

SLUTTRAPPORT

Prosjekt: Innebygde datasystemer for mekatronikk

THOMAS LØNNE STIANSEN
JAN LAKSESVELA HAUGSTAD
ADRIAN MATHIAS LERVIK LING

VEILEDER
Kristian Muri Knausgård



Figur 1: Peli Case med komponentene til delprosjektene

Sammendrag

Faget har delt prosjektet i tre deler, hvor hver del er dedikert til spesifikke aspekter relevant til innebygde datasystemer. Første delprosjekt tar for seg bruken av mikrokontroller og relevant kretsdesign, samt konfigurasjon av innebygde funksjoner og moduser som blant annet interrupt protokoller og pulsbreddmodulasjon. Andre delprosjekt fokuserer på CAN bus og kommunikasjonsprotokoller, samt bruk av biblioteker og grensesnitt. Fokuset på siste delprosjekt er på design av operativsystem, nærmere bestemt en distribusjon av Linux. Delprosjektene benytter tre forskjellige målenheter; Atmel mikrokontroller, Teensy mikrokontroller utviklingssystem og en Raspberry PI.

Figurer

1	Peli Case med komponentene til delprosjektene	1
2.1	Fast PWM Mode [6]	5
2.2	Phase Correct PWM Mode [6]	5
2.3	CAN bus differensiell sampling	6
2.4	Datamaskinens hardware og software arkitektur [28]	8
3.1	ATMEGA168-20PU Mikrokontroller pinout [6]	10
3.2	Linear Dropout Regulator [14]	11
3.3	Decoupling: Kondensator og spole [9]	11
3.4	Reset kondensator og diode [9]	11
3.5	Reset Bryter og seriemotstand [8]	11
3.6	Simulering i MATLAB, kode i A.5.1.1	12
3.7	Ballens fysikk avhengig av hvor den treffer paddle	16
3.8	Raspberry PI Systemarkitektur	16
3.9	Teensy skjema [24]	18
3.10	Teensy loddig	18
3.11	Caption	18
4.1	Oscilloskop, 25% Duty Cycle	22
4.2	Oscilloskop, 75% Duty Cycle	22
4.3	Kompileringssjekk, modifisert prosjektkode	23
4.4	OLED oppgave med bruk av bibliotek fra Adafruit	23
4.5	"Receive" og "Transmit" CAN meldinger vist henholdsvis i PCAN-View og Serial Monitor	23
5.1	Oversikt over programmeringsoppsett	26
A.1	LDO regulator krets, tegnet i Autodesk Eagle	31
A.2	Mikrokontrollerkrets, tegnet i Autodesk Eagle	31
A.3	Krets over SK Pang kort og Teensy 3.6 [12]	32
A.4	Krets til Raspberry Pi [27]	33
A.5	Oppkobling av mikrokontrollerkrets	59
A.6	Oppkobling av to Teensy 3.6 med CAN bus	59

Tabeller

2.1	Spenningsstørskler til digitale signaler for ATMEGA168-20PU [6]	3
3.1	Utstyr utlevert for delprosjekt 1	10
3.2	Pinner for pin change interrupt [6]	13
3.3	Utstyr utlevert for delprosjekt 2	14
4.1	Komponentvalg for delprosjekt 1	20
4.2	Pin-relasjoner fra Atmel ICE User Guide [7]	21

Innhold

Sammendrag	i
List of Figures	ii
List of Tables	iii
1 Introduksjon	1
2 Teori	2
Electromagnetic Compatibility (EMC)	2
Electrostatic Discharge (ESD)	2
Avkoblingskondensatorer (Decoupling Capacitors)	2
Low Dropout Regulator (LDO-regulator)	2
In-System Programming (ISP)	2
Fuse bits and Register bytes	3
I/O Pins	3
Sinking og sourcing	4
Interrupt protokoller	4
Pulsbreddemodulasjon og prescaler	4
ADC	6
CAN bus (Controlled Area Network)	6
CAN Meldingsoverføring	7
Buildroot	7
Operativsystem (OS)	8
Nettverk, IP-addresser, DHCP og SSH	8
3 Metode	10
3.1 Delprosjekt 1	10
3.1.1 Utstyr	10
3.1.2 Forsyningskrets	10
3.1.3 Mikrokontroller krets	11
3.1.4 Mikrokontroller programmering	12
3.1.5 PWM og Soft Blink	12
3.1.6 Pin change interrupt og tilstandsbytting av lysdiode	13
3.1.7 ADC og variabel PWM settpunkt ved potensiometer	13
3.2 Delprosjekt 2	14
3.2.1 Utstyr og digitale forberedelser	14
3.2.2 CAN mellom Teensy og PC	14
3.2.3 Sending og mottak av CAN-meldinger	15
3.2.4 OLED-display og CAN	15
3.2.5 CAN mellom Teensy og Teensy: Pong!	15
3.3 Delprosjekt 3	16
3.3.1 Oppsett	16
3.3.2 Kommunikasjon mellom to CAN-noder	17

4 Resultater	20
4.1 Delprosjekt 1	20
4.1.1 Komponentvalg	20
4.1.2 Oppkoblinger	21
4.1.3 Microchip Studio til MCU	21
4.1.4 PWM og Soft Blink	21
4.1.5 Pin change interrupt og tilstandsbytting av lysdiode	22
4.1.6 ADC og variabel PWM settpunkt ved potensiometer	22
4.2 Delprosjekt 2	23
4.3 Delprosjekt 3	24
5 Diskusjon	25
5.1 Software Arkitektur	25
5.2 Delprosjekt 1	25
5.2.1 Komponentvalg	25
5.2.2 Programmering	26
5.2.3 Source vs. Sink	26
5.2.4 PWM og Soft Blink	26
5.2.5 Interrupt	27
5.2.6 ADC	27
5.3 Delprosjekt 2	27
5.4 Delprosjekt 3	28
6 Konklusjon	29
A Appendix	30
A.1 Github Repository	30
A.2 Videoer	30
A.2.1 Delprosjekt 1	30
A.2.2 Delprosjekt 2	30
A.2.3 Delprosjekt 3	30
A.3 Koblingsskjemaer	31
A.3.1 Delprosjekt 1	31
A.3.2 Delprosjekt 2	32
A.3.3 Delprosjekt 3	33
A.4 C++ Kode	34
A.4.1 Delprosjekt 1	34
A.4.2 Delprosjekt 2	41
A.5 MATLAB kode	52
A.5.1 Delprosjekt 1	52
A.5.2 Delprosjekt 3	53
A.6 Fysisk oppkobling	59
A.6.1 Delprosjekt 1	59
A.6.2 Delprosjekt 2	59
Bibliografi	60

1 Introduksjon

I industrien er det viktig å kjenne til forskjellige innebygde datasystemer og deres egenskaper. Prosjektene tar for seg kretsdesign med bruk av en Atmel mikrokontroller, samt komponenter for å få den til å fungere optimalt. Det tar for seg strømforsyning, støyreduksjon, pullup motstander og mye mer; det gir en god introduksjon til design av innebygde datasystemer som er skreddersydd for spesifikke applikasjoner. For slike systemer er en sentral del optimalisering, hvor det er ønskelig å oppnå ønsket resultat på enklest mulig måte. Prosjektet gir også en god introduksjon for dette gjennom software, hvor det blir gått gjennom bitvise operasjoner og konfigurasjoner for innebygde funksjoner. Det blir også utforsket forskjellige moduser som er relevante for "real-time" applikasjoner som er tidskritiske. Prosjektene tar også for seg kommunikasjonsprotokoller som CAN bus, hvor det blir benyttet forskjellige grensesnitt for å bruke dem til forskjellige ting. Avslutningsvis tilbyr prosjektet kunnskap for utvikling av operativsystem, hvor det går gjennom hvordan man kan lage en distribusjon basert på Linux-kernelen.

2 Teori

Electromagnetic Compatibility (EMC)

Ved design av elektriske kretser er det viktig å ta hensyn til EMC, spesielt med komplekse systemer som er utsatt for støy. EMC er en betegnelse på hvor godt systemet er tilpasset miljøet og den elektromagnetisk støyende verden. Selv om det ikke finnes en fasit, er det mange designforslag for å gi bedre funksjonalitet. For eksempel kan galvanisk støy forhindres med riktig jording, induktiv støy reduseres med filterkomponenter og kapasitiv støy kan skjermes fysisk. Generelt er det best praksis å koble ting så nært som mulig hverandre eller langt unna støykildene for å unngå støy, som gjør det viktig å identifisere disse kildene.

Electrostatic Discharge (ESD)

For elektronikk er det viktig å ta hensyn til ESD. Det er en forkortelse for Electrostatic Discharge, eller på norsk: triboelektrisk ladning. Det henvender til fenomenet hvor statisk elektrisitet kan bygges opp ved friksjon, hvor elektronene blir ladet opp og kan utløses på kort tid ved kontakt. Et kjent eksempel er hvis man subber beina i et teppe, vil det bygges opp en statisk ladning som kan utløses hvis man da tar tak i elektriske komponenter. Et menneske kjenner ikke statiske utladninger under 2 [kV], derfor er det viktig å ha visse tiltak for å forebygge det når man jobber med elektriske komponenter [11]. Dette er spesielt viktig når man jobber med mer sensitive komponenter som mikrokontrollere. Slike tiltak kan variere alt fra løsninger for selve kretsen som dioder, til eksterne tiltak som ESD-lenker som jorder utladningene til en matte og sko som er ESD-sertifiserte. EPA er et vanlig uttrykk i denne sammenheng, som refererer til "ESD Protected Area".

Avkoblingskondensatorer (Decoupling Capacitors)

Avkoblingskondensatorer skjermer systemet mot støy ved å glatte ut variasjonene i spenningen. De forsyner systemet med spenning ved fall, og filtrerer ut høyfrekvent støy. Derfor blir de ofte benyttet i sammenheng med strømforsyninger, slik at systemer som er utsatt for støy ikke blir like sterkt påvirket.

Low Dropout Regulator (LDO-regulator)

En LDO regulator er en lineær spenningsregulator som gjør varierende spenning om til varme. Det resulterer i konstant spenning på output, selv når forsyningen dens er nært output spenningen. Den er svært relevant i forbindelse med mikrokontrollere, da de kan gi substansielle switchetap ved PWM regulering. Baksiden med LDO regulatorer er at de kan generere en del varme om spenningsforskjellen er for stor.

In-System Programming (ISP)

ISP er "Atmel AVR-språk" for SPI-basert (Serial Peripheral Interface) programmering [20]. Den benytter i all hovedsak fire ledninger: MISO (Master In Slave Out), MOSI (Master Out Slave In), SCK (Serial Clock) og RESET, i tillegg til Vcc og GND. Det benytter en hovednode (master) som styrer de andre enhetene (slaves), som oppdateres ved klokkesykler fra SCK.

Fuse bits and Register bytes

Mikrokontrollere har mange innebygde konfigurasjoner, hvor det er mulig å aktivere ved hjelp av software. Sentralt for dette er fuse bits og register bytes. Sistnevnte er bytes som kan endres, hvor spesifikke bits er relatert til de forskjellige konfigurasjonene. I databladet til den aktuelle mikrokontrolleren er det listet opp de forskjellige funksjonene og hvilke register som må konfigureres for å aktivere disse. Eksempelvis aktivering av forskjellige typer ADC og pulsbreddemodulasjon kan settes opp ved avlesning av tabellene og programmering av disse register bytene. Fuse bits kan ligne, men har en viktig forskjell: de endrer fysiske transistorer i mikrokontrolleren som vil vedvare frem til de blir endret igjen, selv etter strømbrudd. Register bytes endrer også transistorer, men blir nullstilt etter mikrokontrolleren blir startet på nytt.

I/O Pins

I/O står for INPUT/OUTPUT og beskriver pinner som kan sende og motta elektriske signaler. I virkeligheten er det sjeldent eksakte verdier, derfor registreres digitale signaler innenfor spenningsområder. For eksempel benytter prosjektets mikrokontroller følgende terskler:

Tabell 2.1: Spenningsterskler til digitale signaler for ATMEGA168-20PU [6]

	Minimum	Maksimum
Output: Høy	4.2 [V]	V_{CC} [V]
Output: Lav	0 [V]	0.9 [V]
Input: Høy	$0.7 \cdot V_{CC}$ [V]	$V_{CC} + 0.5$ [V]
Input: Lav	-0.5 [V]	$0.1 \cdot V_{CC}$

Mikrokontrollere har vanligvis en mengde porter med et sett antall I/O pinner hver. For å konfigurere ønskede moduser, kan mikrokontrolleren initialiseres ved å aktivere register bytes.

Ved forklaring av flere porter og pinner, er det vanlig at port verdien er beskrevet med en x og den individuelle pinnen med n. For eksempel med PINxn: PINB5 er pin 5 i port B. I register bytene beskriver hver bit en individuell pinne, hvor for et 8-bit system, beskriver den mest venstre bitten (MSB) pin 7 og mest høyre bit (LSB) beskriver pin 0. Derfor kombineres denne syntaksen med register bytes for å gjøre det mer lesbart, for eksempel at DDB3 er definert med verdi 3, som benyttes med shift operatorer for å sette retningen til den korresponderende pinnen med denne indeksen.

Programmering av I/O pinnene krever først at retningen av de individuelle pinnene er satt. Dette blir gjort ved å definere portens data-direction verdier eller DDRx, hvor en 1 på den relative bitten til I/O pinnen betyr at det er en output, og 0 er input.

Det betyr at til å skru på en pinne i C++, for eksempel til pinne 5 (siffer nummer 6 fra høyre) av port B, kan man sette DDRx og PORTx som:

```
DDRB = 0010 0000  
PORTB = 0010 0000
```

Men et problem med dette er at det setter alle pinnene i PORT B som 0 unntatt pinne 5. Da er en bedre måte å benytte boolske operatorer sammen med shift operatorene << og >> som gjør koden mere lesbar. Disse operatorene skifter rekkefølgen av bits i en byte, hvor i operasjonen $a << x$ blir byten a skiftet til venstre x ganger, og $a >> x$ skifter a til høyre x ganger:

```
0000 0001 << 3  
Samme som: 1 << 3  
Resultat: 0000 1000
```

Ved bruk av OR operatoren (|) kan man sammenligne portens gjeldende verdier og pinnen som skal skrus på, for eksempel hvis porten har allerede pinne 4 (nummer 5 fra høyre) på og man vil sette på pinne 5 (nummer 6 fra høyre) uten å skru av pinne 4:

$$a = 0001\ 0000$$

$$a = a|00100000$$

Samme som: $a| = (1 << 5)$
Resultat: $a = 0011\ 0000$

AND og NOT ($\&$ og \sim) operatorene kan bli brukt til å skru av en individuell pinne ved å sammenligne portens nåverende verdier og en NOT av bitten som skal skrus av:

$$a = 0010\ 0100$$

$$a = a \& \sim(00100000)$$

Samme som: $a\& = 11011111$

Samme som: $a\& = \sim(1 << 5)$
Resultat: $a = 0000\ 0100$

For å gjøre koden lettles, så er pinneverdiene Pxn ofte definert som en int for n i programmet, for eksempel istedefor å skrive $(1 << 5)$, så kan man skrive: $(1 << PB5)$, som forklarer leseren av programmet nøyaktig hvilken pinne som er ment til å bli definert uten bruk av kommentarer.

Sinking og sourcing

Ved sinking tar enheten inn spenning fra en annen enhet og drar det til jord. Sourcing fungerer motsatt, hvor det sender ut spenning til en annen enhet. Det kan være lurt å reflektere hvilken av de to man velger for utgangene til en mikrokontroller, da de har forskjellige egenskaper. Om man ønsker et høyere strømtrekk, kan det være lurt å benytte sinking fra en annen kilde da det vil kreve mye av mikrokontrolleren, hvor dens ressurser kan være ønsket andre steder. En ekstern strømkilde vil også gi mulighet for større begrensning for støy.

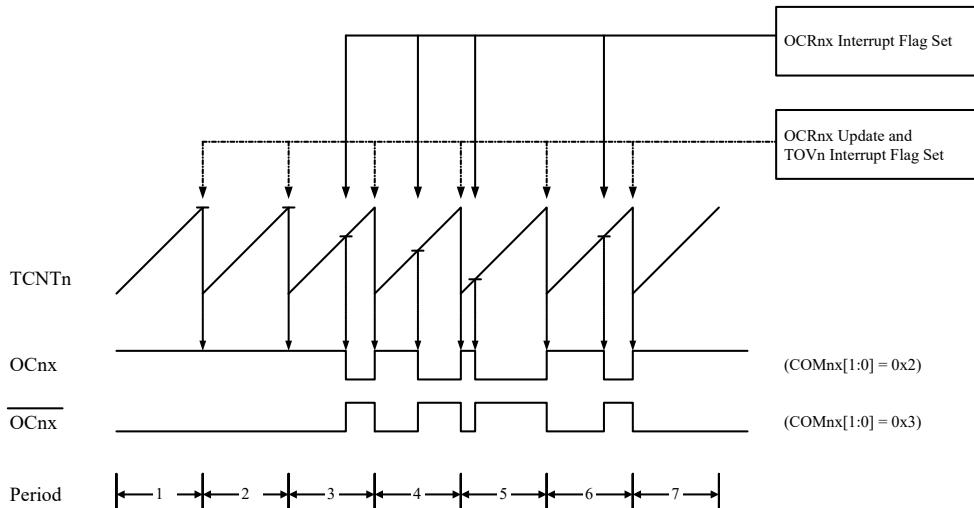
Interrupt protokoller

Interrupts er et alternativ til polling, som er når det blir kontinuerlig sjekket om det har blitt en endring i signalet. Polling bruker opp veldig mye prosessorkraft for å konstant sjekke signalet, så interrupts blir brukt hvor det gir beskjed ved endring på signalet og avbryter syklen momentant. Det finnes en god del forskjellige interrupt protokoller som kan benyttes til varierte applikasjoner. Et kjent eksempel er pin change interrupt, som utløses ved eksterne signaler som for eksempel en knapp. Interrupt kan også benyttes internt som en del av større systemer, som for eksempel ved analog til digital konvertering eller pulsbreddemodulasjon. Da benyttes interrupt protokollen ved logikk med klokkesignaler. Når en interrupt blir aktivert, stopper CPUen der den er i main programmet og kjører koden skrevet på funksjonen for interrupt service routine (ISR), også kalt interrupt handler. Når en interrupt blir aktivert på en av de 3 pin change interrupt requestene, så blir den tilhørende ISRen kjørt.

Pulsbreddemodulasjon og prescaler

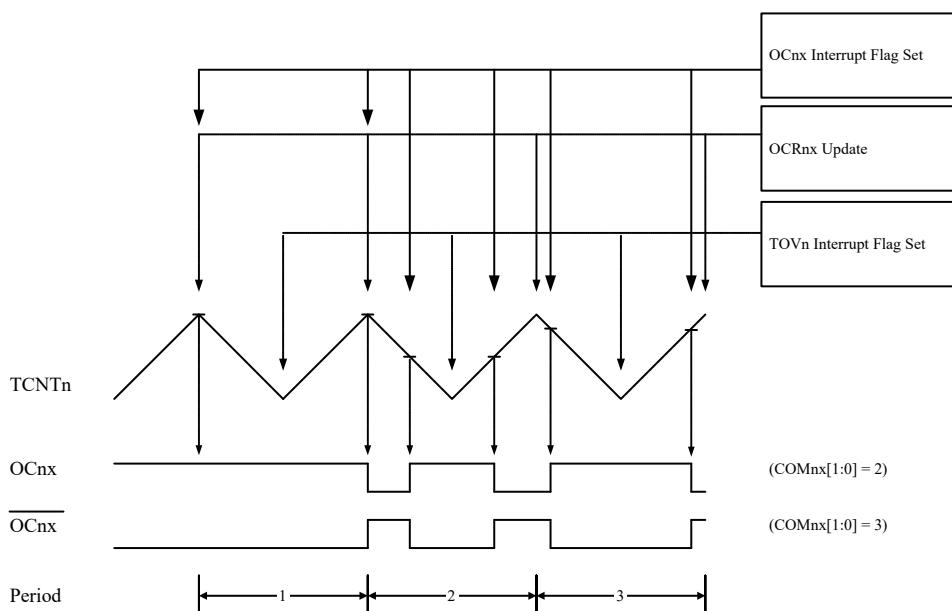
Pulsbreddemodulasjon, eller Pulse Width Modulation (PWM) på engelsk, er en utbredt metode for å kontrollere enheter med digitale signal. Da blir det periodevis sendt ut enten høy eller lav, over en gitt del av perioden. Det har et bredt spekter av bruksområder, alt fra LED-dimming til motorkontroll. Det finnes flere algoritmer for PWM-kontroll. Om pulsbreddemodulasjon blir gjort ved hjelp av if-setninger i en hovedløkke, kan det føre til at det blir uforutsigbare signaler, da maskinen gjør beregninger som gjerne tar variabel tid mellom hver runde i løkka. I mikrokontrollere finnes det flere algoritmer for å synkronisere signalet med klokkefrekvensen til kontrolleren, sånn at slike problemer blir moderert. To slike algoritmer er **Fast PWM** og **Phase Correct PWM**. Det er også vanlig å skalere ned klokkefrekvensen til mikrokontrolleren med en **prescaler** (N), slik at timeren til sammenligning med PWM-algoritmen går saktere. Det er slik at det blir mindre konsekvenser ved feil. Om klokkesignalet ikke ble skalert ned, kunne signalet blitt forskjøvet en hel periode, men ved prescaling vil det bare bli forskjøvet med en periode delt på N .

Fast PWM mode er en algoritme som sammenligner ett signal med klokkesignalet, og gir en output OCnx avhengig av denne sammenligningen, som vist i figur 2.1. Klokka teller opp til den når TOP verdien, for så å starte på nyt. Det resulterer i konstant periode fra klokkefrekvensen og en duty cycle som er på fram til sammenligningen ikke er sann (eller motsatt om man bruker Inverted Mode). Den gir en presis og enkel kontrollering av PWM, som starter på starten av perioden. Sammenligningsverdien benytter en interrupt-protokoll for å oppdateres der og da, istedet for å vente til løkka er ferdig.



Figur 2.1: Fast PWM Mode [6]

Phase Correct PWM mode er en algoritme som ligner på fast PWM. Forskjellen er at den teller ned igjen fra den når TOP, i stedet for å starte fra bunnen igjen. Dette resulterer i at PWM signalet er sentrert i midten av perioden, i stedet for at den starter på begynnelsen, som vist i figur 2.2. Denne klokkealgoritmen gjør også at den ikke har like god mulighet for høy frekvens som fast PWM mode, men symmetrien gir gode egenskaper for andre applikasjoner. Et godt eksempel er motorer, som drar nytte av symmetrien. Det har en jevnere overgang mellom periodene, siden flankene endres inne i periodene, i stedet for starten av dem. Om det går fra høy til lav duty cycle for fast PWM, vil det gå veldig kort tid fra slutten av lang duty cycle til starten av neste periode. For phase correct PWM derimot, vil det gå lengre tid, da duty cycle signalet er sentrert i midten av periodene.



Figur 2.2: Phase Correct PWM Mode [6]

ADC

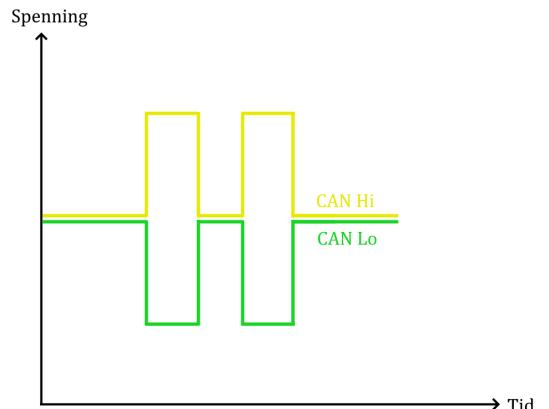
ATMEGA168 har en innebygd analog til digital konverter (ADC) med en intern suksessiv aproksimasjon konverterer som gir en 10-bits utgang fra input spenningen til port C pinnene 0-5. Siden det ikke er mulig å legge inn hele utgangen i en register så blir utgangen satt inn i 2 8-bits registere, ADCL og ADCH, en innebygde funksjon i MicroChip Studio setter disse to sammen sånn at hele verdien kan lett bli lest fra variablet med navn ADC. Den interne Vref velger hvilken spenningen som skal bli satt som toppunktet til ADCen, dette kan f.eks være AREF, Avcc med en avkoblingskondensatorer på AREF eller en intern referanse basert på hva som blir satt i register bytene for ADCen. Bunnpunktet for ADCen er GND.

ADCen kan kjøre i "Free Running Mode", som aktiverer når det blir en endring i signalet, eller i "Single Conversion Mode" som aktiverer ADCen bare en gang når det er ønsket i programmet. Valget mellom Free Running mode og Single Conversion mode blir satt i ADCSRA registeret. Sukcessiv approksimasjons ADCen kjører best når det har en frekvens mellom 50-200 [kHz]. Frekvensen kan bli valgt av en prescaler som tar CPU frekvensen og deler det med prescale verdien. [26]. ADCen sammenligner input spenningen fra ADC[0:5] pinnene med spenningen på AREF, AVCC, eller en intern spennings referanse på 1.1 [V]. Valget av spenningsreferansen blir satt i ADMUX registeret.

ADCen kan bli satt opp til å gi en interrupt når ADC konverteringen er ferdig; denne interrupten kan da bli brukt for å kjøre en ISR som kan f.eks gi ADC utgangen til resten av programmet uten å måtte sjekke om det er en endring i ADC eller ADMUX registerene.

CAN bus (Controlled Area Network)

CAN bus er en standardisert protokoll som muliggjør kommunikasjon mellom enhetene som er tilkoblet. Det benytter differensiell sampling, hvor signalet blir overført som en forskjell i spenningen mellom to ledere, CAN Hi og CAN Lo. Denne metoden bevarer signalintegriteten til signalene, da ekstern støy vil kanselleres ut siden det påvirker begge i samme retning og vil bli kansellert ut ved differansemåling. CAN bus har termineringsmotstand parallellkoblet i begge ender mellom CAN Hi og CAN Lo. Termineringsmotstandene benyttes for å begrense elektrisk støy og matche den nominelle impedansen. Det tilsvarer impedansen systemet opererer mest optimalt ved. De forhindrer også refleksjoner ved å absorbere energien i enden av ledningene, slik at det ikke blir sendt energi tilbake igjen og degraderer signalet; men dette er helst et problem med større systemer med lange ledninger. CAN bus kan settes opp på flere måter, hvor eksempelvis biblioteket "flexcan_t4" har laget et egendefinert grensesnitt for å sende CAN-meldinger.



Figur 2.3: CAN bus differensiell sampling

CAN 2.0 Meldingsoverføring

CAN meldinger har fire typer av meldinger som er kalt datarammer, disse har forskjellige funksjoner, typene er DATA FRAME, REMOTE FRAME, ERROR FRAME, og OVERLOAD FRAME. DATA FRAME er meldingen som overfører data fra en kontroller til nettverket, REMOTE FRAME er en melding som blir sendt til å be om en annen kontroller til å sende en DATA FRAME med samme identifier, en ERROR FRAME blir sendt når en kontroller merker et feil i noen av CAN meldingene, og en OVERLOAD FRAME er en melding som sier at det kommer en annen DATA FRAME eller REMOTE FRAME på nettverket snart.

DATA FRAME er sentralt i prosjektet og er satt opp med syv bitfelt med varierende lengde: START OF FRAME (SOF), ARBITRATION FIELD, CONTROL FIELD, DATA FIELD, CRC FIELD, ACKNOWLEDGEMENT FIELD (ACK FIELD), og END OF FRAME (EOF). START OF FRAME bitfeltet er en singel lav verdi og sier at en melding begynner, ARBITRATION FIELD bitfeltet har med meldingens IDENTIFIER og RTR BIT. IDENTIFIERen er en 11 bits verdi som gir betydningen av meldingen, dette kan da bli brukt til filtrering av meldinger. IDENTIFIER bitfeltet kan ikke inneholde bare nullere, RTR BIT må være høy på en DATA FRAME men er lav i en REMOTE FRAME. Det neste bitfletet som blir sendt er CONTROL FIELD som har to 6 bits, fire av disse blir brukt til å gi lengden av informasjonen i neste bitfelt, og to av de er reservert for framtid utvidelse. Den fire bits data lengde koden kan ha maks verdi på 8 som sier at maks lengde på informasjonen som kan bli sendt er 8 bytes lang. DATA FIELD inneholder informasjonen som vil bli sendt i 0-8 bytes basert på lengden valgt i CONTROL FIELD. dette betyr at en melding med ingen informasjon kan også bli sendt, hver bit er sendt MSB først. Bitfeltet CRC FIELD er en kalkulert sekvens som blir brukt for feiloppdagning og inneholder to deler, CRC SEQUENCE, og CRC DELIMITER. CRC SEQUENCE er kalkulert fra resten av koden, CRC DELIMITER er en singel lav verdi som kommer i etterkant av CRC SEQUENCE. Bitfeltet ACK FIELD er to bits lang og inneholder ACK SLOT og ACK DELIMITER bitsene, ACK SLOT blir brukt til å kommunisere at den riktige kalkulerte CRC SEQUENCE ble sendt, ACK SLOT blir sendt som en høy verdi av senderen av melding og må bli overskrevet med en lav verdi hvis CRC SEQUENCE matchet. ACK DELIMITER er en høy verdi som blir sendt etter ACK SLOT blir satt til lav av motakkeren av meldingen, og kan bli brukt til å skille ACK eller CRC feil. END OF FRAME bitfeltet blir da sendt til slutt med 7 høye bits [13].

Buildroot

Buildroot prosjektet baserer seg på en Linux kernel, med andre ord en helt nedstrippet versjon av Linux hvems hovedoppgave er å kommunisere med hardware [25]. LTS (long term support) versjoner er å foretrekke da disse fortløpende fikser bugs og sikkerhetsbrister. Gjennom buildroot menyen legges ekstra funksjonalitet i operativsystemet via pakker for å skape en skreddersydd distribusjon til formålet. Følgende pakker legges til:

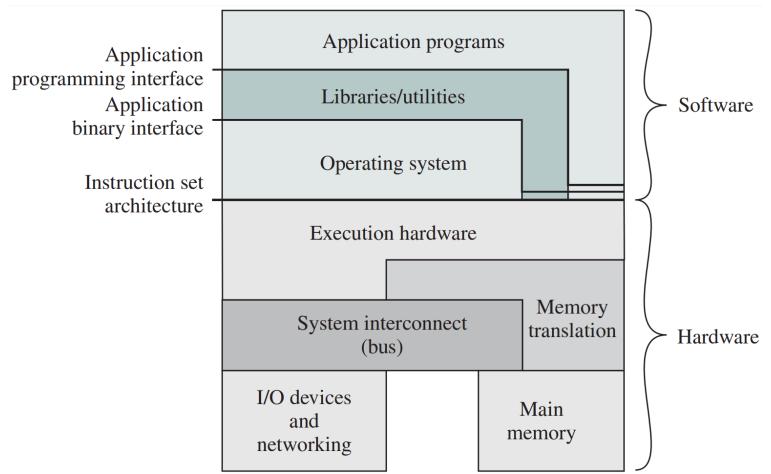
1. IProute2 for ethernetkommunikasjon
2. QT5 for QTcreator støtte
3. LIBSOCKETCAN for CAN-bus støtte
4. LIBSTDCPP for implementasjon av C++ sitt standardbibliotek (CPP toolchain)

Pakkene blir lagt til med BR2-external mekanismen. BR2-external tilbyr ett av to alternativer for å legge til prosjektspesifikke pakkekonfigurasjoner. Som standard legges pakkekonfigurasjonen inn i buildroot mappen, men ved bruk av BR2-external kan disse konfigurasjonene legges til utenfor selve buildroot mappesystemet.

Operativsystem (OS)

"Et operativsystem er en systemprogramvare som administrerer hardware- og softwareressurser på en datamaskin" [17]. Et OS kan være laget for et spesifikk bruk eller for generelt bruk som på PC'er. Ved utvikling av OS burde utvikleren passe på at maskinvare har muligheten til å bli endret eller at mere kan bli satt på i etterkant. Operativsystemer håndterer hvordan en applikasjon blir kjørt med hardwaren av maskinen. Dette grensesnittet mellom operativsystemet og applikasjonene gjør at et program kan bli laget for å fungere på flere typer av maskiner uten at utvikleren må ta hensyn til det.

Sammenhengen mellom OS og resten av systemet er vist i figur 2.4. Systemkall er betegnelsen på når programmer bruker tjenester fra operativsystemet, gjennom grensesnittet nevnt ovenfor. I OS design så blir i de fleste tilfellene programvare kjørt enten på user eller kernel mode ved CPUen; denne separasjonen av kjøringsmetoder gir en enkel metode til å håndtere kræsj i programvare, hvor ett kræsj i kernel mode kan være katastrofalt men en feil fra programvare i user mode kan bli enklere håndert. Ulempen med å bruke user mode er at programvaren kan oppleves som tregere, da det vil kjøre flere prosesser i bakgrunnen samtidig, om en slik modus er aktivert [1]. Operativsystemer også håndterer hvordan privilegier et programvare kan ha, hvor forskjellige privilegier gir f.eks tilgang til hvilken filer kan bli endret eller ikke, hvor den høyeste privilegen i Unix er root. Beste praksis er å gi så lav privilegier som hver enkelt programvare trenger, som kan være en del av jobben til OSen å gjøre dette [18].



Figur 2.4: Datamaskinens hardware og software arkitektur [28]

Nettverk, IP-addreser, DHCP og SSH

Det finnes mange typer **nettverk**, alt fra fysiske nettverk som ethernet og bus nettverk som CAN, til Wireless LAN (Local Area Network). Sentralt for prosjektene er blant annet ethernet. Det er en protokoll som er brukt i mange applikasjoner, deriblant kommunikasjon mellom datamaskiner.

For å etablere forbindelse mellom enhetene, må **IP-adressene** være konfigurert. Det er en adresse fra Internett Protokollen (IP), som er en numerisk identifikasjon tilegnet enheter som er koblet til nettverk. Mest utbredt per dags dato er IPv4. Den er den fjerde versjonen av internett protokollen, som erstattet de tidligere versjonene på 80-tallet [31]. IPv4 bruker 32-bits adresse lagring, som er delt i fire oktetter som er separert med punktum [31]. Hver oktett har maksimal verdi fra 8-bit, altså 255 ($2^8 - 1$). Den er mest kjent som den første allment anerkjente IP-versjonen for kommersiell bruk, men allerede nå er verden i gang med en migrering over til IP versjon 6. "Internet Stream Protocol" var navnet til protokollen etter IPv4, som brukte samme 32-bits adresse lagring [32]. Det var en eksperimentell versjon som aldri ble introdusert til kommersiell bruk, men en god del av aspektene dens ble overført til andre ting som "Voice over IP" [32].

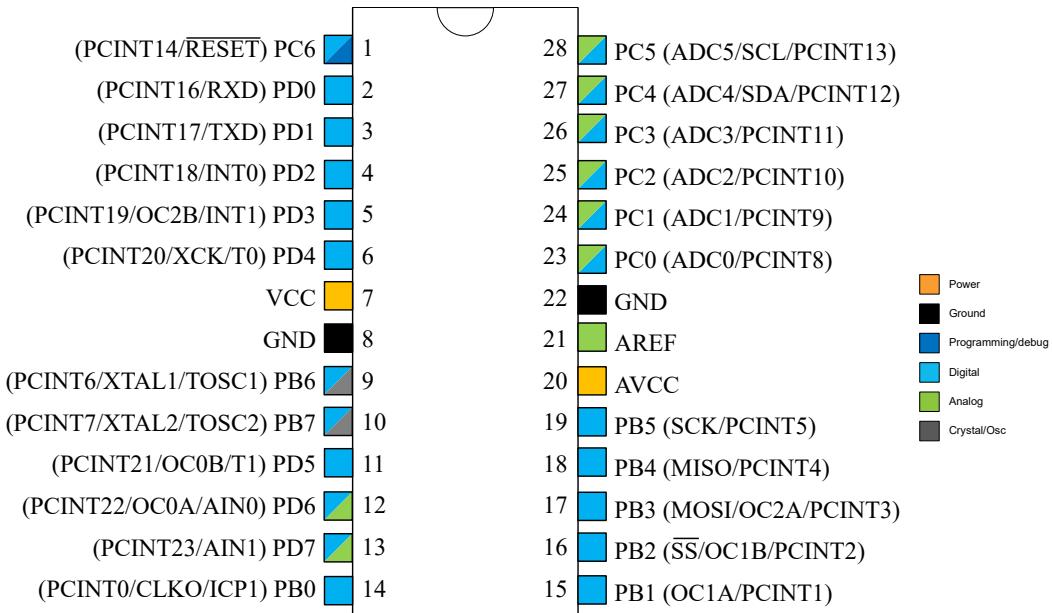
For å koble til nettverk, kan man ha konstant IP-adresse, eller benytte **DCHP**. Det står for "Dynamic Host Configuration Protocol" og er en administrasjonsprotokoll for å automatisk tildele IP-addreser. Dette er veldig utbredt i mange applikasjoner, som ved tilkobling på trådløst nettverk, men i noen tilfeller er det mer hensiktsmessig å benytte statisk IP-adresse. Et eksempel på dette er for PLS-programmering der man kan koble til et lokalt ethernet-nettverk, hvor det benyttes statisk

adresse på PLSen. Derfor kan det være nødvendig å sette IP-addressen til adapteren som datamaskinen kobles på med til statisk, slik at verten (host) ikke bruker samme adresse som målenheten (target). I prosjektet benyttes en virtuell maskin på datamaskinen som vert og en Raspberry PI som målenhet; denne konfigurasjonen er veldig lik det forrige eksempelet da det benyttes statiske IP-addreser.

Ved tilkobling på nettverk, kan det være lurt å tenke på informasjonssikkerhet. **SSH** refererer til Secure Shell protokollen, som er nært knyttet kryptografi. "Kryptografi" kommer fra de to greske ordene "*kryptos*" og "*graphein*" som betyr respektivt "*skjult*" og "*å skrive*" [33]. Kryptografi er kort fortalt et uttrykk for å gjøre informasjon utelukkende tilgjengelig for dem som skal motta det. SSH bruker asymmetrisk kryptografi, hvor det setter opp offentlige og private nøkler. Hvem som helst kan kryptere ned informasjon med den offentlige nøkkelen, men for å dekryptere informasjonen igjen må man ha den private nøkkelen, ergo assymmetri grunnet det faktum at det er enveis. Man burde være varsom på usikre nettverk, da man kan ha sårbarheter uten å være klar over det. Selv om man ikke sender viktig informasjon, kan brukeren som er logget inn ha privilegier som gir tilgang til sårbarheter. "*Granulariteten for privilegiene varierer fra operativsystem til operativsystem*" [18]. Eksempelvis har Linux muligheten for å logge inn som root-bruker med sudo kommandoen. Root-brukere har tilgang til alt, som derfor burde brukes varsomt.

3 Metode

Gruppen benyttet ATMEGA168-20PU mikrokontroller, med følgende konfigurasjon:



Figur 3.1: ATMEGA168-20PU Mikrokontroller pinout [6]

Versjonskontroll

Versjonskontroll ble brukt med git og github for alle delprosjektene. Branching ble kun brukt for eksperimentelle endringer. For delprosjekt 3 ble det kun lastet opp de relevante filene som ble endret, da det ville blitt store mengder data ved opplasting av hele buildroot-prosjekt mappen.

3.1 Delprosjekt 1

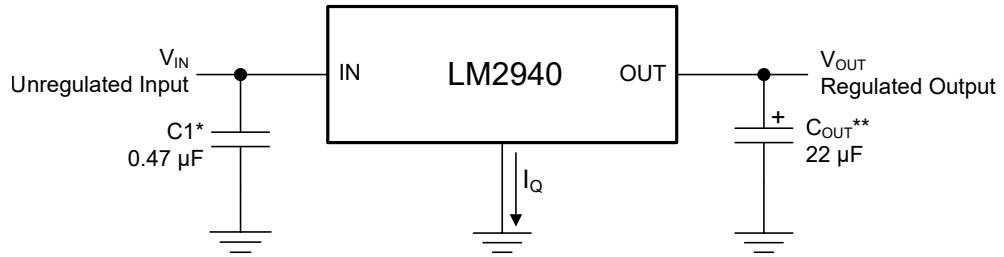
3.1.1 Utstyr

Tabell 3.1: Utstyr utlevert for delprosjekt 1

Navn	Antall
Strømforsyning EA-PS 2042-10B	1
Oscilloskop TDS 2012B	1
Koblingsbrett, 82.5 x 56 [mm]	2
Jumper wire kit	1
Atmel ICE	1
Blekksprutkabel	1
USB-kabel (USB Type A ↔ MicroUSB Type B), 0.5 [m]	1

3.1.2 Forsyningskrets

Forsyningskretsen ble koblet opp slik at den består av LDO regulator som tar inn strøm fra lab-strømforsyningen sammen med kondensatorer.

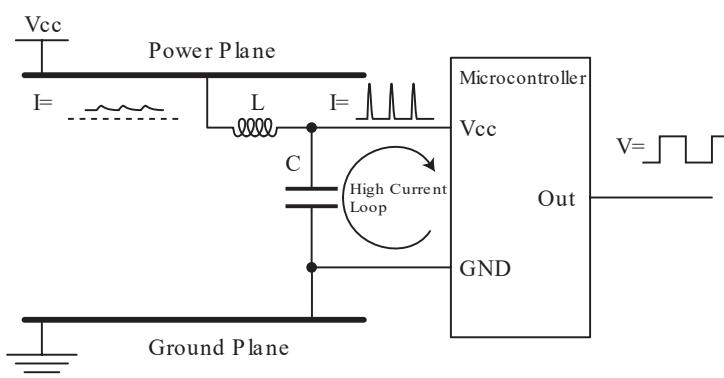


Figur 3.2: Linear Dropout Regulator [14]

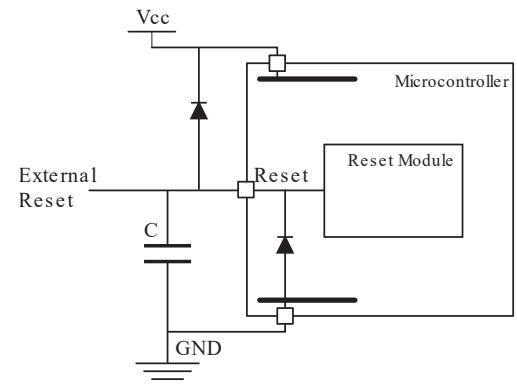
For LDO kretsen ble det lagt til et strømmindikasjonslys, som består av en LED. For at den ikke skal brenne opp, ble det lagt til en strømbegrensende motstand. Kretsen ble koblet opp med strømforsyning, hvor verdiene ble bekreftet med multimeter.

3.1.3 Mikrokontroller krets

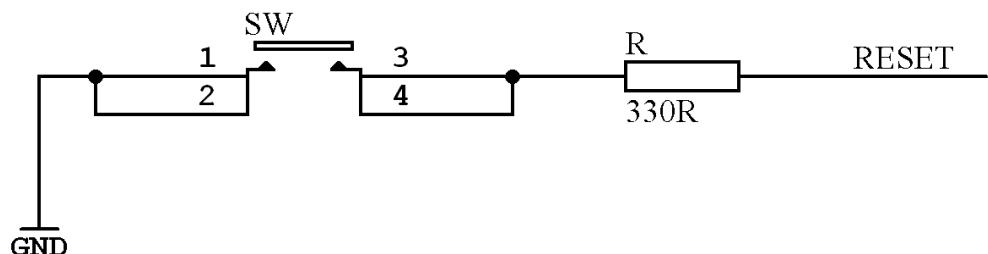
Ifølge AVR Hardware Design Considerations burde mikrokontrolleren ha avkoblingskondensatorer mellom Vcc og jord, samt spole/ferittring, som vist i figur 3.3 [8]. Gruppen la til kondensatorene og vurderte spole/ferittring, men da prosjektene er relativt grunnleggende konkluderte gruppen med at spole/ferittring ikke var nødvendig.



Figur 3.3: Decoupling: Kondensator og spole [9]



Figur 3.4: Reset kondensator og diode [9]



Figur 3.5: Reset Bryter og seriemotstand [8]

Ifølge databladet hadde mikrokontrollerens **reset pin** ikke ESD beskyttelse, i motsetning til de andre I/O pinnene [6]. Derfor ble koblet en ekstern diode fra reset til Vcc for ESD beskyttelse. Det ble også lagt til en kondensator. Disse to implementasjonene er vist i figur 3.4. En pullup motstand ble vurdert lagt til eksternt, men på grunnlag av datablad konkluderte gruppen med at en intern pullup motstand var innebygd [6]. Dette ble dobbeltsjekket med måling av multimeter mellom reset pin og Vcc. Videre ble det koblet inn en bryter og seriemotstand, som vist i figur 3.5.

Oppgaven spesifiserer programmering med ISP, hvor prosjektet benytter Atmel ICE som programmeringsverktøy. For å bruke denne med systemet, koblet gruppen opp mini blekksprut kabel og 90° pin header sammen med mikrokontrolleren.

3.1.4 Mikrokontroller programmering

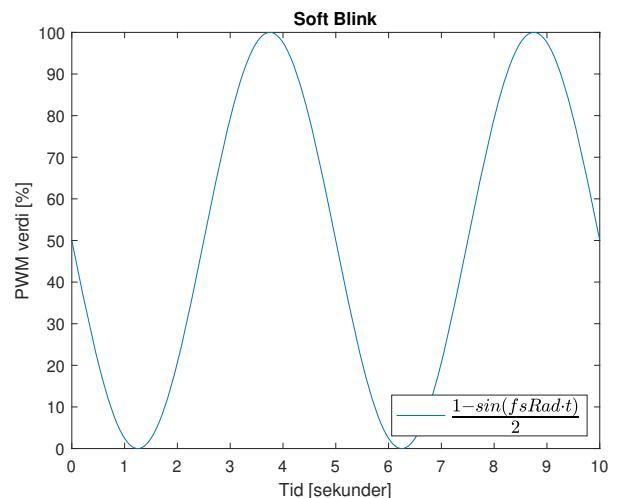
ATMEGA168-20PU har 3 digitale porter med 8 I/O pinner hver, port B, C og D. Hver port har 3 register bytes som beskriver virkemåten og tilstanden av hver pinne: DDRx (data-direction), PORTx (pin output), og PINx (pin input) [22]. DDRx er et register som setter retningen til pinnene, som refereres til som DDxn. PORTx har forskjellig funksjon avhengig av hvilken retning pinnen er satt til. Om den er output, setter man denne høy eller lav for å skrive til pinnen. Om den er input, kan man velge om den skal aktivere intern pullup motstand, eller om den skal sette på tri-state mode. PINxn brukes for å lese data til den korresponderende pinnen. Ved input leser den mottatt data, og ved output leser den dataen som blir skrevet.

C++ og Atmel Microchip Studio ble brukt for programmering av mikrokontrolleren. Atmel ICE koblet til, slik at ISP-protokollen kan benyttes i Microchip Studio. For å krysskompile, ble det lastet opp i menyen "Memories" under "Device Programming". For å få tilgang til denne, må man velge riktig programmeringsverktøy og enhet, samt lese verdien fra den, med andre ord sjekke at mikrokontrolleren er koblet opp riktig og sender riktig spenning inn til verktøyet. Her kan man også verifisere at det er korrekt signature byte. En test ble gjennomført hvor et program som utførte enkel aritmetikk ble brukt til å sjekke at koden ble buildet riktig i Microchip Studio. Deretter ble et program for en blinkende LED laget på port D Pin D6 hvor pin D6 ble satt som en output i DDRD. For dette programmet ble strømmen for LEDen sourcet fra mikrokontrolleren.

3.1.5 PWM og Soft Blink

Det ble gjort programmering av mikrokontrolleren for å kontrollere en LED. Målet var å ha en "soft blink", hvor lysdioden går gradvis mellom 0% og 100% lysstyrke. PWM-algoritmen som ble valgt for prosjektet var "Fast PWM, 10-bit" som ble konfigurert ved hjelp av databladets konfigurasjoner, hvor avlesninger fra tabeller er beskrevet i resultater. For å få lyset til å ha variabel lysstyrke som går gradvis, ble det sendt inn en variabel duty cycle til PWM-algoritmen. Denne variable duty cyclen er definert av følgende funksjon:

$$f(t) = \frac{1 - \sin(fsRad \cdot t)}{2} \quad (3.1)$$



Figur 3.6: Simulering i MATLAB, kode i A.5.1.1

3.1.6 Pin change interrupt og tilstandsbytting av lysdiode

Pin change interrupt er asynkronisk fra klokkesignalet, som vil si at det blir utløst av en høy eller lav flanke, tilsvarende en veksling av signalet på pinnene med PCINT egenskaper. Det er 3 pin change interrupt requests som er utdelt mellom de 23 pinnene med PCINT egenskaper, Tabell 3.2 beskriver disse pinnene.

Tabell 3.2: Pinner for pin change interrupt [6]

Pin change interrupt request	Pinner
0	PCINT[0:7]
1	PCINT[8:14]
2	PCINT[16:23]

Til å aktivere pin change interrupt så må Pin Change Mask register og Pin Change Interrupt Control register være aktivert for den ønskede pinnen. For Pin Change Interrupt Control register (PCICR) kan de siste 3 (LSB) bittene settes til 1 til å fortelle mikrokontrolleren hvilken av requestene 0:2 skal bli aktivert ved en veksling av signal på valgte PCINT pinnene. I Pin Change Mask registeret (PCMSK[0:2]) blir PCINT pinnene som er ønsket å bruke med pin change interrupt valgt. I-bit masken i status register (SREG) må også bli skrudd på. Det er ikke en måte å gjørde det direkte, så ”Set Enable Interrupt” funksjonen sei() benyttes. Denne aktiverer interrupt globalt. Det ble satt opp logikk for å skru på lysdioden ved knappetrykk som jordet pinnen.

PCINT22 på pin D6, som er på Pin Change Interrupt Request 2, ble brukt. Dette ble satt opp i programmet ved å sette PCICR til 0000 0100 til å sette PCIE2 til 1, som ble funnet fra avlesning fra seksjonen Pin Change Interrupt Control Register av databladet [6], og pin D6 som en interrupt pinne i masken ved å sette PCMSK2 til 0100 0000. Til slutt ble sei() funksjonen kjørt, for å sette på I-bitten i SREG til å aktivere global interrupts.

Dette oppsettet med interrupts ble brukt for å benytte en knapp til å endre tilstand til en LED, mens polling programmet ble brukt slik at LEDen skrus på kun mens knappen er holdt inne. Interrupts ble brukt sånn at trykkene ikke måtte bli pollet, som tar unødvendig prosessortid og har muligheten til å gå glipp av knapp trykket.

3.1.7 ADC og variabel PWM settpunkt ved potensiometer

Pin PC3 ble koblet opp som ADC input med et potensiometer som ble brukt som en spenningsdeler til å gi ut en variabel spenning for bruk som ADC input, sammen med en strømbegrensende motstand i serie slik at strømmen ikke ble for høy da potmeteren ble skrudd til lavere motstand. 3 registere ble satt opp for å bruke ADCen: ADCSRA (ADC Control and Status Register A), ADMUX (ADC Multiplexer Selection Register) og PRR (Power Reduction Register). Selv om det ikke var en del av oppgaven, valgte gruppen å også sette opp SREG (Status Register) for interrupt funksjoner. For å sette opp ADC3 (tilhørende PC3) som input til ADCen ble ADMUX bittene MUX[3:0] satt til 0,1,1,1, som forteller mikrokontrolleren at ADC3 er inputen til ADCen. Bittene REFS[1:0] på ADMUX velger referanse signalet; det ble valgt til å bruke samme referanse som AVCC med en ekstern kondensator koblet til ground og AREF til å fjerne støy, da ble REFS[1:0] bittene satt til 0,1. ADLAR bitten på ADMUX velger om signalet er left-shifted eller right-shifted som bytter rekkefølgen av ADC utgang byttene. Right-shifted ble brukt slik at ADC registere kan bli lest uten noe bitwise operasjoner; derfor ble ADLAR satt til 0.

I ADCSRA registeret ble konfigurasjonen av ADCen satt til "Free Running mode" med interrupts; da ble ADIE og ADATE bittene satt til 1. Til å begynne free running mode, må ADSC bitten bli satt til 1 og ADEN satt til 1. ADPS[2:0] bittene velger prescaleren til ADCen. Prescaleren ble satt til 8 for å gi en ADC frekvens på 125 [kHz] med CPU frekvens på 1 [MHz]. Siden det tar 13 klokkesykluser til å kjøre ADC konvertering, vil hver konvertering ta:

$$\frac{13[\text{sykluser}]}{125\ 000 \left[\frac{\text{sykluser}}{\text{sekund}} \right]} = 104[\mu\text{s}] \quad (3.2)$$

PRADC bitten i PRR må bli skrudd til 0 slik at ADCen blir skrudd på. Den er av standard satt til 0, så den trenger ikke å endres. For at interrupts skal fungere, så må I-bitten i SREG også bli skrudd på med sei() funksjonen. Interrupten fra ADC conversion completed blir gitt til ADC_vect vektoren for ISR funksjonen og ADC verdien blir da lest fra ADC registeret. ADC registeret sender ut en verdi i 10-bits, som ble skalert om til spennin.

ADCen ble brukt sammen med et potensiometer som ble brukt som variabel spenningsdeler for å sende et analogt signal til mikrokontrolleren, som videre ble brukt til å variere PWM ingangen til å styre lysstyrken til en LED. For dette ble det brukt "Fast PWM Mode" fra 3.1.5.

3.2 Delprosjekt 2

3.2.1 Utstyr og digitale forberedelser

Tabell 3.3: Utstyr utlevert for delprosjekt 2

Navn	Antall
Teensy 3.6	1
SK Pang Teensy CAN med OLED-skjerm	1
120 $[\Omega]$ motstand	2
Peak PCAN-USB FD <i>IPEH-004022</i>	1
D-sub connector på PCB	1

For å sende CAN meldinger, ble Teensyen satt på SK Pang kortet sammen med ledninger og termineringsmotstander i begge ender som vist i figur A.3. Motstandene ble satt på eksternt, da det ikke var ønskelig å lodde på brettet. Can0 porten blir ansett som primærport, og er koblet til pin 3 (TX) og 4 (RX) på teensy3.6 brettet. For delene av prosjektet som sender CAN meldinger til og fra PCen, benyttes en D-sub connector og PCAN-USB adapteren. For delene som sender CAN meldinger mellom to Teensyer, går ledningene (og motstandene) mellom Teensyene uten adapter.

For å krysskompilere kodene til Teensy, ble det benyttet utvidelsen PlatformIO til Visual Studio Code [21]. Videre ble det brukt PCAN-View for å ha oversikt over alt som går inn og ut av CAN bussen [23]. Det ble også installert bibliotek fra Adafruit for PlatformIO, for enklere bruk av OLED-skjermen [3]. Til slutt ble CAN-biblioteket "FlexCAN_T4" inkludert i PlatformIO [2].

3.2.2 CAN mellom Teensy og PC

For å sjekke at alt fungerte, ble det kompilert en eksempelkode fra SK Pang [12]. Det ble også sjekket at eksempelkoden til prosjektet også kompilerte riktig [19].

3.2.3 Sending og mottak av CAN-meldinger

Etter oppstarten av programmer og oppkoblinger, begynte gruppen med å sende og motta meldinger. Meldingen ble opprettet via PCAN-View og sendt gjennom CAN bussen til Teensy, hvor den ble returnert til PCAN-View. I PlatformIO ble det brukt en ”canSniff” funksjon, som kjører hver gang det blir sendt eller mottatt en melding på CAN nettverket. Dette er en metode som senere ble brukt og videreført. I dette programmet ble det brukt to separate ”sniffere” til å hente ut og vise informasjon om meldingene i Serial Monitor. ”Snifferne” ble satt opp i forkant av hovedløkka, da det er innebygd i biblioteket at det skal sjekke hver gang det blir mottatt eller sendt, respektivt ved hjelp av funksjonene `onReceive()` og `onTransmit()`.

3.2.4 OLED-display og CAN

Da det var bygd opp kunnskap for de grunnleggende CAN-funksjonene, ble det laget et program for å skrive til OLED-skjermen. Dette ble gjort ved hjelp av GFX-biblioteket til Adafruit for funksjoner som tegner firkanter, sirkler, skriver tekst, osv. Tekst ble skrevet ved hjelp av bibliotekets print funksjon. Det var ønskelig med en bue under tittelen på skjermen; dette ble gjort med to metoder. Den første var å tegne en sirkel utenfor skjermen med `drawCircle`, som hadde en del av buen innenfor toppen. Den andre metoden var å tegne en linje med høyde fra en periodisk funksjon:

$$y(i) = 5 \cdot \sin \left((i - 1) \cdot \frac{\pi}{OLED_{width}} \right) + 9 \quad (3.3)$$

fra starten av skjermen til slutten av skjermen, hvor i ble iterert gjennom en for-løkke. Det resulterte med at horisontalt koordinat gikk fra start til slutt, mens vertikalt koordinat fulgte sinusfunksjonen. Videre ble det brukt ”canSniff” som hentet ut IDen fra CAN-meldingen, for å vise denne på OLED-skjermen.

3.2.5 CAN mellom Teensy og Teensy: Pong!

Avslutningsvis ble det laget et lite Pong spill for to spillere, hvor hver spiller har en Teensy. For å utføre dette ble det sendt CAN-meldinger mellom dem, slik at de kunne oppdatere hverandre om spillets status. For CAN-implementasjonen til denne oppgaven, ble ”canSnifferne” videreført med logikk slik at de kunne sjekke hvilken type melding som gikk over nettverket.

Spillet ble utviklet slik at begge spillene bevegde sin paddle på høyre side av skjermen, som ble oppnådd ved å speilvende informasjonen om ballposisjonen i horisontal retning. Det ble lagt til logikk slik at første spiller til å trykke in joysticken på brettet blir host; det vil si at deres Teensy sender informasjonen. ”Slave”-Teensyen mottar informasjon om ballposisjon som blir beregnet og sendt fra ”host”-Teensyen. Ballens bevegelse blir endret ved å sette en fart i x-retning og y-retning, og oppdatere posisjon fra dette i løkka.

Videre ble det implementert ”**ball-fysikk**”, hvor ballen spretter tilbake fra paddlen med forskjellig vinkel avhengig av hvor den traff. For å oppnå dette ble det først satt et nullpunkt i midten av hver paddle, da koordinatene deres startet i øverste venstre hjørne (fra Adafruit GFX bibliotekets ”firkant-tegne funksjon” [3]), som vist i (3.4).

$$padMid = padY + \frac{padHeight}{2} \quad (3.4)$$

Deretter differansen mellom ballposisjonen og midtpunktet til paddle, skalert om til radianer for de trigonometriske funksjonene:

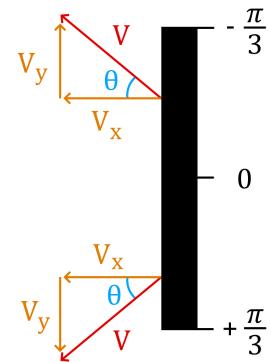
$$\theta = (yPosBall - padMid) \cdot \left(\frac{\pi \div 3}{padHeight \div 2 + ballRadius} \right) \quad (3.5)$$

Til slutt blir denne vinkelen brukt for å kalkulere de to nye komponentene til ballens fart:

$$V_x = -ballMagnitude \cdot \cos(\theta) \quad (3.6)$$

$$V_y = +ballMagnitude \cdot \sin(\theta) \quad (3.7)$$

Hvor magnituden ble satt til en konstant, slik at ballen alltid har samme fart, men forskjellig vinkel. Da ballfunksjonen var implementert, ble det lagt til et scoringssystem som viser spillernes score i toppen av skjermen. Første spiller til 10 poeng vinner, hvor en "end-screen" ble lagt til. Både på denne og "startup-screen" ble det animert en ball som spretter og svever ved hjelp av sinusfunksjoner.



Figur 3.7: Ballens fysikk avhengig av hvor den treffer paddle

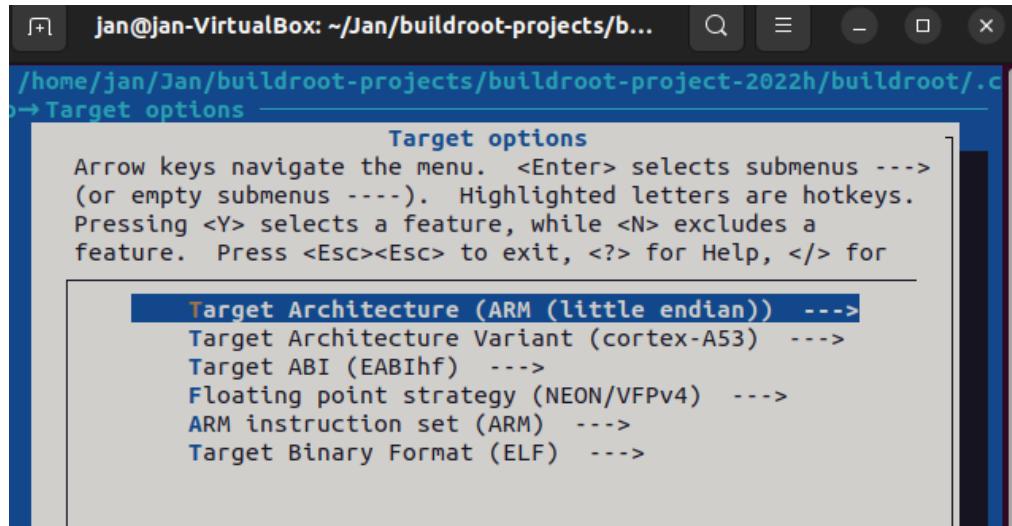
3.3 Delprosjekt 3

3.3.1 Oppsett

Buildroot versjon 2022.08.01 anskaffes og pakkes ut. Terminalen navigeres til buildroot folderen og kommandoene:

```
make raspberrypi3_defconfig
```

Det noteres at prosessorarkitekturen er listet i "target options" menyen som "ARM" med prosessor-typen Cortex-A53.



Figur 3.8: Raspberry PI Systemarkitektur

Det halvferdige buildroot prosjektet anskaffes fra Canvas og pakkes ut. Samtlige filer fra "buildroot-2022.08.01" buildroot mappen kopieres og limes inn i den tomme buildroot folderen i "buildroot-project-2022h"

Filen "mas245-2022h-can-dropbear-static-ip.config" endres navn til ".config" og flyttes til "buildroot-output" folderen. Kommandoene:

```
make BR2-EXTERNAL=\${PWD}/external -C \${PWD}/buildroot menuconfig
```

kjøres, og det observeres at selv om Can_Pingpong programmet nå er tilstede i oppsettet, er ingen av de nødvendige pakkene lagt til, og forhåndsinnstilt passord er heller ikke på plass. Dette skyldes at .config filen som ligger i foretrukket outputmappe inneholder informasjon for menuconfig programmet, og uten filstien vet ikke menuconfig hvor den skal lete. Uttrykket

```
\${PWD}
```

refererer til nåværende absolutt filsti fra terminalen.

Kommandoene:

```
make O=\${PWD}/buildroot -output BR2-EXTERNAL=\${PWD}/external -C ...  
\${PWD}/buildroot menuconfig
```

kjøres. Nå er passord og alle nødvendige pakker for Can_Pingpong automatisk lagt til i menuconfigoppsettet. Det navigeres til "buildroot-output" mappen, og byggeprosessen gjennomføres.

Operativsystemet overføres til minnekort med "disk destroyer" funksjonen dd. Før utløsning av minnekortleseren redigeres config.txt i /boot mappen til å inneholde nødvendig informasjon for å starte opp CAN-busen.

```
# CAN controller  
dtoverlay=mcp2515-can0,oscillator=16000000,interrupt=25  
dtoverlay=spi-bcm2835
```

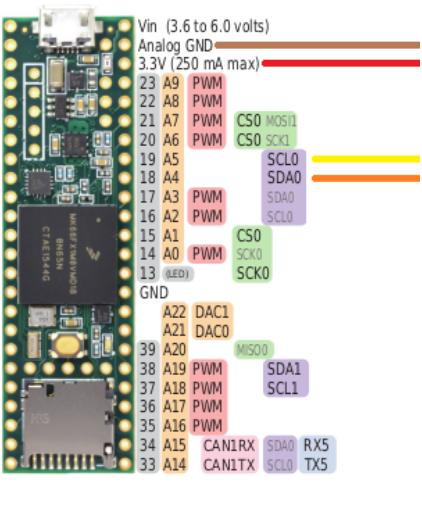
For å SSH inn på Raspberry Pien må nettverksadapteret settes til bridge modus på den virtuelle maskinen. Om NAT adapteret er aktivt samtidig ser terminalen ut til å prioritere dette for ping og ssh forsøk. Ved å deaktivere NAT og kun ha ethernetkabeladapteret aktivt oppnås det kontakt med Raspberry. Modulene og konfigurasjon for CAN grensesnitt lastes inn med:

```
Modprobe spi_bcm2835  
modprobe mcp251x  
/sbin/ip link set can0 up type can bitrate 250000
```

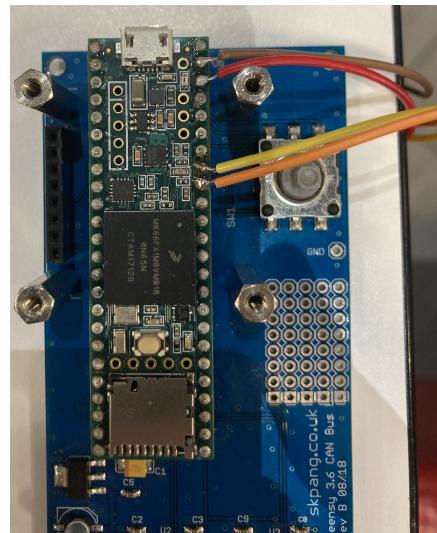
Can-bus funksjonalitet bekreftes ved å sende CAN meldinger gjennom PCAN-view til Raspberry som vises gjennom candump can0 funksjonen. Før oppkoblingen av D-SUB CAN adapteren sjekkes det for intern termingeringsmotstand på Raspberry. Knutepunktet JP2 er ikke koblet, så termingeringsmotstand legges til. Se krets skjema i A.4.

3.3.2 Kommunikasjon mellom to CAN-noder

Gruppen velger en IMU av modell MPU6050 som sensor for å generere data som kan sendes til Raspberry Pien. IMUen loddet med ledninger til toppen av teensy3.6 brettet.



Figur 3.9: Teensy skjema [24]



Figur 3.10: Teensy loddning

For å hente ut data fra MPU6050en anvendes biblioteket fra Electronic Cats [10]. Dataen som hentes ut fra IMUen blir først konvertert fra float til en string, slik at hver karakter kan hentes ut individuelt og lastes inn i en tilsvarende CAN-melding data buffer. En variabel mengde karakterer sendes ut fra IMUen avhengig av om akselerasjonen er positiv eller negativ, eller flere siffer. Minste mengde karakterer oppstår ved ensifret positiv akselerasjon hvor output har lengden 4, f.eks 9.81. Lengste mulige melding innenfor rimelighetens grenser ($< 100 \frac{m}{s^2}$), har en lengde på 6 karakterer, f.eks -30.85. For å ta høyde for disse avvikene finnes lengden på stringen, som så brukes til å definere hvor lenge løkken som transporterer karakterer fra string til CAN skal kjøre, samt definerer lengden på CAN-meldingen. Operasjonssekvensen gjøres en gang hvert 3. sekund.

På mottakende ende kjører Raspberry Pi en periodisk funksjon som plukker opp meldinger fra CAN RX-bufferen. Nødvendig syntaks for bruk av SocketCAN hentes fra dokumentasjonen på [4]. Denne funksjonen kjøres hyppigere enn hvert 3. sekund, siden en tregere avlesningsrate vil føre til en kø av meldinger i mottaksbufferen, og eventuell overflow. Igjen benyttes lengden på meldingen til å styre hvor lenge løkken som henter ut siffer skal kjøre. Karakterene som sendes over CANbusen blir representert som heksadesimaler. Dette gjør det lett å gjenopprette opprinnelig melding, da alt som må gjøres er å legge hver karakter over i en string array som kan konverteres til en float. Det utføres så en matematisk operasjon på verdien, og prosessen gjentas for å sende dataen tilbake til teensy3.6en. Gruppen noterer seg at CAN-bus er en broadcast only, og at flere enheter ville skapt krøll på systemet da det for øyeblikket ikke har blitt implementert noen filtreringsfunksjon for hvilke meldinger som plukkes opp av read-funksjonen. Ved oppkobling av mer enn 2 enheter på CAN nettverket ville dette vært strengt nødvendig.

Ved kompilering av Raspberry Pi programmer oppstår det en srror ved bruke av frame.len medlemsvariabelen.

```
main.cpp:47:38: error: 'struct can_frame' has no member named 'len'
47 |         int meldingLengde = receiveFrame.len;
```

Figur 3.11: Caption

Ved inspeksjon av CAN structen i can.h headerfilen, finnes det at .len er pakket inn i en ”union” brakett sammen med .can_dlc.

```

struct can_frame {
    canid_t can_id; /* 32 bit CAN_ID + EFF/RTR/ERR flags */
    union {
        /* CAN frame payload length in byte (0 .. CAN_MAX_DLEN)
         * was previously named can_dlc so we need to carry ...
         * that
         * name for legacy support
         */
        __u8 len;
        __u8 can_dlc; /* deprecated */
    };
    __u8 __pad; /* padding */
    __u8 __res0; /* reserved / padding */
    __u8 len8_dlc; /* optional DLC for 8 byte payload length ...
                     (9 .. 15) */
    __u8 data[8] __attribute__((aligned(8)));
};

```

For å unngå problemstillingen, kopieres structen til main.cpp og redigeres til følgende.

```

struct can_frame_edit {
    canid_t can_id; /* 32 bit CAN_ID + EFF/RTR/ERR flags */
    __u8 len;
    __u8 __pad; /* padding */
    __u8 __res0; /* reserved / padding */
    __u8 len8_dlc; /* optional DLC for 8 byte payload length (9 .. ...
                     15) */
    __u8 data[CAN_MAX_DLEN] __attribute__((aligned(8)));
};

```

Dette løser problemstillingen fullstendig.

4 Resultater

4.1 Delprosjekt 1

4.1.1 Komponentvalg

Tabell 4.1: Komponentvalg for delprosjekt 1

Navn/verdi	Komponent type	Antall	Datablad
ATMEGA168-20PU	Mikrokontroller	1	Mouser [6]
LM2940CT	5 [V] LDO Regulator	1	Texas Instruments [14]
RND 150KSK100M220D11S	22 [μ F] Elektrolyttkondensator	1	RND Components [5]
N/A	0.1 [μ F] kondensator (upolarisert)	7	N/A
L-7113SRD-D	rød LED, 5 [mm]	1	Kingbright [16]
1N4001	Diode	1	Vishay [29]
330 [Ω]	Motstand	2	N/A
10 [$k\Omega$]	Motstand	1	N/A
Pin header	90° knekk	1	N/A
Tactile push button	Knapp	2	N/A
2 [$k\Omega$] Pot	Potensiometer	1	N/A

Mikrokontroller og LDO ble utdelt fra utstyrsboks i lab. På grunnlag av databladet til LDO, ble den første kondensatoren med 22 [μ F] valgt [14]. Siden det var mangel av den anbefalte 470 [n F] kondensatoren som var anbefalt i databladet [14], valgte gruppen å parallellkoble tre 100 [n F] kondensatorer. Ifølge faglærer vil dette være tilstrekkelig for de grunnleggende oppgavene prosjektet skal dekke. Dette resulterer i en verdi tilsvarende summen av dem:

$$C_1 + C_2 + C_3 = 3 \cdot 100 = 300[nF]$$

Verdien til den strømbegrensende motstanden for lysdioden ble valgt slik at den ikke brenner opp. Avlesninger fra diagram til LEDen viser sammenhengen mellom Forward Current og Forward Voltage [16]. Ved avlesning på grafen:

$$R_{LED} = \frac{\Delta V}{I_{ForwardDrop,LED}} = \frac{V_{LDO} - V_{ForwardDrop,LED}}{I_{ForwardDrop,LED}} = \frac{5 - 1.8[V]}{0.01[A]} = 320[\Omega] \quad (4.1)$$

Gruppen benyttet da en 330 [Ω] motstand.

Strømforsyningen ble satt opp med spenning lik 7.5 [V], da spenningen på LDO skal være mellom 6.25 og 26 [V] ifølge databladet [14]. Strømforsyningen har også en innstilling for øvre strømgrense som ble satt til 60 [mA], i tilfelle en kortslutning.

For mikrokontrolleren ble verdien til avkoblingskondensatorene satt til 100 [n F] med inspirasjon fra designforslag til AVR Hardware Considerations [8]. For reset pin skal pullup resistoren ifølge databladet til ATMEGA168-20PU ha verdi mellom 30 [$k\Omega$] og 60 [$k\Omega$] [6]. Den faktiske verdien ble målt til 37 [$k\Omega$]. Det ble valgt en 14001 diode med maksimalt 50 [V] likespenning og maksimalt 30 [A] strøm eller 1 [A] gjennomsnittlig forward likerettet strøm. Kondensatorverdien til reset pin ble satt til 100 [n F], etter design forslag fra AVR Microcontroller Design Considerations [8]. Videre ble seriemotstanden satt til 330 [Ω], ved inspirasjon fra hardware considerations [8].

Motstand og kondensator er et første orden system med en tidskonstant τ :

$$\tau = R_{pullup} \cdot C = 37000[\Omega] \cdot 100 \cdot 10^{-9}[F] = 3.7[\text{ms}] \quad (4.2)$$

Dette gir følgende settling time:

$$T_s = \tau \cdot 4 = 14.8[\text{ms}]$$

4.1.2 Oppkoblinger

ISP oppkoblingen nevnt i 3.1.3 har følgende sammenheng:

Tabell 4.2: Pin-relasjoner fra Atmel ICE User Guide [7]

SPI Navn	SPI Pin	Mini-Squid Pin	MCU Pin
/RESET	5	6	1
V _{cc}	2	4	7
GND	6	2	8
PDI/MOSI	4	9	17
PDO/MISO	1	3	18
SCK	3	1	19

Den fullstendige kretsen til LDO for lysindikasjon, uten tilkobling til MCU er vist i figur A.1. For mikrokontrolleren ble det først laget et kretsskjema for de første oppgavene, deretter ble det bygget videre på samme skjematikk, vist i figur A.2.

4.1.3 Microchip Studio til MCU

Ved programmering av mikrokontrolleren blir koden hentet som en .elf fil fra "debug" mappen til programmet. Dette er en "executable" fil, som er brukt for Atmel AVR-arkitektur [30]. To bokser er krysset av, hvor den ene tilbakestiller mikrokontrolleren før opplasting og den andre verifiserer at den opplastede koden er identisk til den som ble sendt (med andre ord at den er blitt sendt riktig). Det ble verifisert den spesifikke 3-byte signatur koden til mikrokontrolleren, som var 0x1E9406. Ved avlesning i datablad, tilhører denne device signaturen ATMEGA168 [6]. Testprogrammet for å sjekke korrekt oppsett og fungerende building i Microchip Studio er vist i A.4.1.1. "Hello World" programmet som blinker LEDen brukte biblioteket <util/delay.h> fra avr-libc for å sette tid mellom blinkene av LEDen. Denne koden er vist i A.4.1.2.

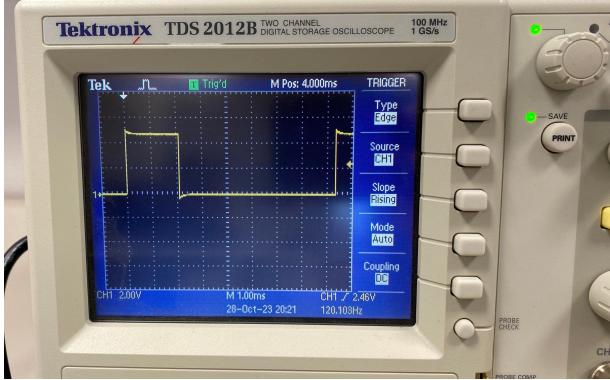
4.1.4 PWM og Soft Blink

For å konfigurerere PWM modus for mikrokontrolleren, måtte kontrollregistre bli satt riktig. Gruppen ønsket å sette mikrokontrolleren i "Fast PWM, 10-bit" modus. Denne modusen krevde tre konfigurasjoner: **Waveform Generation Mode**, **Compare Output Mode** og **prescaler**.

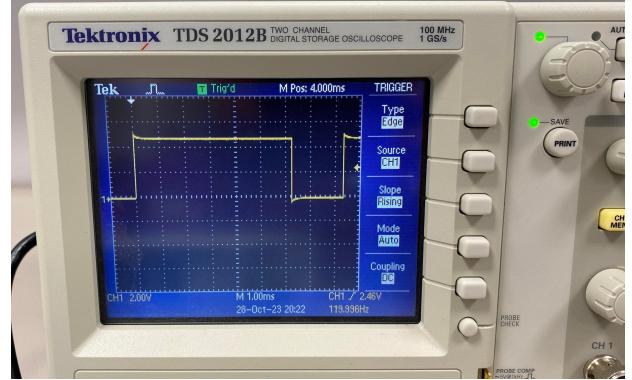
Først ble **Waveform Generation Mode** valgt til mode 7 ved å sette WGM1[3:0] til 0,1,1,1, fra tabell 20-6 i datablad [6]. Denne modusen tilhører Timer 1 (16-bit) fra Output Compare Register 1, som kun er tilgjengelig for PB1 og PB2 (pin 15 og 16), hvor gruppen valgte PB1. Deretter ble **Compare Output Mode** satt til NON-INVERTING Mode ved å sette COM1A[1:0] til 1,0, fra tabell 20-4 i datablad [6]. COM1B trengte ikke konfigurering, da kun PB1 er brukt (OC1A tilsvarer Output Compare Register 1 A, se figur 3.1). Til slutt ble det konfigurert en **prescaler** på $N = 8$. Det tilsvarer å sette CS1[2:0] til 0,1,0, fra tabell 20-7 i datablad [6]. For å beregne forventet frekvens på utgangen, måtte gruppen finne ut av frekvensen som ble sendt inn før prescaling. Ifølge databladet, skal denne frekvensen være resultatet av en intern RC oscillator med 8 [MHz], som er preskalet ned til 1 [MHz] av CKDIV8 fusebit [6]. Dette ble bekreftet i Microchip Studio, hvor man kan se oversikt over fusebits under Device Programming. Dermed kunne gruppen kalkulere forventet frekvens fra databladets formel [6]:

$$f_{OCnx,PWM} = \frac{f_{clkI/O}}{N \cdot TOP} = \frac{1[\text{MHz}]}{8 \cdot 2^{10}} \simeq 122[\text{Hz}] \quad (4.3)$$

PWM-konfigurasjonen var satt opp i Microchip Studio, som vist i A.4.1.3, og kretsen ble oppkoblet sammen med lab-strømforsyning og et oscilloskop. Oscilloskopet ble brukt for å verifisere forventet frekvenser og PWM-signal. Før programmet ble satt opp med variabel duty cycle fra sinusfunksjonen, ble det satt konstant 25% og 75% respektivt:



Figur 4.1: Oscilloskop, 25% Duty Cycle



Figur 4.2: Oscilloskop, 75% Duty Cycle

Da korrekt PWM signal var verifisert, ble det lagt inn den variable duty cyclen fra sinusfunksjonen. For å få en riktig tidsoppdatering på den, ble det lagt inn et vilkårlig tidsinkrement. Deretter ble det målt tiden for en periode, og det nye tidsinkrementet ble skalert med forholdet til den ønskede og målte perioden. Dette ble gjort iterativt, til ønsket kvalitet.

Demonstrasjonsvideo er vist i A.2.1.1.

4.1.5 Pin change interrupt og tilstandsbytting av lysdiode

I ISR løkken ble en if setning som sjekker om knappen er trykket ned eller opp laget, slik at en synkende flanke ikke skrudde av LEDen da knappen ble sluppet. Da knappen ble trykket ned så ble en integer variabel inkrementert, hvor main programmet sjekket om den var oddetall eller partall ved bruk av modulo operatoren. Dette ble brukt siden det var problemer med å flippe en boolean i ISR løkken.

Prell ble observert fra knappen, hvor det registrerte signalet fluktuerte, derfor ble et system som setter en bolsk variabel sann og settes lav igjen etter 150 [ms] har gått. Da bytter lyset bare når booleanen er satt til false; det betyr at lyset kan maksimalt bytte tilstand med en frekvens på 6.67 [Hz]. Interrupt-programmet er vist i A.4.1.5, mens polling-programmet er vist i A.4.1.4.

4.1.6 ADC og variabel PWM settpunkt ved potensiometer

Kondensator på 100 [nF] ble satt opp mellom jord og AREF for å fjerne støy, da det benyttes felles referanse. ADC-skaleringen fra bits til spenning ble gjort som følger:

$$ADC_V = ADC_{bit} \cdot \frac{V_{CC}}{\max Bit} = ADC_{bit}[bit] \cdot \frac{AREF[V]}{2^{10} - 1[bit]} \quad (4.4)$$

Koden til ADC er vist i A.4.1.6. Demonstrasjonsvideo er vist i A.2.1.2.

4.2 Delprosjekt 2

Eksempelkode ferdig kompilert, med OLED-skjerm er vist i figur 4.3. Koden hvor det ble tegnet en bue er lagt ved i A.4.2.2 og resultatet er vist i figur 4.4. For-løkka med sinusfunksjonen resulterte i en animasjon på skjermen.



Figur 4.3: Kompileringssjekk, modifisert prosjektkode



Figur 4.4: OLED oppgave med bruk av bibliotek fra Adafruit

Videre ble det tatt et skjermdump for å vise hvordan informasjonen om sendte og mottatte CAN meldinger ble vist. I figur 4.5 er meldingene vist i PCAN-View øverst, og i Serial Monitor fra PlatformIO under. Meldingen ble sendt med ID 245 med en periode på 2.5 [sekunder] (tidligere testet med høyere frekvens, gruppen har ikke kjørt den i over 5 timer).

Receive							Tx		
CAN-ID	Type	Length	Data	Cycle Time	Count		Trigger	Comment	
245h		4	12 34 56 78	2000,0	75				
Connected to hardware PCAN-USB FD, Device ID 21h									
Loop number: 10									
- Message transmitted:	Mailbox 8	Overrun: 0	Length: 4	ID: 245	Buffer: 12 34 56 78				
- Message received:	Mailbox 99	Overrun: 0	Length: 4	ID: 245	Buffer: 12 34 56 78				

Figur 4.5: "Receive" og "Transmit" CAN meldinger vist henholdsvis i PCAN-View og Serial Monitor

Til slutt ble spillet testet, det ble gjort med en spiller/hånd for å ikke blokkere sikten for videoen. Video er lagt ved i A.2.2.1 og kode er lagt ved i A.4.2.3.

4.3 Delprosjekt 3

Det oppstår hyppig problemet med å få satt i gang ssh tilkoblingen mellom Raspberry Pi og virtuell Linux maskin, men problemstillingen lar seg løse ved hjelp av å skru av og på enheter i tilfeldig rekkefølge frem til Raspberryen responderer på ping. Ved tilkobling av skjerm viser Raspberry Pi oppstartsskjermen initialisering av eth0 porten, og det regnes med at feilkilden er den virtuelle maskinen. Både operativsystem og programvare oppfyller sine oppgaver, som demonstrert i vedlegg A.2.3.1. Det noteres at noen små avrundingsfeil oppstår når data returneres fra Raspberry, f.eks at produktet 18.14 blir sendt tilbake som 18.13. All kode for delprosjekt 3 er lagt ved i A.2.3.

5 Diskusjon

5.1 Software Arkitektur

I prosjektet har det generelt sett fokuset vært på funksjonaliteten av de forskjellige delprosjektene. I større prosjekter er det sterkt anbefalt å benytte klasser og dele systemet opp i biter. Det gjør systemene mer organisert og begrenser tilgangen til variabler og funksjoner slik at de kun kan benyttes der de skal brukes. Delprosjektene brukta hovedsakelig globale variabler, som ikke vanligvis er best praksis, men siden det har vært isolert med relativt små oppgaver ble det ansett som akseptabelt slik at alle oppgavene ble fullført. Unntaket her er at pong-spillet fra delprosjekt 2 kunne gjerne blitt optimalisert, gitt mer tid.

5.2 Delprosjekt 1

5.2.1 Komponentvalg

Elektrolytt kondensatorer ble forsøkt unngått, da de har større ESR (ekvivalent serieresistans) og tregere reaksjonstid. Det gikk i de fleste tilfeller, utenom 22 [μF] da dette er en såpass stor kapasitans at det er vanskelig å få tak i alternativer til elektrolytter.

LDO regulatoren ble valgt med 5 [V], da gruppen hadde muligheten for mikrokontrollere som var ratet for spenninger tilsvarende begge LDO regulatorene. 5 [V] regulatoren kan gi mulighet for bedre opplosning og mer stabile signaler, sammenlignet med 3.3 [V]. Siden prosjektene er ganske grunnleggende og ikke har noe vesentlig strømtrekk var det betraktet som tilstrekkelig. Dessuten hadde lab-strømforskyningen muligheten til tilføre systemet ønsket strøm opp til det satte maksimale strømtrekket.

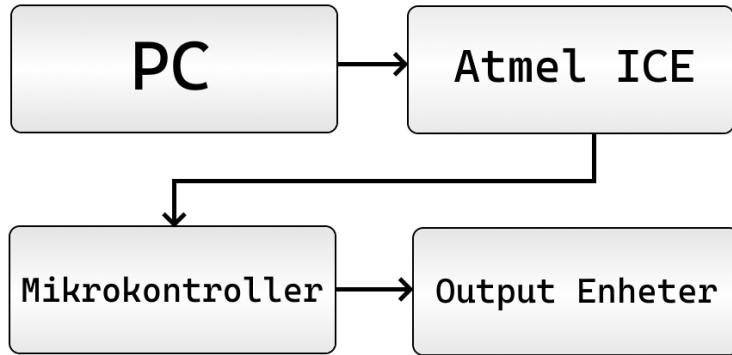
Gruppen jordet input og output til LDO separat, da felles jording potensielt kunne resultert i galvanisk støy fra strømkilden. Dette er hovedsakelig grunnen til bruk av LDO, da den gjør spenningsfluktusjoner om til varme og gir konstant spenning ut. Den strømbegrensende motstanden for lysdioden ble valgt til 330 [Ω], som er tilstrekkelig da det er langt innenfor de maksimale ratingene, sammenlignet med avlesning og beregning fra datablad [16]. Det er spesielt tilstrekkelig senere i prosjektet, da den blir koblet opp til PWM signaler som resulterer i lavere gjennomsnittlig effekt. Dette gjenspeiles i databladet, hvor forward current er langt høyere for PWM signaler enn konstant verdi. Eksempelet der gir 30 [mA] DC Forward Current og 155 [mA] Peak Forward Current ved 10% duty cycle, 1 [ms] pulsbredde [16].

Avkoblingskondensatorene vist i figur 3.3 ble brukt for å redusere støyen til mikrokontrolleren, da handlinger som PWM switching vil føre til støy på strømforskyningslinja. Kondensatoren supplerer med spenning i tilfelle spenningsfall.

Alle generelle I/O-pins har intern ESD-beskyttelse i form av dioder til jord og Vcc, men for reset-pin er det kun til jord, som er grunnen til diode oppkoplingen. Dioden ble ansett som tilstrekkelig, da kretsen trolig ikke overstiger de nominelle verdiene. Kondensatoren glatter ut støy, i tilfelle intern lavpassfilter i mikrokontrolleren ikke er tilstrekkelig. Når reset knappen blir trykket, kortslutter det kondensatoren og kan ha høye peak verdier. Dette fører til prell i bryteren, slik at det vil bli generert peak verdier i perioder mellom 2 og 10 [ms], som kan begrenses av motstanden [8]. Tatt i betraktning at settling tiden RC-oppkoplingen er 14.8 [ms], konkluderte gruppen med at verdiene var tilstrekkelige, da reset knappen ikke forventes å bli trykket så frekvent.

5.2.2 Programmering

Programmering ble gjort ved hjelp av ISP. Dette er en protokoll hvor det er nødvendig med et programmeringsverktøy som Atmel ICE. Da brukes en datamaskin til å krysskompilere (upload) koden til mikrokontrolleren gjennom programmeringsverktøyet (Atmel ICE) ved hjelp av ISP-protokollen. Da vet mikrokontrolleren hva den skal gjøre gjennom koden, og sender signaler ut til enheter (eller inn ved målingsenheter). I dette tilfellet er datamaskinen som sender ut koden verten (host) og målenheten er mikrokontrolleren (target). Oversikten er vist i figur 5.1.



Figur 5.1: Oversikt over programmeringsoppsett

5.2.3 Source vs. Sink

For LED outputs ble det benyttet sourcing, da LEDen ikke trenger noe vesentlig strømtrekk. Ved sourcing vil polariteten være vanlig oppkoblet og den vil lyse ved å sette output høy og slukke ved output lav. Dette gir en intuitiv funksjon som er lettere å feilsøke. I de fleste andre tilfeller er det mer hensiktsmessig å benytte sinking for mikrokontrollere. Det gir mulighet for mer pålitelig energi med mindre støy, samt muligheten for flere spenningskilder med forskjellige egenskaper som spenning og strømtrekk. Faren ved sinking er at det er større sannsynlighet for å koble feil og kortslutte komponentene. En siste nevneverdig ting angående dette, er at ikke alle pins (eller mikrokontrollere) har mulighet for både sink og source. Derfor er det lurt å undersøke hva man skal bruke før oppkobling.

5.2.4 PWM og Soft Blink

Som vist i figur 4.1 og 4.2, er frekvensen ganske lik den beregnede fra 4.3. Det at den er litt avvikende kan være mange grunner til, da det er et komplekst system med mange komponenter. For eksempel kan varierende klokkesignal og støy være en faktor. I dette prosjektet er dette langt innenfor toleransen, da det resulterer med under 1% differanse mellom faktisk og beregnet frekvens. Om man ønsker mer pålitelig klokkesignal til system med høyere krav, kan det være lurt å legge til en annen klokkekilde, som en ekstern krystalloscillator. Overshoot i spenning blir også observert ved både stigende og synkende flanke. Ved grov beregning ser dette ut til å være under 10% overskredelse, som anses å være akseptabelt. Nok en gang kan det være mange grunner til det i et slikt system, som for eksempel kapasitans som utlader og gir en ekstra "boost". "Fast PWM Mode" ble brukt for dette prosjektet, da det gir presis og høy-frekvent output med enkel kontroll.

Programmeringen av "Soft Blink" ble gjort ved hjelp av et sinussignal som oscillerte rundt 50%. Siden menneskets øyne reagerer logaritmisk og ikke lineært [15], kunne et lineært varierende settpunkt for PWM vært tilstrekkelig, selv om det er en lineær sammenheng mellom lysstyrke og strøm. Da ville øyet oppfattet det som en myk og jevn pulsing på grunn av menneskeøyets funksjon, selv om det i realiteten var mer hakkete. Tidsinkrementet ble gjort "quick and dirty" med å skalere om til ønsket verdi. Det kunne også blitt gjort ved bruk av en variabel som telte oppover i hovedløkka, for så å benytte en interrupt protokoll for å få eksakt måling, om bedre kvalitet hadde vært ønsket.

5.2.5 Interrupt

Det ble benyttet to metoder for å bytte tilstanden til LED lyset med en knapp; en metode hvor polling blir brukt og den andre med interrupts. Det ble observert at metoden med interrupts var langt mer responsiv til trykk på knappen. Oppsettet med interrupt protokoll for stigende og synkende flanker resulterte i mer kompleks kode sammenlignet med polling. Som nevnt i 4.1.5 ble det lagt inn logikk slik at tilstanden til lysdioden ble endret ved stigende flanke, men kun mulighet for endring hvert 150 [ms]. Grunnen til dette var at det ble observert prell ved knappetrykkene, som refererer til støy i form av at signalet gikk av og på under trykk. Dette er et fenomen som er velkjent, som det finnes flere løsninger på. I dette prosjektet ble det benyttet en timer-løsning, men en alternativ løsning kunne vært å legge inn et lavpassfilter (eller båndpassfilter) som sørget for at så høyfrekvente endringer ikke ble registrert.

Et alternativ til pin change interrupts kunn ha vært å bruke INT0 eller INT1 pinnene, som bruker External Interrupt Request istedefor Pin Change Interrupts; med denne metoden kan man velge å sjekke etter en høy eller lav flanke på signalet til å sende en interrupt, som kunne fjernet litt av kompleksiteten til systemet.

Det kan være ønskelig med minst mulig interrupt rutiner, da det fort gjør systemet komplekst. Hvis man skal ha flere, kan det være mulig å sette opp kø hvor det blir registrert hvilken som kom først, men det er utenfor prosjektets omfang. Selv om det er lurt å begrense dem, er de veldig anvendelige i "real-time" applikasjoner hvor det er viktig å ha presis tidskontroll.

5.2.6 ADC

I ADCen ble det tatt et valg å bruke Free Running mode i stedet for manuelt aktivert ADC. Free Running mode skrur på ADCen når det oppdages en endring av signalet, mens manuelt aktivert ADC kan konfigureres til å bruke ADCen et konstant antall ganger per tidsenhet. Ved full rotasjon av potensiometeret vil dette føre til at den kjører ADCen mange ganger, men dette er ansett som akseptabelt da koden kun brukes for denne funksjonen. Ved bruk i mer komplekse systemer, ville muligens manuelt aktivert ADC vært mer anvendelig da det oppdateres med en fast frekvens, som kan kreve mindre prosessorkraft og gjør systemet mer pålitelig.

5.3 Delprosjekt 2

For oppkoblingen av CAN bus ble det brukt termineringsmotstander i begge ender. Carrier-kortet (SK Pang) har egne knutepunkt som kan kortsluttes med en liten ledningsstamp for å aktivere innebygde termineringsmotstander, JP1 og JP2 (se figur A.3). Siden det var ønskelig å unngå loddning på carrier-kortene, ble det skrudd fast fysiske motstander i connectorene. Størrelsen på termineringsmotstandene ble satt til 120 [Ω], da det er en vanlig verdi som er mye brukt for mindre lengder. For å beregne eksakte motstandsverdier, kan man benytte transmisjonslinje teori, som innebærer komplekse ligninger som er utenfor pensum. Det kan være aktuelt i andre situasjoner, spesielt når lederene er såpass lange at signalene reflekterer og integriteten går ned. "The term applies when the conductors are long enough that the wave nature of the transmission must be taken into account." [34]

Slike bibliotek som ble brukt i prosjektet er veldig nyttige, da det gjør prosjekter som dette mer effektive og enklere. Det er også en veldig god læringsressurs for å gi en introduksjon til mer komplekse temaer, som CANbus. Gruppen lærte mye av å se gjennom bibliotekene hvordan de var satt opp med grensesnitt og funksjonalitetene deres. Verktøy som PCAN-View er veldig nyttige, da muliggjør feilsøking utenfor koden. Det gjør det enklere å isolere feilene, som effektiviserer design prosessen.

Prosjektet ble gjort med en meget uoptimalisert kode. Det ble brukt mye globale variabler og for eksempel klasser kunne heller blitt brukt for å gjøre det mer effektivt. Programmene ble skrevet slik for å simplifisere prosjektet, og rekke å gjøre det ferdig innen tidsfristen. Under andre omstendigheter ville gruppen ha gjort koden mer oversiktlig og effektiv, for eksempel ved bruk av grensesnitt som

ville delt opp funksjonaliteten til spillet Pong. Videre avviker programmet fra spesifikasjonen på 100hz da koden kjører og tegnes, før programmet så tar en pause på 10 millisekunder. Pausen mellom hver tegnet ramme blir da tiden det tar å gjøre alle spillkalkulasjonene + 10 millisekund. Det ble gått ut i fra at kalkulasjonene ikke ville ta nevneverdig lang tid, men gitt et mer komplisert program kunne denne løsningen ha påvirket oppfriskningsraten betraktelig. For å oppnå sann 100hz oppfriskningsrate skulle kalkulasjonene og tegningen referert til en intern klokke på teensyen. I Pong-spillet ble det gjort en endring for sjekkingen av kontakt mellom ball og paddle, hvor størrelsen den sjekket over ble økt. Det var på grunn av at om ballens sentrum passerte rett over paddlen, så det ut som den gikk gjennom den. Derfor ble ballens radius lagt til beregningene, inkludert skaleringen til radianer. Det resulterte i at ballen også kunne sprette av kanten på paddlen.

5.4 Delprosjekt 3

En av svakhetene som åpenbarer seg med valgte oppsett er at programmet som leser CAN meldinger på Raspberry Pien må startes opp før IMUen begynner å sende ut data. Ved avvik fra omtenk som oppstartsprosedyre fyller CAN RX bufferen seg med meldinger fra IMU, og det er ikke lenger samsvar mellom sist sendte melding, og sist mottatte melding på teensy3.6. Gruppen formulerer tre potensielle løsninger på problemet. Den første er å implementere en "onReceive" funksjon som leser meldinger så fort som de kommer inn, likt funksjonaliteten som finnes i Flexcan_T4 biblioteket. Den andre løsningen er å kjøre den periodiske funksjonen med større hyppighet, men denne metoden medfører at programmet ikke kommer seg forbi lesesteget hvis det ikke befinner seg noen meldinger i RX bufferen. Siden programmets nåværende funksjon ekslusivt er å lese av meldinger, anses ikke dette som noe stort problem, men gitt at andre kalkulasjoner eller prosesser skulle kjøres samtidig ville dette vært katastrofalt. Den tredje løsningen ville vært å isolere lesingen av CAN meldinger til en separat tråd, hvor det å vente på en CAN melding ikke forårsaker avbrudd i resten av programmet. Sistnevnte løsning anses som beste løsning for fremtidig kompatibilitet med flere funksjoner i programvaren.

Andre svakheter er mangelen på filtrering av CAN meldinger basert på ID. I pong-programmet har filteringsmetoden vært å sortere bort irrelevante IDer etter meldingen har blitt lest. I tilfellet på Raspberry Pi vil denne metoden forårsake problemer siden meldingen fremdeles må plukkes opp med den periodiske funksjonen fra RX buffer, hvor den så blir forkastet eller pakket ut basert på ID. Fra dokumentasjonen ser CAN_RAW_FILTER funksjonaliteten ut til å være en løsning på problemet, men denne stien forble uutforsket grunnet tidsbegrensninger.

6 Konklusjon

Prosjektene ble betraktet som en suksess, da alle oppgaveforslagene ble gjennomført og reflektert.

Delprosjekt 1 ble gjennomført for Atmel mikrokontrolleren, hvor det tok inn strøm fra labforskyning som ble regulert gjennom en LDO-regulator. Delprosjektet la et godt fundament for kretsdesign, og hvor viktig det er å ta hensyn til pålitelige og robuste strømforskyninger, samt støyreduksjon. Det ble også utforsket måter for å sende og motta signaler på forskjellige vis, og hvordan det kan gjøres på en trygg måte uten å ødelegge komponentene, samt ESD-beskyttende tiltak for å verne komponentene. Fra et software-perspektiv ble det utforsket en god del innebygde funksjoner og hvordan disse kan konfigureres ved hjelp av register bytes og fuse bits.

Delprosjekt 2 fokuserte hovedsakelig på bruken av CAN bus protokollen og Teensy. Det ga en kort introduksjon til hvordan det kan kobles opp og viktigheten av matching av nominell impedans. Hovedfokuset var i software, hvor det ble utforsket hvordan CAN protokollen kunne brukes til å sende og motta meldinger ved hjelp av et grensesnitt som flexcan_t4, samt hvordan det kan videre brukes til større ting. Selv om det bare var et lite spill, ga pong-oppgaven en introduksjon til hvordan det kan implementeres kommunikasjon. Det ga gruppen et innblikk i tankeprosessen for slike kommunikasjonsprotokoller.

Delprosjekt 3 ga gruppen en introduksjon til hvordan operativsystemer er satt opp, og hvordan man kan lage en distribusjon basert på Linux. Det utforsket hvordan strukturen deres er bygd opp og hvordan man kan aksessere de forskjellige delene dens. Videre utbytte var forsåelsen av hvordan CAN-bus meldingsstandarden kan tolkes av forskjellige biblioteker på separate enheter, som gjør kommunikasjon enkel gitt en grunnforståelse av CAN-meldingens oppbygning.

A Appendix

A.1 Github Repository

Lenke til github repository: https://github.com/creamytoad/mas245_Projekt

A.2 Videoer

A.2.1 Delprosjekt 1

A.2.1.1 PWM Soft Blink Demonstrasjon

<https://www.youtube.com/watch?v=F80oI1ZD1LE>

A.2.1.2 ADC Demonstrasjon

<https://www.youtube.com/shorts/v5W2hq8G5fk>

A.2.2 Delprosjekt 2

A.2.2.1 Pong demonstrasjon

https://www.youtube.com/shorts/bh_fi18cQM4

A.2.3 Delprosjekt 3

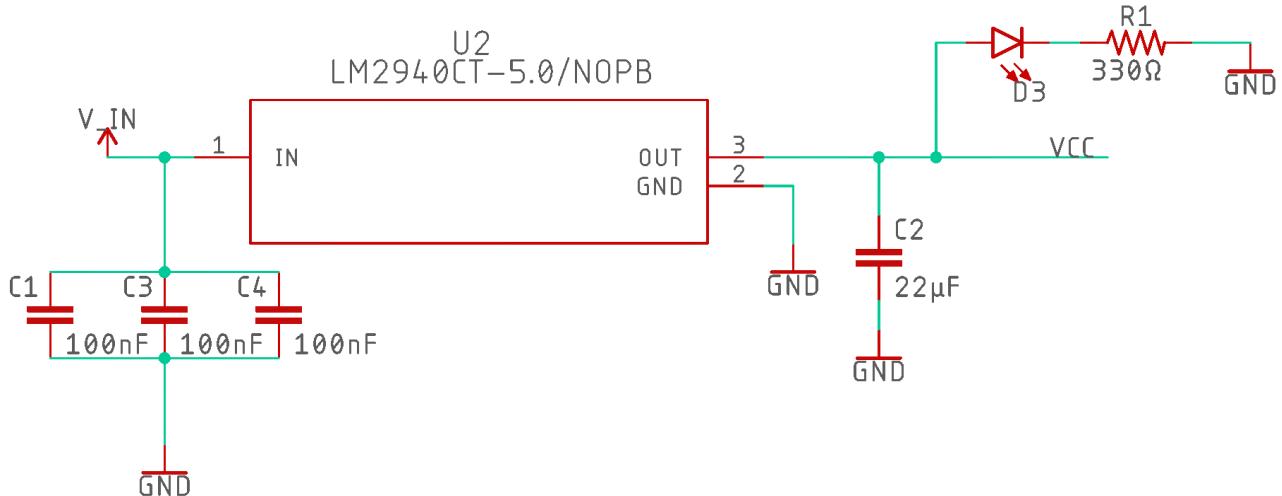
A.2.3.1 IMU-data over Can bus Demonstrasjon

<https://www.youtube.com/watch?v=3Lgbkens8SQ>

A.3 Kablingsskjemaer

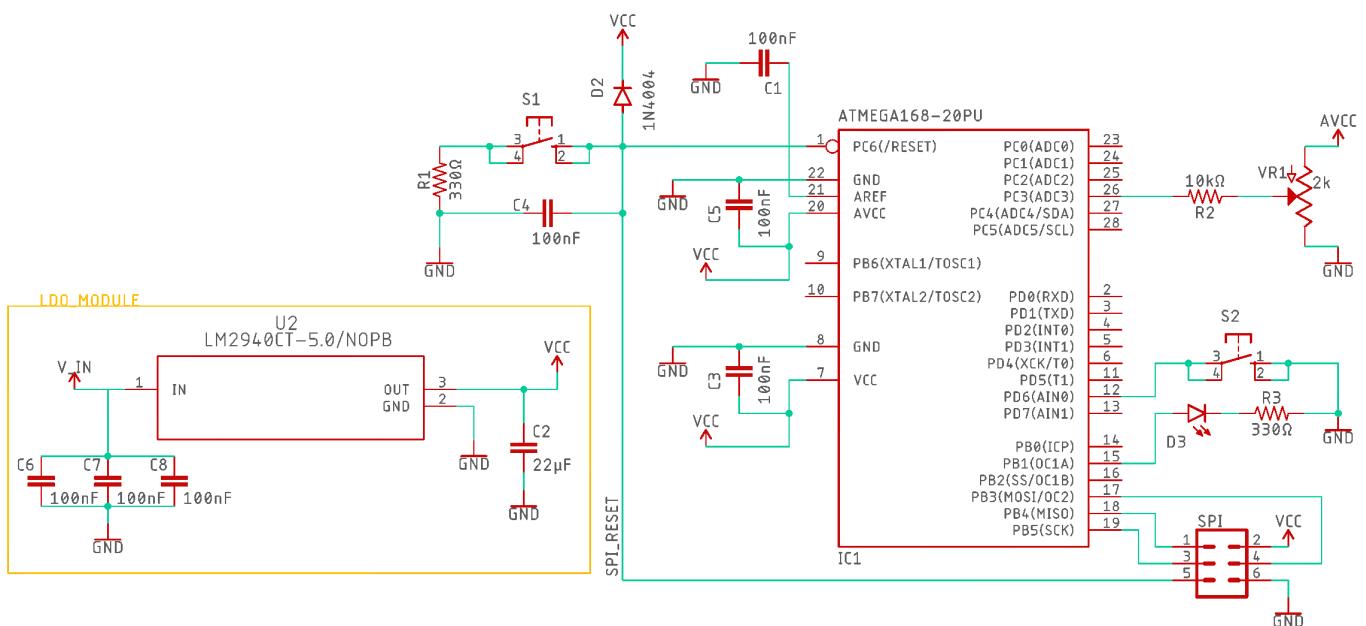
A.3.1 Delprosjekt 1

A.3.1.1 LDO regulator og "Power-on" indikator



Figur A.1: LDO regulator krets, tegnet i Autodesk Eagle

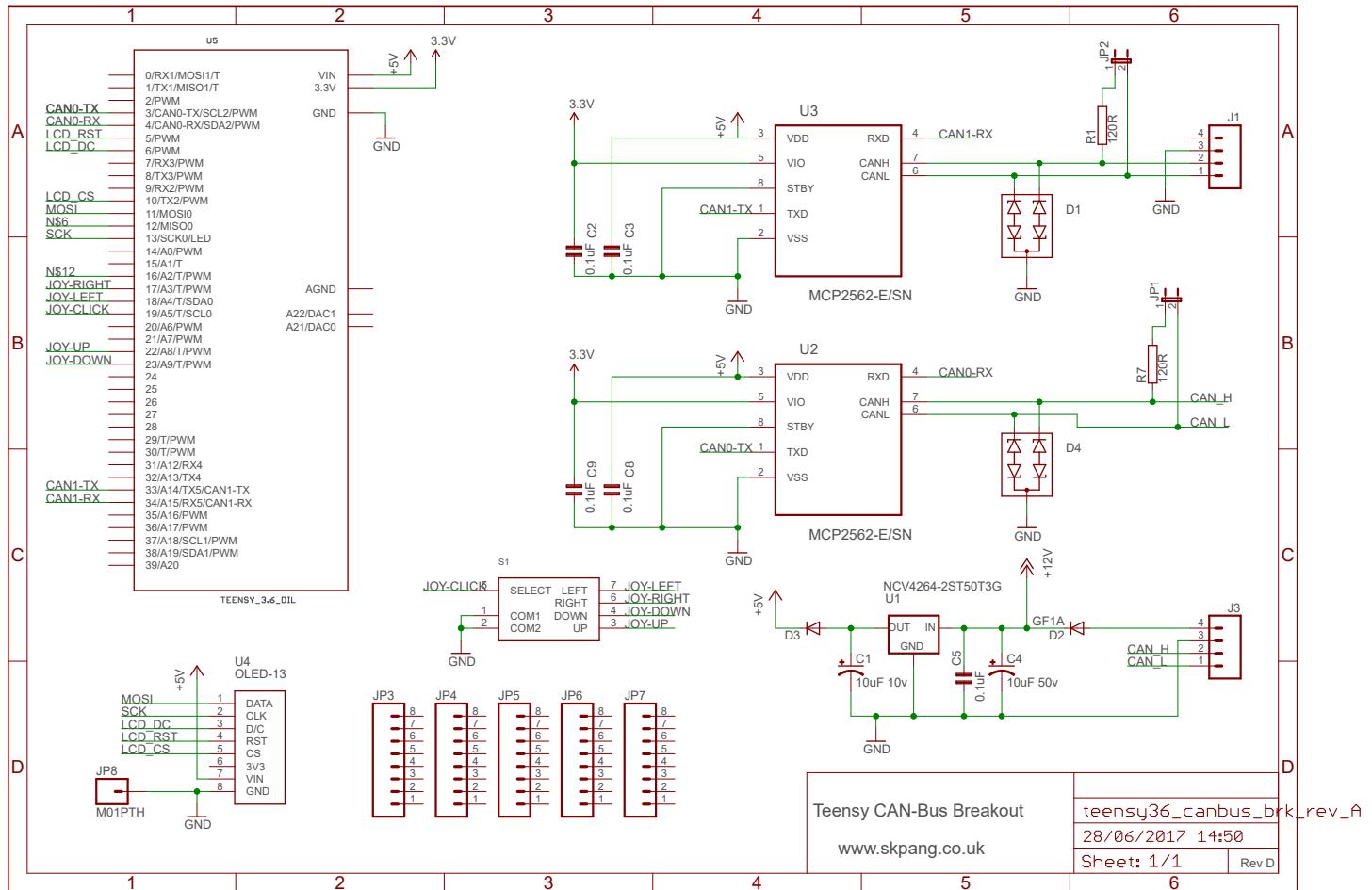
A.3.1.2 Mikrokontrollerkrets



Figur A.2: Mikrokontrollerkrets, tegnet i Autodesk Eagle

A.3.2 Delprosjekt 2

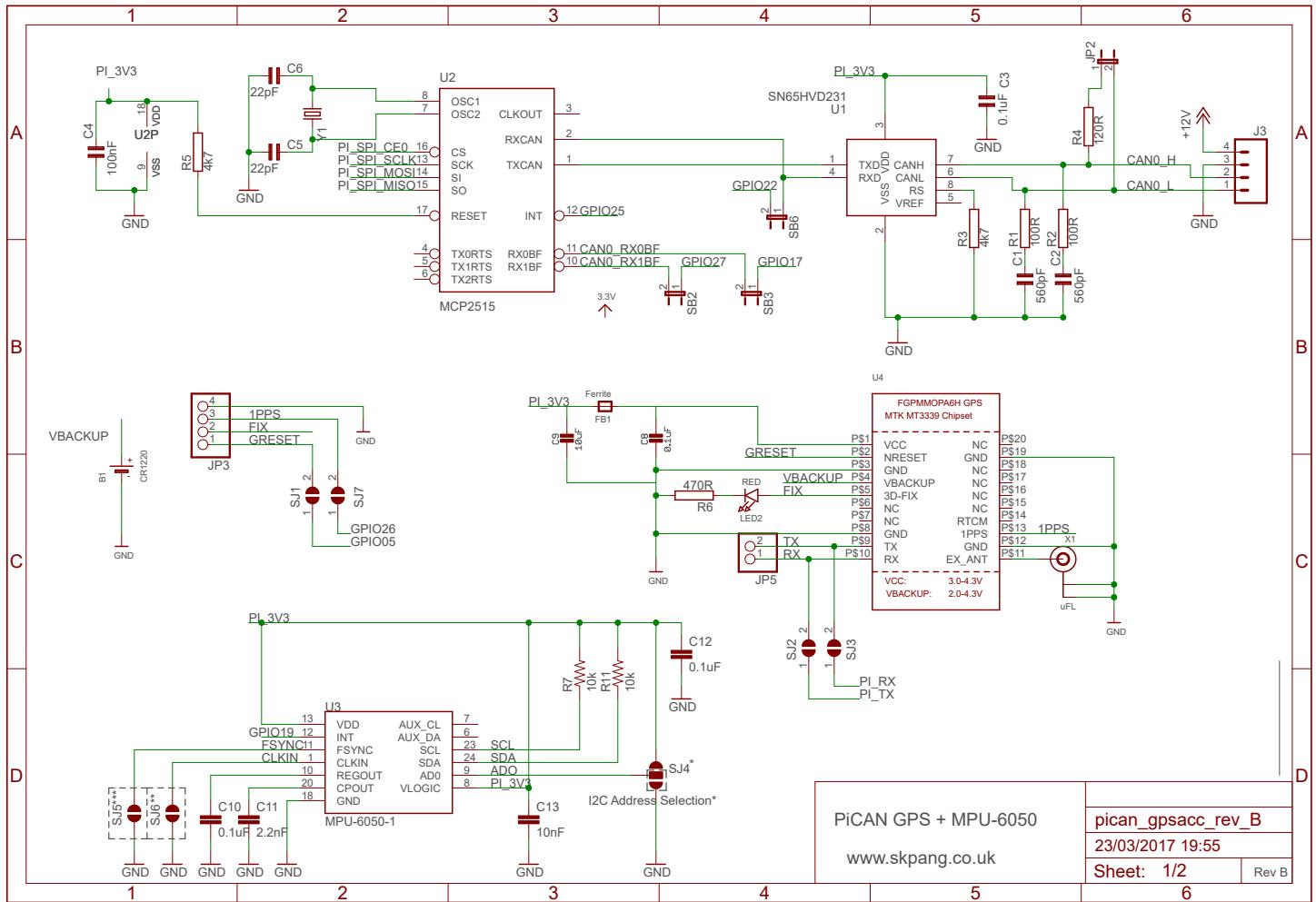
A.3.2.1 Teensy 3.6 Dual CAN-Bus Breakout Board, inkludert Teensy 3.6



Figur A.3: Krets over SK Pang kort og Teensy 3.6 [12]

A.3.3 Delprosjekt 3

A.3.3.1 Raspberry Pi



Figur A.4: Krets til Raspberry Pi [27]

A.4 C++ Kode

A.4.1 Delprosjekt 1

A.4.1.1 Komplilering test kode

```
/*
 * helloWorld2.cpp
 *
 * Created: 17/10/2023 10:43:29
 * Author : mathi
 */

#include <avr/io.h>

int main( void )
{
    int a = 1;
    int b = 1;
    int c;
    c = a + b;
}
```

A.4.1.2 Hello World / Blink

```
/*
 * helloWorld.cpp
 *
 * Created: 17/10/2023 10:43:29
 * Author : mathi
 */

#define F_CPU 1000000 //1MHz, set cpu frequency in Hz, datablad side ...
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>

void init()
{
    DDRB |= (1 << DDB1); //set pin 7 as output in data-direction of port D
    DDRD &= ~(1 << PIND6); //set pin 6 of port D as input
    PORTB &= ~(1<<PB1); //set pin 7 low as initial value
    PORTD |= (1<<PD6); //set pin 6 high to enable pull up-resistor to read state
}

void blink()
{
    PORTB |= (1<<PB1); //high
    _delay_ms(100);
    PORTB &= ~(1<<PB1); //low
    _delay_ms(100);
}
```

```

int main(void)
{
    // initialize
    init();

    bool a = true;
    uint8_t pin6Value = 0;

    while (true) // run forever
    {
        pin6Value = (PIND & (1 << PD6)) >> PD6; // read pin 6 value, ...
                                                    // 0 if low, 1 if high
        if (pin6Value == 0) // if pin 6 of port D is low
        {
            a = ~a; //turn on or off blinking
        }
        if (a)
        {
            blink();
        }
    }
    return 0;
}

```

A.4.1.3 Soft Blink

```

/*
 * PWM_SoftBlink_V2.cpp
 * Created: 25.10.2023 14:26:14
 * Author : Thomas
 */

// CPU frequency is 1 [MHz], after prescaler of 8 with fuse bit.
#include <avr/io.h>
#include <avr/interrupt.h>
#include <math.h> // instead of cmath, less work since std library missing
#define PI 3.14159265

void init()
{

    // ----- PWM Configuration -----
    // Fast PWM mode - 10 [bit] (Mode 7 from table 20-6)
    // Waveform Generation Mode
    TCCR1A |= (1 << WGM11) | (1 << WGM10);
    TCCR1B |= (1 << WGM12);
    TCCR1B &= ~(1 << WGM13);

    // Set Fast PWM Mode NON-INVERTING
    // PB1 --> Only COM1An
    TCCR1A &= ~(1 << COM1A0); // COM1A0 = 0
    TCCR1A |= (1 << COM1A1); // COM1A1 = 1

    // Set prescaler:
    // f_Clk = 16 [MHz], f_des = 250 [kHz]
    // Ratio = prescaler = N = 16/0.25 = 64
    TCCR1B |= (1 << CS11);
    TCCR1B &= ~(1 << CS12) & ~(1 << CS10);

    // ----- PWM Configuration -----
    // Configure Output Compare Pin: PB1 = pin 15
    DDRB |= (1 << DDB1);
}

```

```

int main(void)
{
    init(); // Initialize

// ----- Main Variables -----
    double t = 0.0;      // [seconds]
    const double f = 0.1;           // Speed of pulse, [Hz]
    const double fsRad = f*2.0*PI;   // Speed of pulse, [rad/sec]

    // Set the Duty Cycle
    double dutyCycle = 0.5; // Default Duty Cycle
    OCR1A = 0x0;

// ----- Main Variables -----
    while (true)
    {
        // Updates PWM (Duty Cycle)
        if ((dutyCycle > 0.0) & (dutyCycle < 1.0)) // Safeguard for [%]
        { OCR1A = static_cast<uint16_t>(dutyCycle * 1023.0); } // [bit]

        // Soft Blink sine
        t += 0.0027; // f_OCnxPWM^(-1)
        dutyCycle = (1.0 - sin(fsRad*t))*0.5; // Sets duty cycle for PWM
    }
    return 0;
}

```

A.4.1.4 Bytting av lysdiode med knapp uten interrupts: Polling

```

/*
 * polling.cpp
 *
 * Created: 17/10/2023 10:43:29
 * Author : mathi
 */

#define F_CPU 1000000 //1MHz, set cpu frequency in Hz, datablad side ...
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>

//questions:
//1. where to write DDRB
//DDRB = 0 = input?

void init()
{
    DDRB |= (1 << DDB1); //set pin B1 as output in data-direction of port D
    DDRD &= ~(1 << PIND6); //set pin D6 of port D as input
    PORTB &= ~(1<<PB1); //set pin B1 low as initial value
    PORTD |= (1<<PD6); //set pin D6 high to enable pull up-resistor to read state
}

```

```

int main(void)
{
    // initialize
    init();

    while (true) // run forever
    {
        if ((( ( PIND & (1 << PD6) ) >> PD6 ) == 0)) //enable only if button is ...
            low/pushed down
        {
            PORTB |= (1<<PB1); //high
        }
        else
        {
            PORTB &= ~(1<<PB1); //low
        }
    }
    return 0;
}

```

A.4.1.5 Pin change interrupt og tilstandsbytting av lysdiode

```

#define F_CPU 1000000 //1MHz, set cpu frequency in Hz, datablad side
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>
#include <avr/interrupt.h> //unused?

namespace intVars //interrupt variables (questions)
{
    volatile int count;
    volatile bool interrupted;
    volatile int timerInterrupt;
}

void init()
{
    DDRD |= (1 << DDD7); //set pin 7 as output in data-direction of port D
    DDRD &= ~(1 << PIND6); //set pin 6 of port D as input
    PORTD &= ~(1<<PD7); //set pin 7 low as initial value
    PORTD |= (1<<PD6); //set pin 6 high to enable pull up-resistor to read state

    // pin change interrupt:
    PCICR |= (1<<PCIE2); // enable pin change interrupt
                           control register PCIE2 for PD6
    PCMSK2 |= (1<<PCINT22); //enable pin change mask register
                           for PCINT22 PD6, not external
    sei();                //enable global interrupt, enable I-bit of SREG

    //namespace initial:
    intVars::count = 0;
    intVars::timerInterrupt = 0;
    intVars::interrupted = false;
}

```

```

ISR(PCINT2_vect)
{
    if ((( ( PIND & (1 << PD6) ) >> PD6 ) == 0) & (intVars::timerInterrupt == 0)) ...
        //enable only if button is low/pushed down
    {
        intVars::interrupted = true;
        intVars::count++;
    }
    else //button is up
    {
        //do nothing
    }
}

int main(void)
{
    //initialize
    init();

    while (true) // run forever
    {
        if (intVars::interrupted)
        {
            intVars::timerInterrupt++;
            _delay_ms(1);
        }

        if (intVars::timerInterrupt >= 150)
        {
            intVars::timerInterrupt = 0;
            intVars::interrupted = false;
        }

        if ((intVars::count % 2) == 1) // if count is odd, turn light on, if ...
            count is even turn light off
        {
            PORTD |= (1<<PD7); //high
        }
        else
        {
            PORTD &= ~(1<<PD7); //low
        }
    }
    return 0;
}

```

A.4.1.6 ADC og variabel PWM settpunkt ved potensiometer

```

/*
 * oppgave8_ADC.cpp
 *
 * Created: 25/10/2023 14:37:49
 * Author : mathi
 */

#ifndef F_CPU 1000000 //8MHZ cpu freq
#include <util/delay.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <math.h>

//questions
//REG = 0b0000 0000 format
//OR:
//REG |= (1<<REG1) | (1<<REG2) .... format
namespace adcVars
{
    volatile bool adcCompleted;
}

void init()
{
    // ----- PWM Configuration -----
    // Fast PWM mode - 10 [bit] (Mode 7 from table 20-6)
    // Waveform Generation Mode
    TCCR1A |= (1 << WGM11) | (1 << WGM10);
    TCCR1B |= (1 << WGM12);
    TCCR1B &= ~(1 << WGM13);

    // Set Fast PWM Mode NON-INVERTING
    // PB1 --> Only COM1An
    TCCR1A &= ~(1 << COM1A0); // COM1A0 = 0
    TCCR1A |= (1 << COM1A1); // COM1A1 = 1

    // Set prescaler:
    // f_Clk = 16 [MHz], f_des = 250 [kHz]
    // Ratio = prescaler = N = 16/0.25 = 64
    TCCR1B |= (1 << CS11);
    TCCR1B &= ~(1 << CS12) & ~(1 << CS10);

    // ----- PWM Configuration -----
    // Configure Output Compare Pin: PB1 = pin 15
    DDRB |= (1 << DDB1);

    //LED:
    //DDRD |= (1 << DDD7); //set pin D7 as output in data-direction of port D
    //PORTD &= ~(1<<PD7); //set pin D7 low as initial value
    //BUTTON:
    DDRD &= ~(1 << PIND6); //set pin 6 of port D as input
    PORTD |= (1<<PD6); //set pin 6 high to enable pull up-resistor to read state
}

```

```

// ----- ADC Configuration -----
//using pin PC3 as ADC input. ADCSRA = 0b 1010 1011;
// bit 7 - enable ADC, bit 6 - start free running mode
// bit 4 - enable auto trigger/ free running mode
// bit 3 - ADC interrupt enabled, edit last 3 bits ...
// later when PWM implemented
ADCSRA |= (1<<ADEN) | (1<<ADATE) | (1<<ADIE) & (~(1<<ADPS2)) | (1<<ADPS1) | ...
(1<<ADPS0);
//ADPS = divided by 8 for 125kHz? //ADCSRA = ADC mode
//ADMUX = 0b 0100 0011;
// bit 7:6-external capacitor on AREF, bit 5- right ...
// adjusted, bit 3:0- using pin PC3
ADMUX |= (1<<REFS0) | (1<<MUX1) | (1<<MUX0); //| (1<<ADLAR); //ADMUX = pins ...
// and AREF selection
adcVars::adcCompleted = false;

//interrupts
sei(); //enable global interrupts, enable I-bit of SREG
}

ISR(ADC_vect) //interrupt for ADC conversion completion
{
    adcVars::adcCompleted = true;
}

int main(void)
{
    init();
    double aVCC = 5.0; //5V reference
    double duty = 0.0;
    double adcVoltage = 0.0;
    uint16_t adc16bit = 0x0;
    //OCRIA = 0x0;
    ADCSRA |= (1<<ADSC); //begin ADC conversion after init() completed

    while(true) //run forever
    {
        if(adcVars::adcCompleted) //alternatively could be ran in the ISR but i ...
// wanted to keep the ISR short
        {
            adc16bit = ADC;

            adcVoltage = (static_cast<double>(adc16bit) / 1024.0) * aVCC; ...
//conversion from 10 bit uint to voltage
            duty = adcVoltage/aVCC; //conversion from voltage to 0-1 percentage ...
// for PWM

            adcVars::adcCompleted = false; // alternatively could read bit 6 of ...
// ADCSRA but might be interfered with by hardware
        }
        else
        {
            //do nothing, wait till next ADC completion
        }

        if((duty > 0.0) & (duty < 1.0))
//if(ADC >= 512)
        {
            OCR1A = static_cast<uint16_t>(duty * 1023.0); //pwm
        }
        else
        {
            //do nothing
        }
    }
}

```

A.4.2 Delprosjekt 2

A.4.2.1 Retur av CAN-melding

```
#include <Arduino.h>
#include <FlexCAN_T4.h>    // FlexCAN Library
#include <SPI.h>
#include <Wire.h>
#include <string.h>

unsigned int meldingerSendt = 0; // unsigned int so overflow goes to 0

// ----- by K. M. Knausgård -----
namespace {
    CAN_message_t msg_tx; // Transmit message

    FlexCAN_T4<CAN0, RX_SIZE_256, TX_SIZE_16> can0;
    //FlexCAN_T4<CAN1, RX_SIZE_256, TX_SIZE_16> can1;
}

// ----- by K. M. Knausgård -----

// canSniff section inspired by tonton81/FlexCAN_T4 on github
// ===== Displaying receive and transmit =====
void canSniff_rx(const CAN_message_t &msg) { // Receive
    Serial.print("- Message received: ");
    Serial.print(" | Mailbox "); Serial.print(msg.mb);
    Serial.print(" | Overrun: "); Serial.print(msg.flags.overrun);
    Serial.print(" | Length: "); Serial.print(msg.len);
    Serial.print(" | ID: "); Serial.print(msg.id, HEX);
    Serial.print(" | Buffer: ");
    for (uint8_t i = 0; i < msg.len; i++) {
        Serial.print(msg.buf[i], HEX); Serial.print(" ");
    }
    Serial.println();
}

msg_tx = msg; // Updates tx = rx
}

void canSniff_tx(const CAN_message_t &msg) { // Transmit
    Serial.print("- Message transmitted:");
    Serial.print(" | Mailbox "); Serial.print(msg.mb);
    Serial.print(" | Overrun: "); Serial.print(msg.flags.overrun);
    Serial.print(" | Length: "); Serial.print(msg.len);
    Serial.print(" | ID: "); Serial.print(msg.id, HEX);
    Serial.print(" | Buffer: ");
    for (uint8_t i = 0; i < msg.len; i++) {
        Serial.print(msg.buf[i], HEX); Serial.print(" ");
    }
    Serial.println();
}

// ===== Displaying receive and transmit =====
```

```

void setup() {
    Serial.begin(9600);
    can0.begin();
    can0.setBaudRate(250000);
    //can1.begin();
    //can1.setBaudRate(250000);
    can0.enableFIFO();
    can0.enableFIFOInterrupt();

    can0.onReceive(canSniff_rx); // Received on Teensy from PCAN
    can0.onTransmit(canSniff_tx); // Transmitted from Teensy to PCAN
}

void loop() {
    delay(2000); // 2 [seconds]

    can0.write(msg_tx); // returns message from Teensy to PCAN

    meldingerSendt = meldingerSendt + 1;
    Serial.print("\nLoop number: ");
    Serial.println(meldingerSendt);
}

```

A.4.2.2 OLED-skjerm og CAN

```

#include <Arduino.h>
#include <FlexCAN_T4.h>
#include <SPI.h>
#include <Wire.h>
#include <string.h>

#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define OLED_DC 6 // Definerer LCD pins fra kortskjematikk - fra OLED test demo ...
               // software
#define OLED_CS 10
#define OLED_MOSI 11
#define OLED_RESET 5
#define OLED_HEIGHT 64
#define OLED_WIDTH 128

Adafruit_SSD1306 display(OLED_WIDTH, OLED_HEIGHT, &SPI, OLED_DC, OLED_RESET, ...
                         OLED_CS); // Instantierer skjerm

int meldingerSendt = 0;
int canAntallCursorX = 0;
int canAntallCursorY = 0;
int canIdCursorX = 0;
int canIdCursorY = 0;
int meldingerMottatt = 0;
String meldingId;

namespace {
    CAN_message_t msg;

    FlexCAN_T4<CAN0, RX_SIZE_256, TX_SIZE_16> can0;
}

void canSniff(const CAN_message_t &msg);

```

```

void setup() {
    Serial.begin(9600);
    can0.begin();
    can0.setBaudRate(250000);

    display.begin(SSD1306_SWITCHCAPVCC); //Generate 3.3v
    display.clearDisplay();

    display.setTextSize(0);
    display.setTextColor(WHITE);

    display.drawRoundRect(0, 0, OLED_WIDTH, OLED_HEIGHT, 5, WHITE); //Tegner rektangel
    display.display(); //Sender endringer i RAM til skjerm

    //Print gruppenavn på midten av skjermen, med pause for dramatikk
    delay(1000); //Dramatisk pause som om dette programmet er tungt og krever ...
                  innlasting
    int16_t x1, y1;
    uint16_t w, h;
    display.getTextBounds("MAS245 - Gruppe 5", 0, 0, &x1, &y1, &w, &h); //Finner ...
                  lengden på tittel og sender til w og h
    display.setCursor(OLED_WIDTH/2 - (w/2), 2); //Setter cursor midt på skjermen ...
                  minus tekstlengde offset. 2 pixler ned
    display.print("MAS245 - Gruppe 5");
    display.display();

    delay(500);

    for(int i = 1; i < 129; i++) // Inspirert av K. M. Knausgård
    {
        display.drawLine((i-1),(5*std::sin((i-1)*(3.1415/128))+9), ...
                      i,(5*std::sin((i-1)*(3.1415/128))+9), WHITE);
        display.display();
        delay(10);
    }
    //display.drawCircle(OLED_WIDTH/2,-500,513,WHITE);
    // ^ Tegne sirkel som går ned 13 piksler fra toppen.

    delay(500);
    display.setCursor(0,15);
    display.println(" CAN-statistikk");
    delay(200);
    display.display();
    display.println(" -----");
    delay(200);
    display.display();
    display.print(" Antall mottatt: ");
    canAntallCursorX = display.getCursorPosition(); //Husker cursorkoordinat for å...
                                                printe tall her senere
    canAntallCursorY = display.getCursorY();
    display.println("0");
    display.display();
    delay(200);
    display.print(" Mottok sist ID: ");
    canIdCursorX = display.getCursorPosition(); //Husker cursorkoordinat for å...
                                                printe tall her senere
    canIdCursorY = display.getCursorY();
    display.println("0");
    display.display();
    delay(200);
    display.println(" -----");
    display.display();
    delay(200);
}

```

```

display.print(" IMU-M");
display.write(0x86); //Nesten "å"
display.println("ling Z: TBA");

display.display();

delay(1000);

can0.enableFIFO(); //Disse må komme _etter_ tegningen, hvis ikke resetter ...
    koordinatene til cursor seg og krøller til alt
can0.enableFIFOInterrupt();
can0.onReceive(canSniff); //Kjører canSniff ved mottak av CANbus meldinger
}

void loop() {
//Venter bare på CAN-meldinger
}

void canSniff(const CAN_message_t &msg) { //Kjører når en melding blir mottatt

meldingerMottatt = meldingerMottatt +1;
meldingId = msg.id;
display.setCursor(canAntallCursorX, canAntallCursorY);
display.fillRect(canAntallCursorX, canAntallCursorY, 20, 7, BLACK); //Sletter ...
    hva enn som stod her tidligere
display.setCursor(canAntallCursorX, canAntallCursorY);
display.print(meldingerMottatt);

display.setCursor(canIdCursorX, canIdCursorY);
display.fillRect(canIdCursorX, canIdCursorY, 20, 7, BLACK);
display.setCursor(canIdCursorX, canIdCursorY);
display.print(msg.id, HEX);

display.display(); //Sender alle endringer til skjermen
}

```

A.4.2.3 Pong!

```

#include <Arduino.h>
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <FlexCAN_T4.h>
#include <math.h>

#define OLED_RESET 5
#define OLED_DC 6
#define OLED_CS 10
#define OLED_WIDTH 128
#define OLED_HEIGHT 64

#define JOY_UP 22
#define JOY_DOWN 23
#define JOY_PRESS 19

#define ballRadius 2

```

```

#define padWidth 4.0      // Pads har lokal origo i top venstre hjørne
#define padHeight 20.0
#define scoreBoxH 11
#define scoreBoxW 35
#define scoreBoxRadius 3

#define periode 10          // Bruker lavere frekvens ved testing. frekvens = ...
    periode^-1 [Hz]
#define winScore 10

// ----- Globale variabler -----
// Ball
float xFartBall = 0.0;
float yFartBall = 0.0;
float xPos = OLED_WIDTH/2.0;           // startPosisjon ball
float yPos = OLED_HEIGHT/2.0;
int xPosNy = 0;
int yPosNy = 0;

// Paddles
int padX = OLED_WIDTH - padWidth; // Start pos høyre pad (lokal)
float padY = (OLED_HEIGHT - padHeight)/2;
int pad2X = 0;                      // Start pos venstre pad
float pad2Y = (OLED_HEIGHT - padHeight)/2; // Lokal oppdatering
float pad2Yoppdatert = pad2Y;        // Fra CANbus ~ starter samme, ...
    endres senere
float xPosSlave = (OLED_WIDTH-1) - OLED_HEIGHT;
float yPosSlave = OLED_HEIGHT/2 - 1;
float padFart = 0;                  // For JoyStick (pass-by-copy)

float padMid; // Definert i sjekkBallsPosisjonOgSprett()
float diffTheta;
float ballMagnitude = 1.0; // Lengden av fartvektoren.

// Misc
bool masterState = false;
int startSpill = 0; // 0 = start Screen, 1 = game on, 2 = game finished
int16_t X, Y;
uint16_t W, H;
int16_t cursorX = OLED_WIDTH/2;           // Middle
int16_t cursorY = 0;                      // Top
int16_t scoreCursorX = cursorX - scoreBoxW/2;
int hostScore = 9;
int slaveScore = 9;
float t = 0.0;
// ----- Globale variabler -----


Adafruit_SSD1306 display(OLED_DC, OLED_RESET, OLED_CS);
FlexCAN_T4<CAN0, RX_SIZE_256, TX_SIZE_16> can0; // Starter can0
CAN_message_t ballUt;
CAN_message_t paddle1Pos;
CAN_message_t masterEllerSlave; //buf[0] = 1 betyr start spill ,
                                buf[1] = 1 betyr at har master state
CAN_message_t score;

// Hvis HOST
void bevegBallHost(float &xPos, float &yPos, float xFartBall, float yFartBall){
    //display.fillCircle(xPos, yPos, ballRadius, BLACK); // Sletter gammel ball
    xPos = xPos + xFartBall;
    yPos = yPos + yFartBall;
    display.fillCircle(xPos, yPos, ballRadius, WHITE); // Tegner ny ball
}

```

```

void bevegBallSlave( float &xPos, float &yPos){ // Speilvender ballposisjon
    //display.fillCircle ((OLED_WIDTH-1)-xPosSlave, yPosSlave, ballRadius, BLACK);
    display.fillCircle ((OLED_WIDTH-1)-xPosNy, yPosNy, ballRadius, WHITE);
    xPosSlave = xPosNy;
    yPosSlave = yPosNy;
}

void bevegPad1( float &padY){ // Sjekker input fra joystick , Opp ned ...
    koordinatsystem
    if (!digitalRead(JOY_UP) && padY != 0){ // Joystick opp aktiv?
        padFart = -1;
    }
    // Joystick ned aktiv?
    if (!digitalRead(JOY_DOWN) && padY != (OLED_HEIGHT- padHeight)){ // Tegn ny pad
        padFart = 1;
    }

    //display.fillRect (padX, padY, padWidth, padHeight, BLACK);
    // Fjern gammel pad
    display.fillRect (padX, padY+padFart, padWidth, padHeight, WHITE); // Tegn ny pad
    padY = padY + padFart; // Oppdater neste posisjon
    padFart = 0; // Reset input sjekk
}

void bevegPad2( float &pad2Y) {
    //display.fillRect (pad2X, pad2Y, padWidth, padHeight, BLACK); ...
    //Sletter gamle pad2
    display.fillRect (pad2X, pad2Yoppdatert, padWidth, padHeight, WHITE); //Tegner ...
    ny pad2 fra canbus data
    pad2Y = pad2Yoppdatert;
}

void sprettY( float &yFartBall){
    yFartBall = -1 * yFartBall;
}

void sprettHorisontalt( float &xFartBall, float &yFartBall){ // Skalerer ...
    vektningen i x og y ...
    retning
    xFartBall = - ballMagnitude * cos(diffTheta);
    yFartBall = ballMagnitude * sin(diffTheta);
}

void canSniff( const CAN_message_t &msg){ // KUN for receiving
    if (msg.id == 50){ // Ballposisjon
        xPosNy = msg.buf[0];
        yPosNy = msg.buf[1];
    }
    else if (msg.id == 30){ // Gruppenummer + 10 - mottar paddleposisjon ...
        (motstanders paddle 1)
        pad2Yoppdatert = msg.buf[0]; // Oversetter fra hexa automatisk
    }
    else if (msg.id == 15){
        startSpill = msg.buf[0];
        masterState = msg.buf[1];
    }
    else if (msg.id == 10){ // Score
        hostScore = msg.buf[0];
        slaveScore = msg.buf[1];
    }
}

```

```

void sendBallPos() {
    //Host funksjon
    ballUt.id = 50;           //Gruppenummer + 50
    ballUt.len = 2;
    ballUt.buf[0] = xPos;
    ballUt.buf[1] = yPos;
    can0.write(ballUt);
}

void sendPaddle1pos() {
    paddle1Pos.id = 30;        //Gruppenummer+20
    paddle1Pos.len = 1;
    paddle1Pos.buf[0] = padY;
    can0.write(paddle1Pos);
}

void sendScore(int hostScore, int slaveScore) {
    //Host funksjon
    score.id = 10;
    score.len = 2;
    score.buf[0] = slaveScore; // Invertert til motstander
    score.buf[1] = hostScore;
    can0.write(score);
}

void sjekkBallsPosisjonOgSprett(){
//HOST FUNKSJON
    if ( (yPos > ((OLED_HEIGHT-1) - ballRadius)) || (yPos < (0+ballRadius)) ) // ...
        Upper & Lower bounds
    {
        sprettY(yFartBall); //Snur y hastighet
    }
    if ((xPos < (padWidth+ballRadius)) && // Sjekker om ball treffer ...
        ( (yPos < (pad2Y+padHeight+ballRadius)) && (yPos > (pad2Y-ballRadius)))) {
        padMid = pad2Y + padHeight/2;
        diffTheta = (yPos - padMid)*(PI/3)/(padHeight/2+ballRadius);
        // Differanse [radianer] fra -pi/3 til +pi/3
        sprettHorisontalt(xFartBall, yFartBall);
        xFartBall *= -1; // Gå til høyre, ikke venstre
    }
    if ((xPos > (padX-ballRadius)) &&
        ((yPos < (padY+padHeight+ballRadius)) && (yPos > (padY-ballRadius)))) // ...
        Sjekker om ball treffer paddle1 (høyre)
    {
        padMid = padY + padHeight/2; // Nullpunkt til pad
        diffTheta = (yPos - padMid)*(PI/3)/(padHeight/2+ballRadius);
        // Differanse [radianer] fra -pi/3 til +pi/3
        sprettHorisontalt(xFartBall, yFartBall); // Trig bouncer
    }
    // -----
    // Serial.print(" | padMid: "); Serial.print(padMid);
    // Serial.print(" | diffTheta: "); Serial.print(diffTheta);
    Serial.print(" | pad1 (top): "); Serial.print(padY);
    Serial.print(" | pad2 (top): "); Serial.print(pad2Y);
    Serial.print(" | Ball x: "); Serial.print(xPos);
    Serial.print(" | Ball y: "); Serial.println(yPos);
    // Serial.print(" | xFartBall: "); Serial.print(xFartBall);
    // Serial.print(" | yFartBall: "); Serial.println(yFartBall);
    // -----
}

```

```

void resetGame(){
    display.clearDisplay();
    xPos = OLED_WIDTH/2.0;           //startPosisjon ball
    yPos = OLED_HEIGHT/2.0;
    padY = (OLED_HEIGHT - padHeight)/2;
    pad2Y = (OLED_HEIGHT - padHeight)/2; // Lokal oppdatering
    pad2Yoppdatert = pad2Y;
}

void gameOver(){
    //slaveScore = (xPos < 0) ? slaveScore++ : slaveScore;
    if ((xPos < 0)){
        hostScore++;
        delay(500);
        resetGame();
        xFartBall = 0.5;
        yFartBall = 0.5;
    }
    else if (xPos > OLED_WIDTH){
        slaveScore++;
        delay(500);
        resetGame();
        xFartBall = -0.5;
        yFartBall = 0.5;
    }
}

void setup() {
    can0.begin();
    can0.setBaudRate(250000);
    Serial.begin(9600);           //Brukes bare til funksjonstesting
    display.begin(SSD1306_SWITCHCAPVCC); //Gir 3.3V til skjerm
    display.clearDisplay();
    display.setTextSize(0);
    display.setTextColor(WHITE);
    //Tegn startskjem TBA
    display.display();
    pinMode(JOY_DOWN, INPUT);      //Setter joystick pins til å være input
    pinMode(JOY_UP, INPUT);
    pinMode(JOY_PRESS, INPUT);

    can0.enableFIFO();
    can0.enableFIFOInterrupt();
    can0.onReceive(canSniff);
}

```

```

void loop() {
    // ===== START SCREEN =====
    while (startSpill == 0) {
        // Tittel
        display.setTextSize(2);
        display.getTextBounds("PONG!", cursorX, cursorY, &X, &Y, &W, &H);
        display.setCursor((cursorX-W/2), (cursorY+2));
        display.print("PONG!");
        // Svevende Ball
        yFartBall = sin(2*PI*t);
        //xFartBall = cos(t);
        display.fillCircle(xPos, yPos, (ballRadius+1), BLACK); // Sletter gammel ball
        yPos = yPos + yFartBall;
        //xPos = xPos + xFartBall;
        display.fillCircle(xPos, yPos, (ballRadius+1), WHITE); // Tegner ny ball

        // tekst
        display.setTextSize(0);
        display.getTextBounds("START: Trykk joystick", cursorX, cursorY, &X, &Y, &W, ... &H);
        display.setCursor((cursorX-W/2), (OLED_HEIGHT - 10));
        display.print("START: Trykk joystick");

        display.setCursor(1, 1);
        display.print("Gr.5");
        display.getTextBounds("2023", cursorX, cursorY, &X, &Y, &W, &H);
        display.setCursor((OLED_WIDTH-W), 1);
        display.print("2023");

        //display.getTextBounds("2023", cursorX, cursorY, &X, &Y, &W, &H);
        //display.setCursor((OLED_WIDTH-W-1), (OLED_HEIGHT-1));
        //display.print("2023");

        can0.disableFIFOInterrupt(); //Stopper interrupts når skjermen tegnes
        display.display(); //Tegner hele bildet etter alle ...
                           kalkulasjoner har blitt gjort

        can0.enableFIFOInterrupt();
        delay(periode);
        t += 0.05;

        if (!digitalRead(JOY_PRESS)) { // Hvis joystick presses
            startSpill = 1;
            masterState = true;
            masterEllerSlave.id = 15;
            masterEllerSlave.len = 2;
            masterEllerSlave.buf[0] = 1; // Starter spill - buf[0] leser til ...
                                         // startSpill variabel
            masterEllerSlave.buf[1] = 0; // Setter lokal maskin til master - motsatt ...
                                         // maskin leser 0 og kjører slavesløyfe
            can0.write(masterEllerSlave);
            xFartBall = 0.5;
            yFartBall = 0.5;
            //display.display();
        }
    }
    // ===== START SCREEN =====
}

```

```

// ===== GAME SCREEN =====
while(startSpill == 1) { // Game on
    display.clearDisplay();
    bevegPad1(padY); // Lokal pad bevegelse (høyre pad)
    bevegPad2(pad2Y); // Henter data fra motstander og beveger (venstre pad)
    sendPaddle1pos();

    if (masterState) {
        //Hvis master
        sendBallPos();
        bevegBallHost(xPos, yPos, xFartBall, yFartBall);
        sjekkBallsPosisjonOgSprett();
        gameOver();
        sendScore(hostScore, slaveScore);
    }
    else if (!masterState){
        //Hvis slave
        bevegBallSlave(xPos, yPos);
    }

    // ----- Score display -----
    display.drawRoundRect(scoreCursorX, cursorY, scoreBoxW, scoreBoxH, ...
        scoreBoxRadius, WHITE); // Hvit boks
    display.drawRoundRect((scoreCursorX+1), (cursorY+1), scoreBoxW-2, ...
        scoreBoxH-2, scoreBoxRadius, BLACK); // Svart boks

    display.setCursor((scoreCursorX+3), (cursorY+2));
    display.print(slaveScore);
    display.print(" - ");
    display.print(hostScore);
    //display.setCursor((OLED_WIDTH/2-3), 0); // Middle minus 5 pixels (width of ...
    //"-")
    //display.print("-");
    // ----- Score display -----
}

can0.disableFIFOInterrupt(); //Stopper interrupts når skjermen tegnes
display.display(); //Tegner hele bildet etter alle ...
                           kalkulasjoner har blitt gjort
can0.enableFIFOInterrupt();
delay(periode);

// startSpill = ((slaveScore==winScore)|| (hostScore==winScore)) ? 2 : 1; // ...
                           Game finished?
if ((slaveScore==winScore)|| (hostScore==winScore)){
    startSpill = 2;
    xPos = OLED_WIDTH/2;
    yPos = OLED_HEIGHT-(ballRadius+8);
}
}

// ===== GAME SCREEN =====

```

```

// ===== GAME FINISHED =====
while (startSpill == 2){
    display.clearDisplay();

    if (slaveScore == 10){
        display.setTextSize(1);
        display.getTextBounds("Motstanderen vant!", cursorX, cursorY, &X, &Y, &W, &H);
        display.setCursor((cursorX-W/2), (OLED_HEIGHT/4));
        display.print("Motstanderen vant!");
        display.setTextSize(0);
        display.getTextBounds("Du suger.", cursorX, cursorY, &X, &Y, &W, &H);
        display.setCursor((cursorX-W/2), (OLED_HEIGHT/4 + 10));
        display.print("Du suger.");
        startSpill = 2;
    }
    else if (hostScore == 10){
        display.setTextSize(1);
        display.getTextBounds("Du vant!", cursorX, cursorY, &X, &Y, &W, &H);
        display.setCursor((cursorX-W/2), (OLED_HEIGHT/4));
        display.print("Du vant!");
        display.setTextSize(0);
        display.getTextBounds("Motstanderen suger.", cursorX, cursorY, &X, &Y, &W, ...
            &H);
        display.setCursor((cursorX-W/2), (OLED_HEIGHT/4 + 10));
        display.print("Motstanderen suger.");
        startSpill = 2;
    }

    // Svevende Ball
    yFartBall = sin(PI*t);
    xFartBall = cos(0.5*t-PI/4);
    //display.fillCircle(xPos, yPos, (ballRadius+1), BLACK); // Sletter gammel ball
    yPos = yPos + yFartBall;
    xPos = xPos + xFartBall;
    display.fillCircle(xPos, yPos, (ballRadius+1), WHITE); // Tegner ny ball

    can0.disableFIFOInterrupt(); // Stopper interrups når skjermen tegnes
    display.display(); // Tegner hele bildet etter alle ...
                        // kalkulasjoner har blitt gjort

    can0.enableFIFOInterrupt();
    delay(periode);
    t += 0.1;
}
// ===== GAME FINISHED =====
// Serial.println(__TIMESTAMP__);
}

```

A.5 MATLAB kode

A.5.1 Delprosjekt 1

A.5.1.1 Soft Blink Simulering i MATLAB

```
clc; close all; clear;

%% Simulering av sinusbølge til soft blink

syms t;
fs = 0.2; % Frekvens [Hz]
fsRad = fs*2*pi; % Frekvens [rad/sek]
f(t) = (1 - sin(t*fsRad))*0.5;

figure
fplot(f*100)
xlim([0, 10])
lgd = legend('$\frac{1-\sin(fsRad \cdot t)}{2}$', 'Location', ...
'southeast', 'Interpreter', 'latex');
fontsize(lgd, 16, 'points')
xlabel('Tid [sekunder]')
ylabel('PWM verdi [%]')
title('Soft Blink')
```

A.5.2 Delprosjekt 3

A.5.2.1 Teensy3.6 IMU over CAN-bus kode

```
#include <Arduino.h>
#include <FlexCAN_T4.h>
#include <SPI.h>
#include <Wire.h>
#include <MPU6050.h>
#include "I2Cdev.h" //I2C bus
#include <string.h>
#include <stdlib.h>
#include <string>
#include <Adafruit_BusIO_Register.h> //MPU6050 dependency

//MPU6050 code from
https://github.com/ElectronicCats/mpu6050/tree/master

MPU6050 mpu; //Create IMU object
CAN_message_t msgInn;
CAN_message_t msgUt;
CAN_message_t sensorOut;

//https://github.com/tonton81/FlexCAN\_T4/blob/master/README.md
FlexCAN_T4<CAN0, RX_SIZE_256, TX_SIZE_16> can0;

void canSniff(const CAN_message_t &msgInn) { //Do operations and ...
    return CAN message
Serial.println("");
Serial.print("Received: ");

char hexToString[7] = {0}; //For printing hex data - size = max ...
    message length + 1 to avoid array termination errors
for (int i = 0; i < msgInn.len; i++) {
hexToString[i] = msgInn.buf[i];
Serial.print(msgInn.buf[i], HEX);
Serial.print("");
}
Serial.println("");
Serial.print("Received from Raspberry: ");
Serial.print(hexToString);
Serial.println("");
Serial.println("");

}

void setup() {
can0.begin();
Serial.begin(9600);
can0.setBaudRate(250000);
can0.enableFIFO();
can0.enableFIFOInterrupt();
```

```

can0.onReceive(canSniff);
Wire.begin(); //Start I2C bus
mpu.initialize(); //After wire.begin() ; or crash program.

Serial.println(mpu.testConnection() ? "MPU6050 connection ... "
    "successful" : "MPU6050 connection failed");

}

void loop() {

float num = mpu.getAccelerationZ() * (9.81 / 16384); //Signed 15 bit ...
    output , adjusted to display m/s^2
String floatString = String(num); //Convert float to string

delay(3000);
sensorOut.id = 13;
sensorOut.len = floatString.length();

for (uint i = 0; i < sensorOut.len; i++) { //Load outbound CAN frame ...
    with float digits
    sensorOut.buf[i] = floatString[i];
}

can0.write(sensorOut);
Serial.println("");
Serial.print("Sent to Raspberry: ");
Serial.print(num);
}

```

A.5.2.2 Raspberry Pi kode

```

// K. M. Knausgård / MAS234 2017
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <net/if.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <linux/can.h>
#include <linux/can/raw.h>
#include <string>
#include <iostream>
#include <time.h>

struct can_frame_edit { //Can.h default frame returns compile error ...
    when using .len. Custom frame circumvents problem
    canid_t can_id; /* 32 bit CAN_ID + EFF/RTR/ERR flags */
    __u8 len;
    __u8 __pad; /* padding */
    __u8 __res0; /* reserved / padding */

```

```

    __u8 len8_dlc; /* optional DLC for 8 byte payload length (9 ...  

                     15) */  

    __u8 data[CAN_MAX_DLEN] __attribute__(( aligned(8)));
};

struct can_frame_edit sendFrame; //Outbound frame  

struct can_frame_edit receiveFrame; //Inbound frame

// Can bus configuration.  

const char *ifname = "can0";  

int canSocketDescriptor;

// Task rate control.  

//const double periodicTaskRate = 2.0; // Run task at 0.5 Hz.  

const int64_t periodicTaskDtNs = static_cast<int64_t>((1.0/0.5) * ...  

1.0e9);

bool createCanSocket(int& socketDescriptor);  

void sendCanMessage(int socketDescriptor);

void myPeriodicTask() //Reads can message from buffer , multiplies ...  

    data by two, returns CAN message
{

    //Read function from https://docs.kernel.org/networking/can.html
    int nbytes = read(canSocketDescriptor, &receiveFrame, ...  

        sizeof(struct can_frame_edit)); //Read CAN from can buffer.  

    int meldingLengde = receiveFrame.len;

    std::cout << "Received from teensy: "; //Print received data

    for (int j = 0; j < meldingLengde; j++) { //Unpack CAN message into  

        std::cout << receiveFrame.data[j];
    }
    std::cout << std::endl;

    std::string hexTilString;

    for (int j = 0; j < meldingLengde; j++) { //Transport CAN data ...  

        bytes to string array
        hexTilString[j] = receiveFrame.data[j];
    }

    std::cout << "Message length: " << meldingLengde << std::endl;

    float tallFraImu = std::stof(hexTilString); //Convert string to ...
        float

    if (tallFraImu < 10.0 && tallFraImu > 4.99) {
}

```

```

meldingLengde = 5; //Sets message length to 5 if operation ...
    increases number of digits (Example: 9.04 * 2 = 18.08, length ...
    increased to 5 from 4)
} //Missing logic for negative values - should be ...
    patched in.

tallFraImu = tallFraImu*2.0; //Arbitrary operation on CAN ...
    message data

std::cout << "Data after operation: " << tallFraImu; //Confirm ...
    operation
std::cout << std::endl;

std::string sendTilTeensy = std::to_string(tallFraImu); //Float ...
    to string conversion

sendFrame.can_id = 13;
sendFrame.len = meldingLengde; //Define outbound message length

//Load operated value into return CAN message
for (int j = 0; j < meldingLengde; j++) {
    sendFrame.data[j] = sendTilTeensy[j];
}

std::cout << "Sent to teensy: "; //Print sent data
for (int i = 0; i < meldingLengde; i++) {
    std::cout << sendFrame.data[i];

}
std::cout << std::endl;

static uint64_t ii(0U);
++ii;
std::cout << "Tick " << ii << " .. " << std::endl;

//Send CAN message
const int bytesWritten = write(canSocketDescriptor, &sendFrame, ...
    sizeof(struct can_frame_edit));
}

int main()
{
    std::cout << "Periodic tick example for MAS234 using ...
        clock_nanosleep.." << std::endl;

    // Set up CAN.
    if (!createCanSocket(canSocketDescriptor)) // Passed by reference; ...
        canSocketDescriptor is the output variable!
    {

```

```

    std::cout << "Could not create CAN socket" << std::endl;
    return 0;
}

bool running = true;
while(running)
{
    // USE CLOCK_MONOTONIC, and NOT NOT NOT CLOCK_REALTIME.
    // "Realtime" sounds tempting but is non-monotonic and actually a
    // wall-time-realtime, not usable for real time tasks.
    // For hard real time systems, use a RTOS or at least linux with ...
    // real time extensions.
    // Note: This example has NO error checking (not good).

    // 1. Get current time stamp.
    struct timespec nextTimerDeadline;
    clock_gettime(CLOCK_MONOTONIC, &nextTimerDeadline);

    // 2. Run the periodic task.
    myPeriodicTask();

    // Add number of nanoseconds the task needs to sleep to the ...
    // initial timestamp.
    // Could add logic here to check if the task used too much time.
    const int64_t nextTvNsec = ...
        static_cast<int64_t>(nextTimerDeadline.tv_nsec) + ...
        periodicTaskDtNs;
    std::cout << " " << nextTvNsec << " , " << ...
        nextTimerDeadline.tv_nsec << std::endl << std::endl;

    if (nextTvNsec >= 1000000000L)
    {
        // The timespec struct has one part for nanoseconds and one ...
        // for seconds,
        // nsec part must be less than one second.
        nextTimerDeadline.tv_sec += static_cast<long int>(nextTvNsec ...
            / 1000000000L);
        nextTimerDeadline.tv_nsec = static_cast<long int>(nextTvNsec ...
            % 1000000000L);
    }

    // 3. Sleep until next deadline.
    // See ...
    // http://man7.org/linux/man-pages/man2/clock_nanosleep.2.html ...
    // for documentation.
    clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, ...
        &nextTimerDeadline, NULL);
}

return 0;
}

```

```

bool createCanSocket(int& socketDescriptor)
{
    struct ifreq ifr;

    if ((socketDescriptor = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
        perror("Error while opening CAN socket.");
        return false;
    }

    strcpy(ifr.ifr_name, ifname);
    ioctl(socketDescriptor, SIOCGIFINDEX, &ifr);

    struct sockaddr_can addr;
    addr.can_family = AF_CAN;
    addr.can_ifindex = ifr.ifr_ifindex;

    if (bind(socketDescriptor, (struct sockaddr *)&addr, sizeof(addr)) < ...
        0) {
        perror("Error in socketcan bind.");
        return false;
    }

    return true;
}

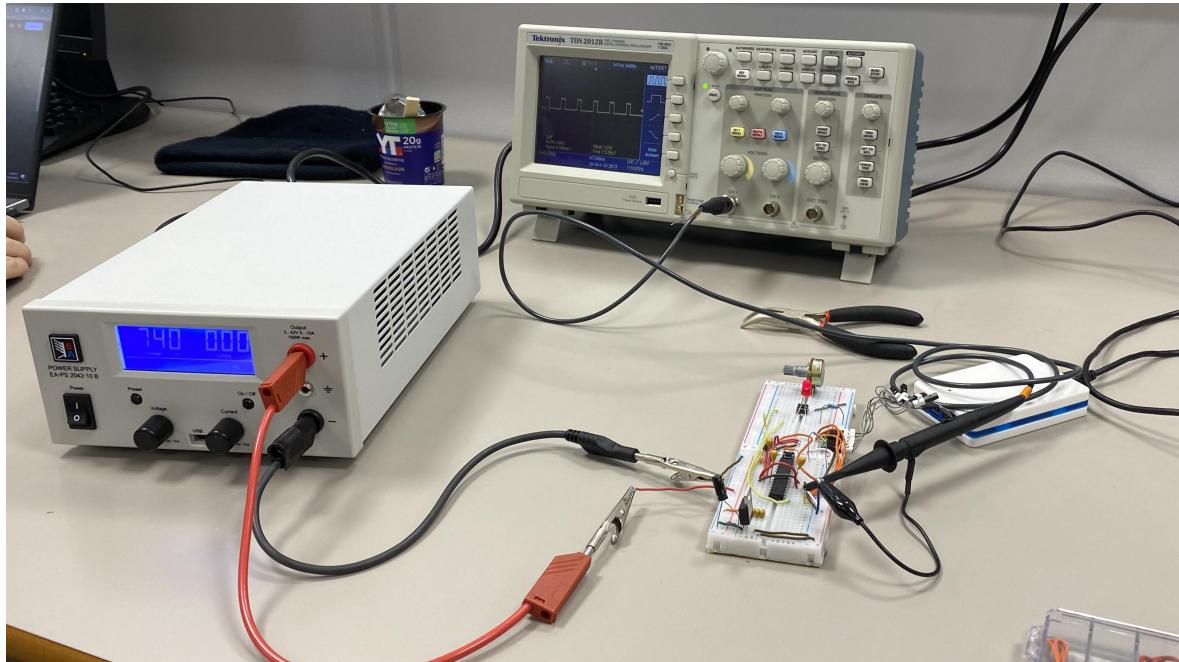
void sendCanMessage(int socketDescriptor)
{
    // The frame should probably be passed to this function somehow..
    sendFrame.can_id = 0x234;
    //sendFrame.can_dlc = 2;
    //sendFrame.len = 6;

    const int bytesWritten = write(socketDescriptor, &sendFrame, ...
        sizeof(struct can_frame));
    std::cout << "sendte can fra modifisert program" << std::endl;
    // Should handle errors here...
}

```

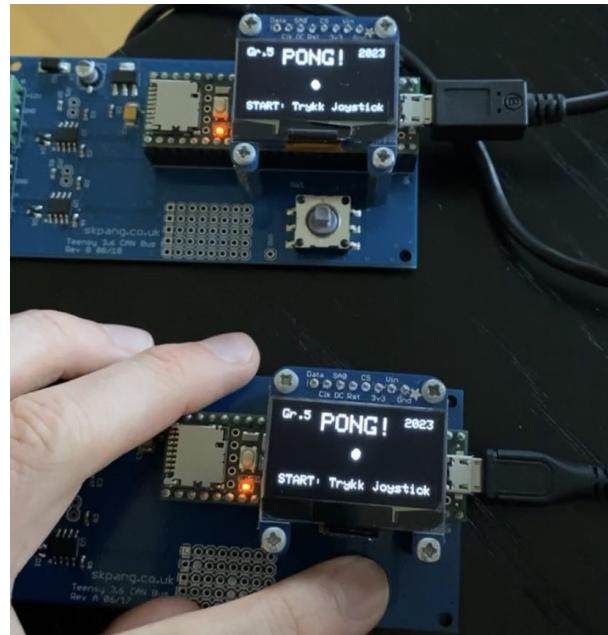
A.6 Fysisk oppkobling

A.6.1 Delprosjekt 1



Figur A.5: Oppkobling av mikrokontrollerkrets

A.6.2 Delprosjekt 2



Figur A.6: Oppkobling av to Teensy 3.6 med CAN bus

Bibliografi

- [1] Jeff Atwood. *Understanding User and Kernel Mode*. URL: <https://blog.codinghorror.com/understanding-user-and-kernel-mode/>. (hentet: 23.11.2023).
- [2] Antonio Brewer. *FlexCAN_T4*. URL: https://github.com/tonton81/FlexCAN_T4. (hentet: 25.10.2023).
- [3] Phillip Burgess. *Adafruit GFX Graphics Library*. URL: <https://learn.adafruit.com/adafruit-gfx-graphics-library>. (hentet: 27.10.2023).
- [4] The kernel development community. *SocketCAN - Controller Area Network*. URL: <https://docs.kernel.org/networking/can.html>. (hentet: 21.11.2023).
- [5] RND Components. *Aluminum Electrolytic Capacitors*. URL: https://media.distrelec.com/Web/Downloads/_t/ds/RND%20150KSK_eng_tds.pdf. (hentet: 20.10.2023).
- [6] Atmel Corporation. *8-bit AVR Microcontroller ATmega48/V / 88/V / 168/V DATASHEET COMPLETE*. URL: https://no.mouser.com/datasheet/2/268/Atmel_2545_8_bit_AVR_Microcontroller_ATmega48_88_1-1315326.pdf. (hentet: 18.10.2023).
- [7] Atmel Corporation. *Atmel-ICE USER GUIDE*. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42330-Atmel-ICE_UserGuide.pdf. (hentet: 11.10.2023).
- [8] Atmel Corporation. *AVR Microcontroller Hardware Design Considerations*. URL: <https://ww1.microchip.com/downloads/en/Appnotes/AN2519-AVR-Microcontroller-Hardware-Design-Considerations-00002519B.pdf>. (hentet: 18.10.2023).
- [9] Atmel Corporation. *AVR040: EMC Design Considerations*. URL: https://ww1.microchip.com/downloads/en/appnotes/atmel-1619-emc-design-considerations_applicationnote_avr040.pdf. (hentet: 17.10.2023).
- [10] ElectronicCats. *MPU6050*. URL: <https://github.com/ElectronicCats/mpu6050>. (hentet: 21.11.2023).
- [11] Bondline Electronics. *What Is Electrostatic Discharge (E.S.D.)?* URL: <https://www.bondline.co.uk/blog/what-is-electrostatic-discharge-e-s-d#:~:text=Why%20Is%20Managing%20ESD%20Important,is%202%2C000%20volts%20or%20higher..> (hentet: 22.11.2023).
- [12] SK Pang Electronics. *Teensy 3.6 Dual CAN-Bus Breakout Board Include Teensy 3.6*. URL: https://www.skpang.co.uk/products/teensy-3-6-dual-can-bus-breakout-board-include-teensy-3-6?_pos=3&_sid=94fb41181&_ss=r. (hentet: 27.10.2023).
- [13] ROBERT BOSCH GmbH. *CAN Specification Version 2.0*. URL: <http://esd.cs.ucr.edu/webres/can20.pdf>. (hentet: 23.11.2023).
- [14] Texas Instruments. *LM2940x 1-A Low Dropout Regulator*. URL: https://www.ti.com/lit/ds/symlink/lm2940-n.pdf?HQS=dis-mous-null-mousermode-dsf-pf-null-wwe&ts=1697004252542&ref_url=https%253A%252F%252Fwww.mouser.es%252F. (hentet: 20.10.2023).
- [15] Thomas J. Fellers og Michael W. Davidson Kenneth R. Spring. *Human Vision and Color Perception*. URL: [https://www.olympus-lifescience.com/en/microscope-resource/primer/lightandcolor/humanvisionintro/#:~:text=The%20human%20visual%20system%20response,range\)%20of%20over%2010%20decades..](https://www.olympus-lifescience.com/en/microscope-resource/primer/lightandcolor/humanvisionintro/#:~:text=The%20human%20visual%20system%20response,range)%20of%20over%2010%20decades..) (hentet: 25.10.2023).
- [16] Kingbright. *T-1 3/4 (5mm) SOLID STATE LAMP*. URL: [https://www.kingbright.com/attachments/file/psearch/000/00/watermark00/L-7113SRD-D\(Ver.15A\).pdf](https://www.kingbright.com/attachments/file/psearch/000/00/watermark00/L-7113SRD-D(Ver.15A).pdf). (hentet: 20.10.2023).
- [17] Kristian Muri Knausgård. *Forelesning 16: Operativsystemer*. URL: https://uia.instructure.com/courses/13944/files/2260907?module_item_id=512645. (hentet: 23.11.2023).

- [18] Kristian Muri Knausgård. *Forelesning 17: Operativsystemer II. Systemkall, privilegier, prosesser og tråder*. URL: https://uia.instructure.com/courses/13944/files/2260905?module_item_id=512647. (hentet: 23.11.2023).
- [19] Kristian Muri Knausgård. *platformio-oled-can-project-template-mas245*. URL: <https://github.com/kristianmk/platformio-oled-can-project-template-mas245>. (hentet: 25.10.2023).
- [20] Kristian Muri Knausgård. *Prosjekt del 1 Mikrokontrollerkretser*. URL: https://uia.instructure.com/courses/13944/files/2342497?module_item_id=532563. (hentet: 10.10.2023).
- [21] PlatformIO Labs. *Thank you for choosing PlatformIO IDE for VSCode*. URL: <https://platformio.org/install/ide?install=vscode>. (hentet: 27.10.2023).
- [22] Inc. Microchip Technology. *Digital Input/Output Ports on AVR*. URL: <https://microchipdeveloper.com/8avr:ioports>. (hentet: 6.11.2023).
- [23] PEAK-system. *PCAN-USB FD*. URL: <https://www.peak-system.com/PCAN-USB-FD.365.0.html?&L=1>. (hentet: 27.10.2023).
- [24] PJRC Electronic Projects. *Welcome to Teensy 3.6*. URL: https://www.pjrc.com/teensy/card9a_rev2_web.pdf. (hentet: 21.11.2023).
- [25] Inc Red Hat. *What is the Linux kernel?* URL: <https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel>. (hentet: 21.11.2023).
- [26] sarful. *Introduction to ADC in AVR Microcontroller*. URL: <https://www.instructables.com/Introduction-to-ADC-in-AVR-Microcontroller-for-Beg/>. (hentet: 6.10.2023).
- [27] SKPang. *Schematic Rev B*. URL: https://cdn.shopify.com/s/files/1/0563/2029/5107/files/pican_gpsacc_rev_B_15ebcf81-f3c6-4f67-a1ee-86eddfa4d1fe.pdf?v=1619985888. (hentet: 23.11.2023).
- [28] William Stallings. *Operating Systems Internals and Design Principles*. URL: https://repository.dinus.ac.id/docs/ajar/Operating_System.pdf. (hentet: 23.11.2023).
- [29] Vishay. *General Purpose Plastic Rectifier*. URL: <https://www.vishay.com/docs/88503/1n4001.pdf>. (hentet: 20.10.2023).
- [30] Wikipedia. *Executable and Linkable Format*. URL: https://en.wikipedia.org/wiki/Executable_and_Linkable_Format. (hentet: 25.10.2023).
- [31] Wikipedia. *Internet Protocol version 4*. URL: https://en.wikipedia.org/wiki/Internet_Protocol_version_4. (hentet: 23.11.2023).
- [32] Wikipedia. *Internet Stream Protocol*. URL: https://en.wikipedia.org/wiki/Internet_Stream_Protocol. (hentet: 23.11.2023).
- [33] Wikipedia. *Kryptografi*. URL: <https://no.wikipedia.org/wiki/Kryptografi>. (hentet: 23.11.2023).
- [34] Wikipedia. *Transmission line*. URL: https://en.wikipedia.org/wiki/Transmission_line. (hentet: 6.11.2023).