

Mérés laboratórium 3

µC/OS és API referencia

Referencia

Ebben a fejezetben a mérés során felhasználandó μ C/OS függvények és a panelhoz készített API-k ismertetésére kerül sor. A referencia – praktikus okokból – **nem** törekszik a teljességre. Az érdeklődőbbek a mérés során tanulmányozhatják a μ C/OS forráskódját, és az API-k fejléc fájljait. (Továbbá a jövőben el fog készülni egy különálló ismertető a μ C/OS-ről.)

1.1. μ C/OS függvények

A μ C/OS forráskódja definiál processzor független, egész típusokat. Ezek 8, 16 és 32 bit szélesek, valamint előjelesek és előjel nélküliek lehetnek. Az elnevezés könnyen értelmezhető. Például a 8 bites, előjel nélküli neve **INT8U**, a 32 bites, előjelesé pedig **INT32S**.

1.1.1. Taszkkezelés

TASZK LÉTREHOZÁSA

`INT8U OSTaskCreate(void (*task)(void *pd), void *pdata, OS_STK *ptos, INT8U prio)`

Leírás: ez a függvény illeszti be a μ C/OS ütemezőjének fennhatósága alá a taszkt. Lehetőség van meghívni a multitasking indulása (**OSStart()** hívása) előtt, vagy egy már futó taszkból. ISR-ből viszont nem!

Paraméterek:

task:	mutató a taszk kódját tartalmazó függvényre (mely függvény visszatérési típusa void , egyetlen paramétere pedig void*),
pdata:	mutató egy opcionális adatterületre, amin keresztül paramétereket adhatunk át a taszknak első futásakor,
ptos:	mutató a taszk vermének tetejére (AVR Atmega128 MCU esetén a verem a magas memóriacímektől nő az alacsonyok felé, ebből következően taszk létrehozásakor (amikor még üres a verem), a verem teteje a számára lefoglalt adatterület utolsó bájta),
prio:	a taszknak adandó prioritás. Minden taszknak egyedi prioritással kell rendelkeznie. Az alacsonyabb szám jelöli a magasabb prioritást.

Visszatérési értékek:

OS_NO_ERR:	ha a függvény sikeresen végrehajtódott,
OS_PRIO_EXIST:	ha a megadott prioritási szinten már található taszk,
OS_PRIO_INVALID:	ha a megadott prioritás alacsonyabb (azaz értéke magasabb), mint a legalacsonyabb kiosztható (OS_LOWEST_PRIO).

TASZK TÖRLÉSE

`INT8U OSTaskDel(INT8U prio)`

Leírás: ezzel a függvénnyel lehet taszkt törölni (azaz a **DORMANT** állapotba helyezni). Szükség esetén a taszkt újból ütemezhetővé lehet tenni egy **OSTaskCreate()** hívással. Lehetőség

nyílik a hívó taszk törlésére is (saját prioritásának vagy az `OS_PRIO_SELF` konstans megadásával). ISR kódból **nem** hívható! Továbbá a rendszer üresjárási folyamata **nem** törölhető!

Paraméterek:

prio: a törlendő taszk prioritása, vagy `OS_PRIO_SELF`.

Visszatérési értékek:

OS_NO_ERR: ha sikeresen végrehajtódott a függvény,
OS_TASK_DEL_IDLE: ha a rendszer üresjárási folyamatának törlésére történt kísérlet,
OS_PRIO_INVALID: ha a megadott prioritás alacsonyabb (azaz értéke magasabb), mint a legalacsonyabb kiosztható (`OS_LOWEST_PRIO`), vagy nem egyezik meg `OS_PRIO_SELF` értékével (`0xFF`),
OS_TASK_DEL_ERR: ha a törlésre kijelölt taszk nem létezik,
OS_TASK_DEL_ISR: ha ISR-ből próbáltunk meg taszkot törölni.

1.1.2. Időkezelés

TASZK KÉSLELTETÉSE (ÓRAÜTÉSEKBE KIFEJEZVE)

```
void OSTimeDly(INT16U ticks)
```

Leírás: a taszkot az óraütésekben kifejezett idő elteltéig várakozó állapotba helyezi. Azt, hogy egy óraütés mennyi időnek felel meg az `OS_TICKS_PER_SEC` konstans mondja meg.

Paraméterek:

ticks: az óraütésekben kifejezett késleltetés. Ha 0, a taszk nem kerül várakozó állapotba.

Visszatérési érték:

Nincs.

TASZK KÉSLELTETÉSE (ÓRA, PERC, MÁSODPERC, EZREDMÁSODPERC ÉRTÉKEKBE KIFEJEZVE)

```
INT8U OSTimeDlyHMSM(INT8U hours, INT8U minutes, INT8U seconds, INT16U milli)
```

Leírás: akkor érdemes ezt a függvényt alkalmazni, ha nem óraütésekben, hanem „emberi” mértékegységekben kifejezve szeretnénk a hívó taszkot késleltetni. (Ez a rutin az `OSTimeDly()` függvényt hívja meg kellő számszor a kellő paraméterekkel. Ebből következik, hogy az ezredmásodpercek felbontását az óraütések felbontása határozza meg. Mégpedig úgy, hogy a legközelebbi óraütést fogja választani a függvény.)

Paraméterek:

hours: a késleltetés órákban kifejezve (max. 255),
minutes: a késleltetés percekben kifejezve (max. 59),
seconds: a késleltetés másodpercekben kifejezve (max. 59),
milli: a késleltetés ezredmásodpercekben kifejezve (max. 999).

Visszatérési érték:

OS_NO_ERR: a függvény hívása sikeres volt,
OS_TIME_INVALID_MINUTES: minutes > 59 esetén,
OS_TIME_INVALID_SECONDS: seconds > 59 esetén,
OS_TIME_INVALID_MS: milli > 999 esetén,
OS_TIME_ZERO_DLY: ha minden paraméter 0.

1.1.3. Szemaforok

SZEMAFOR LÉTREHOZÁSA

```
OS_EVENT *OSemCreate(INT16U cnt)
```

Leírás: ezzel a függvénnyel lehet szemafor létrehozni.

Paraméterek:

cnt: a szemafor kezdeti értéke.

Visszatérési érték:

!= NULL ha volt szabad esemény vezérlő blokk (*event control block – ECB*), akkor az arra mutató pointerrel tér vissza a függvény. A későbbiek során (**OSemPend()**, **OSemPost()**) ez a mutató az adott szemaforhoz mint „*handle*” használható,

NULL ha nem volt szabad ECB.

VÁRAKOZÁS SZEMAFORRA

```
void OSemPend(OS_EVENT *pevent, INT16U timeout, INT8U *err)
```

Leírás: ezzel a függvénnyel lehet egy adott szemaforra várakozni. Amíg a szemafor nem szabad (azaz értéke 0), a hívó taszk a várakozó állapotba kerül. Ha szabad volt, a függvény csökkenti értékét eggyel, és visszatér a hívóhoz. Lehetőség van időkorlátot is megadni a várakozáshoz.

Paraméterek:

pevent az adott szemaforral összerendelt ECB-re mutató pointer,

timeout	ha nem 0, akkor maximum az itt megadott óráutésig várakozik a hívó fél foglalt szemafor esetén. Ha 0, akkor addig, amíg a szemafor szabad nem lesz,
err	arra az adatterületre mutat, ahova a függvény a hibaüzeneteit helyezheti. A lehetséges értékek:
OS_NO_ERR	a hívás sikeres volt, és megkaptuk a várt szemafort,
OS_TIMEOUT	az adott időkorlát lejárt, és ez idő alatt nem kaptuk meg a szemafort,
OS_ERR_EVENT_TYPE	ha nem egy szemaforhoz (hanem pl. egy <i>mailbox</i> hoz) tartozó ECB-t adtunk meg a függvény paramétereiként,
OS_ERR_PEND_ISR	ha ISR-ből hívtuk a függvényt (de csak akkor, ha a hívás várakozáshoz vezetne),
OS_ERR_PEVENT_NULL	ha a paraméterül adott pevent mutató NULL pointer.

Visszatérési érték:

Nincs.

SZEMAFOR ELENEDÉSE

INT8U OS_SemPost(OS_EVENT *pevent)

Leírás: ezzel a függvénnyel lehet elengedni a szemafort.

Paraméterek:

pevent mutató az adott szemaforral összerendelt ECB-re.

Visszatérési érték:

OS_NO_ERR	a hívás sikeres volt, a szemafor elengedésre került,
OS_SEM_OVF	a szemafor többször került elengedésre, mint ahányszor várakoztak rá,
OS_ERR_EVENT_TYPE	ha nem egy szemaforhoz (hanem pl. egy <i>mailbox</i> hoz) tartozó ECB-t adtunk meg a függvény paramétereiként,
OS_ERR_PEVENT_NULL	ha a paraméterül adott pevent mutató NULL pointer.

1.2. A mérőpanelhez készített API-k ismertetése

A programozás megkönnyítése végett készült LCD, soros port és A/D konverter kezelő API a mérőpanelhez. Az API-k fejléc fájljai az alapértelmezett *include* könyvtár „board” alkönyvtára alatt találhatóak. A lefordított kódjuk pedig az alapértelmezett *library* könyvtárban a „libboard.a” fájlban.

Az AVR standard C könyvtár definiál processzor független, egész típusokat (<stdint.h>). Ezek 8, 16, 32 és 64 bit szélesek, valamint előjelesek és előjel nélküliek lehetnek. Az elnevezés könnyen értelmezhető. Például a 8 bites, előjel nélküli neve **uint8_t**, a 64 bites, előjelesé pedig **int64_t**. (A µC/OS forráskódja is definiál hasonló célú típusokat. Azért nem azokat használják az API-k, mert így µC/OS nélkül is használhatóak.)

1.2.1. LCD kezelés

A panelen található egy 4x20 karakteres, LED háttérvilágítással ellátott LCD panel. Az LCD API ennek kezelését hivatott megkönnyíteni. Használata előtt a következő direktívával hivatkozhatunk rá: **#include <board/lcd.h>**. **Fontos:** a helyes működés érdekében engedélyezni kell a fordítói optimalizációt. Ez legalább az "-O2" kapcsolót jelenti. AVR Studio 4.13 (b528) alatt: *Projects / Configuration Options / General / Optimization: -O2*.

Ha az LCD panelre szeretnénk irányítani a standard kimenetet, akkor azt a következő utasítással tehetjük meg: **stdout = &LCD_stdout;**. (Ha nem kívánjuk az LCD-re irányítani a standard kimenetet, de mégis a C standard kiíró függvényeit szeretnénk használni, akkor azt megtehetjük például így: **fprintf(&LCD_stdout, <format string>);**.)

A speciális karakterek implementálása:

- \n:** kocsi vissza + soremelés, szükség esetén görgetés.
- \r:** kocsi vissza.
- \t:** vízszintes tabulátor. A következő 5-tel osztható pozícióba visz. (0, 5, 10, 15). Ha kell, a következő sorban folytatódik. Szükség esetén görgetés.
- \v:** függőleges tabulátor. Páros sorból a másik páros sorba, páratlan sorból a másik páratlan sorba ugrik. Nincs görgetés!
- \a:** "Alarm". Megvillogtatja a kijelzőt.

AZ LCD PANEL INICIALIZÁLÁSA

```
void LCD_init()
```

Leírás: az LCD panel inicializálása, a kurzor elrejtése és a háttérvilágítás maximum értéken való bekapcsolása. **A többi LCD kezelő függvény használata előtt meg kell hívni!**

Paraméterek: nincs.

Visszatérési érték: nincs.

A HÁTTÉRVILÁGÍTÁS BEKAPCSOLÁSA (TELJES FÉNYERŐN)

```
void LCD_light_on();
```

Leírás: bekapcsolja az LCD háttérvilágítását maximális fényerőre állítva.

Paraméterek: nincs.

Visszatérési érték: nincs.

A HÁTTÉRVILÁGÍTÁS KIKAPCSOLÁSA

```
void LCD_light_off();
```

Leírás: kikapcsolja az LCD háttérvilágítását.

Paraméterek: nincs.

Visszatérési érték: nincs.

A HÁTTÉRVILÁGÍTÁS BEKAPCSOLÁSA (ADOTT FÉNYERŐRE)

```
void LCD_light(uint8_t intensity);
```

Leírás: az LCD panel fényerejét a 256 fokozat egyikére állítja.

Paraméterek:

intensity a kívánt fényerő: 0 (kikapcsolva) ... 255 (teljes fényerő).

Visszatérési érték: nincs.

1.2.2. A soros port kezelése

A panelen található egy RS232 csatlakozó (**figyelem:** ez nem azonos a programletöltő soros porti csatlakozójával). A soros API a panelen található soros port kezelését hivatott megkönnyíteni. Használata előtt a következő direktívával hivatkozhatunk rá: **#include <board/serial.h>**.

Figyelem: az API jelen pillanatban nem megszakításos elven működik! Ebből következően karakter küldésekor addig, amíg az esetleges előző karakter elküldésre nem kerül, valamint karakter fogadásakor addig, amíg az be nem érkezik, **blokkol!**

Ha a soros portra szeretnénk irányítani a standard be- vagy kimenetet, akkor azt a következő utasítással tehetjük meg: **stdin = &serial_stdin;** ill. **stdout = &serial_stdout;**. (Ha nem kívánjuk a soros interfészre irányítani a standard be- vagy kimenetet, de mégis a C standard I/O függvényeit szeretnénk használni, akkor azt megtehetjük például így: **fprintf(&serial_stdout, <format string>);** vagy **getc(&serial_stdin);**.)

A speciális karakterek implementálása: minden karakter változtatás nélkül elküldésre kerül, kivéve az „új sor” karaktert (**\n**), itt ugyanis egy „kocsi vissza” (**\r**) is beszűrődik.

A SOROS INTERFÉSZ INICIALIZÁLÁSA

```
void serial_init()
```

Leírás: a soros vonalat 9600 baud jelváltási sebességre inicializálja 8 adatbit és 1 stop bit mellett (paritás bit nincs).

Paraméterek: nincs.

Visszatérési érték: nincs.

ADAT KÜLDÉSE

```
void serial_transmit(uint8_t data)
```

Leírás: egy bájt elküldése a soros vonalon. (A függvény blokkol, amíg a kiviteli puffer ki nem ürül!)

Paraméterek:

data az elküldendő 8 bites adat.

Visszatérési érték: nincs.

ADAT FOGADÁSA

```
uint8_t serial_receive()
```

Leírás: egy bájt fogadása a soros vonalon. (A függvény blokkol, amíg nem érkezik adat!)

Paraméterek: nincs.

Visszatérési érték: a fogadott bájt.

1.2.3. Az A/D konverter kezelése

Az ATmega128 mikrovezérlőben található egy A/D átalakító egység. Ennek több bemenete is van. Közülük négyre a mérőpanelon található BNC bemenet, NTK hőmérsékletfüggő ellenállás, fényellenállás és potenciométer csatlakozik. Az API megszakításosan kezeli a konvertert. A digitalizált értékek 10 bit felbontásúak. Továbbá az átalakító kétféle működési módban használható: az egyikben minden egyes konvertálás előtt el kell indítani (*single conversion*), a másikban csak az első előtt kell, a többi automatikusan megtörténik az előző befejezte után (*free running*). (Ez utóbbihoz még annyi információt érdemes tudni, hogy az API az ADC órajelének előállítására a legmagasabb (128) előosztót alkalmazza. Ez a 11,0592 MHz CPU órajel esetén 86,4 kHz-et jelent. És az átalakítás 13 periódusnyi ADC órajelet vesz igénybe.)

AZ A/D ÁTALAKÍTÓ INICIALIZÁLÁSA

```
void ADC_init(uint8_t channel, uint8_t mode)
```

Leírás: inicializálja a konvertert, és beállítja a kívánt bemeneti csatornát valamint működési módot.

Paraméterek:

channel a kiválasztandó bemeneti csatorna, értéke az alábbi lehet:

ADC_AIN	az analóg BNC bemenet kiválasztása,
ADC_NTK	a negatív hőmérséklet együtthatójú ellenállás kiválasztása,
ADC_OPTO	a fényellenállás kiválasztása,
ADC_POT	a potenciométer kiválasztása.

mode a beállítandó működési mód, lehetséges értékei:

ADC_SINGLE	<i>single conversion</i> üzemmód,
ADC_RUNNING	<i>free running</i> üzemmód.

Visszatérési érték: nincs.

KONVERTÁLÁS INDÍTÁSA

```
void ADC_start()
```

Leírás: az átalakítás indítása.

Paraméterek: nincs.

Visszatérési érték: nincs.

A KONVERTÁLÁS EREDMÉNYÉNEK BEOLVASÁSA

```
uint16_t ADC_read()
```

Leírás: a konvertálás eredményének kiolvasása. Meghívni az A/D átalakítóhoz tartozó interruptból (ADC_vect) célszerű!

Paraméterek: nincs.

Visszatérési érték: a beolvasott 10 bites érték.

CSATORNA VÁLTÁS

```
void ADC_set_channel(uint8_t channel)
```

Leírás: bemeneti csatorna váltása. (Ha épp átalakítás közben hívjuk meg ezt a függvényt, hatása a következő konverziótól érvényesül.)

Paraméterek:

channel lásd: ADC_init().

Visszatérési érték: nincs.

ÜZEMMÓD VÁLTÁS

```
void ADC_set_mode(uint8_t mode)
```

Leírás: az átalakító üzemmódjának váltása.

Paraméterek:

mode lásd: ADC_init().

Visszatérési érték: nincs.

1.3. ISR vektor nevek és a hozzájuk tartozó megszakítások

Vektor név	Megszakítás	Vektor név	Megszakítás
INT0_vect	External Interrupt 0	TIMER3_CAPT_vect	Timer3 Capture Event
INT1_vect	External Interrupt 1	TIMER3_COMPA_vect	Timer3 Compare Match A
INT2_vect	External Interrupt 2	TIMER3_COMPB_vect	Timer3 Compare Match B
INT3_vect	External Interrupt 3	TIMER3_COMPC_vect	Timer3 Compare Match C
INT4_vect	External Interrupt 4	TIMER3_OVF_vect	Timer3 Overflow
INT5_vect	External Interrupt 5	ADC_vect	ADC Conversion Complete
INT6_vect	External Interrupt 6	ANALOG_COMP_vect	Analog Comparator
INT7_vect	External Interrupt 7	USART0_RX_vect	USART0 RX Complete
TIMER0_COMP_vect	Timer0 Compare Match	USART0_UDRE_vect	USART0 Data Register Empty
TIMER0_OVF_vect	Timer0 Overflow	USART0_TX_vect	USART0 TX Complete
TIMER1_CAPT_vect	Timer1 Capture Event	USART1_RX_vect	USART1 RX Complete
TIMER1_COMPA_vect	Timer1 Compare Match A	USART1_UDRE_vect	USART1 Data Register Empty
TIMER1_COMPB_vect	Timer1 Compare Match B	USART1_TX_vect	USART1 TX Complete
TIMER1_COMPC_vect	Timer1 Compare Match C	SPI_STC_vect	SPI Transfer Complete
TIMER1_OVF_vect	Timer1 Overflow	TWI_vect	Two-wire Serial Interface
TIMER2_COMP_vect	Timer2 Compare Match	EE_READY_vect	EEPROM Ready
TIMER2_OVF_vect	Timer2 Overflow	SPM_READY_vect	Store Program Memory Ready

1. táblázat: AVR GCC alatt használható megszakítás vektor nevek az ATmega128 mikrovezérlőhöz