Without Objection - Touring the uvm_objection implementations - uses and improvements

Rich Edelman

Siemens EDA

Simple objections

- Every test needs one
- Controls “ending”

```
task run_phase(uvm_phase phase);
  phase.raise_objection(this);
  `uvm_info(get_type_name(), "...running", UVM_MEDIUM)
  ...
  phase.drop_objection(this);
endtask
```

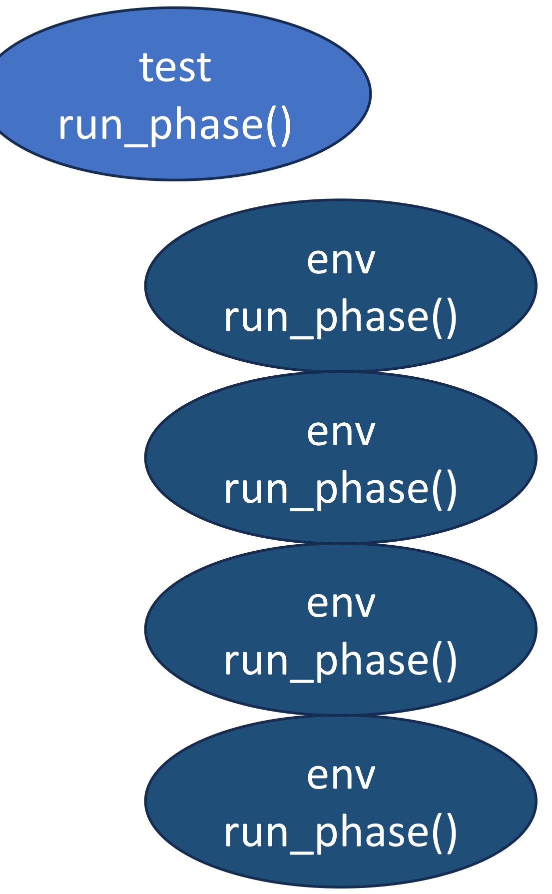
- It is hidden in the phase

- It’s synchronization
- It’s a barrier

You shall not pass until all are ready to pass

Imagine a test and 4 envs

- They all need to be ready to end before anyone ends
- For some reason they need to be synchronized on end



Simple Test

- raise_objection()
- Construct 4 sequences
- Start the sequences in parallel
- The sequences are done
- Print the "**ready to exit**" message
- drop_objection()

```
class test extends uvm_test;
  `uvm_component_utils(test)

  env e1, e2, e3, e4;
  seq s1, s2, s3, s4;

 task run_phase(uvm_phase phase);
    phase.raise_objection(this);
    s1 = seq::type_id::create("s1");
    s2 = seq::type_id::create("s2");
    s3 = seq::type_id::create("s3");
    s4 = seq::type_id::create("s4");
    fork
      s1.start(e1.a.sqr);
      s2.start(e2.a.sqr);
      s3.start(e3.a.sqr);
      s4.start(e4.a.sqr);
    join

    `uvm_info(get_type_name(),
      "...Objection Testing - 'test' ready to exit",
      UVM_MEDIUM)

    phase.drop_objection(this);
  endtask
endclass
```

What’s That Env Doing?

- Not much
- Wait, then Print the “READY” message

```
class env extends uvm_component;
  `uvm_component_utils(env)
  ...

  task run_phase(uvm_phase phase);
    phase.raise_objection(this);
    #100000;
    `uvm_info(get_type_name(), "...Objection Testing - 'env' ready to exit", UVM_MEDIUM)
    phase.drop_objection(this);
  endtask
endclass
```

Multiple 'env's – not ready to exit…then ready

- Test is ready to exit at 6484
- The remainder is ready at 100,000

```
UVM_INFO sequence.svh(19) @ 6476: uvm_test_top.e3.a.sqr@@s3 [transaction] ...started
UVM_INFO driver.svh(20) @ 6476: uvm_test_top.e3.a.d [driver] Executing: id: 7 READ(48, 5) #3999
UVM_INFO sequence.svh(23) @ 6484: uvm_test_top.e3.a.sqr@@s3 [transaction] ...finished
UVM_INFO test.svh(46) @ 6484: uvm_test_top [test] ...Objection Testing - 'test' ready to exit
UVM_INFO env.svh(18) @ 100000: uvm_test_top.e4 [env] ...Objection Testing - 'env' ready to exit
UVM_INFO env.svh(18) @ 100000: uvm_test_top.e3 [env] ...Objection Testing - 'env' ready to exit
UVM_INFO env.svh(18) @ 100000: uvm_test_top.e2 [env] ...Objection Testing - 'env' ready to exit
UVM_INFO env.svh(18) @ 100000: uvm_test_top.e1 [env] ...Objection Testing - 'env' ready to exit
```

What's the scope of the UVM Objections?

- The "objection" code has remained stable across releases
- The Objection code itself is more than 1000 lines

uvm-1.1d	uvm 1.2	uvm-1800	
90	90	94	src/base/uvm_event_callback.svh
361	394	432	src/base/uvm_event.svh
211	209	204	src/base/uvm_barrier.svh
1502	**1453**	**1241**	**src/base/uvm_objection.svh**
338	341	349	src/base/uvm_heartbeat.svh
1192	1198	1212	src/base/uvm_callback.svh
300	300	307	src/macros/uvm_callback_defines.svh
3994	3985	3839	total

What is it?

- The uvm_objection class has
  - tasks (5)
  - functions (28)
  - data (17)
- Key entry points
  - raise_objection()
  - drop_objection()

Classes

- Tasks
  - virtual all_dropped
  - m_execute_scheduled_forks
  - m_forked_drain
  - wait_for
  - wait_for_total_count
- Functions
  - virtual clear
  - virtual convert2string
  - virtual create
  - display_objections
  - virtual do_copy
  - virtual drop_objection
  - virtual dropped
  - get_drain_time
  - get_objection_count
  - get_objection_total
  - get_objectors
  - get_propagate_mode
  - get_type
  - virtual get_type_name
  - m_display_objections
  - m_drop
  - m_get_parent
  - m_init_objections
  - m_propagate
  - m_raise
  - m_report
  - m_set_hier_mode
  - new
  - virtual raise_objection
  - virtual raised
  - set_drain_time
  - set_propagate_mode
  - trace_mode
- Data
  - bit m_cleared
  - uvm_objection_context_object m_context_pool
  - process m_drain_proc
  - memory m_drain_time
  - uvm_objection_events m_events
  - uvm_objection_context_object m_forked_contexts
  - uvm_objection_context_object m_forked_list
  - uvm_objection m_objections
  - bit m_prop_mode
  - bit m_register_cb_uvm_objection_callback
  - uvm_objection_context_object m_scheduled_contexts
  - uvm_objection_context_object m_scheduled_list
  - memory m_source_count
  - uvm_root m_top
  - bit m_top_all_dropped
  - memory m_total_count
  - bit m_trace_mode

uvm_objection.svh - uvm_objection

```
//------------------------------------------------------------------------------
// Objections provide a facility for coordinating status information between
// two or more participating components, objects, and even module-based IP.
//
// Tracing of objection activity can be turned on to follow the activity of
// the objection mechanism. It may be turned on for a specific objection
// instance with <uvm_objection::trace_mode>, or it can be set for all
// objections from the command line using the option +UVM_OBJECTION_TRACE.
//------------------------------------------------------------------------------

// @uvm-ieee 1800.2-2020 auto 10.5.1
// @uvm-ieee 1800.2-2020 auto 10.5.1.1
class uvm_objection extends uvm_report_object;
  `uvm_register_cb(uvm_objection, uvm_objection_callback)

  protected bit     m_trace_mode;
  protected int     m_source_count[uvm_object];
  protected int     m_total_count [uvm_object];
  protected time    m_drain_time  [uvm_object];
  protected uvm_objection_events m_events [uvm_object];
  /*protected*/ bit     m_top_all_dropped;

  protected uvm_root m_top;

  static uvm_objection m_objections[$];

  //// Drain Logic

  // The context pool holds used context objects, so that
  // they're not constantly being recreated.  The maximum
  // number of contexts in the pool is equal to the maximum
  // number of simultaneous drains you could have occuring,
  // both pre and post forks.
  //
  // There's the potential for a programmability within the
  // library to dictate the largest this pool should be allowed
  // to grow, but that seems like overkill for the time being.
  local static uvm_objection_context_object m_context_pool[$];

  // These are the active drain processes, which have been
  // forked off by the background process.  A raise can
  // use this array to kill a drain.
`ifndef UVM_USE_PROCESS_CONTAINER
  local process m_drain_proc[uvm_object];
`else
  local process_container_c m_drain_proc[uvm_object];
`endif

  // These are the contexts which have been scheduled for
  // retrieval by the background process, but which the
  // background process hasn't seen yet.
  local static uvm_objection_context_object m_scheduled_list[$];

  // Once a context is seen by the background process, it is
  // removed from the scheduled list, and placed in the forked
  // list.  At the same time, it is placed in the scheduled
  // contexts array.  A re-raise can use the scheduled contexts
  // array to detect (and cancel) the drain.
  local uvm_objection_context_object m_scheduled_contexts[uvm_object];
  local uvm_objection_context_object m_forked_list[$];

  // Once the forked drain has actually started (this occurs
  // ~1 delta AFTER the background process schedules it), the
  // context is removed from the above array and list, and placed
  // in the forked_contexts list.
  local uvm_objection_context_object m_forked_contexts[uvm_object];

  protected bit m_prop_mode = 1;
  protected bit m_cleared; /* for checking obj count<0 */


  // Function -- NODOCS -- new
  //
  // Creates a new objection instance. Accesses the command line
  // argument +UVM_OBJECTION_TRACE to turn tracing on for
```

Design | Library | UVM Testbench | Classes | t.sv | uvm_objection.svh | uvm_objection.svh

uvm_phase::raise_objection()

- phase.raise_objection()
  - objection.raise_objection()
    - objection.raise()
      - objection.raised()
        - component.raised() -> *For you to implement*
- I mean. It's code… Here's the code
  - And the code and the code…

uvm_phase::raise_objection()

```
function void uvm_phase::raise_objection (uvm_object obj, string description="", int count=1);
  uvm_objection phase_done;
  phase_done = get_objection();
  if (phase_done != null)
    phase_done.raise_objection(obj,description,count);
  else
    m_report_null_objection(obj, description, count, "raise");
endfunction
```

uvm_objection::raise_objection()

```
virtual function void raise_objection (uvm_object obj=null, string description="", int count=1);
   if(obj == null)
     obj = m_top;
   m_cleared = 0;
   m_top_all_dropped = 0;
   m_raise (obj, obj, description, count);
 endfunction
```

uvm_objection::m_raise() 1

```
function void m_raise (uvm_object obj, uvm_object source_obj, string description="", int count=1);
   int idx;
   uvm_objection_context_object ctxt;

   // Ignore raise if count is 0
   if (count == 0)return;

   if (m_total_count.exists(obj))
     m_total_count[obj] += count;
   else
     m_total_count[obj] = count;

   if (source_obj==obj) begin
     if (m_source_count.exists(obj))
       m_source_count[obj] += count;
     else
       m_source_count[obj] = count;
   end

   if (m_trace_mode)
     m_report(obj,source_obj,description,count,"raised");
   raised(obj, source_obj, description, count);
```


accellera SYSTEMS INITIATIVE
DVCON CONFERENCE AND EXHIBITION UNITED STATES

uvm_objection::m_raise() 2

```
// Handle any outstanding drains...

   // First go through the scheduled list
   idx = 0;
   while (idx < m_scheduled_list.size()) begin
       if ((m_scheduled_list[idx].obj == obj) &&
           (m_scheduled_list[idx].objection == this)) begin
           // Caught it before the drain was forked
           ctxt = m_scheduled_list[idx];
           m_scheduled_list.delete(idx);
           break;
       end
       idx++;
   end

   // If it's not there, go through the forked list
   if (ctxt == null) begin
       idx = 0;
       while (idx < m_forked_list.size()) begin
           if (m_forked_list[idx].obj == obj) begin
               // Caught it after the drain was forked,
               // but before the fork started
               ctxt = m_forked_list[idx];
               m_forked_list.delete(idx);
               m_scheduled_contexts.delete(ctxt.obj);
               break;
           end
           idx++;
       end
   end

   // If it's not there, go through the forked contexts
   if (ctxt == null) begin
       if (m_forked_contexts.exists(obj)) begin
           // Caught it with the forked drain running
           ctxt = m_forked_contexts[obj];
           m_forked_contexts.delete(obj);
           // Kill the drain
`ifndef UVM_USE_PROCESS_CONTAINER
           m_drain_proc[obj].kill();
           m_drain_proc.delete(obj);
`else
           m_drain_proc[obj].p.kill();
           m_drain_proc.delete(obj);
`endif
       end
   end
```


accellera SYSTEMS INITIATIVE
DVCON CONFERENCE AND EXHIBITION UNITED STATES

uvm_objection::m_raise() 3

```
if (ctxt == null) begin
    // If there were no drains, just propagate as usual

    if (!m_prop_mode && obj != m_top)
      m_raise(m_top,source_obj,description,count);
    else if (obj != m_top)
      m_propagate(obj, source_obj, description, count, 1, 0);
end
else begin
    // Otherwise we need to determine what exactly happened
    int diff_count;

    // Determine the diff count, if it's positive, then we're
    // looking at a 'raise' total, if it's negative, then
    // we're looking at a 'drop', but not down to 0.  If it's
    // a 0, that means that there is no change in the total.
    diff_count = count - ctxt.count;

    if (diff_count != 0) begin
        // Something changed
        if (diff_count > 0) begin
            // we're looking at an increase in the total
            if (!m_prop_mode && obj != m_top)
              m_raise(m_top, source_obj, description, diff_count);
            else if (obj != m_top)
              m_propagate(obj, source_obj, description, diff_count, 1, 0);
        end
        else begin
            // we're looking at a decrease in the total
            // The count field is always positive...
            diff_count = -diff_count;
            if (!m_prop_mode && obj != m_top)
              m_drop(m_top, source_obj, description, diff_count);
            else if (obj != m_top)
              m_propagate(obj, source_obj, description, diff_count, 0, 0);
        end
    end

    // Cleanup
    ctxt.clear();
    m_context_pool.push_back(ctxt);
end
```


accellera SYSTEMS INITIATIVE
2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

uvm_objection::raised()

```
virtual function void raised (uvm_object obj, uvm_object source_obj, string description, int count);
  uvm_component comp;
  if ($cast(comp,obj))
    comp.raised(this, source_obj, description, count);
  `uvm_do_callbacks(uvm_objection,uvm_objection_callback,raised(this,obj,source_obj,description,count))
  if (m_events.exists(obj))
     ->m_events[obj].raised;
endfunction
```

comp.raised()

```
// Function -- NODOCS -- raised
//
// The ~raised~ callback is called when this or a descendant of this component
// instance raises the specified ~objection~. The ~source_obj~ is the object
// that originally raised the objection.
// The ~description~ is optionally provided by the ~source_obj~ to give a
// reason for raising the objection. The ~count~ indicates the number of
// objections raised by the ~source_obj~.

// @uvm-ieee 1800.2-2020 auto 13.1.6.1
virtual function void raised (uvm_objection objection, uvm_object source_obj, string description, int count);
endfunction
```

- I’m not saying there’s a lot of code there, but…
- Not good or bad code
- Just a lot of code
- But that’s just ‘raise_objection’ on to ‘drop_objection’

uvm_phase::drop_objection()

```
function void uvm_phase::drop_objection (uvm_object obj, string description="", int count=1);
  uvm_objection phase_done;
  phase_done = get_objection();
  if (phase_done != null)
    phase_done.drop_objection(obj,description,count);
  else
    m_report_null_objection(obj, description, count, "drop");
endfunction
```

uvm_objection::drop_objection()

```
virtual function void drop_objection (uvm_object obj=null, string description="", int count=1);
   if(obj == null)
     obj = m_top;
   m_drop (obj, obj, description, count, 0);
 endfunction
```

uvm_objection::m_drop() 1

```
function void m_drop (uvm_object obj, uvm_object source_obj, string description="",
                      int count=1, int in_top_thread=0);

   // Ignore drops if the count is 0
   if (count == 0) return;

   if (!m_total_count.exists(obj) || (count > m_total_count[obj])) begin
     if(m_cleared) return;
     uvm_report_fatal(... "count below zero"});
     return;
   end

   if (obj == source_obj) begin
     if (!m_source_count.exists(obj) || (count > m_source_count[obj])) begin
       if(m_cleared) return;
       uvm_report_fatal(... "count below zero"});
       return;
     end
     m_source_count[obj] -= count;
   end

   m_total_count[obj] -= count;

   if (m_trace_mode)
     m_report(obj,source_obj,description,count,"dropped");

   dropped(obj, source_obj, description, count);
```

uvm_objection::m_drop() 2

```
// if count != 0, no reason to fork
if (m_total_count[obj] != 0) begin
  if (!m_prop_mode && obj != m_top)
    m_drop(m_top,source_obj,description, count, in_top_thread);
  else if (obj != m_top) begin
    this.m_propagate(obj, source_obj, description, count, 0, in_top_thread);
  end
end
else begin
    uvm_objection_context_object ctxt;
    if (m_context_pool.size())
      ctxt = m_context_pool.pop_front();
    else
      ctxt = new;

    ctxt.obj = obj;
    ctxt.source_obj = source_obj;
    ctxt.description = description;
    ctxt.count = count;
    ctxt.objection = this;
```

uvm_objection::m_drop() 3

```
        // Need to be thread-safe, let the background
        // process handle it.

        // Why don't we look at in_top_thread here?  Because
        // a re-raise will kill the drain at object that it's
        // currently occuring at, and we need the leaf-level kills
        // to not cause accidental kills at branch-levels in
        // the propagation.

        // Using the background process just allows us to
        // separate the links of the chain.
        m_scheduled_list.push_back(ctxt);

    end // else: !if(m_total_count[obj] != 0)

  endfunction
```


accellera SYSTEMS INITIATIVE
2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

uvm_objection::dropped()

```
// Function -- NODOCS -- dropped
//
// Objection callback that is called when a <drop_objection> has reached ~obj~.
// The default implementation calls <uvm_component::dropped>.

// @uvm-ieee 1800.2-2020 auto 10.5.1.4.2
virtual function void dropped (uvm_object obj, uvm_object source_obj, string description, int count);
  uvm_component comp;
  if($cast(comp,obj))
    comp.dropped(this, source_obj, description, count);
  `uvm_do_callbacks(uvm_objection,uvm_objection_callback,dropped(this,obj,source_obj,description,count))
  if (m_events.exists(obj))
     ->m_events[obj].dropped;
endfunction
```

uvm_component::dropped()

```
// Function -- NODOCS -- dropped
 //
 // The ~dropped~ callback is called when this or a descendant of this component
 // instance drops the specified ~objection~. The ~source_obj~ is the object
 // that originally dropped the objection.
 // The ~description~ is optionally provided by the ~source_obj~ to give a
 // reason for dropping the objection. The ~count~ indicates the number of
 // objections dropped by the ~source_obj~.

 // @uvm-ieee 1800.2-2020 auto 13.1.6.2
 virtual function void dropped (uvm_objection objection, uvm_object source_obj, string description, int count);
 endfunction
```

uvm_objection::m_propagate()

```
// Function- m_propagate
//
// Propagate the objection to the objects parent. If the object is a
// component, the parent is just the hierarchical parent. If the object is
// a sequence, the parent is the parent sequence if one exists, or
// it is the attached sequencer if there is no parent sequence.
//
//        obj : the uvm_object on which the objection is being raised or lowered
// source_obj : the root object on which the end user raised/lowered the
//              objection (as opposed to an anscestor of the end user object)
//      count : the number of objections associated with the action.
//      raise : indicator of whether the objection is being raised or lowered. A 1 indicates the objection is being raised
//
function void m_propagate (uvm_object obj, uvm_object source_obj, string description,
                           int count, bit raise, int in_top_thread);
  if (obj != null && obj != m_top) begin
    obj = m_get_parent(obj);
    if(raise)
      m_raise(obj, source_obj, description, count);
    else
      m_drop(obj, source_obj, description, count, in_top_thread);
  end
endfunction
```

uvm_objection::m_get_parent()

```
// Function- m_get_parent
 //
 // Internal method for getting the parent of the given ~object~.
 // The ultimate parent is uvm_top, UVM's implicit top-level component.

 function uvm_object m_get_parent(uvm_object obj);
   uvm_component comp;
   uvm_sequence_base seq;
   if ($cast(comp, obj)) begin
     obj = comp.get_parent();
   end
   else if ($cast(seq, obj)) begin
      obj = seq.get_sequencer();
   end
   else
     obj = m_top;
   if (obj == null)
     obj = m_top;
   return obj;
 endfunction
```

The object is a ‘uvm_component’

The object is a ‘uvm_sequence_base’

The object is something else

Final Check: If null, use TOP

That’s a lot. And we really didn’t cover much

- There’s still more code - Lots of code
- Does lots of things

Uvm_objection is big and has layers

- What if you just want to “synchronize”?
- What about something SIMPLE?
  - That still acts like a BARRIER?
- Can I just use the UVM_BARRIER?
  - Yes.
  - 204 lines, so much better.
- Can I write something even smaller?

```
class barrier;
  int count;

  task raise();
    //#0;
    count = count + 1;
  endtask

  task drop();
    //#0;
    if (count == 0)
      return;
    count = count - 1;
    wait (count == 0);
  endtask
endclass
```


accellera SYSTEMS INITIATIVE
2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Imagine coordinating…

- Every “coordinated place” needs a reference to the “coordinator”
  - Objection handle
  - Barrier handle

```
objection start;
objection middle;
objection endx;

start.raise();
start.drop();

middle.raise();
middle.drop();

endx.raise();
endx.drop();
```

```
barrier start;
barrier middle;
barrier endx;

start.raise();
start.drop();

middle.raise();
middle.drop();

endx.raise();
endx.drop();
```

```
user_objections ua;


ua.raise(“start”);
ua.drop(“start”);

ua.raise(“middle”);
ua.drop(“middle”);

ua.raise(“end”);
ua.drop(“end”);
```


accellera SYSTEMS INITIATIVE
2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Barrier-by-Name

- Associative array
  - Lookup barrier by name
- barrier get(name)
- raise(name)
- drop(name)

```
class user_objections;
  barrier barriers[string];

  function barrier get(string barrier_name);
    barrier b;
    if (barriers.exists(barrier_name)) begin
      b = barriers[barrier_name];
    end
    else begin
      b = new();
      barriers[barrier_name] = b;
    end
    return b;
  endfunction

  task raise(string barrier_name, string thread_name);
    barrier b;
    b = get(barrier_name);
    b.raise();
  endtask

  task drop(string barrier_name, string thread_name);
    barrier b;
    b = get(barrier_name);
    b.drop();
  endtask
endclass
```

Used In A Class

```
user_objections ua = new();

class Class1;
  task body();
    forever begin
      int n;
      n = $urandom_range(10, 1);
      ua.raise("start", thread_name);
      for (int i = 0; i < n; i++) begin
        #n;
      end
      ua.drop("start", thread_name);
      $display("@%t: %m done done done with %s", $time, "start");

      n = $urandom_range(10, 1);
      ua.raise("middle", thread_name);
      for (int i = 0; i < n; i++) begin
        #n;
      end
      ua.drop("middle", thread_name);
      $display("@%t: %m done done done with %s", $time, "middle");

      n = $urandom_range(10, 1);
      ua.raise("end", thread_name);
      for (int i = 0; i < n; i++) begin
        #n;
      end
      ua.drop("end", thread_name);
      $display("@%t: %m done done done with %s", $time, "end");
    end
  endtask
endclass
```


accellera SYSTEMS INITIATIVE
2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

In A Module

```
module a();
  initial begin
    forever begin
      int n;
      n = $urandom_range(10, 1);
      ua.raise("start", thread_name);
      for (int i = 0; i < n; i++) begin
        #n;
      end
      ua.drop("start", thread_name);
      $display("@%t: %m done done done with %s", $time, "start");

      ua.raise("middle", thread_name);
      n = $urandom_range(10, 1);
      for (int i = 0; i < n; i++) begin
        #n;
      end
      ua.drop("middle", thread_name);
      $display("@%t: %m done done done with %s", $time, "middle");

      ua.raise("end", thread_name);
      n = $urandom_range(10, 1);
      for (int i = 0; i < n; i++) begin
        #n;
      end
      ua.drop("end", thread_name);
      $display("@%t: %m done done done with %s", $time, "end");
      $display("@%t  %m ENDEND", $time);
    end
  end
endmodule
```

The top – the whole thing

- Construct the classes
- Start them running
- Module initial blocks already running

```
module top();
  a A();
  b B();
  c C();
  Class1 c1;
  Class2 c2;

  initial begin
    setup();
    c1 = new();
    c2 = new();
    fork
      c1.body();
      c2.body();
    join_none
    repeat(1000)
      #10;
    $finish(2);
  end
endmodule
```


accellera SYSTEMS INITIATIVE

Running…

"start"

```
 81: t_sv_unit.Class1.body done done done with start
 81: t_sv_unit.Class2.body done done done with start
 81: top.A done done done with start
 81: top.B done done done with start
 81: top.C done done done with start
```

"middle"

```
145: top.C done done done with middle
145: top.B done done done with middle
145: t_sv_unit.Class1.body done done done with middle
145: top.A done done done with middle
145: t_sv_unit.Class2.body done done done with middle
```

"end"

```
209: top.A done done done with end
209: t_sv_unit.Class2.body done done done with end
209: top.C done done done with end
209: top.B done done done with end
209: t_sv_unit.Class1.body done done done with end
```

"start"

```
290: top.C done done done with start
290: top.B done done done with start
290: t_sv_unit.Class2.body done done done with start
290: top.A done done done with start
290: t_sv_unit.Class1.body done done done with start
```

```
$display("@%t: %m done done done with %s", $time, "start");
```

The Real Code

- A synchronization region is a stream
- The barrier raise/drop is a transaction

Adding "Visualizations" → Transactions

```
int streams[string];

function void setup();
  streams["start"]  = $create_transaction_stream("start");
  streams["middle"] = $create_transaction_stream("middle");
  streams["end"]    = $create_transaction_stream("end");
endfunction

class barrier;
  int count;
  int tr[string];
  ...
endclass

class user_objections;
  barrier barriers[string];

  task raise(string barrier_name, string thread_name);
    barrier b;
    b = get(barrier_name);
    b.tr[thread_name] = $begin_transaction(streams[barrier_name], thread_name);
    b.raise();
  endtask

  task drop(string barrier_name, string thread_name);
    barrier b;
    b = get(barrier_name);
    b.drop();
    $end_transaction(b.tr[thread_name]);
    $free_transaction(b.tr[thread_name]);
  endtask
endclass
```

Seeing it

- “middle” synchronization blown up

_unit.middle
middle(0)
middle(5)
0
class1
S0
payload
1
B
S0
payload
2
A
S0
payload
3
class2
S0
payload
4
C
S0
payload

accellera
SYSTEMS INITIATIVE

Expanding “start”, “middle”, “end”

“start”

“middle”

“end”

Summary

- Use objections.
  - But sparingly. End of Test is a good place to use them
  - Never use them in a high-frequency operation
- Synchronize with something simpler
  - uvm_barrier
  - Write your own code – test it → It's NOT illegal to write your own code – just keep it simple
  - The example barrier code is trivially simple
    - Add debug – "what objects are synchronized?"
    - Add Visualization

Questions?

- Source code available – email rich.edelman@siemens.com