Mérés 3 - Ellenörzö mérés - 3. mérés : uC/OS

3 - 1.) LCD háttérvilágítás szabályzás. (Sw időzítés+Soros port)
Írjon 3 taszkot az alábbiak szerint:

- TaskA: Globális változóból alapján beállítja az LCD háttérvilágítás értékét, és kiírja a nevét (TaskA) az LCD-re.

- TaskB: 2 másodpercenként kb. 10%-al növeli a háttérvilágítás intenzitás változóját (Azaz 10 lépésben éri el a maximális intenzitást). A maximális érték elérése után kialszik az LCD, majd újra kezdődik az intenzitás növelés. Minden intenzitás változtatáskor a task kiírja a nevét.

- TaskC: A soros portról karaktereket olvas. Ha az ’w’, vagy ’x’, ezt kiírja az LCD- re, és 10%-al ’w’-re növeli, ’x’-re csökkenti a háttérvilágítás intenzitás változóját.

Megoldás:

```
#include "includes.h"

#define TASK1_STK_SIZE  128

#define TASK1_PRIO  15
#define TASK2_PRIO  16
#define TASK3_PRIO  19
#define TASK4_PRIO  17

OS_STK  Task1Stk[TASK1_STK_SIZE];
OS_STK  Task2Stk[TASK1_STK_SIZE];
OS_STK  Task3Stk[TASK1_STK_SIZE];
OS_STK          Task4Stk[TASK1_STK_SIZE];

void Task1(void *data);
void Task2(void *data);
void Task3(void *data);
void Task4(void *data);

unsigned int intensity = 5;

int main(void)
{
   OSInit();
   OSTaskCreate(Task1, NULL, &Task1Stk[TASK1_STK_SIZE - 1], TASK1_PRIO);
   OSStart();
   return 0;
}



void Task1(void* data) {
   OS_ENTER_CRITICAL();
      TCCR0=0x07;
      TIMSK=_BV(TOIE0);
      TCNT0=256-(CPU_CLOCK_HZ/OS_TICKS_PER_SEC/1024);
   OS_EXIT_CRITICAL();
```

```
    LCD_init();
    serial_init();
    LCD_light_on();
    stdin = &serial_stdin;
    stdout = &LCD_stdout;
    OSTaskCreate(Task2, NULL, &Task2Stk[TASK1_STK_SIZE - 1], TASK2_PRIO);
    OSTaskCreate(Task3, NULL, &Task3Stk[TASK1_STK_SIZE - 1], TASK3_PRIO);
    OSTaskCreate(Task4, NULL, &Task4stk[TASK1_STK_SIZE - 1], TASK4_PRIO);

    OSTaskDel(TASK1_PRIO);

}

void Task2(void* data) {
    while(1){
            LCD_light(intensity);
            putchar('a');
    }
}

void Task3(void* data) {
    while(1) {
        OSTimeDlyHMSM(0,0,2,0);
        if(intensity == 255) intensity = 5;
            else intensity+=25;
            putchar('b');
    }
}

void Task4(void* data) {
    while(1) {
            char temp = getchar();
            if(temp == 'x' && intensity != 5) intensity-=25;
            if(temp == 'w' && intensirt != 255) intensity+=25;
    }
}
```

3 - 2.) LCD háttérvilágítás szabályzás. (Sw időzítés+Gombok)
Írjon 3 taszkot az alábbiak szerint:
- TaskA: Globális változóból alapján beállítja az LCD háttérvilágítás értékét, és kiírja a nevét (TaskA) az LCD-re.

- TaskB: 2 másodpercenként kb. 10%-al növeli a háttérvilágítás intenzitás változóját (Azaz 10 lépésben éri el a maximális intenzitást). A maximális érték elérése után kialszik az LCD, majd újra kezdődik az intenzitás növelés. Minden intenzitás változtatáskor a task kiírja a nevét.

- TaskC: Két gombot figyel. BTN0-ra növeli, BTN1-re csökkenti a háttérvilágítás intenzitás változóját. Oldja meg a pergésmentesítést is!

Megoldás:

```
#include "includes.h"

#define TASK1_STK_SIZE  128

#define TASK1_PRIO  15
#define TASK2_PRIO  16
#define TASK3_PRIO  19
#define TASK4_PRIO  17

OS_STK  Task1Stk[TASK1_STK_SIZE];
OS_STK  Task2Stk[TASK1_STK_SIZE];
OS_STK  Task3Stk[TASK1_STK_SIZE];
OS_STK          Task4Stk[TASK1_STK_SIZE];

void Task1(void *data);
void Task2(void *data);
void Task3(void *data);
void Task4(void *data);

unsigned int intensity = 5;

int main(void)
{
   OSInit();
   OSTaskCreate(Task1, NULL, &Task1Stk[TASK1_STK_SIZE - 1], TASK1_PRIO);
   OSStart();
   return 0;
}

void Task1(void* data) {
   OS_ENTER_CRITICAL();
      TCCR0=0x07;
      TIMSK=_BV(TOIE0);
      TCNT0=256-(CPU_CLOCK_HZ/OS_TICKS_PER_SEC/1024);
   OS_EXIT_CRITICAL();

   LCD_init();
   LCD_light_on();
   stdout = &LCD_stdout;
   OSTaskCreate(Task2, NULL, &Task2Stk[TASK1_STK_SIZE - 1], TASK2_PRIO);
   OSTaskCreate(Task3, NULL, &Task3Stk[TASK1_STK_SIZE - 1], TASK3_PRIO);
   OSTaskCreate(Task4, NULL, &Task4stk[TASK1_STK_SIZE - 1], TASK4_PRIO);
```

```
    OSTaskDel(TASK1_PRIO);

}

void Task2(void* data) {
    while(1) {
            LCD_light(intensity);
            putchar('a');
    }
}

void Task3(void* data) {
    while(1) {
        OSTimeDlyHMSM(0,0,2,0);
        if(intensity == 255) intensity = 5;
            else intensity+=25;
            putchar('b');
    }
}

void Task4(void* data) {
    while(1) {
            /* csunya megoldas
            OSTimeDlyHMSM(0,0,0,100); //pergesmentesites
            if(!(PINE & 32) && intensity != 255)
                intensity+=25;
            if(!(PINE & 64) && intensity != 5)
                intensity-=25;
            */
            // szep megoldas
            char NEW1=0;
            char OLD1=0;
            char NEW2=0;
            char OLD2=0;
            NEW1=PINE&32;
            NEW2=PINE&64;
            if((OLD1!=NEW1) && intensity != 255)
                    intensity+=25;
            if((OLD2!=NEW2) && intensity != 5)
                    intensity-=25;
            OLD1=NEW1;
            OLD2=NEW2;
            OSTimeDlyHMSM(0,0,0,20); //pergesmentesites
    }
}
```

3 - 3.) Futófény IT-vel.

Írjon futófény programot, ahol a LED felgyulladás sorrendje a BT0 gomb megnyomásával megsordítható. Ne legyen olyan fázis, hogy egy LED sem világít! A gombot IT-vel kezelje!

Megoldás:

```
char dir=1;

ISR(INT5_vect) {
   dir = dir ^ 1;
}

int main() {
   DDRC = 0xFF;
   PORTC = 1;
   EIMSK |= 32;
   EICRB = 0x02;

   sei();
   while(1) {
          cli();
          _delay_ms(1000);
          if(dir == 1) {
             if(PORTC == 128) PORTC = 1;
             else PORTC <<= 1;
          } else {
             if(PORTV == 1) PORTC = 128;
             else PORTC >>= 1;
          }
          sei();
   }

   return 0;
}
```

3 - 4.) LCD háttérvilágítás szabályzó + futófény
Írja meg az alábbi, egy ISR-böl és két taszkból álló µC/OS alkalmazást:

- ISR – A/D átalakító kiolvasása: az interrupt rutin egy globális változóba olvassa ki az optokapuról történö A/D átalakítás eredményét, és ezen esemény megtörténtét egy szemaforral jelezze!

- Egyik taszk – LCD háttérvilágítás szabályzó: a háttérvilágítás szabályozására a globális változó értékét használja, melynek érvényességét a szemafor jelzi. Az A/D átalakítást ez a taszk kezdeményezi. (A konvertert single conversion módban használja!)

- Másik taszk – futófény: a LED-soron „vándoroljon" egy LED folyamatosan valamelyik irányba, szemmel még követhetı sebességgel. Ha a fény „kilépett" a LED- sor egyik végén, „jöjjön vissza" az indulási oldalon. Ne legyen olyan állapot, amikor egyik LED sem ég! A késleltetésre használja az OS idökezelö függvényeit!

Megoldás:

```
#include "includes.h"

// Stack sizes
#define TASK1_STK_SIZE  128

#define TASK1_PRIO 5
#define TASK2_PRIO 6
#define TASK3_PRIO 7

// Stacks
OS_STK Task1Stk[TASK1_STK_SIZE];
OS_STK Task2Stk[TASK1_STK_SIZE];
OS_STK Task3Stk[TASK1_STK_SIZE];

void Task1(void *data);
void Task2(void *data);
void Task3(void *data);

INT16U volatile adc_value;
OS_EVENT *sem;


int main (void) {

        ADC_init(ADC_OPTO, ADC_SINGLE);

        LCD_init();
        stdout = &LCD_stdout;


        /* Initializing inner OS variables, creating idle task, ... */
        OSInit();
        /* Create LCD backlight controlling task */
        OSTaskCreate(Task1, NULL, &Task1Stk[TASK1_STK_SIZE - 1], TASK1_PRIO);
        /* Create semaphore for event signaling */
        sem = OSSemCreate(1);
        /* Start multitasking. ( OSStart() never returns! ) */
        OSStart();

        return 0;
}
```

```
void Task1(void *data) {
	OS_ENTER_CRITICAL();

	TCCR0=0x07; /* Set TIMER0 prescaler to CLK/1024 */
	TIMSK=_BV(TOIE0); /* Enable TIMER0 overflow interrupt */
	TCNT0=256-(CPU_CLOCK_HZ/OS_TICKS_PER_SEC/1024); /* Set the counter initial value */

	OS_EXIT_CRITICAL();
	/* Create other tasks */
	OSTaskCreate(Task2, NULL, &Task2Stk[TASK1_STK_SIZE - 1], TASK2_PRIO); //Task2 lÈtrehoz·sa
	OSTaskCreate(Task3, NULL, &Task3Stk[TASK1_STK_SIZE - 1], TASK3_PRIO); //Task3 lÈtrehoz·sa
	/* Delete current task (single-shot task architecture) */

	OSTaskDel(OS_PRIO_SELF); //taszk tˆrlÈse
}

void Task2(void *data) {

	DDRC = 255;
	PORTC = 1;
	while(1){
		if(PORTC == 128)
			PORTC = 1;
		else
			PORTC <<= 1;

		OSTimeDlyHMSM(0,0,1,0);
	}
}

void Task3(void *data) {

	while(1){
		ADC_start();
		OSSemPend(sem, 0, NULL);
		LCD_light((unsigned char)(adc_value>>2));
	}
}

UCOSISR(ADC_vect) {
	PushRS();
	OSIntEnter();
	if (OSIntNesting == 1)
		OSTCBCur->OSTCBStkPtr = (OS_STK *)SP;
	/* Begin of application specific code */

	adc_value = ADC_read();
	OSSemPost(sem);

	OSIntExit();
	PopRS();
}
```

3 - 5.) LCD háttérvilágítás szabályzó + másodperc számláló (LCD)
Írja meg az alábbi, egy ISR-bıl és két taszkból álló µC/OS alkalmazást:
- ISR – A/D átalakító kiolvasása: az interrupt rutin egy globális változóba olvassa ki az optokapuról történı A/D átalakítás eredményét, és ezen esemény megtörténtét egy szemaforral jelezze!

- Egyik taszk – LCD háttérvilágítás szabályzó: a háttérvilágítás szabályozására a globális változó értékét használja, melynek érvényességét a szemafor jelzi. Az A/D átalakítást ez a taszk kezdeményezi. (A konvertert single conversion módban használja!)

- Másik taszk – másodperc számláló (LCD): egy másodperc késleltetésre az OS idıkezelı függvényeit használja! A számláló az LCD kijelzın jelenjen meg, a kiíráshoz pedig a C standard I/O kezelı függvényeit használja!

Megoldás:

```
#include "includes.h"

// Stack sizes
#define TASK1_STK_SIZE  128

#define TASK1_PRIO 5
#define TASK2_PRIO 6
#define TASK3_PRIO 7

// Stacks
OS_STK Task1Stk[TASK1_STK_SIZE];
OS_STK Task2Stk[TASK1_STK_SIZE];
OS_STK Task3Stk[TASK1_STK_SIZE];

void Task1(void *data);
void Task2(void *data);
void Task3(void *data);

INT16U volatile adc_value;
OS_EVENT *sem;


int main (void) {

        ADC_init(ADC_OPTO, ADC_SINGLE);

        LCD_init();
        stdout = &LCD_stdout;


        /* Initializing inner OS variables, creating idle task, ... */
        OSInit();
        /* Create LCD backlight controlling task */
        OSTaskCreate(Task1, NULL, &Task1Stk[TASK1_STK_SIZE - 1], TASK1_PRIO);
        /* Create semaphore for event signaling */
        sem = OSSemCreate(1);
        /* Start multitasking. ( OSStart() never returns! ) */
        OSStart();

        return 0;
}

void Task1(void *data) {

        OS_ENTER_CRITICAL();
```

```
	TCCR0=0x07; /* Set TIMER0 prescaler to CLK/1024 */
	TIMSK=_BV(TOIE0); /* Enable TIMER0 overflow interrupt */
	TCNT0=256-(CPU_CLOCK_HZ/OS_TICKS_PER_SEC/1024); /* Set the counter initial value */
	OS_EXIT_CRITICAL();
	/* Create other tasks */
	OSTaskCreate(Task2, NULL, &Task2Stk[TASK1_STK_SIZE - 1], TASK2_PRIO); //Task2 lÈtrehoz·sa
	OSTaskCreate(Task3, NULL, &Task3Stk[TASK1_STK_SIZE - 1], TASK3_PRIO); //Task3 lÈtrehoz·sa
	/* Delete current task (single-shot task architecture) */

	OSTaskDel(OS_PRIO_SELF); //taszk tˆrlÈse
}
void Task2(void *data) {
	int s = 0;
	int m = 0;

	while(1){
		if(s<59){s++;}
		else{s=0; m++;}
		printf("%d, %d\n",m,s);
		OSTimeDlyHMSM(0,0,1,0);
	}
}

void Task3(void *data) {

	while(1){
		ADC_start();
		OSSemPend(sem, 0, NULL);
		LCD_light((unsigned char)(adc_value>>2));
	}
}

UCOSISR(ADC_vect) {
	PushRS();
	OSIntEnter();
	if (OSIntNesting == 1)
		OSTCBCur->OSTCBStkPtr = (OS_STK *)SP;
	/* Begin of application specific code */

	adc_value = ADC_read();
	OSSemPost(sem);

	OSIntExit();
	PopRS();
}
```

3 - 6.) LCD háttérvilágítás szabályzó + másodperc számláló (soros)
Írja meg az alábbi, egy ISR-bıl és két taszkból álló µC/OS alkalmazást:
- ISR – A/D átalakító kiolvasása: az interrupt rutin egy globális változóba olvassa ki az optokapuról történı A/D átalakítás eredményét, és ezen esemény megtörténtét egy szemaforral jelezze!

- Egyik taszk – LCD háttérvilágítás szabályzó: a háttérvilágítás szabályozására a globális változó értékét használja, melynek érvényességét a szemafor jelzi. Az A/D átalakítást ez a taszk kezdeményezi. (A konvertert single conversion módban használja!)

- Másik taszk – másodperc számláló (soros): egy másodperc késleltetésre az OS idıkezelı függvényeit használja! A számláló a soros porton jelenjen meg, a kiíráshoz pedig a C standard I/O kezelı függvényeit használja!

Megoldás:

```
#include "includes.h"

// Stack sizes
#define TASK1_STK_SIZE  128

#define TASK1_PRIO 5
#define TASK2_PRIO 6
#define TASK3_PRIO 7

// Stacks
OS_STK Task1Stk[TASK1_STK_SIZE];
OS_STK Task2Stk[TASK1_STK_SIZE];
OS_STK Task3Stk[TASK1_STK_SIZE];

void Task1(void *data);
void Task2(void *data);
void Task3(void *data);

INT16U volatile adc_value;
OS_EVENT *sem;


int main (void) {

        ADC_init(ADC_OPTO, ADC_SINGLE);
        LCD_init();
        serial_init();
        stdout = &serial_stdout;


        /* Initializing inner OS variables, creating idle task, ... */
        OSInit();
        /* Create LCD backlight controlling task */
        OSTaskCreate(Task1, NULL, &Task1Stk[TASK1_STK_SIZE - 1], TASK1_PRIO);
        /* Create semaphore for event signaling */
        sem = OSSemCreate(1);
        /* Start multitasking. ( OSStart() never returns! ) */
        OSStart();

        return 0;
}

void Task1(void *data) {
```

```
	OS_ENTER_CRITICAL();
	TCCR0=0x07; /* Set TIMER0 prescaler to CLK/1024 */
	TIMSK=_BV(TOIE0); /* Enable TIMER0 overflow interrupt */
	TCNT0=256-(CPU_CLOCK_HZ/OS_TICKS_PER_SEC/1024); /* Set the counter initial value */
	OS_EXIT_CRITICAL();
	/* Create other tasks */
	OSTaskCreate(Task2, NULL, &Task2Stk[TASK1_STK_SIZE - 1], TASK2_PRIO); //Task2 lÈtrehoz·sa
	OSTaskCreate(Task3, NULL, &Task3Stk[TASK1_STK_SIZE - 1], TASK3_PRIO); //Task3 lÈtrehoz·sa
	/* Delete current task (single-shot task architecture) */

	OSTaskDel(OS_PRIO_SELF); //taszk tˆrlÈse
}

void Task2(void *data) {

	int s = 0;
	int m = 0;
	while(1){
		if(s<59){s++;}
		else{s=0; m++;}
		printf("%d, %d\n",m,s);

		OSTimeDlyHMSM(0,0,1,0);
	}
}


void Task3(void *data) {

	while(1){
		ADC_start();
		OSSemPend(sem, 0, NULL);
		LCD_light((unsigned char)(adc_value>>2));
	}
}

UCOSISR(ADC_vect) {
	PushRS();
	OSIntEnter();
	if (OSIntNesting == 1)
		OSTCBCur->OSTCBStkPtr = (OS_STK *)SP;
	/* Begin of application specific code */

	adc_value = ADC_read();
	OSSemPost(sem);

	OSIntExit();
	PopRS();
}
```

3 - 7.) Potméter állás figyelı (LED) + másodperc számláló (soros)
Írja meg az alábbi, egy ISR-bıl és két taszkból álló µC/OS alkalmazást:

- ISR – A/D átalakító kiolvasása: az interrupt rutin egy globális változóba olvassa ki a potenciométerrıl történı A/D átalakítás eredményét, és ezen esemény megtörténtét egy szemaforral jelezze!

- Egyik taszk – potméter állás figyelı (LED): a LED-eken jelenítse meg bináris formában a globális változó értékét ([9:2] bitek), melynek érvényességét a szemafor jelzi. Az A/D átalakítást ez a taszk kezdeményezi. (A konvertert single conversion módban használja!)

- Másik taszk – másodperc számláló (soros): egy másodperc késleltetésre az OS idıkezelı függvényeit használja! A számláló a soros porton jelenjen meg, a kiíráshoz pedig a C standard I/O kezelı függvényeit használja!

Megoldás:

```
#include "includes.h"

// Stack sizes
#define TASK1_STK_SIZE  128

#define TASK1_PRIO 5
#define TASK2_PRIO 6
#define TASK3_PRIO 7

// Stacks
OS_STK Task1Stk[TASK1_STK_SIZE];
OS_STK Task2Stk[TASK1_STK_SIZE];
OS_STK Task3Stk[TASK1_STK_SIZE];

void Task1(void *data);
void Task2(void *data);
void Task3(void *data);

INT16U volatile adc_value;
OS_EVENT *sem;


int main (void) {

        ADC_init(ADC_POT, ADC_SINGLE);
        serial_init();
        stdout = &serial_stdout;

        /* Initializing inner OS variables, creating idle task, ... */
        OSInit();
        /* Create LCD backlight controlling task */
        OSTaskCreate(Task1, NULL, &Task1Stk[TASK1_STK_SIZE - 1], TASK1_PRIO);
        /* Create semaphore for event signaling */
        sem = OSSemCreate(1);
        /* Start multitasking. ( OSStart() never returns! ) */
        OSStart();

        return 0;
}

void Task1(void *data) {

        OS_ENTER_CRITICAL();
```

```
	TCCR0=0x07; /* Set TIMER0 prescaler to CLK/1024 */
	TIMSK=_BV(TOIE0); /* Enable TIMER0 overflow interrupt */
	TCNT0=256-(CPU_CLOCK_HZ/OS_TICKS_PER_SEC/1024); /* Set the counter initial value */
	OS_EXIT_CRITICAL();
	/* Create other tasks */
	OSTaskCreate(Task2, NULL, &Task2Stk[TASK1_STK_SIZE - 1], TASK2_PRIO); //Task2 lÈtrehoz·sa
	OSTaskCreate(Task3, NULL, &Task3Stk[TASK1_STK_SIZE - 1], TASK3_PRIO); //Task3 lÈtrehoz·sa
	/* Delete current task (single-shot task architecture) */

	OSTaskDel(OS_PRIO_SELF); //taszk tˆrlÈse
}

void Task2(void *data) {
	int s = 0;
	int m = 0;

	while(1){
		if(s<59){s++;}
		else{s=0; m++;}
		printf("%d, %d\n",m,s);

		OSTimeDlyHMSM(0,0,1,0);
	}
}


void Task3(void *data) {
	DDRC = 0xFF;

	while(1){
		ADC_start();
		OSSemPend(sem, 0, NULL);
		PORTC = (unsigned char)(adc_value>>2);
	}

}

UCOSISR(ADC_vect) {
	PushRS();
	OSIntEnter();
	if (OSIntNesting == 1)
		OSTCBCur->OSTCBStkPtr = (OS_STK *)SP;
	/* Begin of application specific code */

	adc_value = ADC_read();
	OSSemPost(sem);

	OSIntExit();
	PopRS();
}
```

3 - 8.) Potméter állás figyelı (LED) + másodperc számláló (LCD)
Írja meg az alábbi, egy ISR-bıl és két taszkból álló µC/OS alkalmazást:

- ISR – A/D átalakító kiolvasása: az interrupt rutin egy globális változóba olvassa ki a potenciométerrıl történı A/D átalakítás eredményét, és ezen esemény megtörténtét egy szemaforral jelezze!

- Egyik taszk – potméter állás figyelı (LED): a LED-eken jelenítse meg bináris formában a globális változó értékét ([9:2] bitek), melynek érvényességét a szemafor jelzi. Az A/D átalakítást ez a taszk kezdeményezi. (A konvertert single conversion módban használja!)

- Másik taszk – másodperc számláló (LCD): egy másodperc késleltetésre az OS idıkezelı függvényeit használja! A számláló az LCD kijelzın jelenjen meg, a kiíráshoz pedig a C standard I/O kezelı függvényeit használja!

Megoldás:

```
#include "includes.h"

// Stack sizes
#define TASK1_STK_SIZE  128

#define TASK1_PRIO 5
#define TASK2_PRIO 6
#define TASK3_PRIO 7

// Stacks
OS_STK Task1Stk[TASK1_STK_SIZE];
OS_STK Task2Stk[TASK1_STK_SIZE];
OS_STK Task3Stk[TASK1_STK_SIZE];

void Task1(void *data);
void Task2(void *data);
void Task3(void *data);

INT16U volatile adc_value;
OS_EVENT *sem;


int main (void) {

        ADC_init(ADC_POT, ADC_SINGLE);

        LCD_init();
        stdout = &LCD_stdout;

        /* Initializing inner OS variables, creating idle task, ... */
        OSInit();
        /* Create LCD backlight controlling task */
        OSTaskCreate(Task1, NULL, &Task1Stk[TASK1_STK_SIZE - 1], TASK1_PRIO);
        /* Create semaphore for event signaling */
        sem = OSSemCreate(1);
        /* Start multitasking. ( OSStart() never returns! ) */
        OSStart();

        return 0;
}

void Task1(void *data) {

        OS_ENTER_CRITICAL();
```

```
	TCCR0=0x07; /* Set TIMER0 prescaler to CLK/1024 */
	TIMSK=_BV(TOIE0); /* Enable TIMER0 overflow interrupt */
	TCNT0=256-(CPU_CLOCK_HZ/OS_TICKS_PER_SEC/1024); /* Set the counter initial value */
	OS_EXIT_CRITICAL();
	/* Create other tasks */
	OSTaskCreate(Task2, NULL, &Task2Stk[TASK1_STK_SIZE - 1], TASK2_PRIO); //Task2 lÈtrehoz·sa
	OSTaskCreate(Task3, NULL, &Task3Stk[TASK1_STK_SIZE - 1], TASK3_PRIO); //Task3 lÈtrehoz·sa
	/* Delete current task (single-shot task architecture) */

	OSTaskDel(OS_PRIO_SELF); //taszk tˆrlÈse
}

void Task2(void *data) {
	int s = 0;
	int m = 0;

	while(1){
		if(s<59){s++;}
		else{s=0; m++;}
		printf("%d, %d\n",m,s);

		OSTimeDlyHMSM(0,0,1,0);
	}
}


void Task3(void *data) {
	DDRC = 0xFF;

	while(1){
		ADC_start();
		OSSemPend(sem, 0, NULL);
		PORTC = (unsigned char)(adc_value>>2);
	}

}

UCOSISR(ADC_vect) {
	PushRS();
	OSIntEnter();
	if (OSIntNesting == 1)
		OSTCBCur->OSTCBStkPtr = (OS_STK *)SP;
	/* Begin of application specific code */

	adc_value = ADC_read();
	OSSemPost(sem);

	OSIntExit();
	PopRS();
}
```