Designing Portable UVM Test Benches for Reusable IPs

XIAONING ZHANG
BAOSHENG WANG

FEBRURARY 2015

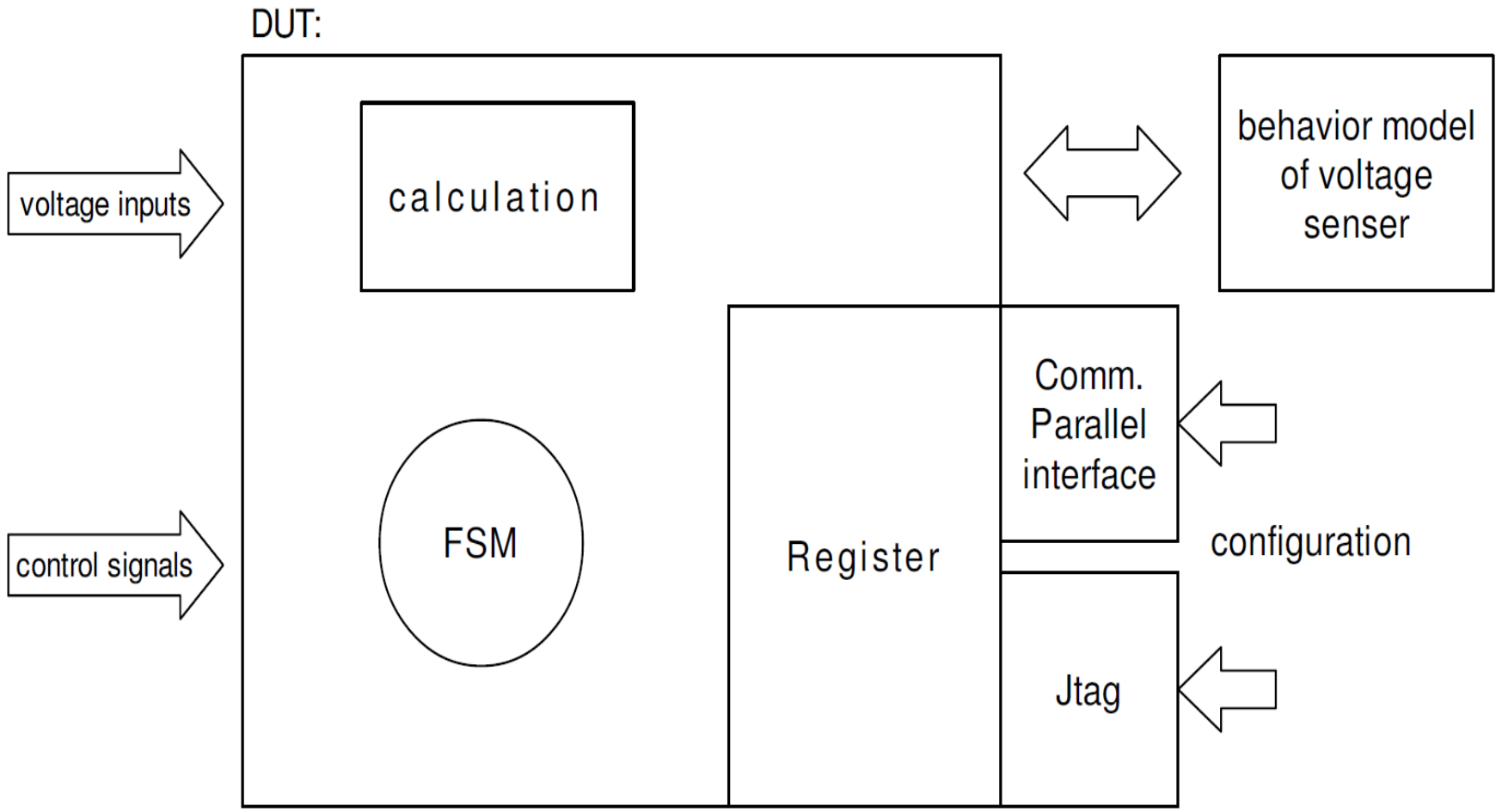
TEST BENCH FOR REUSABLE INTELLECTUAL PROPERTIES

AMD

- Introductions
  - IP reuse is a major design methodology for modern semiconductor industry
  - Exhaustively verifying highly reusable Intellectual Properties (IPs) is challenging due to multiple usage scenarios
  - Seamlessly reusing IP-level verification components at usage level requires well planning at an early test bench (TB) development stage because of various SoC environments
- TB Requirements for Highly Reusable IPs
  - Portable
  - Generic
  - Scalable
  - Capable for seamless integration
- Proposal
  - Generic UVM-based verification environments and development process
  - Integration-aware test bench

STEP 1: DUT ANALYSIS ON FUNCTIONALITY

- A highly reusable IP, DVR (Digital Voltage Regulator), is utilized as a demonstration example

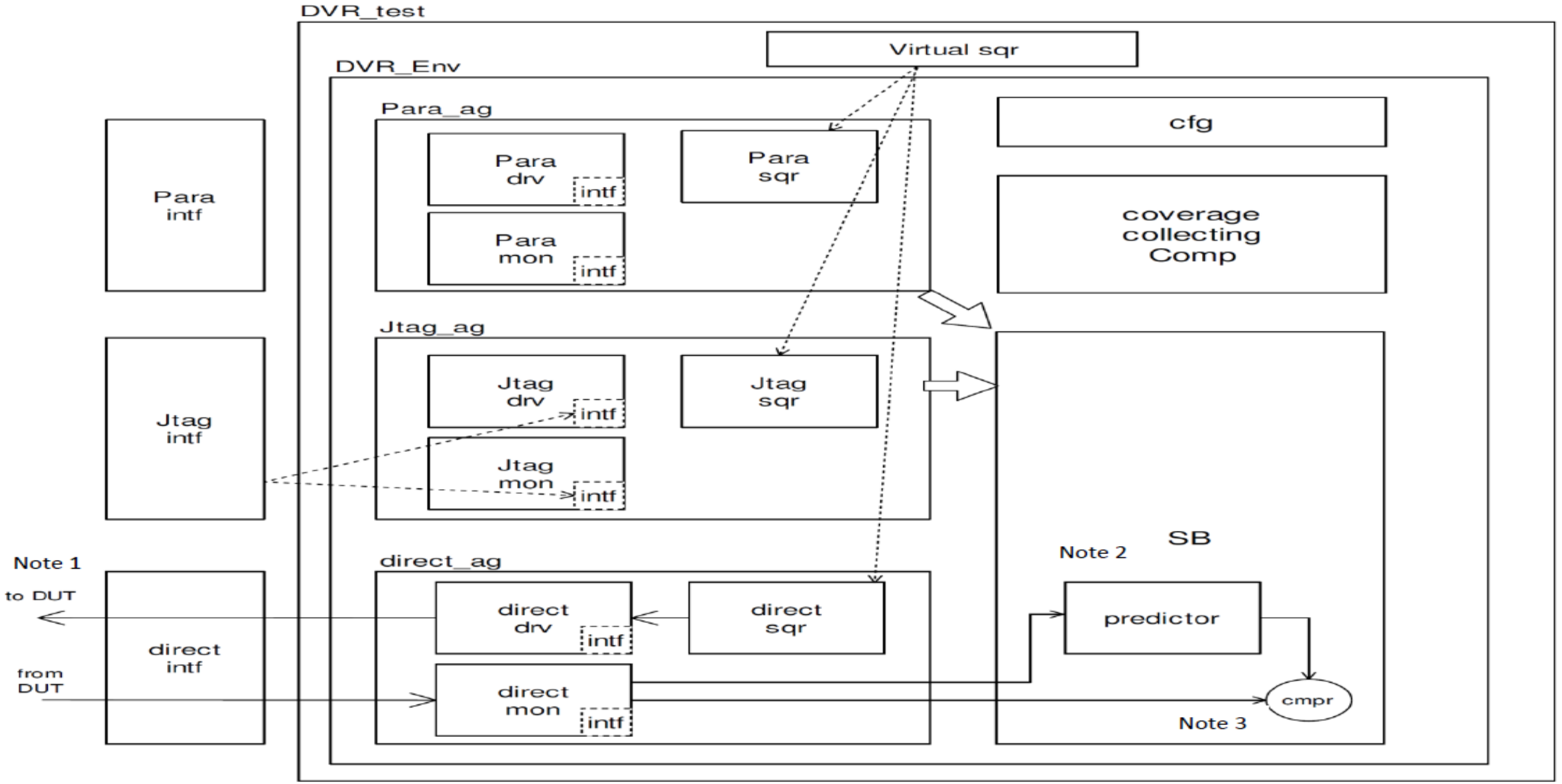


STEP 2: DUT ANALYSIS ON VERIFICATION CONSIDERATION

AMD

- Understand the DUT from verification perspective:
  - Two interfaces to configure internal registers
    - An Industrial-compatible JTAG interface
    - An AMD-private parallel bus interface
  - The DVR can accept an input voltage (reference voltage) and control the voltage of the block it manages in several states
    - An internal FSM for state control
    - Control signals and input voltage can be grouped together into the third interface (named as direct in this example)
  - A behavior model for the analog type of voltage sensor is provided by RTL designers for verification
    - RTL Designers verify its correctness by validating it against its real spice model
  - A general directory structure is recommended
    - UVCs: agent, driver, env, intf, include model, monitor, scoreboard, tb, tests, transactions and coverage
    - Verification automation: test_list, scripts

STEP 3: OVERALL UVC DESIGN



STEP 4: UVCS - AGENTS AND ENVIRONMENT

- Three interfaces, three agents:
  - JTAG agent
  - Common parallel agent
  - Direct agent
  - Each agent contains:
    - a sequencer
    - a driver
    - a monitor
  - The handle of virtual interfaces are passed down to driver and monitor inside agent through “uvm_config_db”
- The configuration object in the ENV can configure each agent to be active/passive modes
  - The mode switching is implementation through ifdefs

STEP 5: UVCS - SEQUENCE, SEQUENCER, SCOREBOARD AND COVERAGE

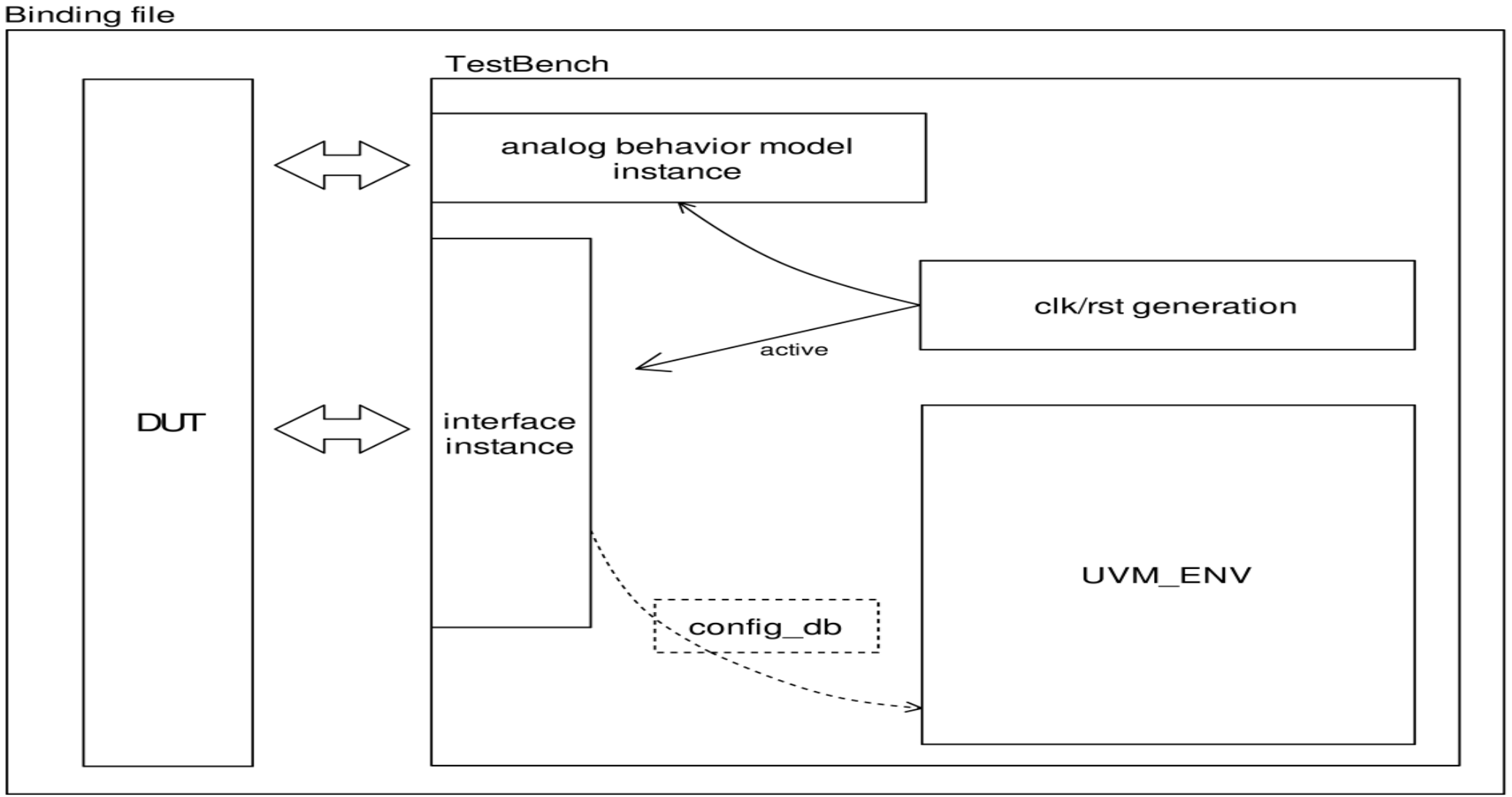
- There is a virtual sequencer containing handles to physical sequencers in agents

- For register access agents, the read sequence can take advantage of the "item_done(tx)/get_response(tx)" pair between driver and sequence to implement register checking test case.

- The scoreboard class contains predictors and comparators
  - There are two analysis ports and two queues in the comparator to receive and hold the transactions coming from two TLM analysis port connections, i.e. from predictor and real RTL output.

- Functional Coverage is implemented by callbacks
  - There is a covergroup container component instantiated in the ENV.
  - The handle of the component is passed down to callbacks "new ()" method.
  - The callbacks are called in the "run_phase" of the monitor.

STEP 6A: TEST BENCH LAYOUT AND CONNECTIVITY

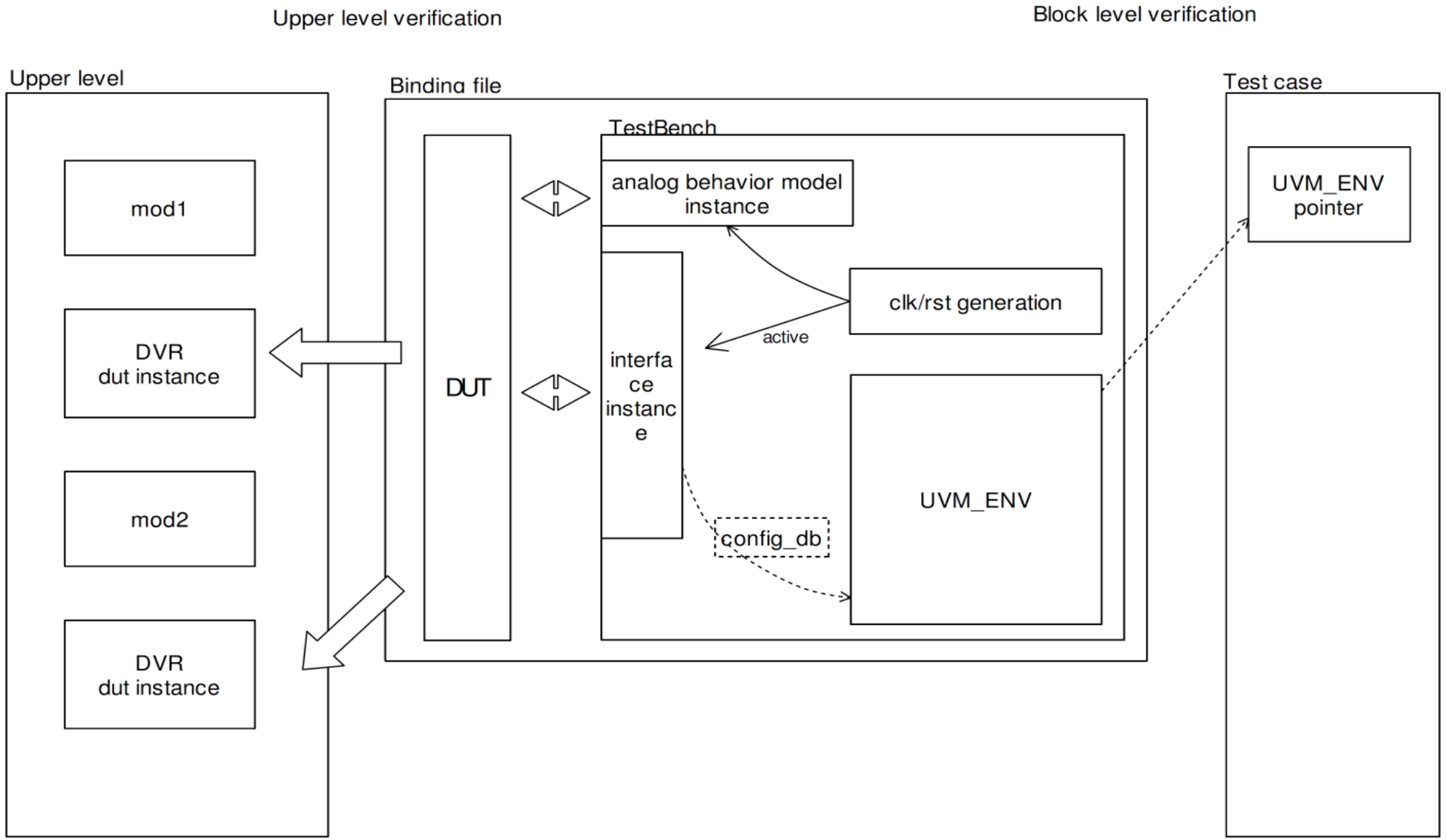


STEP 6B. TEST BENCH LAYOUT AND CONNECTIVITY

AMD

- Components in test bench module:
  - UVC instances
  - Interfaces
  - Clock/reset generation logic
  - SystemVerilog behavioral models
- Handling of TB module ports and signals
  - Ports: “inout” type, i.e., bidirectional ports.
  - Signals: “logic” type.
  - Connections: bidirectional since they are connected together through instance pin assignments
- Clocking blocks in UVM driver and monitor define the direction of interface signals
  - The input signals to DUT will be inside driver clocking block
  - All the signals are included in the monitor clocking block
- “ifdefs” are used to control the direction of the signals for active/passive modes switching

STEP 7A: INTEGRATION OVERVIEW



STEP 7B: UVC HANDLING FOR INTEGRATION

- Use SystemVerilog binding method to bind test bench module to DUT
  - TB is really portable
- TB UVCs at default are passive through ifdefs
  - There will be no test cases to start sequences on the sequencers and drivers path in block level UVC.
  - Sequencers and drivers path will be disabled.
  - configuration object is used to set passive to the mode variables inside each agent.
- New drivers at upper level
  - The signals in the interface which were driven by the UVM drivers in block level will now be driven by the upper-stream RTL in the upper level data-path.
- Scoreboard and reference models can be used at upper level
- Coverage collection is turned off at default to avoid potential performance issues at upper level

```
bind DVR_rtl DVR_tb_module#(
   .DVR_PARAM1   ( DVR_PARAM1 ),
   .DVR_PARAM2   ( DVR_PARAM2 ),
   .DVR_PARAM3   ( DVR_PARAM3 )
)
DVR_tb_inst(
//use .* if the tb port list is same as DUT
   .clk               ( clk           ),
   .rst               ( rst           ),
   .dvr_in_port1      ( dvr_in_port1  ),
   .dvr_in_port2      ( dvr_in_port2  ),
       ...
   .dvr_out_port1     ( dvr_out_port1 )
);
_
_
```

STEP 7C: CODE SNIPPETS FOR INTEGRATION

```
module DVR_tb_module #(
            ... ...
)(
inout wire clk,
inout wire rst,
inout wire dvr_in_port1,
   ...
inout wire dvr_out_port1
);

//import uvm pkg for using uvm configDB in the module
import uvm_pkg::*;


```

```
interface DVR_interface (input bit clk, input bit rst);
   logic drive_sig1;
       ...
   logic mon_sig1;

   clocking driver_clocking_block @(posedge clk);
               output drive_sig1;
               ...
   endclocking : driver_clocking_block

   clocking mon_clocking_block @(posedge clk);
               input drive_sig1;
               ...
            input mon_sig1;
   endclocking : mon_clocking_block

endinterface : DVR_interface
```

```
`ifdef PASSIVE_MODE_FOR_BLOCK_LEVEL
   assign clk = clk_tb;
   assign rst = rst_tb;
   assign dvr_in_port1 = dvr_interface_inst.drive_sig1;
       ...
   assign dvr_interface_inst.mon_sig1 = dvr_out_port1;
`else
   assign clk_tb = clk;
   assign rst_tb = rst;
   assign dvr_interface_inst.drive_sig1 = dvr_in_port1;
       ...
   assign dvr_out_port1 = dvr_interface_inst.mon_sig1;
`endif

//behavioral model instantiation and connectivity


//instantiate the UVC
//and pass the interface handle into it
DVR_UVC dvr_uvc_inst;
string  m_name;

initial begin
   m_name = $psprintf("%m");
   dvr_uvc_inst = dldo_env::type_id::create({m_name, ".dvr_uvc_inst"},
uvm_top);

   dvr_uvc_inst.interface_inst = dvr_interface_inst;
end
endmodule
```