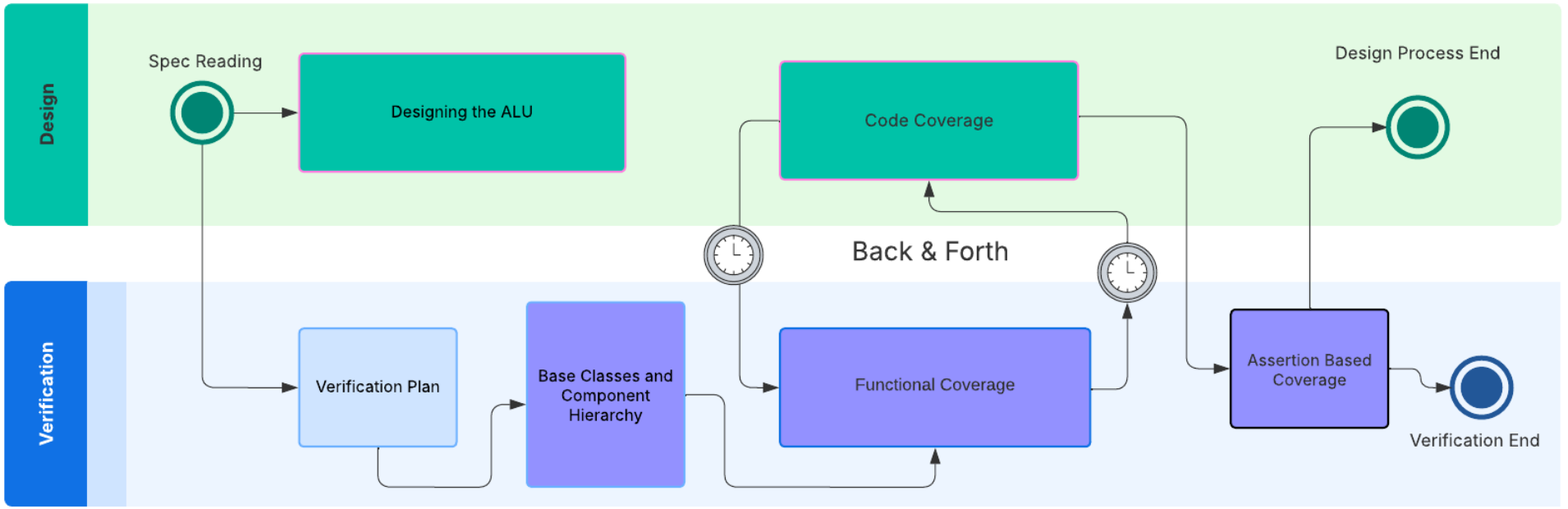
Design & Verification of an ALU using UVM & SVA

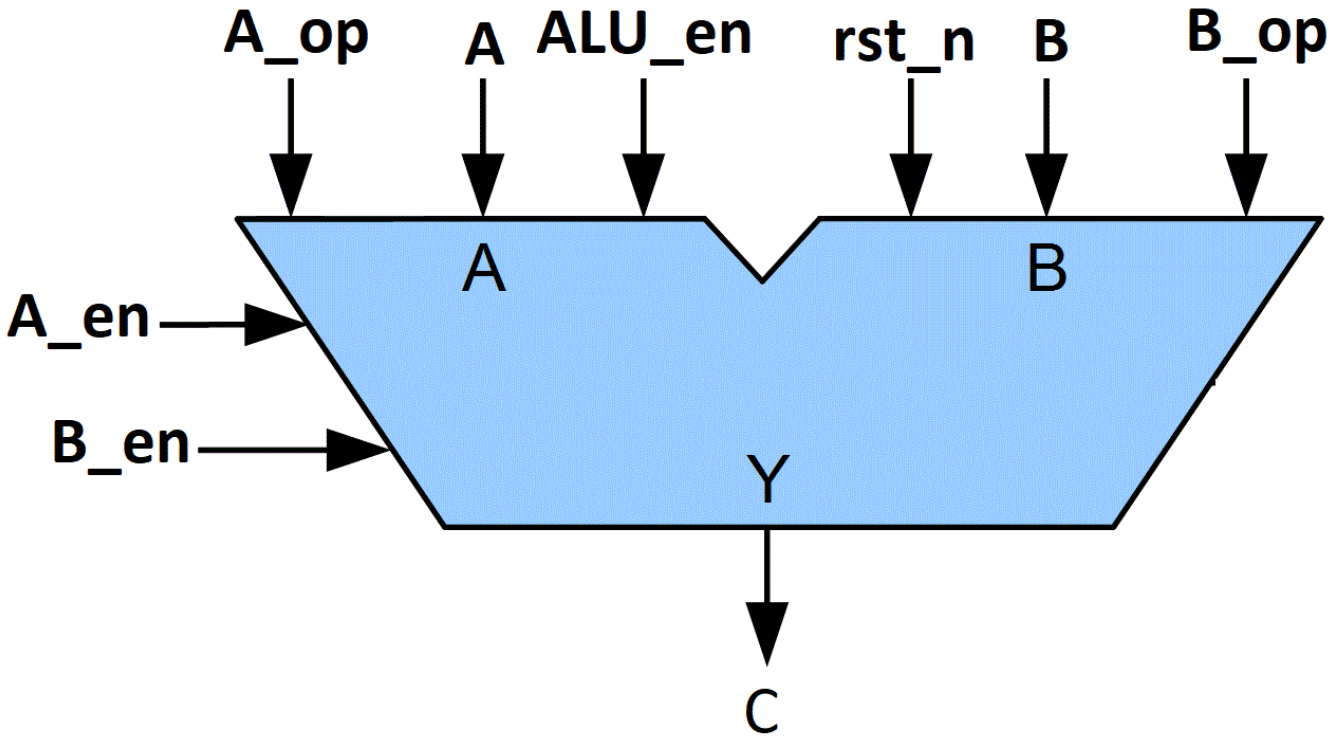
Abdelrahman Mohamed Yassien

dir:
/home/svgpdv25abohamad/alu_uvm_project

Project Timeline



ALU DESIGN SPECIFICATIONS



(A_EN & ~B_EN)

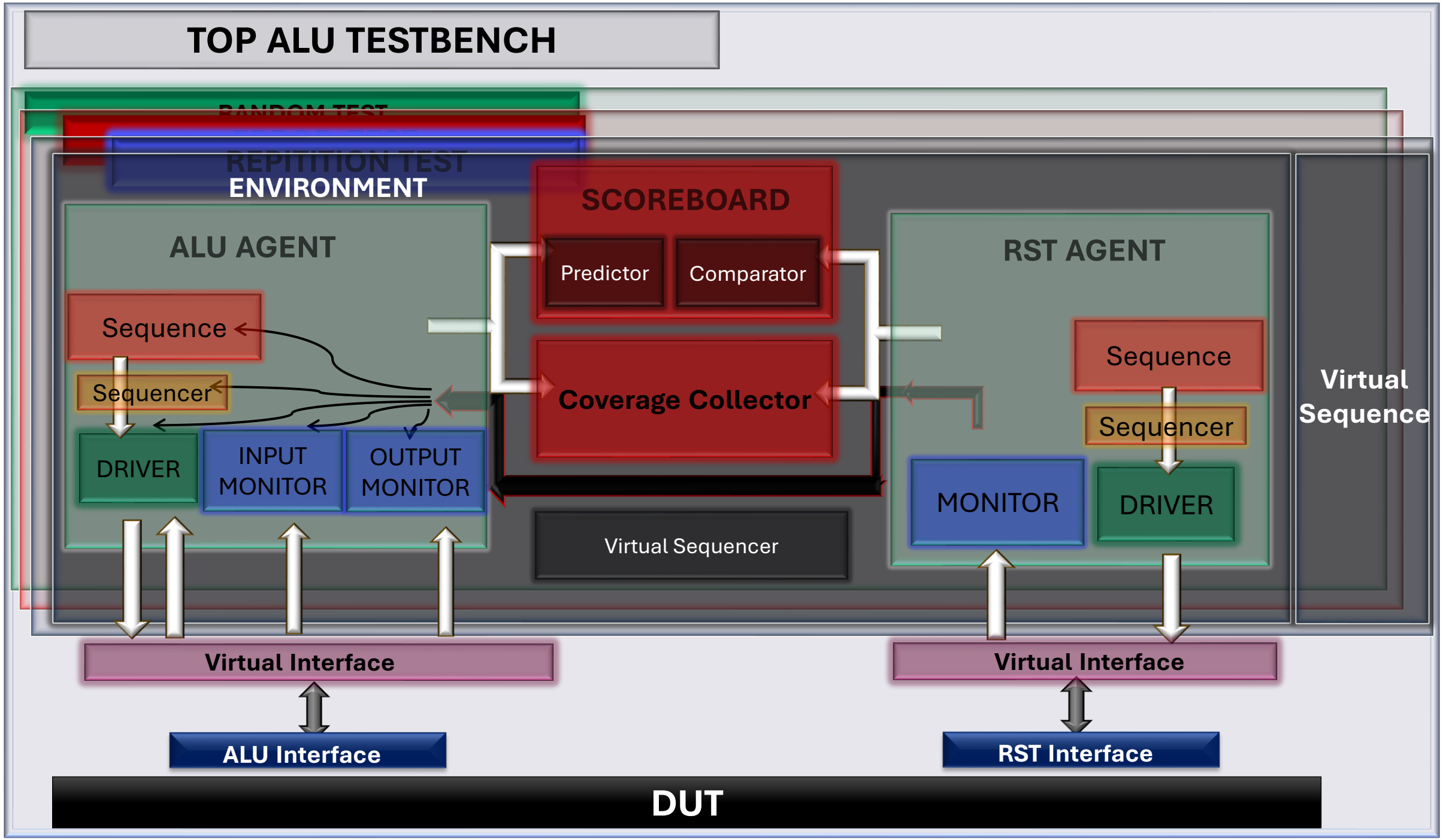
A_op	C = ?	Description
0	A + B	Sum of inputs
1	A – B	B subtracted from A
2	A ^ B	A bitwise XOR B
3	A & B	A bitwise & B
4	A & B	A bitwise & B
5	A | B	A bitwise OR B
6	~(A ^ B)	A XNOR B
7	NULL	Not permitted to take this value

(A_EN & B_EN)

B_op	C= ?	Description
0	A + B	Sum of inputs
1	A – B	B subtracted from A
2	A ^ B	A bitwise XOR B
3	A & B	A bitwise & B

(~A_EN & B_EN)

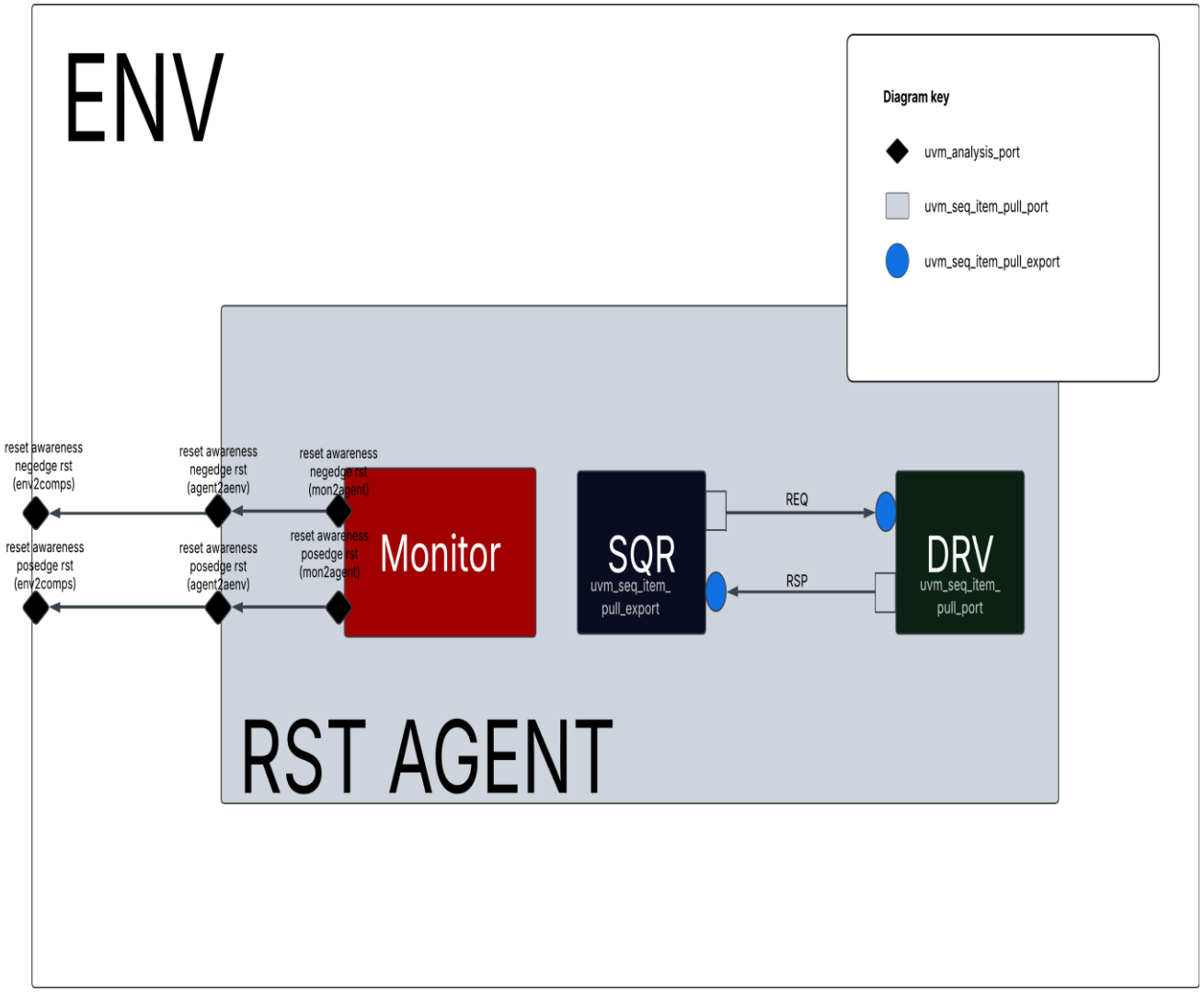
B_op	C = ?	Description
0	~(A & B)	A NAND B
1	A + B	Sum of inputs
2	A + B	Sum of inputs
3	NULL	Not permitted to take this value


TOP ALU TESTBENCH
RANDOM TEST
REPITITION TEST
ENVIRONMENT
ALU AGENT
Sequence
Sequencer
DRIVER
INPUT MONITOR
OUTPUT MONITOR
SCOREBOARD
Predictor
Comparator
Coverage Collector
Virtual Sequencer
RST AGENT
Sequence
Sequencer
MONITOR
DRIVER
Virtual Sequence
Virtual Interface
Virtual Interface
ALU Interface
RST Interface
DUT

RST AGENT TLM DIAGRAM

- The reset awareness was inspired from The published paper [Handling Asynchronous Reset(s) Testing by building reset-awareness into UVM testbench components by Wei Wei Cheong, Katherine Garden, Ana Sanz Carretero which was previewed @DVCON 2021]

- As shown in the figure, the rst agent monitor class is always waiting for reset assertion, and as soon as it is asserted, the monitor class writes to the uvm_tlm_analysis_port so it reaches every reset_aware component and hence allows them to act accordingly.

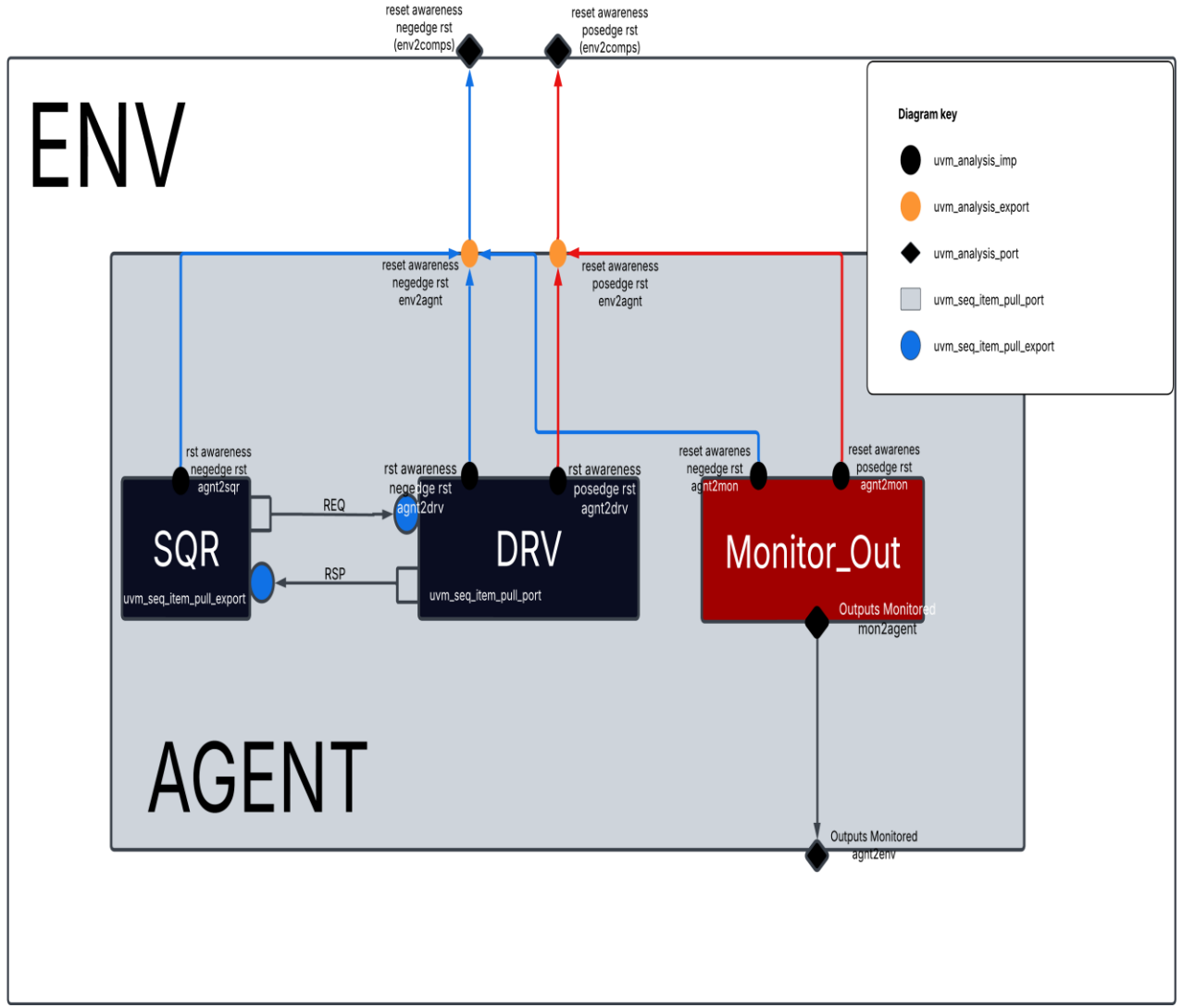


ALU AGENT TLM DIAGRAM

- The reset awareness was inspired from The published paper [Handling Asynchronous Reset(s) Testing by building reset-awareness into UVM testbench components by Wei Wei Cheong, Katherine Garden, Ana Sanz Carretero which was previewed @DVCON 2021]

- As shown in the figure, the alu agent is fully reset aware, every component is connected with uvm_tlm_analysis_imp s to allow it to react to asynchronous reset assertion.

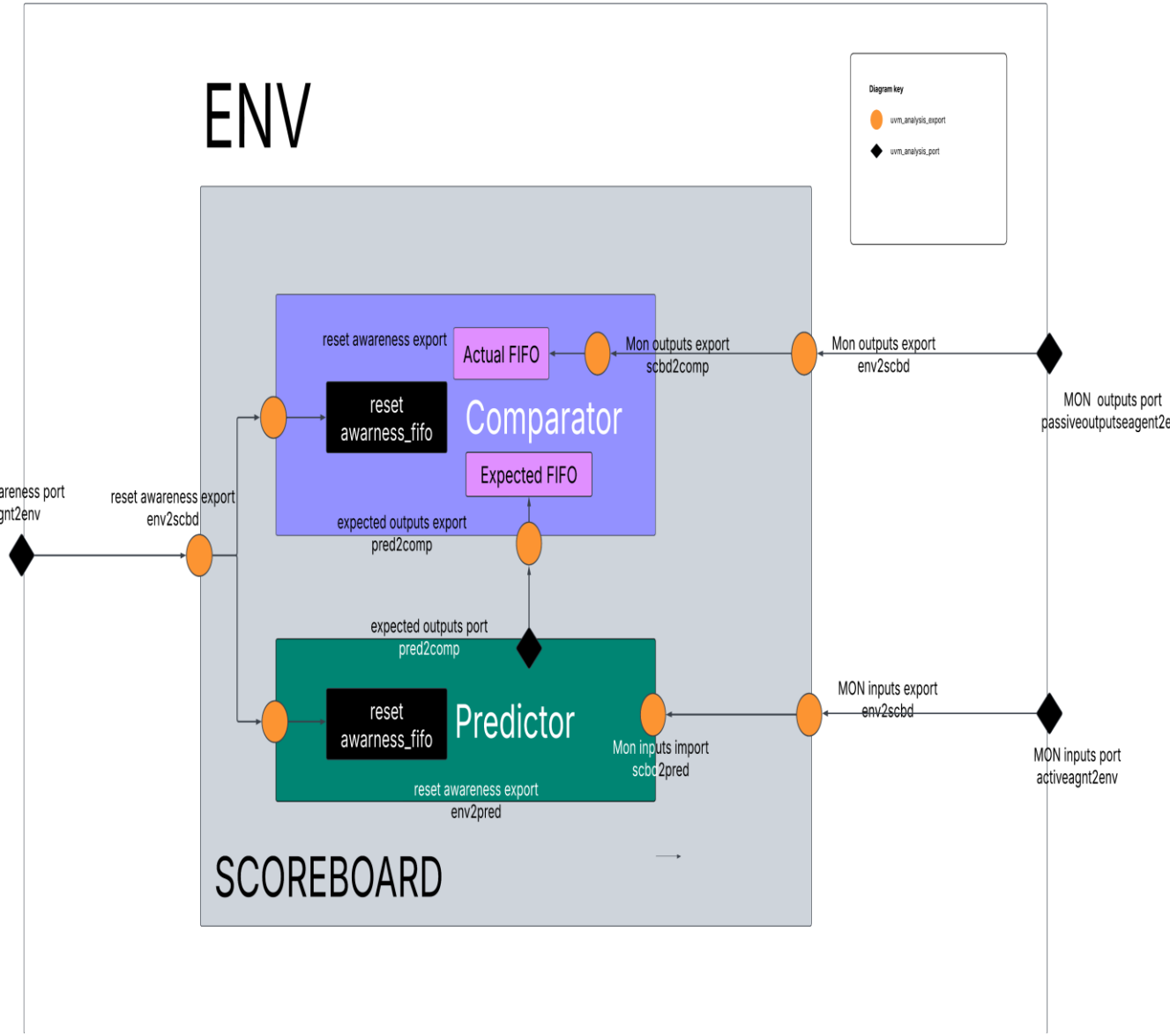
- Note: the sequencer in this agent the **stop_sequences()** function is called within the write function body defined @it, therefore stopping all sequences and calling item_done to all the current running items by the driver.



SCOREBOARD TLM DIAGRAM

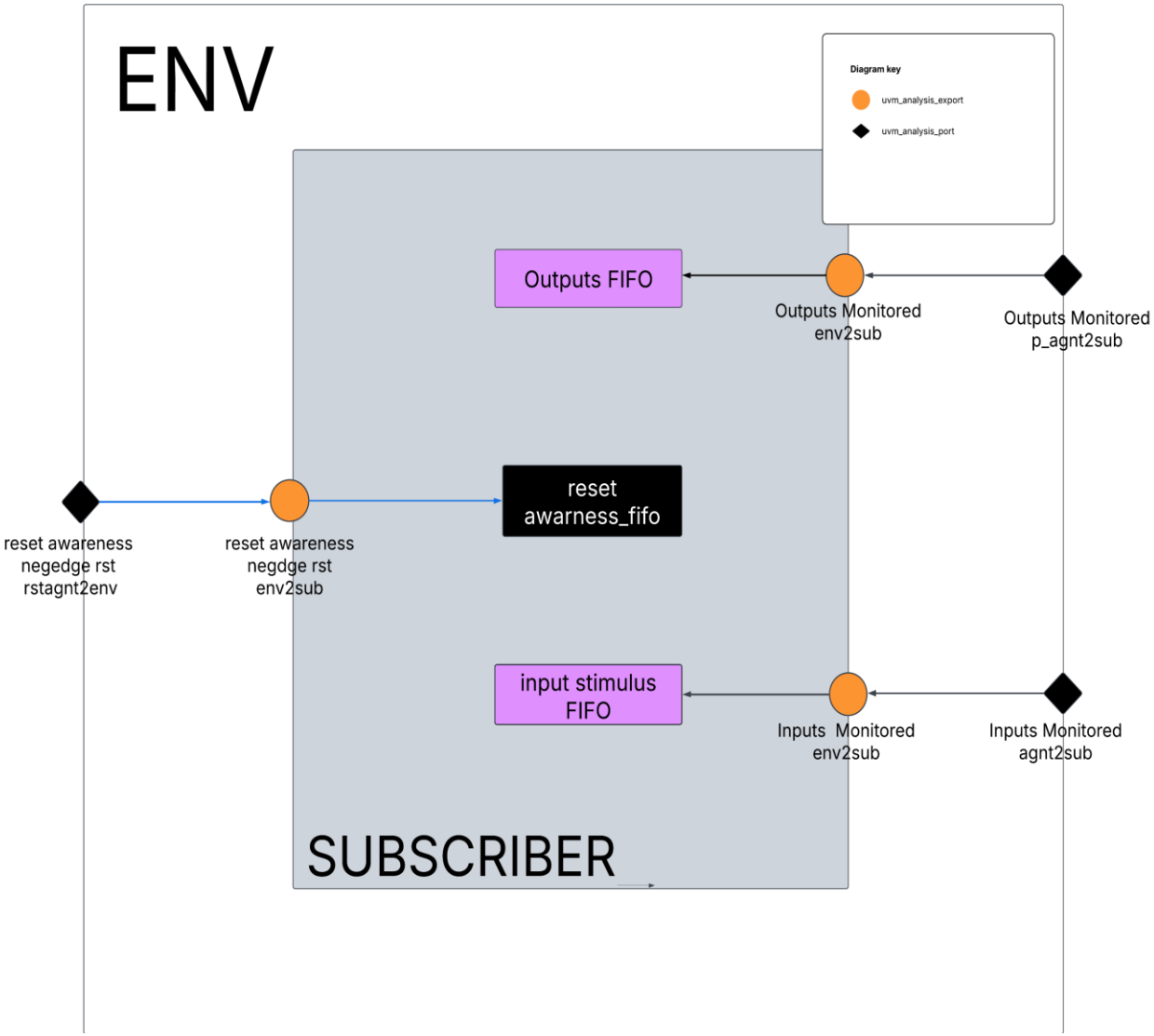
The hierarchy of this scoreboard is inspired from [Clifford Cumming's OVM-UVM Scoreboards-Fundamental Architectures]

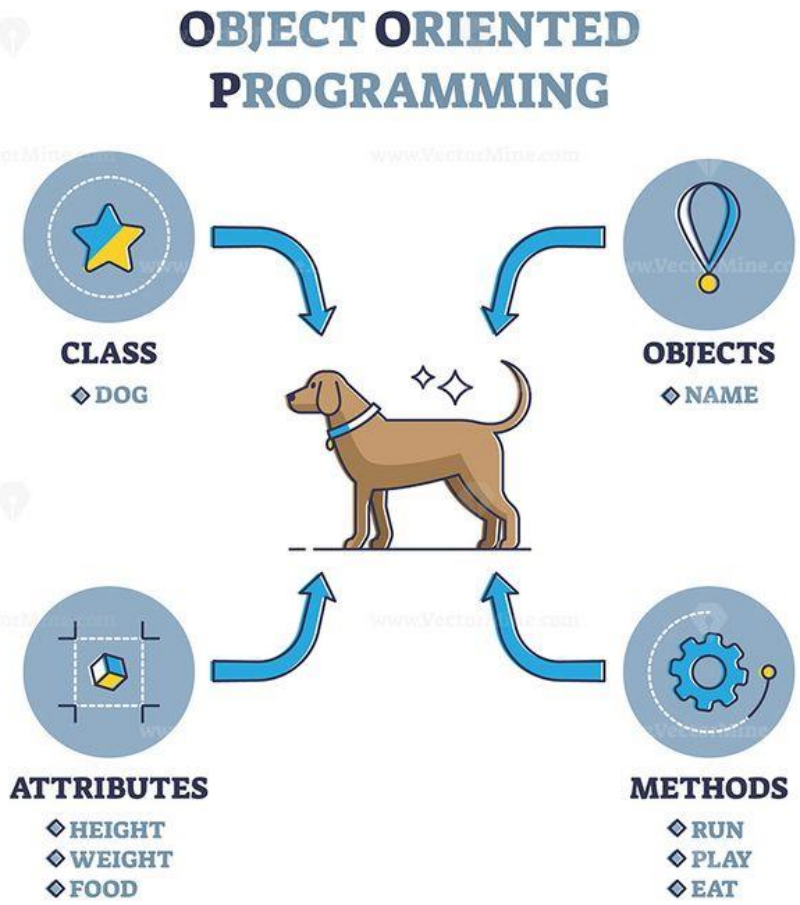
The reset awareness was inspired from The published paper [Handling Asynchronous Reset(s) Testing by building reset-awareness into UVM testbench components by Wei Wei Cheong, Katherine Garden, Ana Sanz Carretero which was previewed @DVCON 2021]



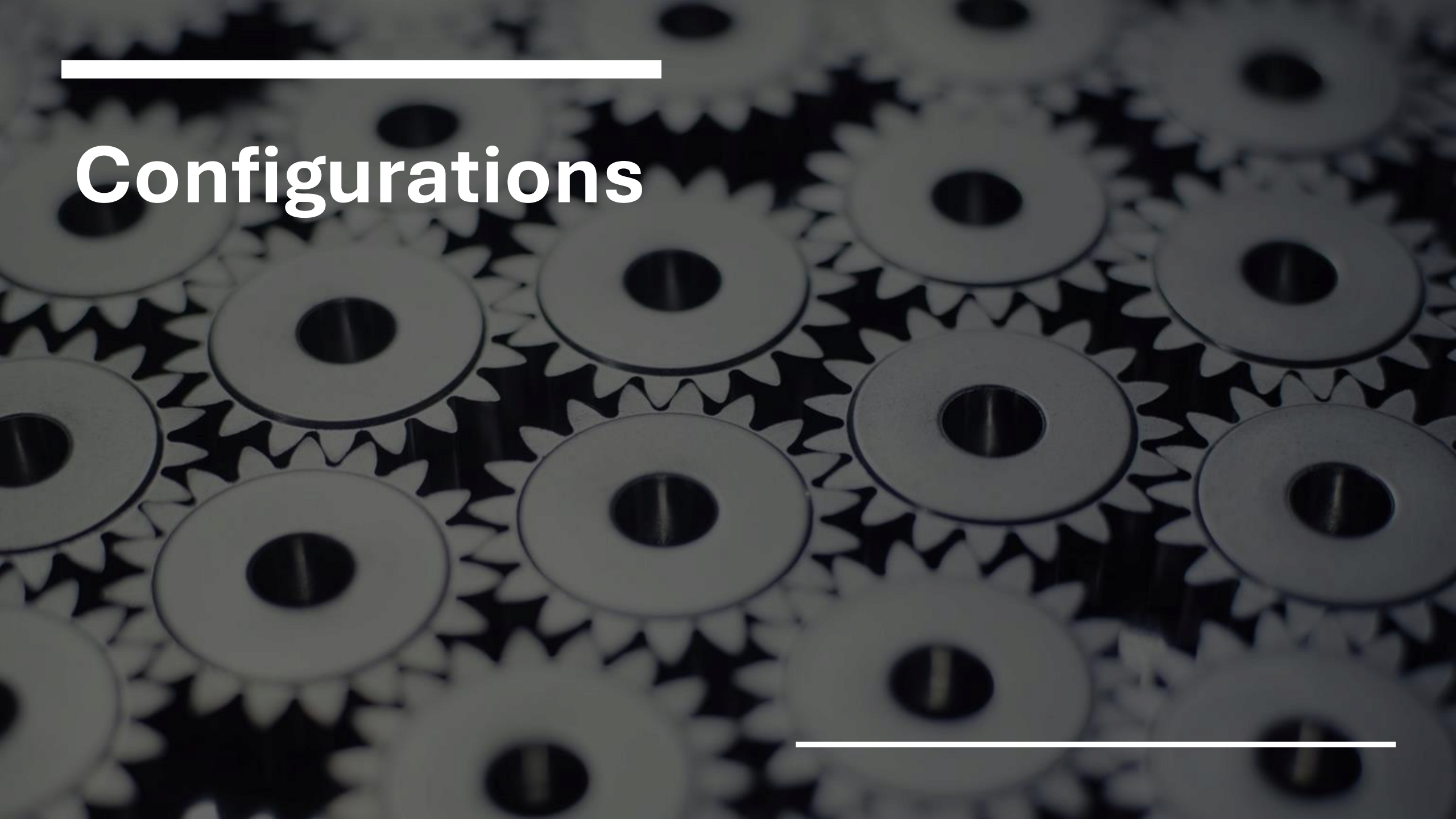
SUBSCRIBER TLM DIAGRAM

The reset awareness was inspired from The published paper The reset awareness was inspired from The published paper [Handling Asynchronous Reset(s) Testing by building reset-awareness into UVM testbench components by Wei Wei Cheong, Katherine Garden, Ana Sanz Carretero which was previewed @DVCON 2021]



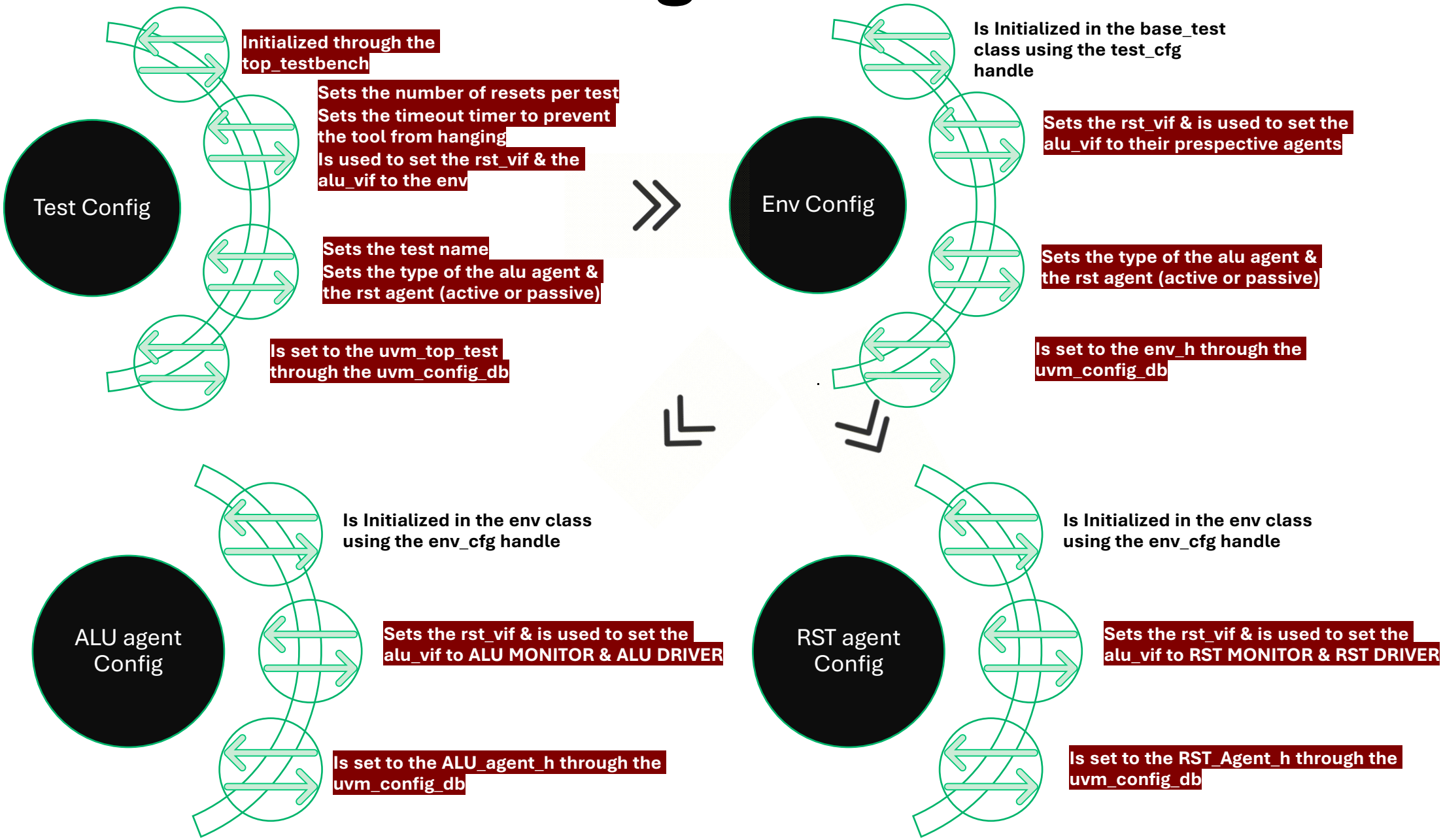


Classes? What do they do?



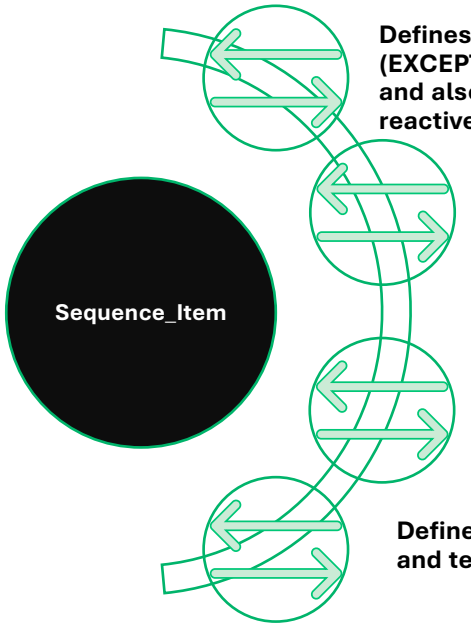
Configurations

Configurations




Sequence
Items

ALU Sequence Item

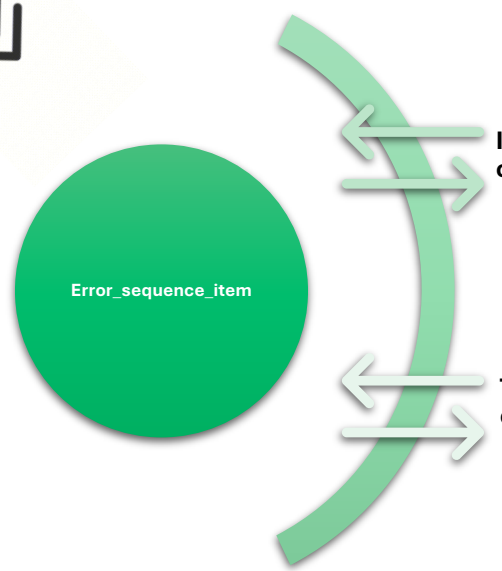


Defines the rand input stimulus (EXCEPT FOR THE RST) of the DUT, and also the output in case of using a reactive agent for responses

Registers all the stimulus within the factory to be compared, copied, cloned, printed, packed or unpacked (overriding each function do_X respectively)

Defines the constraints for the design specifications

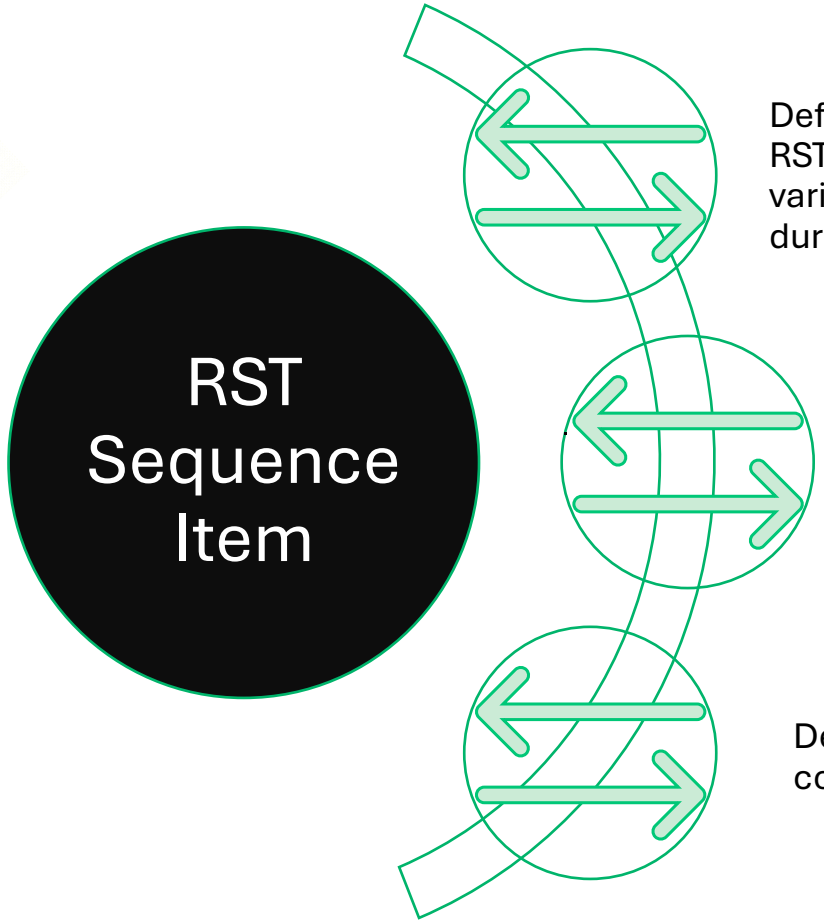
Defines multipile static flags to control coverage and test termination



Inherits the sequence_item and all of its methods, constraints..etc

Turns off the design specification constraints

RST Sequence Item



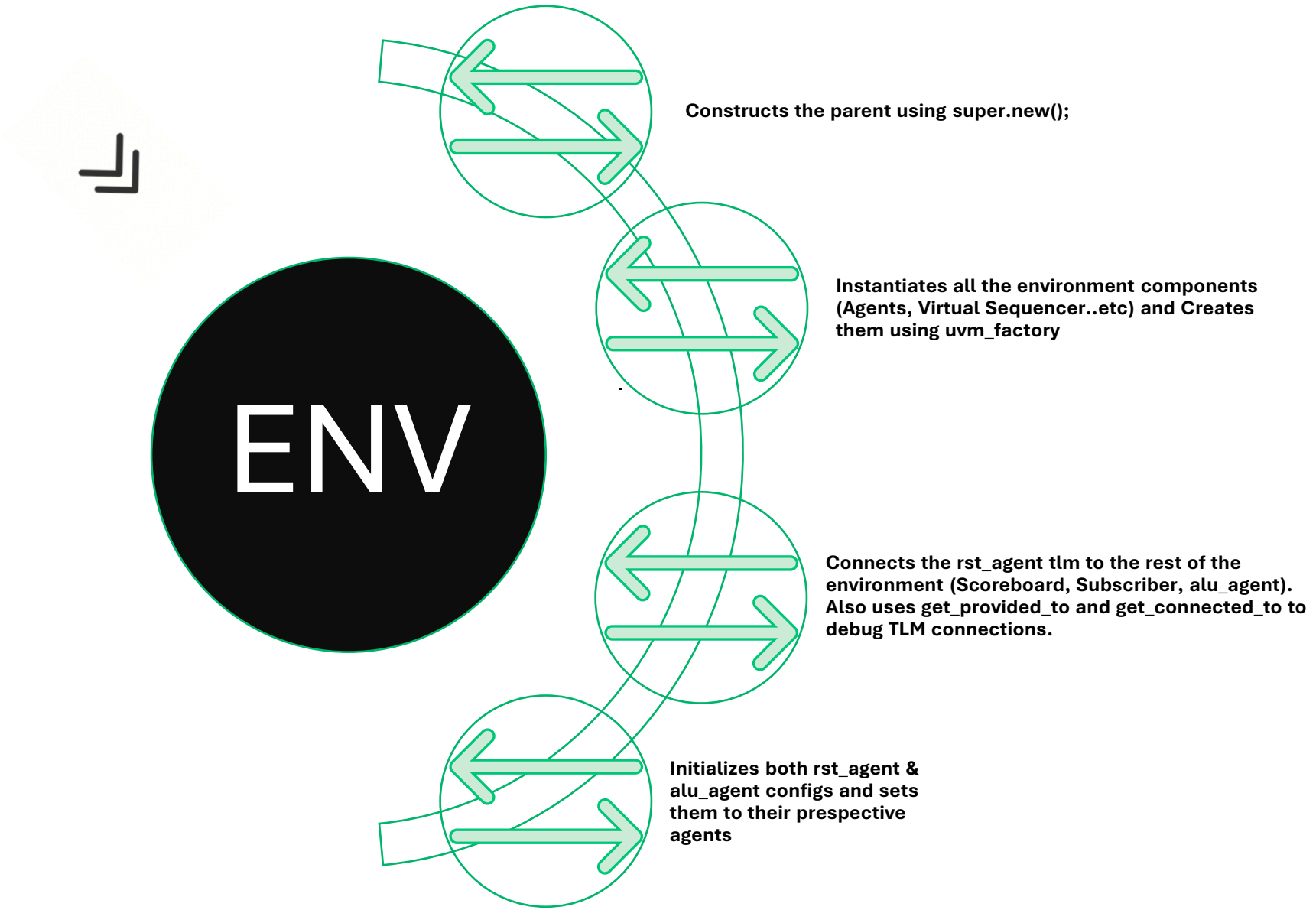
Defines RAND RST stimulus of the DUT, and also RST_DURATION, RST_DELAY...and other static variables to control coverage and rsts assertion during the test

Registers all the stimulus within the factory to be compared, copied, cloned, printed, packed or unpacked (overriding each function do_X respectively)

Defines the constraints for the rst stimulus and control variables/signals

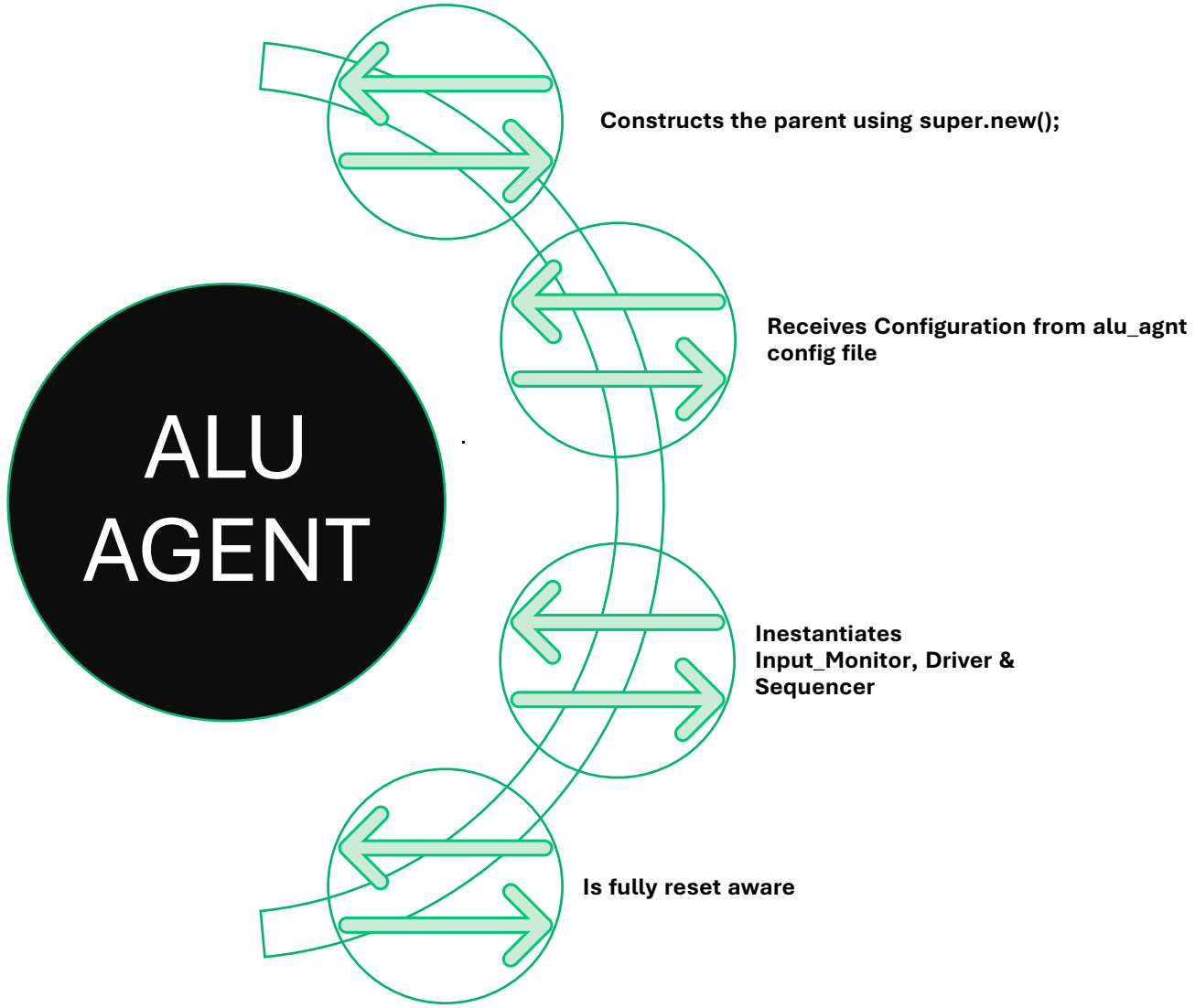
Environment & It's Components

ENVIRONMENT

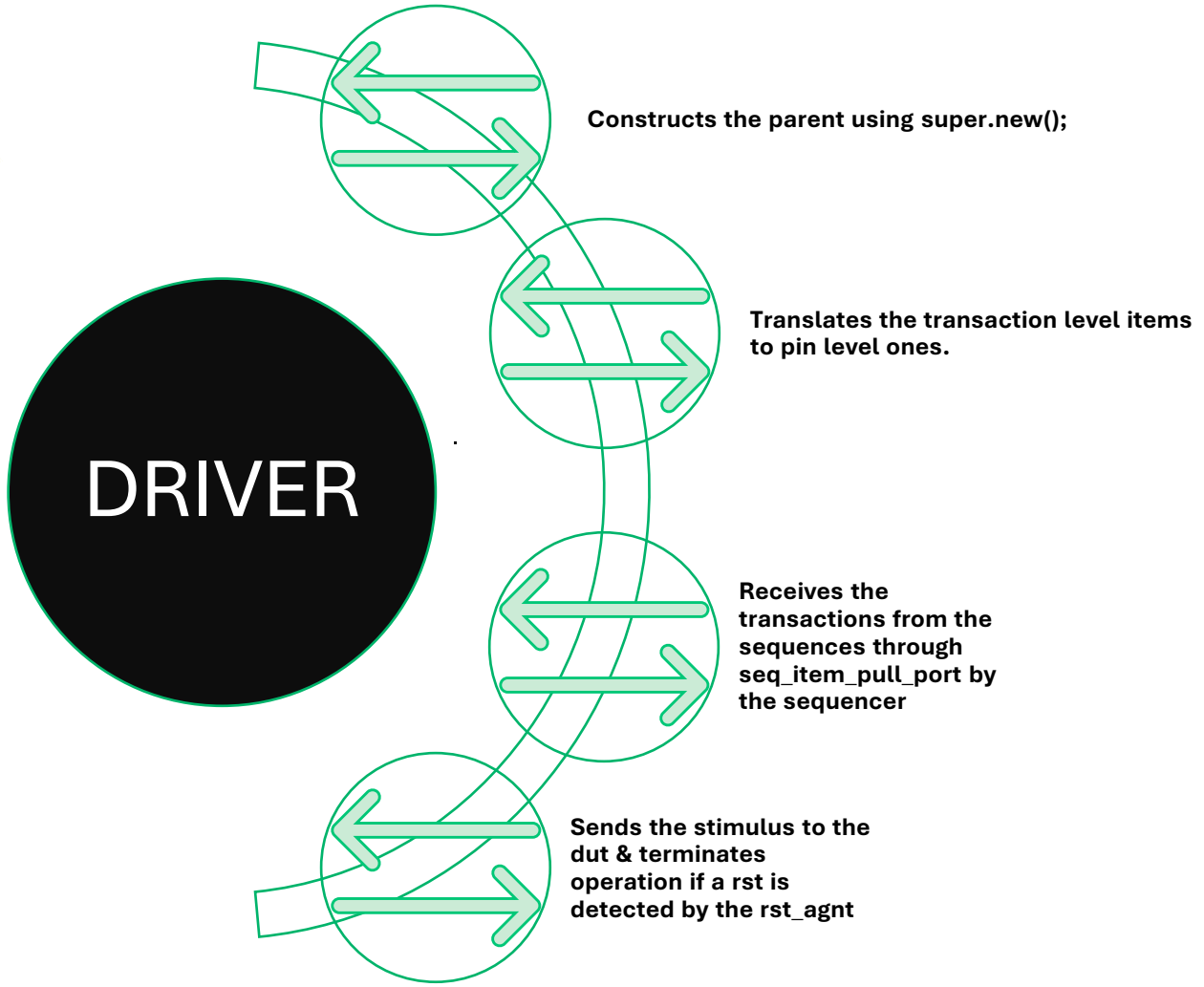


AGENT

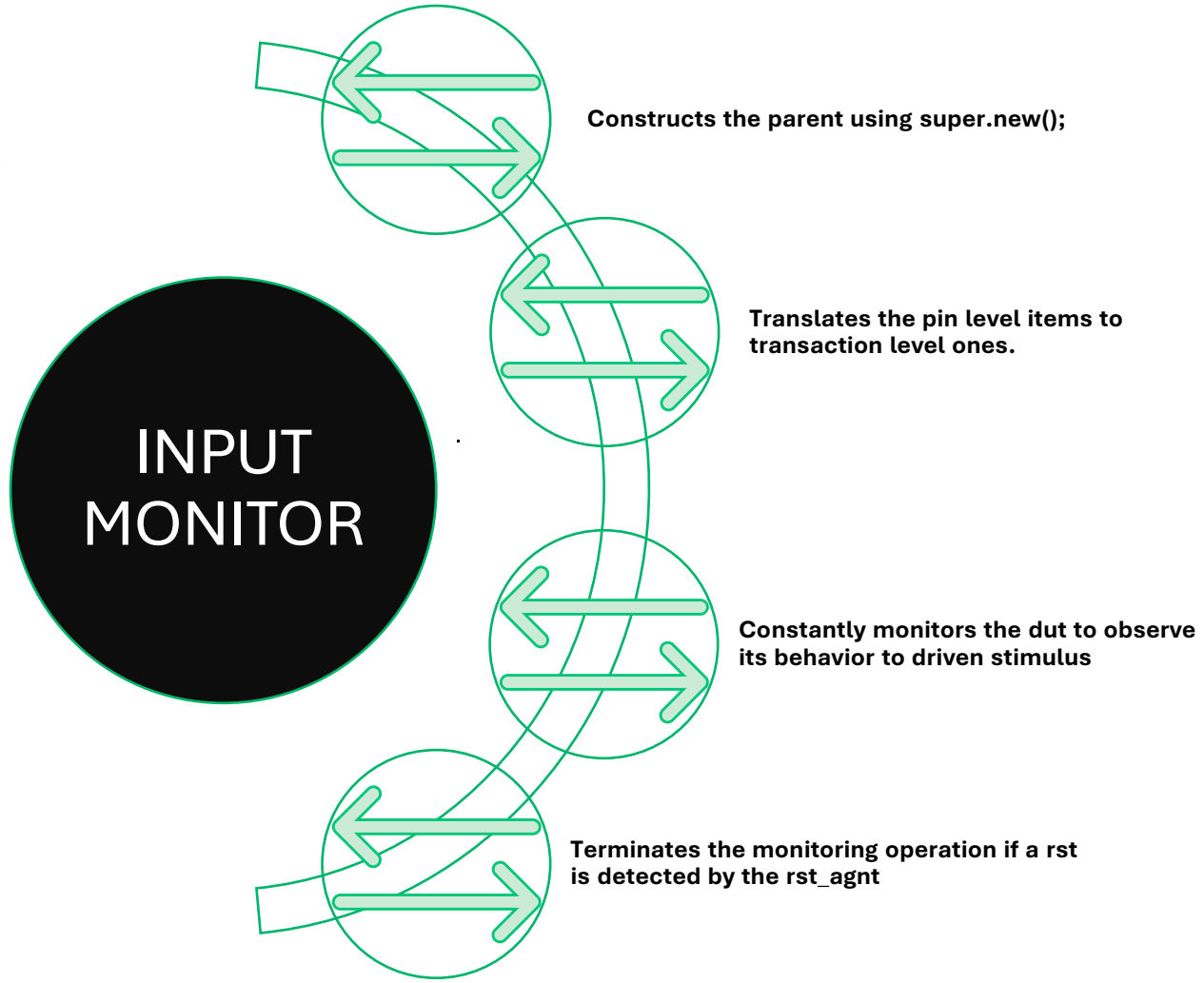
ALU ACTIVE AGENT



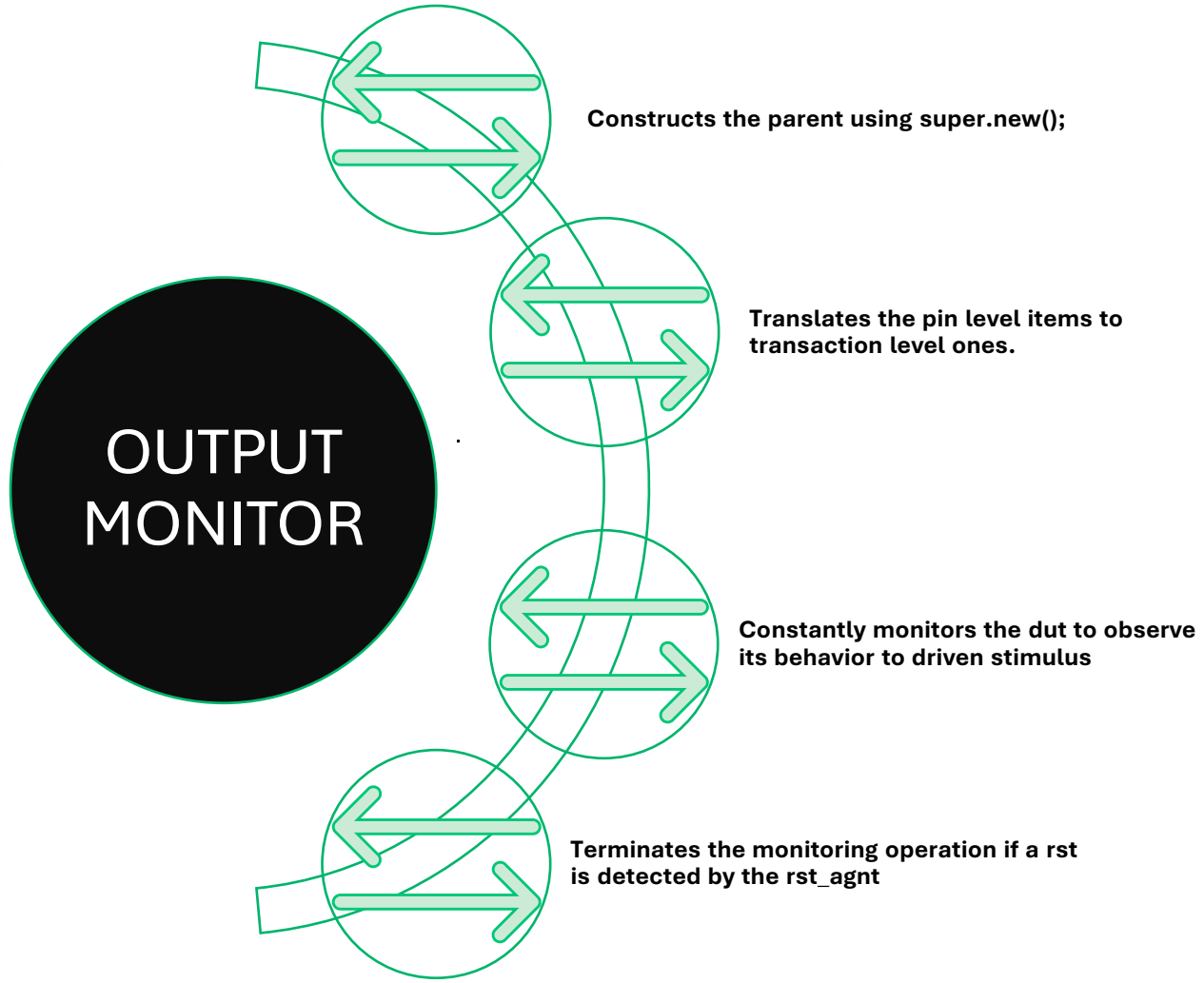
ALU DRIVER


DRIVER
Constructs the parent using super.new();
Translates the transaction level items to pin level ones.
Receives the transactions from the sequences through seq_item_pull_port by the sequencer
Sends the stimulus to the dut & terminates operation if a rst is detected by the rst_agnt

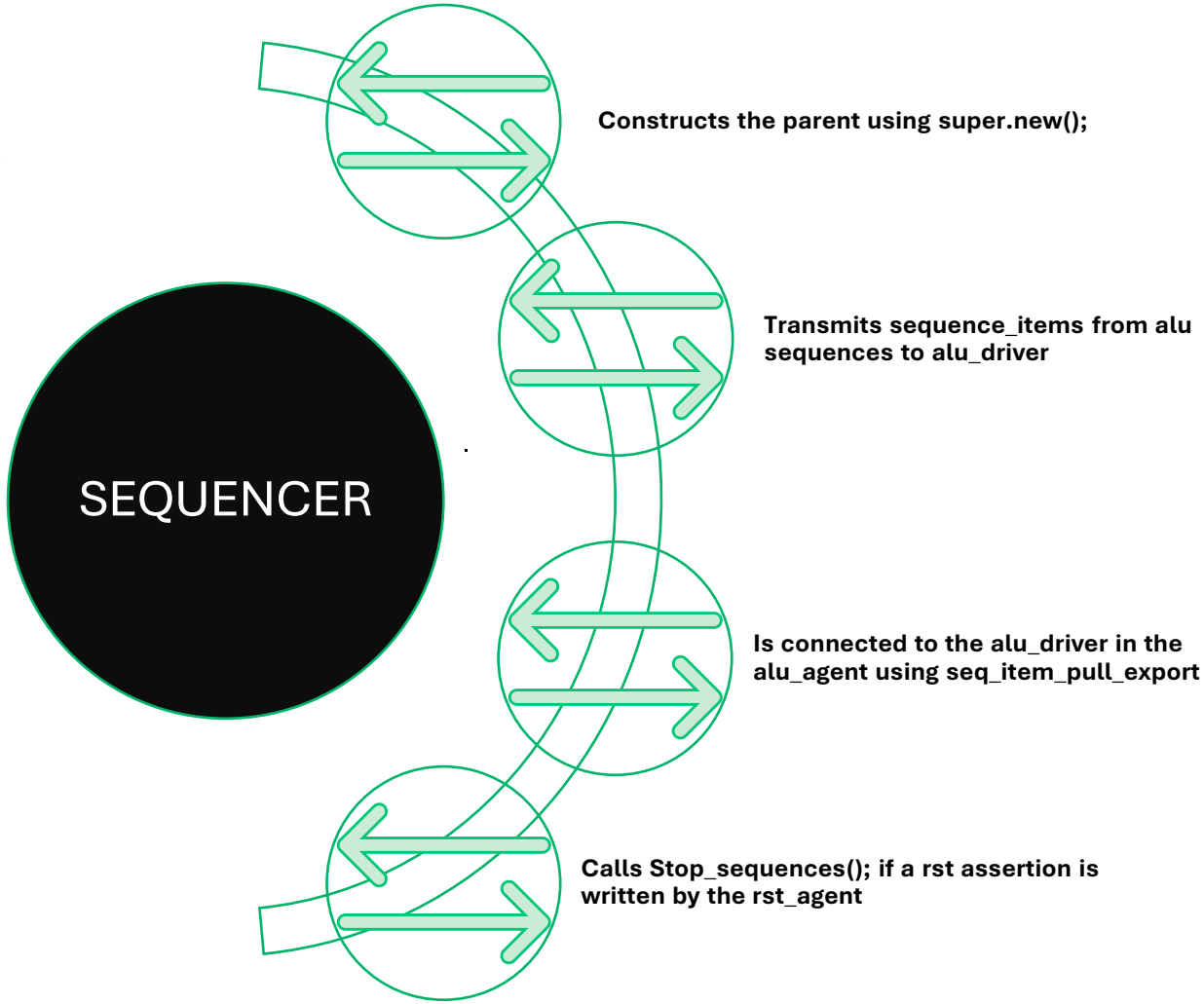
ALU INPUT MONITOR



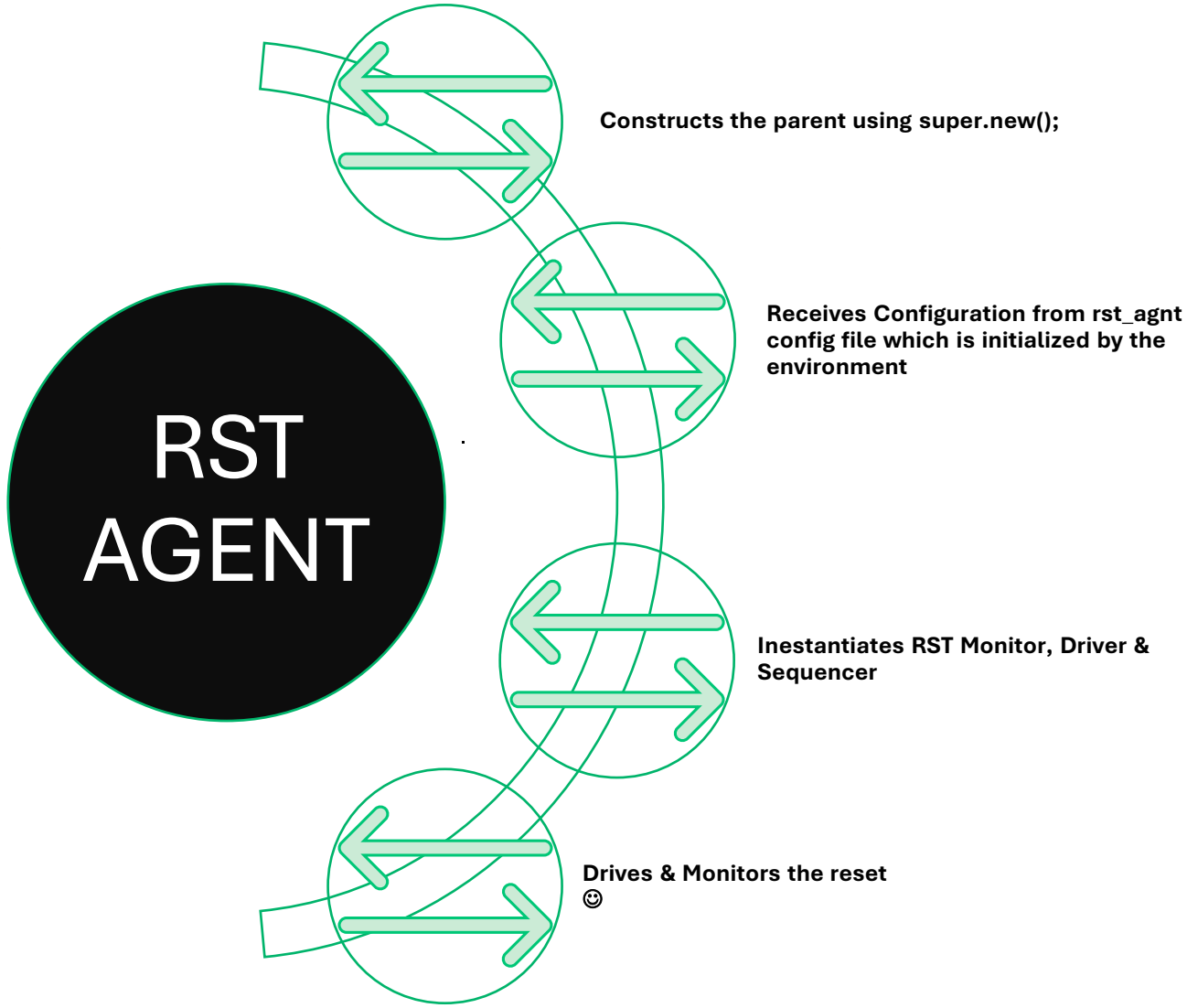
ALU OUTPUT MONITOR



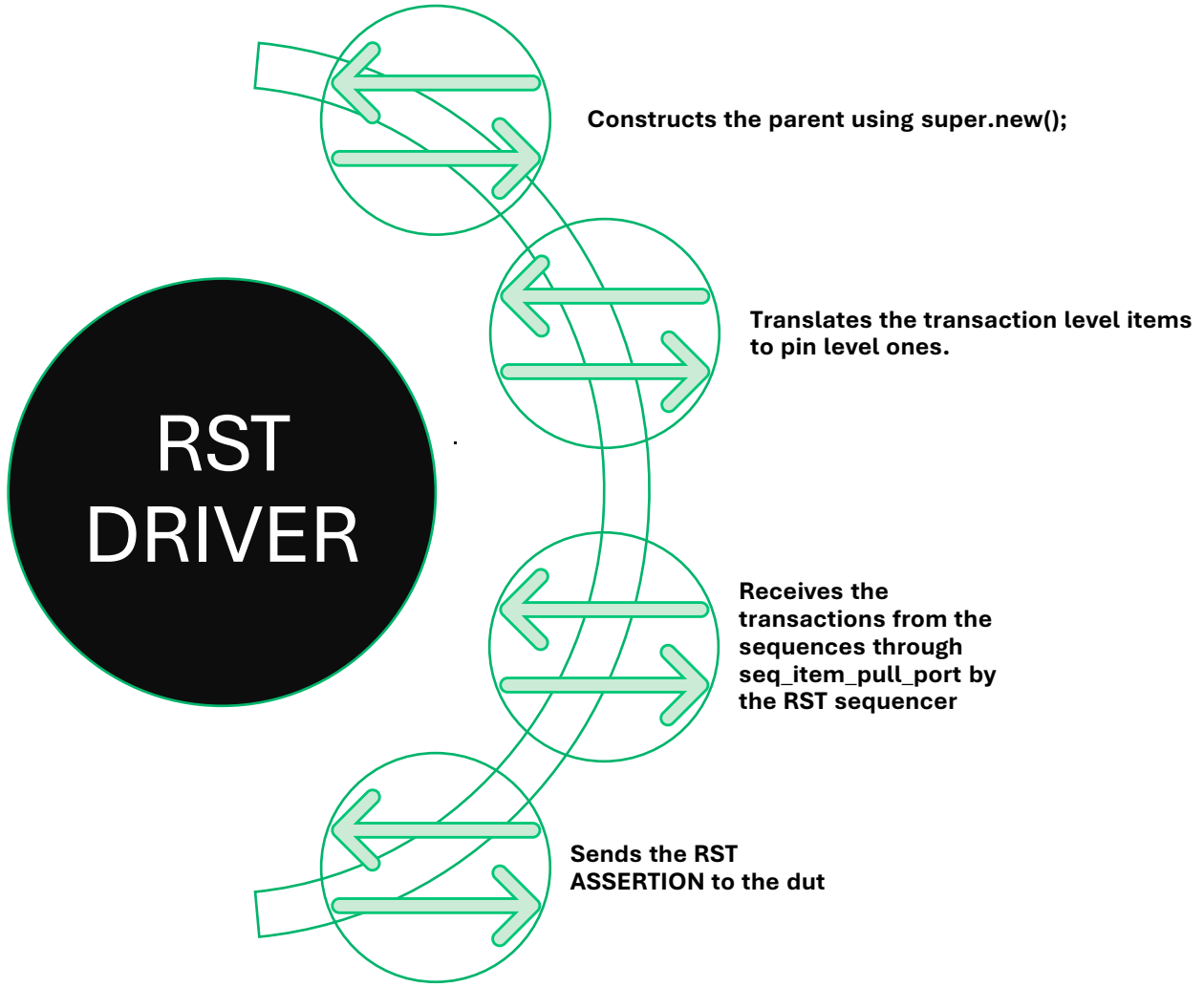
ALU SEQUENCER



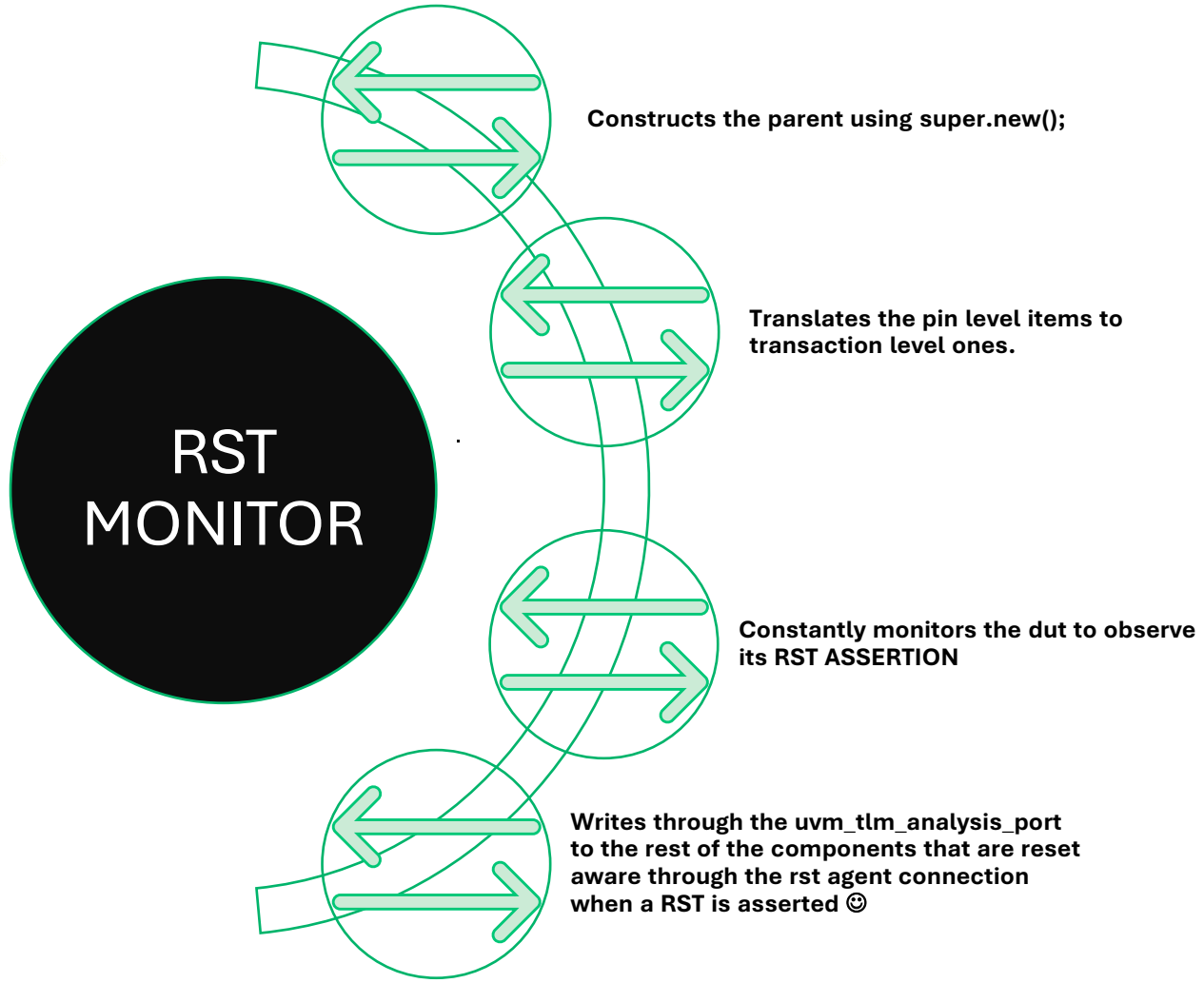
RST ACTIVE AGENT



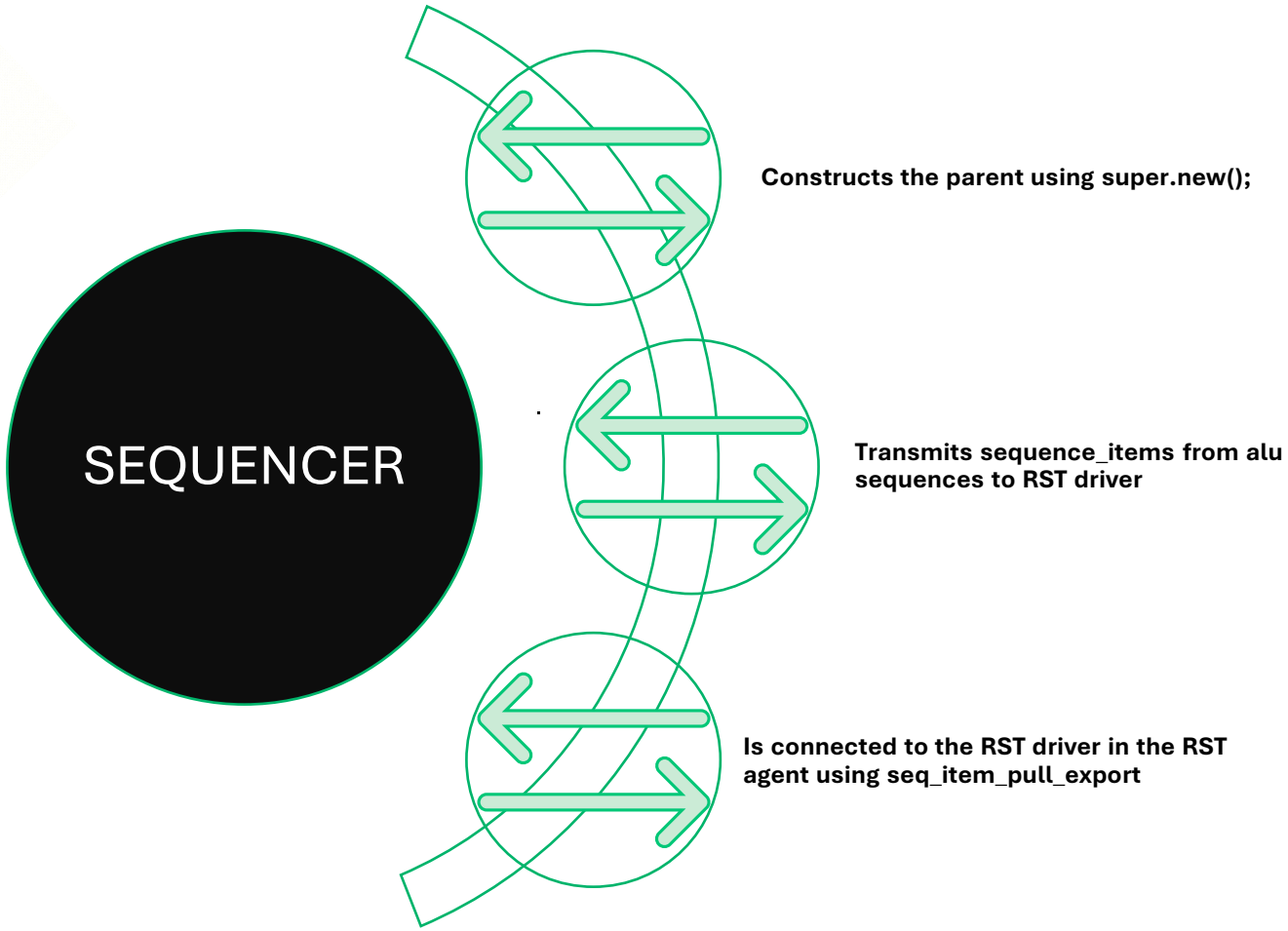
RST DRIVER


RST
DRIVER
Constructs the parent using super.new();
Translates the transaction level items to pin level ones.
Receives the transactions from the sequences through seq_item_pull_port by the RST sequencer
Sends the RST ASSERTION to the dut

RST MONITOR


RST
MONITOR
Constructs the parent using super.new();
Translates the pin level items to transaction level ones.
Constantly monitors the dut to observe its RST ASSERTION
Writes through the uvm_tlm_analysis_port to the rest of the components that are reset aware through the rst agent connection when a RST is asserted ☺

RST SEQUENCER



Constructs the parent using super.new();

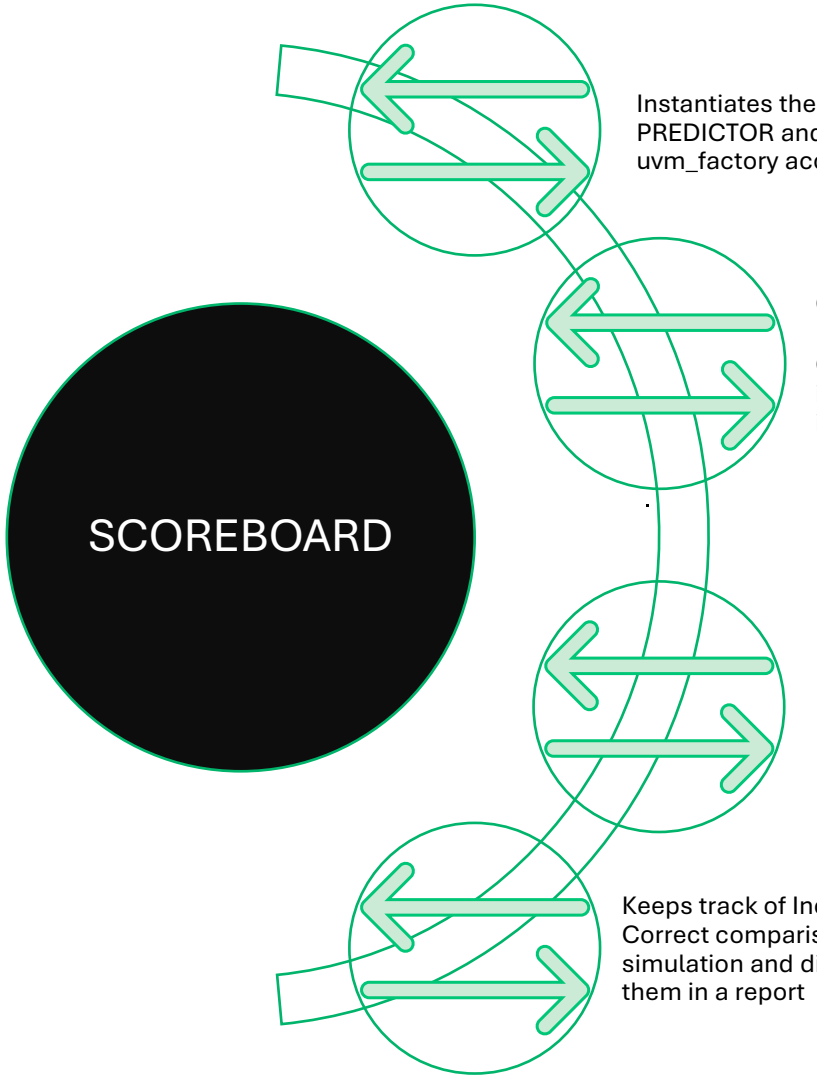
Transmits sequence_items from alu sequences to RST driver

Is connected to the RST driver in the RST agent using seq_item_pull_export

SCOREBOARD


PHILADELPHIA
OTTAWA
LEIPZIG
DUESSELDORF
GENF
LINZ
BREMEN
NUERNBERG
HANNOVER
BRUESSEL
OSNABR

SCOREBOARD



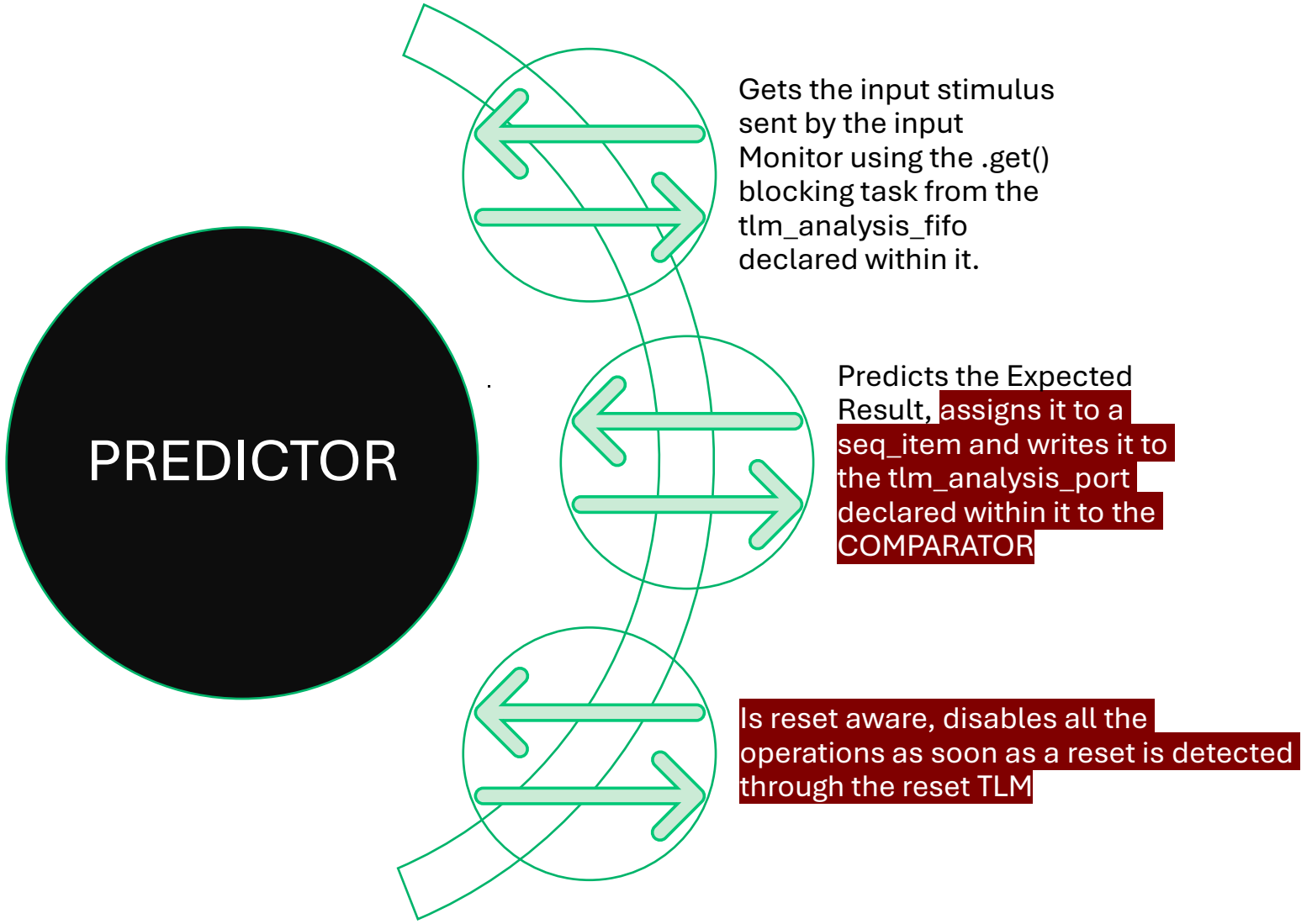
Instantiates the COMPARATOR & PREDICTOR and creates them using uvm_factory accordingly

Connects the EXPORTS coming from the environment to the EXPORTS inside the COMPARATOR & the PREDICTOR
Connects the EXPORT Inside the Comparator to the PORT inside the predictor to get the PREDICTED ITEM and compare it to the ACTUAL SAMPLED ITEM

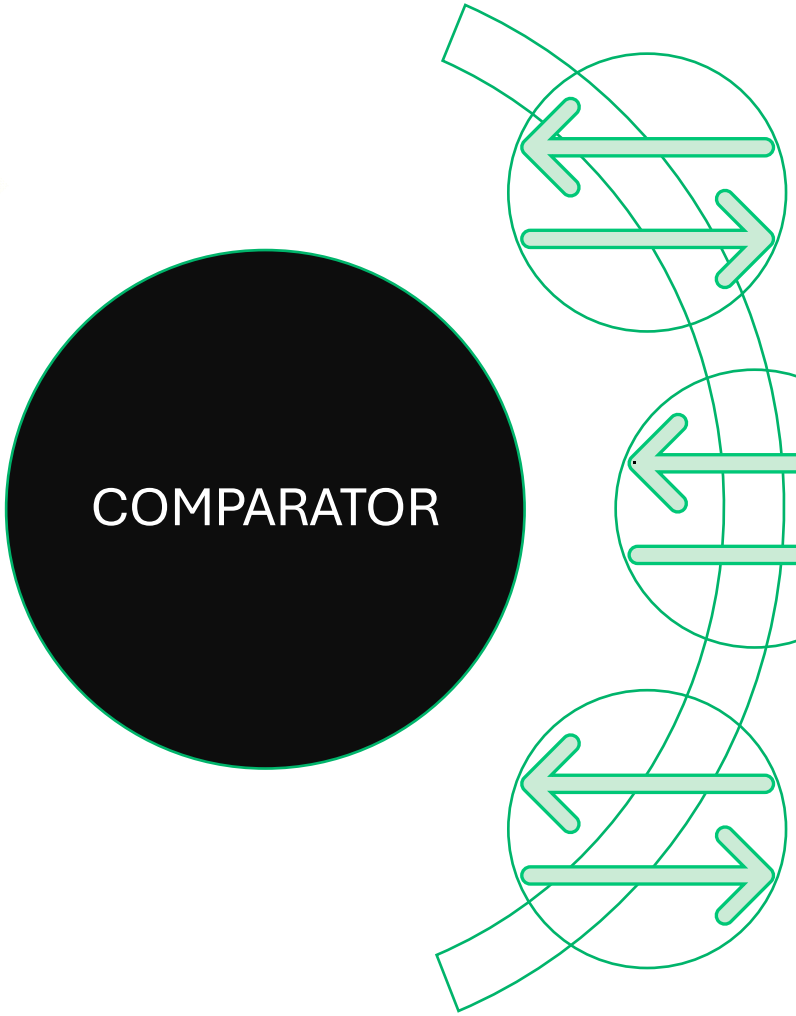
Halts the test using TEST TERMINATION TECHNIQUE used in Clifford Cummings' OVM & UVM TECHNIQUES FOR TERMINATING TESTS. Raising the objection using the Phase_ready_to_end_phase

Keeps track of Incorrect & Correct comparisons during simulation and displays them in a report

PREDICTOR


PREDICTOR
Gets the input stimulus sent by the input Monitor using the .get() blocking task from the tlm_analysis_fifo declared within it.
Predicts the Expected Result, assigns it to a seq_item and writes it to the tlm_analysis_port declared within it to the COMPARATOR
Is reset aware, disables all the operations as soon as a reset is detected through the reset TLM

COMPARATOR



Gets the output that was sampled by the output monitor through the uvm_tlm_analysis_fifo using .get() blocking task

Compares the sampled outputs to the expected seq_item sent by the predictor

Is reset aware, disables all the operations as soon as a reset is detected through the reset TLM

SUBSCRIBER

SUBSCRIBER

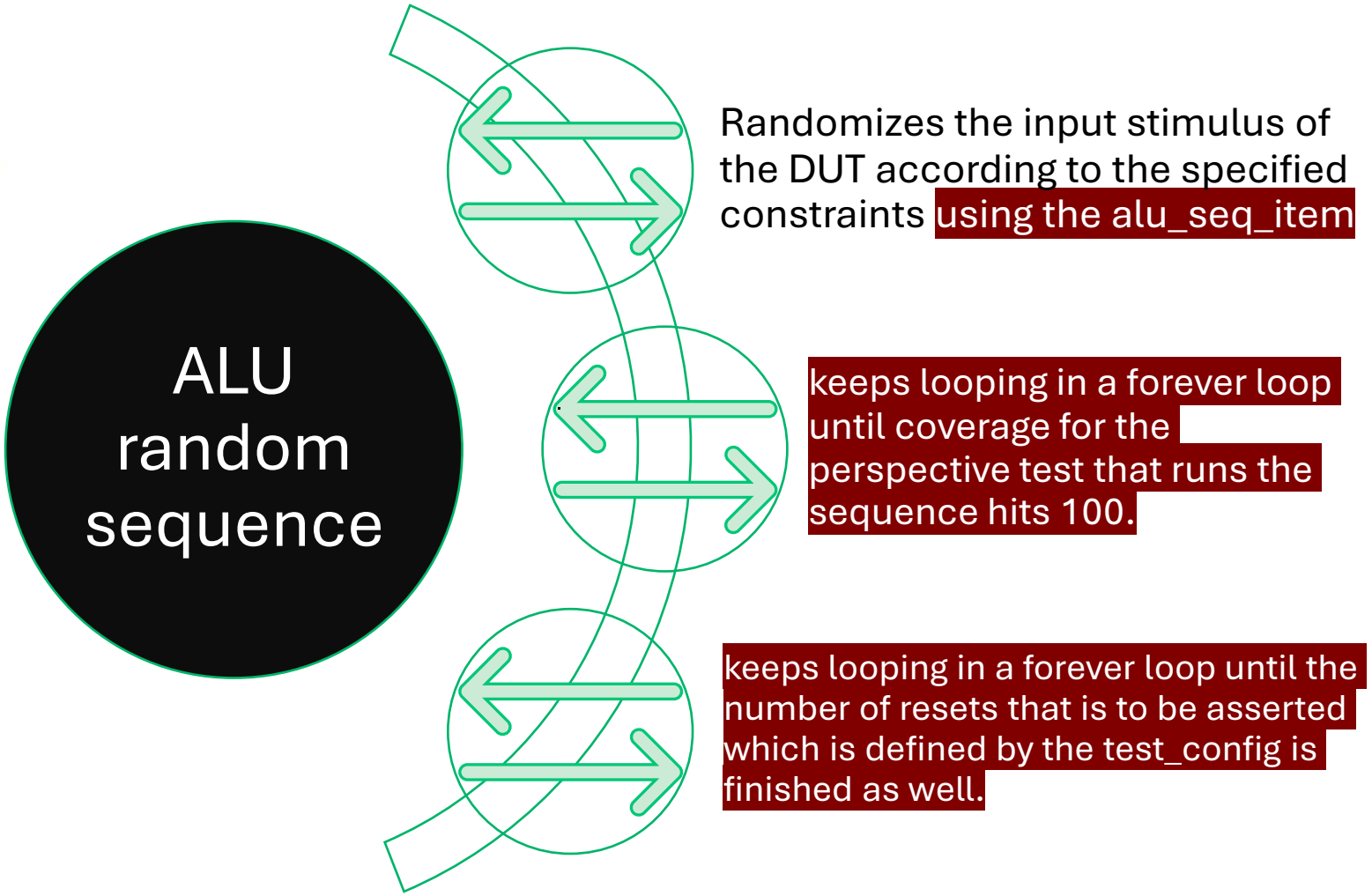
Receives input stimulus & outputs monitored by the monitor using fork_joined .get() task to each analysis FIFOS which are declared for INPUT STIMULUS OUTPUTS MONITORED RESPECTIVELY and samples each time the write function is called by the uvm_tlm_analysis_port inside the monitors.

When the parametrized coverage group of each test reaches 100% coverage, it asserts cov_target accordingly and therefore the test ends if all the resets are finished as well.

Is reset aware therefore it is disabled when a reset assertion happens.

SEQUENCES

ALU random sequence

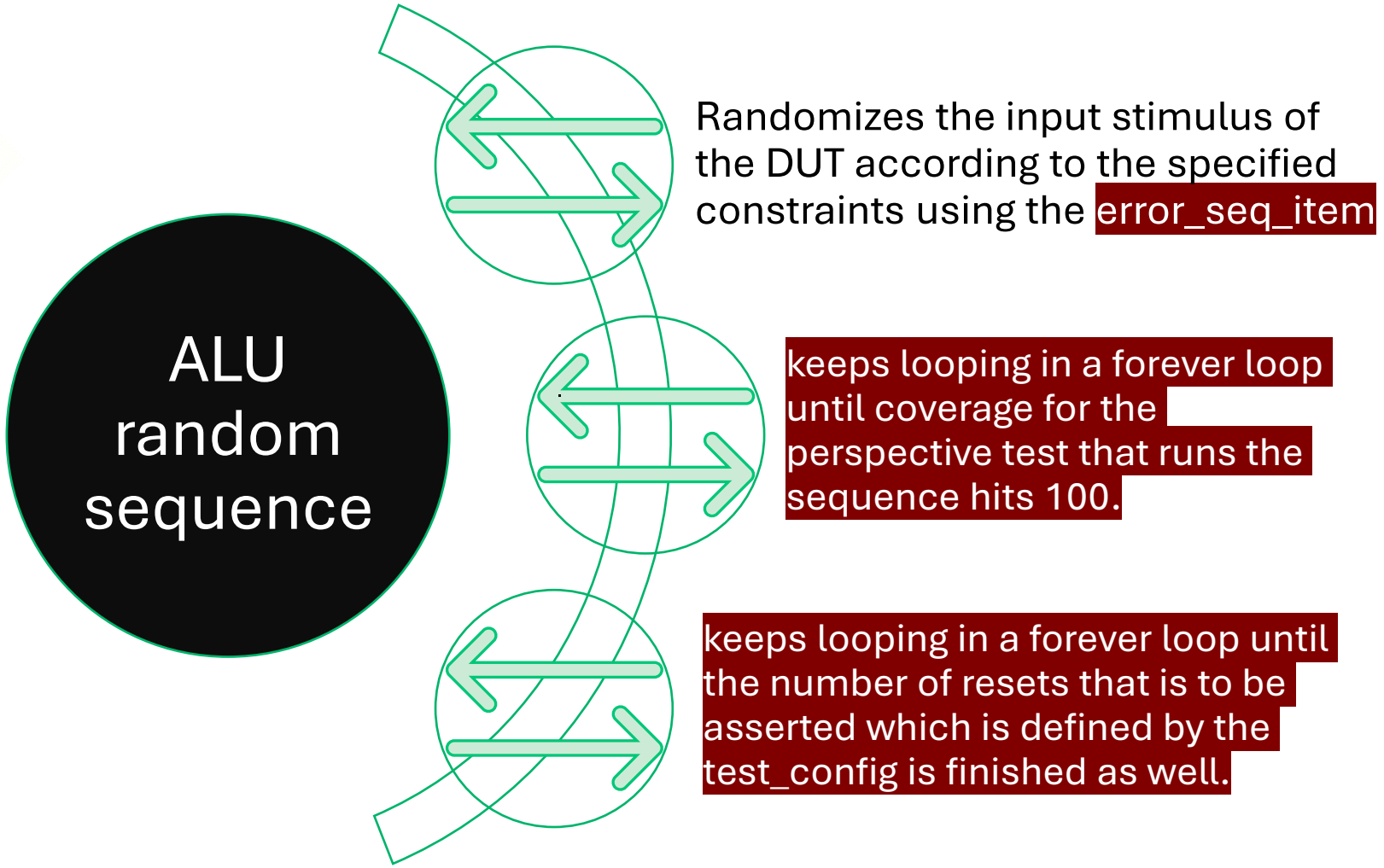


Randomizes the input stimulus of the DUT according to the specified constraints using the alu_seq_item

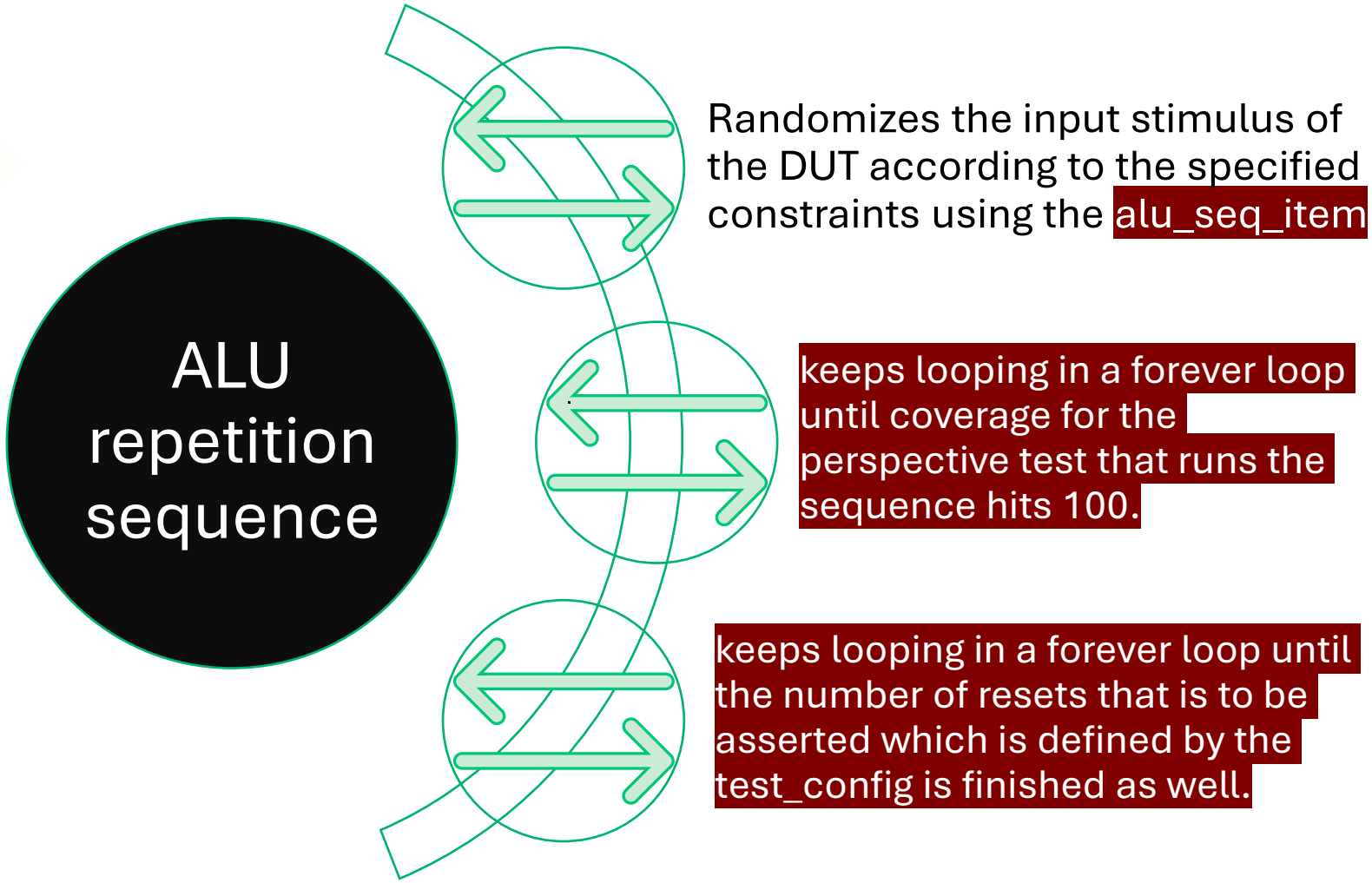
keeps looping in a forever loop until coverage for the perspective test that runs the sequence hits 100.

keeps looping in a forever loop until the number of resets that is to be asserted which is defined by the test_config is finished as well.

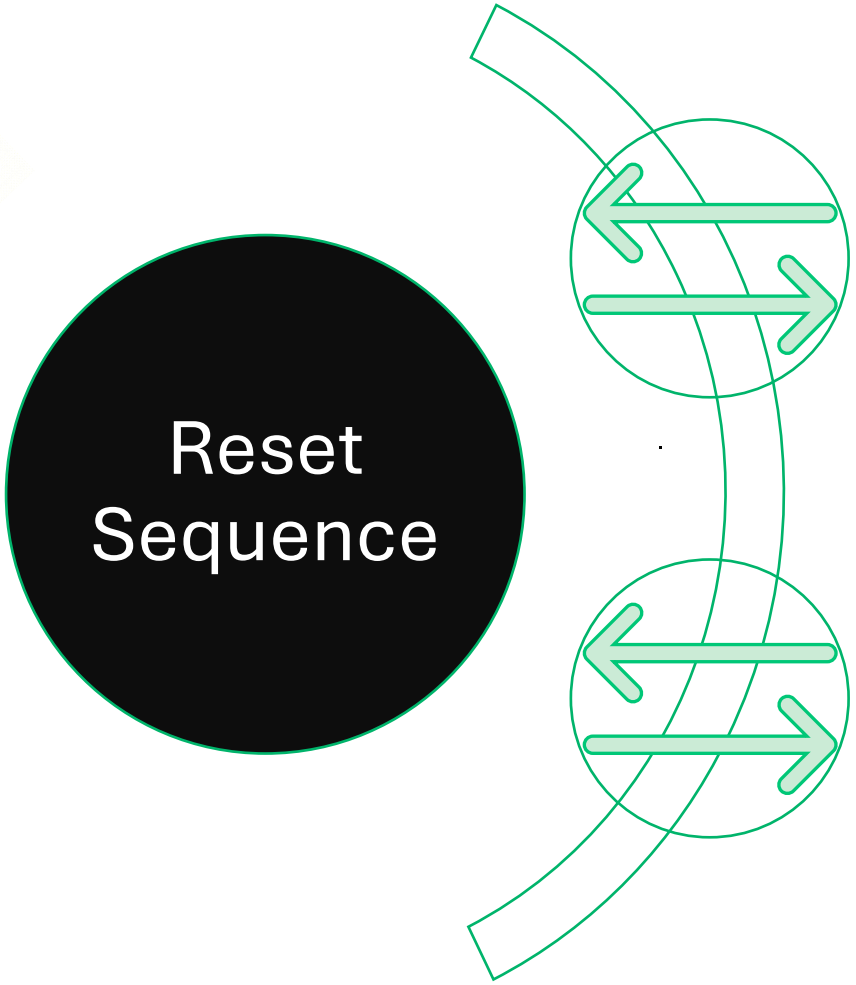
ALU Error sequence



ALU repetition sequence



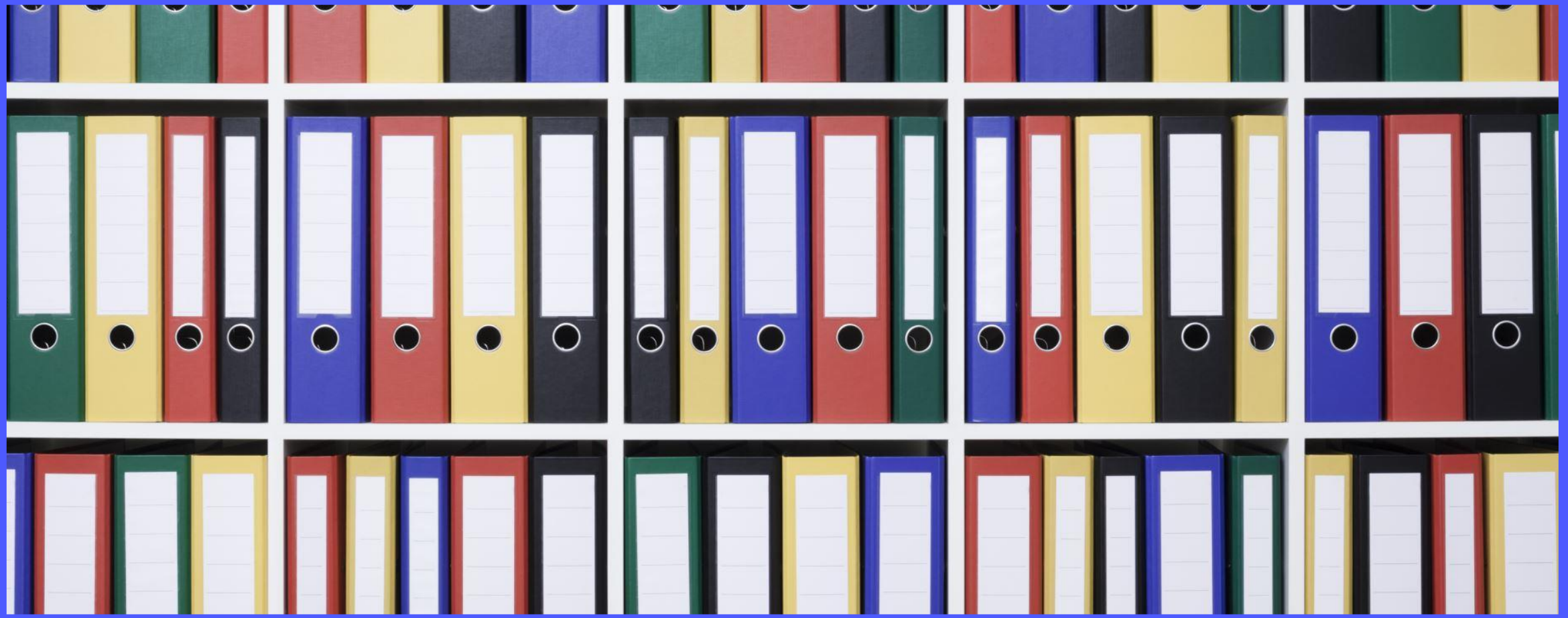
Reset Sequence



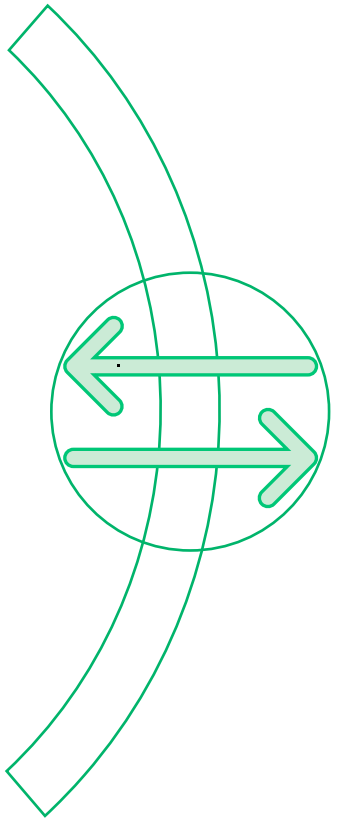
Randomizes the rst_seq_item properties (rst_duration, rst_delay, rst_on..etc)

keeps looping in a forever loop until the number of resets that is to be asserted which is defined by the test_config is finished as well.

Virtual Sequencer

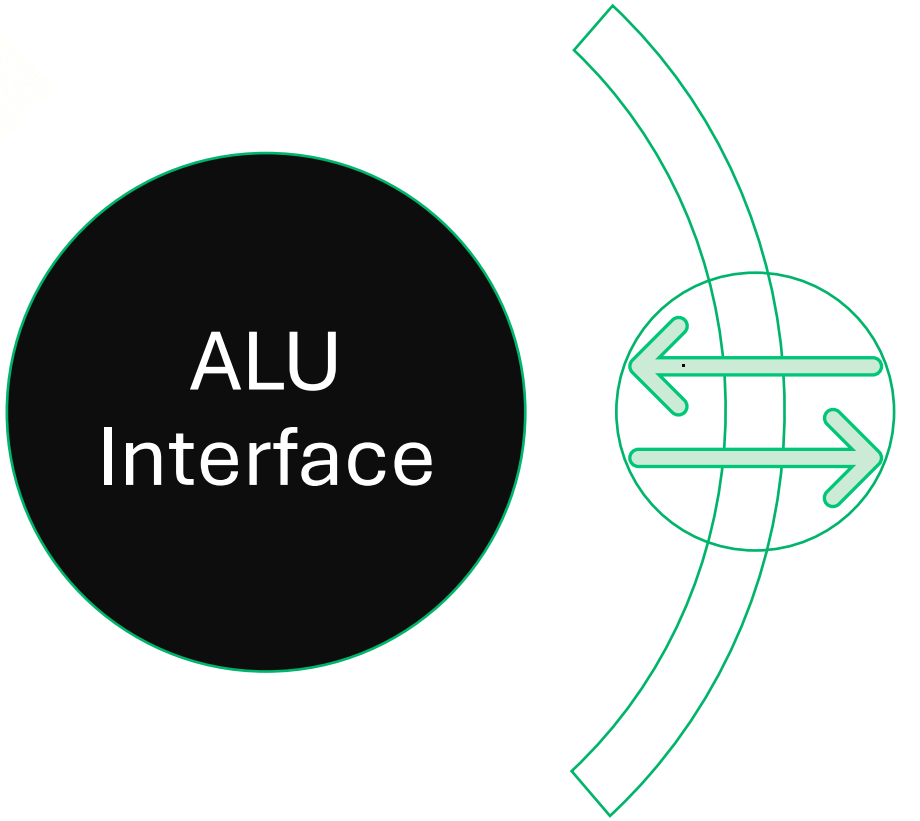


Virtual Sequencer



Instantiates the alu_sequencer & the rst_sequencer so they can be used later to point to the sequencers of the same type inside the virtual sequence and start sequences respectively on them

ALU Interface



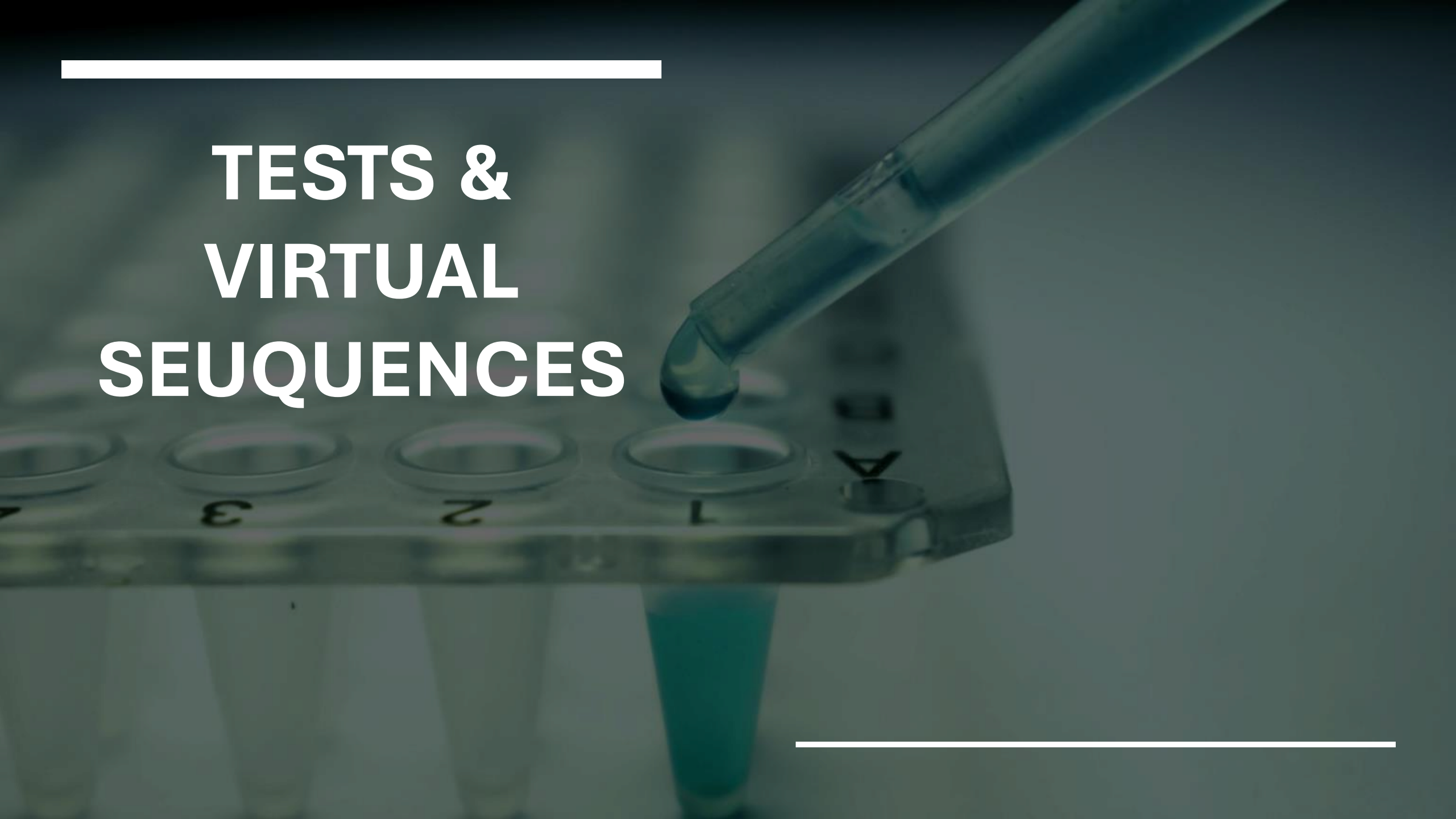
ALU Interface has all of the input signals of the DUT except the rst

RST Interface

RST Interface has rst_n

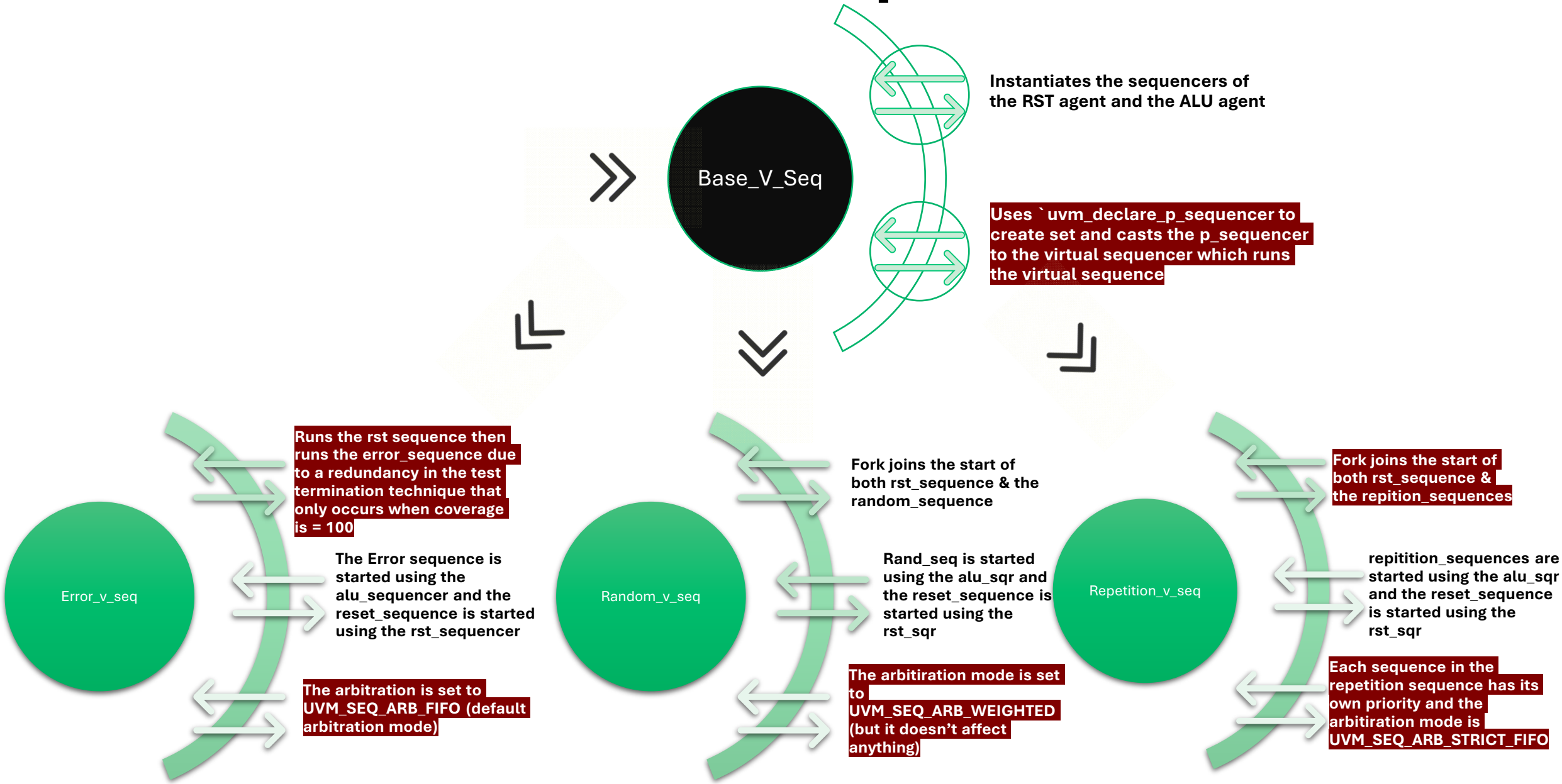
Also has the event which is triggered to start the rst assertion from the top_testbench

Receives the rst_delay & rst_duration from the verification environment and asserts the rst according to those values

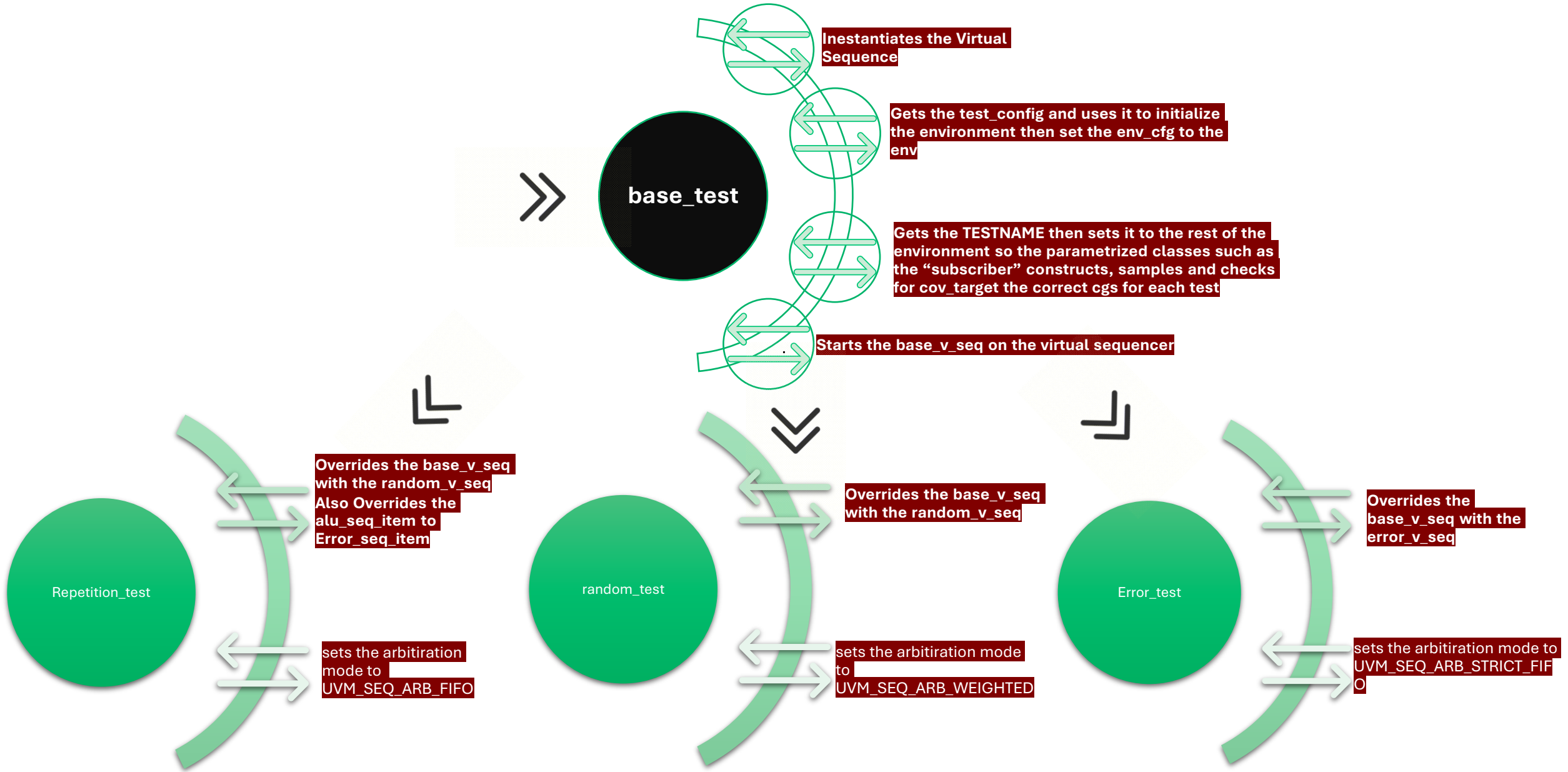


TESTS & VIRTUAL SEUQUENCES

Virtual Sequences



Test & Test Base



Reset Awareness & Asynchronous reset

Asynchronous RST

- To assert a true asynchronous reset one would need to do it through the top_testbench @random times, which is exactly what the RST Agent does, it asserts the reset @Randomized times Asynchronously by triggering an event in the RST Interface which in turn starts a reset assertion in the top_testbench.
- The **resets are spaced using @Reset_DELAY** which is also randomized and constrained.
- The **reset_DURATION** is also randomized and constrained.
- They are **both passed to the interface task and then to the top_testbench.**

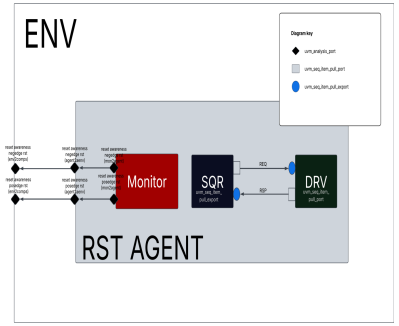
```
  //-------------------------------------
  // Connect phase
  //-------------------------------------
  function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);

    // Connect alu agent to env TLM
    alu_agent_h.inputMon2agent.connect(alu_in_env_ap);
    alu_agent_h.outputMon2agent.connect(alu_out_env_ap);

    // Connect env to scoreboard
    alu_in_env_ap.connect(scb_h.analysis_export_inputs);
    alu_out_env_ap.connect(scb_h.analysis_export_actual_outputs);

    // Connect env to subscriber
    alu_in_env_ap.connect(sub_h.analysis_export_inputs);
    alu_out_env_ap.connect(sub_h.analysis_export_outputs);

/*****************************************************************************************************/

    // Connect reset agent to env
    rst_agent_h.reset_collected_port_n.connect(reset_collected_port_n); // reset_agent_to_env
    rst_agent_h.reset_collected_port_p.connect(reset_collected_port_p); // reset_agent_to_env

    // Connect env to alu agent
    reset_collected_port_n.connect(alu_agent_h.reset_collected_export_n);
    reset_collected_port_p.connect(alu_agent_h.reset_collected_export_p);

    // Connect reset to alu agent
    reset_collected_port_n.connect(scb_h.reset_collected_export); // scoreboard we may send posedge rst later
    reset_collected_port_n.connect(sub_h.reset_collected_export); // subscriber we may send posedge rst later

/*****************************************************************************************************/
    // sequencers connection
    v_seqr.seqr_ALU = alu_agent_h.sequencer_h;
    v_seqr.seqr_RST = rst_agent_h.sequencer_h;
```

Reset Awareness

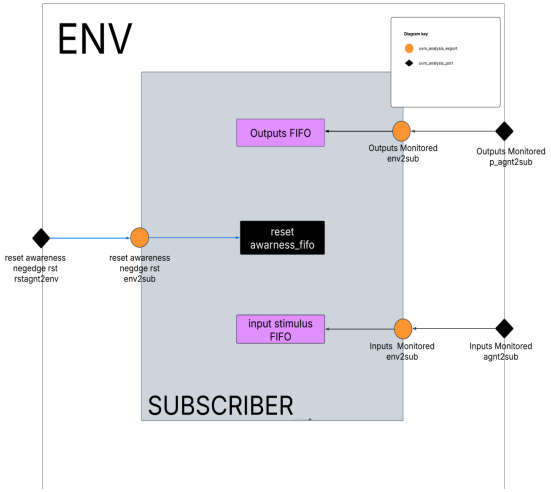
- So, the **driver calls the interface task**, but what does the monitor do?
- **The monitor permanently observes the state of the reset through the virtual interface and as soon as it is asserted either high or low.**

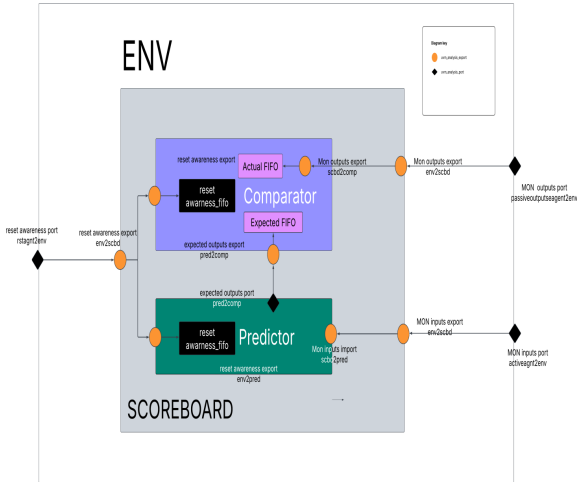
it notifies all the other components within the Environment through UVM_TLM_ANALYSIS_PORTS passing through the RST Agent, Env to UVM_TLM_ANALYSIS_EXPORTS in the the ALU Agent and its components, Subscriber & Scoreboard.

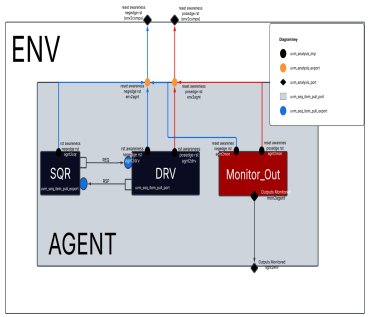
Some of these components have direct UVM_TLM_ANALYSIS_IMPS or a mix of UVM_TLM_ANALYSIS _EXPORTS & UVM_TLM_ANALYSIS_FIFOS




```
//-------------------------------------
// Connect phase
//-------------------------------------
function void connect_phase(uvm_phase phase);
  super.connect_phase(phase);

  // Connect alu agent to env TLM
  alu_agent_h.inputMon2agent.connect(alu_in_env_ap);
  alu_agent_h.outputMon2agent.connect(alu_out_env_ap);

  // Connect env to scoreboard
  alu_in_env_ap.connect(scb_h.analysis_export_inputs);
  alu_out_env_ap.connect(scb_h.analysis_export_actual_outputs);

  // Connect env to subscriber
  alu_in_env_ap.connect(sub_h.analysis_export_inputs);
  alu_out_env_ap.connect(sub_h.analysis_export_outputs);

/*****************************************************************************************/

  // Connect reset agent to env
  rst_agent_h.reset_collected_port_n.connect(reset_collected_port_n); // reset_agent_to_env
  rst_agent_h.reset_collected_port_p.connect(reset_collected_port_p); // reset_agent_to_env

  // Connect env to alu agent
  reset_collected_port_n.connect(alu_agent_h.reset_collected_export_n);
  reset_collected_port_p.connect(alu_agent_h.reset_collected_export_p);

  // Connect reset to alu agent
  reset_collected_port_n.connect(scb_h.reset_collected_export); // scoreboard we may send posedge rst later
  reset_collected_port_n.connect(sub_h.reset_collected_export); // subscriber we may send posedge rst later

/*****************************************************************************************/
  // sequencers connection
  v_seqr.seqr_ALU = alu_agent_h.sequencer_h;
  v_seqr.seqr_RST = rst_agent_h.sequencer_h;
```







TLM debugging & Tracing

The get provided to & get connected to functions help debugging the tlm of your uvm hierarchy if there is ever an issue with it.

It gets passed a property of type uvm_port_list which I named list in this case.

```
//---------------------------------------
// End of elaboration phase -  for reporting connection details
//---------------------------------------
function void end_of_elaboration_phase(uvm_phase phase);
  super.end_of_elaboration_phase(phase);

  // Set verbosity level for the scoreboard
  scoreboard_h.set_report_verbosity_level_hier(UVM_HIGH);

  // Log connections and provided ports/exports
  alu_agnt.driver.reset_collected_imp.get_provided_to(list);
  `uvm_info(get_name(), $sformatf("%p", list), UVM_LOW);

  alu_agnt.driver.reset_collected_imp.get_connected_to(list);
  `uvm_info(get_name(), $sformatf("%p", list), UVM_LOW);

  alu_reset_agnt.monitor.reset_collected_port.get_provided_to(list);
  `uvm_info(get_name(), $sformatf("%p", list), UVM_LOW);

  alu_reset_agnt.monitor.reset_collected_port.get_connected_to(list);
  `uvm_info(get_name(), $sformatf("%p", list), UVM_LOW);

  alu_reset_agnt.reset_collected_port.get_provided_to(list);
  `uvm_info(get_name(), $sformatf("%p", list), UVM_LOW);

  alu_reset_agnt.reset_collected_port.get_connected_to(list);
  `uvm_info(get_name(), $sformatf("%p", list), UVM_LOW);

  alu_agnt.reset_collected_export.get_provided_to(list);
  `uvm_info(get_name(), $sformatf("%p", list), UVM_LOW);

  alu_agnt.reset_collected_export.get_connected_to(list);
  `uvm_info(get_name(), $sformatf("%p", list), UVM_LOW);
endfunction
```

Virtual Sequence Structure

- If a reset is asserted, all the reset aware components get disabled and restart their processes again, therefore a forever loop is used in the sequences, and a do-while loop is used in the virtual sequences(in the repititon_v_seq).
- **Both sequences (RST & repititon sequence) need to be running together not to ruin the driver/sequencer handshake, therefore causing errors.**
- If the reset sequence finishes, it doesn't affect the other **alu_sequence since they have stop_sequences() called within their sequencers.**
- On the other hand, if the roles were reversed, **the reset_sequencer would generate errors due to wait for grant task being terminated without an ack**

```
fork
  begin
    rst_seq.start(p_sequencer.seqr_RST);
    `uvm_info(get_name, $sformatf("seqr_RST Arbitration mode = %s", seqr_RST.get_arbit
  end
  begin
    fork
      op_a_seq_h.start(p_sequencer.seqr_ALU  , .this_priority(100));
      op_b01_seq_h.start(p_sequencer.seqr_ALU , .this_priority(200));
      op_b11_seq_h.start(p_sequencer.seqr_ALU , .this_priority(300));
    join
  end
  begin
    #test_timeout;
    `uvm_info(get_name,("TIME OUT!!"), UVM_LOW)
    alu_seq_item::cov_target=1;
    rst_seq_item::resets_done=1;
  end
join_any
            disable fork;
nd while ((!alu_seq_item::cov_target) || (!rst_seq_item::resets_done));
task
```

Virtual Sequence Structure

- **A timeout is added to ensure the test doesn't hang if the coverage target isn't hit.**
- The **timeout is set by the top testbench** and is transferred to the environment **using the test_config** through the base_test.
- Sequence Arbitiration is added for practice, in this mode, the sequence **with the highest priority inside the fork_join gets started first**
- The base of the Virtual Sequence (base_v_seq) is used **to instantiate the sequencers, declare the p_sequencer**
- Each **v_seq** is responsible for **starting the sequences using the p_sequencer**

```
fork
  begin
    rst_seq.start(p_sequencer.seqr_RST);
    `uvm_info(get_name, $sformatf("seqr_RST Arbitration mode = %s", seqr_RST.get_arbit
  end
  begin
    fork
      op_a_seq_h.start(p_sequencer.seqr_ALU  , .this_priority(100));
      op_b01_seq_h.start(p_sequencer.seqr_ALU , .this_priority(200));
      op_b11_seq_h.start(p_sequencer.seqr_ALU , .this_priority(300));
    join
  end
  begin
    #test_timeout;
    `uvm_info(get_name,("TIME OUT!!"), UVM_LOW)
    alu_seq_item::cov_target=1;
    rst_seq_item::resets_done=1;
  end
join_any
            disable fork;
nd while ((!alu_seq_item::cov_target) || (!rst_seq_item::resets_done));
task
```

Test Termination

This Test termination technique is mentioned in [Clifford cummings' OVM & UVM Techniques for Terminating Tests]

- The **phase ready to end** checks if the run phase is still running, then checks for the test's termination flag.
- **(alu_seq_item::cov_target) which is asserted only when respective coverage for the respective test reaches 100%.**
- **(rst_seq_item::resets_done) which is only asserted when the specified number of resets which were defined through the test_config file are finished.**
- Used the **Final Phase** for the scoreboard to determine when the scoreboard was shutting down to make sure no items are slipping through the test without being compared properly.

```
//    ---------------------------------------
//    phase_ready_to_end phase
//    ---------------------------------------
  function void phase_ready_to_end(uvm_phase phase);
    if (phase.get_name() != "run") return;
    if (~alu_seq_item::cov_target || ~rst_seq_item::resets_done) begin
      phase.raise_objection(.obj(this));
      fork
        begin
          delay_phase(phase);
        end
      join_none
    end
  endfunction

  //---------------------------------------
  // delay phase
  //---------------------------------------
  task delay_phase(uvm_phase phase);
    wait(alu_seq_item::cov_target && rst_seq_item::resets_done);
    phase.drop_objection(.obj(this));
  endtask

  //---------------------------------------
  // final phase
  //---------------------------------------
  function void final_phase(uvm_phase phase);
    super.final_phase(phase);
    `uvm_info("SCOREBOARD", "Scoreboard is stopping.", UVM_LOW)
  endfunction
```

Ensuring Coverage target is hit

- **Both reset sequence & random the other sequence which is running alongside it is in a forever loop waiting to hit 100% coverage or to hit test timeout**
- **Breaking from the body task only when the number of specified resets is done, and the cov_target is done.**

```
virtual task body();

  forever begin
  req = alu_seq_item::type_id::create("req");
    start_item(req);

    if(!req.randomize())
      `uvm_error(get_type_name(), "Randomization failed")

    finish_item(req);

    if(alu_seq_item::cov_target && rst_seq_item::resets_done) break;
```

```
virtual task body();
    req = rst_seq_item::type_id::create("req");
  forever begin
    start_item(req);
    ok = req.randomize();
    if(!ok)
      `uvm_error(get_type_name(), "Randomization @rst_seq_item failed")
    finish_item(req);
    if(rst_seq_item::resets_done)begin
      $display("atyab? ma fe atyab mn hek");
      break;
    end
  end
```

Type Overrides

```
virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
// override virtual sequence
  base_v_seq::type_id::set_type_override(random_v_seq::type_id::get());
  v_seq    = base_v_seq::type_id::create("v_seq", this);

  cmd = uvm_cmdline_processor::get_inst();
  cmd.get_plusargs(plusargs_queue);
  `uvm_info (get_type_name(), $sformatf("PLUSARGS = %p", plusargs_queue), UVM_LOW)
endfunction : build_phase
```

```
virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  // override virtual sequence
  base_v_seq::type_id::set_type_override(error_v_seq::type_id::get());
  alu_seq_item::type_id::set_type_override(error_seq_item::type_id::get());
```

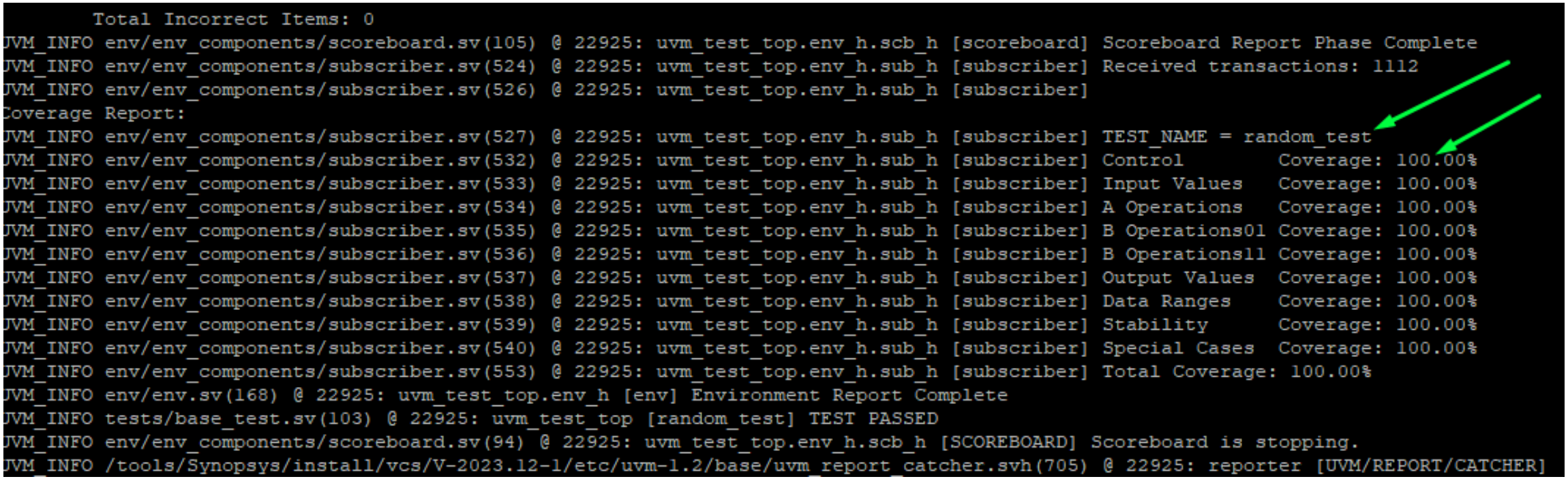
Changing verbosity level of a certain part in the hierarchy

```
// Set verbosity level for the scoreboard
scoreboard_h.set_report_verbosity_level_hier(UVM_HIGH);
```

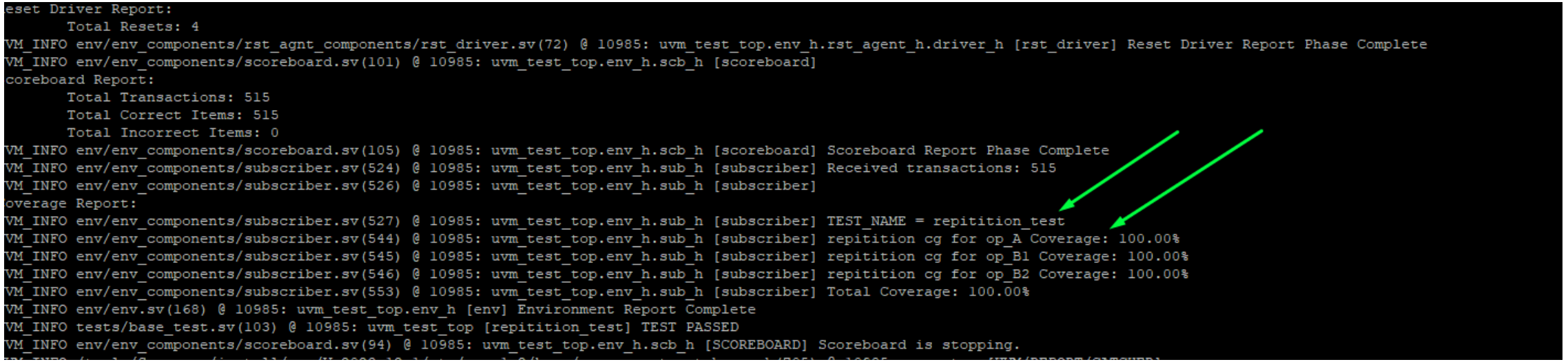
Utilizing report phase to report after the conclusion of the test

```
//---------------------------------------
// Report phase
//---------------------------------------
function void report_phase(uvm_phase phase);
  `uvm_info(get_type_name(),
            $sformatf("\nReset Driver Report:\n\tTotal Resets: %0d",rst_seq_item::reset_count), UVM_LOW)

  `uvm_info(get_type_name(), "Reset Driver Report Phase Complete", UVM_LOW)
endfunction : report_phase
```

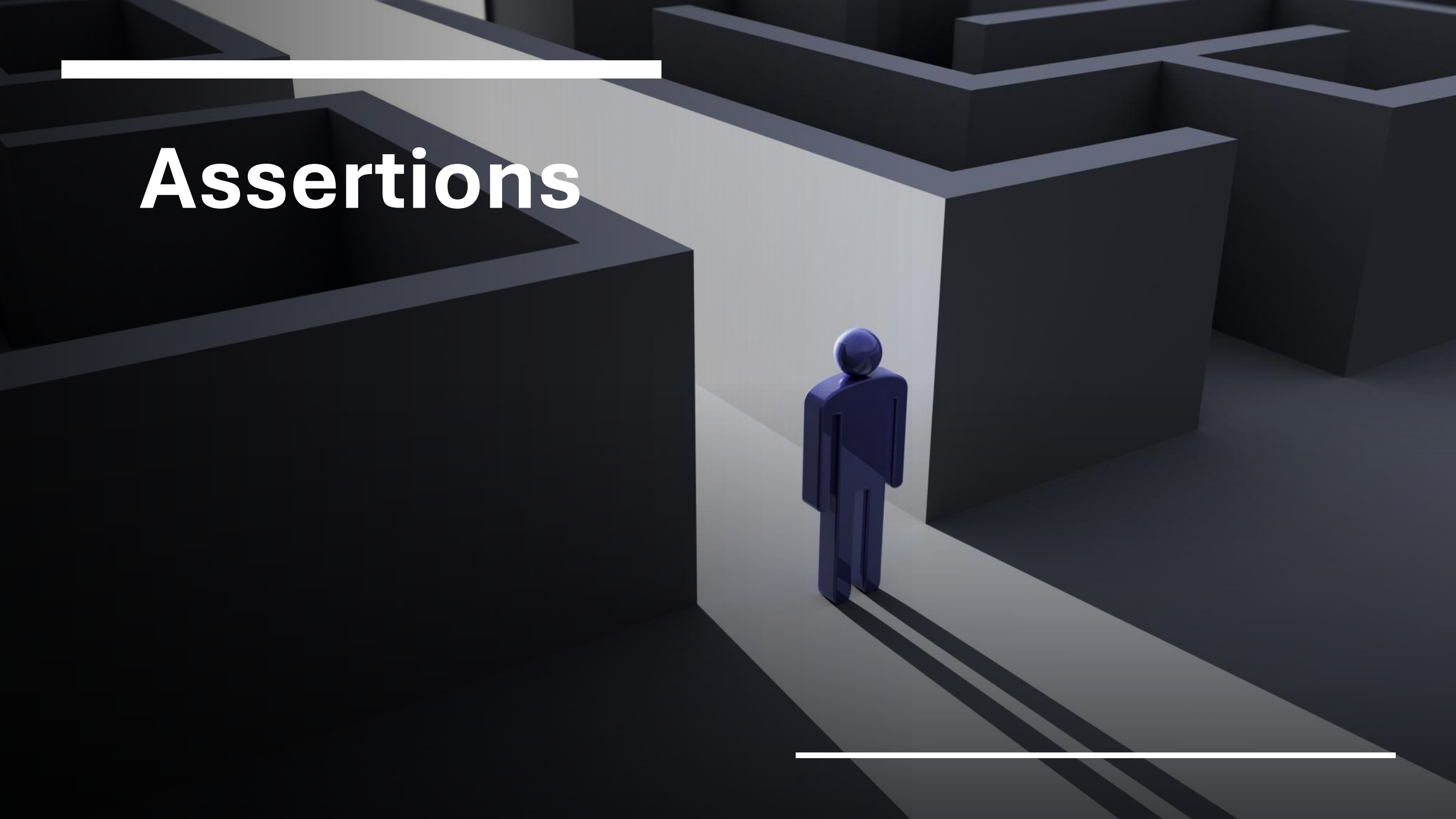
Random Test coverage target hit



Repetition Test Coverage target hit



Assertions



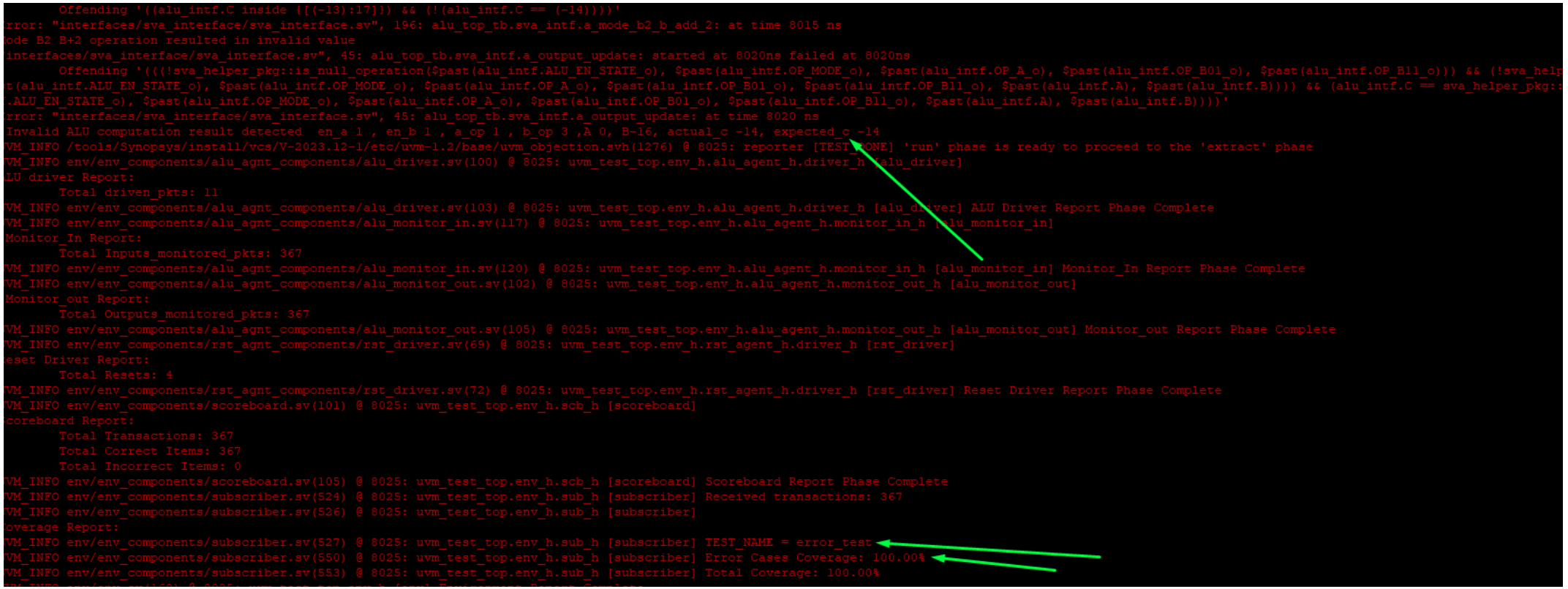
Assertions

- Asserting over every possible scenario, even the error ones, providing CEXs to see them being triggered during the error_test run
- Asserting over each property with an $error if the property fails
- Covering each property

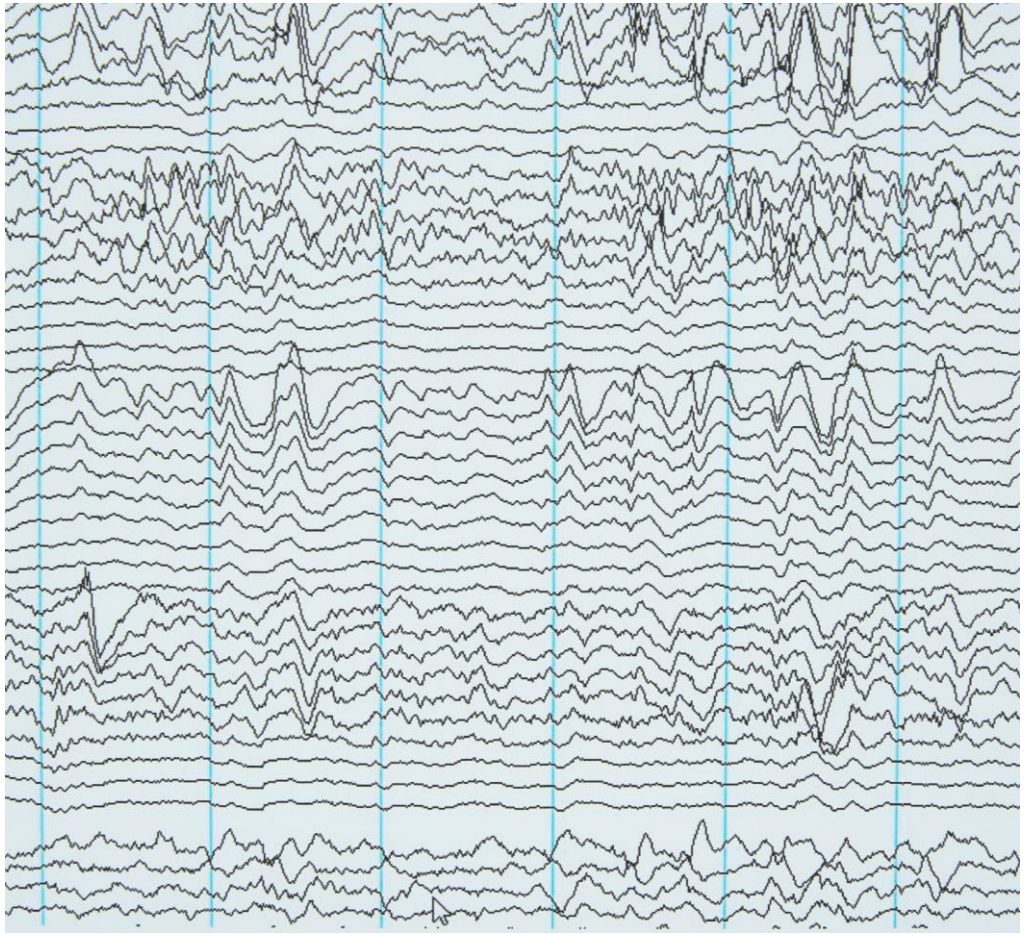
```
property p_output_update;
  disable iff (~rst_n)
  @(alu_intf.driver_cb)
  $changed({alu_intf.ALU_EN_STATE_o, alu_intf.OP_MODE_o, alu_intf.OP_A_o, alu_intf.OP_B01_o,
            alu_intf.OP_B11_o, alu_intf.A, alu_intf.B}) |=>

  // First check for NULL operation - this should cause assertion failure
  !is_null_operation($past(alu_intf.ALU_EN_STATE_o), $past(alu_intf.OP_MODE_o), $past(alu_intf.OP_A_o),
                     $past(alu_intf.OP_B01_o), $past(alu_intf.OP_B11_o)) &&

  // Then check for forbidden results (-32, or MODE_B2 special forbidden values)
  !is_forbidden_result($past(alu_intf.ALU_EN_STATE_o), $past(alu_intf.OP_MODE_o), $past(alu_intf.OP_A_o),
                       $past(alu_intf.OP_B01_o), $past(alu_intf.OP_B11_o),
                       $past(alu_intf.A), $past(alu_intf.B)) &&

  // Then check for expected output value based on the mode
  alu_intf.C == compute_result(rst_intf.rst_n, $past(alu_intf.ALU_EN_STATE_o), $past(alu_intf.OP_MODE_o), $past(alu_intf.OP_A_o),
                               $past(alu_intf.OP_B01_o), $past(alu_intf.OP_B11_o),
                               $past(alu_intf.A), $past(alu_intf.B));
endproperty
```

```
  OP_B11_t op_B11, // Operation
  int signed low_range, int signed high_range,          // Valid range [low:high]
  int signed invalid_min, int signed invalid_max        // Invalid values or specific exclusions
);
  @(alu_intf.monitor_cb) disable iff (!rst_n)
    (alu_intf.ALU_EN_STATE_o == ALU_ON &&
     alu_intf.OP_MODE_o == op_MODE &&
    (op_MODE == MODE_A   ? (alu_intf.OP_A_o == op_A) : 1'b1) &&
    (op_MODE == MODE_B01 ? (alu_intf.OP_B01_o == op_B01) : 1'b1) &&
    (op_MODE == MODE_B11 ? (alu_intf.OP_B11_o == op_B11) : 1'b1) &&
   (!(op_MODE == MODE_IDLE)))
   |=>  (alu_intf.C inside {[low_range:high_range]} && !(alu_intf.C inside {[invalid_min:invalid_max]}));
endproperty

// Properties for specific operations using parameterized templates
// Mode A operations
property p_mode_a_add;
  p_op_result_range(MODE_A, A_ADD, 'x, 'x, -30, 30, -32, -31);
endproperty

property p_mode_a_sub;
  p_op_result_range(MODE_A, A_SUB, 'x, 'x, -30, 30, -32, -31);
```

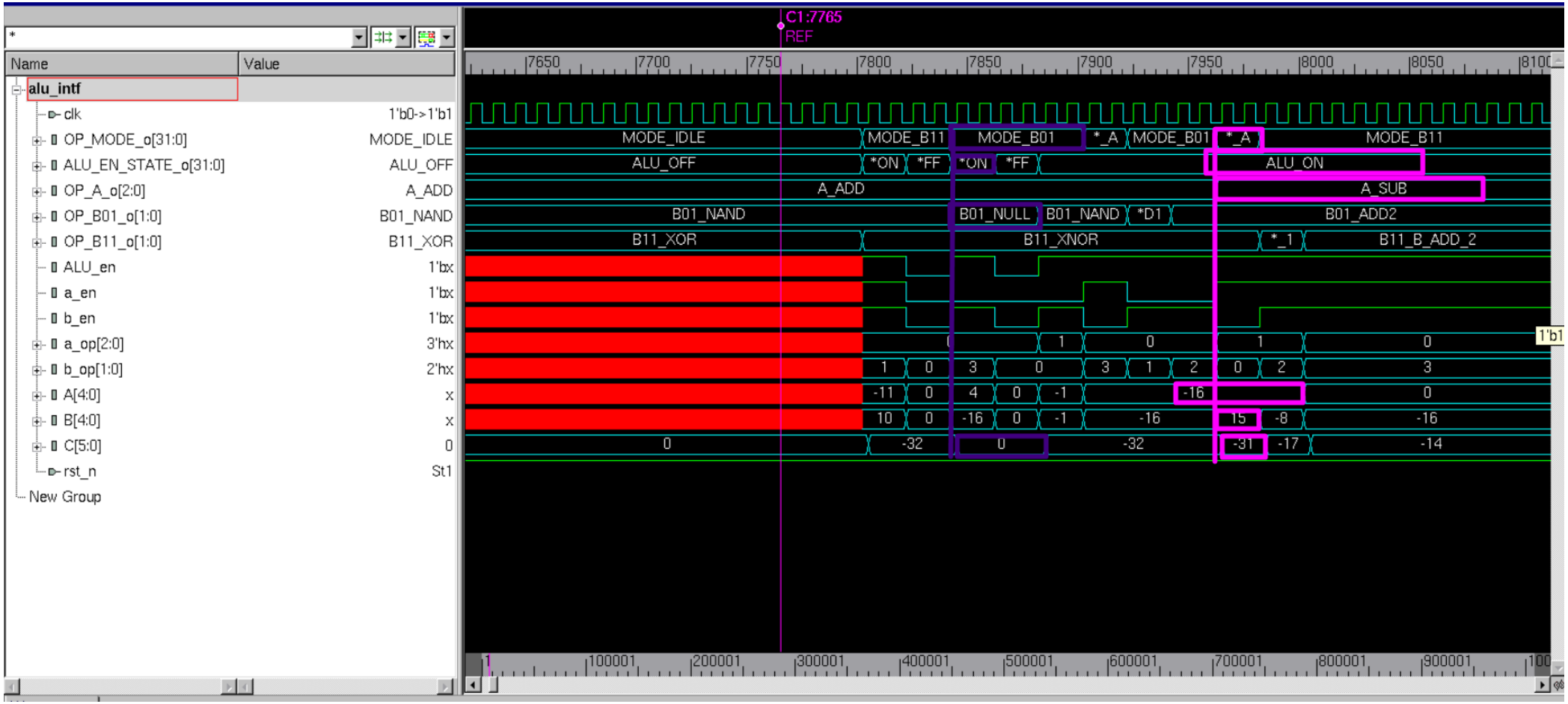
Error Test CEX assertions firing



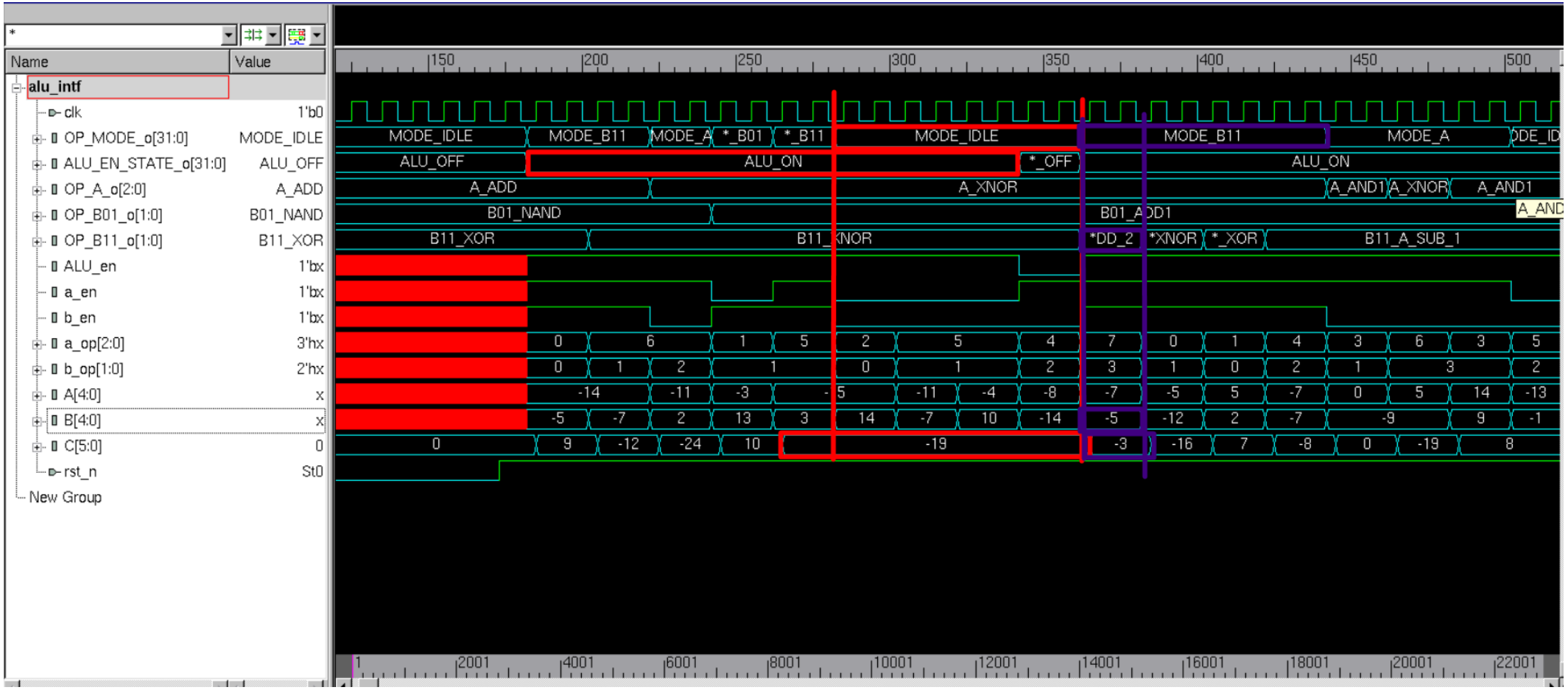
Waveforms



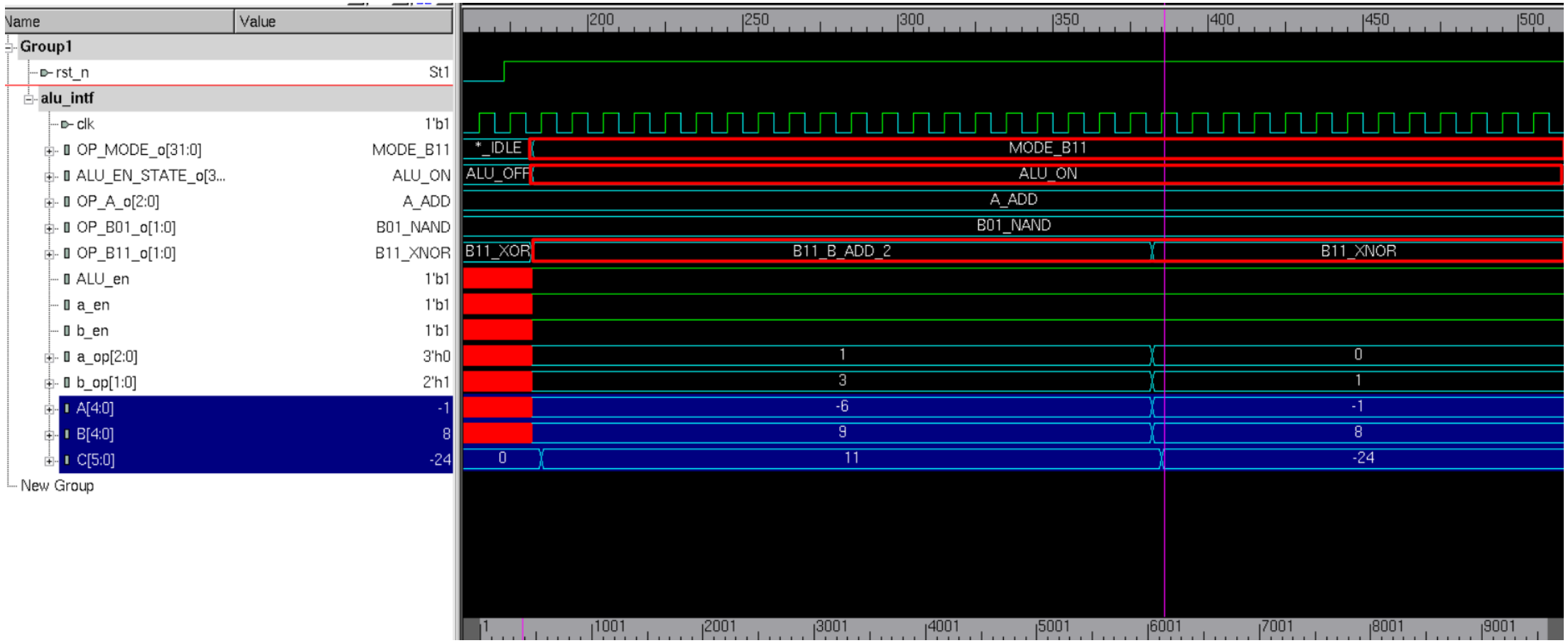
Error Test Waveform

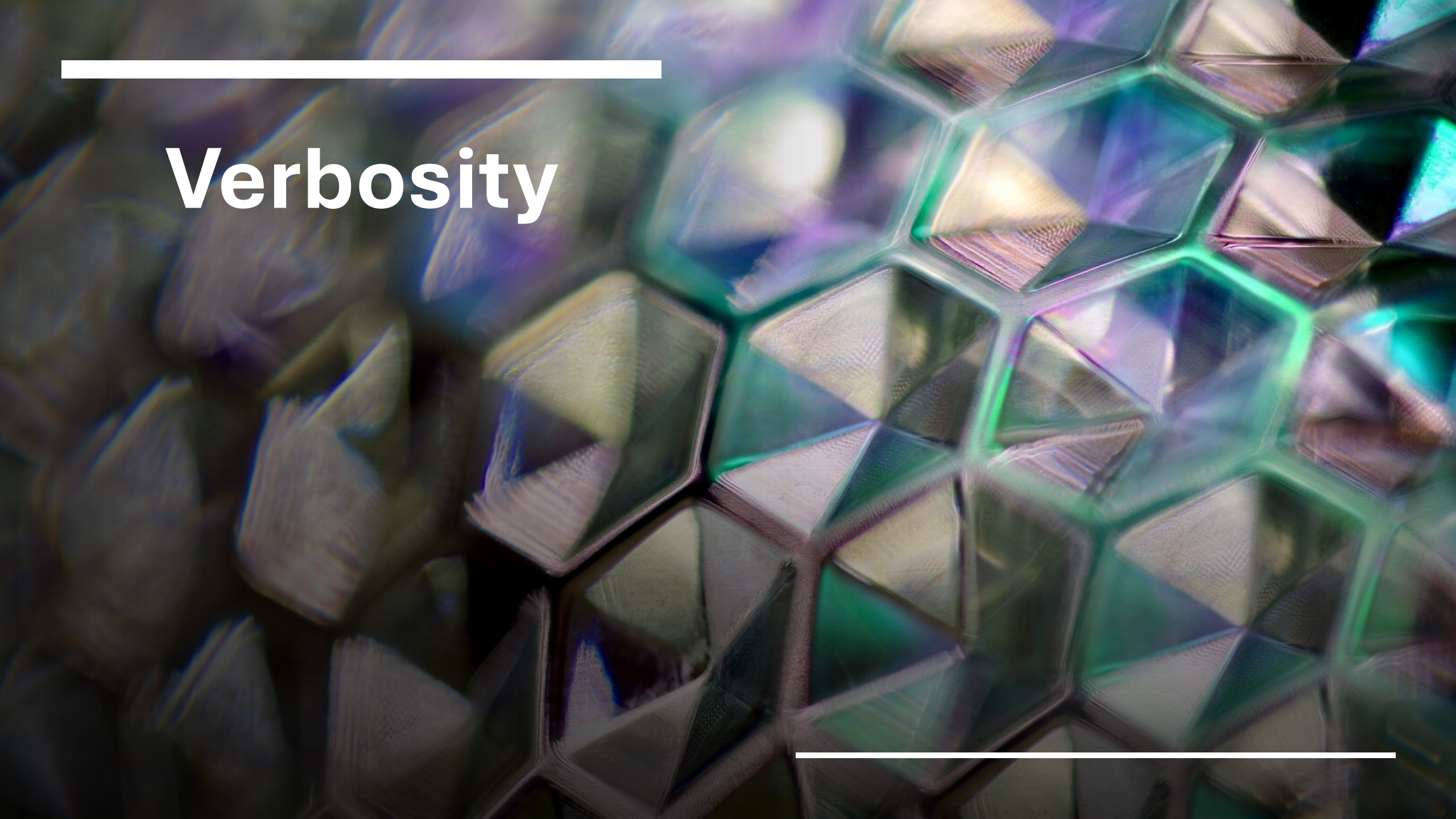


Random Test Waveform



Repetition Test Waveform




Verbosity

Different Verbosity for Different Debugging Needs

UVM_LOW for Reporting & Checking that the hierarchy is starting correctly & Scoreboard Mismatches

```
function void report_phase(uvm_phase phase);
  `uvm_info(get_type_name(), $sformatf("Received transactions: %0d", count_trans), UVM_LOW)

  `uvm_info(get_type_name(), "\nCoverage Report:", UVM_LOW)
  `uvm_info(get_type_name(), $sformatf("TEST_NAME = %s", test_name), UVM_LOW)


  case(test_name)
    "random_test": begin
      `uvm_info(get_type_name(), $sformatf("Control        Coverage: %.2f%%", control_cg.get_coverage()), UVM_LOW)
```

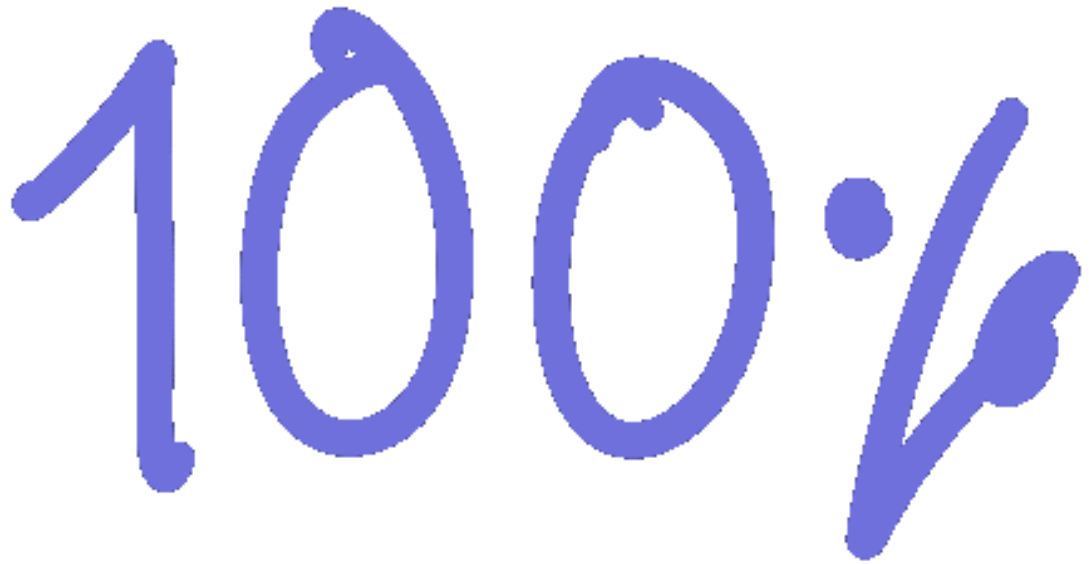
UVM_HIGH for deep-level & case Debugging (usually during the development phase)

```
  //------------------------------------
  virtual task run_phase(uvm_phase phase);
    //-------------------------------------
    //Wait for reset de-assertion
    //-------------------------------------
    @(posedge reset_deassertion); // first reset obologatry
    `uvm_info(get_full_name(), $sformatf("first reset detect :\n"), UVM_HIGH)
```

UVM_MEDIUM for Correct Cases tracing & Expected behaviour

```
  // Compare expected vs actual
  if (expected_item.C === actual_item.C) begin
    correct_counter++;
    `uvm_info(get_type_name(), $sformatf("CORRECT %0d: Expected = %0d, Actual = %0d",
              correct_counter, expected_item.C, actual_item.C), UVM_MEDIUM)

    `uvm_info(get_type_name(),$sformatf("CORRECT %0d: Expected_TXN_counter = %0d, Actual_TXN_counter = %0d",
              correct_counter, alu_seq_item::txn_counter_expected, alu_seq_item::txn_counter_actual), UVM_MEDIUM)
  end
```

COVERAGE? What? For What?



Coverage Examples

Fixed/Cross – based coverage method

```
covergroup inputs_cg();
   A_df:coverpoint input_cov_copied.A iff (input_cov_copied.ALU_en & input_cov_copied.rst_n){
      option.weight = ((input_cov_copied.A == -16)?0:1);
      bins A[] = {[-16:15]};
      `ifdef RANDOM_ERROR_INJECTION_TEST
          ignore_bins A_ignored[] = {-16};
      `else
          illegal_bins A_illegal[] = {-16};
      `endif
   }

   B_df:coverpoint input_cov_copied.B iff (input_cov_copied.ALU_en & input_cov_copied.rst_n){
```

```
A_B_Aen_Ben_Aop_Bop: cross A_df, B_df, A_en_df, B_en_df, A_op_df, B_op01_df, B_op11_df{

    bins A_op_cases = binsof(A_df) && binsof(B_df) && binsof(A_en_df) intersect {1} &&
                      binsof(B_en_df) intersect {0} && binsof(A_op_df);

    bins B_op01_cases = binsof(A_df) && binsof(B_df) && binsof(A_en_df) intersect {0} &&
                        binsof(B_en_df) intersect {1} && binsof(B_op01_df);

    bins B_op11_cases = binsof(A_df) && binsof(B_df) && binsof(A_en_df) intersect {1} &&
                        binsof(B_en_df) intersect {1} && binsof(B_op11_df);
}
```

BODY

Instance – based coverage method

```
covergroup A_cg_df(input int signed i, input transaction_base cov);
    option.weight = ((i == -16)? 0:1);
    option.name = $sformatf("df = %0d",i);
    option.per_instance = 1;
    df: coverpoint cov.A iff (cov.rst_n && cov.ALU_en) {
        bins A[] = {i};
        `ifdef RANDOM_ERROR_INJECTION_TEST
            ignore_bins A_ignored[] = {-16};
        `else
            illegal_bins A_illegal[] = {-16};
        `endif
    }
endgroup : A_cg_df
```

```
covergroup A_op_cg_dt(input int i, input int k , input transaction_base cov);
    option.weight = ((k == 7 || i == 7)? 0:1);
    option.name = $sformatf("dt %0d => %0d", i, k);
    option.per_instance = 1;
    dt: coverpoint cov.a_op iff (cov.rst_n && (cov.a_en & ~cov.b_en) && cov.ALU_en) {
        bins A_op[] = (i => k);
        `ifdef RANDOM_ERROR_INJECTION_TEST
            ignore_bins A_op_ignored1[] = (i => 7);
            ignore_bins A_op_ignored2[] = (7 => k);
        `else
            illegal_bins A_op_illegal1[] = (i => 7);
            illegal_bins A_op_illegal2[] = (7 => k);
        `endif
    }
endgroup : A_op_cg_dt
```

Construction

```
inputs_cg = new();
```

```
outputs_cg = new();
```

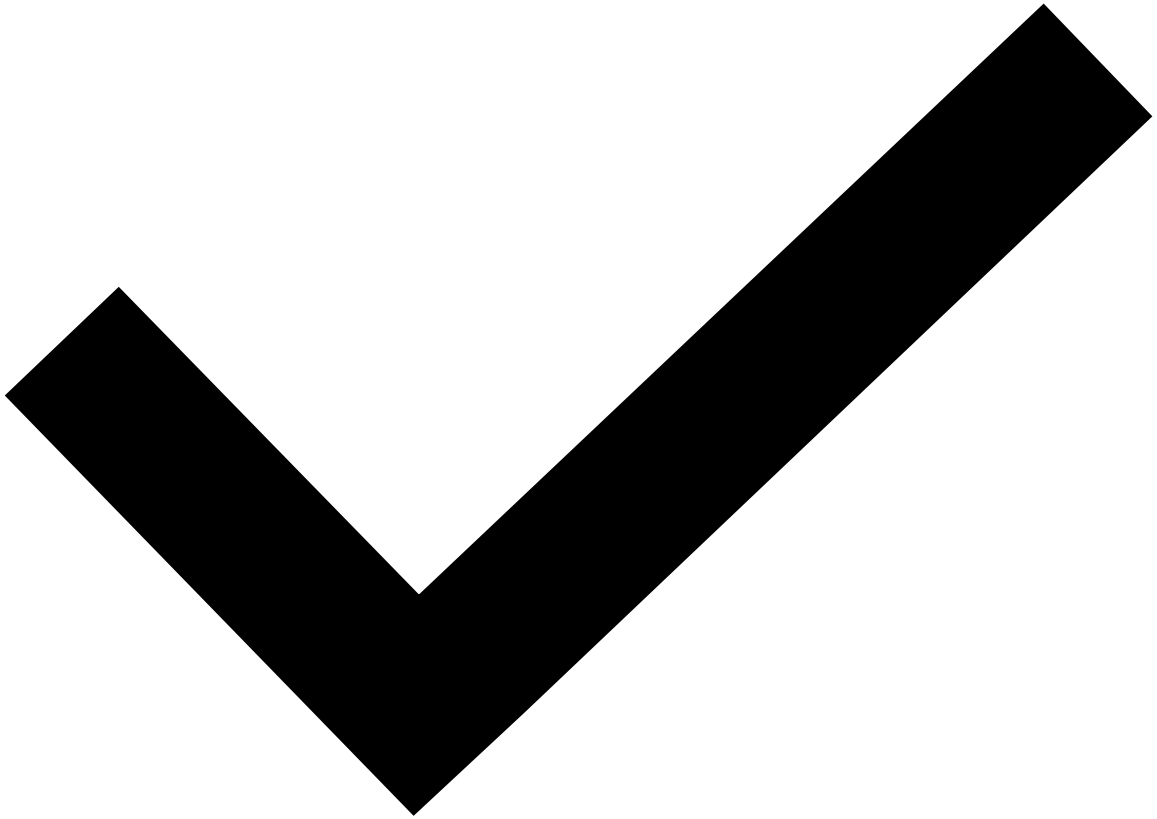
```
foreach(A_op_cg_df_vals[i]) A_op_cg_df_vals[i]     = new(i, input_cov_copied);
```

Sampling

```
inputs_cg.sample();
outputs_cg.sample();
```

```
foreach(A_cg_df_vals[i]) A_cg_df_vals[i].sample();
foreach(B_cg_df_vals[i]) B_cg_df_vals[i].sample();

foreach(A_op_cg_dt_vals[i,j]) A_op_cg_dt_vals[i][j].sample();
foreach(B_op01_cg_dt_vals[i,j]) B_op01_cg_dt_vals[i][j].sample();
foreach(B_op11_cg_dt_vals[i,j]) B_op11_cg_dt_vals[i][j].sample();
```

Coverage Closure



Dashboard.txt for valid Test

```
Dashboard

Date: Mon Mar 24 10:16:12 2025

User: svgpdv25abohamad

Version: V-2023.12-1
Command line: urg -dir random_test_simulation_executable.vdb -elfile random_test.el -dir repitition_test_simulation_executable.vdb -elfile repitition_test.el -format text -report coverage_closure/
valid_tests_cov_merged

Number of tests: 2

------------------------------------------------------------------------------
Total Coverage Summary
SCORE  LINE   COND   TOGGLE BRANCH ASSERT GROUP
 99.07 100.00 100.00 100.00 100.00  94.44 100.00


------------------------------------------------------------------------------
Hierarchical coverage data for top-level instances
SCORE  LINE   COND   TOGGLE BRANCH ASSERT NAME
 99.61 100.00 100.00 100.00 100.00  98.04 alu_top_tb(x)
 33.33 --     --     --     --      33.33 uvm_pkg


------------------------------------------------------------------------------
Total Module Definition Coverage Summary
SCORE  LINE   COND   TOGGLE BRANCH ASSERT
 80.81  75.59  78.12  92.41  63.49  94.44


------------------------------------------------------------------------------
Total Groups Coverage Summary
SCORE  WEIGHT
100.00 1
```

```
[svgpdv25abohamad@academysvr02 valid_tests_cov_merged]$ ls
asserts.txt  dashboard.txt  groups.txt  grpinfo.txt  hierarchy.txt  modinfo.txt  modlist.txt  session.xml  tests.txt
```

Dashboard.txt for Error Injection Test

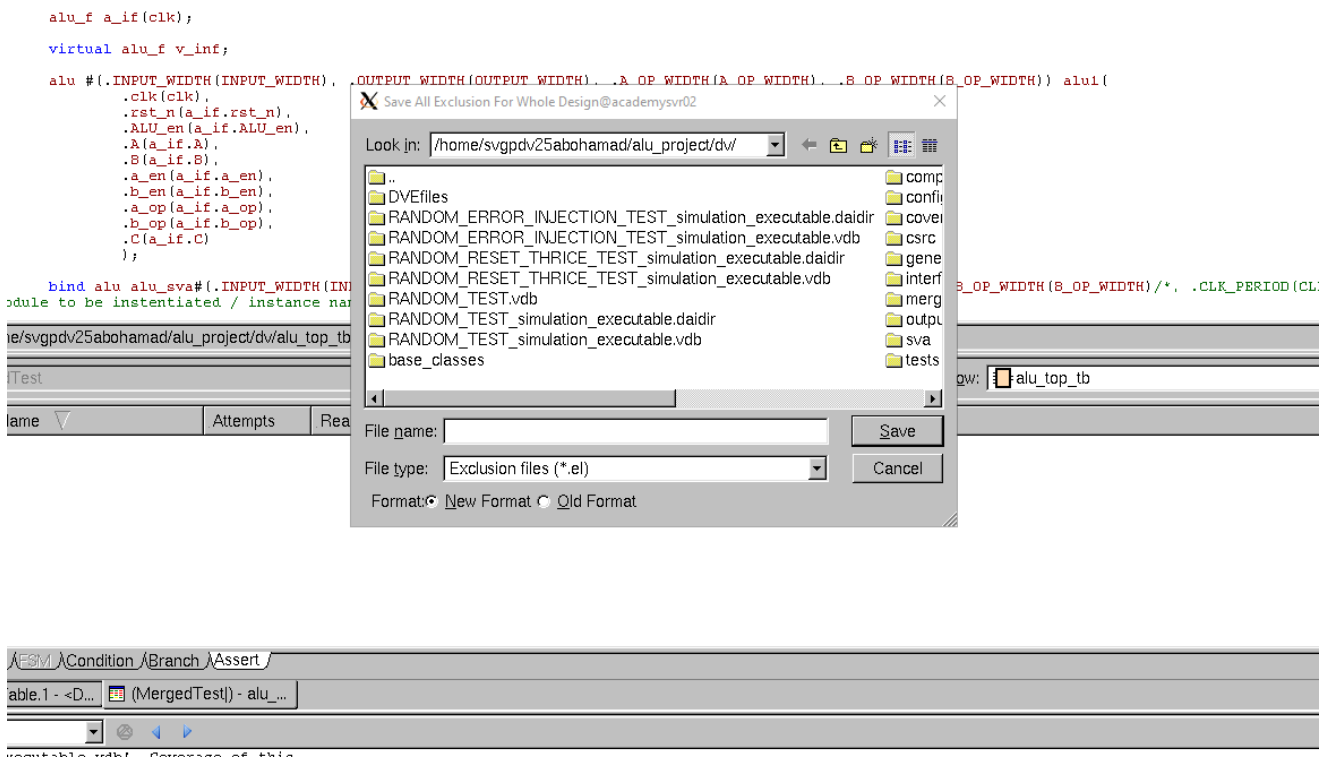
```
Date: Mon Mar 24 10:16:12 2025

User: svgpdv25abohamad

Version: V-2023.12-1
Command line: urg -dir error_test_simulation_executable.vdb -format text -elfile error_test.el -report coverage_closure/error_injection_cov_merged

Number of tests: 1

------------------------------------------------------------------------------
Total Coverage Summary
SCORE  LINE   COND   TOGGLE BRANCH ASSERT GROUP
 86.73 100.00 100.00 100.00 100.00  20.37 100.00


------------------------------------------------------------------------------
Hierarchical coverage data for top-level instances
SCORE  LINE   COND   TOGGLE BRANCH ASSERT NAME
 83.92 100.00 100.00 100.00 100.00  19.61 alu_top_tb(x)
 33.33 --     --     --     --      33.33 uvm_pkg


------------------------------------------------------------------------------
Total Module Definition Coverage Summary
SCORE  LINE   COND   TOGGLE BRANCH ASSERT
 55.26  68.50  56.25  83.54  47.62  20.37


------------------------------------------------------------------------------
Total Groups Coverage Summary
SCORE  WEIGHT
100.00 1
```

Most assertions failing due to error injection which is a good sign...

```
[svgpdv25abohamad@academysvr02 error_injection_cov_merged]$ ls
asserts.txt  dashboard.txt  groups.txt  grpinfo.txt  hierarchy.txt  modinfo.txt  modlist.txt  session.xml  tests.txt
```

Exclusions and .el files

Exclusions can be done in the merge command line using –exclude but the naming of the branches, points, ..etc is often hard to make out and understand to just do it from the line, therefore using dve to make the .el file is much easier and thereafter including them in the report command.



Scripting & Makefiles

```
COVERAGE_DIR := coverage_closure
VERDI := verdi
DVE := dve

# Define the default test names (can be overridden via command line)
TEST_NAME  ?= random_test # Default test name if not specified
TEST_NAME1 ?= random_test
TEST_NAME2 ?= repitition_test
TEST_NAME3 ?= error_test

# Default tools
WAVEFORM_VIEWER ?= dve  # Default waveform viewer (can be 'verdi' or 'dve')
COVERAGE_MERGE_TOOL ?= urg  # Default coverage merge tool (can be 'urg' or 'verdi')

# Derived variables based on TEST_NAME
SIM_EXEC := $(TEST_NAME)_simulation_executable
VPD_FILE := $(TEST_NAME).vpd
COMPILE_LOG := $(TEST_NAME)_compile.log
SIM_LOG := $(TEST_NAME)_sim.log
COVERAGE_DB := $(SIM_EXEC).vdb

# Compilation and simulation flags
FILE_LIST_FLAGS := -f
COVERAGE_FLAGS := -cm branch+cond+tgl+line+assert
VCS_UVM_LIB_FLAG := -ntb_opts uvm-1.2
VCS_FLAGS := -full64 -sverilog $(VCS_UVM_LIB_FLAG) -timescale=$(TIME_SCALE) $(COVERAGE_FLAGS) -l $(OUTPUT_DIR)/$(COMPILE_LOG) -debug_access+r +R
VPD_FLAGS := -vpd $(OUTPUT_DIR)/$(VPD_FILE)
UVM_TESTNAME := +UVM_TESTNAME=
VERBOSITY_LEVEL ?= UVM_LOW
UVM_VERBOSITY := +UVM_VERBOSITY=

# Create necessary directories
$(OUTPUT_DIR) $(COVERAGE_DIR) $(VDB_OUTPUT_DIR):
        @mkdir -p $@
        @echo "Created directory: $@"

# Compile the design and testbench with the specified test name
compile: $(OUTPUT_DIR) $(VDB_OUTPUT_DIR) $(DESIGN_FILE) $(TESTBENCH_FILE)
        @echo "Compiling design and testbench for test: $(TEST_NAME)..."
        @$(VCS) $(VCS_FLAGS) $(FILE_LIST_FLAGS) $(DESIGN_FILE) $(FILE_LIST_FLAGS) $(TESTBENCH_FILE) -o $(SIM_EXEC) || { echo "Compilation failed"; exit 1; }
        @echo "Compilation complete for $(TEST_NAME)."

# Run the simulation and dump the VPD file with the test-specific name
run: compile
        @echo "Running simulation for test: $(TEST_NAME)..."
        @./$(SIM_EXEC) $(COVERAGE_FLAGS) $(UVM_TESTNAME)$(TEST_NAME) $(UVM_VERBOSITY)$(VERBOSITY_LEVEL) +ntb_random_automatic -l $(OUTPUT_DIR)/$(SIM_LOG) $(VPD_FLAGS)  || { echo "Simulation failed"; exit 1; }
        @echo "Simulation complete. VPD file dumped to $(OUTPUT_DIR)/$(VPD_FILE)."

# View waveform with selected viewer (verdi or dve)
view-waveform: $(OUTPUT_DIR)
        @if [ "$(WAVEFORM_VIEWER)" = "verdi" ]; then \
```

default name for tests incase of no inputs

each file parametrised by the test_name

running with test_names

using file lists

random_seed generation

Makefile to compile, run, check the simulation waveform using the .vdp files & clean each time the clean task is called or when the run_all.sh is executed

Saving compile and simulation.log files so they can be searched for any errors using a script if I have the time to make one

Choosing between Verdi & DVE for wave viewing
Choosing between urg and Verdi for vdb merging
USE make help to understand the tasks

```
# Generate coverage reports
generate-reports:
        @echo "Generating coverage reports......"
        urg -dir $(TEST_NAME1)_simulation_executable.vdb $(TEST_NAME2)_simulation_executable.vdb -format text -elfile RANDOM_TEST.el -report $(COVERAGE_DIR)/valid_tests_cov_merged.txt
        urg -dir $(TEST_NAME3)_simulation_executable.vdb -format text -elfile RANDOM_ERROR_INJECTION_TEST.el -report $(COVERAGE_DIR)/error_injection_cov_merged.txt

clean:
        @echo "Cleaning up..."
        @rm -rf "$(OUTPUT_DIR)"/* "$(COVERAGE_DIR)"/* \
                "$(TEST_NAME1)_simulation_executable" \
                "$(TEST_NAME2)_simulation_executable" \
                "$(TEST_NAME3)_simulation_executable" \
                "$(TEST_NAME1)_simulation_executable.daidir" \
                "$(TEST_NAME2)_simulation_executable.daidir" \
                "$(TEST_NAME3)_simulation_executable.daidir" \
                "$(TEST_NAME1)_simulation_executable.vdb" \
                "$(TEST_NAME2)_simulation_executable.vdb" \
                "$(TEST_NAME3)_simulation_executable.vdb"
        @echo "Clean up complete."


# Full build and simulation flow
full-sim: compile run
```

Generating the coverage reports of the valid tests _merging them and of the error_injection test

Cleaning the previous run's output files as mentioned in the bash_script slide

Scripting & Makefiles

```
# Create log directory
mkdir -p "$LOG_DIR"
log "Starting regression test suite"
log "Using makefile: $MAKEFILE"
log "Waveform viewer: $WAVEFORM_VIEWER"
log "Coverage merge tool: $COVERAGE_MERGE_TOOL"

# Interactive tool selection
if $INTERACTIVE; then
    echo "Would you like to select a different waveform viewer? [verdi/dve/no] (default: $WAVEFORM_VIEWER): "
    read viewer_choice
    if [[ "$viewer_choice" == "verdi" || "$viewer_choice" == "dve" ]]; then
        WAVEFORM_VIEWER="$viewer_choice"
        log "Changed waveform viewer to: $WAVEFORM_VIEWER"
    fi

    echo "Would you like to select a different coverage merge tool? [urg/verdi/no] (default: $COVERAGE_MERGE_TOOL): "
    read coverage_choice
    if [[ "$coverage_choice" == "urg" || "$coverage_choice" == "verdi" ]]; then
        COVERAGE_MERGE_TOOL="$coverage_choice"
        log "Changed coverage merge tool to: $COVERAGE_MERGE_TOOL"
    fi
fi

# Clean environment
log "Cleaning environment"
run_cmd "make -f $MAKEFILE clean"

# Run tests
log "Running random_test"
run_cmd "make -f $MAKEFILE TEST_NAME=random_test full-sim"

log "Running repiition_test"
run_cmd "make -f $MAKEFILE TEST_NAME=repiition_test full-sim"

log "Running error_test"
run_cmd "make -f $MAKEFILE TEST_NAME=error_test full-sim"

# Generate reports with specified tools
log "Generating coverage reports using $COVERAGE_MERGE_TOOL for merging"
run_cmd "make -f $MAKEFILE TEST_NAME1=random_test TEST_NAME2=repitition_test TEST_NAME3=error_test COVERAGE_MERGE_TOOL=$COVERAGE_MERGE_TOOL generate-reports"

# Optionally view a waveform if in interactive mode
if $INTERACTIVE; then
    echo "Would you like to view a waveform? [yes/no]: "
    read view_choice
    if [[ "$view_choice" == "yes" || "$view_choice" == "y" ]]; then
        echo "Which test would you like to view? [random_test/repitition_test/error_test]: "
        read test_choice
        run_cmd "make -f $MAKEFILE TEST_NAME=$test_choice WAVEFORM_VIEWER=$WAVEFORM_VIEWER view-waveform"
    fi
```

Bash script that runs regression using the Makefile tasks