Getting started with Co-emulation

Primer on Why and How to Transition your Design and UVM Testbench to an Emulator

Jigar Savla

Juniper Networks

October 23, 2018

Austin

Agenda

Introduction

Co-Emulation?

Architecture

Design changes

Testbench changes

Guidelines + Planning

Conclusion

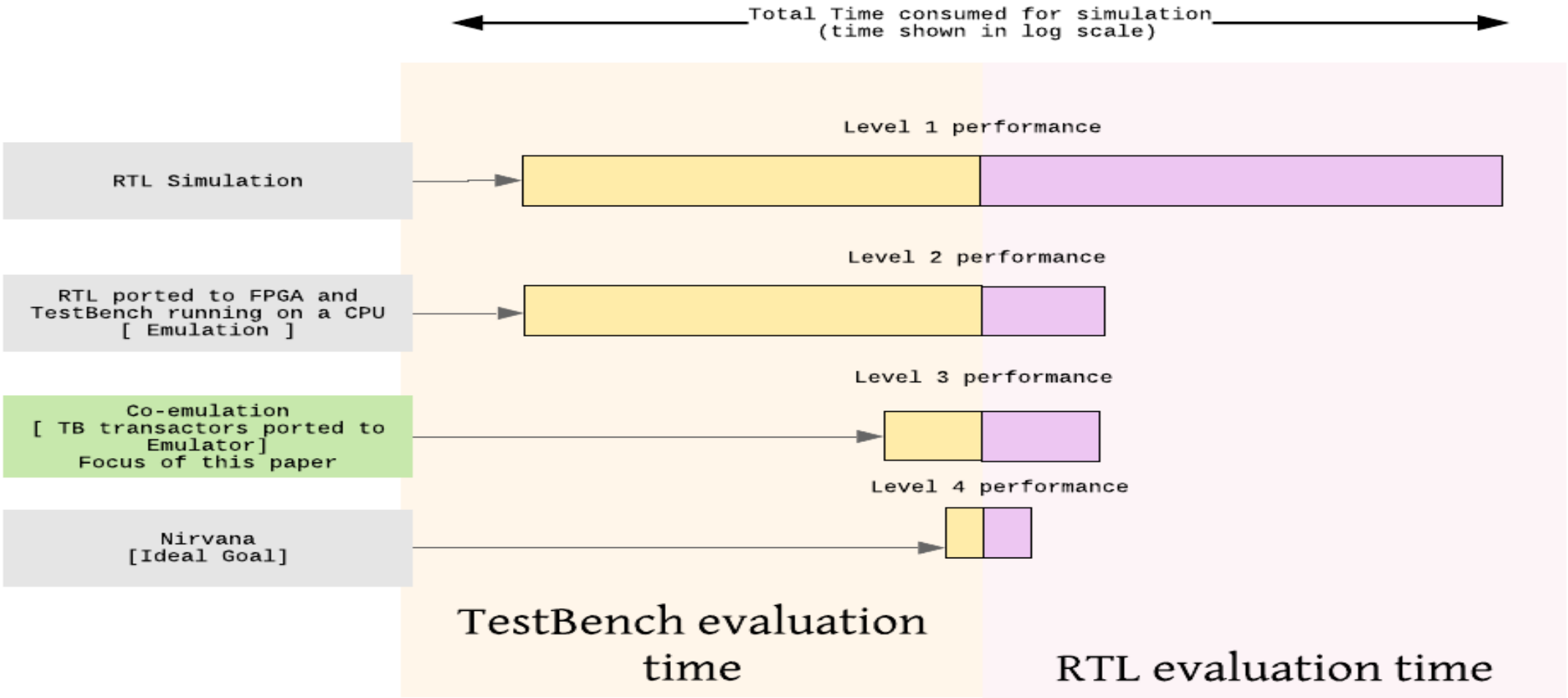
Co-emulation?

Sub-goals:

1. TB speedup
2. Software / Driver bringup

Co-emulation?

1. TB speedup



Co-emulation?

Good test candidates

We found that performance, long running tests which have already undergone significant cleanup (high level of testing and running clean for some time) in simulation are great candidates.

- Performance tests - be careful of Memory vendor models
- Long running tests
- Reproduced *wedges*
- Host interface connectivity check
- Infrastructure check test (with IOs)

Remember that in emulators, you generally have support for only two state values. Tests which depend on ‘X’ and ‘Z’ values, are better checked using X-prop and Formal solutions.

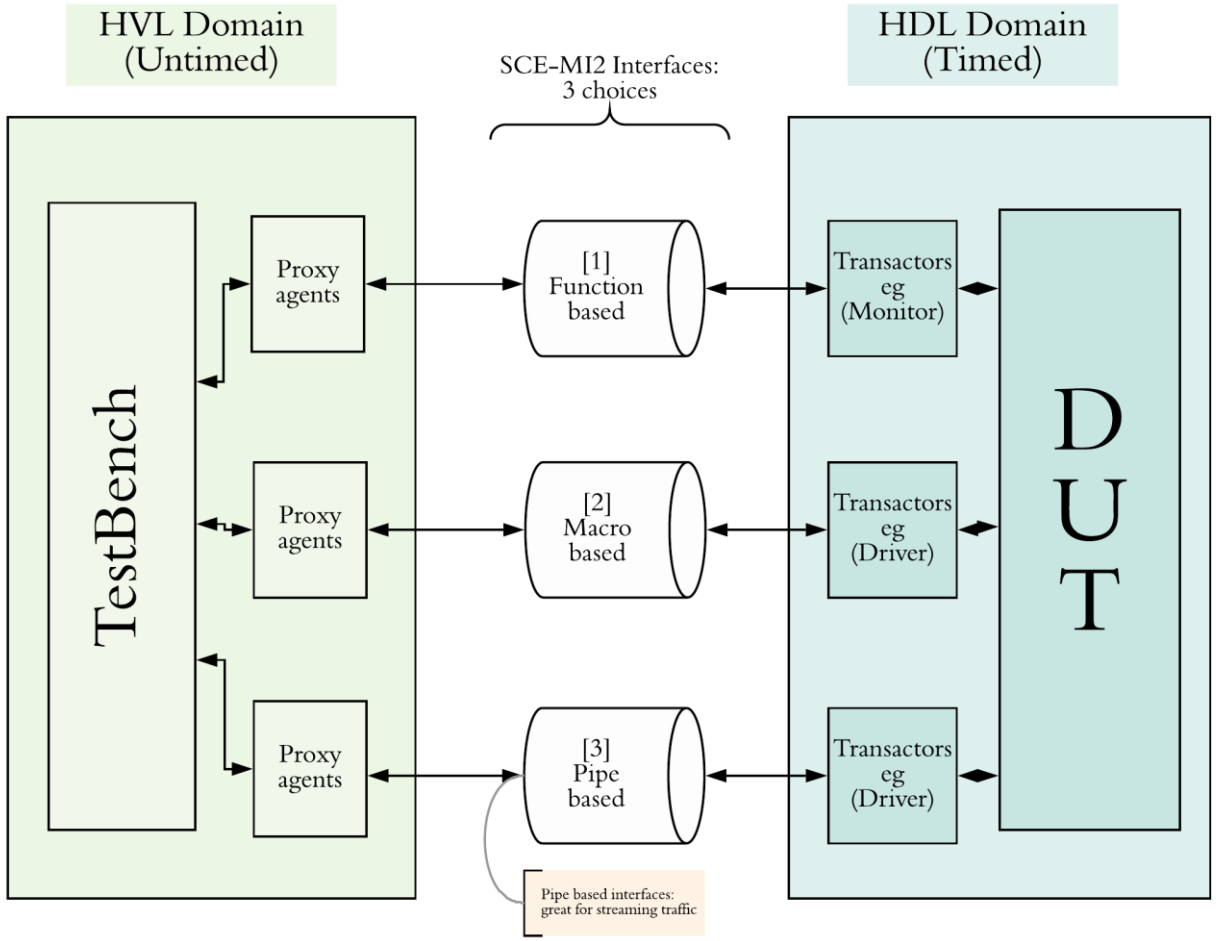
Architecture

SCE-MI2 modes of communication

Introduction
Co-Emulation?
Architecture
Design changes
Testbench changes
Guidelines + Planning
Conclusion

Architecture

SCE-MI2 modes of communication



Design changes

Isn’t RTL design always synthesizable and hence portable?

Design changes

Isn’t RTL design always synthesizable and hence portable?

```
module test_design ( input clock,
    …
  );

always_ff @ (posedge clk) begin : test_block
  // some RTL code
end

// synopsys translate_off
// assert start

property test_property (clk);
…
endproperty: test_property

test_assert : assert property
(test_property(clk)) else
$display(“test_assert failed”);

// assert end

// fcov start
  // A Functional coverage instantiation here
// fcov end

// synopsys translate_on

endmodule
```

Design changes

Potential design problems and ways to address them

1. Multiple clock domains. Analog logic such as Serdes or PLL's must be replaced with simple digital models.
2. Mimicing atomic multiple register reads in passive mode
3. Block or system configuration
4. If writing a command line variable grab, can't use $plusargs but have to tie it to a pin which can toggle.
5. Avoid concurrent statements in SVAs or guard them with another pragma.
6. Analyze your SVAs and FCovs for vacuous passes
7. To reduce state space issues while dealing with SVAs and FCov, code up some small combo logic (RTL) to set the trigger and counting bits, instead of having it all sit in the property itself.

TB changes

Interface

Monitor

Driver

TB changes

Key things to remember & Driver code

1. Keep the HVL and HDL sections separate.
2. Keep the HVL sections untimed. Most UVM code is naturally untimed. Use events wherever time is needed. We do this to avoid needing clocks in HVL untimed domain. (eg: Incorrect Driver code: Will have ##s instead of @vi.driver_cb)
3. HDL sections to not have any non-synthesizable code.
4. Aggregate sequences

```
// Transmit packet to model and DUT
task send_to_dut( my_packet pkt );

 // DONT use any ##1 here !
   vif.send_trans_to_dut(1'b1,
pkt.data);

endtask : send_to_dut
```

```
// Drive idles
 task assign_idle(my_packet pkt = '0 );

  logic [7:0] data_to_be_sent = (pkt != '0)
pkt.data : $urandom();
   vif.send_trans_to_dut(1'b0,
data_to_be_sent);

endtask : assign_idle
```

TB changes

Synopsys Users Group
Austin

Virtual Interface++ , Monitor

```
task send_trans_to_dut ( input logic vld_i, input logic[7:0] data_i);
  @(drive_from_sender_cb);
  vld     <= vld_i;
  data    <= data_i;
endtask : send_trans_to_dut

// generic clock to be used anywhere in driver, monitor
task wait_one_clk();
  begin
    @(drive_from_sender_cb);
  end
endtask
task wait_one_drv_credit_clk();
  begin
    @(drive_from_sender_cb);
  end
endtask
```

emulation compatible monitor

```
// Collect Responses
task my_monitor::collect_ack_response();
    logic ack, vld;
    logic [7:0] data;
    forever begin
        vif.get_monitor_trans(.vld_m(vld), .ack_m(ack), .data_m(data));
        if (ack)  bp_ack_port.write(num_ack_col);
        ...
        end
endtask : collect_ack_response
// Collect packets
task my_monitor::collect_packet();
    logic ack, vld;
    logic [7:0] data;
    my_packet packet_collected;
    packet_collected = my_packet::type_id::create("packet_collected", this);
    forever begin
        if (!reset) begin
            vif.get_monitor_trans(.vld_m(vld), .ack_m(ack),
.data_m(packet_collected.data));
            if (vld) begin
                        sb_port.write(packet_collected);
            end
        end
    end // forever begin
endtask : collect_packet
```

Guidelines + Planning

Summary

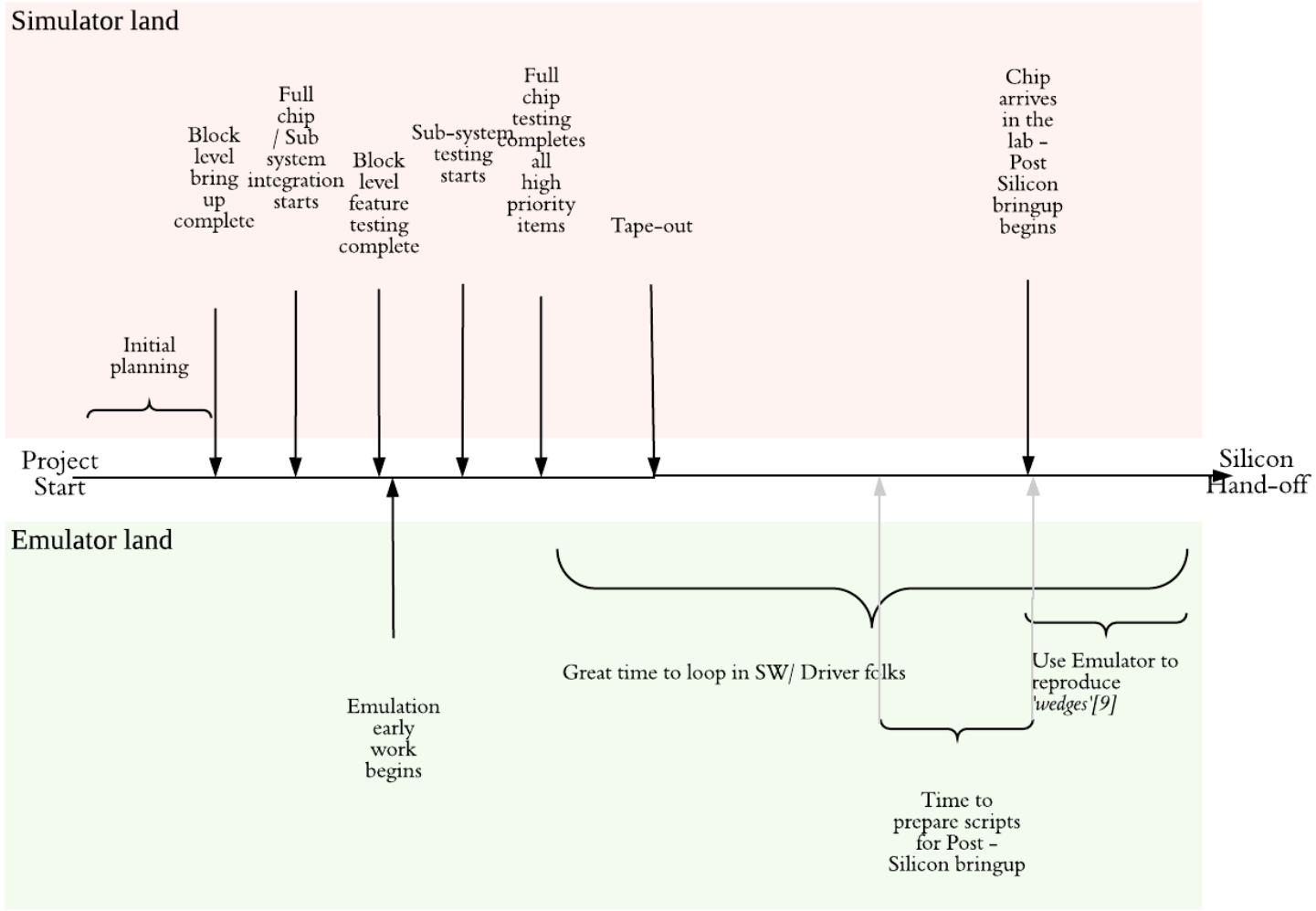
Guidelines + Planning

Guidelines – What helped us

- Instead of doing inline or end of file additions, create a separate file with your SVAs and FCovs, So that you can create file-lists during compilation.
- Don’t use ##1 anywhere. Clocking blocks exist for a reason. For the rest, use events.
- Don’t peek / poke into the RTL. Have the Design expose it as a register.
- In config_db, every time you have a set or a get with “*” as your context or Instance name.

Guidelines + Planning

Planning



Conclusion

Results

Conclusion

Results

- Our networking chips don’t have widely used benchmarks like SPEC for CPUs, ResNet inference for ML, DL ASICs. What we do have our throughput benchmarks.
- Across the board, we **saw 5x improvement in run times**. Our best improvement was on the order of **100x for a full fledged UVM TB**. Clearly, we have more room to improve.
- For a rather barebones UVM TB, we saw even higher improvement in run times.

Synopsys Users Group
Austin

Thank You

snug@jigarsavla.com

Complete The Survey For This Presentation

- From the App, access from the “All Surveys” widget on the home screen or listed on the left navigation
- Select **11:30/11:45 am Session Surveys**
- Select this presentation title and rate using a 5-star scale
- Enter any additional feedback and **hit submit**