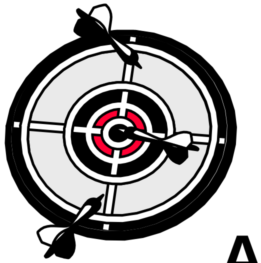
Agenda: Day 3

DAY 3

9	UVM Advanced Sequence/Sequencer
10	UVM Phasing and Objections
11	UVM Register Abstraction Layer (RAL)
12	Summary

Unit Objectives



After completing this unit, you should be able to:

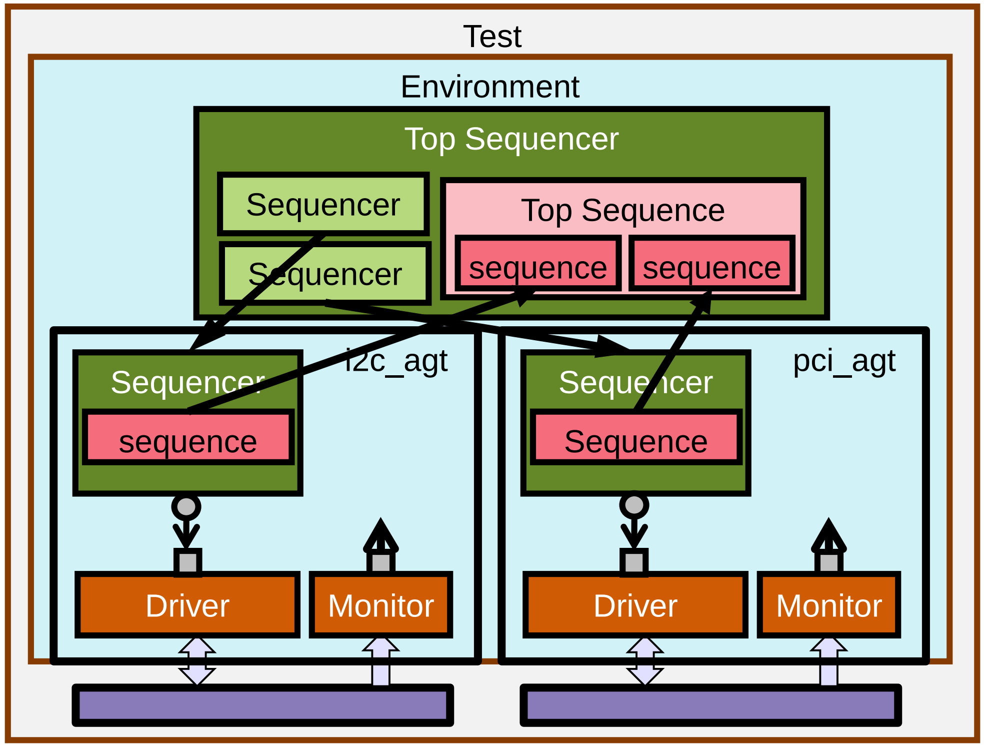
- Control execution order of sequences within a phase with a Top Sequence
- Manage synchronization of concurrent sequence executions within a phase with uvm_event

Managing Sequence Execution

- **How can one coordinate sequences running across multiple agents?**
  - e.g. Reset of one part of the DUT must be done before another part of the DUT can go through a reset sequence
- **Solution: Top Sequence (Sequence Execution Manager)**
  - Explicitly manage the execution of sequences across multiple agents within a phase
  - Allows fine grained control of the sequence execution order
  - Doesn’t have its own sequence item, only used to manage the execution of other sequences
  - Resources required by the sequence execution manager resides in a support sequencer

Managing Sequence Execution

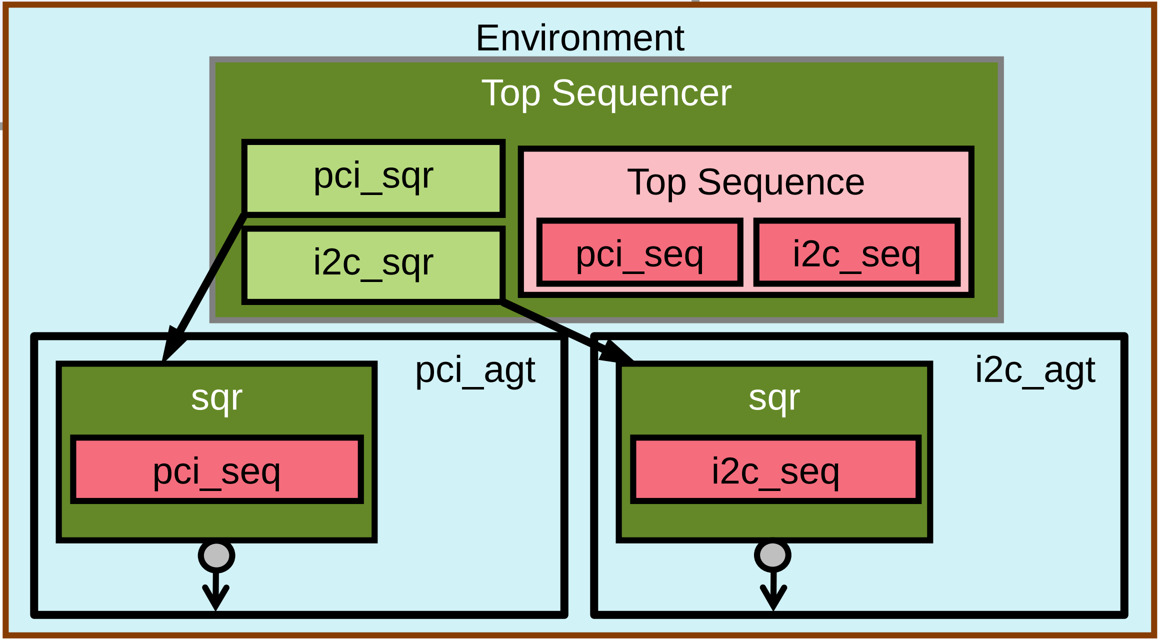
- Typically, a sequence only interacts with a single agent to manage the activities of a single DUT interface
- When multiple DUT interfaces need to be synchronized, a upper layer sequence and sequencer are required



Top Sequencer

- **Not associated with any agent**
- **Contains resources needed by the top Sequence**
  - Example: sequencer handles

```
class top_sequencer extends uvm_sequencer;
  `uvm_component_utils(top_sequencer)
  // Constructor not shown
  pci_sequencer pci_sqr;
  i2c_sequencer i2c_sqr;
endclass
```



Top Sequence

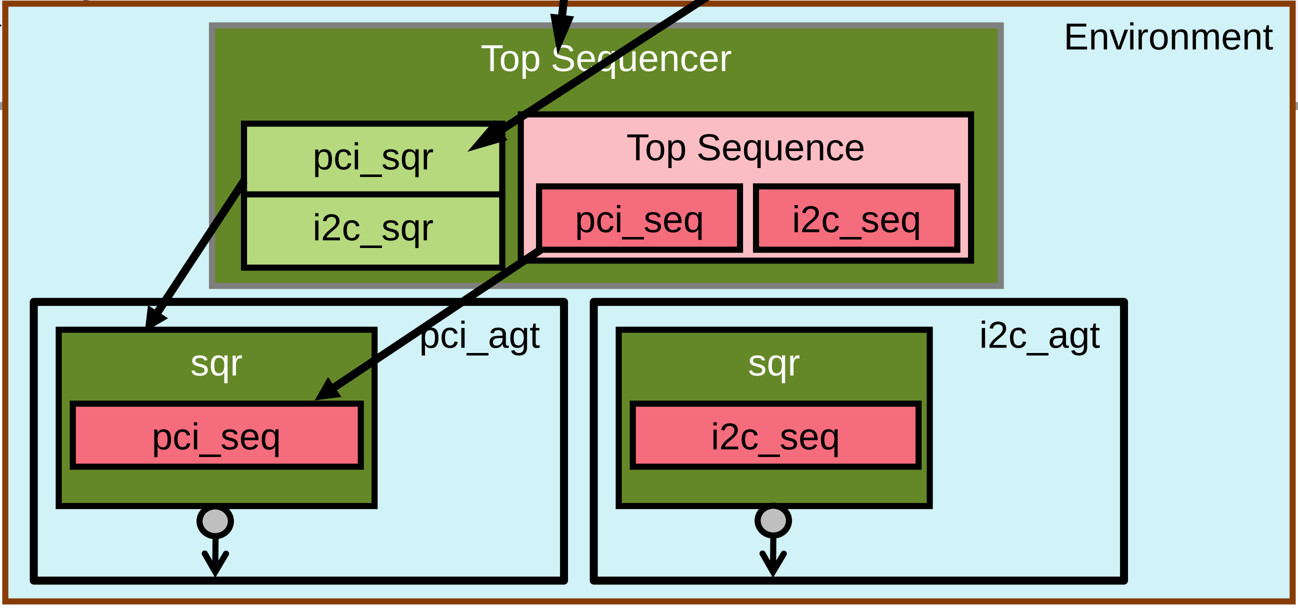
- **Embed sequences to be managed**
- **In `body()` method, manage these sequence execution**

```
class top_sequence extends uvm_sequence;
  `uvm_object_utils(top_sequence)
  `uvm_declare_p_sequencer(top_sequencer)
  pci_sequence  pci_seq;
  i2c_sequence  i2c_seq;
  virtual task body();
    `uvm_do_on(pci_seq, p_sequencer.pci_sqr)
    `uvm_do_on(i2c_seq, p_sequencer.i2c_sqr);
  end
endcl
```

Derived from **`uvm_sequence`** class

Sequences to be managed

Macro creates a handle called **`p_sequencer`** to access top sequencer content



Executing Top Sequence

- Create the top sequencer in build phase
- Disable the managed sequencers by setting their "default_sequence" to null
- Configure the top sequencer to execute the top sequence in the desired phase

```
class test_new extends test_base; // other code not shown
  top_sequencer top_sqr;
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    // creation of other objects not shown
    top_sqr = top_sequencer::type_id::create("top_sqr", this);
    uvm_config_db#(uvm_object_wrapper)::set(this,
                  "env.pci_agt.sqr.main_phase", "default_sequence", null);
    uvm_config_db#(uvm_object_wrapper)::set(this,
                  "env.i2c_agt.sqr.main_phase", "default_sequence", null);
    uvm_config_db#(uvm_object_wrapper)::set(this, "top_sqr.main_phase",
                       "default_sequence", top_sequence::get_type());
  endfunction
//  continued on the next page
```

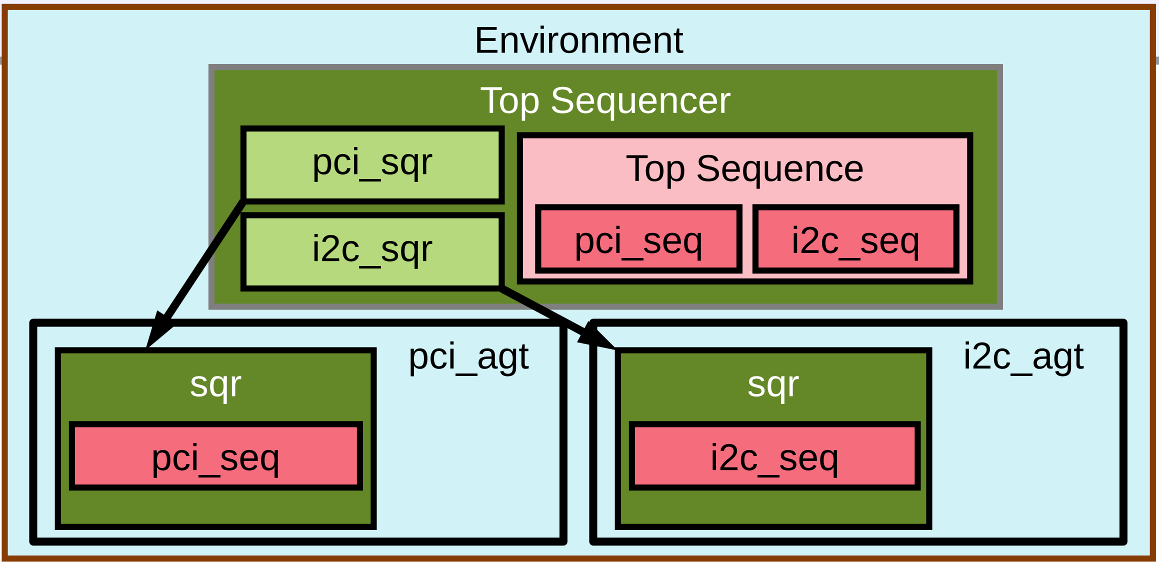
Disable managed sequencers

Set support sequencer to execute sequence manager

Set Top Sequencer Content

- **In connect phase, set top sequencer's sequencer handles to the sequencers to be managed**

```
//  Continued from previous page
  virtual function void connect_phase(uvm_phase phase);
    top_sqr.pci_sqr = pci_agt.sqr;
    top_sqr.i2c_sqr = i2c_agt.sqr
  endfunction
endclass
```



Sequence Execution Management

What if sequence needs to start in middle of another sequence?

```
class top_sequence extends uvm_sequence;  // other code not shown
  virtual task body();
    fork
      `uvm_do_on(seq0, p_sequencer.bfm0_sqr);
      begin
        wait(...);    // wait for some event before processing
        `uvm_do_on(seq1, p_sequencer.bfm1_sqr);
      end
    join
  endtask
endclass
```

- **Instead of SystemVerilog wait, use the more powerful `uvm_event` via `uvm_pool`**

Synchronization Mechanism: uvm_event

- Wait for one trigger to releases all waits
- Essential methods of the class:

UVM-1.1

```
class uvm_event extends uvm_object;
  function new(string name="");
  virtual function void trigger(uvm_object data=null); // data optional

  virtual function void reset(bit wakeup=0);

  virtual function bit is_on();
  virtual function bit is_off();

  virtual task wait_on(bit delta=0);
  virtual task wait_off(bit delta=0);

  virtual task wait_trigger();  // equivalent to @(event);
  virtual task wait_trigger_data(output uvm_object data); // with data

  virtual task wait_ptrigger(); // equivalent to wait(event.triggered)
  virtual task wait_ptrigger_data(output uvm_object data); // with data

  virtual function void cancel();
endclass : uvm_event
```

Emulates Verilog event with more features

Triggers uvm_event

After trigger, uvm_event stays on until reset is called

Query state of uvm_event

Wait for state. Does not block if true.

Cancels wait

Synchronization Mechanism: uvm_event

- **Essentially the same, except allows user to specify default transaction type**

UVM-1.2

```
class uvm_event_base extends uvm_object;
  function new(string name="");
  virtual function void reset(bit wakeup=0);
  virtual function bit is_on();
  virtual function bit is_off();
  virtual task wait_on(bit delta=0);
  virtual task wait_off(bit delta=0);
  virtual task wait_trigger();   // equivalent to @(event);
  virtual task wait_ptrigger(); // equivalent to wait(event.triggered)
  virtual function void cancel();
endclass : uvm_event_base
```

```
class uvm_event #(type T=uvm_object) extends uvm_event_base;
  function new(string name="");
  virtual function void trigger(T data=null);
  virtual task wait_trigger_data(output T data);
  virtual task wait_ptrigger_data(output T data);
  virtual function T get_trigger_data();
  virtual function void cancel();
endclass : uvm_event
```

Protecting Stimulus (Grab/Ungrab)

- **Sequence can reserve a sequencer for exclusive use**
  - Until explicitly released
  - Other requests for exclusive use or stimulus injection are blocked
  - Typically needed for interrupt sequence execution

```
class interrupt_sequence extends uvm_sequence #(packet);
  // utils macro and constructor not shown
  virtual task body();
   grab();
   repeat(10) begin
    `uvm_info("INTR", "In body()", UVM_HIGH)
    `uvm_do_with(req,{addr == 6; data == 6;})
   end
   ungrab();
  endtask
 endclass
```

Sequence requesting grab() reserves parent sequencer for exclusive use

ungrab() method releases grab for default/specified sequencer.

Unit Objectives Review

Having completed this unit, you should be able to:

- **Control execution order of sequences within a phase with a Top Sequence**
- **Manage synchronization of concurrent sequence executions within a phase with uvm_event**

Appendix

Sequence Arbitration & Priority

Reactive Sequence

Interrupt Sequence

Sequence Library

Resource Pool

Debugging uvm_event_pool issues

Sequence Arbitration & Priority

Sequence Arbitration and Priority (1/2)

- **Once a sequence is started on a sequencer, it must be arbitrated for access to the sequencer resources**
  - Such as the seq_item_port
- **By default, all sequences have priority of 100**
  - Child sequence defaults to that of the parent sequence
- **The higher the value, the higher the priority**
- **The priority can be set via macro**
  - `uvm_do_pri(seq, pri)
- **This priority can change dynamically through the course of simulation**
  - seq.set_priority(300);

Sequence Arbitration and Priority (2/2)

- **Sequencers use one of several arbitration schemes**
  - SEQ_ARB_FIFO - Requests are granted in FIFO order (default)
  - SEQ_ARB_WEIGHTED - Requests are granted randomly by weight
  - SEQ_ARB_RANDOM - Requests are granted randomly
  - SEQ_ARB_STRICT_FIFO - Requests at highest priority granted in fifo order
  - SEQ_ARB_STRICT_RANDOM - Requests at highest priority granted in randomly
  - SEQ_ARB_USER Arbitration is delegated to the user-defined function, user_priority_arbitration.
- **The default arbitration scheme (SEQ_ARB_FIFO) is unaffected by sequence priority.**

Code Example

- **Set interrupt sequence to the highest priority**

```
task my_sequence::body(uvm_phase phase);
  get_sequencer().set_arbitration(SEQ_ARB_STRICT_FIFO)
  fork
    `uvm_do_pri(burst_seq);
    `uvm_do_pri(noise_seq);
    `uvm_do_pri(inter_seq, 3000);
  join
endtask
```

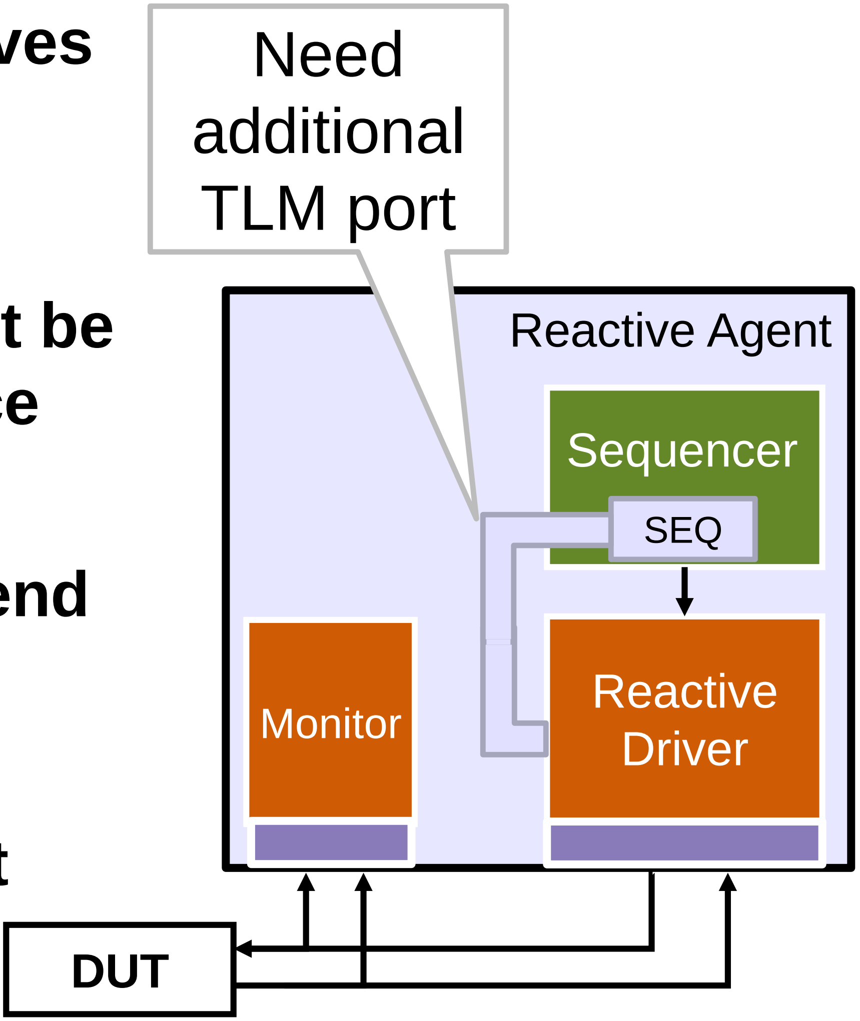
Set the priority when started

```
task my_test::run_phase(uvm_phase phase);
  phase.raise_objection(this);
  env.agt.sqr.set_arbitration(SEQ_ARB_STRICT_FIFO);
  fork
    burst_seq.start(env.agt.sqr);
    noise_seq.start(env.agt.sqr);
    inter_seq.start(env.agt.sqr,,3000);
  join
  phase.drop_objection(this);
endtask
```

Reactive Sequence

Reactive Sequences

- **Activity is initiated by the reactive driver when it receives a transaction from the DUT**
  - Not the Sequencer
- **The Partial Transaction must be sent to the reactive sequence**
- **The sequence needs to formulate a response and send it back to the driver**
- **These requirements make reactive sequences different**



Reactive TLM Port Setup

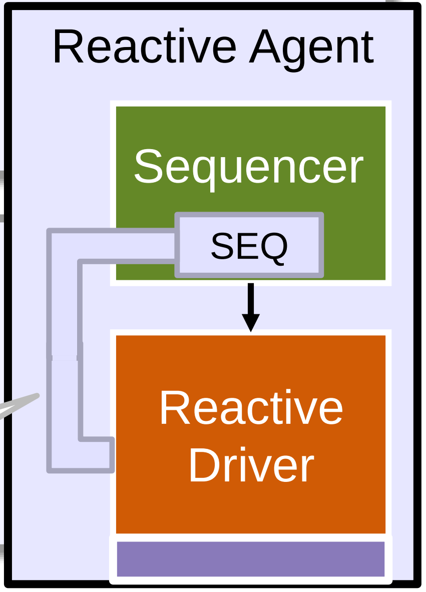
```
class reactive_driver extends uvm_driver#(trans);
  uvm_blocking_get_imp#(trans, reactive_driver) get_export;
  virtual task get(output trans tr);
    // Shown later
  endtask
endclass
```

Declare a TLM “get” port and associated get method

```
class reactive_sqr extends uvm_sequencer (trans);
  uvm_blocking_get_port#(trans) get_port;
  virtual task wait_for_req(uvm_sequence_base seq, output trans req);
    wait_for_grant(seq);
    get_port.get(req);
  endtask
endclass
```

```
class reactive_agent extends uvm_agent;
  virtual function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    sqr.get_port.connect(drv.get_export);
  endfunction
endclass
```

Connect reactive TLM port



Reactive Sequence Request

```
class reactive_sequence extends uvm_sequence #(trans);
  `uvm_declare_p_sequencer(reactive_sequencer)
  virtual task body();
    forever begin
      p_sequencer.wait_for_req(this, req);
      // ...
    end
```

Wait for request from driver

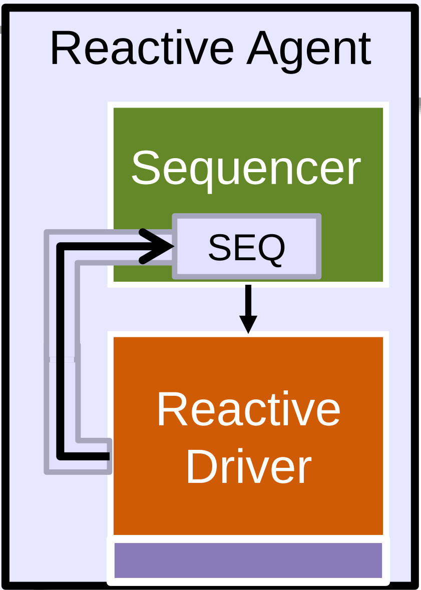
Do not raise or drop objection in sequence

```
class reactive_sequencer extends uvm_sequencer #(trans);
  virtual task wait_for_req(uvm_sequence_base seq, output trans req);
    wait_for_grant(seq);
    get_port.get(req);
  endtask
enc
```

Get request from driver

```
class reactive_driver extends uvm_driver#(trans);
  trans m_tr;
  virtual task get(output trans tr);
    wait(m_tr != null); tr = m_tr;  m_tr = null;
  endtask
  virtual task run_phase(uvm_phase phase);
    forever begin
      get_tr(m_tr);
      // ... get and send response from sequence
    end
  endtask
endclass
```

Get DUT output



Reactive Sequence Response

```
class reactive_sequence extends uvm_sequence #(trans);
  `uvm_declare_p_sequencer(reactive_sequencer)
  virtual task body();
    forever begin
      p_sequencer.wait_for_req(this, req);
      // generate response to req
      p_sequencer.send_rsp(this, rsp);
    end
  endtask
endcl
```

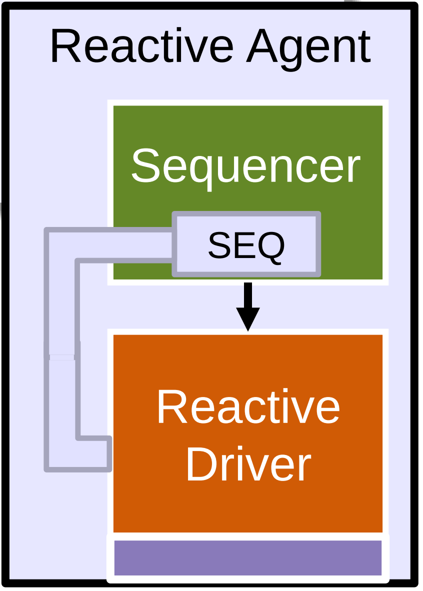
Give response to driver through sequencer

```
class reactive_sequencer extends uvm_sequencer #(trans);
   virtual task send_rsp(uvm_sequence_base seq, trans rsp);
    rsp.set_item_context(seq);
    seq.finish_item(rsp);
  endtask
endclass
```

Give response to driver

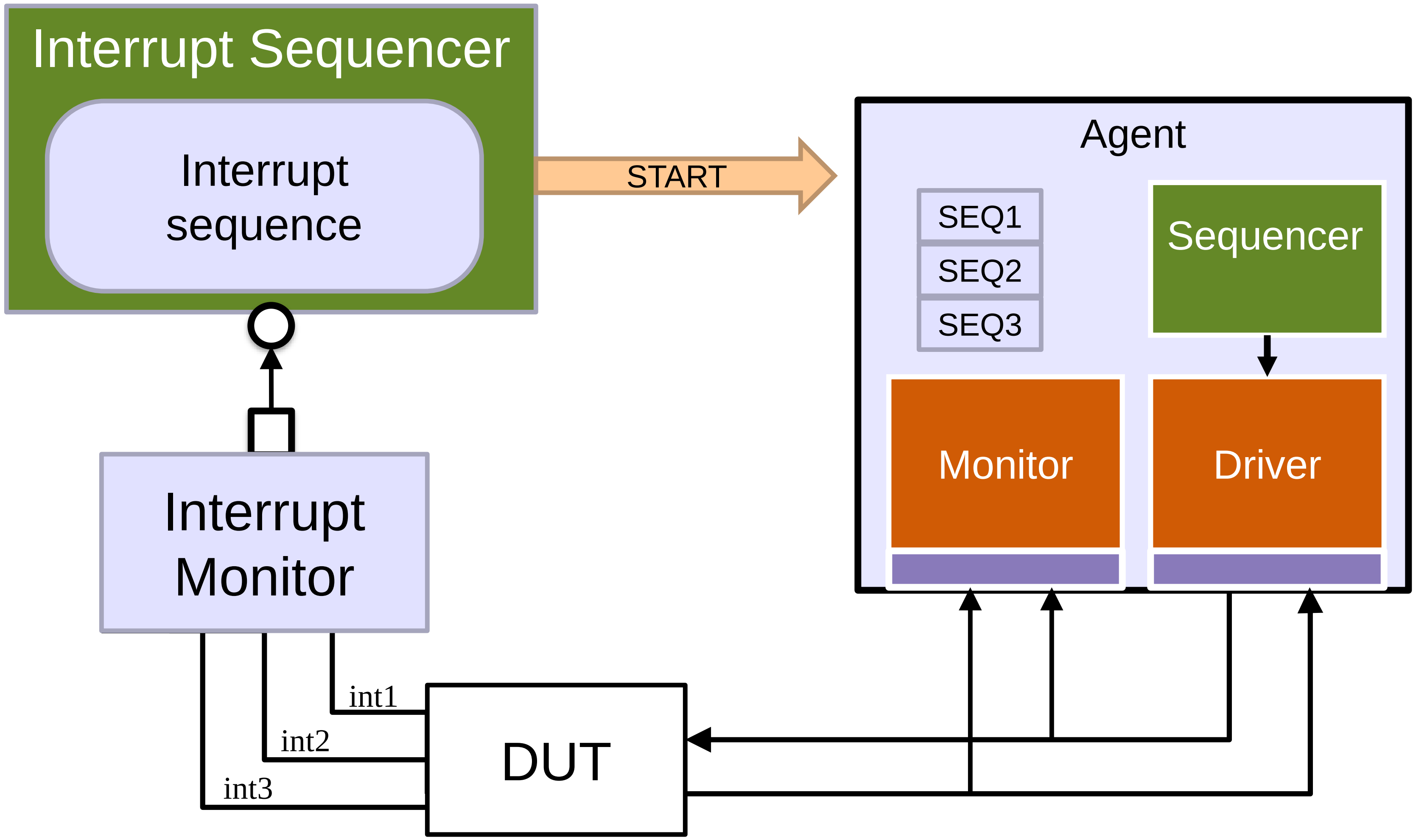
```
class reactive_driver extends uvm_driver#(trans);
  forever begin
      get_tr(); // get DUT output
      seq_item_port.get_next_item(rsp);
      drive_rsp(rsp);
      seq_item_port.item_done();
    end
endclass
```

Get response from sequence



Interrupt Sequence

Interrupt Sequences



Detect Interrupt

```
task interrupt_monitor::tx_monitor();   // simplified code
  interrupt_trans tr = interrupt_trans::type_id::create("tr", this);
  bit [3:0] interrupts = vif.monClk.interrupts;
  @(vif.monCLk iff (vif.monClk.interrupts != interrupts));
  tr.interrupts = vif.mon.interrupts;
  analysis_port.write(tr);
endtask
```

Monitor interrupts and send out via analysis port

```
class interrupt_sequencer extends uvm_sequencer;
 my_sequencer sqr;
 interrupt_trans tr; event interrupt_event;

 uvm_analysis_imp#(interrupt_trans,interrupt_sequencer) analysis_export;
 virtual function void write(interrupt_trans tr);
  if(tr.interrupts) begin // at least one interrupt set
   this.tr = tr;
    ->interrupt_event;
  end
 endfunction
en
```

Needs sequencer to pass interrupt responses to driver

Connects to the interrupt_monitor's analysis port

```
function void environment::connect_phase(uvm_phase phase);
  super.connect_phase(phase);
  intr_sqr.sqr = agt.sqr;
  intr_mon.analysis_port.connect(intr_sqr.anaylsis_export);
endfunction
```

React to Interrupt

```
class interrupt_sequence extends uvm_sequence; // other code left off
  intr_sequence_1 seq1;
  intr_sequence_2 seq2;
  intr_sequence_3 seq3;
  virtual task body();
    p_sequencer.set_arbitration(SEQ_ARB_STRICT_FIFO);
    forever begin
      @(p_sequencer.interrupt_event);
      case(p_sequencer.tr.interrupts)
        4'b0001: `uvm_do_on_pri(seq1, p_sequencer.sqr, 500);
        4'b0010: `uvm_do_on(seq2, p_sequencer.sqr);
        4'b0100: `uvm_do_on(seq3, p_sequencer.sqr);
      endcase
    end
  endtask
endclass
```

Set arbitration

Wait for the interrupt event from the sequencer

Priority

```
function void intr_test::build_phase(uvm_phase phase);
  super.build_phase(phase);
  uvm_config_db#(uvm_object_wrapper)::set(this,"env.intr_sqr.run_phase",
                "default_sequence", interrupt_sequence::get_type());
endfunction
```

Sequence Library

Sequence Library

- **When multiple sequences need to be executed by a sequencer, how does one manage these sequences?**
  - e.g. Execute a directed sequence before other randomly chosen sequences
- **Solution: Sequence Library**
  - Helps to organize and manage the execution of multiple sequences
  - By default executes sequences in the library randomly
  - Allows user to implement user algorithm of picking which sequence out of the sequence library to execute

Building a Sequence Library Package (1/2)

To enhance reuseability, encapsulate sequence library classes in a package

```
package packet_seq_lib_pkg;
import uvm_pkg::*;
class packet extends uvm_sequence_item;
  ...
endclass
class packet_seq_lib_base extends uvm_sequence_library #(packet);
  `uvm_object_utils(packet_seq_lib_base)
  `uvm_sequence_library_utils(packet_seq_lib_base)
  function new(string name = "packet_seq_lib");
    super.new(name);
    init_sequence_library();
  endfunction
endclass
// continued on next slide
```

Similar to the idea behind test_base, create a base library class without any sequences.

Macro builds the infrastructure of the sequence library

Method populates library with registered sequences

Building a Sequence Library Package (2/2)

- **Add common sequences to package**
  - Leave sequence library base empty

```
// continued from previous slide
class packet_seq_base extends uvm_sequence #(packet);  // macro not shown
  function new(string name = "packet_seq_base"); super.new(name);
    `ifdef UVM_POST_VERSION_1_1
    set_automatic_phase_objection(1);
    `endif
  endfunction
  `ifdef UVM_VERSION_1_1
  task pre_start();
    if ((get_parent_sequence() == null) && (starting_phase != null))
      starting_phase.raise_objection(this);
  endtask
  task post_start();
    if ((get_parent_sequence() == null) && (starting_phase != null))
      starting_phase.drop_objection(this);
  endtask
  `endif
endclass
endpackage
```

Reference Sequence Library in Environment

- **In the environment, make sequence library the default sequence for the sequencer**

```
class router_env extends uvm_env;
  router_agent agt;
  // utils macro and constructor not shown

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    agt = router_agent::type_id::create("agt", this);
    uvm_config_db#(uvm_object_wrapper)::set(this,"agt.sqr.main_phase",
                              "default_sequence", packet_seq_lib_base::get_type());
  endfunction
endclass
```

If the sequence library does not contain any sequence, nothing happens.
If the sequence library is populated with sequences, by default, 10 sequences will be randomly picked out of the library and executed.

Register and Execute Sequences

- **Import library package in program block**
- **Register sequence(s) with sequence library in test**
  - Use **`add_typewide_sequence()`** to register sequence in **<u>all</u>** instances of sequence library in the test

```
program automatic test;
import uvm_pkg::*; import packet_seq_lib_pkg::*;
// include other classes
class test_scenario extends test_base;
 //  component_utils and constructor not shown
 virtual function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  packet_seq_lib_base::add_typewide_sequence(scenario_seq::get_type());
  uvm_config_db#(int)::set(this, "*.sqr", "item_count", 20);
 endfunction
endclass
...
endprogram
```

Compile with vcs: (using UVM in VCS installation)

```
vcs -sverilog -ntb_opts uvm-1.2 packet_seq_lib_pkg.sv test.sv
```

Simulate with:

```
simv +UVM_TESTNAME=test_scenario
```

Resource Pool