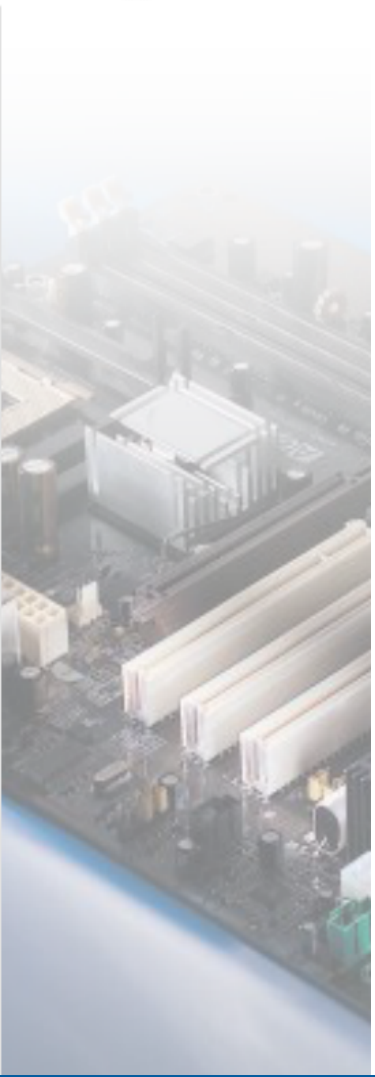


An Advanced Constrained Random Verification Environment for SystemC

Daniel Große, Finn Haedicke,
Hoang M. Le, Rolf Drechsler

Group of Computer Architecture, University of Bremen, Germany

Outline

1. Motivation
2. Constraint Specification
   - Random Variable
   - Random Object
   - Constraint Inheritance
3. Dynamic Constraints
   - Dynamic Data Structures
   - Constraint Management
   - References in Constraints
4. Inline Generators usage
5. Parallel Constraint Solving
6. Summary

Motivation

- ESL design to handle complexity (abstraction!)

Motivation

- ESL design to handle complexity (abstraction!)
- Widely accepted ESL language: SystemC

Motivation

- ESL design to handle complexity (abstraction!)
- Widely accepted ESL language: SystemC
- Verification dictates design costs

Motivation

- ESL design to handle complexity (abstraction!)
- Widely accepted ESL language: SystemC
- Verification dictates design costs
- Manage verification effort by **testbench automation**
  $\Rightarrow$ Constrained Random Verification (CRV)

Motivation

- ESL design to handle complexity (abstraction!)
- Widely accepted ESL language: SystemC
- Verification dictates design costs
- Manage verification effort by **testbench automation**
  $\Rightarrow$ Constrained Random Verification (CRV)
  - CRV for SystemC through SCV library

Motivation

- ESL design to handle complexity (abstraction!)
- Widely accepted ESL language: SystemC
- Verification dictates design costs
- Manage verification effort by **testbench automation**
  $\Rightarrow$ Constrained Random Verification (CRV)
  - CRV for SystemC through SCV library
  - SCV has several weaknesses

Motivation

- ESL design to handle complexity (abstraction!)
- Widely accepted ESL language: SystemC
- Verification dictates design costs
- Manage verification effort by **testbench automation**
  $\Rightarrow$ Constrained Random Verification (CRV)
  - CRV for SystemC through SCV library
  - SCV has several weaknesses
  - Development of Advanced CRV Environment

Random Variable

```
randv<T> x;
```

- `x`: variable of built-in or SystemC type `T`

Random Variable

```
randv<T> x;
```

- `x`: variable of built-in or SystemC type `T`
- Simple constraints:

  ```
  addRange(l, r)
  addWeightedRange(l,r,w)
  ```

Random Variable

```
randv<T> x;
```

- `x`: variable of built-in or SystemC type `T`
- Simple constraints:

  ```
  addRange(l, r)
  addWeightedRange(l,r,w)
  ```

- `x()`: symbolic variable used in complex constraints

Random Variable

```
randv<T> x;
```

- `x`: variable of built-in or SystemC type `T`
- Simple constraints:
  `addRange(l, r)`
  `addWeightedRange(l,r,w)`
- `x()`: symbolic variable used in complex constraints
- C++ operators available for `x` and `x()` (+,-,*,/,%, ==, !=, <,>, >=, <=, <<, >>, &&, ||, !, ~, &, |, ^)

Random Variable

`randv<T> x;`

- `x`: variable of built-in or SystemC type `T`
- Simple constraints:
  `addRange(l, r)`
  `addWeightedRange(l,r,w)`
- `x()`: symbolic variable used in complex constraints
- C++ operators available for `x` and `x()` (+,-,*,/,%, ==, !=, <,>, >=, <=, <<, >>, &&, ||, !, ~, &, |, ^)

```
randv<int> x;

// 30% in [0, 9]
x.addWeightedRange(0, 9, 30);

// 70% in [90, 99]
x.addWeightedRange(90, 99, 70);

// get value
x.next();
```

Random Object

`rand_obj`

- a random object extends `rand_obj`, contains `randv`s and other `rand_obj`s.

Random Object

`rand_obj`

- a random object extends `rand_obj`, contains `randv`s and other `rand_obj`s.
- constraint specification in constructor/later via API call

Random Object

`rand_obj`

- a random object extends `rand_obj`, contains `randv`s and other `rand_obj`s.
- constraint specification in constructor/later via API call
- `constraint([name,] expr)`: [named] hard constraint

Random Object

`rand_obj`

- a random object extends `rand_obj`, contains `randv`s and other `rand_obj`s.
- constraint specification in constructor/later via API call
- `constraint([name,] expr)`: [named] hard constraint
- `soft_constraint(expr)`: ignorable constraint

Random Object

`rand_obj`

- a random object extends `rand_obj`, contains `randv`s and other `rand_obj`s.
- constraint specification in constructor/later via API call
- `constraint([name,] expr)`: [named] hard constraint
- `soft_constraint(expr)`: ignorable constraint
- `next()` invokes constraint solver, returns `true` on success

Random Object

`rand_obj`

- a random object extends `rand_obj`, contains `randv`s and other `rand_obj`s.
- constraint specification in constructor/later via API call
- `constraint([name,] expr)`: [named] hard constraint
- `soft_constraint(expr)`: ignorable constraint
- `next()` invokes constraint solver, returns `true` on success

```
struct packet : public rand_obj {

  randv<unsigned> src;
  randv< sc_uint<16> > dest;

  packet() : ... {
    constraint(src() <= 0xFFFF);

    constraint("diff",
               src() != dest());

    soft_constraint(
               dest() % 4 == 0);
  }

}
```

Constraint Inheritance

- realized by C++ inheritance

Constraint Inheritance

- realized by C++ inheritance
- inherit everything of the base class

Constraint Inheritance

- realized by C++ inheritance
- inherit everything of the base class
- allows to add more random variables/objects and constraints to the base class

Constraint Inheritance

- realized by C++ inheritance
- inherit everything of the base class
- allows to add more random variables/objects and constraints to the base class

```
struct packet1 : public packet {
  randv<char> data;

  packet1() : ... {
    constraint('a' <= data() && data()
        <= 'z');
    constraint(dest_addr() % 2 == 1);
  }
}
```

Vector Constraints

```
rand_vec<T> v;
```

- no support in SCV, mimic via fixed-size arrays (inconvenient and not always possible)

Vector Constraints

```
rand_vec<T> v;
```

- no support in SCV, mimic via fixed-size arrays (inconvenient and not always possible)
- `v` mimics a STL vector `std::vector<T>` (`T` built-in type)

Vector Constraints

```
rand_vec<T> v;
```

- no support in SCV, mimic via fixed-size arrays (inconvenient and not always possible)
- `v` mimics a STL vector `std::vector<T>` (`T` built-in type)
- `v()`: symbolic vector used in constraints

Vector Constraints

```
rand_vec<T> v;
```

- no support in SCV, mimic via fixed-size arrays (inconvenient and not always possible)
- `v` mimics a STL vector `std::vector<T>` (`T` built-in type)
- `v()`: symbolic vector used in constraints
- `v().size()`: size of symbolic vector

Vector Constraints

```
rand_vec<T> v;
```

- no support in SCV, mimic via fixed-size arrays (inconvenient and not always possible)
- `v` mimics a STL vector `std::vector<T>` (`T` built-in type)
- `v()`: symbolic vector used in constraints
- `v().size()`: size of symbolic vector
- `v()[_i]`: symbolic element (`_i` predefined constant)

Vector Constraints

`rand_vec<T> v;`

- no support in SCV, mimic via fixed-size arrays (inconvenient and not always possible)
- `v` mimics a STL vector `std::vector<T>` (`T` built-in type)
- `v()`: symbolic vector used in constraints
- `v().size()`: size of symbolic vector
- `v()[_i]`: symbolic element (`_i` predefined constant)
- `v()[_i-c]`: previous element relative to `v()[_i]` (`c > 0`)

Vector Constraints

```
rand_vec<T> v;
```

- no support in SCV, mimic via fixed-size arrays (inconvenient and not always possible)
- `v` mimics a STL vector `std::vector<T>` (`T` built-in type)
- `v()`: symbolic vector used in constraints
- `v().size()`: size of symbolic vector
- `v()[_i]`: symbolic element (`_i` predefined constant)
- `v()[_i-c]`: previous element relative to `v()[_i]` (`c > 0`)
- `constraint.foreach` and `constraint.soft_foreach` to build constraints over the elements

Vector Constraints - Example

```
struct packet2 : public packet {
  rand_vec<char> data;

  packet2() : ... {
    constraint(data().size() % 4 == 0);
    constraint(data().size() < 100);

    // data[0] upper case
    constraint.foreach(data, _i,
      IF_THEN(_i == 0, 'A' <= data()[_i] && data()[_i] <= 'Z'));
    // the rest lower case
    constraint.foreach(data, _i,
      IF_THEN(_i != 0, 'a' <= data()[_i] && data()[_i] <= 'z'));
    // forbid aa, ab and ba
    constraint.soft_foreach(data, _i,
      data()[_i] + data()[_i-1] > 'a' + 'b');
  }
};
```

Constraint Management

- Enable/disable specific constraints during verification process

Constraint Management

- Enable/disable specific constraints during verification process
- Not available in SCV, mimic via auxiliary variables and implication constraints (inconvenient and inefficient)

Constraint Management

- Enable/disable specific constraints during verification process
- Not available in SCV, mimic via auxiliary variables and implication constraints (inconvenient and inefficient)
- Supported via named constraints `constraint(name, expr).`

Constraint Management

- Enable/disable specific constraints during verification process
- Not available in SCV, mimic via auxiliary variables and implication constraints (inconvenient and inefficient)
- Supported via named constraints `constraint(name, expr)`.
- API: `enable_constraint(name)` and `disable_constraint(name)` of `rand_obj`

Constraint Management

- Enable/disable specific constraints during verification process
- Not available in SCV, mimic via auxiliary variables and implication constraints (inconvenient and inefficient)
- Supported via named constraints `constraint(name, expr).`
- API: `enable_constraint(name)` and `disable_constraint(name)` of `rand_obj`
- Disabled constraints have no effect in the randomization via `next()` until enabled again

References

- Randomization depends on dynamically changing environment state

References

- Randomization depends on dynamically changing environment state
- In SCV inefficient and tedious (additional variables, manual update)

References

- Randomization depends on dynamically changing environment state
- In SCV inefficient and tedious (additional variables, manual update)
- Here via `reference(x)`: link a C++ variable `x` to a symbolic variable used in constraints

References

- Randomization depends on dynamically changing environment state
- In SCV inefficient and tedious (additional variables, manual update)
- Here via `reference(x)`: link a C++ variable `x` to a symbolic variable used in constraints
- Each call to *next()* uses actual value of `x`

References

- Randomization depends on dynamically changing environment state
- In SCV inefficient and tedious (additional variables, manual update)
- Here via `reference(x)`: link a C++ variable `x` to a symbolic variable used in constraints
- Each call to *next()* uses actual value of `x`

```
struct packet3 : public packet {
 rand_vec<char> data;

 packet3(int &expected_max_size) : ... {
  constraint(data().size() % 4 == 0);

  constraint(data().size() <=
             reference(expected_max_size));
  // data[0] upper case
  ...
 }
};
```

Inline Generators

- Constraint specification without `random_obj`

Inline Generators

- Constraint specification without `random_obj`
- Based on Generator object

Inline Generators

- Constraint specification without `random_obj`
- Based on Generator object
- Usable anywhere independent of other constraints

Inline Generators

- Constraint specification without `random_obj`
- Based on Generator object
- Usable anywhere independent of other constraints
- All features except for inheritance

Inline Generators

- Constraint specification without `random_obj`
- Based on Generator object
- Usable anywhere independent of other constraints
- All features except for inheritance

```
randv<int> x,y;
Generator gen;

gen( x() < y() );
gen( x() > 100 || y() < 50);

if (gen.next())
   run_test(x,y);
```

Incremental Generator usage

- Incrementally add new constraints

Incremental Generator usage

- Incrementally add new constraints
- Previous constraints stay valid

Incremental Generator usage

- Incrementally add new constraints
- Previous constraints stay valid
- Trigger specific behavior after generic behavior executed

```
randv<int> x,y;
Generator gen;
gen(x != y)
for (int i =0 ; i < n; ++i) {
  gen.next();
  run_test(x,y);
}

gen( x*x == y );
for (...) {
  gen.next(); run_test(x,y) ;
}

gen( y%2 == 0 );
for (...) {
  gen.next(); run_test(x,y) ;
}
```

Parallel Constraint Solving

- SCV: Limits in constraint complexity due to BDD-based constraint-solving

Parallel Constraint Solving

- SCV: Limits in constraint complexity due to BDD-based constraint-solving
- Alternatives: SAT/SMT

Parallel Constraint Solving

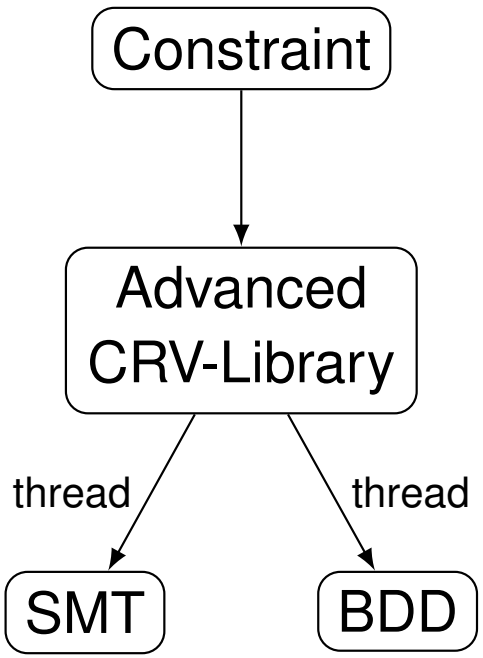
- SCV: Limits in constraint complexity due to BDD-based constraint-solving
- Alternatives: SAT/SMT
- But: BDD guarantees uniform distribution, hard for SAT/SMT

Parallel Constraint Solving

- SCV: Limits in constraint complexity due to BDD-based constraint-solving
- Alternatives: SAT/SMT
- But: BDD guarantees uniform distribution, hard for SAT/SMT
- Here: Portfolio approach

Parallel Constraint Solving

- SCV: Limits in constraint complexity due to BDD-based constraint-solving
- Alternatives: SAT/SMT
- But: BDD guarantees uniform distribution, hard for SAT/SMT
- Here: Portfolio approach
  - Multi-threaded
  - SMT provides fast solutions
  - BDD provides uniform distribution
  - use BDD once created



Parallel Constraint Solving (2)

- Comparison to SCV
- Focus: How fast can the solutions be generated?
- Example: Constraint inputs of ALU such that output is computed w/o overflow or divide-by-zero exceptions

Parallel Constraint Solving (2)

- Comparison to SCV
- Focus: How fast can the solutions be generated?
- Example: Constraint inputs of ALU such that output is computed w/o overflow or divide-by-zero exceptions

```
randv< sc_bv<2> >  op;
randv< sc_uint<16> > a;
randv< sc_uint<16> > b;

constraint(
  op()!=0 || 65535 >= a() + b() );

constraint(
  op()!=1 || (65535 >= a() - b() &&
              b() <= a() )  );

constraint(
  op()!=2 ||  65535 >= a() * b()  );

constraint( op()!=3 || b() != 0 );
```

Parallel Constraint Solving (2)

- Comparison to SCV
- Focus: How fast can the solutions be generated?
- Example: Constraint inputs of ALU such that output is computed w/o overflow or divide-by-zero exceptions
- run-time in sections for first/all stimuli
- SCV 1.0e 32 bit vs. new approach

		ALU4	ALU12	ALU16	ALU24	ALU32
SCV	first	< 0.01	13.77	MO	TO	TO
	finished	0.09	19.84	MO	TO	TO
New	first	< 0.01	< 0.01	0.01	0.01	0.01
	finished	0.14	0.30	0.37	0.40	0.49

Summary and Future Work

- Advanced Constraint Random Verification Environment
- Natural C++ Syntax, easy to use API
- Dynamic constraints/ constraint management
- Parallel constraint solving with advanced decision engines

Summary and Future Work

- Advanced Constraint Random Verification Environment
- Natural C++ Syntax, easy to use API
- Dynamic constraints/ constraint management
- Parallel constraint solving with advanced decision engines
- Extended support for dynamic data structures
- SMT distribution quality
- Automated constraint debugging

Summary and Future Work

- Advanced Constraint Random Verification Environment
- Natural C++ Syntax, easy to use API
- Dynamic constraints/ constraint management
- Parallel constraint solving with advanced decision engines
- Extended support for dynamic data structures
- SMT distribution quality
- Automated constraint debugging

Thank you for your attention.

References I

[1] OSCI, “SystemC,” 2011, available at http://www.systemc.org.

[2] *IEEE Standard SystemC Language Reference Manual*, IEEE Std. 1666, 2005.

[3] J. Yuan, C. Pixley, and A. Aziz, *Constraint-based Verification*. Springer, 2006.

[4] *SystemC Verification Standard Specification Version 1.0e*, SystemC Verification Working Group, http://www.systemc.org, 2003.

[5] J. Rose and S. Swan, *SCV Randomization Version 1.0*, 2003.

[6] C. N. Ip and S. Swan, “A tutorial introduction on the new SystemC verification standard,” www.systemc.org, White Paper, 2003.

[7] F. Haedicke, S. Frehse, G. Fey, D. Große, and R. Drechsler, “metaSMT: Focus on your application not on solver integration,” in *DIFTS’11: 1st International workshop on design and implementation of formal tools and systems*, 2011.

[8] D. Große and R. Drechsler, *Quality-Driven SystemC Design*. Springer, 2010.

[9] D. Große, R. Ebendt, and R. Drechsler, “Improvements for constraint solving in the SystemC verification library,” in *ACM Great Lakes Symposium on VLSI*, 2007, pp. 493–496.

References II

[10] D. Große, R. Wille, R. Siegmund, and R. Drechsler, “Debugging contradictory constraints in constraint-based random simulation,” in *Languages for Embedded Systems and their Applications: Selected Contributions on Specification, Design, and Verification from FDL’08*, M. Radetzki, Ed. Springer, 2009, pp. 273–290.

[11] R. Wille, D. Große, F. Haedicke, and R. Drechsler, “SMT-based stimuli generation in the SystemC verification library,” in *Advances in Design Methods from Modeling Languages for Embedded Systems and SoC’s: Selected Contributions on Specification, Design, and Verification from FDL 2009*, D. Borrione, Ed. Springer, 2010, pp. 227–244.