```
/*
**   flame.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 make a ambient light destection system. **
**                                                                            **
**   ========================================================================  **
*/



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

    Product used :

    - Ben-Gi Mini ADS115 Module 4 channels 16 bits I2C ADC
      www.amazon.fr/gp/product/B07PK1Z5H1
      datasheet-ads115.pdf - Texas Instruments
                           16-bit Analog-to-Digital Converter

    - Analog Ambient Light Sensor from DFRobot available at

      wiki.dfrobot.com/DFRobot_Ambient_Light_Sensor_SKU_DFR0026

      and also available in LattePanda Starter Sensor at www.lattepanda.com


     Ambient Light Sensor:

     - We use the Analog ambient light sensor for Arduino from DfRobot. This
       sensor deliver an analog voltage depends on ambient light. Information
       from DfRobot are not very precise so we cannot have a clear conversion
       between input ligh (in lux) and output voltage. So values given by this
       app are not precise and can be considered just as a simple "darlk/light"
       detection.

     - Sensor specification :

       Supply Voltage: 3.3V to 5V
       Illumination range : 1 Lux to 6000 Lux

     - In this tutorial, we power de light sensor with a 3.3V pin (taken for
       example on the free ASY2 scart).

     - Because we have just few info about the light sensor, the app just make
       a linear conversion from analog read value en lux value. We consider than
       a "O" analog value means "O lux" (no light) and a "100" value (with PGA
       of the ADC) means "max value" for the light sensor. When we put the
       light sensor just in front of an electric light, we ge a value of "100"
       in the first byte of the ADC read register. So we assume dans this analog
       value corresponds to 6000 Lux.


     Analog to Digital Converter (ADC):

     - The ADC use the potentiometer as analog input signal et send digital
       output data to the Pyboard via I2C interface.

     - The ADC converter include one single I2C device. There is 4 analog
       input channels. In this app we use Input 0 (data signal). This
        configuration is setted with bit 14-12 of the configuration register.

     -  We select FS=+/-4.096V for the PGA (Programmable gain amplifier) of the
        ADC (cf.  datasheet of the ADC) because we have no info about output
        voltage signal from the light sensor. This range seems correct.
```

```
        I2C Interface:

        - I2C address are 0x90 (write) and 0x91 (read).

        - The ADS1115 have a configuration register. The app write de configuration
          in the initialisation step, and read the configuration register to check
          that the write operation is ok. Configuration register is set as follow
          (cf.  datasheet page 18/19/20).

          bit 15       Operational status                      0    (default)
          bit 14-12    Multiplexer config - INp=0 INn=GND     100   (INp=0 Nn=GND)
          bit 11-9     Gain amplifier FS= +/- 6.144V          001   (+/-4.096V)
          bit 8        Device operating mode                    0   (continuous)
          bit 7-5      Data rate                              111   (860SPS)
          bit 4        Comparator mode - Traditional            0   (default)
          bit 3        Comparator polarity - Active low         0   (default)
          bit 2        Nn-latching comparator                   0   (default)
          bit 1-0      Disable comparator                      11   (default)

                                        0100001011100011 = 0x42E3

        Serial output:

         The ASY0 output is use to send information messages about the current
         value read from the light sensor.

*/
/* 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 ADC_AD_W            0x90      // I2C dev address - ADC (write)
#define ADC_AD_R            0x91      // I2C dev address - ADC (read)

#define CONVERSION_REG      0x00      // I2C register - Converted value
#define CONFIG_REG          0x01      // I2C register - Configuration

#define INIT                  0      // I2C command - ADC initialisation


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

    static int      idto              ; // Timer identifier.


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

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

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

      3, ADC_AD_W , CONFIG_REG       , // Write the configuration register
                    0x42, 0xE3, // Cf. datasheet page 19)
      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 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
    char            mess[16]         ; // Message to be sent on ASY0
    unsigned char   frame[16]        ; // I2C read frame
    int             ret = 0          ; // Return procedure code

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

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

/* Step 2 - ADC initialisation ...........................................*/

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

    set_command(INIT)               ; // ADC initialization

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

    ret = i2c_read(0,ADC_AD_R       , // Just for checking, read the register
              frame,2,CONFIG_REG);  // we have just write.

    hsprintf(mess,
        "Configuration register 0x%02x%02x (%d)\r\n",
                          frame[0],frame[1],ret);

    asy_write(0                      , // Write on ASY0 the value of the
        (unsigned char*)mess         , // configuration register.
              strlen(mess)          ); // Count of bytes to be sent


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

    wait_ev :                        // Beginning of loop label

    if ( ev == TICK )                // If the event is the tick event
      {                              //
        frame[0]=0                  ; // Reset frame.
        frame[1]=0                  ; // Reset frame.

        ret=i2c_read(0,ADC_AD_R     , // Read the current value get from
                  frame,2           , // the ADC.
```

```
                   CONVERSION_REG) ; //

        hsprintf(mess                  , // Message with analog read and message.

            "Ambient light sensor - Analog read  %03d - Light: %d lux\r\n",

            frame[0],(frame[0]*6000)/120,ret); // Just as approximation:
                                         // 6000lux = max component value
                                         //         from documentation.
                                         // 120    = max analog value
                                         //         read from component.

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

      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
}
```