How to finish the UVM Test?:

Lets analyze below the approaches to finish the Test in UVM.

Raising & Dropping Objections:

Broadly UVM Testbench is having **3 Phase groups**. Two of these Phase groups i.e. "**Build & Connect**" and "**Cleanup**" are executed in zero time means these 2 groups do not consume time. One Phase group which is called "**Run**" Phase is the one which consumes times. End of Test occurs when all the time consuming phases are ended. Each Phase ends when there is no pending objections for that particular Phase. Hence it is essential to comprehend the objection mechanism in UVM? Lets try to understand it in one of the simplest possible form:

The uvm_objection class provides a means for sharing a counter between the participating Components and Sequences. It means in UVM Testbench Hierarchy, the Components and Sequences which uses the Objection mechanism share a counter between them. Each participating member can raise or drop the objections which in turn increments or decrements the counter value. **raise_objection()** and **drop_objection()** are the methods to be used to do that. Once the value of the shared counter reaches to zero from a non-zero value – we can say that "**all dropped**" condition is achieved. Now what action is supposed to take on "all dropped" condition will depend on the application.

One good example of Objection mechanism usage is **UVM Phasing** which uses the **uvm_objection object** to co-ordinate the end of each run-time phase. Since all the Components & Sequences of a UVM Testbench Hierarchy are provided equal opportunities to participate in the Standard Phasing mechanism by executing user-defined process. So if a Component or Sequence starts a user-process for a selected phase, an objection is raised and when this user-process is finished, an objection is dropped. Similarly when all the raised objections for a particular phase are dropped & that phase gets into "all dropped" condition, action taken is to move to the next Phase.

After following this approach simulation reaches to the end of UVM Phasing i.e. Run-time phase, at this moment it indicates that all the Components and Sequences are in agreement to finish the Test and nothing is left pending on anyone's end. This is the stage where an UVM Test moves to non-time consuming Cleanup Phase and finally Test ends.

Lets understand it through some an example UVM Test containing different run-time sub-phase code:

```
//// UVM Test code
class my_test extends uvm_test;
 `uvm_component_utils(my_test)


/// Constructor
 function new (string name, uvm_component parent);
  super.new(name, parent);
 endfunction: new


/// Build Function (Build Sequences & Lower Level Components)
 function build_phase (uvm_phase phase);
  super.build_phase(phase);
    ...
    ...
 endfunction: build_phase
```

```
/// Reset Phase
task reset_phase (uvm_phase phase);
 phase.raise_objection(this);
 reset_seq.start(Env.Agnt.Sqnr);
 phase.drop_objection(this);
endtask: reset_phase

/// Configure Phase
task configure_phase (uvm_phase phase);
 phase.raise_objection(this);
 progam_cntrl_reg_seq.start(Env.Agnt.Sqnr);
 phase.drop_objection(this);
endtask: configure_phase

/// Main Phase
task main_phase (uvm_phase phase);
 phase.raise_objection(this);
 read_after_write_seq.start(Env.Agnt.Sqnr);
 phase.drop_objection(this);
endtask: main_phase
```

```
 /// Shutdown Phase
 task shutdown_phase (uvm_phase phase);
  phase.raise_objection(this);
  read_status_reg_seq.start(Env.Agnt.Sqnr);
  phase.drop_objection(this);
 endtask: shutdown_phase

endclass: my_test

//// Top Level Module
module top_tb;

  ...
  ...
  ...
 /// UVM Test Start Here
 initial
 begin
 uvm_config_db #(virtual dut_if) :: set(null,"*",
"BUS_vif", BUS);
 run_test(my_test);
 end
endmodule: top_tb
```

We can see in the above UVM Test Run-time Phases i.e. **reset_phase**, **configure_phase**, **main_phase** and **shutdown _phase,** each of the task is following a consistent pattern i.e. **raise an objection**, **execute a particular functionality using a sequence** and f**inally drop the objection**.

An important thing to understand here is – all the components in the UVM Testbench Hierarchy will execute the same sub-phase in parallel & even if few components are done with their user-process or sequence execution but these components have to wait until all the other components are done with their user-process execution. Once all the raised objection for that sub-phase are dropped that is the moment next phase can be entered. In the above example, we're dealing with the time consuming **Run phases** defined in the Test. So more the other components raises the objections more complex and complicated will be the objection mechanism process. **Hence, it is recommended as per the UVM Coding guidelines, Objections should only be raised and dropped from the Test**.

phase_ready_to_end:

For many of the UVM Testbenches, raising and dropping of the phase objections, as described above, during the normal lifetime of phases is quite sufficient. However, sometimes a component which does not raise and drop objections for every transaction due to performance issues likes to delay the transition from one phase to the next phase. The most common example is the **Scoreboard** doing some kind of analysis to the last transaction that may include some time-consuming behavior.

UVM recommended solution to delay the end of phase, after all the components have agreed to end the phase with 'all dropped' condition, is that the component should raise objection in the **phase_ready_to_end** method which is executed automatically by UVM once 'all dropped' condition is achieved during Run Phase.

Following is an example implementation code for **phase_ready_to_end** method.

```
function void scoreboard::phase_ready_to_end
(uvm_phase phase);

 if(phase.is(uvm_run_phase::get)) begin

  if (!check_state == 1'b1) begin

   phase.raise_objection(this, "Test Not Yet Ready
To End");

   fork begin

     `uvm_info("PRTE", "Phase Ready Testing",
UVM_LOW);

     wait_for_ok_to_finish();

     phase.drop_objection(this, "Test Ready to
End");

     end

   join_none

  end

 end

endfunction: phase_ready_to_end


task bidirect_bus_test::wait_for_ok_to_finish();

  #50; // Could be the logic here

  check_state = 1;

endtask: wait_for_ok_to_finish
```

In the above example code, at the end of Run Phase, if check_state is LOW, **Objection is raised** and then after the required processing is done, **Objection is dropped**. After that simulation is moved to the cleanup phase to finish off the Test.

set_drain_time:

Another approach supported by UVM is setting the drain time for the simulation environment. Drain time concept is related to the extra time allocated to the UVM environment to process the left over activities e.g. last packet analysis & comparison etc after all the stimulus is applied & processed. Lets see it via a example code:

```
class my_test extends uvm_test;

 `uvm_component_utils(my_test)




 ...

 ...


 task run_phase (uvm_phase phase);

   phase.raise_objection(this);

   my_seq.start(m_sequencer);

   /// Set a drain time for the Test

   phase.phase_done.set_drain_time(this, 20ns);

   phase.drop_objection(this);

 endtask: run_phase
```

```
endclass: my_test
```

While selecting the drain time value, some random time value may not be sufficient for all the Tests, hence it is necessary to study the requirements in detail and have good understanding of the scenarios which require extra time to complete the processing. Choosing the worst case time can be a good strategy so that all other scenarios could be covered with-in that time frame.