# Digitális technika

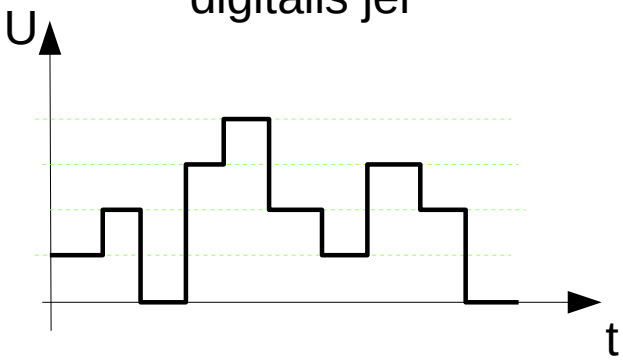
I.
Alapok,
bináris számrendszer,
kódok

# 1.1. Alapfogalmak

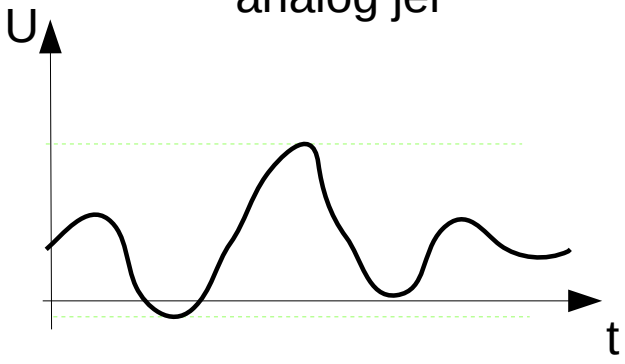
## Digitális jel

csak diszkrét (meghatározott) értékeket vehet fel,
szemben egy analóg jellel, amely bármilyen értékeket fel vehet
(egy minimum és egy maximum között)

digitális jel




analóg jel




- Sok diszkrét érték is lehetne, de nehéz megvalósítani !!
- Két értékű rendszereket használunk → 0, 1

- Két értékű rendszerek leírására, tervezésére jó a logikai algebra (Boole-algebra),
  a számolás esetén pedig a 2-es számrendszer használható jól

# 1.1. Alapfogalmak

## Logikai értékek

A digitális áramkörök azok logikai hálózatok →
leírásukra, tervezésükre a Boole-algebra használatos.

Két érték van:
hamis –> 0
igaz –> 1
(így használjuk, bár fordítva is hozzárendelhetnénk a számokat)

A gyakorlatban a két értékhez két különböző feszültség tartomány tartozik, ezeket L és H betűkkel jelöljük
( Low illetve High, alacsony és magas feszültség tartomány)

1 → H

0 → L

# 1.1. Alapfogalmak

## Logikai hálózat

Több bemenettel, és akár több kimenettel rendelkező
logikai áramkör (digitális áramkör)


Két típusa van:
kombinációs hálózat és sorrendi hálózat (szekvenciális)

A kimenetek csak a
bemenetektől függenek.
időtől független !

A kimenetek nemcsak a
bemenetektől függenek,
hanem a hálózat állapotától is.
időtől függő !

Logikai hálózatok leírása → logikai függvényekkel →
'n' számú kimenet esetén 'n' db függvény (minden kimenethez egy függvény !!)

# 1.2. Számrendszerek

## 10-es számrendszer (decimális)

10 db számjegy → '0' '1' '2' '3' … '8' '9'

helyi értékek → … 10000 1000 100 10 1

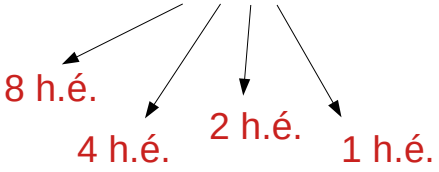
tehát pl. a 7439 azt jelenti hogy van 7db 1000-esünk, meg 4 db 100-asunk, meg 3db 10-esünk és 9db 1-esünk

## 2-es számrendszer (bináris)

csak 2 számjegy !! → '0' és '1'

helyi értékek → … 256 128 64 32 16 8 4 2 1

pl. $1011_2 = 1*8+0*4+1*2+1*1 = 8 + 2 + 1 = 11_{10}$




pl2. $1101101_2 = 1*64+1*32+0*16+1*8+1*4+0*2+1*1 = 64+32+8+4+1 = 109_{10}$

$101011010_2$= …... ? → 256 128 64 32 16 8 4 2 1 melyikből mennyi van?

$1*256 + 0*128 + 1*64 + 0*32 + 1*16 + 1*8 + 0*4 + 1*2 + 0*1 =$

$= 256+64+16+8+2 = 346_{10}$

# 1.2. Számrendszerek

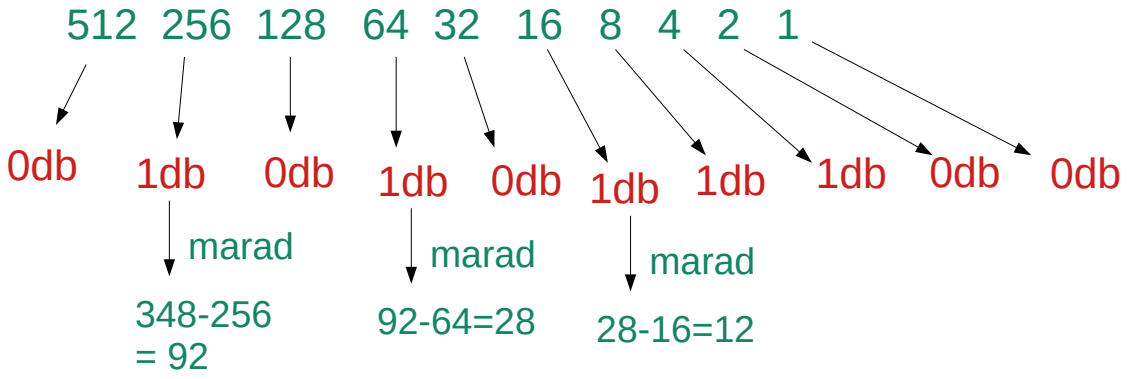
## Bit, byte

1db bináris számjegy (helyiérték) → bit (binary digit)
8 bit = byte 1024 bit = 1kilobit

## 10-es → 2-es átalakítás

Átváltás 10 számrendszerből 2-es számrendszerbe,
ha nem túl nagy szám (<1000) akkor egyszerűen összerakjuk, hogy
melyik helyi értékből kell 1db és melyikből 0db

pl. $348_{10}$ = …… $_2$ ? → 512 256 128 64 32 16 8 4 2 1 melyikből mennyi kell?



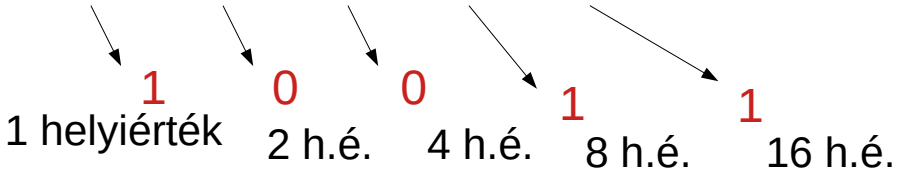

Megoldás: $0101011100_2$

# 1.2. Számrendszerek

## 10-es → 2-es átalakítás másképpen

Algoritmussal:
sorozatos osztás 2-vel, és a maradékok adják a számjegyeket
(az először az utolsó számjegyet kapjuk meg, majd az előtte lévőt, …
és legvégül a legelsőt)

pl. $25_{10}$ = ….. $_2$ ?  25 → 12 → 6 → 3 → 1 → 0




eredmény: $11001_2$

# 1.2. Számrendszerek

## 16-os számrendszer (hexadecimális)

16 számjegy → '0' '1' '2' ….. '8' '9' 'A' 'B' 'C' 'D' 'E' 'F'

helyi értékek → … . 256 16 1

10 11 12 13 14 15

pl. $3A4_{16}$= 3*256+10*16+4*1=932

$1EC_{16}$ = …... ? → 256 16 1 melyikből mennyi van?

hexa → bináris konverzió:
számjegyenként 4 bitre !

pl. $2E_{16}$ → $00101110_2$

0010 1110

bináris → hexa konverzió:
4 bites csoportokra osztás jobbról,
csoportonként hexa számjegyekké alakítás!

pl. $1111010111_2$ → $3D7_{16}$

3 13 7

D

Miért használjuk a 16-os számrendszert ?
Mert nagy számokat egyszerűbb leírni így (kevesebb számjegy)
mint kettes számrendszerben

# 1.3. Gyakorló feladatok

Végezd el a számrendszerek közötti átváltásokat !

1. $11001010_2$ = ……..…… $_{10}$
2. $106_{10}$ = ……..…. $_2$
3. $101011001_2$ = ……...$_{10}$
4. $10110101011_2$ = ……..$_{16}$
5. $189_{10}$ = ……..$_{16}$
6. $5F_{16}$ = ……..$_2$

# 1.3. Gyakorló feladatok

## Megoldások

1. $11001010_2 = 1*128 + 1*64 + 1*8 + 1*2 = 202$
2. $106_{10} = 64 + 32 + 8 + 2 = 1101010_2$
3. $101011001_2 = 256 + 64 + 16 + 8 + 1 = 345$
4. $10110101011_2 = 5AB_{16}$
5. $189_{10} = 11*16 + 13*1 = BD_{16}$
6. $5F_{16} = 01011111_2$

# 1.4. Bináris kódok

## Kód, kódolás

- Kód: egyezményes jelrendszer
- Kódolás: információ leírása valamilyen kódban
- A kód kódszavakból áll
- lehet:
  - alfanumerikus (betűk, számok)
  - numerikus (csak számok)
- Bináris kódok: csak '0' és '1' szám

### Bináris kódok

Csak ’0’ és ’1’
Sokféle lehet !

- 'sima' bináris szám
- egyes komplemens kód
- kettes komplemens kód
- BCD kódok
- egylépéses kódok
- 1 az N-ből kódok
- hiba ellenőrző, hiba javító kódok

# 1.4. Bináris kódok

## 1 az N-ből kód

- minden kódszóban csak 1db bit 1-es értékű, a többi 0
  (vagy fordítva is lehet: csak 1db 0 minden kódszóban, a többi 1-es
- hátrány: pazarló (sok bit kell)

1 az 5-ből kód

0 → 00001
1 → 00010
2 → 00100
3 → 01000
4 → 10000

1 a 8-ból kód

0 → 00000001
1 → 00000010
2 → 00000100
3 → 00001000
4 → 00010000
5 → 00100000
6 → 01000000
7 → 10000000

1 a 10-ből kód

0 → 0000000001
1 → 0000000010
2 → 0000000100
3 → 0000001000
4 → 0000010000
5 → 0000100000
6 → 0001000000
7 → 0010000000
8 → 0100000000
9 → 1000000000

# 1.5. BCD kódok

## BCD kódok

- binárisan kódolt decimális szám
- a tízes számrendszerű számot számjegyenként kódoljuk 4 bites csoportokban →
- 4 biten csak 10 számot (0-9) kódolunk le (6 kódszót nem használunk)
- több típusa van

### Normál BCD kód

N-BCD

- normál bináris helyi értékek !
- 8-4-2-1 súlyozás
  - 0 → 0000
  - 1 → 0001
  - 2 → 0010
  - ....

| | 4 bit | | |
|---|---|---|---|
| 0 | 0000 | N-BCD kódszó | 0 |
| 1 | 0001 | | 1 |
| 2 | 0010 | | 2 |
| 3 | 0011 | | 3 |
| 4 | 0100 | | 4 |
| 5 | 0101 | | 5 |
| 6 | 0110 | | 6 |
| 7 | 0111 | | 7 |
| 8 | 1000 | | 8 |
| 9 | 1001 | | 9 |
| 10 | 1010 | érvénytelen N-BCD kódszó | |
| 11 | 1011 | | |
| 12 | 1100 | | |
| 13 | 1101 | | |
| 14 | 1110 | | |
| 15 | 1111 | | |

pl. $47_{10}$ = ……..$_{NBCD}$ ?

0100 0111 → = $01000111_{NBCD}$

pl. 2. $11001010011_{NBCD}$ = ……..$_{10}$ ?

$11001010011_{NBCD}$ = $653_{10}$

6 5 3

# 1.5. BCD kódok

## Három többletes kód

Stibitz, Excess3

8-4-2-1 súlyozású BCD, De !
minden decimális számhoz
3-al nagyobb bináris értéket rendel

0 → 0011
1 → 0100
2 → 0101
3 → 0110
...

pl. $47_{10}$ = ……..Stibitz ?

4+3 7+3

0111 1010 → = $01111010_{Stibitz}$

pl. 2. $11001010011_{Stibitz}$ = ……$_{10}$ ?

$11001010011_{Stibitz}$ = $320_{10}$

6-3 5-3 3-3

4 bit

| | | | |
|---|---|---|---|
| 0 | 0000 | érvénytelen Excess3 kódszó | |
| 1 | 0001 | | |
| 2 | 0010 | | |
| 3 | 0011 | Excess3 kódszó | 0 |
| 4 | 0100 | | 1 |
| 5 | 0101 | | 2 |
| 6 | 0110 | | 3 |
| 7 | 0111 | | 4 |
| 8 | 1000 | | 5 |
| 9 | 1001 | | 6 |
| 10 | 1010 | | 7 |
| 11 | 1011 | | 8 |
| 12 | 1100 | | 9 |
| 13 | 1101 | érvénytelen Excess3 kódszó | |
| 14 | 1110 | | |
| 15 | 1111 | | |

# 1.5. BCD kódok

## Aiken kód

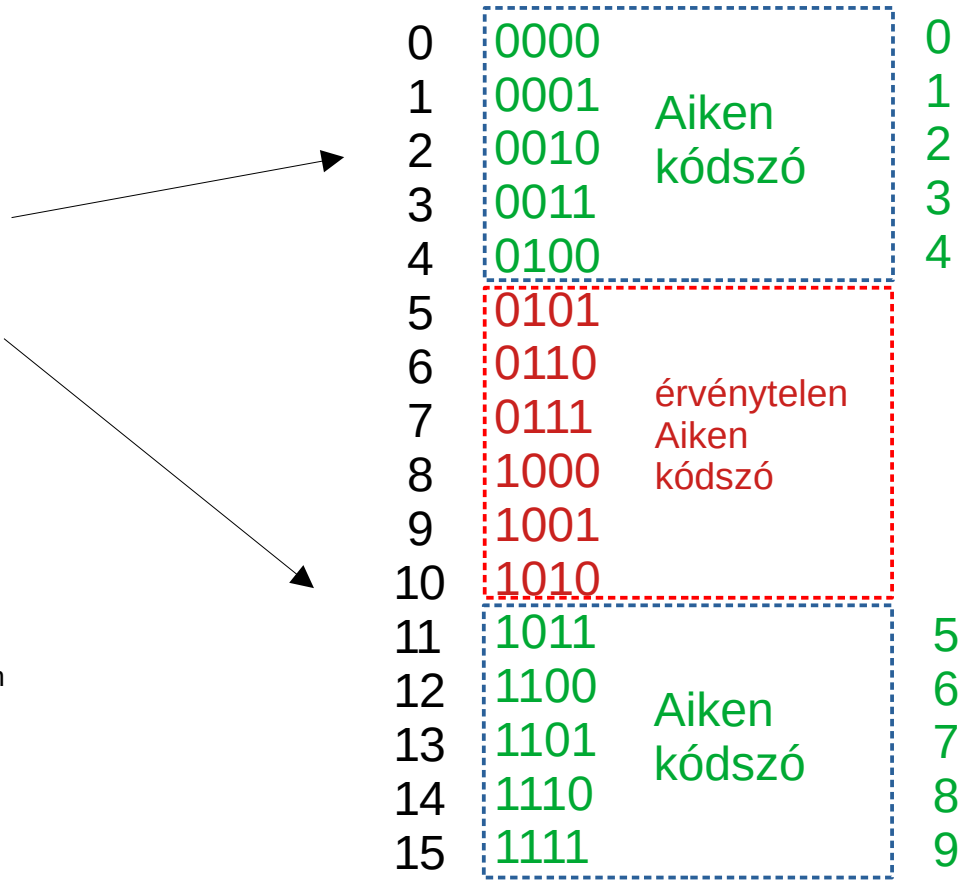
2-4-2-1 súlyozású BCD
(mintha a legelső bit is 2-es helyiértékű volna 8-as helyett)

- 0-4 számok → 1. bit 0
- 5-9 számok → 1. bit 1

| | |
|---|---|
| 0 → 0000 | 5 → 1011 |
| 1 → 0001 | 6 → 1100 |
| 2 → 0010 | 7 → 1101 |
| 3 → 0011 | 8 → 1110 |
| 4 → 0100 | 9 → 1111 |

pl. $47_{10}$ = ……Aiken ?

0100 1101 → = $01001101_{Aiken}$



# 1.6. Egylépéses kódok

## Egylépéses kódok

- ciklikusan permutált kódok
- az egymást követő kódszavak csak 1 bitben térnek el egymástól

**Gray kód (2 bites)**

0 → 00
1 → 01
tükrözés
2 → 11
3 → 10

**Gray kód (3 bites)**

0 → 000
1 → 001
2 → 011
3 → 010
tükrözés
4 → 110
5 → 111
6 → 101
7 → 100

**Gray kód (4 bites)**

0 → 0000
1 → 0001
2 → 0011
3 → 0010
4 → 0110
5 → 0111
6 → 0101
7 → 0100
tükrözés
8 → 1100
9 → 1101
10 → 1111
11 → 1110
12 → 1010
13 → 1011
14 → 1001
15 → 1000

# 1.6. Egylépéses kódok

## Normál Gray kód

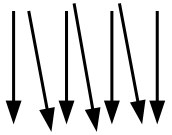
N-Gray

- BCD kód is egyben !

| | |
|---|---|
| 0 → 0000 | 5 → 0111 |
| 1 → 0001 | 6 → 0101 |
| 2 → 0011 | 7 → 0100 |
| 3 → 0010 | 8 → 1100 |
| 4 → 0110 | 9 → 1101 |

N-BCD → N-Gray

pl. 0 1 1 0



XOR

0 1 0 1

# 1.6. Egylépéses kódok

## Johnson kódok

- a 0 értékű kód először jobbról feltöltődik 1-ekkel, majd a csupa 1 bites kódszó után 0 értékekkel

- több variációja létezik, a bitszámban különböznek

4 bites Johnson kód

0 → 0000
1 → 0001
2 → 0011
3 → 0111
4 → 1111
5 → 1110
6 → 1100
7 → 1000

5 bites Johnson kód

0 → 00000
1 → 00001
2 → 00011
3 → 00111
4 → 01111
5 → 11111
6 → 11110
7 → 11100
8 → 11000
9 → 10000

# 1.7. Gyakorló feladatok

1. Végezd el az átváltásokat (10-es számrendszer és BCD kód között) !

   a. $389_{10}$ = ……..…….....NBCD

   b. $587_{10}$ = ……..…….....Stibitz

   c. $100010010111_{NBCD}$ = ……..10

   d. $100001011001_{Stibitz}$ = ……..10

2. Milyen kódban lehetnek az alábbi számok ? (normál BCD vagy Stibitz kód ?)
   Add meg milyen számot kódolnak (10-es számrendszer)!

   a. $10110010110_{?}$ = ……..10

   b. $11010111010_{?}$ = ……..10

   c. $10101111100_{?}$ = ……..10

   d. $11010000111_{?}$ = ……..10

# 1.8. Negatív számok

## Negatív számok ábrázolása

az előjel ábrázolására/tárolására → plusz egy előjel bit (a legelső)
előjel bit: 0 → pozitív szám 1 → negatív szám
de ez még nem elég, a műveletvégzés így még okozhat hibákat !
pl. +2 és -2 összeadása → 0010+1010=1100 → -4 !!!

előjel

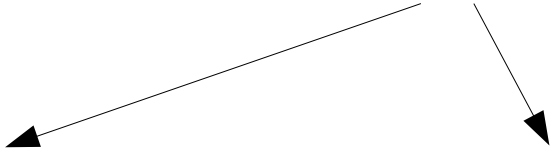
A negatív számokat 2-es komplemens kódban ábrázoljuk
2-es komplemens kód: a megfelelő pozitív szám bitenkénti negáltja, majd utána a számhoz hozzáadunk még 1-et

pl. 4 bites számok (ebből az első előjel)

| | |
|---|---|
| 0 → 0000 | |
| 1 → 0001 | -1 → 1111 ( 0001 → 1110 → 1111) |
| 2 → 0010 | -2 → 1110 ( 0010 → 1101 → 1110) |
| 3 → 0011 | -3 → 1101 ( 0011 → 1100 → 1101) |
| 4 → 0100 | -4 → 1100 ( 0100 → 1011 → 1100) |
| 5 → 0101 | -5 → 1011 ( 0101 → 1010 → 1011) |
| 6 → 0110 | -6 → 1010 ( 0110 → 1001 → 1010) |
| 7 → 0111 | -7 → 1001 ( 0111 → 1000 → 1001) |
| | -8 → 1000 ( 1000 → 0111 → 1000) |

# 1.8. Negatív számok

Másik oldalról a negatív számok használata felére csökkenti a használható számtartományt !
pl. ha 8 bites számokkal dolgozunk akkor két eset lehetséges



Csak pozitív számokat használunk!
tehát:
0 → 00000000
1 → 00000001
2 → 00000010
3 → 00000011
4 → 00000100
……
253 → 11111101
254 → 11111110
255 → 11111111

Pozitív és negatív számokat használunk!
Tehát:
0 → 00000000
1 → 00000001
2 → 00000010
……
126 → 01111110
127 → 01111111
-1 → 11111111
-2 → 11111110
-3 → 11111101
……
-127 → 10000001
-128 → 10000000