# SystemVerilog Assertion Based Verification of AMBA-AHB

Prince Gurha[1]
Dept. of Electronics
R.C.O.E.M, Nagpur
princegurha.ecb@gmail.com

R. R. Khandelwal[2]
Dept. of Electronics
R.C.O.E.M, Nagpur
richar.khandelwal@yahoo.com

***Abstract*—Assertion Based Verification (ABV) is one of the widely used verification technique to enhance the verification quality and reduce the debugging time of complex system-on-chip (SOC) designs in order to speedup the verification process. A verification environment to verify an AMBA-AHB (Advanced High Performance Bus) by using SystemVerilog Assertion (SVA) is presented in this paper as it can easily be turned ON or OFF at any instant during simulation as needed. First the AMBA-AHB is modeled using 3 masters and 4 slaves in verilog language. This design is then verified using SVA binding construct in ModelSim. Binding allows verification engineers to add assertions to design without touching the design files. The different properties of AMBA-AHB and its corner cases properties are verified using ModelSim and the total coverage report of the design is calculated. In this paper, we define the assertions in separate modules and use the BIND SystemVerilog feature to bind the assertion modules to the Verilog RTL modules. Here, we have clear separation between the RTL modules and the assertion modules.**

***Keywords- Verification, AMBA-AHB, SVA, Bind, Assertion.***

## I. INTRODUCTION

Verification is a very challenging task for the designer in the whole design and verification cycle, because bugs which are uncovered in the earlier stages of the design carry on in the next stages of the design and later it is too complex to diagnose it[1]. Verification is a task of verifying whether our implementation matches with the micro-architectural specifications or not. Any design is incomplete without proper verification of that design.

The time required for verifying the design is becoming monotonous as the complicacy of the chip design is increasing exponentially. Nowadays, about 70% of the design time is needed for developing the verification environment. Hence in this condition, it is important to reduce the verification time in order to speed up the whole development process. But, on the other hand, the complicacy of the design makes it difficult to cover all the corner cases in minimum time[1].

Therefore, to improve the design observability and to detect and interpret its faults, Assertion Based Verification (ABV) is introduced. Assertion Based Verification is one of the recommended verification techniques to enhance the verification quality and reduce the debugging time of complex system-on-chip designs[2]. ABV is a technique in which assertions are used to detect specific design behavior either through simulation, formal verification or emulation of these assertions[1]. In today's scenario, ABV plays an important role in the detection phase and has been well accepted among the design and verification team. Assertions helps to detect the failure and thus reduce the effort to establish the exact reason of the failure[3]. Using AHB with assertions implemented on the design can speed up the verification process. In verification process, debugging is categorized into three stages i.e. error detection, error diagnosis and error correction.

SystemVerilog Assertion (SVA) is a type of ABV. SystemVerilog Assertion is a formal specification and verification language. It is also an integral part of SystemVerilog. SVA is a declarative language which has enormous control over time[4]. It is used to describe design properties unambiguously and precisely. SVA provides several in-built functions to analyse certain design behavior and also provides temporal domain functional coverage[5]. SVA can't be written directly into HDLs (Hardware Description Languages) other than SystemVerilog, but tool support for the use of SVA with other HDLs is possible through binding directives and comment pragmas[4].

Deploying assertions has several advantages which can be summarized as below: SVA can easily be turned ON or OFF at any instant during simulation, as needed. SVA is not only used to debug pre-silicon violations, but is also extended to debug post-silicon violations. SVA provides the capability to write assertions that ranges from system level to RTL level. It helps to detect bugs not easily observed at primary outputs. Improve observability[2]. SVA supports multi-clock domain crossing logic. Uses only concurrent assertion. The Concurrent assertions describe behaviour that spans over time. In a concurrent assertion evaluation is performed only at the occurrence of a clock tick[6]. It is also used for the creation of complex properties[2].

The paper is presented as follow: Section II and III introduces the overall architecture of AMBA-AHB and its applications, Section IV gives detail about SVA building block, its binding properties and also gives detail about property description of AMBA-AHB using SVA properties, Section V gives Simulation results of AMBA-AHB and also gives report of coverage analysis while Section VI concludes the paper.

978-1-5090-3411-6/16 $31.00 © 2016 IEEE
DOI 10.1109/ICMETE.2016.67

## II. Architecture of AMBA AHB

The Advanced Microcontroller Bus Architecture (AMBA) is an open-standard, on Chip communication protocol for designing high performance buses on low power devices. The AHB is a pipelined bus, designed for high performance and high bandwidth operations. It can support upto 16 bus masters and slaves. It consist of AHB-masters, AHB-slaves, an AHB-arbiter and an AHB-decoder. The address bus can be upto 32 bit wide and the data bus can be upto 128 bit wide. There is a single global clock[1].

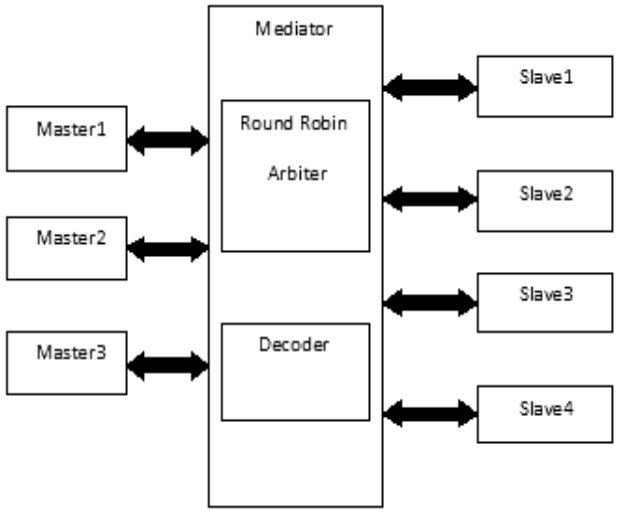


Fig. 1. Architecture of AMBA-AHB

Fig. 2. shows high-performance bus which nourish the external memory bandwidth, on which the high bandwidth external memory, on chip Memory and Direct Memory Access (DMA) devices situated. It also provides a high bandwidth interface between the different components that are intricated in the bulk of transfers[1].

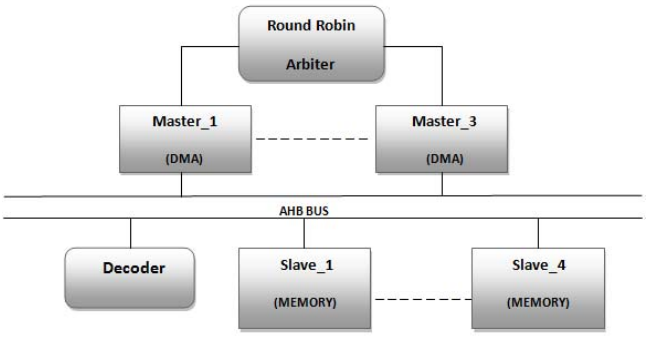


Fig. 2. Typical AMBA based microcontroller

## III. Components of AMBA AHB

The main components of AMBA-AHB are:

### A. AHB Master

It is used to initiate read and write operations by giving address and control information. At one time only single master can access the bus. To initiate a transfer, it drives its bus request signal to high[7].

### B. AHB Slave

It is used for performing read and write operations. Slave determines how the transfer should process after master has started a transfer. Whenever the slave is accessed it must provide a response which denotes the status of the transfer[7].

### C. AHB Arbiter

Arbiter is the main controlling component in the design. It is used to grant access to a particular master for the bus at a time. In round robin arbitrations mechanism, the accesses to the Bus is given to the masters as per there sequence. Round robin arbiter gives 1st priority to the 1st master, 2nd to the 2nd masters and so on. If two or more master is ready at a time it always gives the priority to the first one. It likes first come first serve basic [8].

### D. AHB Decoder

The AMBA-AHB needs a single centralized decoder for implementation of the bus. The decoder decodes the address given by the master and selects the slave for the data transfer process. The AHB decoder simply perform a direct decode of the address bus[7].

- Transfer of signals from master to slave involves:
  1) HCLK: global clock signal.
  2) HWDATA: write data from Master to Slave.
  3) HRDATA: read data from Slave to Master.
  4) HREADY: Ack signal from Slave to Master.
  5) HADDR: target address for transfer[9].

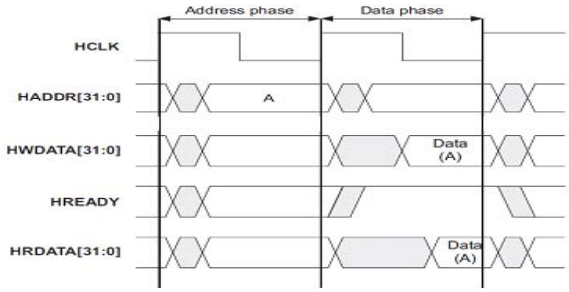


Fig. 3. Basic transfer

With respect to posedge of the clock, Master 'X' is putting its address and control information. If it is write operation, then data corresponds to first address comes to second clock cycle. Simultaneously,the address for next location is coming. This is a pipelining operation. Instead of wasting the address bus we are utilizing the address bus. Therefore, data for the second address comes in next clock cycle. The HREADY is a

"ack" signal from the slave. If it is ready to receive data it makes ready signal high. If it is not ready to receive data it makes ready signal low, so master will hold the same address and data. For read operation, data corresponds to first address comes to second clock cycle and read data from slave to master whenever HREADY is high[9].

## IV. Implementation Process

Verification is the process of reviewing, inspecting, testing, and documenting that the product behaves in a manner as defined by the product requirement specification.

The process of implementation follows these steps:

*1) Design of RTL Module*: Here, AMBA-AHB is modeled using 3 masters and 4 slaves in verilog language.

*2) Formation of SVA Properties:* In any design model, the functionality is represented by the combination of multiple logical events. These logical events are simple boolean expressions which is assessed on the same clock edge or events that get assessed over a period of time involving multiple clock cycles[10]. SVA provides a key word to define these events called "sequence". A number of sequences can be joined logically or sequentially to produce more complex sequences. SVA provides a keyword to define these complex sequential behavior called "property". The property is verified during a simulation. SVA provides a keyword called "assert" to check the property[11].

*property_name*: *assert property* ( @(*sample_signal*) disable iff ( *expression*)
// *optional disable condition*
*property_expression_or_sequence* );

The steps involved in the creation of a SVA are:

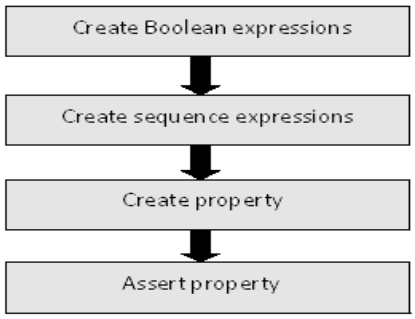


Fig. 4. SVA building block

- *Property Description of AMBA using SVA Properties:* Based on the protocol description of the AMBA-AHB, the following SVA checks can be extracted. These properties checks basic transfer of signals from master to slave and vice-versa. Creating such intermediate expressions makes the SVA checkers more readable.

*CHECK #1:*
**property** *DMA_INVALID_DMA_STATE*; @(posedge clk)
disable iff ( ! reset)
(dma_disable==1'b1); **endproperty**
**assert property**(not *DMA_INVALID_DMA_STATE*);

*CHECK #2:*
**property** *DMA_INVALID_DMA_REQUEST*;
@(posedge clk) disable iff ( ! reset)
(ahb_slv_dmareq==1'b0); **endproperty**
**assert property**(not *DMA_INVALID_DMA_REQUEST*);

*CHECK #3:*
**property** *DMA_INVALID_DMA_ACCESS*;
@(posedge clk) disable iff ( ! reset)
(slv1_en == 1'b0 && slv2_en == 1'b0 && slv3_en == 1'b0 && slv4_en == 1'b0); **endproperty**
**assert property**(not *DMA_INVALID_DMA_ACCESS*);

*CHECK #4:*
**property** *DMA_SUCCESS_PROPER_DMA*;
@(posedge clk) disable iff ( ! reset)
(slv1_grant==1'b1 || slv2_grant==1'b1 || slv3_grant==1'b1 || slv4_grant==1'b1); **endproperty**
**assert property**(*DMA_SUCCESS_PROPER_DMA*);

- *Properties using Sequences:*
*CHECK #5:*
**property** *DMA_INVALID_GRANTS*;
@(posedge clk) disable iff ( ! reset)
sx |=> sy **endproperty**
**sequence** sx; (hmaster==8'b00000000) ; **endsequence**
**sequence** sy; (slv1_grant == 1'b0 && slv2_grant==1'b0 && slv3_grant == 1'b0 && slv4_grant==1'b0); **endsequence**
**assert property**(*DMA_INVALID_GRANTS*);

- *Controlling Assertions:*
`ifdef *ASSERT_ON*
**assert property**(@(posedge clk) disable iff (! reset) !(dma_disable==1'b1));
`endif;

`ifdef *ASSERT_OFF*
**assert property**(@(posedge clk) disable iff (! reset) !(dma_disable==1'b1));
`endif;

*3) Binding of RTL Module to the Property Module:* To facilitate verification separate from design, it is possible to specify properties and bind them to specific modules or instances. It permits verification engineers to verify with minimum changes to the design files. No semantic changes to the assertions are introduces due to this feature. Other advantage of keeping assertions in a separate file is that they can be independently verified without the need to have control of RTL files. A big advantage when you want to make sure that both the design and verification progress in parallel[12].

Binding of two different modules:

**module** dma ( a, b, c) // module containing the design

< RTL Code >

**endmodule**

**module** dma_property ( a, b, c) //module containing properties

< Assertion Properties >

**endmodule**

**Bind** dma dma_property dma_inst_1(a, b, c); //module that binds design module to property module

- dma and dma_property are the module name.
- dma_inst_1 is dma_property instance name.
- Ports (a, b, c) gets bound to the signals (a, b, c) of the

module dma[12].

*1) Assertion Based Verification of AMBA-AHB using Different SVA Properties:* The different properties of AMBA-AHB are verified by writing assertion for the properties and verified it using ModelSim. The following table describes some of the SVA properties that are verified for the AMBA-AHB design. The AMBA-AHB is verified by 8 properties which are described below:

TABLE I. VERIFICATION OF AMBA AHB

| S.No. | Property Description | Status |
|---|---|---|
| **1.** | Master is in valid state. (!dma_disable==1'b1)<br>**Fault:** Master is in invalid state. | **PASS**<br>**FAIL** |
| **2.** | Arbiter gives grant signal to master_2 and master_2 sends the data to the destination.<br>**Fault:** Arbiter gives the grant signal to master_2, but it fail to transfer the data to the destination. | **PASS**<br>**FAIL** |
| **3.** | Arbiter gives grant signal to master_3 and master_3 sends the data to the destination.<br>**Fault:** Arbiter gives the grant signal to master_3, but it fail to transfer the data to the destination. | **PASS**<br>**FAIL** |
| **4.** | Slave_1 request for bus master while other slaves are idle.<br>**Fault:** Slave_1 does not respond to the read/write operation of its master. | **PASS**<br>**FAIL** |
| **5.** | Slave_2 request for bus master while other slaves are idle.<br>**Fault:** Slave_2 does not respond to the read/write operation of its master. | **PASS**<br>**FAIL** |
| **6.** | Slave_3 request for bus master while other slaves are idle.<br>**Fault:** Slave_3 does not respond to the read/write operation of its master. | **PASS**<br>**FAIL** |
| **7.** | Slave_4 request for bus master while other slaves are idle.<br>**Fault:** Slave_4 does not respond to the read/write operation of its master. | **PASS**<br>**FAIL** |
| **8.** | Master_1 makes its bus request signal high and drives the address and control information to the bus.<br>**Fault:** Master_1 does not response to the signal from slave_1 | **PASS**<br>**FAIL** |

*2) Corner Cases of AMBA-AHB using SVA:* Assertion can also be used to verify the corner cases of the design. Corner cases are those cases that are hard to find, which occurred very rarely. E.g. for arbiter corner cases are arbiter generating grants when there is no request to the arbiter. The following table describes some of the SVA properties that are used to verify the corner cases for AMBA-AHB using assertion based verification.

TABLE II. CORNER CASES

| S.No. | Property Description | Status |
|---|---|---|
| **1.** | Arbiter does not gives grant to any master even if it is free. | **FAIL** |
| **2.** | Master_1 does not send a request to arbiter but arbiter sends a grant to master_1 | **FAIL** |
| **3.** | Slave_1 never inform the arbiter that it is now able to service master_1 | **FAIL** |
| **4.** | Even after Slave_1 has informed its ability to service Master_1, the arbiter ignores the bus request from Master_1 forever. | **FAIL** |

## V. SIMULATION RESULTS OF AMBA AHB

Here the dma1, dma2 and dma3 acts as the 3 masters and slv1, slv2, slv3 and slv4 acts as the 4 slaves of the design. Fig. 5 (a) shows the read and write operation of the bus, and the data transfer between the master and slave. It shows that at any one clock cycle any one master can transfer the data. The data is first written into the memory by making the signal data-in high for that master and then memory read the data by making the data-out signal high, the destination slave read the data from the memory. The data is written into the address bus and data is read from the data bus of the design. Here in our case we take a 8 bit data.

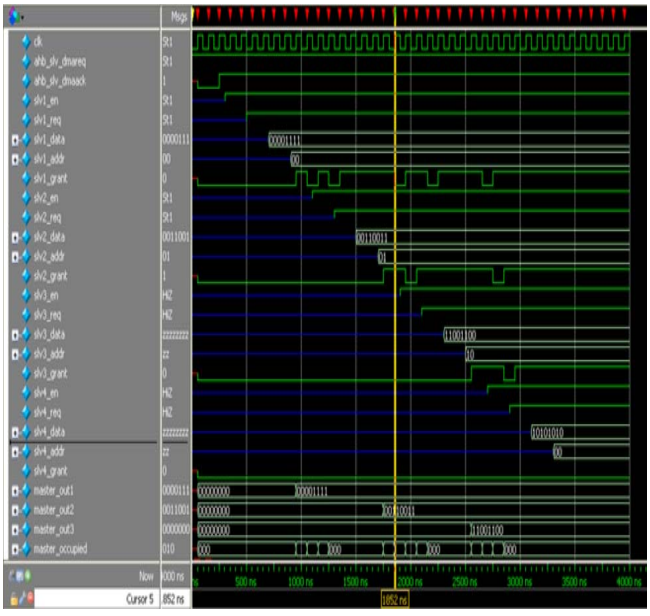

Fig. 5(a). Simulation of AMBA-AHB

If a design has been compiled with assertion, we can view the assertion waveform in wave window. As shown in the fig. 5(b). and fig. 5(c). The left column of the figure shows the name of assertion directive and the name of each directive comes from assertion code.

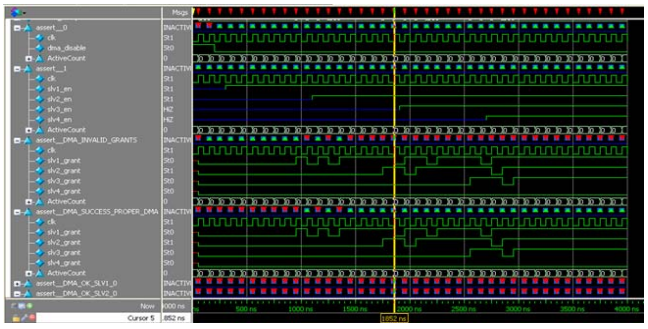

Fig. 5(b). Simulation of Assertions

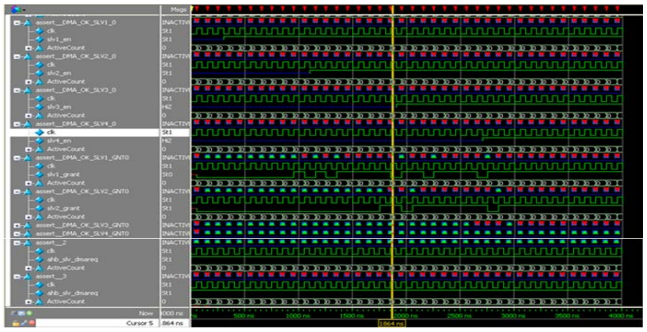

Fig. 5(c). Simulation of Assertions

TABLE III. ASSERTIONS REPORT

| Assertion name | Language | Fail count | Pass count | Attempt count |
|---|---|---|---|---|
| DMA_INVALID_DMA_STATE | SVA | 2 | 38 | 40 |
| DMA_INVALID_DMA_ACCESS | SVA | 0 | 40 | 40 |
| DMA_INVALID_GRANTS | SVA | 29 | 11 | 40 |
| DMA_SUCCESS_PROPER_DMA | SVA | 12 | 28 | 40 |
| DMA_OK_SLV1_GNT0 | SVA | 26 | 14 | 40 |
| DMA_OK_SLV2_GNT0 | SVA | 21 | 19 | 40 |
| DMA_OK_SLV3_GNT0 | SVA | 14 | 26 | 40 |
| DMA_OK_SLV4_GNT0 | SVA | 1 | 39 | 40 |
| DMA_INVALID_DMA_REQUEST | SVA | 0 | 40 | 40 |

TABLE IV. shows coverage analysis of AMBA-AHB using different number of assertions. As the number of assertions increases, the overall coverage increases. Four assertions are used initially giving the coverage of 27.5%. As the number of assertions increments to twenty, the coverage report increased to 80.3%.

TABLE IV. COVERAGE REPORT

| Number of Assertions | Total Coverage Report (%) |
|---|---|
| 4 | 27.5 |
| 8 | 37.6 |
| 15 | 68.5 |
| 20 | 80.3 |

## VI. CONCLUSION

SystemVerilog Assertion (SVA) ensure true assertion based verification integrated into Verilog/SV language. Assertion have full visibility to all design code, do not hide in comments as in PSL. Binding permits verification engineers to add assertions to design without touching the design files. SVA are a team effort some assertions are written by the design team and some are written by the verification team. We design the AMBA-AHB in Verilog language and obtained coverage of 80.3% using 20 number of assertions written in SVA and verified in ModelSim. First the AMBA-AHB bus is designed using 3 masters and 4 slaves in verilog language. This design is then verified using SVA bind construct. The different properties of AMBA-AHB and its corner cases properties are verified using ModelSim and the total coverage report of the design is calculated.

## REFERENCES


[1] Akshay Mann, Ashwani Kumar," Assertion Based Verification of AMBA-AHB Using Synopsys VCS®"International Journal of Scientific & Engineering Research, Volume 4, Issue 11, November-2013 ISSN 2229-5518).

[2] Mostafa, Moaz, Mona Safar, M. Watheq El-Kharashi, and Mohamed Dessouky. "System Verilog Assertion Debugging Based on Visualization, Simulation Results, and Mutation", 2014 15th International Microprocessor Test and Verification Workshop, 2014.

[3] P.P. Chakrabarti. "H-DBUG: A High-level Debugging Framework for Protocol Verification using Assertions", 2005 Annual IEEE India Conference - Indicon, 2005.

[4] PSL AND SVA: TWO STANDARD ASSERTION LANGUAGES ADDRESSING COMPLEMENTARY ENGINEERING NEEDS John Havlicek, Freescale Semiconductor, Inc., Austin, TX Yaron Wolfsthal, IBM Haifa Research Lab, Haifa, Israel.

[5] SystemVerilog Assertions and Functional Coverage,2014.

[6] Sonny, Ashley T, and S. Lakshmiprabha. "OVL, PSL, SVA: ABV using checkers and standard assertion languages" 2013, International Conference on Advanced Computing and Communication Systems, 2013.

[7] Divekar, Shraddha, and Archana Tiwari. "Interconnect matrix for multichannel AMBA AHB with multiple arbitration technique", 2014 International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE), 2014.

[8] M. Conti, M. Caldari, G.B. Vece, S. Orcioni, C. Turchetti, "Performance Analysis of Different Arbitration Algorithms of the AMBA AHB Bus", in Proc. of IEEE Conference on Design Automation , 2004, San Diego, USA, pp. 618 – 621.

[9] M.J. Dougherty. "A SEM-E module avionics computer with PI-Bus backplane communication", IEEE Conference on Aerospace and Electronics, 1990.

[10] Sonny, Ashley T, and S. Lakshmiprabha. "OVL, PSL, SVA: Assertion based verification using checkers and standard assertion languages", 2013 International Conference on Advanced Computing and Communication Systems, 2013.

[11] Ashok B. Mehta Los Gatos, CA USASystemVerilog Assertions and Functional Coverage, Guide to Language, Methodology and Applications, ISBN 978-1-4614-7323-7 ,DOI 10.1007/978-1-4614-7324-4 Springer New York Heidelberg Dordrecht London

[12] "Introduction to SVA", A Practical Guide for SystemVerilog Assertions, 2005, Microprocessor Systems, 1995 & Understanding Behavioral Synthesis, 1999 & ARM Ltd., AMBA specification (rev. 2) 1999.