Image Processing Accelerator

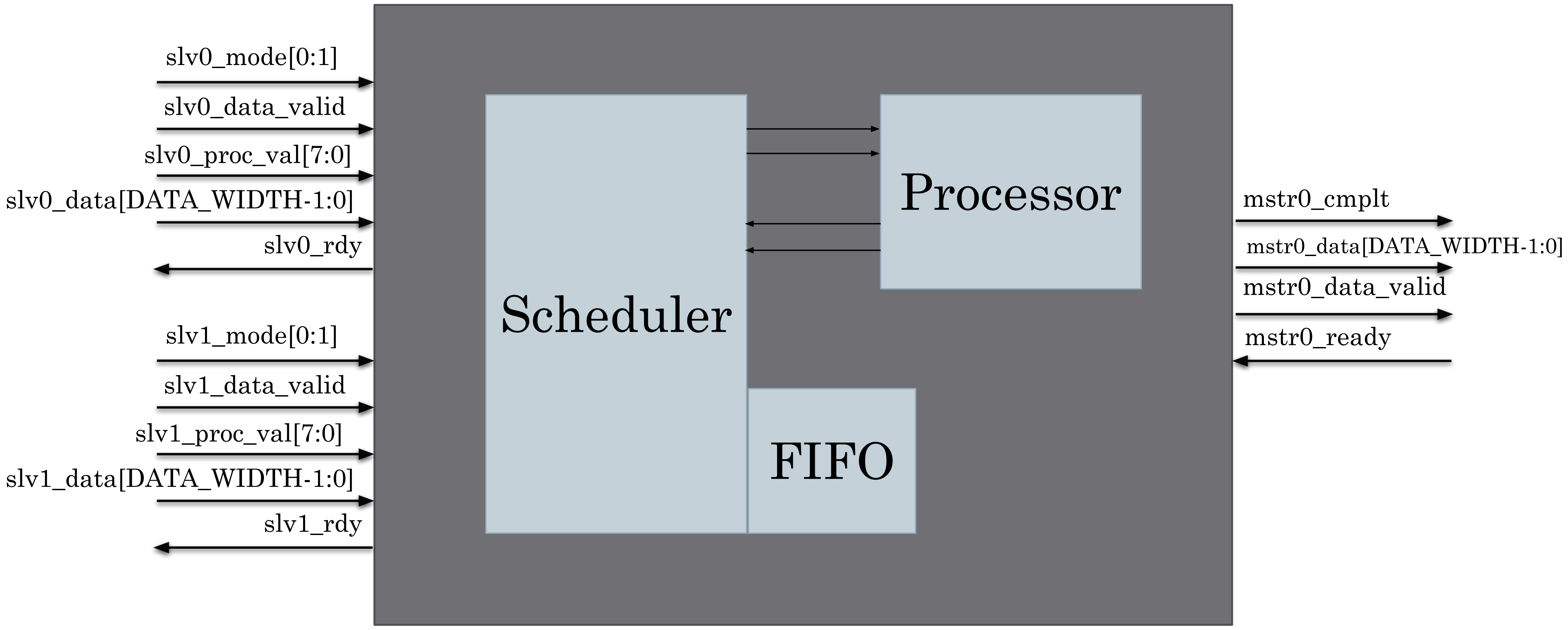
ITC – Design Verification Final Project

Itai Yifrach & Alon Volkind

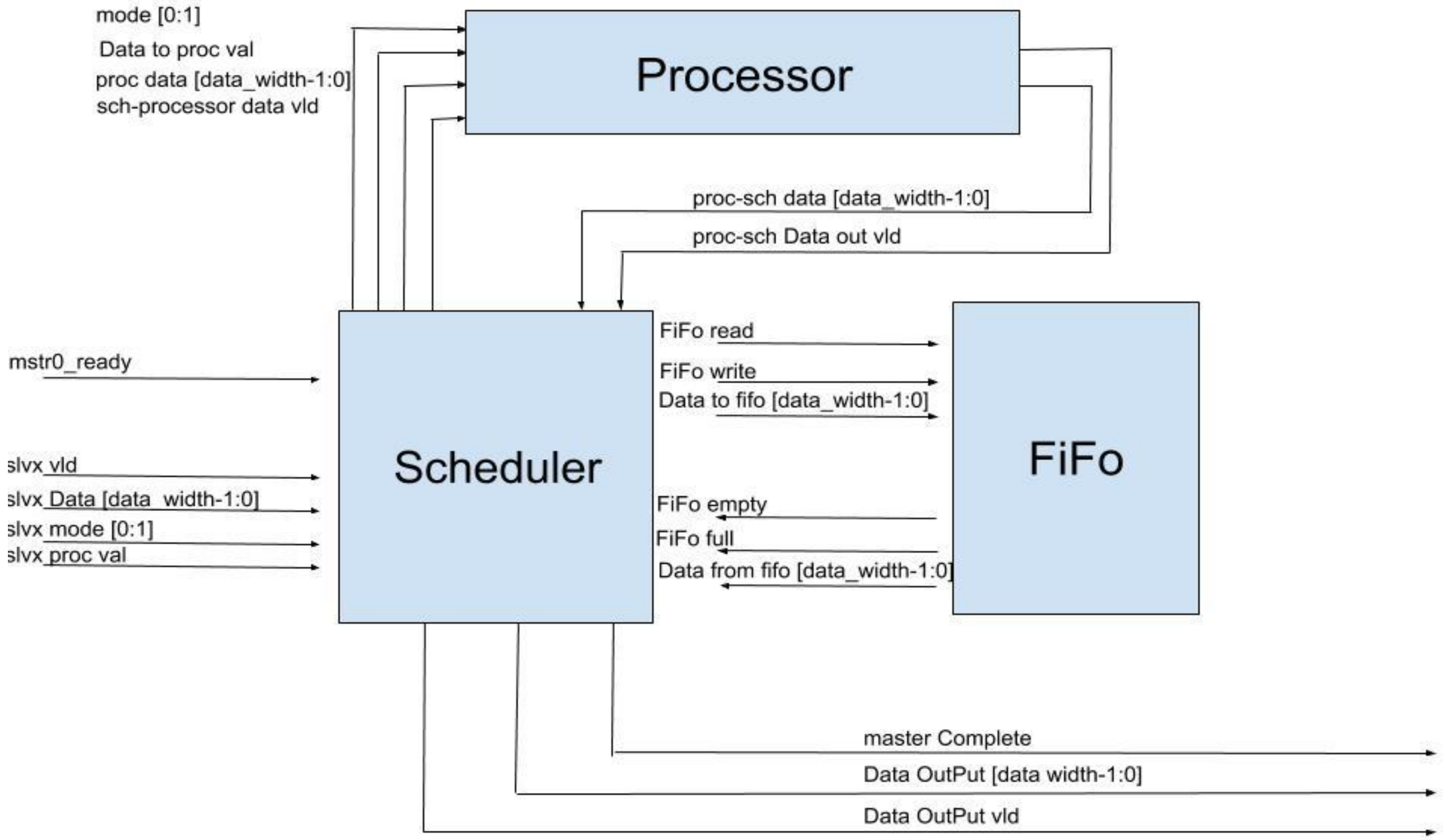
Content

1. Design Overview
2. Verification Working flow
3. Verification Environment
4. Coverage & Assertions
5. Bugs Overview
6. Results:
   a. Coverage
   b. Images Examples
7. Challenges / Problems
8. Further Work

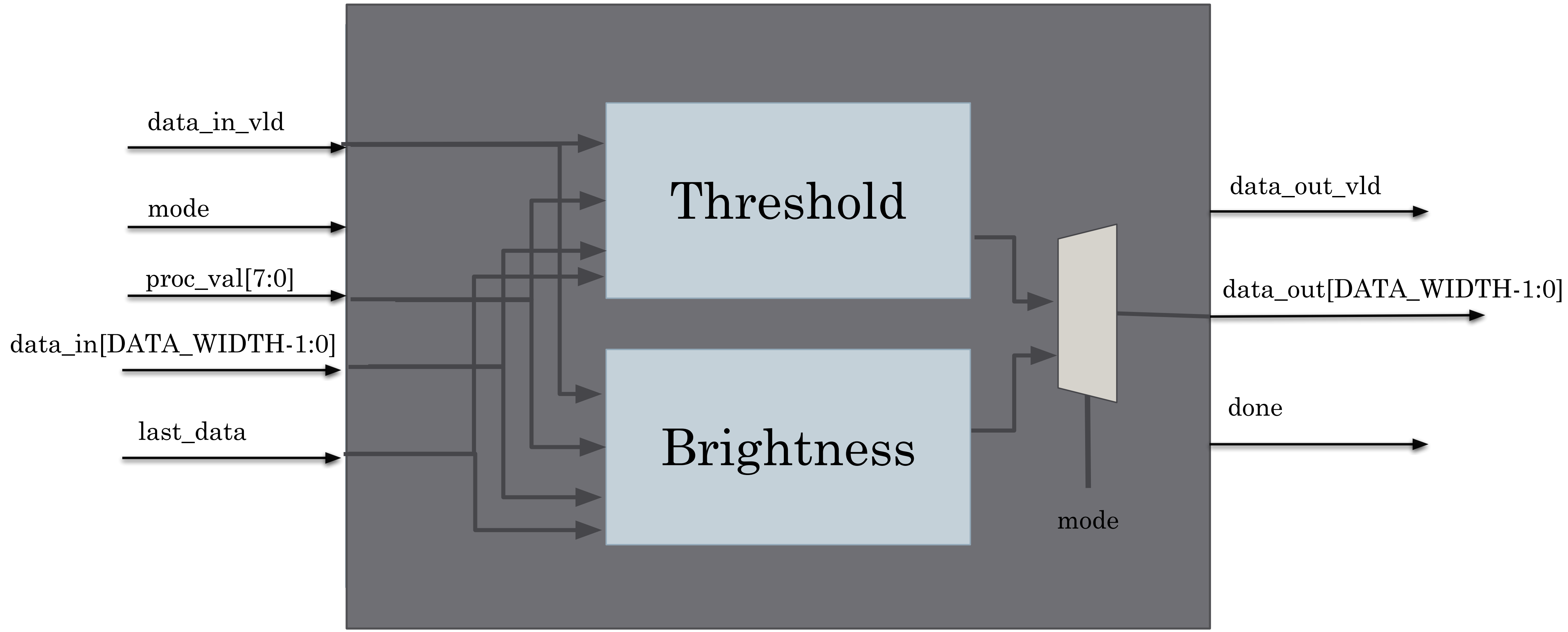
Design


slv0_mode[0:1]
slv0_data_valid
slv0_proc_val[7:0]
slv0_data[DATA_WIDTH-1:0]
slv0_rdy
slv1_mode[0:1]
slv1_data_valid
slv1_proc_val[7:0]
slv1_data[DATA_WIDTH-1:0]
slv1_rdy
Scheduler
Processor
FIFO
mstr0_cmplt
mstr0_data[DATA_WIDTH-1:0]
mstr0_data_valid
mstr0_ready

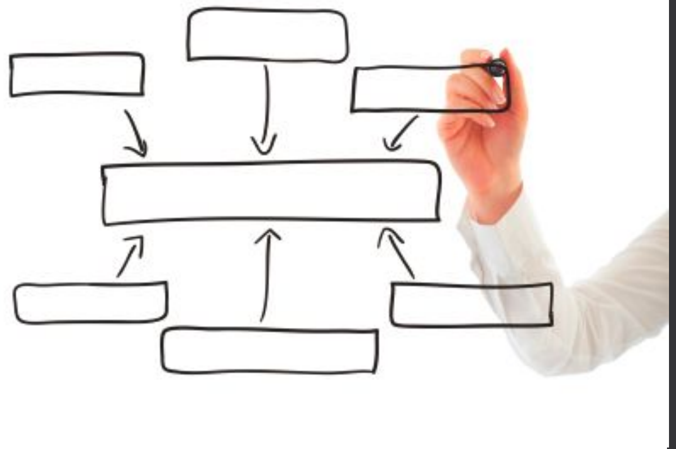
The DUT


mode [0:1]
Data to proc val
proc data [data_width-1:0]
sch-processor data vld
Processor
proc-sch data [data_width-1:0]
proc-sch Data out vld
mstr0_ready
slvx_vld
slvx_Data [data_width-1:0]
slvx_mode [0:1]
slvx_proc val
Scheduler
FiFo read
FiFo write
Data to fifo [data_width-1:0]
FiFo empty
FiFo full
Data from fifo [data_width-1:0]
FiFo
master Complete
Data OutPut [data width-1:0]
Data OutPut vld

Processor


data_in_vld
mode
proc_val[7:0]
data_in[DATA_WIDTH-1:0]
last_data
Threshold
Brightness
mode
data_out_vld
data_out[DATA_WIDTH-1:0]
done

Verification Working Flow

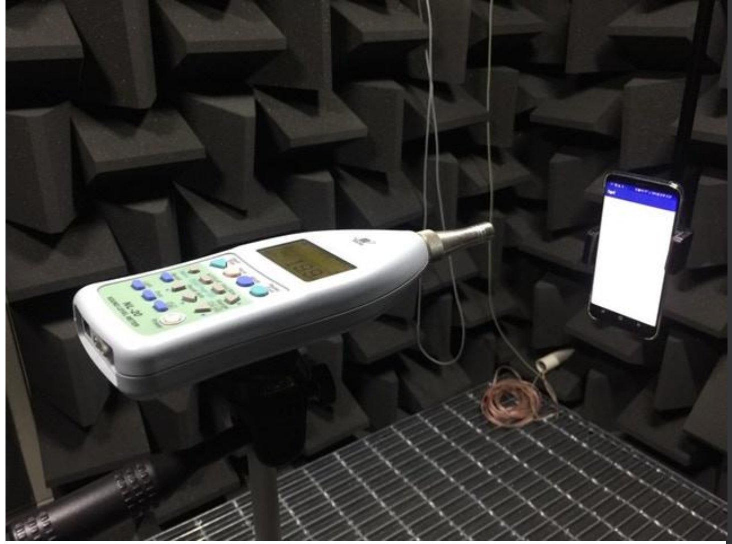


- Gather all information sources
- Define main verification goals
- Design verification plan - Verification environment, Stimulus Generation, Checkers Coverage
- Integrating all components
- *Focus on the whole DUT model rather than it's sub-models*
- Repeat till coverage goals been reached - Random Testing -> Coverage analyze -> Constrained random testing -> Coverage analyze

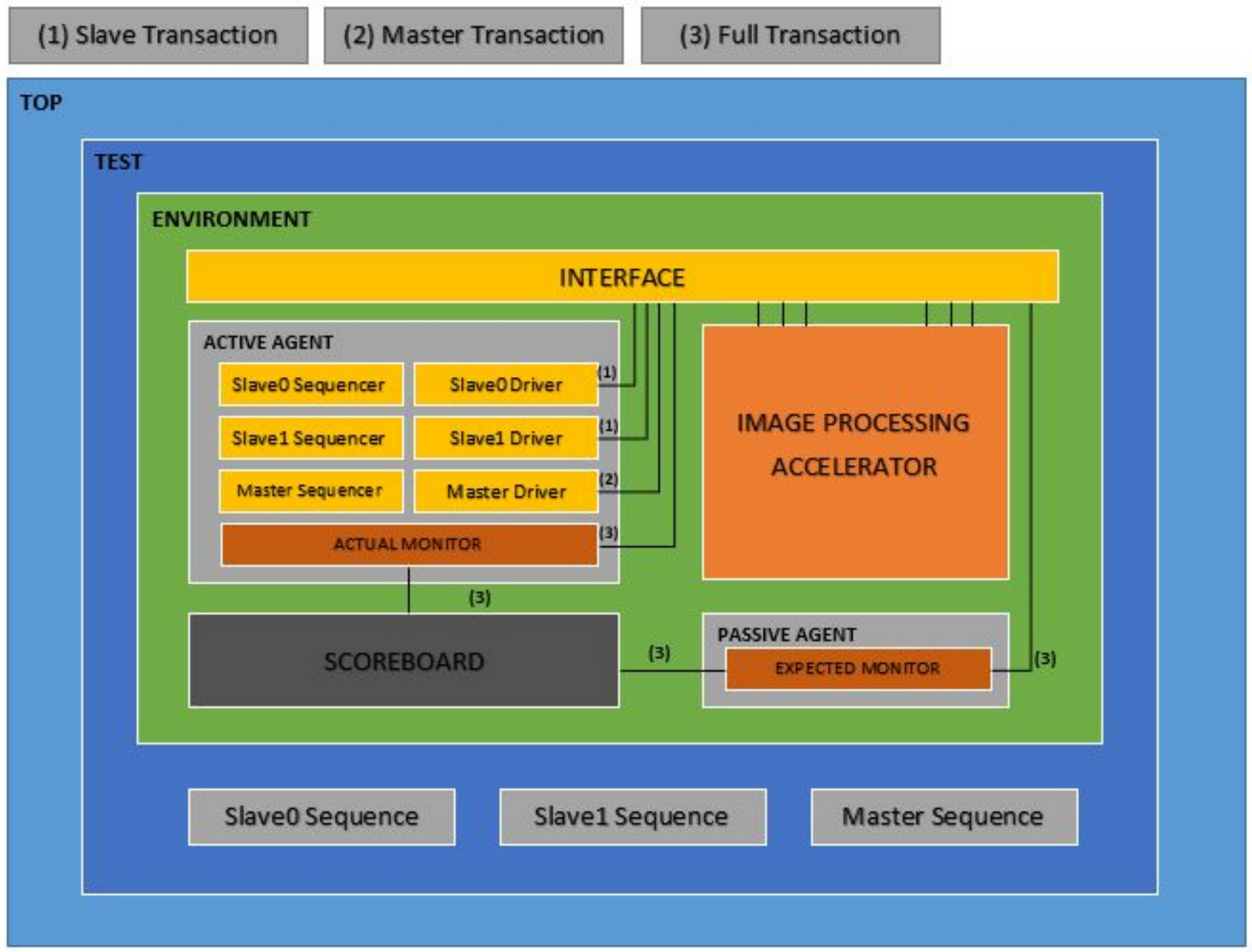
Main Verification Goals

- Data reliability
- Data Validity
- Correct flow of Continues file transfer - with alt file size
- Granting priority
- Correct behavior at slaves and master pausing
- Timing – Continues (unless paused) output to the master
- All parts of design are reachable

Verification Environment



- Top
- Test
- Environment
- Interface
- Active Agent - Includes A monitor and a sequencer & driver for each slave and master
- Passive Agent - Includes monitor only
- Scoreboard
- Slave0 Sequence, Slave1 Sequence, Master Sequence
- Slave Transaction, Master Transaction, Full Transaction
- Image - A class generator for random images.


(1) Slave Transaction
(2) Master Transaction
(3) Full Transaction
TOP
TEST
ENVIRONMENT
INTERFACE
ACTIVE AGENT
Slave0 Sequencer
Slave0 Driver
Slave1 Sequencer
Slave1 Driver
Master Sequencer
Master Driver
ACTUAL MONITOR
(1)
(1)
(2)
(3)
IMAGE PROCESSING ACCELERATOR
(3)
SCOREBOARD
(3)
PASSIVE AGENT
EXPECTED MONITOR
(3)
Slave0 Sequence
Slave1 Sequence
Master Sequence

Slave/Master Transaction

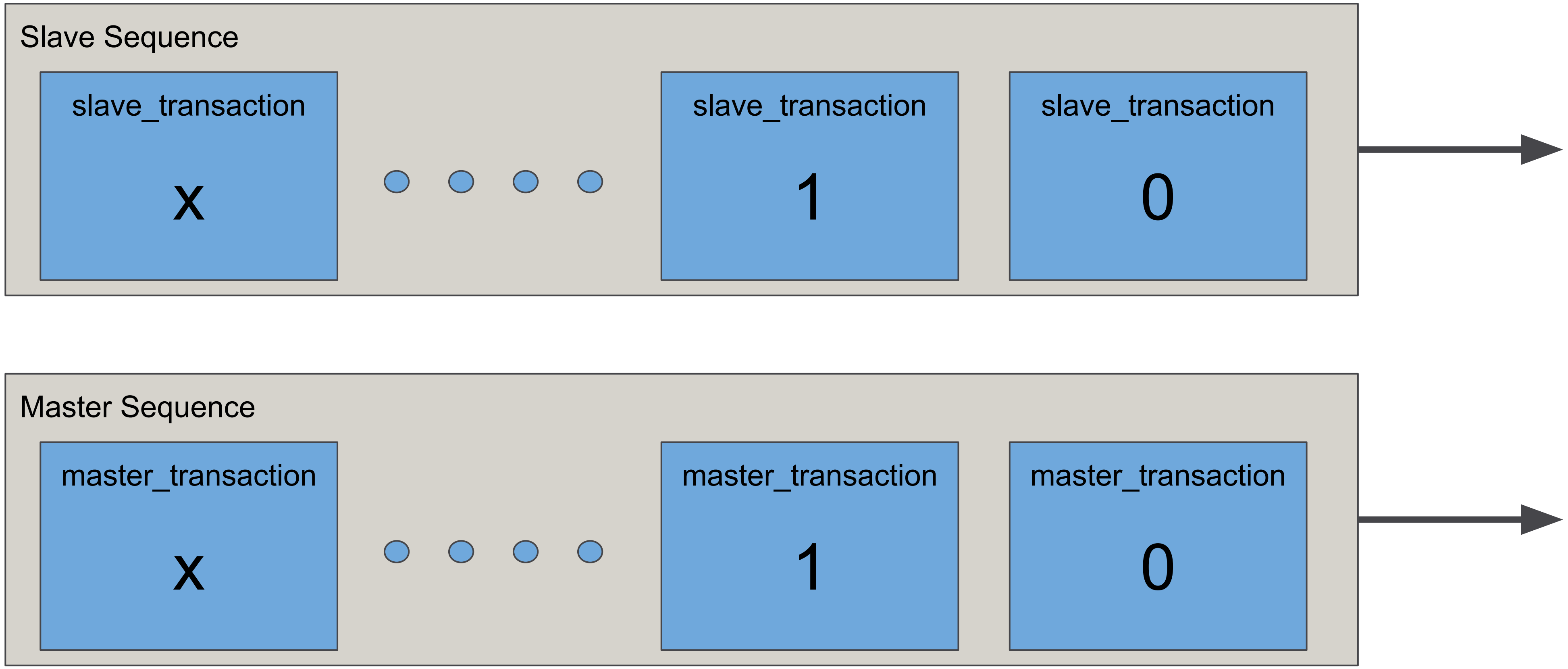
Slave Transaction:

- mode
- proc_val
- data_valid_stop
- data_valid_stop_for
- data_valid_stop_at
- Image

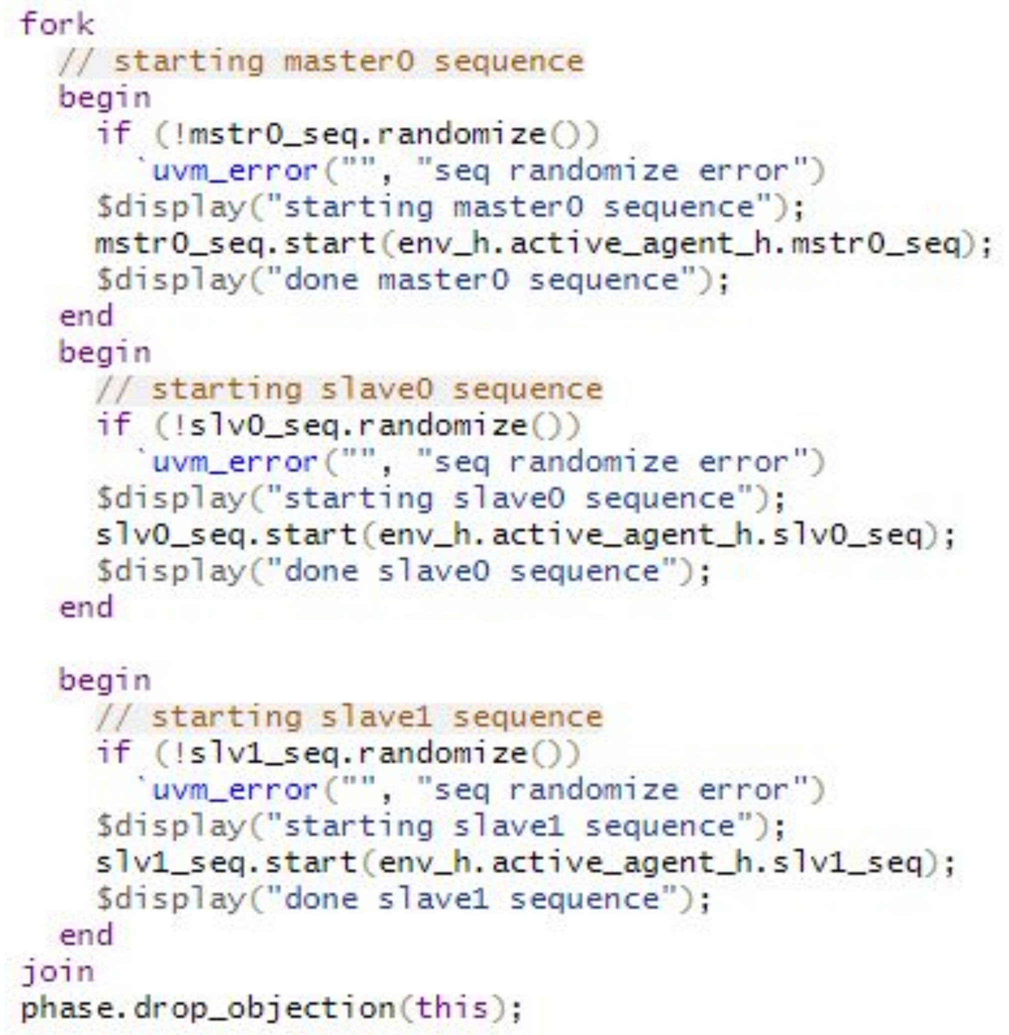
Master Transaction:

- ready

Slave/Master Sequence

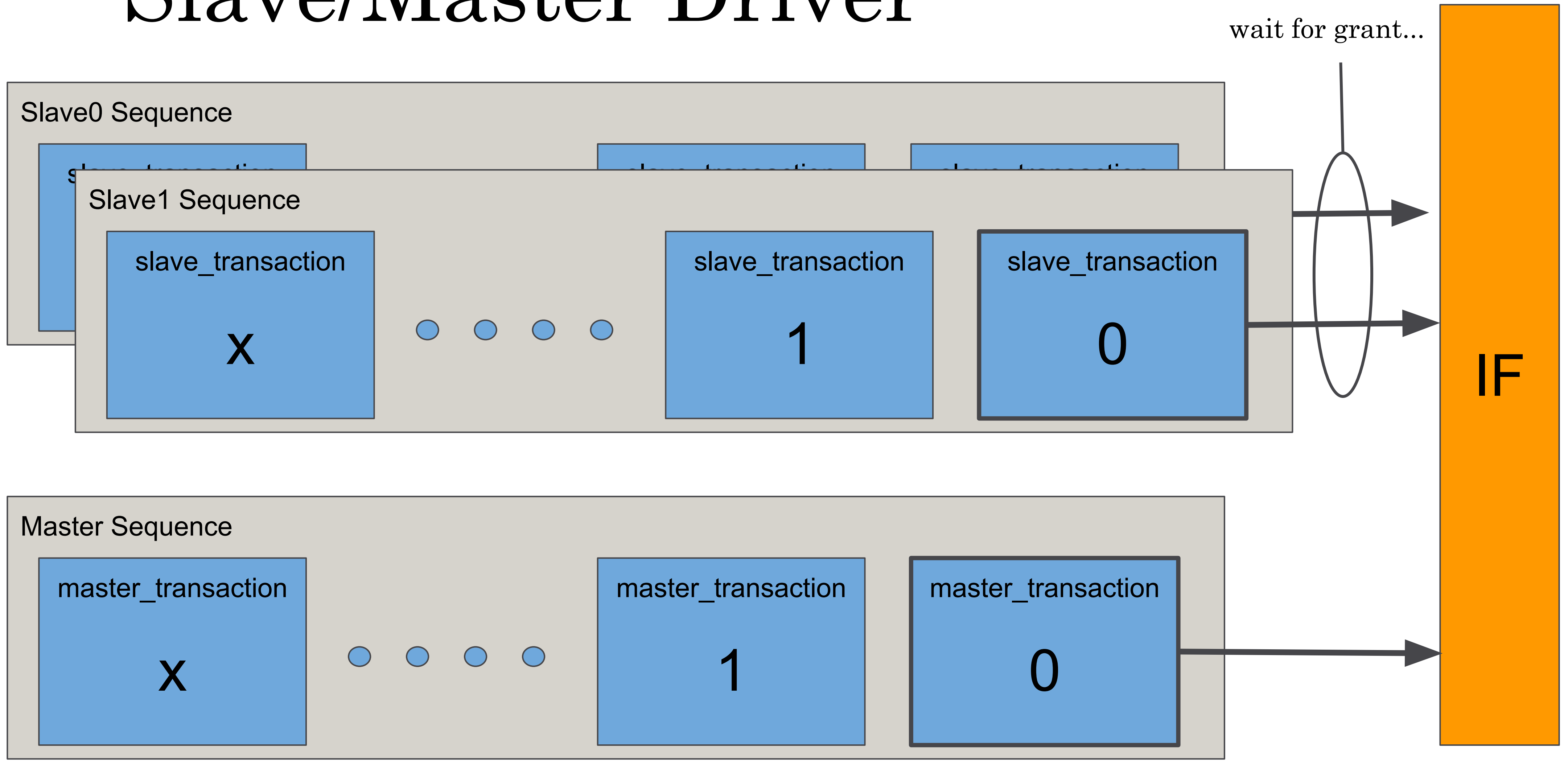


Slave/Master Sequence



```
fork
  // starting master0 sequence
  begin
    if (!mstr0_seq.randomize())
      `uvm_error("", "seq randomize error")
    $display("starting master0 sequence");
    mstr0_seq.start(env_h.active_agent_h.mstr0_seq);
    $display("done master0 sequence");
  end
  begin
    // starting slave0 sequence
    if (!slv0_seq.randomize())
      `uvm_error("", "seq randomize error")
    $display("starting slave0 sequence");
    slv0_seq.start(env_h.active_agent_h.slv0_seq);
    $display("done slave0 sequence");
  end

  begin
    // starting slave1 sequence
    if (!slv1_seq.randomize())
      `uvm_error("", "seq randomize error")
    $display("starting slave1 sequence");
    slv1_seq.start(env_h.active_agent_h.slv1_seq);
    $display("done slave1 sequence");
  end
join
phase.drop_objection(this);
```

Slave/Master Driver



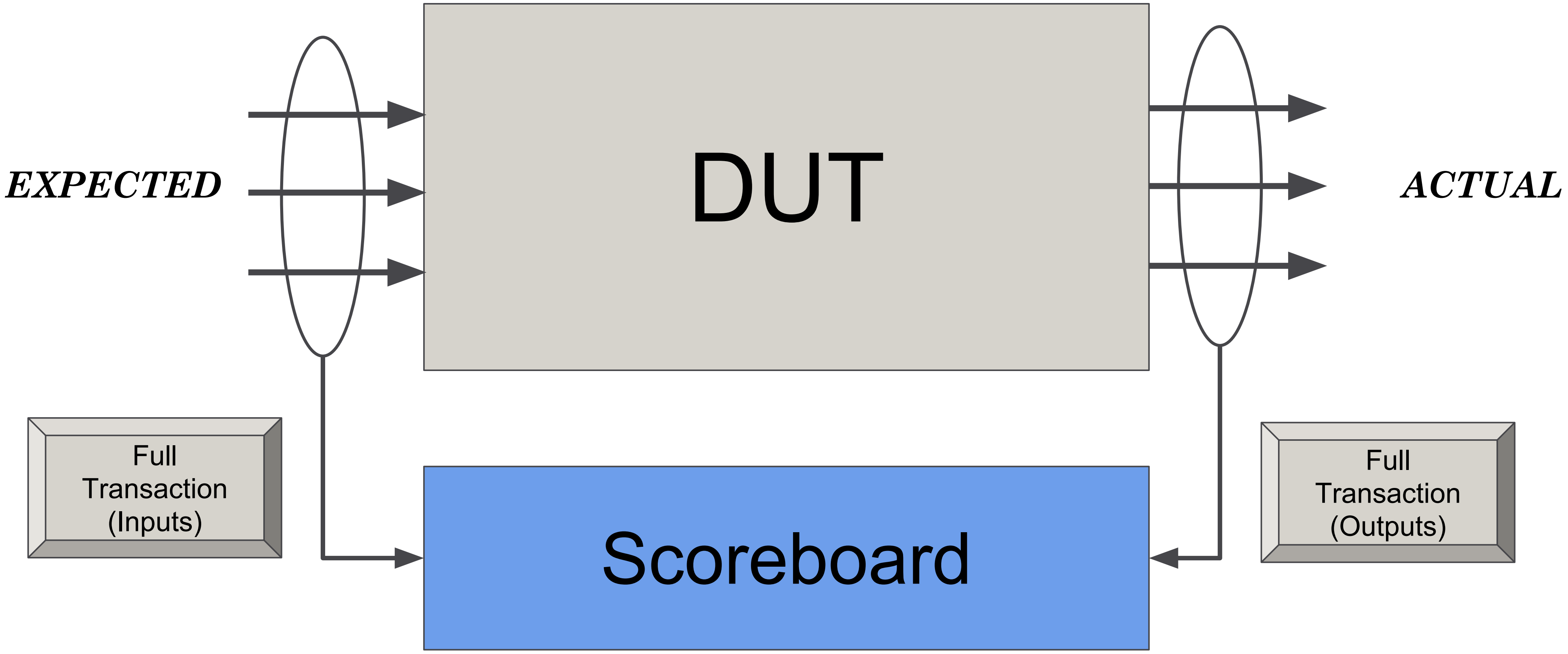
Full Transaction

- slv0_mode
- slv0_data_valid
- slv0_proc_val
- slv0_data
- slv0_rdy

- mstr0_cmplt
- mstr0_ready
- mstr0_data
- mstr0_data_valid

- slv1_mode
- slv1_data_valid
- slv1_proc_val
- slv1_data
- slv1_rdy

Actual and Expected Monitor



Scoreboard

Receiving actual and expected full transactions. Three major checkers:

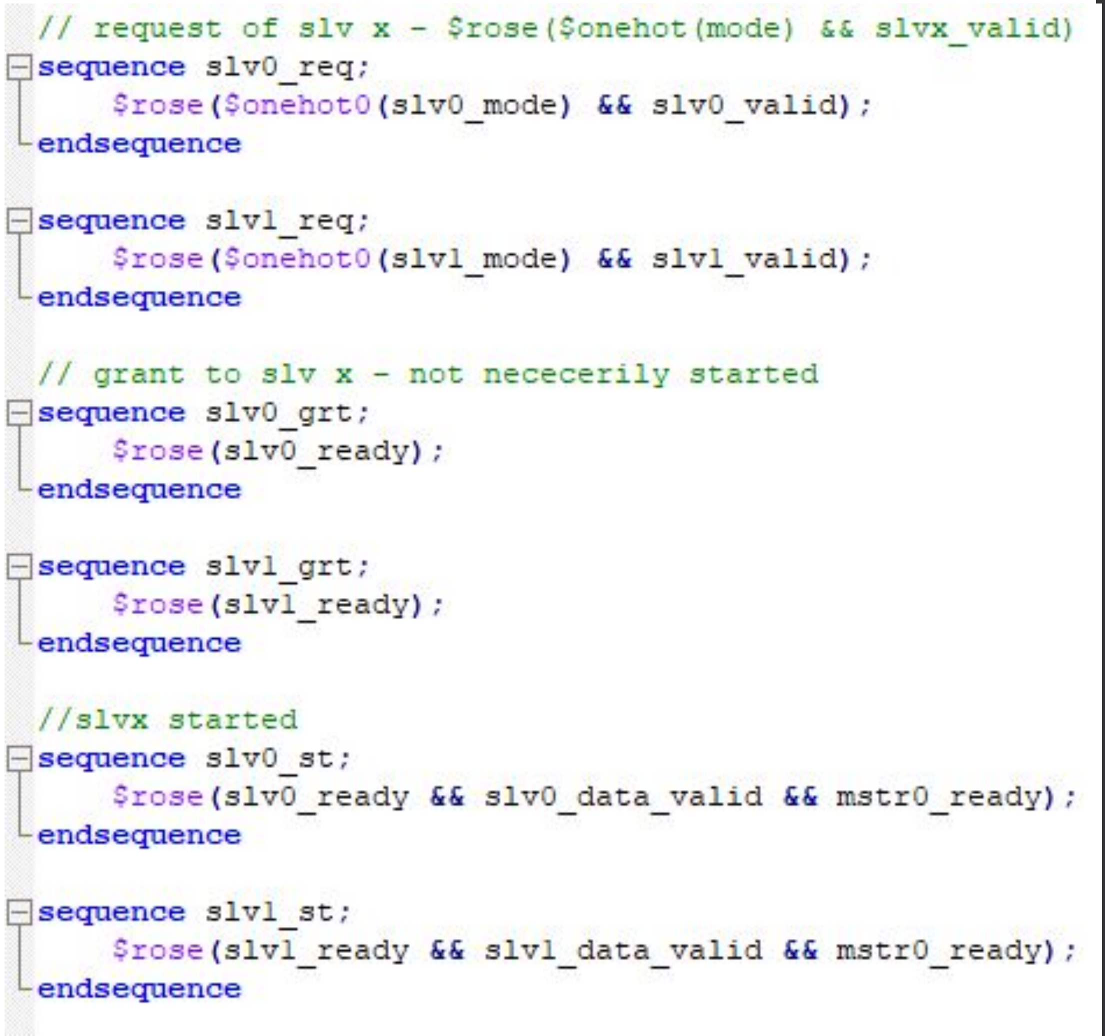
1. Arbiter functionality:
   a. Is the chosen slave the right one?
2. Processor works as expected according to the requested mode:
   a. header == header
   b. and f_mode(pixel_in) = pixel_out

Scoreboard

3. Delays:
   a. Threshold mode delays the initial output for 4 clocks
   b. No outputs if no data valid from the slave
   c. Master complete in time
   d. If master ready is down, slave ready is down also

Events To Cover

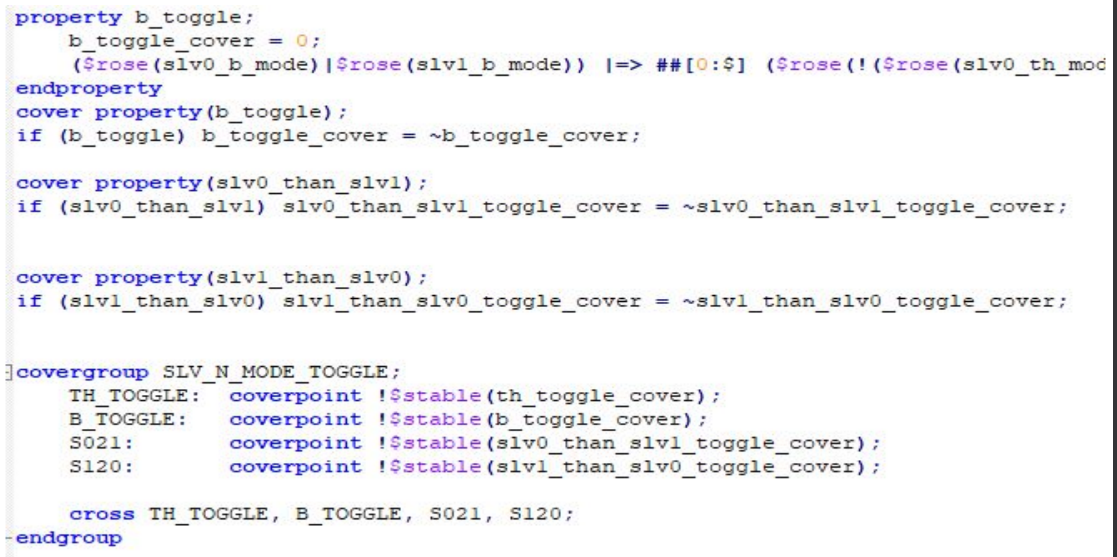
- Slaves Grants
- Slaves Requests
- Slaves start delivering
- start master outputting event
- finish master outputting event
- start outputting to the processor
- finish outputting to the processor
- slvx started in TH mode
- slvx granted in BR mode
- slv drop its vld before completion (pausing)



```
// request of slv x - $rose($onehot(mode) && slvx_valid)
sequence slv0_req;
    $rose($onehot0(slv0_mode) && slv0_valid);
endsequence

sequence slv1_req;
    $rose($onehot0(slv1_mode) && slv1_valid);
endsequence

// grant to slv x - not nececerily started
sequence slv0_grt;
    $rose(slv0_ready);
endsequence

sequence slv1_grt;
    $rose(slv1_ready);
endsequence

//slvx started
sequence slv0_st;
    $rose(slv0_ready && slv0_data_valid && mstr0_ready);
endsequence

sequence slv1_st;
    $rose(slv1_ready && slv1_data_valid && mstr0_ready);
endsequence
```

More interesting events to cover

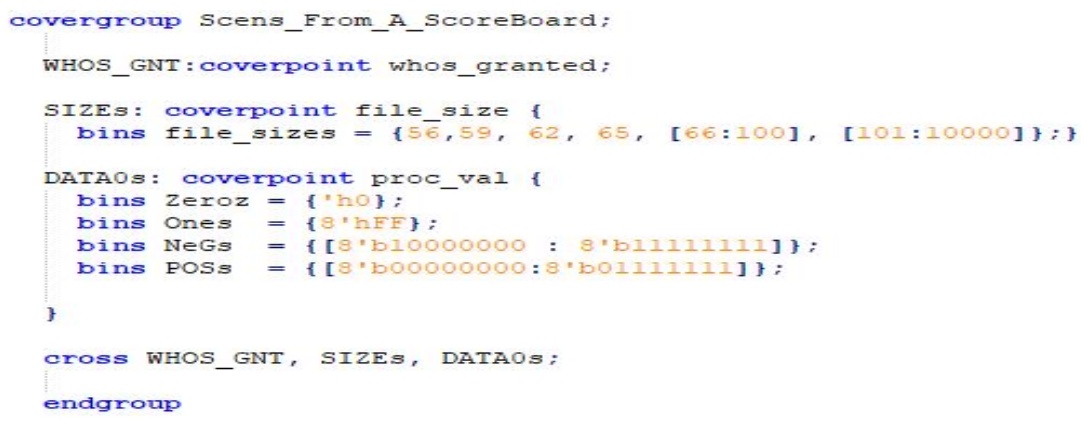
- processing TH multiple files continuously from slvx
- processing b multiple files continuously from slvx
- serving slv 0, than slv 1 with alternating modes (TH, than Brightness and In the opposite way)
- serving slv 1, than slv 0 (TH, than Brightness and In the opposite way)
- serving slv 0 with TH, than slv 1 with TH
- serving slv 1 with TH, than slv 0 with TH



```
property b_toggle;
    b_toggle_cover = 0;
    ($rose(slv0_b_mode)|$rose(slv1_b_mode)) |=> ##[0:$] ($rose(!($rose(slv0_th_mod
endproperty
cover property(b_toggle);
if (b_toggle) b_toggle_cover = ~b_toggle_cover;

cover property(slv0_than_slv1);
if (slv0_than_slv1) slv0_than_slv1_toggle_cover = ~slv0_than_slv1_toggle_cover;


cover property(slv1_than_slv0);
if (slv1_than_slv0) slv1_than_slv0_toggle_cover = ~slv1_than_slv0_toggle_cover;


covergroup SLV_N_MODE_TOGGLE;
    TH_TOGGLE:  coverpoint !$stable(th_toggle_cover);
    B_TOGGLE:   coverpoint !$stable(b_toggle_cover);
    S021:       coverpoint !$stable(slv0_than_slv1_toggle_cover);
    S120:       coverpoint !$stable(slv1_than_slv0_toggle_cover);

    cross TH_TOGGLE, B_TOGGLE, S021, S120;
endgroup
```

Corner Cases to test and cover

- Sizes – Files with only header, 1Pix, 2Pix, 3Pix, VLSI (Very Large Sized Images).
- Brightness & Threshold Values – All Zeros, All Ones, Neg, Pos
- Pausing – At beginning/during/ending of Scenes To Cover
- Pausing duration – more than a cycle
- Resetting – At beginning/during/ending of Scenes To Cover
- master ready falling after file input finished entering but still outputting to the master (Due to Dead Time)
- Different pixel indentation on bus



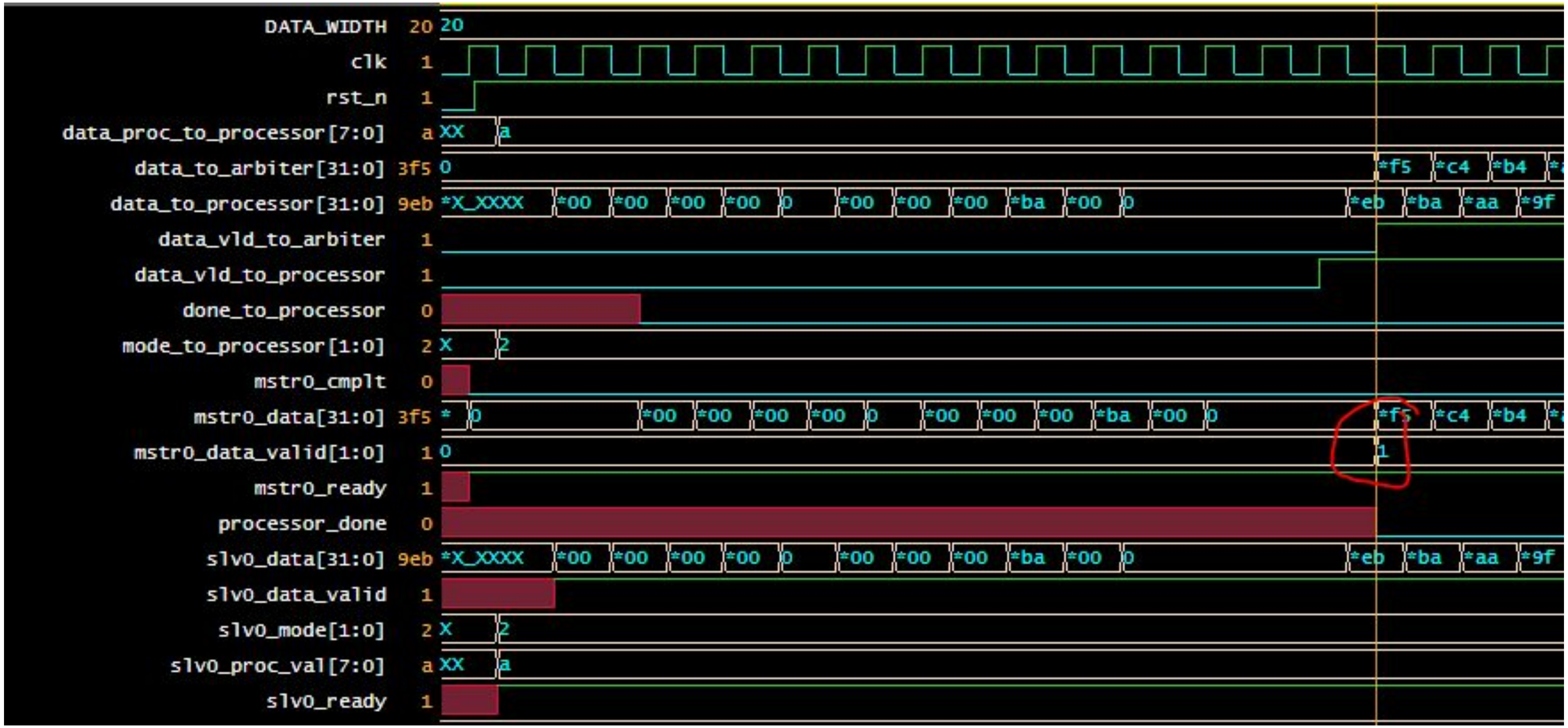
```
covergroup Scens_From_A_ScoreBoard;

  WHOS_GNT:coverpoint whos_granted;

  SIZEs: coverpoint file_size {
    bins file_sizes = {56,59, 62, 65, [66:100], [101:10000]};}

  DATA0s: coverpoint proc_val {
    bins Zeroz = {'h0};
    bins Ones  = {8'hFF};
    bins NeGs  = {[8'b10000000 : 8'b11111111]};
    bins POSs  = {[8'b00000000:8'b01111111]};

  }

  cross WHOS_GNT, SIZEs, DATA0s;

  endgroup
```

Assertions

- If any of the slv valid is on than its mode should be 01 or 10
- if (slv0_req && slv0_data_vld), than it will get grant some time in the future but before slave 1 will get new one
- Not fairness - if slv 0 granted, and its valid high than next grant should be gnt[0]
- if slvx granted in TH mode, master out == slv_data (from 3rd clk till the end of header)
- if slvx granted in B mode, master out == slv_data (from 1st clk till the end of header)
- sticky mode from grant till the end of the file transfer
- In scheduler, processor and master – If Valid |→ data != x or z
- master ready flag at the end of outputting

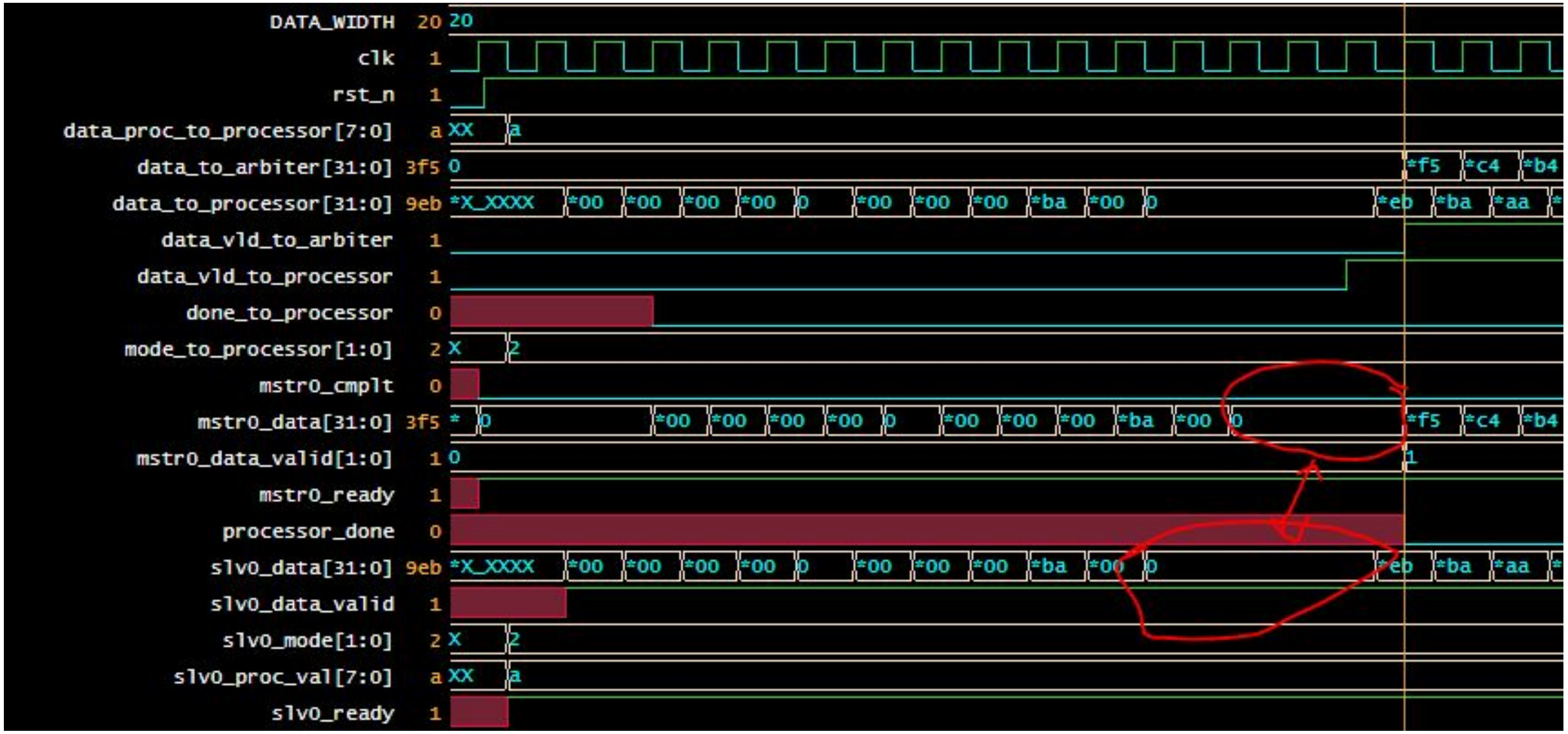
DUT Bugs...

No master valid for the header

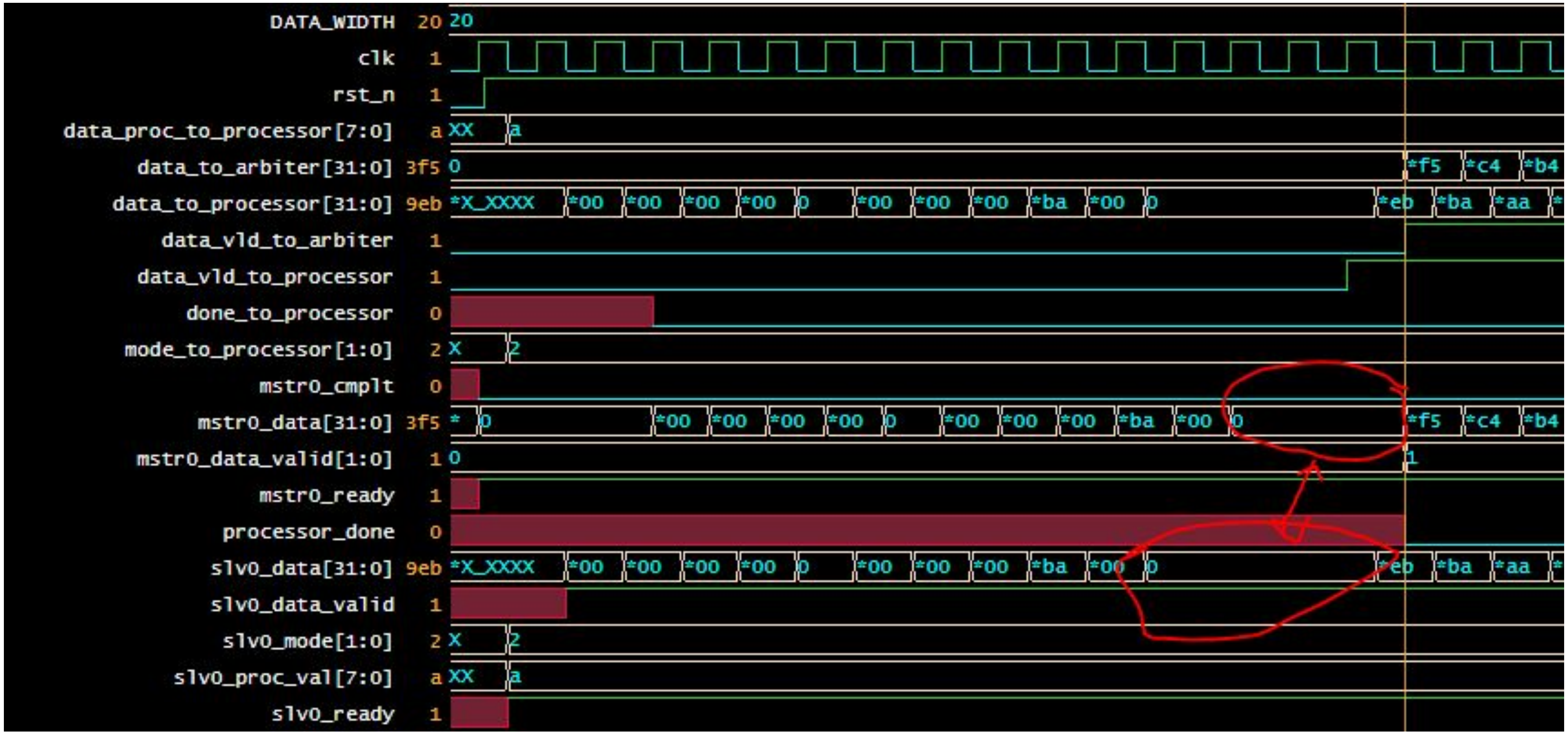


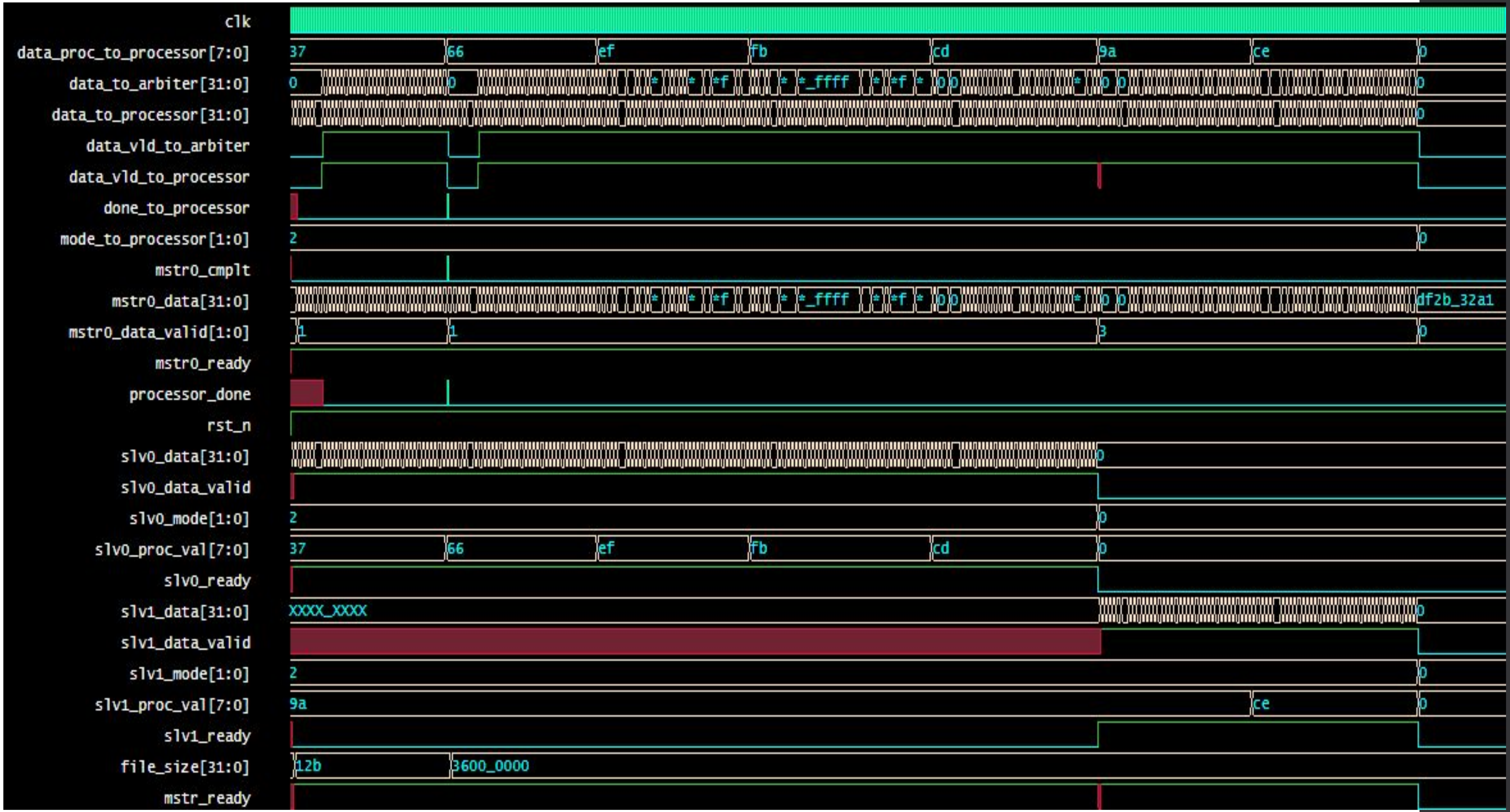
```
DATA_WIDTH                  20
clk                          1
rst_n                        1
data_proc_to_processor[7:0]  a
data_to_arbiter[31:0]      3f5
data_to_processor[31:0]    9eb
data_vld_to_arbiter          1
data_vld_to_processor        1
done_to_processor            0
mode_to_processor[1:0]       2
mstr0_cmplt                  0
mstr0_data[31:0]           3f5
mstr0_data_valid[1:0]        1
mstr0_ready                  1
processor_done               0
slv0_data[31:0]            9eb
slv0_data_valid              1
slv0_mode[1:0]               2
slv0_proc_val[7:0]           a
slv0_ready                   1
```

Two data's of the header are tossed



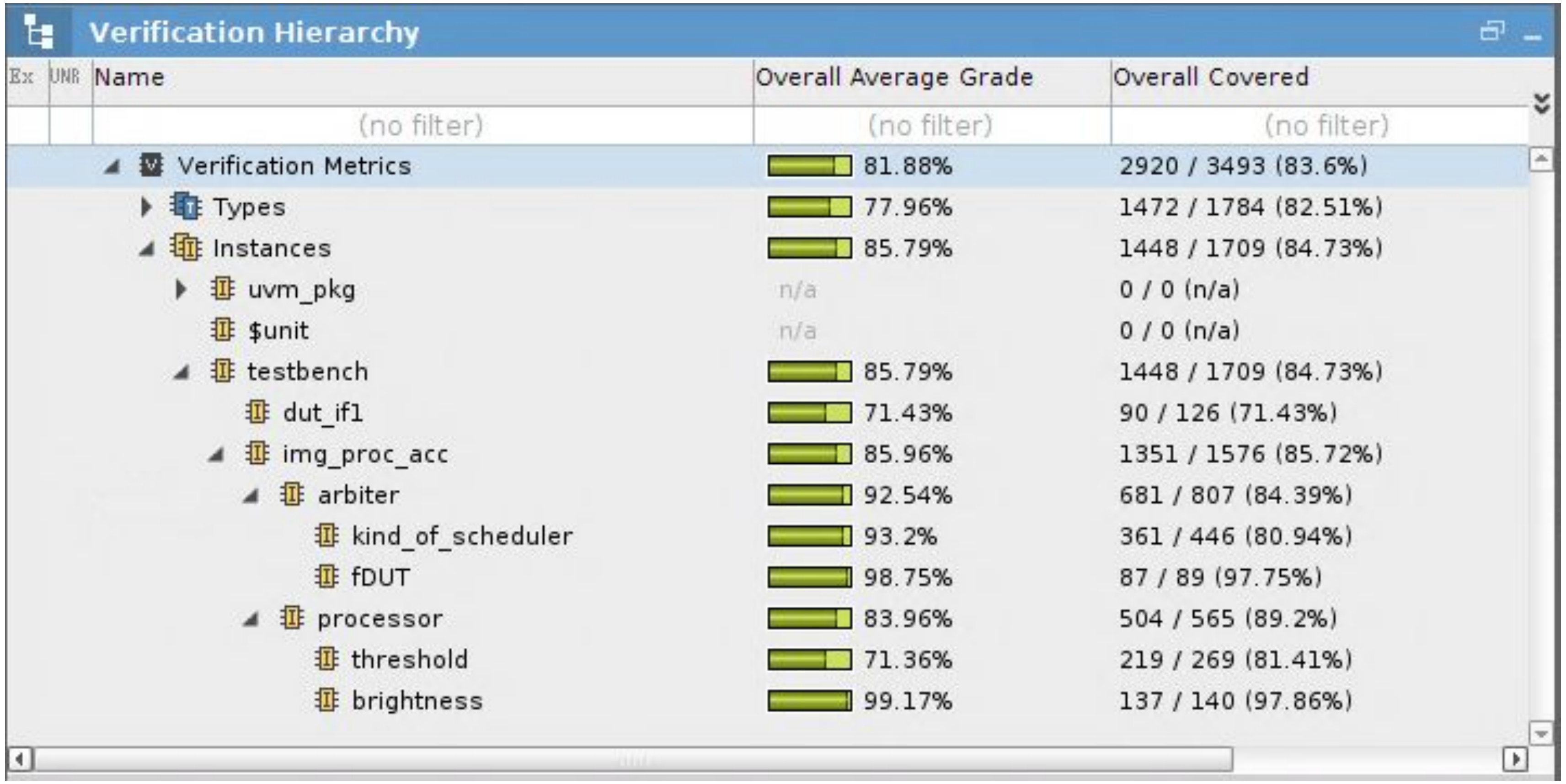
Two data's of the header are tossed




clk
data_proc_to_processor[7:0]
37
66
ef
fb
cd
9a
ce
0
data_to_arbiter[31:0]
*_ffff
data_to_processor[31:0]
data_vld_to_arbiter
data_vld_to_processor
done_to_processor
mode_to_processor[1:0]
2
mstr0_cmplt
mstr0_data[31:0]
df2b_32a1
mstr0_data_valid[1:0]
1
3
mstr0_ready
processor_done
rst_n
slv0_data[31:0]
slv0_data_valid
slv0_mode[1:0]
slv0_proc_val[7:0]
slv0_ready
slv1_data[31:0]
XXXX_XXXX
slv1_data_valid
slv1_mode[1:0]
slv1_proc_val[7:0]
slv1_ready
file_size[31:0]
12b
3600_0000
mstr_ready

Results

We ran 200 images, 100 per slave:

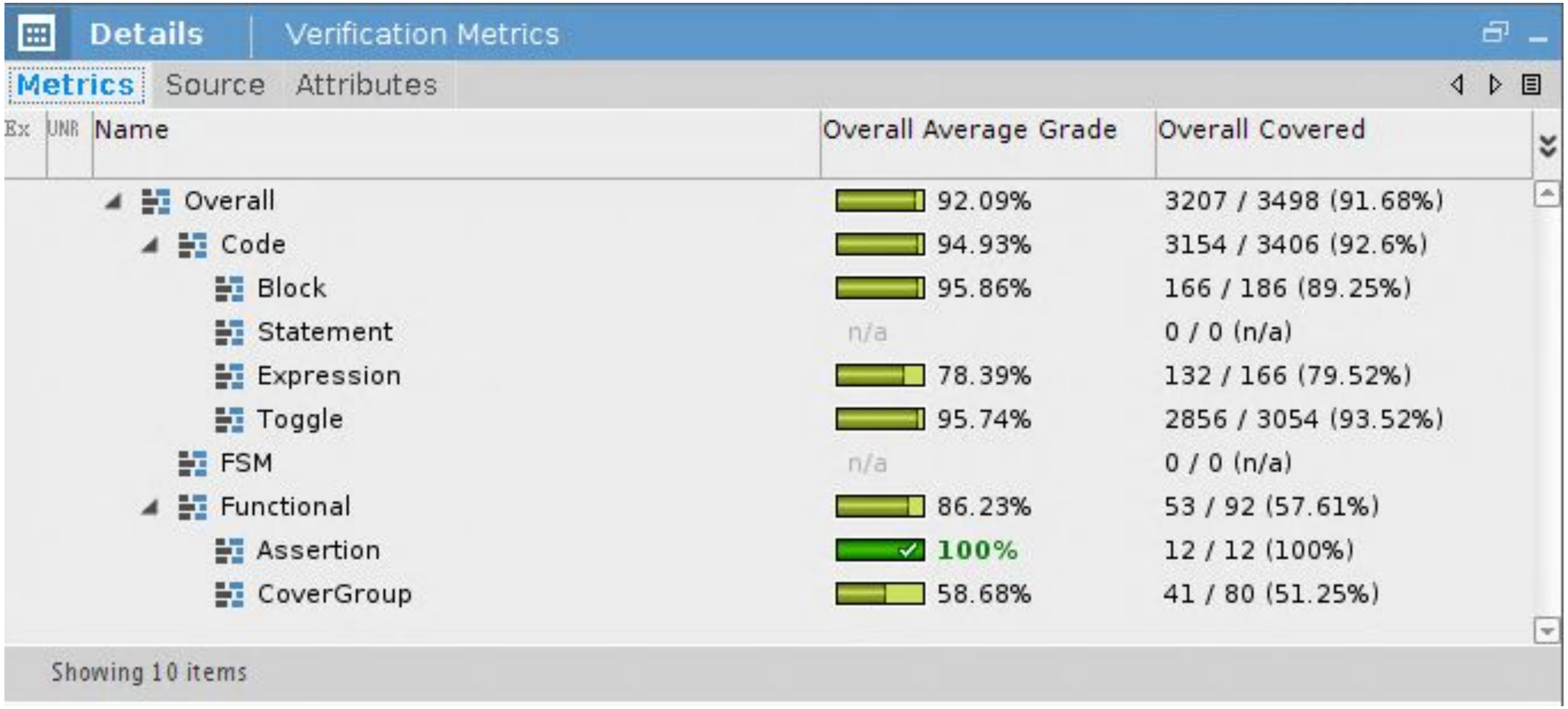


Verification Hierarchy

Ex	UNR	Name	Overall Average Grade	Overall Covered
		(no filter)	(no filter)	(no filter)
		Verification Metrics	81.88%	2920 / 3493 (83.6%)
		Types	77.96%	1472 / 1784 (82.51%)
		Instances	85.79%	1448 / 1709 (84.73%)
		uvm_pkg	n/a	0 / 0 (n/a)
		$unit	n/a	0 / 0 (n/a)
		testbench	85.79%	1448 / 1709 (84.73%)
		dut_if1	71.43%	90 / 126 (71.43%)
		img_proc_acc	85.96%	1351 / 1576 (85.72%)
		arbiter	92.54%	681 / 807 (84.39%)
		kind_of_scheduler	93.2%	361 / 446 (80.94%)
		fDUT	98.75%	87 / 89 (97.75%)
		processor	83.96%	504 / 565 (89.2%)
		threshold	71.36%	219 / 269 (81.41%)
		brightness	99.17%	137 / 140 (97.86%)

Results

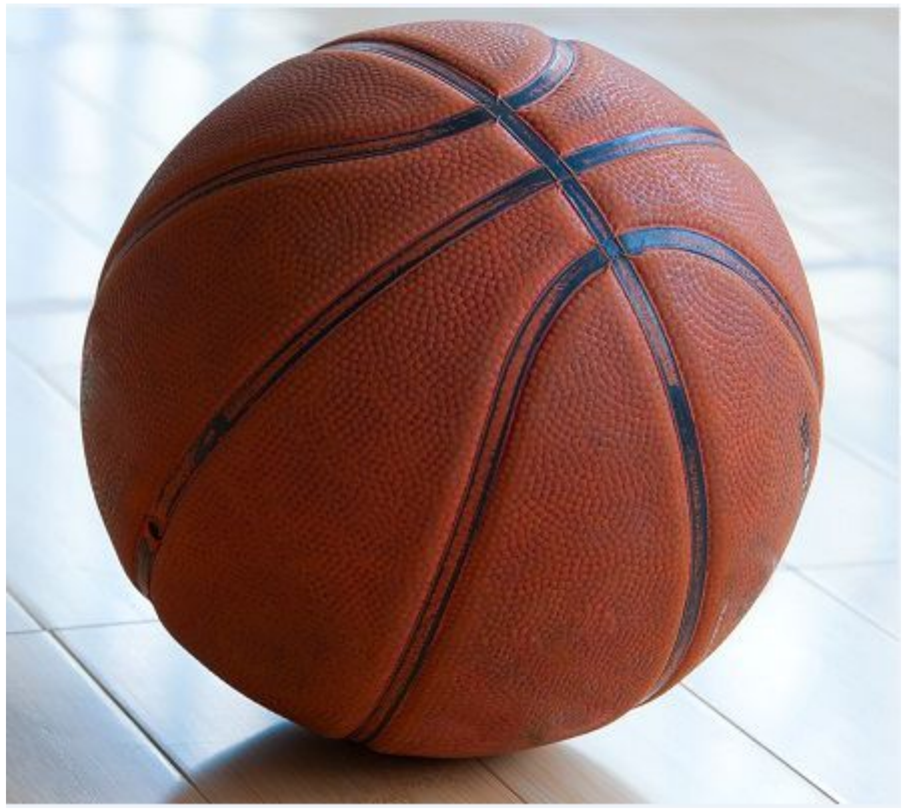
We ran 200 images, 100 per slave:

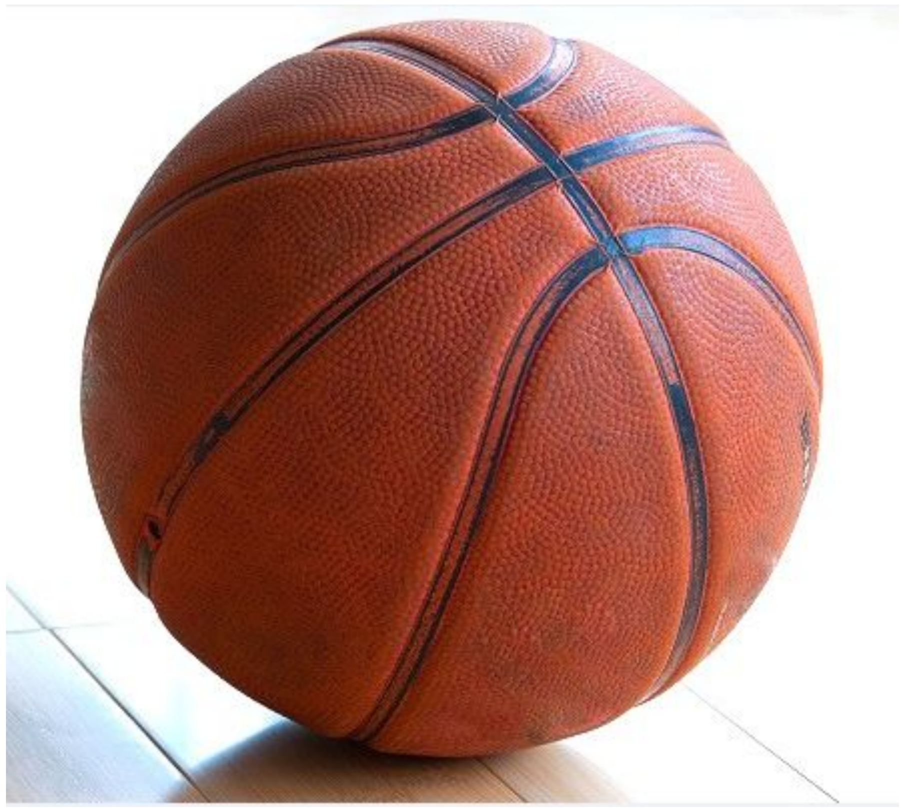


Ex	UNR	Name	Overall Average Grade	Overall Covered
		Overall	92.09%	3207 / 3498 (91.68%)
		Code	94.93%	3154 / 3406 (92.6%)
		Block	95.86%	166 / 186 (89.25%)
		Statement	n/a	0 / 0 (n/a)
		Expression	78.39%	132 / 166 (79.52%)
		Toggle	95.74%	2856 / 3054 (93.52%)
		FSM	n/a	0 / 0 (n/a)
		Functional	86.23%	53 / 92 (57.61%)
		Assertion	✓ 100%	12 / 12 (100%)
		CoverGroup	58.68%	41 / 80 (51.25%)

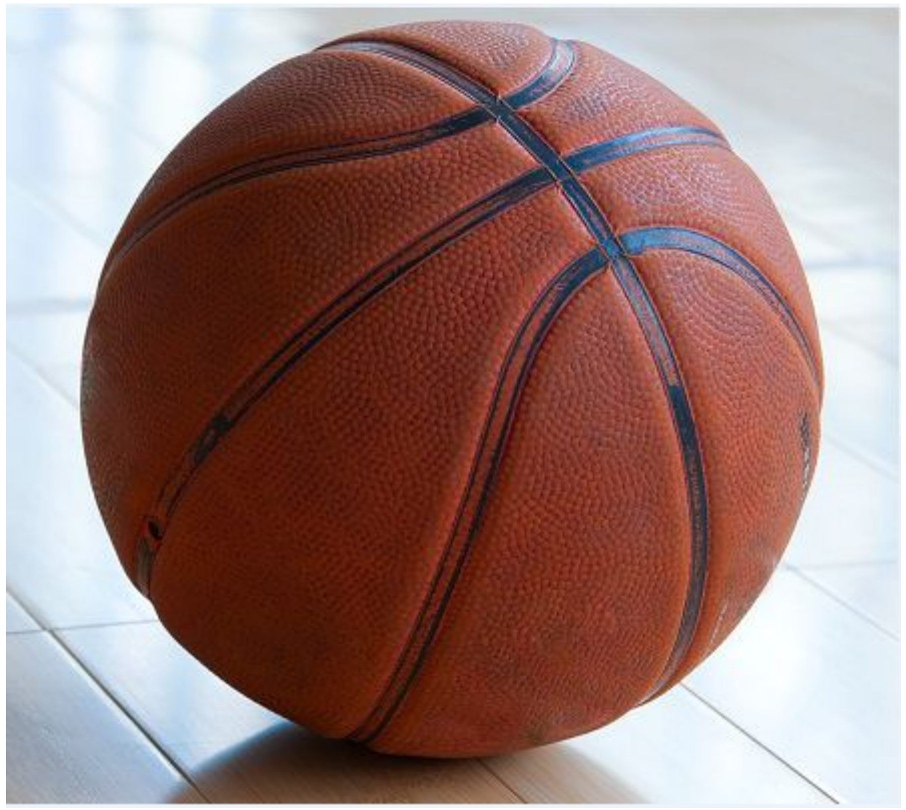
Showing 10 items

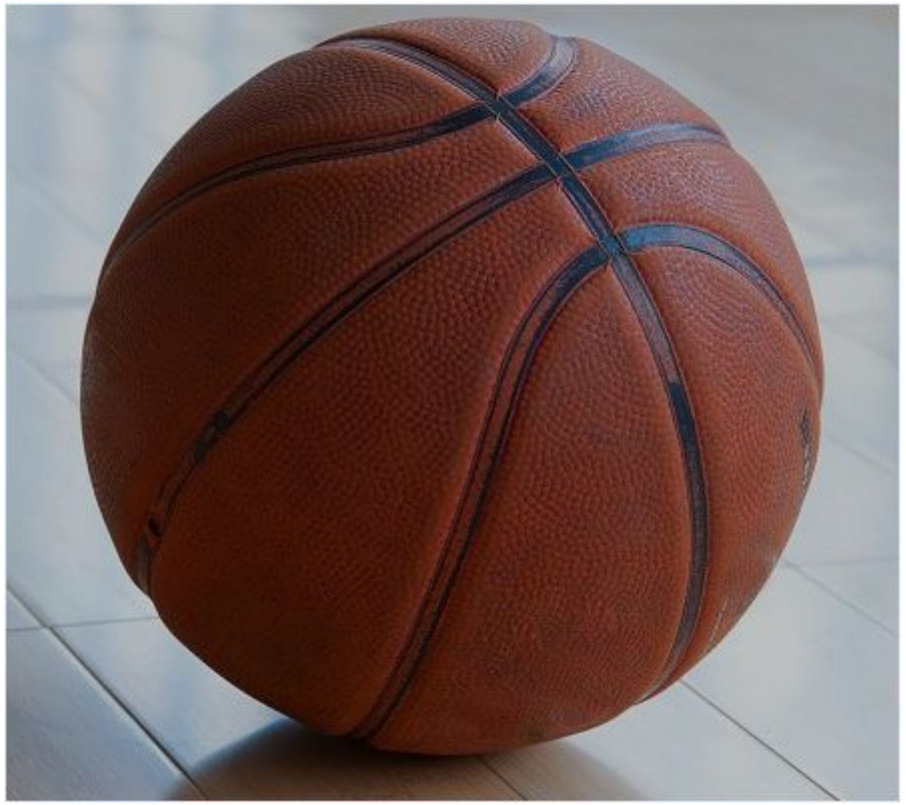
Brightness (+100)



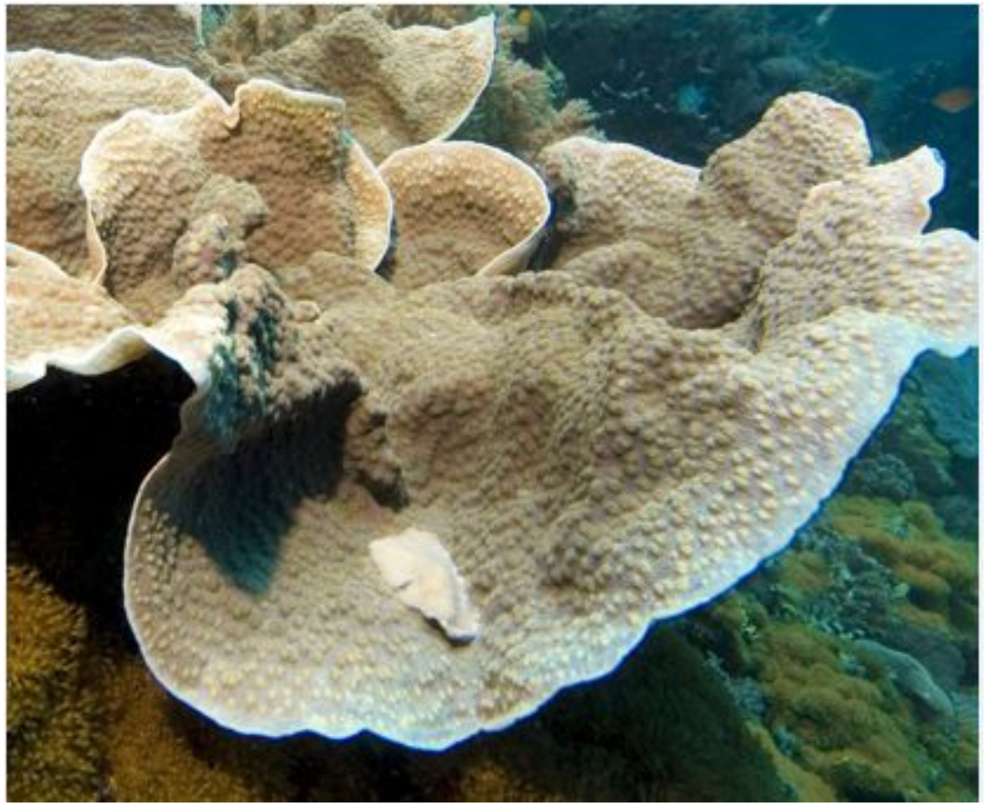


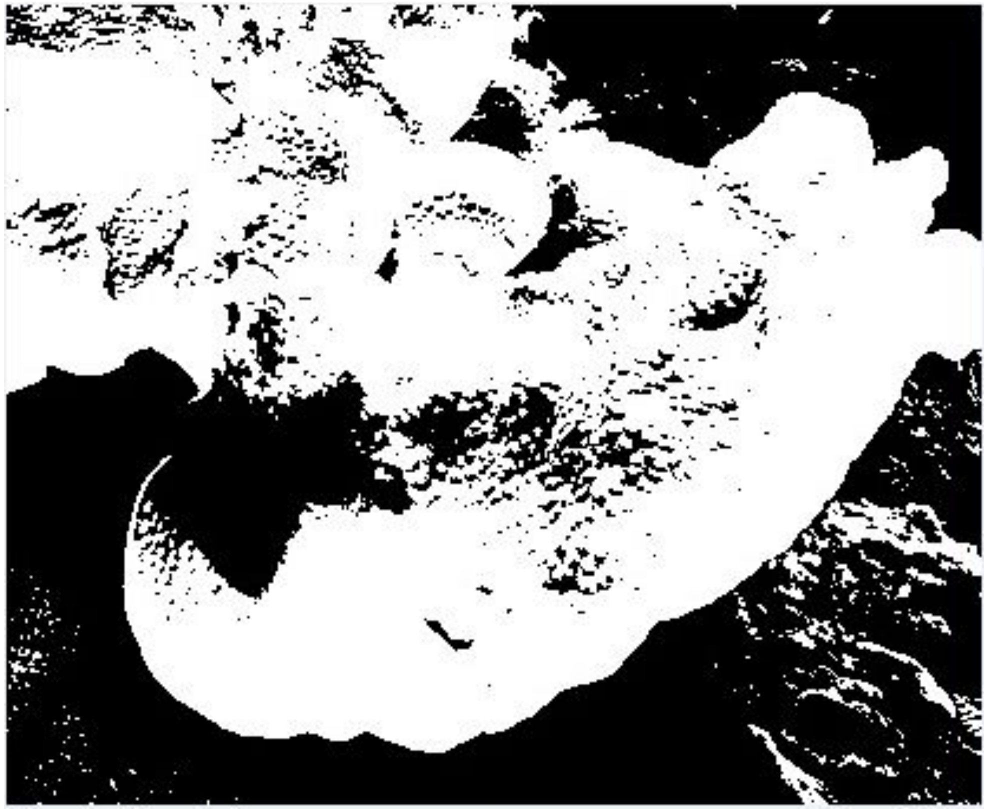
Brightness (-100)





Threshold (200)





Challenges / Problems

1. The threshold mode was difficult to compute because of padding issue.
2. On the fly changes with the integration of the sub-modules of the DUT.
3. Running the sequences in parallel - didn’t work at start but was solved later.
4. DUT bugs which stalled the verification environment.
5. Because we chose to focus on the whole DUT, it was hard to reach corner cases of the sub-modules.
6. Running many large images - took too much time due to the resources we had.

What’s Next?

- Take care of the bugs found up to now
- Analyze coverage results
- Regression
- Direct the random inputs to get more cover on the groups defined
- Show an histogram of the image sizes we tested
- Test the 64bit width
- To come up with more special cases to test