LUCID VLSI

Learning Unlimited. Coding InDelible.

10 SystemVerilog LAB10

After completing this lab, you should be able to:

Implement class with variables/methods.
Implement transaction class.
Implement stimulus generation method using class as argument.
Construct object of class and call the methods from it.
Generate stimulus by passing objects as arguments.
Verify DUT behavior with the help of self-checking mechanism in program block(testbench).

LAB10: Use lab10 directory for this lab

Step 1: Implement transaction class with name packet with "Packet format" in TB (refer topic3 slides).

1. Create new file packet.sv.
2. Convert existing structure packet format to class packet. (refer topic2 slides and dut specification).
3. Define the class packet with all the fields from structure packet.

Ex : Add this code in packet.sv

```
class packet ;
        bit [7:0] sa;
        bit [7:0] da;
        bit [31:0] len;
        bit [31:0] crc;
        bit [7:0] payload[];

        bit [7:0] inp_stream[$];
        bit [7:0] outp_stream[$];
endclass
```

Step 2: Implement pack method in class packet.

1) Define pack method in class packet.

Example: Add this code in packet.sv

```
function void pack(ref bit [7:0] q_inp[$]);
  q_inp = {<< 8 {this.payload,this.crc,this.len,this.da,this.sa}};
endfunction
```

Step 3: Implement unpack method in class packet.

1) Define unpack method in class packet.

Example: Add this code in packet.sv

```
function void unpack(ref bit [7:0] q_inp[$]);
  {<< 8 {this.payload,this.crc,this.len,this.da,this.sa}} = q_inp;
endfunction
```

Step 4: Implement print method in class packet.

1) Define print method in class packet.
2) Print the content of all class members including payload.

Example: Add this code in packet.sv

```
function void print();
$write("[Packet Print] Sa=%0d Da=%0d Len=%0d Crc=%0d",sa,da,len,crc);
$write(" Payload:");
foreach(payload[k])
 $write(" %0h",payload[k]);

$display("\n");
endfunction
```

Step 5: Implement method generate_stimulus() in testbench.sv.

1) Define method generate_stimulus with class packet as argument.

Add the below code in section 5.1 of testbnch.sv

```
//Section 5.1 : Define generate_stimulus() method

function automatic void generate_stimulus(ref packet
gen_pkt , input int pkt_id);
gen_pkt.sa=$urandom_range(1,8);
gen_pkt.da=$urandom_range(1,8);
gen_pkt.payload=new[$urandom_range(10,1900)];
```

```
foreach(gen_pkt.payload[i])
  gen_pkt.payload[i]=$urandom;

gen_pkt.len=gen_pkt.payload.size() + 1+1+4+4;
gen_pkt.crc=gen_pkt.payload.sum();

$display("[Packet Generate] Packet %0d (size=%0d)
Generated at time=%0t",pkt_id,gen_pkt.len,$time);
endfunction
```

Step 6: Implement drive method in testbench.sv.

1) Define drive method with class packet as argument.
2) Add this code in Section 5.2 in testbench.sv

```
//Section 5.2 : Define drive() method

task automatic drive(ref packet pkt , input int ptk_id);
wait(vif.cb.busy==0);
@(vif.cb);
$display("[TB Drive] Driving of packet %0d (size=%0d) started
at time=%0t",pkt_id,pkt.len,$time);
vif.cb.inp_valid<=1;
foreach(pkt.inp_stream[i]) begin
vif.cb.dut_inp <= pkt.inp_stream[i];
@(vif.cb);
end
$display("[TB Drive] Driving of packet %0d (size=%0d) ended at
time=%0t \n",pkt_id,pkt.len,$time);
@(vif.cb);
vif.cb.inp_valid<=0;
vif.cb.dut_inp<='z;
repeat(5) @(vif.cb);
endtask
```

Step 7: Implement compare method in testbench.sv.

1) Implement compare method with class types as argument
2) Add this code in Section 5.3 in testbench.sv

```
//Section 5.3 : Define compare method()

function bit compare(input packet ref_pkt,input packet dut_pkt);
bit status;
status =1;
foreach(ref_pkt.inp_stream[i]) begin
   status = status && (ref_pkt.inp_stream[i] ==
dut_pkt.outp_stream[i]);
end
return status;
endfunction
```

Step 8: Capture output from dut.

1) Collect complete packet from dut and store it for self-checking purpose
2) Add this code in section 8 in testbench.sv

```
//Section 8: Collecting DUT output
initial begin
   bit [15:0] cnt;
 forever begin
   @(posedge vif.cb.outp_valid);
   while(1) begin
   //Section 8.1 : Capture complete packet from DUT
outp_stream.push_back(vif.cb.dut_outp);

//Section 8.2 : Collect untill outp_valid becomes 0.
        if(vif.cb.outp_valid==0) begin

//Section 8.3 : Increment the cnt to track how many packets
collected
         cnt++;
```

```
  //Section 8.4 : Construct dut_pkt object
       dut_pkt=new;

  //Section 8.5 : Unpack collected outp_stream into dut_pkt fields
       dut_pkt.unpack(outp_stream);

  //Section 8.6 : Copy local outp_stream to outp_stream in dut_pkt
       dut_pkt.outp_stream=outp_stream;

//Section 8.7 : Store the actual packet from DUT for sel-checking
       q_outp.push_back(dut_pkt);
       //dut_pkt.print();

   $display("[TB Output Monitor] Packet %0d collected size=%0d
time=%0t",cnt,outp_stream.size(),$time);

 //Section 8.8 : Delete local outp_stream queue
       outp_stream.delete();

 //Section 8.9 : Break out of while loop as collection of packet
completed.
       break;
  end
  //Section 8.10 : Wait for posedge of clk to collect all the dut output
       @(vif.cb);

   end//end_of_while
  end//end_of_forever
end//end_of_initial
```

Step 9: Start the verification flow.

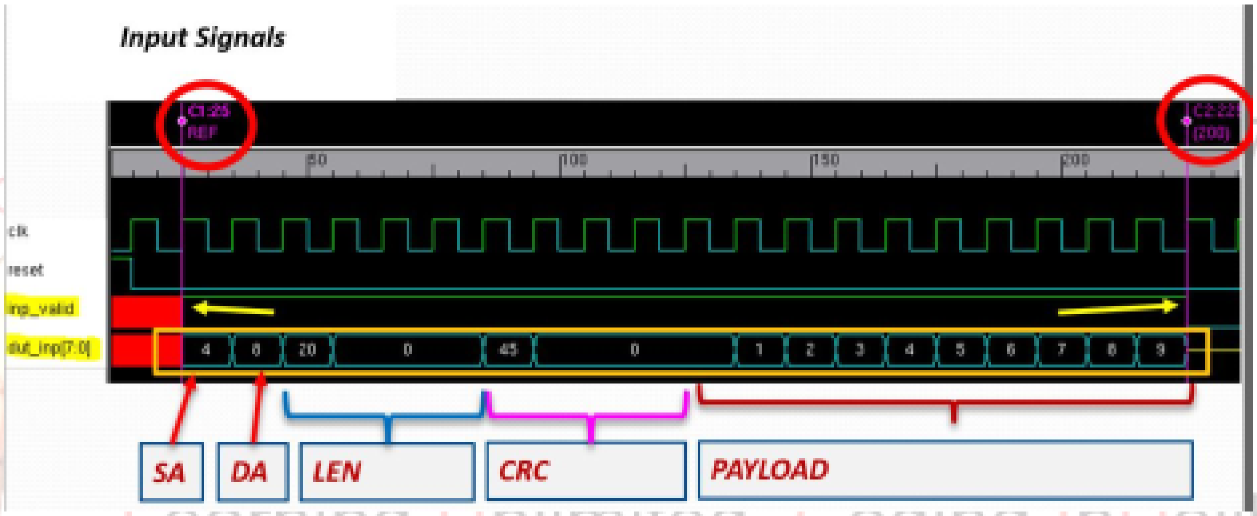
1) Start the flow by calling all the required methods.
2) Ad this code in section 6 in testbench.sv

```
//Section 6: Verification Flow
initial begin
//Section 6.1 : How many number of packets to generate
pkt_count=10;

//Section 6.2 : Call apply_reset() method.
apply_reset();

repeat(pkt_count) begin
wait(vif.cb.busy==0);
pkt_id++;
//Section 6.3 : Construct stimulus packet Object
stimulus_pkt=new;

//Section 6.4 : Call generate_stimulus() method.
generate_stimulus(stimulus_pkt,pkt_id);

//Section 6.5 : Call pack() method.
stimulus_pkt.pack(stimulus_pkt.inp_stream);

//Section 6.6 : Store Reference/Golden packet into q_inp.
q_inp.push_back(stimulus_pkt);

//Section 6.7 : Call drive() method.
drive(stimulus_pkt,pkt_id);
end
//Wait for dut to process the packet and to drive on output
wait(vif.cb.busy==0);//drain time
repeat(10) @(vif.cb);//drain time
result();
$finish;
end
```

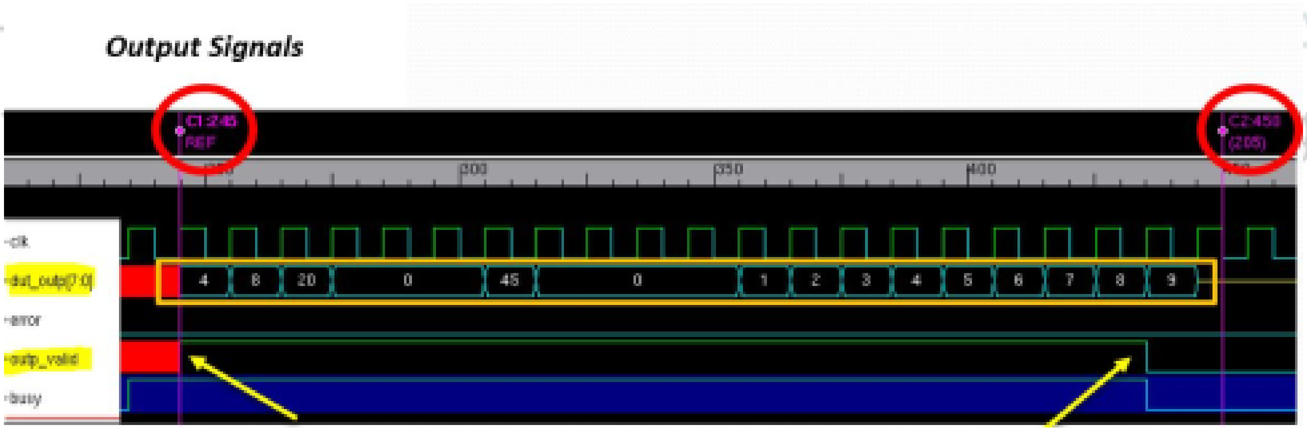
Step 10: Run the simulation and validate the output of DUT with the results printed by self-checking mechanism.

1. *run the simulation.*
2. *Check the test Passed or Failed and debug the if there are any failures.*

Reference input waveform:



Reference output waveform:


Output Signals
C1:245
REF
C2:450
(205)
300
350
400
clk
dut_outp[7:0]
error
outp_valid
busy
4
8
20
0
45
0
1
2
3
4
5
6
7
8
9