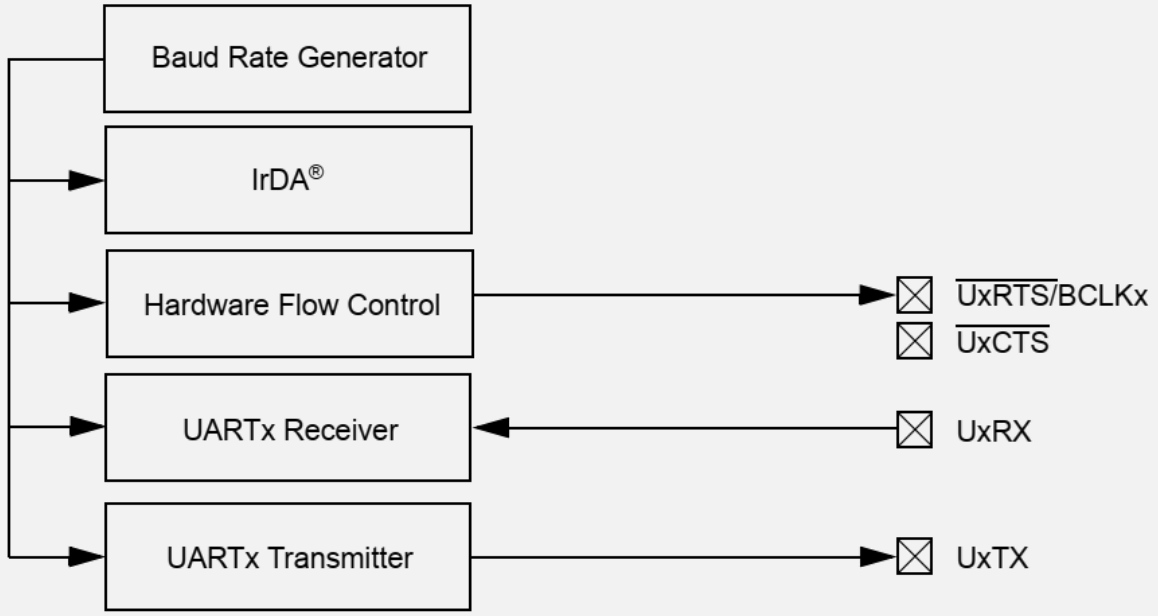
UART modul alkalmazása

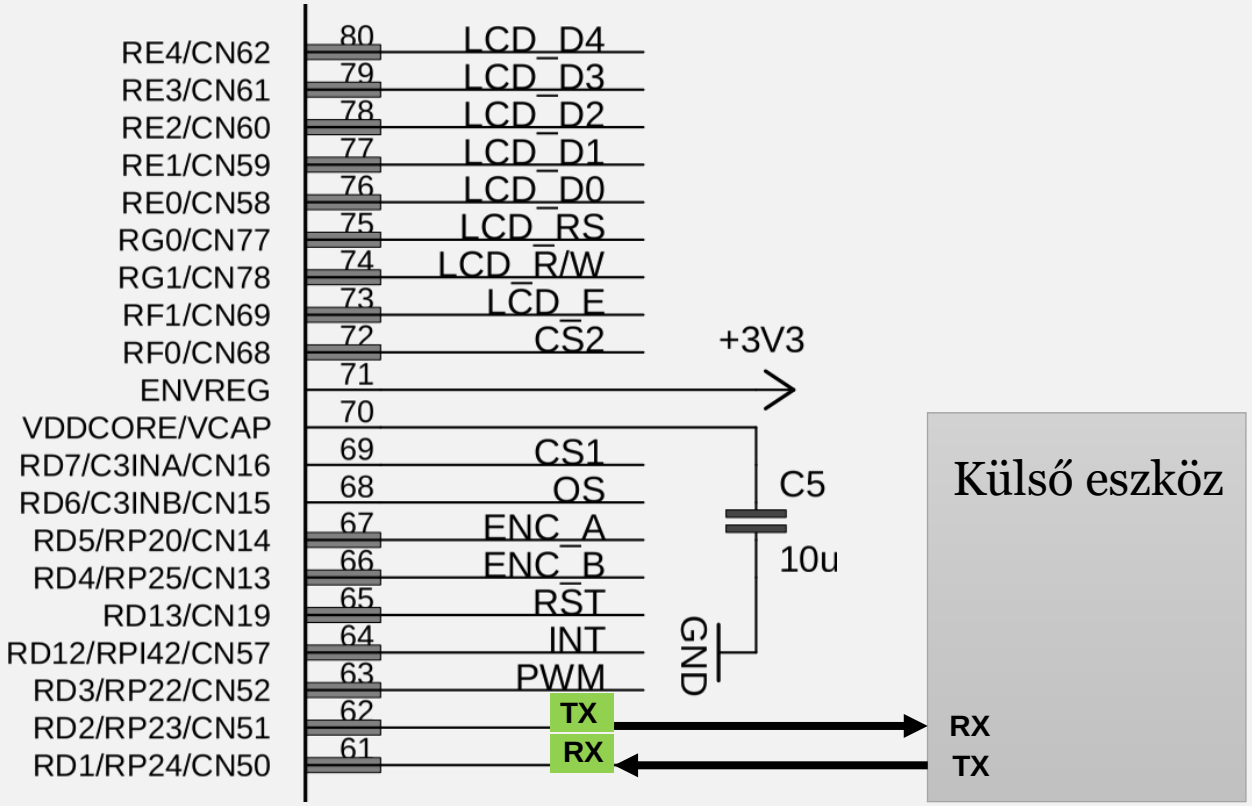
Universal Asynchronous Receiver Transmitter modul

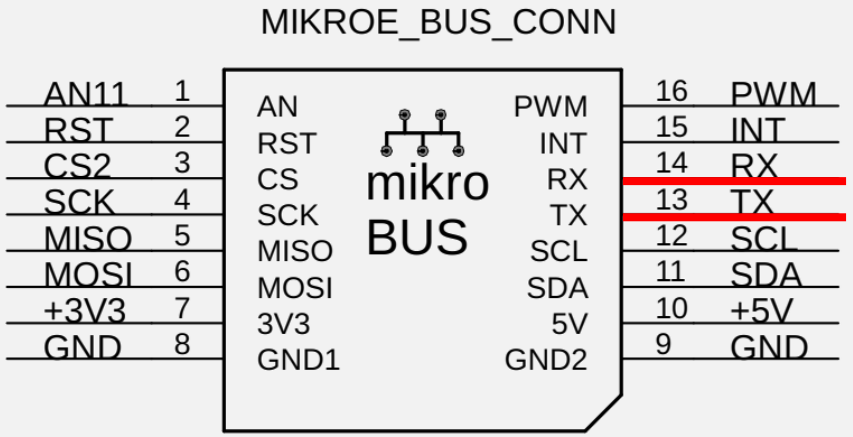
- Négy UART modul található ebben a mikrokontrollerben
- Támogatja az RS-232, RS-485 és a LIN2.0 interfészeket
- Hardware-sen támogatja az átvitelvezérlést
  - A µMOGI2 panelon nincsenek használva a handshake jelvezetékek
- Támogatja az infravörös (IrDA®) adatátvitelt
- 4 mélységű FIFO az adat küldéshez és fogadásához
- Adat küldés és fogadás interruptok
- Különböző interruptok az UART hibák kezelésére
- A jelvezetékeket ki kell vezetni a megfelelő lábra

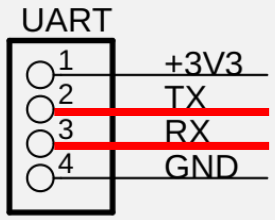


UART modul bekötése

- A µMOGI2 panelon használt lábak: (UART1 modul)
  - U1TX ⇨ RP23
  - U1RX ⇨ RP24







UART modul bekötése

- A µMOGI2 panelon használt lábak: (UART1 modul)
  - U1RX ⇨ RP23
  - U1TX ⇨ RP24
- A lábak felkonfigurálása:

```
// Periferia - lab osszerendeles PPS (pp.135)
// PPSUnLock;
__builtin_write_OSCCONL(OSCCON & 0xbf);
  //UART
  RPOR11bits.RP23R = 3;      // 62-es láb TX
  RPINR18bits.U1RXR = 24;    // 61-es láb RX
// PPSLock
__builtin_write_OSCCONL(OSCCON | 0x40);
```

UxMODE: UARTx Model Register

R/W-0	U-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
UARTEN	—	USIDL	IREN	RTSMD	—	UEN1	UEN0
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8

R/C-0,HC	R/W-0	R/W-0,HC	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WAKE	LPBACK	ABAUD	RXINV	BRGH	PDSEL1	PDSEL0	STSEL
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

UARTEN: UARTx Enable bit
1 = UARTx is enabled; all UARTx pins are controlled by UARTx as defined by UEN<1:0>
0 = UARTx is disabled; all UARTx pins are controlled by PORT latches; UARTx power consumption minimal

USIDL: Stop in Idle Mode bit
1 = Discontinue module operation when device enters Idle mode
0 = Continue module operation in Idle mode

RTSMD: Mode Selection for UxRTS Pin bit
1 = UxRTS pin in Simplex mode
0 = UxRTS pin in Flow Control mode

UEN1:UEN0: UARTx Enable bits
11 = UxTX, UxRX and BCLKx pins are enabled and used; UxCTS pin controlled by port latches
10 = UxTX, UxRX, UxCTS and UxRTS pins are enabled and used
01 = UxTX, UxRX and UxRTS pins are enabled and used; UxCTS pin controlled by port latches
00 = UxTX and UxRX pins are enabled and used; UxCTS and UxRTS/BCLKx pins controlled by port latches

IREN: IrDA® Encoder and Decoder Enable bit
1 = IrDA encoder and decoder enabled
0 = IrDA encoder and decoder disabled

UxMODE: UARTx Model Register

R/W-0	U-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
UARTEN	—	USIDL	IREN	RTSMD	—	UEN1	UEN0
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8

R/C-0,HC	R/W-0	R/W-0,HC	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WAKE	LPBACK	ABAUD	RXINV	BRGH	PDSEL1	PDSEL0	STSEL
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

WAKE: Wake-up on Start Bit Detect During Sleep Mode Enable bit
1 = UARTx will continue to sample the UxRX pin; interrupt generated on falling edge, bit cleared in hardware on following rising edge
0 = No wake-up enabled

LPBACK: UARTx Loopback Mode Select bit
1 = Enable Loopback mode
0 = Loopback mode is disabled

ABAUD: Auto-Baud Enable bit
1 = Enable baud rate measurement on the next character – requires reception of a Sync field (55h); cleared in hardware upon completion
0 = Baud rate measurement disabled or completed

RXINV: Receive Polarity Inversion bit
1 = UxRX Idle state is '0'
0 = UxRX Idle state is '1'

BRGH: High Baud Rate Enable bit
1 = High-Speed mode (baud clock generated from FCY/4)
0 = Standard mode (baud clock generated from FCY/16)

PDSEL<1:0>: Parity and Data Selection bits
11 = 9-bit data, no parity
10 = 8-bit data, odd parity
01 = 8-bit data, even parity
00 = 8-bit data, no parity

STSEL: Stop Bit Selection bit
1 = Two Stop bits
0 = One Stop bit

UxSTA: UARTx Status and Control Register

R/W-0	R/W-0	R/W-0	U-0	R/W-0 HC	R/W-0	R-0	R-1
UTXISEL1	UTXINV	UTXISEL0	—	UTXBRK	UTXEN	UTXBF	TRMT
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8

R/W-0	R/W-0	R/W-0	R-1	R-0	R-0	R/C-0	R-0
URXISEL1	URXISEL0	ADDEN	RIDLE	PERR	FERR	OERR	URXDA
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

UTXISEL<1:0>: Transmission Interrupt Mode Selection bits
11 = Reserved; do not use
10 = Interrupt when a character is transferred to the Transmit Shift Register (TSR), and as a result, the transmit buffer becomes empty
01 = Interrupt when the last character is shifted out of the Transmit Shift Register; all transmit operations are completed
00 = Interrupt when a character is transferred to the Tr

UTXINV: IrDA® Encoder Transmit Polarity Inversion bit
IREN = 0:
1 = UxTX Idle '0'
0 = UxTX Idle '1'
IREN = 1:
1 = UxTX Idle '1'
0 = UxTX Idle '0'

UTXBRK: Transmit Break bit
1 = Send Sync Break on next transmission – Start bit, followed by twelve '0' bits, followed by Stop bit; cleared by hardware upon completion
0 = Sync Break transmission disabled or completed

UTXBF: Transmit Buffer Full Status bit (read-only)
1 = Transmit buffer is full
0 = Transmit buffer is not full, at least one more character can be written

UTXEN: Transmit Enable bit
1 = Transmit enabled, UxTX pin controlled by UARTx
0 = Transmit disabled, any pending transmission is aborted and buffer is reset. UxTX pin controlled by port.

TRMT: Transmit Shift Register Empty bit (read-only)
1 = Transmit Shift Register is empty and transmit buffer is empty (the last transmission has completed)
0 = Transmit Shift Register is not empty, a transmission is in progress or queued

UxSTA: UARTx Status and Control Register

R/W-0	R/W-0	R/W-0	U-0	R/W-0 HC	R/W-0	R-0	R-1
UTXISEL1	UTXINV	UTXISEL0	—	UTXBRK	UTXEN	UTXBF	TRMT
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8

R/W-0	R/W-0	R/W-0	R-1	R-0	R-0	R/C-0	R-0
URXISEL1	URXISEL0	ADDEN	RIDLE	PERR	FERR	OERR	URXDA
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

URXISEL<1:0>: Receive Interrupt Mode Selection bits
11 = Interrupt is set on RSR transfer, making the receive buffer full (i.e., has 4 data characters)
10 = Interrupt is set on RSR transfer, making the receive buffer 3/4 full (i.e., has 3 data characters)
0x = Interrupt is set when any character is received and transferred from the RSR to the receive buffer. Receive buffer has one or more characters.

ADDEN: Address Character Detect bit (bit 8 of received data = 1)
1 = Address Detect mode enabled. If 9-bit mode is not selected, this does not take effect.
0 = Address Detect mode disabled

RIDLE: Receiver Idle bit (read-only)
1 = Receiver is Idle
0 = Receiver is active

PERR: Parity Error Status bit (read-only)
1 = Parity error has been detected for the current character (character at the top of the receive FIFO)
0 = Parity error has not been detected

FERR: Framing Error Status bit (read-only)
1 = Framing error has been detected for the current character (character at the top of the receive FIFO)
0 = Framing error has not been detected

OERR: Receive Buffer Overrun Error Status bit (clear/read-only)
1 = Receive buffer has overflowed
0 = Receive buffer has not overflowed (clearing a previously set OERR bit (1 -> 0 transition) will reset the receiver buffer and the RSR to the empty state

URXDA: Receive Buffer Data Available bit (read-only)
1 = Receive buffer has data, at least one more character can be read
0 = Receive buffer is empty

UxBRG: UARTx Baud Rate Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BGR15	BGR14	BGR13	BGR12	BGR11	BGR10	BGR9	BGR8
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BGR7	BGR6	BGR5	BGR4	BGR3	BGR2	BGR1	BGR0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

BGR<15:0>: Baud Rate Divisior bits

Baud Rate meghatározása:

- ha BRGH = 0

$$\text{Baud Rate} = \frac{FCY}{16 \cdot (UxBRG + 1)}$$

$$UxBRG = \frac{FCY}{16 \cdot \text{Baud Rate}} - 1$$

- ha BRGH = 1

$$\text{Baud Rate} = \frac{FCY}{4 \cdot (UxBRG + 1)}$$

$$UxBRG = \frac{FCY}{4 \cdot \text{Baud Rate}} - 1$$

Universal Asynchronous Receiver Transmitter (UART)

- Baud Rate számítása:
  - FCY = 16 MHz
  - Baud Rate = 115200 bps

```
#define BAUDRATE 115200
#define BRGVAL ((FCY/BAUDRATE)/4) - 1
```

$$\text{Baud Rate} = \frac{FCY}{16 \cdot (UxBRG + 1)}$$

$$UxBRG = \frac{FCY}{16 \cdot \text{Baud Rate}} - 1$$

$$\text{Baud Rate} = \frac{FCY}{4 \cdot (UxBRG + 1)}$$

$$UxBRG = \frac{FCY}{4 \cdot \text{Baud Rate}} - 1$$

$$UxBRG = \frac{16 \cdot 10^6}{16 \times 115200} - 1 = 7{,}68 = 8$$

$$Baud\ Rate = \frac{16 \cdot 10^6}{16 \times (8 + 1)} = 111111\text{bps}$$

$$Error = \frac{111111 - 115200}{115200} = -3{,}5\%$$

$$UxBRG = \frac{16 \cdot 10^6}{4 \times 115200} - 1 = 33{,}7 = 34$$

$$Baud\ Rate = \frac{16 \cdot 10^6}{4 \times (34 + 1)} = 114285\text{bps}$$

$$Error = \frac{114285 - 115200}{115200} = -0{,}8\%$$

Universal Asynchronous Receiver Transmitter (UART)

- Baud Rate számítása:

BRGH = 0 BRGH = 1

BAUD RATE	FCY = 16 MHz			FCY = 16 MHz		
	Actual Baud Rate	% Error	BRG Value (Decimal)	Actual Baud Rate	% Error	BRG Value (Decimal)
110	110.0	0.00	9090	110.0	0.00	36363
300	300.0	0.01	3332	300.0	0.01	13332
1200	1200.5	0.04	832	1200.1	0.01	3332
2400	2398.1	-0.08	416	2399.5	-0.01	1666
9600	9615.4	0.16	103	9592.3	-0.07	416
19.2K	19230.8	0.16	51	19230.7	0.16	207
38.4K	38461.5	0.16	25	38461.5	0.16	103
56K	55555.6	-0.79	17	56338.0	0.60	70
115K	111111.1	-3.38	8	114285.7	-0.62	34
250K	250000.0	0.00	3	250000.0	0.00	15
300K				307692.3	2.50	12
500K	500000.0	0.00	1	500000.0	0.00	7
Min.	15.0	0.00	65535	61.0	0.00	65535
Max.	1000000.0	0.00	0	4000000.0	0.00	0

Egyéb regiszterek

- UxRXREG: UARTx Receive Register

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R-0
-	-	-	-	-	-	-	URX8
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
URX7	URX6	URX5	URX4	URX3	URX2	URX1	URX0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

URX8: Data bit number 8 of the Received Character (in 9-bit mode)

URX<7:0>: Data bits 7-0 of the Received Character

- UxTXREG: UARTx Transmit Register (Write-Only)

U-0	U-0	U-0	U-0	U-0	U-0	U-0	W-x
-	-	-	-	-	-	-	UTX8
bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8

W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x
UTX7	UTX6	UTX5	UTX4	UTX3	UTX2	UTX1	UTX0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

UTX8: Data bit number 8 of the Transmitted Character (in 9-bit mode)

UTX<7:0>: Data bits 7-0 of the Transmitted Character

UART1 inicializálása

```
#define BAUDRATE 115200
#define BRGVAL ((FCY/BAUDRATE)/4) - 1

// serial port (UART1, 8, N, 1, CTS/RTS )
void uartInit()
{
    // UART1
    U1MODE = 0;  // UART1 alapállapotban történő visszaállítása
    U1STA = 0;   // UART1 státuszregiszter nullázása
    U1MODEbits.BRGH = 1;          // Magas Baudrate opció
    U1BRG = BRGVAL;               // Baudrate beállítása

    U1MODEbits.UARTEN = 1;        // UART engedélyezése
    U1STAbits.UTXEN = 1;          // Küldés engedélyezése
}
```

Küldés

```
// byte/karakter küldése
void uartPut(char c)
{
    while(U1STAbits.UTXBF);      // várakozás, amíg a küldő buffer nem lesz üres
    U1TXREG = c;                 // küldés
}

// karakterfüzér küldése
void uartPutStr(const char *str)
{
    while(*str) uartPut(*str++); // karakterfüzér küldése
}
```

```
// Inicializálást követően használható a printf() függvény is
printf("HELLO MOGI\n");
```

Fogadás

```
// érkezett-e új byte/karakter?
#define uartRdyData() U1STAbits.URXDA

// várakozás új byte/karakter kiolvasásáig
char uartRead(){
    while (!uartRdyData());          // várakozás új byte érkezésére
    return U1RXREG;                  // byte kiolvasása
}

// Karaktertömb olvasása, adott hosszig, vagy Enter-ig
void uartReadStr(char *s, int len){
    do{
        *s = uartRead();     // várakozás karakter érkezésére
        //uartPut(*s);       // echo küldése a terminálnak
        if ( *s=='\r')       // \r kihagyása
            continue;
        if ( *s=='\n')       // kilépés a ciklusból
            break;
        s++;
        len--;
    } while ( len > 1 );     // buffer végéig
    *s = '\0';               // \0 végződésű karakterlánc
}
```

UART használata

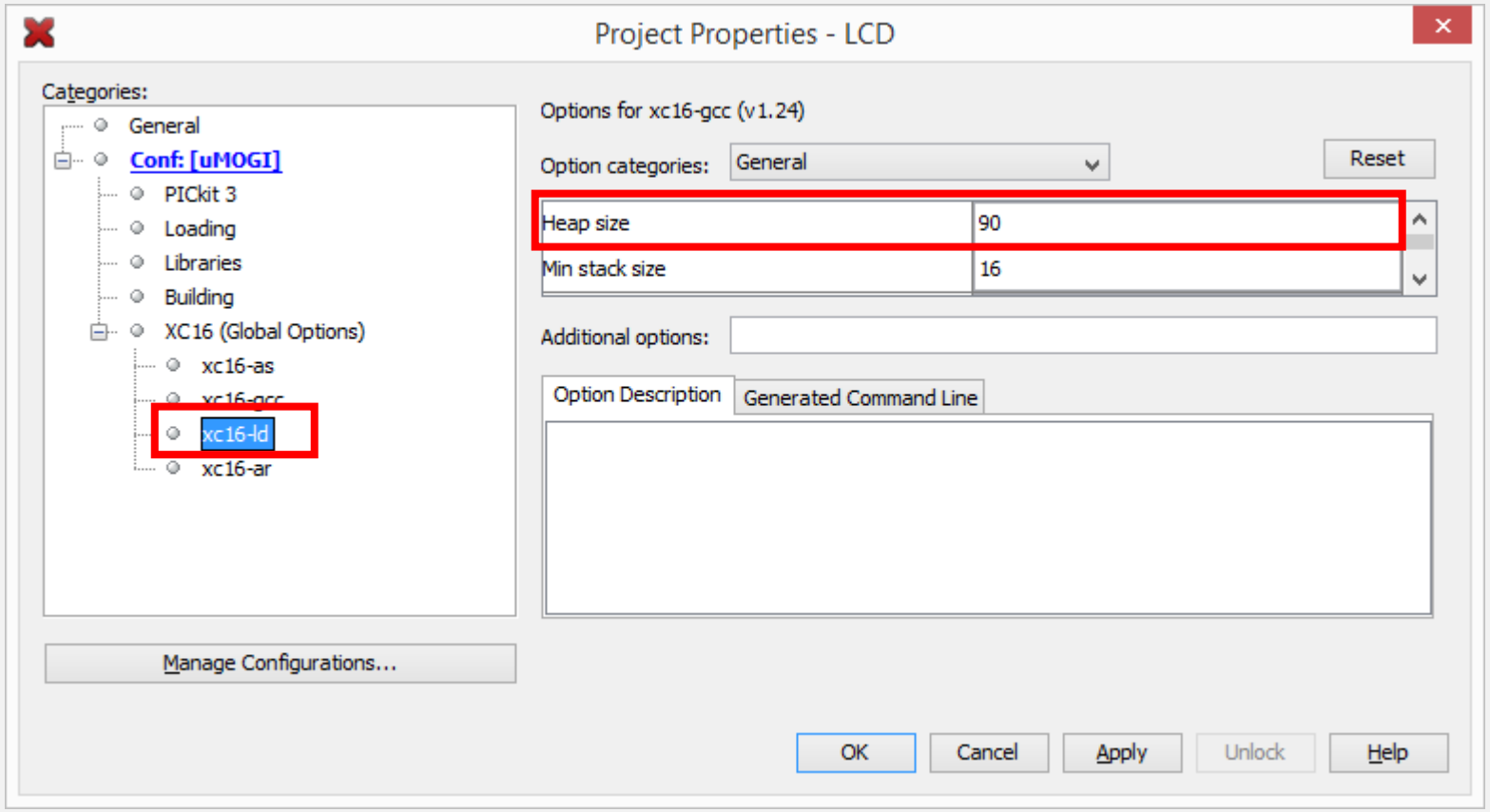
```
// Periferia - lab osszerendeles PPS (pp.135)
__builtin_write_OSCCONL(OSCCON & 0xbf); // PPSUnLock
  //UART
  RPOR11bits.RP23R = 3;      // 62-es láb TX
  RPINR18bits.U1RXR = 24;    // 61-es láb RX
__builtin_write_OSCCONL(OSCCON | 0x40); // PPSLock

uartInit();                       // UART1 inicializálása
uartPutStr("UART Demo\r\n");      // Szöveg kiküldése az UART1-re
char UARTBUFF[80];                // Statikus karaktertömb

int i = 0;
while(1) {
    LEDR = !LEDR;
    DELAY_MS(1000);
    i++;
    sprintf(UARTBUFF,"%i.\r\n",i); // Szám konvertálása
    uartPutStr(UARTBUFF);          // Szöveg kiküldése az UART1-re
}
```

UART használata

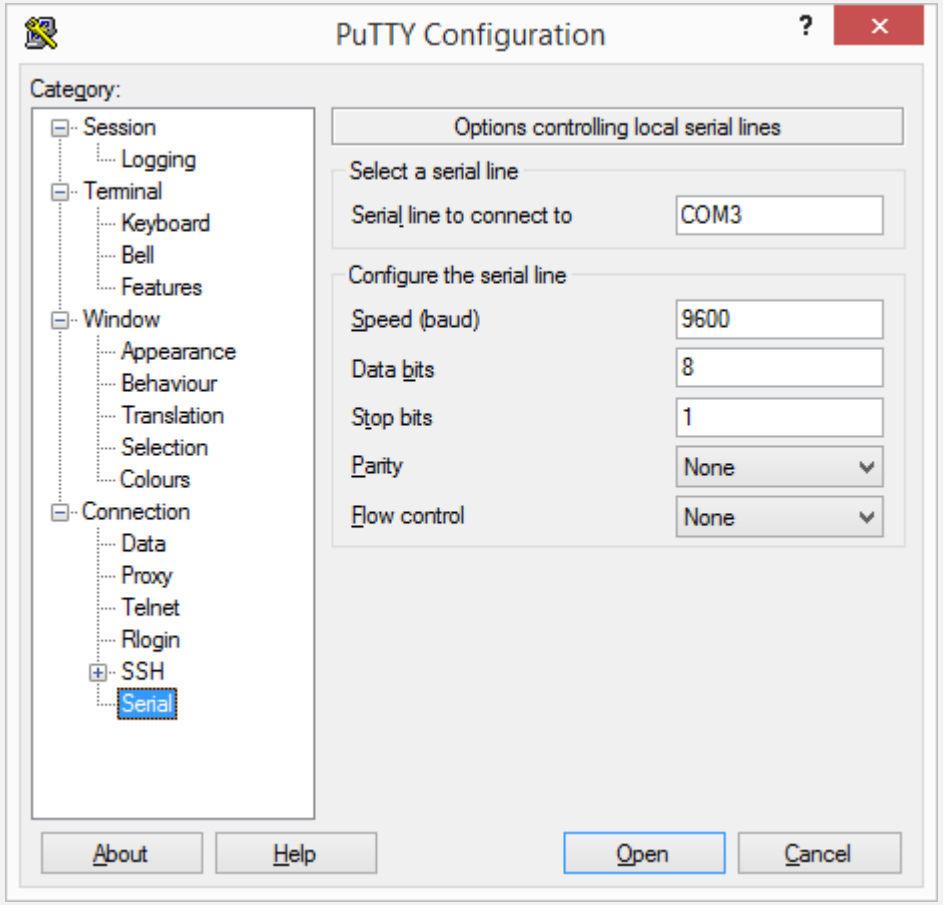
```
// Periferia - lab osszerendeles PPS (pp.135)
__builtin_write_OSCCONL(OSCCON & 0xbf); // PPSUnLock
  //UART
  RPOR11bits.RP23R = 3;      // 62-es láb TX
  RPINR18bits.U1RXR = 24;    // 61-es láb RX
__builtin_write_OSCCONL(OSCCON | 0x40); // PPSLock

uartInit();                       // UART1 inicializálása
uartPutStr("UART Demo\r\n");      // Szöveg kiküldése az UART1-re
char *UARTBUFF = (char *)malloc(80);  // Dinamikus: A linker heap size mérete legalább 90 legyen

int i = 0;
while(1) {
    LEDR = !LEDR;
    DELAY_MS(1000);
    i++;
    sprintf(UARTBUFF,"%i.\r\n",i); // Szám konvertálása
    uartPutStr(UARTBUFF);          // Szöveg kiküldése az UART1-re
}
```

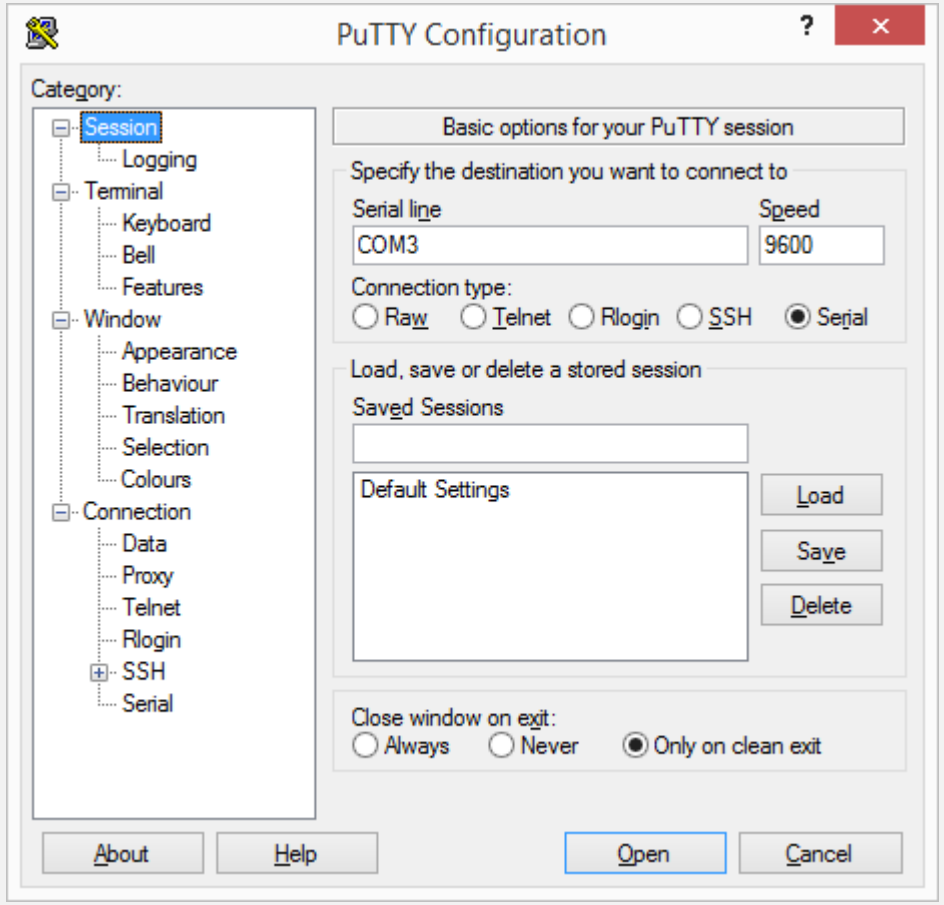
Heap size beállítása a dinamikus memóriafoglaláshoz

- File -> Project Properties



PuTTY beállítása





UART használata

```
// Periferia - lab osszerendeles PPS (pp.135)
__builtin_write_OSCCONL(OSCCON & 0xbf); // PPSUnLock
  //UART
  RPOR11bits.RP23R = 3;      // 62-es láb TX
  RPINR18bits.U1RXR = 24;    // 61-es láb RX
__builtin_write_OSCCONL(OSCCON | 0x40); // PPSLock

uartInit();                       // UART1 inicializálása
printf("UART Demo\n");            // Szöveg kiküldése az UART1-re

int i = 0;
while(1) {
    LEDR = !LEDR;
    DELAY_MS(1000);
    i++;
    printf("%i.\r\n",i);          // Szám konvertálása és kiküldése az UART1-re
}
```

UART használata

```
// Periferia - lab osszerendeles PPS (pp.135)
__builtin_write_OSCCONL(OSCCON & 0xbf); // PPSUnLock
  //UART
  RPOR11bits.RP23R = 3;      // 62-es láb TX
  RPINR18bits.U1RXR = 24;    // 61-es láb RX
__builtin_write_OSCCONL(OSCCON | 0x40); // PPSLock

uartInit();                       // UART1 inicializálása
char str[10];

while(1) {
      uartReadStr(str,10);        // beérkező karakterekre várunk vagy Enterre
      char *s = str;
      while (*s) {
        switch (*s) {
            case 'R': LEDR = 1; break; // R LED világít
            case 'G': LEDG = 1; break; // G LED világít
            case 'B': LEDB = 1; break; // B LED világít
            case '?': uartPutStr("uMOGI2 Panel\r\n"); break;
            default: uartPut('?'); break;
        }
        s++;
    }
}
```

Hibakezelés

- Túlcsordulás (Overrun Error, OERR)
  - Akkor fordul elő, ha a beérkező adatokat nem olvassák ki időben, és új adat érkezik, miközben a fogadó puffer tele van.
  - RX Buffer 4 mélységű
- Kerethiba (Framing Error, FERR)
  - Akkor történik, ha egy karakter érvénytelen stopbittel érkezik meg, ami jelzi, hogy az adatbitek nem voltak megfelelően szinkronizálva.
- Paritáshiba (Parity Error, PERR)
  - Akkor fordul elő, ha a paritásellenőrzés nem egyezik az elküldött és fogadott adatok között.

- Ha nem akarunk lemaradni semmiről, használjunk interruptot ciklikus bufferrel

Hibakezelés

```
void uartErrorHandler()
{
    char dummy;

    if (U1STAbits.OERR) // Túlcsordulás hiba
    {
        U1STAbits.OERR = 0; // Túlcsordulás hiba bit törlése
    }

    if (U1STAbits.FERR) // Kerethiba
    {
        // Kerethiba kezelése (például az adat figyelmen kívül hagyása)
         dummy = U1RXREG; // Hibás adat kiolvasása és eldobása
    }

    if (U1STAbits.PERR) // Paritáshiba
    {
        // Paritáshiba kezelése (például az adat figyelmen kívül hagyása)
         dummy = U1RXREG; // Hibás adat kiolvasása és eldobása
    }
}
```

Interrupt ciklikus bufferrel

```
// Megszakítás beállítása
IFS0bits.U1RXIF = 0;   // UART1 Receive Interrupt Flag törlése
IEC0bits.U1RXIE = 1;   // UART1 Receive Interrupt engedélyezése
```

```
#define BUFFER_SIZE 128 // A buffer méretének meghatározása

volatile char buffer[BUFFER_SIZE]; // Ciklikus buffer
volatile int eleje = 0;            // Eleje mutató a bufferben
volatile int vege = 0;             // Vége mutató a bufferben
```

```
void _ISR _U1RXInterrupt(){
    while (U1STAbits.URXDA)                  // Amíg van elérhető adat a beérkező regiszterben
    {
        buffer[eleje] = U1RXREG;             // Adat beolvasása és bufferbe írása
        eleje = (eleje + 1) % BUFFER_SIZE;   // Eleje pozíció növelése ciklikusan

        // Ütközés kezelése, ha a buffer megtelik
        if (eleje == vege)  vege = (vege + 1) % BUFFER_SIZE; // Legidősebb adat eldobása
    }
    _U1RXIF = 0; // UART1 RX Flag törlése
}
```

Interrupt ciklikus bufferrel, byton-kénti feldolgozás

```
// Következő adat kiolvasása a ciklikus bufferből
int uartReadData(char *data)
{
    if (eleje == vege) return 0;              // Ha a buffer üres, nincs adat

    *data = buffer[vege];                     // Adat kiolvasása
    vege = (vege + 1) % BUFFER_SIZE;          // Vége pozíció növelése ciklikusan
    return 1;                                 // Adat sikeresen kiolvasva
}
```

```
    // főprogramban
    char data;
    while (1)
    {
        if (uartReadData(&data))              // Ha van elérhető adat
        {
            // Továbbítsuk vagy dolgozzuk fel az adatot
            uartPut(data);                    // Például küldjük vissza az adatot
        }
    }
```

Interrupt ciklikus bufferrel, szövegfüzér alapú feldolgozás

```
// Teljes szövegfüzér kiolvasása \n végjelig a ciklikus bufferből
int uartReadLine(char *line, int maxLength) {
    int index = 0;
    int next = vege;                          // Indulunk a végéről
    while (eleje != next) {                   // Ellenőrizzük, hogy van-e elérhető adat a bufferben
        char c = buffer[next];                // Karakter kiolvasása a bufferből
        next = (next + 1) % BUFFER_SIZE;      // Vége pozíció növelése ciklikusan
        if (c == '\r') continue;              // \r kihagyása
        if (c == '\n'){                       // Sorvége karakter észlelése
            line[index] = '\0';               // String végét jelző karakter
            vege = next;                      // Vége pozíció a kiolvasott \n helye lesz
            return 1;                         // Teljes sor sikeresen kiolvasva
        }
        if (index < maxLength - 1)            // Ellenőrizzük, hogy van-e hely a sorban
            line[index++] = c;                // Karakter hozzáadása a sorhoz
        else return 0;                        // Ha elfogyott a hely
    }
    return 0;                                 // Nincs teljes sor a bufferben
}
```

Interrupt ciklikus bufferrel, szövegfüzér alapú feldolgozás

```
#define MAX_SORHOSSZ 128
char sor[MAX_SORHOSSZ];
```

```
    // főprogramban
    while (1)
    {
        if (uartReadLine(sor, MAX_SORHOSSZ))   // Ha van teljes sor
        {
            // Továbbítsuk vagy dolgozzuk fel a kiolvasott sort
            uartPutStr("Received: ");
            uartPutStr(sor);
            uartPutStr("\r\n");                // Windows terminál
        }
    }
```