BLG 231E

(Digital Circuits)

analog signals : continuous, infinite resolution

digital signals : discrete, two possible values
(1-0, true-false, on-off, high-low)

digital coding

- n bits → 2^n different "things"
- Binary coded decimal (BCD) → (0-9 → four bit)

  805 = 1000 0000 0101

- Positional (weighted) coding

  $11010 = 1.2^4 + 1.2^3 + 0.2^2 + 1.2^1 + 0.2^0 = 26$

  most significant bit (MSB) ↓ ... ↓ least significant bit (LSB)

- Hamming distance : number of bit positions in which they are different. (011 and 101 ⇒ h.d = 2)
- Adjacent codes : a pair codes that hamming d. = 1

▽ if the hamming distance between the first and last 0 code word is 1 in a coding system then this code is called <u>circular.</u>

<u>Gray Code</u> : binary, non-redundant and circular
(n bits - 2^n things)

- integers are represented in computers by "natural binary weighted (positional) coding".

▽ repr. of signed ⇒ MSB → 0 ⇒ positive
0 → 1 ⇒ negative
↓
2's complement system
(invert the number and add 1)

<u>extension</u>

- unsigned ⇒ 4-bit $3_{10} = 0011$ → 8-bit $3_{10} = 0000\ 0011$
- signed ⇒ 4-bit $-7_{10} = 1001$ → 8-bit $-7_{10} = \underbrace{1111}\ 1001$
filled with the sign of MSB (sign extension)

Hexadecimal Numbers

- (0-9) and (A-F) → base 16 system
- $01011101_2 = 5D_{16} = (5.16) + 13 = 93_{10}$
  ↳ (usually #5D or 5Dh)

Arithmetical Operations

- addition of two n-bits can be (n+1)-bits then $(n+1)^{th}$ bit → carry (=0 ⇒ no carry)
- addition of two n-bits signed if $(n+1)^{th}$ bit arises it should be ignored.
- in signed integers if:

  pos + pos → neg and neg + neg → pos

  ⇒ there is an <u>overflow</u> and $(n+1)^{th}$ bit is ignored.

- substraction of two n-bit signed if $(n+1)^{th}$ bit is 0 the first operand is smaller than the second one and there is <u>borrow</u> (no carry)
- pos − neg → neg or neg − pos → pos ⇒ <u>overflow</u>

Boolean Algebra

* the dual of $a + a \cdot b$ is $a \cdot (a+b)$ and they are equal.

Logical Functions

1) simple (basic) functions → multiple inputs, single output

→ for n binary variables there are $2^{(2^n)}$ possible basic logical functions.

2) general functions → multiple inputs, multiple outputs

3) incompletely specified functions

→ they have unspecified outputs for some input combinations

Representation of Logical functions

1) truth table representation
2) numbered (indexed) representation

truth table:

row num	input x_1 x_2	output y
✓ 0	0 0	1
1	0 1	0
✓ 2	1 0	1
3	1 1	

numbered (indexed):

$y = f(x_1 x_2) = \cup_1 (0,2)$

"union" or "set of"

$\cup_1$ → set of 1-generating points

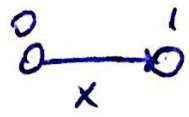
Graphical Representation

n - bits input → n-dimensional cube

(1's are marked)

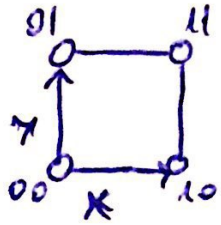
1-dimensional

f(x)



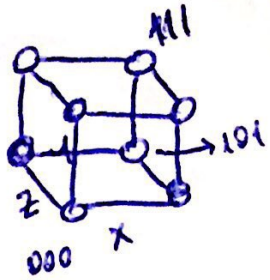
2-dimensional

f(x,y)



3-dimensional

f(x,y,z)

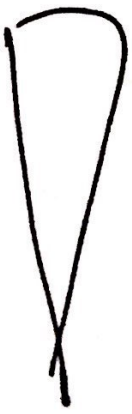


* the variables and outputs transferred into a table that is in a matrix form and rows and columns labeled according to the Gray Code property. → Karnaugh Maps

F

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

→ Gray code



absorbtion → $E + E.F = E$, $E.(E+F) = E$

$E + \bar{E}.F = E+F$, $E.(\bar{E}+F) = E.F$

consensus → $x_1E_1 + \bar{x}_1E_2 + E_1E_2 = x_1E_1 + \bar{x}_1E_2$

$(x_1+E_1).(\bar{x}_1+E_2).(E_1+E_2) = (x_1+E_1).(\bar{x}_1+E_2)$

Canonical forms

1st) Sum of Products → SOP ($\Sigma\Pi$)

(minterms)
→ sum of products, each of which corresponds to a "1" - generating combination.

Minterm : a product of n literals (in a n-variables function) in which each variable appears exactly once (a or a').

→ abc, a'bc, a'bc', ab'c' → 3 variables
4 minterms

* indexing minterms → we assign the indexes of minterms like m0, m1, m2 ... and we can represent the function:

$F(A,B,C) = \Sigma m(1,3,5,6,7)$ for example.

2nd) Products of Sum → POS ($\Pi\Sigma$)

(maxterm)
→ products of sum, each of which corresponds to a "0" - generating combination.

Maxterm : the difference of minterm is each maxterm has a value of "0".

! while finding maxterms → we substitute variable for zeros and complements of veriables for ones. ($A \equiv 0$, $A' \equiv 1$)

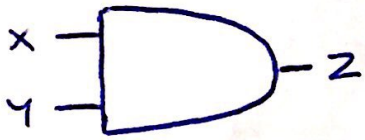
* to indexing maxterms we use M and b represent like $F(A,B,C) = \Pi M(0,2,4)$

Logic Gates

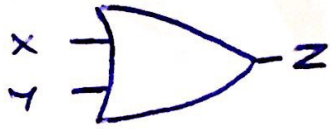
Buffer ($Y = X$)

Inverter ($Y = \bar{X}$)

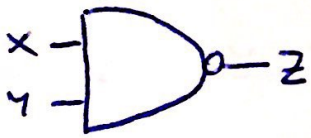
And ($Z = X \cdot Y$)



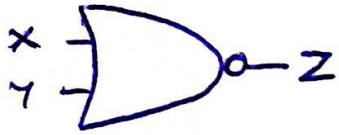
Or ($Z = X + Y$)



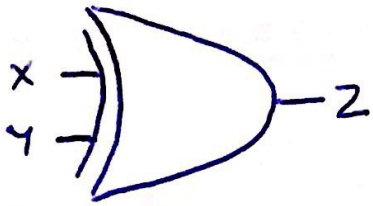
Not And (NAND) ($Z = \overline{(XY)}$)



Not Or (NOR) ($Z = \overline{(X+Y)}$)

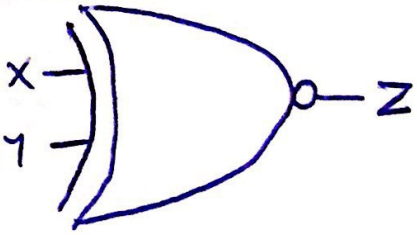


Difference (XOR) ($Z = X\bar{Y} + \bar{X}Y$)



→ $Z = X \oplus Y$

Equality (XNOR) ($Z = XY + \bar{X}\bar{Y}$)



→ $Z = X \odot Y$

1 = high value and 0 = low value ⇒ positive logic

1 = low value and 0 = high value ⇒ negative logic

NAND and NOR gates are called **universal logic gates.**

Simplification of Logic Functions

* objectives of minimization are decreasing
  - the size of circuit
  - power consumption
  - the delay (increasing the speed)
  - the cost

* **prime implicant** : implicants that connot be simplified any further.

* **the set of all prime implicants** : all possible prime implicants, groups of all 1's.

* **sufficient base (sufficient covering sum)** : the minimum set of prime implicants that covers all 1's.

* **distinguished points** : 1's that may be covered only by a single prime implicant.

* **essential prime implicant** : prime implicant that covers a distinguished point. (must be included in the minimal covering sum)

Simplification

1) find the set of all prime implicants
2) select a subset with minimum cost that covers the function

* Prime Implicant Chart

1) assign simple symbols to each prime implicants (A, B, C)
2) calculate the cost of each prime implicent with the criteria
3) symbols → rows (put * on intersection)
   1 points → columns
   costs → last column
   } make a matrix
4) simplificate the prime implicant chart
   1) single X in a column → distinguished point
      the row of this point and columns that are covered are removed.
   2) if row i covers row j and the cost at i is smaller or equal to the cost at j, then row j is removed.
   3) if column a covers column b, then column a is removed.

! when we select the don't care values (ⓓ)
→ we can take the values as 1 in Karnaugh Map.
→ we can take the values as 0 in the prime implicant chart.

0

Tabular (Quine - McCluskey) method

* we put true points and don't cares in a truth table.

if only one variable changes between two minterm, we merge them.

we do that until there is no minterms that can be merged.

therefore, prime implicants are the minterms that can't merged with the other ones.

Simplification in the PoS form

* we use the same process with SoP simplification but there is only one difference : we group zeros ▽₀

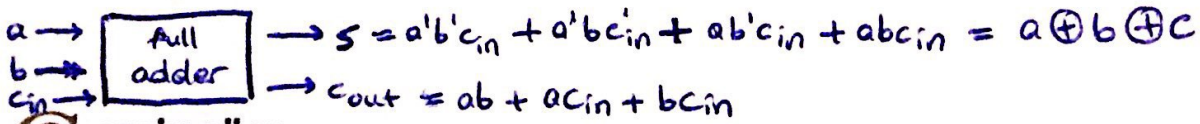
end of dc 04 ——— ——— end of dc04

* adders, multiplexers, decoders are common structures that used building blocks in larger systems. they are manufactured and sold as integrated circuits (ICs).

* generations of ICs according to integration scale → 5.1

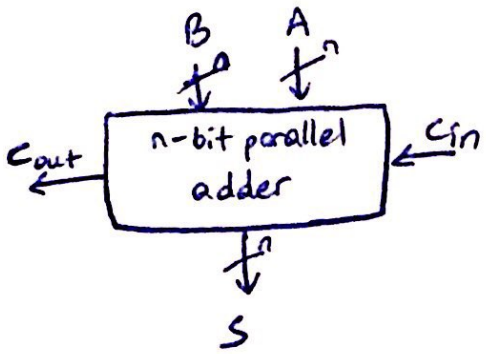
* **half adder**

* **full adder**



* n-bit binary parallel adder

→ it adds two n-bit binary number



Ex:

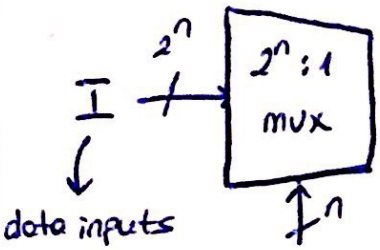
1. num = 0110 → 6 } 18
2. num = 1100 → 12
result = 0010 → 2 > 18
carry = 1 → 16

! a 4-bit substraction circuit = S = A − B = A + 2's com(B) = A + ($\bar{B}$ + 1)

* multiplexer (mux) (data selector)

→ 2^n

general $Z = \sum_{j=0}^{k-1} (m_j I_j)$ → j. minterm

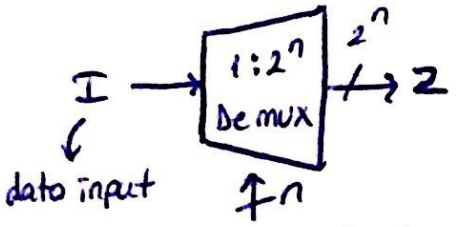


→ Z → output = $s'I_0 + sI_1$ (2:1 mux)

S → control inputs → select a data to connect to the output

→ logic circuits designs with multiplexers → 5.13, 5.14, 5.15

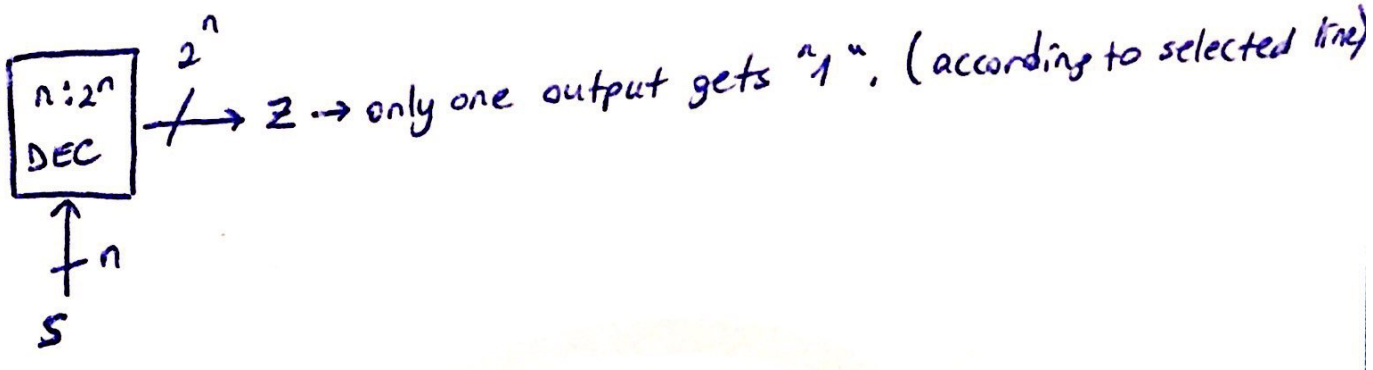
* demultiplexer



Z → the value of the not-selected output lines is "0".

S → select a data output to connect to single input

* decoder (minterm generator)



! decoders may have "enable" (EN) input.
if EN input = 1 ⇒ normal
0 =0 ⇒ all outputs = "0"

Programmable Logic Device (PLD)

* some PLD's also include memory units (flip-flops)
* there are different kinds of PLD's:
  - programmable logic array (PLA) } - early versions
  - programmable array logic (PAL) } - bipolar transistors
    - one-time programmable (OTP)
  - generic array logic (GAL) } - todays devices
  - complex PLD (CPLD) } - Cmos transistors, Memory
  - field-programmable gate array (FPGA) } - can be reprogrammed
* there are various Hardware Description Languages (HDL) and programming devices to program PLDs.
  - PALASM
  - ABEL
  - Verilog
  - VHDL (very high speed integrated circuits HDL)

Programmable Logic Array - PLA

* AND (product) units → in the input layer
  OR (sum) units → in the output layer

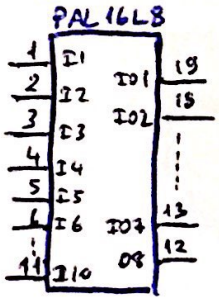
! Both AND, OR arrays are flexible programmable.

* they called as "n x m PLA with p products"
  - n: input
  - m: output
  - p: AND gates

* simple representation in 5.23 (X's)

Programmable Array Logic - PAL

* inputs of AND gates can be flexible programmed as in PLA's. But inputs of OR gates are fixed. However, PAL's can be easily programmed.



Generic Array Logic (GAL)

* its logical properties are similar to PAL.
* it is made of CMOS transistors → can be erased and programmed many.

Complex PLD (CPLD)

* IC that contains several PLDs (macro cell)
* internal structures of macro cells and connections between them can be programmed.

Field-Programmable Gate Array (FPGA)

* they contain many logical blocks and interconnections between these blocks.
* can be used implement complex digital circuits (special purpose microprocessors)
* more flexible and can implement more circuits than CPLDs but their delay cost is higher.