Verification Without DUT

Thinh Ngo
Broadcom

October 8, 2014
SNUG Canada

Agenda

RTL, Testbench & Coverage Development

Testbench Development

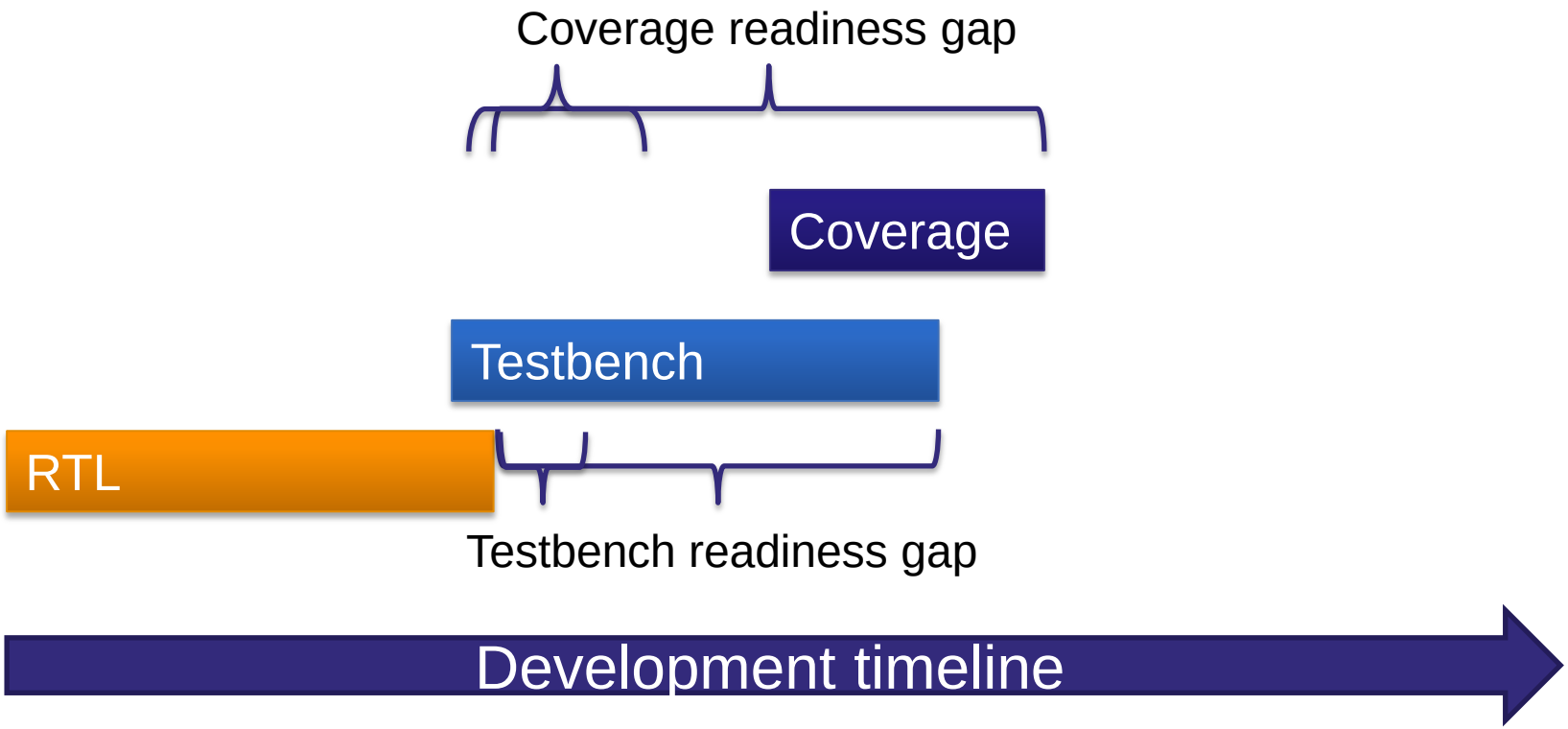
Virtual DUT & Verification

Examples

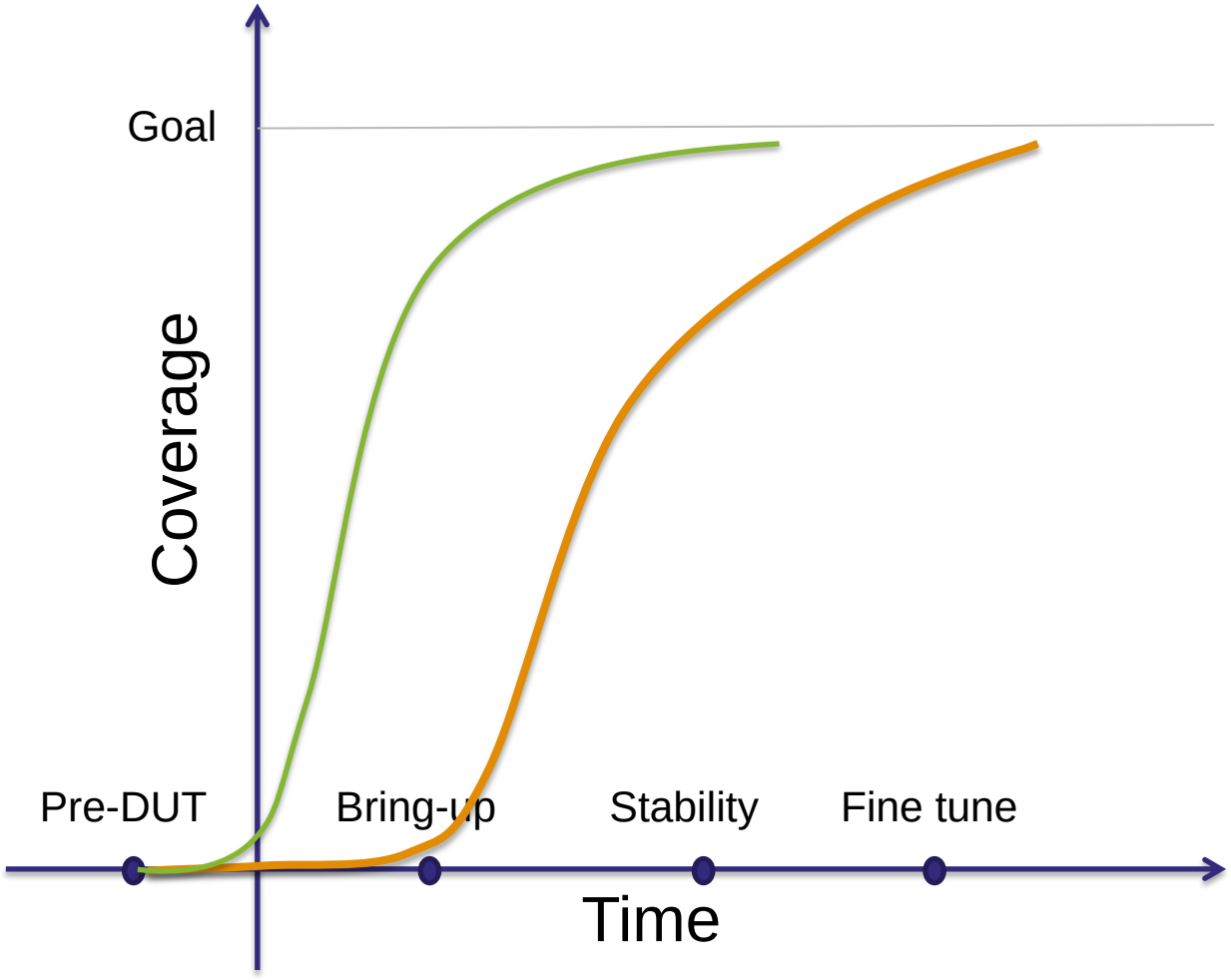
Benefits & Considerations

Summary

Development Timetable



Testbench Development



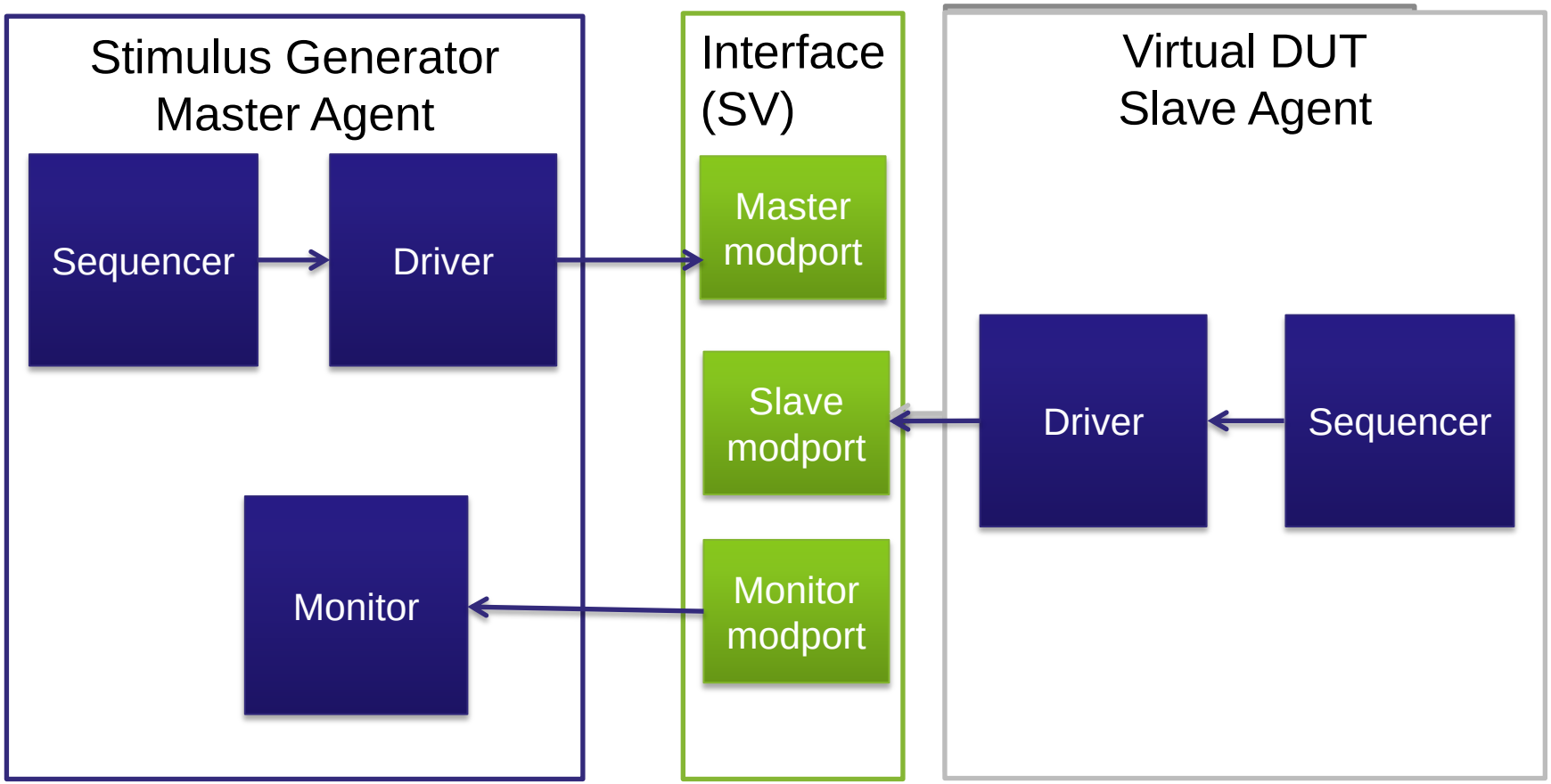
Testbench Development
Before DUT is ready

- Is “free” as it overlaps with RTL coding
- The more that can contribute to the testbench maturity after the DUT is ready the less left to be done afterwards
- Is limited by the inability to simulate

Virtual DUT & Virtual Verification Concept

- Virtual DUT is a testbench component used in place of the DUT to enable testbench simulation
  - To facilitate testbench development including stimulus generators, drivers, monitors, checkers, coverage and assertions
  - By providing needed responses to the testbench to allow transactions to flow between the testbench and the virtual DUT
  - Not a full model of the DUT – much simpler
- Virtual verification is verification with a virtual DUT

Virtual DUT for Stimulus Generator



1

Virtual DUT for Stimulus Generator (2)

Provides transactions with needed responses

Example:

- Memory read/write transaction generator
- Provides read data and acknowledgement

Shared interface code:

```
interface mem_if (input logic clk, rstb);
  logic [31:0] addr, wdata, rdata;
  logic req, wren, done;

  clocking mcb @(posedge clk);
    output req, addr, wren, wdata;
    input done, rdata;
  endclocking

  clocking scb @(posedge clk);
    input req, addr, wren, wdata;
    output done, rdata;
  endclocking

  modport mst(clocking mcb);
  modport slv(clocking scb);
endinterface
```

Virtual DUT for Stimulus Generator (3)

Synopsys Users Group

Stimulus Generator Driver

```
task drive_txn(_txn txn);
  vif.mst.mcb.req   <= 1'b1;
  vif.mst.mcb.addr  <= txn.addr;
  vif.mst.mcb.wren  <= txn.wren;
  vif.mst.mcb.wdata <= txn.wdata;
  wait(vif.mst.mcb.done == 1);
  txn.rdata <= vif.mst.mcb.rdata;
  repeat(txn.delay)@(vif.mst.mcb);
endtask: drive_txn
//reset interface after each txn
task reset_signals();
```

Virtual DUT driver

```
task drive_txn(_txn txn);
  wait(vif.slv.req == 1);
  repeat(txn.delay)@(vif.slv.scb);
  vif.slv.scb.done  <= 1'b1;
  vif.slv.scb.rdata <= txn.rdata;
endtask: drive_txn

task reset_signals();
```

Virtual DUT for Stimulus Generator (4)

Synopsys Users Group

Stimulus Transaction

```
class _txn extends
uvm_sequence_item;
  bit req = 1'b1;
  rand bit [31:0] addr;
  rand bit wren;
  rand bit [31:0] wdata;
  rand int delay;

  constraint c_delay
    {delay inside {[0:3]};}
  …
endclass
```

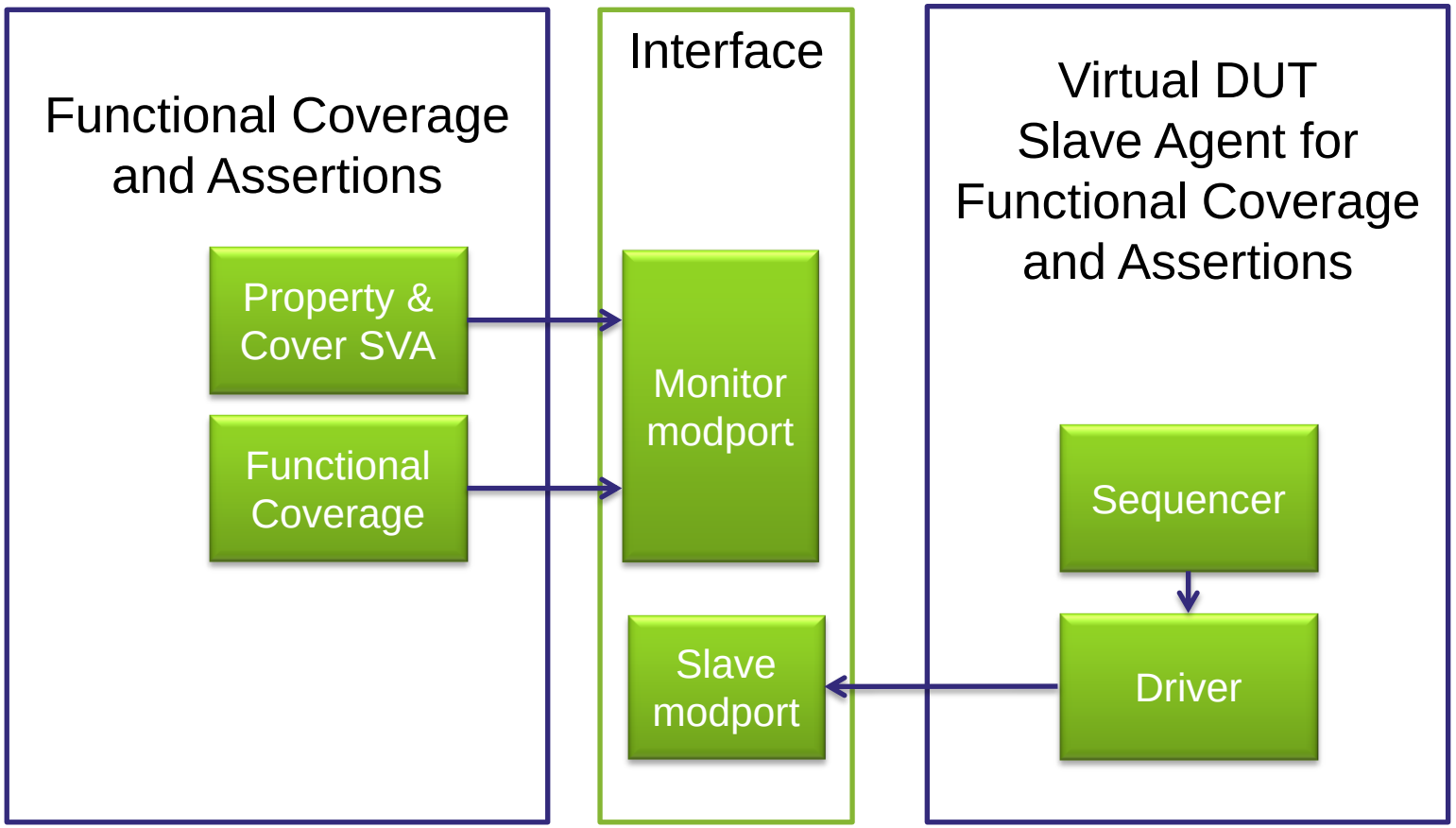
Virtual DUT Transaction

```
class _txn extends
uvm_sequence_item;
  bit done = 1'b1;
  rand bit [31:0] rdata;
  rand int delay;

  constraint c_delay
    {delay inside {[0:3]};}
  …
endclass
```

- No DUT initialization – fast simulation

Virtual DUT for Coverage & Assertions



Virtual DUT for Coverage & Assertions (2)

- Provides transactions/sequences with necessary signals for 100% coverage
- Example 1:
  - A covergroup with coverpoints and cross coverage
  - Provides transactions to trigger all coverpoints and cross coverage
- Example 2:
  - A property assertion or cover assertion
  - Provides sequences of transactions to assert needed signals at the required timing relationship

Virtual DUT for Coverage & Assertions (3)

Synopsys Users Group

Functional Coverage

```
enum { red, green, blue } color;
bit [3:0] pixel_adr, pixel_offset,
pixel_hue;

covergroup g2 @(posedge clk);
  Hue: coverpoint pixel_hue;
  Offset: coverpoint pixel_offset;
  AxC: cross color, pixel_adr;
  all: cross color, Hue, Offset;
endgroup
```

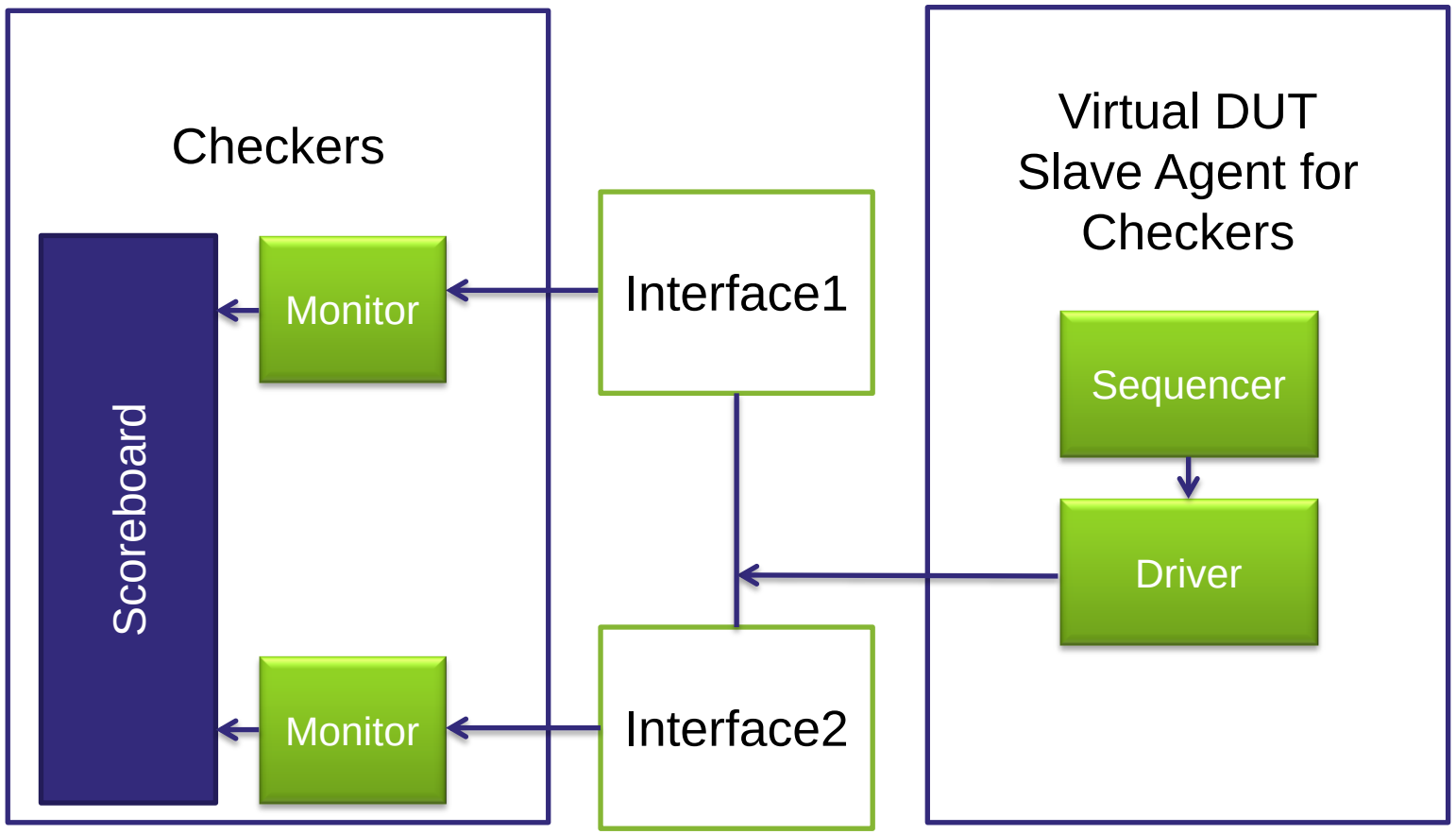
Virtual DUT Transaction

```
class _txn extends
            uvm_sequence_item;
  rand color;
  rand bit [3:0] pixel_adr;
  rand bit [3:0] pixel_offset;
  rand int unsigned delay;

  constraint c_delay
    {delay inside {[0:3]};}
  …
endclass
```

- Constrained random responses to trigger coverage

Virtual DUT for Checkers



Virtual DUT for Checkers (2)

- Provides transactions/sequences with necessary triggering signals for every checker (e.g. checker coverage)
- Example:
  - Exception checkers (i.e. underflow, overflow)
  - Provides transactions/sequences to trigger all exception types
  - Ignores errors

Virtual DUT for Checkers (3)

Synopsys Users Group

Checker Coverage

```
If (signal_A) do_check_signal_A;
If (signal_B) do_check_signal_B;
…
If (txn_A) do_check_txn_A;
If (txn_B) do_check_txn_B;
```

- Constrained random responses to exercise checkers

Virtual DUT Transaction

```
class _txn extends
     uvm_sequence_item;
  rand bit signal_A;
  rand bit signal_B;
  rand bit txn_A;
  rand bit txn_B;
  rand int unsigned delay;

  constraint c_delay
   {delay inside {[0:3]};}
  …
endclass
```

Virtual DUT
Benefits & Considerations

- Testbench is up and running when RTL arrives
  - Shift Left => Plug and ~~Play~~ Verify
  - Testbench components are more mature
  - At integration, focus on RTL bugs, not testbench problems
- Faster compile & simulation without RTL initialization
  - TB compilation under a minute, vs. 15 min for full system
  - Edit-compile-run in a minute vs. 30 minutes
- Experiment and debug stimulus constraints
  - Perform what-if analysis
  - Interactive debug in Verdi or DVE is quick
  - Explore OOP code structure in debugger

Virtual DUT
Benefits & Considerations (2)

- Virtual DUT is simple to build
  - Shared interfaces w/ master/slave connectivity
  - Mirrored drivers & transactions
  - SystemVerilog, UVM, VMM, OVM – not Verilog
  - Most VIPs (e.g. AXI) provide master and slave components
  - Testbench components (e.g. stimulus generators, checkers) can be independently developed
  - A virtual DUT for the stimulus generator of an 8-master arbiter UVM testbench was implemented in 2 days (see details in the paper)
- Post-DUT adjustments are needed such as signal names and their properties (e.g. timing)
- The better the interfaces/spec are defined, the less post-DUT adjustments are needed

2

Summary

- Verification without DUT - virtual verification
  - Virtual DUT
- Earlier testbench development
- Faster testbench debug
- More done pre-DUT, less left post-DUT
- Shorter testbench development time

Thank You