Making RAL Jump, an Introspection

Jeremy Ridgeway
LSI Corporation, Inc.
San Jose, CA

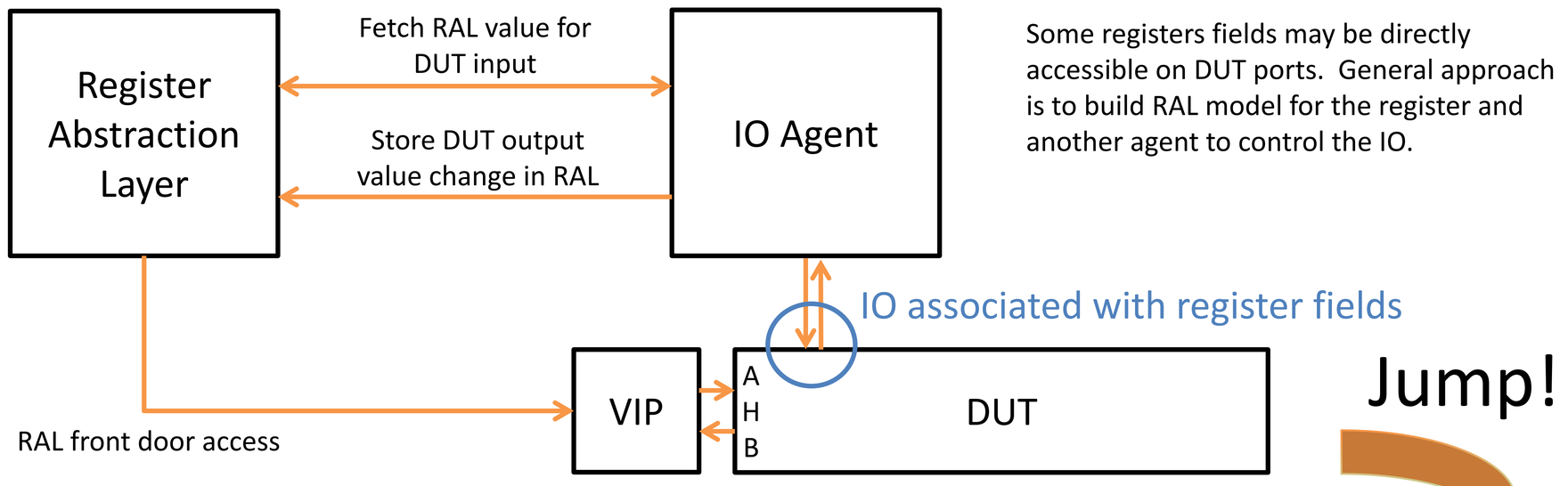
Karishma Dhruv
LSI Corporation, Inc.
San Jose, CA

Manmohan Singh
LSI Corporation, Inc.
Bangalore, India

Abstract

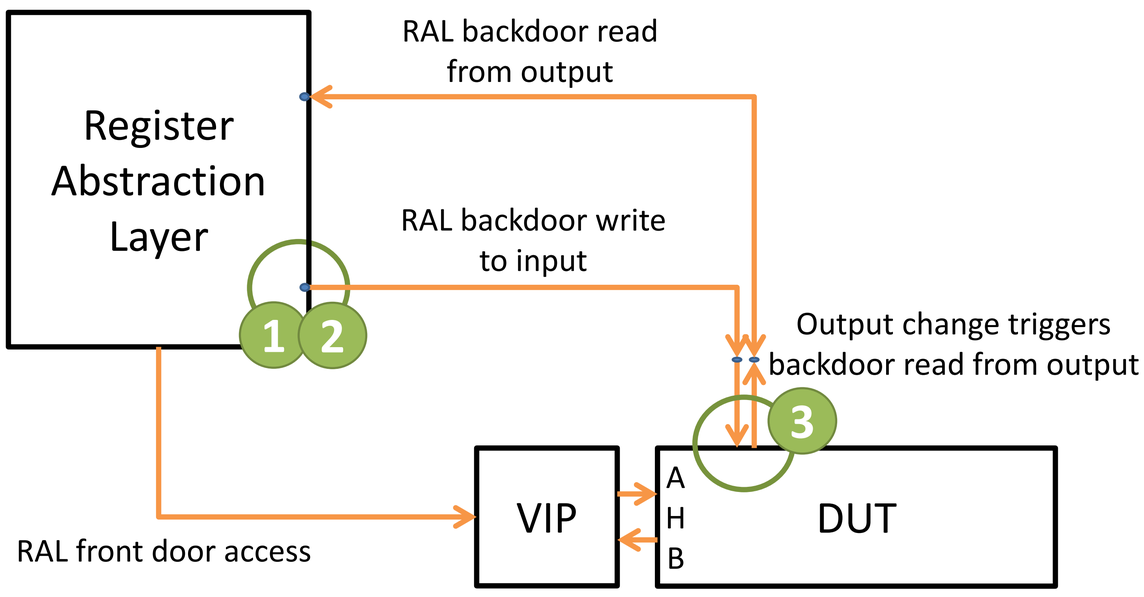
The register abstraction layer (RAL) in the universal verification methodology (UVM) library provides a valuable code base for easy re-use throughout the environment. In a recent PCI-Express translation and link layer verification project, the RAL model was connected to the data path via back door access to primary DUT inputs and outputs. With field-level constraints, scoreboarding, and special back door access handling, our RAL model served as a full verification component. We present details on how to make RAL jump as well as some pros and cons to consider.

DUT Registers with IO access



Some registers fields may be directly accessible on DUT ports. General approach is to build RAL model for the register and another agent to control the IO.

Combine RAL and IOs with Backdoor Access



Access to DUT IOs are transparent to the test bench. No SystemVerilog interface! However, a few points must be taken care of: resets, access, and output changes.

1 Reset values must be driven to DUT inputs

```
RAL_model.reset();
RAL_model.drive_reset();
```

2 Reset values must propagate to inputs

```
class project_reg_field extends uvm_reg_field;
  string m_def_access;
  bit m_drive_reset;
  function void drive_reset();
    m_def_access = get_access();
    m_drive_reset = 1;
  endfunction
  function bit needs_update();
      return m_drive_reset || super.needs_update();
  endfunction
endclass
```

Register field volatility and access type require special handling in a base class.

3 Monitors need to trigger backdoor read from outputs

```
interface my_ral_interface_monitors;
  bit my_reg_request_updatep = 0;
  always @(posedge my_reg_request_updatep) begin
    RAL_model.my_reg.read(.path(UVM_BACKDOOR));
    my_req_request_updatep = 0;
  end
  always @(my_reg_field_A_output)
    my_reg_request_updatep <= 1;
  always @(my_reg_field_B_output)
    my_reg_request_updatep <= 1;
endinterface
```

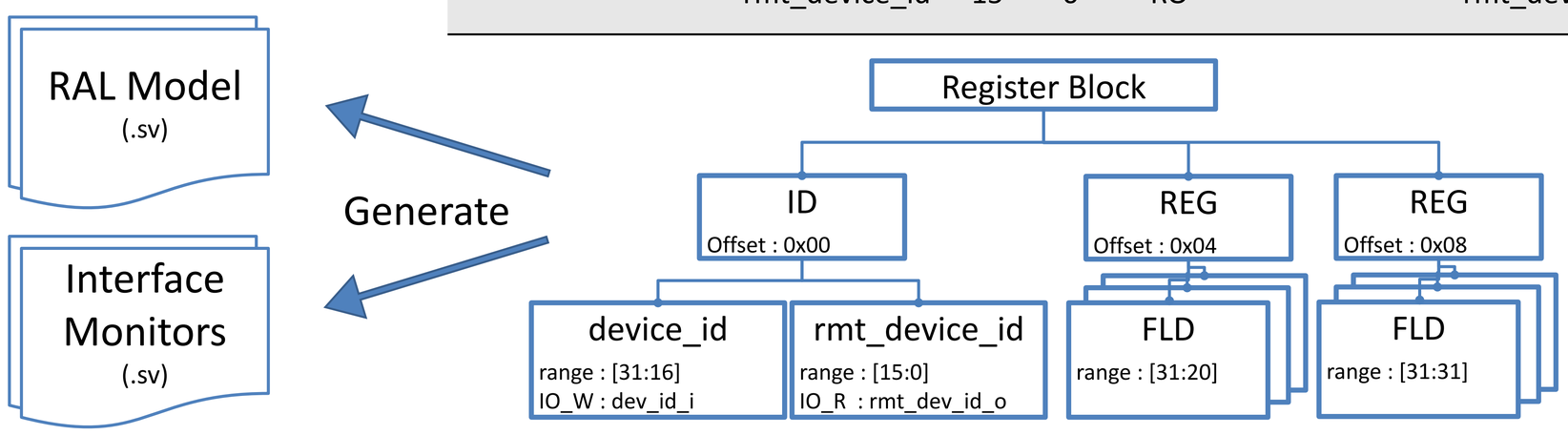
RAL Automation

RAL base class changes are transparent and monitors algorithmic, thus the whole register model generation may be automated.

Name	Address	Bit field	MSB	LSB	Access	IO Path	IO Read Path	Description
ID	0x00							
		device_id	31	16	RW	dev_id_i		Our device ID
		rmt_device_id	15	0	RO		rmt_dev_id_o	Partner's device ID

a. UVM RAL SystemVerilog model instantiated with global access.

b. RAL IO signals instantiated in SystemVerilog interface with known hierarchy.



Conclusions

We presented an approach for UVM RAL model back door access to DUT primary inputs and outputs. Several challenges to this approach lead us to back door path mapping, driving reset to IOs, and active monitoring as solutions. Our implementation was simple *enough* to integrate within the existing RAL model flow with some extensions. The base test was aware of these extensions and required some special care to build the model and drive the reset values, but test code was oblivious.

The sheer volume of registers requires automation. There is a trade-off here between custom automation and third-party tool. Unless it is a company requirement to distribute one of the popular register formats, then translation into RALF, CSRSpec or SystemRDL is a schedule burden. Furthermore, if field level randomization and back door path mapping is not supported by the tool vendor for these formats, then customization should be considered. Nonetheless, it may be simpler for a one-off or short-lifespan project to randomize at the register and build components to interact with a generated RAL model. This was not the case for our project.