Design and Verification of AMBA AHB Protocol Using UVM

Spoorthi Kumari and Prabha Niranjan

Abstract SOC's have multiple blocks integrated on a single chip, to interact between these blocks, on-chip bus architecture is required. ARM's AMBA (Advanced Microcontroller Bus Architecture) is one of the extensively used on-chip architecture used in the VLSI industry. AHB (Advanced High-Performance Bus) is an AMBA Bus protocol with high-performance, supports multiple masters and multiple slaves and has wider bandwidth; as a result of these features, AHB is generally used in SOC's. Previously, AHB design was verified using either Verilog or system Verilog, where the time spent on verification was more than the time spent on design. A more effective verification methodology, like UVM (Universal Verification Methodology), must be adopted to reduce the average time spent in verification and increase efficiency. An AHB protocol with one master and three slaves is designed in ModelSim using Verilog and verified in QuestaSim using UVM, and a coverage report is generated. QuestaSim and ModelSim are the EDA tools developed by Mentor Graphics for design and verification purposes.

Keywords SOC · AMBA · AHB · Verilog HDL · UVM

1 Introduction

ARM established the open standard bus architecture known as AMBA, which is commonly utilized for on-chip bus systems. The purpose of AMBA is to reduce the design by enabling the usage of replaceable parts inside the SoC. The AMBA includes bus types like APB (Advanced Peripheral Bus), ASB (Advanced System Bus), AHB (Advanced High-performance Bus) and AXI (Advanced Extensible Interface).

S. Kumari (✉) · P. Niranjan
Department of Electronics and Communication Engineering, Nitte (Deemed to Be University), NMAM Institute of Technology, Nitte, Karnataka 574110, India
e-mail: Spoorthibangera21@gmail.com

P. Niranjan
e-mail: prabhan@nitte.edu.in

S. Kalya et al. (eds.), *Advances in VLSI, Signal Processing, Power Electronics, IoT, Communication and Embedded Systems*, Lecture Notes in Electrical Engineering 1062,
https://doi.org/10.1007/978-981-99-4444-6_6

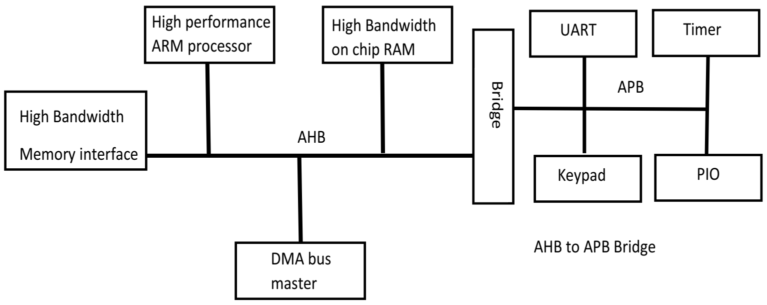

High performance ARM processor
High Bandwidth on chip RAM
UART
Timer
APB
Bridge
High Bandwidth Memory interface
AHB
Keypad
PIO
DMA bus master
AHB to APB Bridge

Fig. 1 AMBA AHB block diagram [1]

AHB is designed to address the requirements of high-performance, high bandwidth synthesizable systems [1].

AMBA AHB includes the following characteristics:

- Burst operation
- Split transaction
- Single-cycle bus master handover
- Single-clock edge operation
- Non-trivial implementation
- Supports up to 128-bit bus configuration

Figure 1 shows the AMBA AHB block architecture, which comprises of high-performance devices like on-chip RAM, memory interface and DMA (Direct Memory Access) all connected through an AHB Bus. On the other side, APB (Advanced Peripheral Bus) connects the low bandwidth peripheral devices like UARTs, PIOs, keypads, and timers. A central bridge connects the high-performance bus to the peripheral bus, allowing signals from on-chip devices to be transmitted to peripheral devices.

The work aims to design an AMBA AHB Protocol with one master and three slaves using Verilog HDL and verify the design using UVM.

The paper is structured as follows. Section 2 includes a literature survey on different AMBA AHB verification methodologies. Section 3 describes the design methodology. Section 4 explains the verification methodology. Section 5 describes the simulation results, and Sect. 6 concludes the work with the final analysis.

2 Literature Review

AHB system with three masters and four slaves designed and verified in ModelSim using system Verilog assertions. Where AHB attributes and corner case properties were evaluated, yielding a total coverage report of 80.3% with 20 system Verilog assertions [2].

AHB Single Master Single Slave, Single Master Multi slave, and Multi Master Single Slave were designed and verified in ModelSim using System Verilog [3]. By developing a Verification environment, a comparison was made between Verilog and System Verilog verification methodologies. The comparison revealed that the System Verilog Verification approach produced more accurate results than Verilog.

An AHB protocol was designed using Verilog and verified using system Verilog in QuestaSim [4]. Where the design under test was verified, and coverage report of nearly 65% was generated.

Using a Xilinx simulator, AMBA AHB was designed and verified for simple write and read operations with basic examples using Verilog [5].

AHB system with three masters and four slaves was designed and implemented in Verilog HDL using Xilinx ISE 14.6 [6]. Random combinations of masters and slaves were given to incremental burst operations of four, eight, and sixteen beats.

UVM was used to verify the AHB Protocol, which was built for three masters and four slaves. According to the UVM study conducted, the results obtained were more precise [7].

A single master single slave and multi master multi slave AHB system was designed and verified in ModelSim using UVM (Universal Verification Methodology) [8]. By interfacing STM32 controller, the performance latency of the AHB bus and the baud rate of the USART peripheral were measured which provided 100% functionally corrected output for UVM verification.

To speed up the verification process, a generic and reusable verification environment for SoC buses was developed. When compared to previous methodologies, the recommended methodology reduced verification costs and speeds the verification process [9].

AMBA AHB system was designed with three masters and three slaves, and was verified using both system Verilog and UVM (Universal Verification Methodology) in QuestaSim. Both verification approaches generated coverage reports, with UVM yielding a higher level of coverage than System Verilog [10].

The majority of the relevant work done in the preceding review is on Verilog hardware descriptive language and System Verilog. Along with the design, more work needs to be done in the verification process by using improved verification methodologies.

3 Design Methodology

AMBA AHB architecture with one master and three slaves is shown in Fig. 2, where the master is provided with the bus address, control information and data. The master sends the address to the decoder through the HADDR signal; the decoder broadcasts this address to all of the slaves in the system. The slave with this address will respond to the decoder by turning on its HSELx signal.

Once the HSELx signal is HIGH, the master will begin transferring data to that slave based on the HTRANS signal, which determines the mode of transfer as shown

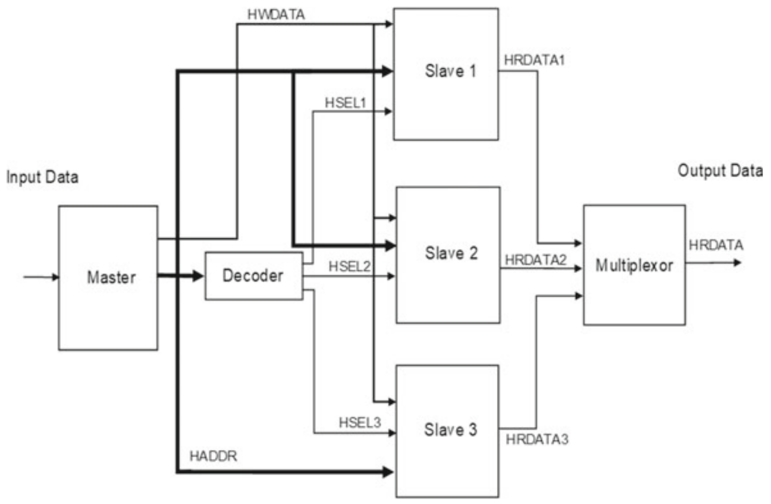


Fig. 2 AMBA AHB with one master and three slaves

in Table 1, where there are four data transfer modes IDLE, BUSY, NON-SEQ, and SEQ. The SEQ mode data transfer is dependent on burst operations, like incrementing or Wrapping burst. Once the transfer mode is selected HWRITE signal is used to execute the Write or Read operation.

When the HWRITE signal is HIGH, data is written to the slave, and when the HWRITE signal is LOW, data is read from the slave. After completing the Write or Read, the response, i.e., the HRDATA signal from all slaves, is received and sent to the multiplexer. The multiplexer will multiplex these signals and transmit the HRDATA along with the HRESP signal to master, stating that the data transaction has been completed.

Table 1 Transfer modes of AHB

HTRANS [1:0]	Type	Description
00	IDLE	Master is given access to the bus but does not intend to execute any data transfer. IDLE transfers will be rejected by slaves
01	BUSY	Master includes IDLE state between transfer of burst, indicating that master is already processing bursts of data and is unable to handle the next data immediately
10	NON-SEQ	Initial transfer in a burst or single transfer
11	SEQ	The succeeding transfer in a burst, where the address is the previous transfer's address plus the size (in bytes)

4 Verification Methodology

In VLSI industry large number of chips are being produced every year with a greater number of features. As all features in a chip must be verified before the specified Time to Market, the verification speed has to be increased. This is achieved through the use of a better verification language and a faster verification methodology.

UVM is an IEEE 1800.2 standard since 2017 which was developed by Synopsis in 2010. It is a hybrid of two methodologies: OVM (Open Verification Methodology) and VMM (Verification Methodology Manual). UVM is basically a set of classes defined in the System Verilog language. UVM is a new technology used for verification that is supported by the majority of EDA vendors and extensively accepted in the VLSI industry. UVM minimizes the work of constructing a standard testbench, and also verification components of UVM could be reused across multiple designs.

A. Components of UVM

Figure 3, illustrates a UVM verification environment that includes an interface connected to a DUT with a testbench, agent, generator, driver and monitor forming the subcomponents of the test bench.

(1) Generator: It generates a set of randomized data (where range of random data is specified by the *use of assertions) for randomization test or a single data which will be sent to the driver.*
(2) Transaction: moves data from generator to driver and also from monitor to scoreboard. Generated by extending UVM_Sequence_item.
(3) Driver: Driver produces stimulus and sends to DUT (Design under Test) through the interface. This is generated by extending UVM_component class.

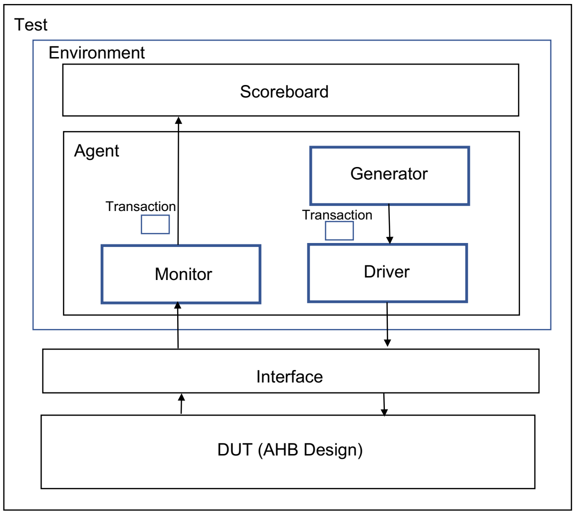


Fig. 3 UVM verification environment

(4) Agent: The agent consists of the verification components like driver, monitor, and generator which are connected using TLM (Transaction Level Modeling) connections. This is generated by extending UVM_component class.
(5) Interface: It connects driver and monitor to DUT.
(6) Monitor: Receives random or single output from DUT. Derived from UVM_component class.
(7) Environment: Generator, agents, driver and monitor are together placed inside the environment.
(8) DUT (Design Under Test): DUT is the actual design which has to be verified.
(9) Reference Model: This is constructed in case of randomization test to determine whether the random output from DUT is valid.
(10) Scoreboard: It collects the DUT output from the monitor and the output from the reference model and compares both the outputs to check whether DUT outputs are valid.
(11) Testbench: It consists of the test cases. It is generated by UVM_component class.

B. Verification Flow:

The verification can be done in two ways,

1. Directed Test.
2. Randomization Test.

In directed test, single input data is generated and given to the DUT, to obtain respective output. If we have multiple testcases, it is time consuming to check the output for each input data given. So, a set of random data is given as input, and output is determined, this process is called as randomization. As shown in Fig. 4, random data is generated from the generator, this random data is then passed to the driver through transactions, this is called as transaction level modeling (TLM). The driver will send this data along with the instructions to be performed through the interface to the DUT. In DUT, the data is executed and random output is obtained and this is sent to the monitor. To verify the result, a reference model is constructed, which has the same functionality as of the DUT, reference model also takes the random data and produces output. The output from monitor and reference model is given to scoreboard to check whether the results are matching. If it matches, the data has been successfully transferred from slave to master.

The scoreboard results contain the following.

(1) Checker: Where assertions are inserted to check whether design is functioning proper or not.

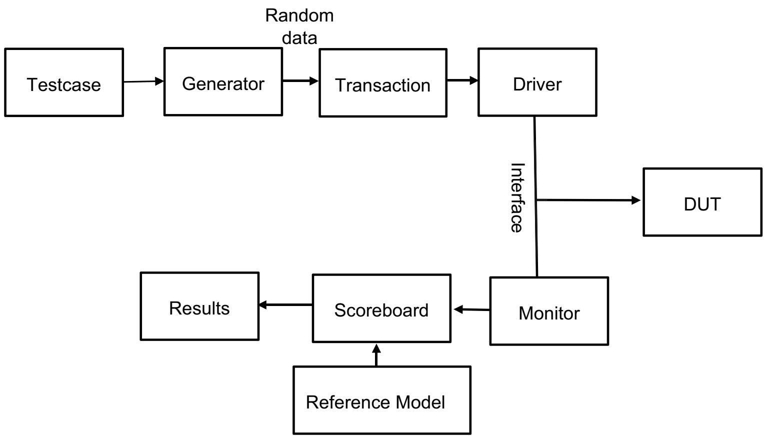


Fig. 4 Verification Flow

(2) Coverage: Coverage is classified into two categories.

- *Code coverage:* The percentage of code that was generated during verification. It can help in locating dead code and false paths, that are not generated for any given combination of inputs.
- *Functional coverage*: Indicates the percentage of successful transactions.

5 Results and Discussion

A. Decoding:

In the first case, the address given by the master is 32' h0034, the slave 1 has this address in its memory so the slave 1 is activated by making its HSEL1 signal HIGH as shown in Fig. 5.

In the second case the address 32'h0054 given by the master matches with address present in slave 2, Hence slave 2 is activated by making HSEL2 signal HIGH and Slave 3 will be having the address 32'h0094 in its memory list, so it is activated by making HSEL3 as HIGH.

B. Data Transfer from Master to Slave 1:

When the HWRITE signal goes HIGH, the data 05 h given by the master, is written to the address 34 h present in the memory of slave 1 as shown in Fig. 6. When the HWRITE signal goes LOW, the data which was written into the slave memory is read back which is shown in Fig. 7.

C. Verification Results

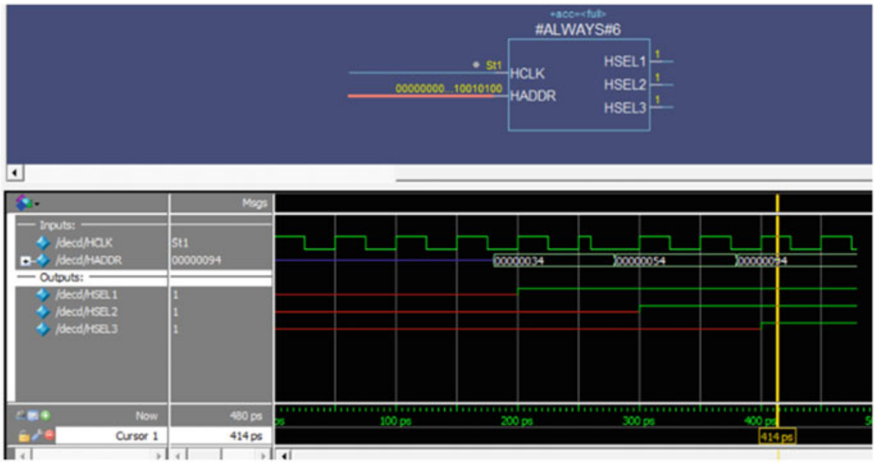


Fig. 5 Decoder Output

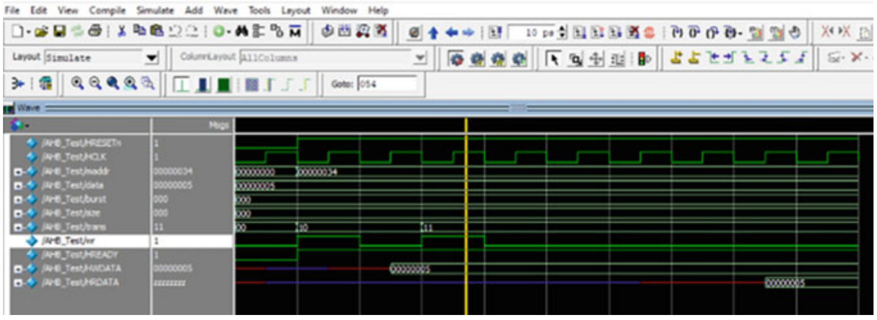
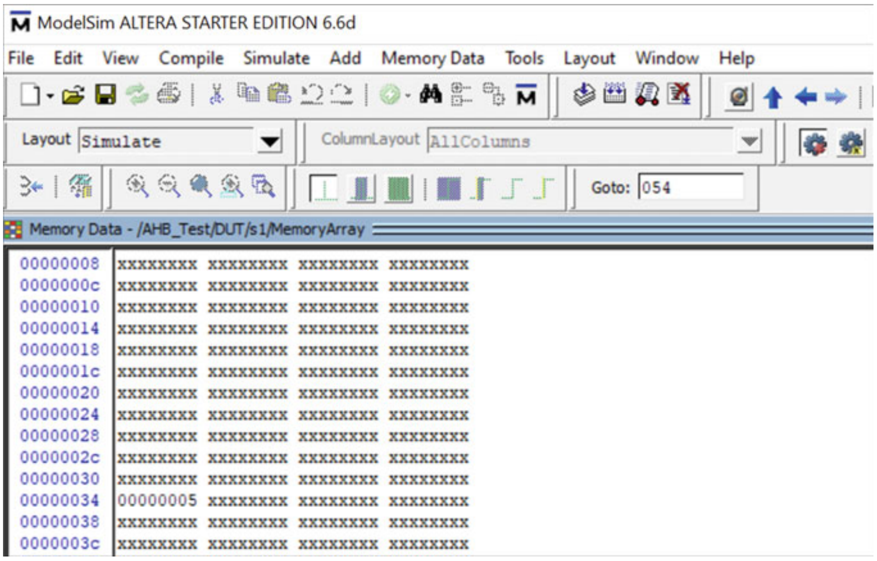


Fig. 6 Output of AHB write and read

The verification results obtained from the scoreboard shown in Fig. 8 show that the output obtained from the DUT and reference model for a random input data is matching, this indicates that the design is generating 100% functionally corrected output.

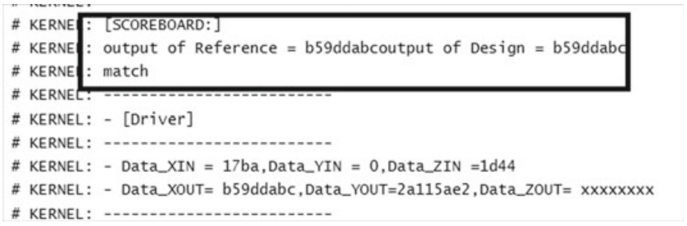
Figure 9, shows the coverage report where Coverage obtained around 96.8%, which shows that most of the test cases are being covered by this method.

Figure 9, shows the coverage report where Coverage obtained around 96.8%, which shows that most of the test cases are being covered by this method.


ModelSim ALTERA STARTER EDITION 6.6d
File Edit View Compile Simulate Add Memory Data Tools Layout Window Help
Layout Simulate
ColumnLayout AllColumns
Goto: 054
Memory Data - /AHB_Test/DUT/s1/MemoryArray

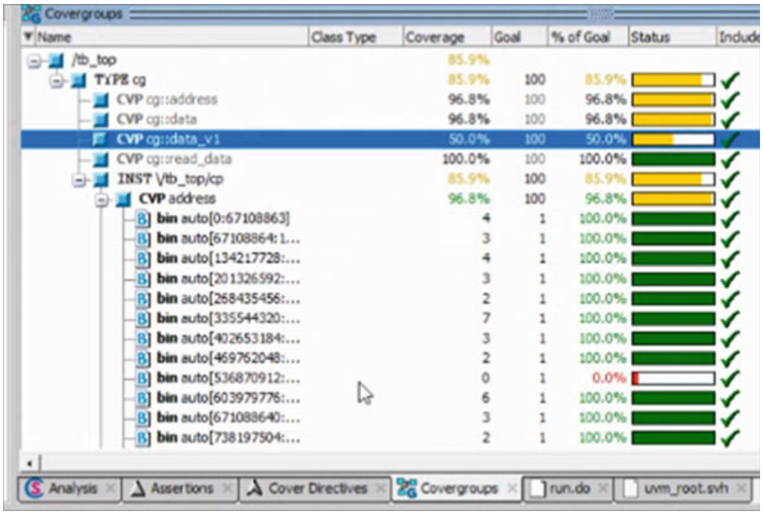
```
00000008 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000000c xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000010 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000014 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000018 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000001c xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000020 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000024 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000028 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000002c xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000030 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000034 00000005 xxxxxxxx xxxxxxxx xxxxxxxx
00000038 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000003c xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
```

Fig. 7 Internal Memory of Slave 1



```
# KERNEL: [SCOREBOARD:]
# KERNEL: output of Reference = b59ddabcoutput of Design = b59ddabc
# KERNEL: match
# KERNEL: -------------------------
# KERNEL: - [Driver]
# KERNEL: -------------------------
# KERNEL: - Data_XIN = 17ba,Data_YIN = 0,Data_ZIN =1d44
# KERNEL: - Data_XOUT= b59ddabc,Data_YOUT=2a115ae2,Data_ZOUT= xxxxxxxx
# KERNEL: -------------------------
```

Fig. 8 Scoreboard Results



Name	Class Type	Coverage	Goal	% of Goal	Status	Include
/tb_top		85.9%				
TYPE cg		85.9%	100	85.9%		✓
CVP cg::address		96.8%	100	96.8%		✓
CVP cg::data		96.8%	100	96.8%		✓
CVP cg::data_v1		50.0%	100	50.0%		✓
CVP cg::read_data		100.0%	100	100.0%		✓
INST \/tb_top/cp		85.9%	100	85.9%		✓
CVP address		96.8%	100	96.8%		✓
bin auto[0:67108863]		4	1	100.0%		✓
bin auto[67108864:1...		3	1	100.0%		✓
bin auto[134217728:...		4	1	100.0%		✓
bin auto[201326592:...		3	1	100.0%		✓
bin auto[268435456:...		2	1	100.0%		✓
bin auto[335544320:...		7	1	100.0%		✓
bin auto[402653184:...		3	1	100.0%		✓
bin auto[469762048:...		2	1	100.0%		✓
bin auto[536870912:...		0	1	0.0%		✓
bin auto[603979776:...		6	1	100.0%		✓
bin auto[671088640:...		3	1	100.0%		✓
bin auto[738197504:...		2	1	100.0%		✓

Fig. 9 Coverage Results

6 Conclusion

This work gives an overview of the AMBA AHB bus protocol, designed with one master and three slaves. The AHB design is implemented in ModelSim tool using Verilog and verified in QuestaSim using Universal Verification Methodology. To verify the exact working and functionality of the design, the entire design is tested under all possible cases in accordance to its specification. The coverage report of 96.8% is obtained from this verification methodology which gives better coverage compared to that of other verification methodologies. In the present work conducted AHB is designed, with one master and three slaves. In future one can design an AHB with 16 masters and 16 slaves and also UVM methodology could be used to verify other buses like APB, AXI, DMA etc.

References

1. https://developer.arm.com/documentation/ihi0011/a/AMBA-AHB. Accessed 08 April 2022
2. Gurha P, Khandelwal RR (2016) System verilog assertion based verification of AMBA-AHB. In: International conference on micro-electronics and telecommunication engineering (ICMETE). IEEE
3. Divya M, Radhakrishna Rao KA (2018) AHB design and verification AMBA 2.0 using system Verilog. Int J Adv Res Ideas Innov Technol 4(3)
4. Choudhury P, Giridhar P (2019) Design and verification of AMBA AHB. In: 1st international conference on the advanced technology in intelligent control, environment, computing & communication engineering (ICATIECE). IEEE

5. Shivakumar BR, Deeksha L (2019) Efficient design and implementation of AMBA AHB bus protocol using verilog. In: International Conference on Intelligent Sustainable System (ICISS). IEEE
6. Bhavana Surya S, Hema C, Shruthi Priya M, Devikarani U, Rathod AS (2021) Design and implementation of multiple master, multi slave interface in AMBA AHB protocol. Int J Res Eng, Sci Manag 4(7)
7. Ramakrishna MS, Sridevi BS (2018) Verification of advanced high-speed bus in UVM methodology. Int J Res Eng, Sci Manag 1(12)
8. Deepthi VD (2019) Performance verification of AMBA multi master AHB bus using system Verilog. Int Res J Eng Technol (IRJET) 6(6)
9. Hussien A, et al (2019) Development of a generic and a reconfigurable UVM-based verification environment for SoC Buses. In: 31st international conference on microelectronics (ICM)
10. Dilip K, Vijaya Prakash AM (2021) Design and verification of AHB protocol using system verilog and universal verification methodology (UVM). Int J Eng Res Appl 11(7)