Author: Adrian Alvarez
Team Member: Seth Bolen
Professor Beichel
ECE:3360 Embedded Systems
Post-Lab Report 4

1. Introduction

The goal of this lab was to construct a fan speed control system using pulse width modulation (PWM), an LCD, a rotary pulse generator (RPG), and a pushbutton switch (PBS). The system allows user interaction via the RPG to adjust the fan's speed (PWM duty cycle) and the PBS to toggle the fan on or off. Additionally, we used interrupts for efficient handling of user input and implemented advanced timer/counter functionality to maintain a fixed PWM frequency of 80 kHz. The LCD displays the fan's duty cycle and current status (ON, OFF) depending on system conditions.

When powered on, the system initializes the LCD, PWM timer, RPG, and PBS. The RPG adjusts the PWM duty cycle from 1% to 100% in approximately 1% increments when the fan is ON. Turning the RPG clockwise increases the duty cycle while counterclockwise rotation decreases it. Pressing the pushbutton immediately toggles the fan between ON and OFF states.

2. Schematics

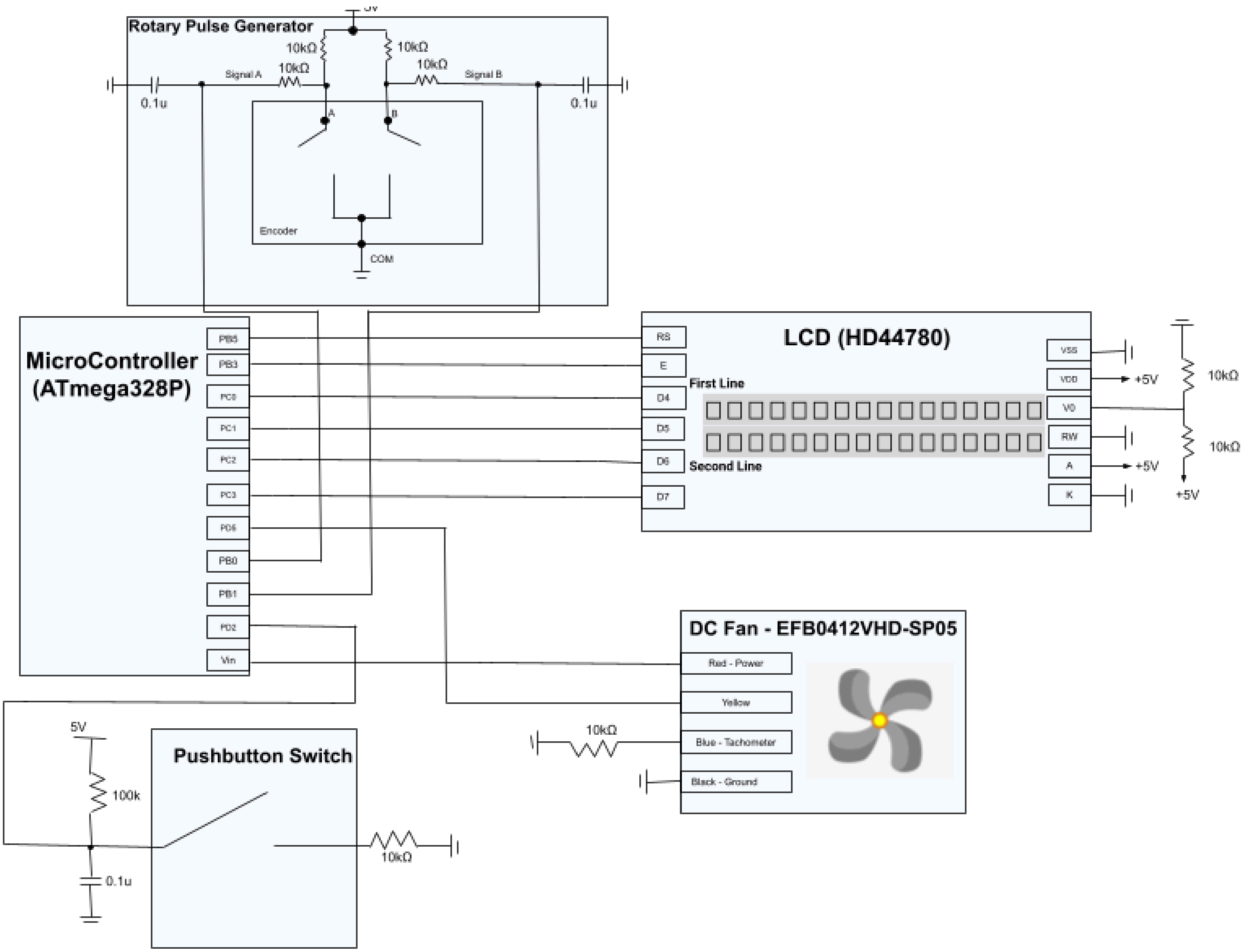


Figure 1: Implemented Circuit Schematic

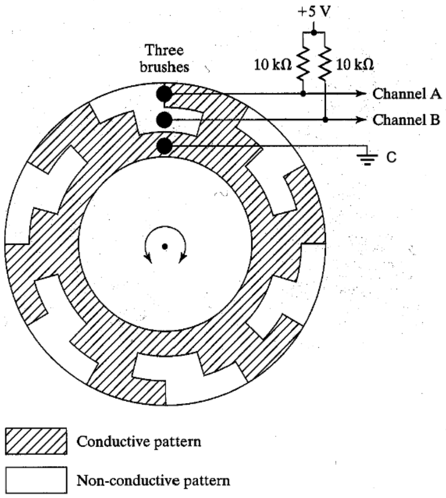


Figure 2: RPG Schematic

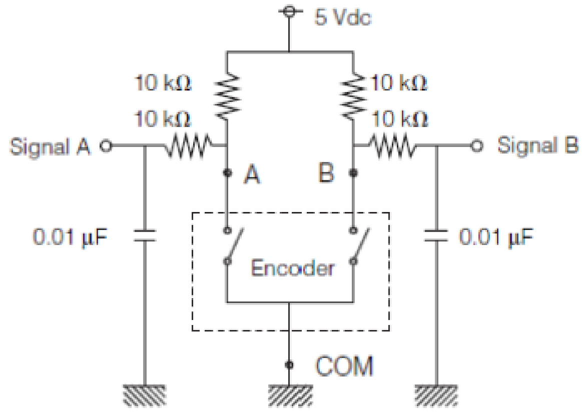


Figure 3: RPG Debounce Schematic

Pin number	Symbol	Level	I/O	Function
1	Vss	-	-	Power supply (GND)
2	Vcc	-	-	Power supply (+5V)
3	Vee	-	-	Contrast adjust
4	RS	0/1	I	0 = Instruction input 1 = Data input
5	R/W	0/1	I	0 = Write to LCD module 1 = Read from LCD module
6	E	1, 1→0	I	Enable signal
7	DB0	0/1	I/O	Data bus line 0 (LSB)
8	DB1	0/1	I/O	Data bus line 1
9	DB2	0/1	I/O	Data bus line 2
10	DB3	0/1	I/O	Data bus line 3
11	DB4	0/1	I/O	Data bus line 4
12	DB5	0/1	I/O	Data bus line 5
13	DB6	0/1	I/O	Data bus line 6
14	DB7	0/1	I/O	Data bus line 7 (MSB)

Figure 4: LCD Pinouts

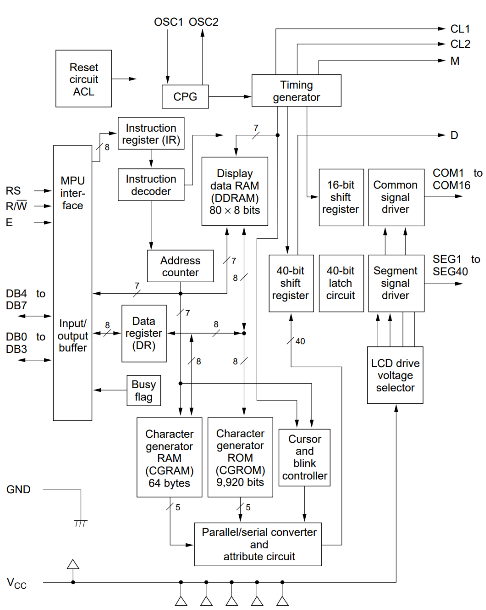
Command	Binary								Hex
	D7	D6	D5	D4	D3	D2	D1	D0	
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	I/D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D/C	R/L	x	x	10 to 1F
Function Set	0	0	1	8/4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

Figure 5: Initialization 4-bit mode



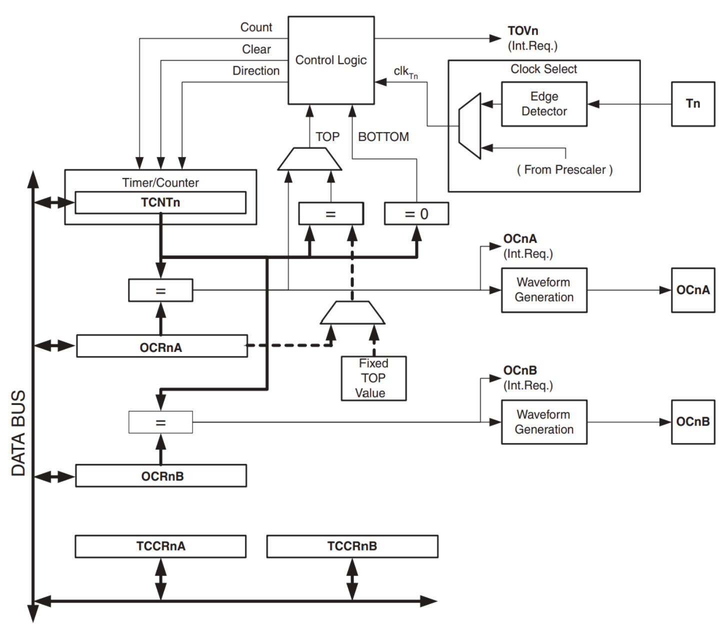


Figure 6: LCD Block Diagram

Figure 7: 8-bit Timer/Counter Block Diagram

3. Discussion

PWM Signal Generation

PWM was generated using Timer/Counter0 in Fast PWM mode with a fixed frequency of 80 kHz. We configured the timer with a prescaler of 1 and set the top value using the OCR0A register. The duty cycle was dynamically adjusted by updating the OCR0B register based on user input from the RPG. The duty cycle ranges between 1% and 100%, ensuring fine-grained control of fan speed. The PWM output was connected to the fan's control input through the OC0B pin. Appropriate limiting resistors were used where necessary to protect the fan circuitry.

Rotary Pulse Generator (RPG) Hardware and Decoding

The RPG was implemented with a hardware-based debouncing circuit, using two 10kΩ resistors to Vcc, two 10kΩ pull-down resistors, and two 0.01μF capacitors to ground for both signals A and B. RPG inputs were monitored through interrupts, detecting rotation direction by evaluating the sequence of signals A and B. A clockwise turn incremented the duty cycle, and a counter-clockwise turn decremented it. Boundary checks were implemented to ensure the duty cycle stayed within the valid range (1%–100%), and adjustments only occur when the fan is ON.

Push Button Switch (PBS) Hardware and Debouncing

The pushbutton was debounced using a 10kΩ resistor and a 0.1μF capacitor connected to ground, similar to previous labs. The PBS was configured to trigger an external interrupt on a falling edge. On press, the interrupt service routine immediately toggles the fan state between ON and OFF. If the fan is turned OFF, duty cycle adjustments via the RPG are disabled until the fan is turned back ON.

Interrupts

All interactions (PBS presses and RPG rotations) were interrupt-driven, ensuring responsive and efficient system operation. The use of interrupts allowed the system to quickly react to user input without polling, conserving processor resources for PWM generation and LCD updates.

LCD Control

The LCD displays the current duty cycle percentage and the fan status. We directly generated the appropriate strings for display updates by converting the duty cycle value to a percentage and appending the corresponding fan status. The displayed values were dynamically calculated and formatted in the program during runtime.

Examples of display messages:

- DC: 50%

  Fan:ON
- DC: 00%

  Fan:OFF

LCD updates occurred whenever the duty cycle or fan status changed.

4. Conclusion

In this lab, we gained valuable experience with advanced timer/counter features, PWM generation, external interrupts, and LCD interfacing in an embedded system. We successfully implemented a fan speed control system where the fan's speed could be adjusted through a rotary pulse generator and toggled on and off with a pushbutton switch. Interrupt-driven design ensured a highly responsive system without polling, and LCD updates kept the user informed about the system state.

Through this lab, we deepened our understanding of precise timer configurations, fast PWM generation, interrupt-driven input handling, and user interface design in assembly language, key skills for complex embedded systems development.

5. Appendix A: Source Code

```
; Lab4.asm
;
; Created: 3/24/2025 2:49:42 PM
; Author : Adrian Alvarez, Seth Bolen
;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Initialize
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.def data = r16 ; data line that includes display or instructions (write to PORTC)
.def char = r17 ; full 8 bit character which is spliced into 4 bit data during display
.def remainder = r23
.def dividend = r19
.def divisor = r25
.def lc = r18
.def ascii_const = r14
.def prev_state = r30
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Interrupts
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.org 0x0000 ; program starts here, source: RESET
        rjmp setup
.org 0x0002 ; interrupt for button, source: INT0
        rjmp button_interrupt
.org 0x0006 ; interrupt for rpg, source: PCINT0
        rjmp rpg_interrupt


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Setups everything for the program
;; Used registers: r16-r31 and r0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.org 0x0032 ; starts the program, souce: SPM_Ready
setup:
        ; intialie input pins
        cbi DDRD, 2 ; PB0 is input for button
        cbi DDRB, 0 ; A bit for rpg
        cbi DDRB, 1 ; B bit for rpg

        ; initialize output pins
        ; RS nd E
        sbi DDRB, 5 ; output
        sbi DDRB, 3 ; output
        ; Port C
        sbi DDRC, 0 ; output
        sbi DDRC, 1 ; output
        sbi DDRC, 2 ; output
        sbi DDRC, 3 ; output
        ; fan port
        sbi DDRD, 5 ; output

        ; button interrupt setup
        ldi r26, 0b00000010 ; The falling edge of INT0 generates an interrupt request.
        sts EICRA, r26 ; External Interrupt Control Register A
        ldi r26, 0b00000001 ; enable external interrupt 0
        out EIMSK, r26 ; External Interrupt Mask Register

        ; rpg interrupt setup
        lds r26, PCICR ; grab the Pin Change Interrupt Control Register
        ori r26, 0b00000001 ; enable Pin Change Interrupt Enable 0
        sts PCICR, r26 ; update Pin Change Interrupt Control Register
        lds r26, PCMSK0 ; grab the Pin Change Mask Register 0
        ori r26, 0b00000011 ; enable Pin Change Enable Mask 1 and 0
        sts PCMSK0, r26 ; grab the Pin Change Mask Register 0

        ; initialie fan constant, fan boolean, and ascii_const
        ldi r26, 0x30
        mov ascii_const, r26

        ; PWM for fan
        ldi r26, 0b00100011 ; configure Timer0 for Fast PWM mode on OCR0B (COM0B1:0 = 10)
                            ; WGM01:0 = 11 for Fast PWM, output on OC0B toggles on compare match
        out TCCR0A, r26 ; write configuration to Timer/Counter Control Register A
        ldi r26, 0b00001001 ; set Timer0 clock prescaler to clk/1 (CS00 = 1)
                                                 ; WGM02 = 1 enables Fast PWM with OCR0A as TOP
        out TCCR0B, r26 ; write configuration to Timer/Counter Control Register B
```

```
        ldi r26, 199 ; set OCR0A = 199, so TOP = 199 for Fast PWM (200 steps total)
        out OCR0A, r26 ; this sets the PWM period
        ldi r21, 100 ; set OCR0B = 100, which gives a 100/200 = 50% duty cycle
        out OCR0B, r21 ; output Compare Register B sets PWM duty cycle on OC0B (fan speed)

        sei ; enable interrupts

        rcall LCD_initialize ; call initialize LCD

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Start Program
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
start:
        ldi r20, 1 ; starts the fan as on

        ; display "DC: XX%"
        ldi data, 0b00000010 ; cursor home
        rcall send_instruction ; send instruction to LCD

        rcall first_line ; move cursor to beggining of first line
        ldi char, 0x44 ; 'D'
        rcall display_char
        ldi char, 0x43 ; 'C'
        rcall display_char
        ldi char, 0x3A ; ':'
        rcall display_char

        ldi data,0b10000111 ; move cursor to location where '%' will be printed
        rcall send_instruction

        ldi char, 0x25 ; '%'
        rcall display_char

        ; display "Fan:" on second line
        rcall second_line

        ldi char, 0x46 ; 'F'
        rcall display_char
        ldi char, 0x61 ; 'a'
        rcall display_char
        ldi char, 0x6E ; 'n'
        rcall display_char
        ldi char, 0x3A ; ':'
        rcall display_char

        rcall dc_location ; position cursor for duty cycle numeric value display

        ; print initial fan status YES
        rcall fan_location
        ldi char, 0x59 ; 'Y'
        rcall display_char
        ldi char, 0x65 ; 'E'
        rcall display_char
        ldi char, 0x73 ; 'S'
        rcall display_char

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Main
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
main:
        rcall dc_location ;place cursor on the duty cycle value

        mov r0, r26 ; store temp into 0
        ldi r26, 0
        cpse r20, r26 ; check if fan should be on or off. skips next if not equal (equal = skip = off)
        rjmp fanIsOn
        mov r26, r0 ;store back

        ;if we are here, fan_boolean equals fan_constant, meaning the fan is off I think
        ;hard code 00s
        rcall dc_location
        ldi char, 0x30 ; load ASCII '0'
        rcall display_char ; display upper digit (0)

        ldi char, 0x30 ; load ASCII '0' again
        rcall display_char ; display lower digit (0)

        rjmp main

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Determines if fan is on
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
fanIsOn:
        mov r26, r0 //store back
        mov r26, r0
        out OCR0B,r21
        mov r26, r21
        lsr r21

        ;divide r21 by 10 to separate digits
        ldi divisor, 10
        mov dividend, r21
        rcall divide

        ;display upper digit
        mov char, dividend
        add char, ascii_const
        rcall display_char

        ;display lower digit
        mov char, remainder
        add char, ascii_const
        rcall display_char
        mov r21, r26
        mov r26, r0
        mov r0, r26

        rjmp main

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;; 8-bit division algorithm, the quotient is stored in dividend and the remainder is stored in remainder
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
divide:
        ldi remainder, 0x00 ;clear remainder and carry flag
        clc
        ldi lc, 9
```

```
d1:
        rol dividend ;left shift dividend
        dec lc
        brne d2 ;if carry set, division is complete
        ret
d2:
        rol remainder ;shift dividend into remainder
        sub remainder, divisor
        brcc d3 ;if result is negative, restore remainder
        add remainder, divisor
        clc
        rjmp d1
d3:
        sec
        rjmp d1

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Display Character
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
display_char:
        sbi PORTB,5 ; RS = 1
        rcall delay_500us

        sbi PORTB, 3 ; set E to high

        mov data, char ; copy character value into data register for manipulation

        swap data ; swap high nibbles and low nibbles
        out PORTC, data ; send upper nibble
        rcall LCD_Strobe ; pulse enable to latch high nibble

        sbi PORTB, 3 ; set E to high

        swap data ; swap again to bring original low nibble to lower 4 bits
        out PORTC, data ;send lower nibble
        rcall LCD_Strobe ; pulse Enable to latch low nibble
        ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Initialize LCD
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
LCD_initialize:
        ; 8-bit mode
        rcall delay_100ms ; wait 100ms for LCD to power up
        rcall set_8bit_mode ; set device to 8-bit mode
        rcall delay_5ms ; wait 5 ms
        rcall set_8bit_mode ; set device to 8-bit mode
        rcall delay_500us ; wait at least 200us
        rcall set_8bit_mode ; set device to 8-bit mode
        rcall delay_500us ; wait at least 200us
        rcall set_4bit_mode ; set device to 4-bit mode
        rcall delay_5ms ; wait at least 5ms

        ; 4-bit mode
        rcall set_interface
        rcall delay_5ms ; wait at least 5ms
        rcall enable_display
        rcall delay_5ms ; wait at least 5ms
        rcall clear_and_home
```

```
        rcall delay_5ms ; wait at least 5ms
        rcall set_cursor_direction
        rcall delay_5ms ; wait at least 5ms

        ; turn on display
        ldi data, 0x00
        out PORTC, data
        rcall LCD_Strobe
        ldi data, 0x0C
        out PORTC, data
        rcall LCD_Strobe
        rcall delay_5ms ; wait at least 5ms
        ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Strobe Enable
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
LCD_strobe:
        sbi PORTB, 3 ; stes E to 1
        rcall delay_250ns ; dealay 250ns
        cbi PORTB, 3 ; sets E to 0
        ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Instructions for 8-bit
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
set_8bit_mode:
        cbi PORTB, 5 ; RS = 0
        ldi data, 0x03 ; 3 hex
        out PORTC, data ; writes out to PORTC (DB4-7)
        rcall LCD_strobe ; strobe enable
        ret

set_4bit_mode:
        ldi data, 0x02 ; 2 hex
        out PORTC, data ; writes out to PORTC (DB4-7)
        rcall LCD_strobe ; strobe enable
        ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Instructions for 4-bit
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

send_instruction: ;general instruction, specific instruction subroutines call this to send data
        cbi PORTB,5 ; RS = 0
        rcall delay_500us ; wait for LCS to be ready

        ; send high nibble
        sbi PORTB, 3
        swap data ; move high nibble to low nibble
        out PORTC, data ; send upper nibble
        rcall LCD_Strobe

        ; send low nibble
        sbi PORTB, 3
        swap data ; move low nibble to high nibble
        out PORTC, data ;send lower nibble
        rcall LCD_Strobe
```

```
        rcall delay_100ms ; wait for command to complete
        ret

; set LCD interface
; 2 lines, 5x8 font
set_interface:
        ldi data, 0x28
        rcall send_instruction
        ret
; enable LCD display and cursor
; 0x0D (Display ON, Cursor OFF, Blink ON)
enable_display:
        ldi data, 0b00001101
        rcall send_instruction
        ret
; clear LCD and return home
; 0x01 = Clear Display
; 0x02 = Return Cursor to Home
clear_and_home:
        ldi data, 0x01
        rcall send_instruction
        ldi data, 0b00000010 ;cursor home
        rcall send_instruction
        ret
; set cursor direction
; 0x06 = Increment cursor, no display shift
set_cursor_direction:
        ldi data, 0x06
        rcall send_instruction
        ret

; move Cursor to First Line, Position 0
; DDRAM address: 0x00 | 0x80 = 0x80
first_line:
        ldi data,0b10000000
        rcall send_instruction
        ret

; move Cursor to Second Line, Position 0
; DDRAM address: 0x40 | 0x80 = 0xC0
second_line:
        ldi data,0b11000000
        rcall send_instruction
        ret

; move Cursor to Duty Cycle Display Location
; DDRAM address: 0x05 | 0x80 = 0x85
dc_location:
        ldi data,0b10000101
        rcall send_instruction
        ret

; move Cursor to Fan Status Location
; DDRAM address: 0x44 | 0x80 = 0xC4
fan_location:
        ldi data,0b11000100
        rcall send_instruction
        ret
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Delays
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
delay:
        ; delay for 250ns
        ; uses three nop abd one ret because each one is one cycle
        ; and each cycle on an ATmega328P is 62.5ns (1/16MHz)
        delay_250ns:
                nop
                nop
                nop
                nop
                nop
                nop
                ret
        ; delay for 100ms
        ; uses the 500us delay to make an 100ms delay
        delay_100ms:
                ldi r27, 200
        delay_100ms_cont:
                cpi r27, 0
                rcall delay_500us
                dec r27
                brne delay_100ms_cont
                ret
        ; delay for 5ms
        ; uses the 500us delay to make an 5ms delay
        delay_5ms:
                ldi r27, 10
        delay_5ms_cont:
                cpi r27, 0
                rcall delay_500us
                dec r27
                brne delay_5ms_cont
                ret
        ; delay for 500us, this is for the initialization, at least 200us
        ; use timer counter 2 for LCD and timer counter 0 for WPM
        delay_500us:
                ldi r24, 0b00000011 ; Set prescaler to 32
                sts TCCR2B, r24
                ldi r24,6
                sts TCNT2, r24 ; Set timer count to 6

                in r31,TIFR2 ; tmp <-- TIFR0
                sbr r31,1<<TOV2 ; clears the overflow flag
                out TIFR2,r31
                wait:
                        in r31,TIFR2 ; tmp <-- TIFR0
                        sbrs r31,TOV2 ; Check overflow flag
                        rjmp wait
                        ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Button Interrupt
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
button_interrupt:
wait_int:
        cpi r20, 1 ; see if button is already on
        breq fan_off ; if so turn it off
```

```
; Turn fan ON
fan_on:
        ldi r20, 1 ; set the register for future use
        out OCR0B, r21 ; turn on fan with previous DC

        rcall fan_location ; move cursor to LCD fan status location
        ; display YES
        ldi char, 0x59 ; 'Y'
        rcall display_char
        ldi char, 0x65 ; 'e'
        rcall display_char
        ldi char, 0x73 ; 's'
        rcall display_char
        rjmp end_button_interrupt

; Turn fan OFF
fan_off:
        ldi r20, 0 ; set the register for future use
        mov r27, r20 ; turn off fan, DC = 0
        out OCR0B, r27

        rcall fan_location ; move cursor to LCD fan status location
        ; display NO
        ldi char, 0x4E ; 'N'
        rcall display_char
        ldi char, 0x6F ; 'o'
        rcall display_char
        ldi char, 0x20 ; clear last character spot
        rcall display_char
        rjmp end_button_interrupt

end_button_interrupt:
        sbis PIND, 2 ; wait for button to be released (high)
        rjmp wait_int
        rcall dc_location ; move cursor back to duty cycle display
        rcall delay_5ms
        reti

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; rpg Interrupt
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
rpg_interrupt:
        mov r0, r26 ; store temp into 0
        ldi r26, 1
        cpse r20, r26 ; check if fan should be on or off. skips next if not equal (equal = off)
        rjmp end_rpg_interrupt
        mov r26, r0

        in r29, PINB ; read RPG signals
        andi r29, (1 << 0) | (1 << 1) ; keep only PD0 & PD1
        cp r29, prev_state ; compare with previous state
        breq end_rpg_interrupt ; if no change exit ISR

        ; only detect movement when PB0 (CH0) changes
        sbrc prev_state, 0 ; if previous PB0 was 0, continue checking
        sbrs r29, 0 ; if new PB0 is 1, this is a valid step
        rjmp update_prev_state ; otherwise, ignore transition
```

```
        ; full Quad Decoding (Detect Clockwise or Counterclockwise)
        sbrc r29, 1 ; if PB1 (CHB) is 1, CW detected
        rjmp CCW
        rjmp CW

update_prev_state:
        mov prev_state, r29
        rjmp end_rpg_interrupt

CCW:  ; Previously CW
        push r28
        ldi r28, 199 ; prevents incrementing past 99% duty cycle
        cpse r21, r28
        inc r21
        sts OCR0B,r21 ; send out new DC
        pop r28
        rjmp end_rpg_interrupt

CW:  ; Previously CCW
        push r28
        ldi r28, 0 ;prevents incrementing below 0% duty cycle
        cpse r21, r28
        dec r21
        sts OCR0B,r21 ; send out new DC
        pop r28
        rjmp end_rpg_interrupt
end_rpg_interrupt:
        rcall delay_5ms ; delay 5ms
        mov r26, r0
        reti
```

6. Appendix B: References

ATmega48A/PA/88A/PA/168A/PA/328/P, Microchip Technologies, 2018. <https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf>

Beichel, Reinhard. Arduino_V2.pdf, Spring 2025. The University of Iowa. 04/09/2025. <https://uiowa.instructure.com/courses/248357/files/29320694?module_item_id=804231>

Beichel, Reinhard. Rotary Pulse Generators & Lab 3 used in ECE:3360 Embedded Systems, Spring 2025. The University of Iowa. 04/09/2025. <https://uiowa.instructure.com/courses/248357/files/29778930?module_item_id=816215>

Beichel, Reinhard. Lecture - Liquid-Crystal Displays (LCDs) used in ECE:3360 Embedded Systems, Spring 2025. The University of Iowa. 04/09/2025. <https://uiowa.instructure.com/courses/248357/files/29883422?module_item_id=8183401>

HD44780, HITACHI, 1998. <https://cdn.sparkfun.com/assets/9/5/f/7/b/HD44780.pdf>