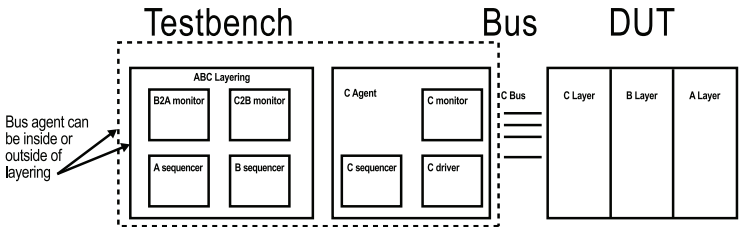
Layering in UVM

(Extracted from the UVM/OVM Online Methodology Cookbook found on verificationacademy.com)

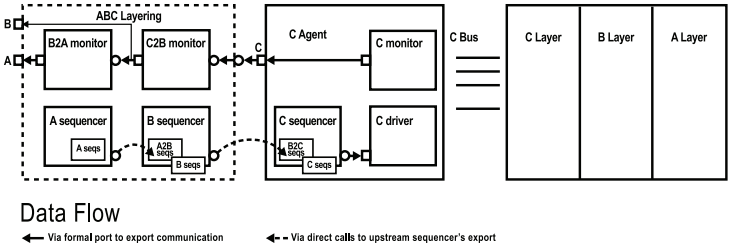
Many protocols have a hierarchical definition. For example, PCI express, USB 3.0, and MIPI LLI all have a Transaction Layer, a Transport Layer, and a Physical Layer. Sometimes we want to create a protocol independent layer on top of a standard protocol so that we can create protocol independent verification components (for example TLM 2.0 GP over AMBA AHB). All these cases require that we deconstruct sequence items of the higher level protocol into sequence items of the lower level protocol in order to stimulate the bus and that we reconstruct higher level sequence items from the lower level sequence items in the analysis datapath.



THE ARCHITECTURE OF A LAYERING

In order to do this we construct a layering component. A layering component:

- Must include a **child sequencer** for each non-leaf level in the layering.
- Must create, connect and start **translator sequence** for each non leaf level in the layering.
- Must have a handle to the leaf **level protocol agent**. This protocol agent may be a **child** of the layering or **external** to it.
- May include a **reconstruction monitor** for each non leaf level in the layering.
- Should create and connect external **analysis ports** for each monitor contained within the layering
- Will usually have a configuration object associated with it that contains the configuration objects of all the components contained within it.



CHILD SEQUENCERS

A child sequencer in a layering is simply the usual sequencer for that protocol. Very often an appropriately paramterized uvm_sequencer is quite sufficient. If the higher level protocol has been modeled as a **protocol UVC**, then the layering should instantiate an instance of the sequencer used by the agent for that protocol so that sequences can be targeted either at the bus agent or the layering.

For example, the ABC layering below has an A_ sequencer and a B_sequencer.

```
class ABC_layering extends uvm_subscriber #( C_item );
  `uvm_component_utils( ABC_layering )

  ...
  A_sequencer a_sequencer;
  B_sequencer b_sequencer;
  ...
  C_agent c_agent;

  function new(string name, uvm_component
parent=null);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    a_sequencer = A_sequencer::type_id::create
                  ("a_sequencer",this);
    b_sequencer = B_sequencer::type_id::create
                  ("b_sequencer",this);
  endfunction
  ...
endclass
```

TRANSLATOR SEQUENCES

A sequence which translates from upstream items to downstream items runs on the downstream sequencer but has a reference to the upstream sequencer. It directly references the upstream sequencer to call get_next_item and item_done to get upstream items and tell the upstream sequencer that it has finished processing the upstream sequence item. Between get_next_item and item_done it sends data to and gets data from the lower level sequencer by calling start_item and finish_item. A simple BtoC translator sequence is shown below:

```
class BtoC_seq extends uvm_sequence #(C_item);
  `uvm_object_utils(BtoC_seq);

  function new(string name="");
    super.new(name);
  endfunction

  uvm_sequencer #(B_item) up_sequencer;

  virtual task body();
    B_item b;
    C_item c;
    int i;
    forever begin
      up_sequencer.get_next_item(b);
      foreach (b.fb[i]) begin
        c = C_item::type_id::create();

        start_item(c);
        c.fc = b.fb[i];
        finish_item(c);
      end
      up_sequencer.item_done();
    end
  endtask
endclass
```

The run phase is responsible for creating the translator sequences, connecting them to their upstream sequencers, and starting them on their downstream sequencers:

```
virtual task run_phase(uvm_phase phase);
  AtoB_seq a2b_seq;
  BtoC_seq b2c_seq;

  a2b_seq = AtoB_seq::type_id::create("a2b_seq");
  b2c_seq = BtoC_seq::type_id::create("b2c_seq");

  // connect translation sequences to their respective
    upstream sequencers
  a2b_seq.up_sequencer = a_sequencer;
  b2c_seq.up_sequencer = b_sequencer;

  // start the translation sequences
  fork
    a2b_seq.start(b_sequencer);
    b2c_seq.start(c_agent.c_sequencer);
  join_none
endtask
```

THE PROTOCOL AGENT

Every layering must have a handle to the leaf level protocol agent. If we are delivering verification IP for a layered protocol, it usually makes sense to deliver the layering with an internal protocol agent. On the other hand, we may be adding a layering for use with a shrink wrapped protocol agent instantiated elsewhere in the testbench. Under these circumstances we will want the leaf level protocol agent to be outside the layering.

Internal Protocol Agent

In the case of an internal protocol agent, the layering component inherits from uvm_component and creates a child layering agent:

```
class ABC_layering extends uvm_component;
  `uvm_component_utils( ABC_layering )

  ...
  A_sequencer a_sequencer;
  B_sequencer b_sequencer;
  ...
  C_agent c_agent;

  function new(string name, uvm_component
parent=null);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    a_sequencer = A_sequencer::type_id::create
                  ("a_sequencer",this);
    b_sequencer = B_sequencer::type_id::create
                  ("b_sequencer",this);
    c_agent = C_agent::type_id::create("c_
sequencer",this);
  endfunction
  ...
endclass
```

External Protocol Agent

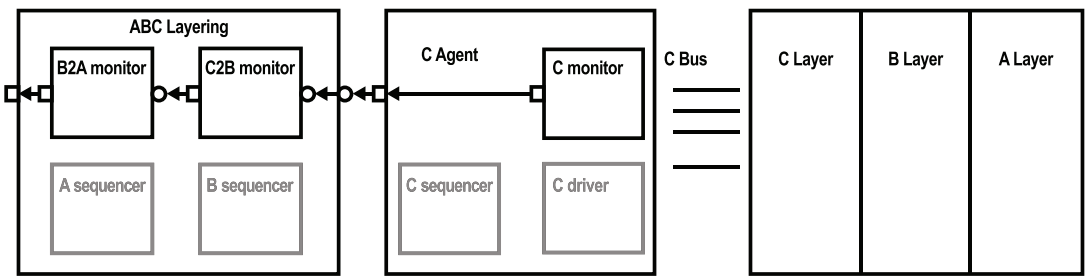
In the case of an external protocol agent, the layering is a subscriber parameterized on the leaf level sequence item and the agent is not constructed inside the layering. The code introduced **above** shows the code for an external agent.

Really, there is nothing special in the analysis path of a layering. For each layer in the monitoring we provide a reconstruction monitor which assembles high level items from low level items. These reconstruction monitors have an analysis export which is connected to the analysis ports of the lower level monitor and an analysis port. This analysis port is connected to an external analysis port and the analysis export of the upstream monitor if there is one.

An outline of a reconstruction monitor is shown below:

```
class C2B_monitor extends uvm_subscriber #(C_item);
// provides an analysis export of type C_item
  `uvm_component_utils(C2B_monitor)

  uvm_analysis_port#(B_item) ap;
  // declarations omitted ...

  function new(string name, uvm_component parent);
    super.new(name, parent);
    ap = new("ap",this);
  endfunction

  function void write(C_item t);
    // reconstruction code omitted ...
    ap.write( b_out );
    ...
  endfunction
endclass
```

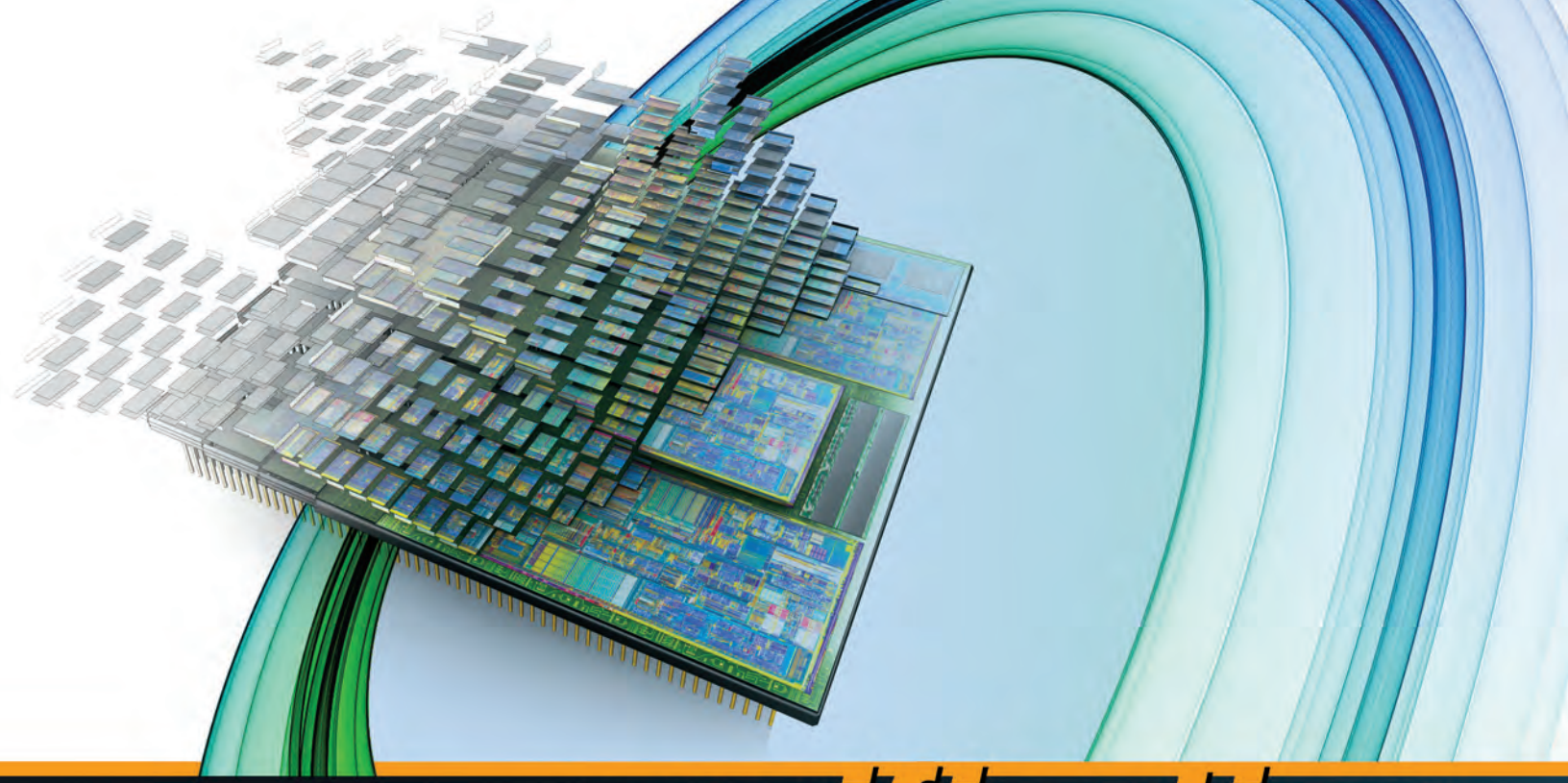
THE ANALYSIS PATH



The reconstruction monitors are connected up in the normal way:

```
class ABC_layering extends uvm_subscriber #
        ( C_item );
  `uvm_component_utils( ABC_layering )

  uvm_analysis_port #( A_item ) ap;

  A_sequencer a_sequencer;
  B_sequencer b_sequencer;

  C2B_monitor c2b_mon;
  B2A_monitor b2a_mon;

  C_agent c_agent;

  function new(string name, uvm_component
         parent=null);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    a_sequencer = A_sequencer::type_id::create
                  ("a_sequencer",this);
    b_sequencer = B_sequencer::type_id::create
                  ("b_sequencer",this);

    c2b_mon = C2B_monitor::type_id::create
                ("c2b_mon",this);
    b2a_mon = B2A_monitor::type_id::create
                ("b2a_mon",this);

    ap = new("ap" , this );
  endfunction

  function void connect_phase(uvm_phase phase);
    c2b_mon.ap.connect(b2a_mon.analysis_export);
    b2a_mon.ap.connect( ap );
  endfunction
  ...
endclass
```

For more details or to download the examples, go to **http://verificationacademy.com/uvm-ovm/Sequences/Layering**



verification HORIZONS

Editor: Tom Fitzpatrick
Program Manager: Rebecca Granquist

Wilsonville Worldwide Headquarters
8005 SW Boeckman Rd.
Wilsonville, OR 97070-7777
Phone: 503-685-7000

To subscribe visit:
www.mentor.com/horizons

To view our blog visit:
VERIFICATIONHORIZONSBLOG.COM

Mentor Graphics®

www.mentor.com