EGC 455

SOC Design &Verification

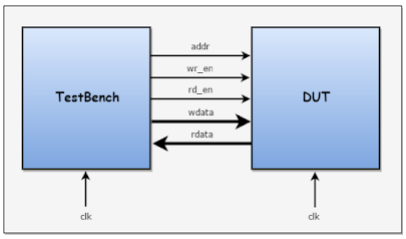
# System Verilog Interfaces and Bus Function Models

New Paltz
STATE UNIVERSITY OF NEW YORK
**Baback Izadi**
Division of Engineering Programs
bai@engr.newpaltz.edu

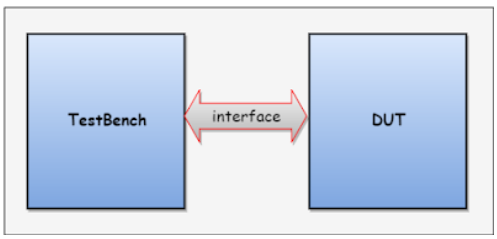
1

## System Verilog Interface

- https://www.youtube.com/watch?v=YQ-HgqjuSuA
- An interface is a named bundle of wires
  - It encapsulate communication
  - Direction information via modports
  - Timing information such as clocking




without interface




SystemVerilog interface

SUNY – New Paltz
Elect. & Comp. Eng.

2

## Advantages of Bus Interface

- Allows the number of signals to be grouped together and represented as a single port
  - The single port handle is passed instead of multiple signal/ports.
- Interface declaration is made once, and the handle is passed across the modules/components.
- Addition and deletion of signals are easy.

SUNY – New Paltz
Elect. & Comp. Eng.

3

## SystemVerilog Interface

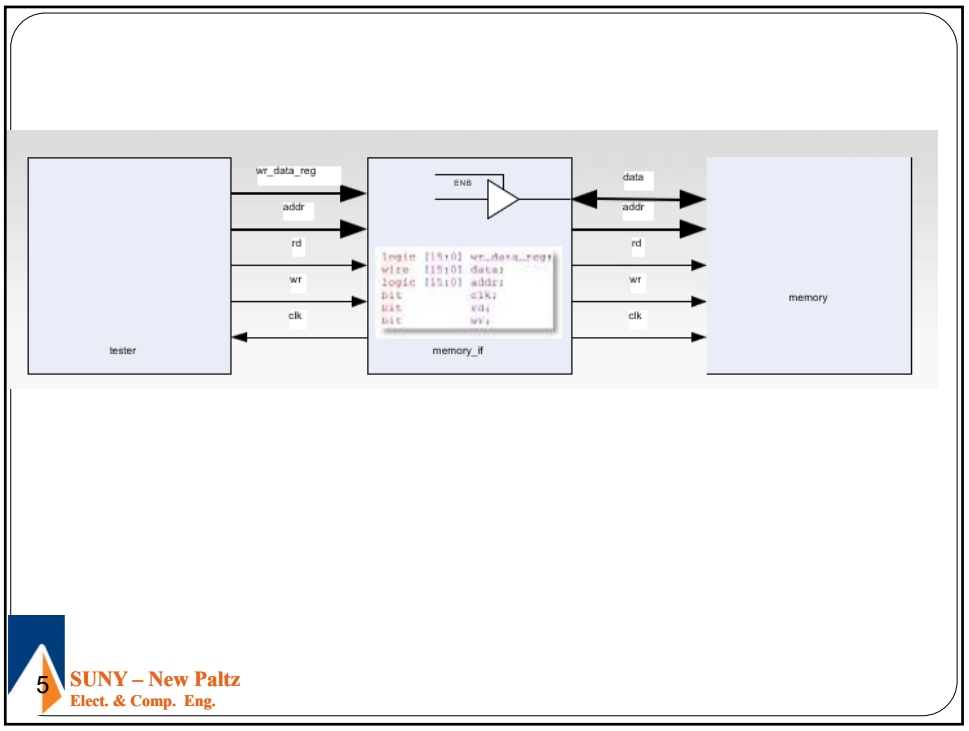
Interface is like a module with signals in it. Instantiate it to use it.

The interface uses `modports` to control signal direction

```
1 module top;
2
3    memory_if mi();
4    memory dut (mi.mem_mp);
5    tester tst (mi.tester);
6
7 endmodule // top
8
```

SUNY – New Paltz
Elect. & Comp. Eng.

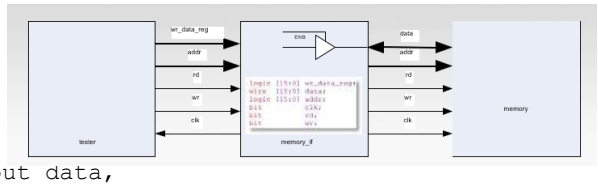
4




5

## Memory_if

```
interface memory_if ;
   logic [15:0] wr_data_reg;
   wire  [15:0] data;
   logic [15:0] addr;
   bit          clk;
   bit          rd;
   bit          wr;
      modport mem_mp (
                    inout data,
                    input addr,
                    input clk,
                    input rd,
                    input wr);
   modport tester(
                   output wr_data_reg,
                   input clk,
                   output addr,
                   output rd,
                   output wr);
   assign data = (wr) ? wr_data_reg : 16'hZZZZ;
```



SUNY – New Paltz
Elect. & Comp. Eng.

6

## Memory_if (Cont.)

```
// monitor data and generate clock
      initial begin
      $monitor("addr: %1h  data: %4h   rd: %1b  wr:
%1b",
                    addr, data, rd, wr);
      clk = 0;
      forever begin
         #10;
         clk = ~clk;
      end
   end
   //  Self Checking Test Bench
      logic [15:0] testmem [2**16-1:0];
   always @(posedge clk) begin
      if (rd) begin
         #1;
         assert(data === testmem[addr]);
      end
      if (wr)
        testmem[addr] = data;
   end
                        endinterface : memory_if
```

SUNY – New Paltz
Elect. & Comp. Eng.

7

## Defining an Interface

Declare an interface in the same way as a module

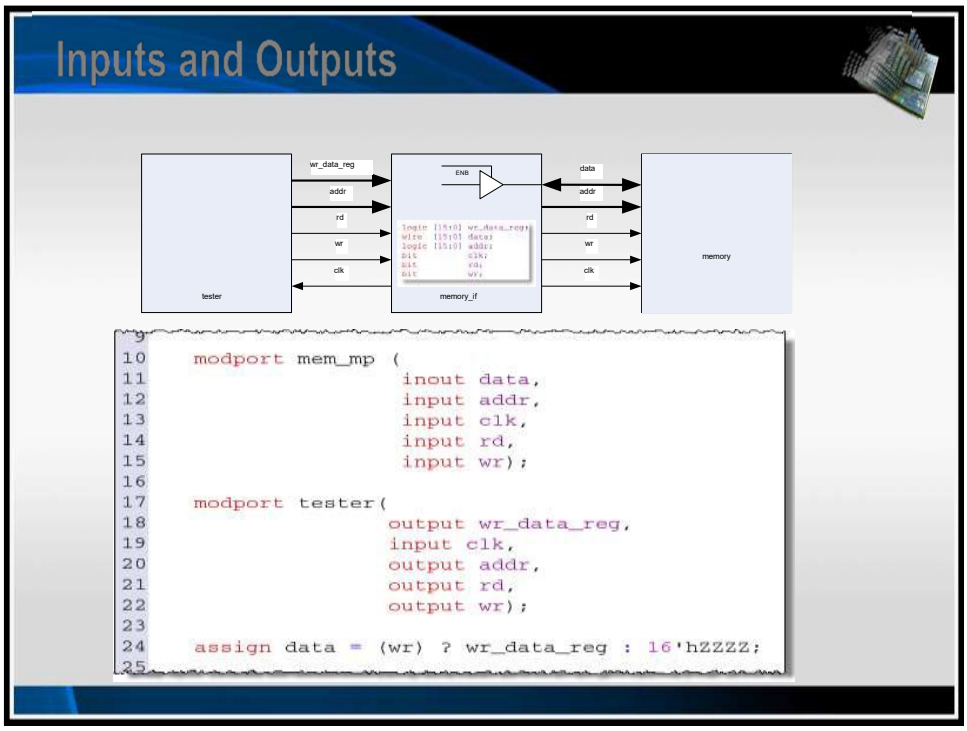
Declare the variables that make the interface

Notice that directions don't go here

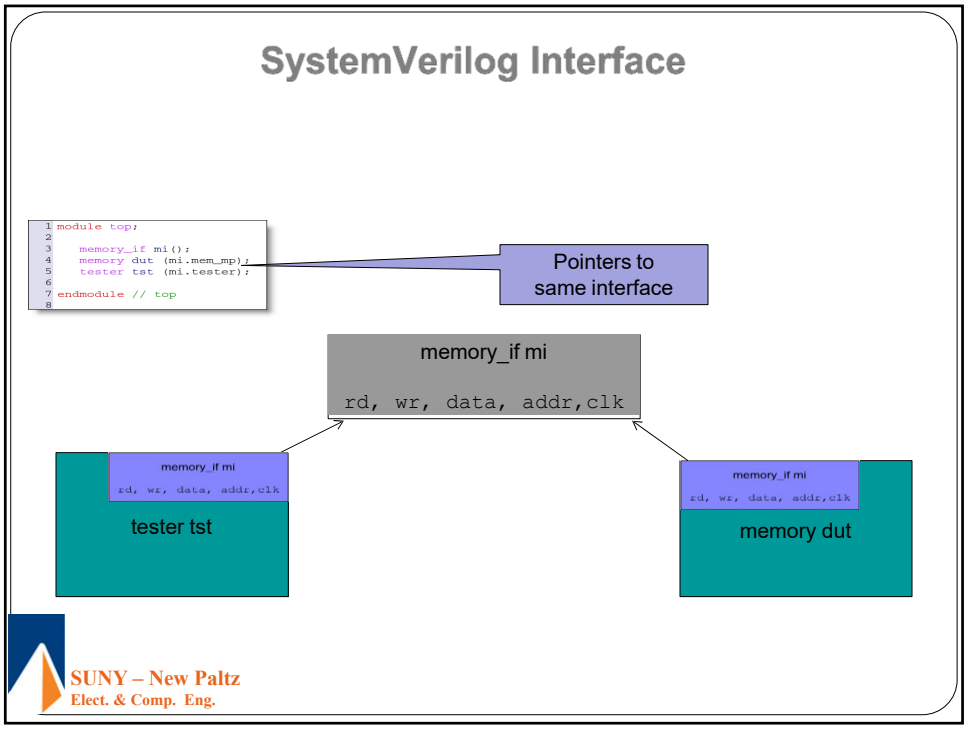
```
1  interface memory_if ;
2
3     logic [15:0] wr_data_reg;
4     wire  [15:0] data;
5     logic [15:0] addr;
6     bit          clk;
7     bit          rd;
8     bit          wr;
9
```

SUNY – New Paltz
Elect. & Comp. Eng.

8


Inputs and Outputs
wr_data_reg
addr
rd
wr
clk
ENB
data
addr
rd
wr
clk
tester
memory_if
memory
9
10 modport mem_mp (
11 inout data,
12 input addr,
13 input clk,
14 input rd,
15 input wr);
16
17 modport tester(
18 output wr_data_reg,
19 input clk,
20 output addr,
21 output rd,
22 output wr);
23
24 assign data = (wr) ? wr_data_reg : 16'hZZZZ;
25

9


SystemVerilog Interface
1 module top;
2
3 memory_if mi();
4 memory dut (mi.mem_mp);
5 tester tst (mi.tester);
6
7 endmodule // top
8
Pointers to same interface
memory_if mi
rd, wr, data, addr,clk
memory_if mi
rd, wr, data, addr,clk
tester tst
memory_if mi
rd, wr, data, addr,clk
memory dut
SUNY – New Paltz
Elect. & Comp. Eng.

10

## Interface in Action

```
1  module top;
2
3     memory_if mi();
4     memory dut (mi.mem_mp);
5     tester tst (mi.tester);
6
7  endmodule // top
8
```

```
interface memory_if ;

   logic [15:0] wr_data_reg;
   wire  [15:0] data;
   logic [15:0] addr;
   bit          clk;
   bit          rd;
   bit          wr;
```

```
module tester (memory_if.tester t);
   logic [3:0]  tiny_addr;

   initial begin
      t.wr = 1'b1;
      for (int i=0; i< 'h10; i++) begin
         @(negedge t.clk);
         t.wr_data_reg = i;
         t.addr = i;
      end
```

```
module memory (memory_if.mem_mp m);
   logic [15:0] mem [2**16-1 : 0];

   assign m.data = (m.rd) ? mem[m.addr] : 16'hZZZZ;

   always @(posedge m.clk)
     if (m.wr)
       mem[m.addr] = m.data;

endmodule // memory
```

SUNY – New Paltz
Elect. & Comp. Eng.

11

## Interfaces can generate signals

Monitoring Data

```
28     initial begin
29        $monitor("addr: %1h  data: %4h   rd: %1b  wr: %1b",
30                 addr, data, rd, wr);
31        clk = 0;
32        forever begin
33           #10;
34           clk = ~clk;
35        end
36     end
37
```

Generating clock

SUNY – New Paltz
Elect. & Comp. Eng.

12

## Separating Self-Checking from Testing

Memory behavioral model and checker

```
37
38      //  Self Checking Test Bench
40      logic [15:0] testmem [2**16-1:0];
41
42      always @(posedge clk) begin
43         if (rd) begin
44            #1;
45            assert(data === testmem[addr]);
46         end
47
48         if (wr)
49           testmem[addr] = data;
50      end
51
52
53  endinterface : memory_if
```

SUNY – New Paltz
Elect. & Comp. Eng.

13

## The tester module generates stimulus

```
 1  module tester (memory_if.tester t);
 2     logic [3:0]  tiny_addr;
 3
 4     initial begin
 5        t.wr = 1'b1;
 6        for (int i=0; i< 'h10; i++) begin
 7           @(negedge t.clk);
 8           t.wr_data_reg = i;
 9           t.addr = i;
10        end
11
12        repeat (50) begin
13           @(negedge t.clk);
14           t.wr = $random;
15           t.rd = ~t.wr;
16           tiny_addr = $random;
17           t.addr = {12'd0,tiny_addr};
18           if (t.wr) begin
19              t.wr_data_reg = $random;
20           end
21        end // repeat (50)
22        $stop;
23     end // initial begin
24  endmodule // tester
```

SUNY – New Paltz
Elect. & Comp. Eng.

14

## Our Interface In Context

```
1 module top;
2
3    memory_if mi();
4    memory dut (mi.mem_mp);
5    tester tst (mi.tester);
6
7 endmodule // top
8
```

SUNY – New Paltz
Elect. & Comp. Eng.

15

## Interface in Action

```
# addr: 1  data: 2f96   rd: 0  wr: 1
# addr: 2  data: 3eae   rd: 1  wr: 0
# addr: 7  data: 0007   rd: 1  wr: 0
# addr: 2  data: 007e   rd: 0  wr: 1
# addr: 9  data: 8f1f   rd: 0  wr: 1
# addr: 5  data: 8878   rd: 0  wr: 1
# addr: 9  data: ae3f   rd: 0  wr: 1
# addr: 8  data: 60b7   rd: 1  wr: 0
```

SUNY – New Paltz
Elect. & Comp. Eng.

16

## Interfaces: Portal to the Class World

**Classes don't have port lists.**
**How can we communicate with them?**

SUNY – New Paltz
Elect. & Comp. Eng.

17

## Class-Based Test Bench

**Definition of `tester` Class**

**Use interface to connect to DUT**

**Declare handle for tester class**

**Make a new tester and pass interface**

**Run test**

```
1  `include "tester.svh"
2
3  module top;
4
5      memory_if mi();
6      memory dut (mi.mem_mp);
7      tester tst;
8
9      initial begin
10         tst = new(mi);
11         fork
12             tst.run;
13         join_none
14     end
15
16 endmodule // top
```

SUNY – New Paltz
Elect. & Comp. Eng.

18

## Tester Class

Declare a virtual interface

```
1  class tester;
2
3     logic [3:0]   tiny_addr;
4     virtual       interface memory_if t;
5
6     function new(virtual interface memory_if it);
7        t = it;
8     endfunction // new
9
10
```

**Dumb Fact: The word virtualin this context has nothing to do with virtual functions.**

**Doh!**

SUNY – New Paltz
Elect. & Comp. Eng.

19

## Tester class

```
class tester;
      logic [3:0]  tiny_addr;
   virtual      interface memory_if t;
   function new(virtual interface memory_if it);
      t = it;
   endfunction // new
    task run;
      t.wr = 1'b1;
      for (int i=0; i< 'h10; i++) begin
         @(negedge t.clk);
         t.wr_data_reg = i;
         t.addr = i;
      end
      repeat (50) begin
         @(negedge t.clk);
         t.wr = $random;
         t.rd = ~t.wr;
         tiny_addr = $random;
         t.addr = {12'd0,tiny_addr};
         if (t.wr) begin
            t.wr_data_reg = $random;
   end
       end
$stop;
    endtask // run
endclass // tester
```

SUNY – New Paltz
Elect. & Comp. Eng.

20

## Put Test in run() Method

Access Signals in Interface

Test stops after all stimulus is run

```
12    task run;
13       t.wr = 1'b1;
14       for (int i=0; i< 'h10; i++) begin
15          @(negedge t.clk);
16          t.wr_data_reg = i;
17          t.addr = i;
18       end
19
20       repeat (50) begin
21          @(negedge t.clk);
22          t.wr = $random;
23          t.rd = ~t.wr;
24          tiny_addr = $random;
25          t.addr = {12'd0,tiny_addr};
26          if (t.wr) begin
27             t.wr_data_reg = $random;
28          end
29       end
30       $stop;
31    endtask // run
32 endclass // tester
```

SUNY – New Paltz
Elect. & Comp. Eng.

21

## Interface in Class

```
include "tester.svh"
module top;
   memory_if mi();
   memory dut (mi.mem_mp);
   tester tst;
   initial begin
      tst = new(mi);
      fork
         tst.run;
      join_none
   end
endmodule // top
```

```
interface memory_if ;
   logic [15:0] wr_data_reg;
   wire  [15:0] data;
   logic [15:0] addr;
   bit          clk;
   bit          rd;
   bit          wr;
```

```
class tester;
   logic [3:0]  tiny_addr;
   virtual      interface memory_if t;
   function new(virtual interface memory_if it);
      t = it;
   endfunction // new
   task run;
      t.wr = 1'b1;
      for (int i=0; i< 'h10; i++) begin
         @(negedge t.clk);
         t.wr_data_reg = i;
```

```
module memory (memory_if.mem_mp m);
   logic [15:0] mem [2**16-1 : 0];
   assign m.data = (m.rd) ? mem[m.addr] : 16'hZZZZ;
   always @(posedge m.clk)
     if (m.wr)
       mem[m.addr] = m.data;
endmodule // memory
```

SUNY – New Paltz
Elect. & Comp. Eng.

22

## What about VHDL?

```
entity memory is

  port (
    clk  : in    std_logic;
    rd   : in    std_logic;
    wr   : in    std_logic;
    addr : in    std_logic_vector(15 downto 0);
    data : inout std_logic_vector (15 downto 0)
  );

end memory;

architecture rtl of memory is

type ram_type is array (65535 downto 0) of  std_logic_vector(15 downto 0);

signal mem : ram_type;


begin  -- rtl
```

SUNY – New Paltz
Elect. & Comp. Eng.

23

## Wrap VHDL DUTs

```
`include "tester.svh"

module top;

    memory_if mi();
    memory_wrapper dut (mi.mem_mp);
    tester tst;

    initial begin
        tst = new(mi);
        fork
            tst.run;
        join_none
    end

endmodule // top
```

SUNY – New Paltz
Elect. & Comp. Eng.

24

## SystemVerilog VHDL Wrapper

```
module memory_wrapper (memory_if.mem_mp m);

   memory VHDL_DUT
      (.clk(m.clk),
       .rd(m.rd),
       .wr(m.wr),
       .addr(m.addr),
       .data(m.data));

endmodule // memory
```

SUNY – New Paltz
Elect. & Comp. Eng.

25

## VHDL Wrapper

```
include "tester.svh"
module top;
   memory_if mi();
   memory dut (mi.mem_mp);
   tester tst;
   initial begin
      tst = new(mi);
      fork
         tst.run;
      join_none
   end
endmodule // top
```

```
interface memory_if ;
   logic [15:0] wr_data_reg;
   wire  [15:0] data;
   logic [15:0] addr;
   bit          clk;
   bit          rd;
   bit          wr;
```

```
class tester;
   logic [3:0]  tiny_addr;
   virtual      interface memory_if t;
   function new(virtual interface memory_if it);
      t = it;
   endfunction // new
   task run;
      t.wr = 1'b1;
      for (int i=0; i< 'h10; i++) begin
         @(negedge t.clk);
         t.wr_data_reg = i;
```

```
module memory_wrapper (memory_if.mem_mp m);
   memory VHDL_DUT
      (.clk(m.clk),
       .rd(m.rd),
       .wr(m.wr),
       .addr(m.addr),
       .data(m.data));
endmodule // memory
```

SUNY – New Paltz
Elect. & Comp. Eng.

26

## Results

```
nterfaces/class_example/class_based/w
# Loading work.memory
# addr: x  data: xxxx   rd: 0  wr: 1
# addr: 0  data: 0000   rd: 0  wr: 1
# addr: 1  data: 0001   rd: 0  wr: 1
# addr: 2  data: 0002   rd: 0  wr: 1
# addr: 3  data: 0003   rd: 0  wr: 1
# addr: 4  data: 0004   rd: 0  wr: 1
# addr: 5  data: 0005   rd: 0  wr: 1
# addr: 6  data: 0006   rd: 0  wr: 1
# addr: 7  data: 0007   rd: 0  wr: 1
# addr: 8  data: 0008   rd: 0  wr: 1
```

SUNY – New Paltz
Elect. & Comp. Eng.

27

## Summary

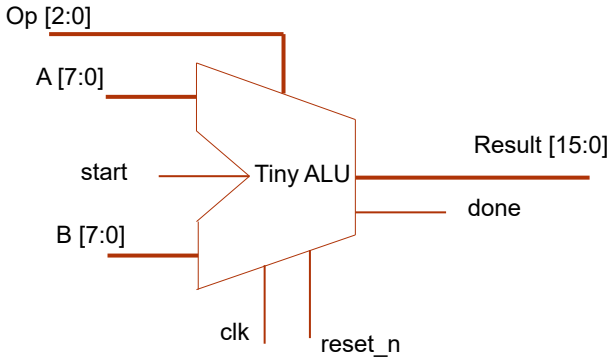
- Define the Interface
- Define the class to contain a virtual interface of the same type.
- Create a new() function that accepts the
- virtual interface as an argument.
- Access the signals using the interface

SUNY – New Paltz
Elect. & Comp. Eng.

28

## Bus Function Model for Tiny ALU

- ALU works at the rising edge of the clock. When `start` is active, it reads the operands. Operations can take any number of cycles. It rises `done` one clock cycle when the operation is complete. The `reset_n` is active low. The `start` must remain high until `done` signal is raised. No `done` signal on `nop`.



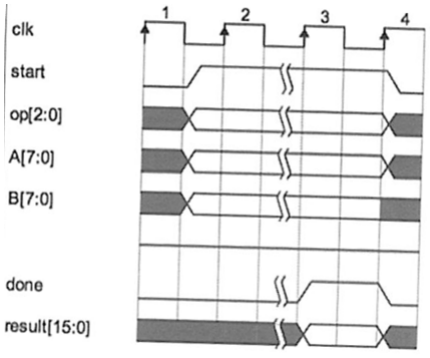

SUNY – New Paltz
Elect. & Comp. Eng.

29

## Test Plan for Tiny ALU

- https://www.youtube.com/watch?v=53vDSE-CQ3I

| Operation | Opcode |
|---|---|
| no_op | 3'b000 |
| add_op | 3'b001 |
| and_op | 3'b010 |
| xor_op | 3'b011 |
| mul_op | 3'b100 |
| unused | 3'b101 - 3'b111 |




SUNY – New Paltz
Elect. & Comp. Eng.

30

## Files

| | |
|---|---|
| tinyalu_dut | Initial code add |
| coverage.sv | This release fixes a bug in one of the covergroups that caused a warn... |
| dut.f | Initial code add |
| run.do | Now works properly with a SystemVerilog DUT. Fixed scoreboard race co... |
| scoreboard.sv | Now works properly with a SystemVerilog DUT. Fixed scoreboard race co... |
| tb.f | Initial code add |
| tester.sv | Initial code add |
| tinyalu_bfm.sv | Initial code add |
| tinyalu_pkg.sv | Initial code add |
| top.sv | Initial code add |

SUNY – New Paltz
Elect. & Comp. Eng.

31

## Module top

```
module top;
   tinyalu_bfm    bfm();
   tester     tester_i    (bfm);
   coverage   coverage_i  (bfm);
   scoreboard scoreboard_i(bfm);

   tinyalu DUT (.A(bfm.A), .B(bfm.B), .op(bfm.op),
                .clk(bfm.clk), .reset_n(bfm.reset_n),
                .start(bfm.start), .done(bfm.done), .result(bfm.result));
endmodule : top
```

SUNY – New Paltz
Elect. & Comp. Eng.

32

## Module tinyalu_bfm

```
interface tinyalu_bfm;
   import tinyalu_pkg::*;

   byte          unsigned        A;
   byte          unsigned        B;
   bit           clk;
   bit           reset_n;
   wire [2:0]    op;
   bit           start;
   wire          done;
   wire [15:0]   result;

   operation_t   op_set;

   assign op = op_set;
```

```
initial begin
      clk = 0;
      forever begin
         #10;
         clk = ~clk;
      end
   end
// Reset task
   task reset_alu();
      reset_n = 1'b0;
      @(negedge clk);
      @(negedge clk);
      reset_n = 1'b1;
      start = 1'b0;
   endtask : reset_alu
```

SUNY – New Paltz
Elect. & Comp. Eng.

33

## Module tinyalu_bfm (Cont.)

```
task send_op(input byte iA,
input byte iB, input
operation_t iop, output
shortint alu_result);

     op_set = iop;

     if (iop == rst_op) begin
         @(posedge clk);
         reset_n = 1'b0;
         start = 1'b0;
         @(posedge clk);
         #1;
         reset_n = 1'b1;
      end else begin
         @(negedge clk);
         A = iA;
         B = iB;
```

```
start = 1'b1;
         if (iop == no_op) begi
            @(posedge clk);
            #1;
            start = 1'b0;
         end else begin
            do
              @(negedge clk);
            while (done == 0);
            start = 1'b0;
         end
      end //else:!if(iop==rst_o

   endtask : send_op

endinterface : tinyalu_bfm
```

SUNY – New Paltz
Elect. & Comp. Eng.

34

## Package tinyalu_pkg

```
package tinyalu_pkg;
      typedef enum bit[2:0] {no_op  = 3'b000,
                             add_op = 3'b001,
                             and_op = 3'b010,
                             xor_op = 3'b011,
                             mul_op = 3'b100,
                             rst_op = 3'b111} operation_t;

endpackage : tinyalu_pkg
```

SUNY – New Paltz
Elect. & Comp. Eng.

35

## Module tester

```
odule tester(tinyalu_bfm bfm);
  import tinyalu_pkg::*;

  function operation_t get_op();
     bit [2:0] op_choice;
     op_choice = $random;
     case (op_choice)
       3'b000 : return no_op;
       3'b001 : return add_op;
       3'b010 : return and_op;
       3'b011 : return xor_op;
       3'b100 : return mul_op;
       3'b101 : return no_op;
       3'b110 : return rst_op;
       3'b111 : return rst_op;
     endcase // case (op_choice)
  endfunction : get_op
```

```
function byte get_data();
      bit [1:0] zero_ones;
      zero_ones = $random;
      if (zero_ones == 2'b00)
        return 8'h00;
      else if (zero_ones == 2'b11)
        return 8'hFF;
      else
        return $random;
   endfunction : get_data
```

SUNY – New Paltz
Elect. & Comp. Eng.

36

## Module tester (Cont.)

```
initial begin
      byte         unsigned        iA;
      byte         unsigned        iB;
      operation_t                  op_set;
      shortint     result;

      bfm.reset_alu();
      repeat (1000) begin : random_loop
         op_set = get_op();
         iA = get_data();
         iB = get_data();
         bfm.send_op(iA, iB, op_set, result);
      end : random_loop
      $stop;
   end // initial begin
endmodule : tester
```

SUNY – New Paltz
Elect. & Comp. Eng.

37

## Module Scoreboard

```
module scoreboard(tinyalu_bfm bfm);
   import tinyalu_pkg::*;

   always @(posedge bfm.done) begin
      shortint predicted_result;
      #1;
      case (bfm.op_set)
        add_op: predicted_result = bfm.A + bfm.B;
        and_op: predicted_result = bfm.A & bfm.B;
        xor_op: predicted_result = bfm.A ^ bfm.B;
        mul_op: predicted_result = bfm.A * bfm.B;
      endcase // case (op_set)

      if ((bfm.op_set != no_op) && (bfm.op_set != rst_op))
        if (predicted_result != bfm.result)
          $error ("FAILED: A: %0h  B: %0h  op: %s result: %0h",
                  bfm.A, bfm.B, bfm.op_set.name(), bfm.result);
   end
endmodule : scoreboard
```

SUNY – New Paltz
Elect. & Comp. Eng.

38

## Module coverage

```
module coverage(tinyalu_bfm bfm);
  import tinyalu_pkg::*;
   byte         unsigned        A;
   byte         unsigned        B;
   operation_t  op_set;
   covergroup op_cov;
      coverpoint op_set {
         bins single_cycle[] = {[add_op : xor_op], rst_op,no_op};
         bins multi_cycle = {mul_op};

         bins opn_rst[] = ([add_op:mul_op] => rst_op);
         bins rst_opn[] = (rst_op => [add_op:mul_op]);

         bins sngl_mul[] = ([add_op:xor_op],no_op => mul_op);
         bins mul_sngl[] = (mul_op => [add_op:xor_op], no_op);

         bins twoops[] = ([add_op:mul_op] [* 2]);
         bins manymult = (mul_op [* 3:5]);
      }
   endgroup
```

SUNY – New Paltz
Elect. & Comp. Eng.

39

## Module coverage (Cont.)

```
 covergroup zeros_or_ones_on_ops;
      all_ops : coverpoint op_set {
         ignore_bins null_ops = {rst_op, no_op};}
      a_leg: coverpoint A {
         bins zeros = {'h00};
         bins others= {['h01:'hFE]};
         bins ones  = {'hFF};
      }
      b_leg: coverpoint B {
         bins zeros = {'h00};
         bins others= {['h01:'hFE]};
         bins ones  = {'hFF};
      }
      op_00_FF:  cross a_leg, b_leg, all_ops {
         bins add_00 = binsof (all_ops) intersect {add_op} &&
                       (binsof (a_leg.zeros) || binsof
(b_leg.zeros));
         bins add_FF = binsof (all_ops) intersect {add_op} &&
                       (binsof (a_leg.ones) || binsof
(b_leg.ones));
```

SUNY – New Paltz
Elect. & Comp. Eng.

40

## Module coverage (Cont.)

```
  bins and_00 = binsof (all_ops) intersect {and_op} &&
                        (binsof (a_leg.zeros) || binsof
(b_leg.zeros));
          bins and_FF = binsof (all_ops) intersect {and_op} &&
                        (binsof (a_leg.ones) || binsof
(b_leg.ones));
          bins xor_00 = binsof (all_ops) intersect {xor_op} &&
                        (binsof (a_leg.zeros) || binsof
(b_leg.zeros));
          bins xor_FF = binsof (all_ops) intersect {xor_op} &&
                        (binsof (a_leg.ones) || binsof
(b_leg.ones));
          bins mul_00 = binsof (all_ops) intersect {mul_op} &&
                        (binsof (a_leg.zeros) || binsof
(b_leg.zeros));
          bins mul_FF = binsof (all_ops) intersect {mul_op} &&
                        (binsof (a_leg.ones) || binsof
(b_leg.ones));
          bins mul_max = binsof (all_ops) intersect {mul_op} &&
                         (binsof (a_leg.ones) && binsof
(b_leg.ones));
```

SUNY – New Paltz
Elect. & Comp. Eng.

41

## Module coverage (Cont.)

```
  ignore_bins others_only =
        binsof(a_leg.others) && binsof(b_leg.others);
      }
endgroup
   op_cov oc;
   zeros_or_ones_on_ops c_00_FF;
   initial begin : coverage_block
      oc = new();
      c_00_FF = new();
      forever begin  : sampling_block
         @(negedge bfm.clk);
         A = bfm.A;
         B = bfm.B;
         op_set = bfm.op_set;
         oc.sample();
         c_00_FF.sample();
      end : sampling_block
   end : coverage_block

endmodule : coverage
```

SUNY – New Paltz
Elect. & Comp. Eng.

42

## Tb.f

```
tinyalu_pkg.sv
tinyalu_bfm.sv
tester.sv
coverage.sv
scoreboard.sv
top.sv
```

SUNY – New Paltz
Elect. & Comp. Eng.

43