```
/*
**  bargraph.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 24-bargraph LED I2C 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 BAR_AD                0xE0     // I2C dev address - RGB.

#define INIT                     0     // I2C command - Bargraph initialisation

#define BAR_OFF                  0     // BAR action - Set to OFF
#define BAR_RED                  1     // BAR action - Set to RED
#define BAR_ORANGE               2     // BAR action - Set to ORANGE
#define BAR_GREEN                3     // BAR action - Set to GREEN


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

    static int      idto             ; // Timer identifier.
    static unsigned short bargraph[4]; // Bars current states.


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

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

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

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

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

      7, BAR_AD   , 0x00, 0x00, 0x00,  // BAR dev  - Clear the bargraph
                            0x00, 0x00,
                            0x00, 0x00,

      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
```

```
      }                             ; //


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

    static int  loop_app_task(void*)     ; // Prototype
    static void set_bar(short,short)     ; // Prototype
    static int  wait_evt(void)           ; // Prototype


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

    loop_app_tsk         Application entry point
    set_bar              Set a bar of the bargraph
*/

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

    Purpose : This is our task main loop.
*/

int loop_app_tsk (void *param)
  {
    int             ev = TICK        ; // Our event
    int             curtime = 0      ; // Current time
    char            mess[16]         ; // Message to be sent on ASY0
    short           i                ; //
    char            t = 0            ;

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

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

/* Step 1 - LCD & RGB screen initialisation ................................*/

    i2c_write(
      0                          ,    // Controller number
     -1                          ,    // Target I2C write address
      (unsigned char*)command[INIT],    // Command
      0                         );    // List of option flags

    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 and on screen ...................*/

        hsprintf(mess               , // Format de timer message.
          "    %02d:%02d:%02d\r\n"  , //
          ((curtime/10)/3600)%24    , // Hour.
          ((curtime/10)/  60)%60    , // Minutes.
           (curtime/10)      %60    , // Seconds.
            curtime%10             ); // Thenths.

        if (!(curtime%10))            // Each second, send a trace.
          {                           //
```

```
            asy_write(0              , // Write on ASY0
              (unsigned char*)mess   , // the "mess" message
              strlen(mess)          ); // Count of bytes to be sent

            t = !t                   ; // Each second, alternate ligh ON and
                                       // light OFF.
            if (!t)                    // Turn OFF the bargraph.
             {
              for (i=23;i >=0;i--)     // Turn off all w
               {set_bar(i,BAR_OFF   );}
             }
            else                       // Turn ON the bargraph.
             {
              for (i=0;i <15;i++)      // [00,...,14] bars are red
               {set_bar(i,BAR_GREEN );}

              for (i=15;i <21;i++)     // [15,...,20] bars are orange
               {set_bar(i,BAR_ORANGE);}

              for (i=21;i <24;i++)     // [12,...,23] bars are red
               {set_bar(i,BAR_RED   );}
              }
          }
      }

    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 set_bar ----------------------------------------------------------*/
/*
    Purpose : Get all values for bar settings.
*/

static void set_bar(short bar, short col)
  {
    unsigned short  reg = 0          ; // Register,value
    unsigned short  val = 0          ; // Bit value
    unsigned char   buf[16]          ; // Buffer of I2C commands

    if (bar < 12)                      // Get the register of the selected bar
      reg = bar / 4        ;           // ("register" is offset in the
    else                               // bargraph array)
      reg = (bar - 12) / 4 ;

    val = bar % 4          ;           // Get the bit value of the
    if (bar >= 12)                     // selected bar
      val += 4             ;

    if (col == BAR_OFF)                // BAR_OFF
      {
        bargraph[reg] &= ~(1<<(val)) & // Turn OFF red and green LED
                         ~(1<<(val+8));
      }

    else if (col == BAR_RED)           // BAR_RED
      {
        bargraph[reg] |=   1<<(val)  ; // Turn ON the red BAR
        bargraph[reg] &= ~(1<<(val+8));// Turn OFF the green BAR
      }

    else if (col == BAR_ORANGE)        // BAR_ORANGE
      {
        bargraph[reg] |= 1<<(val)    | // Turn ON the red LED
                         1<<(val+8)  ; // Turn ON the green LED
```

```
      }

    else if (col == BAR_GREEN)         // BAR_GREEN
      {
        bargraph[reg] |=   1<<(val+8); // Turn ON the green LED
        bargraph[reg] &= ~(1<<(val)) ; // Turn OFF the red LED
      }

    memset(buf   , 0x00  , 9);         // Reset all the array

    memset(buf   ,    7  , 1);         // Lenght -1 ot the message
    memset(buf+1 , BAR_AD, 1);         // I2C address of the bargraph

    memset(buf+3,bargraph[0]&0xFF, 1); // First short
    memset(buf+4,bargraph[0]>>8  , 1); //

    memset(buf+5,bargraph[1]&0xFF, 1); // Second short
    memset(buf+6,bargraph[1]>>8  , 1); //

    memset(buf+7,bargraph[2]&0xFF, 1); // Third short
    memset(buf+8,bargraph[2]>>8  , 1); //

    memset(buf+9 , 0,      1)        ; // End of frame

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


/*  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
  }
```