International Journal of Scientific & Engineering Research, Volume 4, Issue 11, November-2013
ISSN 2229-5518

Assertion Based Verification of AMBA-AHB Using Synopsys VCS®

Akshay Mann, Ashwani Kumar

Abstract-The successof assertion based functional verification depends on the debugging environment associated with it. It helps user to get information about the environment in a refined manner. It also helps in realizing visualization support which tracks the behavioral aspects of the design under verification. This paper presents the design and verification of widely used AdvancedHigh-performanceBus(AHB)protocoloftheAdvancedMicroprocessorBusArchitecture(AMBA) using Assertion Based Functional Verification approach.This paper contributes as follows: (1) Designing of AMBA AHB protocol using System Verilog language in Synopsys VCS® tool. (2) Implementing assertion for different components of the design. (3) Functional verification of the overall design using all the assertions and analyzing the obtained coverage report. Hence, with this systematic description of the work done, we are able to automatically and completely synthesize and functionally verify an important and widely used industrial protocol.

Index terms-Functional Verification, Assertion, AMBA AHB, Synopsys VCS®, Functional Coverage.

IJSER

1 Introduction

THE time taken for verifying the design is becoming tedious everyday as the complexity of the chip design is increasing exponentially. Nowadays, about 70% of the design time is needed for developing the verification environment [1]. The implementation of reused-based design, where the main blocks are recycled by using the existing designs, results in shifting of efforts from the design field to the verification field. Although, the present electronic devices have different functionalities, but the market demands new functionalities for which more efforts are required in design and verification from the same device. Also considering the constrained time-to-market pressures, new advanced techniques should be implemented [2]. Hence in this condition, it is important to reduce the verification time so as to speed up the whole development process [2]. But, on the other hand, the complexity of the design makes it difficult for the engineers to cover all the corner cases in minimum time. Therefore, to increase the design observability and to find and interpret its faults, a new technique is earnestly needed.

In the verification process, debugging is divided into three phases. The first step is error detection, which finds that the design is not working properly in a particular environment [3]. The second phase is error diagnosis in which the verification engineer identifies the exact location of the design which is causing the incorrect behavior [4]. The third step is error correction, in which the faulty part of the design is replaced by the corrected components. In today's scenario, Assertion-Based Verification (ABV) has been well accepted among the design and verification community as it plays an important role in the error detection phase [5]. Assertions aim to localize the failure and thus minimize the effort to locate the exact reason of the failure. However, assertions are mainly specified at the signal level, and thus do not automatically diagnose design behavior at higher levels of abstraction (phase level, transaction level) [6].

Assertion-based verification (ABV) is such a method, which combines assertion, simulation and formal techniques to the traditional functional verification [2]. Thispaperdescribesthe functional verificationoftheAHBprotocoloftheAdvancedMicroprocessorBusArchitecture(AMBA).Throughoutthisworkthe given protocol is functionally verified by writing assertions for different AHB components. AMBA AHB architecture basically contains four different modules (Master, Slave, Arbiter and Decoder). This paper describes how assertions for different components help in the overall functional verification of the design.

The paper is organized as follows: Section 1 gives the introduction about assertion based verification and AMBA AHB architecture. Section 2 describes the AMBA AHB architecture and its different components. Section 3 gives the assertion based functional verification of AMBA AHB architecture using assertions for different modules and overall functional coverage of the design. Finally, section 4 concludes the work presented in this paper.

2. AMBA AHB Architecture

A widely used Advanced Microprocessor Bus Architecture (AMBA) aims at easing the component design by using the combination of interchangeable components in the SoC design [7]. It supports the reusability of intellectual property components, so that a least part of the design can become a composition.

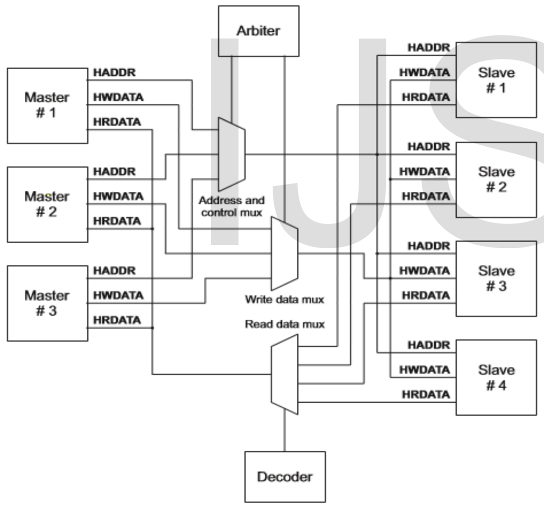


Fig. 1: AMBA AHB Architecture.

Fig. 1 shows the basic architecture of AMBA AHB. It consists of four main components- Master, Slave, Arbiter and Decoder. Arbiter is the main controlling component in the design. Decoder is used for decoding the addresses of different slaves. Master and Slave are used for performing read and write operations.

2.1 Components of AMBA AHB

1. AHB Master- A master initiates the read and write operations by providing the address and control information to the interconnected design [8]. Only single master can access the system bus at a time. Fig. 2 shows the schematic of AHB Master. When access is granted to any of the masters through Hgrant, address (HADDR) and control operations are performed. Also other signals like burst, Htrans, Hready and Hlock signals are activated to initiate read and write operations in AHB protocol.

2. AHB Slave- The slave responds to the read and write operations of the master within the given address space range [9]. Also, it acknowledges to the master whether the read and write operations are successfully implemented. Fig. 3 shows the schematic of AHB Slave. A particular slave is selected by sel signal and data write (Hwdata) and read (Hrdata) operations are performed. The slave selects the data to be read from signals indata1, indata2 and indata3. After data is read by the slave through read signal, it acknowledges to the master and arbiter by respective signal.

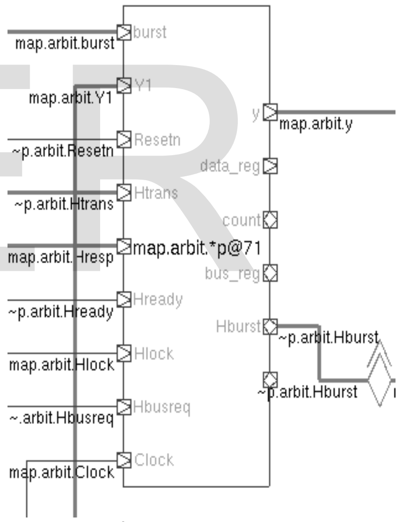


Fig. 2: AHB Master.

IJSER

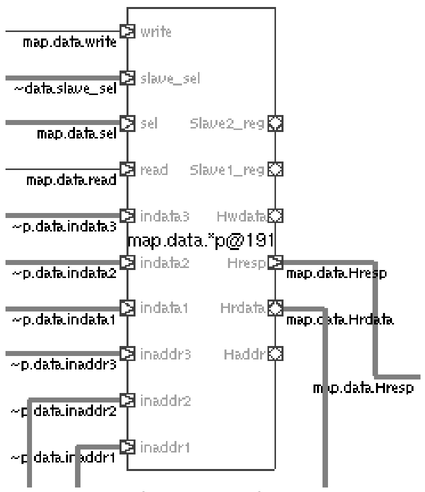


Fig. 3: AHB Slave

3. AHB Arbiter- An arbiter is used to grant access to a particular master for the bus at a time. The access can be granted to master according to any arbitration algorithm which decides the priority of the masters [10]. An AHB includes only single arbiter that would work as trivial in the single bus master systems. Fig. 4 shows the schematic of AHB Arbiter.

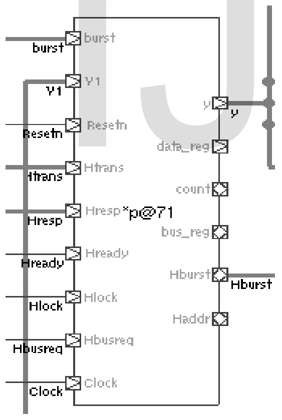


Fig. 4: AHB Arbiter.

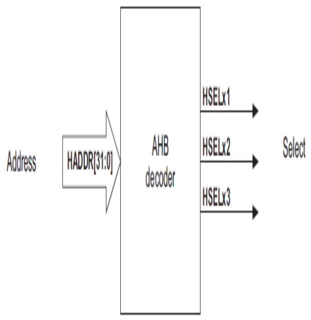


Fig. 5: AHB Decoder.

An arbiter is used for bus request (Hbusreq) and bus grant (Hgrant). The arbiter also initiates other signals like burst, Hready, Hlock, Hbusreq and Htrans when the master is ready for the data transfer. Arbiter is the main controlling component of the AMBA AHB design.

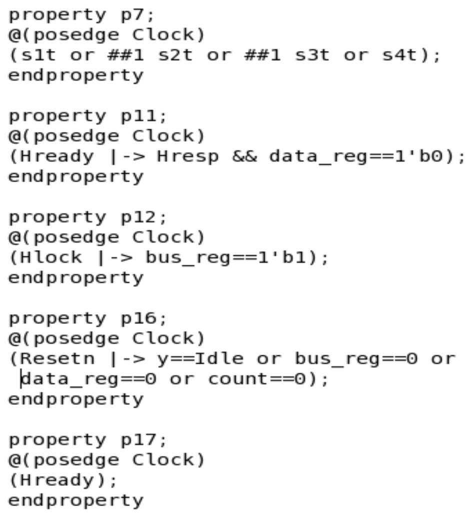
4. AHB decoder- The decoder decodes the address given by the master and enables the slave by selecting it to complete the transfer process [11]. A single centralized decoder is needed for all AHB implementations. Fig. 5 shows the schematic of AHB Decoder. From the corresponding addresses(HADDR), a decoder is used to select slaves (HSELx) from the address signals inaddr1, inaddr2, inaddr3.

3 Assertion based Functional Verification

Assertion based functional verification of AMBA AHB has been done by using twenty assertions. These assertions are implemented to find the overall functional coverage of the AMBA AHB design. First, separate assertions are used for different components (master, slave, arbiter and decoder) and their functional coverage is analyzed from the coverage report. After then, all the assertions are implemented to find the overall functional coverage of the design. The analysis of different components is described below.

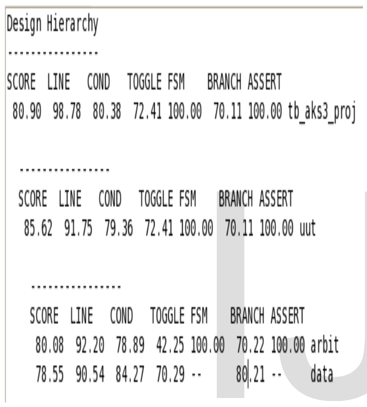
3.1 AHB Master-Fig. 6 shows the assertion example of AHB master. The assertions used here are concurrent which depends on clock pulse. Property p7 initially checks the Hready signal and later on, it asserts Hresp and data_reg signals to be true. Property p12 checks the Hlock and bus_reg signals. Property p16 first assert the signal Resetn and then check other signals like y=idle, data_reg, bus_reg and count to be zero while property p17 separately checks the Hready signal. Fig. 7 gives the coverage report obtained after applying master assertions. The report shows the functional coverage along with code coverage (line, condition, toggle, fsm, assert and branch). The assertion waveform of the assertions and cover points a12, c12, a16 and c16 are shown in the fig. 8. The successful assertions are shown with 'up' arrows and failed assertions with 'down' arrows.

IJSER



```
property p7;
@(posedge Clock)
(s1t or ##1 s2t or ##1 s3t or s4t);
endproperty

property p11;
@(posedge Clock)
(Hready |-> Hresp && data_reg==1'b0);
endproperty

property p12;
@(posedge Clock)
(Hlock |-> bus_reg==1'b1);
endproperty

property p16;
@(posedge Clock)
(Resetn |-> y==Idle or bus_reg==0 or
 data_reg==0 or count==0);
endproperty

property p17;
@(posedge Clock)
(Hready);
endproperty
```

Fig. 6: Assertion examples for AHB Master.



```
Design Hierarchy
................
SCORE  LINE   COND   TOGGLE FSM    BRANCH ASSERT
 80.90  98.78  80.38  72.41 100.00  70.11 100.00 tb_aks3_proj


  ................
  SCORE  LINE   COND   TOGGLE FSM    BRANCH ASSERT
   85.62  91.75  79.36  72.41 100.00  70.11 100.00 uut


    ................
    SCORE  LINE   COND   TOGGLE FSM    BRANCH ASSERT
     80.08  92.20  78.89  42.25 100.00  70.22 100.00 arbit
     78.55  90.54  84.27  70.29 --      80.21 --     data
```

Fig. 7: Assertion report for AHB Master

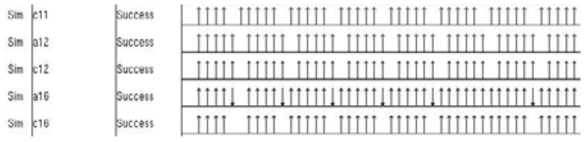
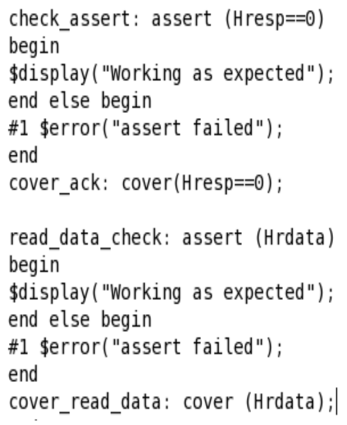


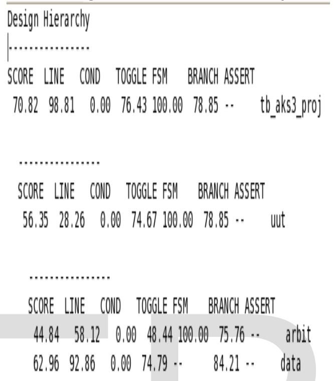
Fig. 8: Assertion waveform for AHB Master.

3.2 AHB Slave-The assertions examples for AHB slave are shown in fig. 9. The assertions used here are immediate ones which do not require any clock pulse. First immediate assertion checks the Hresp signal to be zero while the other assertion checks Hrdata signal. Fig. 10 shows the assertion report obtained when slave assertions are used. Since the immediate assertion are not covered under the assert coverage, it is shown as zero in the figure. Also other coverage is also less as compared to all the component assertions. Fig. 11 shows the assertion waveform for the assertions check_assert, cover_ack and read_data_check.



```
check_assert: assert (Hresp==0)
begin
$display("Working as expected");
end else begin
#1 $error("assert failed");
end
cover_ack: cover(Hresp==0);

read_data_check: assert (Hrdata)
begin
$display("Working as expected");
end else begin
#1 $error("assert failed");
end
cover_read_data: cover (Hrdata);
```

Fig. 9: Assertion examples for AHB Slave



```
Design Hierarchy
................
SCORE  LINE   COND   TOGGLE FSM    BRANCH ASSERT
 70.82  98.81   0.00  76.43 100.00  78.85 --     tb_aks3_proj


  ................
  SCORE  LINE   COND   TOGGLE FSM    BRANCH ASSERT
   56.35  28.26   0.00  74.67 100.00  78.85 --     uut


    ................
    SCORE  LINE   COND   TOGGLE FSM    BRANCH ASSERT
     44.84  58.12   0.00  48.44 100.00  75.76 --     arbit
     62.96  92.86   0.00  74.79 --      84.21 --     data
```

Fig. 10: Assertion report for AHB Slave

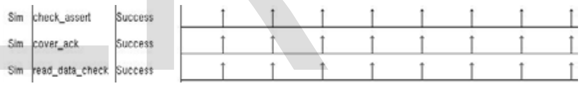
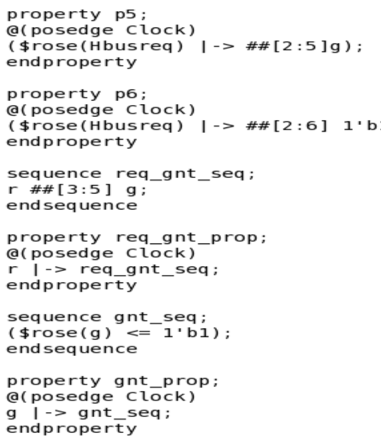


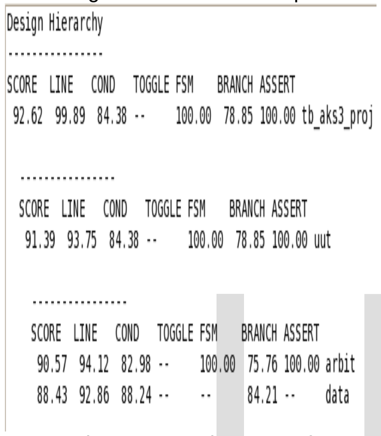
Fig. 11: Assertion waveform for AHB Slave

3.3 AHB Arbiter- The assertion examples for most important component in the AHB protocol, i.e. AHB Arbiter, are shown in fig. 12. The assertions used here are concurrent which depends on positive edge of clock pulse. Property p5 checks the Hbusreq signal through $rose command, after that it asserts the grant (g) signal between second and fifth clock pulse. Property p6 checks the Hbusreq signal between two adjacent clocks by $rose command and gives the result between second and sixth clock pulse while property gnt_prop checks the grant (g) signal with the same procedure. Fig. 13 shows the assertion coverage report for AHB arbiter. As seen from the fig. 13, arbiter assertions gives the highest functional and code coverage excluding the toggle coverage. Fig. 14 shows the assertion waveform for the assertions a6 and a7 with their cover points c6 and c7 where successful assertions are shown by ‘up’ arrows.



```
property p5;
@(posedge Clock)
($rose(Hbusreq) |-> ##[2:5]g);
endproperty

property p6;
@(posedge Clock)
($rose(Hbusreq) |-> ##[2:6] 1'b1);
endproperty

sequence req_gnt_seq;
r ##[3:5] g;
endsequence

property req_gnt_prop;
@(posedge Clock)
r |-> req_gnt_seq;
endproperty

sequence gnt_seq;
($rose(g) <= 1'b1);
endsequence

property gnt_prop;
@(posedge Clock)
g |-> gnt_seq;
endproperty
```

Fig. 12: Assertion examples for AHB Arbiter



```
Design Hierarchy
----------------
SCORE  LINE   COND   TOGGLE FSM    BRANCH ASSERT
 92.62  99.89  84.38 --     100.00  78.85 100.00 tb_aks3_proj


  ----------------
  SCORE  LINE   COND   TOGGLE FSM    BRANCH ASSERT
   91.39  93.75  84.38 --     100.00  78.85 100.00 uut


    ----------------
    SCORE  LINE   COND   TOGGLE FSM    BRANCH ASSERT
     90.57  94.12  82.98 --     100.00  75.76 100.00 arbit
     88.43  92.86  88.24 --     --      84.21 --     data
```

Fig. 13: Assertion report for AHB Arbiter

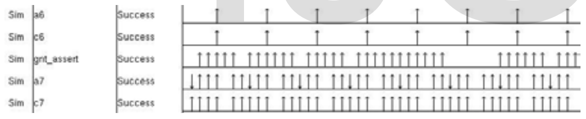
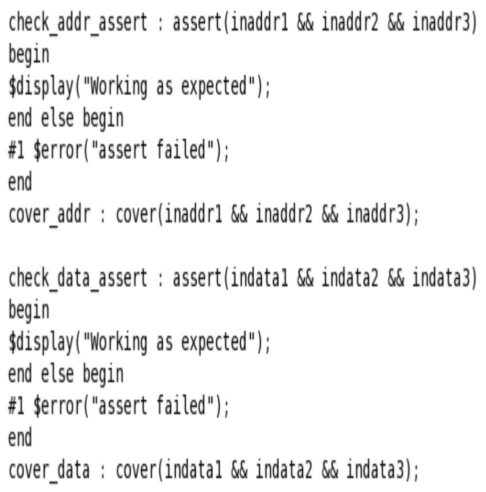
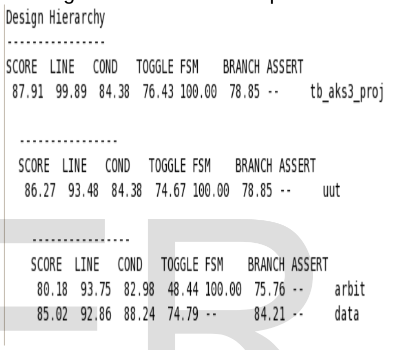


Fig. 14: Assertion waveform for AHB Arbiter



```
check_addr_assert : assert(inaddr1 && inaddr2 && inaddr3)
begin
$display("Working as expected");
end else begin
#1 $error("assert failed");
end
cover_addr : cover(inaddr1 && inaddr2 && inaddr3);

check_data_assert : assert(indata1 && indata2 && indata3)
begin
$display("Working as expected");
end else begin
#1 $error("assert failed");
end
cover_data : cover(indata1 && indata2 && indata3);
```

Fig. 15: Assertion examples for AHB Decoder



```
Design Hierarchy
----------------
SCORE  LINE   COND   TOGGLE FSM    BRANCH ASSERT
 87.91  99.89  84.38  76.43 100.00  78.85 --     tb_aks3_proj


  ----------------
  SCORE  LINE   COND   TOGGLE FSM    BRANCH ASSERT
   86.27  93.48  84.38  74.67 100.00  78.85 --     uut


    ----------------
    SCORE  LINE   COND   TOGGLE FSM    BRANCH ASSERT
     80.18  93.75  82.98  48.44 100.00  75.76 --     arbit
     85.02  92.86  88.24  74.79 --      84.21 --     data
```

Fig. 16: Assertion report for AHB Decoder

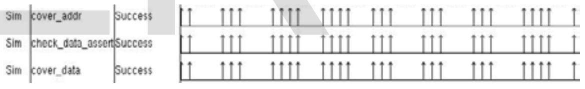


Fig. 17: Assertion waveform for AHB Decoder

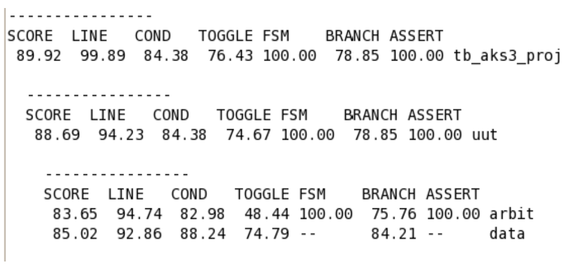
3.4 AHB Decoder-The assertion examples for the centralized AHB decoder is shown in fig. 15 where immediate assertions are used. The first assertion check_addr_assert checks the addresses inaddr1, inaddr2 and inaddr3 while the second assertion check_data_assert checks the data indata1, indata2 and indata3. Fig. 16 shows the assertion report for the AHB decoder which gives the functional and code coverage for decoder assertions.

The assertions waveform for cover_addr, check_data_assert and cover_data are shown in fig. 17.

3.5 Functional Coverage of AMBA AHB

The overall functional coverage of AMBA AHB design using all the assertions is shown in the fig. 18. All twenty assertions are implemented to find the functional as well as code coverage of the AHB design. The overall functional coverage is 89.92% while the line, condition, toggle, fsm, branch and assert coverage are 99.89%, 84.38%, 76.43%, 100%, 78.85% and 100% respectively.

IJSER



```
----------------
SCORE  LINE   COND   TOGGLE FSM    BRANCH ASSERT
 89.92  99.89  84.38  76.43 100.00  78.85 100.00 tb_aks3_proj

  ----------------
  SCORE  LINE   COND   TOGGLE FSM    BRANCH ASSERT
   88.69  94.23  84.38  74.67 100.00  78.85 100.00 uut

    ----------------
    SCORE  LINE   COND   TOGGLE FSM    BRANCH ASSERT
     83.65  94.74  82.98  48.44 100.00  75.76 100.00 arbit
     85.02  92.86  88.24  74.79 --      84.21 --     data
```

Fig. 18: Overall functional coverage report of AMBA AHB

Table 1 shows coverage analysis of AMBA AHB using different number of assertions. From the table, as the number of assertions increases, the overall functional and code coverage increases. Four assertions are used initially giving the functional coverage 65.23%. As the number of assertions increase to twenty, the functional coverage increased to 89.92%.

Table 1: Coverage Analysis table for AMBA AHB

No. of Assertions	Line %	Condition %	FSM %	Toggle %	Branch %	Functional %
4	96.67	64.06	100	44.66	69.23	65.23
7	96.69	64.06	100	44.66	69.23	74.93
9	96.73	65.33	100	44.66	70.11	75.01
11	96.89	68.54	100	44.83	71.97	77.83
13	97.46	70.31	100	44.99	73.08	80.97
15	97.48	70.31	100	56.91	73.08	80.98
16	98.77	75.31	100	65.20	73.08	83.51
17	99.21	78.54	100	69.31	74.27	85.88
19	99.54	81.88	100	72.48	75.58	87.08
20	99.98	84.38	100	74.67	78.85	89.92

IJSER

4 Conclusion

In this paper, assertion based functional verification of AMBA with AHB protocol is implemented using two types of assertions (immediate and concurrent). Initially, different components of AHB architecture (master, slave, arbiter and decoder) are described. After that, assertions are implemented separately for different components and obtaining their coverage results. While writing assertions, all the corner cases have been covered using immediate and concurrent assertions. For a 32-bit AHB bus, a number of test cases are required. Therefore, ABV is very helpful at pinpointing the bugs in the design with less time. Finally, all assertions are implemented altogether in the complete AMBA AHB architecture and the overall functional coverage report is analyzed. The result shows that the implemented method increases the observability and coverage of the design. Future work will focus on improving the functional and code coverage metric and approaching towards full coverage of the design.

Acknowledgments:I would like to take this opportunity to express my gratitude to the people whose assistance has been invaluable in this paper.

References

[1] YashdeepGodhal, KrishnenduChatterjee, Thomas A. Henzinger, "Synthesis of AMBA AHB from Formal Specification: A Case Study", in *International Journal on Software Tools for Technology Transfer*, July 2011.

[2] Yangyang Li, Wuchen Wu, LigangHou, Hao Cheng, "A Study on the Assertion-Based Verification of Digital IC", in Proc. of *Second International Conference on Information and Computing Science*, 2009, pp. 25-28.

[3] A. Nandi, B. Pal, N. Chhetan, P. Dasgupta and P. P. Chakrabarti, "H-DBUG: A High-level Debugging Framework for Protocol Verification using Assertions", in Proc. of *IEEE Indicon Conference*, 2005, Chennai, India, pp. 115-118.

[4] Roychoudhury, T. Mitra, S.R. Karri, "Using formal techniques to Debug the AMBA System-on-Chip Bus Protocol", Proc. of *IEEE Computer Society of Design, Automation and Test in Europe*, 2003,Munich, Germany, pp. 828 – 833.

[5] P. Chauhan, E. Clarke, Y. Lu, and D. Wang, "Verifying IP core based system-on-chip designs", in Proc. of *12th Annual IEEE ASIC SOC Conference*, 1999, India, pp. 27-31.

[6] M. Benjamin, D. Geist, A. Hartman, G. Mas, R. Smeets, and Y. Wolfsthal, "A Study in Coverage-Driven Test Generation," in Proc. of 36th *Design Automation Conference (DAC'99)*, 1999, Los Angeles, USA, pp. 970-975.

[7] ARM Ltd., AMBA specification (rev. 2) 1999, http://arm.com/products/solutions/AMBA_Spec.html.

[8] Han Ke, D. Zhongliang, S. Qiong, "Verification of AMBA Bus Model Using System Verilog", in Proc. of *IEEE Conference on Electronic Measurement and Instruments (ICEMI '07)*, 2007, Xian, China, pp. 1-776, 1-780.

[9] Y. Lin, C. C. Wang, I. J. Huang, "AMBA AHB Bus Protocol Checker with Efficient Debugging Mechanism", in Proc. of *IEEE International Symposium*

on Circuits and Systems (ISCAS 2008), 2008, Washington, USA, pp. 928 – 931.

[10] 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.

[11] L. Ivanov and R. Nunna, ''Specification and formalverification of interconnect bus protocols,'' in Proc. of *43rd IEEE Midwest Symposium on Circuits and Systems*, vol. 1, Aug. 2000, pp. 378-382.

IJSER