FUNCTIONAL COVERAGE OF REGISTER ACCESS VIA SERIAL BUS INTERFACE USING UVM

Darko M. Tomušilovic

Acknowledgement

The author would like to thank the entire **Veriest Vtool Serbia d.o.o.** organization for their support during the implementation of proposed solutions. The code extracts have been generated using the in-house developed **Vtool** tool.

Vtool, thevtool.com

1. REQUIREMENTS

Register space verification and coverage collection

- **Field values**
- **Written and read data**
- **Accessed addresses**
- **Partial and overflow access coverage (bit-level access not natively supported by the UVM documentation!)**
- **Low-level communication coverage (frequency, …)**
- **Power management scenarios (the registers whose power is shut off act as read only; a read attempt results in read value isolation)**
- **Register interaction scenarios (consecutive access to various addresses, …)**

2. UVM_REG COVERAGE API

A) FUNCTIONAL COVERAGE TYPE IDENTIFIERS

No implicit coverage provided

UVM proposes usage of functional coverage type identifiers, in order to determine whether certain covergroups are to be instantiated or not

UVM_NO_COVERAGE	No covergroups built
UVM_CVR_REG_BITS	Read and written data covergroups built
UVM_CVR_ADDR_MAP	Accessed addresses covergroups built
UVM_CVR_FIELD_VALS	Field values covergroups built
UVM_CVR_ALL	All covergroups built

To include coverage models: uvm_reg::include_coverage("*", UVM_CVR_REG_BITS
+ UVM_CVR_FIELD_VALS
+ UVM_CVR_ADDR_MAP);

2. UVM_REG COVERAGE API

B) REGISTER LEVEL COVERGROUPS INSTANTIATION AND DEFINITION

```
class dvcon_reg1 extends uvm_reg;
  `uvm_object_utils(dvcon_reg1)

  uvm_reg_field dvcon_field1;
  uvm_reg_field dvcon_field2;

  function new (string name = "dvcon_reg1");
    super.new(name, 8, build_coverage(UVM_CVR_REG_BITS +
                                      UVM_CVR_FIELD_VALS));

    if (has_coverage(UVM_CVR_REG_BITS))
      cg_bits = new();
    if (has_coverage(UVM_CVR_FIELD_VALS))
      cg_vals = new();
  endfunction

  ...
```

```
covergroup cg_bits with function sample(uvm_reg_data_t data,
                                        bit            is_read);
  DATA_0: coverpoint data[0];
  ...
  RW    : coverpoint is_read;
  //cross coverage
endgroup

covergroup cg_vals;
  coverpoint dvcon_field1.get_mirrored_value()
  {
    bins ALL[] = {[0:15]};
  }
  coverpoint dvcon_field2.get_mirrored_value()
  {
    bins ZERO  = {0};
    bins OTHER = {[1:15]};
  }
endgroup
```

2. UVM_REG COVERAGE API

C) REGISTER BLOCK LEVEL COVERGROUPS INSTANTIATION AND DEFINITION

```
class dvcon_block extends uvm_reg_block;
  `uvm_object_utils(dvcon_block)

  dvcon_reg1 my_reg1;
  dvcon_reg2 my_reg2;

  function new(string name = "dvcon_block");
    super.new(name, build_coverage(UVM_CVR_ADDR_MAP +
                                   UVM_CVR_FIELD_VALS));

    if (has_coverage(UVM_CVR_ADDR_MAP))
      cg_addr = new();
    if (has_coverage(UVM_CVR_FIELD_VALS))
      cg_vals = new();
  endfunction

  ...
```

```
covergroup cg_addr with function sample (uvm_reg_addr_t offset,
                                         bit            is_read);
  OFFSET: coverpoint offset
  {
    bins REG1={`REG1_O};
    bins REG2={`REG2_O};
  }
  RW:     coverpoint is_read;
  OFFSET_x_RW: cross OFFSET, RW;
endgroup

covergroup cg_vals;
  R1F1: coverpoint my_reg1.dvcon_field1.get_mirrored_value()
  {
    bins ZERO={0};
    bins ONE ={1};
  }
  R2F2: coverpoint my_reg2.dvcon_field2.get_mirrored_value()
  {
    bins ZERO={0};
    bins TWO ={2};
  }
  CROSS: cross R1F1, R2F2;
endgroup
```

2. UVM_REG COVERAGE API

D) SAMPLING

By default, the sampling of all covergroups in the register model should be disabled

To enable the sampling, the set_coverage method is used

void'(dvcon_rm.set_coverage (UVM_CVR_REG_BITS +
UVM_CVR_FIELD_VALS +
UVM_CVR_ADDR_MAP));

Register level coverage sampling

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

  if (get_coverage(UVM_CVR_REG_BITS))
    cg_bits.sample(data, is_read);
endfunction

virtual function void sample_values();

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

Register block level coverage sampling

```
protected virtual function void sample(uvm_reg_addr_t offset,
                                       bit            is_read,
                                       uvm_reg_map    map);

  if (get_coverage(UVM_CVR_ADDR_MAP))
    cg_addr.sample(offset, is_read);
endfunction

virtual function void sample_values();

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

2. UVM_REG COVERAGE API

E) DRAWBACKS

By following the proposed guidelines, the coverage of some simple items can be successfully performed

However, the usage of UVM_REG Coverage API tends to be very error-prone

Typical mistakes and drawbacks include:

- Using value of uvm_reg_field class in place of written or read data
- Using value of uvm_reg_field class in place of the mirrored value
- Failing to understand the order of predictor operation – sampling occurs before the prediction
- Failing to understand the meaning of API methods – the role of include_coverage, build_coverage, has_coverage, set_coverage, get_coverage can be confusing
- Forgetting to enable sampling
- Only partially following the guidelines (for example, the sampling is done unconditionally)
- Failing to understand the usage model of sample and sample_values methods – sample_values is not called automatically by the predictor
- Providing references to the rest of the environment in a register, affecting reusability
- Covergroups defined within a register class reduce code readability
- Any scenario involving consecutive accesses to various addresses, the remaining transaction fields, the nonregister content creates an undesired dependency between the register model and the rest of the testbench

3. EXTERNAL FUNCTIONAL COVERAGE SUBSCRIBER

A) STRUCTURE

Having all the limitations of UVM_REG Coverage API in mind, it turns out that the usage of the **External Functional Coverage Subscriber** is a much more convenient solution

Very advantageous in the case that a serial bus interface is used for register access

```
class dvcon_reg_cov_subscriber extends uvm_subscriber #(dvcon_spi_item);
  `uvm_component_utils(dvcon_reg_cov_subscriber)

  dvcon_reg_model        dvcon_rm;
  dvcon_cfg                   cfg;
  dvcon_vals_wrapper  vals_wrapper;
  ...

  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);

    if (cfg.build_vals)
    begin
      vals_wrapper = dvcon_vals_wrapper::type_id::create("vals_wrapper");
      vals_wrapper.dvcon_rm = dvcon_rm;
    end
    ...
  endfunction

  virtual function void write(dvcon_spi_item t);
    // sample
  endfunction
  ...
```

3. EXTERNAL FUNCTIONAL COVERAGE SUBSCRIBER

B) COVERGROUP WRAPPER

To support covergroup creation on demand, all implemented covergroups are wrapped within uvm_object

```
class dvcon_vals_wrapper extends uvm_object;
  `uvm_object_utils(dvcon_vals_wrapper)

  dvcon_reg_model dvcon_rm;

  covergroup cg_vals;
    option.per_instance = 1;

    R1F1: coverpoint dvcon_rm.my_block.my_reg1.dvcon_field1.get_mirrored_value();
  endgroup

  function new (string name="dvcon_vals_wrapper");
    super.new(name);
    cg_vals = new();
  endfunction

  virtual function void sample();
    cg_vals.sample();
  endfunction
endclass
```

Register access coverage

```
covergroup cg_access with function sample (uvm_reg_addr_t addr,
                                           uvm_reg_data_t data,
                                           bit            is_read);
  option.per_instance = 1;

  ADDR: coverpoint addr; // bins
  DATA: coverpoint data; // bins
  RW  : coverpoint is_read;
  // mirrored value
  // cross coverage
endgroup
```

3. EXTERNAL FUNCTIONAL COVERAGE SUBSCRIBER

C) PARTIAL ACCESS, TRANSITION, SPI CLOCK FREQUENCY COVERAGE

```
class dvcon_partial_access_wrapper extends uvm_object;
  `uvm_object_utils(dvcon_partial_access_wrapper)

  covergroup cg_partial_access with function sample (uvm_reg_addr_t addr,
                                                     int            length,
                                                     bit           is_read);
    option.per_instance = 1;

    ADDR:   coverpoint addr;   // bins
    LENGTH: coverpoint length; // bins
    RW:     coverpoint is_read;
    CROSS:  cross ADDR, LENGTH, RW;
endgroup

  function new (string name="dvcon_partial_access_wrapper");
    super.new(name);
    cg_partial_access = new();
  endfunction

  virtual function void sample(uvm_reg_addr_t addr,
                               int            length,
                               bit           is_read);

    cg_partial_access.sample(addr, length, is_read);
  endfunction
endclass
```

```
covergroup transition with function sample (uvm_reg_addr_t addr,
                                            rw_e         is_read);
  option.per_instance = 1;

  ADDR: coverpoint addr
  {
    bins TRAN_12=(`REG1_O => `REG2_O);
  }
  RW  : coverpoint is_read
  {
    bins TRAN_RW[]=(READ, WRITE => READ, WRITE);
  }
  ADDR_x_RW: cross ADDR, RW;
endgroup
```

```
covergroup cg_frequency with function sample (uvm_reg_addr_t addr,
                                              real             freq,
                                              bit           is_read);
  option.per_instance = 1;

  ADDR:   coverpoint addr; // bins
  FREQ:   coverpoint freq; // bins
  RW:     coverpoint is_read;
  CROSS:  cross ADDR, FREQ, RW;
endgroup
```

3. EXTERNAL FUNCTIONAL COVERAGE SUBSCRIBER

D) POWER SUPPLY MODELING CALLBACK, POWER SUPPLY COVERAGE

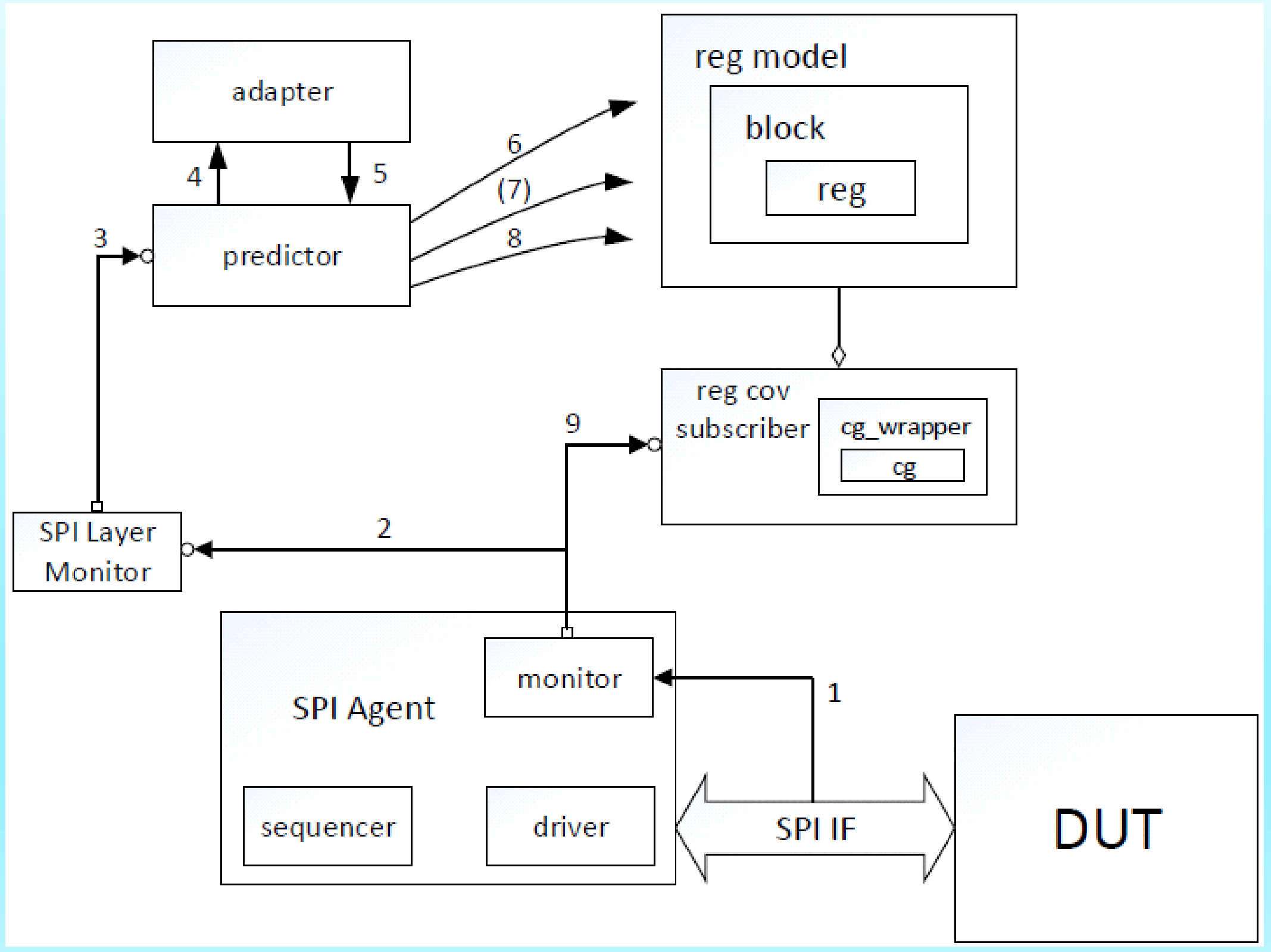
A locking field callback technique is utilized to prevent access to registers within power domains that are turned off

```
covergroup cg_power with function sample (uvm_reg_addr_t addr,
                                          power_e        power,
                                          bit            is_read);
  option.per_instance = 1;

  ADDR :  coverpoint addr;  // bins
  POWER:  coverpoint power; // bins
  RW   :  coverpoint is_read;
  CROSS:  cross ADDR, POWER, RW;
endgroup
```

```
class dvcon_power_callback extends uvm_reg_cbs;
  `uvm_object_utils(dvcon_power_callback)

  function new(string name="dvcon_power_callback");
    super.new(name);
  endfunction

  virtual function void post_predict(input uvm_reg_field  fld,
                                     input uvm_reg_data_t previous,
                                     inout uvm_reg_data_t value,
                                     input uvm_predict_e  kind,
                                     input uvm_path_e     path,
                                     input uvm_reg_map    map);
    dvcon_power_reg temp_reg;
    temp_reg = dvcon_power_reg'(fld.get_parent());

    if (temp_reg.POWER == OFF)
      value = fld.get_reset();
  endfunction
endclass

//  The callback needs to be added to a register field
//  using following piece of code in uvm_reg::build method
    dvcon_power_callback dvcon_power_cb=new();
    dvcon_power_cb.set_name("dvcon_power_cb");
    uvm_reg_field_cb::add(field, dvcon_power_cb);
```

3. EXTERNAL FUNCTIONAL COVERAGE SUBSCRIBER

E) COVERAGE COLLECTION DIAGRAM



THANK YOU!