TinyALU UVM Verification

Design, Plan, and Implementation

Alex Verfication Team

Alexandria University - Faculty of Engineering
Department of Electronics & Communication

December 14, 2025

Table of contents

TinyALU Design Specs

Design Under Test (DUT): An 8-bit Arithmetic Logic Unit.

Ports:

- Inputs: A [7:0], B [7:0]
- Control: clk, reset_n, start, op [2:0]
- Outputs: result [15:0], done

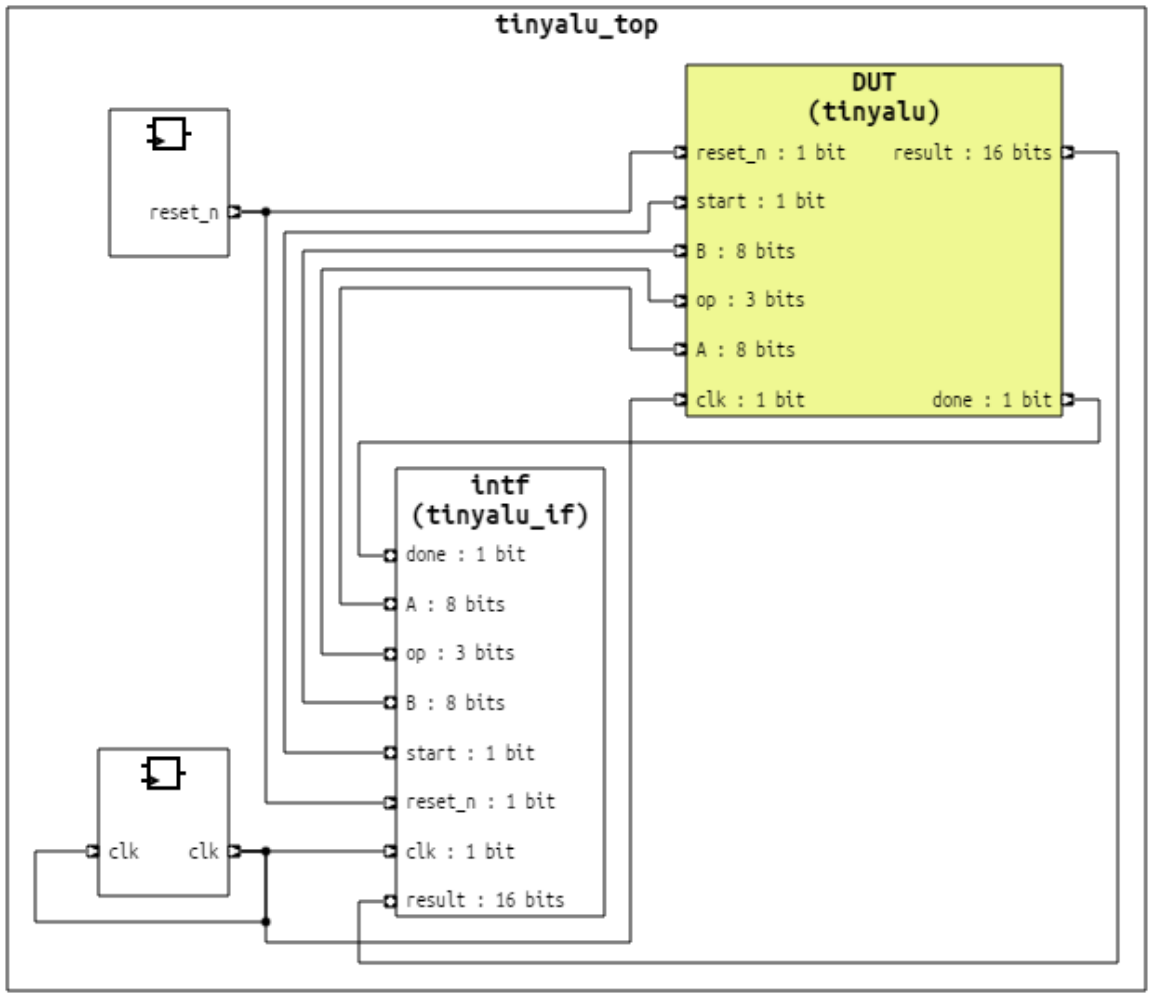
Operations:

- 000: No Op
- 001: Add (+)
- 010: AND (&)
- 011: XOR (^)
- 100: Mul (*)

TinyALU Design Specs

- The `start` signal must remain high until the TinyALU asserts the `done` signal.
- The operator and operands must stay stable while `start` is high and until `done` is raised.
- The `done` signal is asserted for exactly one clock cycle.
- For a NOP operation, no `done` signal is generated.
- For a NOP, the requester deasserts `start` one clock cycle after asserting it.

TestBench Top Connection



Coverage Plan

Plan Overview

The following tables outline our verification strategy, including Signals, Features, and Covergroups.

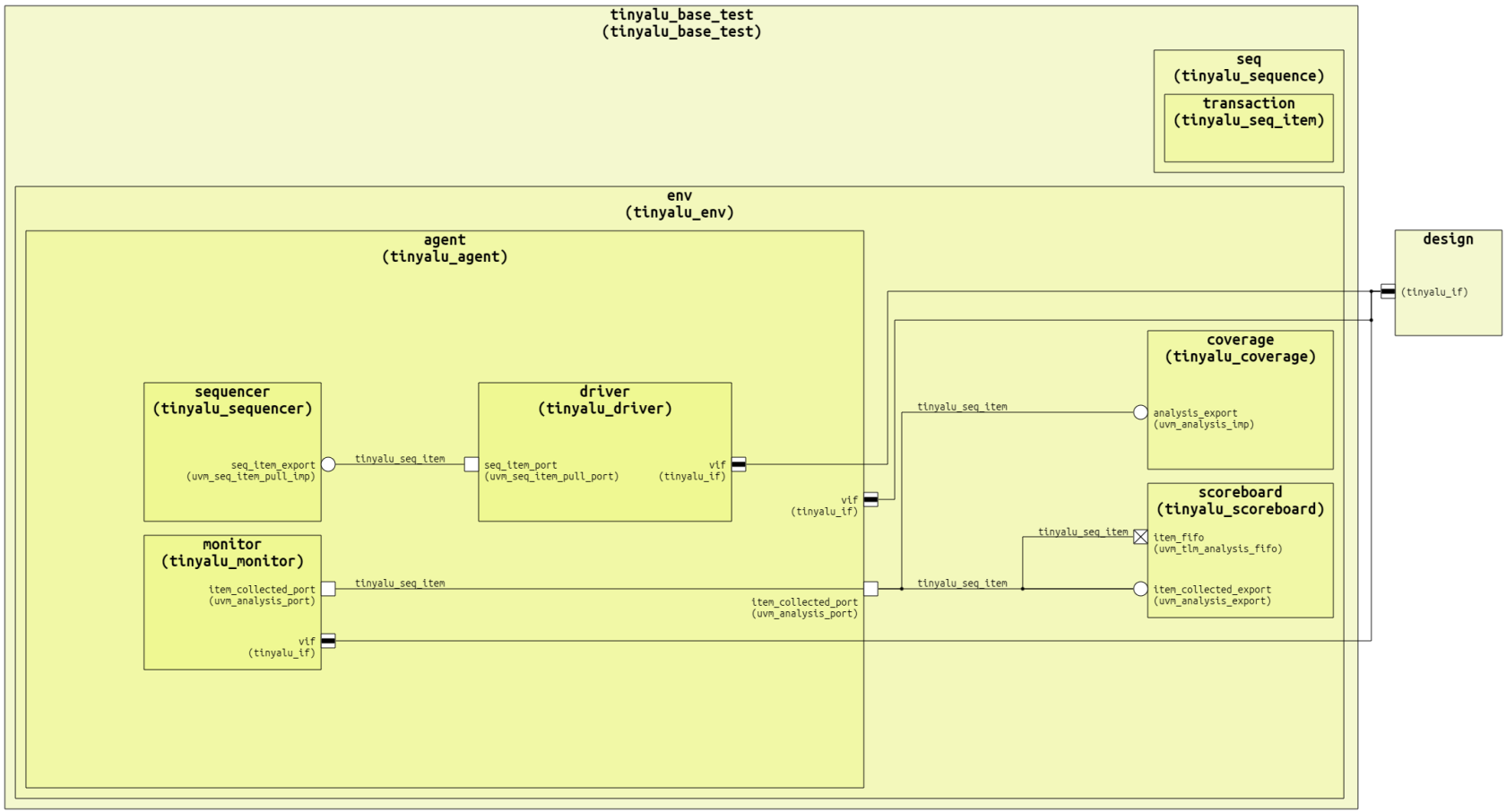
DUT Signals

Name	Direction	Width	Description	Type	Verification Note
A	Input	[7:0]	Operand A	Data	Randomized 0x00-0xFF
B	Input	[7:0]	Operand B	Data	Randomized 0x00-0xFF
op	Input	[2:0]	Operation Code	Control	Covers 5 valid, 3 invalid states
clk	Input	1	System Clock	Clock	Driven by Tb Agent
reset_n	Input	1	Active Low Reset	Reset	Synchronous assertion/deassertion
start	Input	1	Start Operation	Handshake	Pulled high to initiate transaction
done	Output	1	Operation Complete	Handshake	DUT asserts high when result is valid
result	Output	[15:0]	ALU Output	Data	Checked against Scoreboard model

Coverage Plan

ID	signals to be in the covgroup	coverage details	coverage type
cop-1	op	all operations execute	cover point
cop-2	op	single cycle operation after a 3 -cycle operation	transition coverage
cop-3	op	a 3-cycle operation after a singel cycle operation	transition coverage
cop-4	op	two consecutive 3-cycle operations in a row	transition coverage
inop-1	op / A / B	some random values for any random operation	cross
inop-2	op / A / B	try max vlaues for inputs with all operations	cross
inop-3	op / A / B	try min values for inputs with all operations	cross
inop-4	op / A / B	try max value for A with min value of B for any operation and vice versa	cross

UVM Environment Topology



Transaction Object: tinyalu_seq_item

tinyalu_seq_item.sv

```
1  class tinyalu_seq_item extends uvm_sequence_item;
2      // ...Signal Fields decleration...
3
4      // constraints
5      constraint constr_a { A dist {0:=1 , [1:254]:/8 ,
           255:=1};}
6      constraint constr_b { B dist {0:=1 , [1:254]:/8 ,
           255:=1};}
7      constraint constr_op { op dist {0:=1, [1:3]:/6,
           4:=3};}
8
9  endclass: tinyalu_seq_item
```

Transaction Object: tinyalu_sequence

tinyalu_sequence.sv

```
1  //..
2  // generation of a sequence os tinyalu_seq_item
3  virtual task body ();
4     tinyalu_seq_item transaction;
5     repeat (5000) begin
6     transaction = tinyalu_seq_item::type_id::create("
          transaction");
7     start_item (transaction); // wait until get_next_item
          from driver
8     if (! transaction.randomize() )
9        `uvm_fatal("FATAL",$sformatf(" RANDOMIZATION
             FAILED !! "))
10    finish_item (transaction); // wait until item_done
          from driver
11 end
12 //..
```

Sequence Item Details

Purpose: Defines the data packet moving through the TB.

- **Randomization:** Inputs A, B, and Op are declared as `rand`.
- **Constraints:**
  - Ensures only valid opcodes are generated.
  - Can be extended in tests to target corner cases (e.g., max values).
- **Automation:** Uses UVM field macros for easy printing and copying.

Interface: tinyalu_if

tinyalu_if.sv

```
1  interface tinyalu_if (input logic clk, reset_n);
2     logic [7:0] A, B;
3     logic [2:0] op;
4     logic start, done;
5     logic [15:0] result;
6
7     clocking cb_drv @(posedge clk);
8        output A, B, op, start;
9        input  done, result;
10    endclocking
11
12    clocking cb_mon @(posedge clk);
13       input A, B, op, start, done, result;
14    endclocking
15 endinterface
```

Interface Details

Purpose: Connects the static DUT to dynamic UVM classes.

- **Clocking Blocks:**
  - `cb_drv`: Used by Driver. Signals are outputs (drive).
  - `cb_mon`: Used by Monitor. All signals are inputs (sample).
- **Timing:** Prevents race conditions by strictly defining sampling offsets.
- **Modports:** Enforces directional access constraints.

Driver: tinyalu_driver

tinyalu_driver.sv

```
1  task run_phase(uvm_phase phase);
2    // ... Reset handling ...
3    forever begin
4      seq_item_port.get_next_item(req);
5      @(vif.cb_drv);
6      vif.cb_drv.A <= req.A; // ... Drive B, Op ...
7      vif.cb_drv.start <= 1'b1;
8
9      if (req.op == no_op) begin
10       @(vif.cb_drv); vif.cb_drv.start <= 1'b0;
11     end else begin
12       wait(vif.cb_drv.done === 1'b1);
13       vif.cb_drv.start <= 1'b0;
14     end
15     seq_item_port.item_done();
16   end
17 endtask
```

Driver Details

Purpose: Drives stimulus to the DUT pin-level interface.

- **Reset Aware:** Waits for `reset_n` to be high before driving.
- **Handshake Protocol:**
  - Asserts `start` signal.
  - For Arithmetic: Waits for `done` signal from DUT.
  - For NOP: Implements specific single-cycle pulse logic (no done signal).
- **Synchronization:** Uses `cb_drv` to remain cycle-accurate.

Monitor: tinyalu_monitor

tinyalu_monitor.sv

```
1  task run_phase(uvm_phase phase);
2    forever begin
3      // Trigger on Start
4      @(vif.cb_mon iff vif.cb_mon.start === 1);
5      trans.A = vif.cb_mon.A;
6      // ... Sample B, Op ...
7
8      // Wait for completion
9      @(vif.cb_mon iff vif.cb_mon.done === 1);
10     trans.result = vif.cb_mon.result;
11
12     // Broadcast
13     item_collected_port.write(trans);
14   end
15 endtask
```

Monitor Details

Purpose: Passively observes the interface and creates transactions.

- **Passive:** Does not drive any signals (uses `cb_mon`).
- **Reconstruction:** Converts pin wiggles back into a `tinyalu_seq_item`.
- **Analysis Port:** Broadcasts the captured transaction to:
  1. Scoreboard (for checking)
  2. Coverage Collector (for metrics)

Agent: tinyalu_agent

tinyalu_agent.sv

```
1  class tinyalu_agent extends uvm_agent;
2  // Connect Phase
3  function void connect_phase(uvm_phase phase);
4      super.connect_phase(phase);
5      //  Distribute Interface from Config to components
6      monitor.vif = vif;
7      driver.vif = vif;
8      // Connect Driver's request port to Sequencer's
           export
9      driver.seq_item_port.connect(sequencer.
           seq_item_export);
10     // Connect Monitor Port to Agent Port
11     monitor.item_collected_port.connect(this.
           item_collected_port);
12 endfunction
13 endclass: tinyalu_agent
```

Agent Details

Purpose: Container for a specific interface's components.

- **Build Logic:** Creates the Monitor, Driver and Sequencer.
- **Connection:** Connects the Driver's request port to the Sequencer.

Scoreboard: tinyalu_scoreboard

tinyalu_scoreboard.sv

```
1  uvm_tlm_analysis_fifo #(tinyalu_seq_item) item_fifo;
2
3  task run_phase(uvm_phase phase);
4    forever begin
5      item_fifo.get(item); // Blocking Get
6      expected = predict(item); // Golden Model
7
8      if (item.result !== expected)
9          `uvm_error("SCBD", "Mismatch!")
10     else
11         `uvm_info("SCBD", "Match", UVM_HIGH)
12   end
13 endtask
```

Scoreboard Details

Purpose: Verifies the correctness of the DUT output.

- **TLM FIFO:** Buffers incoming transactions from the monitor to decouple timing.
- **Predictor:** Implements a behavioral ”Golden Model” of the ALU (using standard SV operators `+`, `*`, `&`, `^`).
- **Comparison:** Checks `Actual vs. Expected`. Reports UVM Errors on mismatch.

Environment: tinyalu_env

tinyalu_env.sv

```
1  function void build_phase(uvm_phase phase);
2     agent = tinyalu_agent::type_id::create("agent", this)
          ;
3     scbd  = tinyalu_scoreboard::type_id::create("scbd",
          this);
4     cov   = tinyalu_coverage::type_id::create("cov", this
          );
5  endfunction
6
7  function void connect_phase(uvm_phase phase);
8     agent.item_collected_port.connect(scbd.
          item_collected_export);
9     agent.item_collected_port.connect(cov.analysis_export
          );
10 endfunction
```

Environment Details

Purpose: Top-level container for the verification components.

- **Instantiation:** Builds the Agent, Scoreboard, and Coverage.
- **Wiring:** Connects the Analysis Ports:
  - Agent $\rightarrow$ Scoreboard
  - Agent $\rightarrow$ Coverage
- Allows the Test to interact with the entire hierarchy as one unit.

Test: tinyalu_base_test

tinyalu_test.sv

```
1  task run_phase(uvm_phase phase);
2      random_op_seq seq;
3      phase.raise_objection(this);
4      `uvm_info("TEST", "Starting Random Sequence...",
           UVM_LOW)
5      seq = tinyalu_sequence::type_id::create("seq");
6      seq.start(env.agent.sequencer);
7
8      phase.drop_objection(this);
9  endtask
```

Test Details

Purpose: Controls the simulation execution.

- **Build:** Configures the environment and the virtual interface.
- **Objections:** Manages `raise_objection` and `drop_objection` to keep the simulator running.
- **Sequences:** Selects and starts specific sequences (e.g., Random, Sanity, Corner Case) on the sequencer.

Coverage: tinyalu_coverage

tinyalu_coverage.sv

```
1  class tinyalu_coverage extends uvm_subscriber #(
       tinyalu_seq_item);
2    //regestration
3    `uvm_component_utils(tinyalu_coverage)
4
5    //variables
6    bit [7 : 0] A ;
7    bit [7 : 0] B ;
8    op_t op ;
9
10     //...Coverage Groups...
```

TB Top: tinyalu_top

tinyalu_top.sv

```
1  module tinyalu_top;
2    bit clk, reset_n;
3    // ... Clock Gen logic ...
4
5    tinyalu_if intf(clk, reset_n); // Interface
6
7    tinyalu DUT ( .A(intf.A), ... ); // DUT
8
9    initial begin
10     uvm_config_db#(virtual tinyalu_if)::set(null,"*","
           vif",intf);
11     run_test();
12   end
13 endmodule
```

Top Module Details

Purpose: The hardware wrapper for the simulation.

- **Clock/Reset:** Generates the physical clock and reset signals.
- **Instantiation:** Instantiates the Interface and the DUT, connecting them.
- **Registration:** Puts the Virtual Interface into the `uvm_config_db` so UVM classes can access pins.
- **Start:** Calls `run_test()` to kick off the UVM phases.

Simulation Output: UVM Report Summary

UVM Report Summary

```
# UVM_INFO :    7
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [SCBD]     1
# [TEST]     1
# [TEST_DONE]     1
# [UVMTOP]     1
```

Simulation Results: Coverage Report

Name	Class Type	Coverage	Goal	% of Goal	Status	Included
/tinyalu_pkg/tinyalu_coverage		100.00%				
TYPE op_cov		100.00%	100	100.00...		✓
CVP op_cov::op_set		100.00%	100	100.00...		✓
INST \/tinyalu_pkg::tinyalu_coverage::op_cov		100.00%	100	100.00...		✓
CVP op_set		100.00%	100	100.00...		✓
ignore_bin others		0	-	-		✓
bin s[no_op]		428	1	100.00...		✓
bin s[add_op]		557	1	100.00...		✓
bin s[and_op]		521	1	100.00...		✓
bin s[xor_op]		511	1	100.00...		✓
bin m		821	1	100.00...		✓
bin m_to_s		558	1	100.00...		✓
bin s_to_m		559	1	100.00...		✓
bin m_to_m		262	1	100.00...		✓
TYPE in_on_all_op_cov		100.00%	100	100.00...		✓
CVP in_on_all_op_cov::all_op		100.00%	100	100.00...		✓
bin Add		557	1	100.00...		✓
bin And		521	1	100.00...		✓
bin Xor		511	1	100.00...		✓
bin Mul		821	1	100.00...		✓
CVP in_on_all_op_cov::in_a		100.00%	100	100.00...		✓
bin all_zeros		279	1	100.00...		✓
bin all_ones		294	1	100.00...		✓
bin random		2265	1	100.00...		✓
CVP in_on_all_op_cov::in_b		100.00%	100	100.00...		✓
bin all_zeros		286	1	100.00...		✓
bin all_ones		277	1	100.00...		✓
bin random		2275	1	100.00...		✓
CROSS in_on_all_op_cov::max_min_on_all_op		100.00%	100	100.00...		✓
bin add_all_zeros		4	1	100.00...		✓
bin add_all_ones		8	1	100.00...		✓
bin and_all_zeros		3	1	100.00...		✓
bin and_all_ones		5	1	100.00...		✓
bin xor_all_zeros		5	1	100.00...		✓
bin xor_all_ones		5	1	100.00...		✓
bin mul_all_zeros		4	1	100.00...		✓
bin mul_all_ones		21	1	100.00...		✓
bin min_max_case		56	1	100.00...		✓
bin random_values		2317	1	100.00...		✓