FUNCTIONAL VERIFICATION TEST TIME REDUCTION THROUGH BEHAVIORAL FUNCTIONAL MODEL

By

Lee Chee Keng

A Dissertation submitted for partial fulfilment of the requirements for the degree of

Master of Microelectronic Engineering

June 2014

ACKNOWLEDGEMENTS

Along the progress of completing this project, a number of people have been giving me a helping hand therefore making this project a success. Advices from those people have been really helpful and make this project flows smoothly. I owe a thousand of appreciation to those who have been helping me all this while and I would like to take the chance to thank them here.

First of all, I would like to send me greatest gratitude to my supervisor Dr. Nor Muzlifah Mahyuddin and co-supervisor Dr. Bakhtiar Affendi. With his personal experience and guidance, this project can be completed successfully within the given time. He has been really helpful while I am having a struggle in conducting this project. Honestly, his guidance was an asset for me. It was really my pleasure to be under his supervision.

Not to forget that I would like to thank all my fellow course mates and colleagues that given me a helping hand during my hardship in accomplishing this project. Moreover, they have given me plenty of brilliant ideas and with encouragement that given endless motivation to complete this project.

Finally, I wish to send my love and appreciation to my family for their support throughout my life which have made me walked this far.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AHB	Advanced High Performance Bus
AMBA	Advanced Microprocessors Bus Architecture
ATLM	Arbitrated Transaction Level Modelling
BCA	Bus Cycle Accurate
BFM	Bus Functional Model
CDG	Coverage Directed Test Generation
DSP	Digital Signal Processing
DUT	Design Under Test
FIFO	First In First Out
HDL	Hardware Description Language
IP	Intellectual Property
MPSoC	Multiprocessor System on Chip
OOP	Object Oriented Programming
OVM	Open Verification Methodology
PCI	Peripheral Component Interconnect
RAL	Register Abstraction Level

RTL	Register Transfer Level
SoC	System on Chip
SV	System Verilog
TLM	Transaction Level Modelling
USB	Universal Serial Bus
VCS	Verilog Compiler Simulator
VHDL	Very High Speed Integrated Circuit Hardware Description Language

PENGURANGAN MASA UJIAN VERIFIKASI FUNGSI MELALUI KELAKUAN MODEL BERFUNGSI

ABSTRAK

Proses verifikasi reka bentuk adalah satu langkah penting dalam setiap proses reka bentuk untuk jaminan kualiti. Walau bagaimanapun, proses verifikasi sentiasa berada dalam masalah cerutan dan mengambil 60% daripada keseluruhan tempoh penciptaan reka bentuk. Tahap kesukaran reka bentuk semakin meningkat lalu memanjangkan masa yang diperlukan untuk verifikasi dan kemudiannya membawa kepada kegagalan reka bentuk untuk memasuki pasaran. Salah satu faktor utama yang melambatkan proses verifikasi reka bentuk adalah masa simulasi yang lambat semasa ujian fungsi pra-silikon. Masa simulasi yang lambat dapat dilihat semasa ujian dijalankan untuk verifikasi pra-silikon NAND Harta Intelek (IP). Oleh itu dalam projek ini, model bas berfungsi (BFM) diimplimentasikan untuk NAND IP bagi memendekkan masa simulasi ujian. BFM telah berjaya direka untuk verifikasi NAND IP. Simulasi ujian dengan scenario verifikasi yang sama telah dilaksanakan pada NAND IP dalam persekitaran ujian sedia ada dan verifikasi dalam persekitaran ujian bersama BFM. Keputusannya, BFM didapati memiliki kelakuan dengan tepat berbanding dengan aras pemindahan daftar (RTL) yang sedia ada untuk verifikasi NAND IP. Perbandingan masa simulasi ujian telah menunjukkan melalui persekitaran ujian dengan BFM dengan menggunakan Verilog Compiler Simulator (VCS) telah menunjukkan purata peningkatan yang ketara sebanyak 92.8%. Oleh itu, BFM yang diimplementasi adalah sesuai digunakan untuk verifikasi NAND IP.

FUNCTIONAL VERIFICATION TEST TIME REDUCTION THROUGH BEHAVIORAL FUNCTIONAL MODEL

ABSTRACT

Design verification is an essential step in every design development process for quality assurance. However, the verification portion is the bottleneck in most of design development which takes up 60% of the overall design development period. As the complexity of the design increases, it increases the verification lead time which will then lead to potential failure of the design to meet market on time. One of the key factor in slowing down the design verification flow is the long simulation time during the pre-silicon functional testing. The long test simulation time issue is seen in NAND Intellectual Property (IP) pre-silicon validation. Therefore in this project, a behavioral Bus Functional Model (BFM) is implemented for NAND IP to improve the test simulation time. The BFM has been successfully implemented to validate NAND IP. Simulation of test with similar functional testing scenarios have been exercised on NAND IP in existing verification environment and in verification environment with BFM integrated. As a result, the BFM is found to have behaved accurately comparing with the existing functional Register Transfer Level (RTL) to validate NAND IP. Comparison has also shown the test simulation time through the environment with BFM integrated using Verilog Compiler Simulator (VCS) had shown significant average improvement of 92.8%. Therefore the implemented BFM is justified to be a suitable use on NAND IP validation.

CHAPTER 1

INTRODUCTION

1.1 Introduction

The fast growth of technology and the increasing complexity of system-on-chips (SoCs) and also with the pressure coming from time to market, circuit level simulation is way too slow to be used for functional verification (Gaj et al., 1997). Moreover, most of the cost spent is on the verification process and the verification of the register transfer level (RTL) could take up to 60% work of the entire design cycle (Song, 2007). This is because testing design to ensure a bug free operation is a very complex and effort-consuming task (Lahti & Wilson, 1999).

Today, a single chip could probably have several different Intellectual Properties (IPs) and each block would have specific bus protocols to communicate with each other (Song, 2007). This shows that there will be multiple different bus protocols to control each block of the circuit within the chip which increase the difficulty to verify the result of RTL verification (Becker, 1996).

There are several types of verification methodologies which can be divided into two major groups. The two groups are verification with and without simulation. Formal verification belongs to the verification without simulation group while simulation-based

verification, functional verification, assertion-based verification and symbolic-based verification belong to the other group (Song, 2007).

In doing the verification with simulation, one of the most important parts of the testing process is the simulation time. Under pre-silicon testing environment, simulation of fully functional RTL which contain all the internal structures of actual device will consume much more simulation time and it is less efficient in driving stimulus (Pesavento & Privett, 1999).

One of the methods to reduce the simulation time and maintaining the testing coverage is by implementing a transaction-level model (TLM) where the details of communication of the blocks are separated and modeled (Yeh et al., 2011). This TLM can speed up simulation time and is a design validation alternative at the higher level of abstraction (Cai & Gajski, 2003) and (Velev & Gao, 2011). One of the TLMs that can be used for design validation which is the bus-functional model (BFM). This project will show the reduction of NAND IP pre-silicon validation simulation time using BFM. Figure 1.1 shows the existing testing environment which consists of multiple blocks of RTL and NAND IP RTL while Figure 1.2 shows the proposed validation environment for NAND IP.

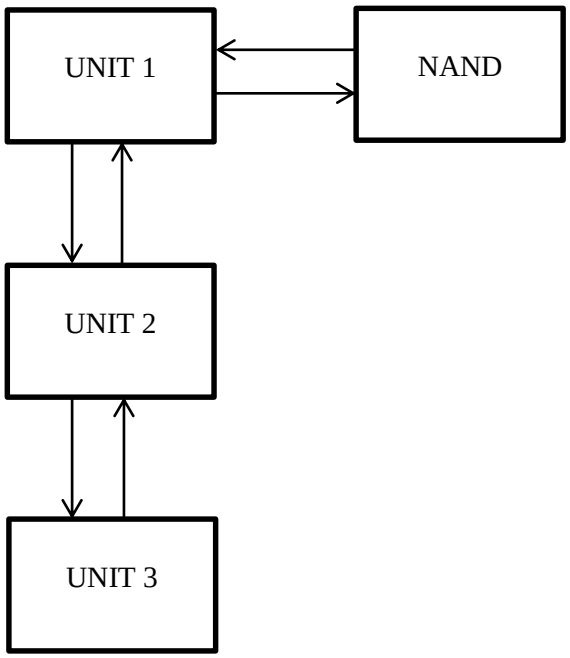

Figure 1.1 Existing testing environment which consists of multiple blocks of RTL and NAND IP RTL

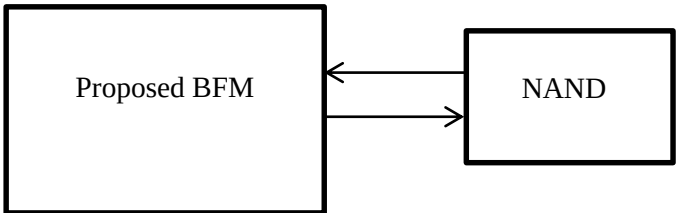

Figure 1.2 Proposed validation environment for NAND IP

1.2 Problem Statement

In producing a healthy design, design validation plays a very important role. The validation process has to be started during the front part of design. It will be too late to

check on the complex design blocks at system level (Pesavento & Privett, 1999). Quality simulation has to be done at the unit level design.

However, the functionality verification of a single unit will require other design block as well during the test simulation. By having multiple design blocks in a simulation will cause the simulator to process more logic. This will lead to the need of more memory used to do the simulation. Hence the simulation of these multiple functional RTLs will consume a very long simulation time (Stehr & Eckmuuller, 2010).

One method to verify the functionality of a particular unit block design with shorter simulation time is by validating that unit design using a BFM. In (Gaj et al., 1997), the simulation time for a circuit using a BFM is found to be shorter. Therefore in this project, a BFM for NAND IP will be developed to shorten the test simulation time for NAND IP validation.

1.3 Objectives

The objectives of this project are as follows:

1. To reduce the NAND IP pre-silicon validation test simulation time.
2. To implement a BFM for NAND and integrate into the testing environment for validation.

1.4 Project Scopes

The scopes of this project are:

1. Design and development of a BFM for NAND IP where the NAND IP is already exist.
2. Integration of the BFM into the existing NAND testing environment.
3. NAND IP validation is performed by using the developed BFM.
4. Evaluation of pre-silicon test simulation time of NAND in the existing testing environment compared to the proposed testing environment.

1.5 Project Contribution

`

The completion of this project has brought to the pre-silicon test time reduction of NAND IP. Large portion of test time is consumed during the NAND model compilation and it is due to the existing of other multiple blocks RTL. With the implementation of the BFM for NAND IP to replace the other RTL blocks that are linked to NAND, the compilation and simulation time can be reduced. More time can be saved by then giving more time to develop more tests to increase the coverage.

1.6 Thesis Outline

This thesis consists of five chapters:

In chapter one, some research background and problems that are aimed to be solved by this project are highlighted here. The objectives and research scopes of this project are stated as well.

Chapter two gives a literature review on several main research areas related to this dissertation, such design validation methodologies, implementation of BFMs of other IPs, and numerous methods used for BFM implementations. Open Verification Methodology (OVM) and System Verilog (SV) are explained briefly so that readers can have a better understanding on the methods to develop the BFM. Information on various BFMs that were implemented by other researchers in the verification process of certain designs are discussed here and also how these BFMs are being developed in different methods by the researchers.

Chapter three consists on the development flow of this research. This project has been divided into three development phases. The first phase of this project is the development of the BFM for NAND IP functional verification. Second phase of the project is the integration of the BFM into the existing NAND IP test environment. Next phase is the validation process of the NAND IP using the BFM. Test simulation is carried out and evaluation is performed between the before and after the usage of the BFM.

In chapter four, several types of tests are carried out and comparison is made between the original testing environment and the one with the BFM integrated. Simulation time and results are compared and discussed.

Finally is chapter five which gives the overall conclusion regarding this research. Possible problems and issues in this research are being discussed in this chapter and some recommendations for future works are also being stated as well.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

After development of a design, a verification environment will be implemented. The main idea of the verification environment is to verify the correctness design under test of the design functionality (Ke et al., 2007). With the complexity of current SoCs design keep increasing and time to market is shorten, the functional verification is a bottleneck (Falconeri et al., 2005) and (Abraham, 1998). Functional verification of such complex design starts with the definition of verification test plan which consists of the set of events that the validation team are expecting from the design (Fine & Ziv, 2003) and then proceed with the implementation of the tests according to the test plan. Hence many ways and methods have been introduced by many research to improve the verification bottleneck. This chapter will discuss a few improvement methods in enhancing design verification process and the most suitable way for this project will be discussed further. In addition, the chosen method to be used in this project will be implemented in OVM approach. A brief explanation of the NAND IP architecture will be given before concluding this chapter.

2.2 Methods of Enhancing Design Verification Time

The functional verification has been the bottleneck for most of the design development flow. (Shen & Abraham, 2000) has mentioned that the current validation capabilities have to be improved to sustain with the rapid growth of semiconductor industry. There are a number of ways or efforts that have been proposed and implemented to improve the verification methodology and environment for certain design.

2.2.1 Coverage Directed Test Generation using Bayesian Networks for Functional Verification

A new way for generating coverage test is proposed by (Fine & Ziv, 2003). Coverage events or called as testing requirements is a major part in a verification plan of certain design. Coverage directed test generation (CDG) is defined as a technique to automate the feedback from coverage analysis to test generation. CDG can help to improve the coverage progress rate, reaching uncovered tasks and have multiple ways to reach given coverage tasks. Figure 2.1 shows the basic idea of verification process with CDG. It can be seen from Figure 2.1 that the tests are random generated through CDG will provide a coverage analysis which then will be feedback to the test generator.

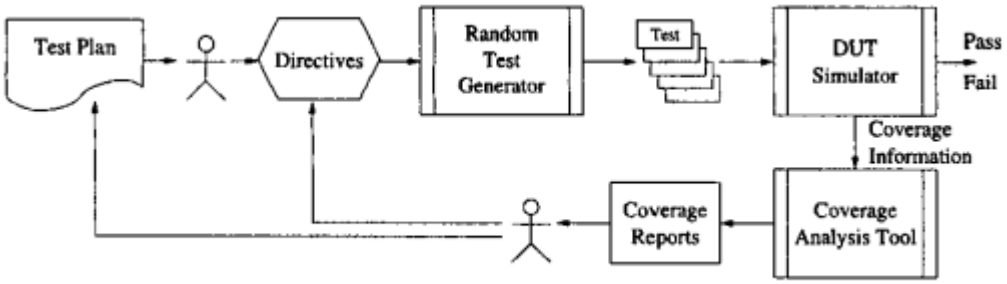

Figure 2.1 Verification process with CDG (Fine & Ziv, 2003)

The main goal of the approach is to model the relationship between the coverage information and the directives to the test generator using Bayesian networks. Bayesian network is a directed graph whose nodes are random variables and whose edges represent direct dependency between their sink and source nodes (Heckerman, 1998). A set of parameters representing its conditional probability given the state of its parent are linked to each node of the Bayesian network. In short, coverage directed test generation process is done in two steps. The first step is the learning of the Bayesian network parameters that models the relationship of coverage information and test directives through a training set. Then proceed to the second step where Bayesian network is used to provide most probable directives that lead to a given coverage. Figure 2.2 illustrates a simple Bayesian network which includes a small part of CDG setup. The network shows the relationship between the directives that affect the type of command generated (cp_cmd_type), active cores (cp_core_enable), coverage attribute command (cmd), its response (resp) and the core generated (core).

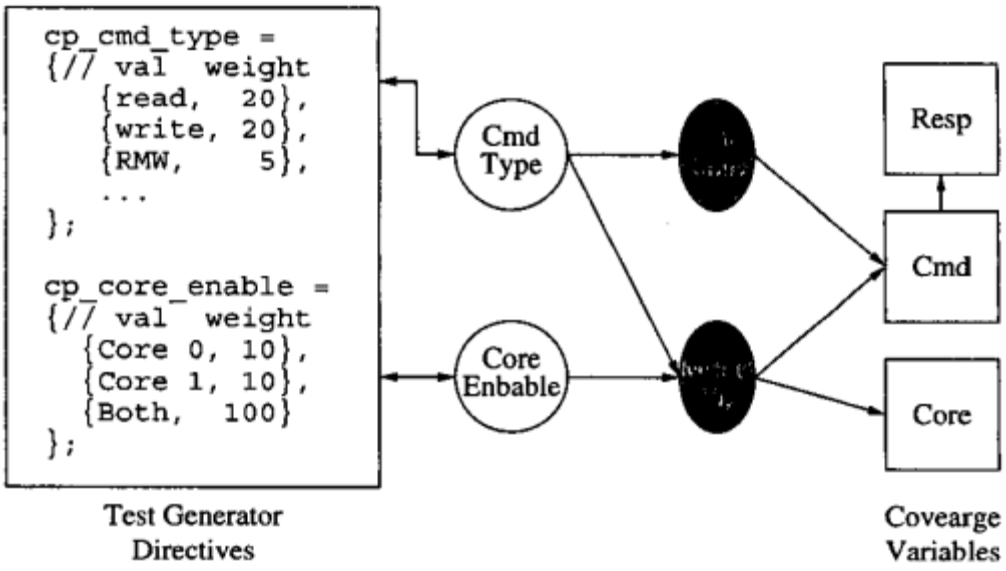

Figure 2.2 Bayesian Network of CDG (Fine & Ziv, 2003)

(Fine & Ziv, 2003) has concluded that CDG using Bayesian networks shows that hard coverage cases can be reached easier and also reduced coverage test development time. However it did not show any improvement on test simulation time which is the main focus of this project.

2.2.2 Matlab and Simulink in a SystemC Verification Environment

A verification framework which is based on SystemC verification standard that uses Simulink and also MATLAB to speed up the testbench development is proposed by (Boland et al., 2005). The MATLAB and SystemC verification framework can be seen in Figure 2.3. (Boland et al., 2005) put the focus on digital signal processing (DSP) applications verification using algorithmic modeling in MATLAB and Simulink environment. The verification specification is first written and then the algorithm is

implemented with MATLAB and Simulink. A variety of algorithm optimization can be done at this stage. The result of this step will then be the main reference for the system level verification modeling with SystemC and C++ languages.

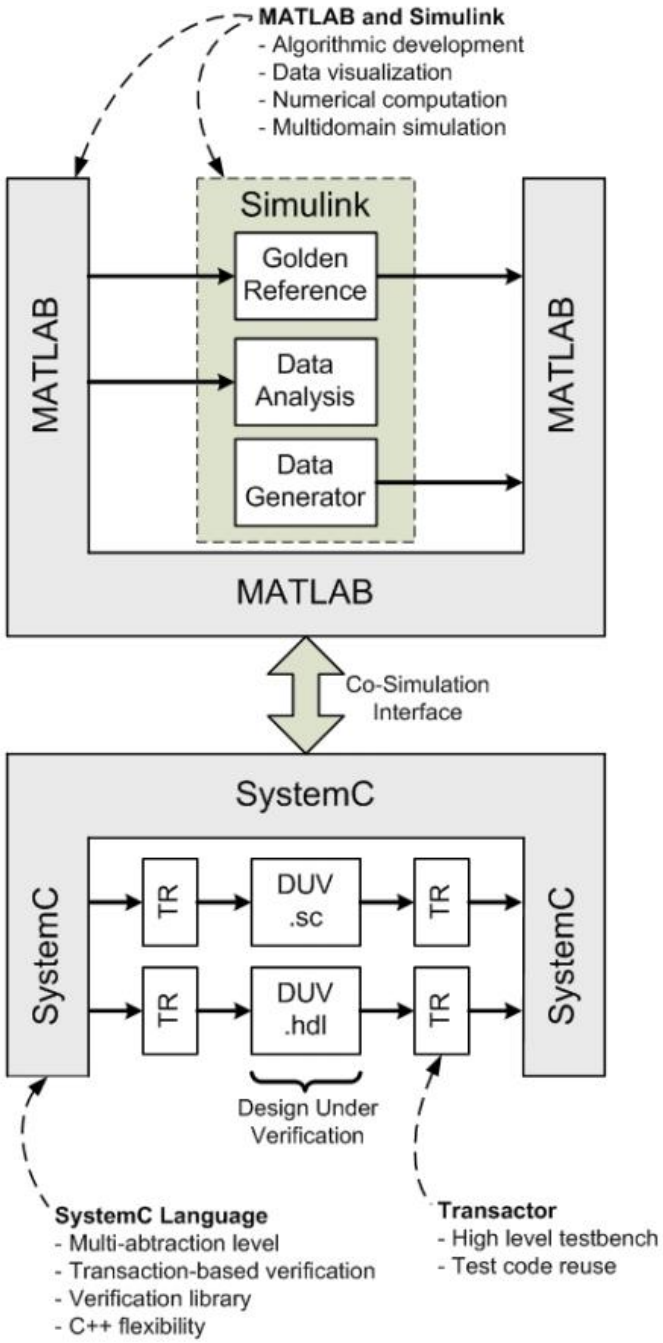

Figure 2.3 MATLAB and SystemC Verification Framework (Boland et al., 2005)

The proposed framework by (Boland et al., 2005) has shown that the hardware verification bottleneck has been greatly improved where a more complete testbench can be developed in a shorter period of time than with the traditional HDL. With the framework, verification environment can be connected to multiple levels of abstraction and verification can be started at early stage of development cycle. However, there are no improvement on test coverage and also no reduction on test simulation time by using the proposed framework.

2.2.3 Common Reusable Verification Environment for Bus Cycle Accurate (BCA) Model and RTL

The common verification methodology and environment can be used for RTL and BCA models are shown by (Falconeri et al., 2005). BCA model is one type of BFM (Cai & Gajski, 2003) and the fast simulation of BCA model compared to RTL model allows fast finding on optimized configuration in terms of bandwidth, area and power consumption (Falconeri et al., 2005) with the BCA model functionality constraints have to be similar as the RTL model.

Since BCA and RTL models has the similar functionality, therefore the requirements for functional verification have to be similar as well. (Falconeri et al., 2005) proposed to use a common verification environment for both BCA and RTL model and it can save effort by not duplicating work in developing the verification environment for the

two different models. Figure 2.4 shows the complete verification flow from functional specification to bus accurate comparison.

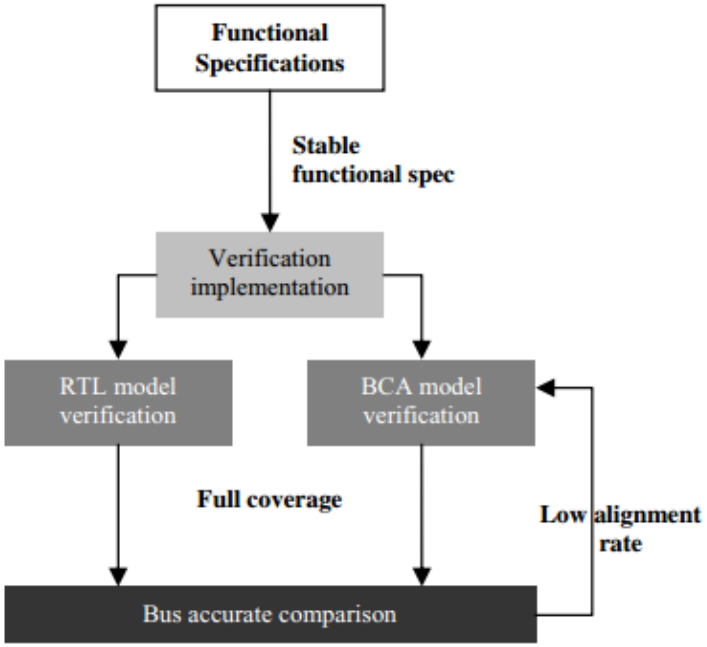

Figure 2.4 Complete verification flow (Falconeri et al., 2005)

Having the common verification environment is not a new idea (Vaumorin & Romanteau, 2004) and this strategy has shown high gain in terms of development time and improved verification accuracy. It is also shown that the simulation time with the BCA model is faster compare to the simulation time of the RTL model.

2.2.4 BFM in Verification Environment

As mentioned by (Yu et al., 2004) and (Song, 2007), time to write testbench can be reduced and functional coverage can be increased using the system level verification methodology. Modern design flow is moving at a higher pace which made traditional simulation-based verification method cannot keep track with the flow (Song et al., 2005). A system level function will be partitioned into several parts, and be implemented at the same time (Sayinta et al., 2003).

BFM is also one of the transaction-based verification methodology strategy to improve functional verification efficiency of RTL using simulation (Labs et al., 2000). BFM basically is a model of bus interface of certain design units (Pesavento & Privett, 1999). The bus interface signals of interconnect between the DUT and BFM will be captured by the BFM. Behavior of the BFM data can be scheduled and captured in a relax manner so that computation of data can be grouped and incremented in chunks with time rather than on a per-transaction basis (Pasricha et al., 2010). This relax scheduling permits the capture of only required data details which means reduction in details of data captured. Correspondingly it will reduce the modeling time and also improve on the simulation speed. BFM in general needs about one-fifth to one-tenth of the effort required for RTL modeling and BFM is one hundred to five hundred times faster than RTL simulation (Pasricha et al., 2010).

(Song, 2007) and (Falconeri et al., 2005) uses the BFM as one of the verification tools used in the system level assertion based verification environment. The BFM used is for the Peripheral Component Interconnect (PCI). As mentioned in (Yu et al., 2004), with

the usage of BFM and also other verification tools in the verification environment, the design under test (DUT) can be tested completely in more complex situations which is useful to validate the robustness of the DUT protocol. In addition, the test simulation time also has been shown to be reduced with the usage of BFM in the test environment (Song, 2007).

In (Schirner & Rainer, 2006), an abstract communication modelling study had been done on Advanced Microprocessors Bus Architecture (AMBA) Advanced High Performance Bus (AHB). Three models were implemented: BFM, arbitrated transaction level model (ATLM) and transaction level model (TLM). BFM shows the best accuracy in both operating modes of AHB while TLM and ATLM shows errors in one of the operating mode of AHB. It is also shown that all three models had improvement in test simulation time.

Implementation of a BFM for the Pentium Processor is proposed by (Hunt et al., 1993). The BFM that were implemented had provided an accurate representation and can be represented in behavioral simulation which is useful for Pentium processor based platforms and system validation and design. While in (Petkov et al., 2005), BFM of a Multiprocessor System on Chip (MPSoC) had been developed in accelerating the hardware or software prototype generation for MPSoC. By using the BFM, (Petkov et al., 2005) shows a time reduction in systematic design process and software integration.

Implementation of USB BFM has been shown in (Chonnad & Needamangalam, 2000). The USB BFM implemented is inherently reusable and it is easier to maintain as it contains the Object Oriented Programming (OOP) features. Randomization of tests has increased the functional coverage. This is possible if and only the BFM is coded using the

modern verification language that supports randomization and the implementation of the BFM has shown reduction in simulation time (Chonnad & Needamangalam, 2000). Figure 2.5 shows the typical architecture for verification of bus interfaces. The BFM in Figure 2.5 will be connected to the device under test and the connection bus will be monitored by a bus monitor.

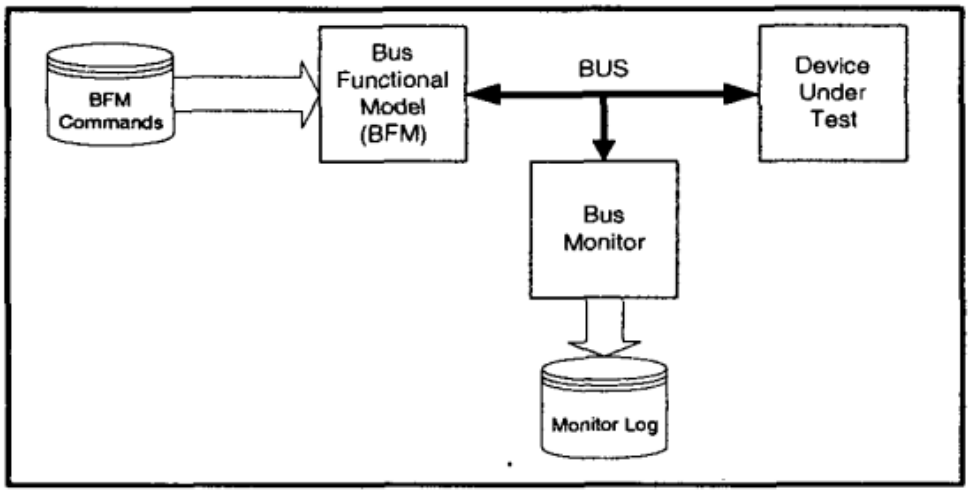

Figure 2.5 Typical architecture for verification of bus interfaces (Chonnad & Needamangalam, 2000)

2.3 BFM Implementation

Usage of BFM is found to be the most suitable in reducing test simulation time and therefore BFM is implemented. There are multiple methods of coding the BFM in the validation environment. BFM can be coded using many types of HDLs such as VHDL, Verilog, System Verilog, C++, System C and etc. The purpose of BFM implementation not only will improve the verification flow in term of reduced simulation time but also

capability to debug, randomization testing, and overall of improved total validation time (Sudhish et al., 2011). The focus in this project is on validation simulation time reduction hence only the key features of HDL on verification process will give an advantage. Only VHDL, Verilog and System Verilog language capabilities will be discussed in this chapter.

2.3.1 Very High Speed Integrated Circuit Hardware Description Language (VHDL)

VHDL is a general purpose digital design language which is supported by multiple verification and synthesis tools. (Smith et al., 1996) has discussed on the comparison on VHDL and Verilog and it is shown that VHDL can do concurrent procedure calls and also design reusability where functions and tasks can be placed in a package to be reused. VHDL also support user-defined types and enumerated types (Maginot, 1992) which is suitable in defining verification data types.

One disadvantages of using VHDL is that it has no simulation control or monitoring capabilities (Bailey, 2003) where this capability is an important feature in verification process and hence VHDL is very dependent on tool environment for debugging activities. VHDL also does not support name based events which is useful in validation. Class inheritance feature (reusable class module) is also not supported by VHDL.

2.3.2 Verilog

Verilog is a HDL used design and verification of digital circuit design at the RTL. (Gordon, 1995) mentioned that Verilog is widely used to model the behavior of digital systems building blocks to complete systems. Verilog can support continuous assignments with delay which makes the verification more realistic and it also has the blocking and non-blocking statements which is able to control the transport delay of certain behavior. Concurrent tasks and functions are also supported by Verilog. A set of basic simulation control capabilities or the system tasks are defined within Verilog.

However, Verilog has its disadvantages too. It does not support user defined data types and enumerated types unlike VHDL and System Verilog. This will be a limitation in improving the validation process. Interface abstraction is also not supported which reduces flexibility in port mapping. In general, (Bailey, 2003) stated that Verilog has limited verification targeted capabilities.

2.3.3 System Verilog

Parenting from Verilog, SV benefited its advantages and adding user defined data types (Bailey, 2003) as well as strong data typing capabilities (Fitzpatrick, 2004). SV is backward compatible with Verilog by retaining weak data typing for the built in Verilog types (Bailey, 2003). The OOP feature of SV can greatly enhance the reusability of the verification environment components (Ke et al., 2007). There are a few more verification features which SV can provide such as dynamic memory, constrained random data

generation, dynamic processes and also assertions to improve the quality of verification. It is concluded in (Fitzpatrick, 2004) that SV is built on the Verilog language with many features were derived from proven VHDL features and extended to be more powerful. Figure 2.7, Figure 2.8, and Figure 2.9 show the overall comparison of VHDL, Verilog and System Verilog.

2.3.4 Open Verification Methodology (OVM)

In (Cadenas & Todorovich, 2009), OVM is described as a framework for functional verification of digital hardware using System Verilog in simulating environment. OVM is defined as a library of verification components (Glasser, 2009). OVM offers TLM interfaces, a class factory for dynamic selection of instantiated object type, verification components classes such as drivers, monitors, and scoreboards and also mechanism for the construction of complex stimulus for a DUT using sequencers and sequences (Poikela et al., 2012). The library also includes its own first in first out (FIFO) which can be directly connected to the TLM ports. These OVM components are written as System Verilog classes.

(Cadenas & Todorovich, 2009) has mentioned that the idea of OVM is to replace the conventional HDL approach in testbench writing and by OVM, it is a more robust methodology based on reusable verification environment. Figure 2.6 shows the simple OVM verification environment.

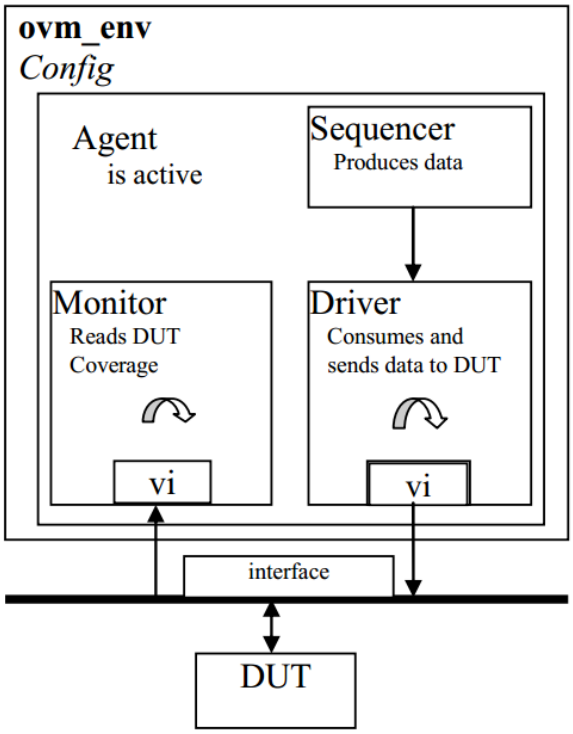

Figure 2.6 Simple OVM verification environment (Cadenas & Todorovich, 2009)

	VHDL	Verilog (2001)	SystemVerilog
Strong typing	**Yes**	**No** • Bit • bit-vector • wire • reg) • unsigned • **signed** • integer • real • String in certain contexts only	**Partial** Not strongly typed in areas backward compatible with Verilog **Yes** Enhanced type system is strongly typed (but not as strong as VHDL)
User-defined types	**Yes**	**No**	**Yes**
Dynamic memory allocation (pointer types)	**Yes**	**No**	**Partial** Class objects can be dynamically created/destroyed, but via handles ("safe pointers")
Physical types	**Yes**	**No**	**No**
Named events	**No**	**Yes**	**Yes**
Enumerated types (FSM modeling)	**Yes**	**No**	**Yes**
Records/structs	**Yes**	**No**	**Yes**
Variant/unions	**No**	**No**	**Yes**
Associative/sparse arrays	**Partial** (But can be modeled using access types)	**No**	**Yes**
Class/inheritance	**No**	**No**	**Yes** (single inheritance)
Data packing	**No**	**No**	**Yes**
Bit (vector) / integer equivalence	**Partial** Not built-in but standard package supports	**Yes**	**Yes**
User defined signal/net resolution	**Yes**	**No**	**No**
Subprograms (procedural)	**Yes** Function & procedure always automatic	**Yes** Static and **automatic** functions and tasks	**Yes** Same as Verilog plus void functions (procedures)
Subprograms (concurrent) aka tasks	**Yes** Concurrent procedure calls	**Yes** Static tasks	**Yes** Static tasks
Methods	**No**	**No**	**Yes** (goes hand-in-hand with classes)
Separate packaging	**Yes** Packages	**Yes** Include files	**Yes** Include files

Figure 2.7 Feature by feature comparison between VHDL, Verilog and System Verilog (Bailey, 2003)

	VHDL	Verilog (2001)	SystemVerilog
Other hierarchy	**Yes** Separate entity / architecture (Interface / implementation)	**No**	**Yes** Programs, Clocking domains, Interfaces
All-read sensitivity	**No**	**Yes** @(*)	**Yes** Same as Verilog. Plus: always_comb
Reactive region processes	**Yes** Postponed processes	**No**	**Yes** Programs, Clocking domains, Final blocks
Dynamic process creation/deletion	**No**	**Yes** Fork/join. Block/task disable.	**Yes** Same as Verilog.
Conditional statements	**Yes** • If-then-else/elsif (priority) • Case (mux) • Selected assign (mux) • Conditional assign (priority) • No "don't care" matching capability	**Yes** • if-else (priority) • case (mux) • casex (mux) • ?: (conditional used in concurrent assignments)	**Yes** Same as Verilog. Adds priority and unique keywords to infer priority encoding/mux implementation
Iteration	**Yes** • Loop • while-loop • for-loop • exit • next Can name the loop to exit or continue with next	**Yes** • repeat • for • while	**Yes** Same as Verilog, Plus: • do-while • break • continue Only closest enclosing loop can be break or continue
Operators & expressions	**Yes** All expected: • arithmetic • logical • bit-wise • shift • concatenation Overloadable (polymorphism). No unary reduction. No logical scalar/vector.	**Yes** All expected: • arithmetic • logical • bit-wise • shift • concatenation • unary reduction • logical scalar/vector • case (in)equality. • conditional (?:) No rotate left/right	**Yes** Same as Verilog. Plus: • wild (in)equality • increment • decrement • assignment (+=, -=, |=, etc.) No rotate left/right
Gate level modeling	**Yes** VITAL. Very good FPGA library support.	**Yes** Builtin primitives. UDPs. Better availability of ASIC library support	**Yes** Same as Verilog. Except, library support yet to be qualified as vendors won't assume Verilog sign-off = SystemVerilog sign-off
Interface abstraction	**Partial** Component abstracts interface from specific module. Two layer binding allows flexibility in generic/port mapping.	**No**	**Yes** Interfaces are a separate construct in language. Supports multiple abstraction level and eases interface reuse. Can reduce coding.

Figure 2.8 Feature by feature comparison between VHDL, Verilog and System Verilog (Bailey, 2003)

	VHDL	Verilog (2001)	SystemVerilog
Configuration & Binding	**Yes** Control of instance or component binding to entity. Incremental (re)binding of generics and ports.	**Partial** **Control of module to instance binding.**	**Partial** Same as Verilog.
Conditional & iterative generation	**Yes** • If (conditional) • For (iterative)	**Yes** • **If** • **if-else (mutually exclusive)** • **case** • **for**	**Yes** Same as Verilog.
Attributes	**Yes** Attributes are typed. Attribute values can be specified. Attribute values can be referenced. Anything labeled with a name can be attributed. Groups allow attributes to relate two or more named entities in the design.	**Partial** **Not-typed.** **Can be placed virtually anywhere.** **What is attributed is determined by lexical proximity.** **Attribute values cannot be referenced.**	**Partial** Same as Verilog.
Verification targeted capabilities	**Partial** • Access types • Recursive subprograms • Extensive File I/O • Postponed processes • Standard package for random number generation	Limited • File I/O • Random number generation • Recursive subprograms • Fork/join	**Yes** Same as Verilog. Plus: • Random and constrained random value generation • Programs • Clocking domains • Associative arrays • Semaphores • Mailboxes • Classes
Assertions	**Partial** • Combinatorial (Boolean) assertions • User-defined severity and message control	**No**	**Yes** • Combinatorial and sequential (concurrent) assertions. • Sequence (temporal) expression. • Sequence-local variables. • User-defined severity and message control. • API extensions for assertions and coverage information for assertions
Foreign interfaces	Limited • Standard 'Foreign attribute • VhPI defined, but not yet standardized	**Yes** Standard C API (tf, acc, vpi)	**Yes** Same as Verilog. Plus: • Extensions to API for assertions and coverage • Direct C language interface

Figure 2.9 Feature by feature comparison between VHDL, Verilog and System Verilog (Bailey, 2003)