PAGE NO. : 01
DATE : 31/01/2024
EXP. NO. :

PROGRAM 1

Steps to build the circuit :

1) Open multisim : Launch N multisim on your windows system.
   • Start a new design by clicking on File > New

2) Place components : • From toolbar, select Place component.

components needed are
XOR Gate (XOR2) : Go to logic > Gate and select an XOR gate (typically 2-input XOR)
→ NOT Gate : Go to logic > Gate and select NOT gate
→ AND Gate : Go to logic > Gate and select 2 input AND gate.
→ Keys (Switches) : Go to source > Switch and select a key.
→ Ground : Place ground by selecting Place > Ground from the menu or toolbar.
→ Indicator : Select X1 and X2 as Probes.

3) Place the components on workspace.

PAGE NO. : 02
DATE :
EXP. NO. :

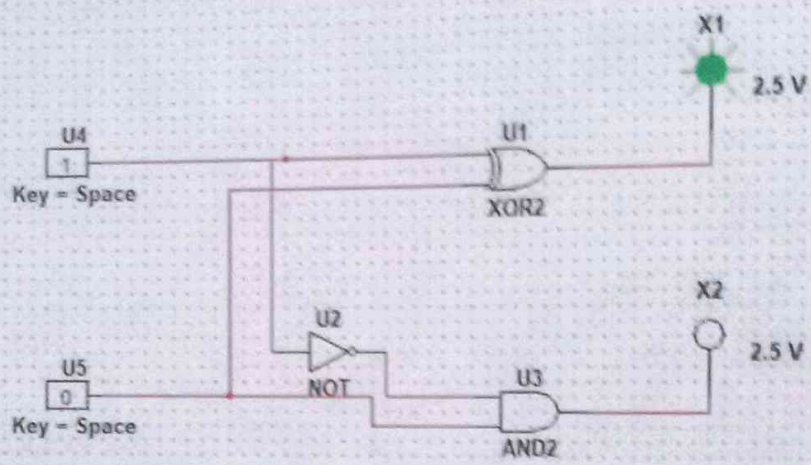
Drag and drop components

4) Wiring components
use wiring tool to connect components
→ Key-Space (U1) connected to one input of XOR gate (U3) and one input of AND gate (U6)
→ Key-space (U2) connected to second input of XOR gate (U3) and to input of NOT gate (U5)
→ output of NOT gate (U5) is connected to second input of AND gate (U6)
→ output of XOR gate (U3) goes to Probe X1
→ output of AND gate (U6) goes to Probe X2

5) Adjust voltage levels
Set up voltage levels of keys if needed to ensure power is available for gate function.

6) Simulate circuit:
go to simulate > Run

7) Note the outputs
Observe output voltage based on input from switches.


X1
2.5 V
U4
1
Key = Space
U1
XOR2
X2
U2
2.5 V
U5
0
Key = Space
NOT
U3
AND2

PAGE NO. : 03
DATE :
EXP. NO. :

PROGRAM - 2

Steps to build and execute the program:

1) Open multisim: Launch multisim software.

2) Create new project: Go to file > New > Design.

3) Add NAND Gate: Go to place components
- browse NAND gate from TTL family
- Select 74LS00NAND

4) Place input switches (A, B)
- Search for SPST switches or similar logic input devices
- Place 2 switches.

5) Connect circuit
U3 and U1 : connect input A and B to NAND gates to generate the signals.
U4 and U2: connect these NAND gates based on the XOR implementation.
U5 : use NAND gates for carry output.

PAGE NO. : 04
DATE :
EXP. NO. :

6) Place output indicator

* Place component 2 indicators one for sum and carry.

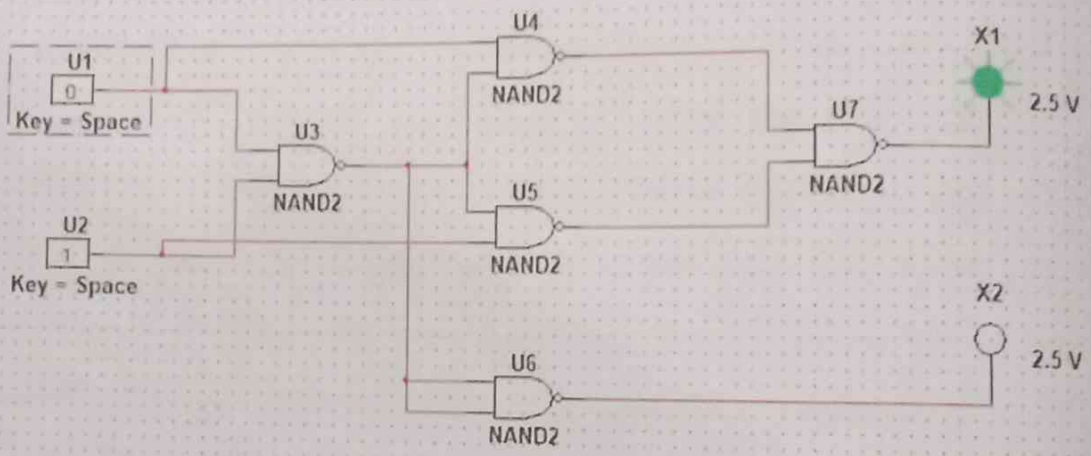
7) Power the circuits:

* Add DC Power source for the logic gates.

8) Run simulator:

* Click on Run and toggle input switches to verify sum and carry

25/12/24


U1
0
Key = Space
U2
1
Key = Space
U3
NAND2
U4
NAND2
U5
NAND2
U6
NAND2
U7
NAND2
X1
2.5 V
X2
2.5 V

PAGE NO. : 05
DATE :
EXP. NO. :

PROGRAM 3

Steps to build and execute circuit :

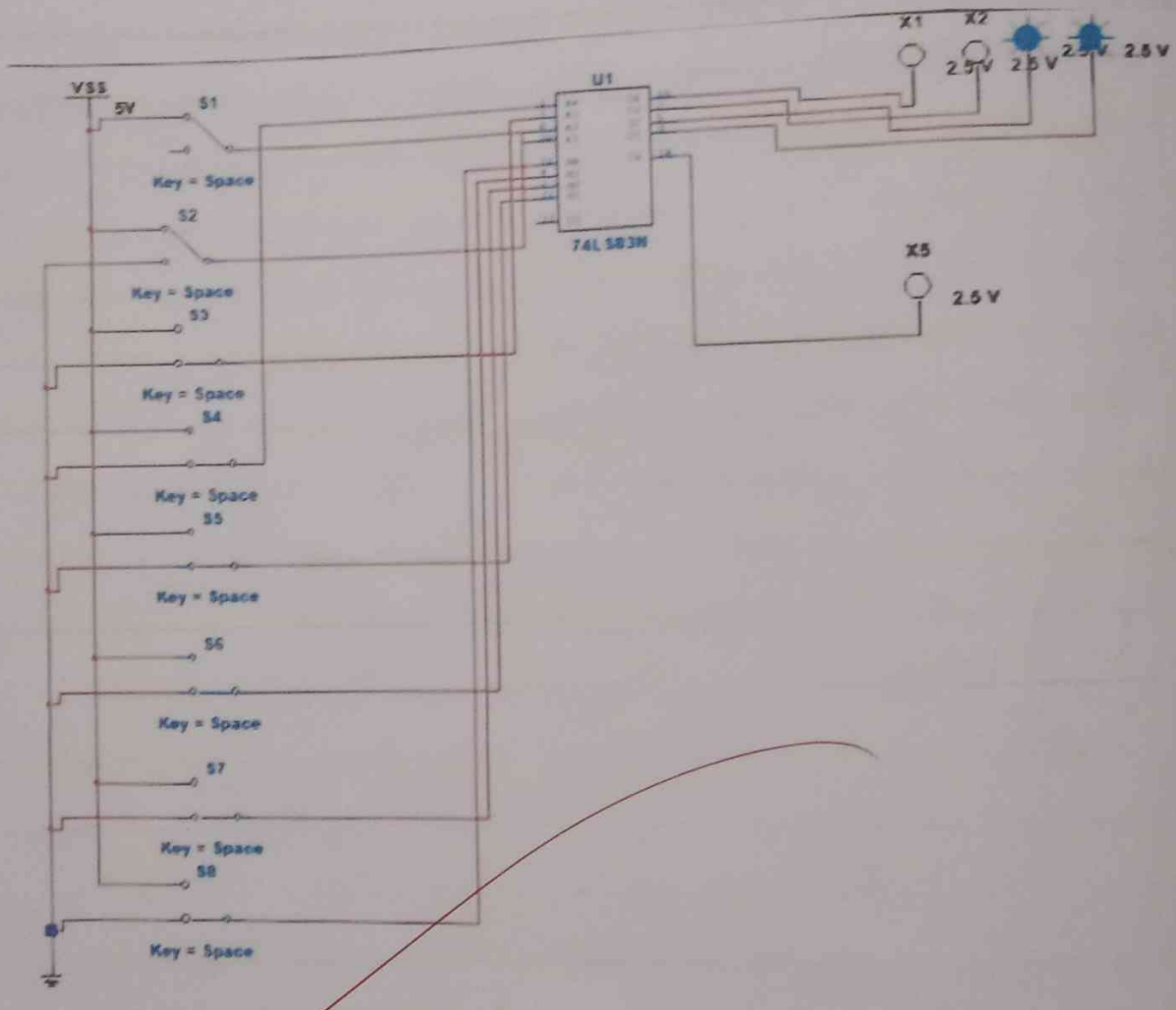
1) Open Multisim : Launch the software

2) Create New Project : File > New > Design.

3) Place components: Add a 74LS83N IC from component library.
* Add SPST switch for each input (S1 to S8) from switches
* Add indicators to display results.
* Add Power source from sources library.
* Place ground symbol.

4) Wire the components
* connect switches to input of 74LS83N IC
* connect output of IC to LED'S (X1, X2)
* connect Power supply

5) Simulate and Report
* Run the circuit and toggle switches.
* Now report the observation.


VSS
5V
S1
Key = Space
S2
Key = Space
S3
Key = Space
S4
Key = Space
S5
Key = Space
S6
Key = Space
S7
Key = Space
S8
Key = Space
U1
74LS83N
X1
X2
2.5 V
2.5 V
2.5 V
2.5 V
X5
2.5 V

PAGE NO. : 06
DATE :
EXP. NO. :

PROGRAM 4

Steps to execute and build circuit

1) Open multisim : Launch Software

2) Create New project : File > New > Design

3) Add components :

→ Based on image, Go to place components and choose logic gates from TTL or CMOS libraries

→ Place the logic gates according to the circuit layout use AND, OR and NOT gate as appro-priate, wiring them as they appear in image

4) Place switches.

* Add SPST switches for input signals.

5) Add Probe

* Go to indicator family and choose a probe for output of each sections of circuit

* Connect these to output.

PAGE NO. : 07
DATE :
EXP. NO. :

6) Power supply:

Add DC voltage source
connect +Vc terminal of it to Vcc and negative to ground.

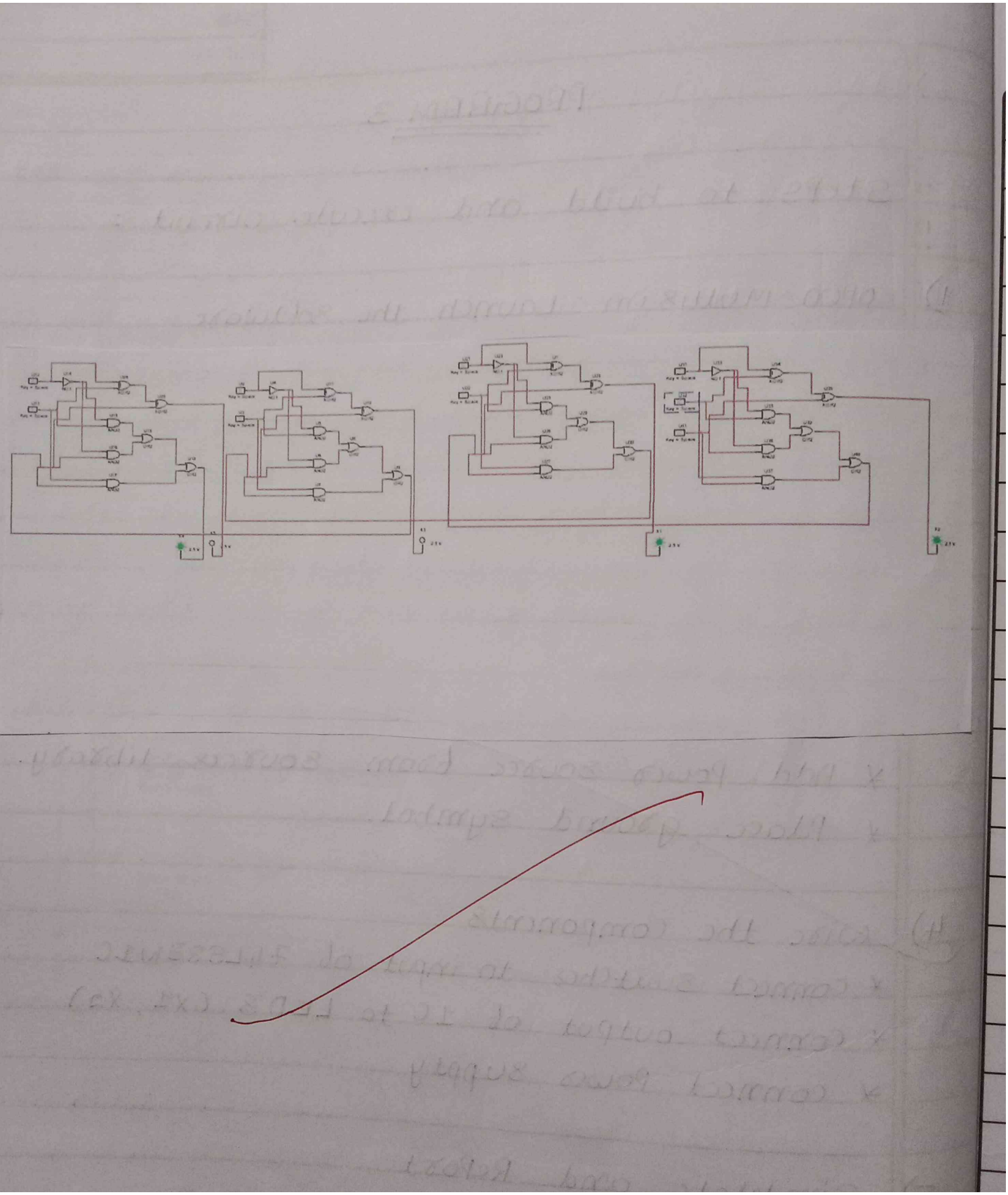
7) wire circuit

wire the circuits according to the circuit diagram.

8) Simulate and Report

Click on Run and toggle switches to observe and reports.

05/12/24



PAGE NO. : 08
DATE : 8/11/2024
EXP. NO. :

08) Design Verilog to implement binary address and subtractor Half and Full adder, Half and Full subtractor

Half Adder

Logic expression

Sum - $A \oplus B = \bar{A}B + A\bar{B}$

Carry = AB

CODE -

```
module ha(a,b,S,C);
input a,b;
output S,C;
assign S=a^b;
assign C=a&&b;
end module
```

PROCEDURE

As half adder contains only two lists so along with addition of two single lists as inputs, 2 outputs are obtained as Sum and Carry.

PAGE NO. : 09
DATE :
EXP. NO. :

STEP 01 : Write verilog code using appropriate modeling style to describe the discred logic. Ensure the code has input and output ports, logic operations and any nessary components.

STEP 02 : Define the inputs and apply stimulus and generate the simulation waveform.

STEP 03 : Review the console output for result and debugging information using the timing diagram.

FULL ADDER

Logic expression

sum = $\bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$

carry = $AB + AC + BC$

```
module fa(a,b,cin,S,c);
input a,b,cin;
output c,S;
assign S = a^b^cin;
carry  c = a&&b|| a&&cin& b&&cin;
endmodule
```

PAGE NO. : 10
DATE :
EXP. NO. :

PROCEDURE

The full adder adds 3 list input and Produces a single list output. Thus it is useful when an extra carry list is available from the previously generated result.

STEP 01

Write the verilog code for full adder and obtain the schematic.

STEP 02 :

Write the test bench code for full adder and obtain the simulation waveform.

STEP 03 :

Get result from output window

05/12/24

PAGE NO. : 11
DATE :
EXP. NO. :

Half Subtractor

Code

```
module hs(x,y,b,d)
input x,y;
output d,b;
assign b = (~x)&&y;
assign d = x^y;
end module
```

PAGE NO. : 12
DATE :
EXP. NO. :

FULL SUBTRACTOR

CODE

```
module fs(x,y,3,b,d);
input x,y,3;
output b,d;
assign b = (~x) && y || (~x) && (3) || y && 3
assign d = x^y^3;
end module
```

05/12/24

Truth Table

INPUt		output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

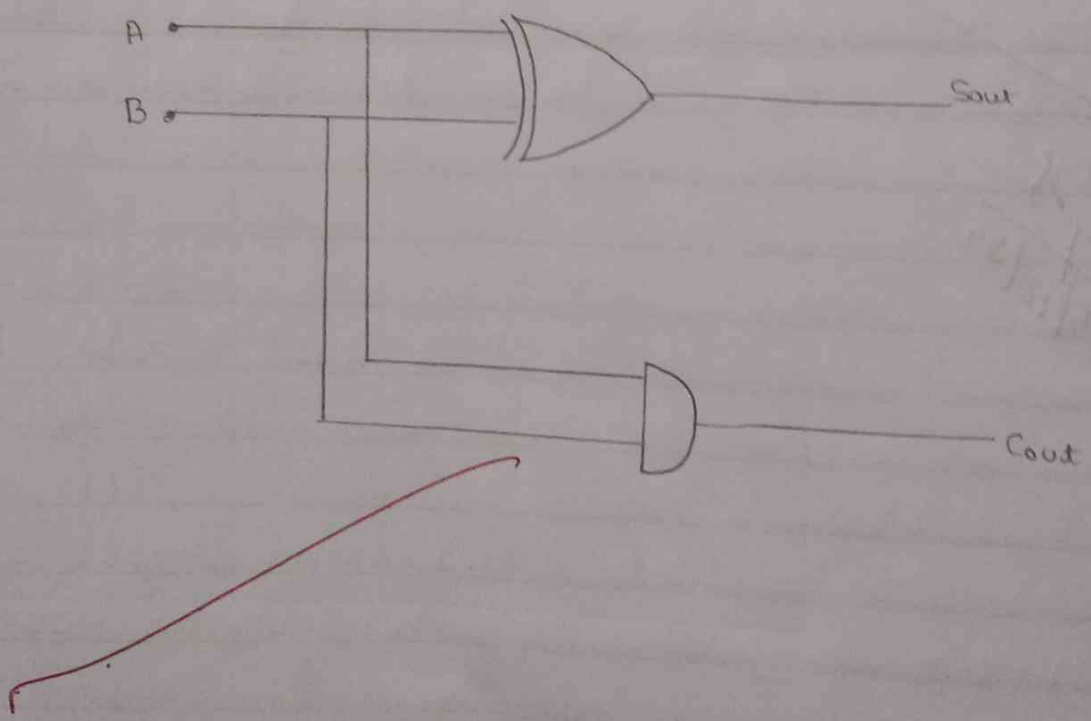
K-map

Sum

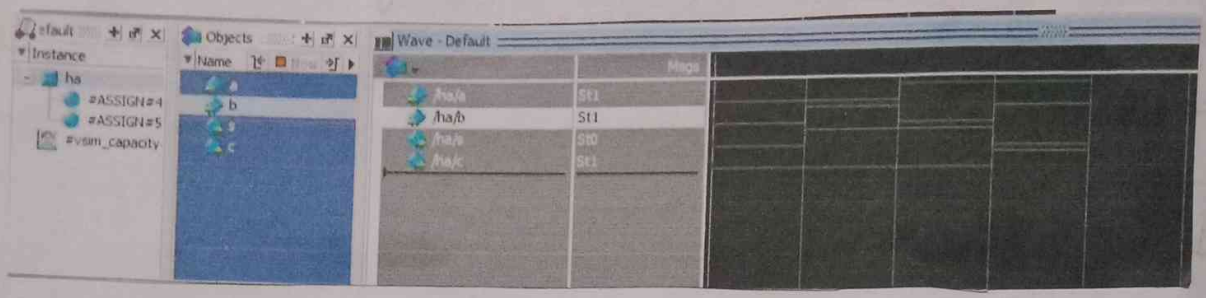
A \ B	0	1
0	0	1
1	1	0

A \ B	0	1
0	0	0
1	0	1

Circuit Diagram



OUTPUT

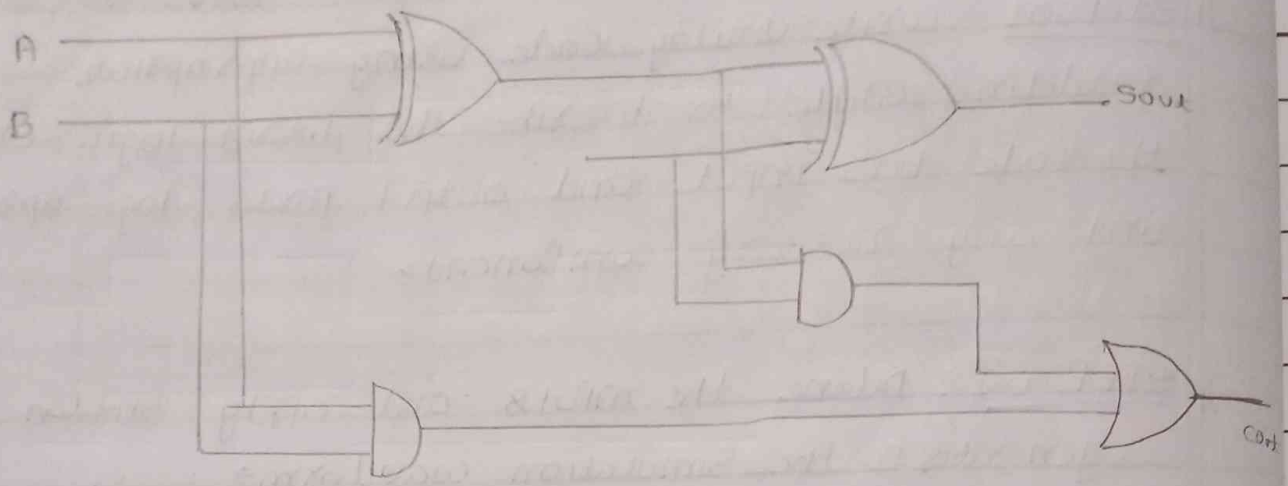


Full adder

TRUTH TABLE

Input			Output	
A	B	Cin	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Circuit diagram



Kmap

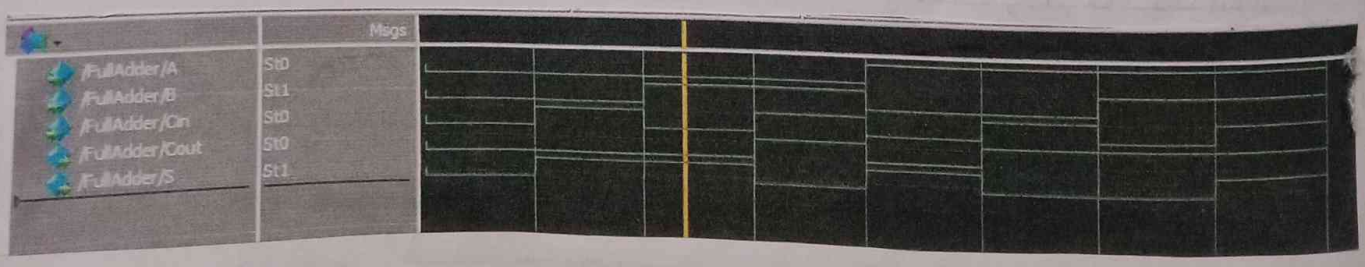
SUM

	00	01	11	10
0		1		1
1	1		1	

CARRY

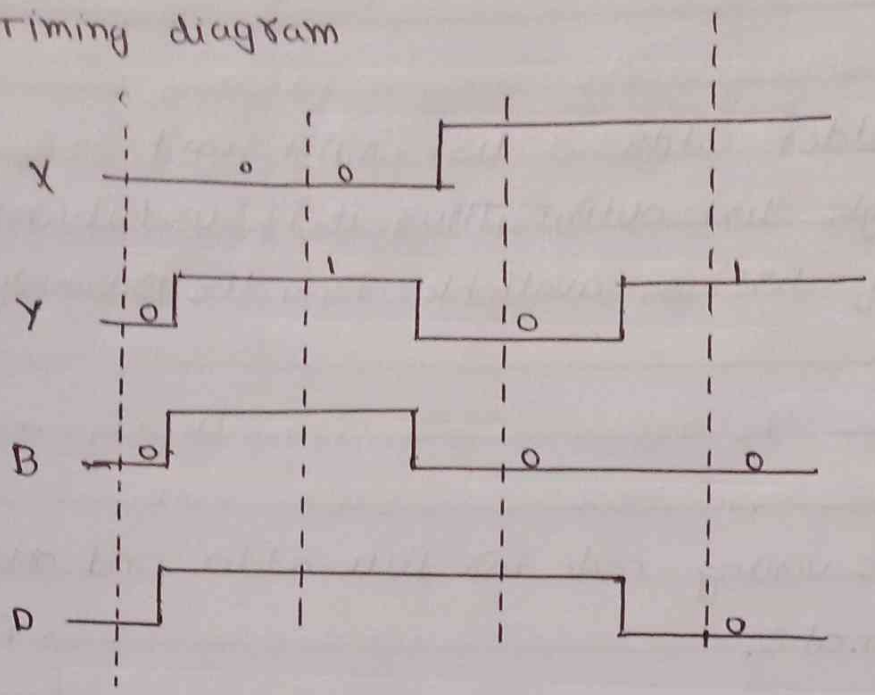
A \ BC	00	01	11	10
0			1	
1		1	1	1

OUTPUT



Half Subtraction

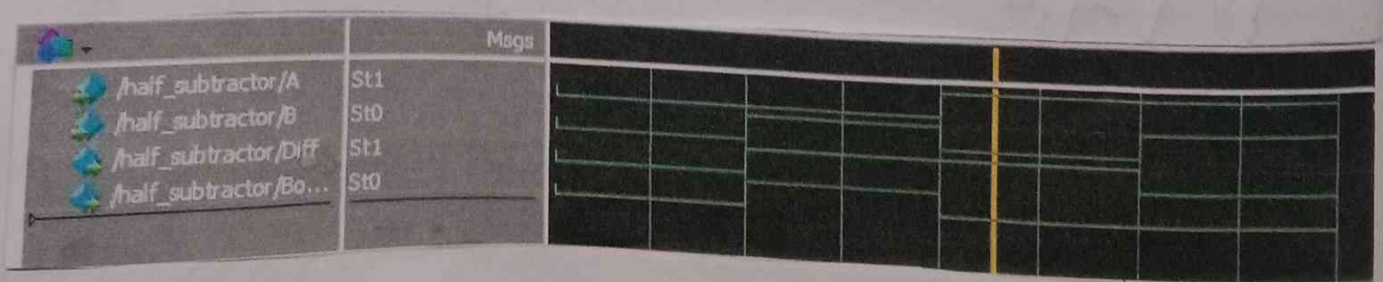
Timing diagram



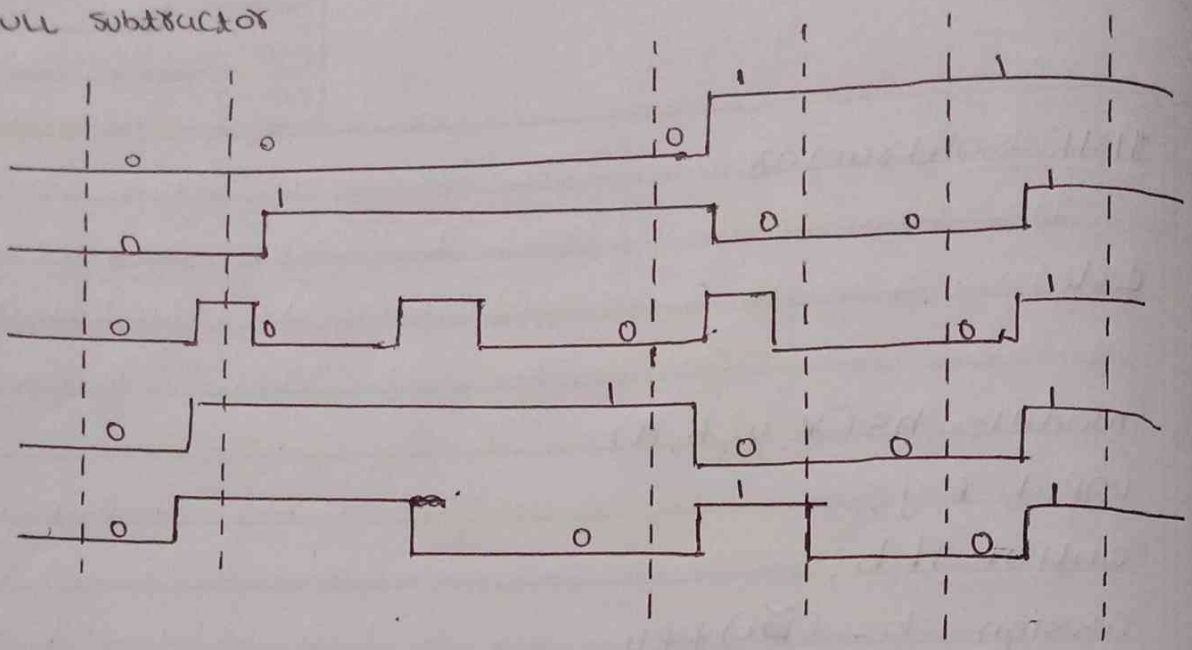
Truth Table

X	Y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

OUTPUT



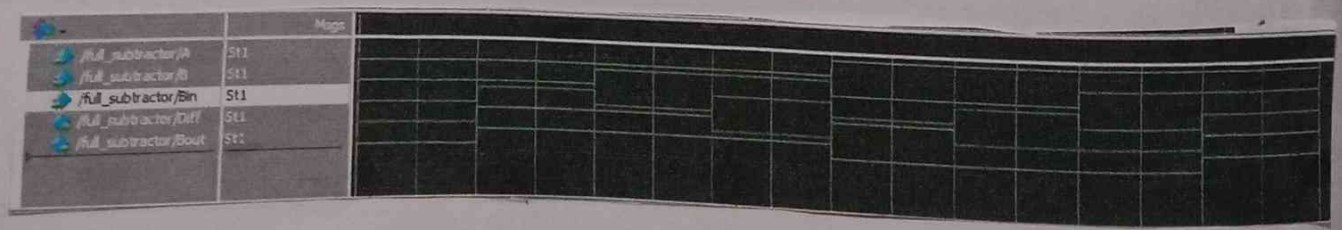
FULL SUBTRACTOR



TRUTH TABLE

x	y	z	b	d
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

OUTPUT



PAGE NO. : 13
DATE : 08/11/2024
EXP. NO. :

06) Full Subtractor

```
module full_subtractor (
input A;
input B;
input Borrowin;
output Difference;
output Borrow out;
);
assign Difference = A^B^Borrowin
assign Borrowout = (~A && (B^Borrowin)) | (B& Borrow in)
end module.
```

Half Subtractor

```
module half subtractor (
input A;
input B;
output Diff;
output Borrow;
);
assign Diff = A^B;
assign Borrow = ~A&B;
end module.
```

PAGE NO. : 13
DATE : 08/11/2024
EXP. NO. :

06) Full Subtractor

```
module full_subtractor (
input A;
input B;
input Borrowin;
output Difference;
output Borrow out;
);
assign Difference = A^B^Borrowin
assign Borrowout = (~A && (B^Borrowin)) | (B& Borrow in)
end module
```

Half Subtractor

```
module half_subtractor (
input A;
input B;
output Diff;
output Borrow;
);
assign Diff = A^B;
assign Borrow = ~A&B;
end module.
```

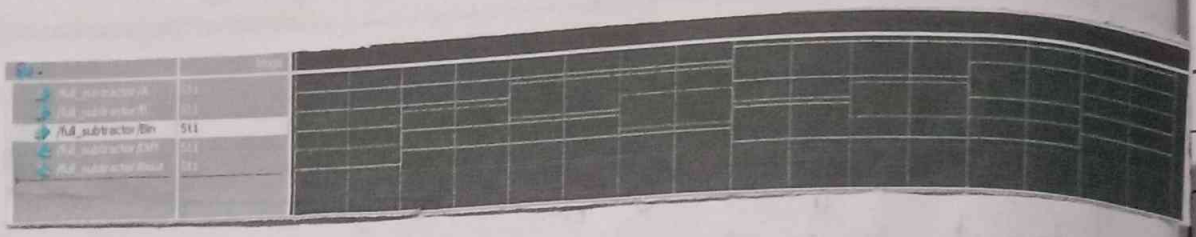
PAGE NO. : 13
DATE : 08/11/2024
EXP. NO. :

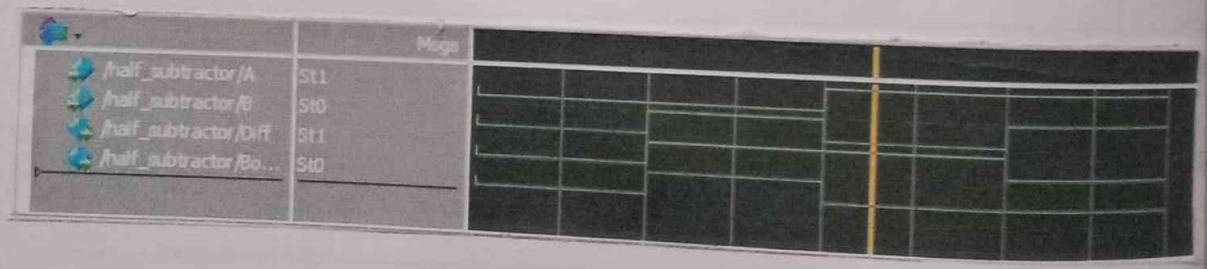
06) Full Subtractor

```
module full_subtractor (
input A;
input B;
input Borrowin;
output Difference;
output Borrow out;
);
assign Difference = A^B^Borrowin
assign Borrowout = (~A && (B ^ Borrowin)) | (B& Borrow in)
end module.
```

Half Subtractor

```
module half_subtractor (
input A;
input B;
output Diff;
output Borrow;
);
assign Diff = A^B;
assign Borrow = ~A&B;
end module.
```


/full_subtractor/Bin
St1


Msgs
/half_subtractor/A
St1
/half_subtractor/B
St0
/half_subtractor/Diff
St1
/half_subtractor/Bo...
St0

PAGE NO.: 14
DATE: 22/11/2024
EXP. NO.:

07) Design verilog program to implement different types of multiplexers 2:1, 4:1 and 8:1

Multiplexers 2:1

```
module mux 2_1 (m1, m2, select, out);
input m1, m2, select;
output out;
assign out = select ? m2 : m1;
end module : mux 2-1
```

PAGE NO. : 15
DATE :
EXP. NO. :

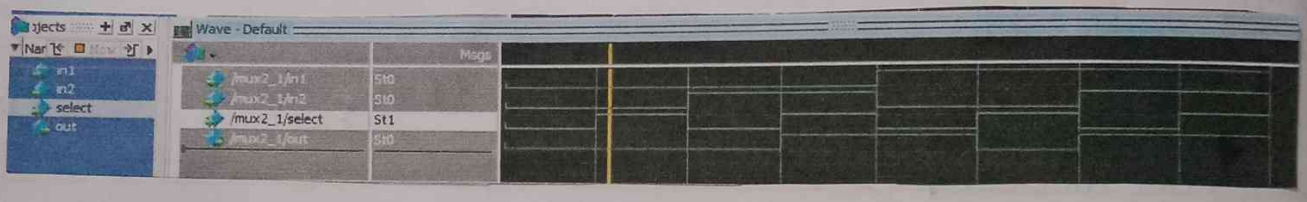
Multiplexer 4:1

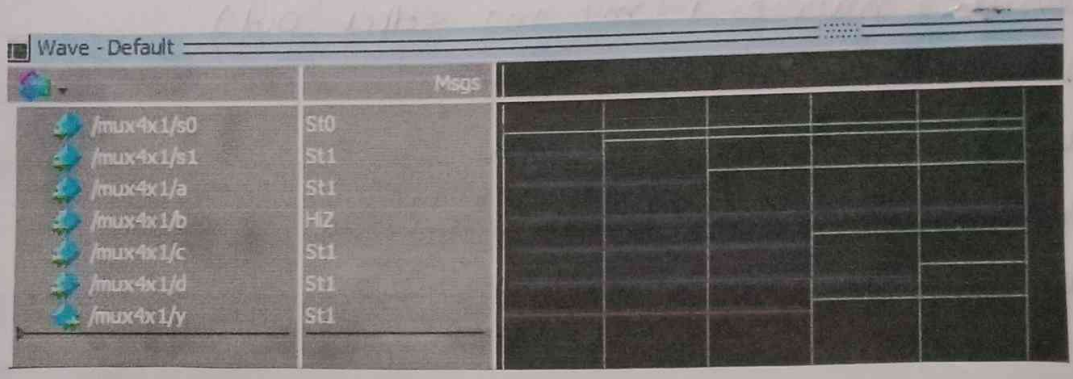
```
module mux4x1(
input s0;
input s1;
input a;
input b;
input c;
input d;
output y;
);
assign y = (a & (~s1) & (~s0)) | (b & (~s1) & (s0)) | (c & s1 &
           (~s0)) | (d & s1 & s0);
end module
```

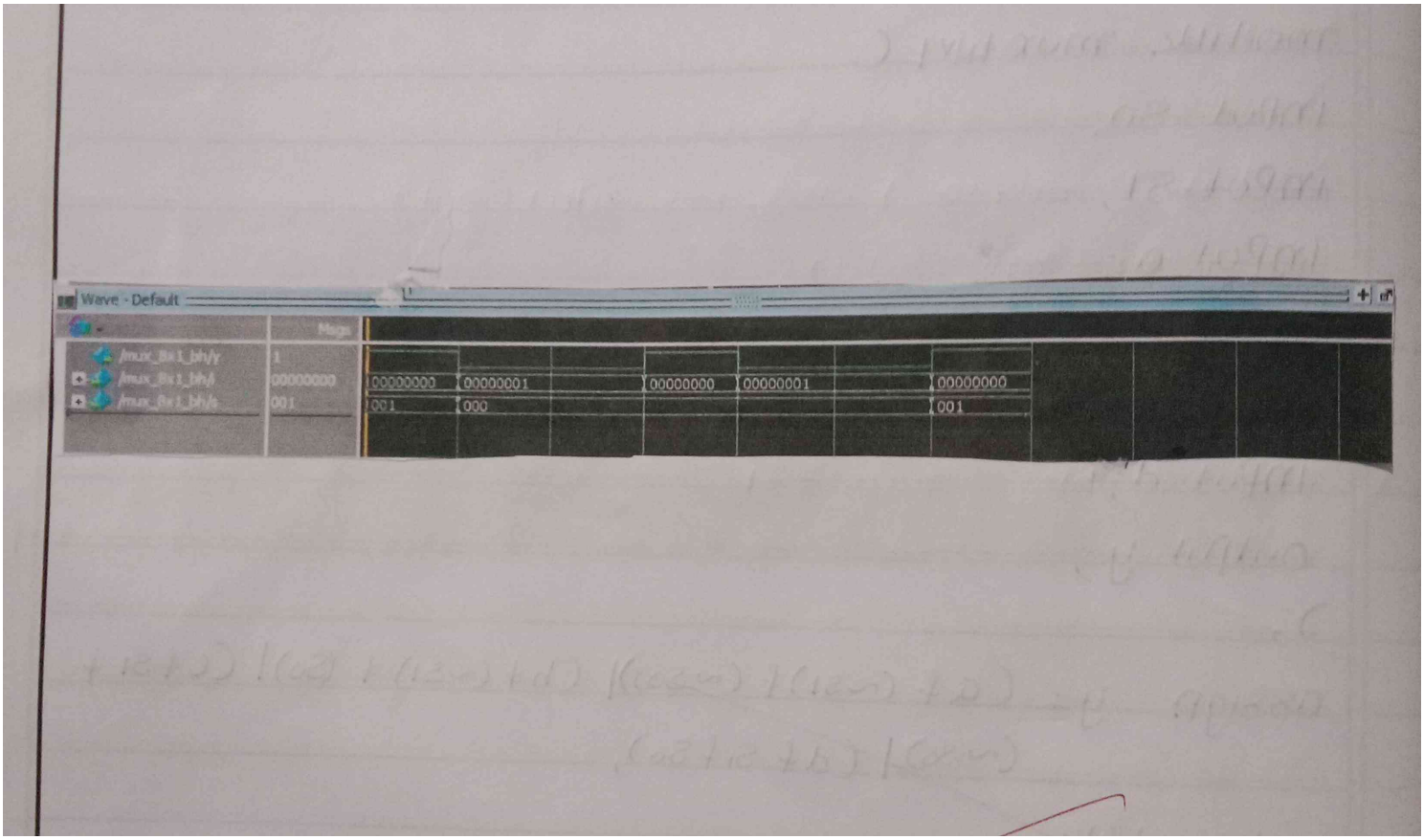
PAGE NO. : 16
DATE :
EXP. NO. :

multiplexer 8:1

```
module mux_8x1_bh(
output reg y;
input [7:0]i;
input [2:0]s;
);
always @(s)
begin
case (s)
3'b000: y = i[0];
3'b001: y = i[1];
3'b010: y = i[2];
3'b011: y = i[3];
3'b100: y = i[4];
3'b101: y = i[5];
3'b110: y = i[6];
3'b111: y = i[7];
end case
end
end module
```


Wave - Default
in1
in2
select
out
/mux2_1/in1
/mux2_1/in2
/mux2_1/select
/mux2_1/out
St0
St0
St1
St0


Wave - Default
Msgs
/mux4x1/s0
/mux4x1/s1
/mux4x1/a
/mux4x1/b
/mux4x1/c
/mux4x1/d
/mux4x1/y
St0
St1
St1
HiZ
St1
St1
St1


Wave - Default
Msgs
/mux_8x1_bh/y
/mux_8x1_bh/i
/mux_8x1_bh/s
1
00000000
001
00000000
00000001
00000000
00000001
00000000
001
000
001

PAGE NO. : 17
DATE : 28/11/2024
EXP. NO. :

08) Design verilog Program to implement different type of de-multiplexers

1:2 De multiplexers

```
module demux 12 (S, y0, y1, I);
input S, I;
output y0, y1;
assign y0 = ~S & I;
assign y1 = S & I;
end module
```

y0 = ~S I

y = S I

PAGE NO. : 13
DATE :
EXP. NO. :

1:4 DC multiplexer

```
module demux 1 4 (S0, S1, V0, V1, V2, V3, I);
input S0, S1, I;
output Y0, Y1, Y2, Y3;
assign Y0 = ~S0 & ~S1 & I;
assign Y1 = ~S0 & S1 & I;
assign Y2 = S0 & S1 & I;
assign Y3 = S0 & S1 & I;
end module
```

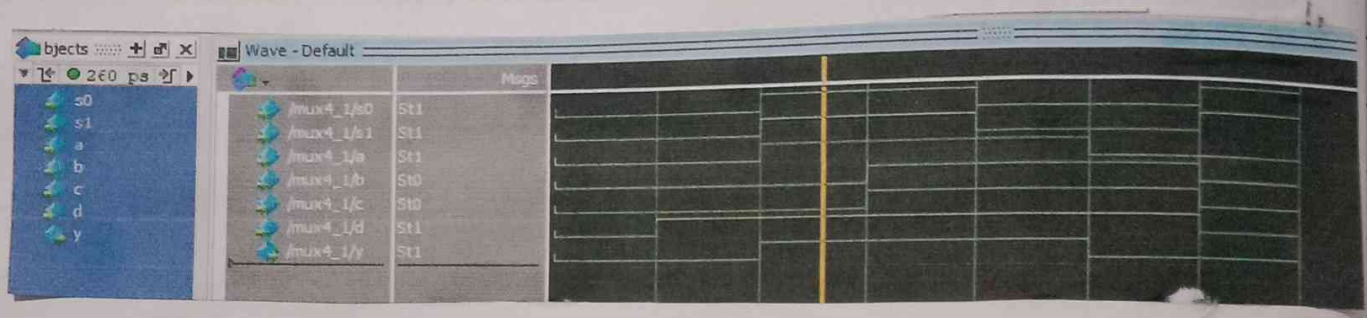
PAGE NO. : 19
DATE : 28/11/2024
EXP. NO. :

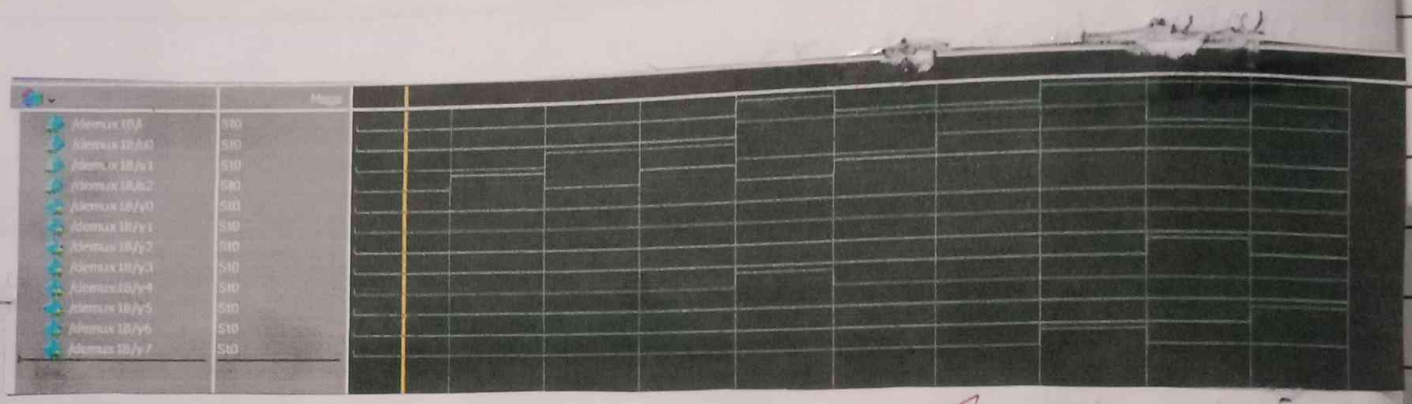
1 : 8 Demultiplexer

```
module dmux 18 (I, y0, y1, y2, y3, y4, y5, y6, y7, S0, S1, S2);
input I, S0, S1, S2;
output y0, y1, y2, y3, y4, y5, y6, y7;
assign y0 = ~S0 & ~S1 & ~S1 & I;
assign y1 = ~S0 & ~S1 & S - & I;
assign y2 = ~S0 & ~S & & ~S = & I;
assign y3 = ~S0 & ~S1 & & S2 & I;
assign y4 = S0 & ~S1 & & S2 & & I;
assign y6 = S0 & S1 & S2 & I;
assign y7 = S0 & S1 & S2 & I;
end module;
```

05/12/24


Wave - Default
Msgs
/demux12/s
St0
/demux12/i
St0
/demux12/Y0
St0
/demux12/Y1
St0


bjects
260 ps
s0
s1
a
b
c
d
y
Wave - Default
Msgs
/mux4_1/s0
/mux4_1/s1
/mux4_1/a
/mux4_1/b
/mux4_1/c
/mux4_1/d
/mux4_1/y
St1
St1
St1
St0
St0
St1
St1



PAGE NO. : 20
DATE : 08/11/2024
EXP. NO. :

09) Design verilog program for implementing various types of Fliptlops such as SR, JK, D

D Flip Flop

```
module RisingEdge_DFlipFlop (D, CLK, Q)
input D;
input CLK;
output Q;
always @ (Posedge CLK)
begin
Q <= D;
end
endmodule
```

PAGE NO. : 20
DATE : 28/11/2024
EXP. NO. :

09) Design verilog program for implementing various types of Flipflops such as SR, JK, D

D Flip Flop

```
module RisingEdge_DFlipFlop (D, CLK, Q)
input D;
input CLK;
output Q;
always @ (Posedge CLK)
begin
Q <= D;
end
endmodule
```

PAGE NO. : 21
DATE :
EXP. NO. :

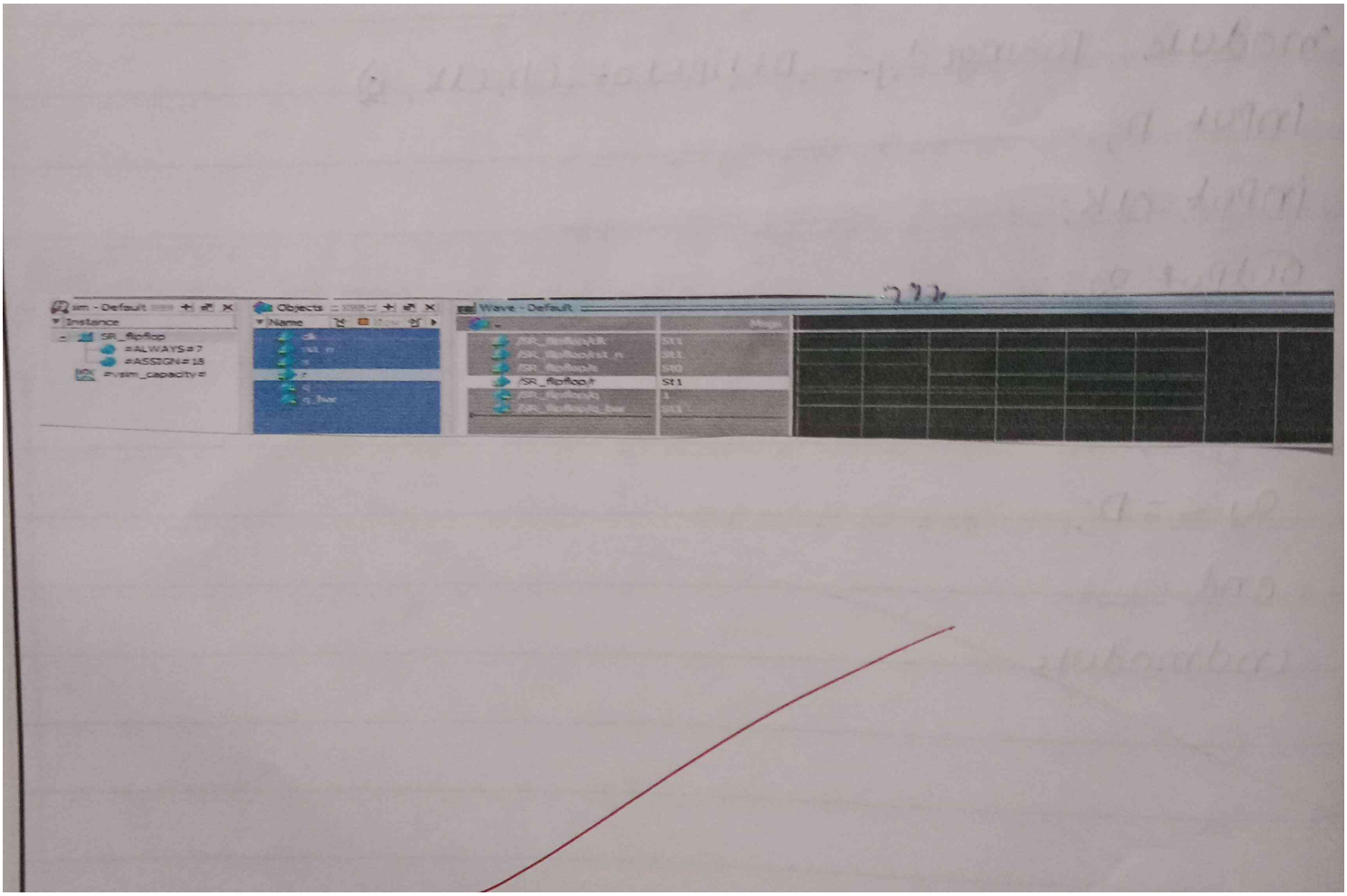
SR Flipflop

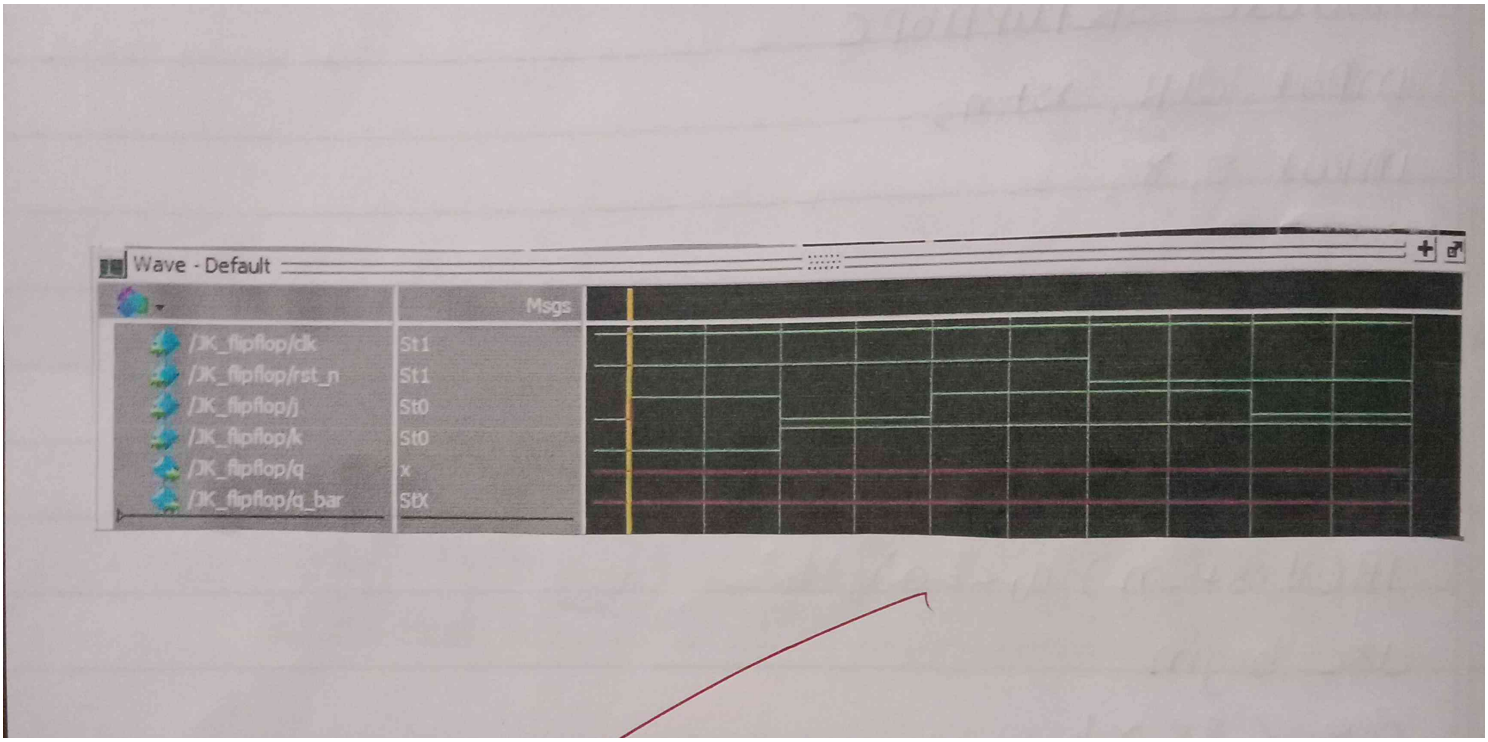
```
module SR flipflop(
input clk, rstn;
input S, R,
output reg Q,
output Q_bar
);
always @ Posedge clk) begin
if(!rst_n) Q<=0);
else begin
case ({S,R}
2'b00 : Q<=Q;
2'b01 : Q<=1'b0;
2'b10 : Q<=1'b1;
2'b11 : Q<=1'bx;
end case
end
end
assign Q_bar = ~Q;
end module
```

JK Flipflop

```
module JK_flipflop (
input clk, rst_n;
input S, K;
output reg Q.
output Q_bar
);
always @ (posedge clk) begin
    if (!rst_n) Q <= 0;
    else begin
    case ({S, J, K})
    2'b00 : Q = Q;
    2'b01 : Q <= 1'b0;
    2'b10 : Q <= 1'b1;
    2'b11 : Q <= ~Q;
    end case
     end
     end
     assign Q_bar = ~Q;
endmodule
```

05/12/24


sim - Default
Instance
SR_flipflop
#ALWAYS# 7
#ASSIGN# 18
#vsim_capacity#
Objects
Name
Wave - Default
/SR_flipflop/r
St1
St0


Wave - Default
Msgs
/JK_flipflop/clk
/JK_flipflop/rst_n
/JK_flipflop/j
/JK_flipflop/k
/JK_flipflop/q
/JK_flipflop/q_bar
St1
St1
St0
St0
x
StX