Methodology focused

uvmgen.com

Intro

- Who am I?
  - Ben Delsol - DV engineer formerly at Intel, Qualcomm, Samsung and Microsoft.
  - Founder of uvmgen.com.
- What I care about?
  - Clean code.
  - Methodology best practices.
  - Not wasting brain energy.
    - Divide, reuse and conquer.
    - Automating redundant problems.

The idea of UVM is spot on

- Common procedures and methodologies across the industry.
- Clear coding and separation of testbench concerns.
- Reusable protocol agents.
- Reusable block level environments.
- Decades of verification best practices rolled into one methodology.

Some best practices from the last 15 years…

- Interface harnesses
- Abstract/concrete classes
- DUT parameter passing to VIP
- Scale UVC, sub-env, config and TLM instances at runtime
- Conditional instantiation of static verification elements at compile time
- Pass down config object over config db
- Use sequence, BFM and config factory overrides
- Use slave sequences with late response randomization
- Reset methodology: don’t kill sequences with the sequencer
- Use virtual sequences over phase jumping
- No virtual sequencers
- Use objections wisely
- Use constraint policies over inheritance
- Use standalone testbench for UVC development
- And many more…

Interface harness

- Problem:
  - Code which handles connectivity of the design to interfaces, access to BFMs and DUT parameters is not reusable from the block level to upper levels of integration.
- Solution:
  - An interface harness defines the connectivity of all interface signals to a design and can be reused/bound into the DUT at block level as well as upper levels of integration.
  - It encapsulates VIP to DUT connectivity.
  - It encapsulates access to BFM creation classes.
  - It encapsulates collection of DUT parameters for the testbench.

async_fifo

UVCs

SELECT UVC INSTANCE

CONTINUE

3 Sub-environments
Optional

Predictor

Coverage

6 Configuration and policies
Optional

7 Signal checker

```
`ifndef __ASYNC_FIFO_INTERFACE_HARNESS_SV__
`define __ASYNC_FIFO_INTERFACE_HARNESS_SV__

module async_fifo_interface_harness
 #(
   parameter DSIZE=0,
   parameter ASIZE=0,
   parameter FALLTHROUGH=0
 )(
   inout wire             wclk,
   inout wire             wrst_n,
   inout wire             winc,
   inout wire [DSIZE-1:0] wdata,
   inout wire             wfull,
   inout wire             awfull,
   inout wire             rclk,
   inout wire             rrst_n,
   inout wire             rinc,
   inout wire [DSIZE-1:0] rdata,
   inout wire             rempty,
   inout wire             arempty
 );

 timeunit 1ns / 1ns;

 import uvm_pkg::*;
 import async_fifo_env_pkg::*;

 string    env_full_name;

 clk_if
   clk_r_if (
     .clk (rclk)
   );
```

async_fifo_interface_harness.sv

DONE

async_fifo_env_pkg.sv

async_fifo

UVCs

SELECT UVC INSTANCE

fifow_master_if

Instance of fifow_if

Parameters

Name	Value
DATA_MSB	Not set (default: 31)
ON_POSEDGE	Not set (default: 1)

Ports

Name		Connection
wclk	⇦	wclk
wrst_n	⇦	wrst_n
winc	⇨	winc
wdata	⇨	wdata
wfull	⇦	wfull
wafull	⇦	awfull

CONTINUE

```
    .clk (wclk),
    .rst (wrst_n)
  );

fifor_if
  fifor_master_if (
    .rclk (rclk),
    .rrst_n (rrst_n),
    .rinc (rinc),
    .rdata (rdata),
    .rempty (rempty),
    .raempty (arempty)
  );

fifow_if
  fifow_master_if (
    .wclk (wclk),
    .wrst_n (wrst_n),
    .winc (winc),
    .wdata (wdata),
    .wfull (wfull),
    .wafull (awfull)
  );

initial begin
  wait (env_full_name != "");

  clk_r_if.add_resources_to_config_db({ env_full_name, ".clk_r_agent" })
  clk_w_if.add_resources_to_config_db({ env_full_name, ".clk_w_agent" })
  rst_master_r_if.add_resources_to_config_db({ env_full_name, ".rst_mast
  fifor_master_if.add_resources_to_config_db({ env_full_name, ".fifor_ma
end
```

async_fifo_interface_harness.sv DONE

async_fifo

UVCs

SELECT UVC INSTANCE

fifow_master_if

Instance of fifow_if

Parameters

Name	Value
DATA_MSB	Not set (default: 31)
ON_POSEDGE	Not set (default: 1)

Ports

Name		Connection
wclk	⇦	wclk
wrst_n	⇦	wrst_n
winc	⇨	winc
wdata	⇨	wdata
wfull	⇦	wfull
wafull	⇦	awfull

CONTINUE

```
module async_fifo_protocol_checker_harness
#(
  parameter DSIZE=0,
  parameter ASIZE=0,
  parameter FALLTHROUGH=0
)(
  input wire             wclk,
  input wire             wrst_n,
  input wire             winc,
  input wire [DSIZE-1:0] wdata,
  input wire             wfull,
  input wire             awfull,
  input wire             rclk,
  input wire             rrst_n,
  input wire             rinc,
  input wire [DSIZE-1:0] rdata,
  input wire             rempty,
  input wire             arempty
);

timeunit 1ns / 1ns;

string    env_full_name;

fifor_protocol_checker
  fifor_master_protocol_checker (
    .rclk (rclk),
    .rrst_n (rrst_n),
    .rinc (rinc),
    .rdata (rdata),
    .rempty (rempty),
    .raempty (arempty)
  );

fifow_protocol_checker
```

async_fifo_protocol_checker_harness.sv

DONE

async_fifo_env_pkg.sv

async_fifo_interface_harne...

async_fifo

TB basics

async_fifo
Selected environment

CONTINUE

2 Default settings
Optional

Virtual sequences and policies
Optional

Tests
Optional

5 DUT parameter packages
Optional

6 Generate

./

```
// ASYNC FIFO Bind statements

bind async_fifo async_fifo_interface_harness #(
  .DSIZE (DSIZE),
  .ASIZE (ASIZE),
  .FALLTHROUGH (FALLTHROUGH)
) async_fifo_interface_harness (.*);

`ifdef ENABLE_ASYNC_FIFO_PROTOCOL_CHECKERS
  bind async_fifo async_fifo_protocol_checker_harness #(
    .DSIZE (DSIZE),
    .ASIZE (ASIZE),
    .FALLTHROUGH (FALLTHROUGH)
  ) async_fifo_protocol_checker_harness (.*);
`endif
```

async_fifo_bind.svh

DONE

policies/

Abstract/concrete classes

- Problem:
  - Any component which uses a virtual interface handle to a parameterized interface must be parameterized, and so too must all its component ancestors (ie test, env, agents, drivers, monitors all must be parameterized!).
- Solution:
  - Define a BFM class in the parameterized interface.
  - Drivers and monitors initiate the BFMs construction with the abstract/concrete design pattern.
  - Ensure the BFM has access to the config object, has context aware UVM printing and can be overridden by the factory.
- Can be used for access of protocol checkers and signal checkers too.

apb

Bus protocol

BFMS | PROTOCOL CHECKER

- [x] Has a master driver
- [x] Has a slave responder
- [x] Has a monitor

SELECT BFM

CONTINUE

5 Sequences and policies
Optional

./

```
virtual class abstract_apb_bfm_creator_c;

  pure virtual function abstract_apb_bfm_c create_master_bfm(string name, u
  pure virtual function abstract_apb_bfm_c create_slave_bfm(string name, uv
  pure virtual function abstract_apb_bfm_c create_monitor_bfm(string name,

endclass
```

abstract_apb_bfm_creator_c.svh

DONE

abstract_apb_bfm

apb_config_c.svh

apb_monitor_c.svh

apb_master_driver_c.svh

apb_master_agent_c.svh

apb

Bus protocol

BFMS | PROTOCOL CHECKER

- [x] Has a master driver
- [x] Has a slave responder
- [x] Has a monitor

SELECT BFM

CONTINUE

5 Sequences and policies
Optional

```
class apb_bfm_creator_c extends abstract_apb_bfm_creator_c;

  function abstract_apb_bfm_c create_master_bfm(string name, uvm_component
    return apb_master_bfm_c::type_id::create(name, parent);
  endfunction

  function abstract_apb_bfm_c create_slave_bfm(string name, uvm_component p
    return apb_slave_bfm_c::type_id::create(name, parent);
  endfunction

  function abstract_apb_bfm_c create_monitor_bfm(string name, uvm_component
    return apb_monitor_bfm_c::type_id::create(name, parent);
  endfunction

endclass
```

apb_bfm_creator_c.svh DONE

apb_pkg.sv

apb_slave_bfm_c.sv

apb_if.sv

policies/

bfm_c.svh

apb

Bus protocol

BFMS | PROTOCOL CHECKER

- [x] Has a master driver
- [x] Has a slave responder
- [x] Has a monitor

SELECT BFM

CONTINUE

5 Sequences and policies
Optional

```
    input penable;
    input pwrite;
    input pwdata;
    input pstrb;
    input pready;
    input prdata;
    input pslverr;
  endclocking

  `include "apb_bfm_base_c.svh"
  `include "apb_master_bfm_c.svh"
  `include "apb_slave_bfm_c.svh"
  `include "apb_monitor_bfm_c.svh"

  `include "apb_bfm_creator_c.svh"

  function void add_resources_to_config_db(string inst_name);
    add_parameters_to_config_db({inst_name, ".cfg"});
    add_bfm_creator_to_config_db(inst_name);
  endfunction

  function automatic void add_parameters_to_config_db(string inst_name);
    apb_parameters_c params = apb_parameters_c::type_id::create("params");
    params.set_addr_msb(ADDR_MSB);
    params.set_addr_lsb(ADDR_LSB);
    params.set_data_msb(DATA_MSB);
    uvm_config_db#(apb_parameters_c)::set(null, inst_name, "params", param
  endfunction

  function automatic void add_bfm_creator_to_config_db(string inst_name);
    apb_bfm_creator_c bfm_creator = new();
    uvm_config_db#(abstract_apb_bfm_creator_c)::set(null, inst_name, "bfm_
  endfunction

endinterface
```

apb_if.sv DONE

policies/

apb

Bus protocol

BFMS | PROTOCOL CHECKER

- [x] Has a master driver
- [x] Has a slave responder
- [x] Has a monitor

SELECT BFM

CONTINUE

5 Sequences and policies
Optional

```
class apb_master_driver_c extends uvm_driver#(apb_sequence_item_c);

  apb_config_c                  cfg;
  abstract_apb_bfm_creator_c    bfm_creator;
  abstract_apb_bfm_c            bfm;

  `uvm_component_utils(apb_master_driver_c)

  function new(string name="apb_master_driver_c", uvm_component parent=nul
    super.new(name, parent);
  endfunction

  virtual function void set_cfg(apb_config_c cfg);
    this.cfg = cfg;
  endfunction

  virtual function void set_bfm_creator(abstract_apb_bfm_creator_c bfm_cre
    this.bfm_creator = bfm_creator;
  endfunction

  virtual function void build_phase(uvm_phase phase);
    if (cfg == null) begin
      `uvm_fatal(get_type_name(), "Must call set_cfg(apb_config_c cfg).")
    end

    if (bfm_creator == null) begin
      `uvm_fatal(get_type_name(), "Must call set_bfm_creator(abstract_apb_
    end

    bfm = bfm_creator.create_master_bfm("bfm", this);
    bfm.set_cfg(cfg);
  endfunction

  virtual task run_phase(uvm_phase phase);
```

apb_master_driver_c.svh

DONE

DUT parameter passing to VIP

- Problem:
  - The verification environment of a parameterized design must also have access to those parameters. Again, type specialization of many classes becomes very cumbersome to manage.
- Solution:
  - Use your interfaces and interface harness' to collect interface and DUT parameters, respectively.
  - Pass these parameter objects to your UVC and environment config objects via the config db.

apb

Interface signals

Interface parameters

Type	Name	Default
parameter	ADDR_MSB	31
parameter	ADDR_LSB	0
parameter	DATA_MSB	31
parameter bit	ON_POSEDGE	1

ADD

Master and slave nets

Source	Type	Name	Reset value
C	wire	pclk	'x
R	wire	preset_n	'x
M	wire	psel	'x
M	wire [ADDR_MS...	paddr	'x
M	wire	penable	'x
M	wire	pwrite	'x
M	wire [DATA_MSB...	pwdata	'x
M	wire [((DATA_MS...	pstrb	'x
S	wire	pready	'x
S	wire [DATA_MSB...	prdata	'x
S	wire	pslverr	'x

./

```
class apb_parameters_c extends uvm_object;

  logic [31:0]                addr_msb;
  logic [31:0]                addr_lsb;
  logic [31:0]                data_msb;
  bit [APB_ADDR_MSB_MAX:0]    addr_mask;
  bit [APB_DATA_MSB_MAX:0]    data_mask;

  `uvm_object_utils_begin(apb_parameters_c)
    `uvm_field_int(addr_msb, UVM_DEFAULT)
    `uvm_field_int(addr_lsb, UVM_DEFAULT)
    `uvm_field_int(data_msb, UVM_DEFAULT)
    `uvm_field_int(addr_mask, UVM_DEFAULT | UVM_NOPRINT)
    `uvm_field_int(data_mask, UVM_DEFAULT | UVM_NOPRINT)
  `uvm_object_utils_end

  function new(string name="apb_parameters_c");
    super.new(name);
  endfunction

  virtual function void set_addr_msb(logic [31:0] addr_msb);
    this.addr_msb = addr_msb;
    update_addr_mask();
  endfunction

  virtual function void set_addr_lsb(logic [31:0] addr_lsb);
    this.addr_lsb = addr_lsb;
    update_addr_mask();
  endfunction

  virtual function void set_data_msb(logic [31:0] data_msb);
    this.data_msb = data_msb;
    update_data_mask();
  endfunction
```

apb_parameters_c.svh DONE

abstract_apb_bfm

apb_config_c.svh

apb_monitor_c.svh

apb_master_driver_c.svh

apb_master_agent_c.svh

apb

Interface signals

Interface parameters

Type	Name	Default
parameter	ADDR_MSB	31
parameter	ADDR_LSB	0
parameter	DATA_MSB	31
parameter bit	ON_POSEDGE	1

ADD

Master and slave nets

Source	Type	Name	Reset value
C	wire	pclk	'x
R	wire	preset_n	'x
M	wire	psel	'x
M	wire [ADDR_MS...	paddr	'x
M	wire	penable	'x
M	wire	pwrite	'x
M	wire [DATA_MSB...	pwdata	'x
M	wire [((DATA_MS...	pstrb	'x
S	wire	pready	'x
S	wire [DATA_MSB...	prdata	'x
S	wire	pslverr	'x

```
    input penable;
    input pwrite;
    input pwdata;
    input pstrb;
    input pready;
    input prdata;
    input pslverr;
  endclocking

  `include "apb_bfm_base_c.svh"
  `include "apb_master_bfm_c.svh"
  `include "apb_slave_bfm_c.svh"
  `include "apb_monitor_bfm_c.svh"

  `include "apb_bfm_creator_c.svh"

  function void add_resources_to_config_db(string inst_name);
    add_parameters_to_config_db({inst_name, ".cfg"});
    add_bfm_creator_to_config_db(inst_name);
  endfunction

  function automatic void add_parameters_to_config_db(string inst_name);
    apb_parameters_c params = apb_parameters_c::type_id::create("params");
    params.set_addr_msb(ADDR_MSB);
    params.set_addr_lsb(ADDR_LSB);
    params.set_data_msb(DATA_MSB);
    uvm_config_db#(apb_parameters_c)::set(null, inst_name, "params", param
  endfunction

  function automatic void add_bfm_creator_to_config_db(string inst_name);
    apb_bfm_creator_c bfm_creator = new();
    uvm_config_db#(abstract_apb_bfm_creator_c)::set(null, inst_name, "bfm_
  endfunction

endinterface
```

apb_if.sv

DONE

apb

6 Configuration
Optional

apb_config_c
Extends uvm_object

Instance variables

Type	Name
No instance variables	

ADD

Subroutines

ADD

CONTINUE

Test
Optional

8 Generate

```
virtual function void set_interface_bound(bit interface_bound);
  `ifndef DISABLE_INTERFACE_HARNESS_NOT_BOUND_WARNINGS
    if (interface_bound == 0) begin
      `uvm_warning(get_type_name(), "Without an interface the following
    end
  `endif
  this.interface_bound = interface_bound;
endfunction

virtual function bit get_interface_bound();
  return interface_bound;
endfunction

function void pre_randomize();
  super.pre_randomize();

  if (inst_name == "") begin
    `uvm_fatal(get_type_name(), "Must call set_inst_name(string inst_nam
  end
  if (!uvm_config_db#(apb_parameters_c)::get(null, inst_name, "params",
    `uvm_fatal(get_type_name(), {"apb_parameters_c must be set for ", in
  end
endfunction

virtual function bit has_driver();
  return (interface_bound && (active_state == UVM_ACTIVE));
endfunction

virtual function bit has_monitor();
  return interface_bound;
endfunction

endclass
```

apb_config_c.svh DONE

Use constraint policies over hard coded constraints

- Problem:
  - Hard coding constraints directly into subclasses runs into scenarios where the constraint code must be duplicated. How to reuse constraints?
- Solution:
  - Write your constraint(s) once in a policy object.
  - Apply policy objects on sequence items, sequences or config objects as needed.
  - Use factory overrides to instantiate the objects which apply these policies.

apb

5 Sequences and policies
Optional

SEQUENCES | POLICIES

SELECT POLICY

apb_slave_response_policy_c
Extends policy_base_c#(apb_sequence_item_c)

Constraints

Add constraint...

apb_sequence_item_c rand instance variables

CONTINUE

6 Configuration
Optional

7 Test
Optional

8 Generate

apb_if.sv

policies/

```
class apb_slave_response_policy_c extends policy_base_c#(apb_sequence_item

  protected apb_sequence_item_c    request_item;

  constraint phase_crv {
    object.phase == APB_PHASE_RESPONSE;
  }

  constraint data_crv {
    if (object.write) {
      object.data == request_item.data;
    }
  }

  constraint sel_crv {
    object.sel == request_item.sel;
  }

  constraint addr_crv {
    object.addr == request_item.addr;
  }

  constraint write_crv {
    object.write == request_item.write;
  }

  constraint strb_crv {
    object.strb == request_item.strb;
  }

  function new(string name="apb_slave_response_policy_c");
    super.new(name);
  endfunction
```

apb_slave_response_policy_c.svh DONE

apb_master_policy

quest_policy_c...

seqs/

```
function void pre_ran
  super.pre_randomize

  if (cfg == null) begin
    `uvm_fatal(get_type_name(), "Mu
  end
endfunction
```

```
`uvm_object_utils(apb_master_sequen

function new(string name="apb_maste
  super.new(name);
endfunction
```

```
request_sequence_c ex
apb_config_c    cfg;

`uvm_object_utils_begin(apb_slave_r
  `uvm_field_object(cfg, UVM_DEFAUL
`uvm_object_utils_end
```

apb

5 Sequences and policies
Optional

SEQUENCES POLICIES

SELECT SEQUENCE

apb_slave_response_sequence_base_c
Extends uvm_sequence#(apb_sequence_item_c)

Apply policies

Constrains apb_sequence_item_c req (apb_slave_resp...

Select or create a new policy

CONTINUE

6 Configuration
Optional

7 Test
Optional

8 Generate

```
    super.pre_randomize();

    if (cfg == null) begin
      `uvm_fatal(get_type_name(), "Must call set_cfg(apb_config_c cfg) bef
    end

    if (request_item == null) begin
      `uvm_fatal(get_type_name(), "Must call set_request_item(apb_sequence
    end
  endfunction

  virtual task body();
    req = apb_sequence_item_c::type_id::create("apb_sequence_item");
    req.set_cfg(cfg);

    start_item(req);
    do_req(req);
    if (!req.randomize()) begin
      `uvm_error(get_type_name(), { req.get_type_name(), " randomization f
    end

    finish_item(req);

    get_response(rsp);
  endtask

  virtual function void do_req(apb_sequence_item_c req);
    apb_slave_response_policy_c apb_slave_response_policy = new("apb_slav

    apb_slave_response_policy.set_request_item(request_item);
    req.add_policy(apb_slave_response_policy);
  endfunction

endclass
```

apb_slave_response_sequence_base_c.svh DONE

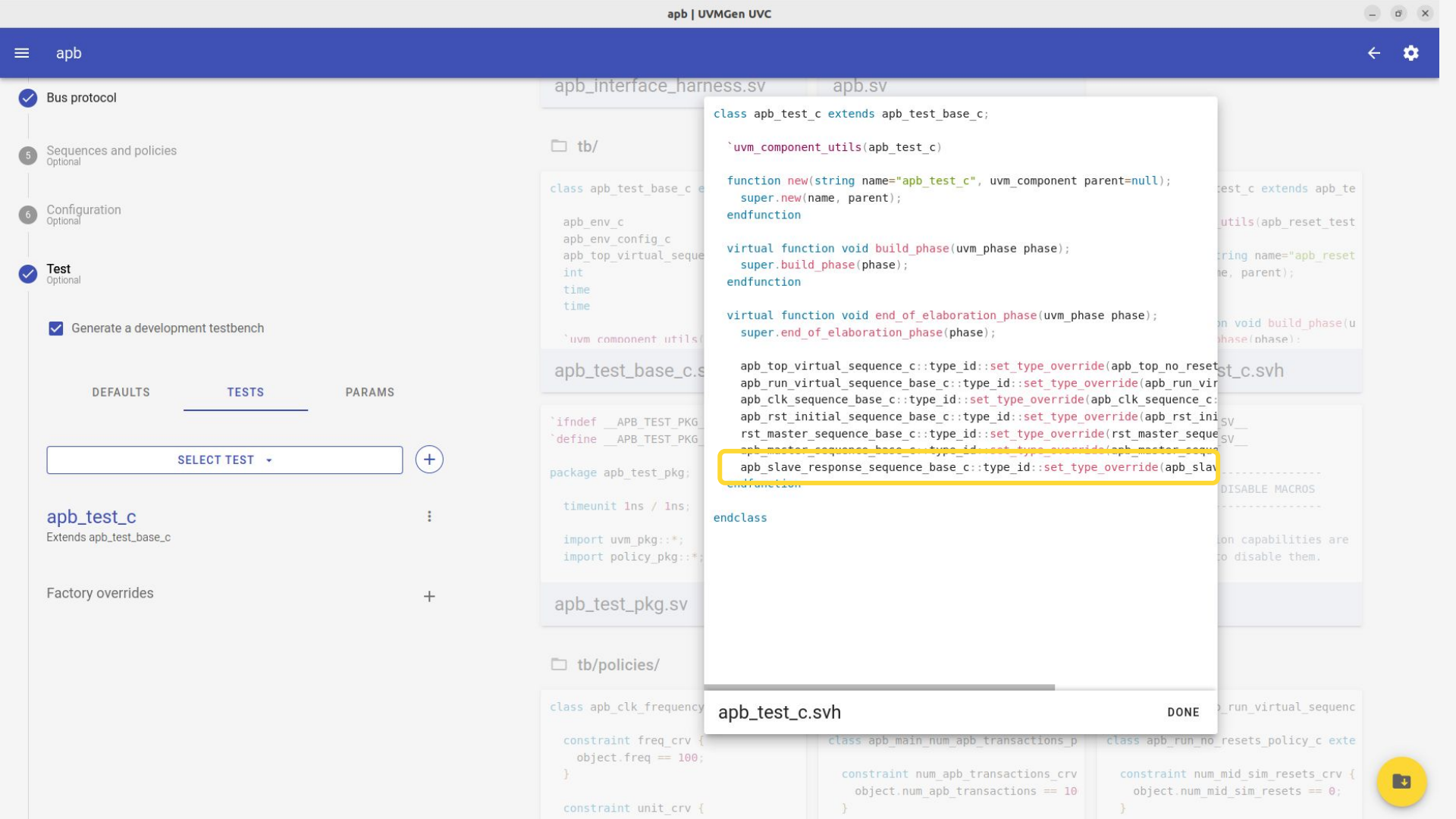
seqs/

apb_slave_respon...

apb_master_sequ...

apb_slave_response_seque...

apb_slave_sequence_c.svh


apb | UVMGen UVC
apb
Bus protocol
5
Sequences and policies
Optional
6
Configuration
Optional
Test
Optional
Generate a development testbench
DEFAULTS
TESTS
PARAMS
SELECT TEST
apb_test_c
Extends apb_test_base_c
Factory overrides
apb_interface_harness.sv
apb.sv
tb/
apb_test_base_c.s
apb_test_pkg.sv
tb/policies/
class apb_test_c extends apb_test_base_c;
`uvm_component_utils(apb_test_c)
function new(string name="apb_test_c", uvm_component parent=null);
super.new(name, parent);
endfunction
virtual function void build_phase(uvm_phase phase);
super.build_phase(phase);
endfunction
virtual function void end_of_elaboration_phase(uvm_phase phase);
super.end_of_elaboration_phase(phase);
apb_top_virtual_sequence_c::type_id::set_type_override(apb_top_no_reset
apb_run_virtual_sequence_base_c::type_id::set_type_override(apb_run_vir
apb_clk_sequence_base_c::type_id::set_type_override(apb_clk_sequence_c:
apb_rst_initial_sequence_base_c::type_id::set_type_override(apb_rst_ini
rst_master_sequence_base_c::type_id::set_type_override(rst_master_seque
apb_slave_response_sequence_base_c::type_id::set_type_override(apb_slav
endclass
apb_test_c.svh
DONE

Scale testbench components/objects at runtime

- Problem:
  - Compile-time instance scaling requires class type specializations. And as we know, that's no fun.
- Solution:
  - Collect DUT parameters at runtime and make them available to env and agent configs.
  - Using these parameter variables and dynamic arrays to:
    - Construct agents and sub-env instances.
    - Construct env-configs and agent configs.
    - Construct/connect scoreboard, predictor and coverage TLM.
    - Construct sub-env predictors.

hub

axir_master_if

Instance of axir_if

Parameters

Name	Value
ARID_MSB	Not set (default: 3)
ARADDR_MSB	ARADDR_MSB
ARADDR_LSB	ARADDR_LSB
ARLEN_MSB	ARLEN_MSB
RID_MSB	RID_MSB
RDATA_MSB	RDATA_MSB
RRESP_MSB	RRESP_MSB
ON_POSEDGE	Not set (default: 1)

Ports

Name		Connection
clk	⇦	clk
reset_n	⇦	reset_n
arid	⇨	arid_slvx[d]
araddr	⇨	araddr_slvx[d]
arlen	⇨	arlen_slvx[d]

```
    for (genvar d1 = 4; d1 < (NUM_Y2 - 2); d1++) begin : GEN_IF_RESOURCES1
      initial begin
        wait (env_full_name != "");

        GEN_IF1_D0[d0].GEN_IF1_D1[d1].sideband_master_y_if.add_resources_t
        GEN_IF1_D0[d0].GEN_IF1_D1[d1].sideband_slave_y_if.add_resources_to
        GEN_IF1_D0[d0].GEN_IF1_D1[d1].apb_master_y_if.add_resources_to_con
        GEN_IF1_D0[d0].GEN_IF1_D1[d1].apb_slave_y_if.add_resources_to_conf
        GEN_IF1_D0[d0].GEN_IF1_D1[d1].axir_master_y_if.add_resources_to_co
        GEN_IF1_D0[d0].GEN_IF1_D1[d1].axir_slave_y_if.add_resources_to_con
        GEN_IF1_D0[d0].GEN_IF1_D1[d1].axiw_master_y_if.add_resources_to_co
        GEN_IF1_D0[d0].GEN_IF1_D1[d1].axiw_slave_y_if.add_resources_to_con
      end
    end
  end

  function void add_resources_to_config_db(string env_full_name);
    hub_interface_harness.env_full_name = env_full_name;
    add_parameters_to_config_db(env_full_name);
  endfunction

  function automatic void add_parameters_to_config_db(string inst_name);
    hub_env_parameters_c params = hub_env_parameters_c::type_id::create("p
    params.set_data_width(DATA_WIDTH);
    params.set_num_x(NUM_X);
    params.set_num_y2(NUM_Y2);
    params.set_insert_bug(INSERT_BUG);
    uvm_config_db#(hub_env_parameters_c)::set(null, inst_name, "env_params
  endfunction

endmodule
```

hub_interface_harness.sv DONE

hub

axir_master_if

Instance of axir_if

Parameters

Name	Value
ARID_MSB	Not set (default: 3)
ARADDR_MSB	ARADDR_MSB
ARADDR_LSB	ARADDR_LSB
ARLEN_MSB	ARLEN_MSB
RID_MSB	RID_MSB
RDATA_MSB	RDATA_MSB
RRESP_MSB	RRESP_MSB
ON_POSEDGE	Not set (default: 1)

Ports

Name		Connection
clk	⇦	clk
reset_n	⇦	reset_n
arid	⇨	arid_slvx[d]
araddr	⇨	araddr_slvx[d]
arlen	⇨	arlen_slvx[d]

hub_predictor_c.sv

hub_env_pkg.sv

hub_signal_checker.sv

```
  env_cfg.set_signal_checker_bound(bound);
end

if (env_cfg.has_agents()) begin
  clk_agent = clk_agent_c::type_id::create("clk_agent", this);
  clk_agent.set_cfg(env_cfg.clk_cfg);

  rst_master_agent = rst_master_agent_c::type_id::create("rst_master_a
  rst_master_agent.set_cfg(env_cfg.rst_master_cfg);

  sideband_master_agent = sideband_master_agent_c::type_id::create("si
  sideband_master_agent.set_cfg(env_cfg.sideband_master_cfg);

  apb_slave_agent = apb_slave_agent_c::type_id::create("apb_slave_agen
  apb_slave_agent.set_cfg(env_cfg.apb_slave_cfg);

  axir_master_agents = new[env_cfg.params.num_x];
  foreach (axir_master_agents[a]) begin
    axir_master_agents[a] = axir_master_agent_c::type_id::create($sfor
    axir_master_agents[a].set_cfg(env_cfg.axir_master_cfgs[a]);
  end

  axir_slave_agents = new[env_cfg.params.num_x];
  foreach (axir_slave_agents[a]) begin
    axir_slave_agents[a] = axir_slave_agent_c::type_id::create($sforma
    axir_slave_agents[a].set_cfg(env_cfg.axir_slave_cfgs[a]);
  end

  axiw_master_agents = new[env_cfg.params.num_x];
  foreach (axiw_master_agents[a]) begin
    axiw_master_agents[a] = axiw_master_agent_c::type_id::create($sfor
    axiw_master_agents[a].set_cfg(env_cfg.axiw_master_cfgs[a]);
  end
```

hub_env_c.svh DONE

hub

Sub-environments
Optional

SELECT SUB-ENV

relay_x_envs[]
Instance of relay_env_c

HDL path
GEN_X[NUM_X].relay_x

CONTINUE

Predictor

5 Coverage

6 Configuration and policies
Optional

Signal checker
Optional

```
    end

    axiw_slave_y_agents = new[(env_cfg.params.num_y1 - 1) - 3];
    foreach (axiw_slave_y_agents[a0]) begin
      axiw_slave_y_agents[a0] = new[(env_cfg.params.num_y2 - 2) - 4];
      for (int a1 = 0; a1 < axiw_slave_y_agents[a0].size(); a1++) begin
        axiw_slave_y_agents[a0][a1] = axiw_slave_agent_c::type_id::creat
        axiw_slave_y_agents[a0][a1].set_cfg(env_cfg.axiw_slave_y_cfgs[a0
      end
    end
  end

  if (env_cfg.relay_env_cfg.has_env()) begin
    relay_env = relay_env_c::type_id::create("relay_env", this);
    relay_env.set_env_cfg(env_cfg.relay_env_cfg);
  end

  relay_x_envs = new[env_cfg.params.num_x];
  foreach (relay_x_envs[e]) begin
    if (env_cfg.relay_x_env_cfgs[e].has_env()) begin
      relay_x_envs[e] = relay_env_c::type_id::create($sformatf("relay_x_
      relay_x_envs[e].set_env_cfg(env_cfg.relay_x_env_cfgs[e]);
    end
  end

  relay_y_envs = new[(env_cfg.params.num_y1 - 1) - 3];
  foreach (relay_y_envs[e0]) begin
    relay_y_envs[e0] = new[(env_cfg.params.num_y2 - 2) - 4];
    for (int e1 = 0; e1 < relay_y_envs[e0].size(); e1++) begin
      if (env_cfg.relay_y_env_cfgs[e0][e1].has_env()) begin
        relay_y_envs[e0][e1] = relay_env_c::type_id::create($sformatf("r
        relay_y_envs[e0][e1].set_env_cfg(env_cfg.relay_y_env_cfgs[e0][e1
      end
    end
```

hub_env_c.svh

DONE

hub

Predictor

hub_predictor_c

Extends uvm_component

Subroutines

- function void write_actual_request_from_rst_master(rst_sequen...
- function void write_actual_request_from_sideband_master(side...
- function void write_actual_response_from_apb_slave(apb_sequ...
- function void write_actual_request_from_axir_master_x(axir_seq...
- function void write_actual_response_from_axir_slave_x(axir_seq...
- function void write_actual_request_from_axiw_master_x(axiw_s...
- function void write_actual_data_from_axiw_master_x(axiw_sequ...
- function void write_actual_response_from_axiw_slave_x(axiw_s...
- function void write_actual_request_from_sideband_master_y(si...
- function void write_actual_response_from_sideband_slave_y(sid...
- function void write_actual_request_from_apb_master_y(apb_se...
- function void write_actual_response_from_apb_slave_y(apb_seq...
- function void write_actual_request_from_axir_master_y(axir_seq...
- function void write_actual_response_from_axir_slave_y(axir_seq...
- function void write_actual_request_from_axiw_master_y(axiw_s...

```
virtual function void set_env_cfg(hub_env_config_c env_cfg);
  this.env_cfg = env_cfg;
endfunction

virtual function void build_phase(uvm_phase phase);
  if (env_cfg == null) begin
    `uvm_fatal(get_type_name(), "Must call set_env_cfg(hub_env_config_c
  end

  actual_request_from_rst_master_imp = new("actual_request_from_rst_mast

  rst_master_export = new("rst_master_export", this);
  sideband_master_export = new("sideband_master_export", this);
  apb_slave_export = new("apb_slave_export", this);

  axir_master_exports = new[env_cfg.params.num_x];
  foreach (axir_master_exports[e]) begin
    axir_master_exports[e] = new($sformatf("axir_master_exports[%0d]", e
  end

  axir_slave_exports = new[env_cfg.params.num_x];
  foreach (axir_slave_exports[e]) begin
    axir_slave_exports[e] = new($sformatf("axir_slave_exports[%0d]", e),
  end

  axiw_master_exports = new[env_cfg.params.num_x];
  foreach (axiw_master_exports[e]) begin
    axiw_master_exports[e] = new($sformatf("axiw_master_exports[%0d]", e
  end

  axiw_slave_exports = new[env_cfg.params.num_x];
  foreach (axiw_slave_exports[e]) begin
    axiw_slave_exports[e] = new($sformatf("axiw_slave_exports[%0d]", e),
  end
```

hub_scoreboard_c.svh DONE

Conditional instantiation of static verification elements at compile time

- Problem:
  - Sub-environments and SVA may not be needed in every test regression and can bog down full-chip simulation performance.
- Solution:
  - Make instantiation of static verification sub-elements, such as interfaces, protocol checkers and signal checkers, conditional at compile time.
  - Create clear, easy to use macro definitions to disable binding of verification elements individually or all at once.
  - Enable verification sub-environments and/or SVA as needed for debug.

hub

2 **Default settings**
Optional

Clock

Frequency: 100
Unit: MHZ

Duty cycle: 0.5

Reset

Number of resets per test: 5

Duration of reset pulse (num clocks): 1 – 25

Transactions

Duration of activity between resets (num clocks): 1 – 1000

Max number of sideband transactions per reset: 100

Max number of axir transactions per reset

```
    .BRESP_MSB (BRESP_MSB)
  ) hub_signal_checker (.*);
`endif


// RELAY Bind statements

`ifdef ENABLE_RELAY_FUNCTIONAL_VERIFICATION
  bind relay: hub.relay1 relay_interface_harness #(
    .ADDRESS_WIDTH (ADDRESS_WIDTH),
    .DATA_WIDTH (DATA_WIDTH),
    .INSERT_BUG (INSERT_BUG),
    .ARID_FOO_MSB (ARID_FOO_MSB),
    .AWID_MSB (AWID_MSB),
    .SHOULD_NOT_BE_IGNORED (SHOULD_NOT_BE_IGNORED),
    .ARADDR_MSB (ARADDR_MSB),
    .ARADDR_LSB (ARADDR_LSB),
    .ARLEN_MSB (ARLEN_MSB),
    .RID_MSB (RID_MSB),
    .RDATA_MSB (RDATA_MSB),
    .RRESP_MSB (RRESP_MSB),
    .AWADDR_MSB (AWADDR_MSB),
    .AWADDR_LSB (AWADDR_LSB),
    .AWLEN_MSB (AWLEN_MSB),
    .WID_MSB (WID_MSB),
    .WDATA_MSB (WDATA_MSB),
    .BID_MSB (BID_MSB),
    .BRESP_MSB (BRESP_MSB)
  ) relay_interface_harness (.*);
`endif

`ifdef ENABLE_RELAY_PROTOCOL_CHECKERS
  bind relay: hub.relay1 relay_protocol_checker_harness #(
    .ADDRESS_WIDTH (ADDRESS_WIDTH),
    .DATA_WIDTH (DATA_WIDTH),
```

hub_bind.svh — DONE

Does your company use all these best practices?

- Interface harnesses
- Abstract/concrete classes
- DUT parameter passing to VIP
- Scale UVC, sub-env, config and TLM instances at runtime
- Conditional instantiation of static verification elements at compile time
- Pass down config object over config db
- Use sequence, BFM and config factory overrides
- Use slave sequences with late response randomization
- Reset methodology: don't kill sequences with the sequencer
- Use virtual sequences over phase jumping
- No virtual sequencers
- Use objections wisely
- Use constraint policies over inheritance
- Use standalone testbench for UVC development
- And many more…

NOPE!

The UVM dream is not today’s reality

- Tight schedules. Large designs. Lots of new features.
- Some UVM best practices are not known to engineers.
- Some UVM best practices are daunting to implement.
- Hard for a large organization to be in sync on methodology.
- Reuse per the UVM dream is not easy and takes a lot of code!

Today’s reality

- ~~Careful implementation of best practices for reuse and scalability~~.
- Make it work! Get it verified. On time. However possible.
- Monolithic verification decisions made to hit deadlines.
    - It makes sense. Reuse is not quick or easy to implement.
- Hacks to fix hacks.
- Code rot ensues…

Introducing UVMGen Technology

- With the best DV practices across the industry distilled and encoded into the UVMGen code generator, users can stand on the shoulders of DV experts.
- Generate world-class VIP in an instant, reuse at a click, integrate and scale with ease.
- Now everybody can code like a guru and capitalize on the UVM promise.

- Verify more features.
- In less time.
- With higher confidence.
- And less brain juice :p

Go to uvmgen.com and Generate your UVCs for free!

uvmgen.com