```
/* 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   */

/* Internal global variables for the AUT_USND automaton ----------------------*/

unsigned short      state_usnd[VLNB] ; // State variables for AUT_USND
unsigned char       sp_usnd [VLNB]   ; // Stack pointers for AUT_USND
S_evt               evt_usnd         ; // Current event for AUT_USND
Usnd_ctx            usnd_ctx[VLNB]   ; // Context tables for AUT_USND
int                 tab_to  [VLNB][2]; // Timer for each logical way
int                 sem_send  = 1    ; // yes you can send a buff

#ifdef BIGTRACE
char                bigtrace[512]    ; // used for debug purpose
#endif


/* Internal data types of the module .....................................*/

static int     producer_proc    (void*                                    ) ;
static int     consumer_proc    (void*                                    ) ;
static void    wait_coderes     (int,int                                  ) ;


static int     hsled            (int ,int                                 ) ;

static int     usnd_launch      (int                                      ) ;
static int     usnd_r2y         (int                                      ) ;
static int     usnd_y2r         (int                                      ) ;
static int     usnd_g2y         (int                                      ) ;
static int     usnd_y2g         (int                                      ) ;
static int     usnd_pop         (int                                      ) ;
static int     usnd_state_led   (int                                      ) ;
static int     trap_snd         (                                         ) ;
static int     trap_evt         (                                         ) ;

/* Internal global variables .............................----------------*/

static void    snd_trace         (char * ,int ,int                        ) ;
static void    snd_proc          (char * ,int ,int                        ) ;
static int     ev_asy_rcv        (int                                     ) ;
static int     ev_asy_snd        (int                                     ) ;
static int     ev_gpio_in        (int                                     ) ;
static int     ev_kbd_rcv        (void                                    ) ;
static int     check_time_out    (void                                    ) ;
```

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

int                 mystack[512]     ; // The stack of our task (2kB)
int                 myapp_taskid     ; // Our task id
int                 myapp_memuid     ; // Our memory user id

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 user of GPIO\DEV0

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                 kbd_iod          ; // IOD of device KBDITF\DEV0
int                 rcusnd_id        ; // ID for RCU SND
int                 rcurcv_id        ; // ID for RCU RCV

int                 idto = -1        ; // 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"
unsigned int        speed            ; // Scheduler for each logicla way
unsigned int        play = 1         ; // By defaut run as fast as requested
int                 showvl = -1      ; // By defaut all logicla vays are shown


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

#define  USTART                20000  // Code for user start event
#define  UPOKE                 20001  // Code for user poke event
#define  PROC                  20002  // Code for inter task event
#define  CONS                  20003  // Code for inter task event

int            producer_stack[128]   ; // The stack of our task (128B)
int            producer_taskid       ; // Our task id

int            consumer_stack[128]   ; // The stack of our task (128B)
int            consumer_taskid       ; // Our task id

int            start_rteid           ; // Route ID for USTART events
int            poke_rteid            ; // Route ID for UPOKE events
int            poke_id  = 0          ; // Identifier of poke
char           message[80]           ; // inter process message


/* Static data for the AUT_USND FSM -----------------------------------------*/

#define SND_START    101               // Start request
#define REQ_LAUNCH   102               // Resquest to launch a VL
#define RES_LAUNCH   103               // Response from a launched vl


#define RED            1               // RED fixed do not cross the line
#define BLK_YELR       2               // GRE Ok you can move forward
```

```
#define BLK_YELG       3               // GRE Ok you can move forward
#define GRE            4               // GRE Ok you can move forward

#define Y2G            5               // GRE Ok you can move forward
#define G2Y            6               // GRE Ok you can move forward
#define R2Y            7               // GRE Ok you can move forward

#define TEMPO          5000            // Tempo of scheduling leds
#define BEAT           50              // Tempo of scheduling logical ways
#define KB_VOLP        626             // Code of the "Vol plus" on the remote
#define KB_VOLM        627             // Code of the "Vol minus" on the remote
#define KB_PAUSE       642             // Code of the "pause" on the remote
#define KB_PLAY        649             // Code of the "play" on the remote

#define STATENB        5               // Transition table size


static S_trans const trans_usnd[] =
  {

/* STOPPED (0) State : Waiting for the SND_START  event .....................*/
#define L_STOPPED  3
RES_LAUNCH  , usnd_launch , 1     , L_STOPPED,// Response from a Logical Way
RESP_WRITE  , ev_asy_snd  , 2     , L_STOPPED ,
SND_START   , usnd_start  , 0     , RED     , // Request to start LIGHT FSM

/* RED     (1) State : RED Light ..........................................*/
#define L_RED     (L_STOPPED + 2)
R2Y         , usnd_r2y    , 1     , BLK_YELR ,// Yellow blinking
RESP_WRITE  , ev_asy_snd  , 2     , RED     ,

/* BLK_YELR (2) State : YELLOW BLINKING FROM RED..........................*/
#define L_BLK_YELR (L_RED     + 2)
Y2G         , usnd_y2g    , 0     , GRE     , // GO TO GREEN STATE
RESP_WRITE  , ev_asy_snd  , 2     , BLK_YELR,

/* BLK_YELG (3) State : YELLOW BLINKING FROM GREEN........................*/
#define L_BLK_YELG (L_BLK_YELR     + 2)
RED         , usnd_y2r    , 0     , RED     , // GO TO RED state
RESP_WRITE  , ev_asy_snd  , 2     , BLK_YELG,

/* GRE     (4) State : GRENN State  ......................................*/
#define L_GRE     (L_BLK_YELG  + 2)
G2Y         , usnd_g2y    , 0     , BLK_YELG, // Back to Yellow state
RESP_WRITE  , ev_asy_snd  , 2     , GRE     ,

  }                                 ; // ---------------------------------

static USHORT const lim_usnd[] =       // List of state limit's in trans_usnd
  {                                    // ---------------------------------
    0         , L_STOPPED    ,         // 1st line for STOPPED, STARTED
    L_RED     , L_BLK_YELR   ,         // 1st line for STOPPED, end of table
    L_BLK_YELG, L_GRE        ,         // 1st line for STOPPED, end of table
  }                                   ; // ---------------------------------


/* 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=115200\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)
  {

    int ret                         ;

    iniaut(AUT_USND   ,               // Automaton number
           trans_usnd ,               // Transition table address
           lim_usnd   ,               // Firt transition number for each state
           STATENB    ,               // Transition table size
           VLNB       ,               // External Logical Ways Number
           VLNB       ,               // Internal Logical Ways Number
           1          ,               // Depth of state stacks
           0          ,               // Depth of the conditions stack
           state_usnd ,               // State variables storage address
           sp_usnd    ,               // Stack pointers storage address
           HNULL      ,               // Conditions stack storage address
           &evt_usnd  ,               // Where to copy the incoming event
           &ret         )           ; // Procedure returned error code


    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
                consumer_proc   ,      // Task entry point
                myfree_proc     ,      // Tasm termination call-back
                consumer_stack  ,      // Stack address
                sizeof(consumer_stack),// 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  ,      //
                &consumer_taskid  )  ; // Task_id = 0x12FFFF00 + LC

    create_task("app"           ,      // Taskgroup name
                producer_proc   ,      // Task entry point
                myfree_proc     ,      // Tasm termination call-back
                producer_stack  ,      // Stack address
                sizeof(producer_stack),// 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  ,      //
                &producer_taskid  )  ; // Task_id = 0x12FFFF00 + LC

    start_task(producer_taskid,&ret) ; // Send an event to VMK in order
                                       // to request the task start
    start_task(myapp_taskid,&ret)    ; // Send an event to VMK in order
                                       // 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 i                            ; // For the loop
    S_evt           evt              ; // SND_START response event

    open_asy()                       ; // Open the serial ports
    open_gpio()                      ; // Open the GPIO driver
    open_led()                       ; // Open the LED driver
    open_kbd()                       ; // Open the KBDITF 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

    speed   = TEMPO                  ; // How long is the tempo for
                                       // one logical way
    for ( i=0 ; i LT VLNB ; i ++ )     // Timers reset
     {
       tab_to  [i   ][0] = 0 ;         // No event for the time out
       tab_to  [i   ][1] = 0 ;         // no value for tne timer
     }

    Memset(&evt,0,sizeof(evt))       ; // Clear the event structure
    evt.aut      = AUT_USND          ; // Target automaton number
    evt.code     = SND_START         ; // Event code
    evt.vl       = 0                 ; // Flag the target logical way
    evt.reserve  = 1                 ; // Flag the target logical way
    evt.res2     = numfa             ; // The good waiting entry point

    putevt_vmk(NFA_STD_PS ,            // VMK Internal queue 2 to 15
                 &evt        )       ; // Event to be written

    set_tto(TIMER      ,               // Timer mode: once
            BEAT       ,               // Duration in milliseconds
            TICK       ,               // Event code
            0          ,               // Event reserve field
```

```
            &idto        )           ; // Timer identifier

    usnd_state_led  (0)              ; // Start leds states

    wait_ev                          : // Start of our event loop
                                       // ------------------------------------
    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
    close_kbd()                      ; // Close the RCU keyboard

    return 0                         ;

  }

/*  Procedure consumer_proc ----------------------------------------------

    Purpose : This is consumer task main loop.
*/

static INT consumer_proc(void *param)
  {
    int             ret              ; // Returned code

/*************************************************************************
 * Step 1 : Then we setup a route to catch the UPOKE event from the producer. *
 * ------                                                                *
 *************************************************************************/

    add_uroute(UPOKE, UPOKE ,          //
               -1, -1     ,            //
               0          ,            //
               &poke_rteid  )        ; //


/*************************************************************************
 * Step 2 : Send a USTART event.                                          *
 * ------                                                                *
 *************************************************************************/

    route_uevent(USTART, 0, HNULL, 0, 0, 0, &ret) ;
    trap_evt();

/*************************************************************************
 * Step 3 : We wait for a new event.                                      *
 * ------                                                                *
 *************************************************************************/

    loop :
    wait_coderes(UPOKE,-1)            ; //

/*************************************************************************
 * Step 5 : Print something then loop back to step 4.                     *
 * ------                                                                *
 *************************************************************************/

    hsprintf((char *) message,"REQID:%08X ",task_evt.reqid) ;
    sendevt_task( myapp_taskid, CONS, &ret );
    goto loop ;

    return 0                         ;
  }
```

```
/*  Procedure producer_proc ----------------------------------------------

    Purpose : This is producer task main loop.
*/

static INT producer_proc(void *param)
  {

    int             ret             ; // Returned code

/*************************************************************************
 * Step 1 : Add route for USTART event.                                  *
 * ------                                                                *
 *************************************************************************/

    add_uroute(USTART, USTART ,        //
               -1, -1       ,          //
               0            ,          //
               &start_rteid   )      ; //

/*************************************************************************
 * Step 2 : Wait for USTART event from consumer and delete route when done.   *
 * ------                                                                *
 *************************************************************************/

    start_task(consumer_taskid,&ret) ; // Send an event to VMK in order
                                       // to request the task start
    wait_coderes(USTART, -1)         ; // Unschedule until USTART
    del_uroute(&start_rteid, 1)      ; // Delete USTART route

/*************************************************************************
 * Step 3 : Send a UPOKE event.                                          *
 * ------                                                                *
 *************************************************************************/

    loop:
    route_uevent(UPOKE, 0, HNULL, 0, poke_id++, 0, &ret) ;

/*************************************************************************
 * Step 4 : Wait for 2 seconds, then loop to step 3.                     *
 * ------                                                                *
 *************************************************************************/

    waitevt_task(HNULL    ,            // Address of waiting list
                 0        ,            // Size of "waitlist[]"
                 2000     ,            // Wait 2000ms
                 0        ,            // Do not purge previous events
                 &ret       )        ; // Return code

//  sendevt_task( myapp_taskid, PROC, &ret );

    goto loop ;

    return 0                         ;
  }

/*  Procedure wait_coderes -----------------------------------------------

    Purpose : Unschedule until one specific event is received. If non awaited
              events are received meanwhile, they will be kept inside the task
              events waiting queue.
*/

static void wait_coderes(int code, int res)
  {
    int             ret             ; // Return code
    int             waitlist[1][3]  ; // Parameter of "waitevt_task
```

```
/*****************************************************************************
 * Step 1 : We build a list of a single pair (code,reserve). The task will   *
 * ------   be unschedule until an event corresponding to the one expected   *
 *          is received by the task. This wait may be infinite if the event  *
 *          is never received.                                               *
 *****************************************************************************/

    waitlist[0][0]  = WAIT_CODERES   ; // Test "code" and "reserve" fields
    waitlist[0][1]  = code           ; // The awaited value for "code"
    waitlist[0][2]  = res            ; // The awaited value for "reserve"

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


/*****************************************************************************/
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
              128          ,           // 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
              128          ,           // 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
              128          ,           // 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 BUT4 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
  }

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


/*  Procedure open_kbd ---------------------------------------------------

*/

static void open_kbd(void)
  {

    int             ret              ; // Procedures returned code

    myio_open(DRVKBD,DEV0,"",&kbd_iod)  ; // Opens KBDITF\DEV0

    get_rcu_id(1          ,              // 0:Receiver 1:Transmitter
               0          ,              // numdev = the first one
               &rcusnd_id ,              // Corresponding id
               &ret        )           ; // Error code

    get_rcu_id(0          ,              // 0:Receiver 1:Transmitter
               0          ,              // numdev = the first one
               &rcurcv_id ,              // Corresponding id
               &ret        )           ; // Error code

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

  }

/*  Procedure close_kbd ----------------------------------------------------*/


static void close_kbd(void)
  {

    int             ret              ; // Procedures returned code

    myio_close(kbd_iod)              ; // Closes KBDITF\DEV0

    free_buf(buf_rcu   , &ret )      ; // Free "buf_rcu"

  }

/*************************************************************************/
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 :
          snd_trace("RED LED ",asy_iod2,1);
          ret = myio_setval(led_iod0, "CMD=POPALL" );
          if (state == 1)
            ret = myio_setval(led_iod0, "PATTERN=ON:2,OFF:8\nCMD=PUSH")  ;
          else
            ret = myio_setval(led_iod0, "PATTERN=OFF:0\nCMD=PUSH"     )  ;
        break;

        case 1 :
          snd_trace("GREEN LED ",asy_iod2,1);
          ret = myio_setval(led_iod1, "CMD=POPALL" );
          if (state == 1)
            ret = myio_setval(led_iod1, "PATTERN=ON:2,OFF:8\nCMD=PUSH")  ;
          else
            ret = myio_setval(led_iod1, "PATTERN=OFF:0\nCMD=PUSH"     )  ;
        break;

        case 2 :
          snd_trace("YELLOW LED ",asy_iod2,1);
          ret = myio_setval(led_iod2, "CMD=POPALL" );
          if (state == 1)
            ret = myio_setval(led_iod2, "PATTERN=ON:2,OFF:8\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
              ---------------  --------       ---------------
              RESP_READ        asy_iod0   ->  EV_ASY0_RCV   0
              RESP_READ        asy_iod1   ->  EV_ASY1_RCV   1
              RESP_READ        asy_iod2   ->  EV_ASY2_RCV   2
              RESP_WRITE       asy_iod0   ->  EV_ASY0_SND   3
              RESP_WRITE       asy_iod1   ->  EV_ASY1_SND   4
              RESP_WRITE       asy_iod2   ->  EV_ASY2_SND   5
              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[9][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]  = RESP_READ      ; // an event code RESP_READ
```

```
   waitlist[0][2]  = -1            ; // whatever the value of "reserve" is

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

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

   waitlist[3][0]  = WAIT_CODERES  ; // All events with
   waitlist[3][1]  = DRV_KEY_RELEASE; // an event code DRV_KEY_RELEASE
   waitlist[3][2]  = -1            ; // whatever the value of "reserve" is

   waitlist[4][0]  = WAIT_CODERES  ; // All events with
   waitlist[4][1]  = EVTS_I2C_END  ; // an event code EVTS_I2C_END
   waitlist[4][2]  = -1            ; // whatever the value of "reserve" is

   waitlist[5][0]  = WAIT_CODERES  ; // All events with
   waitlist[5][1]  = TICK          ; // an event code TICK
   waitlist[5][2]  = -1            ; // whatever the value of "reserve" is

   waitlist[6][0]  = WAIT_CODERES  ; // All events with
   waitlist[6][1]  = PROC          ; // an event code TICK
   waitlist[6][2]  = -1            ; // whatever the value of "reserve" is

   waitlist[7][0]  = WAIT_CODERES  ; // All events with
   waitlist[7][1]  = CONS          ; // an event code TICK
   waitlist[7][2]  = -1            ; // whatever the value of "reserve" is

   waitevt_task(waitlist ,           // Address of waiting list
                8        ,           // 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, we can have RESP_READ *
 *          RESP_WRITE, IND_REPORT, EVTS_I2C_END, DRV_KEY_PRESS,             *
 *          DRV_KEY_RELEASE but also the VMK generated event TICK (every     *
 *          second). According to the value of "task_evt.code" but also      *
 *          "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 RESP_READ              : // For a RESP_READ, the "reserve"
         if      (res EQ asy_iod0)   // field is the "iod" value. So
           ret = ev_asy_rcv(0)     ; // we compare it to our IOD's,
         else if (res EQ asy_iod1)   // that are "asy_iod0" for ASY\DEV0
           ret = ev_asy_rcv(1)     ; // "asy_iod1" for ASY\DEV1 and then
         else if (res EQ asy_iod2)   // "asy_iod2" for ASY\DEV2
           ret = ev_asy_rcv(2)     ; //
         break                     ; //

       case RESP_WRITE             : // For a RESP_WRITE, the "reserve"
         if      (res EQ asy_iod0)   // field is the "iod" value. So
           ret = ev_asy_snd(0)     ; // we compare it to our IOD's,
         else if (res EQ asy_iod1)   // that are "asy_iod0" for ASY\DEV0,
           ret = ev_asy_snd(1)     ; // "asy_iod1" for ASY\DEV1 and
```

```
          else if (res EQ asy_iod2)    // then
            ret = ev_asy_snd(2)      ; // "asy_iod2" for ASY\DEV2
          break                      ; //

        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 EVTS_I2C_END            : // For a EVTS_I2C_END, the "reserve"
          ret = EV_I2C_END           ; // field is a copy of the one received
          break                      ; // and we don't care.

        case TICK                    : // For a TICK, the "reserve"
//        snd_trace("TIC......",asy_iod2,0);
          check_time_out ()          ;
          break                      ; // and we don't care.

        case PROC                    : // For a inter task evt
          snd_trace("PROC..........",asy_iod1,1);
          break                      ; // and we don't care.

        case CONS                    : // For a inter task evt
          snd_trace(message,asy_iod1,1);
          break                      ; // and we don't care.

        case DRV_KEY_PRESS           : // For a DRV_KEY_PRESS, the "reserve"
          ret = ev_kbd_rcv()         ; // field is the code of the key that
          break                      ; // has been pressed.

        case DRV_KEY_RELEASE         : // For a DRV_KEY_RELEASE, the "reserve"
          ret = ev_kbd_rcv()         ; // field is the code of the key that
          break                      ; // has been released.

        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,int blk)
  {

    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
    int             ret              ;

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

```
            &ms   )                  ; // Milliseconds (0..999)

    if ( iod == asy_iod0 )             // be carefull about the buffer to use
      snd = buf_snd[0]               ; // Buffer to be sent
    if ( iod == asy_iod1 )             // be carefull about the buffer to use
      snd = buf_snd[1]               ; // Buffer to be sent
    if ( iod == asy_iod2 )             // be carefull about the buffer to use
      snd = buf_snd[2]               ; // Buffer to be sent

    if (blk == 1 )                     // Bloking mode
      {
       hsprintf((char*)snd      ,        // Put in the transmit buffer
             "%02d:%02d:%02d:%03d <%s>\n" ,// 8 characters HH:MM:SS:MS
             h, mi, s ,ms, 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
      }
    else                               // non blocking
      if ( sem_send  EQ 1 )            // nothing pending ?
      {
      hsprintf((char*)snd      ,       // Put in the transmit buffer
             "%02d:%03d <%s>\n" ,// 8 characters HH:MM:SS:MS
              s ,ms, mes      ); //
      lg = strlen((char*)snd)         ; // Number of characters to send
      sem_send  = 0                   ; // yes you can send a buff
      myio_send   (iod      ,          // I/O descriptor for serial line
                   snd      ,          // Address of buffer
                   lg       ,          // Number of bytes to send
                   &ret       );
      }
      else
       {
       trap_evt() ;                    // to trap this event in the debugger
#ifdef BIGTRACE
       hsprintf((char*)bigtrace,"%s\n%s",bigtrace,mes);
#endif
       }
  }

/*  Procedure ev_asy_rcv ----------------------------------------------------*/

static int ev_asy_rcv(int n)
  {
    unsigned char  *buf              ; // Buffer address
    int             len              ; // Data length
    int             err              ; // Return code
    int             vasy             ; // iod

    int             val              ; // in roder to read a value

    buf = task_evt.adresse           ; // Read buffer address from event
    len = task_evt.longueur          ; // Read data length from event

    if ( n == 0 ) vasy = asy_iod0    ; //  iod0
    if ( n == 1 ) vasy = asy_iod1    ; //  iod0
    if ( n == 2 ) vasy = asy_iod2    ; //  iod0

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


    if (buf EQ HNULL)                  // If no buffer received,
      goto end                       ; //   exit the procedure.
```

```
    myio_write  (vasy       ,          // I/O descriptor
                 buf        ,          // Address of buffer
                 len          )      ; // Number of bytes to send

    *(buf+len) = 0                    ; // String must be nul terminated
    val = hsscanf((HYPER_CHAR*)buf,"%d",&showvl)   ; // try to read a value
    if (val NE -1 )                   ; // yes we got an integer
      {                                 // showvl between 0 ans VLNB -1
        showvl = (showvl GE VLNB )? VLNB-1 :showvl;
        showvl = (showvl LT 0    )? 0      :showvl;
      }
    free_buf(buf, &err)               ; // Free the buffer.

    end                               : // Early exit

    return OPT_IGNO                   ; // Ignore these events
  }


/*  Procedure ev_asy_snd   ----------------------------------------------------

    Purpose : Select the treatment when EV_ASY1_SND/EV_ASY2_SND event is
              received. The "n" input parameter is 1 for ASY1 and 2 for ASY2
*/

static int ev_asy_snd(int n)
  {

    unsigned char  *snd               ; // Buffer address
    int             ret               ; // Return code

    snd = task_evt.adresse            ; // Get sent buffer address.
    sem_send  = 1                     ; // yes you can send a buff

#ifdef BIGTRACE
    if (strlen(bigtrace))
    {
      snd_trace(bigtrace,asy_iod2,0);
      memset(bigtrace,0,sizeof(bigtrace));
    }
#endif
    if ( snd NE buf_snd[n] )            // If an "echo" buffer has been sent,
      free_buf(snd, &ret)             ; // free it.
    else
      memset(snd,sizeof(snd),0)       ; // RAZ

    return OPT_IGNO                   ; // Ignore these events
  }


/*  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)
  {

    S_evt           evt               ; // SND_START response event
    int ret ;
    int  i  ;

    if (task_evt.longueur NE 0) goto ignore;  // If INPUT has been released

    if (n EQ 4 )                      // the USR button have been pressed
     {
       hsled(0,0)                   ; // OFF
```

```
       hsled(1,0)                  ; // OFF
       hsled(2,0)                  ; // OFF
       speed = TEMPO               ; // Default value is the original tempo
       play  = 1                   ; // Default value is to play
       for (i=0;i LT VLNB; i ++ )
        {
          state_usnd[i]     = 0 ;    // back to initial states
          tab_to  [i   ][0] = 0 ;    // No event for the time out
          tab_to  [i   ][1] = 0 ;    // No event for the time out
        }
         evt.aut      = AUT_USND   ; // Target automaton number
         evt.code     = SND_START  ; // Event code
         evt.vl       = 0          ; // Flag the target logical way
         evt.reserve  = 1          ; // Flag the target logical way
         evt.res2     = numfa      ; // The good waiting entry point
         putevt_vmk(NFA_STD_PS ,     // VMK Internal queue 2 to 15
                 &evt        )     ; // Event to be written
      }
    else
      ret= hsled(n-1,1);

    ignore:
    return ret                  ; // is it OK ?

  }


/*  Procedure ev_kbd_rcv ----------------------------------------------------

*/

static int ev_kbd_rcv(void)
  {

    char  trace[60];                 // just for a trace



    if (task_evt.code                  // We just ignore all the "release key"
          EQ DRV_KEY_RELEASE)          // events. We return the OPT_IGNO
      return OPT_IGNO               ; // code.

    switch (task_evt.reserve)
      {
        case KEY_0                  : // Digit keys, 0 to 9
        case KEY_1                  :
        case KEY_2                  :
        case KEY_3                  :
        case KEY_4                  :
        case KEY_5                  :
        case KEY_6                  :
        case KEY_7                  :
        case KEY_8                  :
        case KEY_9                  :
           hsprintf((char*)trace ,"AFFICHAGE VL:%03d " ,
                     task_evt.reserve-48               );
           snd_trace(trace,asy_iod2,1);
           return (showvl = task_evt.reserve-48 );

        case KEY_ENTER              : // OK key
          trap_evt  ()              ;
          return OPT_ENTER          ;

        case KB_PAUSE               : // PAUSE key
          play = 0                  ; // Old on
          snd_trace("PAUSE.....",asy_iod2,1);
          return KB_PAUSE           ;
```

```
        case KB_PLAY                : // PLAY  key
          play = 1                  ; // resume
          snd_trace("PLAY......",asy_iod2,1);
          return KB_PLAY            ;

        case KB_VOLP                : // VOL PLUS  key
          (speed GT BEAT)?speed-=BEAT:0; // lower and lower between 2 events
          snd_trace("SPEED +",asy_iod2,1);
          return OPT_ESC            ;

        case KB_VOLM                : // VOL MINIS  key
          speed += BEAT              ; // lower and lower between 2 events
          snd_trace("SPEED -",asy_iod2,1);
          return OPT_ESC            ;

        case KEY_F1                 : // RED key
          hsled(0,0)                 ; // OFF
          return OPT_BUT1           ;

        case KEY_F2                 : // GREEN KEY
          hsled(1,0)                 ; // OFF
          return OPT_BUT2           ;

        case KEY_F3                 : // YELLOW key
          hsled(2,0)                 ; // OFF
          return OPT_BUT3           ;

        case KEY_F4                 : // BLUE key
          return OPT_BUT4           ;

        case KEY_BACK               : // release the showvl
          showvl = -1               ;
          return OPT_CAPT           ;

        default                     :
          return OPT_INVALID        ;
      }
  }


/*  Procedure usnd_launch-----------------------------------------------------

    Purpose : RES_LAUNCH event treatment.

    par is the parameter idincating from where we have been luanched
        1 : Request to start from an other logical REQ_LAUNCH
        0 : Response from a logical way ok started
*/
static int usnd_launch (int par)
  {

    int             ret             ;
    S_evt           evt             ; // SND_START response event
    int             ufa             ;
    char            trace[60]       ; // just for a trace

    Memset(&evt,0,sizeof(evt))      ; // Clear the event structure
    evt.aut      = AUT_USND         ; // Target automaton number
    evt.code     = SND_START        ; // Event code

    ufa    = (par EQ 0 ) ? NFA_STD_PS : numfa ;
    evt.vl = (par EQ 0 ) ? 1 : voielog+1 ;

    if ( evt.vl EQ VLNB-1 ) goto stepout; // we are on the roof !!

    set_to(TIMER          ,         // One shot timer
           50             ,         // Timer duration in milliseconds
           ufa             ,        // Wait queue for the expiration event
```

```
            SND_START        ,          // End of period notification event code
            0                ,          // Reserve field for notification event
            AUT_USND         ,          // FSM number for notification event
            evt.vl           ,          // Logical way for notification event
            &ret                )    ; // Returned time-out identifier

    stepout:

    hsprintf((char*)trace ,
      "usnd_launch PAR:%02d  -> VLT:%02d EVT:%X" , par,evt.vl,ret);
    if ( par NE 0 ) snd_trace(trace,asy_iod2,0);

    return 0                         ; // Exit without any error

  }

/*  Procedure usnd_start ----------------------------------------------------

*/

static int usnd_start(int par)
  {

    char  trace[60];                  // just for a trace
    int   ret      ;
    S_evt evt      ;                  // SND_START response event

    if (voielog EQ 0 )
      {
        hsprintf((char*)trace , "START usnd_start VL:%0d NFA:%d " ,
                  voielog,evt_usnd.res2);
        snd_trace(trace,asy_iod2,1);
      }
    usnd_state_led(voielog)          ; // show led states

    tab_to[evt_usnd.vl][0] = R2Y ;       // Next time out
    tab_to[evt_usnd.vl][1] = speed+50*voielog;      // Next time out

    if ((voielog ) EQ VLNB-1 )
      {
        sendevt_task( myapp_taskid, TICK, &ret );
        hsprintf((char*)trace , "END  usnd_start VL:%0d NFA:%d RET:%X " ,
                  voielog,evt_usnd.res2,ret);
        snd_trace(trace,asy_iod2,0);
        goto roof ;                     // No need to move forward
      }
    Memset(&evt,0,sizeof(evt))       ; // Clear the event structure
    evt.aut      = AUT_USND          ; // Target automaton number
    evt.vl       = voielog + 1       ; // Target Logical way number
    evt.code     = SND_START         ; // Event code OK started
    evt.reserve  = voielog + 1       ; // Target Logical way number
    evt.res2     = numfa             ; // The good waiting entry point

    ret = putevt_vmk(evt_usnd.res2,    // VMK Internal queue 2 to 15
                &evt        )       ; // Event to be written
    if (ret) trap_evt();               // in order to catch issues

    roof:                              // We are on the top
    return 0                         ; // Exit without any error
  }

/*  Procedure usnd_r2y   ------------------------------------------------- */

static int usnd_r2y  (int par)
  {

    char  trace[60];                  // just for a trace
```

```
    if (par)                          // With or without trace ?
    {
      hsprintf((char*)trace , "usnd_r2y VL:%02d |20 " ,evt_usnd.vl);
      if ((evt_usnd.vl % 20) EQ 0 )   // no to much traces
        snd_trace(trace,asy_iod2,0);  // only each 20 logical way
    }

    usnd_state_led(voielog) ;// show led states

    tab_to[evt_usnd.vl][0] = Y2G ;       // Next event
    tab_to[evt_usnd.vl][1] = speed;      // Next time out

    return 0                         ; // Exit without any error
  }

/*  Procedure usnd_y2r   ------------------------------------------------- */

static int usnd_y2r  (int par)
  {

    char  trace[60];                  // just for a trace

    if (par)                          // With or without trace ?
    {
      hsprintf((char*)trace , "usnd_y2r VL:%02d " ,evt_usnd.vl);
      snd_trace(trace,asy_iod2,0);
    }

    usnd_state_led(voielog)          ;   // show led states

    tab_to[evt_usnd.vl][0] = R2Y ;       // Next time out
    tab_to[evt_usnd.vl][1] = speed;      // Next time out

    return 0                         ; // Exit without any error
  }

/*  Procedure usnd_g2y   -----------------------------------------------------

*/
static int usnd_g2y  (int par)
  {

    char  trace[60];                  // just for a trace

    if (par)                          // With or without trace ?
    {
      hsprintf((char*)trace , "usnd_g2y VL:%02d " ,evt_usnd.vl);
      snd_trace(trace,asy_iod2,0);
    }

    usnd_state_led(voielog)         ;   //  show led states

    tab_to[evt_usnd.vl][0] = RED ;       // Next time out
    tab_to[evt_usnd.vl][1] = speed;      // Next time out

    return 0                         ; // Exit without any error
  }


/*  Procedure usnd_y2g   ------------------------------------------------- */

static int usnd_y2g  (int par)
  {

    char  trace[60];                  // just for a trace

    if (par)                          // With or without trace ?
    {
```

```
      hsprintf((char*)trace , "usnd_y2g VL:%02d " ,evt_usnd.vl);
      snd_trace(trace,asy_iod2,0);
    }

    usnd_state_led(voielog)          ;   // show led states

    tab_to[evt_usnd.vl][0] = G2Y ;       // Next time out
    tab_to[evt_usnd.vl][1] = speed;      // Next time out

    return 0                         ; // Exit without any error
  }

/*  Procedure check_time_out------------------------------------------------ */

static int check_time_out ()
  {

    int i                            ;
    int ret                          ;
    S_evt           evt              ; // SND_START response event


    clear_to (idto,numauto,voielog);     // first stop the timer while proceeding
    if ( play EQ 0 ) goto pause    ;     // Pause requested do nothing
    for ( i=0 ; i LT VLNB ; i ++ )       // Timers reset
     {
       if ((tab_to[i][1] LE BEAT ) LOGAND // near to the end
           (tab_to[i][1] GT 0  )  )      // Go one timer to launch
         {
           Memset(&evt,0,sizeof(evt))  ; // Clear the event structure
           evt.aut      = AUT_USND     ; // Target automaton number
           evt.vl       = i            ; // Target Logical way number
           evt.code     = tab_to[i][0] ; // Event code
           ret = putevt_vmk(NFA_STD_PS , // VMK Internal queue 2 to 15
                 &evt        )         ; // Event to be written
           suspend_task (0,&ret)       ; // Please proceed
           tab_to[i][0] = 0            ; // reset the tempo
           tab_to[i][1] = 0            ; // reset the event
         }
       else
         if (tab_to[i][1] GT BEAT )      // far from to the end
          tab_to[i][1] -= BEAT    ;      // decrease
     }

    pause:                              // next please
                                        // don't forget to relaunch the timer
    set_tto(TIMER      ,                // Timer mode: once
            BEAT       ,                // Duration in milliseconds
            TICK       ,                // Event code
            0          ,                // Event reserve field
            &idto        )          ;  // Timer identifier

    return 0                         ; // Exit without any error
  }

/*  Procedure usnd_state_led   ---------------------------------------------- */

static int usnd_state_led  (vl)
  {

    if (showvl LT 0 LOGOR              // shall we show this logical way ?
        showvl EQ vl        )          // 2 conditions to check
    switch (state_usnd[vl])            // Selecte states led
      {
        case 0                       : // the begining of the state
          usnd_pop   (led_iod0)                          ;
          myio_setval(led_iod0, "PATTERN=ON:0\nCMD=PUSH"     )  ;
          break                      ;
```

```
        case 1                      : // Just the red one is on
          usnd_pop   (led_iod0)                          ;
          usnd_pop   (led_iod2)                          ;
          myio_setval(led_iod0, "PATTERN=ON:0\nCMD=PUSH"     )  ;
          myio_setval(led_iod2, "PATTERN=ON:2,OFF:8\nCMD=PUSH")  ;
          break                     ;

        case 2                      : // red is on and the yellow blinking
          usnd_pop   (led_iod0)                          ;
          usnd_pop   (led_iod2)                          ;
          usnd_pop   (led_iod1)                          ;
          myio_setval(led_iod0, "PATTERN=OFF:0\nCMD=PUSH")  ;
          myio_setval(led_iod2, "PATTERN=OFF:0\nCMD=PUSH")  ;
          myio_setval(led_iod1, "PATTERN=ON:0\nCMD=PUSH")  ;
          break                     ;

        case 3                      : // Just the gree one is on
          usnd_pop   (led_iod0)                          ;
          usnd_pop   (led_iod2)                          ;
          usnd_pop   (led_iod1)                          ;
          myio_setval(led_iod2, "PATTERN=OFF:0\nCMD=PUSH")  ;
          myio_setval(led_iod1, "PATTERN=OFF:0\nCMD=PUSH")  ;
          myio_setval(led_iod0, "PATTERN=ON:0\nCMD=PUSH")  ;
          break                     ;

        case 4                      :
          usnd_pop   (led_iod2)                          ;
          myio_setval(led_iod2, "PATTERN=ON:2,OFF:8\nCMD=PUSH")  ;
          break                     ;
      }

    return 0                        ; // Exit without any error
  }


/*  Procedure usnd_pop   ---------------------------------------------------- */

static int usnd_pop  (led)
  {

    myio_setval(led, "CMD=POPALL" );

    return 0                        ; // Exit without any error
  }



/*  Procedure trap_snd   ---------------------------------------------------- */

static int  trap_snd  ()
  {


    return 0                        ; // Exit without any error
  }

/*  Procedure trap_evt   ---------------------------------------------------- */

static int  trap_evt  ()
  {

    int i = 0;
    return i                        ; // Exit without any error
  }
```