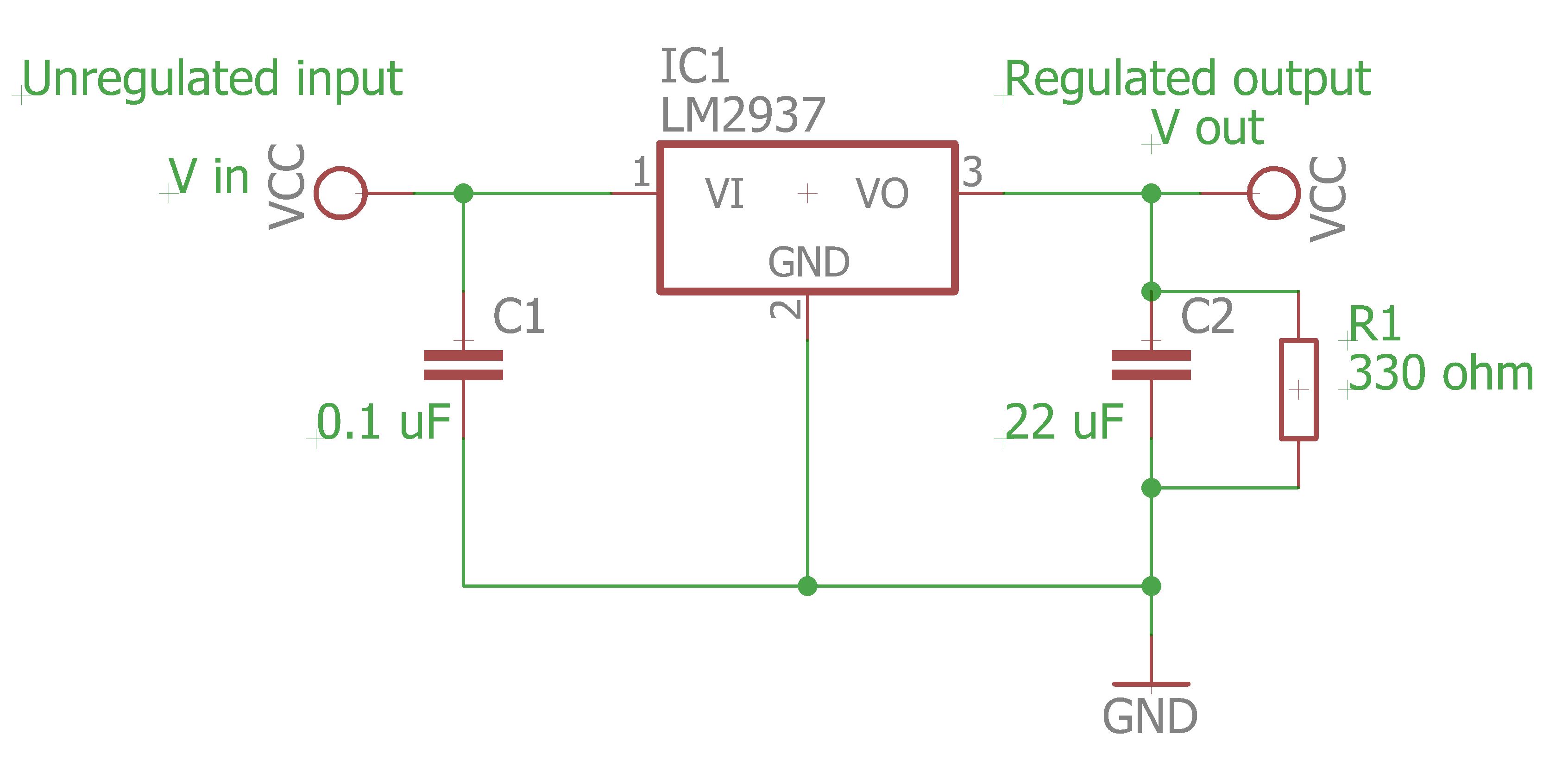
Intro

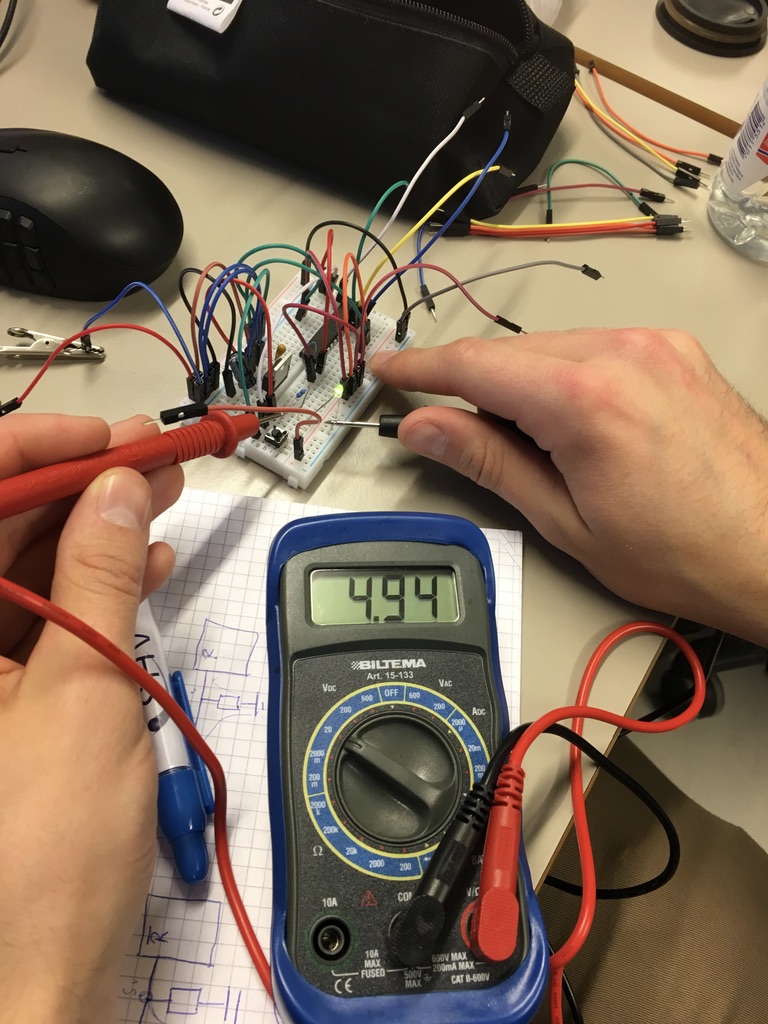
* Skrive til slutt. Hva har vi gjort i dette prosjektet?

**Spenningsregulator**

Vi startet prosjektet med å laste ned to programmer, henholdsvis Autodesk Eagle og Atmel Studio, for tegning av kretser og programmering av mikrokontrolleren. De første timene gikk ut på å sette oss inn i hvordan Eagle fungerer og de forskjellige funksjonene vi måtte bruke for å kunne tegne opp kretsen vår.

Vår første utfordring var å lage en strømforsyning som vi skulle bruke videre i prosjektet. Dette gjorde vi for å kunne forsyne resten av kretskortet våres med en helt stabil spenning. Vi valgte oss en 5V lineær spenningsregulator, LM2931T, som utgangspunkt og fant videre forklaring på oppkobling i databladet til denne.

For å kontrollere at denne fungerte optimalt, ønsket vi å måle spenningen ut med et multimeter. Det står da i databladet til spenningsregulatoren at den ikke vil levere en stabil spenning uten en minimumslast på 5mA, samtidig som den maks tåler en last på 100mA. Vi sjekket derfor etter tilgjengelige resistanser som vi kunne bruke som last, og regnet ut at dette gav en last som var: 5mA < X < 100mA.

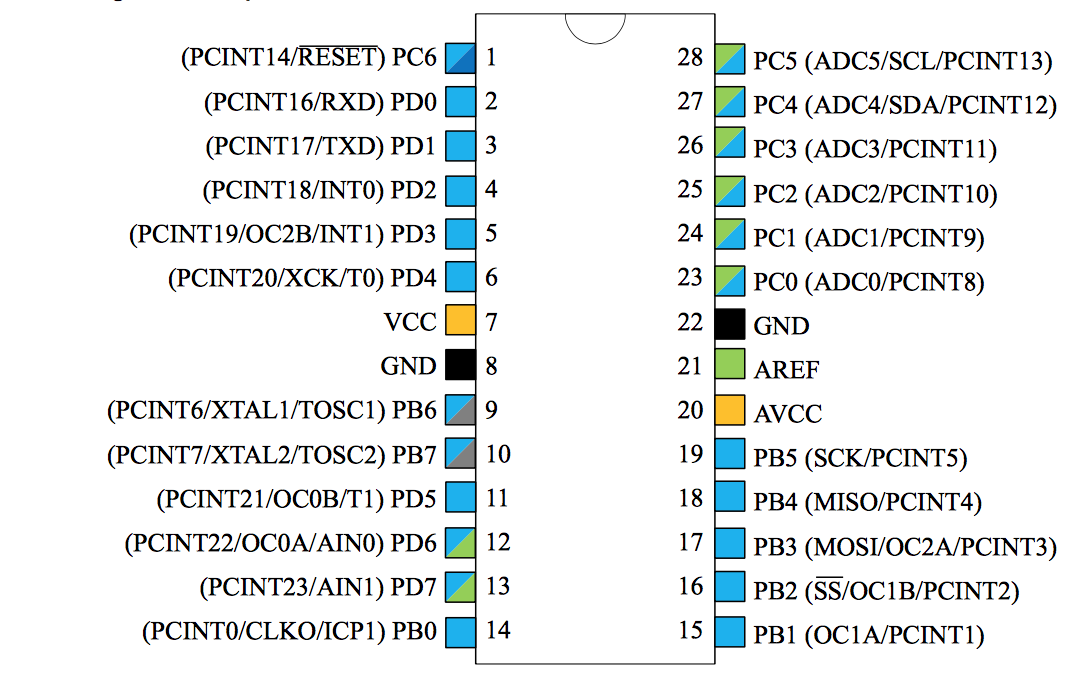
Databladet viste også at spenningen inn på regulatoren måtte være 6V ≤ X ≤ 26V. For å simulere enn ustabil spenning inn på regulatoren, regulerte vi spenningen fra 6-15V med strømforsyningen og målte med multimeteret at vi fikk stabilt 4,94V hele veien.

Vi tror at grunnen til at vi ikke får nøyaktig 5V ut av regulatoren, er fordi vi ble nødt til å erstatte 100uF-kondensatoren fra databladet med 22uF som var tilgjengelig.

**Tegning og oppkobling av mikrokontrolleren**

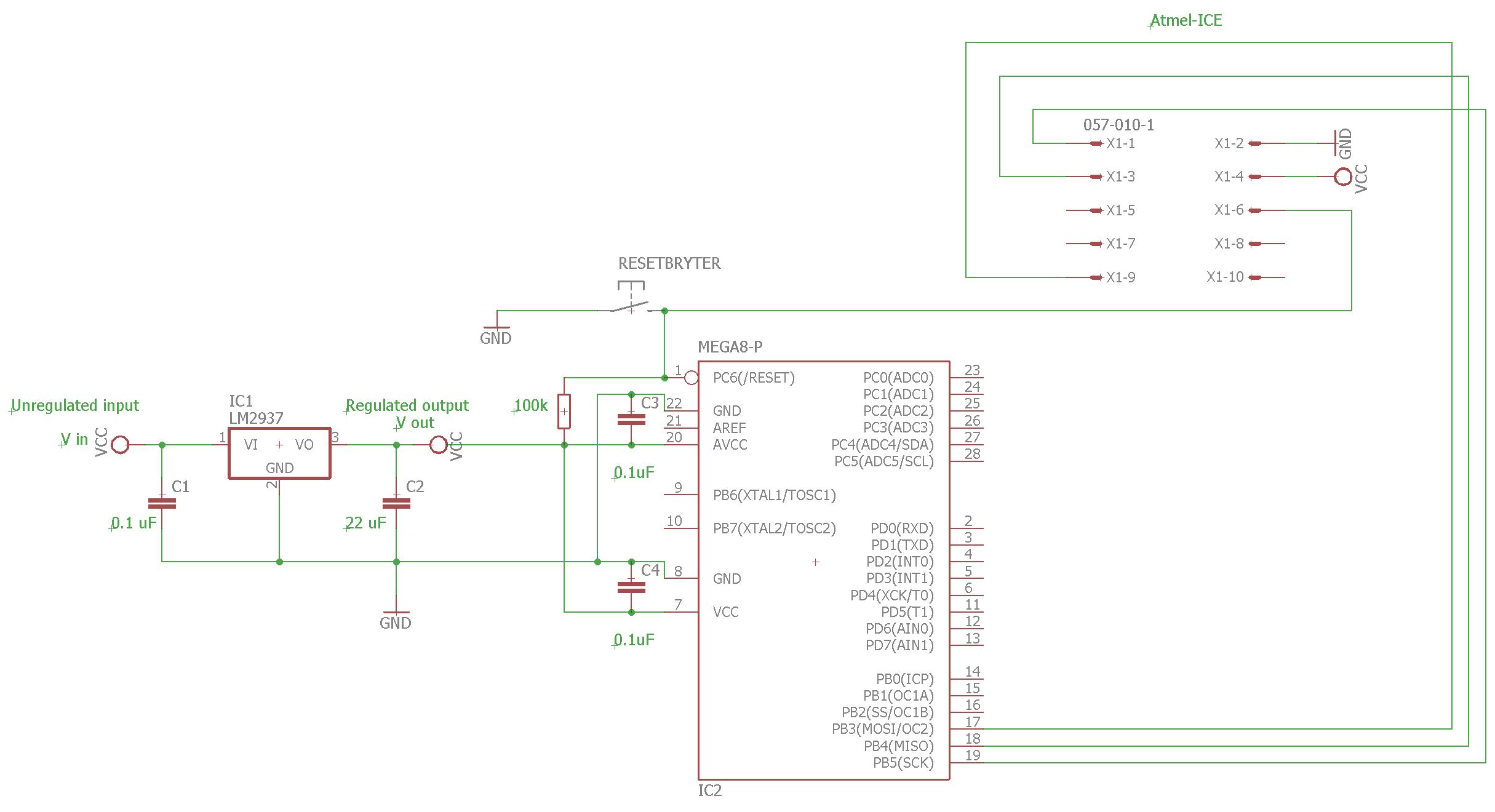
Etter å ha kontrollert at spenningsregulatoren fungerte tilfredsstillende, skulle vi tegne og koble opp mikrokontrolleren vår; en Atmega 168.

Vi søkte opp databladet og fant en oversikt over pin-konfigurasjonen som vi brukte som utgangspunkt for tegningen.

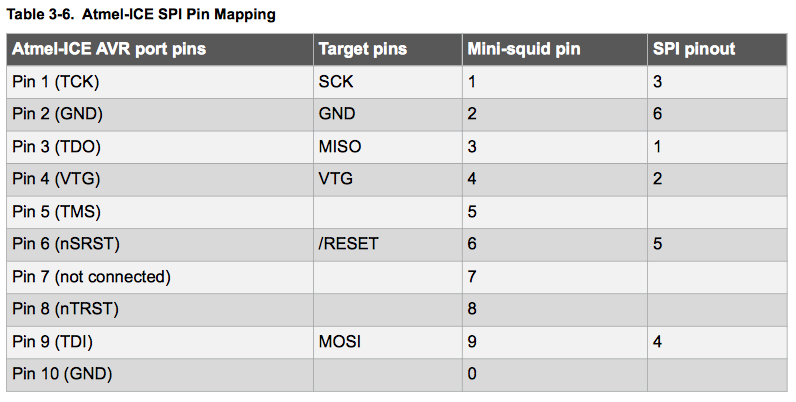


For å kunne resette mikrokontrolleren, valgte vi å lage en ”pull-up” reset. Denne ligger med konstant 5V inn på en invertert reset-pin. Mikrokontrolleren vil altså resette seg først når spenningen går til 0V. Dette skjer da resetbryteren blir trykket og lager en kobling til jord. En motstand på 100k ble brukt mellom Vcc og reset for å hindre støy, samt redusere størrelsen på strømmen gjennom bryteren dersom denne ble trykt.

Vi koblet inn kondensatorer mellom VCC/AVCC og ground så tett som mulig på mikrokontrolleren, for å fjerne uønsket støy fra resten av kretsen. Anbefalt størrelse og plassering fant vi i databladet.

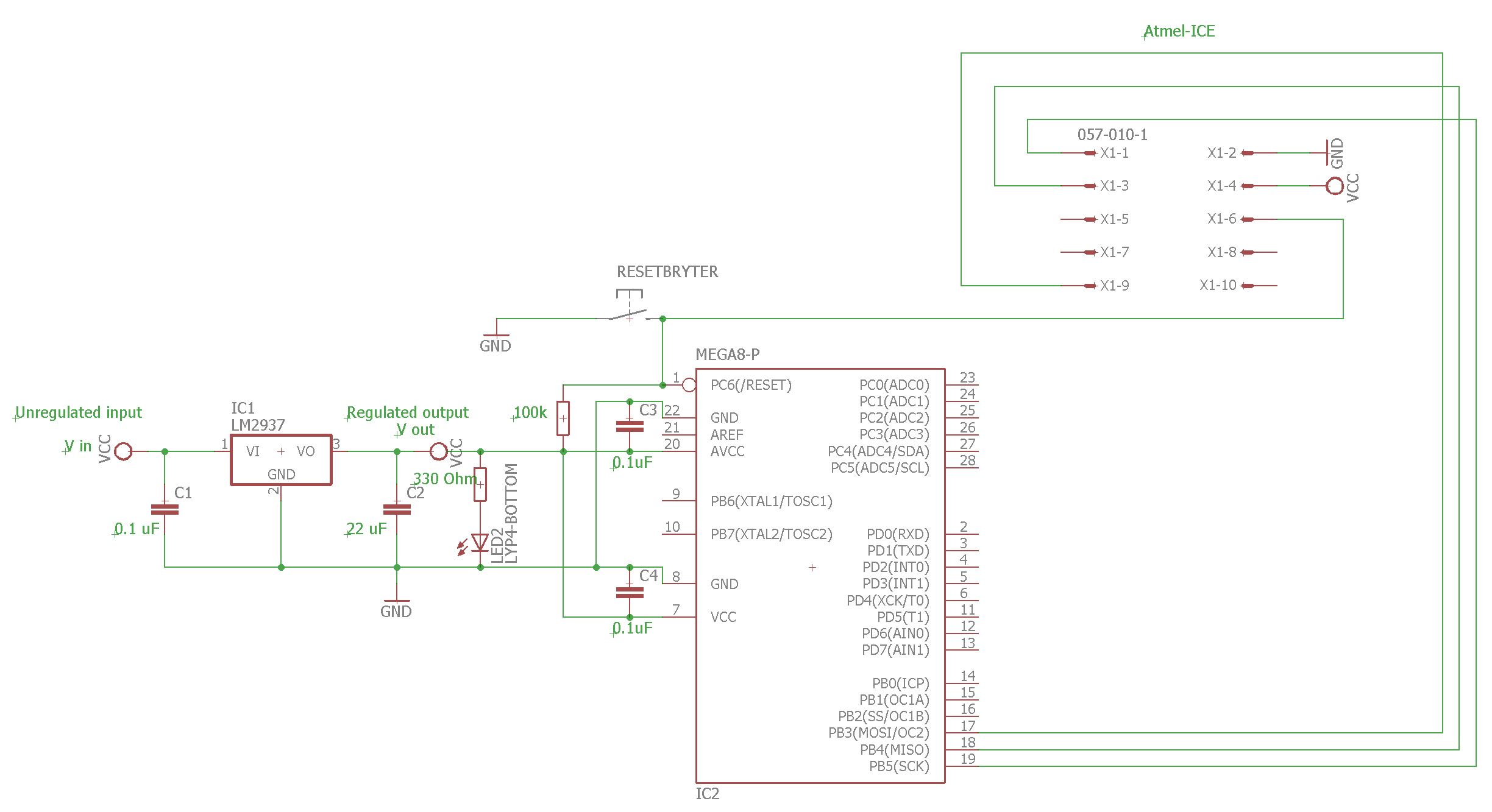


Videre tegnet vi inn oppkoblingen for en Atmel-ICE. Denne ble som en overgang mellom mikrokontrolleren og pcen, slik at vi kunne programmere Atmega-en våres. Tabell for oppkobling fant vi under avsnittet for SPI-target, i databladet til ICE-en. SCK, MISO og MOSI er alle pinner under PB på mikrokontrolleren, mens ground og VTG er henholdsvis jord og spenningsforsyning.

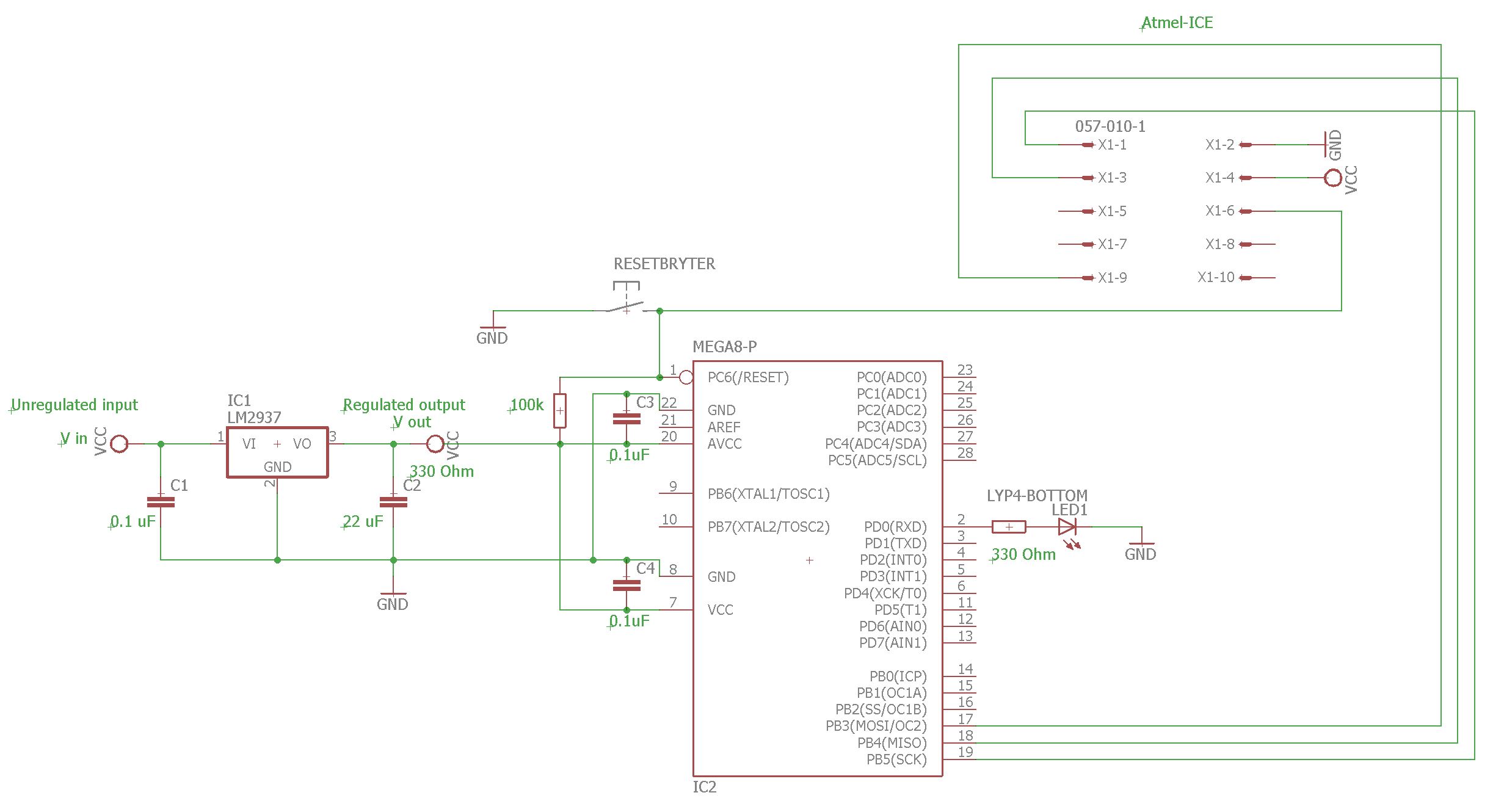


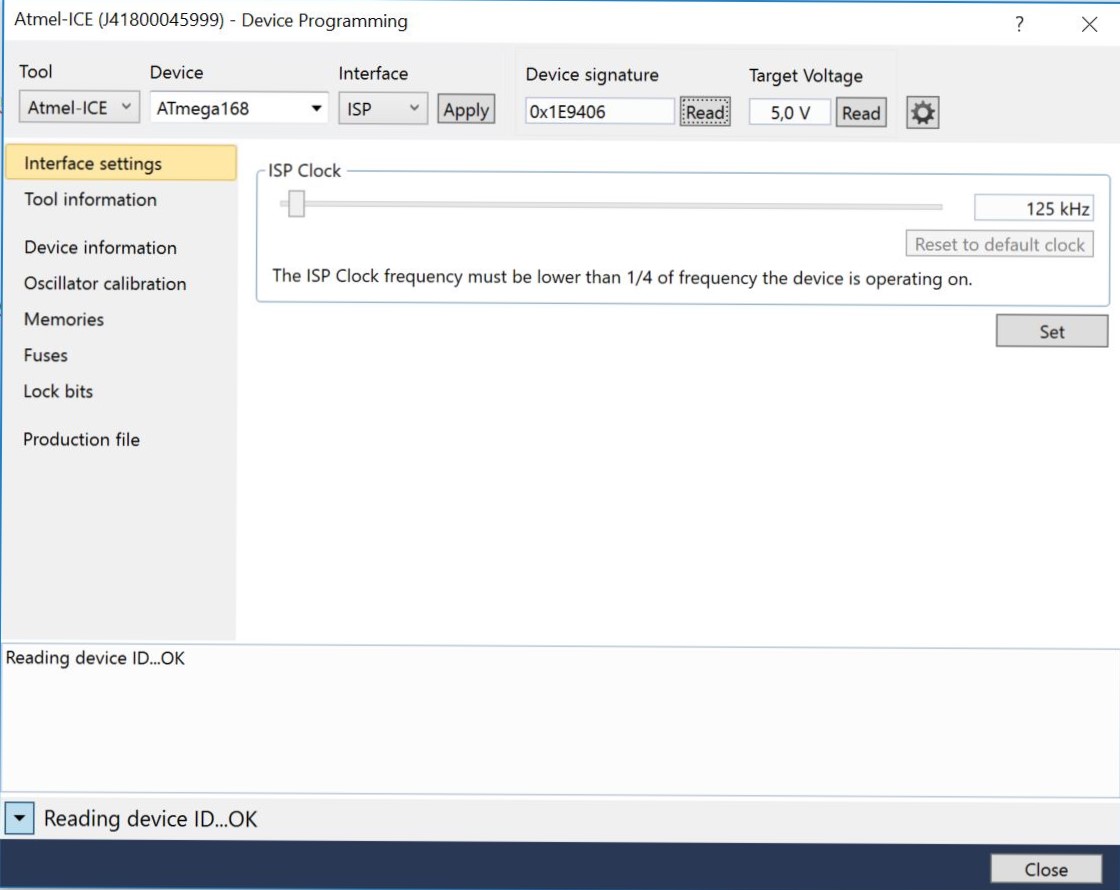
**AtmelStudio og LED-diode**

I neste del av oppgaven koblet vi opp en led mellom utgangen på spenningsregulatoren og jord. Denne skulle lyse for å indikere at vi hadde ”power on”. Vi leste oss opp i databladet til led-en vi brukte, og fant ut at den tålte en maksimal strøm på 25mA. For å sikre at det ikke gikk mer strøm igjennom dioden, koblet vi inn en motstand på 330 i serie med denne.



Etter å ha funksjonstestet LED-en, skulle vi koble den om til en utgang på mikrokontrolleren. 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 gikk for 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.



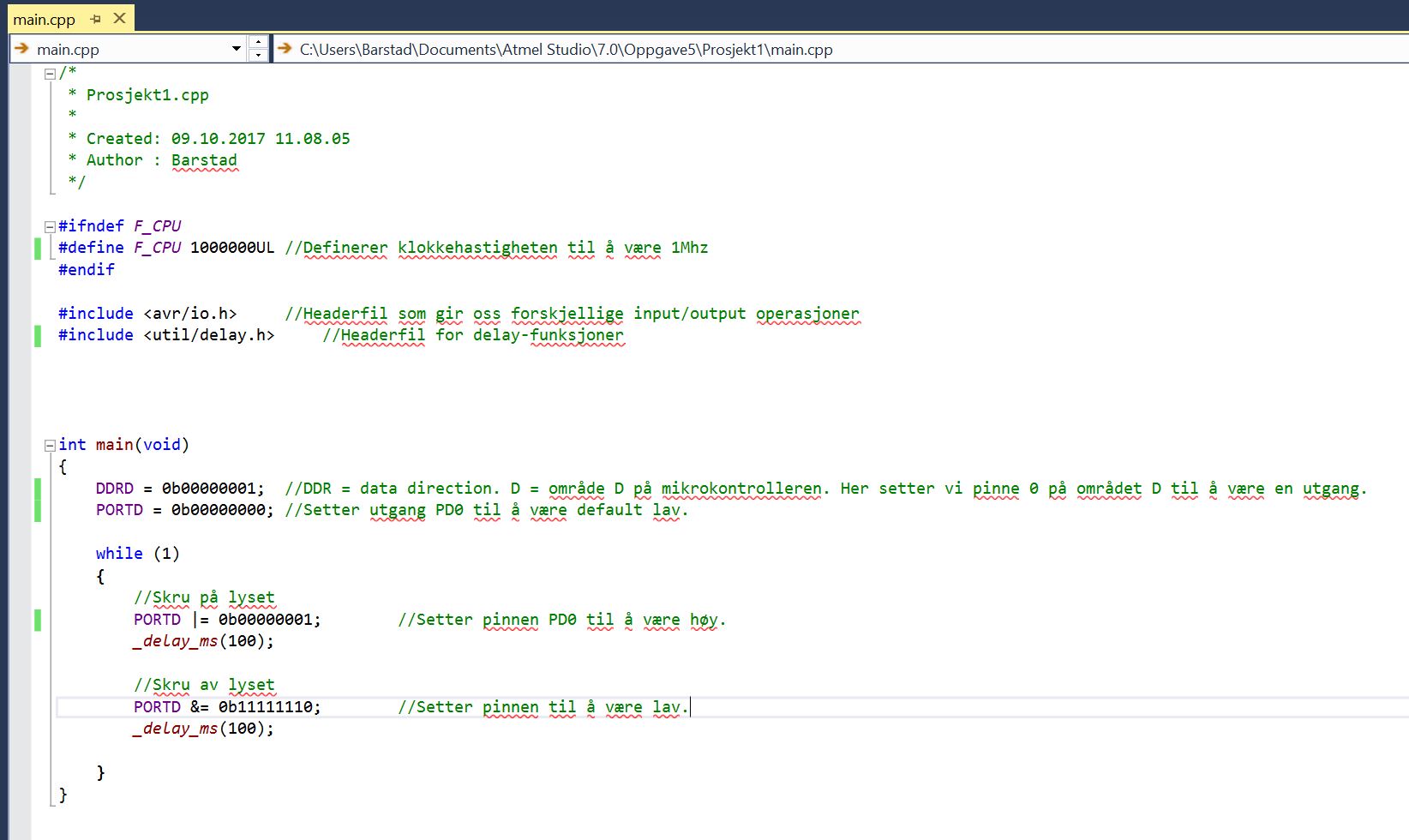
Når kretsen var ferdig koblet opp, starte vi AtmelStudio for første gang og gjorde oss kjent med programmet. Vi gikk inn på ”Device Programming” og leste av target volt til å være 5V. Vi fant signature bytes til Atmega168 i databladet til å være 0x1E9406, og dette stemte overens med vår avleste signature fra AtmelStudio.

Videre gikk vi inn på ”Fuses”. Fuses er en form for hovedinnstillinger som lagres og opprettholdes på mikrokontrolleren, selv om man kutter strømmen. Vi sjekket:

* 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.

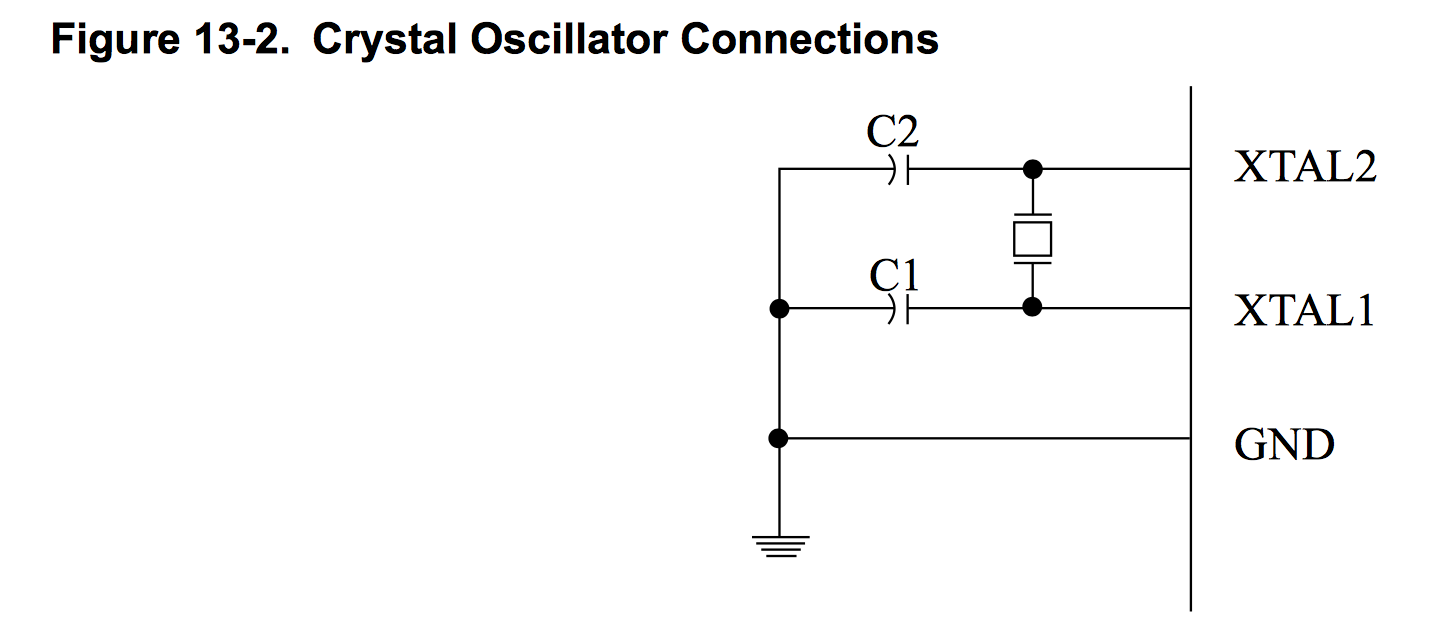
Da oppsettet med ”fuses” var ferdig, var vi klare til å programmere kontrolleren. Oppgaven gikk ut på å få en LED til å blinke kontinuerlig.

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 500ms.

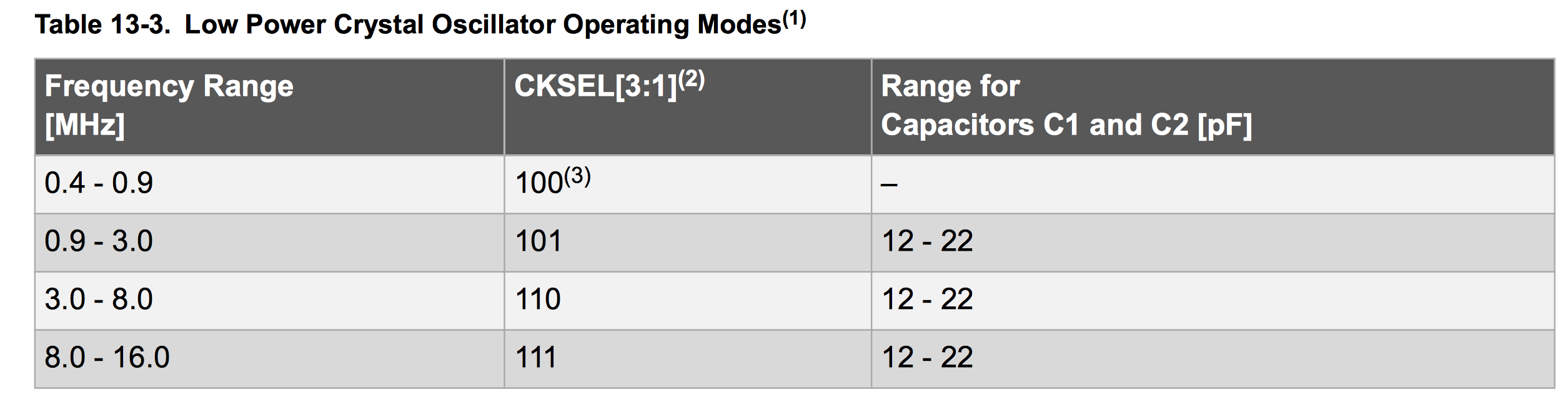
**Arduino og PWM**

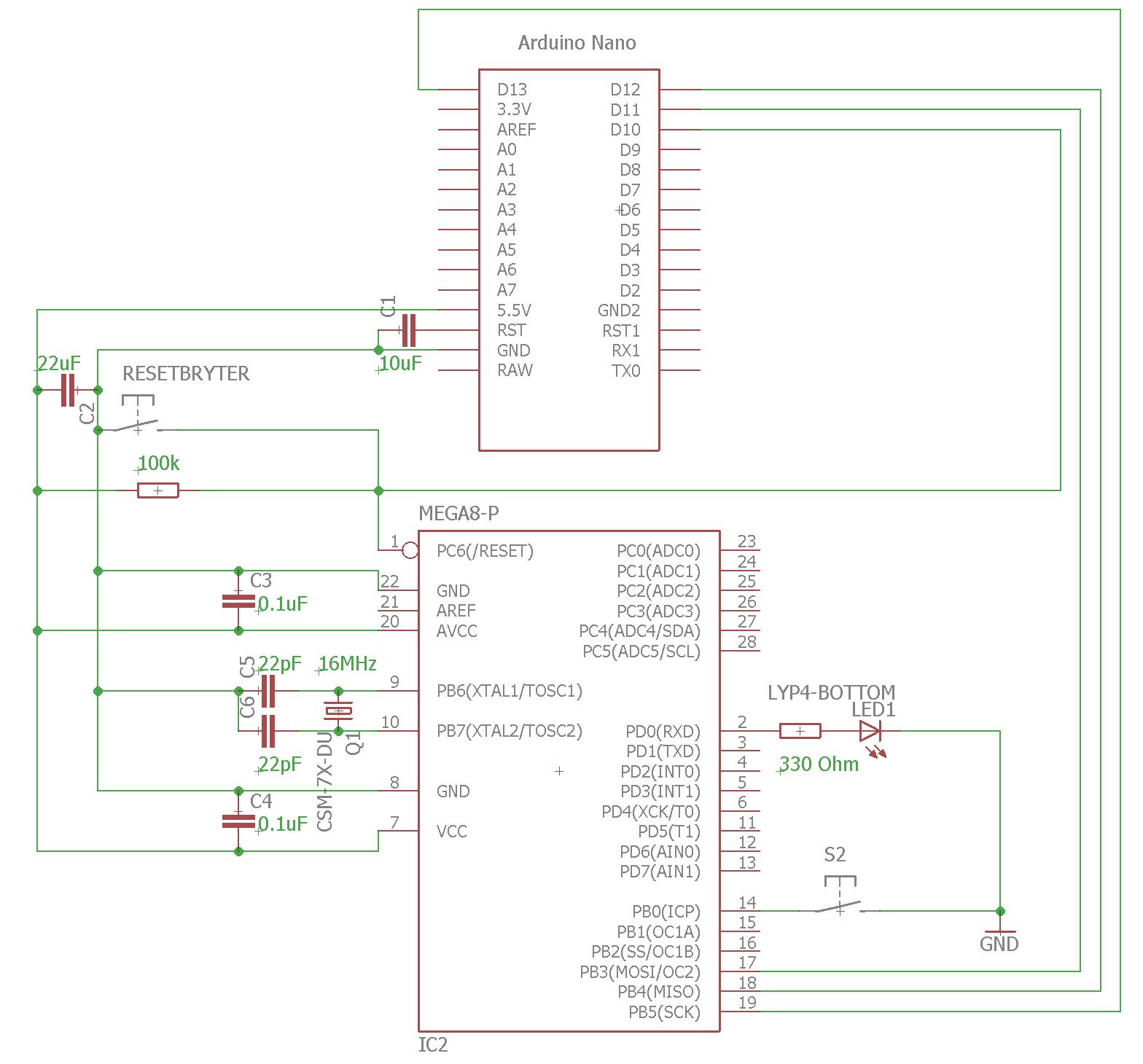
For å gjøre oss mindre avhengige av Atmel-ICE for å kjøre programmene underveis, fikk vi tak i en Arduino Nano som vi ønsket å bruke som en erstatning. Vi benyttet et eksempelprogram fra Arduino som gjør den om til en AVRISP. Videre opprettet vi et nytt verktøy i AtmelStudio som vi kalte for ”Upload”. Dette verktøyet ble brukt for å laste opp programmet til mikrokontrolleren vår, og konfigurerer fuses.



For å få en mer nøyaktig og stabil klokkehastighet, koblet vi opp en ekstern krystall på 16MHz. Vi fant veiledning for oppkoblingen i databladet til Atmega168.

(Usikker på om vi har en ”low power crystal” som tabellen nedenfor





1. Hva skjer med input-pinnen når knappen ikke er trykket og mikrokontrolleren ikke driver denne? Hvordan kan dette løses?

Se skriveboka!

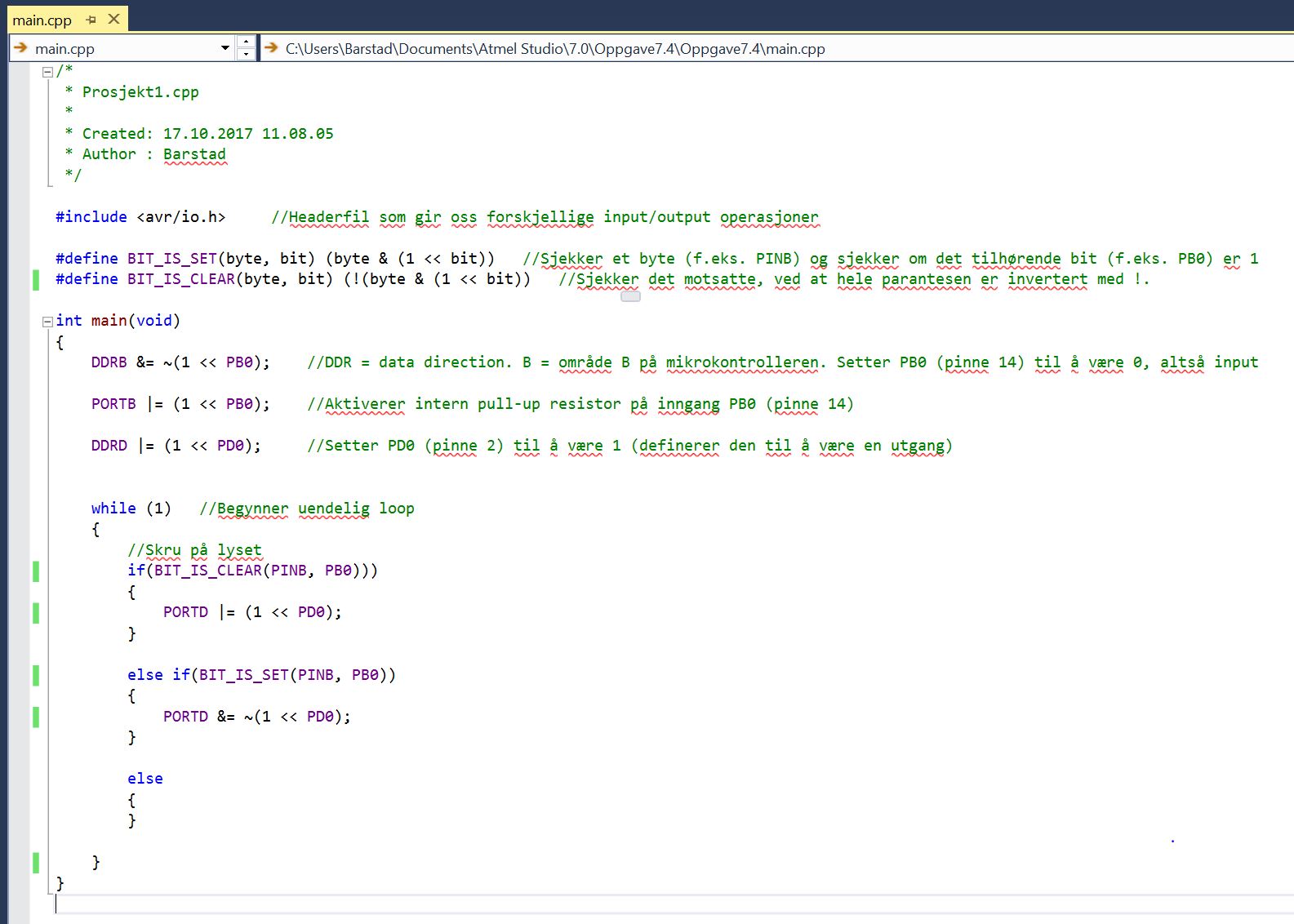
6.1) I PWM-mode tåler LED-en en peakstrøm på 140mA ved 1/10 duty cycle, 0,1ms pulse width. Dette er større enn maksimal konstant strøm på 25mA.

6.2)

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

Frekvensen vi valgte:

Vi koblet inn en bryter mellom jord og pinne 14. Denne ble brukt som et input-signal for å skru LED-en vår av/på. I programmet under definerte vi PB0 (pinne 14) til å være en inngang, samt aktiverte den interne pull-up motstanden til denne pinnen. Vi definerte også PD0 (pinne 2), hvor vi har koblet inn LED-en vår, til å være en utgang.



Gå til memory og utfør programmering. Beskriv hva som skjer. Hvilken filtype leses av programmereren, og hvor havner innholdet i denne?

1. Lag en enkel skjematisk oversikt over programmeringsoppsettet (ikke Eagle, men "boksologi"). Beskriv hva som er target og hva som er host.
2. Hvorfor sier vi at det utføres krysskompilering når vi kompilerer i Atmel Studio for AVR-mikrokontrollere?

i benytter Port B, pin 14. Denne kan være både input og output, og har en intern pull-up resistans.

Dersom vi ikke aktiverer den interne pull-up funksjonen, må vi fysisk koble opp en på brettet. Hvis ikke, vil vi kun lage en connection mellom input og jord.

