Simpler Register Model

Sanjeev Singh
Verification Engineer
@Juniper Networks
Background

• A register model is a model of the software visible registers and memories in the design.
  – Most test benches need one.

• UVM system verilog package is the de facto standard for test benches.
  – Uvm_reg provides the base classes for creating a register model.
Problems With UVM_REG

- Large Memory Footprint.
  - Static Allocation of memory for all locations.

- Randomization fails for Large Tables.
  - Implements tables as static array of registers.

- API is confusing to use.
  - Mirror, Update, Predict.

- Hard to Understand source code. (22K Lines in 26 files).
Simpler Register Model

• A open source register model package for replacing uvm_reg in uvm test benches.

• Source code available under MIT license on github. https://github.com/Juniper/simple_reg_model

• Demo code available under MIT license on github. https://github.com/sanjeevs/srm_sap
SRM Goals

• Scalable to large system level test benches.
  – Memory efficient.
  – Scalable Randomization.

• Simple & Concise API.
  – Single value stored for each field.

• Reusable Sequences.
  – Reusable across multiple hierarchy.
  – Independent of access mechanism (backdoor/frontdoor/etc).
SRM Based Test bench

Pio Sequence

SRM Reg Model

Host Adapter

Pio Adapter

Backdoor Adapter

UVM Test

UVM Env

Host Agent

Pio Agent

Host block

DUT

BlockA

r1
t1

Pio_bus
SRM Base Classes

- Two basic data types.
  - `srm_register`: a single register.
  - `srm_table`: array of registers.

- Access parameters.
  - `srm_base_handle`:
    - Selects the correct adapter to use.
    - Configuration options.
typedef struct packed {
  reg[31:0] field0;
} r1_struct_t;

class r1_reg extends srm_reg#(r1_struct_t);
  srm_field#(bit[31:0]) field0;

  function new(string name, srm_component parent);
  ....
  endfunction
endclass
class r1_table extends srm_table#(r1_struct_t);
    // A entry of the table
    class r1_table_entry extends srm_table_entry#(r1_struct_t);
        ....
    endclass

    function new(string name, ...);
        r1_table_entry entry;
        super.new(.name()...);
        entry = new(..);
        _prototype = entry;
    endfunction
endclass

Table is a template class on the struct

A table entry is dynamically allocated on writes.

Prototype Design Pattern
class host_handle extends srm_base_handle;
    function new(...);
        skip_read_error_msgs = 0;
    .
    endfunction

    virtual function srm_bus_adapter get_adapter(srm_node obj);
    .
    .
    endfunction
endclass

Configuration Options
Logic to select adapter
Walk from the current node to the root looking for the correct adapter to use.
## Register Access API

<table>
<thead>
<tr>
<th>Task Access</th>
<th>Effect On DUT</th>
<th>Effect On SRM Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>model.r1.write(handle, 32'h0)</code></td>
<td>Writes the value to DUT</td>
<td>Updates the model.</td>
</tr>
<tr>
<td><code>model.r1.read(handle, rd_data);</code></td>
<td>Reads the value from DUT</td>
<td>Checks that the non volatile fields match and updates the volatile fields.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Access</th>
<th>Effect On DUT</th>
<th>Effect On SRM Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>model.r1.set(32'h0)</code></td>
<td>None</td>
<td>Updates the model.</td>
</tr>
<tr>
<td><code>model.r1.get();</code></td>
<td>None</td>
<td>Read from the model.</td>
</tr>
</tbody>
</table>
Table Access API

• A table is an array of entries of type srm_register.

• Use ‘entry_at(index)’ to get the table entry.

```c
model.t1.entry_at(100).write(handle, 32’h0);
```

Write to table at offset 100.

```c
model.t1.entry_at(100).read(handle, rd_data);
```

Read from table at offset 100.
Register Randomization

```verilog
class r1_constr extends uvm_object;
    rand bit [31:0] field0;
    function r1_struct_t get_data();
    endclass

    r1_struct_t wr_data;
    r1_constr c1 = r1_constr::type_id::create("r1_constr");
    c1.randomize();
    wr_data = c1.get_data();
    regmodel.r1.write(handle, wr_data);
```

Auto generated constraint class.
Convert to packed struct.
Generate random value
Write the random value
r1_struct_t wr_data;
r1_constr c1 = r1_constr::type_id::create("r1_constr");

for(int i = 0; i < depth_of_table; i++) begin
  c1.randomize();
  wr_data = c1.get_data();
  model.t1_table.entry_at(i).write(handle, wr_data);
end

Generate random value for each entry in table
Write the random value
Scalable Randomization
task block_test::run_phase();

...  
blockA_cfg_sequence.regmodel = blockA.r1;
blockA_cfg_sequence.start(null);
...
endtask

Block Level Test

task chip_test::run_phase();

...
blockA_cfg_sequence.regmodel = chip.blockA.r1;
blockA_cfg_sequence.start(null);
...
endtask

Chip Level Test
class base_test;
    frontdoor_handle f_handle;  // Handle for each adapter
    backdoor_handle b_handle;
    block_regmodel regmodel;

    task run_phase(..);
        reg_sequence.initialize(regmodel, f_handle);
        reg_sequence.start(null);

        reg_sequence.initialize(regmodel, b_handle);
        reg_sequence.start(null);

    endtask
endclass
Other Features

• Callbacks to model special types of registers.

• Functional coverage.

• Introspection API to write generic sequences.

• Composite API.
  – model.load(), model.store(), model.store_update()
Register Model Generation

• System RDL.
  – Use open source tool https://github.com/Juniper/open-register-design-tool
  – Developed at Juniper Networks.

• DSL written in Ruby. https://github.com/sanjeevs/srm
Simple As Possible Demo

- https://github.com/sanjeevs/srm_sap
Acknowledgements

• Thanks to my employer “Juniper Network” for releasing the code under MIT license.

• Special thanks.
  – Scott Nellenbach for open source tool.
    • https://github.com/Juniper/open-register-design-tool
  – David Meador for valuable feedback.
Thanks

Please contact me if you would like to contribute.

sanjeevs@juniper.net
Limitations of SRM

• API is not compatible with uvm_reg.
  – Use for new test benches.

• New
  – Yet to be proven in a real chip tape out.

• No commercial register generation tool support.
  – Open Source Tool Generation in works.
  – Support for Python, Ruby scripting.