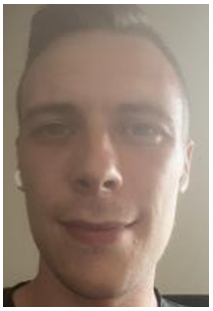
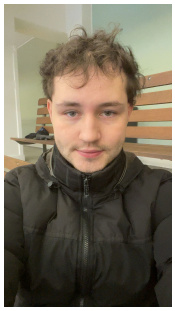
Metaldetektor projekt
Gruppe 1

34621
Electromagnetic sensors and digital signal processing
Danmarks Tekniske Universitet



Sebastian Sørensen,
s233986



Oliver Holm,
s233988

Bilal Alali,
s171678

January 22, 2026

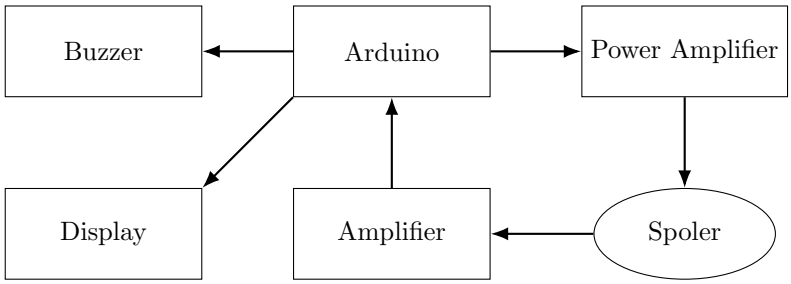
Indholdsfortegnelse

1 Introduktion

Dette projekt omhandler udviklingen af en metaldetektor, herunder skal vi selv designe og implementere hhv. software og hardware. Software delen skrives i embedded C, på en Arduino AtMega2560, hvor data vi sampler fra spolerne skal behandles og outputtes til brugeren.

Hardware-delen af projektet omhandler primært design og implementeringen af sensoren til detektion af metaller, samt diverse kredsløb, der har til formål at drive spolen, og filtrere samt forstærke signalet. Systemet skal outputte en beregnet amplitude og fase til OLED-display, men vi har valgt også at implementere en buzzer til audio-feedback.

2 Analyse



Figur 1: Overordnet moduldiagram af systemet

2.1 Analog

2.1.1 Design af spole

Ved sammenligning af de forskellige spoleudformninger (som er nævnt i "Coil Basics" fra kursets DTU Learn side), ligger vi mærke til følgende forskelle:
DD-spolen har bedre dækning til siderne, hvorimod den konsentriske er mere følsom under selve spolen og kan scanne dybere. Vi ser også at DD-spolen har en mere kompleks struktur, hvilket vil kræve mere arbejde i form af strengere krav til 3D-print. Det er desuden også nævnt i artiklen at det, ved anvendelse af den konsentriske form, er nemmere at udligne feltet fra TX-spolen i Bucking-spolen.
Grundet den mere simple opbygning - og bedre scannings dybde, vælger vi at anvende den konsentriske opbyging.

2.1.2 Signalbehandling

Ved signalfiltreringen ser vi aflæsningen af fasen som noget vi skal være særligt opmærksomme på, da det er denne som skal bruges til at skelne imellem metallerne.

Skin-effekten medfører både amplitudedæmpning og faseskift i det indtrængende felt. For en god leder er dæmpningskonstanten α og fasekonstanten β givet ved $\alpha = \beta = \frac{1}{\delta}$, således at faseskiftet ϕ over en indtrængningsdybde z kan beregnes ved $\phi = \frac{z}{\delta}$, hvor

$$\delta = \sqrt{\frac{2}{\omega\mu\sigma}}$$

Ved detektionsfrekvensen på $f_d = 2\text{kHz}$ (krav 12), og anvendelse af materialernes ledningsevner, forventer vi de relative størrelsesforhold mellem materialernes faseskift til at være

Table 1: Forventet relativt faseskift i materialerne, ved antagelse af $\mu = 1$

Metal	**Ledningsevne [S/m]**	**Faseskift** [°]
Kobber	$5.96 \cdot 10^7$	$75 - 85$
Messing	$1.5 \cdot 10^7$	$60 - 75$
Aluminium	$3.5 \cdot 10^7$	$55 - 70$

Dette er dog kun en approksimation, da faseskiftet er afhængigt af flere faktorer såsom temperatur, materialets geometri, samt afstanden mellem sensoren og materialet.
Da de 3 metaller som vi jf. kravspecifikationen skal kunne skelne imellem ligger relativt tæt i faseskift, er vi særligt opmærksomme på ikke at forstyrre fasen i signalbehandlingen.

2.2 Digital

2.2.1 Timing og sampling

Systemet bruger Timer1 til vores 2 frekvenser, hhv. sampling- /spole frekvens. Selve Timer1 generere de 8 kHz til samplingsfrekvensen og ved at toggle udgangen hver anden interrupt fås spolefrekvensen til 2 kHz ved

$$\frac{8\text{kHz}}{2 \cdot 2} = 2\text{kHz}$$

Dette bruger vi også til at sikre at sampling kun sker på rising_edge, som sikrer synkronisering med spolen. Ved at sample med 8 kHz kan de 2 kHz signal samples 4 gange per periode, hvilket tillader en simpel DFT implementering.

2.2.2 DFT

Vi er opmærksomme på optimering af algoritmen da DFT'en skal køre konstant. For at undgå beregninger med komplekse tal, udnytter vi at ved at sample 4 gange hurtigere end vi sender til spolen, vil vi kun ramme de samme 4 punkter på enhedscirklen.

Alternativt, kan man lave en FFT-algoritme. Dog er denne mere beregningstung, og vi ser derfor DFT'en som den mest passende løsning, da systemet udføre realtidsberegninger er CPU-tid kritisk.

2.2.3 Double buffer

For at undgå data konflikt mellem ISR og main benyttes et double buffer system, bestående af adc_bud[0] og adc_buf[1]. Bufferne skiftes automatisk når den ene er fuld, så den tomme kan begynde at samle data og den allerede indsamlede data kan behandles. Et flag giver signal til main, når nye DFT data er klar til behandling.

2.2.4 DSP funktioner

DSP_fast_magnitude() beregner $|Z| = \sqrt{(\Re(z)^2 + \Im(z)^2)}$ ved brug af floating point sqrt(), som hentes fra biblioteket math.h. Denne tilgangsmåde blev valgt grundet dens præcision og enkelthed.
Vi opbyggede som alternativ en algoritme til heltals approksimation (se ukommenteret funktion i appendix DSP.c), men vurderede at den ikke ville være den optimale løsning trods større hastighed, dette skyldes mangel på præcision.
DSP_fast_atan2_deg() er funktionen til fase beregning. Den returnerer fase i grader fra -180 til 180 ved brug af atan2() ligeledes hentet fra math.h biblioteket. Fasen bruges til at skelne mellem metaltyper, hvilket indgår i opgavekravene. Her opbyggede vi ligeledes en algoritme til hastighed og optimering af hukommelses brug, dog kom vi til samme konklusion at atan2() havde bedre præcision.

2.2.5 Display og buzzer

Designvalg og begrundelse

Da kun en frekvens på 2 kHz er relevant, er DFT algoritmen betydeligt mere effektiv end at foretage en stor FFT. Denne løsning vil reducere CPU'ens belastning samt tillade realtidsbehandling på en 16 MHz ATmega2560.

Double buffer system

Sikrer at der kan samples parallelt med at main behandler den samplede data. Denne tilgang benyttes for at undgå timings problemer.

Glidende gennemsnit

Filtrerer støj fra ved at tage gennemsnittet af 32 blokke før data vises på OLED som sikre stabile aflæsninger. Alternativer findes til denne metode, men denne tilgang virkede hermed beholdes den.

Kalibrering

Ved at måle et sted uden metal (tom luft), kompenseres der for DC-offset og miljømæssige faktorer såsom temperaturskift. Dette er nødvendigt for at deltamålingerne er pålidelige. Ved at tage gennemsnittet af 32 baseline samples kommer det på bekostning af CPU'ens hastighed. Hvis der kun benyttes 1 blok i stedet, havde kalibreringen været betydelig hurtigere, dog mere upræcis, her vurderes at præcision var vigtigst.

2.2.6 Sleep

Da krav 5 i kravspecifikationen vedrører strømforbruget i systemet, er vi opmærksomme på at MCU'en kan konfigureres med sleep-funktioner for at forlænge batterilevetiden, hvis nødvendigt.

3 Design

3.1 Analogt design

3.1.1 Energiberegninger

Vi undersøgte os frem til at batteriet har en kapacitet på 550mAh. Da batteriet aflades fra 9V til 6V, har vi lavet udregningen:

$$E_{tot} = V \cdot I \cdot 3600 \iff E_{tot} = \frac{9V + 6V}{2} \cdot 0.55\text{A} \cdot 3600\text{s} = 14850\text{J}$$

Ved måling af strømforbruget finder vi at OLED-display og Arduinoen forbuger $\approx 70.5\text{mA}$. Da vi skal kunne drive systemet i 100 minutter, kan vi derfor beregne

$$E_{MCU} = 5\text{V} \cdot 70.5 \cdot 10^{-3}\text{A} \cdot (100 \cdot 60)\text{s} = 2115\text{J}$$

Vi reserverer 10% af batteriet som buffer, for at være sikker på at overholde kravene. Resten af energien kan bruges til TX-spolen.

$$E_{TX} = E_{tot} \cdot (1 - 0.15) - E_{MCU} = 11250J$$

3.1.2 Spoleberegninger

TX-spole

Ved resonans kan spændingen gennem spolen approksimeres ved grundtonen i fourierrækken

$$a_n \approx 2\left(\frac{A}{n\pi}\right)\sin\left(\frac{n\pi}{2}\right) = 2\left(\frac{9\text{V}}{\pi}\right)\sin\left(\frac{\pi}{2}\right) = 5.73\text{V}$$

Hvorved RMS-spændingen kan bestemmes

$$V_{RMS} = \frac{a_n}{\sqrt{2}} = 4.05\text{V}$$

Derved kan vi nu beregne strømmen vi kan sende igennem spolen

$$I_{TX} = \frac{E_{TX}}{t \cdot V_{RMS}} = \frac{11250\text{J}}{(60 \cdot 100)\text{s} \cdot 4.05\text{V}} = 0.4628\text{A}$$

Radiussen på TX-spolen vælges til $r_{TX} = 10cm$, med $D_{TX} = \text{Ø}0.4\text{mm}$. Derved kan antallet af viklinger beregnes ved

$$R_{TX} = \frac{V_{RMS}}{I_{TX}} = 8.754\Omega$$

$$N_{TX} = \frac{R_{TX} \cdot \left(\frac{D_{TX}}{2}\right)^2}{2 \cdot r_{TX} \cdot \rho_{cu}} = \frac{8.754\Omega \cdot \left(\frac{0.4\text{mm}}{2}\right)^2}{2 \cdot 10\text{cm} \cdot 1.77 \cdot 10^{-8}\Omega \cdot m} = 98.9$$

hvilket afrundes til $N_{TX} = 99$.
Selvinduktansen i spolen findes ved

$$L = \frac{0.394 \cdot r^2 \cdot N^2}{9 \cdot r + 10A} = \frac{0.394 \cdot 10^2 \cdot 99^2}{9 \cdot 10 + 10} = 3.855\text{mH}$$

For at kunne lave Arduino'ens firkantsignal til en sinuskurve anvendes et resonanskredsløb, som har resonansfrekvens på 2kHz. Størrelsen af kondensatoren findes ved at løse ligningen

$$f = \frac{1}{2\pi\sqrt{LC}} \iff 2\text{kHz} = \frac{1}{2\pi\sqrt{3.855\text{mH} \cdot C}} \implies C = 1.642\mu\text{F} \qquad (1)$$

Magnetfeltet fra TX-spolen beregnes, da bucking-spolen skal udligne B-feltet fra TX-spolen, for på den måde at tilsikre at vi kun måler på magnetfeltet fra metallet i detektoren.

$$|B| = \frac{\mu_0 I_{TX} N_{TX}}{2r_{TX}} = \frac{4\pi \cdot 10^{-7}\frac{\text{H}}{\text{m}} \cdot 0.4628\text{A} \cdot 99}{2 \cdot 10\text{cm}} = 287.6\mu\text{T}$$

Bucking-spole

Derved kan Bucking-spolen beregnes. Vi vælger en radius på $r_{Buck} = 5\text{cm}$, og antallet af viklinger beregnes ved

$$|B| = \frac{\mu_0 \cdot I_{TX} \cdot N_{Buck}}{2 \cdot r_{Buck}} \iff 287.6\mu\text{T} = \frac{4\pi \cdot 10^{-7} \cdot 0.4628\text{A} \cdot N_{Buck}}{2 \cdot 5\text{cm}} \implies N_{Buck} = 49.458$$

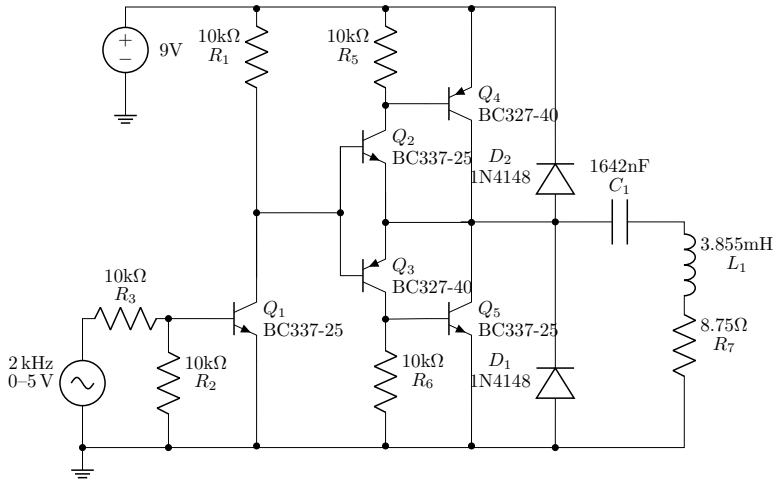
hvilket afrundes til $N_{Buck} = 50$.

RX-spole

For at opnå en god følsomhed på sensoren, er vi opmærksomme på at induktansen i RX-spolen skal være minimum $L_{RX} = 10\text{mH}$. Derved beregnes antallet af viklinger ved

$$10 \cdot 10^3 = \frac{0.394 \cdot r_{RX}^2 \cdot N_{RX}^2}{9r_{RX} + 10A_{RX}} = \frac{0.394 \cdot 5^2 \cdot N_{RX}^2}{9 \cdot 5 + 10} \implies N_{RX} = 236$$

3.1.3 Power Amplifier

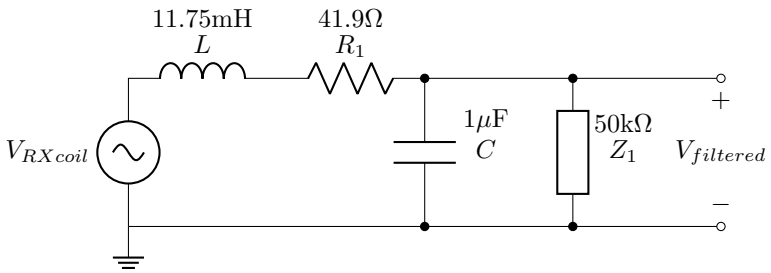


Figur 2: Power amplifier kredsløb til at øge strømmen gennem TX-spolen.

Ovenfor ses vores design af effektforstærkeren, som bruges til at forstærke signalet fra Arduino'en til at kunne drive TX-spolen. Kredsløbet virker ved:

- Firkantsignalet sendes ind i forstærkeren gennem en modstand som beskytter Arduino'en og styrer strømmen ind i transistoren Q_1.
- Transistor-konfigurationen fungerer som en forstærker som via. en styrestrøm kontrollerer en større strøm fra batteriforsyningen (9V batteriet), som driver TX-spolen.
- TX-spolen (L_1) genererer et magnetfelt ved resonansfrekvensen
- Kondensatoren (C_1) er beregnet (Ligning 1) således at der opstår resonans ved detektionsfrekvensen på 2kHz
- Modstanden R_7 repræsenterer den ohmske modstand i TX-spolen.
- Dioderne D_1 og D_2 beskytter kredsløbet fra den induserede spænding i spolen.

3.1.4 Filtrering af spolesignal



Figur 3: Kredsløb til at filtrere støj væk inden signalet bliver forstærket (som vist i Figur 5). Impedansen på 50kΩ repræsenterer indgangsimpedansen i operationsforstærkerkredsløbet.

Som det ses i figuren, er $V_{filtered} = V_{Z_1}$. Filterets overføringsfunktion kan opstilles ved almindelig spændingsdeling, dvs.

$$H(\omega) = \frac{Z_1 \cdot \frac{1}{j\omega C}}{j\omega L + R_1 + Z_1 \cdot \frac{1}{j\omega C}}$$

hvorefter fasen kan analyseres ved

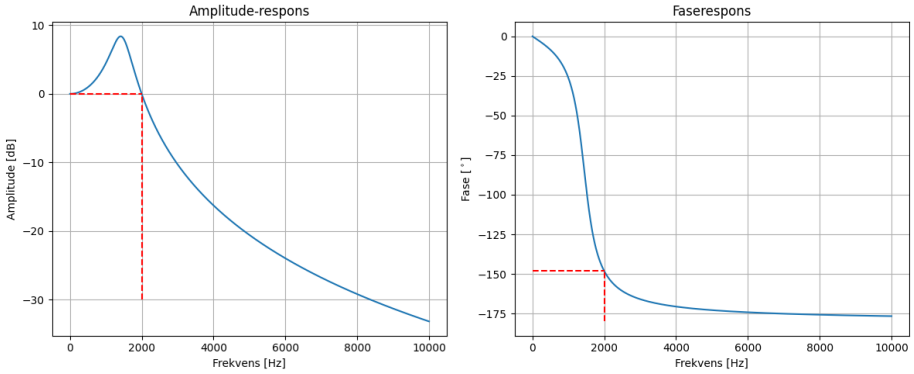
$$\phi(\omega) = \tan^{-1}\left(\frac{\Im\{H\}}{\Re\{H\}}\right)$$

og amplitude responset ved

$$H_{dB} = 20\log 10(H(\omega))$$

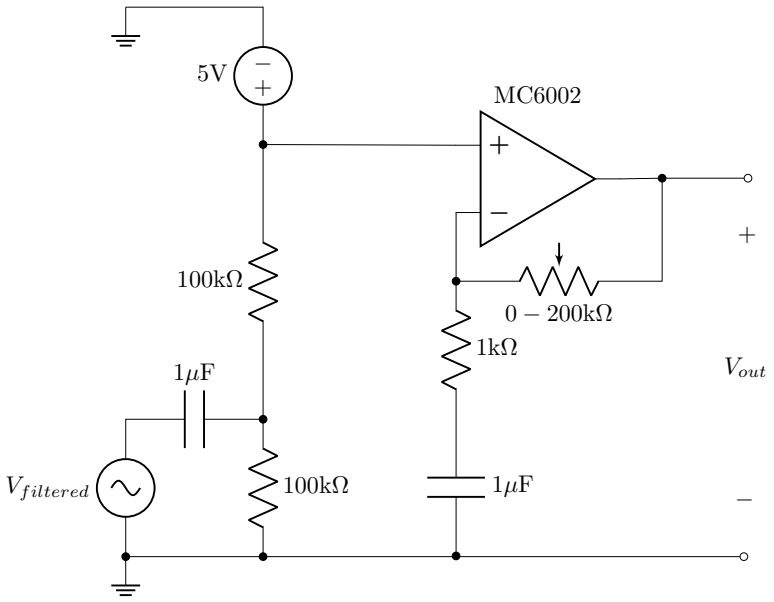
Ifølge Shannon's sampling teorem ved vi at $F_{max} < \frac{F_s}{2}$ skal være overholdt for at forhindre aliaseringer. Det eneste signal som vi ønsker at få igennem, ligger $\approx$ 2kHz, og derfor har vi valgt at alle frekvenser over 2kHz skal attenueres. Designet af RX-filteret er en balancegang imellem at dæmpe amplituden af uønskede frekvenser, mens vi bibeholder en rimeligt præcis faserespons ved 2kHz, for at kunne skelne imellem forskellige metaller.

Figur 4: RX-filterets fase- og magnitude respons, med markering af 2kHz



Da metaldetektorens følsomhed overfor frekvenser afhænger af bl.a temperatur, har vi valgt at designe filteret til at opnå mindst muligt forstyrrelse omkring de 2kHz, samtidigt med vi garanterer at der ikke opstår aliaseringer (Samplingteoremet er overholdt). Som det ses i plottet, ændrer fasen sig ikke voldsomt ved signalfrekvensen, og selv hvis frekvensen skrider ±100Hz går signalstyrken relativt uberørt igennem.

3.1.5 Forstærkning af spolesignalet

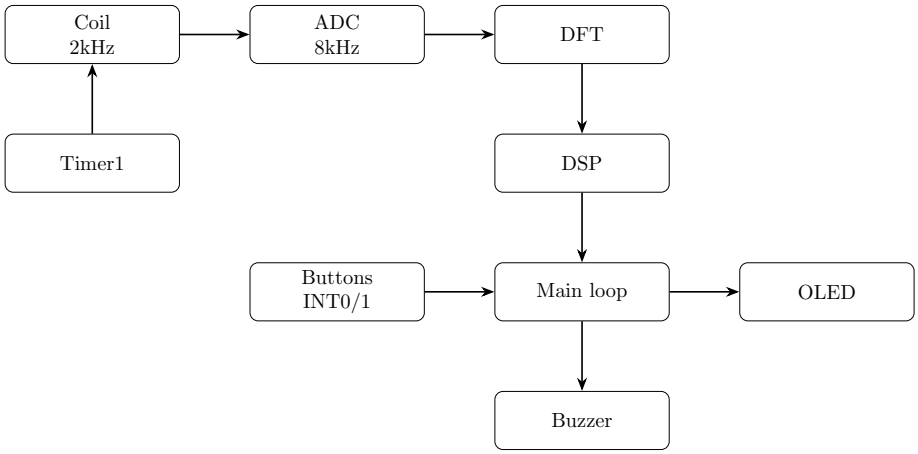


Figur 5: Kredsløb til at forstærke den filtrerede spænding fra RX-spolen op, inden det læses ind i MCU ADC'en. $V_{filtered}$ er outputtet fra RX-filteret.

Da spændingen fra RX-spolen ikke er højt nok til at vi kan aflæse det på MCU'en, bruger vi dette operationsforstærker kredsløb, til at trække spændingen op til $\approx$ 5V, for på den måde at anvende flest mulige bits i ADC'en og opnå bedst mulig opløsning. Vi anvender et potentiometer som feedback-modstand, for på den måde at kunne forstærke et mindre signal op ved behov, så følsomheden på detektoren kan indstilles.

3.2 Digitalt design

3.2.1 Moduldiagram og introduktion



Figur 6: Overordnet moduldiagram af systemet

3.2.2 Timer1

1. Funktion:
   Generer en samplingsfrekvens på 8 kHz (Fs), og en spolefrekvens på 2 kHz.

2. Designvalg
   Timer1 opstilles i CTC-mode med en prescaler på 8. Sammenligningsværdien beregnes ved:

$$OCR1A = \frac{F_{CPU}}{prescaler \cdot F_s} - 1 = \frac{16\text{MHz}}{8 \cdot 8\text{kHz}} - 1 = 249$$

   Spolefrekvensen sættes ved at toggle PB5 hver 2. interrupt : $\frac{F_s}{2 \cdot 2}$ = 2kHzTimer1 toggler COIL_PIN ved 2 kHz. Hver samplingblok synkroniseres til spolens rising edge, så sample[0] altid har samme faserelation til vores Tx signal. Blokkens startposition drifte ville drifte uden dette og det vil give inkonsistente fase værdier.

3. Begrundelse
   Krav: Metaldetektor med 2 kHz drivfrekvens og passende samplerate til fasebestemmelse.

   Alternativ: Separat timer til spole og ADC. Denne idé blev droppet, idet en enkel timer med en divider er tilstrækkeligt, det sikrer også synkronisering af ADC og spole, derudover undgås en potentiel konflikt mellem de 2 ISR også.

Register	Værdi	Funktion
TCCR1A	0x00	Normal port operation
TCCR1B	(1 << WGM12) | (1 << CS11)	CTC mode, prescaler 8
OCR1A	249	8 kHz compare match
TIMSK1	(1 << OCIE1A)	Interrupt enable (ISR)

3.2.3 ADC

1. Funktion:
   Det analoge signal konverteres til digitale samples ved 8 kHz.

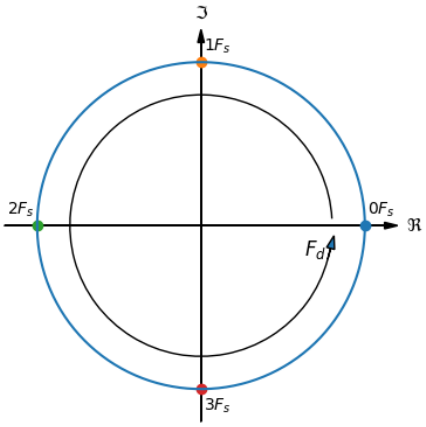
2. Designvalg
   ADC sættes med AVcc som reference, og en prescaler på 64. Denne opstilling giver en ADC clock på 250 kHz: $F_{clock} = \frac{F_{CPU}}{prescaler} = \frac{16\text{MHz}}{64} = 250\text{kHz}$ En ADC konvertering tager $13 \cdot f_c = 52\mu s$, dette ligger indenfor $125\mu s$ perioden mellem samples, hvor konverteringstiden er givet ved $\Delta t = \frac{13}{250\text{kHz}} = 52\mu s$

3. Begrundelse
   Krav: Passende ADC opløsning og hastighed til at kunne sample et 2kHz signal præcist (fra spolen).

   Alternativ: Prescaler på 128, hvilket vil give 125 kHz. Denne konfiguration giver et mindre støjet signal, men en konverteringshastighed på $104\mu s$ som vurderes at være for tæt på sampletiden, dermed benyttes prescaleren på 64.

Register	Værdi	Funktion
ADMUX	(1 << REFS0)	ADC0, Avcc ref
ADCSRA	(1 << ADEN) | (1 <<ADIE) | (1 << ADPS2) | (1 << ADPS1)	ADC Enable, ISR Enable, prescaler 64
DIDR0	(1 << ADC0D)	Disable digital input

3.2.4 DFT



Figur 7: Grafisk illustration over princippet bag DFT-algoritmen. Ved at vælge $F_d = \frac{F_s}{4}$, undgår vi at lave komplicerede komplekse beregninger.

1. Funktion:
   Beregner amplitude og fase ved 2 kHz fra de indsamlede samples.

2. Designvalg
   En DFT algoritme opstilles ud fra k, således: $k = \frac{f \cdot N}{Fs} = \frac{2000*64}{8000} = 16$. Dette udnyttes idet k følger $\cos(2\pi\frac{k \cdot n}{N})$ og $\sin(2\pi\frac{k \cdot n}{N})$ gennem "1,0,-1,0". Hermed kan multiplikationer hvilket spare CPU'en en del.

n	$\cos\left(\frac{\pi n}{2}\right)$	Operation
0	cos = 1, sin = 0	re += sample
1	cos = 0, sin = 1	im -= sample
2	cos = -1, sin = 0	re -= sample
3	cos = 0, sin = -1	im += sample

3. Begrundelse
   Krav: Realtids bestemmelse af fase ved 2 kHz på MCU.

   Alternativ: En fuld FFT på 64 punkter, men det ville kræve mange komplekse multiplikationer per blok, hvilket ikke ønskes, idet den nærmest konstant skal være i gang, det vil betyde en alt for stor reduktion i CPU-tid.

4. Ingeniørmæssigt bidrag:
   Genkendelse af den specifikke frekvensrelation $\frac{Fs}{f} = 4$ til at fjerne trigonomiske beregninger er absolut nødvendig optimering, der muliggør realtidsbehandling.

3.2.5 Double buffer

1. Funktion:
   Sørger for kontinuerlig dataflow mellem ISR og main.

2. Designvalg
   buffere adc_buf[0] og adc_buf[1] skiftes automatisk mellem aktivt at sample ved blok afslutning. Flags dft_ready[b] og adc_ready[b] signalerer til main der, med FSM, styre programmets dataflow.

3. Begrundelse
   Krav: Kontinuerlig sampling uden at miste data.

   Alternativ: En enkelt buffer med et stop-and-go princip vil kunne give huller i samplingen som potentielt kan misse korte metal detektioner. Alternativt er en ringbuffer en løsning, dog mere kompleks og muligvis ville data kunne overskrives hvilket igen leder til missede metaldetektorer.

3.2.6 DSP

1. Amplitude Implementation:
   abs(z)=sqrt(re*re+im*im) med floating point sqrt() fra math.h.

   Alternativ:
   En heltals approksimations algoritme: abs(Z) = max + 0.375 * min. Dette blev fravalgt, idet præcision prioriteres frem for hastighed og sqrt() kaldes kun 1 gang hver 8ms (per 64 samples), altså er hastighedsreduktionen ikke kritisk.

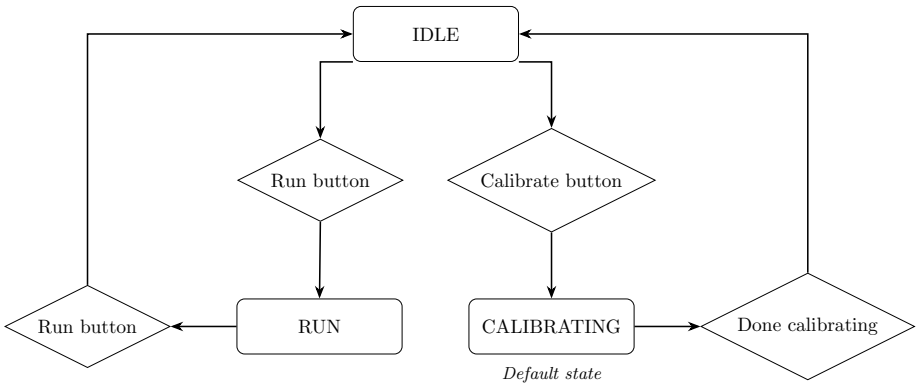
2. Fase Implementation:
   fase = atan2(im,re) * 180/pi, returner grader i intervallet [-180, 180].

   Alternativ:
   Atan2 approksimations algoritme, men fravalgt da math.h bibliotekets atan2() er hurtig nok og atan2() yder bedst præcision.

3. Glidende gennemsnit Implementation:
   32-sample circulær buffer for amplituden og fasen. Gennemsnittet beregnes ved summation over hele bufferen.

   Tidsrespons: 32 blokke * 8ms/blok = 256 ms for fuld beregning af gennemsnit. Antallet samples blev valgt efter flere test, hvori 32 havde bedst støjfiltrering i forhold til responstid.

3.2.7 Tilstandsmaskine



Figur 8: Flowchart af tilstandsmaskinen

Softwares dataflow styres i main af en FSM bestående af 3 tilstande, herunder: CALIBRATING, IDLE, RUNNING. Kalibrerings tilstanden indsamler 32 DFT blokke (uden metal), hvor den beregner et gennemsnit af disse som bruges til et nulpunkt.
Denne tilstand styres af en interrupt baseret knap, som tillader adgang til CALIBRATING efter behov, da temperaturskift kan påvirke metaldetektoren er det nødvendigt at kunne danne et nyt nulpunkt.
IDLE tilstanden bruges som en homescreen. Herinde er sampling deaktiveret, altså bruges der betydelig mindre strøm hvilket ligeledes er nødvendigt hvis man ikke aktivt søger metal. Ved opstart af metaldetektoren tilgås IDLE efter en automatisk kalibrering, herfra kan man enten genkalibrere eller gå til RUNNING ved brug af interrupt styret knapper.
RUNNING tilstanden er her metaldetektoren aktivt søger metal, beregner delta ud fra det kalibrerede nulpunkt, anvender 32-sample glidende gennemsnit til at undgå alt for hurtigere skift i amplitude og fase på OLED display.

3.2.8 Buzzer

1. Funktion:
   Generer akustisk feedback hvor frekvens indikerer metaltype og lydstyrke indikerer signalstyrke.

2. Designvalg
   Timer2 opstilles i fast PWM-mode med OCR2A som TOP og OCR2B som duty cycle. PH6 på Arduino ATMega2560 sættes på OC2B som output pin. Frekvens beregnes ved:

   f_out=F_CPU/(prescaler*(1+OCR2A)=16 MHz/(64*(1+OCR2A))

Register	Værdi	Funktion
TCCR2A	(1 << WGM21) | (1 << WGM20) | (1 << COM2B1)	Fast PWM, OC2B
TCCR2B	(1 << WGM22) | (1 << CS22)	TOP = OCR2A, prescaler 64
OCR2B	0-255 (styret af amplitude)	Volumen

3.2.9 Knap input

1. Funktion: Styre brugerinput på knapperne "run" og "pwr". Designvalg Knapperne er forbundet til PE4 og PE5 med interne pull-ups. Falling edge trigger starter ISR, som venter 50 ms derefter godkender, om knappen stadig er trykket.

2. Begrundelse Alternativ: Hardware debounce med et RC-filter, ville spare CPU-tid men ligeledes kræve ekstra komponenter, altså er software debounce et valg for simplicitet og fleksibilitet, dette tillades idet software løsningen virker uden problemer.

3. Opsummering af designvalg

Komponent	Valgt løsning	Primær fordel
DFT	Algoritme for "1,0,-1,0"	Undgår multiplikationer
Buffering	Double buffer	Ingen datatab
Amp/Fase	Amp() floating point, atan2()	Præcision
Filter	32 tap glidnde gennemsnit	Støjreduktion
Debounce	Software, 50 ms delay	Ingen ekstra hardware

4 Test og implementering

I vores gruppe har vi testet hvert modul løbende, og lavet eventuelle fejlrettelser som en del af implementeringsdelen i projektet.

4.1 Test

DFT: Vi har simuleret DFT-algoritmen ved at ligge samples i flash-hukommelsen, hentet det i bufferen og sammenholdt det beregnede resultat med vores egne beregninger, resultaterne var som forventet.

Forstærkerkredsløb: Begge vores forstærkerkredsløb (både Power amplifier og signalforstærkeren) er bekræftet igennem målinger og simulationer, og virker efter hensigten.

RX-filter: Vores RX-filter er både beregnet, og simuleret. Når vi måler på filteret opfører det sig præcis som forventet.

FSM: Via test_mode.c modul, testes FSM samt knapper ved at gennemløbe en cyklus af opdigtede målinger. Her fandt vi et bug, idet vores forrige FSM ikke tillod rekalibrering via knaptryk. Bugget er rettet til, og modulet virker nu.

Batteri levetid: Den komplette metaldetektor tilsluttes 9V batteri, kalibreres og stilles i run mode. Vi sætter en timer til at overvåge levetid, og efter 100 min frakobles 9V batteriet kredsløbet hvor restspænding måles til _. Den samlede levetid på batteriet i run-mode noteres til 3 timer 45 min.

Øvrige krav: De øvrige (mindre målbare) krav anser vi som testet, i og med at Per har godkendt vores projekt som helhed.

4.2 Opfyldelse af krav

Herunder ses kravspecifikationen metaldetektor projektet har haft til formål at opnå, samt hvilke vi har opnået.

Nr:	Navn:	Beskrivelse:	Prioritet
1	Amplitude/Fase detektion	Det skal være muligt at kunne detektere amplitude og fase i det reflekterede signal	1✓
2	Metal type	Skal kunne skelne jern fra kobber, messing og aluminium	1✓
3	Distance krav	Skal kunne detektere en jerngenstand (radius på ca. 15mm. Og en længde på 50mm.) i en dybde på 50mm. I fri luft	1✓
4	Strømforsyning	Hele metaldetektoren skal kunne køre på et 9 volts batteri (batteristørrelse E/6LR61) - kun et batteri.	1✓
5	Funktionstid	Metaldetektoren skal kunne køre minimum 100 minutter på batteriet og den ubelastede rest-spænding skal være større end 6 volt efter de 100 minutter	1✓
6a	Processor design	Kredsløbet skal være baseret på en Arduino	1✓

6b	Hardware design	Kredsløbene til at drive spolerne og forstærke det modtagne signal skal bygges med diskrete komponenter såsom: transistorer, operationsforstærkere, m.m.	1✓
7	Muligt software design	Brug Arduino'ens ADC og kontroller denne med timer interrupts	2✓
8a	Bruger interface display	Skal kunne udskrive amplitude og fase i et display	1✓
8b	Display udlæsningsstabilitet	Værdierne i displayet skal være læsbar for et flertal af brugere (spørg om hjælp)	1✓
8c	Forbedret display udlæsningsstabilitet	Der anvendes et FIR eller IIR filter til at lavpas-filtrere (midle) udlæsningen.	2✓
9a	Bruger interface primære knapper	Der skal være en start/stop knap	1✓
9b	Bruger intercace sekundære knapper	Der skal være en nulstillingsknap (fratrækker værdierne når detektoren ikke er i nærheden af metal)	1✓
10	Detektions princip	Very Low Frequency (VLF)	1✓
11	Samplingsfrekvens	8 kHz +/- 100Hz	1✓
12	Detektionsfrekvens	2 kHz +/- 100Hz	1✓
13	Metaldetektorudformning	3D-printet/trækonstruktion/plast	2X
14	Spolernes bæreenhed	3D-printet spoleform /læserskåret plastik	2✓
15	Spoleudformning	Spolerne anvender en centreret udformning med sende-spolen yderst og feedback/modtager spolen inderst.	2✓
16	Modtager spolens selvinduktans	Modtager spolen skal have en selvinduktans på mindst 10 mHy.	1✓

Hvilke krav er nået?

Krav 1) Vores system opfylder til fulde dette krav.

Krav 2) Systemet skelner mellem de påkrævede metaltyper.

Krav 3) Ved test af systemet kan vi detektere en lille jern-spidstang på $\approx$ 18cm afstand. Vi har fået Per til at kontrollere, og dette krav er opfyldt

Krav 4) Hele systemet er forsynet af ét 9V-6LR61 batteri.

Krav 5) Ved test kan vores system driftes fra et 9V-6LR61 i 3 timer og 45 minutter, inden batteriet er afladet og restspænding >6V efter 100 min.

Krav 6) Systemet er baseret på en Arduino Atmega2560

Krav 7) Forstærkerkredsløbene er bygget af diskrete komponenter

Krav 8) I designet anvender vi ADC og timer interrupts.

Krav 9) Værdierne bliver udskrevet korrekt til displayet.

Krav 10) Displayet læsbart og stabilt, dette er testet ved flere forskellige antal samples

Krav 11) Vi har anvendt et FIR-filter til at midle udlæsningen.

Krav 12) Systemet har en start/stop knap.

Krav 13) Systemet har en knap til kalibrering.

Krav 14) Dette krav er opfyldt.

Krav 15) Dette krav er opfyldt.

Krav 16) Dette krav er opfyldt.

Krav 17) Vi har ikke lavet kroppen/stangen til metaldetektoren, derved er dette krav ikke opfyldt.

Krav 18) Spolernes bæreenhed er 3D-printet.

Krav 19) Spolerne har en centreret udformning, som beskrevet i kravet.

Krav 20) Modtagerspolen har en selvinduktans på 11.75mH

5 Konklusion

Vi har helt fra begyndelsen haft stor fokus på projektplanlægningen, og har i starten været grundige med at holde morgenmøder, for at holde hvert enkelt gruppemedlem opdateret. Denne tilgang har hjulpet til konstant at have et overblik over både fremgangen i arbejdet, samt eventuelle problemer.
Den første dag snakkede hvert gruppemedlem om deres individuelle kompetancer og derfra tildeltes hvert medlem de individuelle ansvarsområder. Da vi er et ulige antal gruppemedlemmer fandt vi hurtigt ud af, at vi blev nødt til at overlappe den digitale- og den analoge del af projektet idet vi ellers ville vente for meget på hinanden.
Dette er også grunden til at vi stoppede morgenmøderne, da vi lavede resten af projektet sammen, og derved har holdt kommunikationen mindre formel og mere ad-hoc.Dette medførte også at vi i den digitale del af projektet været tvunget ud i at tænke mere i hvorledes programmet kunne simuleres, og det blev en rigtig positiv opdagelse vi tager med os videre.
Efter løbende at have testet og sammenlignet med forventet resultater på hver del af projektet både for sig selv, og derefter samlet konkluderede vi, at vores metaldetektor var tilfredsstillende, da de formelle krav var overholdt. Hvis vi havde haft mere tid, ville vi have implementeret/optimeret flere dele af projektet, herunder afprelning af knap ved lavpasfiltrering, implementering af sleep-funktioner, bedre brugerinterface bla. med amplitude-/fase diagram. Med mere tid havde vi lavet det sidste 3D-print, således at det ville være en rigtig håndholdt metaldetektor, fremfor "bare" spolen.
Alt i alt er det færdige produkt velfungerende og overholder alle kravspecifikationer på nær 1, hvilket vi stiller os tilfredse med.

6 Ansvarsområder

Fælles

- Vikling af spoler
- Fejlfinding og rettelser

Bilal

- 3D-print af spole
- Power Amplifier
- Spoleberegninger
- Energiberegninger

Sebastian

- C-programmering
- Test og simulation af C-program
- RX-filter
- Grafik til rapport (plots, kredsløbstegninger,indskrivning af rapport)

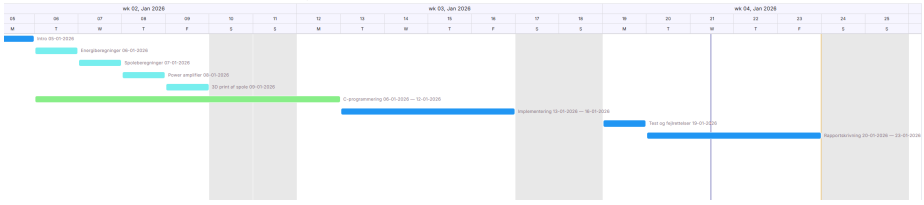
Oliver

- C-programmering
- Test og simulation af C-program
- Implementering af kredsløb

7 Appendix

7.1 Gantt kort

Figur 9: Her ses arbejdsfordelingen. Mørkeblå = Fælles, Grøn=Sebastian & Oliver, Turkis = Bilal



7.2 Mødereferater

Møde dag 2

- Bilal starter på energiberegninger, samt opstart på beregninger vedrørende spolerne.
- Sebastian og Oliver foretager målinger på Arduino, for bedre at kunne estimere strømforbruget til energiberegningerne. Derefter starter de op på C-programmering.

Møde dag 3

- Sebastian og Oliver fortsætter med C-programmering, herunder moduler til DFT, DSP og tilstandsmaskine, samt generel struktur på programmet.
- Bilal fortsætter med at arbejde på spolen, og får startet på at lave en prototype til Power amplifieren.

Møde dag 5

- Bilal fortsætter med simulation af Power Amplifier, samt 3D tegning af spolen.
- Sebastian og Oliver fortsætter med C-programmering, herunder moduler til PWM, OLED-display, ADC. Efterfølgende påbegyndes modul til buzzer (til lydindikation af fase/amplitude), samt opsætning af sleep-funktion i Arduino for strømbesparelse. Desuden laves der et python-script til simulering af data som ligges i flashhukommelse på Atmega2560'eren, og det kontrolleres om DFT-algoritmen samt display og DSP virker som ønsket, ved sammenligning mellem Atmega2560 output og resultatet fra Python-scriptet.

Møde dag 6

Planen for idag er at vi skal have testet C koden for sig, samt testet Power Amplifier for sig. Hvis begge ting virker enkeltvis, sættes de sammen og testes. Efterfølgende skal vi vikle spolerne, samt have opdateret C-koden til at kunne skelne imellem metaller.

Møde dag 7

Idag har vi fælles foretaget målinger på spolen, samt viklet buck-spolen tilstrækkeligt ud, så vi får udlignet det uønskede felt. Samt fælles kontrol af Power Amplifier. Alt er testet OK, således at der imorgen kan påbegyndes filtrering og forstærkning af spolens output, så vi snarest muligt kan sætte de analoge og digitale dele sammen, og teste det samlede system.

Møde dag 8

- Sebastian og Oliver laver et modul i C koden til sleep-funktion for strømbesparelse. Herefter forbereder de operationsforstærker kredsløbet til imorgen.
- Bilal tegner 3D printet til stangen til metaldetektoren.

7.3 Kodebilag

7.3.1 ADC.c

Listing 1: ADC modul (ADC.c)

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "config.h"
#include "DFT.h"
#include "test_mode.h"

// Two N-sample buffers, double buffer
volatile uint16_t adc_buf[2][ADC_BLOCK_N];
volatile uint8_t adc_buf_full[2] = {0, 0};

// DFT results - DEFINED here (extern in DFT.h)
volatile z_struct dft_latched[2];
volatile uint8_t  dft_ready[2] = {0, 0};

static volatile uint8_t adc_cur_buf = 0;     // 0 or 1
static volatile uint16_t adc_cur_idx = 0;    // 0 - N-1

volatile uint8_t sampling_enabled = 1;

// Coil pin setup
#define COIL_DDR  DDRB
#define COIL_PORT PORTB
#define COIL_PIN  PB5

void timer1_init_8kHz(void)
{
    COIL_DDR |= (1 << COIL_PIN);

    TCCR1A = 0;
    TCCR1B = (1 << WGM12); // CTC mode
    OCR1A = (F_CPU / (8UL * FS_HZ)) - 1; // 249 for 8 kHz

    TIMSK1 = (1 << OCIE1A); // enable compare match ISR
    TCCR1B |= (1 << CS11);  // prescaler = 8, start timer
}

static uint8_t coil_div = 0; // divider to get 2 kHz from 8 kHz
static volatile uint8_t wait_for_rising = 1;  // Start synchronized

ISR(TIMER1_COMPA_vect)
{
```

```
    if (!sampling_enabled)
        return;

    uint8_t coil_rising = 0;
    // Toggle coil every 2nd tick: 8000/2 = 4000 toggles/s = 2000 Hz
    if (++coil_div >= 2) {
        coil_div = 0;
        COIL_PORT ^= (1 << COIL_PIN);
        // Check if we just set it HIGH (rising edge)
        if (COIL_PORT & (1 << COIL_PIN)) {
            coil_rising = 1;
        }
    }

    // If waiting for sync, only start on rising edge
    if (wait_for_rising) {
        if (coil_rising) {
            wait_for_rising = 0;  // Start sampling
        } else {
            return;  // Skip until rising edge
        }
    }

    // Normal sampling
    ADCSRA |= (1 << ADSC);
}

void adc_init(void)
{
    ADMUX = (1 << REFS0); // AVcc reference, ADC0
    ADCSRA = (1 << ADEN) | (1 << ADIE) | (1 << ADPS2) | (1 << ADPS1); // prescal
    DIDR0 |= (1 << ADC0D); // disable digital input on ADC0 pin
}

ISR(ADC_vect)
{
    int16_t sample;
    uint16_t s;

#if TEST_MODE_ENABLED
    // Use simulated test signal instead of real ADC
    sample = test_get_sample(adc_cur_idx);
    s = (uint16_t)(sample + 512);  // convert back to 0-1023 for buffer
#else
    // Real ADC reading
    s = ADC;
```

```
    sample = (int16_t)s - 512; // center around 0
#endif

    uint8_t buf = adc_cur_buf;
    uint16_t i = adc_cur_idx;

    // Reset DFT at start of each block
    if (i == 0) {
        DFT_reset();
    }

    DFT_accum(sample, i);

    // Store raw sample
    adc_buf[buf][i] = s;

    i++;

    if (i >= ADC_BLOCK_N) {
        // Latch completed DFT result
        dft_latched[buf] = DFT_get();
        dft_ready[buf] = 1;
        adc_buf_full[buf] = 1;

        // Swap buffer
        buf ^= 1;
        i = 0;

        wait_for_rising = 1;
    }

    adc_cur_buf = buf;
    adc_cur_idx = i;
}
```

7.3.2 ADC.h

Listing 2: ADC header (ADC.h)

```
#ifndef ADC_H
#define ADC_H

#include <stdint.h>
#include "config.h"
#include "DFT.h"

// Double buffer - each with ADC_BLOCK_N samples
extern volatile uint16_t adc_buf[2][ADC_BLOCK_N];
extern volatile uint8_t  adc_buf_full[2];

// Sampling on/off flag (used in timer ISR)
extern volatile uint8_t sampling_enabled;

// Configure Timer1 for 8 kHz interrupts and 2 kHz coil drive
void timer1_init_8kHz(void);

// Configure ADC (single-conversion mode, interrupt enabled)
void adc_init(void);

#endif
```

7.3.3 button.c

Listing 3: Button modul (button.c)

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "button.h"
#include "config.h"

// Button pins on PORTE (INT4 and INT5 for Mega 2560)
// INT4 = PE4 = Arduino Pin 2
// INT5 = PE5 = Arduino Pin 3
#define BTN_RUN_PIN   PE4  // INT4 - Arduino Pin 2
#define BTN_PWR_PIN   PE5  // INT5 - Arduino Pin 3

volatile bool btn_run_pressed = false;
volatile bool btn_pwr_pressed = false;

void buttons_init(void)
{
    // Set as inputs
    DDRE &= ~((1 << BTN_RUN_PIN) | (1 << BTN_PWR_PIN));

    // Enable internal pull-ups (buttons connect to GND)
    PORTE |= (1 << BTN_RUN_PIN) | (1 << BTN_PWR_PIN);

    // Configure INT4 and INT5 for falling edge (button press)
    EICRB |= (1 << ISC41) | (1 << ISC51);  // Falling edge for both
    EICRB &= ~((1 << ISC40) | (1 << ISC50));

    // Enable INT4 and INT5
    EIMSK |= (1 << INT4) | (1 << INT5);
}

// INT4 - Run button (Pin 2)
// Can this be made using a capacitor (hardware) instead of polling?
ISR(INT4_vect)
{
    _delay_ms(50);  // Simple debounce delay
    // Check if button still pressed (pin is LOW)
    if (!(PINE & (1 << BTN_RUN_PIN))) {
        btn_run_pressed = true;
    }
}
```

```
// INT5 - Power button (Pin 3)
ISR(INT5_vect)
{
    _delay_ms(50);  // Simple debounce delay
    // Check if button still pressed (pin is LOW)
    if (!(PINE & (1 << BTN_PWR_PIN))) {
        btn_pwr_pressed = true;
    }
}

void buttons_debounce_tick(void)
{
    // Not needed with delay-based debounce
}
```

7.3.4 button.h

Listing 4: Button header (button.h)

```
#ifndef BUTTON_H
#define BUTTON_H

#include <stdint.h>
#include <stdbool.h>

// Button states (directly usable in main)
extern volatile bool btn_run_pressed;
extern volatile bool btn_pwr_pressed;

// Initialize button pins and interrupts
void buttons_init(void);

// Call periodically to handle debounce (from main loop)
void buttons_debounce_tick(void);

#endif
```

7.3.5 Buzzer.c

Listing 5: Buzzer modul (Buzzer.c)

```
#include <avr/io.h>
#include <stdint.h>

#define F_CPU 16000000UL
#define LOW_THRESHOLD 50 // No sound for amplitudes below this threshold

// OC2B = PH6 (Arduino Mega pin 9)
// OC2A as TOP (frequency)

void buzzer_init(void)
{
    // PH6 (pin 9) as output
    DDRH |= (1 << PH6);

    // Timer2: Fast PWM, TOP = OCR2A
    // WGM22:0 = 7 -> Fast PWM, OCR2A as TOP
    TCCR2A = (1 << WGM21) | (1 << WGM20);
    TCCR2B = (1 << WGM22);

    // Non-inverting PWM OC2B (clear on compare match)
    TCCR2A |= (1 << COM2B1);

    // Stop timer until sound is wanted
    TCCR2B &= ~((1 << CS22) | (1 << CS21) | (1 << CS20));

    // Initially mute the buzzer
    OCR2B = 0;
}

// Tone frequency (Hz)
// f_out = F_CPU / (prescaler * (1 + OCR2A))
void buzzer_set_frequency(uint16_t freq)
{
    if (freq == 0) return;

    // Prescaler = 64
    uint32_t top = (F_CPU / (64UL * freq)) - 1;
    if (top > 255) top = 255;

    OCR2A = (uint8_t)top;
}
```

```
// Set volume by adjusting the duty cycle (0-255)
void buzzer_set_volume(uint8_t vol)
{
    OCR2B = vol;    // 0 = mute, 255 = max
}

void buzzer_on(void)
{
    // Start timer with prescale = 64
    TCCR2B = (TCCR2B & ~((1 << CS22) | (1 << CS21) | (1 << CS20))) | (1 << CS22)
}

void buzzer_off(void)
{
    // Stop timer
    TCCR2B &= ~((1 << CS22) | (1 << CS21) | (1 << CS20));
    OCR2B = 0;
}

// Amplitude -> volumen, Fase -> tone
void update_buzzer(uint16_t amp, int16_t phase)
{
    if (amp < LOW_THRESHOLD) // To stop the buzzer from making sound below the t
    {
        buzzer_off();
        return;
    }

    buzzer_on();
    // We represent different phases with different frequencies (tones) for the
    if (phase > 30)                 // Non-iron
        buzzer_set_frequency(1200);
    else if (phase < -30)           // Iron
        buzzer_set_frequency(300);
    else
        buzzer_set_frequency(700);

    // Volume of the buzzer to be adjusted by the amplitude of DFT
    const uint16_t AMP_MAX = 1000;   // This is the highest amplitude we want to
    if (amp > AMP_MAX) amp = AMP_MAX;

    uint8_t volume = (uint32_t)amp * 255 / AMP_MAX;
    buzzer_set_volume(volume);
}
```

7.3.6 Buzzer.h

Listing 6: Buzzer header (Buzzer.h)

```
#ifndef Buzzer_H
#define PWM_H_

#include <stdint.h>

void buzzer_init(void);
void buzzer_set_frequency(uint16_t freq);
void buzzer_set_volume(uint8_t vol);
void buzzer_on(void);
void buzzer_off(void);
void update_buzzer(uint16_t amp, int16_t phase);

#endif
```

7.3.7 config.h

Listing 7: Config header (config.h)

```
#ifndef config_H
#define config_H

#define F_CPU           16000000UL
#define FS_HZ           8000        // sample rate
#define COIL_HZ         2000        // drive frequency
#define ADC_BLOCK_N     64          // samples per DSP block (Fs/N = 125 Hz bin)

#endif
```

7.3.8 data.h

Listing 8: Data header (data.h)

```
/*
 * 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
 */
#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, 0
0xF0, 0xF0, 0xF0, 0xE0, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0xFF, 0xFF, 0xFF, 0x
0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x80, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x07, 0x
0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x07, 0x03, 0x

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

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

{0x00, 0x00, 0x00, 0xE0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0
0xF0, 0xF0, 0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x81, 0xC1, 0x
0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xE1, 0xFF, 0xFF, 0xFF, 0x7F, 0x00, 0x
0x00, 0x00, 0x00, 0x81, 0x83, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x
0xFF, 0xFF, 0xFF, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x0F, 0x
0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x07, 0x00, 0x

{0x00, 0x00, 0x00, 0xE0, 0xF0, 0xF0, 0xF0, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0
0xF0, 0xF0, 0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x
0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xE0, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x
0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x
```

```
0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0C, 0x0F, 0x0F, 0x0F, 0x07, 0x00, 0x

{0x00, 0x00, 0x00, 0xE0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0
0xF0, 0xF0, 0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x
0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC1, 0x81, 0x00, 0x
0x00, 0x00, 0x00, 0x81, 0x83, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x
0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x0F, 0x
0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x07, 0x00, 0x

{0x00, 0x00, 0x00, 0xE0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0
0xF0, 0xF0, 0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x
0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC1, 0x81, 0x00, 0x
0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0x87, 0x03, 0x03, 0x03, 0x03, 0x03, 0x
0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x0F, 0x
0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x07, 0x00, 0x

{0x00, 0x00, 0x00, 0xE0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0
0xF0, 0xF0, 0xF0, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x01, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0C, 0x0F, 0x0F, 0x0F, 0x07, 0x00, 0x

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

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

{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x3C, 0x7E, 0x7E, 0x7E, 0x7E, 0x3C, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0xF8, 0xF8, 0xF8, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x01, 0x01, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
};
```

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

// 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,
        0x80, 0x80, 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, 0x80, 0xC0, 0xE0, 0xF0, 0xF8, 0xFC, 0xF8, 0xE0,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x80, 0x80, 0x00, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x80,
        #if (SSD1306_LCDHEIGHT * SSD1306_LCDWIDTH > 96*16){
        0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x00, 0x00,
        0x80, 0xFF, 0xFF, 0x80, 0x80, 0x00, 0x80, 0x80, 0x00, 0x80, 0x80, 0x80,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x00, 0x00, 0x8C, 0x8E, 0x84,
        0xF8, 0xF8, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
        0x00, 0xE0, 0xFC, 0xFE, 0xFF, 0xFF, 0xFF, 0x7F, 0xFF, 0xFF, 0xFF, 0xFF,
```

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFE,
0x01, 0x01, 0x83, 0xFF, 0xFF, 0x00, 0x00, 0x7C, 0xFE, 0xC7, 0x01, 0x01,
0xFF, 0xFF, 0x00, 0x38, 0xFE, 0xC7, 0x83, 0x01, 0x01, 0x01, 0x83, 0xC7,
0x01, 0xFF, 0xFF, 0x01, 0x01, 0x00, 0xFF, 0xFF, 0x07, 0x01, 0x01, 0x01,
0x80, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0x7F, 0x00, 0x00, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x03, 0x0F, 0x3F, 0x7F, 0x7F, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0x8F, 0x9F, 0xBF, 0xFF, 0xFF, 0xC3, 0xC0, 0xF0, 0xFF, 0xFF, 0xFF, 0xFF,
0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xF8, 0xF8, 0xF0, 0xF0, 0xE0, 0xC0, 0x00,
0x03, 0x03, 0x01, 0x03, 0x03, 0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x03,
0x03, 0x01, 0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x03, 0x03, 0x01, 0x01,
0x00, 0x03, 0x03, 0x00, 0x00, 0x00, 0x03, 0x03, 0x00, 0x00, 0x00, 0x00,
0x03, 0x03, 0x03, 0x03, 0x03, 0x01, 0x00, 0x00, 0x00, 0x01, 0x03, 0x01,
0x03, 0x01, 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,
0x87, 0xC7, 0xF7, 0xFF, 0xFF, 0x1F, 0x1F, 0x3D, 0xFC, 0xF8, 0xF8, 0xF8,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7F, 0x3F, 0x0F, 0x07, 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, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x30,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xC0, 0xFE, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0x0F, 0x07, 0x1F, 0x7F, 0xFF, 0xFF, 0xF8, 0xF8, 0xFF, 0xFF, 0xFF, 0xFF,
0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xFC, 0xFE, 0xFC, 0x0C, 0x06, 0x06, 0x0E, 0xFC, 0xF8, 0x00, 0x00,
0x06, 0x06, 0x06, 0x0C, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0xFE, 0xFE, 0x00,
0xFE, 0xFC, 0x00, 0x18, 0x3C, 0x7E, 0x66, 0xE6, 0xCE, 0x84, 0x00, 0x00,
0x06, 0xFC, 0xFE, 0xFC, 0x0C, 0x06, 0x06, 0x06, 0x00, 0x00, 0xFE, 0xFE,
0xFC, 0x4E, 0x46, 0x46, 0x46, 0x4E, 0x7C, 0x78, 0x40, 0x18, 0x3C, 0x76,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x01, 0x07, 0x0F, 0x1F, 0x1F, 0x3F, 0x3F, 0x3F,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x0F, 0x0F, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x0F, 0x00, 0x00,
0x18, 0x18, 0x0C, 0x06, 0x0F, 0x0F, 0x0F, 0x00, 0x00, 0x01, 0x0F, 0x0E,
0x07, 0x01, 0x00, 0x04, 0x0E, 0x0C, 0x18, 0x0C, 0x0F, 0x07, 0x00, 0x00,
0x00, 0x0F, 0x0F, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x0F,
0x07, 0x0C, 0x0C, 0x18, 0x1C, 0x0C, 0x06, 0x06, 0x00, 0x04, 0x0E, 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, 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
};*/
```

7.3.9 DFT.c

Listing 9: DFT modul (DFT.c)

```
#include <avr/io.h>
#include <stdint.h>
#include <avr/interrupt.h>
#include "DFT.h"

static z_struct z;  // persistent accumulator

void DFT_reset(void) {
    z.re = 0;
    z.im = 0;
}

z_struct DFT_get(void) {
    return z;
}

// Goertzel-style accumulation for bin k=16 (2kHz at 8kHz sample rate)
// cos/sin values at k=16 cycle through: 1,0,-1,0 and 0,-1,0,1
void DFT_accum(int16_t sample, uint16_t n) {
    switch (n & 3) {
        case 0: z.re += sample; break;
        case 1: z.im -= sample; break;
        case 2: z.re -= sample; break;
        case 3: z.im += sample; break;
    }
}
```

7.3.10 DFT.h

Listing 10: DFT header (DFT.h)

```
#ifndef DFT_H
#define DFT_H

#include <stdint.h>
#include "config.h"

typedef struct {
    int32_t re;
    int32_t im;
} z_struct;

// Declared in ADC.c - extern only here
extern volatile z_struct dft_latched[2];
extern volatile uint8_t  dft_ready[2];

// Accumulate sample into DFT (Goertzel at bin k=16 for 2kHz)
void DFT_accum(int16_t sample, uint16_t n);

// Reset accumulator
void DFT_reset(void);

// Get current accumulator value
z_struct DFT_get(void);

#endif
```

7.3.11 DSP.c

Listing 11: DSP modul (DSP.c)

```
#include "config.h"
#include <avr/io.h>
#include <avr/pgmspace.h>
#include "DSP.h"
#include <math.h>
#define RAD2DEG 57.295780 // 180/pi

int32_t DSP_fast_magnitude(int32_t re,int32_t im){
    int64_t acc = (int64_t)re * re +(int64_t)im * im;
    return (int32_t)sqrt((double)acc);
}

/*
int32_t DSP_fast_magnitude(int32_t re, int32_t im) {
    if (re < 0) re = -re;
    if (im < 0) im = -im;

    int32_t max_val, min_val;
    if (re > im) {
        max_val = re;
        min_val = im;
    } else {
        max_val = im;
        min_val = re;
    }

    // Approximation: max + (3/8)*min      max + 0.375*min
    // Using shift: min*3/8 = (min >> 2) + (min >> 3)
    return max_val + (min_val >> 2) + (min_val >> 3);
}
*/
/*
// Fast atan2 approximation returning degrees (-180 to +180)
// Uses CORDIC-style polynomial approximation
// Max error ~0.3 degrees
int16_t DSP_fast_atan2_deg(int32_t im, int32_t re) {
    int16_t angle;
    int32_t abs_re = (re < 0) ? -re : re;
    int32_t abs_im = (im < 0) ? -im : im;

    // Handle zero case
    if (re == 0 && im == 0) return 0;
```

```
    // Compute atan(y/x) for |y| <= |x| using approximation:
    // atan(z)      z * 45 degrees (for small z, scaled)
    // More accurate: atan(z)      z * 45 * (1 - 0.28 * z^2) degrees

    int32_t ratio;
    int16_t base_angle;

    if (abs_re >= abs_im) {
        // |angle| <= 45 degrees from x-axis
        // ratio = im/re scaled by 1024
        ratio = (im * 1024) / re;
        // angle      ratio * 45 / 1024 = ratio * 45 >> 10
        angle = (int16_t)((ratio * 45) >> 10);

        // Adjust for quadrant
        if (re < 0) {
            angle = (im >= 0) ? (180 + angle) : (-180 + angle);
        }
    } else {
        // |angle| > 45 degrees from x-axis
        // Use atan(y/x) = 90 - atan(x/y) for y > 0
        ratio = (re * 1024) / im;
        base_angle = (int16_t)((ratio * 45) >> 10);

        if (im > 0) {
            angle = 90 - base_angle;
        } else {
            angle = -90 - base_angle;
        }
    }

    return angle;
}
*/

int16_t DSP_fast_atan2_deg(int32_t im, int32_t re) {
    if (re == 0 && im == 0) return 0;

    // atan2 returns radians, convert to degrees
    double radians = atan2((double)im, (double)re);
    double degrees = radians * RAD2DEG; // RAD2DEG is 180/pi. Precalculated and

    return (int16_t)degrees;
}
```

7.3.12 DSP.h

Listing 12: DSP header (DSP.h)

```
#ifndef DSP_H
#define DSP_H

#include <stdint.h>

int16_t DSP_apply_hanning(int16_t sample, uint8_t n);
int16_t DSP_IIR_filter(int16_t x, int16_t y_prev);

// Fast integer amplitude approximation (avoids sqrt)
int32_t DSP_fast_magnitude(int32_t re, int32_t im);

// Fast integer atan2 approximation
// Returns phase in degrees (-180 to +180)
int16_t DSP_fast_atan2_deg(int32_t im, int32_t re);

#endif
```

7.3.13 I2C.c

Denne fil er lånt fra et tidligere kursus og er udarbejdet af Ole Schultz, DTU.

Listing 13: I2C modul (I2C.c)

```
/*
 * 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 no
        return 0;                        /* Return 0 to indicate start condition
        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 rece
        return 1;                        /* Return 1 to indicate ack received */
        if(status==0x20){                /* Check for SLA+W transmitted &nack rec
```

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

}
        else
        return 3;                       /* Else return 3 to indicate SLA+W faile
}
 /** 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
        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 rece
        return 1;                       /* Return 1 to indicate ack received */
        if(status==0x20)                /* Check for SLA+R transmitted &nack rec
        return 2;                       /* Return 2 to indicate nack received */
        else
        return 3;                       /* Else return 3 to indicate SLA+W faile
}
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 recei
        return 0;                       /* Return 0 to indicate ack received */
        if(status==0x30)                /* Check for data transmitted &nack rece
        return 1;                       /* Return 1 to indicate nack received */
        else
        return 2;                       /* Else return 2 for data transmission f
}
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 */
}
```

7.3.14 I2C.h

Denne fil er lånt fra et tidligere kursus og er udarbejdet af Ole Schultz, DTU.

Listing 14: I2C header (I2C.h)

```
/**
 * 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<<TWPS
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_ */
```

7.3.15 Main.c

Listing 15: Main, hovedprogram (main.c)

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdbool.h>
#include <stdio.h>
#include <util/delay.h>
#include <stdlib.h>
#include "config.h"
#include "ADC.h"
#include "DSP.h"
#include "DFT.h"
#include "button.h"
#include "test_mode.h"
#include "ssd1306.h"
#include "I2C.h"
#include "Buzzer.h"
#include "sleep.h"

#define AVG_SIZE 32
int32_t amp_history[AVG_SIZE] = {0};
int16_t phase_history[AVG_SIZE] = {0};
uint8_t avg_index = 0;
int N = AVG_SIZE;

// Result structure
typedef struct {
    int32_t amp;
    int16_t phase_deg;  // -180 to +180 degrees
    int32_t re;
    int32_t im;
} results_t;

// Process DFT result from buffer b
results_t calc_results(uint8_t b) {
    results_t res;
    z_struct z = dft_latched[b];

    // Normalize by block size
    res.re = z.re / ADC_BLOCK_N;
    res.im = z.im / ADC_BLOCK_N;

    // Fast magnitude (no sqrt)
    res.amp = DSP_fast_magnitude(res.re, res.im);
```

```
    // Fast phase in degrees
    res.phase_deg = DSP_fast_atan2_deg(res.im, res.re);

    return res;
}

typedef enum {
    STATE_IDLE,
    STATE_CALIBRATING,
    STATE_RUNNING
} system_state_t;

int main(void)
{
    system_state_t state = STATE_CALIBRATING;

    int32_t baseline_amp = 0;
    int16_t baseline_phase = 0;
    bool system_calibrated = false;
    results_t res = {0, 0, 0, 0};

    // Display buffer for numbers
    char str_buf[32];

    // Initialize peripherals
    cli();
    timer1_init_8kHz();
    adc_init();
    buttons_init();
    I2C_Init();
    buzzer_init();
    sei();

    InitializeDisplay();

    while (1)
    {
        // Handle button presses (clear flags after reading)
        if (btn_run_pressed) {
            btn_run_pressed = false;
            // RUN always toggles between IDLE and RUNNING
            if (state == STATE_IDLE && system_calibrated) {
                clear_display();
                state = STATE_RUNNING;
            } else if (state == STATE_RUNNING) {
```

```
                clear_display();
                state = STATE_IDLE;
            }
        }

        if (btn_pwr_pressed) {
            btn_pwr_pressed = false;
#if TEST_MODE_ENABLED
            // In test mode: cycle through test signals (while running)
            if (state == STATE_RUNNING) {
                test_next_signal();
            } else {
                // Recalibrate if not running
                clear_display();
                system_calibrated = false;
                state = STATE_CALIBRATING;
            }
#else
            // Normal mode, always recalibrate
            clear_display();
            system_calibrated = false;
            state = STATE_CALIBRATING;
#endif
        }

        // Check both DFT buffers for ready data
            for (uint8_t b = 0; b < 2; b++) {
                if (dft_ready[b]) {
                    cli();
                    dft_ready[b] = 0;
                    sei();
                    res = calc_results(b);
                }
            }

        // Always tick debounce (runs every loop iteration)
        buttons_debounce_tick();

        switch (state)
        {
        case STATE_CALIBRATING:
    sampling_enabled = 1;
    sendStrXY("Calibrating...", 0, 0);

    // Use static variables to accumulate across loop iterations
    static uint8_t cal_count = 0;
```

```
    static int64_t cal_amp_sum = 0;
    static int64_t cal_phase_sum = 0;
    static bool cal_started = false;

    #define CAL_SAMPLES 32

    // Only accumulate when we got NEW data (check if res changed)
    if (res.amp > 0) {
        if (!cal_started) {
            // Reset sums on first valid sample
            cal_count = 0;
            cal_amp_sum = 0;
            cal_phase_sum = 0;
            cal_started = true;
        }

        cal_amp_sum += res.amp;
        cal_phase_sum += res.phase_deg;
        cal_count++;

        // Show progress
        snprintf(str_buf, sizeof(str_buf), "%d/%d   ", cal_count, CAL_SAMPLES);
        sendStrXY(str_buf, 1, 0);

        if (cal_count >= CAL_SAMPLES) {
            baseline_amp = cal_amp_sum / CAL_SAMPLES;
            baseline_phase = cal_phase_sum / CAL_SAMPLES;
            system_calibrated = true;
            cal_started = false;

            clear_display();
            state = STATE_IDLE;
        }
    }
    break;

        case STATE_IDLE:
            sampling_enabled = 0;

            sendStrXY("IDLE", 0, 0);
            sendStrXY("RUN: Start", 2, 0);
            sendStrXY("PWR: Recal", 3, 0);
            break;

case STATE_RUNNING:
    sampling_enabled = 1;
```

```
        // Compute deltas from baseline
        int32_t delta_amp = res.amp - baseline_amp;
        int16_t delta_phase = res.phase_deg - baseline_phase;

        // Wrap phase difference to -180..+180
        if (delta_phase > 180) delta_phase -= 360;
        if (delta_phase < -180) delta_phase += 360;

        // Store in circular buffer
        amp_history[avg_index] = delta_amp;
        phase_history[avg_index] = delta_phase;
        avg_index = (avg_index + 1) % AVG_SIZE;

        // Compute averages
        int32_t sum_amp = 0;
        int32_t sum_phase = 0;
        for (uint8_t k = 0; k < AVG_SIZE; k++) {
            sum_amp += amp_history[k];
            sum_phase += phase_history[k];
        }

        int32_t delta_amp_avg = sum_amp / AVG_SIZE;
        int32_t delta_phase_avg = sum_phase / AVG_SIZE;

        // Display amplitude
        sendStrXY("AMP:", 0, 0);
        snprintf(str_buf, sizeof(str_buf), "%ld---", delta_amp_avg);
        sendStrXY(str_buf, 0, 6);

        // Display phase
        sendStrXY("PHS:", 1, 0);
        snprintf(str_buf, sizeof(str_buf), "%ld-deg---", delta_phase_avg);
        sendStrXY(str_buf, 1, 6);

        update_buzzer((uint16_t)abs(delta_amp_avg), delta_phase_avg);

        //go_to_sleep(); // To save power

    #if TEST_MODE_ENABLED
        sendStrXY("SIM:", 2, 0);
        sendStrXY((char*)test_get_signal_name(), 2, 6);
    #endif

        break;
```

```
            default:
                state = STATE_CALIBRATING;
                break;
            }
        }
    }
```

7.3.16 Sleep.c

Listing 16: Sleep modul (sleep.c)

```
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <avr/interrupt.h>
#include "sleep.h"

ISR(WDT_vect){ // To wake up CPU from watchdog timer
}

void go_to_sleep(void){
    cli();
    wdt_reset();
    WDTCSR = (1<<WDCE) | (1<<WDE); // Allow changes to Watchdog timer

//    WDTCSR = (1<<WDIE) | (1<<WDP1) | (1<<WDP0); // Watchdog timer, approx 125
    WDTCSR = (1<<WDIE) | (1<<WDP1); // Watchdog timer interrupt mode, approx 64
    sei();
    set_sleep_mode(SLEEP_MODE_PWR_DOWN); // Power-down sleep schedule. Woken up
    sleep_enable();
    sleep_cpu();
    // The CPU is sleeping at this point, only to be waken up by interrupt.
    // Once CPU is waken by interrupt, sleep_disable() is called, and system is
    sleep_disable();
}
```

7.3.17 Sleep.h

Listing 17: Sleep header (sleep.h)

```
#define sleep .h
#ifndef sleep.h



void go_to_sleep(void)

#endif
```

7.3.18 ssd1306.c

Denne fil er lånt fra et tidligere kursus og er udarbejdet af Ole Schultz, DTU.

Listing 18: OLED driver (ssd1306.c)

```
/**
 * ssd1306.c
 *
 * Created: 03-01-2018 17:33:26
 *  Author: osc
 *this modules purpose is to initiate and control the OLED display SSD1306 drive
 *The plus can be connected to PIN 24 and the GND to PIN 26 and sda goes to PIN
 *http://microcontrolandos.blogspot.dk/2014/12/pic-ssd1306.html
 * SSD1306 Advanced information Matrix apr. 2008  rev. 1.1 www.solomon-systech.c
 *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 trie
        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

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 addres

        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 uint
{
        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))
                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);
}
```

7.3.19 ssd1306.h

Denne fil er lånt fra et tidligere kursus og er udarbejdet af Ole Schultz, DTU.

Listing 19: OLED header (ssd1306.h)

```
/*
 * ssd1306.h
 *based upon Adafruit_SSD1306 at github  -adjusted to C using i2c address 0x78 f
 * 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 t
  *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 uint
//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);
```

7.3.20 test_mode.c

Listing 20: Til simulation af data (test_mode.c)

```
#include "test_mode.h"
#include "config.h"

volatile test_signal_t current_test_signal = TEST_NONE;

// Simulated 2 kHz signals with different phase shifts
// At 8 kHz sample rate, 2 kHz = 4 samples per period
// Phase shift is achieved by offsetting the waveform

// Amplitude of test signal (arbitrary units, centered around 0)
#define TEST_AMPLITUDE 200

int16_t test_get_sample(uint16_t sample_index) {
    // Base 2 kHz sine wave: period = 4 samples at 8 kHz
    // Values for one period: {0, +A, 0, -A} approximates sine

    uint8_t phase_in_period;
    int16_t sample = 0;

    switch (current_test_signal) {
        case TEST_NONE:
            // Baseline signal - clean 2 kHz, no shift
            // sin(2   * 2000 * n / 8000) = sin( n /2)
            phase_in_period = sample_index & 3;  // mod 4
            switch (phase_in_period) {
                case 0: sample = 0; break;
                case 1: sample = TEST_AMPLITUDE; break;
                case 2: sample = 0; break;
                case 3: sample = -TEST_AMPLITUDE; break;
            }
            break;

        case TEST_FERROUS:
            // Ferrous metal: larger amplitude + positive phase shift
            // Shift by +1 sample     +90 degrees (exaggerated for visibility)
            phase_in_period = (sample_index + 1) & 3;  // shift +1
            switch (phase_in_period) {
                case 0: sample = 0; break;
                case 1: sample = TEST_AMPLITUDE + 50; break;
// stronger
                case 2: sample = 0; break;
                case 3: sample = -(TEST_AMPLITUDE + 50); break;
```

```
            }
            break;

        case TEST_NON_FERROUS:
            // Non-ferrous metal: larger amplitude + negative phase shift
            // Shift by -1 sample     -90 degrees (exaggerated for visibility)
            phase_in_period = (sample_index + 3) & 3;  // shift -1 (same as +3 m
            switch (phase_in_period) {
                case 0: sample = 0; break;
                case 1: sample = TEST_AMPLITUDE + 50; break;
// stronger
                case 2: sample = 0; break;
                case 3: sample = -(TEST_AMPLITUDE + 50); break;
            }
            break;

        default:
            sample = 0;
            break;
    }

    return sample;
}

void test_next_signal(void) {
    current_test_signal = (current_test_signal + 1) % TEST_COUNT;
}

const char* test_get_signal_name(void) {
    switch (current_test_signal) {
        case TEST_NONE:         return "BASELINE";
        case TEST_FERROUS:      return "FERROUS";
        case TEST_NON_FERROUS: return "NON-FERROUS";
        default:                return "UNKNOWN";
    }
}
```

7.3.21 test_mode.h

Listing 21: Test header (test_mode.h)

```
#ifndef TEST_MODE_H
#define TEST_MODE_H

#include <stdint.h>
#include <stdbool.h>

// Enable this to bypass real ADC and inject test signals
#define TEST_MODE_ENABLED 0

// Test signal types
typedef enum {
    TEST_NONE = 0,        // No metal (baseline)
    TEST_FERROUS,         // Simulated iron/steel (positive phase shift)
    TEST_NON_FERROUS,     // Simulated copper/aluminum (negative phase shift)
    TEST_COUNT
} test_signal_t;

// Current test signal (change with button in test mode)
extern volatile test_signal_t current_test_signal;

// Call this instead of real ADC reading in test mode
int16_t test_get_sample(uint16_t sample_index);

// Cycle to next test signal
void test_next_signal(void);

// Get name of current test signal for display
const char* test_get_signal_name(void);

#endif
```