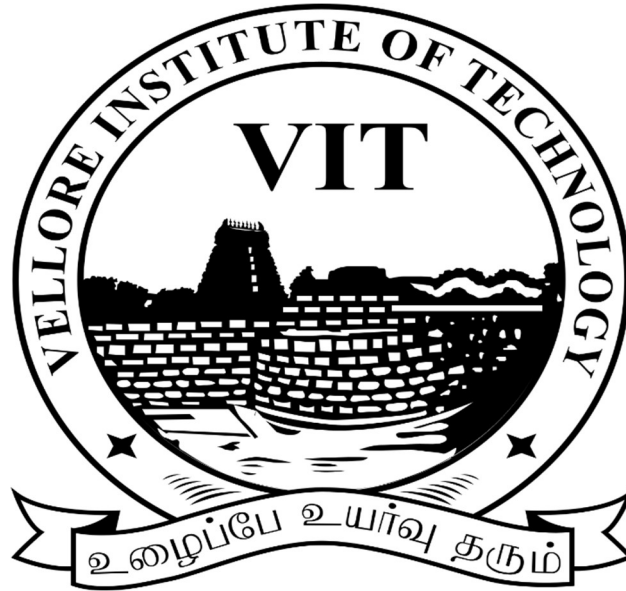


VELLORE INSTITUTE OF TECHNOLOGY : VELLORE

SCHOOL OF ELECTRONICS ENGINEERING



Winter Semester 2024-2025

MVLD505P - ASIC Design Lab

Lab Assessment -1

L23+24

Faculty name:

NITHISH KUMAR.V - SENSE

Name: CJ kiran

Reg no: 24MVD0166

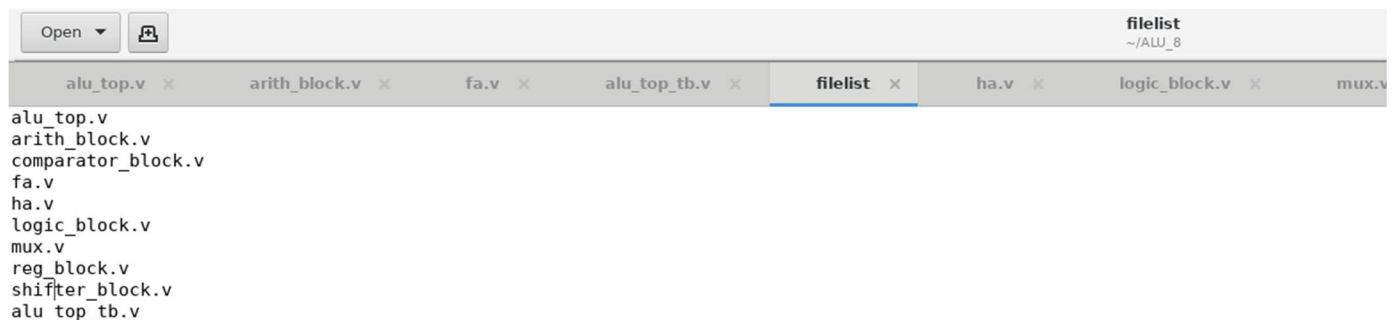
Aim: The aim of performing functional simulation for an ALU is to verify the correctness of its design by ensuring that it performs all intended arithmetic and logical operations accurately according to the specified functionality.

Procedure:

1. **Server Name** – 10.10.5.247
2. **Password** – student
3. **Setup the Environment** - Load Synopsys simulation tools and ensure the RTL file (ALU_8.v) and testbench are in the working directory.

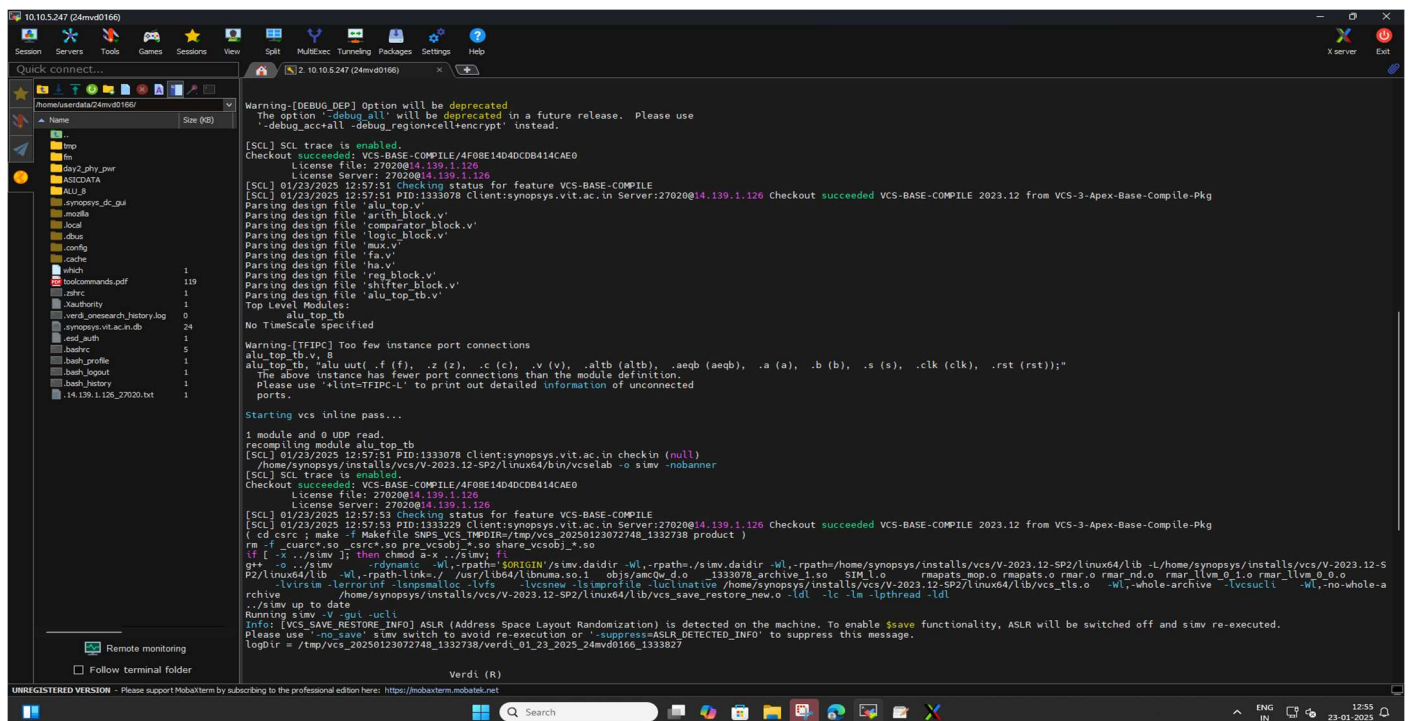
cp -rf /tmp/ALU_8 .

4. **Define Inputs and Outputs:** Set up the control signals, operand inputs, and expected outputs for each operation. In the testbench
5. Include all operations (addition, subtraction, logical AND/OR, etc.).
6. **Run the Simulation:** Execute the compiled simulation using ./run.
7. And verify the output waveform in the synopsys Verdi.
8. **Analyze Waveforms:** Inspect the output waveforms and compare them against expected results for each test case.
9. Copy the filelist from the ASICDATA/ folder.



The filelist Contains:

alu_top.v
arith_block.v
comparator_block.v
fa.v
ha.v
logic_block.v
mux.v
reg_block.v
shifter_block.v
alu_top_tb.v



```
Warning-[DEBUG_DEP] Option will be deprecated
The option '-debug_all' will be deprecated in a future release. Please use
'-debug_accrall -debug_regioncellencrypt' instead.


[SCL] SCL trace is enabled
Checkout succeeded: VCS-BASE-COMPILER/4F08E1404DCDB414CAE0
License file: 27020014.139.1.126
License Server: 27020014.139.1.126
[SCL] 01/23/2025 12:57:51 Checking status for feature VCS-BASE-COMPILER
[SCL] 01/23/2025 12:57:51 PID:1333078 Client:synopsys.vit.ac.in Server:27020014.139.1.126 Checkout succeeded VCS-BASE-COMPILER 2023.12 from VCS-3-Apex-Base-Compile-Pkg
Parsing design file 'alu_top.v'
Parsing design file 'arith_block.v'
Parsing design file 'comparator_block.v'
Parsing design file 'logic_block.v'
Parsing design file 'mux.v'
Parsing design file 'ha.v'
Parsing design file 'reg_block.v'
Parsing design file 'shifter_block.v'
Top Level Modules:
alu_top_tb
No TimeScale specified

Warning-[TFIPC] Too few instance port connections
alu_top_tb.v, 8
alu_top_tb: 'alu_uut(.f(f), .z(z), .c(c), .v(v), .altn(altn), .aegb(aegb), .a(a), .b(b), .s(s), .clk(clk), .rst(rst));'
The above instance has fewer port connections than the module definition.
Please use '+lint=TFIPC-L' to print out detailed information of unconnected
ports.

Starting vcs inline pass...
1 module and 0 upp read.
recompiling module alu_top_tb
[SCL] 01/23/2025 12:57:51 PID:1333078 Client:synopsys.vit.ac.in checkin (null)
/home/synopsys/installs/vcs/V-2023.12-SP2/linux64/bin/vcselab -o simv -nobanner
[SCL] SCL trace is enabled
Checkout succeeded: VCS-BASE-COMPILER/4F08E1404DCDB414CAE0
License file: 27020014.139.1.126
License Server: 27020014.139.1.126
[SCL] 01/23/2025 12:57:53 Checking status for feature VCS-BASE-COMPILER
[SCL] 01/23/2025 12:57:53 PID:1333229 Client:synopsys.vit.ac.in Server:27020014.139.1.126 Checkout succeeded VCS-BASE-COMPILER 2023.12 from VCS-3-Apex-Base-Compile-Pkg
( cd csrc ; make -f Makefile SNPS_VCS_TMPDIR=/tmp/vcs_20250123072748_1332738 product )
rm -f _csrc*.so _csrc*.so.pre_vcsobj.*.so share_vcsobj.*.so
if [ -x ./simv ]; then chmod +x ./simv; fi
g++ -o ./simv -rdynamic -Wl,-rpath=$ORIGIN/simv.daidir -Wl,-rpath=/home/synopsys/installs/vcs/V-2023.12-SP2/linux64/lib -L/home/synopsys/installs/vcs/V-2023.12-SP2/linux64/lib -Wl,-rpath-link=/usr/lib64/libnuma.so.1 -obj/am64.o -l333078.archive.1.so -lSIMI.o -lrapats_mep.o -lrapats.o -lrmr.o -lrmr_nd.o -lrmr_llve_0.1.o -lrmr_llve_0.0.o -lrvrsim -lerrortf -lspasmalloc -lvs -lvsnew -lvsimprofile -luclnative -lhome/synopsys/installs/vcs/V-2023.12-SP2/linux64/lib/vcs_tls.o -Wl,-whole-archive -lvsucl -Wl,-no-whole-archive
/home/synopsys/installs/vcs/V-2023.12-SP2/linux64/lib/vcs_save_restore_new.o -ldl -lc -lm -lpthread -ldl
./simv up to date
Running simv -V -gui -ucll
Info: [VCS SAVE RESTORE INFO] ASLR (Address Space Layout Randomization) is detected on the machine. To enable $save functionality, ASLR will be switched off and simv re-executed.
Please use '-no_save' simv switch to avoid re-execution or '-suppress=ASLR DETECTED INFO' to suppress this message.
logDir = /tmp/vcs_20250123072748_1332738/verdi_01_23_2025_24mvd0166_1333827

Verdi (R)
```

Copy the run file from the ASICDATA/



```
Open [icon]

alu_top.v x arith_block.v x fa.v x alu_top_tb.v x filelist x ha.v x lo

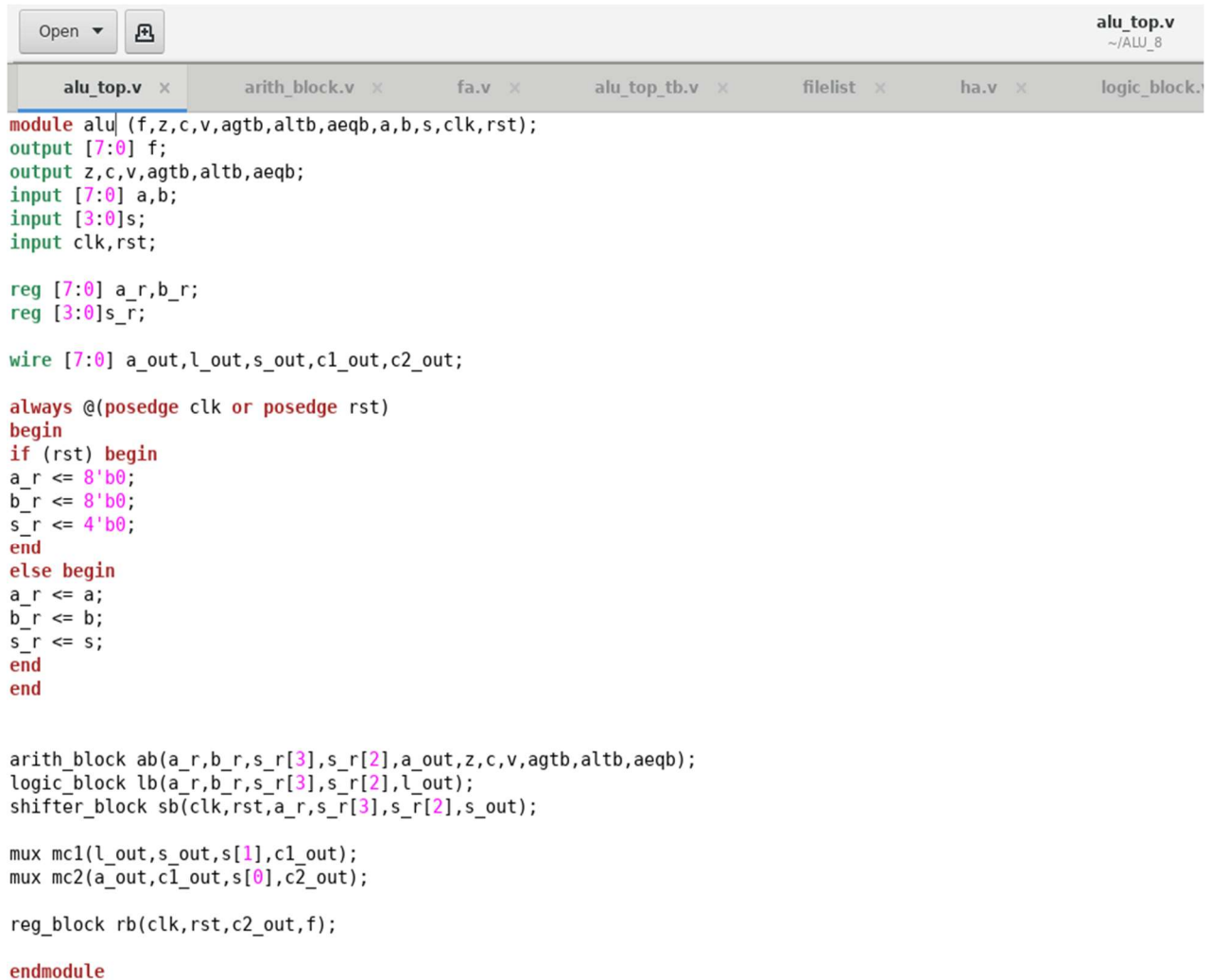
#!/bin/sh
vcs -V -R -f filelist -full64 -debug_all -kdb -O $1 -gui
```

The run file contains:

`#!/bin/sh`

`vcs -V -R -f filelist -full64 -debug_all -kdb -o $1 -gu`

ALU RTL Code:



The image shows a screenshot of a Verilog code editor. The editor has a menu bar with 'Open' and a search icon. The title bar shows 'alu_top.v' and the path '~/ALU_8'. The editor displays the following Verilog code:

```
module alu (f,z,c,v,agtb,altb,aeqb,a,b,s,clk,rst);
output [7:0] f;
output z,c,v,agtb,altb,aeqb;
input [7:0] a,b;
input [3:0] s;
input clk,rst;

reg [7:0] a_r,b_r;
reg [3:0] s_r;

wire [7:0] a_out,l_out,s_out,c1_out,c2_out;

always @(posedge clk or posedge rst)
begin
if (rst) begin
a_r <= 8'b0;
b_r <= 8'b0;
s_r <= 4'b0;
end
else begin
a_r <= a;
b_r <= b;
s_r <= s;
end
end

arith_block ab(a_r,b_r,s_r[3],s_r[2],a_out,z,c,v,agtb,altb,aeqb);
logic_block lb(a_r,b_r,s_r[3],s_r[2],l_out);
shifter_block sb(clk,rst,a_r,s_r[3],s_r[2],s_out);

mux mc1(l_out,s_out,s[1],c1_out);
mux mc2(a_out,c1_out,s[0],c2_out);

reg_block rb(clk,rst,c2_out,f);

endmodule
```

ALU RTL TB Code:

```
Open [icon] alu_top_tb.v ~/ALU_8
alu_top.v x arith_block.v x fa.v x alu_top_tb.v x filelist x ha.v x logic_block.v

module alu_top_tb();
wire [7:0] f;
wire z,c,v,agtb,altb,aeqb;
reg [7:0] a,b;
reg [3:0] s;
reg clk,rst;

alu uut (.f(f), .z(z), .c(c), .v(v), .altb(altb), .aeqb(aeqb), .a(a), .b(b), .s(s), .clk(clk), .rst(rst));

always #5 clk = ~clk;

initial
begin
clk = 0;
rst = 1;

a=8'b0;
b=8'b0;
s=4'b0;

#10 rst = 0;

{a, b, s} = {8'b00000001, 8'b00000010, 4'b0000};
#10;
{a, b, s} = {8'b11111111, 8'b00000001, 4'b0010};
#10;
{a, b, s} = {8'b00000010, 8'b00000001, 4'b0011};
#10;
{a, b, s} = {8'b00000010, 8'b00000001, 4'b0100};
#10;
{a, b, s} = {8'b00000010, 8'b00000001, 4'b0101};
#10;
{a, b, s} = {8'b00000010, 8'b00000001, 4'b0110};
#10;
{a, b, s} = {8'b00000010, 8'b00000001, 4'b1111};
#10;
{a, b, s} = {8'b00000010, 8'b00000001, 4'b1000};
#10;
{a, b, s} = {8'b00000010, 8'b00000001, 4'b1001};
#10;

$finish;
end
endmodule
```

arith_block.v:

```
Open [icon] alu_top.v x arith_block.v x fa.v x alu_top_tb.v

module arith_block(a,b,s3,s2,f,z,c,v,agtb,altb,aeqb);
input [7:0] a,b;
input s3,s2;
output [7:0] f;
output z,c,v,agtb,altb,aeqb;

wire [7:0] bbar,m1out,sum1,sum;
wire [7:1] cp;

assign bbar = ~b;
mux m1(b,bbar,s3,m1out);
assign sum1 = m1out+s3;

ha h1(sum[0],cp[1],a[0],sum1[0]);
fa f1(sum[1],cp[2],a[1],sum1[1],cp[1]);
fa f2(sum[2],cp[3],a[2],sum1[2],cp[2]);
fa f3(sum[3],cp[4],a[3],sum1[3],cp[3]);
fa f4(sum[4],cp[5],a[4],sum1[4],cp[4]);
fa f5(sum[5],cp[6],a[5],sum1[5],cp[5]);
fa f6(sum[6],cp[7],a[6],sum1[6],cp[6]);
fa f7(sum[7],c,a[7],sum1[7],cp[7]);

assign z = ~(|sum);
assign v = c^cp[7];

mux m2(sum,sum1,s2,f);

comparator_block cb(a,b,aeqb,altb,agtb);

endmodule
```

comparator_block.v



```
module comparator_block(A,B,aeqb,altb,agtb);
input [7:0] A,B;
output aeqb,altb,agtb;

assign agtb = (A>B)?1'b1:1'b0;
assign altb = (A<B)?1'b1:1'b0;
assign aeqb = (A==B)?1'b1:1'b0;

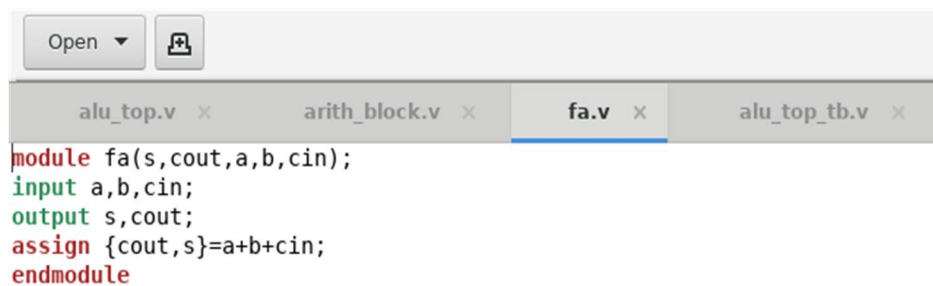
endmodule
```

ha.v



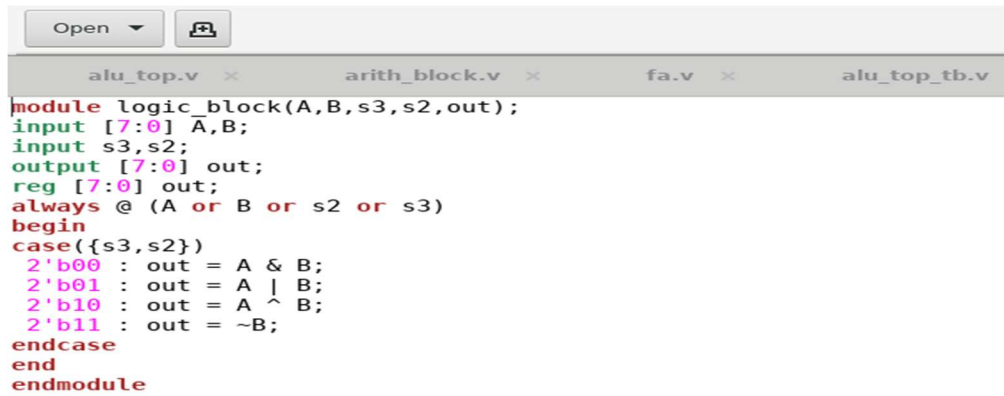
```
module ha(s,c,a,b);
input a,b;
output s,c;
assign s=a^b;
assign c=a&b;
endmodule
```

fa.v



```
module fa(s,cout,a,b,cin);
input a,b,cin;
output s,cout;
assign {cout,s}=a+b+cin;
endmodule
```

logic_block.v



The screenshot shows a text editor window with the file name 'logic_block.v' in the title bar. The code is written in Verilog and implements a logic block with two select inputs, s3 and s2, and two data inputs, A and B. The output is a 7-bit bus 'out'. The code uses a case statement to implement a 2-to-1 multiplexer based on the select inputs. The tabs at the top of the editor are 'alu_top.v', 'arith_block.v', 'fa.v', and 'alu_top_tb.v'.

```
module logic_block(A,B,s3,s2,out);
input [7:0] A,B;
input s3,s2;
output [7:0] out;
reg [7:0] out;
always @ (A or B or s2 or s3)
begin
case({s3,s2})
2'b00 : out = A & B;
2'b01 : out = A | B;
2'b10 : out = A ^ B;
2'b11 : out = ~B;
endcase
end
endmodule
```

mux.v



The screenshot shows a text editor window with the file name 'mux.v' in the title bar. The code is written in Verilog and implements a 2-to-1 multiplexer. It has two data inputs, i0 and i1, a select input 's', and a 7-bit output 'y'. The code uses an assign statement to implement the multiplexing logic. The tabs at the top of the editor are 'alu_top.v', 'arith_block.v', 'fa.v', and 'alu_top_t'.

```
module mux(i0,i1,s,y);
input [7:0] i0,i1;
input s;
output [7:0] y;
assign y = s?i1:i0;
endmodule
```

reg_block.v



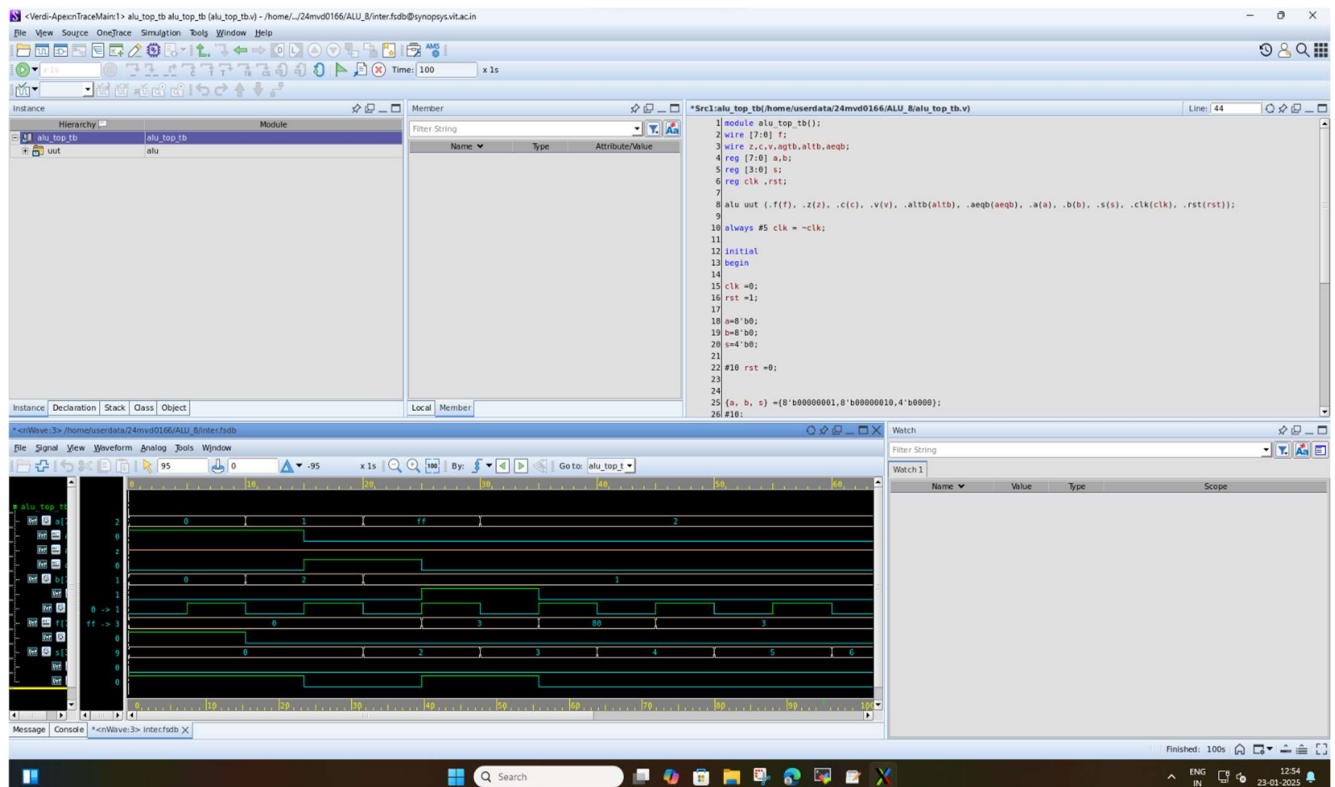
The screenshot shows a text editor window with the file name 'reg_block.v' in the title bar. The code is written in Verilog and implements a register block. It has a clock input 'clk', a reset input 'rst', an 8-bit data input 'in', and an 8-bit output 'out'. The code uses an always block with a posedge trigger to update the output 'out' to the value of 'in' on the clock edge, unless 'rst' is asserted, in which case 'out' is set to 8'b0. The tabs at the top of the editor are 'alu_top.v', 'arith_block.v', 'fa.v', and 'alu_top_tb.v'.

```
module reg_block(clk,rst,in,out);
input rst,clk;
input [7:0] in;
output [7:0] out;
reg [7:0] out;
always @(posedge clk or posedge rst)
if(rst)
out <= 8'b0;
else
out <= in;
endmodule
```

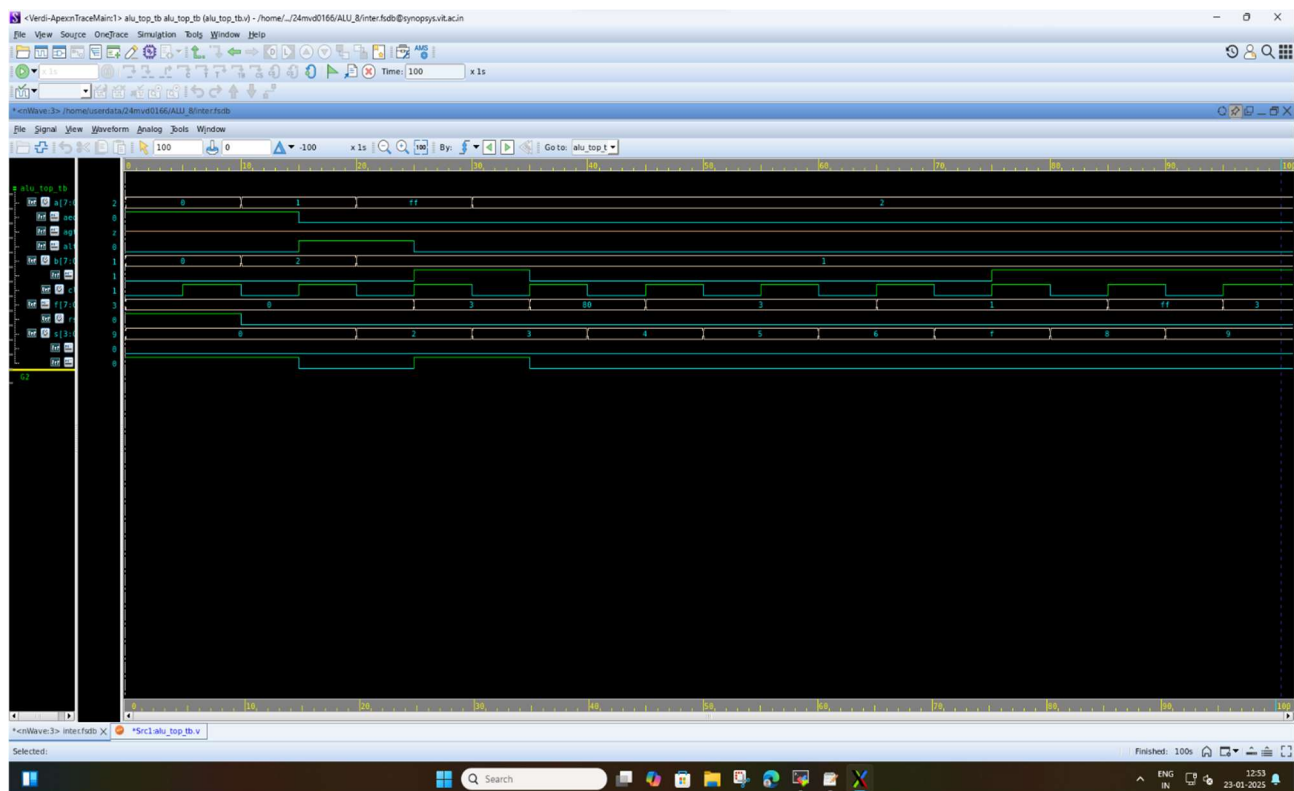
shifter_block.v

```
module shifter_block(clk,rst,in,s3,s2,sh);
input [7:0] in;
input s3,s2,clk,rst;
output [7:0] sh;
reg [7:0] sh;
always @(posedge clk or posedge rst)
begin
if(rst)
sh <= 0;
else begin
case({s3,s2})
2'b00: sh <= {in[0],in[7:1]};
2'b10: sh <= {in[6:0],in[7]};
2'b01: sh <= {1'b0,in[7:1]};
2'b11: sh <= {in[6:0],1'b0};
endcase
end
end
endmodule
```

10. Use the ./run command



Output waveform



Inference : The purpose of functional simulation for an ALU is to check if it performs all arithmetic and logical operations correctly as per the design specifications.