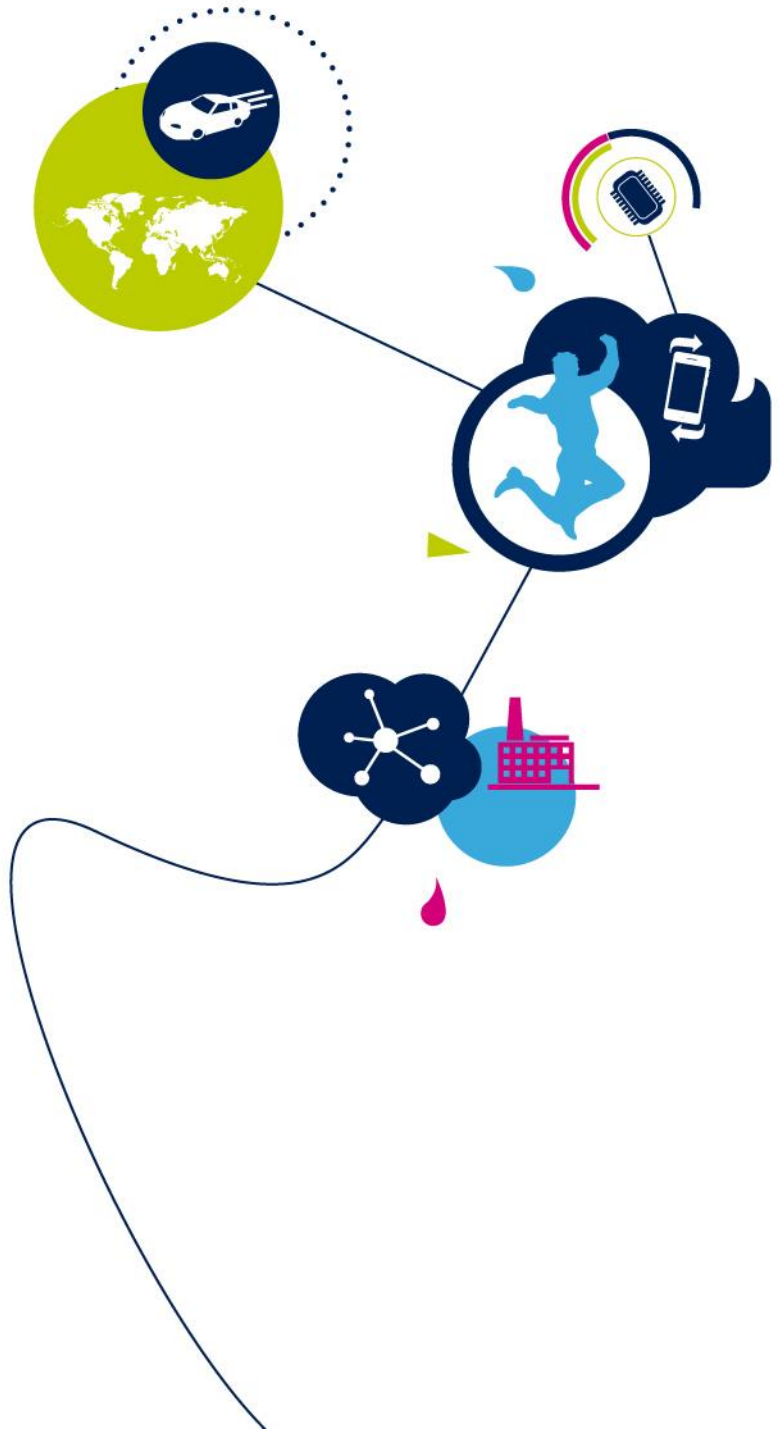
UVM Register Map Dynamic Configuration

Matteo Barbati, STMicroelectronics

Alberto Allara, STMicroelectronics



life.augmented

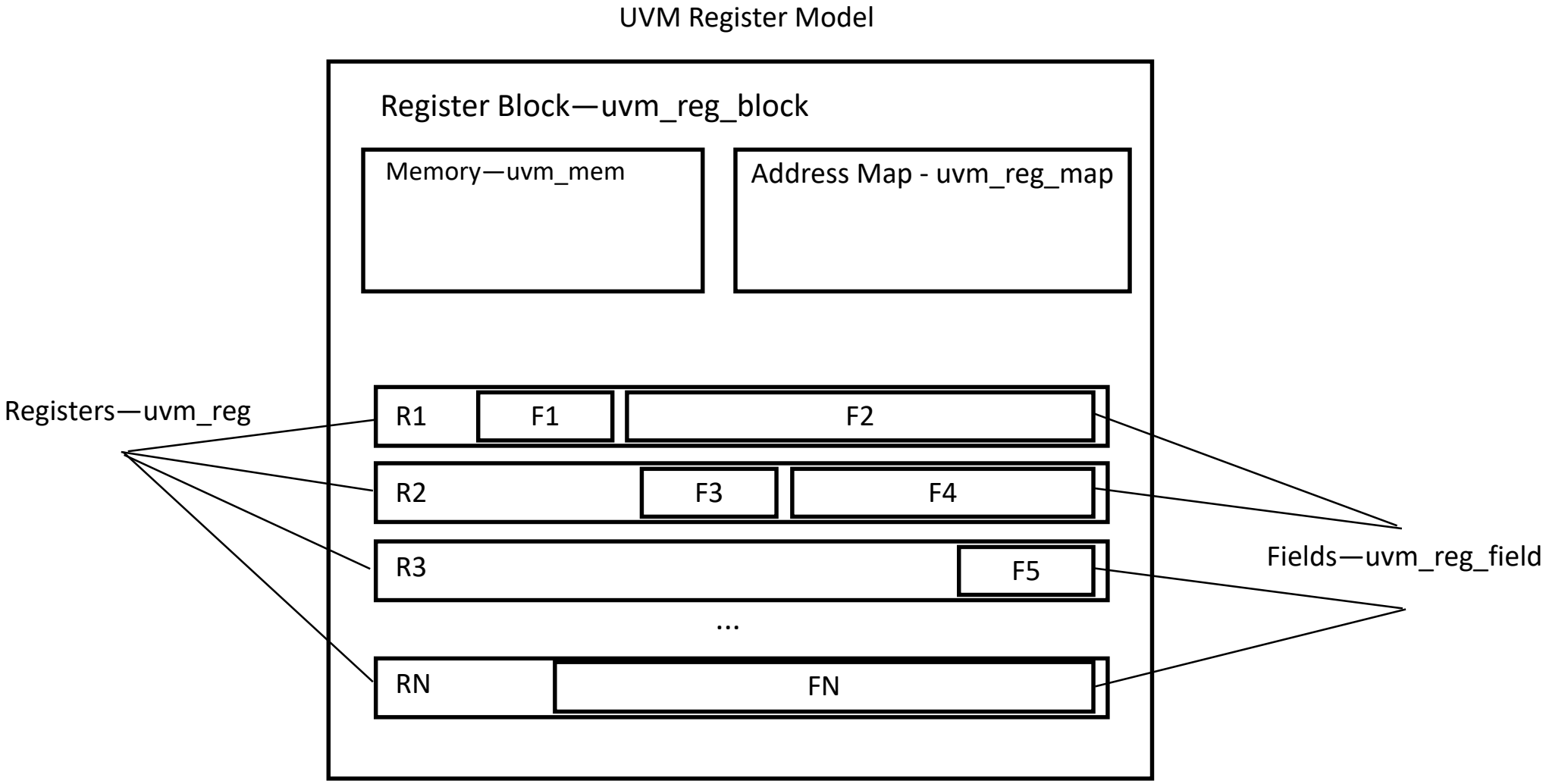
Outline

- Motivation
- UVM Register Model
- Case Study
- Custom Code Example
- IP-XACT Example
- Proposed Approach
- Real Case Example
- PROs and CONs
- Conclusion & Future Works

Motivation

- Automatize the synchronization of the UVM register model with any verification component in the environment
- Reduce the usage of ad-hoc solutions that are error prone

UVM Register Model



Case Study

- Scenario
  - Create a verification environment that dynamically change behavior according to the configuration of a register.
    - For instance consider register called “MODE”
  - Generate an event every time the fields of our case study register are written

- State of the Art
  - Custom Code
  - IP-XACT

Custom Code Example

- Extend “MODE_type” class
  - Adding logic to correctly manage UVM events

```
class trig_MODE_type extends MODE_type;
  ...
  uvm_event_pool ep;
  uvm_event e;
  string event_name = "mode_type_e";
...
```

Custom Code Example (2)

- Generating event at each write operation

```
virtual task post_write(uvm_reg_item          rw);
  `uvm_info("trig_MODE_type", "**** Firing Write trigger ****", UVM_HIGH);
  e.trigger();
 endtask: post_write
Endclass
```

- Overwrite the original class in the environment

```
…
set_type_override_by_type(MODE_type::get_type(),trig_MODE_type::get_type());
…
```

IP-XACT example

- Customize IP-XACT regmap description

```
<vendorExtensions:description>My post_write </vendorExtensions:description>
<vendorExtensions:code>
        virtual task post_write(uvm_reg_item rw);
                uvm_event_pool ep;
                uvm_event e;
                `uvm_info("trig_MODE_type","**Firing Write trigger**",UVM_HIGH);
                ep=uvm_event_pool::get_global_pool();
                e  = ep.get("mode_type_e");
                e.trigger();
        endtask: post_write
</vendorExtensions:code>
```

IP-XACT Example (2)

- Run the UVM register map generation flow

```
...
virtual task post_write(uvm_reg_item      rw);
        uvm_event_pool ep;
        uvm_event e;
        `uvm_info("trig_MODE_type", "** Firing Write trigger **", UVM_HIGH);
        ep = uvm_event_pool::get_global_pool();
        e  = ep.get(event_name);
        e.trigger();
endtask: post_write
…
```

Proposed Approach

- Overview
  - Modify uvm_reg_field adding logic to generate triggers
  - Substitute uvm_reg_field with the new one

- Implementation
  - Modify pre_read and pre_write to manage configuration through uvm_config_db
  - Add a lazy initialization mechanism
  - Modify pre_read, pre_write, post_read and post_write to trigger events

Proposed Approach (2)

- New uvm_reg_field class
  - Add required uvm events and support variables

```
uvm_event_pool ep;
uvm_event pre_rd_e, pre_wr_e, post_rd_e, post_wr_e;
local string pre_rd_event_name; local string pre_wr_event_name;
local string post_rd_event_name; local string post_wr_event_name;
local bit en_pre_rd_event, en_pre_wr_event, en_post_rd_event, en_post_wr_event;
…
local bit conf;
```

Proposed Approach (3)

- New uvm_reg_field class
  - Implement configuration mechanism

```
virtual task pre_read(uvm_reg_item   rw);
  if(conf) begin
    uvm_config_db#(bit)::get(uvm_root::get(), get_full_name(), "en_pre_rd_event", en_pre_rd_event);
    uvm_config_db#(bit)::get(uvm_root::get(), get_full_name(), "en_pre_wr_event", en_pre_wr_event);
    uvm_config_db#(bit)::get(uvm_root::get(), get_full_name(), "en_post_rd_event", en_post_rd_event);
    uvm_config_db#(bit)::get(uvm_root::get(), get_full_name(), "en_post_wr_event", en_post_wr_event);
    conf = 1'b0;
  end
  …
endtask : pre_read
```

Proposed Approach (4)

- New uvm_reg_field class
  - Customize callbacks to generate event if required

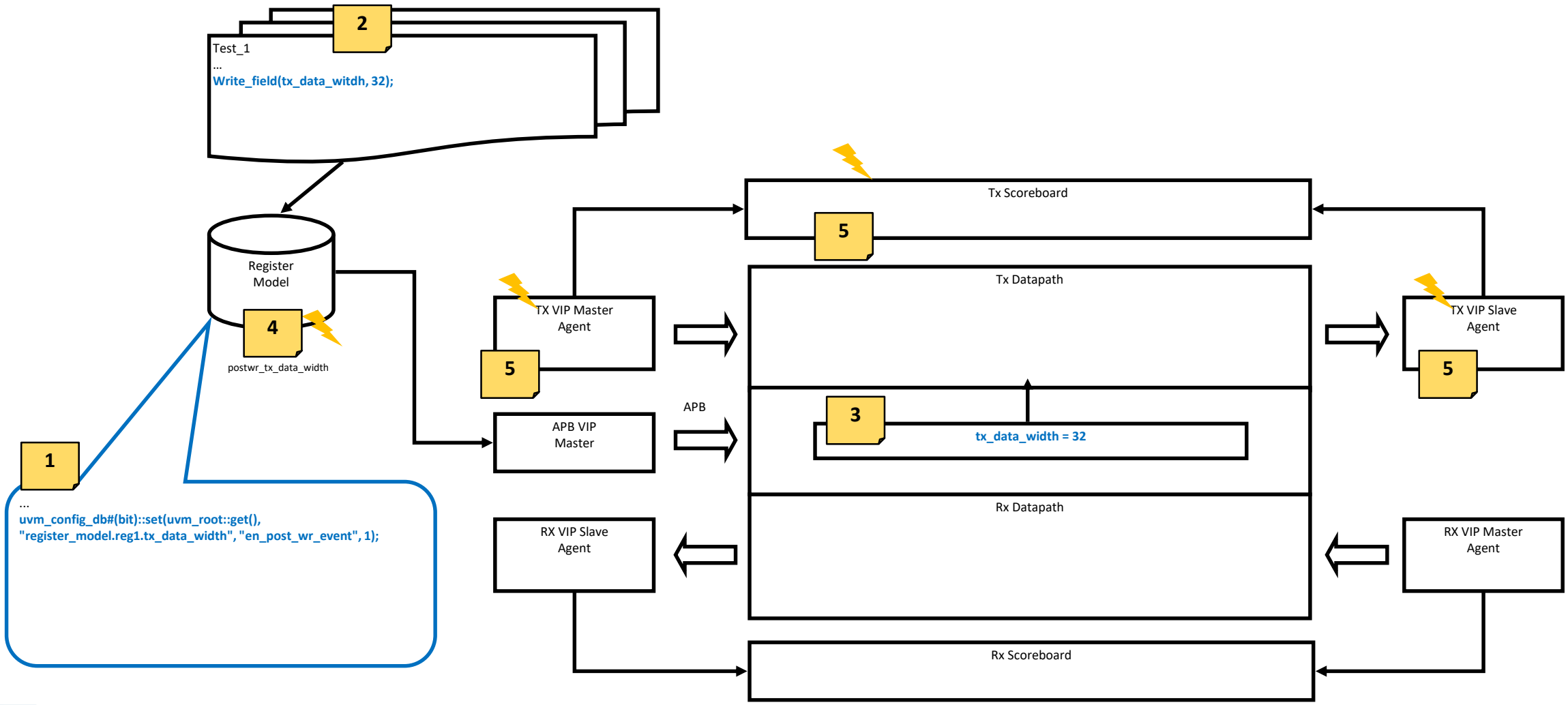
```
virtual task pre_read(uvm_reg_item   rw);
  …
  if(en_pre_rd_event) begin
    `uvm_info(get_name(), "**** Firing Pre Read trigger ****", UVM_HIGH)
    pre_rd_e.trigger();
  end
endtask : pre_read

virtual task post_read(uvm_reg_item  rw);
  if(en_post_rd_event) begin
    `uvm_info(get_name(), "**** Firing Post Read trigger ****", UVM_HIGH)
    post_rd_e.trigger();
  end
endtask : post_read
```

Proposed Approach (5)

- Replace uvm_reg_field class with the new class in the environment
- Turn on event generation for the desired fields

```
…
set_type_override_by_type(uvm_reg_field::get_type(),uvm_reg_field_ext::get_type());
uvm_config_db#(bit)::set(uvm_root::get(), "regmap.MODE.field1", "en_post_wr_event", 1);
uvm_config_db#(bit)::set(uvm_root::get(), "regmap.MODE.field3", "en_post_wr_event", 1);
uvm_config_db#(bit)::set(uvm_root::get(), "regmap.MODE.field4", "en_post_wr_event", 1);
```

Real Case Example



PROs & CONs

	Custom Code	IP-XACT flow	Proposed Approach
Pros	• Code optimized: events generated only where needed	• Code optimized: events generated only where needed	• More flexibility: Event generation can be dynamically configured • Reduce code impact in case of customization needed on high number of registers

Cons	• Error prone in case of high number of registers to customize	• Error prone in case of high number of registers to customize
		• Need to rerun the entire flow in case of changes in register customization
		• Different version of the same IP-XACT description. One for Design and one for Verification
			• Size Overhead affecting all the fields of the entire register map
			  • 9 extra variables
			  • 4 extra uvm events

Conclusions & Future Works

- Reduced error prone approach

- To reduce impact related to Data Size Overhead
  - “cluster of events”