

PROSJEKT INNEBYGDE DATASYSTEMER FOR MEKATRONIKK

Et prosjekt som tar for seg styring av ulike elektriske komponenter ved hjelp av mikrokontrollere og Raspberry pi

JONAS HOLMEN, SIGURD KVALSVIK, THOMAS EIKELAND

VEILEDER
Kristian Muri Knausgård

Sammendrag

Dette prosjektet er basert på flere typer mikrokontrollere og programering av disse ved hjelp av programeringsspråket C++. Blant annet Arduino Teensy og Atmel ICE ble brukt. Prosjektet ble delt opp i tre delprosjekter. Disse gikk henholdsvis ut på mikrokontrollerkretser, CAN-nettverk med Teensy og PCAN-view, og konfigurering av Raspberry pi 3, samt kommunikasjon innad CAN-nettverk mellom flere noder. For å skaffe en oversikt over hvordan de forskjellige kretsene skulle kobles opp, ble det tegnet kretsskjemaer i Eagle. Deretter ble koden implementert, programmene kjørt og resultater registrert. Gjennom dette prosjektet har gruppen fått en dypere forståelse for hvordan innebygde datasystemer med medfølgene restriksjoner på prosessorkraft og størrelse, kan programmeres og kobles for å oppnå et ideelt resultat.

1 Introduksjon

Formålet med dette prosjektet var å lære mer om hvordan tilnærme seg programering av innebygde datasystemer, og lære hvordan å kommunisere med disse. Dette ble gjort gjennom tre delprosjekter med forskjellige temaer. Delprosjekt 1 handlet om å koble hardware selv og få denne til å stemme overens med det som ble satt i IDE-en. I delprosjekt 2 ble CAN-nettverk introdusert. Der var formålet å lære hvordan CAN-meldinger blir sendt og motatt, og også hvordan man printer data på en LED-skjerm. Delprosjekt 3 handlet om å konfigurere buildroot og deretter sende CAN-melding fra raspberry pi 3. Her kobles det også inn både teensy og raspberry pi inn på samme CAN-nettverket.

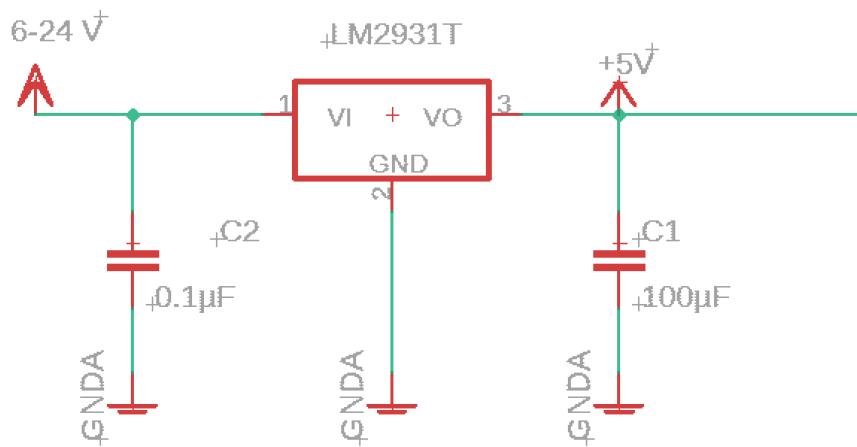
2 Metode

2.1 Spenningsregulator

For å få en konstant spenningskilde med ønsket verdi, benyttes en LM2931-T. Dette er en lineær spenningsregulator, som tar inn 6-24V og gir ut 5V. Atmegaen har et virkningsområde på 2.7 - 5.5 Volt.

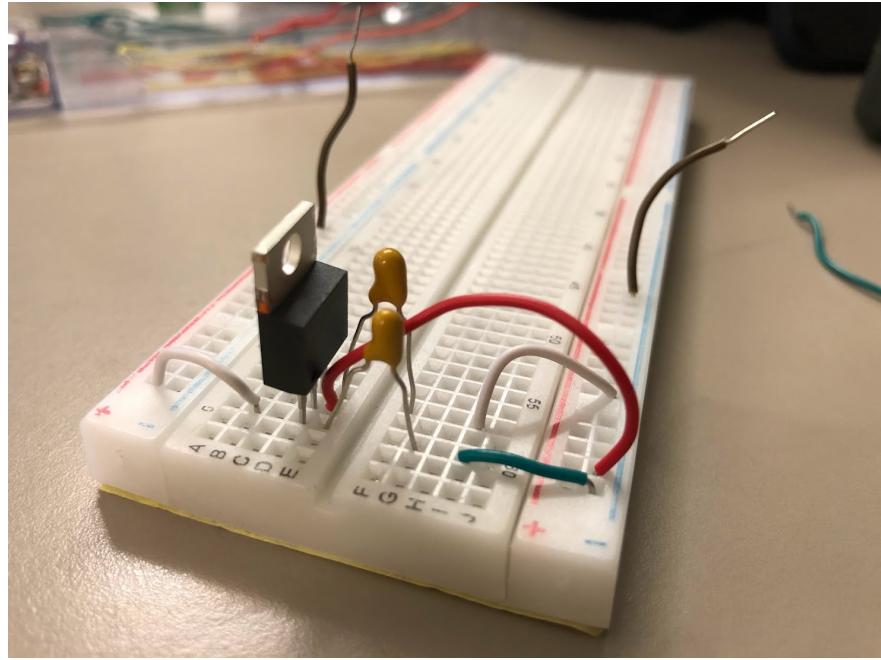
Dersom vi benytter en inngangsspenning på 4.5-5.5 V,får vi et hurtigere system, med potensielt høyere frekvens. I dette systemet skal vi styre en LED. Anbefalt innspenning til spenningsregulatoren er fra 6-24 Volt.

LM2931-T kobles opp med en kondensator som går til jord ved inngangen og utgangen. Her er det kondensatoren som er koblet til utgangen som spiller størst rolle.



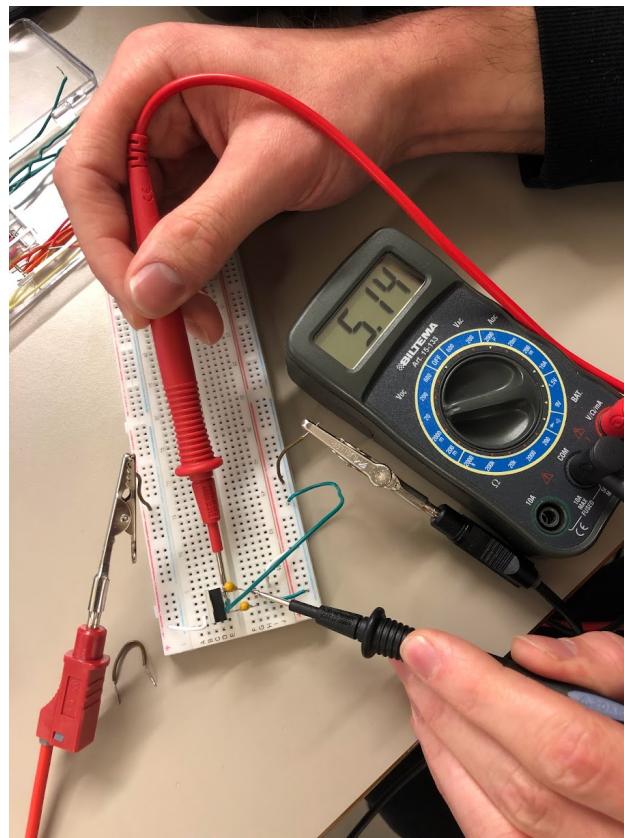
Figur 2.1: Kretsskjemaet til spenningsregulatoren.

Etter verdiene til kondensatorene er valgt, kobles kretsen opp på et breadboard. Strømforsyningen på laben brukes som innspenning på regulatoren, og settes til 10V. Videre kontrolleres utspenningen fra spenningsregulatoren med et multimeter. Denne skal være 5V.



Figur 2.2: Spenningsregulator koblet på breadboard.

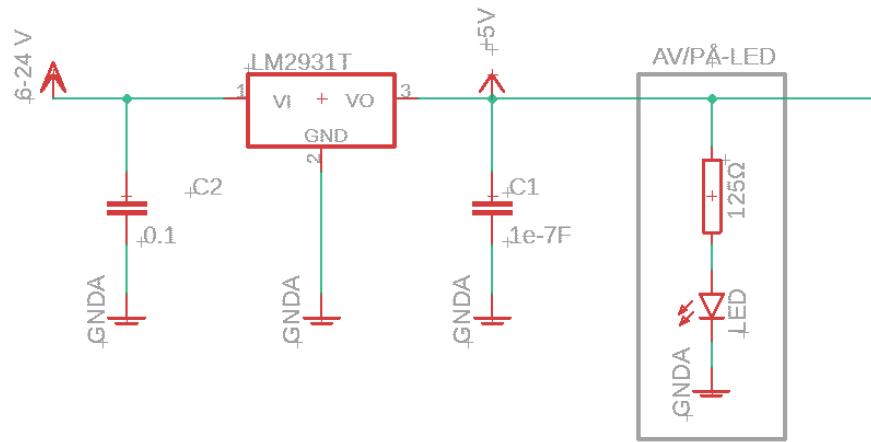
Utspenningen måles på kapasitansen etter ut-pinen. Multimeteret viser 5V, som stemmer med databladet til LM2931-T [5].



Figur 2.3: Kontrollmåling med multimeter.

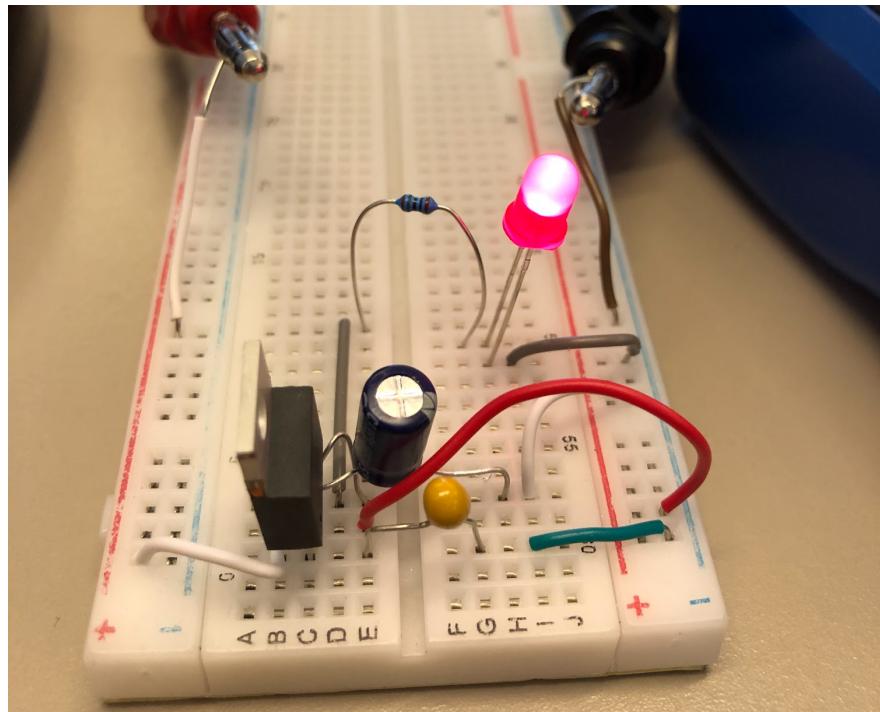
2.2 Kobling av lysdiode

Resistansen i av/på-indikatoren må være minimum 125Ω .

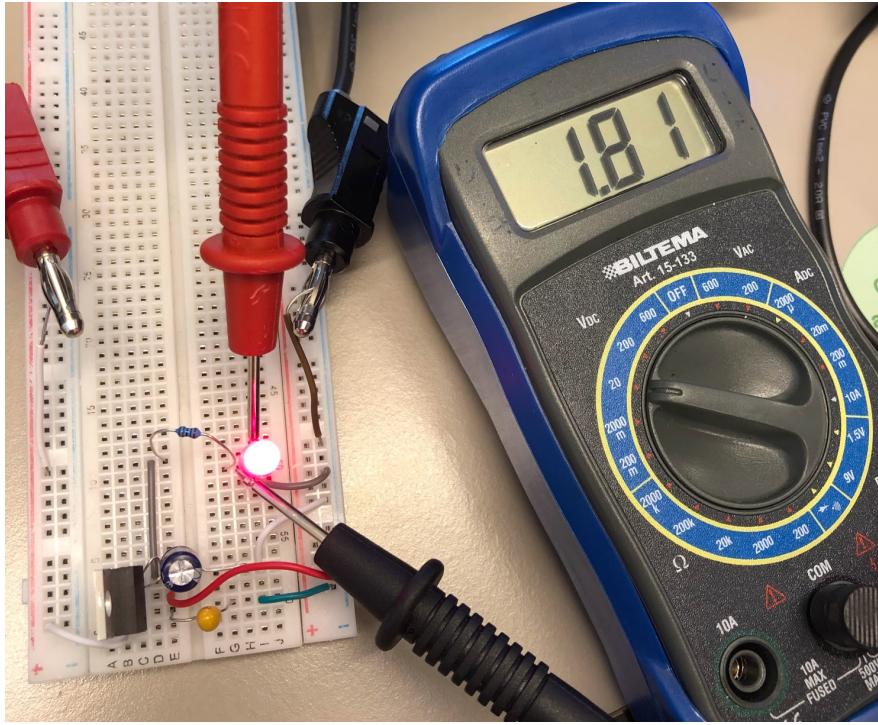


Figur 2.4: koblingskjema av spenningsregulator med av/på indikator på utgangen.

Ved kobling av LED-kretsen benyttes en 125Ω resistanse, for å ha en god sikkerhetsmargin.



Figur 2.5: Kobling av krets med spenningsregulator og av/på-indikator. Denne kretsen er en fysisk modell av figur 2.4



Figur 2.6: Kontrollmåling av spenningen over lysdioden. Multimeteret måler 1.8V, som er innenfor rammene til denne modellen.

2.3 Atmega168 og Pin-Header

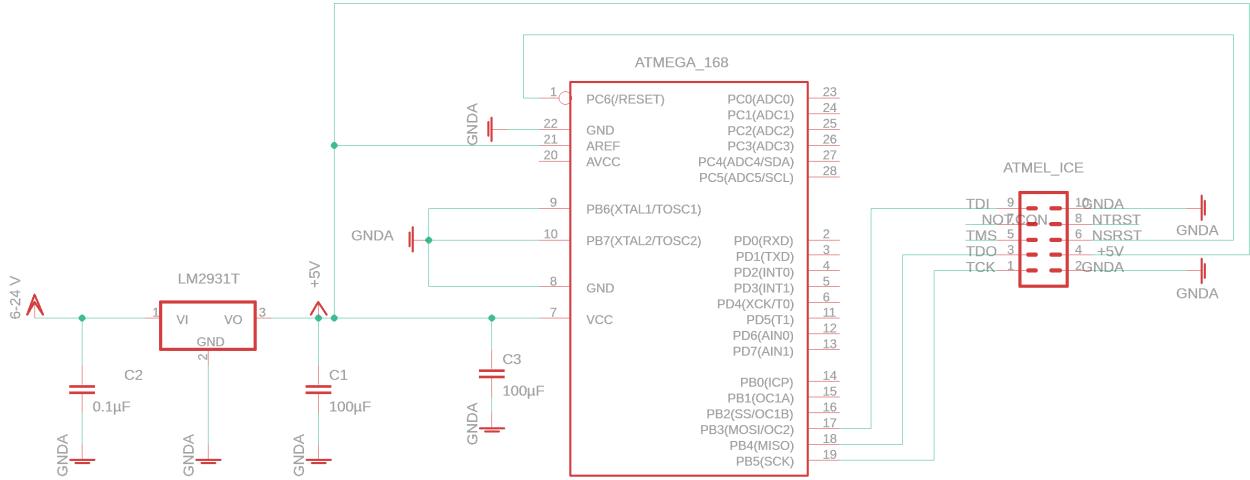
Ved oppkobling av Atmega168 benyttes databladet til å velge riktige komponenter, og koblinger. VCC kobles til utspeningen fra spenningsregulatoren, og en kapasitans som går til jord kobles på inngangen. Hensikten med dette er å redusere støy fra ledningsnettet, slik at spenningen holdes stabil ved inngangen. Begge GND -utgangene kobles til jord. Det er viktig at begge desse er jordet, da de ulike koblingene internt i mikrokontrolleren kan være koblet til bare en av desse, dermed sikrer vi at alle kontakter er hensiktsmessig jordet. $XTAL1$ og $XTAL2$ kobles også til jord. Dette skal gjøres når klokkene ikke er i bruk.

Videre settes en 10-pin header inn. Denne representerer Atmel-ICE, og utgangene kobles opp slik at den kan programmere med ISP (SPI). Her følges pin-kartet fra databladet.

Table 3-6. Atmel-ICE SPI Pin Mapping

Atmel-ICE AVR port pins	Target pins	Mini-squid pin	SPI pinout
Pin 1 (TCK)	SCK	1	3
Pin 2 (GND)	GND	2	6
Pin 3 (TDO)	MISO	3	1
Pin 4 (VTG)	VTG	4	2
Pin 5 (TMS)		5	
Pin 6 (nSRST)	/RESET	6	5
Pin 7 (not connected)		7	
Pin 8 (nTRST)		8	
Pin 9 (TDI)	MOSI	9	4
Pin 10 (GND)		0	

Figur 2.7: Pin-kart for ISP(SPI)-programmering med Atmel-ICE.[2]



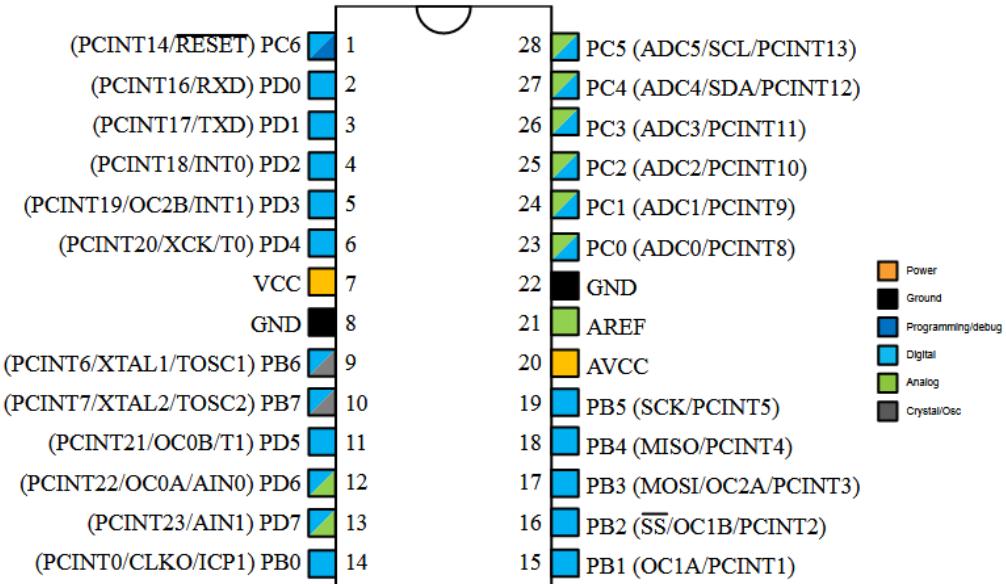
Figur 2.8: Kretsskjema tegnet i Eagle. Sammenkobling av Atmega168 og Atmel-ICE i henhold til SPI pin-mapping.

2.4 Microchip Studio

I Microchip Studio opprettes et GCC C++ Executable Project. Hensikten er å kontrollere at kommunikasjonen fungerer. For å teste dette, blir en av mikrokontrollerpinnene satt som utgang, før den blir tilskrevet en verdi. Det første som må undersøkes er pin-out skjemaet til mikrokontrolleren.

5.1. Pin-out

Figure 5-1. 28-pin PDIP



Figur 2.9: ATMega 168 pin config

Fra illustrasjonen velger vi PB1 som utgang. Databladet inneholder oversikt over de aktuelle minneadressene som videre brukes for å lese og skrive data til PB1.

0x00 (0x00)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	V
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	92

Figur 2.10: ATMega 168 PORTB/DDRB register [1]

For å anvende PB-pinsene, må de først initialiseres som output. Ved bruk av left shift settes den

aktuelle PB1-pinnen til output ved hjelp av følgende kode:

$$DDRB| = 1 << PB1 \quad (2.1)$$

Videre tilskrives output-pinnen verdien høy med følgende kode

$$PINB| = 1 << PB1 \quad (2.2)$$

Som fører til følgende kode som kompilerer uten feil:

The screenshot shows the Atmel Studio interface. In the code editor, there is a single line of C code:

```
int main(void)
{
    DDRB |= 0x0;           // Init DDRB
    DDRB |= 1 << 1;       // Init PB1
    PORTB |= 1 << 1;      // Set PB1 high
    // PORTB &= ~(1 << 1); // Set PB1 low

    while (1)
    {
    }
}
```

In the bottom right corner of the code editor, there is a small green icon indicating successful compilation.

In the 'Output' window, the build process is shown:

```
Done executing task "RunCompilerTask".
Task "RunOutputFileVerifyTask"
    Program Memory Usage : 150 bytes 0,9 % Full
    Data Memory Usage : 0 bytes 0,0 % Full
    Warning: Memory Usage estimation may not be accurate if there are
Done executing task "RunOutputFileVerifyTask".
Done building target "CoreBuild" in project "prosjektioppgave3.cppproj".
Target "PostBuildEvent" skipped, due to false condition: ('$(PostBuildEvent)' != ''
Target "Build" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Avr.common.targets".
Done building target "Build" in project "prosjektioppgave3.cppproj".
Done building project "prosjektioppgave3.cppproj".

Build succeeded.
===== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ======
```

Figur 2.11: Programbyggingen lyktes

Oppgaven blir sjekket inn til versjonskontrollsystemet Git.

The screenshot shows a GitHub commit page for a repository named 'helloAtmega'. The commit message is 'First commit' and it has 1 contributor. The commit details show 22 lines (17 sloc) and 279 Bytes. The commit was created on 24.10.2021 at 16:53:08 by 'Author : S21H'. The commit message includes the C code for the program.

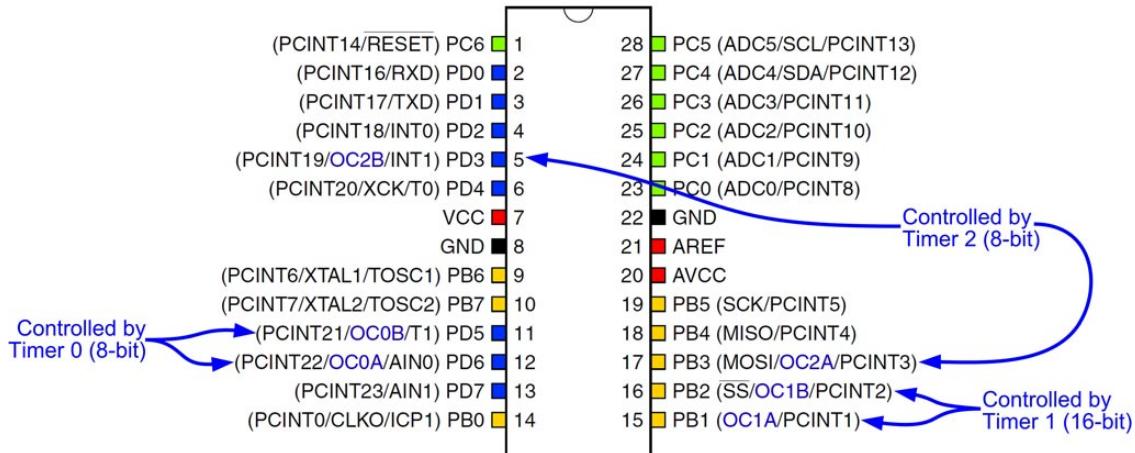
The terminal output on the right shows the command being run:

```
MINGW64 /c/Users/47415/Documents/Atmel studio/7
$ git push origin master:origin
Everything up-to-date
S21H@LAPTOP-ASCETOHE MINGW64 ~/Documents/Atmel studio/7
$ git push -f origin master:main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/sigurk/helloAtmega.git
+ 58470c1..fc0e8ab master -> main (forced update)
S21H@LAPTOP-ASCETOHE MINGW64 ~/Documents/Atmel studio/7
$
```

Figur 2.12: Git repo

2.5 LED med PWM

Hittil er LED kontrollert med utgangen PB1. Denne utgangen har en 16-bits intern timer. Ettersom denne øvelsen skal ha et lavt delay for å gradvis endre lysstyrken, er det kun behov for en 8-bits timer. LED blir derfor flyttet til PD6 som har en intern 8-bits timer. For å anvende denne timeren, referers det til minneadressen OCR0A i registeret. Pulsforholdet til PWM styres av tilført verdi i denne.



Figur 2.13: ATMega 168 pin config med timere [9]

0x28 (0x48)	OCR0B	Timer/Counter0 output compare register B							
0x27 (0x47)	OCR0A	Timer/Counter0 output compare register A							
0x26 (0x46)	TCNT0	Timer/Counter0 (8-bit)							
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00

Figur 2.14: ATMega 168 timer register [1]

For at OCR0A skal fungere optimalt, trenger den en timer å sammenlignes med. Da benyttes den innebygde timeren TCCR0A, som vist i figur 2.14, der WGM01 og WGM00 blir satt til 1. I denne modusen vil TCCR0A telle opp til 0xFF i heksadesimal, før den eventuelt nullstilles og fortsetter tellingen fra 0x0. Videre beholdes COM0A0 som lav, og COM0A1 settes til høy. Dette for å nullstille når timeren har kommet til settpunktet OCR0A. Ettersom LED skal faze opp og ned ved hjelp av et lavt delay, er det ikke behov for prescaling til denne oppgaven. CS00 blir derfor satt til høy, som initialiserer denne.

Table 42. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk _{I/O} /No prescaling)
0	1	0	clk _{I/O} /8 (From prescaler)
0	1	1	clk _{I/O} /64 (From prescaler)
1	0	0	clk _{I/O} /256 (From prescaler)
1	0	1	clk _{I/O} /1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Figur 2.15: CS0x prescaler [3]

Den resulterende koden blir lastet opp til Git Repository, og vedlagt med rapporten.

```

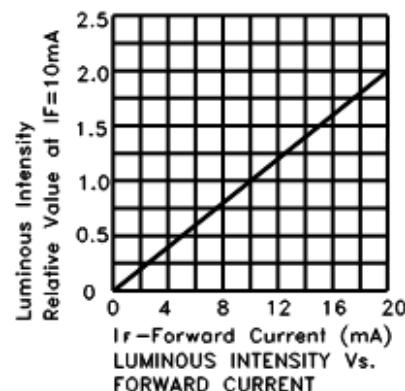
MINGW64:/c/Users/47415/Documents/Atmel studio/7.0/helloAtmega/helloAtmega... - MINGW64 ~/Documents/Atmel studio/7.0/helloAtmega/helloAtmega
$ git push origin master:main
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 12 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (16/16), 14.67 KiB | 4.89 MiB/s, done.
Total 16 (delta 0), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/sigurk/helloAtmega.git
  fc06a2b..16f2269  master -> main
$ |

```

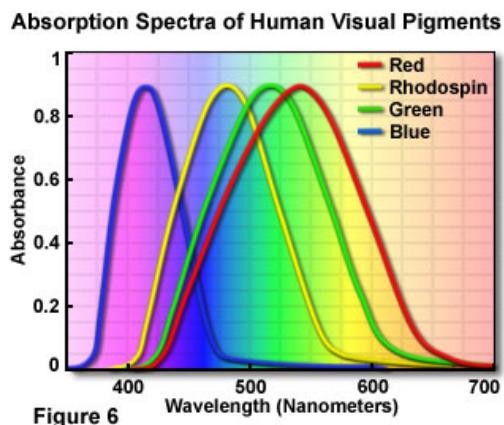
Figur 2.16: Kode og versjonskontroll

Et viktig punkt ved bruk av PWM, er at ved en temperatur på 25 grader vil lysdioden yte en maks strøm på 155 mA. Dette er ved 1/10 av driftssyklusen, med andre ord en pulsbredde på 0.1 ms. Denne verdien er betraktelig større enn maksimal konstant strøm, ettersom lysdioden vil få flere avbrekk til å kjøle seg ned med en såpass lav driftssyklus. Dette må taes hensyn til, og maks strøms ved bruk av PWM bør under ingen omstendigheter overstiges.

Figur 2.17 viser til den lineære sammenhengen mellom lysutbytte som funksjon av strøm. Desto mer strøm tilført, jo mer vil lysstyrken øke. Denne sammenhengen korresponderer ikke med menneskers øyne. I motsetning anses dette forholdet til å være logaritmisk, og understrekkes ved blant annet øynene sin funksjon til å tilpasse seg dags-og nattlys. 2.18 viser forskjellige bølgelengder opp mot prosentvis absorberingsrate til menneskeøyne.



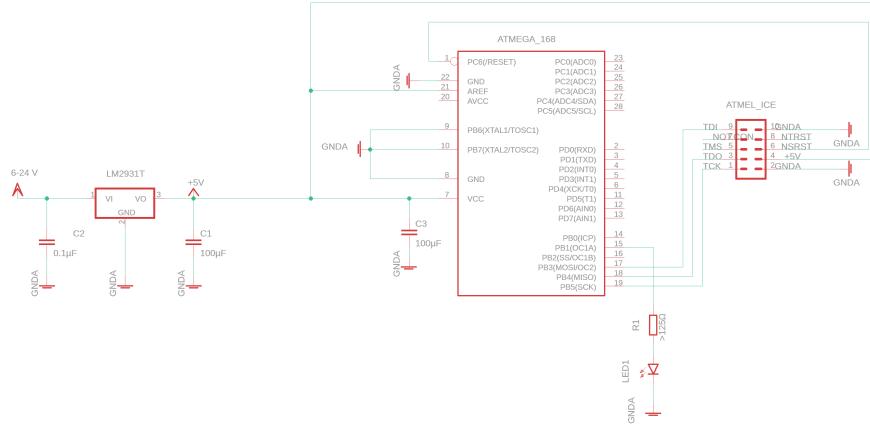
Figur 2.17: Lysutbytte som funksjon av strøm [7]



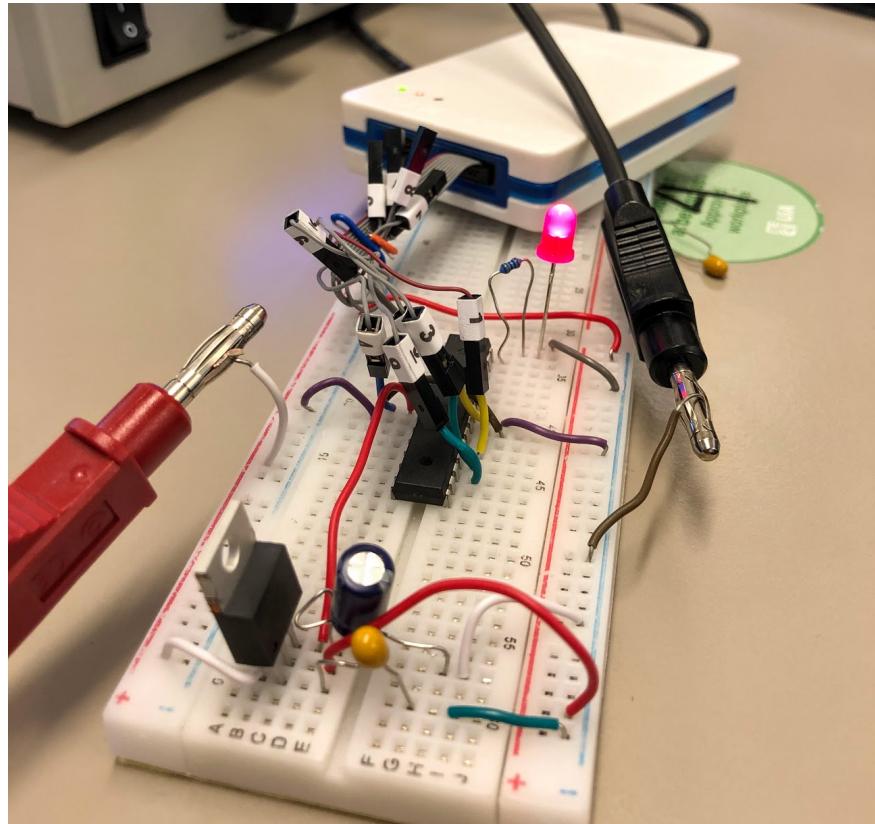
Figur 2.18: Absorbert bølgelende spekter [6]

2.6 Kobling av LED som av/på-indikator

Innledningsvis tegnes et kretsskjema i Eagle, som viser hvordan lysdioden skal implementeres. Deretter kalkuleres den nødvendige resistansen i LED-kretsen.



Figur 2.19: ATMega168 og Atmel ICE rekonfigurerert til å ha lysdiode



Figur 2.20: Fysisk kobling av krets i figur 2.19

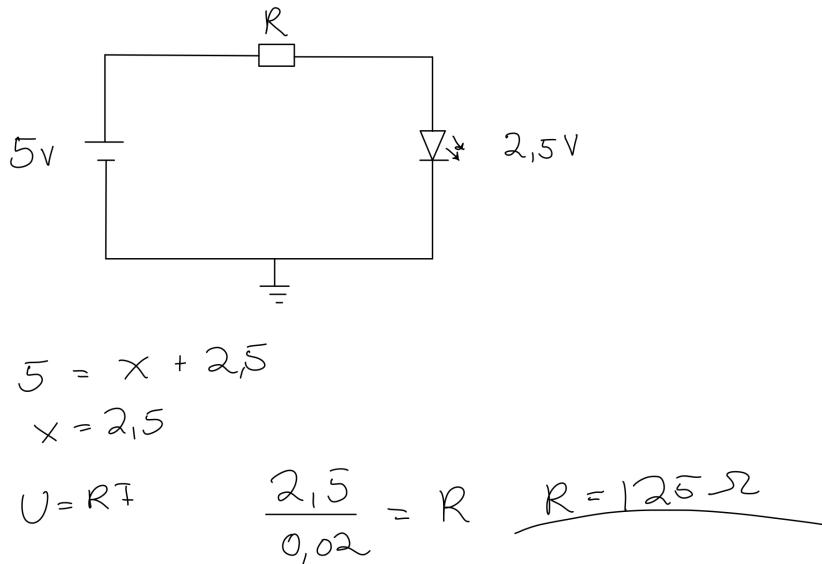
I databladet til lysdioden (figur 2.21) er det spesifisert at max-spennin er 2.5 V.

Electrical / Optical Characteristics at $T_A=25^\circ C$

Symbol	Parameter	Device	Typ.	Max.	Units	Test Conditions
λ_{peak}	Peak Wavelength	Super Bright Red Super Bright Green	660 565		nm	IF=20mA
λ_D	Dominant Wavelength	Super Bright Red Super Bright Green	640 568		nm	IF=20mA
$\Delta\lambda_{1/2}$	Spectral Line Halfwidth	Super Bright Red Super Bright Green	20 30		nm	IF=20mA
C	Capacitance	Super Bright Red Super Bright Green	45 15		pF	VF=0V; f=1MHz
V_F	Forward Voltage	Super Bright Red Super Bright Green	1.85 2.2	2.5 2.5	V	IF=20mA
I_R	Reverse Current	All		10	uA	$V_R = 5V$

Figur 2.21: datablad for LED

Utspenningen fra spenningsregulator LM2931-T er på 5V, dermed må vi implementere en resistanse som har et spenningsfall på minst 2.5V i av/på indikator-kretsen.



Figur 2.22: Beregning av resistanse for lysdiode-krets.

Siden lysdioden har en max spenning på 2.5V ble det brukt en seriemotstand med nok motstand til å gi et spenningsfall på minst 2.5V. For å kalkulere nødvendig resistanse, brukes Omhs lov og lysdiodens typiske strømpå A, 0.02 som vist i figur 2.22.

Etter kretsskjema var ferdigstilt og nødvendige utregninger fullførte ble lysdioden koblet opp fysisk, og en funksjonstest ble utført ved å koble til powersupply på spenningsregulatorens inngang.

2.7 Oppkobling av krets, og programmering av mikrokontroller.

Etter å ha benyttet lysdioden som av/på-indikator på utgangen av spenningsregulatoren, skal mikrokontrolleren kobles opp, og lysdioden skal kobles slik at den kan styres av mikrokontrolleren.

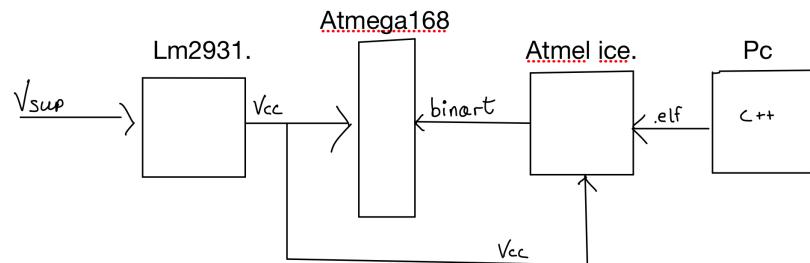
Kretsen i figur 2.8 kobles opp på breadboard i henhold til SPI pin-mapping, og lysdioden kobles til PB1 på mikrokontrolleren. Dette fordi PD6 er satt som utgang i koden som styrer mikrokontroleren.

Kretsen kobles slik at mikrokontrolleren sourcer strøm til lysdioden. Dette betyr at mikrokontrolleren leverer spenningen som får lysdioden til å lyse. Så lenge mikrokontrolleren kan levere nok spenning til å drive lysdioden, er det mest hensiktsmessig at den sourcer.

Videre benyttes Atmel-ICE for å sammenkoble krets og PC. For å verifisere at sammenkoblingen er vellykket, kontrolleres device signature av i Microchip studio.

2.8 Krysskompilering

Når programmet kompileres i Atmel Studio for AVR-mikrokontrollere, utføres det en krysskompilering. Dette vil si at programmet først komplieres på windows slik at koden kan kjøres og deretter kompileres på en annen platform enn den originale platformen, i dette tilfellet Atmel-ICE.

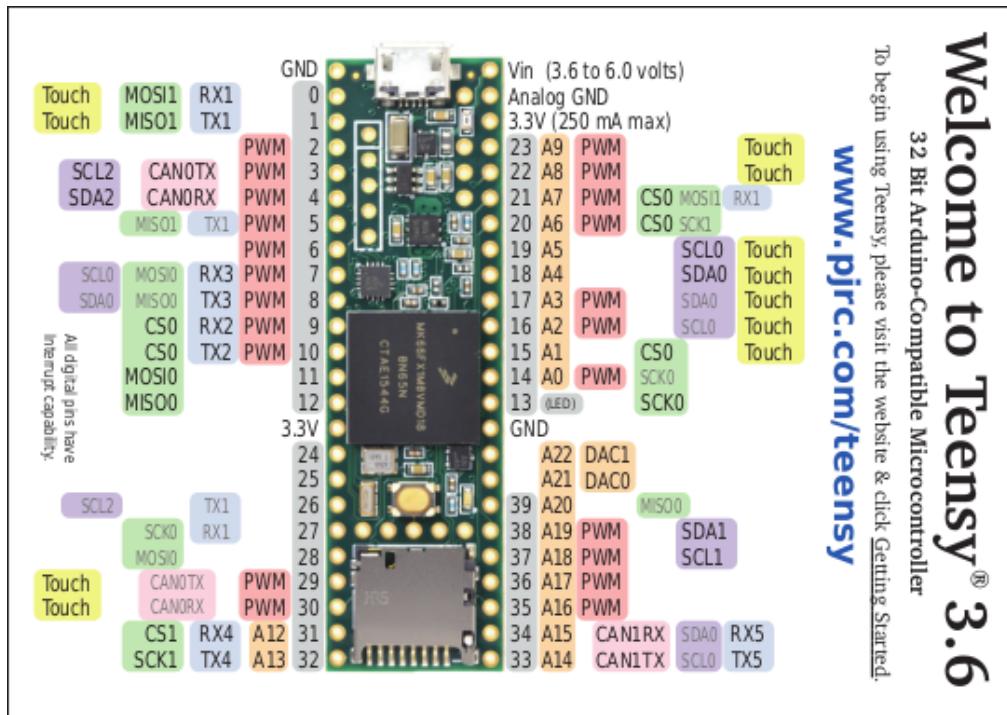


Figur 2.23: Skjematisk oversikt over hvordan systemet fungerer.

I dette blokkdiagrammet er det PC som er host og target er Atmega168.

2.9 Meldingsoverføring ved hjelp av flexCAN

2.9.1 Teensy flexCAN



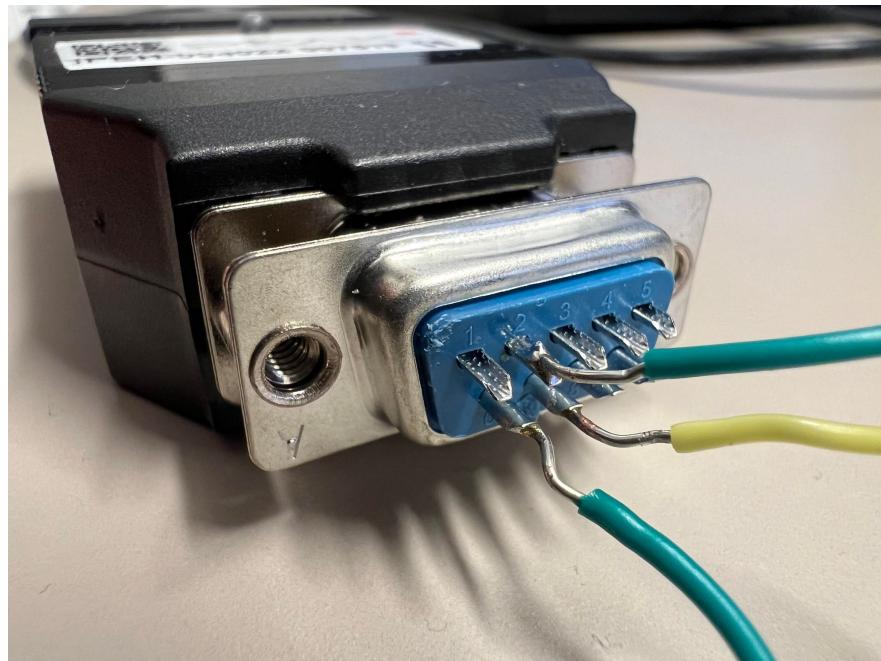
Figur 2.24: Teensy 3.6 pin-kart [8]

På Teensy 3.6 carrier board er det implementert to CAN-tranceivere. Disse brukes til å sende og motta meldinger. CAN0 er koblet på pin 3 og 4, og CAN1 er koblet på pin 33 og 34.

I utgangspunktet skulle en D-sub connector med et breakoutkort benyttes for å koble sammen Teensy 3.6 med PCAN-adapteren. Ved å kontrollmåle koblingene mellom CANH, CANL, VCC og GND, ble en kortslutning oppdaget. Derfor erstattes overgangen med en D-sub connector med male-utgang, hvor kontaktene fra CAN-busen loddet rett på D-sub connectoren. Her følges pin-mapping for PEAK og PCAN. Dette for å få kontakt mellom CAN-busen og PCAN view.

D-Sub	Pin	Pin assignment
	1	Not connected / optional +5V
	2	CAN-L
	3	GND
	4	Not connected
	5	Not connected
	6	GND
	7	CAN-H
	8	Not connected
	9	Not connected / optional +5V

Figur 2.25: Peak pin-mapping [4]

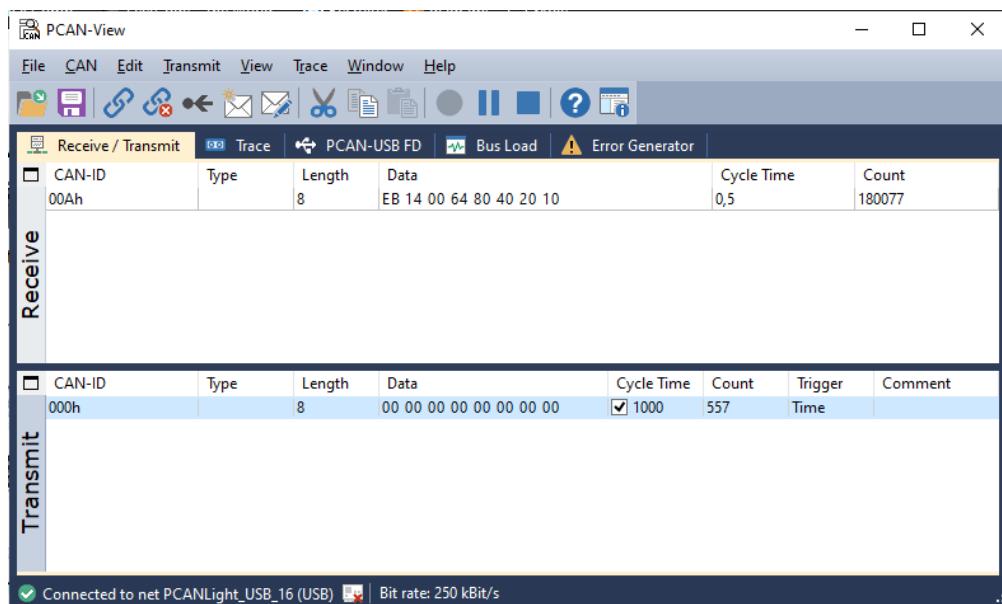


Figur 2.26: Loddet D-sub connector

For å terminere støy og skader på signalet, kan en terminalresistanse benyttes på CAN-inngangen. Dette er spesielt nødvendig dersom man sender meldinger over lange kabler, og dersom man kjører med høy bitrate. I dette systemet benyttes verken lange kabler, eller høy bitrate. Derfor er det i utgangspunktet ikke nødvendig med termineringsmotstand. Carrier kortet har en innebygget termineringsmotstand, men den er ikke aktivert. Om nødvendig, er det enkleste da å koble inn en enkel mostand mellom CAN_L og CAN_H. Denne skal være på 120Ω , i henhold til ISO 11898-2 (CAN-bus standard).

2.9.2 PCAN-View

Ved bruk av PCAN-View og eksempelprogrammene til SK Pang skal gruppen overføre en melding mellom Teensy og Peak PCAN. Etter PEAK PCAN er koblet til, nødvendige drivere er lastet inn og koden kjører blir meldingene sendt riktig.



Figur 2.27: PCAN-View transceiver

Fra figur 2.27 sender Flexcan 8 bytes data kontinuerlig. Flexcan kan sende fra 0 til 8 bytes med data i hver melding ved bruk av den aktuelle CAN.

2.9.3 Nødvendige steg for å sende og motta CAN-meldinger

For å sende en melding fra Arduino IDE til PCAN-View, må man gjøre følgende steg:

Først og fremst må teensyen være korrekt koblet sammen med CAN-trancieveren. Dette gjøres i henhold til datablad, som vist ovenfor. Deretter må programvaren settes opp. Først initialiseres CAN-biblioteket i setup-funksjonen. Her brukes følgende kode:

```
Can1.begin(250000); // her settes Bitraten til 250 kBit/s
```

Videre må man definere meldingen man ønsker å sende. Dette kan gjøres på følgende måte:

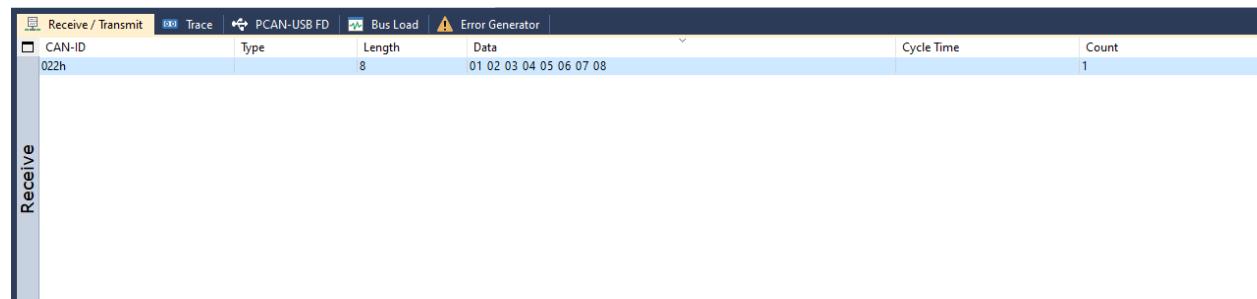
```
msg.len = 8;      // størrelsen på meldingen
msg.id = 0x22;    // meldings-ID

msg.buf[0] = 1;   // meldingen man ønsker å sende
msg.buf[1] = 2;
msg.buf[2] = 3;
msg.buf[3] = 4;
msg.buf[4] = 5;
msg.buf[5] = 6;
msg.buf[6] = 7;
msg.buf[7] = 8;
```

Deretter gjenstår det bare å sende meldingen. For å gjøre det brukes følgende kode:

```
Can1.write(msg); // send meldingen via CAN-bus 1
```

Når meldingen er sendt, dukker den opp i receive-feltet i PCAN-view. Her ser man meldings-ID, antall meldinger motatt, cycle time, størrelsen på meldingen, og data. Data er den faktiske meldingen.



The screenshot shows the PCAN-View software interface. The top menu bar includes 'Receive / Transmit', 'Trace', 'PCAN-USB FD', 'Bus Load', and 'Error Generator'. On the left, there's a vertical sidebar labeled 'Receive' which is currently active. The main window displays a table of received messages. The table has columns for CAN-ID, Type, Length, Data, Cycle Time, and Count. One row is visible, showing CAN-ID 022h, Type 8, Length 8, Data bytes 01 02 03 04 05 06 07 08, and Count 1. The 'Cycle Time' column is empty.

CAN-ID	Type	Length	Data	Cycle Time	Count
022h		8	01 02 03 04 05 06 07 08		1

Figur 2.28: Mottatt melding i PCAN-view.

2.10 Vise informasjon på OLED-skjerm

For å vise informasjon på OLED-skjermen, brukes biblioteket Adafruit SSD1306. For å skrive tekst på skjermen, kan følgende kode brukes:

```
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(8, 7);      // Posisjonen teksten skal printes
display.write("MAS245 - Gruppe 8");
```

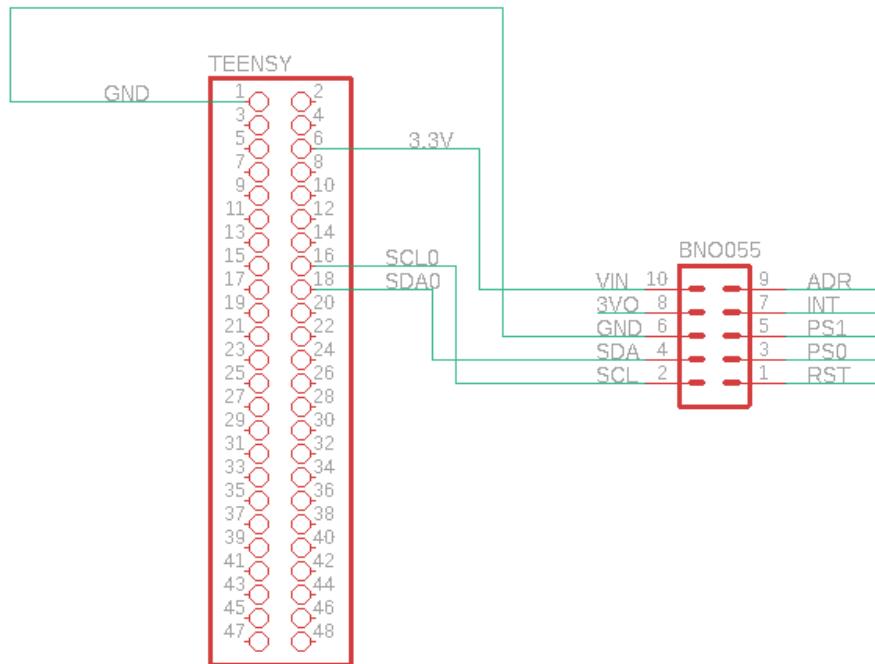
På denne måten settes skjermen opp for å liste opp CAN-statistikken, og IMU-målinger. For å få live oppdatering fra CAN-nettverket, brukes følgende kode:

```
while (Can1.read(rxmsg)) {
    msgCount += 1;

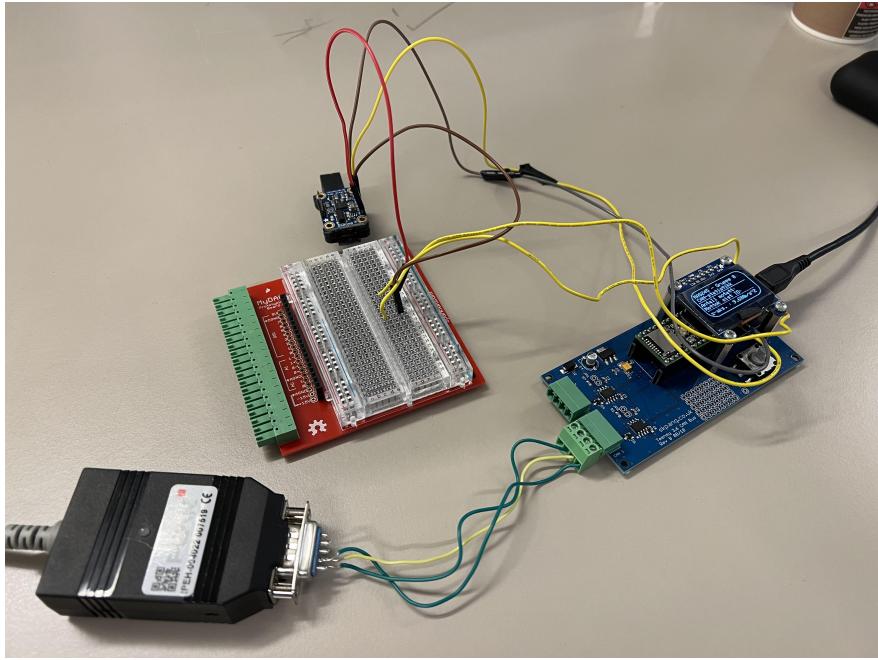
    display.setTextSize(1);
    display.setTextColor(SSD1306_WHITE, BLACK );
    display.setCursor(96, 28);
    display.cp437(true);
    display.println(msgCount);
}
```

Hver gang Can1.read(rxmsg) leser en melding, øker msgCout med en, og printes på skjermen. På samme måte printes melding-ID, da brukes display.println(rxmsg.id, HEX).

2.11 Lesing og sending av IMU-målinger



Figur 2.29: Kretskjema som viser koblingene mellom IMU og Teensy



Figur 2.30: Kobling av Teensy og IMU. Her brukes et breadboard for å kobling med IMU

I denne seksjonen skal det måles IMU-verdier i form av akselerasjon. Desse skal deretter kunne sendes via et CAN-nettverk, ved etterspørsel. I tillegg skal de også vises på LED skjermen som er montert på Teensyen.

I utgangspunktet skulle en MPU6050 brukes for å lese av akselerasjonsdata. Etter å ha lastet ned de nødvendige bibliotekene i Arduino IDE, og koblet IMU i henhold til datablad, var det ikke mulig å få kontakt med IMU. Her ble det også forsøkt med en ny sensor, men det fungerte ikke. Dermed ble MPU6050 byttet ut med en BNO055, og biblioteket Adafruit_BNO055.h ble brukt. Denne sensoren fungerte umiddelbart, og akselerasjon ble lest av i x-, y- og z-retning.

For å få en meldingsrate på 1Hz, brukes Metro-biblioteket. Med følgende kode initialiserer Metro-biblioteket et intervall på 1 sekund:

```
Metro serialMetro = Metro(1000);
```

For å sende IMU-målingene, brukes samme metode som i seksjon 2.9.3. Her defineres meldingen ved å sette msg.buf[0] = x;; msg.buf[1] = y; og msg.buf[2] = z;. For å initialisere utsendingen av data med en rate på 1 Hz, brukes følgende kode:

```
Can1.read(rxmsg);
if (( rxmsg.id == 0x21) & (serialMetro.check() == 1)) {
    msg.len = 8;
    msg.id = 0x22;
    msg.buf[0] = x;
    msg.buf[1] = y;
    msg.buf[2] = z;
    msg.buf[3] = 0;
    msg.buf[4] = 0;
    msg.buf[5] = 0;
    msg.buf[6] = 0;
    msg.buf[7] = 0;

    Can1.write(msg);
}
```

I denne koden brukes Can1.read(rxmsg) for å lese inn alle meldingene som blir sendt via CAN-

nettverket. Videre brukes if-setnigen for å bestemme når meldingsoverførsel skal initialisere. Med rxmsg.id == 0x21, bestemmer den at sensordata skal sendes dersom meldings-ID er 21. SerialMetro.check() == 1 sier at hver gang serialMetro har gjennomført 1 intervall, skal en melding defineres og sendes. Dette gir en fast rate på 1Hz.

For å vise IMU-målingene på LED-skjermen brukes følgende kode:

```
display.setTextColor(WHITE, BLACK );
display.setCursor(60, 50);
display.println(z);
display.display();
```

Ved å sette display.setTextColor(WHITE, BLACK), skrives teksten i hvit, og bakgrunnen i svart. Dette gjør at LED-skjermen blir klarert i det området hvor akselerasjonen blir printet, før hver nye måling.

2.12 Konfigurasjon av buildroot

For at Linux-distribusjonen skal fungere med CAN-bus på kombinasjonen av Raspberry Pi 3 og Skpang CAN-kort, benyttes Buildroot. Dette verktøyet brukes for å konfigurerer og bygge Linux-distribusjoner for innebygde datasystemer. Fra dette biblioteket benytter gruppen raspberrypi3_defconfig, som kjører på 32-bits prosessarkitektur.

I tillegg installeres libncurses-dev. Dette programvarebiblioteket tillater gruppen å anvende et tekst-basert brukergrensesnitt til datasystemet. I denne oppgaven skal vi kjøre raspberrypi3_3config. Dette er en default konfigurasjon, som benytter 32-bits prosessarkitektur.

Prosjektet blir pakket ut i Linux, og en kopi av buildroot klones inn i prosjektmappen. Herfra opprettes en menuconfig ved hjelp av følgende kode:

```
make O=${PWD}/buildroot-output/ BR2_EXTERNAL=${PWD}/external -C ...
${PWD}/buildroot menuconfig
```

PWD står for Print Working Directory. \$PWD tilsier mappeadressen fra root til nåverende mappe, mens \${PWD}/buildroot-output/ bygger videre til den aktuelle mappen som skal lese eller skrives til. Videre er BR2_EXTERNAL en funksjon som tillater å lagre konfigurasjons- og brett support-filer utenfor Buildroot tree, mens de fortsatt er integrert i byggeprosessen.

MAS245-konfigurasjonsfilen blir lagt inn i prosjektet sin output mappe med følgende kode:

```
cp 2021_mas245_rpi3_can_qt5qmake_staticip_iproute2.config ...
buildroot-output/.config
```

Lokale variabler i konfigurasjonsfilen blir endret, navnet blir endret til .config, før den limes inn i buildroot-output mappen der den overskriver eksisterende konfigurasjonsfil. Ved å kjøre menuconfig igjen, sjekker gruppen om alle filstier er korrekte. Disse er konfigurert til en annen bruker, og blir endret til det lokale systemet sine filstier. Når dette er korrekt starter gruppen kompileringen av prosjektet, ved å kjøre make fra buildroot-output mappen.

Kompileringsprosessen til buildroot tar omlag 70 minutter, og ved vellykket kompilering blir det opprettet en image fil som skal skrives til det utleverte mikroSD-kortet. MikroSD-kortet blir koblet til en USB 3.0 kortleser. For at operativsystemet skal kunne lese og skrive til denne, må Oracle VM oppdateres til nyeste versjon slik at VM Extension Pack kan installeres. Ved hjelp av kommandoen fdisk -l i terminalen viser systemet at sd kortet ligger på måldiskken /dev/sdb/. Imagefilen blir skrevet til måldiskken ved hjelp av følgende kommando:

```
sudo dd if=buildroot-output/images/sdcard.img of=/dev/sde bs=1M
```

For at operativsystemet skal kunne kommunisere med RPI-maskinen, kobles det til med SSH. Under nettverksinnstillinger i VM Manager settes nettverket til "Attached to Bridged Adapterpå Ethernet Controlleren. Promiscuous Mode gjør at all trafikk blir sent til enheten, i stedenfor kun programerte verdier. Siden prosjektet blir utført i et kontrollert miljø blir denne satt til "Allow All". Videre blir den statiske IP adressen til den virtuelle maskinen satt til 192.168.245.1 med nettmaske 255.255.255.0. Denne IP adressen er for å sørge for at enheten er på samme subnet, da image filen er initialisert til å være på 192.168.245.245. SSH forbindelsen blir verifisert med følgende kode:

```
ssh root@192.168.245.245
```

Can_pingpong-pakken blir bygget ved hjelp av følgende kode:

```
make -C /home/student/buildroot-projects/  
buildroot-output/ can_pingpong-rebuild
```

Før den videre krysskomplieres til RPI-maskinen med følgende kode:

```
scp /home/student/buildroot-projects/buildroot-output/build/  
can_pingpong-3.14/can_pingpong root@192.168.245.245:/usr/bin/can_pingpong
```

2.13 Sende CAN-meldinger fra Raspberry pi

For å sende og motta meldinger via et CAN-nettverk, kobles Raspberry Pi til Teensy via CAN-busen. SKpang-kortet har en innebygd termineringsresistans, men den er ikke aktivert. Om nødvendig, er det enkleste da å koble inn en enkel mostand mellom CAN_L og CAN_H. Denne skal være på 120Ω , i henhold til ISO 11898-2(CAN-bus standard).

CAN-meldingene skal nå sendes fra terminal. Operativsystemet og RPI-maskinen er fra tidligere koblet sammen ved hjelp av SSH. For at RPI-maskinen skal kunne prosessere CAN-meldingene, må konfigurasjonen for maskinen legges til. Drivere for SPI og CAN-tranceiveren på SKPANG-kortet legges inn i config.txt med følgende kode:

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

RPI-maskinen startes. Ethernet tilkoblingen blir aktivert med følgende kommando:

```
ifup eth0
```

SPI og CAN-tranceiveren, tidligere lagt inn i config.txt, blir aktivert med følgende kommandoer:

```
modprobe spi_bcm2835  
modprobe mcp251x
```

Før CAN-grensesnittet blir konfigurert med følgende kommando:

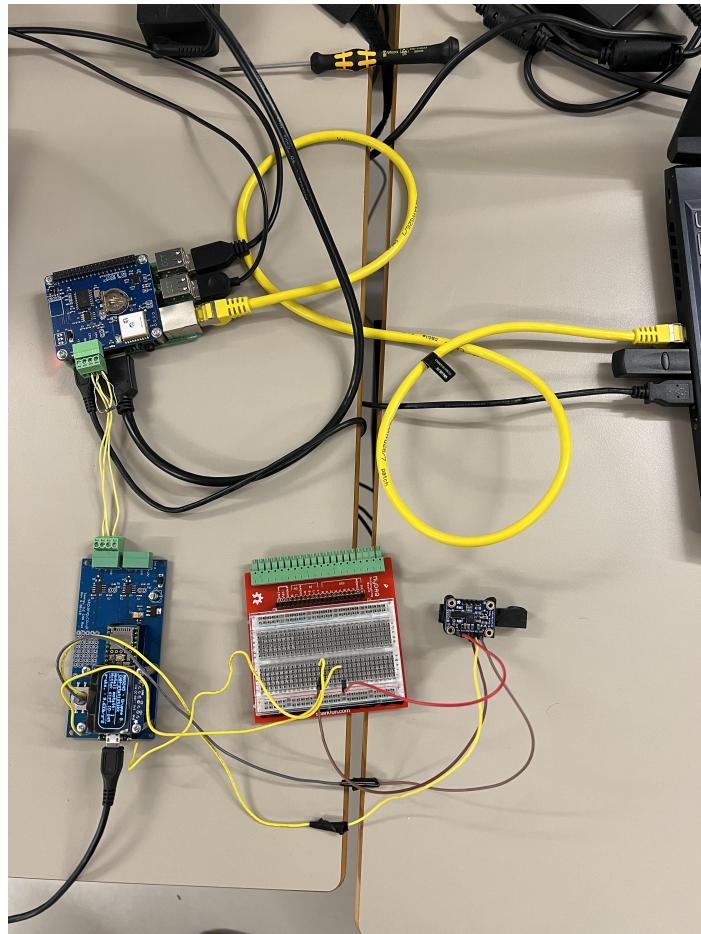
```
/sbin/ip link set can0 up type can bitrate 250000
```

PCAN-View og RPI-systemet er nå satt med samme bitrate, og det er mulig å sende og motta CAN-meldinger med følgende kommandoer:

```
candump can0 // Printer mottatte meldinger  
cansend can0 245#23.40.00.00.FF.FF.FF.00 // Sender melding
```

2.14 Kommunikasjon over CAN-nettverket

I denne seksjonen skal Teensy og Raspberry Pi kobles sammen via et CAN-nettverk, slik at meldinger kan overføres frem og tilbake mellom enhetene. For å koble enhetene sammen, kobles CAN-bus 1 på Teensy til CAN-bus 0 på Raspberry Pi.



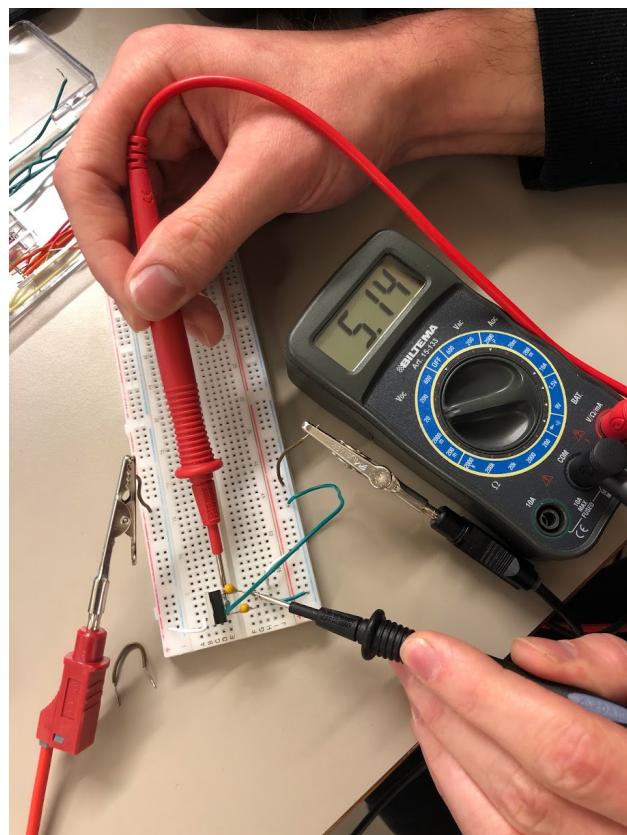
Figur 2.31: Sammenkobling av Teensy og Raspberry pi

Teensyen er satt opp på samme måte som i seksjon 2.11, hvor den måler akselerasjon med en IMU-sensor og ved etterspørrelse sender disse målingene via CAN-nettverket som leses av PCAN-view.

Her skal det samme konseptet utføres, men med Raspberry pi som sender/mottaker. Teensyen skal lese sensordata fra IMU. Når man sender en melding fra Raspberry Pi med meldings-ID 245, vil Teensy starte å sende sensordata en gang i sekundet til Raspberry pi.

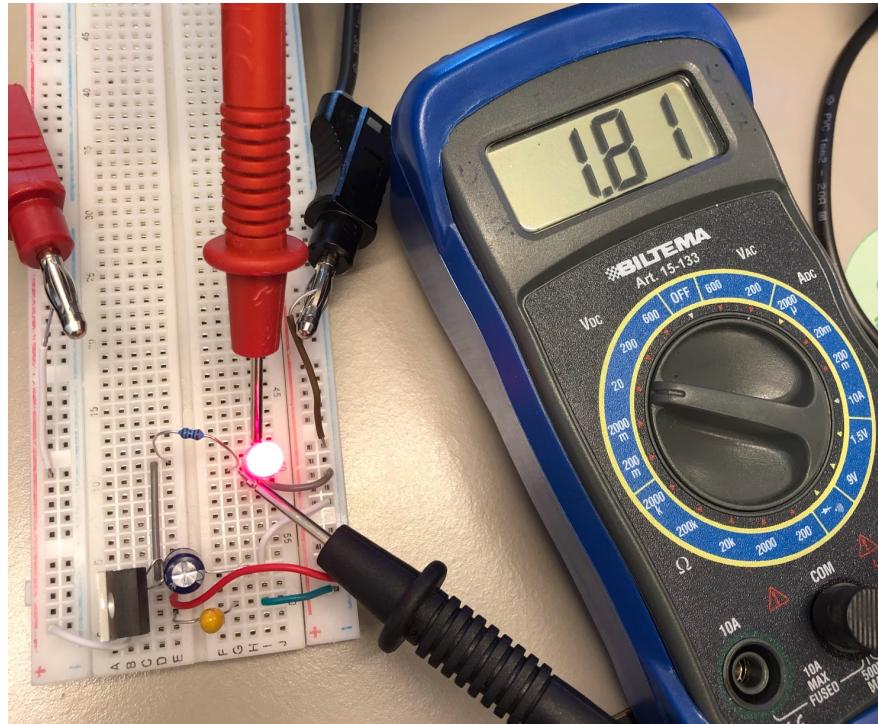
3 Resultater

3.1 Spenningsregulator



Figur 3.1: Kontrollmåling med multimeter.

3.2 Kobling av lysdiode



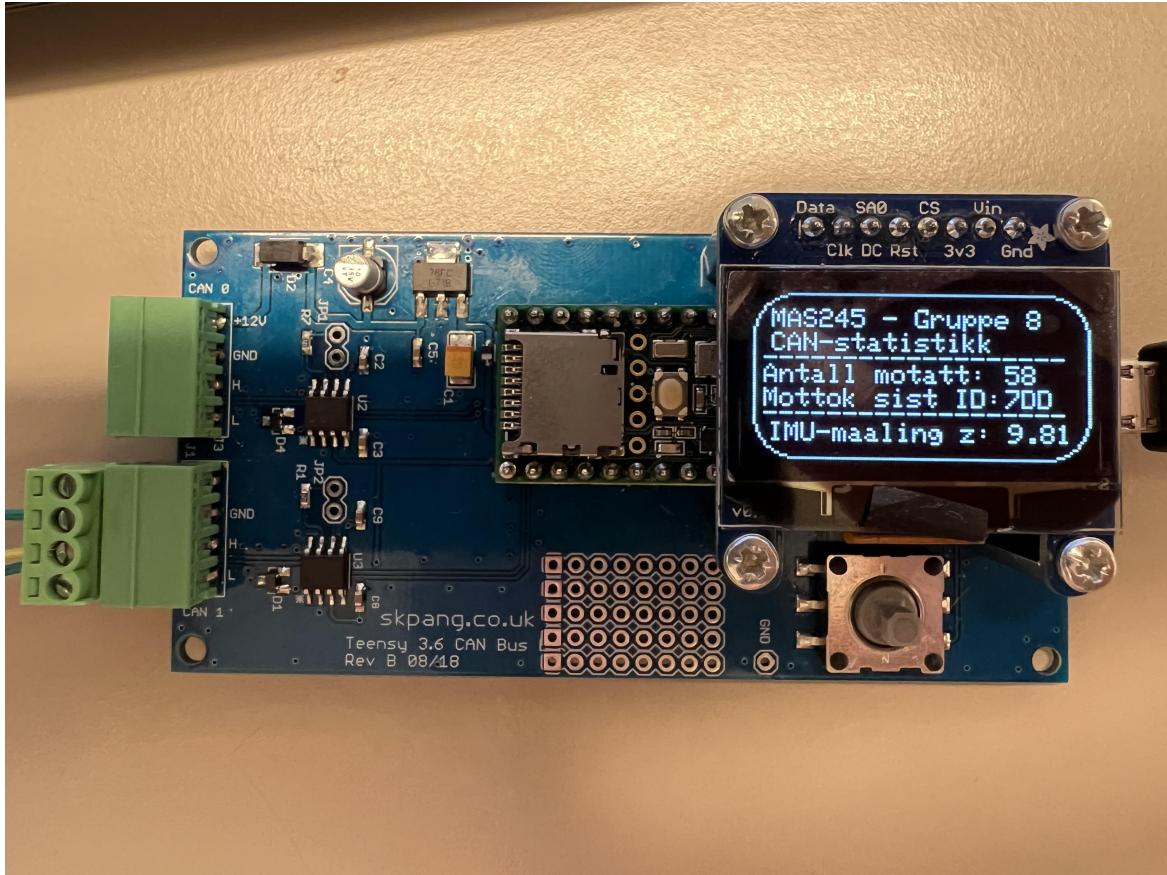
Figur 3.2: Kontrollmåling av spenningen over lysdioden. Multimeteret måler 1.8V, som er innenfor rammene til denne modellen.

3.3 Soft blink

Gruppen forsøker flere pulsbredder for PWM. Ved høyere pulsbredder, har lysdioden en lengre og mykere soft-blink. Ved lavere er den kortere. Med en pulsbredde på 40/255 oppnår gruppen det mest optimale resultatet. Lysdioden er rask, men samtidig myk. Denne koden blir pushet opp til prosjektets Git Repository. Video av resultatet ved bruk av pulsbredde på 40/255, 125/255 og 255/255 ligger vedlagt rapporten.

3.4 CAN message receiver

Teensykort med programert display av antall meldinger motatt og ID på den siste. Og IMU måling av akselerasjon lang z-aksen. En demonstrasjonsvideo av dette er vedlagt i den leverte zip-mappen, og heter "msgCount.MOV". Her ser man at meldingstelleren på skjermen øker for hver nye melding.



Figur 3.3: Display av Can statistikk

3.5 Rapportere IMU verdier via CAN bus og vise på skjerm OPG 4a

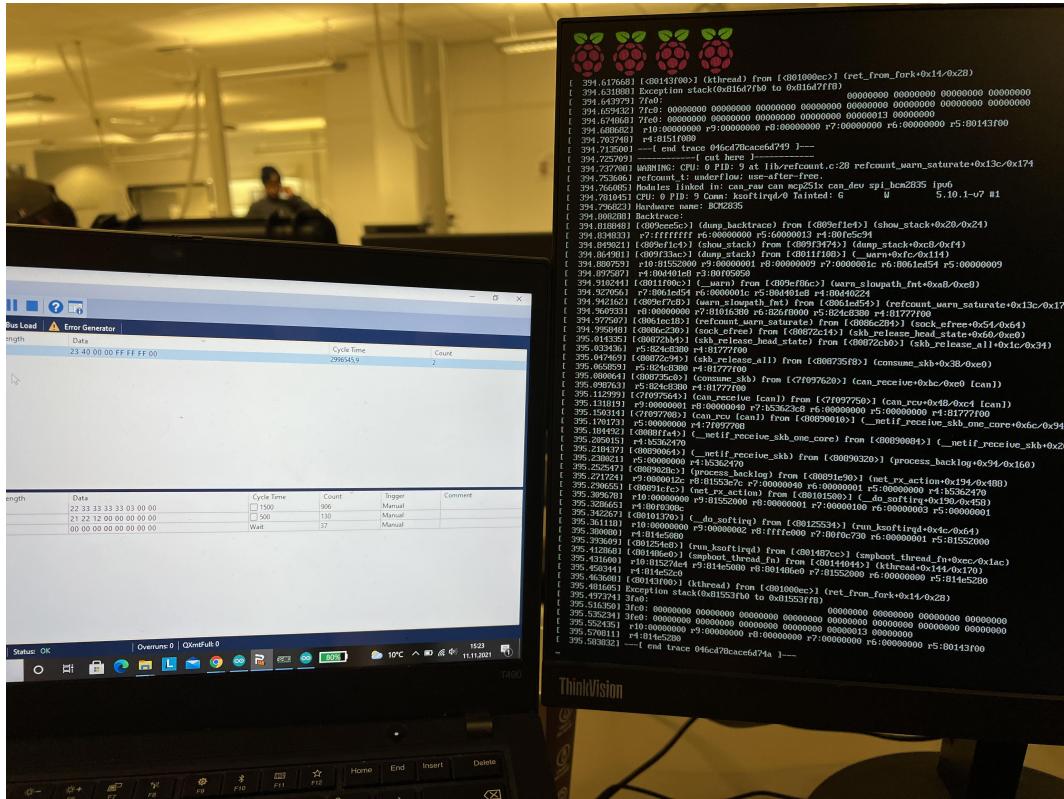
CAN-ID	Type	Length	Data	Cycle Time	Count
022h		8	00 05 07 00 00 00 00 00	948,8	49

Figur 3.4: Motatt sensordata i PCan-view. Cycle time viser at data blir sendt $\approx 1\text{Hz}$

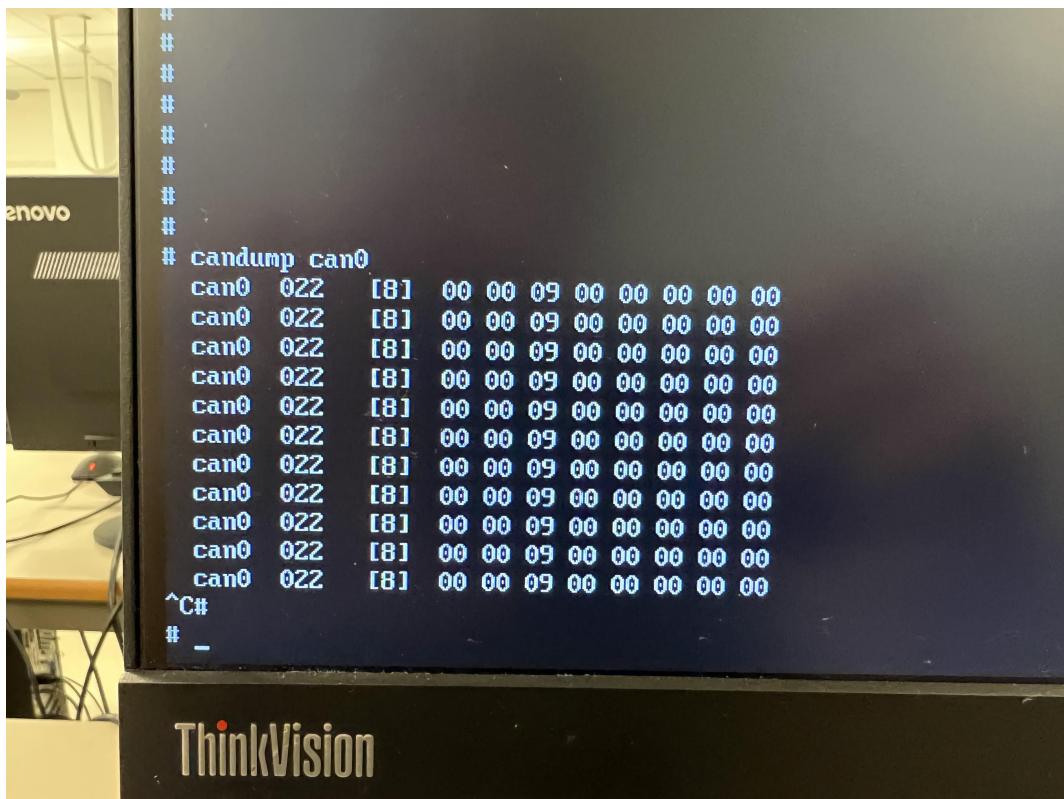
Implementeringen av en IMU sensor resulterte i en velfulgerende akselerasjonsmåling, som ble videreført via CAN-nettverket med en meldingsrate på 1 Hz, ved etterspørrelse. Akselerasjon langs z-aksen ble også printet fortøpende på OLED-skjermen. En demonstrasjonsvideo av dette er vedlagt i Zip-filen, og heter "IMUmålingDisplay.mp4".

3.6 CAN-meldinger med Raspberry pi

Etter fullført konfigurasjon, kunne gruppen nå sende meldinger over CAN-nettverket.



Figur 3.5: Sender CAN-meldinger fra Raspberry pi og tar imot i PCAn-view



Figur 3.6: Mottar CAN-meldinger i Raspberry pi

Videre skulle Teensyen lese av sensordata, og deretter sende disse via CAN-nettverket, dersom en melding med ID 245 inkommer. Dette resulterte i et velfungerende system. En demonstrasjonsvideo av dette er vedlagt i zip-mappen, og heter TeensytoRaspberry.MOV".

4 Diskusjon

Vi valgte å bruke spenningsregulator med output på 5V. Dette for å forsikre oss om at vi hadde tilstrekkelig spennin til at mikrokontrolleren kunne source spenning. I ettertid ser vi at 3.3V hadde vært tilstrekkelig til å utføre oppgavene. Derfor ville vært fordelaktig å bruke 3.3V, da man helst vil holde spenningen så lav som det elektriske systemet tillater.

Videre støttet vi på store problemer med MBU6050 IMU-en, og valgte etterhvert å benytte oss av en BNO055 IMU. Med denne fikk Arduinoen kontakt med IMU-en på første forsøk. Vi eliminerte muligheten for at akkurat vår utdelte IMU ikke funket ved å prøve 2 andre av samme type, men Teensy fikk heller ikke kontakt med disse. At vi brukte en annen imu har nok ikke hatt noe virkning på resultatene ettersom målet med oppgaven var å få inn verdier fra Teensy til PCAN-view, ikke hvorvidt om disse resultat-verdiene var nøyaktige.

I oppgaven hvor aksellerasjonsdata skulle sendes via CAN-nettverket, var det noe rom for forbedring. De målte verdiene var desimaltall, mens de verdiene som ble motatt var heltall. Dette resulterte i at målingene på 9.81 ble rundet ned til 9. Dette er noe vi ville gått mer i detalj på og forbedret dersom vi hadde hatt mer tid.

Felles for alle delprosjektene har vi blitt påminnet viktigheten av å sjekke koblingene på de fysiske kretsene nøyne. Her har vi flere ganger støtt på problemer, som har vist seg å ha rot i dårlig kontakt mellom komponentene. Dermed har vi lært at man burde sjekke kontakter med multimeter, slik at man eliminerer denne typen feil.

5 Konklusjon

Gjennom prosjektet har gruppen fått god praktisk erfaring og innsikt med hvordan mikrokontrolere skal kobles og programmeres. Varierene typer mikrokortollere med Arduino, Atmel ICE og Raspberry Pi-3 med sine styrker og svakheter har belyst hvordan man må tenke for å velge rett mikrokontroller til spesifikke bruksområder. Gjennom prosjektet har det vært en rød tråd med sentrale elementer innenfor innebygde datasystemer. Nyttigheten av CAN som en rask og sikker overførings- eller kommunikasjonsplattform har blitt belyst, samt innføring og tilvenning av Linux som operativsystem. Gruppene har også møtt på flere diverse blindveier og feilmeldiger gjennom delprosjektene, noe som er forberedende til fremtidig arbeid. All kunnskapen gruppen har vært igjennom på dette semesterprosjektet er viktige elementer for den fremtidige bacheloroppgaven.

Bibliografi

- [1] Atmel. *ATMega168*. URL: <https://www.farnell.com/datasheets/2047865.pdf>. (accessed: 06.10.2021).
- [2] Atmel. *Atmel ICE*. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-ICE_UserGuide.pdf. (accessed: 05.10.2021).
- [3] Prakhar Bhatt. *Timer0 in Atmega16-AVR*. URL: <https://avrgeeks.com/timer0-atmega16-avr/>. (accessed: 06.10.2021).
- [4] *CAN USB Interfaces*. URL: <https://www.computer-solutions.co.uk/gendev/can-usb.html>. (accessed: 06.10.2021).
- [5] Texas Instruments. *LM2931-N Series Low Dropout Regulators*. URL: https://www.ti.com/lit/ds/symlink/lm2931-n.pdf?ts=1634125194297&ref_url=https%253A%252F%252Fwww.google.com%252F. (accessed: 06.10.2021).
- [6] Thomas J. Fellers Kenneth R. Spring og Michael W. Davidson. *Human Vision and Color Perception*. URL: <https://www.olympus-lifescience.com/en/microscope-resource/primer/lightandcolor/humanvisionintro/>. (accessed: 06.10.2021).
- [7] Kingbright. *T-1 3/4 (5mm) SOLID STATE LAMP*. URL: [https://no.mouser.com/datasheet/2/216/L-7113SRD-D\(Ver.16A\)-795224.pdf](https://no.mouser.com/datasheet/2/216/L-7113SRD-D(Ver.16A)-795224.pdf). (accessed: 06.10.2021).
- [8] PJRC. *Teensy® 3.6 Development Board*. URL: <https://www.pjrc.com/store/teensy36.html>. (accessed: 06.10.2021).
- [9] Protostack. *ATMEGA168A PULSE WIDTH MODULATION – PWM*. URL: <https://protostack.com.au/2011/06/atmega168a-pulse-width-modulation-pwm/>. (accessed: 05.10.2021).