```
;***************************************************************
;**********        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  ************
;*********************************************************************

.org PORTE_PORT_vect
    jmp porte_isr       ;vector for all PORTE pin change IRQs

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

    rcall update_lcd_dog


    cbi VPORTE_DIR, 0   ;PE0 input- gets output from pushbutton debouce ckt.


    ;Configure interrupt
    lds r16, PORTE_PIN0CTRL ;set ISC for PE0 to pos. edge
```

```
50      ori r16, 0x02           // positive edge detect
51      sts PORTE_PIN0CTRL, r16
52
53      sei                 ;enable global interrupts
54
55      main_loop:      ;infinite loop, program's task is complete
56          //cbi VPORTD_OUT, 0
57      rjmp main_loop
58
59  ;*********************************************************************
60  ; start subroutine
61  ;*********************************************************************
62  start:
63      sbi VPORTA_DIR, 4    //MOSI output
64
65     // sbi VPORTB_DIR, 4    // clear flip flop output
66      //sbi VPORTB_OUT, 4 // set clear to 1
67
68      ldi r17, 0x00
69      out VPORTC_DIR, r17 // input 4 dip switch + 16 keypads
70      sbi VPORTD_DIR, 0   // pulse generator
71
72      //cbi VPORTB_DIR, 5    // check if the keypad is pressed
73
74      ldi XH, high (dsp_buff_1+15) ; Load ZH and ZL as a pointer to 1st
75      ldi XL, low (dsp_buff_1+15)  ; byte of buffer for line 1.
76
77      ret
78
79  ;*********************************************************************
80  ; interrupt service routine
81  ;*********************************************************************
82  ;Interrupt service routine for any PORTE pin change IRQ
83  porte_ISR:
84      cli             ;clear global interrupt enable, I = 0
85
86      push r16        ;save r16 then SREG, note I = 0
87      in r16, CPU_SREG
88      push r16
89
90      ;Determine which pins of PORTE have IRQs
91      lds r18, PORTE_INTFLAGS ;check for PE0 IRQ flag set
92      sbrc r18, 0
93      rcall output            ;execute subroutine for PE0
94
95      pop r16         ;restore SREG then r16
96      out CPU_SREG, r16   ;note I in SREG now = 0
97      pop r16         ;restore SREG then r16
98      sei             ;SREG I = 1
```

```
 99      reti            ;return from PORTE pin change ISR
100  ;Note: reti does not set I on an AVR128DB48
101
102  ;*****************************************************************
103  ; keypad subroutine
104  ;*****************************************************************
105  table: .db $31, $32, $33, $46
106         .db $34, $35, $36, $45
107         .db $37, $38, $39, $44
108         .db $41, $30, $42, $43
109
110
111  output:
112
113  in r18, VPORTC_IN   // gets the input from DIP switch and keypad
114
115  lsr r18     // shifting  to right 4 bits
116  lsr r18
117  lsr r18
118  lsr r18
119
120
121  // lookup table from lecture
122  lookup:
123      ldi ZH, high (table*2)
124      ldi ZL, low (table*2)
125      ldi r16, $00
126      add ZL, r18
127      adc ZH, r16
128      lpm r18, Z
129
130      st X, r18  // storing into SRAM buffer
131
132      /*
133      clear_flipflop:     // clear the flip flop for next input
134      cbi VPORTB_OUT, 4
135      sbi VPORTB_OUT, 4
136      */
137      ldi r16, PORT_INT0_bm   ;clear IRQ flag for PE0
138      sts PORTE_INTFLAGS, r16
139
140      cpi r18, $41    // if the pressed key is clear
141          breq push_clear
142
143      cpi r18, $43    // if the pressed key is Enter
144          breq push_enter
145
146      rcall shift_by_1
147
```

```
148     rcall delay_break
149
150     rcall update_lcd_dog
151
152
153 ret
154 //rjmp main_loop    // go back to the start
155
156
157
158 ;*********************************************************************
159 ; delay break
160 ;*********************************************************************
161 delay_break:            ;delay lable for break delay
162     ldi r16, 80
163     outer_loop_break:
164         ldi r17, 133
165         inner_loop_break:
166             dec r17
167     brne inner_loop_break
168         dec r16
169 brne outer_loop_break
170
171 ret
172 ;*********************************************************************
173 ; push_clear
174 ;*********************************************************************
175
176 push_clear:
177     ldi r16, PORT_INT0_bm   ;clear IRQ flag for PE0
178     sts PORTE_INTFLAGS, r16
179     ret
180
181 ;*********************************************************************
182 ; error loop
183 ;*********************************************************************
184 line2_testmessage: .db 1, "ERROR, press CLEAR", 0  ; message for line #1.
185
186 error_loop:
187    ldi  ZH, high(line2_testmessage<<1)  ; pointer to line 1 memory buffer
188    ldi  ZL, low(line2_testmessage<<1)   ;
189    rcall load_msg          ; load message into buffer(s).
190    rcall update_lcd_dog
191
192    ldi r16, PORT_INT0_bm    ;clear IRQ flag for PE0
193     sts PORTE_INTFLAGS, r16
194    ret
195
196 ;*********************************************************************
```

```
197 ; push enter
198 ;*********************************************************************
199 addition_100th:
200     dec r17
201     ldi r16, 100
202     mul r18, r16 // multiply by 100 for the 100th place value
203     add r19, r0 // and then add the next digit on 1st
204     adiw ZH:ZL, $0001
205 rjmp lookup2
206
207 addition_10th:
208     dec r17
209     ldi r16, 10 // to multiply ; shift to the left on 10th
210     mul r18, r16    //shift to the left on 10th
211     add r19, r0
212     adiw ZH:ZL, $0001
213 rjmp lookup2
214
215
216 push_enter:
217
218     ldi r17, 3
219     ldi r18, 0x00
220     ldi r19, 0x00
221     ldi ZH, high (dsp_buff_1+12) ; Load ZH and ZL as a pointer to 1st
222     ldi ZL, low (dsp_buff_1+12)  ; byte of buffer for line 1.
223
224     ld r18, Z
225     andi r18, 0x0F  // mask to translate from ascii code to numerical value
226
227 lookup2:
228
229     cpi r17, 3
230     breq addition_100th
231
232     cpi r17, 2
233     breq addition_10th
234
235     // 1th addition
236     add r19, r18
237
238
239
240     cpi r19, 101    // check if the value is over 100
241     brge error_loop // branch if it is equal or greater than 101
242
243     // now convert the percentage value into value out of 255, and generate ↵
        pulse
244
```

```
245     cpi r19, 100
246     breq brightness_full
247
248     //cpi r19, 0
249     //breq brightness_zero
250
251     /*
252     mov r20, r19
253     lsr r20
254
255     lsl r19
256
257     add r19, r20
258     */
259     ldi r20, 255
260     sub r20, r19
261
262 ;*********************************************************************
263 ; execute
264 ;*********************************************************************
265 execute:
266 timing_loop:
267 mov r16, r19    // move it to r16 r19 dont change
268 mov r18, r20    // r20 dont change
269
270     loop:
271         sbi VPORTD_OUT, 0
272
273     dec_loop:
274         dec r16
275         brne loop
276
277     loop2:
278         cbi VPORTD_OUT, 0
279
280     dec_loop2:
281         dec r18
282         brne loop2
283
284     rjmp timing_loop
285
286 ;*********************************************************************
287 ; shift_by_1
288 ;*********************************************************************
289
290 shift_by_1:
291     ldi ZH, high (dsp_buff_1+15) ; Load ZH and ZL as a pointer to 1st
292     ldi ZL, low (dsp_buff_1+15)  ; byte of buffer for line 1.
293     ldi r20, 0x20   //r20 is blank
```

```
294
295     sbiw ZH:ZL, $0002
296     ld r19, Z
297
298     sbiw ZH:ZL, $0001
299     st Z, r19
300
301     adiw ZH:ZL, $0002
302     ld r19, Z
303
304     sbiw ZH:ZL, $0001
305     st Z, r19
306
307     adiw ZH:ZL, $0002
308     ld r19, Z
309
310     sbiw ZH:ZL, $0001
311     st Z, r18
312
313     adiw ZH:ZL, $0001
314     st Z, r20
315
316     ret
317 ;*********************************************************************
318 ; brightness full (100%)
319 ;*********************************************************************
320 brightness_full:
321     sbi VPORTD_OUT, 0
322     rjmp brightness_full
323
324 ;*********************************************************************
325 ; brightness zero (0%)
326 ;*********************************************************************
327 brightness_zero:
328     loop43:
329         sbi VPORTD_OUT, 0
330     rjmp loop43
331     cbi VPORTD_OUT, 0
332     rjmp brightness_zero
333
334
335 ;*********************************************************************
336 ;   clear line 1
337 ;*********************************************************************
338
339 line1_testmessage: .db 1, "Setting 1 : 000 ", 0  ; message for line #1.
340
341 clear_line:
342       ;load_line_1 into dbuff1:
```

```
343     ldi  ZH, high(line1_testmessage<<1)  ; pointer to line 1 memory buffer
344     ldi  ZL, low(line1_testmessage<<1)   ;
345     rcall load_msg          ; load message into buffer(s).
346
347     ret
348
349 ;*******************
350 ;NAME:      load_msg
351 ;FUNCTION:  Loads a predefined string msg into a specified diplay
352 ;           buffer.
353 ;ASSUMES:   Z = offset of message to be loaded. Msg format is
354 ;           defined below.
355 ;RETURNS:   nothing.
356 ;MODIFIES:  r16, Y, Z
357 ;CALLS:     nothing
358 ;CALLED BY:
359 ;*****************************************************************
360 ; Message structure:
361 ;   label:  .db <buff num>, <text string/message>, <end of string>
362 ;
363 ; Message examples (also see Messages at the end of this file/module):
364 ;   msg_1: .db 1,"First Message ", 0   ; loads msg into buff 1, eom=0
365 ;   msg_2: .db 1,"Another message ", 0 ; loads msg into buff 1, eom=0
366 ;
367 ; Notes:
368 ;   a) The 1st number indicates which buffer to load (either 1, 2, or 3).
369 ;   b) The last number (zero) is an 'end of string' indicator.
370 ;   c) Y = ptr to disp_buffer
371 ;      Z = ptr to message (passed to subroutine)
372 ;*****************************************************************
373 load_msg:
374      ldi YH, high (dsp_buff_1) ; Load YH and YL as a pointer to 1st
375      ldi YL, low (dsp_buff_1)  ; byte of dsp_buff_1 (Note - assuming
376                               ; (dsp_buff_1 for now).
377      lpm R16, Z+               ; get dsply buff number (1st byte of msg).
378      cpi r16, 1                ; if equal to '1', ptr already setup.
379      breq get_msg_byte         ; jump and start message load.
380      adiw YH:YL, 16            ; else set ptr to dsp buff 2.
381      cpi r16, 2                ; if equal to '2', ptr now setup.
382      breq get_msg_byte         ; jump and start message load.
383      adiw YH:YL, 16            ; else set ptr to dsp buff 2.
384
385 get_msg_byte:
386      lpm R16, Z+               ; get next byte of msg and see if '0'.
387      cpi R16, 0                ; if equal to '0', end of message reached.
388      breq msg_loaded           ; jump and stop message loading operation.
389      st Y+, R16                ; else, store next byte of msg in buffer.
390      rjmp get_msg_byte         ; jump back and continue...
391 msg_loaded:
```

```
392       ret
393
394   ;--------------------------- SUBROUTINES ---------------------------
395
396
397   ;===================================
398   .include "lcd_dog_asm_driver_avr128.inc"   ; LCD DOG init/update procedures.
399   ;===================================
400
401
402   ;************************
403   ;NAME:      clr_dsp_buffs
404   ;FUNCTION:  Initializes dsp_buffers 1, 2, and 3 with blanks (0x20)
405   ;ASSUMES:   Three CONTIGUOUS 16-byte dram based buffers named
406   ;           dsp_buff_1, dsp_buff_2, dsp_buff_3.
407   ;RETURNS:   nothing.
408   ;MODIFIES:  r25,r26, Z-ptr
409   ;CALLS:     none
410   ;CALLED BY: main application and diagnostics
411   ;*********************************************************************
412   clr_dsp_buffs:
413       ldi R25, 48               ; load total length of both buffer.
414       ldi R26, ' '              ; load blank/space into R26.
415       ldi ZH, high (dsp_buff_1) ; Load ZH and ZL as a pointer to 1st
416       ldi ZL, low (dsp_buff_1)  ; byte of buffer for line 1.
417
418      ;set DDRAM address to 1st position of first line.
419   store_bytes:
420       st  Z+, R26       ; store ' ' into 1st/next buffer byte and
421                         ; auto inc ptr to next location.
422       dec  R25          ;
423       brne store_bytes  ; cont until r25=0, all bytes written.
424       ret
425
426
427
428   ;*********************************************************************
429
430
431   ;***** END OF FILE ******
432
```