2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Engineered SystemVerilog Constraints

Jeremy Ridgeway

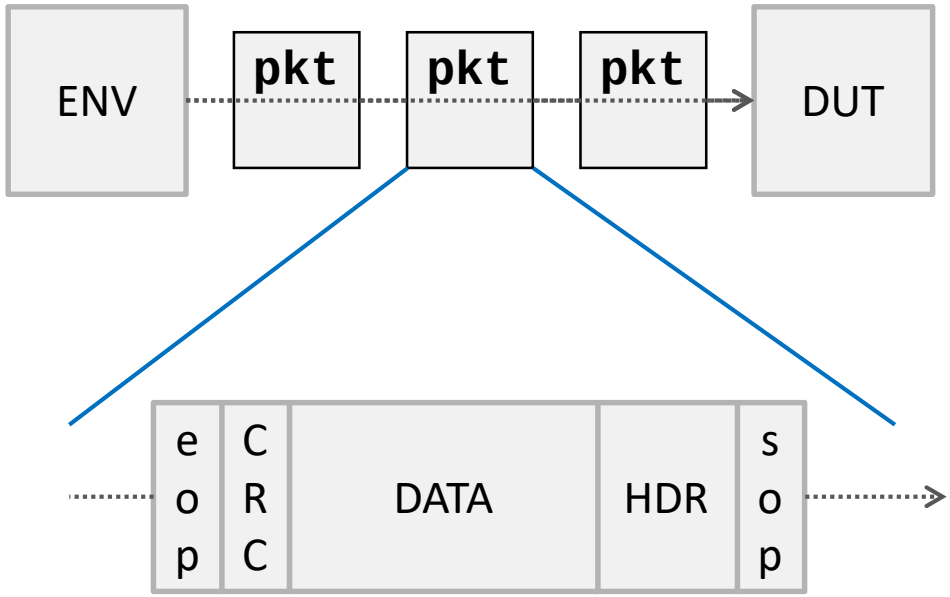
Avago Technologies, Inc.

Agenda

- Example Constraints
- Constraint Solver
- Engineering Constraints

Example Testbench

- Packet based device
  - Start of Packet
  - Header fields
  - Data to send
  - CRC
  - End of Packet
- Length 0 – 512 bytes



Stimulus Packet Class

- Encapsulate the packet in a class
- Focus only on length

```
class base_packet;
  rand sop_t sop;
  rand header_t hdr;
  rand byte data[];
  rand int len;
  crc_t crc;
  rand eop_t eop;




endclass
```

Valid-type Constraint

- Length 0 – 512 bytes
- Use class inheritance to focus randomization

```
class base_packet;
  rand byte data[];
  rand int len;
  constraint valid {
    len inside { [0:512] };




endclass
```

Coverage-based Constraints

- Length 0 – 512 bytes
- Determine what values are important to cover
- Bin important values

- Regress & Rely on random

```
class base_packet;
  rand byte data[];
  rand int len;
  constraint imp {
    len inside {
      0, [1:511], 512 };
  };
  covergroup imp_cg;
    coverpoint len {
      bins a = { 0 };
      bins b = { [1:511] };
      bins c = { 512 };
    }
  endgroup
endclass
```

Regress & Rely on Random

- How many tests must execute to hit coverage goals?
  - 10? 1,000? 10,000?
- Does the number change between regressions?
  - Today 1,321. Tomorrow 1,999?
- Knowing what we **need** to cover, can we do better?

len **inside** { 0, [1:511], 512 }

len != 0

Declarative Constraints

- Cannot generate constraints on-the-fly:

```
constraint c { len inside { 0, [1:511], 512 }; }
constraint d { len != 0; }
```

- d can not be instantiated on-the-fly **kind of …**
- d can be enabled on-the-fly:

```
task update_constraints();
  if(len == 0) d.constraint_mode(1);
  …
endtask
```

Agenda

- Example Constraints
- Constraint Solver
- Engineering Constraints

Constraint Formula

constraint c { len **inside** { 0, [1:511], 512 }; }

- Convert to Boolean:

(len == 0) || (len >= 1) || (len <= 511) || (len == 512)

- Identify Unique Components:
  - Variable: len
  - Predicate: a Boolean quantity (arg1 op arg2)
  - Literal: a predicate or its negation
  - Clause: logical OR of literals
  - Formula (in CNF): logical AND of clauses

(len <= 511) == !(len > 511)
Predicate: p == (len > 511)
(Negative) Literal: !p

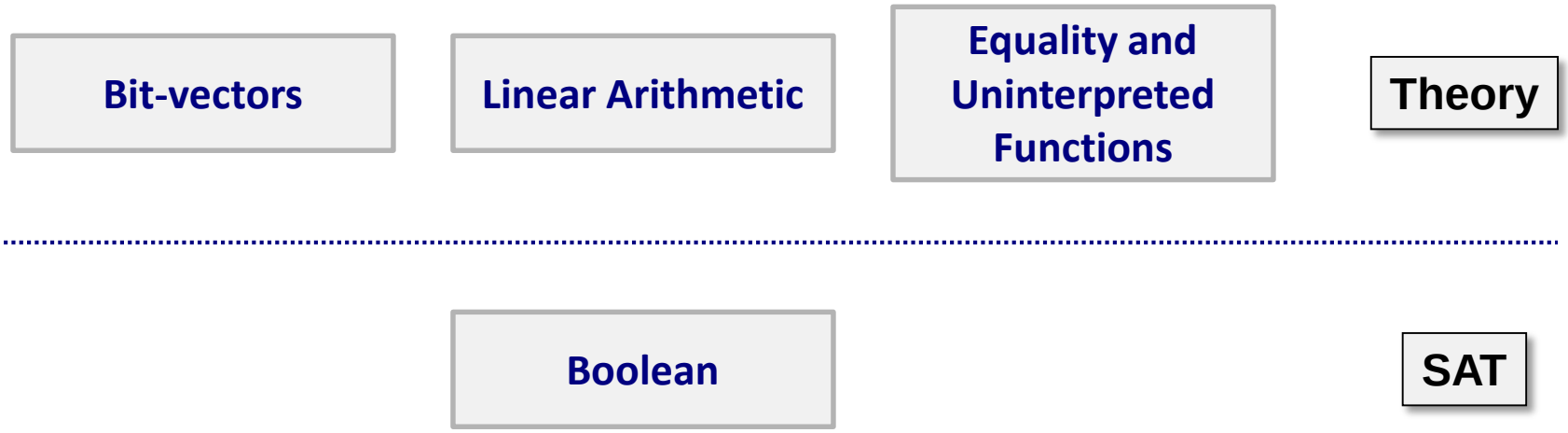
Conjunctive Normal Form:
Logial AND of all Logical ORs

Constraint Solver

- Determines if a formula is satisfiable **sat == true**
- Returns an assignment on variables otherwise **unsat**

((len == 0) || (len >= 1) || (len <= 511) || (len == 512)) ↔ **false**

- Layered and incremental solver



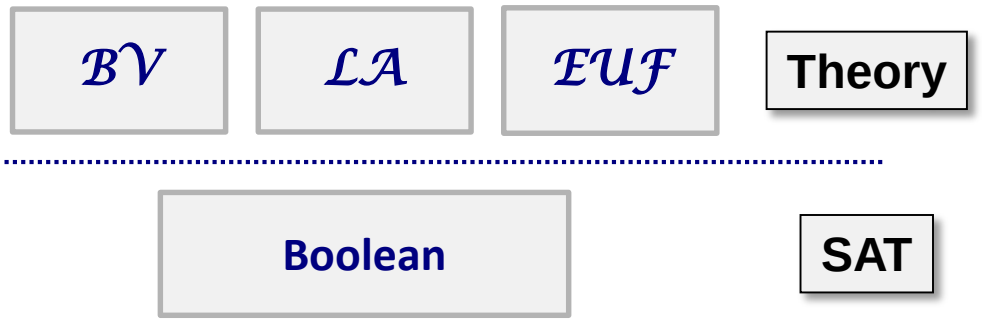
Incremental Solving

(len == 0) || (len >= 1) || (len <= 511) || (len == 512)

a b c d

A || B || C || D

- SAT Solver assignment: A==1, D==1
  - len cannot be both 0 and 512
  - Must be rectified by theory solver



$\mathcal{T}(\mathcal{LA})$ Adds Clauses

(len == 0) || (len >= 1) || (len <= 511) || (len == 512)

A || B || C || D

- Instantiate new constraints:

$$A \leftrightarrow !D$$

- Simplify to CNF Clauses:

(!A || !D) && (D || A)

- Add to the formula and solve again

Backtrack and Solve Again

((len == 0) || (len >= 1) || (len <= 511) || (len == 512))

A B && C D

!(len == 0) || !(len == 512) && (len == 0) || (len == 512)

!A !D A D

- Re-used predicates and literals

  (A || B || C || D) && (!A || !D) && (A || D)

- Formula cannot satisfy when A == 1, D == 1

- Let's stop right here!

Agenda

- Example Constraints
- Constraint Solver
- Engineering Constraints

Return to the Ideal

- Want to generate **constraint** d on-the-fly:

  constraint c { len **inside** { 0, [1:511], 512 }; }

  constraint d { len != 0; }

- Represent constraint as CNF formula
  - Clauses
    - Positive or Negative Literals
      - Predicates

Assume everything is an int data type

- Decompose constraint into predicates and literals

Engineered Predicates

- Boolean Quantity
- Result flag
- Unique instances in formula

len == 0

- Extend to implement quantity type

Assume there's a constraint value container class

```
class constraint_base;
```

```
class constraint_pred;
  rand bit result;
endclass
```

```
class constraint_binary_op;
  rand constraint_val lhs;
  rand constraint_val rhs;
endclass
```

```
class pred_equality;
  constraint valid {
    if(result) lhs.val == rhs.val;
    else       lhs.val != rhs.val;
  }
endclass
```

Engineered Literals

- Result flag
- Negation flag
- Predicate
- At most **2** literals need be instantiated for a predicate

```
class constraint_base;
```

```
class constraint_pred;
  rand bit result;
endclass
```

```
class constraint_literal;
  const bit neg;
  rand constraint_pred pred;
  constraint valid {
    result -> pred.result == !neg;
  }
endclass
```

(len <= 511) == !(len > 511)
Predicate: p == (len > 511)
(Negative) Literal: !p

Implication allows literals to be ignored

Engineered Clauses

- Queue of Literals
- Array reduction implements the logical OR

A || B || C || D

```
class constraint_base;
```

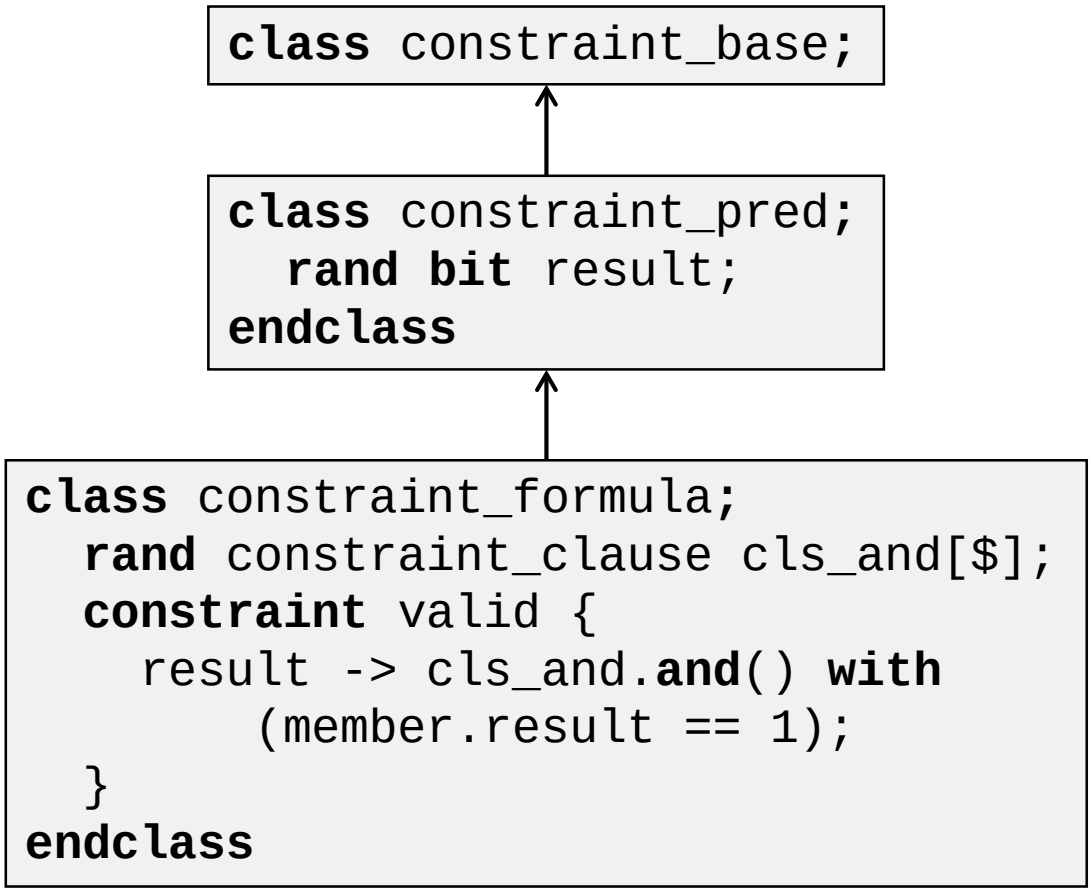
```
class constraint_pred;
  rand bit result;
endclass
```

```
class constraint_clause;
  rand constraint_literal lit_or[$];
  constraint valid {
    result -> lit_or.or() with
        (member.result == 1);
  }
endclass
```

Engineered Formula

- Queue of Clauses
- Array reduction implements the logical AND

(A || B || C || D)
&&
(!A || !D)
&&
(A || D)



Engineered Values

- Contain a single value
- Can nest constraints

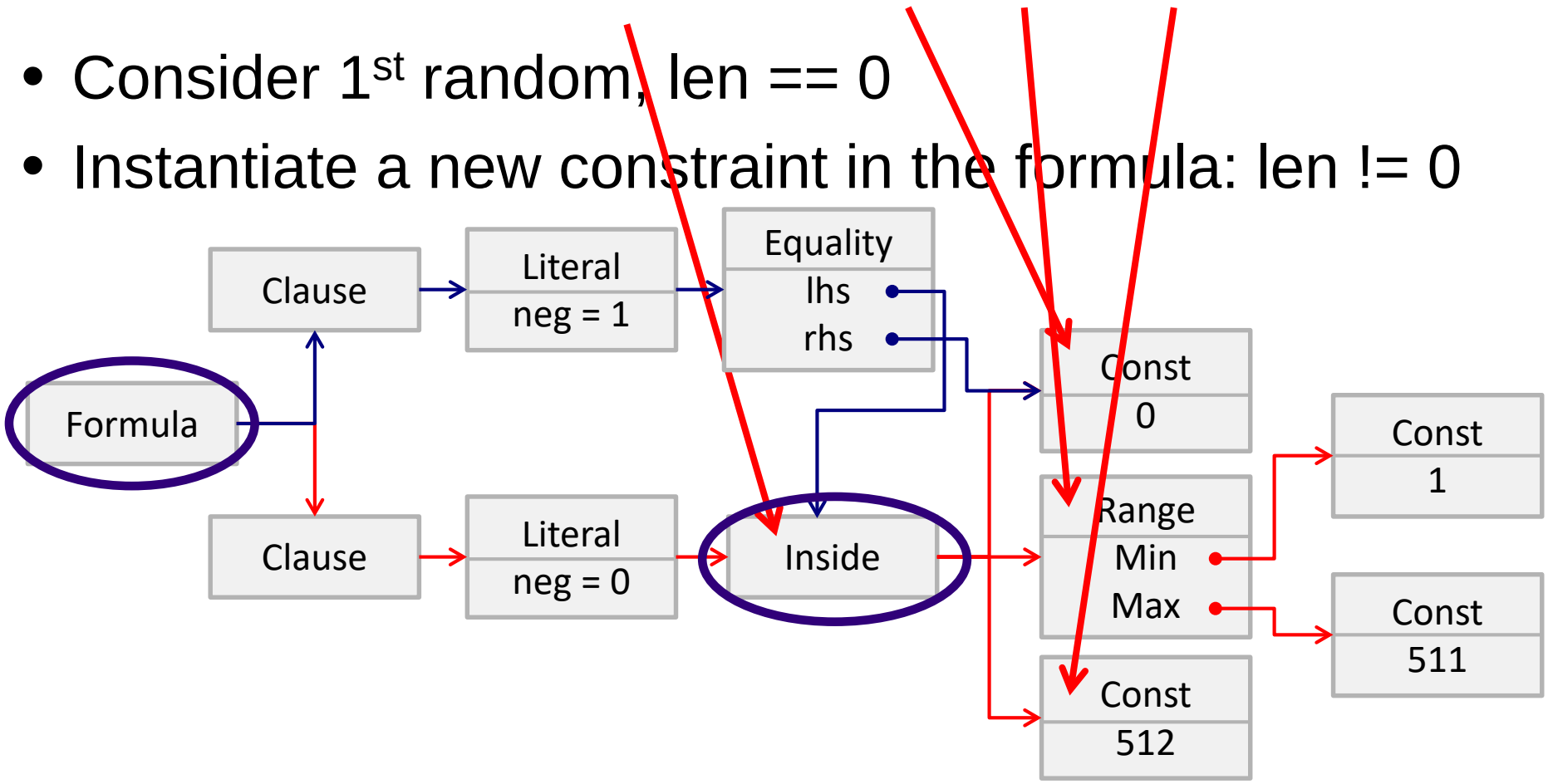
Refer to the paper!

Type	Example
Value	len
Inside	len **inside** {0, [1:511], 512}
Distribution	len **dist** { 0 := 10, [1:511] := 20, 512 := 10 }
Sequence	len **seq** [0, 512, **dist** {0 := 10, [1:512] := 50}]
Constant	512

Engineered Formula

constraint c { len **inside** { 0, [1:511], 512 }; }

- Consider 1st random, len == 0
- Instantiate a new constraint in the formula: len != 0



Engineered Random Variable

- Random value
- Constrained value
- Formula
- Constraint Factory

(Almost) all classes are type-parameterized

```
class loc_rand_base;
```

```
class loc_rand_param#(T);
```

```
class loc_rand_static#(T);
  rand T value;
  constraint_pred#(T) val;
  constraint_pred#(T) formula;
  static constraint_factory#(T) factory;
  constraint valid {
    solve formula.result before value;
    formula.result == 1;
    value == val.value; }
endclass
```

```
class loc_rand#(T);
```

Agenda

- Example Constraints
- Constraint Solver
- Engineering Constraints

Thank You!

Questions?