Attacking Constraint Complexity in Large Scale
Random Verification Environment

Ben Chen
Srinath Atluri
Harish Krishnamoorthy

Alex Wakefield
Balamurugan Veluchamy
Agenda

- Challenges
- Case Study 1: Performance and capacity
- Case Study 2: Constraint inheritance
- Conclusion
## ASICs Snapshot

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical ASIC Gate Count</td>
<td>5M – 25M</td>
</tr>
<tr>
<td>Target Technologies</td>
<td>45 – 65nm</td>
</tr>
<tr>
<td>Design Cycle</td>
<td>9mo – 18mo</td>
</tr>
<tr>
<td>Verification Cycle</td>
<td>6mo – 9mo</td>
</tr>
<tr>
<td>Post Silicon</td>
<td>~ 3mo</td>
</tr>
<tr>
<td>Constrained Random</td>
<td>3000 – 5000 random variables</td>
</tr>
<tr>
<td>Environment</td>
<td>10000 – 30000 lines of constraints</td>
</tr>
</tbody>
</table>
Challenges

- Write smart constraints
- Optimize constraints for performance
- Manage hierarchical constraints
- Work around restrictions in tools and languages
Constraint Failures!

```plaintext
constraint seg_data_user_constraint // (from start_seg[0].my_next_segment[0].my_next_segment[0].my_next_segment[0]) (constraint_mode = ON)
{
}
constraint required // (from start_seg[0].my_next_segment[0].my_next_segment[0].my_next_segment[0]) (constraint_mode = ON)
{
  ( start_seg[0].my_next_segment[0].my_next_segment[0].my_next_segment[0].data_length <=
    16'b0111111111111111 ) ;
  ( start_seg[0].my_next_segment[0].my_next_segment[0].my_next_segment[0].data_length >=
    16'b0000000000000000 ) ;
}
constraint legal_segment_length // (from start_seg[0].my_next_segment[0].my_next_segment[0].my_next_segment[0].my_next_segment[0]) (constraint_mode = ON)
{
  ( start_seg[0].my_next_segment[0].my_next_segment[0].my_next_segment[0].my_next_segment[0].segment_length == 16'b0000000000000000 ) ;
}
Solver failed when solving following set of constraints
rand bit[15:0] start_seg[0].local_segment_length; // rand_mode = ON
rand bit[15:0] start_seg[0].next_segment.segment_length; // rand_mode = ON
rand bit[5:0] start_seg[0].my_next_segment[0].tci; // rand_mode = ON
rand bit[7:0] start_seg[0].my_next_segment[0].short_len; // rand_mode = ON
rand bit[15:0] start_seg[0].my_next_segment[0].next_segment.segment_length; // rand_mode = ON
rand bit[15:0] start_seg[0].my_next_segment[0].local_segment_length; // rand_mode = ON
constraint legal_frame_length // (from this) (constraint_mode = ON)
{
  ( cts_frame => ( start_seg.segment_length == ( ( frame_length - pad_length ) - 16 ) ) ) ;
}
constraint WITH_CONSTRAINT // (from this) (constraint_mode = ON)
{
  ( frame_length == 16'b0000000000000001 ) ;
}
=======================================================================
Constraint solver failed - Constraints are inconsistent and cannot be solved.
```
Case study I: Random Packet Generator

- **Background** – Random packet generator for a flexible packet parser

- **Requirements**
  - Randomize the order of network protocol headers
  - Constrain the position of one or more headers

**Dynamic Packet Configuration:**

<table>
<thead>
<tr>
<th>HDR</th>
<th>L2</th>
<th>L3</th>
<th>DATA</th>
</tr>
</thead>
</table>

Any internal hdrs …
Ethernet, Dot1Q, SAP …
IPV4, IPV6 …
Case study I: Constrained array shuffling

- Identify problem

```c
int hdr [\`SIZE];
hdr = {\`DOT1Q, \`ETHERNET, \`SECURITY, ...};

hdr.shuffle() with {
    hdr[0] == \`ETHERNET;
};
```

- Applications
  
  Network packet parser verification
  Random CPU instruction set generation
Case study I – Nested foreach loop

- Simple constraint code
- Poor solver performance

```plaintext
class packet;
    rand int hdr[`SIZE];

    constraint cst {
        foreach (hdr[i]) {
            hdr[i] inside {
                `DOT1Q,
                `ETHERNET,
                `SECURITY, ...
            };
            foreach (hdr[j]) {
                (i != j) -> hdr[i] != hdr[j];
            }
        }
        hdr[0] == `ETHERNET;
    }
endclass;
```
Solution Space: Matrix vs. Network

- Matrix: faster randomization
- Network: larger capacity

```class packet;
  rand bit [1:0] a, b;
  constraint c1 {
    a>=2 -> b==3;
  }
endclass```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td>10%</td>
<td>10%</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>10%</td>
<td>10%</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>10%</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>10%</td>
</tr>
</tbody>
</table>

```class packet;
  rand bit [3:0] a, b, c;
  constraint cst2 { a > b; }
  constraint cst3 { c == a + b; }
endclass```

```bit [3:0] a```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
Case study I – 1\textsuperscript{st} Optimization

- 2x speed-up, but still exponential

```plaintext
class packet;
    rand int hdr[`SIZE);

    constraint cst {
        foreach (hdr[i]) {
            hdr[i] inside {
                `DOT1Q,
                `ETHERNET,
                `SECURIY, ...};
            foreach (hdr[j]) {
                (i != j) -> hdr[i] != hdr[j];
                (i < j) -> hdr[i] != hdr[j];
            }
        }
        hdr[0] == `ETHERNET;
    }
endclass;
```
Case study I – Final solution

```plaintext
class packet;
    header_l2 hdr[`SIZE];
    function void pre_randomize();
        foreach (hdr[i])
            hdr[i].randomize() with
            {index == i;};
    endfunction
endclass

class packet;
    header_l2 hdr[`SIZE];
    function void pre_randomize();
        foreach (hdr[i])
            hdr[i].randomize() with
            {index == i;};
    endfunction
endclass

class header;
    static int exist_hdrs ['$'];
    rand int value, index;

    constraint seg_id_val_constr {
        !(value inside exist_hdrs);
    }

    function void pre_randomize();
        exist_hdrs = '{};
    endfunction

    function void post_randomize();
        exist_hdrs.push_back(value);
    endfunction
endclass

class header_l2 extends header;
    constraint legal_seg_id_values {
        value inside {`ETH, `SAP, ...};
    }

    constraint eth_position_0 {
        (index == 0)-> (value == `ETH);
    }
endclass
```
Case study I – Solver Performance

- Bit Slicing

```plaintext
rand bit [15:0] pkt_len;
rand bit [7:0] pkt_len_lsb8;

constraint pkt_len_cnstr {
    pkt_len[7:0] == 0;
    pkt_len[7:0] == pkt_len_lsb8;
    pkt_len_lsb8 == 0;
    pkt_len > 64 && pkt_len < 9000;
}
```

- Multiplication/Division

```plaintext
rand bit [32:0] A, B, C;

constraint factorization {
    A * B == C;
    A > B;
    solve A, B before C;
}
```

- Performance diagnostics

```plaintext
csh% ./simv +ntb_enable_solver_diagnostics=<N>
+ntb_solver_diagnostics_filename=<filename>
```
Case study II – Constraint Inheritance

Across test-bench components

Extended Class (Tesetcase)
 pkt_len == 64

Base Class (pkt library)
 pkt_len >= 32 && pkt_len <= 1512
Case study II – Inheritance Behavior

- Background – Legacy random environment

Problem – Difference in constraint inheritance behavior across languages
  - Between OpenVera to SV
  - Between “e” and SV
Case study II – “e” Soft Constraints

- Performs UNION of constraints
- Disabled only if a conflict occurs (no solver failure)

```c
struct packet {
    rand int data_len;
    keep soft data_len in [32:1512];
}

struct short_packet extends packet {
    keep data_len < 64;
}

struct long_packet extends packet{
    keep data_len > 64;
}

struct err_packet extends packet {
    keep data_len in [1513:2000];
}
```
Case study II - OpenVera default constraints

- Always overridden by another non-default constraint
  Defeats purpose of inheritance

```java
class packet {
    rand int len;
    default constraint valid_len {
        len inside {[32:1512]};
    }
}

class short_packet extends packet {
    constraint ok { len < 64; }
}

class long_packet extends packet {
    constraint ok { len > 64; }
}

class err_pkt extends packet {
    constraint oops { len > 1512 && len <= 2000; }
}
```
Case study II - Drawbacks

- Soft constraints remain enabled unless a conflict occurs
  Could result in a small range or fixed values
- Default constraints are replaced entirely
  Must ensure a complete set of replacement constraints

```c
struct packet {
    rand uint data_len;
    keep soft data_len == 64;
}

struct short_packet extends packet {
    keep data_len in [32 : 1512];
}

class packet {
    rand int len;
    default constraint valid_len {
        len inside {[32:1512]};
    }
}

class long_packet extends packet {
    constraint valid_len {
        len > 64
    }
}
```

Exp Result = 32 : 1512
Act Result = 64

Exp Result = 64 : 1512
Act Result = 64 : max 32bit value
Case study II - SystemVerilog Constraints

- Different constraint name - additive
- Same constraint name - replacement

```verilog
class packet;
    rand int data_len;
    constraint valid_data_len {
        data_len inside {[32:1512]};
    }
endclass

class short_packet extends packet;
    constraint short_data_len {
        data_len < 64;
    }
endclass

class long_packet extends packet;
    constraint valid_data_len {
        data_len > 64;
    }
endclass

class err_len_packet extends packet;
    constraint err_data_len {
        data_len inside {[1513:2000]};
    }
endclass
```

Solver failure!!
Case study II – Alternative to “soft” in SV

- Turn constraint off
- Use `dist` operator

```plaintext
class packet;
    rand int data_len;
    constraint c1 {
        data_len >= 32 && data <= 1512;
    }
endclass

packet p1;
initial begin
    p1.c1.constraint_mode(0);
    p1.randomize() with {
        data_len == 2000;
    }
end

class packet;
    rand int data_len;
    constraint c1 {
        data_len dist {32:10000000,
                        [33:2000] := 1};
    }
endclass

packet p2;
initial begin
    p2.randomize() with {
        data_len == 64;
    }
end
```
Case Study II: Take-away

- Good understanding of constraint inheritance
- Disciplined methodology of base class constraints
- Can mimic behavior of other languages in SV
Conclusion

Case Study I:
- Knowledge of constraint solution space
- Constraints susceptible to performance

Case Study II:
- Knowledge of Constraint inheritance