ANALOG DEVICES

cadence

Reboot your Reset Methodology: Resetting Anytime with the UVM Reset Package

Courtney Fricano, ADI
Stephanie McInnis, Cadence
Uwe Simm, Cadence
Phu Huynh, Cadence

accellera SYSTEMS INITIATIVE

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

- Reset Verification Requirements
- uvm_thread Reset Package
- A recipe with 4 steps
- Summary & Conclusions

Reset Requirements

- A common verification requirement is to perform reset
  - At the start of the simulation
  - In the middle of the DUT normal operation
- Special care is required for verification environment during reset:
  - Activities and stimulus needs to stop and possibly restart once the reset is de-asserted
  - Assertions and checkers need to shut down gracefully
  - Data structures need to reset to proper initial values

Reset Challenges at ADI

- Reset is usually an afterthought that occurs during the middle of the DV effort
  - Can be very difficult to modify existing code to handle reset
- Often we end up having to compromise on reset verification to get something running
  - Turn off checks during reset
  - Create separate testbench to handle reset
  - Single very directed reset test
- This custom reset code usually has problems in higher system-level testbench environments

Reset and UVM Run-Time Phases

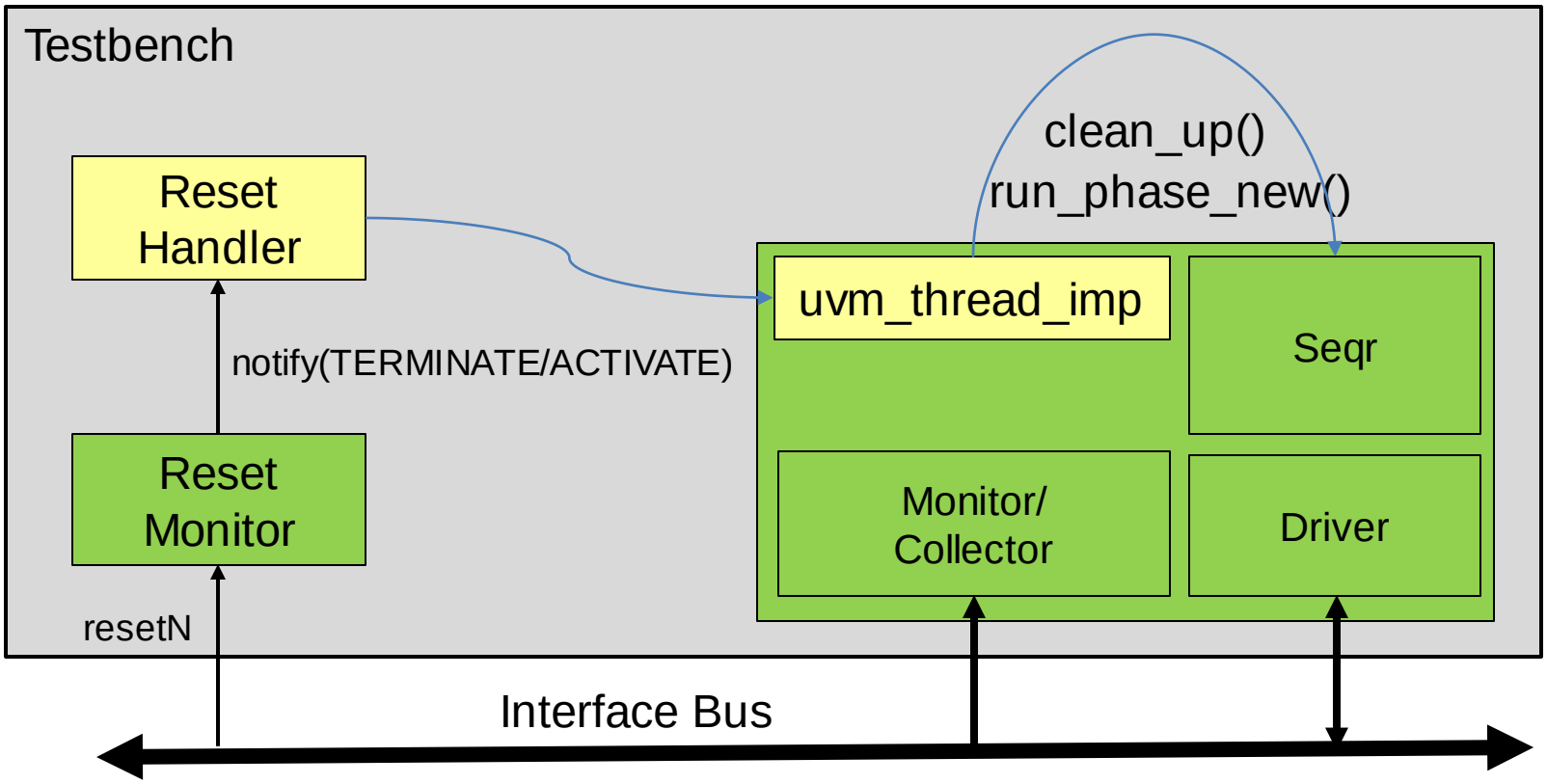
- UVM includes three phases “related” to reset
- The Accellera UVM phasing sub-committee has been trying to resolve how to handle resets using runtime phases
  - The solution still not user friendly
  - Still contention about use models
  - Phasing awareness model broken (sequences vs components)
- Our proposed methodology uses the run_phase()
  - Based on the standard UVM library
  - Works with existing UVCs already designed to handle reset

uvm_thread Package

- Available from http://forums.accellera.org/files/file/111-cadence-reset-example-and-package/

- Notification plus thread management instead of phases
  - Integrates easily with existing components that handle reset within the run_phase (drivers/monitors)
  - Can be extended to multiple reset domains
  - No changes to the UVM library are required
  - Does not interfere with UVM runtime phases

UVM Thread Reset Methodology



Reset Methodology Components

- Reset Monitor
  - Monitor reset signal(s)
  - Notify reset_handler of reset status
- Reset Handler
  - An instance of uvm_thread, provided by the package
  - Manages“reset-aware” components using their reset API:
    - Reset goes away → Invokes run_phase_new()
    - Reset active → Terminates threads, Invokes clean_up()
- Reset-aware components:
  - Register themselves with the reset_handler
  - Implement new reset APIs: clean_up() and run_phase_new()

A recipe with 4 steps

1. Instantiate a (uvm_thread) reset_handler
2. Implement Reset Monitoring to notify upon Reset Changes
3. Signup for notification
4. Handle notifications

Step 1 – Instantiate a (uvm_thread) reset_handler

```
class wd_osc_env extends uvm_env;
 ...
  reset_monitor reset_mon;
  uvm_thread reset_handler;

 function void build_phase(uvm_phase phase);
   ...
   reset_mon = reset_monitor::type_id::create("reset_mon", this);
   reset_handler = uvm_thread::type_id::create("reset_handler", this);
   uvm_config_db#(uvm_thread)::set(this, "*", "reset_handler", reset_handler);

 endfunction
...
endclass
```


© Accellera Systems Initiative

Step 2: Implement Reset Monitoring to notify upon Reset Changes

```
class reset_monitor extends uvm_monitor;
  virtual reset_if vif;
  uvm_thread  reset_handler;

 function void connect_phase(uvm_phase  phase);
    uvm_config_db#(virtual  reset_if)::get(this, "", "vif", vif);
    uvm_config_db#(uvm_thread)::get(this,  "", "reset_handler",  reset_handler);
 endfunction

virtual task run_phase(uvm_phase  phase);
forever begin
        @(vif.resetN);
        if (vif.resetN)       reset_handler.notify(ACTIVATE);
        else                  reset_handler.notify(TERMINATE);
    end
  endtask
endclass
```

Step 3: Signup for Notification

In each reset-sensitive object

- Add a (uvm_thread_imp) reset_export for notification
- Register reset_export with the reset_handler

```
class wd_osc_drv extends uvm_driver #(wd_osc_data);
  ...
 function void connect_phase(uvm_phase phase);
     uvm_thread reset_handler;
     uvm_thread_imp#(wd_osc_drv) reset_export;
     reset_export = new(„reset_exp“,this);
     uvm_config_db#(uvm_thread)::get(this, "", "reset_handler", reset_handler)
     reset_handler.register(reset_export, uvm_thread_pkg::default_map);
  endfunction
endclass
```

TLM-like port reset_export for notification

Step 4a: handle notifications

(Optionally) implement clean_up() method to put component back into “reset” state; for example:

- Scoreboard: empty internal FIFOs
- Driver: drive all pertinent signals to idle state
- Sequencer: terminate all running sequences; clear internal state

```
class some_driver extends uvm_driver;
  virtual function void clean_up();
    vif.OSC1 <= $urandom_range(1,0);
  endfunction
...
endclass
```

```
class some_seqr extends uvm_sequencer#(...);
  virtual function void clean_up();
    stop_sequences();
  endfunction
...
endclass
```

Step 4a: handle notifications

- Implement run_phase_new() for
  - activities to be invoked when reset goes away
  - Sequencer can re-invoke the sequence(s)
- Same signature as run_phase()

```
class some_driver extends uvm_driver;
 virtual task run_phase_new(uvm_phase phase);
   get_and_drive();
 endtask
  ...
endclass
```

Summary & Conclusions

- uvm_thread Reset package provides a good methodology for adding reset verification to UVM testbenches

- Flexible and extendible
  - Minimal modifications to existing UVCs
  - Used as standard reset methodology across projects
  - Can extend to multiple resets and adjust to different reset configurations
  - clean_up() and run_phase_new() can be implemented based on specific UVC requirements

- Works with standard UVM library source code

Questions ?