February 28 – March 1, 2012

OVM & UVM Techniques for On-the-fly Reset

By

Muralidhara Ramalingaiah         Boobalan Anantharaman
Design Engineer Staff Sr.       Design Engineer Sr.
Cypress Semiconductors
Agenda

• On-The-Fly Reset
• What should happen?
• OVM Reset Techniques
• Issues with OVM Reset Techniques
• Solutions for OVM Reset Technique Issues
• UVM Reset Techniques
• OVM to UVM migration – Tips for reset
• Conclusion
On-The-Fly Reset

• When the DUT is reset during normal operation, the testbench must act accordingly and must not behave abnormally or give ambiguous results.

• Hard Reset
  – Reset through External Reset I/O

• Soft Reset
  – Reset through Firmware Register Write
What should happen?

- Following things should happen in Test bench
  - Scoreboard variables should get reset
  - Monitor should recognize it and qualify the transactions
  - Coverage update logic should not trigger any transition coverage except reset coverage
  - Driver should recognize it and stop transaction at right time
  - Data Checkers should delete the transactions which are collected in FIFOs or QUEUEs
  - Protocol Checkers should recognize On-the-fly reset
OVM Reset Techniques

- Scoreboard variables get reset by Reset Monitor event(s)
- Interface OVCs/Agents works independent of reset monitor event(s)

**RESET monitor**

```plaintext
//reset event creation
oep = oep.get_global_pool();
is_reset = oep.get("RESET_EVENT");
//reset event creation
run() begin
  forever begin
    @(negedge rstagent_if_monitor.reset)
      is_reset.trigger();
    end
  end
end
```

**Scoreboard**

```plaintext
run() begin
  ovm_event_pool oep;
  ovm_event is_reset;
  oep = oep.get("RESET_EVENT");
  forever
    begin //{
      is_reset.wait_trigger();
      register_map.reset();
    end //}
end
```
Issues with OVM Reset Techniques

• Issue 1
  - Drivers don’t recognize the on-the-fly reset in between transactions

• Issue 2
  - Irrespective of the on-the-fly reset, monitor will send data to scoreboard/coverage

• Issue 3
  - During on-the-fly reset, it is difficult control the sequencers of the sub-sequences from the virtual_sequence
Issues with OVM Reset Techniques Contd..

- Issue 1
  - Drivers don’t recognize the on-the-fly reset in between transactions

```plaintext
Driver

task run();
fork
get_and_drive();
reset_signals();
join_none
```
Issues with OVM Reset Techniques
Contd..

Drive Task

```verilog
task get_and_drive();
    @(wait for reset to finish);
    ovm_report_info(get_type_name(), "RESET DISABLE");
    forever begin
        seq_item_port.get_next_item(req); //or try_next_item
        $cast(rsp, req.clone());
        rsp.set_id_info(req);
        //If try next item then need to qualify is the item is there??
        drive_transfer(rsp);
        // Advance clock
        @(posedge intf.cb);
        send_idle(rsp);
        seq_item_port.item_done(rsp);
    end
endtask : get_and_drive
```

Reset Task

```verilog
task reset_signals();
    ovm_report_info(get_type_name(), "reset_signals ...");
    // USER: Add implementation
    forever begin
        @(wait for reset);
        ovm_report_info(get_type_name(), "reset_happened ...", OVM_LOW);
        //Reset value of DAT_out
        intf.TB.cb.DAT_out <= 'b1;
        intf.clk_en = 'b0;
        end
endtask : reset_signals
```
Issues with OVM Reset Techniques
Contd..

• Issue 2
  - Irrespective of the on-the-fly reset, monitor will send data to scoreboard/coverage

Monitor
  //FOR Interface OVC MONITOR
  task run();
  fork
    monitor_transactions();
  //Reset Phase is Required to check
  reset_trascations(); //Reset
  join_none
  endtask // run
Issues with OVM Reset Techniques Contd..

- Issue 3
  - During on-the-fly reset, it is difficult control the sequencers of the sub-sequences from the virtual_sequence

![Diagram showing virtual sequence, normal layered sequence, and fork relationships]
Solutions for OVM Reset Technique

Issues

• Solution 1
  - Agent with qualified clock approach
  - Qualified data from monitor to Scoreboard/Coverage

• Solution 2
  - Stopping the sequencer from the sequence

• Solution 3
  - State Machine approach
Solutions for OVM Reset Technique

Issues Contd..

• Solution 1
  - Agent with qualified clock approach along with stopping the driver
  - Qualified data from monitor to Scoreboard/Coverage

**IF block**
```
assign qclk = (sig_reset==1) ? clk:0;
// Actual Signals
// USER: Add interface signals
clocking cb @(posedge qclk);
```

**DRIVER**
```
task run();
  fork
    //Variable or event to say reset is there
    reset_t = RESET_DE;
    get_and_drive();
    reset_signals();
  join_none
```

Muralidhara Ramalingaiah, Boobalan Anantharaman, Cypress Semiconductors
Solutions for OVM Reset Technique Issues Contd..

Drive Task

```verilog
task get_and_drive();
    //wait for the RESET event to complete to negedge of reset
@ (reset)
    reset_t = RESET_DE;  //disable event or variable
fork
    forever begin
        seq_item_port.get_next_item(req);
        ovm_report_info(get_type_name(), "sequencer got next itemM");
        drive_transfer(rsp);  //Works on qualified clock
        @(posedge intf.clk);  //normal clock
        seq_item_port.item_done(rsp);
    end
    begin
        //Block to suspend the driver and say end of transfer(solution)
        forever begin
            if(reset_t==RESET_EN)
                begin
                    ovm_report_info(get_type_name(), "FLY ON RESET");
                    this.end_tr(rsp);
                    reset_t = RESET_DE;
                    drive_transfer.kill(); //Never use this will make to hang
                    drive_transfer.suspend(); //Yes but still it will send data after reset
            @ (posedge intf.clk);
        end//forever
    end
    end//forever
endtask : get_and_drive
```
Solutions for OVM Reset Technique

Issues Contd..

Monitor

```verbatim
task run();
  fork
    monitor_transactions();
    reset_transactions(); // To reset all monitor variables
  join_none
endtask // run
```
Solutions for OVM Reset Technique

Issues Contd..

Monitor Task

```vsys
task monitor_transactions(); begin
    //wait for the RESET event to complete to negedge of reset
    @(Reset event); //From negative to positive
    forever begin
        trans_collected = new();
        @(posedge intf.cb); //ensure that wont send any data when there is no clock
        collect_transfer();
        data_trans();
        end
        if (checks_enable) // Check transaction
            perform_transfer_checks();
        if (coverage_enable) // Update coverage
            perform_transfer_coverage();
        // Publish to subscribers
        item_collected_port.write(trans_collected);
    end
endtask // monitor_transactions

task collect_transfer();
    void'(this.begin_tr(trans_collected));
    trans_collected.trans_kind = WRITE;
    @(posedge intf.cb); //qualified clock
    data.push_back(intf.TB.cb.DAT_out);
    forever begin
        @(posedge intf.cb); //qualified clock
    end
end task
```
Solutions for OVM Reset Technique Issues Cont'd..

• Solution 1
  
  - **Waveform:** No data transfer during the reset from agent, by ensuring that no clock in the driver block; but reset driving logic works fine
  
  - **Limitation:** During reset, driver won't drive anything. But if the driver has some wait like @posedge of some signals, then driver will run that signals again before suspending the driver task which leads to malfunction of driver/monitor and AP
Solutions for OVM Reset Technique
Issues Contd..

• Solution 2
  – Stopping the sequencer from the sequence
  – **Limitation:** Verification Engineer should know when to stop the sequencer from the test case

**Sequence/Test Case**
```
begin
  fork
    begin
      int_agent_seq.start(int_agent_sequencer);
    end
    begin
      wait(flyonreset);
    end
  fork
    begin
      reset_seq.start(reset_agent_sequencer);
      stop int_agent_sequencer;
    end
end
join
```
Solutions for OVM Reset Technique
Issues Contd..

- **Solution 3**
  - State Machine approach
  - **Benefit:** Makes life easy with additional logic (end transaction logic), very accurate comparing to all above methods. Highly advisable to use this in OVM
  - **Limitation:** Driver and Monitor code is bound on reset for every clock cycle. Complex Driver code

```plaintext
Driver
task run();
  fork
    get_and_drive();
  join_none

task get_and_drive();
  //wait for the RESET event to complete to negedge of reset
  @(reset)
  forever begin
    seq_item_port.get_next_item(req);
    ovm_report_info(get_type_name(), "sequencer got next itemM");
    drive_transfer(rsp); // Works on qualified clock
    @(posedge intf.cb); // normal clock
    seq_item_port.item_done(rsp);
  end
endtask : get_and_drive
```
Solutions for OVM Reset Technique
Issues Contd..

Driver
//STATE MACHINE
task drive_transfer()
begin
STATE = IDLE;
while(END==TRUE) begin

case(STATE)
   RESET: begin
      ovm_report_info(get_type_name(), "RESET STATE BYE BYE");
      //STopping TEST CASE
      END = FALSE;
      end
   IDLE : begin //default values
      If(if.reset) STATE= RESET;
      else begin //put next logic
         STATE = IRDA_DRIVE;
      end
   IRDA_DRIVE : begin
      for( number of data) begin
         If(if.reset) STATE= RESET;
         else begin //put next logic
            STATE = IRDA_DRIVE;
         end
      end //End of test
      END = FALSE;
   end
   endcase
@ (posedge intf.cb); //normal clock
end
tendtask
Solutions for OVM Reset Technique
Issues Contd..

Monitor

```plaintext
task run();
fork
  monitor_transactions();
  reset_trascations(); // To reset all monitor variables
  join_none
endtask // run
```
Solutions for OVM Reset Technique

Issues Contd..

Monitor Tasks

task monitor_transactions();
   //wait for the RESET event to complete to negedge
   of reset
   @(Reset event);//From negative to positive
   forever begin
      trans_collected = new();
      @(posedge intf.cb);
      collect_transfer();
      data_trans();
      end
   // Check transaction
   if (checks_enable)
      perform_transfer_checks();
   // Update coverage
   if (coverage_enable)
      perform_transfer_coverage();
   // Publish to subscribers
   item_collected_port.write(trans_collected);
endtask // monitor_transactions

task collect_transfer();
   void'(this.begin_tr(trans_collected));
   case(STATE) //State machine to collect data
      RESET: begin //default values
         if(if.reset)
            begin
               STATE= RESET;
               trans_collected.state=Reset;
               trans_collected.data=0;
            end
      else begin //put next logic
         STATE = IRDA_COLLECT;
         trans_collected.state=STATE;
         trans_collected.data=0;
      end
   endcase
   @(posedge intf.cb); //wait for one normal clk cycle
   //During reset, Transaction will have reset
   //information and default values
endtask
UVM Reset Techniques

- UVM Run-time Phases
  - The run-time schedule is the pre-defined phase schedule which runs concurrently to the uvm_run_phase global run phase
  - The run-time phases are executed in specific order
Pre Reset Phase

- Typical Uses
  - Wait for power good
  - Components connected to Virtual interfaces should initialize their o/p to X’s or Z’s
  - Initialize the clock signals to a valid value
  - Assign reset signals to X (power-on reset)
  - Wait for reset signal to be asserted if not driven by the Verification environment
Reset Phase

• Typical Uses
  - Assert reset signals
  - Components connected to Virtual interfaces should drive their o/p to their specified reset or idle value
  - Components and environments should initialize their state variables
  - Clock generators start generating active edges
  - De-assert the reset signal(s) just before exit
  - Wait for reset signal(s) to be de-asserted
Post Reset Phase

• Typical Uses
  - Components should start behavior appropriate for reset being inactive.
    • For example, components may start to transmit idle transactions
UVM Reset Techniques

- UVM Reset_phase
  - Diagram showing how reset_phase propagates from Test or ENV
UVM Reset Techniques Contd..

**Agent**

```verilog
task reset_phase(uvm_phase phase);
    phase.raise_objective(this, "Resetting agent");
    reset_and_suspend();
    phase.drop_objective(this);
endtask

virtual task reset_and_suspend();
    fork
        drv.reset_and_suspend();
        tx_mon.reset_and_suspend();
        rx_mon.reset_and_suspend();
    join
        sqr.stop_sequences(); // Stop sequences but can't
        // stop driver immediately
endtask
```

**Monitor**

```verilog
task reset_phase(uvm_phase phase);
    phase.raise_objective(this, "Resetting driver");
    reset_and_suspend();
    phase.drop_objective(this);
endtask

virtual task reset_and_suspend();
    // Clean all local variables
    end
endtask

virtual protected task run_phase(uvm_phase phase);
    forever begin
        // Call all methods to collect coverage
        end
```
UVM Reset Techniques Contd..

### Driver

```vhdl
-- This will be called from RUN_PHASE
task get_and_drive();
    `uvm_info("TEST", "Jumping back to reset phase", UVM_NONE);
    phase.jump(uvm_reset_phase::get());
endtask

virtual protected task run_phase(uvm_phase phase);
    forever begin
        get_and_drive();
    end
endtask
```

```vhdl
// This will be called from RUN_PHASE
task reset_and_suspend();
    phase.raise_objection(this, "Resetting driver");
    phase.drop_objection(this);
endtask
```

```vhdl
virtual task reset_and_suspend();
    // Drive default values on to the interface
endtask
```

```vhdl
task main_phase(uvm_phase phase);
    `uvm_info("TEST", "Jumping back to reset phase", UVM_NONE);
    phase.jump(uvm_reset_phase::get());
endtask
```

- Control Reset phase from test case or ENV or Reset Monitor (like below):

```vhdl
task reset_phase(uvm_phase phase);
    phase.raise_objection(this, "Resetting driver");
    reset_and_suspend();
    phase.drop_objection(this);
endtask
```

```vhdl
virtual task reset_and_suspend();
    // Wait for the RESET event to complete to negedge of reset
    reset_and_suspend();
    forever begin
        seq_item_port.get_next_item(req);
        drive_transfer(rsp); // Works on qualified clock
        @(posedge intf.cb); // Normal clock
        seq_item_port.item_done(rsp);
    end
endtask : get_and_drive
```

```vhdl
virtual protected task get_and_drive();
    // Wait for reset and suspend
    reset_and_suspend();
    forever begin
        seq_item_port.get_next_item(req);
        drive_transfer(rsp); // Works on qualified clock
        @(posedge intf.cb); // Normal clock
        seq_item_port.item_done(rsp);
    end
endtask
```

```vhdl
virtual protected task run_phase(uvm_phase phase);
    forever begin
        // Call all methods to start
        get_and_drive();
    end
endtask
```
Benefits and Limitation

- **Benefits in UVM**
  - All components can be controlled from Env/Test
  - Reset_phase of each component is in sync
- **Limitation in UVM**
  - Driver will still process the last received item
- **How to overcome above limitation?**
  - State machine approach again
OVM to UVM migration - Tips for reset

- Ensure that all the reset logic which is implemented in OVM should move to reset phase in UVM
- Ensure to move Reset triggering logic in all the places like monitors/drivers to Test or Env in UVM
- Control reset phases of each components from the Env or Test
Conclusion

• State Machine approach is a better one to follow in OVM Test benches comparing to other approaches
• UVM has the Reset_Phase with which reset logic of all the components can be controlled from Test or Env. To avoid the Driver limitation specified, State Machine approach should be followed in UVM Test benches
Questions?