Agenda: Day 2

DAY
2

- 5 UVM Configuration & Factory
- 6 UVM Component Communication 💻
- 7 UVM Scoreboard & Coverage
- 8 UVM Callback 💻

Unit Objectives

After completing this unit, you should be able to:

- Build re-usable self checking scoreboards by using the in-built UVM comparator classes
- Implement functional coverage

Scoreboard - Introduction

- **Today's challenges**
  - Self checking testbenches need scoreboards
  - Develop a scoreboard once, re-use many times in different testbenches
  - Need different scoreboarding mechanisms for different applications
  - Must be aware of DUT's data transformation

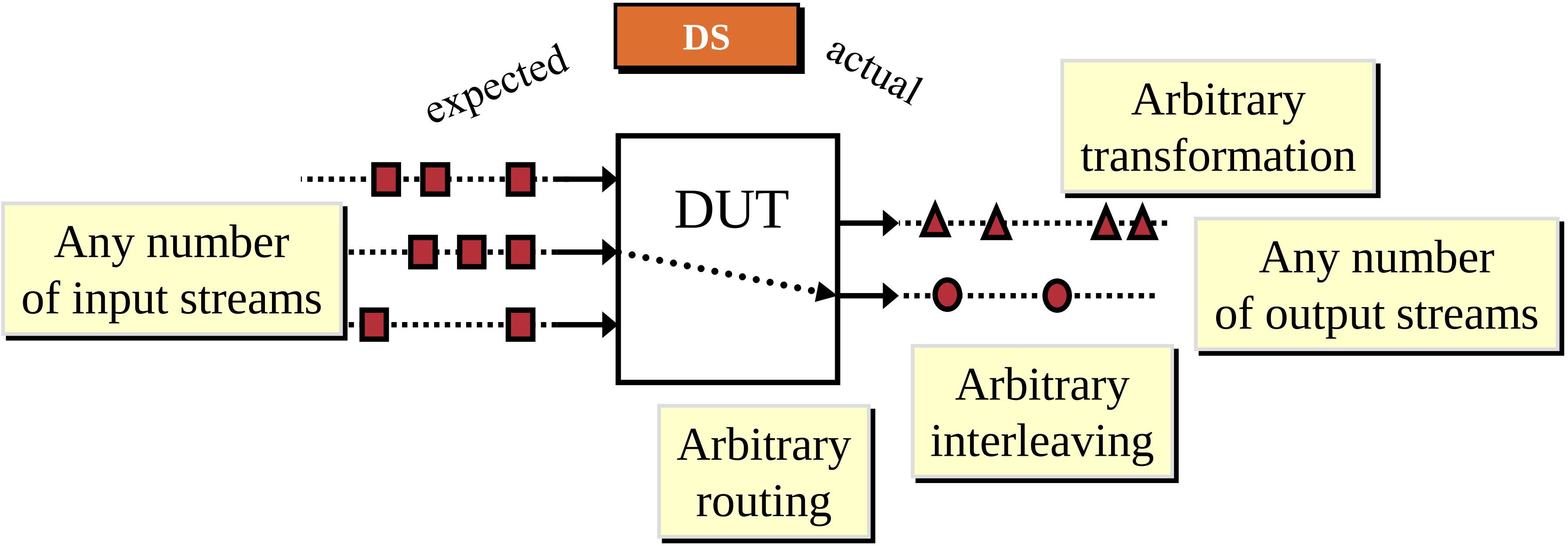
Solution: UVM scoreboard class extension with tailored functions for matching expected & observed data:

- In-order expects
- Data transformation
- Built in Analysis Exports in the Comparator classes

Scoreboard – Data Streams

Any ordered data sequence.   Not just packets.

Application	Streams
Networking	Packets in, packets out
DSP	Samples in, samples out
Modems, codecs	Frames in, code samples out
Busses, controllers	Requests in, responses out

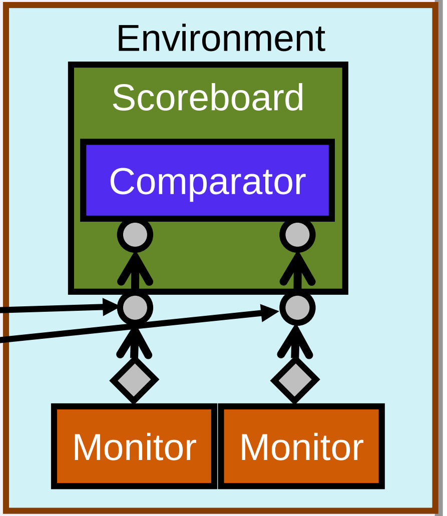


Scoreboard Implementation

- Use `uvm_in_order_class_comparator` for checking

```
class scoreboard extends uvm_scoreboard; // utils macro and constructor left off
  typedef uvm_in_order_class_comparator #(packet) cmpr_t;
  cmpr_t cmpr;
  uvm_analysis_export #(packet) before_export;
  uvm_analysis_export #(packet) after_export;
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    cmpr = cmpr_t::type_id::create("cmpr", this);
    before_export = new("before_export", this);
    after_export = new("after_export", this);
  endfunction
  virtual function void connect_phase(uvm_phase phase);
    before_export.connect(cmpr.before_export);
    after_export.connect(cmpr.after_export);
  endfunction
  virtual function void report_phase(uvm_phase phase);
    `uvm_info("Scoreboard Report",
          $sformatf("Matches = %0d, Mismatches = %0d",
          cmpr.m_matches, cmpr.m_mismatches), UVM_MEDIUM);
  endfunction
endclass
```

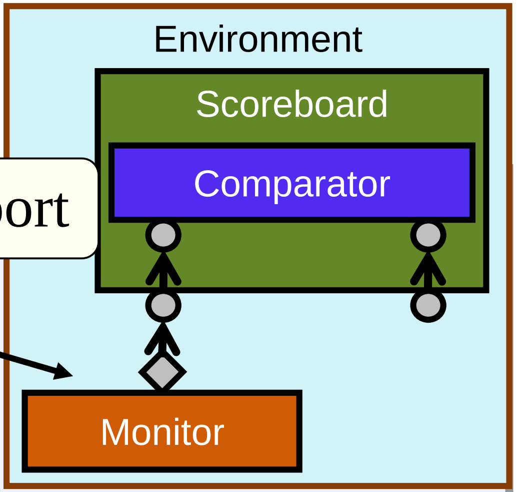
(See note)



Pass-through

Scoreboarding: Monitor

- **Monitors supplies scoreboard with expected and actual transactions**



```
class iMonitor extends uvm_monitor;
  virtual router_io vif;
  uvm_analysis_port #(packet) analysis_port;
  // uvm_component_utils macro and constructor
  virtual function void build_phase(…); ...
    analysis_port = new("analysis_port", this);
    if (!uvm_config_db#(virtual router_io)::get(this,"","vif",vif))
      `uvm_fatal("CFGERR", …);
  endfunction
  virtual task run_phase(uvm_phase phase);
    forever begin
      packet tr = packet::type_id::create("tr");
      get_packet(tr);
      analysis_port.write(tr);
    end
  endtask
  virtual task get_packet(packet tr); ...
endclass
```

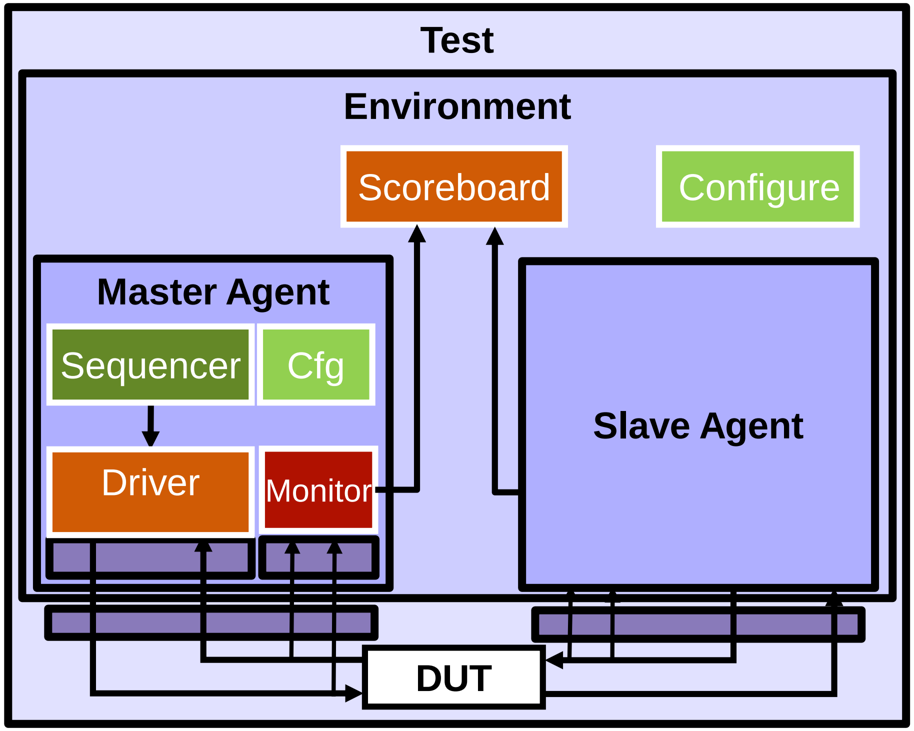
Embed analysis port

Get DUT interface

Pass observed transaction to collector components via TLM analysis port

Embed Monitor in Agent

- **Agent extends from `uvm_agent` class**
  - Contains a driver, a sequencer and a monitor
  - Contains configuration and other parameters
- **Two operating modes:**
  - Active:
    - Emulates a device in the system interacting with DUT
    - Instantiates a driver, sequencer and monitor
  - Passive:
    - Operates passively
    - Only monitor instantiated and configured



UVM Agent Example

```
class master_agent extends uvm_agent;
  uvm_analysis_port #(packet) analysis_port;
  // utils macro and constructor not shown
  sequencer sqr;
  driver    drv;
  iMonitor  mon;
  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    analysis_port = new("analysis_port", this);
    if (is_active == UVM_ACTIVE) begin
      sqr = packet_sequencer::type_id::create("sqr",this);
      drv  = driver::type_id::create("drv",this);
    end
    mon    = iMonitor::type_id::create("mon",this);
  endfunction: build_phase
  function void connect_phase(uvm_phase phase);
    mon.analysis_port.connect(this.analysis_port);
    if(is_active == UVM_ACTIVE)
      drv.seq_item_port.connect(sqr.seq_item_export);
  endfunction: connect_phase
endclass
```

Sequencer, Driver and Monitor

is_active flag is built-in

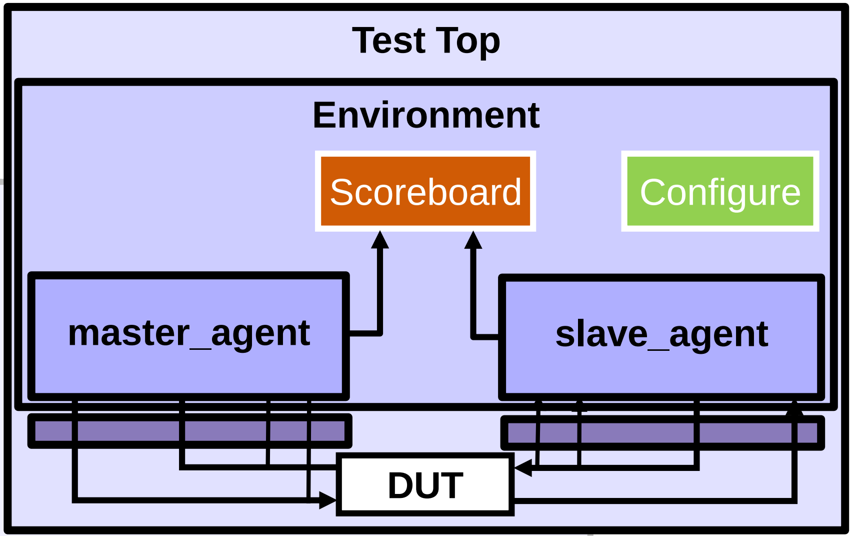
Create sequencer and driver if active

Pass-through

Connect sequencer to driver

Using UVM Agent in Environment

- **Agent simplifies environment**
  - Easier to maintain and debug



```
class router_env extends uvm_env;
  master_agent m_agt;
  slave_agent  s_agt;
  scoreboard   sb;
  // utils and constructor not shown
  virtual function void build_phase(…);
    super.build_phase(phase);
    m_agt = master_agent::type_id::create("m_agt", this);
    s_agt = slave_agent::type_id::create("s_agt", this);
    sb = scoreboard::type_id::create("sb", this);
    uvm_config_db#(uvm_active_passive_enum)::set(this, "m_agt",
                                      "is_active", UVM_ACTIVE);
    uvm_config_db#(uvm_active_passive_enum)::set(this, "s_agt",
                                      "is_active", UVM_ACTIVE);
  endfunction
  virtual function void connect_phase(uvm_phase phase);
    m_agt.analysis_port.connect(sb.before_export);
    s_agt.analysis_port.connect(sb.after_export);
  endfunction
endclass
```

Scoreboard Can Be Parameterized

- **Create a base class without parameter**

```
virtual class scoreboard_base extends uvm_scoreboard;
  pure virtual task wait_for_expected_q_empty();
       virtual function void clear_expected_q(); end function
endclass
```

Embed common methods

- **Then create parameterized class with required members**
  - Including **uvm_component_param** macro, **type_name** string and **get_type_name()** method

```
class packet_sb #(type T = packet) extends scoreboard_base;
 typedef packet_sb #(T) this_type;
 `uvm_component_param_utils(this_type)
 const static string type_name = $sformatf("packet_sb#(%s)", T::type_id::type_name);
 virtual function string get_type_name();
  return type_name;
 endfunction
 virtual task wait_for_expected_q_empty(); … endtask
 virtual function void clear_expected_q(); … endfunction

endclass
```

Scoreboard: User Implementation (1/2)

- **User can implement out-of-order scoreboard**
  - Need a transaction queue to store in-coming transactions

```
class scoreboard #(type before = uvm_object, after = before)
                 extends scoreboard_base;
  typedef scoreboard #(before, after) this_type;
  // required methods left off - see note
  `uvm_analysis_imp_decl(_before)
  `uvm_analysis_imp_decl(_after)
  uvm_analysis_imp_before #(before, this_type) before_export;
  uvm_analysis_imp_after  #(after, this_type) after_export;
  int m_matches = 0, m_mismatches = 0;
  after expected[$];
  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction
  virtual function void build_phase(uvm_phase phase);
    super.build_phase();
    before_export = new("before_export", this);
    after_export  = new("after_export", this);
  endfunction
// continued on next slide
```

Scoreboard: User Implementation (2/2)

- **Narrow down potential matches with a tag search**
  - Otherwise, performance may be an issue

```
// continued from previous slide
  virtual function void write_before(before b_tr);
    after a_tr;
    // transform here
    expected.push_back(a_tr);
  endfunction
  virtual function void write_after(after a_tr);
    int index[$];
    index = expected.find_index() with (item.find(a_tr));
    foreach(index[i]) begin
      if (pkt.compare(expected[index[i]])) begin
        expected.delete(index[i]);
        m_matches++;
        return;
      end
    end
    `uvm_warning("SB_ERR", "No match found");
    m_mismatches++;
  endfunction
endclass
```

Do a quick search first

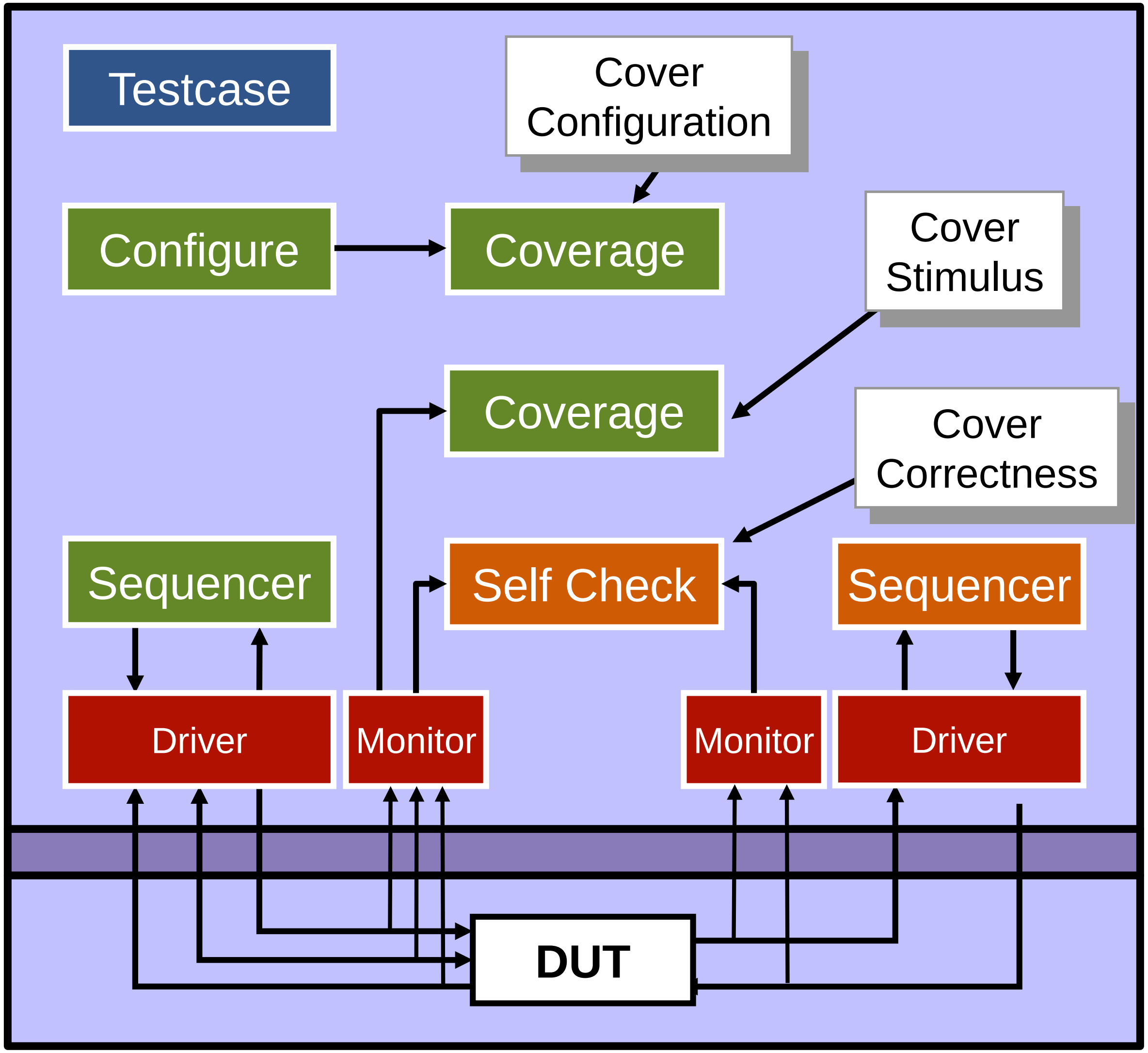
Do full compare on results of quick search

Functional Coverage

- **Measure the random stimulus to track progress towards verification goal**
- **What to measure?**
  - Configuration: Has testbench tried all legal environment possibilities?
    - N drivers, M Slaves, bus addresses, etc.
  - Stimulus: Has testbench generated all representative transactions, including errors?
    - Reads, writes, interrupts, long packets, short bursts, overlapping operations
  - Correctness: Has DUT responded correctly to the stimulus?
    - Reads, writes, interrupts, long packets, short bursts, overlapping operations

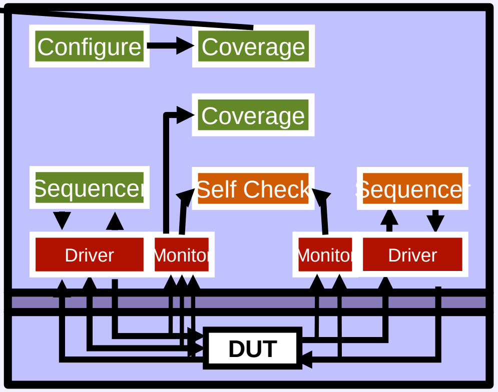
Connecting Coverage to Testbench

- **SystemVerilog Testbench Structure**



Component Configuration Coverage (1/2)

```
covergroup cfg_cg() with function sample(env_cfg cfg); … endgroup
class config_coverage extends uvm_component;
  bit coverage_enable = 0;
  env_cfg cfg; cfg_cg cg;
  `uvm_component_utils_begin(config_coverage)
    `uvm_field_object(cfg, UVM_DEFAULT)
    `uvm_field_int(coverage_enable, UVM_DEFAULT)
  `uvm_component_utils_end
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    uvm_config_db#(int)::get(this,"","coverage_enable",coverage_enable);
    if (coverage_enable) begin
      if (!uvm_config_db #(env_cfg)::get(this, "", "cfg", cfg)) begin
       `uvm_fatal(…);
      end
      cg = new();
    end
  endfunction
  virtual function void start_of_simulation_phase(uvm_phase phase);
    if (coverage_enable)
      cg.sample(cfg);
  endfunction
endclass
```



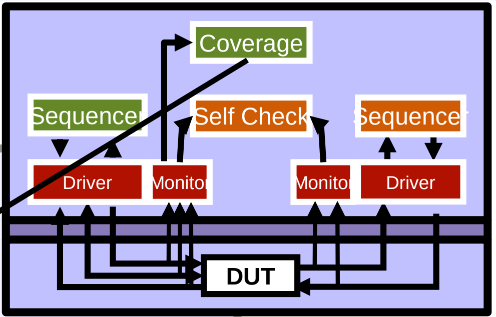
Component Configuration Coverage (2/2)

- **Build configuration coverage component in test**

```
class test_ports extends test_base; // utils and constructor not shown
  env_cfg cfg;
  config_coverage cfg_cov;

  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    cfg = env_cfg::type_id::create("cfg", this);
    if (!cfg.randomize()) begin
      `uvm_fatal(…);
    end

    cfg_cov = config_coverage::type_id::create("cfg_cov", this);

    uvm_config_db #(env_cfg)::set(this, "env", "cfg", cfg);
    uvm_config_db #(env_cfg)::set(this, "cfg_cov", "cfg", cfg);
    uvm_config_db #(int)::set(this, "cfg_cov", "coverage_enable", 1);
  endfunction
endclass
```

Stimulus Coverage

- **Cover monitor's observed transactions**



```
covergroup pkt_cg with function sample(packet pkt);
  coverpoint pkt.sa;
endgroup: pkt_cg
class packet_coverage extends uvm_subscriber #(packet); ...;
  pkt_cg cov; bit coverage_enable;
  virtual function void build_phase(uvm_phase phase); ...;
    if (coverage_enable) cov = new();
  endfunction
  virtual function void write(T t);
    if (coverage_enable) cov.sample(t);
  endfunction
e
```

Class with built-in analysis port

Sample method called with monitored packet

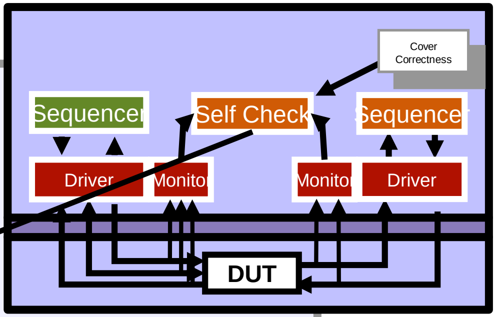
```
class test_stimulus_coverage extends test_base; ...;
  packet_coverage cov_comp;
  virtual function void build_phase(uvm_phase phase); ...;
    cov_comp = packet_coverage::type_id::create("cov_comp", this);
  endfunction
  virtual function void connect_phase(uvm_phase phase); ...;
    env.agt.analysis_port.connect(cov_comp.analysis_export);
  endfunction
endclass
```

Correctness Coverage

- **Cover verified transaction in scoreboard**

Packet cover group

```
covergroup sb_pkt_cg with function sample(packet
  coverpoint pkt.sa;
  coverpoint pkt.da;
  cross pkt.sa, pkt.da;
endgroup: sb_pkt_cg

class scoreboard #(type T = packet) extends scoreboard_base;
  // component_utils and other code not shown
  bit coverage_enable = 0;

  virtual function void write_after(T pkt);
    if (pkt.compare(pkt_ref) begin
      m_matches++;
      if (coverage_enable) sb_pkt_cg.sample(pkt_ref);
    end else begin
      m_mismatches++;
    end
  endfunction
endclass
```



TLM imp method called with monitored packets

Unit Objectives Review

Having completed this unit, you should be able to:

- **Build re-usable self checking scoreboards by using the in-built UVM comparator classes**
- **Implement functional coverage**

Appendix

Multi-Stream Scoreboard

Scoreboard: Multi-Stream

```
class scoreboard extends uvm_scoreboard;
  uvm_analysis_imp_before #(packet, scoreboard) before_export;
  uvm_analysis_imp_after  #(packet, scoreboard) after_export;
  typedef uvm_in_order_class_comparator #(packet) cmpr_t;
  cmpr_t cmpr[16];
  `uvm_component_utils(scoreboard)
  function new(string name, uvm_component parent);
    super.new(name, parent);
    `uvm_info("TRACE", $sformatf("%m"), UVM_HIGH);
  endfunction
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    `uvm_info("TRACE", $sformatf("%m"), UVM_HIGH);
    before_export = new("before_export", this);
    after_export  = new("after_export", this);
    for (int i=0; i < 16; i++) begin
      cmpr[i] = cmpr_t::type_id::create($sformatf("cmpr_%0d", i), this);
    end
  endfunction

...  // Continued on next page
```

Scoreboard: Multi-Stream

```
  virtual function void write_before(packet pkt);
    `uvm_info("TRACE", $sformatf("%m"), UVM_HIGH);
    cmpr[pkt.da].before_export.write(pkt);
  endfunction
  virtual function void write_after(packet pkt);
    `uvm_info("TRACE", $sformatf("%m"), UVM_HIGH);
    cmpr[pkt.da].after_export.write(pkt);
  endfunction
  virtual function void report();
    `uvm_info("TRACE", $sformatf("%m"), UVM_HIGH);
    foreach (cmpr[i]) begin
      `uvm_info("Scoreboard_Report",
        $sformatf("Comparator[%0d] Matches = %0d, Mismatches = %0d",
          i, cmpr[i].m_matches, cmpr[i].m_mismatches), UVM_MEDIUM);
    end
  endfunction
endclass
```