UVM ADDER

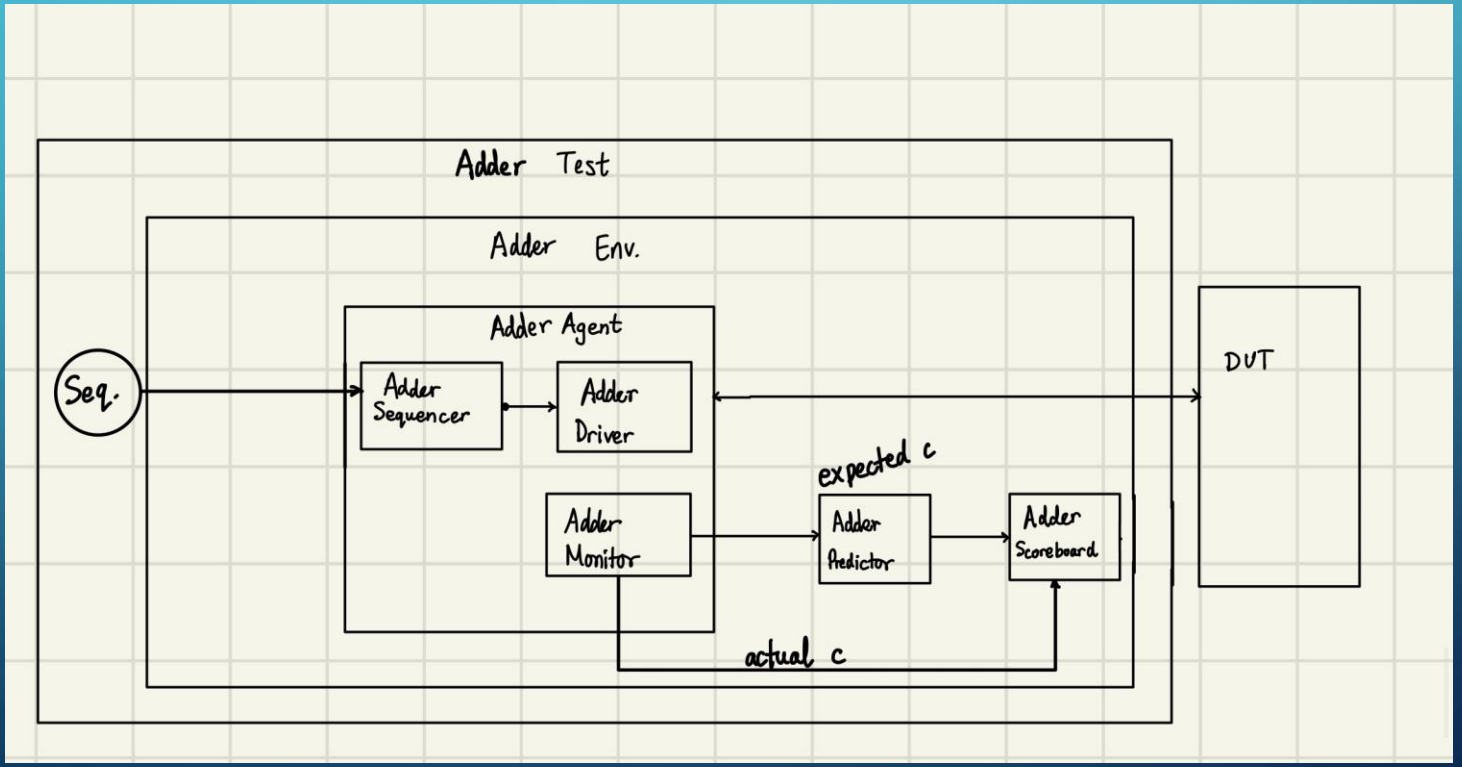
TEAM MENTOR: DR. JOHNSON

GRADUATE MENTOR: SARANG PRAMOD

TEAM MEMBERS: PRANAY JAGGI, AVANISH KARLAPUDI

UVM FOR ADDER

- Adder: combinational logic circuit that performs the addition operation
- Architecture for Adder:



UVM ADDER TESTBENCH COMPONENTS

- Top: top-level environment for adder DUT
- Test: defines testcases to verify addition operation of adder
- Environment: responsible for coordinating UVM adder components in verification environment
- Agent: drives inputs to DUT and captures output
- Driver: responsible for driving inputs to DUT
- Monitor: responsible for capturing inputs and outputs of DUT and sending to Scoreboard
- Sequence: randomizes two values a and b and sends to the Driver
- Scoreboard: compares expected output from Predictor with actual output from Monitor

SNIPPETS/STATUS

```
UVM_INFO @ 1150: uvm_test_top.env.comp [Comparator] Data match
UVM_INFO @ 1210: uvm_test_top.env.comp [Comparator]
Expected:
result_sum:  9
result_overflow: 1
----------
Actual:
sum:  9
overflow: 1
```

```
UVM_INFO @ 1450: uvm_test_top.env.comp [Comparator] Data match
UVM_INFO @ 1510: uvm_test_top.env.comp [Comparator]
Expected:
result_sum: 10
result_overflow: 0
----------
Actual:
sum: 10
overflow: 0
```

```
task run_phase(uvm_phase phase);
  transaction req_item;
  vif.check = 0;

  forever begin
    seq_item_port.get_next_item(req_item);
    DUT_reset();
    vif.a = req_item.a;
      vif.b = req_item.b;
      vif.carry_in = req_item.carry_in;
      #(0.2)
      @(posedge vif.clk);
      seq_item_port.item_done();
  end
endtask: run_phase
```

Driver

```
function void build_phase(uvm_phase phase);
  super.build_phase(phase);

  if( !uvm_config_db#(virtual adder_if)::get(this, "", "adder_vif", vif) ) begin

    `uvm_fatal("Driver", "No virtual interface specified for this test instance");
              end
endfunction: build_phase
```

Driver

SNIPPETS/STATUS

```
class adder_sequence extends uvm_sequence #(transaction);
  `uvm_object_utils(adder_sequence)
  function new(string name = "");
    super.new(name);
  endfunction: new

  task body();
    transaction req_item;
    req_item = transaction#(4)::type_id::create("req_item");

    repeat(25) begin
      start_item(req_item);
      if(!req_item.randomize()) begin

        `uvm_fatal("sequence", "not able to randomize")
      end
      finish_item(req_item);
    end
  endtask: body
endclass
```

Sequence

```
class agent extends uvm_agent;
  `uvm_component_utils(agent)
  sequencer sqr;
  driver drv;
  monitor mon;

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

  virtual function void build_phase(uvm_phase phase);
    sqr = sequencer::type_id::create("sqr", this);
    drv = driver::type_id::create("drv", this);
    mon = monitor::type_id::create("mon", this);
  endfunction

  virtual function void connect_phase(uvm_phase phase);
    drv.seq_item_port.connect(sqr.seq_item_export);
  endfunction

endclass: agent
```

Agent

UVM APPLICATIONS

UVM is widely used in the semiconductor industry for verifying complex hardware designs.

Its main application is simply testing various designs and verifying that they perform as expected.