11.10.2017

Bård Barstad, Halvard Yri Adriaenssens og Fredrik Kartevoll

Prosjekt Del 1

Introduksjon til mikrokontrollerprogrammering

Innhold

[Introduksjon 2](#_Toc497212121)

[Autodesk Eagle 3](#_Toc497212122)

[Spenningsregulator 3](#_Toc497212123)

[Funksjonstest av spenningsregulator 4](#_Toc497212124)

[Atmega168 4](#_Toc497212125)

[4](#_Toc497212126)

[Reset-funksjon 5](#_Toc497212127)

[LED-diode 6](#_Toc497212128)

[Konklusjon 7](#_Toc497212129)

[Kilder 8](#_Toc497212130)

# 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.

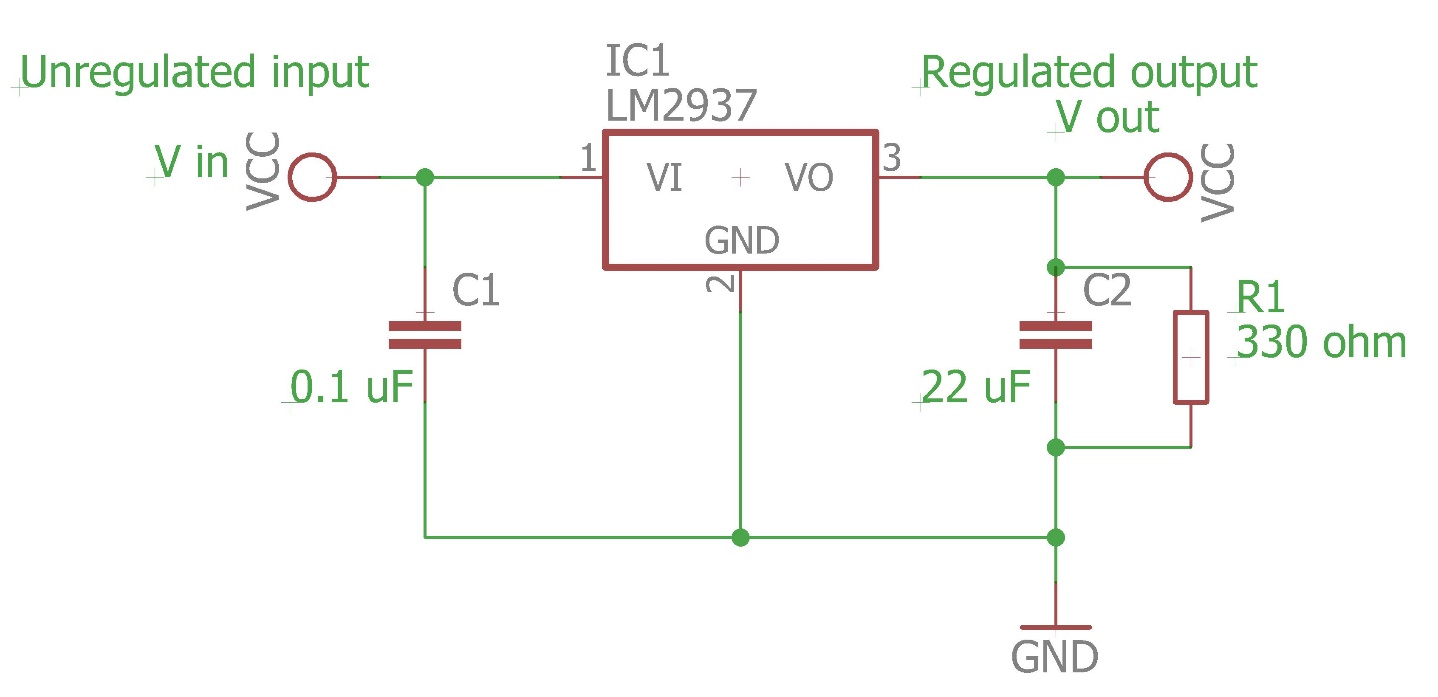
# Spenningsregulator

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.

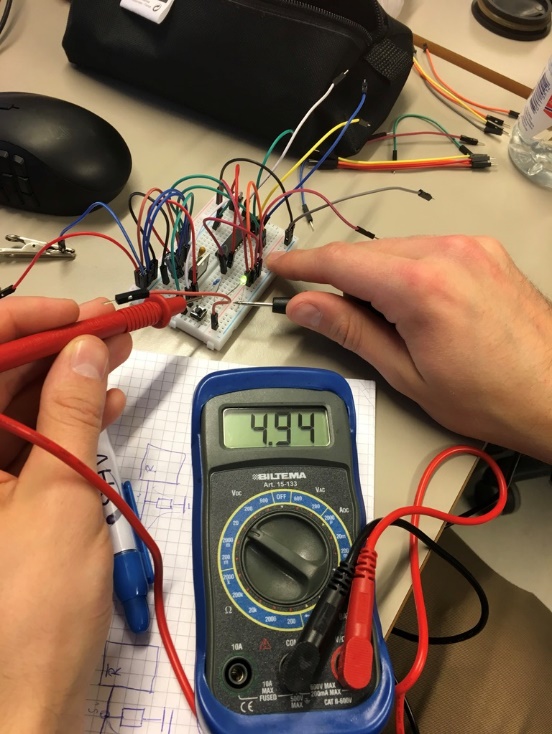
Til prosjektet vårt brukte vi en 5V lineær regulator, LM2931T. I følge databladet krever denne kondensatorer som ligger nærme regulatoren, mellom jord og input/output. Størrelsene på kondensatorene var oppgitt til å være 0,1uF og 100uF. Fordi vi ikke hadde 100uF tilgjengelig under prosjektet, benyttet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



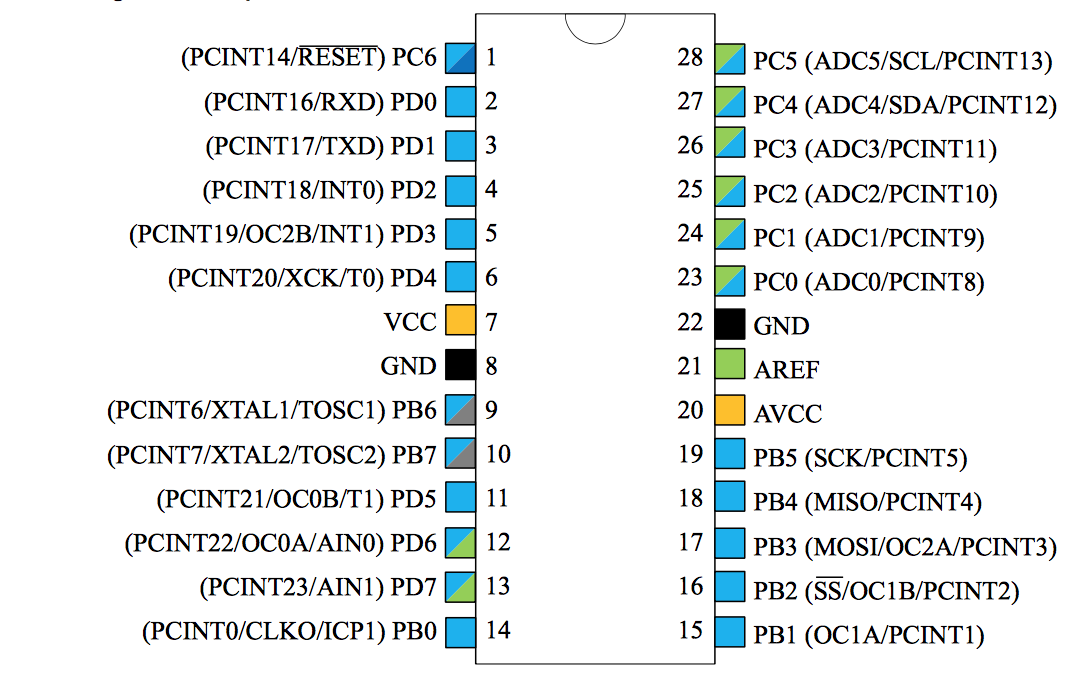
## Funksjonstest av spenningsregulator

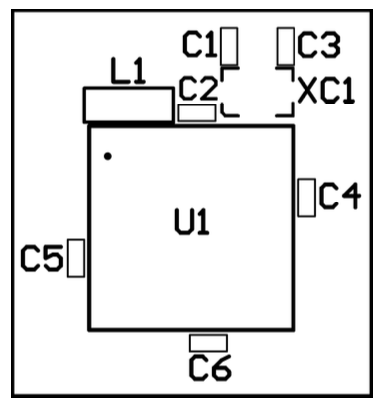
For å sikre oss at spenningsregulatoren leverte stabil spenning, ønsket vi å funksjonsteste den med et multimeter. I databladet står det beskrevet at spenningen inn på regulatoren må være 6V-26V. Vi brukte en lab strømforsyning for å simmulere en ustabil spenning inn ved å regulere strømforsyningen kontinuerlig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

# 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.



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.

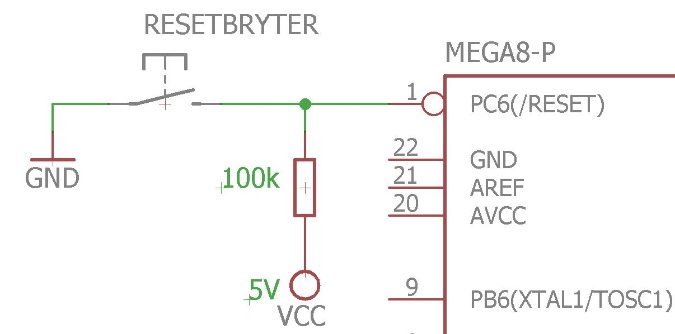
For å hindre dette, anbefales det i databladet å koble inn kondensatorer mellom Vcc og jord. Det spesifiseres at disse må kobles så fysisk nærme som mulig, slik at loopen vår med høyt strømtrekk blir så liten som mulig. Anbefalt størrelse på kondensatorene er 0,1uF.

## 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.

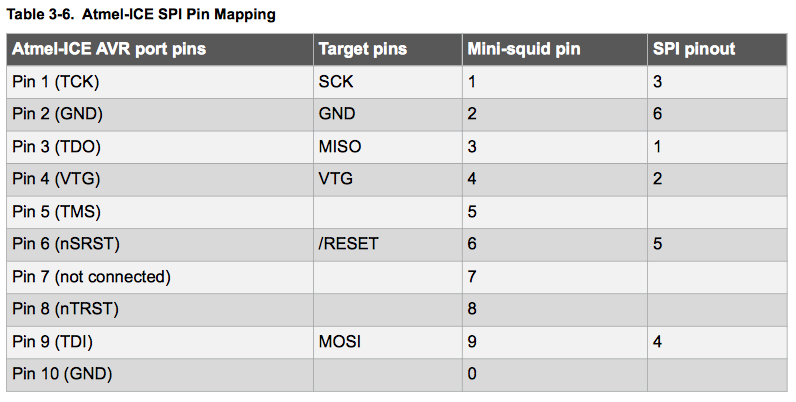
For å sikre oss mot uventede resetter, koblet vi opp en ekstern ”pull-up” reset. Denne står med konstant 5V inn på PC6, og vil gå lav dersom resetbryteren blir trykket. Ved å koble inn en motstand på 100k, så:

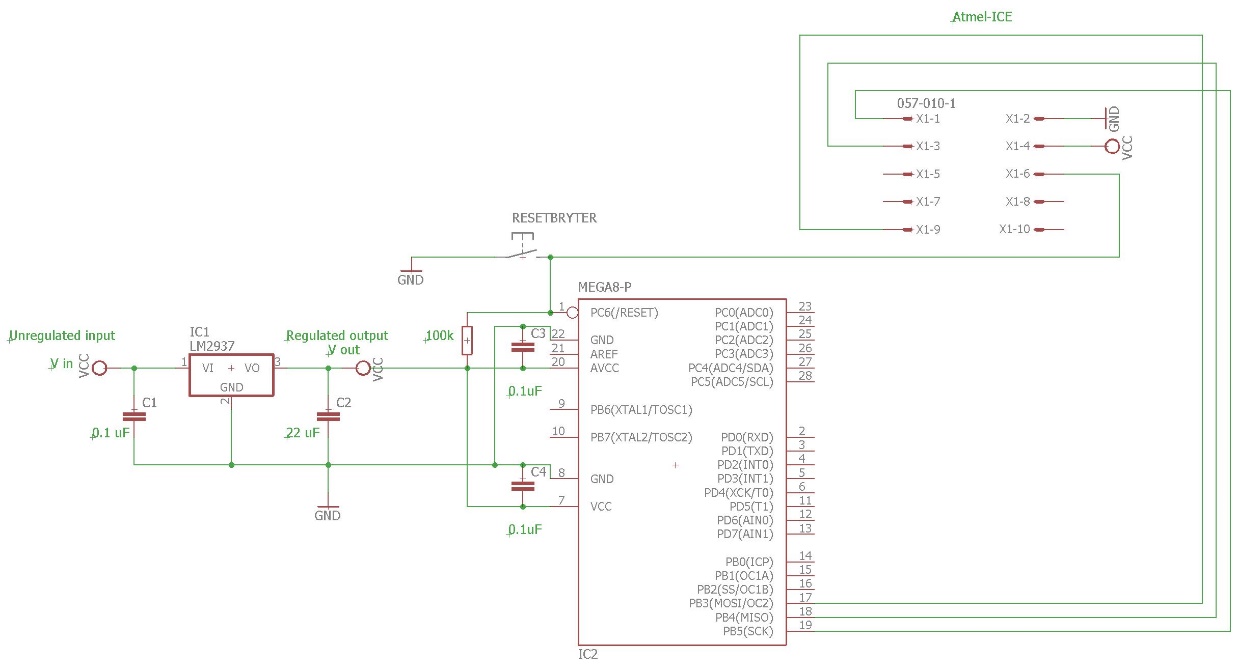
* Hindrer vi sporadiske resetter.
* Hindrer vi kortslutning mellom Vcc og jord når bryteren blir trykt.
* Reduserer vi strøm- og spenningspeaks fra kondensatorene når bryteren blir trykt.

# Atmel-ICE

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

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

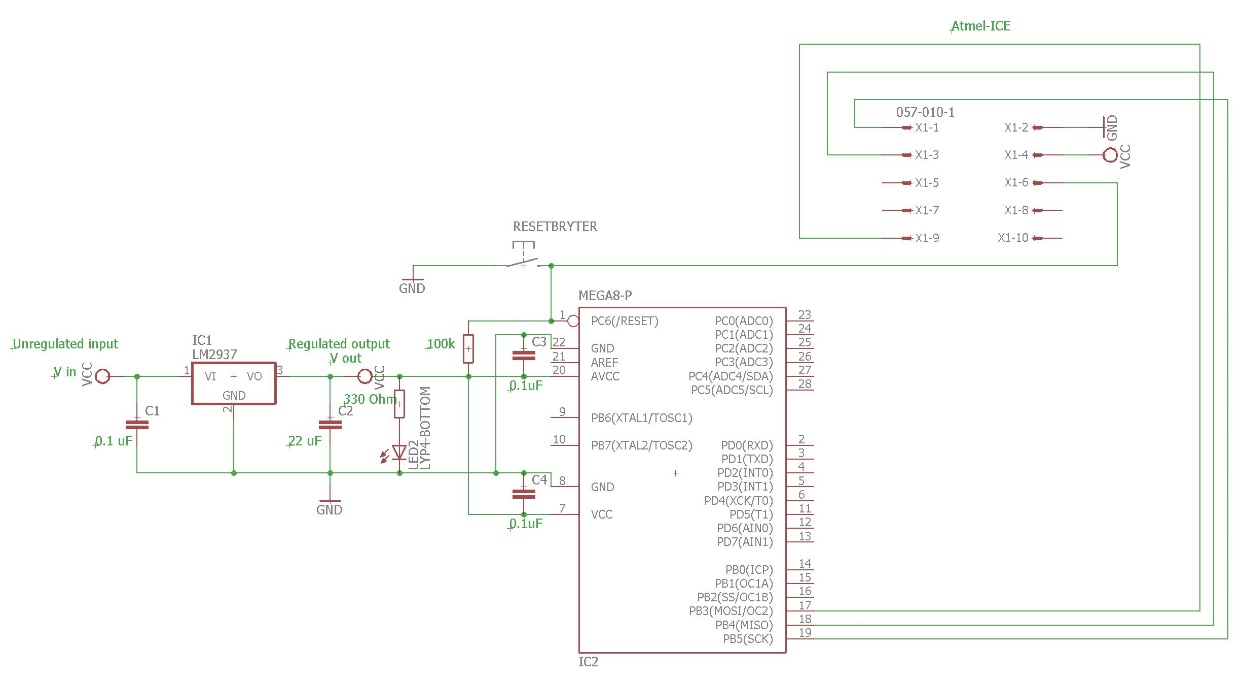




# LED-diode

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 LEDens datablad leste vi av maksimal driftsstrøm til å være 25mA og driftsspenning til å være 5VDC. For å begrense strømgjennomgangen i dioden, koblet vi en 330 i serie med denne.

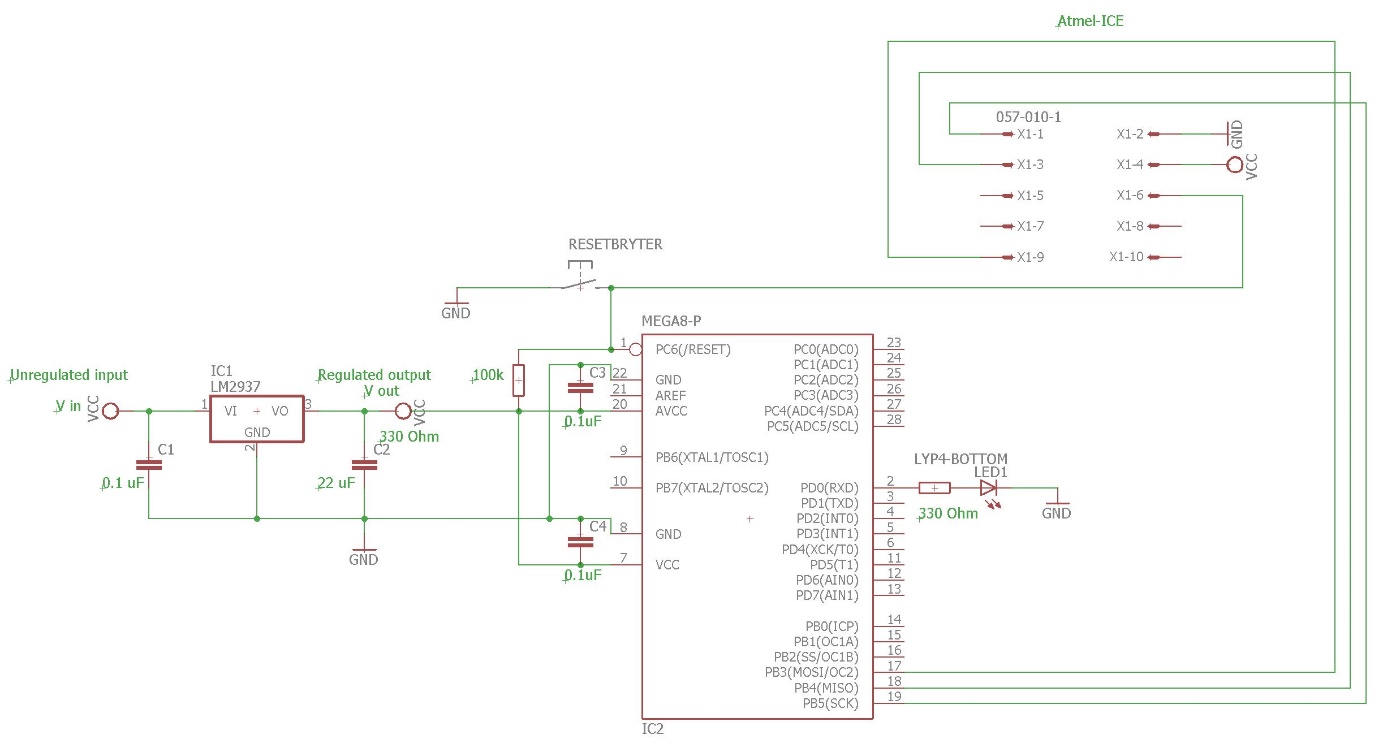
Vi koblet LEDen mellom utgangen på spenningsregulatoren og jord. Denne skulle lyse for å indikere at vi hadde ”power on”.



# Styre LED ved bruk av Atmega168

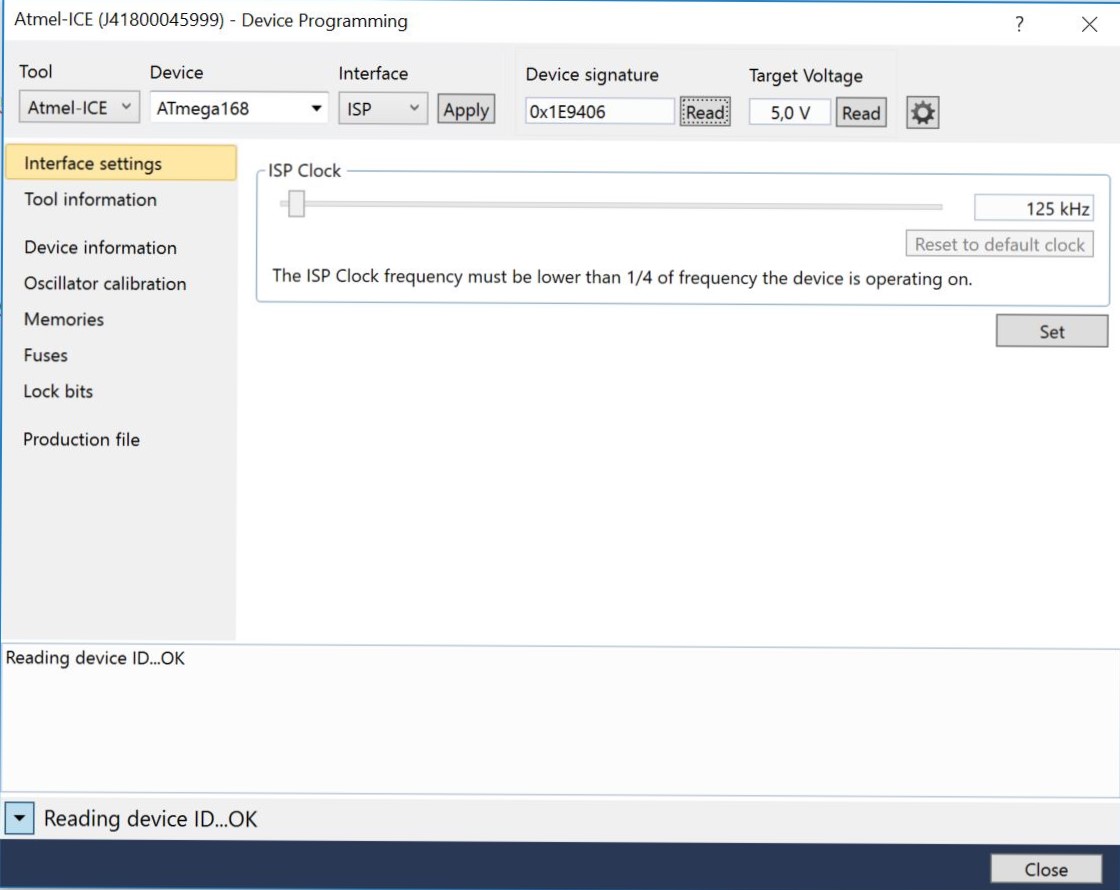
Man kan programmere en mikrokontroller for å styre f.eks en lysdiode. Før man kan lage et program for å styre lysdioden, må den kobles.

## Oppkobling

Vi koblet den 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 LED-en kun lyse når mikrokontrolleren gir ut 5V fra PD0, slik at vi får en fullstendig krets til jord. 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. 

## 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. Vi fant signature bytes til Atmega168 i databladet til å være 0x1E9406, og dette stemte overens med vår avleste signature fra AtmelStudio.

### 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.

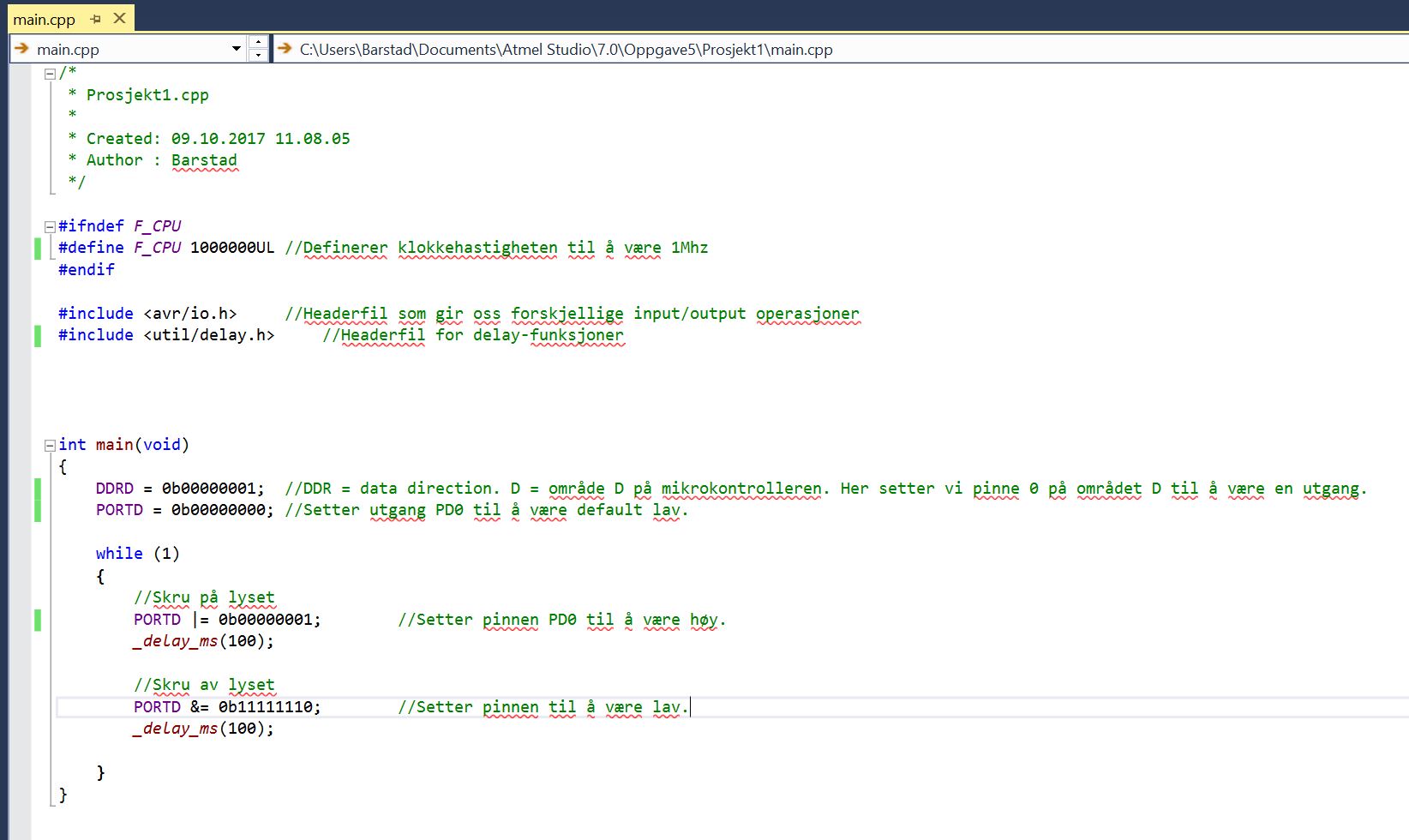
Slik konfigurerte vi fuses:

* 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.

Gå til memory og utfør programmering. Beskriv hva som skjer. Hvilken filtype leses av programmereren, og hvor havner innholdet i denne? 7. Lag en enkel skjematisk oversikt over programmeringsoppsettet (ikke Eagle, men "boksologi"). Beskriv hva som er target og hva som er host. 8. Hvorfor sier vi at det utføres krysskompilering når vi kompilerer i Atmel Studio for AVR-mikrokontrollere?

## Programmering i AtmelStudio

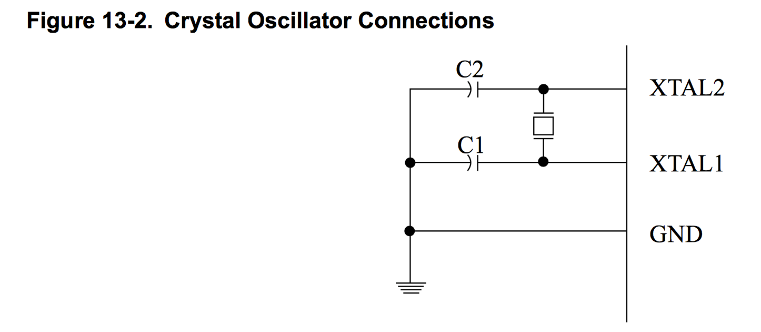
Før main-programmet definerte vi klokkesyklusen vår til å være 1 MHz, slik den ble satt i ”fuses”. Vi inkluderte også en headerfil for I/O, samt en headerfil for delay-funksjoner.

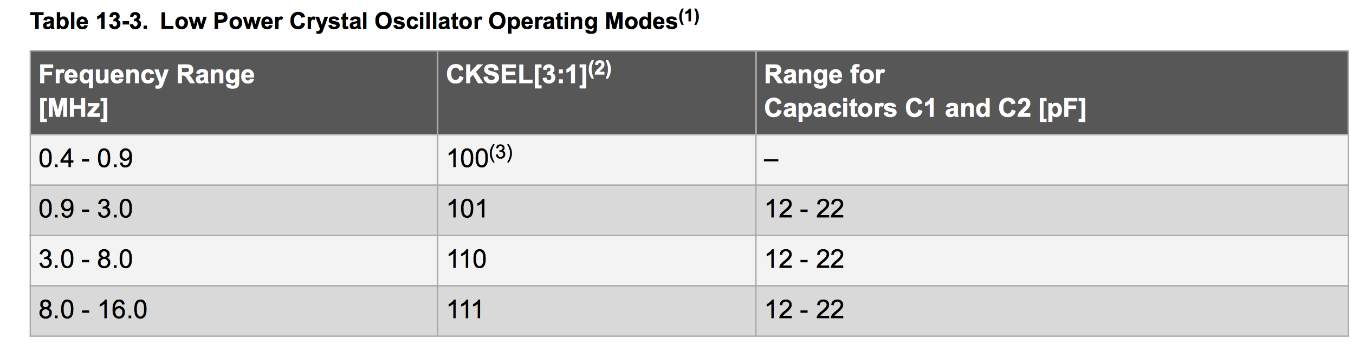
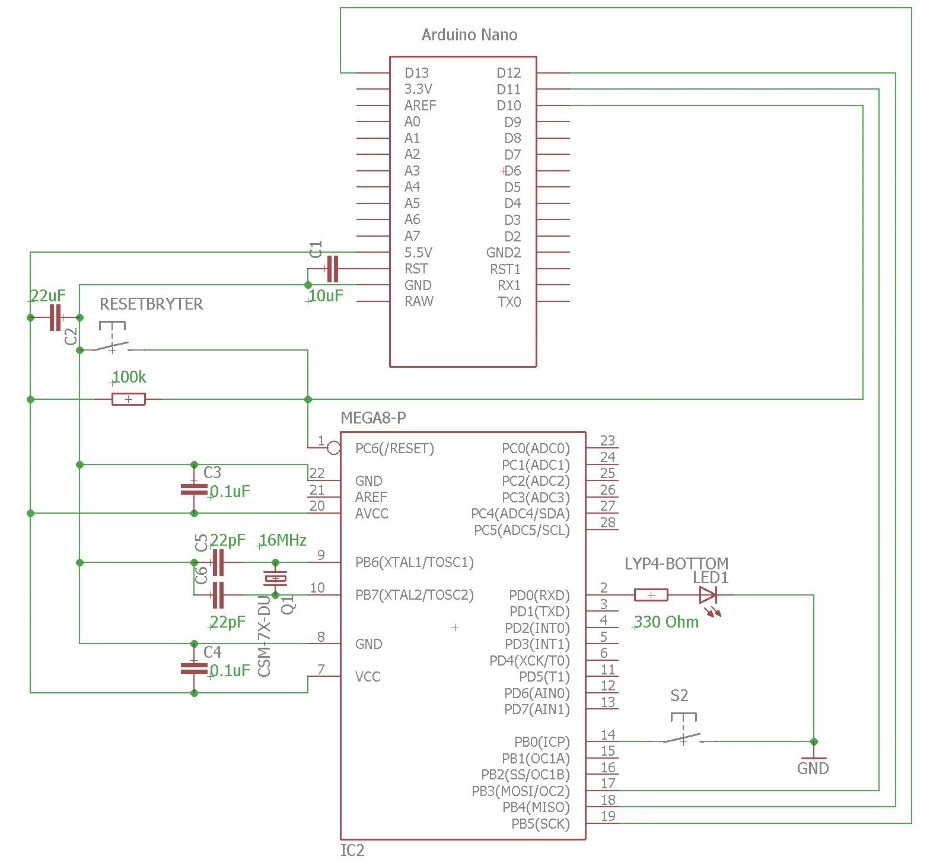
Vi definerte PD0 til å være en utgang ved å sette bit 1 i DDRD høy. Inni while-løkken vår programmerte vi PD0 til å gå høy/lav med en syklus på 200ms.

# 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 vfri 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.



# 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 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)

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

Når registre 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-bit fast PWM med clear OC0A ved compare match. Denne modusen vil kjøre en teller, som teller fra 0 til 255. Clear on compare match betyr: utgangen knyttet til OC0A (PD6) starter høy ved teller-verdi 0, og blir satt lav ved dutycycle\*255. 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. Dette er større enn maksimal konstant strøm på 25mA.

Eksempel på dårlig Hz (frekvensen er så lav at man ser LED-en flimrer)

Frekvensen vi valgte:

# Konklusjon

# Kilder

LM2931T datasheet: <https://www.jameco.com/Jameco/Products/ProdDS/836895.pdf>

Diode datasheet: <http://pdf1.alldatasheet.com/datasheet-pdf/view/167067/KINGBRIGHT/L7104GC.html>

Atmel ICE datasheet: <http://www.atmel.com/Images/Atmel-42330-Atmel-ICE_UserGuide.pdf>

Atmega168 datasheet: <http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2545-8-bit-AVR-Microcontroller-ATmega48-88-168_Datasheet.pdf>

AVR tutorial: <http://www.elecrom.com/avr-tutorial-2-avr-input-output/>

Source and sink: <https://en.wikipedia.org/wiki/Current_sources_and_sinks>

Fusecalc: http://www.engbedded.com/fusecalc/