Implementation of SystemVerilog Environment for Functional Verification of AHB-DMA Bridge

Disha Goswami
P.G Student, Department of Electronics Engineering,
Gujarat Technological University,
Ahmedabad, India

Dr. Dharmesh J. Shah
Director, Shruj LED Technologies
Ahmedabad, India,
djshah99@gmail.com

***Abstract*—** Now day's functional verification is a very hot topic. With the growing complexity of modern digital systems and embedded system designs, the task of verification has become the key to achieving faster time-to-market requirement for such designs. Verification is the most important aspects of the ASIC design flow. It is estimated that between 40 to 70 percent of total development effort is consumed by verification task. This paper describes the verification of AHB-DMA interface using system Verilog. System Verilog is the special hardware description language used in functional verification. The verification environment designed using System Verilog.

Keywords-*System Verilog, DMA, AHB, APB*

*****

I. INTRODUCTION

The enormous growth of VLSI technology enables the integration of more than several million transistors on a single chip to make a Soc (System-on-Chip). The process of verification consumes from 60% to 80% of the design cycle which is expected to continually increase. To overcome this problem is to adopt a reuse-oriented, coverage-driven verification methodology.

System Verilog is the industry's first unified Hardware Description and Verification Language (HDVL)[1]. The verification features of SystemVerilog include:

- Assertion-based verification
- Random constrained stimuli generation
- Functional coverage
- Advanced object-orientation

AMBA AHB-Lite provides requirements of high-performance designs. It is a bus interface that supports a single bus master and provides high-bandwidth operation [2]. AHB-Lite slaves are internal memory devices, external memory interfaces, and high bandwidth peripherals and also low-bandwidth peripherals can be included as AHB-Lite slaves like AMBA Advanced Peripheral Bus (APB). Bridging between this higher level of the bus and APB is done using an AHB-Lite slave, known as an APB bridge.

AMBA APB is low bandwidth and low performance bus. The bridge connects the high performance AHB bus to the APB bus. So, for APB the bridge acts as the master and all the devices connected on the APB bus acts as the slave. The bridge is used for communication between the high performance bus and the peripheral devices. Direct memory access, a technique for transferring data from main memory to a device without passing it through the CPU.

II. VERIFICATION CHALLENGES

Due to the increasing complexity of integrated circuits, verification engineers need a method to reduce the complexity of the design verification, So to overcome this, it is always good to have hardware verification language as a tool and SystemVerilog is preferred here. Some of the benefits using system Verilog as verification languages are here:

In the directed testcases it becomes difficult to make a perfect set of stimulus required to determine functionality. We can write directed testcases to check certain set of features, but it is always difficult to write direct testcases for all possible valid as well as error scenarios, when the number of features keeps doubling. Constrained random generation allows verification engineers to automatically generate tests for functional verification to hit all possible scenarios. By using constrained one can set the range within which test bench exercises the target device. By specifying constraints, one can easily create tests that can obtain hard-to-reach corner cases. The constraints define the legal values that can be specified to the random variables.

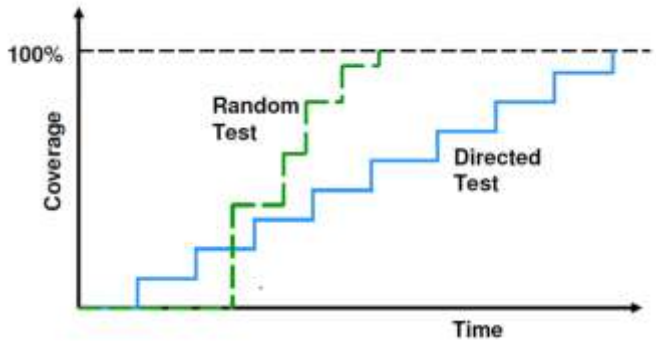


Figure.1. Random test Vs Directed test

 ISSN: 2321-8169
2799 - 2802

It is really significant to notice bugs in the early phase of design, to shorten time to market, that is the reason we need assertion based verification. Assertions are primarily used to validate the behavior of a design. Piece of verification code that monitors a design implementation for compliance with the specifications. ABV define properties that specify expected behavior of the design. By using ABV one can improve Improves observability of the design, Improves debugging of the design.

There is another advantage of using system verilog to reduce verification time is the use of code coverage and functional coverage. Code coverage is a tool that can identify what code has been executed in the design under verification. Code coverage basically gives the information about how many lines of code have been executed. Code Coverage indicates the how much of RTL has been exercised. The Functional Coverage indicates which features or functions have been covered under verification testsuits. Both of them are very important. With only Code Coverage, it may not represent the real features coverage. On the other hand, the functional coverage may leave out some unused RTL coverage. By using all these methods, verification engineers save their time in the test bench generation, bugs found in the early phase and product easily available at the time limit.

III. TESTBENCH ARCHITECTURE

A test bench is a layer of code that is created to apply input patterns (stimulus) to the design under test (DUT) and to see whether the DUT produces the outputs expected.

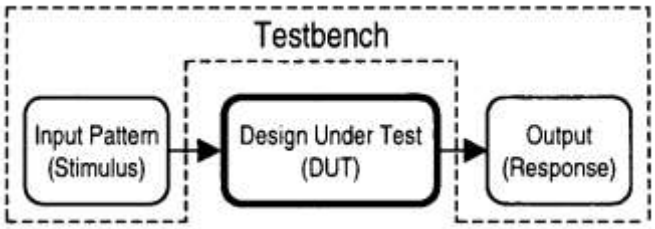


Figure.2.Testbench

A central concept for any modem verification methodology is

a layered testbench which helps you control the complexity, that is coming about in the instance of a testbench design itself, by breaking the problem into manageable pieces.

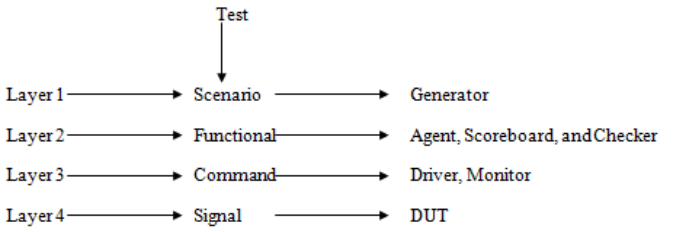


Figure.3. Layered Testbench

In figure 3 signal layer work with the DUT. We drive all the input signals of the DUT and observe the output from the output signals of the DUT. In command layer driver, and monitor exist. Here in the driver, we actually have the generated stimulus which we apply to a DUT. The driver drives the DUT according to driving protocol. Monitor performs the exact and the opposite operation from the driver. It collects all output signals from the DUT, which is applied to the scoreboard. In functional layer transactor, scoreboard, checker etc. exists in which we perform functional operation. In scoreboard we collect generated stimulus and monitor outputs. We keep track of passing and failed transactions in scoreboard. Checker check the functionality whether the output is as expected or not. Here I present testcase environment of AHB-DMA.

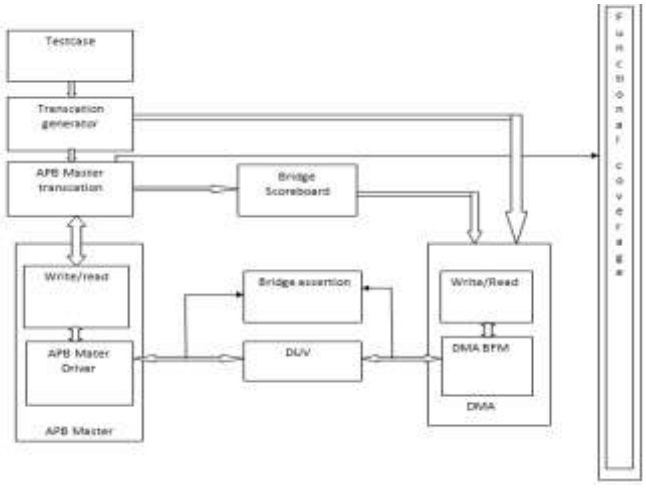


Fig.4.AHB-DMA verification testcase environment

IV. SYSTEMVERILOG VERIFICATION ENVIRONMENT

The following subsections describe the components of a verification component.

 ISSN: 2321-8169
2799 - 2802

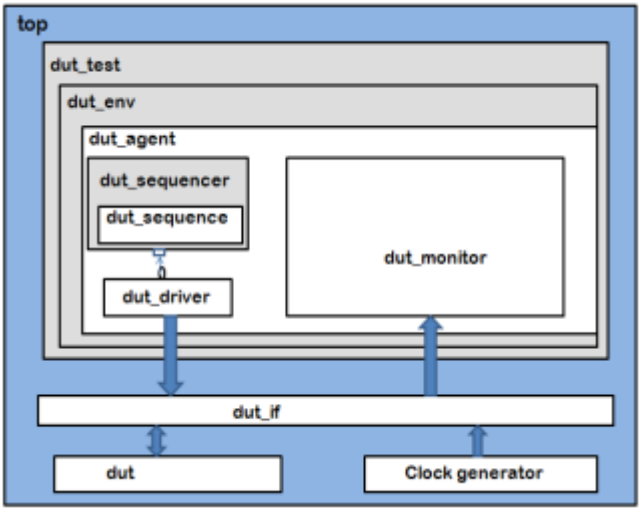


Figure.5.Verification environment

1) Top module

Top module is the highest level of layered testbench. In top module all modules are combined and instantiated. This is test case which is class of system Verilog which contains instances of apb_env.

2) dut_test

Test cases are identified from the design specification. Normally the requirement in test cases becomes a test case.

```
Program testcase (apb_interface intf);
```

3) dut_env

We collect all the modules and keep them one module named environment module. Thus the environment module has all other modules except top module.

```
class apb_env;

  virtual apb_interface intf; // interface instance
  master_agent agent;         //
       -----
       -----
       task run();
endtask: run
endclass : apb_env
```

4) dut_agent

The agent is basically a container. It contains driver, monitor and sequencer .it has instance of interface, driver and generator.

```
  class master_agent;
   virtual apb_interface intf; // interface instance
       master_generator generator; // instance of
generator
    master_driver driver;
```

5) dut_driver

All generated stimuli are collected to a driver module through the mailbox and then, according to driving protocol all these input stimuli are applied to the DUT. According to test plan requirement different scenarios are created and applied to DUT.

```
class apb_driver;
       //Declare objects
      //Define virtual interfaces
       //Define mailbox
       //Construction
       //Allocate memory to define objects
       //Make task which for APB read and write
         driver operation
endclass: apb_driver
```

6) dut_stimulus

The stimulus module contains all stimulus signal definitions which need to be randomized.

```
class master_generator;

       //Define objects
       //Define mailbox
       //Construction
       //Make one task which start the generator
       operation
       //Inside the task all signals are randomized
       which   are declared as rand in stimulus
       module
       //Put randomized data in to mailbox
endclass:master_generator
```

7)dut_monitor

Monitor module is used for monitoring all signals. It just sample the dut signal from the interface but does not drive them.

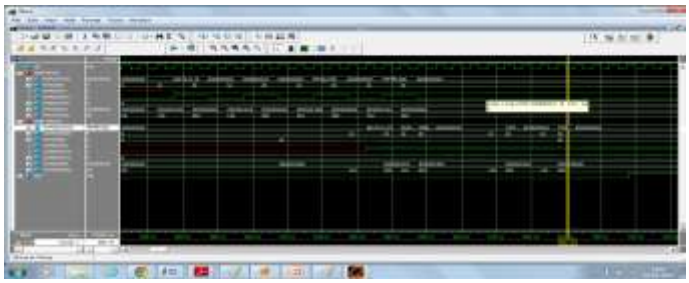
8) Mailbox

A mailbox is used to exchange messages between processes. Mailbox is a convenient method for inter-process communication. We can place and retrieve message with put() and get() method.

V. SIMULATION RESULTS

Here I represent the simulation result of AHB-DMA interface. In this first reads addresses and read data fetch from the memory to the FIFO and when the FIFO is full, the data write to the DMA.

 ISSN: 2321-8169
2799 - 2802



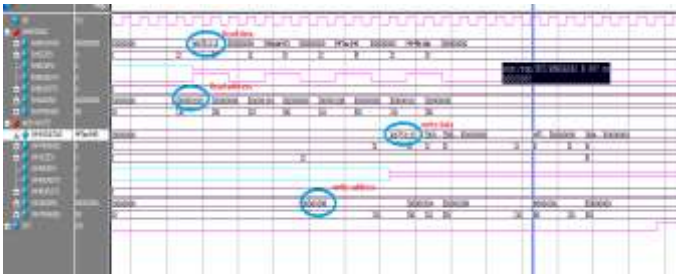


Figure.6 and 7 Simulation results with read /write data

ADDR	RD_DATA	TIME	ADDR	WR_DATA	TIME
32'h100	8'h15	455	32'h300	8'h15	575
32'h101	8'h11	455	32'h301	8'h11	575
32'h102	8'h75	455	32'h302	8'h75	575
32'h103	8'he6	455	32'h303	8'he6	575
32'h104	8'h35	495	32'h304	8'h35	605
32'h105	8'ha9	495	32'h305	8'ha9	605
32'h106	8'h6d	495	32'h306	8'h6d	605
32'h107	8'h58	495	32'h307	8'h58	605
32'h108	8'h40	535	32'h308	8'h40	655
32'h109	8'hc5	535	32'h309	8'hc5	655
32'h10a	8'h7a	535	32'h30a	8'h7a	655
32'h10b	8'h4f	535	32'h30b	8'h4f	655
32'h10c	8'hbb	575	32'h30c	8'hbb	685
32'h10d	8'hb1	575	32'h30d	8'hb1	685
32'h10e	8'h4f	575	32'h30e	8'h4f	685
32'h10f	8'h49	575	32'h30f	8'h49	685

Figure.8. Read/write data and address

VI. CONCLUSION

In this paper proposed uniform verification environment of AHB-DMA interface proposed using system verilog. To ensure the correctness of functionality of the design verification is much needed. Verification reduces time-to-market and makes design error free. To overcome the verification challenges system verilog is preferred because of its constrained random stimulus generation, code coverage and functional coverage.

REFERENCES

[1] Purvi Mulani,Jignesh Patoliya,Hitesh Patel, Dharmendra Chauhan," verification of i2c dut using systemverilog", IJAET, Vol.I, Oct.-Dec, 2010.

[2] Myoung-Keun You and Gi-Yong Song,"SystemVerilog- based Verification Environment using SystemC Custom Hierarchical Channel", 2009 IEEE.

[3] J. Bergeron, (2003), "Writing Testbenches: Functional Verification of HDL models", Kluwer Academic Publishers, 2003.

[4] System Verilog 3.1a Language Reference Manual: Accellera's Extensions to Verilog, Accellera, Napa, California,2004.