**AHB2AXI Bridge Verification Environment Specification**

Table of Contents

[Document history 2](#_Toc109746558)

[1.Introduction 4](#_Toc109746559)

[1.1 Scope 4](#_Toc109746560)

[1.2 Abbreviations 4](#_Toc109746561)

[1.3 References 4](#_Toc109746562)

[2. AHB2AXI Bridge Overview 5](#_Toc109746563)

[2.1. AHB2AXI Features 5](#_Toc109746564)

[2.2. AHB2AXI Interfaces 5](#_Toc109746565)

[3.Verification Environment 5](#_Toc109746566)

[3.1 Environment Overview 5](#_Toc109746567)

[3.2 Components description 6](#_Toc109746568)

[3.4 Checkers 6](#_Toc109746569)

[3.5 Coverage 6](#_Toc109746570)

[3.6 Tests 6](#_Toc109746571)

[4. Simulation flow 6](#_Toc109746572)

[5. How to run a test 7](#_Toc109746573)

[6. Limitations and assumptions 7](#_Toc109746574)

# Document history

| Version | Date | Approved by | Created by | Description |
| --- | --- | --- | --- | --- |
| 0.1 | 10.07.2023 | Liviu Ababei | Ioana Ailenei | Document created |
| 1.0 |  | Liviu Ababei | Ioana Ailenei | Revised by Liviu Ababei |

# 1.Introduction

## 1.1 Scope

This document describes the verification environment for the AHB2AXI Bridge.

## 1.2 Abbreviations

**DUT =>** Device Under Test

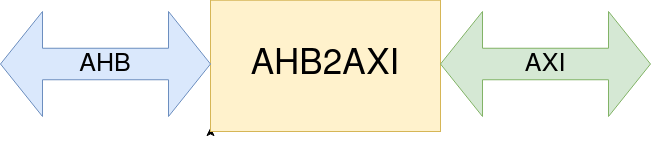
**AHB**  **=>** Advanced High-permormance Bus

**AXI**  **=>** Advanced eXtensible Interface

## 1.3 References

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Name** | **Revision** | **Description** |
| 1. | AHB2AXI Metric plan.xlsx | 1.0 | Coverage, Checkers and Test plans |
| 2. | 1. AHB2AXI\_bridge\_specs.xlsx | x.x | AHB2AXI Bridge specification |

# 2. AHB2AXI Bridge Overview



1. Figure 1. The AHB2AXI Bridge overview diagram

## 2.1. AHB2AXI Bridge Features

* single clock domain, i.e. same clock is used for AHB and AXI interface
* translates AHB to AHB2AXI Bridge; both buses are 32 bit data, 32 bit address
* for write, a data buffer is used to allow proper transfer from AHB to AXI; 32x16x32 (32 outstanding, 16 cycles per burst max, 32 bits per cycle)
* for read, there is no data buffering as AHB protocol implies data for previous transfer to be available before issuing a new request
* AXI out of order responses is supported for writes. Not the case for read
* AXI 4K protection is implemented - a transfer going over 4K will be split into 2 transfers (consider also the WRAP case here !!! If applicable ??? )
* AXI can generate the output transactions in 2 modes: no-outstanding, outstanding (configurable by writing a CTRL register)
* in case outstanding transfers are used, the AXI maximum number of outstanding transfers is defined by the CTRL register
* AXI can be configured to allow WRAP transfers or not, using the CTRL register
* in case on the AHB we have a WRAP transfer and the AXI does not allow WRAPs, transfer will be converted to INCR on AXI
* CTRL register is accessible using AHB at address 0x100
  + [0] - Outstanding transfers enable; 0: no-outstanding; 1 - outstanding
  + [1] - WRAP supported; 0: WRAP not supported; 1 - WRAP supported
  + [7:2] - reserved
  + [13:8] - max outstanding transfers. Possible values: 4, 8, 16, 32. It applies for both read and write channels.
  + [31:14] - reserved
* the CTRL register must be updated only when there is no outstanding traffic on AHB/AXI interfaces, we can consider this is programmed first after HW reset and can be changed only after a HW reset
* access to addresses: 0x1000\_0000 to 0xFFFF\_FFFF is for accessing the control space (including the CTRL register above). Writes to addresses that are not allocated here are ignored, reads will return 0 as read data + OKAY resp on AHB.
* access to addresses: 0x1000\_0000 to 0xFFFF\_FFFF is for data transfers
* an error response received on AXI for reads, must be propagated on the AHB interface. For writes this is not possible as writes can be accepted in advance by the bridge.

## 2.2. AHB2AXI Bridge Interfaces

### AXI Interface

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1. Signal name | 1. Direction | 1. Width | 1. Description |
| 1. global | 1. aclk | 1. I | 1. 1 | 1. Clock |
| 1. aresetn | 1. I | 1. 1 | 1. HW reset, active low |
| 1. write address channel | 1. awvalid | 1. O | 1. 1 | 1. Write request |
| 1. awid | 1. O | 1. 4 | 1. Write ID. Between 0 and 15 |
| 1. awaddr | 1. O | 1. 32 | 1. Write address. Access is: Byte ([1:0] can take any value) , Halfword ([1:0] can be 0 or 2) or Word alligned ([2:0] is always 0) |
| 1. awlen | 1. O | 1. 4 | 1. Burst length (no of cycles in burst - 1) |
| 1. awsize | 1. O | 1. 3 | 1. Byte, Halfword, Word. Byte and Halfword is supported only if burst length is 1. |
| 1. awburst | 1. O | 1. 2 | 1. Burst type: FIXED, INCR, WRAP |
| 1. awready | 1. I | 1. 1 | 1. Write request acknowledge |
| 1. write data channel | 1. wvalid | 1. O | 1. 1 | 1. Write data valid |
| 1. wdata | 1. O | 1. 32 | 1. Write data |
| 1. wstrb | 1. O | 1. 4 | 1. Write strobe. For bursts (more than 1 cycle) is all ones. For bursts of 1 cycle, it can be between 1 and 15. If size is not Word, make sure the bits set to 1 are matching the selected Bytes. |
| 1. wlast | 1. O | 1. 1 | 1. Write data last |
| 1. wready | 1. I | 1. 1 | 1. Write data ready |
| 1. write  response channel | 1. bvalid | 1. I | 1. 1 | 1. Write response valid |
| 1. bid | 1. I | 1. 4 | 1. Write response ID. |
| 1. bready | 1. O | 1. 1 | 1. Write response ready - stuck to 1 |
| 1. read address channel | 1. arvalid | 1. O | 1. 1 | 1. Read request |
| 1. arid | 1. O | 1. 4 | 1. Read ID. Between 0 and 15 |
| 1. araddr | 1. O | 1. 32 | 1. Read address. Access is: Byte ([1:0] can take any value) , Halfword ([1:0] can be 0 or 2) or Word alligned ([2:0] is always 0) |
| 1. arlen | 1. O | 1. 4 | 1. Burst length (no of cycles in burst - 1) |
| 1. arsize | 1. O | 1. 3 | 1. Byte, Halfword, Word. Byte and Halfword is supported only if burst length is 1. |
| 1. arburst | 1. O | 1. 2 | 1. Burst type: FIXED, INCR, WRAP |
| 1. arready | 1. I | 1. 1 | 1. Read request acknowledge |
| 1. read data  channel | 1. rvalid | 1. I | 1. 1 | 1. Read data valid |
| 1. rdata | 1. I | 1. 32 | 1. Read data |
| 1. rlast | 1. I | 1. 1 | 1. Read data last |
| 1. rid | 1. I | 1. 4 | 1. Read response ID. |
| 1. rresp | 1. I | 1. 2 | 1. Read response status: all posibilities supported |
| 1. rready | 1. O | 1. 1 | 1. Read data ready. |

### AHB Interface

* *global*

|  |  |  |
| --- | --- | --- |
| 1. **Name** | 1. **Direction** | 1. **Width** |
| 1. clk | 1. input |  |
| 1. rst\_n | 1. input |  |

* *response*

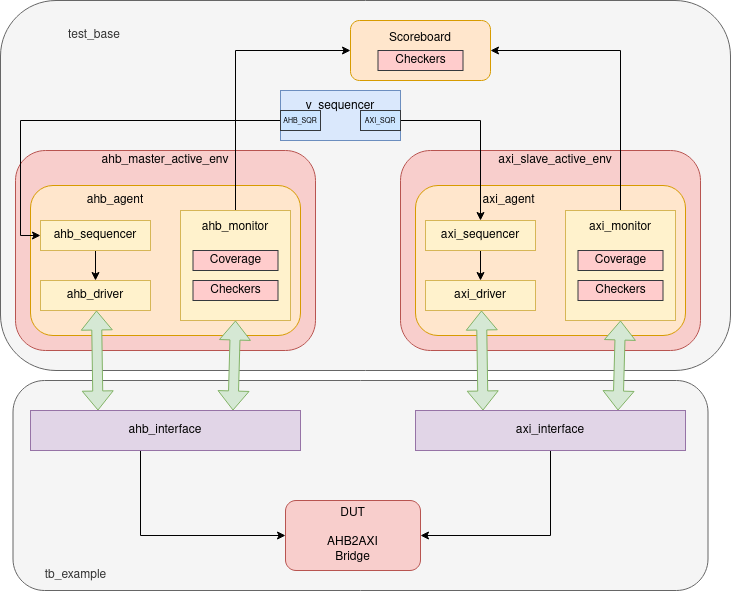
|  |  |  |
| --- | --- | --- |
| 1. **Name** | 1. **Direction** | 1. **Width** |
| 1. hrdata | 1. output | 1. [31:0] |
| 1. hreadyout | 1. output |  |
| 1. hresp | 1. output |  |

* *request*

|  |  |  |
| --- | --- | --- |
| 1. **Name** | 1. **Direction** | 1. **Width** |
| 1. hsel | 1. input |  |
| 1. htrans | 1. input | 1. [1:0] |
| 1. hburst | 1. input | 1. [2:0] |
| 1. hsize | 1. input | 1. [2:0] |
| 1. haddr | 1. input | 1. [31 : 0] |
| 1. hwrite | 1. input |  |
| 1. hwdata | 1. input | 1. [31 : 0] |
| 1. hwstrobe | 1. input | 1. [3:0] |

# Verification Environment

## 3.1 Environment Overview

* 1. The AHB2AXI Bridge verification environment diagram is presented in Figure 2.
  2. 

1. Figure 2. The AHB2AXI Bridge Verification Environment Diagram

## 3.2 Components description

**Sequence**: series of transactions

**Sequencer**: It generates sequences of transaction for the driver with different constraints based on the test sequences. The AHB Sequencer controls the generation and sequencing of AHB transactions. The AXI Sequencer controls the generation and sequencing of AXI transactions

**Virtual Sequencer**: It defines the high-level sequence of transactions to be executed on the bus. It provides a sequence of transactions to the Sequencer, which then generates the corresponding bus cycles. The Virtual Sequencer allows for flexible and dynamic test scenario creation by specifying the desired sequence and conditions for different transactions.

**Driver**: The AHB Driver is responsible for generating AHB transactions on the AHB bus. It receives commands from the AHB Sequencer and converts them into AHB bus cycles, ensuring proper formatting of the AHB protocol. The AXI Driver generates AXI transactions to be sent to the AXI bus. It receives commands from the AXI Sequencer and converts them into AXI bus cycles, adhering to the AXI protocol specifications.

**Monitor**: It extracts relevant information from the bus activity, performs checks on the transactions, and updates coverage information. The AHB Monitor captures and observes the AHB bus activity. The AXI Monitor captures and observes the AXI bus activity.

**Agent:** The AHB Agent acts as the interface between the testbench and the DUT. It includes the AHB Driver, AHB Sequencer, and AHB Monitor. The AXI Agent acts as the interface between the testbench and the DUT. It includes the AXI Driver, AXI Sequencer, and AXI Monitor. The Agent coordinates the generation of transactions, sequencing, and monitoring of the bus activity. It provides the necessary connectivity and control to interact with the DUT.

**Scoreboard:** receives transaction information from both the AHB and AXI Monitors. It compares the expected and observed transactions, verifies the correctness of the data transfers, and raises any discrepancies or mismatches as errors. The Scoreboard is responsible for monitoring and ensuring the protocol compliance and data integrity between the AHB and AXI interfaces

## 3.4 Checkers

The checkers list can be found in AHB2AXI Bridge metric driven plan.xlsx, the “Checkers” sheet.

## 3.5 Coverage

The functional coverage description can be found in AHB2AXI Bridge metric driven plan.xlsx, the “Coverage” sheet.

## 3.6 Tests

The tests’ description can be found in AHB2AXI Bridge metric driven plan.xlsx, the “Tests” sheet.

# 4. Simulation flow

# 5. How to run a test

[Script to use]

1. ../example/run.ius

[From where to run]

1. ../example

[Example of command line parameters]

1. run\_ius test\_example\_1

# 6. Limitations and assumptions

[What doen’t work]

[What assumptions you took on DUT functionality]

[What is still to be verified]

[What is not supported in the VE from the DUT functionality]