11: Echtzeitbetriebssystem uCOS-II

Sie lernen anhand aufeinander aufbauender Übungen, welche Möglichkeiten ein Echtzeitbetriebssystem wie das uCosII bietet und wie sich damit MC-Applikationen realisieren lassen.

Aufgabe 1: Blinklicht

Implementieren Sie einen Task, welcher die LED 1 von Port D (PTDD0) und einer, der die LED 3 von Port D (PTDD3) blinken lässt. Damit die Unabhängigkeit der Tasks sichtbar wird, lassen Sie die beiden LEDs unterschiedlich schnell blinken.

aufgabe1.c

```
#include <hidef.h>      // for EnableInterrupts macro
#include "platform.h"   // include peripheral declarations
#if AUFGABE == 1

#define PRIO_INIT           0           // max. priority for the init task.
#define PRIO_BLINK_D1       1
#define PRIO_BLINK_D3       2

#define TASK_STK_SIZE       128         // Stacksize for one Task

OS_STK InitTaskStack[TASK_STK_SIZE];    // allocate memory for the stack of the init task
OS_STK BlinkD1Stack[TASK_STK_SIZE];     // allocate memory for the stack of the BlinkD1 task
OS_STK BlinkD3Stack[TASK_STK_SIZE];     // allocate memory for the stack of the BlinkD3 task


/**
 * This Task lets Bit1 of Port D blinking.
 *
 * @param pdata optional data (unused).
 */
void BlinkD1(void *pdata)
{
    int i = 0;

    (void)pdata;                        // prevents compiler warning
    for(;;)
    {
      // @todo implement BlinkD1-Task here
      PTDD_PTDD1 = 0;
      OSTimeDly(50); // 50 * 20ms = 1'000ms delay

      PTDD_PTDD1 = 1;
      OSTimeDly(50); // 50 * 20ms = 1'000ms delay
    }
}


/**
 * This task lets Bit3 of Port D blinking.
 *
 *  @param pdata optional data (unused).
 */
void BlinkD3(void *pdata)
{
    (void)pdata;

    for(;;)
    {
      // @todo implement BlinkD3-Task here...
      PTDD_PTDD3 = 0;
      (void)OSTimeDlyHMSM(0,0,0,50);

      PTDD_PTDD3 = 1;
      (void)OSTimeDlyHMSM(0,0,0,50);
    }
}
```

```
/**
 * Timer etc. initialisieren...
 *
 * @param pdata Optionale Daten für den Task beim Start.
 */
void InitTask(void* pdata)
{
    PTDDD_PTDDD1 = 1; // Output
    PTDDD_PTDDD3 = 1; // Output
    PTDD_PTDD1 = 0;   // Enable LED
    PTDD_PTDD3 = 0;   // Enable LED
    // PTDD = 0xFF;   // alle LED Off
    // PTDDD = 0xFF;  // set Output

    (void)pdata;
    OSTimerInit();                  // start timer (RTI)

    (void)OSTaskCreate(BlinkD1, (void*)0, &BlinkD1Stack[TASK_STK_SIZE-1], PRIO_BLINK_D1);
    (void)OSTaskCreate(BlinkD3, (void*)0, &BlinkD3Stack[TASK_STK_SIZE-1], PRIO_BLINK_D3);

    (void)OSTaskDel(OS_PRIO_SELF);  // destroy the init task.
}


/**
 * main program
 *
 * This function has the following tasks:
 * - init system clock
 * - init operating system
 * - create the first (init-) task
 * - start the os
 */
void main(void)
{
    OSInit();                       // initialize the operating system

    // create the init task
    (void)OSTaskCreate(InitTask, (void*)0, &InitTaskStack[TASK_STK_SIZE-1], PRIO_INIT);

    OSStart();                      // start the operating system
}
#endif
```

Aufgabe 2: Zwei parallele Lauflichter

In dieser Aufgabe geht es darum, zwei unabhängige Lauflichter zu implementieren. Sobald ein Lauflicht seine Randstelle erreicht, soll es die Richtung wechseln. Beachten Sie dazu, ausgehend von Aufgabe 1, folgendes:

- Verwenden Sie für das erste Lauflicht die LEDs PTDD0..3 und für das zweite Lauflicht die LEDs PTDD4..7
- In einem ersten Schritt wird die Schrittgeschwindigkeit als fixer Wert implementiert.
- Implementieren Sie das Multitasking-Programm mit Hilfe von 3 Tasks. Einen Task für die Initialisierung, sowie je einen Task für die Lauflichter.
- Testen Sie das Programm
- Implementieren Sie mit Hilfe eines weiteren Tasks die Geschwindigkeitsregelung mit Eingabe via den Potentiometern auf dem L-Print. PTB6 stellt die Geschwindigkeit für das linke Lauflicht (PTDD4..7) und PTB7 für das rechte Lauflicht (PTDD0..3) ein.

aufgabe2.c

```
#include <hidef.h>      // for EnableInterrupts macro
#include "platform.h"   // include peripheral declarations
#if AUFGABE == 2

#define PRIO_INIT           0           // max. priority for the init task.
#define PRIO_ADC            1
#define PRIO_LAUFL_1        2
#define PRIO_LAUFL_2        3

#define TASK_STK_SIZE       128         // Stacksize for one Task

OS_STK InitTaskStack[TASK_STK_SIZE];    // allocate memory for the stack of the init task
OS_STK ADCStack[TASK_STK_SIZE];         // allocate memory for the stack of the BlinkD1 task
OS_STK LAUFL1Stack[TASK_STK_SIZE];      // allocate memory for the stack of the BlinkD1 task
OS_STK LAUFL2Stack[TASK_STK_SIZE];      // allocate memory for the stack of the BlinkD3 task

// init ADC global var
unsigned char ucDelayKitRight = 10;
unsigned char ucDelayKitLeft  = 10;

/**
 * This task reads alternately the value of ATDCH 6 and 7 and
 * stores the 8 bit value in the variables ucDelayKitLeft and
 * ucDelayKitRight.
 *
 * @param pdata optional data (unused).
 */
void AdcTask(void *pdata)
{
    (void)pdata;
    for(;;)
    {
        OSTimeDly(5);
        if(ATD1SC_ATDCH == 7)
        {                                       // if +1 is neglected KitLeft
            ucDelayKitRight = (ATD1RH / 4) + 1; // is blocked when poti7 is in
            ATD1SC_ATDCH=6;                     // left most position, because
        }                                       // KitLeft has lower prio
        else
        {
            ucDelayKitLeft = (ATD1RH / 4) + 1;
            ATD1SC_ATDCH=7;
        }
    }
}


/**
 * Timer etc. initialisieren...
 *
 * @param pdata Optionale Daten für den Task beim Start.
 */
void adcInit(void)
{
  ATD1C_ATDPU=1;                // ATD power on
  ATD1C_PRS = 2;                // set the prescaler. Valid for a busclock of 3-12 MHz.
  ATD1C_RES8=1;                 // 8BitData
  ATD1PE_ATDPE7=1;              // Pin enabled Channel 7
  ATD1PE_ATDPE6=1;              // Pin enabled Channel 6
  ATD1SC_ATDCO=0;               // continuous conversion
  ATD1SC_ATDCH=7;               // Select Input Channel 7
}
```

```
/**
 * Lauflicht PTDD0..3
 *
 * @param pdata optional data (unused).
 */
void Lauflicht1(void *pdata)
{
    unsigned char ucBitPos = 1;
    unsigned char ucGoLeft = 1;

    (void)pdata;                          // prevents compiler warning
    for(;;)
    {
      switch(ucBitPos)
      {
        case 1: PTDD_PTDD0 = 1; break;
        case 2: PTDD_PTDD1 = 1; break;
        case 4: PTDD_PTDD2 = 1; break;
        case 8: PTDD_PTDD3 = 1; break;
      }

      if (ucGoLeft) {
        ucBitPos <<= 1;
        if (ucBitPos == 8) ucGoLeft = 0;
      } else {
        ucBitPos >>= 1;
        if (ucBitPos == 1) ucGoLeft = 1;
      }

      switch(ucBitPos)
      {
        case 1: PTDD_PTDD0 = 0; break;
        case 2: PTDD_PTDD1 = 0; break;
        case 4: PTDD_PTDD2 = 0; break;
        case 8: PTDD_PTDD3 = 0; break;
      }

      OSTimeDly(ucDelayKitRight); // 50 * 20ms = 1'000ms delay
    }
}

/**
 * Lauflicht PTDD4..8
 *
 * @param pdata optional data (unused).
 */
void Lauflicht2(void *pdata)
{
    unsigned char ucBitPos = 1;
    unsigned char ucGoLeft = 1;

    (void)pdata;                          // prevents compiler warning
    for(;;)
    {
      switch(ucBitPos)
      {
        case 1: PTDD_PTDD4 = 1; break;
        case 2: PTDD_PTDD5 = 1; break;
        case 4: PTDD_PTDD6 = 1; break;
        case 8: PTDD_PTDD7 = 1; break;
      }

      if (ucGoLeft) {
        ucBitPos <<= 1;
        if (ucBitPos == 8) ucGoLeft = 0;
      } else {
        ucBitPos >>= 1;
        if (ucBitPos == 1) ucGoLeft = 1;
      }

      switch(ucBitPos)
      {
        case 1: PTDD_PTDD4 = 0; break;
        case 2: PTDD_PTDD5 = 0; break;
        case 4: PTDD_PTDD6 = 0; break;
        case 8: PTDD_PTDD7 = 0; break;
      }

      OSTimeDly(ucDelayKitLeft); // 50 * 20ms = 1'000ms delay
    }
}
```

```
/**
 * Timer etc. initialisieren...
 *
 * @param pdata Optionale Daten für den Task beim Start.
 */
void InitTask(void* pdata)
{
    PTDD = 0xFF;   // LED off

    PTDDD = 0xFF;  // set Output

    (void)pdata;
    OSTimerInit();                  // start timer (RTI)

    adcInit();  // init ADC


   (void)OSTaskCreate(AdcTask, (void*)0, &ADCStack[TASK_STK_SIZE-1], PRIO_ADC);
   (void)OSTaskCreate(Lauflicht1, (void*)0, &LAUFL1Stack[TASK_STK_SIZE-1], PRIO_LAUFL_1);
   (void)OSTaskCreate(Lauflicht2, (void*)0, &LAUFL2Stack[TASK_STK_SIZE-1], PRIO_LAUFL_2);

    (void)OSTaskDel(OS_PRIO_SELF);  // destroy the init task.
}


/**
 * main program
 *
 * This function has the following tasks:
 * - init system clock
 * - init operating system
 * - create the first (init-) task
 * - start the os
 */
void main(void)
{
    OSInit();                       // initialize the operating system

    // create the init task
    (void)OSTaskCreate(InitTask, (void*)0, &InitTaskStack[TASK_STK_SIZE-1], PRIO_INIT);

    OSStart();                      // start the operating system
}
#endif
```

Aufgabe 3: "Rendezvous" der Lauflichter (Semaphore)

Die zwei unabhängigen, „parallel“ laufenden Tasks aus Aufgabe 2 sollen sich jeweils beim MSB (also bei PTDD7 bzw. PTDD3) synchronisieren. Damit dieses Rendezvous stattfinden kann, muss das schnellere Lauflicht auf das langsamere Lauflicht warten. Realisieren Sie diese Kooperation mit Semaphoren.

Vorgehen:

- Verschaffen Sie sich einen Überblick über die Funktionsweise und den Einsatz von Semaphoren.
- Implementieren Sie das Rendezvous mit Hilfe von Semaphoren und testen es aus.

aufgabe3.c

```
#include <hidef.h>      // for EnableInterrupts macro
#include "platform.h"   // include peripheral declarations
#if AUFGABE == 3

#define PRIO_INIT           0           // max. priority for the init task.
#define PRIO_ADC            1           // high priority for the adc task.
#define PRIO_KIT_RIGHT      2
#define PRIO_KIT_LEFT       3

#define TASK_STK_SIZE       128         // Stacksize for one Task


OS_STK InitTaskStack[TASK_STK_SIZE];    // allocate memory for the stack of the init task
OS_STK KitRightStack[TASK_STK_SIZE];    // allocate memory for the stack of the KitRight task
OS_STK KitLeftStack[TASK_STK_SIZE];     // allocate memory for the stack of the KitLeft task
OS_STK AdcStack[TASK_STK_SIZE];         // allocate memory for the stack of the ADC task

OS_EVENT *semKitLeft;                   // Semaphor
OS_EVENT *semKitRight;                  // Semaphor

unsigned char ucDelayKitLeft = 10;
unsigned char ucDelayKitRight = 10;


/**
 * This task reads alternately the value of ATDCH 6 and 7 and
 * stores the 8 bit value in the variables ucDelayKitLeft and
 * ucDelayKitRight.
 *
 * @param pdata optional data (unused).
 */
void AdcTask(void *pdata)
{
    (void)pdata;
    for(;;)
    {
        OSTimeDly(5);
        if(ATD1SC_ATDCH == 7)
        {
            ucDelayKitRight = (ATD1RH / 4) + 1;
            ATD1SC_ATDCH=6;
        }
        else
        {
            ucDelayKitLeft = (ATD1RH / 4) + 1;
            ATD1SC_ATDCH=7;
        }
    }
}
```

```
/**
 * Kit Right (bit D0..D3)
 *
 * @param pdata optional data (unused).
 */
void KitRightTask(void *pdata)
{
    unsigned char ucBitPos = 1;
    unsigned char ucGoLeft = 1;
    unsigned char ucResult;
    (void)pdata;                        // prevents compiler warning

    for (;;)
    {
        switch (ucBitPos)
        {
            case 1: PTDD_PTDD0 = 1; break;
            case 2: PTDD_PTDD1 = 1; break;
            case 4: PTDD_PTDD2 = 1; break;
            case 8: PTDD_PTDD3 = 1; break;
        }

        if (ucGoLeft)
        {
            ucBitPos <<= 1;
            if (ucBitPos == 8) ucGoLeft = 0;
        }
        else
        {
            ucBitPos >>= 1;
            if (ucBitPos == 1) ucGoLeft = 1;
        }

        switch (ucBitPos)
        {
            case 1: PTDD_PTDD0 = 0; break;
            case 2: PTDD_PTDD1 = 0; break;
            case 4: PTDD_PTDD2 = 0; break;
            case 8:
                PTDD PTDD3 = 0;
                (void)OSSemPost(semKitLeft);
                OSSemPend(semKitRight, 0, &ucResult);
                break;
        }
        OSTimeDly(ucDelayKitRight);
    }
}
```

```
/**
 * Kit Left (bit D4..D7)
 *
 *  @param pdata optional data (unused).
 */
void KitLeftTask(void *pdata)
{
    unsigned char ucBitPos = 1;
    unsigned char ucGoLeft = 1;
    unsigned char ucResult;
    (void)pdata;                        // prevents compiler warning

    for (;;)
    {
        switch (ucBitPos)
        {
            case 1: PTDD_PTDD4 = 1; break;
            case 2: PTDD_PTDD5 = 1; break;
            case 4: PTDD_PTDD6 = 1; break;
            case 8: PTDD_PTDD7 = 1; break;
        }

        if (ucGoLeft)
        {
            ucBitPos <<= 1;
            if (ucBitPos == 8) ucGoLeft = 0;
        }
        else
        {
            ucBitPos >>= 1;
            if (ucBitPos == 1) ucGoLeft = 1;
        }

        switch (ucBitPos)
        {
            case 1: PTDD_PTDD4 = 0; break;
            case 2: PTDD_PTDD5 = 0; break;
            case 4: PTDD_PTDD6 = 0; break;
            case 8:
                PTDD PTDD7 = 0;
                (void)OSSemPost(semKitRight);
                OSSemPend(semKitLeft, 0, &ucResult);
                break;
        }
        OSTimeDly(ucDelayKitLeft);
    }
}


/**
 * Timer etc. initialisieren...
 *
 * @param pdata Optionale Daten für den Task beim Start.
 */
void InitTask(void* pdata)
{
    (void)pdata;
    OSTimerInit();                  // start timer (RTI)
    PTDD = 0xFF;                    // disable LED's on Port D
    PTDDD = 0xFF;                   // configure Port D as output

    ATD1C_PRS = 2;                  // set the prescaler. Valid for a busclock of 3-12 MHz.
    ATD1C_ATDPU=1;                  // ATD power on
    ATD1C_RES8=1;                   // 8BitData
    ATD1PE_ATDPE7=1;                // Pin enabled Channel 7
    ATD1PE_ATDPE6=1;                // Pin enabled Channel 6
    ATD1SC_ATDCO=0;                 // continuous conversion
    ATD1SC_ATDCH=7;                 // Select Input Channel 7

    semKitLeft=OSSemCreate(0);
    semKitRight=OSSemCreate(0);

    (void)OSTaskCreate(AdcTask, (void*)0, &AdcStack[TASK_STK_SIZE-1], PRIO_ADC);
    (void)OSTaskCreate(KitRightTask, (void*)0, &KitRightStack[TASK_STK_SIZE-1], PRIO_KIT_RIGHT);
    (void)OSTaskCreate(KitLeftTask, (void*)0, &KitLeftStack[TASK_STK_SIZE-1], PRIO_KIT_LEFT);

    (void)OSTaskDel(OS_PRIO_SELF);  // destroy the init task.
}
```

```
/**
 * main program
 *
 * This function has the following tasks:
 * - init system clock
 * - init operating system
 * - create the first (init-) task
 * - start the os
 */
void main(void)
{
    OSInit();                       // initialize the operating system

    // create the init task
    (void)OSTaskCreate(InitTask, (void*)0, &InitTaskStack[TASK_STK_SIZE-1], PRIO_INIT);

    OSStart();                      // start the operating system
}
#endif
```

Aufgabe 4: " Producer-Consumer" der Lauflichter

In dieser Aufgabe soll nun das linke Lauflicht zum Producer und das rechte Lauflicht zum Consumer werden: Der Producer produziert Daten, d.h. bei jedem Schritt gibt er die Werte der Portbits PTDD4..7 zur Verarbeitung an den Consumer weiter. Die Daten werden über eine Queue, die maximal 4 Messages aufnehmen kann, an den Consumer-Task übermittelt. Der Consumer liest die Daten aus der Queue und zeigt sie an den Portbits PTD0..3 an. Falls keine Daten in der Queue vorhanden sind, so wartet der Consumer-Task bis dies der Fall ist. Falls die Queue voll ist, wartet der Producer-Task, bis wieder Platz für mindestens eine Message vorhanden ist.

Ein fünfter Task "ShowMailStore" zeigt den Füllgrad der Queue in Form einer Balkenanzeige am Port F an. Der dazu benötigte Code steht nachfolgend zur Verfügung:

```
/**
 * Shows the level (0..4) at port F as a bar graph
 *
 * @param pdata optional data (unused).
 */
void ShowMsgQueue(void *pdata)
{
    INT8U count;
    OS_Q_DATA tempData;
    (void)pdata;          // prevents compiler warning

    PTFDD = 0x0F;

    for(;;)
    {
        (void)OSQQuery(msgQueue, &tempData);
        count = (INT8U)tempData.OSNMsgs;
        PTFD = (0x0f << count);
        OSTimeDly(4);
    }
}
```

Implementieren Sie das Producer-Consumer Problem und fügen Sie den fünften Task "ShowMailStore" hinzu.

Anmerkung: Normalerweise wird bei der Queue nur ein Pointer auf die eigentliche Message übermittelt. In unserem Fall verwenden wir diesen 16-Bit Speicherplatz direkt als Datenwert und verwenden ihn nicht als Pointer. Im uCOS-II Handbuch steht dazu:

A message queue (or simply a queue) is a mC/OS-II object that allows a task or an ISR to send pointer size variables to another task. Each pointer would typically be initialized to point to some application specific data structure containing a ‘ message’.

aufgabe4.c

```
#include <hidef.h>      // for EnableInterrupts macro
#include "platform.h"   // include peripheral declarations
#if AUFGABE == 4

#define PRIO_INIT           0           // max. priority for the init task.
#define PRIO_ADC            1           // high priority for the adc task.
#define PRIO_PRODUCER       2
#define PRIO_CONSUMER       3
#define PRIO_QUEUE_STATUS   4

#define TASK_STK_SIZE       128         // Stacksize for one Task
#define MSG_QUEUE_SIZE      4           // Size of the Message Queue

OS_STK InitTaskStack[TASK_STK_SIZE];     // allocate memory for the stack of the init task
OS_STK ConsumerStack[TASK_STK_SIZE];     // allocate memory for the stack of the consumer task
OS_STK ProducerStack[TASK_STK_SIZE];     // allocate memory for the stack of the producer task
OS_STK AdcStack[TASK_STK_SIZE];          // allocate memory for the stack of the ADC task
OS_STK ShowMailStoreStack[TASK_STK_SIZE];// allocate memory for the stack of the ShowMailQueue task

OS_EVENT *msgQueue;                     // Message Queue
void *daten[MSG_QUEUE_SIZE];            // memory for the message queue

unsigned char ucDelayProducer = 10;
unsigned char ucDelayConsumer = 10;

/**
 * This task reads alternately the value of ATDCH 6 and 7 and
 * stores the 8 bit value in the variables ucDelayProducer and
 * ucDelayConsumer.
 *
```

```
 * @param pdata optional data (unused).
 */
void AdcTask(void *pdata)
{
    (void)pdata;
    for(;;)
    {
        OSTimeDly(5);
        if(ATD1SC_ATDCH == 7)
        {
            ucDelayConsumer = (ATD1RH / 4) + 1;
            ATD1SC_ATDCH=6;
        }
        else
        {
            ucDelayProducer = (ATD1RH / 4) + 1;
            ATD1SC_ATDCH=7;
        }
    }
}

/**
 * Consumer (bit D0..D3)
 *
 * @param pdata optional data (unused).
 */
void ConsumerTask(void *pdata)
{
    unsigned char ucResult;
    unsigned char ucMessage;
    (void)pdata;                        // prevents compiler warning

    for (;;)
    {
        ucMessage = (unsigned char)OSQPend(msgQueue, 0, &ucResult);

        OS_ENTER_CRITICAL();
        PTDD = (PTDD & 0xF0) | ucMessage;
        OS_EXIT_CRITICAL();

        (void)OSTaskResume(PRIO_PRODUCER);
        OSTimeDly(ucDelayConsumer);
    }
}


/**
 * Producer (bit D4..D7)
 *
 *  @param pdata optional data (unused).
 */
void ProducerTask(void *pdata)
{
    unsigned char ucBitPos = 1;
    unsigned char ucGoLeft = 1;
    unsigned char ucResult;
    (void)pdata;                        // prevents compiler warning

    for (;;)
    {
        switch (ucBitPos)
        {
            case 1: PTDD_PTDD4 = 1; break;
            case 2: PTDD_PTDD5 = 1; break;
            case 4: PTDD_PTDD6 = 1; break;
            case 8: PTDD_PTDD7 = 1; break;
        }

        if (ucGoLeft)
        {
            ucBitPos <<= 1;
            if (ucBitPos == 8) ucGoLeft = 0;
        }
        else
        {
            ucBitPos >>= 1;
            if (ucBitPos == 1) ucGoLeft = 1;
        }
```

```
        switch (ucBitPos)
        {
            case 1: PTDD_PTDD4 = 0; break;
            case 2: PTDD_PTDD5 = 0; break;
            case 4: PTDD_PTDD6 = 0; break;
            case 8: PTDD_PTDD7 = 0; break;
        }

        do
        {
            // save the message into the queue
            ucResult = OSQPost(msgQueue, (void *)(PTDD >> 4) );

            // Suspend the Task, if there is no space in the queue
            if (ucResult == OS_Q_FULL) OSTaskSuspend(OS_PRIO_SELF);
        }
        while (ucResult != OS_NO_ERR); // repeat until there is no error.

        OSTimeDly(ucDelayProducer);
    }
}


/**
 * Shows the level (0..4) at port F as a bar graph
 *
 * @param pdata optional data (unused).
 */
void ShowMsgQueue(void *pdata)
{
    INT8U count;
    OS_Q_DATA tempData;
    (void)pdata;                        // prevents compiler warning

    PTFDD = 0x0F;

    for(;;)
    {
        (void)OSQQuery(msgQueue, &tempData);
        count = (INT8U)tempData.OSNMsgs;
        PTFD = (0x0f << count);
        OSTimeDly(4);
    }
}


/**
 * Timer etc. initialisieren...
 *
 * @param pdata Optionale Daten für den Task beim Start.
 */
void InitTask(void* pdata)
{
    (void)pdata;
    OSTimerInit();                  // start timer (RTI)
    PTDD = 0xFF;                    // disable LED's on Port D
    PTDDD = 0xFF;                   // configure Port D as output

    ATD1C ATDPU=1;                  // ATD power on
    ATD1C_PRS = 2;                  // set the prescaler. Valid for a busclock of 3-12 MHz.
    ATD1C_RES8=1;                   // 8BitData
    ATD1PE_ATDPE7=1;                // Pin enabled Channel 7
    ATD1PE ATDPE6=1;                // Pin enabled Channel 6
    ATD1SC_ATDCO=1;                 // continuous conversion
    ATD1SC_ATDCH=7;                 // Select Input Channel 7

    msgQueue = OSQCreate(daten, MSG_QUEUE_SIZE);

    (void)OSTaskCreate(AdcTask, (void*)0, &AdcStack[TASK_STK_SIZE-1], PRIO_ADC);
    (void)OSTaskCreate(ConsumerTask, (void*)0, &ConsumerStack[TASK_STK_SIZE-1], PRIO_CONSUMER);
    (void)OSTaskCreate(ProducerTask, (void*)0, &ProducerStack[TASK_STK_SIZE-1], PRIO_PRODUCER);
    (void)OSTaskCreate(ShowMsgQueue, (void*)0, &ShowMailStoreStack[TASK_STK_SIZE-1],
PRIO_QUEUE_STATUS);

    (void)OSTaskDel(OS_PRIO_SELF);  // destroy the init task.
}
```

```
/**
 * main program
 *
 * This function has the following tasks:
 * - init system clock
 * - init operating system
 * - create the first (init-) task
 * - start the os
 */
void main(void)
{
    OSInit();                       // initialize the operating system

    // create the init task
    (void)OSTaskCreate(InitTask, (void*)0, &InitTaskStack[TASK_STK_SIZE-1], PRIO_INIT);

    OSStart();                      // start the operating system
}
#endif
```