TLM based AMBA AXI4 protocol implementation using verilog with UVM environment

Harini H G¹, Kavitha V²

¹ M.Tech Student, Department of Electronics and Communication Engineering
CMR Institute of Technology, Bangalore, affiliated to VTU Belgaum India
² Ph.D scholar, Department of Electronics and Communication Engineering
Jain University Bangalore India

Abstract

During the entire phase of any project verification plays an important role. Most of the time and the effort are spend on verification. Transaction-level modeling (TLM) and Bus Functional modeling (BFM) are used in order to reduce this effort. Transaction-level modeling (TLM) is a technique used to describe the system by using the standard function calls which defines all the transactions which are required to verify the functionality of the system at the architecture level. Abstraction level rises by using this technique. Simulation speed and the modeling speed are more than the Register transfer level (RTL) Before building and testing the actual hardware a Bus Functional modeling (BFM) or Transaction Verification Models (TVM) are used to simulate the bus transactions. Standard buses enable the reusability of IP cores. Standard busses include AMBA from ARM, CORE CONNECT from IBM and others. There are various versions of AMBA like AMBA 1.0 to AMBA 5.0 in which AXI4 lite is an advanced extensible interface targeted for register style interface, it comes under AMBA4.0 specification. AXI4 is having excellent throughput, than AHB.

This project introduces the development of AXI4 protocol and verifying the same by UVM based test bench which supports TLM modeling.

Keywords: Transaction Level modeling (TLM), Advanced Extensible interface (AXI), Advanced Micro Controller Bus Architecture (AMBA), Methodology (OSVVM), Universal Verification Methodology (UVM), Verification Methodology Manual (VMM), Design under test (DUT).

1. INTRODUCTION

To meet the customer demands and the time to market always designers and the verification engineers look for methods which can reduce the effort as well as the time. Adopting Transaction-level modeling (TLM) technique and developing intellectual properties (IP) and flexible automated tools for design as well as verification are some of the methods which are targeted towards the same.

In section 1 this paper gives an introduction to the protocol used, and the modeling methodology used to design and verify the protocol, in section 2 it explains the top view of the verification environment, in section 3 it explains the sequence flow in this environment, and then the simulation results, conclusion and the future work are explained.

1.1 Introduction to BUS

In the early days as each chip manufacture had their own bus, standardization of the bus become necessary to enable the reusability of the intellectual properties (IP). Many manufacturer developed standard busses among which some of them become very popular due to their performance, hierarchy and the advanced features.

Standard busses which become very popular are Advanced Micro Controller Bus Architecture (AMBA) from ARM, CORE CONNECT from IBM. Using these standards interconnects not only enables intellectual properties (IP) reusability, provides flexibility, compatibility. These interconnects have multi-layer architecture. Between masters and slaves it can be used as a crossbar switch. Developing and verifying these interconnects or the busses become important. Because of the availability of wide variety of intellectual properties (IP) from ARM and advanced intellectual properties (IP) from third party vendors for Advanced Micro Controller Bus Architecture (AMBA) bus, this is more popular than others. There are various versions of Advanced Micro Controller Bus Architecture (AMBA) from AMBA1.0 to AMBA 5.0 [9]. This project particularly concentrates on AXI4 Lite protocol which is under AMBA4.0.AMBA 3 specification with five interface protocols forms AMBA 4.0 specification. In February 2013 AXI 4 was released, this adds new optional properties like ordering in advanced extensible Interface (AXI), Cache behaviour in Advanced Coherency Extensions (ACE) and Digital Visual Messaging (DVM).
AXI4 protocol main features are [2]

- Separate channels are defined for transferring the address, data and control information.
- Separate phases are defined for address, data and control information.
- By issuing only the start address burst transaction can be achieved.
- If any transaction is out of order due to some issues, it support for completing such transaction.
- Register stages can be added to enable the early timing closure.
- Byte strobes are present, can be used to support unaligned data transfer.
- Subset of AXI4 i.e AXI4lite and AXI4 stream are present.
- It has got five different channels for read data, read address, write data, write address and write response.

The main features of AXI4 lite are [2]

- Burst length = 1 for all the transactions.
- The length of the data that is accessed should be same as the width of the data bus (i.e. 32 bit or 64 bit).
- There is no support for exclusive access.
- Data accesses are Non-bufferable and Non-cacheable.

1.2 Related work with Respect to Transaction-level modeling (TLM)

As per the survey paper [17] Transaction-level modeling (TLM) is better than Register transfer level (RTL) in developing and verifying an interconnect.

1.3 UVM [10][11][12][13][14]

UVM means Universal Verification Methodology Reuse Methodology (eRM) was mainly used to develop the Open verification methodology (OVM). Universal Verification Methodology (UVM) was derived from Open verification methodology (OVM) version 2.1.1 and Verification Methodology Manual (VMM). Most of the previous methodologies are developed by the separate simulator vendors. But the Universal Verification Methodology (UVM) was developed by Accellera by taking the support from various vendors like Mentor, Synopsys, Cadence, AMD, Cisco, Freescale, and Intel. UVM increases the productivity of verification engineers. Important features of this methodology are

1. It is mainly used to write a test bench for all those designs which makes use of Verilog, VHDL, and SystemC.
2. It has a System Verilog classes Library which helps to achieve the reusability.
3. It supports the random verification which is constrained based where there is a generation of stimulus vectors along with the features of functional coverage and the checkers. This helps to cover an unexpected bug, and all the compute resources are utilized to the maximum extent.
4. It helps to develop a verification environment which is generic, reusable, scalable, and modular (i.e. where each component has a clear predefined functionality).
5. Even though it is simulation oriented, it also supports verification which is based on assertion, emulation and hardware acceleration.
6. It improves the quality of Verification process and is more transparent.
7. The architecture of Universal Verification Methodology (UVM) is layered based. Where each layer has components which can be reused. Without modification to the source code, configurability can be done to each of these components, so that they can be reused across different design under test (DUT).
8. The framework provided by the Universal Verification Methodology (UVM) supports the Coverage-Driven Verification.

Hence these methodologies which support TLM are adopted in this project to reduce the effort and the time

2. TOP VIEW OF UVM BASED AXI4 LITE VERIFICATION ENVIRONMENT

![Fig 1: AXI4 Lite UVM Test benches and Environments](image-url)

UVM test bench consists of verification components which are reusable. These verification components can be configured for different interface protocol or for any
system and can be instantiated. All these have a consistent architecture. And can be used for simulating, driving, monitoring, collecting.

The UVM environment contains virtual sequencer or multi-channel sequence this helps to provide the control of the test environment and synchronizes the timing and data between the different interfaces.

In this project all the UVM components are configured for AXI4 lite protocol.

2.1 AXI4 Lite UVM Verification Components

Depending on the requirement UVM environment consists of various verification components.

2.1.1 AXI4 Lite UVM Data item (Transaction):

In an UVM environment the input to the device under test is called the data item. They might be transactions over the bus, data packets over the bus or instructions to the design under test (DUT). Data item have got fields and attributes depending on the specification. These data are generated according to the requirement and are send to the design under test (DUT). By using System Verilog constraints, constrained based random data can also be generated. This helps to achieve the coverage which is maximum. In this project “rand logic” data type is used to generate the data item randomly. i.e

```systemverilog
... rand logic S_AXI_AWREADY,
  rand logic S_AXI_WREADY,
  rand logic [0:0] S_AXI_BID,
  rand logic [1:0] S_AXI_BRESP,
  rand logic [0:0] S_AXI_BUSER,
  rand logic S_AXI_BVALID,
  rand logic [0:0] S_AXI_ARREADY,
  rand logic [0:0] S_AXI_RID,
  rand logic [31:0] S_AXI_RDATA,
  rand logic [1:0] S_AXI_RRESP,
  rand logic S_AXI_RLAST,
  rand logic [0:0] S_AXI_RUSER,
  rand logic S_AXI_RVALID ...
```

2.1.2 AXI4 Lite UVM Driver:

It is used to drive design under test (DUT). Depending on the requirement it repeatedly receives the data item on request from the sequencer and then samples before driving it to design under test (DUT). If the operation that needs to be performed is read and write, the signals like address, data, read or write, enable and others are controlled by the driver for all the clock cycles in which these operations are performed.

```systemverilog
// get_and_drive will get the random data from the Sequencer
task axi4_lite_driver::get_and_drive();
begin
  // item is a transaction reference for the sequencer
  send_to_dut(item); // send the item to DUT
end
end task : get_and_drive
// item. kind indicates different operations done by the driver
case(item.kind)
  // AXI_Lite one sequence write operation (support single burst)
  WR_DATA: this.AXI_LiteM_1Seq_Write (input [31:0] awaddr, input [31:0] wdata, input [7:0] wait_clk_bready, input [7:0] rmax_wait);
  // AXI_Lite one sequence read operation (support single burst)
  AXI_LiteM_1Seq_Read (input [31:0] araddr, input [7:0] rmax_wait);
  // AXI_Lite Write address channel operation
  WAC:this.AXI_LiteM_WAC(input [31:0] awaddr);
  // AXI_Lite Write data channel operation
  WDC:this.AXI_LiteM_WDC(input [31:0] wdata, input [7:0] wmax_wait);
  // AXI_Lite write response channel operation
  WRC:this.AXI_LiteM_WRC(input [7:0] wait_clk_ready);
  // Read address channel operation
  AXI_LiteM_RAC(input [31:0] araddr);
  // read data channel operation
  WRDC:this.AXI_LiteM_RDC(input [7:0] rmax_wait);
  endcase
endcase
```

2.1.3 AXI4 Lite UVM Sequencer

It is used send and controls the data items to the UVM Driver. Even though it looks like a simple stimulus generator but it has got advanced features. That is random data item generated can be made constrained based, so as to capture all the unknown bugs in the design under test. These values generated can be made to have the distribution such that the coverage is maximum.

Axi4_lite_sequencer is derived uvm_sequencerbase class.
2.1.4 AXI4 Lite UVM monitor and checker

UVM monitor is not an active entity but it is a passive entity. Checker samples all the signals which are received from the DUT. All the coverage information is collected here and any checking if required is also done.

- A monitor collects all the signal information and converts it into a form, so that it is available to the tester and other components in the UVM environment.
- All the coverage information is also collected in the checker. It provides an option where we can even print the trace information.
- If the Design under Test (DUT) is based on some protocol specification, then the data and protocol checkers available can be used to perform the checking.

//Collect and Check operation

```plaintext
task axi4_lite_monitor::collect_data();
//Here all the data from the dutoutput are collected and the information is put into analysis report.
endtask :collect_data

function void axi4_lite_checker::check_it();
if((m_xactn.S_AXI_ADDR==m_xactn.S_AXI_ARADDR)&&(m_xactn.S_AXI_ADDR==m_xactn.S_AXI_RDATA))
//Here write address is same as the read address and
//write data is same as the read data is checked
endfunction :check_it
```

2.1.5 AXI4 Lite UVM Agent

With an aim to reduce the knowledge, effort and the time of a verification engineer while writing the test bench, a more abstract container called an agent is supported by the UVM environment. Sequencers, drivers, and monitors provide the reusability. In order to hook up each of these entities an environment integrator is required which is nothing but an agent. All these entities are encapsulated in an agent.

Depending on the requirement there can be more than one agent in a verification component. Those agent that are used to start the transaction are called transmit agents (master) and those which respond to these transaction are called receive agents (slave). Agents can be configured as passive or active. Those agents that generate the transaction according to the requirement to the DUT are called the active agent and those which do only monitoring the activities of DUT are called passive agent.

In this project agent is of type UVM_ACTIVE (i.e. active agent). Hence it contains all the components. All these are created by factory method supported by UVM environment.

```plaintext
monitor=axi4_lite_monitor::type_id::create("monitor",this);
m_checker=axi4_lite_checker::type_id::create("m_checker",this);/checker
driver=axi4_lite_driver::type_id::create("driver",this);
sequencer=axi4_lite_sequencer::type_id::create("sequencer",this);/sequencer
```

2.1.6 AXI4 Lite UVM environment

Depending on the requirement one or many agents or components like bus monitor are encapsulated inside the UVM environment. The configuration properties supported by the environment helps to customize the behavior, topology and reusability can also be achieved. An environment level monitor can also be configured. A passive agent can be converted into active and vice versa.

The uvm_env is the environment class provided by the UVM class library. It helps to

- Generate the constrained based random data item.
- Performs the protocol activity validation (checking).
- Coverage information is collected.
- And monitors all the DUT activity.
//build the object of agent type and configured it to be active.
function void axi4_lite_env::build_phase(uvm_phase phase);
    super.build_phase(phase);
    agent0=axi4_lite_agent::type_id::create("agent0",this);
    set_config_int("*","is_active",UVM_ACTIVE);
endfunction :build_phase

3. WORKING ENVIRONMENT OF AXI4 LITE
As shown in Fig 3 Sequences are generated randomly. Sequences will generate the transactions from the transactor which are given to sequencer. To pass to the driver from the sequencer TLM technique is used. From the driver the data is transferred to the design under test. Output and input data from the design under test are given to the monitor and the checker. And the checker will generate the checker log.

Fig 3: Working Environment of AXI4 Lite

4. SIMULATION RESULTS

Fig 4: AXI4 Lite write operation

5. CONCLUSION
Transaction-level modeling is an advanced method used to model the system at the architectural level and to perform the functional verification. Using this modeling reduces the effort and the time as it increases the simulation and modeling speed then RTL technique [4] [5] [6] [8]. And there are open source packages available which support Transaction-level modeling (TLM) method like OSVVM and UVM.

Hence in this project we have implemented the subset of AMBA AXI4 bus interconnect that is AMBA AXI4 Lite protocol. And then it is verified by using the Transaction level modeling method. The simulation results obtained by using this method matches with the handshaking which is defined in the protocol specification document. And also the developed model can be used as a TLM component in the development of prototype

6. FUTURE WORK
The project can be extended for the development and verification of AXI4 protocol. This has the following features.

1. Burst of data can be up to 256 beats.
2. Data width can be varied from up to 1024 bits.
3. Various types of handshaking are supported by the protocol. All these can be implemented
4. Different type of burst are supported which can be implemented like
   • FIXED
   • INCR
   • WRAP

REFERENCES
[1]. Transaction-based SoC Design Techniques for AMBA AXI4 Bus Interconnects using VHDL (By Daniel C.K. KhoTaulhop Solutions and Kumar Munusamy, Multimedia University, Faculty of Engineering).
[2]. AXI4_specification.pdf  

[3]. Comparing AMBA AHB to AXI Bus using System Modeling. (By Deepak Shankar)

[4]. TRANSACTION LEVEL MODELING, An Abstraction Beyond RTL. (By Laurent Maillet-Contoz and Frank Ghenassia, STMicroelectronics, France)


[7]. J. Lewis. The OS-VVM packages.[Online http://osvvm.org/]


[15]. Transaction Level Modeling in SystemC (Adam Rose, Stuart Swan, John Pierce, Jean-Michel Fernandez Cadence Design Systems, Inc)


[17]. Modeling method to develop an amba axi4lite bus interconnect : A survey. (By Harini H G In IJERT, Volume. 4, Issue. 04 , April – 2015)