2024
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE

MUNICH, GERMANY
OCTOBER 15-16, 2024

Improved Performance of Constraints

Milos Pericic, Verification Engineer, Veriest Solutions

Veriest

accellera
SYSTEMS INITIATIVE

Agenda

- *SystemVerilog* constraints
- How to improve constraints
- Circle randomization
- Constraints performance
- Questions

SystemVerilog constraints

- Randomization cases
- Resolved with constraint solver
- Variables dependencies can be large
- Eliminates randomization order
- Performances are crucial

How to improve constraints

- Engineering formulas definition
- Order helpers
- Function in constraint
- Unique values
- Delayed randomization
- Soft constraints

Engineering formulas definition

- Everything starts with definitions
- Real-life usage should be analyzed
- Engineering formulas simplification

Order helpers (1)

```
typedef enum bit[1:0] { TOP, BLOCK, DIRECT, RANDOM } test_type_t;
rand test_type_t test_type;
bit use_long_values = 1;
const int LONG_FACTOR = 256;

constraint packets_data_c {
  solve test_type before sub, mac;
  solve req, resp, sub, mid before mac;

  if (use_long_values && test_type inside { TOP, RANDOM }) {
    sub inside { [2 : 14] }; sub % 2 == 0;
    mac == (req – resp) * LONG_FACTOR - sub + mid;
  }
  ...
}
```

Order helpers (2)

- *SystemVerilog* solve-before construct for explicit order
  - *solve* A *before* B => randomize A and then B
- Multiple constructs allowed
- Reduce decision time for ordering
- Used only for random variables (*rand*)
- *test_type* is randomized before *sub* and *mac*
- *req*, *resp*, *sub* and *mid* are randomized before *mac*

Function in constraint (1)

```
constraint rsize_c {
  solve dir, stl, pic, fir, prec, speed, mult, volt, len before rsize;

  if ((is_fill || is_low) && (not_empty || (dir inside { LEFT, RIGHT }))) {
    if (speed != ZERO) {
      if (is_low) { pulse == volt; }
      else        { pulse == stl;  }

      if (mult >= 32) {
        eq == mult * pulse + 10 * pic + 2 * (len + 1) + 32;
      }
      else {
        eq == mult + 3 * pulse + 5 * pic + 12;
      }
```

Function in constraint (2)

```
      if ((prec == FAST) || (speed == ONE) || (speed == TWO)) {
        rsize == eq;
      }
      else {
        fir_con  == 0.8;
        fir_mul  == fir_con * fir;
        fir_trun == int'(fir_mul);

        (vrange >= eq      ) && (vrange <= fir_trun) ||
        (vrange >= fir_trun) && (vrange <= eq      );

        round == real'(vrange) + eq * 2;
        rsize == int'(round / 4.0) * 16 + (len + 1);
      }
    }
    ...
```

Function in constraint (3)

- Complex constraint can cause slowness
- Move complex logic to function
- Function accepts arguments and return value
- It relaxes constraint solver and simplify constraint
- Used variables could have their own complex constraints
- Relational operators are used for variable ranges
  - *vrange* = [*eq* : *fir_trun*] = [*fir_trun* : *eq*]

Function in constraint (4)

```
function int calc_rsize(
  int m_stl, int m_pic, int m_fir, speed_t m_speed, prec_t m_prec,
  bit m_is_low, int m_mult, int m_volt, int m_len
);
  int m_rsize, m_eq, m_pulse;

  m_pulse = m_is_low ? m_volt : m_stl;

  if (mult >= 32) begin
    m_eq = m_mult * m_pulse + 10 * m_pic + 2 * (m_len + 1) + 32;
  end
  else begin
    m_eq = m_mult + 3 * m_pulse + 5 * m_pic + 12;
  end
```

Function in constraint (5)

```
  if ((m_prec == FAST) || (m_speed == ONE) || (m_speed == TWO)) begin
    m_rsize = m_eq;
  end
  else begin
    real m_round, m_fir_con, m_fir_mul;
    int m_vrange, m_fir_trun;

    m_fir_con  = 0.8;
    m_fir_mul  = m_fir_con * m_fir;
    m_fir_trun = int'(m_fir_mul);

    m_vrange   = $urandom_range(m_eq, m_fir_trun);
    m_round    = real'(m_vrange) + m_eq * 2;
    m_rsize    = int'(m_round / 4.0) * 16 + (m_len + 1);
  end

  return m_rsize;
endfunction : calc_rsize
```

Function in constraint (6)

```
constraint rsize_c {
  solve dir, stl, pic, fir, prec, speed, mult, volt, len before rsize;

  if ((is_fill || is_low) && (not_empty || (dir inside { LEFT, RIGHT }))) {
    if (speed != ZERO) {
      rsize == calc_rsize(
        stl, pic, fir, speed, prec, is_low, mult, volt, len
      );
    }
    ...
  }
  ...
}
```

Function in constraint (7)

- Sign == is replaced with =
- Ternary operator *?* can replace *if-else*
- System calls are allowed in functions (*$urandom_range*)
- Debug functions with prints (*$display* or *`uvm_info*)
- Function calls can be nested

Unique values (1)

```
constraint phy_c {
 if (free_addr) {
  phy inside { 2, 8, 14, 20, 26, 32 };

  if (skew_part) {
   phy inside { 2, 20, 32 };
  }
  else if (max_part) {
   phy == 32;
  }
 }
 ...
}
```

Unique values (2)

- Pre-randomized values (*uvm_object*::*pre_randomize*)
  - Constant or enumerated values
  - Fixed or discrete numeric ranges
- Randomized variables are used in other constraints
- Pre-randomization does not consume time
- Constraint solver is relaxed

Unique values (3)

```
function void eth_config::pre_randomize();
  ...
  std::randomize(phy_pre_rand) with {
    phy_pre_rand inside { 2, 8, 14, 20, 26, 32 };

    if (skew_part) {
      phy_pre_rand inside { 2, 20, 32 };
    }
    else if (max_part) {
      phy_pre_rand == 32;
    }
  }
  ...
endfunction : pre_randomize
```

```
constraint phy_c {
  if (free_addr) {
    phy == phy_pre_rand;
  }
  ...
}
```

Unique values (4)

- Pre-randomized parts or whole constraint
- Randomization with *std*::*randomize* and *“with”* block
- Variables with pre-randomized values are not random (*rand*)
- Simple assign variable to constraint result

Delayed randomization

- Post-randomized values (*uvm_object*::*post_randomize*)
  - Constant, enumerated or numeric values
  - Other variables
- Post-randomized variables are totally independent
- They can be randomized in any function
- Pre-randomization → randomization → post-randomization
- Post-randomization does not consume time
- It improves constraint solver performance

Soft constraints

- Unexpected randomization results (e.g. coverage gaps)
- Default values for randomizations?
- Range of values vs soft constraint
- Can be disabled with *disable soft* construct
- Best to use when really need them

Circle randomization

- Circle of dependencies
- A ← B ← C ← D ← A
- A before B, C, D and D before A
- Impossible to randomize
- Requires definition change

Constraints performance

Tools	Constraints improvements cases [a]		
	Initial implementation	First optimization	Final optimization
Synopsys VCS Verdi	4464	2640	5
Cadence Xcelium Logic	4356	2400	4

[a]. *Time measured in seconds [s]*

Questions?