Another Take on Writing Reusable Testbench Code

A Generic UVM Agent with Fine-grain Command-line Configuration

Nikhil Kikkeri and Daniel Wei

Oracle Inc.

March 30, 2016

SNUG Silicon Valley

Agenda

Problem Description

Developing the Generic UVM Agent

Example Protocols using the Generic UVM Agent

Summary

Q & A

ORACLE

snug
Synopsys Users Group
Silicon Valley

Problem Description

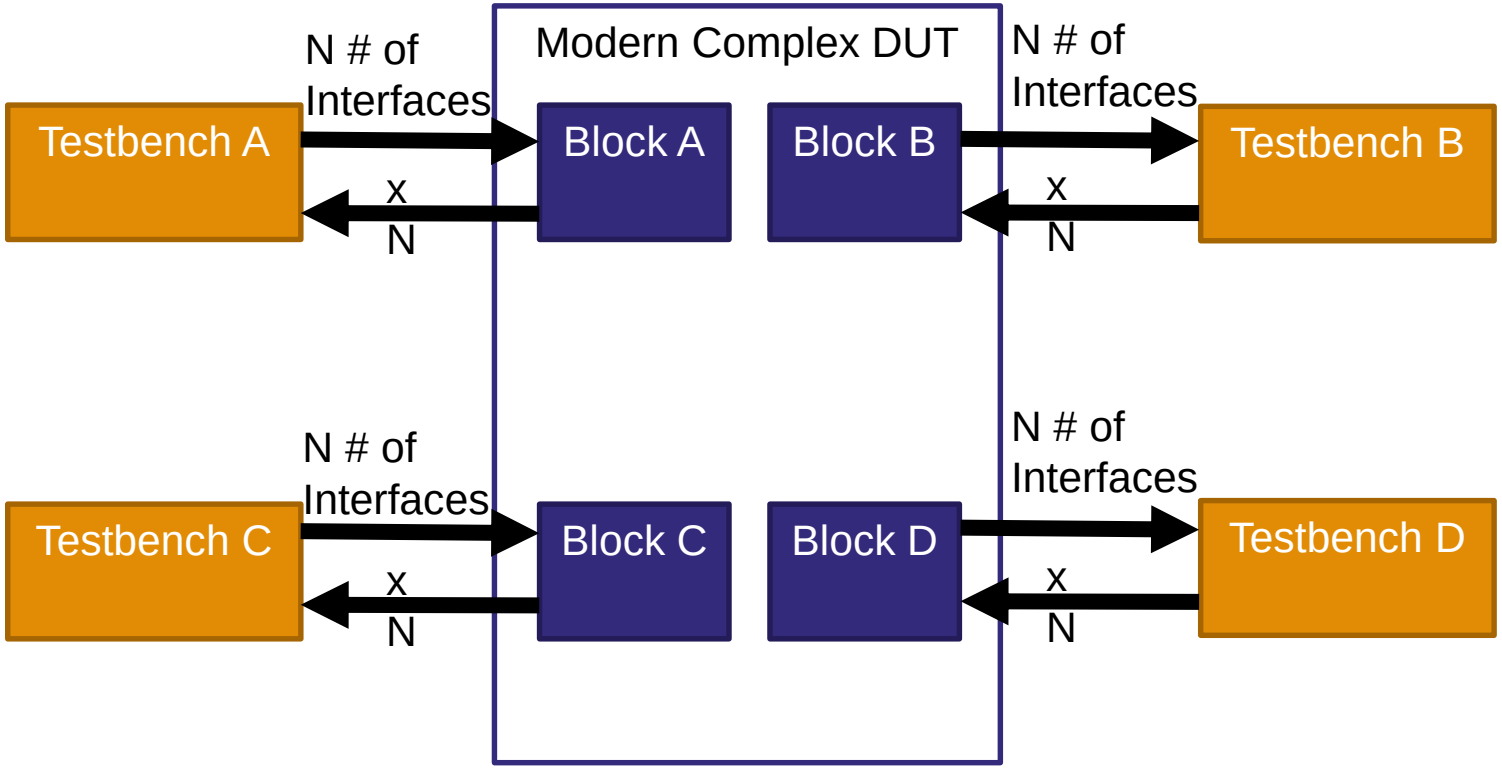
Challenges of DV Engineering

ORACLE

snug
Synopsys Users Group
Silicon Valley

- Tight schedule, late features, and changing specifications
- Explore various technologies (emulation, FPGA, formal, etc.)
  - Running simulations still main method of verification
- Explore methods of minimizing effort to bring up a testbench
  - Use standard base-class libraries, e.g., UVM
  - Devise/streamline project methodology

Modern Devices-Under-Test



Usual Verification Approach

ORACLE

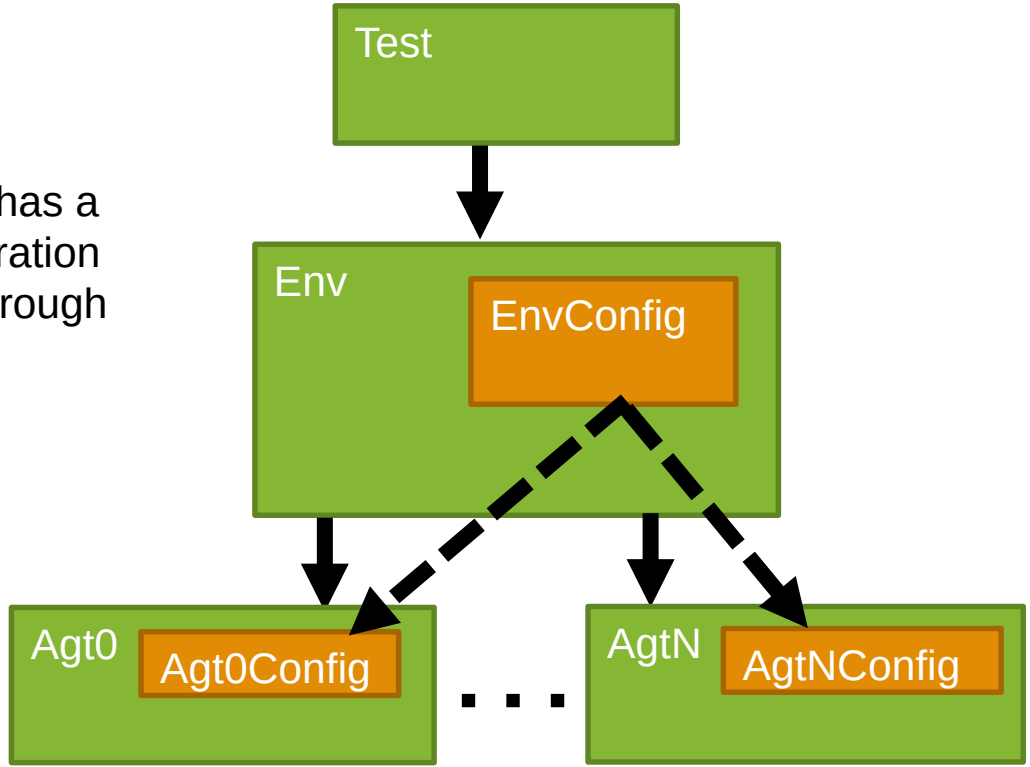
snug
Synopsys Users Group
Silicon Valley

- For every unit-level testbench in modern DUT a DV engineer must:
  - Write agents (driver, monitor, sequencer), environment
  - Virtual sequences, virtual sequencers
  - Hookup of testbench with design
- Consequently every testbench requires:
  - Different monitors and drivers
  - Different sequence classes
  - Separate hookup between testbench and DUT

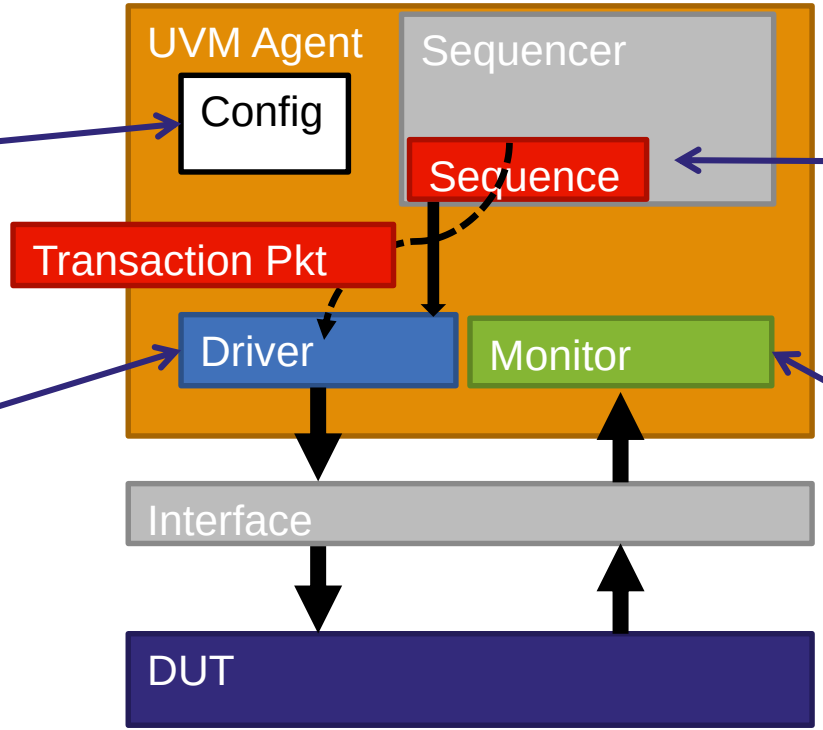
Typical UVM Bench Structure

Run-time configuration

each uvm component has a corresponding configuration object passed down through uvm_config_db



Generic UVM Agent



- Run-time configuration
- Virtual interface handle

Generate transaction packets stimulus

Converts transaction packets to signals

Converts signals to transaction packets. Send to rest of TB

Areas for Improvement

Observations	Objectives
Monitors and Drivers connect TB and DUT	- Need not be aware of underlying protocol - Reusable if written in a protocol-agnostic fashion
TB only needs to understand DUT from a transactional level	- Signals organized and hooked up in a way to facilitate generic drive/sample - Focus more time on writing sequences and closing functional coverage

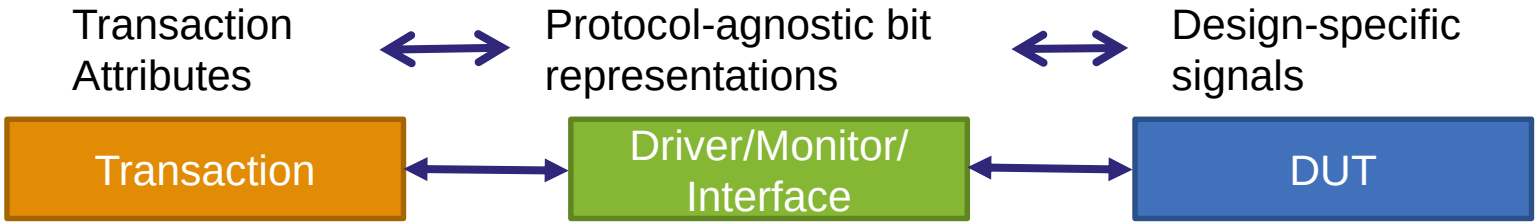
ORACLE

snug
Synopsys Users Group
Silicon Valley

Developing the Generic UVM Agent

Objectives

- Write a reusable UVM Agent
  - Protocol-agnostic Driver and Monitor
  - Generic virtual interface between specific DUT signals and protocol agnostic representations
- Provide transactional hooks to provide a bridge between specific attributes of a transaction and protocol-agnostic vector representations



Hookup DUT and Bench

Categorize DUT Signals Into "Data" and "Control"

DUT:

```
module memory_controller (
  input clk,
  input rdWr,
  input addrDataPhase,
  input [63:0] dataIn,
  input advalid,
);
endmodule
```

Interface:

```
interface PortIntf
#(PACKET_WIDTH=144,CONTROL_WIDTH=4)
(
  input bit clk,
  inout wire [PACKET_WIDTH-1:0] data,
  inout wire [CONTROL_WIDTH-1:0] ctrl
);
...
endinterface: PortIntf
```

```
PortIntf #(65,1)
  reqIntf (.clk (clk), .ctrl (mcu.advalid),
           .data({mcu.rdWr,
                  mcu.addrDataPhase,
                  mcu.dataIn}));
```

Port Interface

Use a BFM class – [David Rich, DVCon 2008]

```
interface PortIntf #(
  int DATA_WIDTH = `V_MAX_DATA_SIZE,
  int CTRL_WIDTH = `V_MAX_CTRL_SIZE)
  (input bit clk,
  inout wire [DATA_WIDTH-1:0] data,
  inout wire [CTRL_WIDTH-1:0] ctrl);

  //clocking blocks...

  class ActualIntfClass
     extends AbstractIntfClass;
    ...
  endclass: ActualIntfClass
  ActualIntfClass intfInst = new("intf");
endinterface: PortIntf
```

Abstract Interface Class

Virtual class defines calls to be used for all actual interface implementations

```
virtual class AbstractIntfClass
extends uvm_object;
  ...
  pure virtual task drive(
    logic [`V_MAX_DATA_SIZE-1:0] data,
    logic [`V_MAX_CTRL_SIZE-1:0] ctrl
  );
  pure virtual task monitor(
    ref logic [`V_MAX_DATA_SIZE-1:0] data,
    ref logic [`V_MAX_CTRL_SIZE-1:0] ctrl
  );
  pure virtual task wait_cycle(int cycles);
  pure virtual task driveIdle();
  pure virtual task driveX();
  ...
endclass: AbstractIntfClass
```

widest size to work for all interfaces

Driver and monitor do not need to be parameterized to different signal widths

Tasks to drive and sample values on DUT interface

Interface Implementation

Define Generic BFM Tasks

```
class ActualIntfClass extends AbstractIntfClass;
  task drive(  logic [`V_MAX_DATA_SIZE-1:0] data,
               logic [`V_MAX_CTRL_SIZE-1:0] ctrl);
    driverCB.ctrl <= ctrl;
    driverCB.data <= data;
    @(driverCB);
    driverCB.ctrl <= 0;
    driverCB.data <= 0;
  endtask

  task monitor( ref logic [`V_MAX_DATA_SIZE-1:0] data,
                ref logic [`V_MAX_CTRL_SIZE-1:0] ctrl);
     wait(|(monitorCB.ctrl) == 1'b1);
     data = monitorCB.data;
     ctrl = monitorCB.ctrl;
  endtask
endclass: ActualIntfClass
```

Drive task drives data and ctrl through clocking block

Sampling task waits on an “or” of the ctrl signals

Transaction Base Class

```
virtual class PacketBase extends
                        uvm_sequence_item;
  ...
  pure virtual function void bitsPackData(
   ref logic[`V_MAX_DATA_SIZE-1:0] data);

  pure virtual function void bitsPackCtrl(
   ref logic[`V_MAX_CTRL_SIZE-1:0] ctrl);

  pure virtual function void bitsUnPackData(
   logic [`V_MAX_DATA_SIZE-1:0] data);

  pure virtual function void bitsUnPackCtrl(
   logic [`V_MAX_CTRL_SIZE-1:0] ctrl);
  ...
endclass: PacketBase
```

Pack and Unpack functions
- Must be implemented
- Used by driver and monitor

Transaction Packet divided into:
- ctrl – signals required for sampling
- data – rest of signals

An Actual Transaction

Map Packet Specific Attributes to Generic Vectors

```
class ReqPacket extends PacketBase;
  `uvm_object_utils(ReqPacket)
  bit valid;
  bit rdWr;
  bit[63:0] data;
  bit cmd;
  ...
  function void bitsPackData(ref logic[`V_MAX_DATA_SIZE-1:0] data);
    data = {rdWr, data};
  endfunction

  function void bitsPackCtrl(ref logic[`V_MAX_CTRL_SIZE-1:0] ctrl);
    ctrl = valid;
  endfunction
```

New packet definition extends the base class

Unpacked attributes are packed to data and control variables
Similar implementation for packed to unpacked

Driver

Reduce to Generic Single-Cycle Drive

```
class PortDriver #(type T=PacketBase) extends uvm_driver#(T);
  ...
  protected PortAgentConfigInfo Cfg;
  logic [`V_MAX_DATA_SIZE-1:0] data;
  logic [`V_MAX_CTRL_SIZE-1:0] ctrl;
  ...
  task run_phase(uvm_phase phase);
    T trans;
    ...
    seq_item_port.get_next_item(trans);
    trans.bitsPackData(data);
    trans.bitsPackCtrl(ctrl);
    Cfg.absIntf.wait_cycle(trans.PktDelay);
    Cfg.absIntf.drive(data, ctrl);
    ...
  endtask:run_phase
endclass: PortDriver
```

Driver packs data and ctrl signals

Driver calls the virtual interface drive task through handle

Monitor

Reduce to Generic Single-Cycle Sample

```
class PortMonitor #(type T=PacketBase) extends uvm_monitor;
  protected PortAgentConfigInfo Cfg;
  logic [`V_MAX_DATA_SIZE-1:0] data;
  logic [`V_MAX_CTRL_SIZE-1:0] ctrl;
  ...
  task run_phase(uvm_phase phase);
     T samplePkt;
     forever begin
       Cfg.absIntf.wait_cycle(1);
       Cfg.absIntf.monitor(data, ctrl);
       samplePkt = T::type_id::...;
       samplePkt.bitsUnPackCtrl(ctrl);
       samplePkt.bitsUnPackData(data);
       //Write to Analysis port
     end
  endtask
endclass: PortMonitor
```

Monitor calls the virtual interface monitor task

Monitor unpacks data and ctrl signals

Final Generic PortAgent

```
class PortAgent #(type T=PacketBase ) extends uvm_agent;
  PortAgentConfigInfo Cfg;
  `uvm_component_param_utils_begin(PortAgent#(T))
     `uvm_field_object(Cfg, UVM_DEFAULT)
  `uvm_component_utils_end

   PortDriver #(T) Driver;
   PortMonitor #(T) Monitor;
   uvm_sequencer #(T) Sequencer;
   ...
```

Parameterized to specific transaction type

Create Agent Instances

Pass Virtual Interfaces To Agent Configuration Objects

```
class Env extends uvm_env;
  `uvm_component_utils(Env)

  PortAgent #(ReqPacket)  reqAgt;
  PortAgent #(RespPacket) respAgt;
  virtual PortIntf #(65,1) input_if;
  virtual PortIntf #(64,1) output_if;

  function void build_phase(uvm_phase phase);

    Cfg.req.absIntf = input_if.getIntfInst();
    Cfg.resp.absIntf = output_if.getIntfInst();

  endfunction
endclass
```

Passing abstract interface object handle

Configuration objects are passed down from env to sub-blocks through uvm_config_db

Create Interfaces in TB Top

```
module tb_top;
  import uvm_pkg::*;
  memory_controller mcu(.clk(clk),
                        .rdWr(),
                        .advalid(),
                        .dvalid(),
                        .dataOut(),
                        .dataIn());

  PortIntf #(65,1) reqIntf (.clk (clk),
                            .ctrl (mcu.advalid),
                            .data ({mcu.rdWr,mcu.dataIn}));

  initial begin
    //add interfaces to uvm_config_db
  end
endmodule: tb_top
```

Bit-splicing is used to combine signals to ctrl and data groups

Interfaces pushed to uvm_config_db

ORACLE

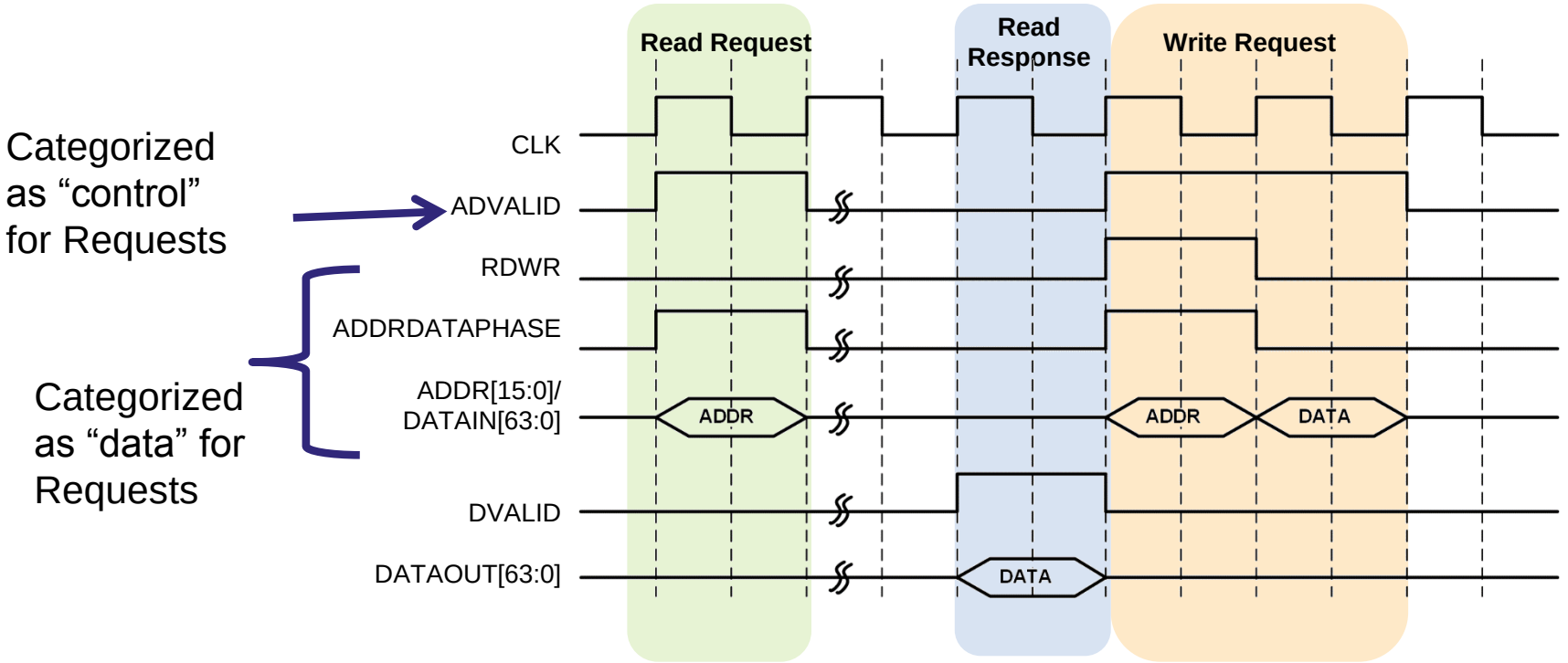
snug
Synopsys Users Group
Silicon Valley

Example Protocols using the Generic UVM Agent

Implementation Steps

- Instantiate
  - Generic port interfaces
    - Identify and categorize “data” and “control” signals and connect them as appropriate
  - Generic port agents parameterized to transaction type
- Pass abstract interface to agent in Env
- Create packet definitions
  - Implement pack/unpack functions
    - Identify transaction attributes and categorize them as “data” and “control”
- Write sequences

Simple Multi-Cycle Protocol



Create Request Packet 1/2

```
class ReqPacket extends PacketBase;
  bit advalid=1'b0;
  bit rdWr;
  bit addrDataPhase;//Address or Data Phase
  rand bit[15:0] addr;
  rand bit[63:0] wr_data;
  rand COMMAND_e cmd;

  function void bitsPackData(ref logic[`V_MAX_PACKET_SIZE-1:0] data);
    if(addrDataPhase)
      data = {rdWr,addrDataPhase,48'h0,addr};
    else
      data = {rdWr,addrDataPhase,this.wr_data};
  endfunction

  function void bitsPackCtrl(ref logic[`V_MAX_CTRL_SIZE-1:0] ctrl);
    ctrl = advalid;
  endfunction
```

control

data

Fill in pack and unpack functions

Create Request Packet 2/2

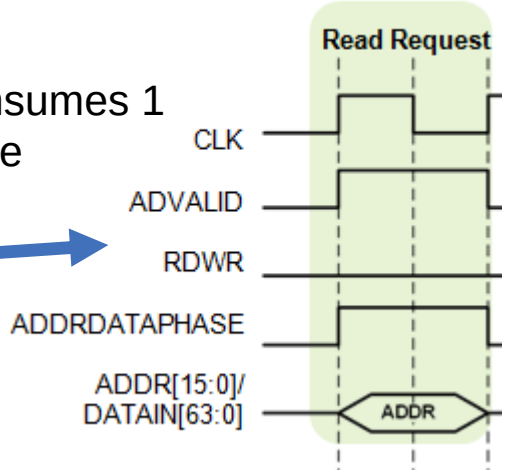
```
function void bitsUnPackData(logic [`V_MAX_PACKET_SIZE-1:0] data);
   rdWr = data[65];
   addrDataPhase = data[64];
    if( addrDataPhase == 1'b1)
      addr = data[15:0];
    else
      wr_data = data[63:0];
   if({rdWr,addrDataPhase} == 2'b01)
     cmd = READ;
   if({rdWr,addrDataPhase} == 2'b11)
     cmd = WRITE;
 endfunction

 function void bitsUnPackCtrl(logic [`V_MAX_CTRL_SIZE-1:0] ctrl);
    advalid = ctrl;
 endfunction
endclass
```

Read Sequence

```
class ReadSequence extends uvm_sequence #(ReqPacket);

  task body();
    `uvm_create(req);
    start_item(req);
    req.randomize();
    req.rdWr = 1'b0;
    req.advalid = 1'b1;
    req.addrDataPhase = 1'b1;
    req.cmd = READ;
    finish_item(req);
  endtask

endclass: ReadSequence
```

Also works with `uvm_do

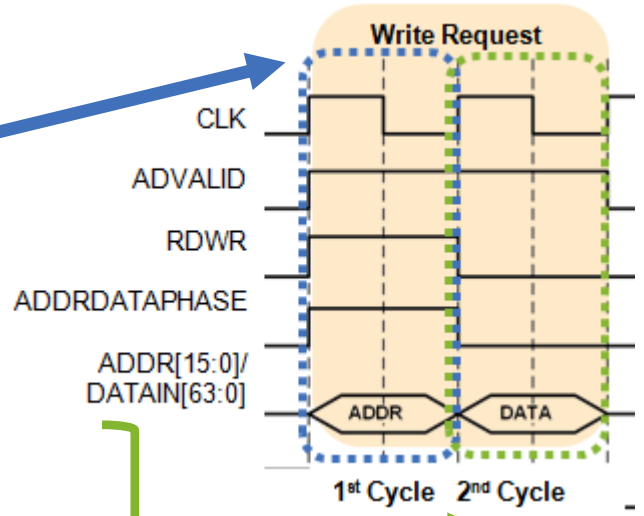
Consumes 1 cycle



Write Sequence

```
class WriteSequence extends uvm_sequence #(ReqPacket);
  task body();
    `uvm_create(req);
    req.randomize();
    start_item(req);
    req.advalid = 1'b1;
    req.cmd = WRITE;
    req.addrDataPhase = 1'b1;
    req.addr = 16'h1234;
    req.rdWr = 1'b1;
    finish_item(req);
    `uvm_create(req);
    start_item(req);
    req.advalid = 1'b1;
    req.addrDataPhase = 1'b0;
    req.wr_data = 64'h1234_5678_9abc_def0;
    finish_item(req);
  endtask
endclass: WriteSequence
```

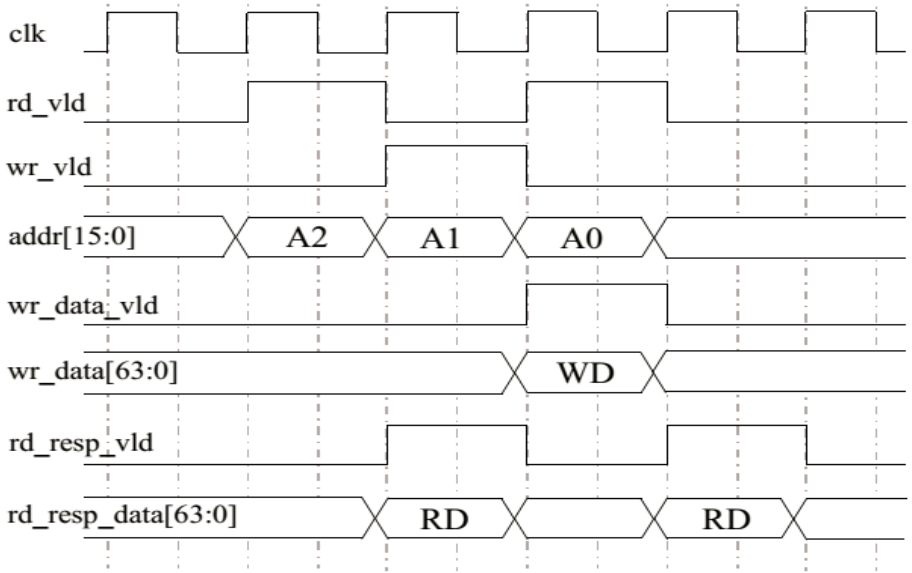
Also works with `uvm_do



Another Multi-cycle Protocol

Concurrent requests and write data cycles

- One cycle read request
  - Address with rd_vld bit held high
- One cycle read response
- Two cycle write request
  - Address with wr_vld high
  - Data sent 1 cycle later with data_vld
- Read and write requests can interleave



MemPacket Class 1/2

```
class MemPacket extends PacketBase;
  ...
  enum {IDLE,READ,WRITE} type_e;
  rand bit [63:0] wr_data;
  rand bit [15:0] address;
  bit wr_vld=1'b0;
  bit rd_vld=1'b0;
  bit wr_data_vld=1'b0;

  rand type_e trans_type;

  constraint c_wr_data_zero_for_reads {
    (trans_type == READ) -> (wr_data == 0);
  }

  constraint c_idle_packet {
    (trans_type == IDLE) -> ({wr_data,address} == 0);
  }
```

data

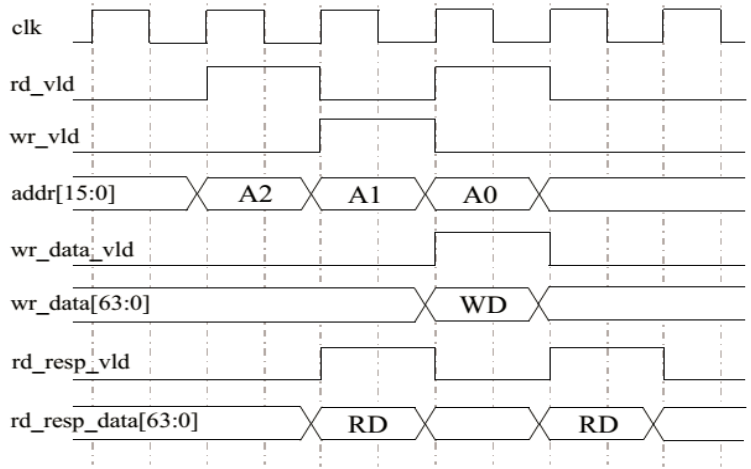
control

MemPacket Class 2/2

```
  function void bitsPackData(ref logic[`V_MAX_DATA_SIZE-1:0] data);
     data = {wr_data,address}
  endfunction: bitsPackData

  function void bitsPackControl(ref logic[`V_MAX_CONTROL_SIZE-1:0 ctrl);
     ctrl = {wr_data_vld,wr_vld,rd_vld};
  endfunction: bitsPackControl

  function void bitsUnPackData(logic [`V_MAX_PACKET_SIZE-1:0] data);
     {wr_data,address} = data;
  endfunction

  function void bitsUnPackCtrl(logic [`V_MAX_CTRL_SIZE-1:0] ctrl);
     {wr_data_vld,wr_vld,rd_vld} = ctrl;
  endfunction
endclass
```

Sequence Class 1/2

```
class Sequence extends uvm_sequence;
  MemPacket WriteDataQueue[$];
  MemPacket req;


  task body();
    do begin
      req.randomize(); end
      case(req.trans_type)
        MemPacket::READ: req.rd_vld = 1'b1
        MemPacket::WRITE: begin
          MemPacket cpy = MemPacket::type_id::create("copy");
          cpy.copy(req); //deep copy
          WriteDataQueue.push_back(cpy);
          req.wr_vld = 1'b1;
        end
      endcase
```



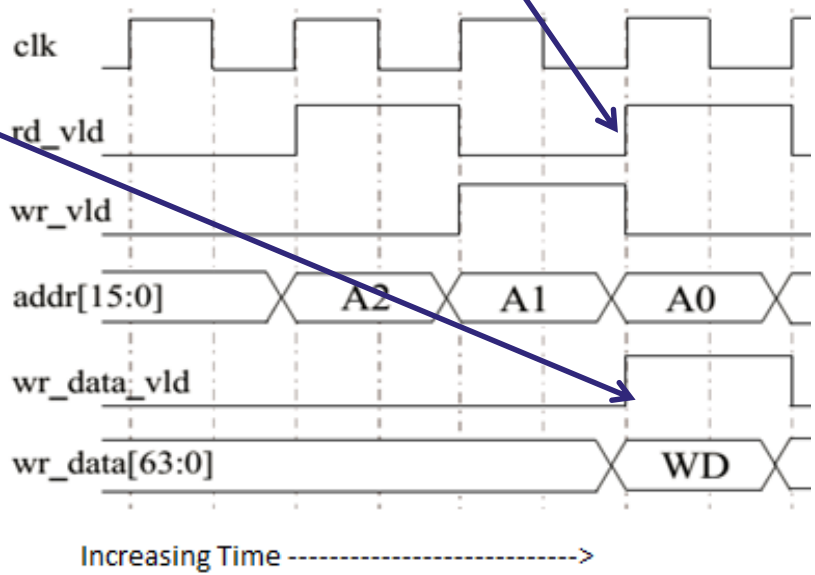
1 cycle read

1st cycle of write

Set the packet attributes based on request type

Sequence Class 2/2

```
 if(((req.trans_type != MemPacket::WRITE) &&  WriteDataQueue.size) ||
     ((req.trans_type == MemPacket::WRITE) && (WriteDataQueue.size > 1)))
   begin
     req.wr_data = WriteDataQueue[0].wr_data;
     req.wr_data_vld = 1'b1;
     WriteDataQueue.pop_front;
   end
   finish_item(req);
end while (i++ < 100 ||
           WriteDataQueue.size);

endtask: body
endclass: Sequence
```



ORACLE

snug
Synopsys Users Group
Silicon Valley

Summary

Conclusion

- Generic UVM Agent fits wide variety of use cases
  - Layered sequences featuring elaborate high-level transactions.
    - Single-cycle protocols are trivial to use with Generic Agent
    - Multi-cycle protocols are handled at sequence level
  - Currently used for 9 protocols and 13 interfaces on a single project
- Leverages an abstract BFM and a parameterized interface
- Difference only in packet definition and sequence writing
- Able to quickly connect testbench to Modern DUT
  - Focus on sequences
- Generic UVM Agent is extendable

More In The Paper

- Fine-grain Command-line configuration
  - Enhanced Wrapper class using UVM command-line processor and UVM string matching functions
  - Ability to quickly set several knobs
  - Ability to set a knob on per-instance level

Acknowledgements

- Our co-authors
  - Anirban Bhattacharjee, Krishna S. Gudlavaletti, Hui Shi and Sandra Shih
- Our SNUG 2016 reviewers
  - John Dickol, Jean Fong, Benjamin Ting, Sumit Vishwakarma
- Our managers, and legal team
  - Suzie Padwal, Madhumita Bhattacharya, Lata Jindal, Pamela Parrish, Janaki Seetharaman, Gary Peterson, Rodrigo Liang
  - Charles T. Cheng, Johanna Sistek, Rick Weber
- Our colleagues
- Our families

Copyright Notice

- All derivative work in the Presentation and Paper is protected by Copyright.

ORACLE

snug
Synopsys Users Group
Silicon Valley

Thank You

ORACLE

snug
Synopsys Users Group
Silicon Valley

Q & A