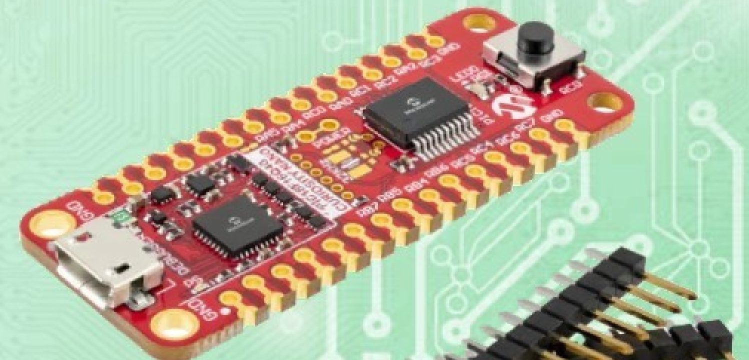
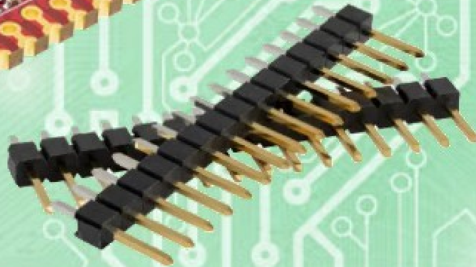
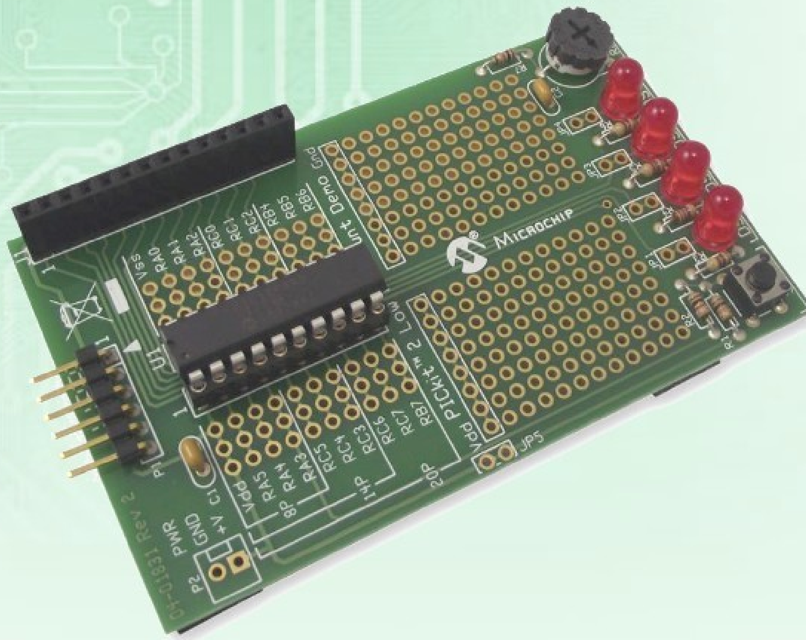


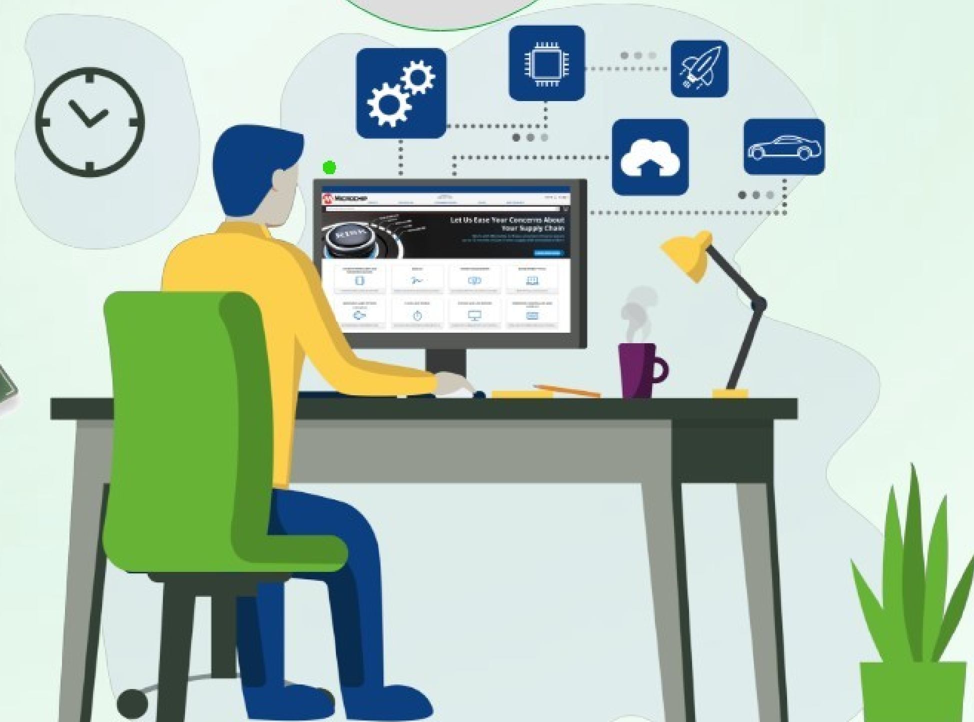


MICROCHIP

PIC mikrovezérlők

2. rész

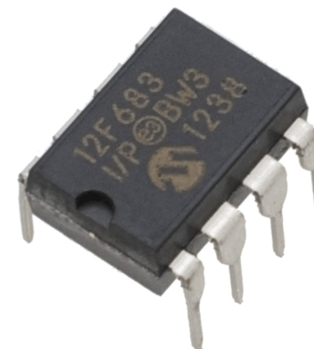




Felhasznált és ajánlott irodalom

- **Milan Verle:** PIC MicrocontrollersProgramming in Assembly
- **Microchip:** PICmicro Mid-Range MCU Family Reference Manual
- **T&T:** Közepes teljesítményű PIC mikrovezérlők Felhasználói Kézikönyv
- **SimulIDE Community:** SimulIDE Tutorials
- **The Jallib Team:**
  - Have fun with PIC microcontrollers, Jal v2 and Jallib
  - Jal v2 Compiler Documentation
  - Installing the JAL Visual Studio Code extension
- **Microsoft:** Visual Studio Code Docs

Adatlapok:



- **PIC12F683** adatlap és termékinfo
- **Microchip:** PICkit2 programmer User's Guide
- **Icircuit Technologies:** iCP02v2 USB PIC/EEPROM programmer manual

Ismerkedés a JAL programnyelvvel

A JAL nyelvről röviden

- A **JAL** (*Just Another Language*) egy magas szintű programnyelv, amelyet arra terveztek, hogy elrejtse a **Microchip PIC** mikrovezérlők programozásának általános kellemetlenségeit

- A **JALLIB** csomaggal telepített **JALv2** fordító **Wouter van Ooijen** eredeti **JAL** fordítójának továbbfejlesztett változata
- A **JAL** némileg hasonlít a Pascal programnyelvre, és nem tesz különbséget a kis- és nagybetűk között
- **Strukturált nyelv:** A **JAL** támogatja a strukturált programozást, amely lehetővé teszi a könnyebb olvashatóságot és karbantarthatóságot (***if-else, for, while,*** stb)
- **Magas szintű nyelv:** A **JAL** könnyen érthető utasításokat tartalmaz, s kiegészítő könyvtárak segítik a hardver absztrakciót
- **Ingyenes és nyílt forráskódú:** A **JAL** ingyenesen elérhető, és a közösség által folyamatosan fejlesztett

Változók típusai

- A változók deklarációja leegyszerűsítve így néz ki:
  `VAR [VOLATILE] [SHARED] type[*cexpr] identifier ['=' cexpr]`
  ahol **VAR** a kulcsszó, **VOLATILE** az illékony változó attribútuma, a **SHARED** változó pedig a bankváltás nélkül elérhető területen van

Type	Description	Range
BIT[1]	1 bit boolean value	0..1
SBIT[1]	1 bit signed value	-1..0
BYTE[1]	8 bit unsigned value	0..255
SBYTE[1]	8 bit signed value	-128..127
WORD	16 bit unsigned value	0..65,535
SWORD	16 bit signed value	-32,768..32,767
DWORD	32 bit unsigned value	0..4,294,967,295
SDWORD	32 bit signed value	-2,147,483,648..2,147,483,647
FLOAT[1]	floating point value	+/-10^-44..10^38

[1]base types A nagyobb típusok az alap típusokból vannak származtatva, pl. WORD = BYTE*2

Tömbök és rekordok

- A **JAL** nyelvben **egydimenziós tömböket** deklarálhatunk
  `VAR type identifier[cexpr]`
  például: `VAR BYTE stuff[5] = {1, 2, 3, 4, 5}`
- Az index 0-tól indul, a fenti példában tehát 0..4 az indextartomány
- A `RECORD` kulcsszóval összetett adatstruktúra típust definiálhatunk

```
RECORD eyeinfo IS
  BYTE left
  BYTE right
END RECORD
VAR eyeinfo eye = { 3, 4 }
eye.left = 1                  -- Így is kaphat értéket
eye.right = 2
```

- Egymásba ágyazott rekordokat is definiálhatunk:

```
RECORD face_r IS
  eyeinfo eyes
  BYTE nose
  BYTE freckels[5]
END RECORD
VAR face_r myface = { { 1,2 }, 3, {4, 5, 6, 7, 8} }
```

Konstansok és álnevek

- A konstansokban tagolásra használhatunk aláhúzás karaktert például: `pragma target CLOCK 8_000_000`
- A **nevesített konstansokat** a változókhoz hasonlóan definiáljuk:
  `CONST type[*cexpr] identifier '=' cexpr`
- A konstansokból is definiálhatunk tömböket, mint például:
  `CONST BYTE str1[] = "Hello world!\r\n"`
- Az **álnevekkel** több azonosító hivatkozhat ugyanarra az objektumra (változóra, nevesített konstansra, függvényre, eljárásra)
- Álnév definiálása: `ALIAS identifier1 IS identifier2`

  Például: `ALIAS led IS pin_A2`
- **Megjegyzés**: ha álnevet definiálunk egy I/O port kivezetéshez, az nem rendeli az új álnévhez az adatirány beállító bitet is, hanem azt külön kell definiálni, például:
  `ALIAS led_direction IS pin_A2_direction`

Operátorok

Operátor	Művelet	Eredmény
COUNT	Tömb elemeinek számát adja meg	UNIVERSAL
WHEREIS	A változó memóriacímét adja meg	UNIVERSAL
DEFINED	Igaz, ha a változó definiálva van	BIT
!	1-es komplemens	Ugyanaz, mint az operandus
!!	Logikai 0, ha 0 követi, egyébként 1	BIT
+, –, *, /	Összeadás, kivonás, szorzás, osztás	promóció
%	Osztási maradék	promóció
<<	Balra léptetés	promóció
>>	Jobbra léptetés	promóció
<	Kisebb	bit
<=	Kisebb, vagy egyenlő	bit
==	Egyenlő	bit
!=	Nem egyenlő	bit
>=	Nagyobb, vagy egyenlő	bit
>	Nagyobb	bit
&	AND (ÉS)	bit
|	OR (VAGY)	bit
^	XOR (kizáró VAGY)	bit

Típuskényszerítés (Casting)

- Típuskényszerítéssel a változó típusát megváltoztathatjuk

```
1   VAR WORD xx
2   VAR BYTE yy
3   ; Az alábbi értékadás miatt figyelmeztetést kapunk,
4   ; mivel csonkítással járhat
5   yy = xx
6   ; Az alábbi értékadás nem generál figyelmeztetést
7   yy = BYTE(xx)
```

- Típuskényszerítés kellhet ahhoz is, hogy egy-egy műveletnél az elvárt promóció történjen (bájtok szorzata nem fér el egy bájtban)

```
1   VAR WORD xx
2   VAR BYTE yy
3   ; ez nem valószű, hogy  a várt eredményt adja
4   xx = yy * yy
5   ;
6   ; ez generálja a jó eredményt
7   xx = WORD(yy) * WORD(yy)
```

Vezérlő struktúrák: BLOCK és CASE

- A **BLOCK** struktúra több utasítást egységbe foglal (mint **C**-ben a {}) a benne deklarált változókat pedig lokálissá teszi

```
BLOCK
  utasítás 1.
  utasítás 2.
END BLOCK
```

- A **CASE** *expr* **OF** struktúra a kifejezés értéke szerint elágaztat

```
CASE xx OF
  1:     yy = 3
  2,5,7: yy = 4            -- Több eset is felsorolható
  10:    BLOCK             -- A CASE ág csak egy utasítás lehet,
           yy = 5          -- ezért össze kell fogni egy BLOCK-ba
           zz = 6
         END BLOCK
  OTHERWISE zz = 0         -- Ez a default ág (opcionális)
END CASE
```

- A **C**-től eltérően a **CASE** struktúrában nincs explicit `break`, minden ág után automatikusan az **END CASE** utáni sorra kerül a vezérlés

Vezérlő struktúrák: FOR és FOREVER

- A **FOR** ciklus egyszerű, mindig nullától indul és egyesével lép

```
FOR expr [ USING var ] LOOP
  statement_block
  [ EXIT LOOP ]
END LOOP
```

- Kilépéskor ***var* = *expr*,** vagy **EXIT** esetén a számláló aktuális értéke. Ha ***expr*** végső értéke eggyel haladja meg a ciklusváltozó számábrázolási határát, akkor kilépéskor nulla lesz az értéke

```
VAR BYTE n
FOR 256 USING n LOOP
 ...
END LOOP                    -- Kilépéskor n értéke 0 lesz
```

- A **FOREVER** végtelen ciklust indít, amelyből csak az **EXIT LOOP** használatával lehet kilépni
- Beágyazott rendszereknél a főprogramnak kell egy végtelen ciklust tartalmaznia, főleg erre használjuk.

Vezérlő struktúrák: WHILE, REPEAT, IF

- A **WHILE** és a **REPEAT** ciklus mindaddig ismétlődik, amíg a kifejezés 1-et ad, vagy ki nem lépünk **EXIT LOOP**-pal

```
WHILE lexpr LOOP
  utasítások
  [ EXIT LOOP ]
END LOOP
```

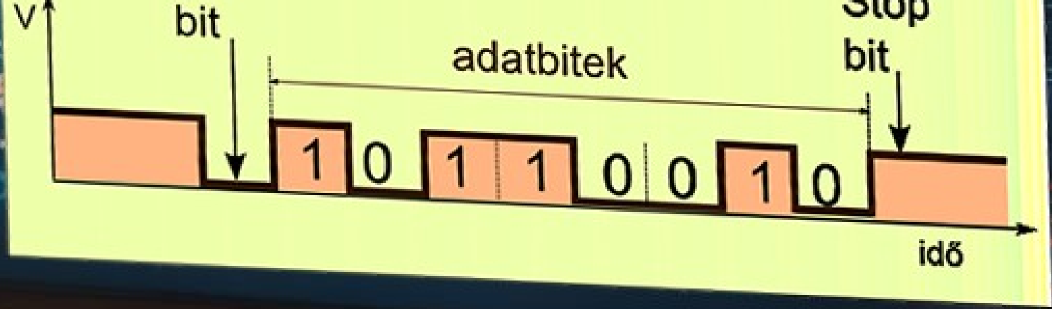
```
REPEAT
  utasítások
  [ EXIT LOOP ]
UNTIL lexpr
```

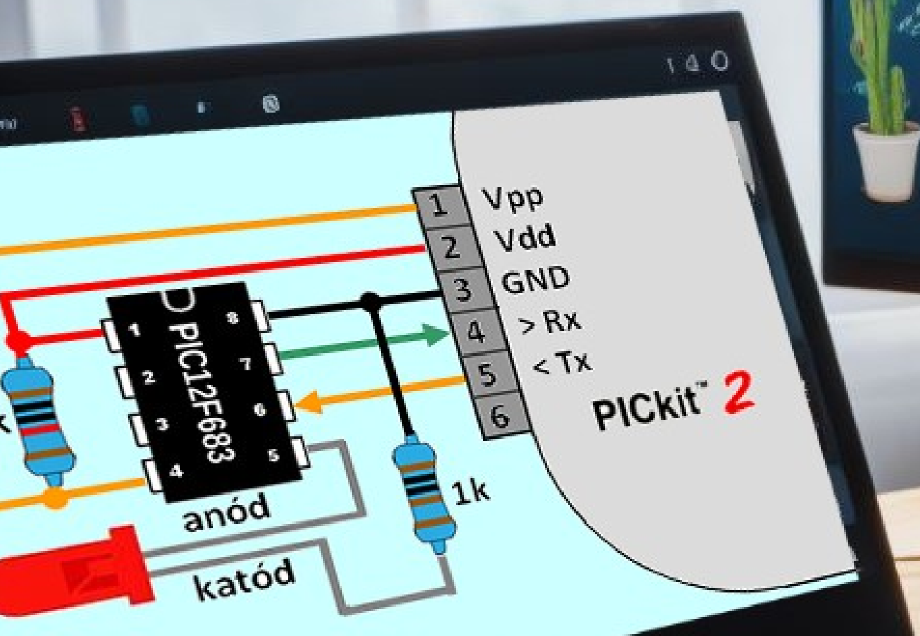
- Az **IF** egy feltételvizsgálatot vagy vizsgálatsorozatot hoz létre. Az első teljesülő feltétel alatti utasításblokk lesz végrehajtva
Ha egyetlen feltétel sem teljesül, és létezik az **ELSE** záradék, akkor annak utasításblokkja kerül végrehajtásra

```
IF lexpr THEN
  statement_block
  [ ELSIF lexpr2 THEN
      statement_block2 ]
  [ ELSE
      statement_block3 ]
END IF
```

példa:

```
IF x == 5 THEN
    y = 7
    ELSIF x == 6 THEN
      y = 12
    ELSE
      y = 0
END IF
```



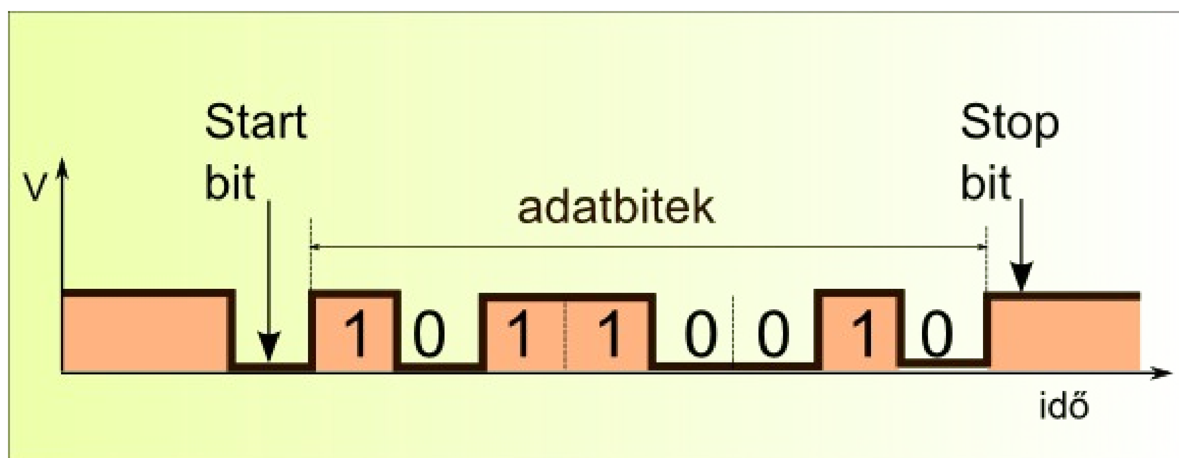


10110010

Aszinkron soros kommunikáció

Aszinkron soros kommunikáció

- A **PIC12F683**-nek nincs soros perifériája, de a GPIO kivezetések szoftveres állítgatásával (bit banging) megoldható a kommunikáció



- **JALLIB** programkönyvtárai támogatják mint a hardveres, mind a szoftveres **UART** kommunikációt:
  - **serial_hardware** – USART periféria kezelés lekérdezéses módban
  - **serial_hardware_int_cts** – USART periféria kezelése megszakításos, bufferelt és adatfolyam-vezérléses módban
  - **serial_software** – szoftveres USART emuláció (bit banging), lekérdezéses módban

JALLIB: serial_software programkönyvtár

A **serial_software** könyvtár használatához az alábbi lépéseket kell követni:

- Definiálni kell az alábbi álneveket:

```
alias serial_sw_tx_pin  is  pin_xy
alias serial_sw_rx_pin  is  pin_xy
```
(bármelyik kimenet)
(bármelyik bemenet)

- Az alábbi konstanst is definiálni kell a felhasználói programban:

```
const serial_sw_baudrate   = 110 .. 240_000
```
(pl. 9600)

- Opcionálisan az alábbi konstansokat is definiálhatjuk a programban:

```
const serial_sw_invert     = FALSE | TRUE
const serial_sw_databits   = 5 .. 8
const serial_sw_stopbits   = 1 | 2
```
(default: TRUE)
(default: 8)
(default: 2)

- Csatoljuk be a könyvtárat!

```
include serial_software
```

- Állítsuk kimenetre a `serial_sw_tx_pin` kivezetést és állítsuk bemenetre a `serial_sw_rx_pin` kivezetést!
- Hívjuk meg a `serial_sw_init()` eljárást!

JALLIB: serial_software programkönyvtár API

- **Eljárások** a **serial_software,jal** programkönyvtárban:
  - **serial_sw_init()** - inicializálja a **Tx** kimenetet
  - **serial_sw_write(*data*)** - egy karakter kiküldése a soros porton
  - **serial_sw_read_wait(*data*)** - egy karakter beolvasása a soros porton, várakozással (ha az **Rx** vonal nem nyugalmi állapotban van)
  - **serial_sw_data'put(*data*)** – a *serial_sw_write()* eljárás segítségével pseudo változót deklarál, hogy pl. így írhassuk: `serial_sw_data =  0x33`
- **Függvények** a **serial_software,jal** programkönyvtárban:
  - **serial_sw_read(*data*)** - egy karakter beolvasása a soros porton a *data* változóba (a visszatérési érték logikai változó, **0**: sikertelen volt a beolvasás, pl. az **Rx** vonal nem volt nyugalmi helyzeteben, **1**: sikeres volt a beolvasás)
  - **serial_sw_data'get()** – a *serial_sw_read_wait()* eljárás segítségével pseudo változót deklarál, hogy pl. így írhassuk: `char = serial_sw_data`

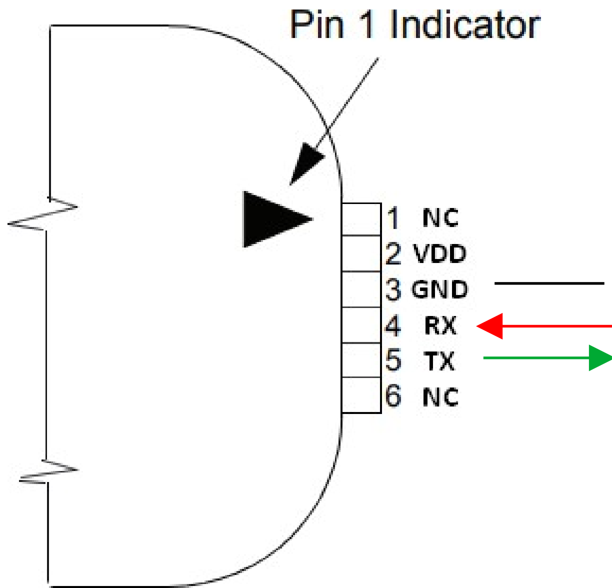
A print programkönyvtár

- Ha egynél több karaktert szeretnénk kiíratni (pl. változók értékét, vagy stringeket), akkor kényelmesebb a **print_xxx()** eljárások valamelyikének használata, leírásuk a **JALLIB API** dokumentációban található (ami a JALLIB csomaggal települő ***doc/html*** mappában található)
- A **print_xxx()** eljárások használatához a `print` könyvtárat be kell csatolni a programba: `include print`
- A **print_xxx()** eljárások első paramétere mindig a kimeneti csatorna, ami esetünkben a `serial_sw_data` lesz
- Példák kiírató eljárásokra:
  - `print_string(serial_sw_data, "Hello world!”)` - szöveg kiírása
  - `print_byte_dec(serial_sw_data, bb)` – bájt kiírása decimális alakban
  - `print_byte_hex(serial_sw_data, bb)` – bájt kiírása hexadecimális alakban
  - `print_byte_bin(serial_sw_data, bb)` – bájt kiírása bináris alakban
- Természetesen a többi változótípusra (WORD, DWORD, előjeles változók) is léteznek hasonló függvények, de ezeket itt nem részletezzük

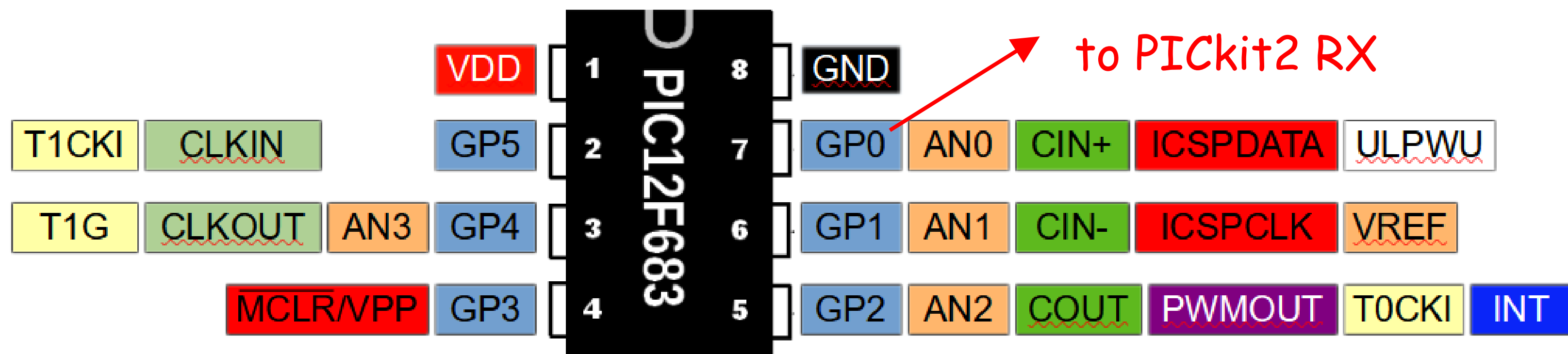
A format programkönyvtár

- A `print_xxx()` eljárásokhoz hasonlóan végezhetünk kiíratásokat a **format** programkönyvtárban definiált eljárásokkal is, azzal a különbséggel, hogy itt két, további paramétert is meg kell adni: a kiíratás szélességét és hogy hány jegyet akarunk tizedesként „levágni” például: `format_dword_dec(serial_sw_data, xx, 6, 3)`
- Például ha az **n_adc** változó egy 8 bites ADC mérés eredménye és az 5 V-os tápfesz. a referencia, akkor **mv = n_adc*5000/256** a mért feszültséget millivoltokban adja meg (legyen pl.
- Legyen pl. `n_adc = 140`, akkor `mv= 140*5000/256 = 2734` akkor `format_dword_dec(serial_sw_data, mv, 6, 3)` hatására ` 2.734` íródik ki, azaz voltokra átszámolva írtuk ki az eredményt
- A fentihez hasonló módon „felszorozva”, azaz kisebb egységekben számolva helyettesíthetjük a törtek hiányát az egészaritmetikában

PICkit2 UART Tool



- Esetünkben az a legcélszerűbb, hogy a programozásnál is használt bekötésnél maradva, a **GPIO0** láb lesz az **UART TX** kimenet, amely a **PICkit2 RX** bemenetére csatlakozik



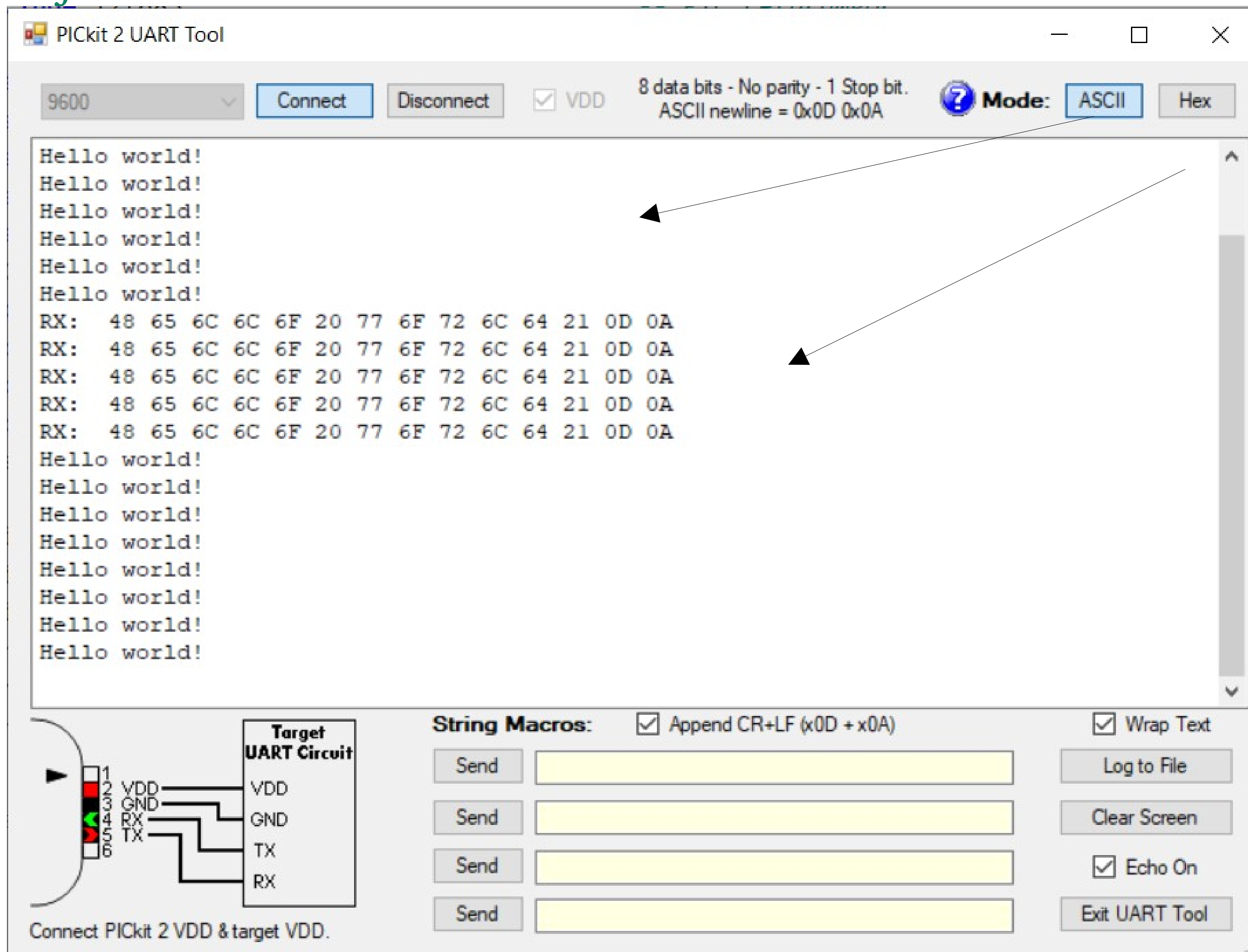
sw_serial_hello.jal

- Öt másodpercenként kiíratjuk a „Hello world!” üzenetet

```
1   include 12f683                                 -- PIC céláramkör
2   pragma target CLOCK      8_000_000             -- oszcillátor frekvencia
3   pragma target OSC        INTOSC_NOCLKOUT       -- belső oszcillátor 8MHz-en
4   pragma target WDT        disabled              -- Watchdog letiltása
5   OSCCON_IRCF   = 0b_111                         -- Fosc = 8 MHz beállítása
6   include delay                                  -- Késleltető függvények
7   include print                                  -- Kiírató függvények
8   enable_digital_io()
9   alias serial_sw_tx_pin   is pin_A0             -- GP0 lesz a TX láb
10  alias serial_sw_rx_pin   is pin_A1             -- GP1 lesz az RX láb
11  const serial_sw_baudrate = 9600                -- bitráta beállítása
12  pin_A0_direction = output                      -- TX az UART kimenet
13  pin_A1_direction = input                       -- RX az UART bemenet
14  include serial_software
15  serial_sw_init()                               -- SW UART inicializálása
16  const byte str1[] = "Hello world!\r\n"         -- string definiálás
17
18  forever loop
19    print_string(serial_sw_data, str1)           -- üzenet kiírása
20    delay_1ms(5000)
21  end loop
```

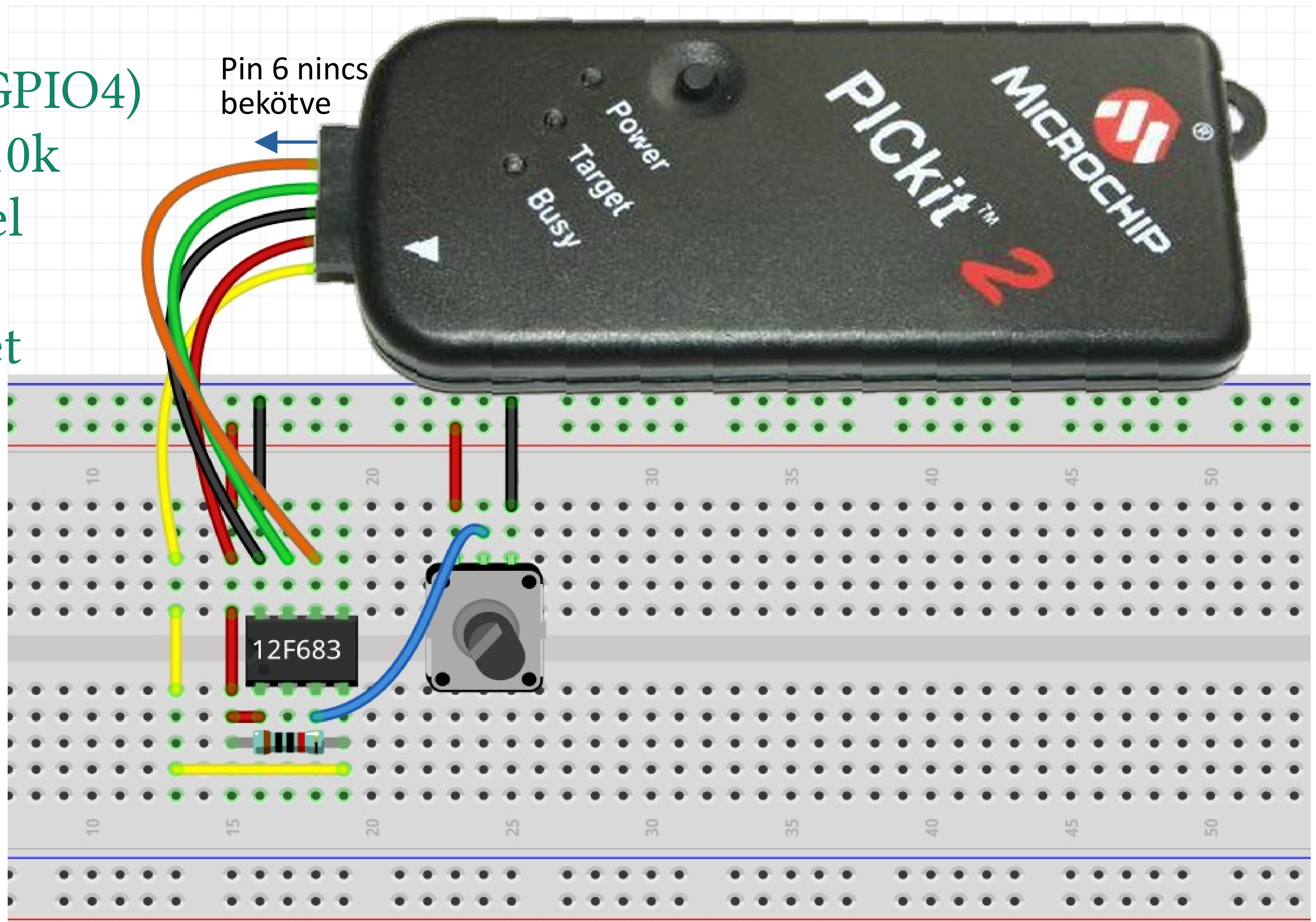
PICkit2 UART Tool

- A beérkező karaktereket ASCII, illetve Hexadecimális formában is kiírathatjuk



Analóg feszültség mérése és kiíratása

- A **PICkit2** programozót **UART Tool**-ként használjuk, a korábbi bekötéssel
- Az **AN3** (GPIO4) lábra egy 10k potméterrel leosztott feszültséget kötünk



sw_serial_ADC.jal – 2/1.

- Az ADC-t 8 bites módba konfiguráljuk és **AN3** lesz a bemenet

```
1   include 12f683                                     -- PIC céláramkör
2   pragma target CLOCK      8_000_000                  -- oszcillátor frekvencia
3   pragma target OSC        INTOSC_NOCLKOUT            -- belső oszcillátor 8MHz-en
4   pragma target WDT        disabled
5   OSCCON_IRCF    = 0b_111                            -- Fosc = 8 MHz beállítása
6   include delay                                      -- Késleltető eljárások
7   include format                                     -- Formázott kiíratás
8   include print                                      -- Kiírató eljárások
9   enable_digital_io()
10
11  -- ADC és AN3 konfigurálás ----------
12  const byte ADC_CHANNEL = 3                         -- A potméter pin_AN3-hoz kötve
13  ANSEL_ANS3 = TRUE                                  -- AN3 analóg móba állítva
14  pin_AN3_direction = input                          -- AN3 bemenetre állítva
15  ADCON0_VCFG = FALSE                                -- VDD és VSS a referencia
16  ANSEL_ADCS = 0b011                                 -- FRC legyen az ADC órajel
17  const ADC_RSOURCE = 10_000                         -- Bemenet: 10K potméter
18  const ADC_HIGH_RESOLUTION = FALSE                  -- 8 bites felbontású ADC mód
19  include adc                                        -- ADC könyvtár becsatolása
20  adc_init()                                         -- ADC inicializálás
21
```

sw_serial_ADC.jal – 2/2.

```
22  -- Soros port konfigurálása ----------
23  alias serial_sw_tx_pin           is pin_A0
24  alias serial_sw_tx_pin_direction is pin_A0_direction
25  serial_sw_tx_pin_direction = output
26  alias serial_sw_rx_pin           is pin_A1
27  alias serial_sw_rx_pin_direction is pin_A1_direction
28  serial_sw_rx_pin_direction = input
29  const serial_sw_baudrate = 9600
30  const serial_sw_invert = true
31  include serial_software
32  serial_sw_init()
33
34  var WORD n_adc
35  var DWORD mv
36  forever loop
37    n_adc = adc_read_low_res(ADC_CHANNEL)
38    print_string(serial_sw_data, "n_adc = ")
39    print_word_hex(serial_sw_data,n_adc)
40    mv = DWORD(n_adc)*5000/256
41    format_dword_dec(serial_sw_data,mv,6,3)
42    print_string(serial_sw_data, " V\r\n")
43    delay_1ms(5000)
44  end loop
```

sw_serial_ADC.jal futtatása

- A potméter tekergetésével változtatjuk a feszültséget…

PICkit 2 UART Tool

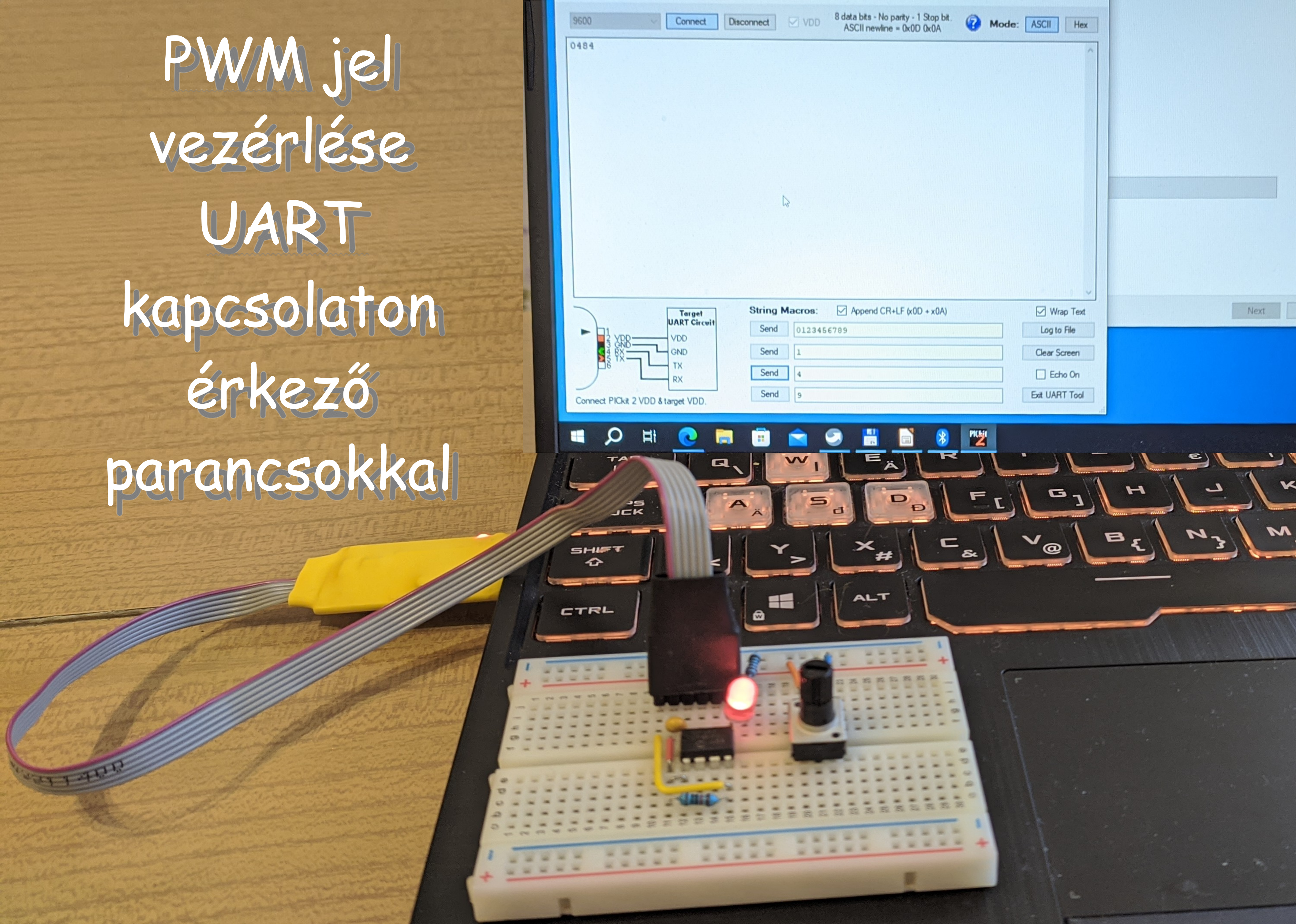
9600 | Connect | Disconnect | VDD | 8 data bits - No parity - 1 Stop bit. ASCII newline = 0x0D 0x0A | Mode: ASCII | Hex

```
n_adc = 0044  1.328 V
n_adc = 0044  1.328 V
n_adc = 00FF  4.980 V
n_adc = 00FF  4.980 V
n_adc = 008D  2.753 V
n_adc = 003C  1.171 V
n_adc = 0035  1.035 V
n_adc = 0034  1.015 V
n_adc = 0032  0.976 V
n_adc = 003C  1.171 V
n_adc = 0035  1.035 V
n_adc = 0000  0.000 V
n_adc = 0018  0.468 V
n_adc = 001C  0.546 V
n_adc = 008D  2.753 V
n_adc = 007D  2.441 V
n_adc = 0080  2.500 V
n_adc = 0080  2.500 V
n_adc = 007F  2.480 V
```

Target UART Circuit
1 2 VDD 3 GND 4 RX 5 TX 6
VDD GND TX RX
Connect PICkit 2 VDD & target VDD.

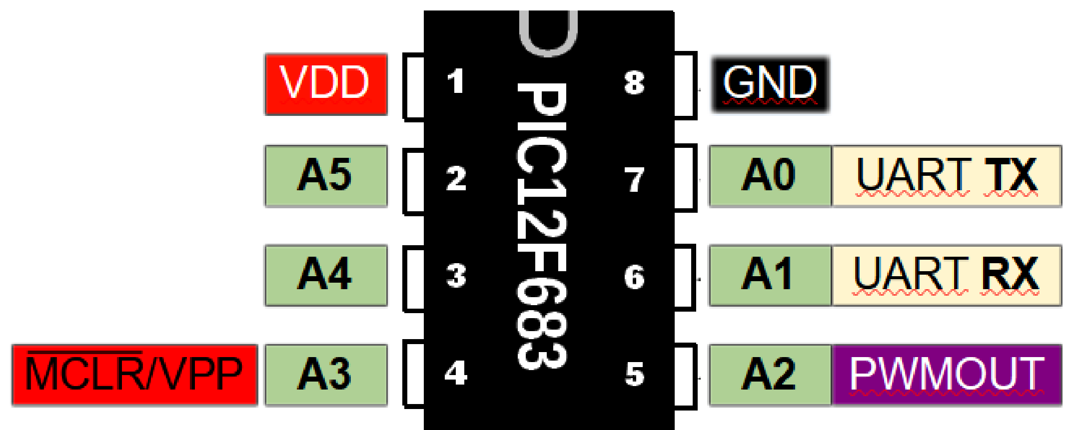
String Macros: Append CR+LF (x0D + x0A) | Wrap Text
Send | Log to File
Send | Clear Screen
Send | Echo On
Send | Exit UART Tool

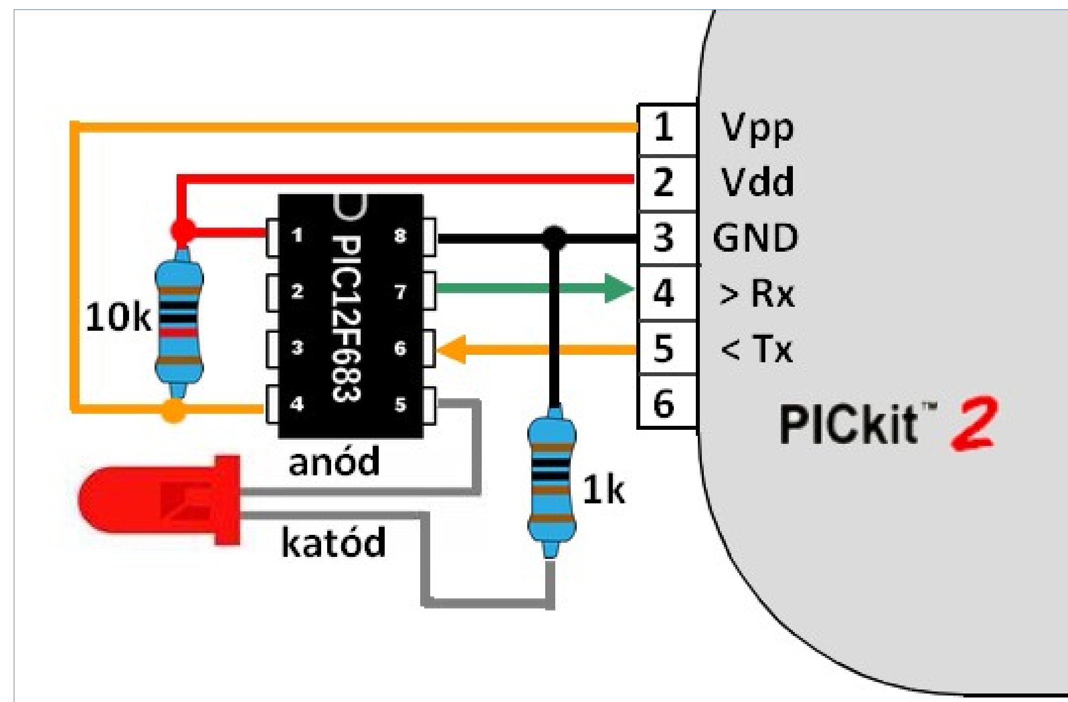
PWM jel vezérlése UART kapcsolaton érkező parancsokkal



Kapcsolási vázlat

- Kétirányú **UART** kapcsolatot alakítunk ki, s a számítógép felől érkező karaktereket fogadjuk, és visszatükrözzük
- A LED fényerejét a **PWM** kimenet vezérli (hardveresen kötött az **A2** kivezetéshez)
- A PWM jel kitöltését a beérkező számmal (0..9) arányosan állítja be a program
- A LED áramát egy 1 kΩ-os ellenállással korlátozzuk
- Az **MCLR** lábat egy 10 kΩ-os ellenállással kötjük **Vdd**-re





sw_serial_pwm.jal – 2/1.

- A soros porton érkező számjegy karakterekkel (0..9) vezéreljük a PWM kitöltését 0 és 252 között

```
1   include 12f683                                    -- PIC céláramkör
2   pragma target CLOCK     8_000_000                 -- oszcillátor frekvencia
3   pragma target OSC       INTOSC_NOCLKOUT           -- belső oszcillátor 8MHz-en
4   pragma target WDT       disabled
5   OSCCON_IRCF   = 0b_111                            -- Fosc = 8 MHz beállítása
6
7   enable_digital_io()                               -- mindegyik GPIO digitális legyen
8
9   -- Soros port konfigurálása ----------
10  alias serial_sw_tx_pin           is pin_A0
11  alias serial_sw_tx_pin_direction is pin_A0_direction
12  serial_sw_tx_pin_direction = output
13  alias serial_sw_rx_pin           is pin_A1
14  alias serial_sw_rx_pin_direction is pin_A1_direction
15  serial_sw_rx_pin_direction = input
16  const serial_sw_baudrate = 9600
17  include serial_software
18  serial_sw_init()
19
```

```
Code area: 319 of 2048 used (words)
Data area: 22 of 128 used
Software stack available: 71 bytes
Hardware stack depth 1 of 8
```

sw_serial_pwm.jal – 2/2.

- A 0..9 számjegyek ASCII kódja 48 és 57 közé esik, így 48 levonásával megkapjuk a számot, ebből számoljuk a kitöltést

```
20  -- PWM konfigurálás -----------
21  include pwm_hardware                              -- a PWM könyvtár becsatolása
22  pwm_max_resolution(16)                            -- Timer2 előosztási arány
23  pin_CCP1_direction = output                       -- a PWM-láb kimenet legyen
24  pwm1_set_dutycycle(0)                             -- PWM kitöltés nulláról indul
25
26  var byte char                                     -- a beolvasott karakter
27  var byte duty                                     -- a kitöltési tényező
28  -- programhurok ------------
29  forever loop
30    if serial_sw_read(char) then
31      serial_sw_write(char)                          -- visszhangozzuk a karaktert
32      if ((char > 47) & (char < 58)) then            -- ha számjegy érkezett
33        duty=(char-48)*28                            -- kitöltés = szám *28
34        pwm1_set_dutycycle(duty)                     -- PWM kitöltés beállítása
35        _usec_delay(2_000)                           -- várunk egy kicsit...
36      end if
37    end if
38  end loop
```

A késleltetés miatt 2-3 karakterküldési ideig „süketek” vagyunk újabb karakter vételére

A késleltetés hatásának szemléltetése

- A String Macros rovatokba előre megírt szöveget helyezhetünk el, amelyek egy gombnyomással kiküldhetők
- A 0123456789 karaktersorozatból csak a megjelöltek érvényesültek