/Volumes/DongyunLee/ESE280 Lab/Lab8/task3.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 start

    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

    // keypad subroutine
    check_press:
        wait_for_1:
        sbis VPORTB_IN, 5   ;wait for PB5 being 1
        rjmp wait_for_1     ;skip this line if PE0 is 1
        dec r20     // chekcing if all character is full
        breq reset_pointer

    rcall output


    end_loop:       ;infinite loop, program's task is complete
```

```
    rjmp end_loop




;---------------------------- SUBROUTINES ----------------------------


;====================================
.include "lcd_dog_asm_driver_avr128.inc"  ; LCD DOG init/update procedures.
;====================================


;************************
;NAME:      clr_dsp_buffs
;FUNCTION:  Initializes dsp_buffers 1, 2, and 3 with blanks (0x20)
;ASSUMES:   Three CONTIGUOUS 16-byte dram based buffers named
;           dsp_buff_1, dsp_buff_2, dsp_buff_3.
;RETURNS:   nothing.
;MODIFIES:  r25,r26, Z-ptr
;CALLS:     none
;CALLED BY: main application and diagnostics
;*********************************************************************
clr_dsp_buffs:
     ldi R25, 48              ; load total length of both buffer.
     ldi R26, ' '             ; load blank/space into R26.
     ldi ZH, high (dsp_buff_1) ; Load ZH and ZL as a pointer to 1st
     ldi ZL, low (dsp_buff_1)  ; byte of buffer for line 1.

    ;set DDRAM address to 1st position of first line.
store_bytes:
     st  Z+, R26       ; store ' ' into 1st/next buffer byte and
                       ; auto inc ptr to next location.
     dec  R25          ;
     brne store_bytes  ; cont until r25=0, all bytes written.
     ret

;*********************************************************************
; start subroutine

start:
    sbi VPORTA_DIR, 4    //MOSI output

    sbi VPORTB_DIR, 4    // clear flip flop

    ; keypad input
    cbi VPORTC_DIR, 7
    cbi VPORTC_DIR, 6
    cbi VPORTC_DIR, 5
    cbi VPORTC_DIR, 4

    cbi VPORTB_DIR, 5    // check if the keypad is pressed

    ldi XH, high (dsp_buff_1) ; Load ZH and ZL as a pointer to 1st
    ldi XL, low (dsp_buff_1)  ; byte of buffer for line 1.

    ldi r20, 48     // check if all character is full


```

```
118
119
120
121
122 ;***********************************************************************
123 ; keypad subroutine
124
125 table: .db $31, $32, $33, $46, $34, $35, $36, $45, $37, $38, $39, $44, $41, $30,
    $42, $43
126
127
128 output:
129 in r18, VPORTC_IN   // gets the input from DIP switch and keypad
130
131 lsr r18     // shifting  to right 4 bits
132 lsr r18
133 lsr r18
134 lsr r18
135
136 mov r19, r18            // copy it to another register
137
138 // lookup table from lecture
139 lookup:
140     ldi r16, 0x00
141     ldi ZH, high (table*2)
142     ldi ZL, low (table*2)
143     ldi r16, $00
144     add ZL, r18
145     adc ZH, r16
146     lpm r18, Z
147
148     st X+, r18  // storing into SRAM buffer
149
150
151
152 delay_break:            ;delay lable for break delay
153     ldi r16, 80
154     outer_loop_break:
155         ldi r17, 133
156         inner_loop_break:
157             dec r17
158     brne inner_loop_break
159         dec r16
160 brne outer_loop_break
161
162 clear_flipflop:     // clear the flip flop for next input
163     cbi VPORTB_OUT, 4
164     sbi VPORTB_OUT, 4
165
166     rcall update_lcd_dog    // display
167
168 rjmp check_press    // go back to the start
169
170 reset_pointer:
171     ldi r20, 47
172
173     ldi XH, high (dsp_buff_1) ; Load ZH and ZL as a pointer to 1st
174     ldi XL, low (dsp_buff_1)  ; byte of buffer for line 1.
175
176     rjmp output
```

```
177
178
179
180 2s_delay:
181     ldi r22, 160  ; Set R22 to introduce a delay of ~160 * 30uS = 4.8ms
182     ldi r23, 125  ; Set R23 to repeat the above delay 250 times for ~2 seconds
183
184      2s_delay_loop:
185           rcall v_delay  ; Call the v_delay subroutine with the specified delay
186           dec r23        ; Decrement the outer loop counter
187           brne 2s_delay_loop  ; Continue the loop until r23 reaches zero
188           ret
189
190 ;********************
191 ;NAME:      load_msg
192 ;FUNCTION:  Loads a predefined string msg into a specified diplay
193 ;           buffer.
194 ;ASSUMES:   Z = offset of message to be loaded. Msg format is
195 ;           defined below.
196 ;RETURNS:   nothing.
197 ;MODIFIES:  r16, Y, Z
198 ;CALLS:     nothing
199 ;CALLED BY:
200 ;*************************************************************************
201 ; Message structure:
202 ;   label:  .db <buff num>, <text string/message>, <end of string>
203 ;
204 ; Message examples (also see Messages at the end of this file/module):
205 ;   msg_1: .db 1,"First Message ", 0   ; loads msg into buff 1, eom=0
206 ;   msg_2: .db 1,"Another message ", 0 ; loads msg into buff 1, eom=0
207 ;
208 ; Notes:
209 ;   a) The 1st number indicates which buffer to load (either 1, 2, or 3).
210 ;   b) The last number (zero) is an 'end of string' indicator.
211 ;   c) Y = ptr to disp_buffer
212 ;      Z = ptr to message (passed to subroutine)
213 ;*************************************************************************
214 load_msg:
215      ldi YH, high (dsp_buff_1) ; Load YH and YL as a pointer to 1st
216      ldi YL, low (dsp_buff_1)  ; byte of dsp_buff_1 (Note - assuming
217                                ; (dsp_buff_1 for now).
218      lpm R16, Z+               ; get dsply buff number (1st byte of msg).
219      cpi r16, 1                ; if equal to '1', ptr already setup.
220      breq get_msg_byte         ; jump and start message load.
221      adiw YH:YL, 16            ; else set ptr to dsp buff 2.
222      cpi r16, 2                ; if equal to '2', ptr now setup.
223      breq get_msg_byte         ; jump and start message load.
224      adiw YH:YL, 16            ; else set ptr to dsp buff 2.
225
226 get_msg_byte:
227      lpm R16, Z+               ; get next byte of msg and see if '0'.
228      cpi R16, 0                ; if equal to '0', end of message reached.
229      breq msg_loaded           ; jump and stop message loading operation.
230      st Y+, R16                ; else, store next byte of msg in buffer.
231      rjmp get_msg_byte         ; jump back and continue...
232 msg_loaded:
233      ret
234
235 ;***** END OF FILE ******
236
```