C programozás – aritmetikai, logikai, összehasonlító és bitszintű operátorok

Mikrovezérlők GPIO kezelése

Tartalomjegyzék

1. Bevezetés

Mikrovezérlő programozás esetén sok esetben regisztereket kell manipulálni. Ebből kifolyólag a következő operátorok használatával mindenképp tisztában kell lenni:

- aritmetikai,
- logikai,
- bitszintű,
- összehasonlító,
- értékadó.

1.1. Aritmetikai operátorok

Mint minden programozási nyelvben a következő aritmetikai operátorok (matematikai műveletek) érhetőek el:

- Összeadás: +
- Kivonás: -
- Szorzás: *
- Osztás: /
- Egész osztás: /
- Modulo osztás / maradék képzés: %

1.2. Logikai operátorok

ÉS kapcsolat: abban az esetben logikai **IGAZ** a kimenet, ha minden bemenet logikai **IGAZ**. Minden más esetben **HAMIS.**

Operátor: **&&**

a	**b**	**a && b**
0	0	0
0	1	0
1	0	0
1	1	1

1.1. táblázat: Logikai ÉS kapcsolat

VAGY kapcsolat: abban az esetben logikai **IGAZ** a kimenet, ha legalább egy bemenet logikai **IGAZ**.

Operátor: **||**

a	**b**	**a||b**
0	0	0
0	1	1

1	0	1
1	1	1

1.2. táblázat: Logikai VAGY kapcsolat

NEGÁCIÓ: a negáció művelet az aktuális érték inverzének a képzésére szolgál. Logikai **IGAZ** esetén logikai **HAMIS**, logikai **HAMIS** esetén logikai **IGAZ** lesz az eredmény.

Operátor: **!**

a	**!a**
0	1
1	0

1.3. táblázat: Logikai NEGÁCIÓ

1.3. Összehasonlító operátorok

Elágazások feltételének vizsgálata esetén alkalmazandó operátorok:

- Kisebb mint: **<**
- Nagyobb mint: **>**
- Kisebb vagy egyenlő mint: **<=**
- Nagyobb vagy egyenlő mint: **>=**
- Egyenlő: **==** (Figyelem! Az értékadó operátor „=”, nem összekeverendő!)
- Nem egyenlő: **!=**

1.4. Bitszintű operátorok

A mikrovezérlőkben található regiszterek beállítására szinte minden esetben bitszintű operátorokat használunk. A következő **BITSZINTŰ műveletek** érhetőek el:

- shiftelés (biteltolás jobbra vagy balra): >>, <<
- ÉS kapcsolat: &
- VAGY kapcsolat: |
- 1-es komplemens: ~
- XOR (kizáró vagy): ^

Balra shiftelés pl.: *unsigned char data = (1<<3)*

- Ebben az esetben a kiindulási érték az operátor baloldalán szereplő decimális 1, binárisan: **0b00000001**.
- A „<<” operátor balra shiftelést jelent, még pedig annyi helyiértékkel, amennyi az operátor jobb oldalán szerepel. Ebben az esetben 3, azaz a *data* változó bitjeit balra el kell tolni 3-al. Figyelem, a kilépő bitek (ebben az esetben nincs) elvesznek!
- Az eredmény binárisan: **0b00001000, deicmálisan: 8**

Jobbra shiftelés pl.: *unsigned char data = (128>>2)*

- Ebben az esetben a kiindulási érték az operátor baloldalán szereplő decimális 128, binárisan: **0b10000000**.
- A „>>” operátor jobbra shiftelést jelent, még pedig annyi helyiértékkel, amennyi az operátor jobb oldalán szerepel. Ebben az esetben 2, azaz a *data* változó bitjeit jobbra el kell tolni 2-vel. Figyelem, a kilépő bitek (ebben az esetben nincs) elvesznek!
- Az eredmény binárisan: **0b00100000, decimálisan: 3**

Figyeljük meg a fenti két shiftelés példát. Az első esetben balra shifteltük a decimális 1-et 3-al, az eredmény 8 lett, a jobbra shiftelésnél a 128-at shifteltük jobbra 2-vel, az eredmény 32 lett. A balra shiftelés megegyezik a 2-vel való szorzással, ebben az esetben: (1 << 3) = 1 × 2 ×2 ×2 = 8 művelettel egyezik meg. A jobbra shiftelés osztással: (128 >> 2) = 128 / 2 / 2 = 32.

Abban az esetben, hogy ha gyors 2-vel való szorzás vagy osztásra van szükség (maradék nélküli) akkor a leggyorsabb művelet a shiftelés!

ÉS kapcsolat pl**:** *unsigned char data = 0b10111111; data &= 0b11110111;*

- A bit szintű **ÉS** kapcsolat segítségével könnyedén ki lehet maszkolni és törölni egy adott byte bitjét, még pedig úgy, hogy az **ÉS** operátor jobb oldalán álló értéknek minden bitje 1, kivéve az amit ki szeretnénk nullázni!
- A *data &= 0b11110111;* művelet eredménye: *0b10110111*, azaz jobbról a 4. bitet kinullázta, a többi maradt változatlanul!
- Ez egy fontos bittörlő művelet, a regiszterek beállításához elengedhetetlen!

VAGY kapcsolat pl: *unsigned char data = 0b01000000; data |= 0b00000100;*

- A bit szintű **VAGY** kapcsolat segítségével könnyedén ki lehet maszkolni és be lehet állítani egy adott byte bitjét, még pedig úgy, hogy az **VAGY** operátor jobb oldalán álló értéknek minden bitje 0, kivéve az, amit be szeretnénk állítani!
- A *data |= 0b00000100;* művelet eredménye: *0b01000100*, azaz jobbról a 3. bitet beállította, a többi maradt változatlanul!
- Ez egy fontos bit beállító művelet, a regiszterek beállításához elengedhetetlen!

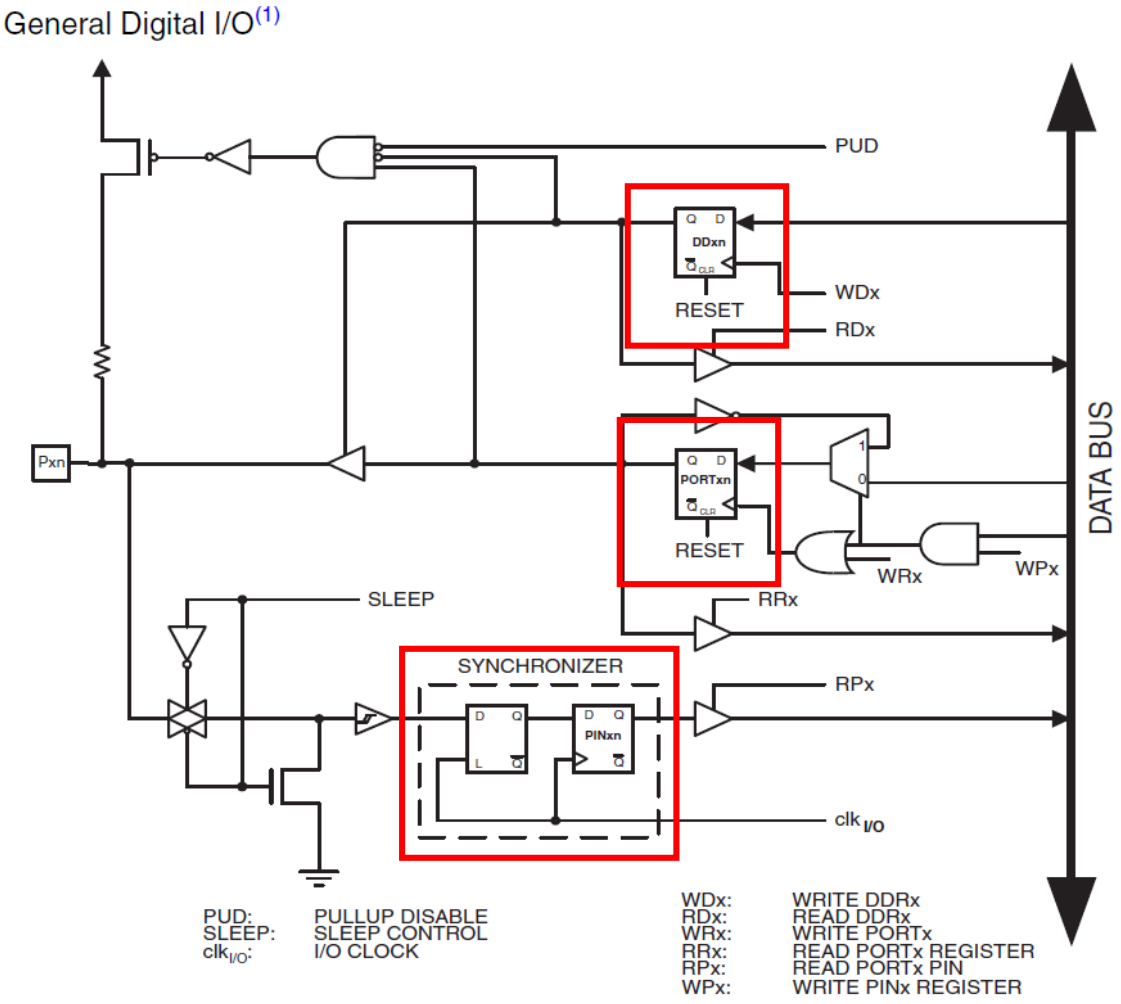
Egyes komplemens vagy negáció: pl. *unsigned char data = 0b00001111; unsigned char data2 = ~data;*

- Beágyazott rendszerek esetén a negáció szintén fontos művelet. Egy bit törlése könnyebben megadható a bit beállításához tartozó shiftelés eredményének a negálásával. (lsd. később)
- A példa kódban a *data2* eredménye: 0b11110000. Minden bitet az ellenkezőjére állított a „~” operátor.

2. ATMega2560 GPIO bemutatása

Az ATMega2560 mikrovezérlő összesen 86db GPIO porttal rendelkezik, melyek rendre a következő csoportokra vannak felosztva:

1. PORTA (PA0..PA7),
2. PORTB (PB0..PB7),
3. PORTC (PC0..PC7),
4. PORTD (PD0..PD7),
5. PORTE (PE0..PE7),
6. PORTF (PF0..PF7),
7. PORTG (PG0..PG5),
8. PORTH (PH0..PH7),
9. PORTJ (PJ0..PJ7),
10. PORTK (PK0..PK7),
11. PORTL (PL0..PL7)



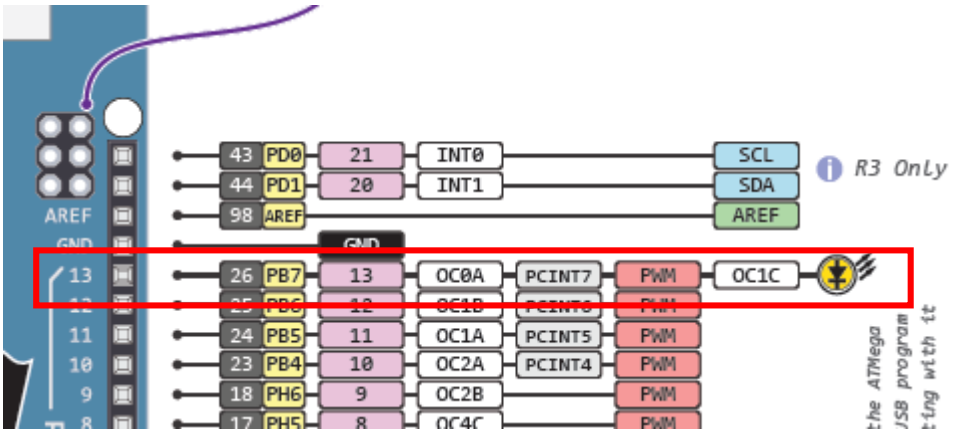
2.1. ábra: GPIO blokdiagram

Minden port 8bites (kivéve a G port) és használható általános ki, illetve bemenetként is! A portok I/O-ként való beállítására összesen 3db írható/olvasható regiszter áll rendelkezésre (2.1 ábra):

- **DDRx (Data Direction Register):**
  - 8 bites regiszter, 8db **PORT** tartozik hozzá
  - Az „x” ebben az esetben A, B, C, …, L érték lehet. A **DDRx** regiszterben lehet beállítani a PORT irányát (ki vagy pedig bement legyen). Alap állapotban minden port bemenet.
- **PINx (Port Input Register):**
  - 8bites regiszter
  - 8db port tartozik hozzá
  - A **PINx** regiszterből lehet kiolvasni az adott port logikai állapotát (pl. bemenet esetén LOW vagy HIGH értéken van).
- **PORTx (Port Output Register):**
  - 8bites regiszter
  - 8db port tartozik hozzá
  - A **PORTx** regiszter segítségével lehet beállítani az adott **PORT** logikai állapotát (**LOW** vagy **HIGH**).
  - Ha egy port bemenetként funkcionál, azaz a **DDRx** regiszter adott bitje 0 értéket vesz fel, akkor a **PORTx**-be írt (beállított portnak megfelelő) 1-essel lehet bekapcsolni a belső felhúzó ellenállást (pull-up resistor)! A felhúzó ellenállás szerepe egy később leckében kerül bemutatásra!

2.1. GPIO kezelés

Az alábbiakban egy egyszerű, LED villogtatásos példán keresztül bemutatásra kerül a GPIO kezelése. Az Arduino Mega fejlesztő panelen található LED-et használjuk.



2.2. ábra: Arduino Mega PinMap

A 2.2 ábra alapján a LED a PB7-es lábra van rákötve. Ahhoz, hogy a LED-et vezérelni tudjuk a következő regiszterekre lesz szükségünk:

- DDRB,
- PORTB,
- PINB (jelen esetben nem releváns, mivel kimenetként használjuk a portot, de bemutatásra kerül).

Az első lépés, hogy a DDRB regiszterben a portot kimenetre kell állítani. A DDRB regiszterhez összesen 8db port tartozik (2.3 ábra).

DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
0	0	0	0	0	0	0	0

2.3. ábra: DDRB regiszter

Alap állapotban minden PORT bemenetként funkcionál (mindenhol 0 érték szerepel). Ahhoz, hogy a LED-et meg tudjuk hajtani, a PB7-hez tartozó DDRB7 bitet be kell állítani 1-re. Ezt egy bitszintű operátorral tehetjük meg, még pedig a következő utasítással:

```
DDRB |= (1 << PB7);
```

Itt a DDRB-vel hivatkozunk a regiszterre, a PB7 egy konstans, aminek az értéke 7. A művelet hatására a DDRB 7. bitje bebillen és a PORT kimenetként funkcionál.

DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
1	0	0	0	0	0	0	0

2.4. ábra: PB7 port kimenetre állítva

Miután beállítottuk kimenetre a PB7 portot, a PORTB és a PINB regiszterek segítségével fogjuk a LED villogtatást megvalósítani.

A PORTB regiszter szintén 8 bites (2.5 ábra).

PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0	0	0	0	0	0	0	0

2.5. ábra: PORTB regiszter

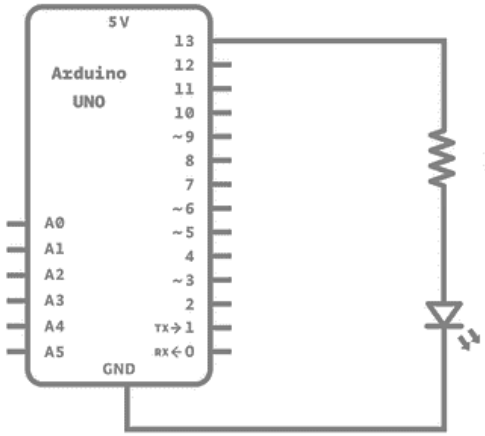
A LED villogtatásához a PORTB7 bitet kell 1-re vagy 0-ra állítani. 1 esetén a kimenet-en megjelenik az 5V, 0 esetén pedig a 0V (természetesen a feszültség attól függ, hogy a processzor milyen logikát használ).

Ha a portunk bemenetként funkcionál és kiszeretnénk olvasni a bemenetünk állapotát, akkor azt a PINB 8bites regiszter segítségével tehetjük meg (2.6 ábra).

PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0	0	0	0	0	0	0	0

2.6. ábra: PINB regiszter

Az Arduino fejlesztő panelen a 13-as lábon (PB7) található a LED.



2.7. ábra: 13-as lábhoz (PB7 Arduino Mega esetén) csatlakoztatott LED

A legtöbb LED általában 10mA körüli árammal hajtják meg, maximum 20-30mA szokott lenni. Minden esetben érdemes megnézni az adatlapot (már ha elérhető) és az alapján méretezni hozzá az ellenállást. Feltételezve, hogy a LED-en kb. 0.7V feszültség esik és a megfelelő fényerőhöz 10mA áram elegendő, az ellenállás értékét a következő egyenlettel határozhatjuk meg:

$$R = \frac{V_{in} - 0.7V}{I} = \frac{5V - 0.7V}{10\ mA} = 430\ \Omega$$

3. Fogalomtár

DDRx (Data Direction Register):

- Az „x” ebben az esetben A, B, C, …, L érték lehet. A **DDRx** regiszterben lehet beállítani a PORT irányát (ki vagy pedig bement legyen). Alap állapotban minden port bemenet.

GPIO (General Purpose Input/Output):

- A GPIO az általános célú bemenet/kimenet rövidítése. Szabványos interfész, amelyet a mikrokontrollerek és más elektronikus eszközök összekapcsolására használnak.

PINx (Port Input Register):

- Az „x” ebben az esetben A, B, C, …, L érték lehet.
- A **PINx** regiszterből lehet kiolvasni az adott port logikai állapotát (pl. bemenet esetén LOW vagy HIGH értéken van).

PORTx (Port Output Register):

- Az „x” ebben az esetben A, B, C, …, L érték lehet.
- A **PORTx** regiszter segítségével lehet beállítani az adott **PORT** logikai állapotát (**LOW** vagy **HIGH**).