Fnob: Command-Line Dynamic Random Generator

Haoxiang Hu, Tuo Wang

Meta Infra ASIC
Intuition

• Say we have a random variable in TB:
  • Need different value of same random type?
  • Need different random type?
  • Need different value/type within the test on-the-fly?
  • Need different combination of random type/value?

• Any easy way to do it?
Before

When one variable and 8 testplan items

```
rand logic [11:0] src_id_x;
```

<table>
<thead>
<tr>
<th>Test 0</th>
<th>Src_id_x within [2:8]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>Src_id_x within [3:9]</td>
</tr>
<tr>
<td>Test 2</td>
<td>Src_id_x within [2:9]</td>
</tr>
<tr>
<td>Test 3</td>
<td>Src_id_x == 2</td>
</tr>
<tr>
<td>Test 4</td>
<td>Src_id_x within [3:8]</td>
</tr>
<tr>
<td>Test 5</td>
<td>Src_id_x 70%==2, 30%==3:9</td>
</tr>
<tr>
<td>Test 6</td>
<td>Src_id_x 70%==9, 30%==2:8</td>
</tr>
<tr>
<td>Test 7</td>
<td>Src_id_x == 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constraint 0</th>
<th>{src_id_x inside {[2:8]};}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint 1</td>
<td>{src_id_x inside {[3:9]};}</td>
</tr>
<tr>
<td>Constraint 2</td>
<td>{src_id_x inside {[2:9]};}</td>
</tr>
<tr>
<td>Constraint 3</td>
<td>{src_id_x == 2;}</td>
</tr>
<tr>
<td>Constraint 4</td>
<td>{src_id_x inside {[3:8]};}</td>
</tr>
<tr>
<td>Constraint 5</td>
<td>{src_id_x dist {2:=70, [3:9]:=30};}</td>
</tr>
<tr>
<td>Constraint 6</td>
<td>{src_id_x dist {9:=70, [2:8]:=30};}</td>
</tr>
<tr>
<td>Constraint 7</td>
<td>{src_id_x == 9;}</td>
</tr>
</tbody>
</table>

8 constraint blocks

8 SV tests

Re-compile
After When one variable and 8 testplan items

```plaintext
fnob_pkg::fnob src_id_x_fnob;
m_fnob_const = new("src_id_x_fnob ", FNOB_CONST, '{"val":{0}});
```

<table>
<thead>
<tr>
<th>Test</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 0</td>
<td>Src_id_x within [2:8]</td>
</tr>
<tr>
<td>Test 1</td>
<td>Src_id_x within [3:9]</td>
</tr>
<tr>
<td>Test 2</td>
<td>Src_id_x within [2:9]</td>
</tr>
<tr>
<td>Test 3</td>
<td>Src_id_x == 2</td>
</tr>
<tr>
<td>Test 4</td>
<td>Src_id_x within [3:8]</td>
</tr>
<tr>
<td>Test 5</td>
<td>Src_id_x 70%==2, 30%==3:9</td>
</tr>
<tr>
<td>Test 6</td>
<td>Src_id_x 70%==9, 30%==2:8</td>
</tr>
<tr>
<td>Test 7</td>
<td>Src_id_x == 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLI</th>
<th>Configuration String</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI 0</td>
<td><code>+uvm_set_config_string=\*,src_id_x_fnob,unif:2:8</code></td>
</tr>
<tr>
<td>CLI 1</td>
<td><code>+uvm_set_config_string=\*,src_id_x_fnob,unif:3:9</code></td>
</tr>
<tr>
<td>CLI 2</td>
<td><code>+uvm_set_config_string=\*,src_id_x_fnob,unif:2:9</code></td>
</tr>
<tr>
<td>CLI 3</td>
<td><code>+uvm_set_config_string=\*,src_id_x_fnob,constant:2</code></td>
</tr>
<tr>
<td>CLI 4</td>
<td><code>+uvm_set_config_string=\*,src_id_x_fnob,unif:3:8</code></td>
</tr>
<tr>
<td>CLI 5</td>
<td><code>+uvm_set_config_string=\*,src_id_x_fnob,dist:2:2:3:9_7:3</code></td>
</tr>
<tr>
<td>CLI 6</td>
<td><code>+uvm_set_config_string=\*,src_id_x_fnob,dist:9:9:2:8_7:3</code></td>
</tr>
<tr>
<td>CLI 7</td>
<td><code>+uvm_set_config_string=\*,src_id_x_fnob,constant:9</code></td>
</tr>
</tbody>
</table>
What’s Fnob

Random Number Generator
- In-line/CLI override
- Simpler & faster

Solution
- Built-in coverage
- Mix random

Developer/User Community
- Developer friendly
APIs

Declaration
fnob_pkg::fnob m_fnob_dist;

Instantiation
m_fnob_dist = new(<FNOB_NAME>,<FNOB_TYPE>,<FNOB_PARAM>);

Generation
m_fnob_dist.gen();

Override
• +uvm_set_config_string="*,m_fnob_dist, dist:0:100:1000_5:1"
• uvm_config_db#(string)::set(null,"*,m_fnob_dist", "unif:10:200");

Equivalent to:
m_fnob_dist dist {[10:25]:=2,
[150:160]:=6,
[0:5]:=6};
OVERRIDE
Two types of override

In-line Override

- UVM config_db based.
- Re-compile needed.
- Support multiple override per test.

Example:
```
uvm_config_db#(string)::set(null, "*", "m_fnob_unif", "unif:0:10");
```

Command Line Override (CLI)

- UVM config_db CLI based.
- No recompile needed.
- Override once per test.

Example:
```
+uvm_set_config_string="*,m_fnob_unif,unif:10:100"
```
Random Types

FNOB_CONST
constant value

FNOB_UNIF
uniform random value

FNOB_C_UNIF
cyclic uniform random value

FNOB_IN_LIST
value from defined list, equivalent to “inside”

FNOB_DIST
random value based on distribution; equivalent to “dist”

FNOB_PATN
value following fixed pattern as defined

FNOB_LOG
random value from logarithmic distribution

FNOB_C_PATN
cyclic value from fixed pattern

FNOB_NORM
random value from normal distribution

FNOB_INV_NORM
random value from inverse normal distribution

And more by developers ...
Fnob Coverage

Consecutive cover-group:
For Fnob types that has fixed length of params

```markdown
//-------------------------------------------------------------
covergroup fnob_cg;
  option.per_instance = 1;
  option.goal = 100;
  option.comment = "fnob_cg";

  fnob_rand_cg: coverpoint m_gen{
    bins val_min = {m_unif_min};
    bins val_max = {m_unif_max};
    bins val_mid = {m_unif_mid};
  }

eンドgroup // fnob_cg
```

Non-consecutive cover-group:
For Fnob types that has variable length of params

```markdown
covergroup fnob_pattern_cg(int val) with function sample(int cp);
  option.per_instance = 1;
  option.goal = 100;
  option.comment = "fnob_cg";

  fnob_rand_cg: coverpoint cp{
    bins val_all = {val};
  }

eンドgroup

fnob_pattern_cg cg[];

//-----------------------------------------------
function new(string name="", T params[]);
  super.new(name);
  m_vals = params;
  cg = new[m_vals.size()];

  for (int ii=0; ii<m_vals.size(); ii++) begin
g[n][ii] = new[m_vals[ii]];
  end

eンドfunction // new
```
We have single type of random per variable:

- Uniform random
- Constant
- Distribution
- Pattern

What if we want to mix them within the same test?

Example: 70% uniformly fast; 30% constant slow;

One variable: my_fnob

m_delay_fnob = fnob#(bit[63:0])::new_multi("m_delay_fnob","(dist:0:0:1:1_7:3)(unif:5:20)(constant:100)")
Flow Reduction

- Reduce extra coding;
- Save compile time;
- Speed-up test dev;
Performance Improvement

• “Constraint” traverse entire value space by size of data type;

• “Fnob” traverse limited value space by pre-defined system function;

<table>
<thead>
<tr>
<th>Random Keyword</th>
<th>Number of Call</th>
<th>Constraint: vendor 1</th>
<th>Fnob: vendor 1</th>
<th>Constraint: vendor 2</th>
<th>Fnob: vendor 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>inside</td>
<td>100,000,000</td>
<td>127s</td>
<td>5s</td>
<td>117s</td>
<td>1s</td>
</tr>
<tr>
<td>dist</td>
<td>100,000,000</td>
<td>69s</td>
<td>4s</td>
<td>119s</td>
<td>2s</td>
</tr>
<tr>
<td>constant</td>
<td>100,000,000</td>
<td>63s</td>
<td>4s</td>
<td>116s</td>
<td>3s</td>
</tr>
<tr>
<td>normal</td>
<td>100,000,000</td>
<td>68s</td>
<td>8s</td>
<td>118s</td>
<td>2s</td>
</tr>
</tbody>
</table>
Fnob Developer Interface

Fnob_pkg.sv

Fnob.sv:
1. Instantiation;
2. Argument parsing;
3. Random type selection;
4. Config_db override;

Fnob_rand_multi.sv:
1. Randomly select rand type

Fnob_common.sv:
1. Random type enum;
2. Enum-related translations;

Fnob_rand.sv:
1. All supported random type implementation (FNOB_UNIF, FNOB_C_UNIF, FNOB_CONST, FNOB_PATTERN, FNON_C_PATTERN ...);

Fnob_test.sv:
1. Standalone testbench for newly added random type

User Layer

Developer Layer
Questions