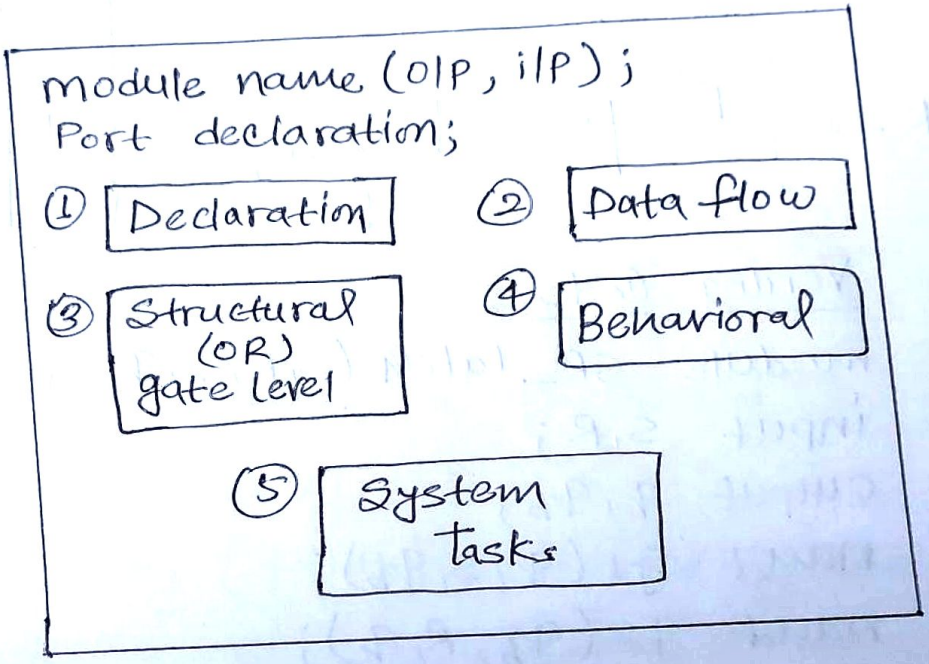
8①. What are the components of SR latch? Write Verilog HDL module of SR latch and write test bench to test the SR latch.

Soln:- Components of SR latch.



→ Declaration:

```
module module_name (input, output);
input a,b;
output sum, cout;
```
} — Port declaration.

→ Data flow: The model where the keyword 'assign' is used is the data flow.

Ex:- assign sum = i1 & i2;

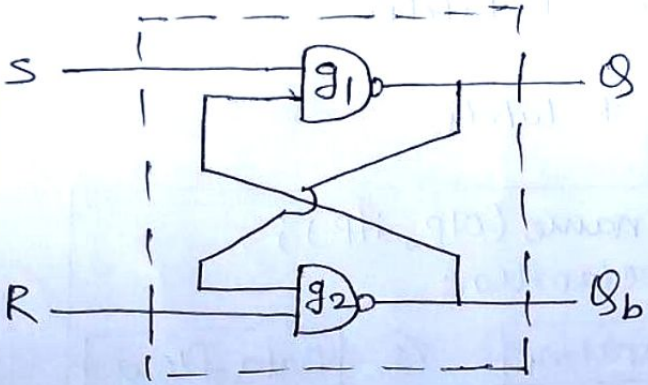
→ Structural / Gate level: Structural model where there is a calling in the main function (block) Gate level is gate name a1 (a, b, cout);

→ Behavioural: The keyword used is @ always @(s or r)

→ System task: where it performs some Routine operations. in simulation displaying, monitoring, stop and finish operation is performed.

SR - latch

Diagram



Truth table

S	R	q	qb
0	0	No change	
0	1	0	1
1	0	1	0
1	1	Toggle.	

Verilog Code :

```
module SR_latch (S, R, q, qb);
input S, R;
output q, qb;
nand g1 (q, S, qb);
nand g2 (qb, R, q);
endmodule
```

Simulation Code / test bench Code

```
module SR_latchtb;
reg S;
reg R;
```

```
wire q;
wire qb;
SR_latch m1(q, qb, ~set, ~reset);
initial
begin
$monitor ($time " set=%b, reset=%b, q=%b, qb=%b",
                set, reset, q, qb);
Set=0; reset=0; #5;
#5; reset=1;
#5; reset=0;
#5; reset=1;
end
endmodule.
```

8(2) What are the uses of $monitor, $display & $finish system tasks? Explain with examples.

Soln:→ ① $display:

The task used (or) performed is displaying the values which can be variable / string.

Syntax: $display

String by default are displayed in new line fo

example:

1. $display ("Hello Verilog World");
   -- Hello verilog World.
2. $display ($time); // display the simul
   -- 2:30 //-tion time
3. $display ("It is a new String \n which has
   %.% sign");
   -- It is a new String.
   -- which has % sign.

display specification:

%b (or) %B → display binary
%d (or) %D → display decimal
%s (or) %S → display string
%t (or) %T → display time
%o (or) %O → display octal.

→ $display is same as printf in C language.

② $monitor:

- operation used to monitor the signal when its value changes is known as monitoring.
- It has two types "$monitor on" & $monitor off"

where, $monitor on – Enable simulation
$monitor off – disables simulation.

- Syntax is $monitor
- $monitor performs same as $display.

Example:

```
$monitor ($time, "s = %b, r = %b, q = %b, qb = %b",
                 s, r, q, qb);
```

③ $finish:

- It is the operation where it terminates the simulation at a given time delay.
- Syntax: $finish

for example:

```
initial
begin
clock = 0;
reset = 1;
#100 $stop;
#900 $finish; //terminates after
                //900 time.
end.
```

③ Explain the two types of connecting ports in Verilog HDL with examples.

Soln:-

Connecting Ports :

Verilog provides two types of connecting ports which specifies the connection b/w module instantiation and module declaration.

There are two types of connecting ports

1. Connecting port by order listing.
2. Connecting port by name.

① → Connecting port by order listing

- This is first used by begineers in verilog.
- The order of the ports declared in the module instantiation. should be some as in port in the module declaration.
- for example module fulladd4 is the

```
module fulladd4 (sum, c_out, a, b, cin);
input [3:0] a,b;
input c_in;
output [3:0] sum;
output c_out;
  :
  :
<module      >
  :
  :
endmodule
module fulladd4_tb;
reg [3:0] a,b;
reg c_in;
wire [3:0] sum;
wire c_out;
```

```
fulladd4 fa0 (sum, c_out, a, b, c_in); // instantia
initial
begin
<module instantiation>
 :
 :
end
endmodule.
```

(2) Connecting Port by name :-

- In large design there will be so ports remmembering the order of the ports is not easy. which is impractical and Error.
- Verilog provides instead to connecting port in port order in portname.

for example :

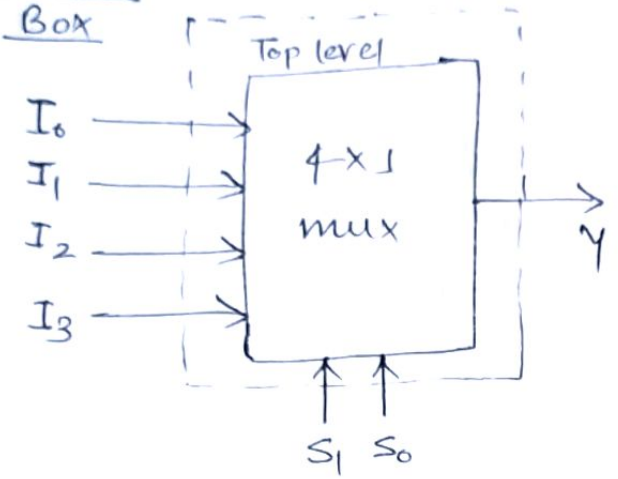
```
fulladd4 fa0 (.a(a) ; .b(b) ;
                .cin (c_in);
                .c_out (c_out);
                .sum (sum);
              );
```

- make sure that all the external connected ports are connected with all port name.
- If there is any unconnected port in the module definition it can be disconnected. for example.

```
fulladd4 fa0(
                .a(a);
                .b(b);
                .cin (cin);
                .sum (sum); // cout is unconnected.
              );
```

3④ Use gate level description of verilog HDL to design 4 to 1 multiplexer. Write truth table, top-level block, logic expression and logic diagram. Also write the Stimulus block for the same.

Soln:→ Black Box

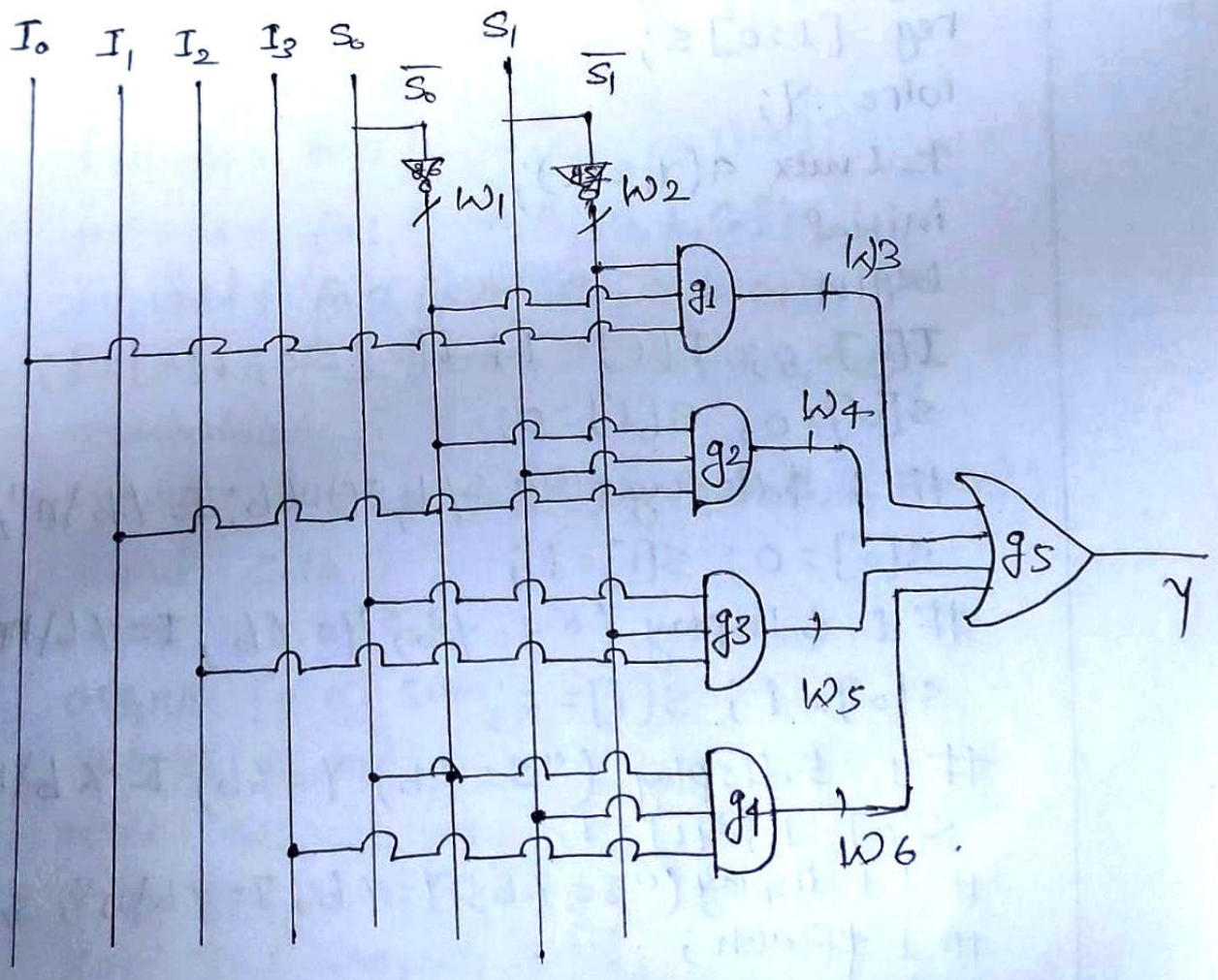


Truth Table

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

Logic Expression: $Y = \bar{S_0}\bar{S_1}I_0 + \bar{S_0}S_1I_1 + S_0\bar{S_1}I_2 + S_0S_1I_3$

Logic diagram:



Code:

```
module 4_1Mux (Y, I, S);
output Y;
input [3:0] I;
input [1:0] S;
wire w1, w2, w3, w4, w5, w6;
and g1(w3, w2, w1, I[0]);
and g2(w4, w1, S1, I[1]);
and g3(w5, S0, w2, I[2]);
and g4(w6, S1, S0, I[3]);
OR g5(Y, w3, w4, w5, w6);
not g6(w1, S0);
not g7(w2, S1);
endmodule.
```

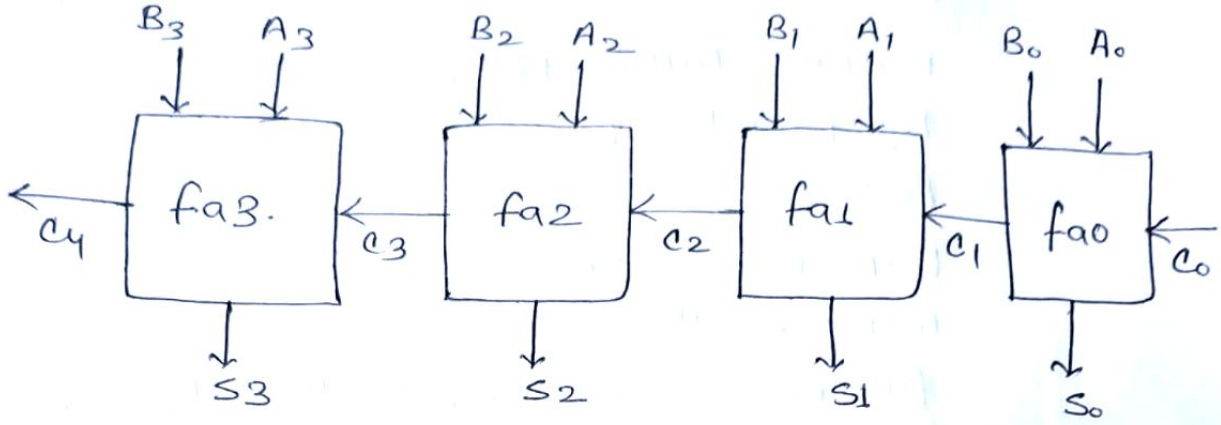
Stimulus Code:

```
module 4_1 muxtb;
reg [3:0] I;
reg [1:0] S;
wire Y;
4_1 mux a(Y, S, I);
initial
begin
I[0]=0; I[1]=1; I[2]=0; I[3]=1;
S[0]=0; S[1]=0;
#1 $display("S=%b, Y=%b; I=%b\n", S, I, Y);
S[0]=0; S[1]=1;
#1 $display("S=%b; Y=%b; I=%b\n", S, Y, I);
S[0]=1; S[1]=0;
#1 $display("S=%b; Y=%b; I=%b\n", S, Y, I);
S[0]=1; S[1]=1;
#1 $display("S=%b; Y=%b; I=%b\n", S, Y, I);
#1 $finish;
end
endmodule
```

⑤ Use gate level description of verilog HDL to describe the 4-bit ripple carry adder. Also write a stimulus block for 4-bit ripple carry adder.

Soln:→

4-bit ripple carry adder :



Verilog code : (Gate level);

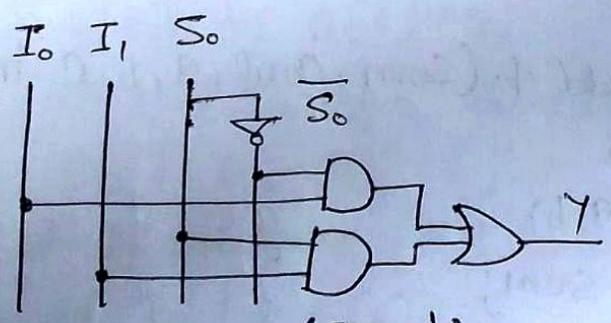
```
module 4_bit_ripple_carry_adder(sum, C_out, a, b, C_in);
input [3:0] a, b;
input C_in;
output [3:0] sum;
output C_out;
fulladd4 fa0 (sum, C0, a, b, C_in);
fulladd4 fa1 (sum, C1, a, b, C0);
fulladd4 fa2 (sum, C2, a, b, C1);
fulladd4 fa3 (sum, C_out, a, b, C2);
endmodule.
module fulladd4 (sum, Cout, a, b, C_in);
input C_in;
input [3:0] a, b;
output [3:0] sum;
output C_out;
wire w1, w2, w3;
xor g1 (w1, a, b);
xor g2 (sum, w1, C_in);
```

```
and g3 (w3, a, b);
and g4 (w2, w1, cin);
or g5 (c_out, w2, w3);
endmodule
```

Stimulus Code:

```
module fulladd4bt;
reg [3:0] a, b;
reg cin;
wire [3:0] sum;
wire c_out;
fulladd4 fab(sum, c_out, a, b, c_in);
initial
begin
$monitor($time,"a=%b b=%b, c_in=%b, sum=%b
          c_out=%b\n", a, b, c_in, sum, cout);
#5; a=4'b0000; b=4'b0; c=1'b0;
#5; a=4'b0001; b=4'b0000; c=1'b1;
    :
#5; $finish
end
endmodule
```

8⑥ Given: 2x1 mux using bufifo & bufif1.

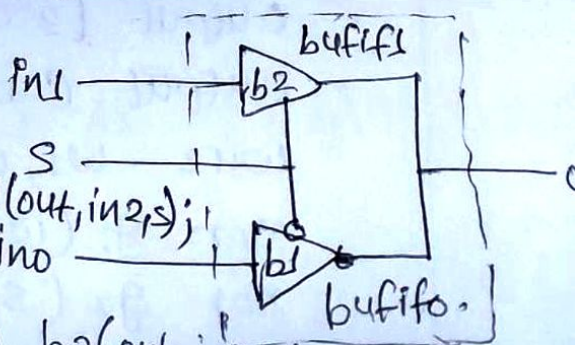


T.T

S_0	Y
0	I_0
1	I_1

Exp: $Y = \bar{S_0} I_0 + S_0 I_1$

```
module 2-1mux (I, Y, S);
input [1:0] I; I/P=S; O/P=Y;
bufifo #(1:2:3, 3:4:5, 5:6:7); b1(out, in2, s);
          rise  fall  turnoff   in0
bufifo #(1:2:3, 3:4:5, 5:6:7) b2(out, in1, s); endmodule
```



⑦ With Syntax describe the Continuous assignment statement. Show how different delays associated with logic circuit are modelled using dataflow description.

Soln:-

Continuous assignment statement

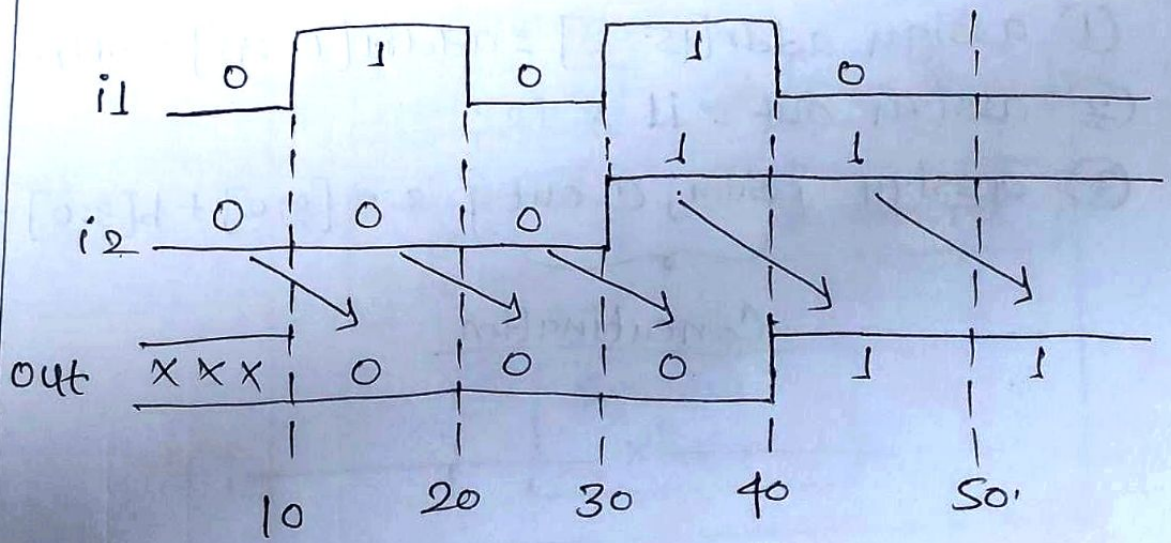
Continous_assignment ::= assign #10.
Continuous_net_of_assignment
net_off_assign.

Delay associated with Continuous assignment are:

① Regular Continuous assignment
② Implicit Continuous assignment
③ Net declaration.

① Regular Continuous assignment:

- The first method used to specify particular delay in Continuous assignment.
- The delay is specified after the assign statement.
- When the value of i1 & i2 varies there is 1 time delay for every before Commutation of i1 & i2.
- Initial delay is formed.



for example : 1. assign #10 out = in1 & i2;
2. assign #20 sum = in1 ^ i2;

② Implicit Continuous assignment

— It is same as Continuous assignment.

Ex: wire out;
assign out = i1 & i2;

③ Net Declaration

— Which can be used in gate level modeling.
— When there is declaration or 'net'.

Characteristic of Continuous assignment

— The LHS of Continuous assignment scalar (or) vector 'NET' / concatination of both are scalar/vector "NETS" it can't be scalar / vector register.
— RHS of the Continuous assignment operators is can be found only when the values of RHS operand changes,
— RHS operands can be NETs / Register / etc.

Ex:

① assign addr[15:0] = addr1[15:0] + addr2[15:0];
② assign out = i1 & i2;
③ assign {sum, C_out} = a[3:0] + b[3:0] + C_in;

Concatination

— x —