YosysHQ SVA AXI Properties - Verification Plan User Guide (Beta version)

October 3, 2021

Date	Comment
02/06/2021	Creation of the document.
17/06/21	Minor updates.
14/07/21	Minor updates.
16/08/21	Improvement: explaining parameters.
17/08/21	Property Matrix AW
18/08/21	Property Matrix AW
03/10/21	Verification methodology

Contents

I Naming Convention **6**

1 Rules or Assertions **6**

2 Constrains or Assumptions **6**

3 Witness or Covers **6**

II Technical Requirements **7**

4 Limitations and Restrictions **7**

III Technical Information **8**

5 AXI SVA Formal IP Configurations **8**
5.1 Source . . . 8
5.1.1 Use case . . . 8
5.2 Destination . . . 8
5.2.1 Use case . . . 9
5.3 Monitor . . . 9
5.3.1 Use case . . . 10
5.4 Constrain . . . 10
5.4.1 Use case . . . 11

6 Architecture **12**
6.1 Structural view . . . 12

7 Using the AXI SVA Formal IPs **15**
7.1 Bind to a source DUT . . . 15
7.2 Bind to a destination DUT . . . 15
7.3 Monitor Example . . . 15
7.4 Constraint Example . . . 15

IV Verification Plan **16**

8 General Methodology **16**
8.1 RTL Bring-up . . . 16
8.2 Protocol certification (verification) of existing IP . . . 18

9 AMBA AXI4 Property Matrix **19**
9.1 Write Address Channel (AW) Properties . . . 19

10 AMBA AXI4 Specification Description **21**

11 Source Functional Rules **21**
11.1 AMBA AXI4 Write Address Channel . . . 21
11.1.1 AWID . . . 22
11.1.2 AWADDR . . . 23
11.1.3 AWLEN . . . 26
11.1.4 AWSIZE . . . 30
11.1.5 AWBURST . . . 32
11.1.6 AWLOCK . . . 34
11.1.7 AWCACHE . . . 36
11.1.8 AWPROT . . . 38
11.1.9 AWQOS . . . 39
11.1.10 AWREGION . . . 40
11.1.11 AWUSER . . . 41
11.1.12 AWVALID . . . 42
11.1.13 AWREADY . . . 44
11.1.14 AW Cover Properties . . . 45
11.1.15 AW Optional Properties . . . 48
11.1.16 AW Configuration Properties . . . 49
11.2 AMBA AXI4 Write Data Channel . . . 51
11.2.1 WID . . . 51
11.2.2 WDATA . . . 51
11.2.3 WSTRB . . . 51
11.2.4 WLAST . . . 51
11.2.5 WUSER . . . 51
11.2.6 WVALID . . . 52
11.2.7 WREADY . . . 52
11.3 AMBA AXI4 Read Address Channel . . . 52
11.3.1 ARID . . . 52
11.3.2 ARADDR . . . 52
11.3.3 ARLEN . . . 52
11.3.4 ARSIZE . . . 52
11.3.5 ARBURST . . . 52
11.3.6 ARLOCK . . . 53
11.3.7 ARCACHE . . . 53
11.3.8 ARPROT . . . 53
11.3.9 ARQOS . . . 53
11.3.10 ARREGION . . . 53
11.3.11 ARUSER . . . 53
11.3.12 ARVALID . . . 53
11.3.13 ARREADY . . . 53

12 Destination Functional Rules **54**
12.1 AMBA AXI4 Write Response Channel . . . 54
12.1.1 BID . . . 54
12.1.2 BRESP . . . 54
12.1.3 BUSER . . . 55
12.1.4 BVALID . . . 55
12.1.5 BREADY . . . 55
12.2 AMBA AXI4 Write Address Channel . . . 55

13 Optional Properties **55**
13.1 Knowledge Articles and Erratas 55
13.1.1 KA001346 55

14 Helper Logic **56**
14.1 The Boundary of 4KB in AxADDR 56

Part I
Naming Convention

1 Rules or Assertions

All assertions are named "ap_CHANNEL_<name>" where:

- ap: **a**ssertion **p**roperty.
- CHANNEL: The AXI4 channel where the property applies: AW, W, B, AR, R.
- <name>: A meaningful name for the rule to check.

2 Constrains or Assumptions

All assumptions are named "cp_CHANNEL_<name>" where:

- cp: **c**onstrain **p**roperty.
- CHANNEL: The AXI4 channel where the property applies: AW, W, B, AR, R.
- <name>: A meaningful name for the rule to check.

3 Witness or Covers

All covers are named "wp_CHANNEL_<name>" where:

- wp: **w**itness **p**roperty.
- CHANNEL: The AXI4 channel where the property applies: AW, W, B, AR, R.
- <name>: A meaningful name for the rule to check.

Part II
Technical Requirements

4 Limitations and Restrictions

- Read data interleave is not supported by the SVA AXI Properties. That means, the IP will assume or enforce that the same RID is kept before the last beat of a transaction.
- The Formal IP is restricted to support single transaction ID, which also implies in-order transactions and no interleaving.
- No transaction caching, buffering and modifications are supported.
- Short bounds on round trip time.

Part III
Technical Information

5 AXI SVA Formal IP Configurations

5.1 Source

When the AXI4 Formal IP is instantiated and configured as a source, it provides constraints for the destination inputs and checks (assertions) for the destination outputs.

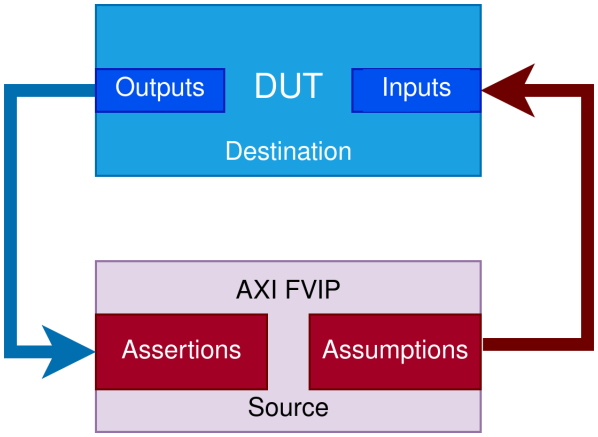

Figure 1: Configuring the FVIP as Source.

5.1.1 Use case

Configure the AXI IP as **source** to verify a destination using model checking.

5.2 Destination

When the AXI4 Formal IP is instantiated and configured as a destination, it provides assertions for source outputs and constraints for source inputs.

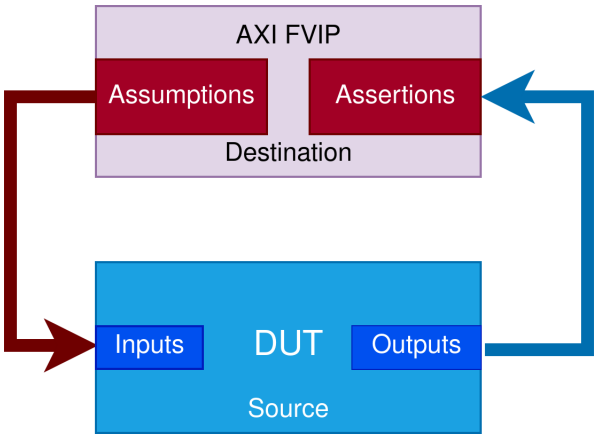

Figure 2: Configuring the FVIP as Destination.

5.2.1 Use case

Configure the AXI IP as destination to verify a source using model checking.

5.3 Monitor

In the monitor configuration, the AXI Formal VIP provides assertions for both source and destination to verify protocol correctness between these two instances.

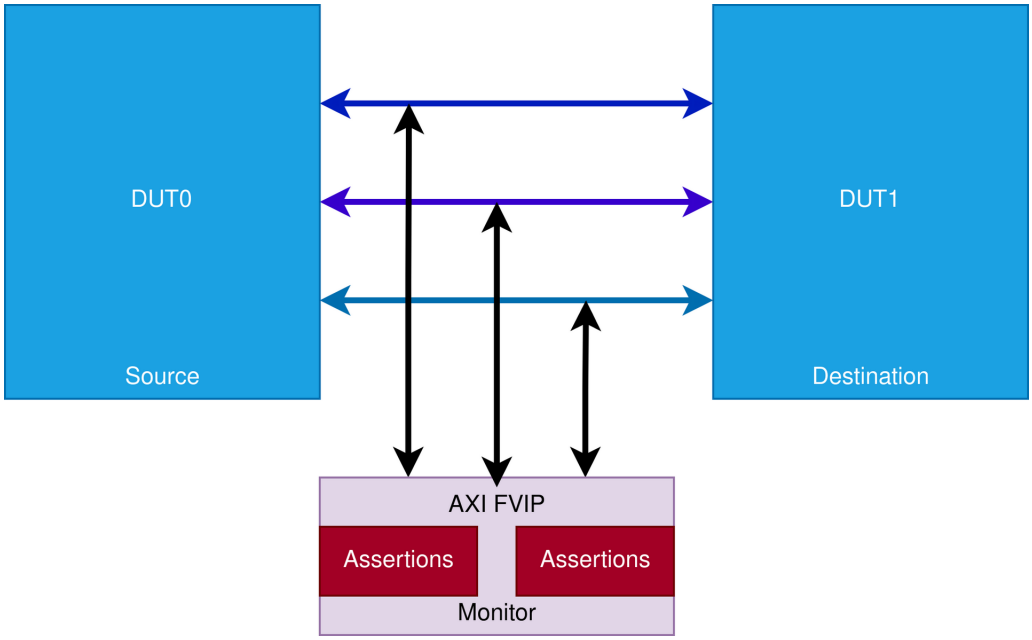

Figure 3: Configuring the FVIP as Monitor.

5.3.1 Use case

Configure the AXI IP as monitor to check protocol correctness in scenarios where no constrains are needed because they are provided by an entity that needs to be checked as well.

5.4 Constrain

The AXI IP can provide "stimulus" or constraints for both source and destination entities, by configuring it as constrain.

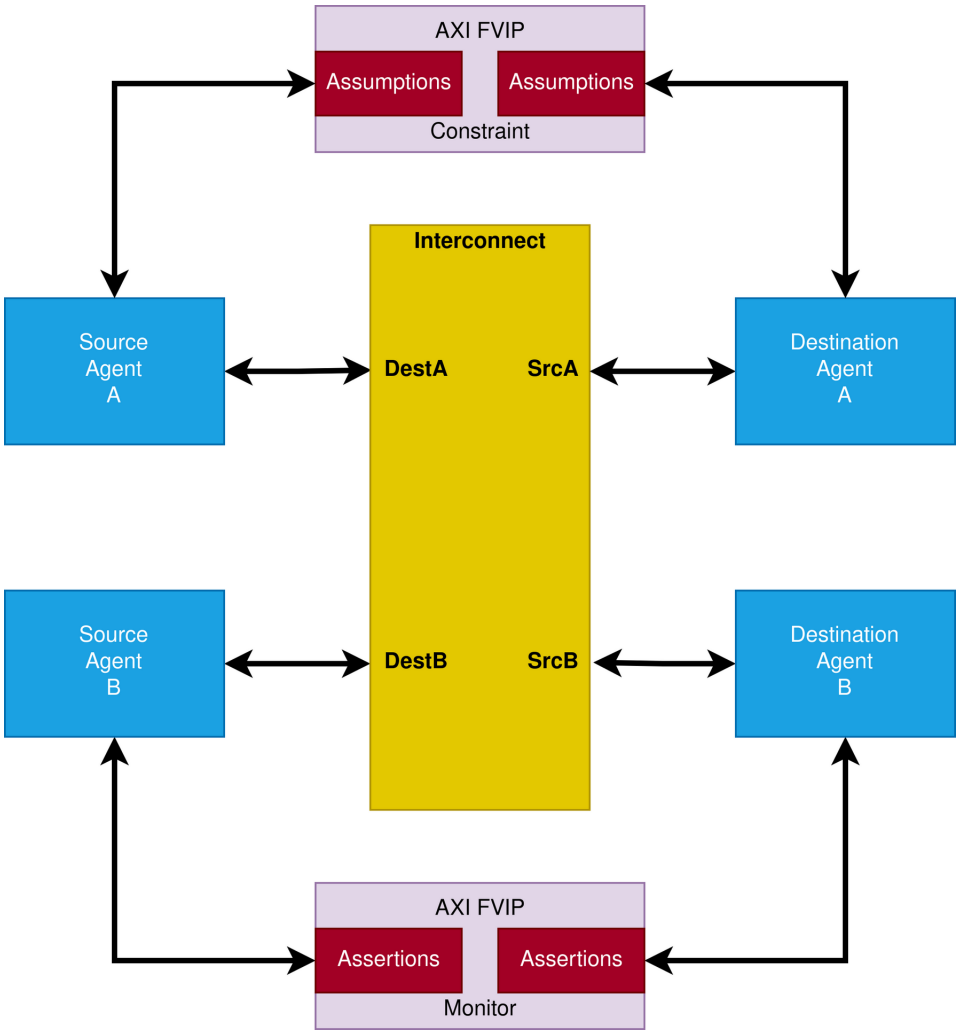

Figure 4: Configuring the FVIP as Monitor.

5.4.1 Use case

Configure the AXI IP as constraint to generate stimulus for certain modules, for example, as source or destination agents for a crossbar validation.

6 Architecture

6.1 Structural view

The AXI SVA Formal IP provides a separate package-module for each of the five AXI channels that can be used either as a separate entity to validate each channel in a different session (helpful for RTL development), or as a single entity to validate an entire IP. the Figure 5 shows a diagram of the structural organisation of the formal IP.

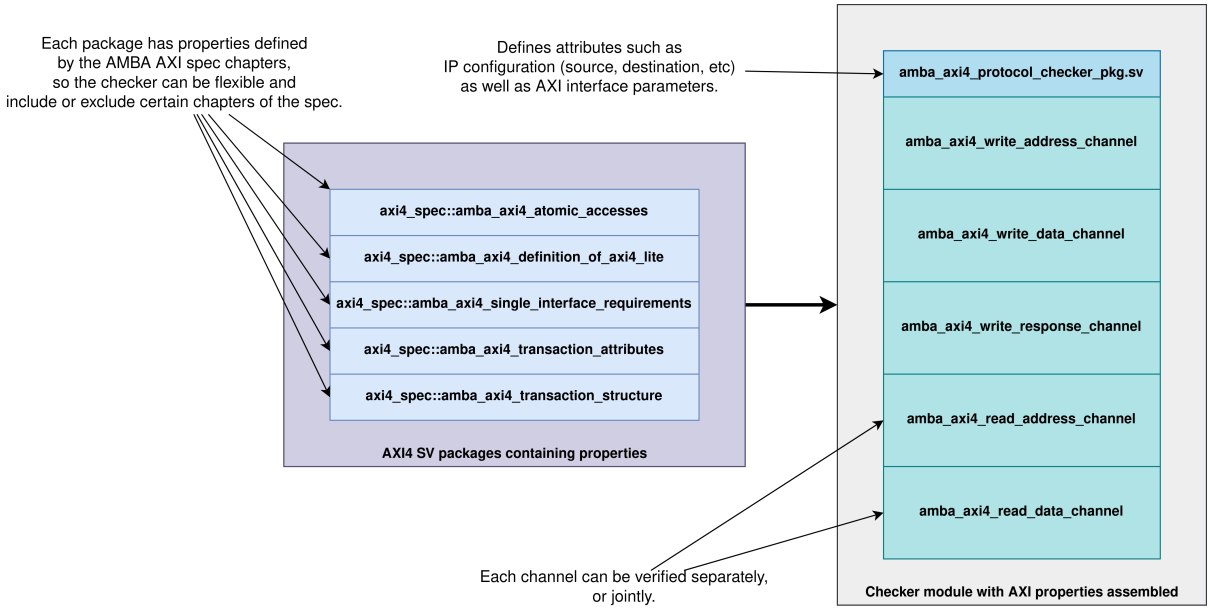

Figure 5: Structural diagram of the AMBA AXI4 SVA Formal IP.

The description of each of the files is shown below, **in the suggested order they should be processed**:

Packages provides encapsulation for various purposes as shown in the Table 2.

Packages (axi4_spec).	Description
amba_axi_protocol_checker_pkg.sv	• Encoded values for xRESP (OKAY, EXOKAY, SLVERR, DECERR). • Encoded values for AxLOCK (NORMAL, EXCLUSIVE). • Busrt type encoding (FIXED, INCR, WRAP and RESERVED). • Values for AxSIZE. • The AXI Formal IP configuration parameters (SOURCE, DESTINATION, MONITOR or CONSTRAINT). • The AXI4 bus types configuration parameters (AXI4LITE, AXI4FULL). • The main configuration parameters for the AXI Formal IP, to configure the IP to match design characteristics as well as enable/disable verification goals (see Section <TBD> for an explanation of this structure).
amba_axi4_single_interface_requirements.sv	• Properties defined in Chapter A3 Single Interface Requirements of the AMBA IHI0022E, except A3.3 Relationships between channels.
amba_axi4_transaction_structure.sv	• Properties defined in Chapter A3.4 Transaction structure of the AMBA IHI0022E. [1]
amba_axi4_transaction_attributes.sv	• Properties defined in Chapter A4 Transaction Attributes of the AMBA IHI0022E.
amba_axi4_atomic_accesses.sv	• Properties defined in Chapter A7 Atomic Accesses of the AMBA IHI0022E.
amba_axi4_definition_of_axi4_lite.sv	• Properties defined in Part B AMBA AXI4-Lite Interface Specification of the AMBA IHI0022E.

Table 2: AXI4 Formal IP packages description.

The modules shown in Table 3serves as assertion containers for the AMBA AXI4 protocol.

The modules (checkers).	**Description**
amba_axi_protocol_checker.sv[2]	Includes the five AXI4 channels (AW, W, R, AR, B) assertion modules, as well as the *low power* checker, the *relationship between channels* checker, and the *exclusive access from the perspective of the source* checker.
amba_axi4_write_address_channel.sv	Assertion container for AMBA AXI4 AW channel only.
amba_axi4_write_data_channel.sv	Assertion container for AMBA AXI4 W channel only.
amba_axi4_write_response_channel.sv	Assertion container for AMBA AXI4 B channel only.
amba_axi4_read_data_channel.sv	Assertion container for AMBA AXI4 R channel only.

Table 3: AMBA AXI4 Modules description.

[1]I can probably merge this package into amba_axi4_single_interface_requirements.sv

[2]This module serves as an example of how to use the modular sources to build the complete Formal Verification IP for AXI4. Modules (assertion containers) can be added/removed easily, depending on the use cases.

7 Using the AXI SVA Formal IPs

7.1 Bind to a source DUT

Example of how to bind the AXI SVA IP to a source DUT.

7.2 Bind to a destination DUT

Example of how to bind the AXI SVA IP to a destination DUT.

7.3 Monitor Example

Example of how to bind the AXI SVA IP to a source/destination DUT.

7.4 Constraint Example

Example of how to use the AXI SVA IP to provide constrains to a DUT.

Part IV
Verification Plan

8 General Methodology

Each design and verification goal is different, therefore there does not exsist a methodology that can invariably generate good results for all of them. A general approach for both RTL bring-up of a new IP, and verification of an existing IP is shown in this section. Also, it is recommended to have the following points in mind:

- The YosysHQ AXI4 protocol checker follows an assume-guarantee methodology, that means, all properties are guaranteed under the assumptions provided by the same assertion checker. If neighbour blocks does not behave as how expected assumptions dictates (i.e., the driver has a bug) then the property cannot be guaranteed in the field. To avoid this scenario, verify both driving and comsumer agents with the YosysHQ protocol checker before integration.
- Starting with a small number of MAX_WR_BURST and MAX_RD_BURST will execute the verification faster. It is important to increase the number to a some more realistic value, depending on the number of transactions that the IP (DUT) can handle when checking potential deadlocks or looking for formal sign-off.
- The transaction structure properties also contains X-prop checks. They are important to detect design errors or even security vulenarabilities. A special solver/tool configuration is needed to verify them.
- Put special attention to the atomic accesses checker specially if the IP (DUT) is a crossbar/interconnect, failure to meet all the exclusive responses properties may lead to an unresponsive system (even the AMBA certified verification IP has a bug that can block the system under certain atomic operation).
- Unreached covers does not necessarily mean that something is wrong, there are cases where the performance of the system makes certain scenarios to never occur, or simply one configuration is wrong, but a considerable amount of uncovered properties definetely is not a good sign.
- Potential deadlocks for interchannel response checks, amba recommended properties and exclusive access limitations are the most difficult properties to prove. It is good to sepparate easy invariants from them if possibe (i.e., running all interface checks first, then disable them and execute only transactional proofs). For continuos integration systems, properties can be cached (resuls reused), for example. Contact the YosysHQ for help if needed.

8.1 RTL Bring-up

The YosysHQ AXI4 Formal IP is built in a modular way, that is, each transaction channel of the AXI protocol can be used in a sepparate way to verify a channel if the design is built hierarchically. In such a case, the workflow can be executed as follows:

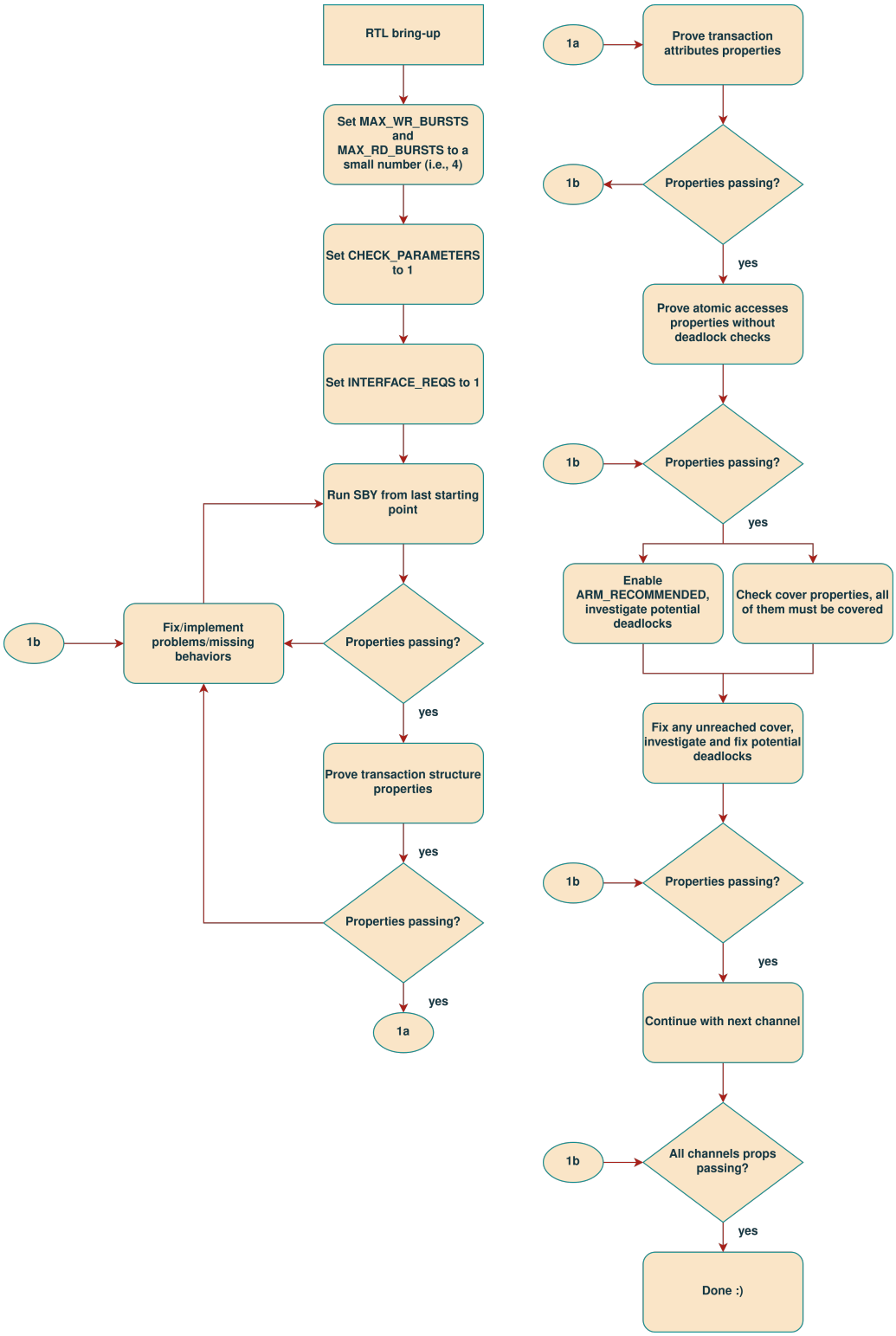

Figure 6: Application methodology for RTL Bring-up.

8.2 Protocol certification (verification) of existing IP

The methodology for verification of an existing IP is similar to the one for the RTL bring-up, except that the goal is different. For RTL bring-up, designes can focus on protocol compliance using the YosysHQ SVA IP as testbench and investigate any existing problem easily, but for verification the goal is more in the sense of certifying that the design complies with the AMBA AXI4 specification, and uncover any possible bugs due RTL errors or misunderstanding of the rules in the AMBA document. As aforementioned, there are certain properties that are more difficult to converge, so abstraction may be needed, for example, if exclusive access checker does not converge, try to use AUTO_SELECT = YES with uninterpreted functions, or with rigid variables if the problem persist.

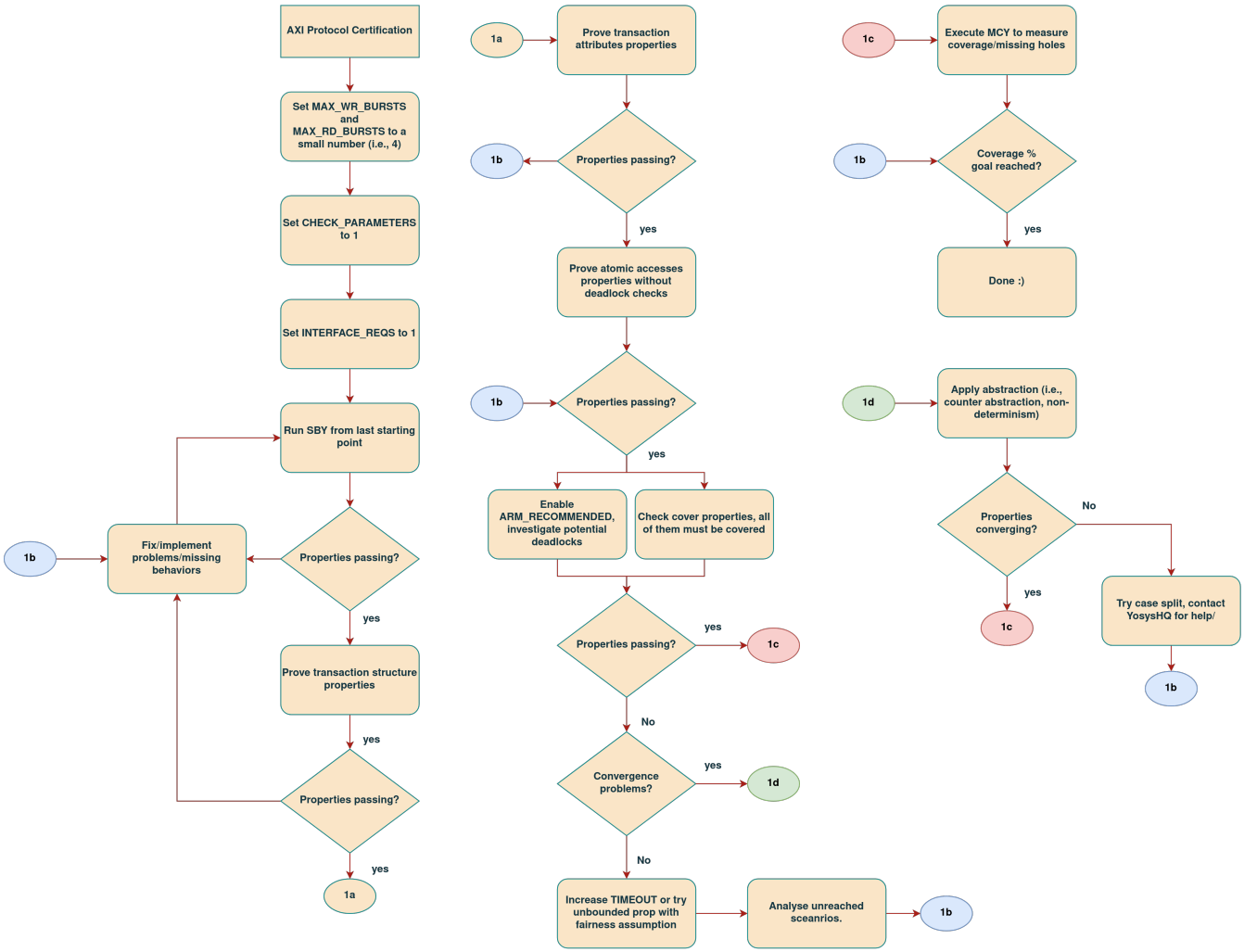

Figure 7: Application methodology for Verification of existinig IP.

9 AMBA AXI4 Property Matrix

9.1 Write Address Channel (AW) Properties

Property	AMBA AXI4 Rule	Reference	Validated
AW_STABLE_AWID	AXI4_ERRM_AWID_STABLE	11.1.1 on page 22	Yes
TBD	AXI4_ERRM_AWID_X		—
AW_AWADDR_BOUNDARY_4KB	AXI4_ERRM_AWADDR_BOUNDARY	11.1.2 on page 24	Yes
AW_ADDRESS_ALIGNMENT	AXI4_ERRM_AWADDR_WRAP_ALIGN	11.1.2 on page 25	Yes
AW_STABLE_AWADDR	AXI4_ERRM_AWADDR_STABLE	11.1.2 on page 23	Yes
TBD	AXI4_ERRM_AWADDR_X		—
AW_VALID_WRAP_BURST	AXI4_ERRM_AWLEN_WRAP	11.1.3 on page 28	Yes
AW_STABLE_AWLEN	AXI4_ERRM_AWLEN_STABLE	11.1.3 on page 26	Yes
TBD	AXI4_ERRM_AWLEN_X		—
AW_STABLE_AWSIZE	AXI4_ERRM_AWSIZE_STABLE	11.1.4 on page 30	Yes
AW_CORRECT_BURST_SIZE	AXI4_ERRM_AWSIZE	11.1.4 on page 31	Yes
TBD	AXI4_ERRM_AWSIZE_X		—
AW_BURST_TYPES	AXI4_ERRM_AWBURST	11.1.5 on page 33	Yes
AW_STABLE_AWBURST	AXI4_ERRM_AWBURST_STABLE	11.1.5 on page 32	Yes
TBD	AXI4_ERRM_AWBURST_X		—
AW_STABLE_AWLOCK	AXI4_ERRM_AWLOCK_STABLE	11.1.6 on page 34	Yes
TBD	AXI4_ERRM_AWLOCK_X		—
AW_MEMORY_TYPE_ENCODING	AXI4_ERRM_AWCACHE	11.1.7 on page 37	Yes
AW_STABLE_AWCACHE	AXI4_ERRM_AWCACHE_STABLE	11.1.7 on page 36	Yes
TBD	AXI4_ERRM_AWCACHE_X		—
AW_STABLE_AWPROT	AXI4_ERRM_AWPROT_STABLE	11.1.8 on page 38	Yes
TBD	AXI4_ERRM_AWPROT_X		—
AW_EXIT_RESET	AXI4_ERRM_AWVALID_RESET	11.1.12 on page 42	Yes
AW_AWVALID_until_AWREADY	AXI4_ERRM_AWVALID_STABLE	11.1.12 on page 43	Yes
TBD	AXI4_ERRM_AWVALID_X		—
TBD	AXI4_ERRS_AWREADY_X		—
AW_READY_MAXWAIT	AXI4_RECS_AWREADY_MAX_WAIT	11.1.15 on page 48	Yes
AW_STABLE_AWUSER	AXI4_ERRM_AWUSER_STABLE	11.1.11 on page 41	Yes

TBD	AXI4_ERRM_AWUSER_X		—
AW_STABLE_AWQOS	AXI4_ERRM_AWQOS_STABLE	11.1.9 on page 39	Yes
TBD	AXI4_ERRM_AWQOS_X		—
AW_STABLE_AWREGION	AXI4_ERRM_AWREGION_STABLE	11.1.10 on page 40	Yes
TDB	AXI4_ERRM_AWREGION_X		—
AW_VALID_LEN_FIXED	AXI4_ERRM_AWLEN_FIXED	11.1.3 on page 27	Yes
AW_VALID_BURST_LEN_EXCLUSIVE	AXI4_ERRM_AWLEN_LOCK	11.1.3 on page 29	Yes
TBD	AXI4_ERRM_AWUSER_TIEOFF		—
TBD	AXI4_ERRM_AWID_TIEOFF		—
Not in DUI0534B			
AW_unsupported_axi4l	—	11.1.16 on page 49	Yes
AW_AXI4LITE_DATAWIDTH	— (Modify name in the src)	11.1.16 on page 50	Yes
AW_UNSUPPORTED_EXCLUSIVE	—	11.1.6 on page 35	Yes
Extra Properties			
AWVALID_before_AWREADY	—	11.1.14 on page 45	Yes
AWREADY_before_AWVALID	—	11.1.14 on page 46	Yes
AWVALID_with_AWREADY	—	11.1.14 on page 47	Yes
AW_B2B	—		Yes
AW_WAIT	—		Yes
AW_NO_WAIT	—		Yes
AW_len_transfers	—		Yes

10 AMBA AXI4 Specification Description

Use the AXI SVA IP to demonstrate various transactions, waveform and recipes of verification scenarios.

11 Source Functional Rules

The sections below describes the rules that the AXI SVA IP implements for AMBA AXI4 protocol compliance using the SymbiYosys FPV tool.

11.1 AMBA AXI4 Write Address Channel

The file amba_axi4_write_address_channel.sv provides the functional rules for the AXI4 AW channel. These rules are described below.

Note: The order in which the ports appear in this document is taken from the ARM specification IHI 0022E Section A2 Signal Descriptions.

11.1.1 AWID

Handshake Process

Rule: Once the master has asserted AWVALID, data and control information from master must remain stable [AWID] until AWREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **AW_STABLE_AWID** checks this requirement.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWREADY and AWID are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, AWREADY is LOW constantly, and AWID can change its current value.

Listing 1: `amba_axi4_single_interface_requirements.sv`

```
52    property stable_before_handshake(valid, ready, control);
53       valid && !ready |-> ##1 $stable(control);
54    endproperty // stable_before_handshake
```

Listing 2: `amba_axi4_write_address_channel.sv`

```
156             else $error("Violation: For AW in AXI4-Lite, only signals described in B1.1 are",
157                         "required or supported (B1.1 Definition of AXI4-Lite, pB1-126).");
158          // Guard correct AXI4-Lite DATA_WIDTH since the parameter is used here.
159          if(cfg.CHECK_PARAMETERS == 1) begin: check_dataw
```

Figure 8: Property AW_STABLE_AWID.

11.1.2 AWADDR

Handshake Process

Rule: Once the master has asserted AWVALID, data and control information from master must remain stable [AWADDR] until AWREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **AW_STABLE_AWADDR** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWREADY and AWADDR are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, AWREADY is LOW constantly, and AWADDR can change its current value.

Listing 3: `amba_axi4_single_interface_requirements.sv`

```
52    property stable_before_handshake(valid, ready, control);
53       valid && !ready |-> ##1 $stable(control);
54    endproperty // stable_before_handshake
```

Listing 4: `amba_axi4_write_address_channel.sv`

```
175                           "from master must remain stable [AWID] until AWREADY is asserted",
176                           "(A3.2.1 Handshake process, pA3-39, Figure A3-2).");
177             if(cfg.ENABLE_XPROP) begin
```

Figure 9: Property AW_STABLE_AWADDR.

Address structure

Rule: AXI has the following rules governing the use of bursts: [...] a burst must not cross a 4KB address boundary (A3.4.1 Address structure, pA3-46).

Property: In accordance with this rule, the property **AW_AWADDR_BOUNDARY_4KB** checks this behavior (see Section 14.1 for more information).

Note: This rule applies only for INCR burst type. The FIXED and WRAP burst types have limitations for the burst length that uses address that cannot cross the given boundary.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWBURST and **aw_4KB_boundary** (that is calculated by **end_addr** using AWSIZE, AWADDR and AWLEN) are the signals of interest. The requirement will fail if, from a given BURST_TYPE= INCR, the final address calculation is not maintained within bits 11:0 of AWADDR. For debugging purposes, signal **end_addr** holds this calculation.

Listing 5: `amba_axi4_transaction_structure.sv`

```
63    property burst_cache_line_boundary(valid, burst, cond);
64       (valid && burst == amba_axi4_protocol_checker_pkg::INCR |-> cond);
65    endproperty // burst_cache_line_boundary
```

Listing 6: `amba_axi4_write_address_channel.sv`

```
186               else $error("Violation: Once the master has asserted AWVALID, data and control
                  ↪  information",
187                           "from master must remain stable [AWID] until AWREADY is asserted",
188                           "(A3.2.1 Handshake process, pA3-39, Figure A3-2).");
189             if(cfg.ENABLE_XPROP) begin
```

Figure 10: Property AW_AWADDR_BOUNDARY_4KB.

Address structure

Rule: The start address must be aligned to the size of each transfer (A3.4.1 Address structure, pA3-48).

Property: In accordance with this rule, the property **AW_ADDRESS_ALIGNMENT** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWBURST, AWADDR and AWSIZE are the signals of interest. The requirement will fail if address is not aligned accordingly to the equations presented in Burst address, pA3-48. For debugging purposes, the let function **addr_align_check** shows a simplification of these equations. If the address is aligned, **addr_align_check** will result in a logic value of HIGH, otherwise it will be LOW.

Listing 7: `amba_axi4_single_interface_requirements.sv`

```
149     *                    Section A3.4.1: Address structure                    *
150     *                  ><><><><><><><><><><><><><><><><><><><><><><><><><><><  */
151
152    /* ,          ,                                                           *
153     * |\\\\ ////|  "The start address must be aligned to the size of  *
154     * | \\\V/// |    each transfer".                                          *
```

Listing 8: `amba_axi4_write_address_channel.sv`

```
190                 cp_AW_AWID_X: assume property(disable iff(!ARESETn) valid_information(AWVALID, AWID))
191                   else $error("Violation : The source can assert the [AW]VALID signal only when it",
192                               "drives valid address and control information (A3.2.2 Channel signaling",
```

Figure 11: Property AW_ADDRESS_ALIGNMENT.

11.1.3 AWLEN

Handshake Process

Rule: Once the master has asserted AWVALID, data and control information from master must remain stable [AWLEN] until AWREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **AW_STABLE_AWLEN** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWREADY and AWLEN are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, AWREADY is LOW constantly, and AWLEN can change its current value.

Listing 9: `amba_axi4_single_interface_requirements.sv`

```
52    property stable_before_handshake(valid, ready, control);
53       valid && !ready |-> ##1 $stable(control);
54    endproperty // stable_before_handshake
```

Listing 10: `amba_axi4_write_address_channel.sv`

```
212         end
213      end
214      else if(cfg.VERIFY_AGENT_TYPE inside {DESTINATION, CONSTRAINT}) begin
```

Figure 12: Property AW_STABLE_AWLEN.

Address structure

Rule: AXI4 extends burst length support for the INCR burst type to 1 to 256 transfers. Support for all other burst types in AXI4 remains at 1 to 16 transfers (A3.4.1 Address structure, pA3-46).

Property: In accordance with this rule, the property **AW_VALID_LEN_FIXED** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWBURST, and AW_FIXED_len are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, AWBURST is set to FIXED and bits 7:4 of AWLEN are modified (because AWLEN[3:0] are sufficient to notify bursts of <= 16 beats, accordingly to A3-46).

Listing 11: `amba_axi4_transaction_structure.sv`

```
34    property supported_burst_transfer(valid, burst, burst_type, len_val);
35       (valid && burst == burst_type |-> len_val);
36    endproperty // supported_burst_transfer
```

Listing 12: `amba_axi4_write_address_channel.sv`

```
217                        "from master must remain stable [AWADDR] until AWREADY is asserted (A3.2.1
                           ↪  Handshake process, pA3-39, Figure A3-2).");
218           if(cfg.ENABLE_XPROP) begin
```

Figure 13: Property AW_VALID_LEN_FIXED.

Address structure

Rule: AXI has the following rules governing the use of bursts: for wrapping bursts, the burst length must be 2, 4, 8, or 16 (A3.4.1 Address structure, pA3-46).

Property: In accordance with this rule, the property **AW_VALID_WRAP_BURST** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWBURST and AWLEN are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, AWBURST is set to WRAP and AWLEN is not inside 1, 3, 7 or 15.

Listing 13: `amba_axi4_transaction_structure.sv`

```
48    property valid_wrap_burst_length(valid, burst, len);
49       (valid && burst == amba_axi4_protocol_checker_pkg::WRAP |->
50        len inside {8'd1, 8'd3, 8'd7, 8'd15});
51    endproperty // valid_wrap_burst_length
```

Listing 14: `amba_axi4_write_address_channel.sv`

```
215           cp_AW_STABLE_AWADDR: assume property(disable iff(!ARESETn) stable_before_handshake(AWVALID,
              ↪  AWREADY, AWADDR))
216             else $error("Violation: Once the master has asserted AWVALID, data and control information",
```

Figure 14: Property AW_VALID_WRAP_BURST.

Exclusive access restrictions

Rule: In AXI4, the burst length for an exclusive access must not exceed 16 transfers (A7.2.4 Exclusive access restrictions, pA7-97).

Property: In accordance with this rule, the property **AW_VALID_BURST_LEN_EXCLUSIVE** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWLOCK and AWLEN are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, AWLOCK is set to EXCLUSIVE and bits 7:4 of AWLEN are modified (because AWLEN[3:0] are sufficient to notify bursts of <= 16 beats, accordingly to A3-46).

Listing 15: `amba_axi4_write_address_channel.sv`

```
68      * |         |===|        |   must not exceed 16 transfers".                    *
69      * |         |A  |        |   Ref: A7.2.4 Exclusive access restrictions, pA7-97 *
70      * |         | X |        |                                                      *
71      *  \ |  I| /                                                  *
```

Listing 16: `amba_axi4_write_address_channel.sv`

```
219             cp_AW_AWADDR_X: assume property(disable iff(!ARESETn) valid_information(AWVALID, AWADDR))
220               else $error("Violation : The source can assert the [AW]VALID signal only when it",
221                           "drives valid address and control information (A3.2.2 Channel signaling",
222                           "requirements pA3-40).");
223         end
```

Figure 15: Property AW_VALID_BURST_LEN_EXCLUSIVE.

11.1.4 AWSIZE

Handshake Process

Rule: Once the master has asserted AWVALID, data and control information from master must remain stable [AWSIZE] until AWREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **AW_STABLE_AWSIZE** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWREADY and AWSIZE are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, AWREADY is LOW constantly, and AWSIZE can change its current value.

Listing 17: `amba_axi4_single_interface_requirements.sv`

```
52    property stable_before_handshake(valid, ready, control);
53       valid && !ready |-> ##1 $stable(control);
54    endproperty // stable_before_handshake
```

Listing 18: `amba_axi4_write_address_channel.sv`

```
248      end // if (cfg.PROTOCOL_TYPE == AXI4FULL)
249   endgenerate
250
```

Figure 16: Property AW_STABLE_AWSIZE.

Address structure

Rule: The size of any transfer must not exceed the data bus width of either agent in the transaction (A3.4.1 Address structure, Burst size, pA3-47).

Property: In accordance with this rule, the property **AW_CORRECT_BURST_SIZE** checks this requirement.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID and AWSIZE are the signals of interest, also with the parameter DATA_WIDTH. The requirement will fail if there exists an state where AWVALID is HIGH and AWSIZE exceeds log2(DATA_WIDTH).

Listing 19: `amba_axi4_transaction_structure.sv`

```
let calc_max_size(w) = $clog2(w);
property burst_size_within_width_boundary(valid, size, strb);
   (valid |-> size <= calc_max_size(strb));
endproperty // burst_size_within_width_boundary
```

Listing 20: `amba_axi4_write_address_channel.sv`

```
/*                 ><><><><><><><><><><><><><><><><><><><><                *
 *                                  AWLEN                                  *
 *                 ><><><><><><><><><><><><><><><><><><><><                 */
```

Figure 17: Property AW_CORRECT_BURST_SIZE.

11.1.5 AWBURST

Handshake Process

Rule: Once the master has asserted AWVALID, data and control information from master must remain stable [AWBURST] until AWREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **AW_STABLE_AWBURST** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWREADY and AWBURST are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, AWREADY is LOW constantly, and AWBURST can change its current value.

Listing 21: `amba_axi4_single_interface_requirements.sv`

```
52    property stable_before_handshake(valid, ready, control);
53       valid && !ready |-> ##1 $stable(control);
54    endproperty // stable_before_handshake
```

Listing 22: `amba_axi4_write_address_channel.sv`

```
272           end
273           ap_AW_VALID_WRAP_BURST: assert property(disable iff(!ARESETn) valid_wrap_burst_length(AWVALID,
              ↪  AWBURST, AWLEN))
274             else $error("Violation: For wrapping bursts, the burst length must be 2, 4, 8 or 16 (A3.4.1
                ↪  Address structure, pA3-46).");
```

Figure 18: Property AW_STABLE_AWBURST.

Address structure

Rule: The AXI protocol defines three burst types, FIXED, INCR and WRAP. RESERVED is undefined and therefore not a valid burst (A3.4.1 Address structure, pA3-48, Table A3-3 Burst type encoding).

Property: In accordance with this rule, the property **AW_BURST_TYPES** checks this requirement.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID and AWBURST are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, and AWBURST is set to RESERVED (0b11 according to Table A3-3 Burst type encoding).

Listing 23: `amba_axi4_write_address_channel.sv`

```
275         ap_AW_VALID_LEN_FIXED: assert property(disable iff(!ARESETn) supported_burst_transfer(AWVALID,
            ↪  AWBURST, FIXED, AW_FIXED_len))
276           else $error("Violation: Support for FIXED burst type in AXI4 remains at 1 to 16 transfers
              ↪  (A3.4.1 Address structure, pA3-46).");
277         if(cfg.EXCLUSIVE_ACCESS) begin
```

Figure 19: Property AW_BURST_TYPES.

11.1.6 AWLOCK

Handshake Process

Rule: Once the master has asserted AWVALID, data and control information from master must remain stable [AWLOCK] until AWREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **AW_STABLE_AWLOCK** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWREADY and AWLOCK are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, AWREADY is LOW constantly, and AWLOCK can change its current value.

Listing 24: `amba_axi4_single_interface_requirements.sv`

```
52    property stable_before_handshake(valid, ready, control);
53       valid && !ready |-> ##1 $stable(control);
54    endproperty // stable_before_handshake
```

Listing 25: `amba_axi4_write_address_channel.sv`

```
296             cp_AW_AWLEN_X: assume property(disable iff(!ARESETn) valid_information(AWVALID, AWLEN))
297               else $error("Violation : The source can assert the [AW]VALID signal only when it",
298                           "drives valid address and control information (A3.2.2 Channel signaling",
```

Figure 20: Property AW_STABLE_AWLOCK.

Definition of AXI4-Lite

Rule: The EXOKAY response is not supported on the read data and write response channels (B1.1.1 Signal List, pB1-126).

Property: In accordance with this rule, the property **AW_UNSUPPORTED_EXCLUSIVE** checks this requirement.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID and AWLOCK are the signals of interest. The requirement will fail if the Formal IP is configured as AXI4LITE and there exists an state where AWVALID is HIGH, and AWLOCK is set to EXCLUSIVE (0b1 according to Table A7-2 AXI4 atomic access encoding).

Listing 26: `amba_axi4_definition_of_axi4_lite.sv`

```
48  property unsupported_exclusive_access(valid, lock, value);
49     valid |-> lock != value;
50  endproperty // unsupported_exclusive_access
```

Listing 27: `amba_axi4_write_address_channel.sv`

```
308                         "(A7.2.4 Exclusive access restrictions, pA7-97).");
309        end
310     end // if (cfg.VERIFY_AGENT_TYPE inside {DESTINATION, CONSTRAINT})
```

Figure 21: Property AW_UNSUPPORTED_EXCLUSIVE.

11.1.7 AWCACHE

Handshake Process

Rule: Once the master has asserted AWVALID, data and control information from master must remain stable [AWCACHE] until AWREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **AW_STABLE_AWCACHE** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWREADY and AWCACHE are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, AWREADY is LOW constantly, and AWCACHE can change its current value.

Listing 28: `amba_axi4_single_interface_requirements.sv`

```
52    property stable_before_handshake(valid, ready, control);
53       valid && !ready |-> ##1 $stable(control);
54    endproperty // stable_before_handshake
```

Listing 29: `amba_axi4_write_address_channel.sv`

```
326                              "requirements pA3-40).");
327            end
328            ap_AW_CORRECT_BURST_SIZE: assert property(disable iff(!ARESETn)
               ↪  burst_size_within_width_boundary(AWVALID, AWSIZE, STRB_WIDTH))
```

Figure 22: Property AW_STABLE_AWCACHE.

Memory Type Encoding

Rule: For memory types, all values not shown in Table A4-5 are reserved (A4.4 Memory types, pA4-67, Table A4-5 Memory type encoding).

Property: In accordance with this rule, the property **AW_MEMORY_TYPE_ENCODING** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, and AWCACHE are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, and AWCACHE is inside 4'h4, 4'h5, 4'h8, 4'h9, 4'hC, 4'hD (accordingly to Table A4-5 Memory type encoding).

Listing 30: `amba_axi4_transaction_attributes.sv`

```
34    let AxCACHE_invalid_type_encoding(AxCACHE) =
35        AxCACHE inside {4'h4, 4'h5, 4'h8, 4'h9, 4'hC, 4'hD};
36    property memory_type_encoding(valid, cache);
37       valid |-> not AxCACHE_invalid_type_encoding(cache);
38    endproperty // memory_type_encoding
```

Listing 31: `amba_axi4_write_address_channel.sv`

```
329             else $error("Violation: The size of any transfer must not exceed the data bus width of
                ↪  either agent in the transaction.",
330                         "(A3.4.1 Address structure, Burst size, pA3-47).");
331         end
```

Figure 23: Property AW_MEMORY_TYPE_ENCODING.

11.1.8 AWPROT

Handshake Process

Rule: Once the master has asserted AWVALID, data and control information from master must remain stable [AWPROT] until AWREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **AW_STABLE_AWPROT** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWREADY and AWPROT are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, AWREADY is LOW constantly, and AWPROT can change its current value.

Listing 32: amba_axi4_single_interface_requirements.sv

```
property stable_before_handshake(valid, ready, control);
   valid && !ready |-> ##1 $stable(control);
endproperty // stable_before_handshake
```

Listing 33: amba_axi4_write_address_channel.sv

```
/*            ><><><><><><><><><><><><><><><><><><><><             *
 *                             AWBURST                             *
 *            ><><><><><><><><><><><><><><><><><><><><             */
```

Figure 24: Property AW_STABLE_AWPROT.

11.1.9 AWQOS

Handshake Process

Rule: Once the master has asserted AWVALID, data and control information from master must remain stable [AWQOS] until AWREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **AW_STABLE_AWQOS** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWREADY and AWQOS are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, AWREADY is LOW constantly, and AWQOS can change its current value.

Listing 34: `amba_axi4_single_interface_requirements.sv`

```
52    property stable_before_handshake(valid, ready, control);
53       valid && !ready |-> ##1 $stable(control);
54    endproperty // stable_before_handshake
```

Listing 35: `amba_axi4_write_address_channel.sv`

```
366                           "and therefore not a valid burst (A3.4.1 Address structure, pA3-48, Table A3-3
                              ↪  Burst type encoding).");
367           end
368           else if(cfg.VERIFY_AGENT_TYPE inside {DESTINATION, CONSTRAINT}) begin
```

Figure 25: Property AW_STABLE_AWQOS.

11.1.10 AWREGION

Handshake Process

Rule: Once the master has asserted AWVALID, data and control information from master must remain stable [AWREGION] until AWREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **AW_STABLE_AWREGION** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWREADY and AWREGION are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, AWREADY is LOW constantly, and AWREGION can change its current value.

Listing 36: amba_axi4_single_interface_requirements.sv

```
52    property stable_before_handshake(valid, ready, control);
53       valid && !ready |-> ##1 $stable(control);
54    endproperty // stable_before_handshake
```

Listing 37: amba_axi4_write_address_channel.sv

```
384
385   /*            ><><><><><><><><><><><><><><><><><><><><><><><><><><><><            *
386    *                                AWLOCK                                         *
```

Figure 26: Property AW_STABLE_AWREGION.

11.1.11 AWUSER

Handshake Process

Rule: Once the master has asserted AWVALID, data and control information from master must remain stable [AWUSER] until AWREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **AW_STABLE_AWUSER** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID, AWREADY and AWUSER are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, AWREADY is LOW constantly, and AWUSER can change its current value.

Listing 38: amba_axi4_single_interface_requirements.sv

```
52    property stable_before_handshake(valid, ready, control);
53       valid && !ready |-> ##1 $stable(control);
54    endproperty // stable_before_handshake
```

Listing 39: amba_axi4_write_address_channel.sv

```
402             cp_AW_STABLE_AWLOCK: assume property(disable iff(!ARESETn) stable_before_handshake(AWVALID,
                ↪  AWREADY, AWLOCK))
403               else $error("Violation: Once the master has asserted AWVALID, data and control
                  ↪  information",
404                           "from master must remain stable [AWLOCK] until AWREADY is asserted (A3.2.1
                              ↪  Handshake process, pA3-39, Figure A3-2).");
```

Figure 27: Property AW_STABLE_AWUSER.

11.1.12 AWVALID

Clock and reset

Rule: AWVALID must be low for the first clock edge after ARESETn goes high (A3.2.1 Reset, pA3-38, Figure A3-1).

Property: In accordance with this rule, the property **AW_EXIT_RESET** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description ARESETn and AWVALID are the signals of interest. The requirement will fail if there exists an state where ARESETn is LOW, or ARESETn changes to HIGH in the current ACLK, and AWVALID is HIGH.

Listing 40: `amba_axi4_single_interface_requirements.sv`

```
34    property exit_from_reset(aresetn, valid);
35       (!aresetn || $rose(aresetn)) |-> !valid;
36    endproperty // exit_from_reset
```

Listing 41: `amba_axi4_write_address_channel.sv`

```
420             cp_AW_UNSUPPORTED_EXCLUSIVE: assume property(disable iff(!ARESETn)
                ↪  unsupported_exclusive_access(AWVALID, AWLOCK, EXCLUSIVE))
421               else $error("Violation: Exclusive read accesses are not supported in AXI4 Lite",
422                           "(Definition of AXI4-Lite, pB1-126).");
```

Figure 28: Property AW_EXIT_RESET.

Handshake Process

Rule: Once AWVALID is asserted it must remain asserted until the handshake occurs (A3.2.1 Handshake process, pA3-39).

Property: In accordance with this rule, the property **AW_AWVALID_until_AWREADY** checks this behavior.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Assume
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID and AWREADY are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, AWREADY is LOW constantly, and AWVALID changes to LOW in the next ACLK cycle (data drop due unacknowledged transmission).

Listing 42: `amba_axi4_single_interface_requirements.sv`

```
66    property valid_before_handshake(valid, ready);
67       valid && !ready |-> ##1 valid;
68    endproperty // valid_before_handshake
```

Listing 43: `amba_axi4_write_address_channel.sv`

```
437                ap_AW_AWCACHE_X: assert property(disable iff(!ARESETn) valid_information(AWVALID,
                   ↪  AWCACHE))
438                  else $error("Violation : The source can assert the [AW]VALID signal only when it",
439                              "drives valid address and control information (A3.2.2 Channel signaling",
```

Figure 29: Property AW_AWVALID_until_AWREADY.

11.1.13 AWREADY

No properties defined.

11.1.14 AW Cover Properties

AW VALID before READY

Scenario: Handshake process pA3-39, Figure A3-2 VALID before READY handshake capability.

Property: The property **AWVALID_before_AWREADY** is defined to witness this scenario.

Property Description AWVALID and AWREADY are the signals of interest. The requirement will be unreachable if no state where AWVALID is asserted before AWREADY can be witnessed (this is a performance-type property, if it is unreachable, it does not mean that the design is not working properly).

Listing 44: `amba_axi4_single_interface_requirements.sv`

```
80    property valid_before_ready(valid, ready);
81       valid && !ready;
82    endproperty // valid_before_ready
```

Listing 45: `amba_axi4_write_address_channel.sv`

```
473                          "the transaction is Cacheable (A7.2.4 Exclusive access restrictions,
                             ↪  pA7-97).");
474           end
```

Figure 30: Property AWVALID_before_AWREADY.

AW READY before VALID

Scenario: Handshake process pA3-39, Figure A3-3 READY before VALID handshake capability.

Property: The property **AWREADY_before_AWVALID** is defined to witness this scenario.

Property Description AWVALID and AWREADY are the signals of interest. The requirement will be unreachable if no state where AWREADY is asserted before AWVALID can be witnessed (this is a performance-type property, if it is unreachable, it does not mean that the design is not working properly).

Listing 46: `amba_axi4_single_interface_requirements.sv`

```
94    property ready_before_valid(valid, ready);
95       ready && !valid;
96    endproperty // ready_before_valid
```

Listing 47: `amba_axi4_write_address_channel.sv`

```
475         end
476      end // if (cfg.PROTOCOL_TYPE == AXI4FULL)
```

Figure 31: Property AWREADY_before_AWVALID.

AW VALID with READY

Scenario: Handshake process pA3-39, Figure A3-4 VALID with READY handshake capability.

Property: The property **AWVALID_with_AWREADY** is defined to witness this scenario.

Property Description AWVALID and AWREADY are the signals of interest. The requirement will be unreachable if no state where AWREADY and AVALID are both asserted is present in the design (If all guarantee invariants are passing but this cover is unreachable, this means that there are serious problems with the environment and results are unsound. To help debugging this kind of scenarios, user can read YosysHQ AppNote 120 - Methodology to Debug Vacuous Properties.).

Listing 48: `amba_axi4_single_interface_requirements.sv`

```
108     property valid_with_ready(valid, ready);
109        valid && ready;
110     endproperty // valid_with_ready
```

Listing 49: `amba_axi4_write_address_channel.sv`

```
477     endgenerate
478
```

Figure 32: Property AWVALID_with_AWREADY.

11.1.15 AW Optional Properties

Potential Deadlock

Rule: AWREADY should be asserted within MAXWAIT cycles of AWVALID being asserted (AMBA recommended).

Property: In accordance with this rule, the property **AW_READY_MAXWAIT** checks this requirement.

Application Table

Agent	Verification directive
Source	Assume
Destination	Guarantee
Monitor	Guarantee
Constraint	Assume

Property Description AWVALID and AWREADY are the signals of interest. The requirement will fail if there exists an state where AWVALID is HIGH, and AWREADY is LOW for more than MAXWAIT cycles.

Listing 50: `amba_axi4_single_interface_requirements.sv`

```
123     property handshake_max_wait(valid, ready, timeout);
124        valid & !ready |-> ##[1:timeout] ready;
125     endproperty // handshake_max_wait
```

Listing 51: `amba_axi4_write_address_channel.sv`

```
454           else if(cfg.VERIFY_AGENT_TYPE inside {DESTINATION, CONSTRAINT}) begin
455              cp_AW_STABLE_AWCACHE: assume property(disable iff(!ARESETn) stable_before_handshake(AWVALID,
                 ↪  AWREADY, AWCACHE))
```

Figure 33: Property AW_READY_MAXWAIT.

11.1.16 AW Configuration Properties

Definition of AXI4-Lite

Rule: For AW in AXI4-Lite, only signals described in B1.1 are required or supported (B1.1 Definition of AXI4-Lite, pB1-126).

Property: In accordance with this rule, the property **AW_unsupported_axi4l** checks this requirement.

Note: Please note that a default and valid value will be set by this property to constraint certain ports, for the case of AGENT_TYPE=AXI4LITE. **It is safe to leave these ports unconnected when targeting AXI4-lite**. If the user sets a value manually during instantiation, this property could be overconstrained, affecting the test.
If overconstrainoccurs, it is suggested to cancel the run and fix any conflictive assumption. It is also suggested to use the template files in SVA-AXI-Properties/AXI/AXI4/templates as a guidance to bind/instantiate this IP.

Application Table

Agent	Verification directive
Source	Assume
Destination	Assume
Monitor	NA[3]
Constraint	Assume

Property Description AWLEN, AWSIZE, AWBURST, AWLOCK, AWCACHE, AWQOS, AWREGION, AWUSER and AWID (bundled by AW_unsupported_sig) are the signals of interest. This Verification IP supports both AXI4 and AXI4-Lite, and when configured as AXI4-Lite, ports defined in B1.1 Definition of AXI4-Lite, pB1-126 should have a valid value.

Listing 52: amba_axi4_definition_of_axi4_lite.sv

```
76    property axi4_lite_unsupported_sig(axi4_lite_sig_bundle);
77       axi4_lite_sig_bundle;
78    endproperty // axi4_lite_unsupported_sig
```

Listing 53: amba_axi4_write_address_channel.sv

```
138
139   always_comb begin
140      // Modifiable bit (renamed from <cacheable> (AXI3)), see A4.3.1
```

Figure 34: Property AW_unsupported_axi4l.

[3]TODO: Convert this into a guarantee for monitor mode.

Definition of AXI4-Lite

Rule: AXI4-Lite supports a data bus width of 32-bit or 64-bit (B.1 Definition of AXI4-Lite, pB1-126)..

Property: In accordance with this rule, the property **AW_AXI4LITE_DATAWIDTH** checks this requirement.

Note: If this property fails, it is suggested to cancel the run and set a valid DATA_WIDTH parameter to avoid unsound results.

Application Table

Agent	Verification directive
Source	Guarantee
Destination	Guarantee
Monitor	Guarantee
Constraint	NA

Property Description DATA_WIDTH is the parameter of interest. if CHECK_AGENT_TYPE = AXI4LITE and DATA_WITH is not 32 or 64, this property will fail.

Listing 54: `amba_axi4_definition_of_axi4_lite.sv`

```
34    property axi4l_databus_width(data_width);
35       data_width inside {32, 64};
36    endproperty // axi4l_databus_width
```

Listing 55: `amba_axi4_write_address_channel.sv`

```
142       // Read-allocate bit
143       ra_bit = AWCACHE[2];
144       // Write-allocate bit
145       wa_bit = AWCACHE[3];
```

Figure 35: Property AW_AXI4LITE_DATAWIDTH.

11.2 AMBA AXI4 Write Data Channel

11.2.1 WID

No properties for AXI4-Lite, WIP.

11.2.2 WDATA

Handshake Process

Rule: Once the source has asserted WVALID, data and control information from source must remain stable [WDATA] until WREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **W_SRC_DST_STABLE_WDATA** checks this behavior.

11.2.3 WSTRB

Handshake Process

Rule: Once the source has asserted WVALID, data and control information from source must remain stable [WSTRB] until WREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **W_SRC_DST_STABLE_WSTRB** checks this behavior.

Data read and write structure

Rule: A master must ensure that the write strobes are HIGH only for byte lanes that contain valid data (A3.4.3 Data read and write structure, pA3-52).

Property: Not implemented yet.

Default Signal Values

Rule: A source is not required to use the write strobe signals WSTRB[3:0] if it always performs full data bus width write transactions. The default value for write strobes is all signals asserted (A10.3.4 Write transactions, Table A10-1).

Property: In accordance with this rule, the property **W_FULL_TRANSACTION_OPTIONAL_WSTRB** checks this behavior.

11.2.4 WLAST

No properties for AXI4-Lite, WIP.

11.2.5 WUSER

No properties for AXI4-Lite, WIP.

11.2.6 WVALID

Clock and reset

Rule: WVALID must be low for the first clock edge after ARESETn goes high (A3.2.1 Reset, pA3-38, Figure A3-1).

Property: In accordance with this rule, the property **W_SRC_DST_EXIT_RESET** checks this behavior.

Handshake Process

Rule: Once WVALID is asserted it must remain asserted until the handshake occurs (A3.2.1 Handshake process, pA3-39).

Property: In accordance with this rule, the property **W_SRC_DST_AWVALID_until_AWREADY** checks this behavior.

11.2.7 WREADY

No properties.

11.3 AMBA AXI4 Read Address Channel

11.3.1 ARID

No properties for AXI4-Lite, WIP.

11.3.2 ARADDR

Handshake Process

Rule: Once the master has asserted ARVALID, data and control information from master must remain stable [ARADDR] until ARREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **AR_SRC_DST_STABLE_ARADDR** checks this behavior.

11.3.3 ARLEN

No properties for AXI4-Lite, WIP.

11.3.4 ARSIZE

No properties for AXI4-Lite, WIP.

11.3.5 ARBURST

No properties for AXI4-Lite, WIP.

11.3.6 ARLOCK

Definition of AXI4-Lite

Rule: Exclusive read accesses are not supported in AXI4 Lite (Definition of AXI4-Lite, pB1-126)..

Property: In accordance with this rule, the property **ap_AR_UNSUPPORTED_EXCLUSIVE** checks this behavior.

11.3.7 ARCACHE

11.3.8 ARPROT

Handshake Process

Rule: Once the master has asserted ARVALID, data and control information from master must remain stable [ARPROT] until ARREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **ap_AR_SRC_DST_STABLE_ARPROT** checks this behavior.

11.3.9 ARQOS

No properties for AXI4-Lite, WIP.

11.3.10 ARREGION

No properties for AXI4-Lite, WIP.

11.3.11 ARUSER

No properties for AXI4-Lite, WIP.

11.3.12 ARVALID

Clock and reset

Rule: ARVALID must be low for the first clock edge after ARESETn goes high (A3.2.1 Reset, pA3-38, Figure A3-1).

Property: In accordance with this rule, the property **ap_AR_SRC_DST_EXIT_RESET** checks this behavior.

Handshake Process

Rule: Once ARVALID is asserted it must remain asserted until the handshake occurs (A3.2.1 Handshake process, pA3-39).

Property: In accordance with this rule, the property **ap_AR_SRC_DST_ARVALID_until_ARREADY** checks this behavior.

11.3.13 ARREADY

No direct properties.

12 Destination Functional Rules

12.1 AMBA AXI4 Write Response Channel

Some properties apply to the Source configuration.

12.1.1 BID

No properties for AXI4-Lite, just of in-order transactions, WIP.

12.1.2 BRESP

Definition of AXI4-Lite

Rule: The EXOKAY response is not supported on the read data and write response channels (B1.1.1 Signal List, pB1-126).

Property: In accordance with this rule, the property **cp_B_UNSUPPORTED_RESPONSE** checks this behavior.

Listing 56: `amba_axi4_write_address_channel.sv`

```
62    property unsupported_transfer_status(valid, response, value);
63       valid |-> response != value;
64    endproperty // unsupported_transfer_status
```

Listing 57: `amba_axi4_write_address_channel.sv`

```
70    // Configure unsupported AXI4-Lite signals
71    bit AW_unsupported_sig;
72    // "all transactions are of burst length 1".
```

Figure 36: Property cp_B_UNSUPPORTED_RESPONSE.

Handshake Process

Rule: Once the source has asserted BVALID, data and control information from source must remain stable [BRESP] until BREADY is asserted (A3.2.1 Handshake process, pA3-39, Figure A3-2).

Property: In accordance with this rule, the property **cp_B_DST_SRC_STABLE_BRESP** checks this behavior.

Listing 58: amba_axi4_write_address_channel.sv

```
52    property stable_before_handshake(valid, ready, control);
53       valid && !ready |-> ##1 $stable(control);
54    endproperty // stable_before_handshake
```

Listing 59: amba_axi4_write_address_channel.sv

```
90                          "from master must remain stable [BID] until BREADY is asserted",
91                          "(A3.2.1 Handshake process, pA3-39, Figure A3-2).");
92             if(cfg.ENABLE_XPROP) begin
```

Figure 37: Property ap_AW_SRC_DST_STABLE_AWSIZE.

12.1.3 BUSER

12.1.4 BVALID

12.1.5 BREADY

12.2 AMBA AXI4 Write Address Channel

13 Optional Properties

13.1 Knowledge Articles and Erratas

13.1.1 KA001346

In the KBA article KA001346 “How can a slave return a BRESP before knowing the AWADDR value for the transaction ?” the document specifies the following at the end of the page: “Note: This behavior has been changed in AXI4. AXI4 requires that both the AW and WLAST have been accepted before a BRESP can be returned”.

Property:

Appendix

14 Helper Logic

14.1 The Boundary of 4KB in AxADDR

Listing 60: `amba_axi4_write_address_channel.sv`

```
import amba_axi4_single_interface_requirements::*;
import amba_axi4_transaction_structure::*;
import amba_axi4_transaction_attributes::*;
import amba_axi4_atomic_accesses::*;

// Default clocking for all properties
default clocking axi4_aclk @(posedge ACLK); endclocking

/*                 ><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><<                *
 *                              Helper logic                                *
 *                 ><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><<                 */

```

Figure 38: Helper logic.