Joshua Monson  
ECEN 620   
Assignment 9

**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 it width 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**);**

**}**

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 **#(**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**;**

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** **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**];**

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!"**);**

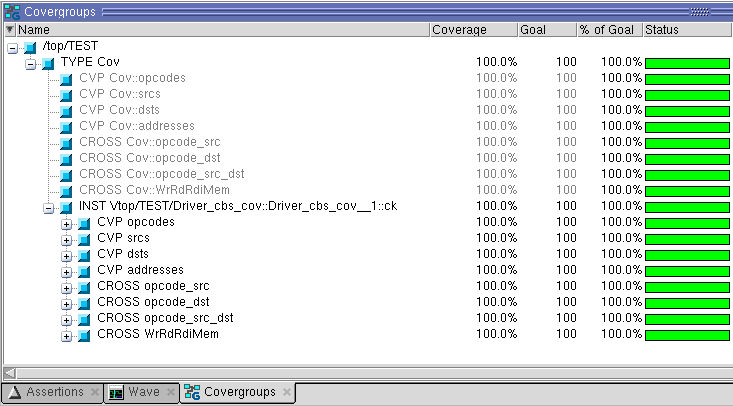
**else**

compare\_data**.**compare**(**"cdataIn"**,** cdataIn**,** DataInQueue**.**pop\_front**());**

**endfunction** //

endclass

**Coverage Window for ADDRESS\_WIDTH==9**

****

**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 I.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.**