Dongyun Lee

ID: 112794190

Lab11: Hardware PWM Using TCA0

ESE280-L03

Bench #3

~/Desktop/SBU 2023 Fall/ESE280/ESE280 Lab/Lab11/lab11task1.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  .equ PERIOD_EXAMPLE_VALUE = 100     ; 1% resolution for duty cycle setting
14
15      .CSEG
16
17      ; interrupt vector table, with several 'safety' stubs
18      rjmp RESET      ;Reset/Cold start vector
19      reti            ;External Intr0 vector
20      reti            ;External Intr1 vector
21
22  ;*******************************************************************
23  ;************ M A I N   A P P L I C A T I O N   C O D E ************
24  ;*******************************************************************
25
26  .org PORTE_PORT_vect
27      jmp porte_isr       ;vector for all PORTE pin change IRQs
28
29  RESET:
30
31      sbi VPORTA_DIR, 7       ; set PA7 = output.
32      sbi VPORTA_OUT, 7       ; set /SS of DOG LCD = 1 (Deselected)
33
34      rcall init_lcd_dog    ; init display, using SPI serial interface
35      rcall clr_dsp_buffs   ; clear all three SRAM memory buffer lines
36
37      rcall update_lcd_dog        ;display data in memory buffer on LCD
38
39      rcall start
40
41  // display setting line
42      rcall clear_line
43
44      rcall update_lcd_dog
45
46
47      cbi VPORTE_DIR, 0   ;PE0 input– gets output from pushbutton debouce ckt.
48
49
50      ;Configure interrupt
51      lds r16, PORTE_PIN0CTRL ;set ISC for PE0 to pos. edge
52      ori r16, 0x02           // positive edge detect
53      sts PORTE_PIN0CTRL, r16
54
55      sei                     ;enable global interrupts
56
57      main_loop:      ;infinite loop, program's task is complete
```

```
 58          nop
 59      rjmp main_loop
 60
 61  ;*******************************************************************
 62  ; start subroutine
 63  ;*******************************************************************
 64  start:
 65      sbi VPORTA_DIR, 4     //MOSI output
 66
 67    // sbi VPORTB_DIR, 4    // clear flip flop output
 68     //sbi VPORTB_OUT, 4 // set clear to 1
 69
 70      ldi r17, 0x00
 71      out VPORTC_DIR, r17 // input 4 dip switch + 16 keypads
 72      sbi VPORTD_DIR, 0   // pulse generator
 73      cbi VPORTD_OUT, 0
 74
 75      sbi VPORTD_DIR, 1
 76      cbi VPORTD_DIR, 1
 77
 78      //cbi VPORTB_DIR, 5    // check if the keypad is pressed
 79
 80      ldi XH, high (dsp_buff_1+15) ; Load ZH and ZL as a pointer to 1st
 81      ldi XL, low (dsp_buff_1+15)  ; byte of buffer for line 1.
 82
 83  ;@@@@@@@@@@@@@@@@@@ new stuff
 84      ;Route WO0 to PD0 instead of its default pin PA0
 85      ldi r16, 0x03        ;mux TCA0 WO0 to PD0
 86      sts PORTMUX_TCAROUTEA, r16
 87
 88      ;Set CTRLB to use CMP0 and single slope PWM
 89      ldi r16, TCA_SINGLE_CMP0EN_bm | TCA_SINGLE_WGMODE_SINGLESLOPE_gc ;CMP0EN and
    single slope PWM
 90      sts TCA0_SINGLE_CTRLB, r16
 91  ;@@@@@@@@@@@@@@@@@@
 92      ret
 93
 94  ;*******************************************************************
 95  ; interrupt service routine
 96  ;*******************************************************************
 97  ;Interrupt service routine for any PORTE pin change IRQ
 98  porte_ISR:
 99      cli              ;clear global interrupt enable, I = 0
100
101      push r16         ;save r16 then SREG, note I = 0
102      in r16, CPU_SREG
103      push r16
104
105      ;Determine which pins of PORTE have IRQs
106      lds r18, PORTE_INTFLAGS ;check for PE0 IRQ flag set
107      sbrc r18, 0
108      rcall output             ;execute subroutine for PE0
109
110      pop r16          ;restore SREG then r16
111      out CPU_SREG, r16   ;note I in SREG now = 0
112      pop r16          ;restore SREG then r16
113      sei              ;SREG I = 1
114      reti             ;return from PORTE pin change ISR
115  ;Note: reti does not set I on an AVR128DB48
116
```

```
117  ;****************************************************************
118  ; keypad subroutine
119  ;****************************************************************
120  table: .db $31, $32, $33, $46
121         .db $34, $35, $36, $45
122         .db $37, $38, $39, $44
123         .db $41, $30, $42, $43
124
125
126  output:
127
128  in r18, VPORTC_IN   // gets the input from DIP switch and keypad
129
130  lsr r18     // shifting  to right 4 bits
131  lsr r18
132  lsr r18
133  lsr r18
134
135
136  // lookup table from lecture
137  lookup:
138      ldi ZH, high (table*2)
139      ldi ZL, low (table*2)
140      ldi r16, $00
141      add ZL, r18
142      adc ZH, r16
143      lpm r18, Z
144
145      st X, r18  // storing into SRAM buffer
146
147
148      ldi r16, PORT_INT0_bm   ;clear IRQ flag for PE0
149      sts PORTE_INTFLAGS, r16
150
151      cpi r18, $41    // if the pressed key is clear
152          breq push_clear
153
154      cpi r18, $43    // if the pressed key is Enter
155          breq push_enter
156
157      rcall shift_by_1
158
159      rcall delay_break
160
161      rcall update_lcd_dog
162
163
164  ret
165  //rjmp main_loop    // go back to the start
166
167
168
169  ;****************************************************************
170  ; delay break
171  ;****************************************************************
172  delay_break:            ;delay lable for break delay
173      ldi r16, 80
174      outer_loop_break:
175          ldi r17, 133
176          inner_loop_break:
```

```
177              dec r17
178        brne inner_loop_break
179              dec r16
180   brne outer_loop_break
181
182   ret
183   ;*********************************************************************
184   ; push_clear
185   ;*********************************************************************
186
187   push_clear:
188        ldi r16, PORT_INT0_bm   ;clear IRQ flag for PE0
189        sts PORTE_INTFLAGS, r16
190        ret
191
192   ;*********************************************************************
193   ; error loop
194   ;*********************************************************************
195   line2_testmessage: .db 1, "ERROR, press CLEAR", 0  ; message for line #1.
196
197   error_loop:
198        ldi  ZH, high(line2_testmessage<<1)  ; pointer to line 1 memory buffer
199        ldi  ZL, low(line2_testmessage<<1)   ;
200        rcall load_msg          ; load message into buffer(s).
201        rcall update_lcd_dog
202
203        ldi r16, PORT_INT0_bm    ;clear IRQ flag for PE0
204         sts PORTE_INTFLAGS, r16
205        ret
206
207   ;*********************************************************************
208   ; push enter
209   ;*********************************************************************
210   addition_100th:
211        dec r17
212        ldi r16, 100
213        mul r18, r16 // multiply by 100 for the 100th place value
214        add r19, r0 // and then add the next digit on 1st
215        adiw ZH:ZL, $0001
216   rjmp lookup2
217
218   addition_10th:
219        dec r17
220        ldi r16, 10 // to multiply ; shift to the left on 10th
221        mul r18, r16    //shift to the left on 10th
222        add r19, r0
223        adiw ZH:ZL, $0001
224   rjmp lookup2
225
226
227   push_enter:
228
229        ldi r17, 3
230        ldi r18, 0x00
231        ldi r19, 0x00
232        ldi ZH, high (dsp_buff_1+12) ; Load ZH and ZL as a pointer to 1st
233        ldi ZL, low (dsp_buff_1+12)  ; byte of buffer for line 1.
234
235        ldi r16, PORT_INT0_bm    ;clear IRQ flag for PE0
236        sts PORTE_INTFLAGS, r16
```

```
237
238        pop r16           ;restore SREG then r16
239        out CPU_SREG, r16    ;note I in SREG now = 0
240        pop r16           ;restore SREG then r16
241        sei               ;SREG I = 1
242
243    lookup2:
244        ld r18, Z
245        andi r18, 0x0F  // mask to translate from ascii code to numerical value
246
247        cpi r17, 3
248        breq addition_100th
249
250        cpi r17, 2
251        breq addition_10th
252
253        // 1th addition
254        add r19, r18
255
256
257
258        cpi r19, 101    // check if the value is over 100
259        brge error_loop // branch if it is equal or greater than 101
260
261
262        ;Load low byte then high byte of PER period register
263        ldi r16, LOW(PERIOD_EXAMPLE_VALUE)        ;set the period
264        sts TCA0_SINGLE_PER, r16
265        ldi r16, HIGH(PERIOD_EXAMPLE_VALUE)
266        sts TCA0_SINGLE_PER + 1, r16
267
268        ;Load low byte and the high byte of CMP0 compare register
269        //ldi r16, LOW(DUTY_CYCLE_EXAMPLE_VALUE)        ;set the duty cycle
270        mov r16, r19
271        sts TCA0_SINGLE_CMP0, r16    ;use TCA0_SINGLE_CMP0BUF for double buffering
272        @ ldi r16, HIGH(DUTY_CYCLE_EXAMPLE_VALUE)
273        @ sts TCA0_SINGLE_CMP0 + 1, r16
274
275        // Set clock and start timer/counter TCA0
276        ldi r16, TCA_SINGLE_CLKSEL_DIV64_gc | TCA_SINGLE_ENABLE_bm
277        sts TCA0_SINGLE_CTRLA, r16
278
279    ret
280        // now convert the percentage value into value out of 255, and generate pulse
281    /*
282        cpi r19, 100
283        breq brightness_full
284
285        cpi r19, 0
286        breq brightness_zero
287
288
289        mov r20, r19
290        lsr r20
291
292        lsl r19
293
294        add r19, r20
295
296        ldi r20, 255
```

```
297        sub r20, r19
298
299    ;*********************************************************************
300    ; execute
301    ;*********************************************************************
302    execute:
303    timing_loop:
304    mov r16, r19      // move it to r16 r19 dont change
305    mov r18, r20      // r20 dont change
306
307        loop:
308            sbi VPORTD_OUT, 0
309
310        dec_loop:
311            dec r16
312            brne loop
313
314        loop2:
315            cbi VPORTD_OUT, 0
316
317        dec_loop2:
318            dec r18
319            brne loop2
320
321        rjmp timing_loop
322    */
323    ;*********************************************************************
324    ; shift_by_1
325    ;*********************************************************************
326
327    shift_by_1:
328        ldi ZH, high (dsp_buff_1+15) ; Load ZH and ZL as a pointer to 1st
329        ldi ZL, low (dsp_buff_1+15)  ; byte of buffer for line 1.
330        ldi r20, 0x20    //r20 is blank
331
332        sbiw ZH:ZL, $0002
333        ld r19, Z
334
335        sbiw ZH:ZL, $0001
336        st Z, r19
337
338        adiw ZH:ZL, $0002
339        ld r19, Z
340
341        sbiw ZH:ZL, $0001
342        st Z, r19
343
344        adiw ZH:ZL, $0002
345        ld r19, Z
346
347        sbiw ZH:ZL, $0001
348        st Z, r18
349
350        adiw ZH:ZL, $0001
351        st Z, r20
352
353        ret
354
355    /*
356    ;*********************************************************************
```

```asm
357  ; brightness full (100%)
358  ;*******************************************************************
359  brightness_full:
360      sbi VPORTD_OUT, 0
361      rjmp brightness_full
362
363  ;*******************************************************************
364  ; brightness zero (0%)
365  ;*******************************************************************
366  brightness_zero:
367      loop43:
368          sbi VPORTD_OUT, 0
369      rjmp loop43
370      cbi VPORTD_OUT, 0
371      rjmp brightness_zero
372
373  */
374  ;*******************************************************************
375  ;    clear line 1
376  ;*******************************************************************
377
378  line1_testmessage: .db 1, "Setting 1 : 000 ", 0  ; message for line #1.
379
380  clear_line:
381          ;load_line_1 into dbuff1:
382      ldi  ZH, high(line1_testmessage<<1)  ; pointer to line 1 memory buffer
383      ldi  ZL, low(line1_testmessage<<1)   ;
384      rcall load_msg             ; load message into buffer(s).
385
386      ret
387
388  ;******************
389  ;NAME:       load_msg
390  ;FUNCTION:  Loads a predefined string msg into a specified diplay
391  ;           buffer.
392  ;ASSUMES:   Z = offset of message to be loaded. Msg format is
393  ;           defined below.
394  ;RETURNS:   nothing.
395  ;MODIFIES:  r16, Y, Z
396  ;CALLS:     nothing
397  ;CALLED BY:
398  ;*******************************************************************
399  ; Message structure:
400  ;    label:  .db <buff num>, <text string/message>, <end of string>
401  ;
402  ; Message examples (also see Messages at the end of this file/module):
403  ;    msg_1: .db 1,"First Message ", 0   ; loads msg into buff 1, eom=0
404  ;    msg_2: .db 1,"Another message ", 0 ; loads msg into buff 1, eom=0
405  ;
406  ; Notes:
407  ;    a) The 1st number indicates which buffer to load (either 1, 2, or 3).
408  ;    b) The last number (zero) is an 'end of string' indicator.
409  ;    c) Y = ptr to disp_buffer
410  ;       Z = ptr to message (passed to subroutine)
411  ;*******************************************************************
412  load_msg:
413      ldi YH, high (dsp_buff_1) ; Load YH and YL as a pointer to 1st
414      ldi YL, low (dsp_buff_1)  ; byte of dsp_buff_1 (Note - assuming
415                                ; (dsp_buff_1 for now).
416      lpm R16, Z+               ; get dsply buff number (1st byte of msg).
```

```
417        cpi r16, 1                  ; if equal to '1', ptr already setup.
418        breq get_msg_byte           ; jump and start message load.
419        adiw YH:YL, 16              ; else set ptr to dsp buff 2.
420        cpi r16, 2                  ; if equal to '2', ptr now setup.
421        breq get_msg_byte           ; jump and start message load.
422        adiw YH:YL, 16              ; else set ptr to dsp buff 2.
423
424    get_msg_byte:
425        lpm R16, Z+                 ; get next byte of msg and see if '0'.
426        cpi R16, 0                  ; if equal to '0', end of message reached.
427        breq msg_loaded             ; jump and stop message loading operation.
428        st Y+, R16                  ; else, store next byte of msg in buffer.
429        rjmp get_msg_byte           ; jump back and continue...
430    msg_loaded:
431        ret
432
433    ;--------------------------- SUBROUTINES ----------------------------
434
435
436    ;==================================
437    .include "lcd_dog_asm_driver_avr128.inc"  ; LCD DOG init/update procedures.
438    ;==================================
439
440
441    ;************************
442    ;NAME:       clr_dsp_buffs
443    ;FUNCTION:  Initializes dsp_buffers 1, 2, and 3 with blanks (0x20)
444    ;ASSUMES:   Three CONTIGUOUS 16-byte dram based buffers named
445    ;           dsp_buff_1, dsp_buff_2, dsp_buff_3.
446    ;RETURNS:   nothing.
447    ;MODIFIES:  r25,r26, Z-ptr
448    ;CALLS:     none
449    ;CALLED BY: main application and diagnostics
450    ;****************************************************************
451    clr_dsp_buffs:
452        ldi R25, 48                 ; load total length of both buffer.
453        ldi R26, ' '                ; load blank/space into R26.
454        ldi ZH, high (dsp_buff_1)   ; Load ZH and ZL as a pointer to 1st
455        ldi ZL, low (dsp_buff_1)    ; byte of buffer for line 1.
456
457        ;set DDRAM address to 1st position of first line.
458    store_bytes:
459        st  Z+, R26         ; store ' ' into 1st/next buffer byte and
460                            ; auto inc ptr to next location.
461        dec  R25            ;
462        brne store_bytes  ; cont until r25=0, all bytes written.
463        ret
464
465
466
467    ;****************************************************************
468
469
470    ;***** END OF FILE *****
471
```