Joshua Monson
ECEN 620
Assignment 9

Overview:

The purpose of this assignment was to improve our **risc_rpm** processor test bench by adding a cover group with several cover points and cross cover points that gather data to help us understand the functionality that we've tested. The coverpoints that were created for this homework help us understand if we've thoroughly tested all instructions. For example, the covergroup helps us determine if we've tested all source and destination register combinations with all instructions that have source and destination registers. Additionally, the covergroup tells us when we have tested a variety of sequences of instructions and read and written all memory addresses for the read and write instructions.

The remaining contents are:

1. System Verilog Code (that was modified since the last assignment)
2. Full Coverage Report
3. Coverage Report showing Partial Coverage

SystemVerilog Code:

test.sv

```
import TbEnvPkg::*;

Instruction cur_inst;
covergroup Cov();

   opcodes:coverpoint cur_inst.opcode {
      //1. All opcodes have been executed.
      bins ops[] = {[0:6], 9};
      illegal_bins bad_ops[] = {[7:8],[10:15]};
      //4. All opcodes have been preceeded and followed all other opcodes
      bins op_order[] = (0,1,2,3,4,5,6,9=>0,1,2,3,4,5,6,9);
   }

   srcs:coverpoint cur_inst.src {
      bins sregs[] = {[0:3]};
   }
```

```
   dsts:coverpoint cur_inst.dst {
      bins dregs[] = {[0:3]};
   }

   //2. The source for every opcode that has a source has been 0,1,2,3
   opcode_src:cross opcodes, srcs {
      ignore_bins no_src = binsof(opcodes.ops) intersect {0,5,9};
      ignore_bins trans = binsof(opcodes.op_order);
   }

   //3. The destination for every opcode that has a dst has been 0,1,2,3
   opcode_dst:cross opcodes, dsts {
      ignore_bins no_dst = binsof(opcodes.ops) intersect {0,6,9};
      ignore_bins trans = binsof(opcodes.op_order);
   }

   //5. Opcodes with both source and destination have had all combs of srcs
and dsts
   opcode_src_dst:cross opcodes, srcs, dsts {
      ignore_bins no_src_dst = binsof(opcodes.ops) intersect {0,5,6,9};
      ignore_bins trans = binsof(opcodes.op_order);
   }


   addresses: coverpoint cur_inst.byte1 iff (cur_inst.opcode != NOP &&
                              cur_inst.opcode != ADD &&
                              cur_inst.opcode != SUB &&
                              cur_inst.opcode != AND &&
                              cur_inst.opcode != NOT &&
                              cur_inst.opcode != HALT){
      bins addrs[] = {[0:255]};
   }

   WrRdRdiMem:cross opcodes, addresses {
      //6 & 7: All mem locations written
      ignore_bins non_read_write = binsof(opcodes.ops) intersect {0,1,2,3,4};
      ignore_bins trans = binsof(opcodes.op_order);
   }
endgroup // Cov

class Driver_cbs_cov extends Driver_cbs;
   Cov ck;
   function new();
      ck=new();
   endfunction // new

   virtual task pre_inst(ref Instruction I);
      cur_inst = I;
      ck.sample();
   endtask // pre_inst
endclass // Driver_cbs_cov


program automatic test(SPM_IF.TEST mem_bus);
   initial begin
      static virtual SPM_IF.TEST vMemBus = mem_bus;
      Driver_cbs_cov cov_callback;
```

```
      Environment E;
      cov_callback = new();
      E = new ();
      E.build(vMemBus);
      E.D.callbacks.push_back(cov_callback);
      E.run(10000);

      $display("Test Complete found %d Errors.", E.D.Scb.ErrorCounter);
   end
endprogram // test
```

driver.sv

```
class Driver_cbs;
   virtual task pre_inst(ref Instruction I);
   endtask // pre_inst
   virtual task post_inst(ref Instruction I);
   endtask // post_inst

endclass // Driver_cbs

class Driver;
   virtual SPM_IF.TEST dut_if;
   Driver_cbs callbacks[$];
   ScoreBoard Scb;
   mailbox #(Instruction) mbx;
   event   handshake;

   function new (virtual SPM_IF.TEST dif,
                 mailbox #(Instruction) mb, event hs);
      dut_if = dif;
      mbx = mb;
      Scb = new ();
      handshake = hs;
   endfunction // new

   function automatic void initialize();
      dut_if.cb.rst <= 1;
      dut_if.cb.data_out <= 8'h0;
   endfunction

   task automatic reset();
      initialize();
      @dut_if.cb;
      dut_if.cb.rst <= 1;
      @dut_if.cb;
      @dut_if.cb;
      dut_if.cb.rst <= 0;
      repeat (4) @dut_if.cb;
      dut_if.cb.rst <= 1;
      repeat (1) @dut_if.cb;
   endtask
```

```
   function automatic string getOpcode(Instruction I);
      case(I.byte0[7:4])
      NOP: return "NOP";
      ADD: return "ADD";
      SUB: return "SUB";
      AND: return "AND";
      NOT: return "NOT";
      RD:  return "RD";
      WR:  return "WR";
      BR:  return "BR";
      BRZ: return "BRZ";
      RDI: return "RDI";
      HALT: return "HALT";
      endcase // case (I.byte0[7:4])
   endfunction // string


   task automatic run(int count);
     Instruction I;
      int i = 0;

     repeat(count) begin
      mbx.get(I);
      foreach(callbacks[i]) callbacks[i].pre_inst(I);
      sendInstruction(I,i);
      foreach(callbacks[i]) callbacks[i].post_inst(I);
      ->handshake;
      i++;
     end // repeat (count)

      if ($root.top.DUT.Controller.state == 4'd11) begin
       $display("Error: Processor has halted");
      end

   endtask // run

   task automatic sendInstruction(input Instruction I, int i);
            $display("@%0d: Starting Instr #%0d :: %s", $time, i,
getOpcode(I));
      //Fetch 1
      Scb.fetch1();
      @dut_if.cb;

      //Fetch 2: Get and Drive I.byte0

      Scb.fetch2(I);

      dut_if.cb.data_out <= I.byte0;
      @dut_if.cb;
      //Decode:
      Scb.check_address(dut_if.cb.address);
        Scb.check_pc($root.top.DUT.Processor.PC_count);
      Scb.check_ir($root.top.DUT.Processor.instruction);

      Scb.decode(I,8'hff);

      case (I.byte0[7:4])
```

```
      NOP: doNOP(I);

      ADD, SUB, AND: doAddSubAnd(I);
      NOT: doNot(I);
      RD, RDI: doRdRdi(I,8'hff);
      WR: doWr(I);
      BR, BRZ, HALT: doBrBrzHalt(I);
   endcase // case (I.byte0[7:4])
endtask // sendInstruction


task automatic doAddSubAnd(input Instruction I);
   $display("Decoded AddSubAnd Instr");
   //Leave Decode State
   @dut_if.cb;
   //Leave S_ext1
   @dut_if.cb;
   Scb.check_regs($root.top.DUT.Processor.R0_out,
                  $root.top.DUT.Processor.R1_out,
                  $root.top.DUT.Processor.R2_out,
                  $root.top.DUT.Processor.R3_out);
endtask // AddSubAnd

task automatic doNOP(input Instruction I);
   $display("Decoded NOP");
   @dut_if.cb;
endtask // doNOP


task automatic doNot(input Instruction I);
   $display("Decoded NOT");
   //Leave Decode State
   @dut_if.cb;
   Scb.check_regs($root.top.DUT.Processor.R0_out,
                  $root.top.DUT.Processor.R1_out,
                  $root.top.DUT.Processor.R2_out,
                  $root.top.DUT.Processor.R3_out);
endtask // doNot

task automatic doRdRdi(input Instruction I, byte rdval);
   $display("Decoded RdRdi Instr");
   //Leave Decode State
   @dut_if.cb;
   dut_if.cb.data_out <= I.byte1;

   if(I.byte0[7:4] == RD) begin
    //Leave RD1
    @dut_if.cb;
    dut_if.cb.data_out <= rdval;
    Scb.check_address(dut_if.cb.address);
    //Leave RD2
    @dut_if.cb;
    Scb.check_address(dut_if.cb.address);
   end
   else begin //RDI
   //Leave RD1
   @dut_if.cb;
```

```
      Scb.check_address(dut_if.cb.address);
      end // else: !if(I.byte0[7:4] == RD)
      Scb.check_regs($root.top.DUT.Processor.R0_out,
                     $root.top.DUT.Processor.R1_out,
                     $root.top.DUT.Processor.R2_out,
                     $root.top.DUT.Processor.R3_out);
   endtask // doRdRdi

   task automatic doWr(input Instruction I);
      $display("Decoded Wr");
      //Leave Decode State
      @dut_if.cb;
      //Leave WR1
      dut_if.cb.data_out <= I.byte1;
      @dut_if.cb;
      Scb.check_address(dut_if.cb.address);
      //Leave WR2
      @dut_if.cb;
      Scb.check_address(dut_if.cb.address);
      Scb.check_dataIn(dut_if.cb.data_in);
   endtask

   task automatic doBrBrzHalt(input Instruction I);
      $display("Error: Encountered Opcode %d", I.byte0[7:4]);
      $display("This Opcode is not yet supported by the Test Bench Env");
      $finish;
   endtask
endclass;
```

Generator.sv

```
`define SV_RAND_CHECK(r) \
   do begin \
      if (!(r)) begin \
       $display("%s:%0d: Randomization failed \"%s\"", \
              `__FILE__, `__LINE__, `"r`");\
       $finish; \
      end \
   end while(0)

class Instruction;
   rand bit [7:0] byte0;
   rand bit [7:0] byte1;

   bit [3:0] opcode;
   bit [1:0] src;
   bit [1:0] dst;

   //Don't allow the Generator to Create BR, BRZ, or HALT Inst
   constraint c_byte0 { byte0[7:4] != BR;
                        byte0[7:4] != BRZ;
                        byte0[7:4] != HALT;
                        byte0[7:4] != 4'hA;
                        byte0[7:4] != 4'hB;
```

```
                        byte0[7:4] != 4'hC;
                        byte0[7:4] != 4'hD;
                        byte0[7:4] != 4'hE; };
    //Don't Randomize byte1 if single byte inst
    constraint c_byte1 { ( byte0[7:4] == NOP ||
                    byte0[7:4] == ADD ||
                    byte0[7:4] == SUB ||
                    byte0[7:4] == AND ||
                    byte0[7:4] == NOT ||
                    byte0[7:4] == HALT) -> (byte1 == 8'd0);};

    function void post_randomize();
       opcode = byte0[7:4];
       src = byte0[3:2];
       dst = byte0[1:0];
    endfunction // post_randomize

endclass // Instruction

class Generator;
    Instruction I;
    mailbox   #(Instruction) mbx;
    event     handshake;

    function new (mailbox #(Instruction) m, event hs);
       mbx = m;
       I = new ();
       handshake = hs;
    endfunction // new

    task run(int count);
       Instruction i;
       repeat(count) begin
        `SV_RAND_CHECK(I.randomize());
        i=new I;
        mbx.put(i);
        wait(handshake.triggered);
       end
    endtask
endclass // Generator
```

scoreboard.sv

```
class ScoreBoard;
    int ErrorCounter;

    bit [7:0] pc;
    bit [7:0] ir;
    bit [7:0] r[4];
    byte DataInQueue[$];
    byte AddressQueue[$];

    function new ();
       ErrorCounter = 0;
```

```
   pc = 0;
   ir = 0;
   r[0]=0; r[1]=0; r[2]=0; r[3]=0;
endfunction // new

function automatic void incr_pc();
   pc++;
   $display("incr pc: %02h", pc);
endfunction // incr_pc

function automatic void update_pc(int val);
   pc = val;
endfunction // update_pc

function automatic void update_ir(Instruction I);
   ir = I.byte0;
endfunction // update_ir

function automatic void update_addrq(input bit [7:0] v);
   AddressQueue.push_back(v);
   $display("Updating Address Queue: %02h %p", v, AddressQueue);

endfunction // update_addrq


function automatic void fetch1();
   //The value of the pc should next appear on the
   // address line.
   $display("@%t:fetch1: %02h", $time, pc);
   update_addrq(pc);
endfunction // fetch1

function automatic void fetch2(Instruction I);
   $display("@%t:fetch2", $time);
   update_ir(I);
   incr_pc();
endfunction // fetch2

function automatic void readbyte2(Instruction I);
   $display("readbyte2 I.byte1=%02x", I.byte1);
   update_addrq(pc);
   update_addrq(I.byte1);
   incr_pc();
endfunction // readbyte2

function automatic void decode(Instruction I, bit [7:0] rdval);

   bit [3:0] opcode = I.byte0[7:4];
   bit [1:0] src = I.byte0[3:2];
   bit [1:0] dst = I.byte0[1:0];
   $display("@%0d SB: Decode", $time);
   case (I.byte0[7:4])
   NOP: ;
   ADD: r[dst] = r[src] + r[dst];
   SUB: r[dst] = r[dst] - r[src];
   AND: r[dst] = r[src] & r[dst];
   NOT: r[dst] = ~r[src];
```

```
      RD: begin
         r[dst] = rdval;
         readbyte2(I);
      end
      RDI:begin
         update_addrq(pc);
         incr_pc();
         r[dst] = I.byte1;
      end
      WR: begin
         readbyte2(I);
         DataInQueue.push_back(r[src]);
      end
      BR,BRZ,HALT: ;
      endcase // case (I.byte0)
   endfunction // decode

   function automatic void check_pc(bit [7:0] cpc);
      compare_value("PC", cpc, pc);
   endfunction // check_pc

   function automatic void check_ir(bit [7:0] cir);
      compare_value("IR", cir, ir);
   endfunction // check_ir

   function automatic void check_regs(bit [7:0] cr0,
                        bit [7:0] cr1,
                        bit [7:0] cr2,
                        bit [7:0] cr3);
      compare_value("r0", cr0, r[0]);
      compare_value("r1", cr1, r[1]);
      compare_value("r2", cr2, r[2]);
      compare_value("r3", cr3, r[3]);
   endfunction // check_regs

   function automatic void check_address(bit [7:0] caddress);
      compare_qvalue("address", "AddressQueue", caddress, AddressQueue);
   endfunction // check_address

   function automatic void check_dataIn(bit [7:0] cdataIn);
      compare_qvalue("cdataIn", "DataInQueue", cdataIn, DataInQueue);
   endfunction // check_address

   function automatic void compare_qvalue(string name, string qname, bit
[7:0] actual, ref byte queue[$]);
      $display("Checking Q=%p", queue);
      if(queue.size() == 0)
      $display("Error: %s is empty!", qname);
      else
      compare_value(name, actual, queue.pop_front());
   endfunction // compare_qvalue

   function automatic void compare_value(string name, bit [7:0] actual, bit
[7:0] expected);
      if(actual != expected) begin
       $display("@%08d: Error: Found Unexpected Value for %s: Expected: %02X
Actual: %02X", $time, name, expected, actual);
```

```
        ErrorCounter++;
      end
      //else
      //$display("@%08d: %s is correct", $time, name);

  endfunction // compare_value



Endclass
```

Complete Coverage Report

COVERGROUP COVERAGE:

Covergroup	Metric	Goal/ At Least	Status
TYPE /Alu_RISC_v_unit/Cov	100.0%	100	Covered
Coverpoint Cov::opcodes	100.0%	100	Covered //Requirements 1 & 4
Coverpoint Cov::srcs	100.0%	100	Covered
Coverpoint Cov::dsts	100.0%	100	Covered
Coverpoint Cov::addresses	100.0%	100	Covered
Cross Cov::opcode_src	100.0%	100	Covered //Requirement 2
Cross Cov::opcode_dst	100.0%	100	Covered //Requirement 3
Cross Cov::opcode_src_dst	100.0%	100	Covered //Requirement 5
Cross Cov::WrRdRdiMem	100.0%	100	Covered //Requirements 6 & 7
Covergroup instance Alu_RISC_v_unit::Driver_cbs_cov::ck	100.0%	100	Covered
Coverpoint opcodes	100.0%	100	Covered
covered/total bins:	72	72	
missing/total bins:	0	72	
illegal_bin bad_ops[7]	0		ZERO
illegal_bin bad_ops[8]	0		ZERO

illegal_bin bad_ops[10]	0	ZERO
illegal_bin bad_ops[11]	0	ZERO
illegal_bin bad_ops[12]	0	ZERO
illegal_bin bad_ops[13]	0	ZERO
illegal_bin bad_ops[14]	0	ZERO
illegal_bin bad_ops[15]	0	ZERO
bin ops[0]	642	1 Covered
bin ops[1]	657	1 Covered
bin ops[2]	624	1 Covered
bin ops[3]	650	1 Covered
bin ops[4]	626	1 Covered
bin ops[5]	2318	1 Covered
bin ops[6]	2229	1 Covered
bin ops[9]	2254	1 Covered
bin op_order[9=>9]	517	1 Covered
bin op_order[9=>6]	501	1 Covered
bin op_order[9=>5]	483	1 Covered
bin op_order[9=>4]	146	1 Covered
bin op_order[9=>3]	149	1 Covered
bin op_order[9=>2]	169	1 Covered
bin op_order[9=>1]	152	1 Covered
bin op_order[9=>0]	137	1 Covered
bin op_order[6=>9]	497	1 Covered
bin op_order[6=>6]	498	1 Covered
bin op_order[6=>5]	569	1 Covered
bin op_order[6=>4]	128	1 Covered
bin op_order[6=>3]	130	1 Covered
bin op_order[6=>2]	135	1 Covered
bin op_order[6=>1]	134	1 Covered
bin op_order[6=>0]	138	1 Covered

bin op_order[5=>9]	509	1 Covered
bin op_order[5=>6]	515	1 Covered
bin op_order[5=>5]	543	1 Covered
bin op_order[5=>4]	148	1 Covered
bin op_order[5=>3]	165	1 Covered
bin op_order[5=>2]	142	1 Covered
bin op_order[5=>1]	142	1 Covered
bin op_order[5=>0]	154	1 Covered
bin op_order[4=>9]	152	1 Covered
bin op_order[4=>6]	147	1 Covered
bin op_order[4=>5]	132	1 Covered
bin op_order[4=>4]	43	1 Covered
bin op_order[4=>3]	40	1 Covered
bin op_order[4=>2]	29	1 Covered
bin op_order[4=>1]	48	1 Covered
bin op_order[4=>0]	34	1 Covered
bin op_order[3=>9]	141	1 Covered
bin op_order[3=>6]	147	1 Covered
bin op_order[3=>5]	148	1 Covered
bin op_order[3=>4]	44	1 Covered
bin op_order[3=>3]	43	1 Covered
bin op_order[3=>2]	40	1 Covered
bin op_order[3=>1]	45	1 Covered
bin op_order[3=>0]	42	1 Covered
bin op_order[2=>9]	149	1 Covered
bin op_order[2=>6]	130	1 Covered
bin op_order[2=>5]	132	1 Covered
bin op_order[2=>4]	53	1 Covered
bin op_order[2=>3]	47	1 Covered
bin op_order[2=>2]	37	1 Covered

bin op_order[2=>1]	36	1 Covered
bin op_order[2=>0]	40	1 Covered
bin op_order[1=>9]	143	1 Covered
bin op_order[1=>6]	152	1 Covered
bin op_order[1=>5]	154	1 Covered
bin op_order[1=>4]	35	1 Covered
bin op_order[1=>3]	44	1 Covered
bin op_order[1=>2]	35	1 Covered
bin op_order[1=>1]	51	1 Covered
bin op_order[1=>0]	43	1 Covered
bin op_order[0=>9]	146	1 Covered
bin op_order[0=>6]	139	1 Covered
bin op_order[0=>5]	157	1 Covered
bin op_order[0=>4]	29	1 Covered
bin op_order[0=>3]	32	1 Covered
bin op_order[0=>2]	36	1 Covered
bin op_order[0=>1]	49	1 Covered
bin op_order[0=>0]	54	1 Covered

The cover bins checking that all write, read, and rdi instructions have written all memory locations have been excluded due to length.

Partially Coverage Report

COVERGROUP COVERAGE:

Covergroup	Metric	Goal/ At Least	Status
TYPE /Alu_RISC_v_unit/Cov	50.5%	100	Uncovered
type_option.weight=1			

type_option.goal=100

type_option.comment=

type_option.strobe=0

type_option.merge_instances=0

Coverpoint Cov::opcodes 48.6% 100 Uncovered //Requirements 1 &4

Coverpoint Cov::srcs 100.0% 100 Covered

Coverpoint Cov::dsts 100.0% 100 Covered

Coverpoint Cov::addresses 13.6% 100 Uncovered

Cross Cov::opcode_src 65.0% 100 Uncovered //Requirement 2

Cross Cov::opcode_dst 55.0% 100 Uncovered //Requirement 3

Cross Cov::opcode_src_dst 17.1% 100 Uncovered //Requirement 5

Cross Cov::WrRdRdiMem 4.5% 100 Uncovered Requirement 6 & 7

Covergroup instance \/Alu_RISC_v_unit::Driver_cbs_cov::ck

50.5% 100 Uncovered

option.name=\/Alu_RISC_v_unit::Driver_cbs_cov::ck

Coverpoint opcodes 48.6% 100 Uncovered

covered/total bins: 35 72

missing/total bins: 37 72

option.weight=1

option.goal=100

option.comment=

option.at_least=1

option.auto_bin_max=64

option.detect_overlap=0

illegal_bin bad_ops[7] 0 ZERO

illegal_bin bad_ops[8] 0 ZERO

illegal_bin bad_ops[10] 0 ZERO

illegal_bin bad_ops[11] 0 ZERO

illegal_bin bad_ops[12] 0 ZERO

illegal_bin bad_ops[13] 0 ZERO

illegal_bin bad_ops[14]	0	ZERO	
illegal_bin bad_ops[15]	0	ZERO	
bin ops[0]	3	1	Covered
bin ops[1]	4	1	Covered
bin ops[2]	2	1	Covered
bin ops[3]	3	1	Covered
bin ops[4]	2	1	Covered
bin ops[5]	15	1	Covered
bin ops[6]	12	1	Covered
bin ops[9]	9	1	Covered
bin op_order[9=>9]	2	1	Covered
bin op_order[9=>6]	0	1	ZERO
bin op_order[9=>5]	5	1	Covered
bin op_order[9=>4]	0	1	ZERO
bin op_order[9=>3]	1	1	Covered
bin op_order[9=>2]	0	1	ZERO
bin op_order[9=>1]	1	1	Covered
bin op_order[9=>0]	0	1	ZERO
bin op_order[6=>9]	1	1	Covered
bin op_order[6=>6]	4	1	Covered
bin op_order[6=>5]	3	1	Covered
bin op_order[6=>4]	1	1	Covered
bin op_order[6=>3]	1	1	Covered
bin op_order[6=>2]	1	1	Covered
bin op_order[6=>1]	1	1	Covered
bin op_order[6=>0]	0	1	ZERO
bin op_order[5=>9]	4	1	Covered
bin op_order[5=>6]	5	1	Covered
bin op_order[5=>5]	1	1	Covered
bin op_order[5=>4]	1	1	Covered

bin op_order[5=>3]	1	1 Covered
bin op_order[5=>2]	0	1 ZERO
bin op_order[5=>1]	0	1 ZERO
bin op_order[5=>0]	3	1 Covered
bin op_order[4=>9]	0	1 ZERO
bin op_order[4=>6]	1	1 Covered
bin op_order[4=>5]	1	1 Covered
bin op_order[4=>4]	0	1 ZERO
bin op_order[4=>3]	0	1 ZERO
bin op_order[4=>2]	0	1 ZERO
bin op_order[4=>1]	0	1 ZERO
bin op_order[4=>0]	0	1 ZERO
bin op_order[3=>9]	1	1 Covered
bin op_order[3=>6]	1	1 Covered
bin op_order[3=>5]	1	1 Covered
bin op_order[3=>4]	0	1 ZERO
bin op_order[3=>3]	0	1 ZERO
bin op_order[3=>2]	0	1 ZERO
bin op_order[3=>1]	0	1 ZERO
bin op_order[3=>0]	0	1 ZERO
bin op_order[2=>9]	1	1 Covered
bin op_order[2=>6]	1	1 Covered
bin op_order[2=>5]	0	1 ZERO
bin op_order[2=>4]	0	1 ZERO
bin op_order[2=>3]	0	1 ZERO
bin op_order[2=>2]	0	1 ZERO
bin op_order[2=>1]	0	1 ZERO
bin op_order[2=>0]	0	1 ZERO
bin op_order[1=>9]	0	1 ZERO
bin op_order[1=>6]	0	1 ZERO

bin op_order[1=>5]	1	1 Covered
bin op_order[1=>4]	0	1 ZERO
bin op_order[1=>3]	0	1 ZERO
bin op_order[1=>2]	0	1 ZERO
bin op_order[1=>1]	2	1 Covered
bin op_order[1=>0]	0	1 ZERO
bin op_order[0=>9]	0	1 ZERO
bin op_order[0=>6]	0	1 ZERO
bin op_order[0=>5]	3	1 Covered
bin op_order[0=>4]	0	1 ZERO
bin op_order[0=>3]	0	1 ZERO
bin op_order[0=>2]	0	1 ZERO
bin op_order[0=>1]	0	1 ZERO
bin op_order[0=>0]	0	1 ZERO
Coverpoint srcs	100.0%	100 Covered
covered/total bins:	4	4
missing/total bins:	0	4
option.weight=1		
option.goal=100		
option.comment=		
option.at_least=1		
option.auto_bin_max=64		
option.detect_overlap=0		
bin sregs[0]	12	1 Covered
bin sregs[1]	7	1 Covered
bin sregs[2]	15	1 Covered
bin sregs[3]	16	1 Covered
Coverpoint dsts	100.0%	100 Covered
covered/total bins:	4	4
missing/total bins:	0	4

option.weight=1

option.goal=100

option.comment=

option.at_least=1

option.auto_bin_max=64

option.detect_overlap=0

bin dregs[0]	8	1 Covered
bin dregs[1]	16	1 Covered
bin dregs[2]	13	1 Covered
bin dregs[3]	13	1 Covered