



PARADIGM[®]
WORKS

Migrating Existing AVM and URM Testbenches to OVM

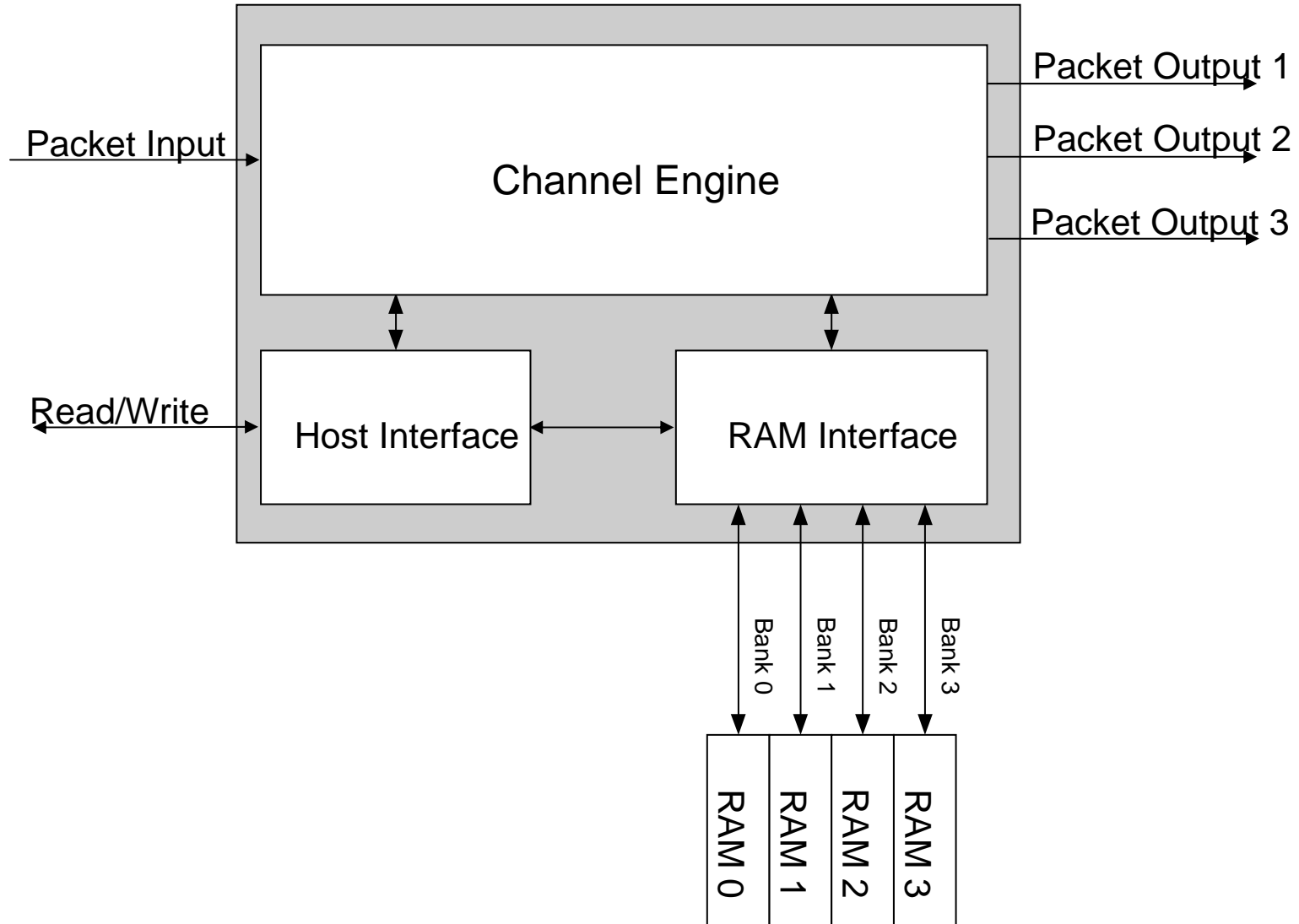
Stephen D'Onofrio
Ning Guo

Outline

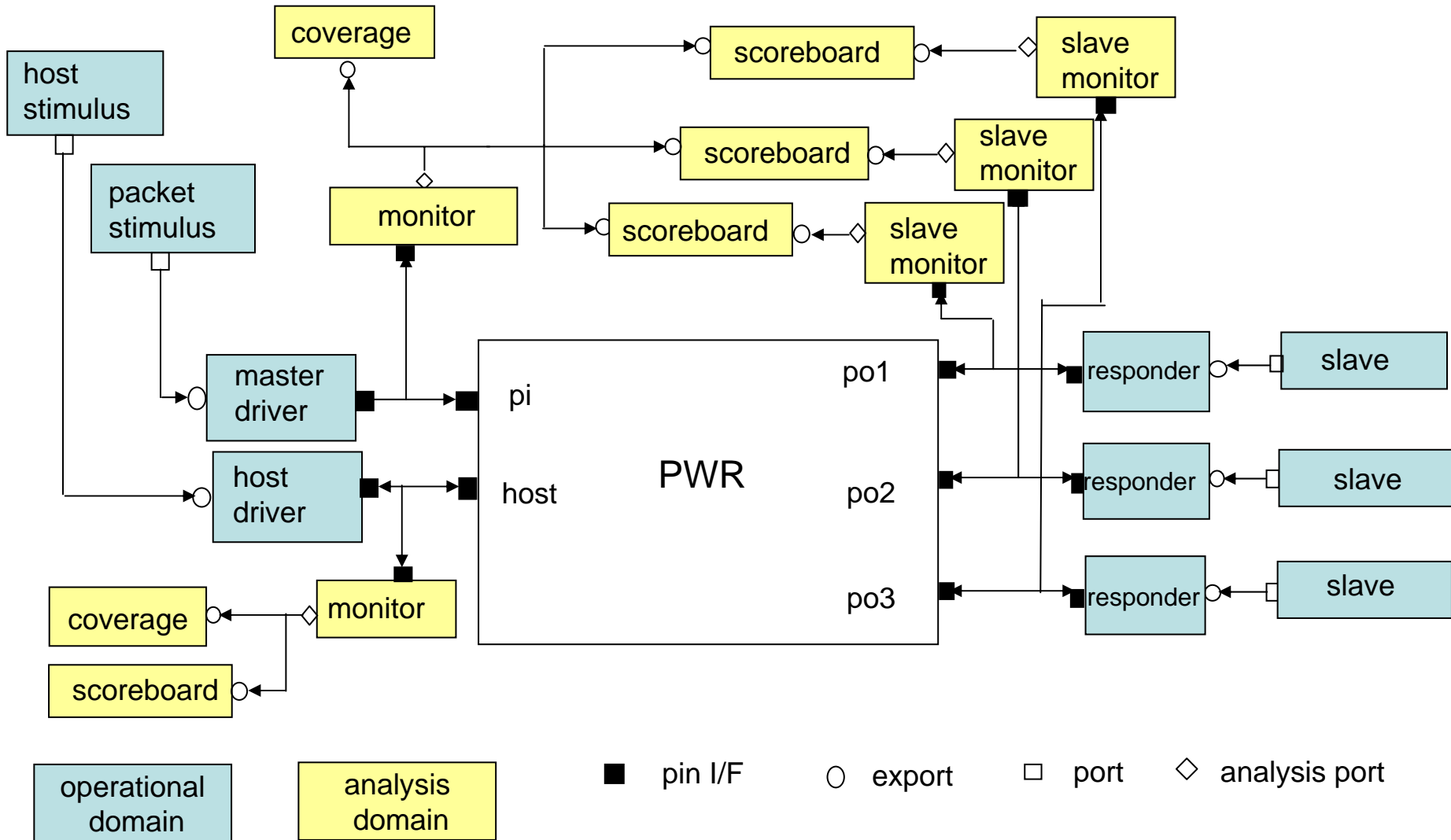
- Testbench Overview
 - AVM & URM
 - Compare AVM & URM
 - OVM testbench overview
- OVM Methodologies
 - Testbench configuration
 - Virtual Sequence
 - Error Test
- Conclusion

Design Under Test

“PWR Router”



AVM PWR Router Testbench



URM PWR Testbench

test cases ...

pw_router_sve

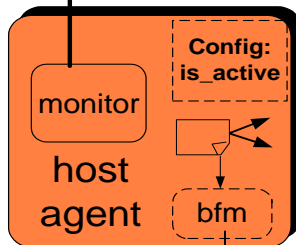
Config:

interrupt
scoreboard

packet
scoreboard

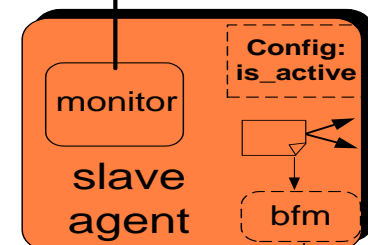
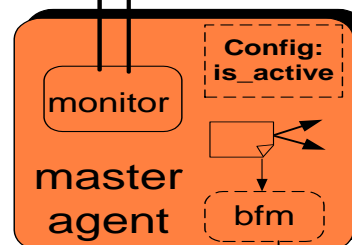
**host
env**

Config:



**pi
env**

Config:
masters=1
slaves=3



PW_ROUTER (DUT)



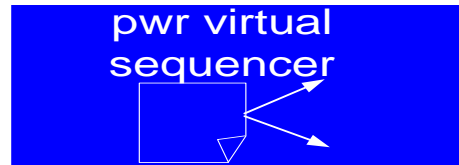
AVM/URM Features

	AVM	URM	OVM
Testbench	Layered - Test Controller - Operational Domain - Analysis Domain	Layered -Test cases -SVE -uVCs	Support both
Components	Flat/ TLM Channels	Layered/ Structured / Configurable	Support both
Stimulus	avm_random_stimulus	Sequences	Support both
Configuration/ Factory	No AVM library features	Configuration/ factory	Configuration/ factory

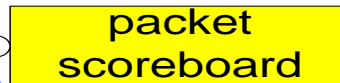
OVM PWR Testbench

test cases ...

pw_router_sve



pw_router env

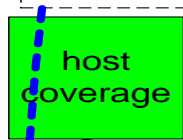


Config:

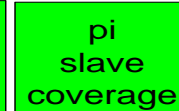


host env

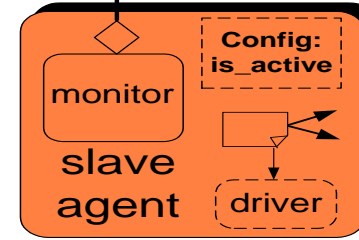
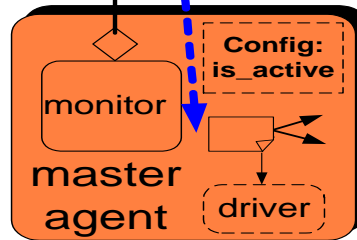
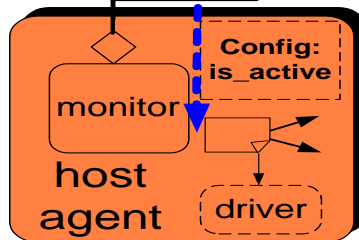
Config:



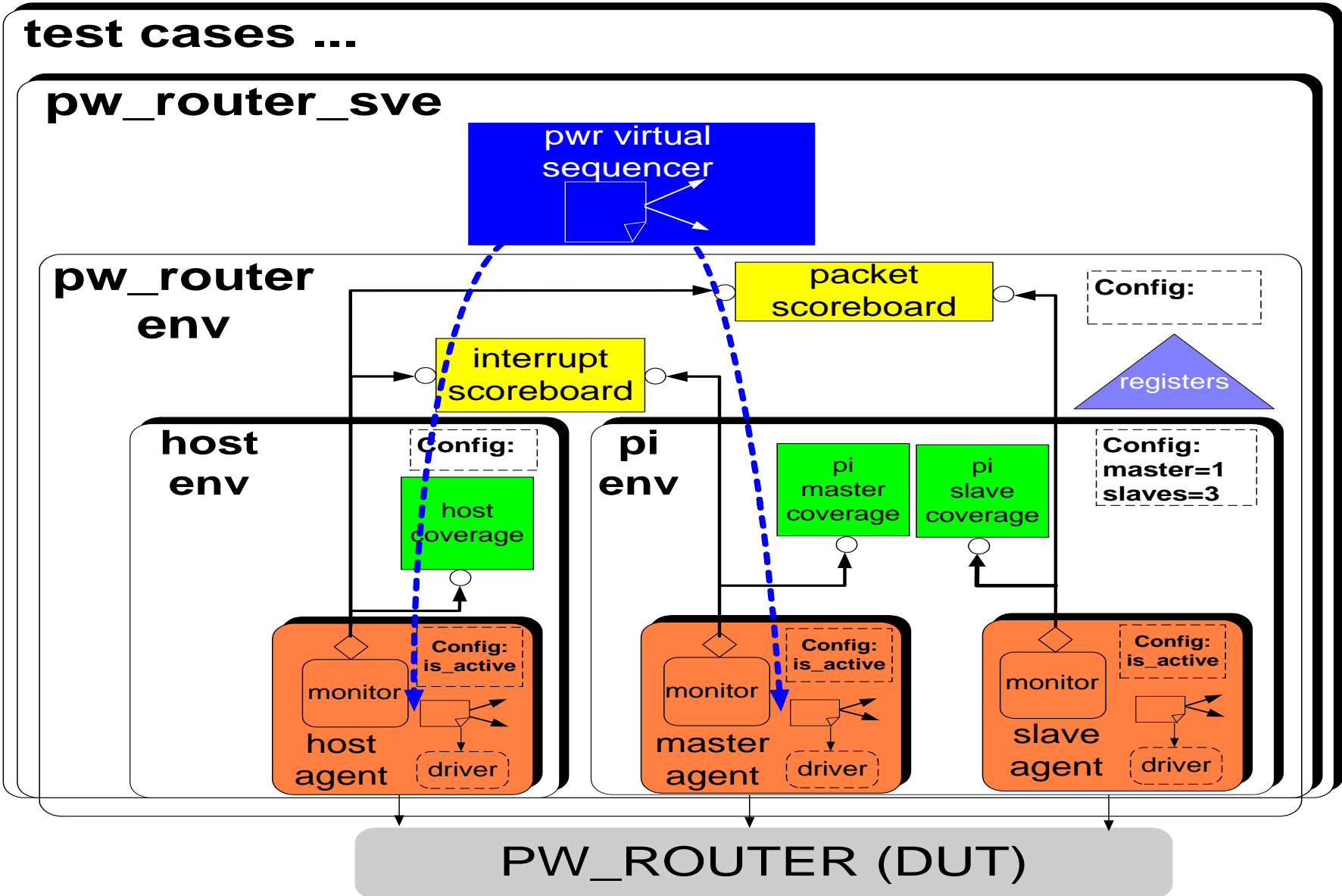
pi env



Config:
master=1
slaves=3



PW_ROUTER (DUT)



Testbench Configuration

What needs to be configurable?

// Topology Parameter: Example 1

```
class pi_env extends ovm_env;
```

```
protected int unsigned num_masters; // number of master agent
```

```
protected int unsigned num_slaves; // number of slave agent
```

// Topology Control : Example 2

```
class pi_agent extends ovm_agent;
```

```
protected bit is_active;
```

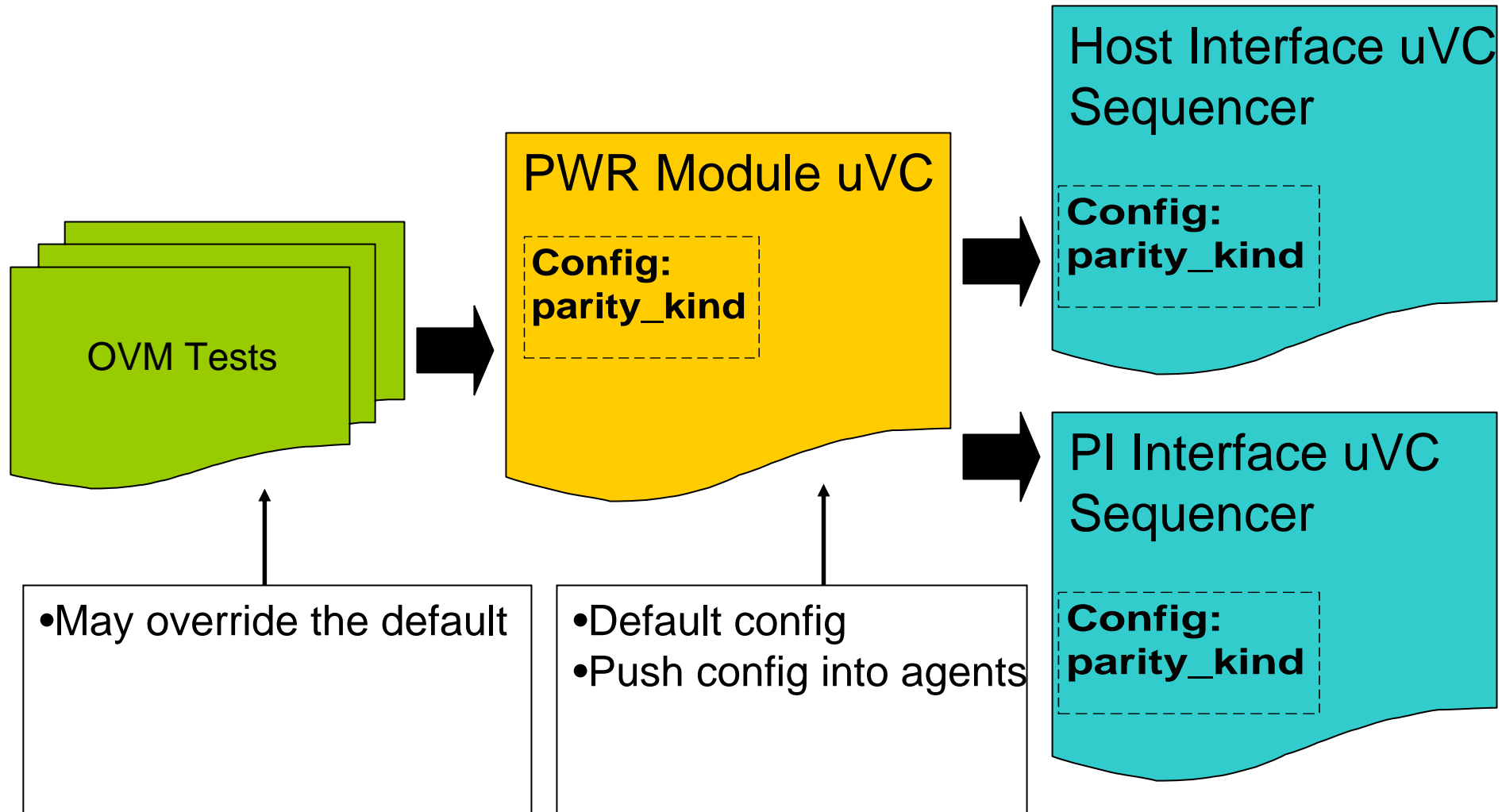
if (is_active) driver is enabled;
else only monitor is enabled

// Testbench Parameter

```
class host_sequencer extends ovm_agent;
```

```
protected pwr_parity_kind parity_kind; // user defined enum {ODD,EVEN}
```

Testbench Configuration (con't)



Testbench Configuration (con't)

STEP 1 – Add field in uVCs and register it with the factory

```
class pi_master_sequencer extends ovm_sequencer;
```

```
protected pwr_parity_kind parity_kind;
```

```
class host_sequencer extends ovm_sequencer;
```

```
protected pwr_parity_kind parity_kind;
```

```
`ovm_component_utils_begin(host_sequencer)
```

```
  `ovm_field_enum (pwr_parity_kind, parity_kind, OVM_ALL_ON)
```

```
  `ovm_component_utils_end
```

TYPE

VALUE

FIELD

Register with
factory

Since **parity_kind** is used by **more** than 1 uVC we add its value in the env

```
class pwr_env extends ovm_env;
```

```
protected rand pwr_parity_kind parity_kind;
```

```
  ovm_component_utils_begin(pwr_env)
```

```
    `ovm_field_enum (pwr_parity_kind, parity_kind, OVM_ALL_ON)
```

```
    `ovm_component_utils_end
```

```
  ...
```

By default **parity_kind** is **random** (ODD or EVEN)!

Testbench Configuration (con't)

STEP 2 – Push down env's parity_kind to the uVCs

```
class pwr_env extends ovm_env;
```

```
virtual function void build();  
  super.build();
```

path

field

value

```
  set_config_int( "*", "parity_kind", parity_kind );
```

```
    // build host env
```

```
    $cast(host0, create_component("host_env", "host0"));  
    host0.build();
```

```
    // build pi env
```

```
    $cast(host0, create_component("pi_env", "pi0"));  
    pi0.build();
```

set_config*

- Searches top-down
- May use wildcards

Note: make sure to call set_config* **before** agent's build

Hint: for debugging set_config_* call host0.print() and pi0.print() here

Testbench Configuration (con't)

STEP 3 – Randomize the pwr's env in the sve

```
class pwr_sve_env extends ovm_env;  
  
virtual function void build();  
    super.build();  
  
    $cast(pwr0, create_component("pwr_env", "pwr0"));  
    assert(pwr0.randomize());  
    pwr0.build();  
    ...  
  
endfunction : build
```

Randomize
after
creating
pwr0
component
and building
pwr0

Testbench Configuration (con't)

STEP 4 – Tests may override the env's default configuration

```
class test_odd_parity_kind extends pwr_base_test;
```

```
  `ovm_component_utils(test_parity_kind)
```

```
virtual function void build();
```

path

field

value

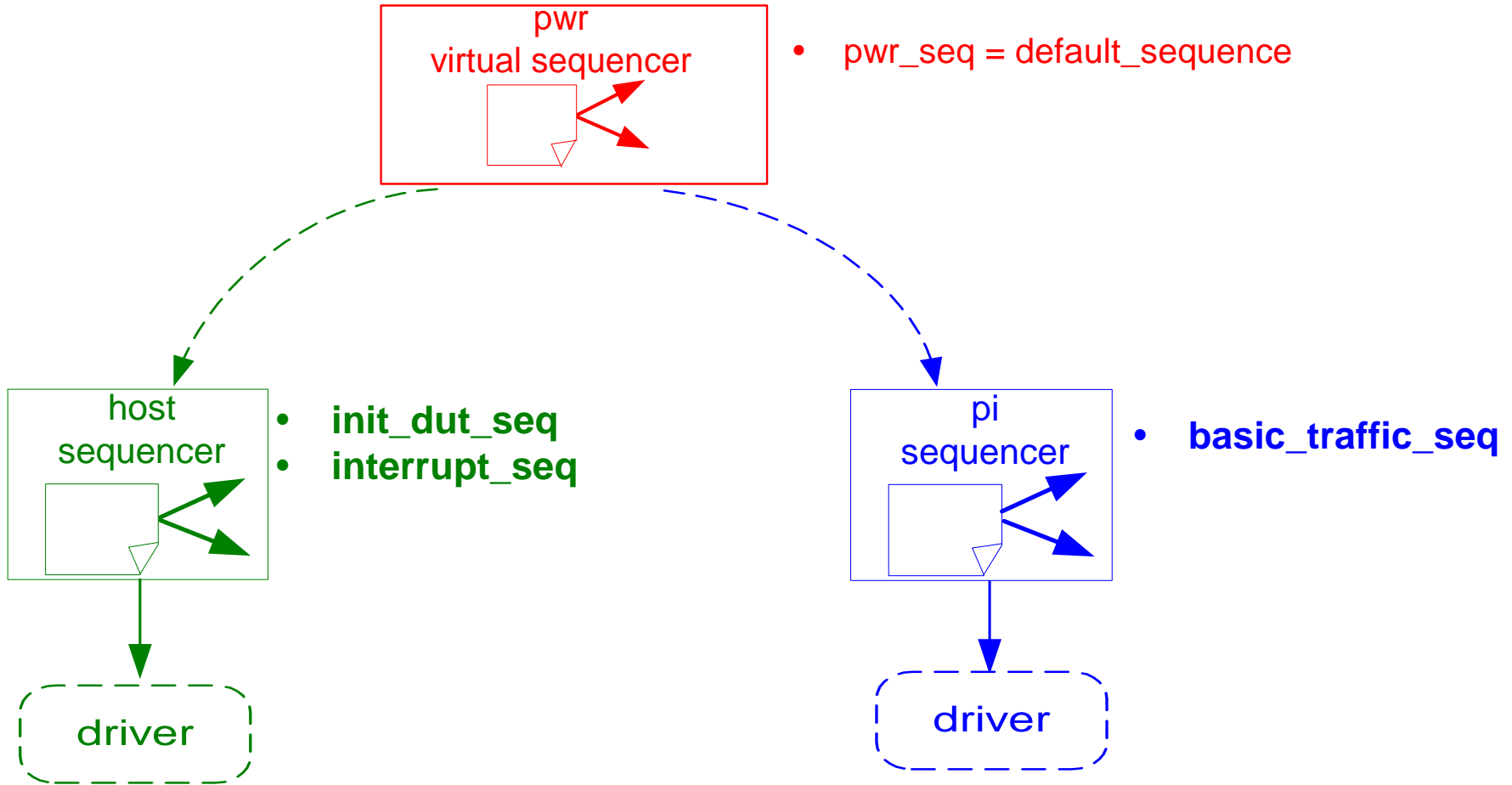
```
  set_config_int("pwr_sve0.pwr0", "parity_kind", ODD);
```

```
  super.build();
```

```
endfunction : build
```

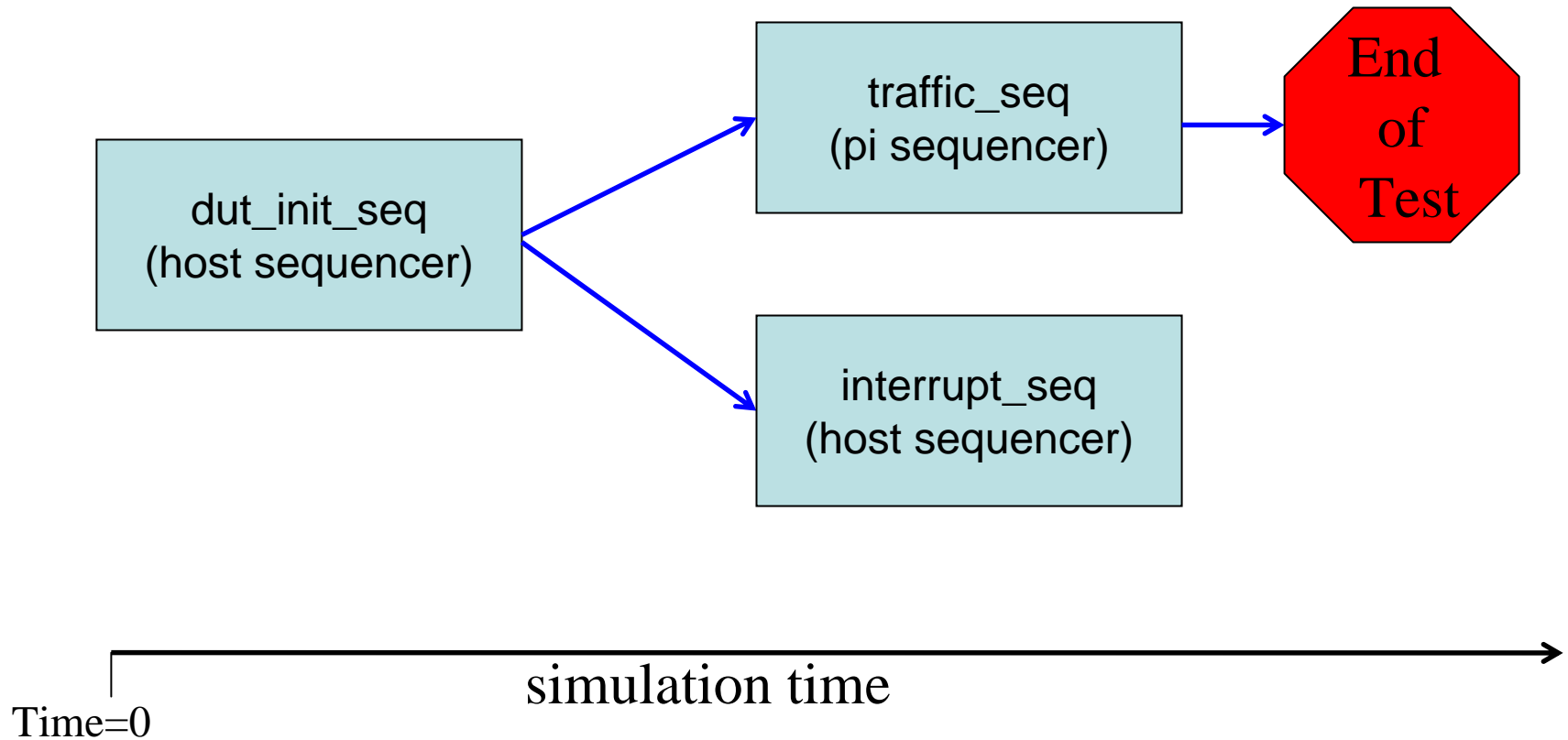
```
endclass : test_odd_packet_parity
```

Virtual Sequencer



PWR Virtual Sequence

pwr_seq virtual sequence progression



PWR Virtual Sequence (con't)

```
class pwr_seq extends ovm_sequence;
```

```
`ovm_sequence_utils(pwr_seq, pwr_virtual_sequencer)
```

Register
sequence

```
init_dut_seq      init_dut_seq_inst; // host sequencer
```

```
interrupt_seq    interrupt_seq_inst; // host sequencer
```

```
traffic_seq      traffic_seq_inst; // pi sequencer
```

Instantiate
sequences

```
virtual task body();
```

Call Init DUT with Host Sequencer

```
`ovm_do_seq(init_dut_seq_inst, p_sequencer.seq_cons_if["host_sequencer"])
```

```
fork
```

Sequence
instance

Sequencer

```
begin
```

```
`ovm_do_seq(traffic_seq_inst, p_sequencer.seq_cons_if["pi_sequencer"])
```

```
end
```

```
begin
```

Call traffic with PI Sequencer

```
`ovm_do_seq(interrupt_seq_inst, p_sequencer.seq_cons_if["host_sequencer"])
```

```
end
```

```
join_any
```

Call Interrupt with Host Sequencer

```
endtask
```

Simple Error Test

STEP 1 – Create new inherited traffic sequence

```
class my_error_traffic_seq extends traffic_seq;
```

```
`ovm_sequence_utils(my_error_traffic_seq, pi_sequencer)
```

Register
sequence

```
pi_transfer this_transfer;
```

```
virtual task body();
```

```
`ovm_create(this_transfer) // create a variable for manipulation
```

```
this_transfer.parity_error_c.constraint_mode(0); // turn off default constraint
```

```
`ovm_rand_send_with(this_transfer, { parity_error == 1'b1; } )
```

```
endtask
```

Force a parity
error!

```
endclass
```

Simple Error Test (con't)

STEP 2 – Create test that calls the error traffic sequence

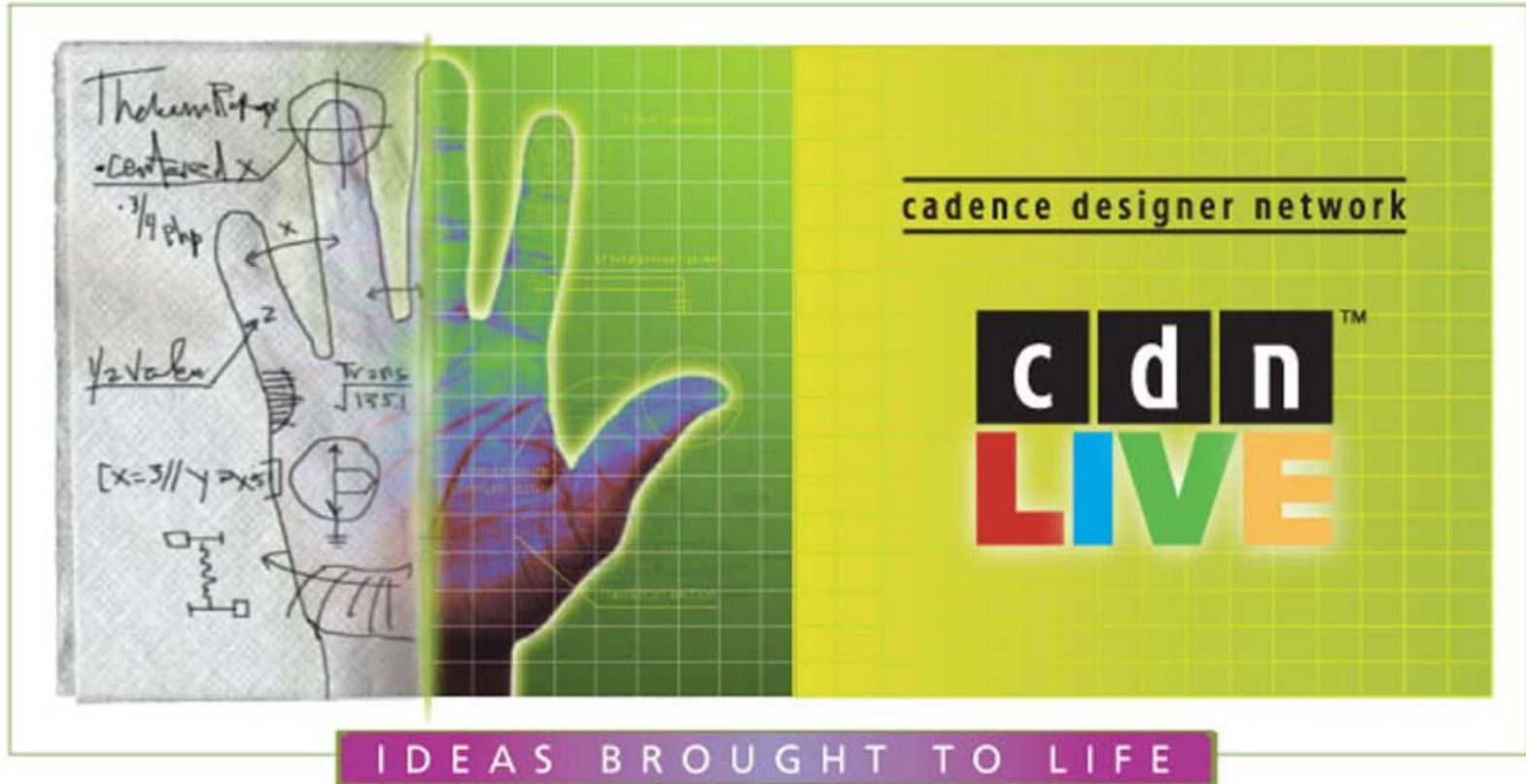
```
class test_error_packet extends pwr_base_test;  
  
  `ovm_component_utils(test_error_packet)  
  
  ...  
  
  virtual function void build();  
  ovm_factory::set_type_override( "traffic_seq",  
                                   "my_error_traffic_seq");  
  
  ----- OR -----  
  ovm_factory::set_inst_override( "*traffic_seq_inst",  
                                   "traffic_seq ",  
                                   "my_error_traffic_seq" );  
  
  // Create the sve  
  super.build();  
  endfunction : build  
endclass
```

These factory overrides will force
"my_error_traffic_sequence" to be invoked!

Hint: for debugging purposes call out
ovm_factory::print_all_overrides();

OVM Conclusion

- Migration effort was minimal
- Upfront learning curve
- Open-source helps debugging
- Promotes readable and maintainable code
- Powerful for reuse
 - Virtual Sequences
 - Configuration Management & Class Factory
 - Hierarchical Structure
 - TLM



Contact Information:

stephen.donofrio@paradigm-works.com

www.paradigm-works.com