Boosting SystemC-based Testbenches with Modern C++ and Coverage-Driven Generation

Hoang M. Le, Rolf Drechsler

University of Bremen, Germany

accellera SYSTEMS INITIATIVE

Universität Bremen

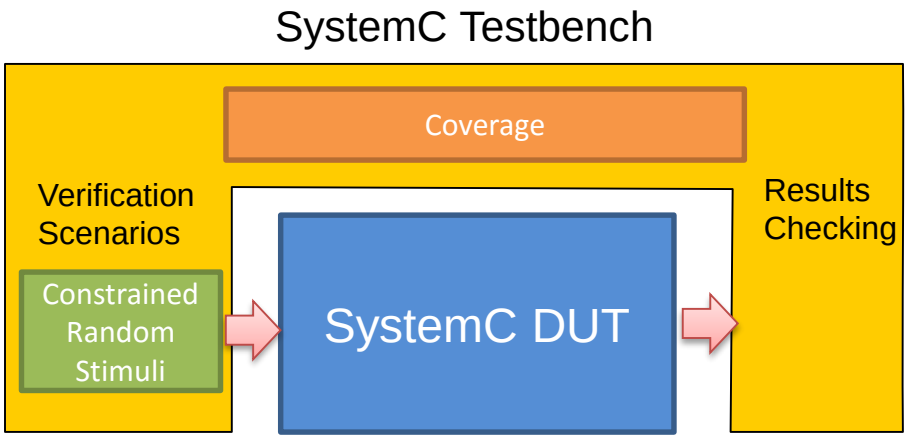
2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

- Motivation
- State of the Art (for SystemC)
- Modern C++11 API incl. Coverage
- Coverage-driven Generation
- Wrap-up

Motivation

- SystemC
  - IEEE 1666-2011
  - C++ modeling @ multiple levels of abstraction
  - Open-source reference simulator
- Verification of SystemC models
  - Constrained Random Verification Methodology (e.g. UVM for SystemC)
  - Randomization (CRAVE or SCV)
  - Coverage?
  - Automated Coverage Closure?



State of the Art (for SystemC)

- Randomization (CRAVE 2.0)
  - SystemVerilog-inspired syntax
  - Random objects
  - Random variables
  - Hard/soft/distribution constraints
  - Efficient constraint solving
    - SMT backends
    - Constraint partitioning

```
struct irqmp_regs : public rand_obj {
  randv<unsigned> level_reg;
  randv<unsigned> force_reg;

  irqmp_regs(rand_obj* parent_obj)
      : level_reg(this), force_reg(this) {
    constraint(level_reg() < (1 << 16));
    constraint((level_reg() & 1) == 0);
    constraint((force_reg() & 0xFFFF0001) == 0);
  }
};
```

  - Still somewhat cumbersome syntax

State of the Art (for SystemC)

- Coverage (SCVX or SVM)
  - SystemVerilog-inspired syntax
  - Covergroups
  - Coverpoints
  - Coverbins
  - Not expressive enough for complex coverage conditions
  - No automated coverage closure

```
class cg: public scvx::scvx_covergroup
{
 public:
 scvx::scvx_coverpoint cp_m;
 scvx::scvx_coverpoint cp_n;
 cg( scvx::scvx_name name, int& m, int& n)
  : cp_m( "cp_m", m ),
    cp_n( "cp_n", n )
  {
    option.auto_bin_max = 16;
    cp_m.bins("bin_a") =
      scvx::list_of(4,  0, 1, 2, 3 );
    cp_m.bins("bin_b", scvx::SINGLE_BIN) =
      scvx::list_of(4,  4, 5, 6, 7 );
    cp_m.ignore_bins("ignore_bins") = 6;
    cp_n.ignore_bins("ignore_bins") = 13;
  }
```

```
// define a new covergroup type and create an instance
CG* cg_one = pFac->new_covergroup(this, "CG_1", "CG_1_inst");

// create a new coverpoint
CP* cp_one = pFac->new_coverpoint(cg_one, "CP_1_SystemC")

// Multiple overlays for connect method, here
// connect a sc_signal<int> to this coverpoint
cp_one->connect(Addr);

// Declare a bin for with intervals 20:30 and 50:60
pFac->new_bins(cp_one,"20to30&50to60",1,4,20,30,50,60);

// Declare an ignore bin with range 50-80
pFac->new_ignore_bins(cp_one,"Ign50-80",1,2,50,80);

// Declare an illegal bin for value 99
pFac->new_illegal_bins(cp_one,"99_BIN",AUTOBINS,2,99,99);

// Add all other ranges of Addr to a default bin
pFac->add_default_bins("DEFAULT_BIN");
```

Our proposed solution

- Using C++11 for more compact constraint syntax

- New coverage API connected to the expression layer of CRAVE 2.0
  - For better expressiveness (same as of the constraint layer)
  - Allow automated coverage closure / coverage-driven generation (coverage expression handled as a special type of constraint in the constraint solving process)

Modern C++11 API incl. Coverage

- Constraint layer
- Automated creation of named object hierarchy
- C++11 allows in-place initialization

```
struct irqmp_regs : public crv_sequence_item {
  crv_variable<unsigned> level_reg { "level_reg" };
  crv_variable<unsigned> force_reg { "force_reg" };

  crv_constraint level_reg_cstr { "level_reg_cstr",
    level_reg() < (1 << 16),
    (level_reg() & 1) == 0
  };

  crv_constraint force_reg_cstr { "force_reg_cstr",
    (force_reg() & 0xFFFF0001) == 0
  };

  irqmp_regs(crv_object_name) { }
};
```

```
struct irqmp_regs : public crv_sequence_item {
  CRV_VARIABLE(unsigned, level_reg);
  CRV_VARIBALE(unsigned, force_reg);

  CRV_CONSTRAINT(level_reg_cstr,
    level_reg() < (1 << 16),
    (level_reg() & 1) == 0
  );
  CRV_CONSTRAINT(force_reg_cstr,
    (force_reg() & 0xFFFF0001) == 0
  );

  irqmp_regs(crv_object_name) { }
};
```

Modern C++11 API incl. Coverage

- Coverage layer
- Example (verification of a SystemC TLM interrupt controller)

```
struct my_covergroup : public crv_covergroup {
  crv_variable<unsigned> lr { "lr" };
  crv_variable<unsigned> fr { "fr" };

  crv_coverpoint fwd_lvl_1 { "fwd_lvl_1" };
  crv_coverpoint fwd_lvl_0 { "fwd_lvl_0" };

  expression forced_lvl_1() { return make_expression(fr() &  lr()); }
  expression forced_lvl_0() { return make_expression(fr() & ~lr()); }

  my_covergroup(crv_object_name) {
    for (int k = 1; k < 16; k++) {
      fwd_lvl_1.bins( (((forced_lvl_1() >> k) & 1) == 1) && (forced_lvl_1() < (2 << k)) );
      fwd_lvl_0.bins( (((forced_lvl_0() >> k) & 1) == 1) && (forced_lvl_0() < (2 << k)) && (forced_lvl_1() == 0) );
    }
  }
};
```

Coverage-driven Generation

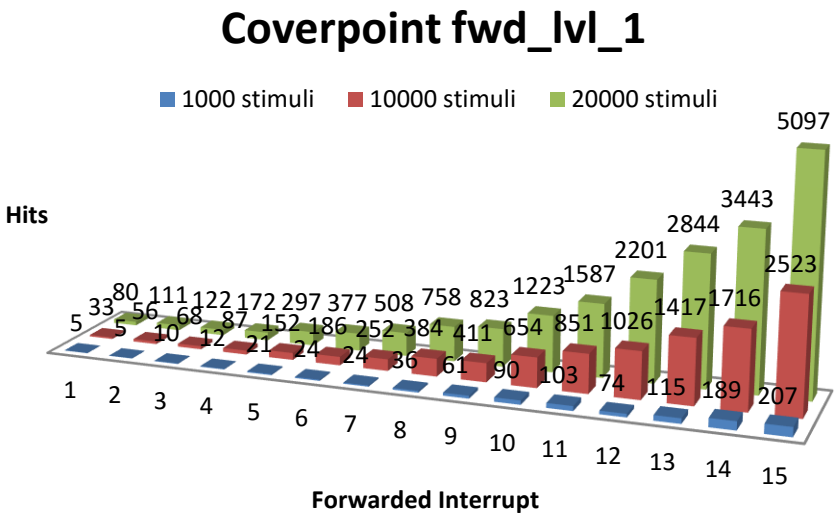
- Coverage sampling

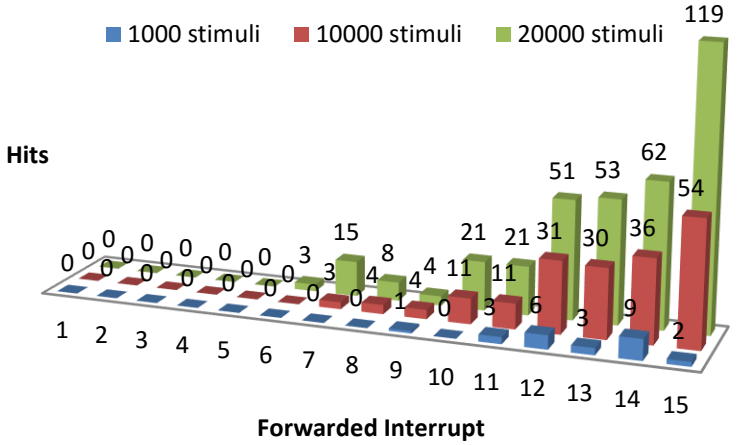
```
irqmp_regs regs("regs");

my_covergroup cg("cg");
cg.lr.bind(regs.level_reg);
cg.fr.bind(regs.force_reg);

assert(regs.randomize());
cg.sample();
```

- Results without CdG





Coverage-driven Generation

- Manual coverage closure
  - Add more (distribution) constraints?
  - Add more directed tests?
  - Both highly non-trivial
- "Push-button" when constraint and coverage are connected!
  - Convert coverage expressions to constraint expressions
  - In each generation pass
    - Pick one unsatisfied "coverage" constraint (i.e. uncovered bin), add to the constraint set, then apply constraint solving
    - If successful, marked the "coverage" constraint as satisfied, otherwise pick another unsatisfied and repeat

Wrap-up

- CRAVE
  - Open-source constrained random stimuli generator for SystemC/C++
  - Powerful constraint solving technologies
  - Frequently added new features, recently: modern C++11 API, coverage layer & coverage-driven generation

- What‘s next?
  - Graph-based specification / Portable Stimulus
  - UVM-SystemC Integration

Acknowledgement

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project EffektiV under contract no. 01IS13022E and by the German Research Foundation (DFG) within the Reinhart Koselleck project DR 287/23-1.

Thanks for your time!
Questions?

accellera SYSTEMS INITIATIVE

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE