AUTOMATED REGRESSION ENVIRONMENT FOR IMPROVING DESIGN VERIFICATION PRODUCTIVITY

By
Keshav Rath
(Roll No. 2013104)

SUPERVISOR(S):

EXTERNAL:
Mrs. RAMA KOWSALYA NAMBURI
Senior Project Manager
Canon ISDC, Bangalore

INTERNAL:
Prof. P.N.KONDEKAR
Professor
PDPM IIIT DM Jabalpur

Electronics and Communication Engineering (B.Tech 2013)

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING, JABALPUR

Report

(15th June 2016 – 11th July 2016)

1. INTRODUCTION

During this period, further studies were done in order to get a bigger understanding of the project. The topics studied were:

- VHDL (VHSIC Hardware Description Language)
- UVM (Universal Verification Methodology)

Some description of the topics I have studied are stated in this report.

2. VHDL

VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language) is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits. VHDL can also be used as a general purpose parallel programming language. VHDL is an acronym for Very High Speed Integrated Circuit Hardware Description Language which is a programming language used to describe a logic circuit by function, data flow behavior, or structure.

The concepts studied are:

- A brief history about VHDL
- Design Styles –
    - Bottom–up approach –
    - Top–down approach –
- Abstraction Level –
    - Behavioral Level - This level describes a system by concurrent algorithms (Behavioral). Each algorithm itself is sequential, that means it consists of a set of instructions that are executed one after the other. Functions, Tasks and Always blocks are the main elements.
    - Gate Level – the characteristics of a system are described by logical links and their timing properties

    - Register-Transfer Level - Designs using the Register-Transfer Level specify the characteristics of a circuit by operations and the transfer of data between the registers.
- A simple “hello world” program and a counter design program along with the test bench.
- Code structure – Each program consists of a design entity. Each entity is modeled by an entity declaration and architecture body.
    - Entity declaration - Defines the NAME of the entity and lists the input and output ports. Before the entity declaration libraries and packages are defined.
    - Architecture body - Specifies how the circuit operates and how it is implemented.
- Lexical Elements in VHDL - A lexical item (lexical unit, lexical entry) is a single word, a part of a word, or catena (a chain of words) that forms the basic elements of a language's lexicon (vocabulary). The lexical elements are:
    - Identifiers – User-defined words
    - Keywords – Reserve words
    - Numbers – decimals (default), binary, octal, hexadecimals
    - Characters, Strings and Bit Strings (a sequence of bit values)
- Data Object - A data object is created by an object declaration and has a value and type associated with it. An object can be a constant, variable, signal or a file.
    - Constant – Single value of a given type which cannot be changed during simulation.
    - Variable - A variable can have a single value, just as a constant, but a variable can be updated using a variable assignment statement. The variable is updated without any delay as soon as the statement is executed. Declared inside a process
    - Signal – Declared outside a process. Updates after a time delay.
- Data Type - Each data object has a type associated with it. The type defines the set of values that the object can have and the set of operations that are allowed on it.
    - Data types defined by standard packages – bit, bit vector, Boolean, character, integer, natural, positive, real, severity level, string, time
    - User defined data types – Data types created by users, with type and range specified

- Enumerated types - An enumerated type consists of lists of character literals or identifiers.
- Composite types - It consist of a collection of related data elements. It can be in the form of an array or a record.
    - Array – Multiple elements with same data types
    - Record – Multiple elements with different data types.
- Type conversion – Converting into another data type so as to get different results.
- Attributes – Used to return various types of information about a signal, variable or file. There are 5 types of attributes in VHDL. Some of them are:
    - Scalar
    - Signal
    - Array

- Operator - An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. VHDL supports different classes of operators that operate on signals, variables and constants. They can be categorized into the following types:
    - Logical – and, or, nand, nor, xor, xnor
    - Relational – equality (=), inequality (/=), lesser than (<), greater than (>), lesser than or equal (<=), greater than or equal (>=)
    - Shift – shift logical left (sll), shift logical right (srl), shift left arithmetic (sla), shift right arithmetic (sra), rotate left (rol), rotate right (ror)
    - Addition – addition (+), subtraction (-), concatenation (&)
    - Unary – Identity (+), negation (-)
    - Multiplication – Multiplication (*), Division (/), modulus (mod), remainder (rem)
    - Miscellaneous – exponential (**), absolute value (abs), logical negation (not)Be
- Modelling
    - Behavioral modelling – this type of modelling deals with sequential statements. The types are:
        - Process - A process statement is the main construct in behavioral modeling that allows you to use sequential statements to describe the behavior of a system over time.

- If statement - The if statement executes a sequence of statements whose sequence depends on one or more conditions
- Case statement - The case statement executes one of several sequences of statements, based on the value of a single expression.
- Loop Statements - A loop statement is used to repeatedly execute a sequence of sequential statements. The types are:
  - Basic loop –
  - While loop –
  - For loop –
- Next and exit statement - The next statement skips execution to the next iteration of a loop statement and proceeds with the next iteration. The exit statement skips the rest of the statements, terminating the loop entirely, and continues with the next statement after the exited loop.
- Wait statement – The wait statement will halt a process until an event occurs.
- Null statement – It states that no action will occur

o Dataflow modelling - The dataflow modeling describes a circuit in terms of its function and the flow of data through the circuit. This is different from the structural modeling that describes a circuit in terms of the interconnection of components. The types are:
  - Simple Concurrent signal assignments.
  - Conditional Signal assignments
  - Selected Signal assignments

o Structural modelling - Structural modeling describes a circuit in terms of components and its interconnection.

3. UVM

The Universal Verification Methodology (UVM) is a standardized methodology for verifying integrated circuit designs. UVM is derived mainly from the OVM (Open Verification Methodology) which was, to a large part, based on the eRM (e Reuse Methodology) for the e Verification Language developed by Verisity Design in 2001. The UVM class library brings much automation to the SystemVerilog language such as sequences and data automation features (packing, copy, compare) etc., and unlike the previous methodologies developed independently by the simulator vendors, is an Accellera standard with support from multiple vendors: Aldec, Cadence, Mentor, and Synopsys.

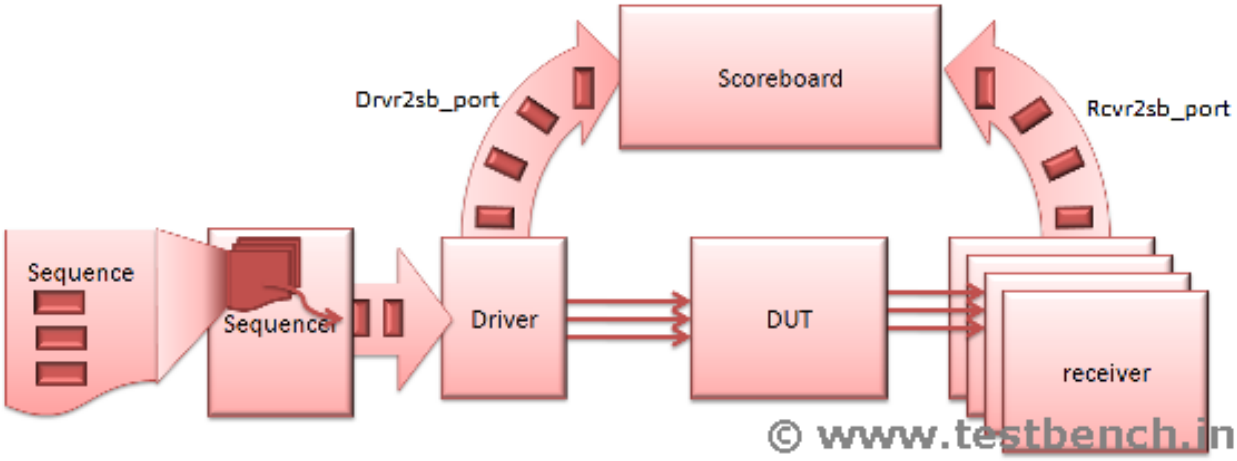

Fig 1 : Basic verification environment

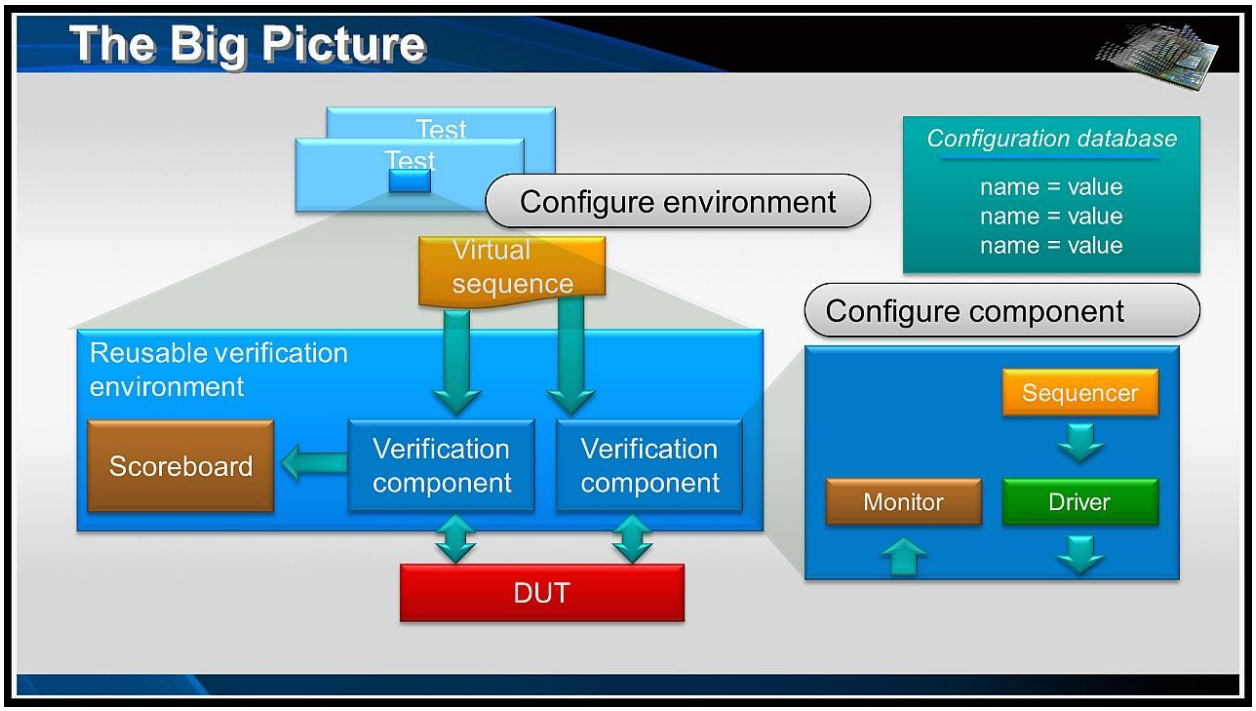

Fig 2: The Bigger Picture of Verification Methodology

The concepts studied in this methodology are:

- A brief introduction of UVM, packages included and installation procedure. The various classes/packages included in the UVM library are:
    - Component classes for building testbench components like generator/driver/monitor etc.
    - Reporting classes for logging,
    - Factory for object substitution.
    - Synchronization classes for managing concurrent process.
    - Policy classes for printing, comparing, recording, packing, and unpacking of uvm_object based classes.
    - TLM Classes for transaction level interface.
    - Sequencer and Sequence classes for generating realistic stimulus.
    - And Macros which can be used for shorthand notation of complex implementation.
- The three basic blocks of UVM testbench
    - uvm_env – Here the environment is created
    - uvm_component – components for verification

    - uvm_test – The name of the testcase file is provided
- UVM reporting: The UVM reporting facility is provided by the class *uvm_report_object*. Through this interface, components issue various messages with different severity levels that occur during simulation. Users can configure what actions are taken and what file/files are output for individual messages from a particular component or for all messages from all components in the environment. This interface consists of reporting methods, actions and configuration.
- UVM transaction: A transaction is data item which is eventually or directly processed by the DUT. The packets, instructions and pixels are data items.
- UVM Configuration: Configuration is a mechanism in UVM that higher level components in a hierarchy can configure the lower level components variables. Without this mechanism, user should access the lower level component using hierarchy paths, which restricts reusability. This mechanism can be used only with components. Sequences and transactions cannot be configured using this mechanism. Higher level component can set the configuration data in level component table. It is the responsibility of the lower level component to get the data from the component table and update the appropriate table.
- UVM Factory: The factory pattern is a well-known object-oriented design pattern. The factory method design pattern defining a separate method for creating the objects. , whose subclasses can then override to specify the derived type of object that will be created. Using this method, objects are constructed dynamically based on the specification type of the object. User can alter the behavior of the pre-build code without modifying the code. From the testcase, user from environment or testcase can replace any object which is at any hierarchy level with the user defined object. There are three basic steps to be followed for using UVM factory.
    - Registration
    - Construction
    - Overriding
- UVM Sequence: A sequence is a series of transaction. User can define the complex stimulus. Sequences can be reused, extended, randomized, and combined sequentially and hierarchically in various ways.

Advantages of UVM sequences:

- Sequences can be reused.
- Stimulus generation is independent of testbench.
- Easy to control the generation of transaction.
- Sequences can be combined sequentially and hierarchically.

A complete sequence generation requires following 4 classes.

- Sequence item.
- Sequence
- Sequencer
- Driver

4. CONCLUSION

Study of these topics and the topics before has given me a better idea of the hardware project.

REFERENCES:

- VHDL Reference Guide, Xilinx, Inc., 1999
- http://www.asic-world.com/vhdl/tutorial.html
- http://esd.cs.ucr.edu/labs/tutorial/
- http://www.testbench.in/
- http://www.testbench.in/UL_00_INDEX.html
- http://www.seas.upenn.edu/~ese171/vhdl/
- http://en.wikipedia.org/