

INNEBYGDE DATASYSTEMER PROSJEKT 2021

Christian Fries

Marcus Eide

Tollak Liland

Veileder

Kristian Muri Knausgård

Sammendrag

Denne rapporten er et svar på prosjekt i emnet MAS245 Innebygde datasystemer. Prosjektet var delt opp i flere oppgaver og temaer blant annet programmering av mikrokontrollere ved hjelp av C++, kommunikasjon mellom enheter (CAN-bus, SPI), sensorer og aktuatorer. Det ble også tegnet kretskjema, beregnet komponentverdier og koblet opp enkle mikrokontrollerkretser. Forskjellig software har blitt brukt for å programmere, kommunisere, tegne skjema og lese ut data. Eksempler på dette er: Arduino IDE, Qt Creator, Eagle og Microchip studio. Det er blitt brukt Git versjonskontroll for sikkerhet backup og samarbeid mellom gruppemedlemmer med kildekode.

Innhold

Sammendrag	i
1 Introduksjon	1
2 Teori	2
2.1 Komponenter	2
2.1.1 ATmega 168	2
2.1.2 LM2931 regulator	3
2.1.3 Atmel ICE	3
2.1.4 PEAK Can bus	4
2.1.5 Teensy 3.6 med CAN-bus carrier-board	4
2.1.6 Raspberry Pi 3 model B (RPi)	5
2.1.7 PiCAN	6
2.2 Software	7
2.3 Formler	7
2.3.1 Beregning av motstand for LED diode	7
2.3.2 Beregning av PWM frekvens i ATmega168 chip	8
2.3.3 Beregning av motstand i parallel	8
2.4 SPI	8
2.5 GIT versjonskontroll	8
2.6 Buildroot	9
2.7 CAN-BUS (Controller Area Network)	9
3 Eksperimenter	11
3.1 Koble og programmere ATmega 168	11
3.1.1 Strømforsyning	11
3.1.2 Tilkobling til PC	11

3.2	Hello World program	11
3.3	Soft-blink	13
3.4	CAN kommunikasjon	14
3.4.1	CAN pinnekonfigurasjon SK Pang Teensy 3.6	14
3.4.2	PCAN adapter og D-sub konnektor	14
3.4.3	Sende CAN-melding	16
3.4.4	CAN meldingsmottaker	16
3.4.5	CAN ping-pong	16
3.5	Raspberry, Buildroot og CAN-bus	17
3.5.1	Installasjon og konfigurering	17
3.5.2	Koble til og sende CAN-meldinger	19
3.5.3	Styre servo med IMU via CAN-bus	21
4	Resultater	22
4.1	Strømforsyning	22
4.2	HelloWorld	22
4.3	Avlesning av device signature	23
4.4	Soft-blink	24
4.5	Sende, motta og sende tilbake CAN melding	24
4.6	CAN meldingsmottaker	24
4.7	CAN Ping-pong	25
5	Diskusjon	26
5.1	Strømforsyning	26
5.2	Tilkobling til PC	26
5.3	Hello World program	26
5.4	Soft-blink	26
5.5	Teensy og CAN bus	27
5.5.1	Sende, motta og sende tilbake CAN melding	27
5.5.2	CAN meldingsmottaker	27
5.5.3	CAN Ping-pong	27

5.6	Raspberry, Buildroot og CAN-bus	27
5.6.1	Installasjon og konfigurering	27
5.6.2	Koble til og sende CAN-meldinger	28
5.6.3	Styre servo med IMU via CAN-bus	28
6	Konklusjon	29
7	Appendiks	30
A	Kretsskjema	30
A.1	Teensy 3.6 med CAN bus tranceivere	30
A.2	ATmega168 med LED og motstand	31
A.3	Teensy med CAN, IMU og servo	32
B	Kode	33
B.1	IMU over CAN	33
B.2	Modifisert can_pingpong pakke	35
B.3	Eksempelprogram med en mikrokontrollerpinne med lav verdi	37
B.4	HelloWorld kode	37
B.5	Soft-blink	38
B.6	CAN Bus Tx-Rx-Tx	39
B.7	CAN Message receiver	40
B.8	CAN_Pong.ino	41
B.9	Ball.cpp	47
B.10	Ball.h	48
B.11	Paddle.cpp	49
B.12	Paddle.h	50

1 Introduksjon

Målet for dette prosjektet er først og fremst å knytte sammen teorien fra forelesingene i emnet med praktiske oppgaver. Ved hjelp av labutstyr og fysiske komponenter lærer det oppkobling, tegning og programmering av mikrokontroller-kretser. Man får erfaring med programmering i C++, kryss-kompilering med ulike programmer/kompilatorer, og overføring til forskjellige mikrokontrolere. Prosjektet inneholder også bruk av versjonskontroll og kjennskap til dette.

Noen oppgaver handler om kommunikasjon mellom komponenter og kontrollere. Det blir brukt kommunikasjonsprotokoller som CAN-bus, SPI og I^2C . Andre går ut på design og implementering av enkle spill eller operativsystemer. Denne rapporten beskriver og svarer på de oppgavene som er gitt i prosjektet MAS245 Innebygde datasystemer.

- **Prosjekt 1**

- 1.1 Strømforsyning
- 1.2 Tegning av mikrokontrollerkrets m/ programmeringsheader
- 1.3 Et enkelt "hello world-program"
- 1.4 Lysdiode (LED)
- 1.5 Oppkobling av krets og programmering av mikrokontroller
- 1.6 Soft-blink

- **Prosjekt 2**

- 2.1 Koble opp CAN-kort på Teensy
- 2.2 Sende en CAN melding
- 2.3 MAS245 CAN message receiver
- 2.4 CAN-ping-pong

- **Prosjekt 3**

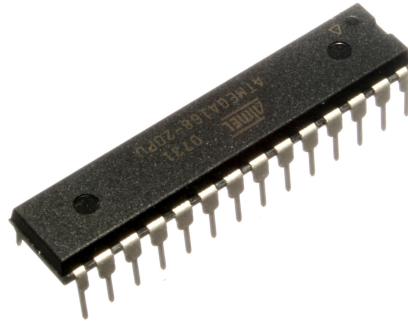
- 3.1 Laste ned og konfigurere buildroot
- 3.2 MAS245-konfigurasjon av buildroot
- 3.4 Koble opp hardware
- 3.5 Sende en CANmelding fra terminal
- 3.6 Fritt spill

2 Teori

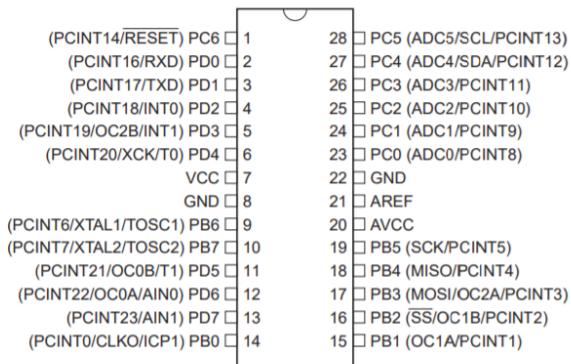
2.1 Komponenter

2.1.1 ATmega 168

Atmega 168 er en mikroknoroller basert på AVR® enhanced RISC arkitektur, som er en familie med 8 bit singel-chip mikrokontrollere, denne har en klokkehastighet på 20 [MHz] ved 4.5[V] - 5.5 [V]. den har også 16 [Kb] flash mine, 512 [bytes] med EEPROM (electrically erasable programmable read-only memory) og 1 [Kb] med SRAM (Static random-access memory). Den har i tillegg en del I/O og perifere funksjoner som 23 programmerbare I/O linjer, Master/slave SPI serial interface og 6 PWM kanaler, for å nevne noen av dem. [3]



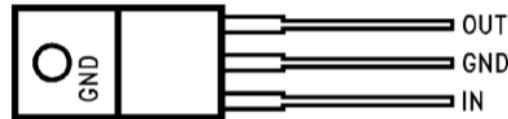
Figur 2.1: ATmega168 [8]



Figur 2.2: ATmega168 Pin layout [3]

2.1.2 LM2931 regulator

LM2931 er en spennings regulator spesielt designet til automobilindustrien med veldig lav stillestrøm (quiescent current) 1 [mA] ved en komponent som trekker 10 [mA] og kan gi ut opp til 100 [mA]. den låste 5 [V] 3 kontakt versjonen gir ut en spenning på 4.75 [V] - 5.25[V] ved input $6 \text{ [V]} \leq V_{IN} \leq 26 \text{ [V]}$, den har ett sikkert temperatur område på $-40 \text{ [C}^{\circ}\text{]} \leq T_J \leq 125 \text{ [C}^{\circ}\text{]}$. [14]



Figur 2.3: TO-220 Pin layout power supply [14]

2.1.3 Atmel ICE

Er et on-chip programmering og debuggings verktøy for Atmel mikrokontrollere, og kan programmere med Jtag og SPI på alle AVR® 8-bit mikrokontrollere, den støtter SPI klokkefrekvens fra 8 [kHz]-5 [MHz]. Debuggeren er også fult kompatibel med Atmel Studio som nå heter Microchip Studio.[4][13]



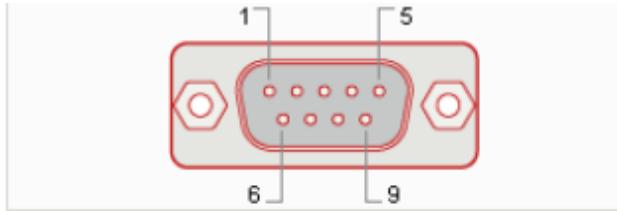
(a) Atmel-ICE kit [4]

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	

(b) Atmel-ICE SPI pinmap (bruker Mini-squid pin)
VTG (Voltege target)=VCC (Voltage Common Collector) på ATMEGA168 [4]

2.1.4 PEAK Can bus

Her brukes det en PCAN-USB FD for å koble 2 pins CAN-bus til USB, de blir brukt original CAN-bus protokoll i henhold til ISO 11898 med 11 [bit] format, som gir en 25 [kbit/s] opp til 1 [Mbit/s] bit rate, der kobles CAN-H og CAN-L til Teensy CAN-bus. breakoutboard 2.6, det blir også implementert en 120 Ω termineringsmotstand mellom CAN-L og CAN-H i hver ende av CAN-bus kretsen [23], da det allerede er en innebygd i breakoutboardet, så må det bare loddes en på PCAN siden.[16]



- 1 not connected / optional +5V
- 2 CAN-L
- 3 GND
- 6 GND
- 7 CAN-H

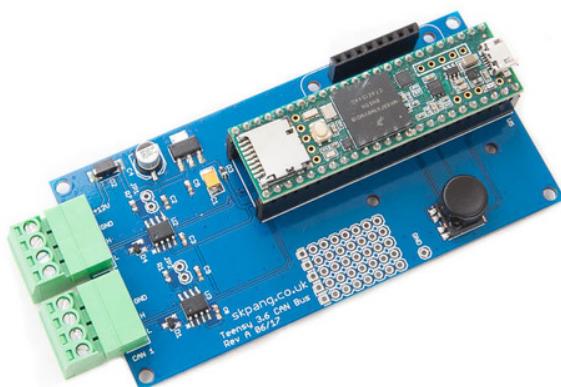
Figur 2.5: PCAN-USB FD D-sub koblingsskjema [16]

2.1.5 Teensy 3.6 med CAN-bus carrier-board

Brettet kommer med innebygd programmeringsmodul og kan kobles til PC via USB. Teensy kan programmeres direkte med C, eller med en arduino add-on (Teensyduino).

CAN-bus carrier-board for Teensy 3.6 inneholder to CAN- transceiver, utgang for OLED skjerm og en joystick. Brettet har også egen spenningsregulator, en 120 Ω termineringsmotstand for begge CAN- bus portene og et spenningsvern.

Selve Teensyen inneholder en CAN- controller men det trengs en CAN- transiver for å sende og motta meldinger fra CAN-bussen defor brukes dette breakoutbrettet som inneholder to MCP2515 CAN-bus transivere. [21]



Figur 2.6: Teensy CAN-bus breakoutboard [9]

Teensy 3.6	
Egenskap	Verdi [Enhett]
ARM Cortex-M4	180 [MHz]
Flash	1024 [kbytes]
RAM	256 [kbytes]
EEPROM	4096 [bytes]
USB device	12 [Mbit/sec]
USB host	480 [Mbit/sec]
digital input/output pins	64 [pins]
PWM output pins	22 [pins]
Analog input pins	25 [pins]
Analog output pins	2 [pins]
Capacitive sense pins	11 [pins]
Serial	6
SPI	3
I2C	4
I2S/TDM digital audio port	1
CAN bus	2
SDIO (4 bit) native SD Card port	1
general purpose DMA channels	32
Cryptographic Acceleration and Random Number Generator	*
RTC for date/time	*

Tabell 2.1: Data fra LED datablad [18]

2.1.6 Raspberry Pi 3 model B (RPi)

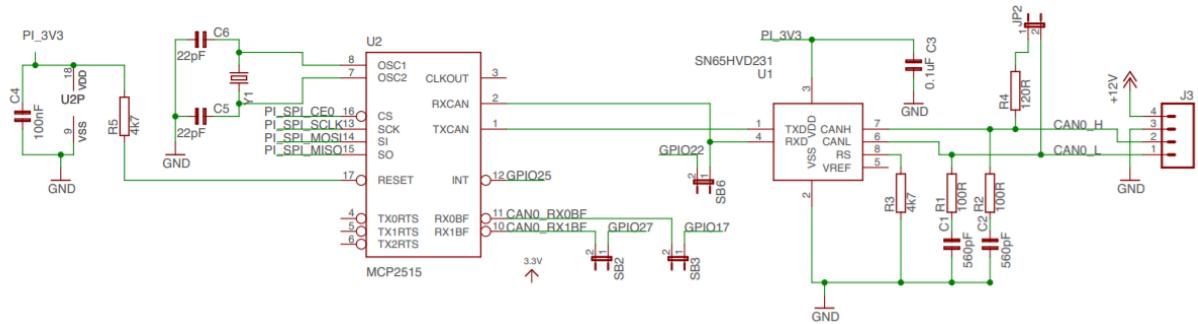
Detter er en singelboard computer, som har en Quad Core (fire kjerner) prosessor med en klokke hastighet på 1.2 [GHz], fra Broadcom av typen BCM2837 [24], med instruksjonsett på 64 [bit], CPU (ARM Cortex-A53) og GPU (VideoCore IV), den er basert på ARM arkitektur. Kortet har 1[GB] RAM. RPI-en har også Innebygd BCM43438 tråløst LAN, Bluetooth lav Energi (BLE), 100 [Mbit/s] Base Ethernet plugg. Av I/O har den en 40-pin forlenget GPIO, 4 USB 2 porter, 4 Pol stereo utgang og sammensatte video port. En full størrelse HDMI utgang for bruk av skjerm, og en micro SD-kort leser for å lagre operasjonssystem og annen data. Ett SD-kort brukes da som en harddisk.[17]



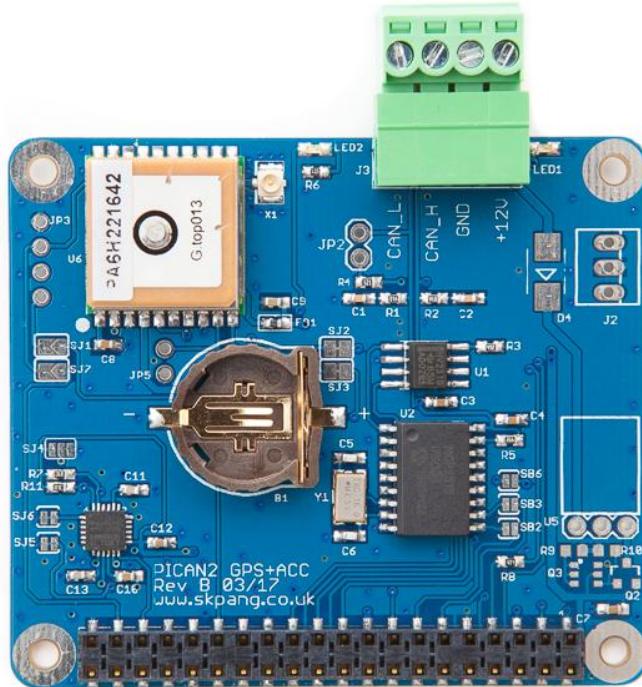
Figur 2.7: Raspberry Pi 3 moddel B [17]

2.1.7 PiCAN

Dette brettet gir Raspberry Pi-en CAN, GPS og IMU egenskaper, ved å bruke en MCP2515 CAN mikrochip, MTK3339 chipset modul og gyro, akselerometer av typen MPU-6050, den har også muligheter for et onboard batteri for å opprettholde nøyaktigheten til relativ klokken. I CAN kretsen har den en innebygd 120Ω termineringsmotstand som blir integrert ved å koble sammen to pinner(1 og 2 på JP2 pluggen) (kapittel 2.8). For å kommunisere med Raspberry Pi-en så bruker den høy hastighets SPI (10 [MHz]) (kapittel 2.4), mens CAN er av typen CAN v2.0B med en hastighet på 1 [Mb/s] [19]



Figur 2.8: PiCAN CAN krets[20]



Figur 2.9: PiCAN Pi hat kort [19]

2.2 Software

Autodesk Eagle til å tegne kretsskjemaer

Microchip Studio for å programere microkontrolleren

Arduino IDE

Teensy utvidelser fra PJRC

PCAN-View

2.3 Formler

2.3.1 Beregning av motstand for LED diode

$$R = \frac{(V_{in} - V_{LED})}{I_{LED}} \quad (2.1)$$

2.3.2 Beregning av PWM frekvens i ATmega168 chip

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{clk\ I/O}}{N \cdot 256}$$

The N variable represents the prescale factor (1, 8, 64, 256, or 1024).

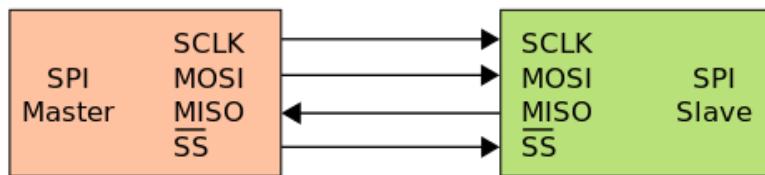
Figur 2.10: Formel for å regne ut PWM frekvens, fra side 109 i datablad [3].

2.3.3 Beregning av motstand i parallell

$$R = \left(\frac{1}{R_1} + \frac{1}{R_2} + \dots + \frac{1}{R_n} \right)^{-1} \quad (2.2)$$

2.4 SPI

Serial Peripheral Interface (SPI) er et kommunikasjonsgrensesnitt brukt for å kommunisere over kortere avstander i innebygde systemer. Kommunikasjon mellom sensorer som akselerometer, eller SD kort og microcontroller er typiske bruksområder for SPI protokoll. Noe som kan være en ulempe med SPI, er at man trenger en chip select (CS) pinne per SPI enhet. Alle CS pinner er til vanlig høye, og når man ønsker å snakke til en gitt SPI enhet, så setter man CS pinnen for den enheten lav. Det som derimot er positivt, er at SPI har høy overføringsfart, i motsetning til for eksempel I2C. Seriell kommunikasjon betyr at data blir sendt bit for bit, i motsetning til parallell kommunikasjon der flere bit kan sendnes samtidig. SPI er synkron, som vil si at data blir sendt basert på klokkesignal. Kommunikasjonen kan skje begge veier i et master/slave system (full duplex). Det blir brukt fire ledninger mellom en master og en slave enhet. SCLK(SCK) for klokkesignal, MOSI står for serial data input og MISO for serial data output. SS for chip select. Denne er for å velge hvilken slaveenhet det kommuniseres med visst det er flere enheter koblet sammen. Normalt satt som lav. [26]



Figur 2.11: SPI kobling [7]

2.5 GIT versjonskontroll

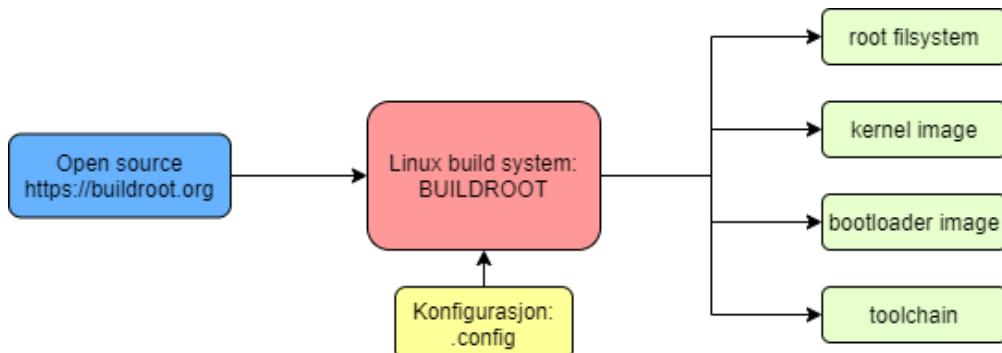
Git er et gratis og åpen-kildekode, distribuert versjons kontroll system utviklet av Linus Torvalds ut ifra Linux kjernen. Det som Git har som ikke andre klient-server systemer ikke har er at hver datamaskin og da bruker er et fullt repository med hele fil historie og versjonskontroll. I tillegg til denne egenskapen kan man legge til en sky lagrings tjenester som foreksempel GitHub eller Gitlab som er optimalisert for Git, dette gir ekstra sikkerhet mot tap av data.

Alle commiter til det lokale repositoriet blir gitt en SHA256 (Secure Hash Algorithm 256 [Bit]) kryptert versjons id basert på innholdet til filen som gjør det umulig å endre filen uten å endre id-en.

Git er også veldig brukervennlig på tvers av de vanligste operativsystemene (OS), og gir god flyt når prosjekter er avhengig av flere OS-er. Git gir også muligheten til å lage nye grener ut ifra hovedstammen eller andre grener, disse lar seg senere sette sammen med hovedstammen eller grener igjen. Det som er bra med grener, er at ting kan testes uten at det påvirker hovedstammen samtidig som de er reverserbare. Mye brukte kommandoer i Git er (git add, git commit, git push og git pull) add legger klar fil for commit, commit legger filer i det lokale repositoriet, push legger filen til at andre kan hente til repositoriene, og pull henter ned filer fra andres repositori. Andre kommandoer som git status og git help, disse gir statusen på dine filer i forhold til andre og hjelpe som er en beskrivelse av kommandoene.

2.6 Buildroot

Buildroot er et verktøy for å bygge og krysskompile et komplett Linux-system for flere typer innebygde datasystemer. Vertøyet kan generere en toolchain, et filesystem, et Linux kjerne bilde og en bootloader satt sammen. Det kan også bygge programmene enkeltvis som for eksempel ved kun å bygge filesystemet. Buildroot passer godt til å bruke sammen med Raspberry 3 sin ARM prosessor som blir brukt i dette prosjektet, men støtter veldig mange forskjellige andre prosessorer.



Figur 2.12: Byggesystem

Buildroot er designet for å kjøres på Linux siden det baserer seg på flere innebygde funksjoner i GNU/Linux, men kan også brukes med andre operativsystemer. [6]

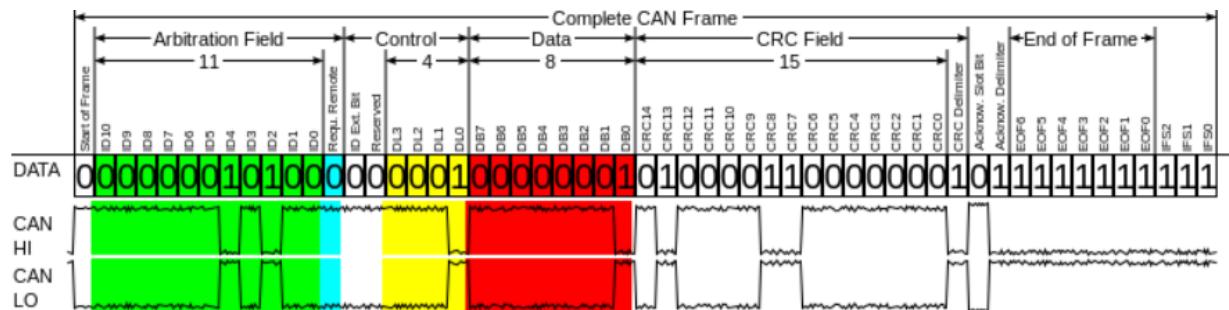
2.7 CAN-BUS (Controller Area Network)

CAN-BUS er en protokoll designet for å la mikrokontrollere og enheter kommunisere med hverandre. Den ble utviklet av Robert Bosch GmbH og utgitt i 1986, og utviklet for kjøretøysindustrien. Grunnet at kjøretøyene begynte å ha en overdreven mengde med kabelstrekk og da mye vekt i kobber. Dette på grunn av at det begynte å bli mange sensorer og logikk i kjøretøyene, den første serie produserte bilen med et CAN-basert multiplex kabel system var den suksessfulle Mercedes-Benz W140 men det er også brukt i mange andre bransjer. Senere kom CAN-BUS 2.0A 2.13 som er standard for både 11-bit (identifiserer) åpner for 2048 unike iD-er og da like mange noder, CAN-BUS 2.0B med 29-bit ID-er (identifiserer) åpner for 536870912 unike iD-er og da like mange noder, ved hjelp av disse ID-ene så kan man implementere en kø med prioriteringer av funksjoner, som foreksempl prioritere kritiske funksjoner og undertrykke mindre kritiske. Typisk eksempel er å prioritere airbag over setevarmeren.

I denne oppgaven brukes ISO 11898-2, eller også kalt high-speed CAN med en hastighet på opp til 1 [Mbit/s] ved bruk av CAN-FD(Fleksibel Data-Rate) så kan man oppnå opp til 5 [Mbit/s]. Men hastigheten avhenger av kabel lengden. [27]

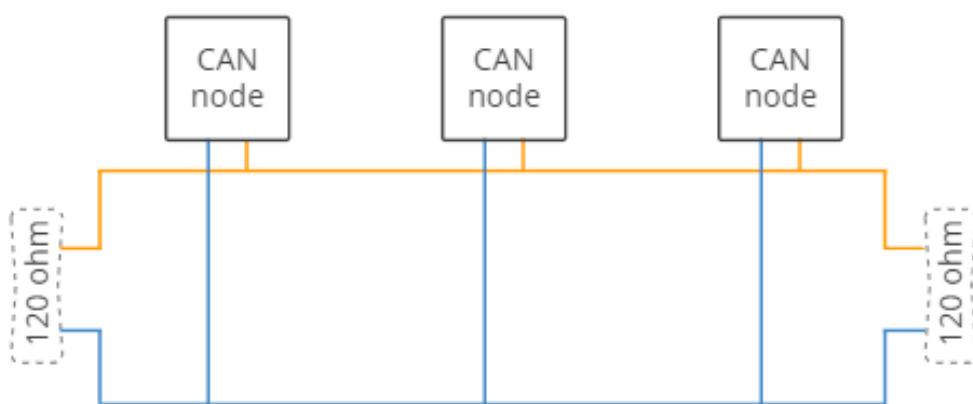
Hastigheter med vanlig CAN 2.0A så er [23]

- 1 [Mbit/s] på max. 40 m
- 500 [kBit/s] på max. 100 m
- 125 [kBit/s] på max. 500 m



Figur 2.13: Hvordan en CAN 2.0A melding er oppbygd [27]

Grunnen til at man må benytte en 120 [Ω] termineringsmotstand i begge ender av et CAN-BUS system 2.14 er at man skal unngå å sprette de sendte signalene tilbake. Dette problemet forsterkes, jo lengre kablene er og samt bit raten til de sendte signalene. Disse reflekterte signalene ville ha tilført mye støy til signalene og potensielt ødelegge signaler helt. Tvinning av CAN-Bus kablene er også en anbefalt ting å gjøre da det blir indusert spenning fra den ene lederen over i den andre, og dette kan på lik måte som tilbake sprett, tilføre støy. [10]



Figur 2.14: Hvordan en CAN-BUS krets er bygd opp [10]

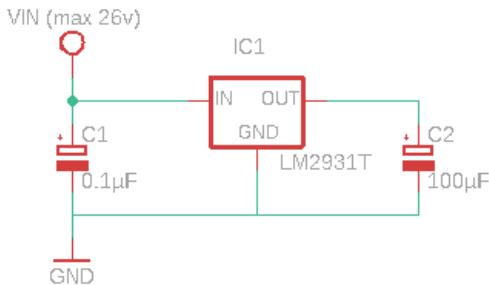
Når det gjelder levetiden til CAN-BUS protokollen så kommer den til å være i bruk en stund til, men det finnes allerede og er i bruk erstatninger sånn som Avionics Full-Duplex Switched Ethernet (AFDX), også kalt ARINC 664. Det er ett protokoll system som allerede er i bruk i avionikkindustrien og hos fly produsenter. ARINC 664 har blitt utviklet av det Franske/Europeiske selskapet Airbus Group SE. Det er basert på det kommersielle 10/100Mbit switched Ethernet. Som gir deterministisk timing og redundancy management, sikker og trofast kommunikasjon av kritisk data. Dette gjør det veldig brukbart for Flay bay wier sytemer som dagens fly moddeler bruker, da feil ved dette ville ha gitt katastrofale følger. En melding sendt med ARINC 664 har også en mye større frame fra 64 til 1518 bytes. [1][25]

3 Eksperimenter

3.1 Koble og programmere ATmega 168

3.1.1 Strømforsyning

I første omgang ble det valgt en 5 [V] strømforsyning med tanke på å koble til en blå led med høyt spenningsfall over seg, i tillegg til å kunne kjøre mikrokontrolleren (MCU) med høyere klokkehastighet. For å gjevne ut spenningstopper (spikes), og stabilisere spenning inn på MCU ble det montert to kondensatorer.



Figur 3.1: Strømforsyning kretsskjema

Det ble brukt en 0.1 [μF] tantalum- og en 100 [μF] aliminium- elektrolytt polarisert kondensator siden disse var tilgjengelig.

3.1.2 Tilkobling til PC

For å kunne programmere MCU ble denne koblet til en Atmel ICE debugger 2.4b 2.4a. Ved å utnytte brukerguiden til programmereren, [5] kobles mikrokontroller og ICE sammen for SPI-basert programmering se 2.4, og koblingsskjema: 3.1

3.2 Hello World program

Det ble opprettet et C++ prosjekt i Microchip Studio, og det ble verifisert at dette lot seg kompile og laste opp til MCU via Atmel-ICE. Etter dette, ble det skrevet kode for å styre en mikrokontrollerpinne som utgang. I første omgang ble denne pinnen satt til lav, med kode i appendiks B.3. I koden kan man se at klokkefrekvensen er satt til 8 [MHz]. For at dette skal være mulig, må man skru av et "fuse bit" i "Device Programming" seksjonen av Microchip Studio. Dette er fordi at i følge databladet [3], i figur 3.2, så ser man at som standard er MCU levert med CKDIV8 fuse bit programmert. Det betyr at 8 [MHz] klokken er delt med 8, som resulterer i 1 [MHz] klokkefrekvens som standard.

9.6 Calibrated internal RC oscillator

By default, the internal RC oscillator provides an approximate 8.0MHz clock. Though voltage and temperature dependent, this clock can be very accurately calibrated by the user. The device is shipped with the CKDIV8 fuse programmed. See “[System clock prescaler](#)” on page 43 for more details.

Figur 3.2: 8 [MHz] fuse bit i datablad

Tool	Device	Interface	Device signature	Target Voltage
Atmel-ICE [Disconnected]	ATmega168	ISP	Apply	0x1E9406 Read 4.9 V Read
Interface settings	Fuse Name			
Tool information	<input checked="" type="checkbox"/> HIGH.EESAVE			
Device information	<input checked="" type="checkbox"/> HIGH.BODLEVEL			
Oscillator calibration	<input checked="" type="checkbox"/> LOW.CKDIV8			
Miscellaneous	<input checked="" type="checkbox"/> LOW.CKOUT			

Figur 3.3: 8 [MHz] fuse bit i Device Programming seksjonen

Når man skrur av “fuse bitet” **LOW.CKDIV8** i “Device Programming” vinduet, så kan man sette klokkefrekvensen til 8 [MHz] i koden med `#define F_CPU 8000000UL`.

Videre ble det koblet opp en LED direkte til spenningsregulatoren. Hensikten med dette var at denne lysdioden skulle fungere som en “power on” indikator.

Etter at oppsettet med lysdioden og motstanden var testet direkte på spenningsregulatoren, ble LED og motstand flyttet over til pinne D6. På denne måten kunne lysdioden styres ved hjelp av MCU. Lysdioden ble koblet til MCU slik at den “sourcer” strøm fra den. Grunnen til at det ble koblet slik, var at i følge databladet kapittel 1.1.6 (figur 3.4), står det at PORT D har symmetrisk karakteristikk på inn/utgang for sink og source. I tillegg ble det sjekket opp hvor mye strøm som kan leveres fra IO pinnene. Fra figur 3.5 kan man lese at “DC current per I/O pin” er 40.0 [mA], som er nok til å levere 20.0 [mA] til lysdioden. Kretsskjema som viser hvordan dette ble koblet finnes i appendiks A.2.

1.1.6 Port D (PD7:0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up

Figur 3.4: PORTD register sink og source egenskaper [3]

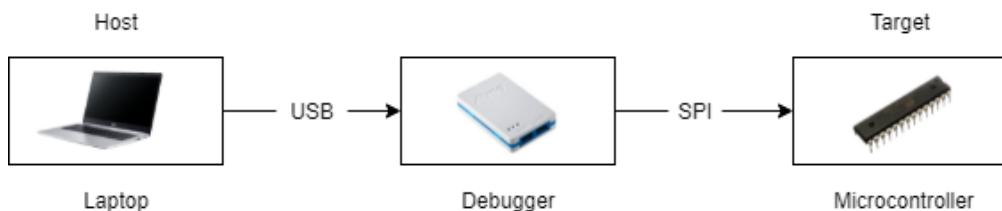
29.1 Absolute maximum ratings*

Operating temperature	-55°C to +125°C
Storage temperature	-65°C to +150°C
Voltage on any pin except <u>RESET</u> with respect to ground	-0.5V to $V_{CC}+0.5V$
Voltage on <u>RESET</u> with respect to ground	-0.5V to +13.0V
Maximum operating voltage	6.0V
DC current per I/O pin.	40.0mA
DC current V_{CC} and GND pins	200.0mA

Figur 3.5: Absolute Max Electrical characteristics [3]

Videre ble Atmel ICE programmerer/debugger koblet til ATmega168 MCU og strøm ble skrudd på etter klarsignal fra faglærer. Riktig enhet ble valgt i Microchip Studio, og “device signature” ble lest ut.

Deretter ble MCU programmert via “Memories” seksjonen i “Device Programming” vinduet. Det som skjer her er at minnetområdet som er allokeret til selve programmet på MCU blir vasket ut før selve programmeringsprosessen begynner. Bootloader seksjonen står urørt. Når minnet er vasket bort, begynner selve programmeringen av MCU. Programmeringsverktøyet leser en .elf fil (“Executable and Linkable Format”), [12]. Denne filen inneholder informasjon om debugging, binærinformasjon som skal skrives til minne, og annet som er relevant for verktøyet. Når debuggingsverktøyet leser .elf filen, blir binærinformasjonen skrevet til Flash minnet i MCU, slik at programmet begynner å kjøre på MCU når det kobles til strøm. Det er en bootloader på MCU som legger binærinformasjonen på riktig plass i minnet.



Figur 3.6: Programmeringsoppsett

Når vi kompilerer i Microchip Studio, utføres det en krysskompliling. Det betyr at selve kompileingen til maskinkode foregår på for eksempel en laptop eller stasjonær maskin med Intel prosessor, selv om denne maskinkoden skal kjøres på en mikrokontroller.

3.3 Soft-blink

Videre ble Hello World programmet utvidet til at LED-en skulle dimmes mykt opp og ned. I følge databladet til LED-en, kan maksimalt strømtrekk bli opptil 100[mA] ved PWM styring, altså fem ganger større enn konstantstrømmen på 20[mA]. Siden I/O pinnen ikke kan levere mer enn 40[mA], må strømtrekket aldri bli større enn det. Dette reguleres med en motstand.

3.4 CAN kommunikasjon

3.4.1 CAN pinnekonfigurasjon SK Pang Teensy 3.6

I krestskjema som ligger i appendiks A.1, ser man CAN tranceivere er koblet til MCU på følgende pinner:

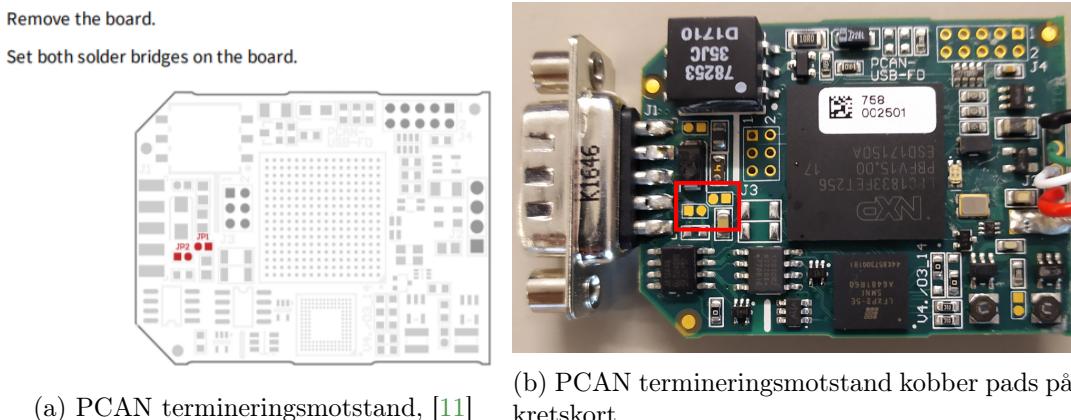
Pinne	I/O
CAN0 TX	3
CAN0 RX	4
CAN1 TX	33
CAN1 RX	34

Tabell 3.1: CAN tranceiver pinout

3.4.2 PCAN adapter og D-sub konnektor

For å bruke PCAN adapteren med Teensy kortet og Raspberry Pi, var det nødvendig å lodde på kabler på en D-sub konnektor som passer i PCAN adapteren. Dette ble gjort i henhold til koblingsskjema i figur 2.5. Det var kun CAN-L og CAN-H som ble benyttet, da det er potensiale mellom L og H som måles og dermed er det ikke behov for å utlikne jordpotensiale. Det ble i tillegg målt at det ikke var noen kortslutning mellom CAN-L, CAN-H, Vcc eller GND.

I selve PCAN adapteren er det mulighet for å legge til termineringsmotstand ved å lodde sammen JP1 og JP2 på kretskortet inni plastdekselet. Dette var ikke gjort, og dermed ble dette lodde direkte på D-sub konnektoren. På grunn av mangel på 120Ω motstander, ble 1 stk 20Ω og 1 stk 100Ω motstand loddet i serie, slik at totalen ble 120Ω i figur 3.8.



Figur 3.7: Hvordan aktivere termineringsmotstand

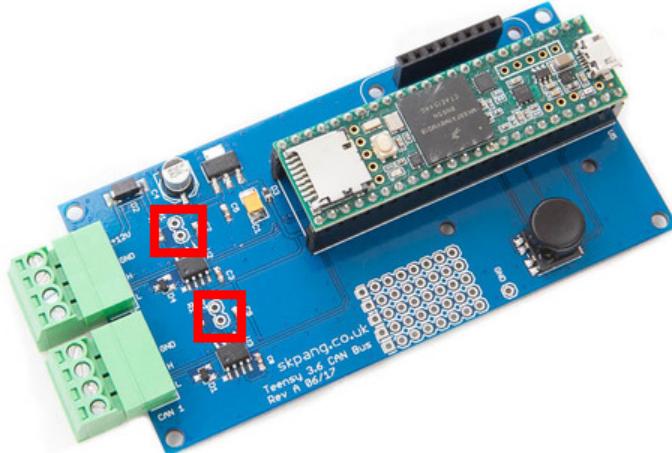


Figur 3.8: D-sub konnektor terminering

Carrier kortet har også intern termineringsmotstand, men på dette er det to hull i kretskortet som man kan kortslutte for at motstanden skal kobles inn.

Hver av disse innebygde termineringsmotstandene er på 120Ω og siden disse er i parallel, så resulterer det i 60Ω over terminalene CAN-L og CAN-H, beregnet på denne måten:

$$R_{terminering} = \left(\frac{1}{120\Omega} + \frac{1}{120\Omega} \right)^{-1} = 60\Omega \quad (3.1)$$



Figur 3.9: Carrier kort intern terminering [9]

3.4.3 Sende CAN-melding

For å sende CAN meldinger mellom de to kanalene på kortet med FlexCAN biblioteket, må man først opprette en meldings struktur av typen `CAN_message_t msg;` (lager en variabel med denne datatypen, som kalles for msg). Deretter må man definere baud raten, samt starte den CAN bus porten man ønsker å bruke, ved hjelp av `Can0.begin(baudrate);` (i dette tilfellet CAN bus port 0).

Videre må man sette en ID til CAN meldingen som skal sendes, ved hjelp av `msg.id = 0x00;`. Meldinger med lav ID har høyere prioritet enn meldinger med høy ID. Så kan man fylle databufferen med data som man vil sende over CAN bussen, ved hjelp av `msg.buf[0] = data;`, hvor data kan være heltall opp til 8 [bit]. Datarammen som sendes kan være inntil 8 [byte] totalt.

Til sist kan man sende datarammen ved å bruke `Can0.write(msg);`. En dataframe består av 8 byte med data. Når CAN meldinger skal mottas, må man først sjekke om det er noen CAN meldinger tilgjengelig med `Can0.available()`. Dersom denne funksjonene returnerer en verdi størren enn 0, så kan meldingen leses inn ved hjelp av `Can0.read(returMsg);`.

Kode for å sende melding, motta melding og så sende samme melding tilbake til mottaker ligger i appendiks B.6.

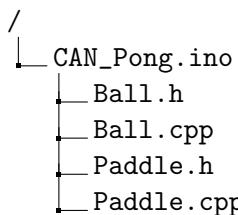
3.4.4 CAN meldingsmottaker

Det ble programmert en CAN meldingsmottaker, som sender CAN meldinger på Can0, og mottar med Can1 (hvert sekund). ID til avsender økes for hver iterasjon, slik at man enkelt kunne verifisere at systemet virker. De mottatte meldingene vises deretter på OLED skjermen. Koden for dette finnes i appendiks B.7.

3.4.5 CAN ping-pong

Videre ble det programmert ping-pong over CAN fra bunn. Første versjon av spillet var en “enkelt-spiller” variant, hvor hver gang ballen endrer retning, så styrer spilleren den andre platen. Koden for denne versjonen er ikke lagt med i appendiks, men finnes i GitHub repository for prosjektet.¹

Videre ble denne versjonen videreutviklet til flerspiller over CAN bus, delt opp i klasser, og er derfor delt inn i flere kodesnutter, disse ligger i appendiks B.8, B.9, B.10, B.11 og B.12. Strukturen på filene i ping-pong spillet ser slik ut:



Spillet er laget slik at to carrier-kort kobles sammen med Can0. Deretter lastes samme kode opp til begge kort, og den første som trykker inn joystick knappen blir tildelt master-rolle. Motparten blir da tildelt slave-rolle automatisk. Deretter starter spillet, og så kan man spille mot hverandre. “Paddle” brikkene (som er 20 piksler høye) flyttes opp og ned med joystick, og når ballen kommer utenfor banen, blir det tildelt poeng til motsatt side av der ballen kom utenfor. Deretter oppdateres poengsummen på toppen av skjermen.

¹https://github.com/eidetech/MAS245Prosjekt/tree/main/Code/Prosjekt%202/Pong_singleplayer

Programmet er skrevet slik at all spillokk kjører hos master. Slaven abонnerer på x og y-data fra ballen, y-data fra "paddle" og poengsummene. Disse verdiene oppdateres hvert 10 millisekund (styres av variabelen updateGame). Dermed får spillet en oppdateringsrate på 100Hz, basert på følgende utregning:

$$10ms = 0.01s \quad (3.2)$$

$$Hz = \frac{1}{s} \quad (3.3)$$

$$\text{Oppdateringsrate} = \frac{1}{0.01s} = 100Hz \quad (3.4)$$

CAN meldinger med informasjon om "paddle" har ID 20 + gruppenummer, dermed 22 for denne gruppen. CAN meldinger med informasjon om ballposisjon har ID 50 + gruppenummer, altså 52. I og med at samme program kjører på begge Teensyene ved spilling, så var det ikke behov for å avtale datastrukturer med motpart.

3.5 Raspberry, Buildroot og CAN-bus

3.5.1 Installasjon og konfigurering

Det første som ble gjort var å laste ned og pakke ut Buildroot. Filene ble funnet hos <https://buildroot.org>, og pakket ut. For å konfigurere byggeoppsettet brukes Linux-verktøyet make. Det anbefales og ikke kjøre disse kommandoene med superbruker (sudo), for å unngå at pakker oppfører seg dårlig under kompilering og installering.

I mappen buildroot ble pakket ut i ble første kommando kjørt:

```
$ make menuconfig
```

For å finne en liste med tilgjengelige forhåndsdefinerte konfigurasjoner kjøres:

```
$ make list-defconfigs
```

Her finnes rett config som passer til raspberry 3. Denne kjøres:

```
$ make raspberrypi3_defconfig
```

I make leses det ut at rett prosessorarkiterktur til Raspberry pi 3 er valgt: ARM cortex-A53. Siden denne standard config trenger skjerm og eller uart tilkobling til terminal brukes heller ferdig tildelt konfigurasjon.

Dette oppsettet er konfigurert med nødvendige CAN-bus-pakker aktivert og det er satt opp en SSH-server slik at det kan logges inn på Raspberry Pi fra egen pc over nettverk.

I neste steg ble tildelt mas245 prosjektpakke med ferdige innstillinger lastet ned og ordnet i ønsket mappestruktur.

```

buildroot-project
├── buildroot
├── builroot-output
│   └── .config
├── external
│   └── package
│       └── mas245
│           ├── can_pingpong
│           │   ├── can_pingpong.mk
│           │   ├── Config.in
│           │   └── Readme.md
│           ├── config.in
│           └── mas245.mk
│           └── .DS_Store
│           └── .DS_Store
├── Config.in
└── external.desc
└── external.mk
mas245
└── can_pingpong
    ├── .git*
    ├── .gitignore
    ├── can_pingpong.pro
    ├── can_pingpong.pro.user
    └── main.cpp
mas245-rootfs-overlay
└── etc
    └── network
        └── interfaces
        └── .DS_Store
        └── .DS_Store
└── 2021_mas245_rpi3_can_qt5qmake_staticip_iproute2.config
└── README.txt

```

For å sette rett mappe hvor output lagres etter bygging

```
0=$PWD/builroot-output/
```

I denne mappen legges tildelt .config fil, og SD-kort-imagefilen som blir bygget lagres her.

For å sette mappe utenfor buildroot treeet hvor egne prosjekt-relaterte filer befinner seg:

```
BR2_EXTERNAL=$PWD/external
```

Her defineres can_pingpong-pakken.

Siste kommando sier hvor selve Buildroot finnes:

```
-C $PWD/buildroot
```

I terminal kjøres:

```
$ make O=$PWD/buildroot-output/BR2_EXTERNAL=$PWD/external
```

```
-C $PWD/buildroot menuconfig
```

\$PWD betyr “print working directory” og benyttes for å skrive hele filstien fram til den mappen man står i.

I steg nummer tre kopieres og legges tildelt konfigurasjonfil inn i buildroot-output mappe med kommandoen:

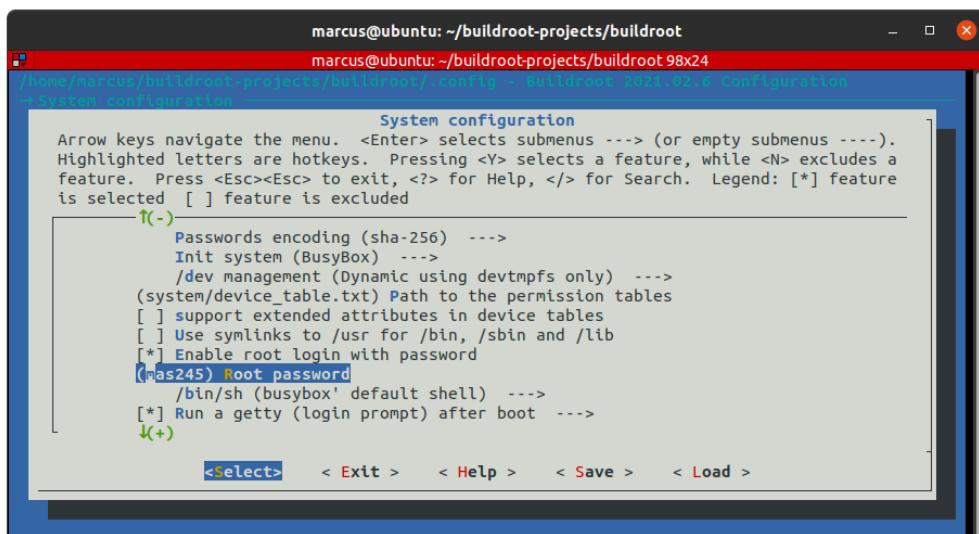
```
cp 2021_mas245_rpi3_can_qt5qmake_staticip_iproute2.config buildroot-output/.config
```

Vi kjører `$ make menuconfig` igjen og sjekker at can_pingpong er valgt, i tillegg til at rett filsti hvor ny .config filen skal lagres. I dette tilfellet `buildroot-output/`

I `buildroot-output/` kjøres `$ make` og imagefilen vi skal montere på sd- kortet blir kompilert basert .config filen lagret her. Under denne prosessen generes flere feilmeldinger og byggingen stoppet. Flere innstillinger måtte endres, blant annet var kernel versjon feil. Tilslutt ble hele byggingen gjennomført og resultatet ble montert på sd-kortet.

3.5.2 Koble til og sende CAN-meldinger

Nå ble det gjort forsøk på å logge inn på Raspberry Pi ved hjelp av ssh via ethernet. Det ble satt statisk IP på hostmaskin (PC), men feil ip-adresse hadde blitt oppgitt og kontakt med RPi ble ikke opprettet. IP settings fil ble funnet i `/mas245-rootfs-overlay/etc/network`. Her ble interfaces-filen endret og ønsket IP satt. Her er det viktig at rett subnett også er riktig. Deretter logget vi på RPi med rett IP fra terminalen på host-maskinen. Root password blir satt i menuconfig under fanen System configuration.



Figur 3.10: root password i menuconfig i terminal

```

login as: root
root@192.168.245.245's password:
# ifconfig
eth0      Link encap:Ethernet HWaddr B8:27:EB:CE:DF:08
          inet addr:192.168.245.245 Bcast:0.0.0.0  Mask:255.255.255.0

```

Figur 3.11: Login og ifconfig

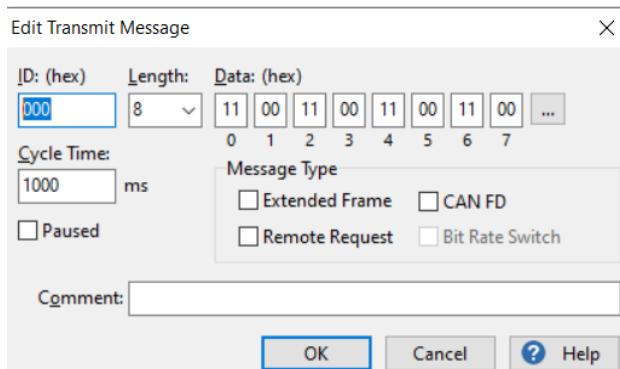
I ifconfig har ikke RPi kontakt med CAN- transiveren på pihatten og derfor må kommandoene `$ modprobe spi_bcm2835` og `$ modprobe bcm2835` kjøres. Dette er for å laste kjernemodulene for SPI og CAN-transiveren på hatten. Neste steg for å konfigurere CAN- grensesnittet var å kjøre:

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

For å printe ut alle CAN-meldinger fra CAN-nettverket i terminalen kjøres kommandoen:

```
$ candump can0
```

Nå ble meldinger sendt og mottatt i PCAN-view fra PC. Denne var koblet til CAN-nettverket vårt med Peak-konverter se 2.1.4.



Figur 3.12: PCAN- view melding

CAN-ID	Type	Length	Data	Cycle Time	Count	Trigger Time	Comment
000h		8	11 00 11 00 11 00 11 00	1000	41		

Figur 3.13: CAN melding sendt fra PCAN- view

```

# candump can0
can0 000 [8] 11 00 11 00 11 00 11 00
can0 000 [8] 11 00 11 00 11 00 11 00
can0 000 [8] 11 00 11 00 11 00 11 00
can0 000 [8] 11 00 11 00 11 00 11 00

```

Figur 3.14: CAN-melding lest i terminal RPI

```

# cansend can0 245#23.40.00.00.FF.FF.FF.00
#

```

Figur 3.15: CAN melding sendt fra RPI

CAN-ID	Type	Length	Data	Cycle Time	Count
245h		8	23 40 00 00 FF FF FF 00		1

Figur 3.16: CAN melding mottatt fra RPI

3.5.3 Styre servo med IMU via CAN-bus

I denne delen av prosjektet ble sensordata lest ut fra en IMU koblet til Teensy. Dataene ble deretter sendt til RPi over CAN-bus-nettverk. På RPi ble vedlagt kode can_pingpong brukt til å gange meldingen med 0x02 for å vise at det ble gjort endringer C++ programmet før de ble sendt tilbake til Teensy. De returnerte dataene ble i neste steg brukt til å styre en liten RC-servo.

```
sensors_event_t a, g, temp;
mpu.getEvent(&a, &g, &temp);

/*Leser IMUData*/
imuReading = a.acceleration.x*10;

/*Sender IMUData ut på CAN-bus*/
msg.buf[0] = imuReading;
Can0.write(msg);
```

Listing 3.1: IMU over CAN

IMU “breakoutboard” GY-521 [15] med MPU6050 chip ble koblet til Teensy. Akselerasjon i x-akse ble hentet ut over I2C med hjelp av arduino-bibliotek fra Adafruit [2]. Data fra IMU ble lagret som float og ble derfor ganget opp med 10 for å kunne sende int over CAN-bussen. Ved hjelp av arduino bilbloteket servo.h ble data sendt til servoen.

```
/*Konverterer IMUdata til Servodata*/
int val = map(inMsg.buf[0], 0,200, 0, 180);

/*Lager "dødgang" for IMU*/
if(inMsg.buf[0] > 80 && inMsg.buf[0] < 120)
{
    /*Setter servo i ro i midtstilling*/
    servo.write(92);
}
else
{
    servo.write(val);
}
```

Listing 3.2: Servostyring

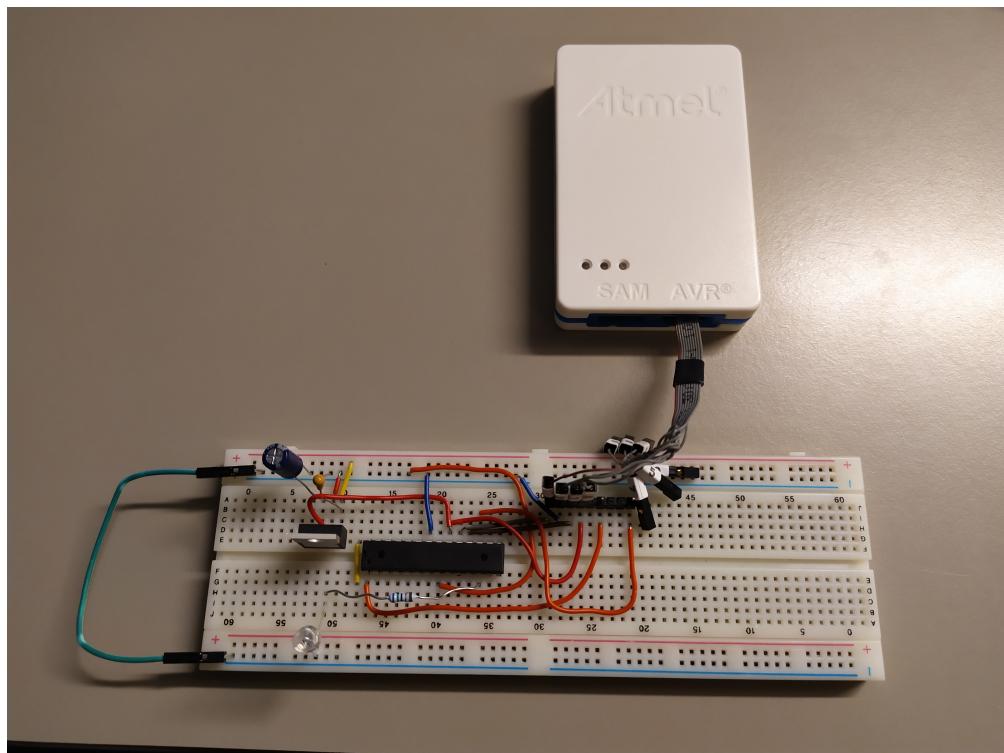
Den redigerte cpp filen i can_pingpong pakken ligger i appendiks B.2.

4 Resultater

I dette kapittelet presenteres funn og resultater fra de tre delprosjektene.

4.1 Strømforsyning

Ut fra strømforsyningen som ble valgt til ATmega168, ble det målt cirka 5 [V], som forventet. ATmega168 ble koblet opp og “Hello World” program ble skrevet til MCU med Atmel ICE.



Figur 4.1: Programmeringsoppsett

4.2 HelloWorld

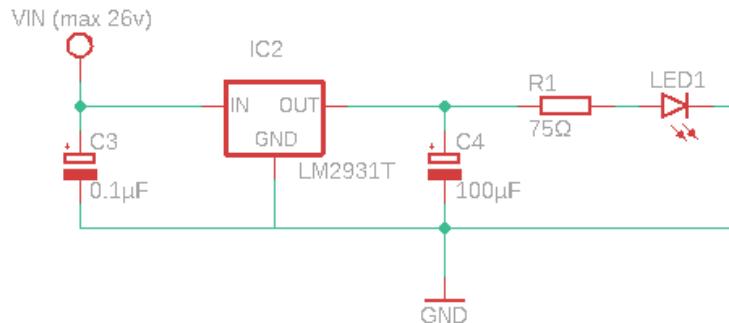
Videre ble det satt inn en lysdiode som skulle blinke. For at ikke lysdioden skulle brenne opp, var det nødvendig å sette inn en strømbegrensende motstand i serie med dioden. Beregningene for denne motstanden ble gjort ved hjelp av formel 4.2 og informasjon fra datablad til LED [22]. Fra databladet kan man lese følgende viktige data:

Data	Verdi (Eenhet)
Forward Current Typical	20 [mA]
Voltage Max	3.5 [V]

Tabell 4.1: Data fra LED datablad [22]

$$R = \frac{(5[V] - 3.5[V])}{0.02[A]} = 75[\Omega] \quad (4.1)$$

Dermed ble det valgt en $75 [\Omega]$ motstand til å være i serie med lysdioden. LED og motstand ble videre koblet direkte på spenningsregulatoren, som i kretsskjema i figur 4.2.



Figur 4.2: LED kretsskjema

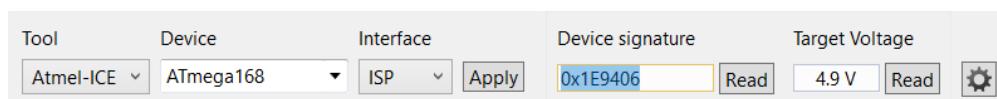
Styring av lysdioden ble gjort med HelloWorld koden i B.4.

4.3 Avlesning av device signature

Table 28-8. Device ID

Part	Signature bytes address		
	0x000	0x001	0x002
ATmega48	0x1E	0x92	0x05
ATmega88	0x1E	0x93	0x0A
ATmega168	0x1E	0x94	0x06

Figur 4.3: Device signature fra datablad



Figur 4.4: Device signature avlest fra MCU i Microchip Studio

Ved å sammenlikne byte adressen til ATmega168 fra tabell 28-8 i datablad til MCU, ser man at "device signature" stemmer overens.

4.4 Soft-blink

I eksperimentet ble det brukt samme motstand (75Ω) som i HelloWorld, men dersom man hadde ønsket høyere lysstyrke på LED-en, så kunne man beregnet ny motstandsverdi slik:

$$R = \frac{(5[V] - 3.5[V])}{0.04[A]} = 37.5[\Omega] \quad (4.2)$$

PWM frekvens for å dimme lysdiode, regnet ut med formel fra teori kapittel 2.3.2:

$$f_{PWM} = \frac{f_{clkIO}}{N * 256} = \frac{8 * 10^6}{1 * 256} = 31250Hz = 31.25kHz \quad (4.3)$$

Koden for å sette denne PWM frevensen, samt styre lysdioden finnes i appendiks B.5.

4.5 Sende, motta og sende tilbake CAN melding

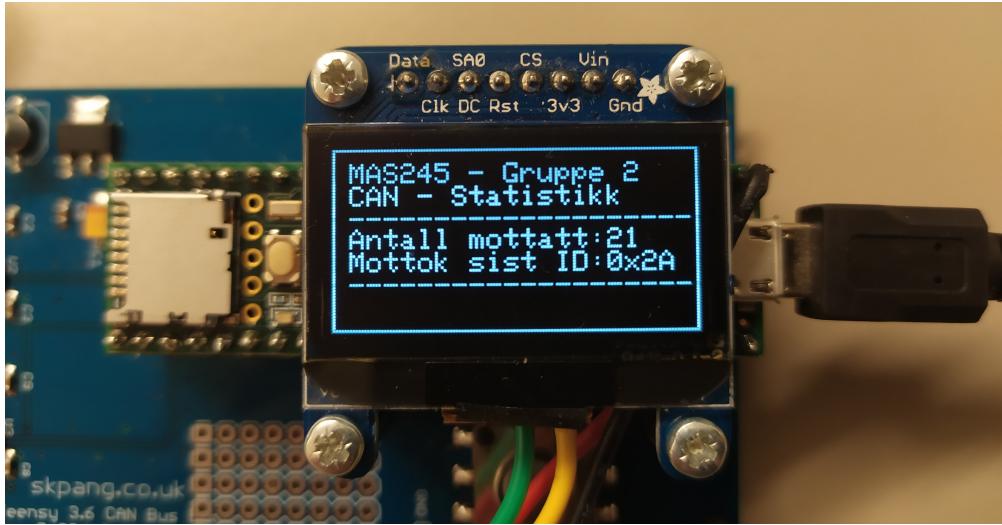
I figur 4.5 ser man sekvensen av at CAN bus port 0 (Can0) sender verdien 0. Så ser man at Can1 har mottat data. Verdien 0 printes, og dermed har Can0 sendt verdien 0 til Can1. Videre blir samme verdi returnert fra Can1 til Can0. Denne printes som før, og deretter blir verdien økt med 1, og så gjøres samme sekvens igjen.

```
CAN Bus send and receive message test (Tx-Rx-Tx)
TX with Can0: 0
Can1 is available.
Can1 reading: 0
TX with Can1: 0
Can0 is available.
Can0 reading: 0
Adding 1 to msg.buf[0]
TX with Can0: 1
Can1 is available.
Can1 reading: 1
TX with Can1: 1
Can0 is available.
Can0 reading: 1
Adding 1 to msg.buf[0]
TX with Can0: 2
```

Figur 4.5: CAN Bus Tx-Rx-Tx

4.6 CAN meldingsmottaker

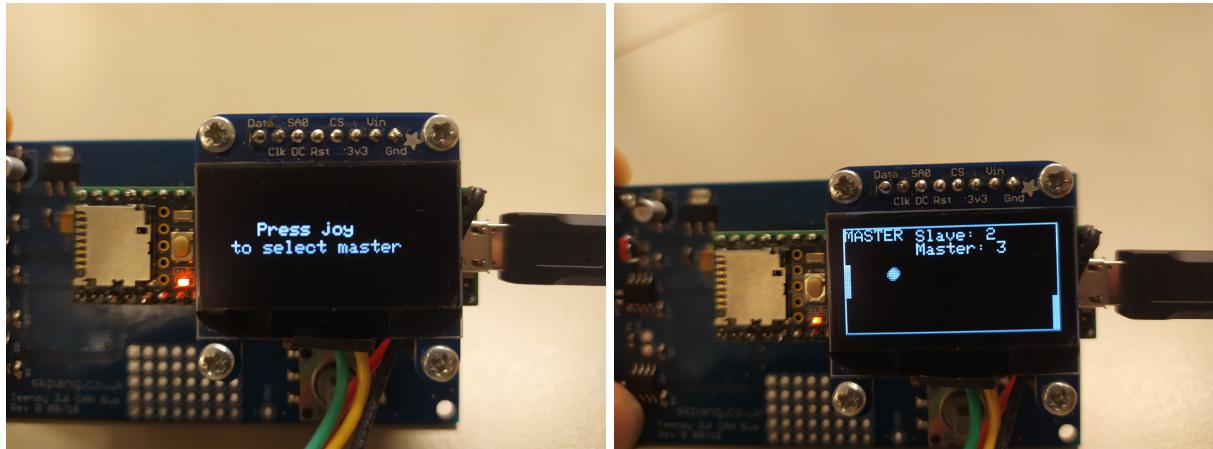
Designet på CAN meldingsmottakeren ble som i figur 4.6. Her vises gruppenummer, hvor mange CAN meldinger som er mottatt og hvilken ID som ble mottatt fra sist.



Figur 4.6: CAN meldingsmottaker design

4.7 CAN Ping-pong

Resultatet av CAN Ping-pong spillet er enklest å presentere i videoformat¹. I videoen ser man to spillere som spiller mot hverandre på hver sin Teensy.



(a) Startskjerm til ping-pong spill.

(b) Ping-pong spill.

Figur 4.7: CAN Ping-pong GUI design

¹<https://youtu.be/NLrhYjp2-kg>

5 Diskusjon

5.1 Strømforsyning

Kondensatoren C1 kreves når spenningsregulatoren er langt fra strømforsyningsfilter. C2 må minst være 100 [μF] for å opprettholde stabilitet, og bør monteres så nærmest regulator som mulig. Det er også viktig at C2 tåler samme temperatur som selve regulatoren. I dette tilfellet tåler regulatoren $-40\text{ }^{\circ}\text{C}$ og elektrolytten i en aluminiums kondensator kan ofte fryse ved $-30\text{ }^{\circ}\text{C}$. Ved å bruke en dyrere tantalum kondensator framfor en aluminiums kondensator kan man halvere verdien på komponenten. Verdien kan i tillegg halveres enda en gang om ikke trenger å bruke regulatoren under $25\text{ }^{\circ}\text{C}$. Se datablad [14].

5.2 Tilkobling til PC

Denne oppgaven tok litt lengre tid enn først tenkt. Grunnen til det var først og fremst dårlig kontakt mellom kontroller og brødbrett. Dette medførte at spenning ble brutt og MCU sluttet og virke. Løsningen ble å sette en tvinge mellom toppen av MCU og bordet den stod på. Dette kunne også vært løst ved å sette MCU i en 28 pin DIP socket, men dette var ikke tilgjengelig.

5.3 Hello World program

Ved å beregne riktig motstanden til LED-en, ble det unngått å brenne den opp, og Hello World programmet virket som forventet.

5.4 Soft-blink

For å få en mest mulig myk blinkeeffekt, så er det gunstig å bruke høy PWM frekvens, slik at øyet ikke får med seg at LED-en i realiteten skrus av og på. Forskjellige personer vil ha forskjellig opplevelse av når lys blinker eller er konstant, men for å være på den sikre siden, ble det valgt så høy PWM frekvens som mulig, beregnet med formel 2.10. Selve verdien presenteres i resultater.

Variabelen N i beregning av PWM frekvens, representerer "prescale" faktoren. Når skaleringsfaktoren velges til 1, så deles ikke PWM frekvensen ned, og vi oppnår høyeste PWM frekvens som er mulig med den interne klokken i MCU. Det ble prøvd ut å dele ned denne frekvensen med alle faktorene. Ved 1024 kunne man se at LED-en blinket, mens den dimmet, og etter hvert som N ble satt lavere, fikk man en mykere blinkeeffekt.

5.5 Teensy og CAN bus

5.5.1 Sende, motta og sende tilbake CAN melding

I eksperimentet hvor en CAN medling skulle sendes, mottas og returneres ble det tydelig hvor stabilt og pålitelig protokollen er. Alle meldinger ble sendt, mottatt, sendt tilbake og så printet til seriell monitor. Det ble aldri observert feil i melding eller ID. Likevel var det relativt enkle meldinger som ble sendt, så hvorvidt dette blir mer komplekst i større systemer ble ikke utforsket.

5.5.2 CAN meldingsmottaker

Samme erfaring ble observert med CAN meldingsmottakeren. Her var det spennende å kombinere både OLED og CAN bus bibliotek i samme oppgave, for å lage et brukbart produkt.

5.5.3 CAN Ping-pong

Denne oppgaven var definitivt den mest tidkrevende i denne delen av prosjektet. Det krevde mange timer å utarbeide all logikken som krevdes for å få til et fungerende CAN Ping-pong spill. Her var det mange utfordringer som skulle løses, blant annet hvor spilloppgikken skulle kjøres og hvordan motpart da skulle få tilgang på det som skjedde i spillet. En annen viktig observasjon var at ved å bruke den innebygde delay funksjonen, som blokkerer kode, så fikk motparten en input lag som gjorde det svært utfordrende å spille. Ved å bytte ut delay med følgende kode, så ble man kvitt dette:

```
millisDiff = currentMillis - lastMillis;
if(millisDiff >= updateRate)
{
    updateGame = true;
}
```

Listing 5.1: Ikke-blokkerende kode

updateRate variabelen er da oppdateringshastigheten (100Hz).

5.6 Raspberry, Buildroot og CAN-bus

5.6.1 Installasjon og konfigurering

Denne oppgaven gav oss en bratt læringskurve. Her var det til å begynne med mye nytt å sette seg inn i. Spesielt er det å bruke Linux og terminal utfordrene for oss som ikke har kjennskap til dette i fra før. Buildroot ble pakket ut og ordnet i mappestruktur og make fungerte som det skulle. Deretter ble vi kjent med make. Med tildelt konfigurasjon ble det problemer med kompileringen av OS-er. 8 bygginger ble gjennomført før imagefil ble ferdig kompilert. Error 2 oppstod hver gang. Det ble endret flere instillinger i .config filen til buildroot. Blant annet var kjerne versjon prøvd byttet til standard raspberry pi template sin og root filsystem overlay mappe var ikke forandret. I tillegg måtte flere tillegspakker installeres for å kunne bruke qt prosjektet can_pingpong. Til slutt fikk vi bygget ferdig imagefilen og fikk montert bildefilen på sd-kortet. Neste problem var å koble til med ssh fra terminal på pc. Oppgitt ipadresse var feil og vi fikk ikke kontakt med RPi. Løsningen nå var

å finne hvor nettverksoppsett var angitt for så og endre ip og nettverksmaske. Aller siste problem tydet på feil med tastaturopsett og innlogging som root var umulig.

Med ny tildelt .config fil fungerte kryss-kompileringen og image filen kunne flashes til minnekortet. Først var det trøbbel med at det ikke var mulig å finne IP adressen til RPi, men etter at det ble koblet på skjerm og tastatur, så virket dette hver gang. Antagelig var det da eth0 grensesnittet som ikke ble startet uten skjermen. Med skjerm koblet til, kunne man enkelt koble til RPi med ssh, for å teste ut kan grensesnittet.

5.6.2 Koble til og sende CAN-meldinger

Først ble modprobe kommandoene utført, for å laste kjernemodulene for SPI og CAN: `modprobe spi_bcm2835` og `modprobe mcp251x`, deretter ble følgende kommando kjørt for å initialisere can0: `/sbin/ip link set can0 up type can bitrate 250000`. Her støtte vi på enda et problem, nemlig at can0 ikke fantes. Da ble det utført en reboot av RPi, og etter at dette var gjort, så ble can0 funnet. Videre ble CAN kommandoer testet ut for å sende/motta CAN meldinger.

5.6.3 Styre servo med IMU via CAN-bus

Ved modifisering av can_pingpong pakken for å motta, gange opp og sende tilbake CAN melinger på RPi, var det tidkrevende å finne ut av hvordan man kunne skrive en fornuftig kode som løste dette problemet, men tilslutt gikk det og resultatet ble veldig stilig.

6 Konklusjon

Dette prosjektet har vært svært lærerikt og mangfoldig. Alle oppgavene ble gjennomført. Flere nye programmer måtte læres, og det har vært veldig nyttig med tanke på videre prosjekter. For å nevne noen, har man det å tegne kretskjema med Eagle, lære å bruke git aktivt, bruke terminal og funksjoner i Linux og ikke minst CAN-bus. Alt i alt har dette prosjektet lagt et godt grunnlag fram mot et bachelorprosjekt. Hele prosjektet med alle kodefiler kan finnes på GitHub¹ og en kort film hvor flere av løsningene er dokumentert finnes på YouTube².

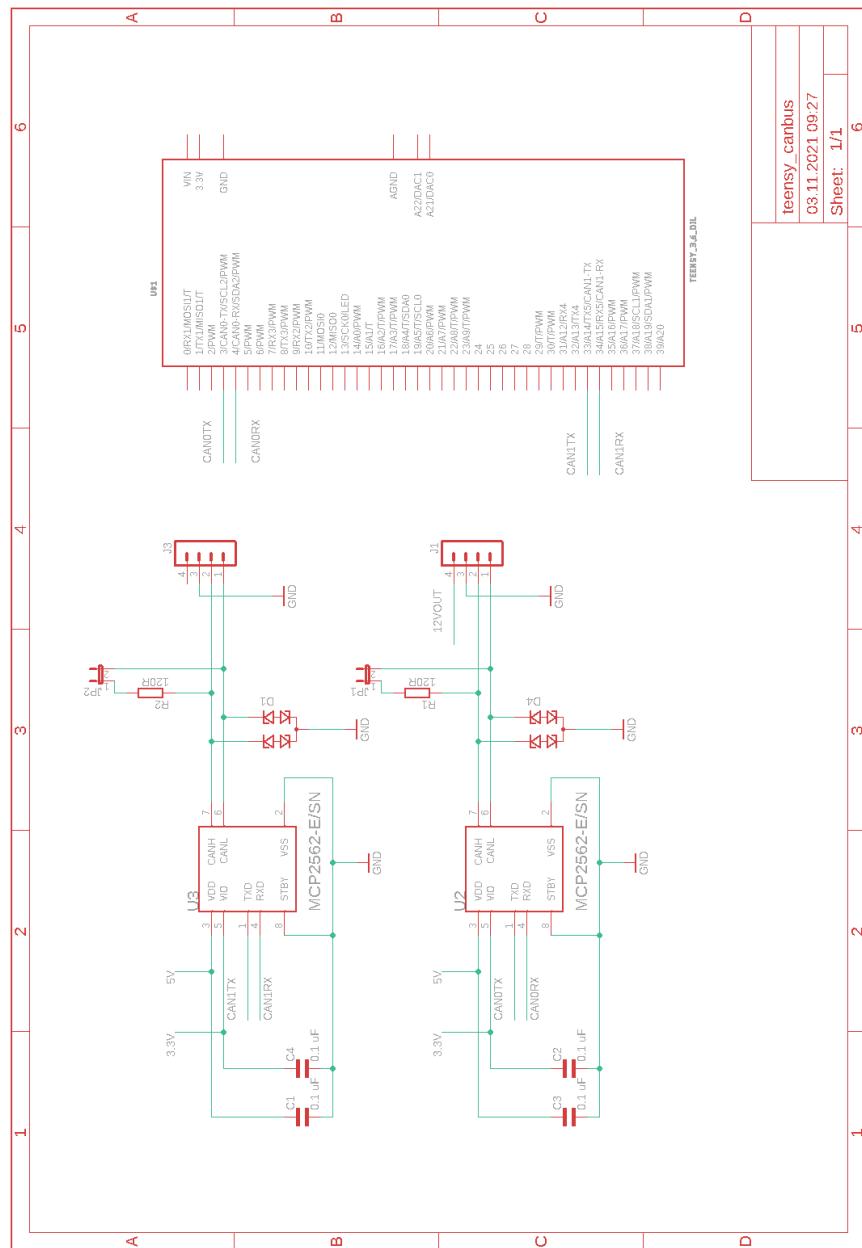
¹<https://github.com/eidetech/MAS245Prosjekt>

²<https://youtu.be/NLrhYjp2-kg>

7 Appendiks

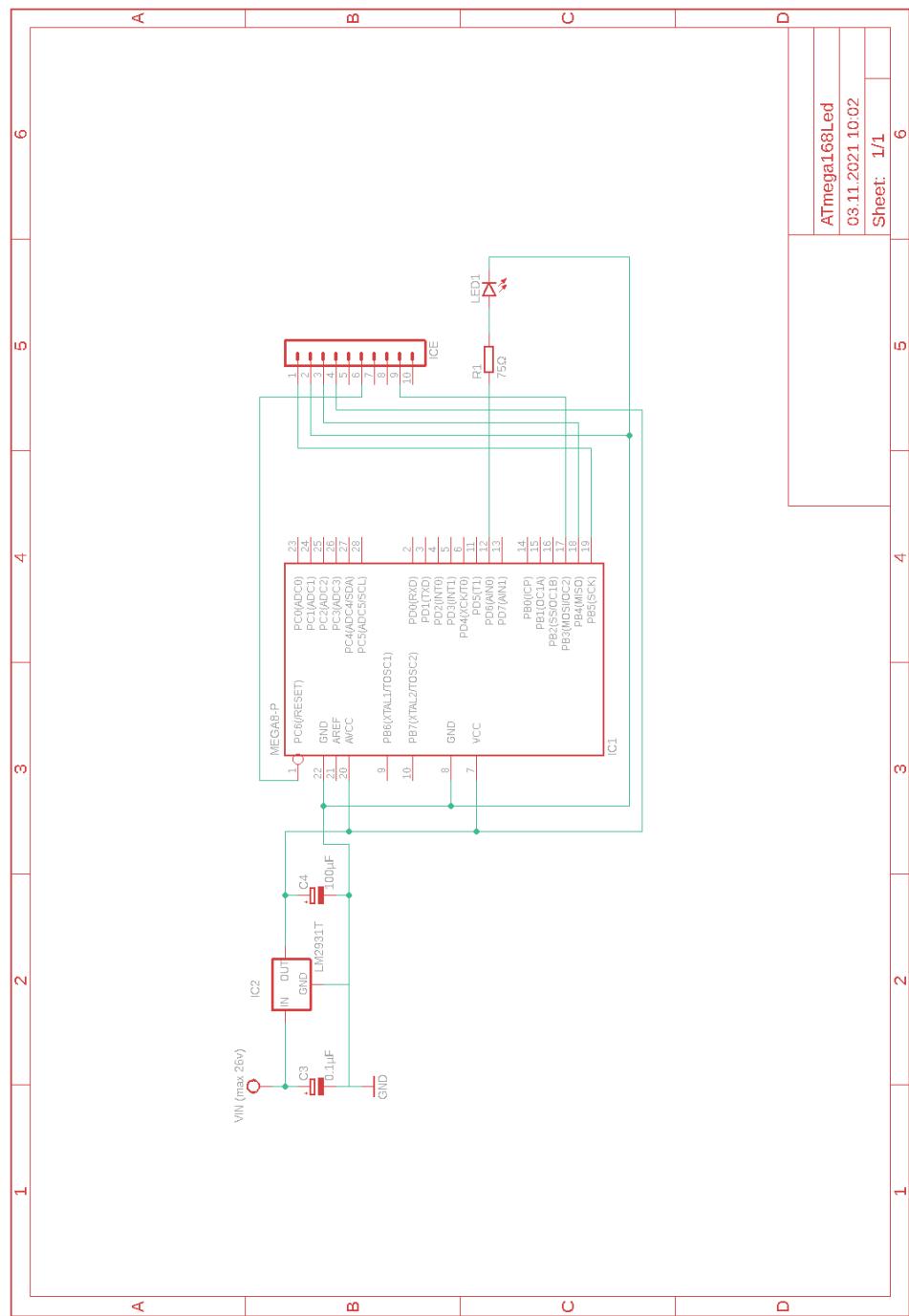
A Kretsskjema

A.1 Teensy 3.6 med CAN bus tranceiver



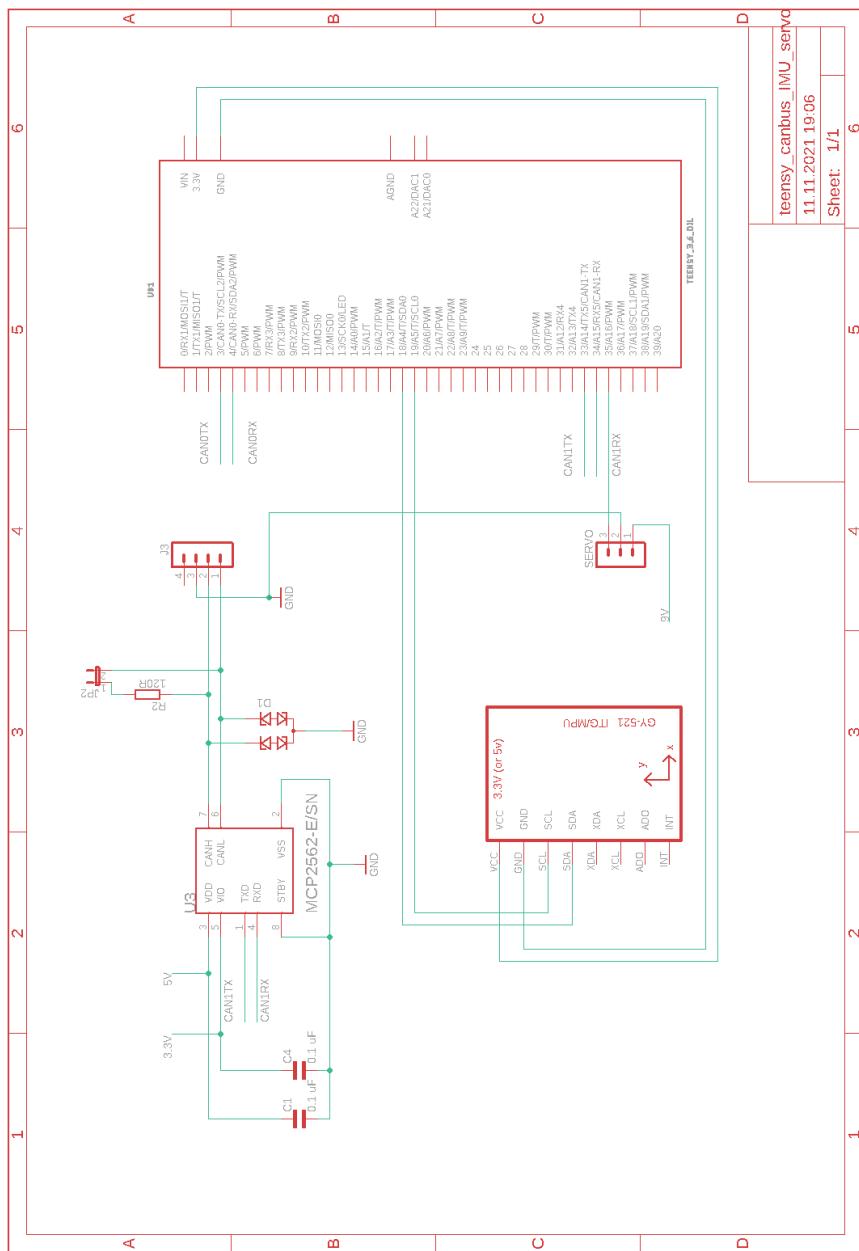
Figur A.1: Teensy 3.6 med CAN bus tranceivere.

A.2 ATmega168 med LED og motstand



Figur A.2: ATmega 168 med LED og motstand

A.3 Teensy med CAN, IMU og servo



Figur A.3: Teensy med CAN, IMU og servo

B Kode

B.1 IMU over CAN

```
#include <FlexCAN.h>
#include <SPI.h>

#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Wire.h>
#include <Servo.h>

#include <Fonts/FreeMono9pt7b.h>

#if (SSD1306_LCDHEIGHT != 64)
#error("Height incorrect, please fix Adafruit_SSD1306.h!");
#endif

#ifndef __MK66FX1M0__
#error "Teensy 3.6 with dual CAN bus is required to run this example"
#endif

#define OLED_DC      6
#define OLED_CS     10
#define OLED_RESET   5
Adafruit_SSD1306 display(OLED_DC, OLED_RESET, OLED_CS);

static CAN_message_t msg;
Adafruit_MPU6050 mpu;
Servo servo;

int imuReading;

// -----
void setup(void)
{
    Serial.begin(9600);
    delay(1000);

    Can0.begin(250000);

    //if using enable pins on a transceiver they need to be set on
    pinMode(2, OUTPUT);
    pinMode(35, OUTPUT);

    digitalWrite(2, HIGH);
    digitalWrite(35, HIGH);

    msg.ext = 0;
    msg.id = 0x200;
    msg.len = 8;
    msg.buf[0] = 10;

    if (!mpu.begin()) {
        Serial.println("Failed to find MPU6050 chip");
        while (1) {
```

```

        delay(10);
    }
}

mpu.setAccelerometerRange(MPU6050_RANGE_16_G);
mpu.setGyroRange(MPU6050_RANGE_250_DEG);
mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
display.fillScreen(BLACK);
display.setTextSize(1);
display.setTextColor(WHITE);

servo.attach(35);
}

// -----
void loop(void)
{
    display.fillScreen(BLACK);
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);

    /*Leser IMUData*/
    imuReading = a.acceleration.x*10;

    /*Sender IMUData ut på CAN-bus*/
    msg.buf[0] = imuReading;
    Can0.write(msg);

    /*Printer info på teensy skjerm*/
    Serial.println(imuReading);
    display.setCursor(1,1);
    display.print("Servostyring fra IMU");
    display.setCursor(1,10);
    display.print("TX til RPi: ");
    display.print(imuReading);

    CAN_message_t inMsg;
    if(Can0.available())
    {
        /* Leser CAN fra RPi*/
        Can0.read(inMsg);
        if(inMsg.id == 0x300)
        {
            /*Printer innkommende CAN*/
            display.setCursor(1,20);
            display.print("RX fra RPi: ");
            display.print(inMsg.buf[0]);
        }

        /*Konverterer IMUdata til Servodata*/
        int val = map(inMsg.buf[0], 0,200, 0, 180);

        /*Lager "dødgang* for IMU*/
        if(inMsg.buf[0] > 80 && inMsg.buf[0] < 120)
        {
            /*Setter servo i ro i midtstilling*/
            servo.write(92);
        }
        else
        {
            servo.write(val);
        }

        /*Printer verdi som sendes til servo*/
        display.setCursor(1,30);
        display.print("Servo signal: ");
        display.print(val);
    }
}

```

```

        }
    /*Oppdateringsfrekvens på skjermen*/
    display.display();
    delay(60);
}

```

Listing B.1: CAN over IMU ino sketch

B.2 Modifisert can_pingpong pakke

```

// 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 <iostream>
#include <time.h>

// 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) * 1.0e6);

bool createCanSocket(int& socketDescriptor);
void RXTX_CAN(int socketDescriptor);

void myPeriodicTask()
{
    static uint64_t ii(0U);
    ++ii;
    std::cout << "Tick " << ii << "..." << std::endl;
    receiveCanMessage(canSocketDescriptor);
}

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;

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 RXTX_CAN(int socketDescriptor)
{
    int nbytes;
    struct can_frame frame;
    nbytes = read(socketDescriptor, &frame, sizeof(struct can_frame));
    if (nbytes < 0) {
        perror("Read");
    }
    printf("0x%03X [%d] ", frame.can_id, frame.can_dlc);
    for (int i = 0; i < frame.can_dlc; i++)
    {
        printf("%d", frame.data[i]);
    }
    printf("\r\n");

    std::cout << "Frame data 0 original: ";
    printf("%d", frame.data[0]);
    std::cout << std::endl;
    frame.can_id = 0x300;
    frame.data[0] = frame.data[0] * 0x02;
    std::cout << "Frame data 0 modified: ";
    printf("%d", frame.data[0]);
    std::cout << std::endl;
    // Write modified CAN message back to Teensy
    write(socketDescriptor, &frame, sizeof(struct can_frame));
}

```

Listing B.2: Redigert can_pingpong.cpp fil

B.3 Eksempelprogram med en mikrokontrollerpinne med lav verdi

```

#include <avr/io.h>
#define F_CPU 8000000UL // Set clock frequency to 8MHz
#include <util/delay.h>

int main(void)
{
    DDRD = (1U << 6U); // Data port direction register, set pin 6 of DDRD to ...
    output.
    PORTD &= ~(1U << 6U); // Setting pin 6 of PORT D register to LOW.
    while (1)
    {
    }
}

```

Listing B.3: Mikrokontrollerpinne med lav verdi

B.4 HelloWorld kode

```

#include <avr/io.h>
#define F_CPU 8000000UL // Set clock frequency to 8MHz
#include <util/delay.h>

```

```

int main(void)
{
    DDRD = (1U << 6U); // Data port direction register, set pin 6 of DDRD to ...
    // output.
    PORTD &= ~(1U << 6U); // Setting pin 6 of PORT D register to LOW.
    while (1)
    {
        _delay_ms(1000);
        PORTD &= ~(1U << 6U); // Turn LED off

        _delay_ms(1000);
        PORTD |= (1U << 6U); // Turn LED on
    }
}

```

Listing B.4: HelloWorld kode

B.5 Soft-blink

```

#include <avr/io.h> // https://www.nongnu.org/avr-libc/user-manual/...
    group__avr__io.html
#define F_CPU 8000000UL // Set CPU speed of 8 MHz
#include <util/delay_basic.h> // https://www.nongnu.org/avr-libc/user-manual/...
    group__util__delay__basic.html

// Datasheet: Microchip ATmega48/V/88/V/168/V - DS40002074A

// Function to configure PWM output on OC0A (pin PD6/12) in TCCROA and TCCR0B ...
    registers
void setupPWM()
{
    /*
    Register operations:
    TCCROA: Table 15-3: Compare output mode, fast PWM mode: COM0A1 = 1 -> Clear ...
        OC0A on compare match, set OC0A at BOTTOM, (non-inverting mode)
    TCCROA: Table 15-8: Waveform generation mode bit description: WGM00 = 1 -> ...
        Fast PWM
    TCCROA: Table 15-8: Waveform generation mode bit description: WGM01 = 1 -> ...
        Fast PWM

    TCCR0B: Table 15-9: Clock source select bit description: CS00 = 1 -> clk I/O ...
        / (no prescaling)

    Data direction register operations:
    DDRD - The port D data direction register: DDD6 = 1 (DDRD |= (1U << 6U);) -> ...
        Set direction of PD6 to output.
    */

    TCCROA |= (1U << COM0A1) | (1U << WGM01) | (1U << WGM00); // Configure all ...
        bits for TCCROA
    TCCR0B |= (1U << CS00); // Configure all bits for TCCR0B

    DDRD |= (1U << 6U); // Set PD6 pin as output
}

/* Function for setting duty cycle on OCROA pin (PD6)
   How to calculate PWM percentage (not 0-255 value):
   x = 100/255 = 0.392157
   y = 0-255 value
   dutyCycle% = x*y
*/
void outputPWM(uint8_t dutyCycle)
{

```

```

    OCROA = dutyCycle;
}
int main()
{
    setupPWM();
    while(1)
    {
        // Increase brightness of LED
        for(int i = 0;i <= 255;i++)
        {
            outputPWM(i);
            _delay_loop_2(1500);
        }

        // Decrease brightness of LED
        for(int i = 255; i >= 0; i--)
        {
            outputPWM(i);
            _delay_loop_2(1500);
        }
    }
}

```

Listing B.5: Soft-blink

B.6 CAN Bus Tx-Rx-Tx

```

#include <FlexCAN.h>

#ifndef __MK66FX1M0__
#error "Teensy 3.6 with dual CAN bus is required to run this example"
#endif

static CAN_message_t msg;

void setup(void)
{
    Serial.begin(9600);
    Serial.println("CAN Bus send and receive message test (Tx-Rx-Tx)");

    Can0.begin(250000);
    Can1.begin(250000);

    msg.ext = 0;
    msg.id = 0x00;
    msg.len = 1;
    msg.buf[0] = 0x00;
}

void loop(void)
{
    CAN_message_t returMsg;

    // Send msg with Can0 bus:
    Serial.print("TX with Can0: ");
    Serial.println(msg.buf[0]);
    Can0.write(msg);
    delay(100);
    // Read msg from Can0 with Can1 bus:
    if (Can1.available())
    {
        Serial.println("Can1 is available.");
        Can1.read(returMsg);
    }
}

```

```

    Serial.print("Can1 reading: ");
    Serial.println(returMsg.buf[0]);
}
delay(100);
// Return msg from Can0 with Can1 to Can0:
Can1.write(returMsg);
Serial.print("TX with Can1: ");
Serial.println(returMsg.buf[0]);
delay(100);
if (Can0.available())
{
    Serial.println("Can0 is available.");
    Can0.read(returMsg);
    Serial.print("Can0 reading: ");
    Serial.println(returMsg.buf[0]);
}
delay(100);
Serial.println("Adding 1 to msg.buf[0]");
msg.buf[0]++;
}

}

```

Listing B.6: CAN Bus Rx-Tx

B.7 CAN Message receiver

```

#include <FlexCAN.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#include <Fonts/FreeMono9pt7b.h>

#define OLED_DC      6
#define OLED_CS     10
#define OLED_RESET   5
Adafruit_SSD1306 display(OLED_DC, OLED_RESET, OLED_CS);

#if (SSD1306_LCDHEIGHT != 64)
#error "Height incorrect, please fix Adafruit_SSD1306.h!";
#endif

#ifndef __MK66FX1M0__
#error "Teensy 3.6 with dual CAN bus is required to run this example"
#endif

static CAN_message_t msg;

void setup() {

    Can0.begin(500000);
    Can1.begin(500000);

    // by default, we'll generate the high voltage from the 3.3v line internally!...
    // (neat!)
    display.begin(SSD1306_SWITCHCAPVCC);

    // Clear the display buffer.
    display.clearDisplay();

    // Configure display settings
    display.setTextSize(0);
}

```

```

display.setTextColor(WHITE);
display.setCursor(0, 0);

display.drawRect(0, 0, 128, 64, WHITE); // Rectangle around whole display
display.setCursor(0, 5);

// Print static text on display
display.println(" MAS245 - Gruppe 2");
display.println(" CAN - Statistikk");
display.println(" -----");
display.display();

// Setup message structure
msg.id = 0x00;
msg.buf[0] = 0x00;

}

void loop() {

CAN_message_t returMsg;

// Send msg with Can0 bus:
Can0.write(msg);

// Read msg from Can0 with Can1 bus:
while (Can1.available())
{
    display.setCursor(0, 28);
    Serial.println("Can1 is available.");
    Can1.read(returMsg);
    Serial.print("CAN bus 1 reading: ");
    Serial.println(returMsg.buf[0]);
    display.fillRect(2, 25, 125, 30, BLACK);
    display.print(" Antall mottatt:");
    display.println(msg.buf[0]);
    display.print(" Mottok sist ID:0x");
    display.println(msg.id, HEX);
    display.println(" -----");

    display.display();

    // Increase data in buffer 0 with 1
    msg.buf[0]++;

    // Add 2 to the ID, for display effect
    msg.id = msg.id + 2;
    delay(1000);
}
}

```

Listing B.7: CAN Message receiver

B.8 CAN_Pong.ino

```

#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <FlexCAN.h>

#include <Fonts/FreeMono9pt7b.h>

```

```

#include "Paddle.h"
#include "Ball.h"

#if (SSD1306_LCDHEIGHT != 64)
#error("Height incorrect, please fix Adafruit_SSD1306.h!");
#endif

#ifndef __MK66FX1M0__
#error "Teensy 3.6 with dual CAN bus is required to run this example"
#endif

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

#define JOY_DOWN 22
#define JOY_UP 23
#define JOY_CLICK 19
#define JOY_LEFT 18
#define JOY_RIGHT 17

#define OLED_DC      6
#define OLED_CS      10
#define OLED_RESET   5
Adafruit_SSD1306 display(OLED_DC, OLED_RESET, OLED_CS);

Paddle leftPaddle(LEFT);
Paddle rightPaddle(RIGHT);
Ball ball(RIGHT);

int joyUp, joyDown, joyClick, joyUpSlave, joyDownSlave, joyClickSlave;
bool gameState = false;
bool gameStart = true;
bool isMaster = false;
bool endGame = false;
bool updateGame = false;
int gruppeNr = 2;
int turn = 0;

unsigned long currentMillis;
unsigned long lastMillis = 0;
const int updateRate = 10; // ms
unsigned long millisDiff;

int scoreMaster = 0;
int scoreSlave = 0;

static CAN_message_t TX_Ball;
static CAN_message_t TX_Paddle;
static CAN_message_t TX_Joy;
static CAN_message_t RX_Ball;
static CAN_message_t RX_Paddle;
static CAN_message_t RX_Joy;

void setup() {
  Serial.begin(9600);
  Can0.begin(1000000);

  pinMode(JOY_DOWN, INPUT);
  pinMode(JOY_UP, INPUT);
  pinMode(JOY_CLICK, INPUT);
  pinMode(JOY_LEFT, INPUT);
  pinMode(JOY_RIGHT, INPUT);

  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.fillScreen(BLACK);
}

```

```

display.setTextSize(1);
display.setTextColor(WHITE);

// Paddle CAN message (for sending data to slave)
TX_Paddle.id = gruppeNr + 20; // 22
TX_Paddle.len = 7;
TX_Paddle.buf[0] = leftPaddle.paddle_y;
TX_Paddle.buf[1] = rightPaddle.paddle_y;
TX_Paddle.buf[2] = joyClick;
TX_Paddle.buf[3] = scoreMaster;
TX_Paddle.buf[4] = scoreSlave;
TX_Paddle.buf[5] = gameState;
TX_Paddle.buf[6] = turn;

// Ball CAN message (for sending data to slave)
TX_Ball.id = gruppeNr + 50; // 52
TX_Ball.len = 2;
TX_Ball.buf[0] = ball.x;
TX_Ball.buf[1] = ball.y;

// Joy CAN message (for sending data to master)
TX_Joy.id = 0;
TX_Joy.len = 3;
TX_Joy.buf[0] = joyUp;
TX_Joy.buf[1] = joyDown;
TX_Joy.buf[2] = joyClick;
}

void loop() {
    currentMillis = millis();

    selectMaster();
    readJoy();
    receiveSlaveJoy();
    gameState = true;

    millisDiff = currentMillis - lastMillis;
    if(millisDiff >= updateRate)
    {
        updateGame = true;
    }
    // Game sequence
    while(gameState == true && updateGame == true)
    {
        if(isMaster)
        {
            movePaddle(); // Move paddles based on read data
        }
        draw(); // Draw
        if (gameOver())
        {
            endGame = true;
        }
        display.display();
        display.fillScreen(BLACK);
        //delay(10);
        lastMillis = millis();
        updateGame = false;
    }
}

void readJoy()
{
    joyUp = digitalRead(JOY_UP);
    joyDown = digitalRead(JOY_DOWN);
    joyClick = digitalRead(JOY_CLICK);
}

```

```

    if (!isMaster)
    {
        TX_Joy.buf[0] = joyUp;
        TX_Joy.buf[1] = joyDown;
        TX_Joy.buf[2] = joyClick;
        Can0.write(TX_Joy);
    }
}

void receiveSlaveJoy()
{
    if (isMaster){
        if (Can0.available())
        {
            Can0.read(RX_Joy);

            if (RX_Joy.id == 0)
            {
                joyUpSlave = RX_Joy.buf[0];
                joyDownSlave = RX_Joy.buf[1];
                joyClickSlave = RX_Joy.buf[2];
            }
        }
    }
}

void movePaddle()
{
    if(joyUp == 0)
    {
        leftPaddle.moveUp();
    }else if(joyDown == 0)
    {
        leftPaddle.moveDown();
    }
    if(joyUpSlave == 0)
    {
        rightPaddle.moveUp();
    }else if(joyDownSlave == 0)
    {
        rightPaddle.moveDown();
    }
}

void draw()
{
    display.drawRect(0, 0, 128, 64, WHITE); // Rectangle around whole display
    if(isMaster)
    {
        display.setCursor(1,1);
        display.print("MASTER");
        display.setCursor(SCREEN_WIDTH/2 - 20, 2);
        display.print("Slave: ");
        display.println(scoreSlave);
        display.setCursor(SCREEN_WIDTH/2 - 20, 10);
        display.print("Master: ");
        display.println(scoreMaster);

        display.fillRect(leftPaddle.paddle_x, leftPaddle.paddle_y, leftPaddle....
paddleWidth, leftPaddle.paddleHeight, WHITE); // Left paddle
        display.fillRect(rightPaddle.paddle_x, rightPaddle.paddle_y, rightPaddle....
paddleWidth, rightPaddle.paddleHeight, WHITE); // Right paddle

        TX_Paddle.buf[0] = leftPaddle.paddle_y;
        TX_Paddle.buf[1] = rightPaddle.paddle_y;
        if(ball.turn == LEFT)
    }
}

```

```

{
    TX_Paddle.buf[6] = 1;
}else
{
    TX_Paddle.buf[6] = 0;
}
Can0.write(TX_Paddle);

ball.moveBall();
ball.limitCheck();

    // Right paddle settings
if(ball.x > (SCREEN_WIDTH - 2*ball.r - rightPaddle.paddleWidth) && (ball.y ...
> rightPaddle.paddle_y - ball.r && ball.y < rightPaddle.paddle_y+rightPaddle...
.paddleHeight + ball.r))
{
    ball.xDir = ball.xDir*(-1);
    // Left paddle settings
}else if(ball.x < (2*ball.r + leftPaddle.paddleWidth - 1) && (ball.y > ...
leftPaddle.paddle_y - ball.r && ball.y < leftPaddle.paddle_y+leftPaddle.....
paddleHeight + ball.r))
{
    ball.xDir = ball.xDir*(-1);
}

display.fillCircle(ball.x, ball.y, ball.r, WHITE); // Ball
TX_Ball.buf[0] = ball.x;
TX_Ball.buf[1] = ball.y;
Can0.write(TX_Ball);
}else
{
    if(Can0.available())
    {
        Can0.read(RX_Paddle);

        if (RX_Paddle.id == 22)
        {
            leftPaddle.paddle_y = RX_Paddle.buf[0];
            rightPaddle.paddle_y = RX_Paddle.buf[1];
            scoreMaster = RX_Paddle.buf[3];
            scoreSlave = RX_Paddle.buf[4];

            display.setCursor(1,1);
            display.print("SLAVE");
            display.setCursor(SCREEN_WIDTH/2 - 20, 2);
            display.print("Slave: ");
            display.println(scoreSlave);
            display.setCursor(SCREEN_WIDTH/2 - 20, 10);
            display.print("Master: ");
            display.println(scoreMaster);

            display.fillRect(leftPaddle.paddle_x, leftPaddle.paddle_y, leftPaddle...
.paddleWidth, leftPaddle.paddleHeight, WHITE); // Left paddle
            display.fillRect(rightPaddle.paddle_x, rightPaddle.paddle_y, ...
leftPaddle.paddleWidth, leftPaddle.paddleHeight, WHITE); // Right paddle
        }
    }

    if(Can0.available())
    {
        Can0.read(RX_Ball);

        if (RX_Ball.id == 52)
        {
            ball.x = RX_Ball.buf[0];
            ball.y = RX_Ball.buf[1];
            display.fillCircle(ball.x, ball.y, ball.r, WHITE); // Ball
        }
    }
}

```

```

        }
    }
}

bool gameOver()
{
    if (ball.x < 0 || ball.x > SCREEN_WIDTH)
    {

        if(Can0.available())
        {
            Can0.read(RX_Paddle);

            if (RX_Paddle.id == 22)
            {
                turn = RX_Paddle.buf[6];
                Serial.print("Turn: ");
                Serial.println(turn);
            }
        }
        gameState = false;
        TX_Paddle.buf[5] = gameState; // Send gameState to slave
        display.fillScreen(BLACK);
        display.setCursor(25,25);
        if(isMaster)
        {
            if(ball.turn == RIGHT)
            {
                display.print("Point to Master");
                scoreMaster += 1;
            }else if(ball.turn == LEFT)
            {
                display.print("Point to Slave");
                scoreSlave += 1;
            }
        }else
        {
            if(turn == 0)
            {
                display.print("Point to Master");
            }else if(turn == 1)
            {
                display.print("Point to Slave");
            }
        }
        resetGame();
        TX_Paddle.buf[3] = scoreMaster;
        TX_Paddle.buf[4] = scoreSlave;
        Can0.write(TX_Paddle);
        display.display();
        delay(1000);
        return true;
    }else
    {
        return false;
    }
}

void selectMaster()
{
    // Loop for selecting master
    while(gameStart)
    {
        // Starting screen before selecting master
        display.setCursor(25,25);
        display.print("Press joy");
    }
}

```

```

display.setCursor(10,35);
display.println("to select master");
display.display();

// Check if other device wants to be master
if(Can0.available())
{
    Can0.read(RX_Joy);

    if (RX_Joy.id == 0 && RX_Joy.buf[2] == 0)
    {
        isMaster = false;
        gameStart = false;
        display.setCursor(1,1);
        display.print("SLAVE");
        display.display();
    }
}

// Check if you want to be master
readJoy();

if(joyClick == 0)
{
    TX_Joy.buf[2] = joyClick;
    isMaster = true;
    gameStart = false;

    Can0.write(TX_Joy);
    display.setCursor(1,1);
    display.print("MASTER");
    display.display();
}
}

//delay(1000);
}

void resetGame()
{
    ball.x = SCREEN_WIDTH/2;
    ball.y = SCREEN_HEIGHT/2 -11;

    leftPaddle.paddle_y = (SCREEN_HEIGHT/2)-(leftPaddle.paddleHeight/2);
    rightPaddle.paddle_y = (SCREEN_HEIGHT/2)-(rightPaddle.paddleHeight/2);

    TX_Ball.buf[0] = ball.x;
    TX_Ball.buf[1] = ball.y;

    TX_Paddle.buf[0] = leftPaddle.paddle_y;
    TX_Paddle.buf[1] = rightPaddle.paddle_y;
}

```

Listing B.8: CAN_Pong.ino

B.9 Ball.cpp

```

#include "Ball.h"

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

```

```

// Constructor
Ball::Ball(side paddleSide) : Paddle(paddleSide)
{
}

Ball::~Ball()
{
}

void Ball::moveBall()
{
    if(xDir == 1)
    {
        x++;
        turn = RIGHT;
    }else
    {
        x--;
        turn = LEFT;
    }
    if(yDir == 1)
    {
        y--;
    }else
    {
        y++;
    }
}

void Ball::limitCheck()
{
    if(y > SCREEN_HEIGHT - 2*r)
    {
        yDir = yDir*(-1);
    }else if(y < 2*r)
    {
        yDir = yDir*(-1);
    }
}

```

Listing B.9: Ball.cpp

B.10 Ball.h

```

#ifndef Ball_h
#define Ball_h

#include "Paddle.h"

class Ball : public Paddle
{
public:
    uint8_t x = 128/2;
    uint8_t y = 64/2;
    uint8_t xDir = 1;
    uint8_t yDir= 1;
    uint8_t r = 3;
    side turn = RIGHT;

    void moveBall();
    void limitCheck();
};

```

```

    Ball(side paddleSide);
    ~Ball();

private:
};

#endif

```

Listing B.10: Ball.h

B.11 Paddle.cpp

```

#include "Paddle.h"

// Constructor
Paddle::Paddle(side paddleSide)
{
    paddleWidth = 3;
    paddleHeight = 20;
    if(paddleSide == LEFT)
    {
        paddle_x = LEFT;
    }else if (paddleSide == RIGHT)
    {
        paddle_x = RIGHT;
    }else
    {
        paddle_x = 0;
    }
    paddle_y = (SCREEN_HEIGHT/2)-(paddleHeight/2);
}

// Destructor
Paddle::~Paddle()
{

}

// Function for moving paddle upwards
void Paddle::moveUp()
{
    if(paddle_y < (SCREEN_HEIGHT-paddleHeight-1))
    {
        paddle_y++;
    }
}

// Function for moving paddle downwards
void Paddle::moveDown()
{
    if(paddle_y > 1)
    {
        paddle_y--;
    }
}

```

Listing B.11: Paddle.cpp

B.12 Paddle.h

```
#ifndef Paddle_h
#define Paddle_h

#define SCREEN_HEIGHT 64
#include "Arduino.h"

enum side
{
    LEFT = 1,
    RIGHT = 128-4
};

class Paddle
{
public:
    uint8_t paddleWidth;
    uint8_t paddleHeight;
    uint8_t paddle_x;
    uint8_t paddle_y;

    Paddle(side paddleSide);
    ~Paddle();

    void moveUp();
    void moveDown();

private:
};

#endif
```

Listing B.12: Paddle.h

Bibliografi

- [1] P Thirumeni; Mainak Ghoshhajra; C M. Ananda. *Lessons learned in software implementation of ARINC 664 protocol stack in Linux*. URL: <https://ieeexplore.ieee.org/document/7057801>. (accessed: 17.11.2021).
- [2] Arduino. *IMU library*. URL: <https://www.arduino.cc/reference/en/libraries/adafruit-mpu6050/>. (accessed: 16.11.2021).
- [3] Atmel. *ATmega 168*. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48_88_168_megaAVR-Data-Sheet-40002074.pdf. (accessed: 07.10.2021).
- [4] Atmel. *Programmers and Debuggers*. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-ICE_UserGuide.pdf. (accessed: 03.11.2021).
- [5] Atmel. *pwm tips*. URL: <https://www.digikey.no/no/resources/datasheets/atmel/atmel-ice-user-guide>. (accessed: 14.10.2021).
- [6] Buildroot. *Builroot documentation*. URL: https://buildroot.org/downloads/manual/manual.html#_getting_started. (accessed: 05.11.2021).
- [7] Cburnett. *Single master to single slave: basic SPI bus example*. URL: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface#/media/File:SPI_single_slave.svg. (accessed: 03.11.2021).
- [8] Wikimedia Commons. *AMEGA168*. URL: https://commons.wikimedia.org/wiki/File:ATMega168_-_IC-ATM168-01.jpg. (accessed: 28.10.2021).
- [9] Copperhill. *Teensy 3.6 Dual CAN Bus Breakout Board*. URL: <https://copperhilltech.com/teensy-3-6-dual-can-bus-breakout-board/>. (accessed: 03.11.2021).
- [10] CSS Electronics. *What role does CAN bus termination play?* URL: https://www.csselectronics.com/products/terminal-resistor-can-bus?fbclid=IWAR2-5lt9ZMb1B8N9gDhMVMtZklkA_01YZPP6bQd0Mv3rXVvxamDa8LaHj0. (accessed: 17.11.2021).
- [11] PEAK-System Technik GmbH. *PCAN-USB FD User Manual*. URL: https://www.peak-system.com/produktcd/Pdf/English/PCAN-USB-FD_UserMan_eng.pdf. (accessed: 11.11.2021).
- [12] Frank Hofmann. *Understanding the ELF File Format*. URL: https://linuxhint.com/understanding_elf_file_format/. (accessed: 03.11.2021).
- [13] Microchip Technology Inc. *Microchip Studio for AVR and SAM Devices*. URL: <https://www.microchip.com/en-us/development-tools-tools-and-software/microchip-studio-for-avr-and-sam-devices>. (accessed: 03.11.2021).
- [14] Texas Instruments. *5v voltage regulator*. URL: https://www.ti.com/lit/ds/symlink/lm2931-n.pdf?ts=1634211427492&ref_url. (accessed: 14.10.2021).
- [15] InvenSense. *IMU datasheet*. URL: <http://www.haoyuelectronics.com/Attachment/GY-521/mpu6050.pdf>. (accessed: 11.11.2021).
- [16] Peak-systems. *Pin assignment D-Sub*. URL: <https://www.peak-system.com/PCAN-USB-FD.365.0.html?&L=1>. (accessed: 28.10.2021).
- [17] Raspberry Pi. *Raspberry Pi 3 Model B*. URL: <https://www.raspberrypi.com/products/raspberry-pi-3-model-b/>. (accessed: 11.11.2021).
- [18] PJRC. *Teensy® 3.6 Development Board*. URL: <https://www.pjrc.com/store/teensy36.html>. (accessed: 21.10.2021).

- [19] skpang. *PiCAN with GPS + Gyro +Accelerometer CAN-Bus for Raspberry Pi 3*. URL: https://www.skpang.co.uk/products/pican-with-gps-gyro-accelerometer-can-bus-for-raspberry-pi-3?_pos=8&_sid=2c3fd622d&_ss=r&variant=39617873510595. (accessed: 11.11.2021).
- [20] 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. (accessed: 04.11.2021).
- [21] skpang. *teensy36_canbus_brk_rev_A*. URL: <https://www.skpang.co.uk/products/teensy-3-6-dual-can-bus-breakout-board-include-teensy-3-6>. (accessed: 21.10.2021).
- [22] Sloan. *LED*. URL: https://www.elfa.se/Web/Downloads/85/60/L5-B5SC_eng_tds.pdf. (accessed: 07.10.2021).
- [23] Jurgen Wagenbach. *CAN bus topology and bus termination*. URL: <https://support.maxongroup.com/hc/en-us/articles/360009241840-CAN-bus-topology-and-bus-termination>. (accessed: 03.11.2021).
- [24] Wikimovel. *Broadcom BCM2837*. URL: https://wikimovel.com/index.php/Broadcom_BCM2837. (accessed: 11.11.2021).
- [25] Wikipedia. *Avionics Full-Duplex Switched Ethernet*. URL: https://en.wikipedia.org/wiki/Avionics_Full-Duplex_Switched_Ethernet. (accessed: 17.11.2021).
- [26] Wikipedia. *Serial Peripheral Interface*. URL: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface. (accessed: 17.11.2021).
- [27] wikipedia. *CAN bus*. URL: https://en.wikipedia.org/wiki/CAN_bus. (accessed: 17.11.2021).