16.11.2017

Rapport, gruppearbeid MAS234 høst 2017

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

Innhold

Del 1 Mikrokontrollerkretser 4
Introduksjon 4
Autodesk Eagle 5
Spenningsregulator 5
Funksjonstest av spenningsregulator 6
Atmega168 6
Støy 7
Reset-funksjon 7
Atmel-ICE 8
LED 9
Styre LED ved bruk av Atmega168 10
Oppkobling 10
AtmelStudio 10
Fuses 11
Memories 11
Krysskompilering 11
”Hello World” 12
Overgang til Arduino 12
PWM – Pulsbredde modulasjon 14
PWM oppsett 14
PWM modus og frekvens 14
LED styrt med PWM med konstant dutycycle 14
Jevnt blinkende LED – Softblink 15
Tilpasning av lysnivå for øyet 15
Knapp for styring av lysdiode 17
Avbrudd - pin change interrupt 17
Debounce 18
ADC – analog til digital konvertering 18
Konfigurasjon av ADC 20
Registre: 20
Del 2 Arduino og I2C 22
Introduksjon 22
Metode 23

Innføring i Teensy 3.6 23
Oppkobling av Teensy 3.6 23
Bruk av terminalen i Arduino IDE 23
IMU mot Teensy 3.6 23
Overføre verdier fra IMU 24
Bruke IMU data 25
G-kraft sensor 25
Programbeskrivelse: 25
Diskusjon 26
Konklusjon 26
Del 3 Arduino/Teensy 3.6 og CAN-bus 27
Introduksjon 27
Metode 28
Dual CAN-bus adapter for Teensy 3.6 28
PCAN-USB FD adapter 28
Sende og motta CAN-meldinger 28
Setup 29
Loop 29
Sende CAN-melding 30
PcanView 30
Sende melding 31
Styre LED med meldingsID 31
Styre LED med innhold i melding 32
Rapportere verdier fra IMU via CAN-bus 32
Kontakt mellom Teensyer 33
Diskusjon 34
PcanView 34
Sende CAN-meldinger 34
Konklusjon 36
Del 4 Raspberry Pi 3/Buildroot og CAN 37
Introduksjon 37
Metode 38
Ubuntu og VirtualBox 38
Buildroot 38

Installasjon av delvis ferdig konfigurert oppsett 39
SD-kort 40
Koble opp hardware 40
Konfigurasjon av ethernettilkobling 41
Kommunikasjon over CAN-bus 42
Konklusjon 44
Diskusjon 44
MicroSD-kortet 44
Root 44
Kilder 45

Del 1 Mikrokontrollerkretser

Introduksjon

I denne delen av prosjektet skal vi koble opp og funksjonsteste en lineær spenningsregulator, basert på spesifikasjoner fra databladet og tilgjengelig utstyr i mekatronikklaben. Denne spenningsregulatoren skal forsyne en Atmega168 mikrokontroller. En lysdiode skal kobles på utgangen til spenningsregulatoren for å indikere at spenningen er OK. Vi skal koble en lysdiode på en av portene til mikrokontrolleren og få denne til å blinke kontinuerlig ved å programmere mikrokontrolleren. Vi skal bruke pulsbredde modulasjon til å dimme en lysdiode. Deretter skal vi justere lysstyrken slik at lysdioden blinker ”mykt”. Vi skal også skalere lysstyrken slik at øyet oppfatter lysendringen som lineær. En knapp skal kobles inn for å kunne endre verdi på en inngangspinne. En endring av verdi skal aktivere pin change interrupt, for å toggle en lysdiode. Videre skal en analog til digital omformer benyttes for å representere et analogt signal som en digital verdi. Den digitale verdien skal endre lysstyrken til dioden ved bruk av puls bredde modulasjon.

All programmering skal skje i AtmelStudio. Både en Atmel-ICE og en Arduino skal brukes som en overgang mellom PC og mikrokontrolleren. Alle valg skal baseres på spesifikasjoner og informasjon fra databladene til de forskjellige komponentene. Alt av kretser skal dokumenteres med tegninger i Eagle.

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.

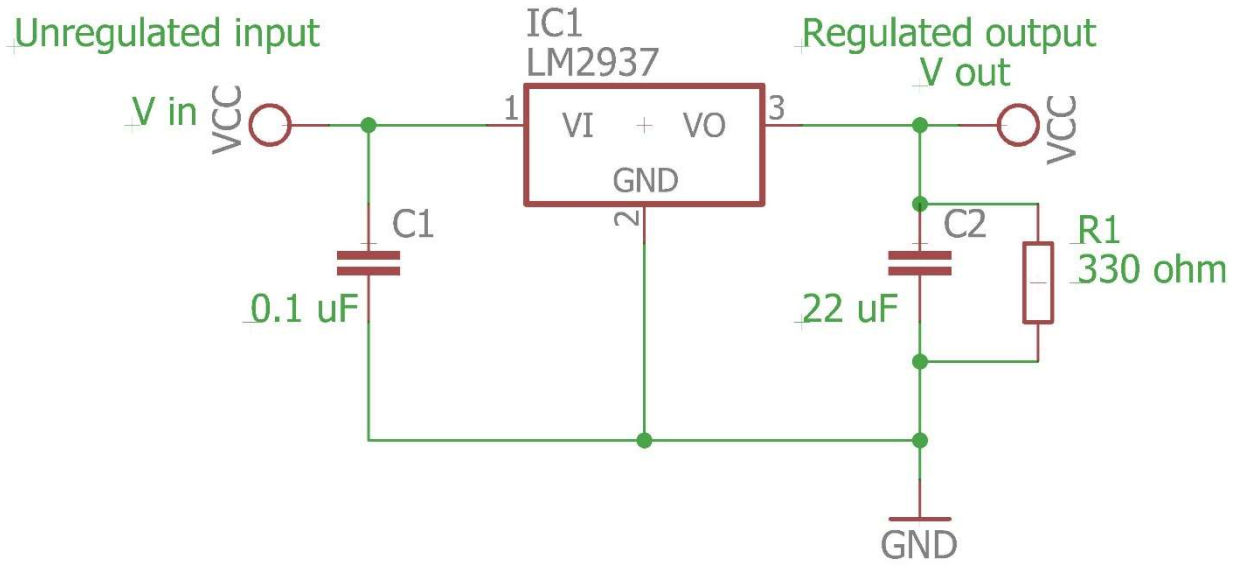
Til prosjektet vårt brukte vi en 5V lineær regulator, LM2931T. I følge databladet krever denne kondensatorer som ligger nærme regulatoren, mellom jord og input/output [1]. Størrelsene på kondensatorene var oppgitt til å være 0.1uF og 100uF. Fordi vi ikke hadde 100uF tilgjengelig under prosjektet, benyttet 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 330Ω.

$$\frac{Vout}{R1} = \frac{5V}{330\Omega} \approx 15mA$$

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

Kretsen tegnet med spenningsregulatoren

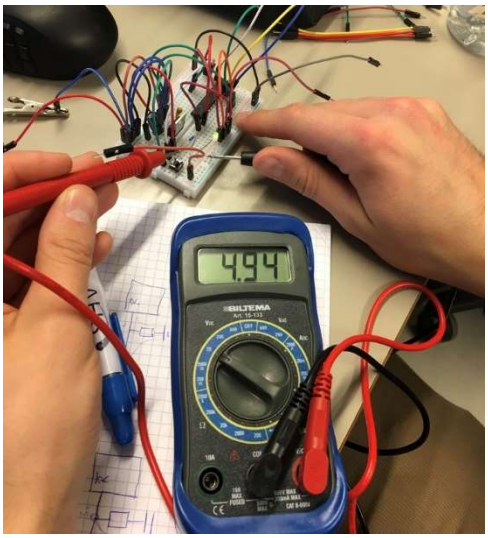


Figur 1

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 [1]. Vi brukte en lab-strømforsyning for å simulere 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).

Tester spenningen ut fra spenningsregulatoren



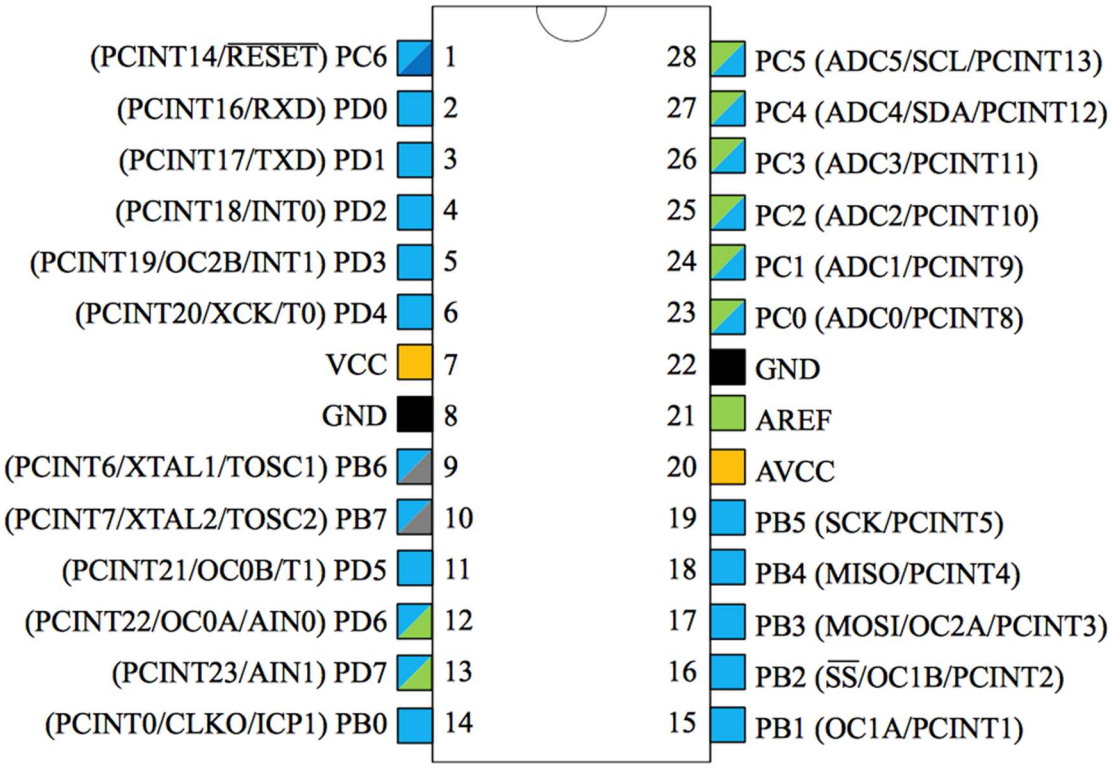
Figur 2

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



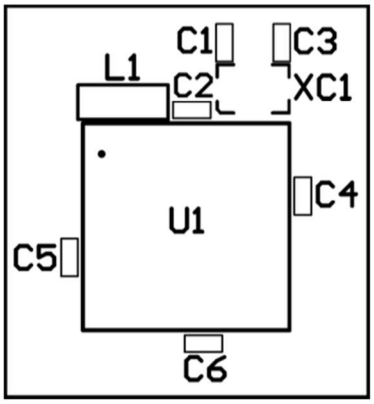
Figur 3

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.

For å hindre dette, anbefales det i databladet å koble inn kondensatorer mellom Vcc og jord [2]. 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).

Annbefalt oppkobling med kondensatorer nærme mikrokontrolleren



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.

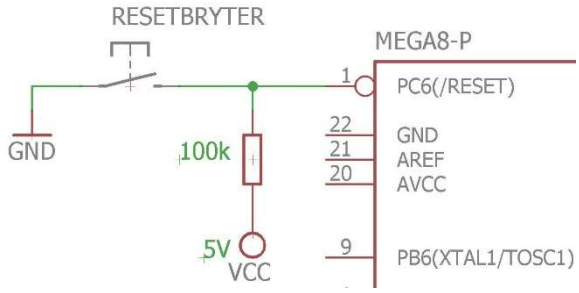
Reset-funksjonen i Atmega168 er aktiv lav. Det vil si at vi aktiverer den når koblingen til pinnen går til jord. Dette gjøres med en ekstern kobling.

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 [2].

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 100kΩ, så:

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

Ekstern *”pull-up” reset*



Figur 5

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].

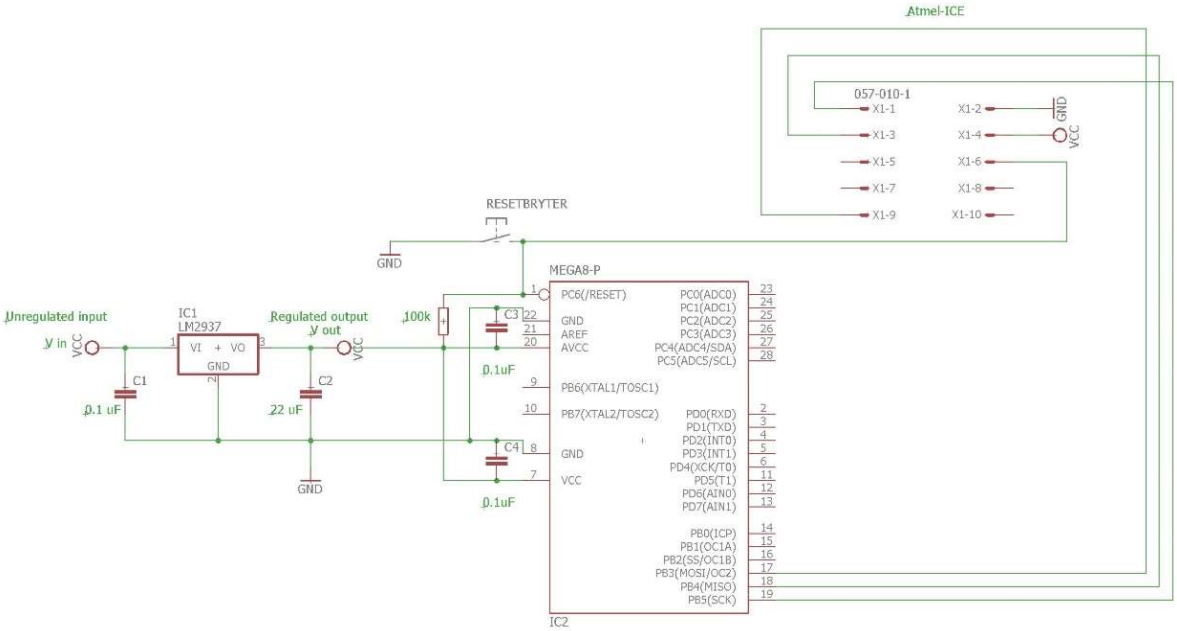
SCK, MISO og MOSI er alle pinner under PB-området på mikrokontrolleren, mens ground og VTG er henholdsvis jord og spenningsforsyning.

Atmel-ICE SPI Pin konfigurasjon

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

Figur 6

Koblingsskjema med Atmel-ICE



Figur 7

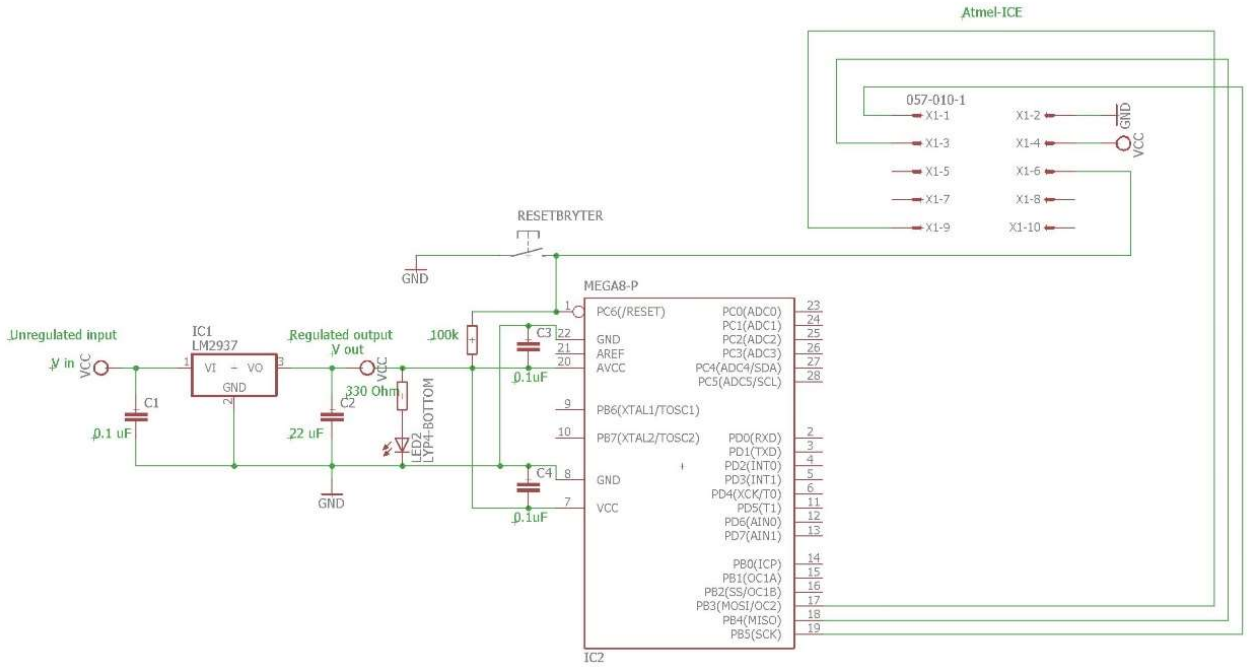
LED

Når man skal bruke en LED, er det viktig at man ikke overskrider spenning- og strømmerkinger, slik at man ikke ødelegger komponenten. Vi brukte en Kingbright L7104GC LED. I lysdioden 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 330Ω motstand i serie med denne.

$$\frac{Vout}{R1} = \frac{5V}{330\Omega} \approx 15mA$$

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

Koblingsskjema med LED på utgangen til spenningsregulatoren



Figur 8

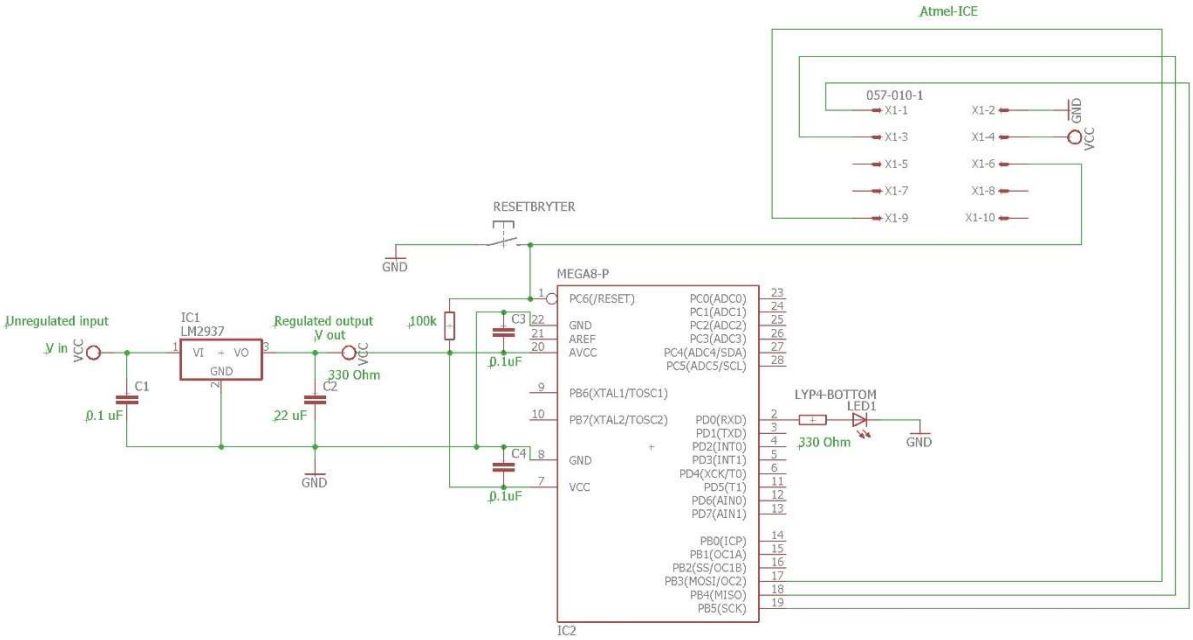
Styre LED ved bruk av Atmega168

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 330Ω, og videre til jord. Ved å koble den til jord, vil mikrokontrolleren ”source” strøm. Altså vil lysdioden 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 9

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 [2], og dette stemte overens med vår avleste signature fra AtmelStudio.

Leser av Target Volt og sjekker Device signature

Figur 10

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.

Memories

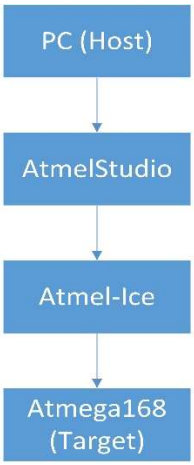
”Memories” er menyen hvor man laster inn programmet sitt til mikrokontrolleren. Her velger man programfilen, som vil være av type ”.hex”, og laster denne over på flashminnet til mikrokontrolleren. Ved å huke av ”Erase Flash before programming” og ”Verify Flash after programming”, vil man slette tidligere 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 til en kjørbar maskinkode. I praksis vil man ofte både skrive og kompilere kildekoden på et annet system enn det maskinkoden er tiltenkt å kjøre på. Ved å bruke en krysskompilator, kan man kompilere kildekoden 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



Figur 11

”Hello World”

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 [9]. Inni while-løkken vår programmerte vi PD0 til å gå høy/lav med en syklus på 1000ms (Figur 12).

”Hello World” i AtmelStudio

```
#ifndef F_CPU
#define F_CPU 1000000UL //Defines clockspeed to be 1 MHz. Internal clock is 8MHz, then divided by 8 in the fuse-settings.
#endif

#include <avr/io.h>     //Headerfile for input/output operations.
#include <util/delay.h>     //Headerfile to access delay functions.


int main(void)
{
    DDRD = 0b00000001;  //DDR = data direction. D = D-area at the controller. Sets PD0 (pin 2) to a value of 1 (defines it to be an output).
    PORTD = 0b00000000; //Sets output PD0 to be default low.

    while (1)
    {
        //Turns on the light.
        PORTD |= 0b00000001;        //Sets PD0 to high.
        _delay_ms(1000);

        //Turns off the light.
        PORTD &= 0b11111110;        //Sets PD0 to low.
        _delay_ms(1000);

    }
}
```

Figur 12

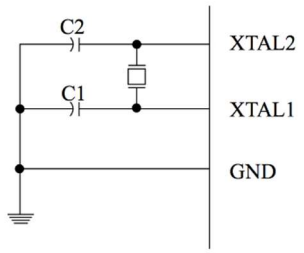
Overgang til Arduino

På grunn av begrenset tilgang på Atmel-ICE, valgte vi å gå over til å bruke en Arduino Nano for å laste opp 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. Se figur 15 for oppkobling med bruk av Arduino Nano.

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 konfigurere fuses. For å enkelt sette fuses, valgte vi å bruke en fuse kalkulator [10].

For å få en mer nøyaktig og stabil klokkehastighet, koblet vi opp en ekstern krystall på 16MHz, med tilhørende kondensatorer. Vi fant veiledning for oppkoblingen i databladet til Atmega168 (figur 13) [2]. Vi måtte da endre innstillingene i fuses til ekstern krystall oscillator, uten clock divider.

Krystall oscillator tilkobling



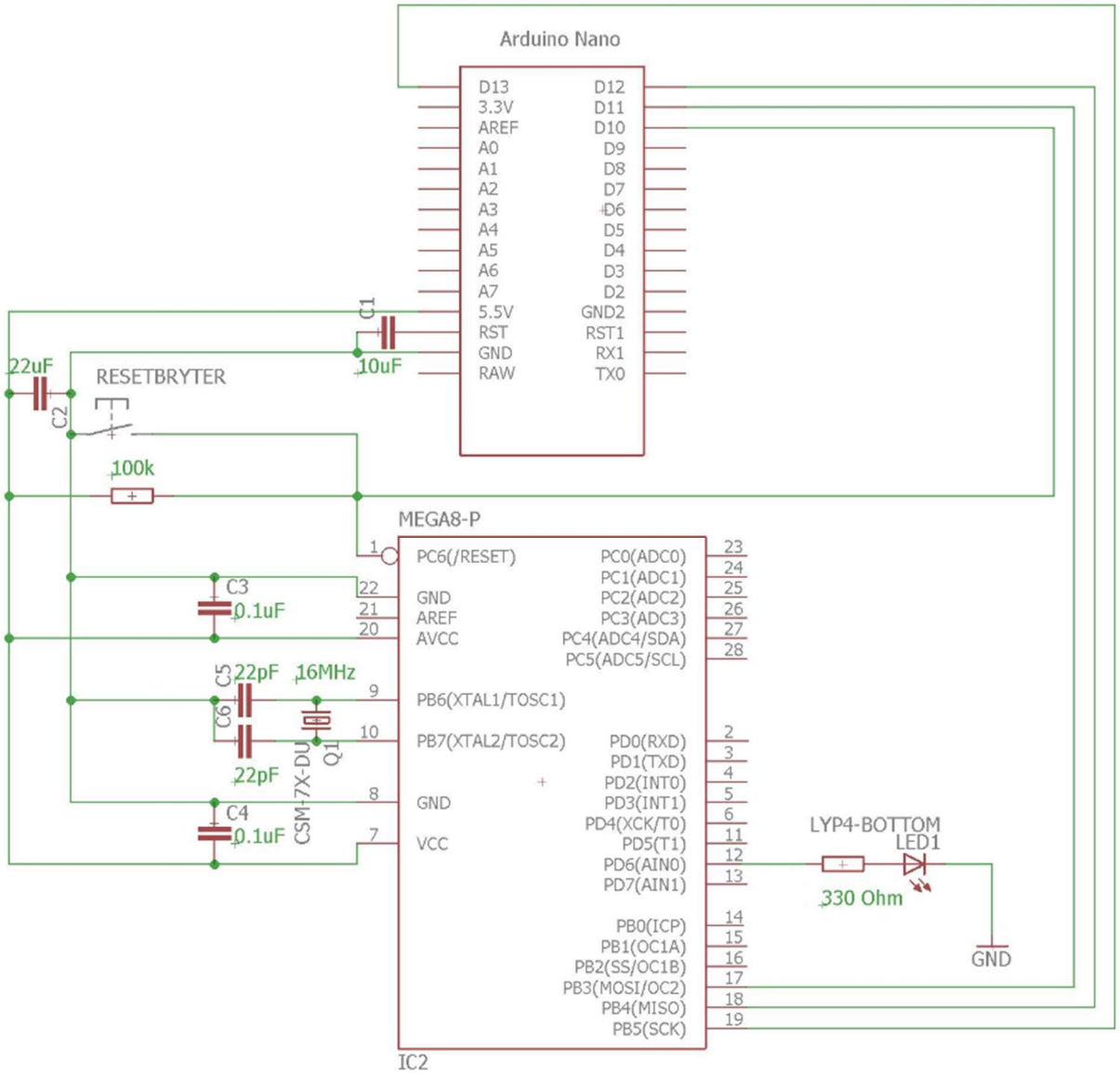
Figur 13

Størrelser for kondensatorer til krystalloscillator

Frequency Range [MHz]	CKSEL[3:1][(2)]	Range for Capacitors C1 and C2 [pF]
0.4 - 0.9	100[(3)]	–
0.9 - 3.0	101	12 - 22
3.0 - 8.0	110	12 - 22
8.0 - 16.0	111	12 - 22

Figur 14

Koblingsskjema med Arduino Nano



Figur 15

PWM – Pulsbredde modulasjon

PWM står for pulsbredde modulasjon. Normalt ved å endre lysstyrke på en diode, eller hastigheten til en motor vil man bruke 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. Det vil si 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 og 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 for eksempel 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)*$2^{\text{antall bit}}$.

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 $2^{\text{antall bit}}$. Clear on compare match betyr: utgangen knyttet til OCA (PD6 ved 8 bit) starter høy ved teller-verdi 0, og blir satt lav ved tellerverdi lik dutycycle*$2^{\text{antall bit}}$. PWM frekvensen er hvor mange telle-sekvenser PWM styringen vil kjøre per sekund.

LED styrt med PWM med konstant dutycycle

I databladet til lysdioden står det at i PWM-mode tåler lysdioden en peakstrøm på 140mA ved 1/10 duty cycle, 0.1ms pulse width [5]. 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:

$$PWM - frekvens = \frac{CLK}{clk{-}dividier \times PWM{-}prescaler \times 10{-}bit\ counter} = \frac{16Mhz}{1 \times 1024 \times 2^{10}} = 15.26\ Hz$$

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 med clear OCA on compare match med prescaler satt til 256.

Disse parameterne førte til følgende frekvens:

$$PWM - frekvens = \frac{CLK}{clk{-}dividier \times PWM{-}prescaler \times 8{-}bit\ counter} = \frac{16Mhz}{1 \times 256 \times 2^{8}} = 244.14\ Hz$$

Denne nye frekvensen på 244.14 Hz resulterte i jevnt lys fra dioden ved dutycycle varierende fra 0 til 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.

$$Økning\ av\ dutycyle\ pr.OVF\ int = \frac{maks\ dutycyle \times 2}{PWM{-}frekvens \times blinkesyklus} = \frac{100 \times 2}{244\ Hz \times 2\ s} = 0.4096$$

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.

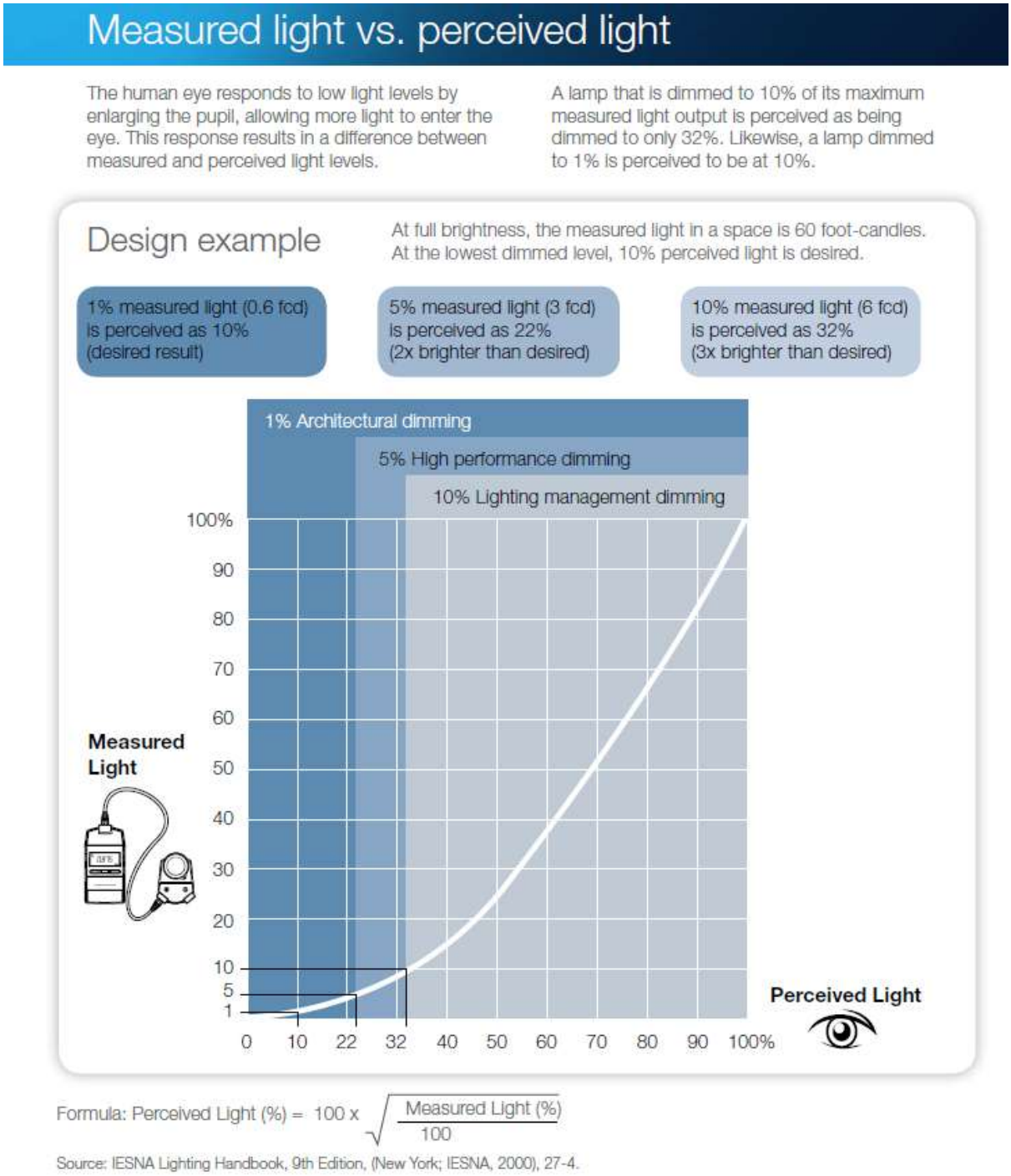
Tilpasning av lysnivå for øyet

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 (figur 16) [11].

$$\text{Øyets lysnivå} = \frac{(\text{teoretisk lys})^2}{100}$$

Vi lage en ny variabel som regner ut lysnivå tilpasset øyet, så bruker vi denne variabelen til å sette OCR0A verdien, da vil man effektivt tilpasse lysnivået relativt til øyets oppfatning.

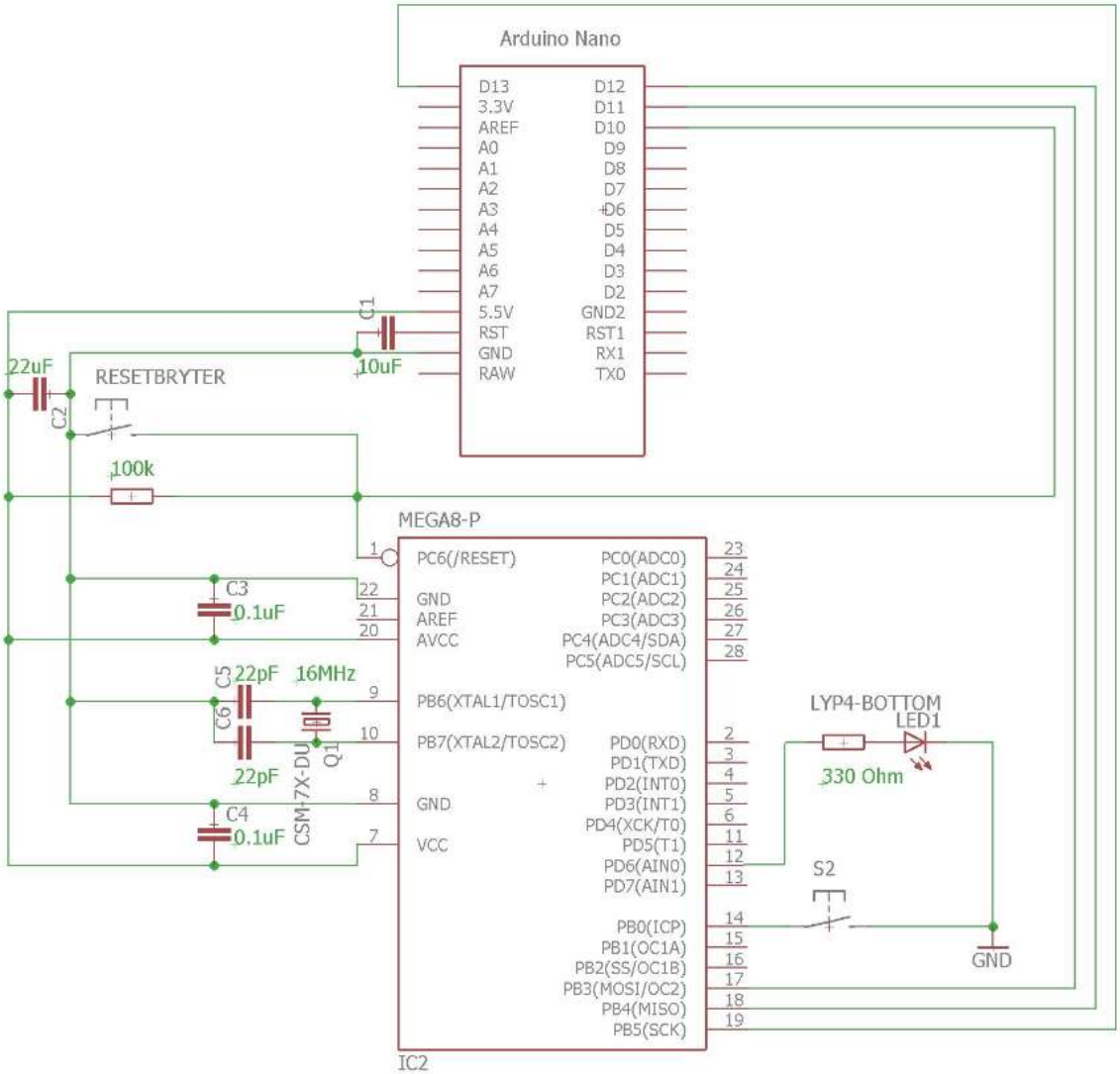


Figur 16

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 (figur 17). 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.

Koblingsskjema for lysdiode som styres av knapp



Figur 17

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 pin change 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 deaktivere globale interrupts i pin change interruptvektorfunksjonen, 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 [12].

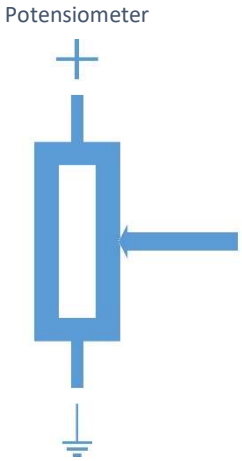
Dette er ikke en optimal løsning, fordi delay vil skape ventetid i systemet som kunne blitt brukt til noe annet. Funksjonsbeskrivelsen oppfylles, da denne løsningen toggler lysdioden uten debounce, og vil fungere bra til dette bruket.

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.



Figur 18

$$Strøm = \frac{5V}{10k\Omega} = 0.0005A = 0.5\ mA$$

$$Spenningsfall = 0.5mA \times potensiometer$$

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.

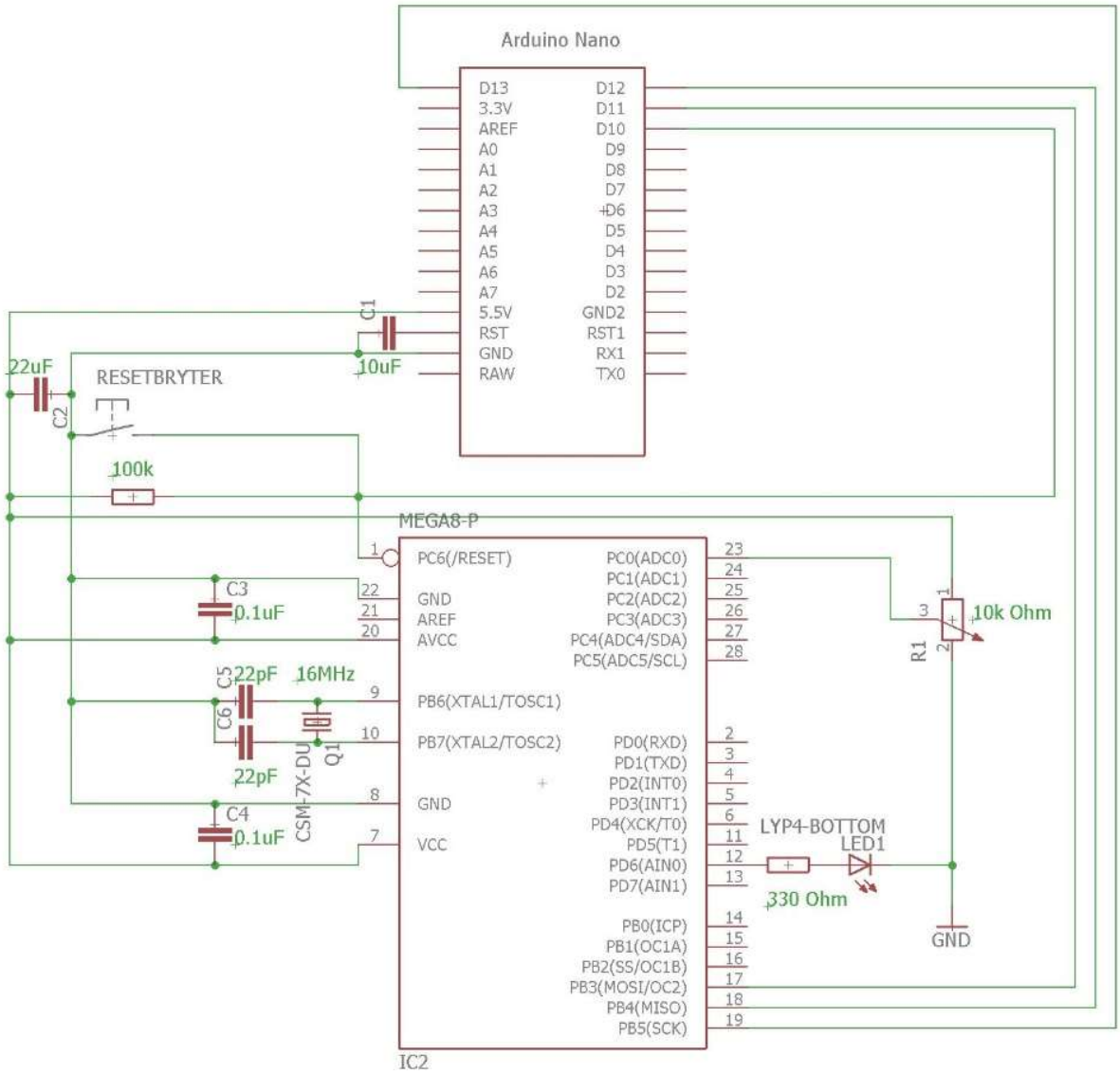
Når man benytter et analogt spenningssignal, trenger man også en referansespenning å sammenligne mot. Med en Atmega168 har man tre valg [2]:

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 (figur 19).

Koblingsskjema for Analog styring av LED



Figur 19

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 [2].

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 prescaler-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”.

Del 2 Arduino og I2C

Introduksjon

I del 2 av prosjektet skal vi bli kjent med mikrokontrolleren Teensy 3.6. Vi skal bruke denne mikrokontrolleren sammen med IMU'en MPU 6050 GY-521 for å lese av vinkelhastighet og akselerasjon i alle 3 retninger, samt temperatur. For å lese av disse verdiene skal vi bruke terminalen i Arduino IDE. Dataen vi leser av MPU 6050 skal brukes til å lage en g-kraft sensor. Denne sensoren skal toggle en lysdiode på Teensey 3.6 hvis endring i g overskrider 1 over en tidsperiode på 1 sekund.

Metode

Innføring i Teensy 3.6

I denne oppgaven skal vi 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 [13]. Man må også sørge for at korttype i Arduino IDE er satt til Teensy 3.6.

Oppkobling av Teensy 3.6

Teensy 3.6 kobles mot PC med mini-USB kabel. Denne kabelen tilfører Teensy 3.6 med drivspenning og den kan både sende og motta data.

For å teste forbindelsen mellom Teensy og PC, lastet vi opp eksempelprogrammet «tutorial 1 – blink». Programmet ble lastet opp til Teensy uten feil og lysdioden på Teensy blinket hvert andre sekund.

Bruk av terminalen i Arduino IDE

Arduino IDE inneholder terminal funksjonen. Denne kan være et viktig verktøy for feilsøking ved Arduino programmering. Man kan bruke terminalen for å sjekke verdier på variabler, inndata, utdata og mye mer.

For å skrive til terminalen må man først kjøre Serial.begin(sett inn baudrate her); Det er da viktig at man bruker samme baudrate i overvåkningsterminalen i Arduino IDE som man printer med Serial.begin funksjonen. Når man har kjørt Serial.begin funksjonen, kan man bruke Serial.print(sett inn det man vil printe her) for å printe informasjon til terminalen.

For å teste om det fungerte å skrive til terminalen, lagde vi et program som printet meldingen «Hello serial monitor» kontinuerlig med en forsinkelse på 1 sekund.

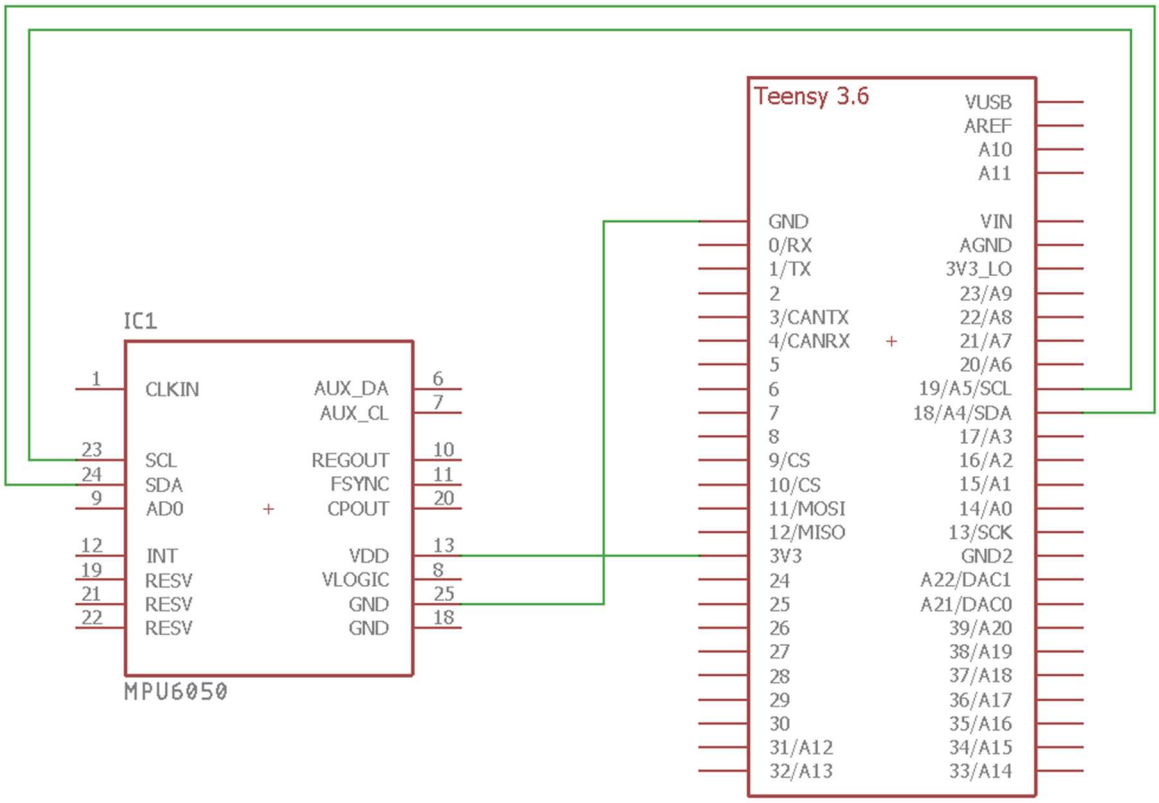
IMU mot Teensy 3.6

Vi skal nå koble en IMU (Inertial Measurement Unit) mot vår Teensy 3.6. IMU'en vi bruker er en MPU 6050 GY-521. Denne har 3 gyroskop som kan måle vinkelhastighet, 3 akselerometer som kan måle akselerasjon, og en temperatursensor [14].

I databladet til IMU'en leste vi at den skulle ha en drivspenning i området 2.375V til 3.46V [15]. Vi valgte derfor å bruke Teensyen utgangsspenning på 3.3V til drifte IMU'en.

Mellom Teensy og IMU blir det brukt I2C-kommunikasjon. Denne kobles slik at SCL og SDA pinne på IMU går til SCL og SDA pinne på Teensy (figur 20). AD0 kan bli brukt for å endre I2C adressen til IMU'en. Hvis AD0 er høy(3.3V) vil IMU'en ha I2C-adresse 0x69. Hvis AD0 er lav vil IMU'en ha I2C-adresse 0x68. Om man vil bruke flere IMU'er, kan man variere I2C-adressene syklisk slik at man kan aksessere IMU'ene en etter en. Med en høy syklusfrekvens kan man da ha flere IMU'er på samme I2C-bus-system.

Koblingsskjema Teensy 3.6 og MPU 6050



Figur 20

Overføre verdier fra IMU

Dataen fra IMU'en blir lagret i et FIFO register. Man kan lese i databladet til MPU 6050 hvilken data som blir lagret i hver adresse av FIFO registeret [15]. For å overføre verdier fra IMU'en bruker vi funksjoner i wirebiblioteket som ligger standard i Arduino IDE [16].

Bruke wire til å overføre data fra IMU:

```
Wire.beginTransmission(MPU_addr);
```

Starter sending fra IMU med I2C-adresse = MPU_addr (0x68).

```
Wire.write(0x3F);  // starting with register 0x3F (ACCEL_ZOUT_H)
```

Denne kommandoen skriver til IMU'en og sier hvilken adresse i FIFO registrene den skal begynne å sende fra.

```
Wire.endTransmission(false);
```

Fortsett å sende.

```
Wire.requestFrom(MPU_addr,2,true);  // request a total of 2 registers
```

Spør IMU'en om å sende 2 bytes fra I2C-adresse = MPU_addr (0x68).

```
AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
```

Akselerasjonsverdien er int_16t, men IMU'en lagrer denne verdien i to int_8t verdier. Hver gang Wire.read blir kjørt, leses neste byte i FIFO registeret. For å lese disse to 8-bit verdiene inn i én 16-bit

verdi skjer følgende: den første byten leses inn i int_16t variabelen AcZ, men hver bit blir forskjøvet 8 plasser. Den andre byten leses også inn i AcZ, men denne blir ikke forskjøvet.

For at vi kan lese av verdiene, bruker vi Serial.print funksjonen for å skrive verdiene til terminalen i Arduino IDE.

Bruke IMU data

Vi sjekket akselerasjonsverdien i Z-retning forårsaket av tyngdeakselerasjonen. Når IMU'en var i ro målte vi verdien til cirka -18000. Basert på denne målingen lagde vi et program der vi fikk meldinger til terminalen ut ifra målte verdier. Hvis verdien var over 10000, fikk vi melding om at brikken var opp ned. Hvis verdien var mindre enn -10000 fikk vi melding om at brikken var i normal stilling.

G-kraft sensor

Vi skal nå lage et program som skal måle endringer i g-krefter. Hvis man får en større endring enn 1g på ett sekund, skal lysdioden på Teensyen toggles.

I databladet til MPU 6050 leste vi at akselerometret kunne skaleres for ±2, 4, 6, 8 eller 16g [15]. Rådataverdi som vil representere 1g følger denne formelen:

$$Rådata\ som\ representerer\ 1g = \frac{32768}{maks\ g}$$

Programbeskrivelse:

Programmet er laget med en for-løkke som kjører 10 ganger, eller frem til LED toggles. I løkken vil akselerasjonsverdien i Z-retning bli målt av IMU'en og lest via wire funksjonen. Vi laget et array med 10 plasser. Hver gang løkken ble kjørt, ble det lest inn rådataverdier fra akselerometeret. Disse rådataverdiene ble kompensert for tyngdeakselerasjonens påvirkning før de ble lest inn i arrayet. Plassnummeret verdien ble lest inn på i arrayet startet på nummer 0, før den ble inkrementert med 1 hver gang løkken ble kjørt. Hvis verdien til plassnummer 9 i arrayet var lik 0, ville akselerasjonsverdien bli lest inn her også. Dette ble gjort for å forhindre en stor differanseverdi første gang for-løkken ble kjørt. Hver gang for-løkken kjørte, ble det regnet ut en differanseverdi mellom innlest verdi i arrayet og forrige innleste verdi. Denne differansen ble akkumulert i en totaldifferansevariabel. Totaldifferansen ble skalert til g-krefter. Hvis denne endringen i g-kraft overskred 1g, ville den toggle lysdioden på Teensyen og nullstille totaldifferansevariabelen. Hvis endringen i g-kraft ikke overskrider 1g etter at for-løkken har kjørt 10 ganger, nullstilles totaldifferansevariabelen og tellinger av for-løkker startes på nytt.

Diskusjon

Oppgaven spesifiserte at man skulle måle endring i g-kraft i Z-retning over en tidsperiode på 1 sekund; hvis denne endringen overskred 1g skulle en lysdiode toggles. Den første idéen vi lagde program for, baserte seg på å sample data hvert sekund, for så å regne ut differansen med data samplet for 1 sekund tidligere. Denne løsningen kunne ikke måle endringer som skjedde mellom samplingspunktene. Dette førte til at vi ikke fikk endring som skjedde mellom disse punktene i differanseutregningen. Vi valgte derfor å sample data hvert 100ms, for så å summere differansen mellom disse. Vi fikk nå en mye mer nøyaktig måling.

Det var vanskelig å få nøyaktig testing av den ferdige g-kraft sensoren, fordi det er umulig å simulere en endring som tilsvarer 1g manuelt. Ved å bevege IMU'en raskt opp og ned, fikk vi utslag på målingene som vi brukte for å konkludere med at g-kraft sensoren fungerte.

Konklusjon

Kommunikasjon mellom terminal og IMU via Teensy fungerte bra. Ved bruk av wirebiblioteket i Arduino IDE fikk vi lest av data fra FIFO registeret til IMU'en. Vi behandlet dataen vi leste av, for så å bruke denne til å styre lysdioden på Teensy 3.6.

Vi lagde en g-kraft sensor som målte endringer i akselerasjon i Z-retning. Ved en endring på mer enn 1g over en tidsperiode på 1 sekund, skulle lysdioden på Teensyen toggle. Hvis en endring større enn 1g ble målt, eller at tidsperioden for måling overskred 1 sekund, ville målt totaldifferanse nullstilles og en ny måling ble startet.

Testing av g-kraft sensoren viste at den reagerte bra på endring av akselerasjon og togglet lysdioden på Teensyen.

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) er en feiltollerant og robust nettverksstandard som brukes veldig mye i bilbygging. Fordelene med CAN 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 bruke biblioteket FlexCAN på Arduino for å få tilgang til CAN-funksjonene [17].

Dual CAN-bus adapteren inneholder to transceivere; en for Can0 og en for Can1. Adapteren inneholder også en 120Ω terminerings motstand [18].

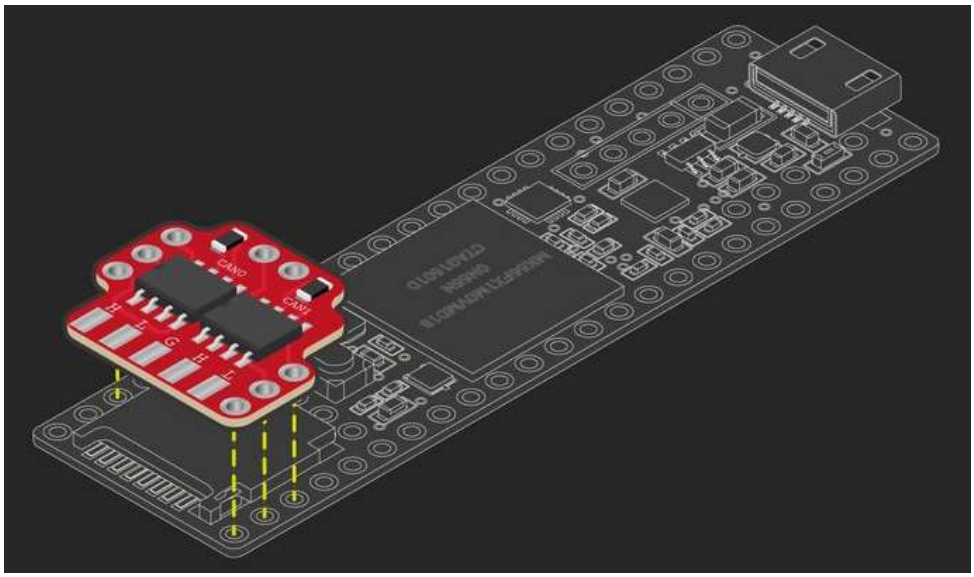
Metode

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. 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 støy, men vi slipper da å ha adapteren permanent festet til Teensy-brikken. Teensyen og CAN-bus adapteren kobles som vist i figur 21. Så må det kobles til jord og Vcc på 3.3V som er oppgitt i databladet [19].

Kobling Dual CAN-bus adapterpå Teensy 3.6



Figur 21

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 [20]. Vi bruker brukermanualen til PCAN-USB, CAN-grensesnitt for USB, som veiledning for å sette opp PcanView [21].

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.

Setup

For å få det til å funke må vi endre på deler av koden. Vi legger til CAN-filteret «defaultMask» i «setup» (figur 22). 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 kretsen. Vi bruker «begin» funksjonen og setter baudraten til Can0 til 250kbit/s. Vi aktiverer «defaultMask» filteret for Can0. Vi setter Tx = 1 og Rx = 1 for å bruke de alternative «receive» og «transmit» pinnene [22].

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 til lav, slik at de kan kommunisere med PCAN (figur 23).

Setup - defaultMask

```
void setup(void)
{
  //defaultMask to apply to all mailboxes
  struct CAN_filter_t defaultMask;

  // Default mask is allow everything
  defaultMask.flags.remote = 0;
  defaultMask.flags.extended = 0;
  defaultMask.id = 0;
```

Figur 22

Setup

```
  Can0.begin(250000,defaultMask, 1, 1);
  Can1.begin();

  //Using the alternate Can0 pins and set the as outputs
  pinMode(28, OUTPUT);
  pinMode(35, OUTPUT);

  //Setting the value of the pins low
  digitalWrite(28, LOW);
  digitalWrite(35, LOW);
```

Figur 23

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 PcanView. Vi kjører en while-løkke som sjekker om vi mottar data i Can0; når dette er sant vil while-løkke kjøres. Først vil meldingen fra Can0 leses inn i variabelen «inMsg». Deretter vil vi printe ut innholdet i alle databitsene som sendes. Maksimalt kan det sendes 8 bytes med data i en melding når FlexCAN-biblioteket brukes. Dette står i README filen [17]. 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 24). Vi kompilerer og laster opp programmet til Teensyen.

Loop

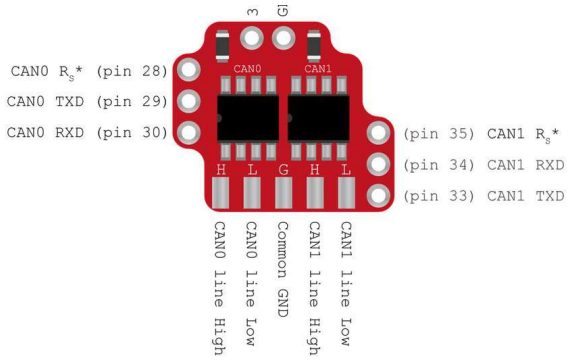
```
void loop(void)
{
  //Declares a variable inMsg which contains the data of the recived CAN-message
  CAN_message_t inMsg;
  while (Can0.available())
  {

    Can0.read(inMsg); //Reads the message which is recived in Can0

    Serial.print("CAN bus 0: "); hexDump(8, inMsg.buf);  //Writes out the informaton in the message
    Serial.println(inMsg.buf[2]);   //Writes out the information in the 3rd byte
    Serial.println(inMsg.id);   //Writes out the message ID

    Can0.write(inMsg);    //Writes a message and sends it through Can0, the message contains the same data as inMsg
  }
}
```

Figur 24

Sende CAN-melding

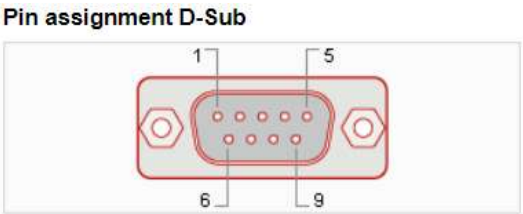
Vi vil nå koble sammen PCAN og Teensy 3.6 for å sende meldinger ved hjelp av CAN-bus. Dette gjøres ved å koble «Can0 line High» fra CAN-adapteren inn på CAN-H på PCAN og «Can0 line Low» fra CAN-adapteren inn på CAN-L på PCAN (Figur 25 og 26).

Dual CAN-bus adapter koblingsskjema



Figur 25

PCAN koblingsskjema



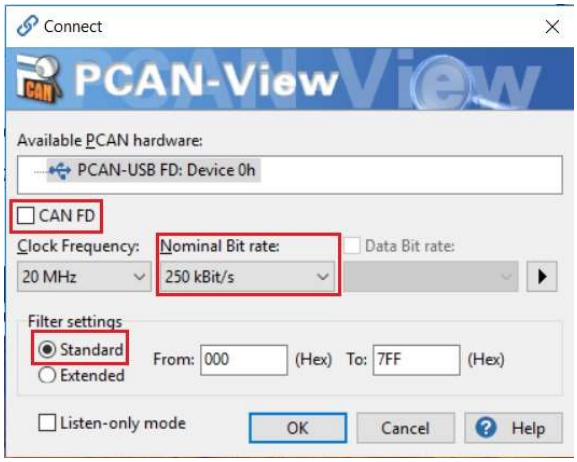
Pin	Pin assignment
1	not connected / optional +5V
2	CAN-L
3	GND
4	not connected
5	not connected
6	GND
7	CAN-H
8	not connected
9	not connected

Figur 26

PcanView

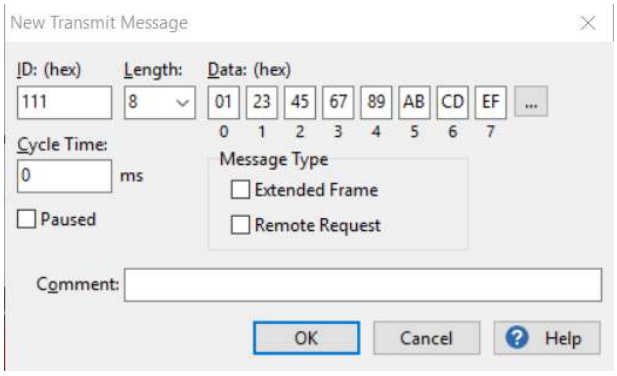
Vi vil nå generere en melding i PcanView som skal leses av Arduino-programmet og returneres til avsenderen. Når programmet startes vil menyen i figur 27 dukke opp. Der velger vi PCAN-kabelen, sørger for at «CAN FD» er umerket, vi sjekker at bitraten er riktig og at vi har valgt standard ramme til CAN-ID'en. For å generere en melding trykker vi på «New message», da kommer menyen i figur 28 frem. Der kan vi velge ID, lengde på meldingen (fra 0 til 8 byte), data og syklus tid. Vi setter meldingslengden 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



Figur 27

PcanView – Ny melding

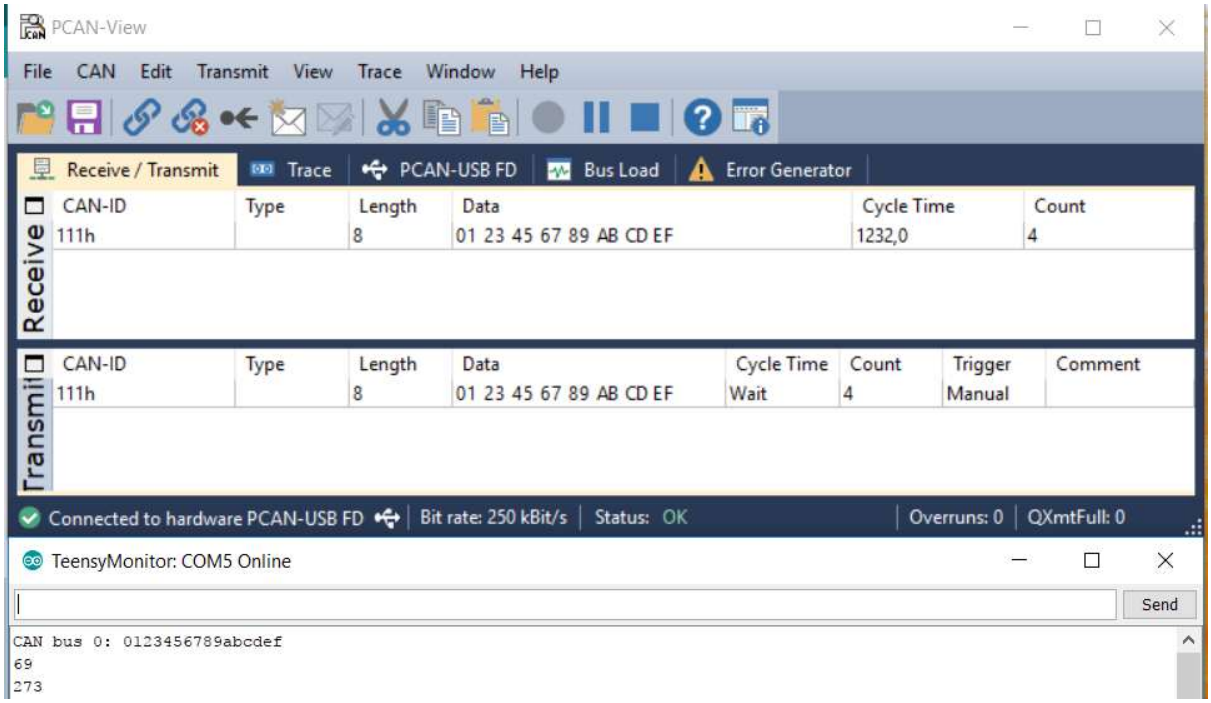


Figur 28

Sende melding

Nå kan vi sende meldingen vi har laget. Hvis vi åpner «Seriell overvåker» i Arduino IDE, 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). Til slutt printes meldingsID'en (0x111 = 273). Som vi ser i figur 29, vil meldingen vi sender også mottas i PcanView.

PcanView receive og transmit



Figur 29

Styre LED med meldingsID

Vi vil styre lysdioden på Teensy 3.6 kortet ved hjelp av meldingsID'en. Når Teensyen leser en melding som har ID 0x21, skal lysdioden toggles. For å få til dette deklarerer vi en konstant «ledPin» som er lik 13. 13 er adressen til lysdioden på Teensy 3.6. I «setup» definerer vi «ledPin» som en utgang, og setter den høy, slik at den skrus på når programmet lastes inn. I «loop» bruker vi en if-setning som sjekker om den mottatte CAN-bus meldingen har ID 0x21. Hvis dette er sant vil «ledPin» inverteres (figur 30).

Styre LED med meldingsID

```
void loop(void)
{
  CAN_message_t inMsg;
  while (Can0.available())
  {
    Can0.read(inMsg);

    if (inMsg.id == 0x21) //Checks if Can0 recives a message on the ID 0x21
    {
      digitalWrite (ledPin, !digitalRead(ledPin));  //If the if-statement is true, the LED on the teensy toggels
    }
  }
}
```

Figur 30

Styre LED med innhold i melding

Vi vil styre lysdioden på Teensy 3.6 ved hjelp av innholdet i meldingen. Når det minst signifikante bit i første byte er 1 vil «ledPin» settes høy. Hvis det minst signifikante bit i første byte er 0 vil «ledPin» settes lav. «inMsg.buf[7]» inneholder kun det første bytet som mottas i meldingen. 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 0x22 skal «ledPin» settes høy. Hvis «inMsg.buf[7]» OG'et med et byte som er xxxxxxx1 er lik 0 og meldingen kommer i fra ID 0x22, skal «ledPin» settes lavt. (figur 31).

Styre LED med innhold i melding kode

```
void loop(void)
{
  while (Can0.available())
  {
    CAN_message_t inMsg;

    // If the least significant bit in the first byte AND 1 equals 1, and the message ID is 0x22
    if (((inMsg.buf[7] &= (1)) == (1)) && (inMsg.id == (0x22)))
    {
      digitalWrite(ledPin, HIGH);   // The LED is written high
    }
    // If the least significant bit in the first byte AND 1 equals 0, and the message ID is 0x22
    if (((inMsg.buf[7] &= (1)) == (0)) && (inMsg.id == (0x22)))
    {
      digitalWrite(ledPin, LOW);    // The LED is written low
    }
  }
}
```

Figur 31

Rapportere verdier fra IMU via CAN-bus

Vi vil rapportere verdier fra akselerometeret på en IMU 6050 ved å benytte meldingsID 0x20. Meldingen skal sendes med en fast rate på 1 Hz. For å ha kontroll på meldingsraten bruker vi Metro-biblioteket [23]. Vi inkluderer «Metro.h» for å få tilgang, så definerer vi en metrovariabel, «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 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 fra «acc». Deretter bruker vi Can0 til å skrive «outMsg» (Figur 32).

Rapportere verdier fra IMU via CAN-bus

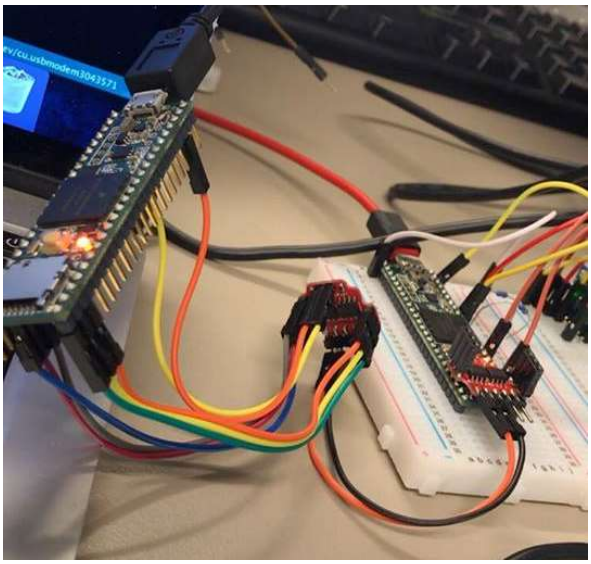
```
void loop(void)
{
  while (ledMetro.check() == 1)
  {
    CAN_message_t outMsg;   // Defines a CAN message calles outMsg

    Wire.beginTransmission(MPU_addr);
    Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
    Wire.endTransmission(false);
    Wire.requestFrom(MPU_addr,6,true);  // request a total of 6 registers

    int8_t acc [6];
    acc[0]=Wire.read();  // 0x3B (ACCEL_XOUT_H)
    acc[1]=Wire.read();  // 0x3C (ACCEL_XOUT_L)
    acc[2]=Wire.read();  // 0x3D (ACCEL_YOUT_H)
    acc[3]=Wire.read();  // 0x3E (ACCEL_YOUT_L)
    acc[4]=Wire.read();  // 0x3F (ACCEL_ZOUT_H)
    acc[5]=Wire.read();  // 0x40 (ACCEL_ZOUT_L)

    outMsg.id = 0x03;   // Gives the message an ID
    outMsg.len = 6U;    // The lenght of the message is 6 unsigned int

    // Copying the values from acc into outMsg.buf, takes all 6 values with
    memcpy (outMsg.buf, acc , 6U);
    Can0.write(outMsg);  // Writes outMsg onto the CAN network
  }
}
```

Figur 33

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 34). Programmet som er lastet inn, 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 lysdioden høy hvis ID'en er 0x21, og lav hvis ID'en er 0x22. Lysdioden vil da blinke.

Kobling mellom to Teensyer



Figur 34

Diskusjon

PcanView

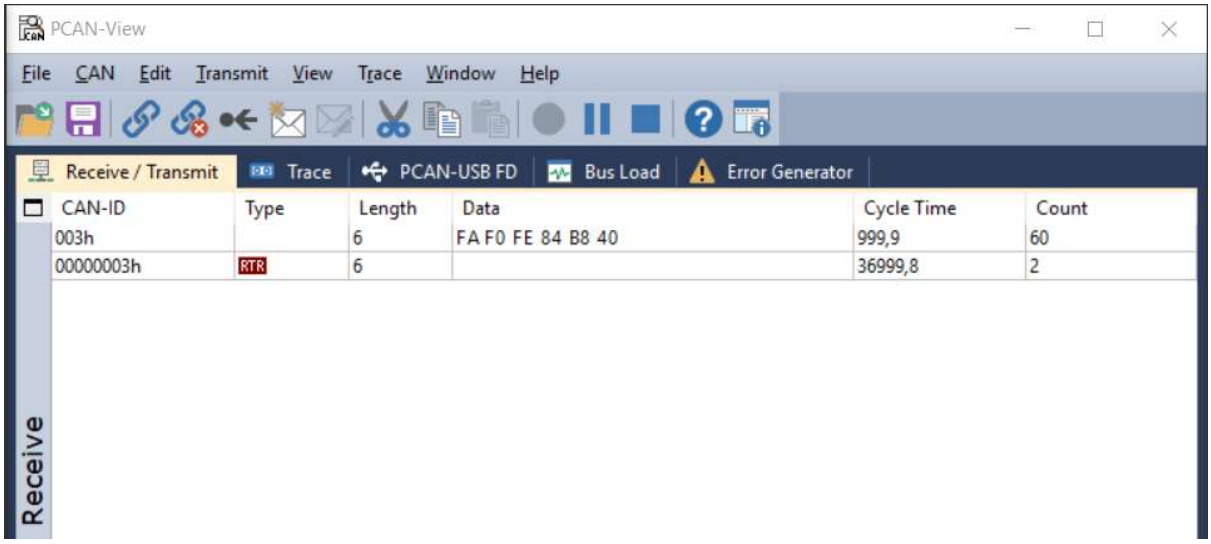
Det første vi må velge er bitrate. Bitraten bestemmer hvor mye og hvor fort data sendes i et CAN-bus nettverk. Bitraten kan variere fra 1Mbit/s til 5kbit/s. For å ha en raskere bitrate må man ofre lengden på bus-kabelen. Vi velger en bitrate på 250kbit/s, fordi det er standardhastigheten i FlexCAN.

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 standardrammer (11 bit ID'er) eller utvidede rammer (29 bit ID'er). En 11 bits ID vil si at vi har 2^{11} = 2048 mulige ID'er. En 29 bits ID vil si 2^{29} = 536 870 912 mulige ID'er [24]. På grunn av at vi kun skal ha ett lite nettverk, velger vi å bruke standardrammer. CAN-bus fungerer slik at meldingene prioriteres ut i fra ID'en. Hvis meldingen har en lav ID, vil den prioriteres over en melding med 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, 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 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, kunne vi motta mange flere meldinger før vi fikk to «RTR». Vi kunne da sende mellom 30 og 150 meldinger. Se figur 35.

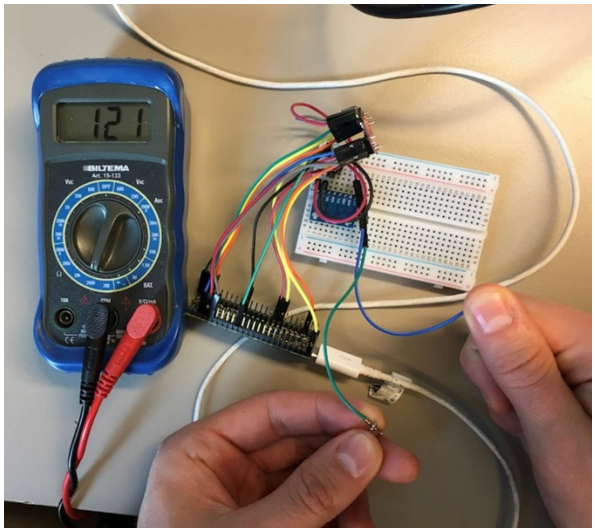
Retry meldingene vi fikk i PcanView



Figur 35

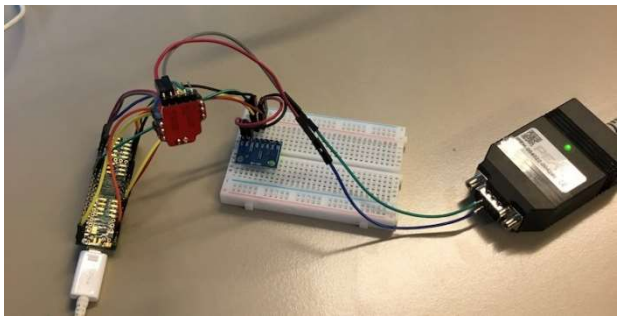
Vi mottok fortsatt data med lengde 6 bytes, men den siste byten var som oftest 0x00. Vi sjekket termineringsmotstanden til Dual CAN-bus adapteren og den var i orden (figur 36). Vi testet også å kjøre programmet uten IMU'en tilkoblet, men vi fikk samme resultater bare uten gyroverdiene.

Sjekker termineringsmotstanden i Dual CAN-bus adapteren



Figur 36

Oppkobling av krets



Figur 37

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

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 meldingsID'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. 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 IMU'en, 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 Linuxdistribusjoner er blant operativsystemene som kan simuleres på disse maskinene [25].

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 [26].

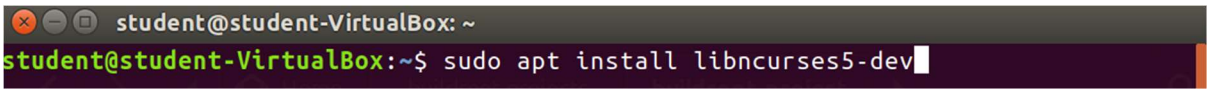
Buildroot

Buildroot er en gratis ”open source”-programvare for å enkelt bygge og konfigurere en komplett Linuxdistribusjon for innebygde datasystemer [27].

Som en del av prosjektet skulle vi konfigurere en egen Linuxdistribusjon som skulle fungere med en Raspberry Pi 3 og et PiCAN-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 (figur 38).

Installerer ncurses-biblioteket

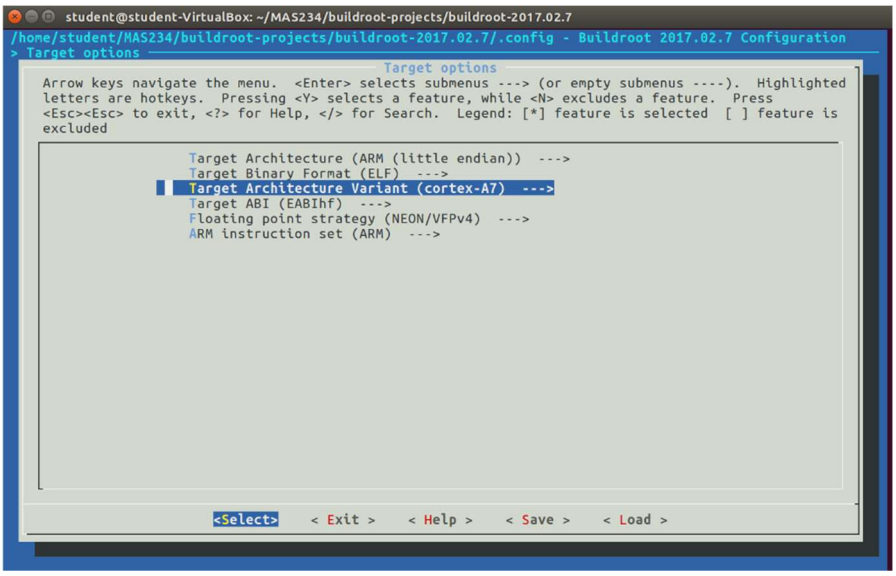


Figur 38

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 (figur 39). Her kunne vi blant annet lese at prosessoren vår var en Cortex-A7 basert på ARM-arkitektur [28].

Ser at vi har en Cortex-A7 prosessor



Figur 39

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 (figur 40).

Åpner konfigurasjonsmenyen

```
student@student-VirtualBox: ~/buildroot-projects/buildroot-project
student@student-VirtualBox:~/buildroot-projects/buildroot-project$ make BR2_EXTE
RNAL=${PWD}/external -C ${PWD}/buildroot O=${PWD}/buildroot-output/ menuconfig
```

Figur 40

For å installere can_pingpong-pakken, kopierte vi mas234-konfigurasjonsfilen inn i buildroot-output-mappen. Vi navnga filen til ”.config”.

Vi kontrollerte nå at can_pingpong var valgt i konfigurasjonsmenyen, og startet kompileringen med kommandoen vist i figur 41.

Starter kompileringen av konfigurasjonsfilen

```
student@student-VirtualBox: ~/buildroot-projects/buildroot-project
student@student-VirtualBox:~/buildroot-projects/buildroot-project$ make -C ${PWD
}/buildroot O=${PWD}/buildroot-output/
```

Figur 41

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 kommandoer.

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 42 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

```
student@student-VirtualBox:~$ sudo dd if=/home/student/buildroot-projects/buildr
oot-project/buildroot-output/images/sdcard.img of=/dev/sdb1
221255+0 records in
221255+0 records out
113282560 bytes (113 MB, 108 MiB) copied, 29,2542 s, 3,9 MB/s
student@student-VirtualBox:~$
```

Figur 42

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 [29].

PiCAN er et tilleggskort til Raspberryen som gir oss GPS, Gyro, Akselerometer og CAN-bus kommunikasjon [30].

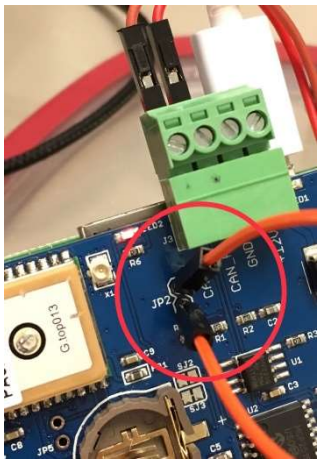
I dette prosjektet har vi brukt en Raspberry Pi 3 modell B, samt et PiCAN-tilleggskort koblet over det 40 pins koblingspunktet til Raspberryen.

I dokumentasjonen til Raspberryen fant vi spesifikasjoner på hvilken strømforsyning som kreves. Raspberryen må bli forsynt med 5.1V over microUSB [31]. 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 [32].

Når Raspberryen 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 koblingspunktet på Raspberryen.

Vi koblet oss på CAN_H og CAN_L på PiCAN-kortet gjennom Peak PCAN-USB FD 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 (figur 43), ved å koble en ledning mellom punktene på JP2 [30].

Terminering av CAN-bus



Figur 43

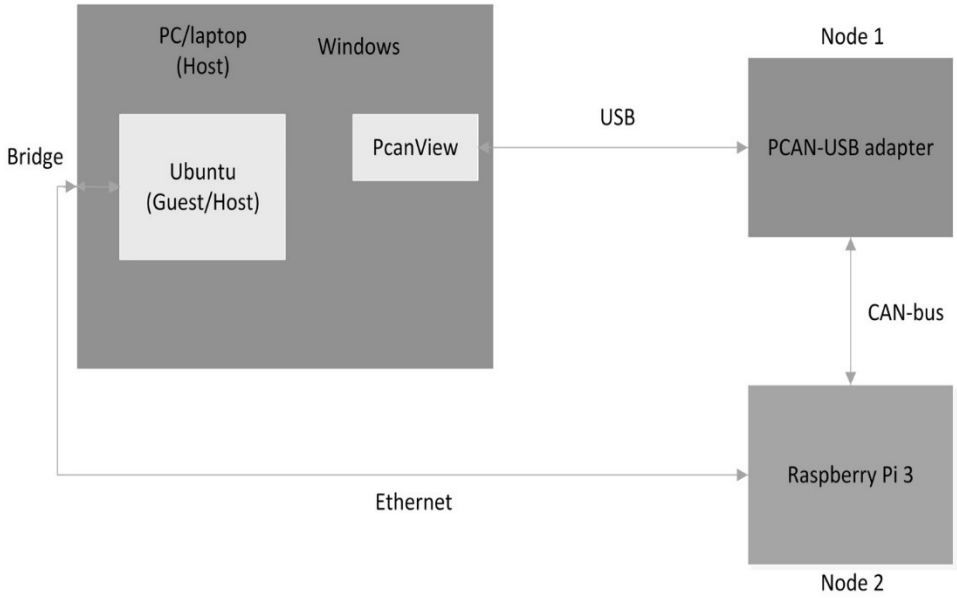
Konfigurasjon av ethernettilkobling

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 44 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



Figur 44

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 [33]. Med kommandoen ”ssh

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 [34]. Dette kalles 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 (figur 45).

Sjekker at Can0 er konfigurert

```
student@student-VirtualBox: ~
# modprobe spi_bcm2835
# modprobe mcp251x
# /sbin/ip link set can0 up type can bitrate 250000
# ifconfig
can0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
-00
          UP RUNNING NOARP  MTU:16  Metric:1
          RX packets:28 errors:0 dropped:28 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:168 (168.0 B)  TX bytes:0 (0.0 B)
```

Figur 45

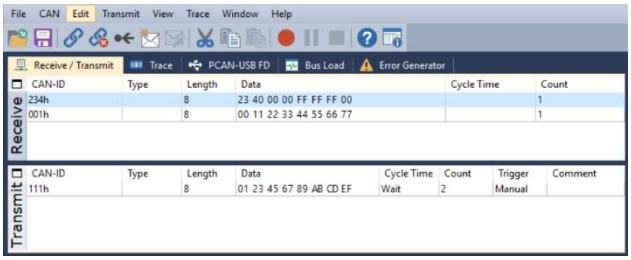
Kommandolinjen "candump can0" printer alle CAN-meldinger som sendes på nettverket. For å teste kommunikasjonen mellom PCAN-adapteren og Raspberryen, forsøkte vi å sende meldinger fra PcanView til terminalen i Raspberry. Figur 46 og figur 47 viser to meldinger med ID 0x111 og likt innhold.

Mottar og sender meldinger fra terminalen i Raspberry

```
# candump can0
  can0  111   [8]  01 23 45 67 89 AB CD EF
  can0  111   [8]  01 23 45 67 89 AB CD EF
^C# cansend can0 234#23.40.00.00.FF.FF.FF.00
# cansend can0 001#00.11.22.33.44.55.66.77
#
```

Figur 46

Sender og mottar meldinger i PcanView



Figur 47

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 IMU'en. Når vi sendte ut en CAN-bus melding med ID 0x234 fra terminalen i Raspberryen, mottok og behandlet Teensyen denne. Teensyen leste så av gyrodataen fra IMU'en og sendte denne dataen i en CAN-melding med ID 0x111.

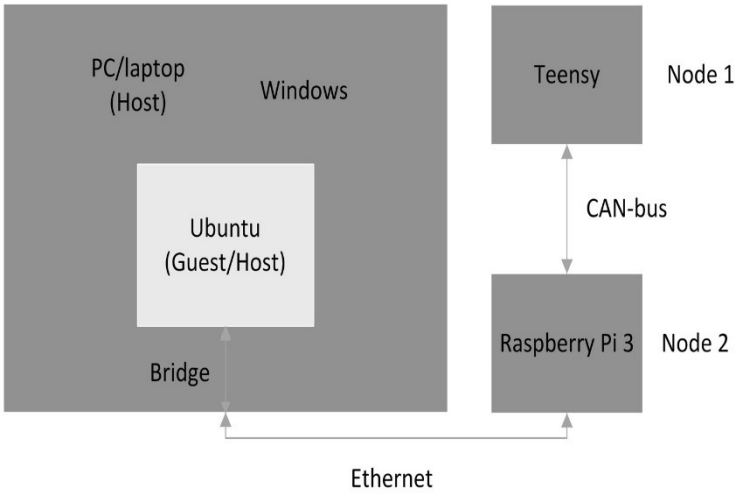
Figur 48 viser mottatt CAN-melding med ID 0x111 og 6 bytes, hvorav de to første representerer gyroverdi for X-retning, de to neste for Y-retning, og de to siste for Z-retning.

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

```
# candump can0
  can0  234   [1]  00
  can0  111   [6]  22 5E 0B 7E FF A2
  can0  234   [1]  00
  can0  111   [6]  22 5E 0B 4F FF 92
  can0  234   [1]  00
  can0  111   [6]  22 59 0B 67 FF 97
  can0  234   [1]  00
  can0  111   [6]  25 7E 0B 56 FE B9
  can0  234   [1]  00
  can0  111   [6]  21 CA 00 3D 04 83
  can0  234   [1]  00
  can0  111   [6]  21 45 0B 7A 00 1C

student@student-VirtualBox: ~
# cansend can0 234#00
# cansend can0 234#00
# cansend can0 234#00
#
```

Figur 48

Oppkobling av nettverk



Figur 49

Konklusjon

Vi løste oppgavene på en god måte, og håndterte utfordringene som dukket opp underveis. Ved å benytte Ubuntu og dens funksjoner, har vi laget og overført en Linuxdistribusjon som vi har benyttet på en Raspberry Pi 3. Vi har kommunisert via både ethernet, USB og CAN-bus i et nettverk med flere noder, og til dels brukt CAN-nettverket i et praktisk eksempel.

Diskusjon

MicroSD-kortet

Vi hadde problemer med å få Raspberryen til å starte opp med microSD-kortet og Linuxdistribusjonen vi lagde. Dette skyldes at vi skrev image-filen til en partisjon på microSD-kortet, fremfor hele kortet. Etter feilsøking sammen med faglærer, fikk vi hjelp til å skrive en ny image-fil til microSD-kortet, og Raspberryen startet opp korrekt med Linuxdistribusjonen.

Root

Det å koble seg på root-nivå til noe over internett kan være en risiko, da man aldri er helt trygg på nettet. Dette gjelder spesielt om man har et svakt passord, noe som i vårt tilfelle var sant. Root lar en gjøre absolutt alt av endringer, og hvis denne tilgangen kommer i feil hender kan utfallet bli katastrofalt dersom systemet er av størrelse.

Vi koblet oss allikevel til på root-nivå for å få full kontroll over Raspberryen. I vårt scenario var verken risikoen eller konsekvensene spesielt høye.

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] «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].

[4] 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].

[5] Z. W. Yan, Kingbright, 12 November 2001. [Internett]. Available: http://pdf1.alldatasheet.com/datasheet-pdf/view/167067/KINGBRIGHT/L7104GC.html. [Funnet 9 Oktober 2017].

[6] «www.wikipedia.org,» 12 Februar 2017. [Internett]. Available: https://en.wikipedia.org/wiki/Current_sources_and_sinks. [Funnet 16 Oktober 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] Elecrom, «www.elecrom.com,» 6 Februar 2017. [Internett]. Available: http://www.elecrom.com/avr-tutorial-2-avr-input-output/. [Funnet 20 Oktober 2017].

[10] M. Hämmerling, «www.engbedded.com,» 2014. [Internett]. Available: http://www.engbedded.com/fusecalc/. [Funnet 1 November 2017].

[11] «www.lutron.com,» Lutron, 27 April 2000. [Internett]. Available: http://www.lutron.com/TechnicalDocumentLibrary/Measured_vs_Perceived.pdf. [Funnet 30 Oktober 2017].

[12] «www.arduino.cc,» 29 August 2015. [Internett]. Available: https://www.arduino.cc/en/Tutorial/Debounce. [Funnet 24 Oktober 2017].

[13] «www.pjcr.com,» PJCR, [Internett]. Available: https://www.pjrc.com/teensy/teensyduino.html. [Funnet 30 Oktober 2017].

[14] «www.arduino.cc,» Arduino, [Internett]. Available: https://playground.arduino.cc/Main/MPU-6050. [Funnet 31 Oktober 2017].

[15] 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].

[16] Arduino, «www.arduino.cc,» Arduino, [Internett]. Available: https://www.arduino.cc/en/Reference/Wire. [Funnet 2 November 2017].

[17] c. m. p. t. t. F. P. b. teachop, 1 August 2017. [Internett]. Available: https://github.com/collin80/FlexCAN_Library/. [Funnet 30 Oktober 2017].

[18] 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].

[19] Texas Instruments, «www.ti.com,» Texas Instruments, Juli 2015. [Internett]. Available: http://www.ti.com/lit/ds/symlink/sn65hvd230.pdf. [Funnet 31 Oktober 2017].

[20] «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].

[21] 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].

[22] «www.pjcr.com,» Freescale Semiconductor, Inc, 2 Mai 2015. [Internett]. Available: https://www.pjrc.com/teensy/K66P144M180SF5RMV2.pdf. [Funnet 14 November 2017].

[23] T. O. Fredericks, 25 Juli 2014. [Internett]. Available: https://www.pjrc.com/teensy/td_libs_Metro.html. [Funnet 6 November 2017].

[24] «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].

[25] Oracle, «www.virtualbox.org,» 3 November 2017. [Internett]. Available: https://www.virtualbox.org/wiki/Downloads. [Funnet 4 September 2017].

[26] Canonical Ltd, «www.ubuntu.com,» Canonical Ltd, 2017. [Internett]. Available: https://www.ubuntu.com/download/desktop. [Funnet 4 September 2017].

[27] «www.wikipedia.org,» 27 Juli 2017. [Internett]. Available: https://en.wikipedia.org/wiki/Buildroot. [Funnet 6 November 2017].

[28] «www.wikipedia.org,» 24 Juni 2017. [Internett]. Available: https://en.wikipedia.org/wiki/ARM_Cortex-A7. [Funnet 6 November 2017].

[29] «www.wikipedia.org,» 15 November 2017. [Internett]. Available: https://en.wikipedia.org/wiki/Raspberry_Pi. [Funnet 13 November 2017].

[30] 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].

[31] «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].

[32] Apple Inc, «www.support.apple.com,» Apple Inc, 27 Februar 2017. [Internett]. Available: https://support.apple.com/en-us/HT204377. [Funnet 13 November 2017].

[33] «www.ubuntu.com,» 27 Februar 2015. [Internett]. Available: https://help.ubuntu.com/community/SSH. [Funnet 13 November 2017].

[34] «www.datamaskin.biz,» [Internett]. Available: http://www.datamaskin.biz/Systems/basic-computer-skills/200343.html#.WgtnxLaDrOQ. [Funnet 13 November 2017].

[35] pighixxx, «www.arduino.cc,» 14 September 2016. [Internett]. Available: https://forum.arduino.cc/index.php?topic=147582.0. [Funnet 2 November 2017].

[36] «www.pjrc.com,» [Internett]. Available: https://www.pjrc.com/teensy/pinout.html. [Funnet 6 November 2017].