DIGITAL SYSTEMS

& MICROCONTROLLERS

- Tutorial Test/Quiz (20%)
- Mid-Sem (20%)
- End-Sem (30%)
- Lab (30%) (20% + 10%)
  End Sem   Report Submission

*] Digital Number Systems:

1] Decimal - $(0 \rightarrow 9)$ Base 10   b/w 2^3 & 2^4 : 4 used as 3 insufficient

2] Binary - (1,0)   Base 2

3] Octal - $(0 \rightarrow 7)$ Base 8 $\Rightarrow 2^3$ possibilities : 3 bits sufficient

4] Hexadecimal - $(0 \rightarrow 9, \underset{10}{A} \rightarrow \underset{15}{F})$ Base 16 $\rightarrow 2^4$: 4 bits

•] Representation:

Ex:1] $(1475)_{10}$ (digits marked $x^3\, x^2\, x^1\, x^0$)

$$= 1\times10^3 + 4\times10^2 + 7\times10^1 + 5\times10^0$$

Ex:2] $(453)_5$ × no.s inside ( ) have to be less than base

Ex:3] $(1011)_2$

$$= 1\times2^3 + 0\times2^2 + 1\times2 + 1\times2^0 = 8+0+2+1 = \underline{\underline{11}}$$

Ex:4] $(11.0101)_2$

$$= 1\times2^1 + 1\times2^0 + 0\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 1\times2^{-4}$$

$$= 2 + 1 + 0 + 0.25 + 0 + 1/16$$

*] Base Conversions:

Ex: $(345)_{10} = (\quad)_2 = (\quad)_8$

		remainder
2	345	
2	172	1
2	86	0
2	43	0
2	21	1
2	10	1
2	5	0
2	2	1
2	1	0
		1

write upwards

until quo^n = 0

10

$= (1011001)_2$

→ IIly, for 8, 16, etc.

Ex: $(5.642)_{10} = (\quad)_2 \Rightarrow 101.xyz$

separately treat

Now for decimal as n^{-1} × instead of ÷

required base

$0.642 \times 2 = 1.254 \rightarrow 1$

$0.254 \times 2 = 0.568 \Rightarrow 0$

$0.568 \times 2 = 1.136 \Rightarrow 1$

$\Rightarrow (5.642)_{10} = (101.101)_2$

→ If integer formed terminate process

Octal to Binary:

Ex: $(345)_8$

011 100 101 as octal 3 bits $\Rightarrow (345)_8 = (011100101)_2$

→ & vicé versa # if decimal not grouping into 3 add zeroes

Hexa to Binary:

Ex: $(345)_{16}$

0011 0100 0101 as hexa 4 bits $\Rightarrow (345)_{16} = (1101000101)_2$

→ & vicé versa # if not grouping into 4 add zeroes

*] BINARY ARITHEMATIC:

1] Addition:

10 → 4 = 100 to be carried over as such

```
  110
  1110
+ 1110
+ 1110
+ 1110
------
111000
```

∴ 111000

2] Subtraction:

⊛ Do not mistake it as 10

When borrowing from multiple zeroes make 2 & borrow 1 from it making 1 from 2 & taking 1

010 → 10 (read as 2 ∵ binary)

```
  1110
- 1011
------
  0011
```

∴ 1110 − 1011 = 11

3] Division:

```
       10101 → Quotient
101 | 1101101
     -101
      00111
      - 101
       01001
       - 101
         100 → remainder
```

∴ 1101101 = 10101 × 101 + 100

4] Multiplication:

```
     1101
   ×  101
   ------
     1101
+   00000
+  110100
  -------
  1000001
```

∴ 1101 × 101 = 1000001

→ Octal & Hexa Arithematic exactly same logics.

_/_/__

*] COMPLEMENTS:

1] Raddix (r):

- for digit N with n bits with base r:

$$\boxed{r = r^n - N}$$

2] Diminished Raddix (r-1):

- for digit N with n bits with base r:

$$\boxed{(r-1) = (r^n - 1) - N}$$

•] Use:

Ex: $(4567)_{10}$ ⇒ 10's : $10^4 - 4567$ → not easy to calc. (Raddix)

⇒ 9's : $(10^4 - 1) - 4567$

= 9999 − 4567 = 5432 → now just +1 to get 10's complement

Diminished Raddix

⊛ $(04567)_{10}$ ⇒ 10's : $10^5 - 4567$

Ex: $(10101101)_2$

⇒ 2's : $2^8 - N$ = 1010011

⇒ 1's : $(2^8 - 1) - N$

= 11111111
− 10101101
= 01010010

NOTE: $2^x - 1 = \underbrace{11\ldots11}_{x}$

•] Subtraction using complements:

→ for M − N do:

1] M + r's complement of N ⇒ $M + (r^n - N)$

2] M > N → discard the carry (last wala only)

3] M < N: $r^n - (N-M)$ ⇒ r's complement of (N−M) (Ex: on next pg)

Ex: 72532 − 3250 (M = 72532, N = 3250)

⊛ Equalize no. of digits first ⇒ 03250

⇒1] 9's : $(10^5 - 1) - 3250$

= 99999 − 3250 = 96749 ⇒ 10's = 96750

⇒ 72532
+ 96750
= ~~1~~69282

⇒ 72532 − 3250 = 69282

__/__/__

Ex: 3250 - 72532

10's : 27468 → 03250

+ 27468

30718

* Now answer is -ve of 10's of → 30718

⇒ -69282

Ex: $(1010100)_2 - (1000011)_2$

→ 1010100 ↙ 2's

= 0111101

*0010001 ⇒ $(10001)_2$

Ex: $(1000011)_2 - (1010100)_2$

→ 1000011 ↙ 2's

+ 0101100

1101111 $\xrightarrow{2's}$ $-(0010001)_2$

*] SIGNED & UNSIGNED SYSTEM:

	Unsigned	Signed	
Sign Bits when system declared { 0 10101	21	+21	→ 0 = +ve
1 10101	53	-21	1 = -ve

*] BCD: BINARY CODED DECIMALS:

→ Arithematic using signed systems

Ex: +6 } always use ⇒ 00000110
+13 } 8 bits + 00001101
19 00010011

Ex: -6 11111010 (2's of +6)
+13 * 00001101
7 00000111 (discard carry)

-6 -13 → 11111010

* 11110011

-19 = 11101101 → Cross checking: 2's of 19 = 11101101

→ When you arrive at answer if it starts with 1 its the 2's complement and hence specify

→ If number itself is 8 bits take 4 more

P.T.O.

__/__/__

→ In BCD each digit to be represented binary

Ex: $16 = (10000)_2 = (0001\ 0110)_{BCD}$

*] Ex-3: Excess 3 Codex:

→ +3 to all digits i.e. 0 = 0011 (as in 3 in binary)
lly, each digit represented

•] BCD Arithematic:

1] Addition:

Ex:
```
 764            0111   0110   0100
+856    =>     +1000   0101   0110
----           ---------------------
1620     1     1111   1011   1010
             + 0110   0110   0110
        0001   0110   0010   0000
          ↓      ↓      ↓      ↓
          1      6      2      0
```
→ These digits can't be represented using BCD thus add 6 ~~to them to shift them to~~ the next system
only that can't be represented

Ex:
```
 56    =>    0101  0110
+44         +0100  0100
---         -----------
100          1001  1010
           +    1  0110
            -----------
             1010  0000
           + 0110
      0001   0000  0000
```
yaha kabhi 18, 17, 16 ban raha ~~ho to~~ take 5 bits

} as 1010 again can't be represented

⊛ No discarding carries

∴ 0001 0000 0000

2] Subtraction:

Ex: 1001 0011 − 0101 0110

→ Use radix complement method, but ⊛ find 10's compliment of each 4-bit digit,

- 9's complement of a BCD number can be obtained by subtracting it from 9.
- 10's = 9's + 1

•] 10's complement method: A − B

1] Find 10's of B.

2] Add A & 10's complement of B.

3] Discard carry.

4] If carry not produced, no. is −ve take 10's of result

5] Validate BCD no. by +6 if necessary.

→ 10's of 0101 0110 :
```
  1001          1001
 -0101         -0110
  0100          0011
 +   1         +   1
  0101          0100     = 0101 0100
```
⊛ Don't do this first find 9's:
```
  0100 0011
+ 0000 0001
  0100 0100
```
```
→   1001  0011
  + 0100  0100
    1101  0111
  + 0110
  ✗ 0011  0111
```
→ validation ke baad −ve check

∴ 1001 0011 − 0101 0110 = 0011 0111

__/__/____

*] LOGIC GATES:

→ AND, OR, NOT, NAND, NOR, XOR, XNOR, etc.

→ self-explainatory

↳ Universal Gates: can be used to form every other gate.

•] Closure:

- A closure set for an operator is the set of which when elements operated upon gives result from the same set.

•] Associative law: $(x \square y) \square z = x \square (y \square z)$

•] Commutative law: $x \square y = y \square x$

•] Identity element:

- Depends on operator: $x \square e = x$

  ⇒ for $*$ $e = 1$ & $+$ $e = 0$

•] Inverse: $x \square y =$ Identity element

•] Distributive law: $x \square (y * z) = (x \square y) * (x \square z)$

*] BOOLEAN ALGEBRA:

- Given in 1854 by George Boole → Boolean
- Two-valued boolean algebra given by Claude E Shannon 1938

•] EV Huntington Postulates of 1904:

① $+$ & $*$ are closure in boolean algebra.

② Identity elements: $+ \longrightarrow 0$, $* \longrightarrow 1$

③ $x + y = y + x$, $xy = yx$

④ $x(y+z) = xy + xz$

⑤ $x + \bar{x} = 1$, $x\bar{x} = 0$

⑥ $\exists$ 2 elements $\in B$ s.t. $x \neq y$

•] Associative law:

- $(x \square y) \square z = x \square (y \square z)$

★ •] Distributive law:

$$[\, x + (y * z) = (x+y) * (x+z) \,]$$

→ Boolean Algebra does not have additive/multiplicative inverses due to the absence of subtraction & division.

__/__/____

∘] **Duality Principle:**

→ $x + 0 = x \longleftrightarrow x \cdot 1 = x$, i.e. operator change can be carried out through changing identity.

∘] **Postulates:**

1] $x+0 = x$ & $x \cdot 1 = x$

2] $x+\bar{x} = 1$ & $x\bar{x} = 0$ (He writes x' so prefer x' over $\bar{x}$)

3] $x+y = y+x$ & $x \cdot y = y \cdot x$

4] $x(y+z) = xy + xz$ & $x + (yz) = (x+y)(x+z)$ ∵ $(x+y)(x+z)$

$= xx + xz + xy + yz$

$x\underbrace{(1+z)}_{1} + xy + yz$

$x + yz \Leftarrow x + xy + yz$

∘] **Theorems:**

1] $x + x = x$ & $x \cdot x = x$

2] $x + 1 = 1$ & $x \cdot 0 = 0$

3] $(\bar{\bar{x}}) = x$

4] $x + (y+z) = (x+y) + z$ & $x(yz) = (xy)z$

De-Morgan 5] $\boxed{\overline{(x+y)} = \bar{x} \cdot \bar{y} \quad \& \quad \overline{(xy)} = \bar{x} + \bar{y}}$

Absorption 6] $x + xy = x$ & $x(x+y) = x$

∘] **Operator Precedence:**

1] Parantheses

2] NOT

3] AND

4] OR

Q] Simplify $\bar{x}\bar{y}z + xyz + \bar{x}yz + x\bar{y}z$.

→ [redacted] $\bar{x}\bar{y}z + yz\cancel{(x+\bar{x})}^{1} + x\bar{y}z$

$= \bar{x}\bar{y}z + yz + x\bar{y}z \Rightarrow \bar{y}z\cancel{(\bar{x}+x)}^{1} + yz$

$= yz + \bar{y}z \Rightarrow z\cancel{(y+\bar{y})}^{1} = z$

∴ $\underline{\bar{x}\bar{y}z + xyz + \bar{x}yz + x\bar{y}z = z}$

__/__/____

*] Complement of a Function:

- $F = A+B+C$

$\Rightarrow F' = (A+B+C)' = A'B'C'$

Ex: $F = xyz + x'yz \Rightarrow F' = (x'+y'+z')\cdot(x+y'+z')$

*] Canonical Form:

Ex:

x	y	z	Canonical form using min terms	Canonical Forms using Max terms
0	0	0	$x'y'z' \rightarrow m_0$	$x+y+z \rightarrow M_0$
0	0	1	$x'y'z \rightarrow m_1$	$x+y+z' \rightarrow M_1$
0	1	0	$x'yz' \rightarrow m_2$	$x+y'+z \rightarrow M_2$
0	1	1	$x'yz \rightarrow m_3$	$x+y'+z' \rightarrow M_3$
1	0	0	$xy'z' \rightarrow m_4$	$x'+y+z \rightarrow M_4$
1	0	1	$xy'z \rightarrow m_5$	$x'+y+z' \rightarrow M_5$
1	1	0	$xyz' \rightarrow m_6$	$x'+y'+z \rightarrow M_6$
1	1	1	$xyz \rightarrow m_7$	$x'+y'+z' \rightarrow M_7$

⊛ Truth Table of ex func is: 0 1 0 0 1 0 0 1

$\Rightarrow F_1(\text{minterms}) = x'y'z + xy'z' + xyz$

$= m_1 + m_4 + m_7$

Truth value of F=1

$\therefore F_m = \sum(m_1, m_4, m_7)$

Illy, $F_M = \prod(M_0, M_2, M_3, M_5, M_6)$

Truth value of F=0

↳ To derive write $(F_m')'$

•] Conversion to canonical form:

Ex: $F = xy + x'z$ as product of max terms

$\Rightarrow F = (xy + x')\cdot(xy+z) \quad \because (x+yz = (x+y)\cdot(x+z))$

$= (x+x')\cdot(x'+y)\cdot(x+z)\cdot(y+z)$

⊛ $= (x'+y+zz')\cdot(x+z+yy')\cdot(z+y+xx')$

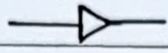
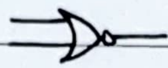
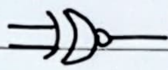
$= \underset{M_4}{(x'+y+z)}\cdot\underset{M_5}{(x'+y+z')}\cdot\underset{M_0}{(x+y+z)}\cdot\underset{M_2}{(x+y'+z)}$

Ex: $F = xy + x'z$ as sum of min terms

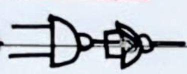
⊛ $= xy(z+z') + x'z(y+y')$

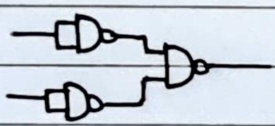
$= \underset{m_7}{xyz} + \underset{m_6}{xyz'} + \underset{m_3}{x'yz} + \underset{m_1}{x'y'z}$

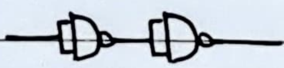
*] Logic Gates contd.:

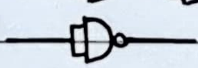
Gate	Representation	Output
AND (·)		00 0 01 0 10 0 11 1
OR (+)		00 0 01 1 10 1 11 1
Buffer/Data		0 0 1 1
NOT		0 1 1 0
NAND (↑)		00 1 01 1 10 1 11 0
NOR (↓)		00 1 01 0 10 0 11 0
XOR ⊕ Non-equivalence		00 0 01 1 10 1 11 0
⊙ XNOR () equivalence		00 1 01 0 10 0 11 1

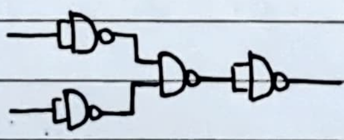
o] All gates using NAND:

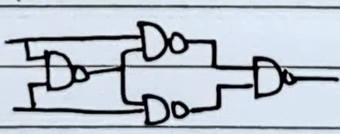
1] AND - 

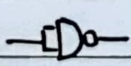
2] OR - 

3] Buffer - 

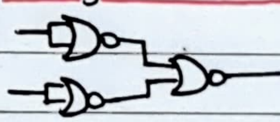
4] NOT - 

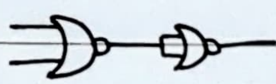
5] NOR - 

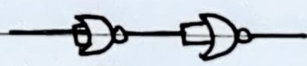
6] XOR - 

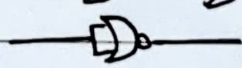
7] XNOR - XOR 

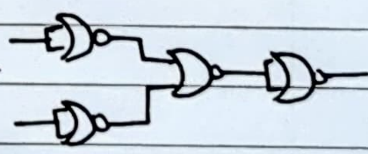
o] All gates using NOR:

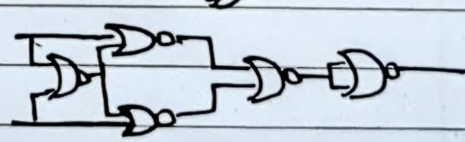








5) NAND - 



XOR