```
/*****************************************************************************/
/*                                                                           */
/*                                  KEYS.S                                   */
/*                               Key handlers                                */
/*                        Digital Oscilloscope Project                       */
/*                                 EE/CS 52                                  */
/*                              Santiago Navonne                             */
/*                                                                           */
/*****************************************************************************/

/*
   Key and rotary encoder control routines for the EE/CS 52 Digital Oscilloscope
   project. Function definitions are included in this file, and are laid out
   as follows:
    - keys_init: Initializes the key handler's shared variables, and enables
                 interrupts from the required sources, effectively preparing
                 the user input section for use;
    - keys_handler: Handles key press (and rotary encoder turn) interrupts;
    - getkey: Returns the currently pending user action, blocking if none is
              available.
    - key_available: Checks whether a user action is currently pending.


   Revision History:
      5/7/14  Santiago Navonne  Initial revision.
      5/14/14 Santiago Navonne  Added additional documentation.
      6/7/14  Santiago Navonne  Changed up/down rotation direction.
*/

/* Includes */
#include "general.h"  /* General constants */
#include "system.h"   /* Base addresses */
#include "interfac.h" /* Software interface definitions */
#include "keys.h"     /* Local constants */


/* Variables */
    .section .data  /* No alignment necessary: variables are bytes */
curr_key: .byte 0   /* Current pending key; 0 if no key available */

    .section .text  /* Code starts here */

/*
 *  keys_init
 *
 *  Description:       This procedure initializes the internal state of the key/
 *                     user input handling system, preparing any shared variables
 *                     for use and configuring interrupts. This function should be
 *                     called in order to start accepting user input.
 *
 *  Operation:         This procedure initializes any shared variables to their
 *                     default states:
 *                      - curr_key: value of the currently pending key (default: 0).
 *                     Additionally, the function registers the key press handler
 *                     as the default interrupt handler for key presses using the HAL
 *                     API alt_ic_isr_register, and finally unmasks all interrupts by
 *                     writing to the corresponding PIO register.
 *
 *  Arguments:         None.
 *
 *  Return Value:      None.
 *
 *  Local Variables:   None.
 *
 *  Shared Variables:  - curr_key (write only).
 *
 *  Global Variables:  None.
 *
 *  Input:             None.
 *
 *  Output:            None.
 *
 *  Error Handling:    None.
 *
 *  Limitations:       None.
```

```
 *
 *  Algorithms:        None.
 *  Data Structures:   None.
 *
 *  Registers Changed: r4, r5, r6, r7, r8, r9.
 *
 *  Revision History:
 *      5/7/14    Santiago Navonne     Initial revision.
 *      5/14/14   Santiago Navonne     Added additional documentation.
 *
 */
    .global keys_init
keys_init:
    ADDI    sp, sp, NEG_WORD_SIZE  /* push return address */
    STW     ra, (sp)

    MOVIA   r9, curr_key           /* no key (r0) available at start */
    STB     r0, (r9)               /* so store it into variable curr_key */

    MOVHI   r8, %hi(PIO_0_BASE)    /* write to the PIO registers */
    ORI     r8, r8, %lo(PIO_0_BASE)
    MOVI    r9, ENABLE_ALL         /*  the ENABLE_ALL value */
    STBIO   r9, EDGE_CAP_OF(r8)    /* sending general EOI to clear ints */

    MOV     r4, r0                 /* argument ic_id is ignored */
    MOVI    r5, PIO_0_IRQ          /* second arg is IRQ num */
    MOVIA   r6, keys_handler       /* third arg is int handler */
    MOV     r7, r0                 /* fourth arg is data struct (null) */
    ADDI    sp, sp, NEG_WORD_SIZE  /* fifth arg goes on stack */
    STW     r0, (sp)               /*  and is ignored (so 0) */
    CALL    alt_ic_isr_register    /* finally, call setup function */
    ADDI    sp, sp, WORD_SIZE      /* clean up stack after call */

    LDW     ra, (sp)               /* pop return address */
    ADDI    sp, sp, WORD_SIZE

    STBIO   r9, INTMASK_OF(r8)     /* enable (unmask) interrupts */

    RET                            /* and finally return */



/*
 *  keys_handler
 *
 *  Description:       This procedure handles hardware interrupts generated by
 *                     key presses and rotary encoder steps. Every time one of
 *                     these fires, the shared variable containing the currently
 *                     pending key is updated to indicate a key press. Note that
 *                     previously pending key presses are overwritten by this
 *                     function.
 *                     The function is designed to support only one key press
 *                     at a time; its behavior in the event of simultaneous key
 *                     presses is undefined.
 *
 *  Operation:         When called, the function first reads the edge capture
 *                     register of the user input PIO interface to figure out
 *                     which interrupt fired. It compares the read value to all
 *                     the known constants, translating it into a key ID. Unknown
 *                     values, which are caused by simultaneous key presses,
 *                     are handled in the else case.
 *                     After the key press is decoded, the identification code is
 *                     saved to the shared variable curr_key.
 *                     Note that the procedure uses multiple comparisons and not
 *                     a jump table in order to save space; furthermore, the
 *                     interrupt register value is not simply used as a key
 *                     identifier to prevent simultaneous key presses from
 *                     breaking the system.
 *
 *  Arguments:         None.
 *
 *  Return Value:      None.
 *
 *  Local Variables:   None.
 *
```

```
 *  Shared Variables:  - curr_key: currently pending key press code (read/write).
 *
 *  Global Variables:  None.
 *
 *  Input:             Key presses and rotary encoder turns from the user interface.
 *
 *  Output:            None.
 *
 *  Error Handling:    If multiple keys are pressed at once, the function's
 *                     behavior is undefined.
 *
 *  Limitations:       Only one simultaneous key press is accepted.
 *                     Any previously recognized but not yet polled key presses
 *                     are lost (overwritten) when a new event is received.
 *
 *  Algorithms:        None.
 *  Data Structures:   None.
 *
 *  Registers Changed: et.
 *
 *  Revision History:
 *      5/7/14    Santiago Navonne     Initial revision.
 *      5/14/14   Santiago Navonne     Added additional documentation.
 *
 */
    .global keys_handler
keys_handler:
    ADDI    sp, sp, NEG_WORD_SIZE   /* save r8 */
    STW     r8, (sp)

    MOVHI   et, %hi(PIO_0_BASE)  /* fetch PIO edge capture register */
    ORI     et, et, %lo(PIO_0_BASE)
    LDBIO   r8, EDGE_CAP_OF(et)

    STBIO   r8, EDGE_CAP_OF(et)  /* and write back to send EOI */
                                 /* figure out what interrupt fired */
    MOVI    et, PUSH1_MASK       /* check if it was pushbutton 1 */
    BEQ     r8, et, keys_handler_push1
    MOVI    et, PUSH2_MASK       /* check if it was pushbutton 2 */
    BEQ     r8, et, keys_handler_push2
    MOVI    et, ROT1R_MASK       /* check if it was rotary enc 1 right */
    BEQ     r8, et, keys_handler_rot1r
    MOVI    et, ROT1L_MASK       /* check if it was rotary enc 1 left */
    BEQ     r8, et, keys_handler_rot1l
    MOVI    et, ROT2R_MASK       /* check if it was rotary enc 2 right */
    BEQ     r8, et, keys_handler_rot2r
    JMPI    keys_handler_rot2l    /* else it must be rotary enc 2 left */

keys_handler_push1:               /* handle pushbutton 1 ints */
    MOVI    et, KEY_MENU          /*  translates into menu key */
    JMPI    keys_handler_done

keys_handler_push2:               /* handle pushbutton 2 ints */
    MOVI    et, KEY_MENU          /*  translates into menu key */
    JMPI    keys_handler_done

keys_handler_rot1r:               /* handle rotary enc 1 right ints */
    MOVI    et, KEY_DOWN          /*  translates into down key */
    JMPI    keys_handler_done

keys_handler_rot1l:               /* handle rotary enc 1 left ints */
    MOVI    et, KEY_UP            /*  translates into up key */
    JMPI    keys_handler_done

keys_handler_rot2r:               /* handle rotary enc 2 right ints */
    MOVI    et, KEY_RIGHT         /*  translates into right key */
    JMPI    keys_handler_done

keys_handler_rot2l:               /* handle rotary enc 2 left ints */
    MOVI    et, KEY_LEFT           /*  translates into left key */
    JMPI    keys_handler_done

keys_handler_done:                /* handling completed */
    MOVIA   r8, curr_key          /* save to curr_key */
    STB     et, (r8)              /*  the processed key */
```

```
226
227     LDW     r8, (sp)              /* restore r8 */
228     ADDI    sp, sp, WORD_SIZE
229     RET                           /* all done */
230
231
232
233 /*
234  *  getkey
235  *
236  *  Description:       This procedure returns the identifier of the last pressed,
237  *                     unpolled key, as described in interfac.h.
238  *                     If no key press is pending, the function blocks.
239  *                     (To ensure non-blocking behavior, getkey calls should be
240  *                     preceded by key_available calls.)
241  *
242  *  Operation:         The function first fetches the value stored in curr_key and
243  *                     compares it to 0, which would indicate that there isn't
244  *                     actually any pending key press. In no key press is pending,
245  *                     the function keeps fetching the value until it is not 0.
246  *                     When the value is not 0, the function clears the value of
247  *                     curr_key (to delete the now reported press) and returns
248  *                     the retrieved value.
249  *
250  *  Arguments:         None.
251  *
252  *  Return Value:      key (r2) - ID code of the pending key, as defined in
253  *                                interfac.h.
254  *
255  *  Local Variables:   None.
256  *
257  *  Shared Variables:  - curr_key: currently pending key press code (read/write).
258  *
259  *  Global Variables:  None.
260  *
261  *  Input:             None.
262  *
263  *  Output:            None.
264  *
265  *  Error Handling:    If no key is available, the funciton blocks until a key
266  *                     is pressed.
267  *
268  *  Limitations:       None.
269  *
270  *  Algorithms:        None.
271  *  Data Structures:   None.
272  *
273  *  Registers Changed: r2, r8.
274  *
275  *  Revision History:
276  *      5/7/14    Santiago Navonne     Initial revision.
277  *      5/14/14   Santiago Navonne     Added additional documentation.
278  *
279  */
280     .global getkey
281 getkey:
282     MOVIA   r8, curr_key      /* return current pending key */
283     LDB     r2, (r8)
284     BEQ     r0, r2, getkey    /* if there is no key (curr_key == r0), block */
285
286     STB     r0, (r8)          /* clear current key */
287     RET                       /* return with current pending key in r2 */
288
289
290
291 /*
292  *  key_available
293  *
294  *  Description:       This procedure checks whether a key has been pressed and
295  *                     is available for polling. The function returns true
296  *                     (non-zero) if there's a key available, and non-zero if no
297  *                     key has been pressed.
298  *                     This function should be called before using getkey to avoid
299  *                     blocking.
300  *
```

```
 *  Operation:          The function simply returns the value stored in the shared
 *                      variable curr_key, taking advantage of the fact that this
 *                      value is zero if no key is available, and non-zero otherwise.
 *
 *  Arguments:          None.
 *
 *  Return Value:       key_available (r2) - true (non-zero) if a key press is
 *                                           available, false (zero) otherwise.
 *
 *  Local Variables:    None.
 *
 *  Shared Variables:   - curr_key: currently pending key press code (read only).
 *
 *  Global Variables:   None.
 *
 *  Input:              Key presses and rotary encoder turns from the user interface.
 *
 *  Output:             None.
 *
 *  Error Handling:     None.
 *
 *  Limitations:        None.
 *
 *  Algorithms:         None.
 *  Data Structures:    None.
 *
 *  Registers Changed: r2, r8.
 *
 *  Revision History:
 *      5/7/14    Santiago Navonne     Initial revision.
 *      5/14/14   Santiago Navonne     Added additional documentation.
 *
 */
    .globl key_available
key_available:
    MOVIA   r8, curr_key        /* return current pending key */
    LDB     r2, (r8)            /* will be zero (FALSE) if no key is pending */

    RET                         /* return with boolean in r2 */
```