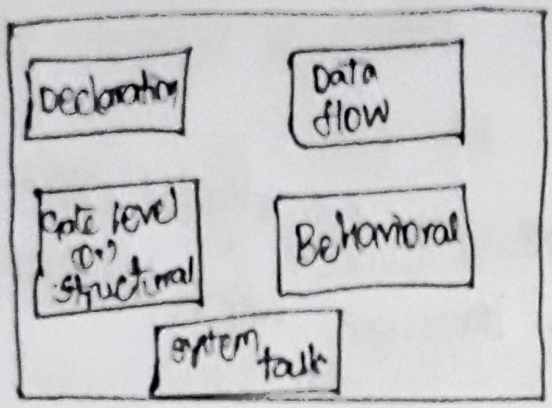
IAT-2

= x =

① Components of SR latch:



Declaration:- It is used to declare the module
= x = ? =

Ex:- module_name (a, b, Cout)

Data flow:- In data flow we will use assign operator
= ? =

Ex:- assign C = a ^ b

Gatelevel or structural:- In this there will be
= = = =

calling of main function

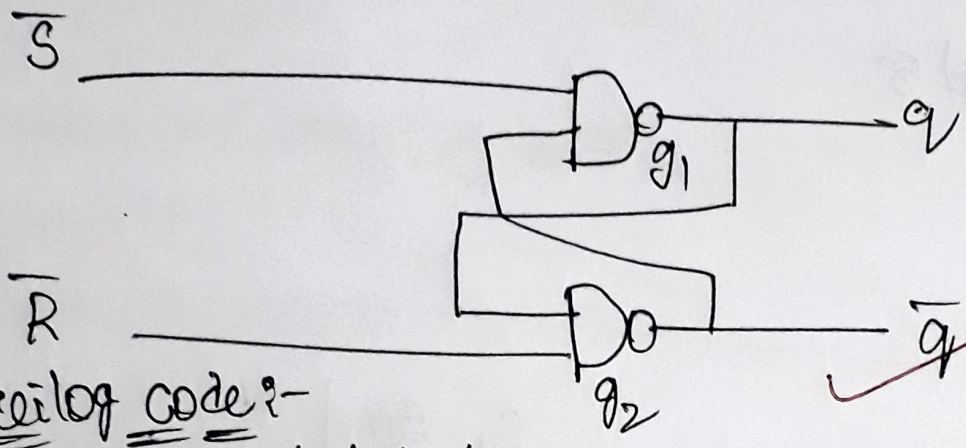
gate_name g1 (a, b, Cout)

Behavioral:- keyword @ always is used
= x =

System task:- where all the routine operations
= x =

Performed.

SR latch



S	R	$\bar{S}$	$\bar{R}$	q(t+1)
0	0	1	1	No ch
0	1	1	0	0
1	0	0	1	1
1	1	0	0	Toggle

Verilog code:-

```
module SR_latch (q, q_bar, S_bar, R_bar);
input S_bar, R_bar;
output q, q_bar;
nand g1 (q, S_bar, q_bar);
nand g2 (q_bar, R_bar, q);
endmodule
```

Stimuls code:

```
module SR_latch_tb;
reg S_bar, R_bar;
wire q, q_bar;
module SR_latch (q, q_bar, S_bar, R_bar);
initial
begin
$monitor ($time q=%b; q_bar=%b; S_bar=%b; R_bar=%b; q, q_bar, S_bar, R_bar);
    S_bar=0; R_bar=0; #5;
    S_bar=0; R_bar=1; #5;
```

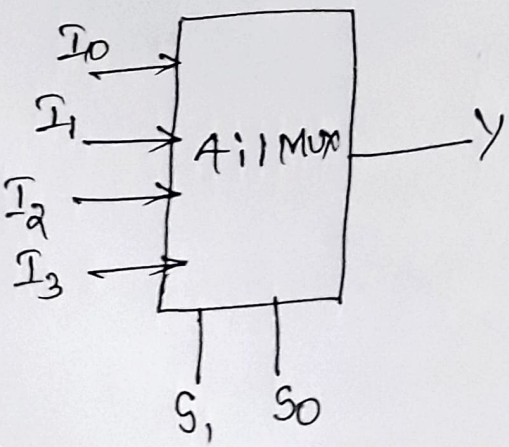
$\bar{S}=0$; $\bar{R}=1$; #5;

$\bar{S}=1$; $\bar{R}=1$; #5;

end

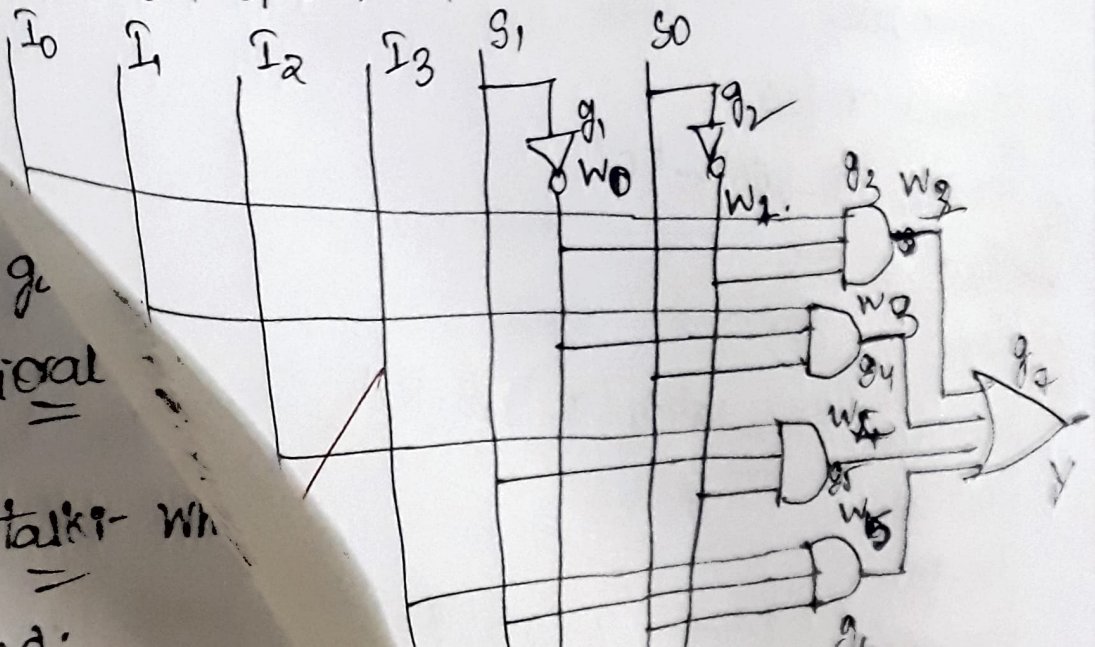
endmodule

③



S_1	S_0	y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$$y = \bar{S_1}\bar{S_0}I_0 + \bar{S_1}S_0I_1 + S_1\bar{S_0}I_2 + S_1S_0I_3$$



Ga
=
calling
g
Behavioral
= x =
System tasks wh
= x =

S = 0 ; R = 1 ; No change

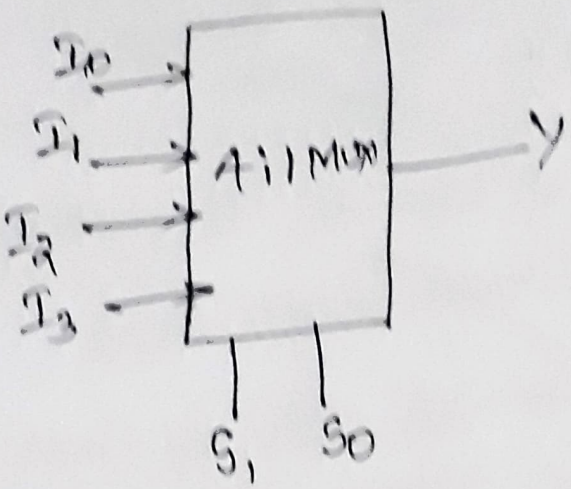
S = 1 ; R = 1 ; Not allowed

1

and

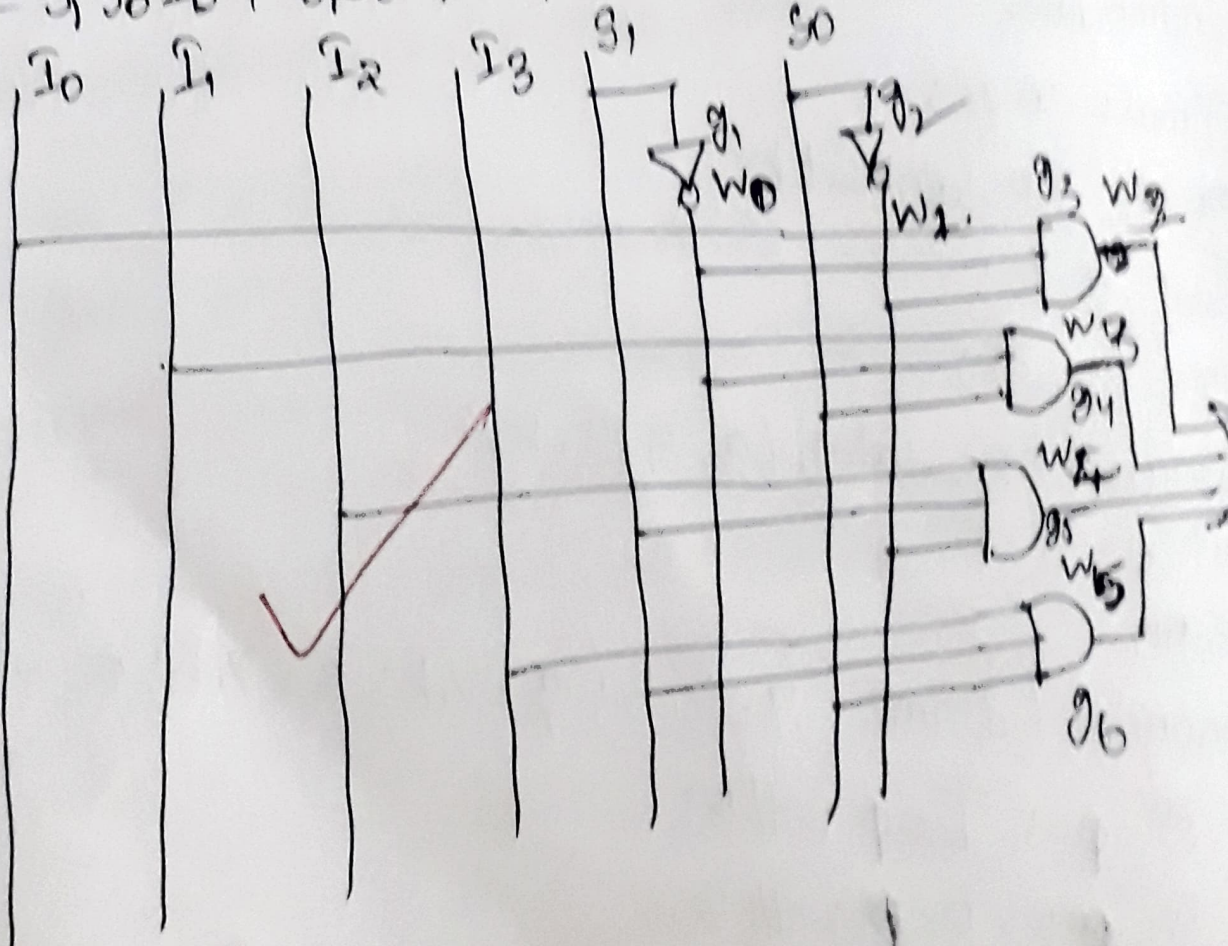
combinable

(3)



S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$$Y = \bar{S}_1 \bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1 \bar{S}_0 I_2 + S_1 S_0 I_3$$



~~module 4:1 MUX~~

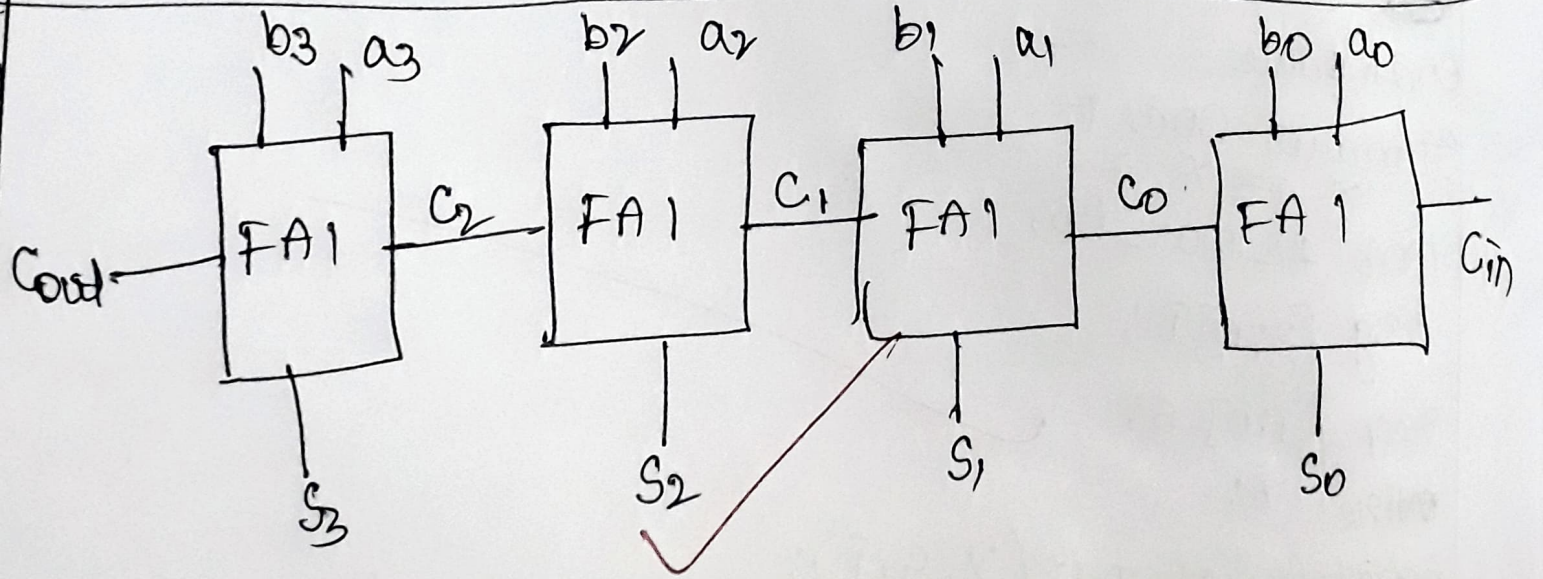
Verilog code:-

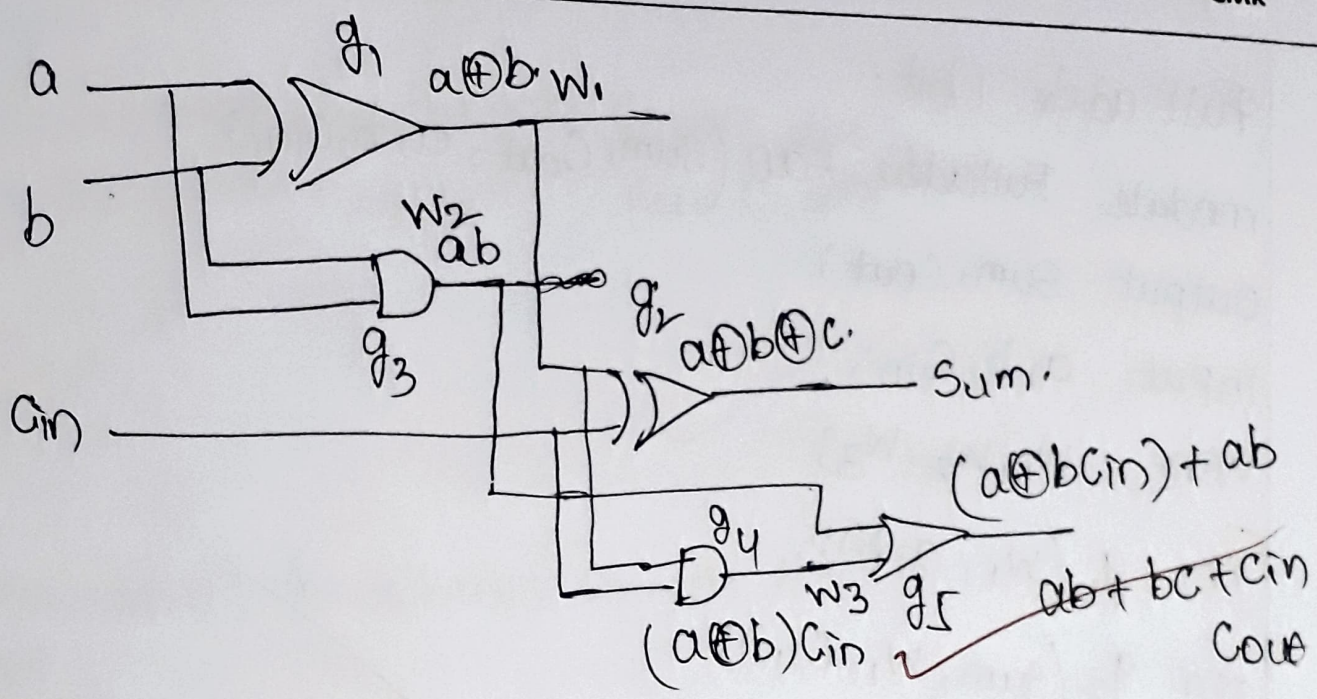
```
module 41_mux(Y, S, I);
output Y;
input [3:0] I;
input [1:0] S;
wire W0, W1, W2, W3, W4, W5;
not g1 (W0, S1);
not g2 (W1, S0);
and g3 (W2, I0, W0, W1);
and g4 (W3, I1, W0, S0);
and g5 (W4, I2, S1, W1);
and g6 (W5, I3, S1, S0);
and g7 (Y, W2, W3, W4, W5);
endmodule
```

Stimulus code:-

```
module 41_mux_tb;
reg [3:0] I;
reg [1:0] S;
wire Y;
module 41_mux (Y, S, I);
```

```
initial
begin
 I[0]=0; I[1]=1; I[2]=0; I[3]=1;
 S[0]= 0; S[1]=0;
 $display ($time, S=%b; Y=%b; I=%b; S, Y, I);
S[0]= 0; S[1]= 1;
 $display ($time, S=%b; Y=%b; I=%b; S, Y, I);
S[0]=1; S[1]=0;
 $display ($time, S=%b; Y=%b; I=%b; S, Y, I);
S[0]=1; S[1]=1;
 $display ($time, S=%b; Y=%b; I=%b; S, Y, I);
end
endmodule
```





Verilog code:-

```
module A_bit_ripple_carry_adder(S3, Cout, a, b, Cin);
output Cout;
output [3:0]S;
input  [3:0]a;
input  [3:0]b;
input Cin;
wire C0, C1, C2;
Fulladder FAD(S0, C0, a0, b0, Cin);
Full adder FA1 (S1, C1, a1, b1, C0);
Full adder FA2 (S2, C2, a2, b2, C1);
Full adder FA3 (S3, Cout, a3, b3, C2);
endmodule.
```

Full adder 1 bit.

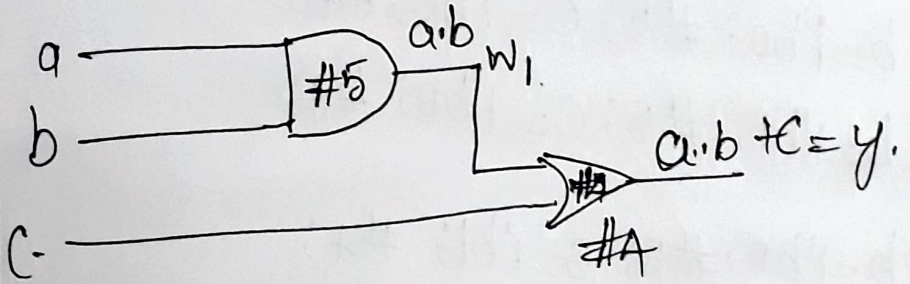
```
module Fulladder_FA0 (Sum, Cout, a, b, Cin);
output Sum, Cout;
input a, b, Cin;
wire w1, w2, w3;
xor g1 (w1, a, b);
xor g2 (Sum, w1, Cin);
and g3 (w2, a, b);
and g4 (w3, w1, Cin);
or g5 (Cout, w2, w3);
endmodule
```

Stimulus Code :-

```
module 4_bit_ripple_Carry_Adder_tb;
reg [3:0] a;
reg [3:0] b;
reg Cin;
wire [3:0] S;
wire Cout;
module 4_bit_ripple_carry_Adder (S3, Cout, a, b, Cin);
initial
```

```
begin
a=4'b0000; b=4'b0000; cin=0; #5;
a=4'b0000; b=4'b0001; cin=0; #5;
                .
                .
                .
a=4'b1111; b=4'b1111; cin=0; #5;
end
endmodule
```

(2)



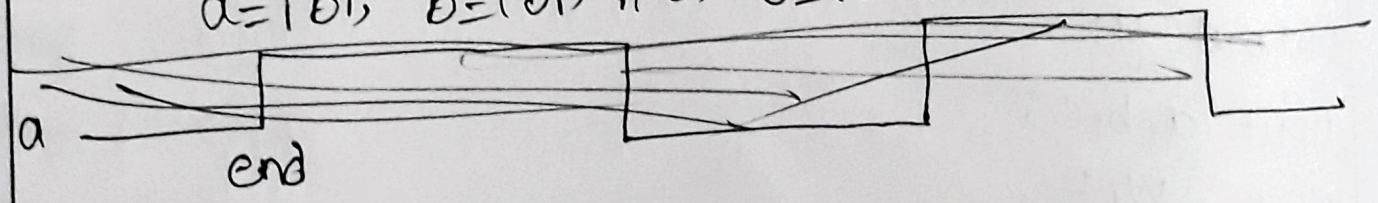
Verilog code:-

```
module a1(y, a, b, c);
output y;
input a, b, c;
wire w1;
and (#5),(w1, a, b);
or (#4),(y, w1, c);
endmodule
```

Stimulus code:-

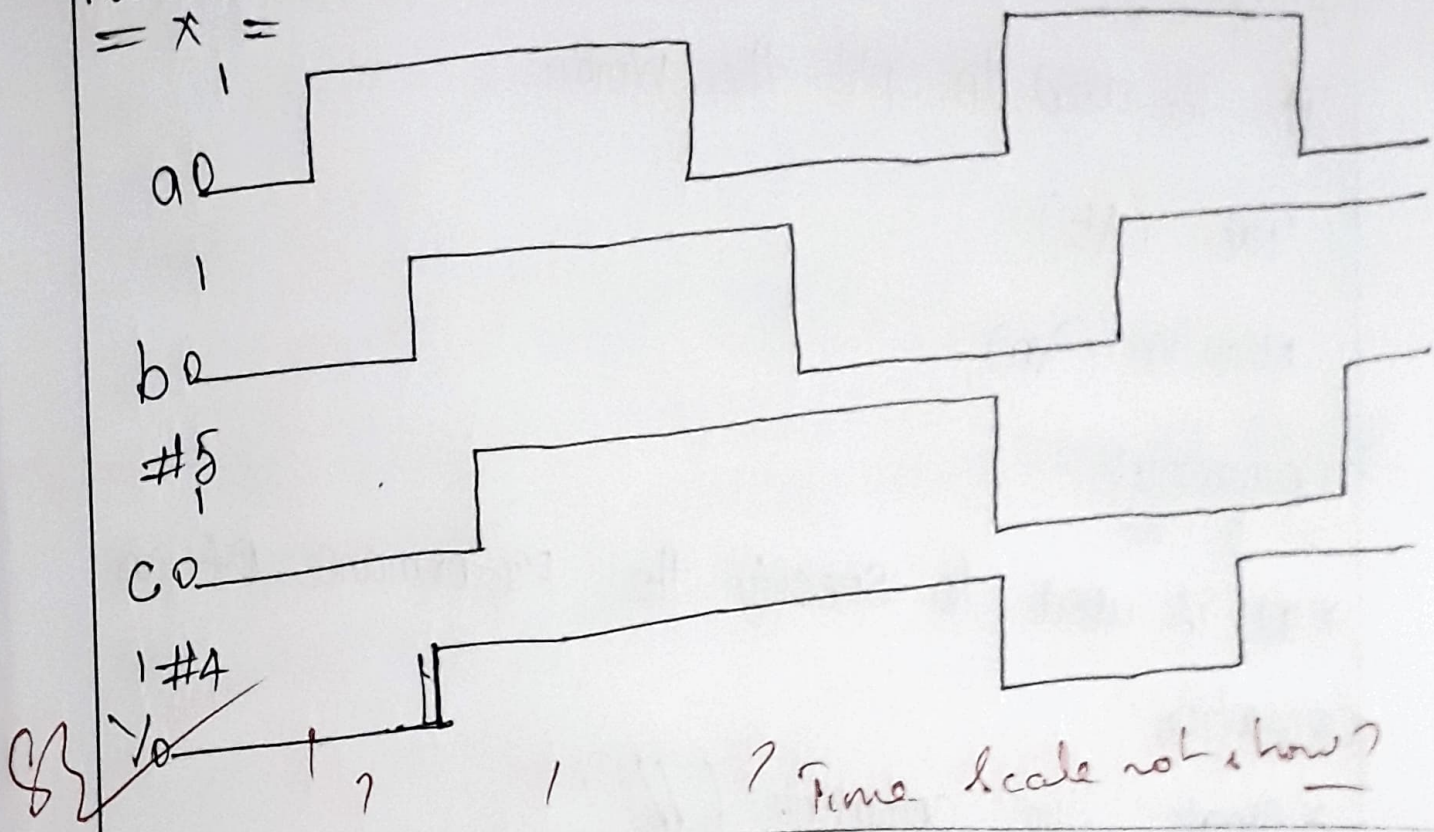
```
module a1_tb;
wire y;
reg a,b,c;
module a1 (y,a,b,c);
initial
begin
    a=1'b0; b=1'b0; #5; c=1'b0; #4;
    a=1'b0; b=1'b1; #5; c=1'b0; #4;
    a=1'b0; b=1'b0; #5; c=1'b0; #4;
    a=1'b1; b=1'b1; #5; c=1'b0; #4;

    a=1'b0; b=1'b0; #5; c=1'b1; #4;
    a=1'b0; b=1'b1; #5; c=1'b1; #4;
    a=1'b1; b=1'b0; #5; c=1'b1; #4;
    a=1'b1; b=1'b1; #5; c=1'b1; #4;
```




```
end
endmodule
```

Wave form:- $Y = a \cdot b + c$.



⑤ Lexical Conventions:-

Lexical conventions are used to design the statements, definition which are further used for programming.

The Lexical conventions are:-

→ White space

→ Comments

→ Operators

→ Numeric Specifications.

→ Strings.

→ Identifiers.

→ White Space:-

It is used to split the words.

Tab (\t)

Newline (\n)

→ Comments:-

*It is used to specify the performance of an operation.

→ single line comments (//)

→ Multiple line comments (/*_ _ _ */)

→ Operators:-

Three are three types of operators

Linear operator:- It will write after the operand

Binary operator:- The operator will be in between two operands.

Ternary operator:- The operator will be in between the multiple operands

→ Numeric specifications:

There are 4 types of specifications.

Signed numbers:

<size> 1 bit <format> <number>

Eg:- 4'b1000

Signed numbers will display size and bit format followed by the number.

Unsigned numbers:

The default size is 32-bit

The default format is decimal.

Eg:- 20

Undefined numbers:

→ The undefined number = x.

→ The high impedance number = z.

4 bits = hexa decimal

3 bits = octal

1 bit = binary.

Negative numbers:

→ The negative sign mention before the size.

Syntax:-

- <size>'<bit-format><number>

Ex: -4'd15

Strings:-

→ Where the tokens are written within double quote

Ex:- "a/b", "Hello".

Keywords and Identifiers:-

→ keywords are predefined words

→ They are written in lower case.

→ They are case sensitive.

→ Ex:- end, endmodule.

Identifiers:-

→ They are defined variables.

Ex:- reg a;