```
/*****************************************************************************/
/*                                                                           */
/*                                TRIGGER.S                                  */
/*                        Data sampling and triggering                       */
/*                        Digital Oscilloscope Project                       */
/*                                EE/CS 52                                   */
/*                              Santiago Navonne                             */
/*                                                                           */
/*****************************************************************************/

/*
   Data sampling and triggering control routines for the EE/CS 52 Digital
   Oscilloscope project. Function definitions are included in this file, and
   are laid out as follows:
    - set_sample_rate: Configures the sampling rate;
    - set_trigger: Configures the manual trigger level and slope;
    - set_delay: Configures the manual trigger delay;
    - start_sample: Starts a new data sample with the previously configured
                    settings and passed auto-trigger configuration;
    - sample_done: Checks whether a new data sample set is available, returning
                   a pointer to a buffer containing it if there is, or a NULL
                   pointer if there isn't;
    - sample_handler: Handles sampling FIFO full interrupts;
    - trigger_init: Initializes the environment's shared variables and the
                    triggering logic circuit (resetting it), effectively
                    preparing the sampling/triggering interface for use.


   Revision History:
      5/29/14 Santiago Navonne  Initial revision.
      6/01/14 Santiago Navonne  Minor fixes; updated documentation.
      6/11/14 Santiago Navonne  Changed division algorithm in set_sample_rate.
*/

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


/* Variables */
    .section .data        /* No alignment necessary: variables are bytes */
sample_pending: .byte 0   /* Logical value: whether a sample is pending */
sample: .skip FIFO_SIZE   /* Sample buffer */

    .section .text        /* Code starts here */

/*
 *  set_sample_rate
 *
 *  Description:       This procedure configures the sampling rate of the sampling
 *                     interface. After execution, the interface will start sampling
 *                     at the requested rate, rounded up to a multiple of the system
 *                     clock. The return value is how many samples will be acquired,
 *                     which is always the size of the FIFO.
 *                     If an argument of 0 is passed, the function has no effect, and
 *                     returns 0. The argument must however by less than or equal to
 *                     the system clock divided by two; no error checking is performed
 *                     on this.
 *
 *  Operation:         The procedure starts by error checking the value of the argument,
 *                     simply returning 0 if it is invalid. Then, it computes the
 *                     required clock period in system clock periods by dividing the
 *                     system clock frequency by the requested sample rate.
 *                     Finally, it saves the computed value to the trigger period
 *                     register, and pulses the reset bit in the control register to
 *                     reset the triggering logic. SIZE_X is ultimately moved into
 *                     r2 as constant return value.
 *
 *  Arguments:         samples_per_sec - positive integer indicating the sample rate
 *                                       in samples per second (r4). The value must
 *                                       be less than or equal to the system clock
 *                                       divided by two.
 *
```

```
 *  Return Value:      sample_num - positive integer, number of samples that will be
 *                                  acquired at the desired rate (r2).
 *
 *  Local Variables:   None.
 *
 *  Shared Variables:  None.
 *
 *  Global Variables:  None.
 *
 *  Input:             None.
 *
 *  Output:            None.
 *
 *  Error Handling:    If the argument is zero, the function has no effect, and returns 0.
 *                     No error checking is performed on the upper bound of the sampling
 *                     rate.
 *
 *  Limitations:       Resulting sample clock is an integer multiple of the system clock;
 *                     corresponding rate will be greater than or equal to the requested
 *                     rate, with a difference in period less than the system clock's.
 *                     Number of samples acquired must be <= FIFO_SIZE per hardware
 *                     limitations (size of FIFO).
 *
 *  Algorithms:        Division is performed using a repeated subtraction algorithm since
 *                     hardware division cannot be assumed to be available. This algorithm
 *                     is acceptable because generally very few iterations will be needed
 *                     to reach the result.
 *  Data Structures:   None.
 *
 *  Registers Changed: r2, r4, r8, r9.
 *
 *  Revision History:
 *      5/29/14   Santiago Navonne     Initial revision.
 *      6/01/14   Santiago Navonne     Added error checking, expanded documentation.
 *      6/11/14   Santiago Navonne     Changed hardware divide instruction to division
 *                                     by repeated subtraction.
 *
 */
    .global set_sample_rate
set_sample_rate:
    MOV     r2, r0                  /* load return value of 0 in case of error */
    BEQ     r4, r0, set_sample_rate_done /* error if argument is 0 */

    MOVHI   r8, %hi(CLK_FREQ)       /* load system clock frequency to */
    ORI     r8, r8, %lo(CLK_FREQ)   /*  find number of system clocks that takes */
    /*DIVU    r9, r8, r4              /*  by dividing the sys clk by the requested rate */
    XOR     r9, r9, r9              /* prepare register for division: r9 is quotient */

div_check:                          /* check if the divisor fits in the dividend */
    BLT     r8, r4, div_done        /* we're done when it doesn't any more */

div_loop:                           /* need to keep subtracting: */
    SUB     r8, r8, r4              /* subtract divisor from dividend */
    ADDI    r9, r9, 1               /* and increment quotient */
    JMPI    div_check               /* thus repeat as needed */

div_done:
    MOVHI   r8, %hi(TRIG_PERIOD_BASE)     /* load period data register address to */
    ORI     r8, r8, %lo(TRIG_PERIOD_BASE) /*  finally save result to trigger period */
    STWIO   r9, (r8)                /*  data, effectively setting the sample rate */

    MOVHI   r8, %hi(TRIG_CTRL_SET) /* load trigger control bit set reg address */
    ORI     r8, r8, %lo(TRIG_CTRL_SET)   /* to reset trigger logic */
    MOVI    r9, FIFO_RESET_BIT     /* by sending reset bit high */
    STWIO   r9, (r8)
    ADDI    r8, r8, WORD_SIZE      /* and then move to bit clr reg */
    STWIO   r9, (r8)               /* to send it low */

    MOVI    r2, SIZE_X             /* number of samples acquired is always size of display */

set_sample_rate_done:              /* all done */
    RET                            /* return value is in r2 */
```

```
/*
 *  set_trigger
 *
 *  Description:       This function configures the triggering settings on the sampling
 *                     interface. After execution, triggering will occur as soon as the
 *                     input passes the value of <level>, in the direction indicated by
 *                     <slope>. Note that these settings are only used when a sample is
 *                     started with manual triggering enabled.
 *
 *  Operation:         The procedure first "corrects" the level, mapping it to the
 *                     right range ([0, 255]) and adding any necessary calibration
 *                     constants.
 *                     Then, it writes the slope bit to either the trigger control set
 *                     or clear register, depending on what action needs to be performed,
 *                     followed by the corrected level argument to the trigger level
 *                     register.
 *                     Finally, the reset bit within the trigger control register is
 *                     pulsed to reset the triggering logic.
 *
 *  Arguments:         level - trigger level to be configured, as a value between 0 and
 *                             127, where 0 is the most negative level, and 127 is the
 *                             most positive level (r4).
 *                     slope - desired trigger slope; 1 for positive slope, 0 for
 *                             negative slope (r5).
 *
 *  Return Value:      None.
 *
 *  Local Variables:   None.
 *
 *  Shared Variables:  None.
 *
 *  Global Variables:  None.
 *
 *  Input:             None.
 *
 *  Output:            None.
 *
 *  Error Handling:    None.
 *
 *  Limitations:       None.
 *
 *  Algorithms:        None.
 *  Data Structures:   None.
 *
 *  Registers Changed: r4, r8, r9, r10.
 *
 *  Revision History:
 *      5/29/14   Santiago Navonne     Initial revision.
 *      6/01/14   Santiago Navonne     Expanded documentation.
 *
 */
    .global set_trigger
set_trigger:
    MOVHI   r10, %hi(TRIG_LEVEL_BASE) /* load trigger level register address to update */
    ORI     r10, r10, %lo(TRIG_LEVEL_BASE) /* the desired trigger level */
    MOVI    r9, TRIG_LEVEL_SHIFT   /* shift the passed argument left as needed to */
    SLL     r4, r4, r9             /*  make sure we output a full byte */
    SUBI    r4, r4, CALIBRATION    /* and correct value with calibration data */

    MOVHI   r8, %hi(TRIG_CTRL_CLR) /* load control register bit clear address to */
    ORI     r8, r8, %lo(TRIG_CTRL_CLR) /*  initially assume that we want to set  */
    MOVI    r9, 2                  /*  slope to negative (clear the bit) */
    SLL     r5, r5, r9             /* subtract argument multiplied by word size */
    SUB     r8, r8, r5             /*  effectively moving to set bit register if enabling */
                                   /*  positive slope */

    MOVI    r9, SLOPE_BIT          /* finally write the appropriate bit to the register */
    STWIO   r9, (r8)               /* enabling or disabling the bit as needed */

    STWIO   r4, (r10)              /* and output desired trigger level */

    MOVHI   r8, %hi(TRIG_CTRL_SET) /* load trigger control bit set reg address */
    ORI     r8, r8, %lo(TRIG_CTRL_SET)     /* to reset trigger logic */
    MOVI    r9, FIFO_RESET_BIT     /* by sending reset bit high */
    STWIO   r9, (r8)
```

```
    ADDI    r8, r8, WORD_SIZE      /* and then move to bit clr reg */
    STWIO   r9, (r8)               /* to send it low */

    RET                            /* all done, so return */


/*
 *  set_delay
 *
 *  Description:       This procedure configures the sampling delay on manual triggers.
 *                     After execution, triggering will occur <delay> samples after the
 *                     configured level and slope settings are satisfied. Note that this
 *                     setting is only used when manual triggering is enabled.
 *                     Also note that delay must be less than MAX_DELAY.
 *
 *  Operation:         The function first corrects the argument by adding the necessary
 *                     hardware constant to it, and then outputs it to the trigger
 *                     delay register.
 *                     Finally, the reset bit within the trigger control register is
 *                     pulsed to reset the triggering logic.
 *
 *  Arguments:         delay - unsigned integer <= MAX_DELAY; trigger delay from
 *                             trigger event in number of samples (r4).
 *
 *  Return Value:      None.
 *
 *  Local Variables:   None.
 *
 *  Shared Variables:  None.
 *
 *  Global Variables:  None.
 *
 *  Input:             None.
 *
 *  Output:            None.
 *
 *  Error Handling:    None.
 *
 *  Limitations:       Only positive delays less than or equal to MAX_DELAY are valid.
 *
 *  Algorithms:        None.
 *  Data Structures:   None.
 *
 *  Registers Changed: r4, r10.
 *
 *  Revision History:
 *      5/29/14   Santiago Navonne     Initial revision.
 *      6/01/14   Santiago Navonne     Expanded documentation.
 *
 */
    .global set_delay
set_delay:
    MOVHI   r10, %hi(TRIG_DELAY_BASE) /* load trigger delay register address to update */
    ORI     r10, r10, %lo(TRIG_DELAY_BASE) /* the desired delay time */
    ADDI    r4, r4, DELAY_CONSTANT    /* add delay constant to correct argument */
    STWIO   r4, (r10)                 /* and output to delay register, effectively */
                                      /* configuring delay */

    MOVHI   r8, %hi(TRIG_CTRL_SET)    /* load trigger control bit set reg address */
    ORI     r8, r8, %lo(TRIG_CTRL_SET)     /* to reset trigger logic */
    MOVI    r9, FIFO_RESET_BIT        /* by sending reset bit high */
    STWIO   r9, (r8)
    ADDI    r8, r8, WORD_SIZE         /* and then move to bit clr reg */
    STWIO   r9, (r8)                  /* to send it low */

    RET                               /* all done, so return */


/*
 *  start_sample
 *
 *  Description:       This procedure immediately starts sampling data. If the argument
 *                     is FALSE, sampling starts upon a trigger event. If the argument
 *                     is TRUE, sampling starts immediately.
 *                     Any previously started but incomplete samples are cancelled and
```

```
 *                      replaced.
 *
 *  Operation:          The procedure sets or clears the auto trigger bit in the trigger
 *                      control register to enable or disable auto triggering.
 *                      Finally, it starts the sample by enabling writing to the FIFO
 *                      through the write enable bit in the control register, and resets
 *                      the triggering logic.
 *
 *  Arguments:          auto_trigger - TRUE if sampling should be started
 *                                     automatically (i.e. as soon as possible),
 *                                     FALSE if it should be started on a trigger
 *                                     event (r4).
 *
 *  Return Value:       None.
 *
 *  Local Variables:    None.
 *
 *  Shared Variables:   None.
 *
 *  Global Variables:   None.
 *
 *  Input:              None.
 *
 *  Output:             None.
 *
 *  Error Handling:     None.
 *
 *  Limitations:        None.
 *
 *  Algorithms:         None.
 *  Data Structures:    None.
 *
 *  Registers Changed: r8, r9.
 *
 *  Revision History:
 *      5/29/14   Santiago Navonne     Initial revision.
 *      6/01/14   Santiago Navonne     Expanded documentation.
 *
 */
    .global start_sample
start_sample:

    MOVHI   r8, %hi(TRIG_CTRL_CLR) /* load trigger control bit clear reg address */
    ORI     r8, r8, %lo(TRIG_CTRL_CLR) /* assuming we'll clear auto trigger bit */
    MOVI    r9, 2                  /* subtract argument multiplied by word size */
    SLL     r4, r4, r9             /* effectively moving to set bit register if enabling */
    SUB     r8, r8, r4             /*  auto trigger*/

    MOVI    r9, AUTO_TRIG_BIT      /* store auto trigger bit in configured register */
    STWIO   r9, (r8)               /* enabling or disabling it as needed */

    MOVHI   r8, %hi(TRIG_CTRL_SET)    /* load trigger control bit set reg address */
    ORI     r8, r8, %lo(TRIG_CTRL_SET)     /* to reset trigger logic */
    MOVI    r9, FIFO_RESET_BIT        /* by sending reset bit high */
    STWIO   r9, (r8)
    ADDI    r8, r8, WORD_SIZE         /* and then move to bit clr reg */
    STWIO   r9, (r8)                  /* to send it low */

    MOVHI   r8, %hi(TRIG_CTRL_CLR) /* load trigger control bit clear reg address */
    ORI     r8, r8, %lo(TRIG_CTRL_CLR) /* to clear fifo write enable (make active) */
    MOVI    r9, FIFO_WE_BIT        /* which allows the fifo to be filled with samples */
    STWIO   r9, (r8)               /* effectively starting a sample */

start_sample_done:
    RET                            /* all done, so return */


/*
 *  sample_done
 *
 *  Description:       This function checks whether the started sample was completed.
 *                     If the sample was completed, a pointer to the buffer containing the
 *                     sampled data is provided. If the sample was not completed, a NULL
 *                     pointer is returned.
 *                     Note that this function returns a non-NULL pointer once per call to
```

```
 *                      start_sample.
 *
 *  Operation:          The function first checks the value of sample_pending to
 *                      ensure that a sample is ready. If no sample is ready, it simply
 *                      returns with NULL in r2.
 *                      Then, it resets the values of the shared variable to indicate that
 *                      a sample was completed.
 *                      Finally, the function clocks the FIFO twice to account for its
 *                      latency, and then reads FIFO_SIZE bytes in a loop, storing them in
 *                      array <samples>. Note that at each iteration, reading is performed
 *                      by bit-banging the FIFO's read clock. Also note that a calibration
 *                      constant is added to each sample to account for the front end's DC
 *                      offset.
 *
 *  Arguments:          None.
 *
 *  Return Value:       *samples - pointer to bytes acquired in sample if any; NULL
 *                                 otherwise (r2).
 *
 *  Local Variables:    r13 - pointer to current place in samples array.
 *                      r10 - number of sample currently being copied.
 *
 *  Shared Variables:   - sample_pending: logical value; zero if no sample is pending,
 *                                        non-zero otherwise. Read/Write.
 *
 *  Global Variables:   None.
 *
 *  Input:              Data samples from the FIFO.
 *
 *  Output:             None.
 *
 *  Error Handling:     None.
 *
 *  Limitations:        None.
 *
 *  Algorithms:         None.
 *  Data Structures:    samples - array of size FIFO_SIZE where samples are stored and
 *                                whose pointer is returned.
 *
 *  Registers Changed: r2, r8, r9, r10, r11, r12, r13, r14.
 *
 *  Revision History:
 *      5/29/14   Santiago Navonne     Initial revision.
 *      6/01/14   Santiago Navonne     Expanded documentation.
 *
 */
    .global sample_done
sample_done:
    MOV     r2, r0                  /* assume no sample ready: null pointer return val */
    MOVIA   r8, sample_pending      /* fetch current pending value to see if this call */
    LDB     r9, (r8)                /* should be ignored */
    BEQ     r0, r9, sample_done_done   /*  which is when value is zero */

    MOVIA   r8, sample_pending      /* reset sample_pending to indicate  */
    STB     r0, (r8)                /* no sample is ready for processing */

    MOVHI   r12, %hi(FIFO_DATA_BASE) /* load fifo data register address */
    ORI     r12, r12, %lo(FIFO_DATA_BASE) /* to actually read data from fifo */
    MOVHI   r8, %hi(TRIG_CTRL_SET) /* load ctrl reg set bit addr for */
    ORI     r8, r8, %lo(TRIG_CTRL_SET)    /* for bit banging */
    MOVIA   r13, sample             /* load array address to store samples */
    MOV     r2, r13                 /* and also use it as return value (pointer) */
    MOV     r10, r0                 /* and start a counter at 0 for looping */
    MOVI    r11, FIFO_SIZE          /* which will stop at FIFO_SIZE */
    MOVI    r9, FIFO_READ_BIT       /* finally load read clk bit for big banging */

                                    /* FIFO has 2 clocks latency */
    STWIO   r9, (r8)                /* send read clock high to output sample */
    ADDI    r8, r8, WORD_SIZE       /* and move to clear register: will send low next time */
    NOP                             /* wait for sample to actually come through */
    STWIO   r9, (r8)                /* send read clock low to prepare for next sample */
    ADDI    r8, r8, NEG_WORD_SIZE   /* and move to set register: will send high next time  */
    NOP                             /* wait for sample to actually come through */

    STWIO   r9, (r8)                /* send read clock high to output sample */
```

```
    ADDI    r8, r8, WORD_SIZE      /* and move to clear register: will send low next time */
    NOP                            /* wait for sample to actually come through */
    STWIO   r9, (r8)               /* send read clock low to prepare for next sample */
    ADDI    r8, r8, NEG_WORD_SIZE  /* and move to set register: will send high next time  */
    NOP                            /* wait for sample to actually come through */

get_data:
    STWIO   r9, (r8)               /* send read clock high to output sample */
    ADDI    r8, r8, WORD_SIZE      /* and move to clear register: will send low next time */
    NOP                            /* wait for sample to actually come through */

    LDBIO   r14, (r12)             /* read sample from fifo */
    ADDI    r14, r14, CALIBRATION  /* add calibration constant */
    STBIO   r14, (r13)             /* and store it in the sample array */

    STWIO   r9, (r8)               /* send read clock low to prepare for next sample */
    ADDI    r8, r8, NEG_WORD_SIZE  /* and move to set register: will send high next time  */

    ADDI    r10, r10, 1            /* increment counter */
    ADDI    r13, r13, 1            /* and sample pointer */
    BNE     r10, r11, get_data     /* and keep getting data until we reach end */

sample_done_done:                  /* all done */
    RET                            /* so return with pointer (or NULL) in r2 */


/*
 *  sample_handler
 *
 *  Description:       This function handles FIFO full hardware interrupts, notifying
 *                     the interface that a sample is ready to be read.
 *
 *  Operation:         The function changes the value of shared variable sample_pending
 *                     to indicate that a sample is now ready.
 *                     Then, it disables writing to the FIFO to make sure no data is
 *                     written as the FIFO is emptied.
 *                     Finally, it sends an EOI to reset the interrupt interface.
 *
 *  Arguments:         None.
 *
 *  Return Value:      None.
 *
 *  Local Variables:   None.
 *
 *  Shared Variables:  - sample_pending: logical value; zero if no sample is pending,
 *                                       non-zero otherwise. Write only.
 *
 *  Global Variables:  None.
 *
 *  Input:             None.
 *
 *  Output:            None.
 *
 *  Error Handling:    None.
 *
 *  Limitations:       None.
 *
 *  Algorithms:        None.
 *  Data Structures:   None.
 *
 *  Registers Changed: r8, r9.
 *
 *  Revision History:
 *      5/29/14   Santiago Navonne     Initial revision.
 *      6/01/14   Santiago Navonne     Expanded documentation.
 *
 */
    .global sample_handler
sample_handler:
    MOVIA   r8, sample_pending     /* mark sample_pending as true to indicate  */
    MOVI    r9, TRUE               /* a sample is ready for processing */
    STB     r9, (r8)

    MOVHI   r8, %hi(TRIG_CTRL_SET) /* load trigger control bit set reg address */
    ORI     r8, r8, %lo(TRIG_CTRL_SET)  /* to set fifo write enable (make inactive) */
```

```
    MOVI    r9, FIFO_WE_BIT        /* which prevents the fifo from being filled again */
    STWIO   r9, (r8)               /* effectively stopping a sample */

    MOVHI   r8, %hi(FIFO_FULL_BASE)/* write to edge capture register */
    ORI     r8, r8, %lo(FIFO_FULL_BASE) /* to send EOI */
    MOVI    r9, FIFO_INT
    STWIO   r9, EDGE_CAP_OF(r8)

    RET                            /* all done, so return */


/*
 *  trigger_init
 *
 *  Description:       This function performs all the necessary initialization of the
 *                     sampling and triggering interface, preparing shared variables
 *                     for use and configuring the triggering logic. It must be called
 *                     before using any of the other provided functions.
 *
 *  Operation:         The procedure first sets the shared variable sample_pending to
 *                     0, indicating that no sample is pending and no sample has been
 *                     started.
 *                     Then, it resets the triggering logic using the reset bit in the
 *                     control register, and configures the default triggering level,
 *                     delay, rate, and other settings.
 *                     Finally, it installs the interrupt handler by sending an EOI,
 *                     using the HAL API alt_ic_isr_register, and enabling interrupts
 *                     in the interrupt mask register.
 *
 *  Arguments:         None.
 *
 *  Return Value:      None.
 *
 *  Local Variables:   None.
 *
 *  Shared Variables:  - sample_pending: logical value; zero if no sample is pending,
 *                                       non-zero otherwise. 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:
 *      5/29/14   Santiago Navonne     Initial revision.
 *      6/01/14   Santiago Navonne     Expanded documentation.
 *
 */
    .global trigger_init
trigger_init:
    MOVIA   r8, sample_pending     /* mark sample_pending as false to indicate  */
    STB     r0, (r8)               /* no sample is ready for processing */

    MOVHI   r8, %hi(TRIG_LEVEL_BASE)     /* load trigger level reg address */
    ORI     r8, r8, %lo(TRIG_LEVEL_BASE) /* to set default value */
    MOVI    r9, TRIG_LEVEL_DEF
    STWIO   r9, (r8)

    MOVHI   r8, %hi(TRIG_DELAY_BASE)     /* load trigger delay reg address */
    ORI     r8, r8, %lo(TRIG_DELAY_BASE) /* to set default value */
    MOVI    r9, TRIG_DELAY_DEF
    STWIO   r9, (r8)

    MOVHI   r8, %hi(TRIG_PERIOD_BASE)    /* load trigger period reg address */
    ORI     r8, r8, %lo(TRIG_PERIOD_BASE)/* to set default value for rate */
```

```
601     MOVI    r9, TRIG_PERIOD_DEF
602     STWIO   r9, (r8)
603
604     MOVHI   r8, %hi(TRIG_CTRL_SET) /* load trigger control bit set reg address */
605     ORI     r8, r8, %lo(TRIG_CTRL_SET)   /* to reset trigger logic */
606     MOVI    r9, FIFO_RESET_BIT     /* by sending reset bit high */
607     STWIO   r9, (r8)
608
609     MOVI    r9, TRIG_CTRL_DEF      /* load default WE, read clock, auto */
610     STWIO   r9, (r8)               /* trigger, and slope values */
611     ADDI    r8, r8, WORD_SIZE      /* and move to clear register */
612     MOVI    r9, FIFO_RESET_BIT     /* to send reset bit low */
613     STWIO   r9, (r8)
614
615     MOVHI   r8, %hi(FIFO_FULL_BASE)/* write to edge capture register to send */
616     ORI     r8, r8, %lo(FIFO_FULL_BASE)  /* EOI to pending interrupts */
617     MOVI    r9, FIFO_INT           /* and to edge capture register to send */
618     STWIO   r9, EDGE_CAP_OF(r8)    /* EOI to pending interrupts */
619
620
621     ADDI    sp, sp, NEG_WORD_SIZE  /* register interrupt handler */
622     STW     ra, 0(sp)              /* push return address */
623     MOV     r4, r0                 /* argument ic_id is ignored */
624     MOVI    r5, FIFO_FULL_IRQ      /* second arg is IRQ num */
625     MOVIA   r6, sample_handler     /* third arg is int handler */
626     MOV     r7, r0                 /* fourth arg is data struct (null) */
627     ADDI    sp, sp, NEG_WORD_SIZE  /* fifth arg goes on stack */
628     STW     r0, 0(sp)              /*  and is ignored (so 0) */
629     CALL    alt_ic_isr_register    /* finally, call setup function */
630     ADDI    sp, sp, WORD_SIZE      /* clean up stack after call */
631     LDW     ra, 0(sp)              /* pop return address */
632     ADDI    sp, sp, WORD_SIZE
633
634     MOVHI   r8, %hi(FIFO_FULL_BASE)/* write to interrupt mask register */
635     ORI     r8, r8, %lo(FIFO_FULL_BASE)  /* to enable interrupts */
636     MOVI    r9, FIFO_INT
637     STWIO   r9, INTMASK_OF(r8)
638
639
640     RET                            /* all done, so return */
641
```