

```

1  /*****
2  /*
3  /*          TRIGGER.S          */
4  /*          Data sampling and triggering          */
5  /*          Digital Oscilloscope Project          */
6  /*          EE/CS 52          */
7  /*          Santiago Navonne          */
8  /*
9  /*****/
10
11 /*
12 Data sampling and triggering control routines for the EE/CS 52 Digital
13 Oscilloscope project. Function definitions are included in this file, and
14 are laid out as follows:
15 - set_sample_rate: Configures the sampling rate;
16 - set_trigger: Configures the manual trigger level and slope;
17 - set_delay: Configures the manual trigger delay;
18 - start_sample: Starts a new data sample with the previously configured
19 settings and passed auto-trigger configuration;
20 - sample_done: Checks whether a new data sample set is available, returning
21 a pointer to a buffer containing it if there is, or a NULL
22 pointer if there isn't;
23 - sample_handler: Handles sampling FIFO full interrupts;
24 - trigger_init: Initializes the environment's shared variables and the
25 triggering logic circuit (resetting it), effectively
26 preparing the sampling/triggering interface for use.
27
28
29 Revision History:
30 5/29/14 Santiago Navonne Initial revision.
31 6/01/14 Santiago Navonne Minor fixes; updated documentation.
32 6/11/14 Santiago Navonne Changed division algorithm in set_sample_rate.
33 */
34
35 /* Includes */
36 #include "general.h" /* General assembly constants */
37 #include "system.h" /* Base addresses */
38 #include "interfac.h" /* Software interface definitions */
39 #include "trigger.h" /* Local constants */
40
41
42 /* Variables */
43 .section .data /* No alignment necessary: variables are bytes */
44 sample_pending: .byte 0 /* Logical value: whether a sample is pending */
45 sample: .skip FIFO_SIZE /* Sample buffer */
46
47 .section .text /* Code starts here */
48
49 /*
50 * set_sample_rate
51 *
52 * Description: This procedure configures the sampling rate of the sampling
53 * interface. After execution, the interface will start sampling
54 * at the requested rate, rounded up to a multiple of the system
55 * clock. The return value is how many samples will be acquired,
56 * which is always the size of the FIFO.
57 * If an argument of 0 is passed, the function has no effect, and
58 * returns 0. The argument must however be less than or equal to
59 * the system clock divided by two; no error checking is performed
60 * on this.
61 *
62 * Operation: The procedure starts by error checking the value of the argument,
63 * simply returning 0 if it is invalid. Then, it computes the
64 * required clock period in system clock periods by dividing the
65 * system clock frequency by the requested sample rate.
66 * Finally, it saves the computed value to the trigger period
67 * register, and pulses the reset bit in the control register to
68 * reset the triggering logic. SIZE_X is ultimately moved into
69 * r2 as constant return value.
70 *
71 * Arguments: samples_per_sec - positive integer indicating the sample rate
72 * in samples per second (r4). The value must
73 * be less than or equal to the system clock
74 * divided by two.
75 */

```

```

76 * Return Value:      sample_num - positive integer, number of samples that will be
77 *                    acquired at the desired rate (r2).
78 *
79 * Local Variables:   None.
80 *
81 * Shared Variables:  None.
82 *
83 * Global Variables:  None.
84 *
85 * Input:             None.
86 *
87 * Output:            None.
88 *
89 * Error Handling:     If the argument is zero, the function has no effect, and returns 0.
90 *                    No error checking is performed on the upper bound of the sampling
91 *                    rate.
92 *
93 * Limitations:        Resulting sample clock is an integer multiple of the system clock;
94 *                    corresponding rate will be greater than or equal to the requested
95 *                    rate, with a difference in period less than the system clock's.
96 *                    Number of samples acquired must be <= FIFO_SIZE per hardware
97 *                    limitations (size of FIFO).
98 *
99 * Algorithms:         Division is performed using a repeated subtraction algorithm since
100 *                    hardware division cannot be assumed to be available. This algorithm
101 *                    is acceptable because generally very few iterations will be needed
102 *                    to reach the result.
103 * Data Structures:    None.
104 *
105 * Registers Changed:  r2, r4, r8, r9.
106 *
107 * Revision History:
108 *     5/29/14    Santiago Navonne    Initial revision.
109 *     6/01/14    Santiago Navonne    Added error checking, expanded documentation.
110 *     6/11/14    Santiago Navonne    Changed hardware divide instruction to division
111 *                                     by repeated subtraction.
112 *
113 */
114 .global set_sample_rate
115 set_sample_rate:
116     MOV     r2, r0                /* load return value of 0 in case of error */
117     BEQ     r4, r0, set_sample_rate_done /* error if argument is 0 */
118
119     MOVHI   r8, %hi(CLK_FREQ)     /* load system clock frequency to */
120     ORI     r8, r8, %lo(CLK_FREQ) /* find number of system clocks that takes */
121     /*DIVU   r9, r8, r4           /* by dividing the sys clk by the requested rate */
122     XOR     r9, r9, r9           /* prepare register for division: r9 is quotient */
123
124 div_check:
125     BLT     r8, r4, div_done      /* check if the divisor fits in the dividend */
126     /* we're done when it doesn't any more */
127
128 div_loop:
129     /* need to keep subtracting: */
130     SUB     r8, r8, r4           /* subtract divisor from dividend */
131     ADDI    r9, r9, 1           /* and increment quotient */
132     JMPI    div_check           /* thus repeat as needed */
133
134 div_done:
135     MOVHI   r8, %hi(TRIG_PERIOD_BASE) /* load period data register address to */
136     ORI     r8, r8, %lo(TRIG_PERIOD_BASE) /* finally save result to trigger period */
137     STWIO   r9, (r8)            /* data, effectively setting the sample rate */
138
139     MOVHI   r8, %hi(TRIG_CTRL_SET) /* load trigger control bit set reg address */
140     ORI     r8, r8, %lo(TRIG_CTRL_SET) /* to reset trigger logic */
141     MOVI    r9, FIFO_RESET_BIT /* by sending reset bit high */
142     STWIO   r9, (r8)
143     ADDI    r8, r8, WORD_SIZE /* and then move to bit clr reg */
144     STWIO   r9, (r8)           /* to send it low */
145
146     MOVI    r2, SIZE_X         /* number of samples acquired is always size of display */
147
148 set_sample_rate_done:
149     /* all done */
150     RET                    /* return value is in r2 */

```

```

151 /*
152  * set_trigger
153  *
154  * Description:      This function configures the triggering settings on the sampling
155  *                  interface. After execution, triggering will occur as soon as the
156  *                  input passes the value of <level>, in the direction indicated by
157  *                  <slope>. Note that these settings are only used when a sample is
158  *                  started with manual triggering enabled.
159  *
160  * Operation:        The procedure first "corrects" the level, mapping it to the
161  *                  right range ([0, 255]) and adding any necessary calibration
162  *                  constants.
163  *                  Then, it writes the slope bit to either the trigger control set
164  *                  or clear register, depending on what action needs to be performed,
165  *                  followed by the corrected level argument to the trigger level
166  *                  register.
167  *                  Finally, the reset bit within the trigger control register is
168  *                  pulsed to reset the triggering logic.
169  *
170  * Arguments:        level - trigger level to be configured, as a value between 0 and
171  *                  127, where 0 is the most negative level, and 127 is the
172  *                  most positive level (r4).
173  *                  slope - desired trigger slope; 1 for positive slope, 0 for
174  *                  negative slope (r5).
175  *
176  * Return Value:      None.
177  *
178  * Local Variables:   None.
179  *
180  * Shared Variables:  None.
181  *
182  * Global Variables:  None.
183  *
184  * Input:             None.
185  *
186  * Output:            None.
187  *
188  * Error Handling:     None.
189  *
190  * Limitations:        None.
191  *
192  * Algorithms:         None.
193  * Data Structures:    None.
194  *
195  * Registers Changed:  r4, r8, r9, r10.
196  *
197  * Revision History:
198  *      5/29/14   Santiago Navonne   Initial revision.
199  *      6/01/14   Santiago Navonne   Expanded documentation.
200  *
201  */
202
203 .global set_trigger
204 set_trigger:
205     MOVHI    r10, %hi(TRIG_LEVEL_BASE) /* load trigger level register address to update */
206     ORI      r10, r10, %lo(TRIG_LEVEL_BASE) /* the desired trigger level */
207     MOVI     r9, TRIG_LEVEL_SHIFT /* shift the passed argument left as needed to */
208     SLL      r4, r4, r9 /* make sure we output a full byte */
209     SUBI     r4, r4, CALIBRATION /* and correct value with calibration data */
210
211     MOVHI    r8, %hi(TRIG_CTRL_CLR) /* load control register bit clear address to */
212     ORI      r8, r8, %lo(TRIG_CTRL_CLR) /* initially assume that we want to set */
213     MOVI     r9, 2 /* slope to negative (clear the bit) */
214     SLL      r5, r5, r9 /* subtract argument multiplied by word size */
215     SUB      r8, r8, r5 /* effectively moving to set bit register if enabling */
216                     /* positive slope */
217
218     MOVI     r9, SLOPE_BIT /* finally write the appropriate bit to the register */
219     STWIO    r9, (r8) /* enabling or disabling the bit as needed */
220
221     STWIO    r4, (r10) /* and output desired trigger level */
222
223     MOVHI    r8, %hi(TRIG_CTRL_SET) /* load trigger control bit set reg address */
224     ORI      r8, r8, %lo(TRIG_CTRL_SET) /* to reset trigger logic */
225     MOVI     r9, FIFO_RESET_BIT /* by sending reset bit high */
226     STWIO    r9, (r8)

```

```

226     ADDI    r8, r8, WORD_SIZE    /* and then move to bit clr reg */
227     STWIO   r9, (r8)             /* to send it low */
228
229     RET                                /* all done, so return */
230
231
232 /*
233  * set_delay
234  *
235  * Description:      This procedure configures the sampling delay on manual triggers.
236  *                  After execution, triggering will occur <delay> samples after the
237  *                  configured level and slope settings are satisfied. Note that this
238  *                  setting is only used when manual triggering is enabled.
239  *                  Also note that delay must be less than MAX_DELAY.
240  *
241  * Operation:        The function first corrects the argument by adding the necessary
242  *                  hardware constant to it, and then outputs it to the trigger
243  *                  delay register.
244  *                  Finally, the reset bit within the trigger control register is
245  *                  pulsed to reset the triggering logic.
246  *
247  * Arguments:        delay - unsigned integer <= MAX_DELAY; trigger delay from
248  *                  trigger event in number of samples (r4).
249  *
250  * Return Value:      None.
251  *
252  * Local Variables:   None.
253  *
254  * Shared Variables:  None.
255  *
256  * Global Variables:  None.
257  *
258  * Input:             None.
259  *
260  * Output:            None.
261  *
262  * Error Handling:     None.
263  *
264  * Limitations:       Only positive delays less than or equal to MAX_DELAY are valid.
265  *
266  * Algorithms:        None.
267  * Data Structures:   None.
268  *
269  * Registers Changed: r4, r10.
270  *
271  * Revision History:
272  *      5/29/14    Santiago Navonne    Initial revision.
273  *      6/01/14    Santiago Navonne    Expanded documentation.
274  *
275  */
276     .global set_delay
277 set_delay:
278     MOVHI    r10, %hi(TRIG_DELAY_BASE) /* load trigger delay register address to update */
279     ORI      r10, r10, %lo(TRIG_DELAY_BASE) /* the desired delay time */
280     ADDI     r4, r4, DELAY_CONSTANT    /* add delay constant to correct argument */
281     STWIO    r4, (r10)                 /* and output to delay register, effectively */
282                                         /* configuring delay */
283
284     MOVHI    r8, %hi(TRIG_CTRL_SET)    /* load trigger control bit set reg address */
285     ORI      r8, r8, %lo(TRIG_CTRL_SET) /* to reset trigger logic */
286     MOVI     r9, FIFO_RESET_BIT        /* by sending reset bit high */
287     STWIO    r9, (r8)
288     ADDI     r8, r8, WORD_SIZE          /* and then move to bit clr reg */
289     STWIO    r9, (r8)                 /* to send it low */
290
291     RET                                /* all done, so return */
292
293
294 /*
295  * start_sample
296  *
297  * Description:      This procedure immediately starts sampling data. If the argument
298  *                  is FALSE, sampling starts upon a trigger event. If the argument
299  *                  is TRUE, sampling starts immediately.
300  *                  Any previously started but incomplete samples are cancelled and

```

```

301 *                replaced.
302 *
303 * Operation:      The procedure sets or clears the auto trigger bit in the trigger
304 *                control register to enable or disable auto triggering.
305 *                Finally, it starts the sample by enabling writing to the FIFO
306 *                through the write enable bit in the control register, and resets
307 *                the triggering logic.
308 *
309 * Arguments:      auto_trigger - TRUE if sampling should be started
310 *                automatically (i.e. as soon as possible),
311 *                FALSE if it should be started on a trigger
312 *                event (r4).
313 *
314 * Return Value:   None.
315 *
316 * Local Variables: None.
317 *
318 * Shared Variables: None.
319 *
320 * Global Variables: None.
321 *
322 * Input:          None.
323 *
324 * Output:         None.
325 *
326 * Error Handling: None.
327 *
328 * Limitations:    None.
329 *
330 * Algorithms:     None.
331 * Data Structures: None.
332 *
333 * Registers Changed: r8, r9.
334 *
335 * Revision History:
336 *     5/29/14    Santiago Navonne    Initial revision.
337 *     6/01/14    Santiago Navonne    Expanded documentation.
338 *
339 */
340 .global start_sample
341 start_sample:
342
343     MOVHI    r8, %hi(TRIG_CTRL_CLR) /* load trigger control bit clear reg address */
344     ORI      r8, r8, %lo(TRIG_CTRL_CLR) /* assuming we'll clear auto trigger bit */
345     MOVI     r9, 2 /* subtract argument multiplied by word size */
346     SLL      r4, r4, r9 /* effectively moving to set bit register if enabling */
347     SUB      r8, r8, r4 /* auto trigger*/
348
349     MOVI     r9, AUTO_TRIG_BIT /* store auto trigger bit in configured register */
350     STWIO    r9, (r8) /* enabling or disabling it as needed */
351
352     MOVHI    r8, %hi(TRIG_CTRL_SET) /* load trigger control bit set reg address */
353     ORI      r8, r8, %lo(TRIG_CTRL_SET) /* to reset trigger logic */
354     MOVI     r9, FIFO_RESET_BIT /* by sending reset bit high */
355     STWIO    r9, (r8)
356     ADDI     r8, r8, WORD_SIZE /* and then move to bit clr reg */
357     STWIO    r9, (r8) /* to send it low */
358
359     MOVHI    r8, %hi(TRIG_CTRL_CLR) /* load trigger control bit clear reg address */
360     ORI      r8, r8, %lo(TRIG_CTRL_CLR) /* to clear fifo write enable (make active) */
361     MOVI     r9, FIFO_WE_BIT /* which allows the fifo to be filled with samples */
362     STWIO    r9, (r8) /* effectively starting a sample */
363
364 start_sample_done:
365     RET /* all done, so return */
366
367 /*
368 * sample_done
369 *
370 * Description:     This function checks whether the started sample was completed.
371 *                If the sample was completed, a pointer to the buffer containing the
372 *                sampled data is provided. If the sample was not completed, a NULL
373 *                pointer is returned.
374 *                Note that this function returns a non-NULL pointer once per call to
375

```

```

376 *          start_sample.
377 *
378 * Operation:      The function first checks the value of sample_pending to
379 *                  ensure that a sample is ready. If no sample is ready, it simply
380 *                  returns with NULL in r2.
381 *                  Then, it resets the values of the shared variable to indicate that
382 *                  a sample was completed.
383 *                  Finally, the function clocks the FIFO twice to account for its
384 *                  latency, and then reads FIFO_SIZE bytes in a loop, storing them in
385 *                  array <samples>. Note that at each iteration, reading is performed
386 *                  by bit-banging the FIFO's read clock. Also note that a calibration
387 *                  constant is added to each sample to account for the front end's DC
388 *                  offset.
389 *
390 * Arguments:      None.
391 *
392 * Return Value:    *samples - pointer to bytes acquired in sample if any; NULL
393 *                  otherwise (r2).
394 *
395 * Local Variables: r13 - pointer to current place in samples array.
396 *                  r10 - number of sample currently being copied.
397 *
398 * Shared Variables: - sample_pending: logical value; zero if no sample is pending,
399 *                  non-zero otherwise. Read/Write.
400 *
401 * Global Variables: None.
402 *
403 * Input:           Data samples from the FIFO.
404 *
405 * Output:          None.
406 *
407 * Error Handling:   None.
408 *
409 * Limitations:      None.
410 *
411 * Algorithms:       None.
412 * Data Structures:  samples - array of size FIFO_SIZE where samples are stored and
413 *                  whose pointer is returned.
414 *
415 * Registers Changed: r2, r8, r9, r10, r11, r12, r13, r14.
416 *
417 * Revision History:
418 *      5/29/14   Santiago Navonne   Initial revision.
419 *      6/01/14   Santiago Navonne   Expanded documentation.
420 *
421 */
422 .global sample_done
423 sample_done:
424     MOV     r2, r0                /* assume no sample ready: null pointer return val */
425     MOVIA   r8, sample_pending    /* fetch current pending value to see if this call */
426     LDB     r9, (r8)              /* should be ignored */
427     BEQ     r0, r9, sample_done_done /* which is when value is zero */
428
429     MOVIA   r8, sample_pending    /* reset sample_pending to indicate */
430     STB     r0, (r8)              /* no sample is ready for processing */
431
432     MOVHI   r12, %hi(FIFO_DATA_BASE) /* load fifo data register address */
433     ORI     r12, r12, %lo(FIFO_DATA_BASE) /* to actually read data from fifo */
434     MOVHI   r8, %hi(TRIG_CTRL_SET) /* load ctrl reg set bit addr for */
435     ORI     r8, r8, %lo(TRIG_CTRL_SET) /* for bit banging */
436     MOVIA   r13, sample           /* load array address to store samples */
437     MOV     r2, r13               /* and also use it as return value (pointer) */
438     MOV     r10, r0               /* and start a counter at 0 for looping */
439     MOVI    r11, FIFO_SIZE        /* which will stop at FIFO_SIZE */
440     MOVI    r9, FIFO_READ_BIT     /* finally load read clk bit for big banging */
441
442                                     /* FIFO has 2 clocks latency */
443     STWIO   r9, (r8)              /* send read clock high to output sample */
444     ADDI    r8, r8, WORD_SIZE     /* and move to clear register: will send low next time */
445     NOP                                           /* wait for sample to actually come through */
446     STWIO   r9, (r8)              /* send read clock low to prepare for next sample */
447     ADDI    r8, r8, NEG_WORD_SIZE /* and move to set register: will send high next time */
448     NOP                                           /* wait for sample to actually come through */
449
450     STWIO   r9, (r8)              /* send read clock high to output sample */

```

```

451     ADDI    r8, r8, WORD_SIZE      /* and move to clear register: will send low next time */
452     NOP                                     /* wait for sample to actually come through */
453     STWIO   r9, (r8)                /* send read clock low to prepare for next sample */
454     ADDI    r8, r8, NEG_WORD_SIZE   /* and move to set register: will send high next time */
455     NOP                                     /* wait for sample to actually come through */
456
457 get_data:
458     STWIO   r9, (r8)                /* send read clock high to output sample */
459     ADDI    r8, r8, WORD_SIZE        /* and move to clear register: will send low next time */
460     NOP                                     /* wait for sample to actually come through */
461
462     LDBIO   r14, (r12)               /* read sample from fifo */
463     ADDI    r14, r14, CALIBRATION    /* add calibration constant */
464     STBIO   r14, (r13)               /* and store it in the sample array */
465
466     STWIO   r9, (r8)                /* send read clock low to prepare for next sample */
467     ADDI    r8, r8, NEG_WORD_SIZE    /* and move to set register: will send high next time */
468
469     ADDI    r10, r10, 1               /* increment counter */
470     ADDI    r13, r13, 1               /* and sample pointer */
471     BNE     r10, r11, get_data       /* and keep getting data until we reach end */
472
473 sample_done_done:                    /* all done */
474     RET                                     /* so return with pointer (or NULL) in r2 */
475
476
477 /*
478 * sample_handler
479 *
480 * Description:      This function handles FIFO full hardware interrupts, notifying
481 *                   the interface that a sample is ready to be read.
482 *
483 * Operation:        The function changes the value of shared variable sample_pending
484 *                   to indicate that a sample is now ready.
485 *                   Then, it disables writing to the FIFO to make sure no data is
486 *                   written as the FIFO is emptied.
487 *                   Finally, it sends an EOI to reset the interrupt interface.
488 *
489 * Arguments:        None.
490 *
491 * Return Value:      None.
492 *
493 * Local Variables:   None.
494 *
495 * Shared Variables:  - sample_pending: logical value; zero if no sample is pending,
496 *                   non-zero otherwise. Write only.
497 *
498 * Global Variables:  None.
499 *
500 * Input:             None.
501 *
502 * Output:            None.
503 *
504 * Error Handling:     None.
505 *
506 * Limitations:       None.
507 *
508 * Algorithms:        None.
509 * Data Structures:    None.
510 *
511 * Registers Changed: r8, r9.
512 *
513 * Revision History:
514 *   5/29/14   Santiago Navonne   Initial revision.
515 *   6/01/14   Santiago Navonne   Expanded documentation.
516 *
517 */
518 .global sample_handler
519 sample_handler:
520     MOVIA   r8, sample_pending       /* mark sample_pending as true to indicate */
521     MOVI    r9, TRUE                  /* a sample is ready for processing */
522     STB     r9, (r8)
523
524     MOVHI   r8, %hi(TRIG_CTRL_SET) /* load trigger control bit set reg address */
525     ORI     r8, r8, %lo(TRIG_CTRL_SET) /* to set fifo write enable (make inactive) */

```

```

526     MOVI    r9, FIFO_WE_BIT           /* which prevents the fifo from being filled again */
527     STWIO   r9, (r8)                 /* effectively stopping a sample */
528
529     MOVHI   r8, %hi(FIFO_FULL_BASE)/* write to edge capture register */
530     ORI     r8, r8, %lo(FIFO_FULL_BASE) /* to send EOI */
531     MOVI    r9, FIFO_INT
532     STWIO   r9, EDGE_CAP_OF(r8)
533
534     RET                                /* all done, so return */
535
536
537 /*
538  * trigger_init
539  *
540  * Description:      This function performs all the necessary initialization of the
541  *                  sampling and triggering interface, preparing shared variables
542  *                  for use and configuring the triggering logic. It must be called
543  *                  before using any of the other provided functions.
544  *
545  * Operation:       The procedure first sets the shared variable sample_pending to
546  *                  0, indicating that no sample is pending and no sample has been
547  *                  started.
548  *                  Then, it resets the triggering logic using the reset bit in the
549  *                  control register, and configures the default triggering level,
550  *                  delay, rate, and other settings.
551  *                  Finally, it installs the interrupt handler by sending an EOI,
552  *                  using the HAL API alt_ic_isr_register, and enabling interrupts
553  *                  in the interrupt mask register.
554  *
555  * Arguments:       None.
556  *
557  * Return Value:    None.
558  *
559  * Local Variables: None.
560  *
561  * Shared Variables: - sample_pending: logical value; zero if no sample is pending,
562  *                  non-zero otherwise. Write only.
563  *
564  * Global Variables: None.
565  *
566  * Input:           None.
567  *
568  * Output:          None.
569  *
570  * Error Handling:  None.
571  *
572  * Limitations:     None.
573  *
574  * Algorithms:      None.
575  * Data Structures: None.
576  *
577  * Registers Changed: r4, r5, r6, r7, r8, r9.
578  *
579  * Revision History:
580  *      5/29/14    Santiago Navonne      Initial revision.
581  *      6/01/14    Santiago Navonne      Expanded documentation.
582  *
583  */
584  .global trigger_init
585 trigger_init:
586     MOVIA   r8, sample_pending        /* mark sample_pending as false to indicate */
587     STB     r0, (r8)                 /* no sample is ready for processing */
588
589     MOVHI   r8, %hi(TRIG_LEVEL_BASE) /* load trigger level reg address */
590     ORI     r8, r8, %lo(TRIG_LEVEL_BASE) /* to set default value */
591     MOVI    r9, TRIG_LEVEL_DEF
592     STWIO   r9, (r8)
593
594     MOVHI   r8, %hi(TRIG_DELAY_BASE) /* load trigger delay reg address */
595     ORI     r8, r8, %lo(TRIG_DELAY_BASE) /* to set default value */
596     MOVI    r9, TRIG_DELAY_DEF
597     STWIO   r9, (r8)
598
599     MOVHI   r8, %hi(TRIG_PERIOD_BASE) /* load trigger period reg address */
600     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

```