Rerun and Restart Methodology

Versions 0.6

Introduction

One of the main requirements of a UVC is the ability to rerun upon request. The need to rerun a component can be an assertion of a reset pin, a decision to abruptly disconnect a certain device and reconnect later on, etc. This request for rerunning a component or objects in the middle of simulation should be part of the standard API of any UVM class instance.

Note that the issues of rerun and restart can also be resolved in phasing or via states. Since the phases sub-committee decided not to use phases for non-stimuli aspects and states not agreed upon we suggest this simple mechanism. The solution is flexible and can co-exist with phases and state moving forward (see more details bellow on propagating rerun request).

Detailed Requirements:

1. Ability to rerun a class instance in the middle of simulation using a standard rerun() api call. This means that if you get a package from an unknown source you already know how to rerun it.
2. Enablement to automatically stop all threads and restart run. The threads could be active or passive (e.g. checkers)
3. Provide a user hook to clean data structures upon rerun request. For example the user may decide to keep memory status as is or initialize it. He can clean a scoreboard expected queue etc'
4. User may choose to have some logic that will be executed in the first run but will not be executed in subsequent reruns.
5. When restart is triggered via reset the mechanism should allow to kill the threads and checkers when reset starts and revive the threads when reset is ending
6. Some functionality should sustain reset (maybe some checkers are checking reset time activity).
7. The capabilities should work for components (typical) and objects (rarely)
8. User may want to reset an instance without any impact on other components. Any coupling of instance rerun should be done explicitly.
9. User may want to revive run only once()

Suggested solution

The suggested solution provides a scheme to rerun a user defined task including thread management for this thread (including all it sub-threads) and reviving it as many times as required.

Suggested API

do_rerun()

Is a callback that is executed once the current thread has been terminated. It allows cleaning of data structures and can be used to rerun other components. The default implementation provides a component based propagation of rerun to sub components. do_rerun() is invoked by a call to rerun() and should not be invoked directly.

Syntax: virtual function void uvm_component:: do_rerun()

rerun()

Is a template method managing a native thread for the user. Managing means starting, stopping and restarting the thread. rerun() is invoking the user callback do_rerun() when appropriate. The function itself has to be invoked initially by the user. The rerun() call can propagate automatically via do_rerun() or the user can invoke subcomponent.rerun() as part of a component.do_rerun()

Syntax: function uvm_component::rerun().

do_run()

The do_run() is invoked from the run_phase of a component and is executed an a managed thread context. This thread context is stopped at the begin of rerun(), all (sub)threads are terminated and afterwards restarted via a new do_run() invocation.

Syntax: virtual task uvm_component::do_run()

Examples

Example(1) rerun enabled ubus driver

```
class ubus_master_driver extends uvm_driver #(ubus_transfer);
 //get_and_drive() started in the managed thread
// will be reinvoked after rerun()
  virtual task do_run();
      get_and_drive();
  endtask

  // callback executed once the managed thread has been terminated
  virtual function void do_rerun();
    super.do_rerun(); // propagating rerun to subcomponents via default do_rerun() impl.
    reset_signals(); //perform cleanup
  endfunction

  // get_and_drive (the driver dut functionality)
  virtual protected task get_and_drive();
  endtask
endclass
```

Example(2) causing rerun of a component via an explicit call to rerun()

```
class ubus_env extends uvm_component;
   virtual task run();
      super.run();
      forever begin
         @(posedge vif.sig_reset);
         rerun();
      end
   endtask

endclass : ubus_env
```

Example(3) manual propagation of rerun() to subcomponents

```
class ubus_slave_agent extends uvm_agent;
  // rerun manual propagation
  function void do_rerun();
    monitor.rerun();
    if(get_is_active() == UVM_ACTIVE) begin
       driver.rerun();
       sequencer.rerun();
    end
  endfunction
endclass : ubus_slave_agent
```

Known issues