16.11.2017

Bård Barstad, Halvard Yri Adriaenssens og Fredrik Kartevoll

Prosjekt del 3

Arduino / Teensy 3.6 og CAN-bus

Innholdsfortegnelse

[Introduksjon 2](#_Toc498528427)

[Metode 3](#_Toc498528428)

[Dual CAN-bus adapter for Teensy 3.6 3](#_Toc498528429)

[PCAN-USB FD adapter 3](#_Toc498528430)

[Sende og motta CAN-meldinger 3](#_Toc498528431)

[Void 3](#_Toc498528432)

[Loop 4](#_Toc498528433)

[Sende CAN-melding 4](#_Toc498528434)

[PcanView 5](#_Toc498528435)

[Sendre melding 5](#_Toc498528436)

[Styre LED med meldings ID 6](#_Toc498528437)

[Styre LED med innhold i melding 6](#_Toc498528438)

[Rapportere verdier fra IMU via CAN-bus 7](#_Toc498528439)

[Kontakt mellom Teensyer 8](#_Toc498528440)

[Diskusjon 9](#_Toc498528441)

[PcanView 9](#_Toc498528442)

[Sende CAN-meldinger 9](#_Toc498528443)

[Resultat 11](#_Toc498528444)

[Kilder 12](#_Toc498528445)

# Introduksjon

I denne delen av prosjektet skal vi bruke CAN-bus. Vi skal sende og motta meldinger mellom en PCAN-USB FD adapter og en Teensy 3.6 ved hjelp av en Dual CAN-bus adapter for Teensy 3.6. CAN (Controller Area Network) feiltollerant og robust nettverksstandard som brukes veldig mye i bil bygging. Fordelene med er mange, for eksempel:

* Rask kommunikasjon mellom noder uten behov for en ekstern datamaskin.
* Prioritering gitt av meldings ID, lav ID gir høy prioritet.
* Feiltolerant design.

Vi skal også ta i bruk en Teensy 3.6. Teensy 3.6 er en USB-basert mikrokontroller som er Arduino-kompatibel. For å programmere Arduinoprogrammer på Teensyen laster vi ned tilleggsbiblioteket Teensyduino [1]. Vi skal også bruke biblioteket FlexCAN for å få tilgang til CAN funksjonene [2].

Dual CAN-bus adapteren inneholder to transivere, en for Can0 og en for Can1. Adapteren inneholder også en 120Ω terminerings motstand [3].

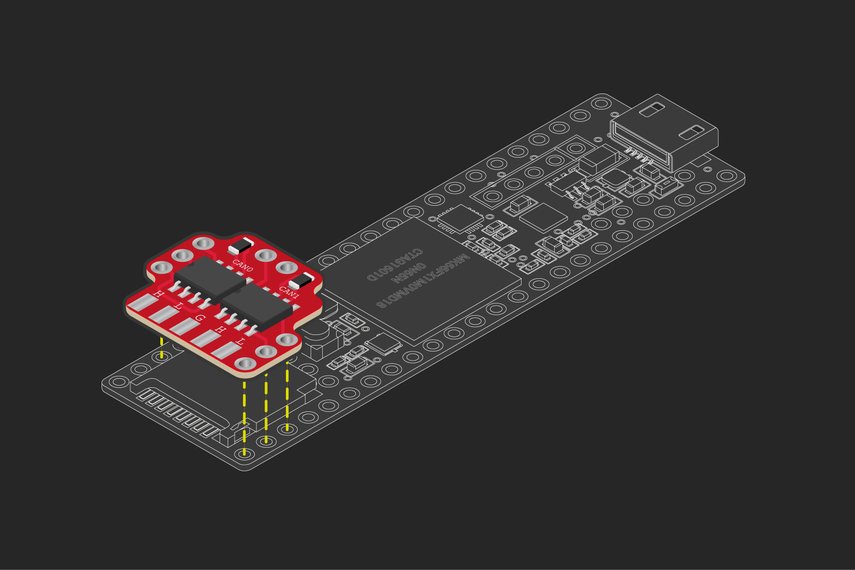
# Metode

Vi laster ned FlexCAN biblioteket og installerer det i Arduino mappen. I dette biblioteket følger det med eksempelprogrammer og CAN-bus funksjoner som vi skal bruke senere. Vi laster inn CANTest programmet og sjekker om det kompilerer. Fordi vi har et lignende bibliotek med samme navn som ikke har de funksjonene vi trenger. Vi må derfor inn å slette dette før programmet vil kompilere. Biblioteket som slettes ligger under: (C:) > Programfiler (x86) > Arduino > hardware > teensy > avr > libraries.

## Dual CAN-bus adapter for Teensy 3.6

Når vi har et program som kompilerer, skal vi koble til CAN-bus adapteren. Vi velger å lodde på pinner på adapteren, og koble den sammen med Teensy brikken ved å bruke ledninger. Dette kan føre til forstyrrelser i ledningene, men vi slipper da å ha permanent festet adapteren til Teensy brikken. Teensyen og CAN-bus adapteren kobles som vist i figur x. Så må det også kobles til jord, og Vcc på 3.3V som er oppgitt i databladet [4].

Kobling Dual CAN-bus adapterpå Teensy 3.6



## PCAN-USB FD adapter

Vi skal nå ta i bruk en Peak PCAN-USB FD adapter for å kunne motta og sende genererte CAN-bus signaler via USB. For å ta i bruk denne trenger vi softwaren slik at vi kan ta i bruk Peak PCAN-USB CAN-grensesnittet for PC. Setup til driveren lastes ned og installeres fra nettsiden [5]. Vi bruker brukermanualen til PCAN-USB, CAN-grensesnitt for USB, som veiledning for å sette opp PcanView [6].

## Sende og motta CAN-meldinger

Når vi skal til å sende og motta CAN-meldinger, velger vi å bruke eksempelprogrammet CANTest og bygger videre på det.

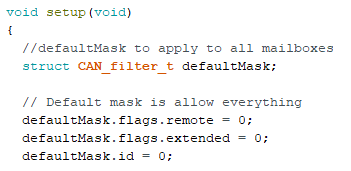
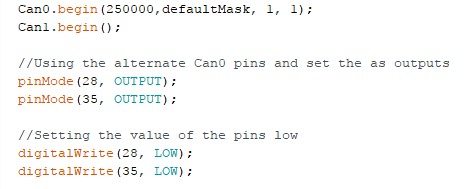
### Void

For å få det til å funke må vi endre på deler av koden. Vi legger til CAN-filteret «defaultMask» i «setup» (Figur x). Vi bestemmer oss for å bruke Can0, og kobler opp kretsen og koder deretter. Vi kunne også brukt Can1 ved å gjøre noen få endringer i koden og i kretsen. Vi bruker «begin» funksjonen og setter baud raten til Can0 til 250kbit/s. Vi aktiverer «defaultMask» filteret for Can0. Vi setter Tx = 1 og Rx = 1 for å bruke de alternative «receive» and «transmit» pinnene [7].

Vi har koblet Dual CAN-bus adapteren til de alternative Can-portene, derfor setter vi pin 28 og 35 til output. Vi setter de samme pinnene lav, slik at de kan kommunisere med PCAN (Figur x+2)

Setup - defaultMask

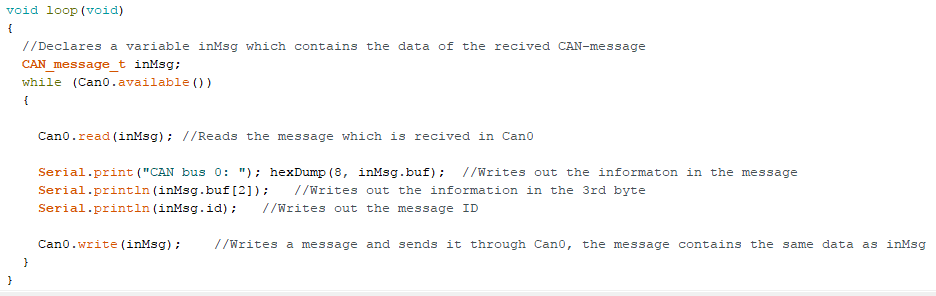
Setup - kode



### Loop

Når de nødvendige endringene er gjort i «setup», går vi over i «loop». Vi deklarerer en variabel «inMsg» som skal inneholde dataene som blir motatt fra PeakView. Vi kjører en «while» loop som sjekker om vi mottar data i Can0, når dette er sant vil «while» loopen kjøres. Først vil meldingen fra Can0 leses inn i variabelen «inMsg». Deretter vil vi printe ut innholdet i alle bit’ene som sendes. Maksimalt kan det sendes 8 byte i en melding når FlexCAN biblioteket brukes, dette står i README filen [2]. I neste linje printer vi ut dataen som befinner seg i det 3. bytet, så vil vi printe ID’en til meldingen. Til slutt vil vi sende ut samme melding som vi har mottatt (Figur x+3). Vi kompilerer og laster opp programmet til Teensyen.

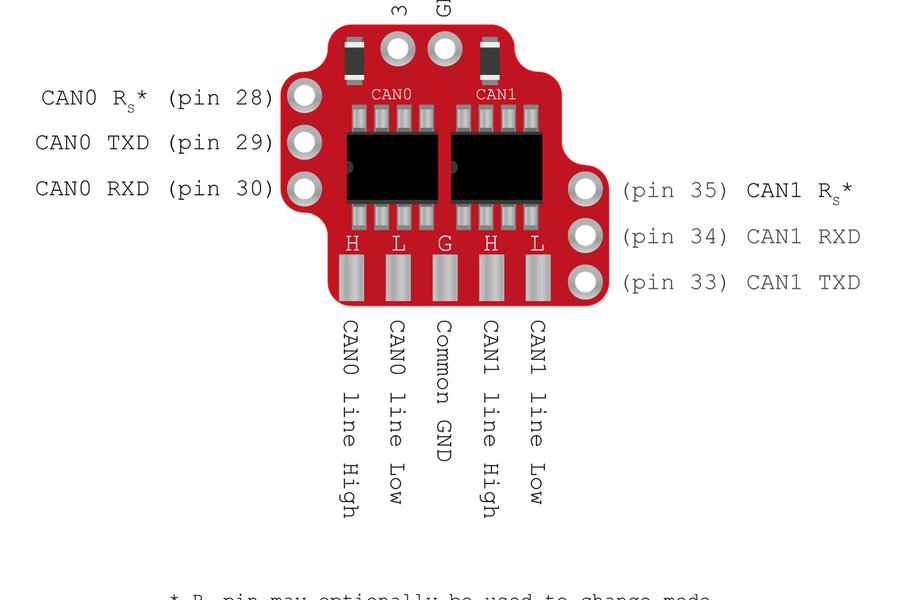
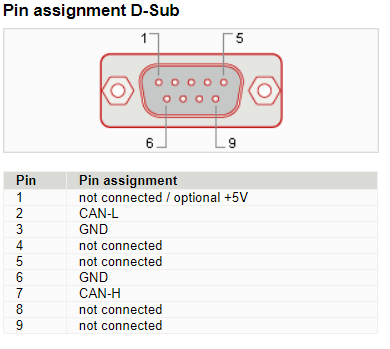
Loop - kode



## Sende CAN-melding

Vi vil nå koble sammen PCAN og Teensy 3.6 for å sende meldinger ved hjelp av CAN-bus. Dette gjøres ved å koble Can0 line High fra CAN adapteren inn på CAN-H på PCAN og Can0 line Low fra CAN adapteren inn på CAN-L på PCAN (Figur x og x).

Dual CAN-bus adapter koblingsskjema



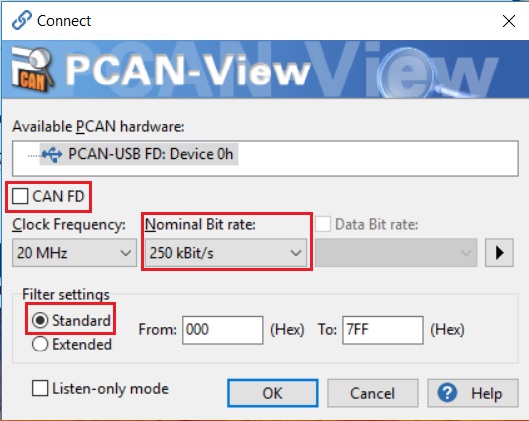
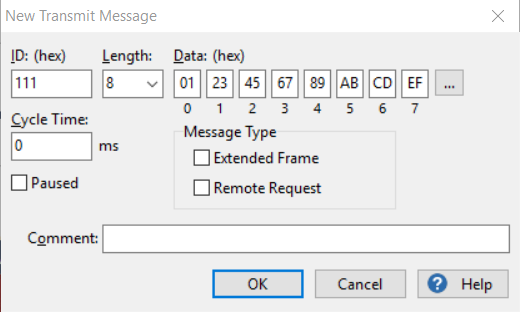
PCAN koblingsskjema

### PcanView

Vi vil nå generere en melding i PcanView som skal leses av Arduino-programmet og returneres til avsenderen. Når programmet startes vil menyen i figur x dukke opp. Der velger i PCAN-kabelen, sørger for at «CAN FD» er umerket, vi sjekker at bit raten er riktig og at vi har valgt standard ramme til CAN-ID’en. For å generere en melding trykker vi på «New message», da kommer menyen i figur x frem. Der kan vi velge ID, lengde på meldingen (fra 0 til 8 byte), data og syklus tid. Vi setter lengden til meldingen til 8 og gir den en tilfeldig ID og innhold. Vi setter syklus tiden til 0ms, da kan vi sende meldingen så ofte og så mange ganger vi vil ved å trykke på mellomromstasten.

PcanView – Velge hardware

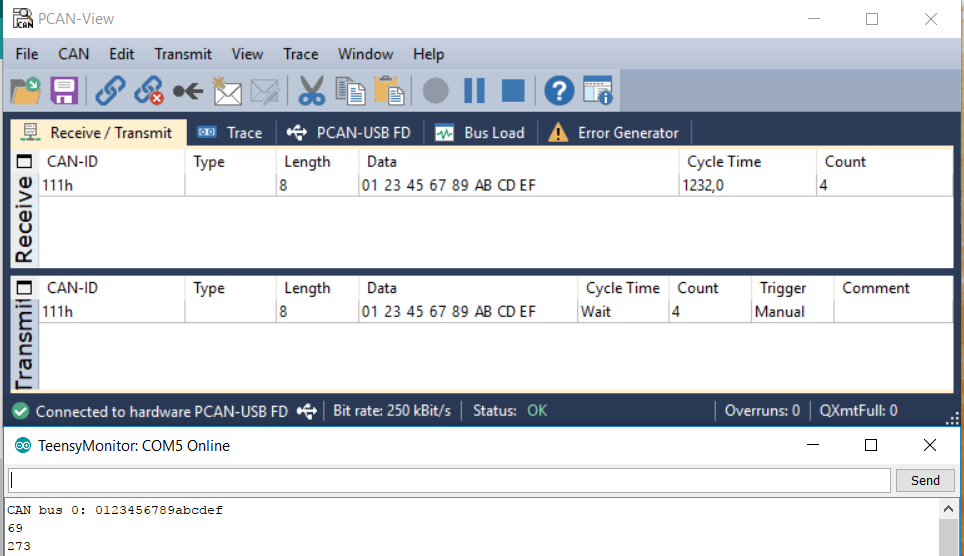
PcanView – Ny melding



### Sendre melding

Nå kan vi sende meldingen vi har laget. Hvis vi åpner «Seriell overvåker» i Arduino, så kan vi se at det printes en melding hver gang vi sender en melding fra PcanView. I overvåkningsvinduet ser vi at all dataen printes først, så printes innholdet i det 3. bytet (0x45 = 69). Tilslutt printes meldings ID’en (0x111 = 273). Som vi ser i figur x, vil meldingen vi sender også mottas i PcanView, så Arduino programmet fungerer som vi ville.

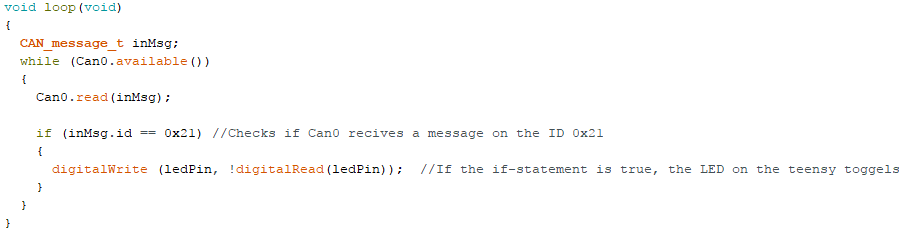
PcanView recive and transmitt



## Styre LED med meldings ID

Vi vil styre LED på Teensy 3.6 brettet ved hjelp av meldings ID’en. Når Teensy brettet leser en melding som har ID 0x21 skal LED’en toggles. For å få til dette deklarerer vi en konstant «ledPin» som er lik 13. 13 er adressen til LED’en på Teensy 3.6. I «setup» definerer vi «ledPin» som en output, og setter den høy, slik at den skrus på når programet lastes inn. I «loop» setter vi inn en if-setning som sjekker om den mottatte CAN-bus meldingen har ID 0x21. Hvis dette er sant vil «ledPin» inverteres (figur 3x+y).

Styre LED med meldings ID kode

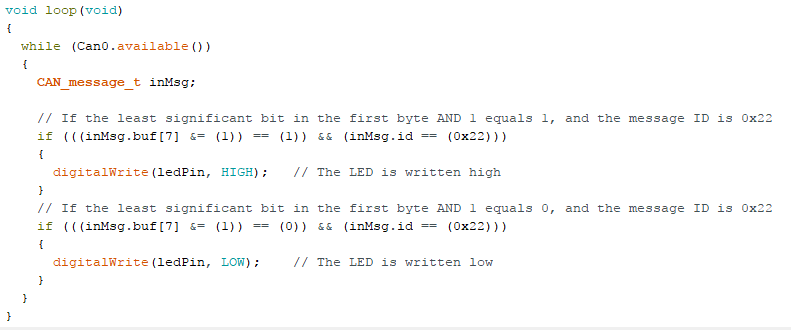


## Styre LED med innhold i melding

Vi vil styre LED på Teensy 3.6 brettet ved hjelp av innholdet i meldingen. Når det minst signifikante bit i første byte er 1 vil «ledPin» settes høyt, hvis det minst signifikante bit i første byte er 0 vil «ledPin» settes lavt. Vi bruker IF-setninger til å sjekke om «inMsg.buf[7]» OG’et med et byte som er xxxxxxx1 er lik 1, hvis dette stemmer og meldingen kommer i fra ID’en 0x22 skal «ledPin» settes høy. Hvis «inMsg.buf[7]» OG’et med et byte som er xxxxxxx1 er lik 0, skal «ledPin» settes lavt. «inMsg.buf[7]» inneholder kun det første bytet som mottas i meldingen (figur).

**Styre LED med innhold i melding**

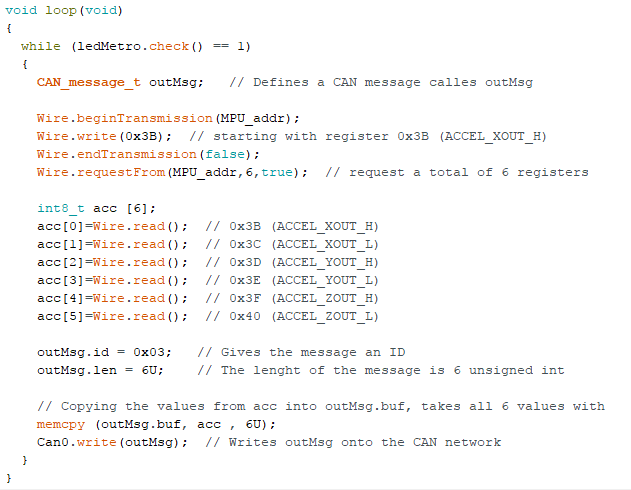
Styre LED med innhold i melding kode



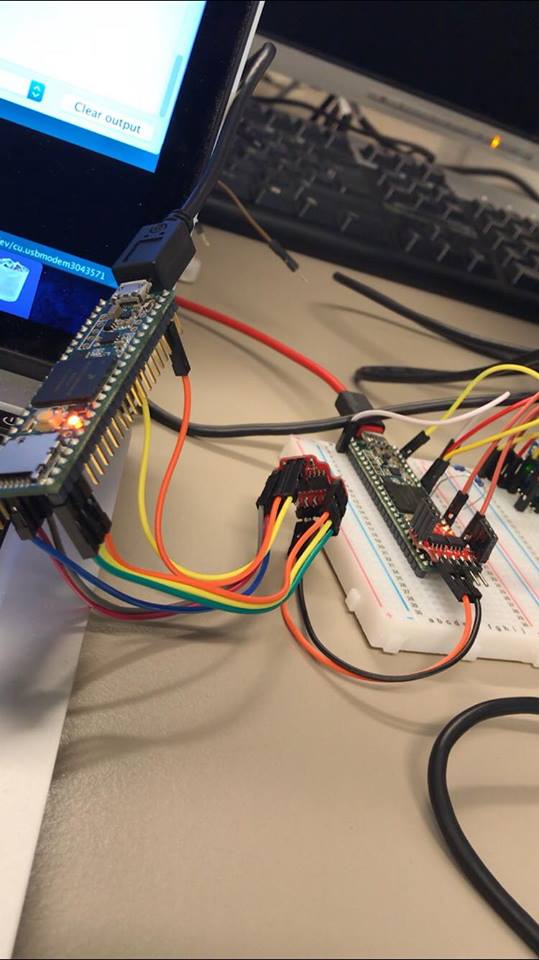
## Rapportere verdier fra IMU via CAN-bus

Vi vil rapportere verdier fra akselerometeret på en IMU 6050 ved å benytte meldings-ID 0x20. Meldingen skal sendes med en fast rate på 1 Hz. For å ha kontroll på meldingsraten bruker vi Metro biblioteket [8]. Vi inkluderer «Metro.h» for å få tilgang, så definerer vi en metro variabel «ledMetro» som vil være sann en gang per 1000ms. I «loop» lager vi en while-løkke som vil kjøre når (ledMetro.check() == 1). Vi velger å bruke deler av kode som er forklart tidligere for å lese akselerasjonen i fra IMU’en. Akselerasjonen i x-, y- og z-retning leses inn i ett int8\_t array som vi kaller «acc». Akselerasjonsdataene er egentlig på 16bit i hver retning, men må deles i to for å kunne sendes over CAN-bus. Vi definerer en CAN-melding som vi kaller «outMsg». Vi gir denne ID = 0x03, alle ID’er mellom 0x00 og 0x7FF kan brukes her. Vi setter lengden til å være 6U, fordi vi har 6 bytes med data som skal sendes. Vi kunne sendt 2 bytes til i denne meldingen. Vi bruker «memcpy» funksjonen til å si at vi skal ha innholdet i «acc» inn i «outMsg.buf», og at vi skal ha med 6 bytes i fra «acc». Deretter bruker vi Can0 til å skrive «outMsg». (Figur jaja)

Rapportere verdier fra IMU via CAN-bus



## Kontakt mellom Teensyer

Vi vil sende signaler fra vår Teensy til en annen Teensy via CAN-bus. To teensyer kobles sammen via Can0 pinnene via Dual CAN-bus adapterne. CAN-H kobles på CAN-H, og CAN-L kobles på CAN-L slik som før (figur xxx). Programmet som er lastet inn på veksler hvert sekund mellom å sende ut en melding med ID 0x21 og 0x22. Dette skjer ved bruk av en variabel «i» som inkrementeres for hver runde den kjøres. Vi bruker if-setninger som sjekker om (i%2 == 1) eller (i%2 == 0). Altså sjekkes det om «i» er et oddetall(=1) eller partall(=0). Hvis «i» er et oddetall sendes en melding med ID 0x21 ut, hvis «i» er et partall blir ID’en til meldingen 0x22. Programmet som tar imot meldingen vil sette LED’en høy hvis ID’en er 0x21, og lav hvis ID’en er 0x22. LED’en vil da blinke.

Kobling mellom to Teensyer

# Diskusjon

## PcanView

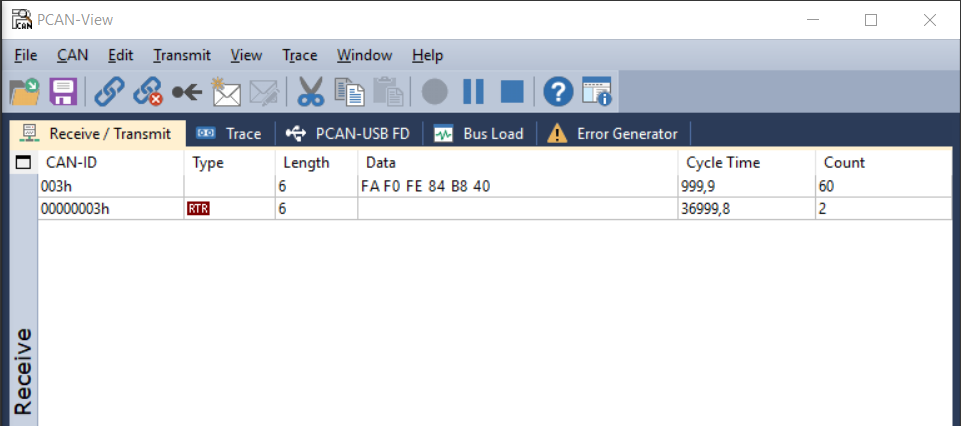
Det første vi må velge er bit rate, bit raten bestemmer hvor mye og hvor fort data sendes i et CAN-bus nettverk. Bit raten kan variere fra 1Mbit/s til 5kbit/s. For å ha en raskere bit rate må man ofre lengden på bussen. Vi velger en bitrate på 250kbit/s, fordi det er standard hastigheten i FlexCAN.

Vi må så velge spekteret av CAN ID’er som vi kan ta imot. Alle meldinger som sendes på ett CAN nettverk vil ha en egen ID. Vi kan velge mellom standard rammer (11 bit ID’er) eller utvidede rammer (29 bit ID’er). En 11 bits ID vil si at vi har 211 = 2048 mulige ID’er, en 29 bits ID vil si 229 = 536 870 912 mulige ID’er [9]. På grunn av at vi kun skal ha ett lite nettverk velger vi å bruke standard rammer. CAN-bus fungerer slik at meldingene prioriteres ut i fra ID’en, hvis meldingen har en lav ID vil den prioriteres over en melding men høy ID.

## Sende CAN-meldinger

Vi fikk noen problemer med å skrive verdier fra akselerometeret ut på CAN-nettverket. Når vi prøvde å sende hele meldingen med all dataen til akselerasjonen, så ville PcanView kun ta imot et par meldinger før den tok imot to meldinger som kom opp som «RTR», eller «retry». Antallet meldinger vi klarte å sende varierte i fra null til maks ti. Etter å ha prøvd litt forskjellig fant vi ut at hvis vi reduserte antallet bytes vi tok med i «outMsg.buf» til 5, så kunne vi motta mange flere meldinger før fikk to «meldinger». Vi kunne da sende mellom 30 og 150 meldinger, det så ut som figur zz.

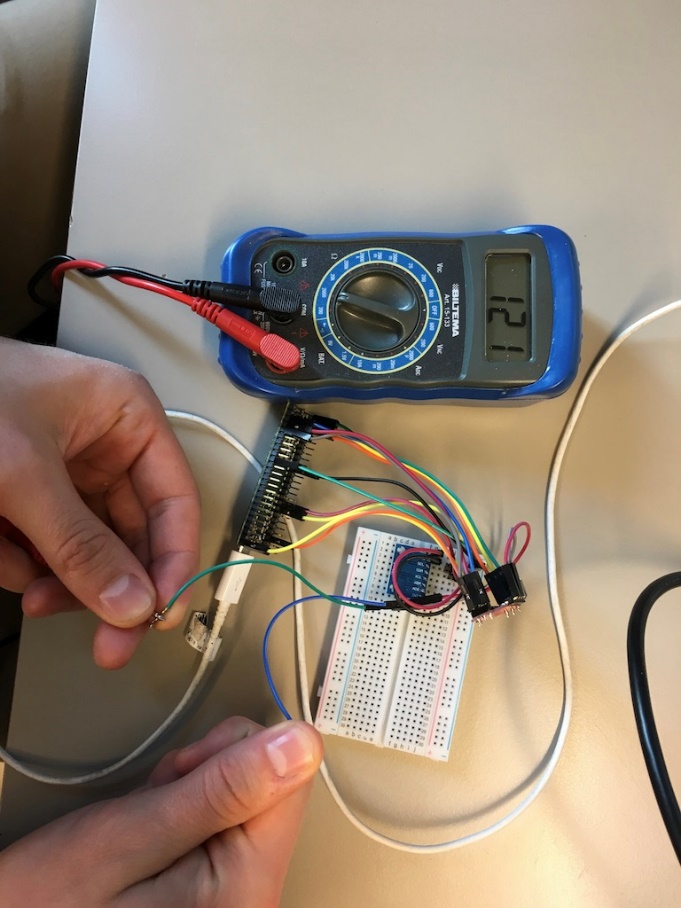
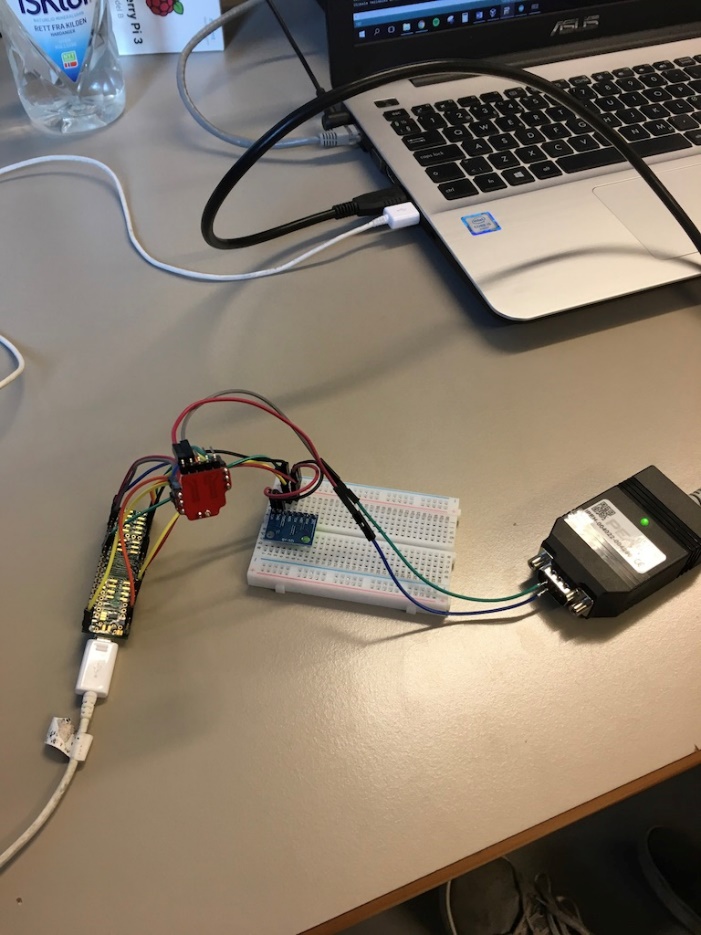
Retry meldingene vi fikk i PcanView



Vi mottok fortsatt data med lengde 6 bytes, men den siste byten var som oftest 0x00. Vi sjekket termineringsmotstanden til Dual CAN-bus adapteren og den var i orden (figur). Vi testet også å kjøre programmet uten IMU-brettet tilkoblet, men vi fikk samme resultater bare uten gyroverdiene.

Sjekker termineringsmotstanden i Dual CAN-bus adapteren

Oppkobling av krets



Vi klarte ikke eliminere feilen, så vi regner med at det enten har noe med støy å gjøre, eller så er det en dårlig kobling.

# Resultat

Vi mener at vi har svart bra på oppgavene som ble gitt. Vi har koblet opp og behandlet utstyret vi brukte på en fornuftig måte. Vi har tatt i bruk CAN-bus til å sende og motta meldinger. Vi har programmert Teensyen til å reagere på meldings ID’er og meldingsinnhold. Vi har regelmessig distribuert data fra en IMU 6050, ut på et CAN-nettverk. Vi har hatt kommunikasjon mellom to mikrokontrollere.

# Kilder

|  |  |
| --- | --- |
| [1] | «www.pjcr.com,» PJCR, [Internett]. Available: https://www.pjrc.com/teensy/teensyduino.html. [Funnet 30 Oktober 2017]. |
| [2] | c. m. p. t. t. F. P. b. teachop, 1 August 2017. [Internett]. Available: https://github.com/collin80/FlexCAN\_Library/. [Funnet 30 Oktober 2017]. |
| [3] | Tindie, «www.tindie.com,» Tindie, INC, 2017. [Internett]. Available: https://www.tindie.com/products/Fusion/dual-can-bus-adapter-for-teensy-35-36/. [Funnet 6 November 2017]. |
| [4] | Texas Instruments, «www.ti.com,» Texas Instruments, Juli 2015. [Internett]. Available: http://www.ti.com/lit/ds/symlink/sn65hvd230.pdf. [Funnet 31 Oktober 2017]. |
| [5] | «www.peak-system.com,» PEAK-System Technik GmbH, 26 Oktober 2017. [Internett]. Available: https://www.peak-system.com/PCAN-USB-FD.365.0.html?&L=1. [Funnet 30 Oktober 2017]. |
| [6] | PEAK-SYSTEM, «www.peak-system.com,» 27 Januar 2017. [Internett]. Available: https://www.peak-system.com/produktcd/Pdf/English/PCAN-USB\_UserMan\_eng.pdf. [Funnet 30 Oktober 2017]. |
| [7] | «www.pjcr.com,» Freescale Semiconductor, Inc, 2 Mai 2015. [Internett]. Available: https://www.pjrc.com/teensy/K66P144M180SF5RMV2.pdf. [Funnet 14 November 2017]. |
| [8] | T. O. Fredericks, 25 Juli 2014. [Internett]. Available: https://www.pjrc.com/teensy/td\_libs\_Metro.html. [Funnet 6 November 2017]. |
| [9] | «www.manual.xanalyser.com,» Warwick Control Technologies, 24 Januar 2017. [Internett]. Available: https://manual.xanalyser.com/CAN%20Frame%20Message%20Format.html. [Funnet 13 November 2017]. |