DVCon 2013
Design & Verification Conference & Exhibition

February 25-28, 2013
DoubleTree, San Jose

accellera
SYSTEMS INITIATIVE

You may add your company logo to this page, ONLY

Unconstrained UVM SystemVerilog Performance

Presented by
Wes Queen
Verifcation Manager
IBM

Randomization Performance is Important!

- Randomization is an integral part of UVM
  - It is used most often in configuring the environment and in writing sequences
- A typical UVM environment can easily contain tens of thousands of randomize calls.
  - Large environments can contain orders of magnitudes more.
- Writing high performance constraints is not hard, but requires engineers to think about it.
- Start thinking about randomization performance from the beginning of the project.

Set some reasonable goals for randomization times

- Understand how constraints are used in the UVM environment
  - Configuration constraints – Test level constraints
    - Only randomized a few times in a run
    - Medium to very high complexity
    - Can afford to run for longer times
  - Data Item constraints – data flow
    - Used many, many times in a run
    - Low to medium complexity
    - Needs to be optimized to run very fast. Small performance differences can greatly impact simulation run times

Coding for Speed

- Relationships between variables cause the solver to work harder. Remove unnecessary relationships.
- Constraint expressions are often invoked many times. Simplifying expressions lets the solver run faster.
  - Move complex math out of constraint code.

Note the lack of “rand”

```
class item extends uvm_sequence_item;
   rand int x_pos, y_pos;
   int y_offset, var_a, var_b. var_c;
   constraint position_c {
     x_pos ==( (12 * (y_pos + y_offset) *
               (var_a + var_b) ) / var_c) -
               ( (var_a * var_b) /var_c);
   }
endclass
```

Coding for Speed

- Use solve...before to simplify complex relationships
  - Constraints are bi-directional by default.
  - We often don’t expect them to be bi-directional.
  - Creating an order can greatly reduce the amount of work needed to solve.

```
class meal extends uvm_sequence_item;
  rand day_t weekday;
  rand meal_t lunch;
  constraint lunch_choice_c {
    (weekday == TUESDAY) -> (lunch != PIZZA);
    (weekday == WEDNESDAY ) -> (lunch == SOUP);

     solve weekday before lunch;
  }
endclass
```

Coding for Speed

- Understand the impact of arrays with foreach constraints
  - foreach constraints result a new constraint for each loop
    - A few lines of foreach code can result in many new constraints
  - Especially watch out for foreach constraints that result in creating new dependencies

Foreach better in procedural

```
class config extends uvm_sequence_item;
  rand int table[];
  constraint table_values {
    table.size() == 50;
    foreach( table[i] ) {
      foreach( table[j] ) {
        if( i != j )
          table[i] != table[j];
      }
    }
  }
endclass
```

Coding for Speed

- Use pre_randomize() and post_randomize() to assign values procedurally.
- Procedural code is most often less complex than similar looking constraint expressions
- Particularly useful in reducing the need for foreach constraints

foreach moved from constraint block from previous slide

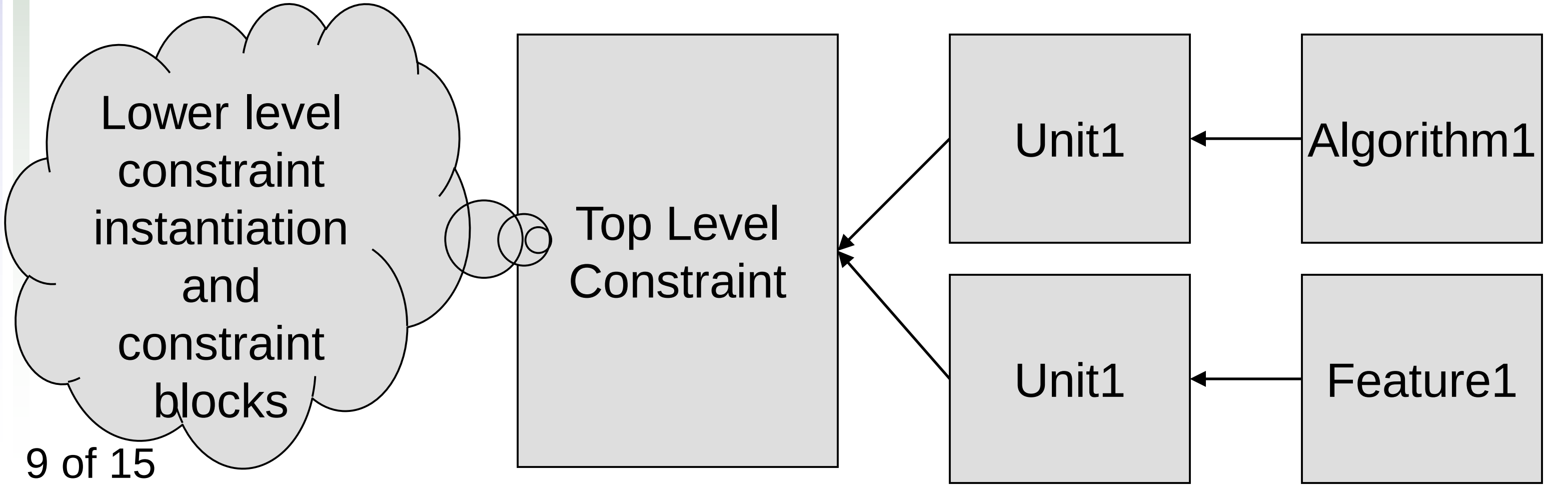
```
class config extends uvm_sequence_item;

  function void post_randomize();
     foreach( table[i] ) {
      foreach( table[j] ) {
        if( i != j )
          table[i] != table[j];
      }
  endfunction
endclass
```

Coding for memory usage

- Speed is not the only measure of randomization performance.
- Large memory usage can drive slow solver performance.
- Very large memory usage can push 32bit simulations to 64bit mode, resulting in a double performance impact.
- Particular trouble spot - look out for classes randomizing large tables simultaneously.

Coding for productivity

- Organize your classes and constraints for re-use & ease of maintenance. Break up into units and features or algorithms
  - Crucial for large classes with many complex constraints
- Separate variables from constraint code.
  - As UVC's evolve and grow, a clear separation makes it easier to upgrade and modify.



Coding for productivity

- Organize your constraints into multiple constraint blocks
  - A single large constraint block is harder for others to understand & debug.
  - Break constraints into blocks based on purpose.
  - This also makes it easier when classes are extended and constraint functionality is augmented or replaced.

```
class device_config extends uvm_object;
  rand mode_type mode;
  rand int max_size;

  constraint half_mode_defaults {
    if( mode == HALF ) { max_size == 1024; }
  }
  constraint full_mode_defaults {
    if( mode == FULL ) { max_size == 512; }
  }
endclass
```

Coding for productivity

- Use macros to replace repetitive code.
  - Simplification and readability

```
constraint_macro_example {
x inside {[-50:50]};
y inside {[-50:50]};
max_value inside {[ 0:100 ]};
if(x < 0 && y < 0 ) –x-y <= max_value;
if(x > 0 && y < 0 )  x-y <= max_value;
if(x < 0 && y > 0 )  y-x <= max_value;
if(x > 0 && y > 0 )  x+y <= max_value;
}
```

Same as above with simplification through macros

```
`define ABS(value) (((value) < 0) ? (-(value)) : (value))

constraint_macro_example {
x inside {[-50:50]};
y inside {[-50:50]};
max_value inside {[ 0:100 ]};
`ABS(x) + `ABS(y) <= max_value;
}
```

Coding for productivity

- Avoid modifying random fields manually after randomization
  - There is a temptation to hand modify the results of a randomize call to achieve a specific result.
  - It can be difficult for users other than the original developer to understand what is happening.
  - Limit modification of rand variables to constraint expressions, and pre_/post_randomize calls.
  - If it is difficult to achieve a specific result, consider restructuring the problem – perhaps breaking it into a number of smaller expressions/
  - *Let the constraint solver do the work for you!*

Coding for productivity

- Build stand alone environments to test classes with complex randomization.
- These environments can often be built in just a few minutes.
- They make it easier to:
  - Compile, build, and simulate
  - Prototype new code and try new experiments quickly
  - Run many more iterations than in a regular test
- These standalone environments can be placed in a regression suite to catch inadvertent errors caused by code changes.

Sponsored By:

Thank You!

- Developing fast, easy to work with randomization code in UVM environments is critical.
- By following simple rules, this is a straightforward process.

- Special thanks to the IBM Cores team for ideas, suggestions, and real world examples that have tested these concepts.