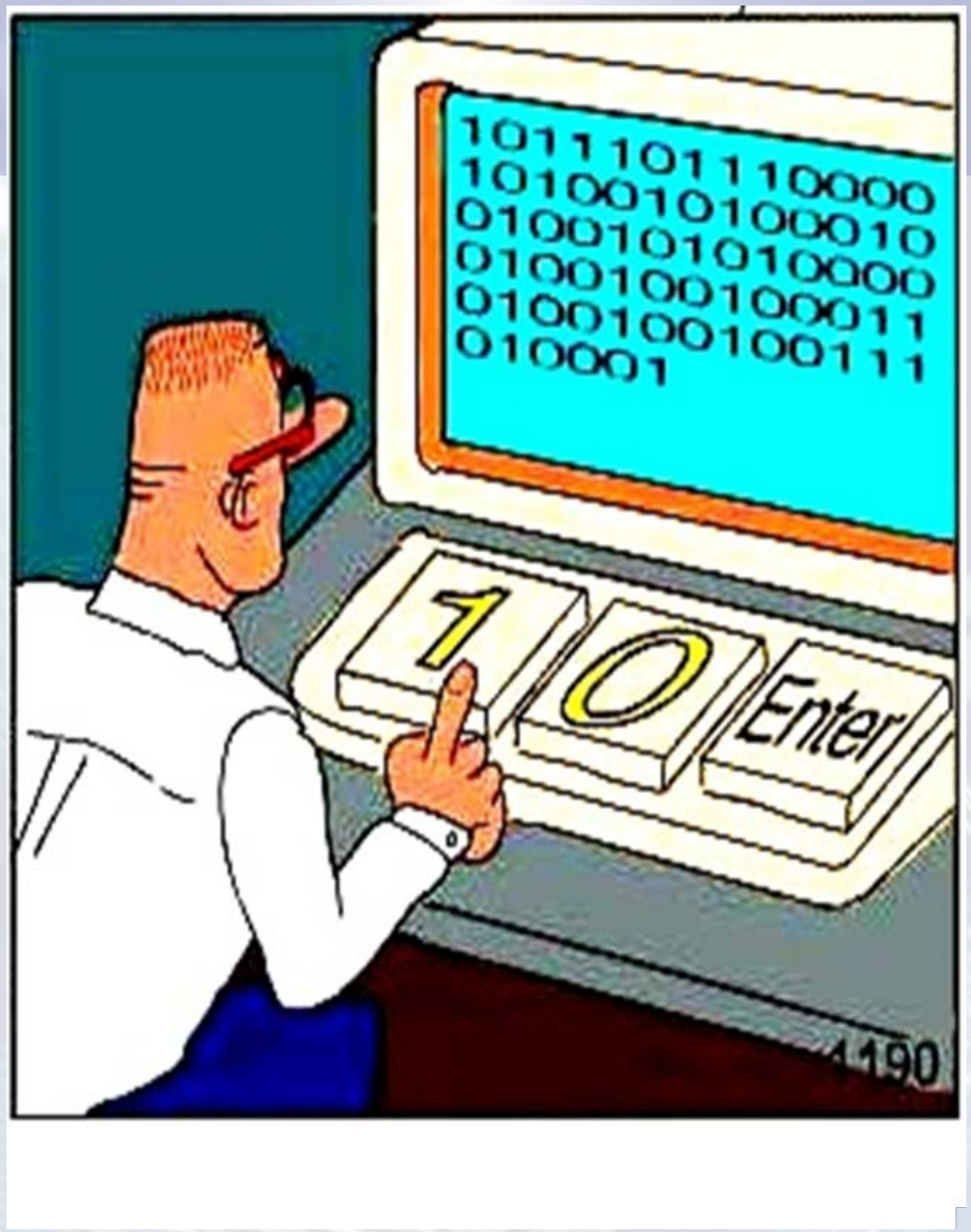
BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
MÉRÉSTECHNIKA ÉS INFORMÁCIÓS RENDSZEREK TANSZÉK

Digitális technika
VIMIAA01

Fehér Béla
BME MIT


10111011110000
10100101010010
01001010100010
01001010100000
01001001000011
01001001001111
010001
1
0
Enter
1190

Digitális technika

- **Rövid visszatekintés, összefoglaló a félévről**
  - **Komoly elképzelésekkel indultunk**
  - **Nehéz félévre számítottunk**
  - **A tárgy jelentős témakört fed le**
  - **Fontos mérnöki, informatikusi bevezető téma**
  - **Az alapoktól indulva több területet is átfog**
  - **HW elemek bemutatása, tervezése, használata**
  - **Összetett processzoros rendszerek egységei**
  - **SW programozási módszerek gépközeli szinten**
  - **Egyedi eszközök, speciális perifériák használata**

Digitális technika

- **Rövid visszatekintés, kiegészítés a félév elejéhez**
  - **Számábrázolások: Bináris, hexadecimális**
  - **Kettes komplemens, (előjel nagyság, ofszet bináris)**
  - **Egész, racionális, (lebegőpontos)**
  - **Alapműveletek:**
    - **Összeadás, kivonás**
    - **(Szorzás)**
    - **(Osztás)**
  - **Számrendszer konverziók**
    - **Bináris → Decimális irány a fontosabb**

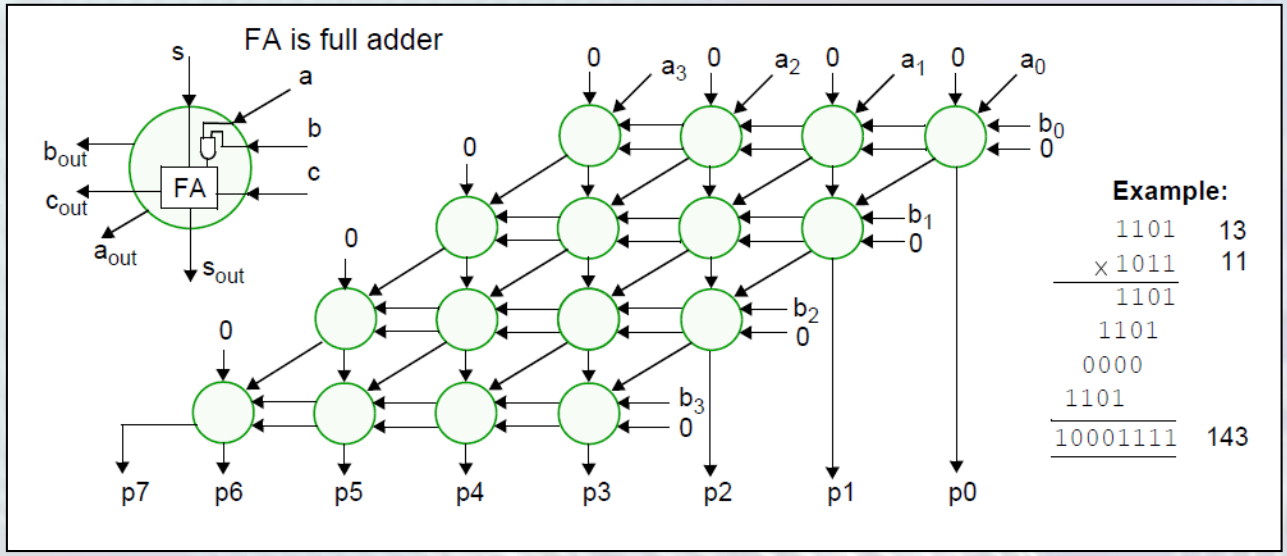
Digitális technika

- **Alapműveletek**
  - **Összeadás, kivonás**
    - **Egybites teljes összeadó, összeg, átvitel kimenet**
    - **Kettes komplemens számábrázolás, ADD/SUB**
  - **Szorzás**
    - **Félév elején csak az elemi bitszorzást tárgyaltuk**
    - **0*0=0, 1*0=0, 0*1=0, 1*1=1**
    - **Több bites adatokra bonyolult művelet**
      - **Bitszorzatok súlyozott összegzése**

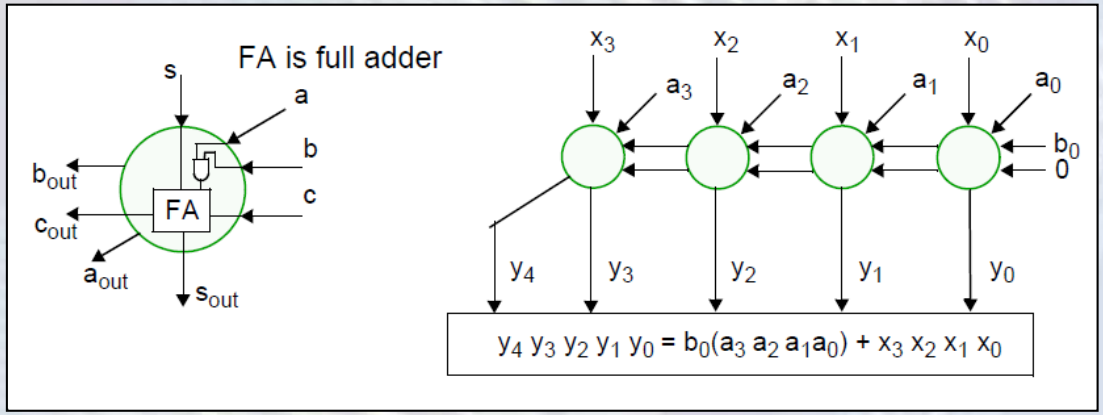
$$P = \sum_i^{N-1} \sum_j^{M-1} a_i * 2^i * bj * 2^j = \sum_i^{N-1} \sum_j^{M-1} a_i * bj * 2^{i+j}$$

Digitális technika - Szorzás

- **Párhuzamos tömbszorzó (HW)**



- **Iteratív soros-párhuzamos szorzó (HW vagy SW)**



Digitális technika - Szorzás

- **HW realizáció Verilog HDL nyelven**
- **Egyszerű másolata a blokkvázlatnak**
- **Léteznek ennél sokkal jobb szorzó megoldások**

```
module mul_4bitu(
   input  wire [3:0] multiplicand,        //Szorzandó
   input  wire [3:0] multiplier,          //Szorzó
   output wire [7:0] product              //Szorzat
);

//Változók a részeredmények kiszámításához.
wire [3:0] partial_product0, partial_product1;
wire [3:0] partial_product2, partial_product3;

wire [4:0] sum1, sum2, sum3;

assign partial_product0 = multiplicand & {4{multiplier[0]}};
assign partial_product1 = multiplicand & {4{multiplier[1]}};
assign partial_product2 = multiplicand & {4{multiplier[2]}};
assign partial_product3 = multiplicand & {4{multiplier[3]}};

// A szorzat a részszorzatok súlyozott iteratív összegzése
assign sum1 = {1'b0, partial_product0[3:1]} + partial_product1;
assign sum2 = sum1[4:1] + partial_product2;
assign sum3 = sum2[4:1] + partial_product3;

//A szorzat előállítása.
assign product = {sum3, sum2[0], sum1[0], partial_product0[0]};
```

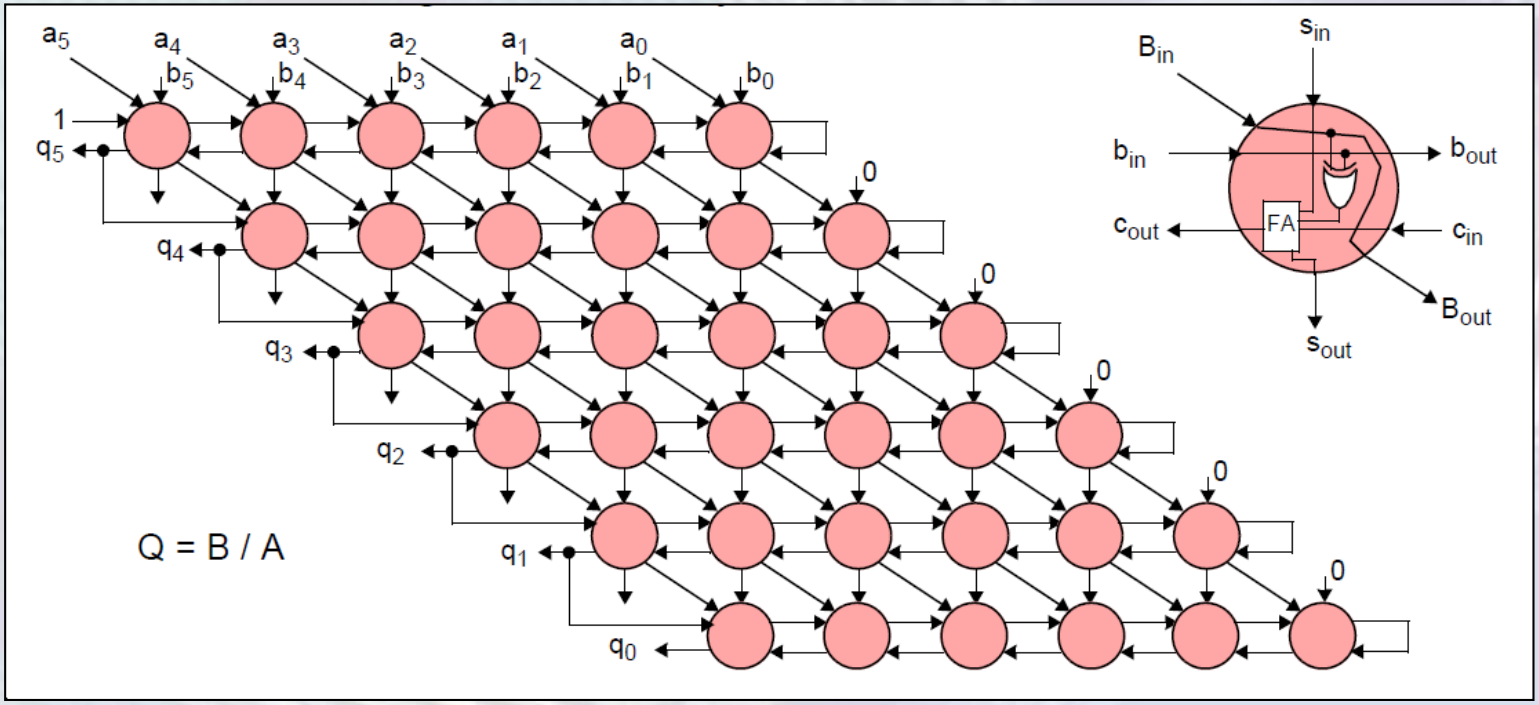
Digitális technika - Szorzás

- **SW realizáció, MiniRISC processzorra**
- **4 bites szorzás 8 bites regiszterekben**

```
start:
    mov     r0, SW          ; Beolvassuk a kapcsolók állapotát.
                            ; SW[7:4] = Szorzandó   SW[3:0] = Szorzó
    mov     r1, r0          ; A szorzót (SW[3:0]) az r1 regiszterbe másoljuk
    and     r1, #0x0f       ; és nullázzuk a felső 4 bitet.
    and     r0, #0xf0       ; Az alsó biteket nullázva r0 tarlmazza a szorzandót.
    sr0     r0              ; Ezt jobbra shiftelve rendelkezésre áll az első
                            ; részszorzat (ha így kezdjük, jobb az algoritmus).
    mov     r2, #0          ; r2 előkészítése a szorzat tárolására
                            ; A szorzást ciklusokban végezzük el, r2-ben
                            ; folyamatosan akkumláljuk a részszorzat összegeket
    mov     r3, #4          ; A 4 bites szorzáshoz a ciklust inicializáljuk
mul_loop:
    tst     r1, #0x08       ; Megvizsgáljuk a szorzó legnagyobb helyiértékű
    jz      no_add          ; bitjét. Ha 0, akkor nem kell a részszorzatösszegzés
    add     r2, r0          ; Hozzáadjuk a részszorzatot az r2 regiszterhez.
no_add:
    sr0     r0              ; A szorzandót jobbra kell shiftelni, hogy
                            ; megkapjuk a kövezkező részszorzatot.
    sl0     r1              ; A szorzót balra kell shiftelni, hogy a
                            ; következő vizsgálandó bit a 3. bitpozicióba kerüljön.
    sub     r3, #1          ; Csökkentjük a ciklusváltozót.
    jnz     mul_loop        ; Ha nem nulla még, akkor visszaugrás.

    mov     LD, r2          ; A szorzatot megjelenítjük a LED-eken.
    jmp     start           ; Ugrás a program elejére.
```

Digitális technika - Osztás

- **Hasonlóan származtatható, a vízszintes sorok adott méretű feltételes kivonók**
- **A következő szint vezérlése az aktuális maradék MSb bitjével történik (ez egyúttal a hányados egy bitje is)**



Digitális technika - Osztás

- **A pozitív számok osztása Verilog HDL nyelven**
- **Feltételes kivonás: (osztandó-osztó) > 0 vizsgálata**
- **Ha igen, q[i]=1 és a különbség az új maradék, egyébként marad az előző részeredmény**

```
// Az osztás már nem szintetizálható művelet,
// az "assign quo = a_inp / b_inp;" sajnos nem működik
// Saját megoldás szükséges hozzá
wire [3:0] a_inp, b_inp;
wire [3:0] quo;
wire [7:0] tmp1, tmp2, tmp3, tmp4;
wire [7:0] rem1, rem2, rem3;

assign tmp1 = {4'b0, a_inp} - {1'b0, b_inp, 3'b0};
assign quo[3] = ~tmp1[7];
assign rem1 = quo[3] ? tmp1 : {4'b0, a_inp};

assign tmp2 = rem1 - {2'b0, b_inp, 2'b0};
assign quo[2] = ~tmp2[7];
assign rem2 = quo[2] ? tmp2 : rem1;

assign tmp3 = rem2 - {3'b0, b_inp, 1'b0};
assign quo[1] = ~tmp3[7];
assign rem3 = quo[1] ? tmp3 : rem2;

assign tmp4 = rem3 - {4'b0, b_inp};
assign quo[0] = ~tmp4[7];
```

Digitális technika - Osztás

- **Az osztás is realizálható természetesen a MiniRISC processzoron assembly nyelven**

```
;*******************************************************************************
;* 8 bites előjel nélküli osztás.                                              *
;* Paraméterek:                                                                *
;*    r8: Osztandó (8 bites nemnegatív egész)                                  *
;*    r9: Osztó    (8 bites pozitív egész)                                     *
;* Visszatérési érték:                                                         *
;*    r8: Hányados                                                             *
;*    r9: Maradék                                                              *
;*******************************************************************************
divide:
    mov     r10, #0     ; Nullázzuk a maradékot.
    mov     r11, #0     ; Nullázzuk a hányadost.
    mov     r12, #8     ; 8 bitet kell shiftelni.
div_loop:
    sl0     r8          ; A maradék:osztandó regiszterpárt balra léptetjük.
    rlc     r10
    mov     r13, r10    ; Tároljuk a maradékot a helyreállításhoz.
    sub     r10, r9     ; A maradékból kivonjuk az osztót.
    jnc     no_restore  ; Ha a maradék >= osztó (C=0), akkor nincs helyreállítás.
    mov     r10, r13    ; Helyreállítjuk a maradék kivonás előtti értékét.
no_restore:
    rlc     r11         ; A C flag negáltját a hányadosba léptetjük.
    xor     r11, #0x01
    sub     r12, #1     ; Csökkentjük a hátralévő shiftelések számát.
    jnz     div_loop    ; Ugrás, ha az osztás még nem ért véget.
    mov     r9, r10     ; Az r9-be írjuk a maradékot.
    mov     r8, r11     ; Az r8-ba írjuk a hányadost.
    rts                 ; Visszatérés a hívóhoz.
```

Digitális technika – BIN2BCD

- **Tipikus igény, elsősorban felhasználói interfészeknél**
- **A bináris/hexadecimális kijelzés nem elfogadható**
  - **Bár érdekes szellemi torna a megfejtése…**
- **A párhuzamos konverzió viszonylag HW igényes**
  - Kis bitszámra egyszerű memóriatáblázat
  - Többszintű, iteratív konverzió-korrekció 4 bites értékeken (ADD3 + SHIFT algoritmus)
- **Soros konverzió**
  - Gyakran kijelzéshez használjuk, időmultiplex eszközzel (~10ms ciklusidő). Bőven van idő a soros konverzióra, nem zavaró az átmeneti részeredmény

Digitális technika – BIN2BCD

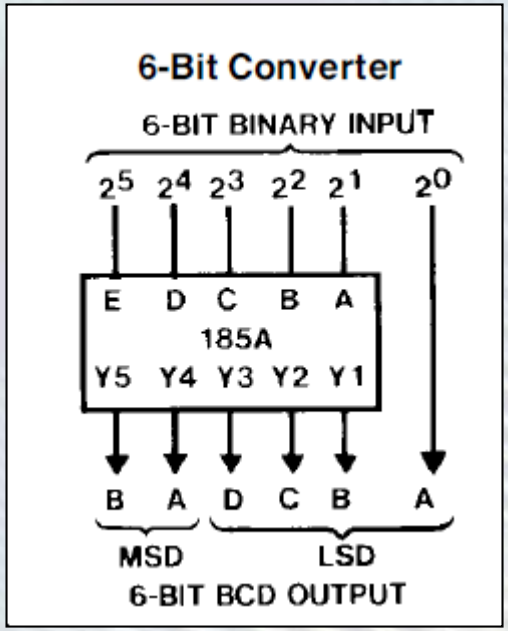
- **A dec[3:0] = DEC(bin[3:0]) 4 bites digiteken működő egyszerű logikai függvény többszörös beépítésével**

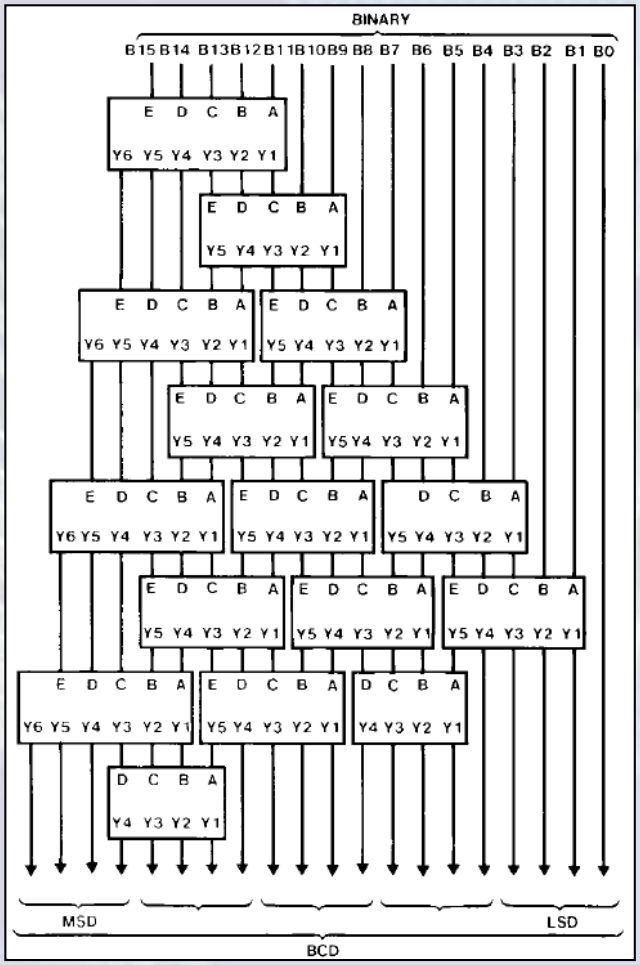
```
module BIN2BCD(
                input [15:0] binary,        // Binary input 16 bit 0 - 65535
                output [19:0] bcd           // Decimal output 5 digit
               );

// Egyetlen 4 bites hexadecimális digit korrekciója-konverziója, a szokásos 3 hozzáadásos módszerrel
function  [3:0] dec (input [3:0] bin);      // BIN -> DEC konverziós függvény 4 bitre, 1 decimális jegyre
       dec = (bin < 5) ? bin : bin + 3;     // Ha a 4 bites bináris érték 5,6,7,8,9, akkor növeljük 3-mal
endfunction

// A konverzió egyetlen tömb áramkörrel történik
// A módszer lépésről lépésre konvertálja-korrigálja a 3 bites mezőket
wire [19:0] t0,t1,t2,t3,t4,t5,t6,t7,t8,t9,ta,tb,tc,td;   // Belső átmeneti változók

assign t0 = {4'b0,binary};                  // Kiterjesztés 20 bitre, az egyszerűbb leirhatóság érdekében
assign t1 = {t0[19:17],dec(t0[16:13]),t0[12:0]};
assign t2 = {t1[19:16],dec(t1[15:12]),t1[11:0]};
assign t3 = {t2[19:19],dec(t2[18:15]),dec(t2[14:11]),t2[10:0]};
assign t4 = {t3[19:18],dec(t3[17:14]),dec(t3[13:10]),t3[ 9:0]};
assign t5 = {t4[19:17],dec(t4[16:13]),dec(t4[12: 9]),t4[ 8:0]};
assign t6 = {t5[19:16],dec(t5[15:12]),dec(t5[11: 8]),t5[ 7:0]};
assign t7 = {t6[19:19],dec(t6[18:15]),dec(t6[14:11]),dec(t6[10:7]),t6[6:0]};
assign t8 = {t7[19:18],dec(t7[17:14]),dec(t7[13:10]),dec(t7[ 9:6]),t7[5:0]};
assign t9 = {t8[19:17],dec(t8[16:13]),dec(t8[12: 9]),dec(t8[ 8:5]),t8[4:0]};
assign ta = {t9[19:16],dec(t9[15:12]),dec(t9[11: 8]),dec(t9[ 7:4]),t9[3:0]};
assign tb = {ta[19:19],dec(ta[18:15]),dec(ta[14:11]),dec(ta[10:7]),dec(ta[6:3]),ta[2:0]};
assign tc = {tb[19:18],dec(tb[17:14]),dec(tb[13:10]),dec(tb[ 9:6]),dec(tb[5:2]),tb[1:0]};
assign td = {tc[19:17],dec(tc[16:13]),dec(tc[12: 9]),dec(tc[ 8:5]),dec(tc[4:1]),tc[0]};

assign bcd = td;

endmodule
```

Digitális technika – BIN2BCD

- **Példa: 74185 TTL 6 bit Binary-to-BCD konverter**
- **Sok alkatrészt igényel a HW**

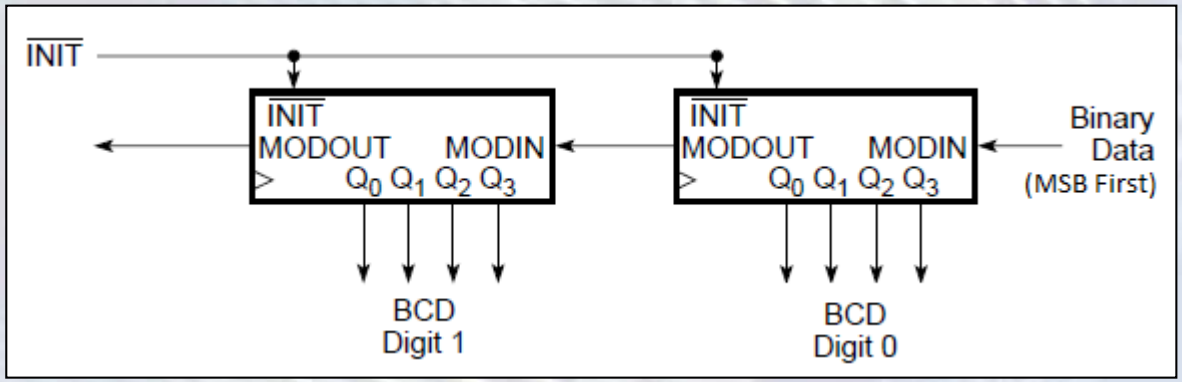
Input (Bits)	Packages Required
4 to 6	1
7 or 8	3
9	4
10	6
11	7
12	8
13	10
14	12
15	14
16	16
17	19
18	21
19	24
20	27





Digitális technika – BIN2BCD

- **Soros BIN2BCD konverter**
  - Lineáris költség, bitszám szerinti lépésben
  - Az ADD3 és SHIFT művelet egyetlen 4 bites egységbe van beépítve.
    - Amennyiben a 4 bites bitérték 5,6,7,8,9, akkor a léptetés (*2) eredménye a decimális feltételeknek megfelelően 0, 2, 4, 6, 8 lesz és 1 (tíz) továbblép



Digitális technika – BIN2BCD

- **Szoftveres algoritmus a MiniRISC processzoron**
- **4 bitenként korrekció és léptetés**

```
;*******************************************************************************
;* 2 digites bináris -> BCD átalakítás a SHIFT & ADD3 algoritmussal.            *
;* Paraméterek:                                                                *
;*   r8: A konvertálandó bináris érték.                                        *
;* Visszatérési érték:                                                         *
;*   r8: A BCD konverzió eredménye.                                            *
;*******************************************************************************
bcd_2digit:
    mov     r2, #0          ; Az r2 tárolja az eredményt ideiglenesen.
    mov     r1, #8          ; A hátralévő shiftelések száma.
bcd_loop:
    mov     r0, r2          ; Az egyesek vizsgálata: ha 5 vagy több, akkor
    and     r0, #0x0f       ; hármat hozzá kell adni.
    cmp     r0, #0x05
    jc      no_corr1
    add     r2, #0x03
no_corr1:
    mov     r0, r2          ; A tizesek vizsgálata: ha 5 vagy több, akkor
    and     r0, #0xf0       ; hármat hozzá kell adni.
    cmp     r0, #0x50
    jc      no_corr10
    add     r2, #0x30
no_corr10:
    sl0     r8              ; Egy bittel balra shiftelünk.
    rlc     r2
    sub     r1, #1          ; Csökkentjük a shiftelések számát.
    jnz     bcd_loop        ; Ugrás, ha nincs még vége a konverziónak.
    mov     r8, r2          ; Az eredmény beírása az r8 regiszterbe.
    rts                     ; Visszatérés a hívóhoz.
```

Digitális technika - Vizsga

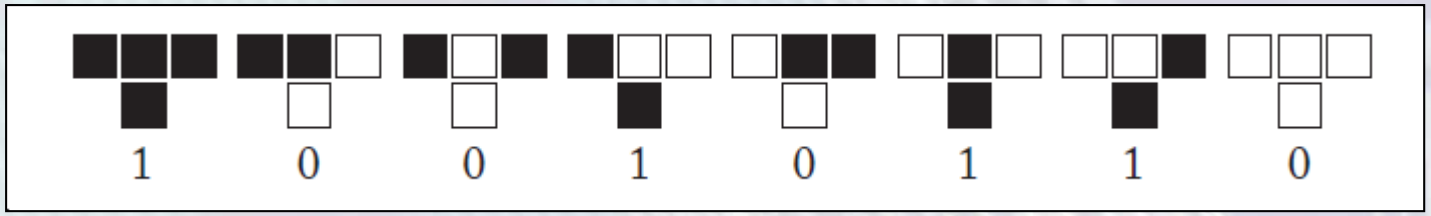
- **Minta vizsga feladatsor**
- **A vizsga tartalmaz kérdéseket**
  - Elemi digitális technikai ismeretekből
  - Verilog HDL tervezői ismeretekből
  - Mikroprocesszoros rendszerek HW ismeretekből
  - Mikroprocesszoros rendszerek SW ismeretekből (gépközeli, assembly szintű programozás)

Digitális technika – Cell. Automaták

- **Sorrendi hálózatok – Véges állapotú vezérlők**
- **Definíciójuk szerint: {I, S, S_0, C1, C2, O}**
  - {Bem, Áll, Kezd.áll, Köv.áll.fgv, Kim.fgv,Kim ért.}
- **A celluláris automaták (CA) is hasonlók, kicsit absztraktabb konstrukció {Z, S, N, f}**
  - Z: Véges vagy végtelen cellamező, 1D, 2D
  - S: Az elemei cellák állapota, ezek összegzése a globális konfiguráció
  - N: A szomszéd cellák halmaza, a szomszédsági szabály alapján (Neumann-◆ vagy Moore-■ alakú szomszédosság)
  - f: A lokális állapot átmeneti függvény, azaz „ a szabály” „Rule”

Digitális technika – Cell. Automaták

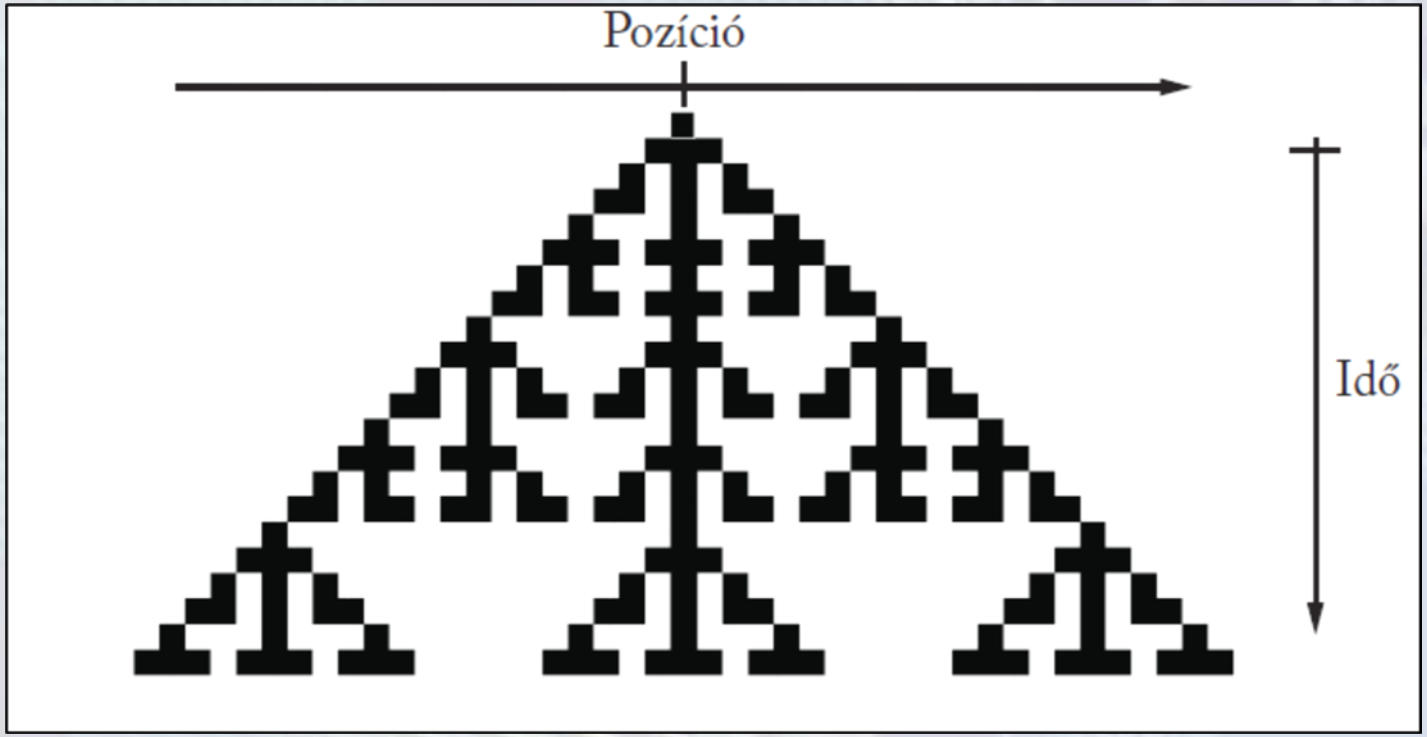
- **A CA lehet egy vagy több dimenziós**
- **Elemi celluláris automaták:**
  - Egydimenziós, lineáris automata, 2 cellaállapottal, 0 vagy 1 és minimális szomszédsággal



  - 8 bemeneti kombináció (111, 110,…001, 000)
  - Összesen csak 256 ilyen automata létezik
  - A kimeneti bitminta az azonosító:
    $1\,0\,0\,1\,0\,1\,1\,0_{bin} = 150_{dec}$ Wolfram-szám

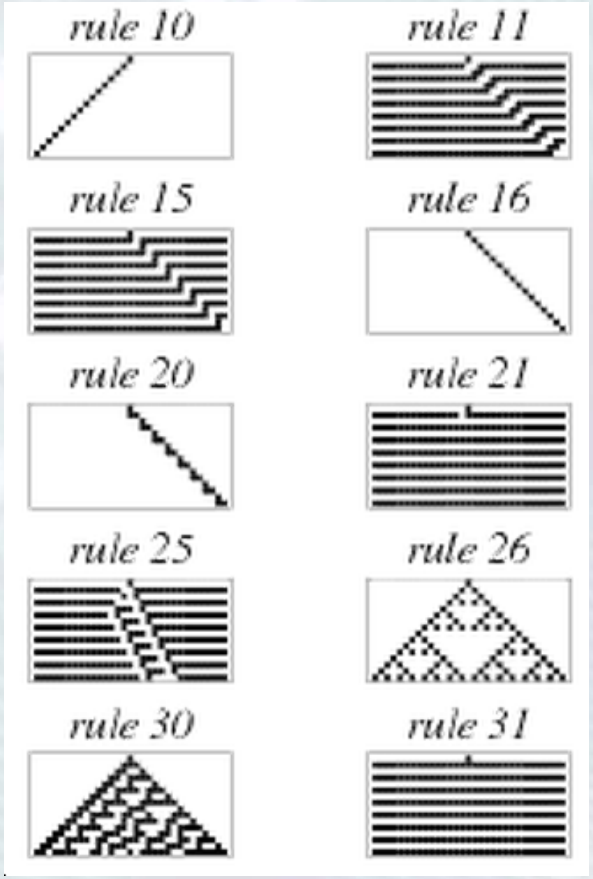
Digitális technika – Cell. Automaták

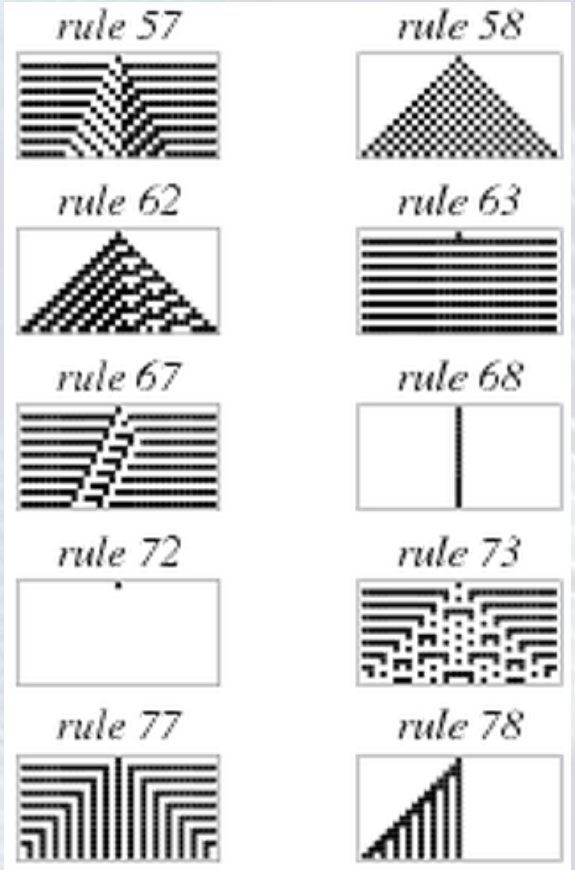
- **Néhány érdekesség**
  - A Rule 150 1D automata evolúciója
  - A kezdőállapot vektor: {…0000001000000….}

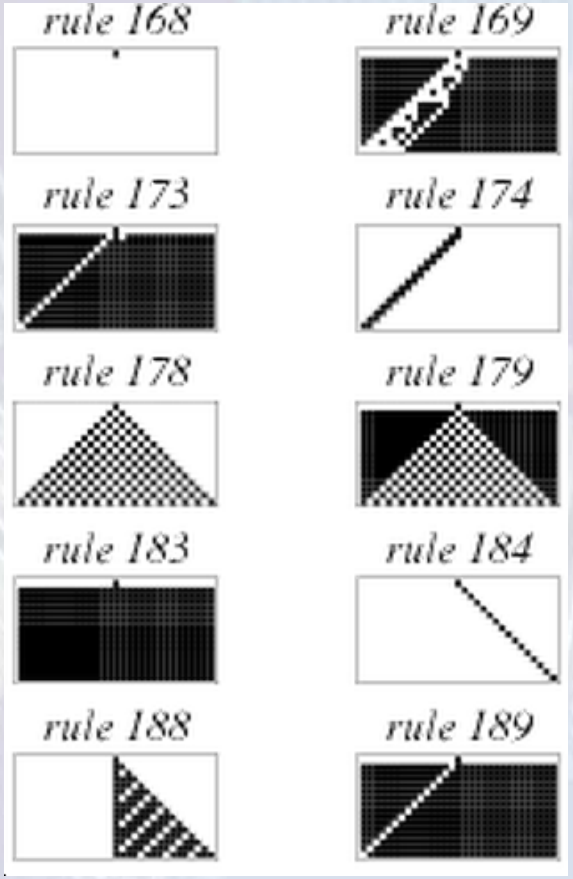


Digitális technika – Cell. Automaták

http://mathworld.wolfram.com/ElementaryCellularAutomaton.html

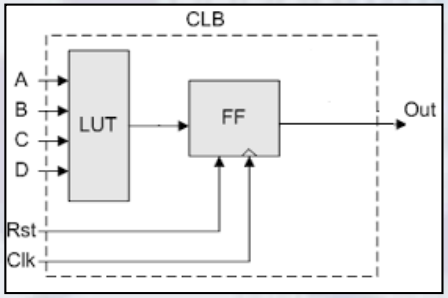


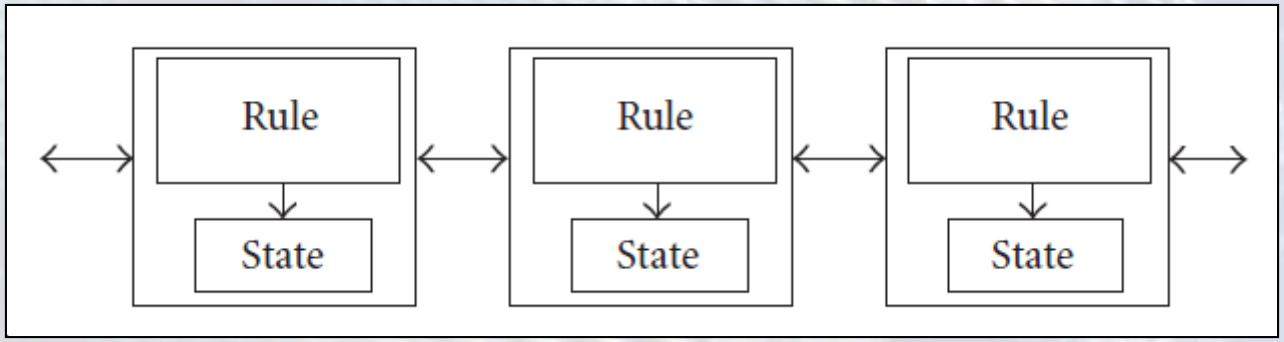




Digitális technika – Cell. Automaták

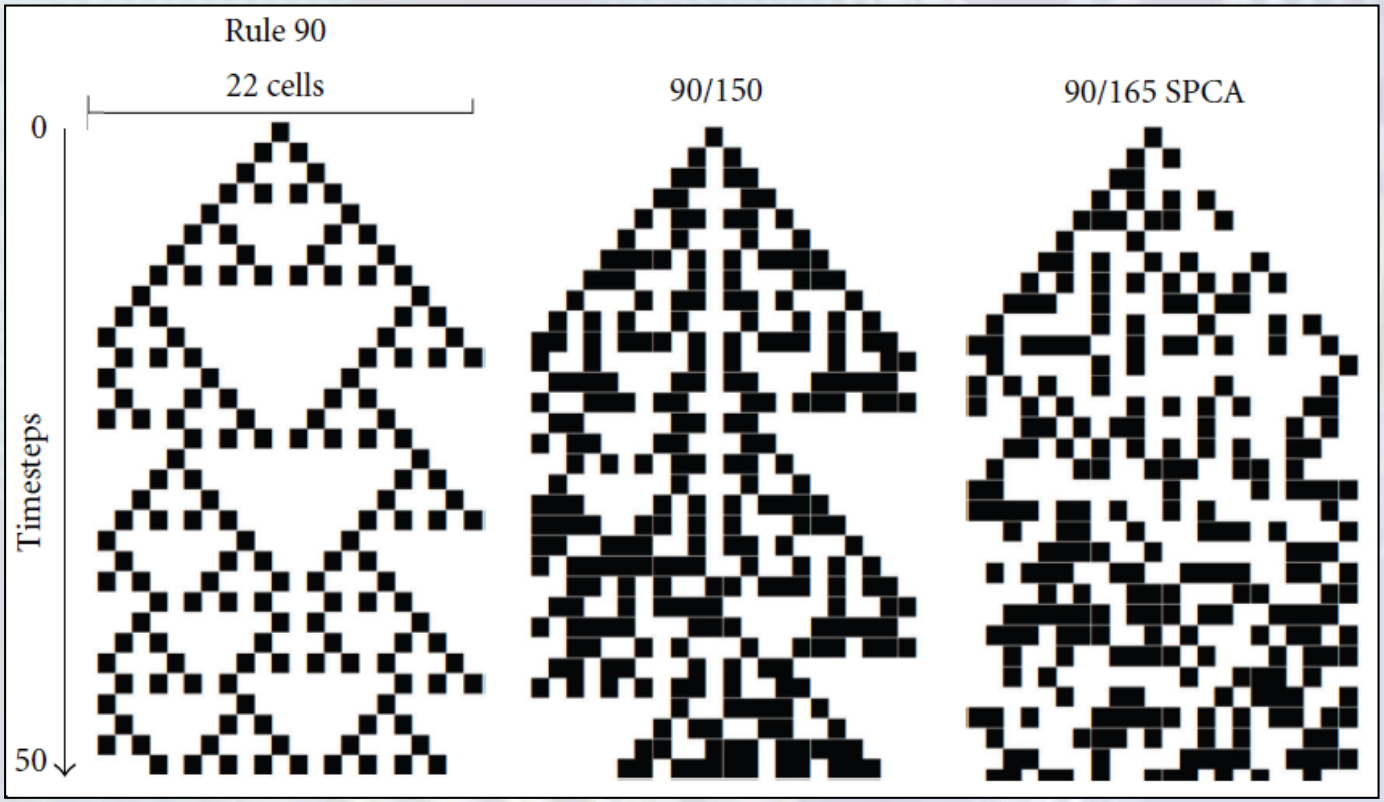
- **Elemi automaták HW realizációja**
  - Egyszerű 3 bemenetű függvény a szabályhoz + 1 bit DFF állapottároló
  - Az FPGA elemei cellája:
  - Egyetlen cella minimális költséggel
  - A „szabály” a LUT-ba léptethető





Digitális technika – CA Alkalmazások

- **Kriptográfiai területen**
- **Véletlenszám generálás: Homogén szabállyal vagy heterogén (több) szabály használatával**



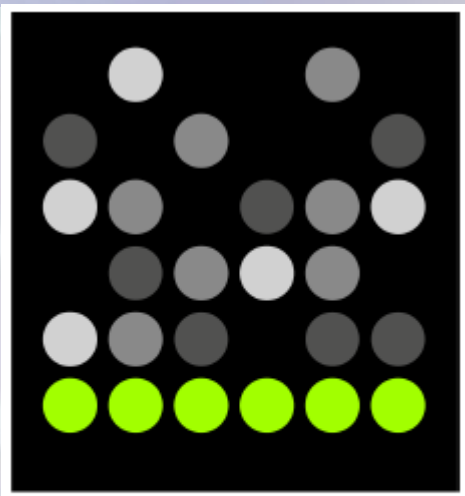
Digitális technika – CA Alkalmazások

- **Általános számítástechnika: Wire World: Logikai függvények, tárolók modellezése**
  - Fehér: háttér, Fekete: vezeték,
    Sárga: Elektromos hatás eleje, ami a vezetéken végigfut
    Piros: Elektromos hatás vége, követi a sárgát
  - Szabály: 2D 8 szomszédos kiértékelés

mathworld.wolfram.com/WireWorld.html

Digitális technika – CA Alkalmazások

- **Micron Automata Processor**
  - Teljesen új megközelítés
  - A számítási egységek és a memória egybe ötvözése speciális feladatokra
  - Alkalmazás: „Big Data” keresések, párhuzamos mintafelismerés, video és képfeldolgozás,



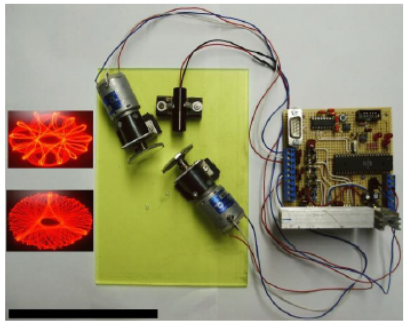
Digitális technika

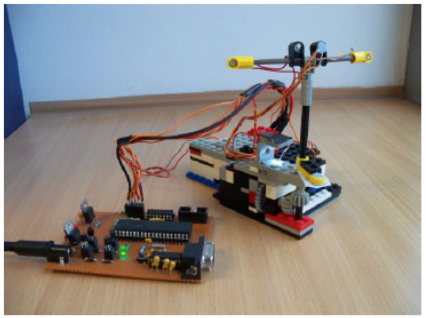
- **Aki továbbra is érez kedvet a témához, szeretne érdekes feladatok kapcsán jobban elmélyülni**

MIKROKONTROLLEREK ALKALAMZÁSTECHNIKÁJA (VIMIJV51)
szabadon választható tárgy (csak tavaszi félévekben indul)

- Mikrokontrollerek működése részletesen
- Mikrokontrolleres perifériák működése, programozása, alkalmazása
- Mikrokontroller programozás C nyelven
- Készülékfejlesztés lépései
- Házi feladatkeretében saját ötleten alapuló készülék építése (a vizsga jegy 49%-át adja), a készülék a hallgatóé marad

Régebbi házi feladatok képei





Digitális technika

Sikeres félévzárást, kellemes ünnepeket,
és eredményes vizsgaidőszakot kívánunk!

A tárgy oktatói