OVM/UVM Scoreboards
Fundamental Architectures

Clifford E. Cummings

Sunburst Design, Inc.

cliffc@sunburst-design.com

www.sunburst-design.com

World-class Verilog, SystemVerilog & OVM/UVM Training

Life is too short for bad or boring training!

OVM/UVM Survey

Experience Level

What you can expect:

- Who is new to OVM/UVM?

  Definition: Never used OVM/UVM -or- just started using OVM/UVM?

  Get an idea of what to expect *(Future reference material)*

- Who has some OVM/UVM experience?

  Definition: Has used OVM/UVM for some testing or dabbling?

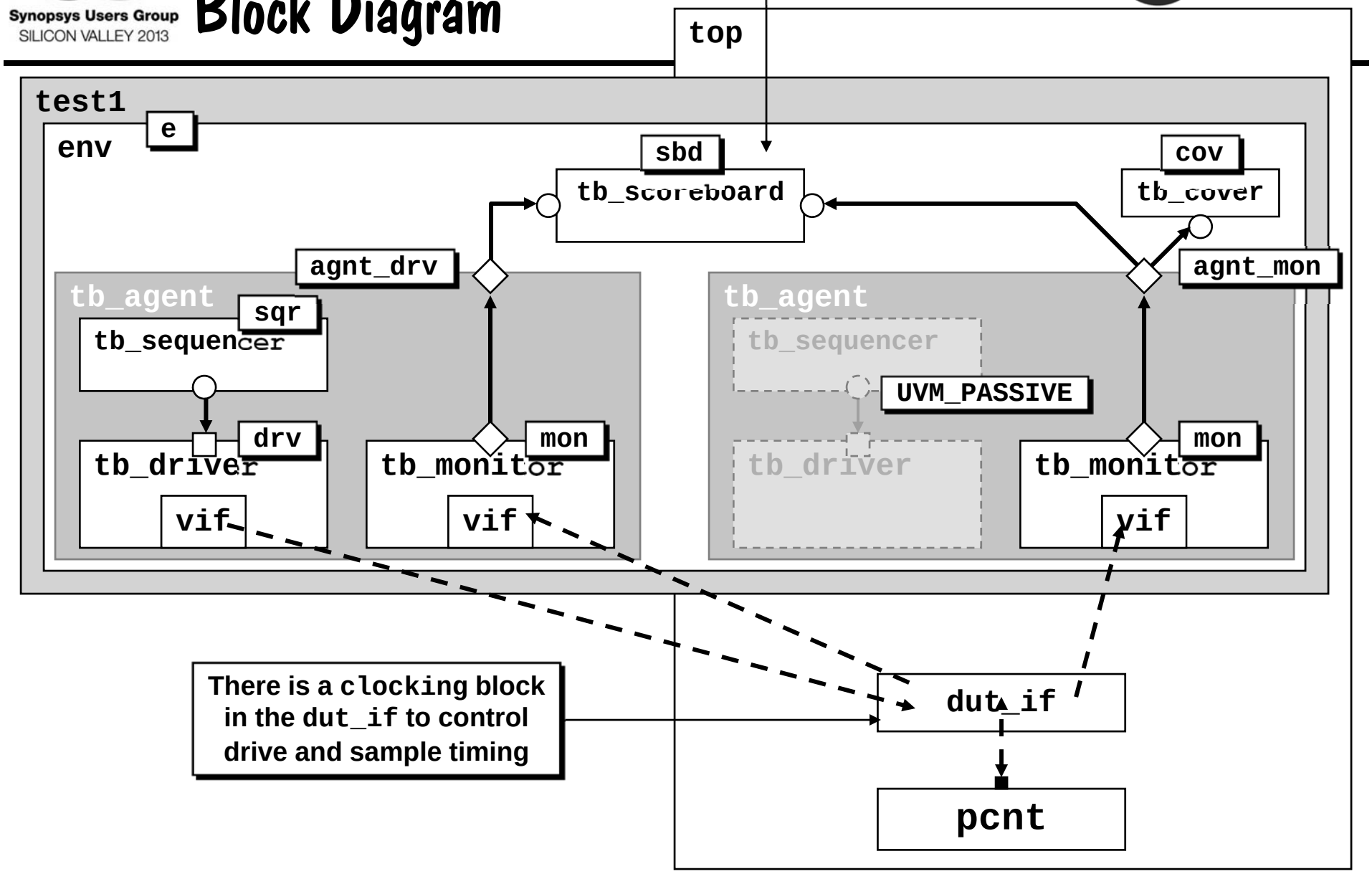
  Learn important techniques not concisely detailed elsewhere

- Who is an OVM/UVM expert?

  Definition: Very experienced OVM/UVM user or expert

  Might learn a few new tricks

Testbench Structure Block Diagram



Introduction

- Simple tutorials on UVM scoreboards are scarce

- The paper covers two *fundamental* scoreboard architectures

Learn to walk before you run !!

Understanding the fundamental architectures can help engineers expand the concepts to advanced scoreboard architectures

- This presentation covers the two most important topics from the paper:
  - Using multiple **`uvm_analysis_imp`** ports on the same component
  - Scoreboard with predictor, comparator & extern **`sb_calc_exp`** function

Most of the scoreboard is fully coded

More information and details in the paper

snug Synopsys Users Group SILICON VALLEY 2013

Scoreboard

What is its job?

Sunburst Design

DUT & scoreboard input-side

- Take sampled transaction from **`tb_monitor`**

Typically on `posedge clk`

Inputs sampled on active `clk` edge
(sampled outputs are ignored)

- Use sampled inputs to predict output

Hardest part of the `tb_scoreboard`

DUT & scoreboard output-side

- Take sampled transaction from **`tb_monitor`**

Outputs sampled `#1step` before active `clk` edge
(sampled inputs are ignored)

- Compare predicted output to actual output

Easy - *if* transaction includes properly coded `compare()` method

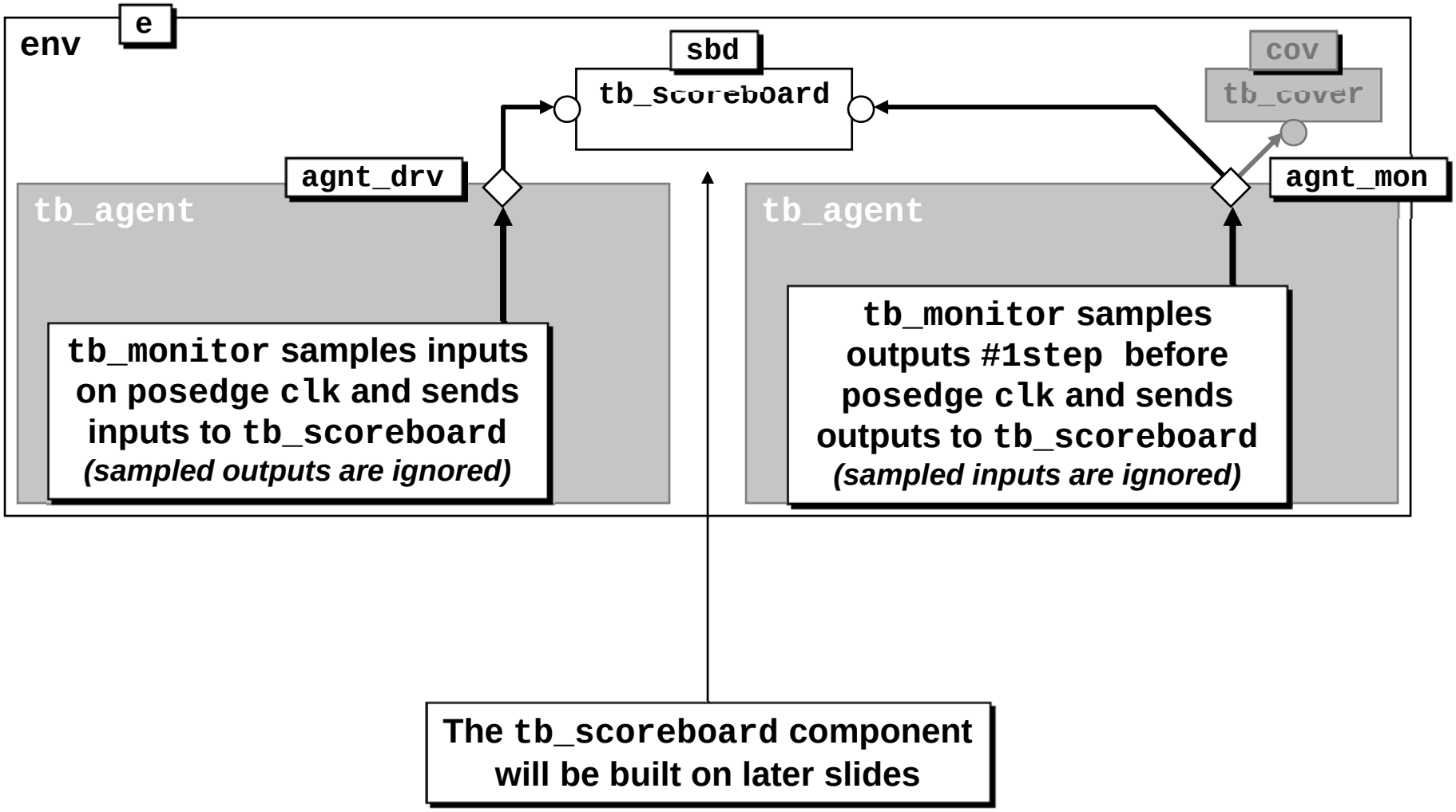
- Update PASS/ERROR counts and report detected errors

Keep track of statistics and report results

snug
Synopsys Users Group
SILICON VALLEY 2013

Scoreboard Connections Block Diagram

Sunburst Design



Fundamental Scoreboard Architectures

- 1st scoreboard architecture:
  - Single file
  - Has some complexities

`tb_scoreboard`

Commonly shown style

- 2nd scoreboard architecture:
  - Partitioned into predictor and comparator
  - Most blocks are pre-coded

`tb_scoreboard`
`sb_predictor`
`sb_comparator`

- External function to calculate expected value

`sb_calc_exp`

Called from the `sb_predictor`

This is the only block that requires user modification

Scoreboard Architecture #1

Includes:

- 2 uvm_analysis_imp ports
- 2 uvm_tlm_fifos

Scoreboard Architecture #1

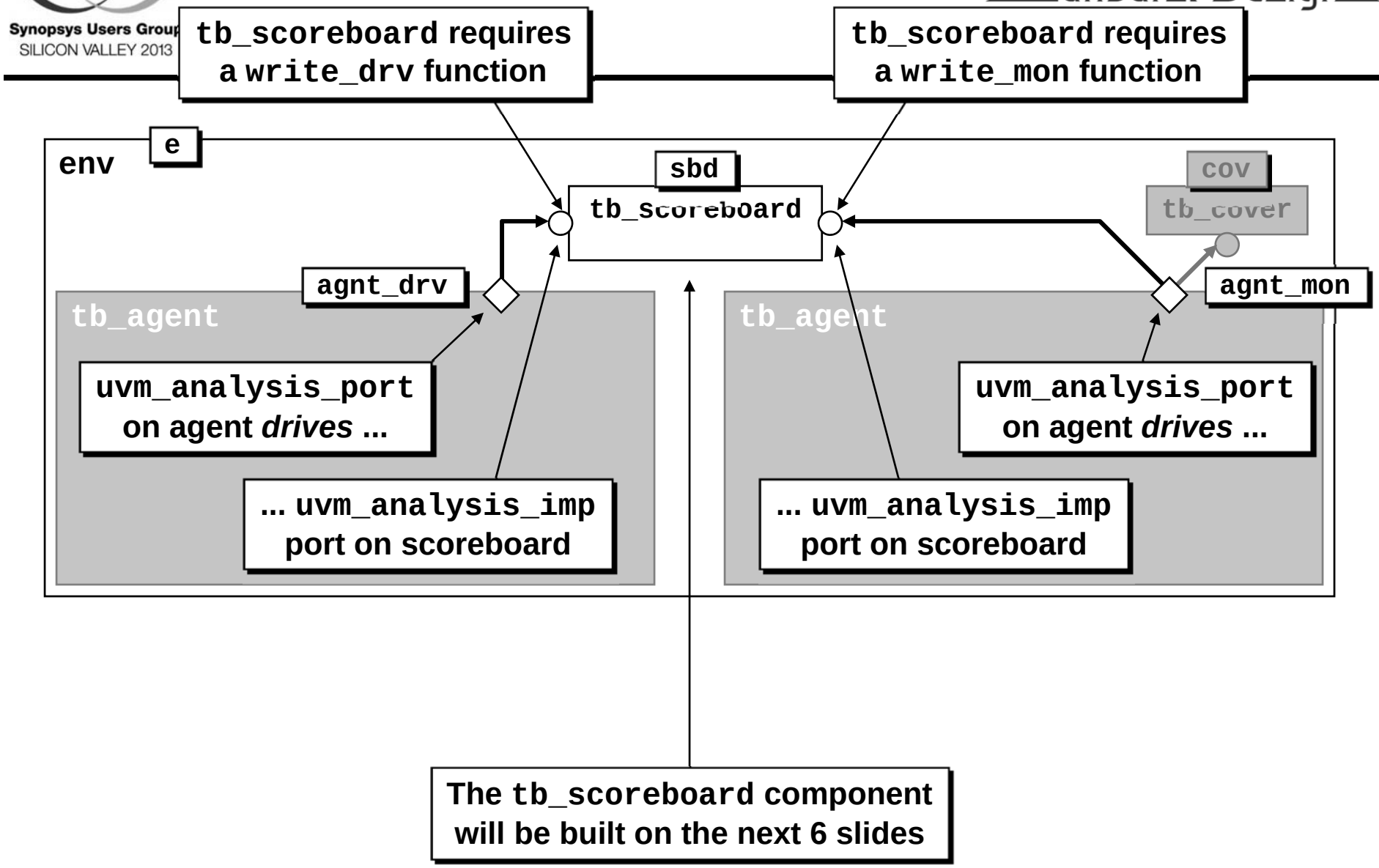
- Single-class scoreboard features:
  - Scoreboard is coded as a single file ← **Somewhat complex file**
  - Two **`uvm_tlm_fifo`**s ← **To queue *expected* and *actual* output values**
  - Two **`uvm_analysis_imp`** ports ← **This causes some issues**

Each `uvm_analysis_imp` port requires a `write()` function to capture broadcast transactions ← **Broadcast from a `uvm_analysis_port`**

PROBLEM: only allowed to have one `write()` function per class

SOLUTION: use `` `uvm_analysis_imp_decl( _*suffix* ) `` macros to create uniquely named `uvm_analysis_imp` ports with corresponding `write()` methods

Scoreboard Connections Block Diagram



Multiple Analysis Implementation Ports

Overview

These will become required <*_suffix*> names for ports and methods

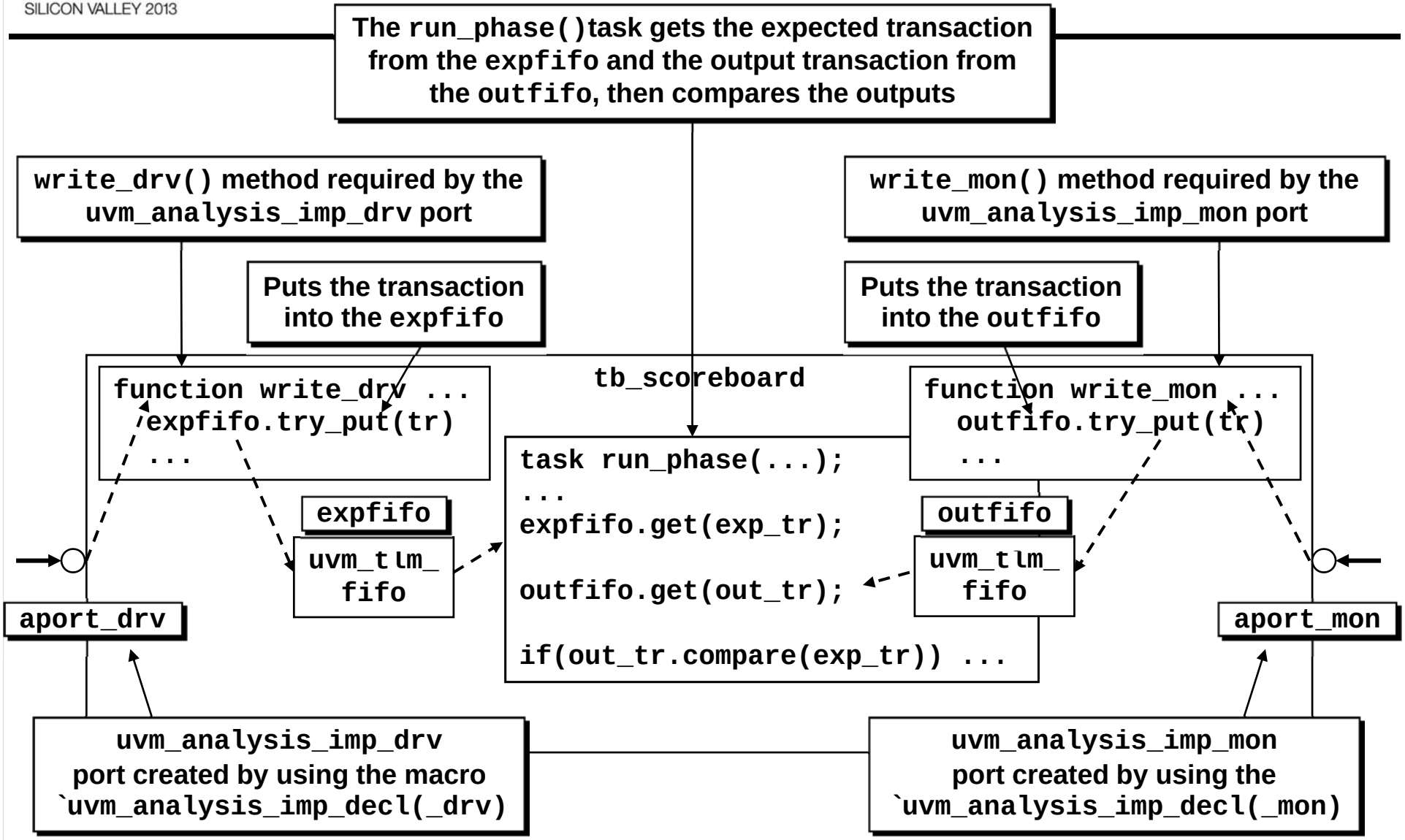
```
`uvm_analysis_imp_decl(_drv)
`uvm_analysis_imp_decl(_mon)

class tb_scoreboard extends uvm_scoreboard;
  `uvm_component_utils(tb_scoreboard)

  uvm_analysis_imp_drv #(...) ...;
  uvm_analysis_imp_mon #(...) ...;
  ...

  function void write_drv(...);
    ...
  endfunction

  function void write_mon(...);
    ...
  endfunction
  ...
endclass
```

`uvm_analysis_imp_decl macros create multiple analysis ports in the same component

The suffix names are then required in the uvm_analysis_imp<*_suffix*> *analysis_imp* port names

The suffix names are also required in the write<*_suffix*> method names

Scoreboard w/ uvm_tlm_fifos



snug
Synopsys Users Group
SILICON VALLEY 2013

TB Scoreboard - Multiple Analysis Ports

tb_scoreboard.sv (Part 1 of 4)

Sunburst Design

```
`uvm_analysis_imp_decl(_drv)
`uvm_analysis_imp_decl(_mon)

class tb_scoreboard extends uvm_scoreboard;
  `uvm_component_utils(tb_scoreboard)

  uvm_analysis_imp_drv #(trans1, tb_scoreboard) aport_drv;
  uvm_analysis_imp_mon #(trans1, tb_scoreboard) aport_mon;

  uvm_tlm_fifo #(trans1) expfifo;
  uvm_tlm_fifo #(trans1) outfifo;

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

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    aport_drv = new("aport_drv", this);
    aport_mon = new("aport_mon", this);
    expfifo   = new("expfifo",   this, 0);
    outfifo   = new("outfifo",   this, 0);
  endfunction
  ...
```

`uvm_analysis_imp_decl macros create multiple analysis ports in the same component

Recommended: use the <*_suffix*> name as part of the port handle-names

Use two uvm_tlm_fifos to build the scoreboard

Construct the two analysis ports using new() constructors

Construct the two *unbounded* TLM FIFOs using new() constructors

Scoreboard Architecture #2

Includes:

- Predictor w/ extern sb_calc_exp function
- Comparator w/ 2 uvm_tlm_analysis_fifos

Scoreboard Architecture #2

- Predictive scoreboard built from pre-coded blocks

- Scoreboard blocks:

`tb_scoreboard` ← **Fully coded scoreboard wrapper**

`sb_predictor` ← **Fully coded predictor with `extern` `sb_calc_exp` function call**

`sb_calc_exp` ← ***Extern* function - requires the expected value to be calculated**

Only scoreboard file that requires coding

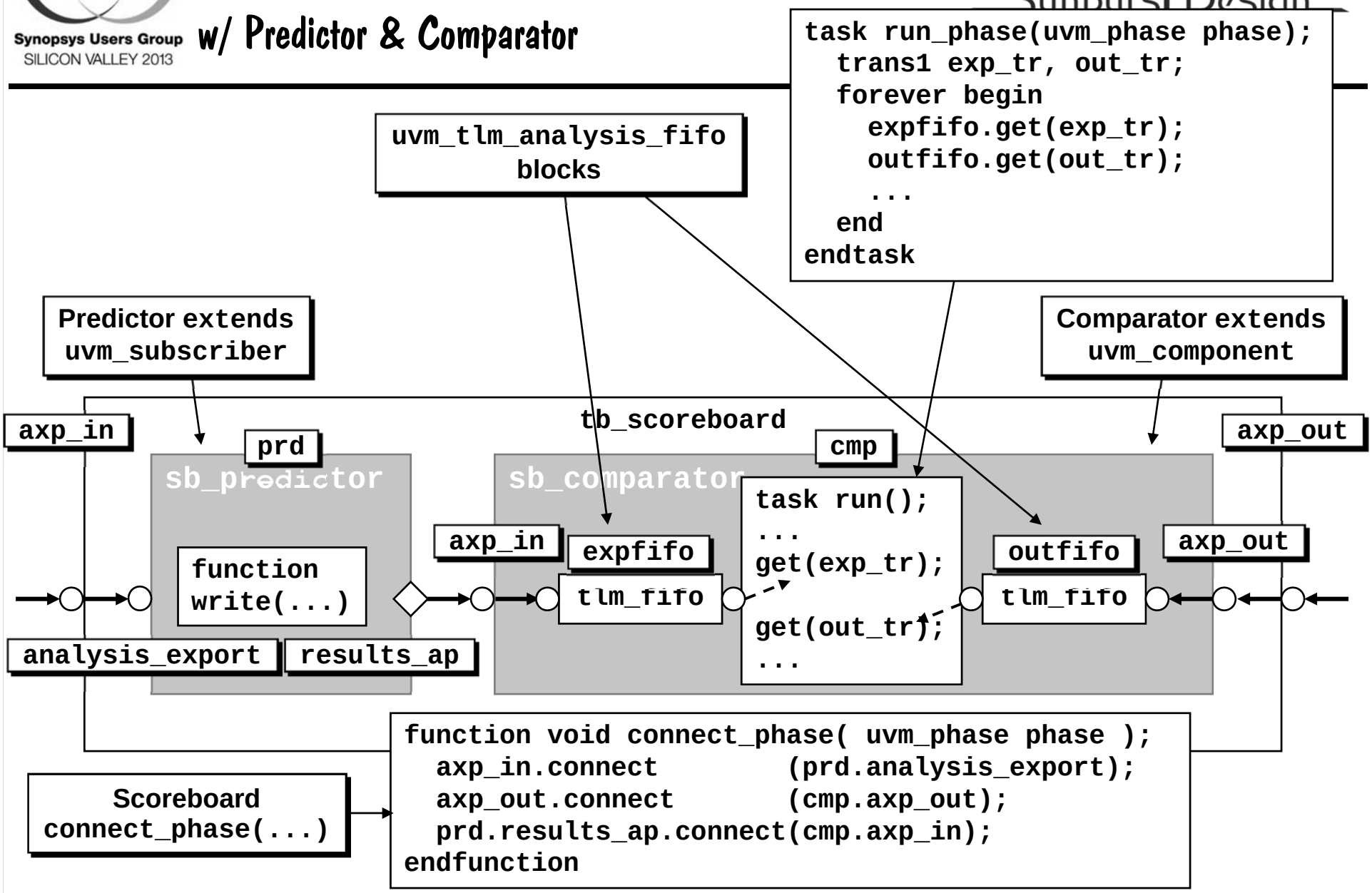
`sb_comparator` ← **Fully coded comparator utilizing `uvm_tlm_analysis_fifos`**

Synopsys Users Group
SILICON VALLEY 2013

Scoreboard Architecture #2

w/ Predictor & Comparator

SunburstDesign



Scoreboard w/ Predictor & Compa

This file is pre-coded - no modification required


```
class tb_scoreboard extends uvm_scoreboard;
  `uvm_component_utils(tb_scoreboard)

  uvm_analysis_export #(trans1) axp_in;
  uvm_analysis_export #(trans1) axp_out;
  sb_predictor                  prd;
  sb_comparator                 cmp;

  function new(string name, uvm_component parent);  ...

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    axp_in  = new("axp_in",  this);
    axp_out = new("axp_out", this);
    prd     =  sb_predictor::type_id::create("prd", this);
    cmp     = sb_comparator::type_id::create("cmp", this);
  endfunction

  function void connect_phase( uvm_phase phase );
    axp_in.connect        (prd.analysis_export);
    axp_out.connect       (cmp.axp_out);
    prd.results_ap.connect(cmp.axp_in);
  endfunction
endclass
```

Declare uvm_analysis_export handles

Declare the predictor and comparator handles

new( )-construct the analysis exports

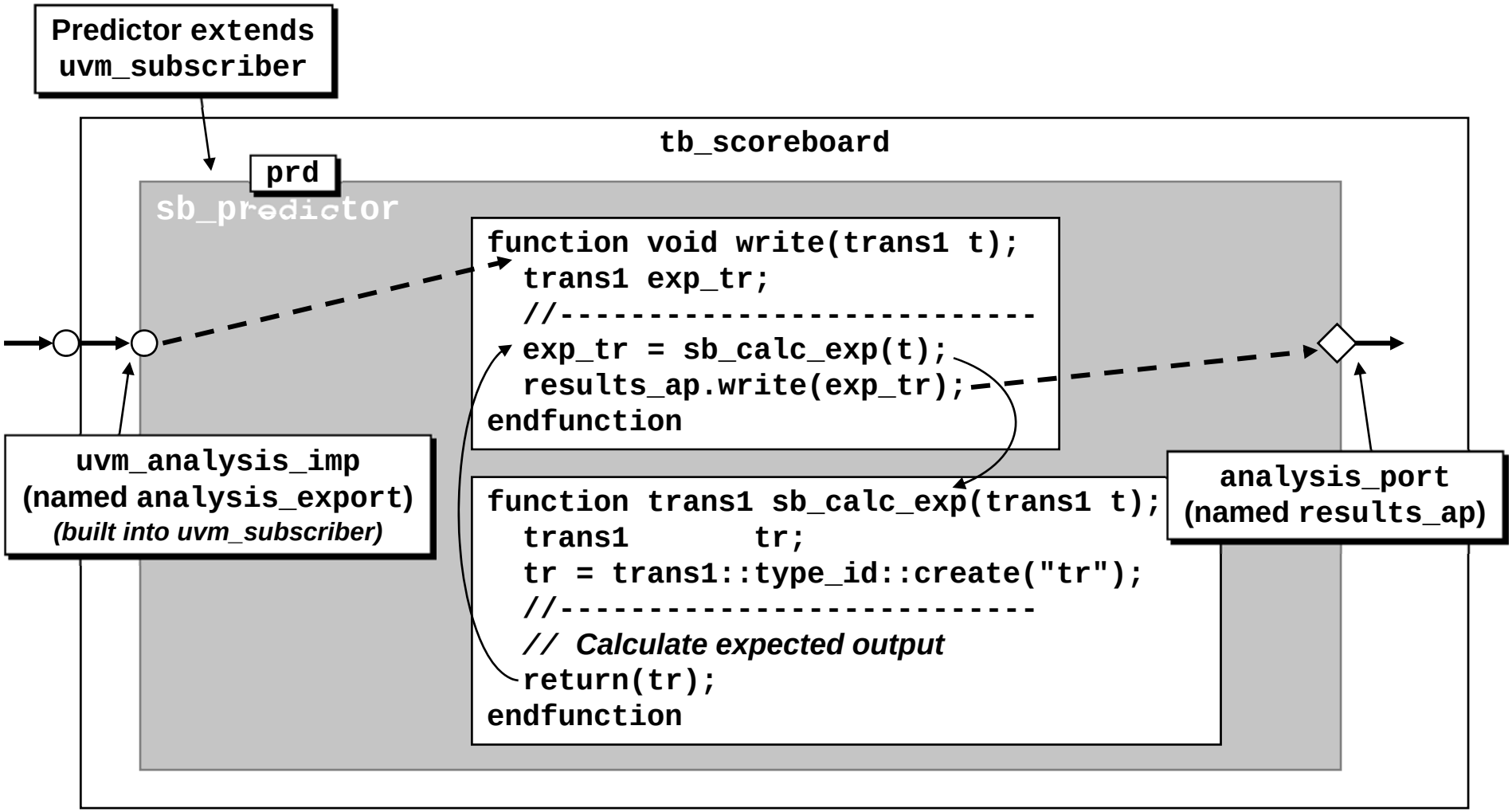
Create the predictor and comparator

Connect:

```
sbd axp_in      to prd analysis_export
sbd axp_out     to cmp axp_out
prd results_ap  to cmp axp_in
```

Predictor

sb_predictor



snug Synopsys Users Group SILICON VALLEY 2013

Predictor

sb_predictor

File: sb_predictor.sv

This file is pre-coded - no modification required

SunburstDesign

```
class sb_predictor extends uvm_subscriber #(trans1);
  `uvm_component_utils(sb_predictor)

  uvm_analysis_port #(trans1) results_ap;

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

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    results_ap = new("results_ap", this);
  endfunction

  function void write(trans1 t);
    trans1 exp_tr;
    //--------------------------
    exp_tr = sb_calc_exp(t);
    results_ap.write(exp_tr);
  endfunction

  extern function trans1 sb_calc_exp(trans1 t);
endclass
```

Declare a `results_ap` analysis port handle

Construct the `results_ap` analysis port using `new()`

`t` transaction is passed to the `write()` method

Declare `exp_tr` *(expected transaction)*

`t` is sent to `sb_calc_exp()` (`exp_tr` gets return value)

`exp_tr` return value is written to the `results_ap`

Declare `sb_calc_exp` to be an `extern` function

snug
Synopsys Users Group
SILICON VALLEY 2013

Calculate Expected Output

sb_calc_exp

This file requires user modification

`sb_calc_exp` *extern* function of `sb_predictor`

```
function trans1 sb_predictor::sb_calc_exp (trans1 t);
  static logic [15:0] next_dout;
         logic [15:0] dout;

  trans1 tr = trans1::type_id::create("tr");
  //---------------------------
  `uvm_info(get_type_name(), t.convert2string(), UVM_HIGH)

  // async reset: reset the next_dout AND current dout values -OR-
  // non-reset  : assign dout values & calculate the next_dout values
  dout = next_dout;
  if      (!t.rst_n) {next_dout,dout} = '0;
  else if ( t.ld)     next_dout       = t.din;
  else if ( t.inc)    next_dout++;

  // copy all sampled inputs & outputs
  tr.copy(t);
  // overwrite the dout values with the calculated values.
  // dout values were either calculated in the previous cycle
  //      or asynchronously reset in this cycle
  tr.dout = dout;
  return(tr);
endfunction
```

Input transaction (t)

`static ... next_dout` Save `next_dout` values between calls to `sb_calc_exp()`

Create an expected transaction (`tr`)

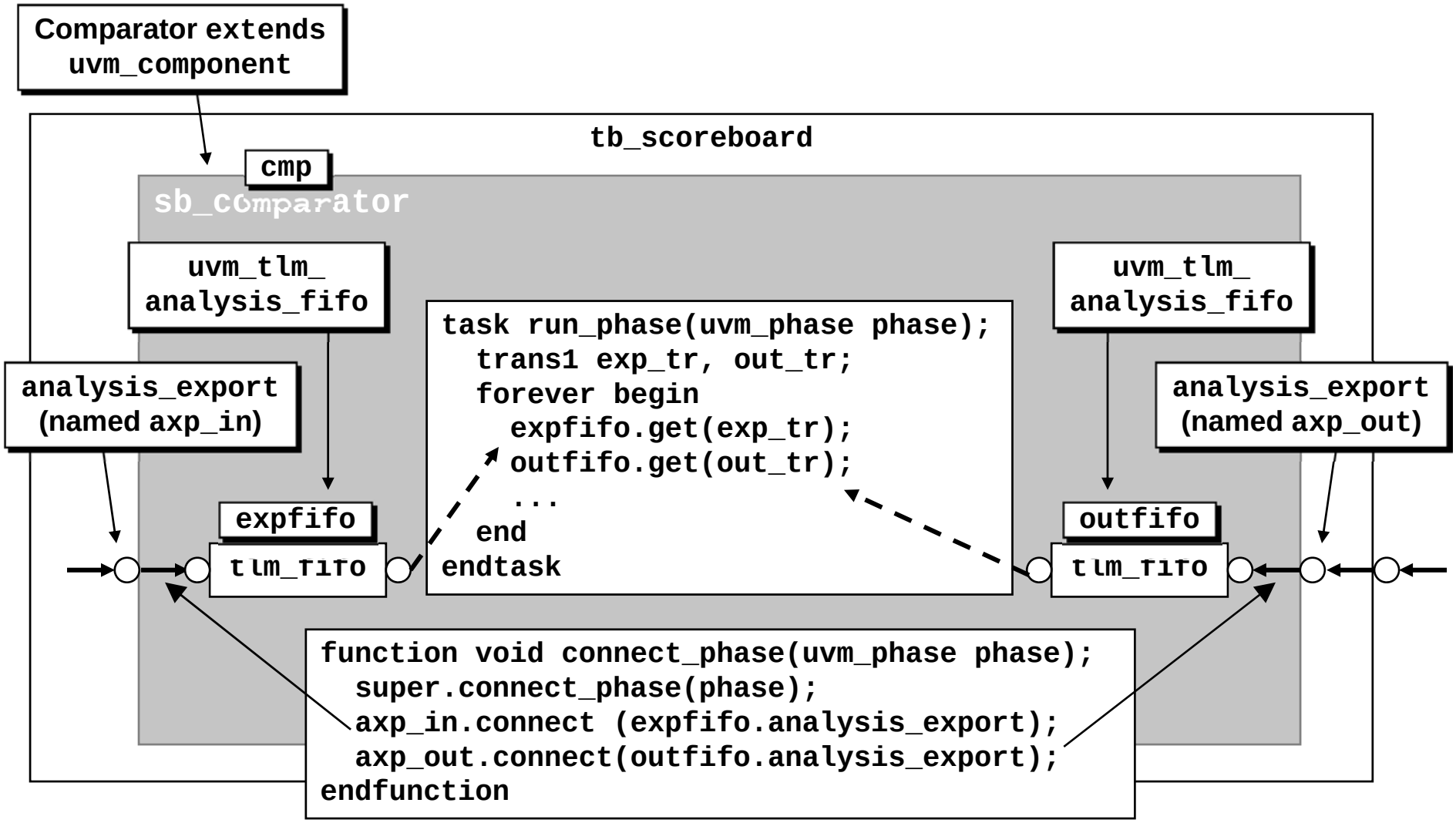
Calculate the expected output value

Copy input `t` to expected `tr`

Overwrite expected `dout` with calculated `dout` ... then return the `tr` transaction

Comparator

sb_comparator



Comparator

sb_comparator

This file is pre-coded - no modification required

File: sb_comparator.sv
(Part 1 of 3)

```
class sb_comparator extends uvm_component;
  `uvm_component_utils(sb_comparator)

  uvm_analysis_export   #(trans1) axp_in;
  uvm_analysis_export   #(trans1) axp_out;
  uvm_tlm_analysis_fifo #(trans1) expfifo;
  uvm_tlm_analysis_fifo #(trans1) outfifo;

  function new (string name, uvm_component parent); ...

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    axp_in  = new("axp_in",  this);
    axp_out = new("axp_out", this);
    expfifo = new("expfifo", this);
    outfifo = new("outfifo", this);
  endfunction

  function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    axp_in.connect (expfifo.analysis_export);
    axp_out.connect(outfifo.analysis_export);
  endfunction
...
```

snug Synopsys Users Group SILICON VALLEY 2013

Comparator

sb_comparator

Sunburst Design

File: `sb_comparator.sv`
(Part 2 of 3)

```
 ...
task run_phase(uvm_phase phase);
  trans1 exp_tr, out_tr;
  forever begin
    `uvm_info("sb_comparator run", "WAITING for expected output", UVM_DEBUG)
    expfifo.get(exp_tr);
    `uvm_info("sb_comparator run", "WAITING for actual output", UVM_DEBUG)
    outfifo.get(out_tr);
    if (out_tr.compare(exp_tr)) begin


      PASS();
      `uvm_info ("PASS ", $sformatf("Actual=%s   Expected=%s \n",
                        out_tr.output2string(), exp_tr.convert2string()), 250)
    end
    else begin
      ERROR();
      `uvm_error("ERROR", $sformatf("Actual=%s   Expected=%s \n",
                         out_tr.output2string(), exp_tr.convert2string()))
    end
  end
endtask
...
```

If the `compare()` method is setup in the `trans1` transaction class, this code will work as is
(no need to reference explicit signals)

… else, explicit comparison code must be added here

Output messages show *actual* outputs, *expected* outputs *AND* inputs

Comparator

sb_comparator

File: sb_comparator.sv
(Part 3 of 3)

```
  ...
  int VECT_CNT, PASS_CNT, ERROR_CNT;

  function void report_phase (uvm_phase phase);
    super.report_phase(phase);
    if (VECT_CNT && !ERROR_CNT)
      `uvm_info("PASSED",
      $sformatf("\n\n\n*** TEST PASSED - %0d vectors ran, %0d vectors passed ***\n",
                VECT_CNT, PASS_CNT), UVM_LOW)
    else
      `uvm_error("FAILED",
      $sformatf("\n\n\n*** TEST FAILED - %0d vectors ran, %0d vectors passed, %0d vectors failed ***\n",
                VECT_CNT, PASS_CNT, ERROR_CNT)
  endfunction

  function void PASS();
    VECT_CNT++;
    PASS_CNT++;
  endfunction

  function void ERROR();
    VECT_CNT++;
    ERROR_CNT++;
  endfunction
endclass
```

report_phase() used to show final PASS/FAIL message

PASS() and ERROR() called from the run_phase() task

Summary & Conclusions

- Scoreboard Architecture style #1:
  - Requires two **`uvm_analysis_imp`** ports
  - Corresponding **`write()`** methods
  - Uses **`` `uvm_analysis_imp_decl() ``** macros

Not just for scoreboards !!

Technique that is required for any multi-*analysis-imp* port component

- Scoreboard Architecture style #2:
  - Uses pre-coded components
  - Only requires user to finish the external **`sb_calc_exp()`** function

Greatly simplifies scoreboard development

- Remember: add a **`compare()`** method to the transaction

Also simplifies scoreboard development

Acknowledgements

- Thanks to my good friends and colleagues

Great reviews of the paper and presentation slides

  - Al Czamara
  - Stu Sutherland
  - Chris Spear
  - Jean Fong

OVM/UVM Scoreboards
Fundamental Architectures

Clifford E. Cummings

Sunburst Design, Inc.

cliffc@sunburst-design.com

www.sunburst-design.com

World-class Verilog, SystemVerilog & OVM/UVM Training

Life is too short for bad or boring training!