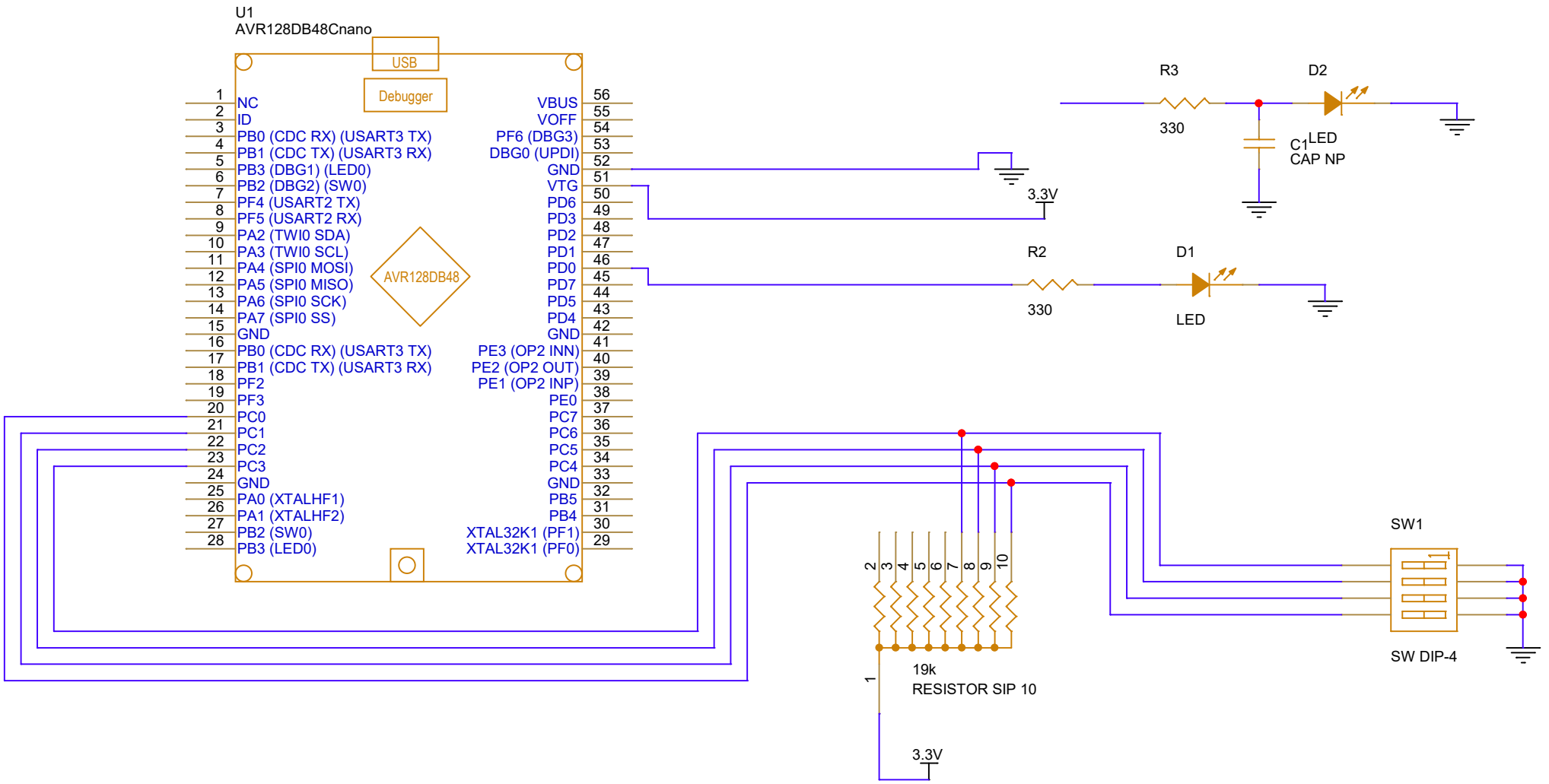
Dongyun Lee

ID: 112794190

PreLab09: Software PWM

ESE280-L03

Bench #3



Title	PreLab09 Dongyun Lee	
Size A	Document Number Dongyun Lee	Rev
Date: Monday, November 06, 2023	Sheet 1 of 1	

/Volumes/DongyunLee/ESE280 Lab/Lab9/task1/task1/main.asm

```
start:
    ldi r17, 0x00
    out VPORTC_DIR, r17
    sbi VPORTD_DIR, 0


main_loop:
    in r16, VPORTC_IN   // get input from switch

    andi r16, 0x0F  // mask 0000 1111 4 least significant bits

    rcall lookup

    cpi r16, 0x00
    breq always_off
    cpi r16, 0x01
    breq r16_is_1
    cpi r16, 0x02
    breq r16_is_2
    cpi r16, 0x03
    breq r16_is_3
    cpi r16, 0x04
    breq r16_is_4
    cpi r16, 0x05
    breq r16_is_5
    cpi r16, 0x06
    breq r16_is_6
    cpi r16, 0x07
    breq r16_is_7
    cpi r16, 0x08
    breq r16_is_8
    cpi r16, 0x09
    breq r16_is_9
    cpi r16, 0x0A
    breq r16_is_A
    cpi r16, 0x0B
    breq r16_is_B
    cpi r16, 0x0C
    breq r16_is_C
    cpi r16, 0x0D
    breq r16_is_D
    cpi r16, 0x0E
    breq r16_is_E
    cpi r16, 0x0F
    breq always_on

    rjmp main_loop


r16_is_1:
    ldi r16, 238
    rjmp timing_loop

r16_is_2:
    ldi r16, 221
    rjmp timing_loop

```

```
r16_is_3:
    ldi r16, 204
    rjmp timing_loop

r16_is_4:
    ldi r16, 187
    rjmp timing_loop

r16_is_5:
    ldi r16, 170
    rjmp timing_loop

r16_is_6:
    ldi r16, 153
    rjmp timing_loop

r16_is_7:
    ldi r16, 136
    rjmp timing_loop

r16_is_8:
    ldi r16, 119
    rjmp timing_loop

r16_is_9:
    ldi r16, 102
    rjmp timing_loop

r16_is_A:
    ldi r16, 85
    rjmp timing_loop

r16_is_B:
    ldi r16, 68
    rjmp timing_loop

r16_is_C:
    ldi r16, 51
    rjmp timing_loop

r16_is_D:
    ldi r16, 34
    rjmp timing_loop

r16_is_E:
    ldi r16, 17
    rjmp timing_loop

r16_is_F:
    ldi r16, 0
    rjmp always_on

always_off:
    cbi VPORTD_OUT, 0

    rjmp main_loop

always_on:
    sbi VPORTD_OUT, 0

```

```
118     rjmp main_loop
119
120 timing_loop:
121
122     ldi r20, 255
123     sub r20, r16
124
125
126     loop:
127         sbi VPORTD_OUT, 0
128
129     dec_loop:
130         dec r20
131         brne loop
132
133         cbi VPORTD_OUT, 0
134
135     loop2:
136         cbi VPORTD_OUT, 0
137
138     dec_loop2:
139         dec r16
140         brne loop2
141
142         sbi VPORTD_OUT, 0
143         rjmp main_loop
144
145
146 table: .db $00, $01, $02, $03, $04, $05, $06, $07, $08, $09, $0A, $0B, $0C, $0D,
    $0E, $0F
147
148 lookup:
149     ldi ZH, high (table*2)
150     ldi ZL, low (table*2)
151     ldi r18, $00
152     add ZL, r16
153     adc ZH, r18
154     lpm r16, Z
155
156     ret
157
```

/Volumes/DongyunLee/ESE280 Lab/Lab9/task2/task2/main.asm

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

    rcall update_lcd_dog

    // 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

    rjmp output

    rjmp check_press

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

;*****************************************************************************
; start subroutine
;*****************************************************************************
start:
```

```
58      sbi VPORTA_DIR, 4    //MOSI output
59
60      sbi VPORTB_DIR, 4    // clear flip flop output
61      sbi VPORTB_OUT, 4   // set clear to 1
62
63      ldi r17, 0x00
64      out VPORTC_DIR, r17 // input 4 dip switch + 16 keypads
65      sbi VPORTD_DIR, 0   // pulse generator
66
67      cbi VPORTB_DIR, 5    // check if the keypad is pressed
68
69      ldi XH, high (dsp_buff_1+15) ; Load ZH and ZL as a pointer to 1st
70      ldi XL, low (dsp_buff_1+15)  ; byte of buffer for line 1.
71
72      ret
73  ;***********************************************************************
74  ; keypad subroutine
75  ;***********************************************************************
76  table: .db $31, $32, $33, $46
77         .db $34, $35, $36, $45
78         .db $37, $38, $39, $44
79         .db $41, $30, $42, $43
80
81
82  output:
83  in r18, VPORTC_IN   // gets the input from DIP switch and keypad
84
85  lsr r18     // shifting  to right 4 bits
86  lsr r18
87  lsr r18
88  lsr r18
89
90
91  // lookup table from lecture
92  lookup:
93      ldi ZH, high (table*2)
94      ldi ZL, low (table*2)
95      ldi r16, $00
96      add ZL, r18
97      adc ZH, r16
98      lpm r18, Z
99
100     st X, r18  // storing into SRAM buffer
101
102     clear_flipflop:     // clear the flip flop for next input
103     cbi VPORTB_OUT, 4
104     sbi VPORTB_OUT, 4
105
106     cpi r18, $41    // if the pressed key is clear
107         breq push_clear
108
109     cpi r18, $43    // if the pressed key is Enter
110         breq enter_clear
111
112     rcall shift_by_1
113
114     rcall delay_break
115
116     rcall update_lcd_dog
117
```

```
118 rjmp check_press    // go back to the start
119
120
121
122 ;***********************************************************************
123 ; delay break
124 ;***********************************************************************
125 delay_break:            ;delay lable for break delay
126     ldi r16, 80
127     outer_loop_break:
128         ldi r17, 133
129         inner_loop_break:
130             dec r17
131     brne inner_loop_break
132         dec r16
133 brne outer_loop_break
134
135 ret
136 ;***********************************************************************
137 ; push_clear
138 ;***********************************************************************
139
140 push_clear:
141     rjmp RESET
142
143 ;***********************************************************************
144 ; error loop
145 ;***********************************************************************
146 line2_testmessage: .db 1, "ERROR, press CLEAR", 0  ; message for line #1.
147
148 error_loop:
149    ldi  ZH, high(line2_testmessage<<1)  ; pointer to line 1 memory buffer
150    ldi  ZL, low(line2_testmessage<<1)   ;
151    rcall load_msg          ; load message into buffer(s).
152    rcall update_lcd_dog
153
154 wait_for_clear:
155         sbis VPORTB_IN, 5   ;wait for PB5 being 1
156         rjmp wait_for_clear     ;skip this line if PE0 is 1
157
158 output_error:
159 in r18, VPORTC_IN   // gets the input from DIP switch and keypad
160
161 lsr r18     // shifting  to right 4 bits
162 lsr r18
163 lsr r18
164 lsr r18
165
166
167 // lookup table from lecture
168 lookup_error:
169     ldi ZH, high (table*2)
170     ldi ZL, low (table*2)
171     ldi r16, $00
172     add ZL, r18
173     adc ZH, r16
174     lpm r18, Z
175
176     cpi r18, $41    // if the pressed key is clear
177         breq push_clear
```

```
178
179
180 rjmp output_error
181 ;***********************************************************************
182 ; push enter
183 ;***********************************************************************
184 addition_100th:
185     dec r17
186     ldi r16, 100
187     mul r18, r16 // multiply by 100 for the 100th place value
188     add r19, r0 // and then add the next digit on 1st
189     adiw ZH:ZL, $0001
190 rjmp lookup2
191
192 addition_10th:
193     dec r17
194     ldi r16, 10 // to multiply ; shift to the left on 10th
195     mul r18, r16    //shift to the left on 10th
196     add r19, r0
197     adiw ZH:ZL, $0001
198 rjmp lookup2
199
200 enter_clear:
201 // clear the flip flop for next input
202     cbi VPORTB_OUT, 4
203     sbi VPORTB_OUT, 4
204
205 push_enter: // error: clear button does not work once enter is pressed
206
207     ldi r17, 3
208     ldi r18, 0x00
209     ldi r19, 0x00
210     ldi ZH, high (dsp_buff_1+12) ; Load ZH and ZL as a pointer to 1st
211     ldi ZL, low (dsp_buff_1+12)  ; byte of buffer for line 1.
212
213 lookup2:
214     ld r18, Z
215     andi r18, 0x0F  // mask
216
217     cpi r17, 3
218     breq addition_100th
219
220     cpi r17, 2
221     breq addition_10th
222
223     // 1th addition
224     add r19, r18
225
226     sbic VPORTB_IN, 5   ;wait for PB5 being 1
227        rjmp output
228
229
230     cpi r19, 101    // check if the value is over 100
231     brge error_loop // branch if it is equal or greater than 101
232
233     // now convert the percentage value into value out of 255, and generate pulse
234
235     cpi r19, 100
236     breq birghtness_full
237
```

```
238     cpi r19, 0
239     breq birghtness_zero
240 
241     ldi r16, 2
242     mul r19, r16 // multiply r19 by 2 (r16)
243     mov r19, r0
244 
245 timing_loop:
246 
247     ldi r20, 255
248     sub r20, r19
249 
250 
251     loop:
252         sbi VPORTD_OUT, 0
253 
254     dec_loop:
255         dec r19
256         brne loop
257 
258     loop2:
259         cbi VPORTD_OUT, 0
260 
261     dec_loop2:
262         dec r20
263         brne loop2
264 
265         //sbi VPORTD_OUT, 0
266     rjmp push_enter
267 
268 
269 
270 
271 ;***************************************************************************
272 ; shift_by_1
273 ;***************************************************************************
274 
275 shift_by_1:
276     ldi ZH, high (dsp_buff_1+15) ; Load ZH and ZL as a pointer to 1st
277     ldi ZL, low (dsp_buff_1+15)  ; byte of buffer for line 1.
278     ldi r20, 0x20   //r20 is blank
279 
280     sbiw ZH:ZL, $0002
281     ld r19, Z
282 
283     sbiw ZH:ZL, $0001
284     st Z, r19
285 
286     adiw ZH:ZL, $0002
287     ld r19, Z
288 
289     sbiw ZH:ZL, $0001
290     st Z, r19
291 
292     adiw ZH:ZL, $0002
293     ld r19, Z
294 
295     sbiw ZH:ZL, $0001
296     st Z, r18
297 
```

```
    adiw ZH:ZL, $0001
    st Z, r20

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

    rjmp push_enter

;*********************************************************************
; brightness zero (0%)
;*********************************************************************
birghtness_zero:
    cbi VPORTD_OUT, 0

    rjmp push_enter


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

```

/Volumes/DongyunLee/ESE280 Lab/Lab9/task3lab9.asm

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

    rcall update_lcd_dog

    // 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

    rjmp output

    rjmp check_press

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

;*****************************************************************************
; start subroutine
;*****************************************************************************
start:
```

```
58      sbi VPORTA_DIR, 4    //MOSI output
59
60      sbi VPORTB_DIR, 4    // clear flip flop output
61      sbi VPORTB_OUT, 4   // set clear to 1
62
63      ldi r17, 0x00
64      out VPORTC_DIR, r17 // input 4 dip switch + 16 keypads
65      sbi VPORTD_DIR, 0   // pulse generator
66
67      cbi VPORTB_DIR, 5    // check if the keypad is pressed
68
69      ldi XH, high (dsp_buff_1+15) ; Load ZH and ZL as a pointer to 1st
70      ldi XL, low (dsp_buff_1+15)  ; byte of buffer for line 1.
71
72      ldi r19, 0x00 // register for storing value
73
74      ret
75  ;***************************************************************************
76  ; keypad subroutine
77  ;***************************************************************************
78  table: .db $31, $32, $33, $46
79         .db $34, $35, $36, $45
80         .db $37, $38, $39, $44
81         .db $41, $30, $42, $43
82
83  second_output:  // changing the pointer to the second line which is for T multiply
84      ldi XH, high (dsp_buff_2+15) ; Load ZH and ZL as a pointer to 1st
85      ldi XL, low (dsp_buff_2+15)  ; byte of buffer for line 1.
86      rjmp output2
87
88  output:
89  cpi r19, 0          // if r19 is not 0, which means that the first enter has been
    pressed yet, so change the pointer to the next line
90  brne second_output
91
92  output2:
93  in r18, VPORTC_IN   // gets the input from DIP switch and keypad
94
95  lsr r18     // shifting  to right 4 bits
96  lsr r18
97  lsr r18
98  lsr r18
99
100
101 // lookup table from lecture
102 lookup:
103     ldi ZH, high (table*2)
104     ldi ZL, low (table*2)
105     ldi r16, $00
106     add ZL, r18
107     adc ZH, r16
108     lpm r18, Z
109
110     st X, r18  // storing into SRAM buffer
111
112     clear_flipflop:     // clear the flip flop for next input
113     cbi VPORTB_OUT, 4
114     sbi VPORTB_OUT, 4
115
116     cpi r18, $41    // if the pressed key is clear
```

```
117         breq push_clear
118
119     cpi r18, $43    // if the pressed key is Enter
120         breq check_which_push_enter
121
122     rcall shift_by_1
123
124     //rcall delay_break
125
126     rcall update_lcd_dog
127
128 rjmp check_press    // go back to the start
129
130 ;***********************************************************************
131 ; check_which_push_enter
132 ;***********************************************************************
133 check_which_push_enter:
134     cpi r19, 0
135     breq enter_clear
136     rjmp second_enter_clear
137 ;***********************************************************************
138 ; delay break
139 ;***********************************************************************
140 delay_break:            ;delay lable for break delay
141     ldi r16, 80
142     outer_loop_break:
143         ldi r17, 133
144         inner_loop_break:
145             dec r17
146     brne inner_loop_break
147         dec r16
148 brne outer_loop_break
149
150 ret
151 ;***********************************************************************
152 ; push_clear
153 ;***********************************************************************
154
155 push_clear:
156     rjmp RESET
157
158 ;***********************************************************************
159 ; error loop
160 ;***********************************************************************
161 line3_testmessage: .db 3, "ERROR, press CLEAR", 0  ; message for line #1.
162
163 error_loop:
164     rcall clr_dsp_buffs
165    ldi  ZH, high(line3_testmessage<<1)  ; pointer to line 1 memory buffer
166    ldi  ZL, low(line3_testmessage<<1)   ;
167    rcall load_msg          ; load message into buffer(s).
168    rcall update_lcd_dog
169
170 wait_for_clear:
171         sbis VPORTB_IN, 5   ;wait for PB5 being 1
172         rjmp wait_for_clear     ;skip this line if PE0 is 1
173
174 output_error:
175 in r18, VPORTC_IN   // gets the input from DIP switch and keypad
176
```

```
177 lsr r18     // shifting  to right 4 bits
178 lsr r18
179 lsr r18
180 lsr r18
181
182
183 // lookup table from lecture
184 lookup_error:
185     ldi ZH, high (table*2)
186     ldi ZL, low (table*2)
187     ldi r16, $00
188     add ZL, r18
189     adc ZH, r16
190     lpm r18, Z
191
192     cpi r18, $41    // if the pressed key is clear
193         breq push_clear
194
195
196 rjmp output_error
197
198 ;*********************************************************************
199 ; push enter
200 ; r19 is the storage
201 ;*********************************************************************
202 100th_addition:
203     dec r17
204     ldi r16, 100
205     mul r18, r16 // multiply by 100 for the 100th place value. Stores in r0
206     add r19, r0 // and then add the next digit on 1st
207     adiw ZH:ZL, $0001
208 rjmp lookup2
209
210 10th_addition:
211     dec r17
212     ldi r16, 10 // to multiply ; shift to the left on 10th
213     mul r18, r16    //shift to the left on 10th  stores in r0
214     add r19, r0
215     adiw ZH:ZL, $0001
216 rjmp lookup2
217
218 enter_clear:
219     // clear the flip flop for next input
220     cbi VPORTB_OUT, 4
221     sbi VPORTB_OUT, 4
222
223 push_enter: // error: clear button does not work once enter is pressed
224
225     ldi r17, 3
226     ldi r19, 0x00
227     ldi ZH, high (dsp_buff_1+12) ; Load ZH and ZL as a pointer to 1st
228     ldi ZL, low (dsp_buff_1+12)  ; byte of buffer for line 1.
229
230     lookup2:
231         ld r18, Z
232         andi r18, 0x0F  // mask
233
234
235     sbic VPORTB_IN, 5   ;wait for PB5 being 1
236         rjmp output
```

```
237
238     cpi r19, 101    // check if the value is over 100
239     brge error_loop // branch if it is equal or greater than 101
240
241     // now convert the percentage value into value out of 255, and generate pulse
242
243     cpi r19, 100
244     breq birghtness_full
245
246     cpi r19, 0
247     breq birghtness_zero
248
249     ldi r16, 2
250     mul r19, r16 // multiply r19 by 2 (r16)
251     mov r19, r0
252
253     rjmp check_press
254
255 ;*********************************************************************
256 ; second push enter
257 ;  should be range of 1 - 100
258 ; r21 is the storage
259 ;*********************************************************************
260 100th_addition_2:
261     dec r17
262     ldi r16, 100
263     mul r18, r16 // multiply by 100 for the 100th place value
264     add r21, r0 // and then add the next digit on 1st
265     adiw ZH:ZL, $0001
266 rjmp lookup3
267
268 10th_addition_2:
269     dec r17
270     ldi r16, 10 // to multiply ; shift to the left on 10th
271     mul r18, r16    //shift to the left on 10th
272     add r21, r0
273     adiw ZH:ZL, $0001
274 rjmp lookup3
275
276 second_enter_clear:
277     // clear the flip flop for next input
278     cbi VPORTB_OUT, 4
279     sbi VPORTB_OUT, 4
280
281 second_enter:   // error: clear button does not work once enter is pressed
282     ldi r21, 0x00 // r21 is the storage for second enter which is T multiply
283     ldi r17, 3
284     ldi ZH, high (dsp_buff_2+12) ; Load ZH and ZL as a pointer to 1st
285     ldi ZL, low (dsp_buff_2+12)  ; byte of buffer for line 1.
286 lookup3:
287     ld r18, Z
288     andi r18, 0x0F  // mask
289
290     sbic VPORTB_IN, 5   ;if no key is pressed then skip next line
291         rjmp output     ; if you see a key is pressed go to output
292
293     cpi r17, 3
294     breq 100th_addition
295
296     cpi r17, 2
```

```
297     breq 10th_addition
298
299     // 1th addition
300     add r21, r18
301
302 ;***************************************************************************
303 ; execute
304 ;***************************************************************************
305 execute:
306     ldi r16, 2
307     mul r19, r16 // multiply r19 by 2 (r16)
308     mov r19, r0
309     mov r16, r19
310
311     ldi r20, 255
312     sub r20, r19
313
314     mov r17, r21    // r17 and r21 is the t multiply
315     rjmp highloop
316 timing_loop:
317     mov r19, r16
318     ldi r20, 255
319     sub r20, r19
320
321     highloop:
322         sbi VPORTD_OUT, 0
323
324     dec_loop:
325         dec r19
326         brne highloop
327         dec r21
328         brne timing_loop
329         rjmp lowloop2
330 timing_loop2:
331     mov r19, r16
332     ldi r20, 255
333     sub r20, r19
334
335     lowloop2:
336         cbi VPORTD_OUT, 0
337
338     dec_loop2:
339         dec r20
340         brne loop2
341         dec r17
342         brne timing_loop2
343
344 rjmp push_enter
345 ;***************************************************************************
346 ; shift_by_1
347 ;***************************************************************************
348 second_line_shift:
349     ldi ZH, high (dsp_buff_2+15) ; Load ZH and ZL as a pointer to 1st
350     ldi ZL, low (dsp_buff_2+15)  ; byte of buffer for line 1.
351     rjmp shift_by_1_2
352 shift_by_1:
353 cpi r19, 0
354 brne second_line_shift
355     ldi ZH, high (dsp_buff_1+15) ; Load ZH and ZL as a pointer to 1st
356     ldi ZL, low (dsp_buff_1+15)  ; byte of buffer for line 1.
```

```
357 shift_by_1_2:
358
359     sbiw ZH:ZL, $0002
360     ld r20, Z
361
362     sbiw ZH:ZL, $0001
363     st Z, r20
364
365     adiw ZH:ZL, $0002
366     ld r20, Z
367
368     sbiw ZH:ZL, $0001
369     st Z, r20
370
371     adiw ZH:ZL, $0002
372     ld r20, Z
373
374     sbiw ZH:ZL, $0001
375     st Z, r18
376 ldi r20, 0x20   //r20 is blank
377     adiw ZH:ZL, $0001
378     st Z, r20
379
380     ret // i am not sure if this return will still work since I have branched to
    somewhere in the middle.
381 ;*********************************************************************
382 ; brightness full (100%)
383 ;*********************************************************************
384 birghtness_full:
385     sbi VPORTD_OUT, 0
386
387     rjmp push_enter
388
389 ;*********************************************************************
390 ; brightness zero (0%)
391 ;*********************************************************************
392 birghtness_zero:
393     cbi VPORTD_OUT, 0
394
395     rjmp push_enter
396
397
398 ;*********************************************************************
399 ;   clear line 1
400 ;*********************************************************************
401
402 line1_testmessage: .db 1, "Setting 1  :000 ", 0  ; message for line #1.
403 line2_testmessage: .db 2, "T multiply :000 ", 0
404
405 clear_line:
406       ;load_line_1 into dbuff1:
407    ldi  ZH, high(line1_testmessage<<1)  ; pointer to line 1 memory buffer
408    ldi  ZL, low(line1_testmessage<<1)   ;
409    rcall load_msg          ; load message into buffer(s).
410
411    ldi  ZH, high(line2_testmessage<<1)  ; pointer to line 1 memory buffer
412    ldi  ZL, low(line2_testmessage<<1)   ;
413    rcall load_msg          ; load message into buffer(s).
414
415    ret
```

```
416
417 ;********************
418 ;NAME:      load_msg
419 ;FUNCTION:  Loads a predefined string msg into a specified diplay
420 ;           buffer.
421 ;ASSUMES:   Z = offset of message to be loaded. Msg format is
422 ;           defined below.
423 ;RETURNS:   nothing.
424 ;MODIFIES:  r16, Y, Z
425 ;CALLS:     nothing
426 ;CALLED BY:
427 ;*****************************************************************
428 ; Message structure:
429 ;   label:  .db <buff num>, <text string/message>, <end of string>
430 ;
431 ; Message examples (also see Messages at the end of this file/module):
432 ;   msg_1: .db 1,"First Message ", 0   ; loads msg into buff 1, eom=0
433 ;   msg_2: .db 1,"Another message ", 0 ; loads msg into buff 1, eom=0
434 ;
435 ; Notes:
436 ;   a) The 1st number indicates which buffer to load (either 1, 2, or 3).
437 ;   b) The last number (zero) is an 'end of string' indicator.
438 ;   c) Y = ptr to disp_buffer
439 ;      Z = ptr to message (passed to subroutine)
440 ;*****************************************************************
441 load_msg:
442      ldi YH, high (dsp_buff_1) ; Load YH and YL as a pointer to 1st
443      ldi YL, low (dsp_buff_1)  ; byte of dsp_buff_1 (Note - assuming
444                                ; (dsp_buff_1 for now).
445      lpm R16, Z+               ; get dsply buff number (1st byte of msg).
446      cpi r16, 1                ; if equal to '1', ptr already setup.
447      breq get_msg_byte         ; jump and start message load.
448      adiw YH:YL, 16            ; else set ptr to dsp buff 2.
449      cpi r16, 2                ; if equal to '2', ptr now setup.
450      breq get_msg_byte         ; jump and start message load.
451      adiw YH:YL, 16            ; else set ptr to dsp buff 2.
452
453 get_msg_byte:
454      lpm R16, Z+               ; get next byte of msg and see if '0'.
455      cpi R16, 0                ; if equal to '0', end of message reached.
456      breq msg_loaded           ; jump and stop message loading operation.
457      st Y+, R16                ; else, store next byte of msg in buffer.
458      rjmp get_msg_byte         ; jump back and continue...
459 msg_loaded:
460      ret
461
462 ;---------------------------- SUBROUTINES ----------------------------
463
464 ;************************
465 ;NAME:      clr_dsp_buffs
466 ;FUNCTION:  Initializes dsp_buffers 1, 2, and 3 with blanks (0x20)
467 ;ASSUMES:   Three CONTIGUOUS 16-byte dram based buffers named
468 ;           dsp_buff_1, dsp_buff_2, dsp_buff_3.
469 ;RETURNS:   nothing.
470 ;MODIFIES:  r25,r26, Z-ptr
471 ;CALLS:     none
472 ;CALLED BY: main application and diagnostics
473 ;*****************************************************************
474 clr_dsp_buffs:
475      ldi R25, 48               ; load total length of both buffer.
```

```
476      ldi R26, ' '              ; load blank/space into R26.
477      ldi ZH, high (dsp_buff_1) ; Load ZH and ZL as a pointer to 1st
478      ldi ZL, low (dsp_buff_1)  ; byte of buffer for line 1.
479
480     ;set DDRAM address to 1st position of first line.
481 store_bytes:
482      st  Z+, R26       ; store ' ' into 1st/next buffer byte and
483                        ; auto inc ptr to next location
484
485 ;===================================
486 .include "lcd_dog_asm_driver_avr128.inc"  ; LCD DOG init/update procedures.
487 ;===================================
488
489
```