2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION TAIWAN

Left-shifting Testbench Development Using Environment Inversion in UVM

Chi-Ming Li and Yu-Ju Su

Synopsys

SYNOPSYS®

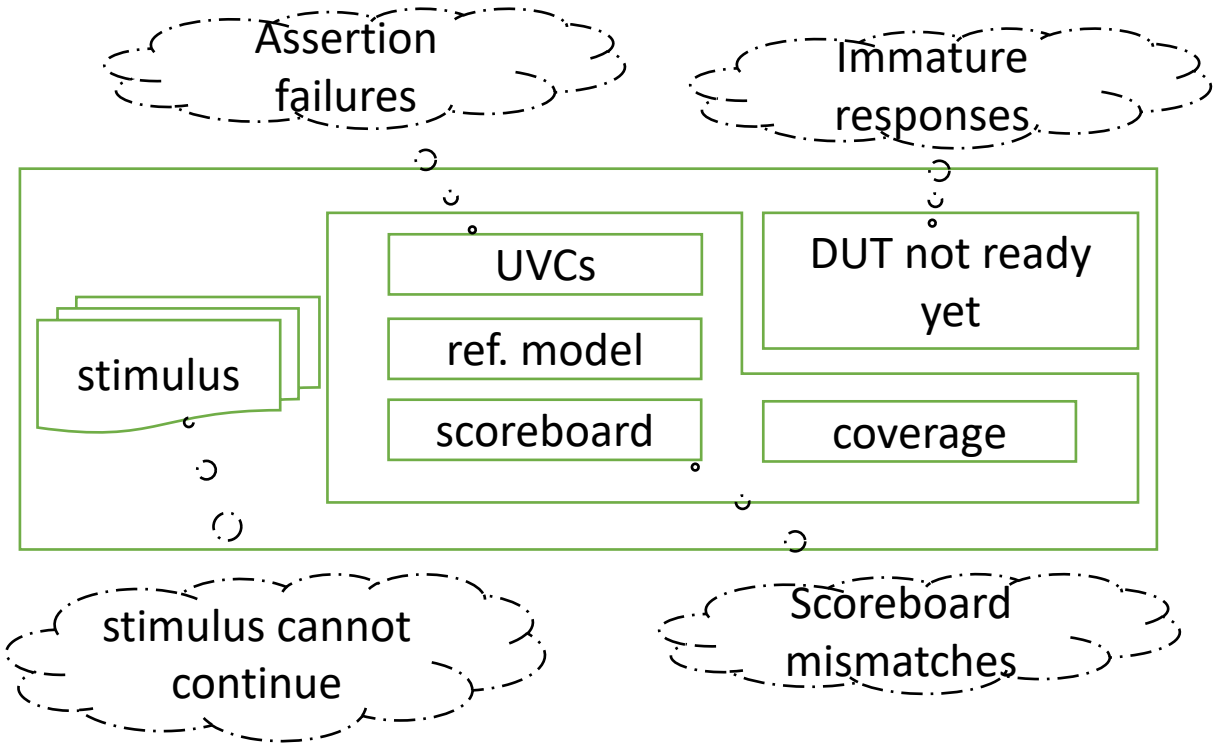
accellera SYSTEMS INITIATIVE

Agenda

- Motivation
- Test Double in Software Testing
- Design Under Test Double (DUTD)
- Environment Inversion for DUTD
- Conclusion

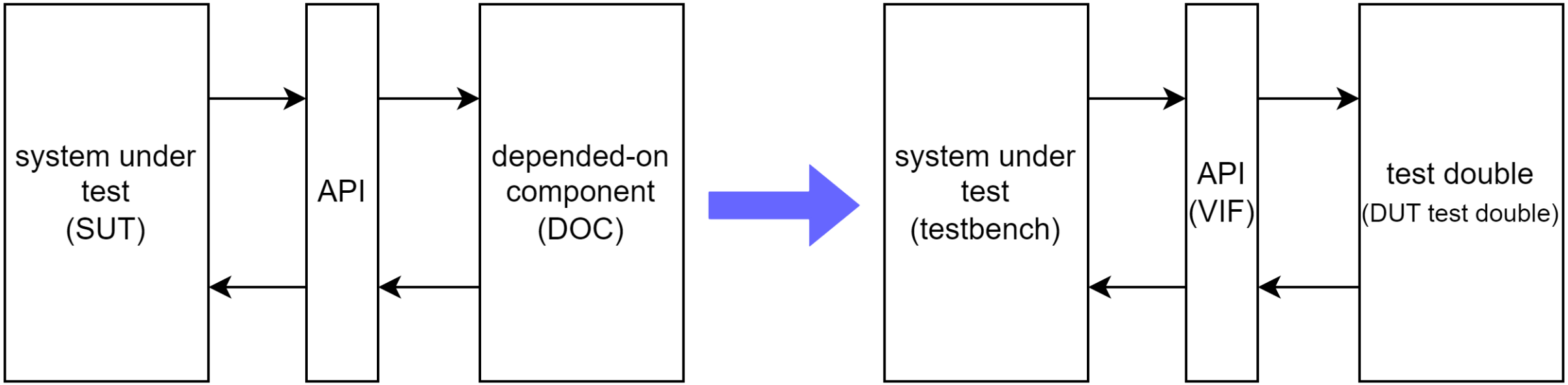
Motivation: Left-shift UVM TB

- TB's completeness depends on DUT's availability – need something responding stimuli
- Conventional either rely on immature design or dummy model – responses are not meaningful enough
- How can we leverage existing methodology and resources to create DUT's test double in an agile manner?



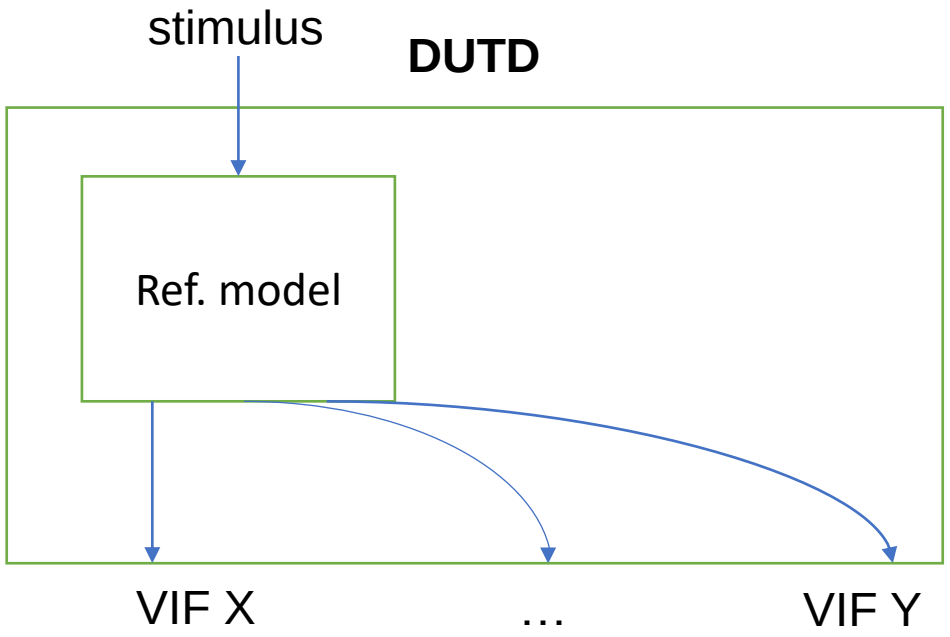
Test Double in Software Testing

- Test double is test-specific equivalent to depended-on component(DOC), only the same API is needed. It's much simpler and lighter weight implementation of the functionality provide by DOC. [1]



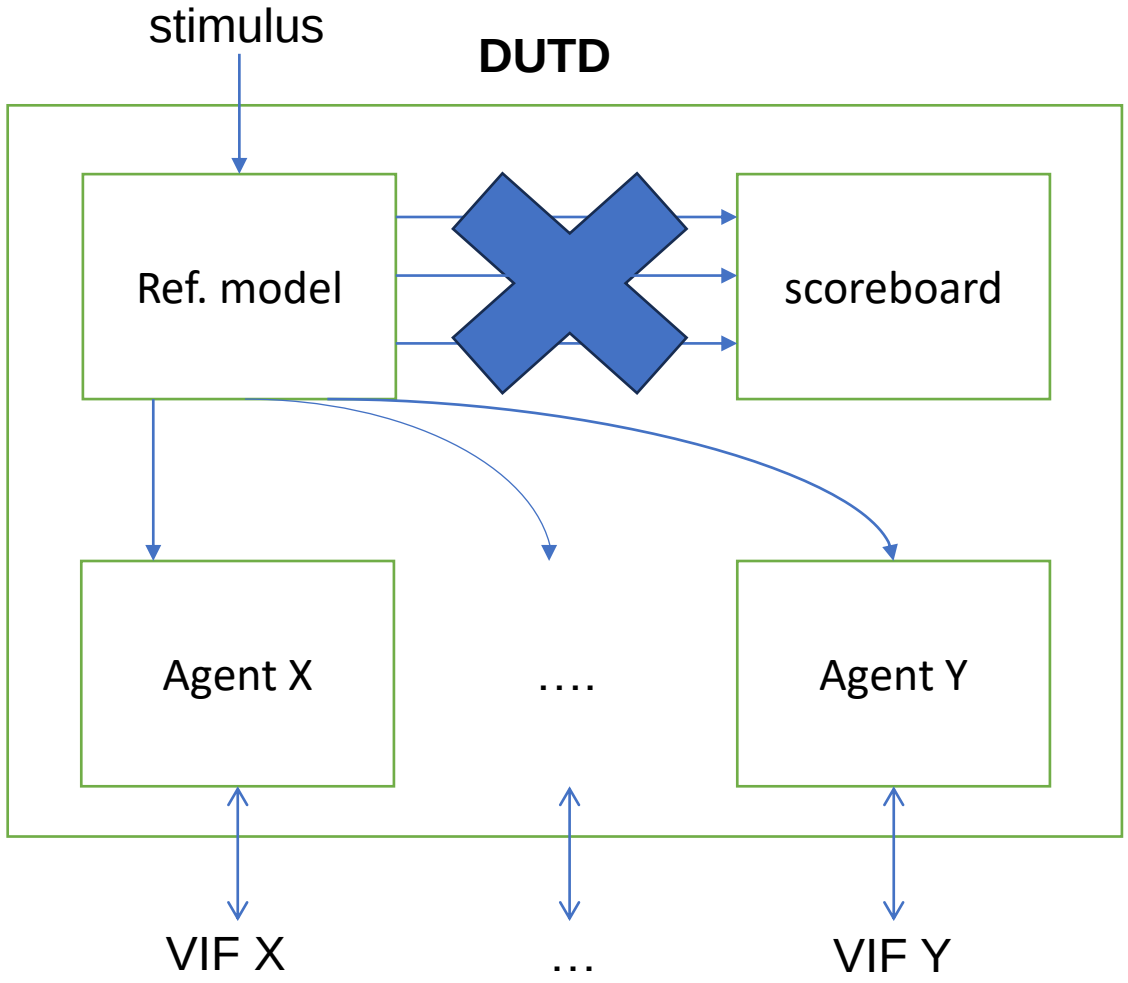
Design Under Test Double (DUTD)

- Have a test double substituting DUT to enable the complete TB – Design Under Test Double (DUTD)
- Reuse existing transaction level reference model in TB
- Convert transactions to signals
- Can we reuse more stuffs to create DUTD?



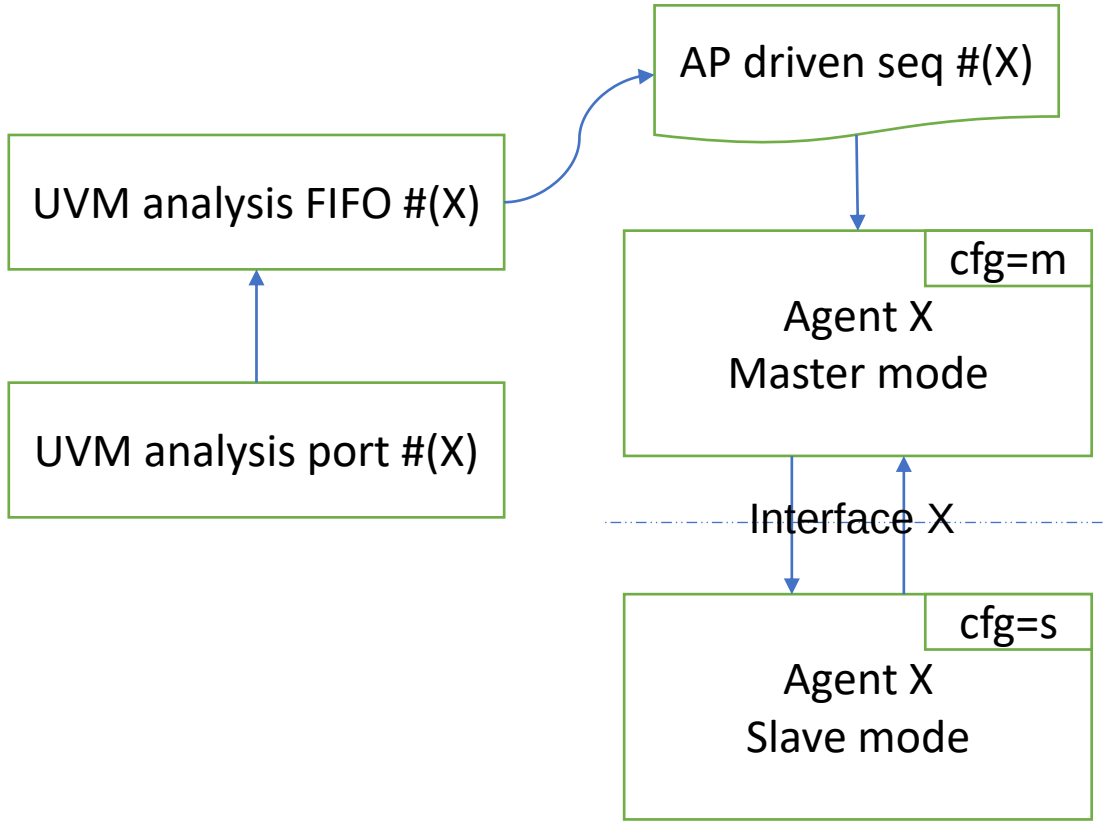
Design Under Test Double (DUTD)

- REF produces output data to scoreboard via analysis ports
- Instantiate another instance of REF but connect its analysis ports to UVM agents instead of scoreboard
- Let agent's driver convert transaction to signal
- AP-driven sequence bridges REF and agents



Invertible UVM agents and analysis-port-driven sequence

- Let every agent can be configured as master or slave mode
- A generic analysis-port-driven sequence can be applied to master agent by overrides
- The sequence listens to the analysis-port and generate master request accordingly

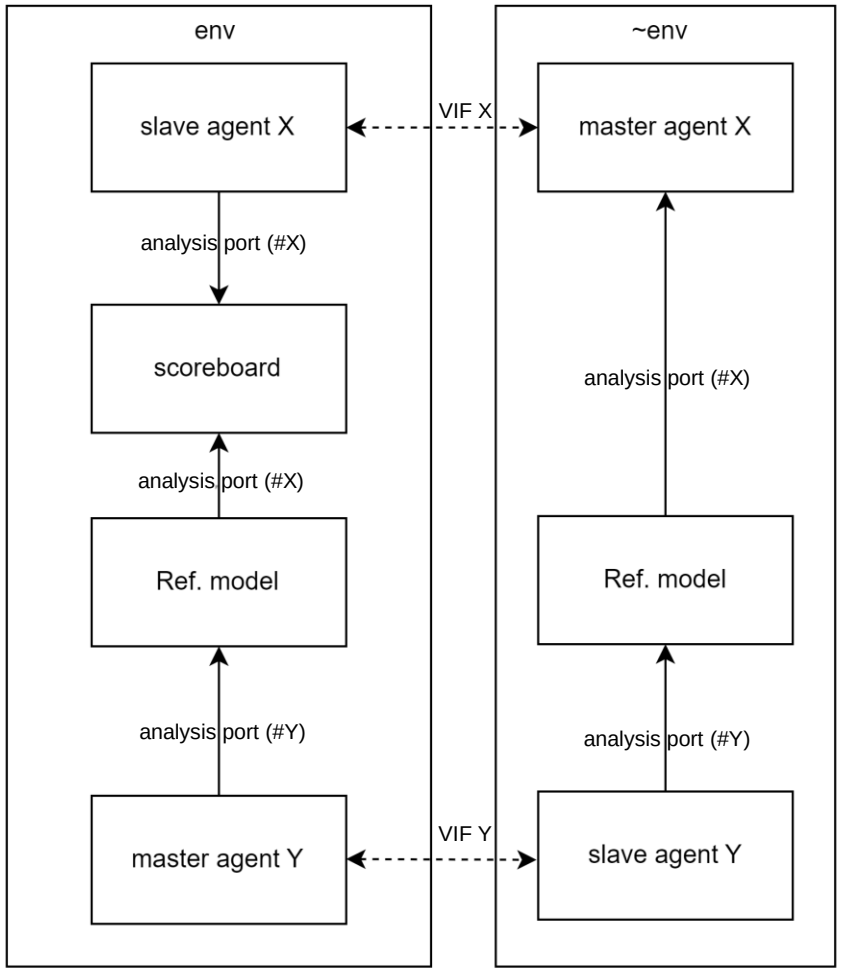


Sample Code for Invertible Agents and AP-Driven Sequence

Invertible Agents	AP-Driven Sequence
virtual function void sample_env::setup_agent_x(); … … m_agent_x_config.is_master = MASTER; Endfunction virtual function void sample_env::setup_agent_y(); … … m_agent_y_config.is_master = SLAVE; endfunction	class sample_ap_driven_sequencer #(type REQ) extends uvm_sequencer#(REQ); … … uvm_tlm_analysis_fifo #(REQ) m_fifo; … … endclass : sample_ap_driven_sequencer class sample_ap_driven_sequence #(type REQ) extends uvm_sequence #(REQ); `uvm_declare_p_sequencer(sample_sample_ap_driven_sequencer #(REQ)) … … virtual task body(); forever begin p_sequencer.m_fifo.get(req); `uvm_send(req); end endtask endclass

Environment Inversion to create DUTD

- Extend original UVM env to derive inverted env (~env)
- ~env inverts master/slave mode of for each agent in build_phase
- Re-direct ~env REF outputs to inverted agents in connect_phase
- Apply AP-driven sequences in run_phase
- Leverage overrides and macros to minimize coding



Sample Code for Environment Inversion

```
class sample_inverse_env extends sample_env;
   …
   …
   virtual function void build_phase (uvm_phase phase);
      super.build_phase(phase);
      set_inst_override_by_type("*",
uvm_sequencer#(uvm_sequence_item)::get_type(),
ap_driven_sequencer#(uvm_sequence_item)::get_type());
   endfunction
   …
   …
endclass

virtual function void sample_inverse_env::setup_agent_x();
   super.setup_agent_x();
   m_agent_y_config.is_master = SLAVE;// flip
endfunction
virtual function void sample_inverse_env::setup_agent_y();
   super.setup_agent_y();
   m_agent_y_config.is_master = MASTER;// flip
endfunction
```

```
virtual function void sample_inverse_env::connect_phase (uvm_phase
phase);
    super.connect_phase(phase);
    ap_driven_sequencer #(uvm_sequence_item) seqr;
   $cast(seqr, m_agent_y.m_sequencer)
    m_reference_model.sample_ap.connect(seqr.m_fifo.analysis_export);
Endfunction

virtual task sample_inverse_env::run_phase(uvm_phase phase);
   super.run_phase(phase);
   ap_driven_sequence#(uvm_sequence_item) m_agent_y_ap_seq;
   ap_driven_sequencer #(uvm_sequence_item) seqr;
   $cast(seqr, m_agent_y.m_sequencer)
   m_agent_y_ap_seq =
ap_driven_sequence#(uvm_sequence_item)::type_id::create("m_agent_y
_ap_seq");
   m_agent_y_ap_seq.start(seqr);
endtask
```

Sample Code for Environment Inversion

uvm_test

```
class sample_test extends uvm_test;
   sample_env        m_env;
   sample_inverse_env  m_inverse_env;

   virtual function void build_phase (uvm_phase phase);
      super.build_phase(phase);
      m_env = sample_env::type_id::create("m_env", this);
      m_vseq = sample_inverse_env
::type_id::create("m_vseq");
   endfunction
   …
   …
endclass
```

Conclusion

- Environment inversion fully decouples TB dependency on DUT's availability – left-shifting TB development significantly
- Whole stimulus-flow, UVCs completeness, checker and functional coverage can be qualified before DUT is ready
- Proposed methodology is complementary to UVM and scalable to VIPs and nested environments
- Coding for environment inversion can be minimized by macros
- Checker could be further qualified by fault injection tool applied to DUTD

Questions?

Reference

- [1] http://xunitpatterns.com/Test%20Double%20Patterns.html