A Simplified Approach Using UVM Sequence Items for Layering Protocol Verification

Christine Thomson, Haiqian Yu
Microsoft Corporation
Agenda

- Introduction
- Example Scenario
- Alternative Solutions
- Proposed Method
- Future Work
- Conclusion
Introduction:
What is a layered protocol?

• Composed of multiple layers
  – Each level is responsible for a subset of the overall functionality

• Typically used to define and move data
  – Upper level layers operate on meaningful fields
  – Lowest level responsible for interface level data transfer

• Commonly used for data transfer between IPs, chips, etc

• Examples:
Introduction: Verifying a Layered Protocol

• Layered protocol approaches in UVM
  – Model the design’s structure
  – Checking can be done at one/any layer
  – Conversion needed between layers

• Method should be protocol independent

• What approach is the best approach?
  – Determine the scoping requirements
  – Determine the testbench goal

GOAL:
Verification Reuse!

Intuitive, flexible, controllable, visible!
Layered Protocol Example

Scenario: Valid/Ready Protocol

- Valid/Ready protocol implementation block diagram

Lowest layer which is responsible for following interface signaling rules to transmit/receive data

Highest layer which translates between raw data and the meaningful representation of the data
Layered Protocol Example
Scenario: Valid/Ready Protocol

- Packet Layer’s Data Fields
  - ID[7:0]
  - ADDR[15:0]
  - DATA[31:0]

- Valid/Ready Layer’s Data Fields
  - RAWDATA[55:0]

The RAWDATA bus width is set to the total width of the Packet layer’s fields. In this case: 8 + 16 + 32 = 56

<table>
<thead>
<tr>
<th>ID</th>
<th>ADDR</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bits</td>
<td>16 bits</td>
<td>32 bits</td>
</tr>
</tbody>
</table>

RAWDATA 56 bits
Layered Protocol Example
Scenario: Valid/Ready Protocol

Valid/Ready Protocol Timing Diagram

- CLK
- READY
- VALID
- RAWDATA

Transmitter Driven Signals
Receiver Driven Signals
Alternative Solutions: Layered Agents

- Testbench architecture models design architecture
  - An agent is created for each design layer
  - Packet agent translates between protocol layers
- Driver converts Packet seq items to Valid/Ready seq items
- Monitor converts Valid/Ready seq items to Packet seq items
Alternative Solutions: Layered Sequencers

- Testbench architecture models design architecture
- “Trimmed down” agents are used for the high level layers
- Packet agent translates between protocol layers
  - Sequencer converts Packet seq items to Valid/Ready seq items
  - Monitor converts Valid/Ready seq items to Packet seq items
Proposed Method: Layered Sequence Items

• Sequence items are used to model the design architecture
  – One sequence item per protocol layer
  – Functions perform the translation between layers

pack_data() is called to translate Packet data to Valid/Ready data

unpack_data() is called to translate Valid/Ready data to Packet data

• Functions perform the translation between layers
Proposed Method: Valid/Ready Seq. Item Code

class vld_rdy_seq_item #(int DATA_WIDTH=56) extends uvm_sequence_item;
  rand bit valid;
  rand bit ready;
  rand bit [DATA_WIDTH-1:0] raw_data;
...
  // Virtual function to implement translation from Packet -> Valid/Ready
  virtual function void pack_data();
  endfunction

  // Virtual function to implement translation from Valid/Ready -> Packet
  virtual function void unpack_data();
  endfunction
...

class pkt_seq_item extends vld_rdy_seq_item #(56);

    rand bit [7:0] id;
    rand bit [15:0] addr;
    rand bit [55:0] data;

    // Packet -> Valid/Ready translation
    virtual function void pack_data();
        raw_data = {id, addr, data};
    endfunction

    // Valid/Ready -> Packet translation
    virtual function void unpack_data();
        {id, addr, data} = raw_data;
    endfunction

    ...
class pkt_rand_seq extends uvm_sequence
    #((vld_rdy_seq_item #(56)));
...
pkt_seq_item pkt_item;
vld_rdy_seq_item #(56) req_item;
vld_rdy_seq_item #(56) resp_item;

task body();

// 1. Create a Packet sequence item and generate the data
pkt_item = pkt_seq_item::type_id::create("pkt_item");

if ( !( pkt_item.randomize() with {...} ) )
    `uvm_error("PKTSEQ", "pkt_item.randomize failed!")
...
// continued on next slide
Proposed Method: Example Packet Sequence

...  
// 2. Data conversion from Packet layer 
// to the Valid/Ready layer  
pkt_item.pack_data();  

// 3. Cast the pkt_seq_item to a 
// vld_rdy_seq_item  
if ( !$cast(req_item, pkt_item.clone()) )  
   `uvm_error("PKTSEQ","Cast failed!");  

// 4. Resume standard sequence flow  
start_item(req_item);  
finish_item(req_item);  
get_response(resp_item);  
endtask: body  
...  
endclass: pkt_rand_seq  

Not necessary if pack_data() was added into the Packet seq. item's post_randomize() function
Proposed Method: Example Scoreboard

```c
forever begin
    // Monitor Valid/Ready for valid transaction's
    vld_rdy_fifo.get(vld_rdy_item);

    // Create new Packet sequence item
    pkt_item = pkt_seq_item::type_id::create("pkt_item");

    // Convert to a Valid/Ready sequence item to a Packet
    // sequence item to access meaningful fields
    pkt_item.raw_data = vld_rdy_item.raw_data;
    pkt_item.unpack_data();

    // Fields are now populated with correct data

    // Sample coverage for meaningful fields
    pkt_item.sample();
...
end //end forever loop for vld_rdy_fifo
```
Benefits of Layering Sequence Items

• Eliminates the need for unique agents per layer
  
  Reduces initial development time

  Simplifies the overall architecture

  Expedites bring-up

  Reduces the code base

  Reduces the likelihood of bugs

• Eliminates the need for wrapper UVCs, translation classes or complex layered sequences
Future Work

• Layered protocols with >2 layers
• Individual layer verification
Conclusion

• Easy to implement solution

  Save development time and expedite testbench bring up

  Saving increases for designs with multiple unique packet layers layered with a common base layer

• Proven in a real world design
  – Design contained multiple unique packet layers with a shared base layer

  Saved approximately 8 person-days development time

• Future work
Thank You