

# Digital Logic

Lab4 Verilog

# Lab4

- Verilog
  - Signed vs Unsigned
  - Practice1
- Sum of Minterms vs Product of Maxterms
  - loop in testbench: repeat, forever, for, while
  - Practice2
  - Practice3

# Signed vs Unsigned(1)

- Care should be taken when declaring the data types that may have to do arithmetic operation on it.

default to signed	default to unsigned
integer	reg wire

```
input signed [2:0] a;
reg signed [3:0] b;
output signed [3:0] c;
wire signed [1:0] d;
```

- Sign extension:** by increasing the number of bits of a binary number while preserving the number's sign (positive/negative) and value.
- Zero extension:** by setting the high bits of the destination to zero, rather than setting them to a copy of the most significant bit of the source.

```
wire signed [1:0] a_signed = 2'b11;
wire [1:0] a_unsigned = 2'b11;
wire [3:0] a_signed_extend = a_signed;
wire [3:0] a_unsigned_extend = a_unsigned;

initial begin
#10 $finish;
end
```

Name	Value	0 ns	10 ns
> a_signed[1:0]	11	11	
> a_unsigned[1:0]	11	11	
> a_signed_extend[3:0]	1111	1111	
> a_unsigned_extend[3:0]	0011	0011	

# Signed vs Unsigned(2)

```
wire signed [1:0] a_signed = -1; //2' b11
wire [1:0] a_unsigned = -1; //2' b11
wire [3:0] a_signed_extend = a_signed;
wire [3:0] a_unsigned_extend = a_unsigned;

initial begin
#10 $finish;
end
```

Name	Value	0 ns	10 ns
> a_signed[1:0]	11	11	
> a_unsigned[1:0]	11	11	
> a_signed_extend[3:0]	1111	1111	
> a_unsigned_extend[3:0]	0011	0011	

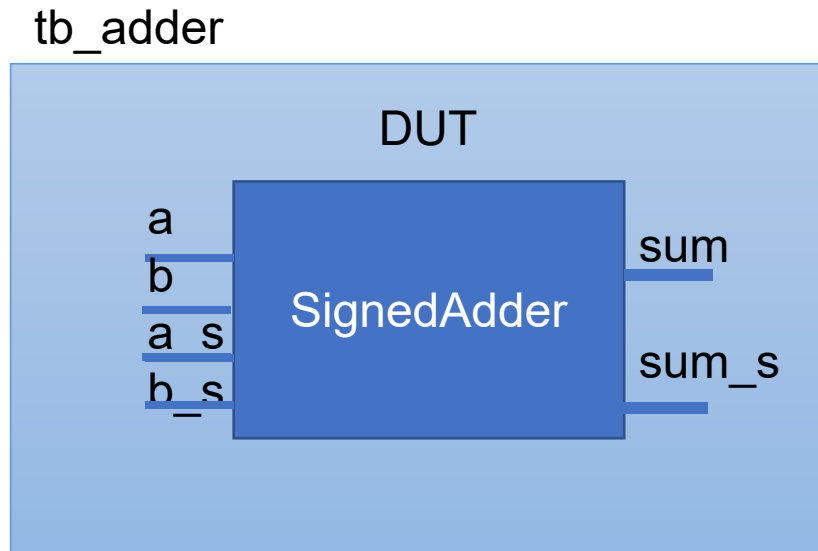
```
wire signed [1:0] a_signed = -1; //2' b11
wire [1:0] a_unsigned = -1; //2' b11
wire [3:0] a_signed_negative = -a_signed;
wire [3:0] a_unsigned_negative = -a_unsigned;

initial begin
#10 $finish;
end
```

Name	Value	0 ns	5 ns	10 ns
> a_signed[1:0]	11	11		
> a_unsigned[1:0]	11	11		
> a_signed_negative[3:0]	0001	0001		
> a_unsigned_negative[3:0]	1101	1101		

# Practice 1

- We design an adder using '+' operator.
  - inputs are 4bits unsigned/signed values: a, b, a\_s and b\_s
  - outputs are 8bits unsigned/signed values: sum and sum\_s



```
module SignedAdder(
    input [3:0] a,
    input [3:0] b,
    output [7:0] sum
    //complete with signed version
);

assign sum = a + b;
//complete with signed version
endmodule
```

```
`timescale 1 ns / 1 ps
module tb_adder();
    reg [3:0] a;
    reg [3:0] b;
    wire [7:0] sum;
    //complete with signed version

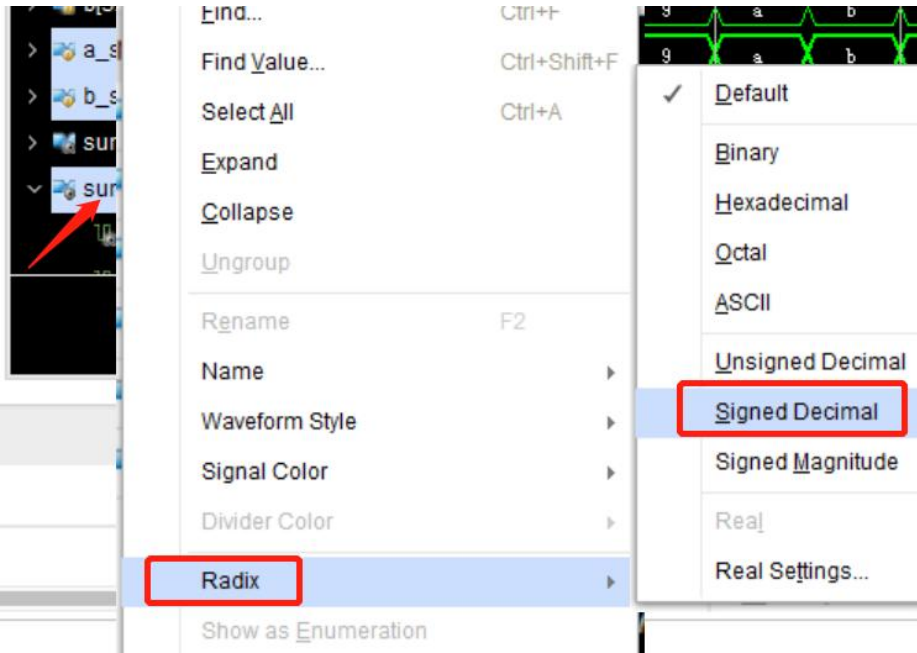
    SignedAdder DUT(
        .a(a),
        .b(b),
        .sum(sum)
        //complete with signed version
    );
    initial begin
        a = 0;
        b = 0;
        //complete with signed version
    end

    always begin
        #10
        a=a+1;
        b=b+1;
        //complete with signed version
    end
endmodule
```

# Practice 1

- Implement the design, run a simulation, and generate the RTL ANALYSIS schematic.
- Answer the following questions:
  - Q1: What are the exact values of a\_s, b\_s, and sum\_s?
  - Q2: Why are they represented in the same way as a, b, and sum in default radix mode?
  - Q3: What type of extension is performed on sum and sum\_s?
  - Q4: Provide your schematic and include a proof of your answer to Q3.

display in default radix mode



display in Signed Decimal mode





# Sum-of-product vs Sum-of-minterms

- $F(A,B,C)=A+B'C$
- $F(A,B,C)=AB(C'+C)+AB'(C'+C)+B'C(A'+A)=A'B'C+AB'C'+AB'C+ABC'+ABC$   
 $=m1+m4+m5+m6+m7=\Sigma(1,4,5,6,7)$

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Minterm	A	B	C	F
m0	0	0	0	0
m1	0	0	1	1
m2	0	1	0	0
m3	0	1	1	0
m4	1	0	0	1
m5	1	0	1	1
m6	1	1	0	1
m7	1	1	1	1

# Sum-of-minterms vs Product-of-maxterms

- $F(A,B,C) = A + B'C = AB(C' + C) + AB'(C' + C) + B'C(A' + A) = A'B'C + AB'C' + AB'C + ABC' + ABC$   
 $= m_1 + m_4 + m_5 + m_6 + m_7 = \sum(1, 4, 5, 6, 7)$

Minterm	A	B	C	F	F'
m0	0	0	0	0	1
m1	0	0	1	1	0
m2	0	1	0	0	1
m3	0	1	1	0	1
m4	1	0	0	1	0
m5	1	0	1	1	0
m6	1	1	0	1	0
m7	1	1	1	1	0

$$\begin{aligned} F(A,B,C) &= F(A,B,C)'' \\ &= (F(A,B,C))' \\ &= (\sum(1,4,5,6,7))' \\ &= (m_0 + m_2 + m_3)' \\ &= (A'B'C' + A'BC' + A'BC)' \\ &= (A'B'C')' \cdot (A'BC')' \cdot (A'BC)' \\ &= (A+B+C) \cdot (A+B'+C) \cdot (A+B'+C') \\ &= M_0 \cdot M_2 \cdot M_3 = \prod(0,2,3) \end{aligned}$$



# Design in dataflow of VERILOG

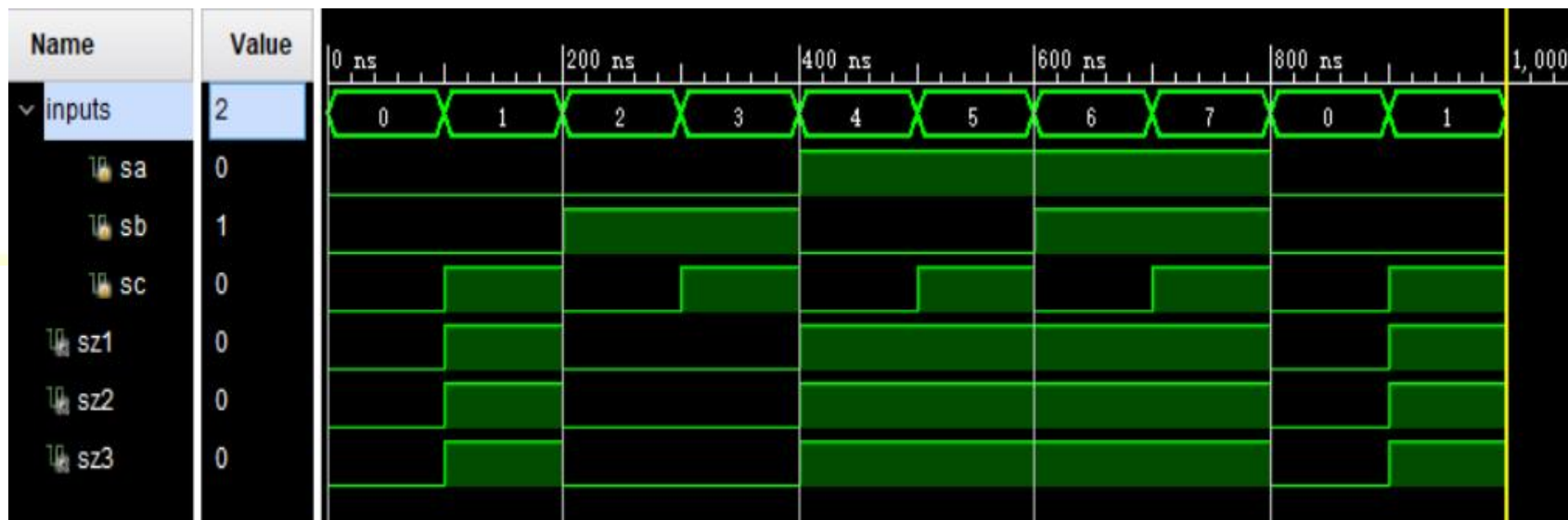
- $z1=F(A,B,C)=A+B'C$
- $z2=F(A,B,C)=\sum(1,4,5,6,7) = A'B'C+AB'C'+AB'C+ABC'+ABC$
- $z3=F(A,B,C)=\prod(0,2,3)= (A+B+C) \cdot (A+B'+C) \cdot (A+B'+C')$

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

```
module sop_som_pom(input a, b, c, output z1, z2, z3);  
    // a+b'c  
    assign z1 = a | (~b & c);  
    //m1(a'b'c) + m4(ab'c') + m5(ab'c) + m6(abc') + m7(abc)  
    assign z2 = (~a & ~b & c) | (a & ~b & ~c) | (a & ~b & c) | (a & b & ~c) | (a & b & c);  
    //M0(a+b+c) . M2(a+b'+c) . M3(a+b'+c')  
    assign z3 = (a | b | c) & (a | ~b | c) & (a | ~b | ~c);  
endmodule
```

# Testbench using loop(1)

```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////
module sop_som_pom_sim( );
reg sa, sb, sc;
wire sz1, sz2, sz3;
sop_som_pom u1(.a(sa),.b(sb),.c(sc),.z1(sz1),.z2(sz2),.z3(sz3));
initial begin
{sa, sb, sc} = 3'b0;
forever #100 {sa, sb, sc} = {sa, sb, sc} + 1;
end
endmodule
```



Tips on Vivado:

- Select the name of the ports you want to see in group (ctrl + select)
- Right click then choose the “New virtual Bus” to group all the selected ports, the name of the virtual bus could be edit

# Testbench using loop(2)

```
initial begin
    {sa, sb, sc} = 3'b0;
    repeat(7) begin
        #100 {sa, sb, sc} = {sa, sb, sc} + 1;
    end
    #100 $finish();
end
```

```
initial begin
    {sa, sb, sc} = 3'b0;
    for(integer i=0; i<7; i=i+1) begin
        #100 {sa, sb, sc} = {sa, sb, sc} + 1;
    end
    #100 $finish();
end
```

```
integer i=0;
initial begin
    {sa, sb, sc} = 3'b0;
    while(i<7) begin
        #100 {sa, sb, sc} = {sa, sb, sc} + 1;
        i=i+1;
    end
    #100 $finish();
end
```



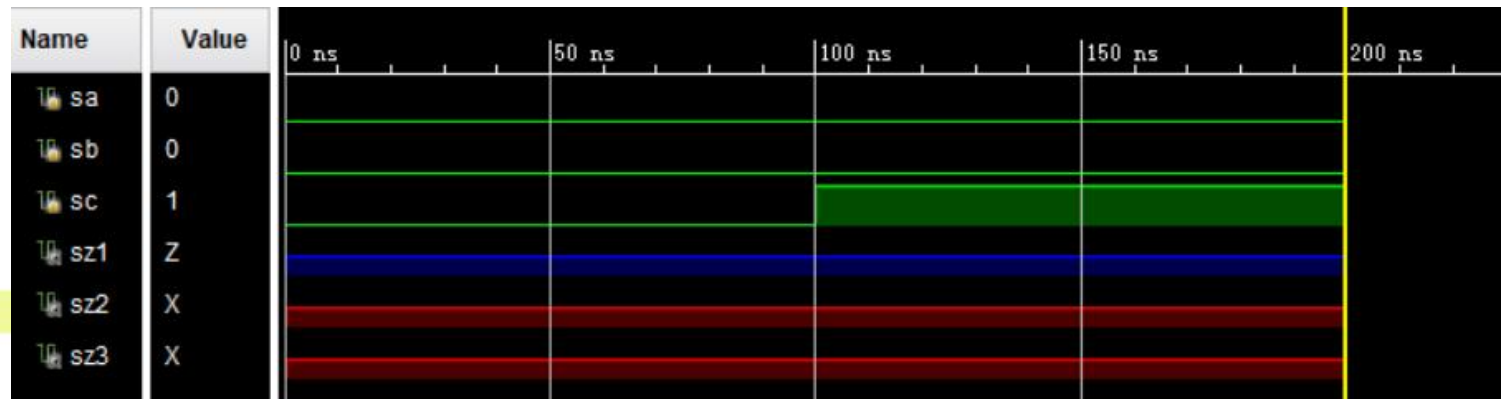
# Practice2

- Do the simulation on sop\_som\_pom\_sim which is the testbench of sop\_som\_pom, it is found that the waveform is not in line with expectations: 1) There are just two values of sa, sb and sc in the simulation, why ?
- 2) sz1 is Z, while the state of sz2 and sz3 is X, why?
- Modify the testbench to make it workable and test the function of module sop\_som\_pom

```

1  `timescale 1ns / 1ps
2  //////////////////////////////////////
21 module sop_som_pom_sim( );
22     reg sa, sb, sc;
23     wire sz1, sz2, sz3;
24     //sop_som_pom u1( . a(sa), . b(sb), . c(sc), . z1(sz1), . z2(sz2), . z3(sz3));
25     sop_som_pom u1(sa, sb, sz1, sz2, sz3);
26     /*...*/
36     initial begin
37         {sa, sb, sc} = 3'b0;
38         repeat(7); begin
39             #100 {sa, sb, sc} = {sa, sb, sc} + 1;
40         end
41         #100 $finish();
42     end
43     /*...*/
59 endmodule

```



# Practice3

Implement the digital logic circuit and test its function:  $F(x,y,z) = x^y z$

- Using structure or dataflow design style in verilog to implement the circuit design in Sum-of-Minterms or Product-of-Maxterms.
- Write the testbench in Verilog to verify the function of the design
- Generate the bitstream and program the device to test the function