DTU Engineering Technology
Department of Engineering Technology and Didactics

date

Opgave 5 ADC-PWM LYSSTYRING

Data Technique and Programming
62734

Group 29

Report title and subtitle is defined by writing in the fields ”Title” and ”Subject” in Document Properties. (Choose File -> Properties -> Summary or Office Button -> Prepare -> Properties). Right click and choose *Update Field,* to update the actual field (on all subsequent pages).

Supervisor(s):

Student name	**Student number**	**Signature**

Table of Contents

Note: The *Table of Contents* above is automatically updated. So, do not change anything on this page. The changes you make on the following pages are updated

when you right click on Table of Contents and choose Update field. Remember to format Heading 1, Heading 2 etc. (Choose format and Style)

1. Introduktion

1.1 System beskrivelse

Vores kode viser et system hvor man læser ADC værdien af en potentiometer. Derefter bliver ADC værdien konverteret til PWM duty cycle, som derefter vises på både OLED-Display og Seriel Monitor. Denne værdi bliver så sendt til en lysdiode, som så justere sin lystyrke baseret på ADC værdien.
Før tællingen af ADC værdien starter, bliver brugeren spurgt om minimum og maximum på PWM duty cycle. Efter man har indtastet værdierne på minimum og maximum, starter systemet med at tælle. Imens systemet kører og beregner ADC værdien, kan man trykke på en ekstern interrupt (knap), der stopper tællingen og spørger om nye maximum og minimums værdier.

1.2 Milestone plan

1.2.1 Microcontroller part

Iteration	Tasks	Responsible	Due-date
9-10/5	Opstille board og vise adc værdi	alle	10/5
10-15/5	Vise pwm cycle og styre led	alle	15/5
15-20/5	Ændre msdelay, så vi bruger registre	alle	20/5
20-25/5	Lave timer1, I stedet for timer0	alle	25/5
25-28/5	Sørger for at bruger kan indstaste maximum og minimums værdi i starten og efter knaptryk	alle	28/5
20/4-4/5	Lave rapport	alle	4/5

Conclusion on milestones plan:

2. Requirements specifikation

2.1 Microcontroller part

2.1.1 Functional requirements

The table below defines the primary set of requirements.

Functional requirements	
R1	Mikrocontrolleren skal have kapacitet til at processere input fra et potentiometer ved at bruge en ADC og vise spændingsresultatet på en OLED-skærm med præcision ned til hundrededele.
R2	Apparatet skal kunne modulere lysstyrken af en LED via PWM, der styres af det aflæste ADC input.
R3	Interfacet til brugerinput via UART skal kunne modtage og implementere brugerdefinerede minimums- og maksimumsværdier for PWM-signalets pulsbredde.
R4	Duty cycle for PWM skal kunne fremvises som en procentdel på OLED-skærmen.
R5	Frekvensen for ADC-klokken skal kunne justeres til enten 125 kHz for en 10-bit opløsning eller 1 MHz for en 8-bit opløsning.
R6	En interrupt-rutine skal håndtere hændelser, hvor ADC-data er klar til aflæsning.
R7	Frekvensen for ADC-sampling, styret af en Timer1 overflow interrupt, skal kunne sættes til for eksempel 9500 Hz.
R8	Der skal være mulighed for at indstille mikrocontrolleren til automatisk udløsning (auto-trigger) for ADC-målinger.
R9	I hovedprogrammet (main-funktionen) skal der benyttes en kontrolstruktur, der enten baseres på switch case eller en tilstandsmaskine for at håndtere PWM-styring baseret på ADC.
R10	Det skal specificeres, hvor mange klokcyklusser der går, fra en ADC-sampling startes, til den er klar til læsning.
R11	Der skal implementeres en kontrolmekanisme, der sikrer, at ADC-værdierne ligger inden for de definerede grænseværdier, før de anvendes til at opdatere PWM-pulsbredden.

Functional requirements	
R12	PWM-timerindstillingen skal kunne konfigureres til enten at være fasekorrekt eller fase- og frekvenskorrekt.

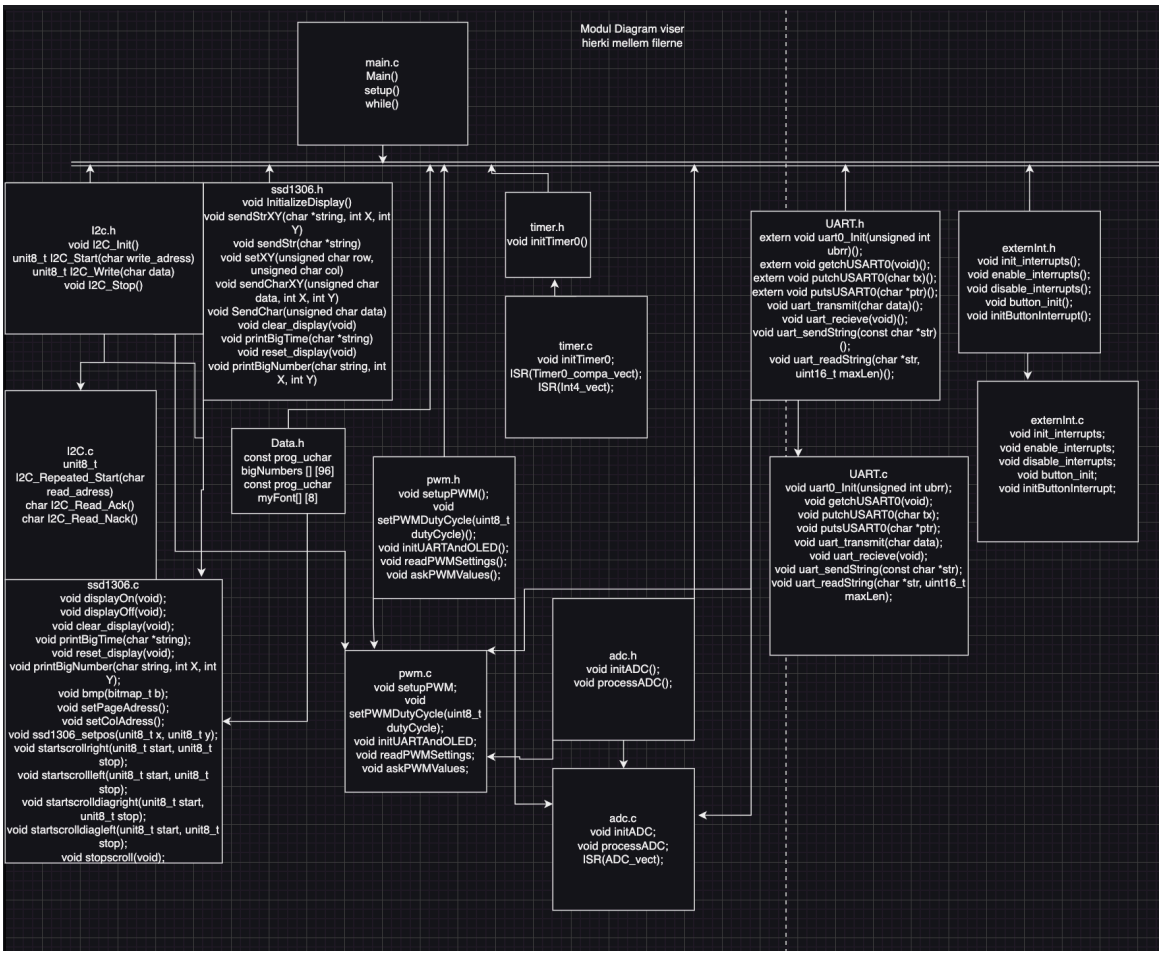
2.1.2 Non functional requirements

Non functional requirements	
NR1	UART-input og OLED-display skal være brugervenligt og give klar og letforståelig feedback om status på systemet
NR2	Mikrocontrolleren skal være kompatibel med andre tilknyttet hardware foreksempel potentiometre og OLED-skærme uden at kræve ændringer.
NR3	Koden skal være veldokumenteret med kommentarer og instruktioner, som gør det nemt at opdatere og fejlfinde
NR4	Systemet skal kunne skaleres op for at håndtere højere frekvenser af input og output uden væsentlige ændringer af den centrale arkitektur
NR5	Softwaren skal udvikles i et bredt anvendt programmeringssprog f.eks. C

3. Problem Solution, Microcontroller

3.1 C program

3.1.1 Modul diagram



Forklaring:

Hvad hver fil gør:

I2C.h: Tillader kommunikation via I2C på mikrokontrolleren

I2C.c: Implementerer funktioner til at styre I2C-kommunikationen

data.h: Indeholder alle static arrays til at tegne ting på OLED-Display og indeholder data til OLED-Display

ssd1306.h: Definerer funktioner og konstanter via I2C til at styre OLED-Display

ssd1306.c: Indeholder funktioner som initialisere og styrer OLED-Display

adc.h: Håndterer opsætning og behandler "analog til digital" konvertering til system

adc.c: Implementerer logikken til at læse data fra analog til digital konvertering

externInt.h: Indeholder logik til opsætning og håndtering af eksterne interrupts (knap)

externInt.c: Implementerer eksterne interrupts (knap) til debouncing og håndtering af knaptryk
pwm.h: Indholder funktioner til styring af PWM og funktioner til at kontrollere LED
pwm.c: Implementerer PWM funktioner, som PWM duty cycles og håndtering af brugerinput for PWM.
timer.h: Styrer opsætning af systemets timere til periodisk at udløse handlinger som ADC-konvertering
timer.c: Implementerer funktioner til at indstille og håndterer timer baseret operationer og interrupts.
UART.h: Definere funktioner for at initialisere UART, både for at sende og modtage data.
UART.c: Definere funktioner for at initialisere UART, både for at sende og modtage data og indeholder opsætning af UART-registrene for at opnå baudraten og dataformat
Main.c: Initialiserer systemets hardwarekomponenter og håndterer et kontinuerligt loop, der reagerer på brugerinput via en knap til at opdatere PWM-indstillinger og behandler ADC-læsninger for dynamisk at justere output baseret på sensorinput.

Hvordan filerne hænger sammen:

Main.c er hjertet i systemet og inittialiserer og bruger alle filerne til at kører. Den kalder de forskellige funktioner i alle filer, til at køre systemet.

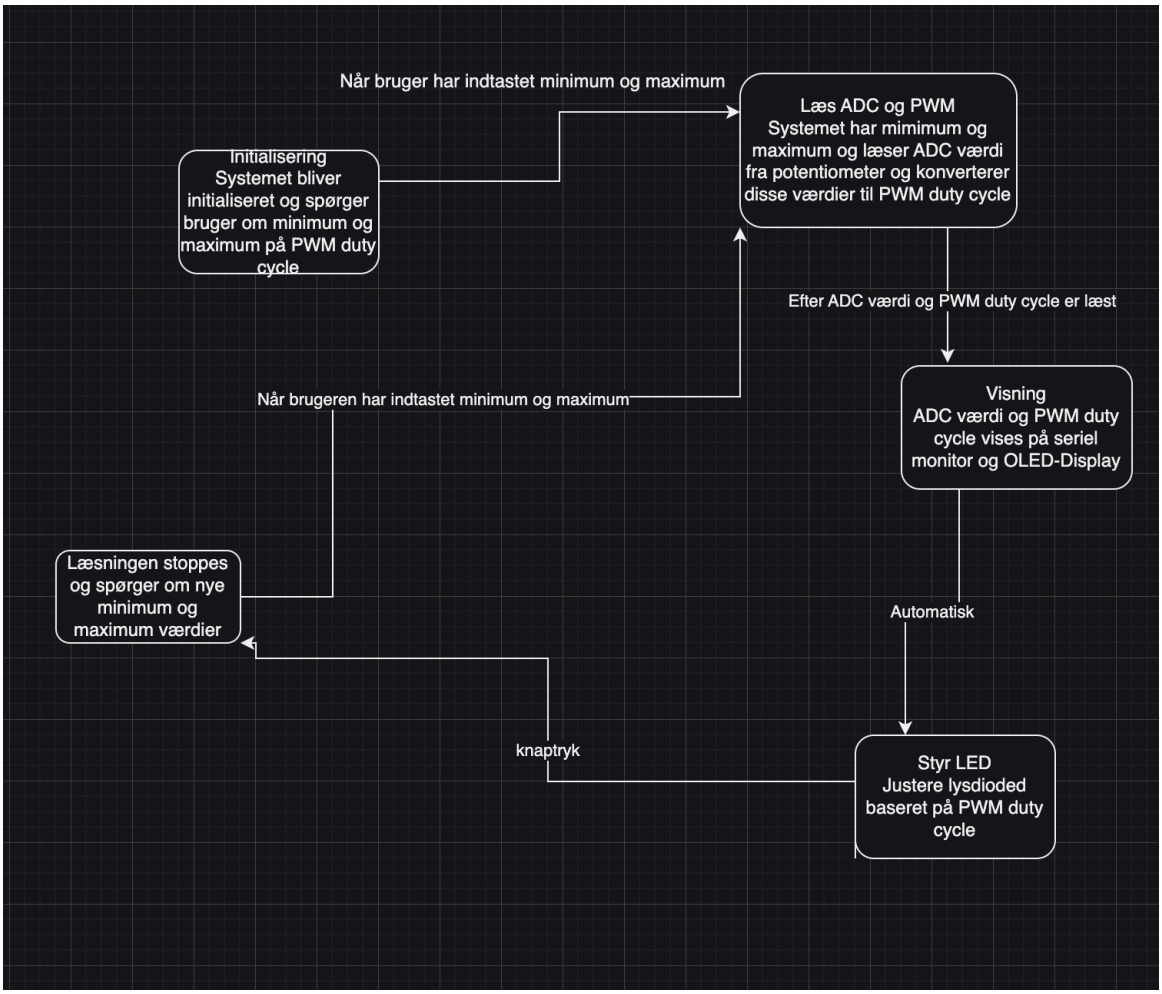
UART filerne implementerer seriel kommunikation med seriel monitor. main.c bruger UART filerne til at sende og modtage data mellem bruger og ATmega2560 mikrokontroller. UART bruges også til kommunikation, som sætter uret og brugerinterface.

externInt filerne Filerne håndterer eksterne interrupts altså knap, samt implementerer den debounce logik for at sørge for et pålideligt input. main.c bruger externInt filernes funktioner til at tillade brugeren at interegere med systemet eksternt, altså at stoppe tælling og skrive nye minimum og maximum værdier.

ADC filerne bruger timer filerne til at udløse ADC læsning ved intervaller, som bruges til at sikre sikker og præcis dataindsmaling. Filerne sender også data og fejlmedellelser til UART filerne for kommunikation med brugeren via seriel monitor.

PWM filerne Modtager og anvender data fra ADC til at justere PWM duty cycle og for at justere lysdiodens lysstyrke. PWM filerne bruger også UART filerne til at få brugerinput for PWM duty cycle minimum og maximum værdier, samt sender den også opdateringer og fejlmedelser.
Timer filerne triger ADC konvertering.
I2C filerne håndterer I2C kommunikation mellem komponenterne og hjælper main.c med at bruge funktionerne til at initiliasiere og bruge OLED-Display
Ssd1306 filerne styrer OLED-Display og bruger I2C til at sende og modtage data fra OLED-Display
Data.h hjælper main.c med at køre programmet og indeholder data der skal bruges til OLED-Display

3.1.2 State diagram and flowchart for main – modul



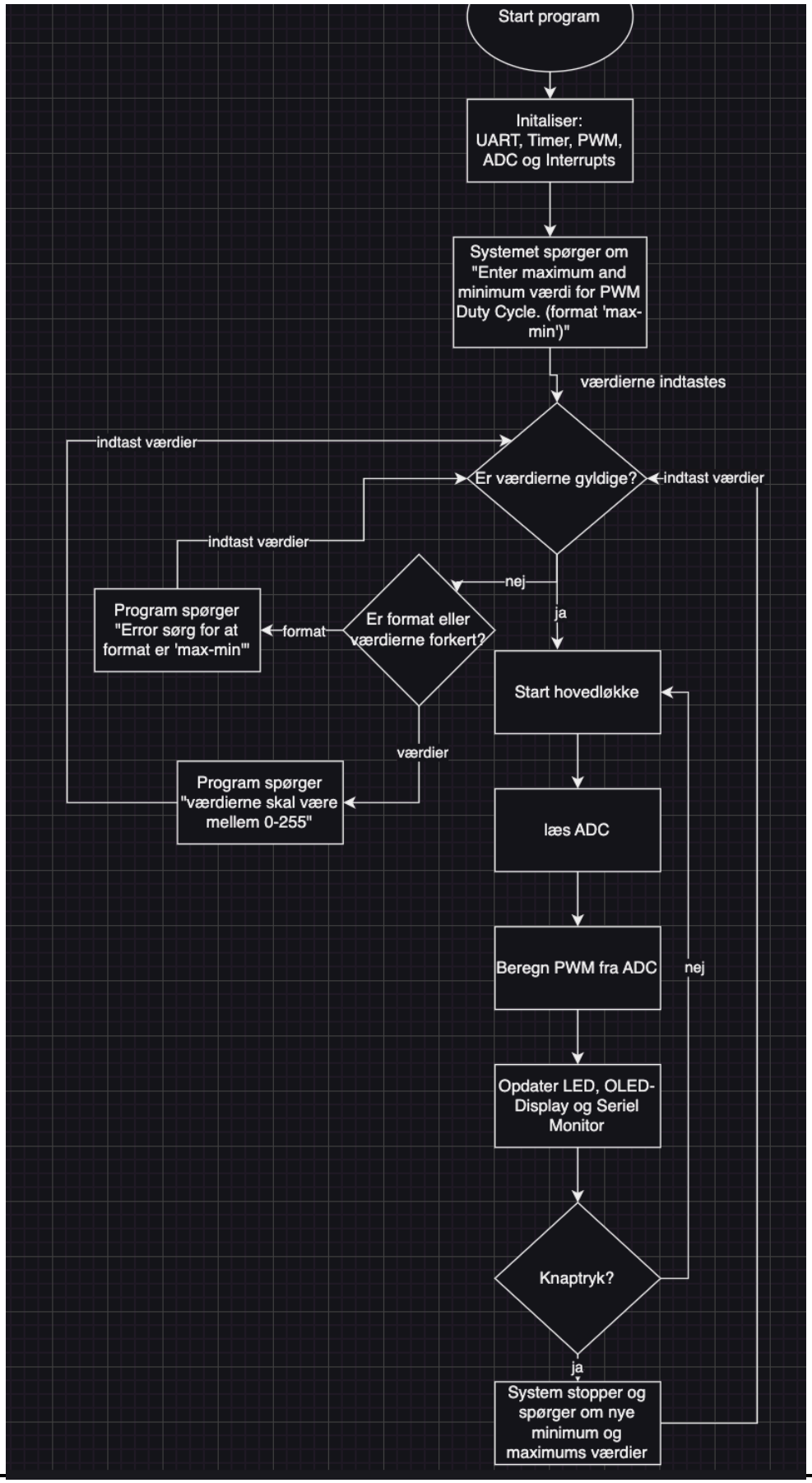
Forklaring:

Statediagrammet viser, hvordan et system bliver sat i gang og hvordan det arbejder sammen med en bruger for at indstille og styre parametre for PWM duty cycles baseret på information fra en ADC. Diagrammet begynder med, at systemet bliver initialiseret, hvilket betyder, at det bliver klargjort og spørger brugeren om at sætte minimums- og maksimumsværdier for PWM duty cycle. Når disse værdier er indstillet, så læser systemet data fra en ADC, som måler den fysiske position af et potentiometer og konverterer disse målinger til PWM duty cycles.

Disse værdier vises derefter på OLED-Display og seriel monitor, så brugeren kan se, hvad de nuværende indstillinger er. Samtidig justeres lysstyrken på en LED baseret på de ADC-værdier, der er blevet læst, så brugeren kan se effekten af deres indstillinger direkte. Systemet giver også brugeren en mulighed for at stoppe og

genindstille minimums- og maksimumsværdierne, hvilket fører til, at systemet stopper med at læse værdierne og lader brugeren lave nye indstillinger.

Flowchart for main()


Start program
Initaliser: UART, Timer, PWM, ADC og Interrupts
Systemet spørger om "Enter maximum and minimum værdi for PWM Duty Cycle. (format 'max-min')"
værdierne indtastes
Er værdierne gyldige?
indtast værdier
indtast værdier
indtast værdier
nej
ja
Er format eller værdierne forkert?
format
Program spørger "Error sørg for at format er 'max-min'"
værdier
Program spørger "værdierne skal være mellem 0-255"
Start hovedløkke
læs ADC
Beregn PWM fra ADC
nej
Opdater LED, OLED-Display og Seriel Monitor
Knaptryk?
ja
System stopper og spørger om nye minimum og maximums værdier

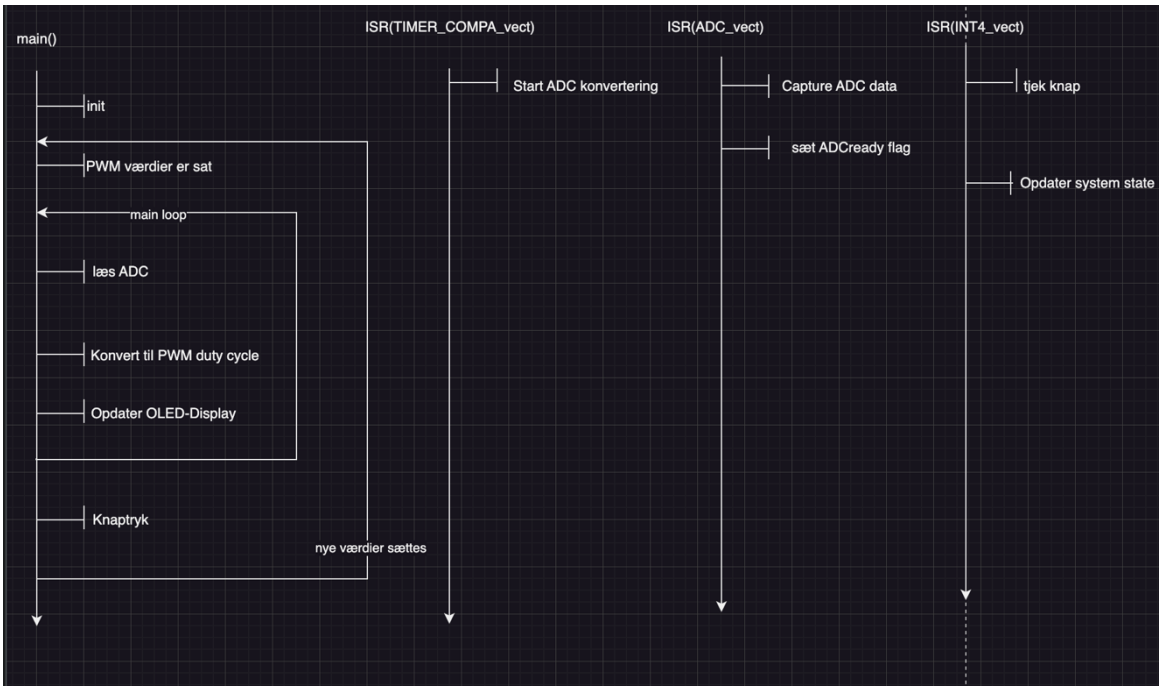
Forklaring:

Dette flowchart skitserer processen i et program, der styrer PWM duty cycles baseret på værdier læst fra en ADC. Programmet starter med at initialisere nødvendige hardwarekomponenter som UART, timer, PWM, ADC og interrupts. Dette sikrer, at alle systemets dele er klare til at modtage og behandle data.

Næste skridt i programmet er at anmode brugeren om at indtaste minimums- og maksimumsværdier for PWM duty cycle. Disse værdier definerer de grænser, som PWM-signalet vil operere indenfor. Hvis brugeren indtaster værdierne korrekt, fortsætter programmet til hoveddelen af sin funktion, hvor det begynder at læse data fra ADC'en.

Disse ADC-værdier bruges til at beregne den nødvendige PWM duty cycle, som justerer outputtet fra en tilknyttet LED. Denne beregnede duty cycle samt ADC-værdierne vises på enten et OLED-display eller en seriel monitor. Dette giver brugeren visuel feedback om, hvordan ADC-input påvirker PWM-output. Programmet tjekker løbende, om der trykkes på en knap. Hvis der trykkes på knappen, stopper programmet, og spørger brugeren om at indtaste nye minimum og maksimum værdier. Dette tillader brugeren at justere grænserne for PWM duty cycle og starte processen forfra.

3.1.3 Interrupt diagram



Dette diagram viser hvordan en mikrocontroller fungerer med forskellige interrupts. Først initialiserer systemet og sætter de nødvendige værdier for at styre ting ved hjælp af PWM. Efter initialiseringen fortsætter systemet med at køre og udfører flere opgaver: det læser data fra en sensor gennem ADC, konverterer disse data til passende PWM-signaler, og viser information på en OLED-skærm.

For at kunne reagere på specifikke situationer bruger systemet tre forskellige interrupts: En timer-interrupt, der starter en konvertering af data fra analogt til digitalt, en ADC-interrupt, som håndterer når datakonvertering er færdig og signalerer, at data er klar, og en knap-interrupt, der lader systemet vide, at en bruger har trykket på en knap, hvilket får systemet til at opdatere sin tilstand.

3.1.4 Tabel over c-modul funktioner

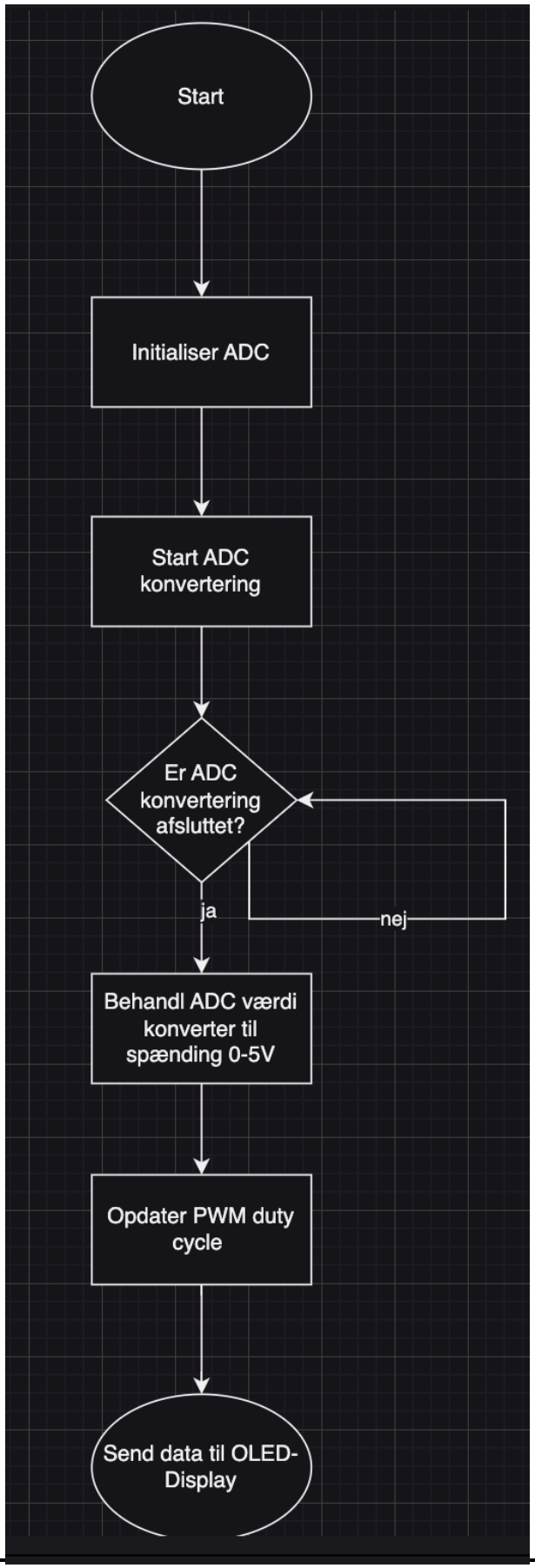
En table per modul

ADC.c

initADC()	Initialiserer ADC, vha register
processADC()	Behandler ADC læsning, konvertere den til spænding, beregning den

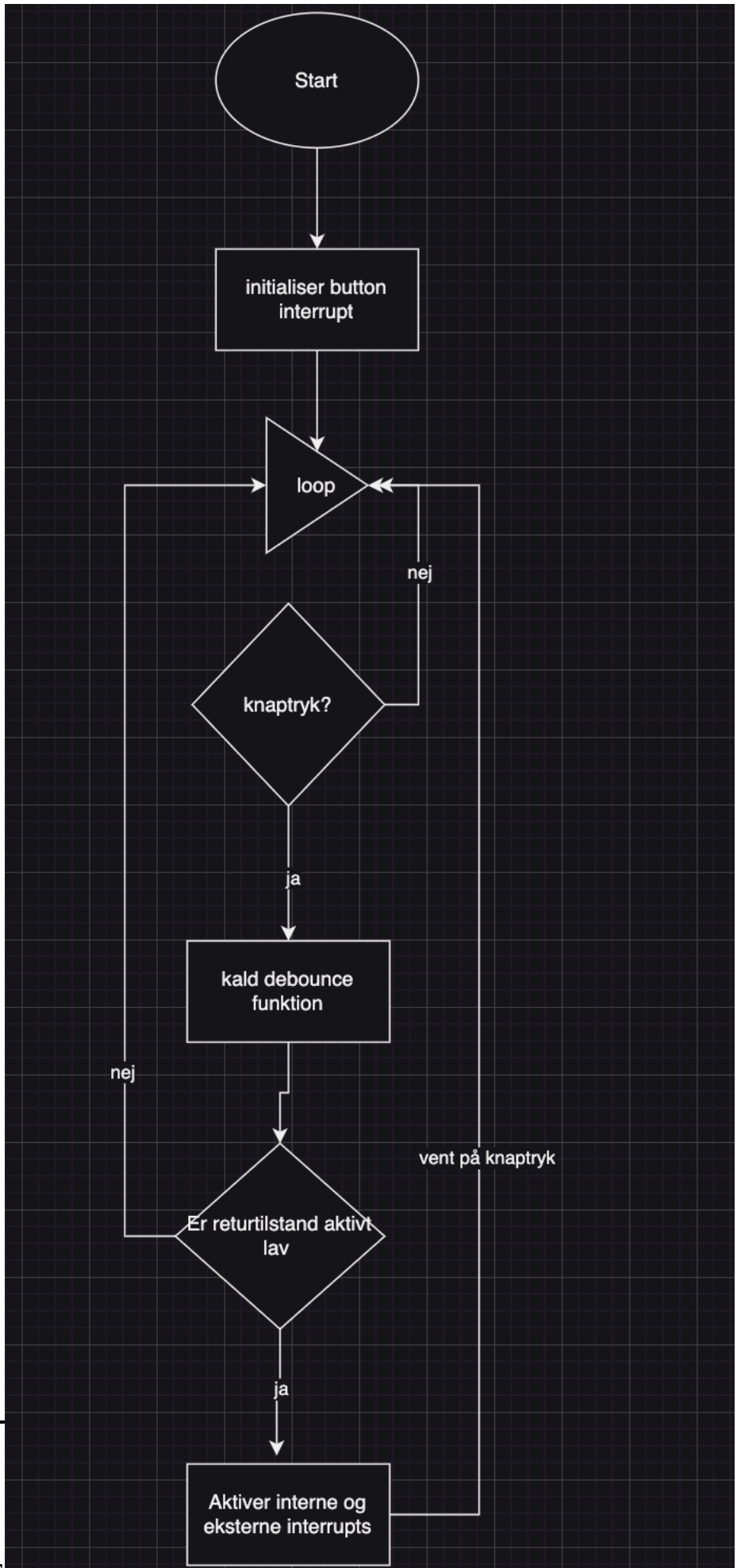
	tilsvarende PWM duty cycle, indstiller PWM duty cycle og viser både ADC spænding og PWM duty cycle på både seriel monitor og OLED-Display, via UART og I2C.
ISR(ADC_vect)	Udføres når en ADC aflæsning er klar. Den læser ADC værdien og sætter et flag, som angiver klarheden af værdien og styrer LED med værdien

Flowchart for adc.c


Start
Initialiser ADC
Start ADC konvertering
Er ADC konvertering afsluttet?
ja
nej
Behandl ADC værdi konverter til spænding 0-5V
Opdater PWM duty cycle
Send data til OLED-Display

externInt.c

uint8_t debounce(uint8_t pin)	Håndterer debounce på pin 2/PE4. Den returnere 1, hvis knap er trykket og returnere 0 hvis knap ikke er trykket
enable_interrupts()	Aktivere globale interrupts
disable_interrupts	Deatktivere globale interrupts
initButtonInterrupt()	Konfigurere ekstern interrupt på PE4, ved at sætte den som input, aktivere pull up modstand og indstiller ekstern interrupt til faldne kant på INT4

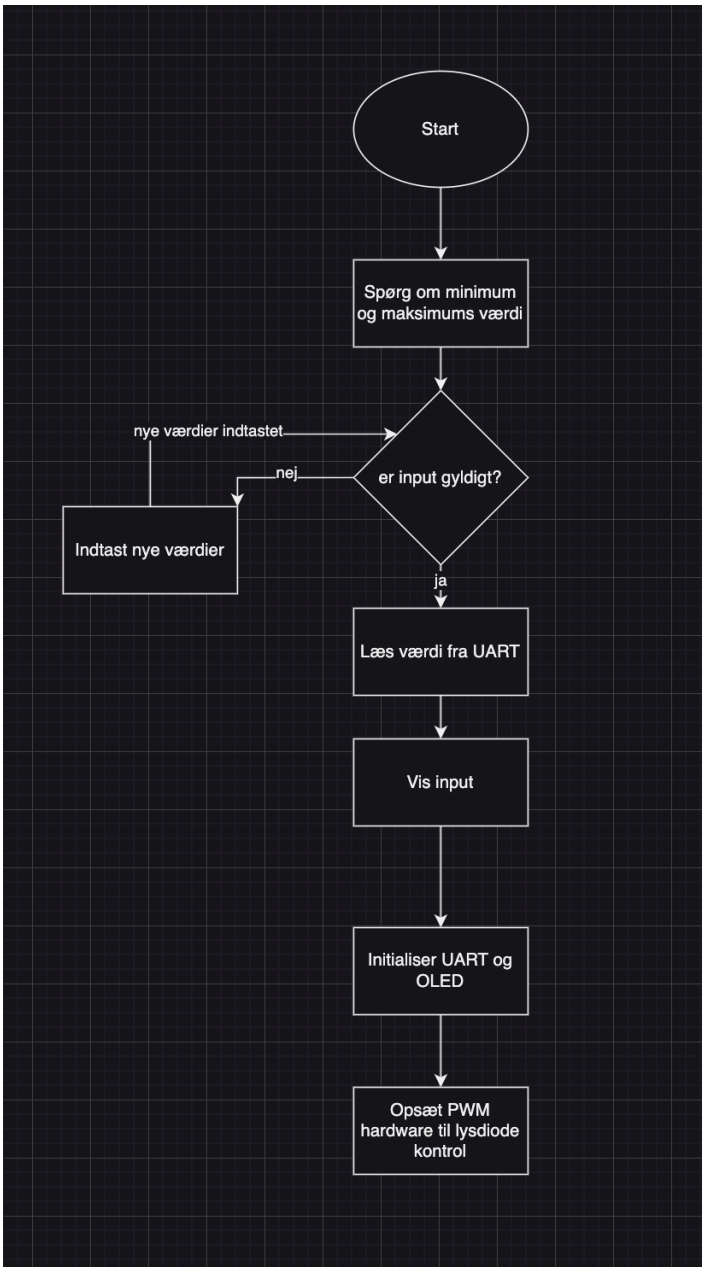

Start
initialiser button interrupt
loop
nej
knaptryk?
ja
kald debounce funktion
nej
vent på knaptryk
Er returtilstand aktivt lav
ja
Aktiver interne og eksterne interrupts

externInt.c

PWM.c

setupPWM()	Opsætter PWM til lysdiode
setPWMDutyCycle()	Sætter PWM duty cycle til lysdiode
askPWMValues	Spørger brugeren om at indtaste minimum og maksimum værdi på PWM duty cycle. Hvis format eller værdier ikke er rigtigt, så kører system ikke og program spørger om nye værdier
initUARTAndOLED()	Initialisere OLED-Display og UART

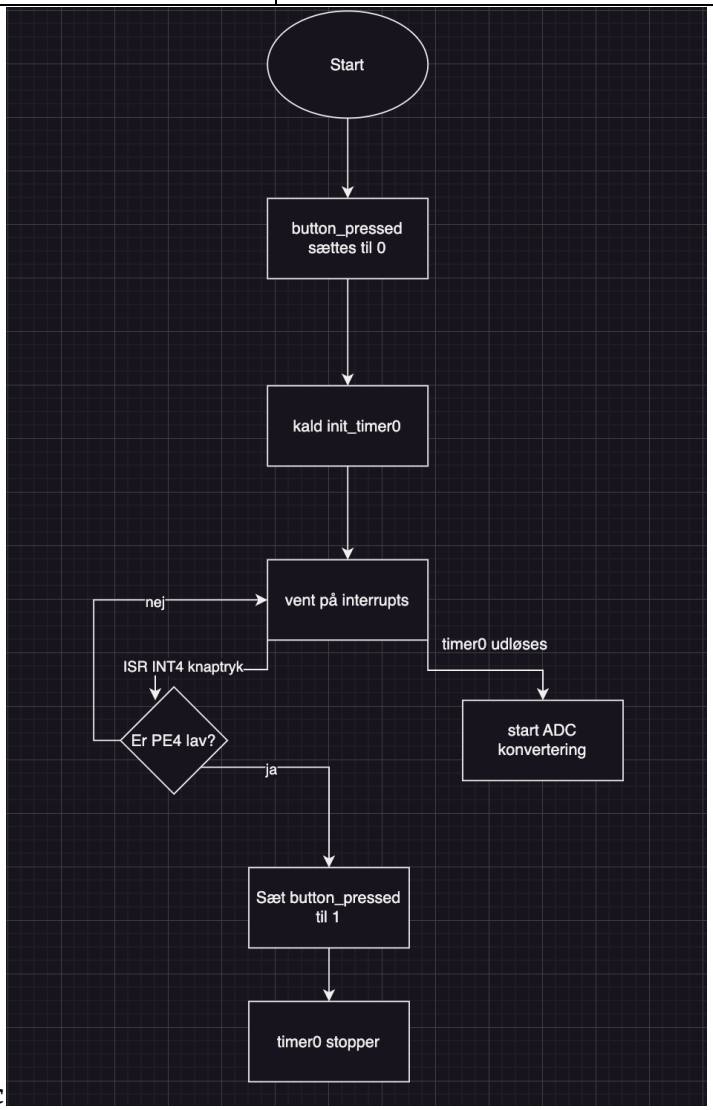
Flowchart for PWM.c



timer.c

initTimer0()	Opsætter timer0 til CTC mode. Den konfigurere dens prescaler og compare match-værdi for at opnå interrupt frekvens
ISR(TIMER0_COMPA_vect)	ISR til Timer0, som starter ADC-konvertering

ISR(INT4_vect)	ISR for INT4, der reagere på knap på PE4, som sætter button_pressed flag hvis knap er trykket



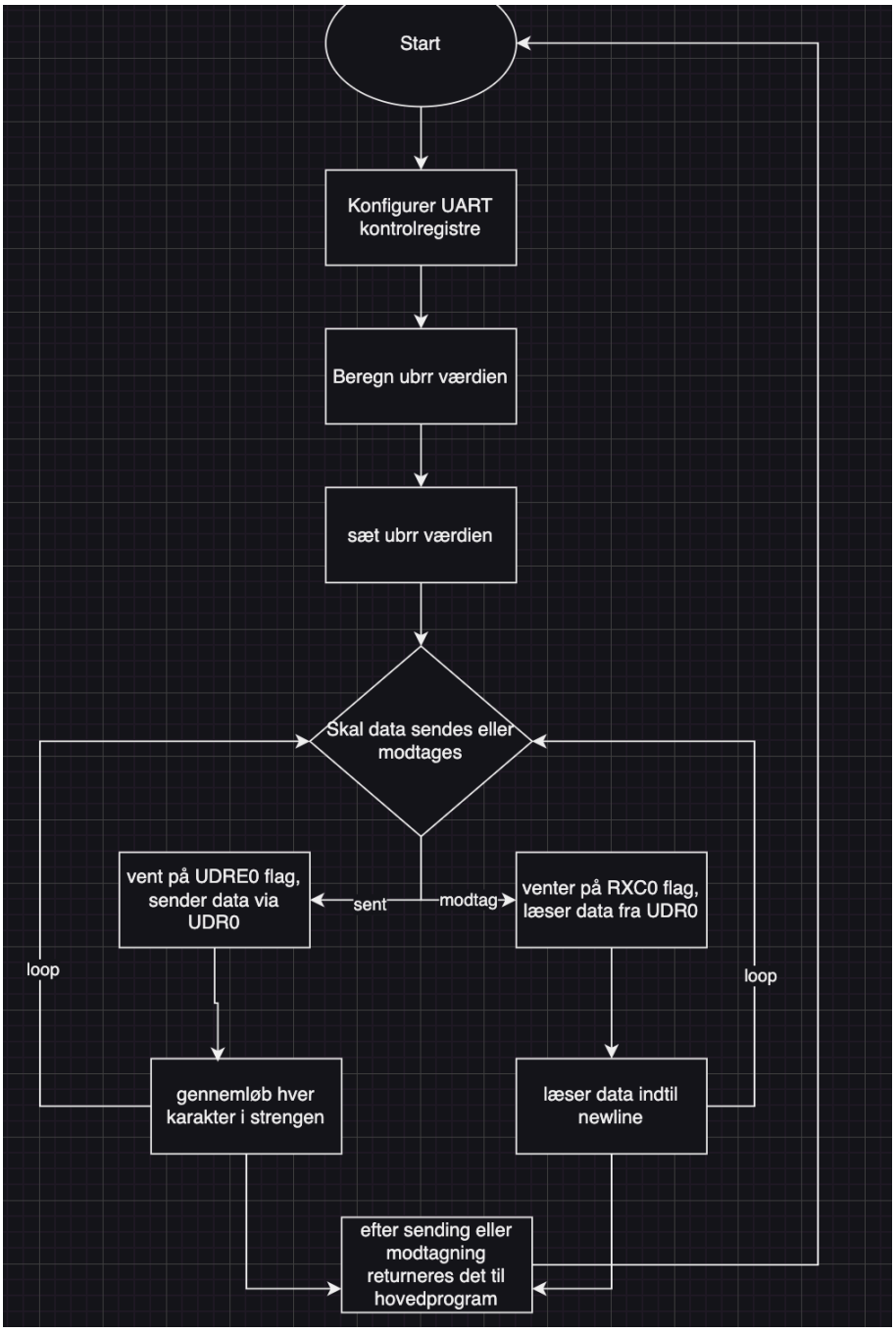
Flowchart for timer.c

UART.c

uart0_Init(unsigned int baudRate)	Initialisere UART kommunikation med 19200 baudrate
putsUSART0(char *ptr)	Sender et null-termineret streng via UART
putchUSART0(char tx)	Sender et enkelt tegn via UART
uart_transmit(char data)	Transmitter et enkelt tegn via UART
uart_recieve(void)	Modtager et enkelt tegn via UART

uart_sendString(const char *str)	Transmitter en streng via UART
uart_readString(char *str, uint8_t maxLen)	Læser en streng fra UART og gemmer den i buffer

Flowchart for UART.c



I2C.c

I2C_Init();	Initialisere I2C kommunikation og indstiller SCL-frekvensen til 100 kHz
I2C_Start(char write_adress);	Starter I2c kommunikation og sender skriveadressen
I2C_Repeated_Start(char read_adress);	Udfører gentagende start for I2C kommunikation og sender læseadressen
I2C_Write(char data);	Skriver en byte data over I2C og returnere status for transmission
I2C_Read_Ack();	Læser en byte data fra I2C med ACK, dette funktion bruges når bytes skal modtages
I2C_Read_Nack();	Læser en byte data fra I2C uden ACK, bruges når den sidste byte der skal modtages
I2C_Stop();	Sender en stop betingelse for at afslutte I2C kommunikationen

ssd1306.c

Ssd1306_command(uint8_t c);	Sender en kommando til SSD1306-driveren.
Ssd1306_data(uint8_t c);	Sender en data-byte til SSD1306-driveren.
setColAdress();	Indstiller kolonneadresserne for horisontal eller vertikal adressering.

setPageAddress();	Indstiller sideadresserne for horisontal eller vertikal adressering.
InitializeDisplay();	Initialiserer displayet med en række SSD1306-specifikke kommandoer.
Reste_display();	Nulstiller displayet ved at slukke, rydde og tænde det igen.
displayOn();	Tænder displayet.
displayOff();	Slukker display
Clear_display();	Rydder displayet ved at fulde skærmen med tomme pixels
printBigTime(char *string);	Printer en streng I stor skrift på displayet
printBigNumber(char string, int X, int Y);	Printer et stort tal på specifikke koordinater på displayet
sendChar(unsigned char dta, int X, int Y);	Sender en karakter til dispalyet via I2C
sendCharXY(unsigned char dta, int X, int Y);	Sender en karakter til specifikke koordinater på displayet
setXY(unsigned char row, unsigned char col);	Indstiller markørens position på displayet
sendStr(char *string);	Sender en streng til displayet uafhængigt af marlørens position
sendStrXY(char *string, int X, int Y);	Sender en streng til spevifikke koordinater på displayet
Ssd1306_setpos(uint8_t x, uint8_t y);	Indstiller positionen for næste skriveoperation på displayet
print_fonts();	Viser alle tilængelige skrifttyper på displayet

ssd1306_draw_bmp(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1, const uint8_t bitmap[]);	Tegner et bitmap på displayet mellem specificerede koordinater
invertDisplay(uint8_t i);	Inverter displayets visning baseret på den givne parameter
startsscrollright(uint8_t start, uint8_t stop);	Starter en horisontal rulning til højre fra start til stop rækker
startsscrollleft(uint8_t start, uint8_t stop);	Starter en horisontal rulning til venstre fra start til stop rækker
startscrolldiagright(uint8_t start, uint8_t stop);	Starter en diagonal rulning til højre
startsscrolldiagleft(uint8_t start, uint8_t stop);	Starter en diagonal rulning til venstre
stopscroll();	Stopper alle rulninger på displayet
dim(bool dim);	Justere lystyrken på displayet, dæmper hvis true normal hvis false

4. Testing

4.1 Microcontroller part

4.1.1 Acceptance Tests

Formålet med accepttesten er at sikre præcis funktionalitet og samarbejde mellem software og hardware, således at lysstyrken på en lysdiode justeres korrekt baseret på input fra en analog-til-digital konverter (ADC). Vores testområde omfatter:

UART kommunikation: Til brugerindstillinger som fx maksimum og minimum.
PWM signalering: Baseret på ADC-værdien.
ADC konvertering: Af analoge input til digitale signaler.
Visning på OLED: Af spænding og PWM duty cycle.
Vores acceptkriterier inkluderer:

Systemet skal korrekt kunne modtage og anvende brugerindstillede PWM duty cycle-værdier inden for intervallet 0 til 255, hvor minimum ikke må være større end maksimum.
PWM duty cycle skal kunne justere lysdiodens lysstyrke præcist.
ADC skal nøjagtigt konvertere analoge signaler fra potentiometeret til digitale værdier mellem 0 og 5V.
OLED-displayet skal korrekt vise ADC-værdien og PWM duty cycle-værdien.
TEST CASE 1:

Navn: TC1: Korrekt modtagelse og anvendelse af brugerindstillede PWM duty cycle-værdier.

Start systemet og initialisér det. UART beder om minimums- og maksimumsværdier.
Indstil værdierne til f.eks. 200-10, og systemet starter.
Tryk på knappen for at starte tælling, og når den stopper, beder systemet om nye maksimums- og minimumsværdier. Indtast f.eks. 255-0, og systemet starter igen.
Bekræft, at systemet korrekt modtager og anvender brugerindstillede PWM duty cycle-værdier.
TEST CASE 2:

Navn: TC2: Justering af lysdiodens lysstyrke ved hjælp af PWM duty cycle.

Forventet resultat: PWM duty cycle skal kunne justere lysdiodeens lysstyrke i overensstemmelse med dens procentværdi.

Forbind breadboardet og mikrokontrolleren til computeren og start systemet.
Indstil minimum og maksimumsværdier til 255-0.
Kontroller PWM duty cycle på både seriel monitor og OLED-display.
Juster potentiometeret til maksimum.
Bekræft, at lysdiodens lysstyrke øges til maksimum i overensstemmelse med PWM duty cycle-værdien.
TEST CASE 3:

Navn: TC3: Verificér, at ADC nøjagtigt konverterer input fra potentiometeret til digital værdi mellem 0 og 5V.

Forventet resultat: ADC skal præcist konvertere analoge signaler til digitale værdier i det forventede spændingsområde.

Forbind breadboardet og mikrokontrolleren til computeren og start seriel monitor.
Indstil minimum og maksimumsværdier til 255-0.
Indstil potentiometeret til maksimum, og verificér på seriel monitor og OLED-display, at værdien læses korrekt som 5V.
TEST CASE 4:

Navn: TC4: Visning af PWM duty cycle på OLED-display.

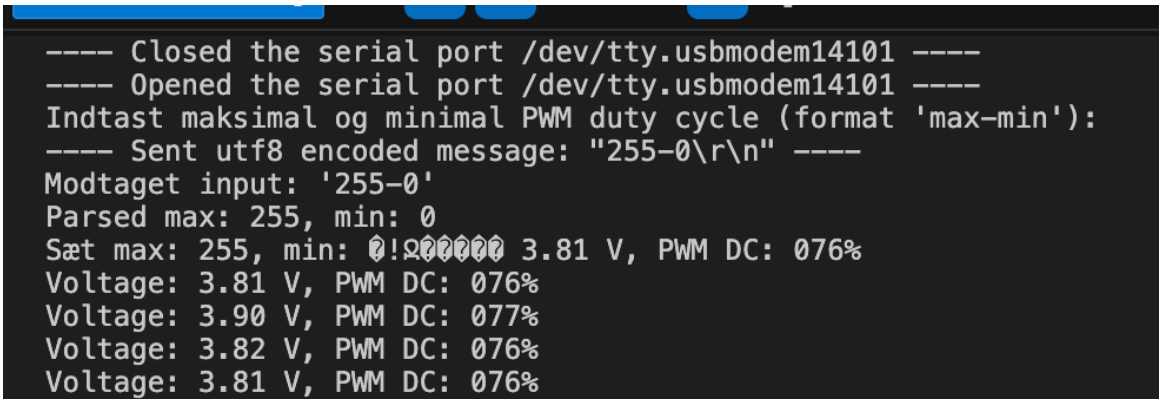
Forventet resultat: OLED-displayet skal korrekt vise PWM duty cycle-værdien.

Start systemet og indtast minimums- og maksimumsværdierne.
Juster potentiometeret, og observer på OLED-displayet, at PWM duty cycle-værdierne bliver korrekt vist.

4.1.2 Unit Tests

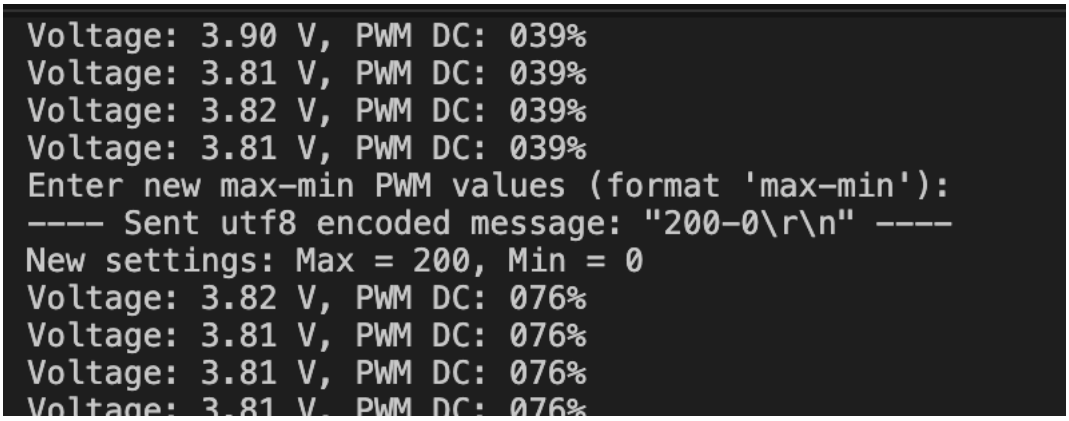
For at sikre korrekt funktionalitet af hvert modul, udfører vi unit tests ved at køre programmet og gennemgå alle funktioner i de respektive filer for at bekræfte deres virkemåde.

ADC.h og ADC.c
Disse filer er ansvarlige for at konvertere analoge signaler til spændingsværdier, omdanne disse til PWM duty cycles, og vise resultaterne på et OLED-display. Efter opkobling af breadboardet og mikrokontrolleren til computeren indtastede vi eksempelværdierne 255-0 og observerede, hvordan ADC-spændingen og PWM duty cycles blev præsenteret på både OLED-displayet og den serielle monitor, hvilket bekræfter funktionaliteten af ADC-modulerne.



ExternInt.h og ExternInt.c

Formålet med disse filer er at håndtere input fra en fysisk knap, som kan pausere målingerne og tillade indtastning af nye minimums- og maksimumsværdier for PWM. Under kørslen af programmet testede vi knappens funktionalitet, hvorved målingen blev pauset som forventet, hvilket bekræfter korrekt funktionalitet af externInt-filerne.



PWM.h og PWM.c

Disse filer konfigurerer og styrer PWM for LED-kontrol via en specifik pin og interagerer med brugeren via UART for at modtage og bekræfte PWM-indstillinger, samtidig med at en OLED-skærm og UART-kommunikation initialiseres for feedback og systemstatusvisning. Da de tidligere tests allerede har bekræftet visning af PWM duty cycle og meddelelser på den serielle monitor, kan vi bekræfte at PWM-koden fungerer efter hensigten.

Timer.h og Timer.c

Disse filer konfigurerer Timer0 i CTC-mode for præcise timingoperationer og håndterer interrupts for at initiere ADC konverteringer og detektere knaptryk. Da

ADC-konverteringerne påbegyndes som forventet, er der ikke behov for yderligere test af timer-funktionaliteten.

UART.h og UART.c
Dette modul initialiserer og håndterer UART-kommunikation, essentiel for dataudveksling med andre enheder. Med robuste funktioner for at sende og modtage data, samt håndtering af bufferlængder og afslutningstegn, blev funktionaliteten bekræftet i tidligere tests, hvilket indikerer at UART-modulerne opererer som forventet.

Disse tests understøtter systemets pålidelighed og sikrer, at hver komponent fungerer korrekt inden implementering i et større system.

5. Konklusion

5.1 Produkt orienteret konklusion

Projektet har demonstreret, at det er muligt at styre en lysdiodens lysstyrke præcist ved hjælp af en ADC-konvertering til at måle spændingsniveauet fra et potentiometer og dernæst anvende denne værdi til at justere en PWM duty cycle. Systemet opfylder alle specificerede funktionelle krav, herunder præcis styring af lysdiodens lysstyrke, responsiv brugerinput via UART, og dynamisk justering af ADC- og PWM-indstillinger baseret på brugerinteraktion.

De opnåede resultater inkluderer:

Implementering af en pålidelig ADC-læsning og konvertering.
Succesfuld modulation af en LED's lysstyrke via PWM baseret på ADC-input.
Effektiv brug af serielle kommunikationer til at justere systemindstillingerne dynamisk.
Integration og samspil mellem forskellige moduler som UART, timer, ADC og PWM uden væsentlige fejl.

5.2 Proces orienteret konklusion

Gruppen har arbejdet godt sammen og har vist, at de kan løse svære problemer sammen. Det gjorde, at vi fik projektet til at virke godt. Vores vejleder var virkelig nyttig og hjalp os med at holde styr på alt og sørge for, at vi ikke kom for sent eller glemte vigtige ting.

I løbet af projektet lærte vi, hvor vigtigt det er at teste tingene grundigt og sørge for, at de er af god kvalitet. Det gjorde os bedre til at håndtere både hardware og software. Samarbejdet var godt, vi snakkede åbent sammen og lærte hele tiden nyt, hvilket gjorde vores færdige produkt bedre.

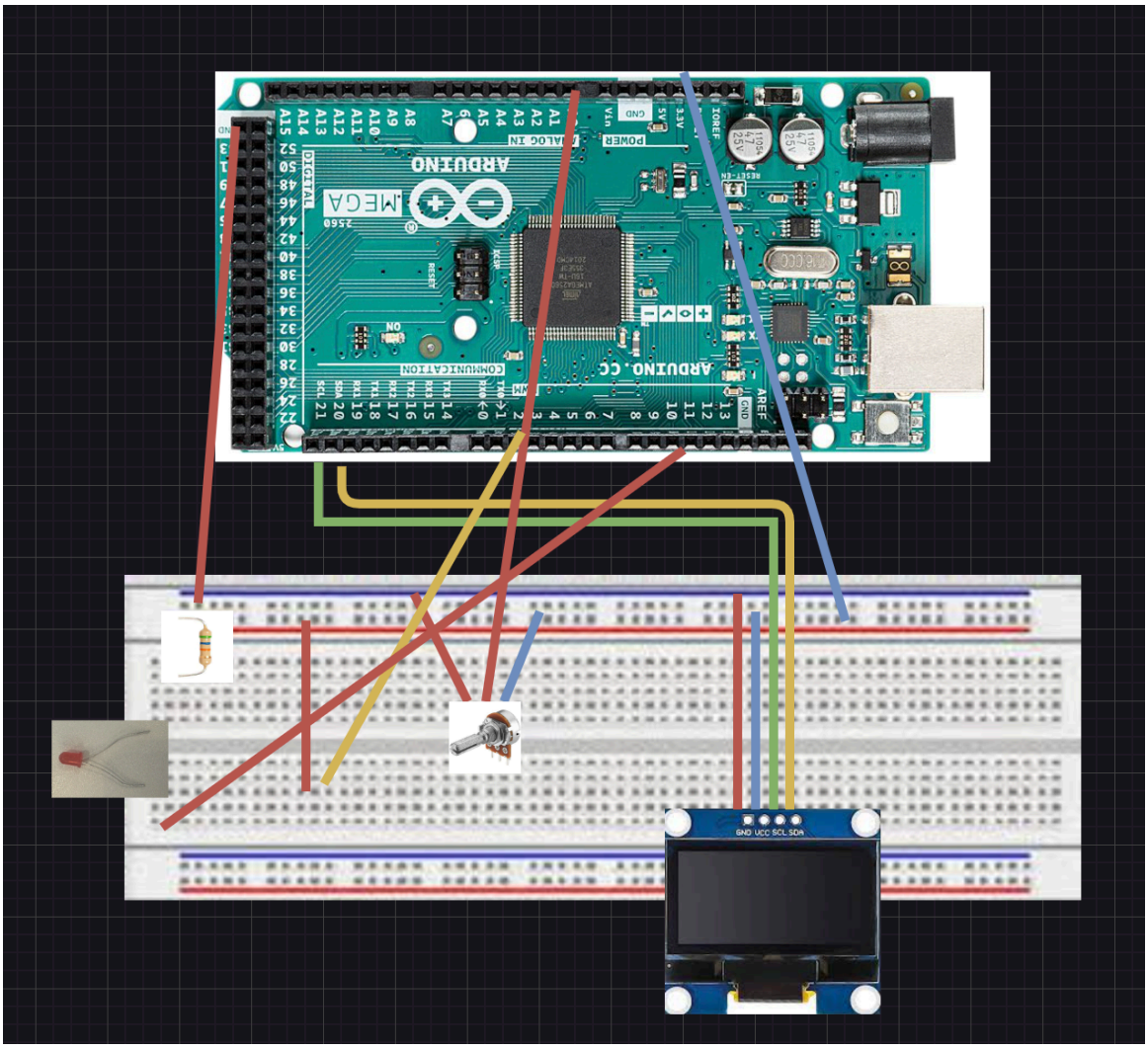
6. Appendix

6.1 Glossary

6.2 Action item list – who did what when

6.2.1 Microcontroller part

6.3 Hardware diagrams



Forklaring

Dette hardware diagram viser, hvordan vores fysiske system er sat op. Der er ledninger fra - til GND og fra + til 5V. Så er OLED-Display forbundet til 5V, GND, SDA og SCL. Potentiometeret er forbundet til A0, GND og 5V. Knappen er forbundet til PIN 2 og GND. Lysdioden er forbundet med det lange ben i PIN 11 og det korte ben til en resistor som er forbundet til GND.

6.4 Source code C

```
/* Purpose: Initialisering og håndtering af ADC på en Arduino platform.
   Input: pwmMax og pwmMin for at justere ADC processering baseret på PWM duty
cycle værdier.
   Output: adcValue - den seneste ADC-læsning. adcReady-flaget sættes til true,
når en ny ADC-værdi er tilgængelig.
   Uses: Standardbiblioteker som <stdint.h>, <Arduino.h>, <stdio.h>, <avr/io.h>,
<stdlib.h>, <util/delay.h>, og <avr/interrupt.h>.
   Author: Benjamin Hadziosmanovic, Ali Al-Sayad, Younes Humadi
   Company: DTU
   Version: 1.0
   Date and year: 4/5 2024
*/

#ifndef ADC_H
#define ADC_H
#include <stdint.h>
#include <Arduino.h>
#include <stdio.h>
#include <avr/io.h>
#include <stdlib.h>
#include <util/delay.h>
#include <avr/interrupt.h>
//globale variabler
extern volatile bool adcReady;
extern volatile uint16_t adcValue;

extern volatile uint8_t pwmMax;  // Maksimal PWM duty cycle, standardværdi
extern volatile uint8_t pwmMin;    // Minimal PWM duty cycle, standardværdi
//Funktioner
void initADC();
void processADC();

#endif

/*
 * This file contains all the static arrays to draw things in the display.
 *you can create your own fonts using this tool https://www.mikroe.com/glcd-
font-creator
 */
#include <avr/pgmspace.h>
typedef uint8_t bitmap_t[8][128];
typedef char PROGMEM prog_uchar;
```

```
// Big numbers font, from 0 to 9. 96 byte each.
const prog_uchar bigNumbers [][96] PROGMEM = {
{0x00, 0x00, 0x00, 0xC0, 0xE0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0, 0xE0, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF,
0xFF, 0xFF, 0x01,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0xFF, 0xFF, 0xFF,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x80, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x07,
0x0F, 0x0F, 0x0F,
0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x07, 0x03,
0x00, 0x00, 0x00},

{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xE0, 0xF0,
0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xFF, 0xFF,
0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x07, 0x07, 0x03, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00},

{0x00, 0x00, 0x00, 0xE0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x81, 0xC1,
0xC0, 0xC0, 0xC0,
0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xE1, 0xFF, 0xFF, 0xFF, 0xFF, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0x87, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0x03,
0x03, 0x03, 0x83, 0x81, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x0F,
0x0F, 0x0F, 0x0F,
0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x07, 0x00,
0x00, 0x00, 0x00},

{0x00, 0x00, 0x00, 0xE0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x81, 0xC1,
0xC0, 0xC0, 0xC0,
0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xE1, 0xFF, 0xFF, 0xFF, 0x7F, 0x00,
0x00, 0x00, 0x00,
```

```
0x00, 0x00, 0x00, 0x81, 0x83, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0x87,
0xFF, 0xFF, 0xFF, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x0F,
0x0F, 0x0F, 0x0F,
0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x07, 0x00,
0x00, 0x00, 0x00},

{0x00, 0x00, 0x00, 0xE0, 0xF0, 0xF0, 0xF0, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x30,
0xF0, 0xF0, 0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF,
0xFF, 0xFF, 0xE0,
0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xE0, 0xFF, 0xFF, 0xFF, 0xFF, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0x07,
0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0C, 0x0F, 0x0F, 0x0F, 0x07, 0x00,
0x00, 0x00, 0x00},

{0x00, 0x00, 0x00, 0xE0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF,
0xFF, 0xFF, 0xE1,
0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC1, 0x81, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x81, 0x83, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0x87,
0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x0F,
0x0F, 0x0F, 0x0F,
0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x07, 0x00,
0x00, 0x00, 0x00},

{0x00, 0x00, 0x00, 0xE0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF,
0xFF, 0xFF, 0xE1,
0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC1, 0x81, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0x87, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0x87,
0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x0F,
0x0F, 0x0F, 0x0F,
0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x07, 0x00,
0x00, 0x00, 0x00},
```

```
{0x00, 0x00, 0x00, 0xE0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x01,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0xFF, 0xFF, 0xFF, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0C, 0x0F, 0x0F, 0x0F, 0x07, 0x00,
0x00, 0x00, 0x00},

{0x00, 0x00, 0x00, 0xE0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7F, 0xFF,
0xFF, 0xFF, 0xE1,
0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xE1, 0xFF, 0xFF, 0xFF, 0x3F, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xFE, 0xFF, 0xFF, 0xFF, 0x87, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0x87,
0xFF, 0xFF, 0xFF, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x0F,
0x0F, 0x0F, 0x0F,
0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x07, 0x00,
0x00, 0x00, 0x00},

{0x00, 0x00, 0x00, 0xE0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0,
0xF0, 0xF0, 0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF,
0xFF, 0xFF, 0xE1,
0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xE1, 0xFF, 0xFF, 0xFF, 0xFF, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
0x03, 0x03, 0x07,
0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0C, 0x0F, 0x0F, 0x0F, 0x07, 0x00,
0x00, 0x00, 0x00},

{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x3C, 0x7E, 0x7E, 0x7E, 0x7E, 0x3C, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0xF8, 0xF8, 0xF8,
0xF8, 0xF0, 0x00,
```

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x01, 0x01, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00}
};

// Big numbers minus symbol.
const prog_uchar  minus [] PROGMEM = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x0C, 0x1E, 0x1E, 0x1E, 0x1E, 0x1E, 0x1E, 0x1E, 0x1E, 0x1E,
0x1E, 0x1E, 0x1E,
0x1E, 0x1E, 0x1E, 0x1E, 0x0C, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00
};

// degrees, outside the ascii table myFont
const prog_uchar  myDregree [8] PROGMEM = {
0x00,0x00,0x0C,0x12,0x12,0x0C,0x00,0x00
};


// Small 8x8 font
const prog_uchar   myFont[][8] PROGMEM = {       //ascii start number 32
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},
{0x00,0x00,0x5F,0x00,0x00,0x00,0x00,0x00},
{0x00,0x00,0x07,0x00,0x07,0x00,0x00,0x00},
{0x00,0x14,0x7F,0x14,0x7F,0x14,0x00,0x00},
{0x00,0x24,0x2A,0x7F,0x2A,0x12,0x00,0x00},
{0x00,0x23,0x13,0x08,0x64,0x62,0x00,0x00},
{0x00,0x36,0x49,0x55,0x22,0x50,0x00,0x00},
{0x00,0x00,0x05,0x03,0x00,0x00,0x00,0x00},
{0x00,0x1C,0x22,0x41,0x00,0x00,0x00,0x00},
{0x00,0x41,0x22,0x1C,0x00,0x00,0x00,0x00},
{0x00,0x08,0x2A,0x1C,0x2A,0x08,0x00,0x00},
{0x00,0x08,0x08,0x3E,0x08,0x08,0x00,0x00},
{0x00,0xA0,0x60,0x00,0x00,0x00,0x00,0x00},
{0x00,0x08,0x08,0x08,0x08,0x08,0x00,0x00},
{0x00,0x60,0x60,0x00,0x00,0x00,0x00,0x00},
{0x00,0x20,0x10,0x08,0x04,0x02,0x00,0x00},
{0x00,0x3E,0x51,0x49,0x45,0x3E,0x00,0x00},
```

```
{0x00,0x00,0x42,0x7F,0x40,0x00,0x00,0x00},
{0x00,0x62,0x51,0x49,0x49,0x46,0x00,0x00},
{0x00,0x22,0x41,0x49,0x49,0x36,0x00,0x00},
{0x00,0x18,0x14,0x12,0x7F,0x10,0x00,0x00},
{0x00,0x27,0x45,0x45,0x45,0x39,0x00,0x00},
{0x00,0x3C,0x4A,0x49,0x49,0x30,0x00,0x00},
{0x00,0x01,0x71,0x09,0x05,0x03,0x00,0x00},
{0x00,0x36,0x49,0x49,0x49,0x36,0x00,0x00},
{0x00,0x06,0x49,0x49,0x29,0x1E,0x00,0x00},
{0x00,0x00,0x36,0x36,0x00,0x00,0x00,0x00},
{0x00,0x00,0xAC,0x6C,0x00,0x00,0x00,0x00},
{0x00,0x08,0x14,0x22,0x41,0x00,0x00,0x00},
{0x00,0x14,0x14,0x14,0x14,0x14,0x00,0x00},
{0x00,0x41,0x22,0x14,0x08,0x00,0x00,0x00},
{0x00,0x02,0x01,0x51,0x09,0x06,0x00,0x00},
{0x00,0x32,0x49,0x79,0x41,0x3E,0x00,0x00},
{0x00,0x7E,0x09,0x09,0x09,0x7E,0x00,0x00},
{0x00,0x7F,0x49,0x49,0x49,0x36,0x00,0x00},
{0x00,0x3E,0x41,0x41,0x41,0x22,0x00,0x00},
{0x00,0x7F,0x41,0x41,0x22,0x1C,0x00,0x00},
{0x00,0x7F,0x49,0x49,0x49,0x41,0x00,0x00},
{0x00,0x7F,0x09,0x09,0x09,0x01,0x00,0x00},
{0x00,0x3E,0x41,0x41,0x51,0x72,0x00,0x00},
{0x00,0x7F,0x08,0x08,0x08,0x7F,0x00,0x00},
{0x00,0x41,0x7F,0x41,0x00,0x00,0x00,0x00},
{0x00,0x20,0x40,0x41,0x3F,0x01,0x00,0x00},
{0x00,0x7F,0x08,0x14,0x22,0x41,0x00,0x00},
{0x00,0x7F,0x40,0x40,0x40,0x40,0x00,0x00},
{0x00,0x7F,0x02,0x0C,0x02,0x7F,0x00,0x00},
{0x00,0x7F,0x04,0x08,0x10,0x7F,0x00,0x00},
{0x00,0x3E,0x41,0x41,0x41,0x3E,0x00,0x00},
{0x00,0x7F,0x09,0x09,0x09,0x06,0x00,0x00},
{0x00,0x3E,0x41,0x51,0x21,0x5E,0x00,0x00},
{0x00,0x7F,0x09,0x19,0x29,0x46,0x00,0x00},
{0x00,0x26,0x49,0x49,0x49,0x32,0x00,0x00},
{0x00,0x01,0x01,0x7F,0x01,0x01,0x00,0x00},
{0x00,0x3F,0x40,0x40,0x40,0x3F,0x00,0x00},
{0x00,0x1F,0x20,0x40,0x20,0x1F,0x00,0x00},
{0x00,0x3F,0x40,0x38,0x40,0x3F,0x00,0x00},
{0x00,0x63,0x14,0x08,0x14,0x63,0x00,0x00},
{0x00,0x03,0x04,0x78,0x04,0x03,0x00,0x00},
{0x00,0x61,0x51,0x49,0x45,0x43,0x00,0x00},
{0x00,0x7F,0x41,0x41,0x00,0x00,0x00,0x00},
{0x00,0x02,0x04,0x08,0x10,0x20,0x00,0x00},
{0x00,0x41,0x41,0x7F,0x00,0x00,0x00,0x00},
{0x00,0x04,0x02,0x01,0x02,0x04,0x00,0x00},
{0x00,0x80,0x80,0x80,0x80,0x80,0x00,0x00},
```

```
{0x00,0x01,0x02,0x04,0x00,0x00,0x00,0x00},
{0x00,0x20,0x54,0x54,0x54,0x78,0x00,0x00},
{0x00,0x7F,0x48,0x44,0x44,0x38,0x00,0x00},
{0x00,0x38,0x44,0x44,0x28,0x00,0x00,0x00},
{0x00,0x38,0x44,0x44,0x48,0x7F,0x00,0x00},
{0x00,0x38,0x54,0x54,0x54,0x18,0x00,0x00},
{0x00,0x08,0x7E,0x09,0x02,0x00,0x00,0x00},
{0x00,0x18,0xA4,0xA4,0xA4,0x7C,0x00,0x00},
{0x00,0x7F,0x08,0x04,0x04,0x78,0x00,0x00},
{0x00,0x00,0x7D,0x00,0x00,0x00,0x00,0x00},
{0x00,0x80,0x84,0x7D,0x00,0x00,0x00,0x00},
{0x00,0x7F,0x10,0x28,0x44,0x00,0x00,0x00},
{0x00,0x41,0x7F,0x40,0x00,0x00,0x00,0x00},
{0x00,0x7C,0x04,0x18,0x04,0x78,0x00,0x00},
{0x00,0x7C,0x08,0x04,0x7C,0x00,0x00,0x00},
{0x00,0x38,0x44,0x44,0x38,0x00,0x00,0x00},
{0x00,0xFC,0x24,0x24,0x18,0x00,0x00,0x00},
{0x00,0x18,0x24,0x24,0xFC,0x00,0x00,0x00},
{0x00,0x00,0x7C,0x08,0x04,0x00,0x00,0x00},
{0x00,0x48,0x54,0x54,0x24,0x00,0x00,0x00},
{0x00,0x04,0x7F,0x44,0x00,0x00,0x00,0x00},
{0x00,0x3C,0x40,0x40,0x7C,0x00,0x00,0x00},
{0x00,0x1C,0x20,0x40,0x20,0x1C,0x00,0x00},
{0x00,0x3C,0x40,0x30,0x40,0x3C,0x00,0x00},
{0x00,0x44,0x28,0x10,0x28,0x44,0x00,0x00},
{0x00,0x1C,0xA0,0xA0,0x7C,0x00,0x00,0x00},
{0x00,0x44,0x64,0x54,0x4C,0x44,0x00,0x00},
{0x00,0x08,0x36,0x41,0x00,0x00,0x00,0x00},
{0x00,0x00,0x7F,0x00,0x00,0x00,0x00,0x00},
{0x00,0x41,0x36,0x08,0x00,0x00,0x00,0x00},
{0x00,0x02,0x01,0x01,0x02,0x01,0x00,0x00},
{0x00,0x02,0x05,0x05,0x02,0x00,0x00,0x00}
};


/*
const uint8_t buffer[SSD1306_LCDHEIGHT * SSD1306_LCDWIDTH / 8] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x80,
    0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
```

```
    0x00, 0x80, 0x80, 0xC0, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x80, 0xC0, 0xE0, 0xF0, 0xF8, 0xFC, 0xF8, 0xE0,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x80, 0x80, 0x80,
    0x80, 0x80, 0x00, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x80,
0x80, 0x80, 0x00, 0xFF}
    #if (SSD1306_LCDHEIGHT * SSD1306_LCDWIDTH > 96*16){
    0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x00, 0x00,
0x80, 0x80, 0x00, 0x00,
    0x80, 0xFF, 0xFF, 0x80, 0x80, 0x00, 0x80, 0x80, 0x00, 0x80, 0x80, 0x80,
0x80, 0x00, 0x80, 0x80,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x00, 0x00, 0x8C, 0x8E, 0x84,
0x00, 0x00, 0x80, 0xF8,
    0xF8, 0xF8, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
0xE0, 0xE0, 0xC0, 0x80,
    0x00, 0xE0, 0xFC, 0xFE, 0xFF, 0xFF, 0xFF, 0x7F, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFE,
0xFF, 0xC7, 0x01, 0x01,
    0x01, 0x01, 0x83, 0xFF, 0xFF, 0x00, 0x00, 0x7C, 0xFE, 0xC7, 0x01, 0x01,
0x01, 0x01, 0x83, 0xFF,
    0xFF, 0xFF, 0x00, 0x38, 0xFE, 0xC7, 0x83, 0x01, 0x01, 0x01, 0x83, 0xC7,
0xFF, 0xFF, 0x00, 0x00,
    0x01, 0xFF, 0xFF, 0x01, 0x01, 0x00, 0xFF, 0xFF, 0x07, 0x01, 0x01, 0x01,
0x00, 0x00, 0x7F, 0xFF,
    0x80, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x7F, 0x00, 0x00, 0xFF, 0xFF, 0xFF,
0x00, 0x00, 0x01, 0xFF,
    0xFF, 0xFF, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x03, 0x0F, 0x3F, 0x7F, 0x7F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xE7, 0xC7, 0xC7, 0x8F,
    0x8F, 0x9F, 0xBF, 0xFF, 0xFF, 0xC3, 0xC0, 0xF0, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFC, 0xFC, 0xFC,
    0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xF8, 0xF8, 0xF0, 0xF0, 0xE0, 0xC0, 0x00,
0x01, 0x03, 0x03, 0x03,
    0x03, 0x03, 0x01, 0x03, 0x03, 0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x03,
0x03, 0x03, 0x01, 0x01,
```

```
    0x03, 0x01, 0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x03, 0x03, 0x01, 0x01,
0x03, 0x03, 0x00, 0x00,
    0x00, 0x03, 0x03, 0x00, 0x00, 0x00, 0x03, 0x03, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x01,
    0x03, 0x03, 0x03, 0x03, 0x03, 0x01, 0x00, 0x00, 0x00, 0x01, 0x03, 0x01,
0x00, 0x00, 0x00, 0x03,
    0x03, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00}
    #if (SSD1306_LCDHEIGHT == 64){
    0x00, 0x00, 0x00, 0x80, 0xC0, 0xE0, 0xF0, 0xF9, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0x3F, 0x1F, 0x0F,
    0x87, 0xC7, 0xF7, 0xFF, 0xFF, 0x1F, 0x1F, 0x3D, 0xFC, 0xF8, 0xF8, 0xF8,
0xF8, 0x7C, 0x7D, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7F, 0x3F, 0x0F, 0x07, 0x00,
0x30, 0x30, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0xFE, 0xFE, 0xFC, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xE0, 0xC0, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x30,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0xC0, 0xFE, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0x7F, 0x7F, 0x3F, 0x1F,
    0x0F, 0x07, 0x1F, 0x7F, 0xFF, 0xFF, 0xF8, 0xF8, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFE, 0xF8, 0xE0,
    0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xFE, 0xFE, 0x00, 0x00,
    0x00, 0xFC, 0xFE, 0xFC, 0x0C, 0x06, 0x06, 0x0E, 0xFC, 0xF8, 0x00, 0x00,
0xF0, 0xF8, 0x1C, 0x0E,
    0x06, 0x06, 0x06, 0x0C, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0xFE, 0xFE, 0x00,
0x00, 0x00, 0x00, 0xFC,
    0xFE, 0xFC, 0x00, 0x18, 0x3C, 0x7E, 0x66, 0xE6, 0xCE, 0x84, 0x00, 0x00,
0x06, 0xFF, 0xFF, 0x06,
    0x06, 0xFC, 0xFE, 0xFC, 0x0C, 0x06, 0x06, 0x06, 0x00, 0x00, 0xFE, 0xFE,
0x00, 0x00, 0xC0, 0xF8,
    0xFC, 0x4E, 0x46, 0x46, 0x46, 0x4E, 0x7C, 0x78, 0x40, 0x18, 0x3C, 0x76,
0xE6, 0xCE, 0xCC, 0x80,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x01, 0x07, 0x0F, 0x1F, 0x1F, 0x3F, 0x3F, 0x3F,
0x3F, 0x1F, 0x0F, 0x03,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x0F, 0x0F, 0x00, 0x00,
```

```
    0x00, 0x0F, 0x0F, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x0F, 0x00, 0x00,
0x03, 0x07, 0x0E, 0x0C,
    0x18, 0x18, 0x0C, 0x06, 0x0F, 0x0F, 0x0F, 0x00, 0x00, 0x01, 0x0F, 0x0E,
0x0C, 0x18, 0x0C, 0x0F,
    0x07, 0x01, 0x00, 0x04, 0x0E, 0x0C, 0x18, 0x0C, 0x0F, 0x07, 0x00, 0x00,
0x00, 0x0F, 0x0F, 0x00,
    0x00, 0x0F, 0x0F, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x0F,
0x00, 0x00, 0x00, 0x07,
    0x07, 0x0C, 0x0C, 0x18, 0x1C, 0x0C, 0x06, 0x06, 0x00, 0x04, 0x0E, 0x0C,
0x18, 0x0C, 0x0F, 0x07,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00}
    #endif
    #endif
};*/




/* Formål: Håndtering af eksterne afbrydelser med debounce-funktionalitet for
AVR mikrocontrollere.
   Input: `uint8_t pin` parameter repræsenterer mikrocontroller pin tilkoblet
til en knap.
   Output: Debounce-funktionen returnerer en `uint8_t` værdi (HIGH eller LOW).
   Brug: Standardbiblioteker som <avr/io.h> og <avr/interrupt.h> anvendes.
   Konstanter som `HIGH`, `LOW`, og `DEBOUNCE_DELAY` er defineret internt.
   Author: Benjamin Hadziosmanovic, Ali Al-Sayad, Younes Humadi
   Company: DTU
   Version: 1.0
   Date and year: 4/5 2024
*/

#ifndef externInt_H_
```

```
#define externInt_H_
#ifndef HIGH
#define HIGH 0x1
#endif
#include <avr/io.h>
#include <avr/interrupt.h>
#define DEBOUNCE_DELAY 50
//Funktioner
uint8_t debounce(uint8_t pin);
void enable_interrupts();
void disable_interrupts();
void initButtonInterrupt();

#endif /* externInt_H_ */
#ifndef LOW
#define LOW 0x0
#endif
/**
 * I2C.h
 * driver for I2C from AVR freaks adjusted with an init function
 * Created: 22-12-2017 19:00:53
 *  Author: osch
 */


#ifndef I2C_H_
#define I2C_H_
#define SCL_CLK 100000
#define F_CPU 16000000UL

char write_address;

#include <util/delay.h>

#define BITRATE(TWSR)    ((F_CPU/SCL_CLK)-
16)/(2*pow(4,(TWSR&((1<<TWPS0)|(1<<TWPS1)))))
char read_addres;

void I2C_Init() ;
uint8_t I2C_Start(char write_address);/* I2C start function */
uint8_t I2C_Repeated_Start(char read_address); /* I2C repeated start function */
uint8_t I2C_Write(char data);    /* I2C write function */
char I2C_Read_Ack() ;    /* I2C read ack function */
char I2C_Read_Nack();        /* I2C read nack function */
void I2C_Stop() ;        /* I2C stop function */
#endif /* I2C_H_ */
/*
```

```
Formål: Dette modul håndterer PWM-funktionaliteter på AVR mikrocontrollere,
 med fokus på LED-belysning og motorstyring. Det understøtter dynamisk justering
af PWM via UART og OLED.

Input:
- uint8_t dutyCycle: Styrer PWM-outputtet.

Output:
- Modulet styrer PWM-outputs og kommunikation via UART/OLED.

Bruger:
- <stdint.h>, <Arduino.h>, <stdio.h>, <avr/io.h>, <stdlib.h>, <util/delay.h>,
<avr/interrupt.h> til funktioner og hardwareinteraktion.

Author: Benjamin Hadziosmanovic, Ali Al-Sayad, Younes Humadi
   Company: DTU
   Version: 1.0
   Date and year: 4/5 2024
*/
#ifndef PWM_H
#define PWM_H
#include <stdint.h>
#include <Arduino.h>
#include <stdio.h>
#include <avr/io.h>
#include <stdlib.h>
#include <util/delay.h>
#include <avr/interrupt.h>

//Globale Variabler
#define LED_PIN PB5  // Definerer pin 11 som PWM-kompatibel
#define BUTTON_PIN PE4 // Definerer pin-2 for knappen som PE4

//Funktioner
void setupPWM();
void setPWMDutyCycle(uint8_t dutyCycle);
void initUARTAndOLED();
void readPWMSettings();
void askPWMValues();
#endif
/*
 * ssd1306.h
 *based upon Adafruit_SSD1306 at github  -adjusted to C using i2c address 0x78
for writing and 0x7A for reading
 * Created: 03-01-2018 17:33:51
 *  Author: osch
```

```
  *page refers to:  SSD1306 Advanced information Matrix apr. 2008  rev. 1.1
www.solomon-systech.com describes all used commands and data used for the
display 128x64 dot matrix
  *p reference to this documentation
  *initialization is from the DD-2864bY-3A rev c p 19
  *both data sheets are at git hub in this project
 */

#include <stdbool.h>

  #define SSD1306_128_64
//#define SSD1306_128_32
//   #define SSD1306_96_16
#if defined SSD1306_128_64 && defined SSD1306_128_32
#error "Only one SSD1306 display can be specified at once in SSD1306.h"
#endif
#if !defined SSD1306_128_64 && !defined SSD1306_128_32 && !defined SSD1306_96_16
#error "At least one SSD1306 display must be specified in SSD1306.h"
#endif

#if defined SSD1306_128_64
#define SSD1306_LCDWIDTH                  128
#define SSD1306_LCDHEIGHT                 64
#endif
#if defined SSD1306_128_32
#define SSD1306_LCDWIDTH                  128
#define SSD1306_LCDHEIGHT                 32
#endif
#if defined SSD1306_96_16
#define SSD1306_LCDWIDTH                  96
#define SSD1306_LCDHEIGHT                 16
#endif

// #define pgm_read_byte(addr) (*(const unsigned char *)(addr))
//command data defined p 28 - 32
#define SSD1306_LCDWIDTH      128
#define SSD1306_LCDHEIGHT      64
#define SSD1306_SETCONTRAST   0x81
#define SSD1306_DISPLAYALLON_RESUME 0xA4
#define SSD1306_DISPLAYALLON 0xA5
#define SSD1306_NORMALDISPLAY 0xA6
#define SSD1306_INVERTDISPLAY 0xA7
#define SSD1306_DISPLAYOFF 0xAE
#define SSD1306_DISPLAYON 0xAF
#define SSD1306_SETDISPLAYOFFSET 0xD3
#define SSD1306_SETCOMPINS 0xDA
#define SSD1306_SETVCOMDETECT 0xDB
```

```
#define SSD1306_SETDISPLAYCLOCKDIV 0xD5
#define SSD1306_SETPRECHARGE 0xD9
#define SSD1306_SETMULTIPLEX 0xA8
#define SSD1306_SETLOWCOLUMN 0x00
#define SSD1306_SETHIGHCOLUMN 0x10
#define SSD1306_SETSTARTLINE 0x40
#define SSD1306_MEMORYMODE 0x20
#define SSD1306_COLUMNADDR 0x21
#define SSD1306_PAGEADDR   0x22
#define SSD1306_COMSCANINC 0xC0
#define SSD1306_COMSCANDEC 0xC8
#define SSD1306_SEGREMAP 0xA0
#define SSD1306_CHARGEPUMP 0x8D
#define SSD1306_EXTERNALVCC 0x1
#define SSD1306_SWITCHCAPVCC 0x2
// Scrolling #defines
#define SSD1306_ACTIVATE_SCROLL 0x2F
#define SSD1306_DEACTIVATE_SCROLL 0x2E
#define SSD1306_SET_VERTICAL_SCROLL_AREA 0xA3
#define SSD1306_RIGHT_HORIZONTAL_SCROLL 0x26
#define SSD1306_LEFT_HORIZONTAL_SCROLL 0x27
#define SSD1306_VERTICAL_AND_RIGHT_HORIZONTAL_SCROLL 0x29
#define SSD1306_VERTICAL_AND_LEFT_HORIZONTAL_SCROLL 0x2A

#define BLACK 0
#define WHITE 1
#define INVERSE 2

typedef uint8_t bitmap_t[8][128];
uint8_t _i2c_address;
void  InitializeDisplay();
void sendStrXY( char *string, int X, int Y);
void sendStr( char *string);
void setXY(unsigned char row,unsigned char col);
void sendCharXY(unsigned char data, int X, int Y);
void SendChar(unsigned char data);
void displayOn(void);
void displayOff(void);
void clear_display(void);
void printBigTime(char *string);
void reset_display(void);
void printBigNumber(char string, int X, int Y);
void bmp(bitmap_t b);
void setPageAddress();
void setColAddress();
void ssd1306_setpos(uint8_t x, uint8_t y);
```

```
void ssd1306_draw_bmp(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1, const
uint8_t bitmap[]);
//scroll
void startscrollright(uint8_t start, uint8_t stop);
void startscrollleft(uint8_t start, uint8_t stop);

void startscrolldiagright(uint8_t start, uint8_t stop);
void startscrolldiagleft(uint8_t start, uint8_t stop);
void stopscroll(void);
void dim(bool dim);
void print_fonts();
void drawPixel(int16_t x, int16_t y, uint16_t color);


/*
Formål: Modulerer håndtering af Timer 0 på AVR
mikrocontrollere til at skabe præcise tidsintervaller og håndtere knaptryk.

Input: Ingen direkte input parametre, men modulet
reagerer på systemtilstande og hardwarehændelser.

Output: Påvirker mikrocontrollerens adfærd via
timingsekvenser, herunder indstilling af flag og udløsning af rutiner via
interrupts.

Bruger: Anvender <Arduino.h>, <stdio.h>, <avr/io.h>, <stdlib.h>, <util/delay.h>,
<avr/interrupt.h> for funktionaliteter som datatyper, I/O operationer, og
interrupt håndtering.

Author: Benjamin Hadziosmanovic, Ali Al-Sayad, Younes Humadi
   Company: DTU
   Version: 1.0
   Date and year: 4/5 2024
*/
#ifndef TIMER_H
#define TIMER_H
#include <Arduino.h>
#include <stdio.h>
#include <avr/io.h>
#include <stdlib.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/io.h>
//Globale variabler
extern volatile uint8_t button_pressed; // Deklarerer en variabel for at angive
om knappen er blevet trykket
//Funktioner
```

```
void initTimer0();

#endif

/*
Formål: Modul til at implementere UART-kommunikation på AVR mikrocontrollere,
tilpasset til en specifik baudrate og CPU-frekvens. Det tilbyder grundlæggende
funktioner til at initialisere UART, sende og modtage tegn og strengedata.

Input:
- unsigned int ubrr: Baud rate register værdi for initialisering af UART.
- char data, const char *str, char *str, uint16_t maxLen: Parametre til
transmission og modtagelse af data.

Output:
- Funktionerne styrer dataoverførsel via UART og returnerer data som modtages
fra UART.

Bruger:
- <stdint.h> for standard integer definitioner.
- Konstanter som BAUD, F_CPU, og MYUBRRF for at definere
kommunikationshastigheder og systemkonfigurationer.

Author: Benjamin Hadziosmanovic, Ali Al-Sayad, Younes Humadi
   Company: DTU
   Version: 1.0
   Date and year: 4/5 2024
*/
#define BAUD 19200
#define F_CPU 16000000UL
#define MYUBRRF F_CPU/8/BAUD-1
#include <stdint.h>
//Funktioner
extern void uart0_Init(unsigned int ubrr);
extern char getchUSART0(void);
extern void putchUSART0(char tx);
extern void putsUSART0(char *ptr);
// Funktion til at sende et enkelt tegn via UART
void uart_transmit(char data);

// Funktion til at modtage et enkelt tegn via UART
char uart_receive(void);

// Funktion til at sende en streng via UART
void uart_sendString(const char *str);
```

```
// Funktion til at læse en streng via UART, med en maksimal længde angivet af
maxLen
void uart_readString(char *str, uint16_t maxLen);

/*
Formål: Modul til at styre ADC (Analog-to-Digital Converter)
og PWM (Pulse Width Modulation) operationer på AVR mikrocontrollere.
Det indsamler analoge data, omdanner dem til digitale værdier, beregner
 spændingsværdier og justerer PWM duty cycle baseret på disse målinger. Dataene
sendes også til UART for visning.

Input:
- Ingen eksterne inputs kræves direkte af funktionerne,
men ADC konfigurationerne afhænger af systemets hardwareopsætning og tilkoblede
sensorer.

Output:
- ADC data behandles til at styre PWM-output og sende formateret tekst via UART
til eksterne enheder eller displayenheder.

Bruger:
- Modulerne 'adc.h', 'UART.h', og 'pwm.h' er nødvendige for at implementere
funktionaliteterne korrekt.
- Benytter standardbibliotekerne som <avr/io.h>, <Arduino.h>, <stdio.h>,
<stdlib.h>, og <util/delay.h> for at interagere med hardwaren og foretage
nødvendige beregninger og timing.

Author: Benjamin Hadziosmanovic, Ali Al-Sayad, Younes Humadi
   Company: DTU
   Version: 1.0
   Date and year: 4/5 2024
*/
#include "adc.h"
#include <avr/io.h>
#include <Arduino.h>
#include <stdio.h>
#include <avr/io.h>
#include <stdlib.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "UART.h"
#include "pwm.h"

// Initialiserer globale variabler for ADC-klar og ADC-værdi med standardværdier
volatile bool adcReady = false;
volatile uint16_t adcValue = 0;
```

```
// Initialiserer globale variabler for maksimal og minimal PWM duty cycle med
standardværdier
volatile uint8_t pwmMax = 255;  // Maksimal PWM duty cycle, standardværdi
volatile uint8_t pwmMin = 0;    // Minimal PWM duty cycle, standardværdi

// Initialiser ADC med de rette indstillinger for spændingsreference og
justering
void initADC() {
    ADMUX = (1 << REFS0) | (0 << ADLAR); // Brug AVCC som reference, juster
resultat til højre
    ADCSRA = (1 << ADEN) | (1 << ADIE) | (7 << ADPS0); // Aktivér ADC og
interrupt, sæt prescaler til 128
    ADCSRB = (0 << ADTS0) | (1 << ADTS1) | (1 << ADTS2); // Auto Trigger Source:
Timer0 Compare Match A
}

// Behandler ADC-værdien og udfører passende handlinger
void processADC() {
    // ADC processing code
    float voltage = adcValue * 5.0 / 1023.0;
    int integerPart = (int)voltage;
    int decimalPart = (int)((voltage - integerPart) * 100);

    uint8_t pwmDuty = (uint8_t)((voltage / 5.0) * 255);
    pwmDuty = (pwmDuty < pwmMin) ? pwmMin : (pwmDuty > pwmMax) ? pwmMax :
pwmDuty;
    setPWMDutyCycle(pwmDuty);

    char buffer[50];
    sprintf(buffer, "Voltage: %d.%02d V, PWM DC: %03d%%\n", integerPart,
decimalPart, (int)((pwmDuty / 255.0) * 100));
    putsUSART0(buffer);
    sendStrXY(buffer, 3, 2);  // Assuming sendStrXY sends data to OLED
}

// Interrupt service rutine for når en ADC læsning er klar
ISR(ADC_vect) {
    adcValue = ADC;  // Læs ADC værdien
    adcReady = true;  // Sæt klar flaget
}

/*
Formål: Modulet håndterer debounce-logik for knapper
og konfiguration af eksterne interrupts til at reagere
på knappetryk. Det er designet til at stabilisere input
fra mekaniske knapper og forenkle håndteringen af hardwarebaserede
brugerinteraktioner.
```

```
Input:
- uint8_t pin: Pinnummeret, som debouncing og interrupt-konfigurationen
anvender.

Output:
- debounce-funktionen returnerer knaptilstanden efter anvendelse af debounce-
logik.
- Interrupts aktiveres eller deaktiveres via enable_interrupts() og
disable_interrupts().

Bruger:
- "externInt.h" for funktionsprototyper og konstantdefinitioner.
- Standardbiblioteker som <avr/io.h> og <util/delay.h> bruges til direkte
hardwarekontrol og forsinkelser.

Author: Benjamin Hadziosmanovic, Ali Al-Sayad, Younes Humadi
   Company: DTU
   Version: 1.0
   Date and year: 4/5 2024
*/
#include "externInt.h"
#define BUTTON_PIN PE4 // Definerer pin-nummeret for knappen som PE4
#include <avr/io.h>
#include <util/delay.h>

#define DEBOUNCE_DELAY 50

// Funktion til at udføre debouncing på en given pin
uint8_t debounce(uint8_t pin) {
    static uint8_t lastState = 1; // Antager pull-up modstand, bruger direkte 1
i stedet for HIGH
    uint8_t currentState = PINE & (1 << pin);

    if (currentState != lastState) {
        _delay_ms(DEBOUNCE_DELAY); // Debounce-forsinkelse
        currentState = PINE & (1 << pin); // Genlæs pin-tilstanden efter
forsinkelsen
    }

    lastState = currentState;
    return currentState ? 0 : 1; // Returnerer 0 hvis høj, 1 hvis lav
(inverteret på grund af pull-up)
}

// Funktion til at aktivere interrupts
void enable_interrupts() {
```

```
    // Aktiverer eksterne interrupts
    sei();
}

// Funktion til at deaktivere interrupts
void disable_interrupts() {
    // Deaktiverer eksterne interrupts
    cli();
}

// Konfigurer eksternt interrupt for BUTTON_PIN
void initButtonInterrupt() {
    DDRB &= ~(1 << BUTTON_PIN);  // Sætter knappens pin som input
    PORTE |= (1 << BUTTON_PIN);  // Aktiverer pull-up modstand

    EICRB |= (1 << ISC41);       // Udløs på faldende kant
    EIMSK |= (1 << INT4);        // Aktiver INT4
}

/*
 * I2C.c
 *
 * Created: 22-12-2017 19:00:37
 *  Author: osc
 *sda goes to PIN 21 and the sck goes to PIN 20*
 */
#include "I2C.h"
#include <avr/io.h>
/**init for I2C scl set to 100000 kHz*/
void I2C_Init()         /* I2C initialize function */
{
     DDRA|=(1<<DDA0);
     PORTA|=(1<<PA0);
     _delay_ms(1000);
    TWBR=18;
    TWSR&=0xFC;
    TWCR=0x05;
}
/** I2C start function
* Return 0 to indicate start condition fail
* Return 1 to indicate ack received
* Return 2 to indicate nack received*/
uint8_t I2C_Start(char write_address)
{
    uint8_t status;     /* Declare variable */
    TWCR=(1<<TWSTA)|(1<<TWEN)|(1<<TWINT);;// /* Enable TWI, generate START */
    while(!(TWCR&(1<<TWINT)));  /* Wait until TWI finish its current job */
```

```
    status=TWSR&0xF8;       /* Read TWI status register */
    if(status!=0x08)        /* Check weather START transmitted or not? */
    return 0;           /* Return 0 to indicate start condition fail */
    TWDR=write_address;     /* Write SLA+W in TWI data register */
    TWCR=(1<<TWEN)|(1<<TWINT);  /* Enable TWI & clear interrupt flag */
    while(!(TWCR&(1<<TWINT)));  /* Wait until TWI finish its current job */
    status=TWSR&0xF8;       /* Read TWI status register */
    if(status==0x18)        /* Check for SLA+W transmitted &ack received */
    return 1;           /* Return 1 to indicate ack received */
    if(status==0x20){       /* Check for SLA+W transmitted &nack received */

    return 2;           /* Return 2 to indicate nack received */

}
    else
    return 3;           /* Else return 3 to indicate SLA+W failed */
}
 /** I2C repeated start function */
uint8_t I2C_Repeated_Start(char read_address)
{
    uint8_t status;     /* Declare variable */
    TWCR=(1<<TWSTA)|(1<<TWEN)|(1<<TWINT);/* Enable TWI, generate start */
    while(!(TWCR&(1<<TWINT)));  /* Wait until TWI finish its current job */
    status=TWSR&0xF8;       /* Read TWI status register */
    if(status!=0x10)        /* Check for repeated start transmitted */
    return 0;           /* Return 0 for repeated start condition fail */
    TWDR=read_address;      /* Write SLA+R in TWI data register */
    TWCR=(1<<TWEN)|(1<<TWINT);  /* Enable TWI and clear interrupt flag */
    while(!(TWCR&(1<<TWINT)));  /* Wait until TWI finish its current job */
    status=TWSR&0xF8;       /* Read TWI status register */
    if(status==0x40)        /* Check for SLA+R transmitted &ack received */
    return 1;           /* Return 1 to indicate ack received */
    if(status==0x20)        /* Check for SLA+R transmitted &nack received */
    return 2;           /* Return 2 to indicate nack received */
    else
    return 3;           /* Else return 3 to indicate SLA+W failed */
}
uint8_t I2C_Write(char data)    /* I2C write function */
{
    uint8_t status;     /* Declare variable */
    TWDR=data;          /* Copy data in TWI data register */
    TWCR=(1<<TWEN)|(1<<TWINT);  /* Enable TWI and clear interrupt flag */
    while(!(TWCR&(1<<TWINT)));  /* Wait until TWI finish its current job */
    status=TWSR&0xF8;       /* Read TWI status register */
    if(status==0x28)        /* Check for data transmitted &ack received */
    return 0;           /* Return 0 to indicate ack received */
    if(status==0x30)        /* Check for data transmitted &nack received */
```

```
    return 1;           /* Return 1 to indicate nack received */
    else
    return 2;           /* Else return 2 for data transmission failure */
}
char I2C_Read_Ack()     /* I2C read ack function */
{
    TWCR=(1<<TWEN)|(1<<TWINT)|(1<<TWEA); /* Enable TWI, generation of ack */
    while(!(TWCR&(1<<TWINT)));  /* Wait until TWI finish its current job */
    return TWDR;            /* Return received data */
}
char I2C_Read_Nack()        /* I2C read nack function */
{
    TWCR=(1<<TWEN)|(1<<TWINT);  /* Enable TWI and clear interrupt flag */
    while(!(TWCR&(1<<TWINT)));  /* Wait until TWI finish its current job */
    return TWDR;        /* Return received data */
}

void I2C_Stop()         /* I2C stop function */
{
    TWCR=(1<<TWSTO)|(1<<TWINT)|(1<<TWEN);/* Enable TWI, generate stop */
    while(TWCR&(1<<TWSTO)); /* Wait until stop condition execution */
}
/*
Formål: Hovedprogrammet koordinerer initialisering og løbende styring af flere
hardwaremoduler i et AVR-baseret system. Det håndterer brugerinput via UART,
justerer PWM-indstillinger, og behandler ADC-læsninger.

Input:
- Brugeren indtaster data via UART for at justere PWM-indstillinger.
- ADC-data indsamles automatisk når de er klar.

Output:
- Systemet justerer PWM-output baseret på brugerinput.
- Viser status og fejlmeddelelser via UART.

Bruger:
- Inkluderer moduler som "UART.h", "ssd1306.h", "I2C.h", "pwm.h", "adc.h",
"timer.h", og "externInt.h" for at håndtere forskellige funktioner i systemet.
- Anvender <Arduino.h>, <stdio.h>, <avr/io.h>, <stdlib.h>, <util/delay.h>,
<avr/interrupt.h> for integration og interaktion med AVR-hardwaren.

Funktioner:
- initUARTAndOLED(), uart0_Init(), initADC(), setupPWM(), initTimer0(),
initButtonInterrupt(): Initialiserer de forskellige systemkomponenter.
- sei(): Aktiverer globale interrupts for at tillade interrupt-drevet handling.
- Hovedløkken behandler knaptryk for at modtage og validere brugerinput og
ajourfører systemindstillinger baseret på dette.
```

```
Author: Benjamin Hadziosmanovic, Ali Al-Sayad, Younes Humadi
   Company: DTU
   Version: 1.0
   Date and year: 4/5 2024
*/
#include <Arduino.h>
#include <stdio.h>
#include <avr/io.h>
#include <stdlib.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "UART.h"
#include "ssd1306.h"
#include "I2C.h"
#include "pwm.h"
#include "adc.h"
#include "timer.h"
#include "externInt.h"

#define LED_PIN PB5  // Definerer pin 11 som PWM-kompatibel
#define BUTTON_PIN PE4 // Definerer pin-2 for knappen som PE4

int main(void) {
    initUARTAndOLED();
    askPWMValues();
    uart0_Init(19200);
    initADC();
    setupPWM();
    initTimer0();
    initButtonInterrupt();
    sei();  // Enable global interrupts
    char buffer[100];  // Buffer to hold strings for output

    while (1) {
        if (button_pressed) {
            int newMax, newMin;
            while (1) {
                uart_sendString("Enter new max-min PWM values (format 'max-
min'):\n");
                char pwmInput[15];
                uart_readString(pwmInput, sizeof(pwmInput));
                int parsedItems = sscanf(pwmInput, "%d-%d", &newMax, &newMin);

                if (parsedItems != 2) {
                    uart_sendString("Error: Invalid input format. Please ensure
the format is 'max-min'.\n");
```

```
                    continue;
                }

                if (newMax > 255 || newMin > 255 || newMax < 0 || newMin < 0) {
                    uart_sendString("Error: Values must be between 0 and
255.\n");
                    continue;
                }

                if (newMin > newMax) {
                    uart_sendString("Error: Minimum value cannot be greater than
maximum value.\n");
                    continue;
                }

                break;  // Ud af loop når rigtige værdier er indtastet
            }
            pwmMax = newMax;
            pwmMin = newMin;
            sprintf(buffer, "New settings: Max = %d, Min = %d\n", pwmMax,
pwmMin);
            uart_sendString(buffer);

            // Clear the button pressed flag
            button_pressed = 0;
        }

        if (adcReady) {
            processADC();
        }
    }
}

/*
Formål: Implementerer styring af PWM for LED-belysning og kommunikation via
UART. Modulerne koordinerer input fra brugeren til at justere PWM-indstillinger
og visualiserer oplysninger på et OLED-display.

Input:
- Brugeren angiver maksimale og minimale PWM duty cycle værdier via UART.

Output:
- Justerer LEDens lysstyrke baseret på PWM-værdier.
- Feedback til brugeren om de indtastede og behandlede værdier via UART og OLED.

Bruger:
```

```
- "pwm.h", "UART.h", "I2C.h", "ssd1306.h", og "adc.h" for at implementere de
nødvendige funktioner for modulinteraktion.
- Standardbiblioteker som <stdio.h> og <avr/io.h> til datahåndtering og
hardwarekontrol.

Funktioner:
- setupPWM(): Konfigurerer PWM-hardware til LED-styring.
- setPWMDutyCycle(): Justerer PWM duty cycle baseret på brugerinput.
- askPWMValues(): Anmoder om og validerer PWM-værdier fra brugeren.
- initUARTAndOLED(): Initialiserer UART for kommunikation og OLED for display-
output.

Author: Benjamin Hadziosmanovic, Ali Al-Sayad, Younes Humadi
   Company: DTU
   Version: 1.0
   Date and year: 4/5 2024
*/
#include "pwm.h"
#include <stdio.h>
#include <avr/io.h>
#include "ssd1306.h"
#include "UART.h"
#include "I2C.h"
#include "adc.h"

#define LED_PIN PB5  // Bruger pin 11, som er i stand til PWM

// Opsæt PWM-hardware til LED-kontrol
void setupPWM() {
    DDRB |= (1 << LED_PIN); // Sæt LED-pin som output
    TCCR1A |= (1 << COM1A1) | (1 << WGM10);
    TCCR1B |= (1 << WGM12) | (1 << CS11);
}

// Sæt PWM duty cycle
void setPWMDutyCycle(uint8_t dutyCycle) {
    OCR1A = dutyCycle; // Sæt PWM duty cycle
}

// Spørg brugeren om PWM-værdier
void askPWMValues() {
    char inputBuffer[15];  // Tilstrækkelig stor buffer til input
    char buffer[50];       // Til feedback via UART
    int tempMax, tempMin;
    int parsedItems;

    while (1) {  // Fortsæt indtil gyldige og logiske værdier er modtaget
```

```
        uart_sendString("Indtast maksimal og minimal PWM duty cycle (format
'max-min'):\n");
        uart_readString(inputBuffer, sizeof(inputBuffer));  // Læs input

        // Debug: Vis modtaget input
        sprintf(buffer, "Modtaget input: '%s'\n", inputBuffer);
        uart_sendString(buffer);

        // Parse input
        parsedItems = sscanf(inputBuffer, "%d-%d", &tempMax, &tempMin);

        // Debug: Vis parsed værdier
        sprintf(buffer, "Parsed max: %d, min: %d\n", tempMax, tempMin);
        uart_sendString(buffer);

        // Tjek parsing og værdiernes gyldighed
        if (parsedItems != 2) {
            uart_sendString("Fejl: Venligst sikre at formatet er 'max-min'.\n");
            continue;
        }

        // Tjek om værdierne er inden for det tilladte interval
        if (tempMax > 255 || tempMin > 255 || tempMax < 0 || tempMin < 0) {
            uart_sendString("Fejl: Værdier skal være mellem 0 og 255.\n");
            continue;  // Fortsætter med at bede om input
        }

        // Tjek om minimumsværdien er større end maksimumsværdien
        if (tempMin > tempMax) {
            uart_sendString("Fejl: Minimumsværdi kan ikke være større end
maksimumsværdi.\n");
            continue;  // Fortsætter med at bede om input
        }

        break;  // Bryder løkken, når gyldige og logiske værdier er modtaget
    }

    // Opdater globale variabler efter gyldige værdier er modtaget
    pwmMax = (uint8_t)tempMax;
    pwmMin = (uint8_t)tempMin;

    // Bekræft indstillede værdier til brugeren
    sprintf(buffer, "Sæt max: %d, min: %d\n", pwmMax, pwmMin);
    uart_sendString(buffer);
}

// Initialiser UART og OLED
```

```
void initUARTAndOLED() {
    uart0_Init(19200); // Indstil baudrate eksplisit
    I2C_Init();
    InitializeDisplay();
    clear_display();
}

/**
 * ssd1306.c
 *
 * Created: 03-01-2018 17:33:26
 *  Author: osc
 *this modules purpose is to initiate and control the OLED display SSD1306
driver for the 128x64 OLED display
 *The plus can be connected to PIN 24 and the GND to PIN 26 and sda goes to PIN
21 and the sck goes to PIN 20
 *http://microcontrolandos.blogspot.dk/2014/12/pic-ssd1306.html
 * SSD1306 Advanced information Matrix apr. 2008  rev. 1.1 www.solomon-
systech.com describes all used commands and data used for the display 128x64 dot
matrix
 *for Mega2560 arduino board
 *p reference to this documentation
 *initialization is from the DD-2864bY-3A rev c p 19
 *both data sheets are at git hub in this project
 */
#include <math.h>
#include <string.h>
#include "I2C.h"
#define F_CPU 16000000UL
#include <util/delay.h>
#include <avr/pgmspace.h>
#include "ssd1306.h"
#include "data.h"
#define ssd1306_swap(a, b) { int16_t t = a; a = b; b = t; }
#define _vccstate 1  //externalVcc

uint8_t _i2c_address=0x78;    //display write address

/**write a command to the ssd1306*/
void  ssd1306_command(uint8_t c)
{
    uint8_t control = 0x00; // some use 0X00 other examples use 0X80. I tried
both
    I2C_Start(_i2c_address);
    //I2C_Write();
    I2C_Write(control); // This is Command
    I2C_Write(c);
```

```
    I2C_Stop();
}
////////////////////////////////////////////
//
/**write a a data byte to the ssd1306*/
void  ssd1306_data(uint8_t c)
{
    I2C_Start(_i2c_address);
    I2C_Write(_i2c_address);
    I2C_Write(0X40); // This byte is DATA
    I2C_Write(c);
    I2C_Stop();
}
////////////////////////////////////////////////////
/** Used when doing Horizontal or Vertical Addressing*/
void setColAddress()
{
    ssd1306_command(SSD1306_COLUMNADDR); // 0x21 COMMAND
    ssd1306_command(0); // Column start address
    ssd1306_command(SSD1306_LCDWIDTH-1); // Column end address
}
////////////////////////////////////////////////////
/** Used when doing Horizontal or Vertical Addressing*/
void setPageAddress()
{
    ssd1306_command(SSD1306_PAGEADDR); // 0x22 COMMAND
    ssd1306_command(0); // Start Page address
    ssd1306_command((SSD1306_LCDHEIGHT/8)-1);// End Page address
}
////////////////////////////////////////////////////////////////////
/** init according to SSD1306 data sheet and using the plus can be connected to
PIN 24 and the GND to PIN 26 */

void  InitializeDisplay()
{



    // Init sequence for 128x64 OLED module
    ssd1306_command(SSD1306_DISPLAYOFF);                    // 0xAE

    ssd1306_command(SSD1306_SETDISPLAYCLOCKDIV);            // 0xD5
    ssd1306_command(0x80);                  // the suggested ratio 0x80

    ssd1306_command(SSD1306_SETMULTIPLEX);                  // 0xA8
    ssd1306_command(0x3F);
```

```
    ssd1306_command(SSD1306_SETDISPLAYOFFSET);              // 0xD3
    ssd1306_command(0x0);                                   // no offset

    ssd1306_command(SSD1306_SETSTARTLINE);// | 0x0);        // line #0

    ssd1306_command(SSD1306_CHARGEPUMP);                    // 0x8D
    ssd1306_command(0x14);  // using internal VCC

    ssd1306_command(SSD1306_MEMORYMODE);                    // 0x20
    ssd1306_command(0x00);                          // 0x00 horizontal addressing
automatic line shift

    ssd1306_command(SSD1306_SEGREMAP | 0x1); // rotate screen 180

    ssd1306_command(SSD1306_COMSCANDEC); // rotate screen 180

    ssd1306_command(SSD1306_SETCOMPINS);                    // 0xDA
    ssd1306_command(0x12);

    ssd1306_command(SSD1306_SETCONTRAST);                   // 0x81
    ssd1306_command(0xCF);

    ssd1306_command(SSD1306_SETPRECHARGE);                  // 0xd9
    ssd1306_command(0xF1);

    ssd1306_command(SSD1306_SETVCOMDETECT);                 // 0xDB
    ssd1306_command(0x40);

    ssd1306_command(SSD1306_DISPLAYALLON_RESUME);           // 0xA4

    ssd1306_command(SSD1306_NORMALDISPLAY);                 // 0xA6

    ssd1306_command(SSD1306_DISPLAYON);                     //switch on OLED

}

/** reset the display*/
void reset_display(void)
{
    displayOff();

    clear_display();


    displayOn();
}
```

```
//============================================================//
/** Turns display on.*/
void displayOn(void)
{
    ssd1306_command(0xaf);        //display on p. 28
}

//============================================================//
/** Turns display off.*/
void displayOff(void)
{
    ssd1306_command(0xae);      //display off p. 28
}

//============================================================//
/** Clears the display by sending 0 to all the screen map.*/
void clear_display(void)
{
    unsigned char i,k;
    for(k=0;k<8;k++)
    {
        setXY(k,0);
        {
            for(i=0;i<128;i++)     //clear all COL
            {
                SendChar(0);         //clear all COL
                //delay(10);
            }
        }
    }
}




//============================================================//
/**print integer string in big font*/
void printBigTime(char *string)
{

    int Y=0;
```

```
    int lon = strlen(string);
    if(lon == 3) {
        Y = 0;
        } else if (lon == 2) {
        Y = 3;
        } else if (lon == 1) {
        Y = 6;
    }

    int X = 4;
    while(*string)
    {
        printBigNumber(*string, X, Y);

        Y+=3;
        X=4;
        setXY(X,Y);
        string++;
    }
}


//============================================================//
/** Prints a display big number (96 bytes) in coordinates X Y,
* being multiples of 8. This means we have 16 COLS (0-15)
* and 8 ROWS (0-7).*/
void printBigNumber(char string, int X, int Y)
{
    setXY(X,Y);               //set the cursor to start
    int salto=0;
    for(int i=0;i<96;i++)
    {
        if(string == ' ') {
            SendChar(0);
        } else
        SendChar(pgm_read_byte(bigNumbers[string-0x30]+i));

        if(salto == 23) {
            salto = 0;
            X++;
            setXY(X,Y);
            } else {
            salto++;
        }
    }
}
```

```
//============================================================//
/** Actually this sends a byte, not a char to draw in the display.
* Displays chars uses 8 byte font the small ones and 96 bytes
* for the big number font.*/
void SendChar(unsigned char data)
{
    I2C_Start(_i2c_address); // begin transmitting
    I2C_Write(0x40);//data mode
    I2C_Write(data);
    I2C_Stop();    // stop transmitting
}

//============================================================//
/** Prints a display char (not just a byte) in coordinates X Y,
* being multiples of 8. This means we have 16 COLS (0-15)
* and 8 ROWS (0-7).*/
void sendCharXY(unsigned char data, int X, int Y)
{
    setXY(X, Y);
    I2C_Start(_i2c_address); // begin transmitting
    I2C_Write(0x40);//data mode

    for(int i=0;i<8;i++)
    I2C_Write(pgm_read_byte(myFont[data-0x20]+i));

    I2C_Stop();    // stop transmitting
}

//============================================================//
/** Set the cursor position in a 16 COL * 8 ROW map.*/
void setXY(unsigned char row,unsigned char col)
{
    ssd1306_command(0xb0+row);                //set page address    p. 31
    ssd1306_command(0x00+(8*col&0x0f));       //set low col address   p. 30
    ssd1306_command(0x10+((8*col>>4)&0x0f));  //set high col address   p.30
}


//============================================================//
/** Prints a string regardless the cursor position.*/
void sendStr(char *string)
{
    unsigned char i=0;
    while(*string)
    {
        for(i=0;i<8;i++)
        {
```

```
            SendChar(pgm_read_byte(myFont[*string-0x20]+i));    //look up ascii
chars (no danish) defined in data.h
        }
        string++;
    }
}

//============================================================//
/** Prints a string in coordinates X Y, being multiples of 8.
* This means we have 16 COLS (0-15) and 8 ROWS (0-7).*/
void sendStrXY( char *string, int X, int Y)
{
    setXY(X,Y);
    unsigned char i=0;
    while(*string)
    {     if (*string=='\n'){
        setXY(X+1,0);
        string++;
        }
        for(i=0;i<8;i++)
        {
            SendChar(pgm_read_byte(myFont[*string-0x20]+i));
        }
        string++;
    }
}
void ssd1306_setpos(uint8_t x, uint8_t y)
{

    ssd1306_command(0xb0 + y);
    ssd1306_command(((x & 0xf0) >> 4) | 0x10); // | 0x10


}
void print_fonts(){
    clear_display();

    uint8_t data=32;
    for(int k=0;k<6;k++){
        setXY(k,0);

        for (int j=0;j<16;j++)
        {

            for(int i=0;i<8;i++){
            SendChar(pgm_read_byte(myFont[(data+j)-0x20]+i));
            }
```

```
        }
        data=data+16;
    }
    }
void ssd1306_draw_bmp(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1, const
uint8_t bitmap[])
{
    uint16_t j = 0;
    uint8_t y;
    if (y1 % 8 == 0) y = y1 / 8;
    else y = y1 / 8 + 1;
    for (y = y0; y < y1; y++)
    {
        ssd1306_setpos(x0,y);

        for (uint8_t x = x0; x < x1; x++)
        {
             ssd1306_data(pgm_read_byte(&bitmap[j++]));
        }

    }
}
/*
void drawPixel(int16_t x, int16_t y, uint16_t color) {
    if ((x < 0) || (x >= width()) || (y < 0) || (y >= height()))
    return;

    // check rotation, move pixel around if necessary
    switch (getRotation()) {
        case 1:
        ssd1306_swap(x, y);
        x = WIDTH - x - 1;
        break;
        case 2:
        x = WIDTH - x - 1;
        y = HEIGHT - y - 1;
        break;
        case 3:
        ssd1306_swap(x, y);
        y = HEIGHT - y - 1;
        break;
    }

    // x is which column
    switch (color)
    {
        case WHITE:   buffer[x+ (y/8)*SSD1306_LCDWIDTH] |=  (1 << (y&7)); break;
```

```
        case BLACK:   buffer[x+ (y/8)*SSD1306_LCDWIDTH] &= ~(1 << (y&7)); break;
        case INVERSE: buffer[x+ (y/8)*SSD1306_LCDWIDTH] ^=  (1 << (y&7)); break;
    }

}
*/
void invertDisplay(uint8_t i) {
    if (i) {
        ssd1306_command(SSD1306_INVERTDISPLAY);
        } else {
        ssd1306_command(SSD1306_NORMALDISPLAY);
    }
}
/** startscrollright
* Activate a right handed scroll for rows start through stop
* Hint, the display is 16 rows tall. To scroll the whole display, run:
* scroll right(0x00, 0x0F)*/
void startscrollright(uint8_t start, uint8_t stop){
    ssd1306_command(SSD1306_RIGHT_HORIZONTAL_SCROLL);
    ssd1306_command(0X00);
    ssd1306_command(start);
    ssd1306_command(0X00);
    ssd1306_command(stop);
    ssd1306_command(0X00);
    ssd1306_command(0XFF);
    ssd1306_command(SSD1306_ACTIVATE_SCROLL);
}
/** startscrollleft
* Activate a right handed scroll for rows start through stop
*Hint, the display is 16 rows tall. To scroll the whole display, run:
* scrollleft(0x00, 0x0F) */
void startscrollleft(uint8_t start, uint8_t stop){
    ssd1306_command(SSD1306_LEFT_HORIZONTAL_SCROLL);
    ssd1306_command(0X00);
    ssd1306_command(start);
    ssd1306_command(0X00);
    ssd1306_command(stop);
    ssd1306_command(0X00);
    ssd1306_command(0XFF);
    ssd1306_command(SSD1306_ACTIVATE_SCROLL);
}
/** startscrolldiagright
*Activate a diagonal scroll for rows start through stop
* Hint, the display is 16 rows tall. To scroll the whole display, run:
* display.scrollright(0x00, 0x0F)*/
void startscrolldiagright(uint8_t start, uint8_t stop){
    ssd1306_command(SSD1306_SET_VERTICAL_SCROLL_AREA);
```

```
    ssd1306_command(0X00);
    ssd1306_command(SSD1306_LCDHEIGHT);
    ssd1306_command(SSD1306_VERTICAL_AND_RIGHT_HORIZONTAL_SCROLL);
    ssd1306_command(0X00);
    ssd1306_command(start);
    ssd1306_command(0X00);
    ssd1306_command(stop);
    ssd1306_command(0X01);
    ssd1306_command(SSD1306_ACTIVATE_SCROLL);
}
/** startscrolldiagleft
* Activate a diagonal scroll for rows start through stop
*Hint, the display is 16 rows tall. To scroll the whole display, run:
* display.scrollright(0x00, 0x0F)*/
void startscrolldiagleft(uint8_t start, uint8_t stop){
    ssd1306_command(SSD1306_SET_VERTICAL_SCROLL_AREA);
    ssd1306_command(0X00);
    ssd1306_command(SSD1306_LCDHEIGHT);
    ssd1306_command(SSD1306_VERTICAL_AND_LEFT_HORIZONTAL_SCROLL);
    ssd1306_command(0X00);
    ssd1306_command(start);
    ssd1306_command(0X00);
    ssd1306_command(stop);
    ssd1306_command(0X01);
    ssd1306_command(SSD1306_ACTIVATE_SCROLL);
}
void stopscroll(void){
    ssd1306_command(SSD1306_DEACTIVATE_SCROLL);
}

// Dim the display
// dim = true: display is dimmed
// dim = false: display is normal
void dim(bool dim) {
    uint8_t contrast;

    if (dim) {
        contrast = 0; // Dimmed display
        } else {
        if (_vccstate == SSD1306_EXTERNALVCC) {
            contrast = 0x9F;
            } else {
            contrast = 0xCF;
        }
    }
    // the range of contrast to too small to be really useful
    // it is useful to dim the display
```

```
    ssd1306_command(SSD1306_SETCONTRAST);
    ssd1306_command(contrast);
}

/*
Formål: Modulet initialiserer og håndterer Timer0 i CTC (Clear Timer on Compare
Match) mode for at generere præcise timings til ADC-konverteringer. Det
håndterer også eksterne interrupts for at detektere knaptryk.

Input:
- Ingen eksterne inputs direkte til funktionerne, men timer og interrupt
reagerer på systemets tidsintervaller og eksterne knaptryk.

Output:
- initTimer0 funktionen konfigurerer Timer0 og aktiverer interrupts, der trigger
ADC-konvertering ved hver timer compare match.
- Interrupt-rutinen for INT4 detekterer, om en specifik knap er blevet trykket
og sætter en flag.

Bruger:
- "timer.h" og <avr/interrupt.h> for interrupt-funktionalitet og
timerkonfiguration.
- Timer- og interrupt-registre manipuleres direkte via hardware-specifikke
operationer.

Funktioner:
- initTimer0(): Sætter Timer0 til CTC mode og definerer frekvensen og
prescaleren for at matche systemets krav.
- ISR(TIMER0_COMPA_vect): Udløser en ADC-konvertering som respons på timer
compare matches.
- ISR(INT4_vect): Kontrollerer tilstanden af en bestemt knap og sætter en flag,
hvis den er trykket ned.

Author: Benjamin Hadziosmanovic, Ali Al-Sayad, Younes Humadi
   Company: DTU
   Version: 1.0
   Date and year: 4/5 2024
*/
#include "timer.h"
#include <avr/interrupt.h>

volatile uint8_t button_pressed = 0; // Variabel til at angive om knappen er
blevet trykket

// Opsæt Timer0 til CTC mode
void initTimer0() {
    TCCR0A |= (1 << WGM01);  // Sæt Timer0 til CTC mode
```

```
    OCR0A = 13;  // For ca. 9500 Hz ved 8 MHz CPU med en prescaler på 64
    TCCR0B |= (1 << CS01) | (1 << CS00);  // Prescaler: 64
    TIMSK0 |= (1 << OCIE0A);  // Aktivér Timer0 Compare Match A Interrupt
}

// Interrupt service rutine for Timer0 CTC afbrydelse
ISR(TIMER0_COMPA_vect) {
    ADCSRA |= (1 << ADSC);  // Start en ADC konvertering
}

// Interrupt service rutine for ekstern interrupt INT4
ISR(INT4_vect) {
    // Tjek om knappen er i lav tilstand (trykket)
    if (!(PINE & (1 << PE4))) {
        button_pressed = 1;
    }
}

/*
Formål: Dette modul implementerer grundlæggende UART (Universal Asynchronous
Receiver-Transmitter) funktionaliteter for at muliggøre dataudveksling mellem
mikrocontrolleren og andre enheder via seriel kommunikation.

Input:
- 'baudRate' for initiering, som bestemmer kommunikationshastigheden.
- Data eller strenge, der sendes eller modtages gennem UART.

Output:
- Sender og modtager data mellem mikrocontrolleren og eksterne enheder.
- Funktioner til at sende og modtage både enkelte karakterer og hele strenge.

Bruger:
- "UART.h" for at definere og referere til nødvendige funktioner og opsætninger.
- Direkte manipulation af AVR's hardware-registre til at kontrollere UART
opsætningen.

Funktioner:
- uart0_Init(): Initialiserer UART med en specificeret baudrate.
- putsUSART0(), putchUSART0(), uart_transmit(): Funktioner til at sende data.
- uart_receive(), uart_sendString(), uart_readString(): Funktioner til at
modtage data og håndtere string inputs.

Author: Benjamin Hadziosmanovic, Ali Al-Sayad, Younes Humadi
   Company: DTU
   Version: 1.0
   Date and year: 4/5 2024
*/
```

```
#include "UART.h"
#include <avr/io.h>
#include <stdio.h>
#include <Arduino.h>
#include <stdio.h>
#include <avr/io.h>
#include <stdlib.h>
#include <util/delay.h>
#include <avr/interrupt.h>

void uart0_Init(unsigned int baudRate) {
    UCSR0A = 0; // Deaktiver U2X mode
    UCSR0B = (1 << RXEN0) | (1 << TXEN0);  // Aktivér receiver og transmitter
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); // 8-bit data
    unsigned int ubrr = F_CPU / 16 / baudRate - 1;
    UBRR0H = (ubrr >> 8);
    UBRR0L = ubrr;
}

void putsUSART0(char *ptr) {
    while (*ptr) {
        putchUSART0(*ptr++);
    }
}

void putchUSART0(char tx) {
    while (!(UCSR0A & (1<<UDRE0)));
    UDR0 = tx;  // Send tegn
}

void uart_transmit(char data) { // Funktion til at transmittere et enkelt tegn
    while (!(UCSR0A & (1 << UDRE0))); // Venter indtil data registeret er klar
til transmission

    UDR0 = data; // Sender data
}

char uart_receive(void) { // Funktion til at modtage et enkelt tegn
    while (!(UCSR0A & (1 << RXC0))); // Venter indtil data er modtaget

    return UDR0; // Returnerer modtaget data
}

void uart_sendString(const char *str) { // Funktion til at transmittere en
streng
    while (*str) { // Loop indtil nulltermineringen af strengen
        uart_transmit(*str++); // Transmitter hvert tegn i strengen
```

```
    }
}

void uart_readString(char *str, uint16_t maxLen) {
    memset(str, 0, maxLen);  // Nulstil hele bufferen før brug
    uint16_t i = 0;
    char c;

    while (i < maxLen - 1) {  // Justeret for at undgå buffer overflow
        c = uart_receive();
        if (c == '\r' || c == '\n') {
            if (c == '\r') {  // Check efterfølgende '\n' hvis tilstede
                c = uart_receive();
                if (c != '\n') {
                    str[i++] = '\r';  // Hvis ingen '\n', gem '\r'
                }
            }
            break;  // Stop læsning ved første linjeafslutning
        }
        str[i++] = c;
    }
    str[i] = '\0';  // Sikrer korrekt string afslutning
}
```