```
-- Snoopy bus cache protocol using write-invalidate
-- Implementation with 3 processors
-- and 2 possible data values.
MODULE main
VAR
  bus : bus(c0.bus-req, c0.bus-line-req,
            c1.bus-req, c1.bus-line-req,
            c2.bus-req, c2.bus-line-req);
  c0 : cache(p0.op, p0.line, bus.master0, bus.op, bus.line);
  p0 : processor(c0.stall);
  c1 : cache(p1.op, p1.line, bus.master1, bus.op, bus.line);
  p1 : processor(c1.stall);
  c2 : cache(p2.op, p2.line, bus.master2, bus.op, bus.line);
  p2 : processor(c2.stall);
  m  : memory(bus.op, bus.line);
SPEC
   -- Block A has unique owner
      AG (c0.ownA + c1.ownA + c2.ownA + m.ownA = 1)
   -- Block B has unique owner
    & AG (c0.ownB + c1.ownB + c2.ownB + m.ownB = 1)
   -- No error states
    & AG (!(c0.state = error) & !(c1.state = error)
    &     !(c2.state = error))

   -- Instruction not stalled indefinitely
      & AG (!EG c0.stall)
      & AG (!EG c1.stall & !EG c2.stall)
```

```
-- Model of a processor:
-- nondeterministically executes operations
-- But must wait until stall cleared before proceeding
MODULE processor(stall)
VAR
  op : {no-op, read, write };
  line: {lnA, lnB} ;
ASSIGN
  init(op) := no-op;
  next(op) :=
    case
      stall : op;
      !stall : {no-op, read, write};
    esac;
  init(line) := {lnA, lnB} ;
  next(line) :=
    case
      stall : line;
      1 : {lnA, lnB} ;
    esac;
```

```
-- Model of cache.
-- Responds to processor operations when master,
-- and to bus operations when not
MODULE cache(proc-op, proc-line, master, bus-op, bus-line)
VAR
  state : { invalid , clean, dirty, error };
  tag : {lnA, lnB} ;

ASSIGN
  init(state) := invalid;
  next(state) :=
    case master :
      -- When bus master, operate on behalf of processor
      case
        state in { invalid, clean } :
          case
            proc-op = read : clean;

            proc-op = write : dirty;
            1 : state;
          esac;
        state = dirty :
          case

            proc-line = tag : dirty;
            proc-op in { read, write } : invalid;
            1 : state;
          esac;
        1 : state;
      esac;
      -- When not master, must watch bus traffic
      1 :
      case
        state = clean :
          case
            !(bus-line = tag) : clean;
            bus-op in { invalidate, xread } : invalid;

            bus-op = write : error;
            1 : state;
          esac;
        state = dirty :
          case
            !(bus-line = tag) : dirty;
            bus-op = read : clean;
            bus-op = xread : invalid;
            bus-op in { write, invalidate } : error;
            1 : state;
          esac;
        1 : state ;
      esac;
    esac;
  init(tag) := {lnA, lnB} ;
  next(tag) :=
    case
      !master : tag;
      -- When bus master, operate on behalf of processor
      state in { invalid, clean } &
        proc-op in {read, write} : proc-line;
      1 : tag;
    esac;
```

```
DEFINE
  -- When have something to do, can issue bus request
  -- Possible request types:
     -- READ: Get copy of block with read access
     -- WRITE: Write block to memory
     -- XREAD: Get copy of block with exclusive  access
     -- INVALIDATE: Invalidate all other copies of block
  bus-req :=
    case bus-op = arbitrate :
      case
        state = invalid :
          case
            proc-op = read : read;

            proc-op = write : xread;
            1 : no-op;
          esac;
        state = clean :
          case
            proc-op = read & ! proc-line = tag : read;

            proc-op = write & proc-line = tag : invalidate;
            proc-op = write : xread;
            1 : no-op;
          esac;
        state = dirty :
          case
            proc-op in {read, write} &
               ! proc-line = tag : write;

            1 : no-op;
          esac;
        1 : no-op;
      esac;
      1 : no-op;
    esac;
-- Stall processor until proper tag & state
stall :=
          proc-op in {read, write} &
          ! proc-line = tag |
          proc-op = read & ! state in {clean, dirty} |
          proc-op = write &
            ! (state = dirty

              );
-- Identity of line for bus request
bus-line-req :=
  case
    state in {invalid, clean} : proc-line;
    state = dirty : tag;
    1 : {lnA, lnB} ;
  esac;
ownA := state = dirty & tag = lnA;
ownB := state = dirty & tag = lnB;
```

```
-- Memory model.
-- Maintains copies of data when not owned elsewhere
-- Does not make any bus requests
MODULE memory(bus-op, bus-line)
VAR
  ownA : boolean;
  ownB : boolean;
ASSIGN
  init(ownA) := 1;
  init(ownB) := 1;
  next(ownA) :=
    case
      ! bus-line = lnA : ownA;
      -- Gaining ownership
      bus-op = write : 1;
      bus-op = read : 1;
      -- Giving up ownership
      bus-op in {invalidate, xread} : 0;
      1 : ownA;
    esac;
  next(ownB) :=
    case
     ! bus-line = lnB : ownB;
      -- Gaining ownership
      bus-op = write : 1;
      bus-op = read : 1;
      -- Giving up ownership
      bus-op in {invalidate, xread} : 0;
      1 : ownB;
    esac;
```

```
-- Bus model.  Model arbitration & granting phases only
-- Grants control (master) among requesters
-- Two versions: one that passes around token round-robin
-- other generates token nondeterministically
MODULE bus(req0, line0, req1, line1, req2, line2)
VAR
  token : {0, 1, 2}; -- Pass around token
  master : {0, 1, 2, no-one};
  op : {arbitrate, read, xread, write, invalidate, no-op};
  line : {lnA, lnB} ;
ASSIGN
  init(token) := {0,1,2};
  next(token) :=
    case
      op = arbitrate : token;
      1 : {0, 1, 2}; -- Nondeterministic
    esac;
  init(master) := no-one;
  next(master) :=
    case
      !(op = arbitrate) : no-one;
      -- Arbitrate for new master
      token = 0 :
        case
         !(req0 = no-op) : 0;
         !(req1 = no-op) : 1;
         !(req2 = no-op) : 2;
         1 : no-one;
        esac;
      token = 1 :
        case
         !(req1 = no-op) : 1;
         !(req2 = no-op) : 2;
         !(req0 = no-op) : 0;
         1 : no-one;
        esac;
      1 :        -- token = 2
        case
         !(req2 = no-op) : 2;
         !(req0 = no-op) : 0;
         !(req1 = no-op) : 1;
         1 : no-one;
        esac;
    esac;
```

```
  init(op) := no-op;
  next(op) :=
    case
      !(op = arbitrate) : arbitrate;
      next(master) = 0 : req0;
      next(master) = 1 : req1;
      next(master) = 2 : req2;
      1 : no-op;
    esac;
  init(line) := {lnA, lnB} ;
  next(line) :=
    case
      !(op = arbitrate) : line;
      next(master) = 0 : line0;
      next(master) = 1 : line1;
      next(master) = 2 : line2;
      1 : {lnA, lnB} ;
  esac;
DEFINE
  master0 := master = 0;
  master1 := master = 1;
  master2 := master = 2;
FAIRNESS
  token = 0
FAIRNESS
  token = 1
FAIRNESS
  token = 2
```