Modelling

i) Behavioral ii) Dataflow iii) Gate level (structural) iv) Switch lvl.

Behavioral

This is the highest lvl of abstraction provided by Verilog. A module can be written in terms of the desired design algorithms without concern for the hardware implementatn detail.

Dataflow.

Module is designed by specifying data flow. The knowledge of Boolean expression & flow of data betn hardware registers & gates

Gate lvl

module is implemented in terms of logic gates & interconnections betn gates.

Switch lvl

Lowest lvl of abstraction. Here module is implemented in terms of switches storage nodes & connectn betn them.

FEBRUARY - 2020

M	T	W	T	F	S	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	

~ - not

^ - XOR

(002-364) Wk 01

JANUARY

THURSDAY

02

#~~Dataflow~~ Modeling

```
module halfsub(d, bw, a, b);
output d, bw;
input a, b;
assign d = a^b;
assign bw = (~a) & b;
endmodule
```

```
module fullsub (d, bw, a, b, c);
output d, bw;
input a, b, c;
assign d = a^b^c;
assign bw = (~a & b) | (~a & c) | (b & c)
endmodule
```

FEB MAR APR MAY

JANUARY - 2020

M	T	W	T	F	S	S	M	T	W	T	F	S	S
		1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24	25	26
27	28	29	30	31									

Lexical conventions.

① White space

Tab, Blank spaces, NewLine ⟹ ignored except in strings.

② Comment

// one line commenting

/* multiple Lines. */

Unsized Numbers

12345 ⟶ Base → decimal (default)

32-bit M/c specific.

16'b 0000_1101_0010_0110 ← underscore is allowed.

Strings:

"Hello world"

Nets:

Wire default 'z'

wand

wor

tri

trireg → default 'x'

triand

trior.

Nets represent connections betn hardware elements. They have values continuously driven on them by the o/p's of devices they are connected to.

Registers :
They represent data storage elements. They retain a value until another value is ~~driven~~ placed onto them
Default - 'x'

```
Vectors
wire a;
wire [7:0] b;
reg [3:0] c;
reg [0:15] addr;
reg [3:0] x;
reg [0:3] y

x = b[3:0]
y = b[0:3]
x = b[0:3]  // illegal since defined as (3:0).
```

Variable vector
Part select

```
reg [0:255] data1;
reg [255:0] data2;
reg [7:0] byte;

byte = data1[24+:8]
```

Sunday 05

06 (006-360) Wk 02
JANUARY
MONDAY

JANUARY - 2020

M	T	W	T	F	S	S	M	T	W	T	F	S	S
		1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24	25	26
27	28	29	30	31									

byte = data 2[31 -: 8] 31-8 places
9 i.e. = data 2[31:24]

10 byte2 = data1[24 +: 8]

11

```
Module mux(out, in, s)
12 output out;
   input [3:0] in;
1  input [1:0] s;

2  assign out = in[3] & (~s[0] & ~s[1]) |
               in[2] & (s[0] & ~s[1]) |
3              in[1] & (~s[0] & s[1]) |
               in[0] & (s[0] & s[1])
```

4

5

Integer, Real & time data types.

6

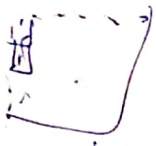
```
integer i
7 initial
  begin
    i=2;
    i=-3; etc.
  end
```

```
Real delta
initial
  delta = 2.63;
```

```
Time sim_time
initial
sim_time = $time
```

begin - end for multiple initialization.

$time predef fn for finding time.



9 Array
reg a[0:7]
a ⇒ array of 8 element each elements of width 1.

10

reg [3:0] b[0:7]
b ⇒ array of 8 elements each element of width 4.

11

~~#not synthesizable~~
12 integer array1[7:0][3:0]; ; 4 column 8 rows

1 Wire [1:0] c[15:0]; → 16 elements of 2 bits each

2

Memories
3 reg membit[0:1023]

4 Parameters
Const. can be defined in a module by the ~~C~~ keyword parameter.
5 they cannot be used as variable

6 parameter size = 1023
parameter width = 7
7 reg membit [0:size]
reg [width:0] mem [0:size]
localparam c=15;

08 (008-358) Wk 02
JANUARY
WEDNESDAY

JANUARY - 2020

M	T	W	T	F	S	S	M	T	W	T	F	S	S
		1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24	25	26
27	28	29	30	31									

```
module parity(p, op, data)
output p, op;
input [7:0] data;

assign p = d[7]^d[6]^d[5]^d[4]^d[3]^d[2]^d[1]^d[0];
       op = ~(d[7]^d[6]^d[5] . . . . . ^d[0]);
```

Strings

Strings can be stored in reg

```
reg [8*18:1] string_a;
string_a = "Hello Everybody";
```

Display

```
$display("Hello")
$display($time)

reg [0:15]addr;
$display("At time %d the addr is %d", $time, addr);
```

decimal - d
Binary - b
ASCII - c

%m - module name.
%e ⇒ real no. in scientific 4e2.
%f ⇒ real decimal

Remember you are special. No one can play your role better than you.

Scanned with CamScanner

123
0111 1011


```
reg a;
initial
begin
$monitor ($time, "value of a is %b", a);
end

0   value of a is 0
5   value of a is
;
;

#100 $stop        = wait for 100ns
#900 $finish
```

Ex. 3.5

a) write 123 in binary.

8'b0111_1011

b)

16'h xxxx

c) 4 bit -ve 2.

~~4'b2~~ -4'd2

d) unsized hex no. 1234

'h 1234

%o%

2. Are following Right or not

a) "This is a string displaying the % sign". Wrong
%%

b) "out = in1 + in2" correct.

c) "Please ring a bell \007" correct
→ \007 is octal code for character. \007 - ●

d) "This is a backslash \ character \n". wrong
→ " " " " \\ " " "

" ~~This is a * character~~

" ~~This is a **~~

e) "This is a asterick * character"
f) " " " " hash \# " "

4. Declare foll. variables.
a) 8-bit net called a_in
wire [7:0] a_in;

b) 32-bit reg. called address & set to 32 bit decimal 3.
reg [31:0] address = 32'd3;
~~address32~~

c) integer called count
→ integer count.

d) time variable snap_shot
→ time snap_shot

e) array called delays. Array contains 20 elements of type integer
→ integer ~~array~~ delay [19:0]

f) A memory MEM containing 256 words of 64 bits each
reg [63:0] MEM [255:0]

g)
parameter cache_size = 512.

5.a) latch = 4'd12 // latch should integer.

The current value of latch = ~~xxx~~ 1100

b) In register value = 2

2:4 decoder.

```
module decoder (out, in);
input [1:0] in;
output [3:0] out;
assign out[0] = (~in[0]) & (~in[1])
       out[1] = (~in[0]) & (in[1])
       out[2] = (in[0]) & (~in[1])
       out[3] = (in[0]) & (in[1])
```

Sunday 12

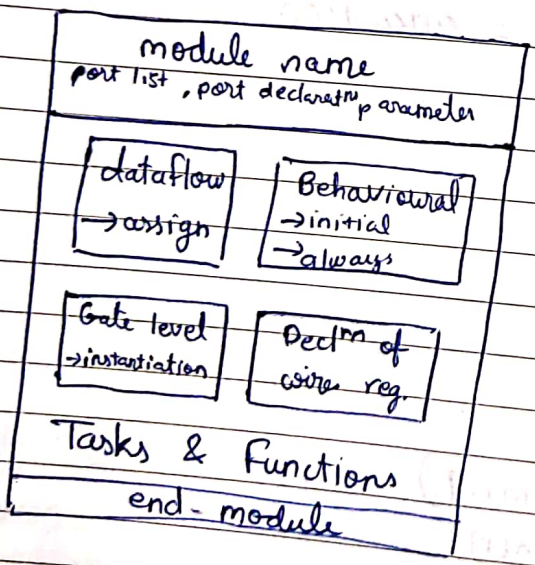
13 (013-353) Wk 03
JANUARY
MONDAY

JANUARY - 2020

M	T	W	T	F	S	S	M	T	W	T	F	S	S
		1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24	25	26
27	28	29	30	31									

```
module encoder(out, in)
input [3:0] in;
output [1:0] out;
assign
```

Modules




```
// a, b
reg
wire
full ad
initial
begin
a <=
#5
```

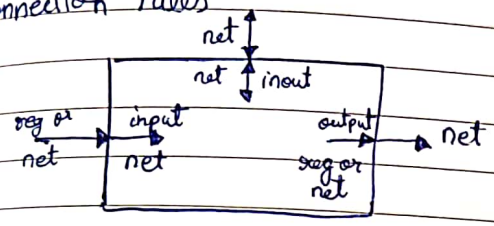
```
#
end
end m
```

A person's true wealth is the good he or she does in the world.

PORTS

input / output / inout

Connection rules



Inside module :
inputs ⇒ wire
outputs ⇒ reg

In testbench (from module)
inputs ⇒ reg
output ⇒ wire

Testbench for Full Adder

```
module testadd ;
//a, b, c , sum , carry
  reg  a, b, c ;
  wire  sum, carry ;
full_adder  f1 (sum, carry , a, b, c);   // same order as module
initial
begin
a <= 1'b0 ; b <= 1'b0 ; c <= 1'b0;
#5  a <= 1'b0 ; b <= 1'b0 ; c <= 1'b1
  .
  .
  .
#  a <= 1'b1 ; b <= 1'b1 ; c <= 1'b1
end
end module .
```

That which he selfishly keeps, he loses.

15 (015-351) Wk 03
JANUARY
WEDNESDAY

JANUARY - 2020

M	T	W	T	F	S	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

```
9  testbench for mux
   module tmux
10 reg [3:0] in;
   reg [1:0] s;
11 wire out;
   mux M1(out, in, s)
12 initial
   begin
1  in[3] <= 1'b0 ; in[2] <= 1'b1 ; in[1] <= 1'b0 ; in[0] <= 1'b1;
   s[1] <= 1'b0 ; s[0] <= 1'b0 ;
2  #5 s[1] <= 1'b0 ; s[0] <= 1'b1 ;
   #5 s[1] <= 1'b1 ; s[0] <= 1'b0 ;
3  #5 s[1] <= 1'b1 ; s[0] <= 1'b1;
   end
4  end module
```

in <= 4'b0000
s <= 2'b00
s <= 2'b01

5

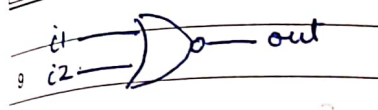
Gate level modelling

6 Verilog supports basic logic gates as predefined primitives. These primitives can be instantiated like modules. There are
7 two classes of basic gates. One is and/or & Buf/Not.

AND/OR

AND : OR NAND
NOR XOR XNOR

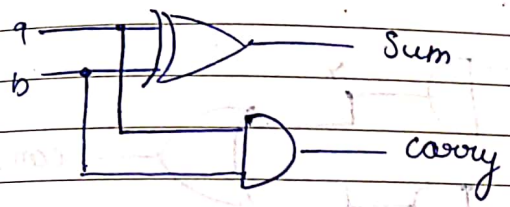
and a1(out, i1, i2);



nor n1 (out, i1, i2);

no. of i/p's can be extended

Code for Half adder



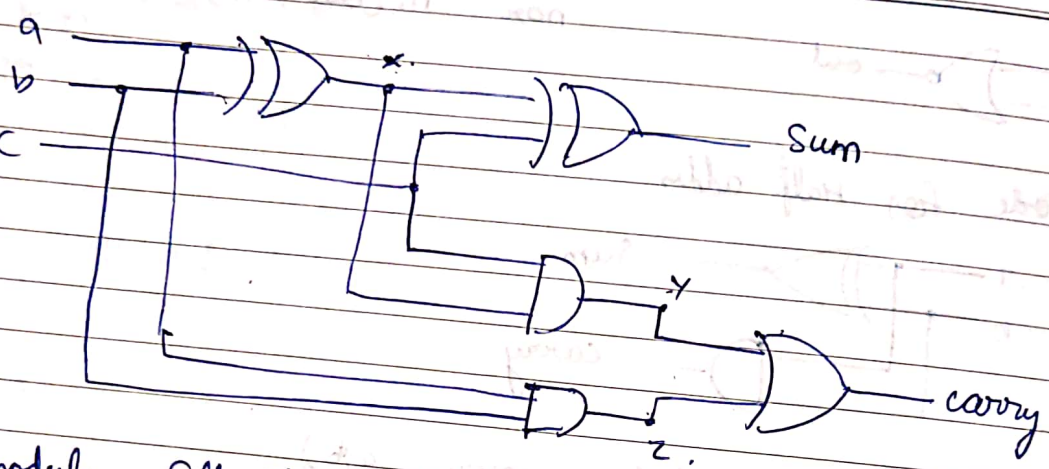

```
module halfadd (sum, carry, a, b)
output sum, carry;
input a, b;
reg sum, carry;
wire a, b;
xor x1 (sum, a, b);
and a1 (carry, a, b);
end module
```

same testbench.

17 (017-349) Wk 03
JANUARY
FRIDAY

JANUARY - 2020

M	T	W	T	F	S	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		




```
module fulladd (sum, carry, a, b, c)
output sum, carry;
input a,b,c;
xreg & sum, carry;
wire a,b,c, x, y, z;
XOR  x1 (x, a, b);
XOR  x2 (sum, x, c);
AND  A1 (y, x, c);
AND  A2 (z, a, b);
OR   O1 (carry, y, z);
end module.
```

A man's dreams are an index to h

JANUARY - 2020
M T W T F S S
6 7 8 9 10 11 12
20 21 22 23 24 25 26

- carry

FEBRUARY - 2020
M T W T F S S
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29

(018-348) Wk 03
JANUARY
SATURDAY 18

Gate Delays : Gate level Modeling.

1. Rise
2. Fall
3. Turn-off Delay
   o/p delay goes to z from any value.

If one delay is specified then this value is used for all 3 transitions.
If two values are specified then they correspond to rise & fall. Minimum of two is considered for turn off delay.

case 1
and #(5) a1 (out, in1, in2);
rise
fall
turn-off } 5

case 2
and #(3,4) a1 (out, in1, in2);
rise = 3
fall = 4
turn-off = 3.

Sunday 19

case 3
and #(3,4,5) a1 (out, in1, in2);
rise = 3
fall = 4
turn off = 5

FEB MAR APR MAY

Toleration is the greatest gift of the mind

JANUARY - 2020

M	T	W	T	F	S	S	M	T	W	T	F	S	S
		1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24	25	26
27	28	29	30	31									

+min delays
→ at run time.
designer can specify the min value of the delay that the designer +min delays.

+type delays
typical value can be specified
+typdelays

+max delays.
Maximum value can be specified as +max delay

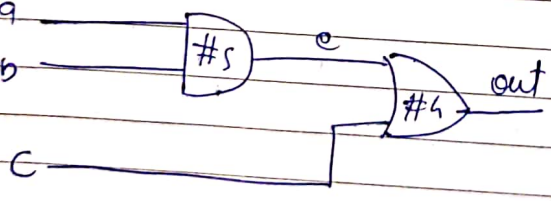
rise time
min = 3
typ = 4
max = 5

fall time
min = 4
typ = 5
max = 6

turn-off Delay
min = 5
typ = 6
max = 7

and #(3:4:5, 4:5:6, 5:6:7) a1 (out, in1, in2);

Q)

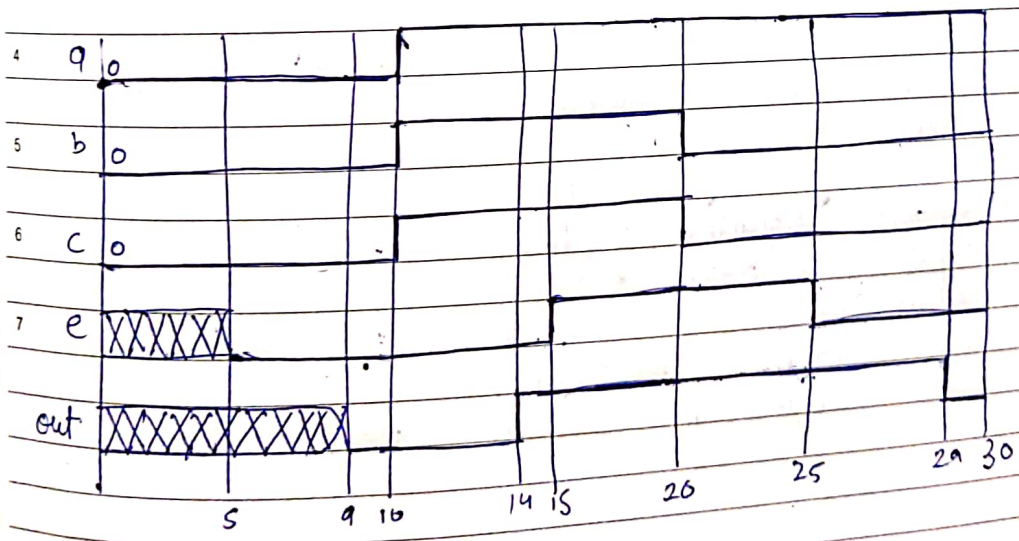



```
module D(out, a, b, c)
output D;
input a, b, c;
wire e;
```

One man gives freely, yet gains even more;

AND ~~x1(e,a,b);~~ #(5) x1(e,a,b);
OR ~~x1(out,e,c);~~ #(4) y1(out,e,c);
endmodule;

module test
wire out;
reg a,b,c;
d1 d1(out,a,b,c);
initial
begin
a <= 1'b0; b <= 1'b0; c <= 1'b0;
#10 a <= 1'b1; b <= 1'b1; c <= 1'b1;
#10 a <= 1'b1; b <= 1'b0; c <= 1'b0;
endmodule



22 JANUARY WEDNESDAY (022-344) Wk 04

Ripple Carry.

JANUARY - 2020

M	T	W	T	F	S	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

assume fulladder is written.

```
fulladd f1 (Sum, carry, a, b, c);

module RCA (Sum, Cout, A, B, Cin);
input [3:0] A, B;
input Cin;
Output [3:0] S;
Output Cout;
wire C1, C2, C3;
fulladd fa0 (S[0], C1, a[0], b[0], Cin);
fulladd fa1 (S[1], C2, a[1], b[1], C1);
fulladd fa2 (S[2], C3, a[2], b[2], C2);
fulladd fa3 (S[3], Cout, a[3], b[3], c3);
endmodule;
```

assume JKFF is written.

```
JKff J1(
```

```
module Counter (Q, clk);
wire JK;
output [3:0] Q;
wire J1, K1, J2, K2, J3, K3, J0, K0;
wire Q0bar, Q1bar, Q2bar, Q3bar;
```

If there is no struggle, there is no progress.

```
9  module Counter (Q, clk);
   input clk;
10 output [3:0] Q;
   wire [1:0]Jk;
11 assign Jk = 1'b1;
   wire Q0bar, Q1bar, Q2bar, Q3bar;
12 JKFF J0 (Q[0], Q0bar, Jk, Jk, clk);
   JKFF J1 (Q[1], Q1bar, Jk, Jk, Q[0]);
1  JKFF J2 (Q[2], Q2bar, Jk, Jk, Q[1]);
   JKFF J3 (Q[3], Q3bar, Jk, Jk, Q[2]);
2  endmodule.

3  module JKFF (Q, Qbar, Jk, clk);
   output Q, Qbar;
4  input [1:0] Jk;
   input clk;
5  always @(negedge clk)
   begin
6  case (Jk)
7  2'b00 : temp <= temp;
   2'b01 : temp <= 1'b0;
   2'b10 : temp <= 1'b1;
   2'b11 : temp <= ~temp;
   end
   endmodule;
```

24 (024-342) Wk 04
JANUARY
FRIDAY

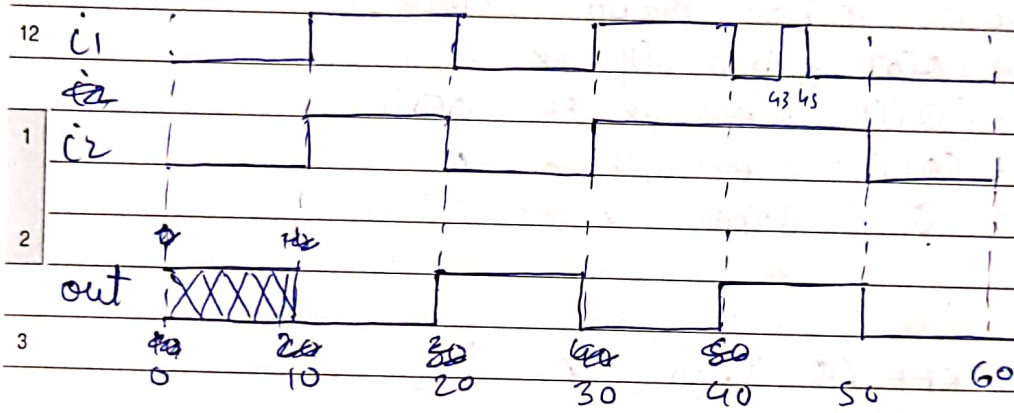
JANUARY - 2020
M T W T F S S M T W T F S S
1 2 3 4 5 6 7 8 9 10 11 12
13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31

Dataflow modeling.

Delays.

① Regular Assignment Delay

```
assign #10 out = i1 & i2;
```



BCD to Gray conv. in Gate lvl.

Ex1 · A = 2 ; B = 0;
A && B evaluates to 0
A || B evaluates to 1.

Ex2 · A = 2'b0x B = 2'b10
A && B evaluates to x.

A wise man knows how little he knows.

Ex 3: a = 2
b = 3
(a == 2) && (b == 3) evaluates to 1

(3) Relational.
>
<
>=
<=

equality
==
case equality
!==

A = 4 B = 3
X = 4'b1010 Y = 4'b1101
Z = 4'b1xxz , M = 4'b1xxz N = 4'b1xx

1) A == B '0'
2) X != Y '1'
3) X == Z 'x'
4) Z === M '1'
5) Z === N '0'
6) M !== N '1'

27

(027-339) Wk 05

JANUARY

MONDAY

JANUARY - 2020

M	T	W	T	F	S	S	M	T	W	T	F	S	S
		1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24	25	26
27	28	29	30	31									

Bitwise.

& and 2

| or 2

^ xor 2

~ not 1

~^ xnor 2

Reduction

&

~&

|

~|

^

~^.

a = 4'b0011

b = 4'b1111

a&b = 0011

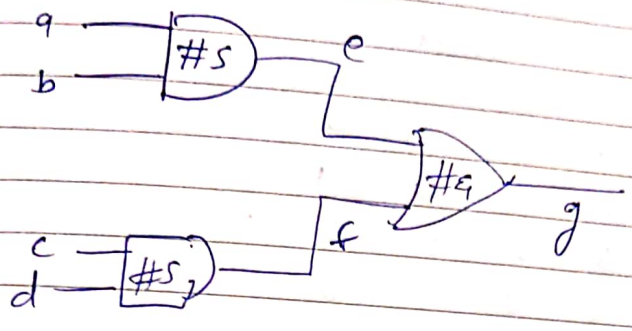
&a = 0

&b = 1

and all digits in aa.

The execution

Using implicit assignment write code.



Regular

```
module ckt(g, a, b, c, d)
input  a, b, c, d;
output g;
wire   e, f;
assign #5 e = a & b;
assign #5 f = c & d;
assign #4 g = e | f;
endmodule.
```

Implicit

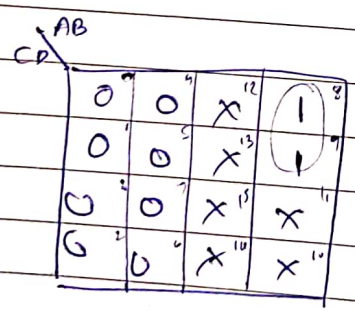
```
module ckt(g, a, b, c, d);
o/p    g
i/p    a, b, c, d;
wire   #5 e = a & b;
wire   #5 f = c & d;
assign #4 g = e | f;
endmodule.
```

JANUARY
M T W T F S S M T W T F
1 2 3 4 5 6 7 8 9 10
13 14 15 16 17 18 19 20 21 22 23 24
27 28 29 30 31

9 BCD to Gray.

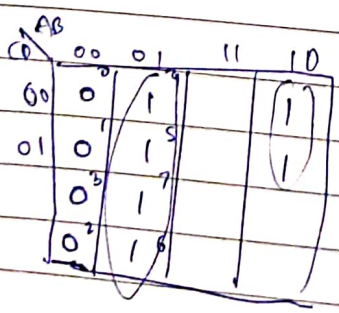
BCD				Gray			
A	B	C	D	P	Q	R	S
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1

For P



$P = A\bar{B}\bar{C}$

For Q

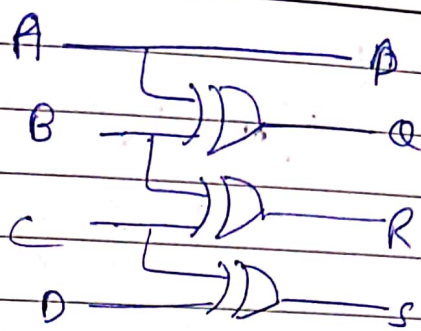


$Q = B + A\bar{B}\bar{C}$. $A+B$.

Lack on money is no obstacle. Lack of an idea is an obstacle

$S = C \oplus D$

$R = B \oplus C$




```
module BCDtoGray (P,Q,R,S,A,B,C,D)
output P,Q,R,S;
input A,B,C,D;
buf b1(P,A);
or o1(Q,A,B);
xor x1(R,B,C);
xor x2(S,C,D);
endmodule
```

BCD to XS-3.

A	B	C	D	P	Q	R	S
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

JANUARY - 2020
M T W T F S S M T W T F S S
1 2 3 4 5 6 7 8 9 10 11 12
13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31

P

CD \ AB	00	01	11	10
00	0 (0)	0 (4)	X (12)	1 (8)
01	0 (1)	1 (5)	X (13)	1 (9)
11	0 (3)	1 (7)	X (15)	X (11)
10	0 (2)	1 (6)	X (14)	X (10)

$P = A + BD + BC = A + B(D+C)$

$Q =$

CD \ AB	00	01	11	10
00	0 (0)	1 (4)	X (12)	0 (8)
01	1 (1)	0 (5)	X (13)	1 (9)
11	1 (3)	0 (7)	X (15)	X (11)
10	1 (2)	0 (6)	X (14)	X (10)

~~$Q = AD + \bar{A}\bar{B}D + \bar{A}\bar{B}C + B\bar{C}\bar{D}$~~

~~$Q = AD + \bar{A}\bar{B}(D+C) + BC\bar{D}$~~

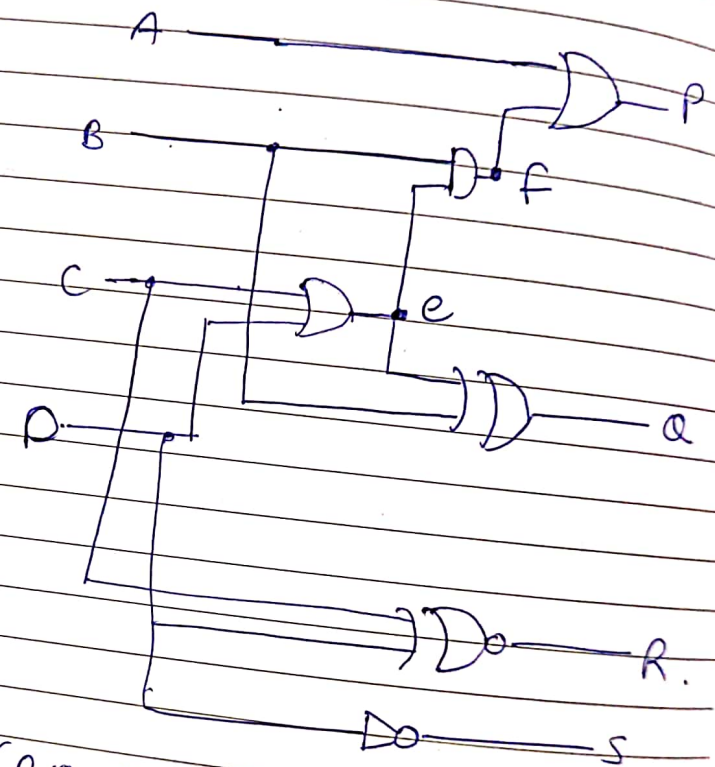
$Q = \bar{B}D + \bar{B}C + B\bar{C}\bar{D}$
$= \bar{B}(D+C) + B\bar{C}\bar{D}$
$= B \oplus (C+D)$

$R = C \odot D$

$S = \bar{D}$

The more you chase money, the harder it is to catch it.

BCD to XS-3.




```
module BCD to XS3 (P, Q, R, S, A, B, C, D)
output P, Q, R, S;
input A, B, C, D;
wire e, f;
or o1(e, c, d);
and a1(f, b, e);
or o2(P, A, f);
not n1(s, D);
xnor xn1(R, C, D);
end module.
```

Concatenation

{ }

A = 1'b1
B = 1'b0
C = {A, B}
C = 2'b10

A = 1'b1;
B = 1'b0;
C = {4{A}, 2{B}}

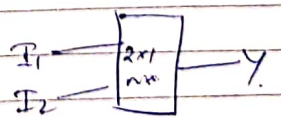
C = 6'b111100

Conditional operator.
(condition) ? true expr : false expr;

(a == 1'b1) ? b = a : b = ~b;

2:1 mux using Condnal opr.

assign y = s ? I2 : I1;



FEBRUARY - 2020

M	T	W	T	F	S	S	M	T	W	T	F	S	S
					1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29								

4:1 mux.

```
assign y = S1 ? (S0 ? I3 : I2) : (S0 ? I1 : I0);
```

~~code for t FF.~~

SR Latch.

```
module SRL(Q, S, R)
inout Q;
  input S, R;
assign S ? (R ? 1'bx : 1'b1) : (R ? 1'b0 : Q);
endmodule
```

Behavioural Modeling

```
module Example;
reg a,b,c,x,y;
initial
  x = 1'b0;
initial
begin
  a = 1'b0
#5  b = 1'b1
#10 c = 1'b1
end
```

```
initial
begin
#15 x = 1'b1;
#5  y = 1'b1;
end
initial
#30 $finish
endmodule
```

In the practice of tolerance, one's enemy is the best teacher.

```
module Ecample
  reg clock;
initial
  clock = 1'bo;
```

case II reg clock = 1'bo

case III module example (a, b, c);
  input b, c;
  output [7:0] a

~~module example (a~~

case IV

```
module example (output reg [7:0] a, input b, c,
                input [3:0] p, output reg x);
initial
  x = 1'bo;
endmodule.
```

always

→ Starts at time 0 and executes all statements inside the always block continuously in a looping manner.

```
module clockgen (output reg clock);
initial
clock = 1'bo;
always
#5 always = ~ clock
  endmodule.
```

06

(037-329) Wk 06

FEBRUARY

THURSDAY

FEBRUARY - 2020

M	T	W	T	F	S	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	

```
module srff(output, q, qbar, input, clk, s, r);
reg q, qbar;
reg temp; = 1'b0;
always @(posedge clk)
begin
case ({s, r})
2'b00 : temp <= temp;
2'b01 : temp <= 1'b0;
2'b10 : temp <= 1'b1;
2'b11 : temp <= 1'bx;
end case
q <= temp;
qbar <= ~temp;
end;
endmodule;

module Tff(q, qbar, t, clk)
output reg q, qbar;
input t, clk;
reg temp;
initial
begin
temp <= 1'b0
always @(negedge clk)
begin
case (T)
1'b0 : temp <= temp;
1'b1 : temp <= ~temp;
endcase
```

Our task now is not to fix the blame for the past, but to fix the course for the future.

```
9   q <= temp;
    qbar <= ~temp
10  end
    endmodule
11
```

Blocking and Non-blocking Assignments

12

```
    reg [15:0] a, b;
1   reg x, y, z;
    reg clock;
2   initial
    begin
3   x = 1'b0 ; y = 1'b1 ; z = 1'b1 ;
    a = 16'b0 ;
4   b = a;
#10   a[12] = 1'b1 ;          -> at time 10
#15   b[15:13] = {x, y, z};   -> at time 25
    clock = 1'b1 ;            -> at time 25
6   end
```

7 = blocks next statement till it is executed i.e value is assigned
→ order matters

<= non-blocking : any order

another way of delay instr.

```
    a[12] = #10 1'b1 ;
    b[15:13] = #15 {x, y, z};
```

08 (039-327) Wk 06
FEBRUARY
SATURDAY

FEBRUARY - 2020
M T W T F S S M T W T F S S
1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29

```
reg a,b;
always @(posedge clk);
a=b;
always @(posedge clk);
b=a;
```

Race conditions.
Not appropriate

```
reg a,b
always @(posedge clk);
a<=b;
always @(posedge clk);
b<=a;
```

} appropriate

Parallel Block & sequential blocks.

begin -end block is sequential.

```
reg x,y,z;
reg [1:0] a;
initial
begin
    x='b0;
    y='b1;
#10 z='b0;
#15 a={x,y};
end
```

} sequential block assignment.

time	Ev
0	x=0 y=1
10.	z=0
25	a=01;

09 Sunday

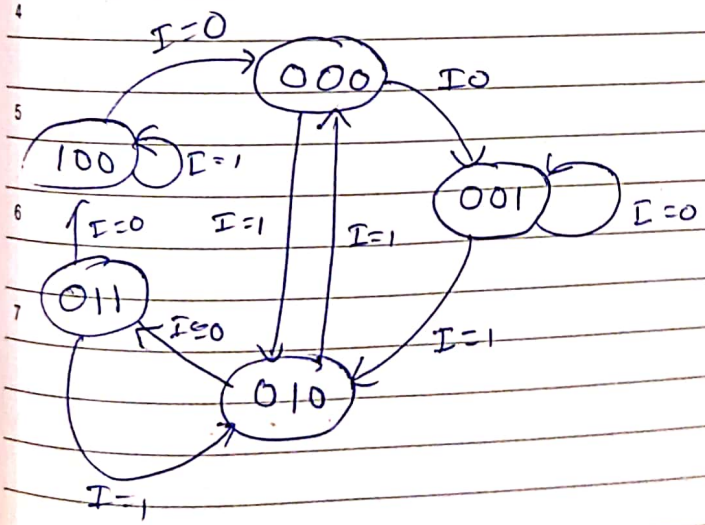
Discontent is the first step in the progress of man or a nation.

```
reg x,y,z;
reg [1:0]a;
initial
fork
x=1'b0;
y=1'b1;
z=1'b0;
#10 z=1'b0;
#15 a={x,y};
join
```

parallel block.

tim	
0	x=0 y=1
10	z=0
15	a=01

fork-join is parallel.



State	O/P
000	000
001	111
010	010
011	101
100	100

10

FEBRUARY - 2020

M	T	W	T	F	S	S	M	T	W	T	F	S	S
					1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29								

```
# for state diag. behind
module FSM(I, clk, Y, rest);
output reg [2:0] y;
input I, clk, rst.
parameter s0 = 3'b000;
parameter s1 = 3'b001;
parameter s2 = 3'b010;
parameter s3 = 3'b011;
parameter s4 = 3'b100;
reg [2:0] ps, ns;
initial ps=s0;
always @(posedge clk);
begin
if(rst == 1'b1)
ns = s0;
else
case (ps)
s0: if(I== 1'b0)
    ns <= s1; else
    ns <= s2;
s1: if (I== 1'b0)
    ns <= s1; else
    ns <= s2;
s2: if (I== 1'b0)
    ns <= s3; else
    ns <= s0;
```

Try not to become a man of success but rather to become a man of value.

FEBRUARY - 2020

F	S	S	M	T	W	T	F	S	S
		1	2	3	4	5	6	7	8	9
14	15	16	17	18	19	20	21	22	23
28	29								

MARCH - 2020

M	T	W	T	F	S	S	M	T	W	T	F	S	S
						1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17	18	19	20	21	22
23	24	25	26	27	28	29	30	31					

(043-323) Wk 07

FEBRUARY

WEDNESDAY

12

```
9   S3: if (I== 1'b0)
        ns <= S4; else
10      ns <= S2;
    S4: if (I == 1'b0)
11      ns <= S0; else
        ns <= S4;
    end case
12  ps <= ns;
    end
1   always @ (posedge clk)
    begin
2       if (rst == 1'b1)
            y <= 3'b000;
3       else
    case (ps)
4   S0: y <= 3'b000;
    S1 : y <= 3'b111;
5   S2 : y <= 3'b010;
    S3 : y <= 3'b101;
6   S4 : y <= 3'b100;
    end case
7   end
```

of value.

The time to repair the roof is when the sun is shining.

MAR

APR

MAY