LAB3: Verify Router Design Using SystemVerilog Testbench

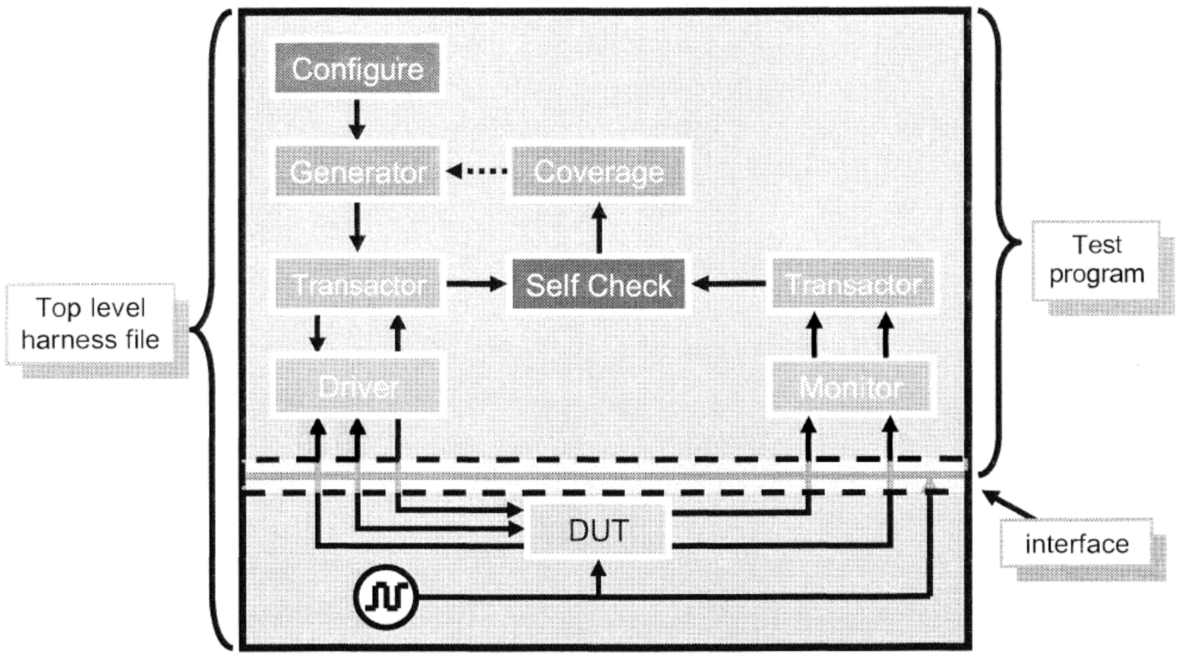
Learning Objectives

After completing this lab, you should be able to:

- Build SystemVerilog testbench for verifying Router design
- Simulate Router design and debug design errors

Description

A typical structure of a SystemVerilog testbench as the following:

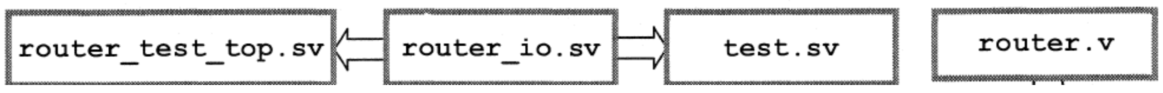


The process to create the SystemVerilog is as follows:

Create an interface to connect test program and DUT

Write test program

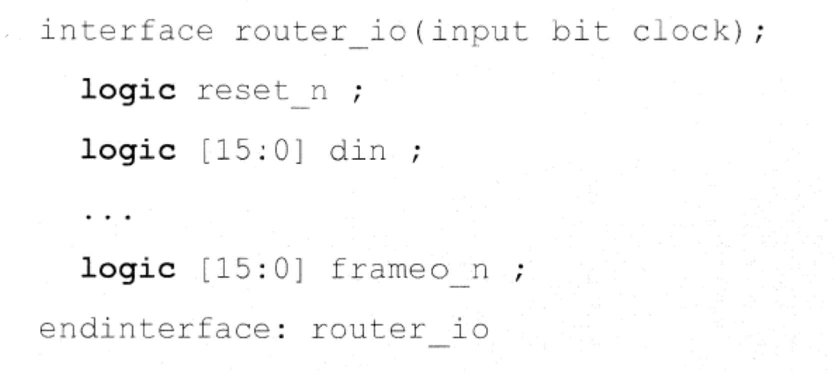
Connect the test to the DIT and using the harness Tesbench including the interface



Tasks

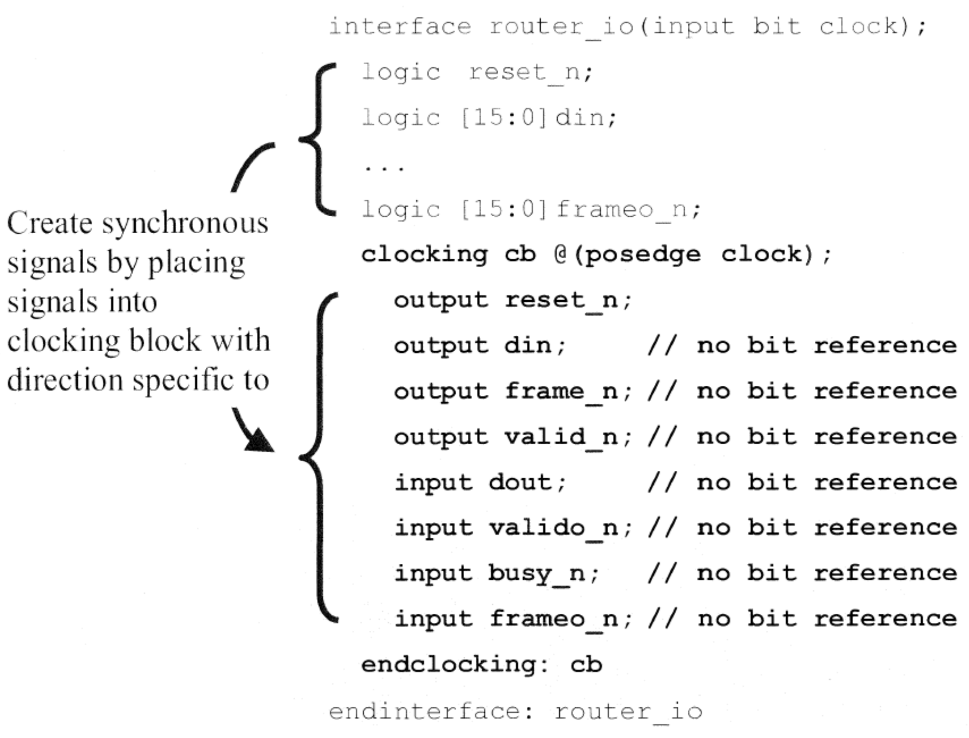
1. Create SystemVerilog interface

The signals needed to connect to the DUT



```
interface router_io(input bit clock);
  logic reset_n ;
  logic [15:0] din ;
  ...
  logic [15:0] frameo_n ;
endinterface: router_io
```

Declare a clocking block driven by the posedge of the signal clock



Create synchronous signals by placing signals into clocking block with direction specific to

```
interface router_io(input bit clock);
  logic  reset_n;
  logic [15:0]din;
  ...
  logic [15:0]frameo_n;
  clocking cb @(posedge clock);
    output reset_n;
    output din;      // no bit reference
    output frame_n; // no bit reference
    output valid_n; // no bit reference
    input dout;      // no bit reference
    input valido_n; // no bit reference
    input busy_n;   // no bit reference
    input frameo_n; // no bit reference
  endclocking: cb
endinterface: router_io
```

Create a modport to used for connection with the test program

```
interface router_io(input bit clock);
  ...
  clocking cb @(posedge clock);
    default input #1 output #1;
    output reset_n;
    output din;     // no bit reference
    ...
  endclocking: cb
  modport TB(clocking cb, output reset_n);
endinterface: router_io
```

2. Create SystemVerilog test program

Declare a test program block with arguments which connects to the TB modport in the interface block

```
program automatic test(router_io.TB rtr_io);
  initial begin
    $display("Hello World!");
  end
endprogram: test
```

Create SystemVerilog Harness Testbench

```
module router_test_top;
  parameter simulation_cycle = 100;
  bit  SystemClock;
  router dut(
    .reset_n    (reset_n),
    .clock      (clock),
    .din        (din),
    .frame_n    (frame_n),
    .valid_n    (valid_n),
    .dout       (dout),
    .valido_n   (valido_n),
    .busy_n     (busy_n),
    .frameo_n   (frameo_n)
  );
  initial begin
    SystemClock = 0;
    forever begin
      #(simulation_cycle/2)
        SystemClock = ~SystemClock;
    end
  end
endmodule
```

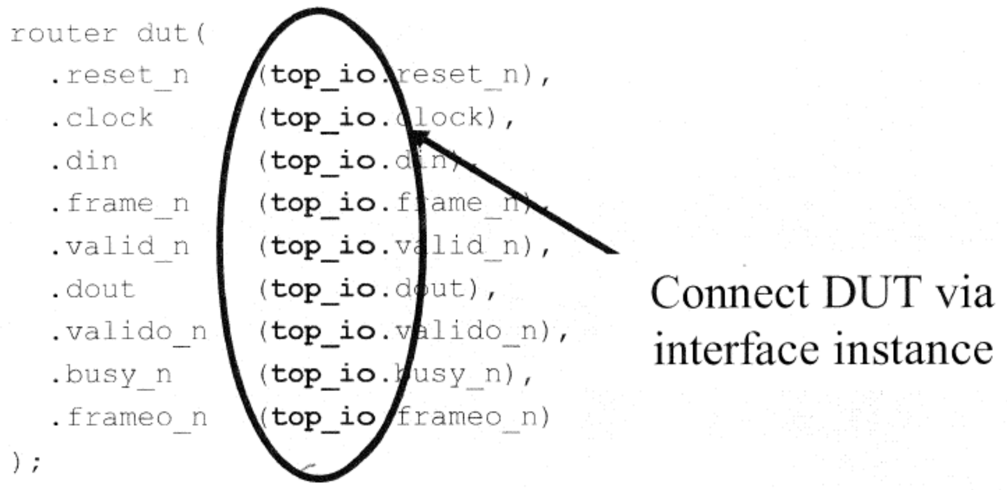
Add an interface instance to the harness testbench

```
module router_test_top;
  parameter simulation_cycle = 100;
  bit  SystemClock;
  router_io top_io(SystemClock);  <-- Instantiate interface
  router dut( … );
  initial begin
    SystemClock = 0;
    ...
  end
endmodule
```

Instantiate the test program

```
module router_test_top;
  parameter simulation_cycle = 100;
  bit  SystemClock;
  router_io top_io(SystemClock);  // instantiating interface
  test t(top_io); // add program
  router dut( … );
  initial begin
    SystemClock = 0;
    ...
  end
endmodule
```

Modify DUT connection to connect via interface



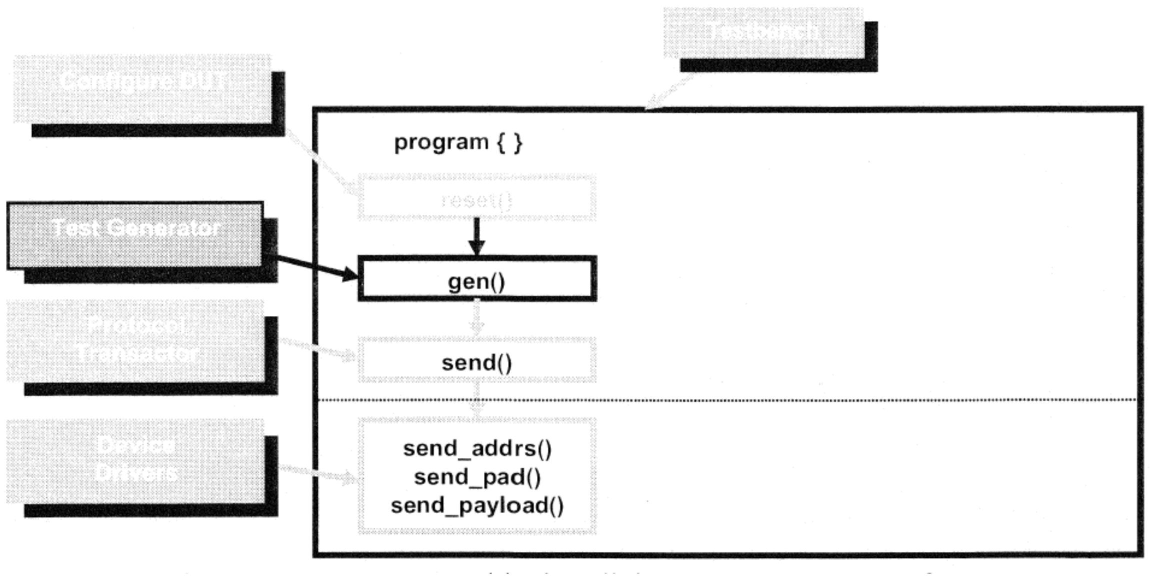
```
router dut(
   .reset_n     (top_io.reset_n),
   .clock       (top_io.clock),
   .din         (top_io.din),
   .frame_n     (top_io.frame_n),
   .valid_n     (top_io.valid_n),
   .dout        (top_io.dout),
   .valido_n    (top_io.valido_n),
   .busy_n      (top_io.busy_n),
   .frameo_n    (top_io.frameo_n)
);
```

Connect DUT via interface instance

Add timescale

```
`timescale 1ns/100ps
module router_test_top;
```

3. Send Packet Through Router



In the program initial block, call the generator gen() after reset()

```
program automatic test(router_io.TB rtr_io);
  bit[3:0] sa;           // source address
  bit[3:0] da;           // destination address
  logic[7:0] payload[$]; // packet data array

  initial begin
    $vcdpluson;
    reset();
    gen();
  end

  task reset();
    ...
  endtask: reset

  task gen();

  endtask: gen
endprogram: test
```

```
task gen();
  sa = 3;
  da = 7;
  payload.delete();
  repeat($urandom_range(2,4))
    payload.push_back($urandom);
endtask: gen
```

Create Send Packet routines

Send destination address, padding bits, payload

```
program automatic test(router_io.TB rtr_io);
  bit[3:0] sa;           // source address
  bit[3:0] da;           // destination address
  logic[7:0] payload[$]; // packet data array

  initial begin
    $vcdpluson;
    reset();
    gen();
  end

  task reset();
    ...
  endtask: reset

  task gen();

  endtask: gen
endprogram: test
```

In initial block, call send() to send packet after the gen()

Add a delay after send()

Add send() task in the program block

In the body of send() task, call: send_addrs, send_pad, send_payload()

Create send() task

In the body of send() task:

+ drive frame_n signal as spec

+ drive din signal with destination address (LSB first)

Example: driving frame_n signal for input port 3 to "1" as following form

```
rtr_io.cb.frame_n[3] <= 1'b1;
```

Create send_pad() task

Create send_payload() task

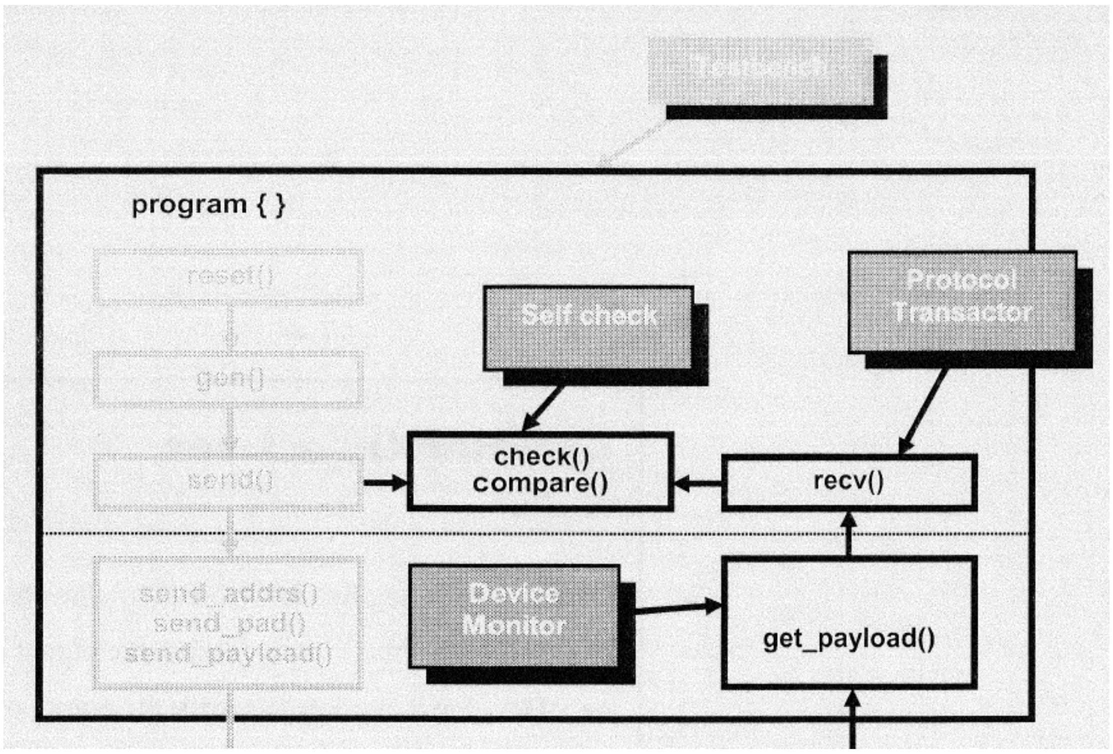
In body of send_paylod() task

+ Write a loop to execute payload.size()

+ In the loop, each 8-bit data of the payload[$] array should be transmitted one bit per clock cycle starting with lsb

Extend the program to send 21 packets

4. Self-checking



Modify program test

Add a global declaration for an 8-bit (logic [7:0]) queue named pkt2ccmp_payload[$]

This queue will be use to store the data sampled from DUT

Add recv() and send() tasks followed by a self-checking routine check()

```
program automatic router_test(router_io.TB router);
  ...
  logic[7:0] pkt2cmp_payload[$];

  initial begin
    ...
    repeat(run_for_n_packets) begin
      gen();
      fork
        send();
        recv();
      join
      check();
    end
  end
  ...
endprogram
```

In body of recv() task:

+call get_payload() task to retrieve a packet payload from router. This payload should stored in pkt2cmp_payload[$] queue

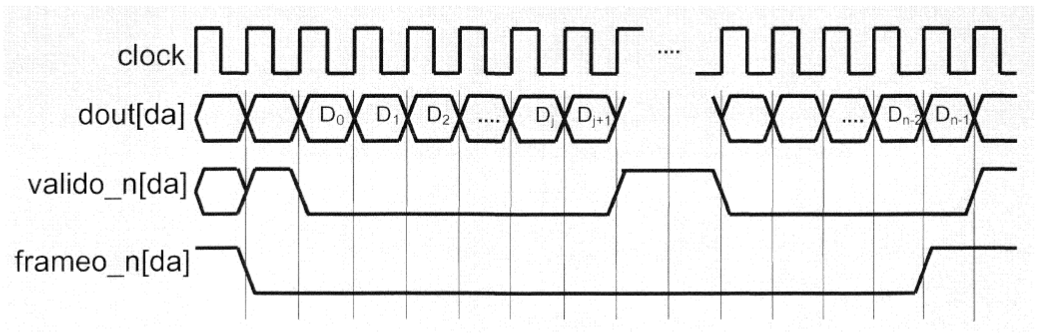
Declare get_payload() task

In get_payload() task,

+store each 8-bit data into pkt2cmp_payload[$]

+waiting for the falling edge of the output frame signal, example

```
@(negedge router.cb.frameo_n[da])
```



Loop until end of frame is detected

Develop the checker

Create a function named compare() which returns a single bit

In the body of compare(), compare data in payload[$] and pkt2cmp_payload[$] to verify that the payload received correctly

In the body of check() task, call compare() function to check the packet received

+if error, print error message and finish simulation

+if check is successful, print message indicating number of packets successfully checked

Expand to detect RTL Errors

*Note: remove $vcdpluson in sample codes, this is a system task of VCS tool.

--END--