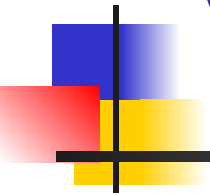
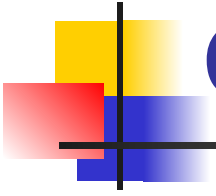


Verification Methodology - Lite



Constrained Random Stimulus

- Directed Testing Problems
  - Time Consuming Test Creation
  - Directed testing detects bugs you expect
  - Can’t think of all potential bug scenarios
- Random Testing
  - Create more interesting stimulus quickly
  - Detects bugs you did not expect

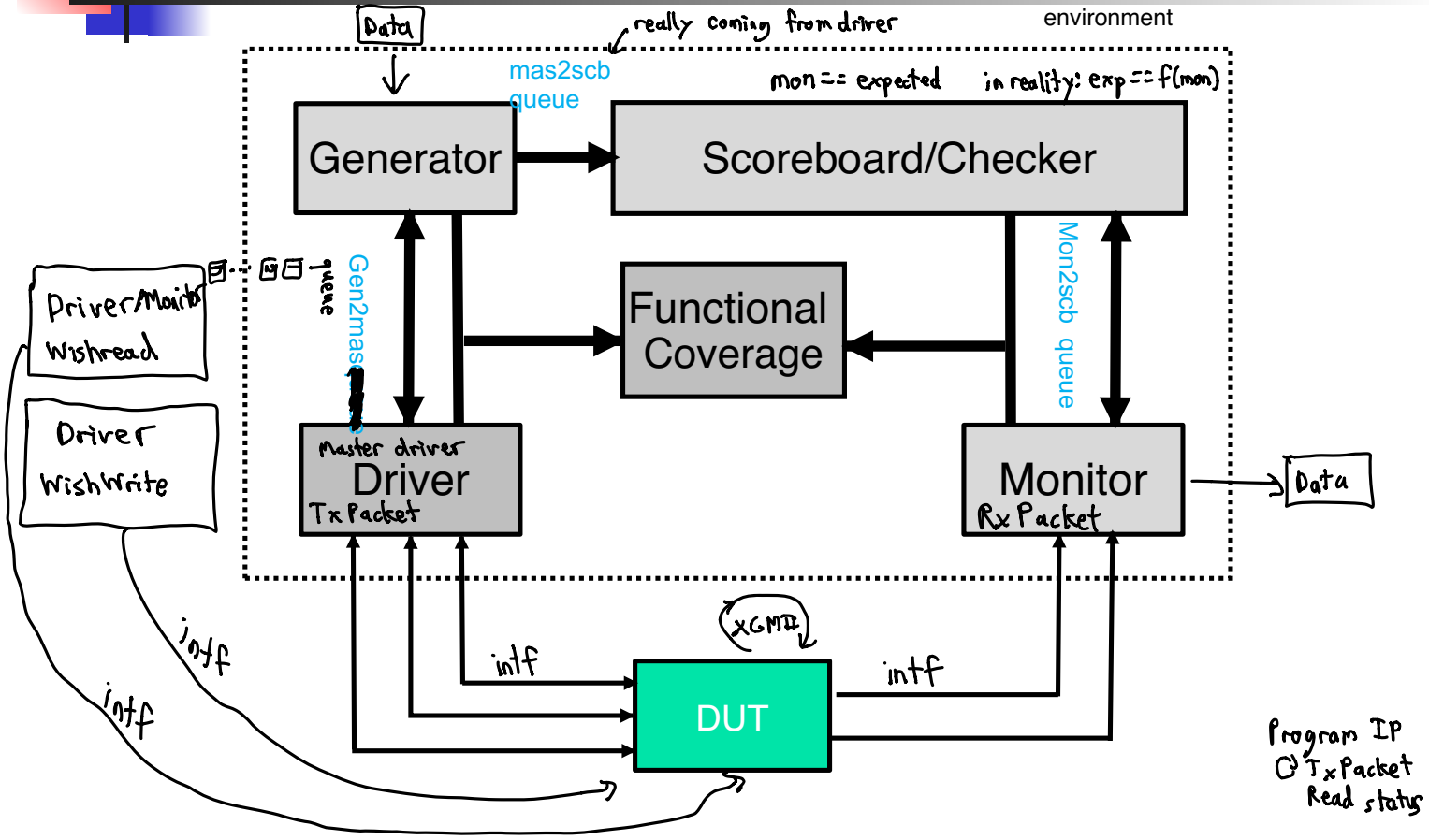
Key Components

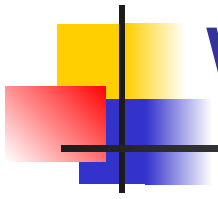
- Data Class
  - Contains all items that can be used to inject stimulus in the design under test
- Generator ← parent of data class
  - Randomizes the data-class
- Driver
  - Applies the random data to some DUT interface
- Monitor
  - Checks the response of the DUT
- Checker/Scoreboard
  - Compares input stimulus and DUT response

TX Packet
Rx Packet

eth_dat

Overview of Components





Why distinct layers?

- Divide and conquer
  - Easier to code separate bits of functionality
  - Re-usable
  - Easier to plug-and-play
  - scalable

Data Class (rt_trans.sv)

holds all random data

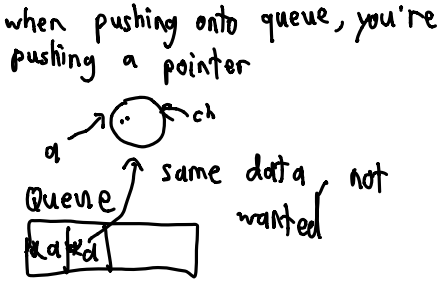
```
class rt_trans;
  rand bit idle;
  rand bit [3:0] delay;
  rand bit [2:0] src,dst;
  rand bit [31:0] payload;
  constraint c2 { delay == 10; }
  constraint c3 { idle == 0; }
  function automatic rt_trans copy();
    rt_trans to   = new();
    to.src        = this.src;
    to.dst        = this.dst;
    to.payload = this.payload;
    to.idle = this.idle;
    to.delay = this.delay;
    copy = to;
  endfunction: copy
endclass
```

Why do we need a copy function?

To avoid same data in the pointer

Handles are passed by reference

```
rt_trans a,b;
rt_trans queue[$];
initial begin
   a = new();
   a.randomize();
   $display("%p",a);
   queue.push_front(a);
   a.randomize();
   $display("%p",a);
   queue.push_front(a);

   b=queue.pop_front();
   $display("%p",b);
   b=queue.pop_front();
   $display("%p",b);
 end
```



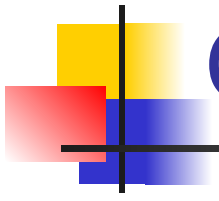
they're the same

```
'{idle:'h0, delay:'ha, src:'h6, dst:'h4, payload:'hcf578554}
'{idle:'h0, delay:'ha, src:'h0, dst:'h3, payload:'h401079a}
'{idle:'h0, delay:'ha, src:'h0, dst:'h3, payload:'h401079a}
'{idle:'h0, delay:'ha, src:'h0, dst:'h3, payload:'h401079a}
```

Handles are passed by reference

```
rt_trans a,b;
rt_trans queue[$];
initial begin
   a = new();
   a.randomize();
   $display("%p",a);
   queue.push_front(a.copy());
   a.randomize();
   $display("%p",a);
   queue.push_front(a.copy());

   b=queue.pop_front();
   $display("%p",b);
   b=queue.pop_front();
   $display("%p",b);
 end
```

```
'{idle:'h0, delay:'ha, src:'h6, dst:'h4, payload:'hcf578554}
'{idle:'h0, delay:'ha, src:'h0, dst:'h3, payload:'h401079a}
'{idle:'h0, delay:'ha, src:'h0, dst:'h3, payload:'h401079a}
'{idle:'h0, delay:'ha, src:'h6, dst:'h4, payload:'hcf578554
```



Generator (own class)

```
while(!end_of_test())
     begin

        tr.randomize();

        trans_cnt++;

        gen2mas.push_front(tr.copy());

 end // while (!end_of_test())
```

Driver (own class)

```
task automatic send1pkt(int i);
    rt_trans tr; → data class
    wait (gen2mas.size!==0);
    tr=gen2mas.pop_front();

    // SEND ADDRESS
    @(rt_if.cb);
    rt_if.cb.frame_n[tr.src] <= 1'b0;
    rt_if.cb.di[tr.src] <= tr.dst[0];
    @(rt_if.cb)  rt_if.cb.di[tr.src] <= tr.dst[1];
    @(rt_if.cb)  rt_if.cb.di[tr.src] <= tr.dst[2];
    @(rt_if.cb)  rt_if.cb.di[tr.src] <= 1'b0;

    // SEND PAYLOAD
    repeat(1) @(rt_if.cb) ;
    for (int i=0;i<32;i=i+1) begin
      rt_if.cb.valid_n[tr.src] <= 1'b0;
      rt_if.cb.di[tr.src] <= tr.payload[i];
      rt_if.cb.frame_n[tr.src] <= i==31;
      @(rt_if.cb);
    end
    // END
    rt_if.cb.valid_n[tr.src] <= 1'b1;
    rt_if.cb.di[tr.src] <= 1'bx;
    repeat (10)  @(rt_if.cb);

 endtask:send1pkt
```

frame_n
valid_n } 3-4

di]31

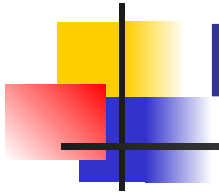
Main task for driver

```
task main();
       reset();

       fork
         forever begin send1pkt (0);  end
         forever begin send1pkt (1);  end
         forever begin send1pkt (2);  end
         forever begin send1pkt (3);  end
         forever begin send1pkt (4);  end
         forever begin send1pkt (5);  end
         forever begin send1pkt (6);  end
         forever begin send1pkt (7);  end
       join
   endtask: main
```

in parallel
all mutually exclusive
apply stimulus
to each channel

Monitor (own class)

```
task automatic rcv1pkt(int i);
   rt_trans tmp = new();

   // SET Destination Address
   tmp.dst=i;
   // BYPASS ADDRESS
   while (rt_intf.cbmon.frameo_n[i]!=='0) @(rt_intf.cbmon) ;
   while (rt_intf.cbmon.valido_n[i]!=='0) @(rt_intf.cbmon) ;

   // GET PAYLOAD
   for (int j=0;j<32;j=j+1) begin
     tmp.payload[j] <= rt_intf.cbmon.dout[i];
     @(rt_intf.cbmon);
   end
   mon2scb.push_front(tmp);
   $display($time,": Received packet on %d",i);

 endtask:rcv1pkt
```



Main task for monitor

```
task main();
     fork
         forever begin  rcv1pkt(0);  end
         forever begin  rcv1pkt(1);  end
         forever begin  rcv1pkt(2);  end
         forever begin  rcv1pkt(3);  end
         forever begin  rcv1pkt(4);  end
         forever begin  rcv1pkt(5);  end
         forever begin  rcv1pkt(6);  end
         forever begin  rcv1pkt(7);  end
       join
     // end
   endtask: main
```

Scoreboard

```
task automatic compare(int i);
    rt_trans mas_tr,mon_tr;

    // GET a PAYLOAD from monitor queue
  → wait (mon2scb.size() !==0 );
  → mon_tr=mon2scb.pop_back();


    foreach (  mas2scb[j] ) begin
       if (mas2scb[j].payload == mon_tr.payload) begin
          $display("Match!!");
          match_cnt++;
          mas2scb.delete(j);
       end
    end
 endtask
```

search for whether transmitted payload is same as monitored payload.

Main task for scoreboard

```
fork  : fork1
      forever compare(0);
      forever compare(1);
      forever compare(2);
      forever compare(3);
      forever compare(4);
      forever compare(5);
      forever compare(6);
      forever compare(7);
      forever begin
        #100; if (max_trans_cnt == match_cnt)  $finish;
      end

    join
```

Environment Class

```
class env;

  // Transactors
  rt_gen     gen;
  rt_master  mst;
  rt_monitor mon;
  scoreboard  scb;

function new(virtual rt_intf rtif_mst, virtual rt_intf rtif
_mon);
    gen      = new(tcfg.trans_cnt, 1);
    mst      = new(rtif_mst, 1);
    mon      = new(rtif_mon, 1);
    scb      = new(tcfg.trans_cnt);
  endfunction: new
```

Environment class methods

```
class env;

  // Transactors
  rt_gen     gen;
  rt_master  mst;
  rt_monitor mon;
  scoreboard  scb;

function new(virtual rt_intf rtif_mst, virtual rt_intf rtif
_mon);
    gen      = new(tcfg.trans_cnt, 1);
    mst      = new(rtif_mst, 1);
    mon      = new(rtif_mon, 1);
    scb      = new(tcfg.trans_cnt);
  endfunction: new
```

Environment class methods

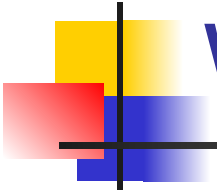
```
virtual task pre_test();
   fork
     scb.main();
     mst.main();
     mon.main();
   join_none
endtask: pre_test


virtual task test();
  mst.reset();
  fork
    gen.main();
  join_none
endtask: test

virtual task post_test();
  fork
    wait(gen.ended.triggered);
    wait(scb.ended.triggered);
  join
endtask: post_test
```

Top Level

```
module top();

logic clock=0;
always #5 clock=!clock;

rt_intf rt_intf(clock);

router dut (.clock(clock), .reset_n(rt_intf.reset_n),
.frame_n(rt_intf.frame_n),
.valid_n(rt_intf.valid_n),
.di(rt_intf.di),
.dout(rt_intf.dout),
.valido_n(rt_intf.valido_n),
.frameo_n(rt_intf.frameo_n));

test test(rt_intf);
```

Test Layer

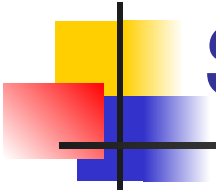
```
rt_trans gen2mas[8][$], mas2scb[8][$], mon2scb[8][$];

module test(rt_intf rt_intf);


// Top level environment
env the_env;

initial begin
  the_env = new(rt_intf, rt_intf);

  // Kick off the test now
  the_env.run();

end
```



Why All the Effort?

The four principles of object-oriented programming are

- encapsulation
- Abstraction
- Inheritance (re-use)
- Polymorphism (runtime-lookup)



Simple Example

Abstract class : only defines the public methods; defines invariant level of functionality

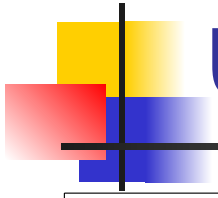
- Cannot be instantiated
- Only useful if derived classes defined

```
virtual class object ;
  virtual function string display();
  endfunction
  virtual function logic [31:0] area();
  endfunction
endclass
```

Derived Classes

```
class rectangle extends object ;
  logic [31:0] x,y;
  function new(int x,y);
    this.x=x;
    this.y=y;
  endfunction
  function string display();
    display="RECTANGLE";
  endfunction
  function logic [31:0] area();
    area=x*y;
    endfunction
endclass
```

```
class circle extends object;
  logic [31:0] r;
  function new(int r);
    this.r=r;
  endfunction
  function string display();
    display="CIRCLE";
  endfunction
  function logic [31:0] area();
    area=3.1415*r*r;
  endfunction
endclass
```



Usage

```
module test;

object o1;
rectangle r1;
circle c1;
initial begin
  o1=new();      // illegal
  r1=new(2,3);
  c1=new(2);


  o1 = r1;
  $display("%s area is %d",o1.display(),o1.area()) ;
  o1 = c1;
  $display("%s area is %d",o1.display(),o1.area()) ;

end

endmodule
```

New Generator

```
class rt_gen_directed extends rt_gen;
  function new(input int max_trans_cnt, input bit verbose=0);
    super.new(max_trans_cnt,verbose);
  endfunction

  task main();
    if(verbose)
      $display($time, ": Starting rt_gen for %0d transactions",
               max_trans_cnt);

    while(!end_of_test())
      begin

      → tr.randomize() with { src==0; dst==7;} ;

          ++trans_cnt;

           gen2mas.push_front(tr.copy());
      end // while (!end_of_test())
    ->ended;

  endtask
endclass
```

New Environment

only modification of test layer
channel 0 → channel 7

```
module test(rt_intf rt_intf);

   env the_env;
   rt_gen_directed new_gen;
initial begin

  the_env = new(rt_intf, rt_intf);
  new_gen = new(the_env.tcfg.trans_cnt,1);
  the_env.gen = new_gen;
  // Kick off the test now
  the_env.run();

end

endmodule
```

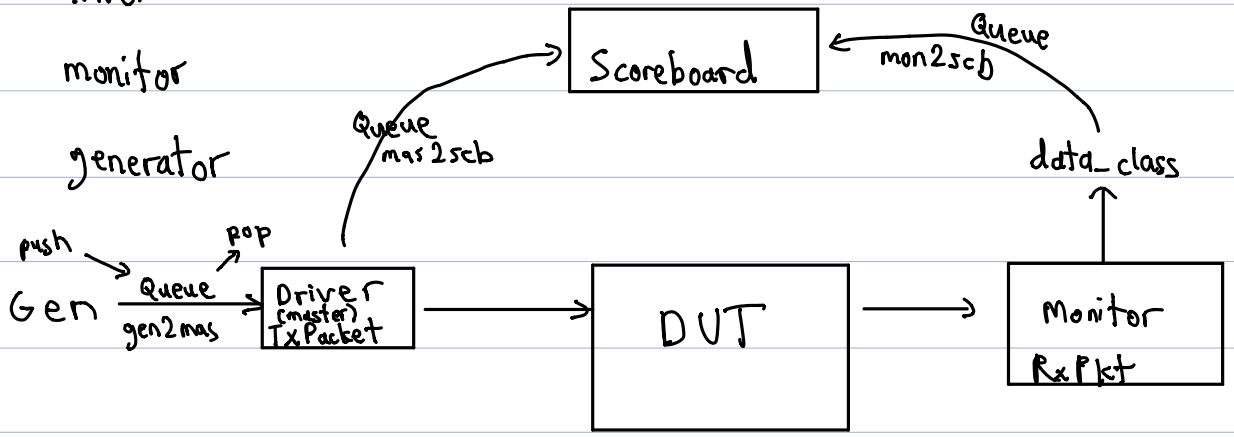
environment class:

scoreboard

driver

monitor

generator



UART

ETHERNET

Rt_layered_testbench