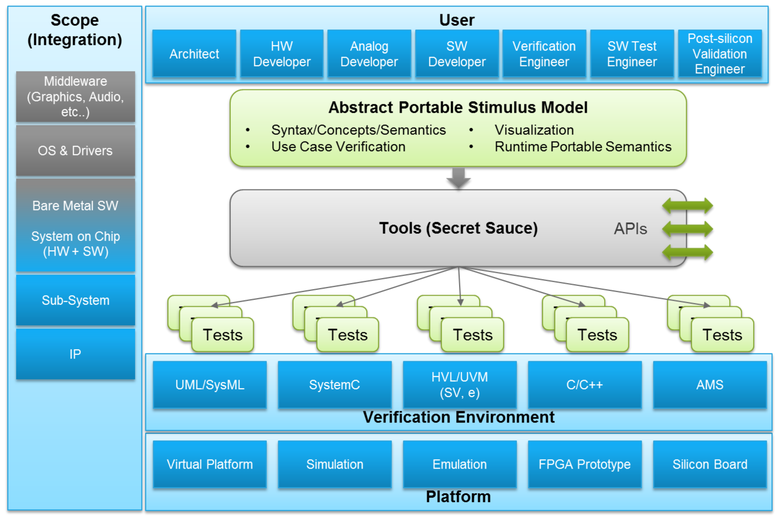
Jump-Start Portable Stimulus Test Creation with SystemVerilog Reuse

Matthew Ballance
matt_ballance@mentor.com

Portable Stimulus Specification

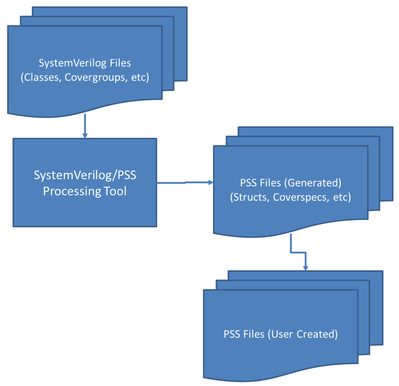


Single specification of test intent
Usable across disciplines
Re-targetable across verification environments and platforms

Why Reuse SystemVerilog

Reuse between UVM and SW-Driven environments desirable
Transactions, configurations already captured as constraints
Already verified base on which to build a PSS description

SV Reuse Flow



Import declarative SystemVerilog description elements
Class variables, constraints
Covergroup coverpoints, crosses
Focus on fully-automatable flow

Guidelines: Structuring SystemVerilog for Reuse

1. Avoid use of inline constraints
2. Avoid procedurally enabling/disabling constraints
3. Avoid constraint references to class-external variables
4. Specify covergroups in terms of a single class

Avoid Inline Constraints

Inline constraints procedural construct, not declarative
Usually contain references to procedural-code variables
→ Refactor as unique classes

```
class transaction extends uvm_sequence_item;
  `uvm_object_utils(transaction)
  rand bit[31:0]  addr;
  rand bit[31:0]  data;
  rand bit[1:0]   size;
  rand bit        rnw;
endclass

class rw_test_sequence extends uvm_sequence;
  `uvm_object_utils(rw_test_sequence)

  task body();
    transaction t1 = transaction::type_id::create("t1");
    transaction t2 = transaction::type_id::create("t2");
    for (int unsigned offset=0; offset<'h1000; offset+=4) begin
      // Do a write
      start_item(t1);
      assert(t1.randomize() with { // Size and data left open
        t1.addr == 'h8000+offset;
        t1.rnw == 0;
      });
      finish_item(t1);
```

```
class read_transaction extends transaction;
  `uvm_object_utils(read_transaction)

  constraint read_c { rnw == 1; }
endclass

class write_transaction extends
transaction;
  `uvm_object_utils(write_transaction)

  constraint write_c { rnw == 0; }
endclass
```

Avoid Procedurally Enabling/Disabling Constraints

Use of constraint_mode() prevents independent evaluation
→ Refactor as classes with additive constraints
→ Refactor as classes with subtractive constraints

```
typedef enum { MODE_NORMAL_1, MODE_NORMAL_2, MODE_EXT_1, MODE_EXT_2} mode_t;

class op_item extends uvm_sequence_item;
  `uvm_object_utils(op_item)

  rand mode_t          mode;
  // ... Other fields
  constraint normal_mode_c {
    mode inside {MODE_NORMAL_1, MODE_NORMAL_2};
  }
  constraint ext_mode_c {
    mode inside {MODE_EXT_1, MODE_EXT_2};
  }
endclass
```

```
typedef enum { MODE_NORMAL_1, MODE_NORMAL_2, MODE_EXT_1, MODE_EXT_2} mode_t;

class op_item extends uvm_sequence_item;
  `uvm_object_utils(op_item)

  rand mode_t          mode;
  // ... Other fields
endclass

class normal_op_item extends op_item;
  `uvm_object_utils(normal_op_item)

  constraint normal_mode_c {
    mode inside {MODE_NORMAL_1, MODE_NORMAL_2};
  }
endclass

class ext_op_item extends op_item;
  `uvm_object_utils(ext_op_item)

  constraint ext_mode_c {
    mode inside {MODE_EXT_1, MODE_EXT_2};
  }
endclass
```

```
typedef enum { MODE_NORMAL_1, MODE_NORMAL_2, MODE_EXT_1, MODE_EXT_2} mode_t;

class normal_op_item extends uvm_sequence_item;
  `uvm_object_utils(op_item)

  rand mode_t          mode;
  // ... Other fields
  constraint normal_mode_c {
    mode inside {MODE_NORMAL_1, MODE_NORMAL_2};
  }
endclass

class ext_op_item extends normal_op_item;
  `uvm_object_utils(ext_op_item)

  constraint normal_mode_c {
    // empty constraint - follows reuse guideline #2
  }

  constraint ext_mode_c {
    mode inside {MODE_EXT_1, MODE_EXT_2};
  }
endclass
```

Avoid Constraint References to Class-External Variables

Class-external references make classes non-modular
→ Convert class-external to class-internal references

```
package global_data_pkg;
  bit[31:0]           coeff_max;
  bit[31:0]           coeff_min;

endpackage

class transform_item extends uvm_sequence_item;
  `uvm_object_utils(transform_item)

  rand bit[31:0]        coeff;

  constraint coeff_c {
    coeff >= global_data_pkg::coeff_min;
    coeff <= global_data_pkg::coeff_max;
  }
endclass
```

```
class transform_item extends uvm_sequence_item;
  `uvm_object_utils(transform_item)

  bit[31:0]             coeff_min;
  bit[31:0]             coeff_max;

  rand bit[31:0]        coeff;

  constraint coeff_c {
    coeff >= coeff_min;
    coeff <= coeff_max;
  }
endclass
```

Specify Covergroups in terms of a Single Class

Specify each covergroup in terms of a single class
Makes specification more modular
Makes coverage goals more reusable, less env-specific

```
class config_item extends uvm_object;
  `uvm_object_utils(config_item)

  rand bit[3:0]           model;
  rand bit[15:0]          model_coeff;
  rand bit[3:0]           mode2;
  rand bit[15:0]          mode2_coeff;

endclass

covergroup config_cg(ref config_item item);

  model_cp : coverpoint (item.model) {
    bins model[] = {[0:15]};
  }

  model_coeff_cp : coverpoint (item.model_coeff) {
    bins model_coeff_min = {0};
    bins model_coeff_mid[14] = {[1:'hfffe]};
    bins model_coeff_max = {'hffff};
  }

  model_coeff_cross : cross model_cp, model_coeff_cp;

  mode2_cp : coverpoint (item.mode2) {
    bins mode2[] = {[0:15]};
  }

  mode2_coeff_cp : coverpoint (item.mode2_coeff) {
    bins mode2_coeff_min = {0};
    bins mode2_coeff_mid[14] = {[1:'hfffe]};
    bins mode2_coeff_max = {'hffff};
  }

  mode2_coeff_cross : cross mode2_cp, mode2_coeff_cp;

endgroup
```