Use the Sequence, Luke

Guidelines to Reach the Full Potential of UVM Sequences

Jeff Vance, Jeff Montesano, Mark Litterick
Verilab

October 23, 2018
Austin

Agenda

Introduction

Sequence API Strategy

Sequence Guidelines

Conclusion

Introduction

Introduction

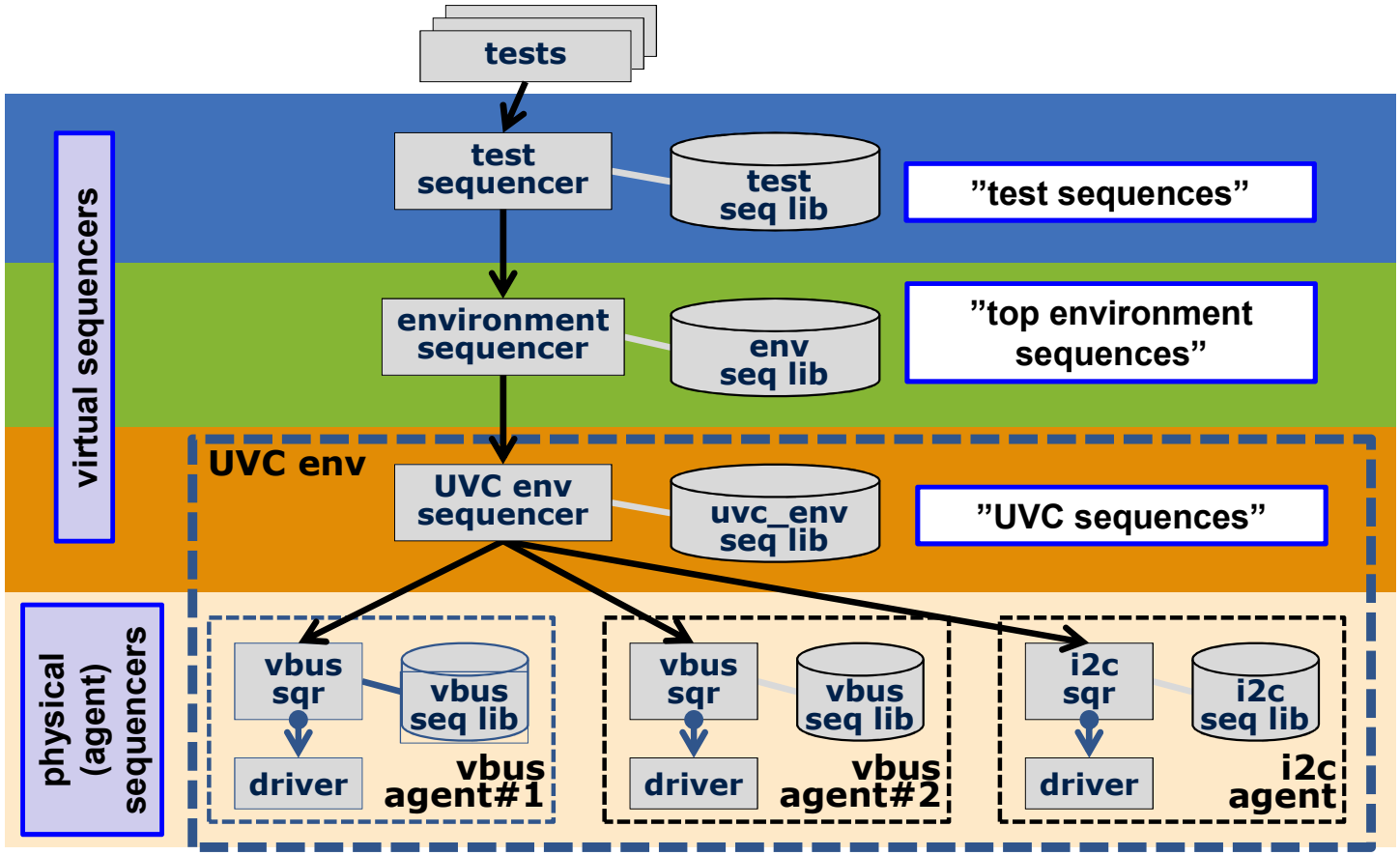
- UVM Sequences are widely (mis)used
  - Testbench complexity impacts project schedules
  - Stimulus cannot be adequately controlled
  - Code cannot be reused

- Our proposed solution
  - Sequence API methodology, Sequence guidelines
  - Addresses complexity, control, and reuse
  - Developed from extensive project experience

Sequence API Strategy

Sequence API Strategy

Sequence Layers

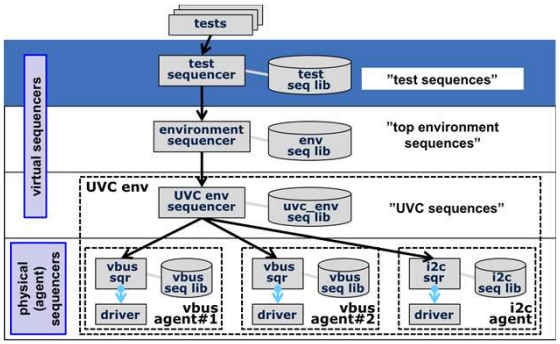


Sequence API Strategy

Example Sequences

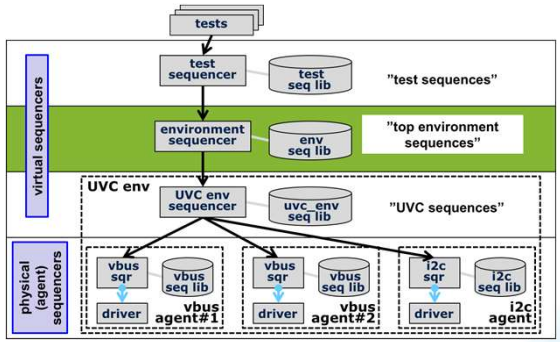
- **Test sequence**

```
class transfer_test_seq extends subsys_base_test_seq;
  all_subsys_reset_seq all_reset;
  ...
  task body();
    `uvm_do(all_reset)
```



- **Top environment sequence:**

```
class all_subsys_reset_seq extends subsys_base_seq;
  subsys1_reset_seq reset_subsys1;
  subsys2_reset_seq reset_subsys2
  ...
  task body();
    `uvm_do(reset_subsys1)
    `uvm_do(reset_subsys2)
    ...
```

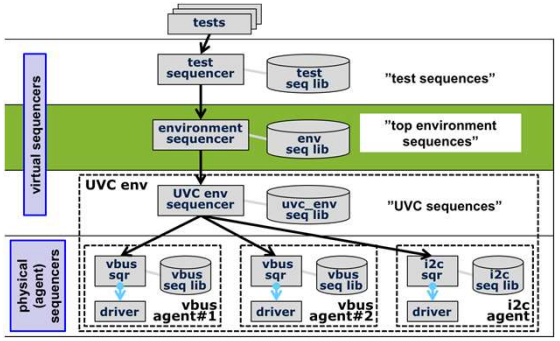


Sequence API Strategy

Example Sequences

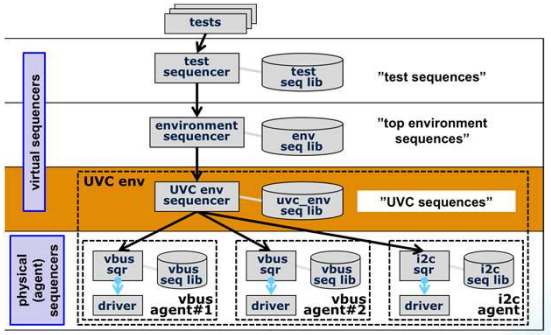
- **Top environment sequence (repeated)**

```
class all_subsys_reset_seq extends subsys_base_seq;
  subsys1_reset_seq reset_subsys1;
  subsys2_reset_seq reset_subsys2
  task body();
    `uvm_do(reset_subsys1)
    `uvm_do(reset_subsys2)
```



- **UVC sequence**

```
class subsys1_reset_seq extends subsys_base_seq;
  vbus1_reset_seq vbus1_rst;
  vbus2_reset_seq vbus2_rst;
  ...
  task body();
    `uvm_do(vbus1_rst)
    `uvm_do(vbus2_rst)
```

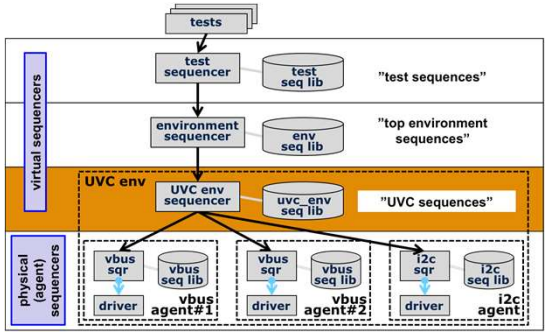


Sequence API Strategy

Example Sequences

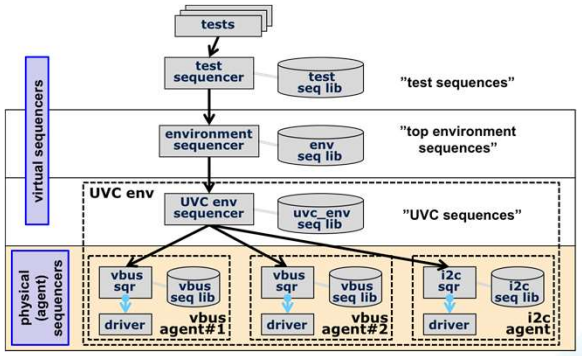
- **UVC sequence (repeated)**

```
class subsys1_reset_seq extends subsys_base_seq;
  vbus1_reset_seq  vbus1_rst;
  vbus2_reset_seq  vbus2_rst;
...
  task body();
    `uvm_do(vbus1_rst)
    `uvm_do(vbus2_rst)
```



- **Agent sequence**

```
class vbus1_reset_seq  extends vbus_base_seq;
  `uvm_declare_p_sequencer vbus_sequencer;
  ...
  task body();
    `uvm_do(req, p_sequencer)
    ...
```



Sequence API Strategy

Goals for Sequences

- Complexity
  - Tests are simple and readable
  - UVC details hidden from tests
- Control
  - Tests use an API to target stimulus scenarios
  - UVC options derived from test options
- Reuse
  - Tests are independent of project/testbench implementation details
  - Sequences are reusable on testbenches with different implementations

Sequence Guidelines

Guideline 1

Produce legal stimulus by default

- Users can provide 0 or more inline constraints.

```
class top_seq extends base_seq;
  …
  task body();
    `uvm_do(ahb_burst_seq_inst)
```

No knobs provided

start sequence

```
class ahb_burst_seq extends base_seq;

  constraint legal_c{
    dir inside {WRITE, READ};
    addr + length < cfg.get_max_addr();
    …
  }
```

Enforce legal stimulus by default

Guideline 1

Produce legal stimulus by default

- Users can provide 0 or more inline constraints.

```
class top_seq extends base_seq;
  …
  task body();
    `uvm_do_with(ahb_burst_seq_inst,
                 {addr == 'hFF00;})
```

Address set by user

start sequence

```
class ahb_burst_seq extends base_seq;

  constraint legal_c{
    dir inside {WRITE, READ};
    addr + length < cfg.get_max_addr();
    …
  }
```

Length still random and legal

Guideline 1

Produce legal stimulus by default

```
class top_seq extends base_seq;
  …
  task body();
    `uvm_do_with(ahb_burst_seq_inst,
          {addr   == 'hFF0;
           length ==  1024;})
```

Users must manage legal rules in higher sequences.

start sequence

Avoid This!

```
class ahb_burst_seq extends base_seq;
  constraint legal_c{
    dir inside {WRITE, READ};
    addr < cfg.get_max_addr();
    …
  }
```

May produce illegal burst length

Guideline 1

Produce legal stimulus by default

- Protect users from illegal stimulus.

```
class top_seq extends base_seq;
  …
  task body();
    `uvm_do_with(ahb_burst_seq_inst,
                 {addr == 'hFFFF_FFF0;
                  length == 256;})
```

Illegal Address and Length

start sequence

```
class ahb_burst_seq extends base_seq;
  constraint legal_c{
    dir inside {WRITE, READ};
    addr + length < cfg.get_max_addr();
    …
  }
```

We ***want*** constraint solver failures on illegal options

Guideline 2

Constrain control knobs with class constraints

Pass results with inline constraints

```
class ahb_write_burst_seq extends ahb_base_seq;
  …
  `uvm_do_with(ahb_seq,
               ahb_seq.hwrite == HWRITE_WRITE;
               ahb_seq.hburst inside {HBURST_SINGLE, HBURST_INCR};
```

Avoid This!

start sequence

```
class ahb_burst_seq extends ahb_base_seq;
   rand write_enum hwrite;
   rand burst_enum hburst;
   …
  }
```

Don't Pass Inline

Guideline 2

Constrain control knobs with class constraints

Pass results with inline constraints

```
class ahb_write_burst_seq extends ahb_base_seq;
  rand hburst_t  hburst;

  constraint c_hburst {
    hburst inside {HBURST_SINGLE, HBURST_INCR};
  }
  task body();
    `uvm_do_with(ahb_seq,
                 ahb_seq.hwrite == HWRITE_WRITE;
                 ahb_seq.hburst inside {HBURST_SINGLE, HBURST_INCR};
                 ahb_seq.hburst == local::hburst;
```

Use Class Constraint

Pass Result to Sequence

start sequence

```
class ahb_burst_seq extends ahb_base_seq;
   rand hwrite_t hwrite;
   rand hburst_t hburst;
   …
```

Two-Step Randomization:

1. Randomize Class Variables
2. Run **body()** to randomize lower sequences.

Guideline 3

Minimize the number of control knobs

```
class ahb_master_write_seq extends ahb_base_seq;
  rand int slave_num;
  …
  protected rand int  slave_id;

  constraint id_c {
    slave_id == p_sequencer.cfg.get_slave_id(slave_num);
  }
  virtual task body();
     ahb_seq_item req;
    `uvm_do_with(req,{req.hwrite == HWRITE_WRITE;
                      req.hprot3 == p_sequencer.cfg.get_hprot3();
                      req.haddr[31:24] == local::slave_num;
                      req.id == local::slave_id})
```

Exposed Control Knob

Encapsulated Control Knob

Users can't control these.

Guideline 4

Constrain each control knob in a dedicated constraint block

```
class ahb_burst_seq extends ahb_base_seq;
  constraint all_burst_c {
    saddr inside ([MIN_ADDR_C:MAX_ADDR_C]);
    length    <=   MAX_LENGTH_C;
    hburst inside {HBURST_SINGLE, HBURST_INCR, …};
    …
  }
```

All constraints defined in a single constraint block.

Avoid This!

```
class my_burst_seq extends ahb_burst_seq;
  constraint all_burst_c {
    saddr inside ([MIN_ADDR_C:MAX_ADDR_C]);
    length <= MAX_LENGTH_C;
    hburst inside {HBURST_SINGLE, ...};
    …
  }
```

Must redefine and duplicate constraints just to change hburst.

Guideline 4

Constrain each control knob in a dedicated constraint block

```
class ahb_burst_seq extends ahb_base_seq;
  constraint saddr_c {saddr inside ([MIN_ADDR_C:MAX_ADDR_C]);}
  constraint length_c {length <= MAX_LENGTH_C;}
  constraint hburst_c {hburst inside {HBURST_SINGLE, HBURST_INCR,…};
}
```

extends

```
class my_burst_seq extends ahb_burst_seq;
  constraint hburst_c {hburst inside {HBURST_SINGLE, …};}
  …
```

Dedicated constraints for each field.

Redefine only 1 constraint

Guideline 5

Use soft constraints carefully and sparingly

```
class ahb_fabric_master_write_seq extends base_seq;
  task body();
    `uvm_do_on_with(ahb_master_write_seqs[b],
                    p_sequencer.agent_sequencer[b],
                    {slave_num inside {[0:3]};})
```

start sequence

```
class ahb_master_write_seq extends ahb_base_seq;
  rand int   slave_num;
  constraint default_c {
    soft slave_num == 0;}
  …
```

Avoid This!

Silently restricts what user asks for

Guideline 5

Use soft constraints carefully and sparingly

```
class ahb_master_write_seq extends ahb_base_seq;
  rand int      slave_num;
  rand burst_t hburst;

  constraint default_c {
    soft  slave_num == 0;
    soft  hburst    == INCR;
    soft (slave_num == 0) -> hburst == SINGLE; }
  …
```

Conflict between **slave_num** and **hburst**.

Implication operator can influence either side!
hburst == INCR implies
slave_num != 0.

We can't control which soft constraint is dropped!

Guideline 5

Use soft constraints carefully and sparingly

- Helpful Soft Constraints
  - Default “Mode of Operation” Flags
  - Extended “Fixed Sequences”

```
class ahb_master_write_seq extends ahb_base_seq;
  rand bit  allow_illegal_paths;
  constraint c_slave_num {
    soft allow_illegal_paths == 0;
  }
```

Default to legal stimulus.

```
class ahb_burst_seq extends ahb_base_seq;
  constraint saddr_c{ saddr inside ([MIN_ADDR_C:MAX_ADDR_C]);}
  constraint length_c {length <= MAX_LENGTH_C;}
  constraint hburst_c {hburst inside {HBURST_SINGLE,HBURST_INCR,…};
```

extends

```
class fixed_burst_seq extends ahb_burst_seq;
  constraint scenario_c {
    soft saddr  == 'h1000;
    soft length == 32;
    soft hburst == HBURST_SINGLE}
```

Extended sequence adds soft constraints on top of base legal constraints.

Guideline 6

Use enumerated types for control knobs

```
class my_test_seq extends ahb_fabric_base_vseq;
  ...
   task body();
     ahb_master_seq master_seq;
     `uvm_do_with(master_seq, {
                    hsize == HSIZE_256)
```

start sequence

```
class ahb_master_seq extends ahb_base_seq;
  rand hsize_t hsize;
  …
  task body();
    `uvm_do_with(req,{hsize == local::hsize;})
```

```
typedef enum bit [2:0] {
  HSIZE_8    = 3'b000,
  HSIZE_16   = 3'b001,
  HSIZE_32   = 3'b010,
  HSIZE_64   = 3'b011,
  HSIZE_128  = 3'b100,
  HSIZE_256  = 3'b101,
  HSIZE_512  = 3'b110,
  HSIZE_1024 = 3'b111
} hsize_t;
```

- Don’t make users memorize encodings
- Changes in encoding are transparent to sequence
- Can randomize and constrain enum types

Guideline 7

Make tests independent of testbench architecture

```
class ahb_fabric_master_write_seq extends base_seq;
  task body();
    ahb_fabric_master_write_seq master_write_seq;
    `uvm_do(master_write_seq)
  endtask: body
```

Test-level sequence decoupled from testbench architecture.

start mid-level sequence

Test sequences are generic and reusable for derivative projects.

```
class ahb_fabric_master_write_seq extends ahb_base_seq;
  ...
  task body();
    ahb_master_write_seq ahb_master_write_seqs[string];
    ...
    `uvm_do_on_with(ahb_master_write_seqs[b],
                    p_sequencer.agent_sequencer[b],
                    {…})
```

Only mid-level sequences reference sequencers.

Guideline 8

Use descriptor objects to encapsulate complex constraint sets

```
class config_shape_seq extends base_seq;
  rand int num_shapes;
  rand shape_descriptor shapes[];

  constraint size_c {
    num_shapes inside {[1:8]};
    shapes.size() == num_shapes;
  }

  task body();
    foreach (shapes[i]) begin
      regmodel.CFG0.RMPSTEP3.set(shapes[i].ramp_step_up);
      `uvm_do_with(start_pll_seq, {
        polarity == shapes[cfg.get_active_shape()].polarity; ... })
```

```
class shape_descriptor extends uvm_object;
  rand shape_kind_t      shape;
  rand shape_polarity_t  polarity;
  rand int               ramp_step_up;
  rand int               ramp_samples_up;
  ... // etc

  constraint legal_c {...}
```

Arrays of Descriptors

Use loops to manage large sets of control knobs.

Apply *different* constraints based on runtime configuration!

Guideline 9

Use configuration objects and accessor methods to adapt to project-specific configurations

```
class ahb_cfg extends uvm_object;
  rand int slv_fifo_depth;
  ...
  constraint {
    slv_fifo_depth inside {[1:`MAX_FIFO_DEPTH]};
  };

  function int get_fifo_depth();
    return(this.slv_fifo_depth);
  endfunction
```

Keep project-specific configuration constraints outside of sequences.

```
class fifo_test_seq extends fabric_base_seq;
  ...
   task body();
     for(int i=0; i<=cfg.get_fifo_depth(); i++) begin
       `uvm_do_with(master_seq, {
                    hsize == HSIZE_32,
                    hburst == SINGLE)
```

Sequence is generic:

- Reusable between projects.
- Changes in spec are transparent.

Guideline 10

Use utility methods to support self-tuning sequences

```
function automatic int calc_data_offset_from_address(ADDR_t addr);
    return(addr / DATA_WORD_SIZE) % DATA_WORDS_PER_ADDR);
endfunction
```

```
package ahb_pkg;
  `include "ahb_common.sv"
  ... //etc
endpackage
```

Package-scope methods perform common calculations for sequences.

```
class write_word_seq extends base_seq;
  rand bit[31:0] addr;

  task body();
    bit[31:0] ram_addr = addr / DATA_WORDS_PER_ADDR;

    `uvm_do_with(write_single_seq, {
                 addr == local::ram_addr;
                 word_sel == calc_data_offset_from_addr(local::addr)})
```

- Derive values using formulas
- Calculate delays for transactions
- Calculate timeouts for waiting and checking

Conclusion

Maximize Efficiency and Productivity

- Testbench is easier to use
- Problems are easier to debug
- Testbench can easily adapt to design changes

verilab snug
Synopsys Users Group
Austin

Thank You

References

- https://en.wikipedia.org/wiki/Encapsulation_(computer_programming)
- https://en.wikipedia.org/wiki/Mutator_method
- Martin, Robert C. “Clean Code: A Handbook of Software Craftsmanship”, Prentice Hall, 2008.
- Accelera, “Standard UVM Class Reference, v1.2” http://www.accellera.org/downloads/standards/uvm
- "599_FACE” by TransformersMan is licensed under CC BY 2.0

Question/Answer

Q: You said produce legal stimulus by default. What if I want to generate illegal scenarios?
A: There are many techniques to enable error injection. Our point is this is not “normal” or “default” behavior, therefore legality should be enforced. Some error injection techniques are to add control knobs to enable illegal behavior, extend sequences with error cases and use factory overrides, or use configuration object flags to enable errors.

Q: Why do I need a sequence library for tests? Why can’t I develop a method API, like what’s done in software?
A: A virtual sequence is similar to a method, but as the title of the paper “Use the Sequence, Luke” implies, virtual sequences are more powerful because of the constrained-random features they provide.

Q: Having tests call sequences from the “top environment sequences” might lead to really verbose tests if there are multiple environments (e.g. reset subsystem 1 sequence, reset subsystem 2 sequence, etc.). What about having another layer of sequences between the test sequence and the “top environment sequences”?
A: You can have another layer, but even better, is to have top environment sequences that compose other top environment sequences.