Extending the UVM register model generation and integration flow to support user-defined scenarios and register mask values

Shuang Han
Kees van Kaam
Martin Barnasconi

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Outline

- Motivation
- Register Verification Requirements
- Proposed Methodology and Flow
- Flow Implementation
- Results
- Conclusion

Motivation

- Register Verification Challenges with UVM
  - UVM register model generated automatically, field access policy is fixed
  - Different user scenarios will lead to different register field access policies
  - How to generate extra access policies
  - How to apply the generated extra access policies in UVM environment

- This work focuses on
  - Methodology to extend UVM register model generation and integration flow
  - Flow implementation with clear steps to address the challenges

Register Verification Requirements

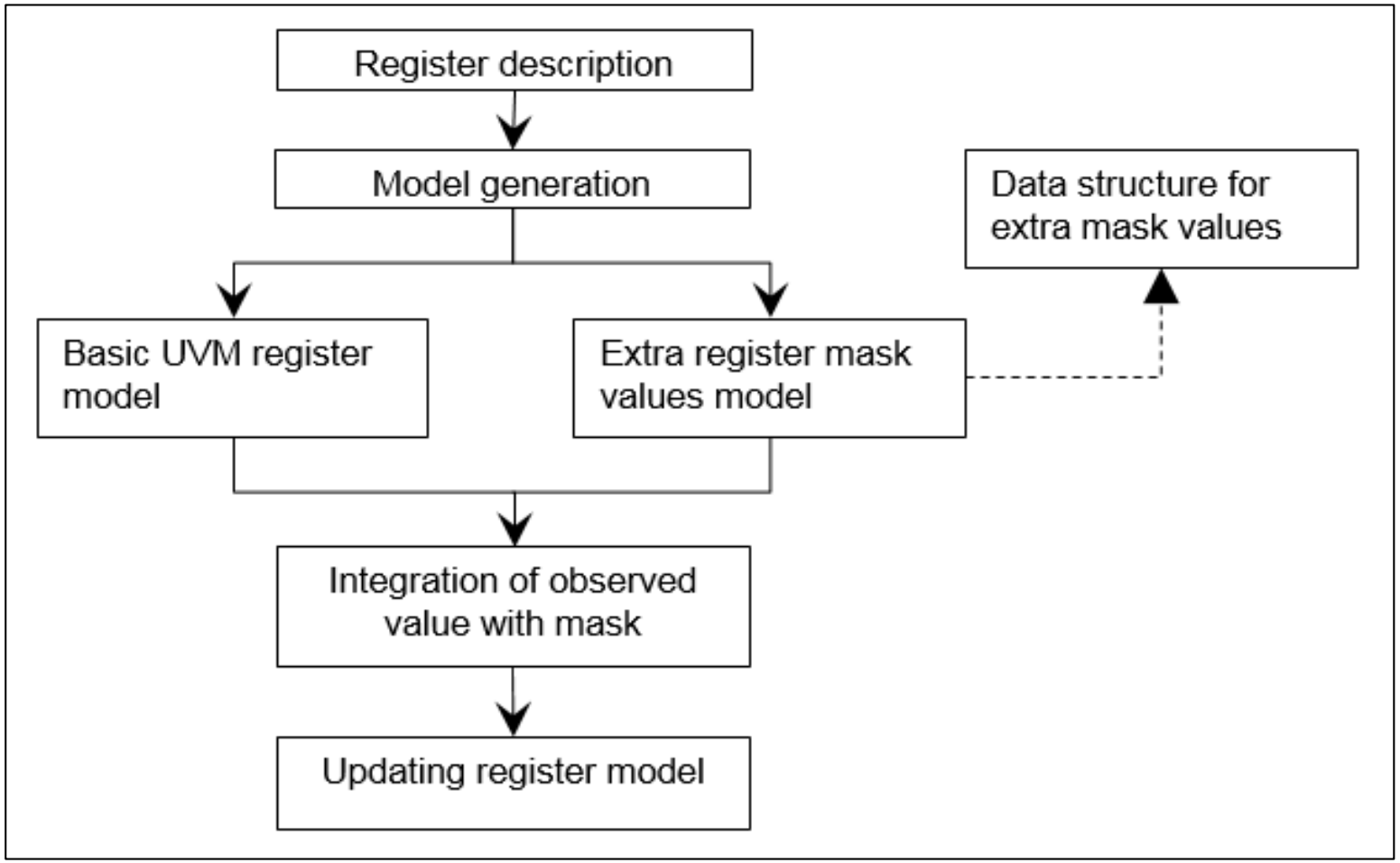
- General requirements
  - Generated register model should reflect all design registers, should be structured in the same hierarchy as design blocks
  - Should support both frontdoor and backdoor access
  - Should be able to check register behavior automatically

- Special requirements
  - N different user scenarios and 2*N register mask values
  - Automatic UVM register model and mask generation flow based on the register description captured in an Excel sheet
  - Project-specific register adapter to support user scenarios
  - Extended UVM predictor to support different field access policies

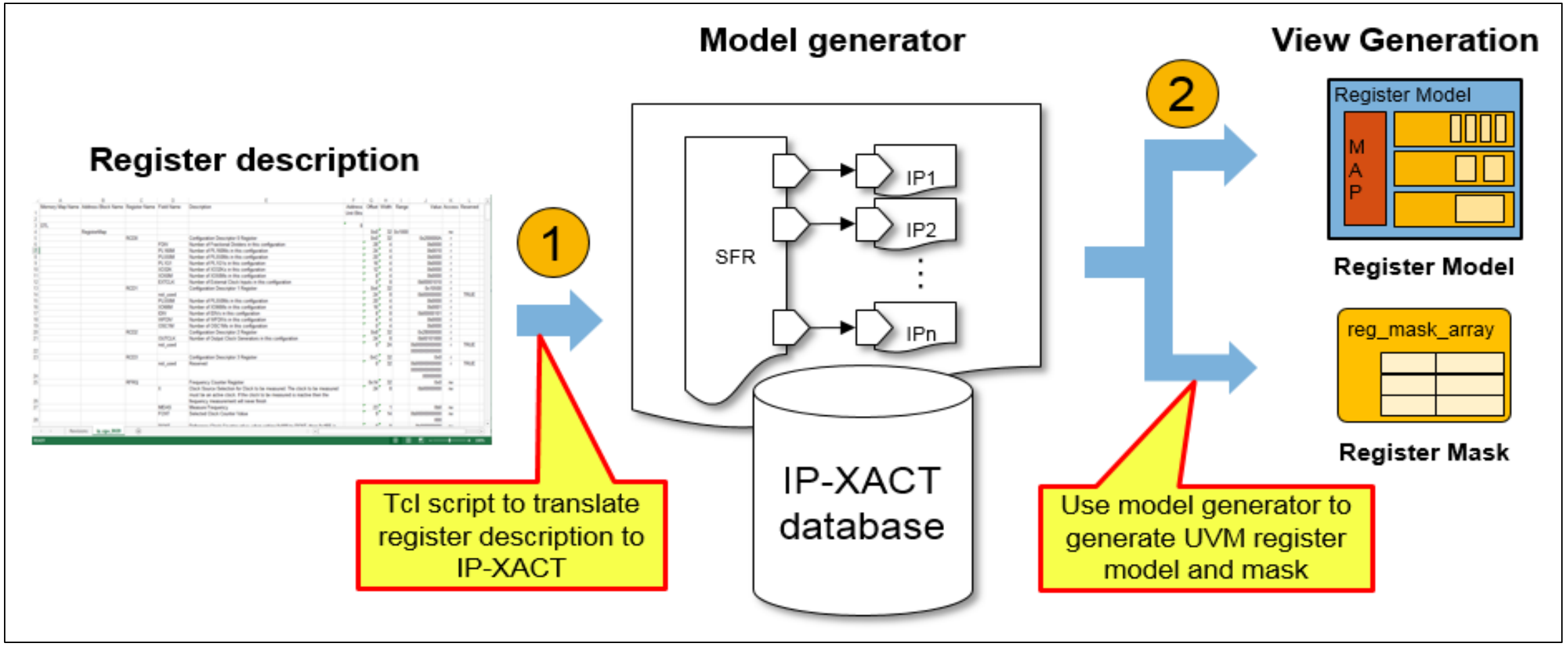
Register Description Excel Sheet

	A	B	C	D	E	F	G	H	I	J
1	Register Name	Field Name	Address	Offset	Reset_Value_h	M1_Read_Mask_h	M1_Write_Mask_h	M2_Read_Mask_h	M2_Write_Mask_h	M3_Re
2	REG_1		0x0000		0x00000000	-----rrrrrrrrrrrrrrrrrrrrrrrrrrrr	----------	-----rrrrrrrrrrrrrrrrrrrrrrrrrrrr	----------	-----rrrrrrrrrrr
3		Field1		0						
4		rfu		27						
5	REG_2		0x0004		0x0000	----rrrrrrrrrrr	----wwwwwwwwwwww	----rrrrrrrrrrr	----wwwwwwwwwwww	--
6		Field1		0						
7		rfu		11						
8	REG_3		0x0006		0x00	r-----rr	w-----ww	r-----rr	w-----ww	
9		Field1		0						
10		Field2		1						
11		rfu		2						
12		Field3		7						
13										

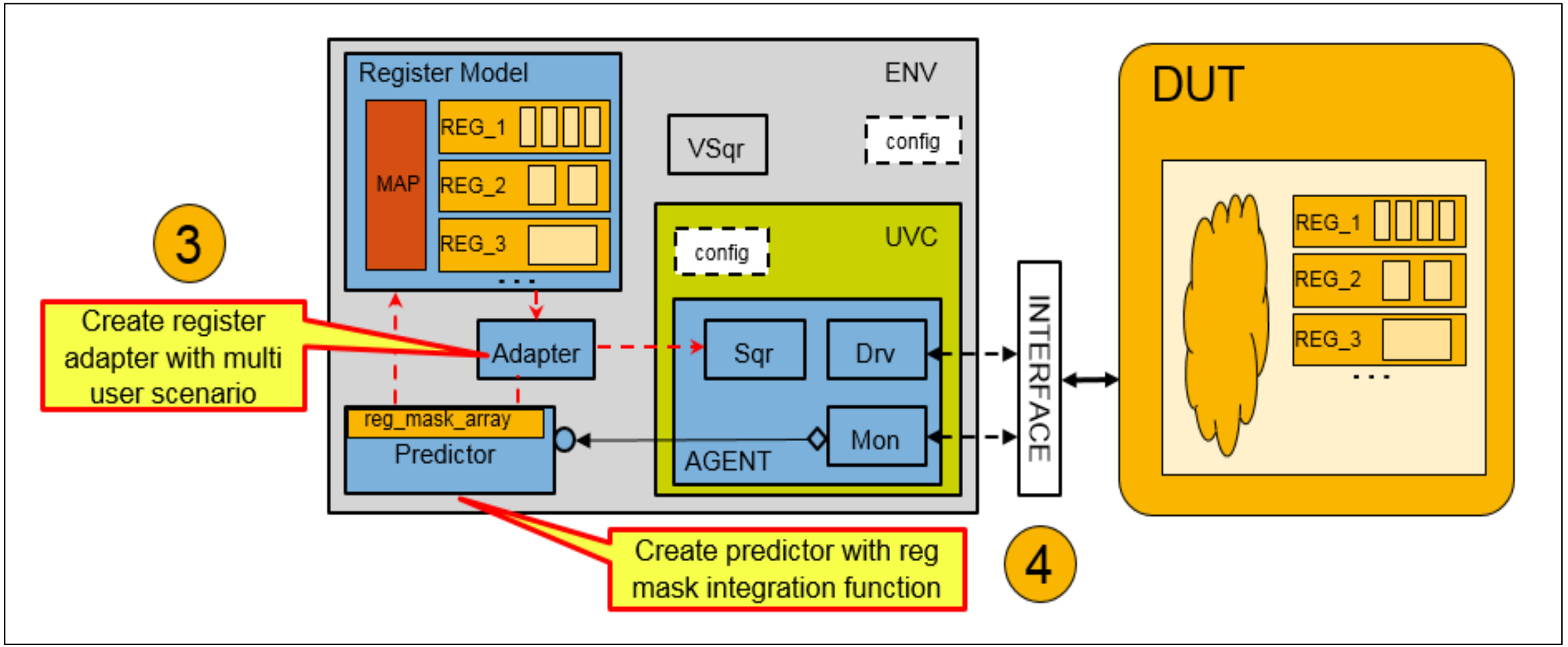
Proposed Methodology



Register Model & Mask Generation Flow



Register Model & Mask Integration Flow



Flow Implementation (1)

- Data structure to save all register mask values

```
class reg_mask extends uvm_object;
  uvm_reg_data_t m1_r_mask  ;
  uvm_reg_data_t m1_w_mask  ;
  ...
  // Constructor
  fucntion new (string name = "reg_mask");
    super.new(name);
  endfunction

  function void add_mask (uvm_reg_data_t m1_r , uvm_reg_data_t m1_w ,...);
  ...
  endfunction

  function uvm_reg_data_t get_mask ( bit wr, user_scenario_e user_scenario);
  ...
  endfunction
endclass
```

```
class reg_mask_block extends uvm_object;
  `uvm_object_utils(reg_mask_block)
  reg_mask reg_mask_array[string];
  // Constructor
  fucntion new (string name = "reg_mask_block");
    super.new(name);
  endfunction

  function void build();
    `include "myblock_reg_mask.sv"
  endfunction

  function void add_reg_mask (string reg_name, uvm_reg_data_t m1_r, uvm_reg_data_t m2_w, ...);
  ...
  endfunction

  function uvm_reg_data_t get_reg_mask ( string reg_name, bit wr, user_scenario_e user_scenario);
    if (reg_mask_array.exist(reg_name))
      return reg_mask_array[reg_name].get_mask(wr,user_scenario);
    else
      `uvm_error("REG_MASK", $sformat("Register %s can't be found in reg_mask_array", reg_name))
  endfunction
endclass
```

Flow Implementation (2)

- Step 1: Convert the XLS/CSV register description into IP-XACT

Register Description

	A	B	C	D	E	F	G	H	I	J
1	Register Name	Field Name	Address	Offset	Reset_Value_h	M1_Read_Mask_h	M1_Write_Mask_h	M2_Read_Mask_h	M2_Write_Mask_h	M3_Re
2	REG_1		0x0000		0x00000000	-----rrrrrrrrrrrrrrrrrrrrrrrrrrrrrr	------------------------------	-----rrrrrrrrrrrrrrrrrrrrrrrrrrrrrr	------------------------------	----rrrrrrrrrr
3		Field1		0						
4		rfu		27						
5	REG_2		0x0004		0x0000	----rrrrrrrrrrrr	----wwwwwwwwwww	----rrrrrrrrrrrr	----wwwwwwwwwww	--
6		Field1		0						
7		rfu		11						
8	REG_3		0x0006		0x00	r-----rr	w-----ww	r-----rr	w-----ww	
9		Field1		0						
10		Field2		1						
11		rfu		2						
12		Field3		7						
13										

1

IP-XACT Metadata

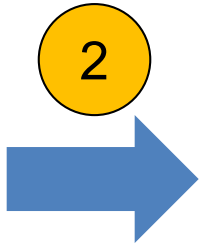
```
<spirit:register>
    <spirit:name>REG_1</spirit:name>
    <spirit:description>Register 1 </spirit:description>
    <spirit:addressOffset>0x0000</spirit:addressOffset>
    <spirit:size>32</spirit:size>
    .
    .
    .
    <spirit:parameters>
        <spirit:parameter>
            <spirit:name>M1_Read_Mask_h</spirit:name>
            <spirit:value>-----rrrrrrrrrrrrrrrrrrrrrrrrrrrr</spirit:value>
        </spirit:parameter>
        <spirit:parameter>
            <spirit:name>M1_Write_Mask_h</spirit:name>
            <spirit:value>--------------------------------</spirit:value>
        </spirit:parameter>
        .
        .
        .
    </spirit:parameters>
</spirit:register>
```

Flow Implementation (3)

- Step 2: Generate register model and mask by model generator

IP-XACT Metadata

```
<spirit:register>
    <spirit:name>REG_1</spirit:name>
    <spirit:description>Register 1 </spirit:description>
    <spirit:addressOffset>0x0000</spirit:addressOffset>
    <spirit:size>32</spirit:size>
    .
    .
    .
    <spirit:parameters>
        <spirit:parameter>
            <spirit:name>M1_Read_Mask_h</spirit:name>
            <spirit:value>-----rrrrrrrrrrrrrrrrrrrrrrrrrrr</spirit:value>
        </spirit:parameter>
        <spirit:parameter>
            <spirit:name>M1_Write_Mask_h</spirit:name>
            <spirit:value>--------------------------------</spirit:value>
        </spirit:parameter>
        .
        .
        .
    </spirit:parameters>
</spirit:register>
```



Generated Register Mask

```
add_reg_mask("REG_1",
    32'b00000111111111111111111111111111,
    32'b00000000000000000000000000000000,
    ...
);
add_reg_mask("REG_2",
    16'b0000111111111111,
    16'b0000111111111111,
    ...
);
add_reg_mask("REG_3",
    8'b10000011,
    8'b10000011,
    ...
);
```

Flow Implementation (4)

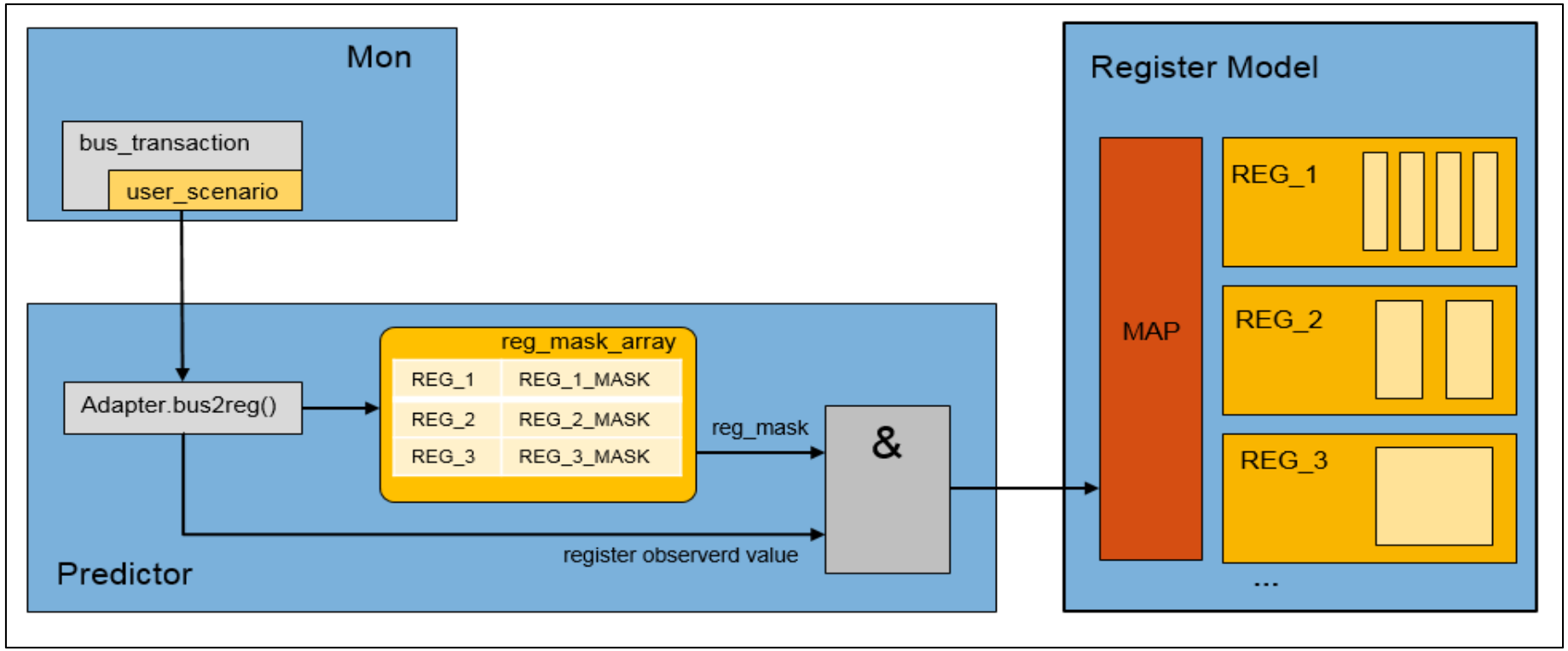
- Step 3: Create register adapter with multi-user scenarios information

```
typedef enum {M1, M2, M3, M4, M5} user_scenario_e;

class extra_bus_info extends uvm_object;

  `uvm_object_utils(extra_bus_info)

  rand user_scenario_e m_user_scenario;
  rand bit             m_parity;
  rand int             m_gap;
```

```
class myblock_reg_adapter extends uvm_reg_adapter;
  ...
  virtual function uvm_sequence_item reg2bus(const ref uvm_reg_bus_op rw);
    myblock_seq_item   myblock_op;
    extra_bus_info     extra_info; //to get extra info
    uvm_reg_item       item;

    myblock_op = myblock_seq_item::type_id::create("myblock_op");
    // to get reg_item extension info from reg.write/reg.read call
    item = get_item(); //this is available only in reg2bus function
    if (item.extension != null) begin
      if (! $cast(extra_info, item.extension))
        `uvm_error(get_name(), "item.extension type is not extra_bus_info.")
        myblock_op.m_user_scenario = extra_info.m_user_scenario;
        myblock_op.m_parity        = extra_info.m_parity;
        myblock_op.m_gap           = extra_info.m_gap;
    end
    else begin
        myblock_op.m_user_scenario = M1;
        myblock_op.m_parity        = 0;
        myblock_op.m_gap           = 0;
    end

    myblock_op.m_direction = (rw.kind == UVM_READ) ? READ : WRITE;
    myblock_op.m_addr = rw.addr;
    myblock_op.m_data = rw.data;

    return myblock_op;
  endfunction: reg2bus
```

Flow Implementation (5)

- Step 4: Integrate register observed value with register mask in predictor



Flow Implementation (6)

- Predictor implementation

```
class reg_predictor extends uvm_reg_predictor#(myblock_seq_item);
  myblock_reg_mask_block    mask;
  ...
  virtual function void write(BUSTYPE tr);
    uvm_reg           rg;
    uvm_reg_bus_op    rw;
    bit               wr;
    user_scenario_e   user_scenario;

     // get user scenario info from monitored transaction
     user_scenario = tr.m_user_scenario;

     adapter.bus2reg(tr,rw);
     rg = map.get_reg_by_offset(rw.addr, (rw.kind == UVM_READ));
     ...
     if (rg != null) begin
       ...
       foreach (map_info.addr[i]) begin
       if (rw.addr == map_info.addr[i]) begin
         found = 1;
         reg_item.value[0] |= rw.data << (i * map.get_n_bytes()*8);
         predict_info.addr[rw.addr] = 1;
         if (predict_info.addr.num() == map_info.addr.size()) begin

         // integration starts from here
         wr = (reg_item.kind == UVM_READ) ? 0 : 1;
         reg_mask = mask.get_reg_mask(rg.get_name(), wr, user_scenario);
         // integrate register observed value with register mask
         reg_item.value[0] = reg_item.value[0] & reg_mask;
         ...
         rg.do_predict(reg_item, predict_kind, rw.byte_en);
         ...
  endfunction
```

Results

```
UVM_INFO my_virtual_seqs.sv(163) @ 880187500: uvm_test_top.m_env.m_v_sqr@@t_seq WRITE register [REG_3],  write_data: ff, user_scenario: M1  parity: 0, gap: 0
UVM_INFO my_monitor.sv(302) @ 880192500: uvm_test_top.m_env.my_agent::m_monitor [MONITOR:monitor_trans] received m_trans_item is:
...
UVM_INFO reg_predictor.sv(162) @ 880192500: uvm_test_top.m_env.reg_predictor [Predictor_Write] BEFORE MASK:register REG_3, reg_item.kind = 1, user_scenario is M1,
reg_item.value[0] = 00000000000000ff,  reg_mask = 00000000000000000000000000000000000000000000000000000010000011

UVM_INFO reg_predictor.sv(165) @ 880192500: uvm_test_top.m_env.reg_predictor [Predictor_Write] AFTER MASK:register REG_3, reg_item.kind = 1, user_scenario is M1,
reg_item.value[0] = 0000000000000083,  reg_mask = 00000000000000000000000000000000000000000000000000000010000011

 register is REG_3, mirrored_value = 83

UVM_INFO my_virtual_seqs.sv(188) @ 990192500: uvm_test_top.m_env.m_v_sqr@@t_seq READ register [REG_3],  read_data: 0, user_scenario: M1, parity: 0, gap: 0
UVM_INFO my_monitor.sv(248) @ 990197500: uvm_test_top.m_env.my_agent::m_monitor [MONITOR:monitor_trans] received m_trans_item is:
...
UVM_INFO reg_predictor.sv(162) @ 990197500: uvm_test_top.m_env.reg_predictor [Predictor_Write] BEFORE MASK:register REG_3, reg_item.kind = 0, user_scenario is M1,
reg_item.value[0] = 0000000000000083,  reg_mask = 00000000000000000000000000000000000000000000000000000010000011

UVM_INFO reg_predictor.sv(165) @ 990197500: uvm_test_top.m_env.reg_predictor [Predictor_Write] AFTER MASK:register REG_3, reg_item.kind = 0, user_scenario is M1,
reg_item.value[0] = 0000000000000083,  reg_mask = 00000000000000000000000000000000000000000000000000000010000011

 register is REG_3, write value = 00000000000000ff, mirrored_value_after_write = 0000000000000083, read back value = 83
```

Conclusion

- Design and implement an UVM-based solution for the case of registers that support field access policies changes according to different user-defined scenarios
- Describe the verification flow implementation procedure
- Use the solution in a real-life project

Questions ?