```
/* Includes files and external references .................................*/

#include <stdtyp.l>
#include <moteur.h>                    /* vmk.c and vmio.c interface         */
#include <drv.h>                       /* drv.c interface                    */
#include <automaton.h>                 /* Automaton definitions.             */
#include <emu32.h>                     /* E32_ctx                            */
#include <genio.h>                     /* Kbd_desc,Rcu_sym,Rcu_key,...       */

#include <rdrv.h>                      /* drv.c tags interface               */
#include <memoire.h>                   /* mem_buf.c interface                */
#include <hypstring.h>                 /* memcpy strcy .. interface          */
#include <text.h>                      /* Prototype hsprintf().              */
#include <keycode.h>                   /* Key codes from RCU                 */
#include <telecom.h>                   /* Constants and definitions for tags.*/
#include <engine.h>                    /* Constants for automatong engine.   */
#include <drv_rcu.h>                   /* RCU procedures prototypes          */

#include <keym.txt>                    /* Applicative key codes              */
#include <systemm.txt>                 /* WAIT_CODERES                       */
#include <vmkm.txt>                    /* TSK_INIT_DATA TASK_INIT_BSS        */

#include "./myio.h"                    /* Prototypes for "myio_xxx" procs    */
#include "./myapp.h"                   /* Constant and structs for myapp.c   */

static int     my_sem_proc      (void*                                    ) ;
static int     hsled            (int ,int                                 ) ;

static void    snd_trace         (char * ,int                              ) ;
static int     ev_gpio_in        (int                                      ) ;


/* Internal global variables for the user task ------------------------------*/

int                 mystack[512]     ; // The stack of our task
int                 myapp_taskid     ; // Our task id
int                 myapp_memuid     ; // Our memory user id

int                 my_sem [512]     ; // The stack of our task
int                 mysem_taskid     ; // Our task id
int                 mysem_memuid     ; // Our memory user id
int                 sem              ; // Id semaphore


int                 asy_iod0         ; // IOD of devive ASY\DEV0
int                 asy_iod1         ; // IOD of device ASY\DEV1
int                 asy_iod2         ; // IOD of device ASY\DEV2

int                 gpio_iod         ; // IOD of device GPIO\DEV0
int                 gpio_iod1        ; // IOD of subchannel 0 of GPIO\DEV0
int                 gpio_iod2        ; // IOD of subchannel 1 of GPIO\DEV0
int                 gpio_iod3        ; // IOD of subchannel 2 of GPIO\DEV0
int                 gpio_iodu        ; // IOD of subchannel 3 of GPIO\USRB

int                 led_iod          ; // IOD of device LED\DEV0
int                 led_iod0         ; // IOD of subchannel 0 of LED\DEV0
int                 led_iod1         ; // IOD of subchannel 1 of LED\DEV0
int                 led_iod2         ; // IOD of subchannel 2 of LED\DEV0

int                 rcusnd_id        ; // ID for RCU SND
int                 rcurcv_id        ; // ID for RCU RCV

int                 idto             ; // Timer identifier

unsigned char      *buf_snd[ASYMAX]  ; // Address of transmit buffer
unsigned char      *buf_rcv[ASYMAX]  ; // Address of received buffer
int                 tok_rcv[ASYMAX]  ; // Count of receive tokens
```

```
unsigned char      *buf_rcu          ; // Address of RCU send buffer

int                 ev_val           ; // Returned by "wait_myevent"

/* Lists of tags for the "open_xyz" procedures ..............................*/

static const char *asy_taglist0 = "BAUDS=9600\nNBBITS=8\n"
                                  "STOPBIT=1\nPARITY=NONE\nBUFSIZE=0\n"
                                  "FLOWCTL=NONE\nCHAR1=10"        ;

static const char *asy_taglist1 = "BAUDS=9600\nNBBITS=8\n"
                                  "STOPBIT=1\nPARITY=NONE\nBUFSIZE=0\n"
                                  "FLOWCTL=NONE\nCHAR1=10"        ;

static const char *asy_taglist2 = "BAUDS=9600\nNBBITS=8\n"
                                  "STOPBIT=1\nPARITY=NONE\nBUFSIZE=0\n"
                                  "FLOWCTL=NONE\nCHAR1=10"        ;

static const char *gpio_taglist = "FUNC=GPIO\nOUTPUT=HI_Z\nWEAKPULL=UP\n"
                                  "EVENTS=REPORT"                           ;


/*****************************************************************************/
int init_vmk_fsm(Task_grp *g,char *mod, char*conf)
  {

    return 0                       ; // Exit without any error
  }


/*****************************************************************************/
int init_myapp(Task_grp *g,char *mod, char*conf)
  {

    int ret  ;

    create_task("app"           ,      // Taskgroup name
                mytask_proc     ,      // Task entry point
                myfree_proc     ,      // Task termination call-back
                mystack         ,      // Stack address
                sizeof(mystack) ,      // Stack size in bytes
                PRIO_TSK_MIN    ,      // Task Priority
                HNULL           ,      // Parameter for "mytask_proc"
                TASK_INIT_BSS  +       // Init Flags
                TASK_INIT_DATA +       //
                TASK_INIT_CODE  ,      //
                &myapp_taskid     )  ; // Task_id = 0x12FFFF00 + LW

   create_task("app"           ,      // Taskgroup name
                my_sem_proc     ,      // Task entry point
                myfree_proc     ,      // Task termination call-back
                my_sem          ,      // Stack address
                sizeof(my_sem ) ,      // Stack size in bytes
                PRIO_TSK_MIN    ,      // Task Priority
                HNULL           ,      // Parameter for "my_sem_proc"
                TASK_INIT_BSS  +       // Init Flags
                TASK_INIT_DATA +       //
                TASK_INIT_CODE  ,      //
                &mysem_taskid     )  ; // Task_id = 0x12FFFF00 + LW


    start_task(myapp_taskid,           // Send an event to VMK in order
               &ret          )       ; // to request the task start


    alloc_memuid(&myapp_memuid,        // Allocates a user id for alloc_buf
                 &ret          )     ; // and other allocation procedures
```

```
    init_telecom()                  ; // Initialize driver/protocol/services
                                      // API
    return 0                        ; // Exit without any error
  }


/*************************************************************************/
static INT mytask_proc(void *param)
  {

    int ret ;
    open_asy()                      ; // Open the serial ports
    open_gpio()                     ; // Open the GPIO driver
    open_led()                      ; // Open the LED driver

    myio_givetok(asy_iod0  ,          // I/O descriptor
                 2         ,          // Number of receive token
                 1         ,          // Size of receive buffer
                 &tok_rcv[0] )      ; // Counter of given tokens

    myio_givetok(asy_iod1 ,           // I/O descriptor
                 2        ,          // Number of receive token
                 LGRCV12  ,          // Size of receive buffer
                 &tok_rcv[1])       ; // Counter of given tokens

    myio_givetok(asy_iod2 ,           // I/O descriptor
                 2        ,          // Number of receive token
                 LGRCV12  ,          // Size of receive buffer
                 &tok_rcv[2])       ; // Counter of given tokens


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

    wait_ev                         : // Start of our event loop
    start_task(mysem_taskid,          // Send an event to VMK in order
               &ret          )      ; // to request the task start
                                      // ------------------------------------
    ev_val  = wait_myevents()       ; // Unschedule until one event is
                                      // received
    goto wait_ev                    ; // Goto wait for the next event or

    close_asy()                     ; // Close the two serial ports
    close_gpio()                    ; // Close the GPIO driver
    close_led()                     ; // Close the LED driver

    return 0                        ;

  }


/*************************************************************************/
static INT my_sem_proc(void *param)
  {

     int i,j                         ; // for managing the loop
     int ret                         ;

     snd_trace("SEMAPHORE TEST PROGRAM WITH HYPERPANEL OS ... ENJOY...",asy_iod0);
     create_semaphore( 1, &sem )     ; // Init semaphore

     for ( i =0 ; i < 1000 ; i ++ )
       {
        snd_trace("ASKING FOR A SEMAPHORE BEFORE THE CRITICAL SECTION...",asy_iod0);
```

```
        hsled(0,1)                    ; // switch on the red led
        get_semaphore( sem, &ret )    ; // Shall I go throw this call ?
        snd_trace("INSIDE THE CRITICAL SECTION...",asy_iod0);
        hsled(0,0)                    ; // switch off the red led
        hsled(1,1)                    ; // switch on the yellow led
        for ( j =0 ; j < 5  ; j ++ )
          {
            hsled(2,1)                ; // switch on the green led
            suspend_task (1000,&ret)  ; // Please holdon but don't waste time
            hsled(2,0)                ; // switch off the green led
            suspend_task (1000,&ret)  ; // Please holdon but don't waste time
          }
        hsled(1,0)                    ; // switch off the yellow led
       }

     return 0                         ;

  }

/*************************************************************************/
static int myfree_proc(void)
  {
    int             ret             ; // Procedurs return code

    free_memuid(myapp_memuid,         // Frees a user id for alloc_buf
                0x00000007  ,         // mask = buffers/links/huge
                &ret          )     ; //
    return 0                        ;
  }


static void open_asy(void)
  {
    int             ret             ; // Procedures returned code


    myio_open(DRVASY,DEV0,"",&asy_iod0) ; // Open "ASY\DEV0"
    myio_open(DRVASY,DEV1,"",&asy_iod1) ; // Open "ASY\DEV1"
    myio_open(DRVASY,DEV2,"",&asy_iod2) ; // Open "ASY\DEV2"


    myio_setval(asy_iod0,asy_taglist0)  ; // Set "ASY\DEV0" tags
    myio_setval(asy_iod1,asy_taglist1)  ; // Set "ASY\DEV1" tags
    myio_setval(asy_iod2,asy_taglist2)  ; // Set "ASY\DEV2" tags


    alloc_buf(myapp_memuid ,          // Memory user id
              1500         ,          // Size in bytes of requested buffer
              0            ,          // Flags
              &buf_snd[0]  ,          // Address of allocated buffer
              &ret          )       ; // Return code

    alloc_buf(myapp_memuid ,          // Memory user id
              256          ,          // Size in bytes of requested buffer
              0            ,          // Flags
              &buf_snd[1]  ,          // Address of allocated buffer
              &ret          )       ; // Return code

    alloc_buf(myapp_memuid ,          // Memory user id
              256          ,          // Size in bytes of requested buffer
              0            ,          // Flags
              &buf_snd[2]  ,          // Address of allocated buffer
              &ret          )       ; // Return code


  }
```

```
/*  Procedure close_asy ----------------------------------------------------

    Purpose : This procedure closes the two USART, then frees those two
              devices, terminates the ASY module, delete the route that has
              been created for the IND_REPORT events, and finally frees the 2
              telecom buffers that were allocated by "open_asy".
*/

static void close_asy(void)
  {

    int             ret             ; // Return code

    myio_close(asy_iod0)            ; // Closes "ASY\DEV0"
    myio_close(asy_iod1)            ; // Closes "ASY\DEV1"
    myio_close(asy_iod2)            ; // Closes "ASY\DEV2"


    free_buf(buf_snd[0], &ret )     ; // Free "buf_snd0"
    free_buf(buf_snd[1], &ret )     ; // Free "buf_snd1"
    free_buf(buf_snd[2], &ret )     ; // Free "buf_snd1"
  }


/*  Procedure open_gpio ----------------------------------------------------

*/

static void open_gpio(void)
  {


    myio_open(DRVGPIO,DEV0,"",&gpio_iod); // Open GPIO\DEV0


    myio_alloc_sub(gpio_iod,"UNAME=BUT1",&gpio_iod1); // Alloc BUT0 GPIO
    myio_alloc_sub(gpio_iod,"UNAME=BUT2",&gpio_iod2); // Alloc BUT1 GPIO
    myio_alloc_sub(gpio_iod,"UNAME=BUT3",&gpio_iod3); // Alloc BUT2 GPIO
    myio_alloc_sub(gpio_iod,"UNAME=BUTUSR",&gpio_iodu); // Alloc BUTUSR GPIO

    myio_setval(gpio_iod1,gpio_taglist) ; // Configure BUTO GPIO
    myio_setval(gpio_iod2,gpio_taglist) ; // Configure BUT1 GPIO
    myio_setval(gpio_iod3,gpio_taglist) ; // Configure BUT2 GPIO
    myio_setval(gpio_iodu,gpio_taglist) ; // Configure USRBUT GPIO
  }


/*  Procedure close_gpio ---------------------------------------------------

*/

static void close_gpio(void)
  {


    myio_free_sub(gpio_iod1)        ; // Free BUT1 GPIO
    myio_free_sub(gpio_iod2)        ; // Free BUT2 GPIO
    myio_free_sub(gpio_iod3)        ; // Free BUT3 GPIO
    myio_free_sub(gpio_iodu)        ; // Free USR GPIO

    myio_close(gpio_iod)            ; // Closes the GPIO device and the driver
  }


/*****************************************************************************/
static void open_led(void)
  {
```

```
    int ret = 0 ;

    ret = myio_open(DRVLED,DEV0,"",&led_iod)  ; // Open LED\DEV0

    ret = myio_alloc_sub(led_iod,"LEDNAME=RED"    ,&led_iod0); // Allocates RED
    ret = myio_alloc_sub(led_iod,"LEDNAME=GREEN"  ,&led_iod1); // Allocates GREEN
    ret = myio_alloc_sub(led_iod,"LEDNAME=YELLOW" ,&led_iod2); // Allocates YELLOW

    if (ret) ret ++ ;
  }



/*************************************************************************/
static void close_led(void)
  {

    myio_free_sub(led_iod0)          ; // Frees RED
    myio_free_sub(led_iod1)          ; // Frees GREEN
    myio_free_sub(led_iod2)          ; // Frees YELLOW

    myio_close(led_iod)              ; // Closes LED device and driver
  }

/*************************************************************************/


static int hsled(int n,int state)

  {
   int ret = 0 ;

// state = 0 switch off the led if state = 1 switch on the led

    switch ( n )
      {
        case 0 :
          ret = myio_setval(led_iod0, "CMD=POPALL" );
          if (state == 1)
            ret = myio_setval(led_iod0, "PATTERN=ON:0\nCMD=PUSH"     )  ;
          else
            ret = myio_setval(led_iod0, "PATTERN=OFF:0\nCMD=PUSH"     )  ;
        break;

        case 1 :
          ret = myio_setval(led_iod1, "CMD=POPALL" );
          if (state == 1)
            ret = myio_setval(led_iod1, "PATTERN=ON:0\nCMD=PUSH"     )  ;
          else
            ret = myio_setval(led_iod1, "PATTERN=OFF:0\nCMD=PUSH"     )  ;
        break;

        case 2 :
          ret = myio_setval(led_iod2, "CMD=POPALL" );
          if (state == 1)
            ret = myio_setval(led_iod2, "PATTERN=ON:0\nCMD=PUSH"     )  ;
          else
            ret = myio_setval(led_iod2, "PATTERN=OFF:0\nCMD=PUSH"     )  ;
        break;
      }

    return ret                       ; // No error

  }

/*  Procedure wait_myevents ----------------------------------------------

    Purpose : Unschedule until any of my events is received or "msec" seconds
```

```
              have been elapsed. A value of 0 for "msec" means no maximum
              time limit. Accordind to the received event, the retourn value
              of this procedure is as follows :

              Code             Reserve        Return value
              ---------------  --------       ---------------
              IND_REPORT       gpio_iod1  ->  EV_GPIO0_IN   10
              IND_REPORT       gpio_iod2  ->  EV_GPIO1_IN   11
              IND_REPORT       gpio_iod3  ->  EV_GPIO2_IN   12
              TICK             any        ->  EV_MSEC       40

              The "ret" value maybe -1 if we receive something else
*/

static int wait_myevents(void)
  {
    int             waitlist[7][3]   ; // Parameter of "waitevt_task
    int             res              ; // Field "reserve" of task_evt.reserve
    int             ret              ; // Return code


    waitlist[0][0]  = WAIT_CODERES   ; // All events with
    waitlist[0][1]  = IND_REPORT     ; // an event code IND_REPORT
    waitlist[0][2]  = -1             ; // whatever the value of "reserve" is

    waitlist[1][0]  = WAIT_CODERES   ; // All events with
    waitlist[1][1]  = TICK           ; // an event code TICK
    waitlist[1][2]  = 0              ; // with "reserve" equal to request

    waitevt_task(waitlist ,            // Address of waiting list
                 2        ,            // 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. As a code event value,                       *
 *          IND_REPORT,                                                      *
 *          but also the VMK generated event TICK (every second).            *
 *          According to the value of "task_evt.code"                        *
 *          "task_evt.reserve" we compute the return value "ret". If we      *
 *          receive something we are not expecting, we will return a default *
 *          value of -1.                                                     *
 *****************************************************************************/

    res = task_evt.reserve           ; // Extract the "reserve" field from the
    ret = -1                         ; // received event.

    switch ( task_evt.code )
      {
        case IND_REPORT              : // For a IND_REPORT, the "reserve" field
          if      (res EQ gpio_iod1)   // value is the "iod". So we compare
            ret = ev_gpio_in(1)      ; // it to all the IOD's that may send us
          else if (res EQ gpio_iod2)   // a IND_REPORT, so here the 3 buttons,
            ret = ev_gpio_in(2)      ; // so the values may be "gpio_iod1",
          else if (res EQ gpio_iod3)   // "gpio_iod2" ou "gpio_iod3" and not
            ret = ev_gpio_in(3)      ; // more
          else if (res EQ gpio_iodu)   // "gpio_iod2" ou "gpio_iod3" and not
            ret = ev_gpio_in(4)      ; // more
          break                      ; //

        case TICK                    : // For a TICK, the "reserve"
          snd_trace("TIC......",asy_iod0);
          ret = EV_MSEC              ; // field is a copy of the one received
          break                      ; // and we don't care.
```

```
        default                       : // This should never occur if there
          break                       ; // is no bug.

      }

    return ret                        ;
   }

/*  Procedure snd_trace-------------------------------------------------------

    Purpose : Send a message on asy_iod
*/

static void snd_trace(char * mes, int iod)
  {

    unsigned char   *snd              ; // Address of buffer to be sent
    int             d, m, y           ; // Day, Month, Year
    int             h, mi, s, ms      ; // Hour, minutes, seconds, milliseconds
    int              lg               ; // to compute the size of the message

    tim_get(&d  ,                       // day
            &m  ,                       // Month
            &y  ,                       // Year
            &h  ,                       // Hour (0..23)
            &mi ,                       // Minutes (0..59)
            &s  ,                       // Seconds (0..59)
            &ms   )                   ; // Milliseconds (0..999)

    snd = buf_snd[1]                  ; // Buffer to be sent
    hsprintf((char*)snd      ,          // Put in the transmit buffer
             "%02d:%02d:%02d <%s>\n" ,  // 8 characters HH:MM:SS
             h, mi, s , mes        ); //

    lg = strlen((char*)snd)           ; // Number of characters to send
    myio_write  (iod      ,             // I/O descriptor for serial line
                 snd      ,             // Address of buffer
                 lg        );          // Number of bytes to send
  }


/*  Procedure ev_gpio_in -----------------------------------------------------

    Purpose : Parse the received EV_GPIO0/1/2_IN event and select the treatment
              to be executed.
*/

static int ev_gpio_in(int n)
  {

    int ret ;

    if (task_evt.longueur EQ 0)         // If INPUT is LOW,

    switch (n)
      {
        case 1                        : // Digit keys, 0 to 9
          snd_trace("RED BUTTON",asy_iod0);
          break                       ;
        case 2                        :
          snd_trace("GREEN BUTTON",asy_iod0);
          break                       ;
        case 3                        :
          snd_trace("YELLOW BUTTON",asy_iod0);
          break                       ;
        case 4                        :
          snd_trace("PUT SEMAPHORE ",asy_iod0);
```

```
        put_semaphore(sem, &ret );         // The critical section is finished
        break                      ;
      }

  return OPT_IGNO                  ; // Ignore these events
}
```