```
;*****************************************************************
;**********        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
    ori r16, 0x02           // positive edge detect
    sts PORTE_PIN0CTRL, r16

    sei                 ;enable global interrupts

    main_loop:      ;infinite loop, program's task is complete
        //cbi VPORTD_OUT, 0
    rjmp main_loop

;*********************************************************************
; start subroutine
;*********************************************************************
start:
    sbi VPORTA_DIR, 4    //MOSI output

   // sbi VPORTB_DIR, 4    // clear flip flop output
    //sbi VPORTB_OUT, 4 // set clear to 1

    ldi r17, 0x00
    out VPORTC_DIR, r17 // input 4 dip switch + 16 keypads
    sbi VPORTD_DIR, 0   // pulse generator

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

    ldi XH, high (dsp_buff_1+15) ; Load ZH and ZL as a pointer to 1st
    ldi XL, low (dsp_buff_1+15)  ; byte of buffer for line 1.

    ret

;*********************************************************************
; interrupt service routine
;*********************************************************************
;Interrupt service routine for any PORTE pin change IRQ
porte_ISR:
    cli             ;clear global interrupt enable, I = 0

    push r16        ;save r16 then SREG, note I = 0
    in r16, CPU_SREG
    push r16

    ;Determine which pins of PORTE have IRQs
    lds r18, PORTE_INTFLAGS ;check for PE0 IRQ flag set
    sbrc r18, 0
    rcall output            ;execute subroutine for PE0

    pop r16         ;restore SREG then r16
    out CPU_SREG, r16   ;note I in SREG now = 0
    pop r16         ;restore SREG then r16
    sei             ;SREG I = 1
    reti            ;return from PORTE pin change ISR
;Note: reti does not set I on an AVR128DB48

;********************************************************************
; keypad subroutine
;********************************************************************
table: .db $31, $32, $33, $46
       .db $34, $35, $36, $45
       .db $37, $38, $39, $44
       .db $41, $30, $42, $43


output:

in r18, VPORTC_IN   // gets the input from DIP switch and keypad

lsr r18     // shifting  to right 4 bits
lsr r18
lsr r18
lsr r18


// lookup table from lecture
lookup:
    ldi ZH, high (table*2)
    ldi ZL, low (table*2)
    ldi r16, $00
    add ZL, r18
    adc ZH, r16
    lpm r18, Z

    st X, r18  // storing into SRAM buffer

    /*
    clear_flipflop:     // clear the flip flop for next input
    cbi VPORTB_OUT, 4
    sbi VPORTB_OUT, 4
    */
    ldi r16, PORT_INT0_bm   ;clear IRQ flag for PE0
    sts PORTE_INTFLAGS, r16

    cpi r18, $41    // if the pressed key is clear
        breq push_clear

    cpi r18, $43    // if the pressed key is Enter
        breq push_enter

    rcall shift_by_1

    rcall delay_break

    rcall update_lcd_dog


ret
//rjmp main_loop    // go back to the start



;********************************************************************
; delay break
;********************************************************************
delay_break:            ;delay lable for break delay
    ldi r16, 80
    outer_loop_break:
        ldi r17, 133
        inner_loop_break:
            dec r17
    brne inner_loop_break
        dec r16
brne outer_loop_break

ret
;********************************************************************
; push_clear
;********************************************************************

push_clear:
    ldi r16, PORT_INT0_bm   ;clear IRQ flag for PE0
    sts PORTE_INTFLAGS, r16
    ret

;********************************************************************
; error loop
;********************************************************************
line2_testmessage: .db 1, "ERROR, press CLEAR", 0  ; message for line #1.

error_loop:
   ldi  ZH, high(line2_testmessage<<1)  ; pointer to line 1 memory buffer
   ldi  ZL, low(line2_testmessage<<1)   ;
   rcall load_msg          ; load message into buffer(s).
   rcall update_lcd_dog

   ldi r16, PORT_INT0_bm    ;clear IRQ flag for PE0
    sts PORTE_INTFLAGS, r16
   ret

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

```
; push enter
;*******************************************************************
addition_100th:
    dec r17
    ldi r16, 100
    mul r18, r16 // multiply by 100 for the 100th place value
    add r19, r0 // and then add the next digit on 1st
    adiw ZH:ZL, $0001
rjmp lookup2

addition_10th:
    dec r17
    ldi r16, 10 // to multiply ; shift to the left on 10th
    mul r18, r16    //shift to the left on 10th
    add r19, r0
    adiw ZH:ZL, $0001
rjmp lookup2


push_enter:

    ldi r17, 3
    ldi r18, 0x00
    ldi r19, 0x00
    ldi ZH, high (dsp_buff_1+12) ; Load ZH and ZL as a pointer to 1st
    ldi ZL, low (dsp_buff_1+12)  ; byte of buffer for line 1.

    ld r18, Z
    andi r18, 0x0F  // mask to translate from ascii code to numerical value

lookup2:

    cpi r17, 3
    breq addition_100th

    cpi r17, 2
    breq addition_10th

    // 1th addition
    add r19, r18



    cpi r19, 101    // check if the value is over 100
    brge error_loop // branch if it is equal or greater than 101

    // now convert the percentage value into value out of 255, and generate pulse

    cpi r19, 100
    breq brightness_full

    //cpi r19, 0
    //breq brightness_zero

    /*
    mov r20, r19
    lsr r20

    lsl r19

    add r19, r20
    */
    ldi r20, 255
    sub r20, r19

;***************************************************************************
; execute
;***************************************************************************
execute:
timing_loop:
mov r16, r19    // move it to r16 r19 dont change
mov r18, r20    // r20 dont change

    loop:
        sbi VPORTD_OUT, 0

    dec_loop:
        dec r16
        brne loop

    loop2:
        cbi VPORTD_OUT, 0

    dec_loop2:
        dec r18
        brne loop2

    rjmp timing_loop

;***************************************************************************
; shift_by_1
;***************************************************************************

shift_by_1:
    ldi ZH, high (dsp_buff_1+15) ; Load ZH and ZL as a pointer to 1st
    ldi ZL, low (dsp_buff_1+15)  ; byte of buffer for line 1.
    ldi r20, 0x20   //r20 is blank

    sbiw ZH:ZL, $0002
    ld r19, Z

    sbiw ZH:ZL, $0001
    st Z, r19

    adiw ZH:ZL, $0002
    ld r19, Z

    sbiw ZH:ZL, $0001
    st Z, r19

    adiw ZH:ZL, $0002
    ld r19, Z

    sbiw ZH:ZL, $0001
    st Z, r18

    adiw ZH:ZL, $0001
    st Z, r20

    ret
;********************************************************************
; brightness full (100%)
;********************************************************************
brightness_full:
    sbi VPORTD_OUT, 0
    rjmp brightness_full

;********************************************************************
; brightness zero (0%)
;********************************************************************
brightness_zero:
    loop43:
        sbi VPORTD_OUT, 0
    rjmp loop43
    cbi VPORTD_OUT, 0
    rjmp brightness_zero


;********************************************************************
;   clear line 1
;********************************************************************

line1_testmessage: .db 1, "Setting 1 : 000 ", 0  ; message for line #1.

clear_line:
      ;load_line_1 into dbuff1:
```

```
   ldi  ZH, high(line1_testmessage<<1)  ; pointer to line 1 memory buffer
   ldi  ZL, low(line1_testmessage<<1)   ;
   rcall load_msg          ; load message into buffer(s).

   ret

;*******************
;NAME:      load_msg
;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

;---------------------------- 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



;*********************************************************************


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