De-mystifying synchronization between various verification components by employing novel UVM classes

Synchronizing Verification Components

UVM class-based test benches have become as complex as the hardware under test (DUV), and are evolving into large object oriented software designs.

This poster is an attempt to lift the veil on some underutilized UVM classes as following:

1) `uvm_event`/`uvm_event_pool`
2) `uvm_barrier`/`uvm_barrier_pool`
3) `uvm_heartbeat`
4) Grab/Ungrab

**UVM EVENT / UVM EVENT_POOL**

The UVM event class congregates the richness of System Verilog/Verilog events and adds a few features of its own, thus achieving leverage.

**Advantages of UVM events:**

1) Provides a knob to the user just before or after the activation of `uvm_event`/`uvm_event_pool`
2) Can be used to return status of `uvm_objects` on triggering of event
3) Ease of accessibility of `Event objects` can be passed around the Test bench using `uvm_config_db`

For example in verification environment two components such as “environment” and “agent” or [driver & monitor etc] may depend on single event say “reset_event” and hence can easily be shared.

**UVM BARRIER**

**UVM_BARRIER / UVM_BARRIER_POOL:**

Its assets come to the fore when a user wishes to block desired number of processes until a threshold/synchronization point is achieved.

The `uvm_barrier_pool` classes make it easier to manage components that share the same barriers as also can be accessed globally just as `uvm_events`

Employing Barriers to Check for Link UP

Scenario: Devices D1 & D2 are waiting for Link Up and once link is up all the threads are unblocked.

**UVM HEARTBEAT**

The UVM Heartbeat behaves as a watchdog timer and is quite powerful. It watches for activity in the test bench components and if it finds that there isn’t the right amount of activity in that window - will issue a fatal message and end the simulation. This can catch a simulation lock-up early on – even before the global timeout kicks in, potentially saving a significant amount of time.

Here are few of the scenarios of interest in ceasing a simulation rather than `UVM_TIMEOUT` being called out:

1) Absent Connections between the Verification Components
2) Simulation Hang in State Machines
3) Simulation Hang in System Verilog

To employ the UVM heart beat we need to associate a specific object related information, and the heartbeat object must raise (or drop) the synchronizing objection during the heartbeat window

**UVM_HEARTBEAT**

The most prevalent API’s for `uvm_heartbeat` class are:

1) `set_mode`: Sets or retrieves the heartbeat mode.
   - The heartbeat can be configured so that all components (UVM_ALL_ACTIVE), exactly one (UVM.ONE_ACTIVE), or any component (UVM_ANY_ACTIVE) must trigger the objection in order to satisfy the heartbeat condition.
2) `Add/remove`: Add/Removes a single component to the set of components to be monitored
3) `set_threshold`: Sets the target list of components that are required to be monitored and setting of the heartbeat event. Soon after invocation of this monitoring is initiated

**GRAB / UNGRAB**

This mechanism provides the sequence with exclusive access to the driver and will allow a sequence to complete its operation without any other sequence operations in between them. The grab() method requests a lock on the specified sequencer. A grab() request is put in front of the arbitration queue. It will be arbitrated before any other requests.

This mechanism is highly recommended in scenarios when a virtual sequencer requires full control over its subchild sequencers for a limited time and then lets the original sequencer continue working. From an implementation perspective it will be highly useful in generating INTERRUPT Sequences where disabling of subchild sequencers is required and highest priority needs to be given to specific sequences

**Pseudo Code:**

```plaintext
virtual task body();
    // Grab the cpu sequencer if not virtual.
    if (sequencer.cpu_seqr != null)
        p_sequencer.cpu_seqr.grab(this);
    // Execute a sequence.
    'uvm_do_on(interrupt_seq,sequencer.cpu_seqr)
    // Ungrab.
    if (sequencer.cpu_seqr != null)
        sequencer.cpu_seqr.ungrab(this);
```