RISC-V®

COMMUNITY MEMBER

CV32E40P UVM Verification

An open-source 32-bit RISC-V processor core designed for embedded systems, featuring a 5-stage pipeline and integrated memory interfaces for efficient instruction fetch and data access.

Prepared by **Team : 3**

SI-VISION ACADEMY

Topics Covered

CV32E40P Core Overview

UVM ENV Architecture.

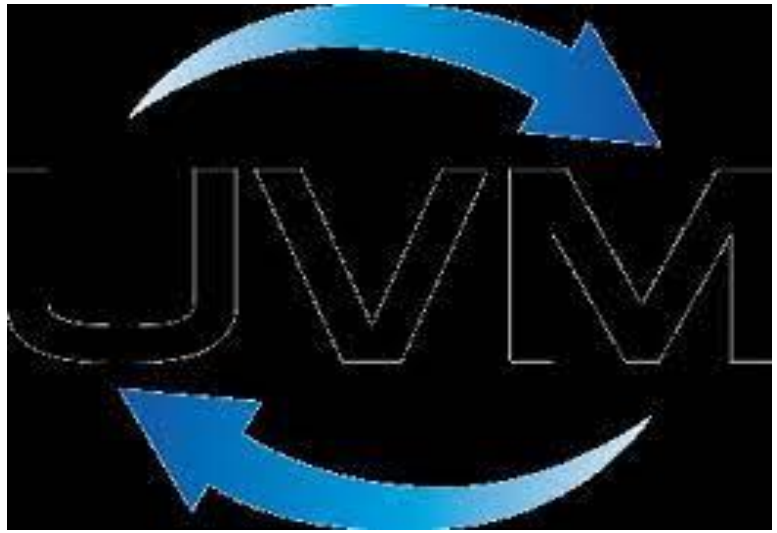
Test Scenatios

Results and Analysis

Simulation results.

Coverage reports.

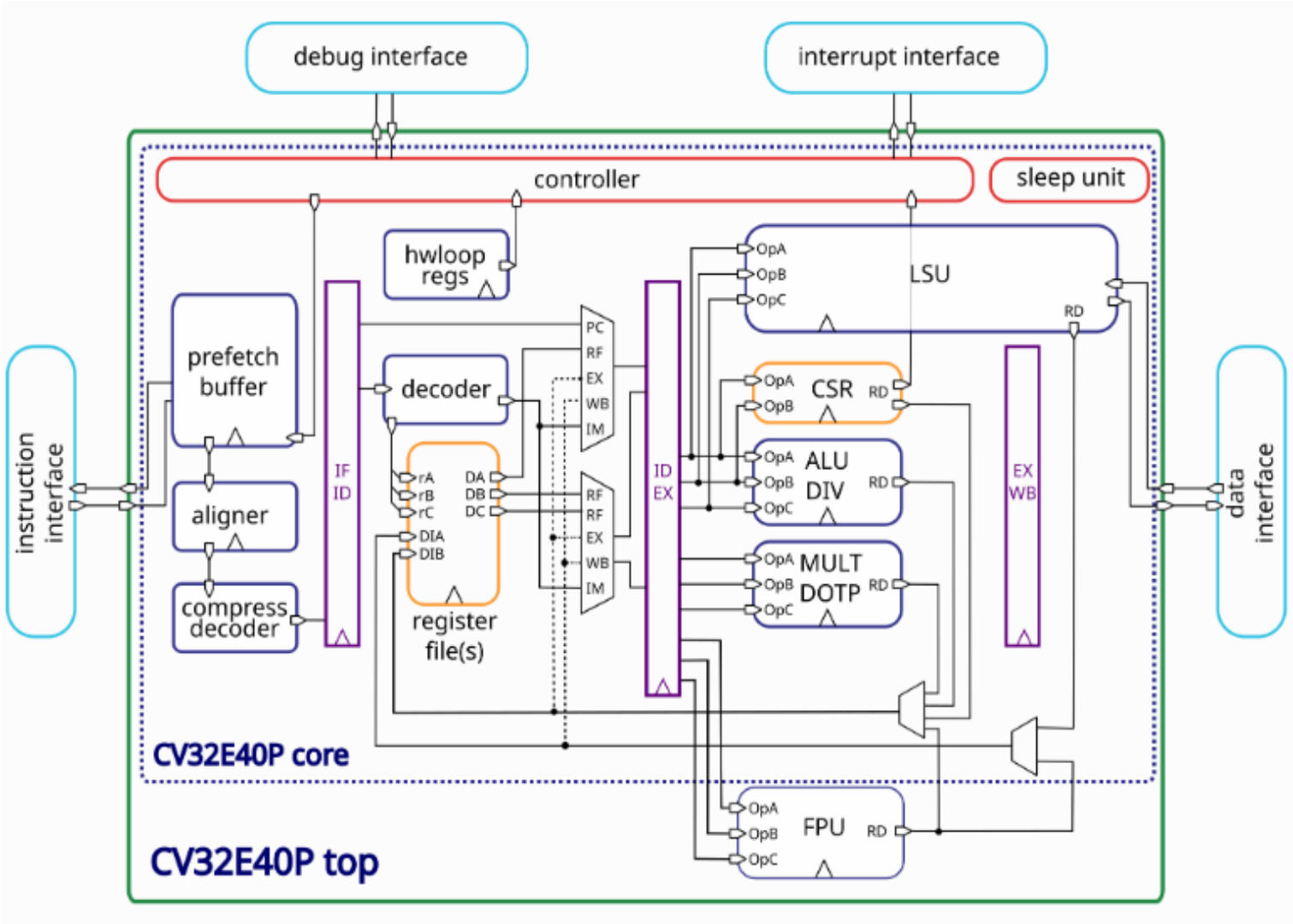
Debugging and error handling.



Core CV32E40P

CV32E40P Core Overview

- The CV32E40P is a 32-bit RISC-V processor core with a 4-stage in-order pipeline.
- It supports the RV32IM instruction set architecture.
- Includes several custom extensions, such as hardware loops, post-increment load/store instructions, enhanced ALU operations, and SIMD instructions.



Verification Goals

Si-VISION
INNOVATIVE IC SOLUTIONS

Instruction Types

- Verify **correct decoding** of all supported instruction types: R, I, S, B, U, J, and M_Extention.
- Validate operand **fetch**, **immediate extraction**, and **sign-extension** logic.
- Ensure correct ALU operation and result forwarding for:
  - Arithmetic (ADD, SUB)
  - Logic (AND, OR, XOR)
  - Shifts (SLL, SRL, SRA)
- Test **immediate** instructions for proper sign extension and functional correctness.
- Validate **branch** resolution and **PC update** for conditional branches (BEQ, BNE, etc.).
- Check **jump and link** operations (JAL, JALR) for correct PC calculation and register update.

Verification Goals

Si-VISION
INNOVATIVE IC SOLUTIONS

Load/Store Unit (LSU)

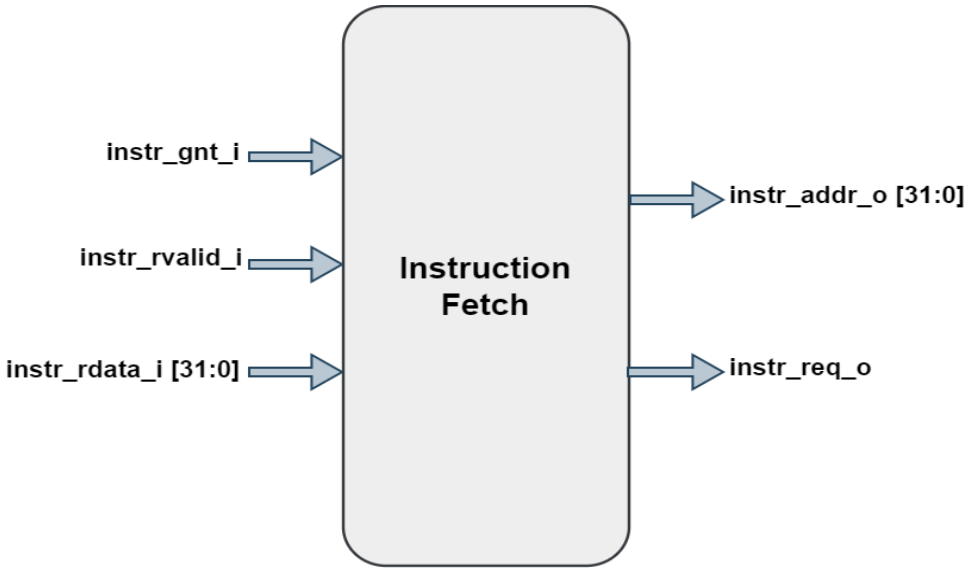
- Confirm **LSU** issues correct memory request signals **(data_req_o, data_we_o, etc.).**
- Verify address generation and alignment checking for load/store operations.
- Ensure proper functionality of load instructions:
  - **Byte, half-word, word widths**.
  - **Sign-extension and zero-extension behavior.**
- Validate correct data writing behavior for store instructions **(SB, SH, SW).**
- Check handling of **misaligned** memory accesses (trap generation).
- Test correct handling of **load-use hazards** (pipeline stall for load followed by dependent instruction).
- Confirm LSU does not issue memory access when **stalled** or **flushed**.

Block Diagrams

Block Diagrams

Instruction Fetch

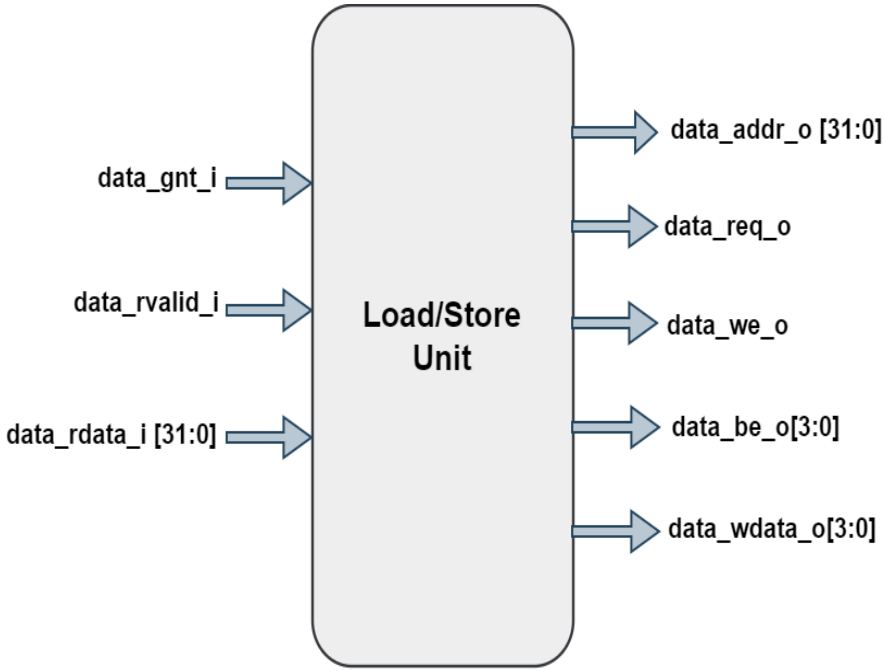
Signal	Direction	Description
instr_addr_o[31:0]	output	Address, word aligned
instr_req_o	output	Request valid, will stay high until instr_gnt_i is high for one cycle
instr_gnt_i	input	The other side accepted the request. instr_addr_o may change in the next cycle.
instr_rvalid_i	input	instr_rdata_i holds valid data when instr_rvalid_i is high. This signal will be high for exactly one cycle per request.
instr_rdata_i[31:0]	input	Data read from memory



Block Diagrams

Load/Store Unit

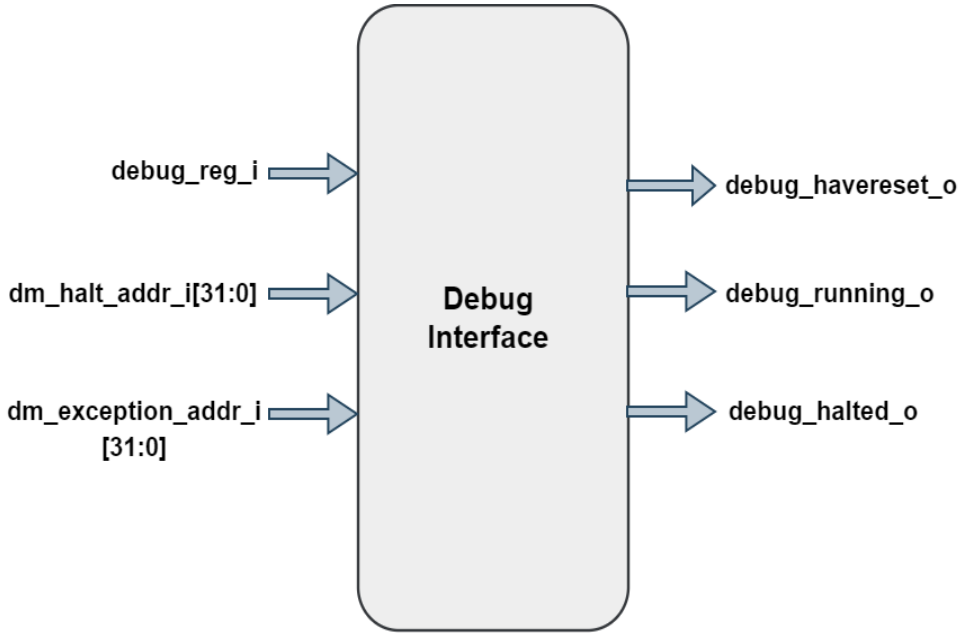
Signal	Direction	Description
data_addr_o[31:0]	output	Address
data_req_o	output	Request valid, will stay high until data_gnt_i is high for one cycle
data_gnt_i	input	The other side accepted the request. data_addr_o may change in the next cycle.
data_we_o	output	Write Enable, high for writes, low for reads. Sent together with data_req_o
data_be_o[3:0]	output	Byte Enable. Is set for the bytes to write/read, sent together with data_req_o
data_wdata_o[31:0]	output	Data to be written to memory, sent together with data_req_o
data_rvalid_i	input	data_rvalid_i will be high for exactly one cycle to signal the end of the response phase of for both read and write transactions. For a read transaction data_rdata_i holds valid data when data_rvalid_i is high.
data_rdata_i[31:0]	input	Data read from memory



Block Diagrams

Debug Interface

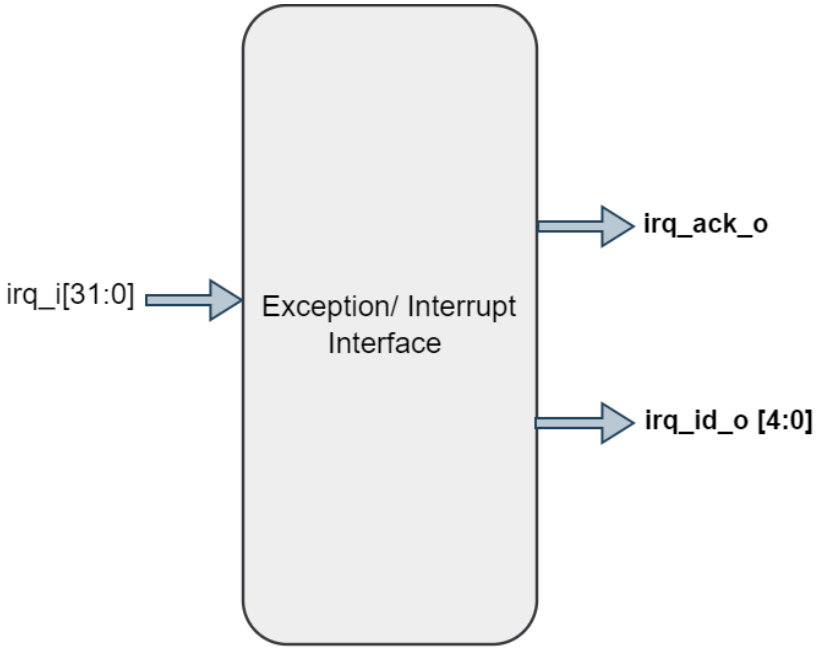
Signal	Direction	Description
debug_req_i	input	Request to enter Debug Mode
debug_havereset_o	output	Debug status: Core has been reset
debug_running_o	output	Debug status: Core is running
debug_halted_o	output	Debug status: Core is halted
dm_halt_addr_i[31:0]	input	Address for debugger entry
dm_exception_addr_i [31:0]	input	Address for debugger exception entry



Block Diagrams

Exception/Interrupt Interface

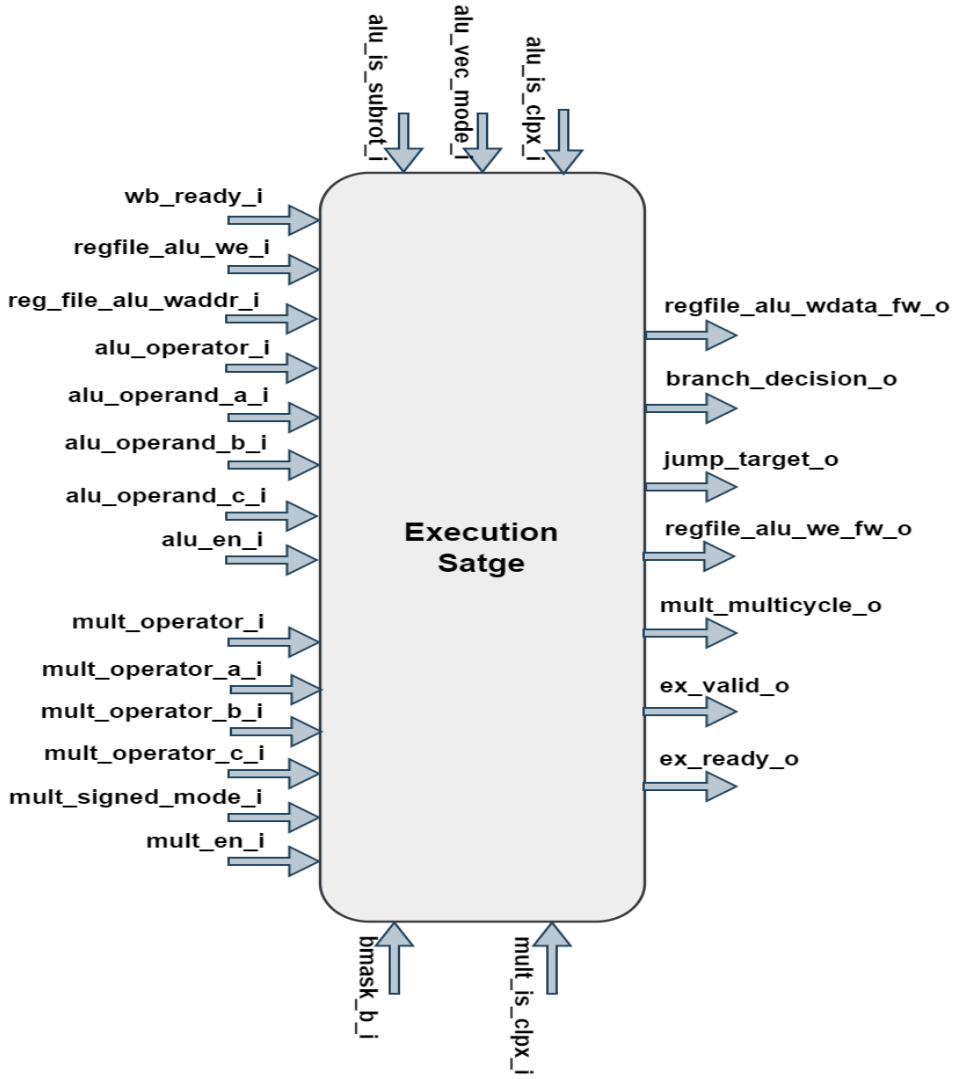
Signal	Direction	Description
irq_i[31:0]	input	Level sensistive active high interrupt inputs. Not all interrupt inputs can be used on CV32E40P. Specifically irq_i[15:12], irq_i[10:8], irq_i[6:4] and irq_i[2:0] shall be tied to 0 externally as they are reserved for future standard use (or for cores which are not Machine mode only) in the RISC-V Privileged specification. irq_i[11], irq_i[7], and irq_i[3] correspond to the Machine External Interrupt (MEI), Machine Timer Interrupt (MTI), and Machine Software Interrupt (MSI) respectively. The irq_i[31:16] interrupts are a CV32E40P specific extension to the RISC-V Basic (a.k.a. CLINT) interrupt scheme.
irq_ack_o	output	Interrupt acknowledge Set to 1 for one cycle when the interrupt with ID irq_id_o[4:0] is taken.
irq_id_o[4:0]	output	Interrupt index for taken interrupt Only valid when irq_ack_o = 1.



Block Diagrams

Exception/Interrupt Interface

Signal	Direction	Description
wb_ready_i	Input	Write-back ready signal
regfile_alu_we_i	Input	EX stage write enable to regfile
regfile_alu_waddr_i	Input	EX stage write address to regfile
alu_operator_i	Input	ALU operation code
alu_operand_a_i	Input	Input ALU operand A
alu_operand_b_i	Input	Input ALU operand B
alu_operand_c_i	Input	ALU operand C
alu_en_i	Input	ALU enable
mult_en_i	Input	Multiplier enable
mult_operator_i	Input	Multiplier operation code
mult_operand_a_i	Input	Multiplier operand A
mult_operand_b_i	Input	Multiplier operand B
mult_operand_c_i	Input	Multiplier operand C
mult_signed_mode_i	Input	Multiplier operands signed mode



Block Diagrams

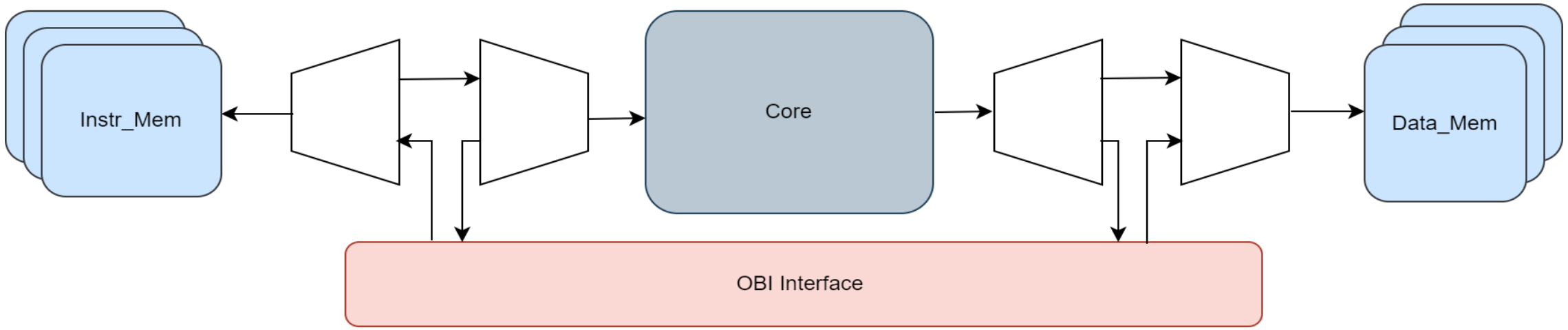
Exception/Interrupt Interface

Signal	Direction	Description
mult_multicycle_o	output	Multiplier is multicycle
regfile_alu_wdata_fw_o	output	EX stage write data to regfile(fw)
branch_decision_o	output	Branch decision output
jump_target_o	output	Jump target address
regfile_alu_waddr_fw_o	output	EX stage write address (fw)
regfile_alu_we_fw_o	output	EX stage write enable (fw)
ex_valid_o	output	EX stage valid output
ex_ready_o	output	EX stage ready output

UVM Environment for ALU

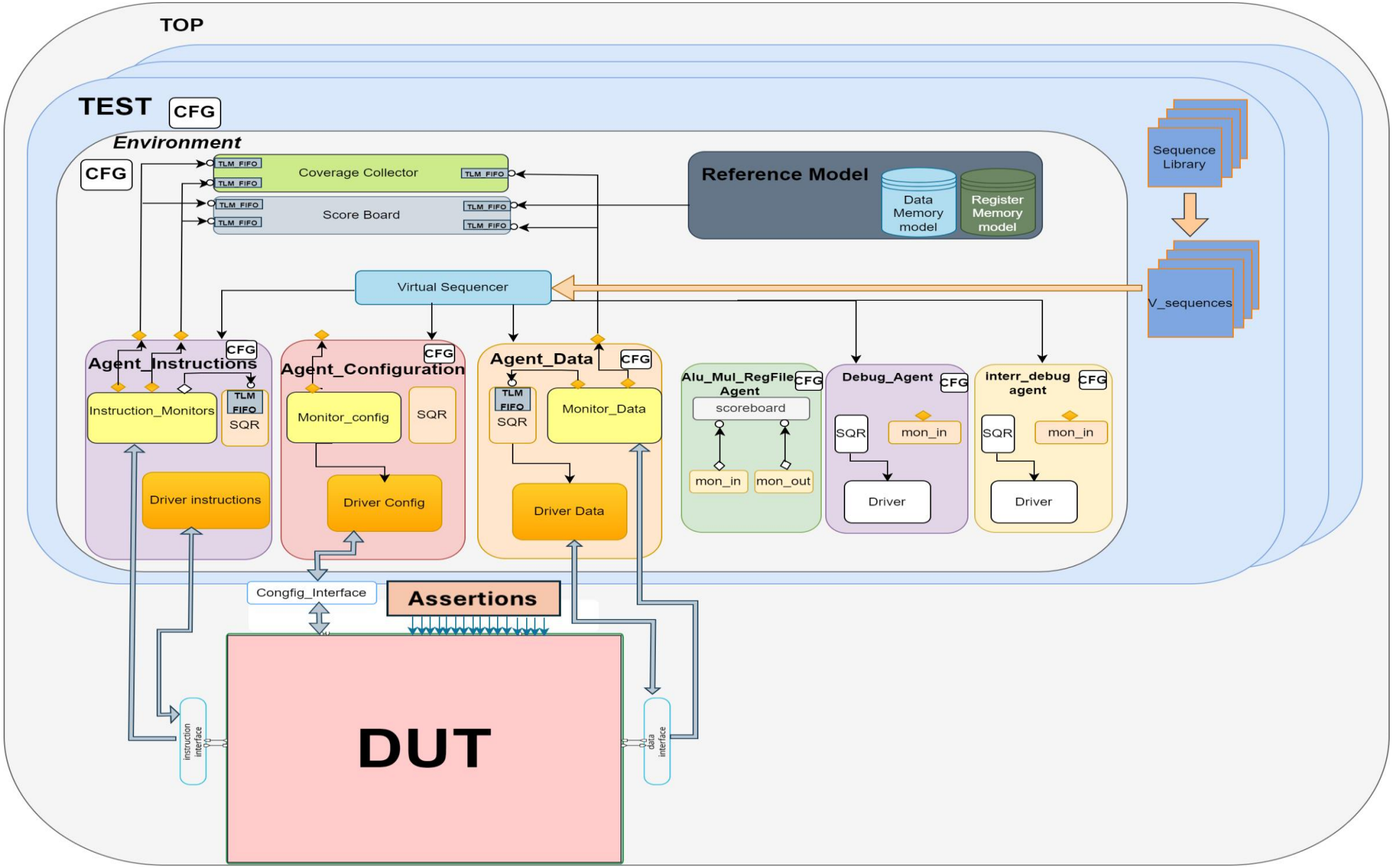
Virtual Interfaces

- The architecture has **multiple agents** so The DUT is connected to the testbench through interface owned to each agent.
- **An OBI interface** is used to allow access for **instruction memory** as well as **data memory** for read and write operations through the OBI interconnect as shown in figure.
- We have also interfaces a set of well-selected internal signals that were **binded** to help in Allowing **deep visibility** into core behavior, that will be described in details in following slides .



UVM Environment Architecture

UVM Environment for CV32E40P



UVM Environment for CV32E40P

Components overview

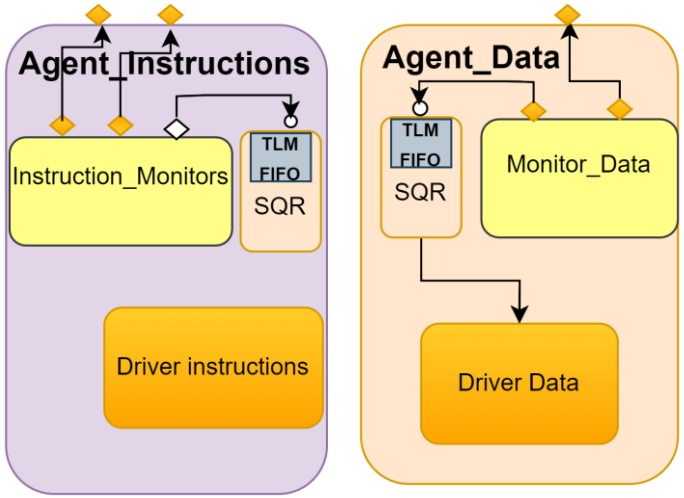
UVM Agents:

Instruction Agent and Data Agent are classified as **reactive agents** where agent has to generate responses based on the observed **requests**. Typically the response information and flow control signals are required to be related to the requests provided by the DUT

The agent retrieves its configuration (instruction_config) from the **UVM config database** → this is set by the environment (env).

- If agent_active == **UVM_ACTIVE** --- >> It creates the sequencer and driver.
- Regardless of mode, it always creates --- >> The input monitor (in_mon)

The output monitor (out_mon)

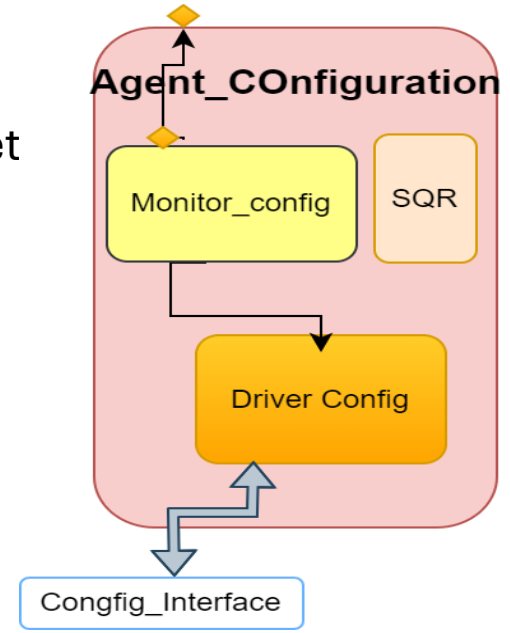


A monitor is a passive UVM component that observes **DUT signals** and converts them into high-level transaction objects. It sends these transactions to both the **sequencer's analysis FIFO** (for reactive stimulus) and the **scoreboard** (for checking correctness). This allows **reactive sequences** to respond to DUT activity, such as detecting when **instr_req_o** is high before sending a valid instruction.

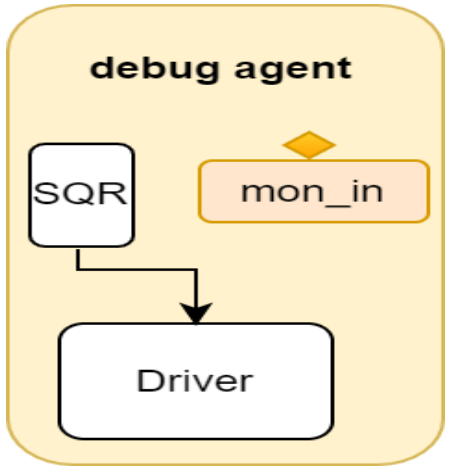
UVM Environment for CV32E40P

Components overview

Configuration Agent is classified as **active agent** to initialize the RISC-V core by asserting reset and applying startup settings like boot_addr, mtvec_addr, and fetch_enable.
It runs **once** at the beginning of simulation to apply necessary configuration values and safely release reset.
This agent ensures the core begins in a known and controlled state, ready for functional testing.



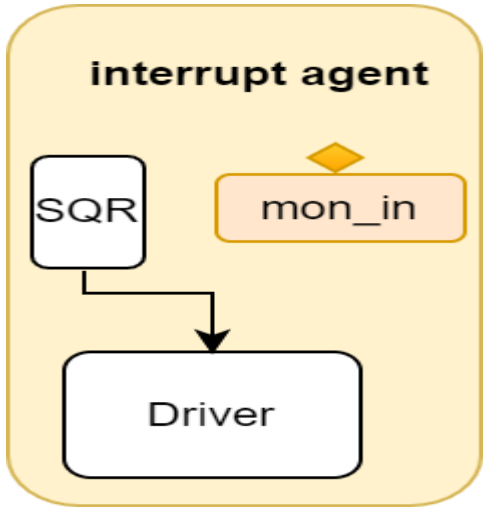
Debug Agent is a **Active agent** that holds all debug interface signals at **zero** during simulation.
It does not generate transactions but ensures the core behaves correctly when the debug interface is idle.
This setup mimics real hardware scenarios where the debug port is **unused**, and prevents **unintended debug activity** from affecting the core.



UVM Environment for CV32E40P

Components overview

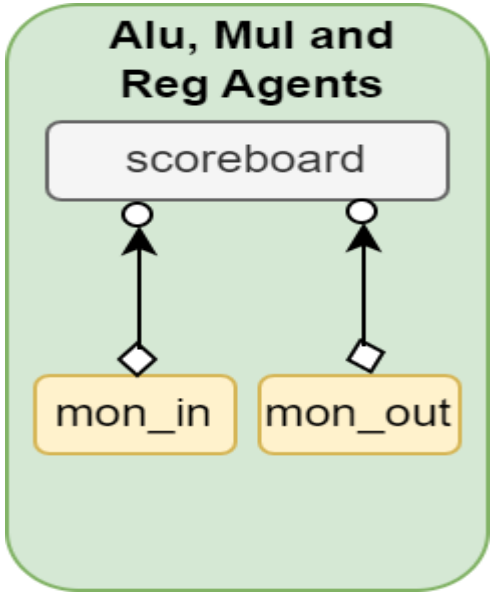
Interrupt Agent is a **Active agent** that forces all interrupt lines **(irq_i)** to zero during simulation.
Used to Isolate Interrupt Effects by keeping interrupts disabled, it ensures deterministic behavior during early-stage or focused testing.



UVM Environment for CV32E40P

Components overview

ALU, Mul and Register Agent passively monitors arithmetic and logical operations, capturing input operands, opcodes, and results without influencing the DUT. Also tracks multiplication instructions, observing signed and unsigned multiplication results to ensure correct functionality. verifies register read and write operations by monitoring source and destination register addresses and their data values.
Each of them includes a monitor that sends observed transactions to a scoreboard for comparison with a golden model.



UVM Environment for CV32E40P

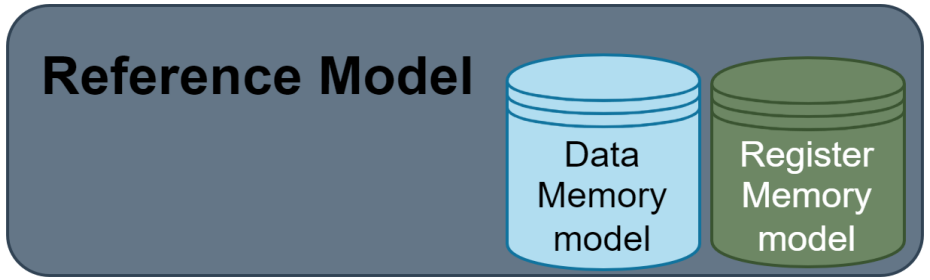
Components overview

Reference Model

In our environment we used Data **Storage_Model** and **Reg_File_Model** for explicit prediction of read, write data and data loaded in register file respectively

The data memory sequnces have to **write values** in Storage Model

The execution and writeback stages control signals and data are **binded** and used to write data in Reg_File Model which will be used in scoreboard later.



UVM Environment for CV32E40P

Components overview

Reg_file Reference Model

Register File Model (Rf_Model) mimics the behavior of the core's register file by storing and tracking the state of 32 general-purpose registers.

It passively receives transactions through a **TLM FIFO**, updates internal reference memory for write operations, and generates expected read data for scoreboard comparison.

Writes to register **x0 are ignored** as per the RISC-V specification, maintaining architectural correctness.

UVM Environment for CV32E40P

Components overview

Data Storage Model

Storage_Model represents a reference memory model used to validate load and store transactions from the core.

It is designed to support **misaligned accesses** by handling them as multiple byte-wise transactions, mimicking the CV32E40P core's behavior.

For misaligned word or halfword accesses crossing word boundaries, the model performs two **aligned memory transactions**, always starting with the **lower address**.

During write operations, only enabled bytes are updated using the **data_enable** mask, ensuring correct partial writes.

UVM Environment for CV32E40P

Components overview

UVM Score Board

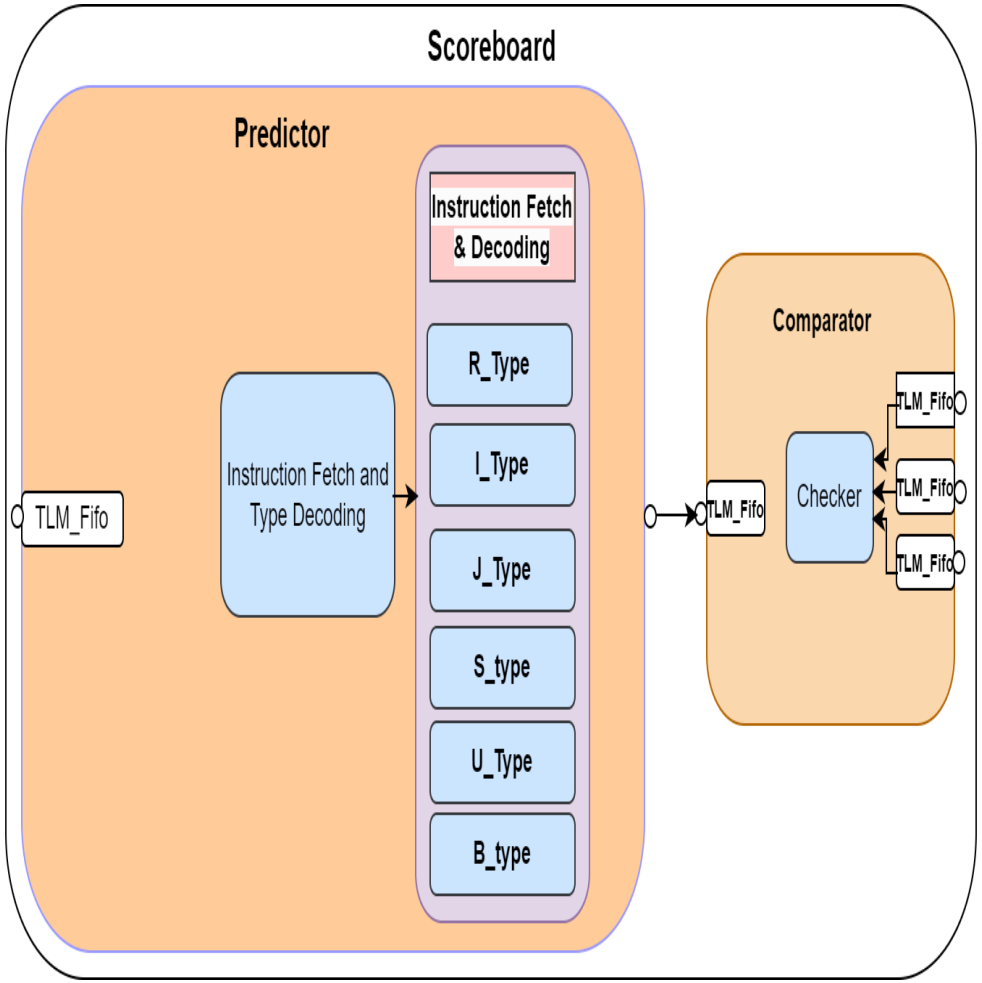
- The scoreboard in our case is to check that the instruction got in the process as expected and taken data is written in a right way to the system from the memory model.
- So it takes data from the instruction agent and the data agent as well.

UVM Environment for CV32E40P

Components overview

UVM Score Board

- **Predictor Module** models the expected behavior of the DUT by **decoding** and executing instructions (R, I, S, B, U types) using reference models for register memory.
- Comparator Module receives actual DUT outputs and predicted results via **TLM FIFOs**, comparing them cycle-accurately using the Checker.
- TLM FIFO Channels ensure synchronized communication between monitors, predictor, and comparator, enabling non-intrusive and modular verification.
- Scoreboard Goal is to validate the functional correctness of instruction execution, memory operations, and register updates through reference modeling and result comparison.



UVM Environment for CV32E40P

Components overview

Coverage Collector

Our cover model acts as a metric of how through the core has been verified and exercised

functional coverage is a metric showing how much of the design functionalities are covered as per the verification requirements. These requirements are directly mapped into cover groups in the model. This functional coverage model aimed to ensure that each instruction in the standard instruction sets is executed **at least once**. Instructions and operands are covered with planned values and order to make sure our DUT is free from **hazards** in case of **pipelining** dependencies on registers.

UVM Environment for CV32E40P

Si-VISION
INNOVATIVE IC SOLUTIONS

Assertions

- Ensures the **request** stays valid until it's accepted by memory and verifying the instruction data is stable when marked as **valid**.
- Ensures the core doesn't drop its **memory request** before it’s accepted and verifies that read data returns correctly after a **granted read**.
- Guarantees that read data is marked valid for only a **single clock** for each outstanding.
- Ensures that write data doesn’t **change** while the request is being accepted.
- Validates correct **partial or full word access** during memory operations.

Sequences and Tests

Sequences and Tests

Instruction Sequences | Sequences

Sequence	Describtion
R_Type_Sequence	Used to generate R-Type register instructions
I_Type_Sequence	Used to generate I-Type register instructions
L_Type_Sequence	Used to generate L-Type Load instructions
S_Type_Sequence	Used to generate S-Types instructions
B_Type_Sequence	Used to generate B-Type instructions
J_Type_Sequence	Used to generate J-Type instructions (each of JAL,JALR)
U_Type_Sequence	Used to generate U-Type instructions(each of AUIPC,LUI)
M_Type_Sequence	Used to generate M-Type instructions

Sequences and Tests

Instruction Sequences | Sequences

Sequence	Describtion
Rand_seq	Random Sequence used to generate Different Type of Instructions
Zero_Ones_I_Sequence	Sequence for I_type (immediate) to generate all ones, Zeros, Min, Max Values if imm
Zero_Ones_L_Sequence	Sequence for I_type (load) to generate all ones, Zeros, Min, Max Values if imm
Zero_Ones_JALR_Sequence	Sequence for J_type (JALR) to generate all ones, Zeros, Min, Max Values if imm
Zero_Ones_Min_Max_AUIPC_Sequence	Sequence for U_type to generate all ones, Zeros, Min, Max Values if imm

Sequences and Tests

Instruction Sequences | Sequences

Sequence	Describtion
Zero_Ones_Min_Max_J_Seq	Sequence for J_type (JAL) to generate all ones, Zeros, Min, Max Values if imm.
Zero_Ones_Min_Max_LUI_Seq	Sequence for U_type (LUI) to generate all ones, Zeros, Min, Max Values if imm.
SUB_SRA_Sequence	Sequence for R_type to generate instructions includes SUB and SRA Functions
SLLI_SRAI_SRLI_Sequence.sv	Sequence for I_type to generate instructions includes SLLI, SRAI and SRLI Functions
Rst_Sequence	Sequence to rest core
Configure_Sequence	Sequence to initialize and apply settings of the core
Data_Sequence	Sequence formed based on request data to configure the data memory

Sequences and Tests

Instruction Sequences | Virtual Sequences

Sequence	Describtion
V_R_Type_Sequence	Used to generate R-Type register instructions
V_I_Type_Sequence	Used to generate I-Type register instructions
V_L_Type_Sequence	Used to generate L-Type Load instructions
V_S_Type_Sequence	Used to generate S-Types instructions
V_B_Type_Sequence	Used to generate B-Type instructions
V_J_Type_Sequence	Used to generate J-Type instructions (each of JAL,JALR)
V_U_Type_Sequence	Used to generate U-Type instructions(each of AUIPC,LUI)
V_M_Type_Sequence	Used to generate M-Type instructions

Sequences and Tests

Instruction Sequences | Virtual Sequences

Sequence	Describtion
V_Rand_seq	Random Sequence used to generate Different Type of Instructions
V_Zero_Ones_I_Sequence	Sequence for I_type (immediate) to generate all ones, Zeros, Min, Max Values if imm
V_Zero_Ones_L_Sequence	Sequence for I_type (load) to generate all ones, Zeros, Min, Max Values if imm
V_Zero_Ones_JALR_Sequence	Sequence for J_type (JALR) to generate all ones, Zeros, Min, Max Values if imm
V_Zero_Ones_Min_Max_AUIPC_ Sequence	Sequence for U_type to generate all ones, Zeros, Min, Max Values if imm

Sequences and Tests

Instruction Sequences | Vitual Sequences

Sequence	Describtion
V_Zero_Ones_Min_Max_J_Seq	Sequence for J_type (JAL) to generate all ones, Zeros, Min, Max Values if imm.
V_Zero_Ones_Min_Max_LUI_Seq	Sequence for U_type (LUI) to generate all ones, Zeros, Min, Max Values if imm.
V_SUB_SRA_Sequence	Sequence for R_type to generate instructions includes SUB and SRA Functions
V_SLLI_SRAI_SRLI_Sequence.sv	Sequence for I_type to generate instructions includes SLLI, SRAI and SRLI Functions
Rst_Sequence	Sequence to rest core
Configure_Sequence	Sequence to initialize and apply settings of the core
Data_Sequence	Sequence formed based on request data to configure the data memory

Sequences and Tests

Instruction Tests

Test_Type	Describtion
Base Test	initializes the complete verification environment, including all major UVM agents (instruction, data, interrupt, debug, and config). It runs a reset sequence to initialize the DUT and then applies a configuration sequence to set up memory and register states.
R_Test	A virtual sequence that runs R-type and I-type instruction sequences on the instruction sequencer in parallel with a data memory sequence on the data sequencer. validates the processor's execution of **R-type** instructions and verify correct ALU operations and data path behavior for register-to-register instructions.
I_Test	A virtual sequence that runs L-type and I-type instruction sequences concurrently on the instruction sequencer while also running a data memory sequence on the data sequencer. stimulates the instruction memory interface with randomized I-type and L-type instructions.

Sequences and Tests

Instruction Tests

Test_Type	Description
S_Test	This test targets S-type instructions and ensure that store operations are issued correctly. It also runs I-type instructions and a memory data sequence in parallel to simulate realistic processor activity.
B_Test	It runs B-type and I-type sequences in parallel to stimulate realistic instruction streams, while data memory responses are provided concurrently. this test verifies the execution of B-type instructions to ensure the core correctly responds to branch conditions, addresses, and instruction validity.
J_Test	This test targets the functionality of J-type instructions and ensures that jump targets are valid and examines whether the core properly commits the jump destinations.
U_Test	This test targets **U-type instructions** It runs a randomized stream of U-type instructions in parallel with I-type instructions and data memory transactions.
M_Test	It runs M-type and I-type instruction sequences in parallel This test validates the core's handling of M-type ensure the core properly recognizes and executes multiplication instructions using the correct opcode and funct7 patterns.

Sequences and Tests

Instruction Tests

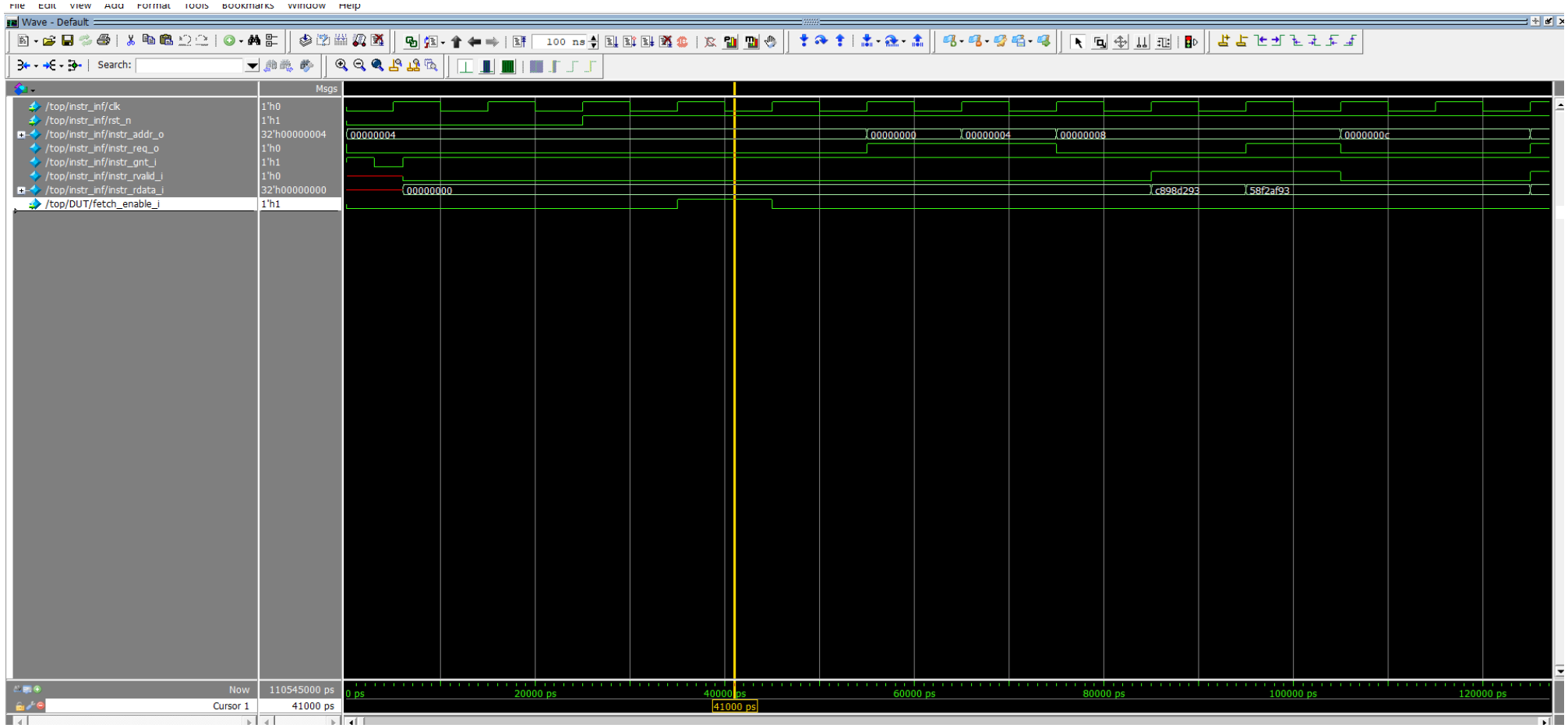
Sequence	Describtion
Rand_test	This test verifies instruction and data memory interface behavior under randomized instruction streams.
Zero_Ones_I_Test	Direct test targets the **Zeros, Ones, Min, Max imm values** of I-Type instruction.
Zero_Ones_L_Test	Direct test targets the **Zeros, Ones, Min, Max imm values** of I-Type load instruction.
Zero_Ones_JALR_Test	Direct test targets the **Zeros, Ones, Min, Max imm values** of J-Type jump and link register instruction.
Zero_Ones_Min_Max_J_Test	Direct test targets the **Zeros, Ones, Min, Max imm values** of J-Type jump and link instruction.
Zero_Ones_Min_Max_AUIPC_Test	Direct test targets the **Zeros, Ones, Min, Max imm values** of U-Type add_upper_immediate to PC instruction.
Zero_Ones_Min_Max_LUI_Test	Direct test targets the **Zeros, Ones, Min, Max imm values** of U-Type load_upper_immediat instruction.
SUB_SRA_Test	Direct test targets the **SUB** and **SRA** functions of R-Type instruction.

Analysis and Results based Testing Scenarios

Simulation Results

Simulation Results | Fetch_Enable

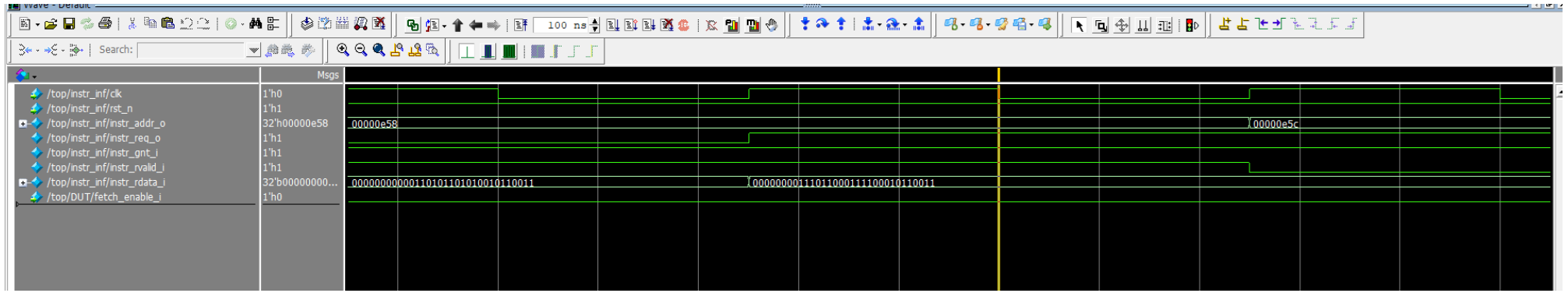
WaveForm

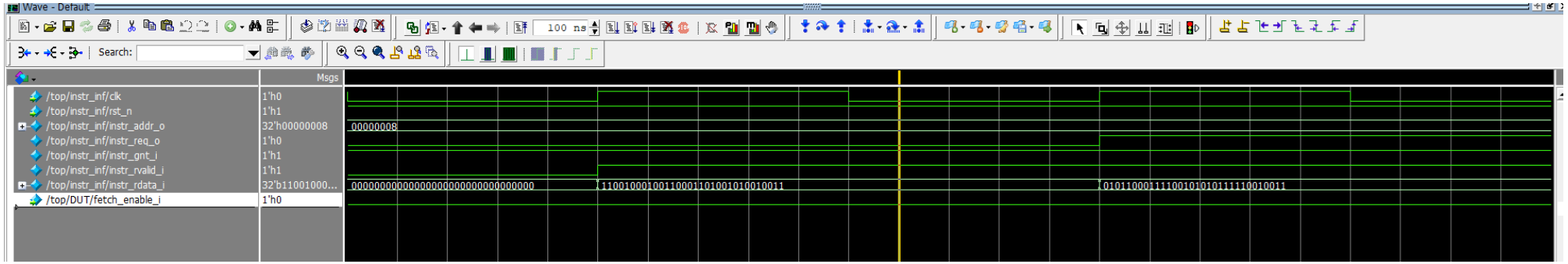


Simulation Results

Simulation Results | R_Test , I_Test

WaveForm

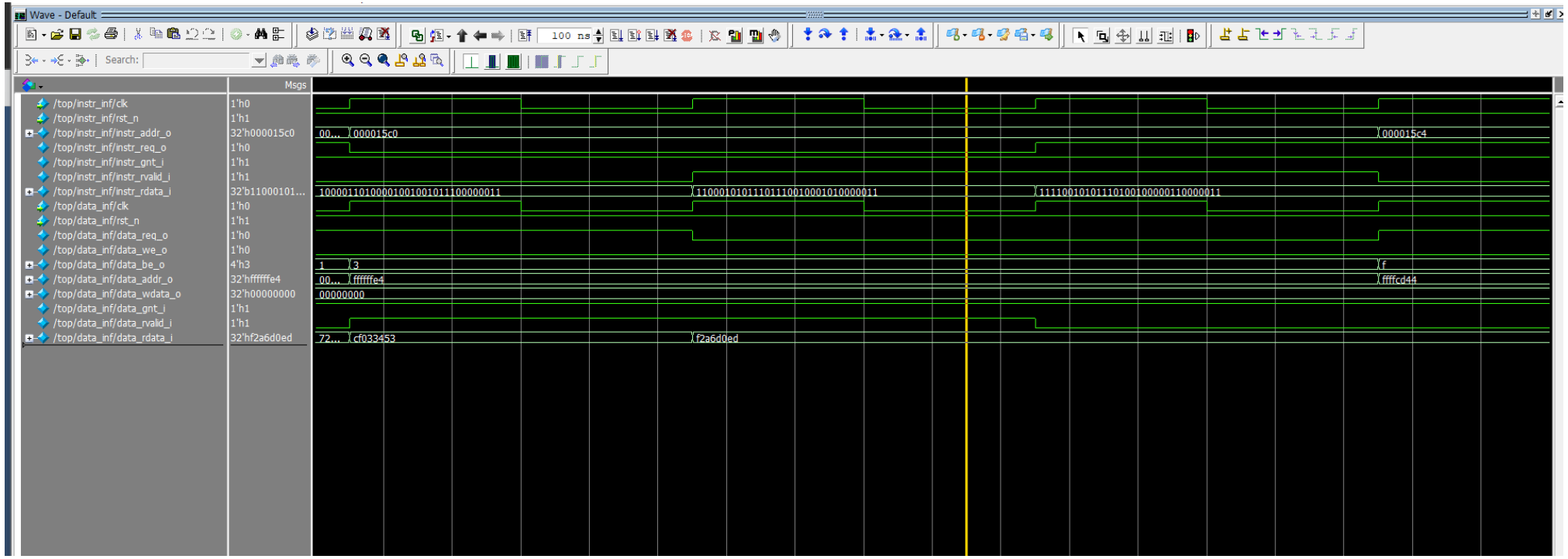




Simulation Results

Simulation Results | Load_Test, Store_Test

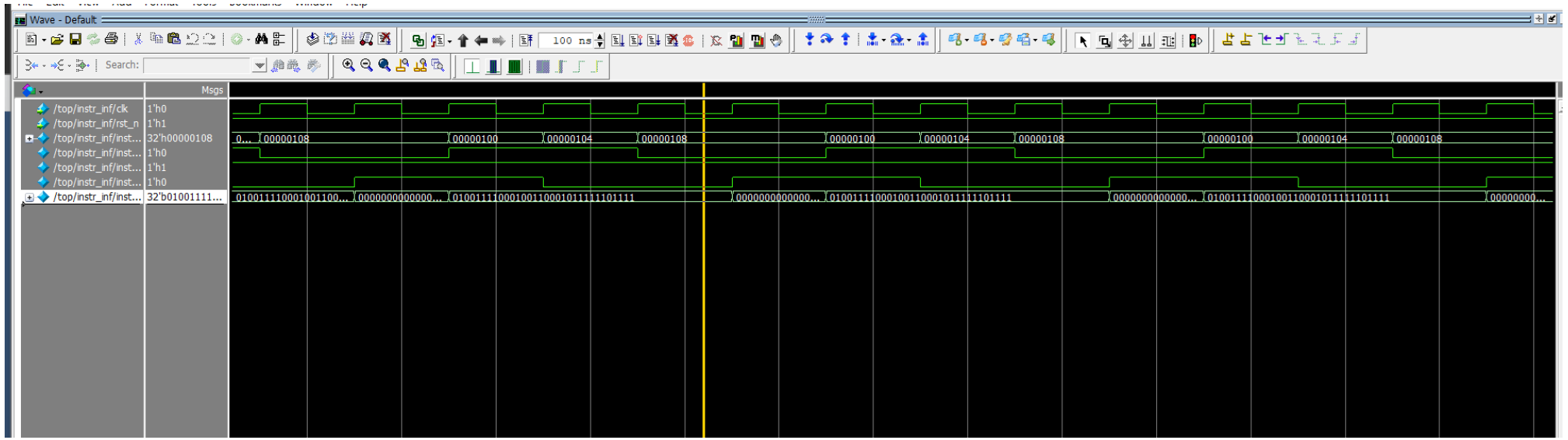
WaveForm



Simulation Results

Simulation Results | Jump_Test

WaveForm



Functional Coverage Analysis

The coverage report confirms **100% coverage** across group metric. This achievement demonstrates that our covergroups and goals have been thoroughly exercised,

SYNOPSYS

Dashboard

dashboard | hierarchy | modlist | groups | tests | asserts

Version: V-2023.12-1

Command line: urg -dir Vdb_Files/base_test Vdb_Files/base_test.vdb Vdb_Files/B_instr_test Vdb_Files/B_instr_test.vdb Vdb_Files/compile_cov.vdb Vdb_Files/J_instr_test Vdb_Files/J_instr_test.vdb Vdb_Files/L_instr_test Vdb_Files/L_instr_test.vdb Vdb_Files/MUL_instr_test Vdb_Files/MUL_instr_test.vdb Vdb_Files/rand_test Vdb_Files/rand_test.vdb Vdb_Files/R_instr_test Vdb_Files/R_instr_test.vdb Vdb_Files/S_instr_test Vdb_Files/S_instr_test.vdb Vdb_Files/slli_srai_srli_i_test Vdb_Files/slli_srai_srli_i_test.vdb Vdb_Files/sub_sra_test Vdb_Files/sub_sra_test.vdb Vdb_Files/U_instr_test Vdb_Files/U_instr_test.vdb Vdb_Files/zero_ones_i_test Vdb_Files/zero_ones_i_test.vdb Vdb_Files/zero_ones_jalr_test Vdb_Files/zero_ones_jalr_test.vdb Vdb_Files/zero_ones_l_test Vdb_Files/zero_ones_l_test.vdb Vdb_Files/zero_ones_min_max_auipc_test Vdb_Files/zero_ones_min_max_auipc_test.vdb Vdb_Files/zero_ones_min_max_j_test Vdb_Files/zero_ones_min_max_j_test.vdb Vdb_Files/zero_ones_min_max_lui_test Vdb_Files/zero_ones_min_max_lui_test.vdb -dbname Merged_db -report Merged_Cov_Report -format both -elfile ex.el

Number of tests: 17

Total Coverage Summary

SCORE	LINE	COND	TOGGLE	FSM	BRANCH	ASSERT	GROUP
54.40	43.76	48.10	40.85	28.57	46.10	73.44	100.00

Hierarchical coverage data for top-level instances

SCORE	LINE	COND	TOGGLE	FSM	BRANCH	ASSERT	NAME
62.50						62.50	pack
51.24	43.83	48.11	40.85	28.57	46.07	100.00	top
55.05	31.82	33.33			100.00		uvm_custom_install_recording
33.33						33.33	uvm_pkg

Total Module Definition Coverage Summary

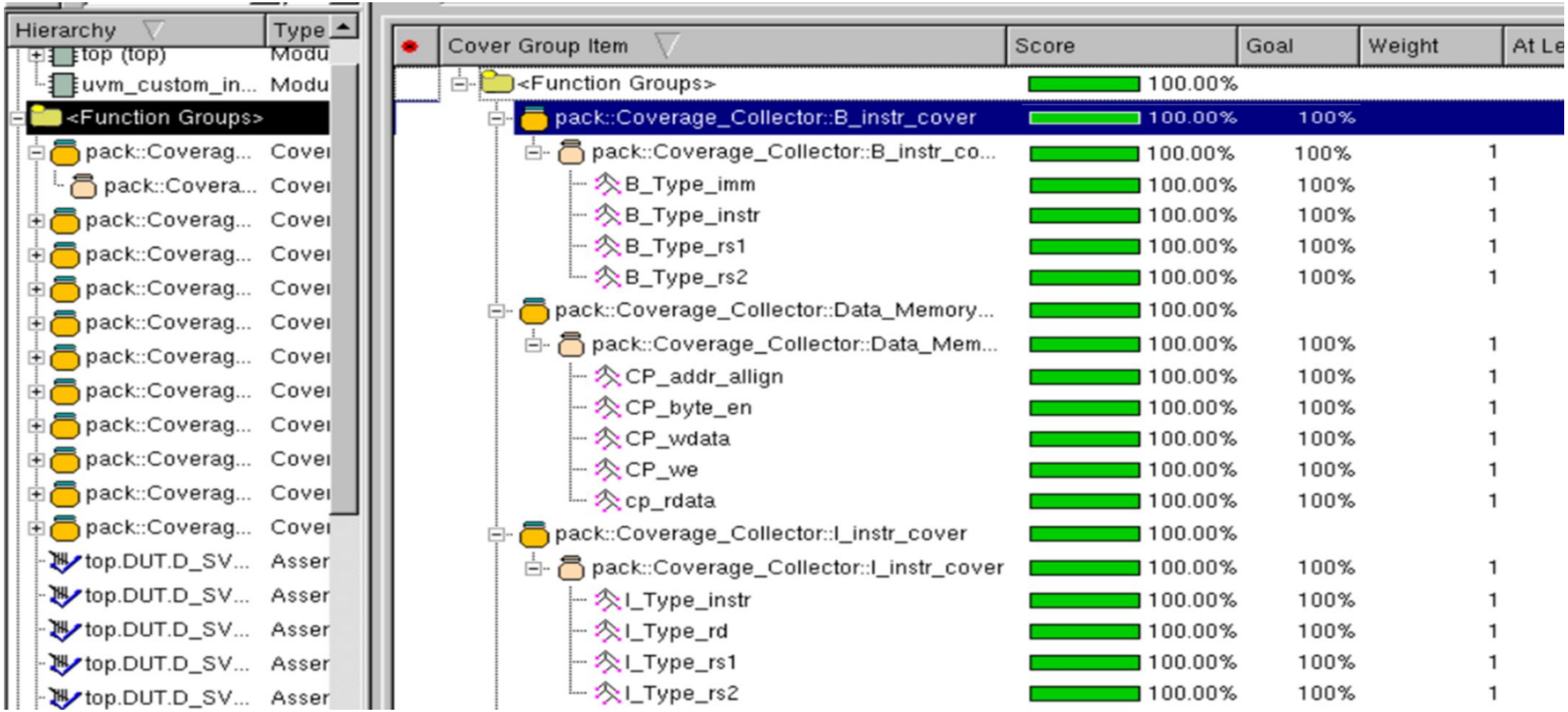
SCORE	LINE	COND	TOGGLE	FSM	BRANCH	ASSERT
45.68	43.76	48.08	35.36	28.57	45.99	72.31

Total Groups Coverage Summary

SCORE	WEIGHT
100.00	1

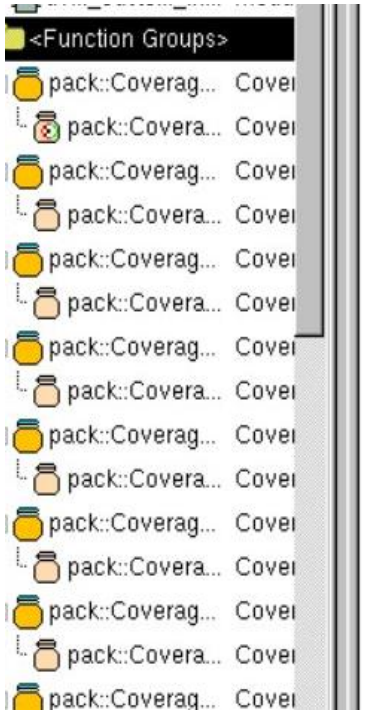
Functional Coverage Analysis

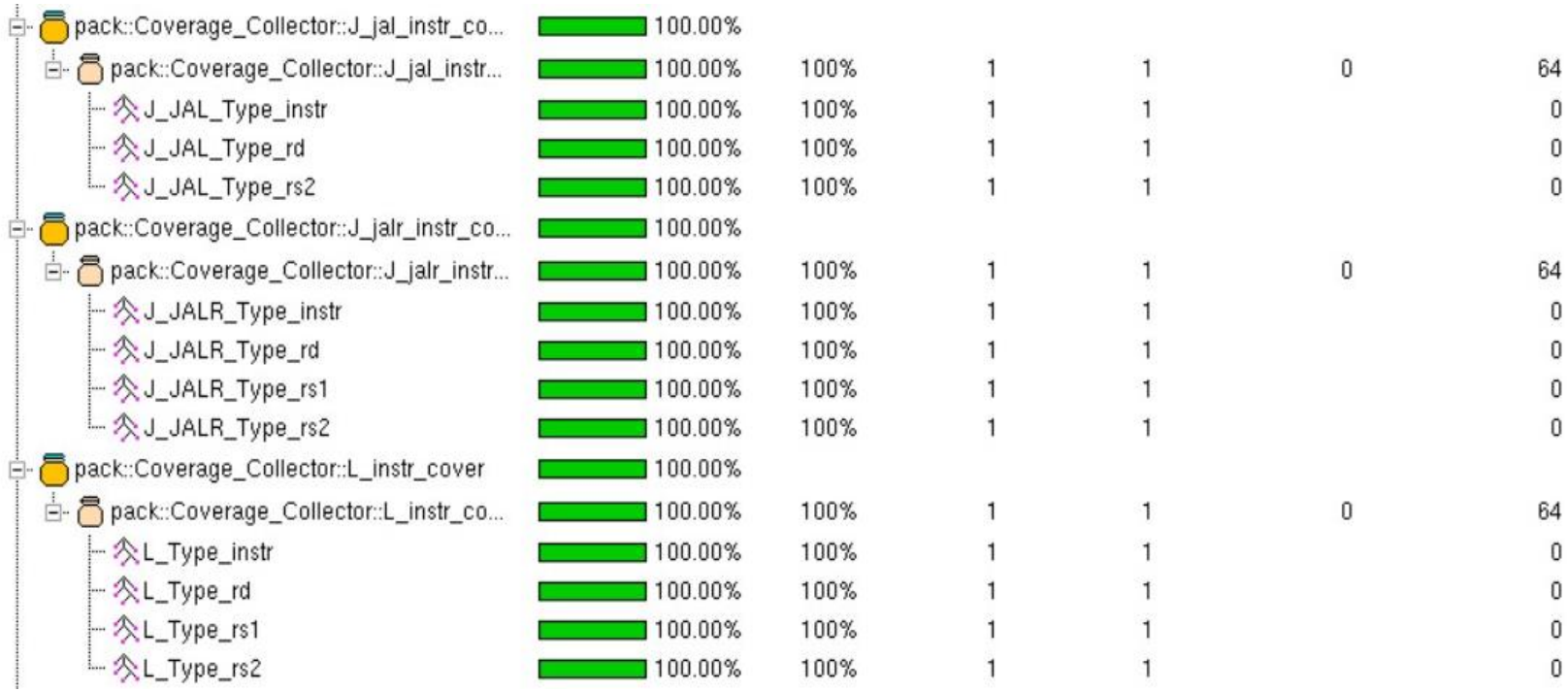
The coverage report confirms **100% coverage** across group metric. This achievement demonstrates that our covergroups and goals have been thoroughly exercised,



Functional Coverage Analysis

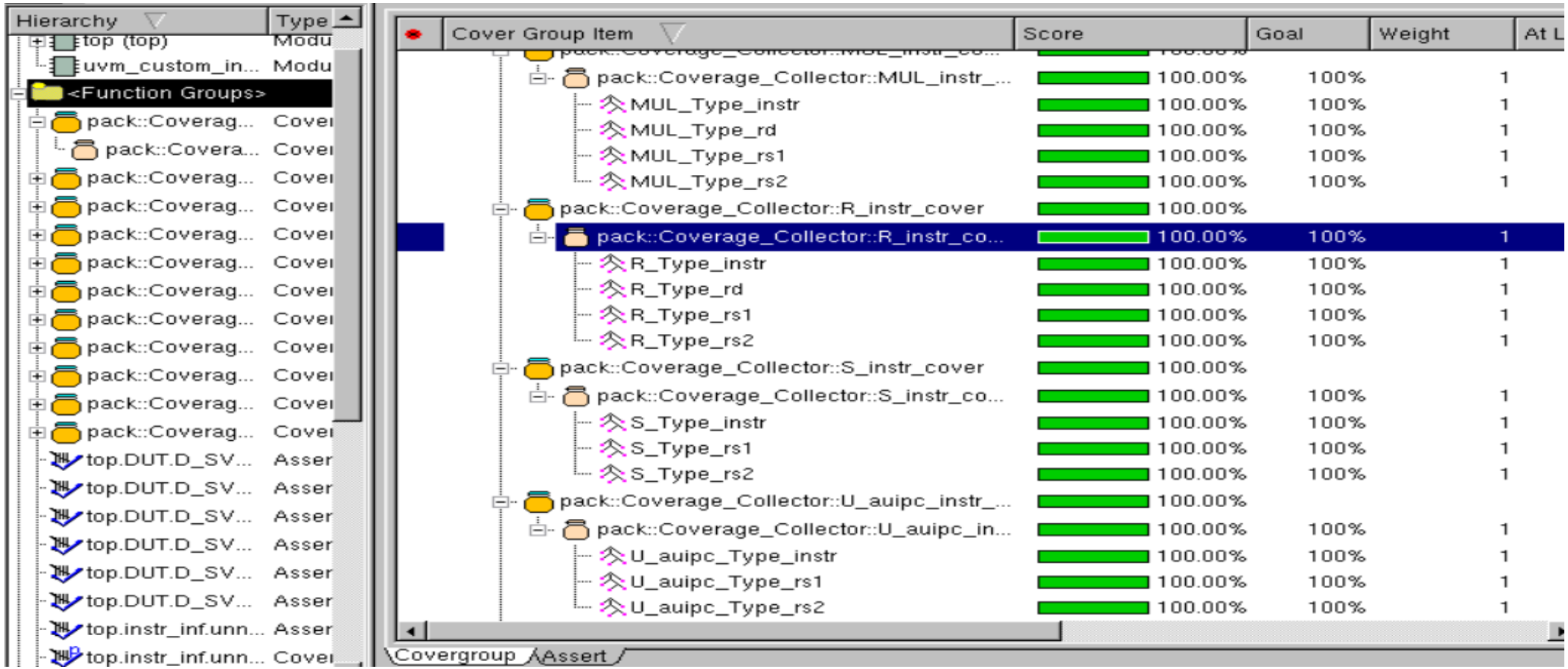
As shown The functional coverage achieved **100% bin coverage** for J_type (jal, jalr), load type instructions.





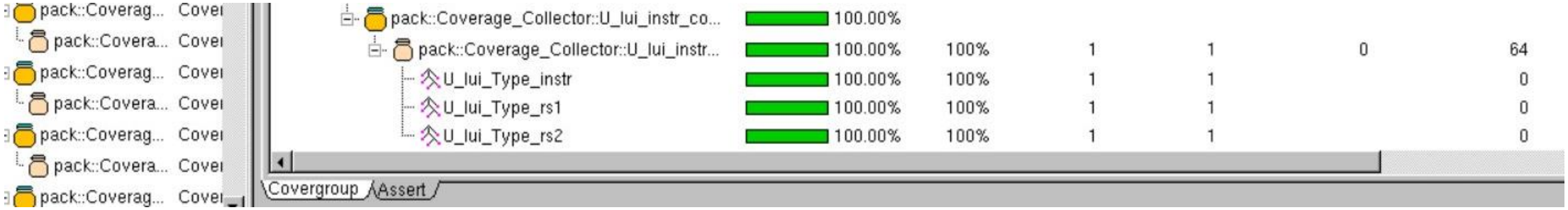
Functional Coverage Analysis

As shown The functional coverage achieved **100% bin coverage** for M-Extension , R_type , S_type and U_Type instructions.



Functional Coverage Analysis

As shown The functional coverage achieved **100% bin coverage** for U_type (lui) instructions.



Coverage Analysis

Assertions

```
108 Byte_Enable: assert property(Byte_Enable_Check)
109         //$display("Byte_Enable: Success assertion pass");
110
111         else
112         $display("Byte_Enable: Failed assertion ");
```

/home/svgpdv25mohaaban/old/RISC-V_UVM_Verification/Assertions/Data_Assertions.sv

Test: MergedTest

Instance	Assert Name	Attempts	Real Successes	Failures	Incompletes	Category	Severity
<Funct...	User defined Group	1279470	149005	2482	0		
top....	Byte_Enable	85298	4530	0	0	0	0
top....	Contigous	85298	877	773	0	0	0
top....	Max_2_Outstanding	85298	4529	0	0	0	0
top....	R_Valid	85298	79	1709	0	0	0
top....	Read_Data	85298	2808	0	0	0	0
top....	Req_till_Gnt	85298	4530	0	0	0	0
top....	Write_Data	85298	1721	0	0	0	0
top....	Address_aligned	85298	40639	0	0	0	0
top....	Branch_Instructions	85298	1089	0	0	0	0
top....	JALR_Instruction	85298	935	0	0	0	0
top....	Legal_instr_rdata_i	85298	40647	0	0	0	0
top....	Load_Instructions	85298	2234	0	0	0	0
top....	R_type_Instructions	85298	2467	0	0	0	0
top....	Store_Instructions	85298	1273	0	0	0	0
top....	valid_opcodes	85298	40647	0	0	0	0

Covergroup | Assert

/home/svgpdv25mohaaban/old/RISC-V_UVM_Verification

the End

Thank You