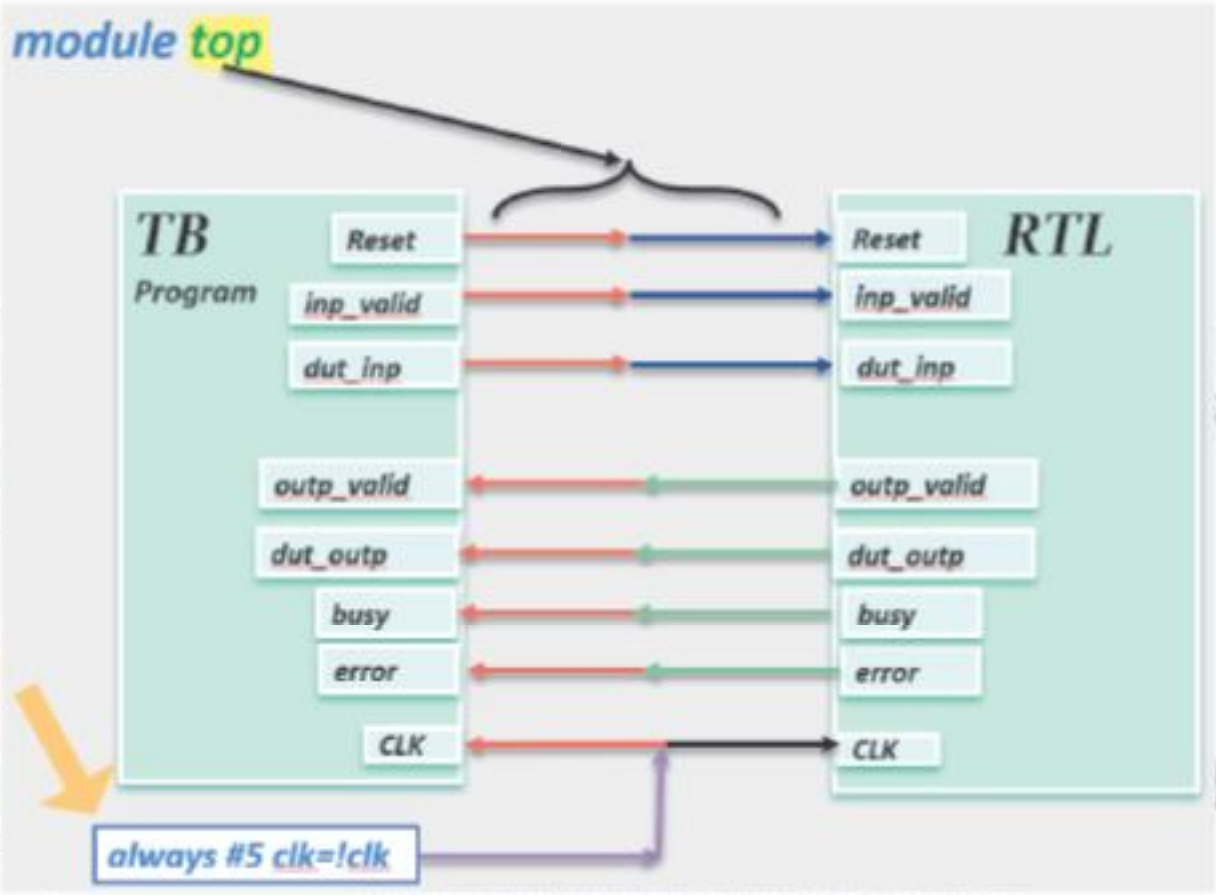
LUCID VLSI

Learning Unlimited. Coding InDelible.

5 SystemVerilog LAB5

After completing this lab, you should be able to:

Implement testbenches using program block.
Understand how to define I/O Signals in program block.
Understand how to Specify directions of the signals in program block.
Implement top module which connects dut and tb.
Verify DUT behavior with the help of self-checking mechanism in program block(testbench).



LUCID VLSI

Learning Unlimited. Coding InDelible.

LAB5: Use lab5 directory for this lab

Step 1: Convert module testbench to program block based testbench.

1) Open file testbench.sv and define program block with required ports.
   Refer specification to know the required signals (I/O Pins).
   Add this code in Section 1 in testbench.sv
   Ex:

```
program testbench(clk,reset,dut_inp,inp_valid,
                  dut_outp,outp_valid,busy,error);
.....
.....
endprogram
```

2) Define signal directions for the above added port list.
   1) Define signal as output if your requirement is to drive signal in TB.
   2) Define signal as input if your requirement is to sample signal in TB

Example: Add this code in Section1 in testbench.sv.

```
input clk;
output reset;
output [7:0] dut_inp;
.....
//define direction for all ports
//Signal directions in TB are opposite to DUT port directions
```

Step 2: Define TB variables required to testbench purpose.

1) Copy the entire Section 4 from lab4/testbench.sv file to lab5/testbench.sv.

Step 3: Define required TB methods.

1. Copy entire Section5 from lab4/testbench.sv file to lab5/testbench.sv.

Step 4: Collect DUT output in TB (program block).

1. Copy entire Section8 from lab4/testbench.sv file to lab5/testbench.sv.

Step 5: Start the Verification flow.

1) Copy entire Section6 from lab4/testbench.sv file to lab5/testbench.sv

Example:

```
initial begin
pkt_count=10;
apply_reset();
    repeat(pkt_count) begin
    inp_stream.delete();
    wait(busy==0);
    generate_stimulus(stimulus_pkt);
    pack(inp_stream,stimulus_pkt);
    q_inp.push_back(stimulus_pkt);
    drive(inp_stream);
    repeat(5) @(posedge clk);
    end
//Wait for dut to process the packet and to drive on output
wait(busy==0);//drain time
repeat(10) @(posedge clk);//drain time
result();
$finish;
end
```

Step 6: Define module top and instantiate DUT and TB.

1) Open file top.sv and define module top with required bunch of signals to connect DUT(rtl router) and TB(program block).
   Refer specification to know the required signals (I/O Pins).
   Add this code in Section 1 in top.sv
   Ex:

```
module top();
logic clk;
logic reset;
logic [7:0] dut_inp;
.....
.....
endmodule
```

2) Initialize clk to 0 in initial block and generate clock.
   Add this code in Section 2 in top.sv
   Example:

```
initial clk=0;
always #5 clk = ~clk;
```

3) Instantiate router_dut with instance name dut_inst along with port connections. Use name based port mapping.
   Add this code in Section 3 in top.sv
   Ex: router_dut dut_inst
   (.clk(clk),.reset(reset),.....);

4) Instantiate program block with instance name tb_inst along with port connections. Use name based port mapping.
   Add this code in Section 4 in top.sv
   Ex: testbench tb_inst (.clk(clk),.reset(reset),.....);

LUCID VLSI

Learning Unlimited. Coding InDelible.

Step 7: Run the simulation and validate the output of DUT with the results printed by self-checking mechanism.

1. *Compile top.sv testbench.sv and router_dut.sv files*
2. *Check the test Passed or Failed and debug the if there are any failures.*