ÓBUDAI EGYETEM
NEUMANN JÁNOS INFORMATIKAI KAR

# UART

ATMega2560 mikrovezérlő, adat fogadás megszakítással és körbufferrel

```
1  #define F_CPU 16000000UL  // Órajel frekvencia
2  #define BAUD 9600
3  #define UBRR_VALUE F_CPU / 16 / BAUD - 1
4  #define BUFFER_SIZE 50  // Körbuffer mérete
5
6  #include <avr/io.h>
7  #include <avr/interrupt.h>
8
9  // Körbuffer struktúra
10 volatile char ring_buffer[BUFFER_SIZE];
11 volatile uint8_t write_index = 0;
12 volatile uint8_t read_index = 0;
```

**Az Arduino IDE saját Arduino.h fájlt használ, amely a legtöbb szükséges AVR-specifikus függvényt és makrót beépítetten tartalmazza.**
**Ha az Arduino magasabb szintű API-jait használod, akkor az „Arduino.h” automatikus felhasználásra kerül, ezért nem kell manuálisan be include-olni pl. az avr/io.h fájlt.**

```
14 //Miért volatile? a write_index pl. csak megszakításon belül változik,
15 //azaz a processzortól, "kódunktól függetlenül".
16 //Az optimalizáló ezt érzékelheti, és mivel azt látja, hogy mi csak olvassuk,
17 //ezért nem fog minden esetben memória->regiszter mozgatást végezni.
18 //Ezért kimenti egyszer egy regiszterbe és onnan tölti be mindig.
19 //De ha kövzben egy periféria mégis megváltoztatja a változó értékét,
20 //akkor az adat nem fog frissülni, mivel ki van mentve.
21 //A volatil kulcsszó azt jelöli, hogy minden egyes esetben
22 //legyen meg ez a memória -> regiszter kiolvasásási művelet
23 //pl.
24 //int x = foo;
25 //int y = foo;
26 //az optimalizáló csak egyszer fogja a foo értéket beolvasni,
27 //hisz x és y értéke is ugyan az. De mi van, ha közben egy megszakításon
28 //belül megváltozik foo értéke a memóriában,
29 //mielőtt belerakná y-ba is? Ehhez kell a volatile, hogy mindig legyen
30 //meg a memória-> regiszter betöltési művelet
```

```
32  void USART_Init(unsigned int ubrr) {
33    cli();  // Globális megszakítások tiltása
34    // 9600 bps sebesség beállítasa
35    // UBRR alsó 8 bitjének betöltése az UBRRL regiszterbe
36    UBRR0L = ubrr;
37    // UBRR felső 8 bitjének betöltése az UBRRH regiszterbe
38    UBRR0H = (ubrr >> 8);
39    // Aszinkron mód,
40    UCSR0C &= ~((1 << UMSEL01) | (1 << UMSEL00));
41    //8 adat bit
42    UCSR0B &= ~(1 << UCSZ02);
43    UCSR0C |= (1 << UCSZ00) | (1 << UCSZ01);
44    //nincs paritás
45    UCSR0C &= ~((1 << UPM01) | (1 << UPM00));
46    //1 stop bit
47    UCSR0C &= ~(1 << USBS0);
48    //Adó és vevő bekapcsolása
49    UCSR0B |= (1 << RXEN0) | (1 << TXEN0);
50    //megszakítás engedélyezése
51    UCSR0B |= (1 << RXCIE0);
52    sei();
53  }
```

```
55  // Karakter küldése az UART-on
56  void USART_Transmit(char data) {
57    // Vár, amíg az adatregiszter nem lesz üres/használható
58    while (!(UCSR0A & (1 << UDRE0))) {};
59    UDR0 = data;
60  }
61
62  // String küldése az UART-on
63  void USART_SendString(const char *str) {
64    //string = karakter tömb
65    //a string végét mindig egy 0x00 értékű karakter zárja le!
66    //ha a while feltétele = 0, akkor kilép a ciklusból
67    while (*str) {
68      USART_Transmit(*str++);
69    }
70  }
```

```
72  // Körbufferbe karakter beolvasása megszakításon belül
73  ISR(USART0_RX_vect) {
74    char received = UDR0;  // Beérkező karakter
75    //következő index meghatározása
76    //a buffer max indexe = BUFFER_SIZE-1;
77    //ha write_index+1 = BUFFER_SIZE, akkor kezdődik elölről az
78    //indexelés pl.  next_index = 9+1 % 10; //9+1 % 10 = 0;
79    uint8_t next_index = (write_index + 1) % BUFFER_SIZE;
80
81    // Túlcsordulás elkerülése
82    //ha a next_index = a read_index-el akkor tele a buffer
83    if (next_index != read_index) {
84      //ha nincs tele, akkor az adat tárolása a körbufferben
85      ring_buffer[write_index] = received;
86      //write index frissítése
87      write_index = next_index;
88    }
89  }
```

```
91  // Körbufferből egy karakter kiolvasása
92  char ring_buffer_read() {
93    //van-e új adat?
94    if (read_index == write_index) {
95      return 0;  // Ha nincs új adat, visszatér 0-val
96    }
97
98    //új adat kioolvasása
99    char c = ring_buffer[read_index];
100   //read_index frissítése
101   read_index = (read_index + 1) % BUFFER_SIZE;
102   return c;
103 }
```

**Ha ezt használjuk, include-olni kell az elején az „avr/io.h” és „avr/interrupt.h” headert**

```
106  int main() {
107      USART_Init(UBRR_VALUE);  // UART inicializálás
108      USART_SendString("Teszt");
109
110      while (1) {
111          char c = ring_buffer_read();  // Körbufferből karakter beolvasása
112          if (c) {
113              USART_Transmit(c);  // Kiírjuk a kapott adatot az UART-ra
114          }
115      }
116      return 0;
117  }
```

**Az előző diával egyenértékű, azonban a setup() és a loop() az Arduino framework által meghatározott struktúra részei. (ha ezt használjuk, akkor nem kell az elején include-olni az avr/io.h stb. header-eket, mert automatikus bekerül fordításkor az Arduino.h ami már tartalmazza ezt)**

```
105  void setup() {
106    USART_Init(UBRR_VALUE);  // UART inicializálás
107    USART_SendString("Teszt");
108  }
109
110  void loop() {
111    char c = ring_buffer_read();  // Körbufferből karakter beolvasása
112    if (c) {
113      USART_Transmit(c);  // Kiírjuk a kapott adatot az UART-ra
114    }
115  }
```

# Köszönöm a figyelmet!