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 Layers**
  - **test sequencer**
    - **test seq lib**
    - **test sequences**
  - **environment sequencer**
    - **env seq lib**
    - **top environment sequences**
  - **UVC env sequencer**
    - **uvc_env seq lib**
    - **UVC sequences**
  - **physical (agent) sequencers**
    - **vbus agent #1**
      - **vbus seq lib**
      - **vbus sqr**
      - **vbus driver**
    - **vbus agent #2**
      - **vbus seq lib**
      - **vbus sqr**
      - **vbus driver**
    - **i2c agent**
      - **i2c seq lib**
      - **i2c sqr**
      - **i2c driver**

---

Copyright 2018, Verilab and Synopsys
Sequence API Strategy
Example Sequences

- **Test sequence**

```verilog
class transfer_test_seq extends subsys_base_test_seq;
    all_subsys_reset_seq all_reset;

    task body();
        `uvm_do(all_reset)
```

- **Top environment sequence**

```verilog
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)
```

*Copyright 2018, Verilab and Synopsys*
Sequence API Strategy

Example Sequences

- **Top environment sequence (repeated)**

```vsys语言
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**

```vsys语言
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)
```

Copyright 2018, Verilab and Synopsys
Sequence API Strategy

Example Sequences

- **UVC sequence (repeated)**

```verilog
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**

```verilog
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.

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

class ahb_burst_seq extends base_seq;

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

No knobs provided

Enforce legal stimulus by default
Guideline 1
Produce legal stimulus by default

• Users can provide 0 or more inline constraints.

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

class ahb_burst_seq extends base_seq;

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

Address set by user

Length still random and legal

start sequence
Guideline 1
Produce legal stimulus by default

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

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

Users must manage legal rules in higher sequences.

Avoid This!

May produce illegal burst length
Guideline 1
Produce legal stimulus by default

- Protect users from illegal stimulus.

```verilog
class top_seq extends base_seq;
  ...
  task body();
    `uvm_do_with(ahb_burst_seq_inst,
    {addr == 'hFFFF_FFF0;
    length == 256;})
```
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};
    
class ahb_burst_seq extends ahb_base_seq;
    rand write_enum hwrite;
    rand burst_enum hburst;
    ...
}
Guideline 2
Constrain control knobs with class constraints
Pass results with inline constraints

```cpp
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

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

```java
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, ...};
        ...
    }

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, ...};
        ...
    }
```

- All constraints defined in a single constraint block.
- Must redefine and duplicate constraints just to change hburst.

Avoid This!
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

```plaintext
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]};})
endtask
```

Avoid This!

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

Silently restricts what user asks for
Guideline 5
Use soft constraints carefully and sparingly

```java
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`.
- `slave_num != 0` implies `hburst != INCR`.

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”

```verilog
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.
```

```verilog
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,...};}
```

```verilog
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

- 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

Test-level sequence decoupled from testbench architecture.

Test sequences are generic and reusable for derivative projects.

Only mid-level sequences reference sequencers.

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

start mid-level sequence

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],
            {...})
```
**Guideline 8**

Use descriptor objects to encapsulate complex constraint sets

```verilog
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;
        ... })
  }
```

```verilog
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

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)
        endfor

Keep project-specific configuration constraints outside of sequences.

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

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)"})

Package-scope methods perform common calculations for sequences.

- 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
Thank You
References

- [https://en.wikipedia.org/wiki/Mutator_method](https://en.wikipedia.org/wiki/Mutator_method)
- "599_FACE" by TransformersMan is licensed under [CC BY 2.0](https://creativecommons.org/licenses/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.