



UVM Update

Register Package

Sandeep Thakur

Verification Engineer

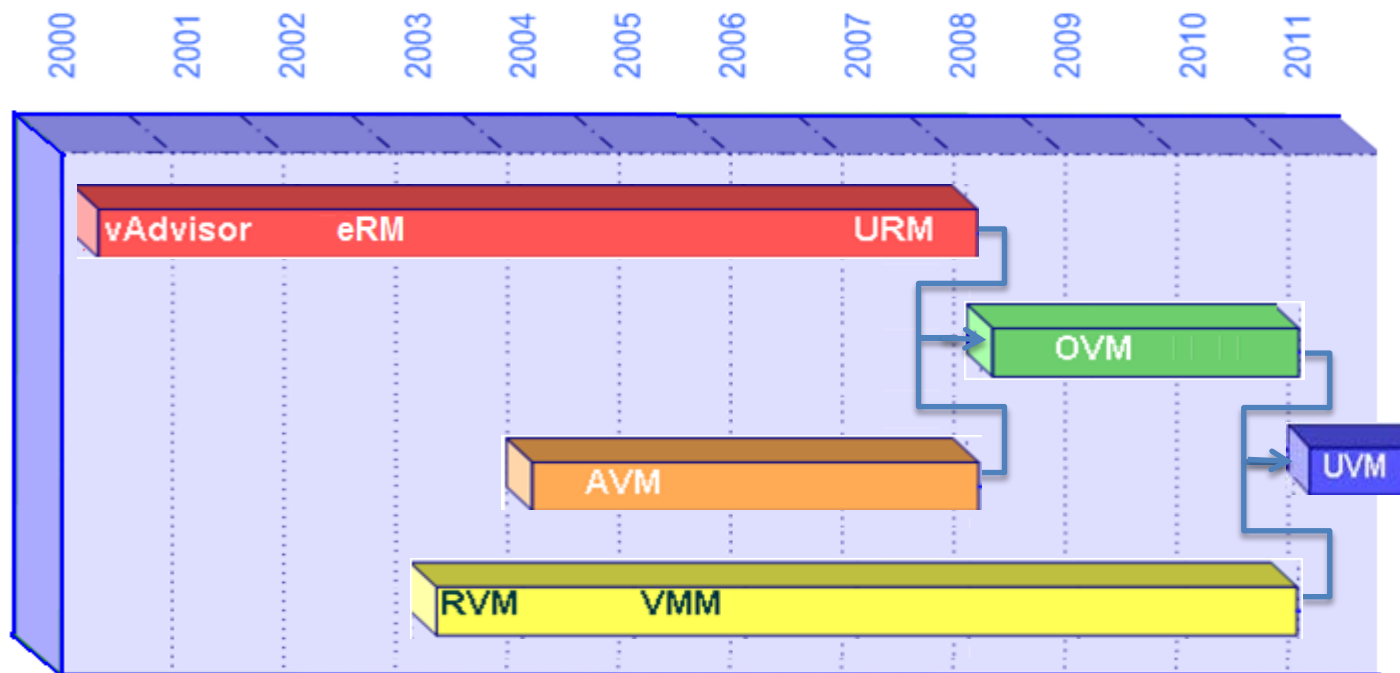
Agnisys Technology Pvt. Ltd.

Agenda

- Introduction to UVM
- UVM Register Model
- Our experience with using Register Model
- Register Model Generator

Verification Methodologies

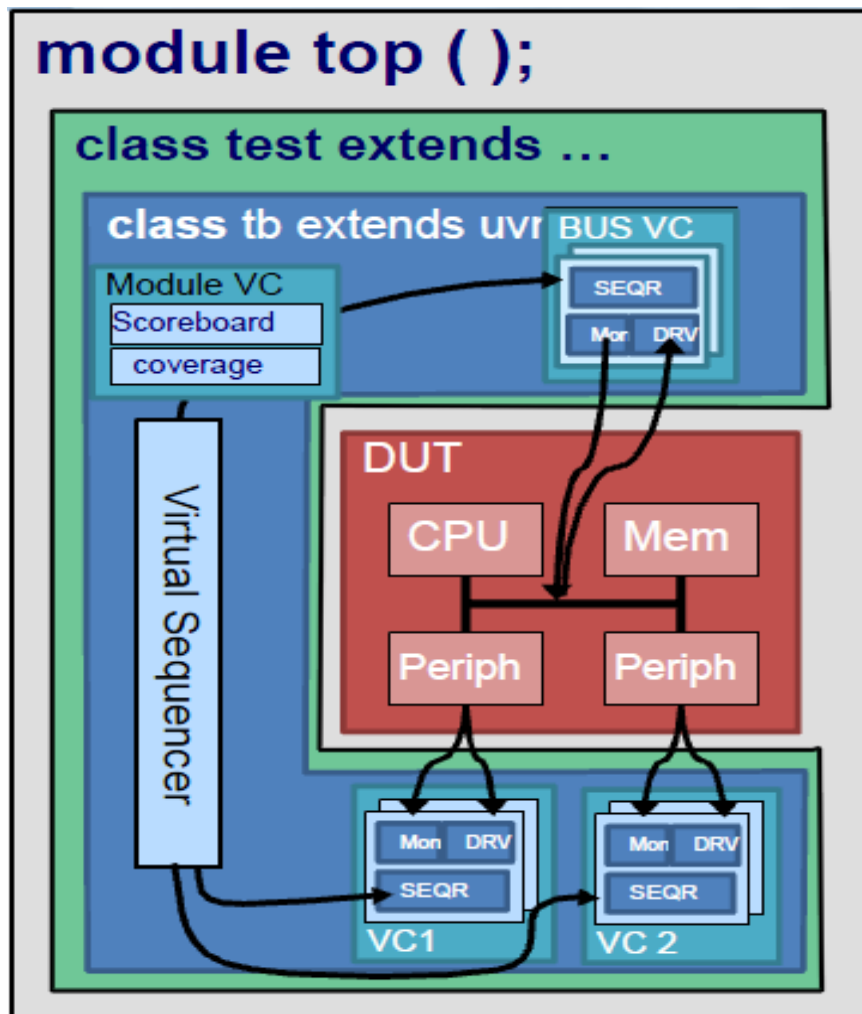
- History
 - **February 2011** Accellera releases UVM 1.0
 - Recently, **June 2011** UVM 1.1 is released



Introduction to UVM

- Universal Verification Methodology
 - A methodology and a class library for building Advanced Reusable Verification Components
- Relies on strong, proven industry foundations
 - Engineers worldwide can write thorough and reusable test environments

UVM Environment



- Module top () as top level element.
 - Test Class
 - Contains Testbench
 - Reusable components with different config

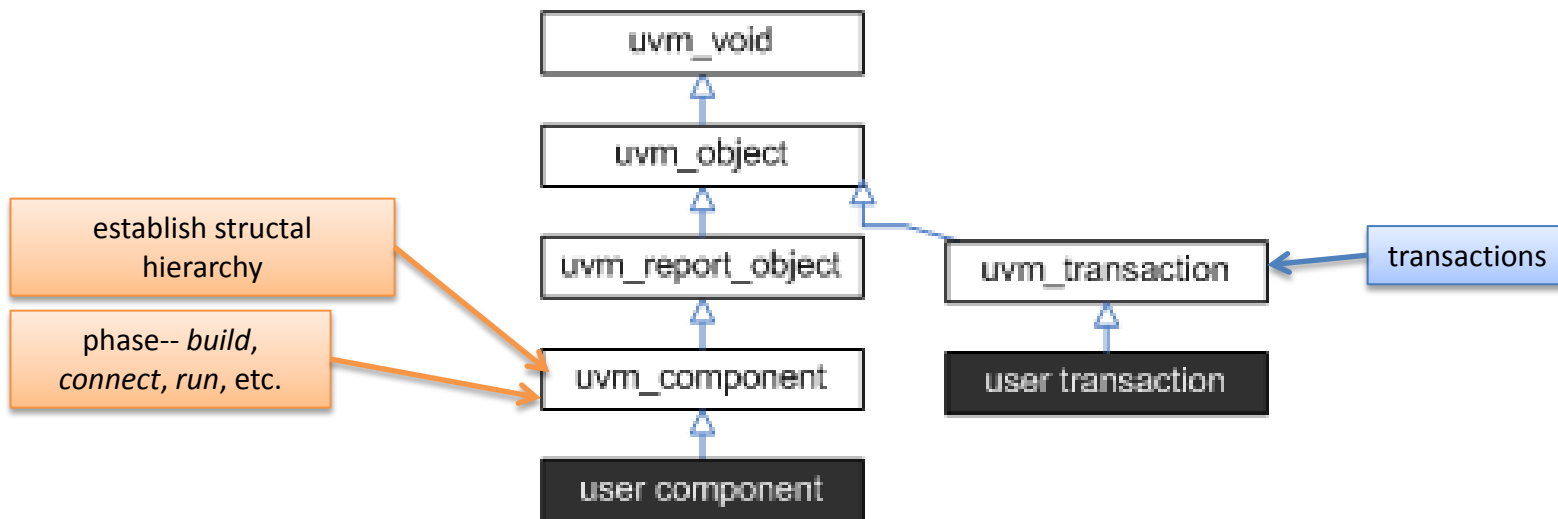
Source: Accellera DAC Presentation

What's in UVM ?

- Base Classes
- Factory Classes
- Phasing
- Configuration
- TLM
- Sequences & Sequencers
- Message Reporting
- Register Model

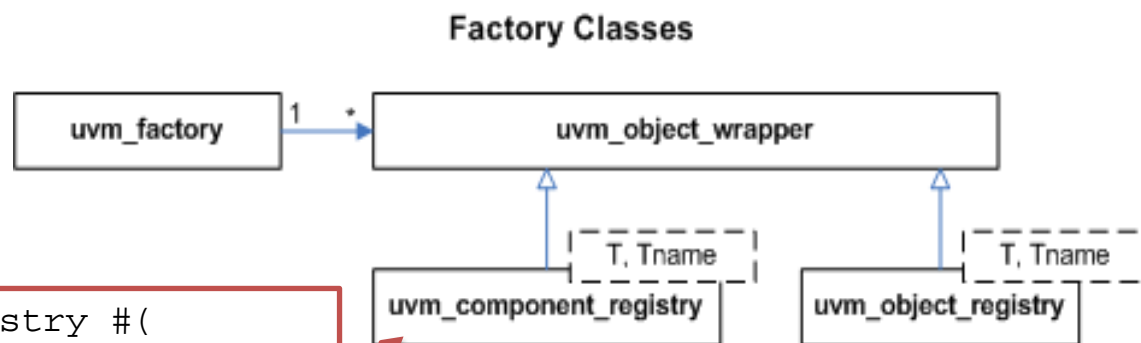
Base Classes

- Facilitate the design of modular, scalable, reusable verification environments
- The basic building blocks for all environments are components and the transactions they use to communicate



Factory Classes

- Manufacture (create) UVM objects and components.
 - Only one instance of the factory is present in a given simulation



```

class uvm_component_registry #(
    type T = uvm_component,
    string Tname = "<unknown>"
) extends uvm_object_wrapper
    
```

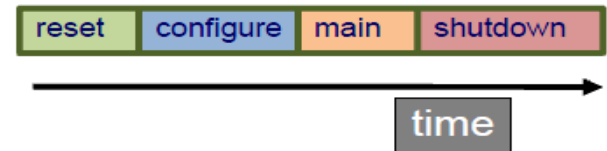
Phasing



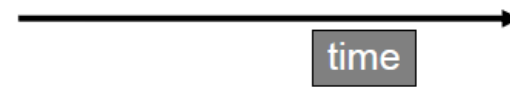
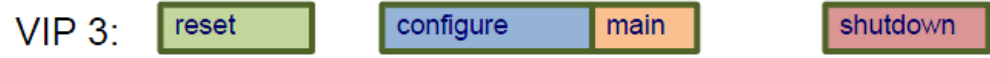
Legacy OVM VIP

new vip

Several new runtime phases in parallel with run_phase()



By default, all components must allow all other components to complete a phase before all components move to next phase



Source: Accellera DAC Presentation

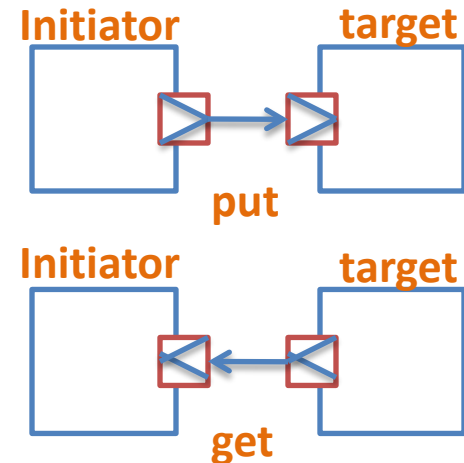
Configuration & TLM

- Configuration

- The configuration & resource classes, access to store or receive from database.
 - uvm_resource_db
 - uvm_config_db
- Configuration mechanism advantages:
 - Wild cards and regular expressions allow configuration of multiple attributes with a single command
 - Run-time configuration support

- TLM

- Unidirectional put/get interfaces
- TLM 2.0
 - Well-defined completion semantics



Sequencers & Sequences

- Sequences
 - User-defined procedures that generate multiple [uvm_sequence_item](#)-based transactions
 - Reused, extended, randomized, and combined sequentially and hierarchically
- Sequencers
 - Arbiter for controlling transaction flow
 - *pull* or *push* semantic between Driver

Message Reporting

- Messages print trace information with advantages over \$display:
 - Aware of its hierarchy/scope in testbench
 - Allows filtering based on **hierarchy, verbosity, and time**

```
`uvm_info("PKT", "Packet Sent", UVM_LOW);
```

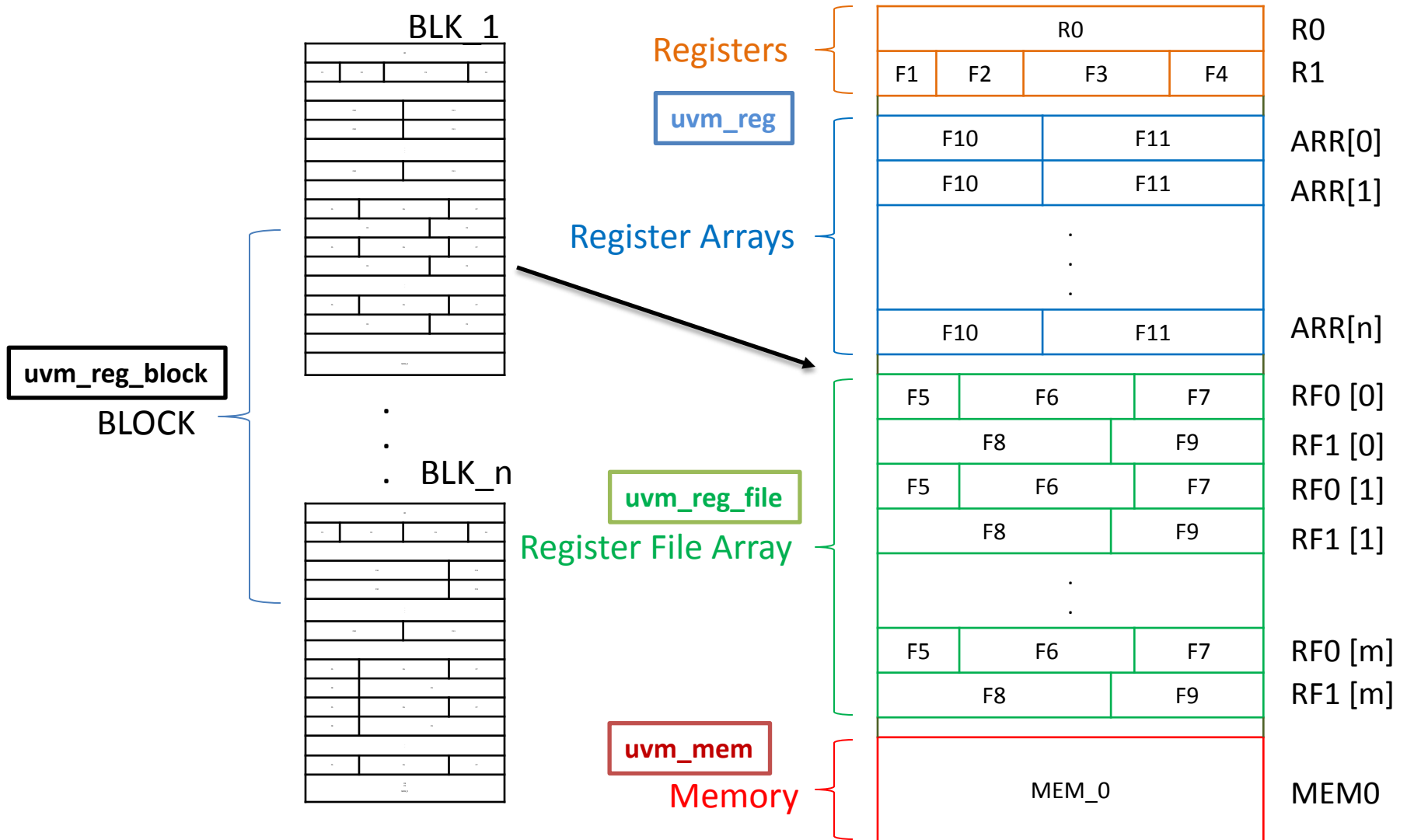
Agenda

- Introduction to UVM
- **UVM Register Model**
- Our experience with using Register Model
- Register Model Generator

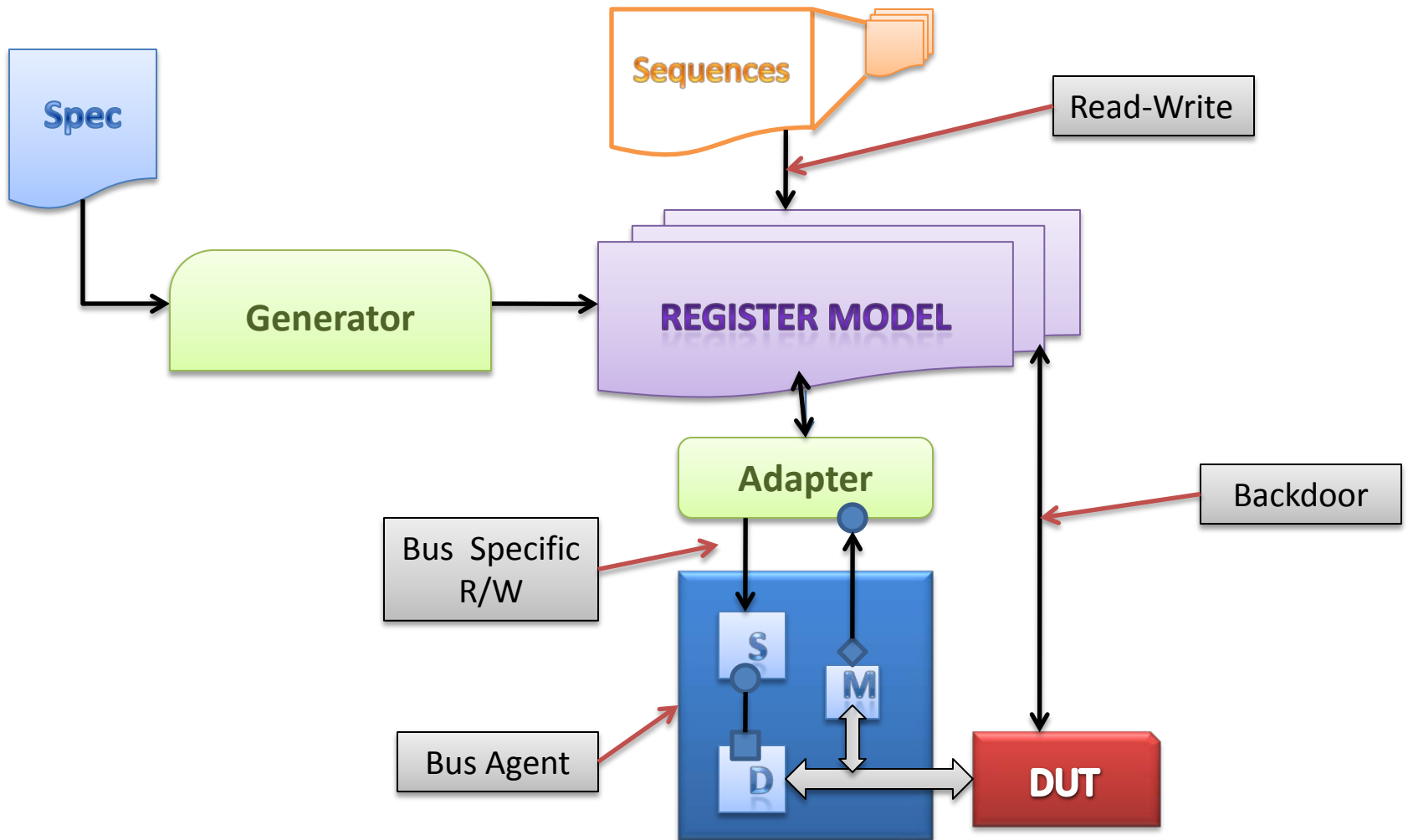
Register Model

- Object oriented Shadow Model for Registers and Memories in DUT
- Components
 - Field
 - Register
 - Register File
 - Memory
 - Block

Register Model

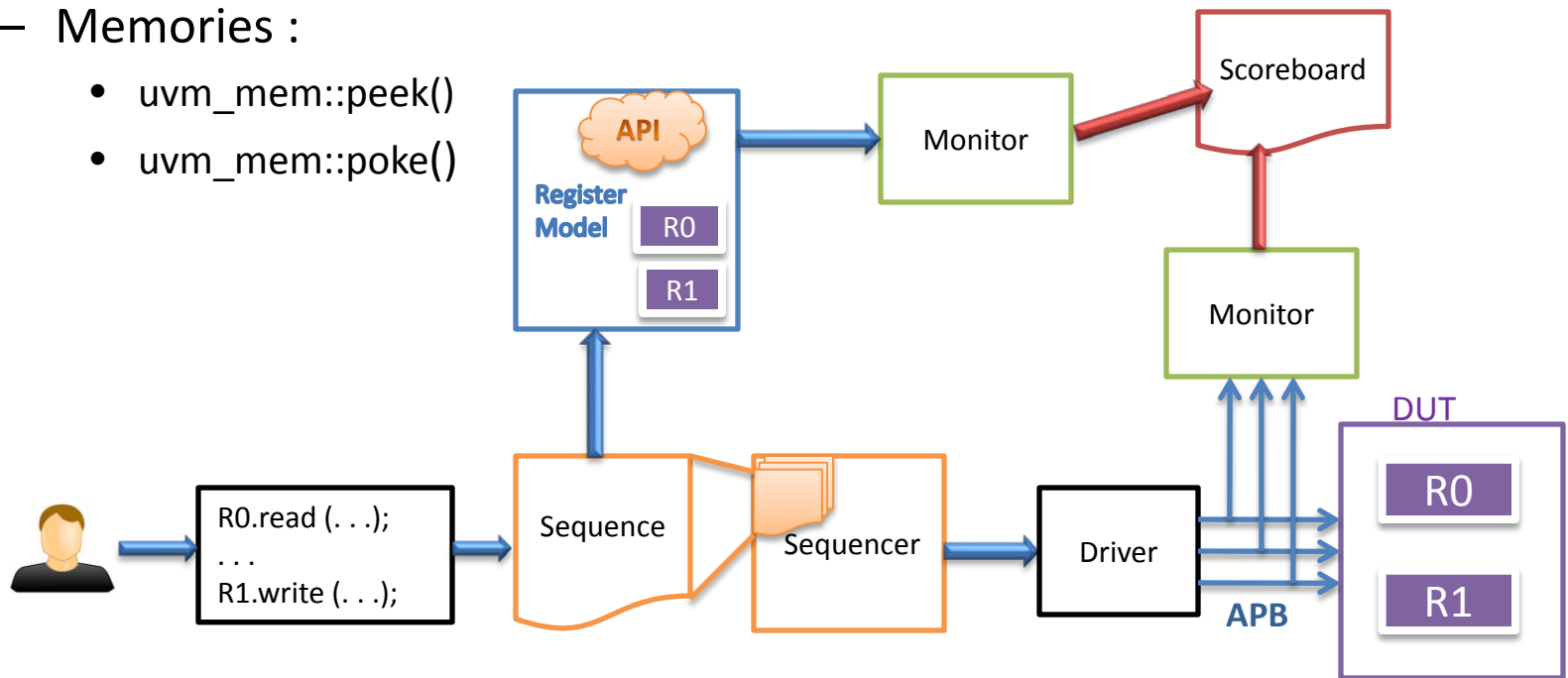


Register Package Usage



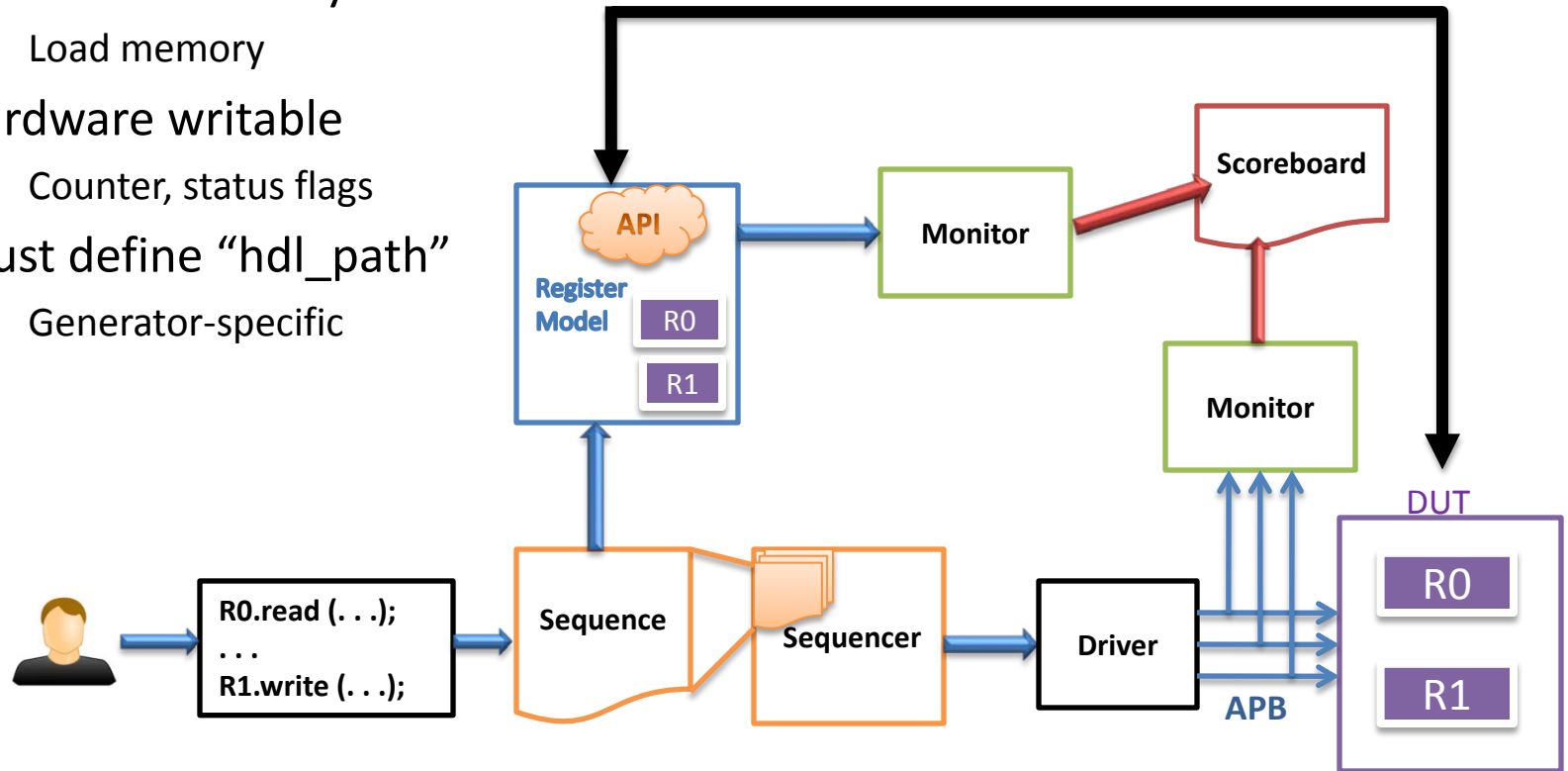
Mirroring

- Register model mirrors content of registers in DUT
 - Updated on read() and write()
 - “Scoreboard” for checking
 - Memories :
 - `uvm_mem::peek()`
 - `uvm_mem::poke()`



Front-door vs. Back-door

- Front Door: Normal bus access
- Back Door
 - Access RTL directly in zero-time
 - Load memory
 - Hardware writable
 - Counter, status flags
 - Must define “hdl_path”
 - Generator-specific



Agenda

- Introduction to UVM
- UVM Register Model
- **Our experience with using Register Model**
- Register Model Generator

HDL Path

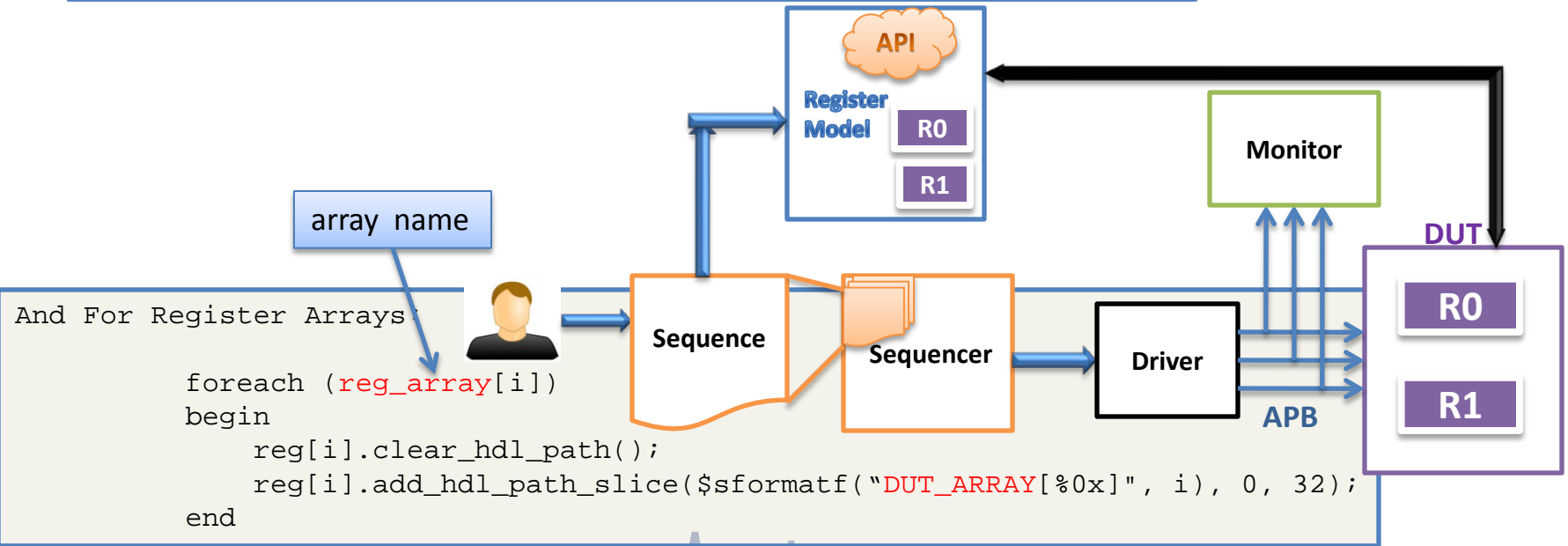
HDL path components are specified using the following methods:

- uvm_reg_block::configure() and `uvm_reg_block::add_hdl_path()`
- uvm_reg_file::configure() and `uvm_reg_file::add_hdl_path()`
- uvm_reg::configure() and `uvm_reg::add_hdl_path_slice()`
- uvm_mem::configure() and `uvm_mem::add_hdl_path_slice()`

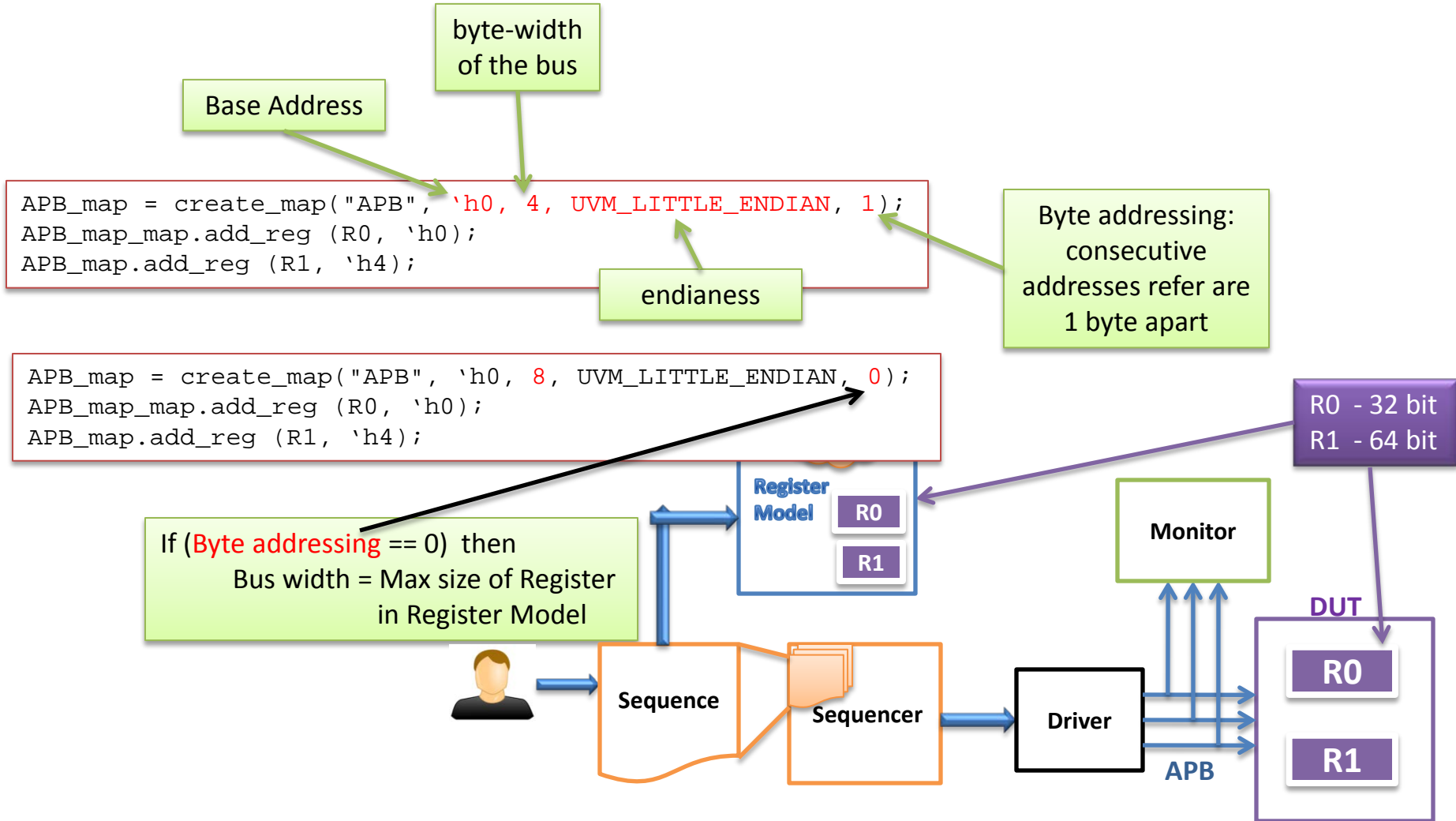
CODE:

```
R0.clear_hdl_path();
  R0.add_hdl_path_slice("dut.R0", 0, 32);
R1.clear_hdl_path();
  R1.add_hdl_path_slice("dut.R1", 0, 64);
```

Clear HDL paths if mentioned above in configure()



Mapping in Block



Coverage

- For all elements except in [Register File](#)
- Pre-defined Functional Coverage Type Identifiers

- UVM_NO_COVERAGE
- UVM_CVR_FIELD_VALS
- UVM_CVR_REG_BITS
- UVM_CVR_ADDR_MAP
- UVM_CVR_ALL

No coverage models.

Coverage models for values of fields.

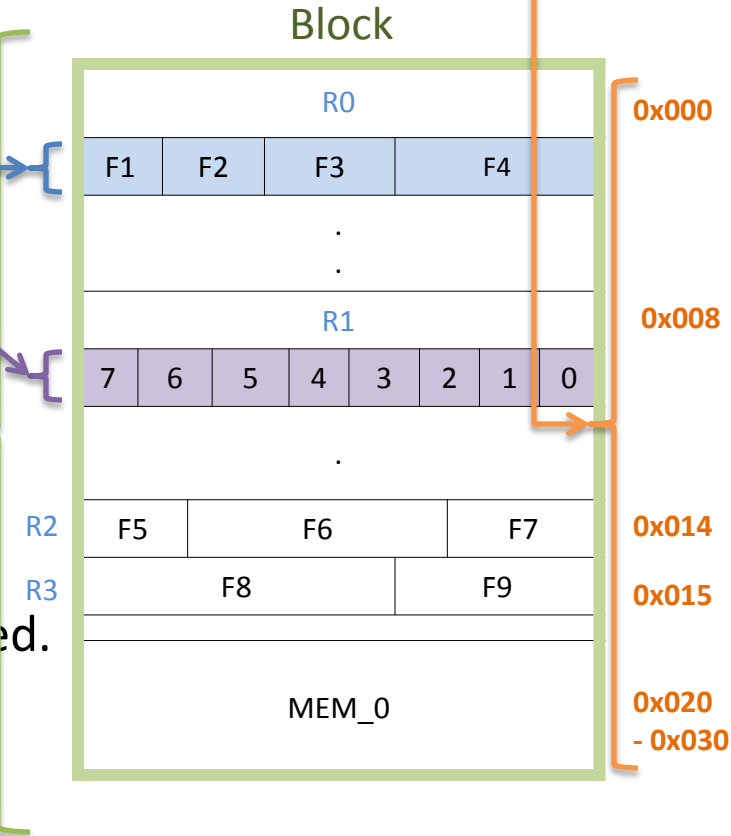
Coverage models for bits read or written in registers.

All coverage models.

Coverage models for addresses read or written in an address map.

- Not instantiated by default
 - Can be large. Instantiate only when needed.
 - To enable:

```
uvm_reg::include_coverage ("*", (UVM_CVR_REG_BITS + .. ));
```



Coverage

```

class block_block extends uvm_reg_block;
  block_MEM0 MEM0;
  block_R1 R1;

  local uvm_reg_addr_t m_offset;

  covergroup cg_addr;
    block_MEM0 : coverpoint m_offset {
      bins hit = { ['h18 : 'h47] };
    }
    block_reg1 : coverpoint m_offset {
      bins hit = { 'h4 };
    }
  endgroup

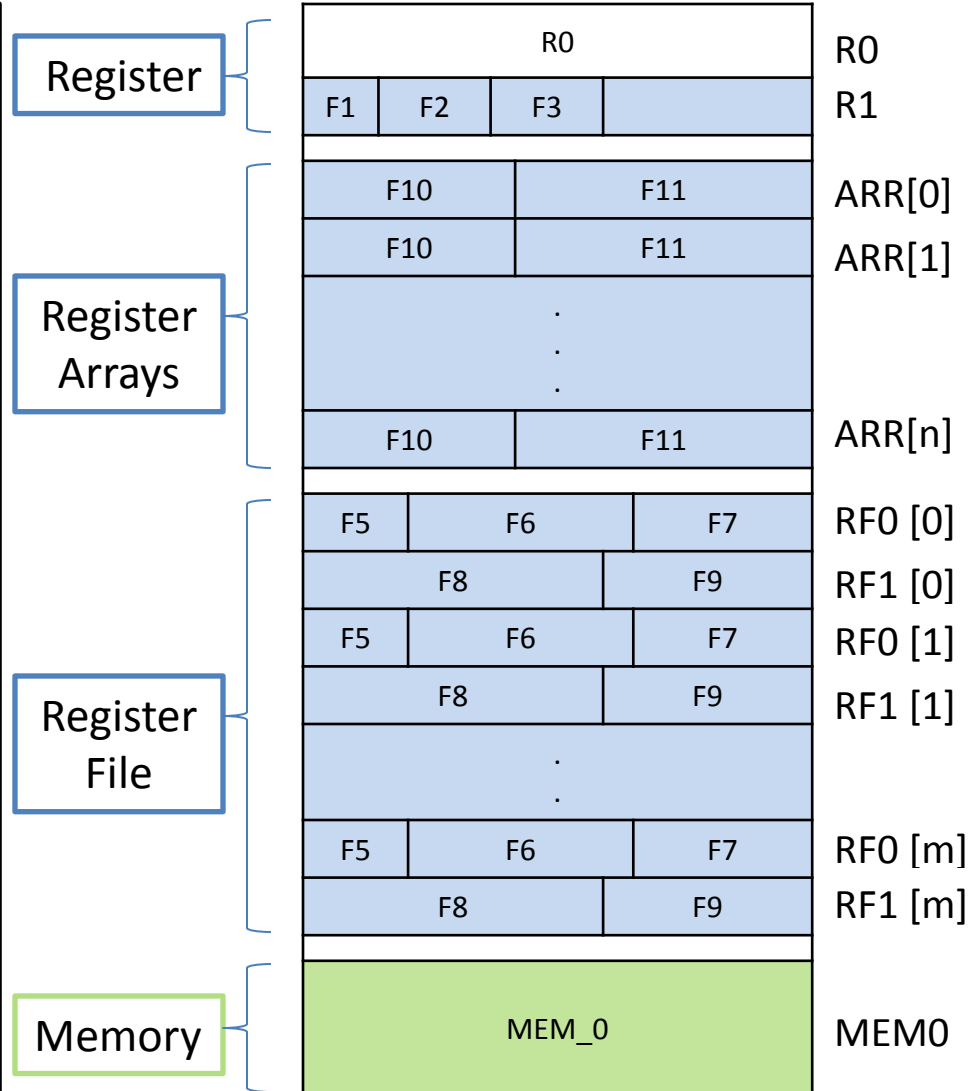
  function new(string name = "block_block");
    super.new(name,
  build_coverage(UVM_CVR_ADDR_MAP));
    if (has_coverage(UVM_CVR_ADDR_MAP))
      cg_addr = new();
  endfunction

  virtual function void sample(uvm_reg_addr_t
offset, bit is_read, uvm_reg_map map);
    if (get_coverage(UVM_CVR_ADDR_MAP)) begin
      m_offset = offset;
      cg_addr.sample();
    end

  endfunction

endclass : block_block

```



Pre-Defined Sequences

- Factory given Sequences
 - hdl_path Access needed

```
uvm_resource_db#(bit)::set({"REG::", regmodel.blk.r0.get_full_name()},  
                            "NO_REG_TESTS", 1, this);
```

Sequence
ignores this
Register

SEQUENCES

```
uvm_reg_hw_reset_seq  
uvm_reg_bit_bash_seq  
uvm_reg_access_seq  
uvm_mem_walk_seq  
uvm_mem_access_seq  
uvm_reg_mem_built_in_seq  
uvm_reg_mem_hdl_paths_seq
```

ATTRIBUTES

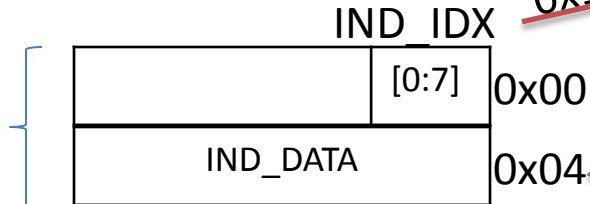
```
NO_REG_TESTS  
NO_MEM_TESTS  
NO_REG_HW_RESET_TEST  
NO_REG_BIT_BASH_TEST  
NO_REG_ACCESS_TEST  
NO_MEM_WALK_TEST  
NO_MEM_ACCESS_TEST
```

Special Registers

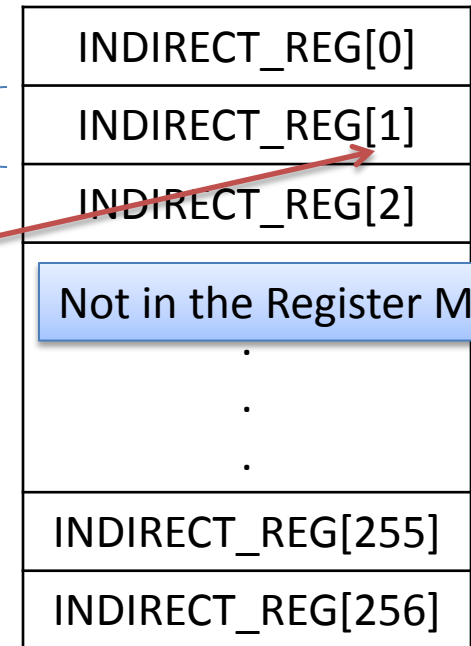
- Pre-Defined Registers
 - Indirect Indexed Registers

```
class my_blk extends uvm_reg_block;
  ind_idx_reg IND_IDX;
  ind_data_reg IND_DATA;
  ind_reg INDIRECT_REG[256];
  virtual function build();
  . . .
`ifdef INCA
  begin
    uvm_reg r[256]; my_blk
    foreach(INDIRECT_REG[i])
      r[i]=INDIRECT_REG [i];

    IND_DATA.configure(IND_IDX, r, this, null);
  end
`else
  IND_DATA.configure(IND_IDX, INDIRECT_REG , this,
null);
`endif
. . .
  default_map = create_map("", 0, 4, UVM_BIG_ENDIAN);
  default_map.add_reg(IND_IDX, 0);
  default_map.add_reg(IND_DATA, 4);
endclass
```



Indirect Register Array
(External)

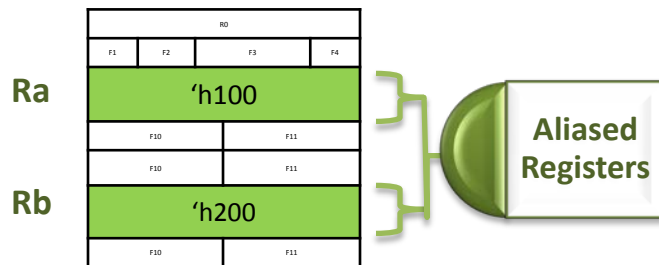


Special Registers

cont. .

- Aliased Registers

- Accessible from multiple addresses in the same address map.
- Fields in aliased registers will have different behavior depending on the address used to access them.



```
class my_blk extends uvm_reg_block;
  rand my_reg_Ra Ra;
  rand my_reg_Rb Rb;
  virtual function build();
  . . . .
  default_map.add_reg(Ra, 'h0100);
  default_map.add_reg(Rb, 'h0200);

  begin
    alias_RaRb RaRb;
    RaRb =
    alias_RaRb::type_id::create("RaRb", ,get_full_name());
    RaRb.configure(Ra, Rb);
  end
endfunction
endclass
```

Special Registers

- FIFO

```
class fifo_reg extends uvm_reg_fifo;  
  function new(string name = "fifo_reg");  
    super.new(name,8,32,UVM_NO_COVERAGE);  
  endfunction: new  
  `uvm_object_utils(fifo_reg)  
endclass
```

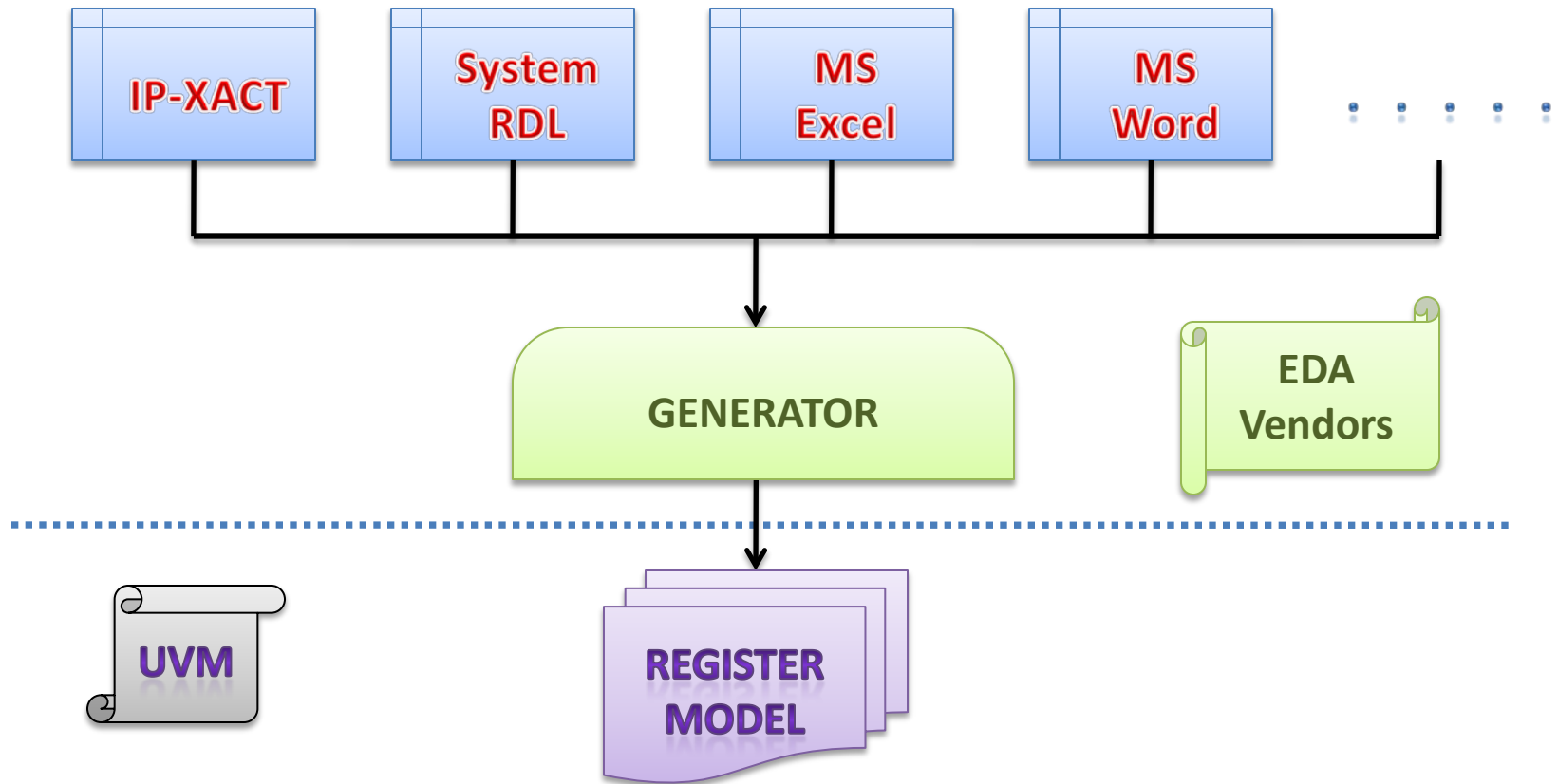
- RO and WO Sharing the Same Address

```
default_map.add_reg(R1, 'h100, "RO");  
default_map.add_reg(W1, 'h100, "WO");  
endfunction : build
```

Agenda

- Introduction to UVM
- UVM Register Model
- Our experience with using Register Model
- **Register Model Generator**

Generation of Register Model



Why use a Generated Register Model

- Create correct-by-construction models
 - Coverage types
 - Constraints
 - Backdoor access
 - Special register
- Sync with specification
- Ease of use

Free UVM Register tools

- Cadence : RGM
 - IP-XACT to UVM
- Synopsys : Ralgen
 - RALF to UVM
- Agnisys : IDSExcels
 - Excels to UVM

Summary

- UVM register package must be used for any serious SoC verification
- Not using a register model is painful
- Not using a generated register model is **very** painful

- Any questions?

THANK YOU

Back Up Slides

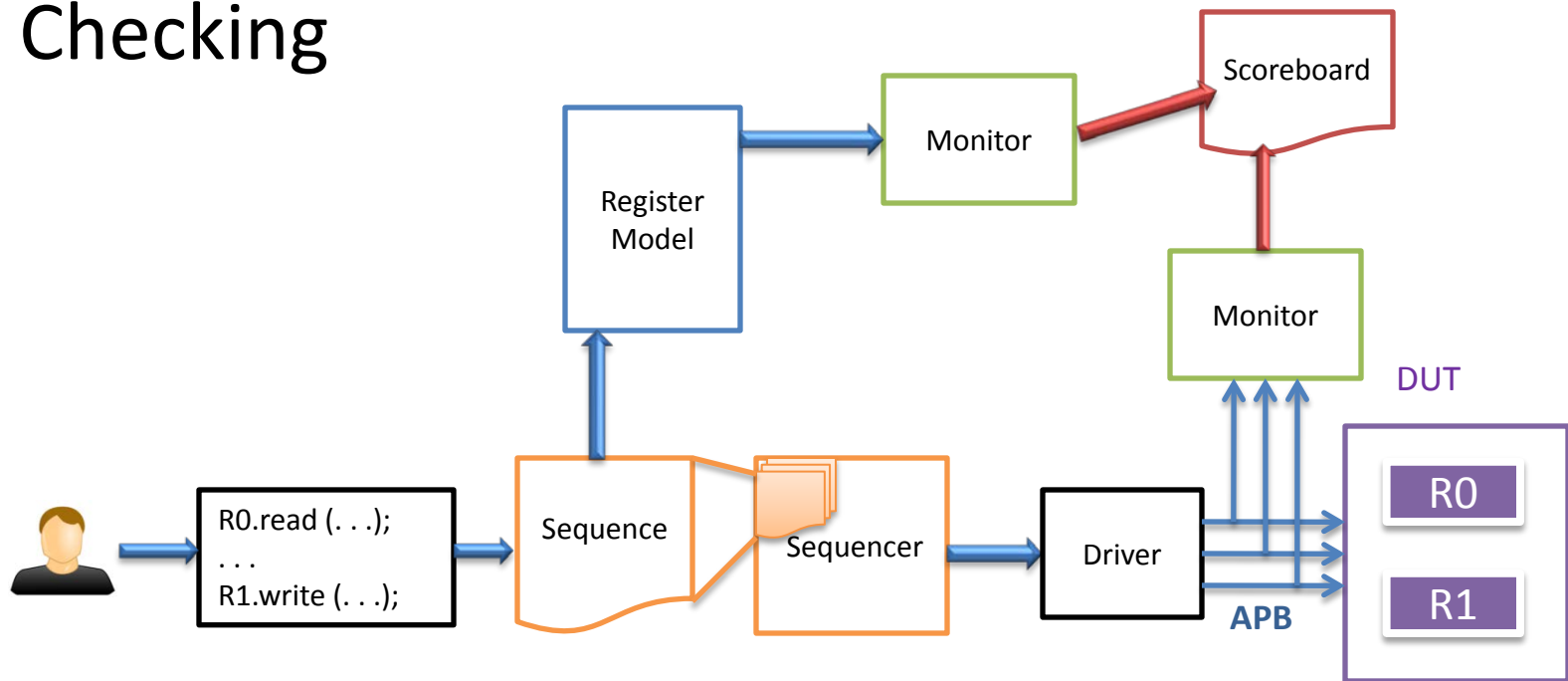
UVM: Factory Classes

cont. .

- Three basic operations for creating components:
 1. Registering objects and components types with the factory
 2. Designing components to use the factory to create objects or components
 3. Configuring the factory with type and instance overrides, both within and outside components

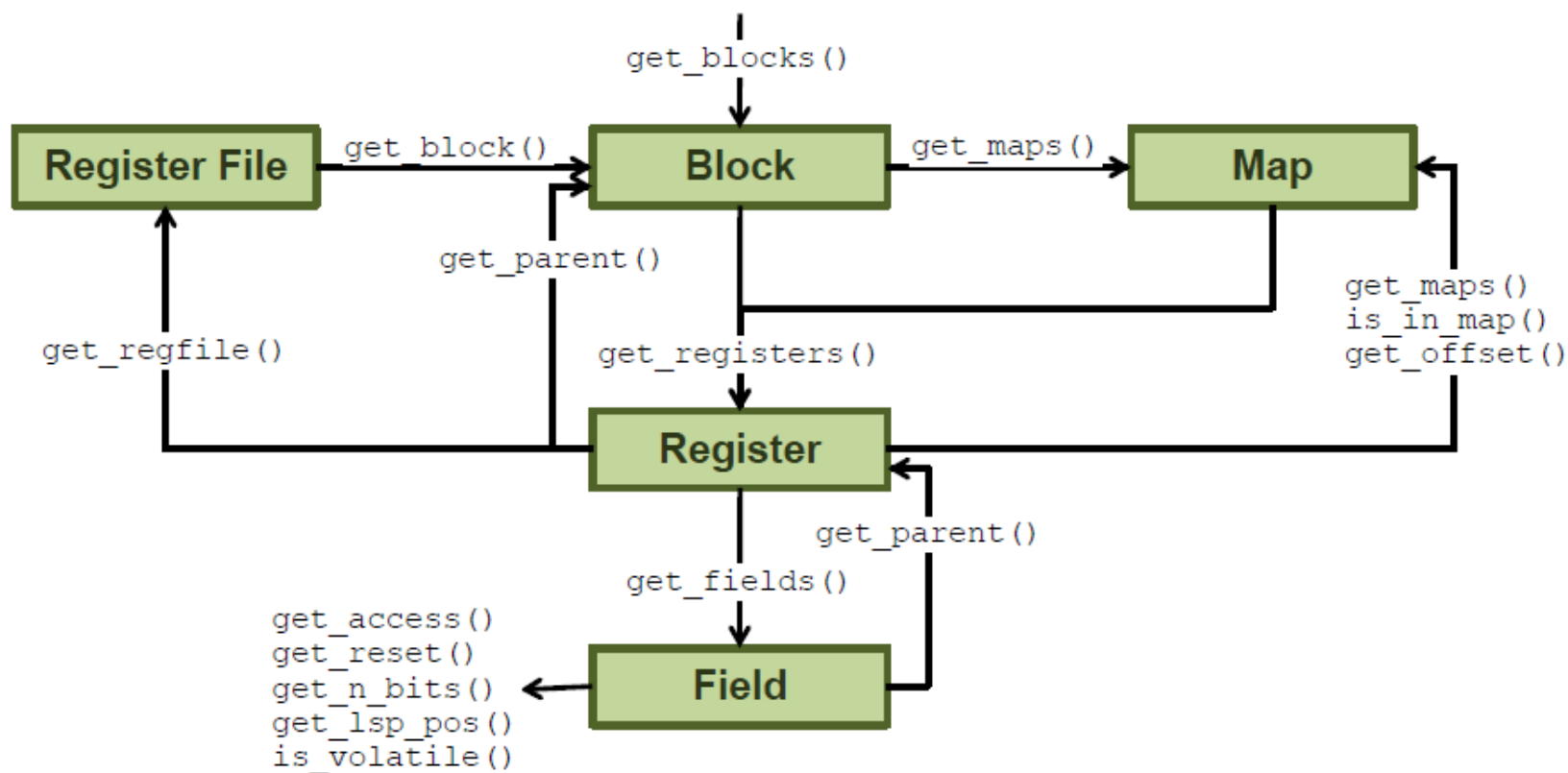
Register Model Usage

- Physical Interface
- Read- Write
- Checking



Introspection

- Rich introspection API



Special Registers

cont. .

- Aliased Registers

```
class write_also_to_F extends uvm_reg_cbs;
    local uvm_reg_field m_toF;
    function new(uvm_reg_field toF);
        m_toF = toF;
    endfunction
```

```
class my_blk extends uvm_reg_block;
    rand my_reg_Ra Ra;
    rand my_reg_Rb Rb;
    virtual function build();
        default_map = create_map("", 0, 4, UVM_BIG_ENDIAN);
        Ra = reg_Ra::type_id::create("Ra", ,get_full_name());
        . . .
        Rb = reg_Rb::type_id::create("Rb", ,get_full_name());
        . . .
        default_map.add_reg(Ra, 'h0100);
        default_map.add_reg(Rb, 'h0200);

    begin
        alias_RaRb RaRb;
        RaRb = alias_RaRb::type_id::create("RaRb", ,get_full_name());
        RaRb.configure(Ra, Rb);
    end
endfunction
endclass
```

```
predict(uvm_reg_field fld,
        uvm_reg_data_t value,
        uvm_predict_e kind,
        uvm_path_e path,
        uvm_reg_map map);
) return;
, UVM_PREDICT_WRITE, path, map));
```

```
extends uvm_object;
Ra m_Ra;
Rb m_Rb;
ils(alias_RaRb)
tring name = "alias_RaRb");
ne);
ew
configure(reg_Ra Ra, reg_Rb Rb);
o_F F2F);

.F1);
    uvm_reg_field_cb::add(Ra.F1, F2F);
endfunction : configure
endclass : alias_RaRb
```