February 22-25, 2010

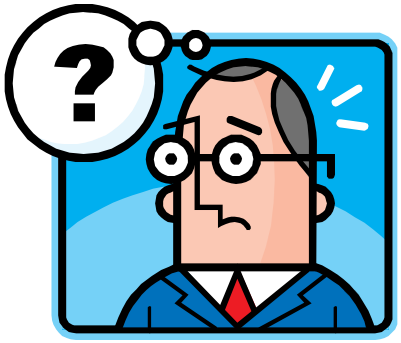
Configuration Mantra

by
Stephen D'Onofrio
Verification Architect

Introduction

- OVM and VMM 1.2 provide **TWO** techniques for declaring *configuration fields*
  - Set/Get “Configuration Mechanism”
  - Configuration Classes

Which techniques should I use???



Configuration Mantra Overview

- Categorizing configuration
- Overview configuration techniques
- Breaking up configuration classes
- Configuration mechanism complexities
  - Randomizing
  - Overriding
  - Duplicate fields
- Conclusion

Configuration Categories

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

```
// Verification Knobs
int unsigned  max_intra_gap_delay;
int unsigned  min_intra_gap_delay;
```

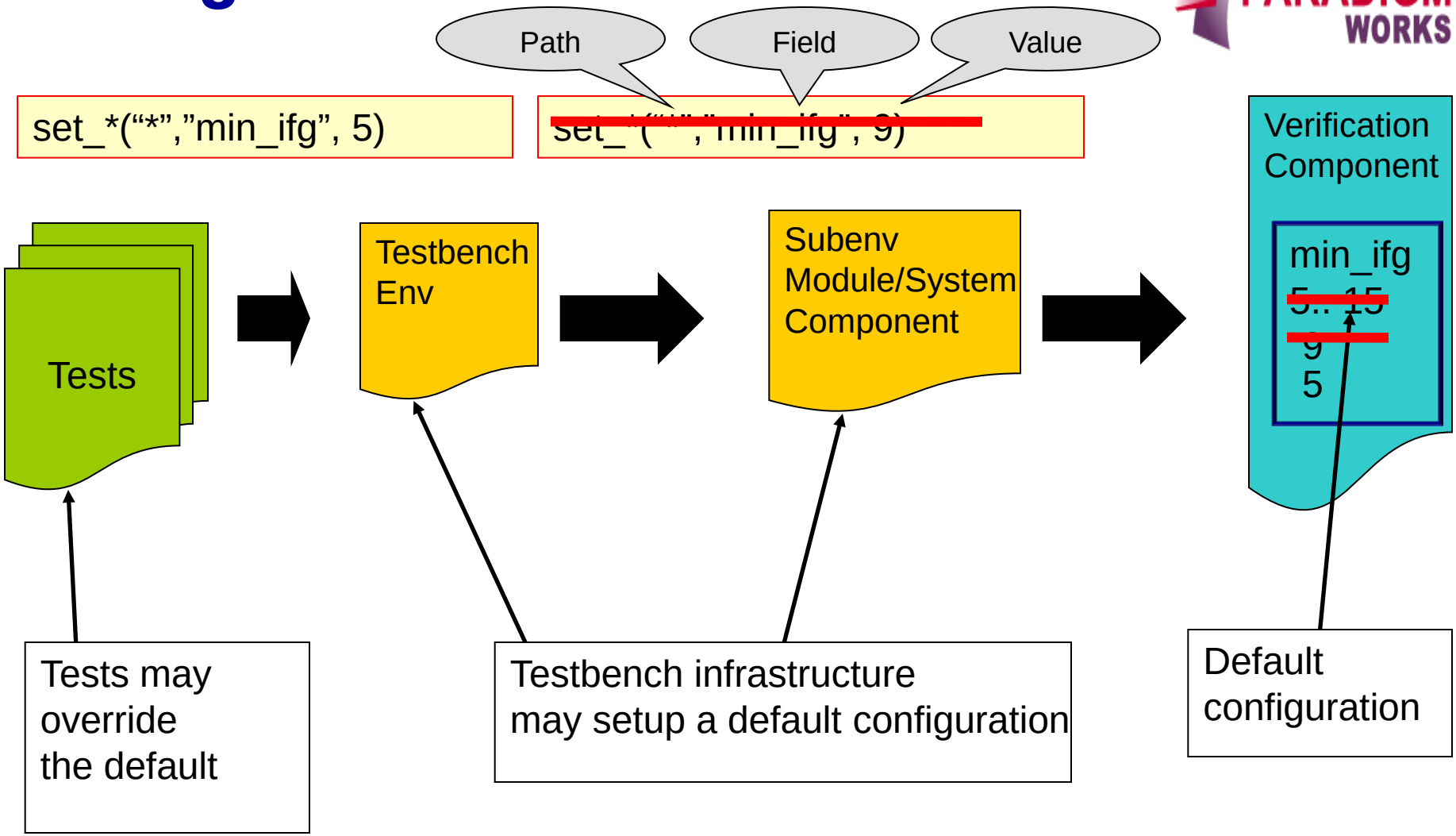
```
// Topology Knobs
ovm_active_passive_enum is_active; // {ACTIVE,PASSIVE}
```

```
// Test Knobs
int unsigned  num_packets;
bit has_coverage;
```

Configuration Mantra Overview

- Categorizing configuration
- **Overview configuration techniques**
- Breaking up configuration
- Configuration mechanism complexities
  - Randomizing
  - Overriding
  - Duplicate fields
- Conclusion

Configuration Mechanism



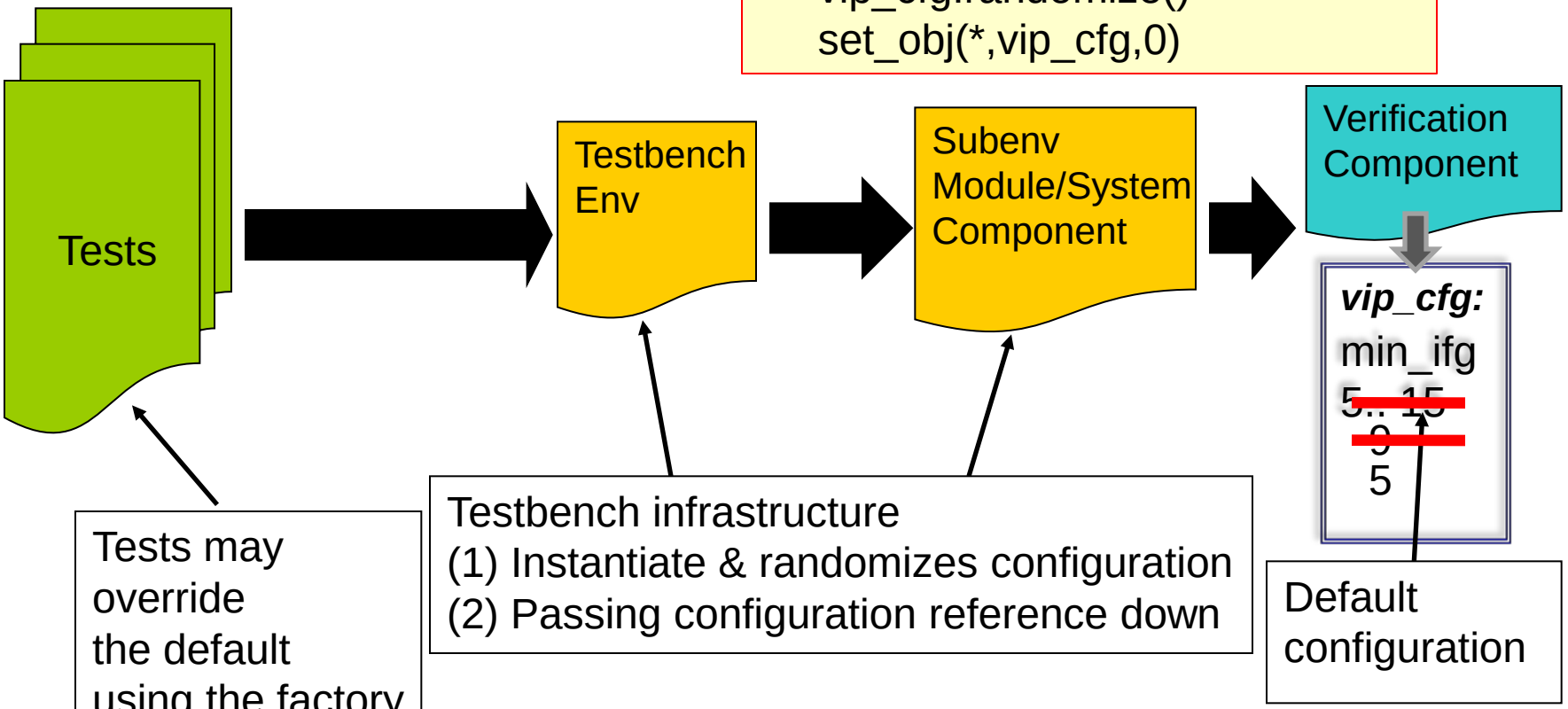
Configuration Classes

```
class tst_env_vip_cfg extends env_vip_cfg;
constraint c_min_ifg{ min_ifg == 5}

test extends ovm/vmm_test;
   factory_set_type_override_by_type…
```

```
class env_vip_cfg extends vip_cfg;
constraint c_min_ifg{ min_ifg == 9}
```

```
testbench extends ovm/vmm_env;
     vip_cfg.randomize()
     set_obj(*,vip_cfg,0)
```

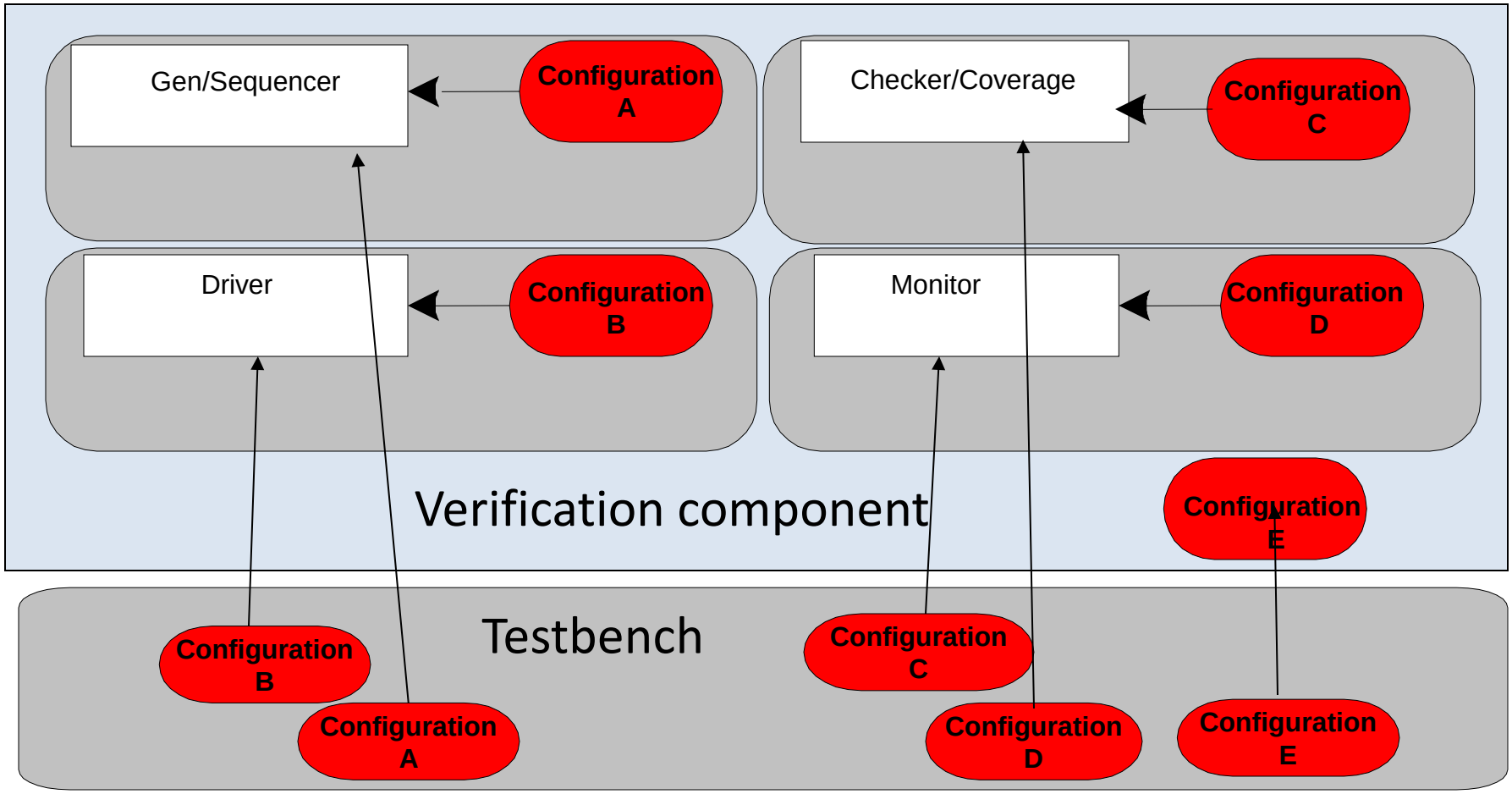


Configuration Mantra Overview

- Categorizing configuration
- Overview configuration techniques
- **Breaking up configuration classes**
- Configuration mechanism complexities
  - Randomizing
  - Overriding
  - Duplicate fields
- Conclusion

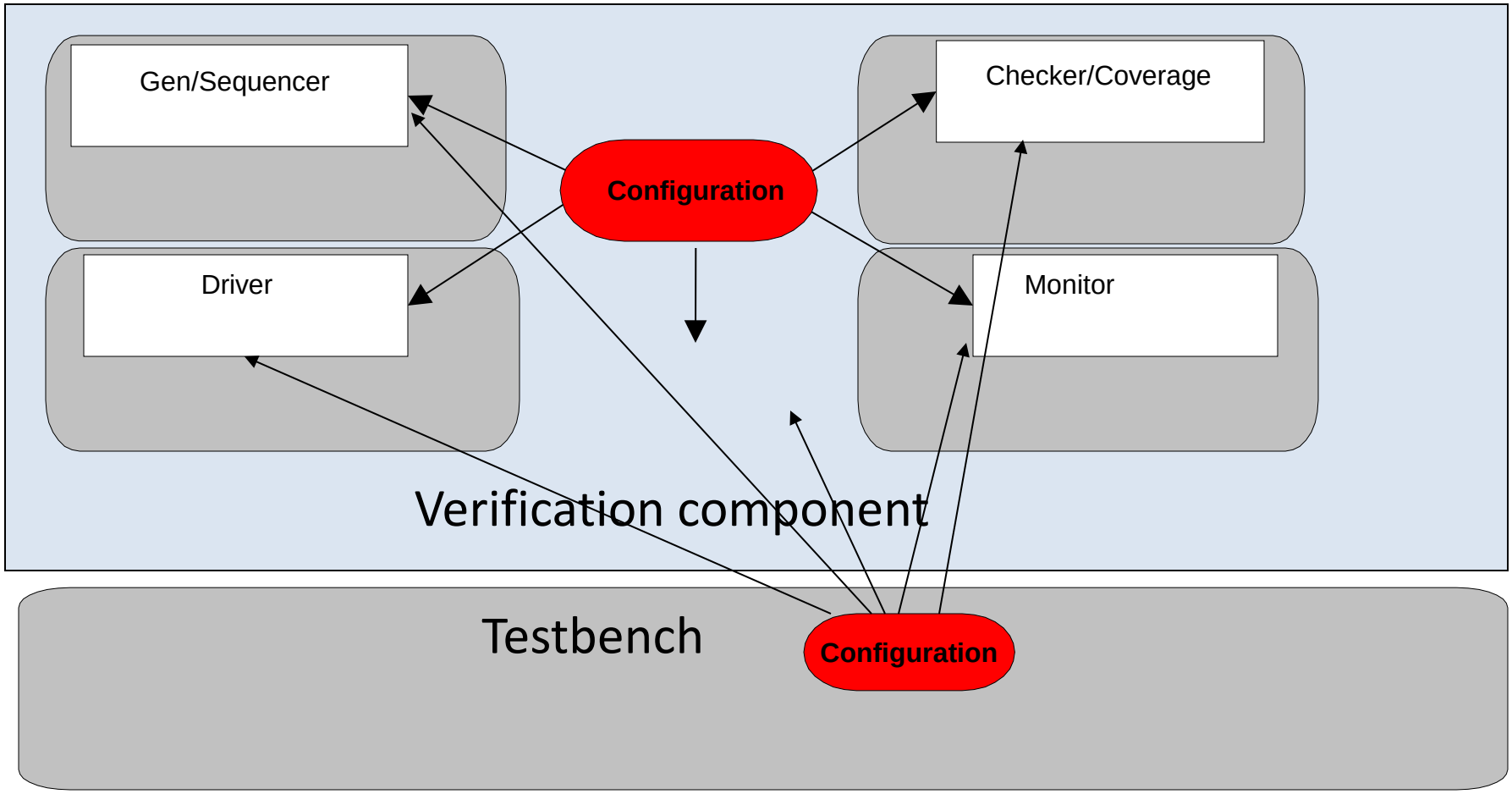
Breaking up Configuration

Option A - configuration class for each component



Breaking up Configuration

Option B – single configuration class



Configuration Mantra Overview

- Overview configuration techniques
- Breaking up configuration
- Categorizing configuration
- **Configuration mechanism complexities**
  - **Randomizing**
  - **Overriding**
  - **Duplicate fields**
- Conclusion

Randomize with Configuration Mechanism

```
class mac_agent extends ovm_env;

 rand int unsigned min_ifg;

`ovm_component_utils_begin(mac_agent)
  `ovm_field_int(min_ifg, OVM_ALL_ON+OVM_DEC)
`ovm_component_utils_end

constraint min_ifg_c { min_ifg inside { [5:15] }; }

function void build();
   this.randomize();
   super.build();
   ....
```

SV rand keyword

Register w/ config

SV constraint

SV randomize

For VMM vmm_unit

For VMM `vmm_data_member

For VMM
(1) start_of_sim_ph
(2) vmm_opts::get_*
(3) shut off rand and constraint

Overriding with Configuration Mechanism

Single Value Override

```
class test_mac extends ovm_test_base;


 virtual function void build();
   set_config_int(“mac_inst.*", "min_ifg", 5);
```

For VMM vmm_test

For VMM configure_test_ph

For VMM vmm_opts:set_int*

Overriding with Configuration Mechanism

Distribution Override Example 1

Add new class

Register w/ factory

Override constraint

```
class test_mac_agent extends mac_agent;

`ovm_object_utils(test_mac_agent)

constraint min_ifg_c { min_ifg dist { [1:4] :/ 5, 3 := 2, 6 := 5};}
```

Factory override

```
class test_mac extends ovm_test_base;

 virtual function void build();

  factory.set_type_override_by_type(mac_agent:get_type(),
                                    test_mac_agent ::get_type());
```

Overriding with Configuration Mechanism

Distribution Override Example 2

```
class test_mac extends ovm_test_base;

 virtual function void build();

   int unsigned my_min_ifg

   randcase
     1 : my_min_ifg = 7;
     2 : my_min_ifg = 8;
   endcase

   set_config_int(“mac_inst.*", "min_ifg", my_min_ifg);
```

Use procedural random distribution call

Override agent’s “min_ifg” using set*

Overriding with Configuration Mechanism

Range/Distribution Override Example 3

```
class mac_agent extends ovm_env;
…
  int unsigned  min_ifg_start= 3;
  int unsigned  min_ifg_end =5;

  constraint min_ifg_c {
                          min_ifg >= min_ifg_start;
                          min_ifg <= min_ifg_end;
                        }


  `ovm_component_utils_begin(mac_agent)
   `ovm_field_int(min_ifg_start, OVM_ALL_ON+OVM_DEC)
    `ovm_field_int(min_ifg_end, OVM_ALL_ON+OVM_DEC)
  `ovm_component_utils_end
```

Add **min_ifg_start** & **min_ifg_end** fields

Duplicate Fields with Configuration Mechanism

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

```
class host_monitor extends ovm_component;

pwr_parity_kind parity_kind;
```

```
class host_driver extends ovm_component;

pwr_parity_kind parity_kind;

`ovm_component_utils_begin(host_driver)
    `ovm_field_enum (pwr_parity_kind, parity_kind, OVM_ALL_ON)
`ovm_component_utils_end
```

Duplicate Fields with Configuration Mechanism

STEP 2 – Add parity_kind and randomize it in host env

```
class host_env extends ovm_env;

rand pwr_parity_kind override_parity;

 `ovm_component_utils_begin(host_env)
   `ovm_field_enum (pwr_parity_kind, override_parity, OVM_ALL_ON)
 `ovm_component_utils_end

virtual function void build();
   if (this.randomize() == 0)
      ovm_report_fatal("build", "randomize failed");
    super.build();

    set_config_int("*", "parity_kind", override_parity);
...
endfunction : build
```

Create random **override_parity** which overrides the *parity_kind* fields in the monitor & driver

Comparing Configuration Techniques

General
- Config class - encapsulate config knobs
- Config mechanism – knobs located where used & set*

Randomize
- Config class – randomize in testbench & pass reference
- Config mechanism – randomize components & duplicate knobs

Override
- Config class – always extend class & factory override class
- Config mechanism - multiple techniques

Dynamic
- Config class - reference
- Config mechanism – build phase natively & config knobs scattered

Conclusion

- Suggest using single configuration class for verification components (VIPs)
- Which configuration technique should I use?
  - Design and Verification Knobs seem more fit for configuration class due to more intense randomization
  - Topology and possibly Test Knobs can take advantage of configuration mechanism's set_* due to less intense randomization
- Teams should deploy consistent configuration approach to ensure interoperability
  - See SVF Template Generator

  http://svf-tg.paradigm-works.com/svftg

Question and Answers

- Thanks!
- Contact Information and OVM/VMM 1.2 examples

  stephen.donofrio@paradigm-works.com