

```

1  /*****
2  /*
3  /*                                MENUACT                                */
4  /*                                Menu Action Functions                    */
5  /*                                Digital Oscilloscope Project              */
6  /*                                EE/CS 52                                  */
7  /*
8  /*****
9
10 /*
11 This file contains the functions for carrying out menu actions for the
12 Digital Oscilloscope project. These functions are invoked when the <Left>
13 or <Right> key is pressed for a menu item. Also included are the functions
14 for displaying the current menu option selection. The functions included
15 are:
16     display_mode      - display trigger mode
17     display_scale     - display the scale type
18     display_sweep     - display the sweep rate
19     display_trg_delay - display the trigger delay
20     display_trg_level - display the trigger level
21     display_trg_slope - display the trigger slope
22     get_trigger_mode  - get the current trigger mode
23     mode_down        - go to the "next" trigger mode
24     mode_up          - go to the "previous" trigger mode
25     no_display        - nothing to display for option setting
26     no_menu_action    - no action to perform for <Left> or <Right> key
27     scale_down        - go to the "next" scale type
28     scale_up          - go to the "previous" scale type
29     set_scale         - set the scale type
30     set_sweep         - set the sweep rate
31     set_trg_delay     - set the trigger delay
32     set_trg_level     - set the trigger level
33     set_trg_slope     - set the trigger slope
34     set_trigger_mode  - set the trigger mode
35     sweep_down        - decrease the sweep rate
36     sweep_up          - increase the sweep rate
37     trg_delay_down    - decrease the trigger delay
38     trg_delay_up      - increase the trigger delay
39     trg_level_down    - decrease the trigger level
40     trg_level_up      - increase the trigger level
41     trg_slope_toggle  - toggle the trigger slope between "+" and "-"
42
43 The local functions included are:
44     adjust_trg_delay  - adjust the trigger delay for a new sweep rate
45     cvt_num_field    - converts a numeric field value to a string
46
47 The locally global variable definitions included are:
48     delay            - current trigger delay
49     level            - current trigger level
50     scale            - current display scale type
51     slope            - current trigger slope
52     sweep            - current sweep rate
53     sweep_rates      - table of information on possible sweep rates
54     trigger_mode     - current triggering mode
55
56
57 Revision History
58     3/8/94   Glen George   Initial revision.
59     3/13/94  Glen George   Updated comments.
60     3/13/94  Glen George   Changed all arrays of constant strings to be
61                             static so compiler generates correct code.
62     3/13/94  Glen George   Changed scale to type enum scale_type and
63                             output the selection as "None" or "Axes".
64                             This will allow for easier future expansion.
65     3/13/94  Glen George   Changed name of set_axes function (in
66                             tracutil.c) to set_display_scale.
67     3/10/95  Glen George   Changed calculation of displayed trigger
68                             level to use constants MIN_TRG_LEVEL_SET and
69                             MAX_TRG_LEVEL_SET to get the trigger level
70                             range.
71     3/17/97  Glen George   Updated comments.
72     5/3/06   Glen George   Changed sweep definitions to include new
73                             sweep rates of 100 ns, 200 ns, 500 ns, and
74                             1 us and updated functions to handle these
75                             new rates.

```

```

76      5/9/06   Glen George      Added new a triggering mode (automatic
77                                     triggering) and a new scale (grid) and
78                                     updated functions to implement these options.
79      5/9/06   Glen George      Added functions for setting the triggering
80                                     mode and scale by going up and down the list
81                                     of possibilities instead of just toggling
82                                     between one of two possibilities (since there
83                                     are more than two now).
84      5/9/06   Glen George      Added accessor function (get_trigger_mode)
85                                     to be able to get the current trigger mode.
86      6/6/14   Santiago Navonne  Added fastest sweep rate and changed their
87                                     values to reflect actual possible rates.
88      6/11/14  Santiago Navonne  Modified delay set function to support faster
89                                     sweep rates.
90  */
91
92
93
94  /* library include files */
95  /* none */
96
97  /* local include files */
98  #include "interfac.h"
99  #include "scopedef.h"
100 #include "lcdout.h"
101 #include "menuact.h"
102 #include "tracutil.h"
103
104
105
106
107 /* local function declarations */
108 static void  adjust_trg_delay(int, int);          /* adjust the trigger delay for new sweep */
109 static void  cvt_num_field(long int, char *);    /* convert a number to a string */
110
111
112
113
114 /* locally global variables
115
116 /* trace parameters */
117 static enum trigger_type  trigger_mode; /* current triggering mode */
118 static enum scale_type    scale;        /* current scale type */
119 static int                sweep;        /* sweep rate index */
120 static int                level;        /* current trigger level */
121 static enum slope_type    slope;        /* current trigger slope */
122 static long int           delay;        /* current trigger delay */
123
124 /* sweep rate information */
125 static const struct sweep_info  sweep_rates[] =
126     { { 19000000L, " 52 ns " },
127       { 9500000L, " 104 ns" },
128       { 4750000L, " 208 ns" },
129       { 2000000L, " 500 ns" },
130       { 1000000L, " 1 \004s " },
131       { 500000L, " 2 \004s " },
132       { 200000L, " 5 \004s " },
133       { 100000L, " 10 \004s " },
134       { 50000L, " 20 \004s " },
135       { 20000L, " 50 \004s " },
136       { 10000L, " 100 \004s" },
137       { 5000L, " 200 \004s" },
138       { 2000L, " 500 \004s" },
139       { 1000L, " 1 ms " },
140       { 500L, " 2 ms " },
141       { 200L, " 5 ms " },
142       { 100L, " 10 ms " },
143       { 50L, " 20 ms " } };
144
145
146
147
148 /*
149     no_menu_action
150

```

```

151 Description:      This function handles a menu action when there is nothing
152                   to be done.  It just returns.
153
154 Arguments:        None.
155 Return Value:     None.
156
157 Input:            None.
158 Output:           None.
159
160 Error Handling:    None.
161
162 Algorithms:        None.
163 Data Structures:   None.
164
165 Global Variables: None.
166
167 Author:           Glen George
168 Last Modified:    Mar. 8, 1994
169
170 */
171
172 void no_menu_action()
173 {
174     /* variables */
175     /* none */
176
177
178
179     /* nothing to do - return */
180     return;
181 }
182
183
184
185
186
187 /*
188 no_display
189
190 Description:      This function handles displaying a menu option's setting
191                   when there is nothing to display.  It just returns,
192                   ignoring all arguments.
193
194 Arguments:        x_pos (int) - x position (in character cells) at which to
195                   display the menu option (not used).
196                   y_pos (int) - y position (in character cells) at which to
197                   display the menu option (not used).
198                   style (int) - style with which to display the menu option
199                   (not used).
200 Return Value:     None.
201
202 Input:            None.
203 Output:           None.
204
205 Error Handling:    None.
206
207 Algorithms:        None.
208 Data Structures:   None.
209
210 Global Variables: None.
211
212 Author:           Glen George
213 Last Modified:    Mar. 8, 1994
214
215 */
216
217 void no_display(int x_pos, int y_pos, int style)
218 {
219     /* variables */
220     /* none */
221
222
223
224     /* nothing to do - return */
225     return;

```

```

226 }
227
228
229
230
231
232 /*
233     set_trigger_mode
234
235     Description:      This function sets the triggering mode to the passed
236                       value.
237
238     Arguments:       m (enum trigger_type) - mode to which to set the
239                       triggering mode.
240     Return Value:    None.
241
242     Input:           None.
243     Output:          None.
244
245     Error Handling:   None.
246
247     Algorithms:      None.
248     Data Structures: None.
249
250     Global Variables: trigger_mode - initialized to the passed value.
251
252     Author:          Glen George
253     Last Modified:   Mar. 8, 1994
254 */
255
256
257 void set_trigger_mode(enum trigger_type m)
258 {
259     /* variables */
260     /* none */
261
262
263
264     /* set the trigger mode */
265     trigger_mode = m;
266
267     /* set the new mode */
268     set_mode(trigger_mode);
269
270
271     /* all done setting the trigger mode - return */
272     return;
273 }
274
275
276
277
278
279 /*
280     get_trigger_mode
281
282     Description:      This function returns the current triggering mode.
283
284     Arguments:       None.
285     Return Value:    (enum trigger_type) - current triggering mode.
286
287     Input:           None.
288     Output:          None.
289
290     Error Handling:   None.
291
292     Algorithms:      None.
293     Data Structures: None.
294
295     Global Variables: trigger_mode - value is returned (not changed).
296
297     Author:          Glen George
298     Last Modified:   May 9, 2006
299 */
300

```

```

301 enum trigger_type get_trigger_mode()
302 {
303     /* variables */
304     /* none */
305
306
307
308
309     /* return the current trigger mode */
310     return trigger_mode;
311 }
312
313
314
315
316
317 /*
318 mode_down
319
320 Description:      This function handles moving down the list of trigger
321                  modes. It changes to the "next" triggering mode and
322                  sets that as the current mode.
323
324 Arguments:      None.
325 Return Value:   None.
326
327 Input:          None.
328 Output:         None.
329
330 Error Handling:  None.
331
332 Algorithms:     None.
333 Data Structures: None.
334
335 Global Variables: trigger_mode - changed to "next" trigger mode.
336
337 Author:         Glen George
338 Last Modified:  May 9, 2006
339
340 */
341
342 void mode_down()
343 {
344     /* variables */
345     /* none */
346
347
348
349     /* move to the "next" triggering mode */
350     if (trigger_mode == NORMAL_TRIGGER)
351         trigger_mode = AUTO_TRIGGER;
352     else if (trigger_mode == AUTO_TRIGGER)
353         trigger_mode = ONESHOT_TRIGGER;
354     else
355         trigger_mode = NORMAL_TRIGGER;
356
357     /* set the new mode */
358     set_mode(trigger_mode);
359
360
361     /* all done with the trigger mode - return */
362     return;
363 }
364
365
366
367
368
369 /*
370 mode_up
371
372 Description:      This function handles moving up the list of trigger
373                  modes. It changes to the "previous" triggering mode and
374                  sets that as the current mode.
375

```

```

376 Arguments:      None.
377 Return Value:   None.
378
379 Input:          None.
380 Output:         None.
381
382 Error Handling:  None.
383
384 Algorithms:     None.
385 Data Structures: None.
386
387 Global Variables: trigger_mode - changed to "previous" trigger mode.
388
389 Author:         Glen George
390 Last Modified:  May 9, 2006
391
392 */
393
394 void mode_up()
395 {
396     /* variables */
397     /* none */
398
399
400
401     /* move to the "previous" triggering mode */
402     if (trigger_mode == NORMAL_TRIGGER)
403         trigger_mode = ONESHOT_TRIGGER;
404     else if (trigger_mode == AUTO_TRIGGER)
405         trigger_mode = NORMAL_TRIGGER;
406     else
407         trigger_mode = AUTO_TRIGGER;
408
409     /* set the new mode */
410     set_mode(trigger_mode);
411
412
413     /* all done with the trigger mode - return */
414     return;
415 }
416
417
418
419
420
421 /*
422 display_mode
423
424 Description:      This function displays the current triggering mode at the
425                   passed position, in the passed style.
426
427 Arguments:        x_pos (int) - x position (in character cells) at which to
428                   display the trigger mode.
429                   y_pos (int) - y position (in character cells) at which to
430                   display the trigger mode.
431                   style (int) - style with which to display the trigger
432                   mode.
433 Return Value:     None.
434
435 Input:           None.
436 Output:          The trigger mode is displayed at the passed position on
437                   the screen.
438
439 Error Handling:   None.
440
441 Algorithms:      None.
442 Data Structures: None.
443
444 Global Variables: trigger_mode - determines which string is displayed.
445
446 Author:         Glen George
447 Last Modified:  May 9, 2006
448
449 */
450

```

```

451 void display_mode(int x_pos, int y_pos, int style)
452 {
453     /* variables */
454
455     /* the mode strings (must match enumerated type) */
456     const static char * const modes[] = { " Normal  ",
457                                           " Automatic",
458                                           " One-Shot " };
459
460
461
462     /* display the trigger mode */
463     plot_string(x_pos, y_pos, modes[trigger_mode], style);
464
465
466     /* all done displaying the trigger mode - return */
467     return;
468
469 }
470
471
472
473
474 /*
475  set_scale
476
477  Description:      This function sets the scale type to the passed value.
478
479  Arguments:       s (enum scale_type) - scale type to which to initialize
480                  the scale status.
481  Return Value:    None.
482
483  Input:           None.
484  Output:          The new trace display is updated with the new scale.
485
486  Error Handling:   None.
487
488  Algorithms:      None.
489  Data Structures: None.
490
491  Global Variables: scale - initialized to the passed value.
492
493  Author:          Glen George
494  Last Modified:   Mar. 13, 1994
495
496 */
497
498 void set_scale(enum scale_type s)
499 {
500     /* variables */
501     /* none */
502
503
504
505     /* set the scale type */
506     scale = s;
507
508     /* output the scale appropriately */
509     set_display_scale(scale);
510
511
512     /* all done setting the scale type - return */
513     return;
514
515 }
516
517
518
519
520 /*
521  scale_down
522
523  Description:      This function handles moving down the list of scale
524                  types. It changes to the "next" type of scale and sets
525                  this as the current scale type.

```

```

526 Arguments:      None.
527 Return Value:   None.
528
529
530 Input:          None.
531 Output:         The new scale is output to the trace display.
532
533 Error Handling:  None.
534
535 Algorithms:     None.
536 Data Structures: None.
537
538 Global Variables: scale - changed to the "next" scale type.
539
540 Author:         Glen George
541 Last Modified:  May 9, 2006
542
543 */
544
545 void scale_down()
546 {
547     /* variables */
548     /* none */
549
550
551
552     /* change to the "next" scale type */
553     if (scale == SCALE_NONE)
554         scale = SCALE_AXES;
555     else if (scale == SCALE_AXES)
556         scale = SCALE_GRID;
557     else
558         scale = SCALE_NONE;
559
560     /* set the scale type */
561     set_display_scale(scale);
562
563
564     /* all done with toggling the scale type - return */
565     return;
566 }
567
568
569
570
571
572 /*
573 scale_up
574
575 Description:      This function handles moving up the list of scale types.
576                  It changes to the "previous" type of scale and sets this
577                  as the current scale type.
578
579 Arguments:        None.
580 Return Value:     None.
581
582 Input:           None.
583 Output:          The new scale is output to the trace display.
584
585 Error Handling:   None.
586
587 Algorithms:      None.
588 Data Structures: None.
589
590 Global Variables: scale - changed to the "previous" scale type.
591
592 Author:         Glen George
593 Last Modified:  May 9, 2006
594
595 */
596
597 void scale_up()
598 {
599     /* variables */
600     /* none */

```



```

601
602
603
604     /* change to the "previous" scale type */
605     if (scale == SCALE_NONE)
606         scale = SCALE_GRID;
607     else if (scale == SCALE_AXES)
608         scale = SCALE_NONE;
609     else
610         scale = SCALE_AXES;
611
612     /* set the scale type */
613     set_display_scale(scale);
614
615
616     /* all done with toggling the scale type - return */
617     return;
618
619 }
620
621
622
623
624 /*
625 display_scale
626
627 Description:      This function displays the current scale type at the
628                   passed position, in the passed style.
629
630 Arguments:      x_pos (int) - x position (in character cells) at which to
631                   display the scale type.
632                   y_pos (int) - y position (in character cells) at which to
633                   display the scale type.
634                   style (int) - style with which to display the scale type.
635 Return Value:    None.
636
637 Input:          None.
638 Output:         The scale type is displayed at the passed position on the
639                   display.
640
641 Error Handling:  None.
642
643 Algorithms:     None.
644 Data Structures: None.
645
646 Global Variables: scale - determines which string is displayed.
647
648 Author:         Glen George
649 Last Modified:  Mar. 13, 1994
650
651 */
652
653 void display_scale(int x_pos, int y_pos, int style)
654 {
655     /* variables */
656
657     /* the scale type strings (must match enumerated type) */
658     const static char * const scale_stat[] = { " None",
659                                                " Axes",
660                                                " Grid" };
661
662
663
664     /* display the scale status */
665     plot_string(x_pos, y_pos, scale_stat[scale], style);
666
667
668     /* all done displaying the scale status - return */
669     return;
670
671 }
672
673
674
675

```

```

676  /*
677  set_sweep
678
679  Description:      This function sets the sweep rate to the passed value.
680                   The passed value gives the sweep rate to choose from the
681                   list of sweep rates (it gives the list index).
682
683  Arguments:       s (int) - index into the list of sweep rates to which to
684                   set the current sweep rate.
685  Return Value:    None.
686
687  Input:           None.
688  Output:          None.
689
690  Error Handling:   The passed index is not checked for validity.
691
692  Algorithms:      None.
693  Data Structures: None.
694
695  Global Variables: sweep - initialized to the passed value.
696
697  Author:          Glen George
698  Last Modified:   Mar. 8, 1994
699
700  */
701
702  void set_sweep(int s)
703  {
704      /* variables */
705      int sample_size;      /* sample size for this sweep rate */
706
707
708
709      /* set the new sweep rate */
710      sweep = s;
711
712      /* set the sweep rate for the hardware */
713      sample_size = set_sample_rate(sweep_rates[sweep].sample_rate);
714      /* also set the sample size for the trace capture */
715      set_trace_size(sample_size);
716
717
718      /* all done initializing the sweep rate - return */
719      return;
720  }
721
722
723
724
725
726  /*
727  sweep_down
728
729  Description:      This function handles decreasing the current sweep rate.
730                   The new sweep rate (and sample size) is sent to the
731                   hardware (and trace routines). If an attempt is made to
732                   lower the sweep rate below the minimum value it is not
733                   changed. This routine also updates the sweep delay based
734                   on the new sweep rate (to keep the delay time constant).
735
736  Arguments:       None.
737  Return Value:    None.
738
739  Input:           None.
740  Output:          None.
741
742  Error Handling:   None.
743
744  Algorithms:      None.
745  Data Structures: None.
746
747  Global Variables: sweep - decremented if not already 0.
748                   delay - increased to keep delay time constant.
749
750  Known Bugs:      The updated delay time is not displayed. Since the time

```

```

751         is typically only rounded to the new sample rate, this is
752         not a major problem.
753
754     Author:          Glen George
755     Last Modified:   Mar. 8, 1994
756
757 */
758
759 void sweep_down()
760 {
761     /* variables */
762     int sample_size;          /* sample size for the new sweep rate */
763
764
765     /* decrease the sweep rate, if not already the minimum */
766     if (sweep > 0) {
767         /* not at minimum, adjust delay for new sweep */
768         adjust_trg_delay(sweep, (sweep - 1));
769         /* now set new sweep rate */
770         sweep--;
771     }
772
773     /* set the sweep rate for the hardware */
774     sample_size = set_sample_rate(sweep_rates[sweep].sample_rate);
775     /* also set the sample size for the trace capture */
776     set_trace_size(sample_size);
777
778
779     /* all done with lowering the sweep rate - return */
780     return;
781 }
782
783
784
785
786
787
788 /*
789 sweep_up
790
791 Description:          This function handles increasing the current sweep rate.
792                      The new sweep rate (and sample size) is sent to the
793                      hardware (and trace routines). If an attempt is made to
794                      raise the sweep rate above the maximum value it is not
795                      changed. This routine also updates the sweep delay based
796                      on the new sweep rate (to keep the delay time constant).
797
798 Arguments:           None.
799 Return Value:        None.
800
801 Input:               None.
802 Output:              None.
803
804 Error Handling:      None.
805
806 Algorithms:          None.
807 Data Structures:     None.
808
809 Global Variables:    sweep - incremented if not already the maximum value.
810                      delay - decreased to keep delay time constant.
811
812 Known Bugs:          The updated delay time is not displayed. Since the time
813                      is typically only rounded to the new sample rate, this is
814                      not a major problem.
815
816 Author:              Glen George
817 Last Modified:       Mar. 8, 1994
818
819 */
820
821 void sweep_up()
822 {
823     /* variables */
824     int sample_size;          /* sample size for the new sweep rate */
825

```

```

826
827
828     /* increase the sweep rate, if not already the maximum */
829     if (sweep < (NO_SWEEP_RATES - 1)) {
830         /* not at maximum, adjust delay for new sweep */
831         adjust_trg_delay(sweep, (sweep + 1));
832         /* now set new sweep rate */
833         sweep++;
834     }
835
836     /* set the sweep rate for the hardware */
837     sample_size = set_sample_rate(sweep_rates[sweep].sample_rate);
838     /* also set the sample size for the trace capture */
839     set_trace_size(sample_size);
840
841
842     /* all done with raising the sweep rate - return */
843     return;
844 }
845
846
847
848
849
850 /*
851 display_sweep
852
853 Description:      This function displays the current sweep rate at the
854                  passed position, in the passed style.
855
856 Arguments:      x_pos (int) - x position (in character cells) at which to
857                  display the sweep rate.
858                  y_pos (int) - y position (in character cells) at which to
859                  display the sweep rate.
860                  style (int) - style with which to display the sweep rate.
861 Return Value:    None.
862
863 Input:          None.
864 Output:         The sweep rate is displayed at the passed position on the
865                  display.
866
867 Error Handling:  None.
868
869 Algorithms:     None.
870 Data Structures: None.
871
872 Global Variables: sweep - determines which string is displayed.
873
874 Author:         Glen George
875 Last Modified:  Mar. 8, 1994
876
877 */
878
879 void display_sweep(int x_pos, int y_pos, int style)
880 {
881     /* variables */
882     /* none */
883
884
885
886     /* display the sweep rate */
887     plot_string(x_pos, y_pos, sweep_rates[sweep].s, style);
888
889
890     /* all done displaying the sweep rate - return */
891     return;
892 }
893
894
895
896
897
898 /*
899 set_trg_level
900

```

```

901 Description:      This function sets the trigger level to the passed value.
902
903 Arguments:        1 (int) - value to which to set the trigger level.
904 Return Value:     None.
905
906 Input:            None.
907 Output:           None.
908
909 Error Handling:    The passed value is not checked for validity.
910
911 Algorithms:        None.
912 Data Structures:   None.
913
914 Global Variables:  level - initialized to the passed value.
915
916 Author:           Glen George
917 Last Modified:    Mar. 8, 1994
918
919 */
920
921 void set_trg_level(int l)
922 {
923     /* variables */
924     /* none */
925
926
927     /* set the trigger level */
928     level = l;
929
930     /* set the trigger level in hardware too */
931     set_trigger(level, slope);
932
933
934     /* all done initializing the trigger level - return */
935     return;
936
937 }
938
939
940
941
942
943 /*
944 trg_level_down
945
946 Description:      This function handles decreasing the current trigger
947                   level. The new trigger level is sent to the hardware.
948                   If an attempt is made to lower the trigger level below
949                   the minimum value it is not changed.
950
951 Arguments:        None.
952 Return Value:     None.
953
954 Input:            None.
955 Output:           None.
956
957 Error Handling:    None.
958
959 Algorithms:        None.
960 Data Structures:   None.
961
962 Global Variables:  level - decremented if not already at the minimum value.
963
964 Author:           Glen George
965 Last Modified:    Mar. 8, 1994
966
967 */
968
969 void trg_level_down()
970 {
971     /* variables */
972     /* none */
973
974
975

```

```

976     /* decrease the trigger level, if not already the minimum */
977     if (level > MIN_TRG_LEVEL_SET)
978         level--;
979
980     /* set the trigger level for the hardware */
981     set_trigger(level, slope);
982
983
984     /* all done with lowering the trigger level - return */
985     return;
986
987 }
988
989
990
991
992 /*
993   trg_level_up
994
995   Description:      This function handles increasing the current trigger
996                     level.  The new trigger level is sent to the hardware.
997                     If an attempt is made to raise the trigger level above
998                     the maximum value it is not changed.
999
1000   Arguments:        None.
1001   Return Value:      None.
1002
1003   Input:             None.
1004   Output:            None.
1005
1006   Error Handling:    None.
1007
1008   Algorithms:        None.
1009   Data Structures:   None.
1010
1011   Global Variables:  level - incremented if not already the maximum value.
1012
1013   Author:            Glen George
1014   Last Modified:     Mar. 8, 1994
1015
1016 */
1017
1018 void trg_level_up()
1019 {
1020     /* variables */
1021     /* none */
1022
1023
1024
1025     /* increase the trigger level, if not already the maximum */
1026     if (level < MAX_TRG_LEVEL_SET)
1027         level++;
1028
1029     /* tell the hardware the new trigger level */
1030     set_trigger(level, slope);
1031
1032
1033     /* all done raising the trigger level - return */
1034     return;
1035
1036 }
1037
1038
1039
1040
1041 /*
1042   display_trg_level
1043
1044   Description:      This function displays the current trigger level at the
1045                     passed position, in the passed style.
1046
1047   Arguments:        x_pos (int) - x position (in character cells) at which to
1048                     display the trigger level.
1049                     y_pos (int) - y position (in character cells) at which to
1050                     display the trigger level.

```

```

1051         style (int) - style with which to display the trigger
1052                        level.
1053 Return Value:      None.
1054
1055 Input:             None.
1056 Output:            The trigger level is displayed at the passed position on
1057                    the display.
1058
1059 Error Handling:    None.
1060
1061 Algorithms:        None.
1062 Data Structures:   None.
1063
1064 Global Variables: level - determines the value displayed.
1065
1066 Author:            Glen George
1067 Last Modified:     Mar. 10, 1995
1068
1069 */
1070
1071 void display_trg_level(int x_pos, int y_pos, int style)
1072 {
1073     /* variables */
1074     char    level_str[] = "          "; /* string containing the trigger level */
1075     long int l;          /* trigger level in mV */
1076
1077
1078
1079     /* compute the trigger level in millivolts */
1080     l = ((long int) MAX_LEVEL - MIN_LEVEL) * level / (MAX_TRG_LEVEL_SET - MIN_TRG_LEVEL_SET) + MIN_LEV
1081
1082     /* convert the level to the string (leave first character blank) */
1083     cvt_num_field(l, &level_str[1]);
1084
1085     /* add in the units */
1086     level_str[7] = 'V';
1087
1088
1089     /* now finally display the trigger level */
1090     plot_string(x_pos, y_pos, level_str, style);
1091
1092
1093     /* all done displaying the trigger level - return */
1094     return;
1095
1096 }
1097
1098
1099
1100
1101 /*
1102 set_trg_slope
1103
1104 Description:        This function sets the trigger slope to the passed value.
1105
1106 Arguments:          s (enum slope_type) - trigger slope type to which to set
1107                    the locally global slope.
1108 Return Value:       None.
1109
1110 Input:              None.
1111 Output:             None.
1112
1113 Error Handling:     None.
1114
1115 Algorithms:         None.
1116 Data Structures:    None.
1117
1118 Global Variables:   slope - set to the passed value.
1119
1120 Author:             Glen George
1121 Last Modified:      Mar. 8, 1994
1122
1123 */
1124
1125 void set_trg_slope(enum slope_type s)

```

```

1126 {
1127     /* variables */
1128     /* none */
1129
1130
1131
1132     /* set the slope type */
1133     slope = s;
1134
1135     /* also tell the hardware what the slope is */
1136     set_trigger(level, slope);
1137
1138
1139     /* all done setting the trigger slope - return */
1140     return;
1141
1142 }
1143
1144
1145
1146
1147 /*
1148     trg_slope_toggle
1149
1150     Description:      This function handles toggling (and setting) the current
1151                      trigger slope.
1152
1153     Arguments:       None.
1154     Return Value:    None.
1155
1156     Input:           None.
1157     Output:          None.
1158
1159     Error Handling:   None.
1160
1161     Algorithms:      None.
1162     Data Structures: None.
1163
1164     Global Variables: slope - toggled.
1165
1166     Author:          Glen George
1167     Last Modified:   Mar. 8, 1994
1168
1169 */
1170
1171 void trg_slope_toggle()
1172 {
1173     /* variables */
1174     /* none */
1175
1176
1177
1178     /* toggle the trigger slope */
1179     if (slope == SLOPE_POSITIVE)
1180         slope = SLOPE_NEGATIVE;
1181     else
1182         slope = SLOPE_POSITIVE;
1183
1184     /* set the new trigger slope */
1185     set_trigger(level, slope);
1186
1187
1188     /* all done with the trigger slope - return */
1189     return;
1190
1191 }
1192
1193
1194
1195
1196 /*
1197     display_trg_slope
1198
1199     Description:      This function displays the current trigger slope at the
1200                      passed position, in the passed style.

```



```

1201
1202 Arguments:      x_pos (int) - x position (in character cells) at which to
1203                  display the trigger slope.
1204                  y_pos (int) - y position (in character cells) at which to
1205                  display the trigger slope.
1206                  style (int) - style with which to display the trigger
1207                               slope.
1208 Return Value:    None.
1209
1210 Input:           None.
1211 Output:          The trigger slope is displayed at the passed position on
1212                  the screen.
1213
1214 Error Handling:   None.
1215
1216 Algorithms:      None.
1217 Data Structures: None.
1218
1219 Global Variables: slope - determines which string is displayed.
1220
1221 Author:          Glen George
1222 Last Modified:   Mar. 13, 1994
1223
1224 */
1225
1226 void display_trg_slope(int x_pos, int y_pos, int style)
1227 {
1228     /* variables */
1229
1230     /* the trigger slope strings (must match enumerated type) */
1231     const static char * const slopes[] = { " +", " -" };
1232
1233
1234
1235     /* display the trigger slope */
1236     plot_string(x_pos, y_pos, slopes[slope], style);
1237
1238
1239     /* all done displaying the trigger slope - return */
1240     return;
1241 }
1242
1243
1244
1245
1246
1247 /*
1248 set_trg_delay
1249
1250 Description:      This function sets the trigger delay to the passed value.
1251
1252 Arguments:        d (long int) - value to which to set the trigger delay.
1253 Return Value:     None.
1254
1255 Input:            None.
1256 Output:           None.
1257
1258 Error Handling:    The passed value is not checked for validity.
1259
1260 Algorithms:       None.
1261 Data Structures:  None.
1262
1263 Global Variables: delay - initialized to the passed value.
1264
1265 Author:           Glen George
1266 Last Modified:    Mar. 8, 1994
1267
1268 */
1269
1270 void set_trg_delay(long int d)
1271 {
1272     /* variables */
1273     /* none */
1274
1275

```

```

1276      /* set the trigger delay */
1277      delay = d;
1278
1279      /* set the trigger delay in hardware too */
1280      set_delay(delay);
1281
1282
1283
1284      /* all done initializing the trigger delay - return */
1285      return;
1286
1287  }
1288
1289
1290
1291
1292  /*
1293  trg_delay_down
1294
1295  Description:      This function handles decreasing the current trigger
1296                    delay. The new trigger delay is sent to the hardware.
1297                    If an attempt is made to lower the trigger delay below
1298                    the minimum value it is not changed.
1299
1300  Arguments:        None.
1301  Return Value:     None.
1302
1303  Input:            None.
1304  Output:           None.
1305
1306  Error Handling:   None.
1307
1308  Algorithms:       None.
1309  Data Structures:  None.
1310
1311  Global Variables: delay - decremented if not already at the minimum value.
1312
1313  Author:          Glen George
1314  Last Modified:   Mar. 8, 1994
1315
1316  */
1317
1318  void  trg_delay_down()
1319  {
1320      /* variables */
1321      /* none */
1322
1323
1324
1325      /* decrease the trigger delay, if not already the minimum */
1326      if (delay > MIN_DELAY)
1327          delay--;
1328
1329      /* set the trigger delay for the hardware */
1330      set_delay(delay);
1331
1332
1333      /* all done with lowering the trigger delay - return */
1334      return;
1335
1336  }
1337
1338
1339
1340
1341  /*
1342  trg_delay_up
1343
1344  Description:      This function handles increasing the current trigger
1345                    delay. The new trigger delay is sent to the hardware.
1346                    If an attempt is made to raise the trigger delay above
1347                    the maximum value it is not changed.
1348
1349  Arguments:        None.
1350  Return Value:     None.

```

```

1351
1352 Input:          None.
1353 Output:         None.
1354
1355 Error Handling:  None.
1356
1357 Algorithms:     None.
1358 Data Structures: None.
1359
1360 Global Variables: delay - incremented if not already the maximum value.
1361
1362 Author:         Glen George
1363 Last Modified:  Mar. 8, 1994
1364
1365 */
1366
1367 void trg_delay_up()
1368 {
1369     /* variables */
1370     /* none */
1371
1372
1373
1374     /* increase the trigger delay, if not already the maximum */
1375     if (delay < MAX_DELAY)
1376         delay++;
1377
1378     /* tell the hardware the new trigger delay */
1379     set_delay(delay);
1380
1381
1382     /* all done raising the trigger delay - return */
1383     return;
1384
1385 }
1386
1387
1388
1389
1390 /*
1391 adjust_trg_delay
1392
1393 Description:      This function adjusts the trigger delay for a new sweep
1394                   rate. The factor to adjust the delay by is determined
1395                   by looking up the sample rates in the sweep_rates array.
1396                   If the delay goes out of range, due to the adjustment it
1397                   is reset to the maximum or minimum valid value.
1398
1399 Arguments:        old_sweep (int) - old sweep rate (index into sweep_rates
1400                           array).
1401                   new_sweep (int) - new sweep rate (index into sweep_rates
1402                           array).
1403 Return Value:     None.
1404
1405 Input:           None.
1406 Output:          None.
1407
1408 Error Handling:   None.
1409
1410 Algorithms:       The delay is multiplied by 10 times the ratio of the
1411                   sweep sample rates then divided by 10. This is done to
1412                   avoid floating point arithmetic and integer truncation
1413                   problems.
1414 Data Structures:  None.
1415
1416 Global Variables: delay - adjusted based on passed sweep rates.
1417
1418 Known Bugs:       The updated delay time is not displayed. Since the time
1419                   is typically only rounded to the new sample rate, this is
1420                   not a major problem.
1421
1422 Author:          Glen George
1423 Last Modified:   Mar. 8, 1994
1424
1425 */

```

```

1426 static void adjust_trg_delay(int old_sweep, int new_sweep)
1427 {
1428     /* variables */
1429     /* none */
1430
1431
1432
1433
1434     /* multiply by 10 times the ratio of sweep rates */
1435     delay *= (10 * sweep_rates[new_sweep].sample_rate) / sweep_rates[old_sweep].sample_rate;
1436     /* now divide the factor of 10 back out */
1437     delay /= 10;
1438
1439     /* make sure delay is not out of range */
1440     if (delay > MAX_DELAY)
1441         /* delay is too large - set to maximum */
1442         delay = MAX_DELAY;
1443     if (delay < MIN_DELAY)
1444         /* delay is too small - set to minimum */
1445         delay = MIN_DELAY;
1446
1447
1448     /* tell the hardware the new trigger delay */
1449     set_delay(delay);
1450
1451
1452     /* all done adjusting the trigger delay - return */
1453     return;
1454 }
1455
1456
1457
1458
1459
1460 /*
1461 display_trg_delay
1462
1463 Description:      This function displays the current trigger delay at the
1464                  passed position, in the passed style.
1465
1466 Arguments:      x_pos (int) - x position (in character cells) at which to
1467                  display the trigger delay.
1468                  y_pos (int) - y position (in character cells) at which to
1469                  display the trigger delay.
1470                  style (int) - style with which to display the trigger
1471                  delay.
1472 Return Value:    None.
1473
1474 Input:           None.
1475 Output:          The trigger delay is displayed at the passed position on
1476                  the display.
1477
1478 Error Handling:  None.
1479
1480 Algorithms:      None.
1481 Data Structures: None.
1482
1483 Global Variables: delay - determines the value displayed.
1484
1485 Author:          Glen George
1486 Last Modified:   June 11, 2014
1487
1488 */
1489
1490 void display_trg_delay(int x_pos, int y_pos, int style)
1491 {
1492     /* variables */
1493     char    delay_str[] = "          "; /* string containing the trigger delay */
1494     long int units_adj; /* adjustment to get to microseconds */
1495
1496     long int d; /* delay in appropriate units */
1497     float    temp_d; /* delay in float to avoid overflows */
1498
1499     /* compute the delay in the appropriate units */
1500     /* have to watch out for overflow, so use float temp */

```

```

1501 if (sweep_rates[sweep].sample_rate > 1000000L) {
1502     /* have a fast sweep rate */
1503     /* first compute with float to avoid overflow */
1504     temp_d = delay * (1000000000L / sweep_rates[sweep].sample_rate);
1505
1506     /* now convert to int */
1507     d = (int) temp_d;
1508     /* need to divide by 1000 to get to microseconds */
1509     units_adj = 1000;
1510 }
1511 else {
1512     /* slow sweep rate, don't have to worry about overflow */
1513     d = delay * (1000000L / sweep_rates[sweep].sample_rate);
1514     /* already in microseconds, so adjustment is 1 */
1515     units_adj = 1;
1516 }
1517
1518 /* convert it to the string (leave first character blank) */
1519 cvt_num_field(d, &delay_str[1]);
1520
1521 /* add in the units */
1522 if (((d / units_adj) < 1000) && ((d / units_adj) > -1000) && (units_adj == 1000)) {
1523     /* delay is in microseconds */
1524     delay_str[7] = '\004';
1525     delay_str[8] = 's';
1526 }
1527 else if (((d / units_adj) < 1000000) && ((d / units_adj) > -1000000)) {
1528     /* delay is in milliseconds */
1529     delay_str[7] = 'm';
1530     delay_str[8] = 's';
1531 }
1532 else if (((d / units_adj) < 1000000000) && ((d / units_adj) > -1000000000)) {
1533     /* delay is in seconds */
1534     delay_str[7] = 's';
1535     delay_str[8] = ' ';
1536 }
1537 else {
1538     /* delay is in kiloseconds */
1539     delay_str[7] = 'k';
1540     delay_str[8] = 's';
1541 }
1542
1543
1544 /* now actually display the trigger delay */
1545 plot_string(x_pos, y_pos, delay_str, style);
1546
1547
1548 /* all done displaying the trigger delay - return */
1549 return;
1550
1551 }
1552
1553
1554
1555
1556 /*
1557 cvt_num_field
1558
1559 Description:      This function converts the passed number (numeric field
1560                   value) to a string and returns that in the passed string
1561                   reference. The number may be signed, and a sign (+ or -)
1562                   is always generated. The number is assumed to have three
1563                   digits to the right of the decimal point. Only the four
1564                   most significant digits of the number are displayed and
1565                   the decimal point is shifted appropriately. (Four digits
1566                   are always generated by the function).
1567
1568 Arguments:      n (long int) - numeric field value to convert.
1569                 s (char *) - pointer to string in which to return the
1570                   converted field value.
1571
1572 Return Value:   None.
1573
1574 Input:          None.
1575 Output:         None.

```

```

1576 Error Handling:   None.
1577
1578 Algorithms:       The algorithm used assumes four (4) digits are being
1579                   converted.
1580 Data Structures:  None.
1581
1582 Global Variables: None.
1583
1584 Known Bugs:       If the passed long int is the largest negative long int,
1585                   the function will display garbage.
1586
1587 Author:           Glen George
1588 Last Modified:    Mar. 8, 1994
1589
1590 */
1591
1592 static void cvt_num_field(long int n, char *s)
1593 {
1594     /* variables */
1595     int  dp = 3;          /* digits to right of decimal point */
1596     int  d;               /* digit weight (power of 10) */
1597
1598     int  i = 0;           /* string index */
1599
1600
1601
1602     /* first get the sign (and make n positive for conversion) */
1603     if (n < 0) {
1604         /* n is negative, set sign and convert to positive */
1605         s[i++] = '-';
1606         n = -n;
1607     }
1608     else {
1609         /* n is positive, set sign only */
1610         s[i++] = '+';
1611     }
1612
1613
1614     /* make sure there are no more than 4 significant digits */
1615     while (n > 9999) {
1616         /* have more than 4 digits - get rid of one */
1617         n /= 10;
1618         /* adjust the decimal point */
1619         dp--;
1620     }
1621
1622     /* if decimal point is non-positive, make positive */
1623     /* (assume will take care of adjustment with output units in this case) */
1624     while (dp <= 0)
1625         dp += 3;
1626
1627
1628     /* adjust dp to be digits to the right of the decimal point */
1629     /* (assuming 4 digits) */
1630     dp = 4 - dp;
1631
1632
1633     /* finally, loop getting and converting digits */
1634     for (d = 1000; d > 0; d /= 10) {
1635
1636         /* check if need decimal the decimal point now */
1637         if (dp-- == 0)
1638             /* time for decimal point */
1639             s[i++] = '.';
1640
1641         /* get and convert this digit */
1642         s[i++] = (n / d) + '0';
1643         /* remove this digit from n */
1644         n %= d;
1645     }
1646
1647
1648     /* all done converting the number, return */
1649     return;
1650

```

1651 }
1652