpoints properly with bins for meaningful measurement.
- **Assertion-Based Checks (CRT)**:
  - Use concurrent assertions (SystemVerilog Assertions) to monitor protocol rules and handshake behavior.

🎯 Coverage Goal:

- Achieve **100% functional coverage**, unless specific exclusions apply.
- Achieve **100% Code Coverage**
- Achieve **100% Assertion coverage**.

If 100% is not achieved:

  - Provide a clear and **justified report** of uncovered bins.
  - Explain why certain cases are excluded (e.g., illegal, unreachable, or out of scope)

Deliverables:

U have to deliver **two separated files**

- Zip Folder contains:
  - The implemented **.sv** files and the **design files** and **run.do file**.

- **Separated PDF** contains:
  - Snippets for the implemented code.
  - Snippet for the waveform shows the different testcases clearly.
  - Snippets for the logs show the all printed values.
  - Snippets for functional coverage report.
  - Snippets for code coverage report for all parameters [Line / toggle / branch /condition/…]
  - Snippets for Assertions Coverage report.
  - Snippet for Do file you have used for automating the process.

The delivered zip file must be named like **your_name_project.rar** for example: **Hassan_Khaled_project.rar** also the PDF file **Hassan_Khaled_project.pdf**

Good Luck