/Volumes/DongyunLee/ESE280 Lab/Lab8/task4.asm

```
;
; dog_lcd_test_avr128.asm
;
; Created: 10/9/2023 2:14:29 PM
; Author : kshort
;


;*********************************************************************
;**********        BASIC DOG LCD TEST PROGRAM           **********
;*********************************************************************
;
;DOG_LCD_BasicTest.asm
;  Simple test application to verify DOG LCD is properly
;  wired.  This test writes simple test messages to each
;  line of the display.
;
;Version - 2.0 For DOGM163W LCD operated at 3.3V
;

     .CSEG

     ; interrupt vector table, with several 'safety' stubs
     rjmp RESET      ;Reset/Cold start vector
     reti            ;External Intr0 vector
     reti            ;External Intr1 vector



;*************************************************************************
;************* M A I N   A P P L I C A T I O N   C O D E  *************
;*************************************************************************

RESET:

    sbi VPORTA_DIR, 7       ; set PA7 = output.
    sbi VPORTA_OUT, 7       ; set /SS of DOG LCD = 1 (Deselected)

    rcall init_lcd_dog    ; init display, using SPI serial interface
    rcall clr_dsp_buffs   ; clear all three SRAM memory buffer lines

    rcall update_lcd_dog        ;display data in memory buffer on LCD

    rcall start

// display setting line

    rcall clear_line_1
    rcall clear_line_2
    rcall clear_line_3

    rcall update_lcd_dog

    // keypad subroutine
    check_press:
        wait_for_1:
        sbis VPORTB_IN, 5   ;wait for PB5 being 1
```

```
58          rjmp wait_for_1     ;skip this line if PE0 is 1
59
60
61      rcall is_digit_full
62      rcall output
63
64      rcall update_lcd_dog
65
66      rjmp check_press
67
68      end_loop:       ;infinite loop, program's task is complete
69      rjmp end_loop
70
71
72      ; press -> convert to ascii -> display (do not shift)-> press -> shift to the
    left
73      ; (but only have to shift the digits not the whole line)
74      ; -> every time we press, have to check if that press is enter or clear
75      ; -> check if 3 digits are full for that line, go in to a loop only looking
    for clear or enter
76      ; -> when you press enter, check if the value on the display is over 100 or
    not
77
78
79
80
81  ;---------------------------- SUBROUTINES ----------------------------
82
83
84  ;====================================
85  .include "lcd_dog_asm_driver_avr128.inc"  ; LCD DOG init/update procedures.
86  ;====================================
87
88
89  ;************************
90  ;NAME:      clr_dsp_buffs
91  ;FUNCTION:  Initializes dsp_buffers 1, 2, and 3 with blanks (0x20)
92  ;ASSUMES:   Three CONTIGUOUS 16-byte dram based buffers named
93  ;           dsp_buff_1, dsp_buff_2, dsp_buff_3.
94  ;RETURNS:   nothing.
95  ;MODIFIES:  r25,r26, Z-ptr
96  ;CALLS:     none
97  ;CALLED BY: main application and diagnostics
98  ;*********************************************************************
99  clr_dsp_buffs:
100      ldi R25, 48               ; load total length of both buffer.
101      ldi R26, ' '              ; load blank/space into R26.
102      ldi ZH, high (dsp_buff_1) ; Load ZH and ZL as a pointer to 1st
103      ldi ZL, low (dsp_buff_1)  ; byte of buffer for line 1.
104
105     ;set DDRAM address to 1st position of first line.
106 store_bytes:
107      st  Z+, R26       ; store ' ' into 1st/next buffer byte and
108                        ; auto inc ptr to next location.
109      dec  R25          ;
110      brne store_bytes  ; cont until r25=0, all bytes written.
111      ret
112
113
114
115 ;*********************************************************************
```

```
116
117
118
119 line1_testmessage: .db 1, "Setting 1  :    ", 0  ; message for line #1.
120 line2_testmessage: .db 2, "Setting 2  :    ", 0  ; message for line #2.
121 line3_testmessage: .db 3, "Setting 3  :    ", 0  ; message for line #3.
122
123
124
125
126
127 ;*********************************************************************
128 ; start subroutine
129 ;*********************************************************************
130 start:
131     sbi VPORTA_DIR, 4    //MOSI output
132
133     sbi VPORTB_DIR, 4    // clear flip flop output
134
135     ; keypad input
136     cbi VPORTC_DIR, 7
137     cbi VPORTC_DIR, 6
138     cbi VPORTC_DIR, 5
139     cbi VPORTC_DIR, 4
140
141     cbi VPORTB_DIR, 5    // check if the keypad is pressed
142
143     ldi XH, high (dsp_buff_1+14) ; Load ZH and ZL as a pointer to 1st
144     ldi XL, low (dsp_buff_1+14)  ; byte of buffer for line 1.
145
146     ldi r20, 48     // check if all character is full
147
148
149 ;*********************************************************************
150 ; keypad subroutine
151 ;*********************************************************************
152 table: .db $31, $32, $33, $46, $34, $35, $36, $45, $37, $38, $39, $44, $41, $30,
    $42, $43
153
154
155 output:
156 in r18, VPORTC_IN   // gets the input from DIP switch and keypad
157
158 lsr r18     // shifting  to right 4 bits
159 lsr r18
160 lsr r18
161 lsr r18
162
163 mov r19, r18            // copy it to another register
164
165 // lookup table from lecture
166 lookup:
167     ldi r16, 0x00
168     ldi ZH, high (table*2)
169     ldi ZL, low (table*2)
170     ldi r16, $00
171     add ZL, r18
172     adc ZH, r16
173     lpm r18, Z
174
```

```
175     cpi r18, $41    // if the pressed key is clear
176         breq push_clear
177 
178     cpi r18, $43    // if the pressed key is Enter
179         breq push_enter
180 
181     rcall shift_by_1
182     st X+, r18  // storing into SRAM buffer
183 
184 
185 
186 delay_break:            ;delay lable for break delay
187     ldi r16, 80
188     outer_loop_break:
189         ldi r17, 133
190         inner_loop_break:
191             dec r17
192     brne inner_loop_break
193         dec r16
194 brne outer_loop_break
195     cbi VPORTB_OUT, 4
196     sbi VPORTB_OUT, 4
197 
198 clear_flipflop:     // clear the flip flop for next input
199     cbi VPORTB_OUT, 4
200     sbi VPORTB_OUT, 4
201 
202 
203 rjmp check_press    // go back to the start
204 
205 
206 ;***************************************************************************
207 ; is digit full
208 ;***************************************************************************
209 
210 is_digit_full:
211     ldi ZL, low(dsp_buff_1+16)
212     ld r21, ZL
213 
214     cpi r21, 0x20
215     brne check_press
216 
217     wait_for_clear_or_enter_loop:   // in a loop that only wait for clear or enter
218         sbis VPORTB_IN, 5
219         rjmp wait_for_clear_or_enter_loop
220 
221               in r18, VPORTC_IN // gets the input from DIP switch and keypad
222 
223             lsr r18     // shifting  to right 4 bits
224             lsr r18
225             lsr r18
226             lsr r18
227 
228             mov r19, r18            // copy it to another register
229 
230             ldi r16, 0x00
231             ldi ZH, high (table*2)
232             ldi ZL, low (table*2)
233             ldi r16, $00
234             add ZL, r18
```

```
235             adc ZH, r16
236             lpm r18, Z
237
238             cpi r18, $41    // if the pressed key is clear
239             breq push_clear
240
241             cpi r18, $43    // if the pressed key is Enter
242             breq push_enter
243
244             rcall delay_break
245
246             rjmp wait_for_clear_or_enter_loop
247
248
249 ;*********************************************************************
250 ; shift_by_1
251 ;*********************************************************************
252
253 shift_by_1:
254     ldi ZH, high (dsp_buff_1+16) ; Load ZH and ZL as a pointer to 1st
255     ldi XL, low (dsp_buff_1+16)  ; byte of buffer for line 1.
256     ldi r20, 0x20   //r16 is zero 0
257
258     loop_outside:
259
260         loop_shifting:
261             ld r16, Z
262             ; adiw XH:XL, $0001     // increament the pointer but it is done br
    the next line
263             sdiw ZH:ZL, $0001   ; decrement the pointer
264             ld r17, Z
265
266             sdiw ZH:ZL, $0001   ; decrement the pointer
267
268             st Z, r17
269
270             sdiw ZH:ZL, $0001   ; decrement the pointer
271
272             st Z, r16
273             brne push_input
274             // rjmp loop_shifting
275
276
277
278
279
280 ;*********************************************************************
281 ; push_clear
282 ;*********************************************************************
283
284 push_clear:
285     // read which line is the pointer at
286     // depending on the line
287
288     rcall delay_break
289     rjmp clear_line_1
290
291 ;*********************************************************************
292 ; push_enter
293 ;*********************************************************************
```

```
294
295 push_enter:
296     // check if the value is over 100
297     // if not
298     ldi XH, high (dsp_buff_1+30)
299     ldi XL, low (dsp_buff_1+30)
300
301     rcall delay_break
302     rjmp check_press
303
304 ;***************************************************************************
305 ; reset pointer
306 ;***************************************************************************
307 reset_pointer:
308     ldi r20, 47
309
310     ldi XH, high (dsp_buff_1) ; Load ZH and ZL as a pointer to 1st
311     ldi XL, low (dsp_buff_1)  ; byte of buffer for line 1.
312
313     rjmp output
314
315
316 ;***************************************************************************
317 ;   clear line 1
318 ;***************************************************************************
319
320 clear_line_1:
321       ;load_line_1 into dbuff1:
322    ldi  ZH, high(line1_testmessage<<1)  ; pointer to line 1 memory buffer
323    ldi  ZL, low(line1_testmessage<<1)   ;
324    rcall load_msg          ; load message into buffer(s).
325
326    rjmp check_press
327
328 ;***************************************************************************
329 ;   clear line 2
330 ;***************************************************************************
331
332 clear_line_2:
333    ldi  ZH, high(line2_testmessage<<1)  ; pointer to line 2 memory buffer
334    ldi  ZL, low(line2_testmessage<<1)   ;
335    rcall load_msg          ; load message into buffer(s).
336
337    rjmp check_press
338
339 ;***************************************************************************
340 ;   clear line 3
341 ;***************************************************************************
342
343 clear_line_3:
344
345    ldi  ZH, high(line3_testmessage<<1)  ; pointer to line 3 memory buffer
346    ldi  ZL, low(line3_testmessage<<1)   ;
347    rcall load_msg          ; load message into buffer(s).
348
349    rjmp check_press
350
351 ;********************
352 ;NAME:      load_msg
353 ;FUNCTION:  Loads a predefined string msg into a specified diplay
```

```
;           buffer.
;ASSUMES:   Z = offset of message to be loaded. Msg format is
;           defined below.
;RETURNS:   nothing.
;MODIFIES:  r16, Y, Z
;CALLS:     nothing
;CALLED BY:
;*********************************************************************
; Message structure:
;   label:  .db <buff num>, <text string/message>, <end of string>
;
; Message examples (also see Messages at the end of this file/module):
;   msg_1: .db 1,"First Message ", 0   ; loads msg into buff 1, eom=0
;   msg_2: .db 1,"Another message ", 0 ; loads msg into buff 1, eom=0
;
; Notes:
;   a) The 1st number indicates which buffer to load (either 1, 2, or 3).
;   b) The last number (zero) is an 'end of string' indicator.
;   c) Y = ptr to disp_buffer
;      Z = ptr to message (passed to subroutine)
;*********************************************************************
load_msg:
     ldi YH, high (dsp_buff_1) ; Load YH and YL as a pointer to 1st
     ldi YL, low (dsp_buff_1)  ; byte of dsp_buff_1 (Note - assuming
                               ; (dsp_buff_1 for now).
     lpm R16, Z+               ; get dsply buff number (1st byte of msg).
     cpi r16, 1                ; if equal to '1', ptr already setup.
     breq get_msg_byte         ; jump and start message load.
     adiw YH:YL, 16            ; else set ptr to dsp buff 2.
     cpi r16, 2                ; if equal to '2', ptr now setup.
     breq get_msg_byte         ; jump and start message load.
     adiw YH:YL, 16            ; else set ptr to dsp buff 2.

get_msg_byte:
     lpm R16, Z+               ; get next byte of msg and see if '0'.
     cpi R16, 0                ; if equal to '0', end of message reached.
     breq msg_loaded           ; jump and stop message loading operation.
     st Y+, R16                ; else, store next byte of msg in buffer.
     rjmp get_msg_byte         ; jump back and continue...
msg_loaded:
     ret

;***** END OF FILE ******
```