

```

1  /*****
2  /*
3  /*          DISPLAY.S          */
4  /*          Display Interface Functions      */
5  /*          Digital Oscilloscope Project     */
6  /*          EE/CS 52                       */
7  /*          Santiago Navonne                */
8  /*
9  /*****
10
11 /*
12 Display interface and control routines for the EE/CS 52 Digital Oscilloscope
13 project. Function definitions are included in this file, and are laid out
14 as follows:
15 - clear_display: Completely clears the display;
16 - clear_trace: Clears the pixels on the display that are the color of the
17 trace;
18 - plot_pixel: Changes the color of the pixel at a given location;
19 - pixel_color: Accesses the color of the pixel currently being displayed at
20 a given location.
21
22
23 Revision History:
24 6/3/14 Santiago Navonne Initial revision.
25 */
26
27 #include "general.h"
28 #include "system.h"
29 #include "interfac.h"
30 #include "display.h"
31
32
33 .section .text /* Code starts here */
34
35
36 /*
37 * clear_display
38 *
39 * Description:      This procedure clears the display, setting the color of every
40 *                  pixel to black immediately.
41 *
42 * Operation:       The procedure loops through every pixel in the display-mapped
43 *                  region of the VRAM, storing 0 (black; clear pixel) into every
44 *                  location.
45 *
46 * Arguments:       None.
47 *
48 * Return Value:    None.
49 *
50 * Local Variables: None.
51 *
52 * Shared Variables: None.
53 *
54 * Global Variables: None.
55 *
56 * Input:           None.
57 *
58 * Output:          Clears every pixel on the display (changes color to black).
59 *
60 * Error Handling:   None.
61 *
62 * Limitations:     None.
63 *
64 * Algorithms:      None.
65 * Data Structures: None.
66 *
67 * Registers Changed: r8, r9, r10, r11, r12.
68 *
69 * Revision History:
70 * 6/03/14 Santiago Navonne Initial revision.
71 *
72 */
73
74 .global clear_display
75 clear_display: /* clear the whole display */
76     MOVHI r8, %hi(VRAM_BASE) /* start at base of VRAM */

```

```

76     ORI      r8, r8, %lo(VRAM_BASE)
77     MOVI     r9, SIZE_X          /* and will loop through all columns */
78     MOVI     r10, SIZE_Y         /* and rows */
79     MOV      r11, r0             /* starting at coordinates (0, 0) */
80     MOV      r12, r0             /* (top left corner) */
81
82 row_loop:                          /* go through an entire row */
83     STWIO    r0, (r8)            /* first clear the current pixel */
84     ADDI     r8, r8, WORD_SIZE   /* then go to next column */
85     ADDI     r11, r11, 1         /* also incrementing the index */
86     BLT      r11, r9, row_loop   /* and if we're still within display, repeat */
87
88 next_row:                          /* move to next row */
89     ADDI     r8, r8, REMAINDER   /* add the remainder to finish up a VRAM row */
90     MOV      r11, r0             /* reset the column index */
91     ADDI     r12, r12, 1         /* and increment the row index */
92     BLT      r12, r10, row_loop  /* if we're still within display, repeat */
93
94     RET                          /* all done, so return */
95
96 /*
97 * clear_trace
98 *
99 * Description:      This procedure clears the trace from the display, changing the
100 *                  color of every pixel that is currently the trace or cursor color
101 *                  to black.
102 *
103 * Operation:        The procedure loops through every pixel in the display-mapped
104 *                  region of the VRAM. For every location, if the current value
105 *                  matches either trace or cursor colors (both part of the trace)
106 *                  the pixel is cleared by storing 0 into that memory location.
107 *
108 * Arguments:        None.
109 *
110 * Return Value:     None.
111 *
112 * Local Variables:  None.
113 *
114 * Shared Variables: None.
115 *
116 * Global Variables: None.
117 *
118 * Input:            None.
119 *
120 * Output:           Clears every trace pixel on the display (sets color to black).
121 *
122 * Error Handling:   None.
123 *
124 * Limitations:      None.
125 *
126 * Algorithms:       None.
127 * Data Structures:  None.
128 *
129 * Registers Changed: r8, r9, r10, r11, r12, r14, r15.
130 *
131 * Revision History:
132 *     6/03/14   Santiago Navonne   Initial revision.
133 *
134 */
135
136 .global clear_trace_old
137 clear_trace_old:                  /* clear all trace pixels on display */
138     MOVHI    r8, %hi(VRAM_BASE) /* start at base of VRAM */
139     ORI      r8, r8, %lo(VRAM_BASE)
140     MOVHI    r13, %hi(PIXEL_TRACE) /* load colors that will be cleared */
141     ORI      r13, r13, %lo(PIXEL_TRACE)
142     MOVHI    r14, %hi(PIXEL_CURSOR) /* which are trace and cursor */
143     ORI      r14, r14, %lo(PIXEL_CURSOR)
144     MOVI     r9, SIZE_X          /* will loop through all columns */
145     MOVI     r10, SIZE_Y         /* and all rows */
146     MOV      r11, r0             /* starting at (0, 0) */
147     MOV      r12, r0             /* (top left corner) */
148
149 trace_check:                      /* check if current pixel is part of trace */
150     LDWIO    r15, (r8)           /* read value from VRAM */
151     BEQ      r13, r15, trace_clear /* definitely clear if color is trace color */

```

```

151
152 cursor_check:                /* check if current pixel is part of cursor */
153     BNE      r14, r15, trace_row_loop /* also clear if part of cursor */
154
155 trace_clear:                  /* pixel is part of trace or cursor */
156     STWIO    r0, (r8)         /* so clear it */
157
158 trace_row_loop:              /* done with current pixel */
159     ADDI     r8, r8, WORD_SIZE /* so go to next */
160     ADDI     r11, r11, 1       /* and also increment column index */
161     BLT      r11, r9, trace_check /* if still within display, repeat */
162
163 trace_next_row:              /* done with current row */
164     ADDI     r8, r8, REMAINDER /* add remainder to finish up VRAM row */
165     MOV      r11, r0          /* reset column index */
166     ADDI     r12, r12, 1      /* and increment row index */
167     BLT      r12, r10, trace_check /* if still within display, repeat */
168
169     RET                          /* all done, so return */
170
171
172 /*
173  * plot_pixel
174  *
175  * Description:      This procedure changes the color to the pixel at the passed x, y
176  *                  coordinates, where the top left corner is (0, 0), to the passed
177  *                  color. Colors are specified with a 24-bit value, where the bottom
178  *                  8 bits represent the amount of blue, the following 8 the amount
179  *                  of green, and the next 8 the amount of red.
180  *
181  * Operation:        The function simply translates the x and y coordinates into a VRAM
182  *                  address by setting the top bits to the offset of the VRAM, and ORing
183  *                  in the shifted row and column indices. Then, it stores the passed
184  *                  color value at that address.
185  *
186  * Arguments:        x - x coordinate of the pixel, where leftmost column is 0 (r4).
187  *                  y - y coordinate of the pixel, where top row is 0 (r5).
188  *                  color - 24-bit value with RGB color the pixel should change to (r6).
189  *
190  * Return Value:     None.
191  *
192  * Local Variables:  None.
193  *
194  * Shared Variables: None.
195  *
196  * Global Variables: None.
197  *
198  * Input:            None.
199  *
200  * Output:           Changes the color of one pixel on the display.
201  *
202  * Error Handling:   None.
203  *
204  * Limitations:      None.
205  *
206  * Algorithms:       None.
207  * Data Structures:  None.
208  *
209  * Registers Changed: r8, r9, r10.
210  *
211  * Revision History:
212  *     6/03/14   Santiago Navonne   Initial revision.
213  *
214  */
215
216 .global plot_pixel
217 plot_pixel:                /* draw a pixel of the specified color */
218     MOVHI    r8, %hi(VRAM_BASE) /* find pixel location by first going to VRAM base */
219     ORI      r8, r8, %lo(VRAM_BASE)
220     MOVI     r9, ROW_ADDR_SHIFT /* shift the row to the row part of the address */
221     SLL      r9, r5, r9
222     MOVI     r10, COL_ADDR_SHIFT /* and the column to the column part */
223     SLL      r10, r4, r10
224     OR       r8, r8, r9         /* OR row, column, and VRAM base together */
225     OR       r8, r8, r10        /* to create final pixel address */
226     STWIO    r6, (r8)          /* and finally save passed color value to that address */

```

```

226 RET /* all done, so return */
227
228
229 /*
230 * pixel_color
231 *
232 * Description: This procedure returns the color of the pixel at the passed x, y
233 * coordinates, where the top left corner is (0, 0). Colors are
234 * specified with a 24-bit RGB value, where the bottom 8 bits
235 * represent the amount of blue, the following 8 the amount of green,
236 * and the next 8 the amount of red.
237 *
238 * Operation: The function simply translates the x and y coordinates into a VRAM
239 * address by setting the top bits to the offset of the VRAM, and ORing
240 * in the shifted row and column indeces. Then, it loads the color word
241 * from VRAM and returns it in r2.
242 *
243 * Arguments: x - x coordinate of the pixel, where leftmost column is 0 (r4).
244 * y - y coordinate of the pixel, where top row is 0 (r5).
245 *
246 * Return Value: color - 24-bit value with RGB color of requested pixel, or NO_TRACE
247 * if no trace was found at the requested coordinate(r2).
248 *
249 * Local Variables: None.
250 *
251 * Shared Variables: None.
252 *
253 * Global Variables: None.
254 *
255 * Input: None.
256 *
257 * Output: None.
258 *
259 * Error Handling: None.
260 *
261 * Limitations: None.
262 *
263 * Algorithms: None.
264 * Data Structures: None.
265 *
266 * Registers Changed: r8, r9, r10, r2.
267 *
268 * Revision History:
269 * 6/03/14 Santiago Navonne Initial revision.
270 *
271 */
272
273 .global pixel_color
274 pixel_color: /* read a pixel from display */
275 MOVHI r8, %hi(VRAM_BASE) /* find pixel location by first going to VRAM base */
276 ORI r8, r8, %lo(VRAM_BASE)
277 MOVI r9, ROW_ADDR_SHIFT /* shift the row to the row part of the address */
278 SLL r9, r5, r9
279 MOVI r10, COL_ADDR_SHIFT /* and the column to the column part */
280 SLL r10, r4, r10
281 OR r8, r8, r9 /* OR row, column, and VRAM base together */
282 OR r8, r8, r10 /* to create final pixel address */
283 LDWIO r2, (r8) /* and finally read color value from that address */
284
285 RET /* storing it in return register */

```