Self-Tuning Coverage

Jonathan Bromley
Overview

• Coverage reuse needs flexibility, configurability

SELF TUNING in response to configuration, parameters etc

• Coverage can mislead!

See Verilab papers DVCon-US 2015 and suggestions in this tutorial

• SV covergroups are not easy to configure

More detail and recommendations later in this tutorial

• SV covergroups are not easy to reconfigure

More detail and recommendations later in this tutorial
COVERAGE GUIDELINES
Coverage Encapsulation

• Coverage should be encapsulated for maintenance
  – isolate coverage code
  – separate class for coverage
  – each coverage class should be in its own file
  – minimize SV assertion-based coverage

• **but** architecture of coverage classes needs care
  – more details in later sections
Implementation Options in SV

- Coverage in a subscriber class
- Extend a component to add coverage groups
- Instantiate a coverage class in some component
Implementation Options in SV

• Coverage in a subscriber class
  
  environment structure

• Extend a component to add coverage groups

• Instantiate a coverage class in some component

Implementations in SV

```verbatim
class ... extends uvm_subscriber#(dvcon_txn);

  dvcon_txn cov_txn; // no need for deep copy

  covergroup cg;
  ...
  coverpoint cov_txn.size {...}
  ...
  endgroup

  function void write(dvcon_txn t);
  cov_txn = t;
  cg.sample();
  endfunction
```

no need for deep copy

no need for deep copy
Implementation Options in SV

- Coverage in a subscriber class
- Extend a component to add coverage groups

```verilog
class dvcon_cov_monitor extends dvcon_monitor;
    `uvm_component_utils(dvcon_cov_monitor)

covergroup cg;
    ...
endgroup
```

- can be sampled at any point in the code
- has access to all class members

```verilog
uvm_factory f = uvm_factory::get();
f.set_type_override_by_type (dvcon_monitor::get_type(),
                            dvcon_cov_monitor::get_type());
no override, no coverage!
```
Implementation Options in SV

- Coverage in a subscriber class
- Extend a component to add coverage groups
- Instantiate a coverage class in some component

```verilog
class dvcon_cov extends uvm_component;
  `uvm_component_utils(dvcon_cov)
  dvcon_monitor p_monitor;
  covergroup cg;
  coverpoint p_monitor.value {...}
  ...
  function new(string name, uvm_component parent);
    super.new(name, parent);
    $cast(p_monitor, parent);
    cg = new();
    endfunction
```

No instance, no coverage!
Implementation Options - review

• Coverage in a subscriber class
  – restricted to contents of a single object (transaction, transaction-set)
  – but otherwise provides good isolation

• extend monitor class to add coverage groups
  – flexible, but limits TB structure
  – instantiate correct monitor type, else coverage missed!

• instantiate coverage class in another component
  – coverage class has handle to component, sees all contents
  – allows all coverage implementation including control/timing
  – most flexible solution

Best choice depends on application!
Coverage Is Distributed

• Coverage of individual transactions: easy but insufficient
  – cover each txn from monitor – *transaction coverage*

• DUT behaviour coverage is also important, but sampling and data gathering likely to be distributed across...
  – parts of verif env
    • activity on other interfaces
    • register state and changes
    • end-to-end matching
  – time
    • what else happened during the life of thistxn/instruction/...?
    • DUT state at start, end, other key points in txn lifetime?
Coverage Is Distributed

- Coverage from *many places* contributes to verif plan

- Coverage of end-to-end behaviours and checks

- Cross with system-wide concerns (reset, congestion, mode changes...)

- Transaction coverage

- DUT internals and register coverage

- Configuration coverage

- Assertion coverage
Coverage Is Distributed

- Coverage from *many places* contributes to verif plan
- Encapsulate!
- Many connections may be required

cross with system-wide concerns (reset, congestion, mode changes...)

© Verilab & Accellera
Coverage Tuning

• Modify the coverage model to suit our requirements
  – for example exhaustive block-level coverage not appropriate for top-level verification requirements, so reduce scope

• Reuse the same mechanism for sampling coverage
  – this is built into the component implementation
  – fully identifies what cover points are sampled and when

• Redefine the coverage groups and coverpoints
  – modify the number of coverpoints in existing cover group
  – modify range and conditions of existing coverpoint bins
  – _e_ uses AOP to redefine existing coverpoints
  – _SystemVerilog_ uses OOP factory to replace existing class
IMPLEMENTING FLEXIBLE COVERAGE
Making Coverage Flexible

- Traditional approaches to configurable points/bins
  - sample() wrapper – map sampled data to coverage-friendly value
  - `enum {normal, fizz, buzz, fizzbuzz} fb;`
  - `bit this_bit;
  int bit_num;
  covergroup cg;
  coverpoint this_bit;
  coverpoint bit_num;
  cross this_bit, bit_num;
  endgroup`

- SV gotchas:
  - “iff” doesn’t remove a bin, just suppresses a sample
  - Effects of illegal, ignore, default – often misunderstood

```verilog
defined enum {normal, fizz, buzz, fizzbuzz} fb;

bit this_bit;
int bit_num;
covergroup cg;
coverpoint this_bit;
coverpoint bit_num;
cross this_bit, bit_num;
cross this_bit, bit_num;
endgroup

function sample (bit [31:0] x);
for (bit_num = 0; bit_num < 32; bit_num++) begin
  this_bit = x[bit_num];
cg.sample();
end
endfunction
```
Recent Language Enhancements

• Available SV-2012 options:
  – bin with() function (value filter)
  – bin set expression (value list specification)
  – cross bin with()/matches (tuple filter, match threshold)
  – cross bin set expression (queue of value tuples)

• Many scope visibility gotchas
  – good examples are hard to find
  – meanwhile, read LRM carefully and try small test examples

• Patchy tool support
  – with() functions are widely supported
  – set expressions in some tools
Bin Specification Improvements

- Illegal cross bins example taken from a real-world problem:
  - Illegal if \texttt{active}==0 or if \texttt{active}==\texttt{ready}
  - Illegal if \texttt{active} has any bits set that are not set in \texttt{ready}

- \texttt{binsof} expressions: messy, inflexible

```verilog
covergroup cg_active_ready(int Nbits);
...

active_X_ready: cross active, ready {
  function CrossQueueType all_illegals(int n);
  for (int R=0; R<(1<<n); R++)
    for (int A=0; A<(1<<n); A++)
      if ( A==0 || A==R || (A & ~R)!=0 )
        all_illegals.push_back( '{A,R} );
  endfunction
  illegal_bins bad_active = all_illegals(Nbits);
}
endgroup
```

Easy in SV-2012!

Self-tuning for Nbits
RECONFIGURABLE COVERAGE
Configurable Coverage

• Any good reusable VC or env has a **configuration object**
• Can we use this config object to tune coverage?
• YES, but there are challenges:
  – covergroup must be created in a class's constructor
  – this is too early in a UVM component; config is not yet known
• Following slides offer two solutions
  1. wrap covergroup in uvm_object, get config from UVM config DB
     • straightforward
  2. embedded class constructed much later, after config is known
     • applicable to both components and objects
     • components locate coverage securely in the topology
Making It Configurable

• Covergroup in a component can't be configured:

```java
class dvcon_monitor extends uvm_monitor;
    covergroup dvcon_cg(int max); 
    function new(string name, uvm_component parent = null);
    super.new(name, parent);
    dvcon_cg = new(...);
endgroup
function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    dvcon_cg = new(...);
endfunction
...
```

• Config information is available much later

```java
function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    dvcon_cg = new(...);
endfunction
```

• We need a workaround...

© Verilab & Accellera
Configurable Coverage Roadblock

• Key problem:

  **CG must be created in enclosing class's `new()`**
  
  **but...**

  UVM classes have fixed constructor arguments

• Solution 1: constructor gets info from `config_db` before CG `new`
  
  – OK for `uvm_object`, supports factory
  
  – Parent object must prepare `config_db` entries before creation
  
  – Unhelpful for `uvm_component`: `config info may not be available`

• Solution 2: Encapsulate CG in a non-UVM class
  
  – pass `config` in constructor arguments

  – factory cannot replace a non-UVM class **`good for dedicated coverage components`**
Workaround 1: Coverage Object

class my_component...
  ...
  dvcon_cov cov_wrap;
  string cov_name = "cov_wrapper";
  function void start_of_simulation_phase(...);
  dvcon_cov_cfg cfg = new("cov_cfg");
  ...
  uvm_config_db#(dvcon_cov_cfg)::set(
    null, cov_name, "cfg", cfg);
  cov_wrap = dvcon_cov::type_id::create(cov_name);
endfunction
...

coverage wrapper class

component needs configurable coverage

coverage object can be factory-overridden

late config and creation

class dvcon_cov extends uvm_object;
  `uvm_object_utils(dvcon_cov)
  covergroup dvcon_cg(int max); ... endgroup
dvcon_cov_cfg cfg;
  function new(string name = "");
    super.new(name);
    uvm_config_db#(dvcon_cov_cfg)::get(
      null, name, "cfg", cfg);
    dvcon_cg = new(cfg.max);
endfunction
endclass

coverage wrapper class
Workaround 2: Coverage Wrapper

- For easy factory replacement, component is coverage only!

```cpp
class dvcon_txn_cvg extends uvm_subscriber #((dvcon_txn);
`uvm_component_utils(dvcon_txn_cvg);

class cov_wrapper;
  covergroup dvcon_cg(int max); ...; endgroup
  function new(string name, dvcon_config cfg); ... 
  ... 
endclass

dvcon_config cfg;
cov_wrapper cov;
virtual function void create_coverage();
  cov = new({get_full_name(), "cov"}, cfg);
endfunction

virtual function void write(dvcon_txn txn);
  cov.sample(txn);
endfunction
```

- register with factory
- nested (local) class definition
- `cfg` set from above in build_phase or later
- do not call until `cfg` is ready
Coverage Wrapper Details

- Delegate sampling to virtual methods for easy extension

```plaintext
class cov_wrapper;
    dvcon_txn txn;
    covergroup dvcon_cg(int max),
        cp_len: coverpoint txn.length {
            bins tiny[] = {[0  :3  ]};
            bins mid   = {[4  :max-4]};
            bins limit[] = {[max-3:max  ]};
        }
    endgroup
    function new(...); ...; endfunction
    virtual function void sample(dvcon_txn t);
        txn = t;
        dvcon_cg.sample();
    endfunction
endclass
```

- arbitrary covergroup arguments used to configure bin shapes
- no need for object copy – `txn` is used only during `sample()`
Configurable Coverage Component

• nested (wrapper) class contains covergroup(s)
• not a uvm_object – arbitrary constructor signature OK
• nested-class object can be constructed any time
  – postpone until config is fully known
• component encapsulates responsibility for:
  – understanding and preparing configuration
  – constructing nested-class object
  – data collection and sampling
• prepare for extension, factory applicability
Reference

- Guidance on coverage design:
  - avoid lies
  - maximise effectiveness

**DVCon US 2015, Mark Litterick**
Lies, Damned Lies, and Coverage

**DVCon US 2015: Jason Sprott, Paul Marriott, Matt Graham**
Navigating The Functional Coverage Black Hole:
Be More Effective At Functional Coverage Modeling