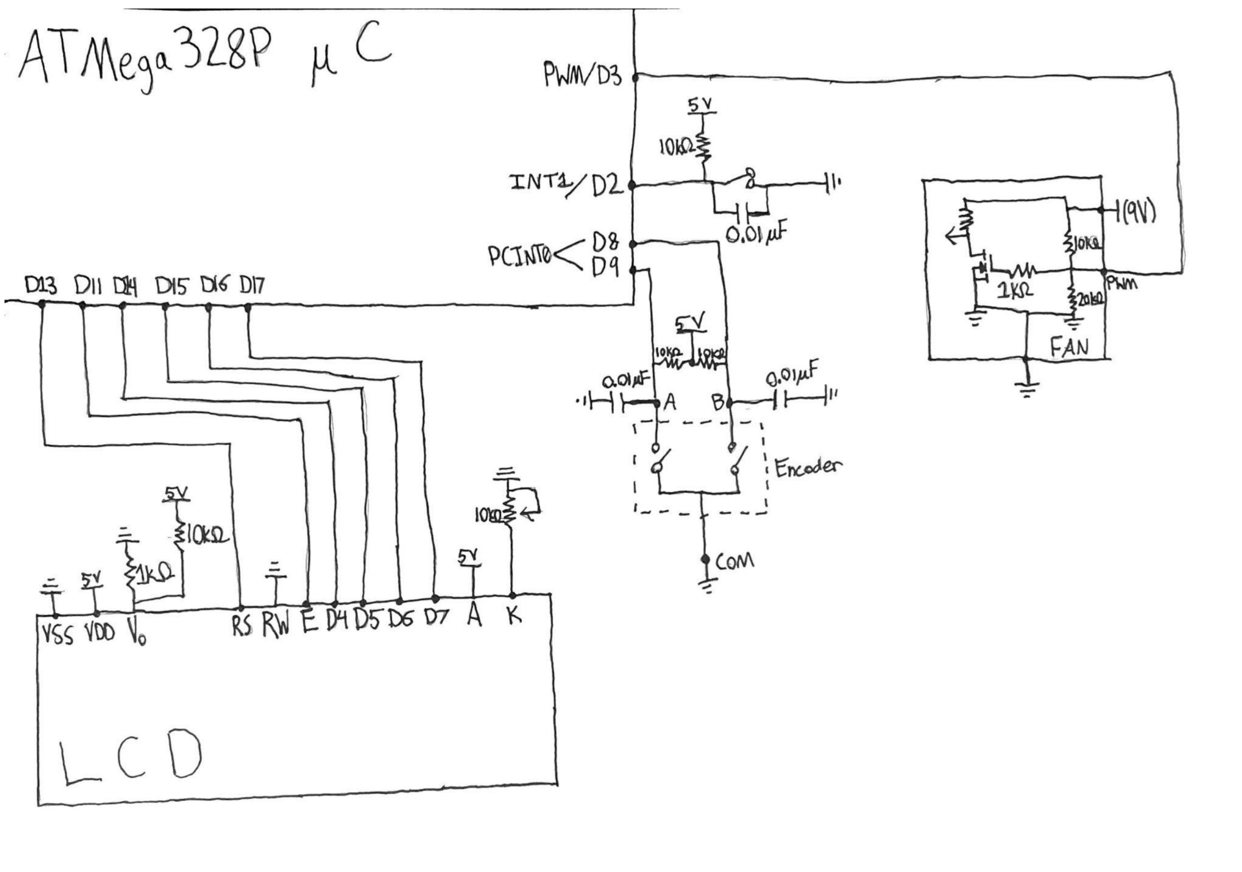
Authors: Max Finch, Tiger Slowinski
Team Members: Max Finch, Tiger Slowinski
ECE:3360 Embedded Systems
Post-Lab Report 4

1. Introduction

The goal of this lab is to create a variable speed fan with an interactive LCD display. The display shows the current fan speed in integer percent values from 1% to 100%, with an initial speed of 50%. A Rotary Pulse Generator (RPG) changes the fan speed and displayed value with a CW and CCW turn incrementing and decrementing the speed by 1%, respectively. A pushbutton switch (PBS) is used to turn the fan on and off, with the display's second line appropriately displaying "ON" or "OFF".

2. Schematic

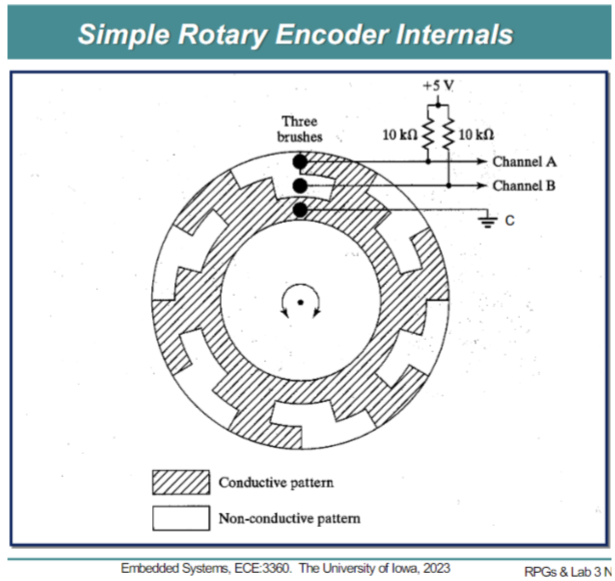


ATMega328P μC
PWM/D3
5V
10kΩ
INT1/D2
0.01μF
PCINT0
D8
D9
D13 D11 D14 D15 D16 D17
5V
10kΩ 10kΩ
0.01μF
A
B
0.01μF
Encoder
COM
(9V)
10kΩ
1kΩ
20kΩ
PWM
FAN
5V
10kΩ
1kΩ
5V
10kΩ
5V
VSS VDD V0 RS RW E D4 D5 D6 D7 A K
LCD

3. Discussion

Rotary Pulse Generator (RPG) for User Input

A description of the inner workings of the RPG follows.



The RPG has a series of conductive and non-conductive ridges. To take in the physical input as a signal, the RPG uses 2 channels and one ground pin. Channels A and B act like pushbuttons, where they can either send a 1 or 0 to the Arduino if it makes contact with the conductive or non conductive ridges respectively. As Channels A and B function like pushbuttons, we must decouple the inputs with capacitors (0.01 µF), and set the button unpressed state to either Logic HIGH or Logic LOW with pull up resistors (10KΩ). Our implementation uses pull-up resistors, meaning Channels A and B are always at Logic HIGH when not pressed, and Logic LOW when pressed.

So how do we use this to detect clockwise/counterclockwise turns to determine what value to display? Channels A and B are read at the same time, which means distinct AB patterns can tell us the direction of the turn. For our implementation (due to the pull-up resistors, a clockwise turn has an AB combination of (11 >> 01>>11>>10>>11), and counter-clockwise turn (11 >> 10>>11>>01>>11). Depending on whether a left or right turn is registered, a decimal value will increment or decrement in the software which will then symbolize the new value to display (software details will be in a later section).

Push Button for User Input

As mentioned earlier, the push button can send a 1 or 0 to the Arduino. Depending on how you configured your pull up resistor, the unpressed state of the button can be a 1 or 0. The push button also needs a decoupling capacitor like the RPG to smooth out any possible unwarranted input from the button. In this lab, the button is used to toggle the Fan to be on or off..

Implementing delays with Timer0

We used TCNT0 (Timer 0), one of the two 8-bit timer registers that the ATmega328P has. TCNT0 is a register that holds the value of where to start counting from to its max value: 256. The smaller the value you put in TCNT0 the longer the delay will be. TCCRB0 is a mode control register that allows you to select a prescaler, set no clock source, and the mode in which the timer operates. In our implementation, setting TCCRB0 with 0b00000101 configures Normal Mode with bit 3 and a prescaler of 1024 with bits 0,1,2 (101).

So why do we need the prescaler and how does the timer give us the delay we want?

The prescaler essentially "slows down" the timer, which normally operates at 16MHz. First we find the period of the clock (T_clock).

$$T\_clock = \left(\frac{1}{\frac{16\ MHz}{prescaler}}\right) = \left(\frac{1}{\frac{16MHz}{1024}}\right) = 6.4 \times 10^{-5}$$

Dividing our desired delay by the clock period gives us the amount of clocks needed to simulate our delay. Our implementation uses a delay of 10ms, so the calculation is as follows:

$$\frac{0.01s}{6.4 \times 10^{-5}} = 156.25 \approx 156 \text{ clocks}$$

Finally, we must find the value of where to start "counting" to make a 10ms delay.

$$startClock = \#MaxClocks - \#Clocks\ Needed = 256 - 156 = 100$$

100 in hexadecimal is 0x64, and this is the value that can be seen being loaded into the numClocks variable, which is loaded into TCNT0 in the source code. NumClocks is loaded with different values to generate different lengths of delays.

How does the clock know when to stop counting?
Register TIFR0 holds the flag bits for all timers on the ATmega328P. We are concerned with bit TOV0, as once this bit is set, that means the timer has finished counting. We check for this change in TOV0 in our code, stop/reset the timer and exit the subroutine.

Implementing PWM with Timer2

Pulse Width Modulation is used for this lab in order to create analog signals from periodic digital pulses, which resemble a square wave with logic high (5V) at the peak, and logic low (0V) at the valley. By keeping the frequency of the pulses constant and modulating the duty cycle (the percent of time the signal is at logic high), the "perceived" voltage can likewise be modulated between 0V and 5V, with duty cycles between 0% and 100%, respectively.

PWM can be implemented using the ATMega328P's timers in a variety of ways using the timer's Timer/Counter Control Registers (in our case TCCR2A and TCCR2B), and setting the Waveform Generation Mode (WGM) bits accordingly. Our lab uses "Fast PWM to OCR2A". This mode is implemented by setting values between 0 and 255 to Output Compare Registers A and B (OCR2A and OCR2B). The Timer Counter Register (TCNT2) starts incrementing with each timer clock cycle (post prescaler) with an initial value of 0, and sets the designated PWM pin on the ATMega328P (in our case D3) to high. Once the value of TCNT2 matches OCR2B, the output at the PWM pin is set to low. TCNT2 continues to increment up to OCR2A, and upon match sets the PWM pin back to high and restarts the counting process. By strategically setting OCR2A to 101 and making OCR2B's value variable between 1 and 101, a duty cycle of integer values between 1% and 100% can be created just by making OCR2A variable in response to CW and CCW rotation of the RPG.

Responding To User Input with Interrupts

Interrupts allow for freed processing power, responsiveness, and program efficiency at the meager cost of programmer torment. The program counter can be anywhere in the program, and when an interrupt occurs the current instruction will be completed and the program counter will go the address of the appropriate Interrupt Service Routine (ISR), with the return address loaded onto the stack for the program counter to go back to upon completion of the ISR. When an interrupt occurs either in response to a pin change or state change (rising or falling edge), the program counter goes to a defined address in memory corresponding to the type of interrupt called and where it was called from, with the addresses listed in the documentation for the microcontroller's Interrupt Vector Table (IVT). While all digital pins of the ATMega328P are associated with a PCINTx value, pin change interrupts can be neatly grouped according to what PORT the pin is a part of, whereas external interrupts have two designated pins (D2 and D3). From there, the program counter may jump to an ISR, and upon completion of the ISR the program returns to the address that was loaded to the stack.

Software Explanation

LCD Initialization
The LCD (1620A) we use in this lab has to follow a key initialization process in order for it to function as intended. For this lab we use 4-bit mode, so we must initialize the LCD in 8-bit mode first and then initialize it in 4-bit mode. After this is complete we can send data to the LCD. In our source code we call subroutines *_8bit_initialization* and *_4bit_initialization* on lines 93 and 95 respectively.

Both 8-bit and 4-bit setups issue commands to the LCD. These commands will be explained in the next section, but know that commands need a certain amount of time to execute and delays (at max 1.64ms) are needed between them.

When in 4-bit mode, we must send the 8-bit commands in pairs of two 4-bit commands with a delay in-between them.

LCD Commands/Data Operations
The LCD is able to accept input as **commands or data**. We toggle the type of input we want with the R/S pin, with (R/S) = 1 being data mode and (R/S) = 0 being command mode. In order to send either a command or data, the enable pin must be strobed.

Command Example: Lines 439-446 in the source code set the LCD to accept 4-bits and display two lines. Strobe is enabled after the high and low nibbles are sent to PORTC, where pins D4-D7 are connected. There is also an adequate delay of 200 microseconds between nibbles. We can see that on line 436 R/S is set low.

Data Example: Lines 330-333 and 382-397 in the source code send the “%” symbol to the LCD. The symbol “%” is stored as a hexadecimal number, so we need to store the LOW bits in R30 and HIGH bits in R31 (R30:R31 is the Z register). In displayCString, we see on line 382 that R/S is set to high.

Below is a table of the possible commands that can be sent to the LCD.

Command	Binary D7	D6	D5	D4	D3	D2	D1	D0	Hex
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	I/D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D/C	R/L	x	x	10 to 1F
Function Set	0	0	1	8/4	2/1	10/7	x	x	20 to 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	1	A	A	A	A	A	A	A	80 to FF

I/D: 1=Increment*, 0=Decrement
S: 1=Display shift on, 0=Display shift off*
D: 1=Display On, 0=Display Off*
U: 1=Cursor underline on, 0=Underline off*
B: 1=Cursor blink on, 0=Cursor blink off*
D/C: 1=Display shift, 0=Cursor move
R/L: 1=Right shift, 0=Left shift
8/4: 1=8 bit interface*, 0=4 bit interface
2/1: 1=2 line mode, 0=1 line mode*
10/7: 1=5x10 dot format, 0=5x7 dot format*
x = Don't care
* = Initialisation settings

Encoder Pin Duty Cycle Inc/Dec with Pin Change Interrupt (PCINT0, 1)

The ISR (Interrupt Service Routine) labeled *rpg_change* on line 18 is called when PB0 or PB1 go high or low (More details in the hardware section). This ISR checks the current turn pattern captured by R18 and determines whether the RPG has completed a full CW or CCW turn. If so, the CW and CCW subroutines will increase the PWM or decrease the PWM respectively. The numerical value displayed on the LCD will also change. R16 in CW and CCW let the display routines know when increase the digit length for the duty cycle. Reti is used to return to where the program counter was before the ISR was called.

Button Press To Turn Fan ON/OFF with External Interrupt (INT1)

The ISR labeled button_p on line 16 is called when PD2 enters a falling edge (1->0). This ISR checks whether the fan was on or off before it was called with R28, and activates the opposite (If 1 become 0, if 0 become 1.). It will also change the text displayed to the LCD with “ON” or “OFF”

Lookup table for DC percents

The lookup table found at the beginning of the code uses the Z pointer to extract string representations of the current PWM value. The PWM value and string representation are NOT linked. The string rep. And PWM both start at “50” and increment and decrement together. The same process explained in the command/data section of this report is used for this lookup table. R16 is simply incremented to get the next table value. (More details in source code lines (152-155)

4. Conclusion

This lab allows for developing an intuitive understanding of how different aspects of a microcontrollers architecture interact with each other, namely in the form of timer PWM and peripherals triggering hardware interrupts. Furthermore, the use of an LCD display makes for a more complete project.

5. Appendix A: Source Code

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Assembly Language file for lab 4 in ECE:3360
; Spring 2023, The University of Iowa
; Authors : Max Finch, Tiger Slowinski
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.include "m328Pdef.inc"

.cseg
.org 0x0000

; We don't want the following lines to run. They should just reserve bytes in program memory
rjmp skip

.org 0x0002     ; Interrupt INT1
    jmp button_p
.org 0x0006     ; Interrupt PCINT0
    jmp rpg_change
.org 0x0008


; indexing starts at 4 because "1" is located at 0x0004, ik it sucks ; starts at memory loc 8 ends at 107
num:   .DB "1",0, "2",0, "3",0, "4",0, "5",0, "6",0, "7",0, "8",0, "9",0, "1", "0"
    .DB "1","1", "1","2", "1","3", "1","4", "1","5", "1","6", "1","7", "1","8", "1","9"
    .DB "2","0", "2","1", "2","2", "2","3", "2","4", "2","5", "2","6", "2","7", "2","8", "2","9"
    .DB "3","0", "3","1", "3","2", "3","3", "3","4", "3","5", "3","6", "3","7", "3","8", "3","9"
    .DB "4","0", "4","1", "4","2", "4","3", "4","4", "4","5", "4","6", "4","7", "4","8", "4","9"
    .DB "5","0", "5","1", "5","2", "5","3", "5","4", "5","5", "5","6", "5","7", "5","8", "5","9"
    .DB "6","0", "6","1", "6","2", "6","3", "6","4", "6","5", "6","6", "6","7", "6","8", "6","9"
    .DB "7","0", "7","1", "7","2", "7","3", "7","4", "7","5", "7","6", "7","7", "7","8", "7","9"
    .DB "8","0", "8","1", "8","2", "8","3", "8","4", "8","5", "8","6", "8","7", "8","8", "8","9"
    .DB "9","0", "9","1", "9","2", "9","3", "9","4", "9","5", "9","6", "9","7", "9","8", "9","9"
    .DB "1","0","0", 0

; Strings to be displayed on LCD
msg: .DB " DC = "
msgg: .DB " Fan: "
off: .DB "OFF "
on: .DB "ON  "
perc: .DB "% "
      .DW 0

skip:
; Grounded pins: PIN_5: R/W (Read/Write), PIN_1: GND (Ground), Vee (1K resistor)
; Powered pins: PIN_2: Vcc (Supply), Vee (10K resistor)

ldi R16, 0xFF    ; Load 0xFF (binary 11111111) into R16
out DDRC, R16    ; Set all PORTC pins as outputs
                 ; PC0 is D4, PC1 is D5, PC2 is D6, PC3 is D7
sbi   DDRB,5     ; PB5 is R/S
sbi   DDRB,3     ; PB3 is Enable
cbi   DDRB,0     ; PB0 is (Rotary A)
cbi   DDRB,1     ; PB1 is (Rotary A)
cbi   DDRD,2     ; PD2 is pushbutton

.def numClocks = R21   ; sets the clock value for delay
.def setup_hex = R22   ; numClock values: 0x64 -> 10ms, 0xB2 -> 5ms, 0xFD -> 200us, for 100ms, do 10ms 10 times
```

```
PWM_init:
    ldi r17, (1<<COM2B1)|(1<<WGM21)|(1<<WGM20);
    sts TCCR2A,r17 ; 8 bit PWM non-inverted
    ldi r17,(1<<CS20) | (1<<WGM22)
    sts TCCR2B,r17 ; Timer clock = I/O clock
    ldi r17,50 ;50% duty cycle        3F-> 50% duty cycle
    sts OCR2B,r17 ; Set compare value/duty cycle ratio
    sbi DDRD, PORTD3 ; Set OC2B pin as output
    ldi r17,101
    sts OCR2A,r17

; Set pinchange interrupt for RPG
lds r16, PCICR
ori r16, 0b00000101
sts PCICR, r16
lds r16, PCMSK0
ori r16, 0b00000011
sts PCMSK0, r16

;Set Ext Int for Pushbutton
lds r16, EICRA
ori r16, 0b00000010
sts EICRA, r16
in r16, EIMSK
ori r16, 0b00000001
out EIMSK, r16

; This subroutine initalizes the LCD to 4-bit mode and displays the default boot
; strings.
; DC = 1  %
; FAN = ON
start_seq:

    rcall _8bit_initialization

    rcall _4bit_initialization
    cbi PORTB, 3   ;enable
    ldi R16, 4 ; start at 1%, memory starts at  4 x (offset=2) = 0x0008
    ldi R26, 1 ; length of string for variable string
    ldi r28, 1 ; R28=1 The FAN is ON, R28=0 the FAN is OFF


    rcall update_dis

    ;moves cursor to second line and displays "on"
    rcall status_cursor
    ldi R30, LOW(2*on)
    ldi R31, HIGH(2*on)
    ldi R24, 2           ; Length of the string
    rcall displayCString
    rcall delay_10ms


sei
```

```
;This code starts the fan at 50% DC for the DISPLAY. The actual DC is set
;to 50 in PWM_init. We loop 54 times because R16 is offset by 4, and must
;correct it.
_54_times:
ldi r23, 54
loop:
cpi r23, 1
breq main
dec r23
inc R16
rcall update_DC
rjmp loop

; main simply waits for an interrupt to be invoked
main:
    nop
    nop
    nop
    rjmp main

;Moves cursor to where percent sign should be
perc_cursor:
    cbi PORTB, 5        ; making RS -> 0 for commands
    cbi PORTB, 3        ; Enable low

    ; Shift cursor to home position
    ldi R17, 0b1000
    out PORTC, R17        ; Send command to LCD (RS = 0)
    rcall strobe_enable
    rcall delay_200us       ; Wait for instruction to complete
    ; Send command again to configure the high nibble
    ldi R17, 0b1001
    out PORTC, R17
    rcall strobe_enable
    sbi PORTB, 5        ; making RS -> 1 for sending data
    ret

; Moves cursor to where the numerical value of duty cycle should be and updates it,
; We use an lsl on r16 to get the right address of the string value with the offset.
; EX: We want the character "1" which is located at 0x0008. 4 will be in r16
; and the lsl will perform a logical shift left, which is the same as multiplying by 2
update_DC:
    rcall dc_cursor

    ;updates the string representation of the number by moving the pointer

    lsl r16       ; multiply by 2
    ldi r30, LOW(num)
    ldi r31, HIGH(num)
    add r30, r16 ; add index to string start address
    adc r31, r1 ; carry to high byte
    mov R24, R26
    rcall displayCString
    lsr r16          ; divide by 2

    rcall delay_10ms

    ret
```

```
; When button is pressed we check whether the fan was previously on or off
; with R28. R27 updates the pwm with the value of (the pointer) - (offset in memory (3))
; if the fan is being turned on. If its being turned off R27 is loaded with 0, (r7)
button_p:
    cli
    mov r27, r16
    subi r27, 3
    inc r28
    sbrs r28,0
    rjmp  fan_off        ;turn off fan -> set r28 to 0
    rjmp  fan_on        ;turn on fan -> set r28 to 1

fan_on:
    sts OCR2B,r27
    rcall status_cursor

    ldi R30, LOW(2*on)
    ldi R31, HIGH(2*on)
    ldi R24, 3   ; Length of the string
    rcall displayCString

    rcall delay_10ms
    reti
fan_off:
    sts OCR2B, r7
    rcall status_cursor

    ldi R30, LOW(2*off)
    ldi R31, HIGH(2*off)
    ldi R24, 3   ; Length of the string
    rcall displayCString

    rcall delay_10ms

    reti

; Moving cursor to where on/off is displayed
status_cursor:
    cbi PORTB, 5        ; making RS -> 0 for commands
    cbi PORTB, 3        ; Enable low

    ldi R17, 0b1100
    out PORTC, R17        ; Send command to LCD (RS = 0)
    rcall strobe_enable
    rcall delay_200us       ; Wait for instruction to complete
    ; Send command again to configure the high nibble
    ldi R17, 0b0110
    out PORTC, R17
    rcall strobe_enable
    sbi PORTB, 5        ; making RS -> 1 for commands
    ret
```

```
off_f:
sts OCR2B, r7
rjmp goto_ret
on_o:
sts OCR2B,r27
rjmp goto_ret

check_fan:
mov r27, r16
subi r27, 3
cpi r28, 0
breq off_f
cpi r28, 1
breq on_o
goto_ret:
ret

; The following is the ISR (Interrupt Service Routine) for the RPG,
; which utilizes a pin change interrupt. When a pin is changed r18
; will be updated with the new values from pins A and B from the RPG
; when r18 becomes filled with a right or left turn pattern, the subroutine
; will branch to either the CW or CCW subroutines respectively. These
; routines will update the Duty Cycle LCD value and the PWM on Timer2
; by incrementing or decrementing those values

rpg_change:
; PB0 is (Rotary A)
; PB1 is (Rotary B)
; current values of r18 are checked with the left turn or right turn bit pattern
cli
lsl r18
lsl r18
sbic PINB, 0
sbr r18, 1
sbic PINB, 1
sbr r18, 2

cpi r18, 0b00011110
breq CW
cpi r18, 0b00101101
breq CCW
reti

CW:                     ; increment
    cpi R16, 12
    breq inc_num_len
    cpi R16, 103
    breq inc_num_len
    cpi R16, 104
    breq reti_s     ;breq to a reti
mark_1:
    inc R16
    rcall check_fan
    rcall update_DC
    reti
```

```
CCW:
    cpi R16, 104
    breq dec_num_len
    cpi R16, 13
    breq dec_num_len
    cpi R16, 4
    breq reti_s
mark_2:
    dec R16
    rcall check_fan
    rcall update_DC
    reti

inc_num_len:
    inc R26
    rjmp mark_1

dec_num_len:
    dec R26
    rjmp mark_2

reti_s:

reti

; Update display sets the strings that will be static throughout the
; program as well as the initial DC value
update_dis:

    ldi R30, LOW(2*msg)
    ldi R31, HIGH(2*msg)
    ldi R24, 6   ; Length of the string
    rcall displayCString

    rcall delay_10ms

    lsl r16       ; multiply by 2
    ldi r30, LOW(num)
    ldi r31, HIGH(num)
    add r30, r16 ; add index to string start address
    adc r31, r1 ; carry to high byte
    mov R24, R26
    rcall displayCString
    lsr r16          ; divide by 2

    rcall delay_10ms
    rcall perc_cursor
    rcall delay_10ms

    ldi R30, LOW(2*perc) ;  Percent sign
    ldi R31, HIGH(2*perc)
    ldi R24, 1   ; Length of the string
    rcall displayCString

    rcall delay_10ms
    rcall shift_cursor
    rcall delay_10ms
```

```
    ldi R30, LOW(2*msgg)
    ldi R31, HIGH(2*msgg)
    ldi R24, 6   ; Length of the string
    rcall displayCString
    ret

; Moves the LCD cursor to where the DC value should be
dc_cursor:
    cbi PORTB, 5        ; making RS -> 0 for commands
    cbi PORTB, 3        ; Enable low

    ; Shift cursor to home position
    ldi R17, 0b1000
    out PORTC, R17        ; Send command to LCD (RS = 0)
    rcall strobe_enable
    rcall delay_200us       ; Wait for instruction to complete
    ; Send command again to configure the high nibble
    ldi R17, 0b0110
    out PORTC, R17        ; Send command to LCD (RS = 0)
    rcall strobe_enable
    sbi PORTB, 5        ; making RS -> 1 for data
    ret

; Moves the LCD cursor to the second line (used only for initialization)
shift_cursor:
    cbi PORTB, 5        ; making RS -> 0 for commands
    cbi PORTB, 3        ; Enable low

    ldi R16, 0b1100
    out PORTC, R16        ; Send command to LCD (RS = 0)
    rcall strobe_enable
    rcall delay_200us       ; Wait for instruction to complete
    ; Send command again to configure the high nibble
    ldi R16, 0b0000
    out PORTC, R16        ; Send command to LCD (RS = 0)
    rcall strobe_enable
    sbi PORTB, 5        ; making RS -> 1 for data

    ret

; DisplayCString sends the string data to the data pins of
; the LCD. It swaps twice to get the lower 4 and high 4 bits of the
; string since the LCD is operating in 4-bit mode.
displayCString:
sbi PORTB, 5
L20:
        lpm
        swap R0
        out PORTC, R0
        rcall strobe_enable
        rcall delay_200us
        swap R0
        out PORTC, R0
        rcall strobe_enable
        rcall delay_200us
        adiw ZH:ZL,1
        dec R24
        brne L20
        ret
```

```
; This subroutine is run before _4bit_initalization as it is best
; practice (and safest) to initalize the LCD in 8-bit mode and THEN 4-bit
; mode.

_8bit_initialization:

    cbi PORTB, 5        ; making RS -> 0 for commands
    cbi PORTB, 3        ; Enable low

    rcall delay_100ms     ; 1

    ldi R16, 0x03         ; 2 and 3
    out PORTC, R16
    rcall strobe_enable
    rcall delay_5ms

    ldi R16, 0x03         ; 4 and 5
    out PORTC, R16
    rcall strobe_enable
    rcall delay_200us

    ldi R16, 0x03         ; 6 and 7
    out PORTC, R16
    rcall strobe_enable
    rcall delay_200us

    ldi R16, 0x02         ; 8 and 9
    out PORTC, R16
    rcall strobe_enable
    rcall delay_5ms

    ret

; This subroutine initalizes the LCD into 4-bit
; mode.
_4bit_initialization:

    cbi PORTB, 5        ; making RS -> 0 for commands
    cbi PORTB, 3        ; Enable low

    ; Write command "Set interface" (Write 28 hex (4-Bits, 2-lines)
    ldi R16, 0b0010
    out PORTC, R16
    rcall strobe_enable
    rcall delay_200us
    ldi R16, 0b1000
    out PORTC, R16
    rcall strobe_enable

    ; Write command "Enable Display/Cursor"(Write 08 hex (don't shift display, hide cursor))
    ldi R16, 0b0000
    out PORTC, R16
    rcall strobe_enable
    rcall delay_200us
    ldi R16, 0b1001
    out PORTC, R16
    rcall strobe_enable
```

```
    ; Write command "Clear and Home"(Write 01 hex (clear and home display))
    ldi R16, 0b0000
    out PORTC, R16
    rcall strobe_enable
    rcall delay_200us
    ldi R16, 0b0001
    out PORTC, R16
    rcall strobe_enable

    ; Write command "Set Cursor Move Direction" (Write 06 hex (move cursor right))
    ldi R16, 0b0000
    out PORTC, R16
    rcall strobe_enable
    rcall delay_200us
    ldi R16, 0b0110
    out PORTC, R16
    rcall strobe_enable

    ; After this the display is ready to accept data (Write 0C hex (turn on display))
    ldi R16, 0b0000
    out PORTC, R16
    rcall strobe_enable
    rcall delay_200us
    ldi R16, 0b1100
    out PORTC, R16
    rcall strobe_enable
ret

; Strobing enable allows for new data to enter the LCD via the displayCString subroutine
strobe_enable:
    sbi PORTB, 3          ; Enable high
    nop
    nop
    nop
    nop
    nop
    cbi PORTB, 3          ; Enable low
    rcall delay_2ms
    ret


delay_100ms:
    rcall delay_10ms
    rcall delay_10ms
    rcall delay_10ms
    rcall delay_10ms
    rcall delay_10ms
    rcall delay_10ms
    rcall delay_10ms
    rcall delay_10ms
    rcall delay_10ms
    rcall delay_10ms
    ret
```

```
delay_10ms:
    ldi numClocks, 0x64        ;100 (base 10) is loaded to counter register
    rcall delay
    ret
delay_5ms:
    ldi numClocks, 0xB2
    rcall delay
    ret
delay_200us:
    ldi numClocks, 0xFD
    rcall delay
    ret
delay_2ms:
    ldi numClocks, 0xE0
    rcall delay
    ret

; Base delay generator for the entire program. Uses timer0
delay:
    out TCNT0, numClocks
    ldi numClocks, 0b00000101  ;starts clock in normal mode, prescaler 1024
    out TCCR0B, numClocks
again:
    in numClocks, TIFR0
    sbrs numClocks, TOV0       ;skip if overflow flag is set
    rjmp again
    ldi numClocks, 0x00
    out TCCR0B, numClocks      ;stops timer
    ldi numClocks, (1<<TOV0)
    out TIFR0, numClocks       ;reset flag bit
    ret



    .exit
```

6. Appendix B: References

Atmel Corporation. *AVR Instruction Set Manual - Microchip Technology*. <https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-0856-AVR-Instruction-Set-Manual.pdf>.

Arduino. *Arduino UNO Rev3 with Long Pins*.<https://docs.arduino.cc/retired/boards/arduino-uno-rev3-with-long-pins>

LCD and Interrupt slides provided by Professor Beichel