February 28 – March 1, 2012

Conscious of Streams
Managing Parallel Stimulus

by
Jeff Wilcox
Principal Consulting Engineer
Paradigm Works, Inc.
Overview

- What’s in a Stream?
- Why use Parallel Stimulus?
- Concerns for Testbench Architecture
- Solution Space
- At What Cost Flexibility?
- Summary
What’s in a Stream?

• Many to one Stimulus
• Autonomous Flows
• Multi-Channel Interface
• Segmenting Interface
  - Refresh Request
  - Status Polling Routine
What’s in a Stream?

- Constraints defining transaction specifics
- Unique transaction types
- TDM mechanisms
- *Could* be handled in single transaction
Why Use Parallel Stimulus?

- Constraint Simplification
  - Disable unneeded constraints
  - Set fields to constant values
  - Mix protocols without adding constraint complexity
  - Result: Better performance

- Stream Autonomy
  - Streams must not block each other
    - Per-channel flow control
  - Metered Delivery
    - Bandwidth provisioning
    - Guaranteed periodicity
Testbench Architecture Concerns

- Performance
  - Runtime image size
  - CPU time

- Scalability
  - 10 to 20 streams required today
  - 500 to 1000 streams next year?

- Test development
  - How easy to manage active streams?
  - How easy to configure specific streams?
Options for Parallel Streams

Parallel UVCs

Parallel Sequences
Protocol Isolation

- Separates high level protocol from interface protocol
- Reuse Protocol UVC on different interface
- Use different Protocols on same interface
Parallel UVCs

- One Protocol UVC provides stimulus for one stream
- Constraint simplification
- Multi-Protocol Support
- Maximum Flexibility
- Poor scalability

Jeff Wilcox, Paradigm Works, Inc.
Parallel Sequences

- One sequence provides stimulus for one stream
- Multi-Protocol support not inherent
- Reasonable flexibility
  - Some added complexity
  - How do we stop?
Managing Test Configuration

class multi_stream_sequence extends uvm_sequence;
...
virtual task body();
    for (int i = 0; i < max_streams; i++) run_stream(i);
    wait (terminal_condition == 1);
    for (int i = 0; i < max_streams; i++)
        p_sequencer.state_kind[i] = DISABLED;
endtask

task run_stream(int i);
    fork
        forever begin
            wait (p_sequencer.state_kind[i] == ENABLED);
            `uvm_do_on(seq, p_sequencer.some_sequencer)
        end
    join_none
endtask
endclass

1) Launch all streams
2) Meet test criteria
3) Disable all streams

• Stall if DISABLED.
• Otherwise issue sequence.
• Metering can be added at top or bottom of loop
At What Cost Flexibility

Image Size (10k Transactions)

- Sequences Scale Better
- 400 MB increase from 100 streams to 1000 streams
At What Cost Flexibility?

Image Size (100k Transactions)

<table>
<thead>
<tr>
<th>Streams</th>
<th>Sequences</th>
<th>UVCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>2500 MB</td>
<td>3000 MB</td>
</tr>
<tr>
<td>500</td>
<td>2700 MB</td>
<td>3000 MB</td>
</tr>
<tr>
<td>1000</td>
<td>2800 MB</td>
<td>3000 MB</td>
</tr>
</tbody>
</table>

Still 400 MB

UVC penalty constant over test length
At What Cost Flexibility?

Performance (100k Transactions)

How long would you rather wait for results?

- 2 times Longer!
- 4 times Longer!

CPU seconds / sim microsecond

- Streams: 1000, 500, 100
- UVCs
- Sequences

Jeff Wilcox, Paradigm Works, Inc.
Going Forward

• Are results consistent across platforms?
• Are results consistent across UVCs?
• API for managing multi-stream configuration
  - Increase sequence reuse
  - Simplify test writing
• API for ending active stimulus phase
  - Total sequences?
  - Sequences issued per stream?
• Handling non-native transactions in sequencer
  - Eliminates need for parallel UVC approach
Summary

- Managing streams as isolated cases is easier
- Managing sequences more natural and simpler than managing UVCs
- Parallel sequences more scalable
- Parallel sequences more efficient