Model Unconventional Register Structures

Using UVM Register Access Layer Hooks and Callbacks

Elihai Maicas
Intel

June 7, 2017
SNUG Israel

Agenda

Background

Introduction

The problem

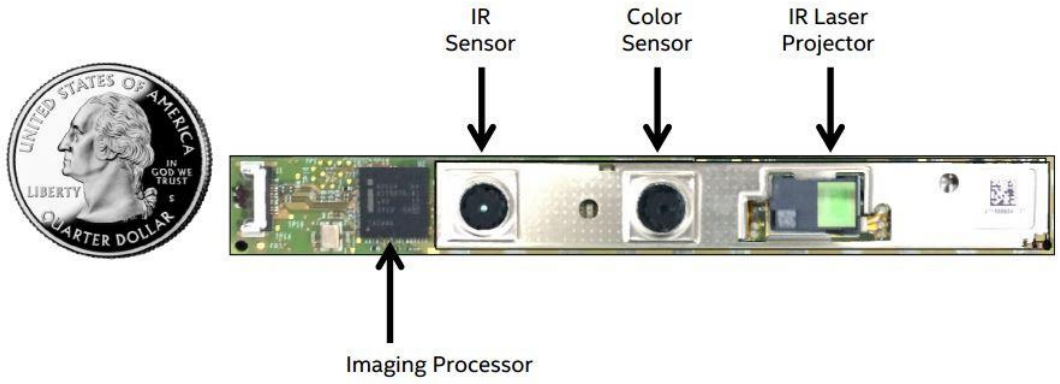
Unconventional Register Structures

Register Update Flow

Summary

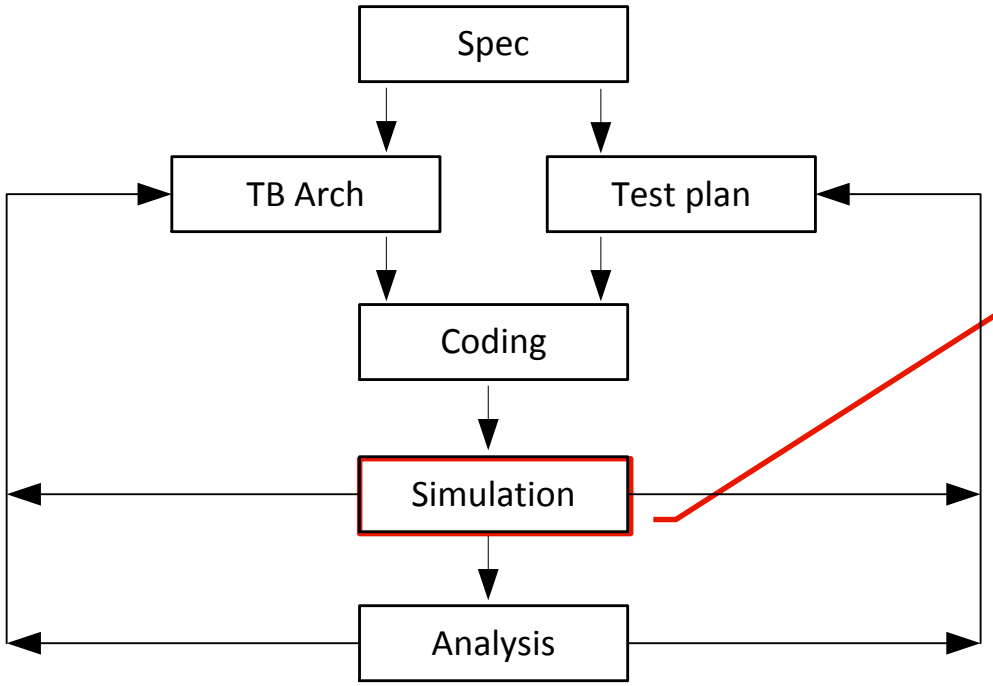
Background

- Intel® RealSense™ camera fits remarkable technology into a small package. There are three cameras that act like one - a 1080p HD camera, an infrared camera, and an infrared laser projector - they “see” like the human eye to sense depth and track human motion



Background

- General work flow (verification is done at the sub-system level):



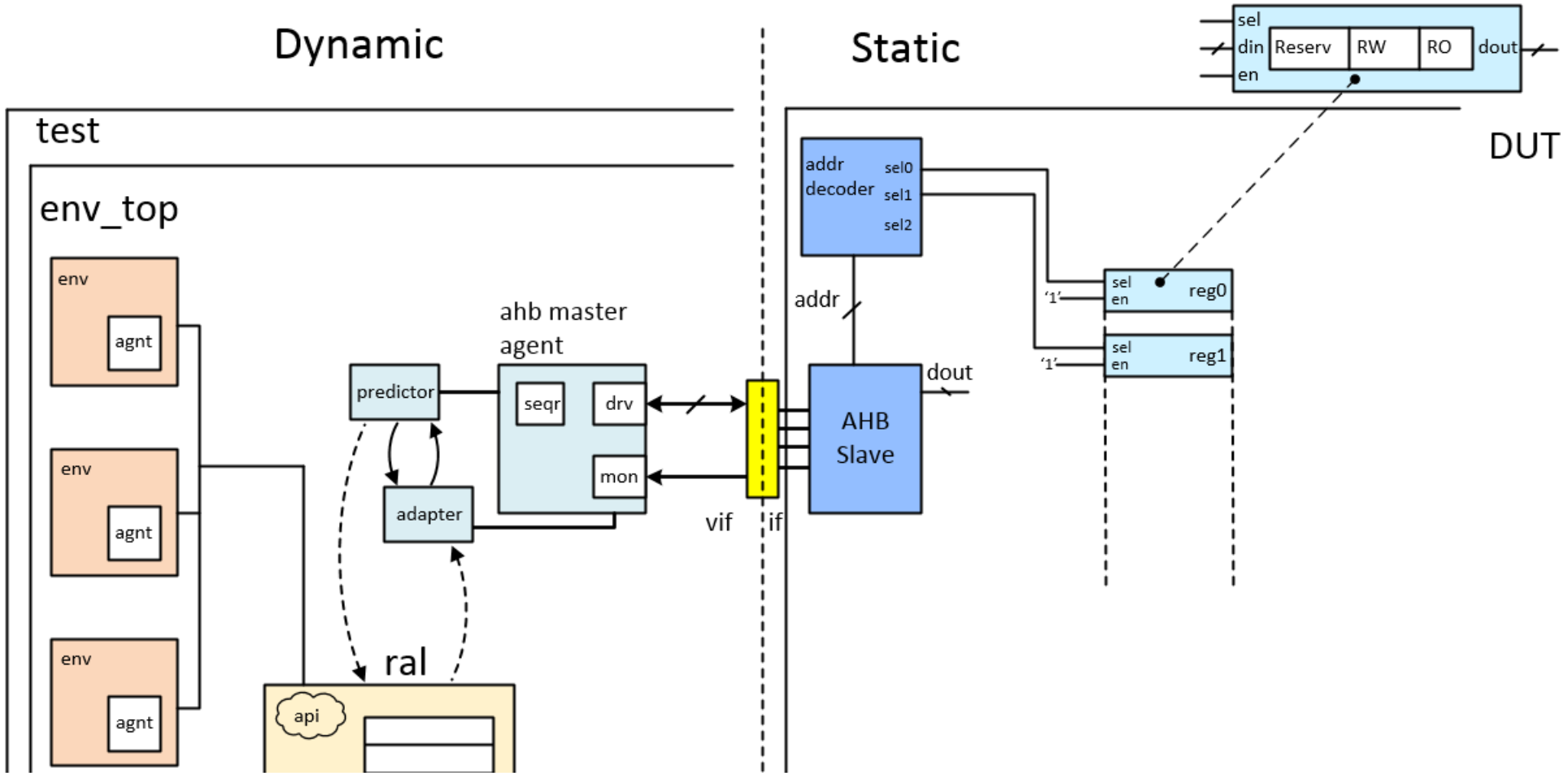
- Randomize configuration
- Init DUT
- Randomize and start stimuli

- **On-the-fly configuration:**
  - **Laser projection related logic**
  - **Multiple frames in pipe**

UVM version: 1.1d
VCS version: vcs-mx-L-2016.06-SP1

Introduction

- DUT-Env testbench



The problem

- Not all registers types exist in the pre-defined field access policies

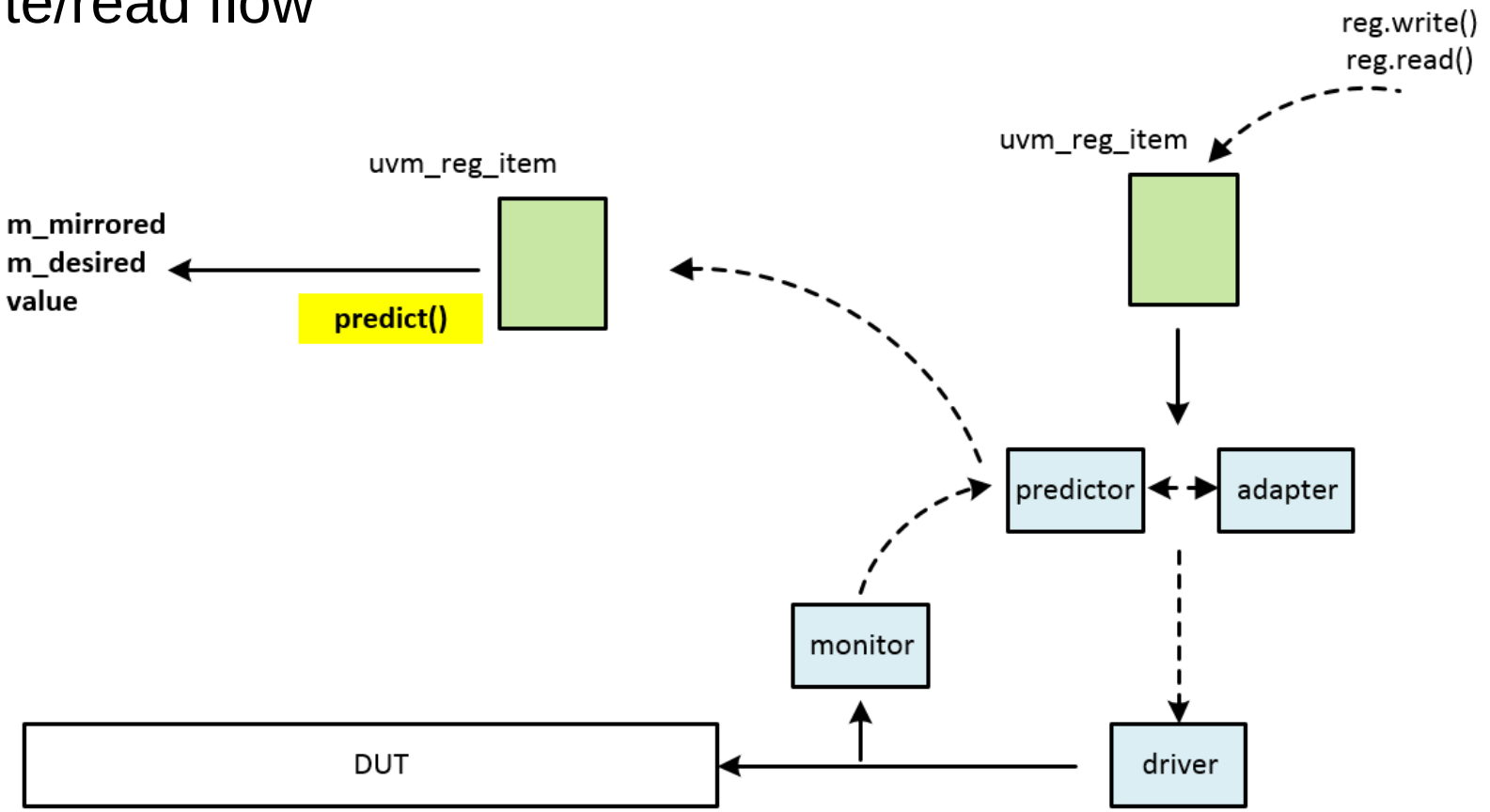
Access Policy	Description	Effect of a Write on Current Field Value	Effect of a Read on Current Field Value	Read-back Value
RO	Read Only	No effect.	No effect.	Current value
RW	Read, Write	Changed to written value.	No effect.	Current value
RC	Read Clears All	No effect.	Sets all bits to 0's.	Current value
RS	Read Sets All	No effect.	Sets all bits to 1's.	Current value
WRC	Write, Read Clears All	Changed to written value.	Sets all bits to 0's.	Current value
WRS	Write, Read Sets All	Changed to written value.	Sets all bits to 1's.	Current value
WC	Write Clears All	Sets all bits to 0's.	No effect.	Current value
WS	Write Sets All	Sets all bits to 1's.	No effect.	Current
WSRC	Write Sets All, Read Clears All	Sets all bits to 1's.		
	Write Clears All,	Sets all bits to 0'		

- Mainly such the rely upon other fields values prior to final update

(Very) quick RAL overview

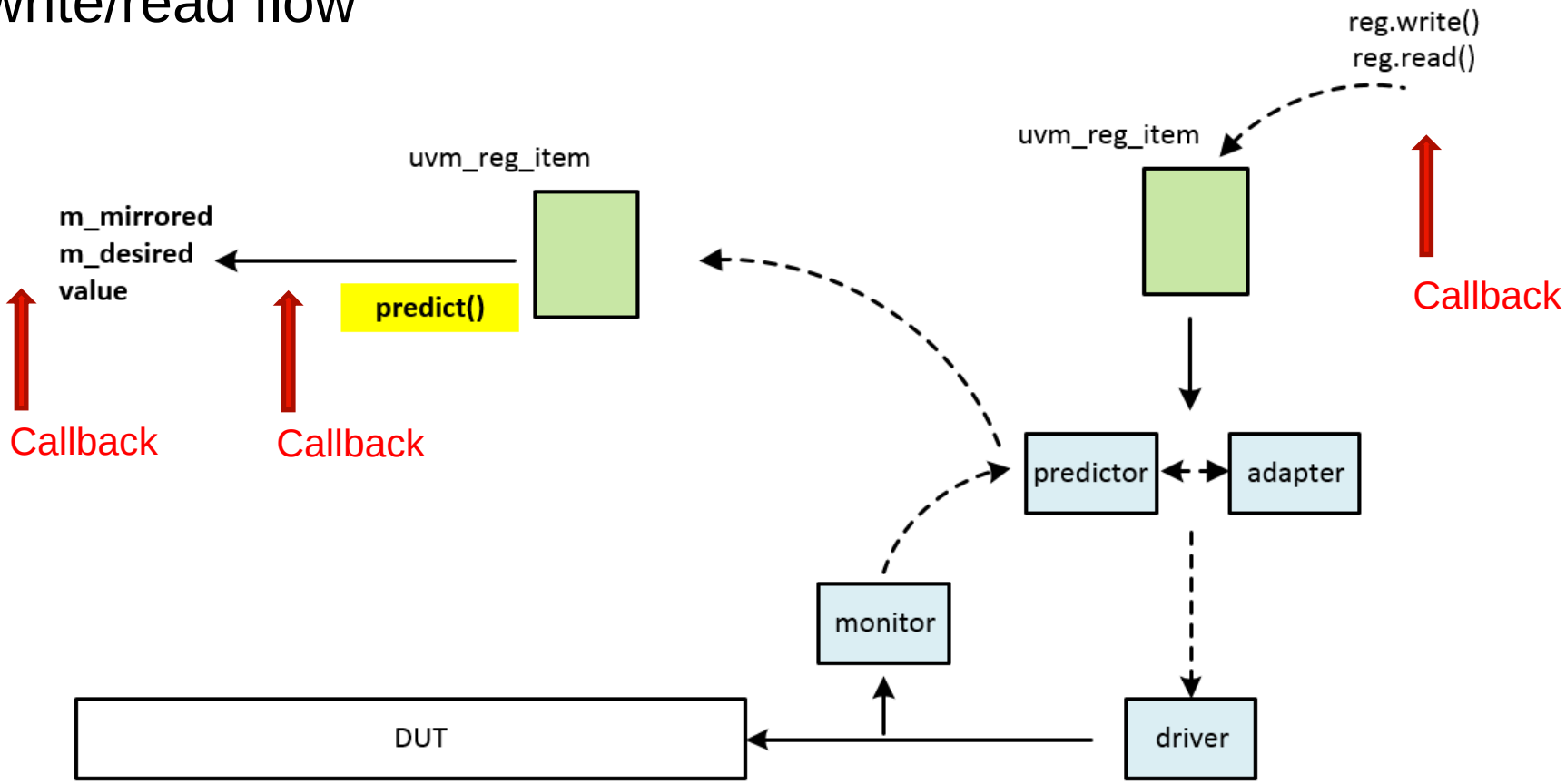
Introduction
RAL overview

- Register write/read flow



Introduction
RAL overview

- Register write/read flow



Introduction
RAL overview

- Register callback (uvm_reg_field.svh)

```
function void uvm_reg_field::do_predict(uvm_reg_item      rw,
                                        uvm_predict_e     kind = UVM_PREDICT_DIRECT,
                                        uvm_reg_byte_en_t be = -1);
 ....
 case (kind)

   UVM_PREDICT_WRITE:
     begin
       uvm_reg_field_cb_iter cbs = new(this);

       if (rw.path == UVM_FRONTDOOR || rw.path == UVM_PREDICT)
          field_val = XpredictX(m_mirrored, field_val, rw.map);

       m_written = 1;

       for (uvm_reg_cbs cb = cbs.first(); cb != null; cb = cbs.next())
          cb.post_predict(this, m_mirrored, field_val,
                          UVM_PREDICT_WRITE, rw.path, rw.map);
 ....
    // update the mirror with predicted value
    m_mirrored = field_val;
    m_desired  = field_val;
    this.value = field_val;

 endfunction: do_predict
```

Introduction
RAL overview

- Register callback (uvm_reg_cbs.svh)

```
virtual class uvm_reg_cbs extends uvm_callback;

   function new(string name = "uvm_reg_cbs");
      super.new(name);
   endfunction

   ....

   virtual task pre_write(uvm_reg_item rw); endtask
   virtual task post_write(uvm_reg_item rw); endtask
   virtual task pre_read(uvm_reg_item rw); endtask
   virtual task post_read(uvm_reg_item rw); endtask

   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);

   endfunction
```

Called upon active agent operation

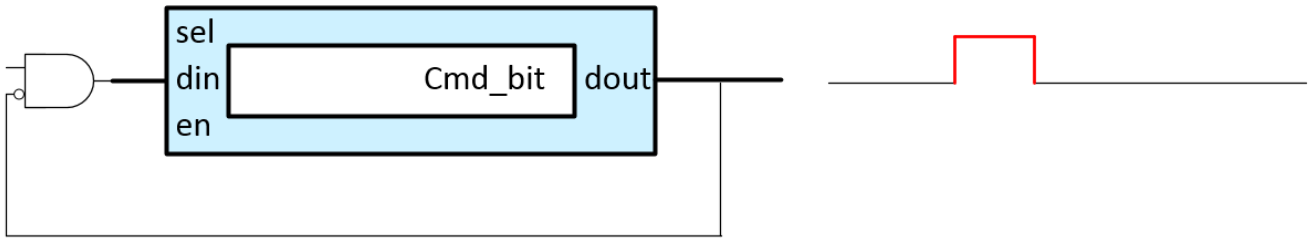
Called always

Unconventional Registers

Command (non-sticky) Register

Unconventional Registers

- Command (non-sticky) Register
  - HW Schematic Behavior

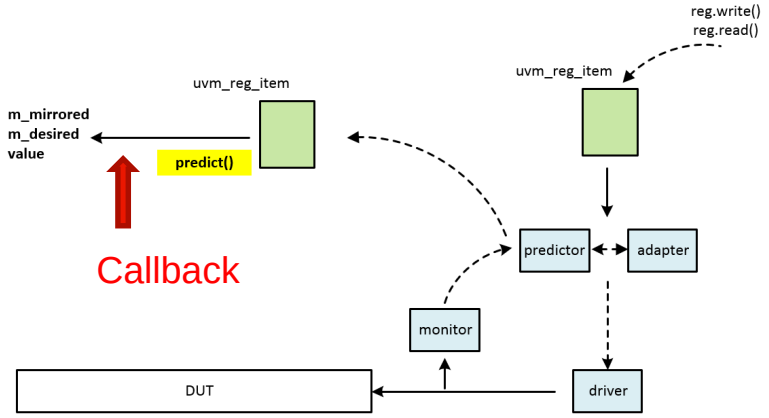


  - RAL register is defined as RW register, so after the first register write the model is out of sync

Unconventional Registers

- Command (non-sticky) Register
  - Defining post_predict callback

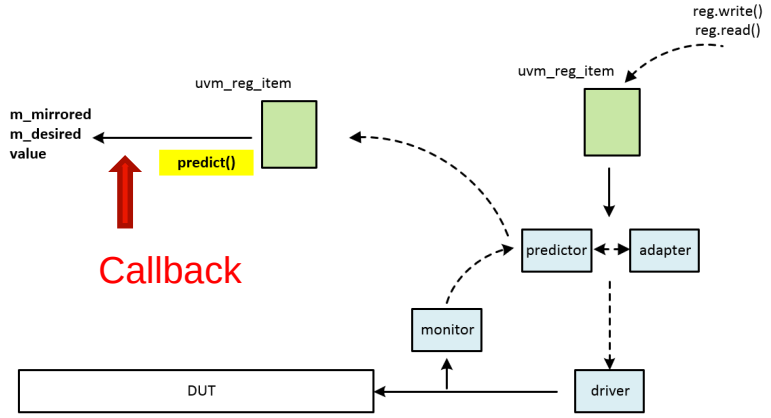
```
//------------------------------------------------------------------------------
// post_predict
//------------------------------------------------------------------------------
virtual function void post_predict( input uvm_reg_field  fld,
                                    input uvm_reg_data_t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

  if (kind==UVM_PREDICT_WRITE && value==1'b1) begin
    `uvm_info(get_type_name(),"Detected level register field cmd write ", UVM_MEDIUM)
    m_cmd_field_prc_trigger.trigger($time/1ns);

    // reset the predict value, so ral will be updated
    if (m_reset_predict_val) begin
      value = 0;
    end
  end

endfunction
```



Unconventional Registers

- Command (non-sticky) Register
  - Create and register the callback class

```
// create hash table with all cmd fields
m_cmd_register_field_cb["RegsAfeSeqStart"] = new("RegsAfeSeqStart");

//..//

// assign each cmd cb with matching cmd field
uvm_reg_field_cb::add( regmodel.afe_afe_seq.RegsAfeSeqStart.RegsAfeSeqStart,
                       m_cmd_register_field_cb["RegsAfeSeqStart"]);
```

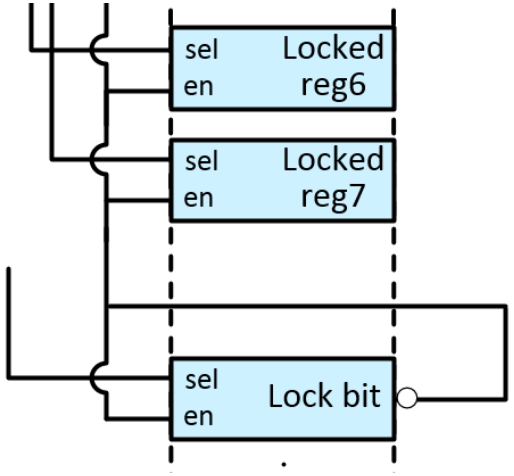


Unconventional Registers

Write Locked Register

Unconventional Registers

- Write Locked Register
  - HW Schematic Behavior

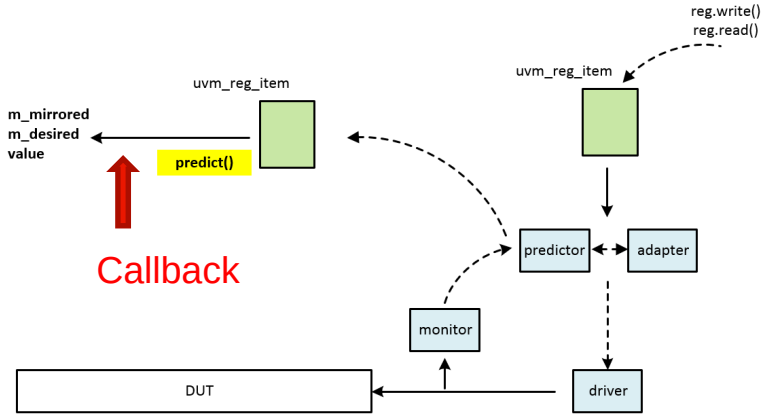


  - RAL register is defined as RW register, so any successful write access modified its value

Unconventional Registers

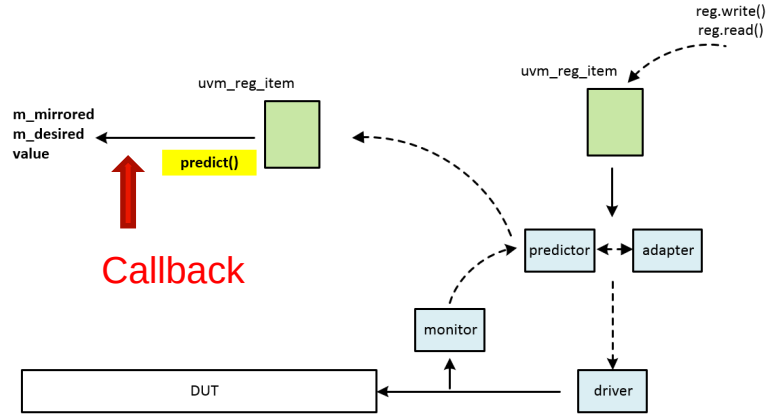
- Write Locked Register
  - Defining post_predict callback

```
//------------------------------------------------------------------------------
// post_predict
//------------------------------------------------------------------------------
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);

  // local fields

  if (kind==UVM_PREDICT_WRITE) begin

    // predict was implicitly executed after a front-door write
    if ( (lock_field_val==1) ) begin
      // revert to previous value if protected
      value = previous;
    end
  end

endfunction
```



Unconventional Registers

- Write Locked Register
  - Defining post_predict callback



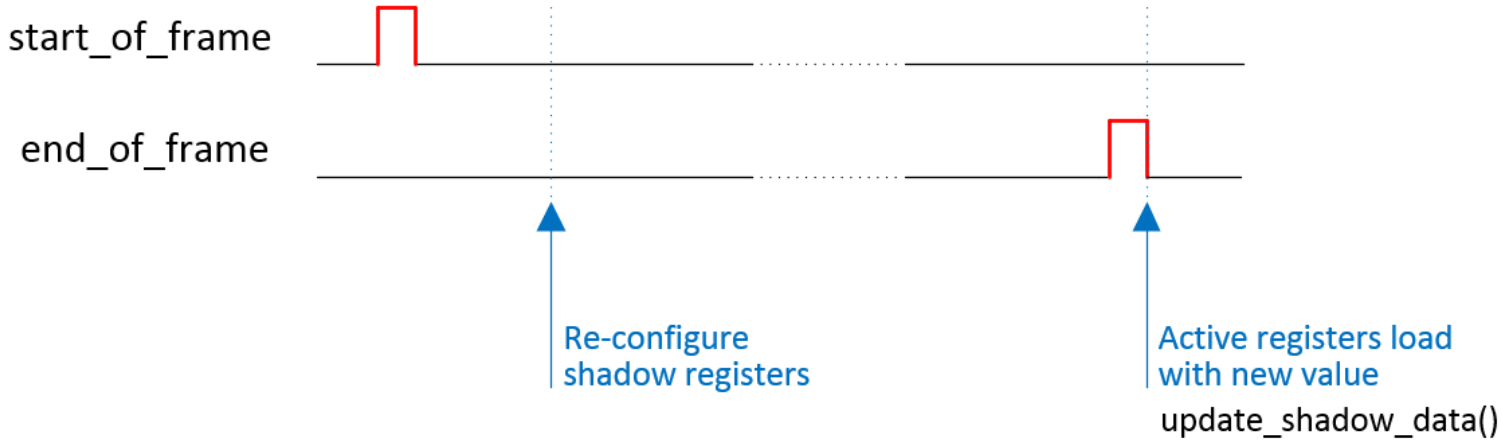

```
//------------------------------------------------------------------------------
// post_predict
//------------------------------------------------------------------------------
virtual function void post_predict( input uvm_reg_field  fld,
                                    input uvm_reg_data_t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

  // local fields

  if (kind==UVM_PREDICT_WRITE) begin

    // predict was implicitly executed after a front-door write
    if ( (lock_field_val==1) && (dbg_strap_val==0) && (m_active_master_ind.m_master!=AHB_SEC) ) begin
      // revert to previous value if protected
      value = previous;
    end
  end

endfunction
```

Unconventional Registers

Shadow Update Register

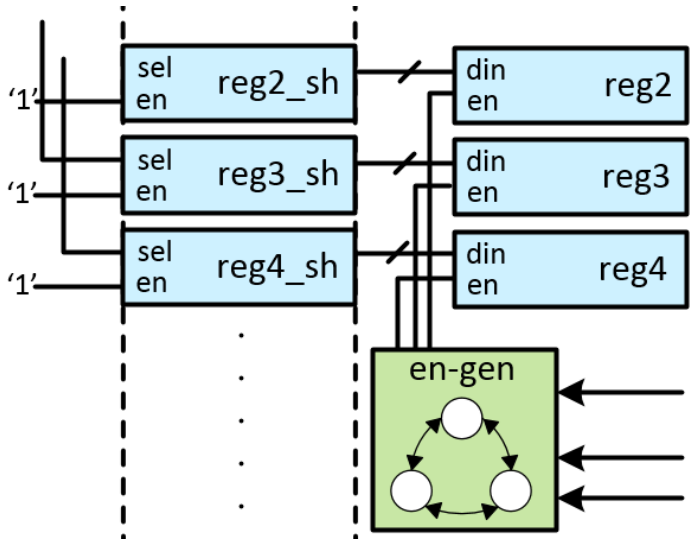
Unconventional Registers

- Shadow Update Register
  - HW Timing Diagram



Unconventional Registers

- Shadow Update Register
  - HW Schematic Behavior

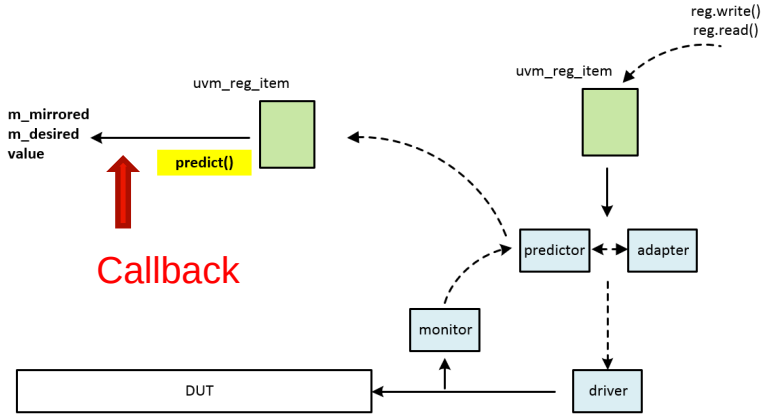


  - RAL register is defined as RW register, so any successful write access modified its value

Unconventional Registers

- Shadow Update Register
  - Defining post_predict callback

```
//------------------------------------------------------------------------------
// post_predict
//------------------------------------------------------------------------------
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 (kind==UVM_PREDICT_WRITE) begin

    m_shadow_value  = value;
    value           = previous;

  end
endfunction
```



Unconventional Registers

- Shadow Update Register
  - Defining update_shadow_data method

```
//------------------------------------------------------------------------------
// update_shadow_data
//------------------------------------------------------------------------------
virtual function void update_shadow_data ();
  if (m_shadow_ready) begin
    m_fld.predict(.value(m_shadow_value), .kind(UVM_PREDICT_DIRECT));
  end

endfunction : update_shadow_data
```

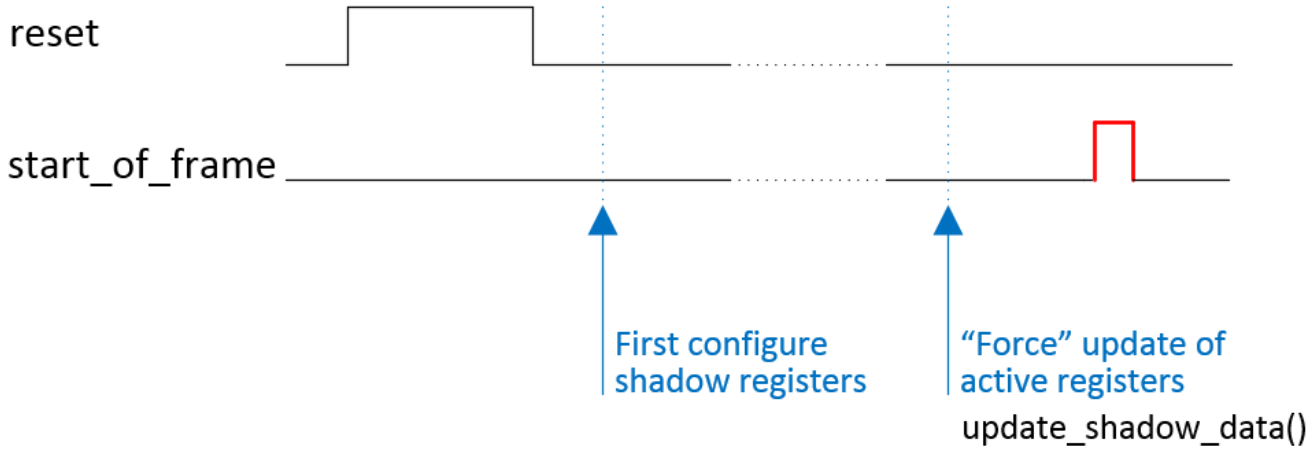
Unconventional Registers

- Shadow Update Register - trigger
  - Calling update_shadow_data method – upon HW trigger

```
forever begin
  m_shadow_update_trigger[cur_evt_name_au].wait_trigger();
  `uvm_info("ma_shadow_update_helper",$psprintf("SHADOW UPDATE TRIGGER - got trigger (%s)",cur_evt_name_au), UVM_LOW)

  // go over all the fields in the shadow trigger group and update their shadow data
  for (bit b=m_reg_cbs_from_event.first(reg_cbs_evt_name); b==1; b=m_reg_cbs_from_event.next(reg_cbs_evt_name)) begin
    foreach (m_reg_cbs_from_event[reg_cbs_evt_name][ii]) begin
      tmp_q =  m_shadow_update_trigger_fields[cur_evt_name_au].find_first_index(field_name)
        with (field_name == m_reg_cbs_from_event[reg_cbs_evt_name][ii].get_name());
      if (tmp_q.size() != 0) begin
        m_reg_cbs_from_event[reg_cbs_evt_name][ii].update_shadow_data();
      end
    end // foreach
  end // for
end // forever
```

Unconventional Registers

- Shadow Update Register - immediate
  - HW Timing Diagram



Unconventional Registers

- Shadow Update Register - immediate
  - Calling update_shadow_data method – upon update immediate command

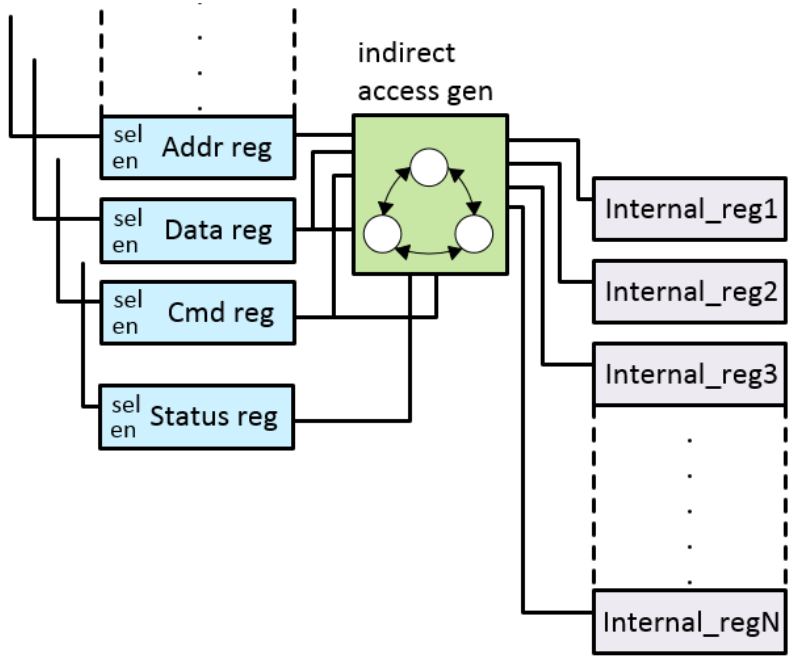
```
//wait for update immediate reg write and then update all the fields in the group
for (bit b=m_imm_update_cb.first(cur_evt_name); b==1; b=m_imm_update_cb.next(cur_evt_name)) begin
  automatic string cur_evt_name_au = cur_evt_name;
  fork
    forever begin
      m_imm_update_cb[cur_evt_name_au].m_cmd_field_prc_trigger.wait_trigger();
      `uvm_info("ma_shadow_update_helper",$psprintf("SHADOW UPDATE IMMEDIATE - got trigger (%s)",cur_evt_name_au), UVM_LOW)
      foreach (m_reg_cbs_from_imm[cur_evt_name_au][ii]) begin
        m_reg_cbs_from_imm[cur_evt_name_au][ii].update_shadow_data();
      end // foreach
    end // forever
  join_none
end
```

Unconventional Registers

Indirect Access Register

Unconventional Registers

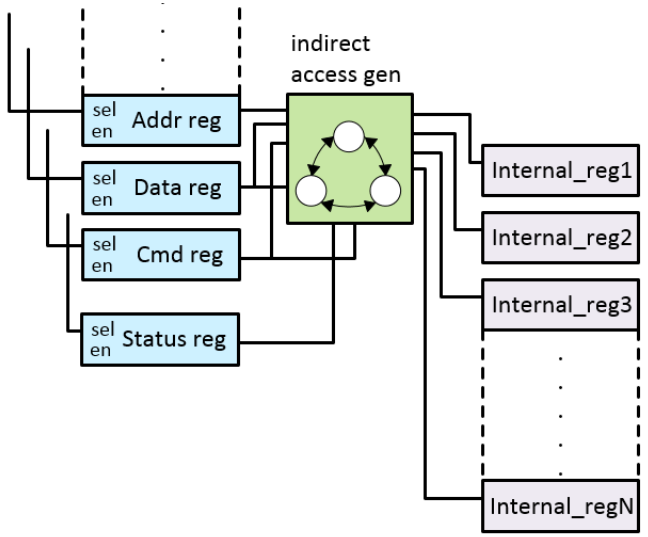
- Indirect Register Access
  - HW Schematic Behavior



  - Internal regs does not have direct access, but may have a RAL block class to be used by the env

Unconventional Registers

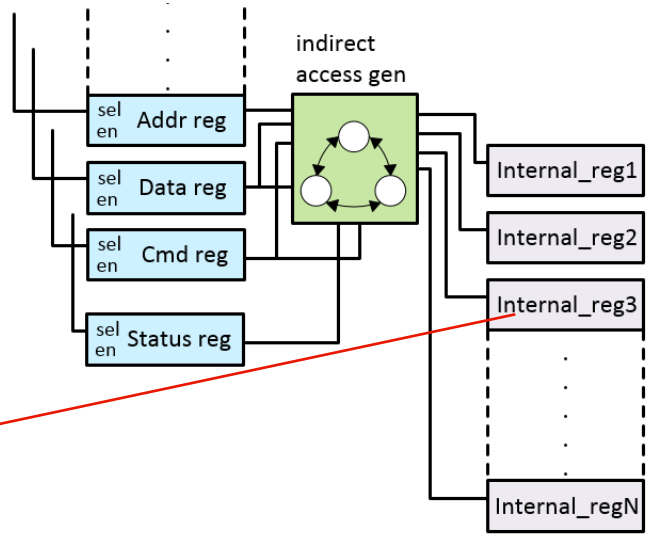
- Indirect Register Access
  - Defining post_predict callback




```
//------------------------------------------------------------------------------
// post_predict
//------------------------------------------------------------------------------
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 (kind==UVM_PREDICT_READ && value==1'b0 && previous==1'b1) begin
    `uvm_info(get_type_name(),"Detected indirect access command completion ", UVM_MEDIUM)

    // fetch target register by address
    m_indirect_target_reg = map.get_reg_by_offset(m_ss_phy_indirect_acc_addr.get_mirrored_value());

    // update target register
    m_indirect_target_reg.predict(.value(m_ss_phy_indirect_acc_data.get_mirrored_value()), .kind(UVM_PREDICT_DIRECT));
  end
```

Unconventional Registers

- Indirect Register Access
  - Defining post_predict callback

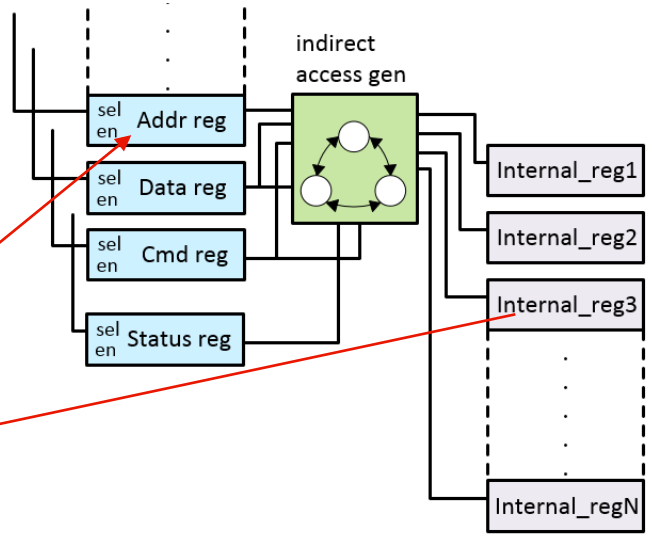



```
//------------------------------------------------------------------------------
// post_predict
//------------------------------------------------------------------------------
virtual function void post_predict( input uvm_reg_field  fld,
                                    input uvm_reg_data_t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

  if (kind==UVM_PREDICT_READ && value==1'b0 && previous==1'b1) begin
    `uvm_info(get_type_name(),"Detected indirect access command completion ", UVM_MEDIUM)

    // fetch target register by address
    m_indirect_target_reg = map.get_reg_by_offset(m_ss_phy_indirect_acc_addr.get_mirrored_value());

    // update target register
    m_indirect_target_reg.predict(.value(m_ss_phy_indirect_acc_data.get_mirrored_value()), .kind(UVM_PREDICT_DIRECT));
  end
```

Unconventional Registers

- Indirect Register Access
  - Defining post_predict callback

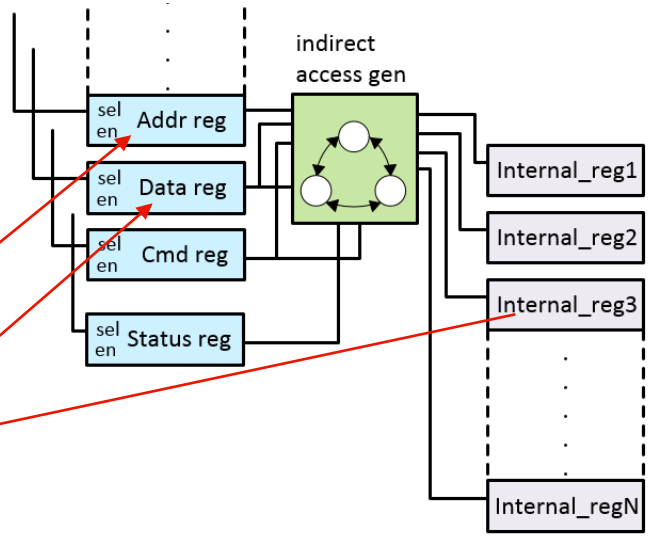
```
//------------------------------------------------------------------------------
// post_predict
//------------------------------------------------------------------------------
virtual function void post_predict( input uvm_reg_field  fld,
                                    input uvm_reg_data_t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

  if (kind==UVM_PREDICT_READ && value==1'b0 && previous==1'b1) begin
    `uvm_info(get_type_name(),"Detected indirect access command completion ", UVM_MEDIUM)

    // fetch target register by address
    m_indirect_target_reg = map.get_reg_by_offset(m_ss_phy_indirect_acc_addr.get_mirrored_value());

    // update target register
    m_indirect_target_reg.predict(.value(m_ss_phy_indirect_acc_data.get_mirrored_value()), .kind(UVM_PREDICT_DIRECT));
  end
```



Unconventional Registers

- Indirect Register Access
  - Defining post_predict callback

```
//------------------------------------------------------------------------------
// post_predict
//------------------------------------------------------------------------------
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 (kind==UVM_PREDICT_READ && value==1'b0 && previous==1'b1) begin
    `uvm_info(get_type_name(),"Detected indirect access command completion ", UVM_MEDIUM)

    // fetch target register by address
    m_indirect_target_reg = map.get_reg_by_offset(m_ss_phy_indirect_acc_addr.get_mirrored_value());

    // update target register
    m_indirect_target_reg.predict(.value(m_ss_phy_indirect_acc_data.get_mirrored_value()), .kind(UVM_PREDICT_DIRECT));
  end
```

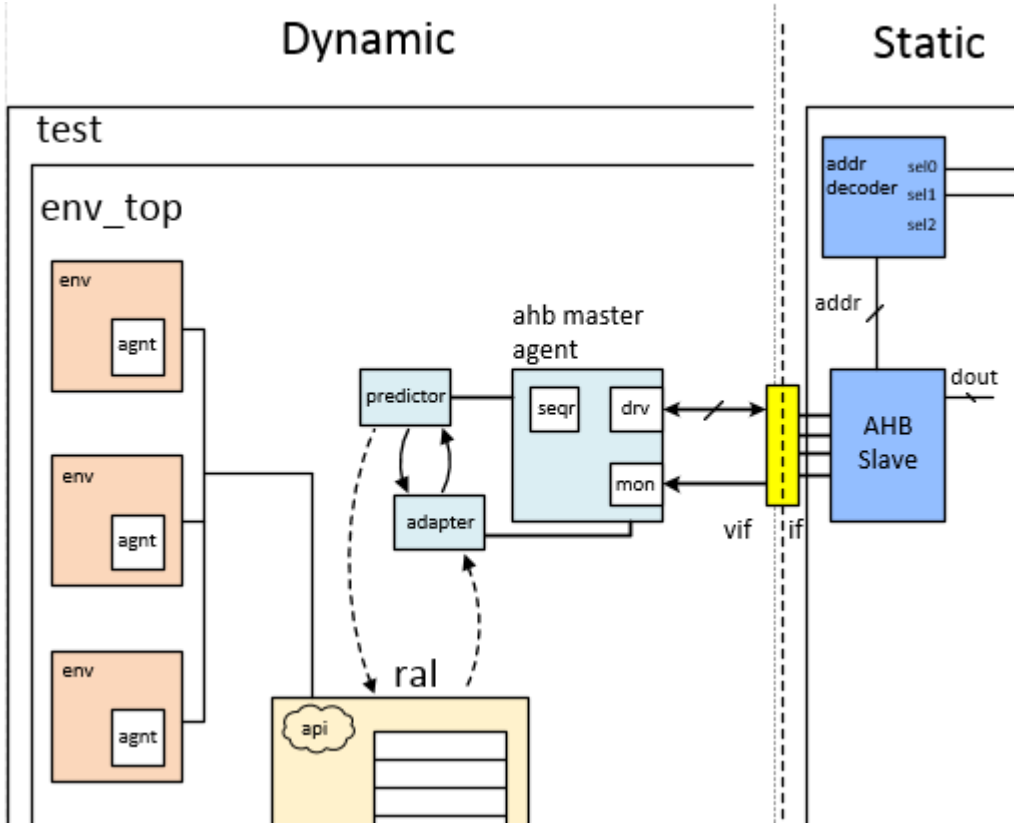


Register Update Flow

Prevent back-to-back access

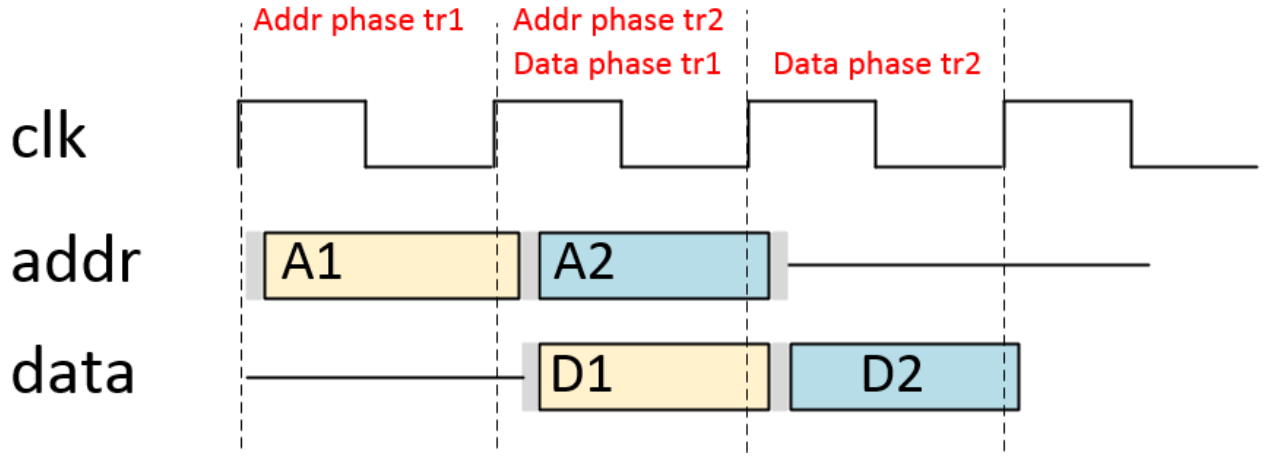
Register Update Flow

- Prevent back-to-back transactions
  - In most tests, a protocol VIP is used instead of the micro-controller



Register Update Flow

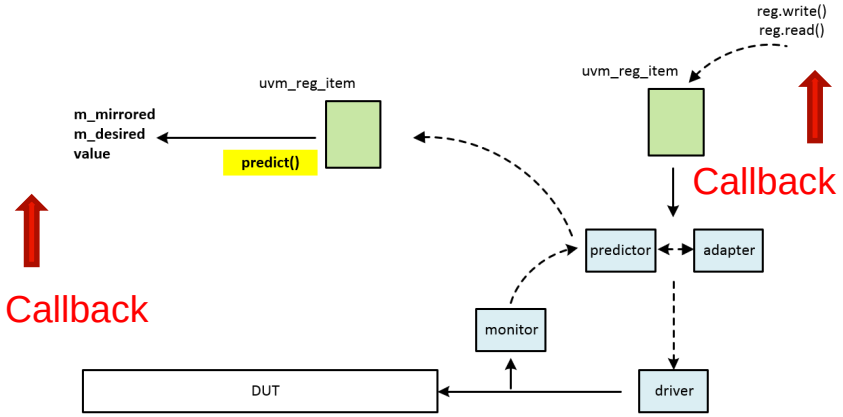
- Prevent back-to-back transactions
  - If more than one register access is done in the same simulation tick the VIP may pipeline two accesses and create a protocol valid back-to-back transactions
  - Some slaves does not support such transaction



Register Update Flow

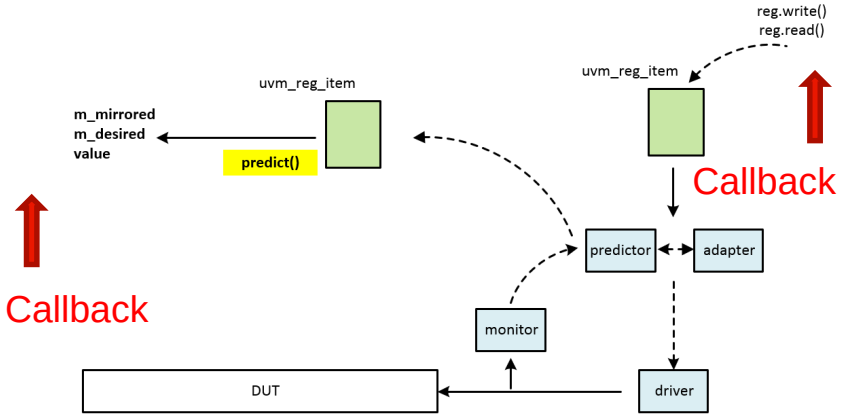
- Prevent back-to-back transactions
  - Defining the callback class methods

```
class ahb_no_b2b_cb extends uvm_reg_cbs;

  protected semaphore m_reg_access_sem;   // In order to prevent b2b access, every
                                          // to the others, by getting the semapho
                                          // back in the post_* callback


  //---------------------------------------------------------------------------
  // new
  //---------------------------------------------------------------------------
  function new (string name = "unnamed-ahb_no_b2b_cb");
    super.new(name);
    m_reg_access_sem = new(1);
  endfunction : new

  task pre_write(uvm_reg_item rw);  m_reg_access_sem.get();  endtask : pre_write
  task post_write(uvm_reg_item rw); m_reg_access_sem.put();  endtask : post_write

  task pre_read(uvm_reg_item rw);   m_reg_access_sem.get();  endtask : pre_read
  task post_read(uvm_reg_item rw);  m_reg_access_sem.put();  endtask : post_read

endclass : ahb_no_b2b_cb
```



Register Update Flow

- Prevent back-to-back transactions
  - Registering the callback class to all RAL registers




```
m_ahb_no_b2b_cb = new("m_ahb_no_b2b_cb");

m_regmodel.get_registers(loc_regs_h);

// assigning each reg with this cb class, that actually forces register ahb access
// to be mutually exclusive by preventing b2b access
if($test$plusargs("DISABLE_AHB_B2B")) begin
  foreach(loc_regs_h[ii]) begin
    uvm_callbacks #(uvm_reg,ahb_no_b2b_cb)::add(loc_regs_h[ii],m_ahb_no_b2b_cb);
  end
end
```

Register Update Flow

- Reset types that can be model using callback/trigger
  - SW reset – top level
  - SW reset – block level
  - Watch-Dog Timer reset
  - Internal HW reset

Summary

- The UVM Register Layer models various type of registers

- Some types does exist out of the box:
  - Command (non-sticky) Register
  - Write-lock Register
  - Shadow-update Register

- But – There are pre-defined callbacks that enable the user to model such types, and other unique SOC behaviors

Thank You