February 28 – March 1, 2012
There’s something wrong between Sally Sequencer and Dirk Driver
(Why UVM sequencers and drivers need some relationship counseling)
by
Mark Peryer
Verification Methodologist
Mentor Graphics
Overview

• The UVM Stimulus generation architecture
  – Sequencer, Driver, Sequences, Sequence Items
  – Dates from the eRM, OVM, now UVM

• Is it still fit for purpose?
• Is it time for an update?

• As an alternative
  – Would TLM2 be a better starting point?
How The UVM Is Positioned

- Consistent API enables reusability
  - Interoperability between components

- Test cases written by engineers with design domain knowledge
  - Rather than detailed testbench (UVM) knowledge
  - Working at a higher level of abstraction (TLM)

Transactions
(High level
Transfer description)
Where The Cracks Appear

• Writing sequences
  – API is confusing with too many choices
  – Implementation has to match the driver

• Methodology is about “freedom from choice”
  – Abstraction can be powerful
    • But not if it’s complicated
Stimulus Generation In The UVM

- **Sequence_items** (aka transactions)
  - Generated by sequences
- **Sequencer**
  - Arbitrates between multiple sequencers
  - Implements TLM 1 port proxy for sequences connecting to drivers

Transactions
(High level Transfer description)
TLM 1 API - Producer, Consumer

- Simple API
  - transport(), put(), get()
  - Unidirectional flow
  - Point to point connection

```
transport(req, rsp)
```

```
put(req)
get(rsp)
```

```
get(req)
put(rsp)
```

Producer
Consumer
Bidirectional Transfer

sequence sequencer driver

start_item(req) get_next_item(req) item_done()
finish_item(req)

sequence sequencer driver

start_item(req) get(req) finish_item(req) get_response(rsp) put(rsp)

item_done() API implementation
put() API implementation
Observations

• There are at least two implementation models

• Departure from TLM principles
  – The sequence writer has to understand something about the driver implementation
Fully Pipelined Transfers

Separate stimulus and response threads

Thread for each pipeline stage
Handling Disruptive Events

- Disruptive Events:
  - Hard or soft resets
  - Errors
    - Deliberately injected
    - DUT error
  - UVM Phase change

- Very easy to deadlock
- Up-front thought required

Sequence waits for response

Driver returns response
But sequencer has deleted sequence
Alternative Using TLM 2

- TLM 2 Initiator and Target Sockets support both:
  - Blocking transports
    - Single method, returns when response ready
    - Equivalent to `item_done()`
  - Non-blocking transports
    - Initiator calls `nb_transport_fw()` method
    - Target calls `nb_transport_bw()` method independently
    - Phase and status information passed together with data
Non-Blocking Transport Implementation

- Target can call `nb_transport_bw()` any number of times
- Transaction always passed with status and phase
- Allows state tracking on either side
Alternative Sequence Driver API

Sequence API:
- transport()
- handle_response()
- exception()

Well defined and simple API

Protocol specific initiator and target state machines handle protocol complexity
Sequence - Driver API

- transport()
  - Can be blocking or non-blocking
    - Depending on the protocol FSM implementation
  - Response may or may not be valid on completion
- handle_response()
  - Call back to process pipelined or out of order responses
- exception()
  - Call back to handle disruptive events
Envisaged Implementation

Sequence API implemented around initiator socket

Sequencer creates target sockets on fly, hooks them into n to 1 switch. Has initiator socket bound to driver target socket with protocol policy object.

Driver has target socket and implements driver side part of protocol FSM
UVM Implementation Issues

- Current (UVM 1.1a) TLM 2 implementation Issues
  - Sockets are components and can only be constructed during the build_phase
    - They don’t need to be
  - Separate blocking and non-blocking sockets
    - This is not compliant to TLM 2
  - Socket _bw transport method registration is per parent rather than per socket
    - This can be worked round, but is awkward
Conclusions

• UVM Sequence – Driver API
  – Inconsistent and difficult to understand
  – Struggles at the extremes of protocol behaviour

• TLM2 based alternative
  – Consistency and ease of use
  – Proven state model for VIP side to handle complex protocols
  – Currently stymied by UVM implementation issues

• Whatever happens
  – The eRM/OVM legacy will be with us for some time
  – There is work to be done to implement the TLM 2 solution

• The TLM2 solution could solve today’s and tomorrow’s problems
  – Particularly cross platform/engine communication
February 28 – March 1, 2012

Thank You!