VLSI - Assignment (2013-Main)

Tasneem Nazneen
1604-11-735-067

Part-A

1) Write two example for system tasks and compiler directives.

System task:

An identifier beginning with a $ character is interpreted as a system task or system function.

Ex:- $display, $monitor, $stop, $finish are some of the system tasks.

Eg:1): ($ display task)

// Display the string in quotes

$ display ("Hello verilog world");

2) (monitor statement)

// monitor time and value of the signal clock and reset

// clock toggle every 5 time units and reset goes down at 10 time units

initial begin

$ monitor ($time, "value of signals clock = %b reset = %b", clock, reset);

end.

Compiler directives:

All compiler directives are defined by using the <keyword> constant. Certain identifier that start with the (backquote) character are compiler directives.

`define, `undef, `ifdef, `ifndef

Ex:1) `define Max_Bus_Size 32

`define Result_Size (Awidth * Bwidth)

...

reg [Max_Bus_Size-1:0] add_reg.

Ex:2 `celldefine

module FD1S3AX (D, CK, Q)

input D, CK;

output Q

...

endmodule

end celldefine

(2) Difference between parallel and sequential blocks.

Sequential blocks	Parallel blocks
→ In this we use begin and end keywords.	The keywords fork and join are us
→ Statements in a sequential block execute in sequence.	statements are executed concurrently
→ The statements in a sequential blocks are processed in the order they are specified	statements in parallel blocks are executed concurrently.
→ A statement is executed only after execution of its preceding statement	ordering of statements are controlled by the delay (or) event control assigned to each statement
→ Syntax begin [block identifier] Procedural statements end	Syntax: fork [block identifier] procedural statements *) join.

(3) Difference between task and function and how can we include task.

Task.	function
→ A task can enable other task and function	A function can enable another function but not task
→ Task may execute in non-zero simulation time	Function always execute in simulation time.
Task may contain delay, event or timing control statement	Functions must not contain any delay, event or timing control statement
Task may have zero or more arguments of type input, output (or) inout.	Function must have atleast one input argument. They can have more than one input.
Tasks do not return with a value, but can pass multiple values through output and inout arguments	Function always return a single value. They cannot have output and inout arguments

Write a verilog code for inverter in switch level of modelling

```
module not (a,b,out);
input a,b;
output out;
wire c;
supply1 power;
supply0 gnd;
Pmos (c,power,b);
Pmos (out, c, a);
nmos (out, gnd, a);
nmos (out, gnd, b);
end module.
```

6) What are the modelling tips for logic synthesis.

Use meaningful names for signals and variables
Avoid mixing positive & negative edge triggered flipflop
Use basic building blocks Vs use of continous statements
Use parenthesis to optimise logic stg structure
Use arithmetic operators *, / and %.
Be careful with multiple assignment to the same variable
Define type and case statements explicitly.

6) List the uses of PLI?

PLI can be used to define additional system tasks and function
Application software like translators and delay calculators can be execution with PLI.
PLI can be used to extract design information such as hierarchy, connectivity, fanout and number of logic elements of a certain type.
PLI can be used to write special purpose or customised output display routines
Several verilog based application software can be written with PLI routine.

(7) Explain the types of UDP's?

UDP stands for user define primitives.

There are several types of UDP's

i) combinational

ii) sequential.

combinational UDP's are defined where the output is solely determined by a logical combination of the inputs. A good example is a 4:1 mux.

sequencial UDP's take the value of the current inputs and the current o/p's to determine the value of the next o/p. The value of the output is also the internal state of the UDP. Good examples of sequential UDPs are latches and flipflops

(8) Explain the parallel connection and full connection with example.

Parallel connection: A parallel connection is specified by the symbol '=>' and is used as shown below

(<source_field> => <destination_field>) = <delay_value>)

In parallel connection, each bit in source field connects to its corresponding bit in the destination field.

Eg. ~~// bit-to-bit connection specifies delay from each bit in source to each bit in~~

// bit-to-bit connection both a and out are single bit

(a => out) = a;

// vector connection, both a and out are 4-bit vector a[3:0], out[3:0]

// a is source field, out is destination field.

(a => out) = a;

// the above statement is short and notation

// for four bit-to-bit connection statements

(a[0] => out[0]) = a;

(a[1] => out[1]) = a;

(a[2] => out[2]) = a;

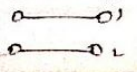
(a[3] => out[3]) = a;

// illegal connection a[4:0] is a 5 bit vector, out[3:0] is 4-bit.

// mismatch between bit width of source and destination fields

(a => out) = a; // bit width does not match.

figure of parallel connection:



Full connections

In a full connection, each bit in the source field connects to every bit in the destiny field

If source and destination are vectors then they need not have the same number of bits

Eg:

```
// full connection
module mlow, a, b, c, d);
output out;
input a, b, c, d;
wire e, f;
// full connection
specify
  (a, b *> out) = 9;
  (c, d *> out) = 11;
  end specify
and a1 (e, a, b);
and a2 (f, c, d);
and a3 (out, e, f);
end module
```

3) Write verilog code for 2:1 mux using conditional operator.

```
module 2 mux (a, b, s, y);
input a, b, s;
output y;
assign y = s ? b : a;
end module
```

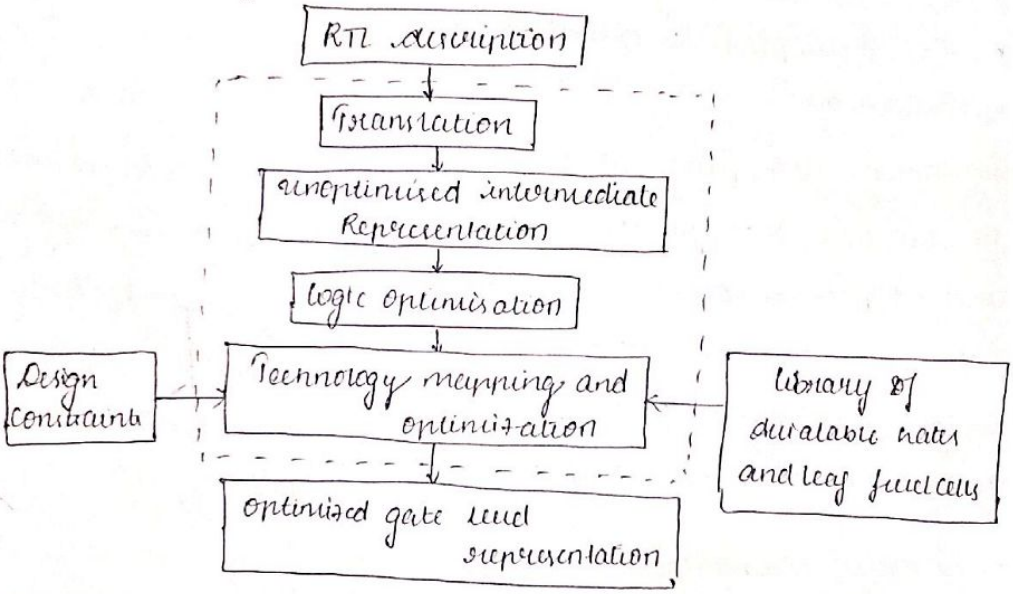
VLSI Assignment - (2013 - Main). Tasneem Nazneen
1604-11-735-067

Part-B

17. Write a short note on:

a) Synthesis design flow:

The RTL synthesis flow follows several stages from RTL description to optimized gate level optimization as shown.



i) RTL description:

The designer describes the design at a high level by using RTL constructs of the HDL language. The designer spends time in functional verification to ensure that the RTL description functions correctly. After the verification of functionality, the RTL description is input to the variable and compatible logic synthesis tool.

ii) Translation:

The RTL description is converted by the synthesis tool to an unoptimized intermediate, internal representation. This process is called translation.

The design constraints such as area, timing and power a not considered in the translation process.

iii) Unoptimized intermediate Representation:

The design is represented internally by the synthesis tool in terms of internal data structures.

iv) Logic optimization

Logic is now optimized to remove redundant logic. The techniques incorporate several algorithms and rules to convert un-optimized Boolean description to optimised one. This process is called logic optimization.

Technology Mapping and optimization:

In this step, the synthesis tool takes the internal representation and implementations the representation in gates, using the cells provided in the technology library, there by the design is mapped to the desired target technology such as CPLD, FPGA and ASIC chip technologies.

Technology library:

It contains library cells provided by the chip manufacturing company depending on technology used. The library cells are the basic building blocks that the chip manufacturing company uses the for fabrication. The library cells can be basic logic gates (or) macro cells such as adders, ALU's, multiplexers and special flip flops.

Optimized Gate Level Description:

After the completion of technology mapping, an optimized gate-level netlist described in terms of target technology

ponents is produced. If the netlist meets the required constraints is handled to the chip manufacturing company.

IC Fabrication.

The functional verification is done by identical stimulus that is run with the original RTL and synthesized gate level descriptions of the design. Then the output is compared to find any mismatches. If the output is not identical the designer needs to check any practical bugs and rerun the whole flow until all the bugs are eliminated.

b) Sequential and Parallel blocks:

Sequential blocks:

In sequential blocks we use begin and end keywords to group statements. Statements in a sequential block execute in sequence.

Characteristics of sequential block:

1) The statement in a sequential block are processed in the order they are specified.

2) A statement is executed only after execution of its preceding statement

3) If delay/ event control is specified, it is relative to the simulation time when the previous statement in the block completed execution.

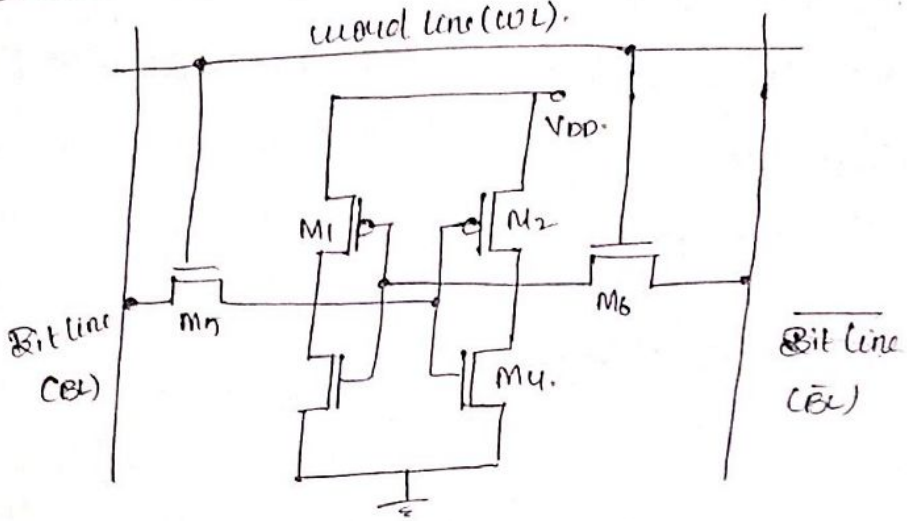
Parallel Blocks:

The keywords fork and join are used in parallel blocks. Statements in parallel blocks execute concurrently.

ordering of statement is controlled by the delay (or) event cont assigned to each statement.

If delay (or) event control is specified, it is relative to the entered ti of the block. The order in which the statements are written in the block is not important.

C) 6T SRAM cell:



A SRAM 1-bit memory cell with 6-transistors is shown above. The two CMOS inverters are connected back-to-back. Transistors M_1 and M_3 form one inverter, and transistors M_2 and M_4 form another inverter. The transistors M_5 and M_6 are used as switch controlled by word line.

Read operation: To read data from the cell, word line is enabled. This makes transistors M_5 and M_6 ON. Hence, the stored data is available in both the true and complimented form in the bit line and $\overline{\text{bit line}}$, respectively.

Write operation:

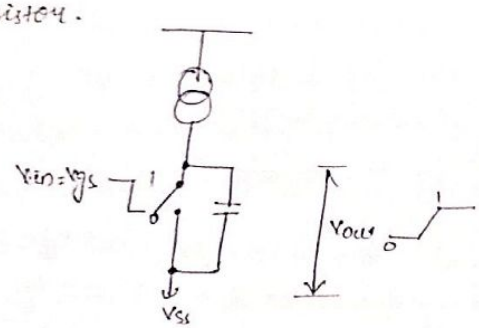
To write data into the cell, again word line is enabled. The data to be written is made available in the bit lines. The data is stored in the latch.

How to estimate the delay of cmos inverter? Write the procedure to obtain delay.

The estimation of delay associated with the cmos inverter is done by splitting the o/p transitions into fall time (T_f) and rise time (T_r) corresponding to charging and discharging of the capacitive load (C_L). The charging and discharging are estimated independently to calculate the overall delay accurately.

Rise-time estimation

The capacitive load is driven by the pull-up p-device and assume that it is to be in saturation ~~and resistive region~~ region for the entire charging period of the load capacitor (C_L). charging of C_L is divided into saturation and resistive region of transistor.



The saturation current for p-transistors is given by,

$$I_{dsp} = \frac{(V_{gs} - |V_{tp}|)^2 \beta_p}{2} \quad -(1)$$

o/p voltage when the above current is being constant is given by

$$V_{out} = \frac{t I_{dsp}}{C_L} \quad -(2)$$

Sub. eq (1) in (2)

$$t = \frac{2 V_{out} C_L}{(V_{gs} - |V_{tp}|^2)^2 \beta_p}$$

If $t = T_r$ when $V_{out} = V_{DD}$, then the corresponding T_r is given by

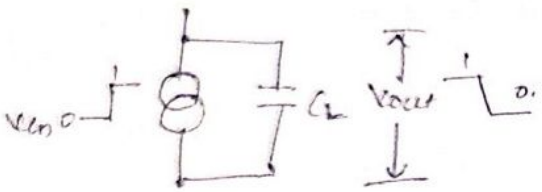
$$V_T = \frac{2C_L V_{DD}}{(V_{DD} - |V_{tp}|)^2 \beta_p}$$

But $|V_{tp}| = 0.2V_{DD}$, then T_r becomes.

$$T_r \approx \frac{3C_L}{V_{DD}\beta_p} \quad \text{— (3)}$$

Fall-time Estimation

Estimation of fall-time associated with the discharging of C_L through the pull-down n-type device.



By taking the similar process of rise-time, the fall-time becomes to

$$T_f \approx \frac{3C_L}{V_{DD}\beta_n}$$

from eq (3) and (1)

$$\frac{T_r}{T_f} \approx \frac{\beta_n}{\beta_p}$$

If $\beta_n \approx 2.5\beta_p$ then $\mu_n = 2.5\mu_p$; here μ_n and μ_p are different from each other. From this we can say that rise-time is slower by a factor of 2.5 when size of both the n and p-devices are minimum.

By putting channel length minimum, symmetrical operation can be achieved by $W_p = 2.5W_n$.

where, W_p = channel width of p-device
W_n = channel width of n-device

As per the minimum size lambda-based design rules,

Input capacitance of inverter for p-device = $2.5C_g$.

Input capacitance of inverter for n-device = $1C_g$

then, the total capacitance = $2.5C_g + 1C_g = 3C_g$.

some analytical models are used to get the optimistic results for rise time and fall time.

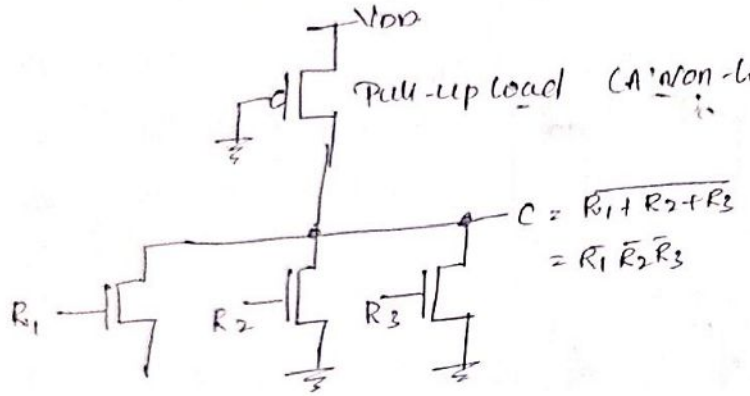
factors affecting rise time and fall time are,

T_f and T_r are proportional to C_L.

T_f and T_r are proportional to $1/V_{DD}$

$T_f = 2.5 T_f$, for n and p-transistor geometries.

16.b) Draw NOR based ROM memory. Explain its operation.



The PMOS pull-up circuitary is replaced by a single pmos with its gate tied upto GND, hence being permanently on acting as a load resistor. If none of the nmos transistors are activated (all Ri being low) then the o/p signal C is high.

If any one of the nmos transistor is activated (Ri being high) then the output signal C is low. To reduce the power consumption the gate of the PMOS pull-up transistor is connected to a clock signal. The power is consumed only during low period of the clock.

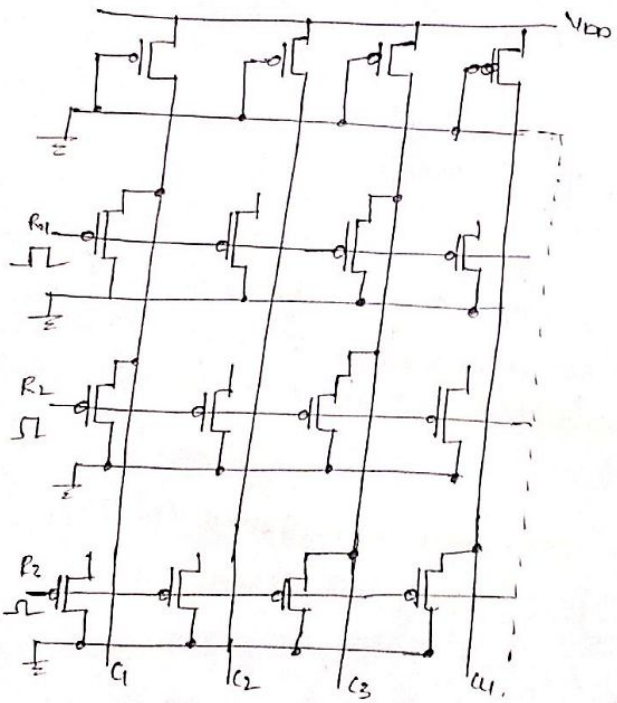
A NOR-based rom consists of m n-inputs pseudo-NMOS NOR gates, one n-input NOR per column. as shown.

Each memory cell is represented by one ~~EMOS~~ nmos transistor and binary information is stored by connected (or) not the drain terminal of such a transistor to the bit line. For every row

4

address only one word line is activated by applying a chip signal to the gate of nmos transistors in a word. If a selected transistor in the j^{th} column is connected to a bit line then the logic '0' is stored in this memory cell. If the transistor is not connected, then the logic '1' is stored.

15a) ~~with the help of~~



	C_1	C_2	C_3	C_4
R_1	0	1	0	1
R_2	0	0	1	0
R_3	1	1	0	0

15a) with the help of circuit diagram, explain 4-bit barrel shifter.

A barrel shifter is a digital logic circuit that can shift a data word by a specified number of bits in one clock cycle. It can be implemented as a sequence of multiplexers and in such an implementation the output of one multiplexer is connected to the input of next multiplexer in a way

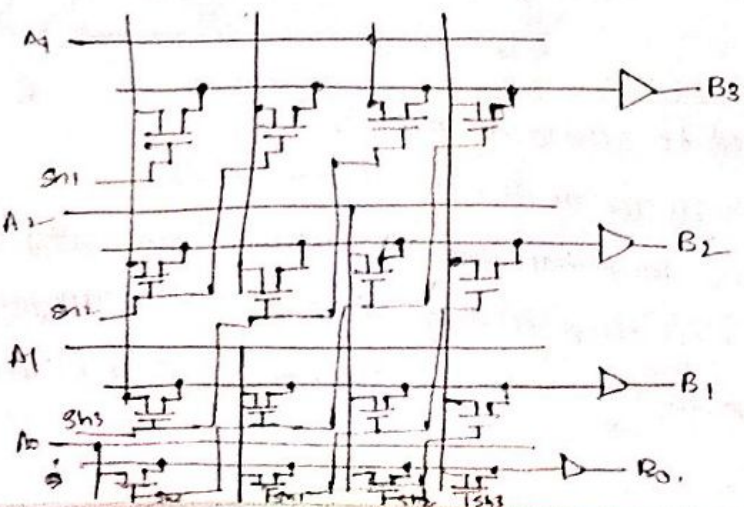
depends on the shift distance

The structure of barrel shifter is shown. It consists of an array of transistors, in which the number of rows equals the word length of the data and the number of columns equal the maximum shift width. In this case, both are set equal to four. The control wires are routed diagonally through the array.

Barrel shifter needs a control wire for every shift bit. For example, a four bit shifter needs four controls signals. To shift over three bits, the signal Sh_3 should be on the value 1000. only one of the signal is high, In a processor, the required shift value normally comes in an encoded binary format, which is substantially more compact. For instance, the encoded control would need only two control signal and as represented as 11 for a shift over three bits. To translate the latter representation into the former, an extra module called a decoder is required.

A major advantage of this shifter is that a signal has to pass through at most one transmission gate. In other words, the propogation delay is theoritically constant and independent of the shift value) for shifter size. This is not true in reality, however, because the capacitance at the input of the buffer also rises Linearly with the maximum shift width.

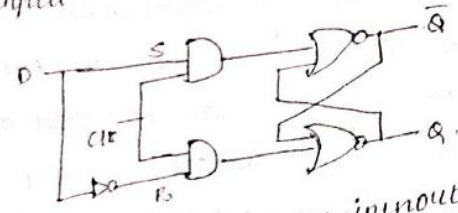
An important property of this circuit is that the layout size is not dominated by the active transistors. as in the case of all other arithmetic circuits, but by the number of wires running through the cells,

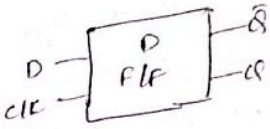


16.b) Design D-flip-flop using transmission logic gate

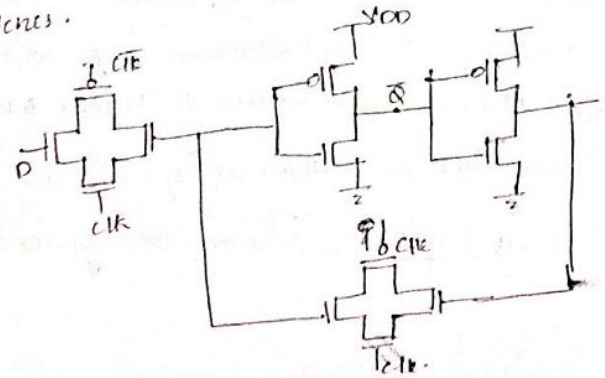
In the digital integrated circuits, a large techniques for selection of ci based sequential circuit have gained popularity,

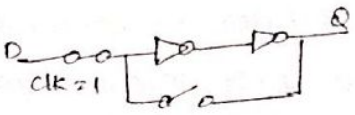
The gate level schematic of D flip-flop by modifying NOR based SR latch. The input variable D is also inverted and connected to the R input of flipflop. It is clear from the schematic that when clock is active (high) the output Q acquires the value of input D. When the clock is equal to logic low, the output will preserve its state. Here, the clock is an enable input which allows data to be accepted into the D flip-flop when active.



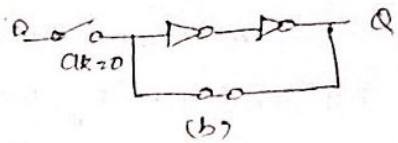


D flip-flop play many important role in digital circuits, primarily for temporary storage of data (or) as a delay element. Now we will examine the simple cmos implementation of D flip-flop as shown, which consists of two inverter loop and two cmos of transmission gate (TG) switches.

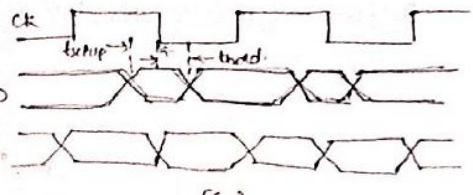




(a)



(b)



(c)

clock signal is used to activated the TGs, when clock is logic high TG at the i/p is activated, whereas the TG in the inverter loop is activated when clock is low (i.e) CLK is logic high. Thus when clock is high input D is transferred into the latch and when clock is logic low the previous state is preserved by the

flop. We can replace the TG by a simple switch to better understand the operation of D-flipflop, for both pulses of clock where I/P and O/P signals should be valid.

It should be noted that the valid input D must be stable for a short time before (setup time, t_s) and after (hold time t_h) the negative clock transition, during which the input switch opens and loop switch closes. Once the inverter loop is completed by closing the loop switch, the output will preserve its valid level. For designing the D latch, the requirements for setup time and hold time should be met. Any mistake for such specification can cause metastable problem which lead to unpredictable result.

14a) Draw and Explain nMOS fabrication process.

nMOS fabrication

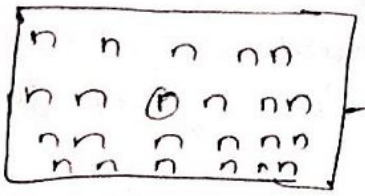
Step1: Processing is carried out on a thin slice wafer cut from a single crystal of silicon of high purity into which the required P-impurities are introduced.

wafer dimensions (typically)

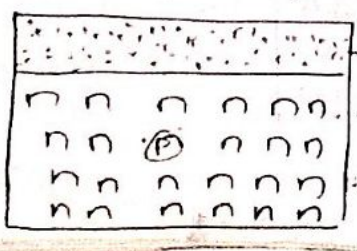
75 to 150mm — diameter

0.4mm — thickness.

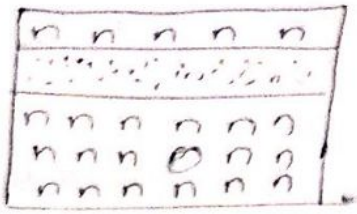
Doped with impurity concentrations of $10^{15}/cm^3$ to $10^{16}/cm^3$ given the resistivity as 25Ωcm to 2Ωcm.



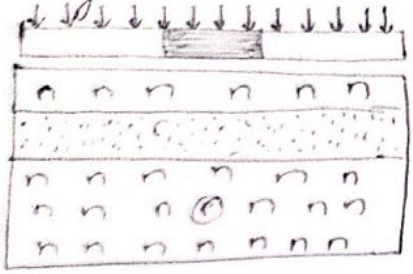
Step:2 A layer of SiO_2, typically 1μm thick is grown.



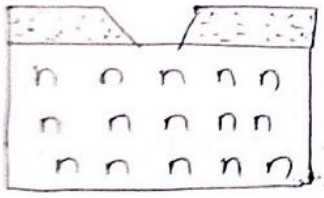
Step2: surface is now covered with a photo resistor.



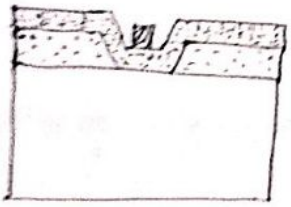
Step3: surface photo resistive layer is then exposed to UV light through a mask which defines these regions into which diffusion is to take place together with transistor channel.



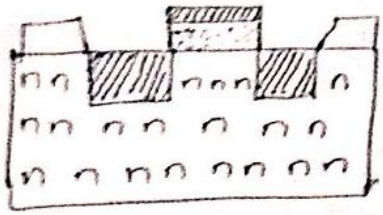
Step5: These area are subsequently readily etched away together with underlying SiO_2.



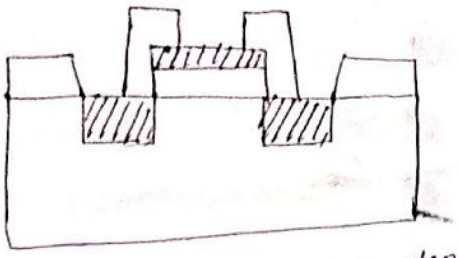
Step6: The remaining photo resistor is removed & a thick layer of SiO_2 [0.5μm typically] is grown over the chip surface & then polysilicon is deposited on top of this to form the gate structure.



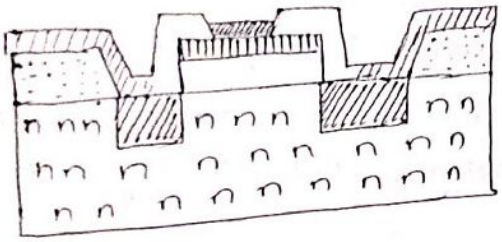
Step7: Further, photoresistive coating & masking allow the polysilicon to be patterned & then the thin oxide is removed to exposed area into which n-type impurities are to be diffused from source to design.



: silicon oxide is grown all over again & is then masked with photo-resist & etched to expose selected areas of the polysilicon gate & the drain & source areas where connections are to be made.



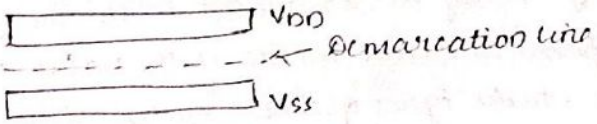
Step 9: The whole chip then has metal deposited over its surface to a thickness typically of 1μm.



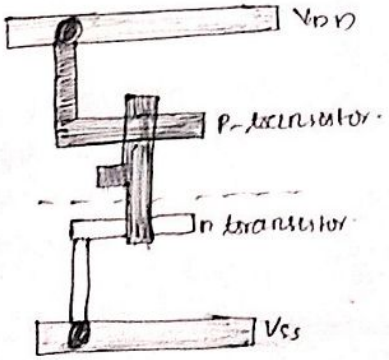
→4b) What is the difference between stick diagram and layout diagram. Draw a stick diagram of a cmos inverter

Cmos inverter $Y = \overline{A}$

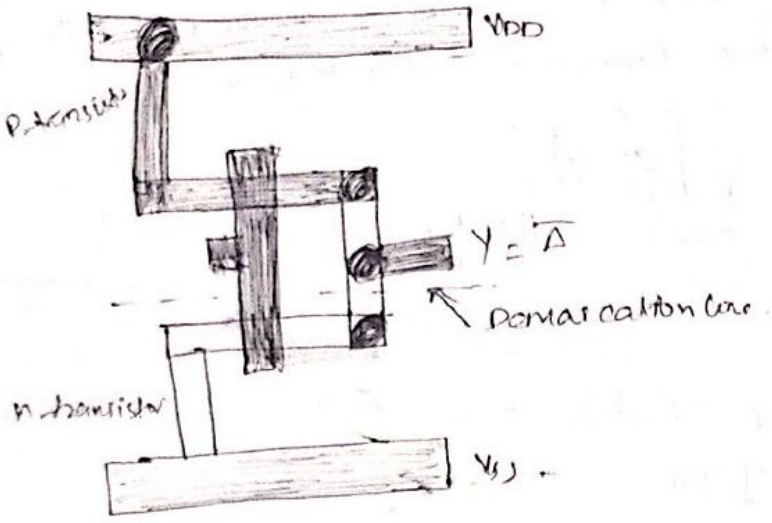
Step1:



Step2:



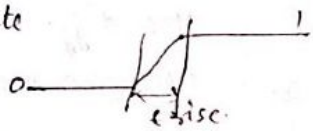
Refer P31



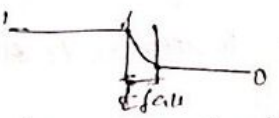
11.a) Explain gate delay with example.

verilog has the facility to account for different types of propogation delay of circuit elements. any connection can cause a delay due to the distributed nature of the resistance & capacitance. there are 3 types of delays from i/p to the o/p of primitive gate

1) Rise delay: The rise delay is associated with gate o/p transition to 1 from another value.



2) fall-delay: The fall delay is associated with gate o/p transition to a '0' from another value.



3) Turn-off delay: The turn off delay is associated with the gate o/p transition to the high impedance from another value. If the value changes to 1. the min of 3 delays is considered. These types of delay specification are allowed.

Delay example:


```
module delay (out, a, b, c);
input a, b, c;
output out;
  wire n;
# 5 and n1 (n, a, b)
# 4 or n2 (out, n, c);
end module.
```

11) b) write program for 16:1 mux using 4:1 mux in dataflow model with test bench.

```
module mux16to1 ( y, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, s0, s1, s2, s3)
```

```
,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,s0,s1,s2,s3;
y;
q,r,s,t;
ign q = ((~s0)&(s1)&a)|((~s0)&s1&b)|((~s1)&(s0)&c)|(s1)&(s0)&d;
ign r = ((~s0)&(~s1)&e)|((~s0)&s1&f)|((~s1)&(s0)&g)|(~s1)&s0&h;
assign s = ((~s0)&(~s1)&i)|((~s0)&s1&j)|((~s1)&(s0)&k)|s1&s0&d;
assign t = ((~s0)&(s1)&m)|((~s0)&s1&n)|((~s1)&s0&o)|(s1&s0&p);
assign y = ((~s2)&(~s3)&q)|((~s2)&s3&r)|((s2)&(~s3)&s)|(s2&s3&t);
end module.

testbench
Instantiate ins/ops.
module mux16to1_test;
reg a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,s0,s1,s2,s3;
wire y;
mux16to1 uut (y,a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,s0,s1,s2,s3);
initial begin
{s0,s1,s2,s3,a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p} = 0;
#5 {s0,s1,s2,s3,a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p} = 1;
=
#5 {s0,s1,s2,s3,a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p} = 16, 2^4
end
endmodule.
```

8a) Explain with help of example, diff bw fork join and begin end.

beginend:-
In sequential blocks we use begin & end keywords to group statements. stmt is a sequential block executes in sequence.

Characteristics
The stmts in a sequential block are processed in the order they are specified.
A stmt is executed only after execution of its preceding stmt.

Example:
```
manyform generation:
begin
#2 ve_stream = 1;
#5 ve_stream = 0;
#3 ve_stream = 1;
#4 ve_stream = 0;
#2 ve_stream
#5
end.
```

fork-join;-

The keywords fork & join are used in parallel blocks. Stmt in a parallel execute concurrently

characteristics

Stmt in a parallel block are executed concurrently

ordering of stmt is controlled by the delay or event control assigned to each stmt

if delay or event control is specified it is relative to the entered of the block.

example:

waveform generation

```
fork
#2 reg-stream=1;
#7 reg-stream=0;
#10 reg-stream=1;
#14 reg-stream=0;
#16 reg-stream=1;
#21 reg-stream=0;
join.
```

12b) 4-bit parallel adder using task & function

using task:

```
module adder4 (sum,co,a,b);
output [3:0] sum, co;
input [3:0] a,b;
reg co;
reg [3:0] sum;
task add4;
output [3:0];
output c;
input [3:0] x,y;
begin
{c,s} = x+y;
end
end-task
always@(a,b)
begin
add4 (sum[3:0], co, a[3:0], b[3:0]);
end
endmodule
```

function.

```
// adder (sum, co, a, b);
output [3:0] sum, co;
input [3:0] a, b;
reg [3:0] sum, a, b;
function [4:0] adder4;
input [3:0] x, y;
begin
adder4 = x + y;
end
end function
always @ (a, b)
begin
{co, sum[3:0]} = adder4 (a[3:0], b[3:0]);
end
end module
```

(3) Different types of delay module with example.

~~Delays are used to control~~

Regular assignment delay.

This method is used to assign delay value in a continous assignment statement. The delay value is specified after the keyword assign. Any change in the values of in1 or in2 will result in a delay of x time units before recomputation of the expression in1 & in2; & the result is assigned to out. If the value of in1 or in2 are changed before x time units then the result -propagation to out, then the values of in1 or in2 at the time of recomputation are considered This is called as inertial delay.

ex: assign #10 out = in1 & in2
// Delay in a continous assign.

8) Implicit continous assignment delay:

The implicit continous assignment delay is used to specify both delay & an assignment on one net.

ex:

```
//Implicit continous assignment delay
wire #10 out = in1 & in2;
// same as
wire out;
assign #10 out = in1 & in2;
```

3) Net declaration delay:

we can specify delay on a net when it is declared without writing a continous assignment on the net. if we specify delay on net out, then the value change applied to the net out is also delayed. Net declaration delay can also be used in gate level modelling.

ex: // Net delays
wire #10 out;
assign out = in1 & in2;
// the above stmt has the same effect as the following
wire out;
assign #10 out = in1 & in2;

13b) up-down counter

```
module updowncounter (R, clock, L, E, up_down, Q);
parameter n=5;
input [n-1:0] R;
input clock, L, E, up_down;
output [n-1:0] Q;
reg [n-1:0] Q;
integer direction;
always @(posedge clock)
begin
if (up_down)
direction = 1;
else
direction = -1;
if (L)
Q <= R;
else if (E)
Q <= Q + direction
end
endmodule
```