Information Engineering and Technology Faculty

The German University in Cairo

UVM for RISC-V processors

Bachelor Thesis

Author: Walaa Samy Ghareeb Hussein Bondok

Supervisor: Dr. Mohamamed Abdelghany

Co-Supervisor: Dr.Hassan Moustafa

Sponsored by : Si-Vision

Submission date: July 25 th, 2020

This is to certify that:

(i) the thesis comprises only my original work towards the Bachelor degree,

(ii) due acknowledgement has been made in the text to all other material used.

Walaa Samy

May st, 2020

Abstract

The Verification methodology of modern processor designs is an enormous challenge. As processor design complexity increases, System-level verification of such large designs has become a significant challenge for a verification engineer. A proper verification environment can bring out bugs that one may never expect in the design. This thesis presents a configurable verification environment for RISC-V processors.The methodology used for verification is based on the Universal Verification Methodology (UVM), a class library written in the System Verilog language. The thesis describes how the verification of a RISC-V processor uses the powerful tools of UVM. The target processors has been successfully verified and the coverage goals are met. The effort has been documented in this paper in detail.

Acknowledgement

I'm really thankful to Allah who gave me power and the ability to work on it and providing me with persistence, patience and commitment to finish my project, I want to thank everyone who was always supportive through-out my five years journey.

I would first like to thank my thesis advisor and co-superviser Dr. Mohammed Abdelghany and Dr. Hassan Moustafa for leading me and giving me the support I will always be grateful for being one of your students and especially in my bachelor project. An exceptional appreciation to my supervisers from Si-Vision,Eng Hussein Galal for his help, supporting, and his valuable advices and professional guidance during the duration of my bachelor thesis, and a special thanks to Eng.Sameh ElAshery who the first one introduced UVM and System Verilog to me in an inspiring way ,He consistently gave me his guidance and teached me how to reach my goal on my own and provide me with the most suitable, productive ideas to accomplish a successful project at the end. I would like to thank my family for their unconditional support and love,specially my sister Dr. Mona Samy and her husband Dr.Mahmoud Hassan they always pushed me forward to be a better person and got my back whatever happens to be successful and satisfied. I take this opportunity to express the profound gratitude to my fiance' Eng.Abdelrahman Elhamalawy who helped me to reach my potential . Thank you for your support

,kindness and willingness to make this process easier. I would like to thank my dormmates who shared with me all the stressful moments Special thanks to Nesrin ,Alyaa and sarah. A very last thank you to Dr.Ashraf Mansour, the prime founder of the German University in Cairo ,who awarded me scholarship to study my bachelor degree in GUC. thank you to people who made the university a very nice place and had too much memories with, for all the Doctors, TAs who taught me a lot throughout my five years journey as they made the GUC the best place and really I will always remember and be proud of.

Walaa Samy

July 25^{th}, 2020

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Contributions 2

1.3 Organization and Notations 2

2 Literature Review **3**

2.1 Introduction 3

2.2 verification 3

2.3 Directed test 4

2.4 Hardware Verification Languages 5

2.5 UVM 5

2.6 RISC-V processors 6

3 UVM Verification Hierarchy **8**

3.1 UVM Environment 8

3.2 UVM Test 11

3.3 UVM Testbench Top 12

3.4 UVM Phasing 13

3.5 UVM Transaction Level Communication Protocol 14

4 System Architecture **15**

4.1 Instruction Fetch 16

4.2 Instruction Decode and Execute 17

4.2.1 Instruction Decode Block 17

4.2.2 Execute Block 18

4.2.3 Exceptions and Interrupts 19

4.2.4 Signals Description 20

5 Test Methodology and Results **24**

5.1 Ibex Verification Plan 25

5.2 Random Instruction Generator 26

5.3 Ibex Interface 27

5.4 Ibex transaction 28

5.5 Ibex sequence library 28

5.6 Ibex virtual sequence library 29

5.7 Ibex virtual sequencer and Ibex sequencer 30

5.8 Ibex Agent 30

5.9 Ibex scoreboard 35

5.10 Ibex coverage 39

5.10.1 Functional coverage 39

5.10.2 Code coverage 44

5.11 Ibex environment 45

5.12 Ibex base test 46

5.13 Ibex Top 46

5.14 Test the configurability of the verification environment 46

6 Conclusions and Future Works **49**

A Source Code **53**

List of Figures

2.1 General Architecture of Risc-V SOC. 7

3.1 UVM Test Hierarchy 11

3.2 UVM Test Top Component. 12

4.1 Ibex architecture. 15

4.2 Instruction Fetch (IF) stage. 16

4.3 Instruction Decode and Execute. 17

5.1 Ibex UVM Verification Environment Architecture. 25

5.2 Instruction Generator communication with UVM Environment. . . 26

5.3 Test bench components connection through the interface 27

5.4 UVM Sequence and Driver Communication. 29

5.5 Driving instruction Agent by Test 1. 31

5.6 Driving data memory Agent by Test 2. 31

5.7 analysis FIFO Connection. 35

5.8 Functional Covergae Loop. 39

5.9 Coverage groups percentage result 40

5.10 code coverage results by test 1 44

5.11 code coverage results by test 2. 44

5.12 code coverage results by test 1. 45
5.13 code coverage results by test 2. 45
5.14 RI5CY processor Architecture 47
5.15 UVM Architecture for RI5CY processor 48

List of Tables

4.1 Signals Description

5.1 Signals Driving

5.2 Signals checking

5.3 Signals Coverage

List of Acronyms

API	Application Programming Interface
ALU	Arithmetic logic unit
CSRs	Control and Status Registers
CISC	complex instruction set computer
DPI	Direct Programming Interface
DUT	Device under test
EX	Instruction Execute Stage
FIFO	First In, First Out
FPGA	Field programmable gate array
HDL	Hardware description language
HVL	Hardware Verification language
ID	Instruction Decode Stage
IF	Instruction Fetch Stage
IG	Instruction Generator
ISA	Instruction set architecture
LSU	Load/Store Unit
MIE	Machine interrupt-enable register.
Mstatus	Machine-mode status register
OVM	Open Verification Methodology
PC	Program counter
Risc-V	Reduced-instruction-set Computing
TLM	Transactional Level Modeling
UVM	Universal Verification Methodology

Chapter 1

Introduction

1.1 Motivation

Verification is a major obstacle in creating the final product and it is the most important component in any chip design. The main goal of verification is to make sure that the design meets the functional requirements as specified in the specification. this thesis presents verification environment used for the validation of RISC-V processors with variable instruction set architectures. All verification system components are configured using a configuration database. The verification environment extensively validates the operation of RISC-v processors with the micro-architecture Model of the RISC-v processor implemented in Verilog Language, and an Random Instruction Generator designed in System verilog which generates random instruction sequences. The verification environment provides a robust control and monitoring environment to validate the cycle by cycle operation of the processor and aid in debug in case of failures

1.2 Contributions

For this thesis Ibex RISC-V processor have been chosen to verifiy but with generic implementation of verification environment to enable us later to verify different RISC-V processors similar but challengingly different systems . this environment able to verify these different designs with minimal effort to show the benefit and potential of such a system. The thesis will show that the UVM can be useful for building a configurable framwork to test different disgns implementations at the same time addressing its features of the methodology efficient.

1.3 Organization and Notations

The rest of the thesis is organized as follows. Chapter 2 provides bibliographical Research . Chapter 3 provides a background on verification environment. Chapter 4 provides an overview of processor architecture .Chapter 5 present the test methdology and results. Chapter 6 draws the conclusions and suggests some future works.

Chapter 2

Literature Review

2.1 Introduction

To understand the need for building a verification framework we need to look into how the methods for testing have developed over time. Before looking at the development of different methodologies, let's first examine the concept of verification and the target processor.

2.2 verification

Verification is simply the process of making sure the design does what it is supposed to do according to a specification. It has essentially two tasks; providing input stimulus that is able to control the design into all possible states, and be able to observe how the design responds. Verification is not just testing, but testing to the extent that you can be confident of the correctness of a design. However, there is no way to completely verify a design, except for really small designs, because

of the increasingly large number of possibilities. Thus we require methods that can provide as good as possible confidence that the verification is satisfactory. Verification is commonly done through simulation where you either create a model that behaves according to the specification for comparison, or specify directly the expected states/values at a given time.

2.3 Directed test

Directed testing is a quick and easy way to write simple tests for new designs and is often used during the initial stages to test specific cases. It requires little to no overhead and allows for direct targeting of expected signal transitions. For small designs this is an efficient way to write basic tests for early checking of correctness, but with increasing complexity and the need for exhaustive testing to confidently verify a design, this method requires significantly more effort to write and quickly becomes a tedious task. While it is still possible to create a solid set of tests this way, it usually requires significant effort to setup and maintain the code. It is also hard to reuse any part of a test being written for a specific design. Another major limitation is the human factor, meaning that we are only able to test the limited set of cases that is conceivable. Though clever minds can think of the most likely problems and take special conditions into account, there may still be a huge set of unknown combinations potentially creating bugs that would be difficult to imagine. Together with the rapid increase in design efficiency it soon became necessary to find better methods for testing and verification. This is what led to the creation of dedicated verification languages and more structured methodologies. It also became necessary with dedicated verification engineers to handle this part of the process.

2.4 Hardware Verification Languages

To meet the needs of verification engineers several HVLs was created, the most widely known of these being e , Vera/OpenVera , SystemC and SystemVerilog . Domain-specific languages like e and Vera provided several efficient verification features later improved with OpenVera and assertion based verification. SystemC using the C++ library provide an all domain language and is a generalpurpose programming approach to hardware description and verification, and serves as a contrast to SystemVerilog that is based on a HVL .

2.5 UVM

Building upon the already extensive library of SystemVerilog, UVM adds a comprehensive library of classes to help increase standardization and interoperability. The library is coded entirely in SystemVerilog source code, enabling not only a highly standardized structure, but also provides the freedom to choose the most appropriate tools for the given task. This makes the UVM a powerful toolkit for verification. Some of the most important advantages of using UVM are described below. Constraint random UVM makes efficient use of the constrain-random functionality enabled by SystemVerilog, and builds its transactions based on this. Single class hierarchy The classes in UVM are all expanded from a single root class, 'uvm object', enabling key functionality to be available throughout all the components. Object factory The factory is the central mechanism for creating objects or components. By registering objects with the factory, we enable a fully configurable hierarchy that can be modified with specialized implementations at run-time. Configuration/resource database One of the most important facili-

ties of UVM is a resource database that makes configurations globally accessible. This allows test specific configurations to be added to the testbench independently. Phases The different class threads running in a UVM environment are coordinated using a phase based execution, ensuring proper ordering of events. Prewritten code As all the classes used in a UVM environment are predefined with significant functionality, there is less work left to the user for each implementation. This is just a brief overview of the advantages, as more detailed information will be discussed when implementing the different classes in Chapter 3

2.6 RISC-V processors

Arm dominates the microprocessor architecture business, as its licensees have shipped 150 billion chips to date and are shipping 50 billion more in the next two years. But RISC-V is challenging that business with an open source ecosystem of its own, based on a new kind of processor architecture that was created by academics and is royalty free ,the figure below [Figure 1]show general architecture of Risc-v SOC. RISC-V started in 2010 at the University of California at Berkeley Par Lab Project, which needed an instruction set architecture that was simple, efficient, and extensible and had no constraints on sharing with others. Krste Asanovic (a founder of SiFive), Andrew Waterman, Yunsup Lee, and David Patterson created RISC-V and built their first chip in 2011. In 2014, they announced the project and gave it to the community. RISC-V is the newest addition to the risc instruction set architectures family. It's an open-source modern ISA, capable of scaling from small embedded systems up to large servers and mainframe computers. The design and verification of a 64-bit RISC-V compliant general-purpose microprocessor is described along this thesis. The first part of the thesis follows

the development of a basic single cycle implementation, supporting the 42 integer instruction required by the RISC-V ISA. The design was implemented on an ALTERA DE1-SoC FPGA development kit, achieving a clock rate of 55 MHz. Various techniques were used to improve the design performance and capabilities. The final design is a five stage pipelined micro-architecture with split cache, supporting integer, atomic, single and double precision floating point, multiplication and division instructions. The FPGA implementation was able to achieve a clock rate of up to 140 MHz with a power consumption of 122 mW.

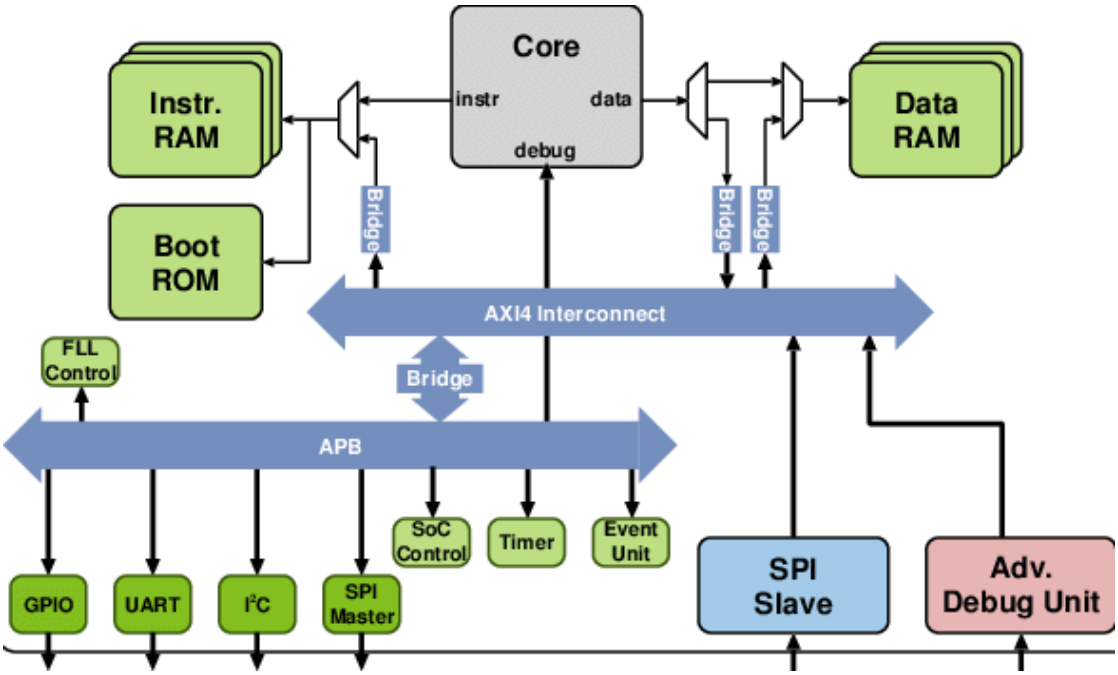


Figure 2.1: General Architecture of Risc-V SOC.

Chapter 3

UVM Verification Hierarchy

The UVM methodology is as a portable, open-source library from the Accellera Systems Initiative, and it should be compatible with any HDL simulator that supports SystemVerilog. UVM is also based on the OVM library which provides some background and maturity to the methodology. A key feature of UVM includes re-usability though the UVM API and guidelines for a standard verification environment. The environment is easily modifiable and understood by any verification engineer that understands the methodology behind it.

3.1 UVM Environment

A top-level UVM environment encompasses agent and scoreboard components and most often, other environments in its hierarchy. It groups several of the critical components of the test bench so that they could easily be configured at one place in any stage if needed. It can have multiple agents for different interfaces and multiple scoreboards for checks on the different type of data transactions. This way, the environment can enable/disable different verification components for specific tasks

.

- **UVM Agent**

  A UVM agent comprises of the sequencer, driver and the monitor of an interface. Multiple agents could be used to drive multiple interfaces, and they are all connected to the test-bench through the environment component. A UVM agent could be active or passive. Active agents include a driver and have the ability to drive signals, but passive agents only have the monitor and cannot drive pins. Even though a passive agent consists of the monitor only, it is vital to maintaining the level of abstraction that UVM promises and to maintain its structure by having all agents in the environment and not sub-components like monitors by themselves. By default, the agent is considered active, but this could be changed using the set() method of the UVM configuration database.

- **UVM sequence item**

  A UVM sequence item is the most fundamental component of the UVM hierarchy. It is a transaction that contains data items, methods, and may also contain the constraints imposed on them. A sequence item is the smallest transaction that can happen in a verification environment.

- **UVM sequence**

  A UVM sequence is a collection of transactions, also called the sequence items. A sequence gives us the ability to use the sequence item as per our requirements and to use as many sequence items as we want. The main job

of a sequence is to generate transactions and pass them to the sequencer.

- **UVM sequencer**

  A sequencer acts as a medium between the sequence items and the driver to control the flow of transactions from multiple sequences. A TLM interface enables communication between the driver and the sequencer.

- **UVM driver**

  A UVM Driver is a component class where the transaction-level sequence item meets the DUT clock/ bit/ pin-level activities. Driver pulls sequences from sequencer as inputs, then converts those sequences into bit-level activities, and finally drive the data onto the DUT interface according to the standard interface protocol. The functionality of driver is restricted to send the appropriate data to the DUT interface. Driver can well off course monitor the transmitted data, but that violates modularity aspects of UVM.

- **UVM Monitor**

  A UVM monitor looks at the pin level activity of the DUT and converts it back into transactions to send it to other components for further analysis. Generally, the monitor processes transactions like coverage collection and logging before sending them to scoreboards. .

- **UVM Scoreboard**

The scoreboard collects the transactions from the monitor and performs checks to verify if the collected data matches the expectation or not. The expectation generally comes from a golden reference model often written in languages such as C/C++ and interfaced with the test bench through Direct Programming Interface(DPI).

3.2 UVM Test

A UVM test is a top-level component that encapsulates all the test-specific information as shown in figure 1. The functions of the test component are- instantiating the environment, configuring it and invoking sequences through it. In test benches where there are multiple focus tests, a base test is first written extending from the uvm test class that instantiates the top-level environment and all the other focus tests are extended from this base test for more specific testing

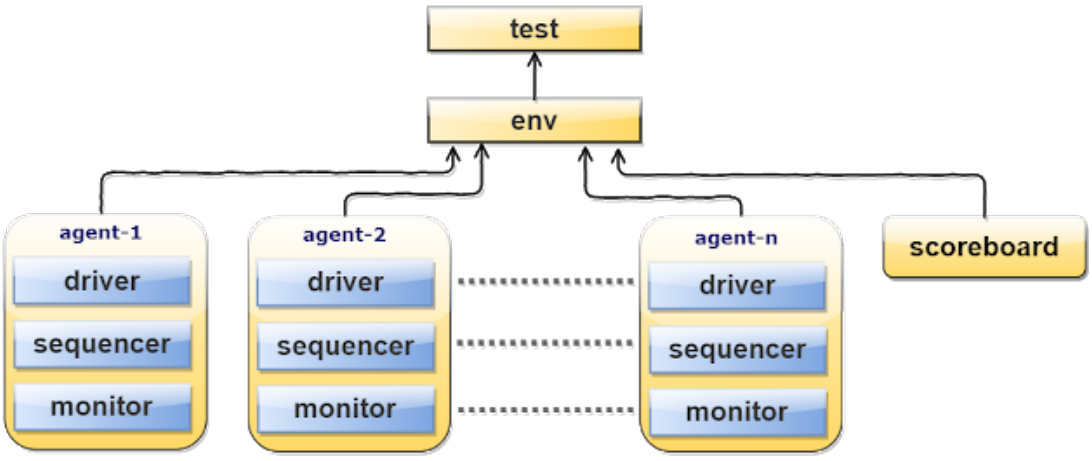


Figure 3.1: UVM Test Hierarchy .

3.3 UVM Testbench Top

The UVM testbench typically includes one or more instantiations design under test modules and interfaces which connect the DUT with the testbench. Transaction Level Modeling (TLM) interfaces in UVM provide communication methods for sending and receiving transactions between components. A UVM Test is dynamically instantiated at run-time, allowing the UVM testbench to be compiled once and run with many different tests

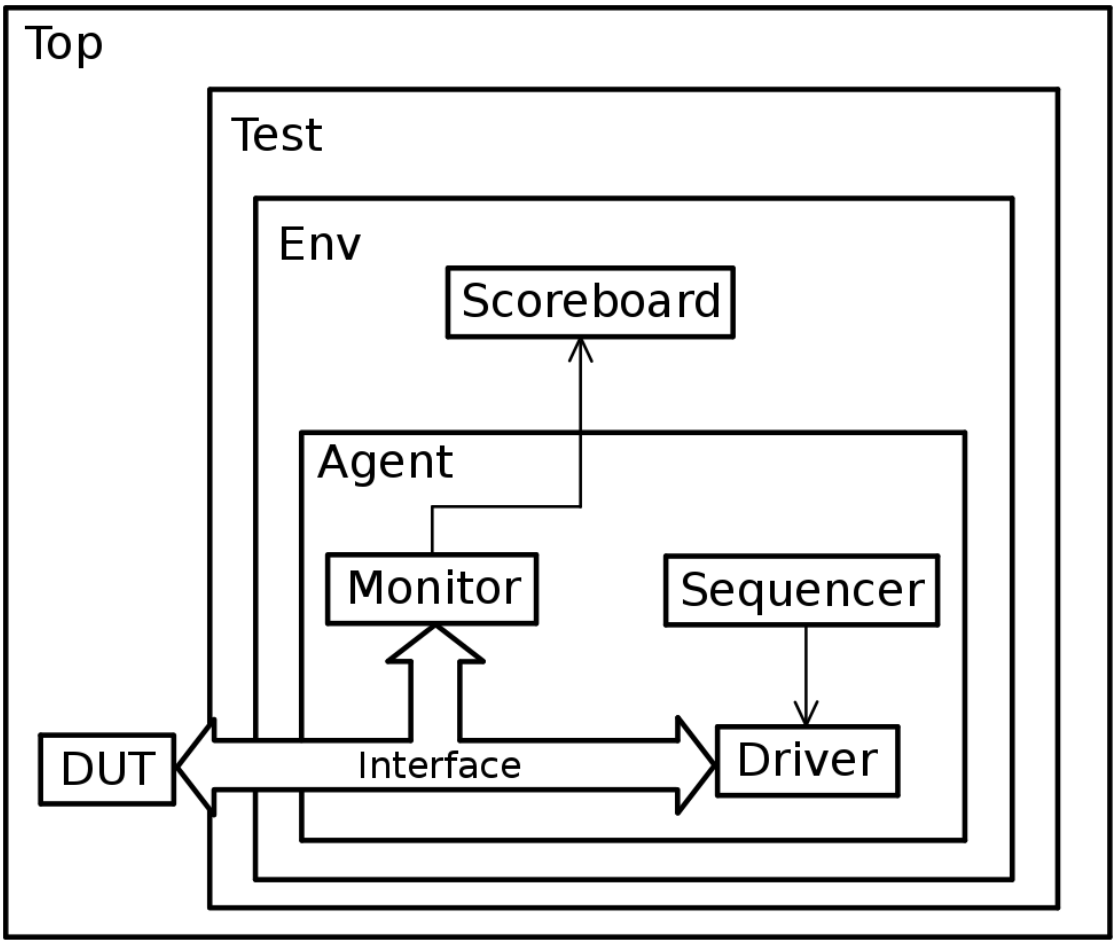


Figure 3.2: UVM Test Top Component.

3.4 UVM Phasing

Phasing is an essential feature of UVM methodology where different phases collect, run and process data to avoid run-time conflicts. Phases are a group of callback methods which could be tasks or functions. All the phases in UVM can be grouped into three main categories which are discussed below.

- **Build** The methods in a build phase enable us to build all the components and connect them. These are executed at the start of the simulation. All methods in this phase are functions only and hence execute in zero simulation time.

- **Connect** The connect phase connects UVM subcomponents of a class. Connect phase is executed from the bottom up. In this phase, the testbench components are connected using TLM connections. Agent connect phase would connect the monitor to the scoreboard.

- **Run** The run phase is the main execution phase, actual simulation of code will happen here. Run phase is a task and it will consume simulation time. The run phases of all components in an environment run in parallel. Any component can use either the run phase or the 12 individually scheduled phase. This phase starts at time 0. It is a better practice to use normal run phase task for drivers, monitors and scoreboards.

3.5 UVM Transaction Level Communication Protocol

Transaction refers to a class object that includes necessary information needed for communication between two components. Simple example could be a read or write transaction on a bus. Transaction-level modeling (TLM) is an approach that consists of multiple processes communication with each other by sending transaction back and forth through channels. The channels could be FIFO or mailbox or queue. The advantages of TLM are it abstracts time, abstracts data and abstracts function.

- **Basic Transaction Level Communication**

  TLM is basis for modularity and reuse in UVM. The communication happens through method calls. A TLM port specifies the API or function call that needs to be used. A TLM export supplies the implementation of the methods. Connections are between ports and exports and not between components. The ports and exports are parameterized by the transaction type being communicated .

- **Analysis ports and Exports**

  Analysis ports supports communication between one to many components. These are primarily used by coverage collectors and scoreboards. The analysis port contains analysis exports connected to it. When a UVM component class calls analysis port write method, then the analysis port iterates through the lists and calls write method of appropriate connected export. Similar to that of TLM FIFO Analysis ports also extends the feature to support multiple transaction.

Chapter 4

System Architecture

Ibex is a 2-stage 32 bit RISC-V processor core. Ibex has been designed to be small and efficient. the core is configurable to support four ISA configurations. Figure 1 shows a block diagram of the core.

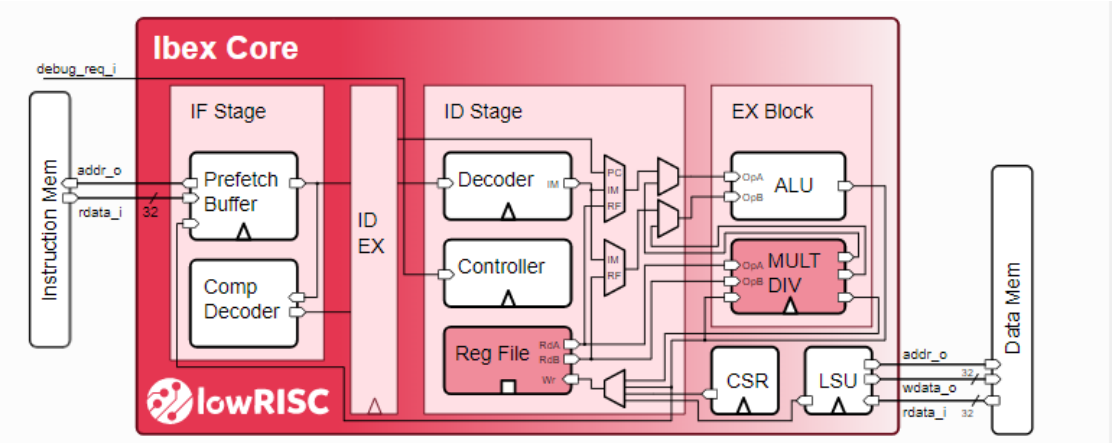


Figure 4.1: Ibex architecture.

4.1 Instruction Fetch

The Instruction Fetch (IF) stage of the core is able to supply one instruction to the Instruction-Decode (ID) stage per cycle if the instruction cache or the instruction memory is able to serve one instruction per cycle.

Instructions are fetched into a prefetch buffer . This buffer simply fetches instructions linearly until it is full. The instructions themselves are stored along with the Program Counter (PC) they came from in the fetch FIFO . A localparam DEPTH gives a configurable depth which is set to 3 by default.

The top-level of the instruction fetch controls the prefetch buffer (in particular flushing it on branches/jumps/exception and beginning prefetching from the appropriate new PC) and supplies new instructions to the ID/EX stage along with their PC. Compressed instructions are expanded by the IF stage so the decoder can always deal with uncompressed instructions (the ID stage still receives the compressed instruction for placing into mtval on an illegal instruction exception).

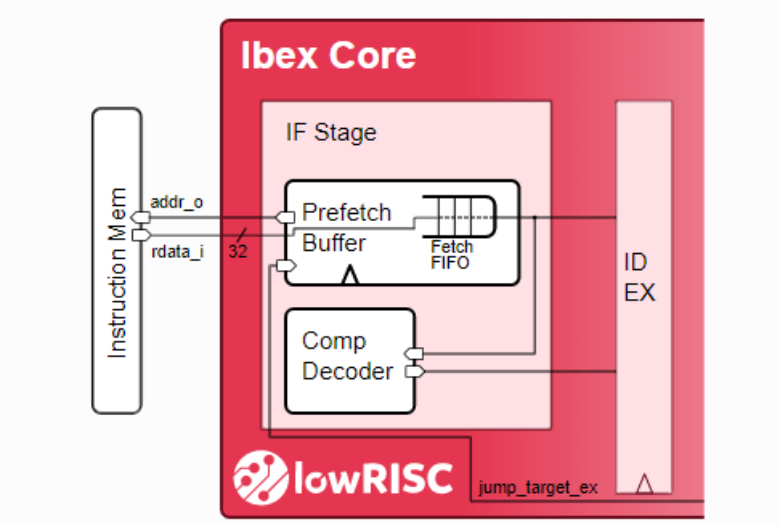


Figure 4.2: Instruction Fetch (IF) stage.

4.2 Instruction Decode and Execute

The Instruction Decode and Execute stage takes instruction data from the instruction fetch stage (which has been converted to the uncompressed representation in the compressed instruction case). The instructions are decoded and executed all within one cycle including the register read and write. The stage is made up of multiple sub-blocks which are described below.

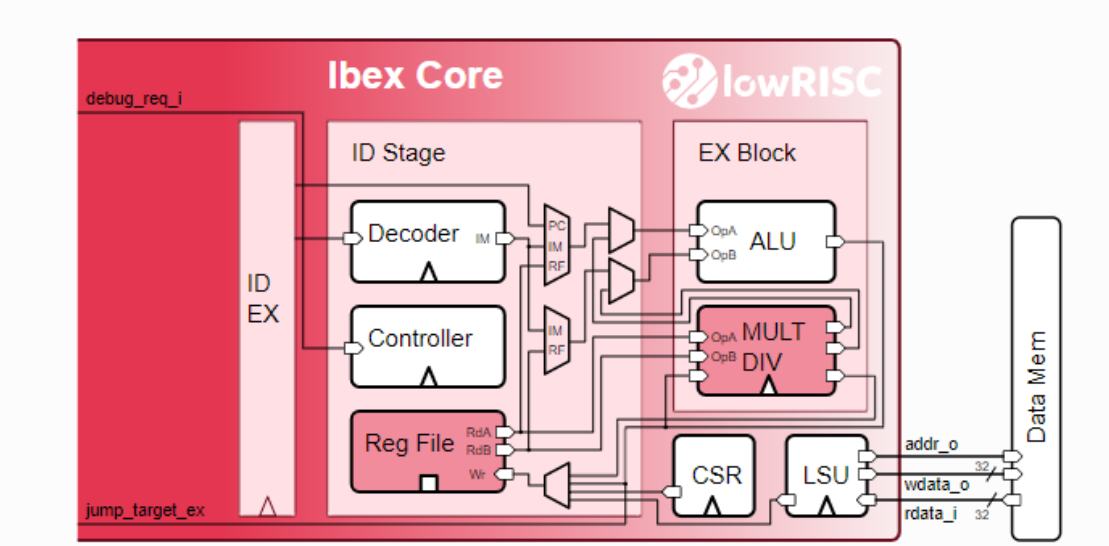


Figure 4.3: Instruction Decode and Execute.

4.2.1 Instruction Decode Block

The Instruction Decode (ID) controls the overall decode/execution process. It contains the muxes to choose what is sent to the ALU inputs and where the write data for the register file comes from. A small state machine is used to control multi-cycle instructions.

- **Controller** The Controller contains the state machine that controls the

overall execution of the processor. It is responsible for:

1-Handling core startup from reset

2-Setting the PC for the IF stage on jump/branch

3-Dealing with exceptions/interrupts (jump to appropriate PC, set 4-relevant CSR values)

5-Controlling sleep/wakeup on WFI

- **Decoder** The decoder takes uncompressed instruction data and issues appropriate control signals to the other blocks to execute the instruction.

- **Register File** Ibex has either 31 or 15 32-bit registers if the RV32E extension is disabled or enabled, respectively. Register x0 is statically bound to 0 and can only be read, it does not contain any sequential logic.

  The register file has two read ports and one write port, register file data is available the same cycle a read is requested. There is no write to read forwarding path so if one register is being both read and written the read will return the current value rather than the value being written.

4.2.2 Execute Block

The execute block contains the ALU and the multiplier/divider blocks, it does little beyond wiring and instantiating these blocks.

- **Arithmetic Logic Unit (ALU)** The Arithmetic Logic Logic (ALU) is a purely combinational block that implements operations required for the Integer Computational Instructions and the comparison operations required for the Control Transfer Instructions in the RV32I RISC-V Specification.

- **Control and Status Register Block** The CSR contains all of the CSRs (control/status registers). Any CSR read/write is handled through this block.

- **Load-Store Unit (LSU)** The Load-Store Unit (LSU) of the core takes care of accessing the data memory. Loads and stores of words (32 bit), half words (16 bit) and bytes (8 bit) are supported.

  Any load or store will stall the ID/EX stage for at least a cycle to await the response (whether that is awaiting load data or a response indicating whether an error has been seen for a store).

4.2.3 Exceptions and Interrupts

Ibex implements trap handling for interrupts and exceptions according to the RISC-V Privileged Specification, version 1.11.

When entering an interrupt/exception handler, the core sets the mepc CSR to the current program counter and saves mstatus.MIE to mstatus.MPIE. All exceptions cause the core to jump to the base address of the vector table in the mtvec CSR. Interrupts are handled in vectored mode, i.e., the core jumps to the base address plus four times the interrupt ID. Upon executing an MRET instruction, the core jumps to the program counter previously saved in the mepc CSR and restores mstatus.MPIE to mstatus.MIE.

The base address of the vector table is initialized to the boot address (must be aligned to 256 bytes, i.e., its least significant byte must be 0x00) when the core is booting. The base address can be changed after bootup by writing to the mtvec CSR. For more information, see the Control and Status Registers documentation.

The core starts fetching at the address made by concatenating the most significant 3 bytes of the boot address and the reset value (0x80) as the least significant byte. It is assumed that the boot address is supplied via a register to avoid long paths to the instruction fetch unit.

4.2.4 Signals Description

The following table descries each signal with its direction and in the next chapter will specify which component in UVM environment that is responsible for drive it

.

DIR	SIGNAL-NAME	SPEC-Description
IN	clk_i	clock signal
IN	rst_ni	Active-low asynchronous reset
IN	hart_id_i	Hart ID, usually static, can be read from Hardware Thread ID (mhartid) CSR
IN	boot_addr_i	First program counter after reset = boot_addr_i + 0x80
IN	fetch_enable_i	Enable the core, won't fetch when 0
OUT	core_sleep_o	Core in WFI with no outstanding data or instruction accesses.
OUT	instr_req_o	Request valid, must stay high until instr_gnt_i is high for one cycle
OUT	instr_addr_o[31:0]	Address, word aligned
IN	instr_gnt_i	The other side accepted the request
IN	instr_rvalid_i	This signal will be high for exactly one cycle per request
IN	instr_rdata_i[31:0]	Data read from instruction memory
IN	instr_err_i	instruction Memory access error
OUT	data_req_o	Request valid, must stay high until data_gnt_i is high for one cycle
OUT	data_addr_o[31:0]	Address, word aligned
OUT	data_we_o	Write Enable, high for writes, low for reads. Sent together with data_req_o
OUT	data_be_o[3:0]	Byte Enable. Is set for the bytes to write/read, sent together with data_req_o
OUT	data_wdata_o[31:0]	Data to be written to memory, sent together with data_req_o
IN	data_gnt_i	The other side accepted the request. Outputs may change in the next cycle.
IN	data_rvalid_i	This signal will be high for exactly one cycle per request.
IN	data_err_i	Error response from the bus or the memory

IN	data_rdata_i[31:0]	Data read from memory
EMB	mstatus	Machine Status
EMB	misa	Machine ISA and Extensions
EMB	mie	Machine Interrupt Enable Register
EMB	mtvec	Machine Trap-Vector Base Address
EMB	mepc	Machine Exception Program Counter
EMB	mcause	Machine Cause Register
EMB	mtval	Machine Trap Value Register
EMB	mip	Machine Interrupt Pending Register
EMB	mcycle	Machine Cycle Counter
EMB	mhartid	Hardware Thread ID
IN	irq_nm_i	Non-maskable interrupt (NMI)
IN	irq_fast_i[14:0]	15 fast, local interrupts
IN	irq_external_i	Connected to platform-level interrupt controller
IN	irq_timer_i	Connected to timer module
IN	irq_software_i	Connected to memory-mapped (inter-processor) interrupt register
IN	LUI rd,imm	Load Upper Immediate
IN	AUIPC rd,offset	Add Upper Immediate to PC
IN	JAL rd,offset	Jump and Link
IN	JALR rd,rs1,offset	Jump and Link Register
IN	BEQ rs1,rs2,offset	Branch Equal
IN	BNE rs1,rs2,offset	Branch Not Equal
IN	BLT rs1,rs2,offset	Branch Less Than
IN	BGE rs1,rs2,offset	Branch Greater than Equal
IN	BLTU rs1,rs2,offset	Branch Less Than Unsigned

IN	BGEU rs1,rs2,offset	Branch Greater than Equal Unsigned
IN	LB rd,offset(rs1)	Load Byte
IN	LH rd,offset(rs1)	Load Half
IN	LW rd,offset(rs1)	Load Word
IN	LBU rd,offset(rs1)	Load Byte Unsigned
IN	LHU rd,offset(rs1)	Load Half Unsigned
IN	SB rs2,offset(rs1)	Store Byte
IN	SH rs2,offset(rs1)	Store Half
IN	SW rs2,offset(rs1)	Store Word
IN	ADDI rd,rs1,imm	Add Immediate
IN	SLTI rd,rs1,imm	Set Less Than Immediate
IN	SLTIU rd,rs1,imm	Set Less Than Immediate Unsigned
IN	XORI rd,rs1,imm	Xor Immediate
IN	ORI rd,rs1,imm	Or Immediate
IN	ANDI rd,rs1,imm	And Immediate
IN	SLLI rd,rs1,imm	Shift Left Logical Immediate
IN	SRLI rd,rs1,imm	Shift Right Logical Immediate
IN	SRAI rd,rs1,imm	Shift Right Arithmetic Immediate
IN	ADD rd,rs1,rs2	Add
IN	SUB rd,rs1,rs2	Subtract
IN	SLL rd,rs1,rs2	Shift Left Logical
IN	SLT rd,rs1,rs2	Set Less Than
IN	SLTU rd,rs1,rs2	Set Less Than Unsigned
IN	XOR rd,rs1,rs2	Xor
IN	SRL rd,rs1,rs2	Shift Right Logical
IN	SRA rd,rs1,rs2	Shift Right Arithmetic
IN	OR rd,rs1,rs2	Or
IN	AND rd,rs1,rs2	And

Table 4.1 Signals Description

Chapter 5

Test Methodology and Results

A testbench is a network of verification components designed to check whether the RTL implementation meets the design specification or not . The general steps to verify a design are: generate inputs, reset DUT, configure the DUT, run the test, capture outputs, verify the correctness, report the errors and provide possible solutions. This chapter discusses about the architecture of the proposed verification architecture in detail. Fig.5.1 shows the architecture and the various components that are a part of it.

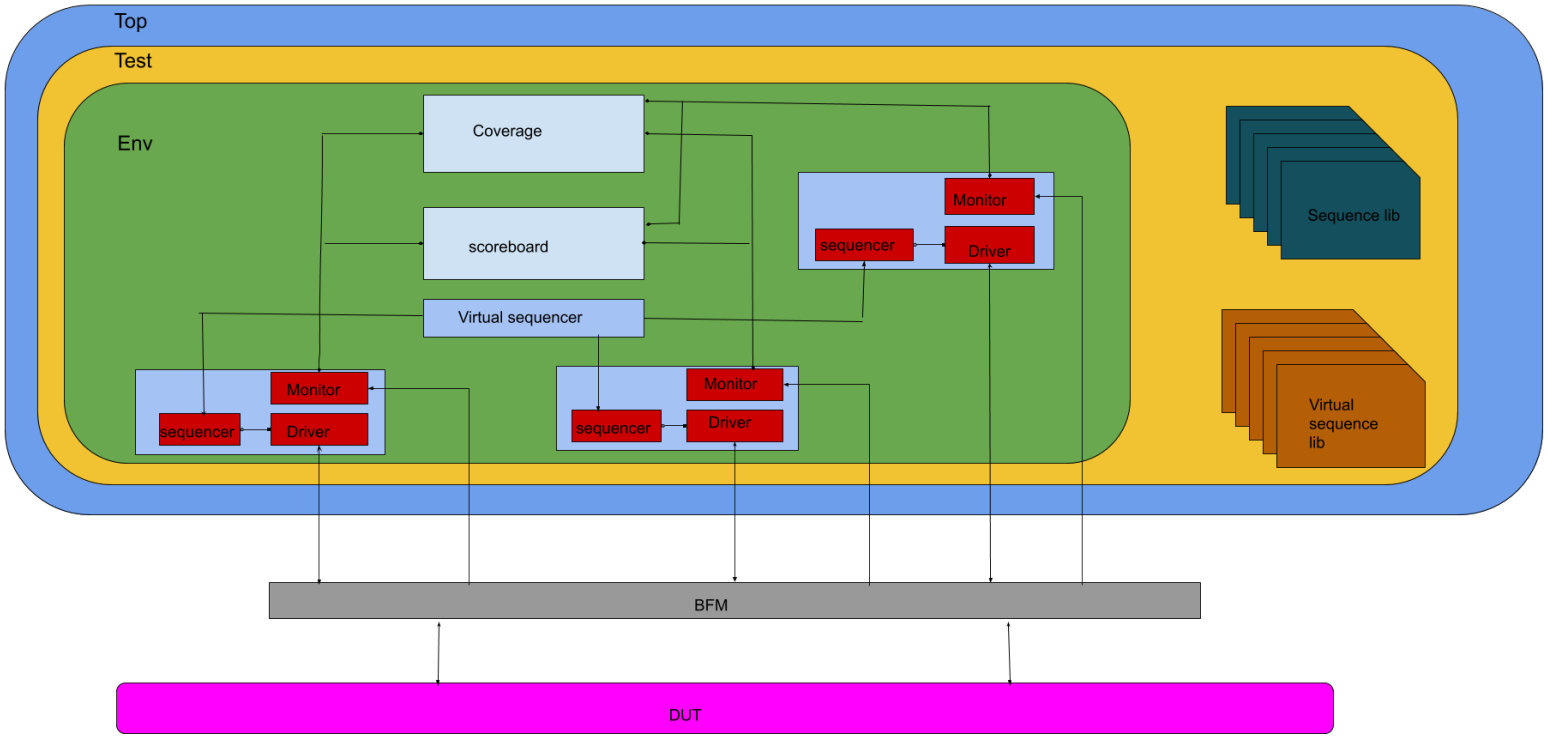


Figure 5.1: Ibex UVM Verification Environment Architecture.

5.1 Ibex Verification Plan

verification plan addresses the items to be verified, checked and covered .the verification plan for Ibex processor will address the items to be verified, including the ISA, the IOs, environment (e.g., ISA mix, memory types,operations etc). it also addresses how the items that need to be verified will be checked . the following sections include tables shows the extracting items with its methodology to verify, cover and check .

5.2 Random Instruction Generator

By extracting processor design specifications from their respective standard files , the input instructions are generated by the Instruction Generator as seen in Figure 5.2. The IG generates random sequences of instructions and is coded in System Verilog.. The IG generates instruction of a particular kind like only the data manipulation instructions, branch instructions, data transfer instructions or any other valid combination can be created.Apart from generating random sequences of instructions, the IG also executes every instruction and writes the computed result in the respective virtual sequence component .

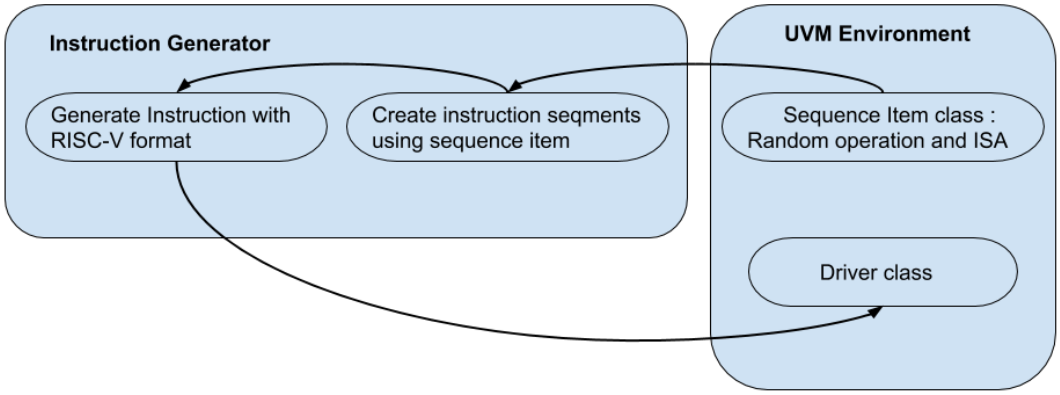


Figure 5.2: Instruction Generator communication with UVM Environment.

5.3 Ibex Interface

The Interface construct in System Verilog is a bundle of wires that contain connectivity which encapsulates the communication between blocks check figure 5.3. The interface consists of all the input and output ports from the DUT. The input ports of DUT are only the Clock and Reset and the rest of the input ports and output ports of DUT. The logic type is used in declaring the signals so it can be driven from procedural statements. The interface is instantiated in the top module as a virtual interface . One of the main advantages of using interface is that when a new signal is added it can be declared only once in the interface and can be accessed by higher level modules with the right reference path.

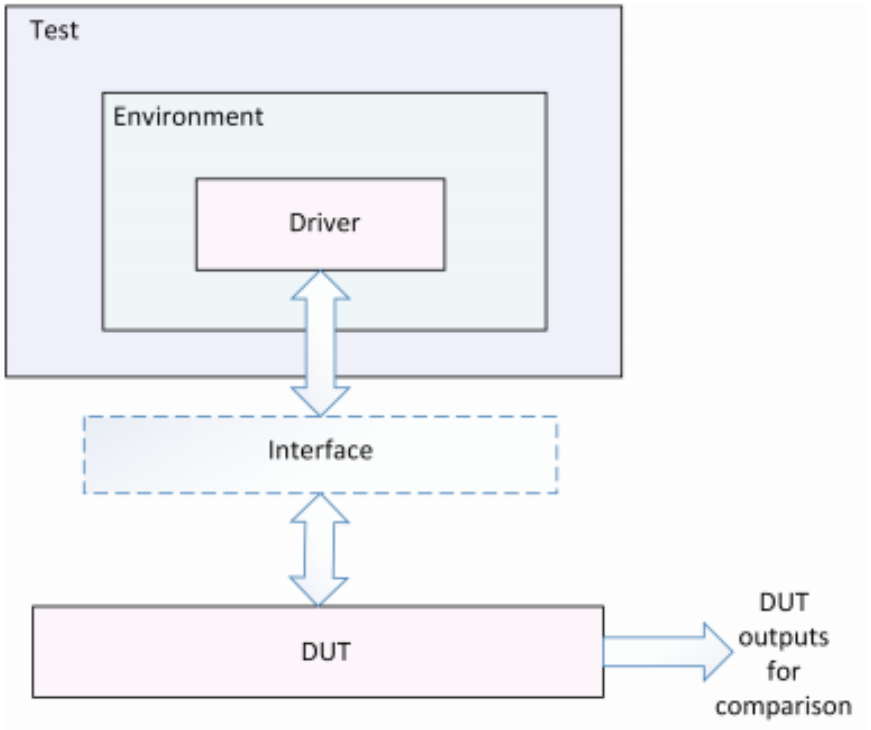


Figure 5.3: Test bench components connection through the interface .

5.4 Ibex transaction

The data flows through the testbench from component to component in the form of packets called as transaction class or sequence item. Three Ibex sequence item classes are created by extending the uvm sequence item class .the first one for driving instruction set and communicate with the core as instruction memory and the other for driving data memory instructions and the last one for driving interrupt signals. The transaction packets consists of configuration inputs to dedicate the type of instruction and control inputs (error bit, valid bit and fetch enable bit) and control knobs. Then register the class and properties to factory using uvm object utils macro. A constructor function is defined for the sequence item. Randomization is applied to sequence items with proper constraints to each class .

5.5 Ibex sequence library

The user-defined Ibex sequence classes use uvm sequence as its virtual base class. These classes are parameterized class with the parameter being one of the Ibex sequence items classes . Body() method is called, and code within this method gets executed when the sequence is run. Within the body() the data of sequence items are constrained randomized and Then start item is called to begin the interaction with the sequencer. At this point the sequencer halts the execution of the sequence until the driver is ready. Once the driver is ready, the sequencer causes start item to return. Once start item has returned, then this sequence has been granted permission to use the driver. After the transaction has been randomized, and the data values set, it is sent to the driver for processing using finish item. Finish item

should really be called execute item. At this time, the driver gets the transaction handle and will execute it. Once the driver calls item done (), then finish item will return and the transaction has been executed figure 5.4 shows the communication between Sequence and Driver through the sequencer .

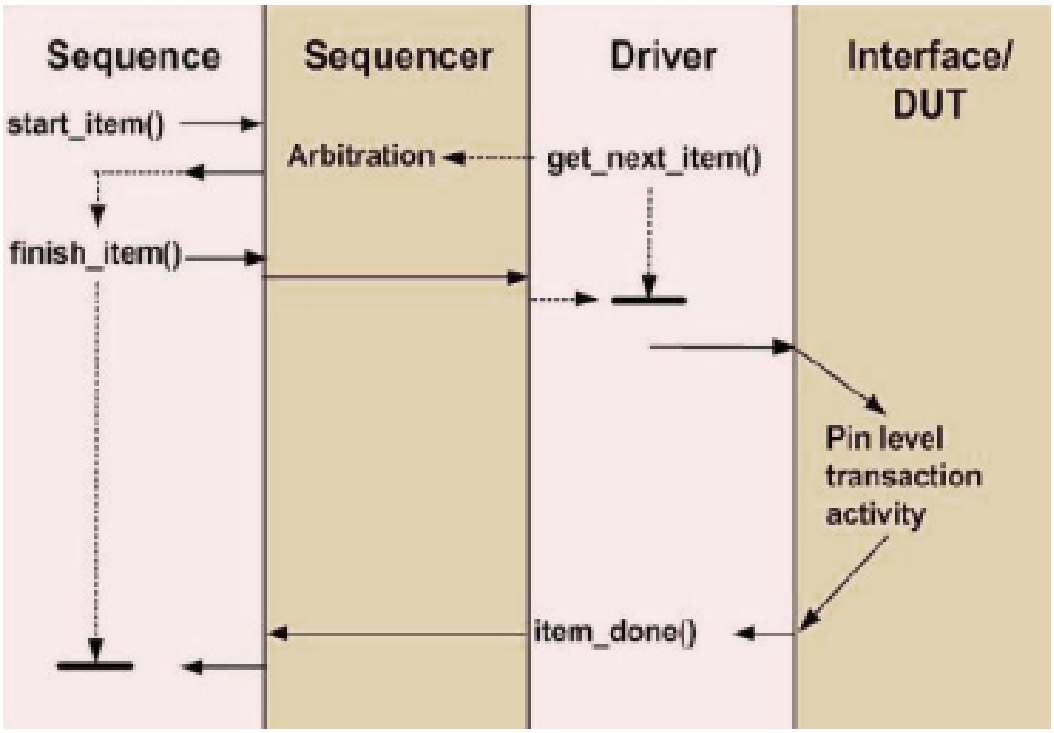


Figure 5.4: UVM Sequence and Driver Communication.

5.6 Ibex virtual sequence library

This library contains multiple classes each class declare one or more of Ibex sequence that can be driven parallel or sequential and it is very important as it used to handle drive two different agents in the same time or sequential .

5.7 Ibex virtual sequencer and Ibex sequencer

Ibex virtual sequencer is the component that contains all sequencers which exists in the whole environment. the role of sequencer class is to run the sequences. The sequencer has a built-in port called sequence item export to communicate with the driver. Through this port, the sequencer can send a request item to the driver and receive a response item from the driver. This class is parameterized with Ibex sequence items used .

5.8 Ibex Agent

The agent encompasses the sequencer, driver and the monitor classes of the Ibex test bench architecture. This class connects the TLM interface between the sequencer and the driver that is vital to allow the flow of transactions to the driver. Ibex environment contains three Active agents each drive and monitor different signals or different inputs to the Dut you can check the following table 5.1 that shows the signals and how to drive and the main component in UVM environment that is responsible for drive it.

The first agent is called instruction agent it is responsible to diver and monitor the instructions related to mathematical ,logical ,branching and jumping signals .the sequencer in this agent is parameterized by instruction sequence item class .the second agent is data memory agent which deal with the Dut as a data memory by providing the proper value for the corresponding address . this agent is only drive and monitor load and store instructions. the last agent is interrupt agent and its role to drive different interrupt signals and provide different scenarios of driving on the Dut . all of these agents are controled by their configuration objects

in the upper layer, the following waveforms describe signals driving by test(1) that targeting instruction agent ,check figure 5.5,and figure 5.6 that targeting data memory agent .

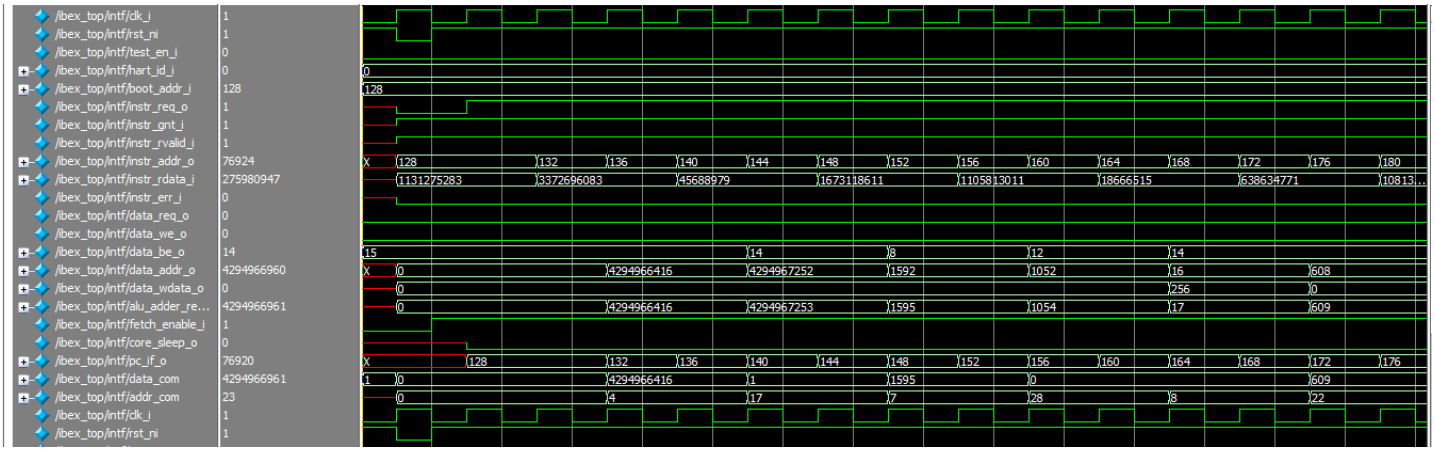


Figure 5.5: Driving instruction Agent by Test 1.

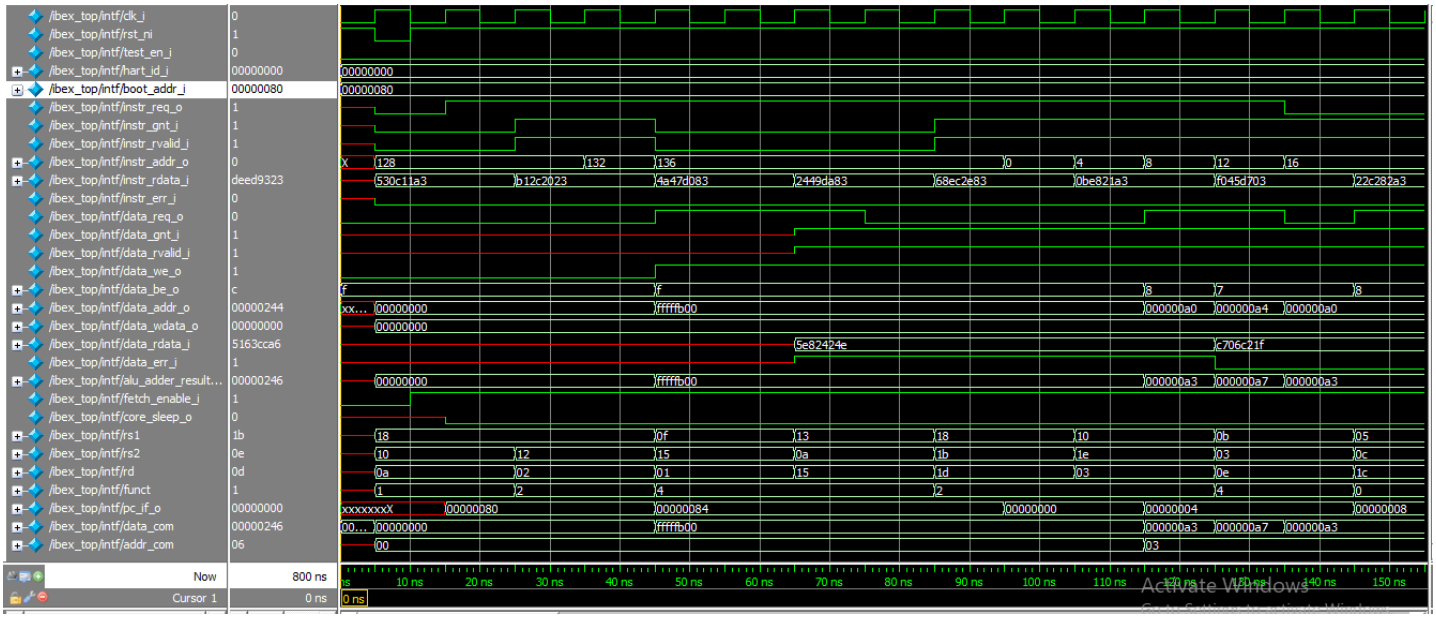


Figure 5.6: Driving data memory Agent by Test 2.

SPEC-ID	DIR	SIGNAL-NAME	DRIVE-VR
general	IN	clk_i	in top.sv
general	IN	rst_ni	set the system under reset
general	IN	hart_id_i	set Hart ID as a parameter in the top module
general	IN	boot_addr_i	set address as a parameter in the top module
general	IN	fetch_enable_i	-set high when drive instruction -deassert signal during running
general	OUT	core_sleep_o	drive WFI instruction
general	OUT	instr_req_o	
instr_agent	OUT	instr_addr_o[31:0]	
instr_agent	IN	instr_gnt_i	-drive high for one cycle when the request is high -rarely reject the request
instr_agent	IN	instr_rvalid_i	-drive high for one cycle when the grant is high -rarely deasset the signal
instr_agent	IN	instr_rdata_i[31:0]	drive accorrding to instr_gen class
instr_agent	IN	instr_err_i	-drive high when fetch instruction -rarely deasset the signal
dm_agent	OUT	data_req_o	
dm_agent	OUT	data_addr_o[31:0]	
dm_agent	OUT	data_we_o	
dm_agent	OUT	data_be_o[3:0]	
dm_agent	OUT	data_wdata_o[31:0]	
dm_agent	IN	data_gnt_i	
dm_agent	IN	data_rvalid_i	
dm_agent	IN	data_err_i	

dm_agent	IN	data_rdata_i[31:0]	
csr_agent	EMB	mstatus	
csr_agent	EMB	misa	
csr_agent	EMB	mie	
csr_agent	EMB	mtvec	
csr_agent	EMB	mepc	
csr_agent	EMB	mcause	drive interrupt and exceptions
csr_agent	EMB	mtval	drive illegal instructions
csr_agent	EMB	mip	drive nested interrupts
csr_agent	EMB	mcycle	counter the number of cycles
csr_agent	EMB	mhartid	random drive / irq_seq
irq_agent	IN	irq_nm_i	random drive / irq_seq
irq_agent	IN	irq_fast_i[14:0]	random drive / irq_seq
irq_agent	IN	irq_external_i	random drive / irq_seq
irq_agent	IN	irq_timer_i	random drive / irq_seq
irq_agent	IN	irq_software_i	random drive / irq_seq
instr_agent	IN	LUI rd,imm	random drive / instr_seq
instr_agent	IN	AUIPC rd,offset	random drive / instr_seq
instr_agent	IN	JAL rd,offset	random drive / instr_seq
instr_agent	IN	JALR rd,rs1,offset	random drive / instr_seq
instr_agent	IN	BEQ rs1,rs2,offset	random drive / instr_seq
instr_agent	IN	BNE rs1,rs2,offset	random drive / instr_seq
instr_agent	IN	BLT rs1,rs2,offset	random drive / instr_seq
instr_agent	IN	BGE rs1,rs2,offset	random drive / instr_seq
instr_agent	IN	BLTU rs1,rs2,offset	random drive / instr_seq

instr_agent	IN	BGEU rs1,rs2,offset	random drive / instr_seq
dm_agent	IN	LB rd,offset(rs1)	random drive /dm_seq
dm_agent	IN	LH rd,offset(rs1)	random drive /dm_seq
dm_agent	IN	LW rd,offset(rs1)	random drive /dm_seq
dm_agent	IN	LBU rd,offset(rs1)	random drive /dm_seq
dm_agent	IN	LHU rd,offset(rs1)	random drive /dm_seq
dm_agent	IN	SB rs2,offset(rs1)	random drive /dm_seq
dm_agent	IN	SH rs2,offset(rs1)	random drive /dm_seq
dm_agent	IN	SW rs2,offset(rs1)	random drive /dm_seq
instr_agent	IN	ADDI rd,rs1,imm	random drive / instr_seq
instr_agent	IN	SLTI rd,rs1,imm	random drive / instr_seq
instr_agent	IN	SLTIU rd,rs1,imm	random drive / instr_seq
instr_agent	IN	XORI rd,rs1,imm	random drive / instr_seq
instr_agent	IN	ORI rd,rs1,imm	random drive / instr_seq
instr_agent	IN	ANDI rd,rs1,imm	random drive / instr_seq
instr_agent	IN	SLLI rd,rs1,imm	random drive / instr_seq
instr_agent	IN	SRLI rd,rs1,imm	random drive / instr_seq
instr_agent	IN	SRAI rd,rs1,imm	random drive / instr_seq
instr_agent	IN	ADD rd,rs1,rs2	random drive / instr_seq
instr_agent	IN	SUB rd,rs1,rs2	random drive / instr_seq
instr_agent	IN	SLL rd,rs1,rs2	random drive / instr_seq
instr_agent	IN	SLT rd,rs1,rs2	random drive / instr_seq
instr_agent	IN	SLTU rd,rs1,rs2	random drive / instr_seq
instr_agent	IN	XOR rd,rs1,rs2	random drive / instr_seq
instr_agent	IN	SRL rd,rs1,rs2	random drive / instr_seq
instr_agent	IN	SRA rd,rs1,rs2	random drive / instr_seq
instr_agent	IN	OR rd,rs1,rs2	random drive / instr_seq
instr_agent	IN	AND rd,rs1,rs2	random drive / instr_seq

Table 5.1 Signals Driving

5.9 Ibex scoreboard

Ibex scoreboard is the component which has transaction level checkers to verify the functional correctness of a given DUT. Scoreboard class is extended from the uvm scoreboard base class.Three TLM analysis FIFOs are connected to the monitor and this way called Twitter pattern system check the following figure 5.7 .

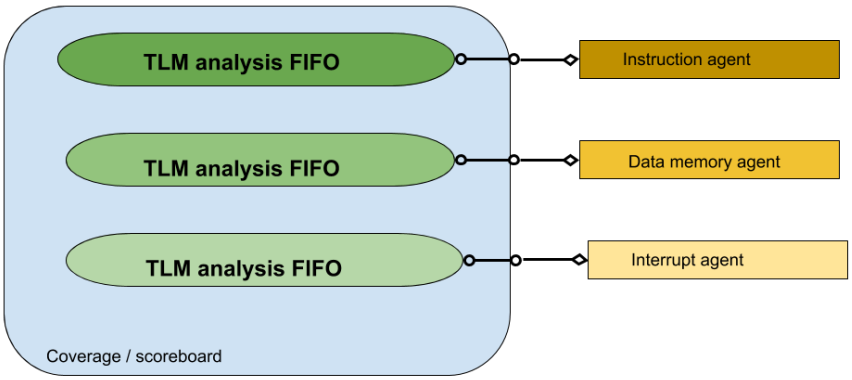


Figure 5.7: analysis FIFO Connection.

In the run phase, the input packet and the output packet is retrieved from the monitor and once deliver to scoreboad it will be connect with analysis fifos suitable with transaction type . the checking happen through calling an external function in scoreboard class that work as reference model and compare the resulting output with the actual output from the Dut, the following table 5.2 that shows the signals and how to check and the main component in UVM environment that is responsible for drive it.

SPEC-ID	DIR	SIGNAL-NAME	CHECK-VR
general	IN	clk_i	system is synchronous with the clk
general	IN	rst_ni	the outputs of the system is zeroes
general	IN	hart_id_i	check the Hardware Thread ID csr register
general	IN	boot_addr_i	pc register in the start operating the system
general	IN	fetch_enable_i	always high when fetch instruction
general	OUT	core_sleep_o	low after waiting otherwise high
general	OUT	instr_req_o	always high when fetch instruction
instr_agent	OUT	instr_addr_o[31:0]	-legal aligned address -start with boot address
instr_agent	IN	instr_gnt_i	
instr_agent	IN	instr_rvalid_i	
instr_agent	IN	instr_rdata_i[31:0]	
instr_agent	IN	instr_err_i	
dm_agent	OUT	data_req_o	always high when fetch instruction
dm_agent	OUT	data_addr_o[31:0]	-legal aligned address accodring to instruction
dm_agent	OUT	data_we_o	check request is vaild when driving dm instructions
dm_agent	OUT	data_be_o[3:0]	
dm_agent	OUT	data_wdata_o[31:0]	
dm_agent	IN	data_gnt_i	
dm_agent	IN	data_rvalid_i	
dm_agent	IN	data_err_i	

dm_agent	IN	data_rdata_i[31:0]	
csr_agent	EMB	mstatus	check mstatus register in csr
csr_agent	EMB	misa	check mhart register in csr
csr_agent	EMB	mie	check loacal and global mip register in csr
csr_agent	EMB	mtvec	check mtvec register with pc when interrupt occure
csr_agent	EMB	mepc	divide by zero and check pc register in csr
csr_agent	EMB	mcause	check mcause register during interrupt and exceptions
csr_agent	EMB	mtval	check the last address in csr
csr_agent	EMB	mip	check pmie according to pirority of the interupt
csr_agent	EMB	mcycle	compare the this register with the counter implemented
csr_agent	EMB	mhartid	chec mip / pmie / mtvec / mstatus /mcause regs
irq_agent	IN	irq_nm_i	chec mip / pmie / mtvec / mstatus /mcause regs
irq_agent	IN	irq_fast_i[14:0]	chec mip / pmie / mtvec / mstatus /mcause regs
irq_agent	IN	irq_external_i	chec mip / pmie / mtvec / mstatus /mcause regs
irq_agent	IN	irq_timer_i	-chec mip / pmie / mtvec / mstatus /mcause regs - check mtimer register
irq_agent	IN	irq_software_i	check mip / pmie / mtvec / mstatus /mcause regs
instr_agent	IN	LUI rd,imm	check register_file / pc_counter
instr_agent	IN	AUIPC rd,offset	check register_file / pc_counter
instr_agent	IN	JAL rd,offset	check register_file / pc_counter
instr_agent	IN	JALR rd,rs1,offset	check register_file / pc_counter
instr_agent	IN	BEQ rs1,rs2,offset	check register_file / pc_counter
instr_agent	IN	BNE rs1,rs2,offset	check register_file / pc_counter
instr_agent	IN	BLT rs1,rs2,offset	check register_file / pc_counter
instr_agent	IN	BGE rs1,rs2,offset	check register_file / pc_counter
instr_agent	IN	BLTU rs1,rs2,offset	check register_file / pc_counter

instr_agent	IN	BGEU rs1,rs2,offset	check register_file / pc_counter
dm_agent	IN	LB rd,offset(rs1)	check register_file / data_memory
dm_agent	IN	LH rd,offset(rs1)	check register_file / data_memory
dm_agent	IN	LW rd,offset(rs1)	check register_file / data_memory
dm_agent	IN	LBU rd,offset(rs1)	check register_file / data_memory
dm_agent	IN	LHU rd,offset(rs1)	check register_file / data_memory
dm_agent	IN	SB rs2,offset(rs1)	check register_file / data_memory
dm_agent	IN	SH rs2,offset(rs1)	check register_file / data_memory
dm_agent	IN	SW rs2,offset(rs1)	check register_file / data_memory
instr_agent	IN	ADDI rd,rs1,imm	check register_file / pc_counter
instr_agent	IN	SLTI rd,rs1,imm	check register_file / pc_counter
instr_agent	IN	SLTIU rd,rs1,imm	check register_file / pc_counter
instr_agent	IN	XORI rd,rs1,imm	check register_file / pc_counter
instr_agent	IN	ORI rd,rs1,imm	check register_file / pc_counter
instr_agent	IN	ANDI rd,rs1,imm	check register_file / pc_counter
instr_agent	IN	SLLI rd,rs1,imm	check register_file / pc_counter
instr_agent	IN	SRLI rd,rs1,imm	check register_file / pc_counter
instr_agent	IN	SRAI rd,rs1,imm	check register_file / pc_counter
instr_agent	IN	ADD rd,rs1,rs2	check register_file / pc_counter
instr_agent	IN	SUB rd,rs1,rs2	check register_file / pc_counter
instr_agent	IN	SLL rd,rs1,rs2	check register_file / pc_counter
instr_agent	IN	SLT rd,rs1,rs2	check register_file / pc_counter
instr_agent	IN	SLTU rd,rs1,rs2	check register_file / pc_counter
instr_agent	IN	XOR rd,rs1,rs2	check register_file / pc_counter
instr_agent	IN	SRL rd,rs1,rs2	check register_file / pc_counter
instr_agent	IN	SRA rd,rs1,rs2	check register_file / pc_counter
instr_agent	IN	OR rd,rs1,rs2	check register_file / pc_counter
instr_agent	IN	AND rd,rs1,rs2	check register_file / pc_counter

Table 5.2 Signals Checking

5.10 Ibex coverage

5.10.1 Functional coverage

Functional coverage is essential to any verification plan,and a way to tell the effectiveness of the test plan. Functional coverage is done by creating multiple coverage groups each is responsible for cover specific inputs and outputs from the Dut Figure 8 shows the mechanism of Functional Coverage

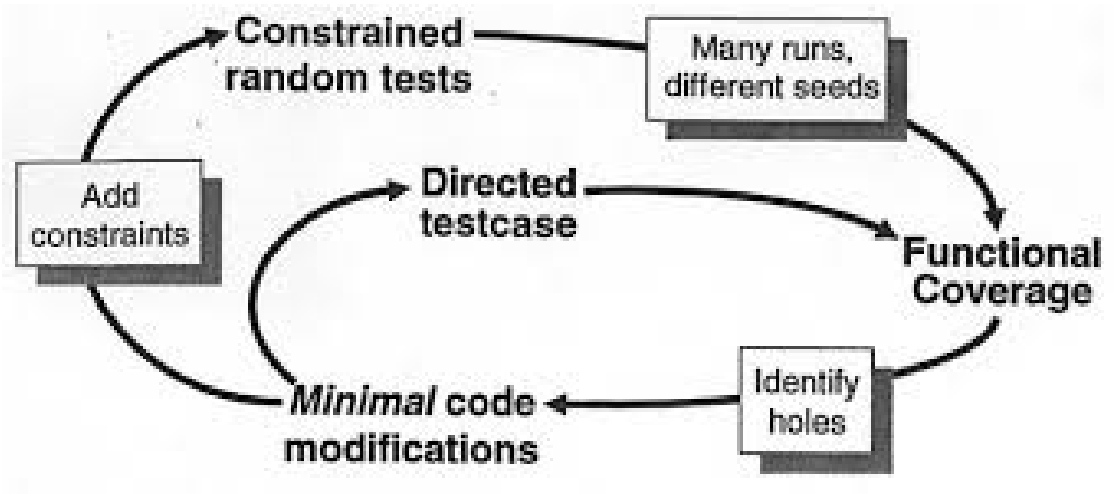


Figure 5.8: Functional Covergae Loop.

I created seven coverage groups ,three groups for instructions related to different operations deal with Alu .two groups for data memory instructions related to load and store data by using data memory . one group for cover interrupt signals.the last group for transitions between different instructions.Figure 9 states the coverage percentage Results , and then there is a table 5.3 describe signals coverage items .

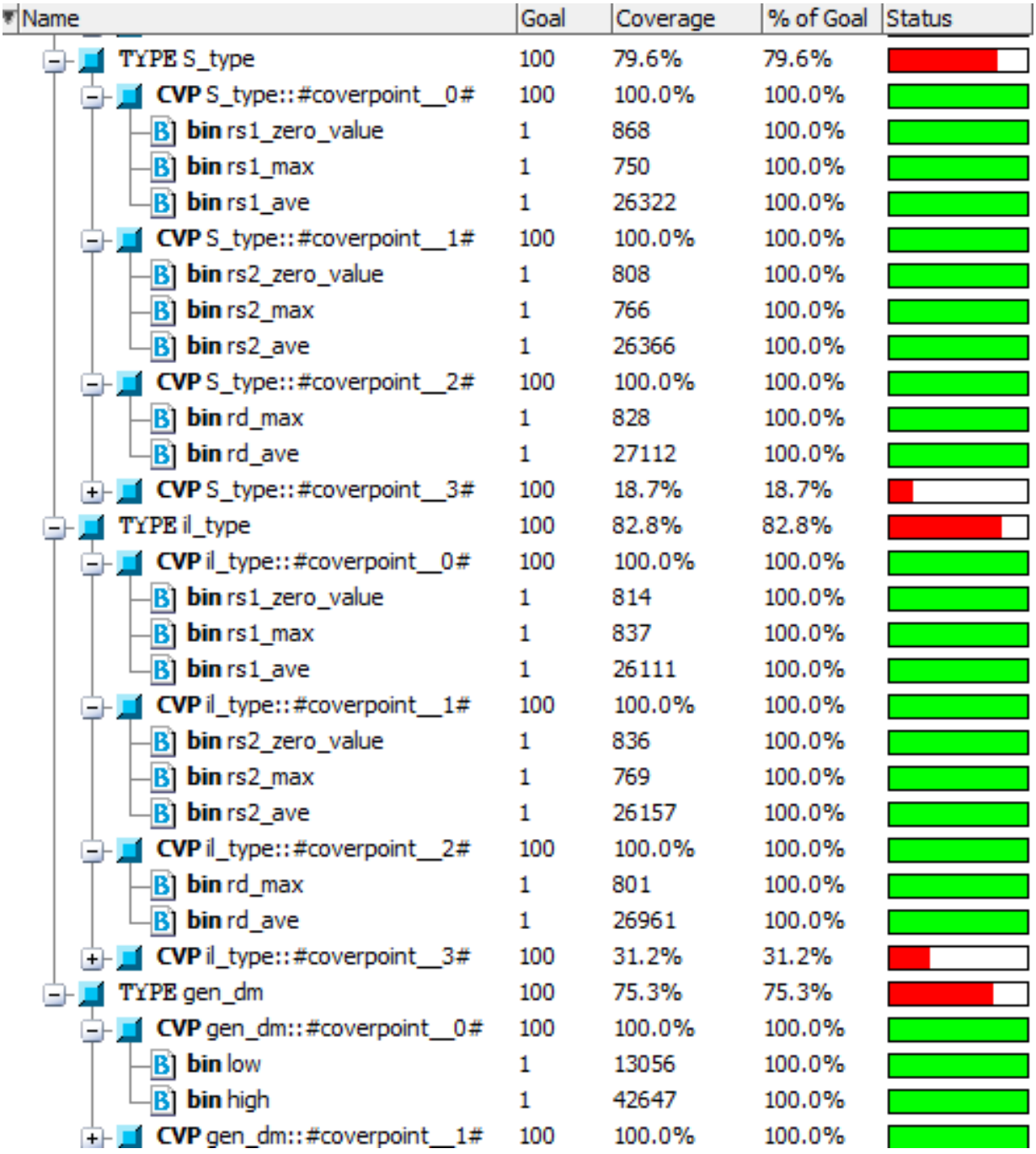


Figure 5.9: Coverage groups percentage result .

SPEC-ID	DIR	SIGNAL-NAME	COVER-VR
general	IN	clk_i	
general	IN	rst_ni	
general	IN	hart_id_i	
general	IN	boot_addr_i	
general	IN	fetch_enable_i	-check 0 / 1 -check zero during run instructions
general	OUT	core_sleep_o	check 0 / 1
general	OUT	instr_req_o	
instr_agent	OUT	instr_addr_o[31:0]	-different range of address -check zero in the start
instr_agent	IN	instr_gnt_i	-check 0 /1 on different instruction
instr_agent	IN	instr_rvalid_i	-check 0 /1 on different instruction
instr_agent	IN	instr_rdata_i[31:0]	-check transtions among instructions
instr_agent	IN	instr_err_i	-check 0 /1 on different instruction
dm_agent	OUT	data_req_o	-check 0/1 on different times and instructions
dm_agent	OUT	data_addr_o[31:0]	-check different range of instructions
dm_agent	OUT	data_we_o	-check 0/1
dm_agent	OUT	data_be_o[3:0]	-check 00/01/10/11 with load and store instructions
dm_agent	OUT	data_wdata_o[31:0]	-check max/min/ave data
dm_agent	IN	data_gnt_i	-check 0/1 during running
dm_agent	IN	data_rvalid_i	-check 0/1 during running
dm_agent	IN	data_err_i	-check 0/1 during running on different instructions

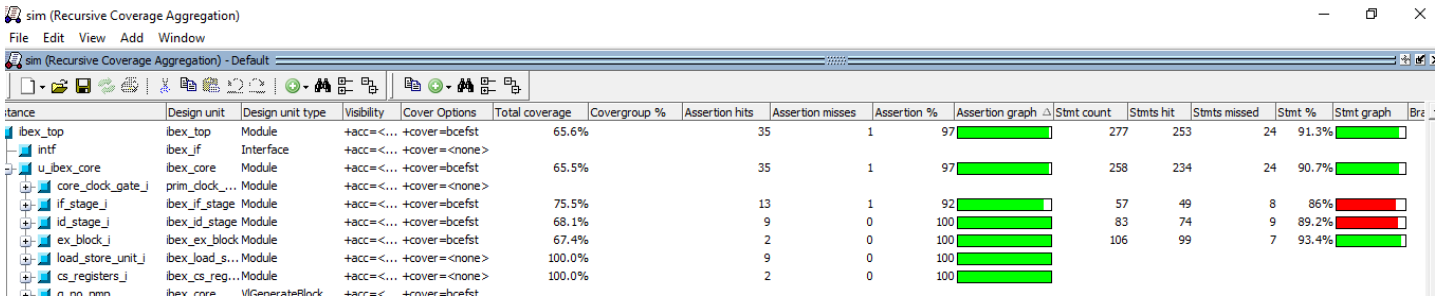
dm_agent	IN	data_rdata_i[31:0]	-check read from top and down from data memory -check read max/min/ ave values -check ave values from dm
csr_agent	EMB	mstatus	-cover read with different instructions
csr_agent	EMB	misa	-cover read with different instructions
csr_agent	EMB	mie	-cover with different interrupt signals
csr_agent	EMB	mtvec	-cover with different interrupt signals
csr_agent	EMB	mepc	-cover with differentdivion operation
csr_agent	EMB	mcause	-cover with different interrupt signals & exceptions
csr_agent	EMB	mtval	
csr_agent	EMB	mip	-cover with different interrupt signals
csr_agent	EMB	mcycle	-cover with different times with different config timer interrupt signal
csr_agent	EMB	mhartid	-check it with different time during different instructions
irq_agent	IN	irq_nm_i	-check it with different time during different instructions
irq_agent	IN	irq_fast_i[14:0]	-check it with different time during different instructions
irq_agent	IN	irq_external_i	-check it with different time during different instructions
irq_agent	IN	irq_timer_i	-check it with different time during different instructions -put time less than current time -put time the same as current time -put time greater than the current time (max - ave)
irq_agent	IN	irq_software_i	-check it with different time during different instructions
instr_agent	IN	LUI rd,imm	1-check min / max / ave (rs1-rs2-rd) 2-check all different functions 3-cross no.1 & no.2 4-using it before addi instruction with values : -abce + 401
instr_agent	IN	AUIPC rd,offset	1-check min / max / ave (rs1-rs2-rd) 2-check all different functions 3-cross no.1 & no.2

instr_agent	IN	JAL rd,offset	1-check min / max / ave (rs1-rs2-rd) 2-check all different functions 3-cross no.1 & no.2
instr_agent	IN	JALR rd,rs1,offset	1-check min / max / ave (rs1-rs2-rd) 2-check all different functions 3-cross no.1 & no.2
instr_agent	IN	BEQ rs1,rs2,offset	1-check min / max / ave (rs1-rs2-rd) 2-check all different functions 3-cross no.1 & no.2
instr_agent	IN	BNE rs1,rs2,offset	1-check min / max / ave (rs1-rs2-rd) 2-check all different functions 3-cross no.1 & no.2
instr_agent	IN	BLT rs1,rs2,offset	1-check min / max / ave (rs1-rs2-rd) 2-check all different functions 3-cross no.1 & no.2
instr_agent	IN	BGE rs1,rs2,offset	1-check min / max / ave (rs1-rs2-rd) 2-check all different functions 3-cross no.1 & no.2
instr_agent	IN	BLTU rs1,rs2,offset	1-check min / max / ave (rs1-rs2-rd) 2-check all different functions 3-cross no.1 & no.2
instr_agent	IN	BGEU rs1,rs2,offset	1-check min / max / ave (rs1-rs2-rd) 2-check all different functions 3-cross no.1 & no.2
dm_agent	IN	LB rd,offset(rs1)	1-check min / max / ave (rs1-rs2-rd) 2-check all different functions 3-cross no.1 & no.2
dm_agent	IN	LH rd,offset(rs1)	1-check min / max / ave (rs1-rs2-rd) 2-check all different functions 3-cross no.1 & no.2
dm_agent	IN	LW rd,offset(rs1)	1-check min / max / ave (rs1-rs2-rd) 2-check all different functions 3-cross no.1 & no.2
dm_agent	IN	LBU rd,offset(rs1)	1-check min / max / ave (rs1-rs2-rd) 2-check all different functions 3-cross no.1 & no.2

Table 5.3 Signals Coverage

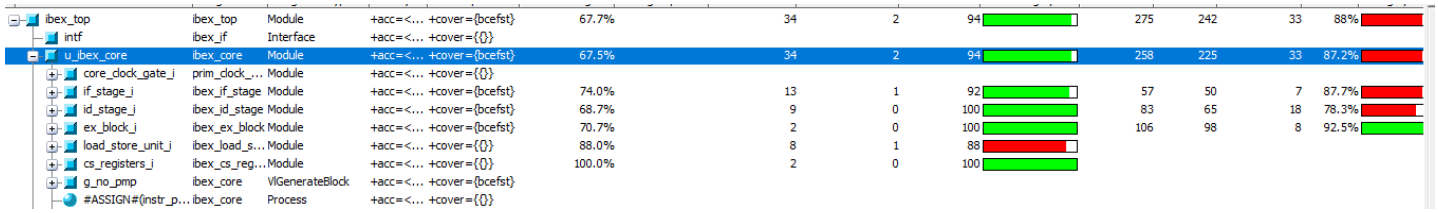
5.10.2 Code coverage

Code coverage tracks information such what lines of code or expression or block have been exercised. However, code coverage is not exhaustive and cannot detect conditions that not present in the code. To address these deficiencies, we go for functional coverage.Figure 5.10 and figure 5.12 show the coverage resulting from test 1 ,and figure 5.11 and figure 5.13 show the coverage resulting from test 2.



itance	Design unit	Design unit type	Visibility	Cover Options	Total coverage	Covergroup %	Assertion hits	Assertion misses	Assertion %	Assertion graph	Stmt count	Stmts hit	Stmts missed	Stmt %	Stmt graph
ibex_top	ibex_top	Module	+acc=<...	+cover=bcefst	65.6%		35	1	97		277	253	24	91.3%	
intf	ibex_if	Interface	+acc=<...	+cover=<none>											
u_ibex_core	ibex_core	Module	+acc=<...	+cover=bcefst	65.5%		35	1	97		258	234	24	90.7%	
core_clock_gate_i	prim_clock_...	Module	+acc=<...	+cover=<none>											
if_stage_i	ibex_if_stage	Module	+acc=<...	+cover=bcefst	75.5%		13	1	92		57	49	8	86%	
id_stage_i	ibex_id_stage	Module	+acc=<...	+cover=bcefst	68.1%		9	0	100		83	74	9	89.2%	
ex_block_i	ibex_ex_block	Module	+acc=<...	+cover=bcefst	67.4%		2	0	100		106	99	7	93.4%	
load_store_unit_i	ibex_load_s...	Module	+acc=<...	+cover=<none>	100.0%		9	0	100						
cs_registers_i	ibex_cs_reg...	Module	+acc=<...	+cover=<none>	100.0%		2	0	100						

Figure 5.10: code coverage results by test 1 .



ibex_top	ibex_top	Module	+acc=<...	+cover={bcefst}	67.7%	34	2	94	275	242	33	88%	
intf	ibex_if	Interface	+acc=<...	+cover={}									
u_ibex_core	ibex_core	Module	+acc=<...	+cover={bcefst}	67.5%	34	2	94	258	225	33	87.2%	
core_clock_gate_i	prim_clock_...	Module	+acc=<...	+cover={}									
if_stage_i	ibex_if_stage	Module	+acc=<...	+cover={bcefst}	74.0%	13	1	92	57	50	7	87.7%	
id_stage_i	ibex_id_stage	Module	+acc=<...	+cover={bcefst}	68.7%	9	0	100	83	65	18	78.3%	
ex_block_i	ibex_ex_block	Module	+acc=<...	+cover={bcefst}	70.7%	2	0	100	106	98	8	92.5%	
load_store_unit_i	ibex_load_s...	Module	+acc=<...	+cover={}	88.0%	8	1	88					
cs_registers_i	ibex_cs_reg...	Module	+acc=<...	+cover={}	100.0%	2	0	100					
g_no_pmp	ibex_core	VlGenerateBlock	+acc=<...	+cover={bcefst}									
#ASSIGN#(instr_p...	ibex_core	Process	+acc=<...	+cover={}									

Figure 5.11: code coverage results by test 2.

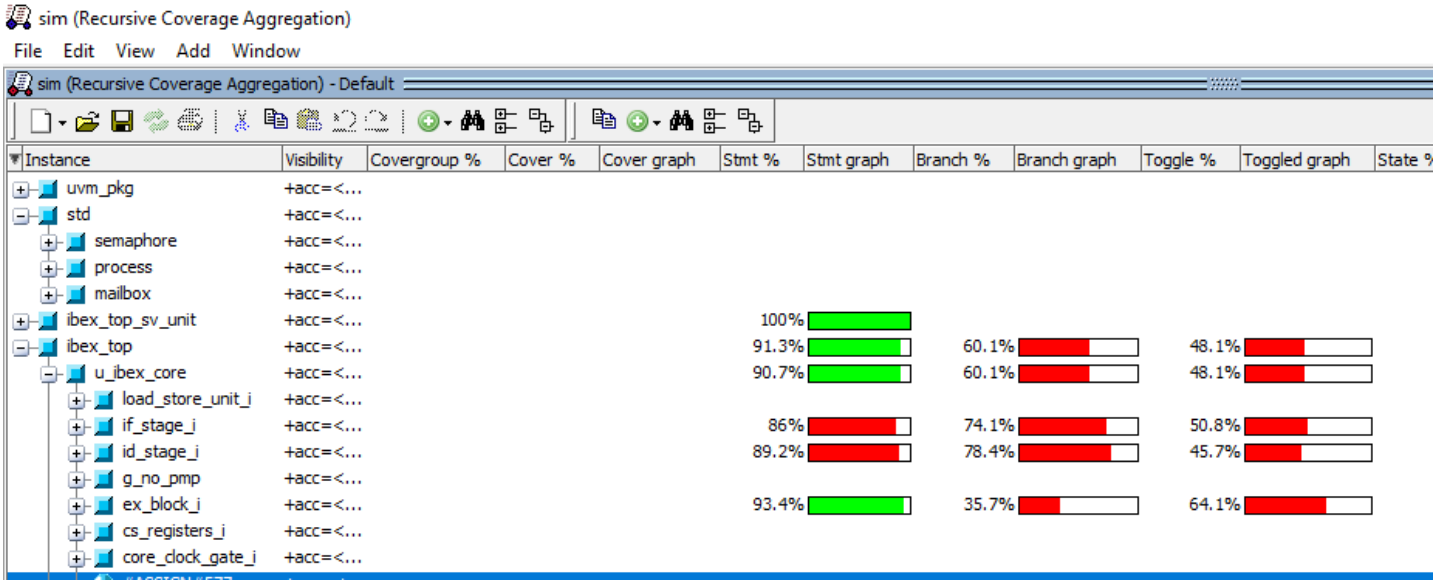


Figure 5.12: code coverage results by test 1.

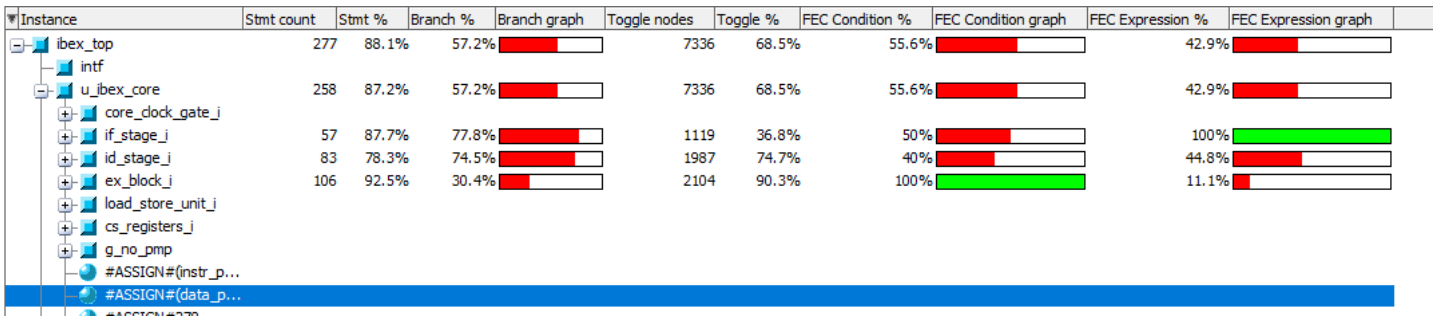


Figure 5.13: code coverage results by test 2.

5.11 Ibex environment

Ibex environment is a container component containing three different agents, score-board and coverage . It is created using uvm env virtual base class. In the build phase components within the environment are instantiated and configured using its configuration objects. And in the connect phase, the connections are made

between components.

5.12 Ibex base test

The test class is created by extending the uvm test class. Then the class is registered to factory using uvm component utils macro. In the build phase, the lower level Ibex environment class is created and configured by its the configuration object and the same for the agent classes ,coverage and scoreboard . this class is extended by any test by running specific virtual sequence .

5.13 Ibex Top

The top-level module is responsible for integrating the testbench module with the device under test. This module instantiates the interface which is wired with Ibex core The top module also generates the clock and registers the interface into the config database so that other subscribing blocks can retrieve. Finally, the module calls the run test function which starts to run the uvm root.

5.14 Test the configurability of the verification environment

To increase the configurability of the verification system I defined the constraints of randomization items as soft and this feature enable us to violate these constraints or override them in any sequence used ,in addition the implementation of this environment that has been described in the previous sections enable us to disable any component or add any new one in the upper layer of the environment

and without going into details of how these components communicate with each other and this is done just using configuration objects which is created for each agent.another feature,I defined three macros for each agent in configuration file which enable us to to handle driving two different agents in the same time ,sequential or just only one .the same for the number of bits used I defined parameter to dedicate bit sizes 32-bit or 64-bit ,I tried to test all of these features by test different core with the same verification environment like RI5CY processor, is one of pulp family but slightly different from Ibex as it is 32 bit 4-stage core with floating point unit,with two different methods.Figure 5.14 show the design architecture for RI5CY core ,Figure 5.15 show UVM Architecture for RI5CY processor First method,I used the instruction agent and some of sequence classes to drive RI5CY core with adding the different signals . Second method, One of the main advantage of UVM is Creating each components using factory enables them to be overridden in different tests or environments without changing underlying code base ,I uesd this feature to override instruction agent created for Ibex processor with customized one for RI5CY processor.

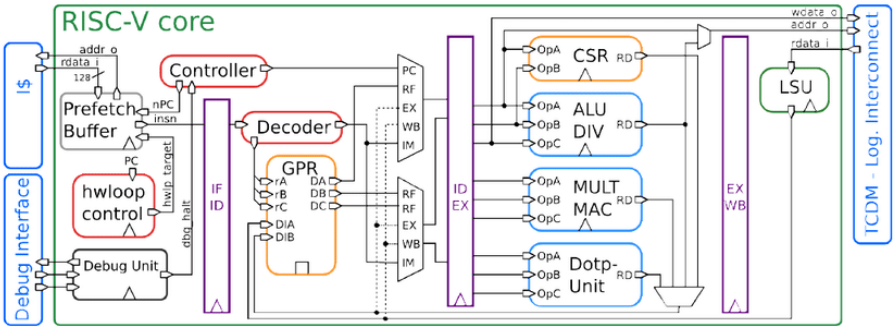


Figure 5.14: RI5CY processor Architecture .

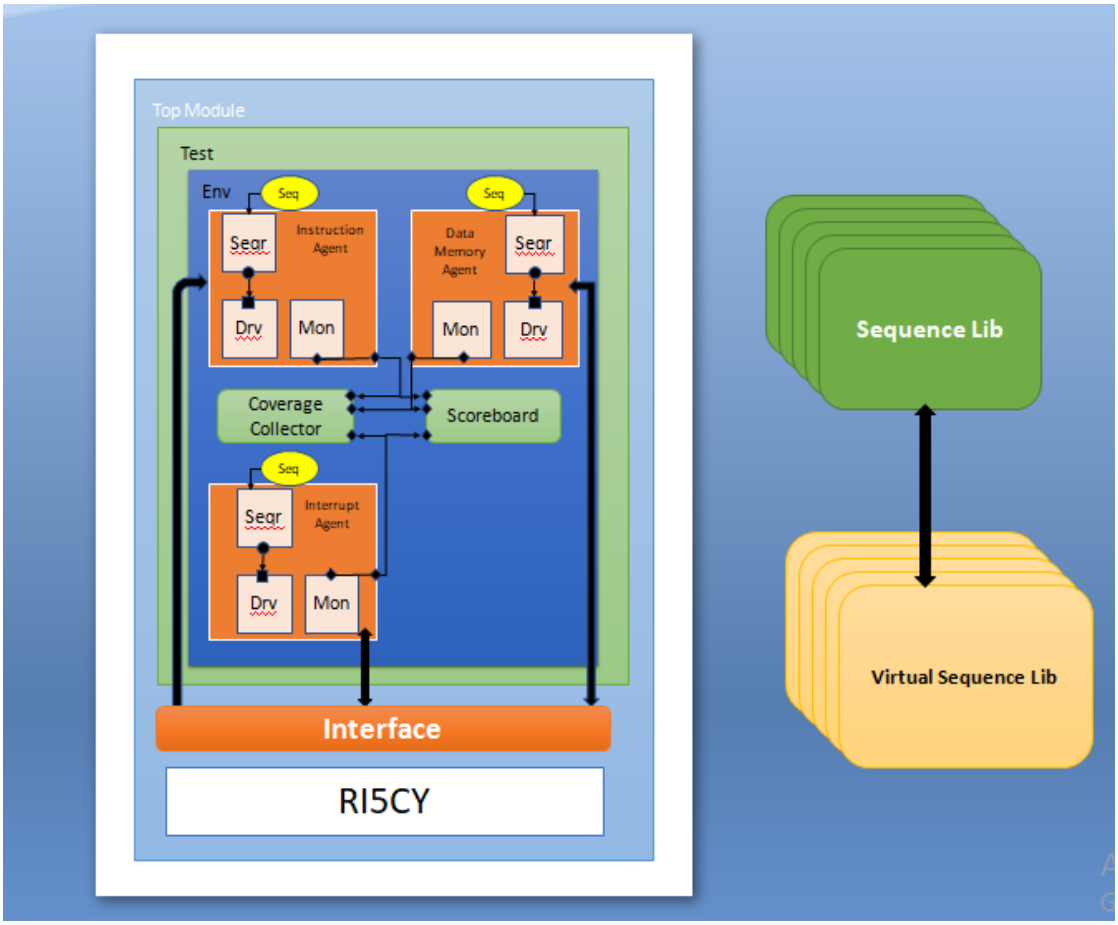


Figure 5.15: UVM Architecture for RI5CY processor .

Chapter 6

Conclusions and Future Works

A configurable SystemVerilog verification environment for 32 bit RISC-V processors is developed in this work, and this verification environment is used for the validation of RISC-V processors with variable instruction set architectures. All verification system components are configured using an configuration object classes in the Test Bench . The verification environment extensively validates the operation of Ibex RISC-V processor with the micro-architecture Model of the RISC-V processor implemented in System verilog, and an intelligent Instruction Generator class designed in System verilog which generates instruction sequences. The verification environment provides a robust control and monitoring environment to validate the end to end operation of the processor and aid in debug in case of failures. The test bench can be configured based on the following processor characteristics: Processor Architecture , instruction word size (32 or 64-bit), number of registers (32 or 64 registers), and type of instructions. The instruction set is classified into three types: Data manipulation, Data transfer, and Flow control instructions. The Interface, Driver, Environment, Test case, and main Test are the major test bench components used to build the test bench

framework. The Interface connects the processors (DUT) with the test bench, the Driver is used to drive the DUT's input signals, the Environment encapsulates the Driver and Test case and is responsible for the flow of operation in the top module. The Test module is the top module that comprises of the clock generator, instances of DUT, interface and the test case. It consists of all the tasks required to perform the processor verification. The research presented in this paper has a lot of scope for future work. Some of the possible ideas that can be developed are presented below:

- **Verification of RISC processors with higher bit sizes e.g., 16-bit, or even 128-bit datapath can be implemented with minimal modifications in the verification system design as bit size is configurable.**
- **Functional Coverage can be done to create the widest possible range of stimuli with all sort of instruction combinations.**
- **USing UVM Register Abstraction Layer that provides a standard base class libraries that enable users to implement the object-oriented model to access the DUT registers and memories .**

Bibliography

[1] UVM–1.2–User–Guide–and–Reference–Manual. http://www.accellera.org/activities/vip, 2015.

[2] A. Traber, "RI5CY core: Datasheet," ETH Zurich and University of Bologna, 2016.

[3] M. Ghoneima, "Reusable processor verification methodology based on UVM."

[4] S. Sutherland and T. Fitzpatrick, "UVM rapid adoption: A practical subset of UVM," 2015.

[5] C. E. Cummings, "OVM/UVM scoreboards-fundamental architectures," 2013.

[6] Valtrix Technologies Pvt. Ltd. (October 2017) RISC-V cpu test plan,revision 1.0. [Online]. Available: http://valtrix.in/announcements/ riscv-test-plan

[7] A. Waterman, Y. Lee, D. A. Patterson and K. Asanovic, "The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA," 13 May 2011. [Online]. Available: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-62.pdf.

[8] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovi, "The risc-v instruction set manual. volume 1: User-level isa, version 2.0," CALIFORNIA UNIV BERKELEY DEPT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCES, Tech. Rep., 2014

[9] RISC-V Foundation, "RISC-V History," RISC-V Foundation, [Online]. Available: https://riscv.org/risc-v-history/. [Accessed 12 Dec. 2019].

[10] A. Waterman, Y. Lee, R. Avizienis, H. Cook, D. Patterson and K. Asanovic, "The RISC-V instruction set," in IEEE Hot Chips 25 Symposium (HCS), Stanford, CA, USA, 2013

[11] SiFive, Inc., "Freedom Studio User Manual," SiFive, Inc., 2 Aug. 2019. [Online]. Available: https://static.dev.sifive.com/dev-tools/FreedomStudio/2019.08/freedom-studio-manual-4.7.2-2019-08-2.pdf. [Accessed 12 Dec. 2019].

Appendix A

Source Code

```
class instr_gen ;

static  bit [31:0] decoded_instr ;

static function bit [31:0] opcode_f ( bit [38:0]
   encoded_instr );
begin
 case (encoded_instr[38:35])
   4'b0000:return r_t(encoded_instr[34:0]);        // R **
   4'b0001:return i_t(encoded_instr[34:0]);        // I **
   4'b0010:return s_t(encoded_instr[34:0]);        // S **
   4'b0011:return b_t(encoded_instr[34:0]);        // b **
   4'b0100:return lu_t(encoded_instr[34:0]);
   4'b0101:return au_t(encoded_instr[34:0]);
```

```
    4'b0110:return j_t(encoded_instr[34:0]);
    4'b0111:return jr_t(encoded_instr[34:0]);
    4'b1000:return il_t(encoded_instr[34:0]);
    default :return r_t(encoded_instr[34:0]);
  endcase
end
endfunction

//-----------------------------------------------------------------------------


static function bit [31:0] r_t (bit [34:0] r_instr);
begin
  case (r_instr [19:16])
    4'b0000:begin decoded_instr [14:12]=3'd0;
    decoded_instr [31:25]=7'd0; end  //add
    4'b0001:begin decoded_instr [14:12]=3'd0;
    decoded_instr [31:25]=7'd32;end  //sub
    4'b0010:begin decoded_instr [14:12]=3'd1;
    decoded_instr [31:25]=7'd0; end  //sll
    4'b0011:begin decoded_instr [14:12]=3'd2;
    decoded_instr [31:25]=7'd0; end  //slt
    4'b0100:begin decoded_instr [14:12]=3'd3;
    decoded_instr [31:25]=7'd0; end  //sltu
    4'b0101:begin decoded_instr [14:12]=3'd4;
```

```
    decoded_instr [31:25]=7'd0; end  //xor
    4'b0110:begin decoded_instr [14:12]=3'd5;
    decoded_instr [31:25]=7'd0; end  //srl
    4'b0111:begin decoded_instr [14:12]=3'd5;
    decoded_instr [31:25]=7'd32;end  //sra
    4'b1000:begin decoded_instr [14:12]=3'd6;
    decoded_instr [31:25]=7'd0; end  //or
    4'b1001:begin decoded_instr [14:12]=3'd7;
    decoded_instr [31:25]=7'd0; end  //and
    default:begin decoded_instr [14:12]=3'd0;
    decoded_instr [31:25]=7'd0; end
  endcase
  decoded_instr [19:15] =r_instr [34:30];
  decoded_instr [24:20] =r_instr [29:25];
  decoded_instr [11:7 ] =r_instr [24:20];
  decoded_instr [6:0  ]=7'b0110011;  //opcode
return decoded_instr ;
end
endfunction

//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------

static function bit [31:0] i_t (bit [34:0] r_instr);
```

```
begin
 case (r_instr [19:16])
   4'b0000:begin decoded_instr [14:12]=3'd0;
   decoded_instr [31:20]=r_instr[15:4]; end          //addi
   4'b0001:begin decoded_instr [14:12]=3'd2;
   decoded_instr [31:20]=r_instr[15:4];end           //slti
   4'b0010:begin decoded_instr [14:12]=3'd3;
   decoded_instr [31:20]=r_instr[15:4]; end          //sltiu
   4'b0011:begin decoded_instr [14:12]=3'd4;
   decoded_instr [31:20]=r_instr[15:4]; end          //xori
   4'b0100:begin decoded_instr [14:12]=3'd6;
   decoded_instr [31:20]=r_instr[15:4]; end          //ori
   4'b0101:begin decoded_instr [14:12]=3'd7;
   decoded_instr [31:20]=r_instr[15:4]; end          //andi
   4'b0110:begin decoded_instr [14:12]=3'd1;
   decoded_instr [31:20]={7'd0,r_instr[8:4]}; end    //slli
   4'b0111:begin decoded_instr [14:12]=3'd5;
   decoded_instr [31:20]={7'd0,r_instr[8:4]}; end    //srli
   4'b1000:begin decoded_instr [14:12]=3'd5;
   decoded_instr [31:20]={7'd32,r_instr[8:4]};end    //srai

   //4'b1001:begin decoded_instr [14:12]=3'd7;
   decoded_instr [31:25]=7'd0; end   //and
   default:begin decoded_instr [14:12]=3'd0;
   decoded_instr [31:20]=r_instr[15:4]; end   //addi
```

```
 endcase
 decoded_instr [19:15] =r_instr [34:30];    //rs1
 decoded_instr [11:7 ] =r_instr [24:20];    //rd
 decoded_instr [6:0  ]=7'b0010011;  //opcode
return decoded_instr ;
end
endfunction

//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------

static function bit [31:0] s_t (bit [34:0] r_instr);
begin
 case (r_instr [19:16])

   4'b0000:begin decoded_instr [14:12]=3'd0;
   decoded_instr [31:25]=r_instr [15:9];decoded_instr
      [11:5]=r_instr [8:4]; end  //sb
   4'b0001:begin decoded_instr [14:12]=3'd1;
   decoded_instr [31:25]=r_instr [15:9];decoded_instr
      [11:5]=r_instr [8:4]; end  //sh
   4'b0010:begin decoded_instr [14:12]=3'd2;
   decoded_instr [31:25]=r_instr [15:9];decoded_instr
      [11:5]=r_instr [8:4]; end  //sw
```

```
    default:begin decoded_instr [14:12]=3'd0;
    decoded_instr [31:25]=r_instr [15:9];decoded_instr
       [11:5]=r_instr [8:4]; end  //sb
 endcase
 decoded_instr [19:15] =r_instr [34:30];    //rs1
 decoded_instr [24:20] =r_instr [29:25];    //rs2
 decoded_instr [6:0   ]=7'b0100011;  //opcode
return decoded_instr ;
end
endfunction

//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------

static function bit [31:0] b_t (bit [34:0] r_instr);
begin
 case (r_instr [19:16])
   4'b0000:begin decoded_instr [14:12]=3'd0;
   decoded_instr [31]=r_instr [11];
                 decoded_instr [7]=r_instr [10];
                 decoded_instr [30:25]=r_instr [9:4];
                 decoded_instr [11:8]=r_instr [3:0]; end
                                                  //beq
```

```
4'b0001:begin decoded_instr [14:12]=3'd1;
decoded_instr [31]=r_instr [11];
              decoded_instr [7]=r_instr [10];
              decoded_instr [30:25]=r_instr [9:4];
              decoded_instr [11:8]=r_instr [3:0]; end
                                                  //bnq


4'b0010:begin decoded_instr [14:12]=3'd4;
decoded_instr [31]=r_instr [11];
              decoded_instr [7]=r_instr [10];
              decoded_instr [30:25]=r_instr [9:4];
              decoded_instr [11:8]=r_instr [3:0]; end
                                                  //blt


4'b0011:begin decoded_instr [14:12]=3'd5;
decoded_instr [31]=r_instr [11];
              decoded_instr [7]=r_instr [10];
              decoded_instr [30:25]=r_instr [9:4];
              decoded_instr [11:8]=r_instr [3:0]; end
                                                  //bge
```

```
    4'b0100:begin decoded_instr [14:12]=3'd6;
    decoded_instr [31]=r_instr [11];
                  decoded_instr [7]=r_instr [10];
                  decoded_instr [30:25]=r_instr [9:4];
                  decoded_instr [11:8]=r_instr [3:0]; end
                                                  //bltu


    4'b0101:begin decoded_instr [14:12]=3'd7;
    decoded_instr [31]=r_instr [11];
                  decoded_instr [7]=r_instr [10];
                  decoded_instr [30:25]=r_instr [9:4];
                  decoded_instr [11:8]=r_instr [3:0]; end
                                                  //bgeu



 endcase
 decoded_instr [19:15] =r_instr [34:30];  //rs1
 decoded_instr [24:20] =r_instr [29:25];  //rs2
 decoded_instr [6:0   ]=7'b1100011;       //opcode
return decoded_instr ;
end
endfunction
```

```
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------

static function bit [31:0] lu_t (bit [34:0] r_instr);
begin

 decoded_instr [31:12] =r_instr [19: 0];    //imm
 decoded_instr [11:7 ] =r_instr [24:20];    //rd
 decoded_instr [6:0  ]=7'b0110111;          //opcode

return decoded_instr ;
end
endfunction
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------

static function bit [31:0] au_t (bit [34:0] r_instr);
begin

 decoded_instr [31:12] =r_instr [19: 0];    //imm
 decoded_instr [11:7 ] =r_instr [24:20];    //rd
 decoded_instr [6:0  ]=7'b0010111; // opcode
```

```
return decoded_instr ;
end
endfunction
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------

static function bit [31:0] j_t (bit [34:0] r_instr);
begin

 decoded_instr [31:12] =r_instr [19: 0];    //imm
 decoded_instr [11:7 ] =r_instr [24:20];    //rd
 decoded_instr [6:0  ]=7'b1101111;          //opcode

return decoded_instr ;
end
endfunction
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------

static function bit [31:0] jr_t (bit [34:0] r_instr);
begin
```

```
 decoded_instr [19:15] =r_instr [34:30];    //rs1
 decoded_instr [31:20] =r_instr [15:4];     //imm 12 bit
 decoded_instr [14:12] =3'b0;               //funct3
 decoded_instr [11:7 ] =r_instr [24:20];    //rd
 decoded_instr [6:0  ]=7'b1100111;          // opcode

return decoded_instr ;
end
endfunction
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

static function bit [31:0] il_t (bit [34:0] r_instr);
begin
 case (r_instr [19:16])
   4'b0000:begin decoded_instr [14:12]=3'd0;
   decoded_instr [31:20]=r_instr[15:4]; end        //lb
   4'b0001:begin decoded_instr [14:12]=3'd1;
   decoded_instr [31:20]=r_instr[15:4]; end        //lh
   4'b0010:begin decoded_instr [14:12]=3'd2;
   decoded_instr [31:20]=r_instr[15:4]; end        //lw
   4'b0011:begin decoded_instr [14:12]=3'd4;
   decoded_instr [31:20]=r_instr[15:4]; end        //lbu
   4'b0100:begin decoded_instr [14:12]=3'd5;
```

```
    decoded_instr [31:20]=r_instr[15:4]; end          //lhu

    default:begin decoded_instr [14:12]=3'd0;
    decoded_instr [31:20]=r_instr[15:4]; end          //lb
 endcase
 decoded_instr [19:15] =r_instr [34:30];    //rs1
 decoded_instr [11:7 ] =r_instr [24:20];    //rd
 decoded_instr [6:0   ]=7'b0000011;         //opcode
return decoded_instr ;
end
endfunction

//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------









/*initial
begin
  opcode_f (38'b000_00001_00010_00011_0011_0000000000000000
     );
```

```
  $display ("%b",decoded_instr);

end*/

endclass

class w_2_c ;

static bit [38:0] instr_2gen;

//-------------------------------------------------------------------

//-------------------------------------------------------------------

 static function bit [38:0] transformer (string
    encoded_instr );
begin
 case (encoded_instr.substr(0,0))
   "r":return r_de(encoded_instr);          // R **
   "i":return i_de(encoded_instr);          // I **
   "s":return s_de(encoded_instr);          // S **
   "b":return b_de(encoded_instr);          // b **
   "u":return u_de(encoded_instr);
   "w":return w_de(encoded_instr);
   "j":return j_de(encoded_instr);
   "a":return a_de(encoded_instr);
```

```
    "l":return l_de(encoded_instr);
    default :return r_de(encoded_instr);
  endcase
end
endfunction
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------


static function bit [38:0] r_de (string r_instr);
begin
  string sub_str ;
  bit [31:0] temp ;
  instr_2gen [38:35] =4'b0000;
  sub_str=r_instr.substr(11,12);  //take rs1 no inside
     string
  $display("%s",sub_str);
  temp=sub_str.atoi();
  //transform string to integer assign to bin
  instr_2gen [34:30] =temp[4:0];  //rs1 //assign the segment
      that include the value needed
  $display("%b",temp);
  sub_str=r_instr.substr(15,16);
  $display("%s",sub_str);
```

```
temp=sub_str.atoi();
instr_2gen [29:25] =temp[4:0];;   //rs2
$display("%b",temp);
sub_str=r_instr.substr(7,8);
$display("%s",sub_str);
temp=sub_str.atoi();
instr_2gen [24:20] =temp[4:0];    //rd
$display("%b",temp);



case (r_instr.substr(1,4))
  "add ":instr_2gen [19:16]=4'd0;                 // R **
  "sub ":instr_2gen [19:16]=4'd1;          // I **
  "sll ":instr_2gen [19:16]=4'd2;          // S **
  "slt ":instr_2gen [19:16]=4'd3;          // b **
  "sltu":instr_2gen [19:16]=4'd4;
  "xor ":instr_2gen [19:16]=4'd5;
  "srl ":instr_2gen [19:16]=4'd6;
  "sra ":instr_2gen [19:16]=4'd7;
  "or  ":instr_2gen [19:16]=4'd8;
  "and ":instr_2gen [19:16]=4'd9;
  default :instr_2gen [19:16]=4'd0;
endcase
```

```
return instr_2gen ;
end
endfunction
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

static function bit [38:0] i_de (string r_instr);
begin
 string sub_str ;
 bit [31:0] temp ;
 instr_2gen [38:35] =4'b0001;
 sub_str=r_instr.substr(11,12);  //take rs1 no inside
    string
 $display("%s",sub_str);
 temp=sub_str.atoi();
 //transform string to integer assign to bin
 instr_2gen [34:30] =temp[4:0];  //rs1 //assign the segment
     that include the value needed
 $display("%b",temp);
 sub_str=r_instr.substr(7,8);
```

```
$display("%s",sub_str);
temp=sub_str.atoi();
instr_2gen [24:20] =temp[4:0];     //rd
$display("%b",temp);
sub_str=r_instr.substr(14,r_instr.len()-1);
$display("%s",sub_str);
temp=sub_str.atoi();
instr_2gen [15:4] =temp[11:0];     //imm
$display("%b",temp);



case (r_instr.substr(1,5))
  "addi ":instr_2gen [19:16]=4'd0;
  "slti ":instr_2gen [19:16]=4'd1;
  "sltiu":instr_2gen [19:16]=4'd2;
  "xori ":instr_2gen [19:16]=4'd3;
  "ori  ":instr_2gen [19:16]=4'd4;
  "andi ":instr_2gen [19:16]=4'd5;
  "slli ":instr_2gen [19:16]=4'd6;
  "srli ":instr_2gen [19:16]=4'd7;
  "srai ":instr_2gen [19:16]=4'd8;

  default :instr_2gen [19:16]=4'd0;
endcase
```

```
return instr_2gen ;
end
endfunction
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

static function bit [38:0] s_de (string r_instr);
begin
 string sub_str ;
 bit [31:0] temp ;
 instr_2gen [38:35] =4'b0010;
 sub_str=r_instr.substr(5,6);  //take rs1 no inside string
 $display("%s",sub_str);
 temp=sub_str.atoi();
 //transform string to integer assign to bin
 instr_2gen [34:30] =temp[4:0];  //rs1 //assign the segment
     that include the value needed
 $display("%b",temp);
```

```
sub_str=r_instr.substr(9,10);  //take rs1 no inside string
$display("%s",sub_str);
temp=sub_str.atoi();
//transform string to integer assign to bin
instr_2gen [29:25] =temp[4:0];  //rs2 //assign the segment
    that include the value needed
$display("%b",temp);

sub_str=r_instr.substr(12,r_instr.len()-1);
$display("%s",sub_str);
temp=sub_str.atoi();
instr_2gen [15:4] =temp[11:0];    //imm
$display("%b",temp);



case (r_instr.substr(1,2))
  "sb":instr_2gen [19:16]=4'd0;
  "sh":instr_2gen [19:16]=4'd1;
  "sw":instr_2gen [19:16]=4'd2;

  default :instr_2gen [19:16]=4'd0;
endcase


return instr_2gen ;
```

```
end
endfunction
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

static function bit [38:0] b_de (string r_instr);
begin
 string sub_str ;
 bit [31:0] temp ;
 instr_2gen [38:35] =4'b0011;
 sub_str=r_instr.substr(7,8);  //take rs1 no inside string
 $display("%s",sub_str);
 temp=sub_str.atoi();
 //transform string to integer assign to bin
 instr_2gen [34:30] =temp[4:0];  //rs1 //assign the segment
     that include the value needed
 $display("%b",temp);

 sub_str=r_instr.substr(11,12);  //take rs1 no inside
    string
```

```
$display("%s",sub_str);
temp=sub_str.atoi();
//transform string to integer assign to bin
instr_2gen [29:25] =temp[4:0];  //rs2 //assign the segment
    that include the value needed
$display("%b",temp);

sub_str=r_instr.substr(14,r_instr.len()-1);
$display("%s",sub_str);
temp=sub_str.atoi();
instr_2gen [15:4] =temp[11:0];    //imm
$display("%b",temp);



case (r_instr.substr(1,4))
  "beq ":instr_2gen [19:16]=4'd0;
  "bnq ":instr_2gen [19:16]=4'd1;
  "blt ":instr_2gen [19:16]=4'd2;
  "bge ":instr_2gen [19:16]=4'd3;
  "bltu":instr_2gen [19:16]=4'd4;
  "bgeu":instr_2gen [19:16]=4'd5;

  default :instr_2gen [19:16]=4'd3;
endcase
```

```
return instr_2gen ;
end
endfunction
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

//-------------------------------------------------------------------------


static function bit [38:0] u_de (string r_instr);
begin
 string sub_str ;
 bit [31:0] temp ;
 instr_2gen [38:35] =4'b0100;

 sub_str=r_instr.substr(6,7);
 $display("%s",sub_str);
 temp=sub_str.atoi();
 instr_2gen [24:20] =temp[4:0];    //rd
 $display("%b",temp);
```

```
 sub_str=r_instr.substr(9,r_instr.len()-1);
 $display("%s",sub_str);
 temp=sub_str.atoi();
 instr_2gen [19:0] =temp[19:0];     //imm
 $display("%b",temp);



return instr_2gen ;
end
endfunction
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

static function bit [38:0] w_de (string r_instr);
begin
 string sub_str ;
 bit [31:0] temp ;
 instr_2gen [38:35] =4'b0101;
```

```
 sub_str=r_instr.substr(8,9);
 $display("%s",sub_str);
 temp=sub_str.atoi();
 instr_2gen [24:20] =temp[4:0];     //rd
 $display("%b",temp);

 sub_str=r_instr.substr(11,r_instr.len()-1);
 $display("%s",sub_str);
 temp=sub_str.atoi();
 instr_2gen [19:0] =temp[19:0];     //imm
 $display("%b",temp);



return instr_2gen ;
end
endfunction
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

//-------------------------------------------------------------------------
```

```
static function bit [38:0] j_de (string r_instr);
begin
 string sub_str ;
 bit [31:0] temp ;
 instr_2gen [38:35] =4'b0110;

 sub_str=r_instr.substr(6,7);
 $display("%s",sub_str);
 temp=sub_str.atoi();
 instr_2gen [24:20] =temp[4:0];    //rd
 $display("%b",temp);

 sub_str=r_instr.substr(9,r_instr.len()-1);
 $display("%s",sub_str);
 temp=sub_str.atoi();
 instr_2gen [19:0] =temp[19:0];    //imm 20 bit
 $display("%b",temp);



return instr_2gen ;
end
endfunction
//-------------------------------------------------------------------------
```

```
//------------------------------------------------------------------------

//------------------------------------------------------------------------

//------------------------------------------------------------------------

static function bit [38:0] a_de (string r_instr);
begin
 string sub_str ;
 bit [31:0] temp ;
 instr_2gen [38:35] =4'b0111;

 sub_str=r_instr.substr(11,12);  //take rs1 no inside
    string
 $display("%s",sub_str);
 temp=sub_str.atoi();
 //transform string to integer assign to bin
 instr_2gen [34:30] =temp[4:0];  //rs1 //assign the segment
     that include the value needed
 $display("%b",temp);


 sub_str=r_instr.substr(7,8);
 $display("%s",sub_str);
 temp=sub_str.atoi();
```

```
 instr_2gen [24:20] =temp[4:0];     //rd
 $display("%b",temp);

 sub_str=r_instr.substr(14,r_instr.len()-1);
 $display("%s",sub_str);
 temp=sub_str.atoi();
 instr_2gen [15:4] =temp[11:0];     //imm 12 bit
 $display("%b",temp);



return instr_2gen ;
end
endfunction
//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

//-------------------------------------------------------------------------

//-------------------------------------------------------------------------


static function bit [38:0] l_de (string r_instr);
begin
```

```
string sub_str ;
bit [31:0] temp ;
instr_2gen [38:35] =4'b1000;

sub_str=r_instr.substr(10,11);  //take rs1 no inside
   string
$display("%s",sub_str);
temp=sub_str.atoi();
//transform string to integer assign to bin
instr_2gen [34:30] =temp[4:0];  //rs1 //assign the segment
    that include the value needed
$display("%b",temp);


sub_str=r_instr.substr(6,7);
$display("%s",sub_str);
temp=sub_str.atoi();
instr_2gen [24:20] =temp[4:0];    //rd
$display("%b",temp);

sub_str=r_instr.substr(13,r_instr.len()-1);
$display("%s",sub_str);
temp=sub_str.atoi();
instr_2gen [15:4] =temp[11:0];    //imm 12 bit
$display("%b",temp);
```

```
  case (r_instr.substr(1,3))
   "lb ":instr_2gen [19:16]=4'd0;
   "lh ":instr_2gen [19:16]=4'd1;
   "lw ":instr_2gen [19:16]=4'd2;
   "lbu":instr_2gen [19:16]=4'd3;
   "lhu":instr_2gen [19:16]=4'd4;

    default :instr_2gen [19:16]=4'd0;
 endcase



return instr_2gen ;
end
endfunction
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
```

```
//------------------------------------------------------------------------------

/*initial
 begin
    transformer("llw   $19 $22 99");
    $display ("%b",instr_2gen);*/



 //end









endclass

\lstset{%
   breaklines=true
}
`ifndef instr_agent_config
```

```
`define instr_agent_config

//------------------------------------------------------------------------------

// instr_agent_config
//------------------------------------------------------------------------------

class instr_agent_config   extends uvm_object;

   localparam string s_my_config_id           = "m_config
      ";
   localparam string s_no_config_id           = "no config
      ";

   // factory registration macro

  `uvm_object_utils(instr_agent_config)

   // agent configuration
   uvm_active_passive_enum is_active = UVM_ACTIVE;

   // virtual interface handle:
   virtual           ibex_if    m_vif;

   //variables
```

```
int                 unsigned symbol_count;

//------------------------------------------------------------------------------

// new
//------------------------------------------------------------------------------

function new(string name = "instr_agent_config");
   super.new(name);
endfunction: new

//------------------------------------------------------------------------------

// Get configuration
//------------------------------------------------------------------------------

static function instr_agent_config  get_config(
   uvm_component c);
   instr_agent_config  t;
   if(!uvm_config_db #(instr_agent_config )::get(c, "" ,
       s_my_config_id , t)) begin
      `uvm_error(s_no_config_id ,$sformatf("no config
         associated with %s",s_my_config_id))
      return null;
   end
```

```
      return t;
   endfunction // get_config

endclass: instr_agent_config
`endif

\lstset{%
   breaklines=true
}
`ifndef instr_agent
   `define instr_agent



//------------------------------------------------------------------------------

// instr_agent
//------------------------------------------------------------------------------


class instr_agent  extends uvm_agent;

    // factory registration macro
  `uvm_component_utils(instr_agent)
   // configuration object
   instr_agent_config  m_config;
```

```
// external interfaces
uvm_analysis_port #(instr_seq_item ) ap;

// internal components
instr_monitor      m_instr_monitor;
instr_driver       m_instr_driver;
instr_sequencer    m_instr_sequencer;

//------------------------------------------------------------------------

// new
//------------------------------------------------------------------------

function new(string name = "instr_agent",uvm_component
   parent = null);
   super.new(name, parent);
endfunction: new

//------------------------------------------------------------------------

// build
//------------------------------------------------------------------------
```

```
virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    m_config = instr_agent_config ::get_config(this);
    ap = new("ap", this);
    m_instr_monitor = instr_monitor ::type_id::create("
        m_instr_monitor",this);
    if (m_config.is_active == UVM_ACTIVE) begin
        m_instr_driver = instr_driver ::type_id::create("
            m_instr_driver",this);
        m_instr_sequencer = instr_sequencer ::type_id::
            create("m_instr_sequencer",this);
    end
endfunction: build_phase

//------------------------------------------------------------------------------

// connect
//------------------------------------------------------------------------------


virtual function void connect_phase(uvm_phase phase);
    m_instr_monitor.ap.connect(ap);
    m_instr_monitor.m_vif = m_config.m_vif;
    if (m_config.is_active == UVM_ACTIVE) begin
        m_instr_driver.m_vif = m_config.m_vif;
```

```
            m_instr_driver.seq_item_port.connect(
                m_instr_sequencer.seq_item_export);
        end
    endfunction: connect_phase
endclass: instr_agent
`endif

\lstset{%
    breaklines=true
}
`ifndef instr_driver
 `define instr_driver

`include "uvm_setup/env/instr_gen.sv"

 import uvm_pkg::*;
//------------------------------------------------------------------------------

// instr_driver
//------------------------------------------------------------------------------


class instr_driver  extends uvm_driver #(instr_seq_item );

    // factory registration macro
```

```
`uvm_component_utils(instr_driver)


// internal components
instr_seq_item   m_instr_seq_item;

// interface
virtual ibex_if   m_vif;

//---------------------------------------------------------------------------

// new
//---------------------------------------------------------------------------



function new (string name = "instr_driver",
    uvm_component parent = null);
    super.new(name,parent);
endfunction: new


//---------------------------------------------------------------------------

// run
```

```
//————————————————————————————————————————————————————————

virtual task run_phase(uvm_phase phase);

     // Reset or initialize the DUT
   forever begin

       seq_item_port.get_next_item(m_instr_seq_item);
   /////////////////////
      fork
       drive_instr();
       time_out ();
       join_any
       disable fork ;
       seq_item_port.item_done();
   end // forever begin
endtask // run_phase


task drive_instr();

   @(posedge m_vif.clk_i );
         begin
```

```
m_vif.instr_rdata_i <= instr_gen::opcode_f
    ({m_instr_seq_item.type_sel,
      m_instr_seq_item.rs1,
      m_instr_seq_item.rs2,
      m_instr_seq_item.rd,
      m_instr_seq_item.funct,
      m_instr_seq_item.imm,
      m_instr_seq_item.left}       );

m_vif.instr_gnt_i     <= m_instr_seq_item.
    instr_gnt_i;
            // m_vif.instr_gnt_i     <= 1;
  m_vif.instr_rvalid_i <= m_instr_seq_item.
      instr_rvalid_i;
  //m_vif.instr_rvalid_i <=1;
  //m_vif.instr_gnt_i     <= m_instr_seq_item.
      instr_gnt_i;
  //m_vif.instr_rdata_i  <= m_instr_seq_item.
      instr_rdata_i;
  m_vif.instr_err_i     <= m_instr_seq_item.
      instr_err_i;
  end


/*  $display ("m_vif.instr_rdata_i  %d",m_vif.
```

```
        instr_rdata_i);
      $display ("m_vif.instr_gnt_i %d",m_vif.instr_gnt_i)
         ;
      $display ("m_vif.instr_rvalid_i %d",m_vif.
         instr_rvalid_i);
      $display ("m_vif.instr_err_i %d",m_vif.instr_err_i)
         ;*/
     @(posedge m_vif.clk_i);
    // @(posedge m_vif.clk_i);

endtask



 task time_out () ;

      #1000000

      $display ("OOOH : TIME OUT IN THE INSTR_DRIVER");
      $display ("m_vif.instr_req_o %b",m_vif.instr_req_o)
         ;
      endtask
```

```
endclass // instr_driver
`endif

\lstset{%
    breaklines=true
}
`ifndef instr_monitor
 `define instr_monitor



//------------------------------------------------------------------------------

// instr_monitor
//------------------------------------------------------------------------------


class instr_monitor  extends uvm_monitor;

    // factory registration macro

    `uvm_component_utils(instr_monitor)


    // external interfaces
    uvm_analysis_port #(instr_seq_item ) ap;
```

```
// interface
virtual ibex_if  m_vif;

//------------------------------------------------------------------------------

// new
//------------------------------------------------------------------------------

function new (string name = "instr_monitor",
    uvm_component parent = null);
    super.new(name,parent);
endfunction: new



//------------------------------------------------------------------------------

// build
//------------------------------------------------------------------------------

virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
  ap = new("ap",this);
endfunction: build_phase
```

```
//-------------------------------------------------------------------

// run
//-------------------------------------------------------------------

virtual task run_phase(uvm_phase phase);

    instr_seq_item    mon_instr_seq_item;
    mon_instr_seq_item= instr_seq_item ::type_id::create
       ("mon_instr_seq_item");

    forever begin

            @(posedge m_vif.clk_i iff m_vif.instr_req_o
               ==1);
                  mon_instr_seq_item.instr_rdata_i  = m_vif
                     .instr_rdata_i;
                  mon_instr_seq_item.pc  = m_vif.pc_if_o;


                  mon_instr_seq_item.mem_data_in =m_vif.
                     data_com;
                 mon_instr_seq_item.mem_addr_in =m_vif.
                    addr_com ;
```

```
            mon_instr_seq_item.instr_gnt_i     <=
              m_vif.instr_gnt_i;
            mon_instr_seq_item.instr_rvalid_i <=
              m_vif.instr_rvalid_i;
            mon_instr_seq_item.instr_gnt_i     <=
              m_vif.instr_gnt_i;
            mon_instr_seq_item.instr_rdata_i  <=
              m_vif.instr_rdata_i;
            mon_instr_seq_item.instr_err_i     <=
              m_vif.instr_err_i;
            mon_instr_seq_item.instr_addr_o    <=
              m_vif.instr_addr_o;
               mon_instr_seq_item.fetch_enable_i
                 <= m_vif.fetch_enable_i;
               mon_instr_seq_item.mem_seq
                 <= m_vif.mem_if;


    /// data sampling
    // mon_txn.input_a = m_vif.input_a;
ap.write(mon_instr_seq_item);
    mon_instr_seq_item= instr_seq_item ::type_id
      ::create("mon_instr_seq_item");
     @(posedge m_vif.clk_i);
```

```
      end // forever

   endtask // run_phase



endclass: instr_monitor
`endif

\lstset{%
   breaklines=true
}
`ifndef instr_sequencer
 `define instr_sequencer


//------------------------------------------------------------------------------

// instr_sequencer
//------------------------------------------------------------------------------



class instr_sequencer  extends uvm_sequencer #(
```

```
    instr_seq_item );

    // declaration macros

    `uvm_component_utils(instr_sequencer)

    //-------------------------------------------------------------------

    // new
    //-------------------------------------------------------------------

    function new(string name,uvm_component parent);
       super.new(name, parent);
    endfunction: new

endclass: instr_sequencer
`endif
```