DVCon 2012

Design & Verification Conference & Exhibition

February 28 – March 1, 2012

Better Living Through Better Class-Based SystemVerilog Debug

by

Rich Edelman, Raghu Ardeishar, John Amouroux

Verification Technologist and Questa Engineering

Mentor Graphics

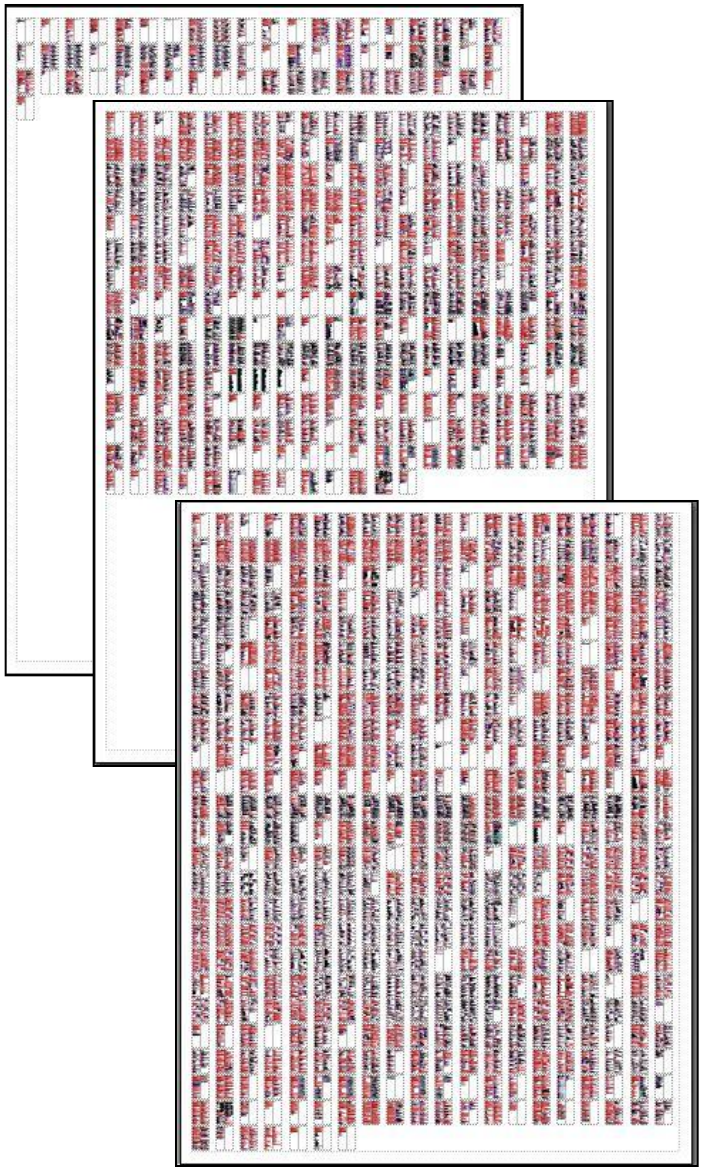
Questa™ ADVANCED VERIFICATION

Why are you here?

1. Already got some free UVM, having a little trouble debugging my code
2. Already got some free UVM, like it so much, want to know where I can pay for it
3. Heard UVM was being given away for free – wanted some
4. Isn't this comedy central auditions?
5. Didn't leave fast enough from the previous topic
6. The next topic is my favorite – wanted to get a good seat

History Tour

- avm-3.0-update-4
  - 106 classes
  - 38 files, **6,589** lines
  - 61 printed pages
- ovm-2.1.2
  - 236 classes
  - 106 files, **37,786** lines
  - 293 printed pages
- uvm-1.1a
  - 316 classes
  - 134 files, **67,298** lines
  - 495 printed pages



Printed Pages displayed in a 20 x 25 grid

We have a problem

- RTL is just out. Manager screaming about getting the "1000's of RTL bugs found and fixed".
  - The verification team has very little time.
  - They need to build and debug their testbenches fast.
  - Testbenches are SystemVerilog class based, using UVM.
- Archie Architect
- Bob Blocktest
- Cindy SOCtest (no relation)
- Doug Debug
- Elaine ESL (always thinking the big thoughts)

Doug Debug's Job

- Doug's job is changing.
- His team has adopted class-based SystemVerilog and specifically the UVM.
- They had neither budget nor time for training.
  - So they train a little every day…
- The testbench is under development.
- Doug has ideas to help his team!

Doug Debug

- Doug decides to use the proprietary debug features available in the simulator vendors they use (three!).
- But, just in case, he has built-in some special debug features for his team to use.
- Rules about Doug's built-in debug:
  - Force the team to do just a little planning and preparation, but no more
  - Adhere to the LRM. No proprietary syntax or features.

What is a dynamic testbench?

- In the old days…
  - Verilog RTL
  - Verilog testbench, gold files?, expected outputs
  - Elaborated by the simulator, run the test
- Dynamic…
  - Verilog RTL
  - SystemVerilog class-based testbench, OOP
    - At time 0, or time 1000, create a testbench.
    - Over the course of simulation create transactions – dynamically.
    - Classes created on the fly.
    - Testbench code looks like software.

What does this mean for debug? → Software debug

- Control Flow
  - if-then-else, case, function calls, task calls
  - Threads (fork/join{any, none})
- Data Types
  - int, bit, packed struct, unpacked struct
  - Queue, Dynamic array, Associative array, regular array
  - Data state (transactions)
  - UVM – class library, data structures, data model
- Software / Object oriented concepts
  - Stack, automatic variables
  - Inheritance, classes, virtual functions

What can Doug do to cope?

- Understand his new testbench
  - What is it doing?
  - Why is it doing it?
- Use a debugger
  - Breakpoint
    - Dynamic object
  - Single step
    - Millions of transactions
- Use '$display'?

Organize code

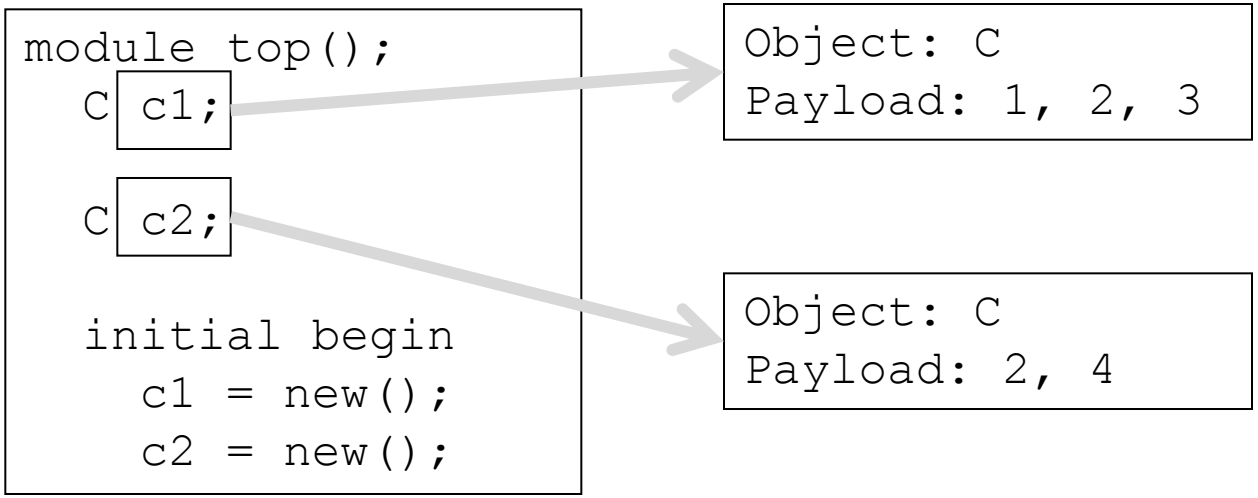
- Name directories and files well
- Name classes well
- Use SystemVerilog packages

- Good naming is really just establishing the rules, and sticking to them.
- Be consistent.

- Many naming conventions exist on the web. Adopt one.

Designing a class

```
class C;
  int payload[];
  function new();
    ...
  endfunction
endclass
```



Designing a class to be "debug useful"

```
class C;
  static int g_id = 0;
  int id;
  int payload[];
  function new();
     id = g_id++;
  endfunction

  function string convert2string();
    return $sformatf("id=%0d, payload=%p", id, payload);
  endfunction
endclass
```

```
C c1, c2;

c1 = new();
c2 = new();

$display("c1=%s",
   c1.convert2string());
```

Designing a class to be "debug useful"

```
class C extends base_class;


  int payload[];
  function new();
    super.new();
  endfunction

  function string convert2string();
    return $sformatf("id=%0d, payload=%p", id, payload);
  endfunction
endclass
```

```
C c1, c2;

c1 = new();
c2 = new();

$display("c1=%s",
   c1.convert2string());
```

```
class base_class;
  static int g_id = 0;
  int id;
  function new();
    id = g_id++;
  endfunction
```

Breadcrumbs – leave a trail

```
32      foreach (payload[i]) begin
33        word = payload[i];
34        if (word == 4) begin
35          $display("DOUG_DBG: %s:%0d", `__FILE__, `__LINE__);
36          if (prev_word == 2) begin
37            $display("DOUG_DBG: %s:%0d", `__FILE__, `__LINE__);
38            word = word+2;
39          end
40          else if (prev_word == 1) begin
41            $display("DOUG_DBG: %s:%0d", `__FILE__, `__LINE__);
42            word = word-1;
43          end
44          else begin
45            $display("DOUG_DBG: %s:%0d", `__FILE__, `__LINE__);
46            return value;
47          end
48        end
          ...
51        $display("i=%0d, word=%0d, value=%0d", i, word, value);
```

Breadcrumbs – leave a trail

```
% grep DOUG_DBG transcript
# DOUG_DBG: t.sv:35
# DOUG_DBG: t.sv:45
# DOUG_DBG: t.sv:35
# DOUG_DBG: t.sv:45
# DOUG_DBG: t.sv:35
# DOUG_DBG: t.sv:45
```

```
# i=0, word=7, value=7
# i=1, word=1, value=15
# i=2, word=7, value=31
...
# i=19, word=7, value=4192223
# DOUG_DBG: t.sv:35
# DOUG_DBG: t.sv:45
# c1=id=0, payload='{7, 1, 7, 3, 6, 7, 1, 7, 2, 9, 2, 3, 2, 10, 10, 1, 3, 7, 9,
7, 4, 8, 10, 3, 2, 4, 10, 7, 1, 0, 10, 1, 9, 1, 5, 9, 4, 0, 1, 6}, calc=3ff7df
# i=0, word=9, value=11
...
# i=13, word=9, value=131071
# DOUG_DBG: t.sv:35
# DOUG_DBG: t.sv:45
# c2=id=1, payload='{9, 9, 0, 2, 10, 1, 6, 9, 7, 0, 7, 3, 3, 9, 4, 7, 1, 7, 5,
1, 3, 4, 5, 3, 4, 4, 6, 4, 1, 3, 1, 4, 9, 4, 2, 5, 3, 7, 6, 9, 10}, calc=1ffff
# DOUG_DBG: t.sv:35
# DOUG_DBG: t.sv:45
# c3=id=2, payload='{4, 4, 2, 9, 3, 8, 10, 8, 7, 7, 10, 5, 7, 5, 0, 2, 8, 5, 9,
4, 5, 5, 7, 8, 5, 6, 8, 8, 5, 2, 1, 0, 4, 7, 8, 1, 5, 2, 10, 2, 7, 1}, calc=1
```

Payload has >= 40 items
We're only visiting 20, 14, 0!

Breadcrumbs – leave a trail (2)

```
32      foreach (payload[i]) begin
33        word = payload[i];
34        if (word == 4) begin
35          $display("DOUG_DBG: %s:%0d", `__FILE__, `__LINE__);
36          if (prev_word == 2) begin
37            $display("DOUG_DBG: %s:%0d", `__FILE__, `__LINE__);
38            word = word+2;
39          end
40          else if (prev_word == 1) begin
41            $display("DOUG_DBG: %s:%0d", `__FILE__, `__LINE__);
42            word = word-1;
43          end
44          else begin
45            $display("DOUG_DBG: %s:%0d", `__FILE__, `__LINE__);
46            return value;
47          end
48        end
          ...
51        $display("i=%0d, word=%0d, value=%0d", i, word, value);
```

Better Breadcrumbs

- Don't leave them on all the time

```
if (debug) $display(…)
`uvm_info()
```

```
`uvm_info
`uvm_warning
`uvm_error
`uvm_fatal
```

- Turn off `uvm_info by default by making the verbosity be above normal. (Normal is UVM_MEDIUM)

```
`uvm_info("ID", "MSG", UVM_HIGH)
```

From uvm_object_globals.svh:

```
UVM_FULL   - Printed if verbosity is set to UVM_FULL or above.
UVM_HIGH   - Printed if verbosity is set to UVM_HIGH or above.
UVM_MEDIUM - Printed if verbosity is set to UVM_MEDIUM or above.
UVM_LOW    - Printed if verbosity is set to UVM_LOW or above.
UVM_NONE   - Report is always printed. Verbosity level setting can not disable it.
```

Better Breadcrumbs (2)

- Turn everything on

```
`uvm_info("ID", "MSG", UVM_HIGH)
+uvm_set_verbosity=test.*,_ALL_,UVM_HIGH,time,0
```

+uvm_set_verbosity=<comp>,<id>,<verbosity>,time,<time>

Better Breadcrumbs (3)

- Turn on by instance (components and sequences)

```
+uvm_set_verbosity=test.e2_b.*,_ALL_,UVM_FULL,time,0

+uvm_set_verbosity=test.e2_b.driver*,_ALL_,UVM_FULL,time,0

+uvm_set_verbosity=test.*.driver*,_ALL_,UVM_FULL,time,0
```

Better Breadcrumbs (4)

- Turn on by ID
  - `uvm_info("ID", "MSG", UVM_HIGH)

```
+uvm_set_verbosity=test.*,sequence_B_grandparent#(sequence_item_A),\
  UVM_FULL,time,0
+uvm_set_verbosity=test.*,sequence_A,UVM_FULL,time,0
```

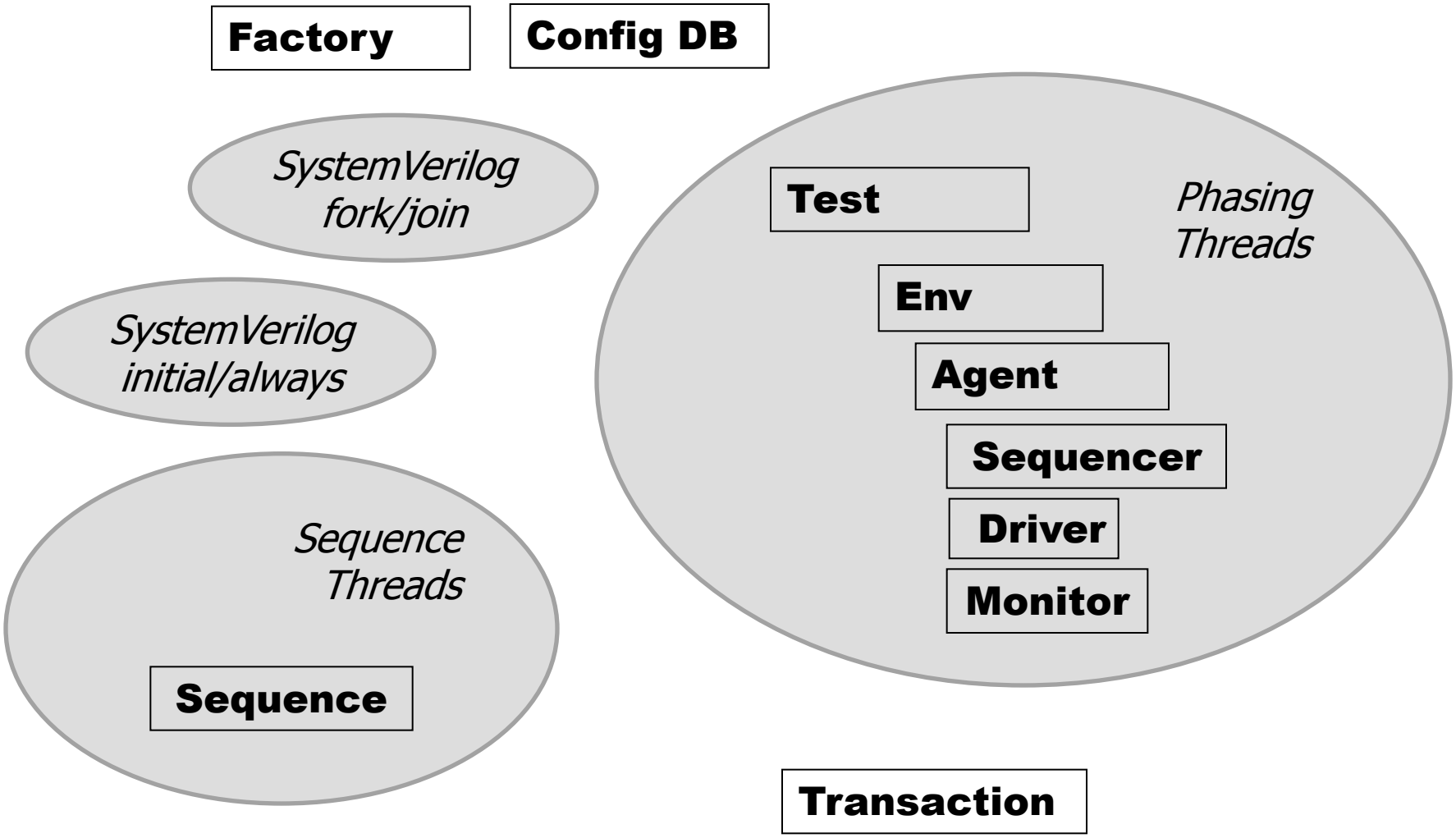
- Good IDs
  - In a class, use the type_name

```
`uvm_info(get_type_name(), "msg, …)
```

  - In a task or function – add the function name

```
`uvm_info({get_type_name(), ":collect_transactions"},
  "msg", …
```

UVM Constructs



Doug's Driver – build_phase

```
249   class driverB #(type T = int) extends uvm_driver #(T);
250     `uvm_component_param_utils(driverB#(T))
251
252     const static string type_name = {"driver2#(", T::type_name, ")"};
253     virtual function string get_type_name();
254       return type_name;
255     endfunction
263
264     function void build_phase(uvm_phase phase);
265       int simple_int = -1;
266       `ALLOCATED_TYPE
267       if( !uvm_config_db#(int)::get( this, "*", "simple_int", simple_int))
268         `uvm_fatal("ENV", "simple_int not set")
          ...
275       `uvm_info("DRVR", $sformatf("simple_int=%0d", simple_int), UVM_MEDIUM)
          ...
277     endfunction
```

```
`define ALLOCATED_TYPE() \
    objects::add(get_type_name());
```

Doug's Driver – run_phase

```
274     task run_phase(uvm_phase phase);
275       integer tr_handle;
276       T t;
277       `uvm_info(get_type_name(), "Starting... ", UVM_MEDIUM)
278       forever begin
279         seq_item_port.get_next_item(t);
280         `uvm_info("DRVR", $sformatf("Got t=%s",
281           t.convert2string()), UVM_MEDIUM)
282         ...
283         seq_item_port.item_done();
284         ...
285       end
286       `uvm_info(get_type_name(), "Finishing...", UVM_MEDIUM)
287     endtask
```

Doug's Driver - +UVM_CONFIG_DB_TRACE

```
 uvm_config_db#(int)::set(this, "driver", "simple_int", simple_int+10);

# [CFGDB/SET] Configuration 'test.e1_b.driver.simple_int' (type int) set by
test.e1_b = ?
```

Config SET

```
 if( !uvm_config_db#(int)::get( this, "*", "simple_int", simple_int))
   `uvm_fatal("ENV", "simple_int not set")

# [CFGDB/GET] Configuration 'test.e1_b.driver.*.simple_int' (type int) read
by test.e1_b.driver = null (failed lookup)
```

Config GET

```
 if( !uvm_config_db#(int)::get( this, "", "simple_int", simple_int))
   `uvm_fatal("ENV", "simple_int not set")

# [CFGDB/GET] Configuration 'test.e1_b.driver.simple_int' (type int) read
by test.e1_b.driver = ?
```

Config GET

Doug's Sequence

```
158   class sequence_B_grandparent #(type T = int) extends
159       sequence_B_parent #(T);
160     `uvm_object_param_utils(sequence_B_grandparent#(T))
161
162     const static string type_name =
163       {"sequence_B_grandparent#(", T::type_name, ")"};
164     virtual function string get_type_name();
165       return type_name;
166     endfunction
        ...
169     sequence_B_parent #(T) seq;
170
171     function new(string name = "sequence_B_grandparent");
172       super.new(name);
173       `ALLOCATED_TYPE
          ...
175     endfunction
```

Doug's Sequence - body

```
178     `uvm_debug_sequence_body
179     task body();
180       `uvm_info(get_type_name(),
181         "Starting... ", UVM_MEDIUM)
182       `BP
183       for(int i = 0; i < 3; i++) begin
184         seq = new("p_seq"); //::create...
185         `SHOW_ACTIVE_SEQUENCES("Sequences")
186         seq.start(m_sequencer);
187       end
188       `uvm_info(get_type_name(),
190         "Finishing...", UVM_MEDIUM)
191     endtask
192     function void do_record(uvm_recorder recorder);
193       super.do_record(recorder);
194       `uvm_record_field("v_int", v_int);
195     endfunction
196   endclass
```

```
`define uvm_debug_sequence_body \
  \
  virtual task pre_body();  \
    `DEBUG("Doug's pre_body()") \
    super.pre_body();  \
    active_sequences::add(this);
  endtask \
  \
  virtual task post_body(); \
    `DEBUG("Doug's post_body()") \
    super.post_body(); \
    active_sequences::remove(this);
  endtask
```

```
`define SHOW_ACTIVE_SEQUENCES(ID) \
  active_sequences::show(this,
    {"Doug's ", ID},
    `__FILE__, `__LINE__);
```

```
`define BP if (BPP::check( get_full_name(), get_name(),
   get_type_name(), `__FILE__, `__LINE__)) $stop;
```

Doug's – Active Sequences

```
185          `SHOW_ACTIVE_SEQUENCES("Sequences")
```

```
# UVM_INFO envB.sv(186) @ 0: test.e2_b.sequencer@@ggp_seq [Doug's Sequences] Active Sequences
# UVM_INFO envB.sv(186) @ 0: test.e2_b.sequencer@@ggp_seq [Doug's Sequences]   test.e1_a.sequencer.seq
# UVM_INFO envB.sv(186) @ 0: test.e2_b.sequencer@@ggp_seq [Doug's Sequences]   test.e2_a.sequencer.ggp_seq
# UVM_INFO envB.sv(186) @ 0: test.e2_b.sequencer@@ggp_seq [Doug's Sequences]   test.e1_b.sequencer.ggp_seq
# UVM_INFO envB.sv(186) @ 0: test.e2_b.sequencer@@ggp_seq [Doug's Sequences]   test.e1_a.sequencer.ggp_seq
# UVM_INFO envB.sv(186) @ 0: test.e2_b.sequencer@@ggp_seq [Doug's Sequences]   test.e2_b.sequencer.sequence_B
# UVM_INFO envB.sv(186) @ 0: test.e2_b.sequencer@@ggp_seq [Doug's Sequences]   test.e2_a.sequencer.seq
# UVM_INFO envB.sv(186) @ 0: test.e2_b.sequencer@@ggp_seq [Doug's Sequences]   test.e1_b.sequencer.sequence_B
# UVM_INFO envB.sv(186) @ 0: test.e2_b.sequencer@@ggp_seq [Doug's Sequences]   test.e2_b.sequencer.ggp_seq
```

```
# UVM_INFO envB.sv(186) @ 0: test.e2_b.sequencer@@ggp_seq

...test.e2_b.sequencer.sequence_B_greatgrandparent#(sequence_item_A)
...test.e2_a.sequencer.seq
...test.e1_b.sequencer.sequence_B_greatgrandparent#(sequence_item_A)
...test.e2_b.sequencer.ggp_seq
```

Objects Allocated

```
`ifndef NO_DEBUG
`define ALLOCATED_TYPE() \
  objects::add(get_type_name());
`else
`define ALLOCATED_TYPE()
`endif
```

```
# Doug's Objects Allocated (by type)
#      324: sequence_item_A
#       80: sequence_A
#       80: sequence_B#(sequence_item_A)
#       26: sequence_B_parent#(sequence_item_A)
#       26: sequence_A_parent
#        6: sequence_B_grandparent#(sequence_item_A)
#        6: sequence_A_grandparent
#        4: driver2A
#        4: uvm_env
#        4: envA
#        2: sequence_B_greatgrandparent#(sequence_item_A)
#        2: driverA
#        2: driver2#(sequence_item_A)
#        2: sequence_A_greatgrandparent
#        1: test
```

```
objects::show();
```

Doug's – Breakpoints

- Breakpoint file named 'bp' contains two lines

```
+bp=gp_.*
+bp:type=.*greatgrand.*
```

- Run Simulation:

```
vsim -f bp
```

```
`define BP                  \
  if (BPP::check(           \
    get_full_name(),        \
    get_name(),             \
    get_type_name(),        \
    `__FILE__,              \
    `__LINE__)) $stop;
```

- Break when a certain "type_name" is hit (get_type_name())

```
+bp:type=
```

- Break when a certain "full name" is hit (get_full_name())

```
+bp:full=
```

- Break when a certain "name" is hit (get_name())

```
+bp=
```

Using the command line (UVM clp)

- UVM Command Line Processor (clp)

```
static function bit parse_args();
  string list[$];
  uvm_cmdline_processor clp;
  clp = uvm_cmdline_processor::get_inst();

  // By Type => +bp:type=
  void'(clp.get_arg_values("+bp:type=", list));
  $display("Doug's parsing breakpoints(by_type)=%p", list);
  foreach (list[i])
    BPP::set_by_type(list[i]);
  …
endfunction
```

UVM Data structures (the easy ones)

- Print Topology
  - `uvm_top.print_topology();`
- Print Factory
  - `factory.print(1);`
- Print Configuration
  - `uvm_top.check_config_usage(1);`
  - Shows configs with a 'write', but no 'read'.
- Show Connectivity
  - `show_connectivity(uvm_top, 0);`

UVM print_topology()

- uvm_top.print_topology();

```
# UVM_INFO @ 0: reporter [UVMTOP] UVM testbench topology:
# ----------------------------------------------------------------
# Name                          Type                        Size  Value
# ----------------------------------------------------------------
# test                          test                        -     @460
#   e1_a                        envA                        -     @480
#     driver                    driver2A                    -     @546
#       rsp_port                uvm_analysis_port           -     @561
#         recording_detail      integral                    32    'd1
#       sqr_pull_port           uvm_seq_item_pull_port      -     @553
#         recording_detail      integral                    32    'd1
#       recording_detail        integral                    32    'd1
#     sequencer                 uvm_sequencer               -     @569
#       rsp_export              uvm_analysis_export         -     @576
#         recording_detail      integral                    32    'd1
#       seq_item_export         uvm_seq_item_pull_imp       -     @670
#         recording_detail      integral                    32    'd1
#       recording_detail        integral                    32    'd1
#       arbitration_queue       array                       0     -
...
```

UVM Data structures (the easy ones)

- show_connectivity(uvm_top, 0);

```
# + UVM Root: Doug's uvm_top
#   + Test: test
#     + Env: test.e1_a
#       + Component: test.e1_a.driver
#          Port: test.e1_a.driver.rsp_port
#            Connected to Port: test.e1_a.sequencer.rsp_export
#          Port: test.e1_a.driver.sqr_pull_port
#            Connected to Port: test.e1_a.sequencer.seq_item_export
#       + Sequencer: test.e1_a.sequencer
#         + Component: test.e1_a.sequencer.req_fifo
#          Port: test.e1_a.sequencer.rsp_export
#            Connected to Port: test.e1_a.sequencer.sqr_rsp_analysis_fifo.analysis
#            Provided to Port: test.e1_a.driver.rsp_port
#          Port: test.e1_a.sequencer.seq_item_export
#            Provided to Port: test.e1_a.driver.sqr_pull_port
#         + Component: test.e1_a.sequencer.sqr_rsp_analysis_fifo
#            Port: test.e1_a.sequencer.sqr_rsp_analysis_fifo.analysis_export
#              Provided to Port: test.e1_a.sequencer.rsp_export
```

What do I need to do?

- Make transactions/classes "debug useful"
- Good names, good type names
- Use `uvm_info (ID and instance verbosity control)
- Instrument thread start/end/execution
  - Run_phase
  - Sequence body()
- Use `BP where you might want to $stop
- Use `ALLOCATED_TYPE where a class is constructed
- Use `SHOW_ACTIVE_SEQUENCES
  - Use active_sequences::add()/::remove()

What do I need to do?

```
import uvm_pkg::*;
`include "uvm_macros.svh"
`include "uvm_debug_macros.svh"
import uvm_debug_package::*;
```

Running simulation

```
vsim +bp=gp_.* +bp:type=.*greatgrand.* +d +debug=23
+UVM_VERBOSITY=UVM_LOW
+UVM_OBJECTION_TRACE
+UVM_CONFIG_DB_TRACE
+UVM_DUMP_CMDLINE_ARGS
+uvm_set_verbosity=test.e2_b.*,_ALL_,UVM_FULL,time,0
+uvm_set_verbosity=test.e2_b.driver*,_ALL_,UVM_FULL,time,0
+uvm_set_verbosity=test.*.driver*,_ALL_,UVM_FULL,time,0
+uvm_set_verbosity=test.*,sequence_B_grandparent#(sequence_item_A)
,UVM_FULL,time,0
+uvm_set_verbosity=test.*,sequence_A,UVM_FULL,time,0
+uvm_set_action=*,DBG,_ALL_,UVM_NO_ACTION
+uvm_set_config_int=*,recording_detail,1 … top
```

UVM Class library

- Paper lists about 100 other interesting UVM controls
  - Command line switches
  - Function calls to get or print information
  - Function calls to filter information

Summary

- The team got organized.
  - Did a little planning. Used SV LRM and UVM
- The team updated or changed their testbench to be "debug useful".
  - Class "content" – what's in the class
  - Flow debug – threads
  - Data structure debug (netlist, factory, config, …)
  - Breakpoint library – dynamic breakpoints
- Testbench debug went quickly.
- Ready to debug the RTL.
- Better Living…

Go see the Lorax Movie Friday