3 Self-Checking

Learning Objectives

After completing this lab, you should be able to:

- Develop a monitor to sample the output of the router
- Develop a checker to verify the output of the router
- Run driver and monitor routines concurrently
- Verify the self-checking mechanism by executing the testbench against a faulty DUT

Lab Duration: 90 minutes

Getting Started

In Lab 3, you will add in the monitors and self-check mechanisms. The architecture is shown below:

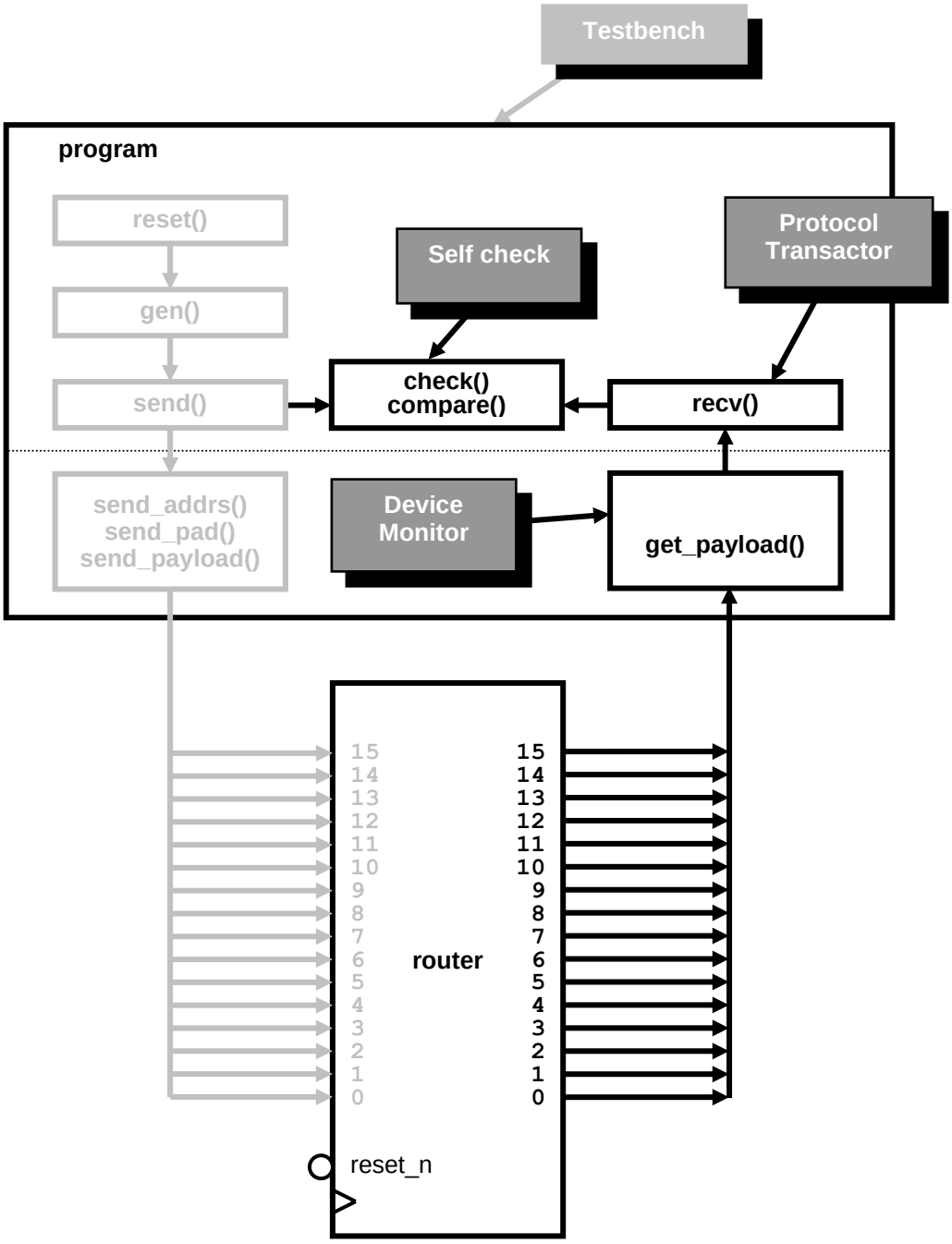


Figure 1. Lab 3 testbench architecture

Lab Overview

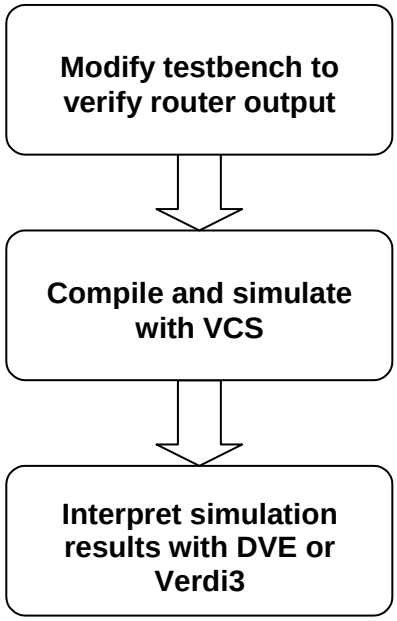


Figure 2. Lab 3 Flow Diagram

Note: You will find Answers for all questions and solutions in the Answers / Solutions at the end of this lab.

Self-Checking & Concurrency

Task 1. Copy Files From Lab 3's Solutions Directory

For consistency, you will copy the files from **../../solutions/lab3** into the **lab3** directory.

1. Go into the lab3 directory.

```
> cd ../lab3
```

2. Copy the source files in the **solutions/lab3** directory into the current directory with **make** script.

```
> make copy
```

(If you chose to use your own lab files from lab2, type "**make mycopy**".)

Task 2. Build The Top-Level Test Environment

In the following steps, you will complete building of the top-level test environment.

1. Edit **test.sv** file.

2. Add a global declaration for an 8-bit (**logic[7:0]**) queue named: **pkt2cmp_payload[$]**

This queue will be used to store the data sampled from the DUT.

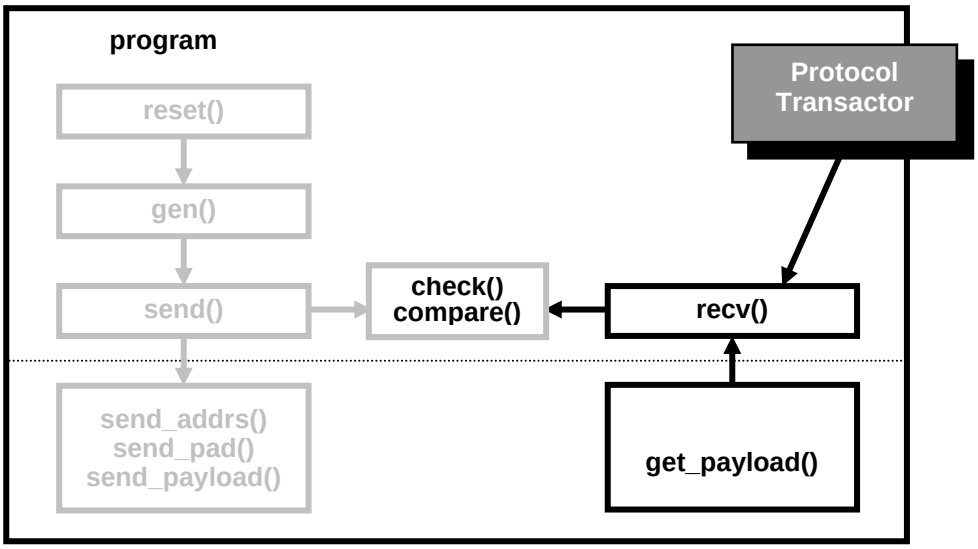
3. For self-checking, modify the program to execute a receive routine **recv()** concurrently with **send()** followed by a self-checking routine **check()**.

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

  initial begin
    ...
    repeat(run_for_n_packets) begin
      gen();
      fork
        send();
        recv();
      join
      check();
    end
    repeat(10) @rtr_io.cb;
  end
  ...
endprogram
```

Task 3. Develop Monitor Routines

In this step, you will build the monitor transactor called **`recv()`**.



This **`recv()`** transactor will call device monitor **`get_payload()`** to retrieve a packet payload from the router. This payload shall be stored in **`pkt2cmp_payload[$]`** queue by the **`get_payload()`** routine.

1. Declare the **`recv()`** task.
2. In the body of **`recv()`** call **`get_payload()`** to retrieve the payload.

   For now, this is the only content of the **`recv()`** routine. More will be added in later labs.
3. Declare the **`get_payload()`** task.
4. In **`get_payload()`**, delete content of **`pkt2cmp_payload[$]`**.

   This is necessary to remove potential residues from previous packet.
5. Continuing in **`get_payload()`**, wait for the falling edge of the output frame signal.

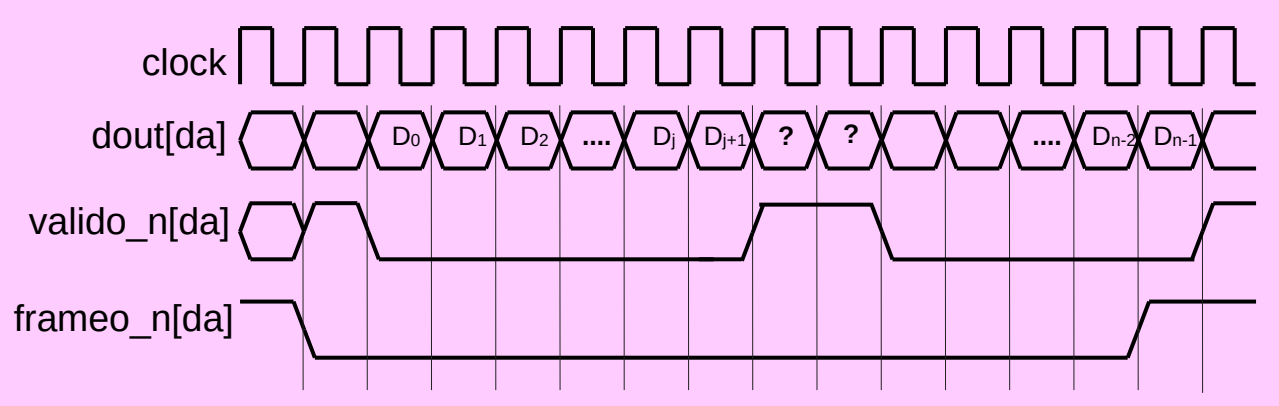
Question 1. How do you implement a watchdog timer for detecting the negative edge of **`rtr_io.cb.frameo_n[da]`**
Hint: Remember to put it in an enclosing fork-join to prevent disable fork problems discussed in the class slides. Also remember the caveats against using
`@(negedge rtr_io.cb.frameo_n[da])`

…….

…….

…….

…….



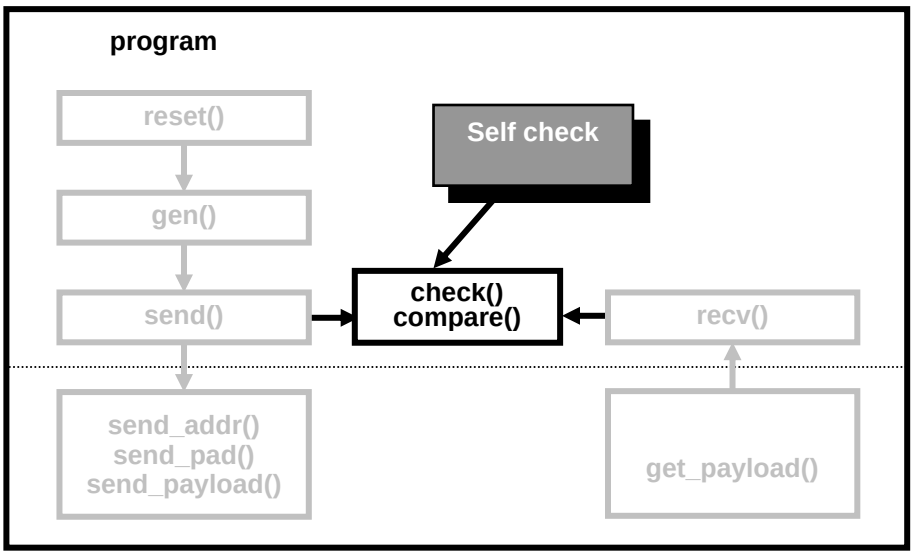
6. Continue to develop **get_payload()** routine by sampling the output of router:

- Loop until end of frame is detected.
- Within the loop, assemble a byte of data at a time (minimum of 8 clock cycles). Then, store each 8-bit datum into the **`pkt2cmp_payload[$]`** queue.

7. If payload is not byte aligned, print a message to terminal and end simulation.

Task 4. Develop The Checker

The last step in completing this basic testbench is to develop the checker for checking the output of the router.



1. Create a function called **`compare()`** which returns a single bit and has a pass-by-reference string argument.
2. In the body of **`compare()`**, compare the data in **`payload[$]`** (reference packet data) and **`pkt2cmp_payload[$]`** (sampled packet data) to verify that the payload is received correctly.
   - If the sizes of the **`payload[$]`** and **`pkt2cmp_payload[$]`** arrays do not match, set the string argument with a description of the error, return a value of **`0`** and terminate the subroutine.
   - If the data in **`payload[$]`** and **`pkt2cmp_payload[$]`** arrays do not match, set the string argument with a description of the error, return a value of **`0`** and terminate the subroutine. You can directly compare arrays in SystemVerilog using the **`==`** operator.
   - If the data in **`payload[$]`** and **`pkt2cmp_payload[$]`** arrays are the same, set the string argument with a description of the success, return a value of 1 and terminate the subroutine.
3. Create a function called **`check()`** which returns a void type**.**
4. In the body of **`check()`**, declare a string variable message and a counter for packets **`pkts_checked`**.

5. In the body of **`check()`**, call the **`compare()`** function to check the packet received.
   - If an error is detected, print the error message to teminal then end simulation.
   - If check is successful, print a message indicating the number of packets successfully checked to terminal.
6. Save and close the file.

Task 5. Compile and Run

1. Use **`make`** script to compile and run your program.
2. Make sure the simulation completes successfully. Correct all errors.

Task 6. Test All Ports

1. Modify your test program to randomly generate **`sa`** and **`da`**.
2. Extend your testbench to send **2000** packets.
3. Use **`make`** script to compile and run your program.
4. Make sure the simulation still completes successfully.

Task 7. Expand To Detect RTL Error

1. Run your testbench against a bad rtl code

A bad RTL code has been included in the rtl directory. The script to run your testbench is already embedded in the Makefile. To run the script, type the following command.

Compile and run simulation

```
> make bad
```

If your simulation stops on an error you are done. If not, your testbench is not working properly. Correct your testbench and try again.

Congratulations, you completed Lab 3!

Answers / Solutions

Task 3. Develop Monitor Routines

5. Continuing in **`get_payload()`**, wait for the falling edge of the output frame signal.

Question 2. How do you implement a watchdog timer for negative edge of **`rtr_io.cb.frameo_n[da]`**

There is a potential that simulation may run forever if there is an error in testbench coding.

Consider the following code:

```
for (i = 0; i < run_for_n_packets; i++)
  gen();
  fork
    begin
      send();
      recv();
    end
  join
```

The **`send()`** and **`recv()`** routines were supposed to be called concurrently inside a **`fork-join`** construct. The coder mistakenly put both inside a **`begin-end`** construct making execution of **`send()`** and **`recv()`** serial rather than concurrent.

If you simply use the negative edge of **`rtr_io.cb.frameo_n[da]`** to detect the beginning of a new packet at the router's output the simulation would run forever, you would never know that there was a mistake or what the mistake was.

To prevent this type of runaway simulation, you can add a watchdog timer and have the routines timeout if a problem similar to the example occurs. The following code sample illustrates just such a watchdog timer implementation:

```
fork
  begin: wd_timer_fork
  fork: frameo_wd_timer
    begin //see class notes to understand this block
      wait(rtr_io.cb.frameo_n[da] !== 0);
      @(rtr_io.cb iff(rtr_io.cb.frameo_n[da] === 0 ));
    end
    begin
      repeat(1000) @rtr_io.cb;
      $display("\n%m\n[ERROR]%t Frame signal timed out!\n", $realtime);
      $finish;
    end
  join_any: frameo_wd_timer
  disable fork;
  end: wd_timer_fork
join
```

test.sv Solution:

```
program automatic test(router_io.TB rtr_io);
  int run_for_n_packets;              // number of packets to test
  bit[3:0] sa;                        // source address
  bit[3:0] da;                        // destination address
  logic[7:0] payload[$];              // expected packet data array
  logic[7:0] pkt2cmp_payload[$];      // actual packet data array


  initial begin
    $vcdpluson;

    run_for_n_packets = 2000;
    reset();
    repeat(run_for_n_packets) begin

      gen();

      fork
        send();
        recv();
      join

      check();

    end
    repeat(10) @rtr_io.cb;

  end

  task reset();
    rtr_io.reset_n = 1'b0;
    rtr_io.cb.frame_n <= '1;
    rtr_io.cb.valid_n <= '1;
    repeat(2) @rtr_io.cb;
    rtr_io.cb.reset_n <= 1'b1;
    repeat(15) @rtr_io.cb;
  endtask: reset

  task gen();

    sa = $urandom;
    da = $urandom;
    payload.delete(); //clear previous data
    repeat($urandom_range(4,2))
      payload.push_back($urandom);

  endtask: gen

  task send();

    send_addrs();
    send_pad();
    send_payload();

  endtask: send
```

Continued…

…Continued from previous page

```
  task send_addrs();

     rtr_io.cb.frame_n[sa] <= 1'b0; //start of packet
     for(int i=0; i<4; i++) begin
       rtr_io.cb.din[sa] <= da[i]; //i'th bit of da
       @rtr_io.cb;
     end

  endtask: send_addrs

  task send_pad();

     rtr_io.cb.frame_n[sa] <= 1'b0;
     rtr_io.cb.din[sa] <= 1'b1;
     rtr_io.cb.valid_n[sa] <= 1'b1;
     repeat(5) @rtr_io.cb;

  endtask: send_pad

  task send_payload();

     foreach(payload[index])
       for(int i=0; i<8; i++) begin
         rtr_io.cb.din[sa] <= payload[index][i];
           rtr_io.cb.valid_n[sa] <= 1'b0; //driving a valid bit
           rtr_io.cb.frame_n[sa] <= ((i == 7) && (index == (payload.size() - 1)));
           @rtr_io.cb;
       end
     rtr_io.cb.valid_n[sa] <= 1'b1;
  endtask: send_payload

  task recv();

    //Lab 3 - Task 3, Step 2
    //
    //In recv() task call get_payload() to retrieve payload
    //ToDo
     get_payload();

  endtask: recv

  task get_payload();

    pkt2cmp_payload.delete();

    fork
      begin: wd_timer_fork
      fork: frameo_wd_timer
      begin
        wait(rtr_io.cb.frameo_n[da] !== 0);
        @(rtr_io.cb iff(rtr_io.cb.frameo_n[da] === 0 ));
      end
     begin                                //this is another thread
      repeat(1000) @rtr_io.cb;
      $display("\n%m\n[ERROR]%t Frame signal timed out!\n", $realtime);
      $finish;
     end
```

Continued…

```
…Continued from previous page
    join_any: frameo_wd_timer
      disable fork;
      end: wd_timer_fork
    join

    forever begin
      logic[7:0] datum;
      for(int i=0; i<8; i=i)  begin //i=i prevents VCS warning messages
        if(!rtr_io.cb.valido_n[da])
          datum[i++] = rtr_io.cb.dout[da];
        if(rtr_io.cb.frameo_n[da])
          if(i==8) begin //byte alligned
            pkt2cmp_payload.push_back(datum);
            return;      //done with payload
           end else begin
            $display("\n%m\n[ERROR]%t Packet payload not byte aligned!\n",
$realtime);
            $finish;
           end
        @rtr_io.cb;
      end
      pkt2cmp_payload.push_back(datum);
    end
  endtask: get_payload

  function bit compare(ref string message);

    if(payload.size() != pkt2cmp_payload.size()) begin
      message = "Payload size Mismatch:\n";
      message = { message, $sformatf("payload.size() = %0d,
       pkt2cmp_payload.size() = %0d\n", payload.size(), pkt2cmp_payload.size()) };
      return (0);
    end
    if(payload == pkt2cmp_payload) ;
    else begin
      message = "Payload Content Mismatch:\n";
      message = { message, $sformatf("Packet Sent:   %p\nPkt Received:   %p",
payload, pkt2cmp_payload) };
      return (0);
    end
    message = "Successfully Compared";
    return(1);
  endfunction: compare

  function void check();

    string message;
    static int pkts_checked = 0;

    if (!compare(message)) begin
      $display("\n%m\n[ERROR]%t Packet #%0d %s\n", $realtime, pkts_checked,
                message);
      $finish;
    end
    $display("[NOTE]%t Packet #%0d %s", $realtime, pkts_checked++, message);
  endfunction: check

endprogram: test
```