Constraint Random Verification

Add a CSP solver to Chisel

Traditional Methodology

Based on Direct tests

- Verification plan list all the feature and corner case to test
- Each test hast to be implemented manually
- Manual inspection of waveform or assertion based testing

Problems with direct tests

- Test cannot be automatically generated, all the test needs to be written manually
- The number of test grow exponentially with the number of features in a product
- Hard to maintain not scalable
- Only feature described in the test plan is not enough, there might be corner cases that test engineer didn’t expect

Constraint Random

- Constraint programming lets users build generic, reusable objects that can later be extended or constrained to perform specific functions.
- One single test can reach different parts of the state space
- Allows users to automatically generate tests for functional verification

CSP Constraint Satisfaction Problem

- X a set of variables $\{X_1, \ldots, X_n\}$
- D a set of domains $\{D_1, \ldots, D_n\}$ one for each of the variables.
- C is a set of constraint that specify allowable combinations of values.
- D consists of a set of allowable values, $\{v_1, \ldots, v_n\}$ for variable X

- A solution is *consistent* if it doesn't violates any constraint
- A solution is *complete* if it includes all the variables

CSP Libraries

Java libraries

- OptaPlanner
- ChocoSolver

Scala libraries

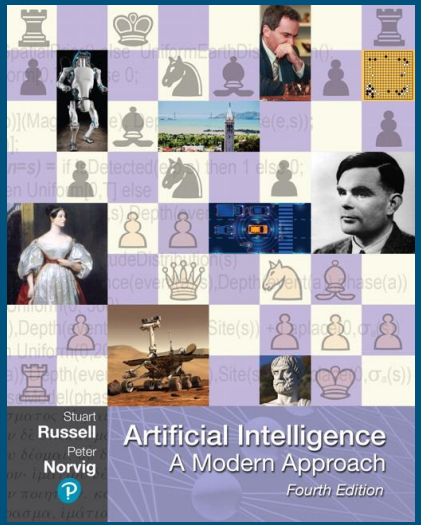
- CSP solver scala

```
val model = new Model("Constraint problem")¬
val bit_domain = (0x0 to 0xFF).toArray¬
val len_domain = (0x0 to 0x0A) ++ Set(0x14, 0x18) toArray¬
val payload_domain = (0x0 to 0xFF).toSet.diff(0x04 to 0xFF toSet).toArray¬
¬
val len = model.intVar("len", bit_domain)¬
val payload = model.intVar("payload", bit_domain)¬
¬
model.member(len, len_domain).post()¬
model.member(payload, payload_domain).post()¬
model.absolute(len, payload).post()¬
model.arithm(len, ">=", 2).post()¬
model.airthm(len, "<=", 4).post()¬
¬
¬
val solver = model.getSolver¬
```

Scala CSP

Base on the book Artificial Intelligence a Modern Approach

- Backtracking search
- Constraint Propagation



Comparison

SystemVerilog

```
class frame_t;
  rand pkt_type ptype;
  rand integer len;
  rand bit  [7:0] payload [];
  constraint common {
    payload.size() == len;
  }
  // Constraint the members
  constraint unicast {
    len <= 2;
    ptype == UNICAST;
  }
  // Constraint the members
  constraint multicast {
    len >= 3;
    len <= 4;
    ptype == MULTICAST;
  }
```

Chisel

```
class Frame extends Random {
  import pktType._
  var pType: RandInt = rand(pType, pktType.domainValues())
  var len: RandInt = rand(len, 0 to 10 toList)
  var noRepeat: RandCInt = randc( noRepeat, 0 to 1 toList)
  var payload: RandInt = rand(payload, 0 to 7 toList)

  val common = constraintBlock (
    binary ((len, payload) => len == payload)
  )

  val unicast = constraintBlock(
    unary (len => len <= 2),
    unary (pType => pType == UNICAST.id)
  )

  val multicast = constraintBlock(
    unary (len => len >= 3),
    unary (len => len <= 4),
    unary (pType => pType == MULTICAST.id)
  )
```

SystemVerilog

```
initial begin
   frame_t frame = new();
   integer j = 0;
   $write("--------------------------------\n");
   // Do contraint for Unicast frame
   frame.multicast.constraint_mode(0);
   if (frame.multicast.constraint_mode() == 0) begin
     if (frame.randomize() == 1) begin
       frame.print();
     end else begin
       $write("Failed to randomize frame\n");
     end
   end else begin
      $write("Failed to disable constraint multicast\n");
   end
   $write("--------------------------------\n");
   // Check the status of constraint multicast
   $write ("Constraint state of multicast is %0d\n",
       frame.multicast.constraint_mode());
   $write("--------------------------------\n");
   // Now disable the unitcast and enable multicast
   frame.unicast.constraint_mode(0);
   frame.multicast.constraint_mode(1);
   if (frame.randomize() == 1) begin
     frame.print();
   end else begin
     $write("Failed to randomize frame\n");
   end
   $write("--------------------------------\n");
end
```

Chisel

```
val frame = new Frame
println("Disable MULTICAST")
frame.multicast.disable()
while (frame.randomize) {
  println(frame)
  assert(frame.len <= 2)
  assert(frame.len == frame.payload)
  assert(frame.pType == UNICAST.id)
}
println("")
println("Enable MULTICAST ---  Disable UNICAST")
frame.unicast.disable()
frame.multicast.enable()
while (frame.randomize) {
  println(frame)
  assert(frame.len <= 4 && frame.len >= 3)
  assert(frame.len == frame.payload)
  assert(frame.pType == MULTICAST.id)
}
```

Current Status

Done

- Random variables, rand randc
- Constraint block
- Inside operator

Missing

- distributions dis
- random arrays
- unique / interactive /conditional constraints

Repository: https://github.com/parzival3/csp

Questions?