Functional-Coverage Sampling in UVM RAL: Use of 2 Obscure Methods

Muneeb Ulla Shariff, Mirafra Technologies Pvt Ltd, Bangalore
Ravi Reddy, Roche Sequencing Solutions, Santa Clara
Agenda

• UVM RAL (Register Abstraction Layer)
• UVM RAL Functional Coverage
• Implementation
• Prediction
  – Auto Prediction
  – Explicit Prediction
• Role of Register Code Generator
• Comparison
• Results
UVM RAL

• Mimics the design hardware register contents at the TestBench (TB) side.

• Provides the abstract accesses to registers and memories.

Figure 1. UVM Register Model Integration [1]
UVM RAL Functional Coverage

• Functional Coverage is a measure of what functionalities/features of the design have been exercised by the stimulus/tests.

• UVM RAL Functional Coverage is helpful in providing the metrics used for gauging all register accesses including individual register bits.
The functional coverage of the RAL model is usually created by the register model generators. However, the sampling of the covergroup requires attentive work.

- sample() and sample_values() methods are used for sampling.

- Due to the lack of information about these methods, they are rarely and improperly used.
Implementation Steps

1. The covergroup and coverpoints must be defined. This is done using the register assistant tools.

```plaintext
class dp_deac_engine1_thresh1 extends uvm_reg;
   `uvm_object_utils(dp_deac_engine1_thresh1)

   uvm_reg_field reserved; // Reserved
   rand uvm_reg_field vote_thresh;
   rand uvm_reg_field compb_thresh;
   rand uvm_reg_field compa_thresh;

   // Function: coverage
   covergroup cg_vals;
   vote_thresh : coverpoint vote_thresh.value[7:0];
   compb_thresh : coverpoint compb_thresh.value[8:0];
   compa_thresh : coverpoint compa_thresh.value[8:0];
endgroup
```

Figure 2. Covergroup definition
2. The coverage model needs to be constructed conditionally.

```plaintext
// Function: new
function new(string name = "dp_deac_engine1_thresh1");
    super.new(name, 32, build_coverage(UVM_CVR_FIELD_VALS));
    add_coverage(build_coverage(UVM_CVR_FIELD_VALS));
    if(has_coverage(UVM_CVR_FIELD_VALS)) begin
        cg_vals = new();
        cg_vals.set_inst_name(name);
    end
endfunction
```

Figure 3. Covergroup construction [2]
Implementation Steps (contd..)

3. Before building the reg model, we need to set the `include_coverage(...)` to indicate which models to be constructed.

```cpp
// Building the register model
if (fpgadp_regs == null) begin
  // Specify which coverage model that must be included in various blocks,
  // register or memory abstraction class instances.
  uvm_reg::include_coverage("**", UVM_CVR_ALL);

  this.fpgadp_regs = fpgadp_register_pkg_uvm::fpgadp_cfg::type_id::create("fpgadp_regs", this);
  fpgadp_regs.build();

  // Enables sampling of coverage
  fpgadp_regs.set_coverage(UVM_CVR_ALL);
  fpgadp_regs.lock_model();
end
```

Figure 4. Enabling building and sampling of coverage [2]
Implementation Steps (contd..)

4. Eventually, you need to tell the compiler to enable coverage collection (The below options qualifies for Cadence Incisive Simulator)

- uvm -write_metrics -covfile cov_config_file -coverage All

```plaintext
# cov config file
set_covergroup -per_instance_default_one
```
Implementation Steps (contd..)

5. Finally we need to sample the coverage using the 2 methods, `uvm_reg::sample()` and `uvm_reg::sample_values()`.

- We need prediction to update the RAL model.

- Based on either auto-prediction mode or explicit-prediction mode, the `sample()` or `sample_values()` methods are used and implemented.
UVM RAL Prediction

• In UVM Register Modelling, a prediction is an art of keeping the Register Model up-to-date with expected results for the design registers.

• This allows us to compare the expected results from the Register Model with actual register values from the DUT.

• There are 2 modes:
  – Auto-Prediction Mode
  – Explicit-Prediction Mode
Auto-Prediction Mode

- In this prediction mode, the sequences using the UVM register API update the RAL model automatically.
Auto-Prediction Mode (contd..)

- On every register access, the `uvm_reg::sample()` method is called.

```plaintext
if (system_map.get_auto_predict()) begin
  uvm_status_e status;
  if (rw.status != UVM_NOT_OK) begin
    sample(value, -1, 0, rw.map);
    m_parent.XsampleX(map_info.offset, 0, rw.map);
  end

  status = rw.status; // do_predict will override rw.status, so we save it here
  do_predict(rw, UVM_PREDICT_WRITE);
  rw.status = status;
end
```

Figure 6. `uvm_reg::sample()` function call
Auto-Prediction Mode (contd..)

• The default uvm_reg::sample() function is empty.

```
protected virtual function void sample(uvm_reg_data_t data,
    uvm_reg_data_t byte_en,
    bit is_read,
    uvm_reg_map map);
endfunction
```

Figure 7. uvm_reg::sample() function definition
Auto-Prediction Mode (contd..)

Thus, to sample the coverage after each register access we need to implement the `uvm_reg::sample()` function.

```cpp
// Function: sample
protected virtual function void sample(uvm_reg_data_t data,
    uvm_reg_data_t byte_en,
    bit is_read,
    uvm_reg_map map);
    super.sample(data,byte_en,is_read,map);

foreach (m_fields[i])
    m_fields[i].value = ((data >> m_fields[i].get_lsb_pos()) &
                      ((1 << m_fields[i].get_n_bits()) - 1));

if (get_coverage(UVM_CVR_FIELD_VALS))
    cg_vals.sample();
endfunction
```

Figure 8. `uvm_reg::sample()` function implementation
Explicit-Prediction Mode

- This prediction mode updates the register model on all monitored transactions. It uses a predictor component and the UVC adapter.

Figure 9. Explicit Prediction Model
Explicit-Prediction Mode (contd..)

• With an explicit predictor, when register access is performed, the monitor sends out a transaction to the analysis port which is connected to uvm_reg_predictor and this triggers uvm_reg_predictor::write()

• This method updates the RAL model. After the update, we can explicitly call the uvm_reg::sample_values() method.

• The default uvm_reg::sample_values() function is empty.

// Function: sample_values
virtual function void sample_values();
endfunction
Explicit-Prediction Mode (contd..)

• Thus, in-order to sample the coverage we need to implement the uvm_reg::sample_values() function.

```cpp
// Function: sample_values
virtual function void sample_values();
    super.sample_values();
    if (get_coverage(UVM_CVR_FIELD_VALS))
        cg_vals.sample();
endfunction
```

Figure 10. uvm_reg::sample_values() function implementation
Explicit-Prediction Mode Example

class uvm_reg_predictor_custom #(type BUSTYPE=int) extends uvm_reg_predictor #(BUSTYPE);

    'uvm_component_param_utils(uvm_reg_predictor#(BUSTYPE))

    // Function : new
    function new (string name, uvm_component parent);
        super.new(name, parent);
    endfunction

    // Function : write
    // Over-riding the function to explicitly call the sample_values method
    virtual function void write(BUSTYPE tr);
        uvm_reg rg;
        uvm_reg_bus_op rw;

        // Calling the parent function
        super.write(tr);

        // Getting the register handle
        adapter.bus2reg(tr, rw);
        rg = map.get_reg_by_offset(rw.addr, (rw.kind == UVM_READ));

        // Sampling the coverage
        rg.sample_values();
    endfunction

class uvm_reg_predictor_custom #(type BUSTYPE=int) extends uvm_reg_predictor #(BUSTYPE);

    'uvm_component_param_utils(uvm_reg_predictor#(BUSTYPE))

    // Function : new
    function new (string name, uvm_component parent);
        super.new(name, parent);
    endfunction

    // Function : write
    // Over-riding the function to explicitly call the sample_values method
    virtual function void write(BUSTYPE tr);
        uvm_reg rg;
        uvm_reg_bus_op rw;

        // Calling the parent function
        super.write(tr);

        // Getting the register handle
        adapter.bus2reg(tr, rw);
        rg = map.get_reg_by_offset(rw.addr, (rw.kind == UVM_READ));

        // Sampling the coverage
        rg.sample_values();
    endfunction
Role Of Register Model Generators

• The sample() and sample_values() method implementations could be done by the register model generators.

• If the generator is unable to do so, the user can write a wrapper script to include the implementations.

• Since the sample() is implicitly called, the user doesn’t have to do anything
Role Of Register Model Generators (contd..)

• However, the sample_values() method has to be called explicitly by the user.

• This is imperative because the place at which to call the sample_values() method is based on the user’s need, hence this cannot be generalized and included by the register generators.
<table>
<thead>
<tr>
<th>sample() method</th>
<th>sample_values() method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protected virtual function</td>
<td>Virtual function</td>
</tr>
<tr>
<td>Cannot be called explicitly</td>
<td>Can be called explicitly</td>
</tr>
<tr>
<td>Called implicitly on every register access</td>
<td>User has to call it explicitly</td>
</tr>
<tr>
<td>Rigid</td>
<td>Convenient and Flexible</td>
</tr>
<tr>
<td>Used in auto-prediction mode</td>
<td>Used in explicit-prediction mode</td>
</tr>
</tbody>
</table>
Results

• Without the implementation of either uvm_reg::sample() or uvm_reg::sample_values() the RAL functional coverage will only be created but not sampled.
Results

• Thus, we need to implement the uvm_reg::sample() for auto-prediction and uvm_reg::sample_values() for explicit-prediction in order to sample the coverage successfully.
Conclusion

• Since the user is oblivious of the 2 obscure methods, `uvm_reg::sample()` and `uvm_reg::sample_values()`, they are rarely used.

• Here, we have shown how to use these methods, along with their implementations, when to use them and their effect on coverage sampling.
References


1) Verification Academy Coverage Cookbook:
   https://verificationacademy.com/cookbook/coverage
Questions?