Joshua Monson
ECEN 620
Assignment 7.1

Overview:

The purpose of this assignment was to create a full test-bench for a **risc_rpm** processor. This test bench was to generate and send 10 instructions into the processor and perform checks to ensure the processor is executing correctly. The generator/agent/dut are all synchronized to be operating on the same transaction through the use of the single mailbox and event synchronization technique.

In addition to a generator, agent, driver, dut, and environment I also create a class of a scoreboard that managed and updated the golden state as time when along. I used a virtual interface to allow the driver access to the interface.

The remaining contents are:

1. System Verilog Code for the Test and Test Environment
2. Waveform showing the execution of the 10 instructions
3. Transcripts showing error free/buggy behavior.

SystemVerilog Code:

top.sv

```
`default_nettype none

module top();
   bit clk = 0;
   always #5 clk = ~clk;

   string instr;

   always_ff @(posedge clk) 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


   SPM_IF mem_bus(clk);

   test TEST(mem_bus);
   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
```

spm_if.sv

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

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

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

endclass // Agent
```

test.sv

```
import TbEnvPkg::*;

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

      Environment E;
      E = new ();
      E.build(vMemBus);
      E.run(10);

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

Environment.sv

```
class Environment;
   Generator G;
   Agent A;
   Driver D;
   event DtoG_hs;

   mailbox #(Instruction) GtoA, AtoD;

   function new ();

   endfunction // new

   task automatic build(virtual SPM_IF.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(10);
       A.run(10);
       D.run(10);
      join
   endtask // run


endclass // Environment
```

Genorator.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;

   //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);};

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
```

Agent.sv

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

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

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

```
endclass // Agent
```

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();
      //$display("incr pc");
      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(bit [7:0] v);
      //$display("Updating Address Queue: %02h", v);
      AddressQueue.push_back(v);
   endfunction // update_addrq


   function automatic void fetch1();
      //The value of the pc should next appear on the
      // address line.
      //$display("Fetch1");
      update_addrq(pc);
   endfunction // fetch1

   function automatic void fetch2(Instruction I);
      //$display("fetch2");
      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
      r[dst] = pc;
   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("Q.size()=%0d", queue.size());
      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
```

Driver.sv

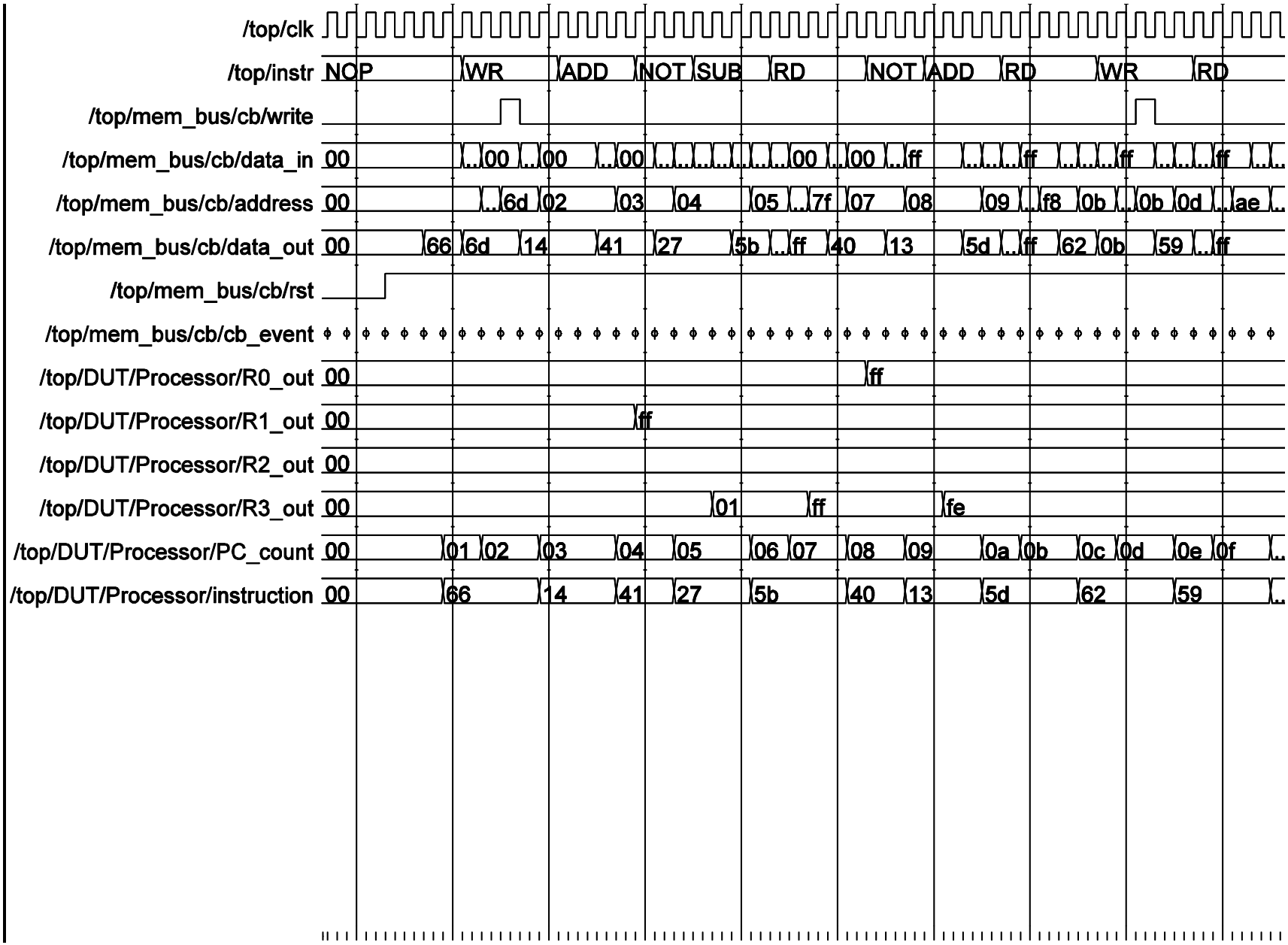
```
class Driver;
   virtual SPM_IF.TEST dut_if;
   ScoreBoard Scb;
   mailbox #(Instruction) mbx;
   event   handshake;

   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);
    $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])
      ADD, SUB, AND: doAddSubAnd(I);
      NOT: doNot(I);
      RD, RDI: doRdRdi(I,8'hff);
      WR: doWr(I);
      BR, BRZ, HALT: doBrBrzHalt(I);
    endcase

    ->handshake;
```

```
    i++;
   end // repeat (count)

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

endtask // run

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 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;
```

Waveforms


/top/clk
/top/instr
NOP
WR
ADD
NOT
SUB
RD
NOT
ADD
RD
WR
RD
/top/mem_bus/cb/write
/top/mem_bus/cb/data_in
/top/mem_bus/cb/address
/top/mem_bus/cb/data_out
/top/mem_bus/cb/rst
/top/mem_bus/cb/cb_event
/top/DUT/Processor/R0_out
/top/DUT/Processor/R1_out
/top/DUT/Processor/R2_out
/top/DUT/Processor/R3_out
/top/DUT/Processor/PC_count
/top/DUT/Processor/instruction

Working Transcript:

```
# @75: Starting Instr #0 :: WR
# @125: Starting Instr #1 :: ADD
# @165: Starting Instr #2 :: NOT
# @195: Starting Instr #3 :: SUB
# @235: Starting Instr #4 :: RD
# @285: Starting Instr #5 :: NOT
# @315: Starting Instr #6 :: ADD
# @355: Starting Instr #7 :: RD
# @405: Starting Instr #8 :: WR
# @455: Starting Instr #9 :: RD

# run -all

# Test Complete found          0 Errors.
```

Non-Working Transcript:

```
# @75: Starting Instr #0 :: WR
# @00000095: Error: Found Unexpected Value for PC: Expected: 01 Actual: 66
# @00000115: Error: Found Unexpected Value for address: Expected: 01 Actual:
66
# @125: Starting Instr #1 :: ADD
# @00000145: Error: Found Unexpected Value for address: Expected: 02 Actual:
00
# @00000145: Error: Found Unexpected Value for PC: Expected: 03 Actual: 14
# @165: Starting Instr #2 :: NOT
# @00000185: Error: Found Unexpected Value for address: Expected: 03 Actual:
00
# @00000185: Error: Found Unexpected Value for PC: Expected: 04 Actual: 41
# @195: Starting Instr #3 :: SUB
# @00000215: Error: Found Unexpected Value for address: Expected: 04 Actual: ff
# @00000215: Error: Found Unexpected Value for PC: Expected: 05 Actual: 27
# @235: Starting Instr #4 :: RD
# @00000255: Error: Found Unexpected Value for address: Expected: 05 Actual:
01
```

@00000255: Error: Found Unexpected Value for PC: Expected: 06 Actual: 5b
@00000275: Error: Found Unexpected Value for address: Expected: 06 Actual: 5b
@285: Starting Instr #5 :: NOT
@00000305: Error: Found Unexpected Value for address: Expected: 07 Actual: ff
@00000305: Error: Found Unexpected Value for PC: Expected: 08 Actual: 40
@315: Starting Instr #6 :: ADD
@00000335: Error: Found Unexpected Value for address: Expected: 08 Actual: ff
@00000335: Error: Found Unexpected Value for PC: Expected: 09 Actual: 13
@355: Starting Instr #7 :: RD
@00000375: Error: Found Unexpected Value for address: Expected: 09 Actual: fe
@00000375: Error: Found Unexpected Value for PC: Expected: 0a Actual: 5d
@00000395: Error: Found Unexpected Value for address: Expected: 0a Actual: 5d
@405: Starting Instr #8 :: WR
@00000425: Error: Found Unexpected Value for address: Expected: 0b Actual: ff
@00000425: Error: Found Unexpected Value for PC: Expected: 0c Actual: 62
@00000445: Error: Found Unexpected Value for address: Expected: 0c Actual: 62
@455: Starting Instr #9 :: RD
@00000475: Error: Found Unexpected Value for address: Expected: 0d Actual: 00
@00000475: Error: Found Unexpected Value for PC: Expected: 0e Actual: 59
@00000495: Error: Found Unexpected Value for address: Expected: 0e Actual: 59

Test Complete found 24 Errors.