UVM based STBUS Verification IP for verifying SoC

Student Name: Pranay Samanta

IIIT-D-MTech-ECE-VLSI & Embedded System-12-13

Indraprastha Institute of Information Technology
New Delhi

Under the Supervision of
Dr. Sujay Deb
Mr. Piyush Kumar Gupta

Submitted in partial fulfillment of the requirement
for the Degree of M.Tech. in Electronics & Communication Engineering
With Specialization in VLSI & Embedded System

This work is performed in and funded by STMicroelectronics, Greater Noida

Student's Declaration

I hereby declare that the work presented in the report entitled **"UVM based STBUS Verification IP for verifying SoC"** submitted by me for the partial fulfillment of the requirements for the degree of *Masters of Technology* in *Electronics & Communication Engineering with specialization in VLSI and Embedded Systems* at Indraprastha Institute of Information Technology, Delhi, is an authentic record of my work carried out under guidance of **Dr. Sujay Deb** of Indraprastha Institute of Information Technology, Delhi and **Mr. Piyush Kumar Gupta** of ST Microelectronics. Due acknowledgments have been given in the report to all material used. This work has not been submitted anywhere else for the reward of any other degree.

.....

Pranay Samanta **Place & Date: .....**

Certificate

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

.....

Dr. Sujay Deb **Place & Date: .....**

.....

Mr. Piyush Kumar Gupta **Place & Date: .....**

Dedicated to

To my beloved parents and a lovely lady

List of Abbreviation

VLSI	Very large scale integration
SOC	System-on-chip
IP	Intellectual Property
EEPROM	Electrically Erasable Programmable ROM
OVM	Open Verification Methodology
RAM	Random Access Memory
UART	Universal Asynchronous Receiver Transmitter
SPI	Serial Peripheral Interface
GPIO	General Peripheral Input Output
DUT	Design under test
VIP	Verification Intellectual Property
UVM	Universal Verification Methodology
eRM	e Reuse Methodology
AVM	Advanced Verification Methodology
TLM	Transaction-Level Methodology
URM	Universal Reuse Methodology
UVC	Universal Verification Component
OVM	Open Verification Methodology
SV	System Verilog
PCI	Peripheral Component Interconnect
AMBA	Advanced Microcontroller Bus Architecture
VCI	Virtual Component Interface
MPEG	Moving Picture Experts Group
GOOOLO	Globally out of order locally ordered
PSL	Property Specification Language
RTL	Register-transfer level
IDP	Image Data pipeline

Abstract

The ever increasing advances in the integrated circuit technology made it possible for electronic system designers to assemble complete systems-on-chips (SoC). As these SoCs have been used in computer, graphics, and networking hardware systems, the complexity of functionality within them have rapidly increased. At the same time shrinking time to market leaves little room for errors in the design. Hence functional verification has become one of the major tasks in committing chips to fabrication.

Just as designs are pushing towards reusable environment so must the verification environment. As verification itself takes 70% of the design time, the need of standalone, pre-verified verification infrastructure is arisen so that verification does not become the bottleneck for the designers. The Verification Intellectual Property (Verification IP/VIP) which can be easily plugged in the simulation-based tests, is an important component of such infrastructure.

In this dissertation the modeling of VIP of STBUS protocol has been presented, and in the process STBUS protocol has been also verified. The use of STBUS VIP has been shown by modeling a verification environment for verifying a SoC. The coverage analysis has been done to check how much jump-start the verification IP can produce in achieving the coverage goal quickly.

Acknowledgments

Acknowledgment is not a mere formality but a genuine opportunity to express the sincere thanks to all those; without whose active support and encouragement this thesis would not have been successful.

I express my deep sense of regards and indebtedness to my advisers Dr. Sujay Deb and Mr. Piyush Kumar Gupta for their valuable guidance, continuous encouragement and wholehearted support, which are of immense help to me in completing this thesis. In spite of their hectic schedule they have always extended their help and invaluable suggestions.

I express my thanks to Prof. R.N.Biswas for giving me the excellent opportunity to work in a reputed company like STMicroelectronics to do industrial research work there.

I express my thanks to my committee members and to all the faculty members and the staff of the Department of Electronics and Communication Engineering for their continuous help and support.

My sincere thanks to Mr. Deepak Chauhan for his continuous encouragement, extremely valuable insights, interesting discussion and explanation of key concepts in Verification.

Words are not enough to express my indebtedness and gratitude toward my parents, to them I owe every success and achievements of my life. Their constant support and encouragement under all odds has brought me where I stand today. Also, my love and respect toward my elder brother Pratip for always believing in me.

Finally, I thank to my friends Supratim Das, Rohan Sinha and Srikrishna Acharya Ballore for their ceaseless effort, constant encouragement, the endless good times, love, cheerful encouragements and valuable criticism during the making of this dissertation.

> "A mediocre person tells. A good person explains. A superior person demonstrates. A great person inspires others to see for themselves" $\sim Harvey Mackay$

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 System On Chip

The semiconductor industry has continued to make impressive improvements in the achievable density of very large-scale integrated (VLSI) circuits [1]. In order to keep pace with the levels of integration available, design engineers have developed new methodologies and techniques to manage the increased complexity inherent in these large chips [2]. One such emerging methodology is system-on-chip (SoC) design, where the engineers use pre-designed and pre-verified blocks which are called intellectual property (IP) blocks. These blocks can be obtained from internal sources, or third parties, and combined on a single chip. So, System-on-Chip (SoC) is an integrated circuit that integrates all components of an electronic system into a single integrated circuit. Figure 1.1 depicts a system level representation of a typical SoC. It can be observed from Figure 1.1 that a system-on-chip contains various intellectual property (IP) elements such as aBone (AC arbiter), Debug interface with direct memory control, Interrupt extender, EEPROM memory controller (I2C) with boot support, User memory mapped RAM block, UART, SPI master interface, I2C master/slave interface, GPIO interface, Hardware Watch dog, System Elapsed Time (SCET) timer interface. In the SoC, IPs such as processors, DSPs and other processing engines execute the associated

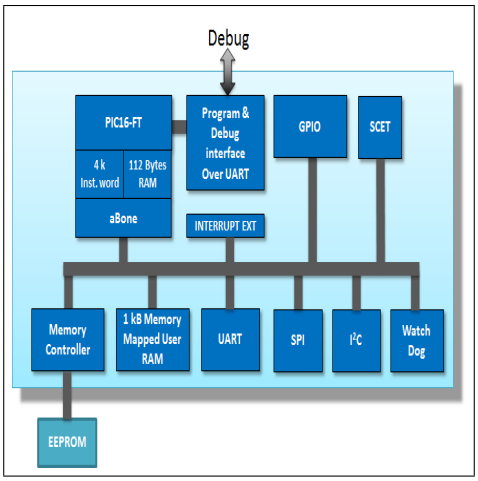

Figure 1.1: A simple SoC

software to provide the intended functionality. Thus, both the hardware (IPs) and the associated software are an integral part of a system-on-chip.

1.2 Motivation

The motivation of this work can be classified as the following:

1. **Function Verification:**

SoCs (System on Chips) are complex designs with heterogeneous modules (CPU, memory, etc.) integrated in them. As like the design of the SoC, verification is also an important stages in designing an SoC [3]. Verification is the process of checking if the designer implemented the correct architectural specification or not. It checks whether the designer has implemented what he has intended to make. Figure 1.2 pictorially shows what functional verification is.

To verify a design the following steps are taken:

(i) a test-plan or verification plan is created that identifies the conditions/features to be verified.

(ii) a test-process to generate the stimuli to verify the conditions.

(iii) a test-bench that applies the stimuli and monitors the output from the design under test (DUT).

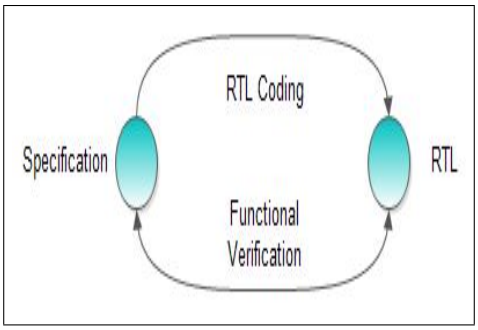

Figure 1.2: Function Verification

Functional verification is very important before committing the design into fabrication. Because improper verification can result in a silicon respin. Figure 1.3 shows the different reason of silicon respin and function verification being the main reason of the silicon respin. Respin can cost a lot as well as company losses the market to its' rivals. Figure 1.4 shows an estimated figure of the respin.

2. **Reusable Verification Environment:**

Due to the immense improvement in the electronics circuit design, the designers now re-use the IP cores to design the SoCs. As the design itself is moving towards a reusable environment, the verification environment should also move toward a reusable environment, so that verification does not become the bottleneck of the design which consumes upto 70% of the total design time. So a reusable, stand-alone verification environment is needed to decrease the time needed to verification as it already consumes 70% of design time.

3. **Verifying Interconnect protocol:**

In complex designs of SoC, interconnection networks are becoming more and more important [4]. Currently, on-chip interconnection networks are mostly implemented using buses. For SoC applications, design reuse becomes easier if standard internal connection buses are used for interconnecting components of the design. Design teams developing modules intended for future reuse can design interfaces for the standard bus around their particular modules. This allows future designers to slot the reuse module into their new design simply, which is also based around the same standard bus [5]. The first step of verifying system-on-chip designs is to verify the standard bus

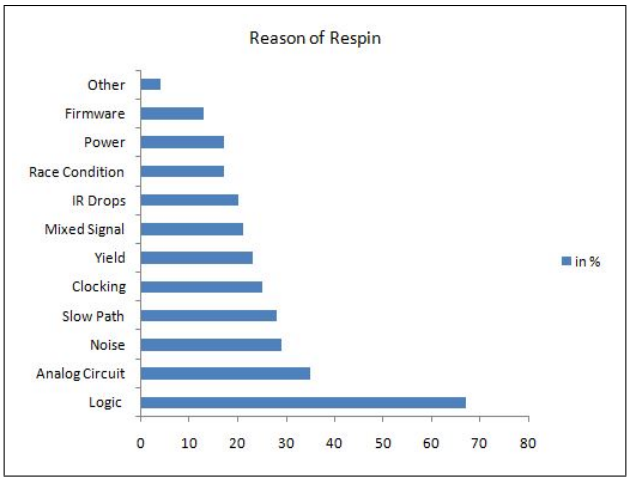

Figure 1.3: Reason of respin
Source: Collett International Research (Apr12)

interconnecting IP cores [6]. So verification of the bus in the SoC is an important aspect of the functional verification of the SoC.

1.3 Problem Statement

- Develop a reusable, stand-alone, pre-verified verification architecture which can be easily plug into the test and verify the design with the help of it. To do so, the development of VIP of the STBUS protocol has been done. VIP can be reused in the functional verification of IP/SoC which uses STBUS as bus protocol. By verifying the developed VIP it can be made sure that STBUS protocol also gets verified as the VIP is nothing but a mimic of the protocol.

- Design a verification testbench to use the designed VIP to verify a SoC and check the coverage result to analyze how much jump-start the designed VIP can produce in achieving coverage goal. It will result in decrease of verification time which is a bottleneck of the design cycle.

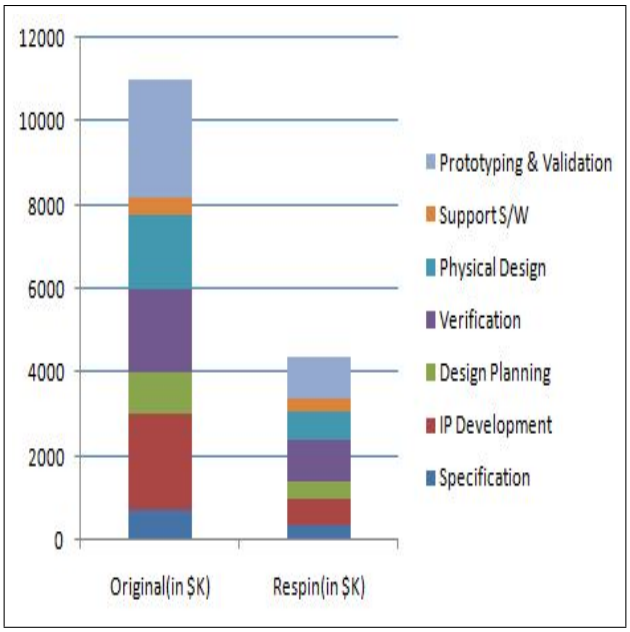

Figure 1.4: Cost of respin
Source: International Business Strategies, 2012

1.4 Literature Survey

Before introduction of UVM, the industry has used several methodologies for hardware verification and making VIPs. In [7], eRM has been described as the first verification library in the industry introduced by Verisity (now Cadence Design Systems, Inc.) [8]. It includes packaging guidelines, architecture requirements, sequences and virtual sequences, messaging facilities, scoreboard examples etc. In [9], Advanced Verification Methodology (AVM) of Mentor Graphics [10] based verification testbench is shown where OSCI SytemC Transaction-Level Methodology (TLM) standard is used. It leaves open higher-level verification needs such as test classes, complex stimuli generation, configuration etc. In [9], Universal Reuse Methodology (URM) is used to make Universal Verification Component (UVC) for verifying large-gate count, IP based SoCs. URM is developed in System Verilog language. The use of System Verilog language helped the user to use constrained variable, randomization, and interface, virtual interface etc. that facilities with C-like control structures and data type features [11]. In [9], a testbench based on Open Verification Methodology (OVM) is

shown too. OVM is supported by both Cadence and Mentor and it is a methodology which has blends of both URM and AVM. UVM is the latest methodology created by Accellera [12]. Verification environment written by UVM methodology can be simulated by any of the vendor simulator namely, VCS of Synopsys, Incisive of Cadence and Questa of Mentor Graphics.
Verification of bus protocol such as PCI local bus has been done widely in various works [13] [14]. But PCI does not support complex data transfer like pipelining transfer or out-of-order transfer. More advanced bus protocol verification like AMBA has been done in [6] which supports burst transfer. In this work, we present a verification framework for SoCs designed with STBUS using UVM based STBUS VIP.

1.5 Chapter Organization

In the second chapter, an overview of the STBUS protocol and UVM verification methodology has been given. The description of the designing of VIP has been narrated in the third chapter. Fourth chapter pictures about experimental environments which has been used to verify the VIP and then RTL with the help of verified VIP. Then the result and the analysis of the result have been given in the next chapter. The dissertation has been concluded in the last chapter.

Chapter 2

Overview of STBUS protocol and Universal Verification Methodology

Universal Verification Methodology and System Verilog language have been used for the implementation of the VIP. In first section in this chapter, a short description about the STBUS is stated. The features of different STBUS protocol, which are supported in the implemented VIP are discussed. A general idea of Universal Verification Methodology (UVM) is given in the second section.

2.1 STBUS Protocol

The STBUS is a set of protocols, interfaces, primitives and architectures specifying an interconnect subsystem. The STBUS is the result of the evolution of the interconnect subsystem developed for microcontroller dedicated to consumer application, such as set top boxes, ATM networks, digital still cameras and others. Such an interconnect was born from the accumulation of ideas converging from different sources, such as the transputer (ST20), the Chameleon program (ST40, ST50), MPEG video processing and VCI (Virtual Component Interface) organization [15].

2.1.1 Components

STBUS can be taken as the working protocol of two types of IP.

- Initiator: Initiator or master IPs are the IPs which generates transaction or traffic towards the interconnect.
- Target: Target or slave IPs are the IPs which are the resources of system like memory or peripheral. This IPs are used by the initiator and generally responds to the transaction initiator sends towards the interconnect.

Talking about the different types of data traffic STBUS can have the following types:

- Cell: This is the shortest data traffic which can be sent by initiator in one cycle of time. The maximum size of the cell depends upon the width of the interconnect. Example: If the interconnect width is 32 bit the maximum cell size would be 32 bit.
- Packet: It is a collection of cells. The number of cells in a packet depends on the operation size and the width of the interconnect.
- Chunk: Chunk is the collection of packets. It is complex data structure. The packets in the chunk are linked by the lck signal.
- Message: Message is the collection of chunks. It is also a kind of complex data structure. The chunks in the message are linked by the msg signal.

2.1.2 Protocols

3 different types of protocols exist in STBUS.

- T1 protocol: It is the simplest of three protocols, and is intended to be used for peripherals registers access. No pipeline applies. Load/store on 1/2/4/8 bytes are supported.

- T2 protocol: It adds pipelines features. It supports all operation code for ordered transactions. The number of the requesting cells (i.e. in a packet) is the same than the number of the response ones. Type 2 protocol has now been declared as obsolete.

- T3 protocol: It is an advanced protocol implementing split transactions for high bandwidth requirements. It supports out of order executions. The packet response size might be different than the packet request size (the number of cells differs between request and response).

In this thesis, the VIPs for the protocols T1 and T3 have been developed as T2 is obsolete now. The general narration of development of T3 protocol is presented later in this chapter. As T1 is a subset of the T3 protocol i.e. all the features of T1 is also included in T3, the description is done for T3 protocol only.

2.1.3 Operations and Features of STBUS T3 Protocols

The operations and features which are supported by the VIP is represented in the Table 2.1. This operations and features has been supported by the implemented VIP. STBUS T1 protocol supports only 1,2,4,8 byte store and load operation from the 2.1.

2.2 Overview of Universal Verification Methodology

The UVM (Universal Verification Methodology), an effort by an Accellera committee, was introduced in December 2009. UVM uses OVM and VMM as its foundation. UVM consists class libraries which provides the building blocks needed to quickly develop well-constructed and reusable verification components and test environments. It uses system verilog as its language. All three of the simulation vendors (Synopsys, Cadence and Mentor) support UVM today which was not the case with other verification methodology [16].

Table 2.1: Operations and Features supported by STBUS T3 Protocol

Operation & Feature	Description
Store Operation	It writes data in a particular address. The length of data can be 1,2,4,8,16,32,64 bytes.
Load Operation	It reads data from a particular address. The length of data can be 1,2,4,8,16,32,64 bytes.
Read Modified Write operation	It first reads from a particular address and then writes data on the same address. Each operation take 1 clock cycle to complete. So RMW takes total 2 clock cycles to complete. 4,8byte RMW operation is possible.
Swap operation	It reads from a particular address and writes data on the same address in same cycle. 4,8byte Swap operation is possible.
Chunk	It is a complex data type where packets linked by lck signal.
Message	It is a complex data type where chunks linked by msg signal.
Shaped packet	This feature enables the initiator to send only 1 cell request for a multicell read operation and the target to send only 1 cell response for a multicell write operation.
Split transaction	It differentiates request and response path. This helps to achieve pipeline in the system as multiple requests can be sent from initiator without getting the response back from target.
Pipeline	It enables initiator to send requests to the target without getting the response of the previous requests. It helps to achieve more bandwidth in the system.
Globally out of order locally order	The responses coming from the targets can be globally out of order but should be locally ordered. It means all the requests sent from the same initiator, should be responded back in same order. But requests sent from different initiators, can be responded back in arbitrary order.
Out of order	The responses can be locally out of ordered too. It means the requests sent from the same initiator, can be responded back in different order.

2.2.1 Testbench Architecture

The UVM testbench architecture generally has the following elements described in below.

- Data Item (Transaction): Data item class declares the stimulus properties. It does not include

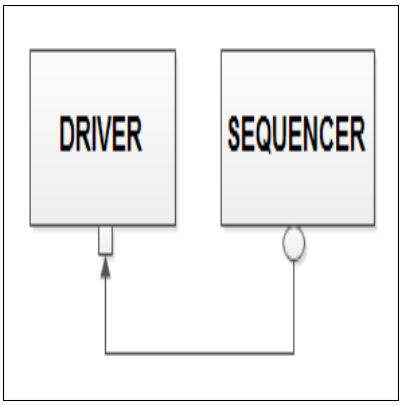

Figure 2.1: Connection between Driver and Sequencer

any phase methods. The properties are defined with *rand* to enable randomization. Transactions are created in UVM sequences and passed on to driver by the sequencer. Transaction class must extend the *uvm_sequence_item* class.

- Sequence: Stimulus transaction are created inside a UVM sequence extended from *uvm_sequence* class. A sequence consists of one or more sequence items. Each sequence item for the sequence is created in the *body()* task, often with pre-defined UVM macros such as *'uvm_do()*. The macro randomizes the sequence item and makes it available for driver to access.

- Sequencer: Sequencer is the component on which the sequences will run. *uvm_sequencer* is parameterized class and it is used to make the sequencer. *uvm_sequencer* is rarely needs to be extended and it is parameterized to a chosen transaction class. It contains a sequence descriptor which must be configured to refer a test sequence. It also contains a built-in TLM port *seq_item_export* which must be connected to a driver.

- Driver (BFM): Driver drives the DUT signals. It typically receives sequence items from the sequencer and processes them. Driver is extended from *uvm_driver* class which has built-in *sequence_item* handle(*req*) and TLM port to communicate with the sequencer. The TLM port connection between driver and sequencer has been shown in the Figure 2.1.

- Monitor: A monitor is the passive element of the verification environment. It monitors the interface, whenever a transaction comes in the interface, the monitor collects the transaction.

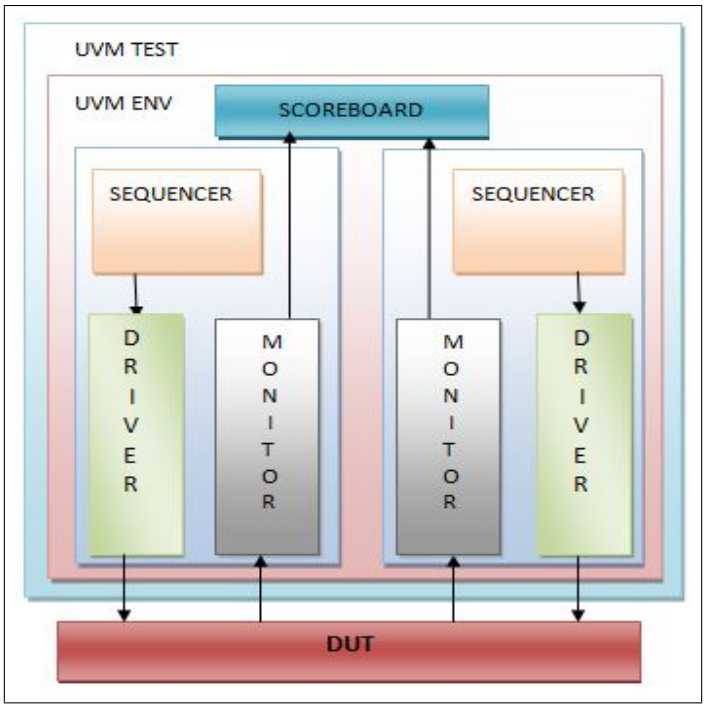

Figure 2.2: Basic UVM based testbench

Monitor is extended from the base class *uvm_monitor* which has built-in TLM analysis port to connect the different elements of the environment. Typically a monitor sends the transaction which it collects from the interface, to the checker component for protocol checking purpose, scoreboard component for scoreboarding purpose, and coverage component for recording the coverage.

- Agent: Agent encapsulates driver, monitor and sequencer. Agent can be 2 types: passive and active. An active agent drives the signal to the DUT and so an active agent will have driver and sequencer. A passive agent just monitors the DUT signals. So only the monitor is instantiated in passive mode.

- Environment: Environment is the top of the test bench architecture. Environment contains one or more agents depending upon the design.

A basic UVM based testbench is shown in figure 2.2.

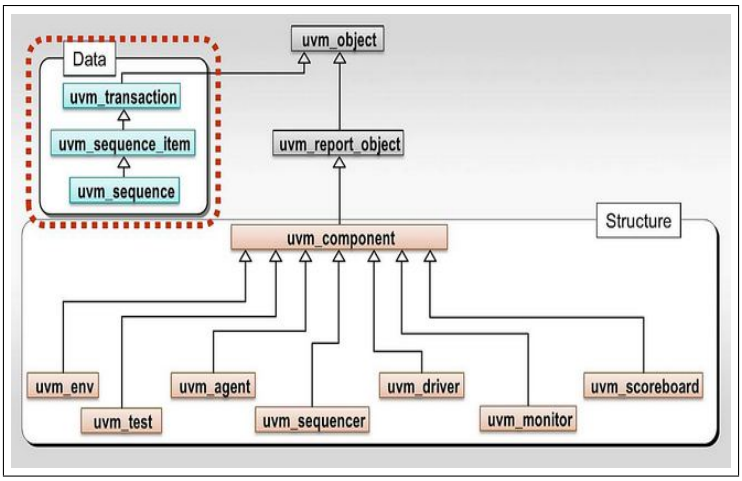

Figure 2.3: Base Library of UVM
[17]

2.2.2 Base classes in UVM

UVM is a methodology and a class library for building advanced reusable verification component. Figure 2.3 shows the base class library of UVM. The advantages of using UVM class library are-

- The UVM class library provides features required for verification and also for printing, copying, test phases and factory methods etc.
- Each component in figure VIP has been derived from corresponding UVM class library components in Figure 2.3. Using base classes increases the readability of the code since each component's role is predetermined by a parent class which is generic.

2.2.3 UVM Phases

In UVM simulation runs in predefined phases so all the components used in the verification environment need to implement phase methods. The phases in all the components are executed in a synchronized way. The predefined phases are shown in the figure 2.4. The importance of the phases are following:

- Build phase: Create and configure testbench structure.

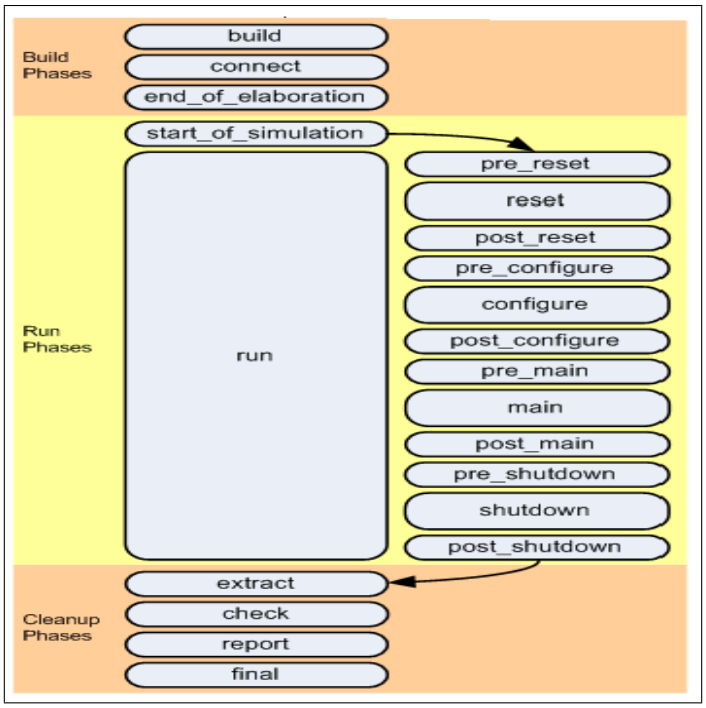

Figure 2.4: UVM Phases
[18]

- Connect phase: Establish cross-component connections.
- End_of_elaboration phase: Check for correctness of testbench structure.
- Start_of_simulation phase: Set configuration for stimulus generation.
- Run phase: Simulate the DUT. Run phase takes time to execute while all the other phases are not time consuming. All phases except run phase run in zero simulation time.
- Extract phase: Extract data from different points of the verification environment.
- Check phase: Check for any unexpected conditions in the verification environment.
- Report phase: Report results of the test.
- Final phase: Terminate the simulation.

Chapter 3

Development of STBUS VIP

The development of the STBUS VIP is illustrated in this chapter. Each and every component of the VIP has been referred here and their functionality has been narrated. STBUS T3 protocol has been taken as the reference and discussion has been done only for it. As T1 has a limited set of opcode which are also there in the T3 protocol, so there is only difference about the functionality and code. Except that the number of components, their functionality and the structure of the VIP are exactly same as of the VIP of STBUS T3 protocol.

3.1 STBUS T3 VIP

STBUS T3 protocol supports the features listed in Table 2.1. Figure 3.1 shows the architecture and all the components of the VIP. Brief discussions about all the building blocks in the diagram has been given in this section.

The modeling of the STBUS VIP has been done using UVM methodology. The structure of the VIP follows the testbench architecture defined in the previous chapter.

As STBUS can be taken as the working protocol of two types of IP, the VIP will also have 2 types of agent. An agent will act as an initiator and other one will act as the target. The need of VIP is to verify IPs in a SoC. The initiator and target part of the VIP will be needed to verify an target IP and initiator IP respectively. The initiator and target part of the VIP will be deactivated when

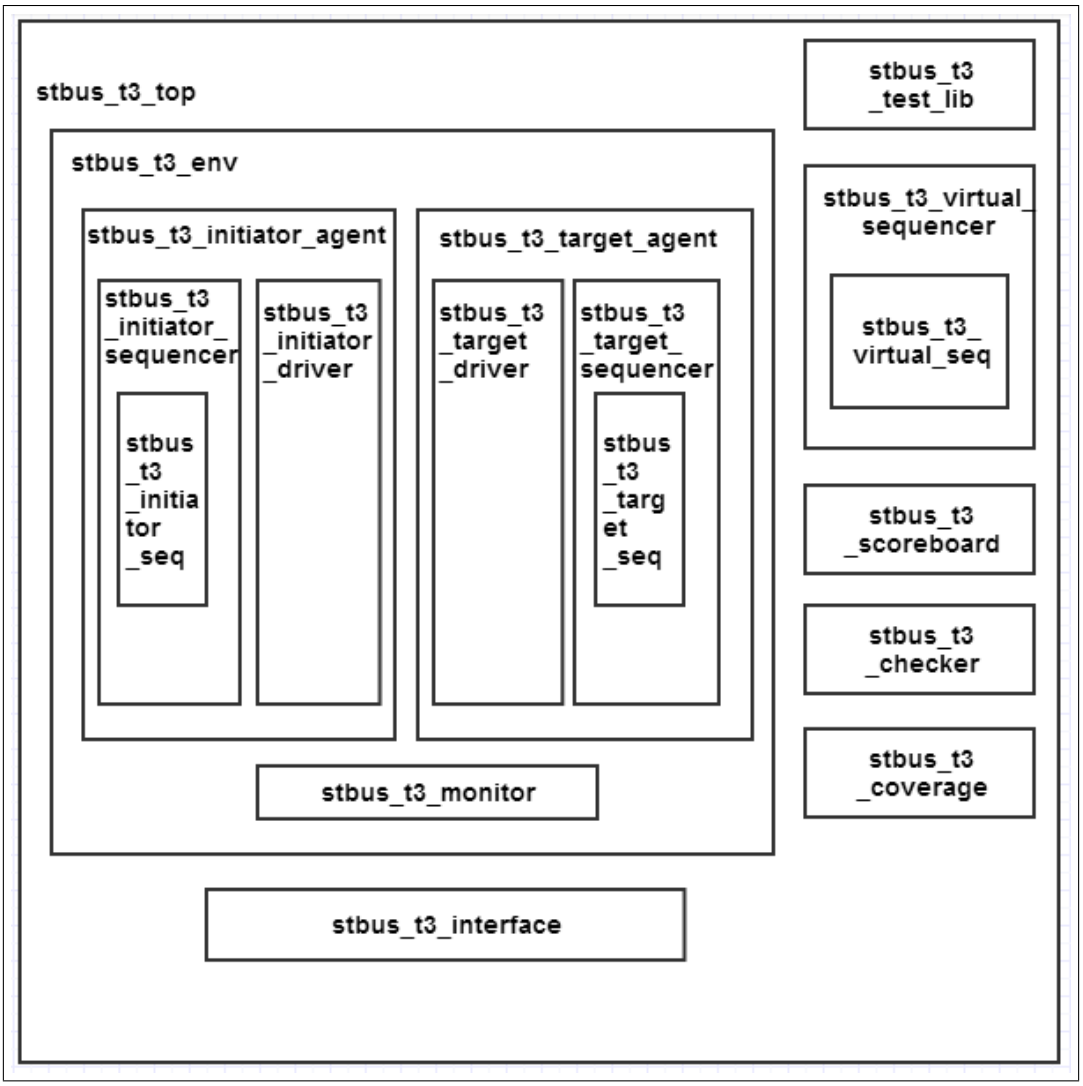

Figure 3.1: Block Diagram STBUS T3 Verification IP

the RTL to be verified is initiator and target respectively. Both the agents will have a driver and a sequencer each as because both the agents are active. VIP has only one monitor as the bus monitor. The idea behind keeping one monitor is it will collect both the transaction (request and response) even if one agent is deactivated while verifying any design. This is the basic structure of the VIP. The other modules such as checker, scoreboard, coverage module are included in the VIP for different reason which has been described in the later part.

As clarified above, the VIP has 2 active agents namely *stbus_t3_initiator_agent* and *stbus_t3_target_agent*.

The agents have their respective driver and sequencer. *stbus_t3_initiator_agent* has *stbus_t3_initiator_driver* and *stbus_t3_initiator_sequencer*, and *stbus_t3_target_agent* has *stbus_t3_target_driver* and *stbus_t3_target_sequencer* components encapsulated. Both the sequencers have respective sequences which run in the sequencers. Some sequences have been given in the VIP which the users can use to execute tests. Except this, the users can add their own sequences too. Both the agents and common bus monitor *stbus_t3_monitor* are encapsulated in the environment *stbus_t3_env*. There is a top module in which the environment, interface, test library, sequence library, checker, coverage are instantiated.

- *stbus_t3_sequence_item*:

  The *sequence_item* or the data item of VIP has all the signals related to STBUS protocol. The signals related to initiator (request packet/transaction) is constrained and randomized intelligently in the top level test so that it becomes a valid transaction for the STBUS protocol. The signals related to target (response packet/transaction) is not randomized as because those signals are generated by the target depending upon the given request by initiator.
  stbus_t3_sequence_item has been inherited from the base class of UVM class library, *uvm_sequence_item*.
  The rand, constrained properties of System Verilog language is used in *sequence_item* to generate random but valid transaction.

Driver drives the DUT signals. It basically receives the transaction object from the sequencer and converts it in to the pin level signal to the interface. Driver is the active part of the verification environment because it drives data to the interface. STBUS T3 VIP has 2 active drivers. The description about the drivers as follows:

- *stbus_t3_initiator_driver*:

  The initiator driver is responsible for driving the signals associated with the request packet to the interface. It first requests sequencer to send the packet. In response of this request, sequencer creates a *sequence_item* and sends it to the driver via implicit TLM port. Driver receives the *sequence_item* and sends to the interface converting the transaction object to the pin-level signals. The driver then sends a *item_done* signal to the sequencer conveying the completion of the process message. Here, *stbus_t3_initiator_driver* has been extended

from the UVM base class *uvm_driver*. This parameterized base class has an input parameter for the *sequence_item* which is used by the driver. The *sequence_item* is received from the sequencer using implicit *seq_item_port* analysis port and *get_next_item(req)* method. After the completion of sending the signal to the interface, driver sends the *item_done* signal via the same analysis port.

- *stbus_t3_target_driver*:

  The target driver is responsible for sending the responses of the requests. To achieve this, the target driver has to do the following objectives in parallel:

  1. Collect the requests from interface.
  2. Create the responses for the requests.
  3. Send the responses to interface.

  The target driver has to take care of the pipeline, out-of-order, GOOOLO features, which has been defined in Table 2.1, for sending the responses. It has the connection with target sequencer via analysis port also, but it does not play any significant role except turning on the driver. Once turned on, the driver has to collect request, generate response and send response by its' own.

- *stbus_t3_initiator_sequencer & stbus_t3_target_sequencer*:

  Sequencers are used to run the sequences when the drivers demand it. The sequencers have been inherited from *uvm_sequencer*.

- *stbus_t3_initiator_agent & stbus_t3_target_agent*:

  Agents are the container of the respective driver and sequencer. So initiator agent is the container of initiator driver and initiator sequencer, and the target agent is the container of target driver and target sequencer. The agents are inherited from the UVM base class *uvm_agent*. As both the agents are active, the driver and sequencer of respective agents are connected by the TLM port.

- *stbus_t3_initiator_seq*:

  The sequence is a series of transaction. There are 59 sequences for initiator in the STBUS VIP.

Table 3.1: List of Initiator Sequences with Targeted Feature

Targeted Feature	Number of Sequences
Store Operation(1,2,4,8,16,32,64 bytes)	7
Load Operation(1,2,4,8,16,32,64 bytes)	7
RMW Operation(4,8 bytes)	2
SWP Operation(4,8 bytes)	2
Chunk of Store and Load	14
Message of Store and Load	14
Out of Order and GOOOLO	1
Store and then Load	7
Store-RMW-Load	1
Store-SWP-Load	1
Error Sequences	3

The sequences help the user to get a jump-start in achieving the desired coverage goal. The sequences can be reused, extended, randomized, and combined sequentially and hierarchically in various ways. The user can add their own sequences in the VIP if needed. Table 3.1 is given to list the number of sequences with targeted feature of STBUS T3 protocol.

The *stbus_t3_uvc_initiator_seq* is inherited from the UVM base class *uvm_sequence*. In the body task, the *sequence_item* is randomized and constrained using *'uvm_do_with* macro.

- *stbus_t3_uvc_target_seq*:

  It has only one built in sequence which is always executed in parallel with the initiator sequence. This particular sequence just makes the driver of target operating.

- *stbus_t3_monitor*:

  The monitor of the VIP is a bus monitor which monitors all the transaction coming at the interface. It means the monitor collects the requests of initiator and responses of target. *stbus_t3_monitor* is inherited from the *uvm_monitor* base class of UVM. The monitor sends the transaction it collects from the interface to the coverage, scoreboard via TLM port for recording coverage and scoreboard purposes respectively.

- *stbus_t3_env*:

  stbus_t3_env is the environment of the VIP. It is used to encapsulate the agents, monitor. The

user can deactivate or activate the agents in this environment. If the user wants to use the VIP as master/slave then the target/initiator agent has to be deactivated. The RTL which is going to be verified using this VIP will take the place of the deactivated target/initiator.

- *stbus_t3_checker*:

  It is a module where PSL assertions and checks are written. It works as the protocol checker. If any transaction happens which violates the protocol, an error message is generated from this checker. The check is instantiated in the *testbench_top* module.

- *stbus_t3_scoreboard*:

  Scoreboard is used for to check whether the different modules are working correctly or not. The basic difference between scoreboard and checker is:

  - Checker checks for protocol specifications like:
    - Every request has to be responded back.
    - Every request has to be granted.
    - If any request happens with *req* high and *eop* (end of packet) low, then there would be a subsequent request with *eop* high. So, all the checks together make the protocol specification.
  - Scoreboard checks for wrong operation rather than illegal operation. Like if the read operation gives a wrong data then the checker will not have any role to identify the defect as it is not a protocol specification error, rather than it would be detected by the scoreboard.

  stbus_t3_scoreboard has been inherited from the UVM base class *uvm_scoreboard*. The base class has the implicit import which has been used to connect the scoreboard with the monitor. Monitor sends the transactions to the scoreboard using the analysis ports.

- *stbus_t3_coverage*:

  stbus_t3_coverage is required for getting the functional coverage. It is inherited from the UVM base class *uvm_component*. The implicit analysis port of the class is used to connect it with

the monitor. Monitor sends the transaction it monitors from the interface for recording the coverage. It has covergroups and coverbins associated with it. The covergroup construct encapsulates the specification of a coverage model. Each covergroup specification can include the following components:

- A clocking event that synchronizes the sampling of coverage points.
- A set of coverage points.
- Cross coverage between coverage points.
- Optional formal arguments.
- Coverage options.

A covergroup can contain one or more coverage points. A coverage point can be an integral variable or an integral expression. A coverage point creates a hierarchical scope, and can be optionally labeled. If the label is specified then it designates the name of the coverage point. Table 5.5 shows the coverbins associated with the coverpoints for T3 protocol.

- *stbus_t3_virual_sequencer*:

  Typical sequence interacts with a single DUT interface. When multiple DUT interfaces need to be synchronized a virtual sequencer is required. It synchronizes timing and transactions between different interfaces. The multiple DUTs in this design are the initiator and target. *stbus_t3_virual_sequencer* is inherited from the UVM base class *uvm_sequencer*. This virtual sequencer has handles of both, initiator sequencer and target sequencer.

- *stbus_t3_virual_sequence*:

  It is the library of sequences associated with the test library. So the library has 59 sequences. Every sequences has a handle of both, initiator sequences and target sequences. Both the sequences have been kept in fork join construct. So both will run in parallel. When a test executes, a particular virtual sequence runs, so both the sequences i.e. initiator sequence and target sequence starts to execute in parallel.

- *stbus_t3_test_lib*:

  It is the library of all the tests of the VIP. All the tests have a unique sequence associated with it as the default sequence. The user can execute any test given in this test library or can make own test. The STBUS T3 VIP has 59 tests which verify all the T3 protocol features.

- *stbus_t3_interface*:

  Interface is where the transaction level signal transforms into pin level signal. In a general system, the driver sends the transaction or packet level signal to the interface where it is transformed into pin-level signal. Monitor receives it from interface and transforms into packet level signal. *stbus_t3_interface* has been inherited from *uvm_interface*. The following list shows which component of the VIP interacts with *stbus_t3_interface*:

  - *stbus_t3_initiator_driver* sends the request packet.
  - *stbus_t3_target_driver* collects the request packet and sends the response packet.
  - *stbus_t3_monitor* collects both the request and response packets.

- *stbus_t3_top*:

  It is the module where instances of interface, checker are given. Clock and reset are generated here. The user can specify the name of the test he wants to execute in this module.

3.2 STBUS T1 VIP

The components of STBUS T1 VIP are exactly same as STBUS T3 VIP. The only difference is T1 supports a limited number of opcode. So it has limited functionally. For the development point of view, it has limited functionality and lesser lines of code. But the numbers of components are same with the implementation of those components are lot more easier than the STBUS T3 VIP. It has 12 in-built tests to support the features supported by STBUS T1 as the interconnect protocol.

Chapter 4

Experimental Environment

In this chapter, the experimental environment of the project has been described. The experimental environment is described in two sections. Firstly, the verification of the STBUS protocol is done by back to back connection of the initiator and target part of the VIP. This is discussed in the first section. Verifying the VIP would also mean that both the protocols(T1 and T3) has been verified too as the VIP is nothing but a mimic of the protocol. As the protocol is verified, automatically the designed VIPs are also verified. In the second section, the experimental environment of verifying an IP by the already verified VIPs is described.

4.1 Verifying the VIPs

The VIPs are verified by connecting the initiator and target back to back. In this type of setups, the initiator is used to generate the stimuli for verifying the protocol and the target is used to generate the response for the stimuli generated by initiator. The STBUS T1 VIP, STBUS T3 VIP has 12 tests and 59 tests respectively, which has been designed in this work, to generate different transactions.

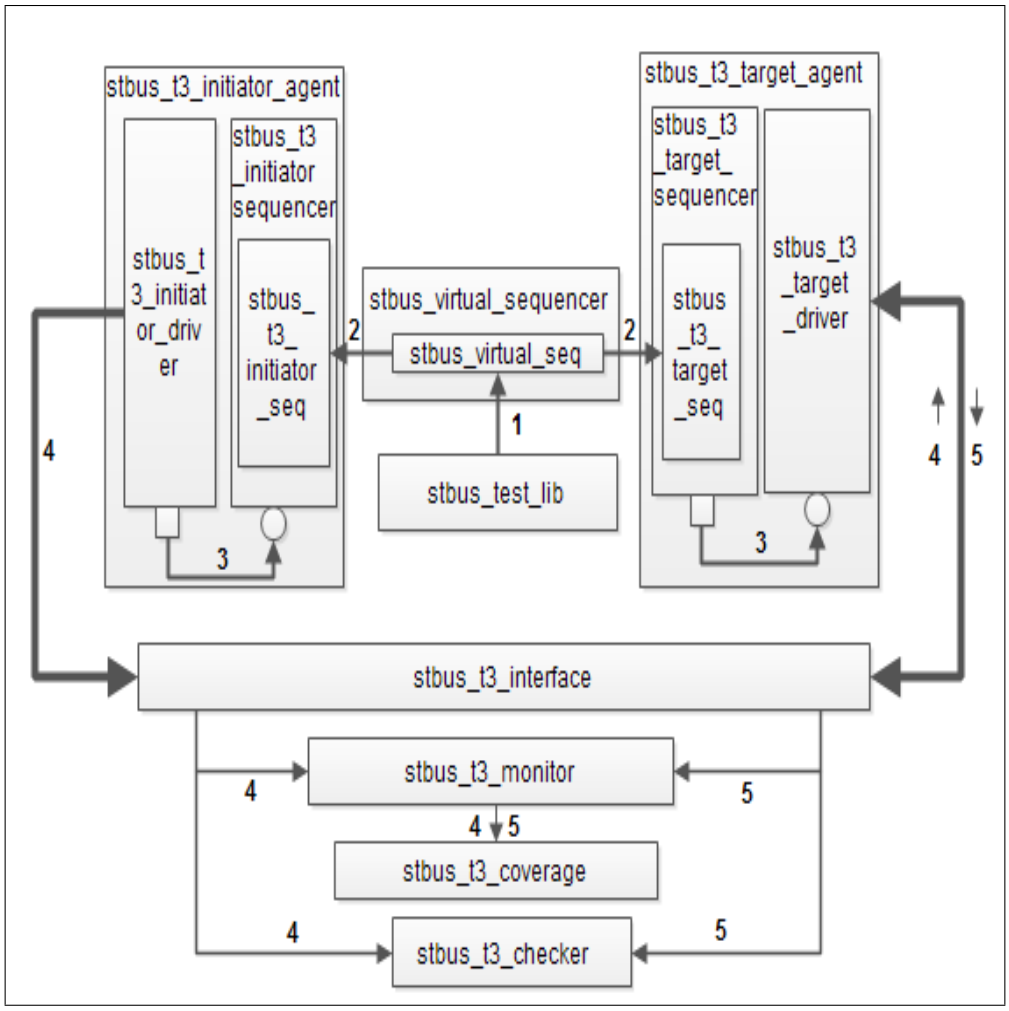

Figure 4.1: Working Principle of STBUS VIP

4.1.1 Steps of verifying VIP

The VIPs are verified connecting a PSL checker file. The PSL [19] checker file has assertions written which completely define the STBUS protocols. The assertions get hit if given transactions are satisfies the conditions written is assertions. There are assertions which checks whether the transactions are generated respecting the protocol. If any transaction violates the protocol then test is terminated and the error message shows which assertion is violated. Figure 4.1 shows steps of verifying VIP. The steps 1 to 5 shown in figure is described below:

Step 1: A test is being executed from the testcase library provided in the VIP. The test stars the

virtual sequence which has been defined as the default sequence in the test. The sequence is written in the virtual sequence library.

Step 2: The virtual sequence has pointers of two different sequences. One is from initiator sequences which is used to define the transaction of the initiator. The other is from target sequences which is used to turn on the target. Both the sequences are started to execute in parallel.

Step 3: Both the driver puts request to the respective sequencers to provide the transactions. While the initiator sequence has an important role to play defining the type of transaction the driver of initiator will send to the interface, the sequence of target has no role to play as such.

Step 4: The driver of initiator sends the transaction to the interface. The same transaction is received by the driver of target to generate the response. The transaction is also collected by the monitor. The monitor sends it to the coverage collector for coverage. The checker also collects the same from interface for checking purposes respectively. If the transaction does not follow STBUS protocol, then the checker generates an error message and the test gets stopped.

Step 5: The driver of target generates a response of the request sent by initiator and drives the response to the interface. The same transaction is collected by the monitor. The monitor sends the transaction to the coverage collector for coverage purposes. The checker also collects the same from interface for checking purposes respectively. If the transaction does not follow STBUS protocol, then the checker generates an error message and the test terminates.

A regression test is executed having all the tests included in it. After the completion of the regression test, the functional and assertion coverage are checked.

4.2 Verifying IP

The verified VIPs has been used to verify an IP, namely Flextf IP which is a part of the SoC Yushan3. Figure 4.2 shows the testbench for the verification of the IP. The IP has connection with STBUS T1 VIP, STBUS T3 VIP, IDP (Image Data Protocol) VIP. IDP VIP is used to give the input image in the IP, STBUS T1 VIP is used for verification of the registers and register initialization, and STBUS T3 VIP is used for normal data transmission. The verification of IDP is

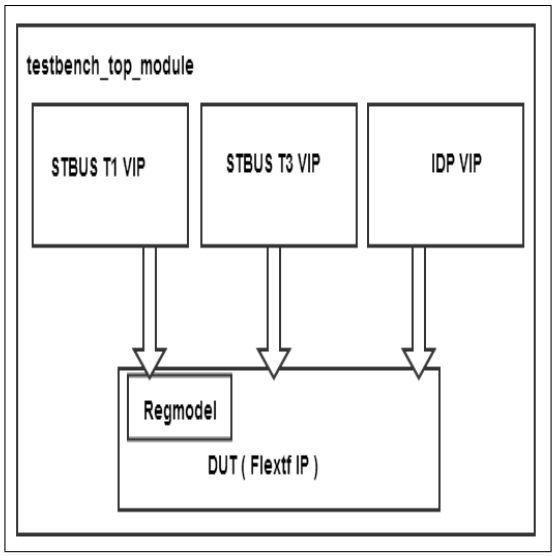

Figure 4.2: Testbench for verifying the IP

not the scope of this dissertation. The focus will be on STBUS protocol only.

4.2.1 Steps of verifying IP

This experiment has been done to show the effectiveness of the VIP in helping the user to achieve the coverage goal quickly by using the given tests. The testbench for verifying an IP has been developed where the designed VIPs have been plugged in the testbench. Figure 4.2 shows that the IP has 3 VIP connected with it. The purpose of the different protocols are the following:

- IDP: IDP is used to give an input picture to the IP.
- STBUS T1: It is used for the register verification and register initialization.
- STBUS T3: All the data flows after register initialization.

The discussion of IDP is not in the scope of this work. After omitting it out, the steps used in this experiment are:

Step 1: Registers are verified by write followed by read operation in all the registers of the RTL.

Step 2: Executing a regression test of STBUS T3 VIP testcases on the RTL.

Chapter 5

Results

First of all, the functional coverage and assertion coverage of the VIPs have been verified. Then, the verified VIPs have been used to verify an IP as the RTL and it has been examined how much jump-start the VIPs can give in achieving the coverage goal. Various coverage like code coverage (block coverage, expression coverage, toggle coverage), functional coverage (covergroup coverage, assertion coverage) of the RTL has been evaluated and the result has been given. The result includes different sections for the functional coverage of the STBUS T1 VIP & STBUS T3 VIP, and code coverage of the IP which has been verified by the VIPs. Each section of result is followed by an analysis of the result.

5.1 Functional Coverage of STBUS T1 VIP

In this section the functional coverage of STBUS T1 VIP has been discussed in the form of assertion coverage and covergroup coverage.

5.1.1 Assertion Coverage of STBUS T1 VIP

The checker file has different blocks written for different interconnect width size. Table 5.1 refers the result of the assertion coverage for STBUS T1 VIP. It shows the coverage in percentage of

Table 5.1: Assertion Coverage (in %) of the checker implemented in the T1 VIP

Block	1 byte	2 byte	4 byte	8 byte
genblk3	100	100	100	100
genblk4	100	100	100	100
genblk6	0	N/A	N/A	N/A
genblk7	N/A	N/A	66.67	N/A
genblk8	N/A	33.33	N/A	N/A

Table 5.2: Coverpoints and associated coverbins

Coverpoint	Number of associated Coverbins
req	2
data	10
addr	10
be(byte enable)	26
eop (end of packet)	2
opc	8
r_req	2
r_data	10
r_opc	8

different blocks for different interconnect width. STBUS T1 protocol allows interconnect widths of 1, 2, 4, 8 bytes.

5.1.2 Covergroup Coverage of STBUS T1 VIP

The T1 VIP has covergroup for keeping the coverage too. The coverpoints of the covergroup are the signals of T1 interface and the there are coverpoints associated with it. Table 5.2 refers the coverpoints and coverbins associated those for the STBUS T1 VIP. The coverpoints namely req, addr, data, be, eop are related with the request packet. Remaining coverpoints r_req, r_data and r_opc are related with the response packet. The result of covergroup coverage is shown. Regression test included 14 tests for each of the 4 types of interconnect width. So the total tests has been recorded as 56 for the regression tests. 5.3 shows the number of coverbins has been hit for all of the coverpoints.

Table 5.3: Coverpoints Hit Results

Coverpoint	Coverbin Hits
req	2/2
data	8/10
addr	8/10
be(byte enable)	26/26
eop (end of packet)	2/2
opc	8/8
r_req	2/2
r_data	6/10
r_opc	8/8

5.1.3 Analysis of Result

The blocks namely genblk6, genblk7, genblk8 in Table 5.1 for 1 , 2, 4 byte respectively are not 100%. These checkers are written to check whether the address for data transmission is changing or not depending on the data size and interconnect width.

For 1 byte interconnect width the address should be increased for 2, 4, 8 byte of transmission as there would be cells associated with. Though the initiator is changing the address, the checks have not been hit. As no checks have been hit, the coverage is 0% for genblk6. For 2 byte interconnect width the address should be increased for 4, 8 byte of transmission as there would be cells associated with. For 2 byte of operation it should not change. The last operation has hit a check. As 1 out of 3 checks have been hit, the coverage is 33.33% for genblk7. For 4 byte interconnect width the address should be increased for 8 byte of transmission as there would be cells associated with. For 2, 4 byte of operation it should not change. The last 2 operation have hit the associated checks. As 2 out of 3 checks have been hit, the coverage is 66.67% for genblk8.

The coverbins for the coverpoints data, addr, r_data in Table 5.3 has not been hit completely by the 56 tests of regression test. To hit those coverbins the user can find out the region of value the non-hit coverbins and write some directed tests to get 100% hit.

Table 5.4: Assertion Coverage (in %) of the checker implemented in the T1 VIP

Block	4 byte	8 byte	16 byte
genblkLdDataDriven	0	0	0
genblkSizeforRMWorswap	100	100	100
genblk32bitbecheck	100	N/A	N/A
genblk64bitbecheck	N/A	100	N/A
genblk128bitbecheck	N/A	N/A	100
genblkErrorResponse	100	100	100
genblkEOP	100	100	100
genblkEOPshortPossible	N/A	100	100

5.2 Functional Coverage of STBUS T3 VIP

In this section the functional coverage of STBUS T1 VIP has been discussed in the form of assertion coverage and covergroup coverage.

5.2.1 Assertion Coverage of STBUS T3 VIP

The checker file has different blocks written for different interconnect width size. Table 5.4 refers the result of the assertion coverage for STBUS T1 VIP. It shows the coverage in percentage of different blocks for different interconnect width. STBUS T1 protocol allows interconnect widths of 1, 2, 4, 8, 16 bytes. But the PSL is written only for 4, 8, 16 bytes as others are rarely used.

5.2.2 Covergroup Coverage of STBUS T3 VIP

The T3 VIP has covergroup for keeping the coverage too. The coverpoints of the covergroup are the signals of T3 interface and the there are coverpoints associated with it. Table 5.5 refers the coverpoints and coverbins associated those for the STBUS T3 VIP.

The coverpoints namely req, addr, data, be, eop, src, tid, icn_info, cid_info, msg, lck, wrp, tid are related with the request packet. Remaining coverpoints r_req, r_data, r_opc, r_eop, r_src, r_tid, r_icn_info, r_cid_info, r_msg, r_lck, r_wrp, r_tid are related with the response packet. The result of covergroup coverage is shown. Regression test included 59 tests for each of the 3 types of

Table 5.5: Coverpoints and associated coverbins

Coverpoint	Number of associated Coverbins
req	2
data	10
addr	10
be(byte enable)	57
eop (end of packet)	2
opc	18
src	4
tid	16
msg	2
lck	2
wrp	2
cid_info	10
icn_info	10
gnt	2
r_req	2
r_data	10
r_eop	2
r_src	4
r_tid	16
r_msg	2
r_lck	2
r_cid_info	10
r_icn_info	10
r_gnt	2

interconnect width. So the total tests has been recorded as 177 for the regression tests. 5.3 shows the number of coverbins has been hit for all of the coverpoints.

5.2.3 Analysis of Result

The block genbllkLdDataDriven in the Table 5.4 has 0% coverage as because that particular check has not been covered by all the tests. The check makes sure that the data is always stable when load operation is done. Though the data is not changing in the load operation, that particular check is not hit.

Table 5.6: Coverpoints Hits Result

Coverpoint	Coverbin Hits
req	2/2
data	10/10
addr	10/10
be(byte enable)	53/57
eop (end of packet)	2/2
opc	18/18
src	4/4
tid	16/16
msg	2/2
lck	2/2
wrp	2/2
cid_info	8/10
icn_info	7/10
gnt	2/2
r_req	2/2
r_data	8/10
r_eop	2/2
r_src	4/4
r_tid	16/16
r_msg	2/2
r_lck	2/2
r_cid_info	8/10
r_icn_info	7/10
r_gnt	2/2

Table 5.6 shows the coverbins hit out of the defined coverbins. 4 bins of 57 of the covergroup be(byte enable) have not been hit due to the fact that the tests are not done for the interconnect width 1, 2 byte as T3 protocol is rarely used for those interconnect widths. cid_info and r_cid_info, icn_info and r_icn_info are not covered fully. Though data are covered fully, r_data is not covered fully because some operations have only the store part but do not have the load part.

5.3 Coverage of verification of IP

The verified VIPs has been used to verify an IP (Flextf IP) which is a part of the SoC (Yushan3). This SoC has been used in the cameras of phone. Flextf IP performs pixel decompression. It is

Table 5.7: Code Coverage (in %) of IP

Module	Code Coverage
flextf	85.71
flextf_dfv_ctrl	100
flextf_core	84.21
flextf_mem_mux	66.67
flextf_interface_lib.notech_resync	100
flextf_interface_lib.req_fifo_control	96.88
flextf_interface_lib.control_block(rtl)	86
flextf_interface_lib.resp_fifo_control	92.16
flextf_interface_lib.u_rst_block	100
flextf_interface_lib.u_rst_resync	100

based on the transposition of pixel value through a table featuring a binned number of elements in comparison to pixel range. The experimental environment has been shown in the figure 4.2. There are VIP of IDP, VIP of STBUS T1 protocol and VIP of STBUS T3 protocol. IDP is used to give the image input, STBUS T1 is used to register initialization, and STBUS T3 is used for the further data transmission. The intention of verifying the IP is to check how much jump-start the VIPs can give in achieving the coverage goal when verifying an IP/SoC. So the tests of STBUS T1 and T3 VIPs have been executed only. The code coverage of this experiment has been shown in the Table 5.7. Code coverage is the important aspect for this experiment as it shows how much the RTL has been verified or covered in the experiment.

5.3.1 Analysis of Result

The code coverage is not 100% as it is not justified code coverage. There are unreachable or unobservable code like testing logics, redundant code, and functionality not under verification. Except that there is code for IDP also which can not be covered by STBUS type transaction. An overall code coverage of 85.71% is achieved only by connecting the VIPs in the testbench and executing the in-built tests. So for the verification of that IP will be a lot of more easier as 85.71% code coverage has been achieved using the VIPs.

Chapter 6

Future Work and Conclusion

There are few features left to be added in the VIP like cache-ability, buffer-ability etc. Those are not used often, but the addition of these features will make the VIP more generic and protocol specific. As the industry is gradually inclining to use System Verilog as the main language for verification, the PSL checker is going to be replaced by system verilog assertions (SVA). I am also planning to use the STBUS VIP in the field of emulation where this platform will be tested in the hardware.

In this dissertation, the VIPs of STBUS T1 and STBUS T3 protocols have been developed and verified by the virtue of PSL protocol checker. The verified VIPs are then used in verifying an IP. The code coverage of that experiment is shown to prove the efficiency of the VIPs in providing the jump-start to achieve the coverage goal. So in the dissertation the following points have been discussed:

1. Verification of the STBUS protocols.

2. Development of reusable, pre-verified verification environment verification IP of STBUS protocols which can be used to verify IPs/SoCs which use STBUS as interconnect bus protocol.

3. Using UVM as the development methodology, it has been made sure that the VIPs can be simulated by any of the tools provided by the EDA vendors.

4. The developed VIPs has been used to verify an IP and the coverage result has been shown as a proof of the effectiveness of the VIP in achieving the coverage goal.

Bibliography

[1] International Technology Roadmap for Semiconductors

[2] Resve Saleh, Steve Wilton, Shahriar Mirabbasi, Alan Hu, Mark Greenstreet, Guy Lemieux, Partha Pratim Pande, Cristian Grecu, Andre Ivanov System-on-chip: Reuse and integration. In *Proceedings of the IEEE* (2006), IEEE, pp. 1050–1069.

[3] Surendran, S. A systematic approach to synthesis of verification test-suites for modular soc designs, 2006.

[4] Ho, W., and Pinkston, T. A design methodology for efficient application-specific on-chip interconnects. In *IEEE Trans. On Parallel and Distributed Systems.*

[5] Horspool, N., and Gorman, P. The asic handbook.

[6] Pankaj Chauhan, Edmund M. Clarke, Yuan Lu and DongWang. Verifying ip-core based system-on-chip designs.

[7] S. Rosenberg, K.A.Meade A pratical guide to adopting the Universal Verification Methodology (UVM). San Jose, CA: Cadence Design Systems, 2010.

[8] www.cadence.com

[9] S.DOnofrio Migrating existing AVM and URM testbenches to OVM. http://www.paradigm-works.com/common/pdfs/papers/AVMandURMtoOVMMigration-Paper.pdf.

[10] www.mentor.com

[11] J.Aynsley Easier UVM for functional verification by mainstream users.

[12] www.accellera.org

[13] F. Aloul and K. Sakallah Efficient verification of the PCI local bus using Boolean satisfiability. In *In International Workshop on Logic Synthesis (IWLS)* (2000).

[14] A.Mokkedem, R.Hosabettu, M. Jones, and G. Gopalakrishnan Formalization and analysis of a solution to the PCI 2.1 bus transaction ordering problem. In *Formal Methods in System Design, 16, 2000 (IWLS).*

[15] ST Microelectronics Pvt. Ltd STBUS Communication System: Concepts and Definations.

[16] Bhaumik Vaidya, Nayan Pithadiy An Introduction to Universal Verification Methodology. In *Journal of Information, Knowledge and Research in Electronics and Communication Engineering* (2012-13), Vol 2, Issue 2, pp. 421.

[17] www.verificationacademy.com

[18] www.soccentral.com

[19] www.doulos.com/knowhow/psl/