30/7/19. Digital Logic & ~~Digital systems~~ Circuits. Pot. Mita Nasipuri ①

Number System

$(N)_{10}$ → No. of unique symbols

$(N)_{10} = \underbrace{\ldots n_2 n_1 n_0}_{n \text{ symbols}}$

$n_0 \ldots n_K$
$\downarrow \quad \downarrow$
$10^0 \quad 10^K$

$0, 1, \ldots, 9$

$n_K \cdot 10^K + n_{K-1} \cdot 10^{K-1} + \ldots + n_1 \cdot 10^1 + n_0 \cdot 10^0 = (N)_{10}$.

$\underbrace{0,1}_{\text{base 2}}$

Decimal → Binary → conversion.

$(0100111)_2 \rightarrow$ Decimal.

$(0100111)_2 \rightarrow$ Octal

$\underline{00}$

$(N)_{10} \rightarrow (N)_5$.

Subtraction using r's complement

Let A and B ~~are~~ be two +ve numbers in base r. and let the no. of places in the integer parts of A & B be n. $(r^n - B)$.

Let us compute A − B

subtrahend B

step 1. Take r's complement of the ~~subtrahend~~

step 2. Add the minuend A to r's complement of B.

step 3. If there is end around carry generated in the addn, ignore that and the result is a positive no. obtained at step 2. If carry is not generated, then take the r's complement of step 2 and add a (−) sign.

$A = (5678)_{10}$ — $B = (2345)_{10}$.

(2)

(1) → 10's complement of B

$10^4 - 2345 = 7655$.

(2) →

$$\begin{array}{r} {\scriptstyle 1\,1\,1\,1} \\ 5678 \\ 7655 \\ \hline 13333 \end{array}$$

Ans. $(3333)_{10}$.

carry generated

B − A. then,

(1) → $10^4 - 5678$
$= 4322$.

(2) →

$$\begin{array}{r} 2345 \\ 4322 \\ \hline 6667. \end{array}$$

(3) →

$$\begin{array}{r} 9999 \\ 6667 \\ \hline 3332 \\ +\ 1 \\ \hline 3333 \end{array}$$

$$\begin{array}{r} 9999 \\ 5678 \\ \hline 4321 \\ +\ 1 \\ \hline 4322. \end{array}$$

Proof

1. $A - B$
$= A + (r^n - B)$.
$= r^n + (A - B)$.

if $A > B$.
$r^n + (A - B) \geq r^n$.

if $A < B$
$r^n + (A - B) < r^n$
$r^n - (r^n + (A - B))$
$\Rightarrow B - A$
$= -(A - B)$.

Using (r-1)'s complement -

Same as r's complement

but :- (1.) Take ~~Add~~ (r-1) complement of the subtrahend and add to the minuend

(2.) If there is end carry generated in the addn, add that carry to the least significant place of the result and the result is a positive no else if no carry is generated & just put a (-) sign to the result obtained & take the (r-1)'s complement.

$A = (5678)_{10}$ $B = (2345)_{10}$,

9s complement:

$(10^4 - 1) - B$

$= 9999 - 2345$

$= 7654$

```
  1 1 1
  5678
  7654
 13332
  ↳ 1
 + 3333
```

$B - A$

$= B + 9999 - 5678$

$= 4321$.

```
  2345
 +6666
 ------
  9999
  6666
 ------
 -3333
```

1s complement & 2s complement.

A → A' (1s complement) → A (2s complement).

A: 10101 ← flipping the bits: 01010

A (2s complement): 01011

and

+0 & -0 is present.

BCD.

45 →

0 . . . 9

10 → digits

↓

4 bits.

Natural Binary Coded Decimal

0 → 0000

⋮

9 → ~~0101~~ 1001

→ last 6 remain unused.

6 → 110

∴ NBCD

0110

```
 ¹67
+ 85
 152
```

```
  6 5
+ 3 3
  9 8
```

```
0110      0101
0011      0011
1001      1000
 (9)       (8)
```

```
1000  0110        0111
      ~~0100~~    0101
      ~~0011~~    1100
      1110          ⇓
      ~~110~~
  ~~1 ← 1100~~   1100
         1       0110 → ⑥.
      1111     1 ← 0010
```

```
    1111
  +  110
1 ← (1)0101
 ①   ⑤        ②.
```

Decimal	Natural BCD	Excess BCD (BCD + 3)
0	0 0 0 0	0 0 1 1
⋮	⋮	⋮
9	1 0 0 1	1 1 0 0

⑤

simplifying.

Using excess BCD :-

65
33
98

~~33~~ ~~6~~ ~~5~~ → ~~0110~~ ~~0110~~

1001	1010
1011	1000
1 ← 0010	
②	
① ← 0101	
⑤	

+3 +3 → already done.

1001	1000
0110	0110
1111	1110
− 110	−110
1001	1000
⑨	⑧

NBCD → aka Weighted BCD.

Please check

2 4 2 1 → BCD

ASCII → American Standard Code for Information Interchange.

7-bit code

⇓

Representing alpha-numeric code

2^7 = 128 no. of diff. symbols.

UNICODE and EBCDIC

Extended BCD code for information interchange.

Parity Bits

Odd Parity & Even Parity

⇓

No. of 1s → Even → Ø 1
Odd → 0.

Books

1. Digital & Comp. Architecture → Morris Mano.
2. Digital Logic & Circuits → Malvino, Leach, Saha.

Steps for NBCD addition

1. Add all the corresponding BCDs
2. Add six to the least significant digit. If no carry is generated, restore the previous value. If carry is generated store the new value and add this carry to the next higher digit position

Steps for NBCD subtraction

1. Take either 9's complement or 10's complement of each BCD digit.

eg. 17 − 57 = −40

0001 0111
+ 0100 0010
→ 0101 0001
+ 110 1001 ←
1011 + 110
1111

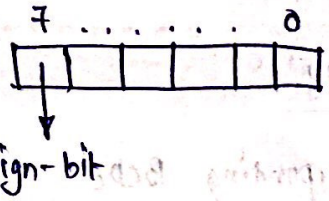
No carry
hence we have to again take 9s complement
∴ −59
↓
−(99 − 59)
= −40.

No carry generated

No carry

(2)

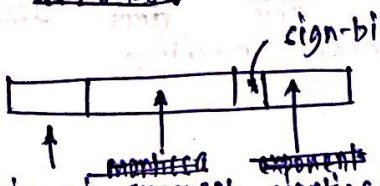
Sign magnitude representation



12345 ← No.

0.12345 × 10

Floating point representation in a computer

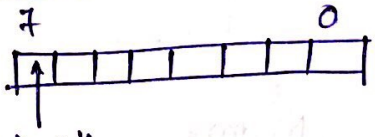


(-) 5 12345

In 32 bit No. Representation.

mantissa → 23
exponent → 8
sign-bit → 1.

Sign-magnitude representation for fixed point number



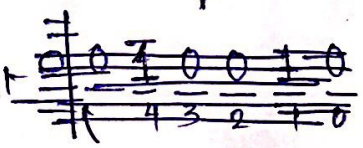
-127 to +127.

1 → negative
0 → positive

+18 → 0,0010010
-18 → 1,1101101

1.11111111
1,0000000

+18



a's complement

a

0 0010010

using 2's complement

+0 & -0 dilemma is neglected.

(+18) + (+6).

0, 0010010
+0, 0000110
(0, 0011000) → +24

(-18) + (+6)

→ no. carry is generated
hence have to take
1s complement again

(-18) + (-6) → carry - generated.
hence add 1 then take
1s, complement - (-ve no.).

Digital Logic

Logical And

Boolean fn.

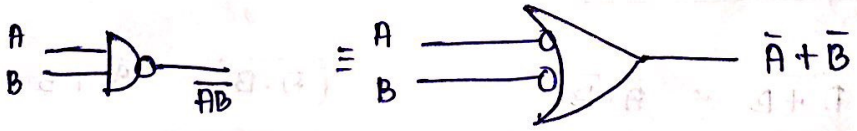
A	B	O
0	0	0
0	1	0
1	0	0
1	1	1

Logical OR.

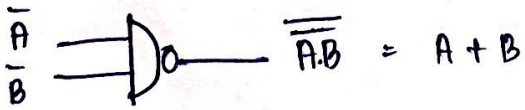
All other fn using NAND Gates.

De-Morgan Laws

NAND Gate



to get A + B



Complement

Boolean Algebra

Summary of Boolean Relations.

1.	Commutative Law	$A+B = B+A$	$A \cdot B = B \cdot A$
2.	Associative Law.	$A+(B+C) = (A+B)+C$	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$
3.	Distributive Law	$A \cdot (B+C) = (A \cdot B) + (A \cdot C)$	$A+(B.C) = (A+B)(A+C)$.
4.	Identity Law	$A+0 = A$	$A \cdot 1 = A$.
5.	Null	$A+1 = 1$	$A \cdot 0 = 0$.
6.	Idempotency	$A+A = A$	$A \cdot A = A$.
7.	Complimentarity	$A+\bar{A} = 1$	$A \cdot \bar{A} = 0$
8.	Absorption	$A+AB = A$	$A \cdot (A+B) = A$
9.	Adsorption	$A+\bar{A}B = A+B$	$A \cdot (\bar{A}+B) = A \cdot B$, $(A+B)(A+\bar{B}) = A$
10.	Uniting	$A \cdot B + A \cdot \bar{B} = A$	
11.	Consensus	$AB + \bar{A}C + BC = AB + \bar{A}C$	$(A+B)(\bar{A}+C)(B+C) = (A+B)(\bar{A}+C)$.

De-Morgan's Theorem

$$\overline{A+B} = \bar{A} \cdot \bar{B} \qquad (\overline{A \cdot B}) = \bar{A} + \bar{B}$$

$$\overline{A+B+C} = \bar{A} \cdot \bar{B} \cdot \bar{C} \qquad (\overline{A \cdot B \cdot C}) = \bar{A} + \bar{B} + \bar{C}$$

Equality ~~vector~~ detector

A	B	O_F
0	0	1
0	1	0
1	0	0
1	1	1

~~$O_F = \bar{A}\bar{B} + AB$~~

$O_F = \bar{A}.\bar{B} + A.B$

Exclusive NOR

or. $O_F = \overline{(\bar{A}B + A\bar{B})}$

$= \overline{(\bar{A}.B)}.\overline{(A\bar{B})}$

$= (A+\bar{B})(\bar{A}+B)$.

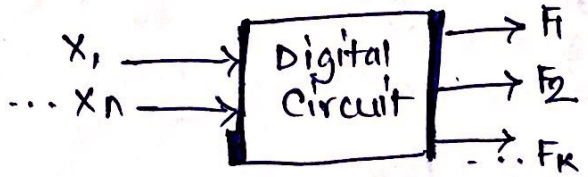
Exclusive OR

$\bar{O_F} = \bar{A}B + A\bar{B}$

07/08/2019

Digital Logic & Digital System/Circuit

Prof Mita Nandan



each F_i, $i = 1, \ldots, K$ can be represented as a boolean fn of the inputs x_i, $i = 1, \ldots, n$ either in normal form or in complemented form

Sum of Product

→ Here the output fn is the summation (OR) of the product (AND) terms involving the inputs or their compliments. (Minterm), If a product term contains all the input variables in the normal form/ complement form, then it is called a minterm.

$$F_1 = x_1 \cdot x_2 + \ldots + \bar{x}_1 x_2 + \ldots$$

Product of sum

The output fn is represented as a product (AND) of terms involving summation (OR) of the input/compliments. If the sum term contains all the input variables in normal form/ complement form, it is called max-term.

Boolean fn.

A	B	C	output ↑ ⊗/F → function output.	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	1	→ $A\bar{B}\bar{C}$
1	0	1	1	→ $A\bar{B}C$
1	1	0	1	→ $AB\bar{C}$
1	1	1	1	→ ABC

In SOP form, we take the min terms.
i.e. $(F = 1)$

$$F = (A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC) = C$$

$$\Rightarrow F = (m_4 + m_5 + m_6 + m_7)$$

for POS form, we take the max terms

Alternate
we take terms having $F = 0$.

$$F' = (\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC)$$

max term
$$\Rightarrow F = \bar{F'}$$

$$\Rightarrow F = (A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(A+\bar{B}+\bar{C}).$$

$$= \pi(M_0 M_1 M_2 M_3) \text{ also } \pi(m_0' m_1' m_2' m_3')$$

using $F = 1$.

Taking POS.

$$F = (\bar{A}+B+C)(\bar{A}+B+\bar{C})(\bar{A}+\bar{B}+C)(\bar{A}+\bar{B}+\bar{C})$$

$F' = \bar{F} =$ ~~A+B~~ $AB'C' + AB'C + ABC' + ABC$

↑ SOP for $F = 0$

$$\bar{F} = A\bar{B}(C+\bar{C}) + AB(\bar{C}+C)$$

$$\Rightarrow \bar{F} = A\bar{B} + AB$$

$$\Rightarrow \bar{F} = A.$$

Simplifying Boolean exp.

To reduce the no. of logic gate involved simplification process although algebraic manipulation is not an easy task as there is no hard & fast rules to predict the next step of multiplication. Veitch has proposed a map method to represent the truth table in pictorial form.

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

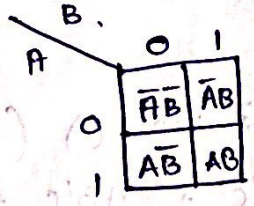
$$F = \bar{A}B + A\bar{B} + AB$$

$$= \bar{A}B + A(B+\bar{B})$$

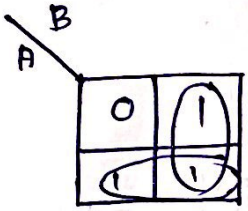
$$= \bar{A}B + A.$$

$$= (A+\bar{A})(A+B). = A+B.$$

Karnaugh Veitch map for 2 variables



Each cell represents the minterm corresponding to the values of input variables.



$= A + B$.

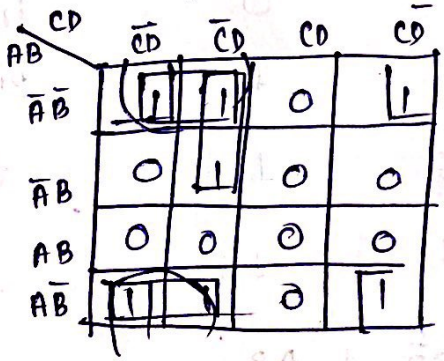
	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_{21}	m_5	m_7	m_6

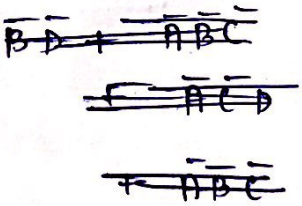
K-Map :-

$= \pi(3, 4, 6, 7, 11, 12, 13, 14, 15)$.

$F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$

$\Rightarrow F(A, B, C, D) =$

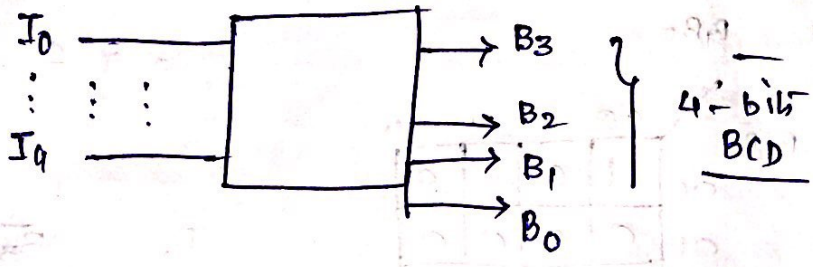




$\bar{B}\bar{D} + \bar{C}\bar{B} + \bar{A}\bar{C}D$

Simplification of Boolean fn

Design an encoder which converts a decimal no. to the corresponding BCD.



$I_0 \ldots I_9$ correspond to decimal digits

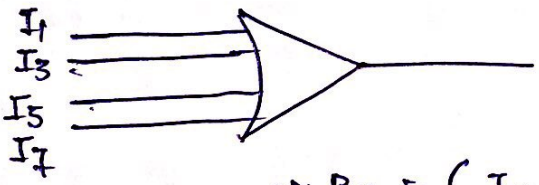
$B_0 = (I_1 + I_3 + I_5 + I_7)$

$B_1 = (I_2 + I_3 + I_6 + I_7)$

$B_2 = (I_4 + I_5 + I_6 + I_7)$

$B_3 = (I_8 + I_9)$

eg for B_0.



$\Rightarrow B_0 = (I_0 . I_2 . I_4 . I_6 . I_8)$,

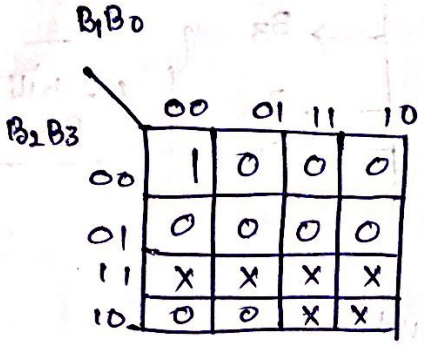
$\rightarrow$ ~~$B_0 = (I_0 I_2 I_4 I_6 I_8)$~~

$\overline{B_0} = (I_0 + I_2 + I_4 + I_6 + I_8)$

..... $\overline{B_3} = (I_0 + I_1 + \ldots + I_7)$

Output & Input (Encoder).

Design a decoder which converts a BCD no. to the corresponding decimal number. ⑥



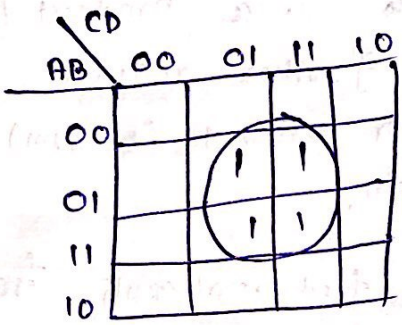
$I_0 = 1$.

$I_0 = \overline{B_3}\,\overline{B_2}\,\overline{B_1}\,\overline{B_0}$

if $I_2 = 1 = \overline{B_2}\,B_1\,\overline{B_0}$

if $I_8 = 1 . = \overline{B_0}B_3$.

Karnaugh-Veitch Map



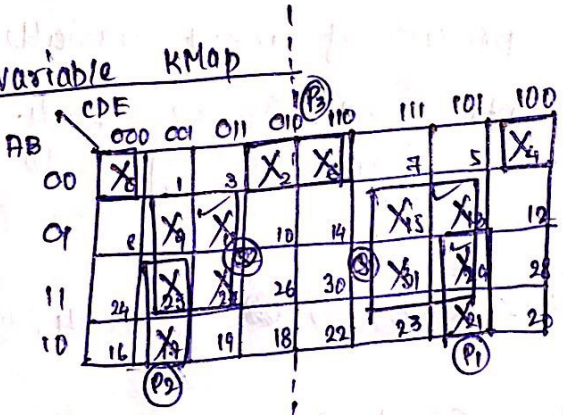
→ For an isolated 1, no minimization

→ For 2 adjacent 1s, one input variable is removed.

→ For 4 adjacent ones, in KMap, 2 input variables are removed.

→ For 8 adjacent ones, in KMap, 3 input variables are removed.

5-variable KMap



$$f(A,B,C,D,E) = \Sigma(0,2,4,6,9,11,13,15,17,21,25,27,29,31)$$

$$= BE(C+\bar{C}) = BE$$

$$= \underbrace{BCE + B\bar{C}E}_{S1} + \underbrace{A\bar{C}DE}_{P1} + \underbrace{A\bar{C}D\bar{E}}_{P2} + \underbrace{\bar{A}\bar{B}D\bar{E}}_{P3} + \underbrace{\bar{A}B\bar{D}\bar{E}}_{P4}$$

$$= BE + A\bar{D}E(C+\bar{C}) + \bar{A}\bar{B}\bar{E}(D+\bar{D})$$

$$= \bar{A}\bar{B}\bar{E} + A\bar{D}E + BE$$

Quine – McCheskey Tabular Method

This is based on the concept of prime implicants. Prime implicants is a product (or sum) term, which cannot be further reduced by combining it with other product (or sum) terms of the given boolean fn.

The tabular method find implicants from the minterms by repeatedly using the relation

$$AB' + AB + A(B' + B) + A.$$

Steps

① Encode each minterm as a binary string of 0s & 1s, depending upon the presence of input variables in its normal / complemental form. Index each minterm by the decimal value corresponding to binary code.

$$F(A,B,C,D) = \Sigma(0,2,6,8,9,10,11,14,15)$$

	A	B	C	D
0	0	0	0	0
2	0	0	1	0
6	0	1	1	0
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
14	1	1	1	0
15	1	1	1	1

(0) — 0 no. of 1s.

(2) 0010

(8) 1000 — 1 no. of 1s

(6) 0110

(9) 1001

(10) 1010 — 2 no. of 1s

(11) 1011

(14) 1110 — 3 no. of 1s

(15) 1111 — 4 no. of 1s

② Group the minterms according to the no. of 1's present in them So. for n no. of input variables there will be (n+1) no. of groups.

③ Sort the groups in an order of no. of 1s.

④ Compare minterms in successive groups. If there is a change in only one bit pos^n, then take that pair of min terms, place a '_' at that different bit pos^n keeping the remaining bits unchanged. The resulting terms are called implicant.

⑤ Repeat step ② - ④ until you get the ~~prime~~ prime implicant.

Group - 0 & Group 1

(0, 2) → 0 0 — 0
(0, 8) → — 0 0 0 ——— 0 no. of 1s

Group - 1 & Group (2)

(2, 6) → 0 — 1 0
~~2, 4~~
(2, 10) → — 0 1 0
(8, 9) → 1 0 0 —
(8, 10) → 1 0 — 0 ——— 1 no. of 1s

Group (2) & Group (3)

(6, 14) → — 1 1 0
(9, 11) → 1 0 — 1
(10, 11) → 1 0 1 —
(10, 14) → 1 — 1 0 ———

Group (3) & Group (4)

(11, 15) → 1 — 1 0
(10, 15) → 1 1 1 —

(0, 2, 8, 10) — 0 — 0
(0, 8, 2, 10) — 0 — 0
(2, 10, 6, 14) — — 1 0
(2, 6, 10, 14) — — 1 0
(8, 10, 9, 11) 1 0 — —
(8, 9, 10, 11) 1 0 — —
(10, 11, 14, 15) 1 — 1 —
(10, 14, 11, 15) 1 — 1 —

⑥ Prepare a prime implement table. It consists of rows representing each prime implement and columns representing minterms. For each row, put a 1 in the column pos^n representing the minterms which are involved in forming the prime implements. ~~Remember~~ Remove the duplicate rows.

Prime implement table

Remove minterms.

Prime implement	m_0	m_2	m_6	m_8	m_9	m_{10}	m_{11}	m_{14}	m_{15}
$-0-0$	1	1		1		1			
$--10$		1	1			1		1	
$10--$				1	1	1	1		
$1-1-$						1	1	1	1

describing

⑦ Find the ~~essential~~ prime applicants by ~~decoding~~ ^ of the columns of a particular column is covered by a single prime implicant, then it is an essential prime implicant and save it as a part of final boolean expression.

⑧ Remove the row containing that essential prime implicant and the columns containing minterms carried by that essential prime implicant.

⑨ Repeat steps ⑦ & steps ⑧ until all the prime implicants in the table is considered.

$-0-0$	$\overline{B}\,\overline{D}$
$--10$	$C\overline{D}$
$10--$	$A\overline{B}$
$1-1-$	AC

HW $f(A,B,C,D,E) = \Sigma(0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 25, 27, 29, 31)$

HW.

$f(A,B,C,D,E) = \ldots\ldots\ldots$

0	0	0	0	0	0
2	0	0	0	1	0
4	0	0	1	0	0
6	0	0	1	1	0
9	0	1	0	0	1
17	1	0	0	0	1
11	0	1	0	1	1
13	0	1	1	0	1
21	1	0	1	0	1
25	1	1	0	0	1
15	0	1	1	1	1
27	1	1	0	1	1
29	1	1	1	0	1
31	1	1	1	1	1

$\ldots\ldots$

$\longrightarrow \ldots\ldots$ Quine - McCluskey Tabular Method

13/8/2019

Digital Logic and Digital Systems

Prof. Mita Nasipuri

Ex. Express $f(A,B,C,D)$ as a sum of product and also as a product of sum form.

$$f(A,B,C,D) = D(\bar{A} + B) + \bar{B}\bar{D}.$$

A	B	C	D	$f(A,B,C,D)$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0

$$F(A,B,C,D) = \Sigma(1,3,5,7,9,11,13,15)$$

$$f(A,B,C,D) = \pi(0,2,4,6,8,10,12,14)$$

(2)

Obtain a simplified expression in product of sum form.

$$F(A,B,C,D) = \Sigma(5,6,7,8,9,12,13,14,5)$$

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	1	1	1	1
10	1	1	0	0

$F'(A,B,C,D) = \bar{A}\bar{B} + \bar{B}\bar{C} + \bar{A}\bar{C}\bar{D}$

Conversion between canonical forms

The complement of a function can be expressed as the sum of minterms missing in original fn. If the original fn considers the minterms for which the value of the fn is equal to 1, then the complement fn will consider the minterms for which the value of the fn is equal to 0.

If $f(A,B,C,D) = \Sigma(1,3,5,7) = m_1 + m_3 + m_5 + m_7$

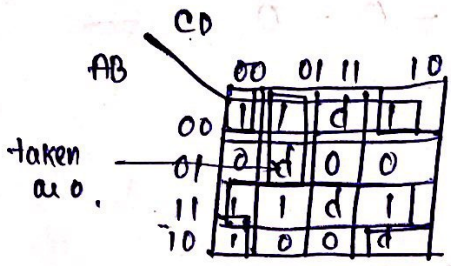
$F'(A,B,C,D) = \Sigma(0,2,4,6) = m_0 + m_2 + m_4 + m_6$.

By DeMorgan's Theorem

$$f(A,B,C,D) = \overline{m_0 + m_2 + m_4 + m_6}$$
$$= \bar{m}_0 \cdot \bar{m}_2 \cdot \bar{m}_4 \cdot \bar{m}_6$$

$$F(A,B,C,D) = \overline{\bar{A}\bar{B} + \bar{B}\bar{C} + \bar{A}\bar{C}\bar{D}}$$
$$= (\overline{\bar{A}\bar{B}})(\overline{\bar{B}\bar{C}})(\overline{\bar{A}\bar{C}\bar{D}})$$
$$= (A+B)(B+\bar{C})(A+C+D).$$

3x. Simplify $F(A,B,C,D) = \Sigma(0,1,2,8,12,13,14) + d(3,5,10,15)$



$F(A,B,C,D) = \bar{B}\bar{D} + AB + \bar{A}\bar{B}$

→ Quinn - McKluskey.

take d = 1. (else it will not be minimized).

0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	1 0 0 0
d_3	0 0 1 1
d_5	0 1 0 1
d_{10}	1 0 1 0
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
d_{15}	1 1 1 1

→

(0,1)	0 0 0 –
(0,2)	0 0 – 0
(0,3)	– 0 0 0
$(1,d_3)$	0 0 – 1
$(1,d_5)$	0 – 0 1
$(2,d_3)$	0 0 1 –
$(8,d_{10})$	1 0 – 0
$(2,d_{10})$	– 0 1 0
$(8,d_{10})$	1 0 – 0
(8,12)	1 – 0 0
$(d_5,13)$	– 1 0 1
$(d_{10},14)$	1 – 1 0
(12,13)	1 1 0 –
(12,14)	1 1 – 0
$(13,d_{15})$	1 1 – 1
$(14,d_{15})$	1 1 1 –

$(0,1,2,d_3)$ 0 0 - -

$(0,2,1,d_3)$ 0 0 - -

$(0,2,8,d_{10})$ - 0 - 0

$(0,8,2,d_{10})$ - 0 - 0

$(8,12,d_{10},14)$ 1 - - 0

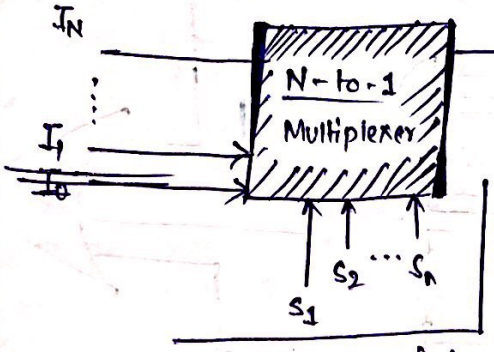
$(8,d_{10},12,14)$ 1 - - 0

$(12,13,14,d_{10})$ 11 - -

$(12,14,13,d_{10})$ 11 - -

Function	minterms										
	m_0	m_1	m_2	d_3	d_5	8	d_{10}	12	13	14	d_{15}
$\bar{A}\bar{B}$ ← 00 - -	x	x	x	x							
- 0 - 0	x		x			x	x				
1 - - 0						x	x	x		x	
11 - -		x						x	x	x	x
0 - 01					x				x		

Multiplexer (MUX).



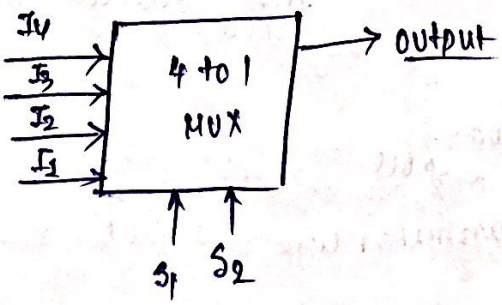
Multiplexer is a combinational logic circuit that selects a particular input line out of N inputs on basis of a n-bit digital ~~input~~ pattern on the selection lines and direct the information coming on the selected input line to the output O. Thus, the combinational circuit act as a decoder which decoder the digital code on the selection lines to enable a particular input line to allow the output and disabling the other inputs. No. of selection line should satisfy the following relⁿ.

$$2^n \geq N \geq 2^{(n-1)}.$$

Functional Table (4 to 1 MUX).

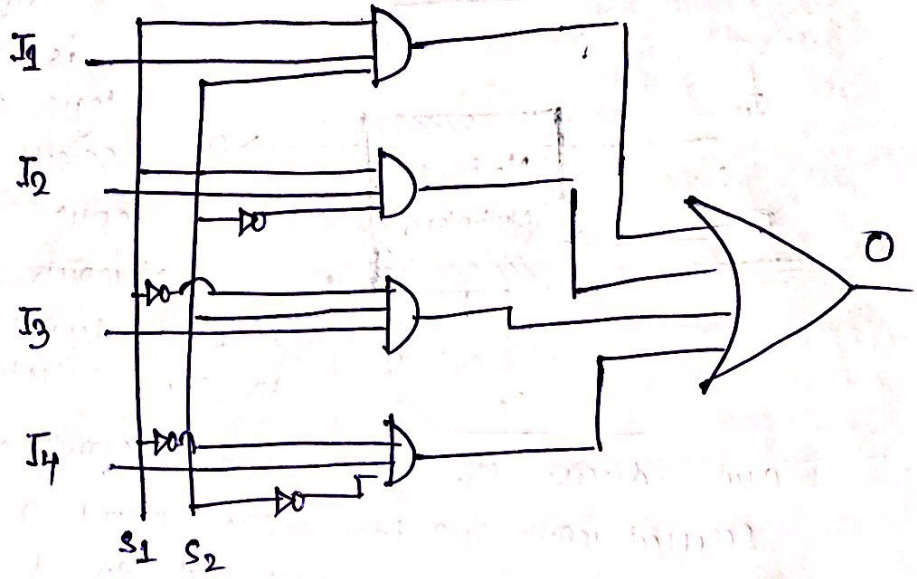
$N = 4$,

$2^2 = N = 4 \quad n = 2$.



S_1	S_2	O
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Logic Diagram

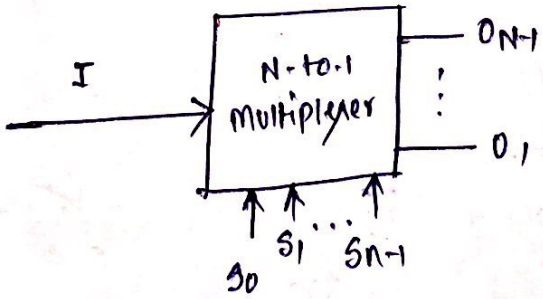


Multiplexer
(Please check with book)

Multiplexer is a combinational logic circuit that selects a particular output line out of N outputs on a n-bit digital pattern on the selection lines and direct the information carrying on the input line to the selected output line.

DeMultiplexer (DeMux)

Block Diagram



Thus, the combinational circuit act as a decoder which decode the digital code on the selection lines to enable the input line to allow a particular line output & disable the other output lines.

No. of selection line n should satisfy

$$2^n \geq N > 2^{n-1}$$

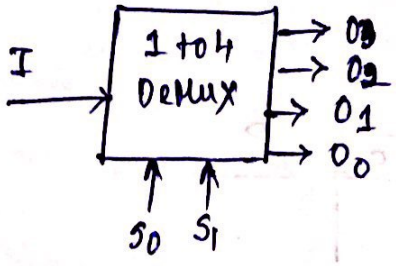
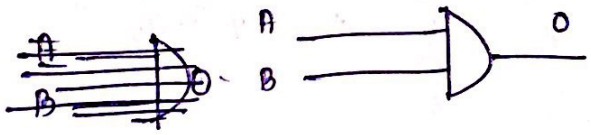


Diagram same as de-multiplexer (Input same, multiple output).

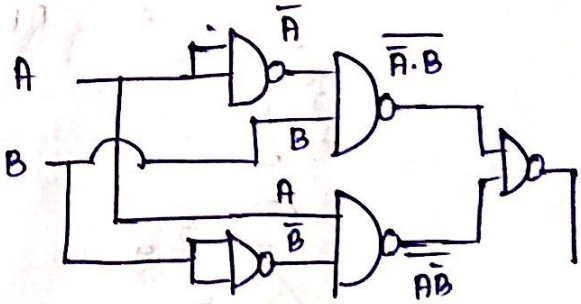
S_1	S_2	O
0	0	O_0
0	1	O_1
1	0	O_2
1	1	O_3

Exclusive OR Gate (XOR Gate)



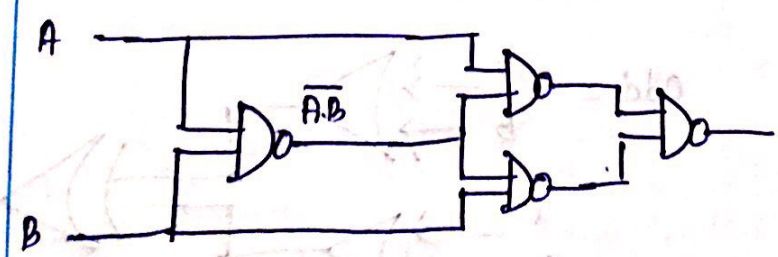
Using Nand.

A	B	O
0	0	0
0	1	1
1	0	1
1	0	0



5 gate

$\bar{A}.B + A\bar{B}$



4 gate

Parity Generator

I_{n-1}	I_0	P

72
84.

a_2	a_1	a_0	Podd	Peven
0	0	0	1	0
1	0	1	1	0
0	0	1	0	1
....			.....	

$$P_{odd} = \bar{a}_2 \bar{a}_1 \bar{a}_0 + \bar{a}_2 a_1 a_0 + a_2 \bar{a}_1 a_0 + a_2 a_1 \bar{a}_0$$

Exclusive NOR

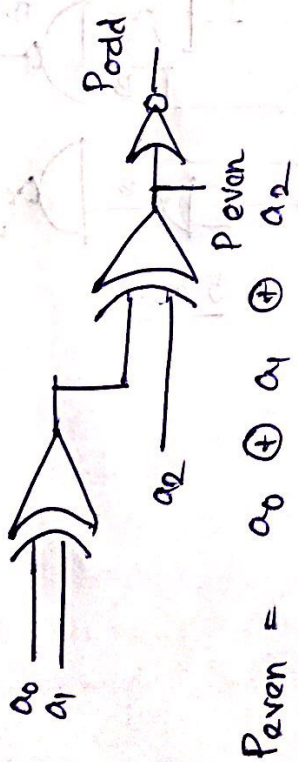
$$0 = \overline{\bar{A}B + A\bar{B}}$$
$$= AB + \bar{A}\bar{B}$$

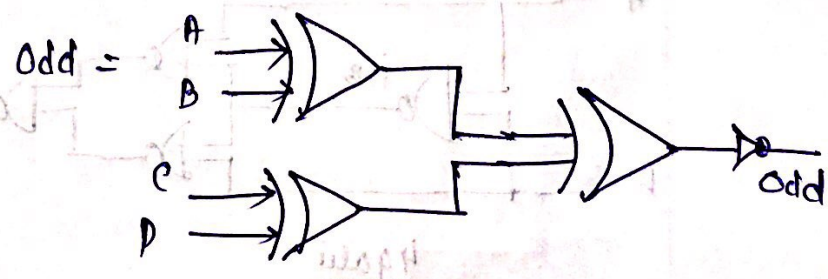
XOR Gate

When system follows.

A	B	C	D	Oodd	Oeven
0	0	0	0	1	0

Faulty.





3 - bit magnitude comparator

2 - 3 bit numbers

a_2 a_1 a_0

b_2 b_1 b_0

↑ ↑

MSB LSB

(3) 00

(5) 84

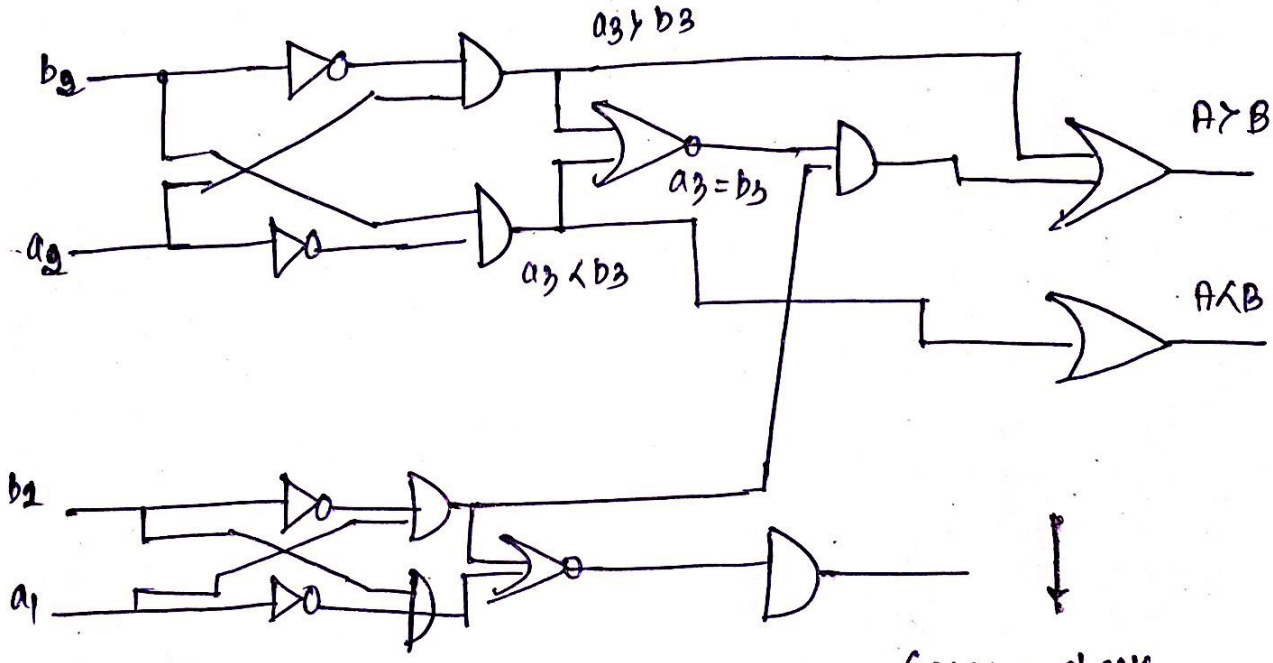
(5) > (3)

(3) 0 0

(3) 8 4

(3) = (3) | (0) < (8)

comparison is done from the most significant bit posn to least significant bit posn.



(please check book/other resource).

Encoder

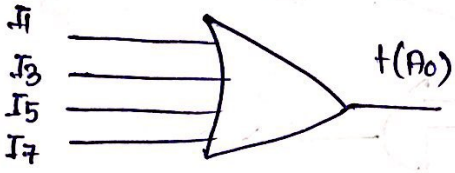
I_0	I_1	.....	I_9	A_3	A_2	A_1	A_0
1	0	.....	0	0	0	0	0
0	1	.....	0	0	0	0	1
		...					

(BCD).

~~$f(A_0) = I_1, I_3, I_5, I_7, I_9$~~ it one of them is 'ON' then A_0 is 'ON'

hence we can represent them

$$f(A_0) = I_1 + I_3 + I_5 + I_7 + I_9$$
$$f(A_1) = I_2 + I_3 + I_6 + I_7$$
$$f(A_2) = I_4 + I_5 + I_6 + I_7$$
$$f(A_3) = I_8 + I_9$$



$$f(\overline{A_0}) = I_0 + I_4 + I_2 + I_6 + I_8.$$

$$\Rightarrow f(A_0) = \overline{I_0 \cdot I_2 \cdot I_4 \cdot I_6 \cdot I_8}$$

$$\overline{I_0} \cdot \overline{I_2} \cdot \overline{I_4} \cdot \overline{I_6} \cdot \overline{I_8}$$

BCD to Decimal Decoder

~~$A_0\ A_1\ A_2\ A_3$~~

A_3	A_2	A_1	A_0	O_0 ..... O_9
0	0	0	0	0 0

$$O_3 = \overline{A_3}\ \overline{A_2}\ A_1\ A_0$$

~~$O_3 = \overline{A_3} A_1 A_0$~~

$$O_3 = \overline{A_2} A_1 A_0$$

$$O_8 = A_3 \overline{A_0}$$

	00	01	11	10
00	0	0	1	0
01	0	0	0	0
11	d	d	d	d
10	0	0	d	d

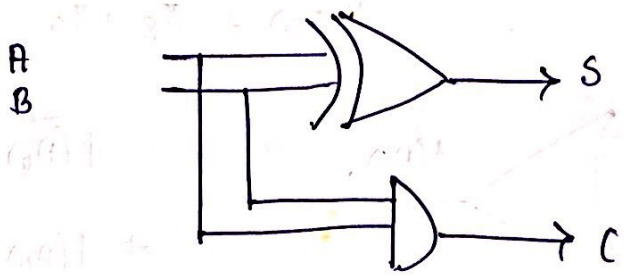
② Adder

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

$S = \bar{A}B + A\bar{B} = A \oplus B$

$C = AB.$

Half adder circuit



$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ + 1 & 0 & 1 & 1 \\ \hline 1 & 1 & 1 & 0 \end{array}$$

Full adder circuit

It considers a ~~carry~~ carry from its just previous lower bit posn.

A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
...				
1	1	1	1	1

using a KMap, we find no minimisation. We find

$S_i = A_i \oplus B_i \oplus C_i$

$C_i = A_iB_i + A_iC_{i-1} + B_iC_{i-1}$

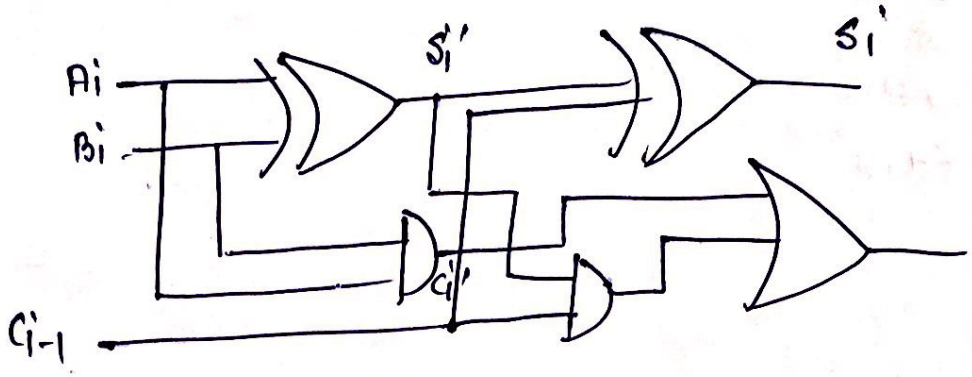
Full adder as a combination of half-adder

$S_i' = A_i \oplus B_i$

$C_i' = A_i \cdot B_i$

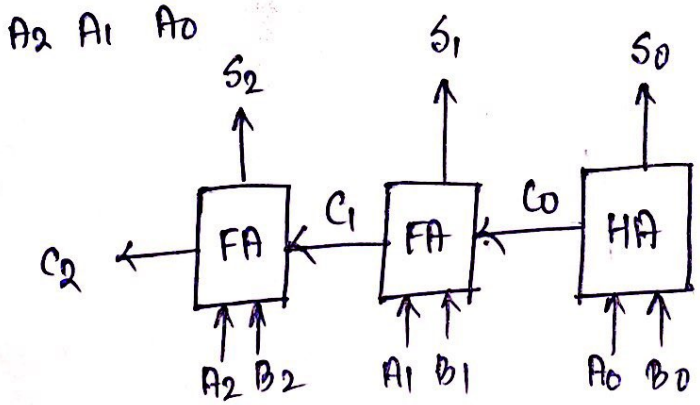
$S_i = S_i' \oplus C_{i-1}$

$S_i' \oplus C_{i-1}$



$C_i'' = S_i' \cdot C_{i-1}'$

3-bit no



$Sum = C_2 S_2 S_1 S_0$

03/9/19 — Digital Logic — Prof. Mitu Nasipuri

1st class test on Digital Logic & Circuit on 13th sep 2019
Syllabus: Upto Quine ~~Mee~~ Mcclusky method

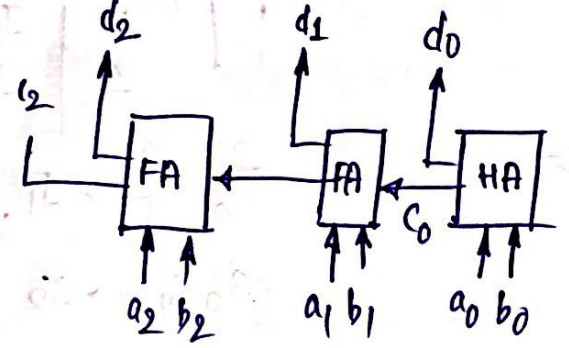
3-bit parallel

$a_2\ a_1\ a_0$

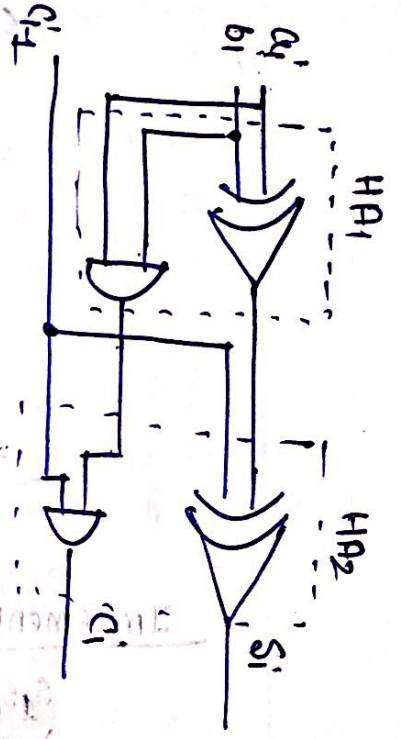
$b_2\ b_1\ b_0$

$c_1\ c_0$

$c_2\ d_2\ d_1\ d_0$



Look ahead Carry Generator



$PS_1 = a_1 \oplus b_1$

$PC_1 = a_1 \cdot b_1$

$c_1 = PS_1 c_0 + PC_1$

$PS_2 = a_2 \oplus b_2$

$PC_2 = a_2 \cdot b_2$

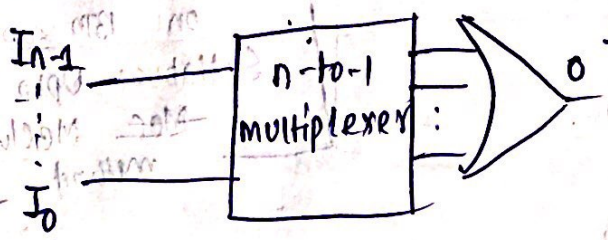
$c_2 = PS_2 c_1 + PC_2$

$PS_3 = a_3 \oplus b_3$

$PC_3 = a_3 \cdot b_3$

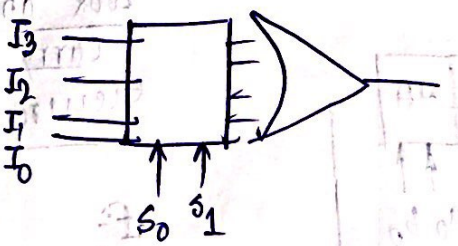
$C_3 = PS_3 \cdot C_2 + PI_3$

Implementation of any boolean fn using multiplexer.



4-to-1 multiplexer

S_1	S_0	O
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



$\overline{S_0} \cdot S_1 \rightarrow I_1$.

$S_0 \overline{S_1} \rightarrow I_2$

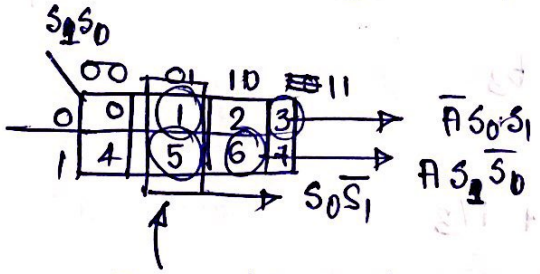
if $I_0 = 0$, $I_1 = 1$, $I_2 = 1$, $I_3 = 1$.

S_0 & S_1 fn as input

$I_0, \ldots, I_3$ control the output

OR-Gate

Implementation Table (4-1 multiplexer)



F has minterm at (x)

connected to A

$I_3 = \overline{A}$

$I_2 = A$

$I_1 = 1$

$I_0 = 0$

In a 2^n to 1 multiplexer a combinatorial logic circuit is designed which decodes the binary pattern on the selection lines $S_{(n-1)} \ldots S_0$ to enable an input line to get access to the output.

Since each multiplexer has an OR Gate at the output, minterms required to form any boolean fn can be made ~~avail~~ available by making the corresponding input lines high and remaining input lines to 0.

Method

i) For a boolean fn of (n-1) variables $A_n, A_{n-1}, \ldots, A_0$ ~~cannot~~ connect n variables $A_{n-1}, A_{n-2}, \ldots, A_0$ to n selection lines $S_{n-1} S_{n-2} \ldots S_0$ of a 2^n to 1 multiplexor. The remaining variable A_n or its component as input to its multiplexor.

ii) Prepare an implementation table containing 3 rows & 2^{n-1} columns. List the serial nos of the minterms corresponding to the first half of the truth table ($A_n = 0$) along the first row & the second half ($A_n = 1$) along the second row.

(iii) Circle the ~~min~~ minterms which are part of the boolean fn.

iv) Check each column ~~element~~ of the implementation table. If both entries are uncircled, set the corresponding input lines equal to 0.

→ If both the minterm are circled set the corresponding input line equal to 1.

→ If the minterm in the second row is circled connect the logical variable A_n to the corresponding input line.

→ If the minterm in the first row is circled connect the logical variable $\bar{A}_n$ to the corr. input line

④

Natural BCD (8-4-2-1) to Excess 3

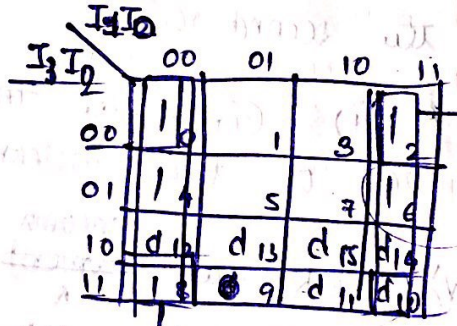
	I_3 I_2 I_1 I_0	8 − 4 − 2 − 1	Excess 3 (Add 3)
0 →	0 0 0 0		0 0 1 1
1 →	0 0 0 1		0 1 0 0
2 →	0 0 1 0		0 1 0 1
3 →	0 0 1 1		0 1 1 0
4 →	0 1 0 0		0 1 1 1
5 →	0 1 0 1		1 0 0 0
6 →	0 1 1 0		1 0 0 1
7 →	0 1 1 1		1 0 1 0
8 →	1 0 0 0		1 0 1 1
9 →	1 0 0 1		1 1 0 0

$\overline{B_3}\ \overline{B_2}\ \overline{B_1}\ \overline{B_0}$

$B_0 = \Sigma(0, 2, 4, 6, 8)$

$B_1 = \Sigma($

Doing it for other B_1, B_2 & B_3.



$I_1 I_0$ ✓

$I_3 I_2 . \overline{I_1}\,\overline{I_0}$ ✓

$\overline{I_1}\,\overline{I_3}$ ✓

$B_0 = \overline{I_1}\,\overline{I_3} + \overline{I_1}\,\overline{I_0} + I_1 I_0$

$= \overline{I_0}(1)$

$B_0 = \overline{I_1}\,\overline{I_2} + \overline{I_0}$ $\quad B_0 = \overline{I_0}$

does not matter

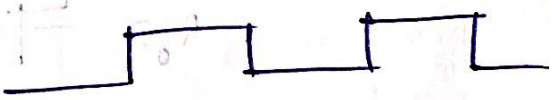
Sequential Circuits

The output of this type of circuit depends not only on the present inputs but also on the just previous state of the circuit.

The sequential logic circuits involve some sort memory element which can hold the previous state of the circuit and provide it as a feedback input to the circuit. 2 types of sequential logic circuits :—

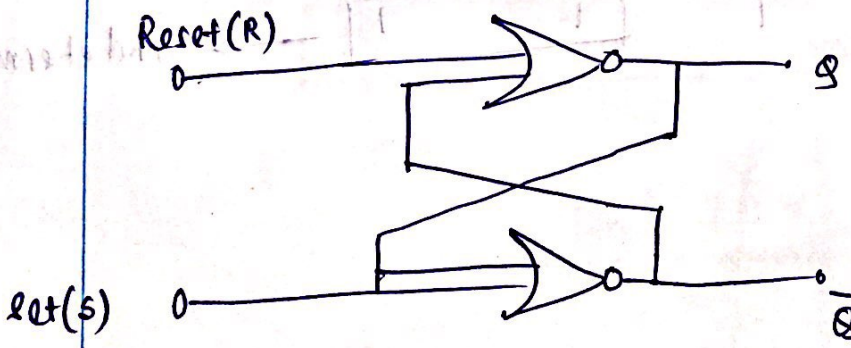
(i) Acynchronous : The state of the circuit changes when the input conditions change.

(ii) Synchronous :- The state of the circuit change at discrete time intervals. The type of the circuit require a synchronizing signal in the form of clock.



Flip - Flops

F/Fs are very important element in a sequential logic circuit as it can be used as a memory device capable of storing 1 bit information indefinitely until its input condn changes.



$Q_{(t)}$	R	S	$Q_{(t+1)}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	①

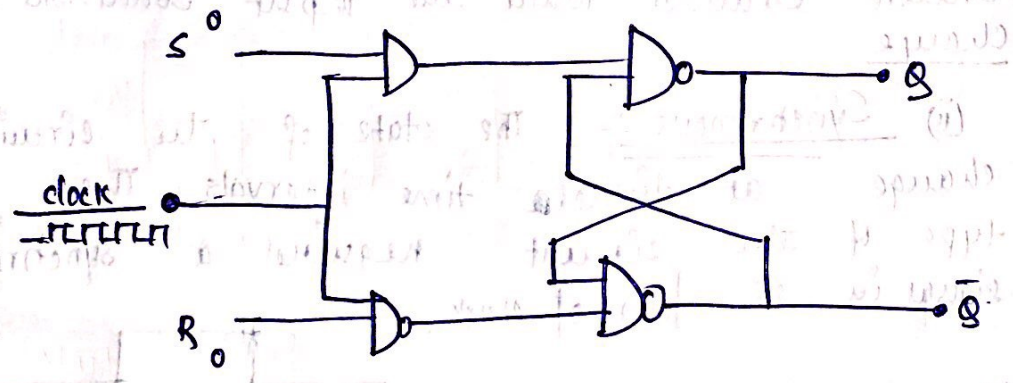
	$\bar{Q}_{(t+1)}$
Indeterminate form.	1
	0
	1

②

Q(t)	R	S	Q(t+1)	Q'(t+1)
1	0	0	1	0
1	0	1	1	0
1	1	0	0	1
1	1	1	0	0

Indeterminate form

Clock R-S implemented with NAND Gate



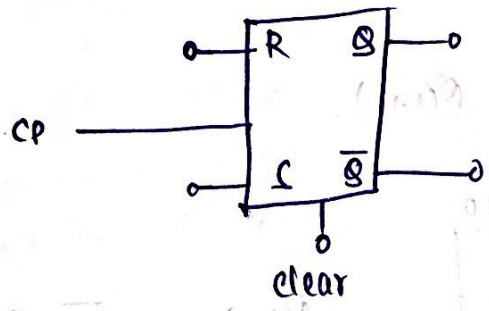
Q(t)	R	S	Q(t+1)	$\bar{Q}$(t+1)
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	1

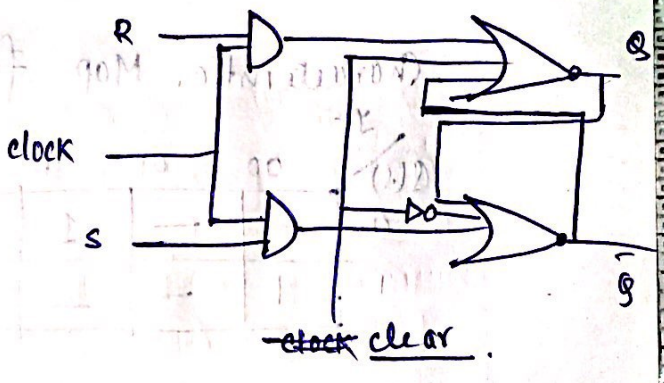
→ Indeterminate

$Q(t)$	R	S	$Q(t+1)$	$\overline{Q}(t+1)$
1	0	0	1	0
1	0	1	1	0
1	1	0	0	1
1	1	1	1	1

→ indeterminate

Logic symbol of RS (F/F) → flip flop

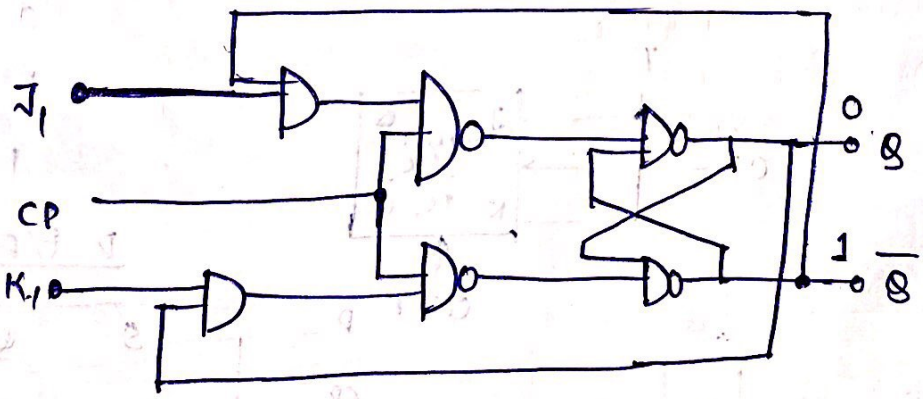




Rs / Q(t)	00	01	11	10
0	0	1	d	0
1	1	1	d	0

$Q(t+1) = Q_t \cdot \overline{R} + S$.

JK Flip Flop.



$Q(t)$	J	K	$Q(t+1)$	$\overline{Q}(t+1)$
0	0	0	0	1 → no change
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1 → complement

④

Q_t	J	K	$Q_{(t+1)}$	$\overline{Q}_{(t+1)}$	
1	0	0	1	0	no change
1	0	1	0	1	
1	1	0	1	0	
1	1	1	0	1	complemented (toggled)

Characteristic Map for $Q_{(t+1)}$.

$Q_{(t)}$ \ JK	00	01	11	10
0	–	–	1	1
1	1	–	–	1

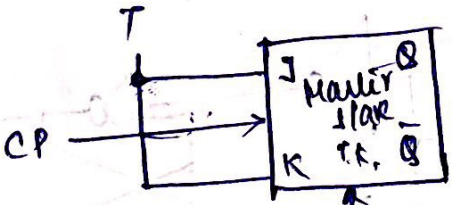
$$Q_{(t+1)} = \overline{Q_t}\, J + Q_t \cdot \overline{K}$$

Master - Slave JK F/F

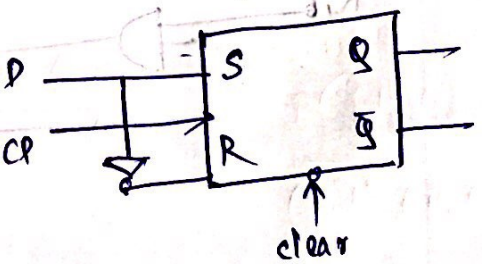
complicated circuit (Refer to Book)

J	K	Q	Q'
0	0	no change	
0	1	1	0
1	0	0	1
1	1	Toggle	

T F/F



D F/F



$Q_{(t)}$	D	$Q_{(t+1)}$
0	1	1
0	0	0
1	1	1
1	0	0

~~30/9/2019~~
14/9/2019

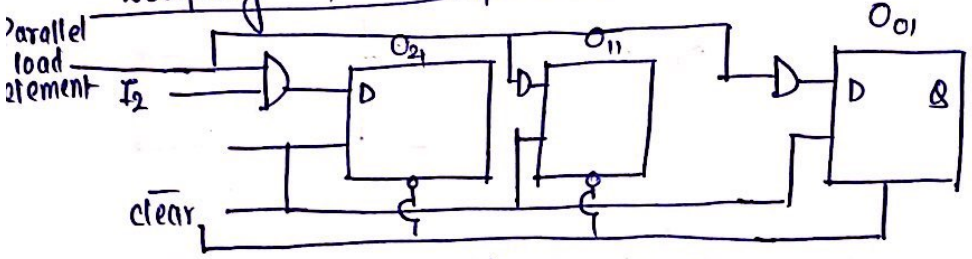
Digital Systems Lab

Prof. Mita Nasipuri

Registers

These constitute of a group of storage cells, each capable of storing 1 bit information for some computational purpose. Since F/F can temporarily hold 1 bit information, a group of F/Fs together with some combinational logic circuits form a register, the combinational logic circuit decides how binary n input to the register or output from it.

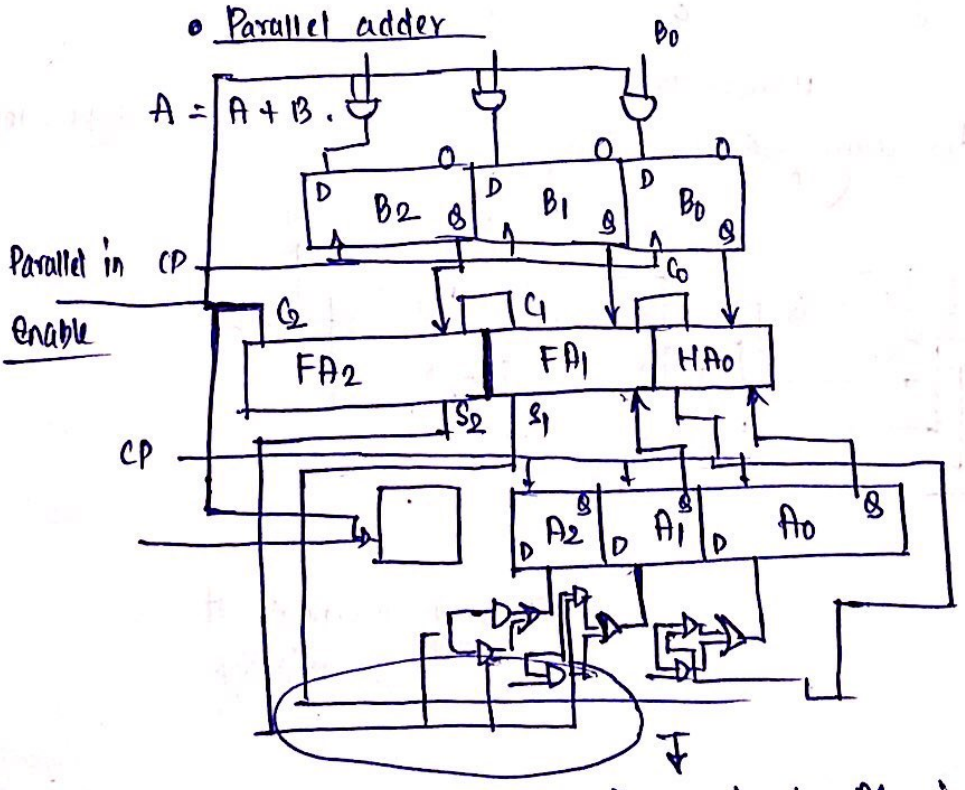
Designing of a register with D F/F



Parallel in Parallel out Register.

Use of parallel in parallel out register

- Parallel adder

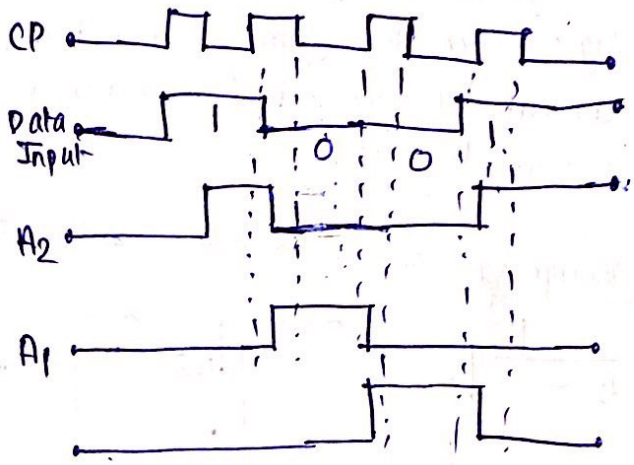


Please refer to Morris Mano

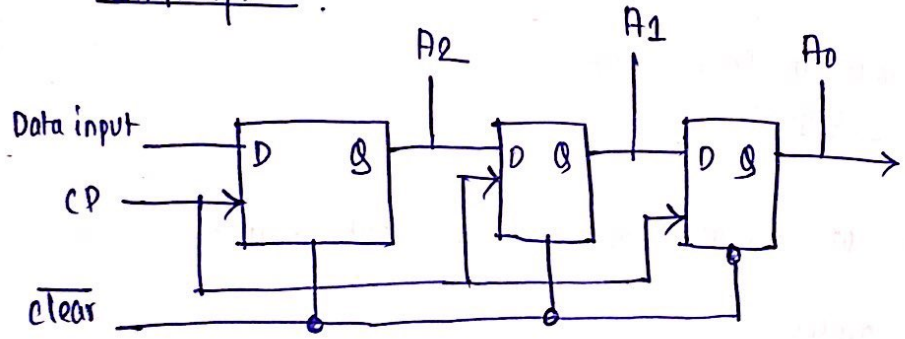
Shift Registers

It consists of F/Fs connected in cascade in a way so that it can shift 1 bit information either to the left or right (as set by control signal) at each clock pulse.

Data Input	1001
A_2	1001
A_1	0100
A_0	0010



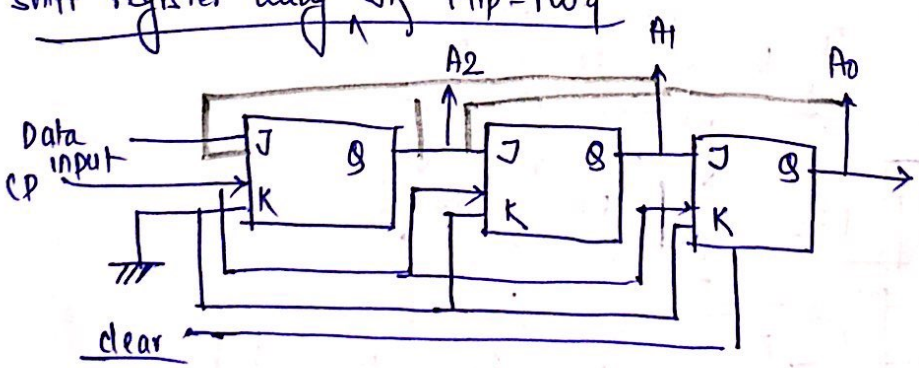
Shift right



clock width min to be 100

Master-Slave

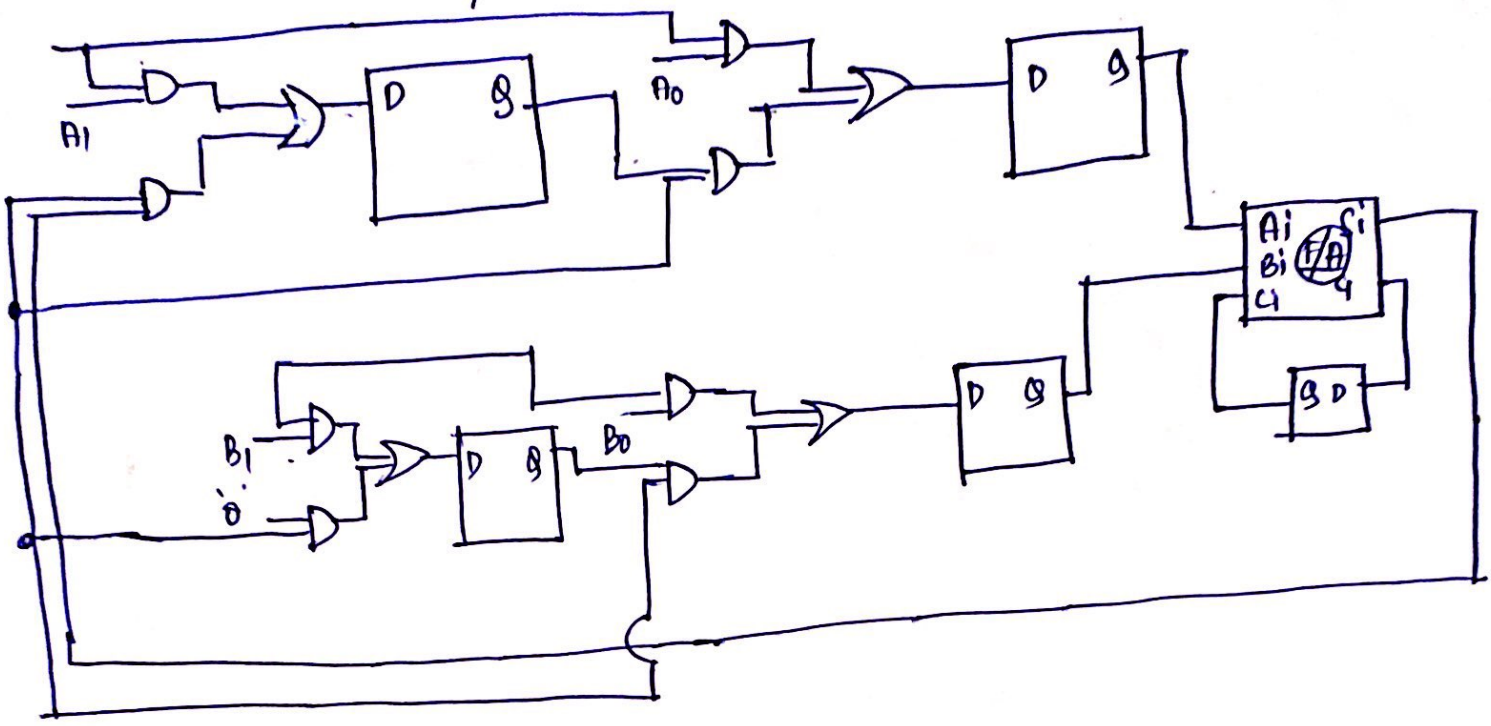
Shift register using JK Flip-flop



→ shift - left

Timing diagram of JK ralp flop

Serial adder.



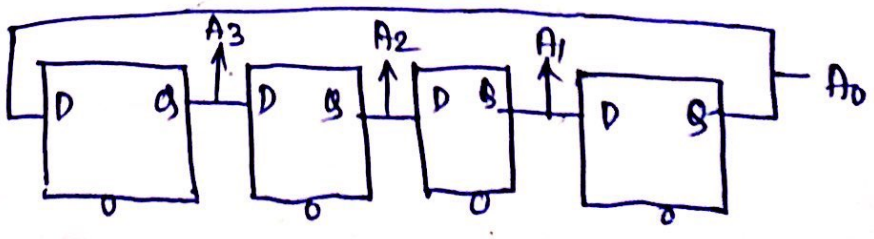
Initialize a circuit by making clear equal to '0' parallel in enable line is High.

CP1. Date A_1A_0 & B_1B_0 are loaded in A & B registers respectively, Next, the sum of A_0 & B_0 is computed resulting in sum bit S_0 & carry bit.

Application of shift registers

1) For binary multiplication/division (shift left/right) by a factor of 2
2) Circulating register/ring counter.

If the output of the last F/F of a n bit register is connected to the input of the first F/F, we get different states.



3) Switch tail ring counter complemented output of last F/F is connected to the input.

CP1: 0001
CP2: 1000
CP3: 0100
CP4: 0010
CP5: 0001

Digital Logic & Systems Lab

Generalized shift register (left/right) with parallel load

Implementation with 4 to 1 MUX

Diagram :- Refer to Morris Mano

$I_2\ I_1\ I_0$ input binary data is to be loaded into the register

1. clear resets the F/F in the register at 0.
2. clock pulse (CP) synchronizes the register operations.
3. when $S_1S_0 = 11$, data coming on the input lines $I_2\ I_1\ I_0$ is parallely loaded into the register
4. When $S_1S_0 = 00$ the register holds the same if the clock pulses are applied.
5. $S_1S_0 = 01$, shift right control is enabled
6. $S_1S_0 = 10$, shift left control is enabled.

Counters

Counters are formed with a group of F/Fs together with some combinational logic circuits and are able to count the no. of clock pulses coming into it.

2 types
1) asynchronous
2) synchronous

} up counter, down counter

up counting mode ↓

A_2	A_1	A_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1
0	0	0

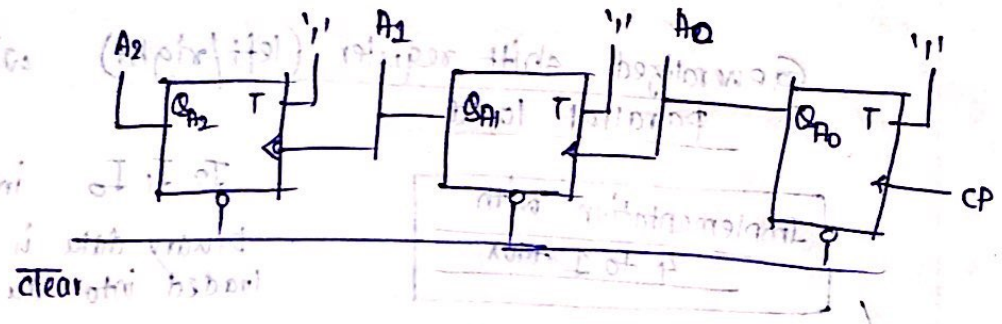
down counting mode

(similar to up-counting mode) but starts with 111 and goes till 000)

Starts with 000.

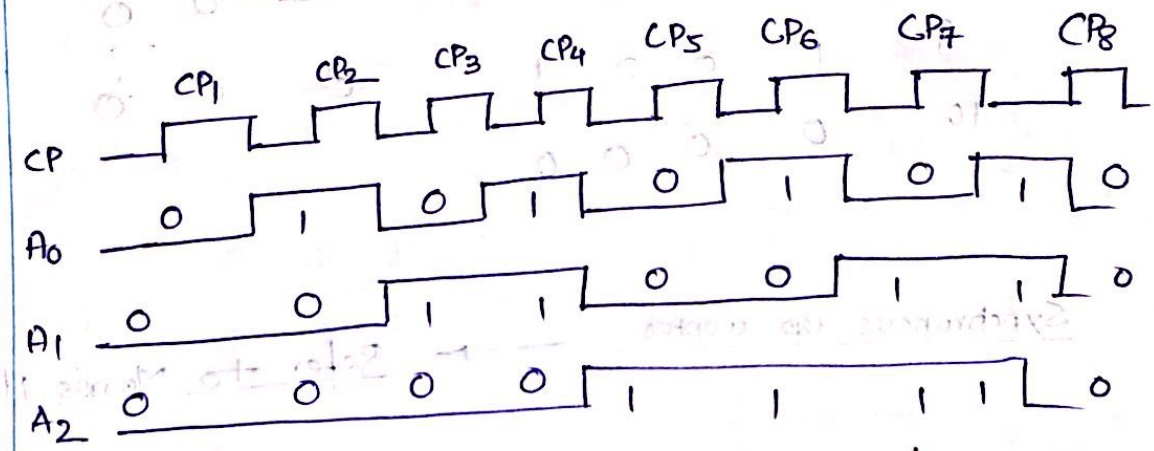
(2)

Asynchronous counter / Ripple counter.

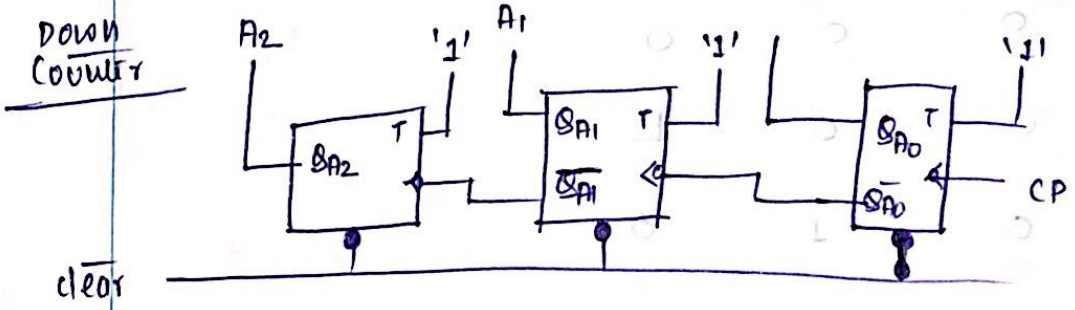


	A_2	A_1	A_0	Q_{A_2}	Q_{A_1}	Q_{A_0}
	0	0	0	0	0	0
CP_1	0	0	1	0	0	0
CP_2	0	1	0	0	1	0
CP_3	0	1	1	0	1	1
CP_4	1	0	0	1	0	0
CP_5	1	0	1	1	0	1
CP_6	1	1	0	1	1	0
CP_7	1	1	1	1	1	1
CP_8	0	0	0	0	0	0

Timing diagram for a 3-bit ripple up counter



Down Counter



$2^4 > 10 > 2^3$

Refer to Morris Mano

BCD ripple up counter

A_3	A_2	A_1	A_0	JA_1	JA_3	$\overline{QA_3}$
0	0	0	0	1	0	1
CP:1	0	0	0	1	1	0	1
CP:2	0	0	1	0	1	0	1
CP:3	0	0	1	1	1	0	1
CP:4	0	1	0	0	1	0	1
CP:5	0	1	0	1	1	0	1
CP:6	0	1	1	0	1	0	1
CP:7	0	1	1	1			

8	1	0	0	0	0	0	0
9	1	0	0	1	0	0	0
10	0	0	0	0			

Synchronous up counter → Refer to Morris Mano

A_2	A_1	A_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1
0	0	0

Home Task → Synchronous down counter

→ Design a ~~BCD~~ Binary modulo 6 ripple counter

↳ 6 diff. state

24/9/19 Digital Logic

Prof. Mita Nasipuri

(1)

Cyclic codes — here successively coded characters differ by one-bit posn

BCD →

0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0

Reflected code / gray code is one-type cyclic code.

Cyclic code

In the coding system, if we consider the three lower bit colums, we can see that the lower half of each of these columns are reflection of the corresponding upper half about the middle line. The highest order bit contains '0' in the upper half and '1' in the lower half.

Reflected BCD

0	0	0	0
0	0	0	1
0	0	1	1
0	0	1	0
0	1	1	0
0	1	1	1
0	1	0	1
0	1	0	0
1	1	0	0
1	1	0	1
1	1	1	1
1	1	1	0
1	0	1	0
1	0	1	1
1	0	0	1
1	0	0	0

middle-line

Reflected excess 3 BCD.

Advantage

Faster computation of 9's complement which can be obtained by complementing the highest order bit.

Error detection & correction

For the codes discussed so far, we cannot detect it any bit gets changed because this change will result in another valid codeword.

Distance b/w any 2 binary code words is the no of bit posn the code words differ.

Minimum distance of any coding system is the no. of bits that must be changed to generate a valid code word.

For normal BCD, minimum distance is 1.

Relationship b/w minimum distance (M) of any coding system and its Error Detection (D) & Error correction ability (C).

$$M - 1 = D + C$$

Since no errors can be corrected unless it is detected.

$$D \geq C$$

M	D	C
1	0	0
2	1	0
3	2	0
	1	1
4	3	0
	2	1
5	4	0
	3	1
	2	2

Minimum distance 2, single error detection ability

Even no. of 1s ⇒ 0

Even Parity (odd no. of 1s ⇒ 1)

Problem.

8	4	2	1	P	
0	0	0	0	0	0
0	0	0	1	1	1
		⋮			2
					⋮ ← 9
1	0	1	0	0	0

This is used instead.

Odd parity is preferred.

Another type of minimum distance 2 coding system

m out of n coding system

2 – out of 5 –

$^5C_2 = \frac{5 \times 4}{2} = 10$

0	1	2	4	7	
0	0	0	1	1	0
1	1	0	0	0	1
1	0	1	0	0	2
0	1	1	0	0	3
1	0	0	1	0	4
0	1	0	1	0	5
0	0	1	1	0	6
1	0	0	0	1	7
0	1	0	0	1	8
0	0	1	0	1	9

	0	1	2	3	6
	0	1	1	0	0
	1	1	0	0	0
	⋮				

other weighting systems

1	2	3	4	5
2	1	3	4	5

In all these cases decimal, 0 is improperly coded another weighting system is 8 4 2 1 0. which can code 8 decimal digits except 0 (coded as 10100) and 7 (coded as 11000).

Another type of fixed-bit code in beginning or qui-binary code which is 7-bit consisting of 1-out-of-2 and 1-out-of-5 group of bits.

bi-quinary

0	5	0	1	2	3	4
1	0	1	0	0	0	0
1	0	0	1	0	0	0
1	0	0	0	1	0	0
1	0	0	0	0	1	0
1	0	0	0	0	0	1

0 5	0 1 2 3 4	
0 1	1 0 0 0 0	5
0 1	0 1 0 0 0	6
0 1	0 0 1 0 0	7
0 1	0 0 0 1 0	8
0 1	0 0 0 0 1	9

qui-binary

0 1	0 2 4 6 8	
1 0	1 0 0 0 0	
0 1	1 0 0 0 0	
1 0	0 1 0 0 0	
0 1	0 1 0 0 0	
1 0	0 0 1 0 0	
0 1	0 0 1 0 0	
1 0	0 0 0 1 0	
0 1	0 0 0 1 0	
1 0	0 0 0 0 1	
1 0	0 0 0 0 1	

⑥

Hamming code (Minm distance 3 code)

If no. of bits in the code word be n. and let

$$2^{k-1} < n \leq 2^k$$

then the no. of redundant bits/check bits/ parity bits to be added to the original code word is k.

Construction of Hamming Code

① Mark all the bit positions in the resultant code word that are powers of 2 (1, 2, 4, 8, 16, 32, 64, ...) as the positions of check bits.

② Remaining posn are kept for binary data in the original code word (3, 5, 6, 7, 9, 10, 11, 13, 14, 15, 17, 18, 19, .)

③ Check bit C_1 is chosen to establish an even parity for int position (1, 3, 5, 7, 9, 11, 13 ...) of resultant code word.

④ Check bit C_2 is establish an even parity to bit posn (2, 3, 6, 7, 10, 11, 14, 15, 18, 19, 22, 23) of resulting cod. word.

⑤ C_4 .. (4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29)

⑥ C_8 (8 - 15, 24 - 31, 40 - 47)

⑦ C_{16} (16 - 31, 48 - 63, 80 - 95, ...)

⑧ C_{32} (32 - 63, 96 - 127, ...)

1001

$n = 4 = 2^2$.

no. of check bits (k) = 3.

c_1	c_2	b_1	c_4	b_2	b_3	b_4
↳ c_1	c_2	1	c_4	0	0	1

check bit c_1 :

c_1	—	b_1	—	b_2	—	b_4	
c_1	—	1		0		1	$c_1 = 1$

check bit c_2

—	c_2	b_1	—	—	b_3	b_4	
—	c_2	1	—	—	0	1	$\because c_2 = 0$

check bit c_3

—	—	—	c_4	b_2	b_3	b_4	
—	—	—	c_4	0	0	1	$\because c_4 = 1$.

hence → 0 0 1 1 0 0 1

At the receiving end,
check for c_4 which gives even parity over the bit posn (1, 3, 5, 7, ...). So the no. of 1's over bit posn (1, 3, 5, 7) is even. c_4 returns 0 else it return 1.

(8)

0 0 1 1 0 0 1

0 0 1 1 0 1 1

C_1 0 − 1 − 0 − 1 = 0

C_2 − 0 1 − − 1 1 = 1

C_4 − − − 1 0 1 1 = 1

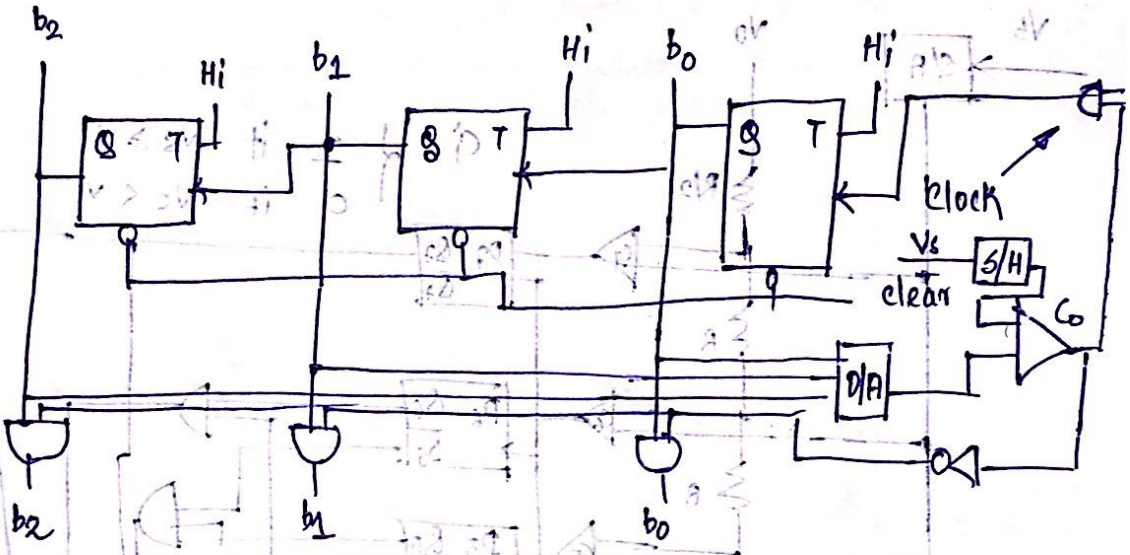
The posn of the errormow bit is $C_4 C_2 C_1 = 110$.

For a give no. of check bit C, the max. no. of information bit Imax is given by $Imax = 2^C - C - 1$.

18/10/2019 Digital Logic

Prof. Mita Nasipuri

Counting Type A/D Converter

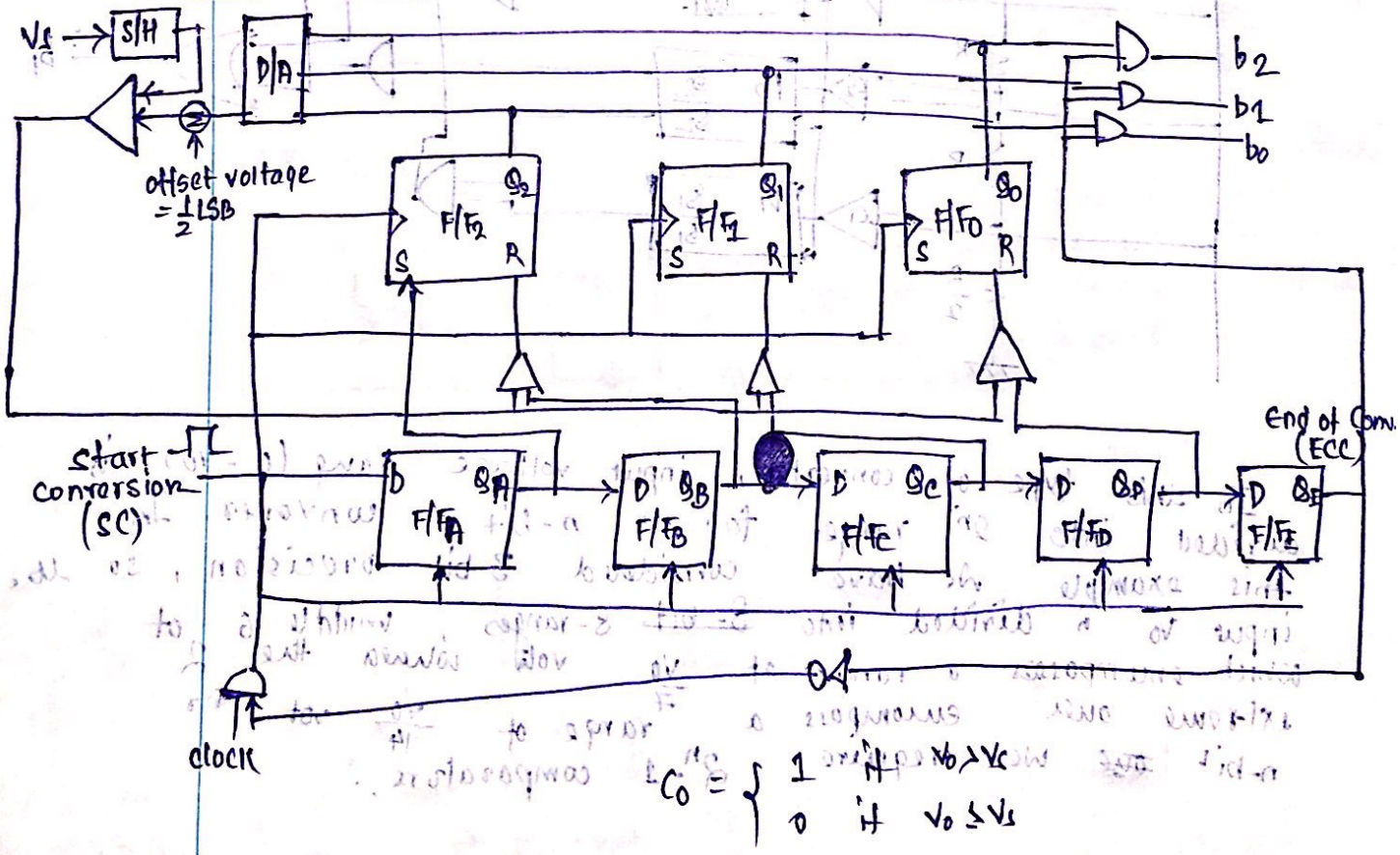


C_o = 1 if $V_o < V_s$

0 if $V_o \geq V_s$

Not feasible for practical applications

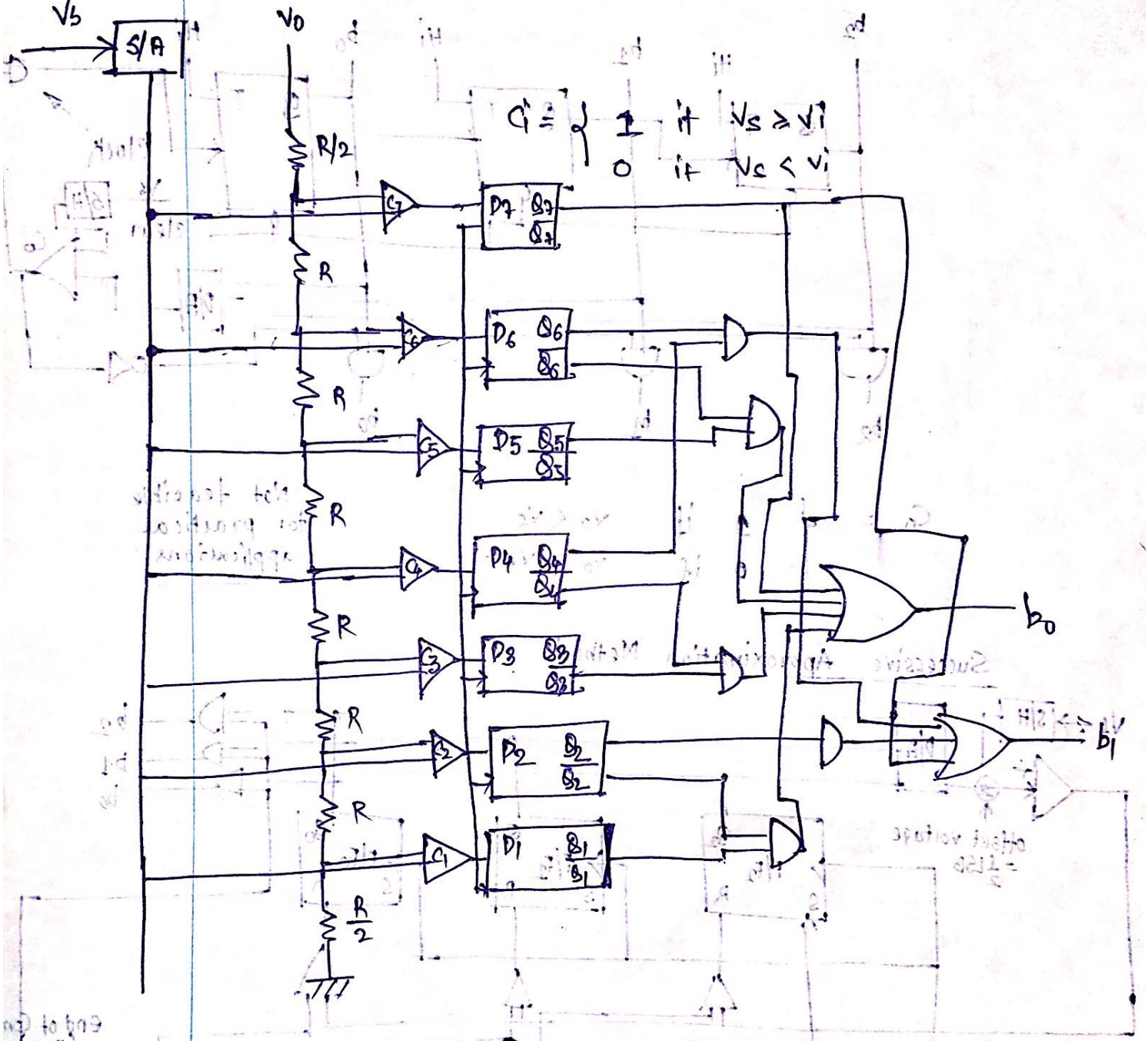
Successive Approximation Method



C_o = 1 if $V_o > V_s$

0 if $V_o \leq V_s$

(2) AD conversion using parallel comparators

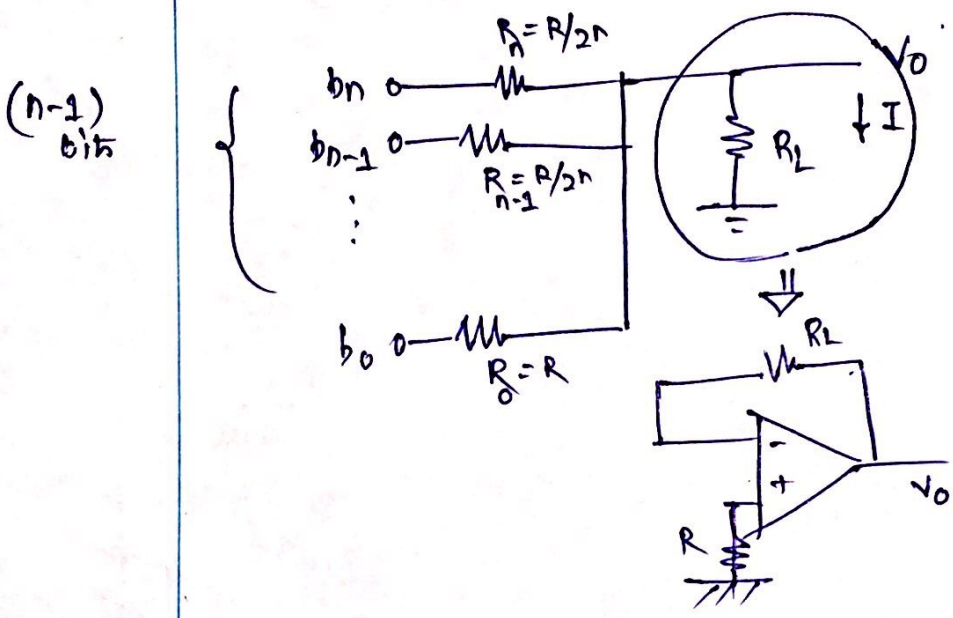


In this type of converter, input voltage range $(0 - V_0)$ is divided into 2^n ranges for a n-bit converter. In this example we have considered 3 bit precision, so the input V_0 is divided into ~~8-bit~~ 8-ranges, middle 6 of which encompasses a range of $\frac{V_0}{7}$ volt whereas the 2 extreme ones encompass a range of $\frac{V_0}{14}$ volt. For n-bit ~~use~~ we require $2^n - 1$ comparators.

This type of converter is very fast since the entire conversion ~~process~~ process takes place simultaneously in contrast to the previous types of converters which work sequentially.

Disadvantage :- Excessive hardware requirement which almost doubles when precision is increased by a single bit.

Digital to Analog Conversion



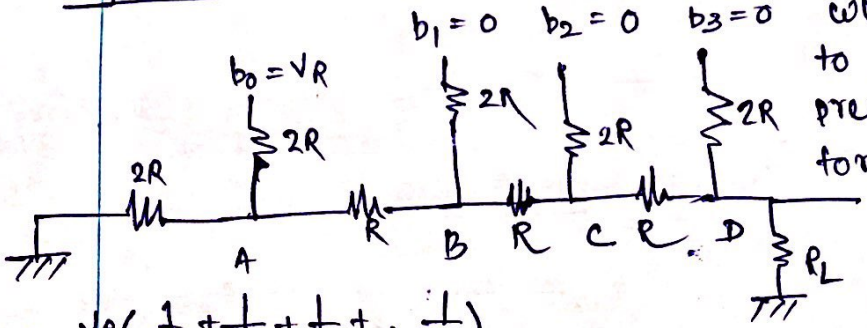
$$I = V_R\left(\frac{1}{R_L+R_0} + \cdots + \frac{1}{R_L+R_n}\right).$$

To overcome the effect of R_L we can use an op-amp

Disadvantage

Requirement of resistors with a wide range of value which are difficult to produce with high precision. It is also difficult for IC fabrication.

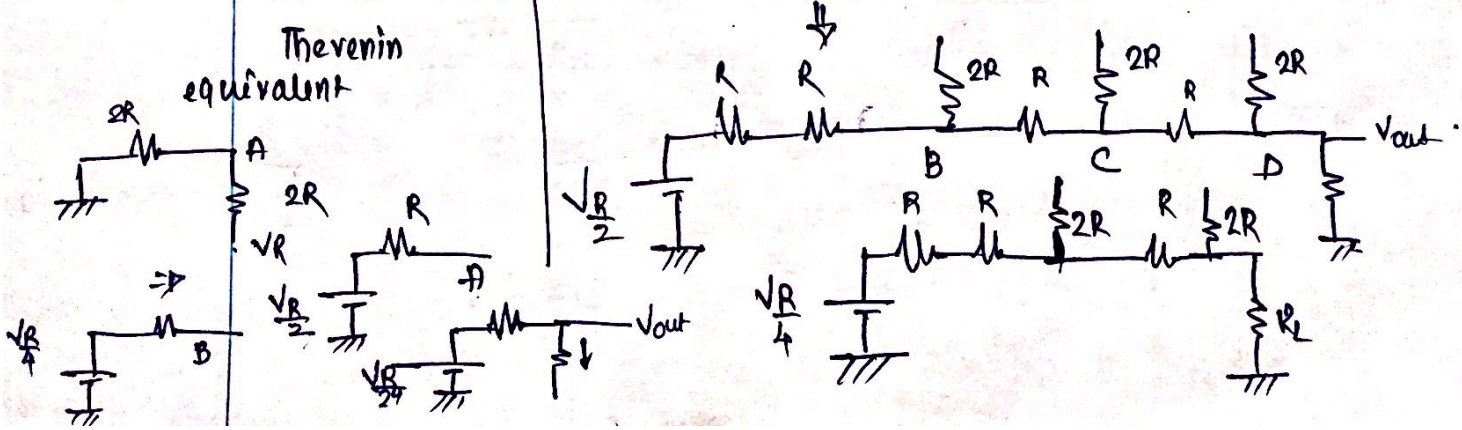
R-2R ladder DAC



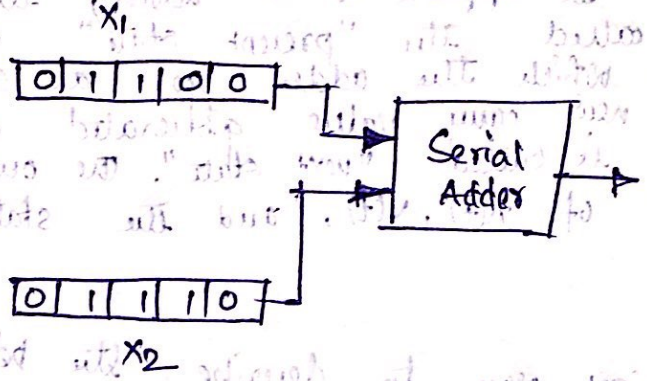
$$V_{out} = V_R\left(\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4}\right).$$

$b_0 = 1$
$b_1 = b_2 = b_3 = 0$

Thevenin equivalent



It is an abstract model for synchronous sequential machines. Let us consider a serial adder:-



It is a synchronous circuit with two inputs x_1 & x_2, and one output representing sum of x_1 & x_2.

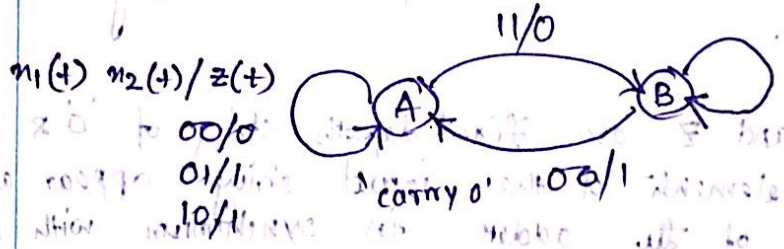
x_1, x_2 and z are fixed length string of '0' & '1'. Here the elements of the input string appear at the input of the adder in synchronism with a clock signal (t) and the elements of the output string also behave in the same way.

t_5	t_4	t_3	t_2	t_1	
0	1	1	0	0	x_1
0	1	1	1	0	x_2
1	1	0	0	0	z

In combinational circuit output depends completely on the inputs but in serial adder at t_1 & t_2, x_1, x_2 are 0,0 but sum is 0 in the case where it is 1, in the other case, same is true for that time instants t_3, t_4. Following the rules of binary arithmetic, the output of the at t_1 is a function of inputs x_1, x_2 at t_2, and a carry generated at t_2 i.e. the past history of the adder. Thus, the adder will have 2 types of states of past histories 'carry 0' & 'carry 1'. These are called adder circuit

Let A represent the state of the adder at t_i when a carry 0 is generated at t_{i-1}, and B be the state corresponding to a carry 1 generated at t_{i-1}. When the inputs $x(t)$ & $y(t)$ is applied to the adder, the state which is called the "present state" and the state to which the adder goes as a result of the new carry value generated as a result of addn is called "next state". The output $z(t)$ is a fn of $x(t)$, $y(t)$, and the state of the adder.

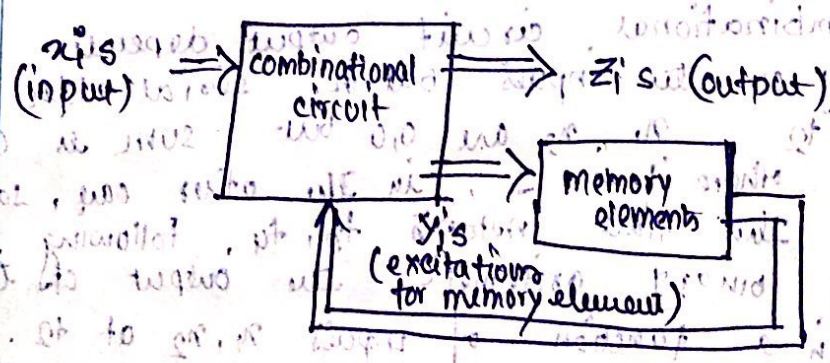
A convenient way to describe the behaviour using directed graph which is called "state diagram"



Another way of representation is through "data table"

Present state	$x_1x_2 = 00$	01	11	10
A	A, 0	A, 1	B, 0	A, 1
B	A, 1	B, 0	B, 1	B, 0

Generalized diagram of a sequential circuit



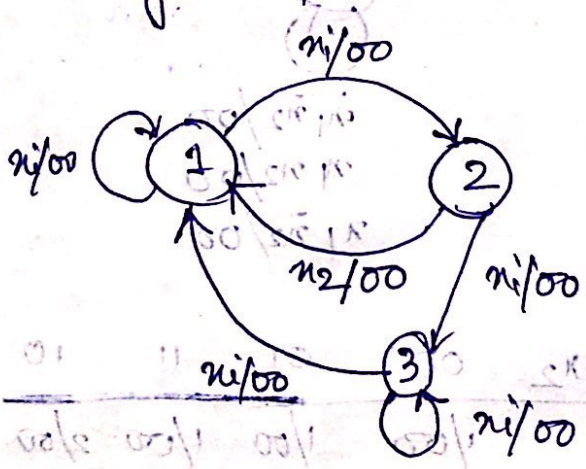
x_i's are inputs
z_i's are outputs
y_i's " excitations for memory elements
s_i's " state information

} can assume '0, 1'

x_i's may be levels or pulses. If all of them are pulses, there must be some ~~synchronous~~ synchronism among themselves, i.e. at least they must arrive at the same time. If some of them are pulses, then each of the pulse are naturally ANDed with or all the level inputs.

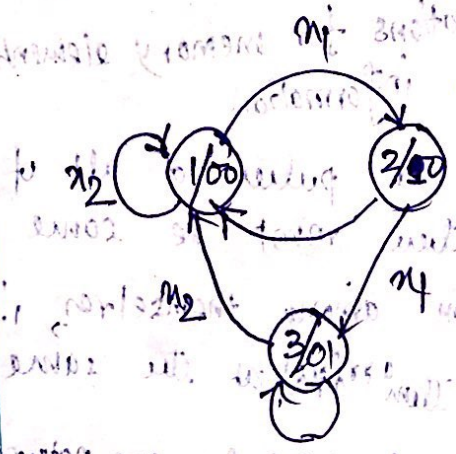
If there is a single pulse input which are ANDed with all level inputs, the pulse input is called 'clock'. Generally levels are logic outputs and clock provide the instants at which the logic levels are sampled. If z_i's are pulses, they are associated with the transitions from one state to another for your x_i's. If z_i's are levels. they are associated with circuit state s_i's.

Example state diagrams and corresponding state table when a single input is active at a time instant.



	x_1	x_2
1	2/00	1/00
2	3/10	1/00
3	3/00	1/01

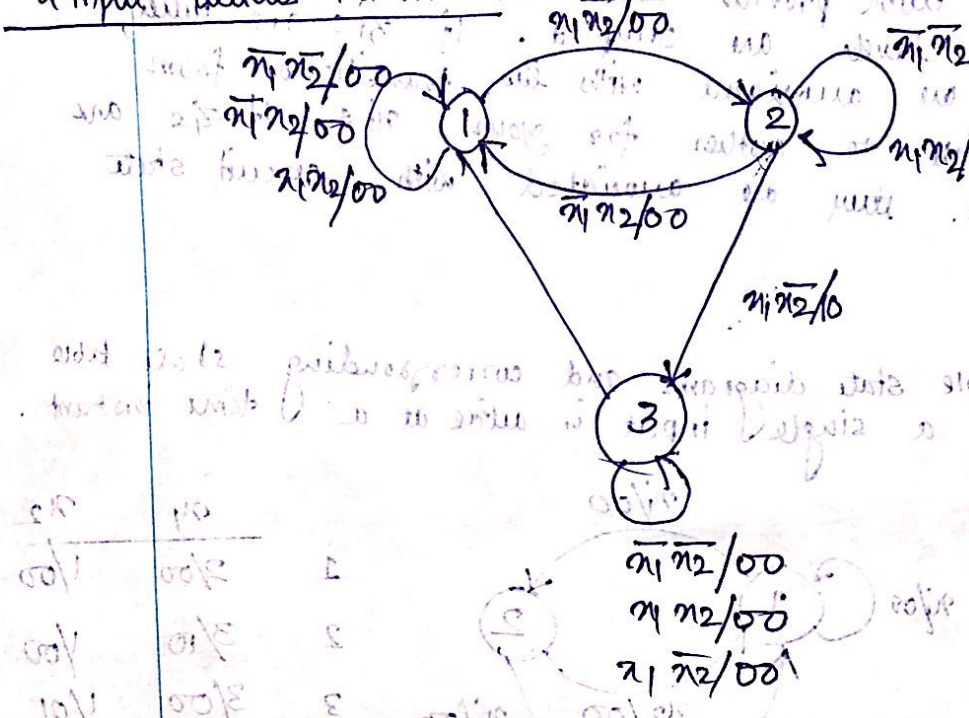
for level output-



	x_1	x_2	$z_1 z_2$
1	2	1	00
2	3	1	10
3	1	1	01

$x_1 x_2/00$
$x_1 x_2/00$
$x_1 x_2/00$

when a combination of input decides transitions.



$x_1 x_2$	00	01	11	10
1	1/00	1/00	1/00	2/00
2	2/00	1/00	2/00	3/10
3	3/00	1/01	3/00	3/00

(2) Finite State Models

Here we are focusing on deterministic machines which possess the property that, the next state of the machine is uniquely determined by the present state $s(t)$ and present input $x(t)$.

$$s(t+1) = \delta\{s(t), x(t)\}$$

where δ is the state transition fn.

Again, the value of $z(t)$ may also be a fn of $s(t)$ & $x(t)$ i.e. $z(t) = \lambda\{s(t), x(t)\}$ where λ is the output fn.

A machine following the above 2 properties are called 'Mealy Machine'.

Another type of machine called 'Moore Machine' is the one where the output function is dependent only on the present state.

$$z(t) = \lambda\{s(t)\}$$

A formal defn of synchronous sequential machine is as follows: A synchronous sequential machine M is the quintuple $M = (x, z, s, \delta, \lambda)$.

where x, z, s are finite non-empty sets of inputs, outputs and states respectively.

$\delta: x \times s \to s$ is the state transition function (the cartesian product $x \times s$ contains all pairs (x_i, s_i))

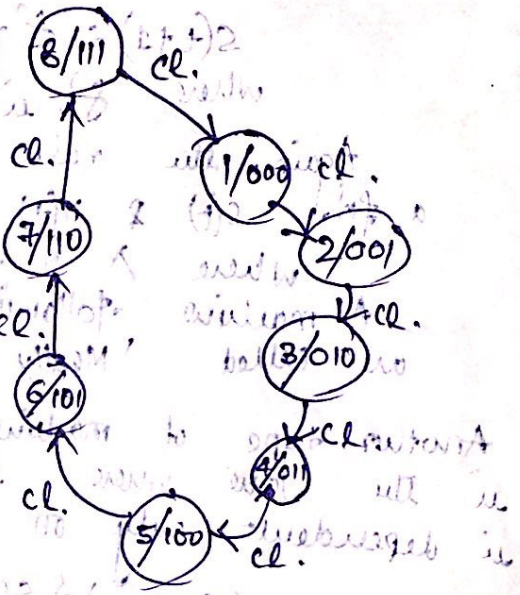
$\lambda: x \times s \to z$ for mealy machines.

$\lambda: s \to z$ for Moore machines.

Examples of Moore machine

Synchronous counter

0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1



Present State (PS)			Next State (NS)		
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

J-K state excitation requirement

Q(t)	Q(t+1)	J	K
0	1	1	–
1	0	–	1
1	1	–	0
0	0	0	–

Excitation maps for F/Fs

Q_3 \ Q_2Q_1	00	01	11	10
0	1	–	–	1
1	1	–	–	1

$J_1 = 1$

Q_3 \ Q_2Q_1	00	01	11	10
0	0	1	–	–
1	0	1	–	–

$J_2 = Q_1$

Q_3 \ Q_2Q_1	00	01	11	10
0	–	1	1	–
1	–	1	1	–

$K_1 = 1$

Q_3 \ Q_2Q_1	00	01	11	10
0	–	–	1	0
1	–	–	1	0

$K_2 = Q_1$

Q_3 \ Q_2Q_1	00	01	11	10
0	0	0	1	0
1	–	–	–	–

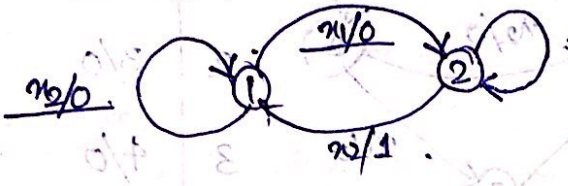
$J_3 = Q_2Q_1$

–	–	–	–
0	0	1	0

$K_3 = Q_2Q_1$

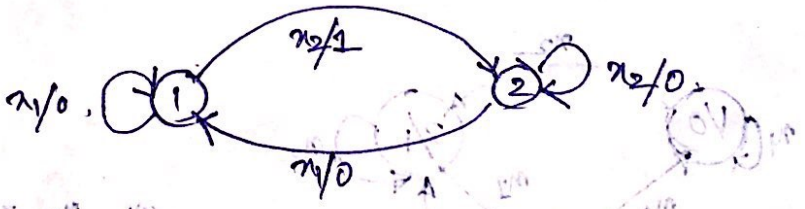
State diagram and state table from word statement

① An output pulse is ~~consistent~~ coincident with the first x_2 pulse immediately following an x_1 pulse.

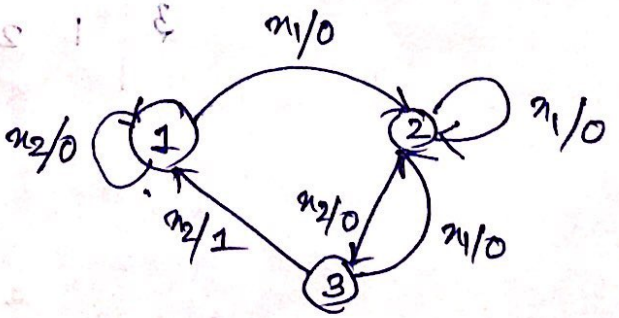


	x_1	x_2
1	2/0	1/0
0	2/0	1/1

② An output pulse z is coincident with the first of a sequence of x_2 pulses.

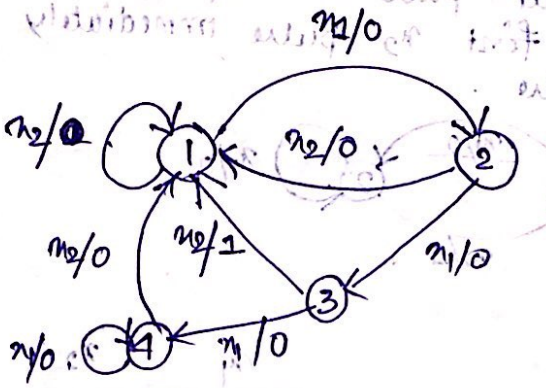


③ An output pulse, z is coincident with the second of a sequence of x_2 pulses following an x_1 pulse.



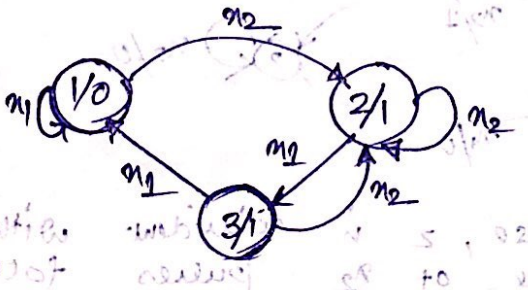
	x_1	x_2
1	2/0	1/0
2	2/0	3/0
3	2/0	1/1

(4) An output pulse z is coincident with the first x_2 pulse immediately following exactly 2 x_1 pulses.

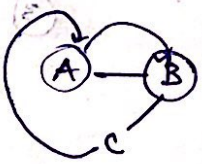


	x_1	x_2
1	2/0	1/0
2	3/0	1/0
3	4/0	1/1
4	4/0	1/0

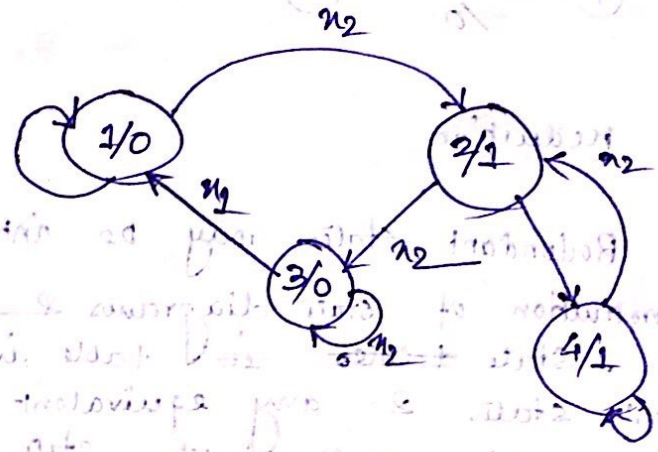
(5) x_1 & x_2 are pulse input & z is a level. output z is to be turned on with an x_2 pulse and to be turned off with the second of a sequence of x_1 pulses. No other input sequences cause any change in z.



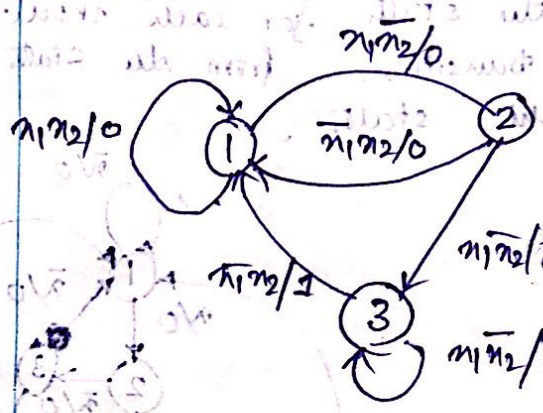
	x_1	x_2	z
1	1	2	0
2	3	2	1
3	1	2	1



⑥ x_1 & x_2 are pulse inputs & z is a level output. z is to be turned on with the first of a sequence of n_2 pulses & to be turned off with the second of a sequence of n_2 pulses. No other input sequences cause any change in circuit operation.

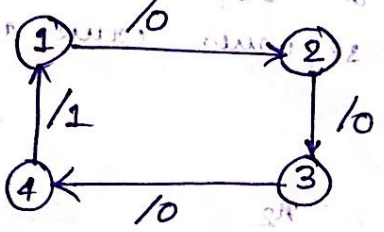


⑦ A sequential circuit has 2 level inputs x_1, x_2 & a clock. An output pulse (z) is coincident with a clock pulse occuring with $x_1x_2 = 01$ immediately following two/more consecutive clock pulses with $x_1x_2 = 10$, $x_1x_2 = 00$ & $x_1x_2 = 11$ never occurs.



	x_1x_2 00	01	11	10
1	—	1/0	—	2/0
2	—	1/0	—	3/0
3	—	1/1	—	3/0.

(8) A sequential circuit has one clock & the output pulse z is co-incident with every 4th clock pulse.



State table reduction

~~Redundancy~~ Redundant states may be introduced during construction of state diagrams & corresponding state tables. State table is tested for any equivalent states & any equivalent states are found, corresponding rows in the state table are removed.

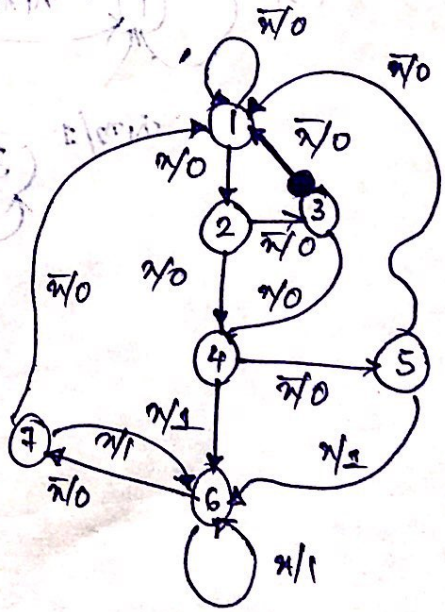
In a state table, 2 states are equivalent if

1) output's state pulse/level associated with both states are same.

2) For each of the states, for each input pulse, there is a transition from the state to the same/equivalent state.

State Table

	x=0	x=1
1	1/0	2/0
2	3/0	4/0
3	1/0	4/0
4	5/0	6/0
(5)	1/0	6/1
6	7/0	6/1
(7)	1/0	6/1



From the table states 5 & 7 are equivalent. Therefore one of them may be removed. ~~But we~~ Let us remove state 7. Any reference to state 7 will be replaced by state 5.

	$x=0$	$x=1$
1	1/0	2/0
2	3/0	4/0
3	1/0	4/0
(4)	5/0	6/1
5	1/0	6/1
(6)	5/0	6/1

$\Rightarrow$

	$x=0$	$x=1$
1	1/0	2/0
2	3/0	4/0
3	1/0	4/0
4	5/0	4/1
5	1/0	4/1

no. of F/Fs required will be n such that

$$2^{n-1} < r \leq 2^n$$

r = no. of rows in the reduced state table.

~~3/~~ 3 F/Fs

$\therefore$ ${}^8C_3 : 51$

Possible state assignment

$${}^{2^n}C_r \cdot r!$$

Universal Map Method

In this method, for each combination of pulse inputs/clocked inputs to memory element, a single memory element excitation map is formed which is independent of the type of memory element. This map is called universal Map. The universal Map required 4 symbols, one for each combination of $S(t)$ & $S(t+1)$ plus one optional symbol.

⑥

S(t)	S(t+1)	Map-Symbol	S	R	J	K	T	D
0	1	I	1	0	1	—	1	1
1	0	∅	0	1	—	1	1	0
1	1	1	—	0	—	0	0	1
0	0	0	0	—	0	—	0	0

optional

Example :- Consider the following state table with state anignment

S_1	S_2	n_1	n_2
0	0	11/0	00/0
0	1	01/0	00/0
1	0	10/0	00/1
1	0	10/0	00/0

universal excitation maps

$S_1 = n_1 \cdot (\bar{S_1} + \bar{S_2})$

$R_1 = n_1 S_1 S_2 + n_2$

Memory element, 1 state bit S_1

S_1 \ S_2	0	1
0	I	I
1	1	∅

n_1

S_1 \ S_2	0	1
0	0	0
1	∅	∅

n_2

Memory element- 2 state bit S_2

S_1 \ S_2	0	1
0	I	∅
1	0	1

S_1 \ S_2	0	1
0	0	∅
1	0	∅

05/11/2019

Digital Logic and Circuits

Prof. Mita Nasipuri

Ex: An output pulse is coincident with the first x_2 pulse immediately following two x_1 pulses

	x_1	x_2
1	2/0	1/0
2	3/0	1/0
3	4/0	1/1
4	4/0	1/0

2 memory elements required for state representation

	S_1	S_0
1	0	0
2	1	1
3	0	1
4	1	0

state table with state assignment

	x_1	x_2
00	11/0	00/0
11	01/0	00/0
01	10/0	00/1
10	10/0	00/0

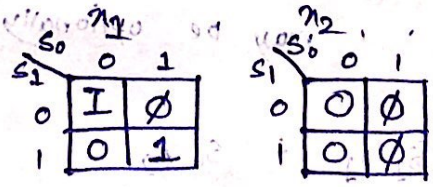
universal map method

$0 \rightarrow 1$ I

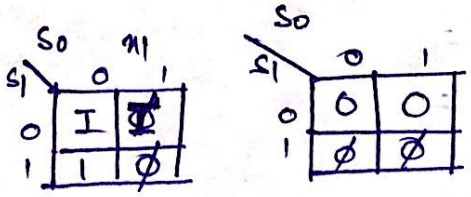
$1 \rightarrow 0$ ϕ

$1 \rightarrow 1$ 1

$0 \rightarrow 0$ 0



Memory element 0



Memory element 1

→ For SR F/F

→ Every I must be considered for S expression
" Ø " " " R expression

Any 1 or — may be considered optionally for S expression

Any 0 or " " " R expression

$S_0 = \bar{S}_0 \bar{S}_1 x_1$

$R_0 = S_0 x_2$

$S_1 = \bar{S}_1 x_1$

$R_1 = S_0 S_1 x_1 + x_2$

$z = \bar{S}_1 S_0 x_2$

Case → 00/1

For JK F/F

Every I must be considered for J expression
Every Ø " " K expression

Any Ø or 1 may be optionally considered for J exp.
Any 1 or 0 may be optionally for K exp.

$J_0 = \bar{S}_1 x_1$

$K_0 = \bar{S}_1 x_1 + x_2$

$J_1 = x_1$

$K_1 = S_0 x_1 + x_2$

$z = \bar{S}_1 S_0 x_2$

For T F/F

Every 1 & Ø must be considered for T-expression
Any '-' may be optionally used for "

For D F/F

Every I & 1 must be considered for D expression

Any '-' may be optionally ...

$T_0 = \bar{S}_1 x_1 + S_0 x_2$

$T_1 = (\bar{S}_1 + S_0) x_1 + S_1 x_2$

$D_0 =$

$D_1 =$

Eg. A sequential circuit has two level inputs x_1 & x_2. and a clock. An output pulse z is coincident with a clock pulse occuring with $x_1x_2 = 01$ immediately following two or more consecutive clocks with $x_1x_2 = 10$ $x_1x_2 = 00$ & $x_1x_2 = 11$ never occurs

State table

	x_1x_2 00	01	11	10
1	—	1/0	—	2/0
2	—	1/0	—	3/0
3	—	1/1	—	3/0

1 - 00
2 - 01
3 - 10

	00	01	11	10
00	—	00/0	—	01/0
01	—	00/0	—	10/0
10	—	00/1	—	10/0

S_1S_0 \ x_1x_2	00	01	11	10
00	—	0	—	I
01	—	∅	—	∅
11	—	—	—	—
10	—	0	—	0

Memory element 0

S_1S_0 \ x_1x_2	00	01	11	10
00	—	0	—	0
01	—	0	—	I
11	—	—	—	—
10	—	∅	—	1

Memory element 1