Uncover
Functional Coverage Made Easy

Akash S
Rahul Jain
Gaurav Agarwal

NVIDIA
You too?
What makes coverage coding tedious?

• **Monotony**
  – Bland work, not challenging enough!

• **Repetition**
  – Widespread across covergroups, projects.

• **Inconsistency**
  – Differences in structure, model, style

“Thou shalt keep verif engineers happy”
The Big Picture

Plan

Code

Implement

Analyse

Shorthand

SV Code

Review XLS

Hook up

Regress
“Thou shalt automate the mundane”
“Thou shalt be TRANSACTION-WISE”
“Thou shalt separate concerns”
“Thou shalt be flexible”

- Flow adaptable to whitebox coverage.
- Use the same infra with additional command line arguments.
“Thou shalt be concise”

Reviewability

• Structured XLS for review of implementation.
• Only coverage model, cut the rest.
“Thou shalt be concise”
Coding: The Shorthand Notation

• One-line CG/CP/cross definitions
• Auto-generate whatever possible
• No repetitions
• Leverage on patterns
• Customizable sequence coverage
```cpp
//Generated by Auto Cov flow. Do not edit!
import uvm_pkg::*;

class cg_basic_txn_cov extends uvm_subscriber#(cg_basic_txn);
  `uvm_component_utils(cg_basic_txn_cov)
  cg_basic_txn obj;

  covergroup cg_basic_txn_1;
    cp_addr : coverpoint obj.addr {
      bins rom = {0:5};
      ignore_bins reserved = {{13:15}};
      illegal_bins rom = {6:12};
    }
    cp_w_rn : coverpoint obj.w_rn {
     }
  endgroup: cg_basic_txn_1

  covergroup bus_txn;
    cp_wr_data : coverpoint obj.wr_data iff (obj.w_rn) {
      option_auto_bin_max = 10;
      bins min[2] = {0};
      bins max[] = {127,126};
    }
    read : coverpoint obj.rd_data iff (obj.w_rn) {
     }
  endgroup: bus_txn

  function new(string name, uvm_component p = null);
    super.new(name,p);
    bus_txn = new;
    cg_basic_txn_1 = new;
  endfunction: new

  function void pre_sample();
  endfunction: pre_sample

  function void post_sample();
  endfunction: post_sample

  function void write(cg_basic_txn t);
    `uvm_info(get_full_name(), "In COVERAGE class write method. About to sample coverage", UVM_NONE)
    obj = t;
    if (posedge clk)
      bus_txn.sample();
    cg_basic_txn_1.sample();
    `uvm_info(get_full_name(), "Done sampling. Exiting COVERAGE class write method", UVM_NONE)
  endfunction: write

endclass: cg_basic_txn_cov
```

---

**Shorthand**

```cpp
1 siglist:
2   bit [3:0] addr;
3   bit [7:0] wr_data;
4   bit w_rn;
5   bit [7:0] rd_data;
6   bit [at_least=1, per_instance = 1];
7   cp addr: (addr) ram([0:5]), -rom([6:12]), *reserved([13:15]);
8   cp: [w_rn];
9   cp_bus_txn: (posedge clk);
10  cp: (wr_data)[w_rn][auto_bin_max = 10] min[2][0], max[127,126];
11  cp read: (rd_data) [1w rn];
```

---

**"The mundane"**

---

© Accellera Systems Initiative 11
# Shorthand: A Glance

## SV snippet

```sv
//Comment for CG bus_txn.
covergroup bus_txn;

if (test_condition) {
  bus_txn.sample();
}
```

*(Inside write function of cov collector)*

## Shorthand snippet

```shorthand
cg bus_txn: [test_condition];
//Comment for CG bus_txn
```

```shorthand
cp cp_addr: {addr} ram([0:5]), -rom([6:12]), *reserved([13:15]);
```

```shorthand
cp_addr: coverpoint obj.addr {
  bins ram = {[0:5]};
  ignore_bins reserved = {[13:15]};
  illegal_bins rom = {[6:12]};
}
```
SV snippet

```sv
# Comment for cross first attempt.
first_attempt : cross wr_data, w_rn {
  option.goal = 100;
  bins bin_1 = binof(cp_w_rn) intersect {1,2,3};
}
```

Shorthand snippet

```shorthand
cp : {wr_data}[w_rn][auto_bin_max = 10] min[2](0), max[](127,126), ?test(4'b11);
cross first_attempt :{wr_data,w_rn}[goal = 100] bin_1(cp_w_rn{1,2,3});
aliases_vars:
  bit write_en = test_alias_1 & test_alias_2;
  // Test var alias comment
  bit [5:0][4:0] write = xyz;
```
“Thou shalt not repeat thyself”

- **Problem:**
  - CPs can’t be used across CGs
  - Define CP in every CG where needed.
  - 100s of repetitions!

- **Solution:**
  - One-point definition of all CPs
  - Command to copy CP/cross to a CG.
  - No repetitions!

- **Scenarios?**
  - speed x gears x sampling rate
  - speed x success; speed x error
  - speed x timeout

```
aliases_cp:
cp cp_addr: {addr} ram([0:5]), -rom([6:12]), *reserved([13:15]);
cross test: {addr, wr_data};
```

```
cp_copy cp_addr;
cross_copy test;
```

Replace above statements with actual shorthand definition before converting to SV
“Harness patterns”

- Many-many relationships
- Irregular address ranges
- One-hot/one_cp_each
- Hop

```cpp
crossExpand("cp_1, cp_2", "cp_a, cp_b, cp_c");
```
## Macros: Gist

<table>
<thead>
<tr>
<th>Macro</th>
<th>Format</th>
<th>Example cfg syntax</th>
<th>O/P SV code</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>^list(vals)</td>
<td>^list(<a href="1">2</a>, *(2), -{3}, {4});</td>
<td>bins bin_list0[2] = {1}; ignore_bins ignore_list0 = {2}; illegal_bins illegal_list0 = {3}; bins bin_list1 = {4};</td>
</tr>
<tr>
<td>hop</td>
<td>^hop(min, max, step)</td>
<td>^hop(1,5,2);</td>
<td>bins bins_hop_1 = {1}; bins bins_hop_3 = {3}; bins bins_hop_5 = {5};</td>
</tr>
<tr>
<td>interval</td>
<td>^interval(h/d, min, max, break_1, break_2)</td>
<td>^interval(d, 1, 10, 5, 7); (Similarly for hex)</td>
<td>bins bins_interval_1 = {{5:1}}; bins bins_interval_2 = {{7:6}}; bins bins_interval_3 = {{8:10}};</td>
</tr>
<tr>
<td>range_interval</td>
<td>^range_interval(min, max, step_size)</td>
<td>^range_interval(1,a,4);</td>
<td>bins bins_range_interval_1 = {{‘h5:’h11}}; bins bins_range_interval_4 = {{‘ha:’h6}};</td>
</tr>
<tr>
<td>expand</td>
<td>^expand(expr)</td>
<td>^expand((1=&gt;2,3), {4=&gt;5});</td>
<td>bins transition_1_2 = {1=&gt;2}; bins transition_1_3 = {3=&gt;3}; bins transition_4_5 = {4=&gt;5};</td>
</tr>
<tr>
<td>list_transition</td>
<td>^list_transition(vals)</td>
<td>^list_transition([2] =&gt; {1=&gt;2,3}, *(4=&gt;5=&gt;6), -{7=&gt;8})</td>
<td>bins bin_list0[2] = {1=&gt;2,3}; illegal_bins illegal_list0 = {4=&gt;5=&gt;6}; ignore_bins ignore_list0 = {7=&gt;8};</td>
</tr>
</tbody>
</table>

### Expanded cfg syntax

<table>
<thead>
<tr>
<th>Macro</th>
<th>Format</th>
<th>Example cfg syntax</th>
<th>Expanded cfg syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>one_cp_each</td>
<td>^one_cp_each(&lt;signal&gt;)</td>
<td>^one_cp_each(test) is: case-1: 2 bit vector case-2: enum with pass/fail</td>
<td>Case-1: - cp test_bit_0 - cp test_bit_1 Case-2: - cp test_enum_pass - cp test_enum_fail</td>
</tr>
<tr>
<td>one_hot</td>
<td>^one_hot(&lt;signal&gt;)</td>
<td>^one_hot(test) is a 2 bit vector</td>
<td>cp test_one_hot: {test} bin_0(test[0]), bin_1(test[1]) ;</td>
</tr>
<tr>
<td>cross_expand</td>
<td>^cross_expand(cp_1,cp_2, “cp_3”, “cp_4”)</td>
<td>^cross_expand(“cp_1,cp_2”, “cp_3”, “cp_4”)</td>
<td>cross cp_1Xcp_3: {cp_1, cp_3}; cross cp_2Xcp_3: {cp_2, cp_3};</td>
</tr>
</tbody>
</table>
The Guiding Commandments

• Thou shalt keep verif engineers happy.
• Thou shalt automate the mundane.
• Thou shalt be TRANSACTION-WISE.
• Thou shalt separate concerns.
• Thou shalt not repeat thyself.
• Thou shalt be concise.
• Thou shalt be flexible.
Results

• Robust and reusable structure
  – Transaction based TLM coverage collectors

• Versatile flow

• Consistent implementation
  – Lesser ambiguity/disruptions

• Easy reviewability of implementation.

• Reduced coding efforts
  – Code to be written reduced by 75-80%

• Happy Engineers! 😊
Future

• Improvisations: patterns, parameterisations
• Leverage Verific
  – For transaction details
Questions?