



**BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM**  
**VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR**  
**MÉRÉSTECHNIKA ÉS INFORMÁCIÓS RENDSZEREK TANSZÉK**

# **Digitális technika**

## **VIMIAA02**

### **2. EA**

**Fehér Béla, Benesóczky Zoltán**  
**BME MIT**

# Digitális technika

## Adatábrázolás *kódokkal*

- A digitális információfeldolgozó gépek bemenetükre érkező információt **bináris kódként** kapják.
  - **Aritmetikai kódok** (számok)
    - Akkor alkalmazzuk, ha a kódolt információnak nagysága van. Pl. Fizikai mennyiség (feszültség, hőmérséklet, stb.) kódolása.
    - Ilyenkor összehasonlíthatók és aritmetikai műveletek végezhetők közöttük.
    - Analóg jelből analóg/digitális konverzióval (A/D konverter) állítják elő.
  - **Egyéb jelek, kódok** (nem, vagy korlátozottan végezhető aritmetika művelet)
    - Nyomógomb állapota, határérték túllépése (1 biten kódolhatók).
    - Halmaz elemeinek megkülönböztetése. (Pl. alma:00, körte:01, sárgabarack:10, őszibarack:11)
    - Közlekedési lámpa állapotai (**PSZ**: 100,110,001,010)
    - Karakterek kódolása (Pl. ASCII kód, 8 bites, sorbarendezésre alkalmas)
    - Speciális egyéb kódok (nem részletezzük)

# Számábrázolási módszerek

- A megszokott **súlyozott helyiértékes (pozicionális) számrendszerekben**, megegyezés szerint, a balról jobbra egymás után a leírt számok  $i$ -edik számjegyének ( $d_i$ ) értéke  $r$  alapú számrendszerben:  
 $d_i * r^i \in \{0, 1 \dots r-1\}$  (Pl. 935-ben a 2. számjegy értéke  $9 * 10^2$ )
- A  $d_i$  számjegyek a **0-tól a  $r-1$  értékűek** lehetnek:  
 $d_i \in \{0, 1 \dots r-1\}$  (Pl. 10-es számrendszerben  $d_i \in \{0, 1 \dots 9\}$ )
- $r^i$ -t a  $d_i$  számjegy **súlyának** nevezik.
- A teljes szám értéke a  $d_i * r^i$ -k összege: (Pl.  $935_{10} = 9 * 10^2 + 3 * 10^1 + 5 * 10^0$ )  
$$D = d_{N-1} * r^{N-1} + d_{N-2} * r^{N-2} + \dots + d_2 * r^2 + d_1 * r^1 + d_0 * r^0$$

Rövid matematikai alakban: 
$$D = \sum_{i=0}^{N-1} d_i * r^i$$

A fenti képletet tetszőleges számrendszerű szám decimális konverziójára szoktuk használni.



# Számábrázolási módszerek

- **Digitális technikában fontos számrendszerek**
- **Tízes/Decimális/Dekadikus**  $r = 10$   
 $d_i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,$
- **Kettes/Bináris**  $r = 2$   
 $d_i = 0, 1,$  (a nevük bit, **binary digit == bit**)
- **Tizenhatos/Hexadecimális**  $r = 16$ 
  - A számjegyek 9-ig a megszokottak, utána az ABC betűit használják az A-tól kezdve (10:A, 11:B,...15:F).  
 $d_i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$   
 $d_i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f$
  - A számjegyek fenti szimbólumait a digitális gépek bináris bitsorozatokkal reprezentálják.

# Konverzió számrendszerek között

- **Hexadecimális → Decimális konverzió**

- A már ismert képletet használjuk  $r=16$  esetére:

$$D = \sum_{i=0}^{N-1} d_i * 16^i$$

$$\begin{aligned} 1A0_{16} &= 1*16^2 + 10*16^1 + 0*16^0 = \\ &= 1*256 + 10*16 + 0*1 = 416_{10} \end{aligned}$$

Tehát  **$1A0_{16} = 416$**

(A szám számrendszerét sokszor a szám utáni alsó index-szel jelöljük, de 10-es számrendszer esetén nem szokás kitenni.)

**Verilogban:**

**$\langle - \rangle \langle bitek\_száma \rangle' \langle s \rangle \langle számrendszer \rangle \langle numerikus\_konstans \rangle$**

**$\langle - \rangle$ :** Negatív előjel (vagy nincs)  **$\langle s \rangle$ :** 2-es komplementes kódú

A **16-os** számrendszer jele: **h**, a **bináris** számrendszeré: **b**

pl. A 12 biten ábrázolt  $1A0_{16}$  Verilogban  **$12'h1A0$**

# Konverzió számrendszerek között

- A Bináris → Decimális konverzió

- A már ismert képletet használjuk  $r=2$  esetére:

$$D = \sum_{i=0}^{N-1} d_i * 2^i$$

$$\begin{aligned} 1110_2 &= 1*2^3 + 1*2^2 + 1*2^1 + 0*2^0 = \\ &= 8 + 4 + 2 + 0 = 14_{10} \end{aligned}$$

Tehát  $1110_2 = 14$

Verilogban a fenti szám 2-es számrendszerben:

4 biten `4'b1110` 8 biten `8'b00001110` vagy

`8'b0000_1110` (az aláhúzás jellel tagolhatjuk), vagy

`8'b1110` (a vezető 0-ák elhagyhatók)



# Konverzió számrendszerek között

- **Hexadecimális  $\rightarrow$  Bináris konverzió**

Mivel  $16 = 2^4$ , 1 db hexadecimális számjegy 4 bináris digit (bit)

Most felírjuk a hexidecimális számjegyek értékét 4 bites bináris számokként:

$0_{16} = 0000_2$   $1_{16} = 0001_2$   $2_{16} = 0010_2$   $3_{16} = 0011_2$   $4_{16} = 0100_2$   $5_{16} = 0101_2$   $6_{16} = 0110_2$   
 $7_{16} = 0111_2$   $8_{16} = 1000_2$   $9_{16} = 1001_2$   $A_{16} (10) = 1010_2$ ,  $B_{16} (11) = 1011_2$ ,  $C_{16} (12) = 1100_2$   
 $D_{16} (13) = 1101_2$   $E_{16} (14) = 1110_2$   $F_{16} (15) = 1111_2$

A konverzió során egyszerűen a hexadecimális számjegyeket a számjeggyel egyező értékű 4 bites bináris számra kell cserélni:

$$\text{Pl. } 2A_{16} = \overset{2_{16}}{00} \overset{A_{16}}{10} 1010_2$$

A vezető (bal oldali) 0-akat elhagyhatjuk: 101010

# Konverzió számrendszerek között

- **Bináris → Hexadecimális konverzió**

Ha a bináris szám bitszáma nem 4 többszöröse, akkor először balról kiegészítjük megfelelő számú 0-val.

Ezután **4 bites csoportokat képzünk**, majd a 4 bites bináris számokat a **hexadecimális megfelelőjükkel helyettesítjük**.

Pl.  $101101101_2$

Kiegészítés 0-kkal a baloldalon és 4-es csoportok képzése:

0001\_0110\_1101

Hexa számjegyek hozzárendelése:      **1**      **6**      **D**

Tehát  $1\_0110\_1101_2 = 16D_{16}$



# Konverzió számrendszerek között

## A Decimális $\rightarrow$ Bináris konverzió 1.

- Az eddigieknél bonyolultabb az algoritmus.
- A konvertálandó decimális számot **addig kell osztogatni 2-vel, amíg nullává nem válik a hányados.**
- Az egyes **osztások maradékai adják a bináris szám bitjeit, a legkisebb (0.) bittől (Least Significant Bit, **LSB**) kezdődően.**
  - Pl.  $13_{10}$  átalakítása:

osztandó	osztó	hányados	maradék
13	2	6	<b>1 (LSB)</b>
6	2	3	<b>0</b>
3	2	1	<b>1</b>
1	2	0	<b>1</b>

- Az eredmény:  $13_{10} = 1101_2$

# Digitális technika

- Néhány fontosabb bináris érték

$2^7$	$2^8$	$2^{10}$	$2^{16}$	$2^{20}$	$2^{30}$	$2^{32}$
128	256	1024	65536	1048576	1073741824	4294967296
~száz		~ezer		~millió	~milliárd	

- Apró kellemetlenség,  $1000 \neq 1024$  ( $10^3 \neq 2^{10}$ )
- A korábban elterjedt k, M, G, T nagyságrendi jelölések nem teljesen precízek.

SI (decimális)				IEC (bináris)			
jel	név	érték		jel	név	érték	
k	kilo	$10^3$	$1000^1$	Ki	kibi	$2^{10}$	$1024^1$
M	mega	$10^6$	$1000^2$	Mi	mebi	$2^{20}$	$1024^2$
G	giga	$10^9$	$1000^3$	Gi	gibi	$2^{30}$	$1024^3$
T	tera	$10^{12}$	$1000^4$	Ti	tebi	$2^{40}$	$1024^4$

- Az új szabványos jelölés lassan terjed, mi is nehezen tanuljuk, de egy informatikusnak illik tudni róla.

# Műveletek bináris számokkal

- **Bináris pozitív egész számok**

- A pozitív egész bináris számokat előjel nélküli bináris számoknak is szokás nevezni.
- **N** biten, a **pozitív egészek értéktartománya 0-tól  $2^N-1$  terjed.** (Pl. 8 biten 0-tól 255, 16 biten 0-65535)

- **Aritmetikai műveletek**

- **Bináris összeadás**

Szabályok 1 bites operandusokra:

**0 + 0 = 0, 1 + 0 = 0 + 1 = 1, 1 + 1 = 10**, ahol az **1** az átvitel a következő helyiértékre. (Úgy is mondhatjuk, hogy **1 + 1 = 0 maradt az 1**)

- Példa:  $6 + 3 = 9$ , 4 biten
- Átvitel a 2. és a 3. pozíción
- Túl nagy számok esetén az eredmény nem fér el a rendelkezésre álló számú biten. (Pl.  $9 + 8 = 17$ )

		1	1				
		0	1	1	0		6
+		0	0	1	1	+	3
		1	0	0	1		9



# Műveletek bináris számokkal

## Bináris szorzás

- **Szabályok egy bites operandusokra.** (Ez a Boole algebrai szorzás művelet, vagyis ÉS, AND):  
 $0 * 0 = 0, \quad 1 * 0 = 0, \quad 0 * 1 = 0, \quad 1 * 1 = 1$
- Bináris számoknál a szorzó és szorzandó számjegyei 2 hatványok. Ezért fontos a 2-vel és 2 hatvánnyal szorzás szabálya.
- A 2-vel szorzás után az egyes számjegyek (digitek,  $d_{Ni}$ ) 1-el nagyobb hatvánnyal szorzódnak mint a szorzás előtt és a legkisebb számjegy 0 lesz:
- $$N * 2 = (d_{Nn} * 2^n + \dots + d_{N1} * 2^1 + d_{N0} * 2^0) * 2 =$$
$$= d_{Nn} * 2^{n+1} + \dots + d_{N1} * 2^2 + d_{N0} * 2^1 + 0 * 2^0$$

Pl.  $0011 * 10 = 0110$  ( $3 * 2 = 6$ )
- 2-es számrendszerben egy N szám 2-vel szorzásnál 1-el balra toljuk (shifteljük) a számjegyeket és a legkisebb számjegy 0 lesz.
- 2-es számrendszerben egy N szám  $2^k$ -al szorzása k-szori balra shiftelésnek felel meg és jobbról k darab 0 lép be.

Pl.  $0101 * 1000 = 0101000$

BME-MIT   $5 * 8 = 40$

# Műveletek bináris számokkal

N szorozva M, ahol  $M = d_{Mn} * 2^n + \dots + d_{M1} * 2^1 + d_{M0} * 2^0$   $d_i \in \{0,1\}$  számmal.

$$N * M = N * d_{Mn} * 2^n + \dots + N * d_{M1} * 2^1 + N * d_{M0} * 2^0$$

A szorzás i-edik lépésében **ha a szorzó i-edik számjegye  $d_{Mi} = 1$ , akkor a szorzandó (N) i-szer balra shifteltjét ( $2^i$ -szeresét) hozzáadjuk az eddigi szorzatösszeghez.** Ha  $d_i = 0$ , akkor 0-át adunk hozzá (semmit nem csinálunk).

**Az eredmény 2 db k-bites szám esetén maximum  $2*k$  bites lehet.**

Példa:  $14_{10} * 10_{10} = 140_{10}$

4 bites bináris számokkal, *a szorzó legkisebb bitjével (LSB) kezdve:*

<b>1110 * 1010</b>	részösszegek 10-es számrendszerben mutatva
+ 0000	0. részszorzat: <b>0</b> *1*14 = 0
+ 11100	1. részszorzat : <b>1</b> *2*14 = 28
+ 000000	2. részszorzat : <b>0</b> *4*14 = 0
+1110000	3. részszorzat : <b>1</b> *8*14 = 112
-----	
<b>=10001100<sub>2</sub></b>	

$= 128_{10} + 8_{10} + 4_{10} = 140$

# Műveletek bináris számokkal

Példa:

4 bites bináris számokkal, a szorzó legnagyobb helyiértékű bitjével kezdve (MSB) kezdve:

<b>1110 * 1010</b>	részösszegek 10-es számrendszerben mutatva
+ 1110000	0. részorzat : $1 * 8 * 14 = 112$
+ 000000	1. részorzat : $0 * 4 * 14 = 0$
+ 11100	2. részorzat : $1 * 2 * 14 = 28$
+ 0000	3. részorzat : $0 * 1 * 14 = 0$
-----	
<b>= 10001100<sub>2</sub> = 128<sub>10</sub> + 8<sub>10</sub> + 4<sub>10</sub> = 140</b>	



# Előjeles számábrázolás

- Eddig csak pozitív egész számokkal foglalkoztunk, azonban negatív számokra is szükségünk van.
- **Előjeles számok**
  - Normál jelölésben van előjel (+ és - jel, előbbit nem szoktuk jelölni)
  - Bináris számoknál csak „0” és „1” szimbólum van.
  - A bináris előjeles számok ábrázolására több formátum is létezik (pl. 1-es komplement, **2-es komplement**, offset stb.).
  - Mi ezekből csak a **kettes komplement** számábrázolással foglalkozunk részletesen. Ezt **2's C**-vel is szokás jelölni.

# Előjeles számábrázolás

- **Egyes komplement (1's C)**
  - Képzési szabálya (egy szám -1-szerese): Minden bináris számjegyet invertálunk ( $0 \rightarrow 1, 1 \rightarrow 0$ )
- **Kettes komplement (2's C)**
  - Képzési szabálya (egy szám -1-szerese):  
**Minden bitet invertálunk (1's C) majd az így kapott számhoz hozzáadunk 1-et és csak az eredeti számú bitet őrizzük meg.**
- Pl. +3: **0011** invertálás: 1100  
1 hozzáadása: + 0001

$$\begin{array}{r} \text{-----} \\ 1101_2 = -3_{10} \end{array}$$

Mégegyszer alkalmazva visszkapjuk az eredeti számot.

$$1101 \text{ inv} \rightarrow 0010 + 1 \rightarrow 0011$$

Bináris	2's C
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

# Előjeles számábrázolás

- **Kettes komplement számábrázolás**

- A pozicionális számábrázolás definíciója alapján:

$$D = -b_{N-1} * 2^{N-1} + \sum_{i=0}^{N-2} b_i * 2^i$$

- $b_{N-1}$  a legnagyobb helyiértékű bit (MSB),  $b_i$  pedig a többi bit.
- Az MSB negatív értékű, ha nem nulla. Másképp fogalmazva  **$b_{N-1}$  az előjel bit**. A szám pozitív vagy 0, ha  $b_{N-1} = 0$ , a szám negatív, ha  $b_{N-1} = 1$
- N biten a számábrázolási tartomány:  $-2^{N-1} \dots + 2^{N-1} - 1$

Pl. 4 biten: -8 ...+7



# Előjeles számábrázolás

- **A 2's C számábrázolás tulajdonságai**

A legnegatívabb számban csak az előjel bit 1 értékű.

Pl. 4 biten a legnegatívabb szám:  $1000_2 = -8$

- **A legnegatívabb szám 2-es komplementese önmaga**, tehát nem önmaga -1-szeresét adja! (Az nem fér be a számtartományba.)

Pl. 4 biten  $-8_{10} : 1000 \rightarrow 0111 + 1 = 1000$  (számtartomány:  $-8 \dots +7$ )

- **A 2's C előjeles számokkal az összeadás szabályai megegyeznek a normál pozitív számokra vonatkozókkal** (az eredmény is 2's C)

Pl:      0101 (+5)

     +1001 (-7)

-----

     1110 (-2)

# Előjeles számábrázolás

- **Kettes komplement (2's C) méretkonverzió**
  - **Előjel kiterjesztés:** A bitjek számának a növelése
  - **Pozitív számokra** egyértelmű, **bal oldalon kiegészítés 0-kal**, a számérték természetesen nem változik.  
Pl: +5 érték 4 biten **0101** és 12 biten **0000\_0000\_0101**
  - **Negatív szám esetén** a **bal oldalon kiegészítés 1-ekkel**.  
Pl: -5 érték 4 biten **1011** és 12 biten **1111\_1111\_1011**  
Mert a 2's C szabályai szerint ennek bitjeit invertálva + 1, azaz  $0000\_0000\_0100 + 1 = 0000\_0000\_0101$
  - Tehát, ha kevesebb bitről **előjel kiterjesztéssel** méretet növelünk több bitre, **a szám értéke nem változik**
  - Jelentősége: pl. konverzió különböző méretű adatformátumok között (pl. 8 bites byte-ból → 32 bites szó)

# Valós számok

## Fixpontos számábrázolás

- Az eddigi pozícionális számrendszer a törtrészre is kiegészíthető, **negatív kitevőkkel**:  $r^{-1}$ ,  $r^{-2}$ , .....  $r^{-n}$ , tört helyiértékek,  $r^0$ -tól jobbra ( $\frac{1}{2}, \frac{1}{4}, \dots$ )
- Bináris előjeles (2-es komplement) számrendszer 8 biten valós számokra, 3 db tört számjegy esetére:**

$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	$2^{-1}$	$2^{-2}$	$2^{-3}$
-16	8	4	2	1	0,5	0,25	0,125

Implicit (nem létező) „kettedes” pont a ↑ megfelelő helyen

- Példák a fenti formátumú számok decimális konverziójára:
  - 00110 101** =  $6 + 0.5 + 0.125 = \mathbf{6.625}$
  - 11111 111** negatív, tehát 2-es kompl. képzés kell az abszolút értékhez:  
 $00000\ 001 = 0.125$  tehát **11111 111** = **-0.125**
- Tetszőleges pontosság, bitszám növelésével elérhető, de**

Probléma a végtelen tizedstörtek:  $0.1_{10} = 0.0001100110011001100\dots_2$



# Fixpontos számábrázolás

- A teljes értéktartományt (FSR, Full Scale Range) a legnagyobb helyiértékű bit (MSB) értéke határozza meg, a példában az előjeles számokra  $\sim \pm 2^4 = \sim \pm 16$

$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	$2^{-1}$	$2^{-2}$	$2^{-3}$
-16	8	4	2	1	0,5	0,25	0,125

- Két érték közötti min. eltérést (felbontás, pontosság) a legkisebb helyiértékű bit (LSB) értéke határozza meg, a példában  $\sim \pm 2^{-3} = \sim \pm 0,125$ 
  - Nagy értéktartományhoz  $\rightarrow$  sok egész bit kell
  - Nagy pontossághoz  $\rightarrow$  sok törtrész bit kell
  - Rögzített bitszámnál kompromisszum kell
- Megoldás  $\rightarrow$  Skálázó tényező alkalmazása

# Lebegőpontos számformátum

- Középiskolából ismerjük a számok normál alakját (pl.  $6.0 \cdot 10^{23}$ ). A **lebegőpontos formátum** ezt modellezi, a választott számrendszer szerinti skálázó tényező ( $r^k$ ) használatával

$$D = (-1)^e \cdot m \cdot r^k$$

- ahol  $e$  az előjel (pozitív: 0, negatív: 1),  $m$  a mantissza,  $r$  a számrendszer (radix) (2 vagy 10),  $k$  a kitevő (előjeles szám). A szabvány több méretet definiál (32/64/128 bit).
- Nagy értéktartomány, nagyon nagy és nagyon kicsi számok is ábrázolhatók.
- Egyenletes relatív pontosság  
Pl. az IEEE754 szabvány szerint, 32 biten a formátum a következő:  $e=1$  bit,  $m=24$  (23+1) bit,  $k=8$  bit, és az érték

$$(-1)^e \cdot (1+m) \cdot 2^{(k-127)}$$

Értéktartománya: 32 biten maximum  $\pm 3,4 \cdot 10^{38}$  A legkisebb értékei  $\pm 1,4 \cdot 10^{-45}$

# Decimális számábrázolás

- Bináris számábrázolás esetén a legegyszerűbbek a hardver műveletvégzők (összeadó, kivonó, szorzó, osztó). Ezért a digitális készülékek bináris számokkal számolnak.
- **Azonban szükség lehet a *decimális* értékre** (kijelzésnél) vagy akár decimális aritmetikára (pontosabb banki számítások)
- **Két megoldás lehetséges, feladattól függ a választás**
  - Decimális adatok tárolása (nem hatékony), decimális műveletvégzés (bonyolultabb), de közvetlen decimális eredmény.
  - Bináris adat tárolás (hatékony), bináris műveletvégzés (egyszerűbb), **kijelzés előtt BIN → DEC konverzió**



# Decimális számábrázolás

- Decimális számjegyek (0-9) kódolása, ábrázolása
- A binárisan kódolt decimális (**BCD**, **B**inary **C**oded **D**ecimal) kódban a bitek **számjegyenként 4 biten**, természetes 8-4-2-1 súlyozással szerepelnek
- Érvényes **NBCD** kódok értéktartmánya: **0000-1001** (Pl. 1010 nem érvényes kód)
- Pl. 1986 NBCD kódja:  
0001\_1001\_1000\_0110  
1      9      8      6

Érték	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

# Egyéb kódok

- A digitális technikában a kódoknak csak kis részét használják számok ábrázolására. Most két elterjedt másféle kódot ismertetünk.

## 1-az-n-ből kód

- Az  $n$ -bites kódban az  $n$  db bitből 1 db 1 értékű.
- Pl. 1-a-6-ból kód.  
Előnye, hogy digitális hardverrel könnyen generálható, és könnyen dekódolható.

1-a-6-ból
100000
010000
001000
000100
000010
000001

# Egyéb kódok

## ASCII kód

- Karakterek ábrázolására használják.
- Eredeti ASCII (American Standard Code for Information Interchange) karaktertáblázat 7 bites, 128 db kódszó, pl.

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EH	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Egy karakter kódja hexadecimálisan úgy olvasható ki a táblázatból, hogy a karakter sorának elején és oszlopának tetején levő egy-egy hexadecimális számot egymás után írjuk.

Pl. ,A' =  $41_{16} = 100\_0001_2$  ,0' =  $30_{16} = 0011\_0000_2$



# Hardver tervezés CAD rendszerrel

- A hardver tervezés során gyakran használnak programozható logikai elemeket (FPGA, CPLD stb.). A laborban FPGA-t használunk (Xilinx XC3S250E típusút).
- A programozható logikai elemek tervezését (is) CAD rendszerek segítik. Mi a XILINX ISE rendszert használjuk.

## A terv bevitele

- A hardver terv bevitelére több lehetőség is van.
  - Grafikusan, a kapcsolási rajz bevitelével.  
Itt használhatunk előre elkészített könyvtári elemeket. (Az első laborban ezt ismertük meg.) Azonban lehetőség van saját logikai elem készítésére is.
  - Bevihetjük a tervet **hardver leíró nyelv** (Hardware Description Language, HDL) használatáva. Erre a továbbiakban a **Verilog** HDL nyelvet fogjuk használni.

# Hardver tervezés CAD rendszerrel

## Szintézis

- A CAD rendszer a terv bevitele és szintaktikai ellenőrzése után, a megadott típusú programozható logikai eszközhöz előállítja az eszközbe programozandó információt. Ez nevezik **szintézisnek**.
- A szintézis több lépésből áll, ezeket nem részletezzük. (Többek között itt történik a logikai függvények egyszerűsítése is.)

# Hardver tervezés CAD rendszerrel

## Szimuláció

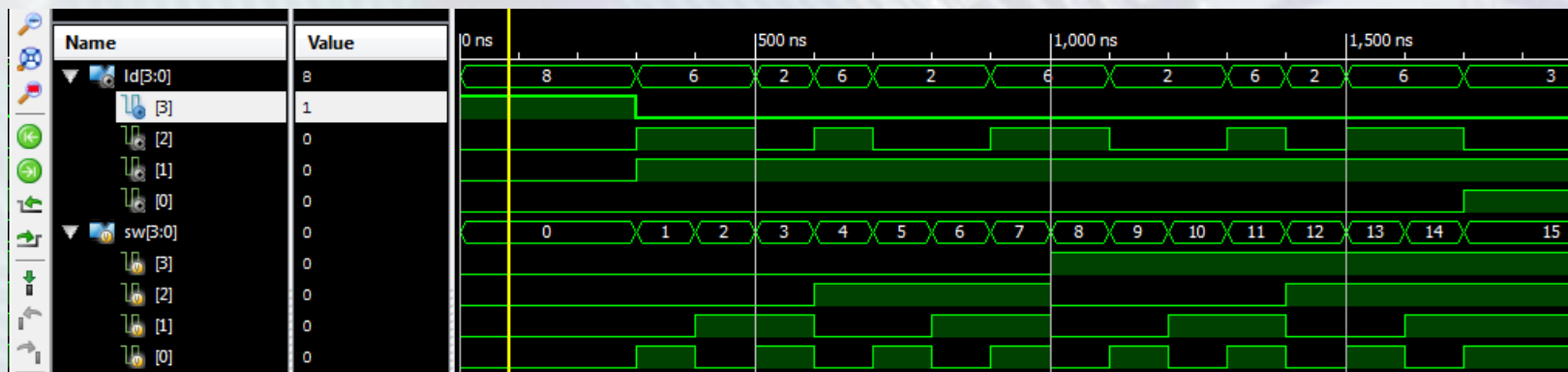
- Az eszközbe programozás előtt lehetőség van hardver nélkül ellenőrizni (tesztelni) a tervet. Ezt a célt szolgálja a szimuláció.
- A CAD rendszer a szintézis által generált adatok alapján elkészíti a logika funkcionális és időzítés **modelljét**.
- A **teszteléshez a logika bementére** a valós működést szimuláló **tesztadatokat kell generálni**. (A tesztadatok megadása is a tervezés része.)
- A modell és tesztadatok alapján a CAD rendszer **szimulátor** része **kiszámítja, hogy milyen kimenő adatok jelennek meg a megtervezett logika kimenetén**.



# Hardver tervezés CAD rendszerrel

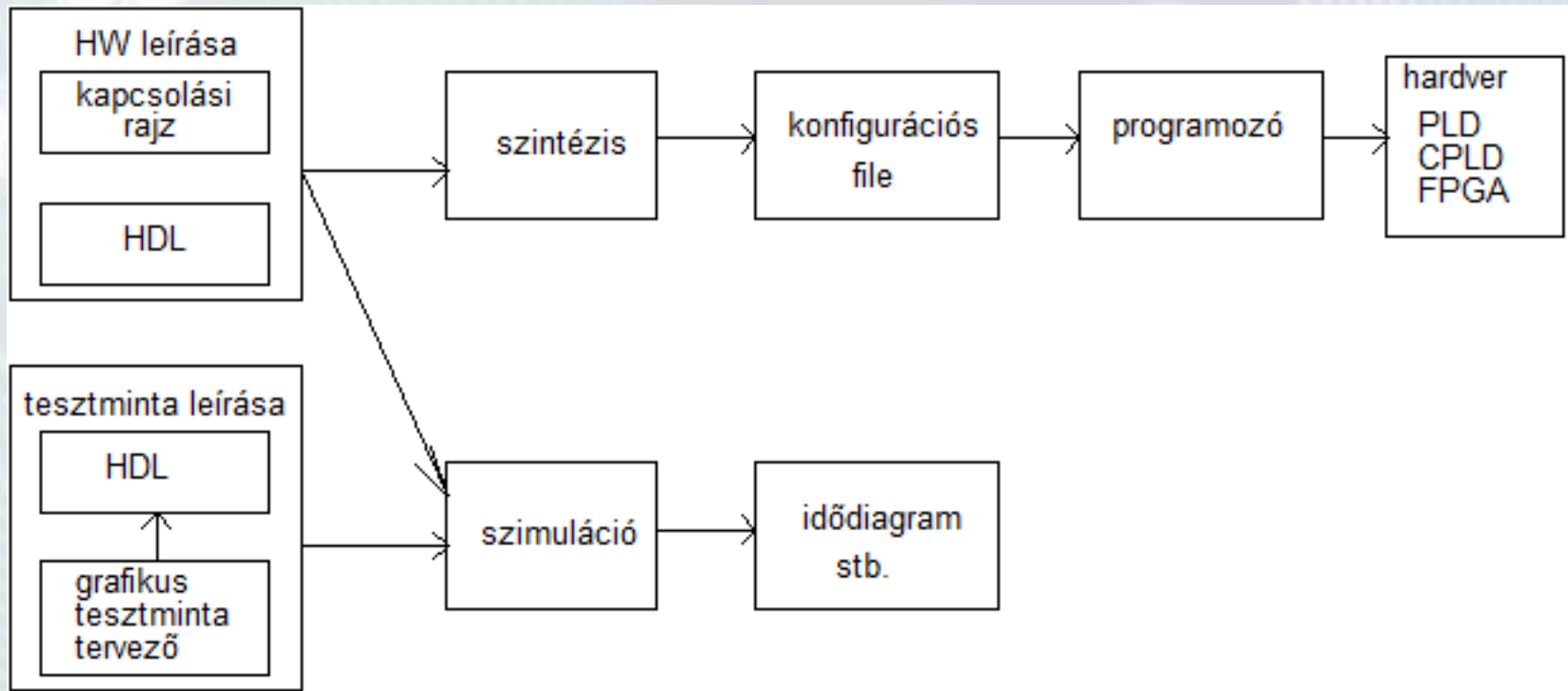
## Szimulációs eredmény megjelenítése

- A szimulátor a **kimeneti adatokat meg tudja jeleníteni grafikusan is.**
- Így a szimulációs eredmények hasonló formában jelennek meg, mintha egy valós logika bementére adtuk volna az tesztadatokat és a be és kimeneti jeleket megfelelő műszer (logikai analizátor) képernyőjén néznénk:



# Hardver tervezés CAD rendszerrel

Az alábbi egyszerűsített ábrán összefoglaltuk a hardver tervezéshez használt CAD rendszer fentiekben leírt működését:



# Hardverleíró nyelvek

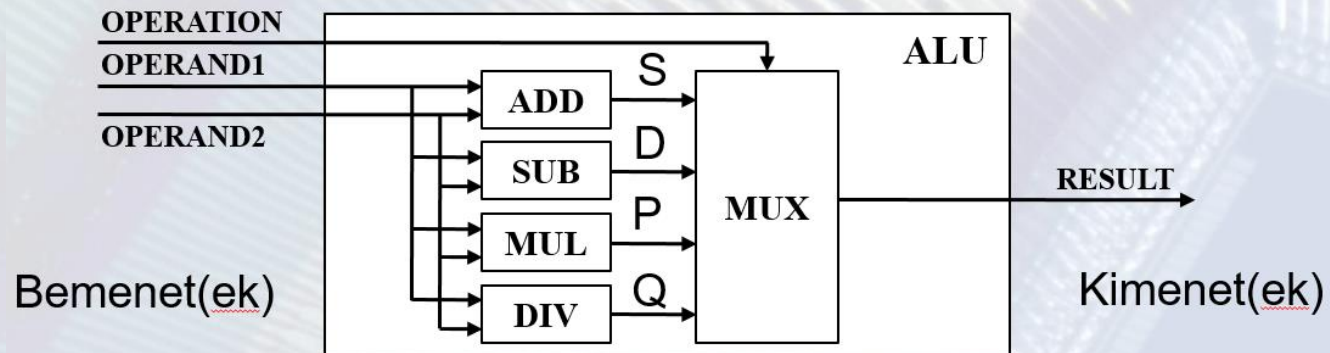
A hardverleíró nyelveket (HDL) digitális áramkörök **tervezéshez** (tervbevitel), **modellezéséhez** és **szimulálásához** fejlesztették ki.

A nyelvi konstrukciók sokszor hasonlítanak a megszokott programozási nyelvekben találhatókhöz. Azonban mivel **a hardver különböző részei egyszerre működnek**, a HDL-el is ilyen (párhuzamos) működés írható le, csak **kapcsolási rajz helyett szövegesen**. Azonban a szimuláció tesztmintáinak (időben egymást követő bemeneti jelek) leírására is képes.



# A Verilog hardverleíró nyelv

- A legismertebb HDL nyelvek az VHDL és a **Verilog**. Mi az utóbbit fogjuk használni.
- A Verilog nyelv **hierarchikus** (többszintű), funkcionális egység alapú tervezési megközelítést használ:
- A teljes rendszer több kisebb **modulból** épül fel. Pl. Az alábbi rendszer legfelső szintje az ALU modul az alatta levő szinten 5 db modult tartalmaz (ADD, SUB, MUL, DIV, MUX). Verilogban a teljes rendszert szövegesen írjuk le.



# A Verilog hardverleíró nyelv

A modul a környezetével a be és kimeneti jelein keresztül tartja a kapcsolatot. Ezt nevezik a modul **interfészének**. A be és kimeneti jeleket a modul **portjainak**.

Egy 4 bites bemenetekkel és 5 bites kimenettel rendelkező összeadó modul Verilog leírása:

```
module ADD(input wire [3:0] OP1, OP2,  
           output wire [4:0] RESULT);
```

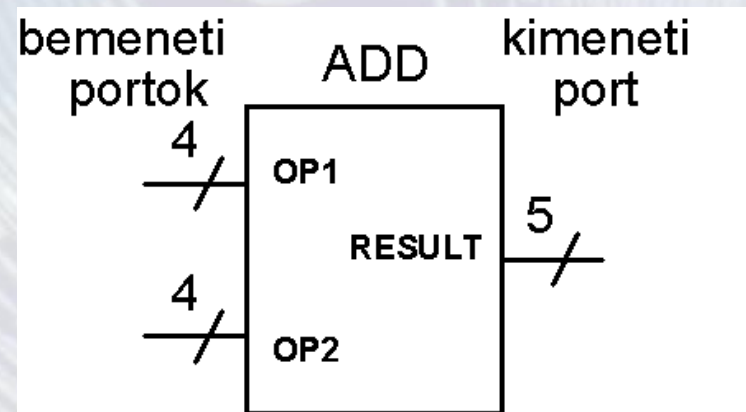
```
// a belső működés leírása:
```

```
assign RESULT = OP1 + OP2;
```

```
endmodule
```

Az **input** bemeneti, az **output** kimeneti portot jelöl. Ezek **típusát** (pl. wire) **nevét** és **méretét** (hány bites) utána kell megadni.

A **wire [3:0] op1, op2** 4 bites változókat jelöl, a **wire [4:0] RESULT** 5 bitest.



# A Verilog hardverleíró nyelv

A több bites változókat vektor változóknak is szoktuk nevezni. A **bitvektor bit tartományát** a változó név elé írt **[i:j]** jelöli. Ebből a bitszám is egyértelmű.

A **wire** olyan változó, amelynek a deklarálása után, később értéket adni csak **assign**-nal lehet. Sokszor modulok között jelek összerendelésére használjuk (összekötés, **huzalozás**). A neve is erre utal.

Az **assign változó = kifejezés**; jobb oldalán Verilog változókkal és operátorokkal leírható kifejezés állhat. A **wire** változóval **csak kombinációs hálózat valósítható meg**. Az értéke megváltozik, amint a fenti értékadás jobb oldali **kifejezése** megváltozik.

Mi leggyakrabban az alábbi operátorokat fogjuk használni:

**Aritmetikai operátorok:** +, -, \* (ahogy eddig is ismertük)

Pl. **assign c = a - b;** **assign c = -b;** **assign c = a\*b;**



# A Verilog hardverleíró nyelv

Logikai operátorok: **&** (AND), **|** (OR), **^** (EXOR), **~** (NOT)

Ezeket n bites változók között használva az eredmény is n bites lesz, a művelet az azonos bitek között végződik el.

Például:

```
wire [1:0] a,b,c;
```

```
assign c = a & b;
```

ugyanaz, mint:

```
assign c[0] = a [0]& b[0];
```

```
assign c[1] = a [1]& b[1];
```

**További részletek a Verilog\_bevező jegyzet 1-14 oldalán.**

# A Verilog hardverleíró nyelv

## Modul példányosítása

- Egy már elkészített modulból tetszőleges számú példány felhasználható egy másik modulban.
- Pl. Készítünk egy topmodult, melynek bemenetei a kapcsolók (sw[7:0]), kimenetei a LED-ek (ld[4:0]). Ebben felhasználjuk az előzőekben definiált ADD modul ADD0 példányát. Az ADD1 OP1 bemenetére rákapcsoljuk a topmodul sw[7:4] jeleit, az OP2 bemenetére rákapcsoljuk az sw[3:0] jeleit a RESULT kimenetére az ld[4:0] jeleit.

```
module top_mod(input [7:0] sw, output [4:0] ld);
```

```
//példányosítás:
```

```
ADD ADD0(.OP1(sw[7:4], .OP2(sw[3:0], .RESULT(ld[4:0])) ;
```

```
endmodule
```

Külső jel összekötése a modul jelével: **.modul\_jele(külső\_jel)**

# 1. EA vége