Date 13/02/025

M T W T F S

DSD

Mid Term

week # 01

Lecture # 01

→ verilog is HDL -

→ we describe hardware in verilog

1) Analyze the circuit
2) identify the input and outputs -
3) Label input and output -
4) Truth table
5) Simplify equation using K-map -
6) Implementation - (logic circuit)

easy-way

Model the behavior -

Half adder in verilog

name of the module

```
module HA (S, Cy, A, B);   → because continuously driven
output S, Cy;              → order is not important
input A, B;                → wire
assign {Cy, S} = A+B;      → wire, reg
```

pre-bit lower-bit

(model

use to Combinational circuit almost end module

we define hardware between the module and end module -

Common data types wire and reg

implementation ga-

synthesizes log

translation of the ~~input~~ discription to hardware

M T W T F S

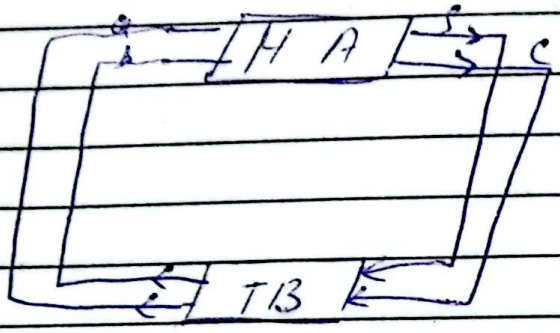
Date

assign use to concatinate.

→ translated into actual hardware

{ }, +, * synthesizable operator.

Now we dut (design under test)



reg
↓
not physical device
it is datatype.

```
module TsT_HA (S, Cy, A, B)
input S, Cy;
output A, B;
reg A, B;
initial begin
#10 A=0; B=0;
#10 A=0; B=1;
#10 A=1; B=0;
#10 A=1; B=1;
end
initial
$monitor("%d, A=%b, B=%b, S=%b, Cy=%b", $time, A, B, S, Cy);
endmodule
```

→ kisi ko value assign krni ho initial block mein to reg type use krna hoga.

→ vector generation part.

initial block is not synthisizable

delay

printf

→ when time change it will print.

simulation time

$display → display only once

$monitor → when time change it will

Date

M T W T F S

connection- Testbench and design

module top-level: to connect two modules

```
wire A, B, S, Cy;
HA ha (S, Cy, A, B);
tst- HA t-ha(S, Cy, A, B);
endmodule
```

data-flow model

primitive

xor (0, 1)

we can omit instance but in user-defined it will must-

→ Full-adder

→ ripple-carry adder.

Lecture # 2

How

Data level

it is a type of behavior modeling.

behavior mean different level of abstraction hai.

```
module CS, Cy, A, B);
input A, B;
output S, Cy;
assign [Cy, S] = A+B;
```

abstract

```
assign S = A ^ B;
assign Cy = A & B;
```

→ data flow

```
xor x1(S, A, B);
and a1(Cy, A, B);
endmodule
```

→ gate level

Date M T W T F S

→ Design mean synthesizable checker use hon gee.

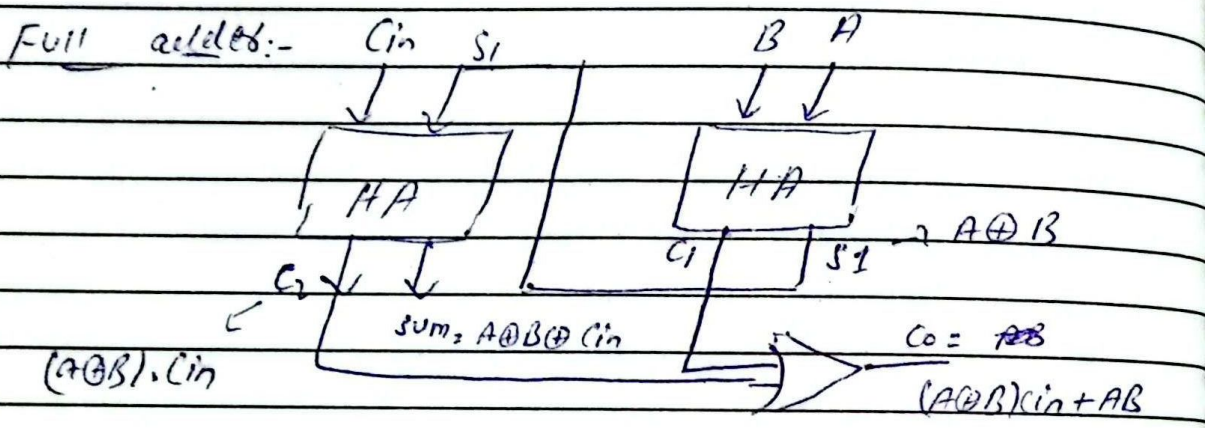
if inputs are fewer the exhaustive verification is possible.

① Design
② Specification
③ Synthesis / implementation
④ Verification
⑤ partitioning

DCA X → mean unknown
wire z → mean high impedence

In wave x is shown by red line.

Full adder:-



→ internal output declare using wire
→ output are special wire using type output.

```
module FA(Co, Sum, a, b, ci);

output Co, sum;
input a, b, ci;

wire s1, c1, c2;

HA ha1 (S1, C1, a, b);
HA ha2 (Sum, C2, S1, Ci);
```

→ structural level
↓
Block & and interconnection.

Date M T W T F S

```
or or1(Co, c1, c2);

endmodule


module tst_FA (Co, sum, a, b, c, ci);
  input Co, sum;
  output a, b, ci;

  reg a, b, ci;

  initial begin
  #5 a=0; b=0; ci=1
  #5 a=1; b=1; ci=1
  #5 a=1; b=0; ci=0
  end
  initial
  $monitor ("%d, a=%b, b=%b; ci=%b,
        "sum=%b", "cout=%b", $time, a, b, ci, sum,
                                        Co)
endmodule.


top
module top_level;
  wire Co, sum, a, b, ci;
  tst_FA tst (Co, sum, a, b, ci);
   FA fa (Co, sum, a, b, ci);
endmodule
```

In this top level is full adder and the lowest level is half adder.

HA

M T W T F S

Date

Week # 03

Lecture 03

RCA3 order not matter

module F.A (cout, sum, A, B, cin); explicit association

HA ha0 (.A(A), .B(B), .s(C1), .Cy(C0));

HA ha1 (Sum, C2, C0, Cin);

OR o1 (cout, C1, C2);

module RCA3 (sum, Cout, A, B, Cin);

//vector → after synthesis it will translated to bus

input [2:0] A, B;

input cin;

output [2:0] sum;

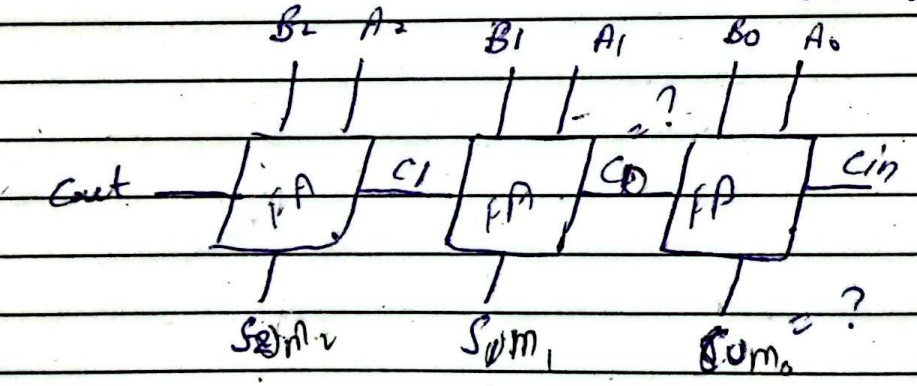
output Cout;

wire C0, C1;

FA fa0(sum[0], C0, A[0], B[0], 1'b0);

size binary ground

collection of wire



FA fa1 (sum[1], C1, A[1], B[1], C0);

FA fa2 (sum[2], Cout, A[2], B[2], C1);

endmodule

structural level modeling

Date MTWTFS

If Cin not here it will not be re-usable.

→ In verilog we have two block
initial and always.

initial
→ only one time
→ not (synthesizable)
↓
translate it actual hardware

always
↓
always change when event occur.
→ always running

we can write

reg cin; → not physical register
initial cin = 1'b0;

it's just type

module RCA3_tb(sum, cout, A, B, Cin);
input [2:0] sum; input cout;
output [2:0] A, B; output cin;

reg [2:0] A, B; reg cin; 2^3 = 128 possible exhausted verification

initial begin
A = 3'b101; B = 3'b001; cin = 1'b0;
#7 A = 3'b101; B = 3'b110; cin = 1'b0;
#10 A = 3'b110; B = 3'b001; cin = 1'b0;

initial
$monitor ("%d, A = %b, B = %b, sum = %b, cout = %b", $time, A, B, sum, cout);
endmodule.

[3:0)

M T W T F S

Date

```
module Top:
    wire [2:0] A, B, sum;
    wire cin, cout;

    RCA3 dut (sum, Cout, A, B, Cin);
    RCA3_tb tb(sum, Cout, A, B, Cin);
endmodule
```

6-bit RCA

→ final carry

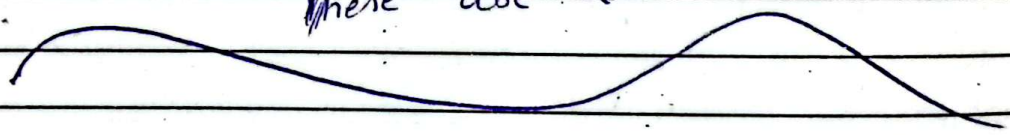
```
module RCA6 (sum, Ct, A, B);

    output [5:0] sum; output ct;
    input [5:0] A, B;
    wire [redacted] c;

    RCA3 inst0 (sum[2:0], c, A[2:0], B[2:0], 1'b0);
    RCA3 inst1 (sum[5:3], Ct, A[5:3], B[5:3], c);
endmodule.
```

↓

There are two blocks



→ gate level is also called structural level

→ we describe algorithm

→ behavioral level mean discription known.

⇒ scalers are wire

→ vectors are buses

next lecture

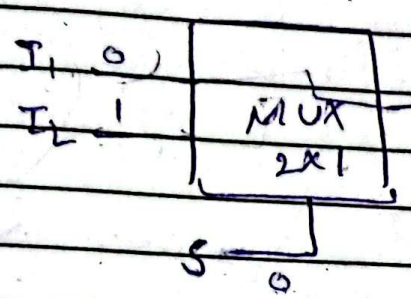
decoder and mux

Date

MTWTFS

Week # 03

Lecture # 02

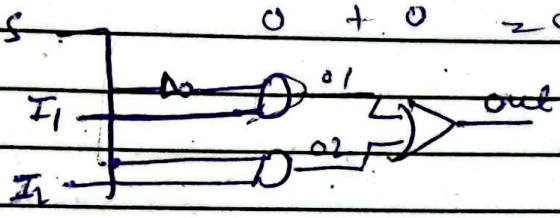


expression

$O = \bar{S} I_1 + S I_2$

$1(0) + 0(1)$

$0 + 0 = 0$



→ for behavior level only need to know the function of the MUX.

```
module m2to1 (out, x, I1, I2, s);

    output out;
    input I1, I2, s;

    wire sbar, o1, o2;

// assign out = !s?I1:I2;        → behavior level
// assign out = ~S&I1 | S&I2;    → Dat flow

    or o1 (out, o1, o2);          explicit
    and a2 (o2, s, I2);           structural
    not n1 (sbar, s);             code
    and a1 (o1, sbar, I1);  → B
```

Date

M T W T F S

→ In verilog statements are executed in parallel order not matter.

④ method always running or

```
B: reg out;
always @(I1, I2, S)        → when I1 change block will execute
                             event triggers.
                           → only input.
    if (S)
        out = I2;
    else
        out = I1;
```

→ alternate always @(*)

→ Synthesizable

endmodule.

★★★

→ if if statement is umpletly satisfied the circuit is combinational circuit - if not the sequential circuit will be ~~satisfied~~ - synthesize.

(5th) method → we don't write output here

```
always @(I1, I2, S)
    case (S)
        1'b0: out = I1;
        1'b1: out = I2;
    endcase
```

execute in sequential (combinational)

↑ blocking

=

<= → Non Blocking

↓

execute in parallel (sequential)

→ Both blocking and unblocking use in always block.

→ Blocking use in combinational circuit.

→ Non Blocking use in sequential circuit.

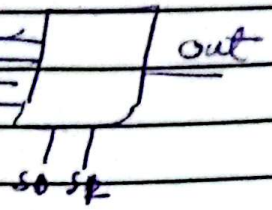
→ we can't write assign in always block because it is ~~combinational~~.

Date

M T W T F S

for 4x1

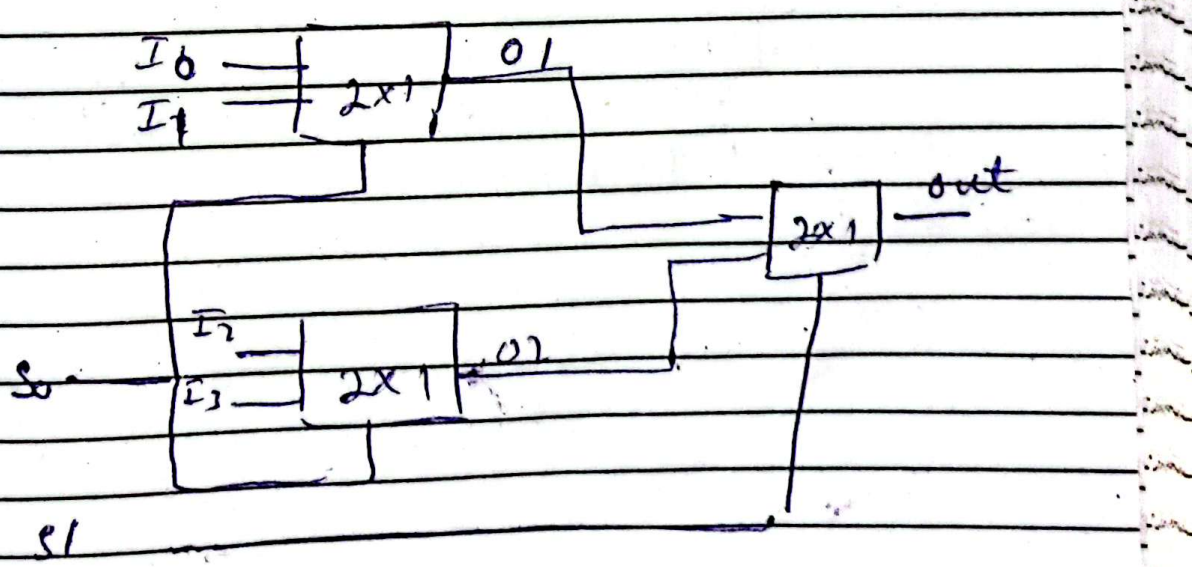
~~always @ (I1, I2, I3, I4, S)~~
~~case (S)~~
~~2'b00 : out = I1 ; I4~~




```
module m4to1 (out, I, s);
  output out;
  input [3:0] I;
  input [1:0] s;
  wire o1, o2;
  m2to1 m0(o1, I[0], I[1], s[0]);
  reg out;
  always @ (I, s)
    case (s)
      2'b00: out = I[0];
      2'b01: out = I[1];
      2'b10: out = I[2];
      2'b11: out = I[3];
    endcase
endmodule
```

out = $\bar{S}_0\bar{S}_1I_0 + \bar{S}_0S_1I_1 + S_0\bar{S}_1I_2 + S_0S_1I_3$

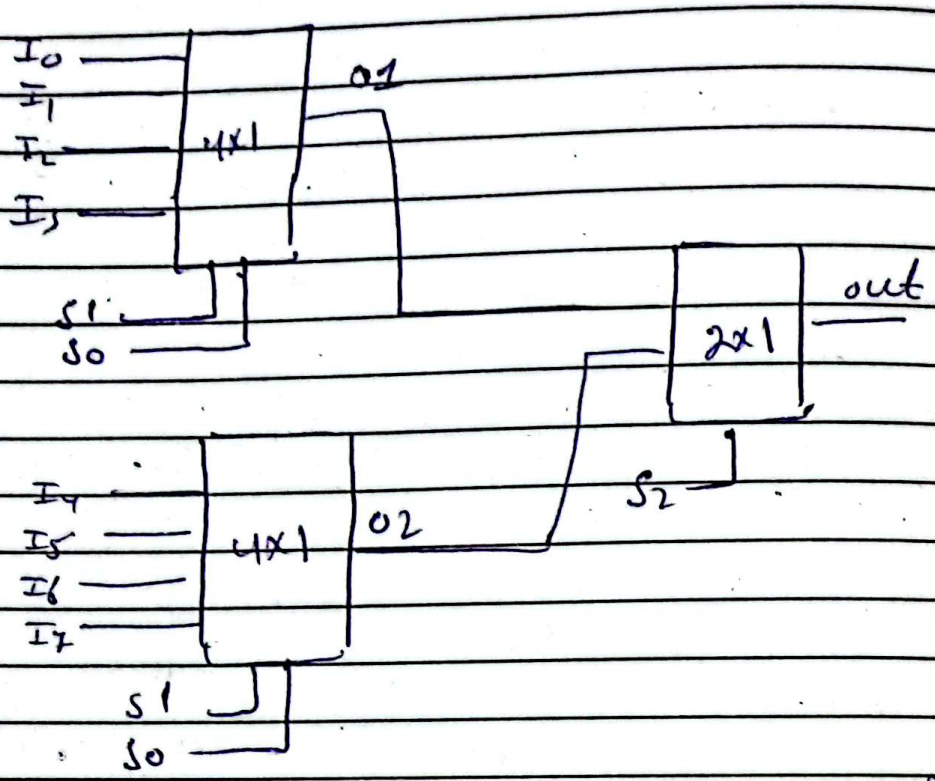
```
m2to1 m1(o2, I[2], I[3], s[0]);
m2to1 m2(out, o1, o2, s[1]);
```



Date

M T W T F S

8x1 MUX



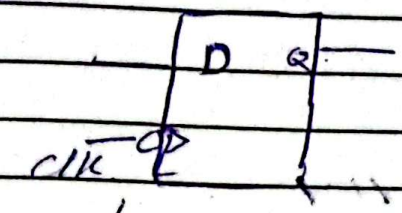

```
module m8to1 (out, I, S);
    output out;
    input [7:0] I;
    input [2:0] S;
    wire O1, O2;

    m4to1 m0 (O1, {I[0], I[1], I[2], I[3]}, S[1:0]);
    m4to1 m1 (O2, {I[4], I[5], I[6], I[7]}, S[1:0]);
    m2to1 m2 (out, O1, O2, S[2]);
endmodule
```

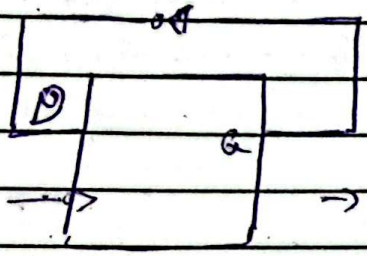
↗ concatinate operator

Date _____ Week # 04 M T W T F S

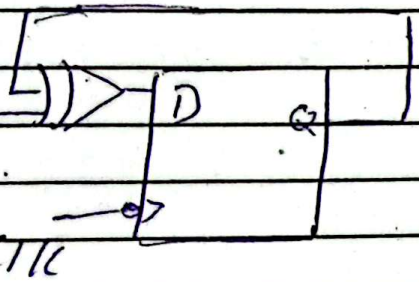
Lecture # 01:



↓
negative edge trigger



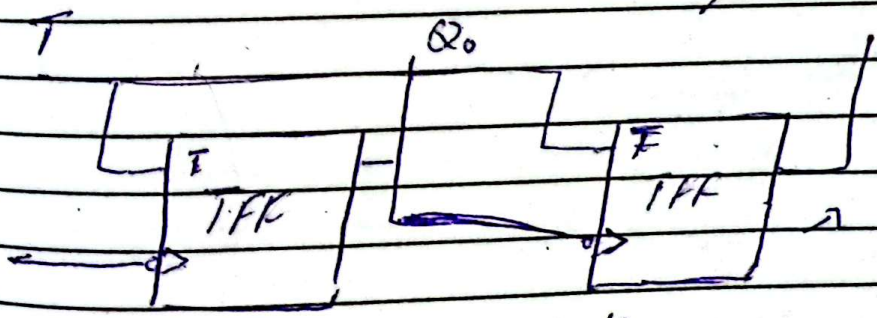
→ T using D-flip flop



T flip-flop

→ negative edge pr toggle hoga
→ possitive edge pr previous state maintain hogee

→ Ripple counter



→ untool counter because of T input

Ripple
Two bit counter 0 – 3

Date

M T W T F S

```
module D-FF (q, d, clock, reset);
  output q;
  input d, clock, reset;
  reg q;

  always @(negedge clock or posedge reset)
    if (reset)
      q = 1'b0;
    else
      q = d;

endmodule
```

⇒ Asynchronous

Iska data den to ye synchronous bn jye ga-

If negedge and posedge not written it will ~~not~~ become combinational or sequential depend on the discription-

```
module T-FF (q, d, clock, reset);
  output q;
  input clock, reset;
  wire d;

  not n1 (d, q);

  D-FF d0 (q, d, clock, reset);
endmodule
```

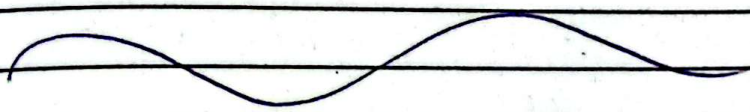
Date

M T W T F S

```
module ripple_counter (q, clock, reset);

    output [3:0] q;
    input clock, reset;

    T_FF tff0 (q[0], clock reset);
    T_FF tff1 (q[1], q[0], reset);
    T_FF tff2 (q[2], q[1], reset);
    T_FF ff3 (q[3], q[2], reset);
endmodule
```

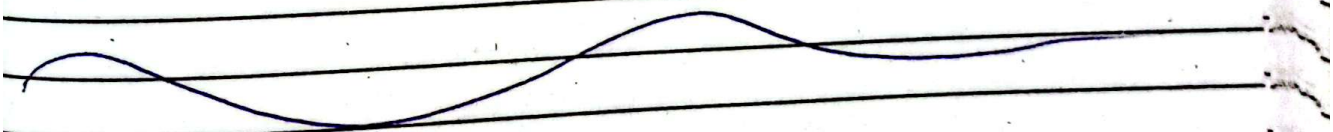


```
module T_FF (q, T, clock, reset);

    output [3:0] q;
    input T, clock, reset;

    wire d;

    xor x1 (d, T, q)
```

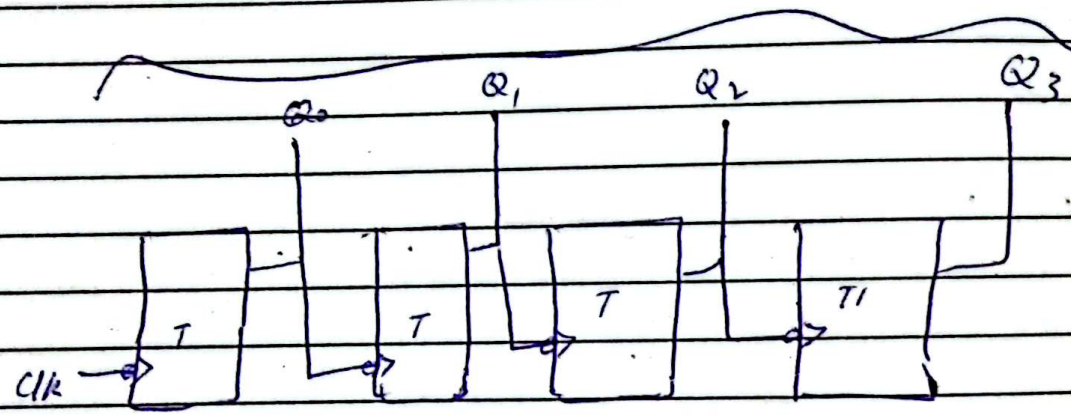


```
module stimulus (q, clock, reset);
    input [3:0] q;
    output clock, reset;
    reg clock, reset;

always
    #5 clock = ~clock;
```

Date

```
initial
begin
        clock = 1'b0;
        reset = 1'b1;
#15 reset = 1'b0;
#500 $finish;
end
// monitor the outputs
initial
$monitor ($time," T=%b, reset=%b, output
= %d", T, reset, q);
endmodule
```



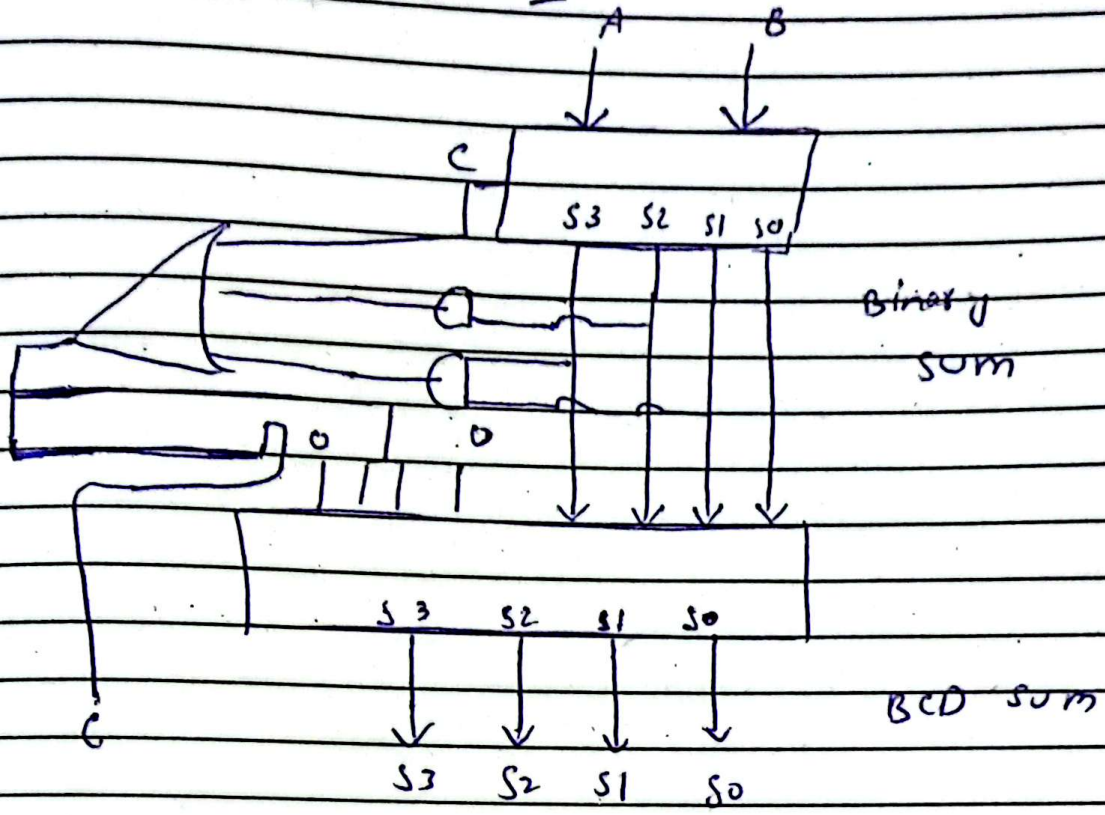
Q3	Q2	Q1	Q0
0	0	0	0
0	0	0	1

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

Date

M T W T F S

Quiz # 01




```
module add4 (C, S, A, B);

    input [3:0] A, B;
    output [3:0] S;
    output C;
    assign {C, S} = A+B;
endmodule

module BCD (C, S, A, B);
    input [3:0] A, B;
    output [3:0] S;
    output C;

    wire Cy, Cz;
    wire [3:0] sum;
    add4 a1 (Cy, sum, A, B);
    assign C = Cy | (sum[3]&sum[2]) | (sum[3]&
                                       sum[1]);
    add4 a2(Cz, S, {1'b0, C, C, 1'b0}, sum);
endmodule
```