Development of Basic Template Environment for Functional Verification of VLSI Design Using UVM

Vijayan U1, Anjo C.A2, Vignesh Raja B3, Arun Kumar N4

1PG Scholar, 2,3,4Assistant Professor, Vel Tech Multi Tech Dr. Rangarajan Dr. Sakunthala Engineering College, Chennai

Abstract—Due to development of semiconductor industries and ICs manufacturing, the functional verification requirement is much higher. The primary goal of functional verification is to verify the functional correctness of design. It ensures that design is free of errors and it can be manufactured to perform its intended functionality. The verification environment is developed using testbench. The testbench generates the stimulus for simulation, apply the stimulus to module under test, collect responses and compare the output response with expected values. The specific design is tested using simulators through which the designer feeds the corresponding inputs and observe the expected output of the system. In this paper, UVM methodology is used to develop the basic template verification environment which can be used for all projects, thus increases reusability feature.

Keywords—Agent, Driver, Monitor, Functional verification, Functional coverage, Sequencer, System Verilog, UVM, Verilog, Verification Methodology.

I. INTRODUCTION

Initially, Verilog language is used to create test benches. For complex projects, developing a Verification environment in Verilog is tedious process and consumes more time. Later, System Verilog is used to create Verification environment which uses OOPs concepts and reduces effort to develop testbench. System Verilog is an extension of Verilog language with all features of Verilog included. It can be used for both hardware design and verification.

In order to reduce time in developing verification environment and to improve reusability of test bench, Verification Methodologies are developed. Methodology is a systematic ways of doing things with a rich set of standard rules and guidelines. Methodology reduces verification efforts with its predefined libraries.

Methodology is basically set of base class library which can be used to build well defined and reusable test benches. It provides standard rules and predefined class library to develop verification environment. Many verifications methodologies are developed by different vendors. In this project, Universal Verification Methodology (UVM) is used to develop the basic template environment.

UVM class library provides the building blocks required to develop well-constructed and reusable verification components and test environment. UVM methodology is based on System Verilog and it is vendor independent. Hence, the verification environment can be simulated using all leading simulation tools.

II. UVM – VERIFICATION ENVIRONMENT

An UVM testbench is composed of reusable verification environments called verification component. Each verification component follows a consistent architecture and consists of a complete set of elements for stimulating, checking, and collecting coverage information for a specific protocol or design. The verification component is applied to the device under test (DUT) to verify the implementation.

Figure 1: UVM verification environment

Data items which represents the input to the device under test (DUT) are transactions. For example, it includes networking packets, bus transactions, and instructions. The fields and attributes of a data item are derived from the data item’s specification. By suitable randomization of data item using System Verilog constraints, meaningful tests can be created and maximize coverage. Functional coverage is used to find the coverage when randomizing packet generation to the DUT.
The environment comprises of one or more Agents, Scoreboard and Functional coverage. Agent includes sequencer, driver and monitor. The sequencer generates packets of data which will be generated based on different test cases. The task of driver is to drive the packets on to the interface which will be fed into DUT.

Each component and object will be created as per UVM standard guidelines. The main difference between component and object is, the components created will be exist throughout the simulation but not objects. All components and objects need to register in factory.

III. IMPLEMENTATION

Each UVM template environment consists of top_module, basic_test, basic_environment, basic_sequencer, basic_sequence, basic_sequenceitem, basic_driver, basic_monitor, basic_scoreboard, basic_coverage. All basic modules are of class data type and code will be developed on System Verilog and the file will be saved in .sv extension. Every class must be extended from base class library such as basic_driver must be extended from uvm_driver. UVM base class library is open source and it can be used for all projects developed using System Verilog.

A. UVM_OBJECT

All components and transactions derive from uvm_object, which defines an interface of core class-based operations: create, copy, compare, print, print, record. It also defines interfaces for instance identification (name, type name, unique id, etc.) and random seeding.

B. UVM_COMPONENT

The uvm_component class is the root base class for all UVM components. Components are quasi-static objects that exist throughout simulation. Every component is uniquely addressable using hierarchical path name.

C. UVM_TRANSACTION

The uvm_transaction is the root base class for UVM transactions. It is object and are transient in nature. It extends uvm_object to include a timing and recording interface. Simple transactions can derive directly from uvm_transaction, while sequence-enabled transactions derive from uvm_sequence_item.

D. UVM_ROOT

The uvm_root class is special uvm_component that serves as the top level component for all UVM components, provides phasing control for all UVM components, and other global services.

E. UVM_VOID

The uvm_void class is the base class for all UVM classes. It is an abstract class with no data members or functions. It allows for generic containers of objects to be created, similar to a void pointer in the C programming language.

F. UVM_CALLBACK

The uvm_callback class is the base class for user-defined callback classes. We define an application-specific callback class that extends from this class. In that, we will define one or more virtual methods, called a callback interface, that represent the hooks available for user override.

IV. SIMULATION AND RESULT ANALYSIS

In order to simulate the basic template environment, Questasim simulation tool is used. UVM source files are included to Verilog -System Verilog directory of Questasim tool. This step is must to use base class library of UVM.
In top module, global clk and reset are generated. Each basic components are created using different classes and each System Verilog is listed in separate file which will be compiled. The transcript window is used to analyse the printed topology and different phases in UVM. The different phases includes Build, Connect, End of elaboration, Start of simulator, Report and Check.

V. CONCLUSION

This paper developed basic verification environment to create testbench using UVM methodology. Also different phases involved in UVM are analysed. This basic template can be used in all projects to develop reusable testbench based on System Verilog.

REFERENCES