2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Modeling a Hierarchical Register Scheme with UVM

Joshua Hardy
Pensar Development
joshua.hardy@pensardevelopment.com

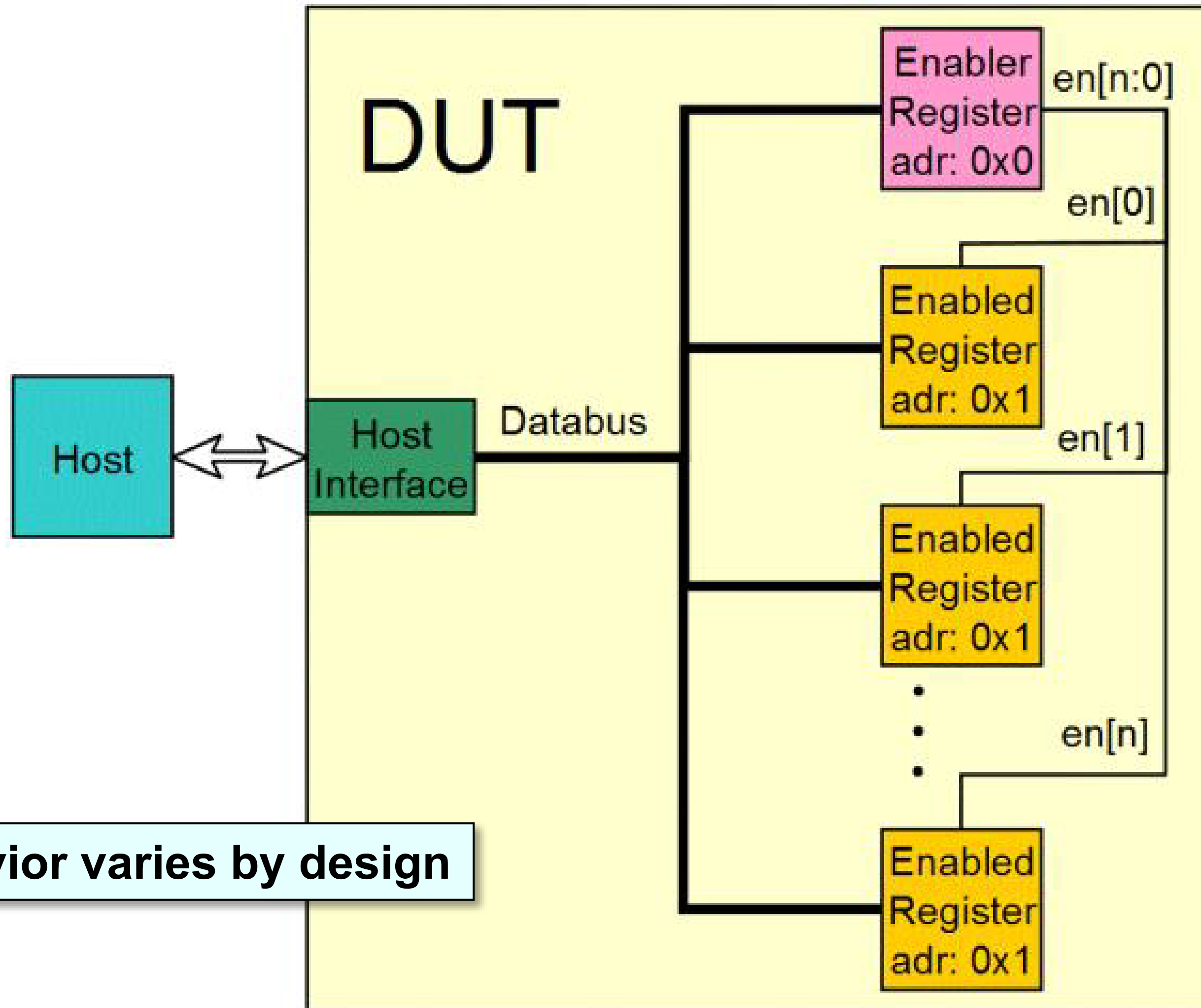
pensar

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2017

What are Hierarchical Registers?

- Replicated multiple times in the design
- Each copy uses the same address
- Each copy is enabled/disabled at a higher level of hierarchy
- Any combination of enables is allowed

Simple Example



Read behavior varies by design

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2017

UVM Register Model Challenge

- Each DUT register copy uses the same address but all UVM registers must have a unique address
- Designs often allow simultaneous access of multiple registers but basic UVM register model use targets a single register
- Multiple levels of hierarchy/enables increase complexity

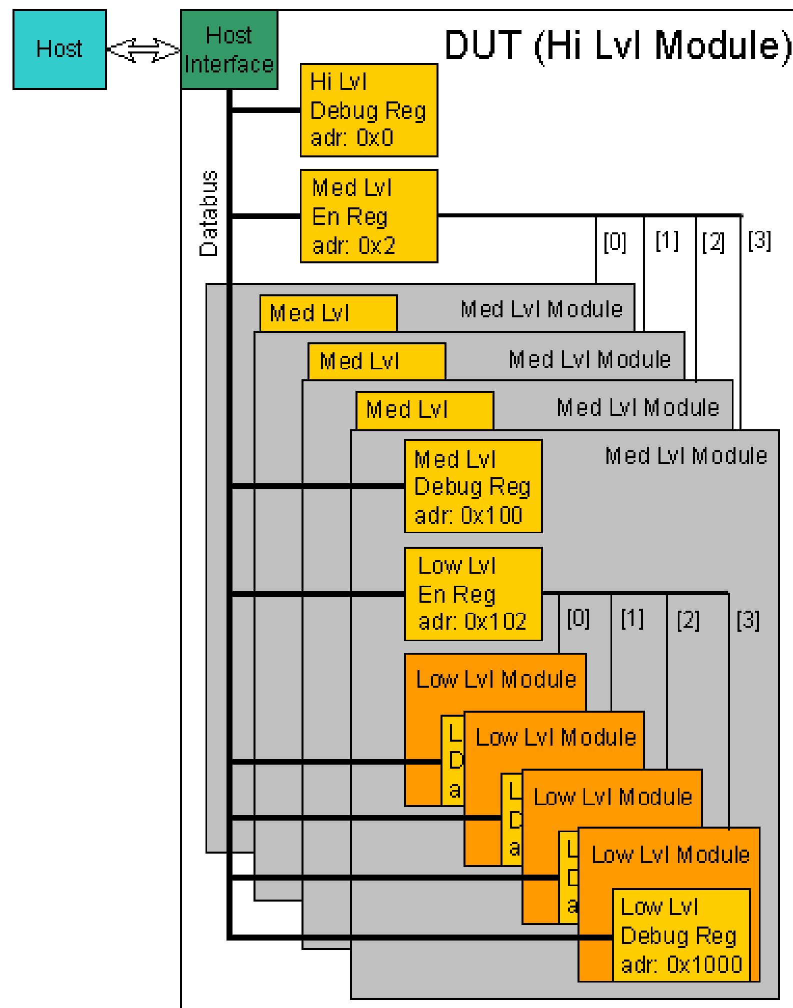
Solution Highlights

See paper for full description of solution.

- Create a unique *uvm_reg* instance in the UVM register model for each design register instance
- Use addresses outside the design's address range to allow each register to have a unique address
- Extend the *uvm_reg_predictor* to predict the multiple copies of the enabled registers
- Use callbacks to implement enabled behavior

Example: 3 Levels of Hierarchy

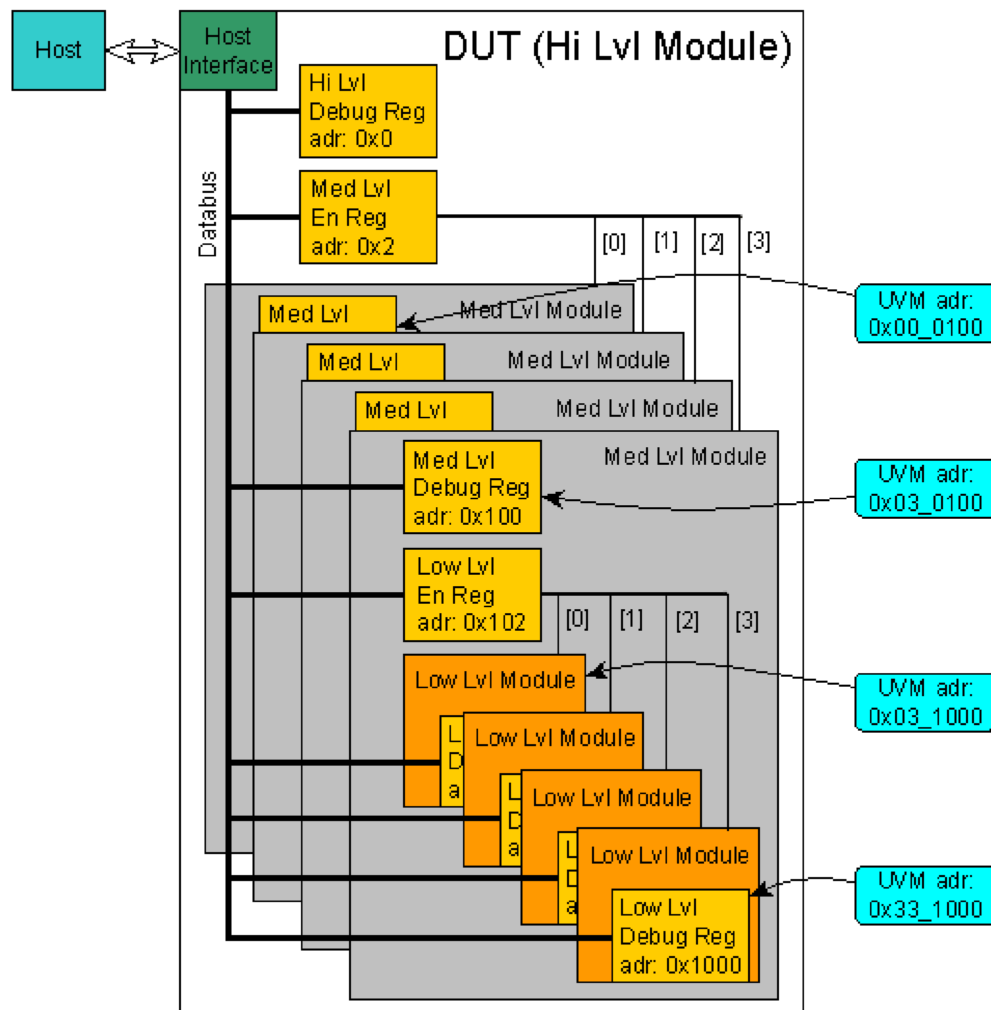
- 16-bit DUT Address width
- Enabled registers are ORed during reads



Create Unique uvm_reg Instances w/Unique Addrs

Steps 1 & 3 of solution in paper

- Register model hierarchy mimics DUT
  - Register blocks within register blocks
- Registers given unique addresses outside address range of DUT
  - 22-bit address width
  - 16 LSBs same as DUT



Create Enabler Register Fields

```
class med_lvl_en_reg extends uvm_reg;
   `uvm_object_utils(med_lvl_en_reg)

   rand uvm_reg_field med_lvl03, med_lvl02;
   rand uvm_reg_field med_lvl01, med_lvl00;

   function new(string name = "med_lvl_en_reg");
      super.new(name, 4, UVM_NO_COVERAGE);
   endfunction

   virtual function void build();
      med_lvl03 = uvm_reg_field::type_id::create("med_lvl03");
                     ...
      med_lvl03.configure(this, 1, 3, "RW", 0, 1'b0, 1, 1, 0);
                     ...
      med_lvl00.configure(this, 1, 0, "RW", 0, 1'b0, 1, 1, 0);
   endfunction
endclass
```

One field per lower level module that this register enables.

Normal register new() and build() methods.

Step 2 of solution in paper

Sequence Item and BFM Address Sizes

- 22-bit Sequence Item address width
  - Must be large enough for register model addresses
- 16-bit BFM address width
  - DUT must ignore extra MSBs of the register model

Steps 4 & 5 of solution in paper

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION 2017 UNITED STATES

Extend uvm_reg_predictor

Step 6 of solution in paper

Extending a predictor

```
class enabled_reg_pred #(type BUSTYPE)
  extends uvm_reg_predictor #(BUSTYPE);
   `uvm_component_param_utils(enabled_reg_pred#(BUSTYPE))

   function new (string name = "enabled_reg_pred",
                 uvm_component parent);
      super.new(name, parent);
   endfunction : new

   function void write(BUSTYPE tr);
      // If we are accessing a hi level register,
      // then we predict as normal
      if (tr.adr < 'h100) begin
         super.write(tr);
      ...
   endfunction : write
endclass : enabled_reg_pred
```

Else: shown on next slide

6. Extend uvm_reg_predictor (cont.) – Lower Level Regs

```
// Predict for each copy.
// Register callbacks will implement enabled behavior
end else begin
    int unsigned low_loop_max = 1;
    for (int med = 0; med < 4; med ++) begin
        if (tr.adr[15:0] >= `h1000) begin
            low_loop_max = 4;
        end

        for (int low = 0; low < low_loop_max; low++) begin
            // Call base predictor write function
            tr.adr = tr.adr[15:0]+(med*'h1_0000)+(low*'h10_0000);
            super.write(tr);
        end
    end
end
```

Generate the unique UVM address for each register copy

Call write() for each register copy

Register Callbacks for Enabled Behavior

Step 7 of solution in paper

```
class enabled_reg_cbs extends uvm_reg_cbs;

   // Handle(s) to the field(s) that enable
   // this copy of this enabled field.
   local uvm_reg_field m_enabler_reg_field_h[$];

   // Handle(s) to the field(s) that enable
   /// the other copies of this enabled field.
   local uvm_reg_field m_other_enabler_reg_field_h[$];

   . . .
enclass
```

These queues are examined in the post_predict function to implement enabled behavior

Handles assigned in the test. See paper for example of how to assign handles.

Register Callbacks for Enabled Behavior (cont.)

```
virtual function void post_predict(
   input uvm_reg_field fld,
   input uvm_reg_data_t previous,
   inout uvm_reg_data_t value,
   input uvm_predict_e kind,
   input uvm_path_e path,
   input uvm_reg_map map);

if (path inside {UVM_FRONTDOOR, UVM_PREDICT}) begin
   bit is_enabled = 1;
   // If any enabler field is not set then don't update
   foreach(m_enabler_reg_field_h[i]) begin
      if (!m_enabler_reg_field_h[i].get_mirrored_value()) begin
         value = previous; is_enabled = 0;
         break;
      end
   end
   . . .
```

First check if we are doing a frontdoor operation.

Check each corresponding enabler field

Register Callbacks for Enabled Behavior (cont.)

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION 2017 UNITED STATES

If this frontdoor operation is a read

```
. . .
if (kind == UVM_PREDICT_READ) begin
// If any enabler field for other copies of this field is set
// then we do not update the mirrored value
foreach(m_other_enabler_reg_field_h[i]) begin
   if (m_other_enabler_reg_field_h[i].get_mirrored_value()) begin
      value = previous;
      if (is_enabled) begin
         `uvm_info("EVENT_REG_CBS", $sformatf("Frontdoor rd
            made with multiple fields enabled including %s!
            Mirrored values not updated", fld.get_full_name()),
            UVM_LOW)
      end
      break;
   end
end
end
```

Message to inform user that mirrored value is not updated.

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2017

Solution Results

- Frontdoor operations on registers with the same DUT address will:
  - Initiate the same transaction to the DUT
  - Predict all enabled registers at that address
  - Update each register based on enables
    - Registers only update as appropriate during reads

- Backdoor operations allow manipulation of each register instance individually

See paper for full description of solution.

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2017

Thank You ☺

- Questions?