2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION TAIWAN

Improving UVM test benches using UVM Run time phases

Karthik Palepu
Lingkai Shi
Prosper Chen

AMD

accellera SYSTEMS INITIATIVE

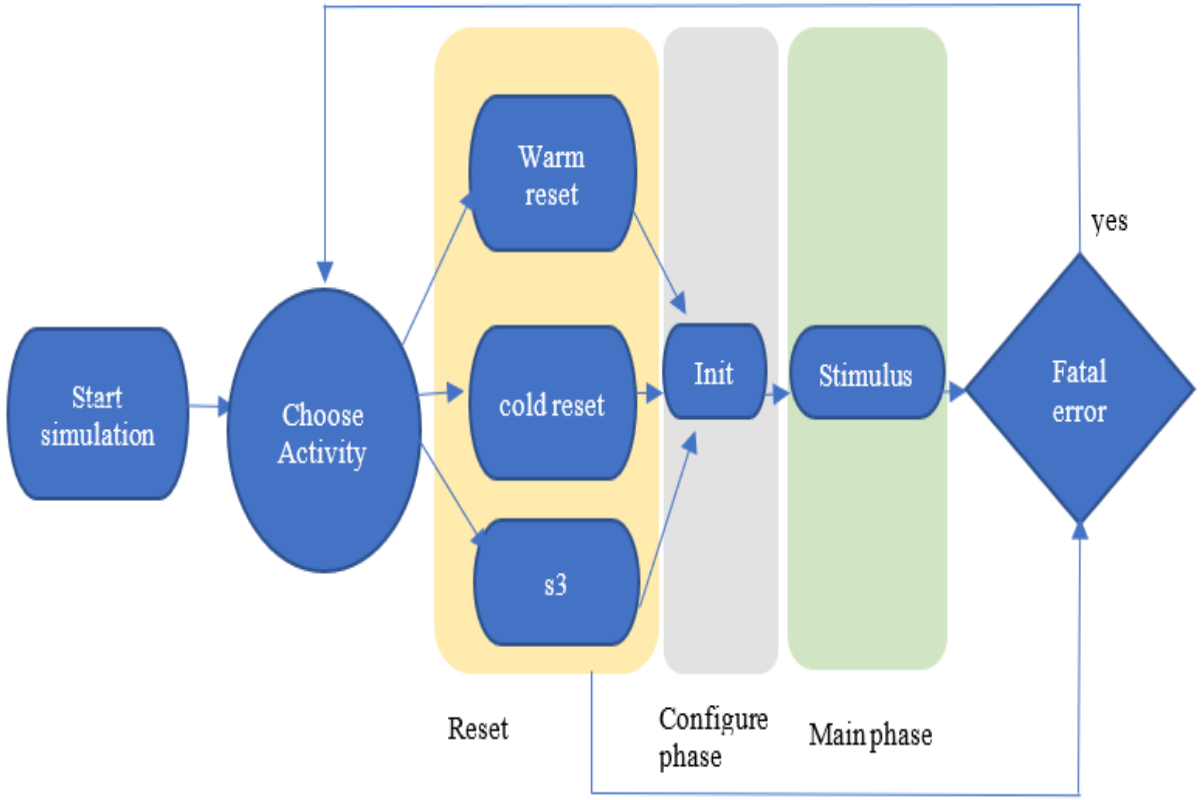
Agenda

- Introduction
  - Common Issues
  - Solution
  - Run time Phase Header Macro
  - Coordination of all reset sources
  - Phase Jump API
- Conclusions
- References
- Appendix

Introduction

- Common Issues Across Testbenches
  - How to end test cleanly?
  - How to handle a fatal case?
  - Where to disable all stimulus?
  - When to enable the stimulus in a particular scenario?
- Reusable Methodology Using UVM Run Time Phases
  - Addresses synchronization issues between various components

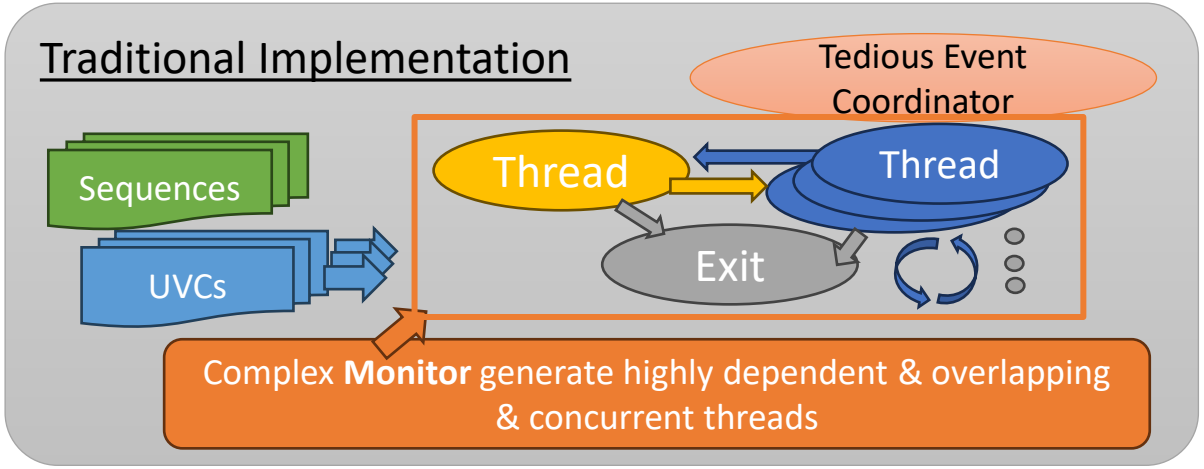
Issue 1: Thread Synchronization

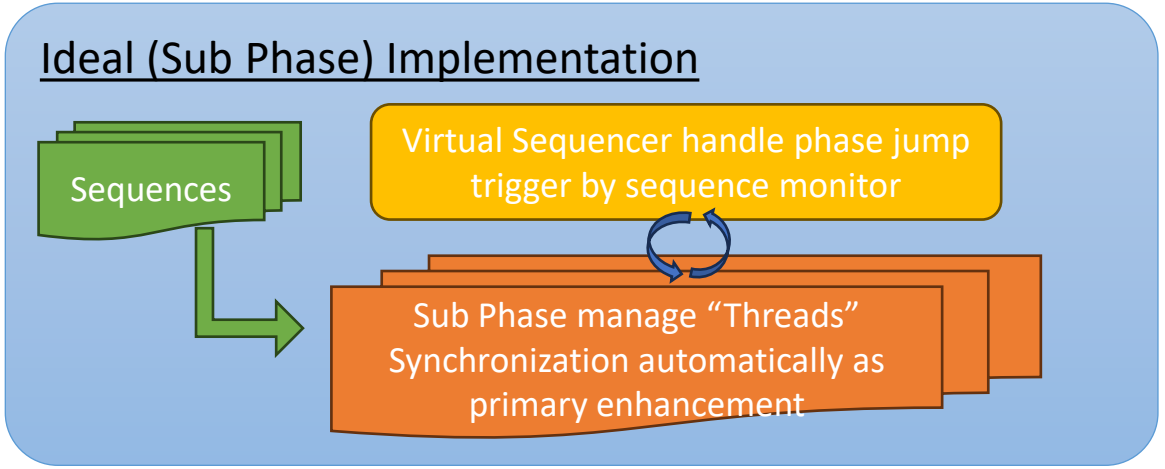


- Multiple reset threads triggered by fatal event
  - Can mess with execution of subsequent events
  - The regression is barely achievable.
- Complex activities in reset phase
  - Warm reset, cold reset, s3 (Ultra Low Power State, Deep sleep, Stutter Mode… etc)
  - Must catch fatal error during execution
  - Must trigger reset event repeatedly
- Potential issue with threads
  - Multiple reset threads running
  - Messing up order of execution

Issue 2: End of Simulation

- End of simulation in UVM testbenches requires disabling all stimulus
  - Communication among multiple components introduces race conditions
- Dependencies between components can result in race conditions
  - Issue exacerbates in System-on-Chip (SOC) with numerous interdependent components
- Scalable UVM architecture needed for comprehensive solution
  - Architectural changes ensure visibility of state and termination process
  - Effective management of dependencies and prevention of race conditions



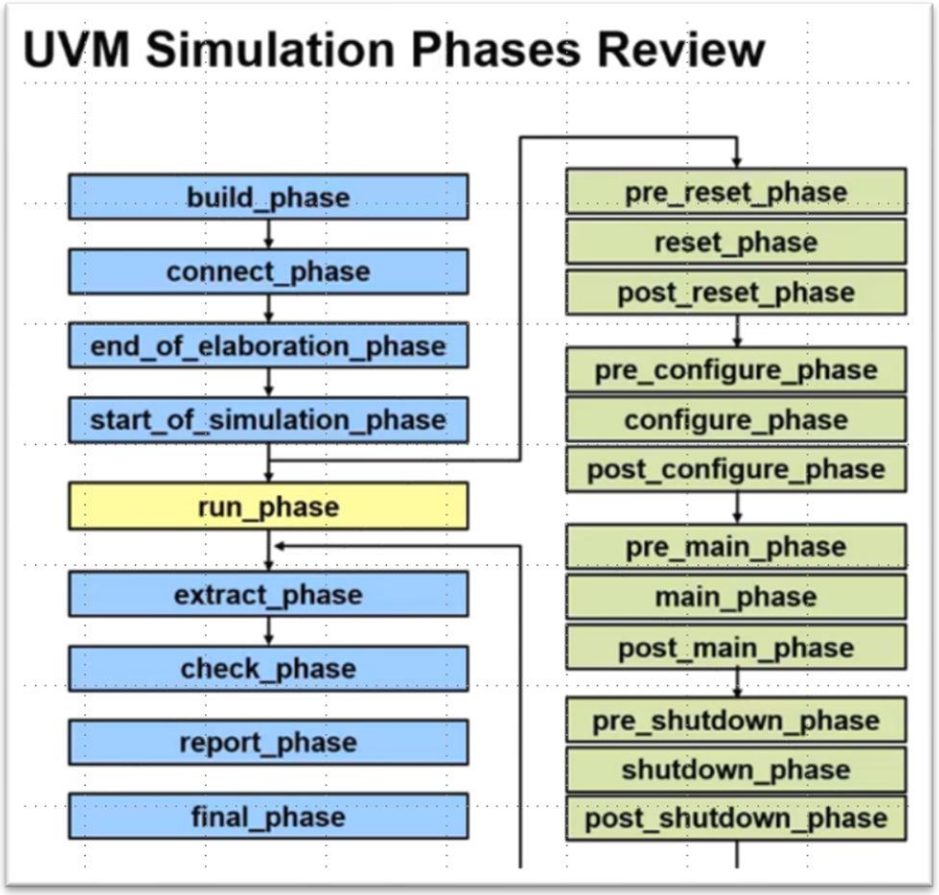


Issue 3: Reusability

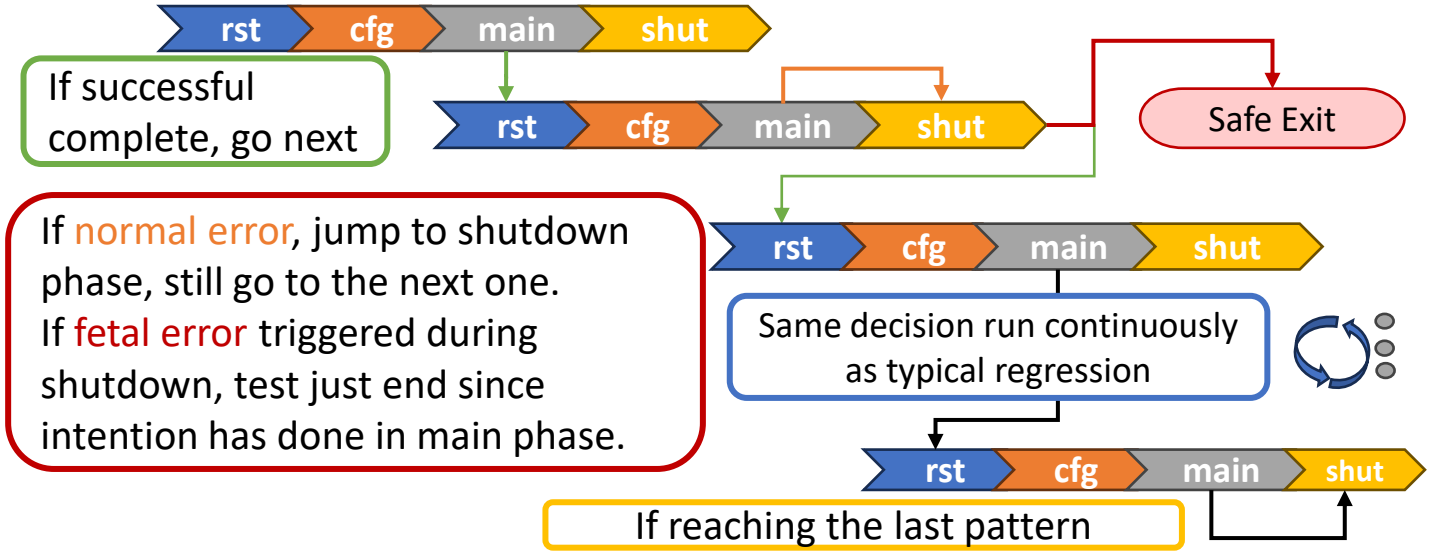
- Run Phase Encapsulates Majority of Logic
  - Modifying or altering presents considerable challenge
- Dealing with Legacy Code
  - Large and intricate, adds extra layer of difficulty
- Transitioning to New Architectures
  - Minimize alterations to maintain robustness
  - Prevent introduction of new issues
- Transparent Transition for Customer
  - Seamless integration of delivered IP for different SOC
  - No unexpected changes or complications
- Target:
  - Merge all codes from different component of TB into one unified place.
    - The test end event spread among sequence, test_base, monitor before.

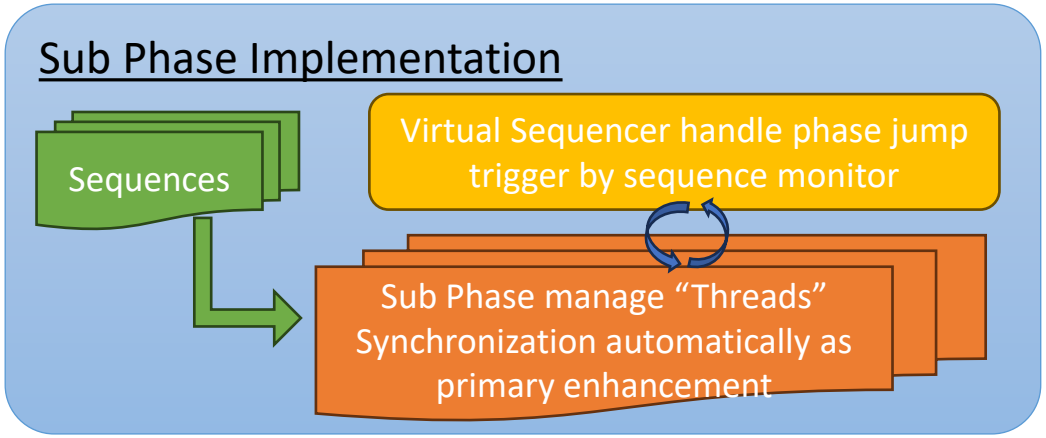
Solution

- UVM supports multiple subphases within the run phase
  - Granular control helps manage interdependencies between components, resolve **issue2, end of simulation.**
- Automatically terminates threads initiated by a subphase while transitioning to another phase
  - Avoids race conditions of **issue1**, **thread synchronization**.
- UVM subphases facilitate the modulation of reset behavior
  - Comprehensive rewrite of reset resolve **issue3, reusability**
- Novel methodology introduced
  - Manipulate subphases of run_phase through API-centric approach to surgically control the very phase jump behavior of reset.
  - Allows for reuse of majority of existing code by tweaking the reset function apart from main logic in main_phase.
  - Streamlines integration of UVM subphases into existing TB by modifying only the TEST without UVM Component rework.

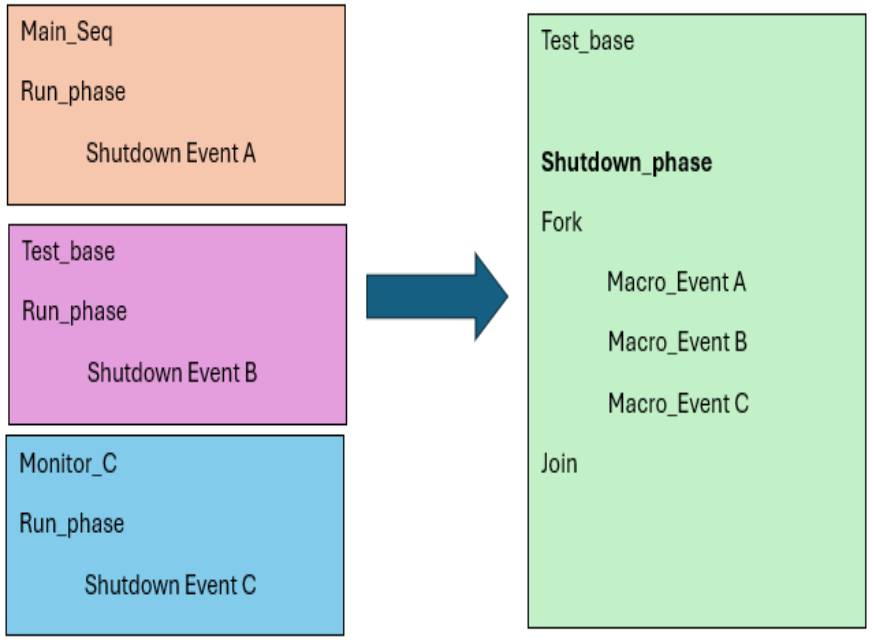


Sub Phase Implementation





- All activities corresponding to one phase are implemented under one sub phase instead of multiple places across the TB.
- All similar activities can implement Macro based approach which increases reusability and reduces clutter.
- Scoreboard is presented by checker/assertion after sequence completion



Implementation1: Runtime Phase Header Macro

```
`define subphase_header(SUBPHASE_NAME, TASK_NAME, EVENT_NAME, PARA1, PARA2) \
begin \
  uvm_event_pool::get_global_pool().get("``EVENT_NAME").wait_on(); \
  phase.raise_objection(this, "test_base ``SUBPHASE_NAME" raised an objection", 1); \
  fork \
    begin \
      fork \
        ``TASK_NAME``(``PARA1``, ``PARA2``); \ //Execute the activity
        @``SUBPHASE_NAME_end; \ //Kill the activity by the Hook
      join_any
      disable fork;
    end
  join
  phase.drop_objection(this, "uvm_test_base dropped object on reset_phase", 1); \
  uvm_event_pool::get_global_pool("``EVENT_NAME".reset(); \
end
```

- Reduces code repetition and enhances robustness
  - Waits for event trigger and forks associated task
  - Incorporates global event to suspend ongoing tasks
  - Accommodates multiple parallel phase jump calls (Next 2 pages)
  - Offers flexibility for global TB control
  - Prevents corner cases from improper user interactions
- Takes in 5 inputs
  - SUBPHASE_NAME, TASK_NAME, EVENT_NAME, and two PARAMETERS
- Standardization ensures predictable, smooth, and controlled task execution
  - Next case managed by different event

Implementation2: Coordination of All Reset Sources

```
task reset_phase(uvm_phase phase);
  event reset_phase_end;
  phase.raise_objection(this, "test_base, reset_phase rasied on objection", 1);
  fork
    begin // Thread 1: cold_reset_event
      `subphase_header(reset_phase, do_cold_reset_all, exec_cold_reset_event,);
    end
    begin // Thread 2: warm_reset_event
      `subphase_header(reset_phase, do_warm_reset_all, exec_warm_reset_event,);
    end
    begin // Thread 3: s3_event
      `subphase_header(reset_phase, do_s3_all, exec_s3_event,);
    end
    begin // Thread 4: Check fatal error happen during reset_phase
      uvm_event_pool::get_global_pool().get("disable_main_thread").wait_on();
      // wait for global kill
      ->reset_phase_end; //Trigger reset_phase_end. This will kill Thread 1, 2, 3.
    end
  join_any
  phase.drop_objection(this, "test_base, reset_phase dropped object on reset_phase", 1);
  `uvm_info(get_report_id("test_base_reset_phase"), $sformatf("Finish reset_phase"),);
endtask: reset_phase
```

- Reset phase implemented as a sub phase in run using sub_phase user API and sub_phase header
  - Three threads execute tasks using subphase_header macro
  - Fourth thread waits on global event to kill reset phase
  - Reset_phase_end event kills threads 1, 2, 3
  - phase_jump API triggers corresponding activity and enters reset phase in run phase (next page)
- Revisions implemented at test level
  - API invoked in main sequence of TB
  - Subphases not advised at Agent/Env level
  - May result in incompatibility with upstream SOC infrastructure

Implementation3: Phase Jump API

```
task phase_jump(string phase_name, uvm_event event_trigger_list[$]);
  uvm_phase m_target_phase;
  uvm_event trigger_event_list[$];
  // (Info) Start jump to -> phase_name
  if ( (phase_name != "reset") & (phase_name != "configure")
    & (phase_name != "shutdown"))
    // (Error) Can't recognize the subphase
  while (current_phase.get_name=="start_of_simulation"
      | current_phase.get_name=="run"
      | current_phase.get_name=="pre_reset" ) begin
      // (Info) Waiting to main_phase
      #1ps;
  end
  m_target_phase=current_phase.find_by_name(phase_name);
  case (phase_name)
    "reset" : begin
          foreach (event_trigger_list[i]) begin
             trigger_event_list.push_back(event_trigger_list[i]);
          end
     end
    "configure" : begin
          foreach (event_trigger_list[i]) begin
             trigger_event_list.push_back(event_trigger_list[i]);
          end
     end
    "shutdown" :  begin
          foreach (event_trigger_list[i]) begin
              trigger_event_list.push_back(event_trigger_list[i]);
          end
     end
  endcase
  if (m_target_phase==null)      // (Error) fail to find the target phase
  else                           // (Info) start did find the target phase
  if (m_target_phase.is(current_phase)) begin // trigger event & let it  happens
    foreach(trigger_event_list[i]) begin
      trigger_event_list[i].trigger;
    end
  end
  else if (m_target_phase.is_before(current_phase)) begin
    // need to jump, just allow jump from main phase to others as for now
    while (current_phase.is(current_phase.find_by_name("main"))==0)
      @(current_phase);
    foreach(trigger_event_list[i])
      trigger_event_list[i].trigger;
    current_phase.jump(m_target_phase);
  end
  else if (m_target_phase.is_after(current_phase)) begin
    // only when target phase is shutdown phase, need to jump
    if ((current_phase.is(current_phase.find_by_name("reset"))==1)
      && (phase_name == "shutdown")) begin
       while (current_phase.is(current_phase.find_by_name("main"))==0)
         @(current_phase);
    end
    foreach(trigger_event_list[i])
        trigger_event_list[i].trigger;
    if (phase_name == "shutdown")
        current_phase.jump(m_target_phase);
  end
  m_target_phase.wait_for_state(UVM_PHASE_READY_TO_END); // Hook
  trigger_event_list.delete();
endtask : phase_jump
```

Conclusions

- UVM run-time phases offer efficient solution to synchronization challenges
  - Enhances control over simulation flows
  - Resolve potential race conditions
  - Requires minimal modifications to existing TB
- Contributions
  - Methodology applied to TB used by over 60 individuals
  - Achieved by single resource within 3 months
  - Did not disrupt existing project timeline
- Strategies can be adapted to various TB structures
  - Ensures robust and adaptable TB architecture
- Provides efficient, flexible, and resource-effective solution
  - Improves UVM TB synchronization and manageability

Q&A

- Thanks for your participation!

References

- Authors: Brian Hunter, Ben Chen and Rebecca Lipon
  - Published in the Proceedings of SNUG SV