2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Advanced UVM Command Line Processor

Siddharth Krishna Kumar (Samsung Austin Research Center)

accellera SYSTEMS INITIATIVE

Agenda

- Introduction
- Scenarios
- Features
- Coding Examples
- Benefits
- Drawbacks
- Future Enhancements
- References
- Q/A

Introduction

- Custom extension of *uvm_cmdline_processor* class
  - Access to all base class methods
- Central maintenance and randomization of control knobs
- Powerful, robust and portable
  - Plug and play into an UVM environment
- Reduced variables in test bench code
  - Removes need for multiple fields
- Readable command line arguments

Scenarios

- Delay knobs
  - Burst behavior in interfaces is desired
  - Typical approach – provide min-delay and max-delay
  - Mix of large and small values not possible
- Opcodes
  - Shaping ISA traffic especially in CPU/GPU verification
  - Mapping opcode to hex/dec value can be challenging
- Instance specific control
  - *set_config_db* in command line is used to control value of fields per instance
  - Takes in a single value

Features

parse_opt(string)

- method to parse argument in special format
- *+cmdline_example_1=min_val_0~max_val_0:wt_0, val_1:wt_1,…min_val_n~max_val_n:wt_n*

parse_val(string)

- method to distinguish hex, bin or dec
- *+cmdline_hex_example=h1:10,hdead,h100~h1000:89*
- *+cmdline_bin_example=b100:10,b111,b011~b110:89*

set_weight_q()

- method to push weights into weight queue
- *<n>* value is pushed = *wt_<n>* times

get_rand_int()

- method returns random value
- Invokes *std::randomize()* in conjunction with the buckets and weight queue to generate random value

acmdline_opts

class to store values/ min-max range/weights

Features

acmdline_opts_enum#(type T=uvm_active_passive_enum)

- class to store values/min-max range/ weights of enums
- extends acmdline_opts

parse_val(string)

- method overridden to interpret each value as string
- *uvm_enum_wrapper#(T)::from_name(string,enum)* used to map string to an enum and stored as long int
- *+cmdline_enum_example=ZERO:3,ONE:2,2:1 : (typedef enum {ZERO, ONE, TWO} count_t;)*

Features

get_inst()

- returns singleton handle of the class

acmdline_processor

- maintains data-base
- provides API methods

is_valid(string, string)

- method for user to check if the argument has been passed in the command line
- returns 1 or 0

get_rand_val(string,string,string,string,opts)

- method invoked by user to get a random value for a given named control knob
- first argument is required, rest all are optional
- errors on illegal format or no values provided

parse_cmdline(string,string,string,string,opts)

- invoked internally to parse and add entry into data-base
- returns 0 if unsuccessful

Coding examples

Example 1

```
typedef enum {ADD,SUB,MUL} opcode_t;

opt_proc = advanced_cmdline_processor::get_inst();

class packet extends uvm_sequence_item;

  rand opcode_t   opcode;
  rand bit [2:0]  return_order;
  rand bit [31:0] operand_a;
  rand bit [31:0] operand_b;

  constraint c_return_order {
    (opcode == ADD) -> (return_order == 3);
    (opcode == SUB) -> (return_order inside {[1:4]};
  }

  constraint c_operand_a {
    (opcode == MUL) -> (operand_a[2:0] == 3'b00);
  }

  function new(string name = "pkt");
    super.new(name);
  endfunction

  function void pre_randomize();

    if(advanced_opt_proc_enum#(opcode_t)::is_valid("opcode")) begin
      opcode = advanced_opt_proc_enum#(opcode_t)::get_rand_enum("opcode");
      opcode.rand_mode(0);
    end

    if(opt_proc.is_valid("operand_a")) begin
      operand_a = opt_proc.get_rand_val("operand_a");
      operand_a.rand_mode(0);
    end
  endfunction
endclass

class packet_sequence extends uvm_sequence;

 task body();
   packet pkt;

   for(int i=0;i<10;i++) begin
     pkt = packet::type_id::create("pkt");
     pkt.randomize();

     start_item(pkt);
     finish_item(pkt);
   end
  end
endclass
```

- *+opcode=ADD:80,SUB:20*
  - 10 packets generated with 8 ADD, 2 SUB and 0 MUL
- *+operand=32'hfffffff0*
  - specify value in accordance with constraints
- *+operand=32'h00000000~32'h0000000f*
  - constraint solver issue if MUL opcode is picked

Coding examples

Example 2

```
class packet_sequence extends uvm_sequence;

 task body();
   packet pkt;

   for(int i=0;i<10;i++) begin
     pkt = packet::type_id::create("pkt");
     pkt.randomize();

     start_item(pkt);
     finish_item(pkt);

     p_sequencer.delay(opt_proc.get_rand_val("pkt_delay",get_full_name(),"0","Control
the delay between pkts in the sequence");  // assumption : parent sequencer has a task
                                           // delay(int num_cycles);
   end
  end
endclass

class packet_virtual_sequence extends uvm_sequence;
  packet_sequence  pkt_seq_1, pkt_seq_2;
  packet_sequencer sqr;

  function new(string name = "pkt_vir_seq");
    super.new(name);
    pkt_seq_1 = packet_sequence::type_id::create("pkt_seq_1");
    pkt_seq_2 = packet_sequence::type_id::create("pkt_seq_2");
    sqr       = packet_sequencer::type_id::create("sqr", this);
  endfunction

  task body();
    pkt_seq_1.start(sqr);
    pkt_seq_2.start(sqr);
  endtask
```

- Show cases use of per instance control
  - *+uvm_set_config_string=*pkt_seq_1,pkt_delay,"0:50,1:50"*
    *+uvm_set_config_string=*pkt_seq_2,pkt_delay,"10~20:50,21~100:40,101~500:10*
- 1st sequence will only have 0 or 1 cycle delays
- 2nd sequence will see a range of delays

Coding examples

Example 3

```
module tb();

dut dut(.A(opt_proc.get_rand_val("dut_A","","0")),
        .B(opt_proc.get_rand_val("dut_B","","1"))
       );

endmodule
```

- Can directly be used in DUT – DV connections
- CAUTION: watch out for port width mis-match warnings

Benefits

- Avoid code redundancy
- Central framework for code optimization
- Scalability and ease of portability to existing test benches
- Re-run/create multiple state space without re-compile
- Add to existing RAL setup for randomization of register settings

Drawbacks

- Not a replacement of traditional randomization or constraints
- Uni-directional implication failure possible
- String based searches affect performance in larger data-base

Future enhancements

- Collect all metrics on each call and dump data-base
- Post simulation processing and visualization of data base to ensure quality DV code
- Feedback loop without having to write, dump and analyze coverage
- Entire code to be made available, right now it is part of the Appendix section of the paper

References

- System Verilog LRM IEEE Std.1800-2017(Revision of IEEE Std. 1800-2012)
- Universal Verification Methodology (UVM) 1.2 Class Reference

Questions?