Dongyun Lee

ID: 112794190

Lab08: LCD, Keypad, and Memory Reference Instructions

ESE280-L03

Bench #3

1. Briefly describe how you converted the hex input to ASCII for your program in Design Task 3.

I used the table to convert the hext input to ASCII, so when every the key is pressed, the program looks for a certain ASCII value in the table and load it in the buffer

2. What addressing mode is used to store blank spaces in the display buffers in the subroutine `clr_dsp_buffs`? How is the value for a blank space represented in the assembly language code? What is the actual value, in hexadecimal, that is written into the display buffer to represent a blank space?

The ASCII value for blank is 0x20.

3. In Task 3, if you clear the flip-flop as soon as you detect that it has been set, before displaying the character on the LCD, and the pushbutton bounces when pressed, what might happen that would make your program operate incorrectly?

If I clear the flip flop as soon as I detect that it has been set, then the program might take the same value (from the key) two times even though when it is pressed one time.

4. In Task 3, if you clear the flip-flop after displaying the character on the LCD, and the pushbutton is pressed for a long time and bounces when released, what might happen that would make your program operate incorrectly?

The next time you press another key, it might update display with one more extra character which has been loaded into the buffer due to bounce from the previous button pushed.

5. In Task 3, if do not clear the flip-flop after displaying the character on the LCD, what would
happen that would make your program operate incorrectly?

If you do not clear the flip flop at all, then the program will not detect any next key pressed value.

6. Since you are not using the PRE input of the 74HC74 in this design, what, if anything, should
be done with it.

I assume that I should preset the flipflop after input detect in order to debounce better.

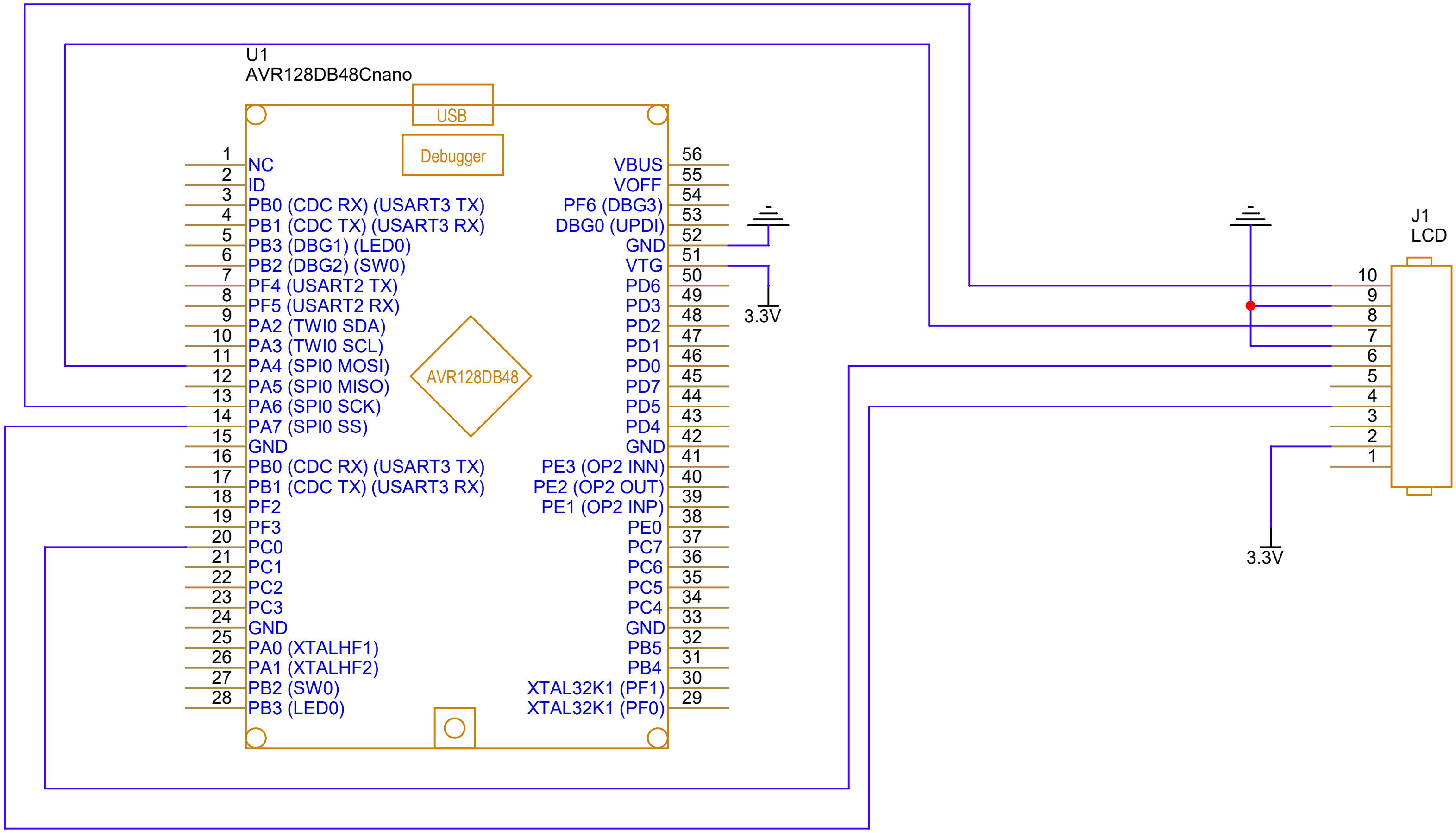
```
;
; dog_lcd_test_avr128.asm
;
; Created: 10/9/2023 2:14:29 PM
; Author : kshort
;


;*****************************************************************
;**********        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
    rcall clear_display
// display setting line

    rcall update_lcd_dog


    // keypad subroutine
    ldi r19, 3 // digit full
    ldi r18, 0x00
    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

    rcall output

    rcall update_lcd_dog

    rjmp check_press

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


    ; press -> convert to ascii -> display (do not shift)-> press -> shift to the left
    ; (but only have to shift the digits not the whole line)
    ; -> every time we press, have to check if that press is enter or clear
    ; -> check if 3 digits are full for that line, go in to a loop only looking for clear or enter
    ; -> when you press enter, check if the value on the display is over 100 or not




;---------------------------- SUBROUTINES ----------------------------

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

dec r19 // is digit full
```

```

// lookup table from lecture
lookup:
    ldi r16, 0x00
    ldi ZH, high (table*2)
    ldi ZL, low (table*2)
    ldi r16, $00
    add ZL, r18
    adc ZH, r16
    lpm r18, Z

    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
    st X, r18  // storing into SRAM buffer



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

clear_flipflop:     // clear the flip flop for next input
    cbi VPORTB_OUT, 4
    sbi VPORTB_OUT, 4

    rcall update_lcd_dog

cpi r19, 0x00   // if digit is full
    breq is_digit_full

rjmp check_press    // go back to the start


;*********************************************************************
; push_clear
;*********************************************************************

push_clear:
```

```
ldi r19, 0x03

rcall clear_display
rjmp delay_break


;********************************************************************
; push_enter
;********************************************************************

push_enter:
ldi r19, 0x03
inc r18
    // check if the value is over 100
    // if not
    ldi XH, high (dsp_buff_1+r18*16)
    ldi XL, low (dsp_buff_1+r18*16)

rjmp delay_break


;********************************************************************
; reset pointer
;********************************************************************
reset_pointer:
    ldi r20, 47

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

    rjmp output

;********************************************************************
; is digit full
;********************************************************************

is_digit_full:
    wait_for_clear_or_enter_loop:   // in a loop that only wait for clear or enter
        sbis VPORTB_IN, 5
        rjmp wait_for_clear_or_enter_loop

            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

            ldi r16, 0x00
            ldi ZH, high (table*2)
            ldi ZL, low (table*2)
            ldi r16, $00
            add ZL, r18
            adc ZH, r16
            lpm r18, Z

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

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

            rjmp wait_for_clear_or_enter_loop


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



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



line1_testmessage: .db 1, "Setting 1  :    ", 0  ; message for line #1.
line2_testmessage: .db 2, "Setting 2  :    ", 0  ; message for line #2.
```

```
line3_testmessage: .db 3, "Setting 3  :    ", 0  ; message for line #3.





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

    sbi VPORTB_DIR, 4    // clear flip flop output
    cbi VPORTB_OUT, 4
    sbi VPORTB_OUT, 4   // clear = 1

    ; keypad input
    cbi VPORTC_DIR, 7
    cbi VPORTC_DIR, 6
    cbi VPORTC_DIR, 5
    cbi VPORTC_DIR, 4

    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

clear_display:
    ;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).

   ldi  ZH, high(line2_testmessage<<1)  ; pointer to line 2 memory buffer
   ldi  ZL, low(line2_testmessage<<1)   ;
   rcall load_msg          ; load message into buffer(s).

   ldi  ZH, high(line3_testmessage<<1)  ; pointer to line 3 memory buffer
   ldi  ZL, low(line3_testmessage<<1)   ;
   rcall load_msg          ; load message into buffer(s).

    ;breakpoint followin instr. to see blanked LCD and messages in buffer
   rcall update_lcd_dog     ;breakpoint here to see blanked LCD

   ret

;*******************************************************************
; shift_by_1
```

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

shift_by_1:
    ldi ZH, high (dsp_buff_1+13)
    ldi ZL, low (dsp_buff_1+13)
    ldi r20, 0x20   //r16 is zero 0


    ld r17, Z

            sbiw ZH:ZL, $0001   ; decrement the pointer

            st Z+, r17
            //adiw XH:XL, $0001

            st Z, r20


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

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


;====================================
.include "lcd_dog_asm_driver_avr128.inc"   ; LCD DOG init/update procedures.
;====================================


```



Title	Prelab7 Schematic Dongyun Lee		
Size A	Document Number Dongyun Lee		Rev
Date:	Thursday, October 19, 2023	Sheet 1 of 1	