16.11.2017

Gruppe 3: Bård Barstad, Halvard Yri Adriaenssens og Fredrik Kartevoll

Rapport, gruppearbeid MAS234 høst 2017

Innhold

[Del 1 Mikrokontrollerkretser 3](#_Toc498546196)

[Introduksjon 3](#_Toc498546197)

[Autodesk Eagle 4](#_Toc498546198)

[Spenningsregulator 4](#_Toc498546199)

[Funksjonstest av spenningsregulator 5](#_Toc498546200)

[Atmega168 5](#_Toc498546201)

[Reset-funksjon 6](#_Toc498546202)

[Atmel-ICE 7](#_Toc498546203)

[LED-diode 8](#_Toc498546204)

[Styre LED ved bruk av Atmega168 9](#_Toc498546205)

[Oppkobling 9](#_Toc498546206)

[AtmelStudio 9](#_Toc498546207)

[Fuses 10](#_Toc498546208)

[Programmering i AtmelStudio 11](#_Toc498546209)

[Krysskompilering 11](#_Toc498546210)

[Overgang til Arduino 12](#_Toc498546211)

[PWM – Pulsbredde modulasjon 13](#_Toc498546212)

[PWM oppsett 13](#_Toc498546213)

[PWM modus og frekvens 13](#_Toc498546214)

[LED styrt med PWM med konstant dutycycle 14](#_Toc498546215)

[Jevnt blinkende LED – Softblink 14](#_Toc498546216)

[Tilpasning av lysnivå for øyet 15](#_Toc498546217)

[Knapp for styring av lysdiode 16](#_Toc498546218)

[Avbrudd - pin change interrupt 16](#_Toc498546219)

[Debounce 17](#_Toc498546220)

[ADC – analog til digital konvertering 17](#_Toc498546221)

[Konfigurasjon av ADC 19](#_Toc498546222)

[Konklusjon 20](#_Toc498546223)

[Del 2 Arduino og I2C 21](#_Toc498546224)

[Del 3 Arduino/Teensy 3.6 og CAN-bus 22](#_Toc498546225)

[Introduksjon 22](#_Toc498546226)

[Metode 23](#_Toc498546227)

[Dual CAN-bus adapter for Teensy 3.6 23](#_Toc498546228)

[PCAN-USB FD adapter 23](#_Toc498546229)

[Sende og motta CAN-meldinger 23](#_Toc498546230)

[Void 24](#_Toc498546231)

[Loop 24](#_Toc498546232)

[Sende CAN-melding 24](#_Toc498546233)

[PcanView 25](#_Toc498546234)

[Sendre melding 25](#_Toc498546235)

[Styre LED med meldings ID 26](#_Toc498546236)

[Styre LED med innhold i melding 26](#_Toc498546237)

[Rapportere verdier fra IMU via CAN-bus 27](#_Toc498546238)

[Kontakt mellom Teensyer 28](#_Toc498546239)

[Diskusjon 29](#_Toc498546240)

[PcanView 29](#_Toc498546241)

[Sende CAN-meldinger 29](#_Toc498546242)

[Resultat 31](#_Toc498546243)

[Del 4 Raspberry Pi 3/Buildroot og CAN 32](#_Toc498546244)

[Introduksjon 32](#_Toc498546245)

[Metode 33](#_Toc498546246)

[Ubuntu og VirtualBox 33](#_Toc498546247)

[Buildroot 33](#_Toc498546248)

[Installasjon av delvis ferdig konfigurert oppsett 34](#_Toc498546249)

[SD-kort 34](#_Toc498546250)

[Koble opp hardware 35](#_Toc498546251)

[Konfigurasjon av ethernettilkobling 35](#_Toc498546252)

[Kommunikasjon over CAN-bus 36](#_Toc498546253)

[Diskusjon: 38](#_Toc498546254)

[Kilder 40](#_Toc498546255)

Del 1 Mikrokontrollerkretser

# Introduksjon

# Autodesk Eagle

Eagle er et program utviklet av Autodesk for tegning og konstruksjon av elektriske kretser. Som studenter kan vi registrere oss og få gratis tilgang til programmet, dog i en begrenset versjon.

Vi lastet ned og installerte Eagle, og gjorde oss kjent med programmet og dets funksjoner.

# Spenningsregulator

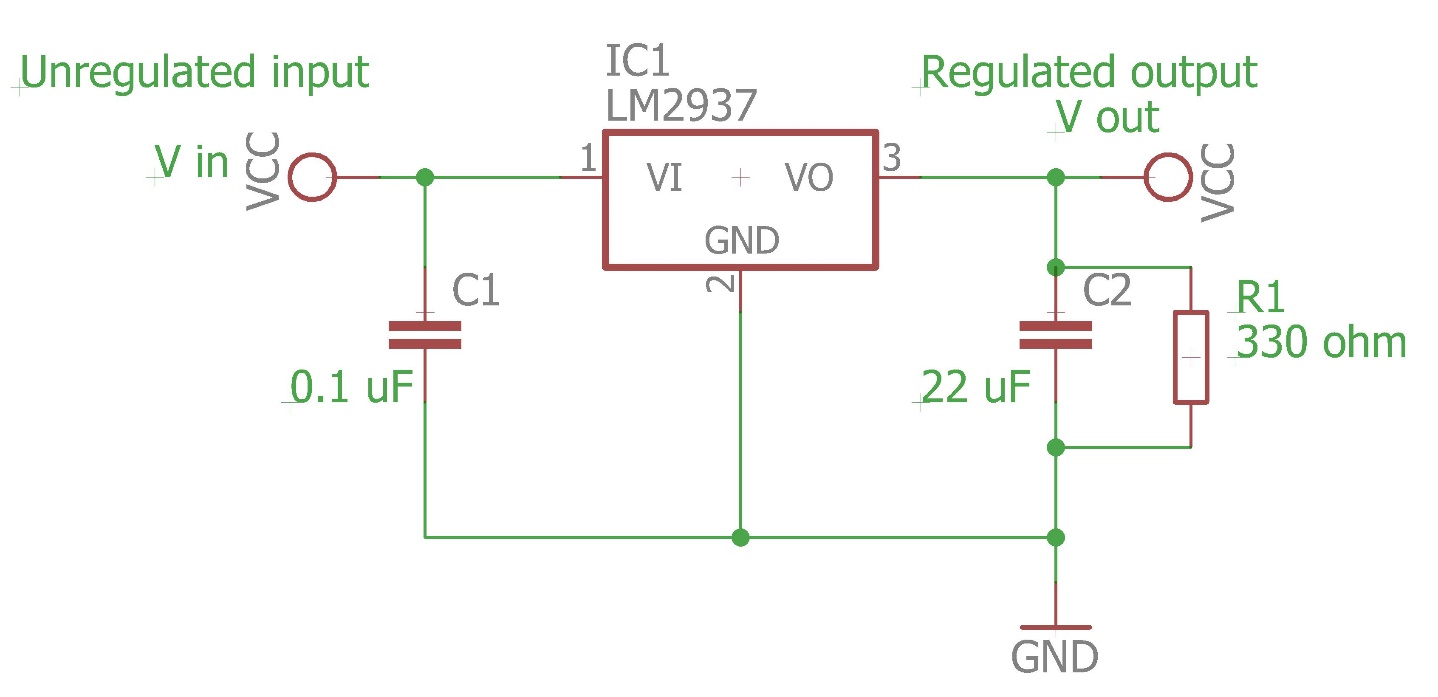
Mikrokontrollere er sårbare for feil dersom kvaliteten på spenningsforsyningen er dårlig. Derfor er en pålitelig, lineær spenningsregulator en av de viktigste komponentene i en slik krets.

Til prosjektet vårt brukte vi en 5V lineær regulator, LM2931T. I følge databladet krever denne kondensatorer som ligger nærme regulatoren, mellom jord og input/output [1]. Størrelsene på kondensatorene var oppgitt til å være 0,1uF og 100uF. Fordi vi ikke hadde 100uF tilgjengelig under prosjektet, benyttet vi 22uF.

Spenningsregulatoren krever en last på minimum 5mA for å levere en stabil spenning, samtidig som maksimal last er 100mA. Vi koblet derfor inn en motstand på 330.

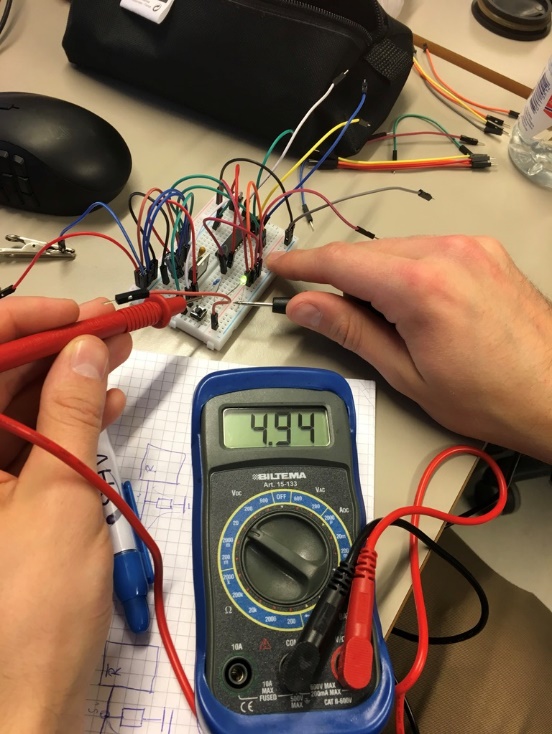
Før vi koblet opp spenningsregulatoren vår, tegnet vi inn kretsen i Eagle og fikk denne godkjent av faglærer (figur 1).

Kretsen tegnet med spenningsregulatoren



Figur

## Funksjonstest av spenningsregulator

For å sikre oss at spenningsregulatoren leverte stabil spenning, ønsket vi å funksjonsteste den med et multimeter. I databladet står det beskrevet at spenningen inn på regulatoren må være 6V-26V. Vi brukte en lab strømforsyning for å simmulere en ustabil spenning inn ved å regulere strømforsyningen kontinuerlig fra 6-15V. Resultatet var en helt stabil spenning på 4,94V ut fra spenningsregulatoren (Figur 2).

Figur

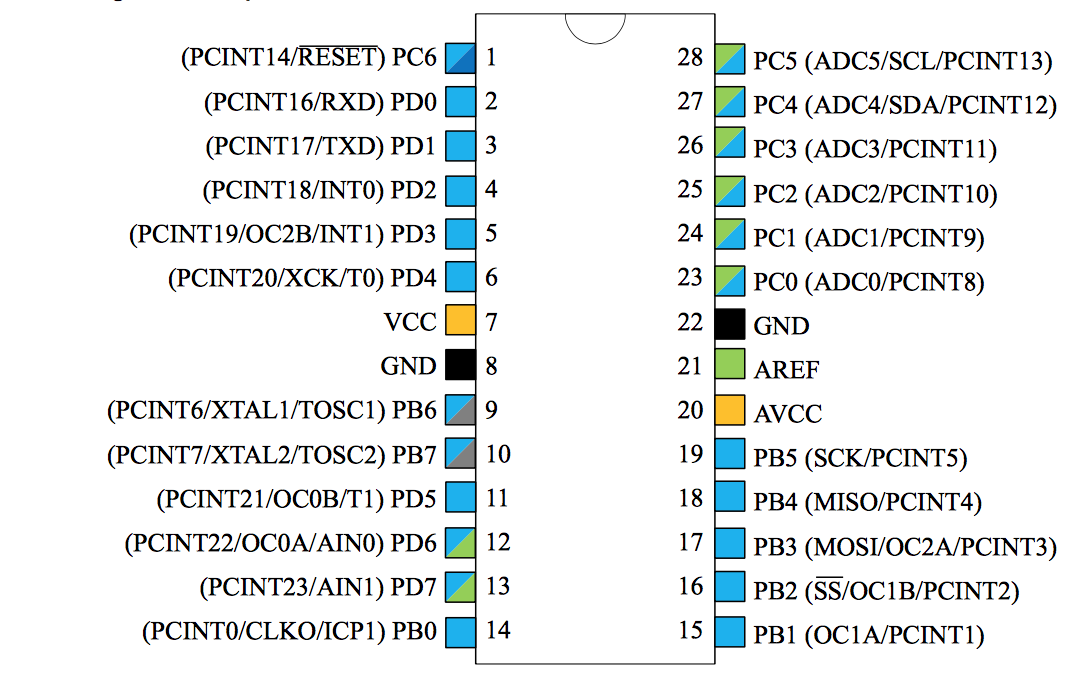
Tester spenningen ut fra spenningsregulatoren

# Atmega168

Mikrokontrollere er små, integrerte kretser med elektroniske komponenter slik som prosessor, minne og diverse I/O-porter. De fungerer ofte som små, dedikerte datamaskiner.

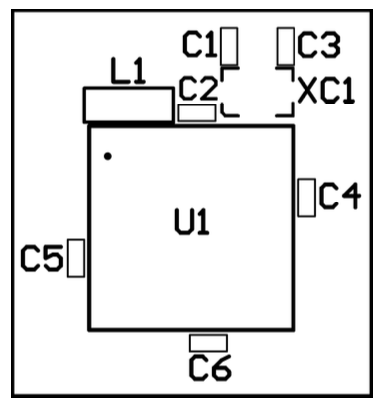
I dette prosjektet har vi benyttet oss av en Atmega168-mikrokontroller. Skjema for oppkobling og pin-konfigurasjon fant vi i databladet (figur3) [2].

Pin-konfigurasjon for en Atmega168-mikrokontroller



Figur

## Støy

Når man kobler til Vcc og jord på mikrokontrolleren, får man en loop med høyt strømtrekk. Dette strømtrekket blir høyere jo flere I/O-er som er i bruk, noe som resulterer i at loopen i større grad vil fungere som en antenne med støy til resten av kretsens komponenter.

Annbefalt oppkobling med kondensatorer kundt mikrokontrolleren

Figur

For å hindre dette, anbefales det i databladet å koble inn kondensatorer mellom Vcc og jord. Det spesifiseres at disse må kobles så fysisk nærme mikrokontrolleren som mulig, slik at loopen vår med høyt strømtrekk blir så liten som mulig. Anbefalt størrelse på kondensatorene er 0,1uF (Figur 4).

## Reset-funksjon

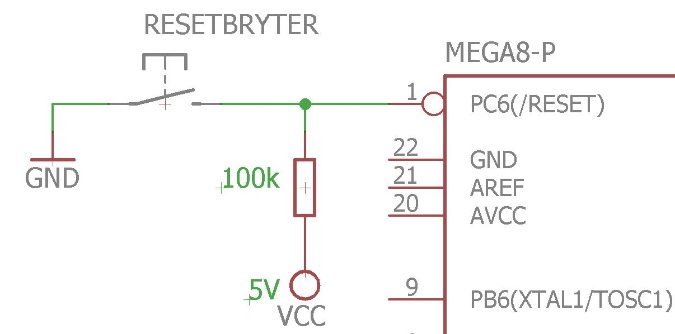
Resetbryteren har som funksjon å initialisere all I/O, og sette program-counteren vår til null. I praksis lar den oss ”restarte” mikrokontrolleren.

Reset-funksjonen i Atmega168 er aktiv lav. Det vil si at vi aktiverer den når koblingen til pinnen går til jord. Dette gjøres med en ekstern kobling.

For å hindre/redusere støy som kan aktivere reset-funksjonen, har mikrokontrolleren en intern ”pull-up” motstand. Databladet spesifiserer at den interne motstanden kan være utilstrekkelig i miljøer med mye støy, og at det vil resultere i sporadiske aktiveringer av reset-funksjonen.

For å sikre oss mot uventede resetter, koblet vi opp en ekstern ”pull-up” reset. Denne står med konstant 5V inn på PC6, og vil gå lav dersom resetbryteren blir trykket (figur 5). Ved å koble inn en motstand på 100k, så:

Ekstern ”pull-up” reset

* Hindrer vi sporadiske resetter.

Figur

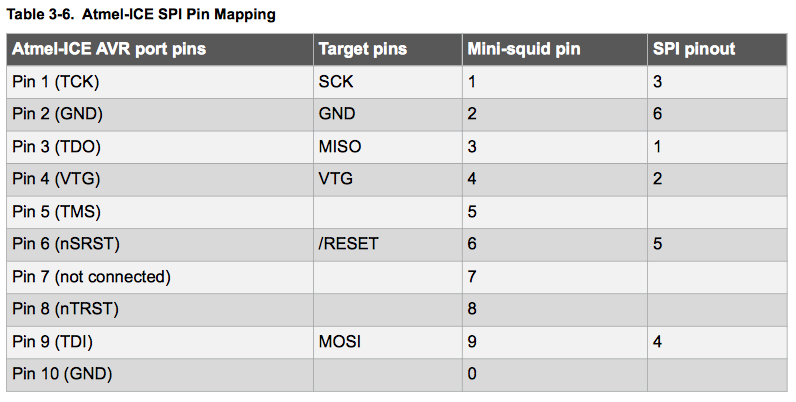
* Hindrer vi kortslutning mellom Vcc og jord når bryteren blir trykt.
* Reduserer vi strøm- og spenningspeaks fra kondensatorene når bryteren blir trykt.

# Atmel-ICE

Atmel-ICE er et redskap for å programmere Atmel AVR-mikrokontrollere, ved å fungere som en overgang mellom kontrolleren og PC-en [3]. Tabell for oppkobling fant vi i databladet, under avsnittet for SPI-target (figur 6) [4].

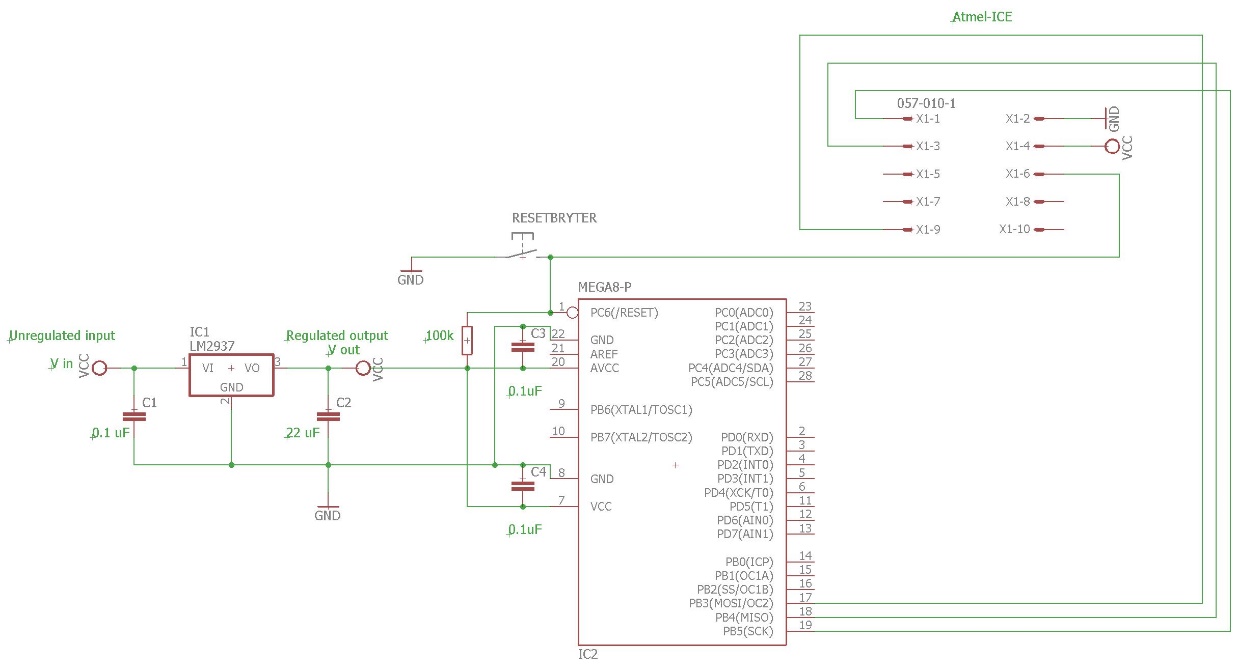
SCK, MISO og MOSI er alle pinner under PB på mikrokontrolleren, mens ground og VTG er henholdsvis jord og spenningsforsyning.

Atmel-ICE SPI Pin mapping



Figur

Koblingsskjema med Atmel-ICE



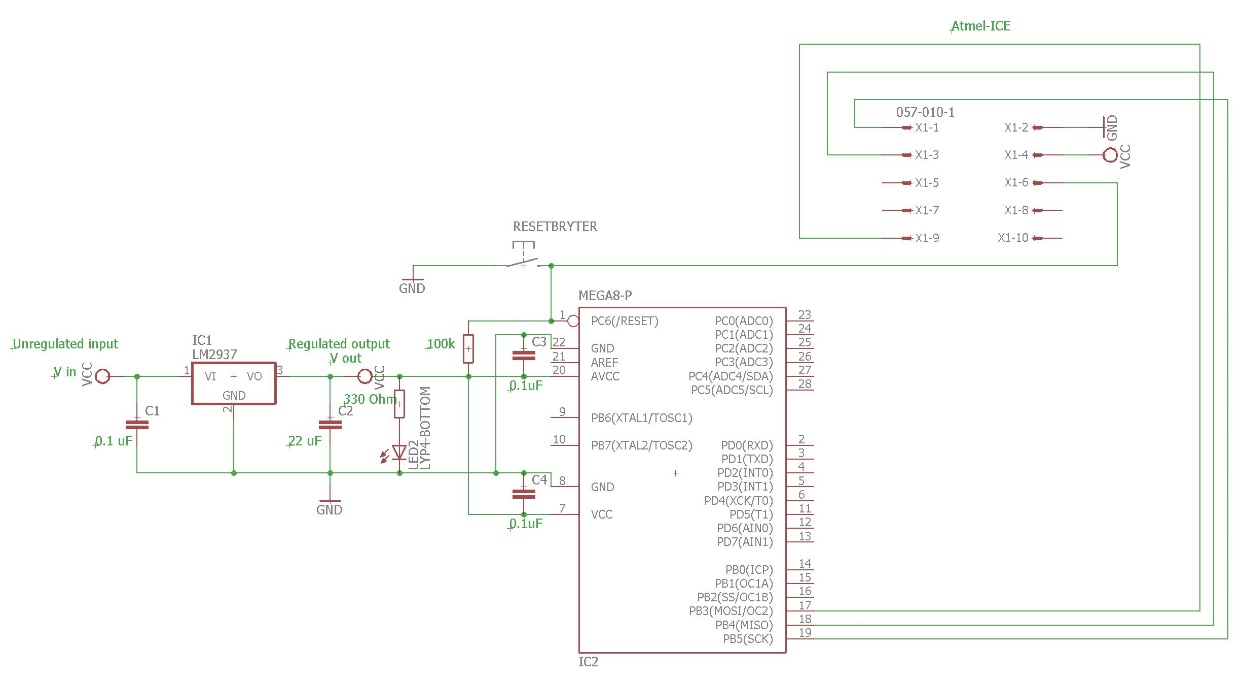
Figur

# LED-diode

Når man skal bruke en LED, er det viktig at man ikke overskrider spenning og strøm merkinger, slik at man ikke ødelegger komponenten. Vi brukte en Kingbright L7104GC LED. I LEDen sitt datablad leste vi av maksimal driftsstrøm til å være 25mA og driftsspenning til å være 5VDC [5]. For å begrense strømgjennomgangen i dioden, koblet vi en 330 i serie med denne.

Vi koblet LEDen mellom utgangen på spenningsregulatoren og jord (Figur 8). Denne skulle lyse for å indikere at vi hadde ”power on”.

Koblingsskjema med LED



Figur

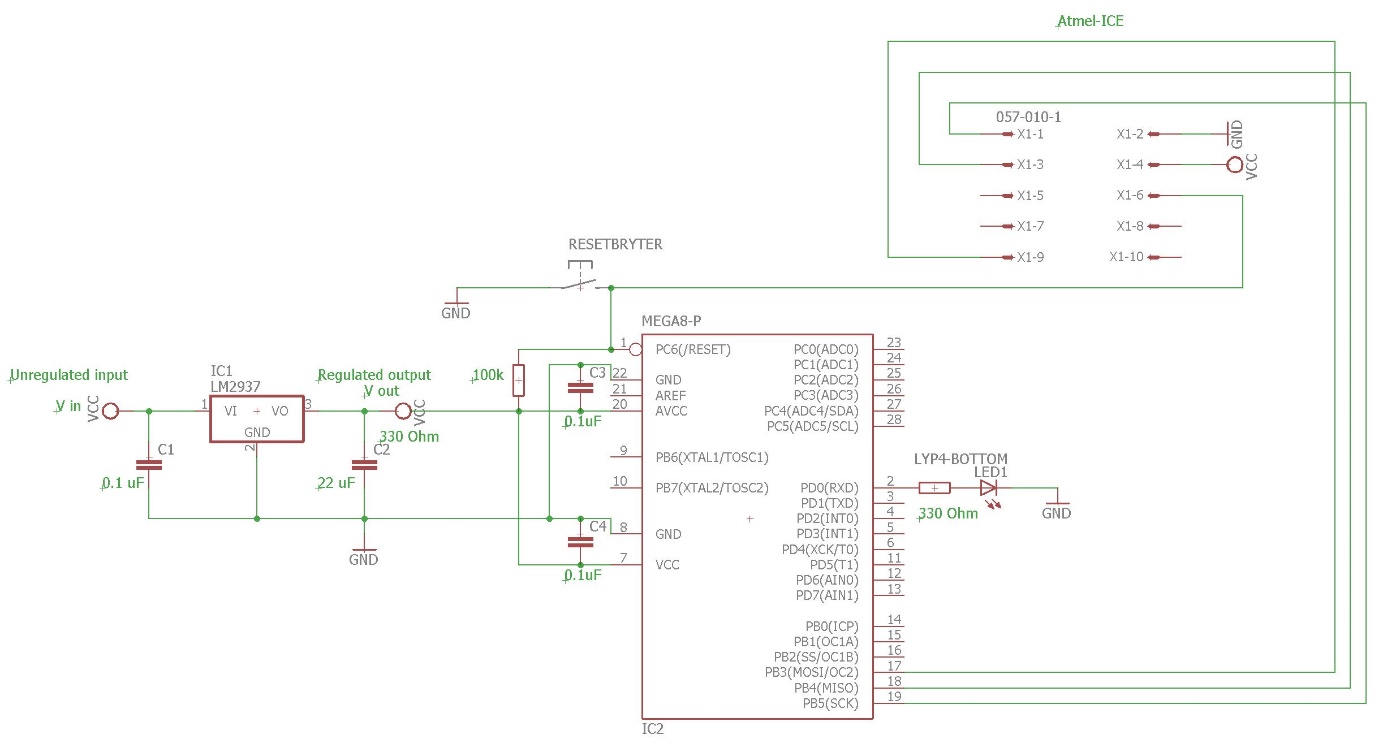
# Styre LED ved bruk av Atmega168

Vi vil programmere en mikrokontroller til å styre en lysdiode. Før man kan lage et program for å styre lysdioden, må den kobles.

## Oppkobling

Vi koblet lysdioden inn på pinne 2, altså PD0, med tilhørende seriemotstand på 330, og videre til jord. Ved å koble den til jord, vil mikrokontrolleren source strøm. Altså vil LEDen kun lyse når mikrokontrolleren gir ut 5V fra PD0, slik at vi får en fullstendig krets til jord (figur 9). Grunnen til at vi valgte å ”source” fremfor å ”sinke”, er at kretsen vi har koblet på utgangen er såpass liten og trekker såpass lite strøm, at mikrokontrolleren fint klarer å levere tilstrekkelig med strøm [6].

Koblingsskjema med LED som skal styres



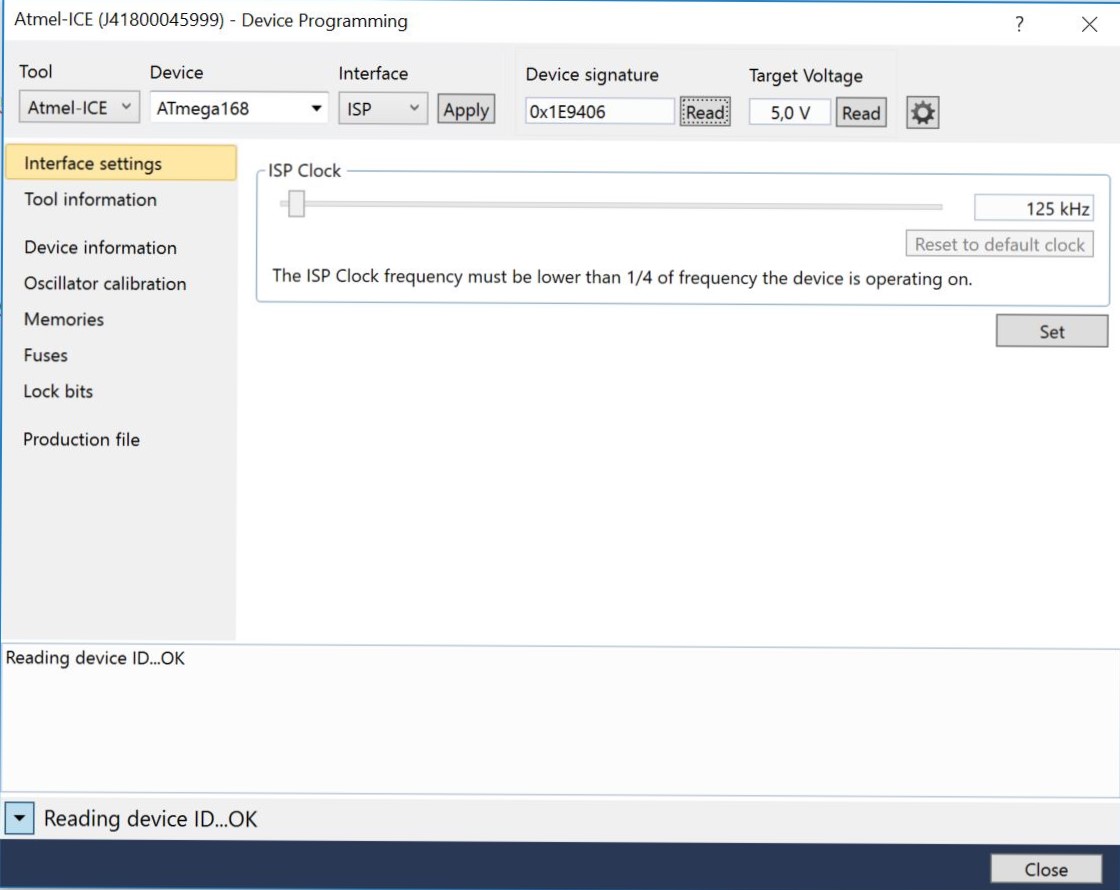
Figur

## AtmelStudio

AtmelStudio er en programvare brukt til å skrive, bygge og debugge kode som er skrevet i C/C++.

Vi startet AtmelStudio for første gang, og gjorde oss kjent med programmet. Vi gikk inn på ”Device Programming” og leste av target volt til å være 5V (figur 10). Vi fant signature bytes til Atmega168 i databladet til å være 0x1E9406, og dette stemte overens med vår avleste signature fra AtmelStudio.

Leser av Target Volt og sjekker Device signature



Figur

### Fuses

Fuses er en form for hovedinnstillinger som lagres og opprettholdes på mikrokontrolleren, selv om man kutter strømmen. Konfigurasjonen for fuses velges i Atmel-ICE programvaren.

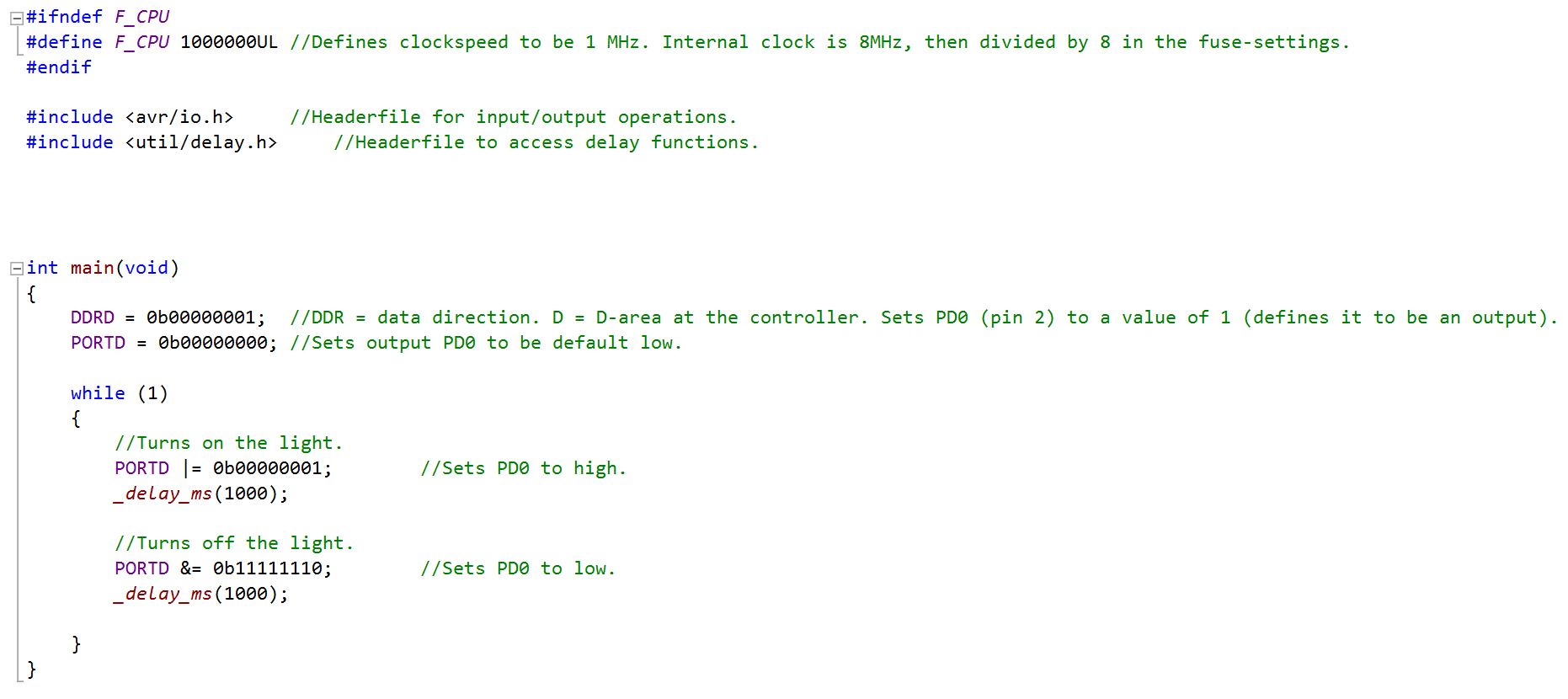
Slik konfigurerte vi fuses:

* Clock selection: Her måtte vi velge riktig ut i fra den interne klokken vi har i kontrolleren vår. Den stod default på ”Internal 8Mhz, 14ck + 65ms”, som også var riktig.
* Clock divider: Denne funksjonen er ”default on” og deler klokkehastigheten vår på 8. Det vil si at vi i praksis får en klokkehastighet på 1 MHz, noe vi måtte ta hensyn til når vi skulle definere klokkesyklusen vår i programmet senere.
* Brown-out detection: Dersom en chip/mikrokontroller får for lav spenning, vil den kjøre ustabilt. Denne funksjonen lar oss sette en grense, slik at mikrokontrolleren vår skrur seg av dersom spenningen skulle gå lavere enn dette. Vi valgte å skru på denne funksjonen, og satt grensen til å være 4,3V [6].

## Programmering i AtmelStudio

Før main-programmet definerte vi klokkesyklusen vår til å være 1 MHz, slik den ble satt i ”fuses”. Vi inkluderte også en headerfil for I/O, samt en headerfil for delay-funksjoner.

Vi definerte PD0 til å være en utgang ved å sette bit 1 i DDRD høy. Inni while-løkken vår programmerte vi PD0 til å gå høy/lav med en syklus på 200ms.



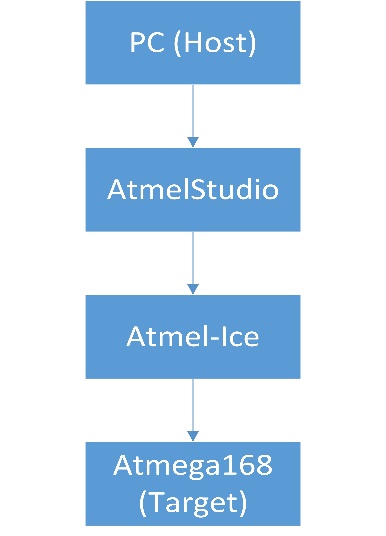
”Memories” er menyen hvor man laster inn programmet sitt til mikrokontrolleren. Her velger man programfilen, som vil være av type .hex, og laster denne over på flashminnet til mikrokontrolleren. Ved å huke av ”Erase Flash before programming” og ”Verify Flash after programming”, vil man slette tidligere programmer før man laster inn det nye, og verifisere at det nye programmet er lastet inn [7].

### Krysskompilering

En kompilator er et program som oversetter(kompilerer) en kildekode(programkode) til en kjørbar maskinkode. I praksis vil man ofte både skrive og kompilere programkoden på et annet system enn det maskinkoden er tiltenkt å kjøre på. Ved å bruke en krysskompilator, kan man kompilere programkoden til en kjørbar maskinkode for andre systemer enn det man jobber på [8].

Når vi skriver programkoden vår i AtmelStudio må vi derfor krysskompilere for at den skal kjøre på en AVR-mikrokontroller.

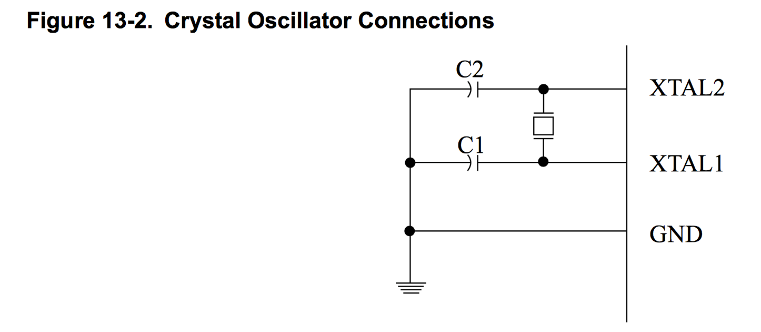
Boksologi av programmeringsoppsettet

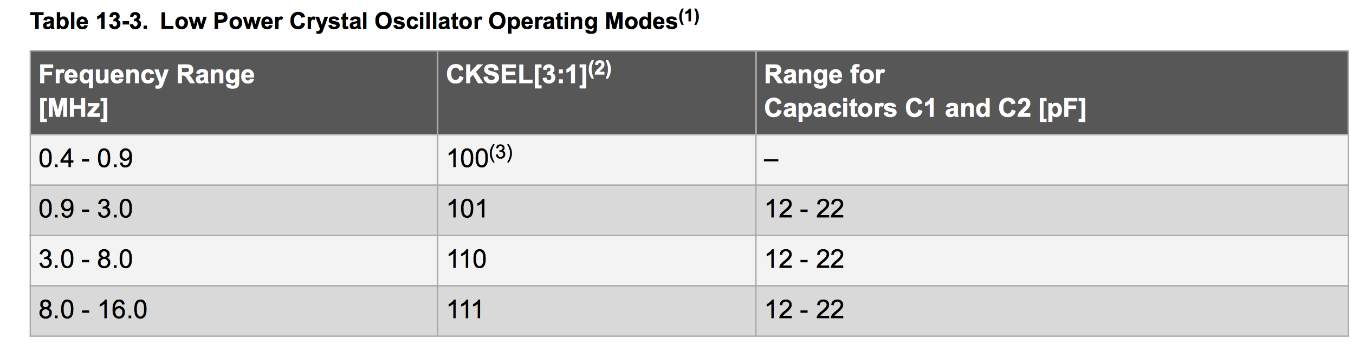
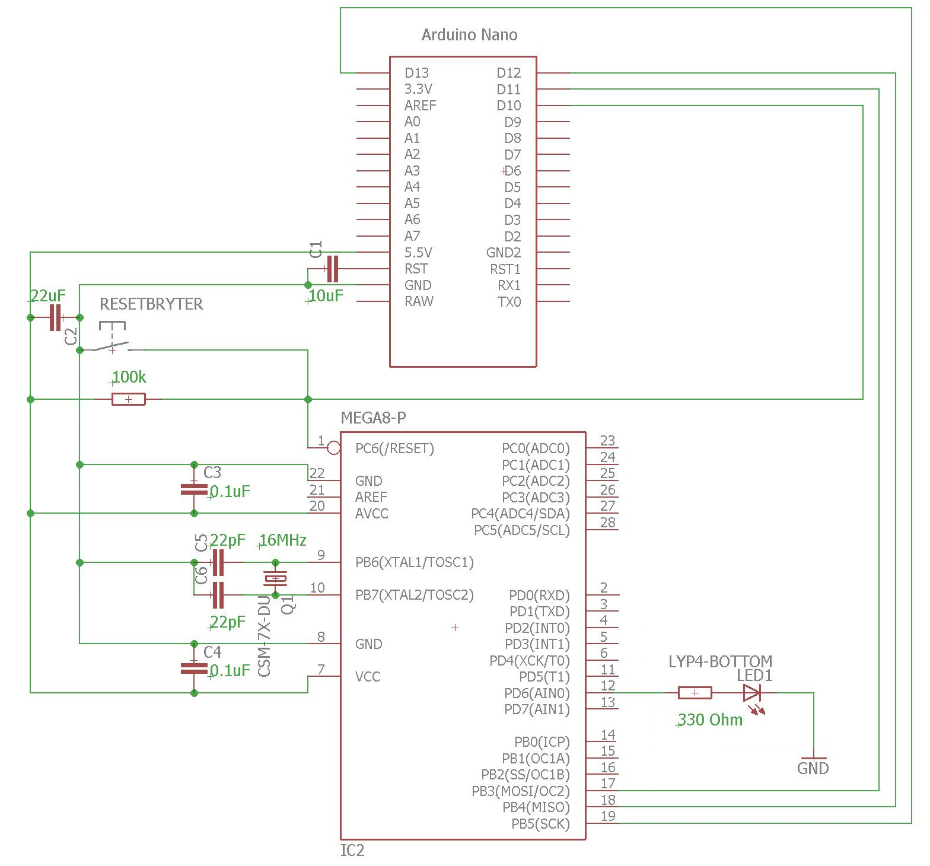


# Overgang til Arduino

På grunn av begrenset tilgang på Atmel-ICE, valgte vi å gå over til å bruke en Arduino Nano for å uploade programmet fra AtmelStudio over til Atmega168. Vi bestemte oss også for å bruke Arduinoen som en spenningskilde for Atmega168 kretsen. Med Arduinoen som spenningskilde var vi fri fra lab strømforsyningen, noe som gjorde det mulig å jobbe med prosjektet hjemme. Arduino Nanoen tilførte en jevn spenning på 5V. Dette gjorde at spenningsregulatoren vi tidligere hadde konstruert ikke lenger var nødvendig.

Vi benyttet et eksempelprogram fra Arduino som gjør den om til en AVRISP. Videre opprettet vi et nytt verktøy i AtmelStudio som vi kalte for ”Upload”. Dette verktøyet ble brukt for å laste opp programmet til mikrokontrolleren vår, og konfigurerer fuses. For å enkelt sette fuses, valgte vi å bruke en fuse kalkulator.

For å få en mer nøyaktig og stabil klokkehastighet, koblet vi opp en ekstern krystall på 16MHz. Vi fant veiledning for oppkoblingen i databladet til Atmega168. Vi måtte da endre innstillingene i fuses til ekstern krystall oscillator, uten prescaler.



# PWM – Pulsbredde modulasjon

PWM står for pulsbredde modulasjon. Normalt ved å endre lysstyrke på en diode, eller hastigheten til en motor vi man et varierende analogt signal. PWM er et alternativ til analog styring. PWM bruker et digitalt signal med en varierende på og av tid. Hvis et digitalt signal er høyt 60% av en tidsperiode og 40% lavt, kan man si at signalet har en dutycycle på 60%. Ved å ha flere slike tidsperioder med en høy frekvens vil signalet oppfattes som konstant, men med en signalstyrke som et direkte resultat av dutycyclen. Dvs. at et høyfrekvent PWM signal med en dutycycle på 60% brukt til å styre en lysdiode, vil oppfattes som en lysstyrke på 60%.

## PWM oppsett

Når man skal bruke PWM, må man velge type PWM, oppløsning, skaleringsfaktorer for klokkehastighet. Disse valgene gjør man ved å sette registre som man kan finne i databladet til mikrokontrolleren.

Før man setter registrene må man velge mellom 8-bit eller 16-bit PWM.

Registre: (ved 8-bit er x = 0) (ved 16-bit er x = 1)

**TCCRxA:** (for å skrive 1 til register: TCCxA = 1 << COMxA1)

* COMxA1 og COMxA0 er register for å velge hva som skjer med OCxA ved visse counter verdier.
* WGMx0, WGMx1, WGMx2 og WGMx3 er register for å velge PWM mode of operation.

**TCCRxB:** (for å skrive 1 til register: TCCxB = 1 << CSx0)

* CSx0, CSx1 og CSx2 er registre for å velge PWM prescaler.

**TIMSKx:** (for å skrive 1 til register: TIMSKx = 1 << TOIE)

* Register for å endre type interrupt. TOIE er f.eks overflow interrupt.

Funksjonen **sei();** må kjøres slik at global interrupt er aktivert.

**OCRxA**-verdi må også settes. Dette gjøres ved å sette OCRxA=(dutycycle/100)\*2antall bit.

Når registre og OCRxA-verdi er satt, må man velge riktig utgangspinne på mikrokontrolleren. Pinnen som skal brukes er pinnen i pinnekonfigurasjonen som er knyttet til OCxA man har valgt.

## PWM modus og frekvens

Vi velger PWM modus til (8-16)-bit fast PWM med clear OCA ved compare match. Denne modusen vil kjøre en teller, som teller fra 0 til 2antall bit. Clear on compare match betyr: utgangen knyttet til OCA (PD6 ved 8 bit) starter høy ved teller-verdi 0, og blir satt lav ved dutycycle\*2antall bit. PWM frekvensen er hvor mange telle-sekvenser PWM styringen vil kjøre pr sekund.

## LED styrt med PWM med konstant dutycycle

I databladet til lysdioden står det at i PWM-mode tåler LED-en en peakstrøm på 140mA ved 1/10 duty cycle, 0,1ms pulse width [9]. Dette er større enn maksimal konstant strøm på 25mA.

Med konstant dutycycle vil lyset fra dioden oppfattes annerledes ved forskjellige frekvenser. Ved lavere frekvenser og dutycycle mindre enn 100% vil det se ut som lyset fra dioden blinker. Dette skyldes at lyset endrer tilstand mellom på og av med en såpass lav frekvens at øyet vil kunne se perioden når dioden ikke lyser.

For å demonstrere hvordan lyset fra dioden blir ved en lavere frekvens, valgte vi 16-bit fast PWM med clear OCA on compare match med prescaler satt til 1024.

Disse parameterne førte til følgende frekvens:

En frekvens på 15,26 Hz resulterte i en blinkende diode når dutycycle var mindre enn 100%.

For å få lys som ikke oppfattes som blinkene må PWM frekvensen økes, slik at øyet ikke lenger klarer å se perioden når dioden ikke lyser. Dette valgte vi å løse med å bruke en 8-bit fast PWM ned clear OCA on compare match med prescaler satt til 256.

Disse parameterne førte til følgende frekvens:

Denne nye frekvensen på 244,14 Hz resulterte i jevnt lys fra dioden ved dutycycle varierende fra 0 – 100 %. Man kunne ikke lenger se perioden når dioden ikke lyste, og øyet oppfattet lyset som konstant dimmet.

## 

## Jevnt blinkende LED – Softblink

For å få en lysdiode til å blinke jevnt med bruk av PWM, må man øke og minke dutycycle gradvis. Dette kan gjøres ved å lage en retningsvariabel som kontrollerer om dutycycle skal øke eller minke, for deretter å justere dutycycle verdien ved bruk av interrupts.

Ved å bruke 8-bit PWM-parameterne fra forrige oppgave, vil vi få en PWM-frekvens på 244,14 Hz. Man kan nå aktivere overflow interrupt for å justere dutycycle. Et overflow interrupt er et avbrudd som vil bli aktivert når tilhørende teller har telt til maks verdi. En 8-bit teller med 244,14 Hz vil aktivere overflow interrupt hvert 4,096 ms. Hvis vi nå bruker interrupt funksjonen(ADC\_vect) til å justere dutycycle med 0,4096 pr interrupt, vil vi få en blinkesyklus på 2 sekunder.

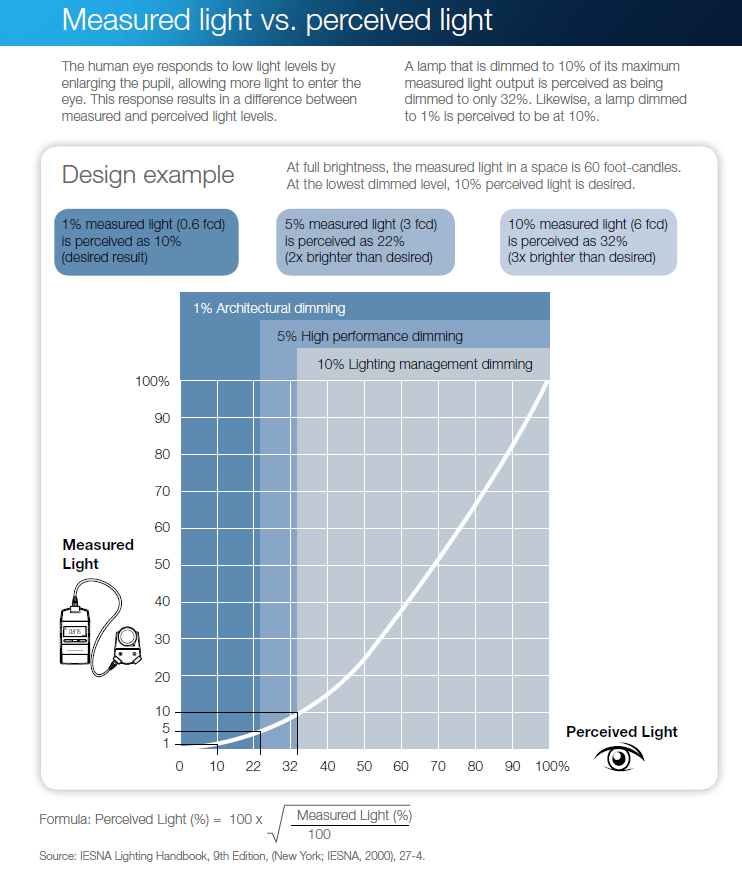
Lysdioden blinket nå med en fast blinkeperiode, men blinkingen så ikke helt jevn ut. Lyset vil nå endre seg helt jevnt matematisk, men menneskeøyne responderer ikke lineært på lys.

### Tilpasning av lysnivå for øyet

Menneskeøyet responderer ikke lineært på lys, fordi pupillen vil utvide seg ved lave lysnivåer slik at øyet tar inn mer lys ved lavere nivåer enn ved høye nivåer.

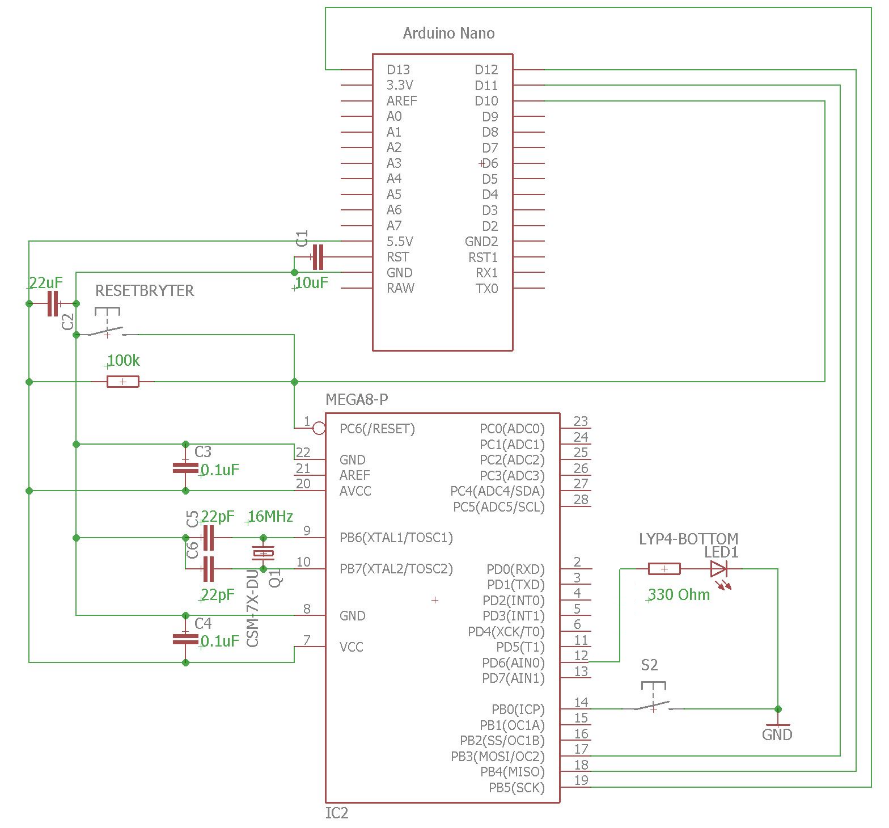
For å løse dette problemet, fant vi en ligning som vi kunne bruke for å regne ut forholdet mellom teoretisk lys og øyets oppfatning av lys.

Ved å lage en ny variabel som regner ut lysnivå tilpasset øyet, og bruke denne variabelen for å sette OCR0A verdien, vil man effektivt tilpasse lysnivået relativt til øyets oppfatning.



# Knapp for styring av lysdiode

I denne oppgaven skal vi bruke en lysdiode for å representere verdien til en knapp. knappen skal kobles fra jord inn på PB0 som vi velger som inngangspinne. Når knappen ikke er trykket og mikrokontrolleren ikke driver inngangspinnen, blir pinnen ladet sakte opp til 5V. Dette fører til en uforutsigbar krets, og kan løses med å sette inngangspinnen til 5V konstant. Dette kan bli gjort ved enten at mikrokontrolleren driver pinnen eller en ekstern pull up spenning. Nå vil inngangspinnens verdi bli satt til 0V når knappen er aktivert og 5V når den er deaktivert. Inngangspinnens verdi kan nå representere knapens tilstand med en lysdiode på utgangspinnen PD6.



# 

# Avbrudd - pin change interrupt

Det finnes mange typer avbrudd/interrupts. Det de har til felles er at de detekterer spesielle hendelser og setter flagg og kjører funksjoner ut i fra hvilken hendelse som inntreffer. Fordelen med bruk av interrupts fremfor å programmere alt i main koden er at interrupts vil bli utført uavhengig av hva som skjer i main koden. Ved å bruke interrupts er main koden fri til annen programmering.

Pin change interrupt er avbrudd forårsaket av tilstandsendring på en inngangspinne på mikrokontrolleren.

For å få aktivert pin change interrupts for en gitt pinne, må Pin Change Interrupt Control Registeret og ett av Pin Change Mask-Registerene stilles inn korrekt:

**PCICR:** (for å skrive 1 til register: PCICR = 1 << PCIE0)

* PCIEx er register for hvilken inngangspinne (x representerer pinne nr.) man skal aktivere pinchange interrupt på.

**PCMSK0:** (for å skrive 1 til register: PCMSK0 = 1 << PCINT0)

* PCINTx er Register for hvilken interrupt mask som skal aktiveres (x representer mask nr.). interrupt masken kjører tilhørende vektor funksjon som inneholder kode for det man vil skal bli utført av interrupten.

Funksjonen **sei();** må kjøres slik at global interrupt er aktivert.

Vi vil bruke pin change interrupt sammen med kretsen fra forrige oppgave til å toggle en lysdiode når bryteren trykkes. Pin change interrupt vil bli aktivert ved både stigende og synkende flanke. Hvis lysdioden toggles ved hver interrupt, vil dioden lyse når vi trykker knappen, for så å slå seg av igjen når vi slipper knappen. For å hindre at lysdioden toggler når vi slipper knappen, må vi programmere slik at lysdioden kun kan toggle hvis knappen er aktivert (PB0 = 0V). Nå vil ikke lysdioden toggle ved synkende flanke på knappen og vi har oppnådd ønsket funksjon.

Når vi testet toggle med pin change interrupt oppdaget vi at lysdioden togglet flere ganger enn ønskelig ved aktivering av knappen. Dette skyldes debounce.

## Debounce

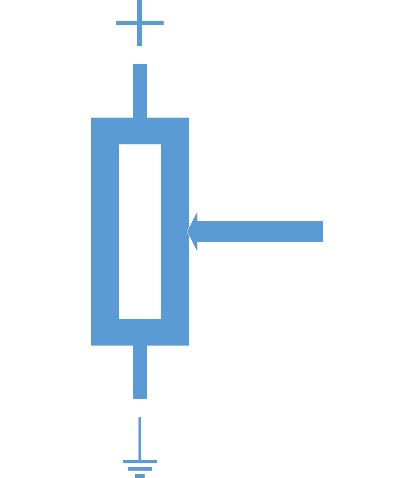
Debounce er ekstra pulser som kan oppstå når en knapp endres mellom åpen og lukket tilstand. Debounce vil føre til ekstra pin change interrupts, dette er ikke ønskelig. Debounce kan fjernes på flere måter. En av de beste måtene vil være å lese knappen med en fast rate, men på grunn av dårlig tid fikk vi bare løst det med delay.

Ved å deaktive globale interrupts i pin change interrupt vektor funksjonen, for så å reaktivere den igjen etter 5ms, vil vi hindre at det kan komme en ny interrupt innen de 5ms. Ved å hindre nye interrupts i denne tidsperioden vil vi hindre interrupts som følge av debounce. Denne tidsperioden er såpass kort at man ikke vil rekke å trykke på knappen før systemet er klart for ny interrupt, og vi vil da registrere interrupt fra neste knappetrykk [10].

Dette er ikke en optimal løsning, fordi delay vil skape ventetid i systemet som kunne blitt brukt til noe annet. Funksjonsbeskrivelsen oppfylles, da denne løsningen toggler lysdioden uten debounce, og vil fungere bra til dette bruket.

# ADC – analog til digital konvertering

ADC (analog til digital konvertering) lar en konvertere analoge signaler til digitale signaler, som kan brukes av mikrokontrolleren. Dette er nyttig ved bruk av analoge sensorer, slik som måling av temperatur, avstand, kraft, eller som i vårt tilfelle, hvor vi ønsket å styre lysstyrken på en diode ved å justere et potensiometer.

Et potensiometer er en justerbar motstand som lar deg regulere spenning. Det fungerer med prinsippet til Ohms lov, ved at høyere motstand gir større spenningsfall mellom 5V og signalpinnen.

Til vårt prosjekt har vi brukt et potensiometer med range på 10kΩ som vi har justert manuelt.

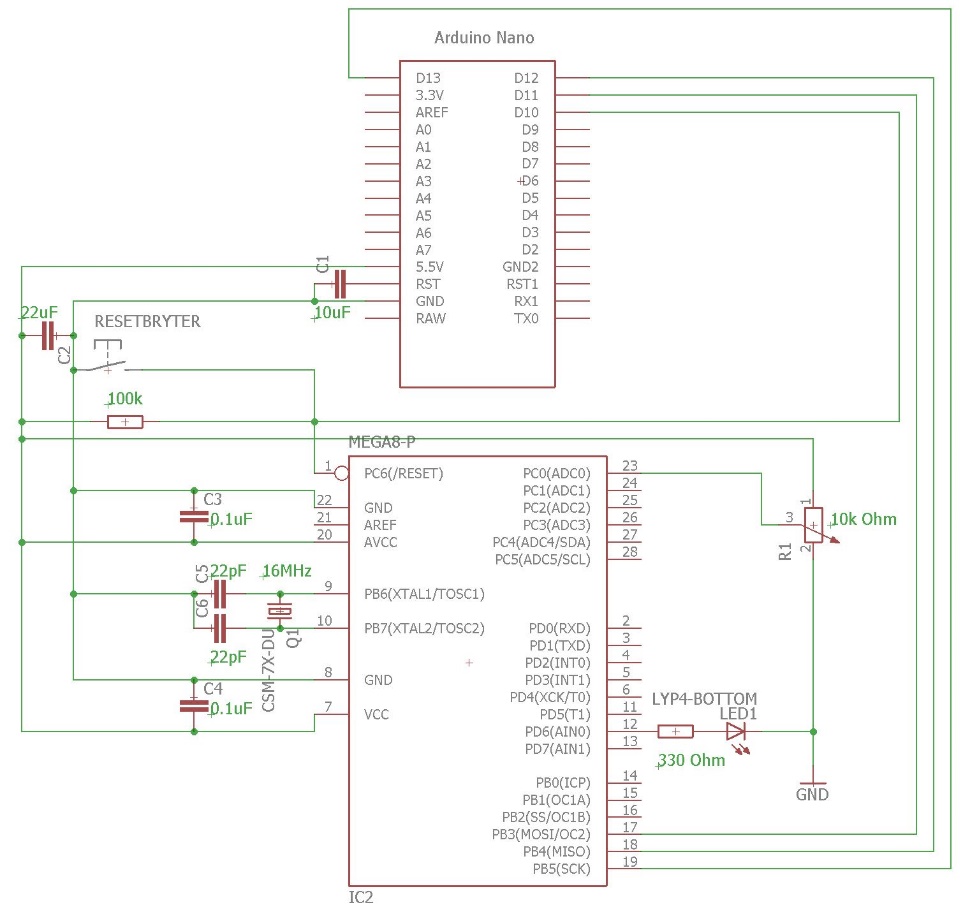
AVcc er en separat strømforsyning til ADC. Denne kan ikke variere mer enn ±0,3V fra Vcc. Vi koblet inn samme 5V strømforsyning som på Vcc, med en kondensator mellom AVcc og jord for å redusere påvirkningen av støy.

Når man benytter et analogt spenningssignal, trenger man også en referansespenning å sammenligne mot. Med en Atmega168 har man tre valg:

1. AVcc
2. Intern 1,1V
3. Ekstern spenning på Aref

Aref skal kobles til en ekstern spenning og bli brukt som referansespenning til ADC dersom det analoge signalet ikke har samme strømforsyning som resten av kretsen.

I vårt prosjekt brukte vi AVcc som referansespenning, da potensiometeret var tilkoblet samme 5V strømforsyning som resten av kretsen. Når man benytter AVcc eller den interne spenningen fra mikrokontrolleren, er det viktig å ikke ha noe tilkoblet på Aref. Aref er direkte tilkoblet ADC og en eventuell spenning her vil kortslutte de andre referansespenningene.



## Konfigurasjon av ADC

Når man skal bruke analog til digital konvertering, må man aktivere en del funksjoner ved å sette registre. Informasjon om dette fant vi i databladet til mikrokontrolleren.

Registre:

**ADMUX:**

* REFS0 og REFS1 er registre for å velge hvilken referansespenning man ønsker.
* MUX0-MUX3 brukes i kombinasjon for å velge hvilken analog inngang som skal benyttes.

**ADCSRA:**

* ADEN aktiverer ADC.
* ADSC starter konverteringen.
* ADATE aktiverer en auto-trigger funksjon for konverteringen.
* ADIE gir en interrupt når konverteringen er ferdig.
* ADPS0-ADPS2 er registre for å bestemme en pre-scaler faktor.

**ADCL og ADCH:**

* AD-konverteren returnerer en verdi på 10 bits som blir lagret i disse registrene. For å hente opp denne verdien i programmet, bruker man ”ADC”.

# Konklusjon

Del 2 Arduino og I2C

Del 3 Arduino/Teensy 3.6 og CAN-bus

# 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 [11]. Vi skal også bruke biblioteket FlexCAN for å få tilgang til CAN funksjonene [12].

Dual CAN-bus adapteren inneholder to transivere, en for Can0 og en for Can1. Adapteren inneholder også en 120Ω terminerings motstand [13].

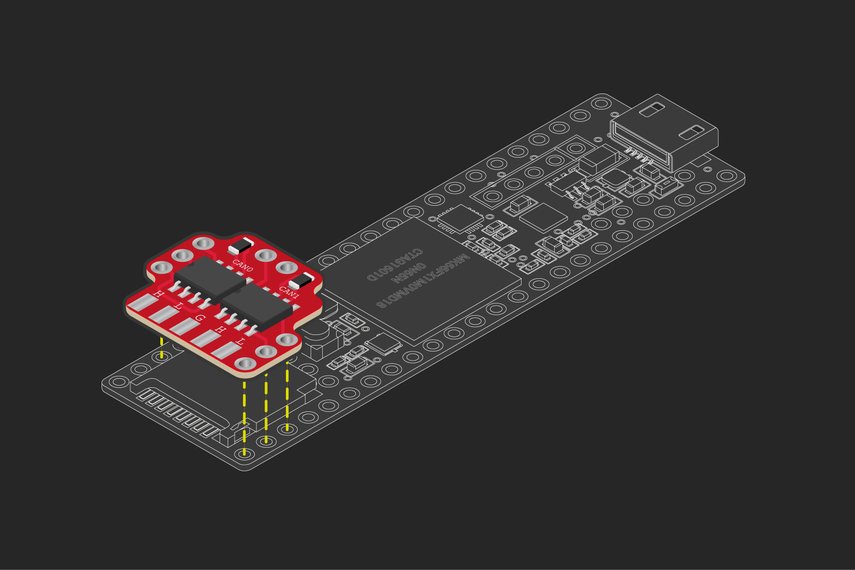
# Metode

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 [14].

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 [15]. Vi bruker brukermanualen til PCAN-USB, CAN-grensesnitt for USB, som veiledning for å sette opp PcanView [16].

## 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.

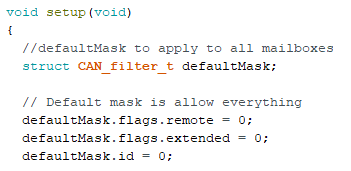
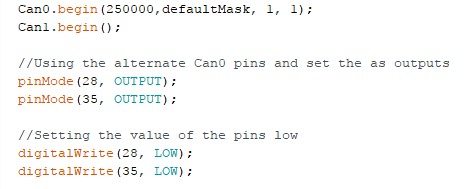
### Void

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 [17].

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

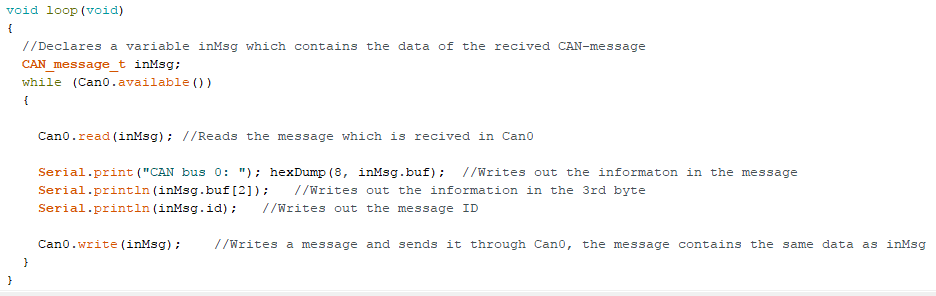
Setup - kode



### 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 [12]. 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.

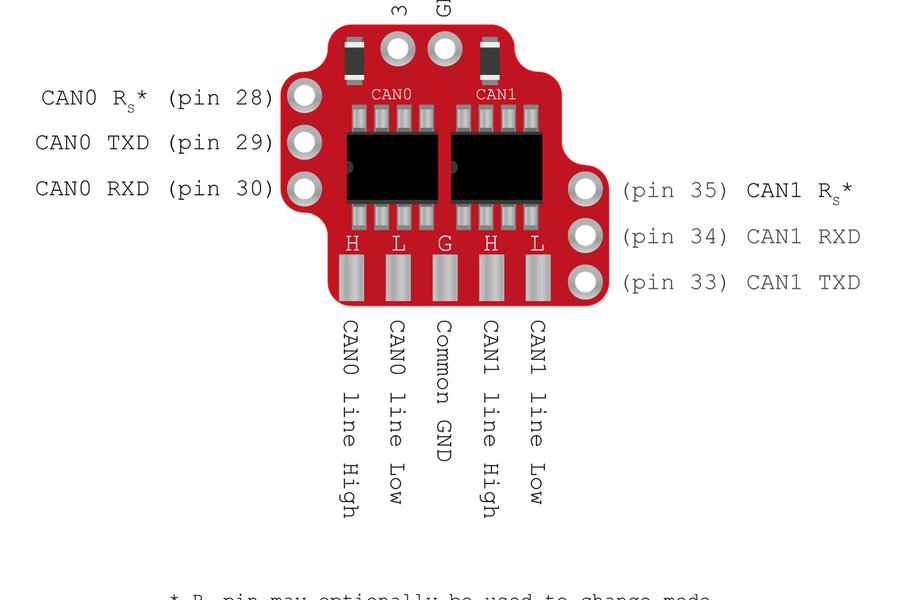
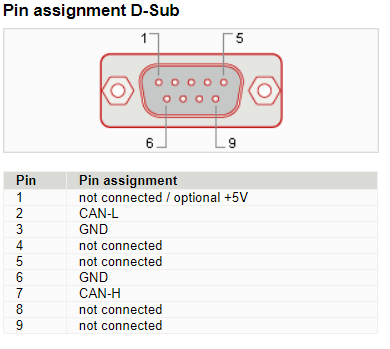
Loop - kode



## 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



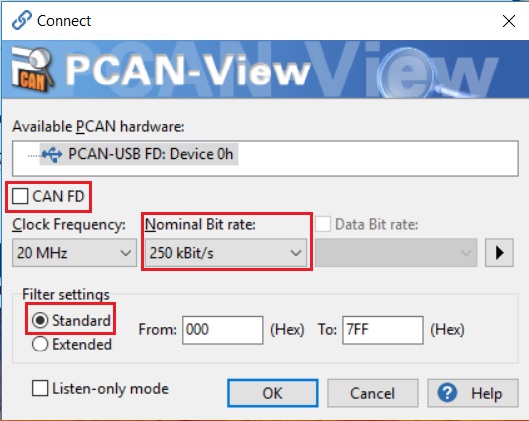
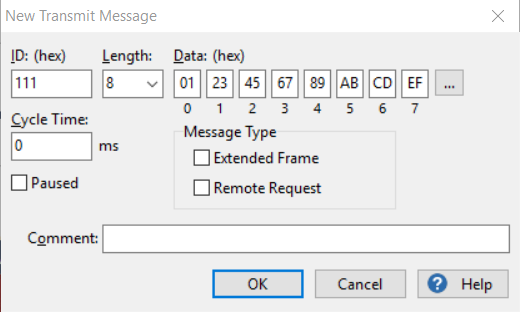
PCAN koblingsskjema

### 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

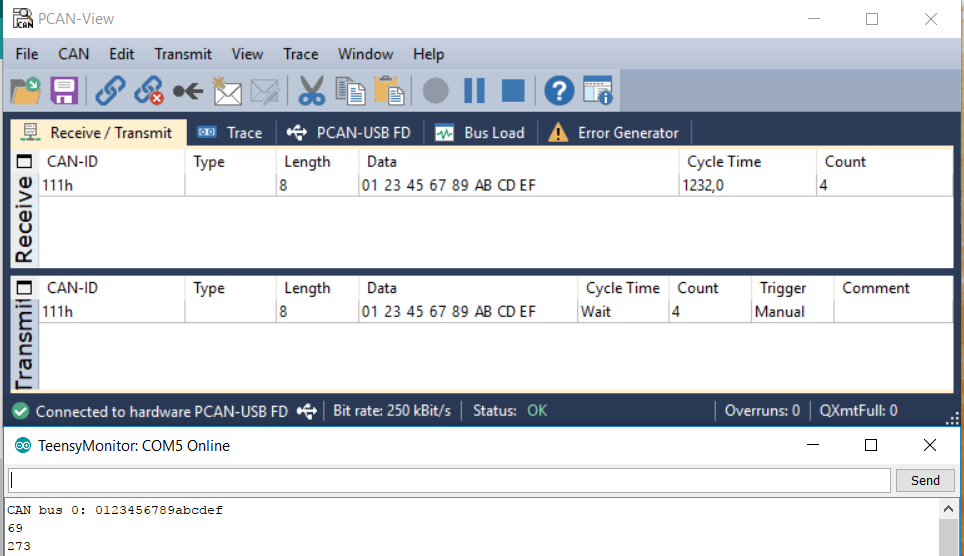
PcanView – Ny melding



### 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.

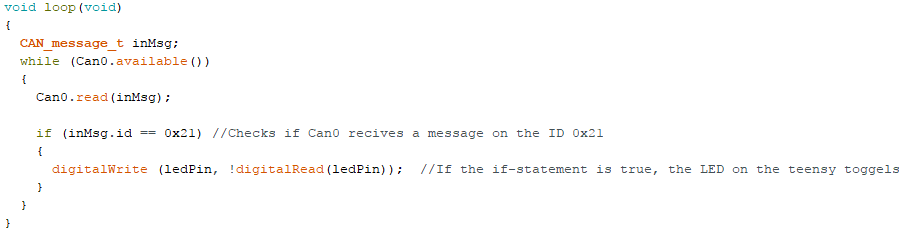
PcanView recive and transmitt



## 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

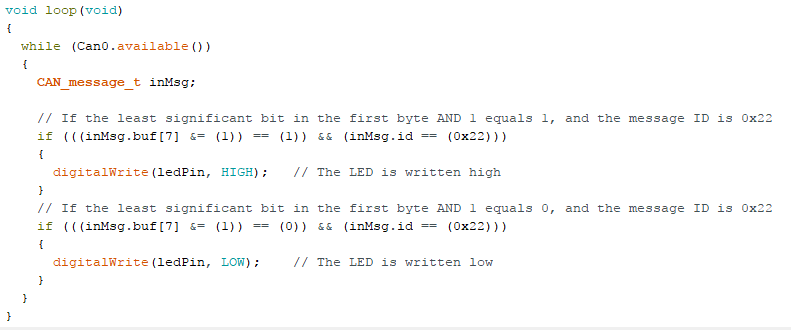


## Styre LED med innhold i melding

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**

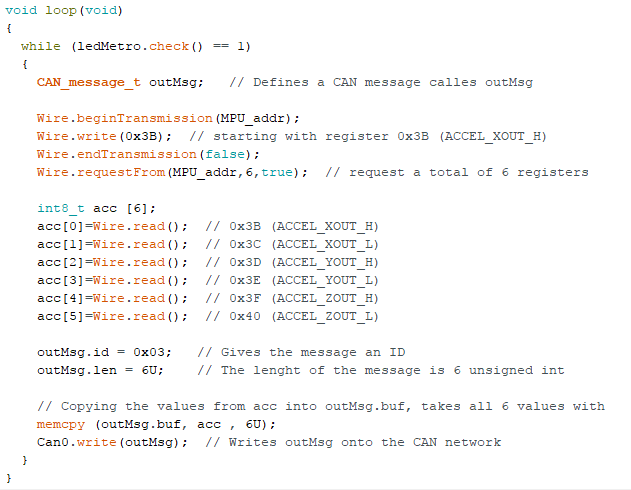
Styre LED med innhold i melding kode



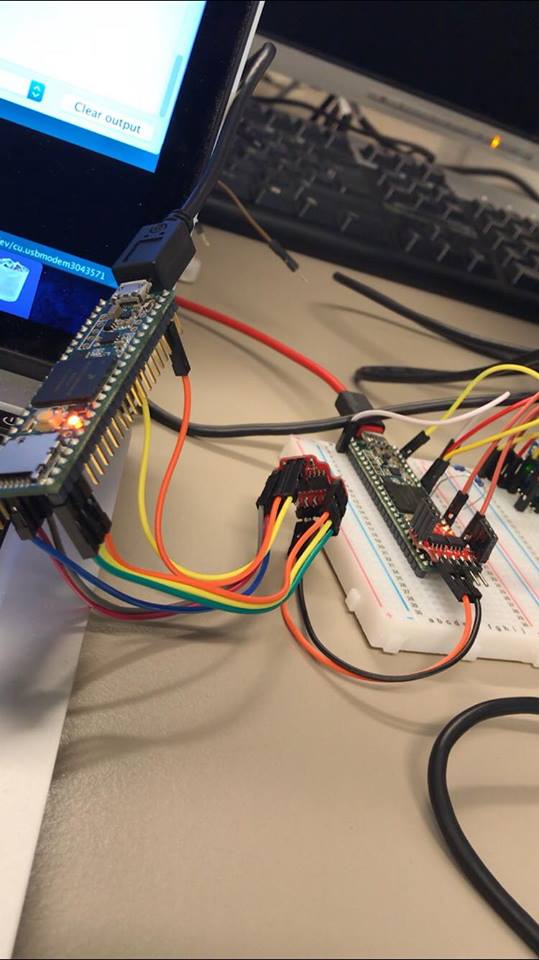
## Rapportere verdier fra IMU via CAN-bus

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 [18]. 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

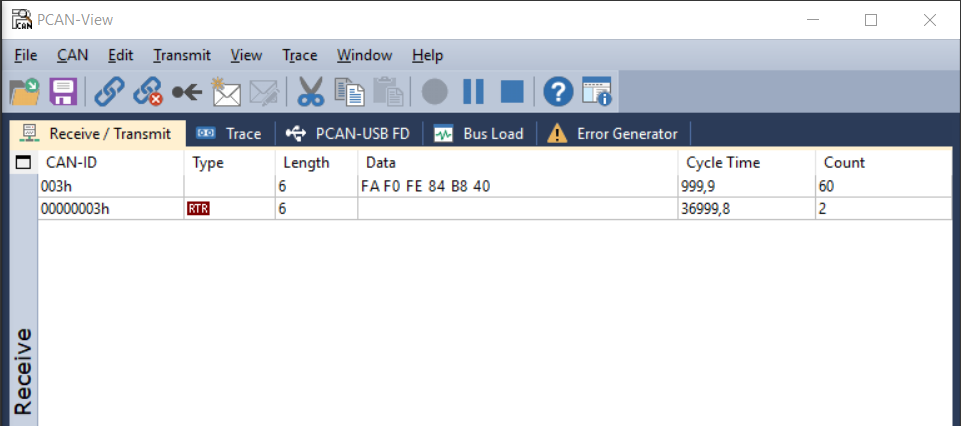
Det første vi må velge er bit rate, bit raten bestemmer hvor mye og hvor fort data sendes i et CAN-bus nettverk. Bit raten kan variere fra 1Mbit/s til 5kbit/s. For å ha en raskere bit rate må man ofre lengden på bussen. Vi velger en bitrate på 250kbit/s, fordi det er standard hastigheten i FlexCAN.

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 [19]. 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.

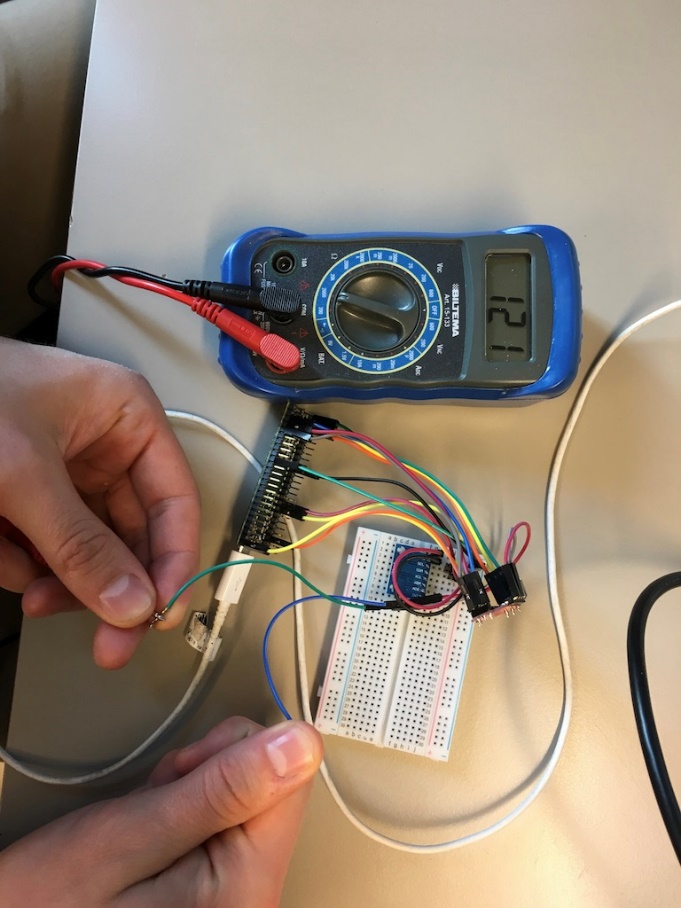
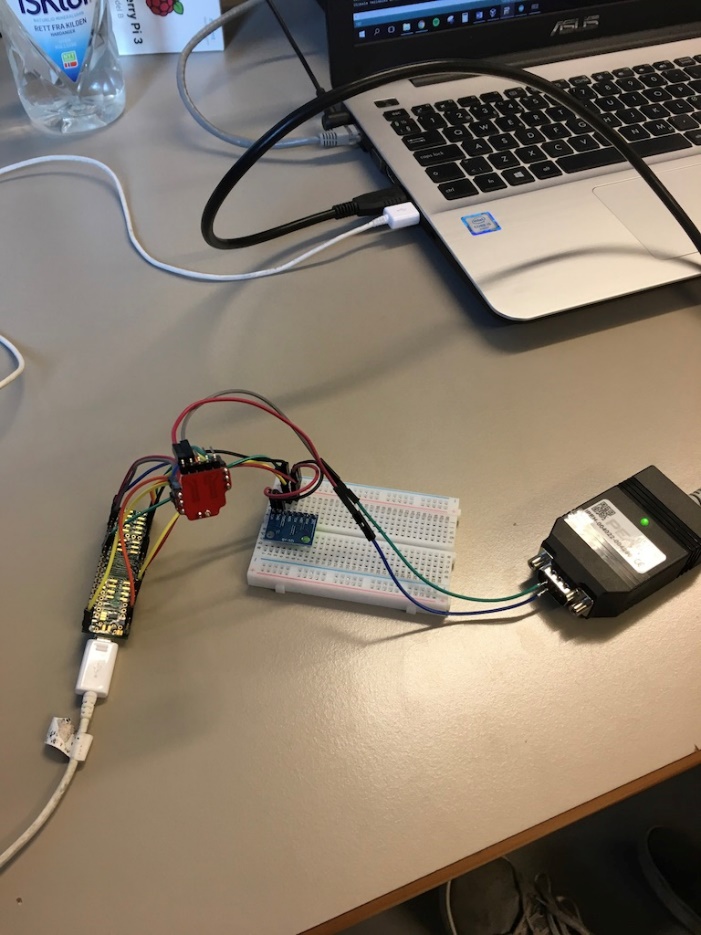
Retry meldingene vi fikk i PcanView



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.

# Konklusjon

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.

Del 4 Raspberry Pi 3/Buildroot og CAN

# Introduksjon

I denne delen av prosjektet skal vi benytte verktøyet Buildroot til å konfigurere en Linuxdistribusjon for Raspberry Pi 3. Dette skal gjøres i en virtuell maskin med Ubuntu som operativsystem. Linuxdistribusjonen skal skrives på et microSD-kort, og brukes til å starte opp en Raspberry Pi 3. Et tilleggskort, PiCAN, med CAN-bus, GPS, gyro og akselerometer skal monteres på Raspberryen. Raspberryen i kombinasjon med PiCAN skal kobles til PCAN-adapteren, og kommunisere over CAN-bus. Deretter skal Raspberryen og Teensyen kobles i et CAN-nettverk. Raspberryen skal forespørre informasjon fra Teensyen og den tilkoblede IMUen, og denne informasjonen skal vises i terminalen til Raspberryen.

# Metode

## Ubuntu og VirtualBox

Ubuntu er et ”open source” operativsystem for datamaskiner, publisert av Canonical Ltd. Operativsystemet bygger på Linuxkjernen, og er per dags dato en av de mest populære Linux-distribusjonene. Det er gratis å bruke, som et resultat av at frivillige utviklere over hele verden som bidrar til å utvikle systemet.

VirtualBox er en gratis programvare utviklet av Oracle Corporation, for å simulere virtuelle maskiner. Windows og diverse Linux-distribusjoner er blant operativsystemene som kan simuleres på disse maskinene [20].

I dette prosjektet har vi lastet ned og installert VirtualBox, for så å installere versjon 16.04.3 LTS av Ubuntu på en virtuell maskin. Vi installerte også VirtualBox Extension Pack for at den virtuelle maskinen skulle støtte enheter med USB 2.0 og USB 3.0 [21].

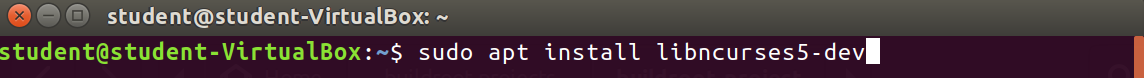
## Buildroot

Buildroot er en gratis ”open source”-programvare for å enkelt bygge og konfigurere en komplett Linux-distribusjon for innebygde datasystemer [22].

Som en del av prosjektet skulle vi konfigurere en egen Linux-distribusjon som skulle fungere med en Raspberry Pi 3 og et Skpang CAN-kort.

Vi lastet ned versjon 2017.02.07 av Buildroot på vår virtuelle Ubuntu-maskin. Via terminalvinduet fra buildrootprosjektet, prøvde vi å åpne konfigurasjonsmenyen med kommandoen ”make menuconfig”. Vi fikk beskjed om at ncurses-biblioteket manglet og måtte installeres. Dette installerte vi fra pakkebrønnen:

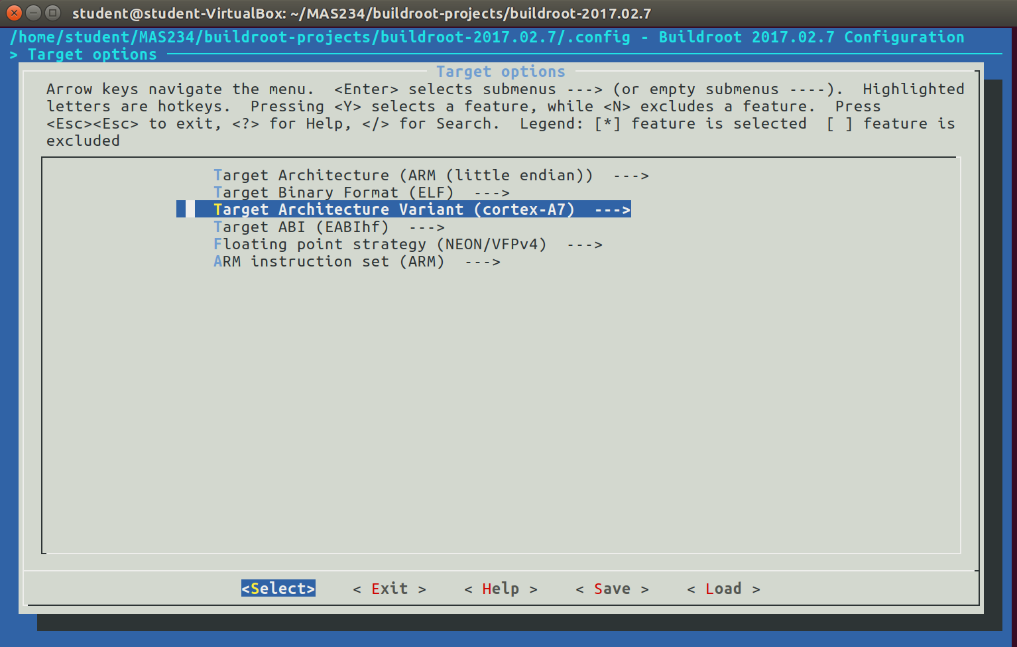
Installerer ncurses-biblioteket



Når ncurses-biblioteket var installert, kontrollerte vi at vi fikk tilgang til konfigurasjonsmenyen til buildroot. Videre kunne vi kjøre kommandoen ”make list-defconfigs” i terminalen. Dette gav oss en liste med alle eksisterende konfigurasjoner for diverse maskinvare. Siden vi bruker en Raspberry Pi 3 i dette prosjektet, kjørte vi kommandoen for å konfigurere denne.

Etter dette åpnet vi konfigurasjonsmenyen igjen, og kontrollerte hvilke ”target options” vi hadde. Her kunne vi blant annet lese at prosessoren vår var en Cortex-A7 basert på ARM-arkitektur [23].

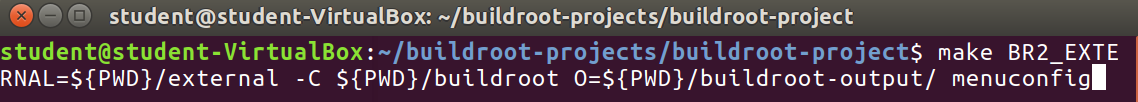
Ser at vi har en Cortex-A7 prosessor



## Installasjon av delvis ferdig konfigurert oppsett

Faglærer la ut en kopi av et delvis ferdig konfigurert oppsett til Raspberry Pi 3. Denne lastet vi ned fra Fronter og fulgte videre veiledning i oppgaven for installasjon. Vi åpnet buildroot konfigurasjonsmenyen fra prosjektmappa for å sjekke oppsettet.

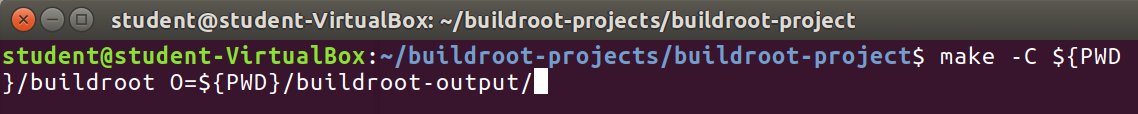
Åpner konfigurasjonsmenyen



For å installere can\_pingpong-pakken, kopierte vi mas234-konfigurasjonsfilen inn i buildroot-output-mappa. Vi navnga filen til ”.config”.

Vi kontrollerte nå at can\_pingpong var valgt i konfigurasjonsmenyen, og startet kompileringen med kommandoen:

Starter kompilering



## SD-kort

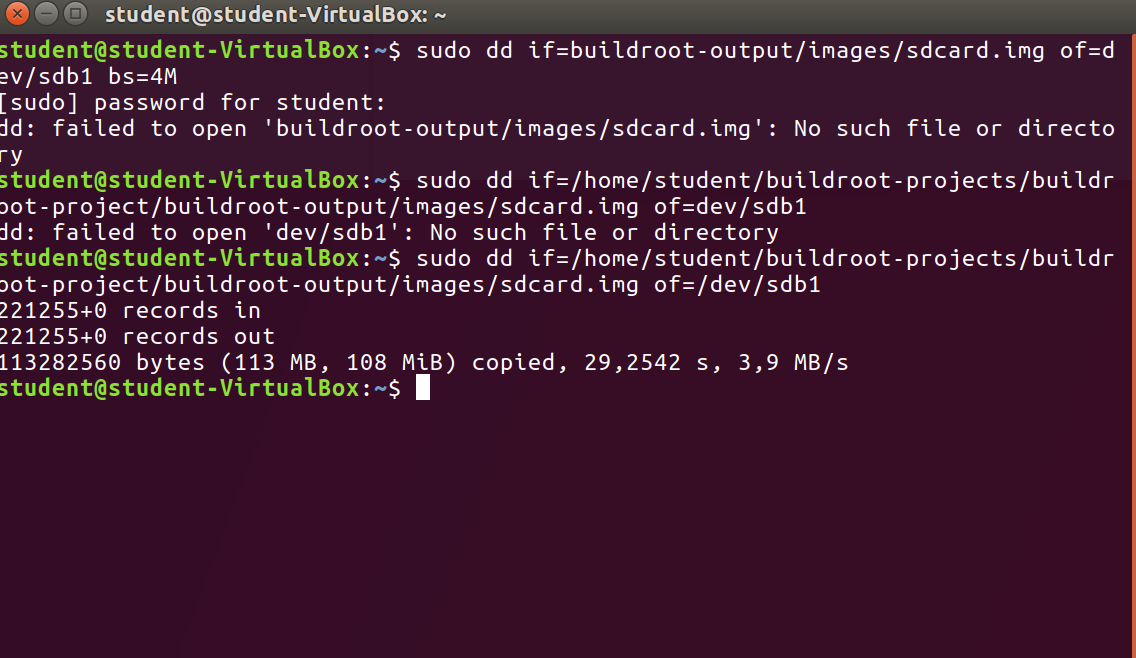
Kompileringen genererte et ferdig SD-kort image. Dette skulle vi skrive til et microSD-kort ved å benytte linux-programmet ”dd”.

dd står for ”data duplicator”, og er et verktøy med primærfunksjon å konvertere og kopiere filer. Dette gjøres direkte i terminalen med kommando-linjer.

Vi koblet til microSD-kortet med en USB-adapter, da den interne SD-kort leseren til datamaskinen ikke fungerte med Ubuntu. For å sikre oss mot at vi skrev SD-kort imaget til vår primærharddisk, kjørte vi kommandoen ”fdisk –l” i terminalen. Denne kommandoen listet opp tilgjengelige disker før og etter vi koblet til SD-kortet. Vi så da at minnekortet var navngitt /dev/sdb.

Ved å bruke ”dd”, skrev vi image-filen vår til SD-kortet. I figur ”????” kan det ses at vi skrev filen til /dev/sdb1. Dette er en partisjon på SD-kortet, og forårsaket feil da vi startet opp Raspberry Pien. Vi endret dette og skrev image-filen til hele SD-kortet da vi oppdaget feilen.

Skriver image-filen til SD-kortet



## Koble opp hardware

Raspberry Pi er en serie av små datamaskiner bygget på et enkelt kretskort. De er utviklet av Raspberry Pi Foundation, hovedsakelig for å promotere undervisning i grunnleggende databehandling på skoler. Produktet har blitt svært populært, og er nå også brukt innenfor robot- og mekatronikk-teknologi. Raspberry Pi 3 modell B bruker en 1.2 GHz 64-bit quadcore prosessor, har 1 GB ram og blant annet USB- og ethernettilkoblinger, trådløst nettverkskort og bluetooth [24].

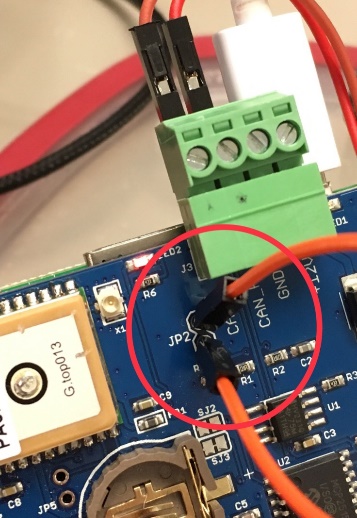
PiCAN er et tilleggskort til Raspberryen som gir oss GPS, Gyro, Akselerometer og CAN-bus kommunikasjon [25].

I dette prosjektet har vi brukt en Raspberry Pi 3 modell B, samt et PiCAN-tilleggskort koblet over det 40 pins store koblingspunktet til Raspberryen.

I dokumentasjonen til Raspberry Pien fant vi spesifikasjoner på hvilken strømforsyning som kreves. Raspberryen må bli forsynt med 5.1V over microUSB [26]. Nøyaktig hvor mye strøm den trekker, varierer ut i fra hva man kobler til av utstyr. Typisk bruker modell B mellom 700-1000mA i seg selv. Fordi vi ikke hadde noe vanlig strømforsyning tilgjengelig, brukte vi en USB port fra en Macbook Pro. Apple oppgir alle USB3.0 til å kunne forsyne utstyr med opptil 900mA ved 5V [27].

Når Raspberry Pien var koblet opp med 5V strømforsyning og gav indikasjon på suksessfull oppstart med microSD-kortet, skulle vi koble til PiCAN-kortet. Vi koblet fra strømforsyningen, og monterte kortet forsiktig på det 40 pins svære koblingspunktet på Raspberryen.

Terminering av CAN-bus

Vi koblet oss på CAN\_H og CAN\_L på PiCAN-kortet gjennom PCAN-USB adapteren. For å kommunisere over CAN-bus, er man avhengig av å terminere nettverket med en motstand. I brukermanualen til PiCAN-kortet står det beskrevet hvordan man kan benytte en intern 120 Ohms motstand til termineringen, ved å koble en ledning mellom punktene på JP2 (figur x).

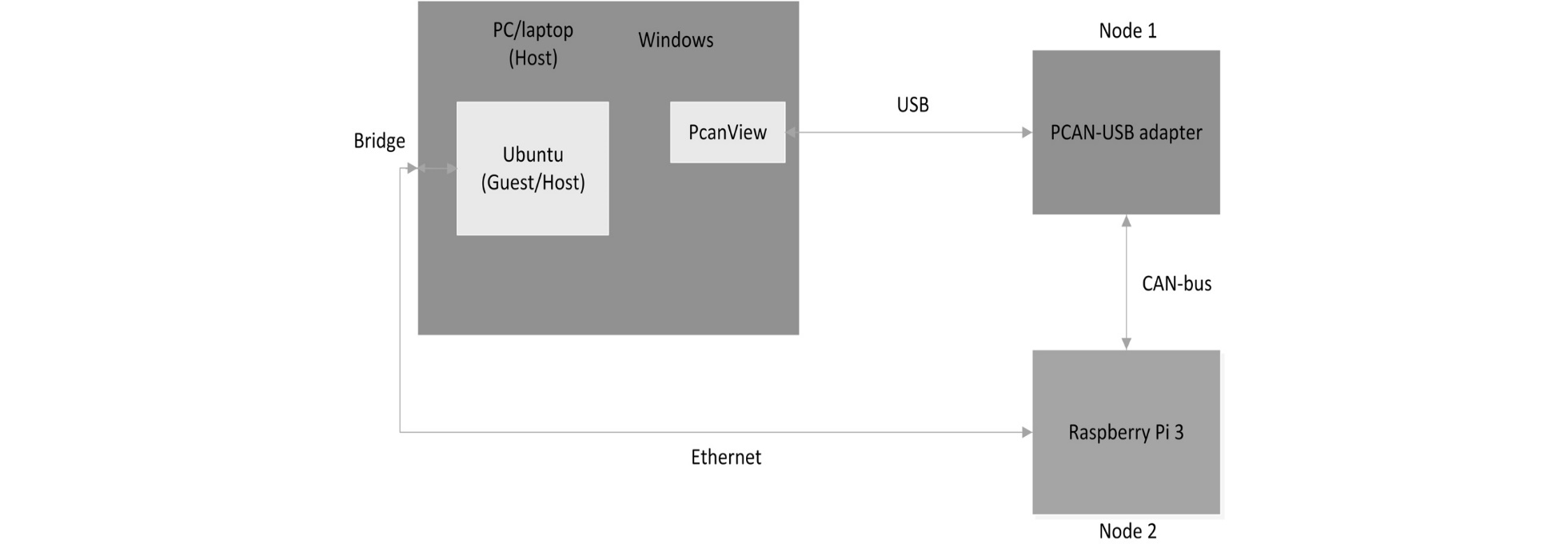
## Konfigurasjon av ethernettilkobling

Vi skulle kommunisere gjennom Raspberry Pi med to metoder:

1. En virtuell Ubuntu-maskin over ethernetkabel.
2. PcanView i Windows over et CAN-nettverk, via en USB-adapter.

Figur ???? viser hvordan vi koblet opp systemet vårt i denne delen av prosjektet

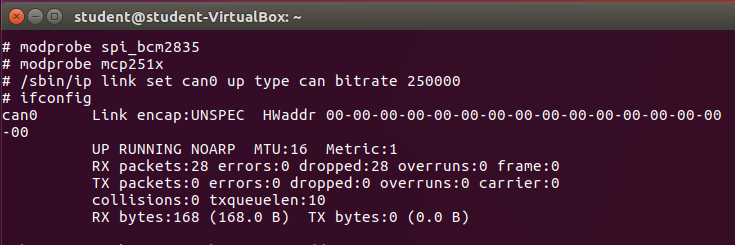
Oppkobling av systemet med: PC, Ubuntu, Bridge, Raspberry Pi 3, PCAN-USB adapter og PcanView



For å kommunisere med Raspberryen over ethernet, måtte vi gjøre noen endringer i nettverksinnstillingene til VirtualBox. Vi lagde en ny ”bridged adapter” og noterte oss MAC-adressen. Under nettverksforbindelsene i Ubuntu, fant vi riktig ethernettilkobling ved å sammenligne adressen mot MAC-adressen til adapteren. IP-adressen til maskinen vår satt vi til å være 192.168.234.1 under IPv4. Vi vet fra før at IP-adressen til Raspberryen er satt til 192.168.234.234.

SSH er en protokoll for sikker tilgang til en datamaskin fra en annen. Dette lar en blant annet overføre filer og kjøre kommandolinjer over internett [28]. Med kommandoen ”ssh [root@192.168.234.234](mailto:root@192.168.234.234)”, fikk vi tilgang til Raspberry Pi på root-nivå. I Linuxdistribusjoner vil det si at vi har tilgang til hele operativsystemet og kan gjøre alle slags endringer [29]. Dette kalles også for ”superbruker”.

## Kommunikasjon over CAN-bus

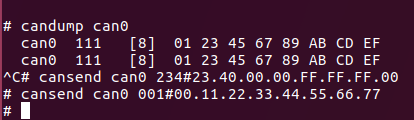
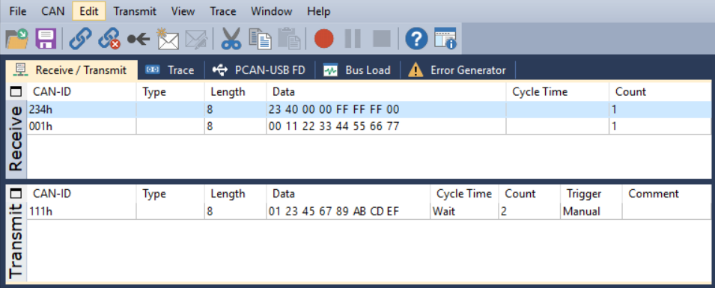
Ved å kjøre kommandoer for å laste inn driverne for SPI og CAN-transceiveren på PiCAN- kortet vårt, samt en kommando for å konfigurere CAN-grensesnittet, gjorde vi det mulig å kommunisere over CAN-bus. For å kontrollere at dette ble utført riktig, kjørte vi ”ifconfig” i terminalen for å se hvilke nettverksgrensesnitt som eksisterte. Can0 kom opp som aktiv. (Se figur)

Sjekker at Can0 er konfigurert

Kommandolinjen ”candump can0” printer alle CAN-meldinger som sendes på nettverket. For å teste kommunikasjonen mellom PCAN-adapteren og Raspberryen, forsøkte vi å sende meldinger fra PcanView til terminalen i Raspberry. Figur (???) og figur (???) viser to meldinger med ID 0x111 og likt innhold.

Mottar og sender meldinger fra terminalen i Raspberry

Sender og mottar meldinger i PcanView

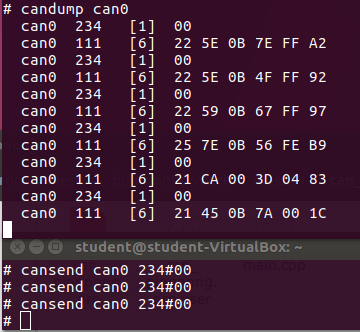
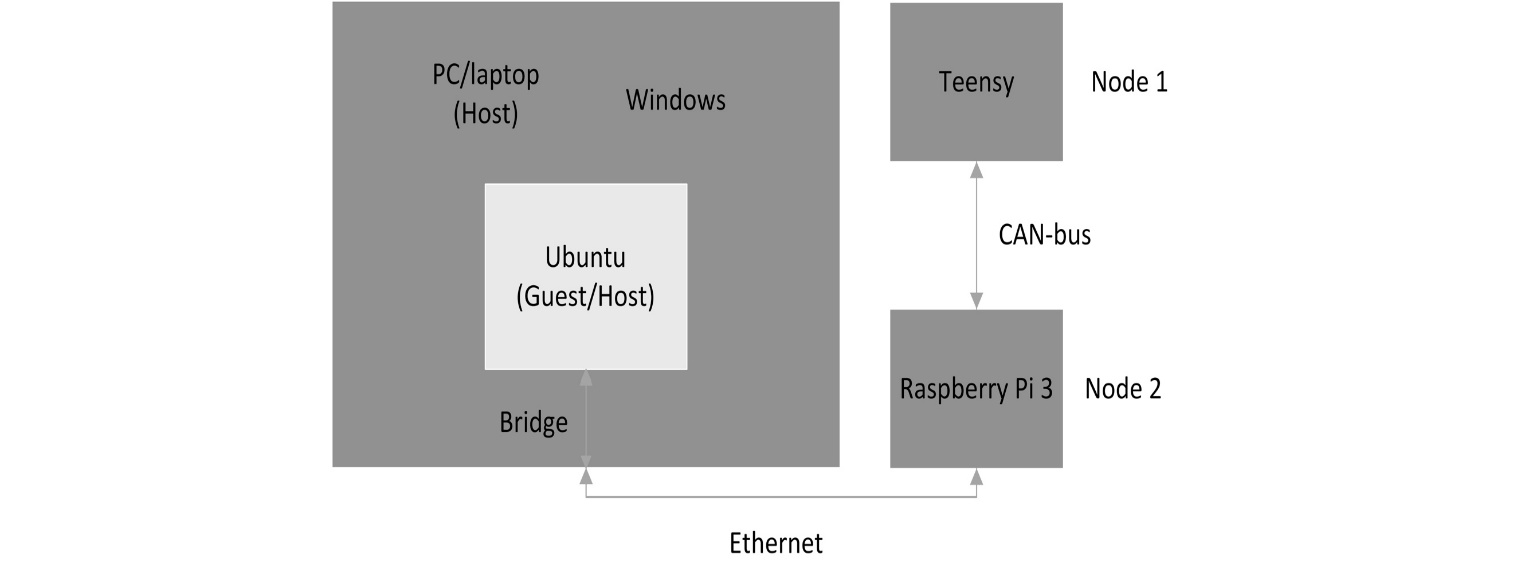


For å ta i bruk CAN-nettverket til et praktisk eksempel, ønsket vi å bruke Raspberryen til å forespørre informasjon fra Teensyen og den tilkoblede IMUen. Når vi sendte ut en CAN-bus melding med ID 0x234 fra terminalen til Raspberryen, mottok og behandlet Teensyen denne. Teensyen vil så lese av gyrodataen fra IMUen og sende denne dataen i en CAN-melding med ID 0x111.

Figur ??? viser mottatt CAN-melding med ID 0x111 og 6 bytes, hvorav de to første representerer gyroverdi for X-retning, de to neste for Y-retning, og de to siste for Z-retning.

Sender melding med ID 0x234 og får tilbake data fra gyroen på ID 0x111

Oppkobling



# Diskusjon

# Resultat

# Kilder

|  |  |
| --- | --- |
| [1] | Jameco electronics, «www.Jameco.com,» Juni 2005. [Internett]. Available: https://www.jameco.com/Jameco/Products/ProdDS/836895.pdf. [Funnet 9 Oktober 2017]. |
| [2] | Atmel Corporation, «www.atmel.com,» Atmel Corporation, November 2016. [Internett]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2545-8-bit-AVR-Microcontroller-ATmega48-88-168\_Datasheet.pdf. [Funnet 6 Oktober 2017]. |
| [3] | Atmel Corporation, «www.atmel.com,» Atmel Corporation, Oktober 2016. [Internett]. Available: http://www.atmel.com/Images/Atmel-42330-Atmel-ICE\_UserGuide.pdf. [Funnet 7 Oktober 2017]. |
| [4] | «www.atmel.com,» Microchip Technology Inc, August 2016. [Internett]. Available: http://www.atmel.com/Images/Atmel-2521-AVR-Hardware-Design-Considerations\_ApplicationNote\_AVR042.pdf. [Funnet 14 November 2017]. |
| [5] | «www.wikipedia.org,» 12 Februar 2017. [Internett]. Available: https://en.wikipedia.org/wiki/Current\_sources\_and\_sinks. [Funnet 16 Oktober 2017]. |
| [6] | M. Hämmerling, «www.engbedded.com,» 2014. [Internett]. Available: http://www.engbedded.com/fusecalc/. [Funnet 1 November 2017]. |
| [7] | Atmel, «www.atmel.com,» Atmel, [Internett]. Available: http://www.atmel.com/webdoc/atmelstudio/atmelstudio.section.hzu\_gbq\_kc.html. [Funnet 14 November 2017]. |
| [8] | H. Dvergsdal, «www.snl.no,» 28 September 2017. [Internett]. Available: https://snl.no/kompilator\_-\_IT. [Funnet 14 November 2017]. |
| [9] | Z. W. Yan, Kingbright, 12 November 2001. [Internett]. Available: http://pdf1.alldatasheet.com/datasheet-pdf/view/167067/KINGBRIGHT/L7104GC.html. [Funnet 9 Oktober 2017]. |
| [10] | «www.arduino.cc,» 29 August 2015. [Internett]. Available: https://www.arduino.cc/en/Tutorial/Debounce. [Funnet 24 Oktober 2017]. |
| [11] | «www.pjcr.com,» PJCR, [Internett]. Available: https://www.pjrc.com/teensy/teensyduino.html. [Funnet 30 Oktober 2017]. |
| [12] | c. m. p. t. t. F. P. b. teachop, 1 August 2017. [Internett]. Available: https://github.com/collin80/FlexCAN\_Library/. [Funnet 30 Oktober 2017]. |
| [13] | 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]. |
| [14] | Texas Instruments, «www.ti.com,» Texas Instruments, Juli 2015. [Internett]. Available: http://www.ti.com/lit/ds/symlink/sn65hvd230.pdf. [Funnet 31 Oktober 2017]. |
| [15] | «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]. |
| [16] | 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]. |
| [17] | «www.pjcr.com,» Freescale Semiconductor, Inc, 2 Mai 2015. [Internett]. Available: https://www.pjrc.com/teensy/K66P144M180SF5RMV2.pdf. [Funnet 14 November 2017]. |
| [18] | T. O. Fredericks, 25 Juli 2014. [Internett]. Available: https://www.pjrc.com/teensy/td\_libs\_Metro.html. [Funnet 6 November 2017]. |
| [19] | «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]. |
| [20] | Oracle, «www.virtualbox.org,» 3 November 2017. [Internett]. Available: https://www.virtualbox.org/wiki/Downloads. [Funnet 4 September 2017]. |
| [21] | Canonical Ltd, «www.ubuntu.com,» Canonical Ltd, 2017. [Internett]. Available: https://www.ubuntu.com/download/desktop. [Funnet 4 September 2017]. |
| [22] | «www.wikipedia.org,» 27 Juli 2017. [Internett]. Available: https://en.wikipedia.org/wiki/Buildroot. [Funnet 6 November 2017]. |
| [23] | «www.wikipedia.org,» 24 Juni 2017. [Internett]. Available: https://en.wikipedia.org/wiki/ARM\_Cortex-A7. [Funnet 6 November 2017]. |
| [24] | «www.wikipedia.org,» 15 November 2017. [Internett]. Available: https://en.wikipedia.org/wiki/Raspberry\_Pi. [Funnet 13 November 2017]. |
| [25] | SK Pang Electronics Ltd, «www.skpang.co.uk,» SK Pang Electronics Ltd, 2017. [Internett]. Available: http://skpang.co.uk/catalog/images/raspberrypi/pican/PICANGPSACC\_V1.pdf. [Funnet 7 November 2017]. |
| [26] | «www.raspberrypi.org,» Raspberry Pi Foundation, 4 August 2017. [Internett]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md. [Funnet 13 November 2017]. |
| [27] | Apple Inc, «www.support.apple.com,» Apple Inc, 27 Februar 2017. [Internett]. Available: https://support.apple.com/en-us/HT204377. [Funnet 13 November 2017]. |
| [28] | «www.ubuntu.com,» 27 Februar 2015. [Internett]. Available: https://help.ubuntu.com/community/SSH. [Funnet 13 November 2017]. |
| [29] | «www.datamaskin.biz,» [Internett]. Available: http://www.datamaskin.biz/Systems/basic-computer-skills/200343.html#.WgtnxLaDrOQ. [Funnet 13 November 2017]. |
| [30] | Elecrom, «www.elecrom.com,» 6 Februar 2017. [Internett]. Available: http://www.elecrom.com/avr-tutorial-2-avr-input-output/. [Funnet 20 Oktober 2017]. |
| [31] | pighixxx, «www.arduino.cc,» 14 September 2016. [Internett]. Available: https://forum.arduino.cc/index.php?topic=147582.0. [Funnet 2 November 2017]. |
| [32] | InvenSense Inc., «www.invensense.com,» InvenSense Inc., 16 Mai 2012. [Internett]. Available: http://www.haoyuelectronics.com/Attachment/GY-521/mpu6050.pdf. [Funnet 2 November 2017]. |
| [33] | Arduino, «www.arduino.cc,» Arduino, [Internett]. Available: https://www.arduino.cc/en/Reference/Wire. [Funnet 2 November 2017]. |
| [34] | «www.pjrc.com,» [Internett]. Available: https://www.pjrc.com/teensy/pinout.html. [Funnet 6 November 2017]. |
| [35] | «www.arduino.cc,» Arduino, [Internett]. Available: https://playground.arduino.cc/Main/MPU-6050. [Funnet 31 Oktober 2017]. |