

```
1 ;
2 ; dog_lcd_test_avr128.asm
3 ;
4 ; Created: 10/9/2023 2:14:29 PM
5 ; Author : kshort
6 ;
7
8
9 ;*****
10 ;*****          BASIC DOG LCD TEST PROGRAM          *****
11 ;*****
12 ;
13 ;DOG_LCD_BasicTest.asm
14 ; Simple test application to verify DOG LCD is properly
15 ; wired. This test writes simple test messages to each
16 ; line of the display.
17 ;
18 ;Version - 2.0 For DOGM163W LCD operated at 3.3V
19 ;
20
21     .CSEG
22
23     ; interrupt vector table, with several 'safety' stubs
24     rjmp RESET      ;Reset/Cold start vector
25     reti            ;External Intr0 vector
26     reti            ;External Intr1 vector
27
28
29
30 ;*****
31 ;*****  M A I N   A P P L I C A T I O N   C O D E   *****
32 ;*****
33
34 RESET:
35
36     sbi VPORTA_DIR, 7      ; set PA7 = output.
37     sbi VPORTA_OUT, 7      ; set /SS of DOG LCD = 1 (Deselected)
38
39     rcall init_lcd_dog     ; init display, using SPI serial interface
40     rcall clr_dsp_buffs    ; clear all three SRAM memory buffer lines
41
42     rcall update_lcd_dog    ;display data in memory buffer on LCD
43
44     rcall start
45     rcall clear_display
46 // display setting line
47
48     rcall update_lcd_dog
49
```

```

50
51     // keypad subroutine
52     ldi r19, 3 // digit full
53     ldi r18, 0x00
54     check_press:
55         wait_for_1:
56         sbis VPORIB_IN, 5    ;wait for PB5 being 1
57         rjmp wait_for_1     ;skip this line if PE0 is 1
58
59     rcall output
60
61     rcall update_lcd_dog
62
63     rjmp check_press
64
65     end_loop:        ;infinite loop, program's task is complete
66     rjmp end_loop
67
68
69     ; press -> convert to ascii -> display (do not shift)-> press -> shift to ↗
70     ; the left
71     ; (but only have to shift the digits not the whole line)
72     ; -> every time we press, have to check if that press is enter or clear
73     ; -> check if 3 digits are full for that line, go in to a loop only looking ↗
74     ; for clear or enter
75     ; -> when you press enter, check if the value on the display is over 100 or ↗
76     ; not
77
78     ;----- SUBROUTINES -----
79
80     ;*****
81     ; keypad subroutine
82     ;*****
83     table: .db $31, $32, $33, $46, $34, $35, $36, $45, $37, $38, $39, $44, $41, ↗
84             $30, $42, $43
85
86     output:
87     in r18, VPORIB_IN    // gets the input from DIP switch and keypad
88
89     lsr r18    // shifting to right 4 bits
90     lsr r18
91     lsr r18
92     lsr r18
93
94     dec r19 // is digit full

```

```
95
96 // lookup table from lecture
97 lookup:
98     ldi r16, 0x00
99     ldi ZH, high (table*2)
100    ldi ZL, low (table*2)
101    ldi r16, $00
102    add ZL, r18
103    adc ZH, r16
104    lpm r18, Z
105
106    cpi r18, $41    // if the pressed key is clear
107    breq push_clear
108
109    cpi r18, $43    // if the pressed key is Enter
110    breq push_enter
111
112    rcall shift_by_1
113    st X, r18 // storing into SRAM buffer
114
115
116
117 delay_break:           ;delay lable for break delay
118     ldi r16, 80
119     outer_loop_break:
120         ldi r17, 133
121         inner_loop_break:
122             dec r17
123             brne inner_loop_break
124             dec r16
125         brne outer_loop_break
126
127 clear_flipflop:        // clear the flip flop for next input
128     cbi VPORTB_OUT, 4
129     sbi VPORTB_OUT, 4
130
131     rcall update_lcd_dog
132
133 cpi r19, 0x00 // if digit is full
134 breq is_digit_full
135
136 rjmp check_press // go back to the start
137
138
139 ;*****
140 ; push_clear
141 ;*****
142
143 push_clear:
```

```

144 ldi r19, 0x03
145
146 rcall clear_display
147 rjmp delay_break
148
149
150 ;*****
151 ; push_enter
152 ;*****
153
154 push_enter:
155 ldi r19, 0x03
156 inc r18
157     // check if the value is over 100
158     // if not
159     ldi XH, high (dsp_buff_1+r18*16)
160     ldi XL, low (dsp_buff_1+r18*16)
161
162 rjmp delay_break
163
164
165 ;*****
166 ; reset pointer
167 ;*****
168 reset_pointer:
169     ldi r20, 47
170
171     ldi XH, high (dsp_buff_1) ; Load ZH and ZL as a pointer to 1st
172     ldi XL, low (dsp_buff_1) ; byte of buffer for line 1.
173
174     rjmp output
175
176 ;*****
177 ; is digit full
178 ;*****
179
180 is_digit_full:
181     wait_for_clear_or_enter_loop:    // in a loop that only wait for clear or enter
182         sbis VPORTB_IN, 5
183         rjmp wait_for_clear_or_enter_loop
184
185         in r18, VPORTC_IN    // gets the input from DIP switch and keypad
186
187         lsr r18    // shifting to right 4 bits
188         lsr r18
189         lsr r18
190         lsr r18
191

```

```

192         ldi r16, 0x00
193         ldi ZH, high (table*2)
194         ldi ZL, low (table*2)
195         ldi r16, $00
196         add ZL, r18
197         adc ZH, r16
198         lpm r18, Z
199
200         cpi r18, $41    // if the pressed key is clear
201         breq push_clear
202
203         cpi r18, $43    // if the pressed key is Enter
204         breq push_enter
205
206         rjmp wait_for_clear_or_enter_loop
207
208
209 ;*****
210 ;NAME:      clr_dsp_buffs
211 ;FUNCTION:  Initializes dsp_buffers 1, 2, and 3 with blanks (0x20)
212 ;ASSUMES:   Three CONTIGUOUS 16-byte dram based buffers named
213 ;           dsp_buff_1, dsp_buff_2, dsp_buff_3.
214 ;RETURNS:   nothing.
215 ;MODIFIES:  r25,r26, Z-ptr
216 ;CALLS:     none
217 ;CALLED BY: main application and diagnostics
218 ;*****
219 clr_dsp_buffs:
220     ldi R25, 48          ; load total length of both buffer.
221     ldi R26, ' '         ; load blank/space into R26.
222     ldi ZH, high (dsp_buff_1) ; Load ZH and ZL as a pointer to 1st
223     ldi ZL, low (dsp_buff_1)  ; byte of buffer for line 1.
224
225     ;set DDRAM address to 1st position of first line.
226 store_bytes:
227     st  Z+, R26          ; store ' ' into 1st/next buffer byte and
228                         ; auto inc ptr to next location.
229     dec R25              ;
230     brne store_bytes    ; cont until r25=0, all bytes written.
231     ret
232
233
234
235 ;*****
236
237
238
239 line1_testmessage: .db 1, "Setting 1 :   ", 0 ; message for line #1.
240 line2_testmessage: .db 2, "Setting 2 :   ", 0 ; message for line #2.

```

```

241 line3_testmessage: .db 3, "Setting 3 : ", 0 ; message for line #3.
242
243
244
245
246
247 ;*****
248 ; start subroutine
249 ;*****
250 start:
251     sbi VPORTA_DIR, 4    //MOSI output
252
253     sbi VPORTB_DIR, 4    // clear flip flop output
254     cbi VPORTB_OUT, 4
255     sbi VPORTB_OUT, 4    // clear = 1
256
257     ; keypad input
258     cbi VPORTC_DIR, 7
259     cbi VPORTC_DIR, 6
260     cbi VPORTC_DIR, 5
261     cbi VPORTC_DIR, 4
262
263     cbi VPORTB_DIR, 5    // check if the keypad is pressed
264
265     ldi XH, high (dsp_buff_1+15) ; Load ZH and ZL as a pointer to 1st
266     ldi XL, low (dsp_buff_1+15)  ; byte of buffer for line 1.
267     ret
268
269 clear_display:
270     ;load_line_1 into dbuff1:
271     ldi ZH, high(line1_testmessage<<1) ; pointer to line 1 memory buffer
272     ldi ZL, low(line1_testmessage<<1)  ;
273     rcall load_msg                      ; load message into buffer(s).
274
275     ldi ZH, high(line2_testmessage<<1) ; pointer to line 2 memory buffer
276     ldi ZL, low(line2_testmessage<<1)  ;
277     rcall load_msg                      ; load message into buffer(s).
278
279     ldi ZH, high(line3_testmessage<<1) ; pointer to line 3 memory buffer
280     ldi ZL, low(line3_testmessage<<1)  ;
281     rcall load_msg                      ; load message into buffer(s).
282
283     ;breakpoint followin instr. to see blanked LCD and messages in buffer
284     rcall update_lcd_dog                ;breakpoint here to see blanked LCD
285
286     ret
287
288 ;*****
289 ; shift_by_1

```

```

290 ;*****
291
292 shift_by_1:
293     ldi ZH, high (dsp_buff_1+13)
294     ldi ZL, low (dsp_buff_1+13)
295     ldi r20, 0x20    //r16 is zero 0
296
297
298     ld r17, Z
299
300         sbiw ZH:ZL, $0001    ; decrement the pointer
301
302         st Z+, r17
303         //adiw XH:XL, $0001
304
305         st Z, r20
306
307
308 ret
309
310 ;*****
311 ;NAME:      load_msg
312 ;FUNCTION:  Loads a predefined string msg into a specified diplay
313 ;           buffer.
314 ;ASSUMES:   Z = offset of message to be loaded. Msg format is
315 ;           defined below.
316 ;RETURNS:   nothing.
317 ;MODIFIES:  r16, Y, Z
318 ;CALLS:     nothing
319 ;CALLED BY:
320 ;*****
321 ; Message structure:
322 ;   label:  .db <buff num>, <text string/message>, <end of string>
323 ;
324 ; Message examples (also see Messages at the end of this file/module):
325 ;   msg_1: .db 1,"First Message ", 0    ; loads msg into buff 1, eom=0
326 ;   msg_2: .db 1,"Another message ", 0 ; loads msg into buff 1, eom=0
327 ;
328 ; Notes:
329 ;   a) The 1st number indicates which buffer to load (either 1, 2, or 3).
330 ;   b) The last number (zero) is an 'end of string' indicator.
331 ;   c) Y = ptr to disp_buffer
332 ;       Z = ptr to message (passed to subroutine)
333 ;*****
334 load_msg:
335     ldi YH, high (dsp_buff_1) ; Load YH and YL as a pointer to 1st
336     ldi YL, low (dsp_buff_1)  ; byte of dsp_buff_1 (Note - assuming
337                               ; (dsp_buff_1 for now).
338     lpm R16, Z+               ; get dsply buff number (1st byte of msg).

```

```
339     cpi r16, 1           ; if equal to '1', ptr already setup.
340     breq get_msg_byte    ; jump and start message load.
341     adiw YH:YL, 16       ; else set ptr to dsp buff 2.
342     cpi r16, 2           ; if equal to '2', ptr now setup.
343     breq get_msg_byte    ; jump and start message load.
344     adiw YH:YL, 16       ; else set ptr to dsp buff 2.
345
346 get_msg_byte:
347     lpm R16, Z+          ; get next byte of msg and see if '0'.
348     cpi R16, 0           ; if equal to '0', end of message reached.
349     breq msg_loaded      ; jump and stop message loading operation.
350     st Y+, R16           ; else, store next byte of msg in buffer.
351     rjmp get_msg_byte    ; jump back and continue...
352 msg_loaded:
353     ret
354
355 ;***** END OF FILE *****
356
357
358 ;=====
359 .include "lcd_dog_asm_driver_avr128.inc" ; LCD DOG init/update procedures.
360 ;=====
361
362
```