Josh Monson  
ECEN 620 Assignment 2.2 & 2.3  
9/29/14

**Assignment 2.2**

**Test Bench Code**

`default\_nettype none

`timescale 1ns**/**100ps

**module** testbench **();**

logic clk**;**

logic write**;**

logic read**;**

logic **[**7**:**0**]** data\_in**;**

logic **[**15**:**0**]** address**;**

logic **[**8**:**0**]** data\_out**;**

int error\_counter**;**

typedef struct **{**

shortint address**;**

byte data\_to\_write**;**

byte expected\_read**;**

byte actual\_read**;**

**}** rd\_wr\_transaction**;**

rd\_wr\_transaction RdWrArray**[];**

shortint addr**;**

byte data**;**

**initial** **begin**

clk **=** 0**;**

error\_counter **=** 0**;**

read **=** 0**;**

RdWrArray **=** new**[**6**];**

foreach**(**RdWrArray**[**i**])** **begin**

addr **=** $random**;**

data **=** $random**;**

RdWrArray**[**i**].**address **=** addr**;**

RdWrArray**[**i**].**data\_to\_write **=** data**;**

RdWrArray**[**i**].**expected\_read **=** **{^**data**,** data**};**

**end**

foreach**(**RdWrArray**[**i**])** **begin**

write\_to\_memory**(**RdWrArray**[**i**].**address**,** RdWrArray**[**i**].**data\_to\_write**);**

**end**

//Randomize the Ordering for the reads.

RdWrArray**.**shuffle**();**

foreach**(**RdWrArray**[**i**])** **begin**

**@(negedge** clk**)**

read **=** 1**;**

address **=** RdWrArray**[**i**].**address**;**

**@(posedge** clk**)**

**#**1

RdWrArray**[**i**].**actual\_read **=** data\_out**;**

**if(**RdWrArray**[**i**].**expected\_read **!=** RdWrArray**[**i**].**actual\_read**)** **begin**

$display**(**"Found Error: %03X %03X"**,** RdWrArray**[**i**].**expected\_read**,** RdWrArray**[**i**].**actual\_read**);**

error\_counter **=** error\_counter **+** 1**;**

**end**

**end**

$display**(**"Print Read Values"**);**

foreach**(**RdWrArray**[**i**])** **begin**

$display**(**"\t%03X"**,** RdWrArray**[**i**].**actual\_read**);**

**end**

$display**(**"Error Count: %d"**,** error\_counter**);**

**end**

**task** write\_to\_memory**(input** shortint addr**,** byte data**);**

write **=** 1**;**

data\_in **=** data**;**

address **=** addr**;**

**@(posedge** clk**);**

write **=** 0**;**

**@(negedge** clk**);**

**endtask;**

**always** **begin**

**#**5 clk **=** **~**clk**;**

**end**

my\_mem mem**(**clk**,**

write**,**

read**,**

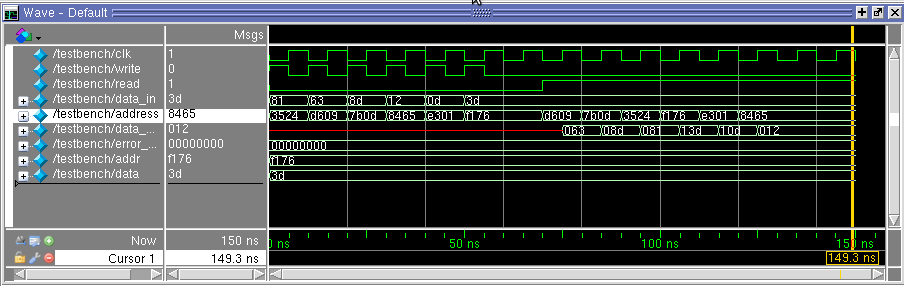
data\_in**,**

address**,**

data\_out**);**

endmodule

**Screen Shot of Waveform**

****

**Working Transcript Window**

**do testbench.do**

**# vsim -novopt testbench   
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment2.1/work.testbench  
# Loading sv\_std.std  
# Loading work.testbench  
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment2.1/work.my\_mem  
# Loading work.my\_mem**

**# Print Read Values  
# 063  
# 08d  
# 081  
# 03d  
# 00d  
# 012**

**# Error Count: 0**

**Non-Working Transcript Window**

# vsim -novopt testbench   
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment2.1/work.testbench  
# Loading sv\_std.std  
# Loading work.testbench  
# Refreshing /net/fpga1/users/joshuas2/ECEn620/Assignments/Assignment2.1/work.my\_mem  
# Loading work.my\_mem  
# Found Error: Expected: 063 Actual: 000  
# Found Error: Expected: 08d Actual: 000  
# Found Error: Expected: 081 Actual: 000  
# Found Error: Expected: 03d Actual: 000  
# Found Error: Expected: 00d Actual: 000  
# Found Error: Expected: 012 Actual: 000  
  
# Print Read Values  
# 000  
# 000  
# 000  
# 000  
# 000  
# 000  
# Error Count: 6

**Assignment 2.3**

**Test Plan**

Each Register Should:

1. Register should reset to the specified value.
2. Register writes should have no side effects (only the addressed register should be updated).
3. Register reads should only return the value of the addressed register and have no side effects.
4. Each bit in a register should set/reset.

Each of the follow tests were performed on each register. Resets tests occurred initially and after each individual register test.

**Reset Test:**

1. Reset the circuit.
2. Read and check that all registers have reset to the correct value.

**Inverse Reset Test:**

1. Write the inverse of the expected reset value to the register.
2. Reset the circuit
3. Read the Register and check for the proper reset value.

**Write Register Test:**

1. Write a value to the target register.
2. Perform a read to the same register
3. Check if the value read is equal to the value written.

**Signal Bit high (or low) Left Rotation Test:**

1. Initialize a 16-bit value with a single bit high (or low), (e.g. 0x8000)
2. Rotate the 16-bit value to the left.
3. Perform “Write Register Test”
4. Repeat Steps 2, 3 until all bits in register have been written.

**Bug Report**

**Bug #1:** Bit 0 of “analog test” does not reset to ‘1’  
 **Test:** Reset Register Test  
 **Observed:** Bit 0 did not reset properly (0xABCC)

**Bug #2:** Bit 15 of “adc0\_reg” stuck at ‘1’  
 **Test:** Single Bit Low Left Rotation Test  
 **Observed:** Bit 15 did not change to 0

**Bug #3:** Upper and Lower bytes of “adc1\_reg” Swapped  
 **Test:** Single Bit Low Left Rotation Test   
 **Observed:** Lower Byte is appears as upper byte

# Test that each bit can be Reset

# Testing bit 0

# Error: Register: "adc1\_reg" Expected: fffe Actual: feff

# Testing bit 4

# Error: Register: "adc1\_reg" Expected: ffef Actual: efff

# Testing bit 8

# Error: Register: "adc1\_reg" Expected: feff Actual: fffe

# Testing bit 12

# Error: Register: "adc1\_reg" Expected: efff Actual: ffef

**Bug #4:** “temp\_sensor0” shifted left by one bit  
 **Test:** Single Bit High Left Rotation Test (also manifests on Low Rot. Test)  
 **Observed:** Register shifted to left by one in all cases.  
 # Error: Register: "temp\_sensor0\_reg" Expected: 0001 Actual: 0002  
 # Error: Register: "temp\_sensor0\_reg" Expected: 0002 Actual: 0004  
 # Error: Register: "temp\_sensor0\_reg" Expected: 0004 Actual: 0008  
 # Error: Register: "temp\_sensor0\_reg" Expected: 0008 Actual: 0010  
 # Error: Register: "temp\_sensor0\_reg" Expected: 0010 Actual: 0020

**Bug #5:** “temp\_sensor1” will not reset after it has been written  
 **Test:** Inverse Reset Test  
 **Expected:** 0x0000  
 **Observed:** 0xFFFF, (Note: Passed High and Low Rotation Tests; shows problem is only in reset)

**Bug #6:** Writes to “digital\_test” update “amp\_gain”  
 **Tests:** Reset Test then low and high rotation tests  
 **Observed:** Reset test passed; during rotation tests “digital gain” always read zero, while “amp\_gain” always updated to expected value.

**Bug #7:** Writes to “amp\_gain” update “digital\_test”  
 **Tests:** Reset Test then low and high rotation tests  
 **Observed:** Reset test passed; during rotation tests “amp\_gain” always read zero, while “digital\_test” always updated to expected value.

**Bug #8:** “digital\_config” bit 15 stuck at 0  
 **Tests:** Single Bit High and Low Rotation Tests  
 **Observed:** Single Bit High Rotation tests passed while signle bit low rotation tests failed with bit 15 always 0

**Test Bench Code**

`default\_nettype none

`timescale 1ns**/**100ps

**module** testbench **();**

logic clk**;**

logic reset**;**

logic write**;**

logic **[**15**:**0**]** data\_in**;**

logic **[**2**:**0**]** address**;**

logic **[**15**:**0**]** data\_out**;**

typedef enum **{** READ\_REG**,** WRITE\_REG**,** RESET\_REG\_FILE **}** Action**;**

int error\_counter**;**

byte regFileResetCntDown**;**

int i**,** j**;**

//bit [15:0] val;

typedef struct **{**

string regname**;**

logic **[**15**:**0**]** reg\_value**;**

logic **[**15**:**0**]** reset\_value**;**

int bugs\_found**;**

**}** register**;**

register regmodel**[**8**];**

**initial** **begin**

clk **=** 0**;**

error\_counter **=** 0**;**

initRegModel**(**3'd0**,** "adc0\_reg"**,** 16'hffff**);**

initRegModel**(**3'd1**,** "adc1\_reg"**);**

initRegModel**(**3'd2**,** "temp\_sensor0\_reg"**);**

initRegModel**(**3'd3**,** "temp\_sensor1\_reg"**);**

initRegModel**(**3'd4**,** "analog\_test"**,** 16'hABCD**);**

initRegModel**(**3'd5**,** "digital\_test"**);**

initRegModel**(**3'd6**,** "amp\_gain"**);**

initRegModel**(**3'd7**,** "digital\_config"**,** 16'h1**);**

$display**(**"Testing inital Reset!"**);**

reset\_regfile**();**

check\_all\_registers**();**

**for(**i **=** 0**;** i **<** 8**;** i **=** i **+** 1**)** **begin**

test\_register**(**i**);**

**end**

foreach**(**regmodel**[**i**])** **begin**

$display**(**"Register: %s Bugs Found: %d"**,** regmodel**[**i**].**regname**,** regmodel**[**i**].**bugs\_found**);**

**end**

$display**(**"Error Count: %d"**,** error\_counter**);**

**end**

**task** automatic test\_register**(**int addr**);**

logic **[**15**:**0**]** val **=** 16'h7fff**;**

int i**;**

$display**(**"\*\*\*\*\*Testing %s @ %d\*\*\*\*\*\n\n"**,** regmodel**[**addr**].**regname**,** $time**);**

$display**(**"Test that each bit can be Reset"**);**

val **=** 16'h7fff**;**

**for(**i **=** 0**;** i**<**16**;** i**=**i**+**1**)** **begin**

$display**(**"Testing bit %d"**,** i**);**

val **=** rotate\_left**(**val**);**

write\_register**(**addr**,** val**);**

//Check that all registers have correct value

check\_all\_registers**();**

**end**

$display**(**"Test that each bit can be set"**);**

val **=** 16'h8000**;**

**for(**i **=** 0**;** i**<**16**;** i**=**i**+**1**)** **begin**

$display**(**"Testing bit %d"**,** i**);**

val **=** rotate\_left**(**val**);**

write\_register**(**addr**,** val**);**

//Check that all registers have correct value

check\_all\_registers**();**

**end**

//Write to Inverse of Reset Value

$display**(**"Testing Inverse Reset"**);**

write\_register**(**addr**,** **~(**regmodel**[**addr**].**reset\_value**));**

reset\_regfile**();**

check\_all\_registers**();**

**endtask;**

**function** automatic logic**[**15**:**0**]** rotate\_left**(**logic **[**15**:**0**]** sig**);**

logic **[**15**:**0**]** tmp**;**

tmp **=** **{**sig**[**14**:**0**],** sig**[**15**]};**

return tmp**;**

**endfunction**

**task** automatic initRegModel**(**logic **[**2**:**0**]** i**,** string name**,** logic **[**15**:**0**]** reset\_val **=** 16'd0**);**

//$display("in init regmodel");

regmodel**[**i**].**regname **=** name**;**

regmodel**[**i**].**reset\_value **=** reset\_val**;**

regmodel**[**i**].**bugs\_found **=** 0**;**

**endtask;**

**task** automatic rstRegModel**()** **;**

foreach **(** regmodel**[**i**]** **)** **begin**

regmodel**[**i**].**reg\_value **=** regmodel**[**i**].**reset\_value**;**

**end**

**endtask;**

**task** automatic reset\_regfile**();**

**@(negedge** clk**)**

reset **=** 1**;**

rstRegModel**();**

**@(negedge** clk**)**

reset **=** 0**;**

**endtask;**

**task** automatic write\_register**(input** bit **[**2**:**0**]** addr**,** **input** logic **[**15**:**0**]** data**);**

**@(negedge** clk**);**

write **=** 1**;**

data\_in **=** data**;**

address **=** addr**;**

regmodel**[**addr**].**reg\_value **=** data**;**

**@(negedge** clk**);**

write **=** 0**;**

**endtask;**

**task** automatic check\_all\_registers**();**

**for(**int i**=**0**;** i**<**8**;** i **=** i**+**1**)** **begin**

read\_register**(**i**);**

**end**

**endtask;**

**task** automatic read\_register**(input** bit **[**2**:**0**]** addr**);**

**@(negedge** clk**)**

address **=** addr**;**

**@(negedge** clk**)**

check\_register**(**addr**,** data\_out**);**

**endtask;**

**task** automatic check\_register**(**logic **[**2**:**0**]** addr**,** logic **[**15**:**0**]** val**);**

**if(!(**regmodel**[**addr**].**reg\_value **==** val**))** **begin**

$display**(**"Error: Register: \"%s\" Expected: %04x Actual: %04x\n"**,** regmodel**[**addr**].**regname**,** regmodel**[**addr**].**reg\_value**,** val**);**

regmodel**[**addr**].**bugs\_found **=** regmodel**[**addr**].**bugs\_found **+** 1**;**

error\_counter **=** error\_counter **+** 1**;**

regmodel**[**addr**].**reg\_value **=** val**;**

**end**

**endtask;**

**always** **begin**

**#**5 clk **=** **~**clk**;**

**end**

config\_reg conf**(** clk**,** reset**,** write**,** data\_in**,** address**,** data\_out**);**

endmodule