Assignment 5 verfication

Name :

Omar mohammed badr

Q1

1- Test plan

* test reset values .
* test writing 0000 in all registers.
* test writing FFFF in all registers.
*test 0 to 1 and 1 to 0 transition.
* test writing in a specified register and see if another register is affected .
* test reading and writing at the same time.
* test writing when write flag =0

2-Testbench code

```
module config_reg_tb();

    logic clk ;
    logic reset;
    logic write ;
    logic [15:0] data_in;
    logic [2:0] address;
    logic [15:0] data_out;
typedef enum bit [15:0] {adc0_reg,adc1_reg,temp_sensor0_reg,temp_sensor1_reg,analog_test,digital_test,amp_gain,digital_config} e_register;

config_reg DUT (clk,reset,write,data_in,address,data_out);

    initial begin
        clk = 0;
        forever
            #1 clk = ~clk ;
    end
e_register my_reg ;
assign address = my_reg;
    // code for reg 1,4,5,7 (error is the reset value)
    /*
    initial begin
        reset =0; #10;
        write = 0 ;
        reset =1; #10;

        my_reg=adc0_reg;#2          check_answer(16'hFFFF);#10 ;
        my_reg=adc1_reg;#2          check_answer(16'h0);#10 ;
        my_reg=temp_sensor0_reg;#2  check_answer(16'h0);#10 ;
        my_reg=temp_sensor1_reg;#2  check_answer(16'h0);#10 ;
        my_reg=analog_test;#2       check_answer(16'hABCD);#10 ;
        my_reg=digital_test;#2      check_answer(16'h0);#10 ;
        my_reg=amp_gain;#2          check_answer(16'h0);#10 ;
        my_reg=digital_config;#2    check_answer(16'h1);#10 ;
        $stop;
    end

*/
 */
    initial begin
        reset =1; #10;
        reset =0; #10;
        write = 1;#10;// here with write =0
        //my_reg=adc0_reg;          data_in=16'h0110;#2 check_answer(16'h0110);#10 ;
        //my_reg=adc1_reg;          data_in=16'h0110;#2 check_answer(16'h0110);#10 ;
        //my_reg=temp_sensor0_reg;  data_in=16'h0110;#2 check_answer(16'h0110);#10 ;
        //my_reg=temp_sensor1_reg;  data_in=16'h0110;#2 check_answer(16'h0110);#10 ;
        //my_reg=analog_test;       data_in=16'h0110;#2 check_answer(16'h0110);#10 ;

        //my_reg=amp_gain;          data_in=16'hA110;#2 check_answer(16'hA110);#10 ;//6

        //my_reg=digital_test;      data_in=16'h0110;#2 check_answer(16'h0110);#10 ;//5

        my_reg=digital_config;   data_in=16'h0110;#2 check_answer(16'h0110);#10 ;

        #500;
            // testing the transition from 0 to 1 for all bits
        my_reg=adc0_reg;          data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;
        my_reg=adc1_reg;          data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;
        my_reg=temp_sensor0_reg;  data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;
        my_reg=temp_sensor1_reg;  data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;
        my_reg=analog_test;       data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;
        my_reg=digital_test;      data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;
        my_reg=amp_gain;          data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;
        my_reg=digital_config;    data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;

        #500;
            // testing the transition from 1 to 0 for all bits
        my_reg=adc0_reg;          data_in=0;#2 check_answer(0);#10 ;
        my_reg=adc1_reg;          data_in=0;#2 check_answer(0);#10 ;
        my_reg=temp_sensor0_reg;  data_in=0;#2 check_answer(0);#10 ;
        my_reg=temp_sensor1_reg;  data_in=0;#2 check_answer(0);#10 ;
        my_reg=analog_test;       data_in=0;#2 check_answer(0);#10 ;
        my_reg=digital_test;      data_in=0;#2 check_answer(0);#10 ;
        my_reg=amp_gain;          data_in=0;#2 check_answer(0);#10 ;
```

Test case

```
        my_reg=digital_config;   data_in=16'h0110;#2 check_answer(16'h0110);#10 ;

        #500;
            // testing the transition from 0 to 1 for all bits
        my_reg=adc0_reg;         data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;
        my_reg=adc1_reg;         data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;
        my_reg=temp_sensor0_reg; data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;
        my_reg=temp_sensor1_reg; data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;
        my_reg=analog_test;      data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;
        my_reg=digital_test;     data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;
        my_reg=amp_gain;         data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;
        my_reg=digital_config;   data_in=16'hFFFF;#2 check_answer(16'hFFFF);#10 ;

        #500;
            // testing the transition from 1 to 0 for all bits
        my_reg=adc0_reg;         data_in=0;#2 check_answer(0);#10 ;
        my_reg=adc1_reg;         data_in=0;#2 check_answer(0);#10 ;
        my_reg=temp_sensor0_reg; data_in=0;#2 check_answer(0);#10 ;
        my_reg=temp_sensor1_reg; data_in=0;#2 check_answer(0);#10 ;
        my_reg=analog_test;      data_in=0;#2 check_answer(0);#10 ;
        my_reg=digital_test;     data_in=0;#2 check_answer(0);#10 ;
        my_reg=amp_gain;         data_in=0;#2 check_answer(0);#10 ;
        my_reg=digital_config;   data_in=0;#2 check_answer(0);#10 ;

        $stop;

    end

    task check_answer (input logic [15:0] data_out_expected);
            if (data_out != data_out_expected)
                $display("error message %t :data_out = %h, data_out_expected = %h ,data_in= %h, address = %d , my_reg %d",$time ,data_out ,data_out_
            //else if (data_out != data_out_expected)
                //$display(" correct result %t :C %d, expected C %d , A = %d , B %d",$time ,C ,C_expected,A,B);
        endtask
endmodule
```

3&4- bug reports and snippets

Reg0 (adc0_reg)

a) Design Input for Bug to Appear:

Write 0000 to register 0

b) Expected Behavior:

Bit 15 of register 0 is writable to a 0

c) Observed Behavior

Bit 15 of register 0 cannot be written to a 1

```
   32 :data_out = 8110, data_out_expected = 0110 ,data_in= 0110, address = 0 , my_reg     0
   44 :data_out = 1001, data_out_expected = 0110 ,data_in= 0110, address = 1 , my_reg     1
   56 :data_out = 0220, data_out_expected = 0110 ,data_in= 0110, address = 2 , my_reg     2
   92 :data_out = 0000, data_out_expected = a110 ,data_in= a110, address = 6 , my_reg     6
  104 :data_out = a110, data_out_expected = 0110 ,data_in= 0110, address = 5 , my_reg     5
  652 :data_out = fffe, data_out_expected = ffff ,data_in= ffff, address = 2 , my_reg     2
  688 :data_out = a110, data_out_expected = ffff ,data_in= ffff, address = 5 , my_reg     5
  712 :data_out = 7fff, data_out_expected = ffff ,data_in= ffff, address = 7 , my_reg     7
 1224 :data_out = 8000, data_out_expected = 0000 ,data_in= 0000, address = 0 , my_reg     0
 1284 :data_out = ffff, data_out_expected = 0000 ,data_in= 0000, address = 5 , my_reg     5
/omerb/OneDrive/Desktop/All about System Verilog/ASSIGNMENT5/testbenchh.sv(77)
```

Reg1 (adc1_reg)

a) Design Input for Bug to Appear:

Write 0110 to register 1

b) Expected Behavior:

output = 0110

c) Observed Behavior

output = 1001 (inverted)

```
   32 :data_out = 8110, data_out_expected = 0110 ,data_in= 0110, address = 0 , my_reg     0
   44 :data_out = 1001, data_out_expected = 0110 ,data_in= 0110, address = 1 , my_reg     1
   56 :data_out = 0220, data_out_expected = 0110 ,data_in= 0110, address = 2 , my_reg     2
   92 :data_out = 0000, data_out_expected = a110 ,data_in= a110, address = 6 , my_reg     6
  104 :data_out = a110, data_out_expected = 0110 ,data_in= 0110, address = 5 , my_reg     5
  652 :data_out = fffe, data_out_expected = ffff ,data_in= ffff, address = 2 , my_reg     2
  688 :data_out = a110, data_out_expected = ffff ,data_in= ffff, address = 5 , my_reg     5
  712 :data_out = 7fff, data_out_expected = ffff ,data_in= ffff, address = 7 , my_reg     7
 1224 :data_out = 8000, data_out_expected = 0000 ,data_in= 0000, address = 0 , my_reg     0
 1284 :data_out = ffff, data_out_expected = 0000 ,data_in= 0000, address = 5 , my_reg     5
'omerb/OneDrive/Desktop/All about System Verilog/ASSIGNMENT5/testbenchh.sv(77)
```

Reg2 (temp_sensor0_reg)

a) Design Input for Bug to Appear:

Write 0110 to register 2

b) Expected Behavior:

output = 0110

c) Observed Behavior

output = 0220 (shifted logically left by 1 bit)

```
  32 :data_out = 8110, data_out_expected = 0110 ,data_in= 0110, address = 0 , my_reg     0
  44 :data_out = 1001, data_out_expected = 0110 ,data_in= 0110, address = 1 , my_reg     1
  56 :data_out = 0220, data_out_expected = 0110 ,data_in= 0110, address = 2 , my_reg     2
  92 :data_out = 0000, data_out_expected = a110 ,data_in= a110, address = 6 , my_reg     6
 104 :data_out = a110, data_out_expected = 0110 ,data_in= 0110, address = 5 , my_reg     5
 652 :data_out = fffe, data_out_expected = ffff ,data_in= ffff, address = 2 , my_reg     2
 688 :data_out = a110, data_out_expected = ffff ,data_in= ffff, address = 5 , my_reg     5
 712 :data_out = 7fff, data_out_expected = ffff ,data_in= ffff, address = 7 , my_reg     7
1224 :data_out = 8000, data_out_expected = 0000 ,data_in= 0000, address = 0 , my_reg     0
1284 :data_out = ffff, data_out_expected = 0000 ,data_in= 0000, address = 5 , my_reg     5
merb/OneDrive/Desktop/All about System Verilog/ASSIGNMENT5/testbenchh.sv(77)
  Instance: /config_reg_tb
```

Reg3 (adc0_reg) ملحقتش اقفشه ☹

a) Design Input for Bug to Appear:

b) Expected Behavior:

c) Observed Behavior

Reg4 (analog_test)

a) Design Input for Bug to Appear:

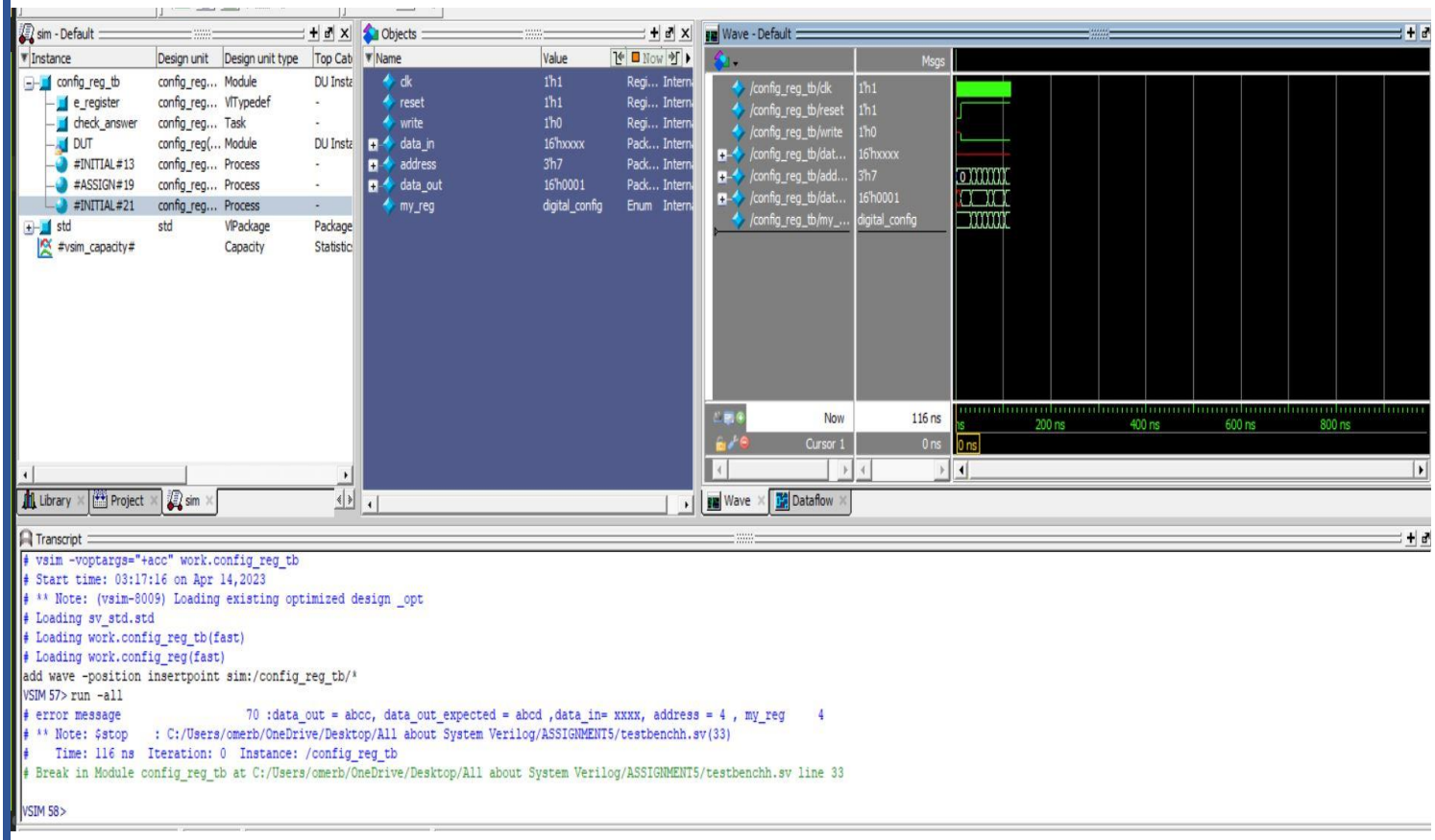
test the reset value (reset=1)

b) Expected Behavior:

output = ABCD

c) Observed Behavior

output = ABCC



Reg5 (digital_test)

a) Design Input for Bug to Appear:

Write any value to register 5

b) Expected Behavior:

the output follows the input

c) Observed Behavior

the output follows reg6 output

```
    32 :data_out = 0000, data_out_expected = a110 ,data_in= a110, address = 6 , my_reg     6
    44 :data_out = a110, data_out_expected = 0110 ,data_in= 0110, address = 5 , my_reg     5
   592 :data_out = fffe, data_out_expected = ffff ,data_in= ffff, address = 2 , my_reg     2
   628 :data_out = a110, data_out_expected = ffff ,data_in= ffff, address = 5 , my_reg     5
   652 :data_out = 7fff, data_out_expected = ffff ,data_in= ffff, address = 7 , my_reg     7
  1164 :data_out = 8000, data_out_expected = 0000 ,data_in= 0000, address = 0 , my_reg     0
  1224 :data_out = ffff, data_out_expected = 0000 ,data_in= 0000, address = 5 , my_reg     5
/omerb/OneDrive/Desktop/All about System Verilog/ASSIGNMENT5/testbenchh.sv(77)
 0  Instance: /config_reg_tb
o at C:/Users/omerb/OneDrive/Desktop/All about System Verilog/ASSIGNMENT5/testbenchh.sv line 77
```

Reg6 (amp_gain)

a) Design Input for Bug to Appear:

Write any value except FFFF to register 6

b) Expected Behavior:

the output follows the input

c) Observed Behavior

the output is stuck to 0000 unless the input is FFFF therefore the output follows the input

```
    32 :data_out = 0000, data_out_expected = a110 ,data_in= a110, address = 6 , my_reg     6
    44 :data_out = a110, data_out_expected = 0110 ,data_in= 0110, address = 5 , my_reg     5
   592 :data_out = fffe, data_out_expected = ffff ,data_in= ffff, address = 2 , my_reg     2
   628 :data_out = a110, data_out_expected = ffff ,data_in= ffff, address = 5 , my_reg     5
   652 :data_out = 7fff, data_out_expected = ffff ,data_in= ffff, address = 7 , my_reg     7
  1164 :data_out = 8000, data_out_expected = 0000 ,data_in= 0000, address = 0 , my_reg     0
  1224 :data_out = ffff, data_out_expected = 0000 ,data_in= 0000, address = 5 , my_reg     5
/omerb/OneDrive/Desktop/All about System Verilog/ASSIGNMENT5/testbenchh.sv(77)
 0  Instance: /config_reg_tb
o at C:/Users/omerb/OneDrive/Desktop/All about System Verilog/ASSIGNMENT5/testbenchh.sv line 77
```

Note:

No error as in the 500+ time the inputs are 0000 so all outputs are 0000 and in the 1000+ time the inputs are FFFF and the outputs are FFFF meaning that the case is just like I said . I could use extra snippets but I think that note will suffice .

Reg7 (digital_config)

a) Design Input for Bug to Appear:

Write FFFF to register 7

b) Expected Behavior:

Bit 15 of register 7 is writable to a 1

c) Observed Behavior

Bit 15 of register 7 cannot be written to a 1

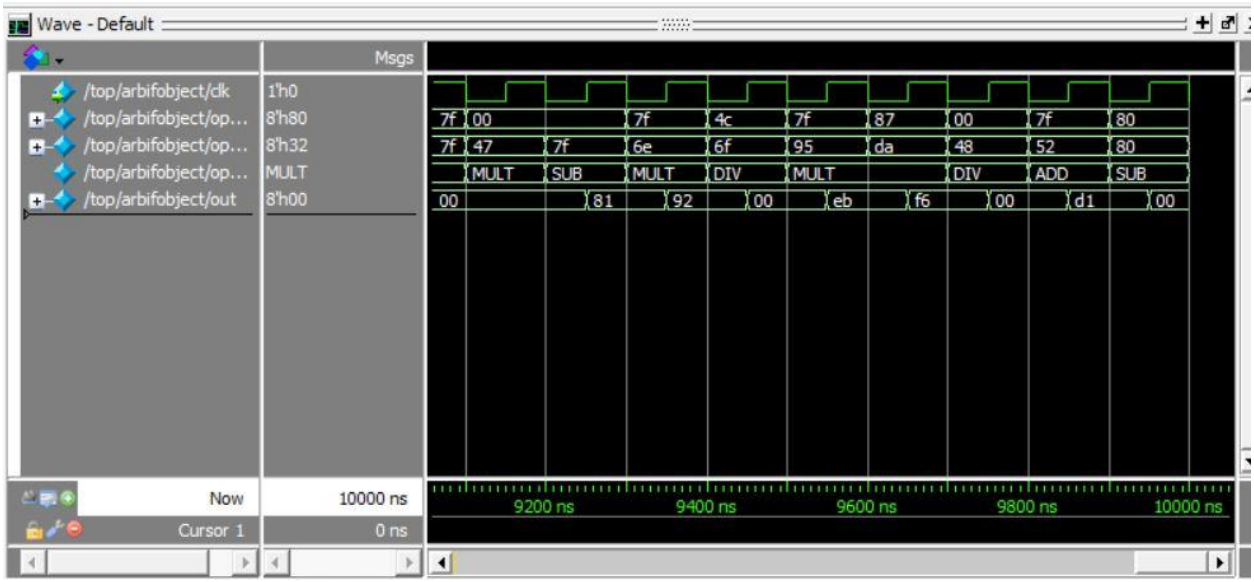
```
  32 :data_out = 8110, data_out_expected = 0110 ,data_in= 0110, address = 0 , my_reg     0
  44 :data_out = 1001, data_out_expected = 0110 ,data_in= 0110, address = 1 , my_reg     1
  56 :data_out = 0220, data_out_expected = 0110 ,data_in= 0110, address = 2 , my_reg     2
  92 :data_out = 0000, data_out_expected = a110 ,data_in= a110, address = 6 , my_reg     6
 104 :data_out = a110, data_out_expected = 0110 ,data_in= 0110, address = 5 , my_reg     5
 652 :data_out = fffe, data_out_expected = ffff ,data_in= ffff, address = 2 , my_reg     2
 688 :data_out = a110, data_out_expected = ffff ,data_in= ffff, address = 5 , my_reg     5
 712 :data_out = 7fff, data_out_expected = ffff ,data_in= ffff, address = 7 , my_reg     7
1224 :data_out = 8000, data_out_expected = 0000 ,data_in= 0000, address = 0 , my_reg     0
1284 :data_out = ffff, data_out_expected = 0000 ,data_in= 0000, address = 5 , my_reg     5
omerb/OneDrive/Desktop/All about System Verilog/ASSIGNMENT5/testbenchh.sv(77)
```

I’m so sorry for late .. again

Out of my control

Q2

1-Overview



2-the topmodule

```
module top();
    bit clk;
    always #50 clk = ~clk ;
    arb_if arbifobject(clk);
    alu_seq_with_if DUT(arbifobject);
    seq_alu_tb_with_if TB(arbifobject);
    monitor MONITOR(arbifobject);
endmodule
```

3-Monitor

```
module monitor (arb_if.MONITOR arbifobject);
    always@(posedge arbifobject.clk) begin
        $display("operand1=%d ,operand2=%d ,opcode = %s ,out = %d",arbifobject.operand1,arbifobject.operand2,arbifobject.opcode,arbifobject.out);
    end
endmodule
```

4- Package

```
package testing_pkg ;
    typedef enum {ADD,SUB,MULT,DIV} opcode_e ;

class transaction ;
    rand byte operand1 ;
    rand byte operand2 ;
    rand opcode_e opcode ;
    bit clk ;
    //byte out ;
    constraint operands_c {
        operand1 dist {127:/20,[-127:126]:/50,-128:/20,0:/10};
        operand2 dist {127:/20,[-127:126]:/50,-128:/20,0:/10};
    }
    function void print_all;
        $display("operand1 = %d,operand2 = %d,opcode = %s",operand1,operand2,opcode);
    endfunction
    // covergroup block
    covergroup CovCode @(posedge clk);
        // coverpoint for operands
        operand1_cp : coverpoint operand1 {
            bins maxpos = {127};
            bins maxneg = {-128};
            bins zero = {0};
            bins others = default ;
        }
        /*
        operand2_cp : coverpoint operand2 {
            bins maxpos = {127};
            bins maxneg = {-128};
            bins zero = {0};
            bins others = default ;
            }
```

```
        }
        /*
        operand2_cp : coverpoint operand2 {
            bins maxpos = {127};
            bins maxneg = {-128};
            bins zero = {0};
            bins others = default ;
            }
        */
        // coverpoint for opcode
        opcode_cp : coverpoint opcode{
            bins add = {ADD,SUB};
            //bins sub = {SUB};
            bins multiply = {MULT};
            illegal_bins division = {DIV};
        }
        cross_opcode_operand1_cp : cross opcode_cp ,operand1_cp{
        option.weight = 5;
        ignore_bins opcode_multiply = binsof(opcode_cp.multiply);
        ignore_bins operand1_zero = binsof(operand1_cp.zero);// equivelantely binsof(operand1_cp) intersect {0};
        }
    endgroup

    function new();
        CovCode = new();
    endfunction

endclass
endpackage
```

5-module

```
import testing_pkg::*;

module alu_seq_with_if(arb_if.DUT arbifobject);
/*
input byte operand1, operand2;
input clk;
input opcode_e opcode;
output byte out;
*/
always @(posedge arbifobject.clk) begin : proc_
    case (arbifobject.opcode)
        ADD: arbifobject.out = arbifobject.operand1 + arbifobject.operand2;
        SUB: arbifobject.out = arbifobject.operand1 - arbifobject.operand2;
        MULT: arbifobject.out = arbifobject.operand1 * arbifobject.operand2;
        DIV: arbifobject.out = arbifobject.operand1 / arbifobject.operand2;
        default: arbifobject.out = 0;
    endcase
end

endmodule
```

6-Interface

```
import testing_pkg::*;
interface arb_if (clk); // kda ana wa5ed el clk mn el top module

    input bit clk ;
    byte operand1, operand2;
    opcode_e opcode;
    byte out;

    modport TB (input clk,out,output operand1,operand2,opcode);
    modport DUT (input operand1,operand2,clk,opcode,output out);
    modport MONITOR(input clk,operand1,operand2,opcode,out);

endinterface
```

7-Testbench

```
import testing_pkg::*;

module seq_alu_tb_with_if(arb_if.TB arbifobject);
    // inst.. the class
    transaction tr ;

    //
    initial begin
        // first things first activate your object (tr)
        //transaction tr ;

        tr = new();

        repeat (100) begin
            @(negedge arbifobject.clk);
            assert(tr.randomize()); //randomize the whole object
            arbifobject.operand1 = tr.operand1;
            arbifobject.operand2 = tr.operand2;
            arbifobject.opcode = tr.opcode;
            tr.print_all();
        end
        $stop ;

    end
endmodule
```

Dofile

dofilee.do - Notepad

File Edit Format View Help

```
vlib work
vlog alu(2).sv testbench.sv +cover -covercells
vsim -voptargs=+acc work.top -cover
add wave *
add wave -position insertpoint  \
sim:/top/arbifobject/clk \
sim:/top/arbifobject/operand1 \
sim:/top/arbifobject/operand2 \
sim:/top/arbifobject/opcode \
sim:/top/arbifobject/out

coverage save top.ucdb -onexit
run -all
```

Coverage reports

```
 File alu(2).sv
------------------------------------CASE Branch-----------------------------------
   11                                     100     Count coming in to CASE
   12            1                         26       ADD: arbifobject.out = arbifobject.operand1 + arbifobject.operand2;

   13            1                         37       SUB: arbifobject.out = arbifobject.operand1 - arbifobject.operand2;

   14            1                         21       MULT: arbifobject.out = arbifobject.operand1 * arbifobject.operand2;

   15            1                         16       DIV: arbifobject.out = arbifobject.operand1 / arbifobject.operand2;

   16            1                   ***0***       default: arbifobject.out = 0;

Branch totals: 4 hits of 5 branches = 80.00%


Statement Coverage:
   Enabled Coverage              Bins      Hits    Misses  Coverage
   ----------------              ----      ----    ------  --------
   Statements                       6         5         1    83.33%

--------------------------------Statement Details--------------------------------
#
# ==============================================================================
# === Instance: /top/DUT
# === Design Unit: work.alu_seq_with_if
# ==============================================================================
# Branch Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Branches                         5         4         1    80.00%
#
# ================================Branch Details================================

==============================================================================
=== Instance: /top/TB
=== Design Unit: work.seq_alu_tb_with_if
==============================================================================

Assertion Coverage:
    Assertions                       1         1         0   100.00%
--------------------------------------------------------------------
Name                 File(Line)                   Failure      Pass
                                                  Count        Count
--------------------------------------------------------------------
/top/TB/#ublk#171972118#34/immed__36
                     testbench.sv(36)                   0          1
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       8         8         0   100.00%
```