2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Dynamic Control Over UVM Register Backdoor Hierarchy

Roy Vincent, Senior Design Engineer, Analog Devices Inc., Bengaluru, India
Unnikrishnan Nath, Design Verification Engineer, Analog Devices Inc., Bengaluru, India
Ashok Chandran, Engineering Manager, Analog Devices Inc., Bengaluru, India

***Abstract*- Recent times, the integrated circuit complexity has increased many fold. A fundamental reason is the increased feature addition into a single chip to sufficiently satisfy multiple customer requirements. This has made the programming of the chip complex. From the DV perspective, proper programming is essential to get the chip into the required state. This initial programming section consumes much of the simulation time before the actual functional test starts, especially if it is done through some slow peripherals like serial ports. UVM register backdoor access is an advantage here but will not directly solve all the use-cases. This paper discusses how to dynamically change the backdoor hierarchy for a register access during simulation time by using the UVM register callbacks from one of our use-case perspectives.**

I. INTRODUCTION

The UVM backdoor write access uses the DPI calls to deposit the required value from the RAL model at the required hierarchical node which is conventionally a register's immediate output node. Similarly, for backdoor read, the required hierarchical node is directly polled, the value taken and updated to the RAL model instantaneously. The mechanism is illustrated in figure 1.

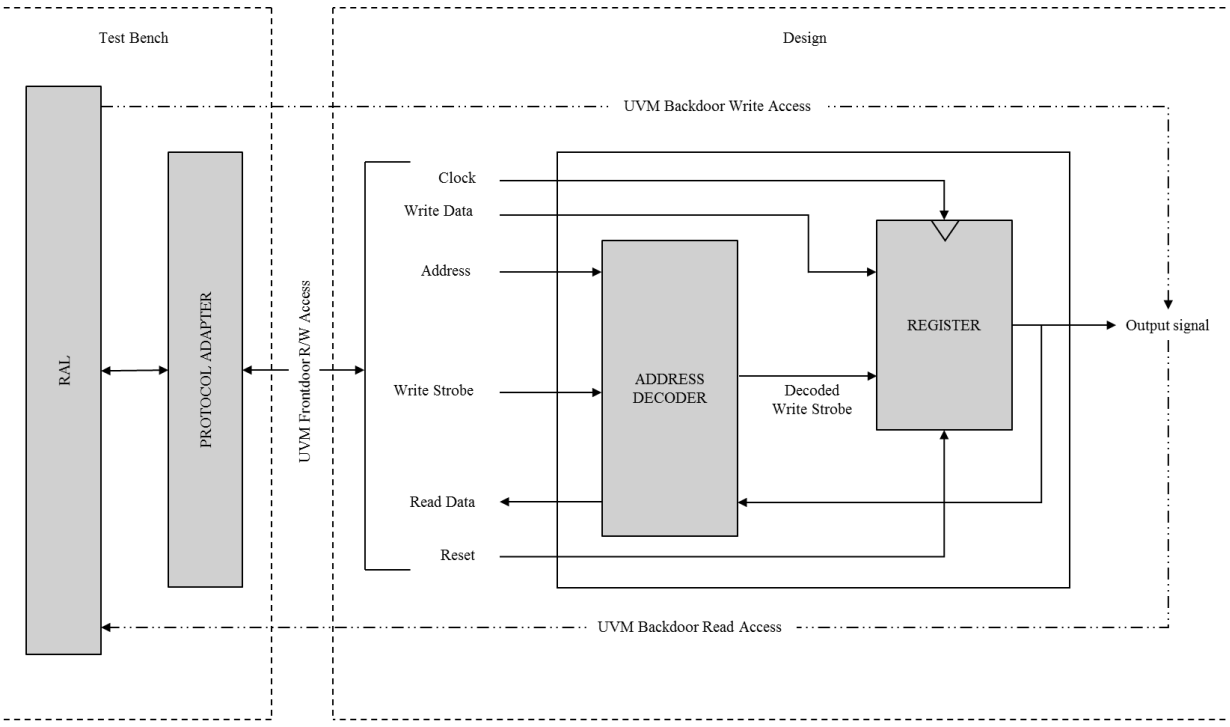

Figure 1

The above-shown mechanism is a straightforward use-case. But in SoC, paged registers are common. This is to program multiple similar blocks inside the SoC at one shot therefore reducing the programming time. One can think of broadcasting the required values to multiple registers at the same time. Principally, these registers will have the

same address and there will be associated extra control bits, often from another register, which controls whether the access should go through or not. For e.g., 4 instances of a register ABCD with same address 0x1234 can be present in a SoC whose programming is controlled by another register say PAGE. So, if PAGE is set to 1111 (binary), a single write to 0x1234 will result in an access to all 4 instances, hence reducing the effective programming time of the chip. Controlling the PAGE register value will help us to target any required instance independently also. This independent control is where the standard usage of the backdoor access has got issues.

II. PROBLEM STATEMENT

UVM backdoor access works with DPI calls which operate on the hierarchy strings set from the top.

The register hierarchy is set in the build phase of the top RAL model.

reg_blk.configure (this, "<Design Hierarchy up to Register top>");

The final endpoint is mentioned in the register block's build phase. The offset and size determine the position of the bitfield node in the bigger reg bus.

reg.add_hdl_path_slice ("<final bitfield node name>", <offset>, <size>);

With more paged registers, we can always add more HDL path slices corresponding to multiple nodes. But if we try to access one of these registers though backdoor, all the mentioned slices will get updated with the same value which is not desirable. Therefore, we need to dynamically edit the slice hierarchy according to the page control register values. This dynamic control is not inherent with the normal backdoor access.

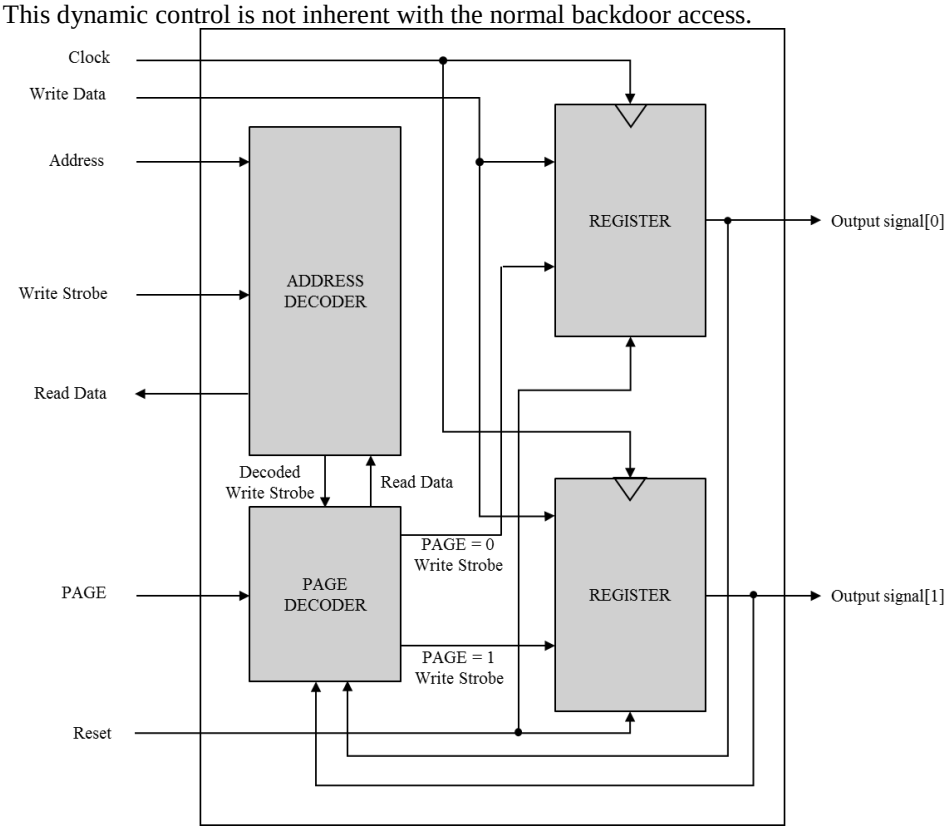

Figure 2

As shown in figure 2, mentioning slice hierarchy 'output signal[0]' and 'output signal[1]' during the build phase will make issues while trying to access these registers individually during simulation runtime.

III. IMPLEMENTATION

The register call back methods are used to implement the node hierarchy edits dynamically during simulation runtime. One major assumption or requirement for the whole setup to work is that the paged registers should provide the outputs in the arrayed format as shown in figure 2. This helps the hierarchy edits through easy indexing, based on paging value. UVM may not allow altering the hierarchies added during the build phase. Therefore, we need to

create a new section under a different uvm_reg_item::bd_kind, add the edited slice node hierarchies to it and execute the backdoor access on the newly created bd_kind. In general, write access is where the problem lies. For read access, we do have a paging based decoded data node which can be probed. Separate classes are autogenerated to get the info related to register and paging register linkage.

The flow chart in figure 3 explains how callbacks are used in the backdoor write.

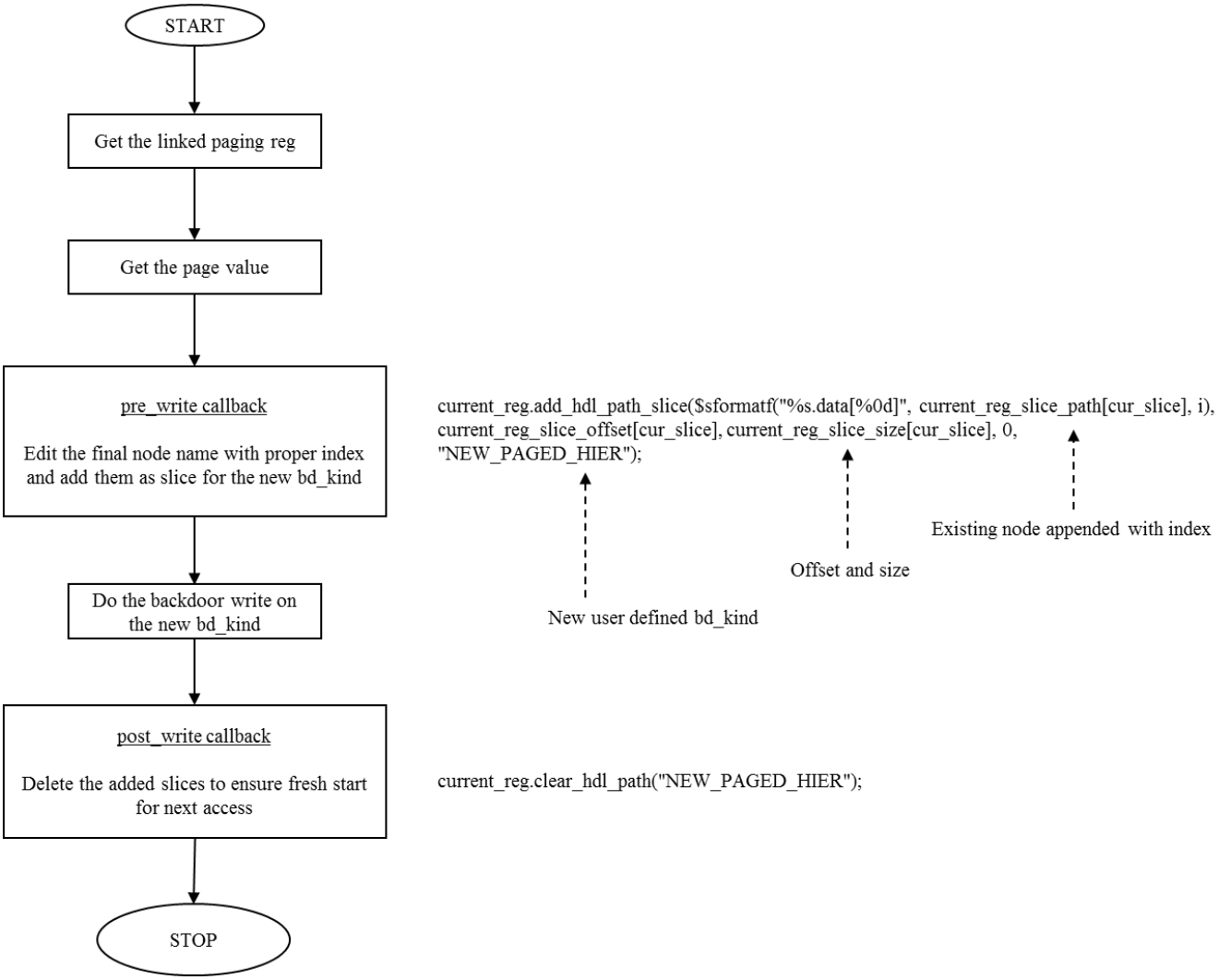

Figure 3

IV. SOURCE CODES

Following is a set of source code with appropriate comments to show how the hierarchy edits are attained.

```
/*
* Hierarchy Editing Logic
* 1. Prerequisites
*     a. The tool/ script dumping out the RAL model classes
*        should dump out the backdoor signal also (E.g. O_reg1)
*     b. The backdoor signal (path slice) will be appended
*        to the top hierarchy to get proper full hierarchy (E.g. `REGBLK1.O_reg1)
*     c. The design signals for the paged registers
*        should be arrayed so that it can be easily
*        maniputated in the call backs (E.g. O_reg1[0], O_reg1[1])
* 2. Hierachy editing
*     a. See if the register is paged or not
*     b. If paged, get all the properties like path,
*        offset and bit width size for duplicating
*     c. See the page setting and add the hierarchy
*        by appending the array number
*             E.g. if page setting is '01', append [0]
*                  to the exiting hierachy and add as a slice (O_reg1[0])
*                  if page setting is '11', append [0]
*                  for one slice and [1] for the next slice (O_reg1[0], O_reg1[1])
*     d. All the new hiearchies should be
*        added to a new bd_kind (e.g NEW_PAGED_HIER).
*             Why???
*             To the original bd_kind mentioned in the build phase,
*             we can add new path slices, but can't delete individually
*             Deleting it completely will make the hierarchy reference
*             to be lost for the subsequent reg access
*             The dynamically created bd_kind,
*             we can safely delete it completly after the operation
*/


// Task to edit the current hierarchy to valid hierarchy
virtual task paged_reg_hier_edit(uvm_reg_item rw);
  // Get current hdl_path/hierarchy
  current_reg.get_hdl_path(current_path);
  // Iterating across paths
  foreach(current_path[path_num]) begin
    // Iterating across slices and adding the components of each slice to thier
    // respective queue to be called back later
    foreach(current_path[path_num].slices[slice_num]) begin
      // Get the current path for editing
      current_reg_slice_path.push_front(current_path[path_num].slices[slice_num].path);
      // Get the offset for duplication
      current_reg_slice_offset.push_front(current_path[path_num].slices[slice_num].offset);
      // Get the bit width size for duplication
      current_reg_slice_size.push_front(current_path[path_num].slices[slice_num].size);
      `uvm_info(get_type_name(),$sformatf("Existing Original Slice
             [Name : %s, Offset : %0d, Size : %0d]",
             current_reg_slice_path[0], current_reg_slice_offset[0],
             current_reg_slice_size[0]), verbosity)
    end
  end
```

```
    foreach(current_reg_slice_path[cur_slice]) begin
      // Get the mask value. There is a differnet
      // section (not shown here) which will get the correct
      // mask_field corresponding to the current paged register
      // (It's basically a LUT)
      msk_value=msk_field.get();
      `uvm_info(get_type_name(),$sformatf("msk_value=%0b",msk_value),verbosity);
      // Iterating over all mask bits
      for(int i=0;i<msk_n_bits;i++)begin
        // Checking if mask bit is one
        if(msk_value[i]==1)begin
          `uvm_info(get_type_name(),$sformatf("Mask Value is set, Adding the
          new hier for deposit (Mask Pos : %0d)", i), verbosity)
          // Edit the hierachy to accomodate the page indexing.
          // Add to the new bd_kind (NEW_PAGED_HIER)
          current_reg.add_hdl_path_slice(
                  $sformatf("%s[%0d]", current_reg_slice_path[cur_slice], i),
                  current_reg_slice_offset[cur_slice],
                  current_reg_slice_size[cur_slice], 0, "NEW_PAGED_HIER");
        end
      end
    end

    //Now you have all new required PATHs in current_reg with bd_kind = NEW_PAGED_HIER
    // Deleting the temp queue
    current_path.delete();
  endtask
```

The above code is put inside the pre_write task. Also, at the end of the pre_write task, the bd_kind of uvm_reg_item need to be changed to “NEW_PAGED_HIER”. Now the write operation will happen as per the new bd_kind. Any traces to be deleted before next register access can be done inside post_write.

```
  virtual task post_write( uvm_reg_item rw );

    //delete all newly added hierarchies (slices)
    //so the next access will be fresh,
    //not accumulated over the current access slices
    current_reg.clear_hdl_path("NEW_PAGED_HIER");

    //Put back the default path
    rw.path = UVM_DEFAULT_PATH;

    //Have some delay to see proper programing order in waveform
    #1ns;

  endtask: post_write
```

V. RESULT

We are already familiar with the simulation time improvements with the backdoor access. The practical limitation of using it for paged registers is being satisfactorily solved by the above method. We have seen around 50% reduction in the simulation time for some of our long-running tests by implementing this method in our project. These long running tests are having many configuration registers programmed though a slow peripheral. Moreover,

these tests are required to be iterated multiple times with the reset in between for many consistency checks which naturally makes the test duration really long. The backdoor approach really helped us in appreciably reducing the test duration and regression time.

ACKNOWLEDGMENT

CAD team of Analog Devices Inc. for the internal tool developments for many registers related automation