A Simplified Approach to Generating Functional Coverage

Neil Bulman
Broadcom®

May 22, 2014
SNUG UK

Agenda

Who is Broadcom Cambridge?

The problem.

Our solution.

Conclusion.

Who are Broadcom Cambridge?

- Broadcom Cambridge is an IP and chip-design centre.
- There are approximately 200 employees on site.
- We are part of Broadcom's Mobile Product Solutions business unit.
- We create market-leading video, imaging, and 3D solutions for use in mobile application processors.
- We are the home of the "Raspberry Pi Apps Processor".

The Problem

The Problem

Why bother with coverage?

- How do we know when we've completed verification?
- How do we know what we've accomplished?
  - What's missing?
  - What's the risk?
- When will we be done?

The Problem

Isn't code coverage good enough?

- Simulation:
  - What have we checked?
  - What have we exercised?
  - Was the signal toggled just before the end of test?
  - Did anything useful happen afterwards?
- Functionality:
  - What functionality has been implemented?
  - What functionality is required?
  - What is implemented is not always what is required.

The Problem

Issues with functional coverage.

- Where do I start?
- What's the language syntax?
- How can I refine coverage points?
- What coverage has been implemented?
  - Designers claiming “I don’t understand systemVerilog, how do I know what has been defined?”
  - How can I find all of the coverpoint/covergroup definitions?
- HELP?

Our solution

The Solution

Where did we start?

- Registers:
  - Designs typically include hundreds of registers.
  - They are used for controlling functionality.
  - Registers are usually well documented.
    - Broadcom has an internal format for describing registers. .
    - This is already used for generating a UVM register model.

The Solution

Example register definition.

```
regtype32         Type_OUTPUT_CTRL
   desc           output control register.
      field       reserved                    31:16
      field       MAX_BURST                   15:13 unsigned
         desc     Maximum burst length
                  default  3h
      field       reserved                    12:01
      field       CLEAR                       00:00 unsigned
         desc     This bit clears all states
                  in output block.
         default  0h
endtype

regtype32         Type_OUTPUT_STATUS
   desc           output status register.
      field       reserved                    31:14
      field       DONE                        13:13 unsigned
         desc     When 1 it indicates that there are no
                  more outstanding transactions.
         default  1h
      field       reserved                    12:08
      field       OUTSTANDING_WRITES          07:00 unsigned
         desc     This field shows the number of
                  outstanding write transactions.
         default  0h
endtype

regset            Regs
   private        ENGINEERING_ONLY
   title          Registers for interface block

   Type_OUTPUT_CTRL        OUTPUT_CTRL      +000h RW
      title                  OUTPUT Control

   Type_OUTPUT_STATUS      OUTPUT_STATUS    +004h RO
      title                  OUTPUT Status
endregset


blockdef TOP_BLOCK
   regset Regs        OUTPUT_REGS        +00100000h
endblockdef
```

The Solution

A possibility:

- Doesn’t RALgen already support generating a register model with functional coverage?
  - Yes…

The Solution

A possibility:

- …But
  - It has limitations:
    - What about things that aren't in the register model?
    - How do I control sampling?

      `I only want to sample a single field in a register.`

      `I want to sample all the fields in all the registers within the register model.`
    - Inability to cross fields across registers.

      `Fields aren't always nicely grouped.`

The Solution

The approach:

- Create a class with pointers to all of the fields in the register model.
  - We can now cross fields between any selection of registers.
  - We can group fields in different registers into the same covergroup allowing things to be sampled based on what makes sense from a logical point of view, rather than how they are structured.

- Doesn’t this create more work?
  - We already need to generate a RAL file.
  - The same script can be used to generate a list of pointers.

The Solution

Example wrapper:

```
class output_regs extends uvm_object;

  ral_block_OUTPUT output_reg_mdl;

  uvm_reg_field Regs_OUTPUT_CTRL_MAX_BURST; // #num_bits#3#0#
  uvm_reg_field Regs_OUTPUT_CTRL_CLEAR; // #num_bits#1#0#

  uvm_reg_field OUTPUT_STATUS_ DONE; // #num_bits#1#0#
  uvm_reg_field OUTPUT_STATUS_OUTSTANDING_WRITES; // #num_bits#8#0#

  function new(string name = "output_regs");
    super.new(name);

    if(!uvm_config_db#(ral_block_OUTPUT)::get(uvm_root::get(), "*",
                                          "output_reg_mdl", output_reg_mdl))
      `uvm_fatal(get_type_name(), "reg_model not found")

    //Regs;
    this.Regs_OUTPUT_CTRL_MAX_BURST = output_reg_mdl.Regs_OUTPUT_CTRL.MAX_BURST;
    this.Regs_OUTPUT_CTRL_CLEAR = output_reg_mdl.Regs_OUTPUT_CTRL.CLEAR;

    this.Regs_OUTPUT_STATUS_DONE = output_reg_mdl.Regs_OUTPUT_STATUS.DONE;
    this.Regs_OUTPUT_STATUS_OUTSTANDING_WRITES =
                      output_reg_mdl.Regs_OUTPUT_STATUS.OUTSTANDING_WRITES;
  endfunction : new
endclass : output_regs
```

The Solution

Now what?

- We now have convenient access to all fields.
- We still need to define the desired coverage.
- We still need to be able to review what has been created.
- There are many possibilities for defining coverage.
  - We want to avoid writing coverage directly.

The Solution

Defining:

- Why?
  - It makes reviewing easier.
    - There is a high probability that if things aren't reviewed, then there will be something missing.
    - All definitions can be in one location.
    - Designers do not have the opportunity to become unnerved by phrases such as "UVM" and "systemVerilog".
  - Data entry is simpler.
    - Using a template guides the engineer through what can be done.
    - Templates make it more efficient to focus on the "what", instead of the "how".

The Solution

Example template:

Coverpoint descriptions										
Coverpoint Comments	**Non-default covergroup**	**Label**	**iff**	**Field**	**Cross**	**Bins**	**ignore_bins**	**illegal_bins**	**wildcard_bins**	**at_least**
Have we used different maximum burst lengths?		Max_burst		OUTPUT_CTRL_MAX_BURST		burst1={0}; burst2={1}; burst4={2}; burst8={3}; burst16={4};	too_large = {[5:$]};			10
Have we cleared the output?				OUTPUT_CTRL_CLEAR		cleared={1};	ignore={0};			5
Have we seen done asserted with different burst restrictions?					Max_burst, Output_done	done={1};	ignore={0};			5
Has the block indicated it is done?		Output_done		OUTPUT_STATUS_DONE		done={1}; not_done={1};				5
				OUTPUT_STATUS_OUTSTANDING_WRITES		none={0}; small={[1:5]}; the_rest={[6:$]}				1

The Solution

Why Microsoft® Excel?

- Parsers are commonly available.
- People rarely get scared when presented with an Excel spreadsheet.
- This application is installed on corporate supported Windows computers.

The Solution

Additional details:

- The script that is used for generating the coverage can check that fields being used actually exist.
- This can provide some automation.
  - Allows the option to include all fields.
  - Allows the ability to automatically apply more complicated schemes to binning.
    - You do not have to just rely on auto bins.
- This provides a consistent look.
- The coverage class is an extension of the wrapper class generated previously.

The Solution

What about items that are not in the register model?

- The previous approach can be extended.
  - The same Excel template can be used.
  - The same generation script can be used.
  - Some minor modifications are required.

The Solution

The minor changes:

- How do we integrate the generated coverage into the transaction class?
  - The coverage code is in a new class and is extended from the transaction class.
  - It is easier to track code changes.
    - Changes for coverage can be easily differentiated from other test bench changes.
  - The template requires a minor update to specify the transaction class.

The Solution

From Excel, through Perl, to what?

```
import vip_axi_trans_pkg::vip_axi_trans;

class top_vip_axi_trans_cov extends vip_axi_trans;

  covergroup cg_vals;
    option.per_instance = 1;
    option.at_least = 10;
    PRIVILEGED : coverpoint is_privileged;
  endgroup

  function new(string name = "top_vip_axi_trans_cov");
    cg_vals = new();
  endfunction

  `uvm_object_utils(top_vip_axi_trans_cov)

  function void sample_values(vip_axi_trans axi_trans);
    this.copy(axi_trans);
    if (cg_vals != null) cg_vals.sample();
  endfunction

endclass
```

The Solution

How is it used?

```
`ifndef TOP_COV
`define TOP_COV

//analyis ports to get objects to collect coverage for
// AXI
`uvm_analysis_imp_decl(_axi)

class hevd_top_cov extends uvm_component;

  top_vip_axi_trans_cov       axi_cov;
  hevd_cov                         reg_cov;

  `uvm_object_utils(top_cov)

  uvm_analysis_imp_axi #(vip_axi_trans, top_cov) axi_cov_port;

  function new(string name = "top_cov", uvm_component parent = null);
    super.new(name, parent);

    axi_cov = new("axi_cov");
    axi_cov_port = new("axi_cov_port", this);

    reg_cov = new("reg_cov");
  endfunction: new
```

The Solution

How is it used?

```
function void write_axi(input vip_axi_trans axi_recd);
    axi_cov.sample_values(axi_recd);
  endfunction

  function void sample_reg_cov();
    reg_cov.sample_values();
  endfunction

endclass : top_cov

`endif
```

Conclusion

Conclusion

How did it go?

- Positives:
  - The number of excuses for not reviewing the defined coverage has been minimized.
  - The definition is easier.
    - A spreadsheet provides a gentle reminder.
  - There is less code churn in general test bench code due to coverage refinements.

- Negatives:
  - It is still not as easy as you would hope.
  - Some additional scripting is required.

Conclusion

Possible future steps.

- Integrate the functional coverage definition with the testplan.
- Automatically generate a template with all of the possible fields.

Thank You