```
/*****************************************************************************/
/*                                                                           */
/*                                 TRACUTIL                                  */
/*                          Trace Utility Functions                          */
/*                        Digital Oscilloscope Project                       */
/*                                 EE/CS 52                                  */
/*                                                                           */
/*****************************************************************************/

/*
   This file contains the utility functions for handling traces (capturing
   and displaying data) for the Digital Oscilloscope project.  The functions
   included are:
      clear_saved_areas  - clear all the save areas
      do_trace           - start a trace
      init_trace         - initialize the trace routines
      plot_trace         - plot a trace (sampled data)
      restore_menu_trace - restore the saved area under the menus
      restore_trace      - restore the saved area of a trace
      set_display_scale  - set the type of displayed scale (and display it)
      set_mode           - set the triggering mode
      set_save_area      - determine an area of a trace to save
      set_trace_size     - set the number of samples in a trace
      trace_done         - inform this module that a trace has been completed
      trace_rdy          - determine if system is ready to start another trace
      trace_rearm        - re-enable tracing (in one-shot triggering mode)

   The local functions included are:
      none

   The locally global variable definitions included are:
      cur_scale    - current scale type
      sample_size  - the size of the sample for the trace
      sampling     - currently doing a sample
      saved_area   - saved trace under a specified area
      saved_axis_x - saved trace under the x lines (axes or grid)
      saved_axis_y - saved trace under the y lines (axes or grid)
      saved_menu   - saved trace under the menu
      saved_pos_x  - starting position (x coorindate) of area to save
      saved_pos_y  - starting position (y coorindate) of area to save
      saved_end_x  - ending position (x coorindate) of area to save
      saved_end_y  - ending position (y coorindate) of area to save
      trace_status - whether or not ready to start another trace


   Revision History
      3/8/94   Glen George       Initial revision.
      3/13/94  Glen George       Updated comments.
      3/13/94  Glen George       Fixed inversion of signal in plot_trace.
      3/13/94  Glen George       Added sampling flag and changed the functions
                           init_trace, do_trace and trace_done to update
                 the flag.  Also the function trace_rdy now
                 uses it.  The function set_mode was updated
                 to always say a trace is ready for normal
                 triggering.
      3/13/94  Glen George       Fixed bug in trace restoring due to operator
                      misuse (&& instead of &) in the functions
                 set_axes, restore_menu_trace, and
                 restore_trace.
      3/13/94  Glen George       Fixed bug in trace restoring due to the clear
                      function (clear_saved_areas) not clearing all
                 of the menu area.
      3/13/94  Glen George       Fixed comparison bug when saving traces in
                      plot_trace.
      3/13/94  Glen George       Changed name of set_axes to set_display_scale
                      and the name of axes_state to cur_scale to
                 more accurately reflect the function/variable
                 use (especially if add scale display types).
      3/17/97  Glen George       Updated comments.
      3/17/97  Glen George       Changed set_display_scale to use plot_hline
                      and plot_vline functions to output axes.
      5/3/06   Glen George       Updated formatting.
      5/9/06   Glen George       Updated do_trace function to match the new
                                 definition of start_sample().
      5/9/06   Glen George       Removed normal_trg variable, its use is now
```

```
                                handled by the get_trigger_mode() accessor.
      5/9/06   Glen George       Added tick marks to the axes display.
      5/9/06   Glen George       Added ability to display a grid.
      5/27/08  Glen George       Added is_sampling() function to be able to
                                tell if the system is currently taking a
                  sample.
      5/27/08  Glen George       Changed set_mode() to always turn off the
                                sampling flag so samples with the old mode
                                setting are ignored.
      6/3/08   Glen George       Fixed problems with non-power of 2 display
                  sizes not working.
      6/3/14   Santiago Navonne  Changed UI display colors; changed plot_trace
                                to clear just trace instead of whole display.
*/



/* library include files */
  /* none */

/* local include files */
#include  "scopedef.h"
#include  "lcdout.h"
#include  "menu.h"
#include  "menuact.h"
#include  "tracutil.h"




/* locally global variables */

static int  trace_status;   /* ready to start another trace */

static int  sampling;           /* currently sampling data */

static int  sample_size;    /* number of data points in a sample */

static int old_sample[SIZE_X]; /* sample currently being displayed */

static enum scale_type  cur_scale;  /* current display scale type */

/* traces (sampled data) saved under the axes */
static unsigned char  saved_axis_x[2 * Y_TICK_CNT + 1][PLOT_SIZE_X/8];  /* saved trace under x lines */
static unsigned char  saved_axis_y[2 * X_TICK_CNT + 1][PLOT_SIZE_Y/8];  /* saved trace under y lines */

/* traces (sampled data) saved under the menu */
static unsigned char  saved_menu[MENU_SIZE_Y][(MENU_SIZE_X + 7)/8];

/* traces (sampled data) saved under any area */
static unsigned char  saved_area[SAVE_SIZE_Y][SAVE_SIZE_X/8]; /* saved trace under any area */
static int        saved_pos_x;    /* starting x position of saved area */
static int        saved_pos_y;    /* starting y position of saved area */
static int        saved_end_x;    /* ending x position of saved area */
static int        saved_end_y;    /* ending y position of saved area */




/*
   init_trace

   Description:      This function initializes all of the locally global
                     variables used by these routines.  The saved areas are
             set to non-existant with cleared saved data.  Normal
             normal triggering is set, the system is ready for a
             trace, the scale is turned off and the sample size is set
             to the screen size.

   Arguments:        None.
   Return Value:     None.

   Input:            None.
   Output:           None.
```

```
   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: trace_status - set to TRUE.
             sampling     - set to FALSE.
             cur_scale    - set to SCALE_NONE (no displayed scale).
             sample_size  - set to screen size (SIZE_X).
             saved_axis_x - cleared.
             saved_axis_y - cleared.
             saved_menu   - cleared.
             saved_area   - cleared.
             saved_pos_x  - set to off-screen.
             saved_pos_y  - set to off-screen.
             saved_end_x  - set to off-screen.
             saved_end_y  - set to off-screen.

   Author:           Glen George
   Last Modified:    May 9, 2006

*/

void  init_trace()
{
    /* variables */
      /* none */



    /* initialize system status variables */

    /* ready for a trace */
    trace_status = TRUE;

    /* not currently sampling data */
    sampling = FALSE;

    /* turn off the displayed scale */
    cur_scale = SCALE_NONE;

    /* sample size is the screen size */
    sample_size = SIZE_X;


    /* clear save areas */
    clear_saved_areas();

    /* also clear the general saved area location variables (off-screen) */
    saved_pos_x = SIZE_X + 1;
    saved_pos_y = SIZE_Y + 1;
    saved_end_x = SIZE_X + 1;
    saved_end_y = SIZE_Y + 1;


    /* done initializing, return */
    return;

}




/*
   set_mode

   Description:      This function sets the locally global triggering mode
                     based on the passed value (one of the possible enumerated
             values).  The triggering mode is used to determine when
             the system is ready for another trace.  The sampling flag
                     is also reset so a new sample will be started (if that is
                     appropriate).

   Arguments:        trigger_mode (enum trigger_type) - the mode with which to
                            set the triggering.
```

```
   Return Value:     None.

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: sampling     - set to FALSE to turn off sampling
                     trace_status - set to TRUE if not one-shot triggering.

   Author:           Glen George
   Last Modified:    May 27, 2008

*/

void  set_mode(enum trigger_type trigger_mode)
{
    /* variables */
      /* none */



    /* if not one-shot triggering - ready for trace too */
    trace_status = (trigger_mode != ONESHOT_TRIGGER);


    /* turn off the sampling flag so will start a new sample */
    sampling = FALSE;


    /* all done, return */
    return;

}




/*
   is_sampling

   Description:      This function determines whether the system is currently
                     taking a sample or not.  This is just the value of the
              sampling flag.

   Arguments:        None.
   Return Value:     (int) - the current sampling status (TRUE if currently
              trying to take a sample, FALSE otherwise).

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: sampling - determines if taking a sample or not.

   Author:           Glen George
   Last Modified:    May 27, 2008

*/

int  is_sampling()
{
    /* variables */
      /* none */



    /* currently sampling if sampling flag is set */
```

```
    return  sampling;

}



/*
   trace_rdy

   Description:      This function determines whether the system is ready to
                     start another trace.  This is determined by whether or
             not the system is still sampling (sampling flag) and if
             it is ready for another trace (trace_status flag).

   Arguments:        None.
   Return Value:     (int) - the current trace status (TRUE if ready to do
             another trace, FALSE otherwise).

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: sampling     - determines if ready for another trace.
             trace_status - determines if ready for another trace.

   Author:           Glen George
   Last Modified:    Mar. 13, 1994

*/

int  trace_rdy()
{
    /* variables */
      /* none */



    /* ready for another trace if not sampling and trace is ready */
    return  (!sampling && trace_status);

}



/*
   trace_done

   Description:      This function is called to indicate a trace has been
                     completed.  If in normal triggering mode this means the
             system is ready for another trace.

   Arguments:        None.
   Return Value:     None.

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: trace_status - may be set to TRUE.
             sampling     - set to FALSE.

   Author:           Glen George
   Last Modified:    May 9, 2006

*/
```

```
376
377 void  trace_done()
378 {
379     /* variables */
380       /* none */
381
382
383
384     /* done with a trace - if retriggering, ready for another one */
385     if (get_trigger_mode() != ONESHOT_TRIGGER)
386         /* in a retriggering mode - set trace_status to TRUE (ready) */
387     trace_status = TRUE;
388
389     /* no longer sampling data */
390     sampling = FALSE;
391
392
393     /* done so return */
394     return;
395
396 }
397
398
399
400
401 /*
402    trace_rearm
403
404    Description:      This function is called to rearm the trace.  It sets the
405                      trace status to ready (TRUE).  It is used to rearm the
406              trigger in one-shot mode.
407
408    Arguments:        None.
409    Return Value:     None.
410
411    Input:            None.
412    Output:           None.
413
414    Error Handling:   None.
415
416    Algorithms:       None.
417    Data Structures:  None.
418
419    Global Variables: trace_status - set to TRUE.
420
421    Author:           Glen George
422    Last Modified:    Mar. 8, 1994
423
424 */
425
426 void  trace_rearm()
427 {
428     /* variables */
429       /* none */
430
431
432
433     /* rearm the trace - set status to ready (TRUE) */
434     trace_status = TRUE;
435
436
437     /* all done - return */
438     return;
439
440 }
441
442
443
444
445 /*
446    set_trace_size
447
448    Description:      This function sets the locally global sample size to the
449                      passed value.  This is used to scale the data when
450              plotting a trace.
```

```
451
452     Arguments:        size (int) - the trace sample size.
453     Return Value:     None.
454
455     Input:            None.
456     Output:           None.
457
458     Error Handling:   None.
459
460     Algorithms:       None.
461     Data Structures:  None.
462
463     Global Variables: sample_size - set to the passed value.
464
465     Author:           Glen George
466     Last Modified:    Mar. 8, 1994
467
468 */
469
470 void  set_trace_size(int size)
471 {
472     /* variables */
473       /* none */
474
475
476
477     /* set the locally global sample size */
478     sample_size = size;
479
480
481     /* all done, return */
482     return;
483
484 }
485
486
487
488
489 /*
490     set_display_scale
491
492     Description:      This function sets the displayed scale type to the passed
493               argument.  If the scale is turned on, it draws it.  If it
494               is turned off (SCALE_NONE), it restores the saved trace
495               under the scale.  Scales can be axes with tick marks
496                       (SCALE_AXES) or a grid (SCALE_GRID).
497
498     Arguments:        scale (scale_type) - new scale type.
499     Return Value:     None.
500
501     Input:            None.
502     Output:           Either a scale is output or the trace under the old scale
503               is restored.
504
505     Error Handling:   None.
506
507     Algorithms:       None.
508     Data Structures:  None.
509
510     Global Variables: cur_scale    - set to the passed value.
511               saved_axis_x - used to restore trace data under x-axis.
512               saved_axis_y - used to restore trace data under y-axis.
513
514     Author:           Glen George
515     Last Modified:   June 03, 2014
516
517 */
518
519 void  set_display_scale(enum scale_type scale)
520 {
521     /* variables */
522     int  p;             /* x or y coordinate */
523
524     int  i;     /* loop indices */
525     int  j;
```

```
526
527
528
529     /* whenever change scale type, need to clear out previous scale */
530     /* unnecessary if going to SCALE_GRID or from SCALE_NONE or not changing the scale */
531     if ((scale != SCALE_GRID) && (cur_scale != SCALE_NONE) && (scale != cur_scale))  {
532
533         /* need to restore the trace under the lines (tick, grid, or axis) */
534
535     /* go through all points on horizontal lines */
536     for (j = -Y_TICK_CNT; j <= Y_TICK_CNT; j++)  {
537
538         /* get y position of the line */
539         p = X_AXIS_POS + j * Y_TICK_SIZE;
540         /* make sure it is in range */
541         if (p >= PLOT_SIZE_Y)
542             p = PLOT_SIZE_Y - 1;
543         if (p < 0)
544             p = 0;
545
546         /* look at entire horizontal line */
547         for (i = 0; i < PLOT_SIZE_X; i++)  {
548             /* check if this point is on or off (need to look at bits) */
549         if ((saved_axis_x[j + Y_TICK_CNT][i / 8] & (0x80 >> (i % 8))) == 0)
550             /* saved pixel is off */
551             plot_pixel(i, p, PIXEL_CLEAR);
552         else
553             /* saved pixel is on */
554             plot_pixel(i, p, PIXEL_TRACE);
555         }
556     }
557
558     /* go through all points on vertical lines */
559     for (j = -X_TICK_CNT; j <= X_TICK_CNT; j++)  {
560
561         /* get x position of the line */
562         p = Y_AXIS_POS + j * X_TICK_SIZE;
563         /* make sure it is in range */
564         if (p >= PLOT_SIZE_X)
565             p = PLOT_SIZE_X - 1;
566         if (p < 0)
567             p = 0;
568
569         /* look at entire vertical line */
570         for (i = 0; i < PLOT_SIZE_Y; i++)  {
571             /* check if this point is on or off (need to look at bits) */
572         if ((saved_axis_y[j + X_TICK_CNT][i / 8] & (0x80 >> (i % 8))) == 0)
573             /* saved pixel is off */
574             plot_pixel(p, i, PIXEL_CLEAR);
575         else
576             /* saved pixel is on */
577             plot_pixel(p, i, PIXEL_TRACE);
578         }
579     }
580     }
581
582
583     /* now handle the scale type appropriately */
584     switch (scale)  {
585
586         case SCALE_AXES:    /* axes for the scale */
587         case SCALE_GRID:    /* grid for the scale */
588
589                     /* draw x lines (grid or tick marks) */
590                 for (i = -Y_TICK_CNT; i <= Y_TICK_CNT; i++)  {
591
592                 /* get y position of the line */
593                 p = X_AXIS_POS + i * Y_TICK_SIZE;
594                 /* make sure it is in range */
595                 if (p >= PLOT_SIZE_Y)
596                     p = PLOT_SIZE_Y - 1;
597                 if (p < 0)
598                     p = 0;
599
600                 /* should we draw a grid, an axis, or a tick mark */
```

```
                if (scale == SCALE_GRID)
                    /* drawing a grid line */
                        plot_hline(X_GRID_START, p, (X_GRID_END - X_GRID_START));
                else if (i == 0)
                    /* drawing the x axis */
                        plot_hline(X_AXIS_START, p, (X_AXIS_END - X_AXIS_START));
                else
                    /* must be drawing a tick mark */
                        plot_hline((Y_AXIS_POS - (TICK_LEN / 2)), p, TICK_LEN);
                }

                    /* draw y lines (grid or tick marks) */
                for (i = -X_TICK_CNT; i <= X_TICK_CNT; i++)  {

                /* get x position of the line */
                p = Y_AXIS_POS + i * X_TICK_SIZE;
                /* make sure it is in range */
                if (p >= PLOT_SIZE_X)
                    p = PLOT_SIZE_X - 1;
                    if (p < 0)
                    p = 0;

                /* should we draw a grid, an axis, or a tick mark */
                if (scale == SCALE_GRID)
                    /* drawing a grid line */
                        plot_vline(p, Y_GRID_START, (Y_GRID_END - Y_GRID_START));
                else if (i == 0)
                    /* drawing the y axis */
                        plot_vline(p, Y_AXIS_START, (Y_AXIS_END - Y_AXIS_START));
                else
                    /* must be drawing a tick mark */
                        plot_vline(p, (X_AXIS_POS - (TICK_LEN / 2)), TICK_LEN);
                }

                /* done with the axes */
                break;

        case SCALE_NONE:    /* there is no scale */
                /* already restored plot so nothing to do */
                break;

    }


    /* now remember the new (now current) scale type */
    cur_scale = scale;


    /* scale is taken care of, return */
    return;

}




/*
   clear_saved_areas

   Description:      This function clears all the saved areas (for saving the
                     trace under the axes, menus, and general areas).

   Arguments:        None.
   Return Value:     None.

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: saved_axis_x - cleared.
             saved_axis_y - cleared.
```

```
             saved_menu   - cleared.
             saved_area   - cleared.

   Author:           Glen George
   Last Modified:    May 9, 2006

*/

void  clear_saved_areas()
{
    /* variables */
    int  i;     /* loop indices */
    int  j;



    /* clear x-axis and y-axis save areas */
    for (j = 0; j <= (2 * Y_TICK_CNT); j++)
        for (i = 0; i < (SIZE_X / 8); i++)
            saved_axis_x[j][i] = 0;
    for (j = 0; j <= (2 * X_TICK_CNT); j++)
        for (i = 0; i < (SIZE_Y / 8); i++)
            saved_axis_y[j][i] = 0;

    /* clear the menu save ares */
    for (i = 0; i < MENU_SIZE_Y; i++)
        for (j = 0; j < ((MENU_SIZE_X + 7) / 8); j++)
        saved_menu[i][j] = 0;

    /* clear general save area */
    for (i = 0; i < SAVE_SIZE_Y; i++)
        for (j = 0; j < (SAVE_SIZE_X / 8); j++)
        saved_area[i][j] = 0;


    /* done clearing the saved areas - return */
    return;

}




/*
   restore_menu_trace

   Description:      This function restores the trace under the menu when the
                     menus are turned off.  (The trace was previously saved.)

   Arguments:        None.
   Return Value:     None.

   Input:            None.
   Output:           The trace under the menu is restored to the LCD screen.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: saved_menu - used to restore trace data under the menu.

   Author:           Glen George
   Last Modified:    June 03, 2014

*/

void  restore_menu_trace()
{
    /* variables */
    int  bit_position;  /* position of bit to restore (in saved data) */
    int  bit_offset;    /* offset (in bytes) of bit within saved row */

    int  x;     /* loop indices */
    int  y;
```

```
751
752
753
754     /* loop, restoring the trace under the menu */
755     for (y = MENU_UL_Y; y < (MENU_UL_Y + MENU_SIZE_Y); y++)  {
756
757         /* starting a row - initialize bit position */
758     bit_position = 0x80;    /* start at high-order bit in the byte */
759     bit_offset = 0;     /* first byte of the row */
760
761         for (x = MENU_UL_X; x < (MENU_UL_X + MENU_SIZE_X); x++)  {
762
763         /* check if this point is on or off (need to look at bits) */
764         if ((saved_menu[y - MENU_UL_Y][bit_offset] & bit_position) == 0)
765             /* saved pixel is off */
766         plot_pixel(x, y, PIXEL_CLEAR);
767         else
768             /* saved pixel is on */
769         plot_pixel(x, y, PIXEL_TRACE);
770
771         /* move to the next bit position */
772         bit_position >>= 1;
773         /* check if moving to next byte */
774         if (bit_position == 0)  {
775             /* now on high bit of next byte */
776         bit_position = 0x80;
777         bit_offset++;
778         }
779         }
780     }
781
782
783     /* restored menu area - return */
784     return;
785
786 }
787
788
789
790
791 /*
792    set_save_area
793
794    Description:      This function sets the position and size of the area to
795                      be saved when traces are drawn.  It also clears any data
796              currently saved.
797
798    Arguments:        pos_x (int)  - x position of upper left corner of the
799                     saved area.
800              pos_y (int)  - y position of upper left corner of the
801                          saved area.
802              size_x (int) - horizontal size of the saved area.
803              size_y (int) - vertical size of the saved area.
804    Return Value:     None.
805
806    Input:            None.
807    Output:           None.
808
809    Error Handling:   None.
810
811    Algorithms:       None.
812    Data Structures:  None.
813
814    Global Variables: saved_area  - cleared.
815              saved_pos_x - set to passed value.
816              saved_pos_y - set to passed value.
817              saved_end_x - computed from passed values.
818              saved_end_y - computed from passed values.
819
820    Author:           Glen George
821    Last Modified:    Mar. 8, 1994
822
823 */
824
825 void  set_save_area(int pos_x, int pos_y, int size_x, int size_y)
```

```
{
    /* variables */
    int  x;     /* loop indices */
    int  y;



    /* just setup all the locally global variables from the passed values */
    saved_pos_x = pos_x;
    saved_pos_y = pos_y;
    saved_end_x = pos_x + size_x;
    saved_end_y = pos_y + size_y;


    /* clear the save area */
    for (y = 0; y < SAVE_SIZE_Y; y++)  {
        for (x = 0; x < (SAVE_SIZE_X / 8); x++)  {
        saved_area[y][x] = 0;
        }
    }


    /* setup the saved area - return */
    return;

}




/*
   restore_trace

   Description:      This function restores the trace under the set saved
                     area.  (The area was previously set and the trace was
             previously saved.)

   Arguments:        None.
   Return Value:     None.

   Input:            None.
   Output:           The trace under the saved ares is restored to the LCD.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: saved_area  - used to restore trace data.
             saved_pos_x - gives starting x position of saved area.
             saved_pos_y - gives starting y position of saved area.
             saved_end_x - gives ending x position of saved area.
             saved_end_y - gives ending y position of saved area.

   Author:           Glen George
   Last Modified:    June 03, 2014

*/

void  restore_trace()
{
    /* variables */
    int  bit_position;  /* position of bit to restore (in saved data) */
    int  bit_offset;    /* offset (in bytes) of bit within saved row */

    int  x;     /* loop indices */
    int  y;



    /* loop, restoring the saved trace */
    for (y = saved_pos_y; y < saved_end_y; y++)  {

        /* starting a row - initialize bit position */
    bit_position = 0x80;    /* start at high-order bit in the byte */
```

```
    bit_offset = 0;     /* first byte of the row */

        for (x = saved_pos_x; x < saved_end_x; x++)  {

        /* check if this point is on or off (need to look at bits) */
        if ((saved_area[y - saved_pos_y][bit_offset] & bit_position) == 0)
            /* saved pixel is off */
        plot_pixel(x, y, PIXEL_CLEAR);
        else
            /* saved pixel is on */
        plot_pixel(x, y, PIXEL_TRACE);

        /* move to the next bit position */
        bit_position >>= 1;
        /* check if moving to next byte */
        if (bit_position == 0)  {
            /* now on high bit of next byte */
        bit_position = 0x80;
        bit_offset++;
        }
        }
    }


    /* restored the saved area - return */
    return;

}



/*
   do_trace

   Description:      This function starts a trace.  It starts the hardware
                     sampling data (via a function call) and sets the trace
              ready flag (trace_status) to FALSE and the sampling flag
              (sampling) to TRUE.

   Arguments:        None.
   Return Value:     None.

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: trace_status - set to FALSE (not ready for another trace).
              sampling     - set to TRUE (doing a sample now).

   Author:           Glen George
   Last Modified:    Mar. 13, 1994

*/

void  do_trace()
{
    /* variables */
      /* none */



    /* start up the trace */
    /* indicate whether using automatic triggering or not */
    start_sample(get_trigger_mode() == AUTO_TRIGGER);

    /* now not ready for another trace (currently doing one) */
    trace_status = FALSE;

    /* and are currently sampling data */
    sampling = TRUE;
```

```
976
977
978     /* trace is going, return */
979     return;
980
981 }
982
983
984 /*
985    plot_trace
986
987    Description:      This function plots the passed trace.  The trace is
988                      assumed to contain sample_size points of sampled data.
989                      Any points falling within any of the save areas are also
990                      saved by this routine.  The data is also scaled to be
991                      within the range of the entire screen.
992
993
994    Arguments:        sample (unsigned char far *) - sample to plot.
995    Return Value:     None.
996
997    Input:            None.
998    Output:           The sample is plotted on the screen.
999
1000   Error Handling:   None.
1001
1002   Algorithms:       If there are more sample points than screen width the
1003             sample is plotted with multiple points per horizontal
1004             position.
1005   Data Structures:  None.
1006
1007   Global Variables: cur_scale    - determines type of scale to plot.
1008             sample_size  - determines size of passed sample.
1009             saved_axis_x - stores trace under x-axis.
1010             saved_axis_y - stores trace under y-axis.
1011             saved_menu   - stores trace under the menu.
1012             saved_area   - stores trace under the saved area.
1013             saved_pos_x  - determines location of saved area.
1014             saved_pos_y  - determines location of saved area.
1015             saved_end_x  - determines location of saved area.
1016             saved_end_y  - determines location of saved area.
1017
1018   Author:           Glen George
1019   Last Modified:    June 03, 2014
1020
1021 */
1022
1023 void  plot_trace(unsigned char *sample)
1024 {
1025     /* variables */
1026     int  x = 0;             /* current x position to plot */
1027     int  x_pos = (PLOT_SIZE_X / 2); /* "fine" x position for multiple point plotting */
1028
1029     int  y;             /* y position of point to plot */
1030
1031     int  p;                             /* an x or y coordinate */
1032
1033     int  i;             /* loop indices */
1034     int  j;
1035
1036
1037     /* clear the saved areas too */
1038     clear_saved_areas();
1039
1040     /* re-display the menu (if it was on) */
1041     refresh_menu();
1042
1043
1044     /* plot the sample */
1045     for (i = 0; i < sample_size; i++)  {
1046
1047         /* determine y position of point (note: screen coordinates invert) */
1048     y = (PLOT_SIZE_Y - 1) - ((sample[i] * (PLOT_SIZE_Y - 1)) / 255);
1049
1050     /* clear previous point on trace */
```

```
    plot_pixel(i, old_sample[i], PIXEL_CLEAR);

        /* plot this point */
    plot_pixel(x, y, PIXEL_TRACE);

    /* and save new value */
    old_sample[i] = y;


    /* check if the point is in a save area */

    /* check if in the menu area */
    if ((x >= MENU_UL_X) && (x < (MENU_UL_X + MENU_SIZE_X)) &&
        (y >= MENU_UL_Y) && (y < (MENU_UL_Y + MENU_SIZE_Y)))
        /* point is in the menu area - save it */
        saved_menu[y - MENU_UL_Y][(x - MENU_UL_X)/8] |= (0x80 >> ((x - MENU_UL_X) % 8));

    /* check if in the saved area */
    if ((x >= saved_pos_x) && (x <= saved_end_x) && (y >= saved_pos_y) && (y <= saved_end_y))
        /* point is in the save area - save it */
        saved_area[y - saved_pos_y][(x - saved_pos_x)/8] |= (0x80 >> ((x - saved_pos_x) % 8));

    /* check if on a grid line */
    /* go through all the horizontal lines */
    for (j = -Y_TICK_CNT; j <= Y_TICK_CNT; j++)  {

        /* get y position of the line */
        p = X_AXIS_POS + j * Y_TICK_SIZE;
        /* make sure it is in range */
        if (p >= PLOT_SIZE_Y)
            p = PLOT_SIZE_Y - 1;
        if (p < 0)
            p = 0;

        /* if the point is on this line, save it */
        if (y == p)
        saved_axis_x[j + Y_TICK_CNT][x / 8] |= (0x80 >> (x % 8));
    }

    /* go through all the vertical lines */
    for (j = -X_TICK_CNT; j <= X_TICK_CNT; j++)  {

        /* get x position of the line */
        p = Y_AXIS_POS + j * X_TICK_SIZE;
        /* make sure it is in range */
        if (p >= PLOT_SIZE_X)
            p = PLOT_SIZE_X - 1;
        if (p < 0)
            p = 0;

        /* if the point is on this line, save it */
        if (x == p)
        saved_axis_y[j + X_TICK_CNT][y / 8] |= (0x80 >> (y % 8));
    }


    /* update x position */
    x_pos += PLOT_SIZE_X;
    /* check if at next horizontal position */
    if (x_pos >= sample_size)  {
        /* at next position - update positions */
        x++;
        x_pos -= sample_size;
    }
    }


    /* finally, output the scale if need be */
    set_display_scale(cur_scale);


    /* done with plot, return */
    return;

}
```