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

```plaintext
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

```plaintext
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

```plaintext
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?

```python
len inside { 0, [1:511], 512 }
len != 0
```
Declarative Constraints

• Cannot generate constraints on-the-fly:

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

• d cannot be instantiated on-the-fly

• d can be enabled on-the-fly:

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

• Example Constraints
• Constraint Solver
• Engineering Constraints
Constraint Formula

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

- Convert to Boolean:
  
  \[(\text{len} == 0) || (\text{len} >= 1) || (\text{len} <= 511) || (\text{len} == 512)\]

- 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

Conjunctive Normal Form:
Logical AND of all Logical ORs

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

- Determines if a formula is satisfiable

- Returns an assignment on variables otherwise

- Layered and incremental solver

\[
\text{(len == 0) || (len >= 1) || (len <= 511) || (len == 512) } \iff \text{false}
\]

- SAT
- Theory
- Equality and Uninterpreted Functions
- Linear Arithmetic
- Bit-vectors
- Boolean
Incremental Solving

\[(\text{len} == 0) \lor (\text{len} >= 1) \lor (\text{len} <= 511) \lor (\text{len} == 512)\]

\[
\begin{align*}
\text{a} & \quad \text{b} & \quad \text{c} & \quad \text{d} \\
A \lor B \lor C \lor D
\end{align*}
\]

- SAT Solver assignment: A==1, D==1
  - len cannot be both 0 and 512
  - Must be rectified by theory solver
\( T(\text{LA}) \) Adds Clauses

\[
(len == 0) \lor (len >= 1) \lor (len <= 511) \lor (len == 512) \\
A \lor B \lor C \lor D
\]

- Instantiate new constraints:

\[
A \leftrightarrow !D
\]

- Simplify to CNF Clauses:

\[
(!A \lor !D) \land (D \lor A)
\]

- Add to the formula and solve again
Backtrack and Solve Again

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

• 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

• Decompose constraint into predicates and literals

Assume everything is an int data type
Engineered Predicates

- Boolean Quantity
- Result flag
- Unique instances in formula
- 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

\[(\text{len} \leq 511) \equiv \neg(\text{len} > 511)\]

(Predicate: \(p \equiv (\text{len} > 511)\) (Negative) Literal: \(!p\)
Engineered Clauses

- Queue of Literals
- Array reduction implements the logical OR

```
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

A || B || C || D
```
Engineered Formula

• Queue of Clauses
• Array reduction implements the logical AND

\[(A \lor B \lor C \lor D) \land (\neg A \lor \neg D) \land (A \lor D)\]
Engineered Values

- Contain a single value
- Can nest constraints

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>len</td>
</tr>
<tr>
<td>Inside</td>
<td>len inside {0, [1:511], 512}</td>
</tr>
<tr>
<td>Distribution</td>
<td>len dist { 0 := 10, [1:511] := 20, 512 := 10 }</td>
</tr>
<tr>
<td>Sequence</td>
<td>len seq [ 0, 512, dist {0 := 10, [1:512] := 50} ]</td>
</tr>
<tr>
<td>Constant</td>
<td>512</td>
</tr>
</tbody>
</table>
Engineered Formula

```plaintext
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

\[
\text{class } \text{loc_rand_base};
\]

\[
\text{class } \text{loc_rand_param}\langle\text{T}\rangle;
\]

\[
\text{class } \text{loc_rand_static}\langle\text{T}\rangle;
\]

\[
\begin{align*}
\text{rand } \text{T} & \text{ value;} \\
\text{constraint_pred}\langle\text{T} \rangle & \text{ val;} \\
\text{constraint_pred}\langle\text{T} \rangle & \text{ formula;} \\
\text{static } \text{constraint_factory}\langle\text{T}\rangle & \text{ factory;} \\
\text{constraint } \text{valid } \{ \\
\text{solve } \text{formula.result before value;} \\
\text{formula.result} & \text{ == 1;} \\
\text{value} & \text{ == val.value;} \}
\end{align*}
\]

\[
\text{endclass}
\]

\[
\text{class } \text{loc_rand}\langle\text{T}\rangle;
\]

(Almost) all classes are type-parameterized
Agenda

- Example Constraints
- Constraint Solver
- Engineering Constraints
Thank You!

Questions?