Mutable Verification environments through Visitor and Dynamic Register map Configuration

Matteo Barbati, STMicroelectronics
Alberto Allara, STMicroelectronics
Outline

• Motivation
• Register Model Synch: State of the Art
• Optimized Register Map Dynamic Configuration
• Common VIP Extension Flow
• Verification Component Extension
• Use Case
• Conclusion
Motivation

• Mutable Verification Environment: a verification environment that is able to change dynamically the self-checking capabilities according to the DUT configuration

• Simplify generation of Mutable Verification Environments
  – Optimize the Synchronization of UVM Register Models with any Verification Component in the environment
  – Provide a mechanism to easily add new capabilities to existing Verification Components
## Register Model Synch: State of the Art

<table>
<thead>
<tr>
<th>Pros</th>
<th>Custom Code</th>
<th>IP-XACT flow</th>
<th>UVM Register Map Dynamic Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Code optimized: events generated only where needed</td>
<td>• Code optimized: events generated only where needed</td>
<td>• More flexibility: Event generation can be dynamically configured</td>
<td></td>
</tr>
<tr>
<td>• More flexibility: Event generation can be dynamically configured</td>
<td></td>
<td>• Reduce code impact in case of customization needed on high number of registers</td>
<td></td>
</tr>
<tr>
<td>• Reduce code impact in case of customization needed on high number of registers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Different version of the same IP-XACT description. One for Design and one for Verification</td>
<td></td>
<td>• Size Overhead affecting all the fields of the entire register map</td>
<td></td>
</tr>
<tr>
<td>• Error prone in case of high number of registers to customize</td>
<td>• Error prone in case of high number of registers to customize</td>
<td>• 9 extra variables</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Need to rerun the entire flow in case of changes in register customization</td>
<td>• 4 extra uvm events</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Performance Impact</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Events associated to fields and not to regs</td>
<td></td>
</tr>
</tbody>
</table>

• Size Overhead affecting all the fields of the entire register map
• 9 extra variables
• 4 extra uvm events
• Performance Impact
• Events associated to fields and not to regs

© Accellera Systems Initiative 4
Define a dedicated structure to store synch event enables and the register callback extension.

```c
typedef struct { bit pre_rd, pre_wr, post_rd, post_wr; } field_desc;
...
class uvm_reg_trig_cbs extends uvm_reg_cbs;
`uvm_object_utils(uvm_reg_trig_cbs)
field_desc fld_list [string];
uvm_event_pool ep;
function new(string name = "uvm_reg_trig_cbs");
super.new(name);
ep = uvm_event_pool::get_global_pool();
endfunction
...
virtual task post_write(uvm_reg_item rw);
...
$cast(fld, rw.element);
d = event_data::type_id::create("d");
fld_d = fld_list[fld.get_name()];
if(fld_d.post_wr) begin
    event_name = {"post_wr_", fld.get_name());
    trigger = ep.get(event_name);
    d.val= fld.get();
    `uvm_info(get_name(), "**** Firing Post Write trigger ****", UVM_NONE)
    trigger.trigger(d);
end
endtask : post_write
endclass
Optimized Register Map Dynamic Configuration (III)

• Instantiate and connect reg callbacks in Verif. Environment

```java
class top_sve extends uvm_env;
...
virtual function void connect_register_callbacks();
...
cbs = uvm_reg_trig_cbs::type_id::create("cbs");
cbs.fld_list = field_list;
foreach(field_list[key]) begin
  ...
  f = regmodel[i].get_field_by_name(key);
  if(f != null)
    uvm_callbacks#(uvm_reg_field, uvm_reg_trig_cbs)::add(f, cbs);
  ...
end
endfunction : connect_register_callbacks
```
... virtual function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
...
connect_register_callbacks();
endfunction : connect_phase
class : top_sve

• Configure synch event generation

//field name, pre_rd, pre_wr, post_rd, post_wr
field1 0 1 0 1
field2 1 0 1 0
field3 0 0 1 1
Common VIP Extension Flow

• Verification Component capabilities can be extended as follow:
  – Create an extension class of the Verification Component with the new capabilities
  – Using UVM override mechanism replace the original version of the Verification Component with the new one

• Cons: Significant overhead in term of lines of code -> error prone
Verification Component Extension (I)
Verification Component Extension (II)

- Define UVM Visitor to manage Synch Event from Register Models

```cpp
class event_manager_visitor extends uvm_visitor;
virtual function void visit(uvm_component node);
if (node.get_object_type() == vip_monitor::type_id::get()) begin
    fork
        manage_field1_pre_wr(node);
        manage_field1_post_wr(node);
    ...
end
join_none
end
... endfunction
```
Verification Component Extension (III)

...  
virtual task manage_field1_pre_wr(uvm_component node);  
...  
event_name = "pre_wr_field1;  
$cast(mon, node);  
...  
forever begin  
  ev.wait_trigger_data(obj);  
  $cast(d, obj);  
  mon.vif.sig1 = d.val;  
end  
endtask  
...  
endclass
Verification Component Extension (IV)

• Define the UVM Visitor Adapter

class basic_adapter extends uvm_visitor_adapter;
virtual function void accept(uvm_component s, uvm_visitor v, uvm_structure_proxy#(uvm_component) p, bit invoke_begin_end=1'b1);
if(invoke_begin_end)
v.begin_v();
v.visit(s);
if(invoke_begin_end)
v.end_v();
endfunction
function new (string name = "");
super.new(name);
endfunction
endclass
Verification Component Extension (V)

• Instantiate and connect adapter and visitor in Verif. Environment

```verbatim
class top_sve extends uvm_env;
...
virtual function void start_of_simulation_phase(uvm_phase phase);
  event_manager_visitor event_manager_v;
  uvm_top_down_visitor_adapter adapter;
  uvm_component_proxy proxy;
  event_manager_v = new("event_manager_v");
  adapter = new("adapter");
  proxy = new("proxy");
  adapter.accept(this, event_manager_v, proxy);
endfunction : start_of_simulation_phase
endclass: top_sve
```
Use Case (I)

• DUT: SerDes IP
• Legacy Verification Component used to measure injected ppm
  – PRE: Enable/Disable Verification Component explicitly using “enable” signal in SV Interface
  – POST: Enable/Disable features according to configurations on DUT
Use Case (II)

Previous Solution

```verilog
regmap_write_field("en_count_tx",1'b1);
// Enable PPM estimator
force tb_top.ppm_tx.en = 1'b1;
#(100us);
// Disable PPM estimator
force tb_top.ppm_tx.en = 1'b0;
regmap_write_field("en_count_tx",1'b0);
#(1us);
current_ppm = tb_top.ppm_tx.actual_ppm;
```

Current Solution

```verilog
// Automatically Enable PPM estimator with write at 1 on en_count_tx
regmap_write_field("en_count_tx1",1'b1);
#(100us);
// Automatically Disable PPM estimator with write at 0 on en_count_tx
regmap_write_field("en_count_tx",1'b0);
#(1us);
current_ppm = tb_top.ppm_tx.actual_ppm;
```
Conclusion

• Simplified approach to create Mutable Verification Environment
• Optimized Register Map Dynamic Configuration
  – Reduced Data Overhead
  – Reduced number of Synch Events.
• Extension through UVM Visitor -> reduced error prone approach
Questions