MARIAM MOHAMED MARZOUK

ASSIGNMENT_4_Extended

Question 1

A configuration register is a bank of registers that is used to store setup information for the system (i.e. the configuration). All registers can be read and written and are 16-bits wide. The design contains 8 registers as defined in Table 1 along with their reset values. All bits are writable. The reset is active high.

The design has the following inputs:

- clk
- reset
- address
- data_in
- write: Write enable signal to enable sequential writing of the appropriate register below with data_in according to the address
- dataout: combinational dataout that holds the value of the appropriate register below according to the address

Register	Width	Address	Reset Value
adc0_reg	[15:0]	0	16'hFFFF
adc1_reg	[15:0]	1	16'h0
temp_sensor0_reg	[15:0]	2	16'h0
temp_sensor1_reg	[15:0]	3	16'h0
analog_test	[15:0]	4	16'hABCD
digital_test	[15:0]	5	16'h0
amp_gain	[15:0]	6	16'h0
digital_config	[15:0]	7	16'h1

Table 1: Config Register

Write a testplan using bullet points describing how you are going to test the design. Then write a self-checking testbench to perform the testing you described in your testplan.

Your testbench will:

1. Use a user defined enumerated type that hold the values of the registers
2. Declare a variable from the enum. This variable will be assigned to the address connected to the DUT.
3. Use associative array, as a golden model for the reset values of the registers.
    a. Name the array "reset_assoc"
    b. Key of the array is string
4. Use a task for reset and a task to check the result
5. Use the enum methods .num and .next (slide 78) inside of your loops to iterate on all of the registers

There is 1 bug per register. An encrypted version of the buggy configuration register code is provided. Compile this module just like an unencrypted module is compiled.

Identify the 8 bugs in config_reg_buggy.

For each bug, report:

a) Design Input for Bug to Appear:

E.g. Write FFFF to register x

b) Expected Behavior:

E.g. Bit 15 of register x is writable to a 1

c) Observed Behavior

E.g. Bit 15 of register x cannot be written to a 1

Remember, your bug report should contain enough detail for the designer of the configuration register to debug the problem.

I. Verification plan

Verification Step	Objective	Actions	Expected Outcome
Testbench Setup	Initialize clock and reset values	- Toggle clk every 1 time unit - Initialize reset_assoc with reset values	Clock toggles correctly, reset values are set
Reset Assertion	Verify reset values	- Assert reset_tb - Iterate through each register address - Check data_out against reset_assoc - Count errors and correct assertions	data_out matches expected reset values, errors and correct assertions are logged
Write Operation	Verify write functionality	- Deassert reset_tb - Set write_tb to 1 - Iterate through each register address - Write values to data_in_tb - Check data_out after each write - Count errors and correct assertions	data_out matches data_in_tb after each write, errors and correct assertions are logged
Assertions and Checks	Ensure correct behavior	- Use assert_reset task to check reset values - Use check_result task to verify write values	Errors are logged if data_out does not match expected values
Coverage	Ensure thorough testing	- Cover all register addresses - Cover all possible values for data_in_tb - Ensure all lines of code are executed	Comprehensive functional and code coverage
Error and Correct Count Reporting	Report results	- Display total errors and correct assertions at the end of the simulation	Summary of errors and correct assertions is displayed

II. RTL design

```
module config_reg(
      input  logic clk,
      input  logic reset,
      input  logic write,
      input  logic [15:0] data_in,
      input  logic [2:0] address,
      output logic [15:0] data_out
);

`protected

    MTI!#DU;Wt7W\E}vp$.h,RJ@F"cok2*:fU|]?KQ7"kDn2YBkJW1x7+D][a92\T\Xo+#4'mT1<VEU
    Y>-jEo-mnC;onUwzW]kB*s~TBlVA61@Ba}+7?|,5XoUXY[@7?]AEK#5u2[,vCY?5WACBAkE-5Q<A
    \1das*pl7+WpVrAz@eu$WA{ebXUv5TrV[v$A1+IB~*kJK~TXv^VW@]DQ]uzjv(G,i{LY]DZ$4^O,
    {cQ#=]/_xJpe_kwG?wE4qLAR5TU$3sI(^E#*,iTK7'RVBKXe\YOiZwxmlmAjGo2]1_1uC2}##l#I
    P#a7n^|e1lLPZ*l[3rTr=iEkb@T>J-pmux,$Z.\vQ2[-lJajw\sjirkh-'$[Ko7r*\?+$WWAB_~,
    ?e_<@'aB.a=Yi7aXRznB;\7ou~=!OV-p2^mDDu'sRiAn!m[J{Ho#-e1=Y7VDVT[G?v=$nL[:Kaw!
    EZTH}xUz5KaliVG][KZ2evwm~|tZIk3xeGs2o[GG]p[BAw!w>eQaTz*R1#TxX{n>^}!EC$ekE#Z7
    ,B5sWuxN^H\2w>3Y=2El3}xW2xWQk5[#Qviz*w1@s+^\UYY=-_nUw5s@-V2B-CjTp]Rw)e<UawXJ
    3+H*vpW@U{x{$R@W^$>,]E$v#BR{a+,uWuA}WxQ}X'JX[O5xkbl#B#/Dxo[*wV,FqZXmAy=I-\^n
    Br*@5+[Jzoh,$11KG>^9J*e~Hr?'TCNH}Bw~1[u@e$u>Q23#BR'$Wxl_]IpI*{J5?OKDrHkA$GsG
    a<j^D{*'Q^XzeaXL$Bsow<3*5lXBz*W*Z$CXMPK^zCvpDU5DT1aTJ=_Das=>o-TCs?aDE1}U5irC
    <A,~vQi}QOuo;!VY@]C-+ROh'?\Ku'pxvE!2(.Ss3JD'Q'eD?Esjp^[6]Z+u3n1]1urmi}@$eiDk
    l2\+73l-ExuR,n<*I!oW^;1BE;*DBxTpVWXBhq][<}=KoTyU>2jsY}R!U[sD+O>_{1U^Ev!27T\Y
    s}1q*$5T6kv!!hLEIwsx'CXe#GeQ*Y'[OvRkpr\a1oxGHpoW'lj}xRpjG!oTem?Q,mIp'7^11G]p
    A*mTLj;;\jrV[V*ps_T*jdrBX]tKE?X#n@<W[kp5XH]FNWoTBC]]Ud;5D[?][5D15}Tt_{]o#-Q*
    ^ep=(IH<oPD}QBJr<1p-'{sm5z@HBG,#rCiEQY}_1r[>;rr]u,wCKpGu}*w>lvvBr#j>s3Bz^I2x
    <\jOW1@Ii!H^[;!R=[}7iu=Z[am[jT>B>IkYmQVyG0,VAQ\E\[Y2*RR=w}o_O$BwT$I3G_OI[HT'
    3v@DZH1kp'@H0~EoI"'Kj3Wo'3*={jE$m*bo_}xBVs]E[psIrjABY+s]usrsi!\L#_#J/Vm\HVCe
    <o}K]@5,Yo!+Re77K>G-'^#si7;KBo[a@=i
`endprotected


endmodule // test


```

III. Test Bench

```
typedef enum logic [2:0]{adc0_reg = 3'b000, adc1_reg = 3'b001,temp_sensor0_reg = 3'b010,temp_sensor1_reg = 3'b011, analog_test = 3'b100,digital_test = 3'b101,
amp_gain = 3'b110, digital_config = 3'b111} reg_t;
module config_reg_tb();
logic clk;
logic write_tb;
logic reset_tb;
reg_t address_tb;
logic [15: 0] data_in_tb;
logic [15: 0] data_out;
int error_counter = 0, correct_counter = 0;
config_reg DUT(.clk(clk), .reset(reset_tb), .write(write_tb),.data_in(data_in_tb),.address(address_tb),.data_out(data_out));
bit [15: 0] reset_assoc [string];
initial begin
clk = 0;forever#1 clk = ~clk;end
initial begin
reset_assoc ["adc0_reg"] = 16'hFFFF;
reset_assoc ["adc1_reg"] = 16'h0;
reset_assoc ["temp_sensor0_reg"] = 16'h0;
reset_assoc ["temp_sensor1_reg"] = 16'h0;
reset_assoc ["analog_test"] = 16'hABCD;
reset_assoc ["digital_test"] = 16'h0;
reset_assoc ["amp_gain"] = 16'h0;
reset_assoc ["digital_config"] = 16'h1;
end
initial begin
data_in_tb = 0;
error_counter = 0;
correct_counter = 0;
write_tb = 0;
assert_reset();
write_tb = 1;
address_tb = address_tb.first();
for (int i = 0 ; i<(address_tb.last+1) ; i++) begin
@(negedge clk);
data_in_tb = 16'b0000000000000000;
check_result();
data_in_tb = 16'b0000000000000001;
check_result();
for (int i = 0 ; i< 15; i++) begin
data_in_tb = {data_in_tb[14:0],1'b0};
check_result();
end
```

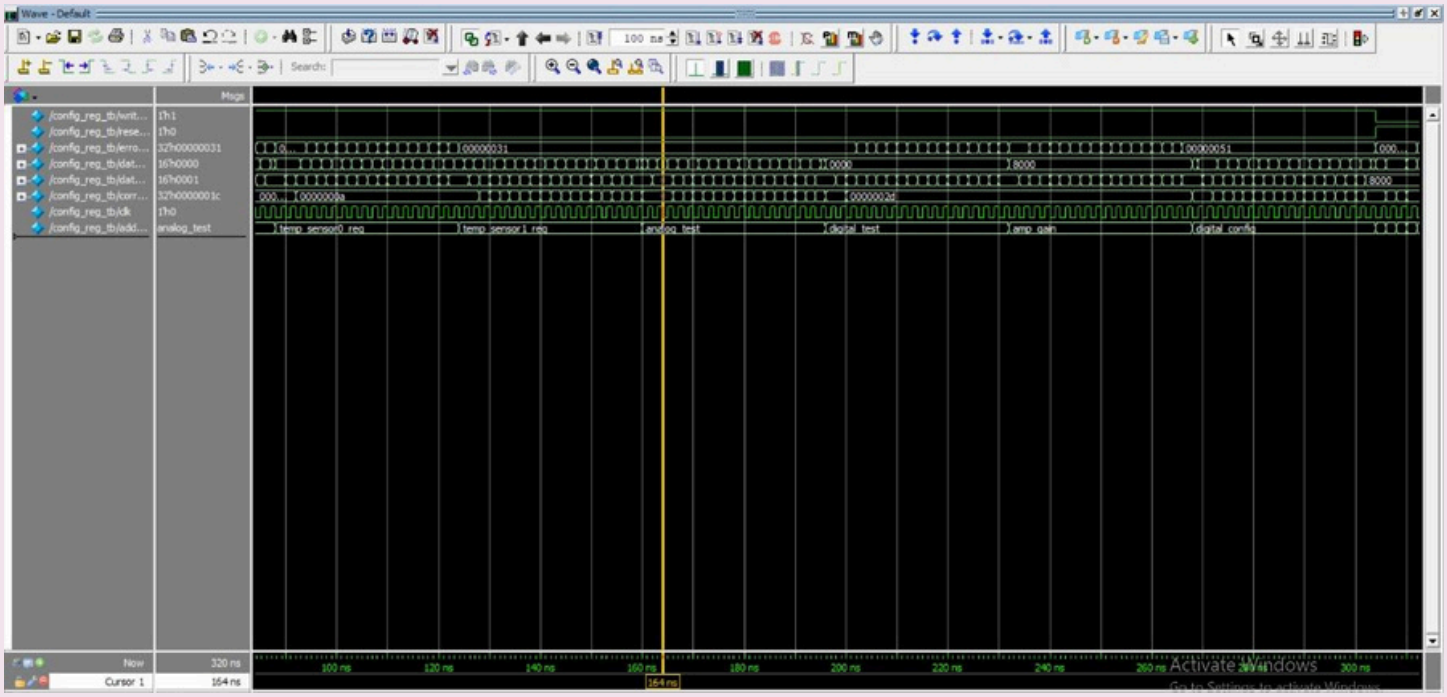
```
address_tb = address_tb.next();
end
write_tb = 0;
assert_reset();
$display("Error Counts = %0d and Correct Counts= %0d",error_counter, correct_counter);
$stop;
end
task assert_reset();
reset_tb = 1;
address_tb = address_tb.first();
for(int i = 0 ; i < (address_tb.last+1) ; i++) begin
@(negedge clk);
if(reset_assoc[address_tb.name()] !== data_out) begin
error_counter++;
$display("The Error is in The Reset reg %0s the Data_out is %0h and the expected is%0h", address_tb.name(), data_out, reset_assoc[address_tb. name()]);
end
else
correct_counter++;
address_tb = address_tb.next();
end
reset_tb =0;
endtask
task check_result();
@(negedge clk);
if (data_out !== data_in_tb) begin
error_counter++;
$display("Error in the Output Data %0b of reg %0s doesn?t match the input data %0b",data_out, address_tb.name(), data_in_tb);
end
else
correct_counter++;
endtask
endmodule
```

IV. Print

```
# The Error is in The Reset reg analog_test the Data_out is abcc and the expected is abcd
# Error in the Output Data 1000000000000000 of reg adc0_reg doesn?t match the input data 0
# Error in the Output Data 1000000000000001 of reg adc0_reg doesn?t match the input data 1
# Error in the Output Data 1000000000000010 of reg adc0_reg doesn?t match the input data 10
# Error in the Output Data 1000000000000100 of reg adc0_reg doesn?t match the input data 100
# Error in the Output Data 1000000000001000 of reg adc0_reg doesn?t match the input data 1000
# Error in the Output Data 1000000000010000 of reg adc0_reg doesn?t match the input data 10000
# Error in the Output Data 1000000000100000 of reg adc0_reg doesn?t match the input data 100000
# Error in the Output Data 1000000001000000 of reg adc0_reg doesn?t match the input data 1000000
# Error in the Output Data 1000000010000000 of reg adc0_reg doesn?t match the input data 10000000
# Error in the Output Data 1000000100000000 of reg adc0_reg doesn?t match the input data 100000000
# Error in the Output Data 1000001000000000 of reg adc0_reg doesn?t match the input data 1000000000
# Error in the Output Data 1000010000000000 of reg adc0_reg doesn?t match the input data 10000000000
# Error in the Output Data 1000100000000000 of reg adc0_reg doesn?t match the input data 100000000000
# Error in the Output Data 1001000000000000 of reg adc0_reg doesn?t match the input data 1000000000000
# Error in the Output Data 1010000000000000 of reg adc0_reg doesn?t match the input data 10000000000000
# Error in the Output Data 1100000000000000 of reg adc0_reg doesn?t match the input data 100000000000000
# Error in the Output Data 100000000 of reg adc1_reg doesn?t match the input data 1
# Error in the Output Data 1000000000 of reg adc1_reg doesn?t match the input data 10
# Error in the Output Data 10000000000 of reg adc1_reg doesn?t match the input data 100
# Error in the Output Data 100000000000 of reg adc1_reg doesn?t match the input data 1000
# Error in the Output Data 1000000000000 of reg adc1_reg doesn?t match the input data 10000
# Error in the Output Data 10000000000000 of reg adc1_reg doesn?t match the input data 100000
# Error in the Output Data 100000000000000 of reg adc1_reg doesn?t match the input data 1000000
# Error in the Output Data 1000000000000000 of reg adc1_reg doesn?t match the input data 10000000
# Error in the Output Data 1 of reg adc1_reg doesn?t match the input data 100000000
# Error in the Output Data 10 of reg adc1_reg doesn?t match the input data 1000000000
# Error in the Output Data 100 of reg adc1_reg doesn?t match the input data 10000000000
# Error in the Output Data 1000 of reg adc1_reg doesn?t match the input data 100000000000
# Error in the Output Data 10000 of reg adc1_reg doesn?t match the input data 1000000000000
# Error in the Output Data 100000 of reg adc1_reg doesn?t match the input data 10000000000000
# Error in the Output Data 1000000 of reg adc1_reg doesn?t match the input data 100000000000000
# Error in the Output Data 10000000 of reg adc1_reg doesn?t match the input data 1000000000000000
# Error in the Output Data 10 of reg temp_sensor0_reg doesn?t match the input data 1
# Error in the Output Data 100 of reg temp_sensor0_reg doesn?t match the input data 10
# Error in the Output Data 1000 of reg temp_sensor0_reg doesn?t match the input data 100
# Error in the Output Data 10000 of reg temp_sensor0_reg doesn?t match the input data 1000
# Error in the Output Data 100000 of reg temp_sensor0_reg doesn?t match the input data 10000
# Error in the Output Data 1000000 of reg temp_sensor0_reg doesn?t match the input data 100000
# Error in the Output Data 10000000 of reg temp_sensor0_reg doesn?t match the input data 1000000
# Error in the Output Data 100000000 of reg temp_sensor0_reg doesn?t match the input data 10000000
# Error in the Output Data 1000000000 of reg temp_sensor0_reg doesn?t match the input data 100000000
# Error in the Output Data 10000000000 of reg temp_sensor0_reg doesn?t match the input data 1000000000
# Error in the Output Data 100000000000 of reg temp_sensor0_reg doesn?t match the input data 10000000000
# Error in the Output Data 1000000000000 of reg temp_sensor0_reg doesn?t match the input data 100000000000
# Error in the Output Data 10000000000000 of reg temp_sensor0_reg doesn?t match the input data 1000000000000
```

```
Error in the Output Data 100000000000000 of reg temp_sensor0_reg doesn?t match the input data 10000000000000
Error in the Output Data 1000000000000000 of reg temp_sensor0_reg doesn?t match the input data 100000000000000
Error in the Output Data 0 of reg temp_sensor0_reg doesn?t match the input data 1000000000000000
Error in the Output Data 0 of reg digital_test doesn?t match the input data 1
Error in the Output Data 0 of reg digital_test doesn?t match the input data 10
Error in the Output Data 0 of reg digital_test doesn?t match the input data 100
Error in the Output Data 0 of reg digital_test doesn?t match the input data 1000
Error in the Output Data 0 of reg digital_test doesn?t match the input data 10000
Error in the Output Data 0 of reg digital_test doesn?t match the input data 100000
Error in the Output Data 0 of reg digital_test doesn?t match the input data 1000000
Error in the Output Data 0 of reg digital_test doesn?t match the input data 10000000
Error in the Output Data 0 of reg digital_test doesn?t match the input data 100000000
Error in the Output Data 0 of reg digital_test doesn?t match the input data 1000000000
Error in the Output Data 0 of reg digital_test doesn?t match the input data 10000000000
Error in the Output Data 0 of reg digital_test doesn?t match the input data 100000000000
Error in the Output Data 0 of reg digital_test doesn?t match the input data 1000000000000
Error in the Output Data 0 of reg digital_test doesn?t match the input data 10000000000000
Error in the Output Data 0 of reg digital_test doesn?t match the input data 100000000000000
Error in the Output Data 0 of reg digital_test doesn?t match the input data 1000000000000000
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 0
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 1
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 10
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 100
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 1000
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 10000
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 100000
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 1000000
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 10000000
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 100000000
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 1000000000
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 10000000000
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 100000000000
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 1000000000000
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 10000000000000
Error in the Output Data 1000000000000000 of reg amp_gain doesn?t match the input data 100000000000000
Error in the Output Data 0 of reg digital_config doesn?t match the input data 1000000000000000
The Error is in The Reset reg temp_sensor1_reg the Data_out is 8000 and the expected is 0
The Error is in The Reset reg analog_test the Data_out is abcc and the expected is abcd
Error Counts = 84 and Correct Counts= 68
```

V. Waveform


Wave - Default
temp_sensor0_reg
temp_sensor1_reg
analog_test
digital_test
amp_gain
digital_config
Now
Cursor 1
Activate Windows

VI. Bugs

1. Bug 1

adc0_reg:

Design Input for Bug to Appear:

Any input where bit 16 is not set to 1

Example: 16'h0001

Expected Behavior:

The output should match the input, maintaining bit 16 as 0

Example: Input 16'h0001 should result in output 16'h0001

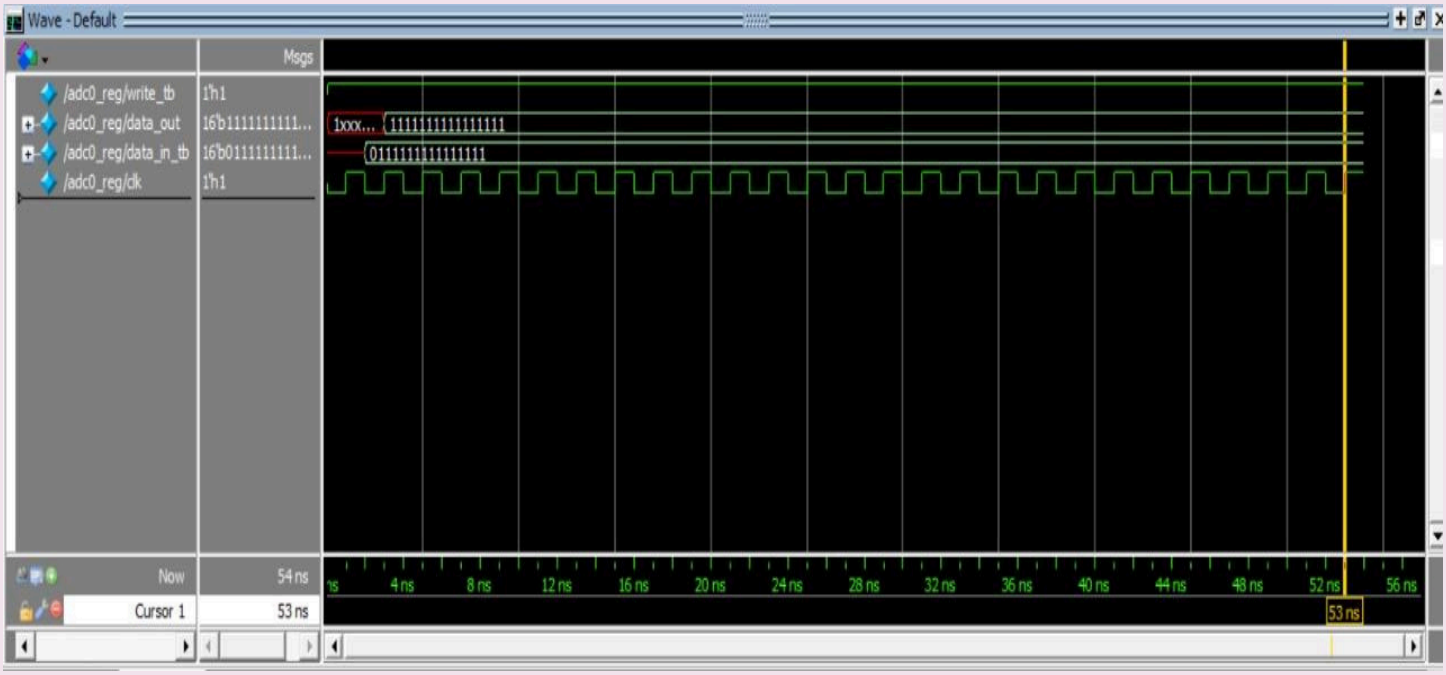
Observed Behavior:

Regardless of the input, bit 16 is always set to 1 in the output

Example: Input 16'h0001 results in output 16'h8001

- This bug appears to be related to the handling of bit 16 within the adc0_reg module. It suggests that there might be a hard-coded assignment or an unintended logic error forcing bit 16 to 1.

```
module adc0_reg();
reg clk, reset;
reg write_tb;
reg [15:0] data_in_tb;
wire [15:0] data_out;
logic [2:0] address_tb;
config_reg DUT
(. clk(clk),
. reset(reset),
. write(write_tb),
. data_in(data_in_tb),
. address(3'b000),
. data_out(data_out));
initial begin
clk = 0;
forever
#1 clk = ~clk;
end
initial begin
write_tb = 1;
@(negedge clk);
data_in_tb = 16'b0111_1111_1111_1111;
check_result();
#50;
$stop;
end
task check_result();
@(negedge clk);
if (data_out !== data_in_tb)
$display("Error in the Output Data %0b and doesn?t match the Expected data %0b",
data_out, data_in_tb);
else
$display("Correct Result");
endtask
endmodule
```



2. Bug 2

adc1_reg:

Design Input for Bug to Appear:

Any input that is not equal to 16'h0000

Example: 16'b0000_0000_0000_0001

Expected Behavior:

The output should match the input exactly

Example: Input 16'b0000_0000_0000_0001 should result in output 16'b0000_0000_0000_0001

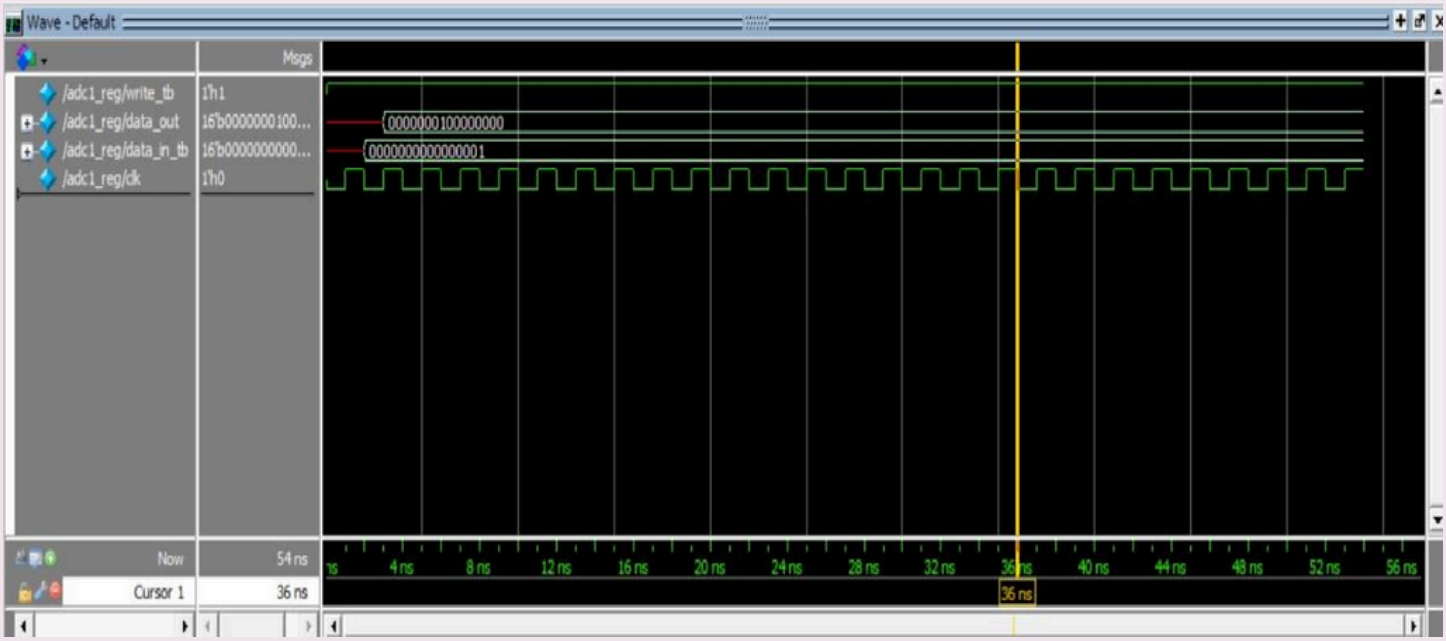
Observed Behavior:

The output is incorrectly rotated left by 8 bits

Example: Input 16'b0000_0000_0000_0001 results in output 16'b0000_0001_0000_0000

- This bug suggests that there might be an unintended shift or rotation operation occurring within the adc1_reg module

```
module adc1_reg ();
reg clk, reset;
reg write_tb;
reg [15: 0] data_in_tb;
wire [15: 0] data_out;
logic [2: 0] address_tb;
config_reg DUT
(. clk(clk),
. reset(reset),
. write(write_tb),
. data_in(data_in_tb),
. address(3'b001),
. data_out(data_out));
initial begin
clk = 0;
forever
#1 clk = ~clk;
end
initial begin
write_tb = 1;
@(negedge clk);
data_in_tb = 16'b0000_0000_0000_0001;
check_result();
#50;
$stop;
end
task check_result();
@(negedge clk);
if (data_out !== data_in_tb)
$display("Error in the Output Data %0b and doesn?t match the Expected data %0b",
data_out, data_in_tb);
else
$display("Correct Result");
endtask
endmodule
```



3. Bug 3

temp_sensor0_reg:

Design Input for Bug to Appear:

Any input that is not equal to 16'h0000

Example: 16'b0000_0000_0000_0100

Expected Behavior:

The output should match the input exactly

Example: Input 16'b0000_0000_0000_0100 should result in output 16'b0000_0000_0000_0100

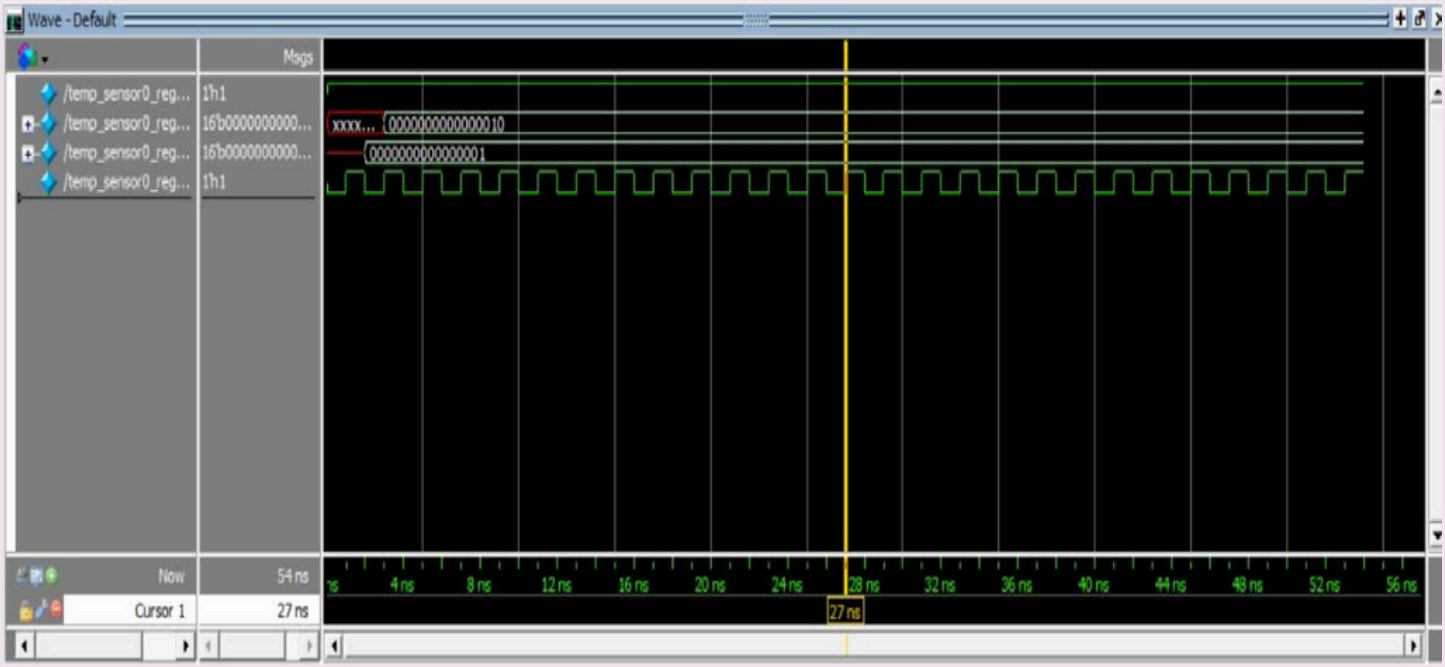
Observed Behavior:

The output is incorrectly shifted left by 1 bit

Example: Input 16'b0000_0000_0000_0100 results in output 16'b0000_0000_0000_1000

- This bug indicates that there might be an unintended shift operation occurring within the temp_sensor0_reg module.

```
module temp_sensor0_reg ();
reg clk, reset;
reg write_tb;
reg [15: 0] data_in_tb;
wire [15: 0] data_out;
logic [2: 0] address_tb;
config_reg DUT
(. clk(clk),
. reset(reset),
. write(write_tb),
. data_in(data_in_tb),
. address(3'b010),
. data_out(data_out));
initial begin
clk = 0;
forever
#1 clk = ~clk;
end
initial begin
write_tb = 1;
@(negedge clk);
data_in_tb = 16'b0000_0000_0000_0001;
check_result();
#50;
$stop;
end
task check_result();
@(negedge clk);
if (data_out !== data_in_tb)
$display("Error in the Output Data %0b and doesn?t match the Expected data %0b",
data_out, data_in_tb);
else
$display("Correct Result");
endtask
endmodule
```



4. Bug 4

temp_sensor1_reg:

Design Input for Bug to Appear:

Raising the reset signal to high

Expected Behavior:

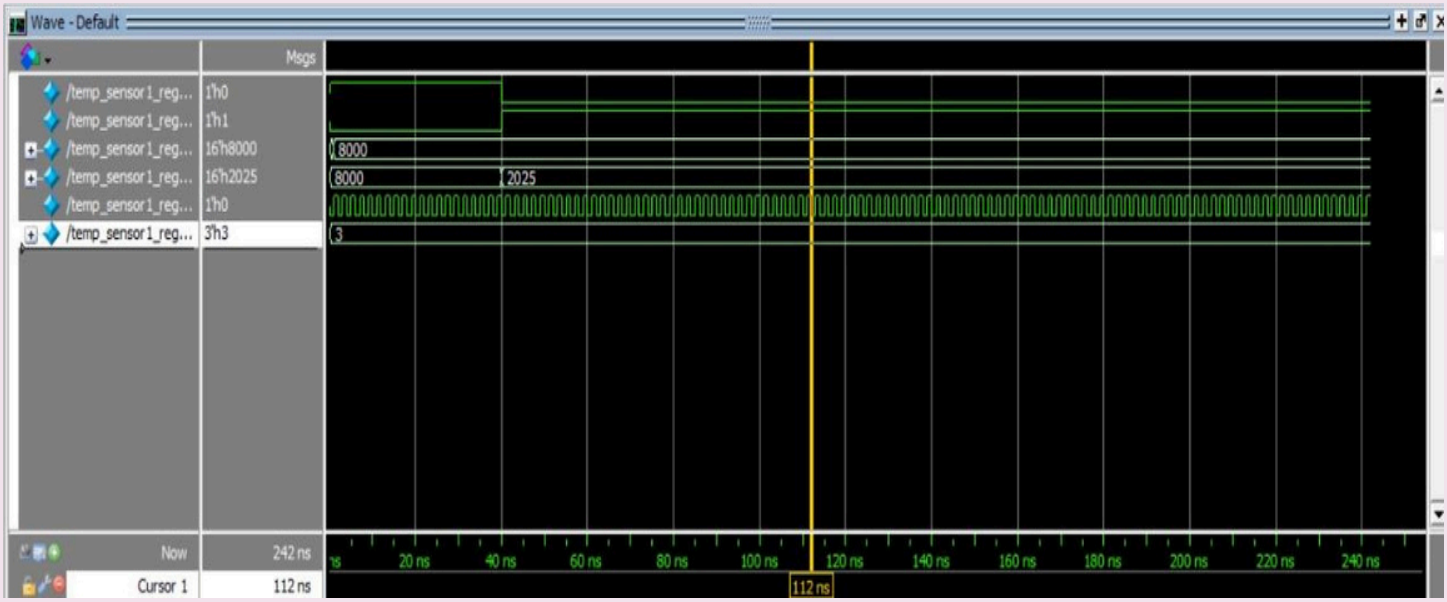
The register should reset and hold the value 16'h0

Observed Behavior:

The output incorrectly holds the value in data_in and does not reset

- This bug suggests that the temp_sensor1_reg module is not properly handling the reset condition. It appears that the reset logic is missing or incorrectly implemented, causing the register to retain its current value instead of resetting to 16'h0.

```
module temp_sensor1_reg ();
reg clk, reset;
reg write_tb;
reg [15: 0] data_in_tb;
wire [15: 0] data_out;
logic [2: 0] address_tb;
config_reg DUT
(. clk(clk),. reset(reset),. write(write_tb),. data_in(data_in_tb),. address(address_tb),. data_out(data_out));
initial begin
clk = 0;
forever
#1 clk = ~clk;
end
initial begin
address_tb = 3'b011;
data_in_tb = 16'h8000;
write_tb = 1;
reset = 0;
#40;
@(negedge clk);
address_tb = 3'b011;
data_in_tb = 16'h2025;
write_tb = 0;
reset = 1;
check_result(16'h0);
#200;
$stop;
end
task check_result(logic [15: 0] data_out_expected);
@(negedge clk);
if (data_out !== data_out_expected)
$display("Error in the Output Data %0b and doesn?t match the Expected data %0b",data_out, data_out_expected);
else
$display("Correct Result");
endtask
endmodule
```



5. Bug 5

analog_test:

Design Input for Bug to Appear:

Raising the reset signal to high

Expected Behavior:

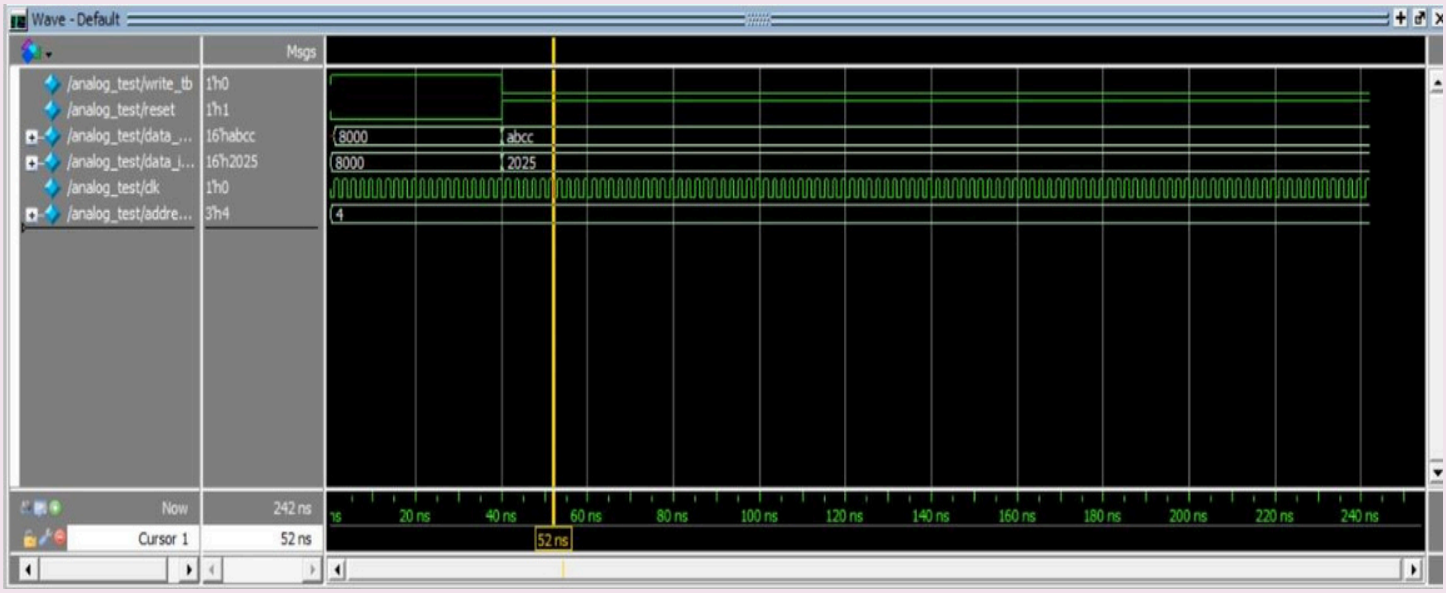
The register should reset and hold the value 16'hABCD

Observed Behavior:

The register incorrectly holds the value 16'hABCC

- This bug suggests that there might be an error in the reset value assignment within the analog_test module. It appears that the reset logic is not correctly setting the register to the expected value of 16'hABCD, possibly due to a typo or incorrect value in the design.

```
module analog_test ();
reg clk, reset;
reg write_tb;
reg [15: 0] data_in_tb;
wire [15: 0] data_out;
logic [2: 0] address_tb;
config_reg DUT
(. clk(clk),. reset(reset),. write(write_tb),. data_in(data_in_tb),. address(address_tb),. data_out(data_out));
initial begin
clk = 0;
forever
#1 clk = ~clk;
end
initial begin
address_tb = 3'b100;
data_in_tb = 16'h8000;
write_tb = 1;
reset = 0;
#40;
@(negedge clk);
data_in_tb = 16'h2025;
address_tb = 3'b100;
write_tb = 0;
reset = 1;
check_result(16'hABCD);
#200;
$stop;
end
task check_result(logic [15: 0] data_out_expected);
@(negedge clk);
if (data_out !== data_out_expected)
$display("Error in the Output Data %0b and doesn?t match the Expected data %0b",data_out, data_out_expected);
else
$display("Correct Result");
endtask
endmodule
```



6. Bug 6

digital_test:

Design Input for Bug to Appear:

Any input that is not equal to 16'h0000

Example: 16'b0000_0000_0000_0100

Expected Behavior:

The output should match the input exactly

Example: Input 16'b0000_0000_0000_0100 should result in output 16'b0000_0000_0000_0100

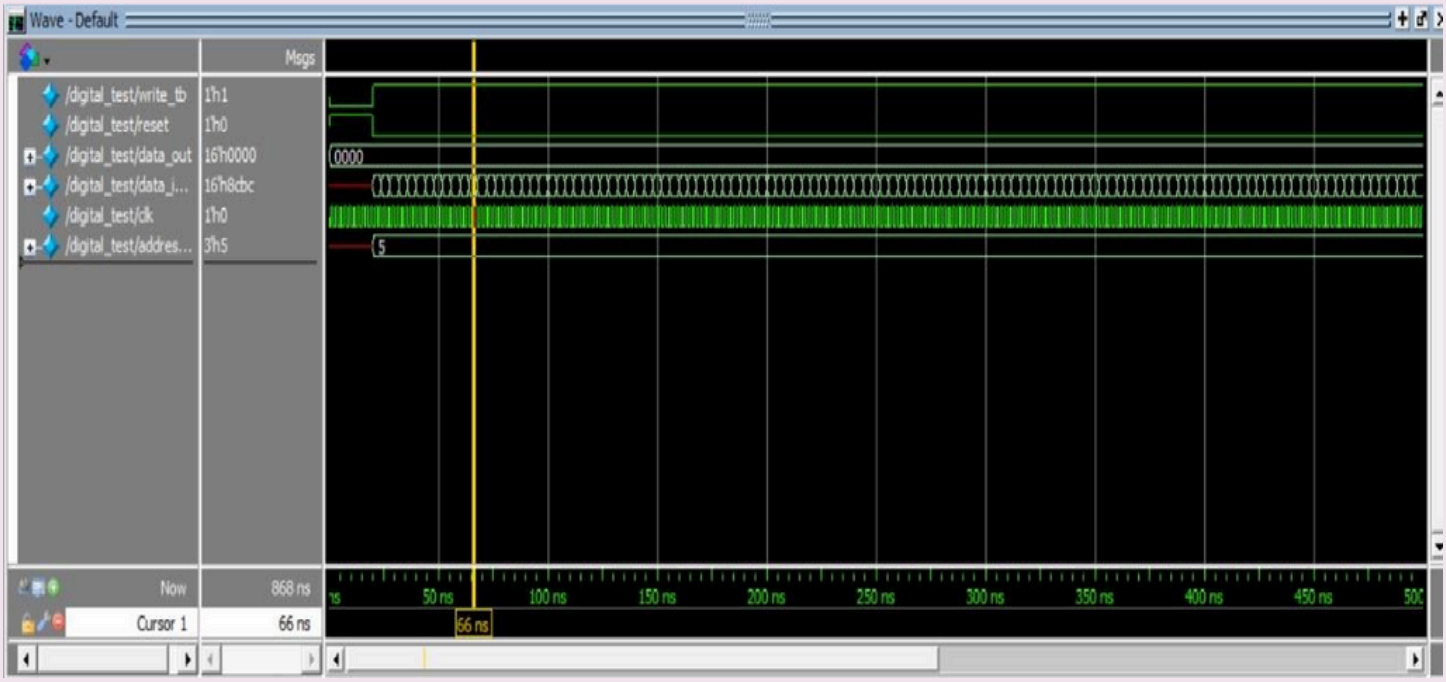
Observed Behavior:

The output is incorrectly shifted left by 1 bit and then assigned to 0

Example: Input 16'b0000_0000_0000_0100 results in output 16'b0000_0000_0000_0000

- The issue could be due to an incorrect assignment or a faulty condition that forces the output to zero regardless of the input

```
module digital_test ();
reg clk, reset;
reg write_tb;
reg [15: 0] data_in_tb;
wire [15: 0] data_out;
Logic [2: 0] address_tb;
config_reg DUT
(. clk(clk),. reset(reset),. write(write_tb),. data_in(data_in_tb),. address(address_tb),. data_out(data_out));
initial begin
clk = 0;
forever begin
#1 clk = ~clk ;
end
end
initial begin
write_tb = 0;
reset = 1;
#20;
reset = 0;
write_tb = 1;
repeat (200) begin
@(negedge clk);
address_tb = 3'b101;
data_in_tb = $urandom_range(1,16'hffff);
check_result();
end
#50;
$stop;
end
task check_result();
@(negedge clk);
if (data_out !== data_in_tb)
$display("Error in the Output Data %0b and doesn?t match the Expected data %0b",data_out, data_in_tb);
else
$display("Correct Result");
endtask
endmodule
```



7. Bug 7

amp_gain:

Design Input for Bug to Appear:

It takes the value of the previous register

Example: 16'b0000_0000_0000_0100

Expected Behavior:

The output should match the input exactly

Example: Input 16'b0000_0000_0000_0100 should result in output 16'b0000_0000_0000_0100

However, it is observed that the output incorrectly takes the last value of the previous register (digital_test), which is 16'h8000

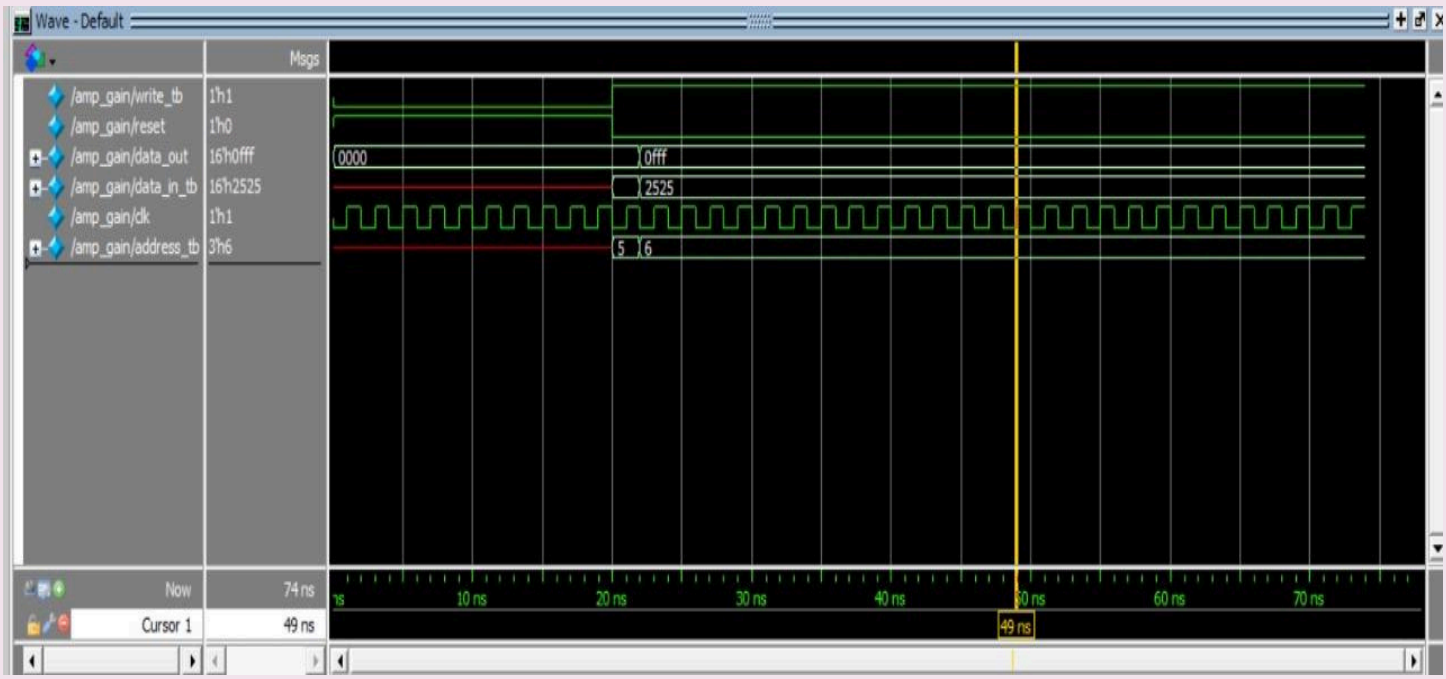
Observed Behavior:

The output is incorrectly assigned the value 16'h8000 instead of the input value

Example: Input 16'b0000_0000_0000_0100 results in output 16'h8000

- This bug suggests that the amp_gain module is not correctly processing the input value and is instead retaining or copying the value from the previous register (digital_test)

```
module amp_gain();
reg clk, reset;
reg write_tb ;
reg [15: 0] data_in_tb;
wire [15: 0] data_out;
logic [2: 0] address_tb;
config_reg DUT
(. clk(clk),. reset(reset),. write(write_tb), . data_in(data_in_tb),. address(address_tb),. data_out(data_out));
initial begin
clk = 0;
forever
#1 clk = ~clk;
end
initial begin
write_tb = 0;
reset = 1;
#20;
reset = 0;
write_tb = 1;
@(negedge clk);
data_in_tb = 16'h0fff;
address_tb = 3'b101;
@(negedge clk);
data_in_tb = 16'h2525;
address_tb = 3'b110;
check_result();
#50;
$stop;
end
task check_result();
@(negedge clk);
if (data_out !== data_in_tb)
$display("Error in the Output Data %0b and doesn?t match the Expected data %0b",data_out, data_in_tb);
else
$display("Correct Result");
endtask
endmodule
```



8. Bug 8

digital_config:

Design Input for Bug to Appear:

Any input where bit 16 is set to 1

Example: 16'b1000_0000_0000_0000

Expected Behavior:

The output should match the input exactly

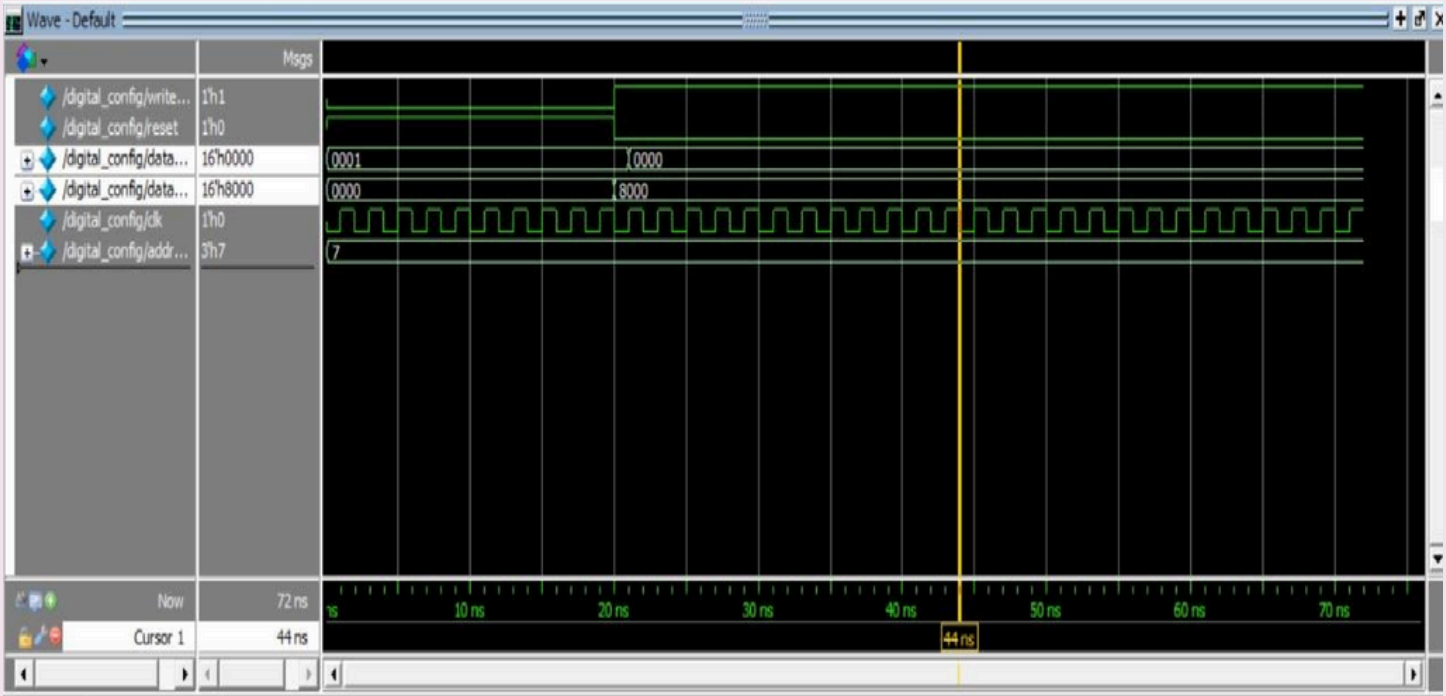
Example: Input 16'b1000_0000_0000_0000 should result in output 16'b1000_0000_0000_0000

Observed Behavior:

Bit 16 is always incorrectly assigned to 0, resulting in an output of 16'b0000_0000_0000_0000

- This bug suggests that there might be an unintended logic error within the digital_config module that forces bit 16 to 0.

```
module digital_config();
reg clk, reset;
reg write_tb ;
reg [15: 0] data_in_tb;
wire [15: 0] data_out;
logic [2: 0] address_tb;
config_reg DUT
(. clk(clk),. reset(reset),. write(write_tb),. data_in(data_in_tb),. address(address_tb),. data_out(data_out));
initial begin
clk = 0;
forever
#1 clk = ~clk;
end
initial begin
data_in_tb = 0;
address_tb = 3'b111;
write_tb = 0;
reset = 1;
#20;
write_tb = 1;
reset = 0;
@(negedge clk);
data_in_tb = 16'b1000_0000_0000_0000;
address_tb = 3'b111;
check_result();
#50;
$stop;
end
task check_result();
@(negedge clk);
if (data_out !== data_in_tb)
$display("Error in the Output Data %0b and doesn?t match the Expected data %0b",data_out, data_in_tb);
else
$display("Correct Result");
endtask
endmodule
```



Question 2

Q2. Write SVA properties for an arbiter design for the following specs:

1- Upon rising of the signal “request” by a master, the arbiter should raise the “grant” within 2 to 5 clock cycles.
2- Once the “grant” is raised, the master should acknowledge acceptance in the same clock cycle by lowering the "frame" and "irdy" signals.
3- Once the master completes the transaction it raises the "frame" and "irdy" signals, followed by that, the arbiter should lower the "grant" signal on the next clock cycle.


//QUESTION 2
assert property (@(posedge clk) $rose(request) |-> ##[2:5] Srose(grant));
assert property (@(posedge clk) $rose (grant) |-> (frame == 0 && irdy ==0));
assert property (@(posedge clk) ($rose (frame)&& $rose(irdy)) |=> (grant == 0));

Question 3

Q3. Suppose you are a design engineer and would like to write assertions to the internals of your design that has an FSM, write down the following SVA properties as per the following specs:

1- Current state internal signal “cs” will always stay one-hot irrespective of the input conditions.
2- If the current state “cs” is "IDLE" and if "get_data" input is raised, then the state is "GEN_BLK_ADDR" the following cycle and 64 cycles later the state should change to "WAITO."


1
2 //QUESTION 3
3 assert property (@(posedge clk) $onehot(cs));
4 assert property (@(posedge clk) (cs == IDLE && ($rose(get_data))) |=> (cs == GEN_BLK_ADDR) [*64] (cs == WAITO));
5