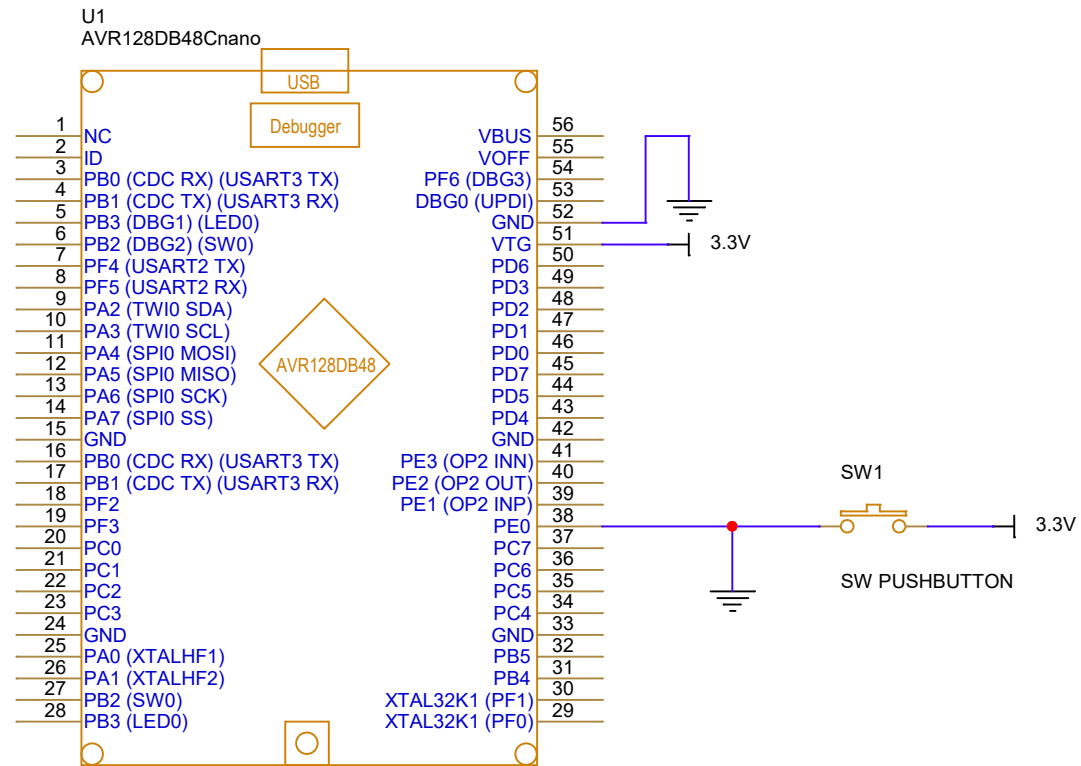Dongyun Lee

ID: 112794190
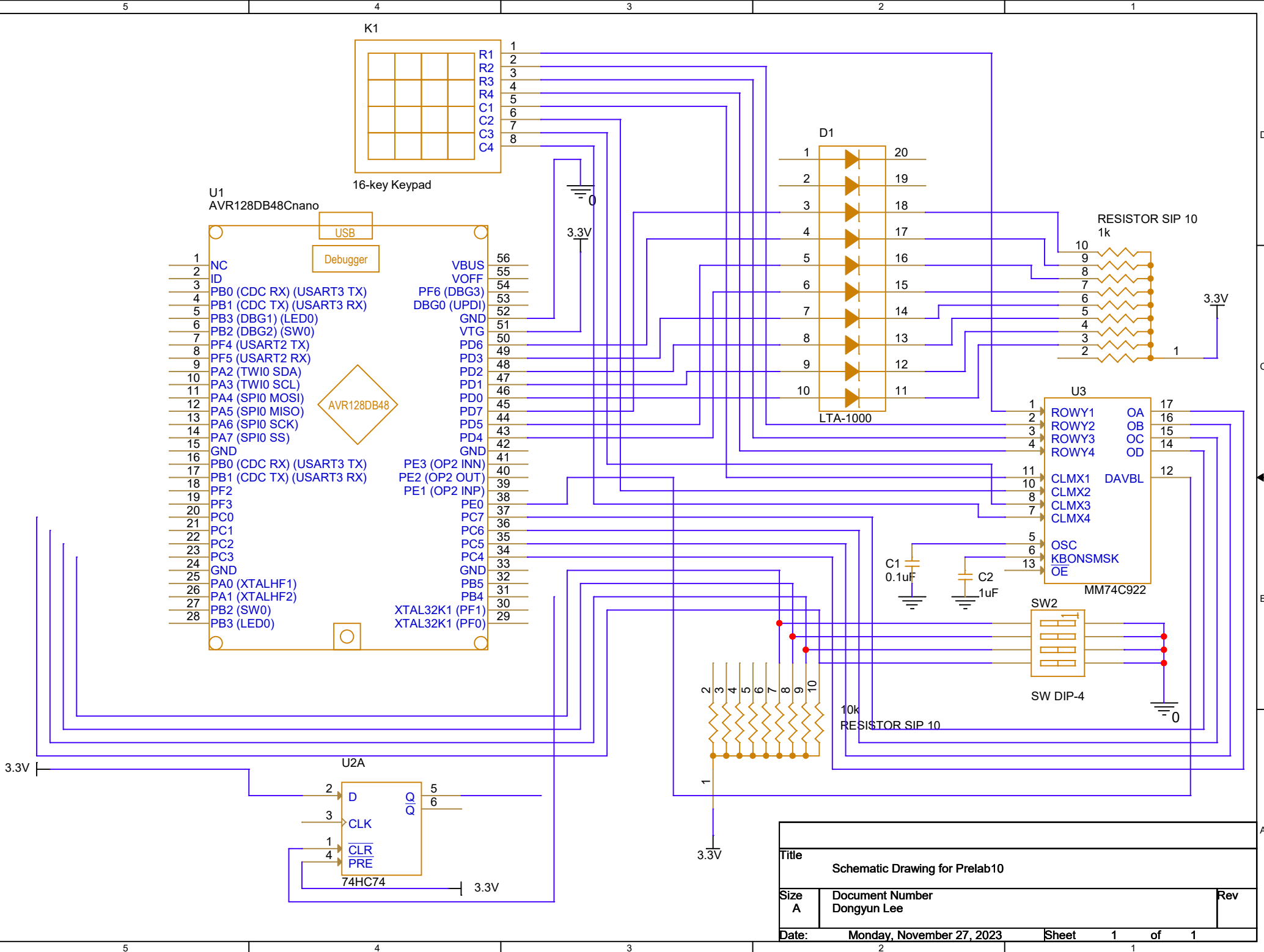
PreLab10: Polling and Interrupts Using Pin Change
Interrupt Flags (INTFs)

ESE280-L03

Bench #3

U1
AVR128DB48Cnano

USB

Debugger

AVR128DB48

Pin	Left Label
1	NC
2	ID
3	PB0 (CDC RX) (USART3 TX)
4	PB1 (CDC TX) (USART3 RX)
5	PB3 (DBG1) (LED0)
6	PB2 (DBG2) (SW0)
7	PF4 (USART2 TX)
8	PF5 (USART2 RX)
9	PA2 (TWI0 SDA)
10	PA3 (TWI0 SCL)
11	PA4 (SPI0 MOSI)
12	PA5 (SPI0 MISO)
13	PA6 (SPI0 SCK)
14	PA7 (SPI0 SS)
15	GND
16	PB0 (CDC RX) (USART3 TX)
17	PB1 (CDC TX) (USART3 RX)
18	PF2
19	PF3
20	PC0
21	PC1
22	PC2
23	PC3
24	GND
25	PA0 (XTALHF1)
26	PA1 (XTALHF2)
27	PB2 (SW0)
28	PB3 (LED0)

Right Label	Pin
VBUS	56
VOFF	55
PF6 (DBG3)	54
DBG0 (UPDI)	53
GND	52
VTG	51
PD6	50
PD3	49
PD2	48
PD1	47
PD0	46
PD7	45
PD5	44
PD4	43
GND	42
PE3 (OP2 INN)	41
PE2 (OP2 OUT)	40
PE1 (OP2 INP)	39
PE0	38
PC7	37
PC6	36
PC5	35
PC4	34
GND	33
PB5	32
PB4	31
XTAL32K1 (PF1)	30
XTAL32K1 (PF0)	29

3.3V

SW1

3.3V

SW PUSHBUTTON

Schematic Drawing for Prelab10

K1 — 16-key Keypad

R1	1
R2	2
R3	3
R4	4
C1	5
C2	6
C3	7
C4	8

U1 — AVR128DB48Cnano

USB
Debugger
AVR128DB48

Pin	Name
1	NC
2	ID
3	PB0 (CDC RX) (USART3 TX)
4	PB1 (CDC TX) (USART3 RX)
5	PB3 (DBG1) (LED0)
6	PB2 (DBG2) (SW0)
7	PF4 (USART2 TX)
8	PF5 (USART2 RX)
9	PA2 (TWI0 SDA)
10	PA3 (TWI0 SCL)
11	PA4 (SPI0 MOSI)
12	PA5 (SPI0 MISO)
13	PA6 (SPI0 SCK)
14	PA7 (SPI0 SS)
15	GND
16	PB0 (CDC RX) (USART3 TX)
17	PB1 (CDC TX) (USART3 RX)
18	PF2
19	PF3
20	PC0
21	PC1
22	PC2
23	PC3
24	GND
25	PA0 (XTALHF1)
26	PA1 (XTALHF2)
27	PB2 (SW0)
28	PB3 (LED0)

Name	Pin
VBUS	56
VOFF	55
PF6 (DBG3)	54
DBG0 (UPDI)	53
GND	52
VTG	51
PD6	50
PD3	49
PD2	48
PD1	47
PD0	46
PD7	45
PD5	44
PD4	43
GND	42
PE3 (OP2 INN)	41
PE2 (OP2 OUT)	40
PE1 (OP2 INP)	39
PE0	38
PC7	37
PC6	36
PC5	35
PC4	34
GND	33
PB5	32
PB4	31
XTAL32K1 (PF1)	30
XTAL32K1 (PF0)	29

D1 — LTA-1000

1	20
2	19
3	18
4	17
5	16
6	15
7	14
8	13
9	12
10	11

RESISTOR SIP 10 1k

U3 — MM74C922

Pin	Name	Name	Pin
1	ROWY1	OA	17
2	ROWY2	OB	16
3	ROWY3	OC	15
4	ROWY4	OD	14
11	CLMX1	DAVBL	12
10	CLMX2		
8	CLMX3		
7	CLMX4		
5	OSC		
6	KBONSMSK		
13	OE		

C1 0.1uF
C2 1uF

SW2 — SW DIP-4

10k RESISTOR SIP 10

U2A — 74HC74

Pin	Name
2	D
3	CLK
1	CLR
4	PRE

Name	Pin
Q	5
Q	6

3.3V

Title: Schematic Drawing for Prelab10

Size: A

Document Number: Dongyun Lee

Rev

Date: Monday, November 27, 2023   Sheet 1 of 1

/Volumes/DongyunLee/ESE280 Lab/Lab10/task2/task2/main.asm

```
1
2   start:
3       ; Configure I/O ports
4       cbi VPORTC_DIR, 0   ;PORTC input
5       ldi r16, 0x0F        ;make initial count value 0
6       out VPORTC_DIR, r16
7       ldi r16, 0xFF
8       out VPORTD_DIR, r16 //output bargraph
9       ldi r16, 0xFF
10      out VPORTD_OUT, r16
11
12      ;Configure interrupt request
13      lds r16, PORTE_PIN0CTRL ;set ISC for PE0 to rising edge
14      ori r16, 0x02        ;ISC = 2 for rising edge
15      sts PORTE_PIN0CTRL, r16
16
17  main_loop:       ;main program loop
18      ;Determine if PE0's INTF is set
19      lds r16, PORTE_INTFLAGS ;check for PE0 IRQ flag set
20      sbrc r16, 0
21      rcall output_bar          ;execute subroutine for PE0
22      rjmp main_loop
23
24  ;Subroutine called for PE0 INTF set
25  output_bar:      ;PE0's task to be done
26
27      in r18, VPORTC_IN   // gets the input from DIP switch and keypad
28
29      lsr r18
30      lsr r18
31      lsr r18
32      lsr r18
33
34      com r18          // complement r19 for display
35      out VPORTD_OUT, r18 // display
36
37      // just like clearing flip flop
38      ldi r16, PORT_INT0_bm   ;clear IRQ flag for PE0
39      sts PORTE_INTFLAGS, r16
40      ret
41
```

/Volumes/DongyunLee/ESE280 Lab/Lab10/task4/task4/main.asm

```asm
1    ;*********************************************************************
2    ;**********          BASIC DOG LCD TEST PROGRAM          **********
3    ;*********************************************************************
4    ;
5    ;DOG_LCD_BasicTest.asm
6    ;  Simple test application to verify DOG LCD is properly
7    ;  wired.  This test writes simple test messages to each
8    ;  line of the display.
9    ;
10   ;Version – 2.0 For DOGM163W LCD operated at 3.3V
11   ;
12
13       .CSEG
14
15       ; interrupt vector table, with several 'safety' stubs
16       rjmp RESET      ;Reset/Cold start vector
17       reti            ;External Intr0 vector
18       reti            ;External Intr1 vector
19
20   ;*********************************************************************
21   ;************ M A I N   A P P L I C A T I O N   C O D E  ************
22   ;*********************************************************************
23
24   .org PORTE_PORT_vect
25       jmp porte_isr       ;vector for all PORTE pin change IRQs
26
27   RESET:
28
29       sbi VPORTA_DIR, 7       ; set PA7 = output.
30       sbi VPORTA_OUT, 7       ; set /SS of DOG LCD = 1 (Deselected)
31
32       rcall init_lcd_dog    ; init display, using SPI serial interface
33       rcall clr_dsp_buffs   ; clear all three SRAM memory buffer lines
34
35       rcall update_lcd_dog        ;display data in memory buffer on LCD
36
37       rcall start
38
39   // display setting line
40       rcall clear_line
41
42       rcall update_lcd_dog
43
44
45       cbi VPORTE_DIR, 0    ;PE0 input– gets output from pushbutton debouce ckt.
46
47
48       ;Configure interrupt
49       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
```

```asm
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        ldi r16, PORT_INT0_bm    ;clear IRQ flag for PE0
225        sts PORTE_INTFLAGS, r16
226
227        pop r16           ;restore SREG then r16
228        out CPU_SREG, r16    ;note I in SREG now = 0
229        pop r16           ;restore SREG then r16
230        sei               ;SREG I = 1
231
232    lookup2:
233        ld r18, Z
234        andi r18, 0x0F  // mask to translate from ascii code to numerical value
235
236        cpi r17, 3
237        breq addition_100th
```

```asm
238
239        cpi r17, 2
240        breq addition_10th
241
242        // 1th addition
243        add r19, r18
244
245
246
247        cpi r19, 101    // check if the value is over 100
248        brge error_loop // branch if it is equal or greater than 101
249
250        // now convert the percentage value into value out of 255, and generate pulse
251
252        cpi r19, 100
253        breq brightness_full
254
255        cpi r19, 0
256        breq brightness_zero
257
258
259        mov r20, r19
260        lsr r20
261
262        lsl r19
263
264        add r19, r20
265
266        ldi r20, 255
267        sub r20, r19
268
269 ;******************************************************************
270 ; execute
271 ;******************************************************************
272 execute:
273 timing_loop:
274 mov r16, r19    // move it to r16 r19 dont change
275 mov r18, r20    // r20 dont change
276
277        loop:
278            sbi VPORTD_OUT, 0
279
280        dec_loop:
281            dec r16
282            brne loop
283
284        loop2:
285            cbi VPORTD_OUT, 0
286
287        dec_loop2:
288            dec r18
289            brne loop2
290
291        rjmp timing_loop
292
293 ;******************************************************************
294 ; shift_by_1
295 ;******************************************************************
296
297 shift_by_1:
```

```
298        ldi ZH, high (dsp_buff_1+15) ; Load ZH and ZL as a pointer to 1st
299        ldi ZL, low (dsp_buff_1+15)  ; byte of buffer for line 1.
300        ldi r20, 0x20   //r20 is blank
301
302        sbiw ZH:ZL, $0002
303        ld r19, Z
304
305        sbiw ZH:ZL, $0001
306        st Z, r19
307
308        adiw ZH:ZL, $0002
309        ld r19, Z
310
311        sbiw ZH:ZL, $0001
312        st Z, r19
313
314        adiw ZH:ZL, $0002
315        ld r19, Z
316
317        sbiw ZH:ZL, $0001
318        st Z, r18
319
320        adiw ZH:ZL, $0001
321        st Z, r20
322
323        ret
324 ;*********************************************************************
325 ; brightness full (100%)
326 ;*********************************************************************
327 brightness_full:
328        sbi VPORTD_OUT, 0
329        rjmp brightness_full
330
331 ;*********************************************************************
332 ; brightness zero (0%)
333 ;*********************************************************************
334 brightness_zero:
335        loop43:
336            sbi VPORTD_OUT, 0
337        rjmp loop43
338        cbi VPORTD_OUT, 0
339        rjmp brightness_zero
340
341
342 ;*********************************************************************
343 ;   clear line 1
344 ;*********************************************************************
345
346 line1_testmessage: .db 1, "Setting 1 : 000 ", 0  ; message for line #1.
347
348 clear_line:
349        ;load_line_1 into dbuff1:
350     ldi  ZH, high(line1_testmessage<<1)  ; pointer to line 1 memory buffer
351     ldi  ZL, low(line1_testmessage<<1)   ;
352     rcall load_msg         ; load message into buffer(s).
353
354     ret
355
356 ;******************
357 ;NAME:      load_msg
```

```asm
358  ;FUNCTION:   Loads a predefined string msg into a specified diplay
359  ;            buffer.
360  ;ASSUMES:    Z = offset of message to be loaded. Msg format is
361  ;            defined below.
362  ;RETURNS:    nothing.
363  ;MODIFIES:   r16, Y, Z
364  ;CALLS:      nothing
365  ;CALLED BY:
366  ;*******************************************************************
367  ; Message structure:
368  ;    label:  .db <buff num>, <text string/message>, <end of string>
369  ;
370  ; Message examples (also see Messages at the end of this file/module):
371  ;    msg_1: .db 1,"First Message ", 0   ; loads msg into buff 1, eom=0
372  ;    msg_2: .db 1,"Another message ", 0 ; loads msg into buff 1, eom=0
373  ;
374  ; Notes:
375  ;    a) The 1st number indicates which buffer to load (either 1, 2, or 3).
376  ;    b) The last number (zero) is an 'end of string' indicator.
377  ;    c) Y = ptr to disp_buffer
378  ;       Z = ptr to message (passed to subroutine)
379  ;*******************************************************************
380  load_msg:
381      ldi YH, high (dsp_buff_1) ; Load YH and YL as a pointer to 1st
382      ldi YL, low (dsp_buff_1)  ; byte of dsp_buff_1 (Note - assuming
383                                ; (dsp_buff_1 for now).
384      lpm R16, Z+               ; get dsply buff number (1st byte of msg).
385      cpi r16, 1                ; if equal to '1', ptr already setup.
386      breq get_msg_byte         ; jump and start message load.
387      adiw YH:YL, 16            ; else set ptr to dsp buff 2.
388      cpi r16, 2                ; if equal to '2', ptr now setup.
389      breq get_msg_byte         ; jump and start message load.
390      adiw YH:YL, 16            ; else set ptr to dsp buff 2.
391
392  get_msg_byte:
393      lpm R16, Z+               ; get next byte of msg and see if '0'.
394      cpi R16, 0                ; if equal to '0', end of message reached.
395      breq msg_loaded           ; jump and stop message loading operation.
396      st Y+, R16                ; else, store next byte of msg in buffer.
397      rjmp get_msg_byte         ; jump back and continue...
398  msg_loaded:
399      ret
400
401  ;-------------------------- SUBROUTINES ----------------------------
402
403
404  ;==================================
405  .include "lcd_dog_asm_driver_avr128.inc"  ; LCD DOG init/update procedures.
406  ;==================================
407
408
409  ;***********************
410  ;NAME:       clr_dsp_buffs
411  ;FUNCTION:   Initializes dsp_buffers 1, 2, and 3 with blanks (0x20)
412  ;ASSUMES:    Three CONTIGUOUS 16-byte dram based buffers named
413  ;            dsp_buff_1, dsp_buff_2, dsp_buff_3.
414  ;RETURNS:    nothing.
415  ;MODIFIES:   r25,r26, Z-ptr
416  ;CALLS:      none
417  ;CALLED BY: main application and diagnostics
```

```
418 ;*****************************************************************
419 clr_dsp_buffs:
420     ldi R25, 48              ; load total length of both buffer.
421     ldi R26, ' '             ; load blank/space into R26.
422     ldi ZH, high (dsp_buff_1) ; Load ZH and ZL as a pointer to 1st
423     ldi ZL, low (dsp_buff_1)  ; byte of buffer for line 1.
424
425     ;set DDRAM address to 1st position of first line.
426 store_bytes:
427     st  Z+, R26         ; store ' ' into 1st/next buffer byte and
428                         ; auto inc ptr to next location.
429     dec  R25            ;
430     brne store_bytes  ; cont until r25=0, all bytes written.
431     ret
432
433
434
435 ;*****************************************************************
436
437
438 ;***** END OF FILE *****
439
```