Complex Constraints

Unleashing the Power of the VCS Constraint Solver

John Dickol

Samsung Austin R&D Center

September 29, 2016

SNUG Austin

Agenda

Introduction

SystemVerilog Constraint Techniques

Constraint Examples

Debugging Constraints

Why Complex Constraints?

When you have eliminated the impossible, whatever remains, however improbable, must be the truth. - Sherlock Holmes

- Use constraints to define legal stimulus
- Add constraints to steer stimulus
- Complex stimulus needs complex constraints

Constraint Techniques

Constraints 101

- SystemVerilog constraints are *declarative*, not procedural.
  - Describe *what* you want - Solver will figure out *how*
  - Constraints specify expressions which must be true
  - Constraint solver choses values for random variables to ensure all constraints are satisfied
  - All constraints solved simultaneously (usually – some exceptions)

- Full specification in SystemVerilog Language Reference
  - Latest version: IEEE Std 1800-2012
  - Download free copy from IEEE Get Program: http://standards.ieee.org/getieee/1800/download/1800-2012.pdf

Constraint Expressions

Simple → Complex

```
1;
0;

x == 1;
x inside {[0:10]};

x < y;

if(a>b) c<d;
...
```

```
foreach(n_txns_per_slave[s]) {
   n_txns_per_slave[s] == txn_map.sum with(
        (item.index(2) == s) ? item : 0);
   );
}


map.ranges.or(r) with (
   item.addr inside
    {[r.start_addr : (r.end_addr - item.size)]}
    && item.cache_type == r.cache_type
);

...
```

Soft Constraints

- Are applied only if not contradicted by other constraints with higher priority
- In general, last constraint specified has higher priority
- See SV LRM for complete list of priorities
- Good for specifying default values for sequence/sequence_item
  which can be overridden in higher-level sequence

Soft Constraint Example

```
class myseq extends uvm_sequence#(mytxn);
  rand int count // number of transactions

  constraint c_count {
    soft count inside {[10:20]};
  }

  task body;
    mytxn  txn;

    repeat(count) begin
      `uvm_do(txn)
    end
  endtask
endclass
```

Default constraint for **`count`**
Soft constraint may be overridden

```
class topseq extends uvm_sequence#(mytxn);

  task body;
    myseq  seq;

     `uvm_do(seq)

     `uvm_do_with(seq, {count == 1000;}
  endtask
endclass
```

Default soft constraint applies

Inline hard constraint overrides soft constraint

Unique Constraints

- Constraint set of variables to have unique values:
  - Scalar integral values
  - Unpacked arrays of integral values

```
rand int x, y, z;
rand int arr[10];
     int q[$] = '{1, 2, 7, 11};

constraint c_unique {
  unique {x, y, z};
  unique {arr};
  unique {x, y, arr};
  unique {x, y, q};
}
```

Equivalent to:
`x!=y; y!=z; z!=x;`

Fill **`arr`** with unique values

Can mix scalars and arrays

Pick **`x`** & **`y`** not equal to elements of non-rand queue

Array Reduction Constraints

- Operate on *unpacked* arrays of integral values
- Apply specified operation between each element of the array

```
rand int a[10];
a.sum     == a[0] + a[1] + …
a.product == a[0] * a[1] * …
a.and     == a[0] & a[1] & …
a.or      == a[0] | a[1] | …
a.xor     == a[0] ^ a[1] ^ …
```

- Methods return single value of the same type as array element
  i.e sum of **`int`** returns **`int`**. xor of **`bit`** returns **`bit`**, etc.
- Optional "**`with`**" expression applied to each element before reduction operation
- Inside "**`with`**" expression:
  - "**`item`**" refers to current array element
  - "**`item.index`**" refers to current array index

Array Reduction Constraint Examples

- Constrain number of 1s in array of bits:

```
rand bit bq[$];

constraint c_sum {
  bq.size inside {[10:20]};
   (bq.sum with (int'(item))) == 7;
}
```

`sum()` of **`bit`** is **`bit`**
Cast to **`int`** to avoid overflow

- Constrain sum of first 3 elements of array:

`sum with(…)` equivalent to:
`foreach(a[i]) if(i<3) sum+= a[i]`

```
rand int a[10];

constraint c_sum_first_3 {
   (a.sum with ((item.index < 3) ? item : 0)) == 10;
}
```

`item.index` equivalent to **`i`**

`item` equivalent to **`a[i]`**

Global (Hierarchical) Constraints

- Class with rand variables may be rand member of another class
- When top-level object is randomized, all rand class members are also randomized
- Top-level object may constrain lower-level object variables

Global Constraint Example: Line Drawing

Rand instances of XY point class

```
class XY;
  rand int x;
  rand int y;

  constraint c_valid {
    x inside {[0:100]};
    y inside {[0:100]};
  }
endclass
```

Top-level object constrains lower-level

```
class Line;
  rand XY                          pt1;
  rand XY                          pt2;
  rand enum {HORIZONTAL, VERTICAL} direction;

  function new;
    if(pt1==null) pt1 = new;
    if(pt2==null) pt2 = new;
  endfunction

  constraint c_valid {
    if(direction == VERTICAL) {
      pt1.x == pt2.x;
      pt1.y != pt2.y;
    }

    if(direction == HORIZONTAL) {
      pt1.y == pt2.y;
      pt1.x != pt2.x;
    }
  }
endclass
```

Reusing Constraints: Policy Classes

- Goals:
  - Define constraints once and reuse in multiple objects
  - Add additional constraints to object at runtime
- Implementation:
  - Variation of hierarchical constraints: Lower-level object constrains parent object
  - Define reusable constraints in “policy class” container
  - Policy classes extended from common base class
  - “item” variable in policy class refers to parent object
  - Parent object has rand queue of policy base class

Policy Class Example

Rand queue of policy base class

```
class rand_policy_base #(type T=uvm_object);
    T item;

    virtual function void set_item(T item);
        this.item = item;
    endfunction
endclass
```

```
class Line;
  rand enum {HORIZ, VERT}      direction;
  rand rand_policy_base#(Line) pcy[$];

  function void pre_randomize;
    foreach(pcy[i]) pcy[i].set_item(this);
  endfunction
  ...
endclass
```

Set “item” in policy objects to point to this instance

```
class line_direction_policy
  extends rand_policy_base#(Line);

  constraint c_direction {
    item.direction dist {
       Line::VERT  := 5,
       Line::HORIZ := 1
    };
  }
endclass
```

“item” refers to parent Line object

Test case

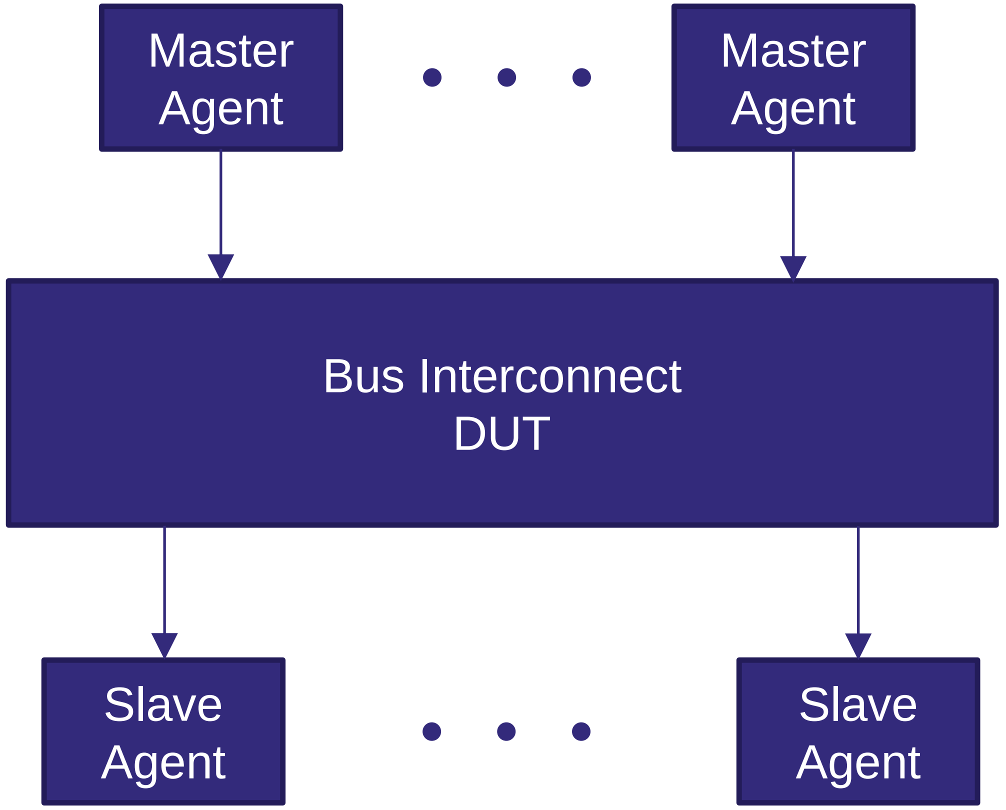
```
Line                   l = new;
line_direction_policy  dir_pcy = new;

l.pcy = {dir_pcy};

l.randomize;
```

Add policy to queue

SAMSUNG snug
Synopsys Users Group
Austin

Constraint Examples

Bus Interconnect Testbench



Tests may want to control:

- Total number of transactions for all masters
- Number of transactions per master
- Number of transactions per slave
- Number of transactions from one master to a particular slave
- Use/avoid using certain masters or slaves
- Etc.

Bus Interconnect Configuration Constraints

- Use 2-D array to model number of transactions
  `rand int unsigned txn_map[N_MASTER][N_SLAVE];`
- Add rand variables and constraints for provide "knobs" controllable by test

Sum rows to get txns per master

Sum columns to get txns per slave

Sum entire array to get total txns

	S0	S1	S2	S3	S4	S5	Total txns per master
M0	2	18	6	17	0	246	289
M1	1	0	3	128	0	155	287
M2	3	55	10	211	0	7	286
M3	15	0	97	2	0	175	286
M4	0	0	0	0	0	0	0
Total Txns per slave	21	73	116	358	0	583	Total txns 1151

Constrain Total Number of Transactions

```
class ms_config#(N_MASTER=5, N_SLAVE=6);
  rand int unsigned txn_map[N_MASTER][N_SLAVE];

  rand int unsigned total_txns;

  constraint c_valid {
    total_txns == txn_map.sum;
  }

endclass
```

`sum with(…)` is equivalent to:

```
foreach(txn_map[M,S])
  sum += txn_map[M][S];
```

Constrain Transactions Per Master

```
class ms_config#(N_MASTER=5, N_SLAVE=6);
  rand int unsigned txn_map[N_MASTER][N_SLAVE];

  rand int unsigned n_txns_per_master[N_MASTER];

  constraint c_valid {
     foreach(n_txns_per_master[m]) {
        n_txns_per_master[m] == txn_map[m].sum;
     }
  }

endclass
```

Sum each row of **`txn_map`** array

Constrain Transactions Per Slave

```
class ms_config#(N_MASTER=5, N_SLAVE=6);
  ...
  rand int unsigned n_txns_per_slave[N_SLAVE];

  constraint c_valid {
    foreach(n_txns_per_slave[s]) {
      n_txns_per_slave[s] == txn_map.sum with(
        (item.index(2) == s) ? item : 0);
      );
    }
  }

endclass
```

`sum with(…)` is equivalent to:

```
foreach(txn_map[M,S])
  if(S == s)  sum += txn_map[M][S];
  else        sum += 0;
```

`item.index(2)` returns
Index for 2nd array dimension;

Constrain Individual Masters/Slaves

```
class ms_config#(N_MASTER=5, N_SLAVE=6);
  ...
  rand bit          use_master[N_MASTER];
  rand bit          use_slave[N_MASTER];
  rand int unsigned use_n_masters;
  rand int unsigned use_n_slaves;

  constraint c_valid {
    foreach(use_master[m]) {
      (use_master[m] == 1) == (n_txns_per_master[m] > 0);
    }
    use_n_masters == use_master.sum with (int'(item));


    foreach(use_slave[s]) {
      (use_slave[s] == 1) == (n_txns_per_slave[s] > 0);
    }
    use_n_slaves  == use_slave.sum  with (int'(item))
  }
endclass
```

Master is used if it has at least 1 transaction

Count number of used masters

Test Scenario Constraints

```
ms_config cfg = new;

cfg.randomize with {
  total_txns == 10000;
  use_n_masters inside {[2:4]};
  use_master[1] == 1;
  use_slave[3] == 0;
}
```

Generate a total of 10000 transactions

Use between 2 and 4 masters

Use Master 1

Don’t use Slave 3

Command Line Control of Constraints

```
function void ms_config::get_plusargs;
  if($value$plusargs("total_txns=%d", total_txns))
     total_txns.rand_mode(0);

  if($value$plusargs("use_n_masters=%d", use_n_masters))
     use_n_masters.rand_mode(0);

  foreach(use_master[m]) begin
   string format = $sformatf("use_master[%0d]=%%b", m);
   if($value$plusargs(format, use_master[m]))
     use_master[m].rand_mode(0);
  end
  ...
endfunction
```

If plusarg specified on command line, disable randomization for corresponding variable

Compute plusarg names for individual array elements

Test case configuration

```
ms_config cfg = new;
  cfg.get_plusargs;
  cfg.randomize;
```

Simulator command line

```
simv +total_txns=10000 +use_n_masters=3 '+use_master[1]=1' ...
```

More Examples in the Paper

- Memory Map Example
  - Allocate non-overlapping address ranges
  - Constrain address range attributes: size, cacheability, …
  - Select transaction addresses from mapped ranges

Debugging Constraints

Debugging Solver Failures

Original intent:
if(y==1) x==2;
else   x == 0;

```
class C;
   rand int x;
   bit y = 1;
   constraint c1 {
      x inside {0, 2};
      x == y ? 2 : 0;
   }
endclass
```

Fixed class

```
class C;
   rand int x;
   bit y = 1;
   constraint c1 {
      x inside {0, 2};
      x == (y ? 2 : 0);
   }
endclass
```

Fix by adding parentheses

Simulator debug message

```
Solver failed when solving following set of constraints

bit[0:0] y = 1'h1;
rand integer x; // rand_mode = ON

constraint c1 // (from this) (constraint_mode = ON) (test.sv:7)
{
    (x inside {0, 2});
    ((x == y) ? 2 : 0);
}
```

Note added parentheses

Bug: == operator has higher precedence than ?: operator

Debugging Missing/Unexpected Solutions

Solver succeeds, but:

- Solver never chooses expected value
- Solver chooses unexpected value

For “never chooses” case:

- Add constraint to force expected value
- If impossible, solver will fail and show conflicting constraints

For other cases:

- Add simv options to enable solver debug

Command-line Control of Solver Debug

- Debug all randomize calls

```
simv +ntb_solver_debug=trace +ntb_solver_debug_filter=all
```

- Get serial number for randomize calls:

```
simv +ntb_solver_debug=serial
…
Randomize serial: 17, Total elapsed time : 0.430 s, Randomize runtime: 0.000 s, Current
constraint memory: 848 KB, Memory usage increment: 744 KB.
```

- Debug specific randomize call:

```
simv +ntb_solver_debug=trace +ntb_solver_debug_filter=17
```

- Extract standalone testcase:

```
simv +ntb_solver_debug=extract +ntb_solver_debug_filter=17
```

Interactive Constraint Debug

- Available in DVE and Verdi
  - Debug soft constraints (show which constraints are dropped)
  - Interactively modify constraints
  - Re-randomize without recompiling

- Details in DVE/Verdi documentation (Or ask your AC)

Summary

Summary

- Constraint solver can handle complex problems – use it!
- Constraint techniques:
  - Soft constraints
  - Unique constraints
  - Array reduction constraints
  - Reusable constraints: policy classes
- Constraint debug: Command-line and interactive

SAMSUNG snug

Synopsys Users Group
Austin

Thank You