Table of Contents

Chapter 1: Binary Number Systems

Questions:

Q1) Define: (a) bit (b) nibble (c) byte (d) word

Q2) What is weighted code? Give example.

Q3) Give an example for Non-weighted code?

Q4) What is the key feature of Excess-3 code?

Q5) In how many different ways can number 5 be represented using 2-4-2-1 code?

Q6) What are all the BCD numbers that can be uniquely represented in 2-4-2-1 weighted code?

Q7) How many unused combinations are there in the representation of BCD numbers using 7-4-2-1 weighted scheme? What are they?

Q8) What is the condition for a weighted code to be self-complementary?

Q9) Convert the binary number 011101010001 to octal and hexadecimal?

Q10) Formulate a simple procedure for converting base-3 numbers to base-9?

Q11) Convert $(211101222211122)_3$ to base-9?

Q12) A number N has 'n' digits in a r-radix number system. What is its (r-1)'s complement and r's complement?

Q13) What is the 9's complement of the BCD number 752?

Q14) Convert the Gray code number 11001 to binary code?

Q15) Give the procedure for converting a binary number to Gray code?

Q16) Represent 45 in the number systems (a) binary (b) BCD (c) Excess-3 (d) Gray code

Q17) Give the range of the numbers that can be represented using n bits in 2's complement method?

Q18) What is the **minimum** number of bits required to represent the numbers in the range of -5 to 23 using 2's complement method?

Q19) What will be the result if all the zeros are retained from LSB side until 1 is seen, including that 1, and complement all the following bits of a binary number?

Q20) In a particular design which uses 5 bits for integral part and 7 bits for fractional part, the result of some operation is 7B8 hex. Find the corresponding decimal equivalent?

Q21) Convert 0.95 to its binary equivalent?

Q22) $AB_{16} - 3E_{16} = ?$

Q23) Solve for x: $(70)_8 + (122)_6 = (211)_x$?

Q24) Solve for X in $(135)_{12} = (X)_8 + (78)_9$?

Q25) Explain the detailed procedure for BCD addition?

Q26) Perform BCD addition: (1001) + (0110)?

Answers:

A1) (a) Bit: Binary digit (Either logic-1 or logic-0)
(b) Nibble: 4-bits together is called a nibble
(c) Byte: 8 bits or 2 nibbles
(d) Word: 16 bits or 2 bytes

A2) The weighted code will have a fixed weight for each position. For example, in normal binary system, the decimal equivalent can be obtained by multiplying the position value with position weight and adding them together.

A3) Unlike weighted code, non-weighted codes will not any weights. For example, Excess-3 code and Gray code. So the numbers that are represented using non-weighted code can not be directly converted to decimal equivalents.

A4) Self-complementing: The 9's complement of an excess 3 number can be obtained simply by replacing its 1's with 0's and 0's with 1's.

A5) 2-4-2-1 represents the weights corresponding to bit positions. So the two possible ways are: 1011, 0101

A6) 0, 1, 8 and 9 (Only these 4 numbers will have unique representations).

A7) As BCD numbers range from 0 to 9, there are 5 unused combinations in 7-4-2-1 code. They are: 1011, 1100, 1101, 1110 and 1111.

A8) A weighted code is self-complimentary if the sum of the weights equals to 9.
E.g.: 2-4-2-1 code. Sum of the weights = 2+4+2+1 = 9.

A9) (a) Binary: 011 101 010 001
Octal: 3 5 2 1
(b) Binary: 0111 0101 0001
Hex: 7 5 1 (Similar concept was explained in A14 & A15)

A10) Consider $(X_{n-1}\ X_{n-2}\ \ldots\ldots X_3\ X_2\ X_1\ X_0)$ a n bit base 3 number.
The corresponding decimal equivalent is given by,

$3^{n-1} X_{n-1} + 3^{n-2} X_{n-2} \ldots\ldots + 3^3 X_3 + 3^2 X_2 + 3^1 X_1 + 3^0 X_0$

$= 9^{(n-2)/2} (3 X_{n-1} + X_{n-2}) + \ldots\ldots\ldots\ldots + 9^1 (3 X_3 + X_2) + 9^0 (3 X_1 + X_0)$

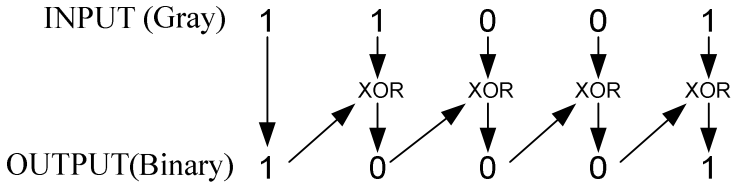
So take every to digits of base-3 number from LSB side, find their decimal equivalent, it will be the corresponding base-9 digit. (Similar to the procedure of converting a binary number to octal or hex)

A11) Base-3: $(2\ 11\ 10\ 12\ 22\ 21\ 11\ 22)_3$
Base-9: 2 4 3 5 8 7 4 8

A12) $(r^n - 1)$-N , $(r^n - N)$

A13) 999 – 752 = 247

A14) Conversion from gray to binary: Retain the MSB as it as. XOR the current input bit with the previous output bit to get the new output bit. In this case, given gray code number is 11001



So, the required binary number is 10001.

A15) Binary to Gray code conversion: Retain MSB. XOR Current bit of Binary input with the previous bit of Binary input to get new bit of Gray code Output.

A16) (a) 101101
(b) To get BCD: Represent each digit separately in binary 0100 0101
(c) Excess-3: Add 3 to each digit and then represent them separately in binary
0111 1000
(d) Gray code: First convert to Binary and use the procedure shown in Q15:
111011

A17) $-(2^{n-1})$ to $+(2^{n-1} - 1)$

A18) $(2^{n-1} - 1) > 23 \Rightarrow 2^{n-1} > 24 \Rightarrow 2^{n-1} = 32 \Rightarrow n-1 = 5 \Rightarrow \underline{n = 6 \text{ bits}}$

A19) 2's compliment of the binary number
E.g.: Consider 10010, Its 2's compliment is given by 01110

A20) 78B in hex = 01111.0111000 (5 integral bits and 7 fractional bits)
= 15.4375

A21) 0.95 x 2 = 1.90 ---- 1
0.90 x 2 = 1.80 ---- 1
0.80 x 2 = 1.60 ---- 1
0.60 x 2 = 1.20 ---- 1
0.20 x 2 = 0.40 ---- 0
0.40 x 2 = 0.80 ---- 0
0.80 x 2 = 1.60 ---- 1 …….
So, 0.95 = 0. 11 1100 1100 1100 1100....

A22) $AB_{16} - 3E_{16} = (171) - (62)$
$= (109) = 6D_{16}$

A23) $2x^2 + x + 1 = 56 + 50 = 106 \Rightarrow x(2x+1) = 105 = 7 \times 15 \Rightarrow x = 7$

A24) $(78)_9 = 63 + 8 = (71)_{10}$
$(135)_{12} = 144 + 36 + 5 = (181)_{10}$
$(71)_{10} - (181)_{10} = (110)_{10} = (156)_8$

So, X = 156

A25) BCD addition is similar to any binary addition. But if the result is above 9, to get valid BCD result, we need to add 6 to the result.

A26)
```
    1 0 0 1
    0 1 1 0
  ----------
    1 1 1 1 (>9)
Adding 6, 0 1 1 0
  ----------
  1 0 1 0 1
```

So, the result is 0001 0101 = (1 5)

Chapter 2: Basic Gates and Boolean Algebra

Questions:

Q1) Which gates are called universal gates? Why?

Q2) How many minterms or maxterms will be there for n-inputs?

Q3) Give the minterm and maxterms corresponding to 6 and 15 numbers (4-inputs)?

Q4) In how many ways can a NAND gate be converted into an inverter? Show all the possibilities?

Q5) How many number of 2 input AND gates are required to generate N I/P AND gate?

Q6) State De-Morgan's Laws?

Q7) (a) If it is given that A & B will not be 1 at the same time, what will be the equivalent logical gate for an XOR gate?

(b) If any of the inputs of an XOR gate are inverted, XOR gate will work as ----- ?

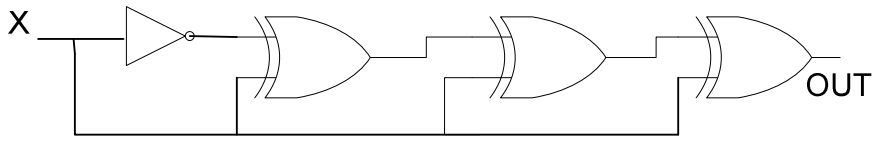
Q8) State the Shannon's expansion theorem for representing a Boolean function by its co-factors?

Q9) Write the cofactors F_A and $F_{A'}$ for F(A,B,C,D) = ABD + BCD' + A'B'C' ?

Q10) How many unique Boolean functions can exist for 'n' number of inputs?

Q11) Mention the logical gates for which the 3 input implementation can not be obtained from two 2 input gates? How do you implement them?

Q12) What is OUT in the circuit shown below?



Q13) Give implementation of XOR using minimum number of NAND gates?

Q14) An assembly line has 3 fail safe sensors and one emergency shutdown switch. The line should keep moving unless any of the following conditions arise:

(i) If the emergency switch is pressed

(ii) If the senor1 and sensor2 are activated at the same time.

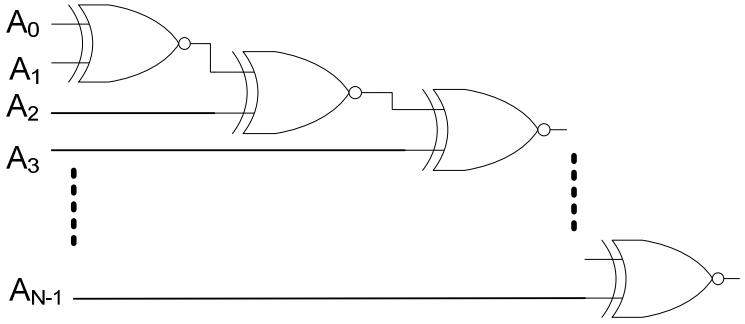
(iii) If sensor 2 and sensor3 are activated at the same time.

(iv) If all the sensors are activated at the same time

Suppose a combinational circuit for above case is to be implemented only with NAND Gates. How many minimum number of 2 input NAND gates are required?

Q15) Majority function is the one which gives 1 if the input has more 1s than 0s. Show the truth table and give the AOI for 3-input majority function?

Q16) N number of XNOR gates are connected as shown below. How does this circuit work? Explain?



Q17) Show the implementation of XNOR gate using minimum number of NOR Gates?

Q18) Explain parity generation and its significance?

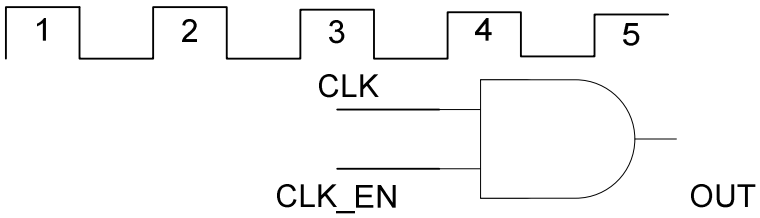
Q19) Which logical gates can be used as parity generators?

Q20) What is the parity of (i) 10111001 (ii) 11001010

Q21) Give a circuit for 4-bit even parity checker? And explain the same how can it be re-used for parity generation?

Q22) Design a combinational circuit using XOR gates that converts a 4-bit gray code number to a 4-bit binary number?

Q23) Draw the enable signal (CLK_EN) such that the OUT will get only the 2 and 3 pulses of CLK? The figure shows the circuit and CLK signal?



Q24) Which logical gate can be used to find out whether the two single bit inputs are equal or not?

Q25) What is the difference between NAND gate and negative AND gate?

Q26) How to obtain the dual of a Boolean equation?

Q27) Match the following:

a) Comparator	(i) NAND
b) Half adder	(ii) NOR
c) Anyone input is 1, output is 0	(iii) XOR
d) Anyone input is 0, output is 1	(iv) XNOR

Answers:

A1) NAND and NOR gates are called universal gates. Because any other logical gate like AND, OR, NOT, XOR, XNOR etc. or any other Boolean function can be implemented only with NAND or NOR gates.

A2) For n inputs, possible minterms/maxterms = 2^n. For example, for 2 inputs the possible 4 minterms are A'B', A'B, AB', AB and maxterms are A+B, A'+B, A+B', A'+B'.

A3) (a) 6 = $(0110)_2$ Minterm = A'BCD', maxterm = A+B'+C'+D
(b) 15 = $(1111)_2$ Minterm = ABCD, maxterm = A'+B'+C'+D'

A4) A NAND gate can be converted into an inverter by using any of the following two methods:

A5) N-1. For example to implement a 4 input AND gate we need three 2-input AND gates.

A6) (A+B+C+D…)' = A'.B'.C'.D'…….
(ABCD……..)' = A' + B' + C' + D'……

A7) (a) OR Gate.
We can conclude this from truth table. Also from Boolean algebra as shown here :
As A=B=1, can not occur, AB = 0 always.
A XOR B = AB' + A'B = A (AB)' + B (AB)' = A (0)' + B(0)' = A + B

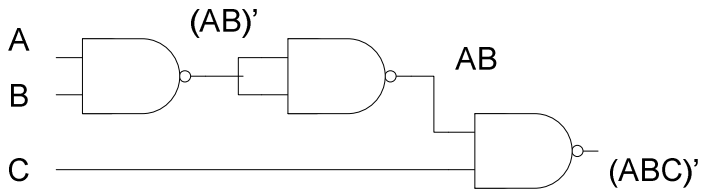
(b) XNOR Gate.
A XOR B = AB' + A'B
Using this, A' XOR B = AXOR B' = A'B' + AB = A XNOR B

A8) According to Shannon's expansion theorem any Boolean function F(A,B,C,D….) can be represented as $F = A F_A + A' F_{A'}$, where the cofactors F_A and $F_{A'}$ are given as, $F_A = F(1,B,C,D….)$ and $F_{A'} = F(0,B,C,D….)$

A9) F_A = BD + BCD' and $F_{A'}$ = BCD' + B'C'

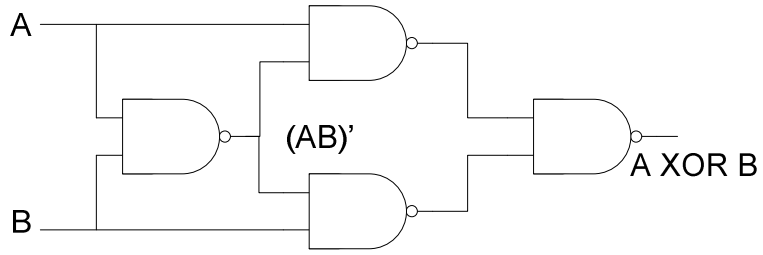
A10) 2^2^n
For n inputs, the possible number of minterms are, k = 2^n.
Any boolean function is combination of minterms. So all possible Boolean functions are $^kC_0 + {}^kC_1 + {}^kC_2 + {}^kC_3 + \ldots\ldots {}^kC_k = (1 + 1)^k = 2^k = 2^{2^n}$

A11) NAND , NOR and XNOR : All these 3 gates need three 2-input gates to get 3-input gates. The 3-input NAND implementation using 2-input NAND gates is shown below. NOR and XNOR also can be implemented in the same way.



A12) First XOR gate output = X XOR X' = 1
Second XOR output = 1 XOR X = X'
Third XOR gate output = OUT = X' XOR X = 1
OUT = 1 irrespective of X

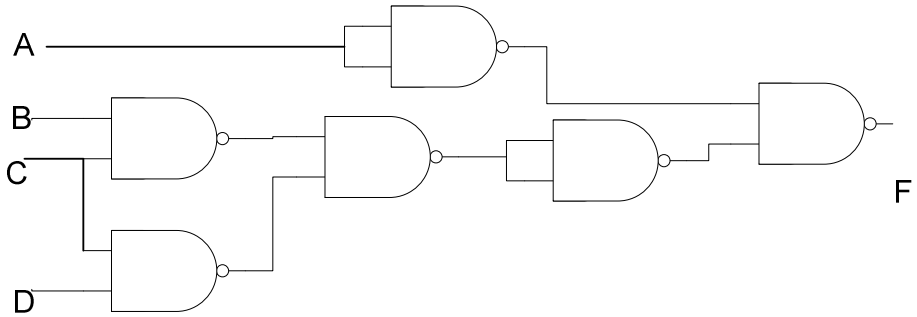
A13) A XOR B = A'B + AB' = A(AB)' + B(AB)'



A14) 6

A = Switch B=Sensor1 C=Sensor2 D=Sensor3 Pressed or sensor activated = 1
F=Shutdown=1

If you use K-Map and simplify, you will get F = A + BC + CD. The implementation of the same is shown below.



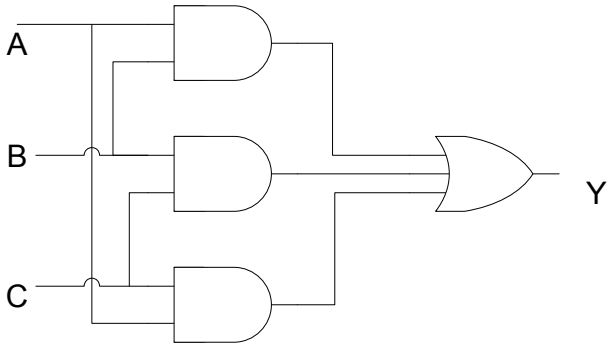
You can always use the truth table and then use Boolean simplification to get the result.

A15) Truth table for 3-input majority function is shown below:

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

Y = AB + BC + AC

The AND-OR Implementation (AOI) of Y is shown below:

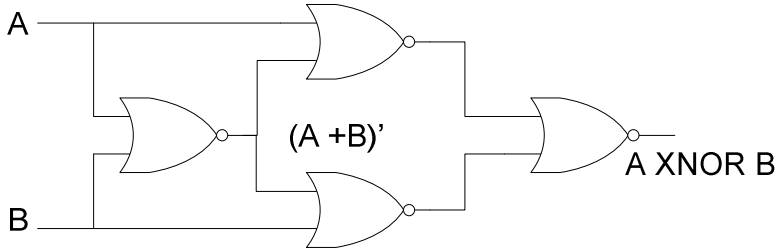


A16) This is circuit will work in two different ways based on N-value. (a) N is odd (b) N is even

(a) If N is odd, there will be even number of XNOR gates in the circuit. Take an example of N=3, So there are 2 XNOR Gates. The two bubbles will get cancelled and this works as XOR. Same works for any odd N. So if N is odd it works as XOR Gate.

(b) If N is even, the circuit works as XNOR Gate. (Apply the same logic). One extra bubble will be there to make XOR to XNOR. You may verify the same for N=4.

A17) Very much similar to A13.



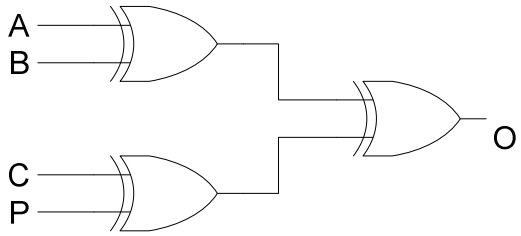
A18) Parity generation adds an extra bit to the data which indicates the parity of input data. Parity generation is of two types: Even-parity and odd-parity generation. Even parity generator gives 1 if the input has odd number of 1's so that overall number of 1's will be even. Similarly odd parity generator gives 1 if the input has even number of 1's.

In data transmission systems the transmission channel itself is a source of data error. Hence the need to determine the validity of transmitted and received data. The validity of data is essential in applications where data is transmitted over long distances. Invalid data is a corruption risk. Parity generation helps in checking the validity of the data. Parity generation and validation is a scheme to provide single bit error detection capabilities.

A19) XOR gate can be used as even parity generator and XNOR can be used as odd parity generator.

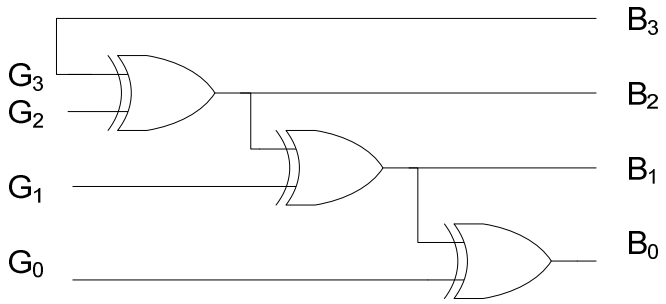
A20) (i) ODD as the number of ones = 5, odd number (ii) EVEN, number of 1s =4,even

A21) The following circuit shows a parity checker for 4 inputs. A, B and C is the actual data. Whereas P is even parity bit generated at the transmitter. P = A xor B xor C. So A, B, C and P together will have even parity always. If all the bit sequences are received properly, O should be zero always. O=1 indicates that some error has occurred during transmission.

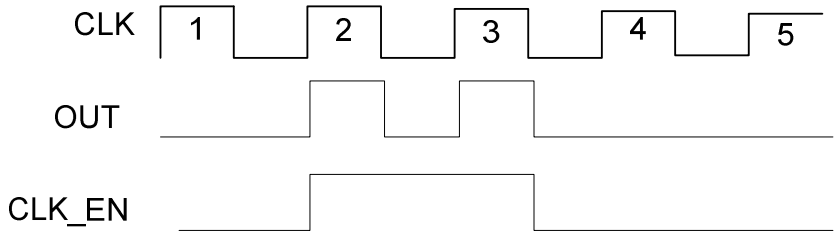


The same circuit can be used for parity generation by putting P = 0. If P=0, the same circuit works as 3-input even parity generator.

A22) The detailed procedure with an example for converting gray code to binary is shown in chapter 1. The same concept is shown with the XOR gates here.



A23) The input clock, the OUT that is needed and the corresponding CLK_EN are shown in the following diagram:

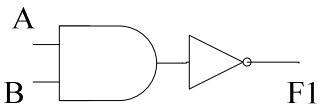


A24) XNOR gate.

If we observe the truth tables, XNOR gate gives 1 if both the inputs are same. Similarly XOR gives 1 if both the signals are different.

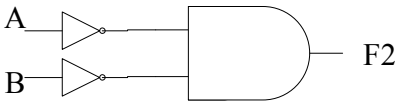
A25)

(a) NAND gate :



F1 = (AB)’ = A’ + B’

(b) Negative AND gate:



F2 = A’.B’ = (A + B)’ (Equivalent of NOR Gate)

A26) Dual : Replacing AND (NAND) with OR(NOR) and OR (NOR) with AND(NAND) in a given boolean equation gives the dual.

A27)

a) iv
b) iii
c) ii
d) i

Chapter 3: K-Maps

Questions:

Q1) Define: SOP from and POS form?

Q2) When is a SOP/POS form is called standard or canonical?

Q3) Write the POS from for a 3-input XNOR gate? Is it canonical?

Q4) Which form is suitable for designing logic circuits using
(a) Only NAND gates
(b) Only NOR gates

Q5) In which order are the bits arranged while drawing K-Maps?

Q6) Why do we write 00 01 11 10 in that order while Drawing K-maps?

Q7) How many cells will a n-input variable have in K-Map?

Q8) How many dimensions (without projections) are there for n karnaugh map (n>2)?

Q9) What do you mean by don't care condition?

Q10) Y = A'C + AC'B' and you are given that A=C=1 will never occur. Simplify Y?

Q11) Y = $\sum$ (0,2,3,4,9,10,12,13)
= d (6,8,14) .Simplify using KMap. Mention Prime Implicants & Essential Prime Implicants?

Q12) Y = F(A,B,C,D) = $\sum$ (0,1,4,5,7,9,12). Express the same using Π ?

Q13) If F(A,B,C,D,E) = B'E, how many terms will be there in the standard or canonical SOP representation of F?

Q14) In a 6 variable K-map, how many literals will the grouping of 4 adjacent cells give in the term?

Q15) Generalization of Q13: The grouping of k adjacent cells, in a N variable K-Map will lead to a term of ----- literals?

Q16) If the number of variables are more,(>5) , which method is suitable for Boolean simplification?

Q17) In the simplification of a Boolean function, F = $\sum$ (0,1,2,6,7,8,9,10,14,15) using Q-M method the following table is obtained:

	0	1	2	6	7	8	9	10	14	15
BC				X	X				X	X
CD'			X	X				X	X	
B'C'	X	X				X	X			
B'D'	X		X			X		X		

(a) Identify the essential prime Implicants?
(b) Find out the simplified expression for F in SOP form?

Q18) Use K-Map to simplify F = ∑ (0,1,2,6,7,8,9,10,14,15) in SOP from? Cross check the essential prime Implicants that are obtained in Q17.

Q19) Simplify the Boolean function Y = ∑ (0,2,3,5,7,10,11,15) in POS form using K-Map?

Q20) Give the AND-OR implementation of a circuit, using minimum gates, that gives HIGH when the input is BCD equivalent of 5,7 or 9 and LOW otherwise.

Answers:

A1) SOP: Sum Of Products : OR of all ANDs Eg: F (A,B,C) = A + BC
POS: Product Of sums: AND of all ORs Eg: F(A,B,C) = (A'+B') (A'+C')

A2) A SOP is called standard if each term is a minterm. Similarly a POS is called standard if each term is a maxterm.

A3) Y = A XNOR B XNOR C

A	**B**	**C**	**Y**
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Y = (A'B'C + A'BC' + AB'C' + ABC)'
So Y is POS form is (A+B+C') (A+B'+C) (A'+B+C) (A'+B'+C')

As each term in the POS form is maxterm, this is canonical.

A4) (a) SOP form (b) POS form

A5) Hamming order (Gray code)

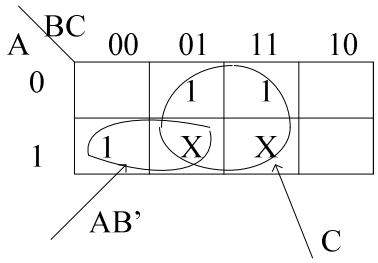
A6) In K-Map, the Boolean simplification is done by grouping the adjacent cells that have 1. To get the simplified expression, the adjacent cells must have 1 bit change. So gray code is used.

A7) 2^n. E.g.: 3 variables, 8 cells. Similarly..4 variables 16 cells.

A8) Ceiling ($\log_2 n$)

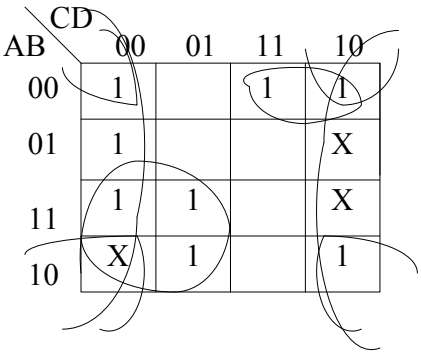
A9) The don't care condition set accommodates input patterns that never occur or outputs that will not be observed.

A10) Y = A'C + AC'B' and the output will be don't care for A = C = 1. So the K-map will be as follows:



Thus the simplified expression for Y is AB' + C

A11) The K-Map is :



The prime implicants are : D', B'D', AC', A'B'C
The essential prime implicants are : D', AC', A'B'C
The simplified expression is : Y = D' + AC'+ A'B'C

A12) Y = Π (2,3,6,8,10,11,13,14,15)

A13) 8 terms, F = B'E (A + A') (C+C') (D+D')

A14) $6 - \log_2 4 = 6-2 = 4$

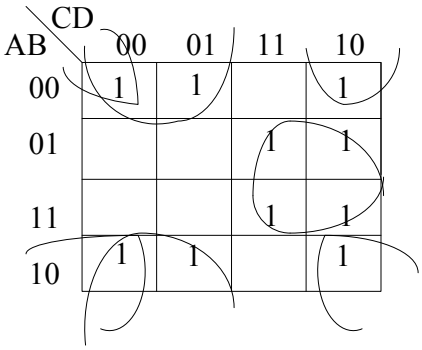
A15) In 3 variable map, grouping all 8 cells will give zero literals in the term as it is logical 1 always. Similarly, in 4 variable map the same grouping will give 1 literals, in 5 a variable map it is 2 and so on..
So the literals in the term = $N - \log_2 k$

A16) Q-M Method

A17) Checkout for the columns which has only one entry (X), that term must be included in the simplified expression. So, that term will be essential.

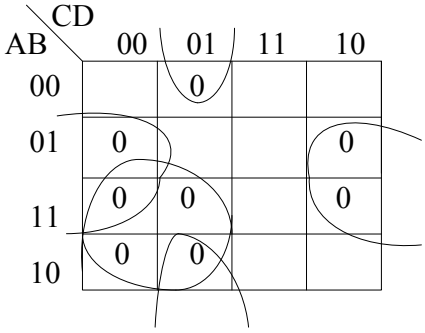
(a) So the essential prime implicants are: BC and B'C'
(b) The simplified expression F = BC + B'C' + CD' = BC + B'C' + B'D'

A18) The K-Map is shown below:



The EPI = BC and B'C'
F = BC + B'C' + CD' = BC + B'C' + B'D'

A19) To simplify the Boolean function in POS form we need to map for 0s and them take the compliment of that function to get Y in POS form.

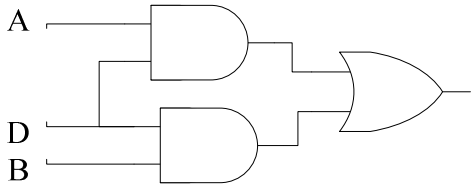


Y = (AC' + BD' + B'C'D)' = (A'+C) (B'+D) (B+C+D')

A20) As the input is a BCD number, the output will be don't care for the input combinations, 10,11,12,13,14 and 15. So the K-Map will be as shown below:

AB \ CD	00	01	11	10
00				
01		1	1	
11	X	X	X	X
10		1	X	X

The simplified Boolean function = Y = AD + BD



Chapter 4: Combinational logic

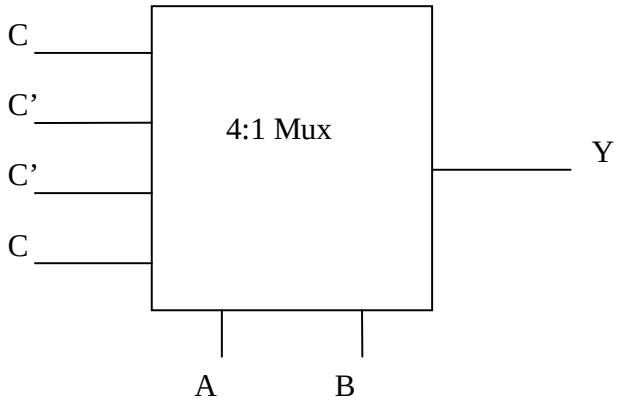
Questions:

Q1) (a) Show the AOI implementation of a 2:1 Mux?
(b) Convert this to 2-input NAND implementation?

Q2) Design the following gates using minimum number of 2:1 Muxes?
(a) NOT (b) AND (c) OR (d) XOR

Q3) Construct a 16:1 Mux with two 8:1 Mux and one 2:1 Mux.

Q4) Find out the simplified expression for Y in terms of A, B and C?



Q5) Design a circuit for 3-input majority function using a 4:1 Mux?

Q6) (a) Expand the Boolean function f(x,y,z) = x'z' + xy + xz in terms of x?
(b) Implement f using a 2:1 Mux and external gates?

Q7) There is a single telephone which needs to transmit the data from 8 different users to the receiving end. Give a design which can accomplish this task?

Q8) You are given a 2:4 decoder, a 2 input OR gate and a 3 input OR gate. Using these components design a system which takes A & B as inputs and generates the following four outputs: AB, (AB)' , A+B, (A+B)'.

Q9) Give the truth table for (a) half-adder and (b) half-subtractor?

Q10) Design a circuit for half-subtractor using basic gates?

Q11) Design "OR" gate using HA's?

Q12) Design a full adder using half-adders and minimum number of external gates?

Q13) Implement a full adder using two 4:1 Muxes?

Q14) A full adder can be implemented using basic gates in many ways. Show the efficient implementation among them, which needs minimal hardware?

Q15) Implement a circuit for adding two 4-bit numbers using (a) Ripple carry adder
(b) Carry Look Ahead (CLA) adder

Q16) Compare the two implementations of Q15.

Q17) If each XOR gate has a propagation delay of 10ns and AND/OR gate has a delay of 5ns each (irrespective of number of inputs), what is the total propagation delay in adding two 4 bit numbers in case of (a) Normal full adder (b) Carry Look Ahead adder.

Q18) Explain how a full adder can be used as majority function?

Q19) Give the truth table of full subtractor? Design the same using full adder?

Q20) There is a sixteen bit adder with ripple carry. Which of the following gives minimum delay for the output? (Fastest output)
F0 + F1, FF + FF, FF + F1

Q21) What is overflow? Under what conditions will it occur?

Q22) Using a 4-bit binary adder, design a circuit which multiplies the input by 3?

Q23) Design a subtractor unit using a 4-bit comparator, 4-bit binary adder and some external gates, which performs A-B if A>B and else B-A. A and B are two 4-bit binary numbers.

Q24) Design an adder/subtractor unit using 4-bit binary adder and some external gates, which gives out A+B if C=0 and A-B if C=1. Also provide an indicator for checking the overflow?

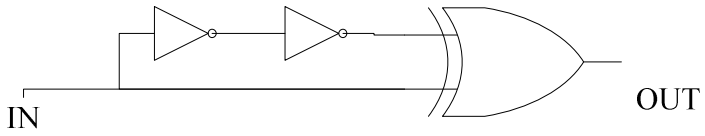
Q25) Use the above designed circuit as block box and give a scheme for finding the absolute value of a 4-bit number?

Q26) Design a circuit that generates 9’s compliment of a BCD digit using binary adder?

Q27) Give the circuit that adds two BCD numbers and gives out a BCD number?

Q28) How will you count the number of 1's that are present in a given 3-bit input using full adder?

Q29)



In the above circuit, the inverters have delay of T1 and T2 respectively. IN is a clock signal with 50% duty cycle and period T. It is given that T1+T2 is less than T/2.

(a) What is the functionality of the circuit shown above?
(b) Derive the duty cycle of output waveform?
(c) What is the condition to get 50% duty cycle at the output also?

Q30) Give the truth table for 4:2 priority encoder in which the LSB(D0) has the highest priority and MSB(D3) has the lowest priority.

Q31) You have three delay elements D1, D2, D3 that delay a clock by 25%,50% and 75% respectively. Design a frequency doubling (fout = 2 * fin) circuit that uses these delay elements along with any combinational logic.

Q32) Give a combinational circuit which checks out whether two 4-bit input signals are same or not?

Q33) Using a 4:16 decoder and **minimum** number of external gates implement the following Boolean functions:

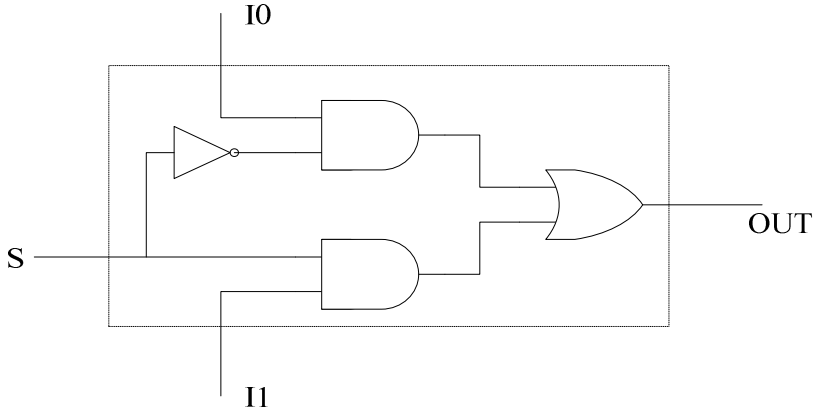
(a) $F(A,B,C,D) = \sum (5,7,9,14)$
(b) $G(A,B,C,D) = \sum (0,1,2,3,4,6,7,8,9,10,11,14,15)$

Answers:

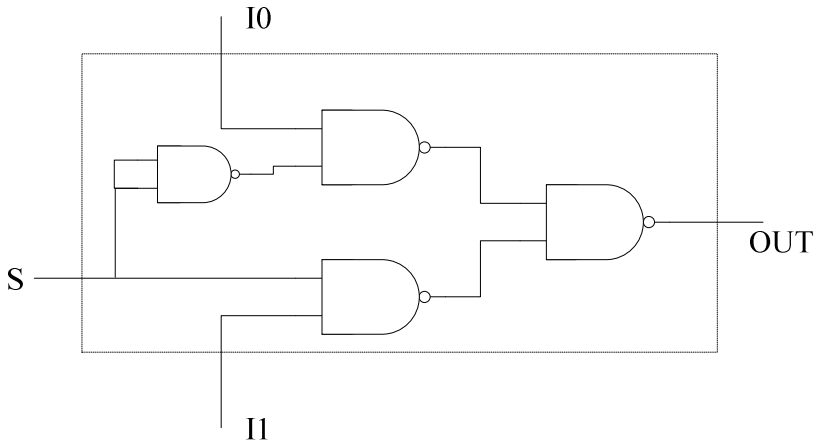
A1) (a) For a 2:1 mux, whose inputs are I0,I1 and select line S, the out put is given by the following boolean expression:

$$Out = S' I0 + S I1$$

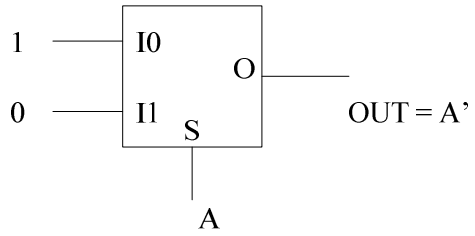
The AND-OR Implementation (AOI) is shown below:



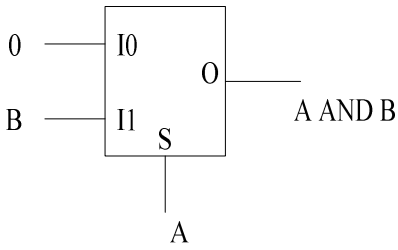
(b) The NAND gate implementation is:



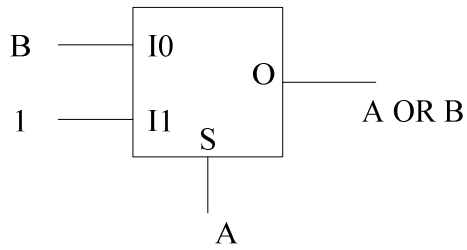
A2) (a) NOT Gate using a 2:1 Mux:



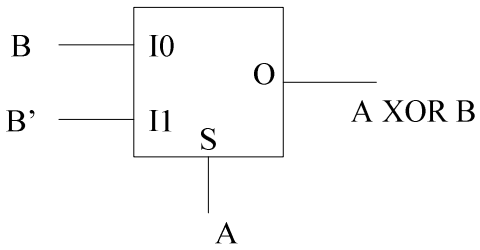
(b) AND gate



(c) OR gate

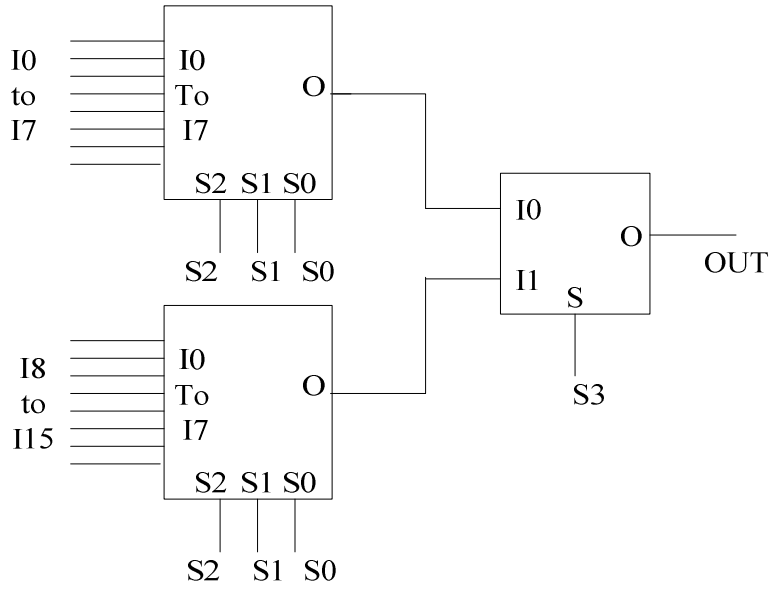


(d) XOR gate



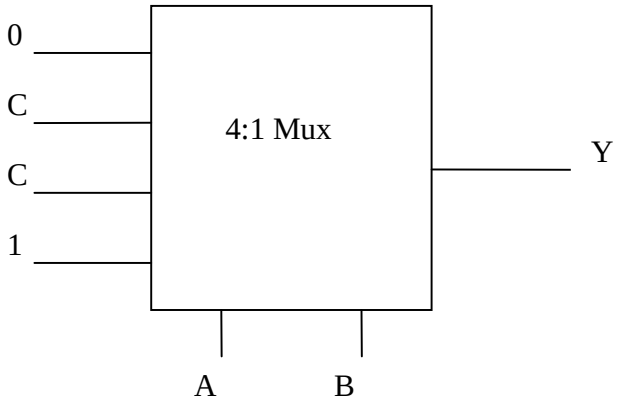
To get B', we need an extra 2:1 Mux, as inverter. Similarly NAND, NOR, XNOR gates can also be implemented. All these implementations need 2 2:1 Muxes (similarly to XOR gate).

A3) 16:1, mux will have 4(S3 to S0) select lines and 16 (I15 to I0) data lines. S3 will be 0 for I0 to I7 and will be 1 for I8 to I15. So S3 is used as a select line for the 2:1 Mux. The complete design is shown below:



A4) Y = A'B'C + A'BC' + AB'C' + ABC = A XOR B XOR C

A5)



The majority function gives 1 if the input has more number of 1s than zeros. The truth table is shown in Chapter 3. If A=B=0, the output will be zero irrespective of C. If A=B=1, output is 1 irrespective of C. But if A=1,B=0 or A=0,B=1, as we can not decide the majority without knowing C. So I0 = 0, I1=I2 = C and I3 =1. The implementation is shown above. If we simplify the expression it gives,

Y = AB+ BC + AC

A6) Shannon's expansion theorem is highly useful in implementing a Boolean function using Multiplexer.

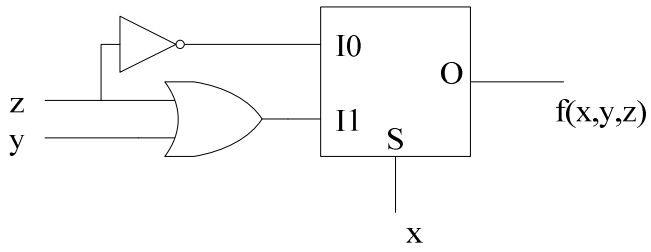
(a) To get f(x,y,z) = x'z' + xy + xz in terms of x, first use Shannon's expansion theorem to get the following co-factors. The cofactors are:

f_x = f(1,y,z) = y + z

$f_{x'} = f(0,y,z) = z'$

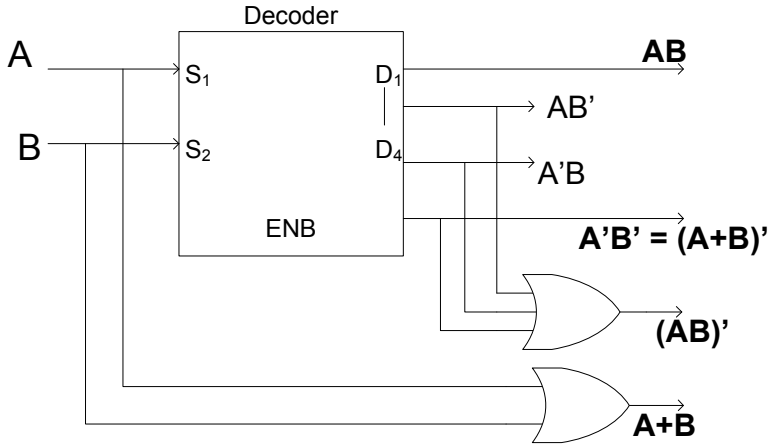
So, $f(x,y,z) = x\, f_x + x'\, f_{x'}$

(b) Now we can use x as select line and implement f(x,y,z) using 2:1 mux



A7) We need a 8:1 Mux at the input side and 1:8 Demux at the receiver side. We may need an 8 bit counter which runs at the same clock speed on both the sides to select one of the user.

A8) A 2:4 decoder will have 4 O/Ps which are the minterms of the 2 inputs : AB, AB', A'B, A'B'. Out of the four outputs that are needed, AB and (A+B)' = A'B' are directly available. Whereas A+B can be obtained using the extra 2 input OR gate (which is given). So only O/P that is needed is (AB)'.
(AB)' = A' + B' = A'(B+B') + B'(A+A') = A'B + AB' + A'B'.
So use 3-input OR gate to obtain (AB)'. The whole design is shown below.



A9) (a) Truth table for Half-adder:

A	B	Cout	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

(b) Truth table for Half-Subtractor:

A	B	Bout	Diff
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

A10) From the above truth table (A9(b)), we can derive the following equations for barrow and difference:

Borrow = Bout = A'B

Difference = Diff = A XOR B

The same equations can be shown using basic gates. (XOR, NOT, AND)

A11) The HA equations are, Cout = AB and Sum = A XOR B = AB' + A'B

Sum XOR Cout = Sum' Cout + Cout' Sum

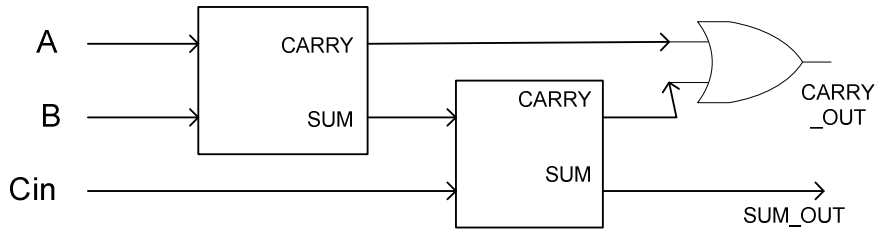
= (AB+A'B') AB + (A'+B') (A'B+AB')

= AB + A'B + AB' = A + B

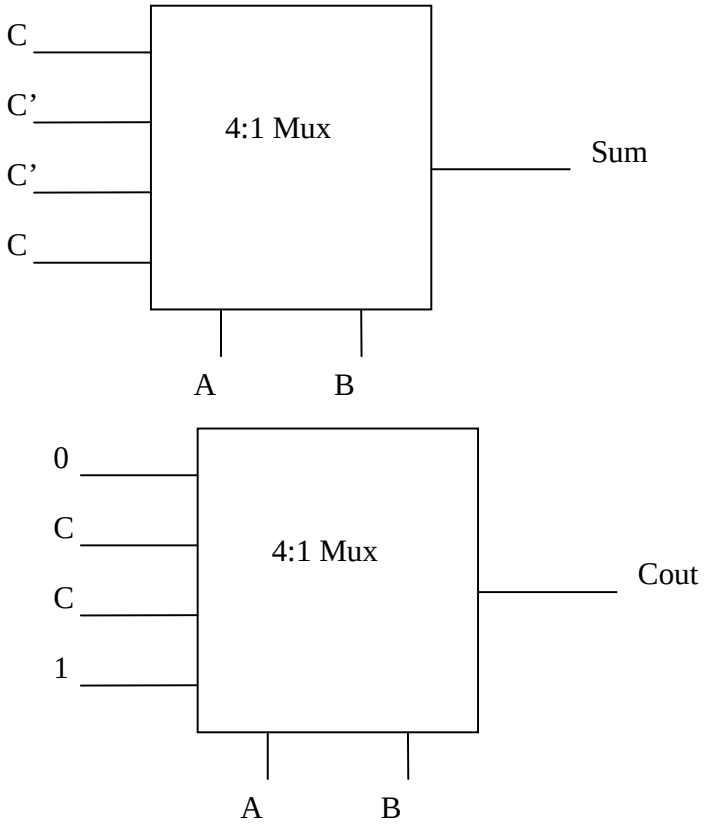
So to get OR gate we need two HA. The Sum and Cout of fist HA are given as inputs to second HA. The Sum of second HA gives the A OR B.

A12) Sum = A XOR B XOR C and Carry = AB + BC + AC

Full adder from 2 HA and one OR gate:



A13) Sum = A XOR B XOR C and Carry = AB + BC + AC

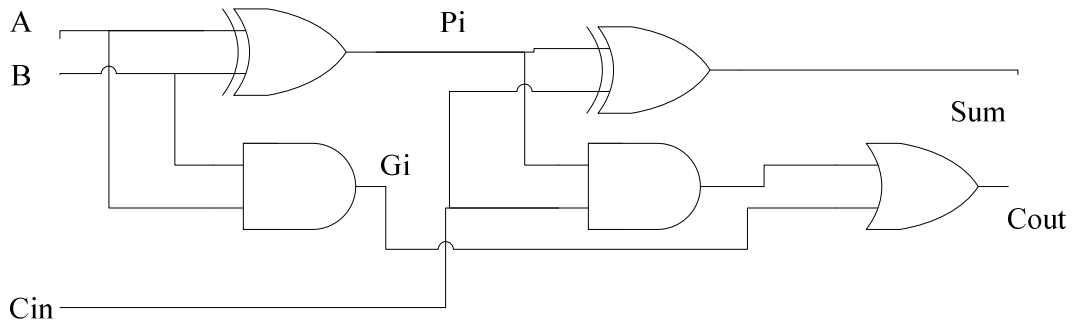


A14) The suitable equations are:

Sum = (A XOR B) XOR C

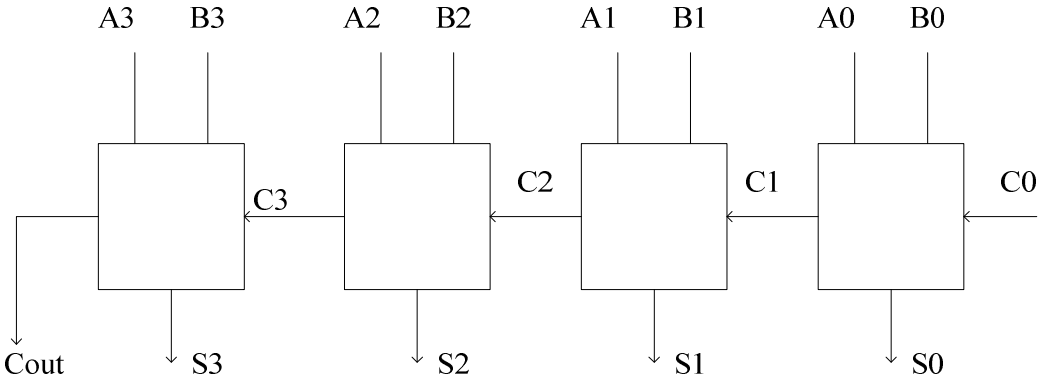
Cout = AB + (A XOR B) C

The implementation is as follows:



A15) Two possible implementations of an adder are (a) Ripple carry adder and (b) Carry Look Ahead adder.

(a) The ripple carry adder for adding two 4 bit numbers, A and B is shown below:



S3-S0 indicates the final result and Cout is the final carry.

(b) Carry Look Ahead (CLA)

From the figure that is shown (A14), we can derive the following intermediate equations:

$P_i = A_i$ XOR B_i and $Gi = A_i$ AND B_i ---------- (1)

Now, Sum = P_i XOR C_i and $C_{i+1} = G_i + P_iC_i$ ---------- (2)

Using this equation for carry, we can generate the following algorithm for carry look ahead:

C_0 = Input carry

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0)$ and so on..----------(3)

By using these recursive equations, we can look a head and decide the carrier, unlike in case of ripple carry adder.

So the three steps involved in CLA adder are:

Step1: Generation of Pi and Gi from Ai and Bi using Eq(1)

Step2: Generation of carrys using Eq(3)

Step3: Finally getting the Sum from Pi and Ci using Eq(2)

A16) In ripple carry adder, the carry propagates from first adder to last. As it has to pass through all the adders, the delay in getting the final output is considerably high. Where as it is hard-ware efficient. The scheme for CLA is explained in the previous question. The major advantage of CLA is faster output. But it needs more hardware.

A17) To perform the delay calculations, use the circuits that are shown in above answers.

(a) For full adder the best implementation is shown in A14.

XOR gate delay = 10ns and AND/OR gate delay = 5ns

The delay for each adder = 10 + 10 + 5 = 25ns.

For adding 4-bits, we need 4 such adders, so <u>overall delay = 100ns</u>

(b) For CLA, as explained in A15, we need 3 main steps:

Step1: Generation of Pi, Gi: The XOR gate delay = 10ns
Step2: Generation of carry: And-Or stage = 5ns + 5ns = 10ns
Step3: Sum generation: One more XOR operation = 10ns
So the overall delay = 30ns

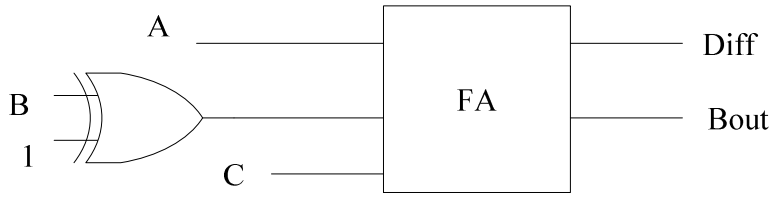
A18) The carry out of a full adder is equivalent to Majority function.
Majority function, Y = Cout = AB + BC + AC

A19) The truth table for full subtractor is shown below:

A	B	C	Bout	Diff
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

Diff = A XOR B XOR C and Bout = BC + A'(B XOR C)

To implement full subtractor from full adder we need to XOR one of the inputs with 1 and then give to full adder as shown in the figure.



A20) F0 + F1 generates 4 carries, FF+FF and FF+F1 generates 8 carries. Whichever generates less number of carries, that one will give the fastest output. So F0+F1 gives fastest output.

A21)

Case1: Unsigned Numbers:

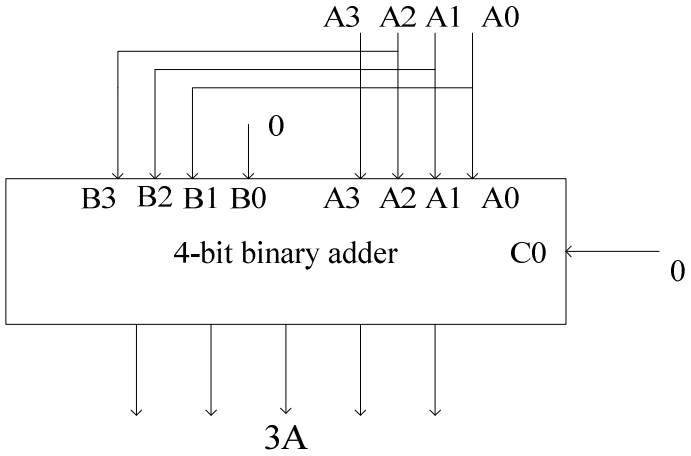
In N-bits, we can represent numbers from 0 to (2^N) - 1. Suppose if we are adding 2 N bit unsigned numbers and if the result is greater than (2^N) - 1 , overflow will occur. To detect this, check whether the MSB addition (Nth bit) + Carry generated from (N-1) bit addition is generating any carry or not. If there is carry out, there is overflow.

Case2: Signed Numbers:

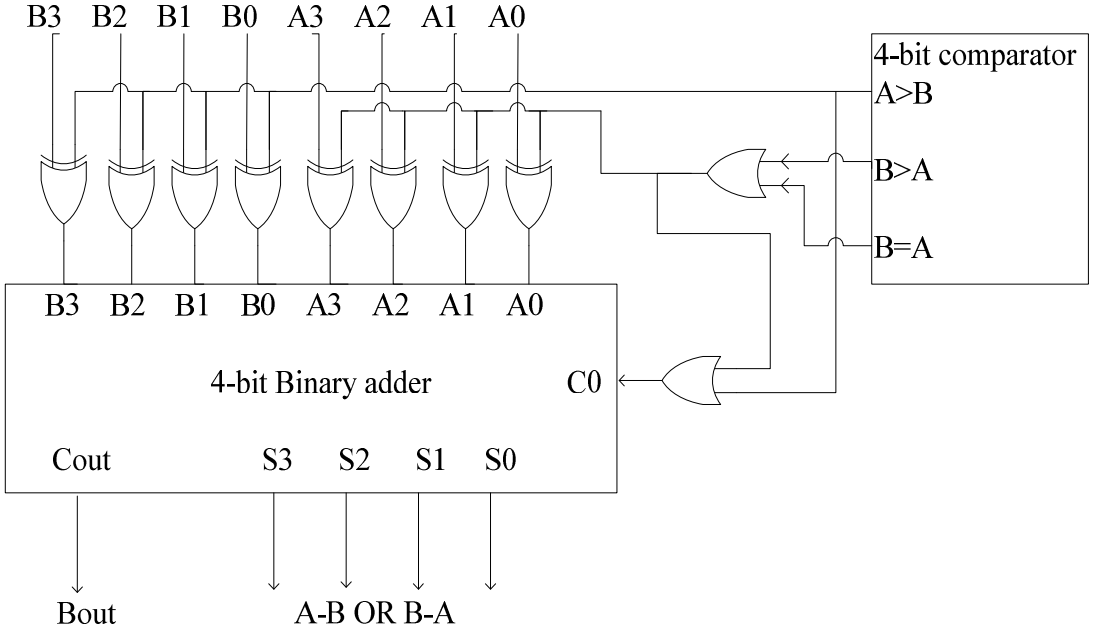
In N-bits, we can represent numbers from -(2^(N-1)) to (2^(N-1)) - 1. Suppose if we are adding 2 N bit signed numbers and if the result is not in the above range, overflow will occur. To detect overflow in this case:

- The sum of two positive numbers is negative;
- The sum of two negative numbers is positive;
- Subtracting a positive number from a negative one yields a positive result; or
- Subtracting a negative number from a positive one yields a negative result.

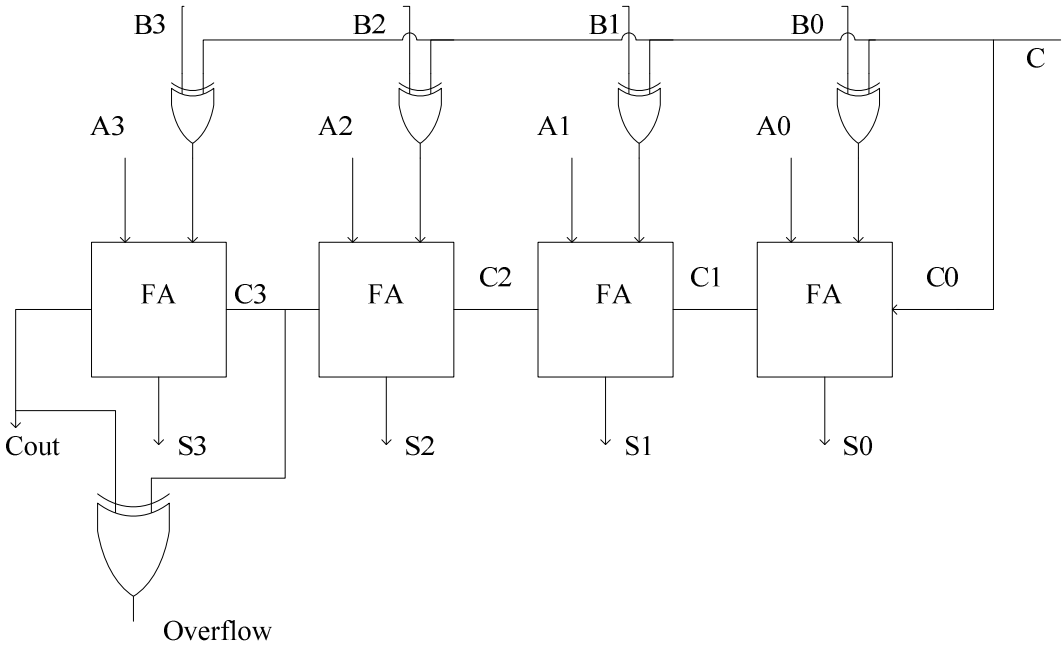
A22) Y = 3A = 2A + A. The 4-bit binary adder which is shown in A15 (a) can be used as a block box here. As 2A can be obtained by simple right shift operation, one binary adder is sufficient for the complete design.



A23) Note that A-B = A + (-B). That is, to subtract B from A, just find the 2's of B and add that to A. If A > B, the comparator gives 1 at A>B and zero at rest of the outputs. That 1 is used as one of the inputs to the XOR gate, to find the 2's compliment of B. Similarly, in case of A<B or A=B, we need to find 2's compliment of A. The complete design is shown below:



A24) This is very much similar to the above design. The extra feature is the indication of overflow, which we can get from XOR of the carries C2 and C3.

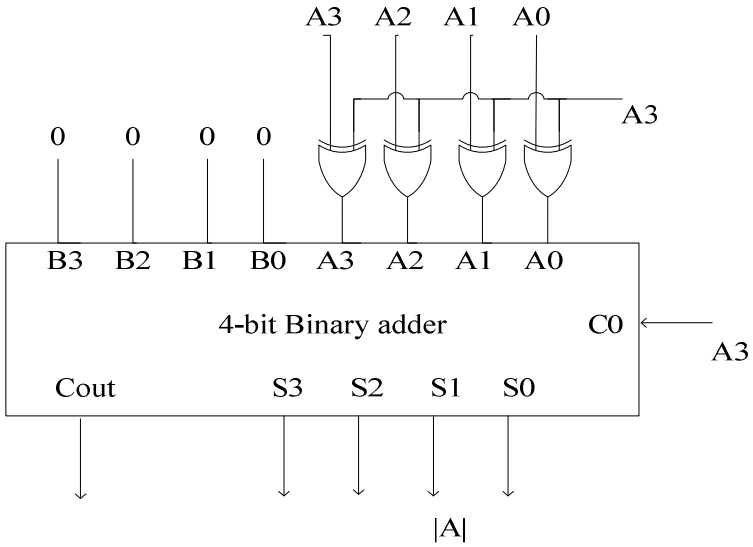


Overflow = 1 indicates that overflow has occurred. (Refer to A21) And the other logic for finding the 2's compliment is exactly same as A24.

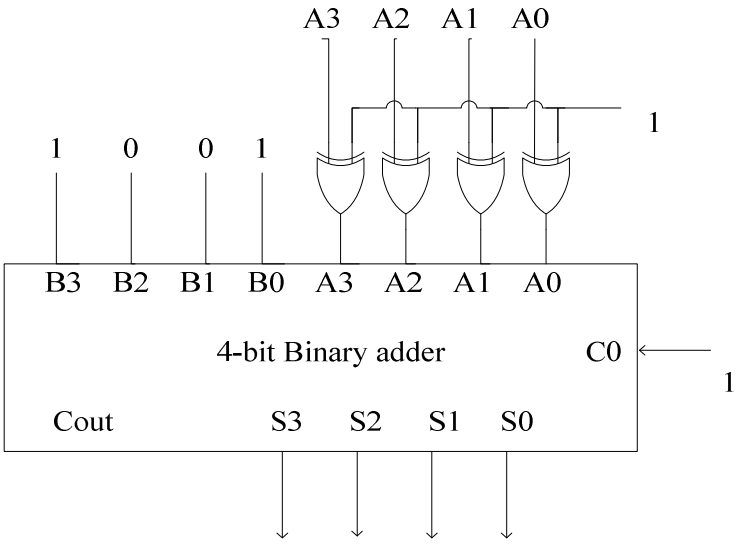
A25) Absolute value of a number is defined as

$|A| = A$ if $A>0$,
$= -A$ otherwise

We can use the above designed adder/subtractor unit to accomplish this task. The MSB of A, which will be 1 if A is negative, can be used as C.



A26)



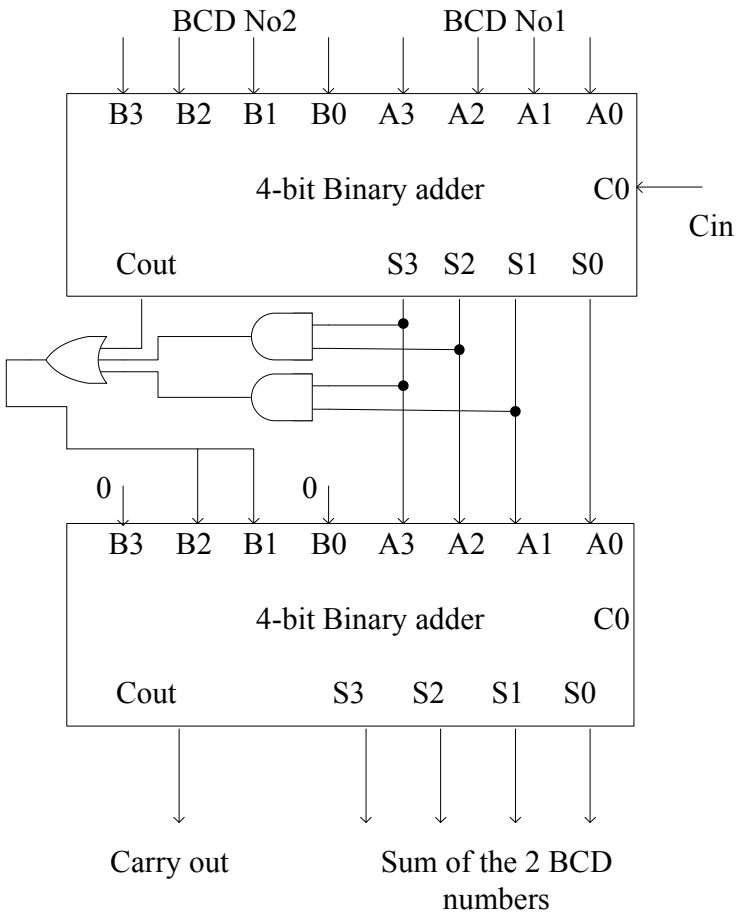
9-A (9's compliment)

9's compliment of a BCD number d is given by 9-d. That is just find the 2's compliment of d and add that to 9. Cout is needed. Just the S3-S0 of the binary adder, shown in the above figure, gives the required result.

A27) The BCD addition is explained in Chapter1. If the result is above 9, it is needed to add 6 to obtain the result in BCD number system. So we need two 4-bit binary adders: One is just to add the two BCD numbers. The second one is for adding 6 or 0 to the result. Extra combination logic is needed to identify the overflow. The condition for detecting the overflow can be derived as,

$$K = Cout + S3\,S2 + S3\,S1$$

K = 1 indicates the overflow and addition of 6 is needed. The complete design is shown below:



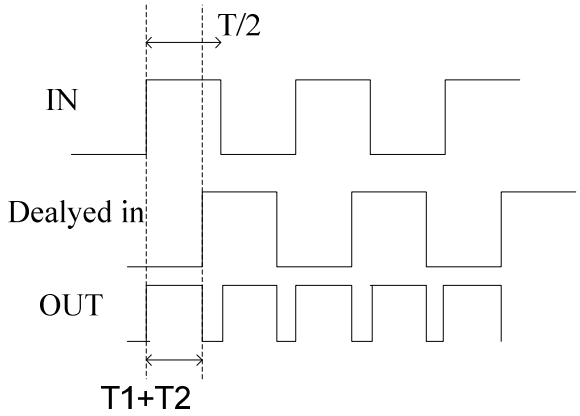
A28) The binary number that is formed from the Carry out as MSB and Sum as LSB, gives the number of 1s of input. The same thing is illustrated in the following table. The sixth column shows Cout-Sum together where as the last column shows the actually number of 1s in the input. Note that both are exactly same.

A	B	C	Cout	Sum	Cout-Sum	No. of 1s in input
0	0	0	0	0	00(0)	0
0	0	1	0	1	01(1)	1
0	1	0	0	1	01(1)	1
0	1	1	1	0	10(2)	2
1	0	0	0	1	01(1)	1
1	0	1	1	0	10(2)	2
1	1	0	1	0	10(2)	2
1	1	1	1	1	11(3)	3

A29)

(a) The circuit works as frequency doubler. That is, it gives double the frequency at the output. But the duty cycle depends upon the delay of the gates.

(b)



If you observe the output, it will be on for (T1+T2) duration. And the total period is T/2. (As output frequency = 2 * input frequency).

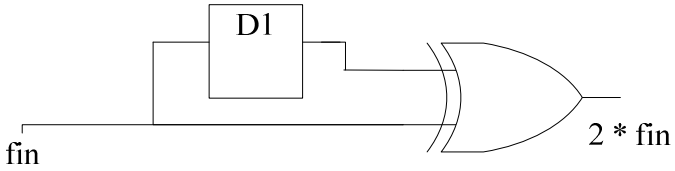
So duty cycle = (T1+T2) / (T/2) = 2 (T1+T2) / T %

(c) To get 50% duty cycle, T1 + T2 should be equal to T/4. So you can use two inverters each with a delay of T/8.

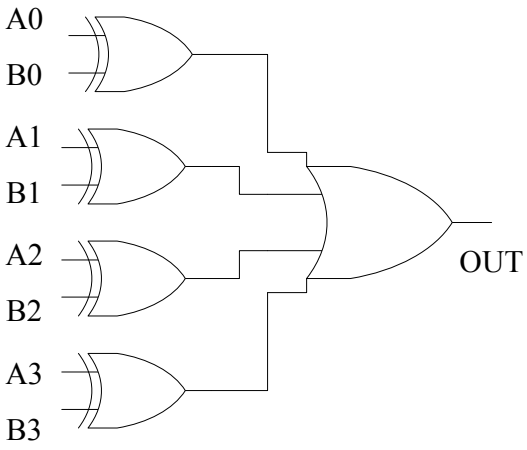
A30) Encoder functionality is opposite of a decoder. The output of an encoder corresponds to the binary code of the input. There is a chance that, in the input stream, more than one 1 may present. In that case, to avoid clash, we need to provide the priority to any one of the bits. Here the truth table for priority encoder which gives, highest priority to its LSB, is shown:

Input				**Output**	
D3	**D2**	**D1**	**D0**	**x**	**y**
0	0	0	0	X	X
X	X	X	1	0	0
X	X	1	0	0	1
X	1	0	0	1	0
1	0	0	0	1	1

A31) The simple design using XOR gate is shown below. (Similar to A29)

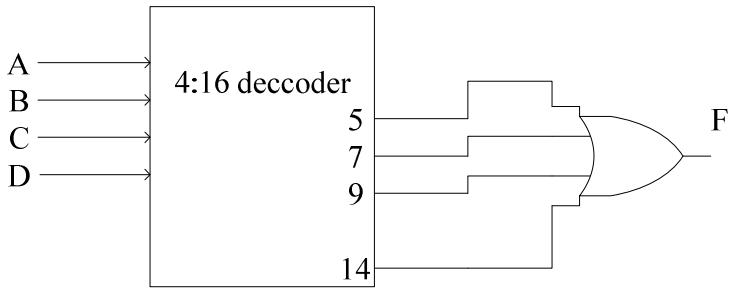


A32) For finding out whether two signals are equal or not, the best logical gate is XOR. The design is shown below. OUT =1 implies that the two binary numbers are not equal.

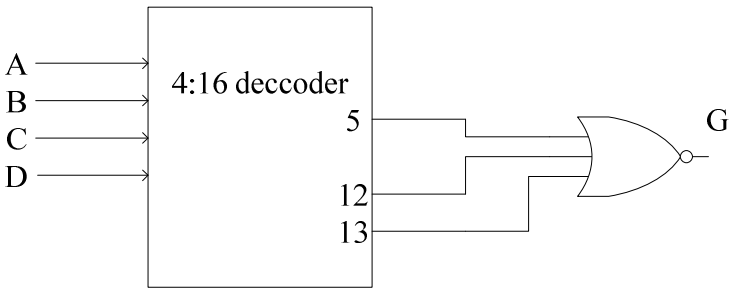


A33) The decoder gives all the possible minterms at the output. We need to just OR all the corresponding minterms to get a particular Boolean functions. If a Boolean function has more minterms which are 1, then take all the minterms which are zero and use NOR gate. Here in case (a) we can directly use OR gate. But case (b), NOR gate is to be used to get the optimal solution.

(a)



(b)



Chapter 5: Introduction to flip-flops

Questions:

Q1) Mention two basic applications of flip-flops?

Q2) What is the difference between a LATCH and a Flip flop?

Q3) What is transparent latch?

Q4) Implement S-R Latch with control input using 2-input NAND gates?

Q5) Which input combinations are not allowed in (a) NAND based (b) NOR based S-R Latch? Explain.

Q6) How to convert S-R Latch to transparent latch?

Q7) Design a D-latch using 2:1 Mux.

Q8) Design a master-slave D-Flip flop using D-Latch?

Q9) What is race-around condition? Explain it in case of J-K Latch and solution to avoid that?

Q10) We have a circular wheel with half painted black and the other half painted white. There are 2 censors mounted 45 degree apart at the surface of this wheel(not touching the wheel) which give a "1" for black and "0" for white passing under them. Design a circuit, using DFF to detect which way the wheel is moving. Can not assume any fixed position for start.

Q11) Give the characteristic table and characteristic equation for J-K Flip-flop?

Q12) Construct a J-K flip flop using a DFF, 2:1 Mux and an-inverter?

Q13) Implement T-flip flop from D-flip flop?

Q14) Show how to convert J-K flip flop into (a) T-flip flop (b) D-flip flop?

Q15) What is excitation table?

Q16) Write the excitation table for T-flip flop?

Q17) Draw the circuit for a D flip flop with Synchronous Reset?

Q18) Which flip-flop can be used as delay switch?

Q19) Which flip-flop can be used as toggle switch?

Q20) Using DFFs and minimum no. of 2×1 Mux, implement the following XYZ flip-flop.

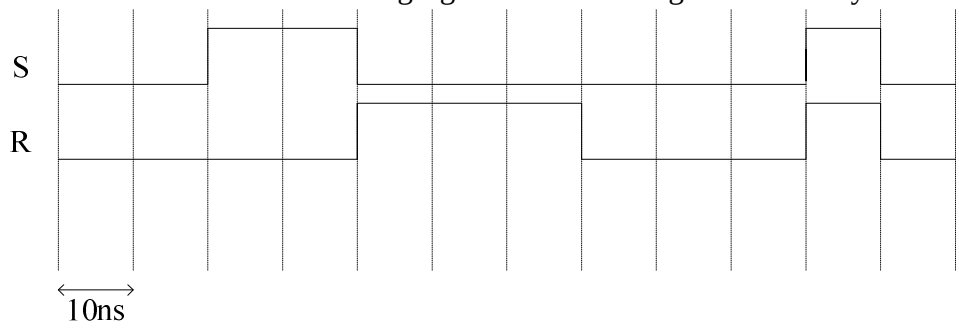
X	Y	Z	Q(t+1)
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	Q(t)
1	0	1	Q(t)`
1	1	0	Q(t)`
1	1	1	Q(t)

Q21) A AB flip flop has 4 operations: clear to 0, no change, compliment and set to 1, when inputs A and B are 00, 01, 10 and 11 respectively. Derive the characteristic equation?

Q22) Give the excitation table for the AB flip flop described in Q20?

Q23) Show how the AB flip-flop can be converted to a D flip-flop?

Q24) Draw the output waveforms Q and Q' for a NOR based S-R Latch for the S and R waveforms shown in the following figure if each NOR gate has a delay of 10ns.



Q25) Design a TFF using only 2:1 Muxes?

Q26) In C-N (Change-No change) flip flop, there won't be any change in output as long as N is 0, irrespective of C. If N=1, then if C = 0 output will change to zero else if C =1 output will be the compliment of previous output.

(a) Write the characteristic table ?
(b) Design this using J-K flip-flop?

Answers:

A1) The major applications of flip-flops are:

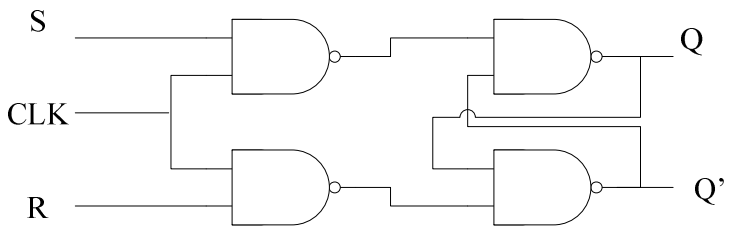
- Data storage
- Data transfer
- Counting and
- Frequency division

A2) The differences between a LATCH and a FLIP-FLOP are:

- Latch is a level sensitive device where as flip-flop is edge sensitive
- Latch is sensitive to glitches on enable pin and flip-flop is immune to glitches
- Latches take less gates(also less power) than flip-flops
- Latches are faster than flip-flops.

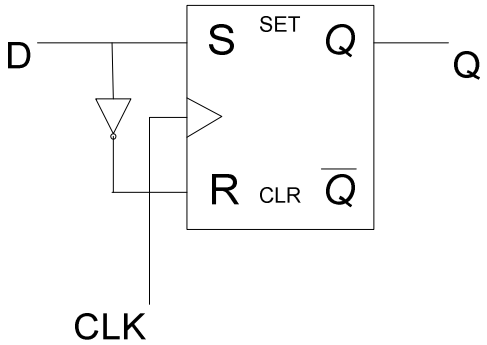
A3) D-Latch is called transparent Latch. As it transfers the data as it is to the output on enable.

A4) S-R Latch with clock using 2-input NAND gates is shown below:

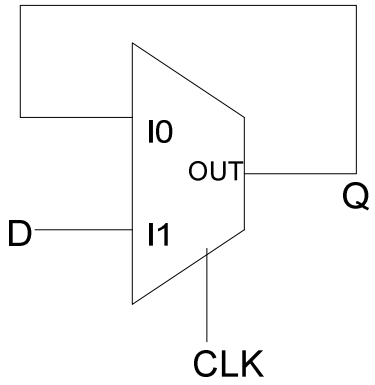


A5) (a) S=R=0 (b) S=R=1

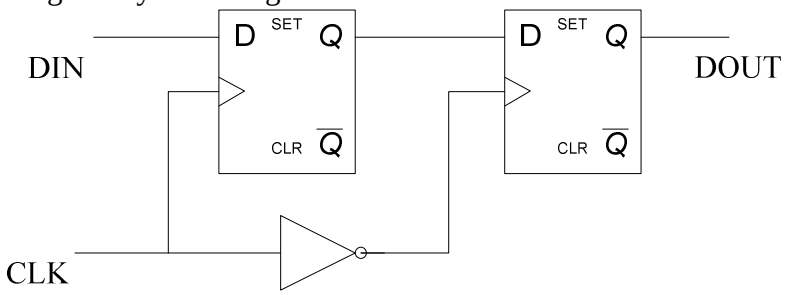
A6) Transparent latch from S-R Latch:



A7) D-Latch using 2:1 Mux:



A8) 2 D-latches and an inverter are needed to get the master-slave configuration. The major advantage of master-slave configuration is avoiding the race-around condition as the input changes only at the edges.



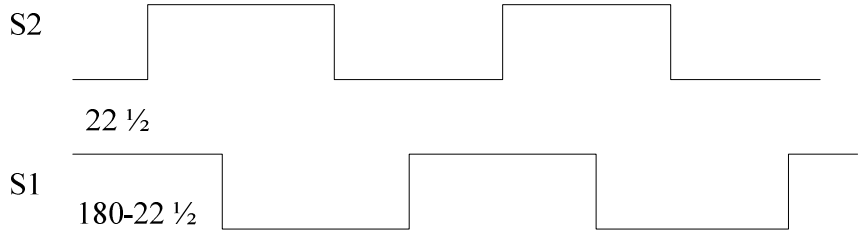
A9) The race around condition means: the output oscillating between 0s & 1s. This problem will occur in Latches especially if the clock is high for long time. In case of J-K Latch, J=K=1 gives Q(t+1) = Q(t)' . Consider the case when clock is high for long time and J=K=1. Then the output oscillates between 0 & 1 continuously as long as the clock is high. To avoid this, use Master-Slave configuration which latches the input only at clock edges. So in that case, irrespective of the duration of clock high, output will be just compliment of previous output. There won't be any oscillation as such.

A10) The sensor will give 1 for Black and 0 for white. First we will draw the outputs of the sensors assuming some position of the wheel. Assume that the initial position of the wheel is as shown in the figure with respect to the sensors. The output waveforms of S1 and S2 will be as follows. Both clock wise and counter clock wise wave forms are shown here. It is clear from the waveforms that there is an initial delay of 22 ½ degrees between the two waveforms (assuming the two sensors are at the same distance from the partition).

Initially S2 = 0 and S1 =1 because of their initial positions(this is just our assumption). Once the wheel started moving in anti-clock wise direction, S1 will go to 0 from 1 after 22 ½ degrees. And then gets complemented once in 180 degrees. In this case, S2 will go to 0 to 1 after (180 -22 ½) degrees and then complements once in 180 degrees.

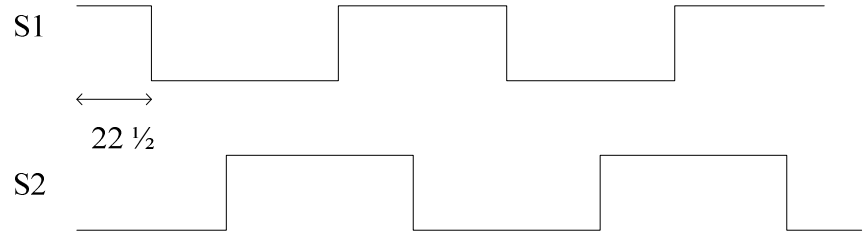
Similarly we can analyze for clock-wise case also. The waveforms for both the cases are shown below.

(a) For Clock-wise case the waveforms are:



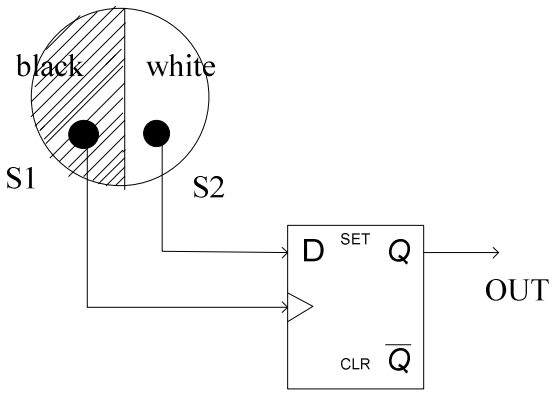
Clock-wise rotation

(b) For anti-clock wise, the waveforms are:



Anti-Clock-wise rotation

These two wave forms are connected to DFF as shown in the following figure. That is S1 to the clock and S2 to the D input.



If you observe the above waveforms, for clock-wise direction, whenever there is a rising edge of S1, S2 is always 0. For anti-clock wise direction, at the rising edge of S1, S2 is 1 always.

That is OUT =1 in the above circuit indicates the clock-wise rotation and OUT = 0 indicates anti-clockwise rotation.

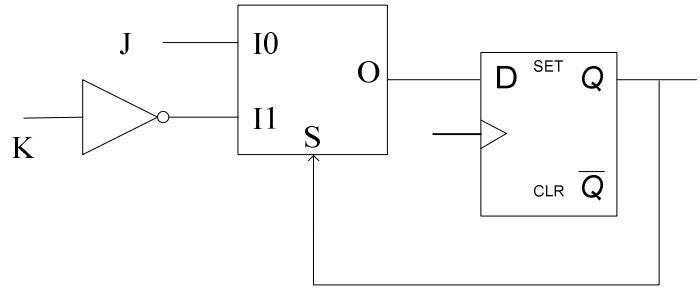
A11) The characteristic table for J-K flip flop is:

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q'(t)

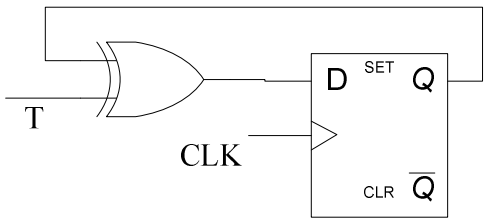
To get the characteristic equation, we can use K-Map with 3 inputs, J,K and Q(t) and obtain,

$$Q(t+1) = J\,Q(t)' + K'\,Q(t)$$

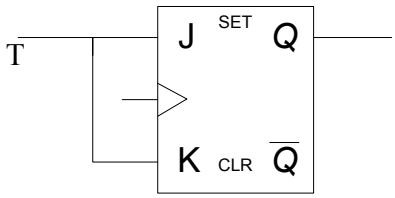
A12) The complete design is shown here. The catch here is to use Q as select line. You can observe the cofactors of Q(t+1) with respect to J,K and Q(t). Using J or K as the select line with 2:1 mux will not do.



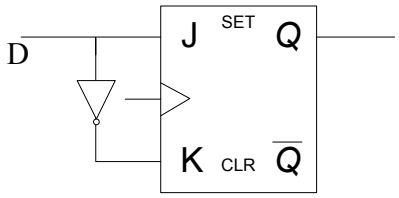
A13) T-flip flop using DFF:



A14) (a) TFF:



(b) DFF:

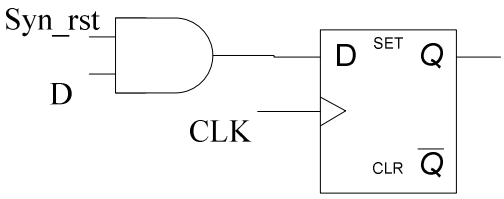


A15) During the design process we usually know the transition from present state to the next state and wish to find the flip-flop input conditions that will cause the required transition. For this reason we will need a table that lists the required inputs for a given change of state. Such a list is called the excitation table.

A16) The excitation table for T flip flop is shown below.

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

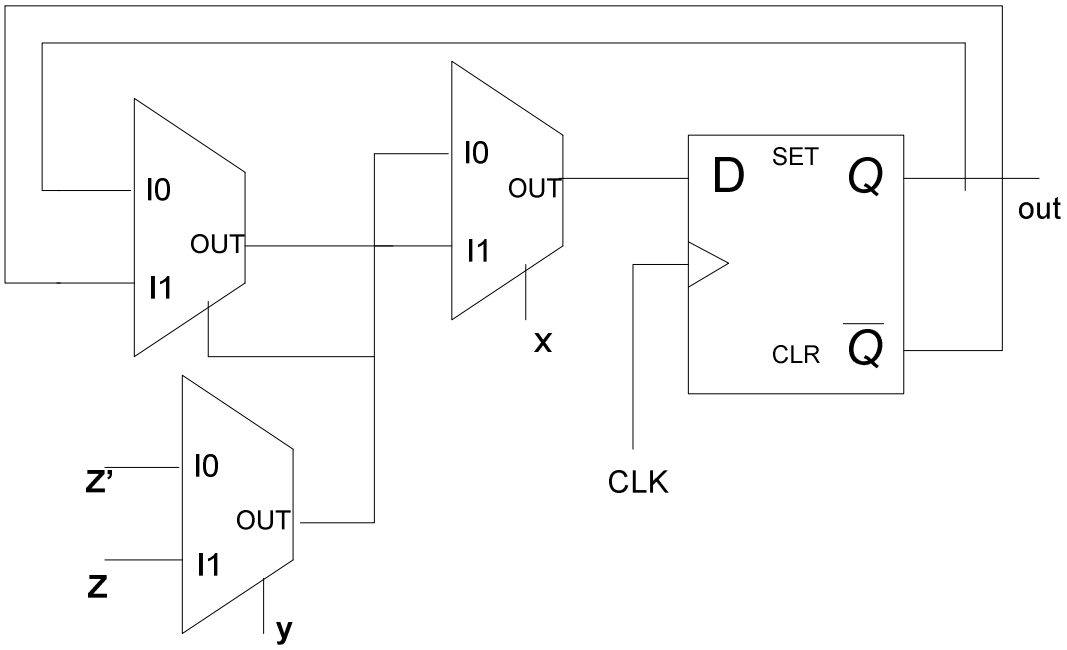
A17) Synchronous reset will clear output only at clock edge unlike asynchronous reset. At clock edge, if syn_rst = 0, output will be 0 otherwise output will be D. So we just need an AND gate before the DFF as shown in the figure.



A18) As DFF transmits the data as it is to the output, it can be used to provide one clock delay.

A19) In TFF, if T=1, output just toggles between 1 and 0. So TFF can be used as toggle switch.

A20) From the given characteristic table, it is clear that if X=0, Q(t+1) = Y XNOR Z.
If X =1 and if (Y XNOR Z), Q(t+1) = Q(t), else Q(t)'. So we need two 2:1 mux to generate Y XNOR Z. One to select Q(t) and Q(t)' and one more to select between X=0 case and X=1 case. Total we need 4 2:1 mux. The design is shown here except the generation of Z'.



A21) The characteristic table is shown below:

A	B	Q(t+1)
0	0	0
0	1	Q(t)
1	0	Q(t)'
1	1	1

The characteristic equation can be derived as,

$Q(t+1) = A\,Q(t)' + B\,Q(t)$

A22) The excitation table for AB flip flop is,

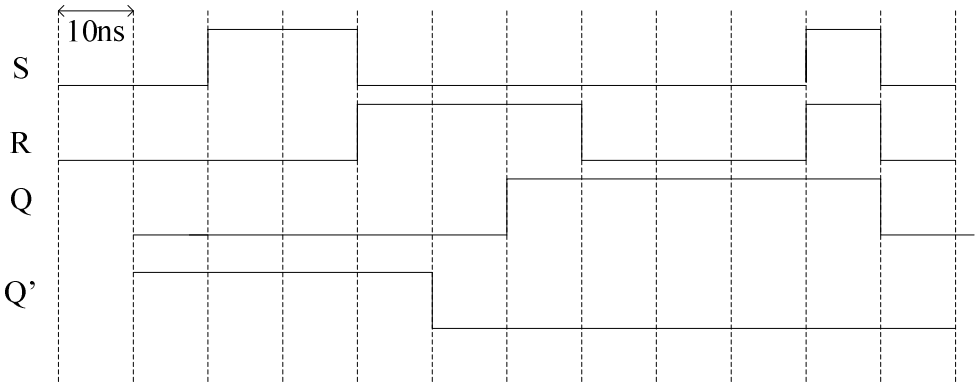
Q(t)	Q(t+1)	A	B
0	0	0	X
0	1	1	X
1	0	X	0
1	1	X	1

A23) To get D from flip-flop from AB flop, just connect A=B=D.

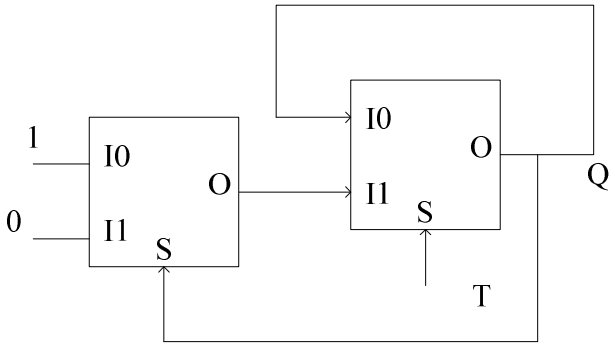
We can prove this from the characteristic equation ,

$Q(t+1) = D\,Q(t)' + D\,Q(t) = D\,(1) = D$

A24) For NOR based S-R Latch, there is no change in output for S=R=0, 0 for S=1,R=0 and 1 for S=0 and R=1. The waveforms are drawn using this. (Delay of NOR gate = 10ns). Assume that initially Q = 0 and so Q' = 1.



A25)



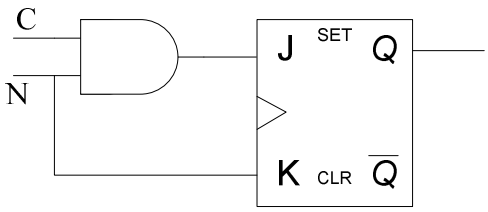
A26) (a) The characteristic table for C-N flip flop is,

C	N	Q(t+1)
0	0	Q(t)
0	1	0
1	0	Q(t)
1	1	Q'(t)

The characteristic equation can be derived as, Q(t+1) = (CN) Q(t)' + N' Q(t)

(b) Comparing the characteristic equation of C-N flip flop with that of J-K flip flop (Q(t+1) = J Q(t)' + K' Q(t)), we will get,

The J-input = CN and K-input = N

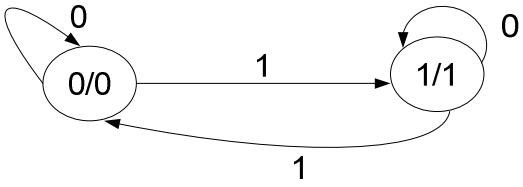


Chapter 6: Finite State Machines – Synchronous Sequential design

Questions:

Q1) Give the State Machine for Serial 2's complementer. Draw the complete design for the same using DFF?

Q2)



(a) Write the output sequence for the input sequence: 001010110110110111
(b) What is the functionality of above FSM?
(c) Which flip-flop can be used to implement above FSM with out any external logical gates?

Q3) Give the state transition diagram for J-K flip flop?

Q4) A sequential circuit with 2 D flip-flops, A and B and an external input x, is specified by the following next-state equations:

A(t+1) = x'A + x B
B(t+1) = x A' + x'B

(a) List the state table showing present state and next states?
(b) Draw the corresponding state diagram?
(c) Explain the functionality of the circuit?

Q5) The sequential circuit oscillates in any of the following cases as long as the external input X = 0 based on the initial state:

Case (a) : 00 → 01 → 00 → 01 ….or
Case (b) : 10 → 11 → 10 → 11 ….

And if X =1, if it is currently in Case(a), the circuit switches to Case(b) and vice versa.

(a) Draw the state diagram?
(b) Implement the circuit using two J-K flip flops and external gates?

Q6) What is Moore model & Mealy model? Explain.

Q7) Give the State Machine for detecting the sequence "1010" from a serially coming data.

Q8) Repeat Q7 for Non-overlapping case.

Q9) Draw the state diagram to output a "1" for one cycle if the sequence "0110" shows up and the leading 0s cannot be used in more than one sequence.

Q10) Describe a finite state machine that will detect three consecutive coin-tosses (of one coin) that result in heads.

Q11) Reduce the following state table:

	Next State		**Output**	
Present State	**x = 0**	**x =1**	**x = 0**	**x =1**
a	a	b	0	0
b	e	c	0	1
c	a	d	0	1
d	e	f	0	1
e	a	f	0	0
f	g	d	0	1
g	a	b	0	0

Q12) Starting from state a, write the next state and output values for the input sequence: 01010101001 for both Original and reduced state table.

Q13) Design a FSM to detect more than one 1s in the last 3 samples.

Eg : I/P 0 1 0 1 0 1 1 0 0 1
O/P 0 0 0 1 0 1 1 1 0 0

Q14) Design FSM for a pattern matching block : Output is asserted 1 if pattern “101” is detected in last 4 inputs.

Eg: I/P 0 1 0 1 0 0 1 1 0 1 0 1 0
O/P 0 0 0 1 1 0 0 0 0 1 1 1 1

Q15) Design an FSM which gives O/P 1 if alternate 1’s & 0’s are present in the last 3 samples else 0.

Eg: I/P : 0 0 1 0 1 0 1 1 0 1 0 0 0
O/P : 0 0 0 1 1 1 1 0 0 1 1 0 0

Q16) It is required to eliminate short length pulses from a sampled data. It means 0’s in continuous 1’s have to be made 1 & similarly 1’s in 0’s are to made 0’s as shown in the following example. Give the FSM for the same.

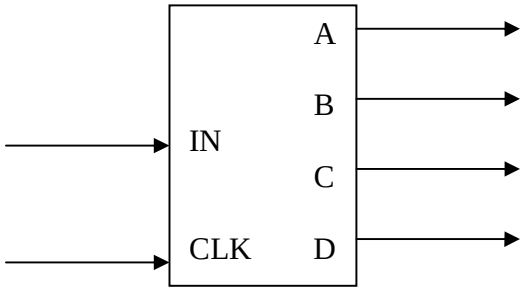
Eg: I/P 0 1 0 0 1 1 0 1 1 0 0
O/P 0 0 0 0 0 1 1 1 1 1 0

Q17) What is one-hot method? List the advantages and disadvantages?

Q18) Show the state assignment using Johnson’s method for a FSM of 6 states?

Q19) How many flip flops are needed to design a FSM having N states if the state assignment is done by using (a) Binary (b) Gray (c) One hot (d) Johnson

Q20)



The block diagram shown above has one input “IN” which is coming serially @ clock, “CLK”. It has 4 outputs A, B,C & D . A will be 1 if IN has even number of 1’s & even number of 0’s. Similarly B will be 1 for even 1’s odd 0’s, C for odd 1’s even 0’s and D for both odd. Give the FSM required to design above block.

Q21) For the above problem, which method is more suitable for state assignment?

Q22) Draw the state diagram for a circuit that outputs a "1" if the aggregate serial binary input is divisible by 5. For instance, if the input stream is 1, 0, 1, we output a "1" (since 101 is 5). If we then get a "0", the aggregate total is 10, so we output another "1" (and so on).

Input	Sequence	Value	Output
1	1	1	0
0	10	2	0
1	101	5	1
0	1010	10	1
1	10101	21	0

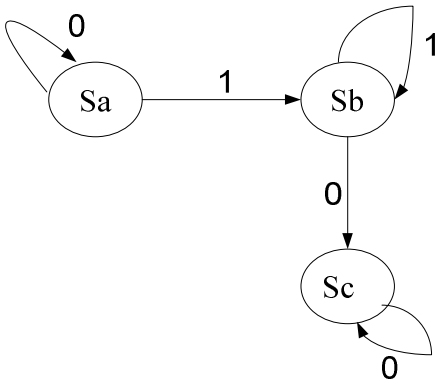
Q23) Draw the FSM for checking whether the two inputs P and Q have same value for the last three continuous samples?

Q24) If the number of 1s in inputs X and Y since reset is multiple of 4, the output is 1 and 0 otherwise. Give the FSM required for designing of the sequential circuit?

Q25) Draw the FSM for a Moore model in which the output is asserted 1 if A had the same value at each of the two previous clock ticks or B has been 1 since the last condition was true . Note that A & B both are inputs.

Eg: A 0 0 1 1 0 0 1 0 1 1 0
B 0 0 0 0 0 0 1 1 0 1 0
O/P 0 1 0 1 0 1 1 1 0 1 0

Q26) A state diagram is shown in the following figure: (States are named as Sa, Sb & Sc)



The system is initially in state Sa. If * represents zero or more occurrence of a logic level and + represents one or more occurrence of a logic level which of the following best describes the sequence of states the system could have gone through if it is finally in state Sc.

a) 0* → 1+ → 0 +
b) 0* → 1* → 1*
c) 0* → 0* → 1
d) 0+ → 1+ → 0*

Answers:

A1)

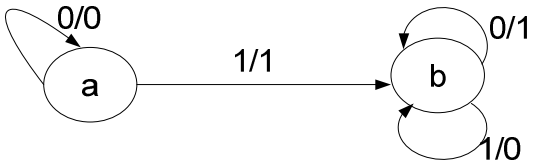
State Machine for Serial 2's complementer:

Logic: Starting from LSB, retain all the bits till first one has occurred including the first one and then complement all the following bits.

State Definition:

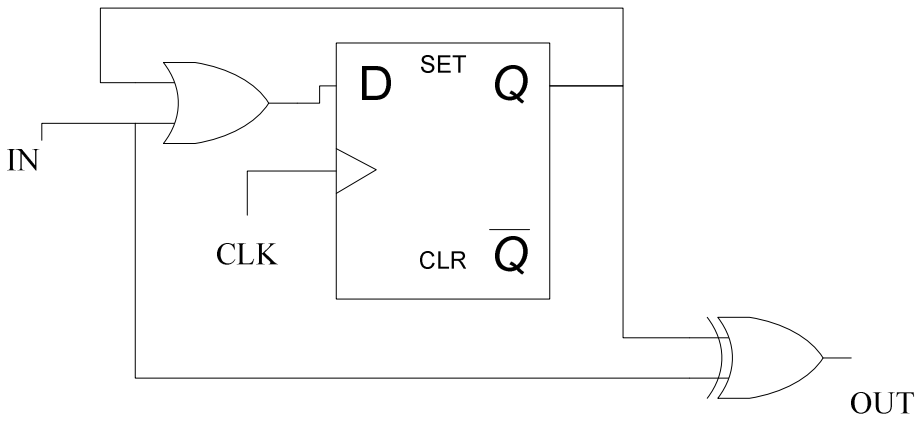
State a : No one has occurred
State b : After first one has occurred

State Diagram:



Design using DFF:

PS	INPUT	NS	OUTPUT
0(a)	0	0(a)	0
0(a)	1	1(b)	1
1(b)	0	1(b)	1
1(b)	1	1(b)	0



A2)

(a) INPUT Sequence : 0 0 1 0 1 0 1 1 0 1 1 0 1 1 0 1 1 1
OUTPUT Sequence : 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 1 0 1

(b) Note that the OUTPUT is 1 if the input (that has arrived till now) contains odd number of 1's. So the FSM shows the functionality of **serial odd parity indicator**.

(c) The state table is as follows:

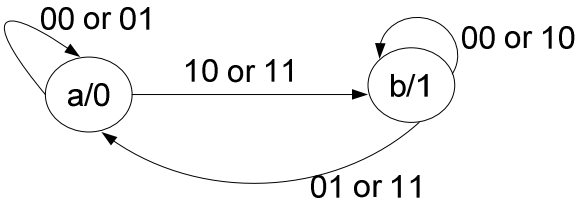
PS	INPUT	NS	OUTPUT
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

The state table shows that NS = OUTPUT = PS XOR INPUT
This is same as the characteristic equation of T- Flip flop.

A3)

State transition diagram for J-K Flip-flop:

The first bit of the two bits is J and the other is for K.

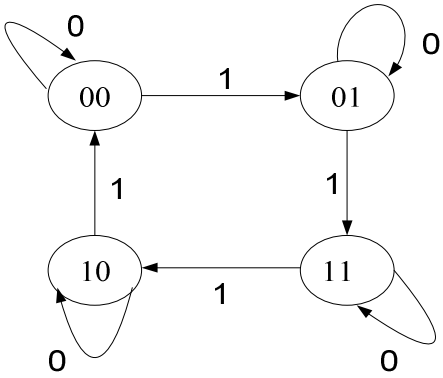


A4) $A(t+1) = x'A + xB$
$B(t+1) = xA' + x'B$

(a) *State Table:*

Present State		x	Next State	
A	B		A(t+1)	B(t+1)
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	1	0
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0

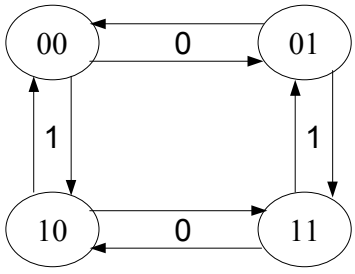
(b) *State Diagram:*



(c) *Functionality:*

This circuit works as enable based gray code counter. If X=1, the transition takes place in gray code counter and if X=0, there won't be any change in the present state.

A5) One of the possible FSMs is shown below:



State table and flip flop inputs:

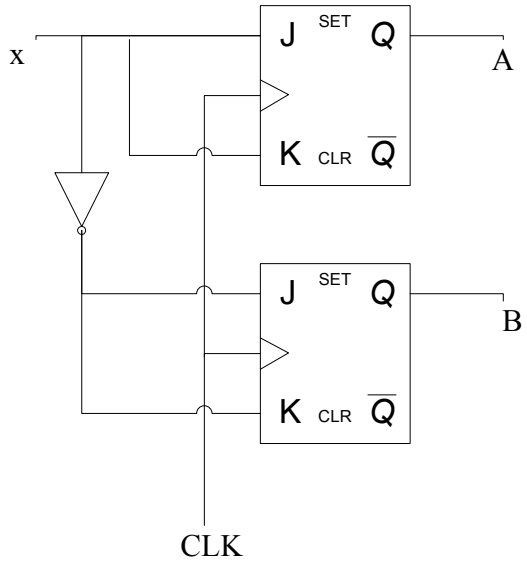
Present State		INPUT	Next State		Flip Flop inputs			
A	**B**	**x**	**A**	**B**	J_A	K_A	J_B	K_B
0	0	0	0	1	0	X	1	X
0	0	1	1	0	1	X	0	X
0	1	0	0	0	0	X	X	1
0	1	1	1	1	1	X	X	0
1	0	0	1	1	X	0	1	X
1	0	1	0	0	X	1	0	X
1	1	0	1	0	X	0	X	1
1	1	1	0	1	X	1	X	0

Design using J-K Flip flop:

The simplification using K-Map gives,

$J_A = x$, $K_A = x$,

$J_B = x'$, $K_B = x'$



A6)

A state machine consists of set of states, initial state, input symbols and transition function that maps input symbols and current state to next state.

Mealy machine: machines having outputs associated with transition of input and the current state. So Mealy machines give you outputs instantly, that is immediately upon receiving input, but the output is not held after that clock cycle.

Moore machine: machines having outputs associated with only states, that is the outputs are the properties of states themselves. The output will vary only after the states are

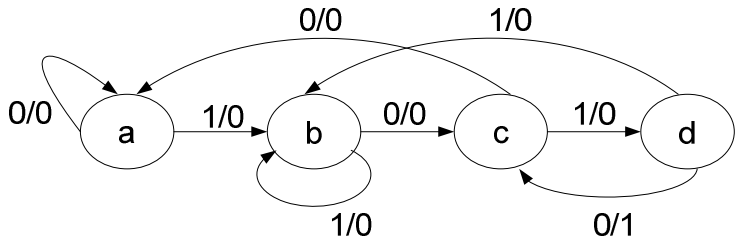
varied. So the changes in the output will be only during clock transitions and the outputs are held until you go to some other state

Advantages and Disadvantages:
In Mealy as the output variable is a function of both input and state, changes of state of the state variables will be delayed with respect to changes of signal level in the input variables, there are possibilities of glitches appearing in the output variables. Moore overcomes glitches as output is dependent on only states and not the input signal level.

A7)
State Machine for detecting "1010":
Logic: Check for the bit pattern 1010. The end "10" has to be reused for next pattern.
State Definition:

State a : No 1 detected state
State b : One 1 detected state
State c : 10 detected state
State d : 101 detected state

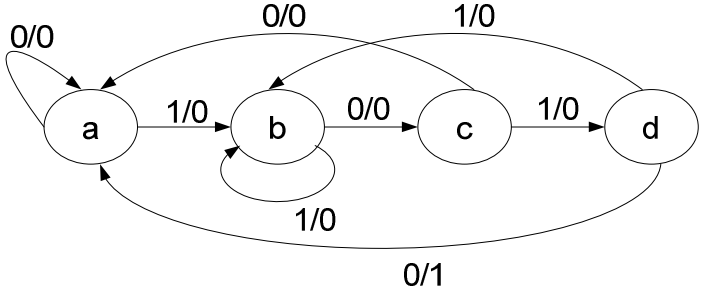
State Diagram:



A8)
State Machine for detecting "1010":
Logic: Check for the bit pattern 1010. The end "10" can't be reused. So after detection of one pattern, just go to initial state. (Here it is State a).
State Definition:

State a : No 1 detected state
State b : One 1 detected state
State c : 10 detected state
State d : 101 detected state

State Diagram:



A9)

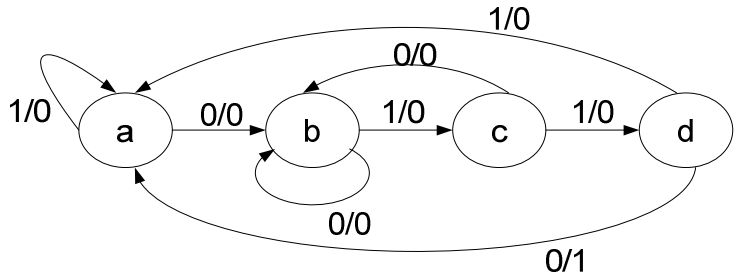
State Machine for detecting "0110":

Logic: Check for the bit pattern 0110. The end "0" can't be reused. So after detection of one pattern, just go to initial state. (Here it is State a). That is non overlapping case.

State Definition:

State a : No 0 detected state
State b : At least one 0 detected state
State c : 01 detected state
State d : 011 detected state

State Diagram:



A10)

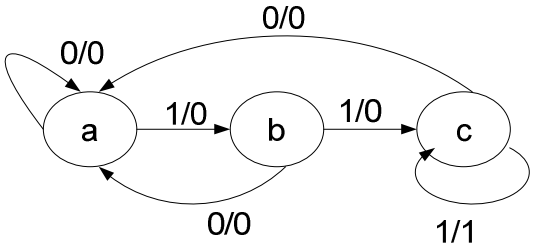
State Machine for detecting "111":

Logic: If we represent Head with logic 1 and tail with logic 0, Checking for 3 consecutive heads is nothing but pattern matching for "111" (overlapping)

State Definition:

State a : No 1 detected state
State b : One 1 detected state
State c : More than Two 1's detected state

State Diagram:



A11) To remove a state, we need to have another state with the same next state values and output values. If we observe the given state table, state g has all entries same as those of state a. So state g can be replaced with a everywhere. Once g is replaced with a, all the entries of f are same as those of c. Thus, replacing f with c, makes state d same as state b. So with all these changes the reduced state table is shown below:

Present State	Next State x = 0	Next State x =1	Output x = 0	Output x =1
a	a	b	0	0
b	e	c	0	1
c	a	b	0	1
e	a	c	0	0

A12)

(a) For Original:

The input sequence: 0 1 0 1 0 1 0 1 0 0 1
Next state : a a b e f g b e a a a b
Output sequence : 0 0 0 0 0 0 0 0 0 0 0

(b) For Reduced:

The input sequence: 0 1 0 1 0 1 0 1 0 0 1
Next state : a a b e c a b e a a a b
Output sequence : 0 0 0 0 0 0 0 0 0 0 0

A13)

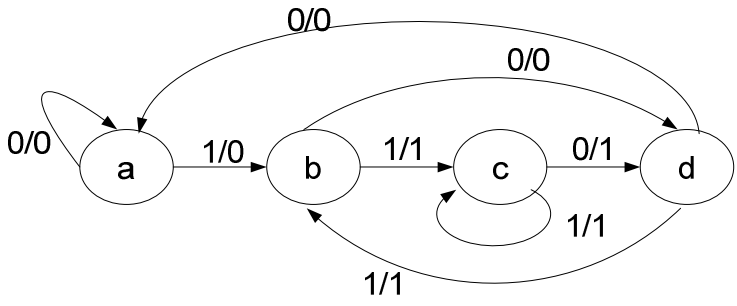
State Machine for detecting more than one 1 in last 3 samples:

Logic: Check for the patterns 011, 101, 110 or 111. These 4 patterns have more than one 1.

State Definition:

State a : No 1 detected state, continuous 0s
State b : One 1 detected state, "01" or "1" detected state
State c : Atleast two 1s detected state, "011" or "111" detected state
State d : "010" or "001" detected state

State Diagram:



A14)

State Machine for detecting "101" in last 4 samples:

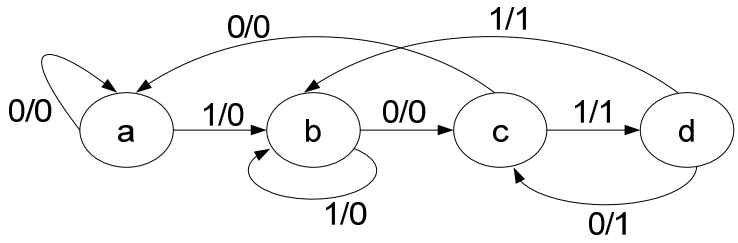
Logic: Possible patterns are: 0101,1101,1010,1011 (overlapping)

State Definition:

State a : Continuous 0s

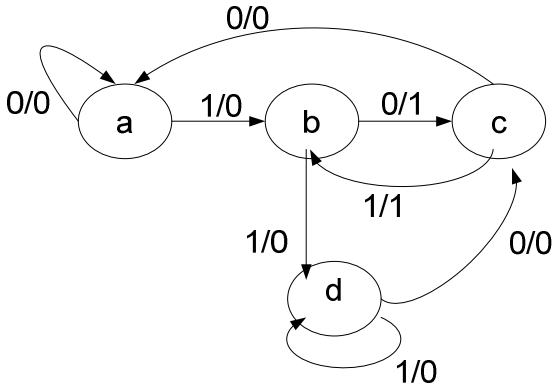
State b : Atleast one 1, that is “01” or “111….”
State c : 010 detected state
State d : 0101 or 1101 detected state

State Diagram:



A15)
State Machine for detecting alternative 1’s and 0’s in last 3 samples:
Logic: Check for the patterns 101 or 010 in last 3 samples.
State Definition:
State a : No 1 detected state, continuous 0s
State b : One 1 detected state, “01” or “1” detected state
State c : “10” detected state
State d : continuous 1’s detected state

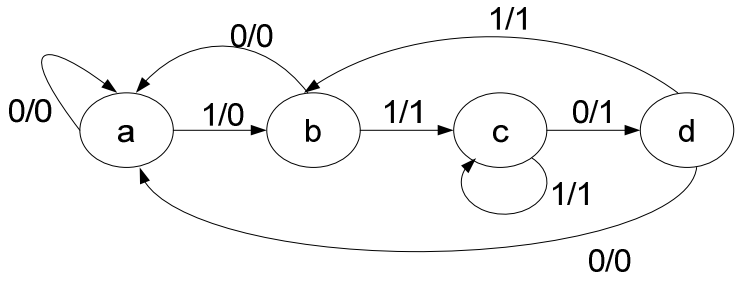
State Diagram:



A16)
State Machine for eliminating short length pulses:
Logic: The 1 after two successive 0’s will be made 0. Similarly the zero after two successive 1’s will be made 1. If you are continuous 1s state, minimum 2 0’s zeros are needed to switch to continuous 0s state and vice versa.
State Definition:
State a : Continuous 0s
State b : One 1 in between 0s

State c : Continuous 1s
State d : One 0 in between 1s

State Diagram:



A17)
One-Hot encoding of FSM uses one flip-flop per state. Only one flip-flop is allowed to have 'logic 1' at anytime. For example, for a five state FSM, the states are "00001", "00010", "00100", "01000", "10000". All other states are illegal. One-Hot encoding trades combinational logic for flip-flops. One hot reduces the next state and output logic complexity. Its good for 'flip-flop' rich implementation technologies. Because the combinational logic is reduced, the length of the critical path can be reduced resulting in a faster FSM. Speed increase is more significant for larger finite state machines. The disadvantage is we end up in using more number of flops.

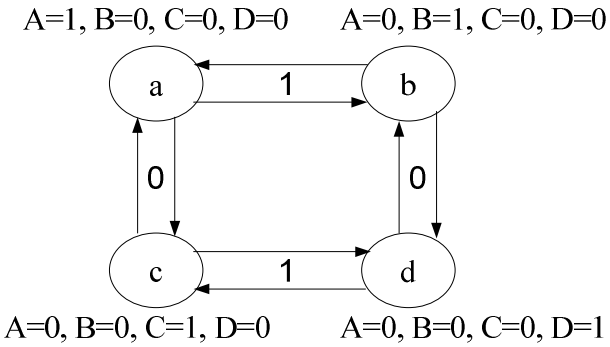
A18) Johnson's method: 000,001,011,111,110,100

A19) (a) Log_2N
(b) Log_2N
(c) N
(d) N/2

A20)
State Machine for identifying whther the 1's and 0's are even or odd:
Logic: The only 4 possibilities are even-even, even-odd, odd-even, odd-odd. So initial state will be even-even as no 1 or no 0. Now if 1 comes it will be even 0s odd 1. Similarly if 0 comes it will be odd 0's even 1's. So the state transition will take place accordingly.
State Definition:
State a : Even 0's Even 1's
State b : Even 0's Odd 1's
State c : Odd 0's Even 1's
State d : Odd 0's Odd 1's
Outputs:
In state a, outputs are: A = 1, B = 0, C = 0, D=0.
In state b, outputs are: A = 0, B = 1, C = 0, D=0.

In state c, outputs are: A = 0, B = 0, C = 1, D=0.
In state d, outputs are: A = 0, B = 0, C = 0, D=1.

State Diagram:



A21) One hot method.

a : 1000, b : 0100, c : 0010, d : 0001

These values are nothing but the four outputs that are needed. So it reduces the output logic complexity.

A22)

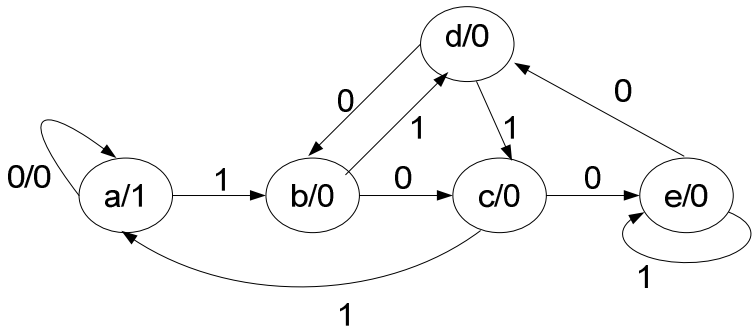
State Machine to detect whether the serial binary number is divisible by 5 or not:

Logic: From the given example we can notice that the data is coming from MSB side. And the possible reminders are 0,1,2,3,4. So we need to have five states each for value and output is made 1 if we reach state 'a', reminder 0 state.

State Definition:

State a : Reminder 0
State b : Reminder 1
State c : Reminder 2
State d : Reminder 3
State e : Reminder 4

State Diagram:



A23)

State Machine to check whether the two inputs have same value for last 3 samples:

Logic: As there are two inputs, at each state we will have four possible transitions based on the two input combinations. If P=Q=1 or P=Q=0 go to next state, otherwise go back to initial state.

State Definition:

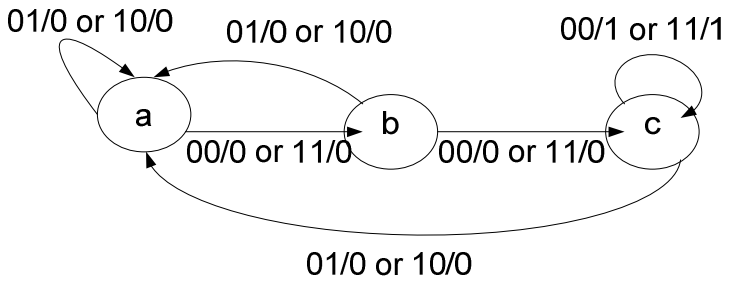
State a : P is not equal to Q

State b : P=Q for last 1 sample

State c : P=Q for Atleast last 2 samples

State Diagram:

Out of the two bits that are shown on the arrows, first 1 is for input A and second one is for input B.



A24)

State Machine with two inputs- Number of 1's together multiples of 4:

Logic: As there are two inputs, at each state we will have four possible transitions based on the two input combinations. Just count the number of 1's in X and Y together and check for the reminder if that number is divided by 4. The possible reminders are 0,1,2,3. The output will be 1 if the reminder is 0.

State Definition:

State a : Reminder 0

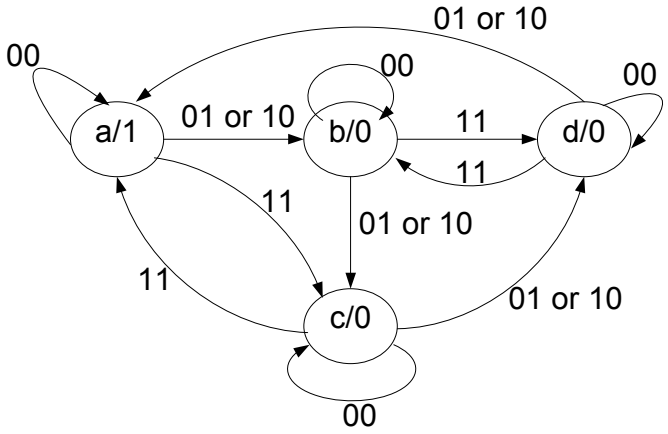
State b : Reminder 1

State c : Reminder 2

State d : Reminder 3

State Diagram:

Out of the two bits that are shown on the arrows, first 1 is for input A and second one is for input B.



A25)

State Machine with two inputs:

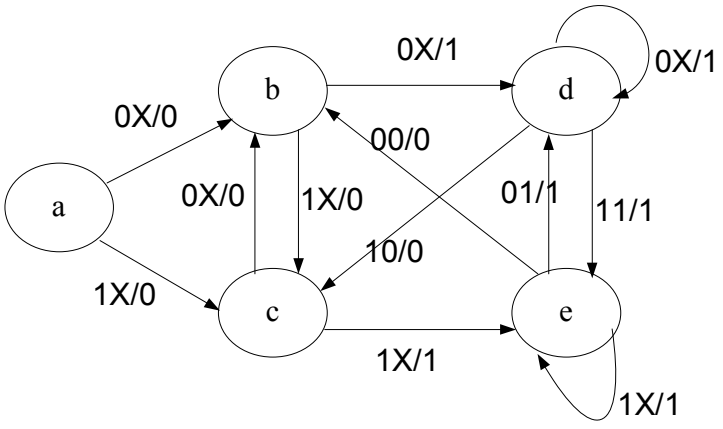
Logic: As there are two inputs, at each state we will have four possible transitions based on the two input combinations. The output will be one for the first time, if A has same value in the last two clock ticks. After that, output can be made 1 either by condition A or condition B. S for initial transitions B will be don't care. But once output is 1, that if you are in state d or e, as long as B is 1, the transition will be between d and e only and the output is also 1.

State Definition:

State a : Initial state
State b : A is 0 once (B can be either 1 or 0)
State c : A is 1 once (B can be either 1 or 0)
State d : A has same value for last two samples and it is equal to logic 0
State e : A has same value for last two samples and it is equal to logic 1

State Diagram:

Out of the two bits that are shown on the arrows, first 1 is for input A and second one is for input B. X indicates don't care, that is the input can be 1 or 0.



A26) a

Chapter 7: Setup time and Hold time

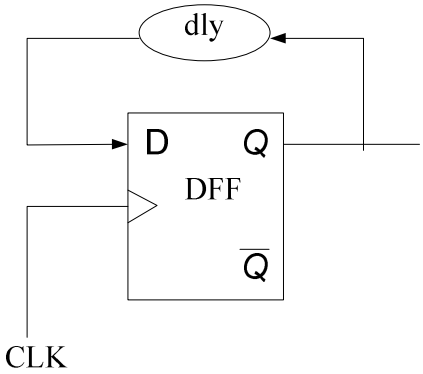
Questions:

Q1) Define (a) setup time (b) hold time (c) clock to Q delay.

Q2) Which of the following flip flops can work at maximum frequency?

	FF1	**FF2**	**FF3**
Clock to Q delay(ns)	5	6	8
Setup time(ns)	3	4	2
Hold time(ns)	2	1	1

Q3) Derive the maximum frequency of operation for the following circuit in terms of Tcq, Tsu and Th of the flip flop?



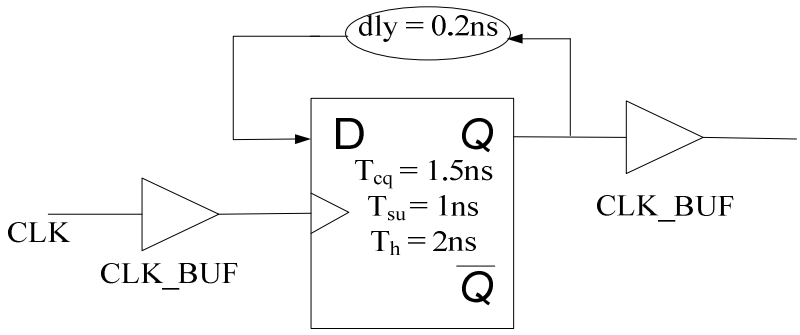
Q4) For the above configuration with dly = 0, which of the flip flops that are shown in Q2, can be used if the available clock period is (a) 5ns (b) 8ns (c) 15ns

Q5) Design a circuit for clock frequency divided by 2 using DFF. Given the following information, find the maximum clock frequency that the circuit can handle?

T_setup = 6ns , T_hold = 2ns and T_propagation = 10ns

Q6) Is there any hold violation in the above circuit? When will the hold violation occur in a given circuit and how can it be solved in circuit level? Describe in detail.

Q7)

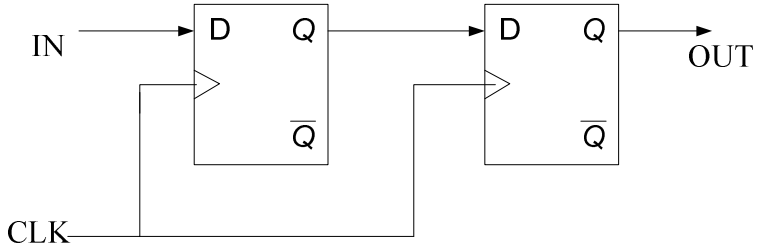


(a) Will the above circuit work without any violation?
(b) Calculate the new value of "dly" to avoid the violation, if any?
(c) Will the delay of CLK_BUF affect the maximum frequency of operation?

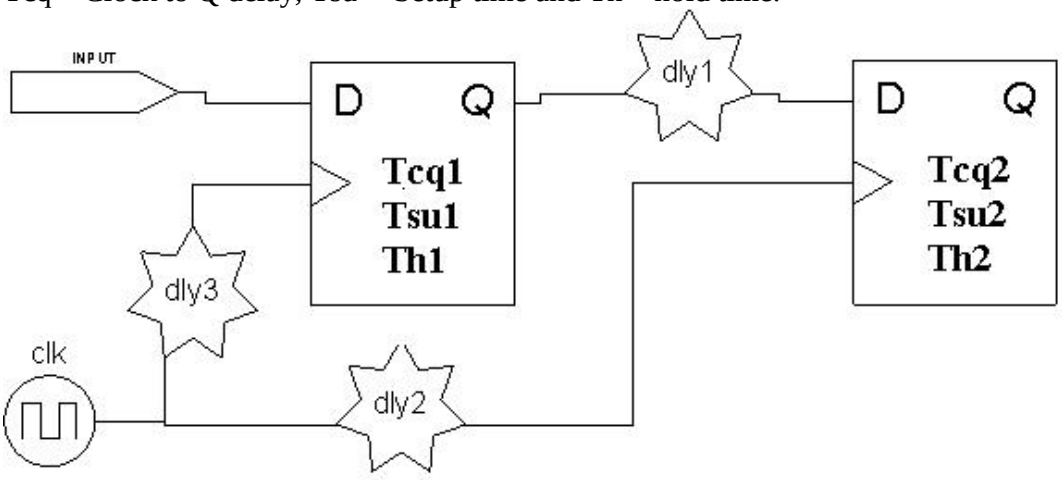
Q8) What is clock skew? Explain.

Q9) Can hold time be negative? Explain.

Q10) Among the flip flops that are shown in Q2, which combination can give maximum frequency of operation for the following circuit?

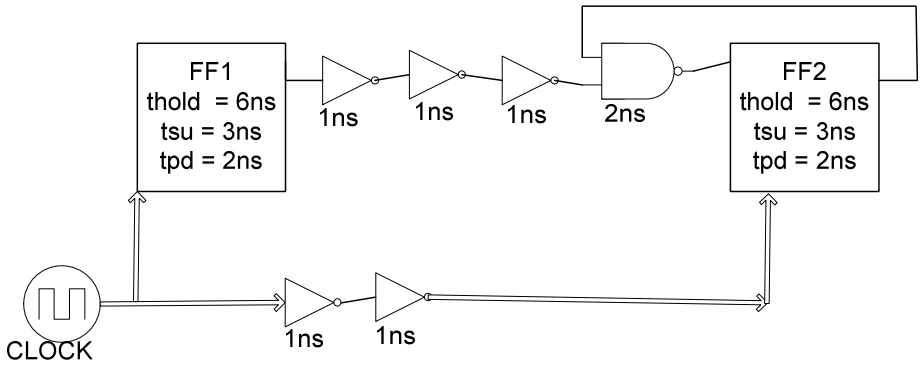


Q11) The following digital circuit shows two flops with a logic delay (dly1) in between and two clock buffer delays (dly2, dly3). Derive the conditions in terms of (dly1,dly2,dly3) to fix setup and hold timing violations at the input of second FF?
Tcq – Clock to Q delay, Tsu -- Setup time and Th – hold time.

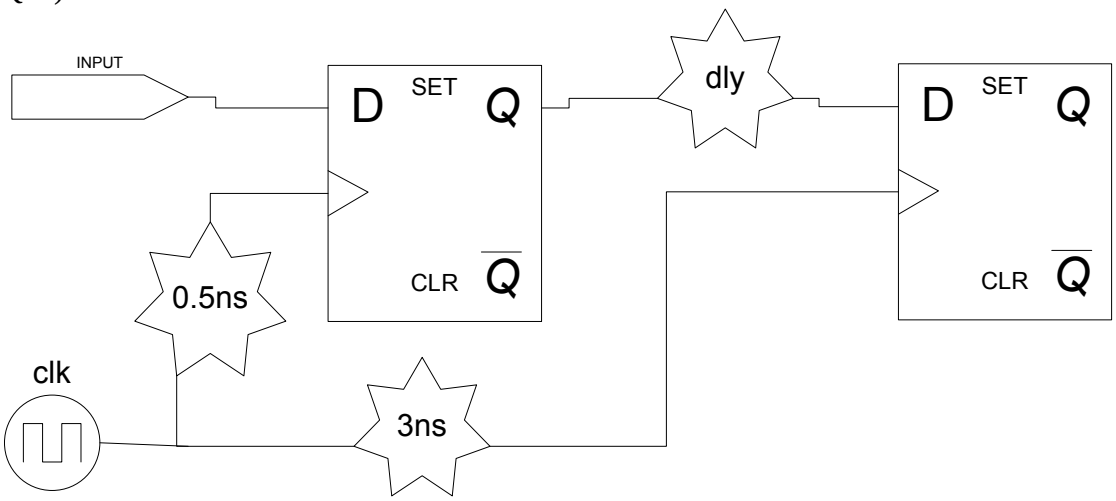


Q12)

(a) Are there any hold time violations for FF2 in the following circuit? If yes, how do you modify the circuit to avoid them?
(b) For the Circuit, what is the Maximum Frequency of Operation?

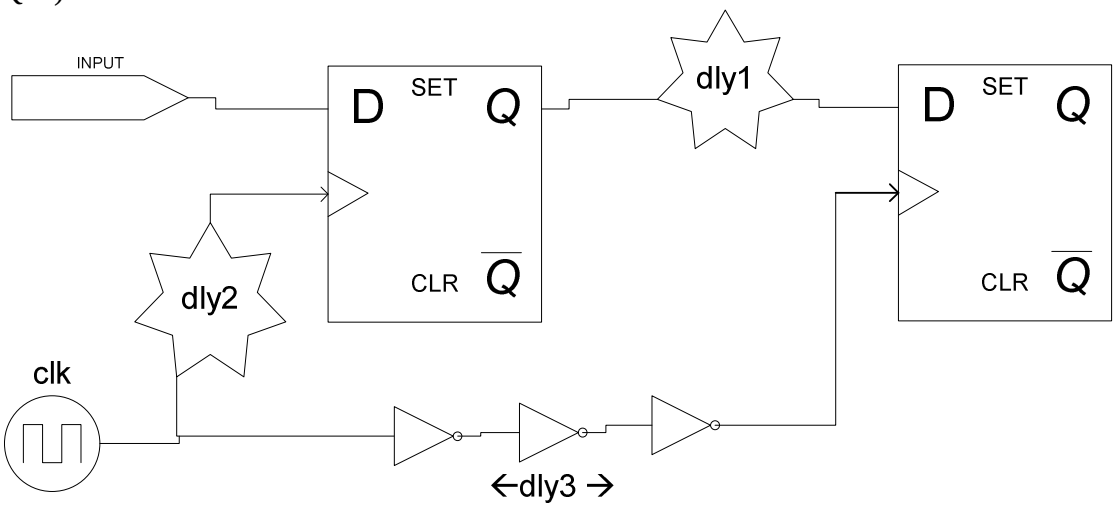


Q13)



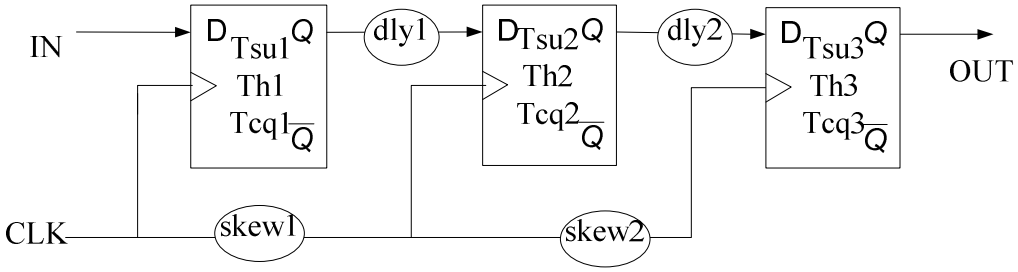
If both the flip flops have same clock to Q delay of 2.5ns, setup time of 2ns and a hold time of 1ns, what is the maximum frequency of operation for the circuit shown in the above figure?

Q14)



Repeat Q1 for the above circuit? Assume Tcq1 – Clock to Q delay, Tsu1 -- Setup time and Th1 – hold time for first FF and similarly Tcq2,Tsu2,Th2 for second FF.

Q15) What is the maximum frequency of operation for the following configuration?



Q16) What is metastability? When/why it will occur? Explain how to avoid this?

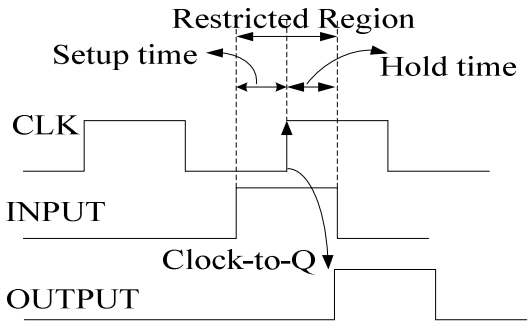
Q17) For the circuit in Q11, two identical flip flops with the following data were used: Tsu = 2ns, Th = -3ns and Tcq = 5ns. Which combination of dly1 and dly2 from the following table will give maximum frequency of operation without any violations? Given: dly3 = 0.

dly1(ns)	**dly2(ns)**
1	7
6	2
2	8

Answers:

A1)

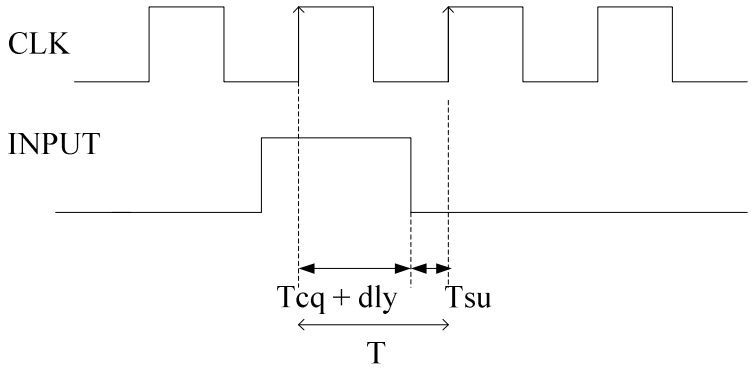
(a) Setup time: Setup time is the minimum amount of time the data signal should be held steady before the clock event so that the data is reliably sampled by the clock.
(b) Hold time: The hold time is the minimum amount of time the data signal should be held steady after the clock event so that the data is reliably sampled by the clock.
(c) Clock to Q delay: The clock to Q delay is the amount of the propagation time required for the data signal to reach the output (Q) of the flip flop after the clock event.



A2) For a single flip flop, lesser the clock-to-Q delay, more the operating frequency. However, the maximum frequency of operation may be limited by the configuration in which the flip flop is connected. This will be clear in the later parts of the chapter.

Among the 3 flops, the first one, FF1 has less clock to Q delay. So it can operate at maximum frequency which is given by 1/5ns = 200MHz

A3) After the posedge of the clock, the output will change after a delay of Tcq. The input of the flop will change after further delay of "dly". It should be available before the Tsu of the flop. So the T >= Tcq + Tsu + dly. The same thing is illustrated in the following waveform.



A4) dly = 0

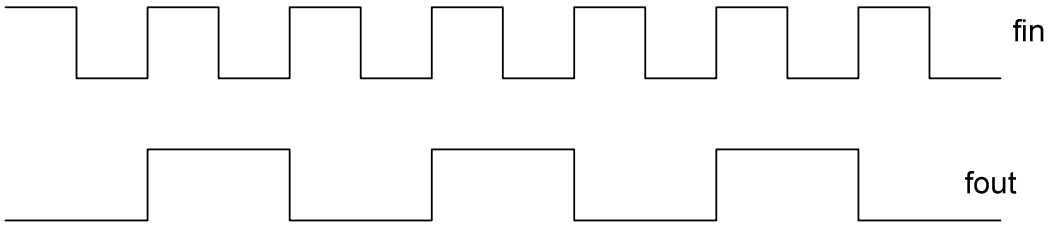
For FF1, Tsu + Tcq = 3 + 5 = 8ns
For FF2, Tsu + Tcq = 6 + 4 = 10ns
For FF3, Tsu + Tcq = 8 + 2 = 10ns

As dly = 0, Tsu + Tcq <= T

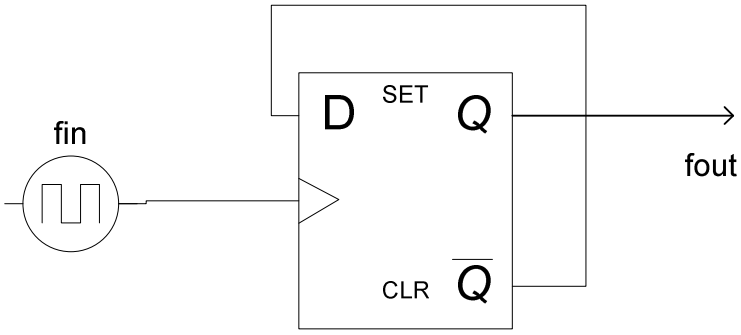
(a) T = 5ns, None of the flip flops has Tsu + Tcq <= T, so no one can be used.
(b) T = 8ns, FF1 can be used
(c) T = 15ns, Anyone can be used

A5)

Waveforms:



Design:



Using the same equation , T >= Tcq + Tsu, T >= 6 + 10. So T >= 16ns.
The maximum clock frequency = 1/16ns = 62.5MHz

A6) There are no hold violations in the above circuit. If the hold time is greater than the propagation delay then there will be hold violation for the above circuit. In that case, buffers (even number of inverters) will be used in the feedback path in order to delay the signal in reaching back to the input.

A7)

(a) Thold <= Tcq + dly. But here, 2ns > 1.5 + 0.5 = 1.7ns
So there is a hold violation in the above circuit.
(b) dly >= Thold – Tcq = 2 – 1.5 = 0.5ns
(c) The delay of the clock buffer will not effect the maximum frequency of operation of the circuit.

A8) Clock-skew: Clock skew is a phenomenon in synchronous circuits in which the clock signal (sent from the clock circuit) arrives at different components at different times.
This is typically due to two causes:

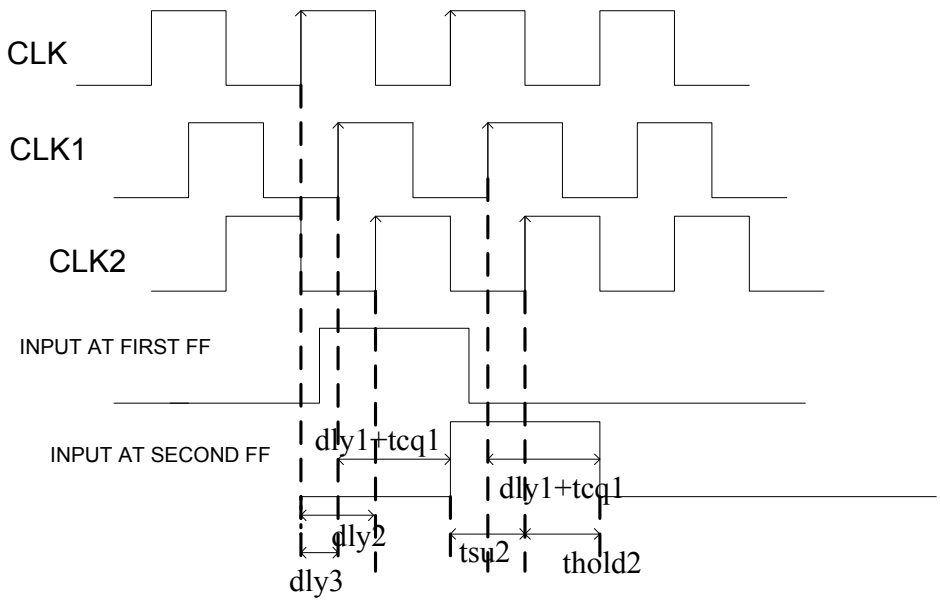
1. The first is a material flaw, which causes a signal to travel faster or slower than expected.
2. The second is distance: if the signal has to travel the entire length of a circuit, it will likely (depending on the circuit's size) arrive at different parts of the circuit at different times.

Skew is only meaningful between adjacent pairs of registers, not between any pair of registers in a clock domain.

A9) Yes, Hold time of a flip flop can be negative. Most of the modern flip flops will have either 0 or negative hold time. Assume Thold = -2ns, there should not be any transitions in the input before 2ns of the clock event.

A10) For the given circuit, T >= Tcq1 + Tsu2.
To get maximum frequency T should be less. So we should select the first flop with less clock to Q delay and second flip flop with less setup time. So FF1 and FF3 give the maximum frequency and it is equal to 1/7ns = 142.8MHz

A11)



The above waveforms show the CLK, CLK1 and CLK2. The input waveform at FF1 is assumed and the input of FF2 is shown accordingly with all the given delays and clock-to-Q delays.
From the waveforms it is clear that, to avoid setup time violation,

T >= (Tsu2 + Tcq1 + dly1 – delta) where **delta = dly2-dly3** (assuming +ve skew)

From this equation we can get maximum freq of operation.

To avoid hold time violation,

Th2 <= Tcq1 + dly1 – delta

These two equations can be used as generalized equations to solve setup time/hold time problems. This works only for synch circuits. If one clock works at pos edge and other is negative edge we need to derive one more set of equations. That also we will at later section.

A12)
(a) There is a hold time violation in the circuit, because of the feedback. Tcq2 +AND gate delay is less than thold2. To avoid this , we need to use even number of inverters(buffers). Here we need to use 2 inverters each with a delay of 1ns. Then the hold time value exactly meets.

(b) In this diagram,

dly3 = 0
dly2 = 2ns
so,delta = 2ns
tsu2 = 3ns, tcq1 = 2ns, dly1 = 5ns

Putting all these values in Eq(1) ,
T >= Tcq1 + dly1 + Tsu2 - delta

so, T >= 2 + 5 + 3 - 2, T >= 8ns, f <= 1/8
Max freq of operation is 125MHz

A13)
Tcq1 = Tcq2 = 2.5ns
Tsu1 = Tsu2 = 2ns
Thold1 = Thold2 = 1ns
delta = clock_skew = 3 - 0.5 = 2.5ns

Equation for hold-violation is,
Thold <= dly + Tcq1 - delta (Eq(2))
1 <= dly + 2.5 - 2.5
So dly >= 1ns
To obtain maximum freq, fixing it to lowest possible value, so dly = 1ns

Using this value and Eqauation (1) of setup time, we can obtain max freq.
T >= Tcq1 + dly +Tsu2 - delta
T >= 2.5 + 1 + 2 - 2.5
So T >= 3ns

Max freq of operation = 1/3ns = 333.33 MHz

A14)
Setup time:
(T/2) + delta >= Tcq1 + dly1 + Tsu2

Hold time:
Th2 <= delta + Tcq1 + dly1

where delta = dly3 - dly2, assuming positive skew
and T is clock period.

Note: The procedure is same as that of Q11.Just draw the waveforms with proper delays, you will get above equations.

A15) For FF1 and FF2, T1 >= (Tsu2 + Tcq1 + dly1 – skew1)
For FF2 and FF3, T2 >= (Tsu3 + Tcq2 + dly2 – skew2)
T >= MAX (T1,T2)

A16)
Metastable state: A un-known state in between the two known logical states is called as Metastable state.
Reason for occurrence: This will happen if the output node capacitance is not allowed to charge/discharge fully to the required logical levels. In case of flip flops, if the input changes in the restricted region, that is if there is a setup time or hold time violations, metastability will occur.

Way to avoid: To avoid this, a series of FFs is used (normally 2 or 3) which will remove the intermediate states. The extra flip flop is called the **synchronizer.**

A17) Given: Tsu = 2ns, Th = -3ns and Tcq = 5ns.
If hold time is negative and if its absolute value is less than Tsu, only thesetup violation equation without any modification will work. But if absolute value of hold time is more than setup time, we need to replace the setup time in the equation with hold time. The modified equation is shown below:

T >= Tcq1 + dly1 + Max(Tsu2, | Th2 |)
T >= 5 + dly1 + 3
T >= 8 + dly1

To get maximum frequency of operation, the minimum possible dly1 = 1ns.
So, T >= 9ns

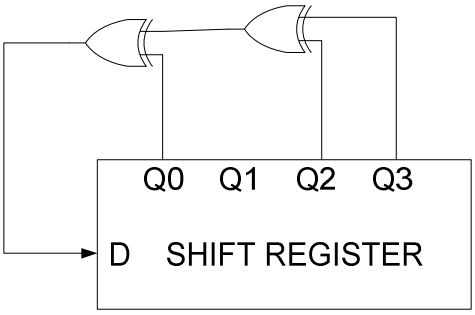
Fmax = 1/9ns = 111MHz

Chapter 8: Counters and Shift Registers

Questions:

Q1) Design a 3-bit shift register using 2:1 Mux and D Flip Flops which shifts right if the control input, C = 0 and shifts left if C = 1?

Q2)



A four bit shift register, which shifts data right Q0 → Q1 → Q2 → Q3, is shown in the above figure. The initial value for Q0, Q1, Q2, Q3 is 1000.
(a) Write the 4 bit values for Q0, Q1, Q2 and Q3 after each clock pulse until the pattern 1000 reappears.
(b) To what values should the shift register be initialized so that the pattern 1000 occurs after 2 clock pulses?

Q3) What is the difference between a ripple counter and a synchronous counter?

Q4) To count from 0 to N-1, how many flip-flops are needed?

Q5) Design a 4-bit binary counter using TFFs?

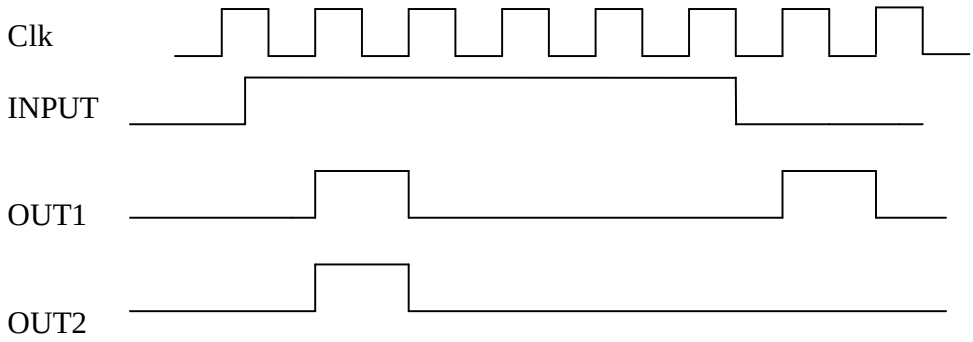
Q6) Give the FSM for a 3-bit gray code counter?

Q7) Design a synchronous sequential circuit using minimum number of DFFs and a 4:1 Mux to check the highest number of ones and zeros in the last 3 input samples. The design should give 1 at the output if the last 3 samples at the input has more number of 1's similarly than the number of zeros. Only one clock is available and the input is sampled at clock rate only.

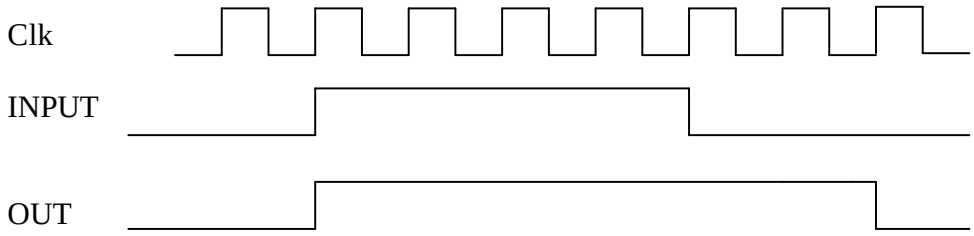
Eg:

IN : 001110110000
OUT: 0111111000

Q8) Obtain OUT1 & OUT2 from INPUT shown below? (Hint: You need a synchronizer to align INPUT with clock)

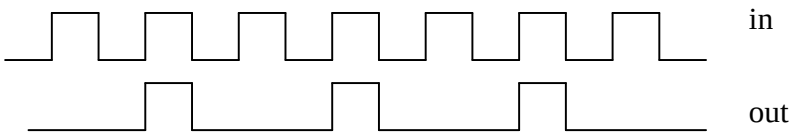


Q9) Give the circuit to extend the falling edge of the input by 2 clock pulses. The waveforms are shown in the following figure.



Q10) Design a frequency divide-by-2 circuit using DFF and external gates which gives (a) 50% duty cycle (b) 25% duty cycle?

Q11)



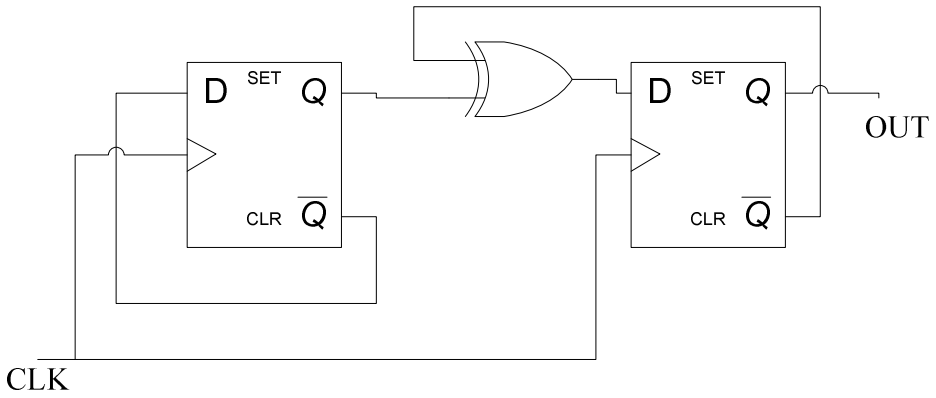
Design a sequential circuit which cuts the every second pulse from the input (clock) as shown above.

Q12) Design frequency divide-by-3 circuit using DFFs and external gates which gives a duty cycle of 1/3rd?

Q13) How do you change the above design to get (a) 66.67% duty cycle (b) 50% duty cycle?

Q14) Design the Digital Circuit which gives $f_{out} = (2/3) f_{in}$

Q15)



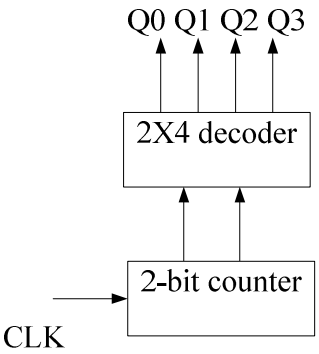
In the above circuit what is the relation between input "CLK" and the output?

Q16) Here is an interesting design question. There is a room which has two doors one to enter and another to leave. There is a sensor in the corridor at the entrance and also at the exit. There is a bulb in the room which should turn off when there is no one inside the room. So imagine a black box with the inputs as the outputs of sensors. What should the black box be?

Q17) Design a BCD counter which counts from 0 to 9999, using BCD decade counter as black box?

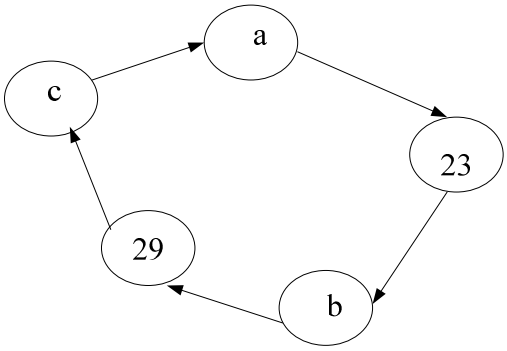
Q18) What is ring counter? Implement a ring counter using shift register?

Q19) Ring counter implementation using 2-bit counter and 2:4 decoder is shown in the following diagram. Draw the output time signals, Q0,Q1,Q2 and Q3 with respect to the clock.



Q20) To generate 8 timing signals using a ring counter (similar to the circuit that is shown in Q19), mention the required size of decoder and the size of the counter?

Q21) The following FSM shows the zero circulating ring counter. Predict the values of the missing states?



Q22) What are the unused states in a 3-bit Johnson counter?

Q23) What is the length of counting sequence for a Johnson counter with N flip-flops?

Q24) How many unused states will be there in a Johnson's counter with N flip-flops?

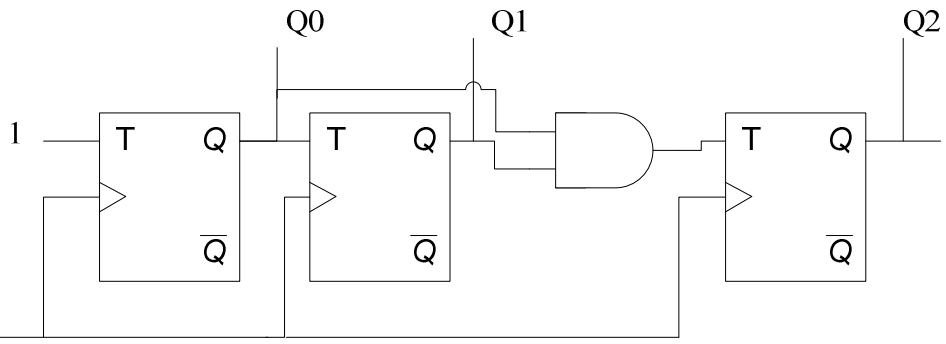
Q25) What is the output frequency of a 4-bit binary counter for an input clock of 160 MHz?

Q26) If each flip flop has a clock-to-Q delay of 10ns, how much time will it take for output to change to its next state in case of (a) 4-bit Ripple Counter (b) 4-bit Synchronous counter?

Q27) How fast can a 11 stage ripple counter be clocked, assuming worst case clock to Q delay of 40ns (of each stage) and extra gate delays of 60ns?

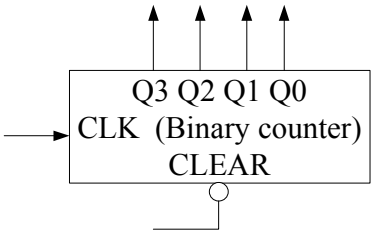
Q28) Design a counter using DFF that counts in the sequence: 0,4,2,7,0,4,2,7,0,4……?

Q29) What is the functionality of the following circuit:

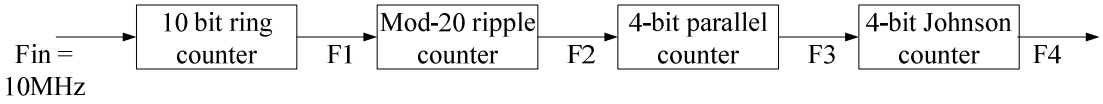


Q30) Using external gates and 4-bit counter, design a circuit which gives '1' if the number of clocks are multiples of 4.

Q31) Design a sequential circuit that produces a logic 1 at the output when the input has been 1 for eight or more consecutive clock pulses using a counter(shown below) and minimum number of basic gates.



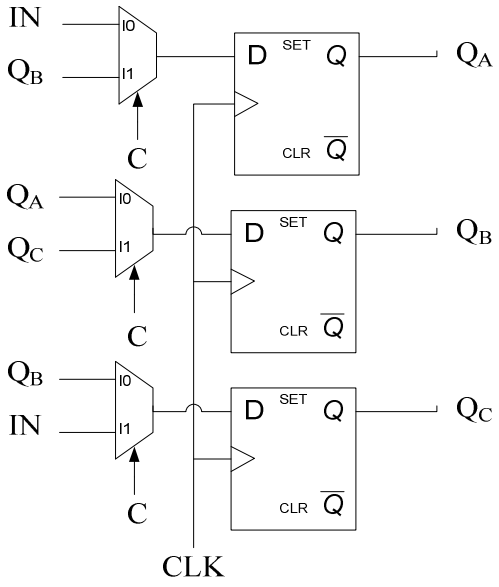
Q32) What are the frequencies F1, F2, F3 and F4 after each stage if an input clock of 10MHz is applied?



Q33) How to swap the contents of two 8-bit registers without using a third register.

Answers:

A1) The shift register is shown below.
If C = 0, the circuit shifts from IN → Q_A → Q_B → Q_C and
If C = 1, the circuit shifts from IN → Q_C → Q_B → Q_A



A2) D_{next} = Q0 xor Q2 xor Q3

(a)

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

(b) Q0 = 0, Q1 = 0, Q2 = 1, Q3 = 1

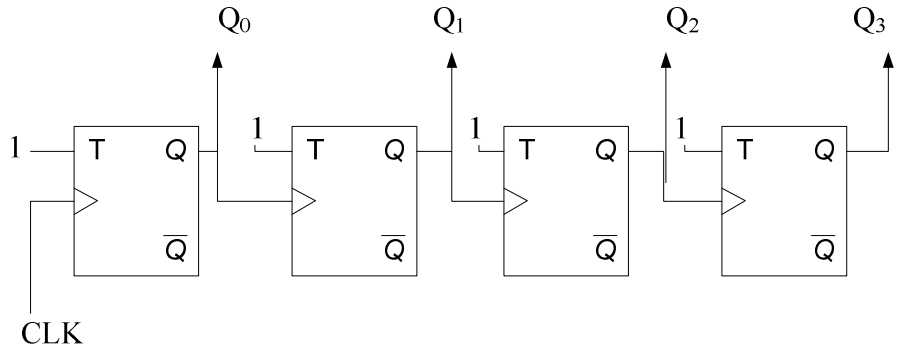
A3) Ripple counter is asynchronous. This means all flip flop outputs will not change at the same time. The output of one flop works as clock to the next flip flop. The state changes consequently “ripple through” the flip flops, requiring a time proportional to the length of the counter.

Where as synchronous counters will have same clock for all the flip flops. All flip flops will change the state at the same time. Design of synchronous counters is easy but needs more hardware.

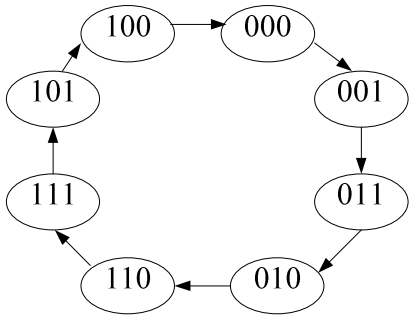
Although the asynchronous counter is easier to implement, it is more "dangerous" than the synchronous counter. In a complex system, there are many state changes on each clock edge, and some IC's (integrated circuits) respond faster than others. If an external event is allowed to affect a system whenever it occurs, a small percentage of the time it will occur near a clock transition, after some IC's have responded, but before others have. This intermingling of transitions often causes erroneous operations. What is worse, these problems are difficult to test for and difficult to foresee because of the random time difference between the events.

A4) $\log_2 N$

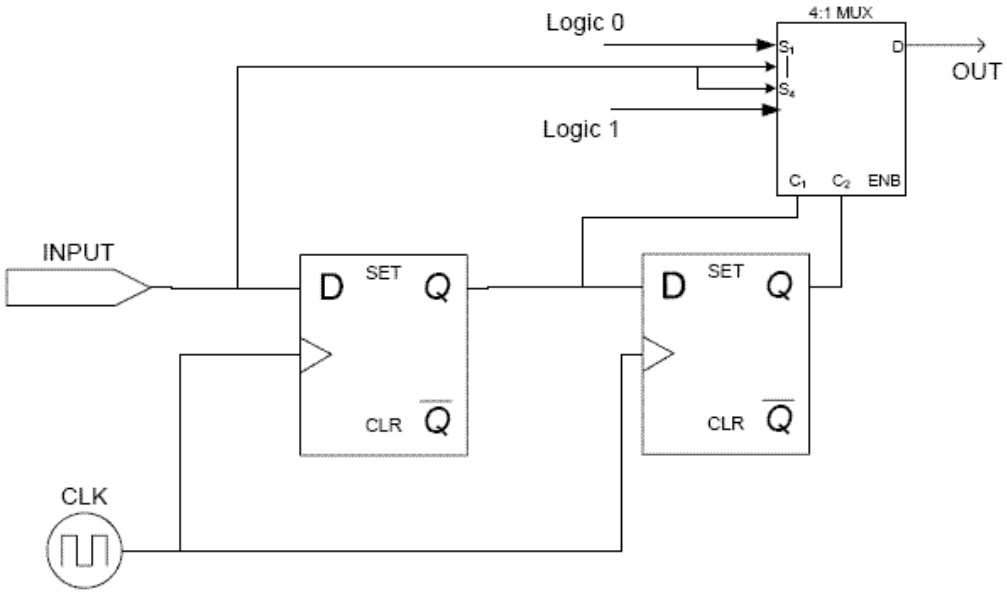
A5) In the problem it is not clearly mentioned whether



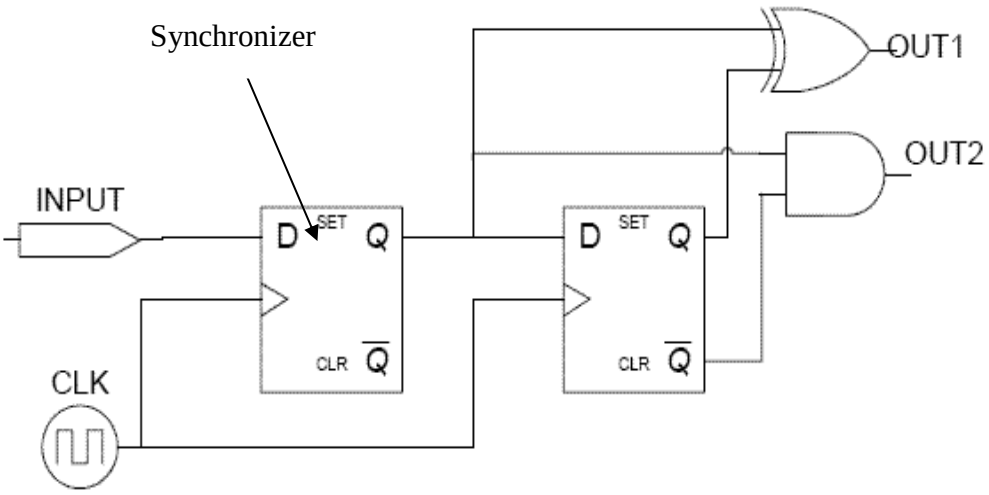
A6) The FSM for 3-bit gray counter is shown below. You can notice the single bit change from one state to another state.



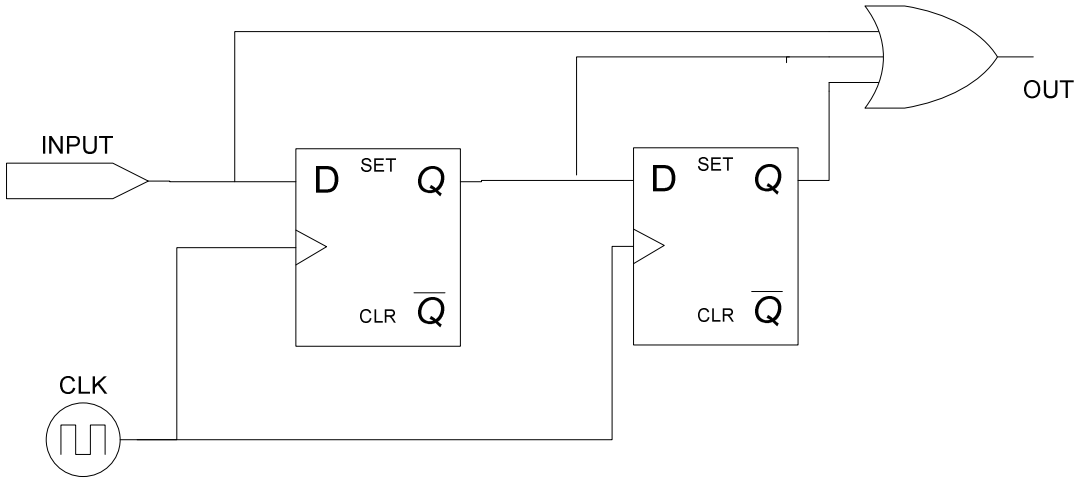
A7) Shift register based: We need current sample, previous sample and previous to previous sample. So a 2-bit shift register is needed. The logic is similar to Majority function (Refer Chapter 4). The following figure shows the design with two DFFs and a 4:1 Mux.



A8) Shift register based: The synchronizer (the first flip flop) aligns the INPUT with clock. The second flip flop delays the input by one clock. Draw the waveforms of output of first and second flip flops and then try to get the relationship between those waveforms and OUT1, OUT2. It gives the complete solution as shown below.



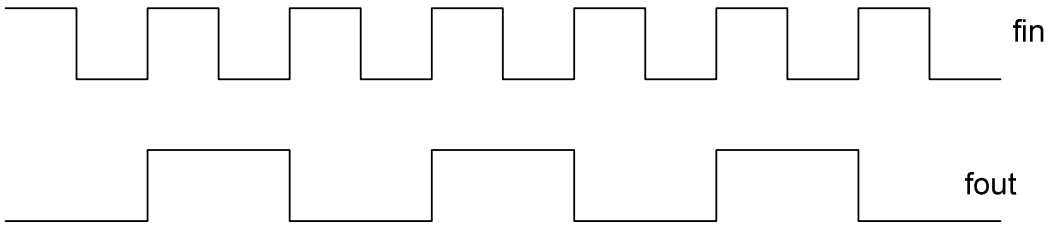
A9) Shift register based: Assumed atleast 3 clock gaps between next falling edge. Shift register of width 2 is needed.



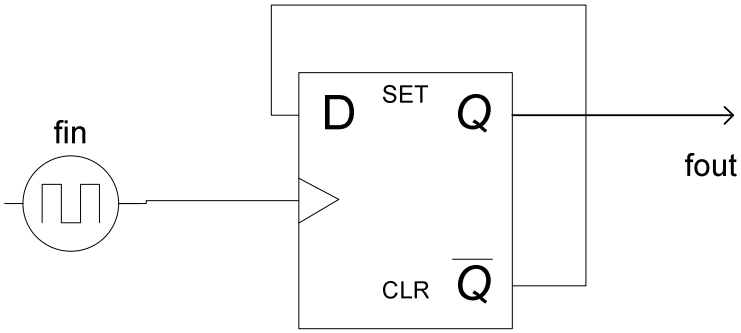
A10)

(a) 50% duty cycle:

Waveforms:



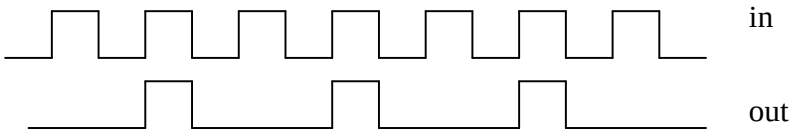
Design:



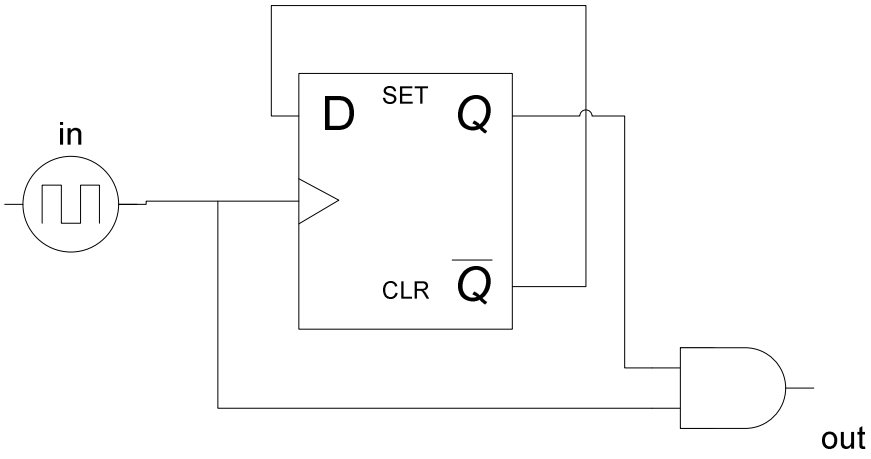
(b) 25% duty cycle:

The above circuit gives 50% duty cycle. To get 25%, we need to use an extra AND gate, which takes fin and fout as the inputs.

Waveforms:



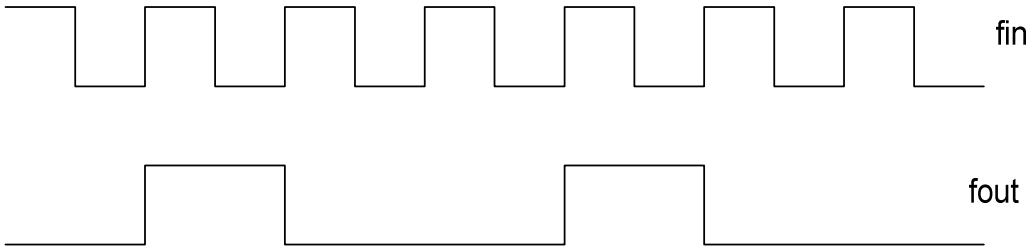
Design:



A11) Same as A10(b)

A12)

Waveforms:



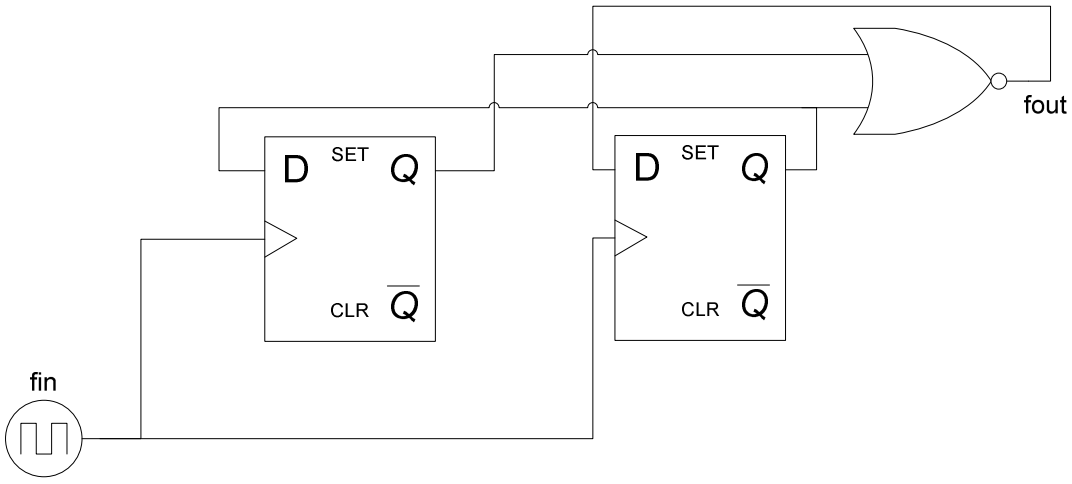
Design:

In the above problem, if you observe the waveforms, they are synchronous. So we can use FSM to design the circuit. If you observe the waveform clearly, output is 100,100,100 and so on. Assume 3 states: a,b & c. Initial state is a and output is 1 in this state. The state transition a→b→c→a. Output is 1 only for state a. The state table is shown below:

PS	NS	O/P
a	b	1
b	c	0
c	a	0

With State assignment: 00-a,01-b and 10-c, we can obtain the following next state equations:

A(t+1) = B, B(t+1) = A NOR B, OUT = A NOR B

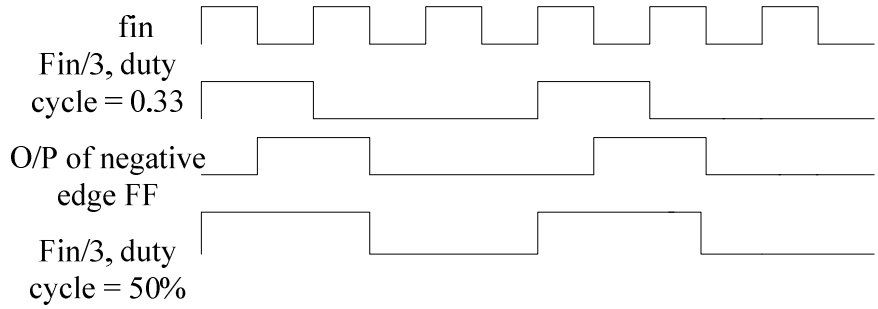


A13)

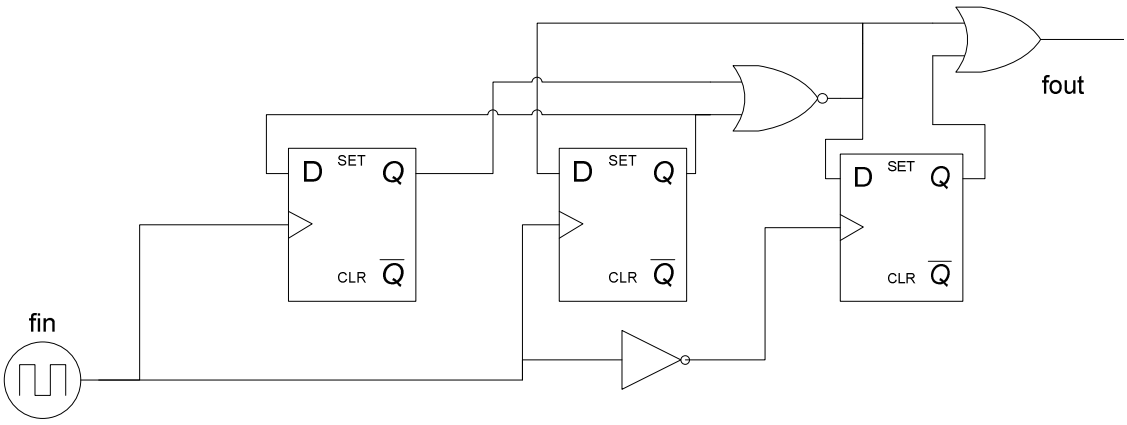
(a) Replacing the NOR gate in the above circuit with NAND gate gives a duty cycle of 2/3rd . That is 66.67%.

(b) To get 50% duty cycle, by observing the waveforms, we can notice that, an extra flop that works at the negative edge of the clock is needed. ORing of the input and output waveforms of this flip flop gives the required waveform. The complete solution is shown below:

(i) *Waveforms:*

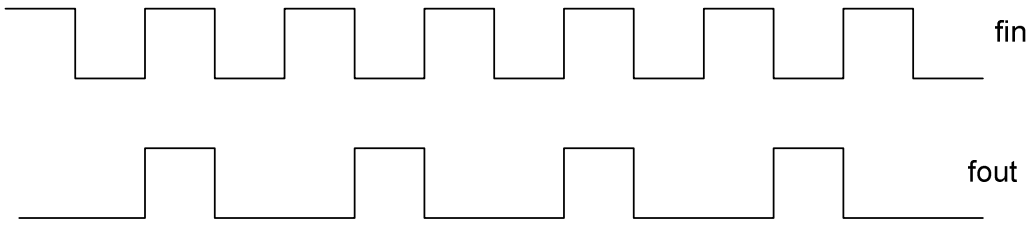


(ii) *Design:*

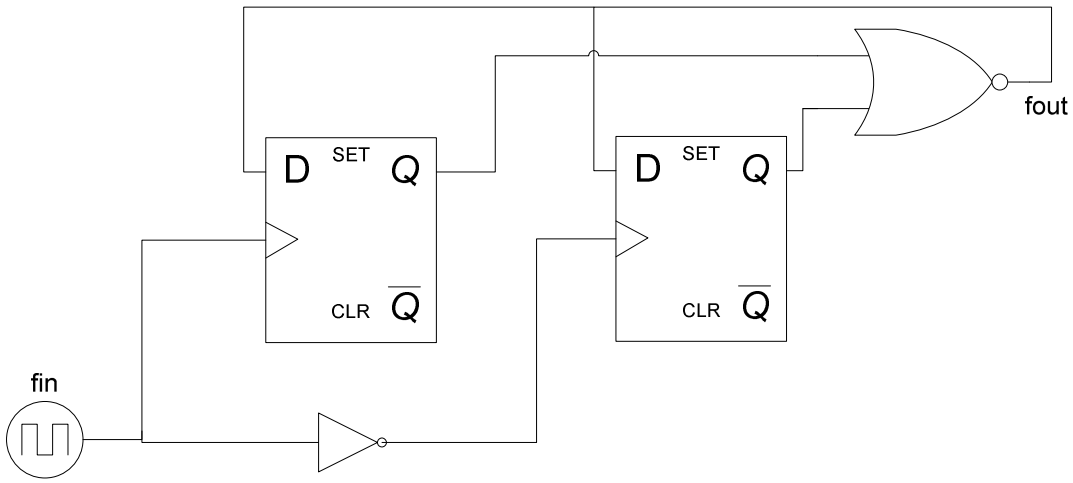


A14)

Waveforms:



Design:



Note: The clue to get the solution is: There is a transition at the falling edge of clock. So the clock to the second flop is inverted one. The waveforms shown in the above figure, **fout** has a duty cycle of 1/3rd To get 2/3rd duty cycle, replace NOR gate with NAND gate in the above design.

A15) From the given circuit we can derive the following next state equations,

A(t+1) = A' and B(t+1) = A XOR B' = AB + A'B'

Taking initial values of A=B=0 and drawing the waveforms with respect to the clock using above equations, we can observe that

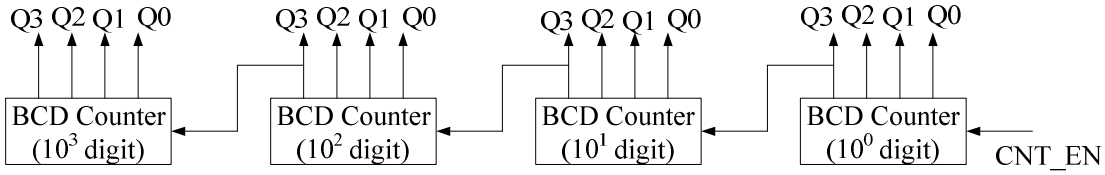
OUT = freq_Clock / 4 with duty cycle = 50%

A16) The block box can be an up-down counter, where the "count_up_enable" is connected to the sensor at the entrance and "count_down_enable" to the sensor at the exit. That is if there is no one in the room, the counter's output will be zero. Whenever this happens make the bulb "OFF".

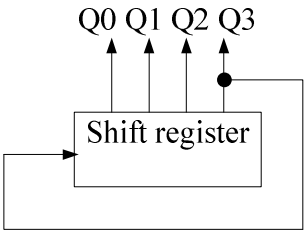
For 200 people, we need 8 bit counter.

So The O/P of entrance sensor will be used as enable for UP count and the other sensor at exit will be used for DOWN count, whenever the counter's O/P is 0, we can make the BULB OFF, Otherwise ON.

A17) Each BCD counter counts from 0-9.

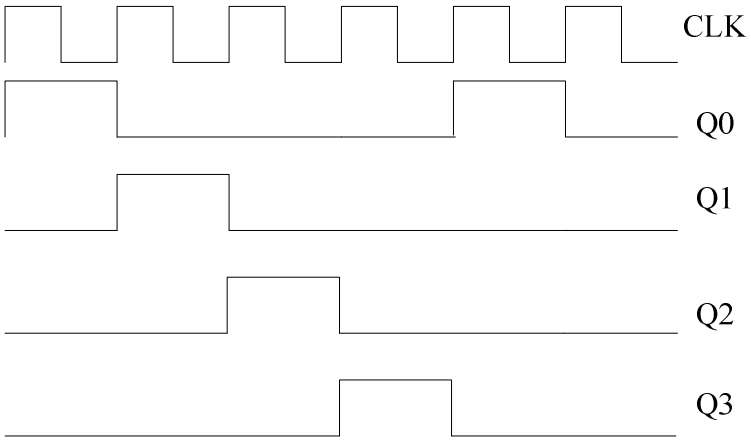


A18) *Ring counter:* A ring counter is a circular shift register with only one flip-flop being set at any particular time, all others are cleared. This single bit is shifted from one flip flop to the next to produce the sequence of timing signals.



The above circuit shows the ring counter. The initial value of the shift register has to be 1000.

A19)



A20) 3-bit counter and 3:8 decoder

A21) In "0" passing ring counter, at any time only one flip flop will be set to 0 others will be 1. The given state values are 23 and 29. The binary representations are 10111 and 11101 respectively. So the states are : 01111, 10111,11011,11101,11110. The decimal values are : 15,23,27,29,30.

a = 15, b = 27 and c=30

A22) The states of 3-bit Johnson counter are: 000,100,110,111,011,001. So the unused states are 010 and 101

A23) 2N

A24) For N-flops, the total possible states = 2^N.
The number of states of a Johnson counter = 2N
So, the number of unused states = 2^N – 2N

A25) The output of last flip flop of a 4-bit counter is equal to the input clock/16.
So output frequency = 160MHz/16 = 10MHz

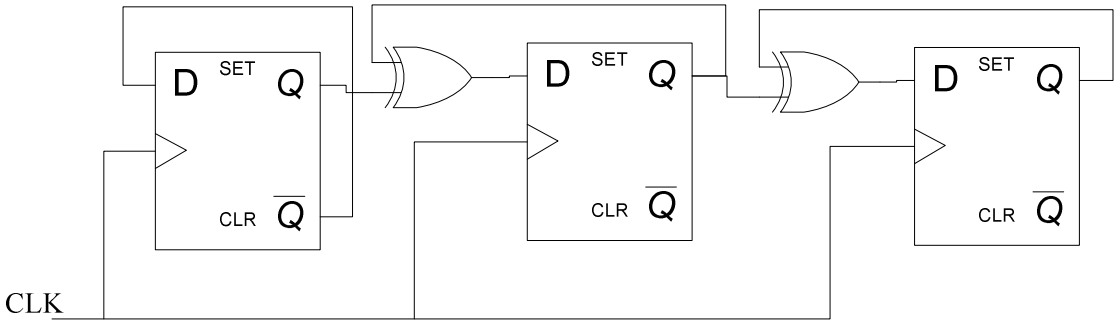
A26)
(a) 10 + 10 + 10 + 10 = 40ns
(b) 10ns

A27) Minimum time period of the clock = 11 x 40 + 60 = 440+60 = 500 ns
So maximum clock frequency = 1/500 = 2 MHz

A28) The present state and next state values are shown in the table and the complete design is shown in the following diagram.

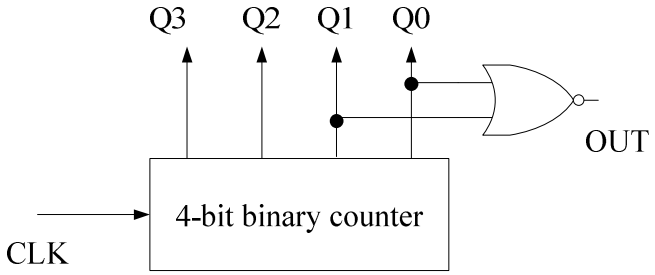
Present state			**Next state**		
A	**B**	**C**	**A**	**B**	**C**
0	0	0	1	0	0
1	0	0	0	1	0
0	1	0	1	1	1
1	1	1	0	0	0

Using this the following next state equations can be derived.
A(t+1) = A'
B(t+1) = A XOR B
C(t+1) = B XOR C

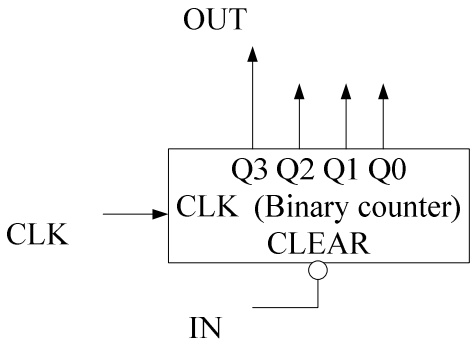


A29) Let us name the 3 flip flops as A,B and C
Q0 = A'
Q1 = A XOR B
Q2 = (AB) XOR C
Starting with A=B=C=0, the next states can be obtained as:
000 → 001 → 010 → 011 → 100 → 101 → 110 → 111
So the circuit works as 3-bit binary counter.

A30) The output should be asserted 1 if the number of clocks is multiple of 4, that is 0, 4 , 8 and 12. The K-Map simplification gives, OUT = Q1'Q0' = (Q1 + Q0)'



A31)

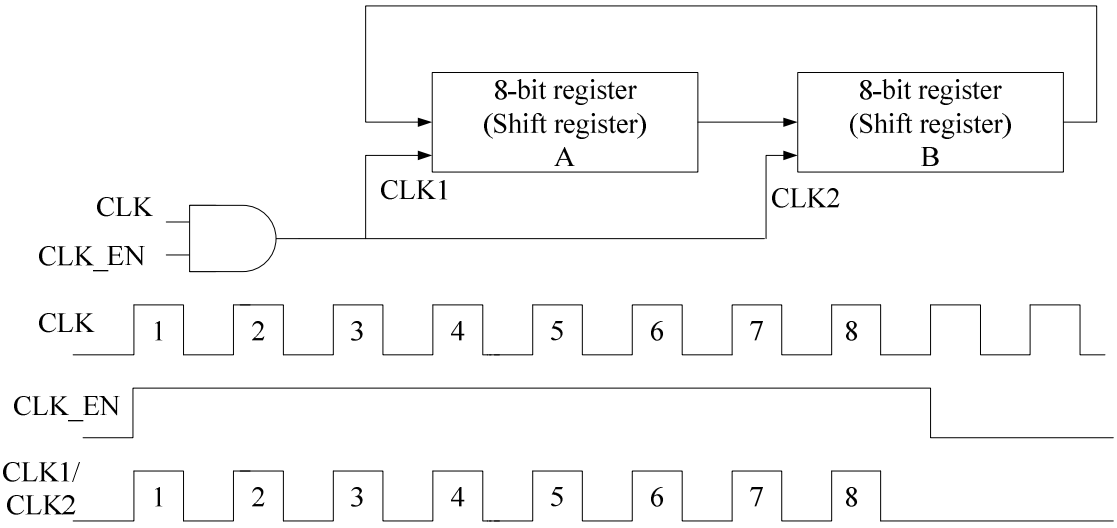


Limitation of the design:
As the design used only one counter, the maximum count is 15.

A32) N-bit ring counter gives 1/N times the input frequency at the output. Johnson's output is 1/2N times the input. Where as the counters output will be 1/(2^N).

F1 = 10MHz / 10 = 1MHz
F2 = 1MHz / 20 = 50KHz
F3 = 50KHz / 16 = 3.125KHz
F4 = 3.125KHz / 8 = 390.625Hz

A33) The complete design using shift registers is shown in the following figure. The main clock is gated with the clock enable so that A and B will be shifted just 8 clocks. After 8 clocks A and B will have their contents swapped.


8-bit register
(Shift register)
A
8-bit register
(Shift register)
B
CLK1
CLK2
CLK
CLK_EN
CLK
1
2
3
4
5
6
7
8
CLK_EN
CLK1/
CLK2
1
2
3
4
5
6
7
8

Chapter 9: Fault Analysis and Hazards

Questions:

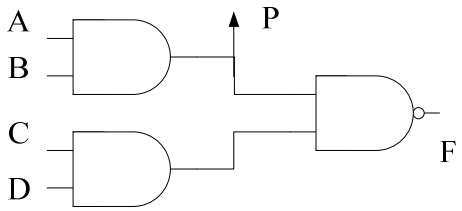
Q1) What are stuck-at problems? Explain the reason for their occurrence?

Q2) How many number of stuck at problems are possible for a 2 input AND Gate? Which of those faults are not testable?

Q3) Define : (a) Test Pattern/Test set (b) ATPG

Q4) Explain the procedure for detecting a specific fault in a given circuit?

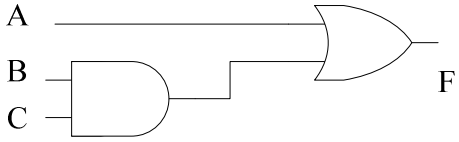
Q5) To detect the Stuck at Zero problem at marked point 'P' in the following diagram, which of the input combinations can be used?



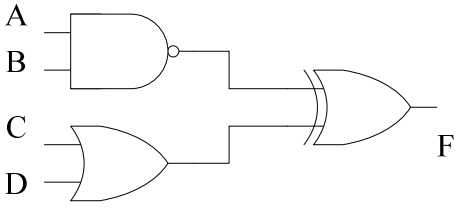
Q6) Give the test patterns that are needed to verify all the stuck at problems of a
(a) 2-Input NAND Gate (b) 3-input NAND Gate

Q7) How many minimum number of test vectors are needed to verify all the stuck at problems of a N-input logical gate?

Q8) Give the complete test set for the following circuit:

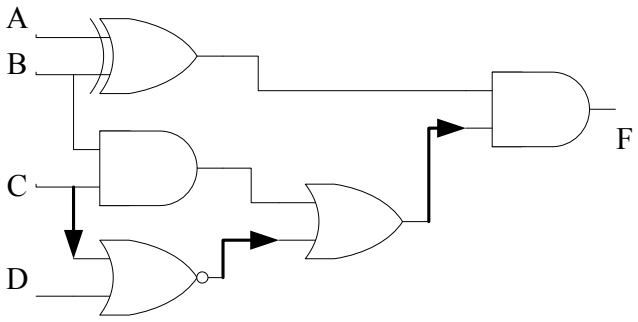


Q9) What is the output F, in the following circuit, if there is
(a) A stuck-at-0 problem at node A
(b) A stuck-at-1 problem at the output of OR gate



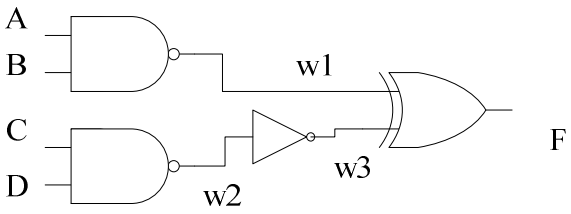
Q10) What do you mean by “Path Sensitized Tests” in testing of logic circuits?

Q11) Which input pattern can be applied to make the path from input C to the output F (via NOR gate – OR gate – AND gate), sensitized, in the following diagram?



Q12) (a) List all the single faults that can be detected in the following circuit, using the test set “0100”?

(b) Show the tests that can detect stuck-at-0 and stuck-at-1 faults at node w3?



Q13) (a) Give the AOI of a 2:1 Mux?

(b) Show all the paths that are possible from inputs to the output?

(c) Give the complete test set for the same using the Path sensitized test?

Q14) (a) Give the circuit for a 4-bit parity generator?

(b) Derive the minimal test set that can detect all stuck-at-faults?

Q15) What is D-notation?

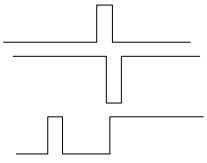
Q16) What are the two hazards that can be there in a combinational circuit?

Q17) Match the following:

(a) Static-one hazard (i)

(b) Static-zero hazard (ii)

(c) Dynamic hazard (iii)



Q18) Give the characteristics of Static-zero Hazard?

Q19) Show the equivalent circuit for Static-one Hazard?

Q20) How many single variable change static-0 hazards the Boolean function, G = AB + A' C + B'C'D has?

Q21) How to avoid static hazards in a given circuit(single variable change hazards)?

Q22) Y = AB + A'C + BC where A,B and C are input Boolean variables. The output Y,

a) Glitches and can be reduced
b) Will not glitch and can be reduced
c) Glitches and cannot be reduced
d) Will not glitch and cannot be reduced.

Q23) Find a hazard-free minimum cost implementation of the function:
$F(A,B,C,D) = \sum (0,4,11,13,15) + d(2,3,5,10)$

Q24) Design 4-input XOR gate using 2 input XOR gates in all possible ways? Discuss the advantages and disadvantages of each?

Q25) Implement the Boolean function Y = AB + CD with the following design constraints:

i) A and B arrive at t = 0ns
ii) C arrives at t = 1ns
iii) D arrives at t = 2ns
iv) All logic gates, mux, adders have constant delay of 1ns.
v) Output should be available at 3ns.

Answers:

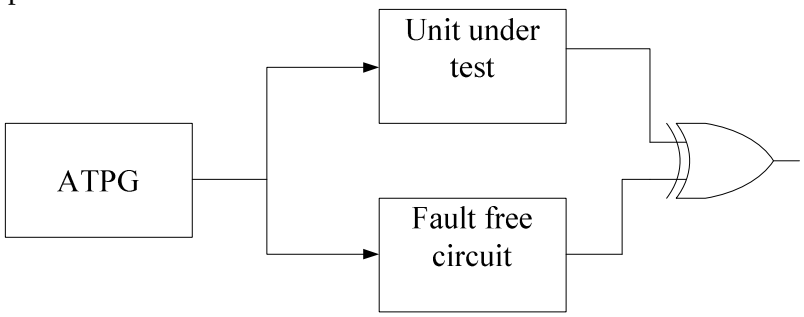
A1) A fault in a manufactured circuit causing a node to be stuck at a logical value of 1 (stuck-at-1) or a logic value of 0 (stuck-at-0), independent of the input to the circuit. If any rail during the layout gets connected to either VDD or GND permanently, it will lead to these stuck at problems.

A2) Total possible faults are 6. (3 nodes, 2faults for each node)
By single fault model, the test patterns that are needed are: 01,10 and 11
The stuck-at-0 problems at any of the inputs and stucl-at-0 problem at the output can not be distinguishable.

A3) (a) Test pattern/set : The set of all input combinations that is needed to find out all the stuck-at faults of a digital circuit. Eg: Test set for 2-input AND gate: { 01, 10,11}

(b) ATPG: ATPG, or Automatic test pattern generation is an electronic design automation tool that generates the complete test set to distinguish between the correct circuit behavior and the faulty circuit behavior caused by a particular fault.

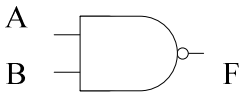
A4) Assume that there is only fault in the given circuit. This is called single fault model. Now apply the input combination such that the correct and faulty circuits would give different outputs.



A5) We need to select the pattern such that none of the inputs at AB,C& D should give 0 at any of the inputs of NAND gate. So the possible pattern is: A = B = C = D = 1
So we need to apply 1111 at the input, if it is correct circuit we will get, 0 at the output and if there is stuck-at-0 problem at P, we will get 1 at the output.

A6) Single Fault method is used here.

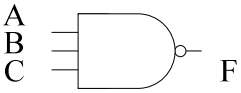
(a) 2-input NAND gate:



Node	For Stuck-at-0	For Stuck-at-1
A	11	01
B	11	10
F	00 or 10 or 10	11

So complete test set = { 10 , 01 , 11}

(b) 3-input NAND gate:

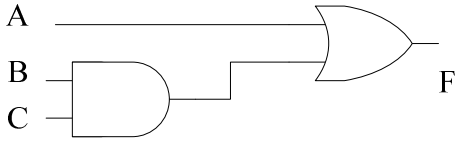


Node	For Stuck-at-0	For Stuck-at-1
A	111	011
B	111	101
C	111	110
F	Atleast one input is 0	111

So complete test set = { 011,101,110,111}

A7) N+1

A8) Single fault model is used.



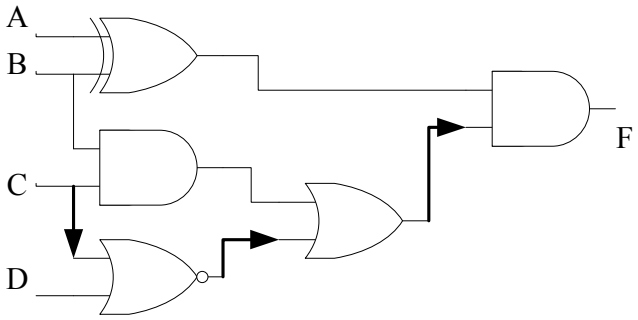
Node	For Stuck-at-0	For Stuck-at-1
A	100	000
B	011	001
C	011	010
F	1xx	000 or 010 or 001

So complete test set = { 000, 001, 010, 011, 100}

A9) (a) F = (C+D)' = C'D'
(b) F = AB

A10) If the number of nodes is more in a given circuit, it is very difficult to derive the test pattern by using single fault model (That is analyzing at each node). The other method called "Path sensitized" can be used to make the test pattern generation more efficient. In this method, all the paths from input to the output will be identified and the input will be applied such that the output will be dependant only on one particular path. And this path is called sensitized.

A11)



- To make F dependant of the high lighted path, the output of XOR gate has to be 1. This can be achieved from A = 0, B = 1 or A = 1, B = 0.
- To make the output of OR gate to make dependant only on one input, the other input has to be 0. This can be achieved either B = 0 or C = 0;
- But C can not fixed to 0. So B has to be 0. That implies A = 1
- To make NOR gate output only C dependant, D has to be 0.

So to make the given path sensitized, the input pattern that is needed is,

A = 1, B = 0, C = 1 or 0 and D = 0

A12)

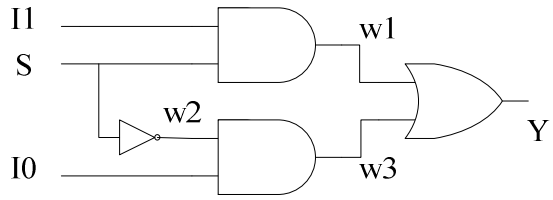
(a) 0100 at the input, makes the path A-w1-F sensitized.
So it can be used to detect the following single stuck-at problems:
Stuck-at-1 at A or Stuck-at-0 at w1 or Stuck-at-0 at F'

(b) Path sensitized method is used here.
To make F dependant only on w3, w1 = 0. So A=B=1
Now for identifying the Stuck-at-0 fault at w3, we need to apply input pattern such that we will get 1 at w3.(C=D=1). So the required pattern is 1111.
For identifying the Stuck-at-1 fault, either C or D has to be 0. So any of the following patterns can be used : 1100,1101 or 1110

A13)

(a) AOI of a 2:1 Mux:

$Y = S' I_0 + S I_1$



(b) All possible paths from inputs to Y:

Path1 : $I_1 – w_1 – Y$,
Path2 : $S – w_1 – Y$,
Path3: $S – w_2 – w_3 – Y$,
Path4: $I_0 – w_3 – Y$

(c) Test vectors needed for each path:

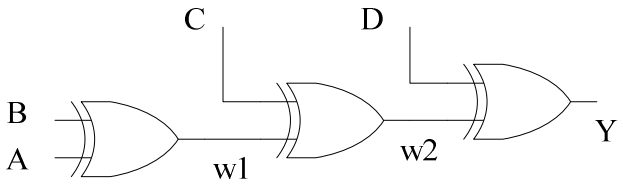
Path	Test vector needed
Path1 : $I_1 – w_1 – Y$	01x, 11x
Path2 : $S – w_1 – Y$	11x, 100
Path3: $S – w_2 – w_3 – Y$	011, x01
Path4: $I_0 – w_3 – Y$	x01, x00

x – indicates don't care

So the complete test set is { 100, 011,11x, x01, x00 }

A14)

(a) The 4-bit even parity generator is shown in the following diagram:



(b) Path sensitized test:

Path	Test vector needed
Path1 :D – Y	0001, 0000
Path2 : C – w_2 – Y	0010, 0000
Path3: B – w_1 – w_2 – Y	0100, 0000
Path4: A – w_1 – w_2 – Y	1000, 0000

So the complete test set is {0000, 0001, 0010, 01000, 1000}

A15) In test of logic circuits, normally the logic levels are represented with D. This is called D-Notation. If Logic-0 is represented with D, logic 1 will be D' and vice versa.

A16) A Static Hazard is defined when a single variable change at the input causes a momentary change in another variable [the output]. A Dynamic Hazard occurs when a change in the input causes multiple changes in the output [i.e. from 1 to 0 and back to 1]. In either case of a Static or Dynamic hazard the product produced is an unanticipated glitch [the hazard]. The resulting glitches in the circuit may or may not induce additional problems, other then increased issues due to switching noise.

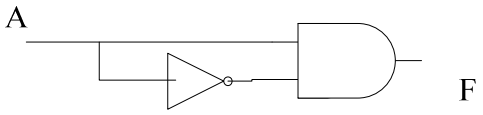
There are two types of Static hazards: the high output transitions to a low and back high [a low going glitch]. Or the low output transitions to a high [1] and back low [0] [a high going glitch]. There are also two types of Dynamic hazards: the 0 output transitions to a 1 back to 0 and then 1 again. Or the 1 output transitions to a 0 back to 1 and then 0 again.

A17) (a) ii
(b) i
(c) iii

A18) Static-zero Hazard's characteristics: Two parallel paths for x, one inverted and reconverge at an AND gate.

F = A . A'

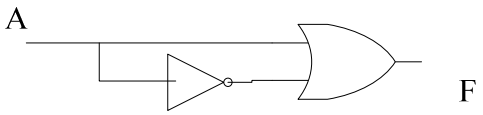
Any circuit with a static-0 hazard must reduce to the equivalent circuit of the following figure:



A19) Static-one Hazard's characteristics: Two parallel paths for x, one inverted and reconverge at an OR gate.

F = A + A'

Any circuit with a static-1 hazard must reduce to the equivalent circuit of the following figure:



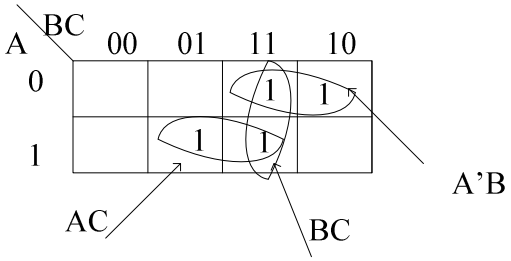
A20) G = AB + A’ C + B’C’D
If B = C = 0, G = A+A’ (Static-0 hazard in A)
If A = 1, C = 0, D = 1, G = B+B’ (Static-0 hazard in B)
If A = 0, D = 1, G = C +C’ (Static-0 hazard in C)

A21) To avoid the static hazards, one of the possible ways is delay matching. Suppose in the circuits shown above (A18 & A19), we can provide buffer whose delay is equal to that of NOT gate. But it becomes very difficult to match the delays exactly.

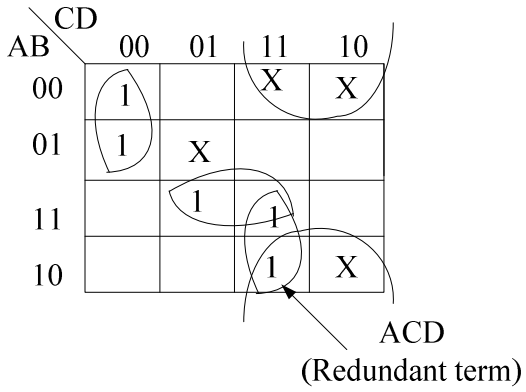
If Static Hazards are removed from the design, Dynamic Hazards will not occur. A Karnaugh map [K-map] is the easiest way to eliminate a Static Hazard or glitches. A K-map for each combinatorial logic function which has an output should be used. Redundant prime implicants should be added to the K-Map (circuit), which will guarantee that all single-bit input changes are covered. Multi-level functions will be reduced to "two-level" functions, and analyzed by the K-map approach. The procedure for designing a static-hazard-free network is a straightforward application. The key is to place the function in such a form that the transient output function guarantees that every set of adjacent 1's in the K-map are covered by a term, and that no terms contain both a variable and its complement. The former condition eliminates 1-hazards and the latter eliminates 0-hazards.

A22) (b)

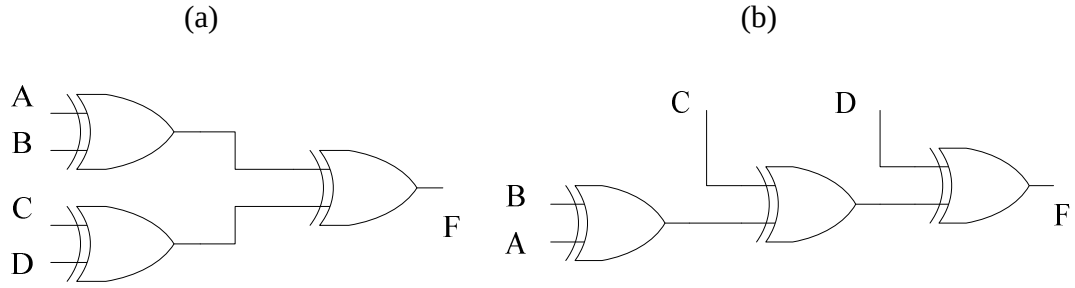
The boolean expression, Y = AB + A’C + BC can be further reduced to Y = A’C + AB. But the second expression will have hazards. So the redundant term BC is added. You can observe the K-Map of the same:



A23) F(A,B,C,D) = ∑ (0,4,11,13,15) + d(2,3,5,10)
From the K-Map shown below, the simplified expression for F is,
F = ABD + A’C’D’ + B’C
But to make it Hazard free, we need to add the redundant term, ACD to this.
F = ABD + A’C’D’ + ACD (Note that B’C is removed from the equation)

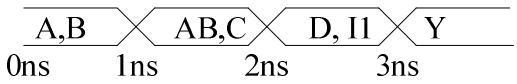


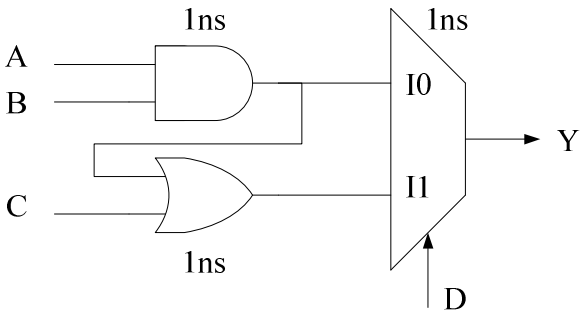
A24) The two possible implementations are shown below:



If we compare both the implementations, in implementation (a), the delays from the inputs to the output, F are uniform. So there is no possibility of glitches. Where as in the implementation (b), the delays are not balanced properly. (a) is hazard free and the better implementation when compared to (b).

A25) Y = AB + CD





Chapter 10: Digital Integrated Circuits

Questions:

Q1) What do you mean by CMOS technology? Explain with a block diagram.

Q2) What are the advantages and disadvantages of CMOS logic?

Q3) Why CMOS tech is widely used?

a) Ease of manufacturing
b) Low power dissipation
c) Speed of the CMOS tech
d) Low cost

Q4) Show the circuit for CMOS inverter and explain the basic operation?

Q5) Draw the VTC of a CMOS inverter?

Q6) Arrange the following in the decreasing order of voltage levels:

$V_{OH}, V_{OL}, V_{IH}, V_{IL}$.

Q7) Define: (a) Fan-out (b) Noise-margin

Q8) Find the noise margin: V_{OH} = 4V, V_{IH} = 3V, V_{OL} = 1V and V_{IL} = 1.5V

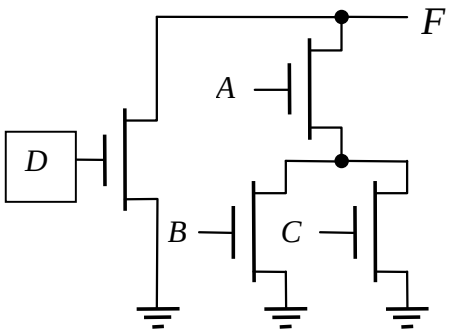
Q9) Find out fan_out and propagation delay of a logical gate for which the following specifications are given: V_{CC} = 5V, I_{CCH} = 1mA, I_{CCL} = 2mA, I_{OH} = 1mA, I_{OL} = 20mA, I_{IH} = 0.05mA and I_{IL} = 2mA

Q10) With increase in frequency of operation the fanout,

a) increases
b) decreases
c) doesn't change
d) none of these

Q11) Draw the CMOS implementation of NAND and NOR gates.

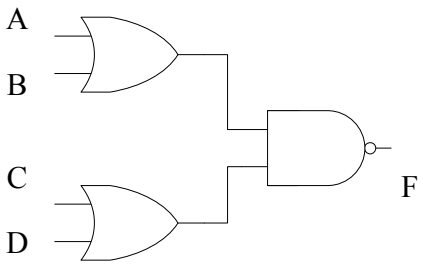
Q12) The following circuit shows the CMOS implementation of a Boolean function. Complete the dual PMOS network and also write the boolean function?



pull-down network

Q13) How many **minimum** number of MOS transistors are required to implement the Boolean function, Y (A,B,C) = AB + A'C + BC using CMOS implementation assuming the inputs and their complements are available?

Q14) Draw the CMOS implementation for the following circuit:



Q15) (a) Design a 4-input NAND gate using only 2-input NAND gates?

(b) What are the minimum number of transistors that are needed to draw the gate level equivalent of the same?

Q16) How many minimum number of MOS transistors are required to implement a Full Adder using CMOS technology?

Q17) Let A & B be two inputs of the NAND gate. Say signal A arrives at the NAND gate later than signal B. To optimize delay, of the two series NMOS inputs A & B, which one would you place near the output?

Q18) Consider CMOS NAND gate. For an input sequence AB = 01→ 11 → 10 → 00, the output logic level is:

a) 5V → 4V → 0V → 0V
b) 0V → 0V → 4V→ unknown
c) 5V → 5V → 0V → 4V
d) None of the above

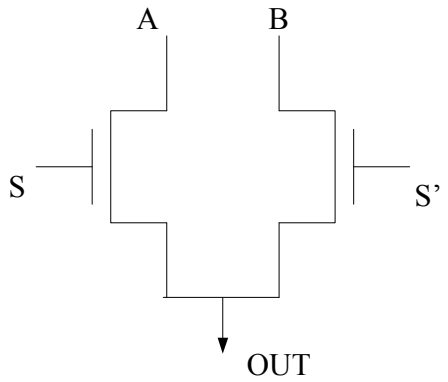
Q19) Draw the stick diagram of CMOS inverter?

Q20) Draw the stick diagram of NOR gate. Optimize it.

Q21) What do you mean by pass transistor logic?

Q22) Show the implementation of AND gate using pass transistor logic?

Q23)



(a) What is the functionality of the following circuit?
(b) Show the Boolean equation?
(c) What is name of logic that is used in implementing the circuit?
(d) Mention the advantages and disadvantages of this method. Also suggest improvements, if any, to overcome the disadvantages?

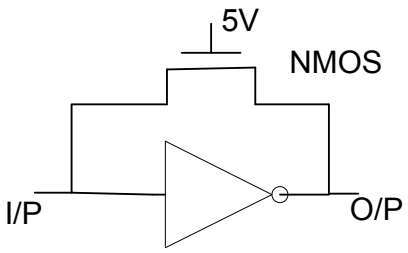
Q24) Show the circuit of Transmission gate and explain the functionality?

Q25) Why don't we use just one NMOS or PMOS in a transmission gate?

Q26) Design a Transmission Gate based XOR. Now, how do you convert it to XNOR? (Without inverting the output)

Q27) Draw a Transmission Gate-based D-Latch.

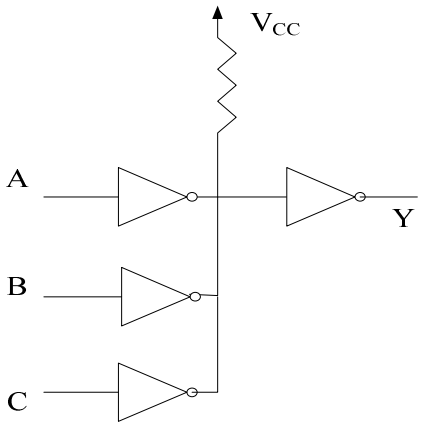
Q28) The output and input of a static CMOS inverter are connected as shown in the above figure. What is the output voltage?



Q29) What are the applications of open-collector gate?

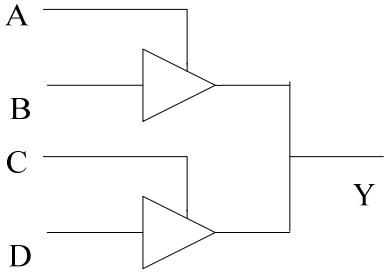
Q30) Using open-collector NAND gate, implement Y = (AB + CD)' ?

Q31) Four open collector gates are connected as shown in following circuit. What is the functionality?



Q32) Describe (a) Three-state buffer gate (b) Three-state inverter gate

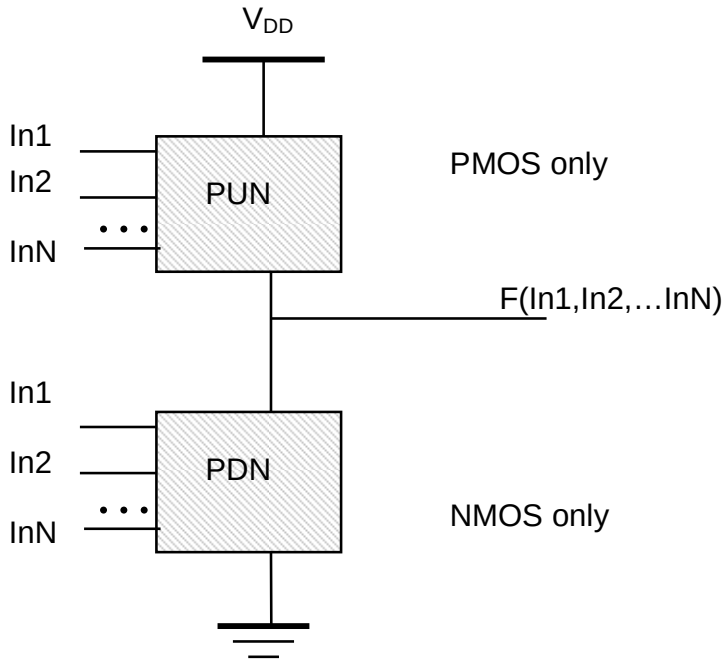
Q33) Two Three-state buffers are shown below. The output of a 4 bit binary counter is connected to the 4 inputs A, B, C and D such that the MSB is connected to D and LSB to A. For how many counter states, the circuit is sure to produce proper output(0 or 1)?



Q34) What is Latch Up? How do you avoid Latch Up?

Answers:

A1) CMOS(Complementary MOS) circuits consist of both types of MOS devices interconnected to form logic functions as shown in the following block diagram. The PUN(Pull up network) will charge the output node in case of Logic-1 and the PDN(Pull down network) will discharge by connecting the output node to ground, in this way the out put is connected either to VDD or GND continuously. PUN and PDN are dual logic networks. CMOS take advantage of the fact that both n-channel and p-channel devices be fabricated on the same substrate.



A2) The CMOS logic has two important advantages:
Advantages:

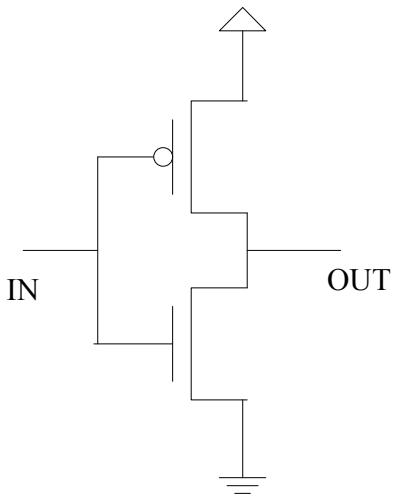
- No direct path in steady state between power and ground, so no static power dissipation(except for small power dissipation due to leakage currents)
- Full logic levels (The VTC exhibits a full output voltage swing between 0 and VDD, and that VTC is usually very sharp)
- High noise margins (Good noise immunity)
- Extremely high input resistance; nearly zero steady-state input current
- Low output impedance. Always a path to Vdd or Gnd from output node, in steady state;
- CMOS provides a greater packing density.

Disadvantages:

- CMOS processing is more complex
- Latch up problem
- Slower compared to TTL
- Higher cost

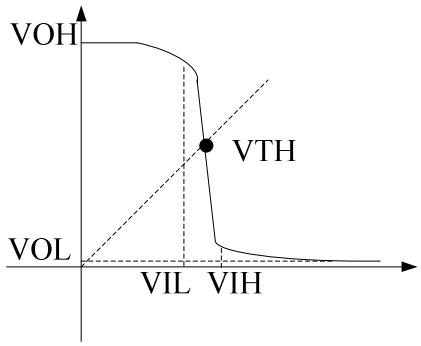
A3) (b)

A4) *CMOS inverter:*



Basic operation: When input goes from 0 to 1, the PMOS will be off and the NMOS will be on. This makes the OUT to get connected with GND and goes to 0. Similarly when input is 0, the NMOS will be OFF and PMOS turns ON making the output logic to VDD. We will get full logic levels at the output.

A5) Voltage Transfer Characteristics (VTC) of a CMOS inverter:



A6) $V_{OH} > V_{IH} > V_{IL} > V_{OL}$ (Refer to VTC shown in A5)

A7)

(a) Fan_out: The fan-out of a gate specifies the number of standard loads that can be connected to the output of the gate without degrading its normal operation. The fan-out is calculated from the amount of current available in the output of a gate and the amount of current needed in each input of a gate.

$$\text{Fan\_out} = \text{Min}\ (I_{OH}/I_{IH},\ I_{OL}/I_{IL})$$

(b) Noise margin: Noise margin is the maximum noise voltage that can be added to an input signal of a digital circuit that does not cause an undesirable change in the circuit output.

$$\text{Noise margin} = \text{Min}\ (V_{OH}-V_{IH},\ V_{IL}-V_{OL})$$

A8) Given: $V_{OH} = 4V$, $V_{IH} = 3V$, $V_{OL} = 1V$ and $V_{IL} = 1.5V$

$V_{OH} - V_{IH} = 4 - 3 = 1$

$V_{IL} - V_{OL} = 1.5 - 1 = 0.5$

Noise Margin = 0.5

A9) (a) Fan_out :

$I_{OH} = 1mA$, $I_{OL} = 20mA$, $I_{IH} = 0.05mA$ and $I_{IL} = 2mA$

$I_{OH}/I_{IH} = 1/0.05 = 20$

$I_{OL}/I_{IL} = 20/2 = 10$

So, Fan_out = 10

(b) Power dissipation :

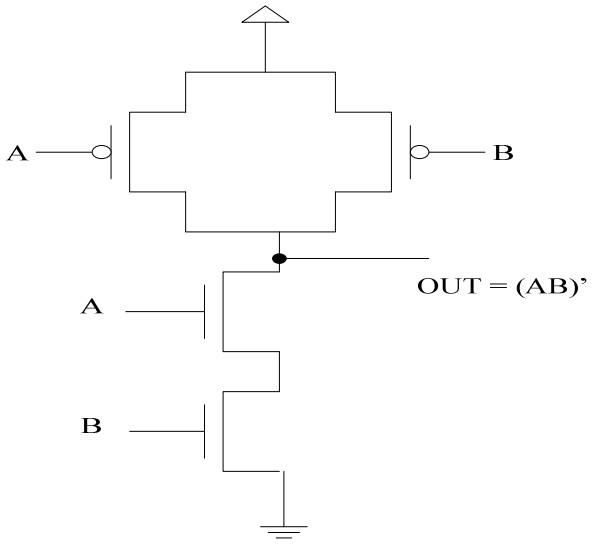
$V_{CC} = 5V$, $I_{CCH} = 1mA$, $I_{CCL} = 2mA$

$I_{cc}(avg) = (I_{CCH} + I_{CCL})/2 = 1.5mA$

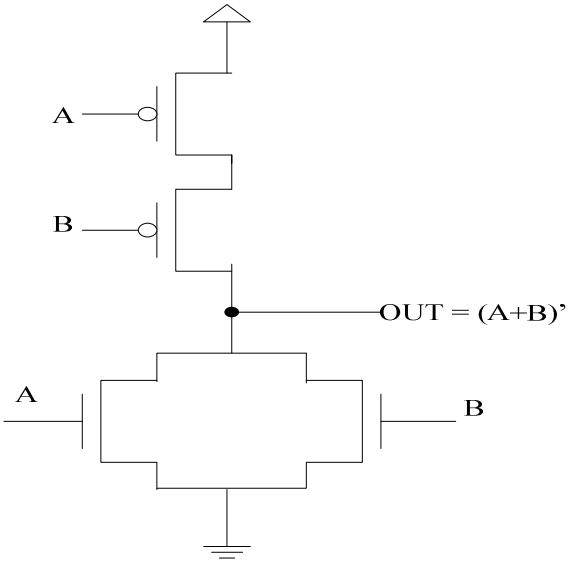
Power dissipation = $V_{CC} * I_{cc}(avg) = 5 * 1.5 = 7.5$ mW

A10) (b)

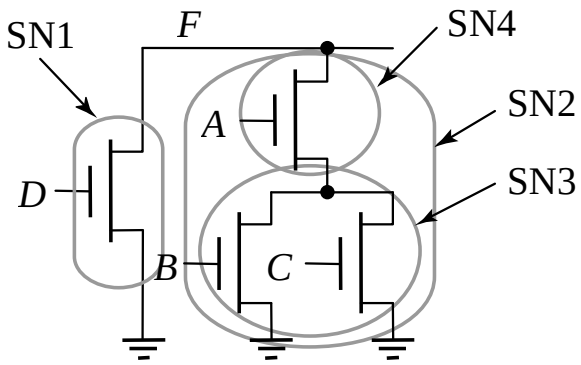
A11) (a) 2-input NAND gate:



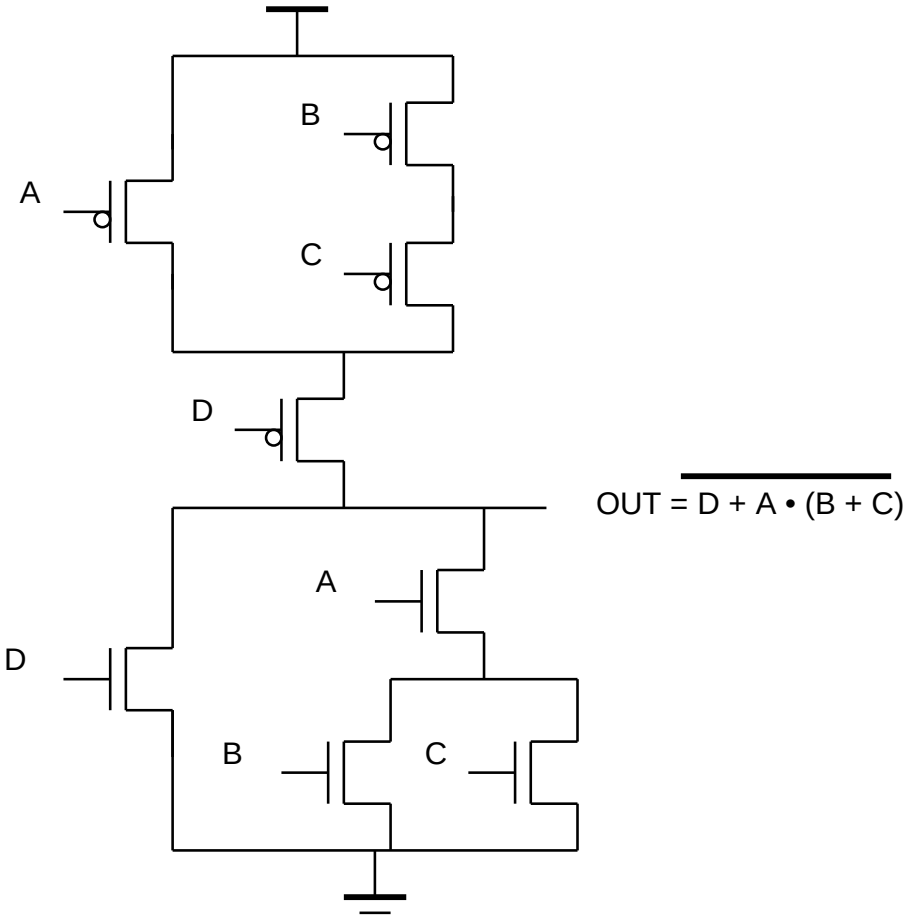
(b) 2-input NOR gate:



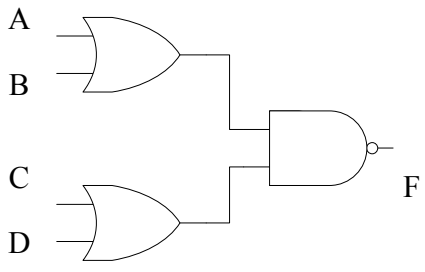
A12)
Deriving the Pull-up network hierarichally identifying sub nets as shown in the following figure:



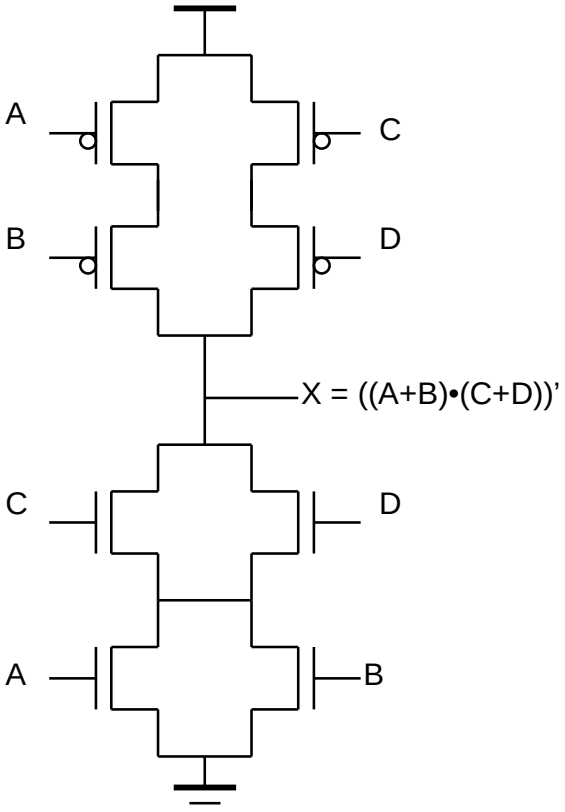
The complete circuit and the output boolean function is shown below:



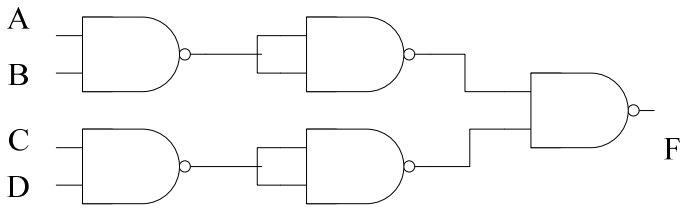
A14)



F = ((A+B) . (C + D))’



A15) (a) 4-input NAND gate from 2-input NAND gates:



(b) Number of NMOS = Number of PMOS transistors = 4 (So total, 8)

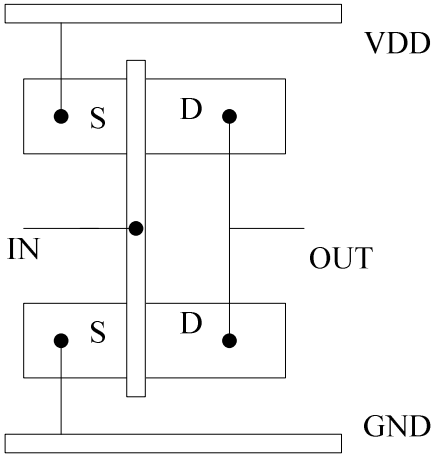
A16) S = A XOR B XOR C and Cout = AB + BC + AC = AB + (A+B)C
S can be rewritten as, S = ABC + (A+B+C) Cout'
For Cout, NMOS = PMOS = 6 (Total 12)
For S, NMOS = PMOS = 8 (Total 16)
So for one bit full adder implementation, minimum number of transistors that are required = 28

A17) The late coming signals are to be placed closer to the output node ie A should go to the NMOS that is closer to the output.

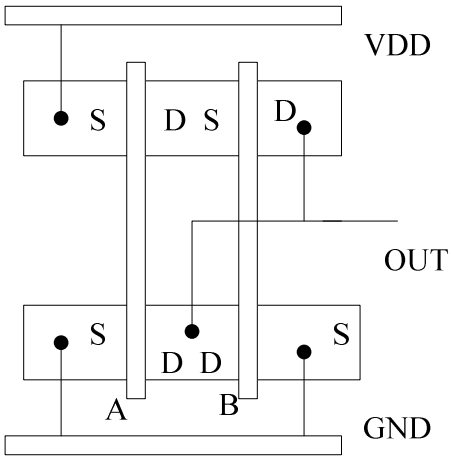
Reason is, by the time A comes, B would have turned on the bottom transistor and discharged the intermediate node between the 2 series NMOS. So by the time A comes, it can discharge the output node very quickly.

A18) (d)

A19) Stick diagram for CMOS inverter



A20) Stick diagram for NOR gate

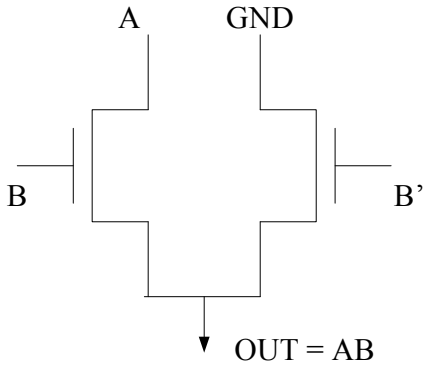


A21) Pass transistor logic:

- A pass transistor is a MOSFET in which an input is applied not only to the gate but also to the drain
- Unlike static CMOS, there is no need for any static power supplies

- More advantageous in terms of number of transistors if the inputs and their complements are available
- Disadvantage is : Degarded logic level as NMOS passes weak logic-1

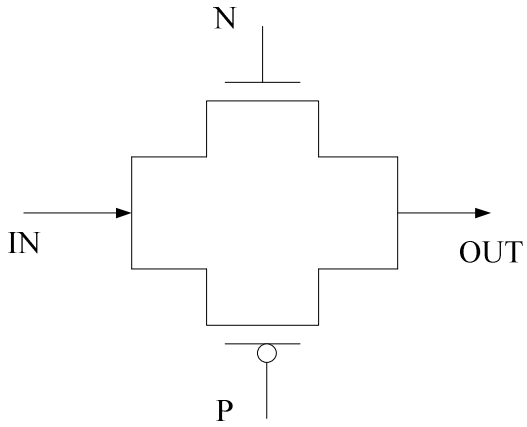
A22) The AND gate using pass transistor logic is shown below:



A23)

(a) 2:1 Mux
(b) OUT = S I_1 + S' I_0
(c) Pass transistor logic
(d) Degraded logic 1. To avoid we need to use both NMOS and PMOS together (That is transmission gate)

A24) Tranmission gate consists of one n-channel and one p-channel MOS transistor connected in parallel. The same thing is shown in the following diagram. When N is at VDD and P is at ground, both transistors conduct and there is a closed path between IN and OUT.



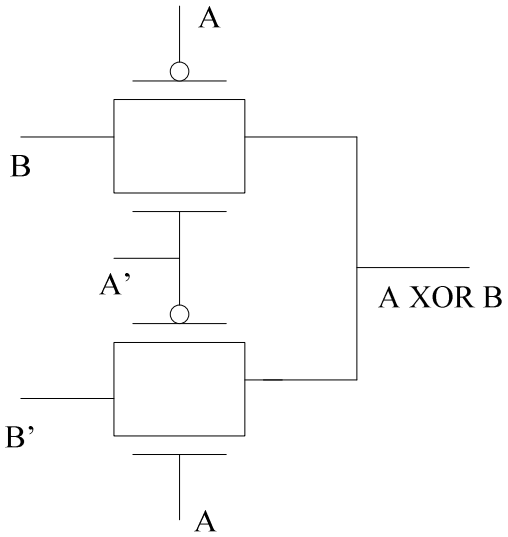
A25) Using only an NMOS will result in an poor 1. Assume the gate voltage on NMOS is 5V. If we connect Drain to 5V, and the source is initially at 0, NMOS will turn on as long as Vgs >Vth, this means, once the source reaches 4.3V (Assuming Vth=0.7), the NMOS

will turn off and there will be no more increase in source voltage. Similarly the opposite happens with PMOS, it doesn't give us a clean 0, but it can give a full 5V. So we use a combination of both NMOS and PMOS so that our signal doesn't get degraded by Vth on either side of VDD and GND.

A26)

XOR Using TG:

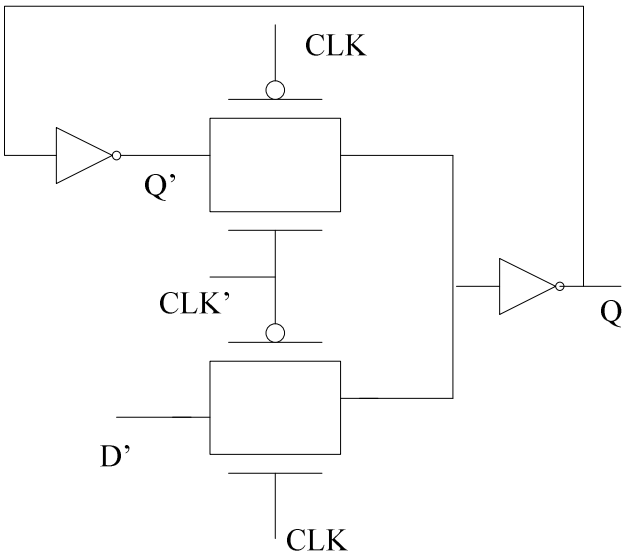
If we observe the truth table of XOR, if A is 1, output is B' and if A is 0 output is B. Using this, we can implement the following circuit.



To get XNOR, just connect B directly to bottom TG and B' to the upper TG.

A27)

D-Latch using TG:

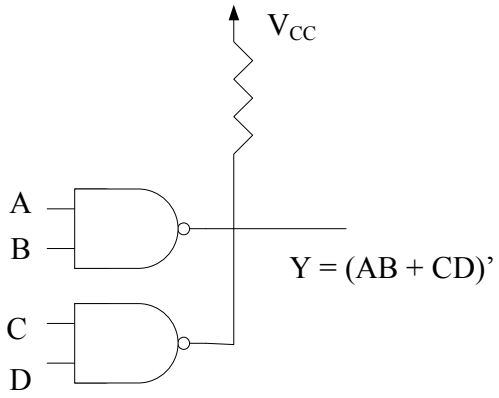


A28) The NMOS transmits the same voltage from drain to source, as long as its value is less than 4V. So in the given diagram the output and input of the inverter are same. If we observe theVTC of an inverter, Vout=Vin at Vth = VDD/2 = 2.5V.So if there is no noise floor, the output will settle to 2.5V (This is theoretical analysis). However practically the circuit will oscillate.

A29) The 3 major applications of open-collector gate are:

- Driving lamp or relay
- Performing wired logic
- Construction of common bus system

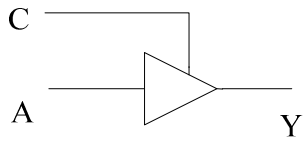
A30)



A31) It forms a common bus system. We can transmit one of the inputs A,B,C to the output Y by making the other inputs 0. Suppose if A=B=0, Y is C.

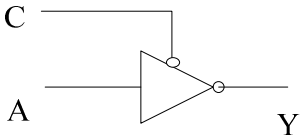
A32)

(a) Three-state buffer:



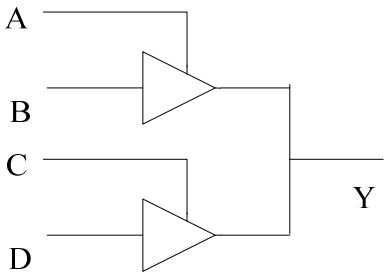
If C =1 , Y = A and if C = 0, Y is high impedance

(b) Three-state inverter:



If C = 0 , Y = A' and if C = 1, Y is high impedance

A33)



If A = 1, Y = B or If C = 1 Y = D otherwise Y is high impedance.
So all combinations with A = 1, C = 0 or A = 0, C =1 or A = 1, C =1
Except the four combinations : 0000,0010,1000 and 1010, The rest all combinations can give either logic-1 or logic-0 at the output.

A34)
Latch up: In fabricating CMOS ICs, parasitic bipolar transistors are formed as by-products of CMOS processing. These parasitic pnp and npn bipolar transistors form a SCR(Silicon controlled Rectifier) with positive feedback and virtually short circuit the power rail to ground. The generation of such a low-impedance path in CMOS chips between the power rail and the ground rail is defined as *Latch-up.*

Guidelines to avoid Latchup:

- Layout n- and p-channel transistors such that all NMOS transistors are placed near close to V_{SS} and all PMOS transistors are placed close to V_{DD} rails. Also maintain sufficient spacings between NMOS and PMOS transistors.
- Use p+ guard rings connected to ground around NMOS transistors and n+ guard rings connected to V_{DD} around PMOS transistors.
- Use minimum area p-wells

Note: Here very basic solution is given. The concept of SCR and guard rings is not given here. It is advisable to read about Latchup.

Chapter 11: Memories, FIFO and Programmable devices

Questions:

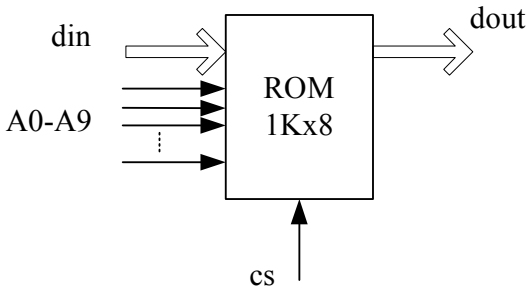
Q1) What is Volatile memory? Give an example.

Q2) Differentiate between RAM and ROM?

Q3) How many address and data lines will be there for a memory of size, 1K X 8?

Q4) How many number of 16X8 size memories are needed to obtain a memory of size 256X16?

Q5) Design a memory of size 8KX8 using a 3:8 decoder and the minimum number of ROMs of size 1KX8 shown in the following diagram. (cs → chip select, active high). Also show the complete address map.



Q6) Using DFF design a binary cell, which can perform read/write operations based on enable, r/w□? Also provide memory enable, mem_en?

Q7) Give the basic circuit for DRAM?

Q8) Draw the circuit for SRAM?

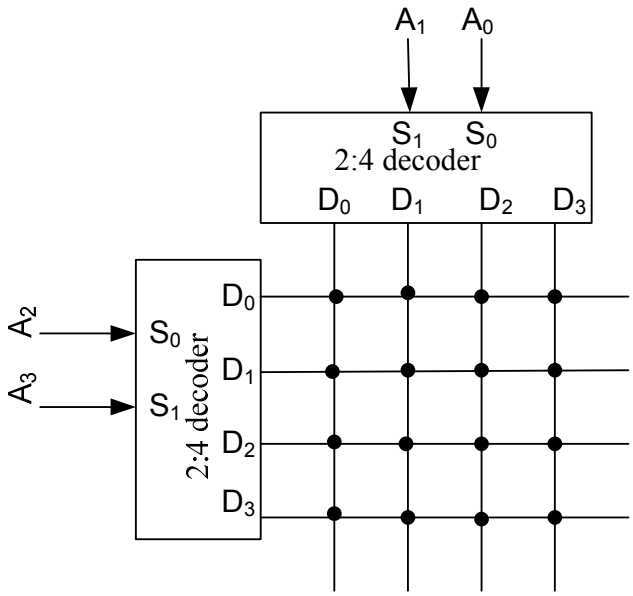
Q9) Define memory access time?

Q10) Which is faster: SRAM or DRAM?

Q11) What are the advantages and disadvantages of DRAM when compared to SRAM?

Q12) Explain read-refresh operation in case of DRAM?

Q13) Using the binary cell as block box, two-dimensional decoding structure of a memory block is shown below:



(a) What is the size of the memory?
(b) What is address of the basic cell at location: Row3, Column3?
(c) Which cell will get selected for the address: C (in hex) ?

Q14) What is dual data ROM?

Q15) Expand the following: (a) PLD (b) PLA (c) PAL (d) FPGA

Q16) What is the difference between PLA and PAL?

Q17) Match the following:

(a) PROM (i) Programmable AND Array and programmable OR array
(b) PAL (ii) Fixed AND array and programmable OR array
(c) PLA (iii) Programmable AND Array and fixed OR array

Q18) Implement the following boolean functions using PLA:

F1 = A'BC + AB'C + B'C'
F2 = A'BC + AB'C
F3 = B'C'

Q19) What is FIFO? Explain the significance?

Q20) Show the basic block diagram of FIFO and explain the basic signals or connections of a FIFO?

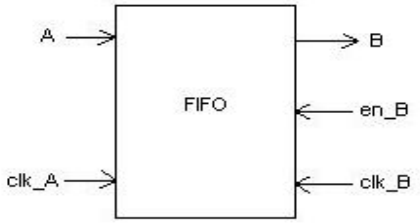
Q21) It is required to connect a Master, which generates data @ 200 Mega Samples/sec to a Slave which can receive the data @ 10 Mega Samples/Sec. If the data lasts in 10Micro Sec, what is the optimal size of FIFO to be used to avoid loose of data?

Q22) In a particular system, the sender sends data at the rate of 80 words / 100 clocks and the receiver can consume at the rate of 8 words / 10 clocks. Calculate the depth of FIFO so that no data is dropped under following assumptions:

- There is no feedback or handshake mechanism.
- Occurrence of data in that time period is guaranteed but exact place in those clock cycles is indeterminate.

Q23) How deep does the FIFO need to be to prevent under flowing or overflowing?
Given Rules:

i) A is input data and B is output data
ii) frequency(clk_A) = frequency(clk_B) / 4
iii) period(en_B) = period(clk_A) * 100
iv) duty_cycle(en_B) = 25%



Q24) What is the difference between Synchronous and Asynchronous FIFOs?

Answers:

A1) Any type of memory that requires power in order to store information is called volatile memory. RAM is volatile whereas ROM is non-volatile. That means ROM can store data without power also.

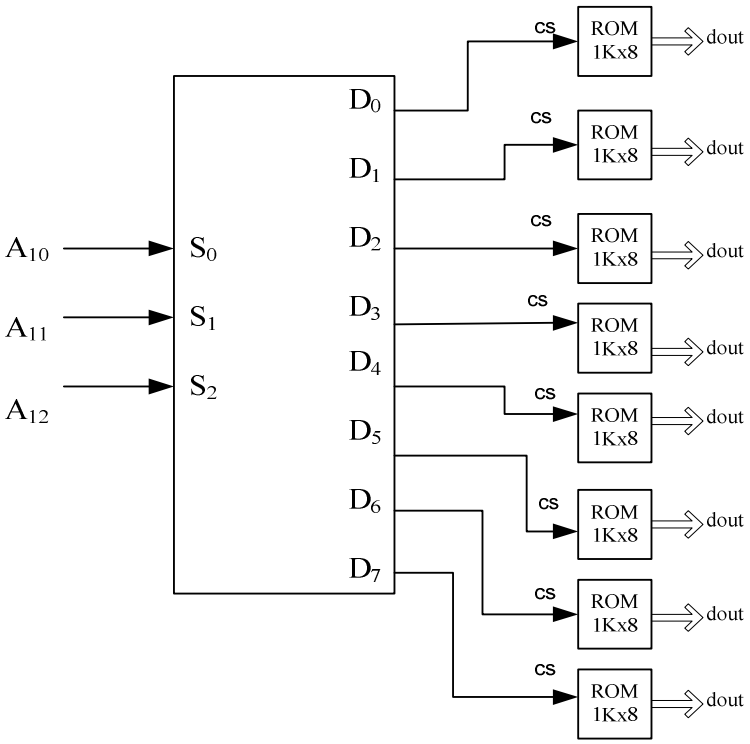
A2) Differences between RAM and ROM:

1. ROM: Read Only Memory. RAM : Random Access Memory
2. ROM has no write operation. RAM has both read and write operations
3. ROMs are non-volatile and RAMs are volatile.

A3) $1K = 2^{10}$, Number of address lines = 10
Number of data lines = 8

A4) 256/16 = 16, 16X8 memories are sufficient to get a memory of size 256 X 8. But to get 256 X 16, we need twice of that.
So, the required number of 16 X 8 memories = 16 * 2 = 32

A5) We need 8, 1KX8 memories. For 8KX8 memory has 13 address (A12-A0) lines. For each 1KX8 memory, there will 10 address lines. So we can connect the A0-A9 address lines directly to these 10 address lines. And the remaining, that is from A10,A11,A12 can be used as select lines for the decoder and the decoders outputs will be connected to “cs” of the ROMs. The complete design is shown below:



The corresponding address map is shown in the following table:

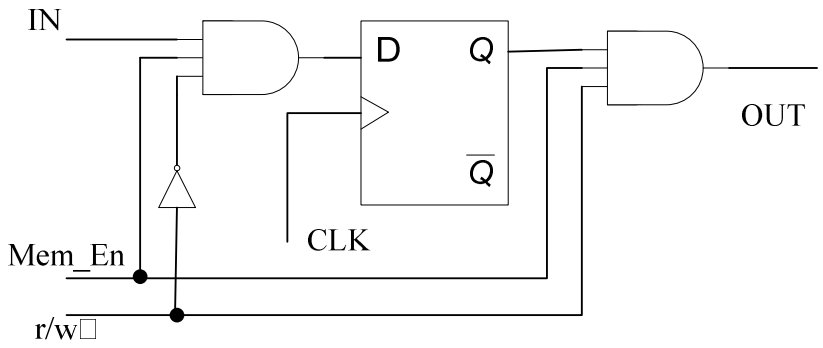
Address range in hex	ROM that will be selected
0000 – 03FF	ROM1
0400 – 07FF	ROM2
0800 – 0BFF	ROM3
0CFF – 0FFF	ROM4
1000 – 13FF	ROM5
1400 – 17FF	ROM6
1800 – 1BFF	ROM7
1CFF – 1FFF	ROM8

A6)

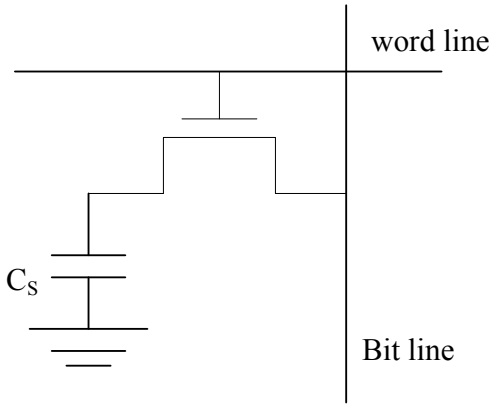
Mem_En	r/w□	Operation
0	X	Output = 0
1	0	Write
1	1	Read

Using the truth table shown in above table, the following equations can be derived: Din = IN AND Mem_En AND (r/w□)' and

OUT = Q AND Mem_En AND (r/w□)



A7) One-transistor Dynamic RAM cell is as follows:



The circuit shows the access transistor, NMOS transistor and the storage capacitance (typically 30-50 fF). Logic-1 at word line makes the MOS transistor conductive. Then the bit line capacitance (nearly 30 times more than C_S) comes in parallel with the C_S. This allows the charge sharing between the two capacitors. (A filled capacitor equals to a logical one while an "empty" capacitor equals to a logical zero.)

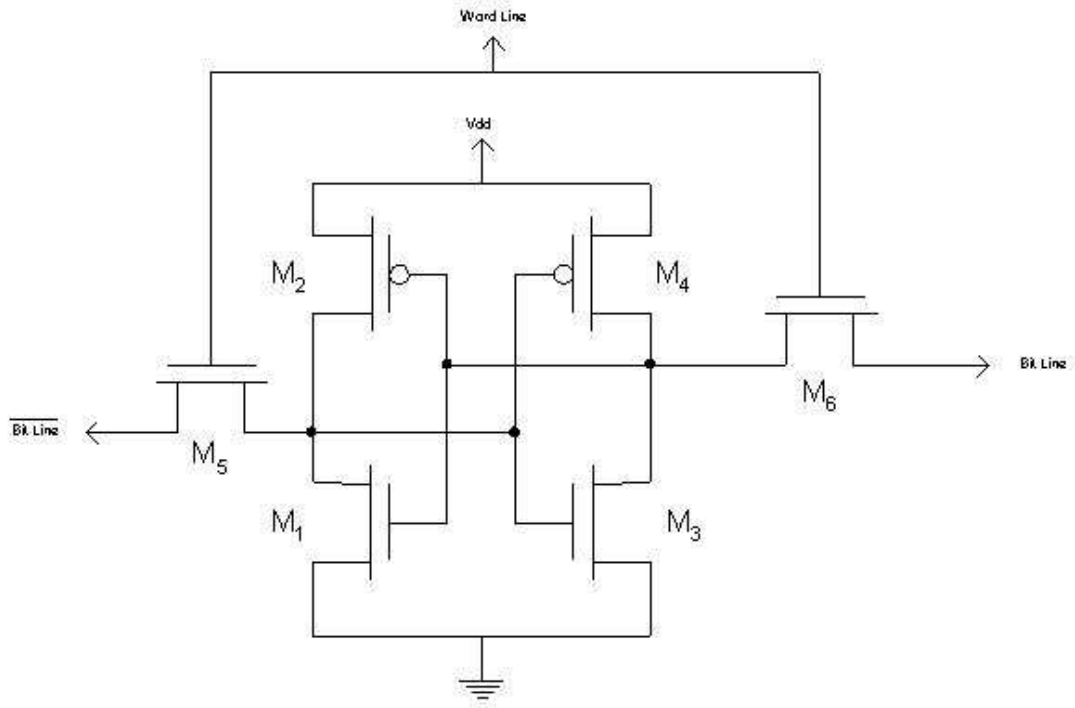
A8) A single SRAM memory cell is shown in the below diagram. As can be noted, six total transistors are required for our design. Two NMOS and two PMOS transistors are used to construct a simple latch to store the data, plus two more pass NMOS transistors are controlled by Word Line to pass Bit Line and Bit Line Bar into the cell.

Write and Read operations are performed by executing a sequence of actions that are controlled by the outside circuit.

Write Operation: A Write operation is performed by first charging the Bit Line and Bit Line Bar with values that are desired to be stored in the memory cell. Setting the Word Line high performs the actual write operation, and the new data is latched into the circuit.

Read Operation: A Read operation is initiated by pre-charging both Bit Line and Bit Line Bar to logic 1. Word Line is set high to close NMOS pass transistors to put the contents stored in the cell on the Bit Line and Bit Line Bar.

6-T SRAM cell:



A9) The time that is required for the data to be available at the memory output after receiving the new address at the input is called memory access time. It is a measure of a memory devices operating speed.

A10) SRAM is faster than DRAM

A11)
DRAM has 3 main advantages over SRAM:

1. DRAM memory cell (1 transistor and capacitor) is simple and smaller than SRAM (6 transistors).DRAM has more density (more cells per chip). The larger memories are always made of made of DRAMs only. (Main memory)
2. DRAM is cheaper than SRAM.
3. DRAM dissipates lesser power.

Disadvantages:

1. DRAM is slower than SRAM. Where speed is critical, SRAM will be used. Eg: Cache memory
2. DRAM requires periodic refreshing.
3. SRAM is compatible with CMOS technology whereas DRAM is not.

A12) In DRAMs, capacitors are used for storing the information. The capacitor discharges after some time. So periodic refreshing is needed. Also, whenever the cell is read, the value will be written back to the cell. This is called read-refresh operation.

A13)

(a) Size of memory = 16 X 1 (Each binary cell is of width 1)
(b) 1010
(c) Row4, Column1

A14) DDR RAM or double-data-rate RAM is a type of memory integrated circuits used in computers. It achieves greater bandwidth than ordinary RAM by transferring data on both the rising and falling edges of the clock signal. This effectively nearly doubles the transfer rate without increasing the frequency of the front side bus. Thus a 100 MHz DDR system has an effective clock rate of 200 MHz when compared to equivalent SDR RAM.

A15)

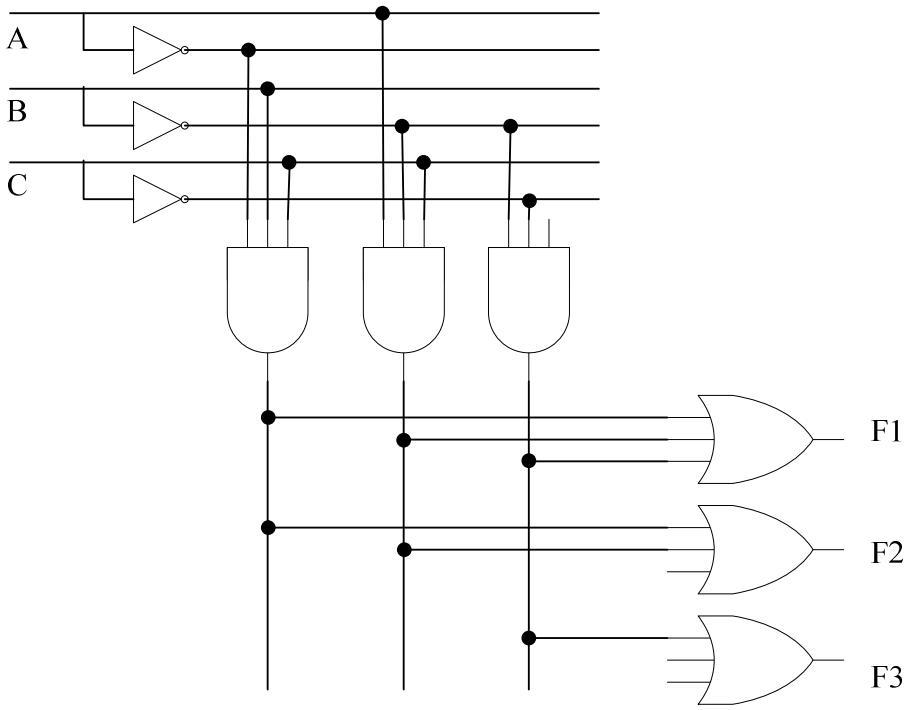
(a) PLD : Programmable Logic Devices
(b) PLA : Programmable Logic Array
(c) PAL : Programmable Array Logic
(d) FPGA : Filed Programmable Gate Array

A16) In PLA both AND and OR arrays are programmable whereas PAL has programmable AND array and a hardwired OR array. When number of functions to be realized is low, PLA is costly. For those cases, PAL is much cheaper.

A17) (a) ii
(b) iii
(c) i

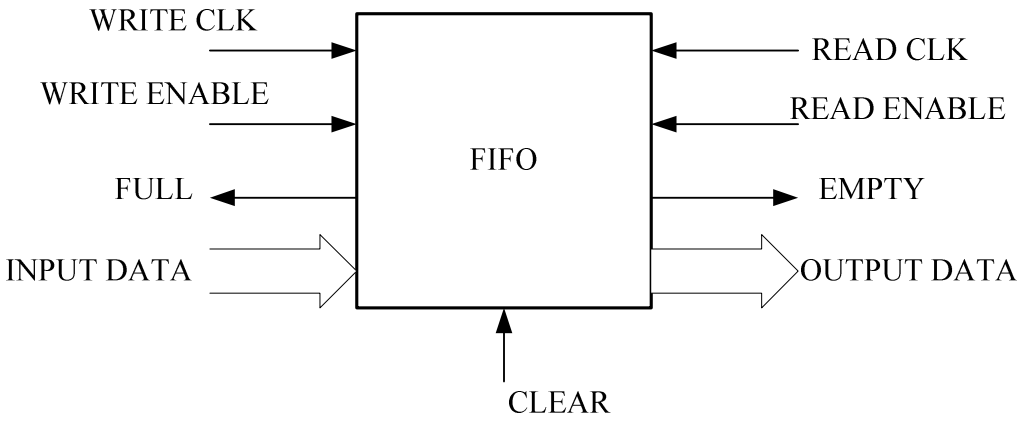
A18)

F1 = A'BC + AB'C + B'C'
F2 = A'BC + AB'C
F3 = B'C'



A19) FIFO (First In First Out) is a special type of storage memory where the first data bit written into the memory is the first to be read out. Putting in another way, FIFO is a storage method that retrieves the data stored for the longest time. The FIFO memory is used when two systems of differing data rates must communicate. Data can be entered into a FIFO register at one end and taken out at the other end at another rate.

A20) A FIFO with all the necessary signal lines is shown in the following block diagram:



A21) Duration of the data = 10 Microsec
Input Data rate (Master) = 200 Mega samples
Output Data rate (Slave) = 10 Mega samples
Depth of FIFO = (Output rate – Input rate) * Duration
= (200-10) * 10 = 1900

A22) In the problem it is given that, out of 100 clocks the sender sends 80 words. The 80 words can occur in any of the 100 clocks. The worst case will be all 80 words coming continuously. So for 10 clocks, the sender sends 10 words where as the receiver can receive only 8 words. So we need to store 16 words in 100 cycles.

Now if we look at the process for long time, the worst case is: During first 100 clocks the sender is idle for 20 clocks and sends the data in the last 80 clocks. In the immediate 100 clocks, data is sent during first 80 clocks. In this case we need to store 32 words.

So the minimum size of FIFO required is 32 words.

A23) This can be solved by taking an example.

Let the frequency of clk_B be 100MHz.

That implies, frequency of clk_A = clk_B/4 = 25MHz

Period of en_B = (1/25M) * 100 = 4 μs

As duty cycle of en_B is 25%, it will be high for a duration of 1 μs. That means B receives the data for 1 μs and will be idle for 3 μs. Where as A sends the data every 0.04 μs. So in 4 μs it can transmit 100 words. And B receives 25 only, so we need to store the rest of 75 words in the FIFO.

So the minimum size of FIFO required is 75 words.

A24) A FIFO where writes to, and read from the FIFO buffer are conducted in the same clock domain is a Synchronous FIFO. For asynchronous FIFO two different and asynchronous clocks would be used. In Synchronous FIFO, as the read and write pointers will be incremented with the same clock it is easy to compare them and enable the FULL and EMPTY signals accordingly. Whereas for asynchronous FIFO this is slightly complicated and involves extra logic for the generation of FULL and EMPTY signals as the read and write pointers are getting incremented with two different clocks.

Chapter 12: Verilog HDL

Questions:

Q1) Declare the following variables in verilog:
- (a) 16 bit wire called "sum"
- (b) An array of intergers of size 16 called "int_arr"
- (c) A memory "mem" of size 128x8

Q2) What is the bit width of data type integer in Verilog?

Q3) What is the difference between "wire" and "reg" data type?

Q4) Which of the following are not legal identifiers?
- (a) $key
- (b) key$
- (c) 123scan
- (d) Scan123

Q5) How many operands will be there for each of the following verilog operators?
- (a) Reduction nand (~&)
- (b) Logical or (| |)
- (c) Right shift (>>)
- (d) Conditional operator (? :)

Q6)

```
wire [5:0] in;
wire [5:0] out = {in[3:0],2'b0};
wire [5:0] out1 = {2'b0,in[5:3]};
```

- (a) What is the name of the operator used in the above verilog code?
- (b) What is the relation between (i) in and out (ii) in and out1?

Q7) What does the "out" give?

```
wire [7:0] in;
wire out = ^in;
```

Q8) Give single line verilog statements which makes output "out" 1 if
- (a) All bits of a 8 bit register are 0
- (b) Atleast one bit of a 8 bit register is 0

Q9) What is the use of non-blocking assignment in behavioral modeling of verilog? How is it different from blocking-assignment?

Q10) Given the following Verilog code, what value of "a" is displayed?

```
always @(clk) begin
        a = 0;
        a <= 1;
        $display(a);
end
```

Q11) What is the difference between inertial and transport delays? What are the applications?

Q12) What is the difference between the following two lines of Verilog code?

```
#5 a = b + c;
a = #5 b + c;
```

Q13) What is the final value of “b” in the following verilog codes?

```
(a) initial                    (b) initial
    begin                          begin
        a = 0;                         a = 0;
     #5 b = a;                         a = #3 1;
        a  = #3 1;                   #5 b = a;
    end                            end
```

Q14) What do x and y specify in the following verilog statement?

`timescale **x** / **y**

Q15) What is the difference between always and initial statements?

Q16) What is the period of “clk” in the following verilog code?

```
`timescale 10 ns / 1 ps
reg clk;
initial clk = 0;
always #5 clk = ~clk;
```

Q17) Explain the following commands of Verilog: case, casex and casez

Q18) Design of a 4:1 mux in Verilog in two styles of coding is shown below:

Using if-else statements:

```
if(sel_1 == 0 && sel_0 == 0) output = I0;
else if(sel_1 == 0 && sel_0 == 1) output = I1;
else if(sel_1 == 1 && sel_0 == 0) output = I2;
else if(sel_1 == 1 && sel_0 == 1) output = I3;
```

Using case statement:

```
case ({sel_1, sel_0})
    00 : output = I0;
    01 : output = I1;
    10 : output = I2;
    11 : output = I3;
    default : output = I0;
endcase
```

(a) What are the advantages / disadvantages of each coding style shown above?
(b) How Synthesis tool will give result for above codes?
(c) What happens if default statement is removed in case statement?
(d) What happens if combination 11 and default statement is removed? (Hint Latch inference)

Q19) Write a verilog code for positive edge triggered D-flip flop with (a) synchronous reset and (b) asynchronous reset.

Q20) What does the following verilog code do?

```
always@( clk or rst ) begin
if (~rst)
    Q <= 1 b'0;
else if ( clk )
    Q <= ~Q ;
end
```

Q21) Is there any difference between "= =" and "= = =" in behavioral modeling of verilog?

Q22) Describe the difference between the system tasks $monitor and $display?

Q23) List the differences between tasks and functions.

Q24) List the advantages or disadvantages over the other statement, if any, among the following two verilog statements:

```
`define SIZE 40
parameter  SIZE 40
```

Q25) Give the verilog code using behavioral modeling for a 4-bit shift register?

Q26) Explain the application of the verilog key words
(a) begin-end
(b) join-fork

Q27) Which is last statement to be executed and at what time of simulation? Assume timescale of 1ns/1ps.

(a)
```
initial
begin
        a = 0;
  #10 b = 0;
  #30 c = 0;
  #20 d = 0;
end
```

(b)
```
initial
join
        a  = 0;
   #10 b = 0;
   #30 c = 0;
   #20 d = 0;
fork
```

Q28) Give the verilog code for the module which indicates any transition (1 to 0 or 0 to 1) in the input. The output should be high only for one clock. Also provide the reset to the design.

Q29) Is there any problem in the following code: (a) For simulations (b) For synthesis and for post synthesis

```
always @(cntrl,a,b)
begin
      if(cntrl)
      a <= b;
end
```

Q30) What is "OUT" if the wire "sel" is X(unknown)?

```
always@(sel) begin
        if(sel)
                OUT <= 1;
        else
                OUT <= 0;
end
```

Q31) The above code is compressed to a single line code as follows:

```
always@(sel)
        OUT <= sel;
```

Is there any difference between the two verilog codes?

Answers:

A1) The declarations are as follows:

(a) wire [7:0] sum;
(b) integer int_arr[0:15];
(c) reg [7:0] mem[0:127];

A2) 32-bit

A3)
wire: It is a net which represents the connections between components. It is used for continuous assignment, that is, the wire has its value continuously driven on it by outputs of the devices connected to them.
Reg: It is used for procedural assignment. Reg always doesn't mean a flop. reg can keep its value until another value is placed into the registers.

A4)

(e) $key : ***Not Valid,*** "$" at the start indicates the system task.
(f) key$: ***Valid,*** Except at the starting, the $ can be there at any other position.
(g) 123scan : ***Not Valid,*** the identifiers can not start with numbers.
(h) scan123: ***Valid,*** except at the starting, the numbers can be there at any other position.

A5)

(e) Reduction nand (~&) : Unary operator, that is only one operand
(f) Logical or (| |) : Two operands
(g) Right shift (>>) : Unary operator
(h) Conditional operator (? :) : Three operands

A6) (a) concatenation operator
(b) (i) out = in * 4 (multiplication by 4)
(ii) out1 = in/4 (division by 4)

A7) ^ operator is reduction operator, which gives the XOR of all the bits of the input. So the "out" gives the parity of the "in". (Even parity indicator)

A8) (a) out = ~| (in)
(b) out = ~& (in)

A9) The nonblocking statements allow the schedule assignments without blocking the procedural flow. Whereas a blocking statement must be executed before the execution of the statements that follow it in a sequential block. The blocking statements are executed in the same order as they are entered. The nonblocking procedural statement cab be used to make several register assignments within the same time step without regard to order or dependence upon each other. It means that nonblocking statements resemble the actual hardware more than the Blocking assignments.

A10) The $display prints 0.

A11)
Inertial delay: The amount of time that the input pulse must be constant in order for the gate to make a transition is the inertial delay of the gate. Verilog uses the propagation delay of the gate as the minimum width of an input that could affect the output. Inertial delay has the effect of suppressing input pulses whose duration is shorter than the propagation delay if the gate. The modeling in verilog is shown in the following example:

Eg: assign #10 out = in1 & in2;

The values of in1 and in2 at the time of computation are considered and any transition within 10 time units will be suppressed.

Transport delay: The time of flight of a signal traveling a wire of a circuit is modeled as a transport delay. With this model narrow pulses are not suppressed, and all the transitions at the driving end of a wire appear at the receiving end after a finite time delay. Whenever input changes, output is immediately evaluated and kept in a event queue and assigned to output after specified "transport" delay.

Eg: always @(in)
```
begin
    out <= #10 in;
end
```

A12)

#5 a = b + c

This is called regular delay control. The execution of the statement a = b+c will be delayed by 5ns.

a = #5 b + c

This is called intra-assignment delay control. The values of b and c at time=0 are used to calculate b+c and that value will be assigned to 'a' after 5ns.

A13) (a) b = 0 (b) b = 1

A14) x is reference time unit that specifies the unit of measurement for times and delays. y is time precision unit that specifies the precision to which the delays are rounded off during simulation.

A15) "always" and "initial" statements are the two structured procedure statements in verilog. An initial block starts at time 0 and executes exactly once during a simulation. The initial blocks are typically used for initializing, monitoring, waveform etc. Initial block is not synthesizable. Whereas the always statement starts at time 0 and executes the statements in the always block continuously in a loop fashion. This statement is used to model a block of activity that is repeated continuously in a digital circuit. Always block is synthesizable.

A16) clock period = 2 * 5 * 10 ns = 100ns

A17) The case statement compares a expression to a series of cases and executes the statement or statement group associated with the first matching case. The case statement compares 0,1,x and z values in the expression.

Syntax: case(expression)
```
        Choice1: statements;
        Choice2: statements;
        .
        .
```

```
            Default : statements; (optional)
        endcase
```

casex and casez are the special versions of the case statements allow the X and Z logic values to be used as don't cares. casez treats all bit positions with z as a don't cares. Casex treats all x and z values in case alternative as a don't cares.

A18)

(a) Functionality wise and simulation point of view, both the implementations are the same and no extra advantage/disadvantage over the other, except case statement makes code more readable and simple.

(b) Coming to synthesis, if the conditions are mutually exclusive, as shown in this example, case will synthesize it to parallel encoder and if-else synthesizes to a priority encoder, which is extra logic. How ever, if the conditions are not mutually exclusive, even case synthesizes it to priority encoder. So if we want parallel encoder in the second case also, we need to use ***parallel_case*** for synthesis as shown below:

```
case(1) : // parallel_case (Note that during simulations this is just a comment)
  sel_a : out <= a;
  sel_b : out <= b;
endcase
```

(c) Even if you remove default, nothing will happen as all other possible input combinations are defined properly.

(d) If both default and 11 are removed, then latch will be formed. Suppose, if the pattern 11 appears at the input, but as nothing matches to 11 in case alternatives the output will not change. This forms the latch, that is the output holds the previous value. To avoid the latch formation, it is better to use always default statement.

A19)

(a) D-Flip flop with synchronous reset:

```
module dff_syn_reset(clk,reset,d,q,qbar);
        input clk,reset,d;
        output q,qbar;
        reg    q,qbar;

        always @(posedge clk)
        begin
            if(!reset)
            begin
                q <= 0;
                qbar <= 1;
            end
            else
            begin
                q <= d;
```

```
                qbar <= ~d;
            end
        end
    endmodule
```

(b) D-Flip flop with asynchronous reset:

```
    module dff_asyn_reset(clk,reset,d,q,qbar);
        input clk,reset,d;
        output q,qbar;
        reg    q,qbar;

        always @(posedge clk or negedge reset)
        begin
            if(!reset)
            begin
                q <= 0;
                qbar <= 1;
            end
            else
            begin
                q <= d;
                qbar <= ~d;
            end
        end
    endmodule
```

A20) This code implements a D flip flop with asynchronous reset whose output Q' is connected back to D input. So it models the clock frequency divide by 4 circuit.

A21) "= =" is used for comparison of only 1's and 0's .It can't compare X(don't cares). If any bit of the input is X output will be X whereas "= = =" is used for comparison of X also. But in reality as no signal will be "X", both will differ only in simulations.

A22) $monitor and $display both are the system tasks used for display. $monitor monitors the input and prints the output only if there is a change in the input. $display prints the formatted output whenever the statement gets executed.

A23) Differences between Functions/Tasks:

1. Functions must have atleast one input argument.Tasks can have zero or more arguments.
2. Functions return always a single value. Tasks do not return with a value but can pass multiple values through output and inout arguments.
3. Functions must not have delay, event or timing control statements. Tasks may contain delay, event or timing control statements.
4. Function can enable another function only but not another task whereas Task can enable other functions and tasks.

A24)
`define SIZE 40 : The value can be passed during run time also. But it can not be passed as argument during instantiations.
parameter SIZE 40: The value can not be changed during run time. It can be passed as argument along with module instantiations.

A25) The 4-bit shift left register which give the outputs of all 4 bits is shown here:

```
module shift(in,clk,out);

input in,clk;
output [3:0] out;
reg q0,q1,q2,q3;
reg [3:0] out;

always @(posedge clk)
begin
  q0 <= in;
  q1 <= q0;
  q2 <= q1;
  q3 <= q2;
 out <= {q3,q2,q1,q0};
end
endmodule
```

A26)

(a) begin-end:
- Used to model the statements which need to be executed sequentially.
- Statements between begin and end are processed top to bottom.
- Delay values are relative to the execution of the previous statement
- Control passes out of the block when the last statement is executed.

(b) join-fork:
- Used to model parallel blocks.
- Statements between join and fork are executed concurrently.
- All delays are absolute time or relative to the start point of the block.
- Parallel blocks start when the fork statement is executed.

A27)

(a) d = 0 is the last statement to get executed and at a time of 60ns. (10+30+20, delays are relative to previous statements)

(b) c = 0 is the last statement to get executed and at a time of 30ns. (delays are absolute)

A28) The verilog code is shown below:

```
module in_trans(in,clk,reset,out);
input in,clk,reset;
```

```
output out;
reg in_d1;
always @(posedge clk or negedge reset)
begin
  if(reset)
    in_d1 <= 0;
  else
    in_d1 <= in;
end
assign out = in ^ in_d1;
endmodule
```

A29) (a) There is no error or problem in the code for simulations.
(b) But as it forms the latch, it may create problem at later stages of the design. Ideal design should not have code which forms this type of latches.

A30) OUT = 0

A31) Both the codes are not same. In the first code, the OUT will get either 0 or 1 always even if sel is unknown(X). But in the second code sel is passed to OUT as it is, so if sel is X, OUT will also be X. However, this matters only in simulations.