End Of Test Mechanisms

End-of-Test and Objection Mechanisms

Topic Overview

End of Test in the UVM

A UVM testbench, if is using the standard phasing, has a number of zero time phases to build and connect the testbench, then a number of time consuming phases, and finally a number of zero time cleanup phases.

End of test occurs when all of the time consuming phases have ended. Each phase ends when there are no longer any pending objections to that phase. So end-of-test in the UVM is controlled by managing phase objections. The best way of using phase objections is described in articles linked to from the Phasing Introduction Page.

A simple test might look like this:

```
task reset_phase( uvm_phase phase);
  phase.raise_objection( this );
  reset_seq.start( m_sequencer );
  phase.drop_objection( this );
endtask

task configure_phase( uvm_phase phase);
  phase.raise_objection( this );
  program_control_registers_seq.start( m_sequencer );
  phase.drop_objection( this );
endtask

task main_phase( uvm_phase phase);
  phase.raise_objection( this );
  data_transfer_seq.start( m_sequencer );
  phase.drop_objection( this );
endtask

task shutdown_phase( uvm_phase phase);
  phase.raise_objection( this );
  read_status_registers_seq.start( m_sequencer );
  phase.drop_objection( this );
endtask
```

Each of the four phases in the test above raises and drops an objection. Since the particular phases above occur in sequence, then one phase cannot start before the previous one has finished. Raising an objection at the beginning of each phase prevents the phase from immediately terminating, and dropping it means that this component no longer has an objection to the phase ending. The phase will then terminate if there are no other pending objections that have been raised by other components or objects. When there are no pending objections to a particular phase, the simulation will move on the next phase. When there are no time consuming phases left to execute, the simulation

moves on to the cleanup phases and the test ends.

phase_ready_to_end

For sequences, tests, and many complete testbenches, the raising and dropping of phase objections during the normal lifetime of the phase, as described above, is quite sufficient.

However, sometimes a component does not want to actively raise and drop objections during the normal lifetime of a phase, but does want to delay the transition from one phase to the next. This is very often the case in transactors, which for performance reasons cannot raise and drop an objection for every transaction, and is quite often the case for end-to-end scoreboards.

To delay the end of phase after all other components have agreed that the phase should end, that component should raise objections in the phase_ready_to_end method. It is then responsible for dropping those objections, either in the main body of the component or in a task fork / join none'd from the phase_ready_end_method.

An example of using fork / join_none is shown below :

```
function void my_component::phase_ready_to_end( uvm_phase phase );
  if( !is_ok_to_end() ) begin
    phase.raise_objection( this , "not yet ready to end phase" );
    fork begin
      wait_for_ok_end();
      phase.drop_objection( this , "ok to end phase" );
    end
    join_none
  end
endfunction : phase_ready_to_end
```

phase_ready_to_end() **without** fork / join_none is used in the Object-to-All and Object-to-One phasing policies often used in components such as transactors and scoreboards.

Objections

Objections

The uvm_objection class provides a means for sharing a counter between participating components and sequences. Each participant may "raises" and "drops" objections asynchronously, which increases or decreases the counter value. When the counter reaches zero (from a non-zero value), an "all dropped" condition occurs. The meaning of an all-dropped event depends on the intended application for a particular objection object. For example, the UVM phasing mechanism uses a uvm_objection object to coordinate the end of each run-time phase. User-processes started in a phase raise and drop objections to ending the phase. When the phase's objection count goes to zero, it indicates to the phasing mechanism that every participant agrees the phase should be ended.

> The details on objection handling are fairly complex, and they incur high overhead. Generally, it is recommended to only use the built-in objection objects that govern UVM end-of-phase. It is not recommended to create and use your own objections.

Note: Objection count propagation is limited to components and sequences. Other object types may participate, but they must use a component or sequence object as context.

Interfaces

The uvm_objection class has three interfaces or APIs.

Objection Control

Methods for raising and dropping objections and for setting the drain time.

- raise_objection (uvm_object obj = null, string description = "" , int count = 1).

  Raises the number of objections for the source object by count, which defaults to 1. The raise of the objection is propagated up the hierarchy, unless set_propagate_mode(0) is used, in which case the propagation is directly to uvm_test_top.

- drop_objection (uvm_object obj = null, string description = "" , int count = 1).

  Drops the number of objections for source object by count, which defaults to 1. The drop of the objection is propagated up the hierarchy. If the objection count drops to 0 at any component, an optional drain_time and that component's all_dropped() callback is executed first. If the objection count is still 0 after this, propagation proceeds to the next level up the hierarchy.

- set_drain_time (uvm_object obj = null, time drain).

  Sets the drain time on the given object.

Recommendations:

- Use phase.raise_objection / phase.drop_objection inside a test's phase methods to have the test control end-of-phase - usually when the execution of a sequence (or set of sequences) has completed.
- Always provide a description - it helps with debug
- Usually use the default count value.
- Limit use of drain_time to uvm_top or the top-level test, if used at all.

Objection Status

Methods for obtaining status information regarding an objection.

- get_objection_count (uvm_object obj)

  Returns objection count explicitly raised by the given object.

- get_objection_total (uvm_object obj = null)

  Returns objection count for object and all children.

- get_drain_time (uvm_object obj)

  Returns drain time for object (default: 0ns).

- display_objections (uvm_object obj = null, bit show_header = 1)

  Displays objection information about object.

Recommendations:

- Generally only useful for debug
- Add +UVM_OBJECTION_TRACE to the vsim command line to turn on detailed run-time objection tracing. This enables debug without having to modify code and recompile.
- Also add +UVM_PHASE_TRACE to augment objection tracing when debugging phase transition issues.

Callback Hooks

The following callbacks are defined for all uvm_component-based objects.

- raised()

  Called upon each raise_objection by this component or any of its children.

- dropped()

  Called upon each raise_objection by this component or any of its children.

- all_dropped()

  Called when drop_objection has reached object and the total count for object goes to zero

Recommendations:

- Do not use callback hooks. They serve no useful purpose, are called repeatedly throughout the simulation degrading simulation performance.

Objection Mechanics

Objection counts are propagated up the component hierarchy and upon every explicit raise and drop by any component. Two counter values are maintained for each component: a count of its own explicitly raised objections and a count for all objections raised by it and all its children, if any. Thus, a raise by component mytest governing the main_phase results in an objection count of 1 for mytest, and a total (implicit) objection count of 1 for mytest and 1 for uvm_top, the implicit top-level for all UVM components. If mytest.myenv.myagent.mysequencer were to than raise an objection, that results in an objection count of 1 for mysequencer, and a total (implicit) objection count of 1 for mysequencer, 1 for myagent, 1 for myenv, 2 for mytest, and 2 for uvm_top. Dropping objections propagates in the same fashion, except that when the implicit objection count at any level of the component hierarchy reaches 0, propagation up the hierarchy does not proceed until after a user-defined drain_time (default: 0) has elapsed and the all_dropped() callback for that component has executed. If during this time an objection is re-raised at or below this level of hierachy, the all-dropped condition is negated and further hierarchical propagation of the all_dropped condition aborted.

Raising an objection causes the following:

1. The component or sequence's source (explicit) objection count is increased by the count argument
2. The component or sequence's total (implicit) objection count is increased by the count argument
3. If a component, its raised() callback is called.
4. If parent is non-null, repeat steps 2-4 for parent.

A sequence's parent is the sequencer component that it currently is running on. Propagation does not occur up the sequence hierarchy.

Virtual sequences (whose m_sequencer handle is null) do not propagate.

Dropping an objection causes the following:

1. The component or sequence's source (explicit) objection count is decreased by the count argument
2. The component or sequence's total (implicit) objection count is decreased by the count argument
3. If a component, its dropped() callback is called.
4. If the total objection count for the object is not zero and parent is non-null, repeat steps 2-4 for parent.
5. If the total objection count for the object is zero, the following is forked (drop_objection is non-blocking)
   - Wait for drain time to elapse
   - Call all_dropped() virtual task callback and wait for completion
   - Adjust count argument by any raises or drops that have occurred since. If drop count still non-zero, go to 4