```
/*
**  led7seg.c - Sample code for HyperpanelOS ==============================  **
**                                                                            **
**  This simple code is located into the application container, it is run    **
**  by the VMK sub-operating system. On the other hand, the I/O container    **
**  runs all the drivers that are VMIO finite state machines.                **
**                                                                            **
**  The goal of this small app is to use a LED 7-segments I2C display device **
**                                                                            **
**  ========================================================================  **
*/


/* Documentation for I2C devices ------------------------------------------

   - HT16K33 18*8 LED Controller data sheet

*/
/* Include files and external reference ----------------------------------*/

#include <hypos.h>                      // Hyperpanel OS basic interfaces.    */
#include <drv_asy.h>                    // Prototype of "asy_write()".        */
#include <drv_i2c.h>                    // Prototype of "i2c_*()".            */


/* Internal defines of this module ---------------------------------------*/

#define  TICK                 1000     // Code for tick event.

#define DUMMY_AD              0xFF     // I2C dev address - Dummy for wait.
#define LED_AD                0xE0     // I2C dev address - LED (8bits address)

#define INIT                     0     // I2C command - Bargraph initialisation


/*  Internal global variables of this module ------------------------------*/

    static int      idto             ; // Timer identifier.


/* INIT command ..........................................................*/

    static const char init[]         = // I2C command - Display initialisation
      {

/*    Length-1, Address, Control byte, Data byte                            */

      1, LED_AD   , 0x21             , // BAR dev  : Turn on  oscillator

      1, LED_AD   , 0x81             , // BAR dev  : Display ON Blinking OFF

                                       // BAR dev  - Clear the display.
     11, LED_AD   , 0x00, 0x00, 0x00,  // Character 1
                          0x00, 0x00,  // Character 2
                          0x00, 0x00,  // Double dot in the middle
                          0x00, 0x00,  // Character 3
                          0x00, 0x00,  // Character 4

      0, 0        , 0    , 0         , // End of command set
      }                                ; //

    static char   *command[] =         // I2C commands table
      {                                //
        (char*)&init[0]                , // I2C command - INIT
        (char*)0                       // End of list
      }                                ; //

    static const char digit[]        = // Digit on 7-segment coding
```

```
      {
        0x3F                        , // "0" digit
        0x06                        , // "1" digit
        0x5B                        , // "2" digit
        0x4F                        , // "3" digit
        0x66                        , // "4" digit
        0x6D                        , // "5" digit
        0x7D                        , // "6" digit
        0x07                        , // "7" digit
        0x7F                        , // "8" digit
        0x6F                        , // "9" digit
      }                             ;

/*  Prototypes ----------------------------------------------------------------*/

    static int  loop_app_task(void*)     ; // Prototype
    static int  wait_evt(void)           ; // Prototype
    static void set_command(int)         ; // Prototype


/*  Beginning of the code -----------------------------------------------------

    loop_app_tsk         Application entry point
*/

/*  Procedure loop_app_tsk ----------------------------------------------------

    Purpose : This is our task main loop.
*/

int loop_app_tsk (void *param)
  {
    int             ev = TICK        ; // Our event
    int             curtime = -1     ; // Current time
    char            mess[16]         ; // Message to be sent on ASY0
    unsigned int    hour             ; // Message to be sent on ASY0
    unsigned char   buf[20]          ;

/* Step 1 - Start a timer that will send an event every second ..............*/

    set_tto(CLOCK      ,              // Timer mode: clock
            1000       ,              // Duration in milliseconds
            TICK , 0   ,              // Event code and reserve field
            &idto     );              // Timer identifier

/* Step 2 - LED 7-segment initialisation ....................................*/

    asy_write(0,(unsigned char*)"Init ... ",9);

    set_command(INIT)                ; // LED 7-segment display initialization

    asy_write(0,(unsigned char*)"done\r\n",6);

/* Step 3 - Main loop .......................................................*/

    wait_ev :                         // Beginning of loop label

    if ( ev == TICK )                 // If the event is the tick event
      {                               //
        curtime ++                   ; // Time incrementation

/* Step 3.1 - Display timer on serial port (each 5 seconds) .................*/

        if (!(curtime%5))             // Each 5 seconds:
          {
            hsprintf(mess            , // Format de timer message.
            "    %02d:%02d:%02d\r\n" , //
            (curtime / 3600)%24      , // Hour.
```

```
            (curtime /   60)%60      , // Minutes.
             curtime        %60     ); // Seconds.
            asy_write(0              , // Write on ASY0
              (unsigned char*)mess   , // the "mess" message
              strlen(mess)          ); // Count of bytes to be sent
         }

/* Step 3.2 - Display timer on the LED 7-segments display (each second) .....*/

        hour=((curtime/60)%60)*100   + // Minutes.
               curtime%60            ; // Seconds.

        memset(buf   , 0x00  , 15);             // Reset display buffer
        memset(buf   ,   11  ,  1);             // Number of byte to write
        memset(buf+1 , LED_AD,  1);             // I2C device address

        memset(buf+3 ,digit[(hour/1000)%10],1); // Digit 1
        memset(buf+5 ,digit[(hour/ 100)%10],1); // Digit 2
        memset(buf+7 ,0x02                 ,1); // Double dot ON
        memset(buf+9 ,digit[(hour/  10)%10],1); // DIgit 3
        memset(buf+11,digit[ hour%10      ],1); // Digit 4

        i2c_write(0,-1,buf,0)            ; // Send I2C commands.
      }

    ev = wait_evt()                  ; // Unschedule until an event is received
    goto wait_ev                     ; // Wait for the next event

    return  0                        ; // Return code of the procedure
  }


/*  Procedure wait_evt ------------------------------------------------------*/
/*
    Purpose : Unschedule until the next event is received, whatever it is.
*/

static int wait_evt (void)
  {
    int             waitlist[1][3]   ; // Parameter of "waitevt_task"
    int             ret              ; // Return code for "waiyevt_task"

/*************************************************************************
 * Step 1 : Build a list with one WAIT_CODEINT entry that will accept all    *
 * ------   the event codes ranging from 0 to 20000. Then call               *
 *          "waitevt_task", we will be unscheduled until the next event will *
 *          be received                                                      *
 *************************************************************************/

    waitlist[0][0]  = WAIT_CODEINT   ; // All events with
    waitlist[0][1]  = 0              ; // a code between 0
    waitlist[0][2]  = 20000          ; // and 20000

    waitevt_task(waitlist ,            // Address of waiting list
                 1        ,            // Size of "waitlist[]"
                 0        ,            // maximum waiting time = no
                 0        ,            // Do not purge previous events
                 &ret       )        ; // Return code

/*************************************************************************
 * Step 2 : Here we are scheduled again. The VMK has written into its        *
 * ------   global variable "task_evt" a copy of the event that has          *
 *          scheduled us again.                                              *
 *************************************************************************/

    return task_evt.code             ; // Return event code
  }
```

```
/*  Procedure set_command ----------------------------------------------------*/
/*
    Purpose : Send a set of commands to I2C devices.
*/

static void set_command(int cmd)
  {
    i2c_write(
      0                           ,    // Controller number
     -1                           ,    // Target I2C write address
      (unsigned char*)command[cmd],    // Command
      0                           );   // List of option flags
  }
```