

Overview:

The purpose of this assignment was to improve our **risc_rpm** processor test bench by adding a virtual interface and parameterizing the test bench with respect to the address width. I had previously added a virtual interface to my test bench so this part of the project was already complete. I did, however, change my method of passing the interface to the test bench from a parameter to use a cross-module reference. This makes it more feasible for all tests to use the same parameter list and reduces the amount of changes that you may have to make in the future.

Modifying the test bench to use the ADDRESS_WIDTH parameter was the most difficult part of this assignment. This required specializing almost every class that I created for this test bench. This was due to the fact that most the *Instruction* class had to be specialized and most classes used the *Instruction* class. I also had to add a specialized *comparator* class to handle comparisons of various widths.

One of the challenges of this assignment was that everything was to be parameterized for address width but the actual DUT. The DUT kept is default address width of 8 bit. The difficulty stems from the fact that the address output of the DUT cannot be specialized independently from the dataIn and dataOut lines. Therefore specializing the DUT would have broken my test bench and forced me to do more than the assignment asked.

The non-specialization of the DUT caused errors and the expected warning when the ADDRESS_WIDTH parameter was set to 9. The errors that you will see reported are due to the fact that I am checking the address values against expected address values. Since the DUT only has an eight bit address line the top a mismatch is flagged whenever the bit 9 of the address should be set.

In addition I also modified my design so that it runs until it achieves 100% coverage on the covergroup. It does this by running for 250 clock cycles and then checking if it has achieved coverage; if it has not it runs for another 250 clock cycles and checks again.

Clock Cycles for 7, 8, 9:

ADDRESS_WIDTH=7: 3000 Cycles

ADDRESS_WIDTH=8: 7500 Cycles

ADDRESS_WIDTH=9: 14250 Cycles

The reason for the difference is that with each extra address bit the address space doubles which means that it takes longer to achieve full coverage of the address space.

The remaining contents are:

1. System Verilog Code
2. Screen Capture of Coverage window for ADDRESS_WIDTH == 9
3. Portions of Transcript window from ADDRESS_WIDTH == 9

SystemVerilog Code:

top.sv

```
`default_nettype none

module top();
  parameter ADDRESS_WIDTH=7;

  bit clk = 0;
  always #5 clk = ~clk;

  string instr;
  string state;

  always_comb begin
    case($root.top.DUT.Processor.instruction[7:4])
      NOP: instr = "NOP";
      ADD: instr = "ADD";
      SUB: instr = "SUB";
      AND: instr = "AND";
```

```

    NOT: instr = "NOT";
    RD:  instr = "RD";
    WR: instr = "WR";
    BR: instr = "BR";
    BRZ: instr = "BRZ";
    RDI: instr = "RDI";
    HALT: instr = "HALT";
    endcase // case (I.byte0[7:0])
end

always_comb begin
    case($root.top.DUT.Controller.state)
        0: state = "idle";
        1: state = "fet1";
        2: state = "fet2";
        3: state = "dec";
        4: state = "ex1";
        5: state = "rd1";
        6: state = "rd2";
        7: state = "wr1";
        8: state = "wr2";
        9: state = "br1";
        10: state = "br2";
        11: state = "halt";
        default state = "unknown";
    endcase
end

SPM_IF #(.ADDRESS_WIDTH(ADDRESS_WIDTH)) mem_bus (clk);

test #(.ADDRESS_WIDTH(ADDRESS_WIDTH)) TEST();
RISC_SPM DUT(.clk(clk),
    .rst(mem_bus.rst),
    .data_out(mem_bus.data_out),
    .address(mem_bus.address),
    .data_in(mem_bus.data_in),
    .write(mem_bus.write));
endmodule

```

test.sv

```

import TbEnvPkg::*;

program automatic test #(ADDRESS_WIDTH=8);

    Instruction #(ADDRESS_WIDTH) 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);
        }
    end

```

```

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.bytel 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 addr[] = {[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 #(ADDRESS_WIDTH=8) extends Driver_cbs
#(ADDRESS_WIDTH);
  Cov ck;
  function new();
    ck=new();
  endfunction // new

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

```

```

initial begin
    static virtual SPM_IF #(ADDRESS_WIDTH).TEST vMemBus =
$root.top.mem_bus;
    static int      cycle_count = 0;

    Driver_cbs_cov #(ADDRESS_WIDTH) cov_callback;
    Environment #(ADDRESS_WIDTH) E;
    cov_callback = new();
    E = new ();
    E.build(vMemBus);
    E.D.callbacks.push_back(cov_callback);

    do begin
        E.run(250);
        cycle_count+=250;
    end
    while (Cov::get_coverage() < 100.0);

    $display("Coverage %d %f", Cov::get_coverage(), Cov::get_coverage());
    $display("Test Executed for %d cycles", cycle_count);
    $display("Test Complete found %d Errors.", E.D.Scb.ErrC.get_errors());
end
endprogram // test

```

driver.sv

```

class Driver_cbs #(ADDRESS_WIDTH=8);
    virtual task pre_inst(ref Instruction #(ADDRESS_WIDTH) I);
endtask // pre_inst
    virtual task post_inst(ref Instruction #(ADDRESS_WIDTH) I);
endtask // post_inst

endclass // Driver_cbs

class Driver #(ADDRESS_WIDTH=8);
    virtual SPM_IF #(ADDRESS_WIDTH).TEST dut_if;
    Driver_cbs #(ADDRESS_WIDTH) callbacks[$];
    ScoreBoard #(ADDRESS_WIDTH) Scb;
    mailbox #(Instruction #(ADDRESS_WIDTH)) mbx;
    event handshake;

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

    function automatic void initialize();
        dut_if.cb.rst <= 1;
    endfunction
endclass

```

```

    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 #(ADDRESS_WIDTH) 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 #(ADDRESS_WIDTH) 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 #(ADDRESS_WIDTH) 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 #(ADDRESS_WIDTH) 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 #(ADDRESS_WIDTH) I);
    $display("Decoded NOP");
@dut_if.cb;
endtask // doNOP

task automatic doNot(input Instruction #(ADDRESS_WIDTH) 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 #(ADDRESS_WIDTH) 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 #(ADDRESS_WIDTH) 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 #(ADDRESS_WIDTH) 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

```



```

        end \
    end while(0)

class Instruction #(ADDRESS_WIDTH=8);
    rand bit [7:0] byte0;
    rand bit [ADDRESS_WIDTH-1:0] 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 == 0);};

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

endclass // Instruction

class Generator #(ADDRESS_WIDTH=8);
    Instruction #(ADDRESS_WIDTH) I;
    mailbox #(Instruction #(ADDRESS_WIDTH)) mbx;
    event handshake;

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

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

```

agent.sv

```
class Agent #(ADDRESS_WIDTH=8);
    mailbox #(Instruction #(ADDRESS_WIDTH)) mbx0, mbx1;

    function new (mailbox #(Instruction #(ADDRESS_WIDTH)) mb0,
                  mailbox #(Instruction #(ADDRESS_WIDTH)) mb1);
        mbx0 = mb0;
        mbx1 = mb1;
    endfunction // new

    task automatic run(int count);
        Instruction #(ADDRESS_WIDTH) I;
        repeat (count) begin
            mbx0.get(I);
            mbx1.put(I);
        end
    endtask // run
endclass // Agent
```

Environment.sv

```
class Environment #(ADDRESS_WIDTH=8);
    Generator #(ADDRESS_WIDTH) G;
    Agent #(ADDRESS_WIDTH) A;
    Driver #(ADDRESS_WIDTH) D;
    event DtoG_hs;

    mailbox #(Instruction #(ADDRESS_WIDTH)) GtoA, AtoD;

    function new ();

    endfunction // new

    task automatic build(virtual SPM_IF #(ADDRESS_WIDTH).TEST vSpm_if);
        GtoA = new ();
        AtoD = new ();

        G = new (GtoA, DtoG_hs);
        A = new (GtoA, AtoD);
        D = new (vSpm_if, AtoD, DtoG_hs);
        D.reset();
    endtask; // build

    task automatic run(int count);
        fork
            G.run(count);
            A.run(count);
            D.run(count);
        join
    endtask // run
```

```
endclass // Environment
```

scoreboard.sv

```
class ErrorCounter;  
    static int count;
```

```
    function new ();  
        count = 0;  
    endfunction // new
```

```
    function void incr();  
        count++;  
    endfunction // incr
```

```
    function int get_errors();  
        return count;  
    endfunction // get_errors
```

```
endclass // ErrorCounter
```

```
class comparator #(type T=bit[7:0]);  
    ErrorCounter EC;
```

```
    function new (ErrorCounter E);  
        EC = E;  
    endfunction // new
```

```
    function automatic void compare(string name, T actual, T expected);  
        if(actual != expected) begin  
            $display("@%08d: Error: Found Unexpected Value for %s: Expected: %X  
Actual: %X", $time, name, expected, actual);  
            EC.incr();  
        end  
    endfunction // compare_value  
endclass // comparator
```

```
class ScoreBoard #(ADDRESS_WIDTH);  
    ErrorCounter ErrC;
```

```
    comparator #(bit [ADDRESS_WIDTH-1:0]) compare_address;  
    comparator #(bit [7:0]) compare_data;
```

```
    bit [7:0] pc;  
    bit [7:0] ir;  
    bit [7:0] r[4];  
    byte DataInQueue[$];  
    bit [ADDRESS_WIDTH-1:0] AddressQueue[$];
```

```
    function new ();  
        ErrC = new();  
        compare_address = new (ErrC);  
        compare_data = new (ErrC);  
        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 #(ADDRESS_WIDTH) I);
    ir = I.byte0;
endfunction // update_ir

function automatic void update_addrq(input bit [ADDRESS_WIDTH-1:0] 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 #(ADDRESS_WIDTH) I);
    $display("@%t:fetch2", $time);
    update_ir(I);
    incr_pc();
endfunction // fetch2

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

function automatic void decode(Instruction #(ADDRESS_WIDTH) 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];
    endcase

```

```

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_data.compare("PC", cpc, pc);
endfunction // check_pc

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

function automatic void check_address(bit [ADDRESS_WIDTH-1:0] caddress);
    //compare_qvalue("address", "AddressQueue", caddress, AddressQueue);
    if(AddressQueue.size() == 0)
        $display("Error: AddressQueue is empty!");
    else
        compare_address.compare("address", caddress, AddressQueue.pop_front());
    endfunction // check_address

function automatic void check_dataIn(bit [7:0] cdataIn);
    //compare_qvalue("cdataIn", "DataInQueue", cdataIn, DataInQueue);
    if(DataInQueue.size() == 0)
        $display("Error: DataInQueue is empty!");
    else
        compare_data.compare("cdataIn", cdataIn, DataInQueue.pop_front());
    endfunction //



















```

```

endclass

```

Coverage Window for ADDRESS WIDTH==9

Covergroups				
Name	Coverage	Goal	% of Goal	Status
[-] /top/TEST				
[-] TYPE Cov	100.0%	100	100.0%	
[-] CVP Cov::opcodes	100.0%	100	100.0%	
[-] CVP Cov::srcs	100.0%	100	100.0%	
[-] CVP Cov::dsts	100.0%	100	100.0%	
[-] CVP Cov::addresses	100.0%	100	100.0%	
[-] CROSS Cov::opcode_src	100.0%	100	100.0%	
[-] CROSS Cov::opcode_dst	100.0%	100	100.0%	
[-] CROSS Cov::opcode_src_dst	100.0%	100	100.0%	
[-] CROSS Cov::WrRdRdiMem	100.0%	100	100.0%	
[-] INST Vtop/TEST/Driver_cbs_cov::Driver_cbs_cov__1::ck	100.0%	100	100.0%	
[+] CVP opcodes	100.0%	100	100.0%	
[+] CVP srcs	100.0%	100	100.0%	
[+] CVP dsts	100.0%	100	100.0%	
[+] CVP addresses	100.0%	100	100.0%	
[+] CROSS opcode_src	100.0%	100	100.0%	
[+] CROSS opcode_dst	100.0%	100	100.0%	
[+] CROSS opcode_src_dst	100.0%	100	100.0%	
[+] CROSS WrRdRdiMem	100.0%	100	100.0%	

Portions of Transcript Window for ADDRESS WIDTH==9

```
# vsim -coverage -do {run -all} -c -novopt top
# // Questa Sim-64
# // Version 10.1c linux_x86_64 Jul 27 2012
# //
# // Copyright 1991-2012 Mentor Graphics Corporation
# // All Rights Reserved.
# //
# // THIS WORK CONTAINS TRADE SECRET AND PROPRIETARY INFORMATION
# // WHICH IS THE PROPERTY OF MENTOR GRAPHICS CORPORATION OR ITS
# // LICENSORS AND IS SUBJECT TO LICENSE TERMS.
# //
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment10/work.top
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment10/work.Alu_RISC_v_unit
```

```
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment10/work.TbEnvPkg
# Loading sv_std.std
# Loading work.TbEnvPkg
# Loading work.Alu_RISC_v_unit
# Loading work.top
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment10/work.SPM_IF
# Loading work.SPM_IF
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment10/work.test
# Loading work.test
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment10/work.RISC_SPM
# Loading work.RISC_SPM
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment10/work.Processing_Unit
# Loading work.Processing_Unit
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment10/work.Register_Unit
# Loading work.Register_Unit
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment10/work.Alu_RISC
# Loading work.Alu_RISC
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment10/work.mux5_1
# Loading work.mux5_1
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment10/work.mux3_1
# Loading work.mux3_1
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment10/work.Control_Unit
# Loading work.Control_Unit
# ** Warning: (vsim-3015) top.sv(55): [PCDPC] - Port size (8 or 8) does not match connection size (9)
for port 'address'. The port definition is at: risc_spm/RISC_SPM.v(21). //Expected Warning
#      Region: /top/DUT
# ** Warning: (vsim-8634) Code was not compiled with coverage options.
# run -all
# @75: Starting Instr #0 :: AND
# @      75:fetch1: 00
```

Updating Address Queue: 00 '{0}

@ 85:fetch2

incr pc: 01

@95 SB: Decode

Decoded AddSubAnd Instr

@115: Starting Instr #1 :: RDI

@ 115:fetch1: 01

Updating Address Queue: 01 '{1}

...

Updating Address Queue: f7 '{247}

@ 617105:fetch2

incr pc: f8

@617115 SB: Decode

Updating Address Queue: f8 '{248}

incr pc: f9

Decoded RdRdi Instr

@617135: Starting Instr #248 :: WR

@ 617135:fetch1: f9

Updating Address Queue: f9 '{249}

@ 617145:fetch2

incr pc: fa

@617155 SB: Decode

readbyte2 l.byte1=123

Updating Address Queue: fa '{250}

Updating Address Queue: 123 '{250, 291}

incr pc: fb

Decoded Wr

@00617185: Error: Found Unexpected Value for address: Expected: 123 Actual: 023 //Example of Error Due to fixed DUT bitwidth

@617185: Starting Instr #249 :: AND


```
# @      617185:fetch1: fb
# Updating Address Queue: fb '{251}'
# @      617195:fetch2
# incr pc: fc
# @617205 SB: Decode
# Decoded AddSubAnd Instr
# Coverage    100 100.000000
# Test Executed for    14250 cycles
# Test Complete found    3255 Errors.
```