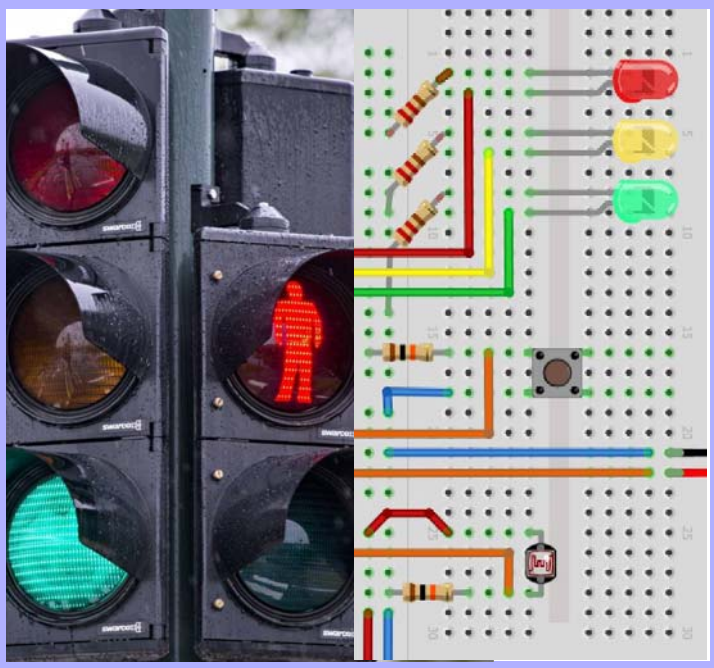
Rev 5.3

Nils Kr. Rossing

Grunnopplæring Arduino:
Oppgaver m/løsningsforslag



NTNU
Trondheim

Institutt for fysikk

Skolelaboratoriet
for matematikk, naturfag og teknologi

Januar 2024

Grunnopplæring Arduino: Oppgaver m/løsningsforslag

En guidet grunnopplæring i Arduino-programmering

Nils Kr. Rossing

Grunnopplæring Arduino: Trafikklys m/løsningsforslag
En guidet grunnopplæring i Arduino-programmering

Trondheim 2024

Layout og redigering: Nils Kr. Rossing, Skolelaboratoriet ved NTNU

Trykk: NTNU Grafisk senter

Tekst og bilder: Nils Kr. Rossing, Skolelaboratoriet ved NTNU

Faglige spørsmål rettes til:
Skolelaboratoriet for matematikk, naturfag og teknologi
Institutt for fysikk
v/ Nils Kr. Rossing nils.rossing@ntnu.no

Skolelaboratoriet ved NTNU
Realfagbygget,
Høgskoleringen 5,
7491 Trondheim

Telefon: 73 55 11 91
http://www.ntnu.no/skolelab/

Rev 5.3 – 31.01.24

Forord

Heftet er ment som en enkel introduksjon til Arduino-programmering. Fordelen er at de fleste oppgavene er lite krevende når det gjelder utstyr. Et vanlig Arduino starter kit vil derfor egne seg godt. Oppgavene egner seg derfor også godt for simulering. Denne utgaven av heftet inkluderer derfor en beskrivelse av kretssimulatoren i TinkerCAD slik flere av oppgavene skal kunne gjennomføres som ren digital undervisning.

Undervisningsopplegget inkluderer bruk av variabler, egen kodete funksjoner, for-sløyfer og if-kommandoer og bruk av både sensorer (bryter og lysfølsom motstand) og aktuatorer (lysdioder og lydgiver). Et trafikklys er dessuten noe alle er kjent med samtidig som få kjenner den tekniske virkemåten.

Fra og med utgave 3.0 er det relativt tidlig innført bruk av egendefinerte funksjoner som er brukt gjennom resten av opplegget. Flytskjema er også omtalt i innledningen og brukt for å vise programstrukturen tydeligere i oppgavedelen. Et gratis program for tegning av flytskjema er beskrevet i kapittel 4, side 44. Programmet er webbasert og finnes på følgende nettsted: https://app.diagrams.net/.

Fra og med utgave 4,0 er det lagt inn løsningsforslag etter hvert oppdrag for trafikklysoppgaven. Løsningsforslaget har en mangel som må rettes opp for å få det til å fungere etter spesifikasjonene. Mangelen, eller feilen er knyttet til et sentralt læringsmål ved det aktuelle oppdraget.

Trafikklysoppgaven har eksistert som undervisningsopplegg i ulike sammenhenger i mange år, men det er først nå at det er laget en egen tutorial for dette opplegget.

Fra og med utgave 5.0 er oppgavene delt inn i tre deler:

- *Inntroduksjonsoppgaver*
  Dette er oppgaver som egner seg som første innføring i programmering. Foreløpig er det bare to slike oppgaver.
- *Guidete oppgaver*
  Disse er oppgaver hvor kandidaten ledes trinn for trinn fram til en komplett løsning. Det gis også tips under veis. I to av oppgavene henvises til egne hefter. Det gjelder *Lag en høydemåler* og *Lag et kolorimeter*.
- *Halvåpne oppgaver*
  I de halvåpne oppgavene overlates en større del av oppdraget til kandidaten selv. Flere av disse oppgavene krever også noe ekstrautstyr ut over Arduino starter kit eller Sparkfun invention kit.

I den 5 utgaven omtales og programmet Fritzing som egner seg utmerket til å illustrere hvordan kretsene kan kobles opp på koblingsbrett.

Skolelaboratoriet ved NTNU
Januar 2024
Nils Kr. Rossing

Innhold

1 Innledning 13

1.1 Kort beskrivelse av oppdragene knyttet til trafikklys 13

1.2 Tilleggsoppgaver 15

2 Programmering med Arduino 17

2.1 Sparkfun Inventors kit: Grunnleggende byggeklosser 17

2.1.1 Komponentoversikter 17

2.1.2 Koblingsbrettet 18

2.1.3 Motstander 19

2.1.4 Sensorer 20

2.1.5 Halvledere 23

2.1.6 Aktuatorer 24

2.1.7 Arduino – mikrokontrollerkortet 27

2.2 Kort innføring i bruk av Arduino editoren (IDE) 28

2.3 Programstruktur og bruk av funksjoner 30

2.3.1 Programstruktur 30

2.4 Viktige kommandoer 31

2.4.1 Initiering av dataoverføring til PC 32

2.4.2 Kommentarer: 33

2.4.3 Bruk av variabler 33

2.4.4 Pause-kommando 34

2.4.5 Bruk av millis() 34

2.4.6 Aritmetiske og logiske operasjoner: 35

2.4.7 Digitale porter 36

2.4.8 Analoge porter 37

2.4.9 Sløyfer 38

2.4.10 If()-setning – For å kunne gjøre "veivalg" i programmet 40

2.4.11 Switch/Case-kommandoen 41

2.4.12 Definisjon av egne funksjoner 42

3 Simulering – TinkerCAD 45

3.1 Komme igang med TinkerCAD – kretssimulering 45

3.2 Oppkobling 49

3.3 Koding 49

3.4 Feilsøking 51

3.4.1 Bruk av "Break points" og skrittvis kjøring av programmet 51

3.4.2 Feilsøking ved hjelp av monitoren 51

3.5 Bruk av biblioteker 52

4 Bruk av flytdiagrammer 54
5 Fritzing 58
5.1 Bruksområde 58
5.2 Installasjon 58
5.3 Introduksjon til bruk 58
5.4 Oppbygging på koblingsbrett (Breadboard) 60
5.4.1 Overføring til koblingsskjema 62
5.5 Trykte kretskort 63
5.5.1 Fra skjema til PCB (Printed Circuit Board) 63
6 Introduksjonsoppgaver 64
6.1 SOS morsesender 64
6.1.1 Deloppdrag 1 – Blinkende lysdiode 64
6.1.2 Deloppdrag 2 – Send SOS med lys 65
6.1.3 Deloppdrag 3 – SOS med variabel hastighet 65
6.1.4 Deloppdrag 4 – SOS med lys og lyd 65
6.1.5 Deloppdrag 5 – Bruk av funksjoner 66
6.2 To blinkende lysdioder – Test øyets reaksjonsevne 66
6.2.1 Deloppdrag 1 – Blinkende lysdioder 66
6.2.2 Deloppdrag 2 – Test øyets reaksjonsevne 67
7 Guidede oppgaver 68
7.1 Lag et trafikklys 68
7.1.1 Prosjektets oppdrag 68
7.1.2 Deloppdrag 0 – Blinkende LED 70
7.1.3 Deloppdrag 1 – Enkelt trafikklys 73
7.1.4 Deloppdrag 2 – Bestill grønt lys 76
7.1.5 Deloppdrag 3 – Lag en funksjon som skifter til grønt og tilbake 81
7.1.6 Deloppdrag 4 – Blinkende grønt lys 84
7.1.7 Deloppdrag 5 – Universell utforming 88
7.1.8 Deloppdrag 6 – Blinkende gult lys 94
7.1.9 Deloppdrag 7 – Sammenkobling av to trafikklys 101
7.2 Lag et kolorimeter 102
7.2.1 Deloppdrag 1 – Montering av lysdiode og lyssensor i kyvettehuset 103
7.2.2 Deloppdrag 2 – Lag et program for å avlese lysstyrken 103
7.2.3 Deloppdrag 3 – Gjør målinger på 10 beger og sett dem i rett rekkefølge 103
7.2.4 Deloppdrag 4 – Koble opp og skriv til et eksternt display 103
7.2.5 Deloppdrag 5 – Skriv ut antall dråper i hvert beger 103
7.3 Lag en høydemåler 104

7.3.1 Deloppdrag 1 – Blinkende lysdiode, nedtelling og bruk av display 104
7.3.2 Deloppdrag 2 – Lag et voltmeter og et termometer 104
7.3.3 Deloppdrag 3 – Måling av lufttrykk med BMP280 104
7.3.4 Deloppdrag 4 – Omregning fra lufttrykk til høyde 104
7.3.5 Deloppdrag 5 – Kalibrering av instrumentet 105
8 Halvåpne oppgaver 107
8.1 Lag en batteritester 107
8.1.1 Deloppdrag 1 – Les av batterispenning 107
8.1.2 Deloppdrag 2 – Godt eller dårlig batteri 107
8.1.3 Deloppdrag 3 – LED som viser godt eller dårlig batteri 107
8.1.4 Deloppdrag 4 – Tegn koblingsskjema 107
8.1.5 Deloppdrag 5 – Uttesting av batteritesteren 107
8.1.6 Utstyr: 107
8.2 Automatisk blomstervanner 108
8.2.1 Deloppdrag 1 – Vanning ved tørr jord: 108
8.2.2 Deloppdrag 2 – Sikkerhet mot vannsøl: 108
8.2.3 Deloppdrag 3 – Differensiert vanning 108
8.2.4 Deloppdrag 4 – Tegn koblingsskjema 108
8.2.5 Utfordring 108
8.2.6 Utstyr: 108
8.2.7 Tips: 109
8.3 Automatisk vannvarmer 110
8.3.1 Deloppdrag 1 – NTC-motstand 110
8.3.2 Deloppdrag 1 – Termostat 110
8.3.3 Deloppdrag 2 – Alarm 110
8.3.4 Deloppdrag 3 – Tegn koblingsskjema 110
8.3.5 Utstyr: 110
8.3.6 Tips: 111
8.4 Elektronisk terning 112
8.4.1 Deloppdrag 1 – Planlegging av “rettferdig” terningen 112
8.4.2 Deloppdrag 2 - Oppkobling av terningen 112
8.4.3 Deloppdrag 3 – Skriv programmet 112
8.4.4 Deloppdrag 4 – Uttesting 113
8.4.5 Deloppdrag 5 – Lag en “urettferdig” terning 113
8.4.6 Deloppdrag 6 – Hemmelig bruk av den “urettferdig” terningen 113
8.4.7 Deloppdrag 7 – Tegn koblingsskjema 113
8.4.8 Utstyr: 113
8.4.9 Tips: 113

8.5 Stoppeklokke 115
8.5.1 Deloppdrag 1 – Lag en klokke som går 115
8.5.2 Deloppdrag 2 – Klokke med minutter, sekunder og 10-deler 115
8.5.3 Deloppdrag 3 – Start klokka med en trykkbryter 115
8.5.4 Deloppdrag 4 – Stopp klokka ved andre gangs trykk på bryteren 116
8.5.5 Deloppdrag 5 – Bruk stoppklokka til å måle reaksjonstida 116
8.5.6 Tips 116
8.6 "Trappelys" 116
8.6.1 Deloppdrag 1 – Tegn koblingsskjema 117
8.6.2 Deloppdrag 2 – Koble opp kretsen 117
8.6.3 Deloppdrag 3 – Skriv programmet 117
8.6.4 Deloppdrag 4 – Automatisk tenning av lyset ved mørkets frambrudd 117
8.6.5 Deloppdrag 5 – Automatisk tenning av lyset av bevegelse 117
8.6.6 Utstyr 118
8.6.7 Tips 118
9 Utdypende teori om aktuelle komponenter 122
9.1 Lysdioder 122
9.1.1 Ordinære lysdioder 122
9.1.2 Flerfargede lysdioder – RGB-dioder 125
9.1.3 Sykliske RGB-dioder 125
9.2 Lydgivere – Buzzere og Piezoelektrisk høyttaler 126
9.2.1 Buzzeren eller piezo-summer (aktiv buzzer) 126
9.2.2 Piezo-elektrisk høyttaler eller piezo-element (passive buzzere) 127
9.3 Spenningsdeleren 128
9.4 Lysfølsom sensor – LDR 130
9.4.1 Fotomotstand (LDR - Light Dependent Resistor) 131
9.5 Temperaturfølsom motstand (NTC- og PTC-motstander) 133
9.5.1 NTC-motstanden 133
9.5.2 NTCLE201E3103S 134
9.5.3 Oppkobling mot ADC 135
9.5.4 Optimal seriemotstand 136
9.5.5 Kalibrering av temperatursensoren 137
9.6 Fuktighetssensorer for bruk i jord 138
9.6.1 SeeedStudio Grove – Moisture Sensor (RB-See-186) 138
9.6.2 Capacitive Soil Moisture Sensor 139
10 Referanser 144
Vedlegg A Løsningsforslag på introduksjonsoppgaver 145
A.1 SOS morsesender 145

Vedlegg B Løsningsforslag guidede oppgaver 155
B.1 Trafikklys 155
Vedlegg C Løsningsforslag Halvåpne oppgaver 169
C.1 Batteritester 169
C.2 Automatisk blomstervanner 174
C.3 Automatisk vannvarmer 177
C.4 Elektronisk terning 179
C.5 Stoppeklokke 182
C.6 Trappelys 186
Vedlegg D "Plantede" feil i løsningsforslagene 193

1 Innledning

Mikrokontrollerkortet Arduino, med alle sine varianter, gir rike muligheter til å koble til sensorer og lagringsenheter av mange slag. Arduino UNO er kanskje den mest brukte i familien av kontrollerkort, men det finnes også mange andre, f.eks. Arduino Nano som er en god kandidat dersom det er viktig med små dimensjoner, eller Arduino MEGA dersom man trenger et stort antall porter. En av fordelene med å bruke Arduino UNO er at det finnes et rikt utvalg av "shield"-kort, som lett kan plugges sammen med UNO'en og gi utvidet funksjonalitet (de fleste av disse kortene kan også plugges på MEGA-en.

Ved bruk av mikrokontrollere og sensorer kan mange fagområder knyttes sammen. Det er f.eks. lett å knytte sensorteknologi til fysikk og kjemi (ev. materialteknologi), programmering og matematikk for beregning og kalibrering av målte størrelser, og datalogging når man beveger seg i skog og mark.

I dette undervisningsopplegget vil vi legge vekt på en grunnleggende innføring som gradvis bygger opp funksjonaliteten til et trafikklys ved hjelp av enkle moduler som settes sammen til et endelig produkt med en definert funksjon. Undervisningsopplegget burde egne seg som en første introduksjon til tekstbasert programmering av Arduino.

Å starte med et trafikklys kan ha flere fordeler:

- **Enkel inngang**
  Man kan begynne svært enkelt ved å få en lysdiode til å blinke.
- **Modulær oppbygging**
  Man kan gradvis bygge opp et stadig mer avansert produkt ved å legge til stadig nye funksjoner som hver for seg er relativt enkle.
- **Relevans**
  Alle kjenner trafikklys og er klar over funksjonen, samtidig som få kjenner de teknologiske løsningene og programmene som ligger bak. Opplegget gir derfor en dypere forståelse av et allerede kjent teknologisk hjelpemiddel.
- **Kreative løsninger og funksjoner**
  Siden alle har et forhold til trafikklys, har også mange tanker om funksjonelle forbedringer, som kan kalle på kreative løsninger og ny funksjonalitet.
- **Tverrfaglighet**
  Trafikklys og trafikk har også elementer av tverrfaglighet som spesielt kan utnyttes overfor barn.

Senere i heftet vil finner du flere oppgaver som illustrerer andre problemstillinger.

1.1 Kort beskrivelse av oppdragene knyttet til trafikklys

Oppdraget kan formuleres slik:

Oppdraget: *Det skal lages et program som styrer et trafikklys primært beregnet for en fotgjengerovergang. Lyset skal skifte fra rødt til grønt når en fotgjenger trykker på "fotgjengerknappen". Når den grønne perioden går mot slutten, skal det grønne lyset blinke for å varsle om at et skifte til rødt er nært forestående. Det skal lagges til lydsignal for synshemmede som indikerer at den grønne perioden snart er over. Når mørket senker seg, skal trafikklyset automatisk gå over til blinkende gult. Som en ekstra utfordring kan en tenke seg å kombinere flere trafikklys slik at lyset på begge sider av gangfeltet oppfører seg synkront. Dessuten er det naturlig å inkludere synkron styring av lyset for kryssende biltrafikk.*

Vi kan betrakte trafikklyset som et lite system som vi gradvis bygger opp gjennom deloppdrag, slik at tilførselen av kunnskap er overkommelig for hvert deloppdrag.

Under har vi ganske kort beskrevet de enkelte deloppdragene:

Deloppdrag 0 – Blinkende lysdiode: *Bygg opp og lag et program som får en lysdiode til å blinke.* Her lærer man å bruke editoren for Arduino, laste opp programmet og **skrive til digitale porter.** En kunnskap som brukes i resten av oppdragene.

Deloppdrag 1 – Enkelt trafikklys: *Bestemme hvordan lysene i et trafikklys skifter når det går fra rødt til grønt og fra grønt til rødt. Koble opp trafikklyset med rød, gul og grønn lysdiode og skriv programmet slik at det er rødt i 5 sek, gult i 1 sek og grønt i 5 sek.* Her er det snakk om mengdetrening mht. å slå av og på porter og å legge inn **tidsforsinkelser**.

Deloppdrag 2 – Bestill grønt lys: *Sekvensen skal nå endres slik at trafikklyset blir stående på rødt lys, helt til noen trykker på fotgjengerknappen. Etter 5 sek. skal lyset skifte til grønt på riktig måte, være grønt i 5 sek., før det igjen skifter tilbake til rødt, også på riktig måte, hvor det blir stående til det kommer et nytt trykk på fotgjengerknappen.* Her er fokus på å **lese av en knapp** (digital input) og legge inn veivalg i programmet, dvs. bruk av **if-setninger og betingelser.** Det er også naturlig å innføre bruk av **variabler** på dette tidspunktet.

Deloppdrag 3 – Lag en funksjon av skiftet til grønt og tilbake: *Det skal lages en funksjon av skiftet til grønt og tilbake igjen. Når det bestilles grønt lys skal programmet hoppe til funksjonen gront_lys() utføre den, og så hoppe tilbake igjen.* Dette deloppdraget er i sin helhet viet å **lage en egen funksjon** for grønt lys sekvensen. Denne kunnskapen vil også være nyttig ved senere deloppdrag.

Deloppdrag 4 – Blinkende grønt lys: *Programmet fra oppdrag 2 skal endres slik at det blir konstant grønt lys i 8 sek, deretter 5 blinkende grønne lys. Hvert blink skal være 500 ms på og 500 ms av. Trafikklyset skal ellers oppføre seg som i deloppdrag 2. I tillegg vil vi at programmet skal skrives slik at det er lett å endre lengden på grønt lys uten blink, lengden på hvert blink og antall blink.* Her er det naturlig å innføre bruk av **for-loop()** for å generere en blinksekvens. Dessuten vil bruk av **variabler eller konstanter** gjøre det enklere å endre lengden på de ulike periodene i lyssekvensen.

Deloppdrag 5 – Universell utforming: *Hver gang det grønne lyset blinker, det vil si slås av eller slås på, skal det etterfølges av et kort pip på 100 ms. Lyden skal lages av en egen funksjon som genererer lydsignalet med en lengde som bestemmes idet funksjonen kalles.* Her handler

det om bruk av lydfunksjonen, samtidig som det gjelder å få **timingen riktig**. Lyden skal komme idet lyset slås på og av, og bare når det er blinkende grønt. Dessuten skal funksjonen gront_lys() endres ved at **det overføres en parameter i argumentet.**

Deloppdrag 6 – Blinkende gult lys: *Når det blir mørkt skal lyset gå over i blinkende gult. Det skal da være 0,5 sek. på og 0,5 sek. av helt til det blir lyst igjen. Det skal være nok å legge hånda over den lysfølsomme motstanden for at det gule lyset skal begynne å blinke.* I dette oppdraget innføres **bruk av en sensor**, hvordan man **leser av en sensor**, **analog til digital omvandling** og **bruk av betingelser og if-setningen**.

Nå er trafikklyset ferdig. Det er imidlertid naturlig å tenke seg noen videreføringer.

Deloppdrag 7 – To eller flere trafikklys som fungerer sammen:
A) Tenk dere en fotgjengerovergang. Når en trykker på knappen på trafikklyset på den ene siden av gata, så skal begge lysene virke samtidig. Dette skal fungere uansett fra hvilken side bestillingen kommer fra.
B) Det skal også lages et lyskryss hvor det ene lyset er for fotgjengerovergangen og det andre for bilene. Løs disse oppgavene ved enten å gå sammen to og to grupper, eller ved å inkludere et ekstra sett med lysdioder i den ene Arduino UNO'en dere har (dette er aktuelt når dere bruker TinkerCAD). Her innføres ikke noe nytt, med mindre man ønsker å **koble sammen to Arduino UNO utviklingskort**. Det handler egentlig om å utvide systemet og trene på å se trafikklyset i en større sammenheng.

Vår intensjon med å lage opp oppgavene på denne måten er å vise hvordan man gradvis kan bygge opp et større system fra mindre moduler og å tilføre ny kunnskap der det er naturlig. Man tilfører kunnskapen når elevene er mest motivert, akkurat når de trenger den, "Just in time".

Det henvises også til to andre guidede undervisningsopplegg :

- Lag et kolorimeter (avsnitt 7.2, side 102)
- Lag en høydemåler (avsnitt 7.3, side 104)

Disse finnes i egne hefter og ligger også på nettet (www.ntnu.no/skolelab/bla-hefteserie).

1.2 Tilleggsoppgaver

For den som arbeider fort har vi laget en samling av "halvåpne" tilleggsoppgaver. Her gir vi en del tips, men guider ikke kandidaten på samme måte som for Trafikklys-oppdraget.

Følgende tilleggsoppdrag er beskrevet:

- Batteritester (avsnitt 8.1, side 107)
- Automatisk blomstervanner (avsnitt 8.2, side 108)
- Automatisk vannvarmer (avsnitt 8.3, side 110)
- Elektronisk terning (avsnitt 8.4, side 112)
- Stoppeklokke (avsnitt 8.5, side 115)
- Trappelys (avsnitt 8.6, side 116)

2 Programmering med Arduino

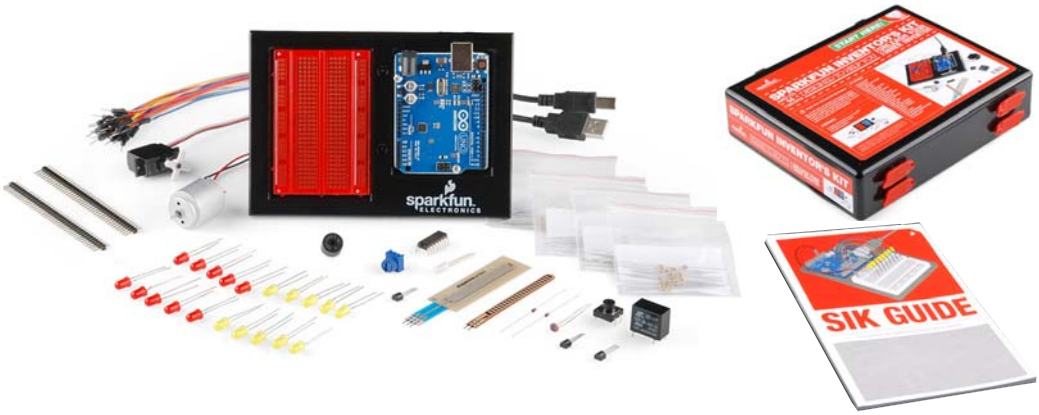
I dette kapittelet skal gi en oversikt over de komponentene vi skal bruke, beskrive hvordan vi installerer programvaren som skal til, hvordan vi bruker editoren, programstrukturen og de vanligste kommandoene i det programspråket: Arduino C++.

2.1 Sparkfun Inventors kit: Grunnleggende byggeklosser

Avsnittet gir en oversikt over de elektroniske byggeklossene vi skal bruke i de følgende undervisningsoppleggene. Komponentene som er nødvendige for å bygge opp et trafikklys er også tilgjengelig i TinkerCAD.

2.1.1 Komponentoversikter

Grunnopplæringen er basert på Sparkfun's *Inventors kit 3.1*.



Med settet følger et relativt rikholdig utvalg av elektroniske komponenter. Her er ei liste over innholdet i Sparkfun Inventor's kit 3.1:

- 1 stk. Arduino UNO R3
- 1 stk. Plastholder for Arduino og koblingsbrett
- 1 stk. SIK Manual
- 1 stk. Oppbevaringsboks for kittet
- 1 stk Koblingsbrett
- 1 stk Skiftregister – 74HC595
- 2 stk. Transistorer – 2N2222
- 2 stk. Dioder – 1N4148
- 1 stk. DC Motor med ledninger (1,5–3V) – 201-A
- 1 stk. Liten Servo (0–160°) A0090 - 9 g
- 1 stk. Rele 5–12V maks 5A – JZC-11F
- 1 stk. Temperatur sensor – +10mV/K – 0,5 V ved 0 °C – TMP36GZ

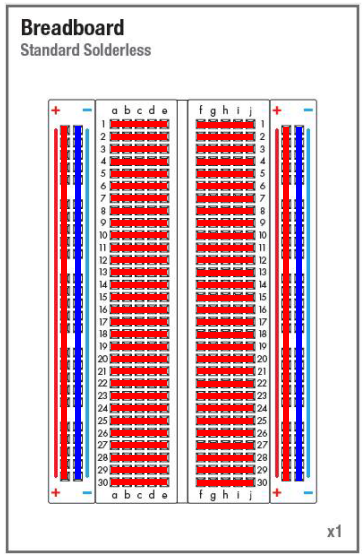
- 1 stk. Bøyesensor, Spectra Symbol 25 kOhm v/flat sensor – FS-L-0055-253-ST
- 1 stk. Membran-potensiometer, 100–10kOhm – SP-L-0050-103-ST
- 1 stk. USB kabel, 6’ vanlig A til B USB kabel
- 30 stk. Koblingsledninger, 7” Male/Male
- 1 stk. Fotomotstand, 8 kOhm (10 lux) – 1 MOhm (0 lux) topp ved 540 nm GL5528
- 1 stk. Tri-color LED, Intensitet (RGB): (800, 4000, 900) mcd YSL-R596CR3G4B5C-C10
- 10 stk. Rød lysdiode, 16–18 mA, maks 20 mA, Intensitet 150–200 mcd YSL-R531R3D-D2
- 10 stk. Gul lysdiode16–18 mA, maks 20 mA, Intensitet 40–100 mcd YSL-R341Y3D-D2
- 1 stk. Alfanumerisk LCD-display 2 x 16 karakterer
- 1 stk. Trimpot med ratt, 10 kOhm
- 1 stk. Magnetisk buzzer 100–10kHz 70–90 dB rel. 20µPa, maks 2048Hz, CEM1203(42)
- 1 stk. Trykkbryter, Big 12mm
- 20 stk. Motstand 330 Ohm 1/6 W
- 20 stk. Motstand 10 kOhm 1/6 W

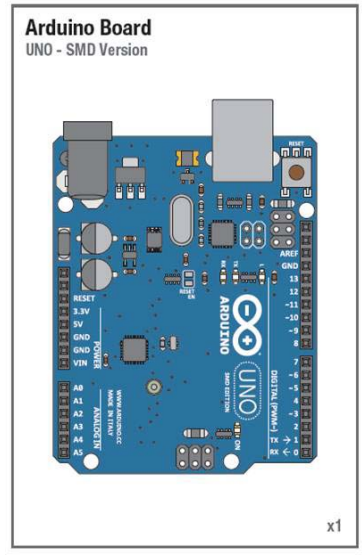
Mer informasjon om komponentene og datablader finnes på nettsidene til Sparkfun: https://www.sparkfun.com/products/11227. Dette kitet er i dag erstattet av *SparkFun Inventor's Kit - v4.1* som er omtalt på nettsiden: https://www.sparkfun.com/products/15267. Dette kitet inkluderer også deler for å bygge en enkel robot.

I dette opplegget skal vi bare bruke et fåtall av disse komponentene.

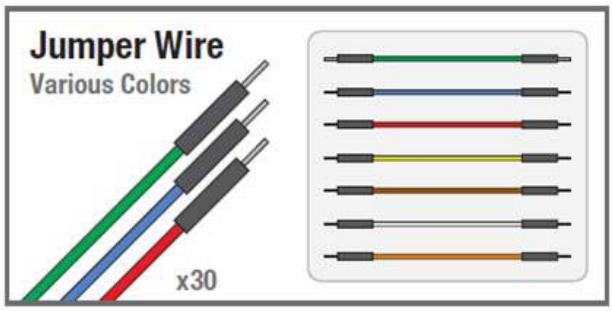
2.1.2 Koblingsbrettet

Normalt vil Arduino UNO og koblingsbrettet være montert på basisplata. Studer hvordan forbindelseslinjene i koblingsbrettet går.



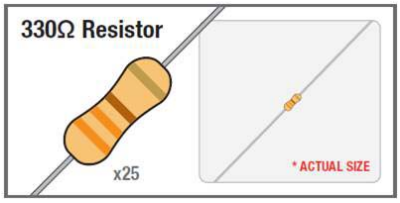


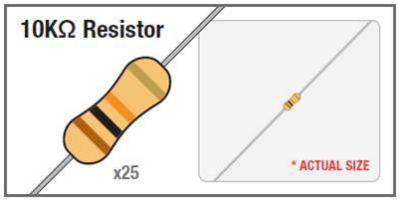
Strekene (røde) på koblingsbrettet viser hvilke koblingspunkter som er koblet sammen på undersiden. To komponentbein som er koblet til samme rad i to av hullene a – e eller f – j er elektrisk sammenkoblet. Komponentene plasseres på koblingsbrettet og forbindes med Arduino-en ved hjelp av jumperer.

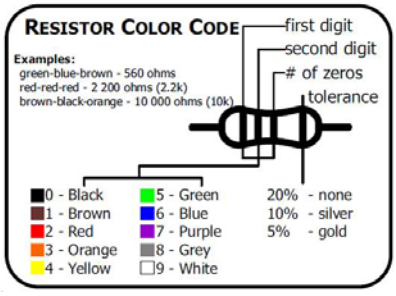


2.1.3 Motstander

Faste motstander







Beskrivelse:

Motstander kommer med mange ulike verdier. Her benyttes kun to: 330 Ohm (oransje, oransje, brun) og 10 kOhm (brun, sort, oransje). Fargekoden bestemmer verdien på motstandene. Det er viktig å velge riktig verdi.

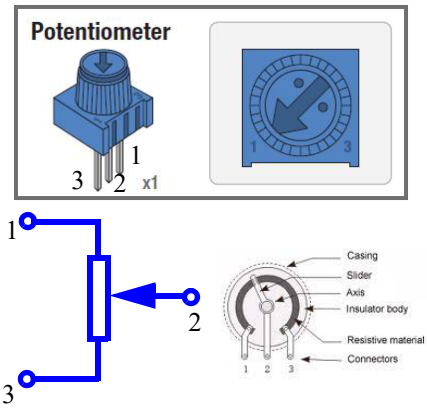
Bruksområder:

Motstander brukes til å begrense strømmen i en komponent, eller for å etablere et spenningsnivå slik som i spenningsdeleren. I følge Ohms lov vil spenningen over en motstand øke proporsjonalt med strømmen. Dette utnyttes når en ønsker å omdanne en varierende motstandsverdi som hos mange sensorer, til en varierende spenning. Det har ingen betydning hvilken vei motstanden kobles.

Bestem verdien:

Fargene på ringene bestemme verdien. Hver farge står for et av sifrene 0 til 9. Hold gullringen til høyre og les av fargene fra venstre mot høyre. Første og andre ring angir første og andre siffer i verdien. Tredje ring angir antall nuller. Er du usikker bruk et Ohm-meter eller multimeter.

Potensiometer



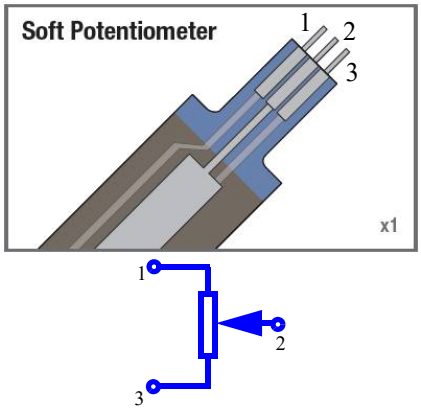
Beskrivelse:

Et potensiometer er en motstand med et variabelt uttak. Ved hjelp av et lite ratt kan uttaket flyttes langs motstanden. På denne måten kan en på pinne 2 ta ut en brøkdel av spenningen mellom pinne 1 og 3.

Bruksområder:

Potensiometeret kan etablere en variabel spenning mellom spenningen på ytterpunktene 1 og 3. Eller fungere som volumkontroll for et signal som sendes inn mellom pinne 1 og 3 og som tas ut mellom 2 og 3.

Trykkpotensiometer



Beskrivelse:

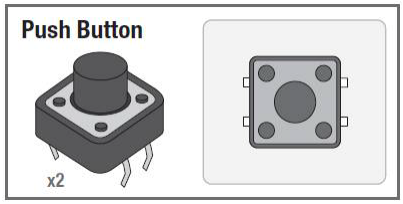
En trykkfølsom motstand fungerer på samme måte som et potensiometer. Istedet for å dreie på en knott, presser man på metallfilmen et sted mellom topp og bunn tilsvarende den spenningen man ønsker at potensiometeret skal gi ut.

Bruksområder:

Jeg har ikke sett slike trykkfølsomme potensiometer brukt andre steder enn i dette settet. Glidepotensiometer med en spak finner man f.eks. i miksebord.

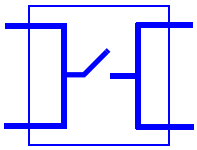
2.1.4 Sensorer

Bryter



Beskrivelse:

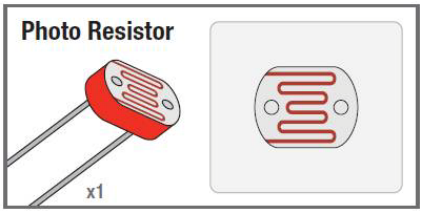
Bryterne har fire bein. De er forbundet med hverandre to og to som vist på tegningen under. Et trykk på knappen vil koble de to parene sammen. Forbindelsen opprettholdes så lenge knappen er trykket inn og brytes når den slippes.



Bruksområder:

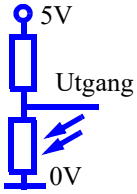
Brytere brukes til å gi en enkel på/av informasjon til kretsen, på samme måte som en lysbryter. For at kretsen skal forstå informasjonen må trykket omdannes til en spenning. Ved å koble bryteren mellom en motstand (10 kOhm) til 5 V og jord vil en på utgangen få en spenning som går fra 5 V til 0 V når bryteren trykkes inn.

Fotomotstand

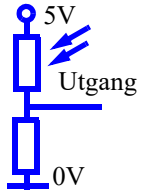


Beskrivelse:

En fotomotstand er en motstand hvor verdien bestemmes av intensiteten på lyset som treffer den. Jo mer den belyses, jo lavere motstandsverdi (mørke gir typisk 300 kOhm, sterkt lys 100 Ohm). Det betyr ingen ting hvilken vei den kobles inn i kretsen.



Høy spenning ut ved mørke



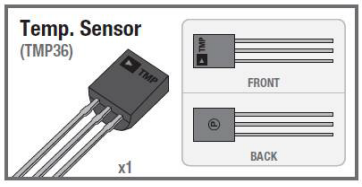
Høy spenning ut ved lys

Bruksområder:

Fotomotstander brukes der en ønsker å styre en funksjon ved hjelp av lysstyrken, som f.eks. tenning av lys når mørket faller på, telleapparater (en lysstråle brytes når noen går gjennom døra) o.l.

Den kobles gjerne i en spenningsdeler. Plasseringen bestemmes av funksjonen. Se figuren til venstre.

Temperatursensor

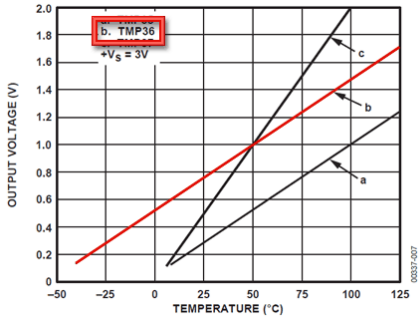


Beskrivelse:

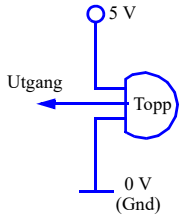
En temperatursensor (TMP36) av denne typen registrerer temperaturen og gir ut en spenning som er proporsjonal med temperaturen. OBS! Unngå å forbytte med transistorene!

Bruksområder:

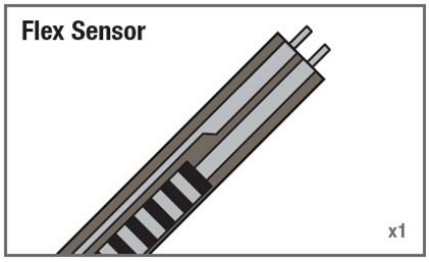
Temperatursensorer brukes i elektroniske termometre eller i termostater for å regulere en varmeovn. Den kan også brukes for å beskytte elektronikk, ved at strømmen brytes når temperaturen overskrider et maksimalt nivå. Hos TMP36 øker spenning med10 mV pr. grad C. Ved 0° C er spenningen 0,5 V.



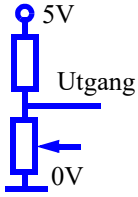
Spenning som funksjon av temperatur (TMP36 rød kurve)



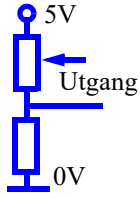
Bøyesensor



Zebrastripenen på motsatt side av trykket



Økende spenning på utgang



Avtagende spenning på utgang

Beskrivelse:

Ved å bøye sensoren vil motstanden endre seg. Bøyes den med sebrastripene på innsiden av bøyen faller motstanden til ca. 25 kOhm. Bøyes den med sebrastripene på utsiden av bøyen øker motstandsverdien til 65 kOhm. Verdiene avhenger av graden av bøyning.

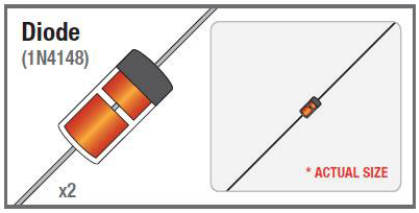
Bruksområder:

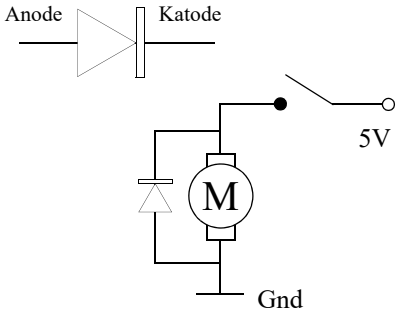
Lignende sensorer brukes i mange sammenhenger for å måle strekk eller sammentrykning i et materiale. Slik deformering kan f.eks. skyldes belastning eller nedbøyning. I denne sammenhengen går en slik sensor under betegnelsen *strekklapp*.

2.1.5 Halvledere

I denne sammenhengen er halvledere, dioder og transistorer.

Diode





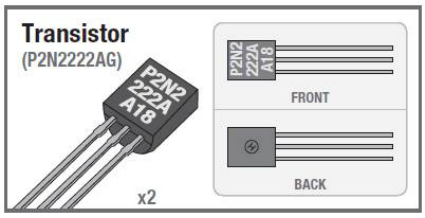
Beskrivelse:

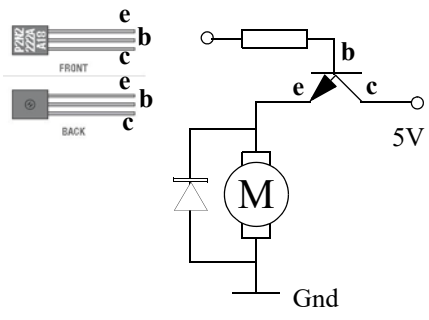
Dioden er en halvleder som kun leder strøm en vei, fra anode til katode, dvs. i pilens retning.

Bruksområder:

I denne sammenhengen skal vi benytte dioden som en "fly back" diode for å kortslutte strømmer som skyldes at strømmen i en spole, i en motor eller et rele skal ødelegge transistoren. Den må derfor monteres slik at den stenger for strømmen som normalt skal gå gjennom motoren. Når strømmen i motoren brytes, oppstår en motspenning som forsøker å hindre at motoren stopper. Denne motspenningen kan være så stor at den ødelegger transistoren. Dersom vi kobler en diode som vist på figuren til venstre, vil motspenningen kortsluttes gjennom dioden slik at den ikke når transistoren og ødelegger den.

Transistor





Beskrivelse:

Transistoren har tre terminaler (bein), Fra collector (c) til emitter (e) kan det gå en relativt stor strøm. Størrelsen på collectorstrømmen kan styres av spenningen mellom basen (b) og emitter (e). Når den blir stor nok begynner det å gå en strøm i transistoren.

Bruksområder:

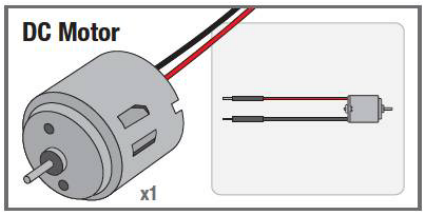
En transistor har gjerne to bruksområder. Som forsterker når basespenningen/strømmen varierer innen et lite område. Som bryter når basespenningen/strømmen er så stor at den slår transistoren på eller av.

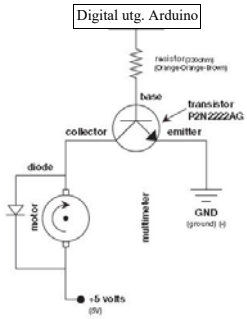
Transistorer som signalforsterker brukes i elektroniske forsterkere, radiosendere og mottakere, TV-er osv. I datamaskiner og i styringssystemer brukes den som bryter.

2.1.6 Aktuatorer

En aktuator er en komponent som kan utføre en handling, enten det er å skape mekanisk bevegelse (motorer, pumper, magneter, releer), gi lyd (øretelefoner, høyttaler, sirener) eller lys (LED, lyspærer) eller varme opp en gjenstand eller et rom (glødetråder).

Motor





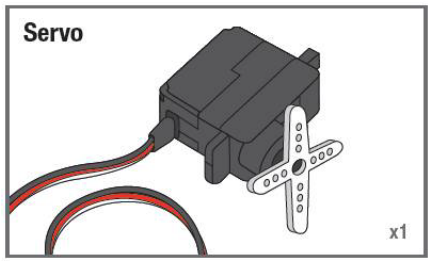
Beskrivelse:

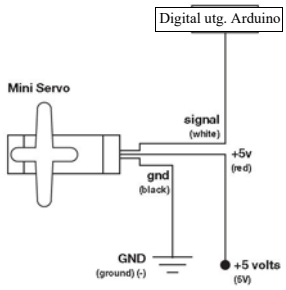
Akslingen begynner å rotere når motoren tilføres en spenning over 3 V. Motoren som følger med kitet kan brukes på spenninger opp til 40 V og strømmer på opp mot 200 mA. Siden Arduino'en ikke klarer å styre så store strømmer, benytter vi en transistorbryter som tåler strømmen. Legg merke til "fly back" dioden over motoren. Denne skal hindre overspenning på transistoren idet strømmen brytes for å stoppe motoren.

Bruksområder:

I dag brukes elektriske motorer til det meste der noe skal gå rundt eller bevege seg. Det være seg elektriske kjøkkenartikler, datadisker, leketøy som skal bevege seg, vaskemaskiner, pumper, vifter, vindusviskere og elektriske biler.

Servo





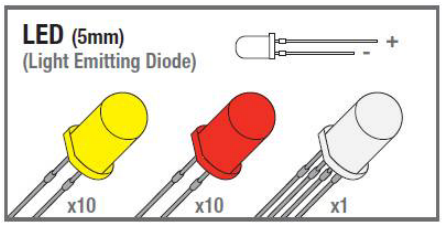
Beskrivelse:

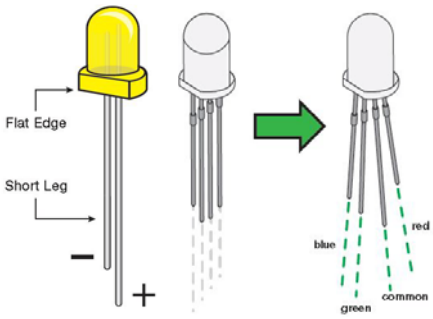
En servo er en slags motor som kan dreie akslingen en bestemt vinkel eller rotere med en definert fart. Denne servoen kan på kommando fra mikrokontrolleren dreie akslingen en vinkel på fra 0 – 180°. Dreiningen skjer ved at servoen mottar pulser, hvor lengden av pulsen bestemmer dreievinkelen. En pulslengde på 1,5 ms gir en vinkel på 90°. Pulsene må gjentas med jevne mellomrom omtrent som man pulsbreddemodulerer et signal. Vi kobler derfor styreinngangen til servoen til en utgang som har denne funksjonen, f.eks. port 9.

Bruksområder:

Servoer brukes ofte i modellfly for å styre side- og høyderor og flaps. De er også en viktig komponent i mange roboter som skal bevege en arm e.l.

Lysdioder

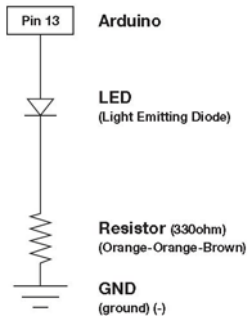




Beskrivelse:

Som vanlige dioder så leder lysdiodene strøm bare den ene veien, fra anoden til katoden. For at den skal lyse må den kobles opp rett vei. Når strømmen i dioden overstiger ca. 1,5–2 mA, begynner den å lyse svakt. Lysstyrken øker etter som strømmen øker. For å begrense strømmen, kobles den gjerne i serie med en motstand på 220–330 Ohm. Uten serie-motstand er det stor sannsynlighet for at dio-den går i stykker.

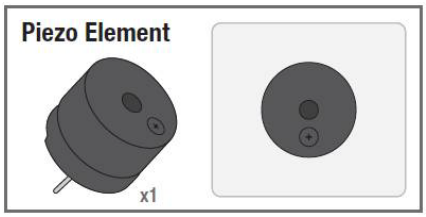
I settet er det vedlagt røde, grønne og gule lysdioder. I tillegg finnes en RGB-diode hvor en rød-, en grønn og en blå lysdiode er montert i samme kapsel.

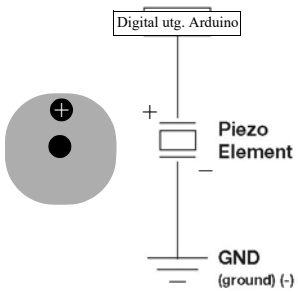


Bruksområder:

Lysdioder eller LED (Light Emitting Diode) brukes til signallamper, displayer og nå også i stor grad til belysning.

Buzzer





Beskrivelse:

Det piezo-elektriske elementet er et krystall som trekker seg sammen når det påføres en spenning og det høres et klikk. Når spenningen forsvinner, vil krystallet få tilbake sin opprinnelige form og det høres et nytt klikk. Ved å la spenningen variere fort kan man høre en lyd som i en høyttaler. For at den skal fungere som forventet, må + og – kobles rett.

Bruksområder:

Piezo-elektriske elementer brukes for å lage lyd og toner med forskjellig frekvens. Den kan også brukes i alarmer som f.eks. røykvarslere. En *buzzer* består av et piezo-element og en enhet som lager en varierende spenning. Buzzeren trenger derfor bare en spenning for å gi en tone. Tonen har ofte fast frekvens.

Display (2 x 16)

Beskrivelse:

Dette er et relativt vanlig LCD display med et parallelt grensesnitt som krever 10 I/O porter. Det er derfor ikke så attraktivt å bruke etter at displayer med I^2C-buss kommunikasjon er blitt mer vanlig. Displayet har sine fordeler ved at karakterene er store og godt synlig. Displayet leveres også med bakgrunnsbelysning.

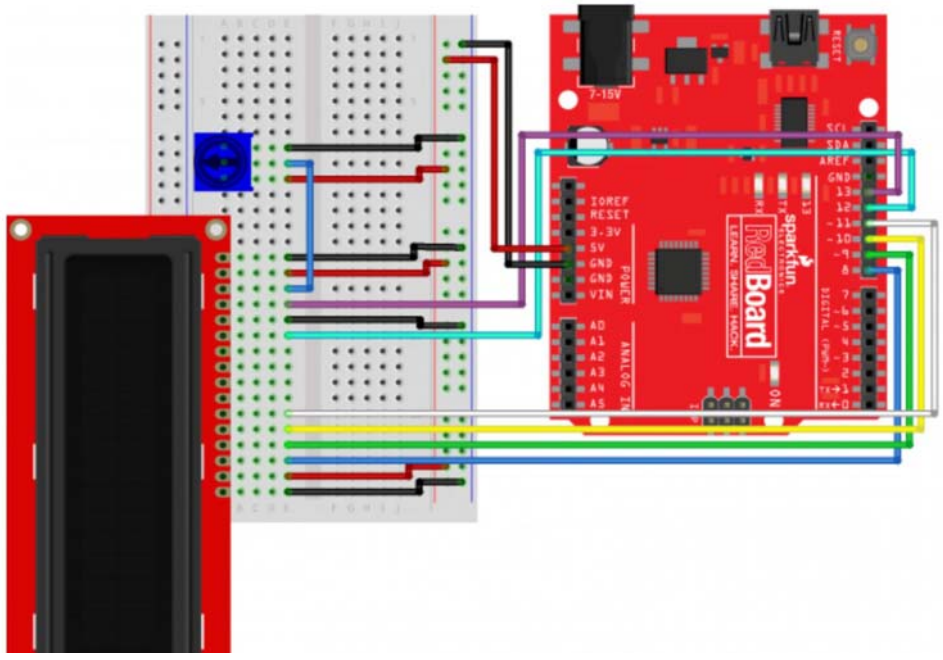
Bruksområder:

Displayet egner seg til å gir enkle beskjeder og måleresultater. Men anvendelsen blir litt begrenset siden det kun viser 2 linjer a 16 karakterer.

Pin Descriptions:
Pin 1 -- Ground
Pin 2 -- VDD Power for the LCD
Pin 3 -- Contrast Adjust
Pin 4 -- Register Select (RS)
Pin 5 -- Read / Write Select (R/W)
Pin 6 -- Enable

Pins 7-10 -- data lines d0 - d3 (not used)
Pins 11 - 14 -- data lines d4 - d7 (data transferred in 4-bits at a time)*

Pin 15 -- Backlight Power
Pin 16 -- Backlight GND (GND)

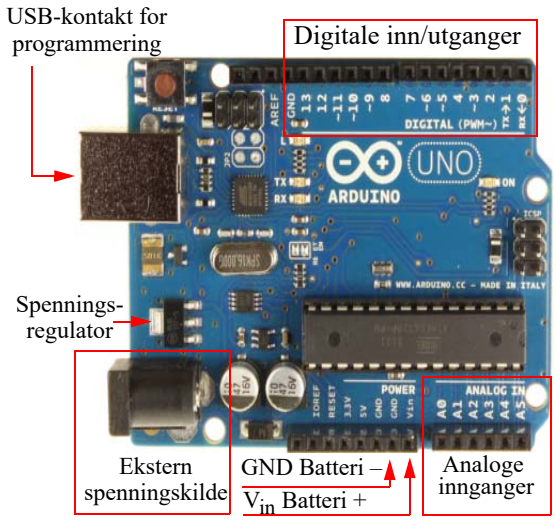


2.1.7 Arduino – mikrokontrollerkortet

Arduino UNO-kortet er databehandlingsenheten med sine inn- og utganger for sensorer og aktuatorer. Den har også en intern timer og internt lager for program og data.

I tillegg har kortet en USB-inngang for å legge inn programvare og ellers for kommunikasjon mellom PC-en og kortet. Det har også en plugg for batteri eller batteriadapter.

Arduino-kortet som følger med settet er Arduino UNO R3, som er ett av de mest populære mikrokontrollerkortene i Arduino-familien, og dermed leveres til en overkommelig pris (ELFA pris kr. 267,- + MVA)



Kortet er bygget opp omkring Atmel mikrokontrolleren ATmega328P med en klokkefrekvens på 16 MHz og et flash lager på 32 kbyte, SRAM 2 kbyte og EEPROM 1 kbyte.

Kortet har dessuten følgende inn- og utganger:

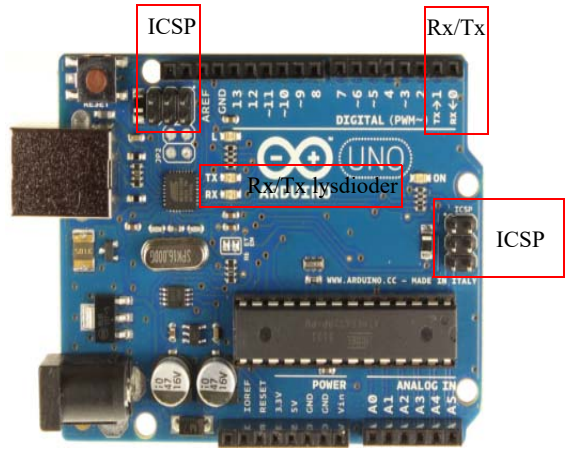
- **Digitale I/O-porter**
  Kortet har 14 digitale inn/utporter (I/O-porter) som kan programmeres til enten å være en inn- eller en utgang. Seks av disse (3, 5, 6, 9, 10 og 11) kan *pulsbreddemoduleres* (pwm), disse er merket med ~ på kortet. Maksimal strøm på hver av I/O portene er 40 mA.
- **Analoge innganger**
  Kortet har 6 analoge innganger som kan tilføres spenninger fra 0 – 5V. Spenninger ut over 5V kan ødelegge mikrokontrolleren. Disse kan også programmeres om slik at de kan brukes som digitale inn- og utganger.
- **USB-kontakt** for direkte tilkobling av PC og for programmering av kortet og overføring av data. I tillegg kan data overføres mellom monitoren på PC-en og kortet. Under programmeringen tilføres kortet spenning fra USB-kontakten. Dersom denne belastes med mer enn 500 mA vil strømforsyningen bli brutt inntil strømtrekket reduseres under denne grensen. USB3.2 Type C kan levere 5A. Imidlertid tåler ikke spenningsregulatoren på UNO-kortet så mye.
- **Strømtilførsel**
  Tilkoblingspunkter for batterieliminator anbefalt spenning 7 – 12 V (grenseverdier 6 – 20 V). Batteri kan enten tilkobles eliminatorpluggen (2.1 mm + senter) eller via V_{in} (+) og GND (–). Enkelte komponenter trenger lavere spenning som ev. kan leveres fra 3.3 V utgangen på kortet.
- **Reset**
  Kortet inneholder en RESET-knapp som resetter programmet og starter det på nytt.

Hylsekontaktene er montert slik at ulike tilleggskort ("shield card") kan monteres rett ned på Arduino-kortet.

Kortet støtter ulik typer datakommunikasjon med omverdenen: UART (Rx/Tx), I^2C-databuss og SPI-databuss.

For den interesserte så kan kretsskjema for Arduino UNO hentes fra følgende nettadresse: http://arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf.

For mer informasjon om Arduino UNO R3 se: http://arduino.cc/en/Main/ArduinoBoardUno/



2.2 Kort innføring i bruk av Arduino editoren (IDE)

Dette er en meget kortfattet innføring i bruk av editoren til Arduino. Hopp over om dette er kjent.

1. **Installasjon av programvaren**

   Det er normalt ikke nødvendig å installere en ny utgave dersom en gammel er installert i maskinen fra før. Eldre versjoner vil sannsynligvis fungere godt.

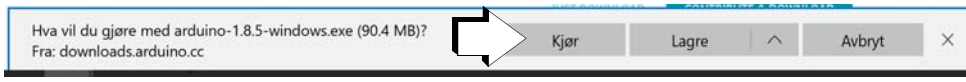
   Programvaren hentes fra:

   https://www.arduino.cc/en/Main/Software

   Velg: *Windows installer* (For Mac – *Mac OS X*)

   Velg: *Just Download*

   Velg: *Kjør*



Velg: *Følg prosedyren i installasjonen*

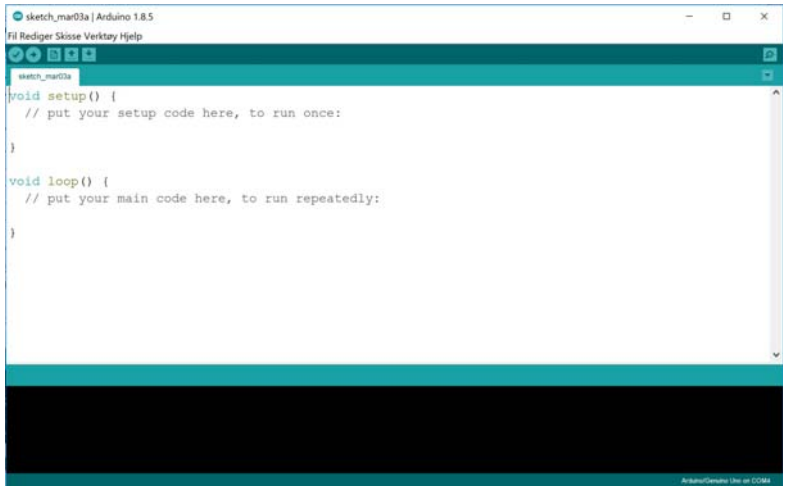
Velg: *Å installere drivere*

Du kan selvfølgelig velge å installer versjon 2.2.1 som også inkluderer verktøy for feilfinning.

2. Start programmet

Klikk på ikonet til Arduino og start programmet.

Du vil forhåpentlig se følgende programvindu åpne seg etter en liten stund.



3. Koble Arduino'en til en USB-kontakt

Se på menylinjen øverst til venstre.

Velg: *Verktøy.*
Velg: *Kort.*
Velg: *Arduino/Genuino UNO.*

Velg: *Verktøy.*
Velg*: Port.*
Velg: Det høyeste COM nummeret, eller velg porten merket med Arduino/Genuino UNO.

Det skal nå være opprettet kontakt mellom program-editoren (IDE) og Arduino-kortet.

4. Kort brukerveiledning for program-editoren

De viktigste kommandoene for å betjene bruken av editoren er oppsummert under:

Sjekk at koden er fri for skrivefeil (syntaksfeil)

Kompiler og send koden til kortet

Åpne en ny *skisse*[1] og start med blanke ark

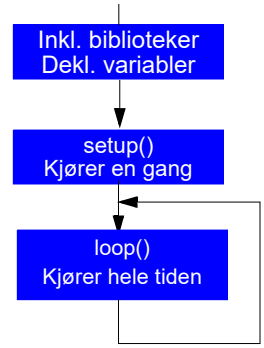
Last opp en programskisse fra disken

Lagre programskisse på disken

2.3 Programstruktur og bruk av funksjoner

2.3.1 Programstruktur

Alle Arduino-programmer består av to hovedfunksjoner:

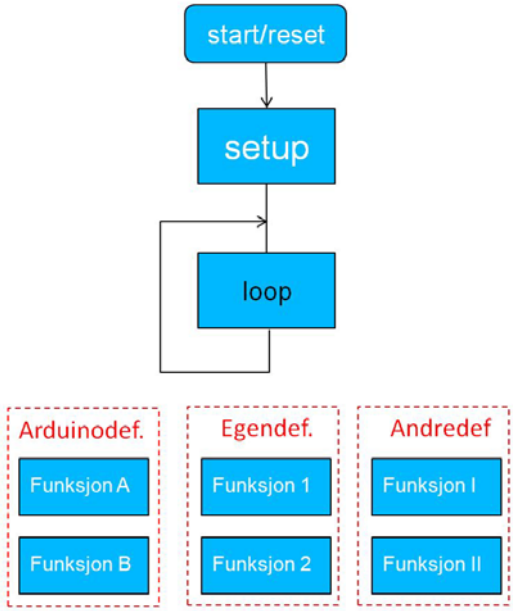


- `void setup()` som kjøres bare *en* gang hver gang programmet starter. Dette skjer etter at programmet er lastet opp, når man trykker på restart-knappen eller åpner monitoren.
- `void loop()` er en funksjon som gjentas så lenge Arduino'en har spenning.
- **Biblioteker og deklarasjoner** av globale variabler plasseres gjerne helt i starten. ***Globale variabler*** kan brukes i alle funksjoner. ***Lokale variabler*** deklareres i den enkelte funksjon og kan kun brukes innenfor denne funksjonen.
- Det er også vanlig å skrive ***egne funksjoner*** som enten legges i begynnelsen eller slutten av programmet, men kan i prinsippet legges hvor som helst utenfor `void setup()`- og `void loop()`-funksjonene.

1. Skisse er Arduinos navn på et program som skrives i editoren

Innholdet eller kroppen til de to hovedfunksjonene i Arduino-programmene er omsluttet av *klammeparenteser*. I `void setup()`-funksjonen initieres mikrokontrolleren og sensorer som evt. er tilkoblet, mens selve programmet legges under `void loop()`-funksjonen, som vist på figuren under.

```
#include <bibliotek> // Inkluderer spesielle biblioteker
Deklarer globale variable

void setup()
{
    // Koden i setup() kjører bare ved start
.... pinMode(<pin>, in/output);
}

void loop()
{
    // Deklarer lokale variable
    // Programlinjer

    funksjon1( ); // Kaller funksjon 1
...
}

void funksjon1()
{
    // Deklarer lokale variabler
    // kode
}
```



`setup()` og `loop()` er *navnet* til funksjonene, mens de to klammeparentesene {} omslutter som sagt *kroppen til funksjonen* hvor selve programmet ligger.

I tillegg til disse to faste funksjonene så bruker vi funksjoner fra Arduino's eget bibliotek (samlet under Arduinodef. på figuren), funksjoner som andre har laget (Andredef.) og vi kan lage våre egne funksjoner (Egendef.). Funksjoner som andre har laget og som de ønsker å tilby fellesskapet, samles ofte i *biblioteker*, som kan lastes ned, installeres og gjøres tilgjengelig for programmerere. For nærmere beskrivelse av hvordan man lager og bruker egne funksjoner, se avsnitt 2.4.12, side 42.

2.4 Viktige kommandoer

Referansemanualen til Arduino C++ for bruk ved programmering av Arduino finnes på følgende nettadresse: http://arduino.cc/en/Reference/HomePage

Referansemanualen kan også nås via *Hjelp* på menylinjen i programeditoren som vist på figuren til høyre.

2.4.1 Initiering av dataoverføring til PC

Under uttestingen og visning av innhold av variabler kan det være praktisk at data leses tilbake til monitoren i Arduino-editoren. Datahastigheten settes opp i `void setup()`-funksjonen med kommandoen: Serial.begin(9600); her er datahastigheten satt til 9 600 baud[2] (her ca. 9600 *bit* pr. sekund) som vist under:

```
void setup()
{
   Serial.begin(9600);
}
```

I noen tilfeller kan det være hensiktsmessig å øke overføringshastigheten. 115 200 tegn pr. sekund er også vanlig å bruke. Pass på at mottakerhastigheten for monitoren er satt til samme hastighet. Monitoren åpnes ved hjelp av det lille forstørrelsesglasset øverst til høyre i editoren. Dette gjøres nederst til høyre i monitoren som vist på figuren under.

Kommandoer for å skrive tilbake til PC, dvs. til monitoren i program-editoren:

Følgende kommandoer sender innholdet av en variabel eller en tekst tilbake til monitoren (terminalvinduet) i programeditoren.

```
Serial.print(a);          // Skriver variabelen a til en linje på skjermen,
                          // neste skrivekommando skriver på samme linje
```

2. Baud rate - Angir hvor mange symboler som kan overføres pr. transmisjonsperiode. For et binært system vil baud-raten tilsvare bit pr. sek. For mer komplekse former for modulasjon (f.eks. QPSK) vil hver transmisjonperiode kunne overføre flere bit (2, 4 eller flere), dog på bekostning av økt følsomhet for støy.

```
Serial.println(a);          // Skriver variabelen a til en linje på skjermen og
                            // skifter linje, neste skrivekommando skrives
                            // derfor på ny linje
Serial.println("Hallo");    // Skriver teksten Hallo til en linje på skjermen
                            // neste skrivekommando skriver på ny linje
```

Det er også mulig å kombinere tekst og variabler i samme print-kommando:

```
Seriel.println("Trykk:",a);// Skriver teksten Trykk: til en linje på Arduino
                            // monitoren, etterfulgt av innholdet i variabelen
                            // a, og skifter deretter til ny linje
Seriel.println(f, 2);       // Skriver desimalvariabelen f til terminal på PC
                            // med to desimaler
```

2.4.2 Kommentarer:

Kommentarer kan skrives hvor som helst og begynner med //

```
// Dette er en kommentar
```

Kommentarer blir fjernet under kompilering og overføres ikke til mikrokontrolleren. Kommentarene er derfor kun en hjelp for den som skriver programmet, og evt. andre som skal lese og forstå programmet senere.

Ønsker man å lage kommentarer over flere linjer kan dette gjøres slik:

```
/*
Alle tre linjene
vil bli betraktet
som kommentarer
*/
```

Midlertidig å *kommentere bort* programlinjer kan også være et nyttig hjelpemiddel under feilsøking.

2.4.3 Bruk av variabler

Bruk av variabler er en praksis som gjør programmering særdeles slagkraftig.

> *Vi kan betrakte variabler som oppbevaringssteder ("skuffer") for tekst eller tallverdier med navn og type. Innholdet vil som oftest være ukjent når vi skriver programmet, men vi reserverer plass til verdiene. Navnene på variablene bør gjenspeile hva de representerer slik at det blir lettere og lese og forstå koden. Variabler må deklareres med type.*

La oss deklarere variablene "`tempForskjell`", "`tempStart`" og "`tempSlutt`" som float (desimaltall). Vi kommer da til den andre viktige egenskapen:

> *Vi kan behandle og utføre beregninger på variablene uten at verdiene til variablene er kjente.*

```
tempForskjell = tempSlutt - tempStart;
```

Som variabelnavnene indikerer, så ønsker vi å beregne temperaturforskjellen mellom start- og sluttidspunktet. Dette fører oss til den tredje viktige egenskapen med variabler:

Vi kan bruke variabler som lagringsplass for verdier over tid.

Deklarasjon av lokale og globale variable

I programspråkene C og C++ må alle variabler deklareres før de kan brukes. Deklarasjonene må inneholde *type* og *navn* på variabelen og gjøres gjerne i starten av programmet før `void setup()`-funksjonen. Variabler deklarert utenfor funksjonene blir ***globale variabler***, dvs. at variablene kan brukes i alle funksjonene og beholder innholdet uavhengig av hvor de brukes.

Deklarering av variabler kan også gjøres *innenfor hver* funksjon. Slike variabler gjelder da bare innenfor denne funksjonen og kalles ***lokale variabler***. Under er vist deklarasjon av de vanligste typer variabler. I dette tilfelle vil de bare gjelde innen funksjonen `void loop()`:

```
void loop()
{
   Int a;          // deklarasjon av 16 bit heltallsvariabel (word)
   char b;         // deklarasjon av 8 bit karaktervariabel (byte)
   char c, d;      // deklarasjon av to 8 bits karaktervariabler (byte)
   float e;        // deklarasjon av variabelen e som et desimaltall f.eks. 1,65
                   // (32 bit, dobbel word)
   unsigned long f;// deklarasjon av 4 byts heltallsvariabel f (32 bit)
                   // uten fortegn
   boolean g;      // deklarasjon av en boolsk variabel g
                   // som kan ha verdiene 0 eller 1
   // Her følger resten av programkoden i funksjonen loop()-
}
```

Dersom vi deklarerer variabler i begynnelsen av loop()-funksjonen så vil de bli redeklarert for hver runde i loopen, hvilket betyr at vi må regne med at de vil miste den verdien de har for hver runde. Dersom vi ikke ønsker at dette skal skje kan vi deklarere dem utenfor loop() og setup() slik at de blir globale. Da blir de bare deklarert og nullstilt en gang, når programmet startes.

2.4.4 Pause-kommando

Dersom vi ønsker at programmet skal ta en pause kan vi skrive følgene:

```
delay(1000);              //Stopper programmet i 1000 msek (1 sek)
delayMicroseconds(100);   //Stopper programmet i 100 µs, 0,1 msek (0,0001 sek)
```

Dette medfører imidlertid at heller ingenting annet kan gjøres i denne tiden. Har man ikke råd til å miste denne tiden, bør man finne andre løsninger, f.eks. bruk av `millis();` som vist i neste avsnitt.

2.4.5 Bruk av millis()

Dersom vi vil unngå at programmet stopper helt opp mens vi venter på at et tiden er inne for å utføre en handling, kan vi bruke `millis()` istedet for `delay().millis()` angir til enhver tid tiden i millisekunder fra vi slo på strømmen eller resatte programmet og oppdateres kontinuerlig.

La oss anta at vi vil at programmet skal utføre en handling hvert 1000 ms (1 sekund) i tillegg til at programmet skal gjøre en del andre ting mellom disse tidspunktene. I så fall kan vi gjøre det slik:

```
long naaTidspunkt;


setup()
{
  naaTidspunkt = millis();
  ...
}

void loop()
{
  ...
  if(millis()- naaTidspunkt > 1000)
   {
    // Gjør det som skal gjøres hvert 1000 ms

    naaTidspunkt = millis(); // Sett nytt nåtidspunkt
   }
 // Resten av programmet
}
```

Dette er f.eks. aktuelt der man ønsker å telle og vise sekunder på et klokkedisplay hvert sekund.

Vi definerer en variabel `long naaTidspunkt`; Grunnen til at vi definerer variabelen som type `long` er for å unngå at variabelen skal tildeles verdier som er utenfor maksimal størrelse til variabelen siden `millis()` endres så fort. Bruker vi en `int` vil denne nå grensen for sitt området i løpet av bare ca. 32 sek. For bruk av `long` vil dette ta i underkant av 25 døgn. Bruker vi `unsigned long` vil det ta over 49 døgn.

I setup()-funksjonen setter vi `naaTidspunktet` lik `millis()`. I `void loop()`-funksjonen tester vi om det er gått 1000 ms siden vi tok vare på `naaTidspunktet` sist. Dette gjør vi ved å trekke det sist oppdaterte `naatidspunktet` fra den løpende `millis()`. Dersom differansen er større enn 1000 ms, så utfører vi vår handling samtidig som vi oppdaterer `naaTidspunktet` til løpende `millis()`.

2.4.6 Aritmetiske og logiske operasjoner:

```
sum = a + b;     // Summen av a + b settes i variabelen sum
diff = a - b;    // Differansen av a - b settes i variabelen diff
prod = a * b;    // Produktet av a * b settes i variabelen prod
kvo = a / b;     // Koeffisienten av a / b settes i variabelen kvo
eksp = pow(g,e);// Et grunntall g, opphøyd i en eksponent e (g^e)
```

Variabelnavnene *sum, diff, prod, kvo* og *eksp* er bare valgt som eksempler og må deklareres med type.

Dersom man ønsker å endre innholdet i en av variablene, kan dette gjøres slik:

```
a = a + b;      // Summen av a + b settes tilbake i variabelen a
```

Dette ser ikke lengre ut som en fornuftig "ligning" slik vi kjenner den fra matematikken og er det heller ikke. Her adderes verdiene *a* og *b* før summen legges tilbake til *a*.

I tillegg til de aritmetiske operasjonene har vi også tre logiske operatorer:

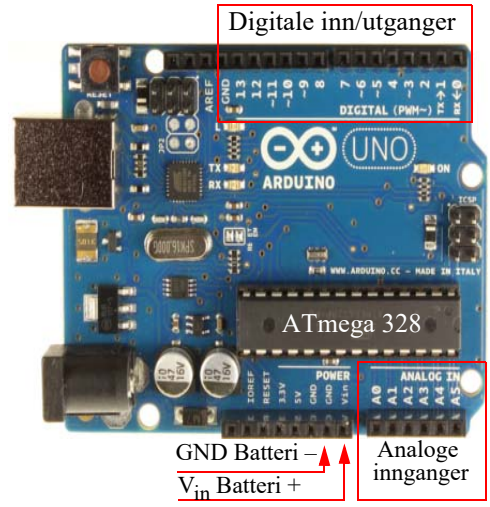
```
&&    // Uttrykker logisk "og", brukes når to betingelser må være sanne
||    // Uttrykker logisk "eller", brukes når en av to betingelser må
      // være sanne
!     // Negasjon av logisk verdi !sant er lik ikke sant
```

2.4.7 Digitale porter

Vi definerer digitale porter som inn- eller utganger:

En port er en fysisk terminal på kretsen som kan kobles til eksterne kretselementer, sensorer eller aktuatorer f.eks. LED. Disse kan enten være analoge eller digitale, innganger eller utganger. Siden en strømkrets må være sluttet, vil alle porter forholde seg til jord på kortet (dvs. minus hos strømforsyningen eller batteriet).

Kontrolleren ATmega 328 (Arduino UNO) har en rekke *porter*, 14 digitale (0 – 13) og 6 analoge (0 – 5, som også kan brukes som digitale I/O-porter). De digitale portene kan være innganger eller utganger. Hver port må derfor defineres som en inn- eller utgang. Dette gjøres gjerne i `void setup()`-funksjonen:




```
void setup()
{
   pinMode(8,OUTPUT);       // Definerer port (pinne) 8 som utgang
   pinMode(7,INPUT);        // Definerer port 7 som inngang
   pinMode(6,INPUT_PULLUP); // Definerer port 6 som inngang med
                            // pullup-motstand, dette er nyttig for at
                            // inngangen ikke skal henge fritt (sveve)
                            // når den ikke er tilkoblet noe
}
```

Dersom en inngang defineres med *pullup-motstand* betyr det at utgangen er koblet til pluss på batteriet ved en "stor" motstand internt i kretsen. Det betyr at dersom porten ikke er tilkoblet noe annet på utsiden så vil den ha verdi 1 (dvs. 5V for Arduino og 3,3V for ESP32).

Les fra og skriv til en digital port:

Digitale porter som er definert som utganger, kan enten settes til høy eller lav spenning. Digitale porter som er definert som innganger, kan lese av om spenningen på porten er høy eller lav. Dette gjøres med følgende kommandoer:

```
boolean boolsk;          // Definerer den boolske variabelen boolsk kan ha
                         // verdien 0 (usann) eller 1 (sann)
int heltall;             // Definerer en heltallsvariabel heltall,
                         // kan meget vel også brukes for 0 og 1

void loop()
{
digitalWrite(8, HIGH);   // Setter port 8 høy (5V evt. 3,3V på ESP32)
digitalWrite(8, LOW);    // Setter port 8 lav (0V)
boolsk = digitalRead(7); // Leser den digitale verdien på port 7
                         // og setter den inn i variabelen bolsk
heltall = digitalRead(6);// Leser den digitale verdien på port 6
                         // og setter den inn i variabelen heltall
}
```

Det er imidlertid vanligere å bruke heltallsvariabler (krever 2 byte) for å holde digitale verdier lest fra innganger. Man sparer imidlertid litt lagerplass dersom man bruker boolske variabler (krever 1 byte lagerplass).

2.4.8 Analoge porter

Les verdi fra en analog inngang og skriv til monitor (PC)

Syntaksen for lesing fra en analog inngang er slik:

```
<variabel> = analogRead(<analog port>);// Den analoge porten kan ha verdier fra
                                       // 0 - 5V hos UNO ev. 0 - 3,3V hos ESP32
```

Eksempel 1:

```
Int verdi;             // Deklarerer variabelen verdi
verdi = analogRead(0);// Leser digital verdi fra analog inngang 0,
                      // verdien leses inn i variabelen verdi
```

Eksempel 2:

```
void loop()
{
   int pressure;              //Deklarerer pressure som en heltalls-variabel
   pressure = analogRead(1);  //Leser av trykksensoren på AD-kanal 1
   Serial.println(pressure);  //Skriv resultatet tilbake til monitoren (PC)
}
```

Legg merke til at *Serial.print();* skriver *uten* påfølgende linjeskift, mens *Serial.println();* skriver ut *med* påfølgende linjeskift.

Eksempel 3:

```
void loop()
{
   int digitemp;               // Deklarerer digitemp som heltallsvariabel
   float anatemp;              // Deklarerer anatemp som float
   digitemp = analogRead(5);   // Leser av temperatursensoren på analog-kanal 5
   anatemp = digitemp * 500/1024;// Regner om til spenning og grader
   Serial.println(anatemp,2);  // Skriv resultatet tilbake til Arduino
                               // monitoren (PC) med 2 desimaler
}
```

De analoge portene er tilkoblet en analog til digital omvandler (AD-konverter) som gjør om spenningen 0 – 5V til en digital verdi fra 0 – 1023 (0000000000 – 1111111111). Dvs. AD-konverteren har 10 bits nøyaktighet. Tilsvarende kan ESP32 håndtere analoge spenninger fra 0 – 3,3V og har AD-konvertere med en oppløsning på 12 bit.

For å regne oss fra en digital avlesning med verdier fra 0 – 1023, til en spenning på 0 – 5V, så bruker vi følgende omregning, hvor `digitemp` er den avleste digitale verdien fra sensoren:

```
digitemp * 5.0/1024; // Svaret er den analoge spenningen i Volt

digitemp * 5000/1024; // Svaret er den analoge spenningen i milliVolt
```

For å regne oss om til en temperatur i grader Celcius gjør vi følgende:

```
anatemp = ((Analog spenning i milliVolt) - 500)/10; // Svaret blir i grader C
```

`anatemp` er en float (desimaltall) som angir temperaturen i grader C, mens `digitemp` er den digitale avleste verdien fra sensoren.

Siden den digitale verdien kan være 0 – 1023 skulle man tro at det ville være riktig å bruke 1023 og ikke 1024 i nevneren på brøken i uttrykket over. Imidlertid er det slik at den målte toppverdien for AD-konverter 1023 tilsvarer en spenning på 4,995 V (og ikke 5,0V). dermed vil det være riktigere å skrive 4,995/1023. Dette er imidlertid det samme som 5,0/1024 som er lettere å huske.

2.4.9 Sløyfer

Noen ganger har man behov for å gjenta en operasjon flere ganger, da er en *for-loop* egnet. Andre ganger vil man ha behov for å gjenta en operasjon så lenge en betingelse er oppfylt, i så fall kan man benytte en *while-loop.*

For-loop – For å gjenta mange like operasjoner (sløyfer)

For()-loop'en egner seg spesielt godt til å gjenta den samme operasjonen et bestemt antall ganger, kanskje med ganske små forandringer mellom gjentagelsene. En for()-loop kan skrives slik:

```
for(int x = 0; x < 100; x++)
{
// Her skrives koden som skal gjentas
```

```
}
```

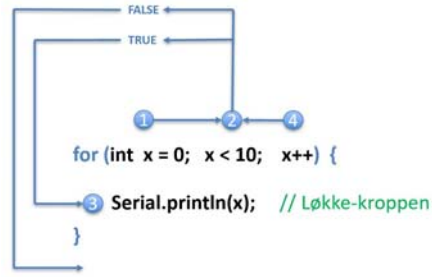
x er en heltallsvariabel (`int x`) som her brukes som teller (int i er også vanlig). Denne starter på verdien 0 (`int x = 0;`) økes med 1 (`x++`) for hver runde i loopen og stopper ikke før den når opp til og med 99 (`x < 100;`). De ulike utrykkene skilles med semikolon. Det som skal gjentas står innenfor klammeparentesene.

Eksempel:

```
for(int x = 0; x < 100; x++)
{
Serial.println(x);
}
```

Denne skriver ut tallene 0 – 99 til monitoren, ett tall for hver linje. Ønsker man å skrive ut tallene 0 – 100, kan man f.eks. skrive:

```
for(int x = 0; x =< 100; x++)
{
Serial.println(x);
}
```



Figuren til høyre viser rekkefølgen av testen og inkrementering av løkkevariabelen[3].

Legg merke til at linjen, `for(int x = 0; x =< 100; x++)` ikke avsluttes med semikolon, men etterfølges av { }.

While-loopen –
For å få programmet til å vente i en sløyfe til betingelsen ikke lenger er oppfylt

Det kan for eksempel være tilfelle når man venter på svar fra en sensor. While()-loopen kan skrives slik:

```
while (<logisk betingelse>)
{
// Her skrives koden som skal gjentas mens programmet venter
}
```

Legg merke til at linjen, `while (<logisk betingelse>)` ikke avsluttes med semikolon, men etterfølges av { } (som hos for-loopen).

3. Figuren er hentet fra en av Arne Midjos presentasjoner.

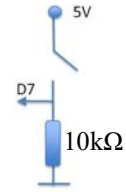
En while()-loop egner seg også for å hindre at en avlesning av en bryter skal medføre et skred av avlesninger.

Vanligvis ønsker vi at endring i tilstanden til en bryter skal medføre kun én hending.

I figuren til høyre ser vi en bryter som, når den trykkes inn, forbinder port D7 til +5V (logisk "1"). Når den ikke er trykket inn ligger port D7 til jord via en motstand på 10kΩ. Siden strømmen ut av port D7 til jord er tilnærmet 0, vil spenningen på D7 også være tilnærmet 0V (logisk "0").

While-lopp for å unngå mange avlesninger

```
int trykkBryterPin = 7;

void loop()
{
while(digitalRead(trykkBryterPin) == 1)
   {
     trykkBryterVerdi = 1;
   }
If(trykkBryterVerdi == 1)
  {
    // Gjør noe en gang når trykkBryterVerdi er lik 1
    trykkBryterVerdi = 0;
  }
}
```



I programmet ser vi at avlesningen av D7 (`trykkBryterPin`) gjøres inne i argumentet til `while()`-loopen. Så lenge avlesningen er lik "1", så skal programmet bli værende i loopen og gjentatte ganger sette `trykkBryterVerdi` til "1". Det er først når vi *slipper* bryteren at programmet forlater `while()`-loopen og går videre. Den påfølgende `if()`-setningen vil så sjekke om bryteren har vært trykket og utføre den ønskede handlingen. Vi må også huske å nullstille variabelen `trykkBryterVerdi` før vi forlater `if()`-setningen, ellers vil vi ved senere runder i loopen på registrert trykk på bryteren uten at vi har gjort det.

2.4.10 `If()`-setning – For å kunne gjøre "veivalg" i programmet

Noen ganger ønsker vi å gjøre forskjellige ting på bakgrunn av ulike betingelser, da kan vi bruke *`if()`-setninger*. Igjen viser vi et konkret eksempel:

```
if (i < 10)
{
   // Gjør dette dersom innholdet i variabelen i < 10
}
else if (i == 10)
{
  // Gjør dette dersom innholdet i variabelen i = 10
}
else
{
// Gjør dette dersom noe annet er tilfelle i dette tilfellet i > 10
}
```

Avhengig av verdien til variabelen *`i`*, utfører programmet ulike operasjoner. Parentesen etter `if()` skal inneholde en betingelse, denne kan være enkel, som her, eller ganske sammensatt. Man bruker da *aritmetiske operatorer* for å undersøke om noe er større enn (>), mindre enn (<) eller lik (==). Legg merke til det doble likhetstegnet. På tilsvarende måte kan en skrive større eller lik (>=) og mindre eller lik (<=).

Man kan lage flere betingelser med ulik respons ved å bruke en eller flere `else if()` med nye betingelser. Men alt som faller utenfor de definerte betingelse kan samles opp i en `else`.

Legg merke til at linjen, `if (i < 10)` ikke avsluttes med semikolon, men etterfølges av { }. Dvs. at `if()` er egentlig en funksjon. Heller ikke klammeparentesene etterfølges med ";" (semikolon). Slik er språket definert.

2.4.11 Switch/Case-kommandoen

Switch/Case kommandoen kan minne om `if()`-setningen, i og med at det er en kommando som styrer retningen til programmet på bakgrunn av verdien til en variabel, `var`.

```
switch (var) {
    case 1:
      //Gjør noe når variabelen var er lik 1
      break;
    case 2:
      //Gjør noe når variabelen var er lik 2
      break;
    default:
      // Om ingenting stemmer med variabelens verdi,
      // gjør default. Default er valgfritt
    break;
  }
```

Her ser vi at det er verdien til variabelen `var` som bestemmer hvilket av alternativene (`case`) som velges. Såfremt verdien til `var` eksisterer i lista over alternativer (`case`), så utføres det som er knyttet til det aktuelle tilfellet. Så snart handlingen er utført, sørger kommandoen `break;` for at programmet forlater lista over alternativer. Dersom ingen av alternativene finner "match" med verdien til variabelen *var*, så utføres `default`- alternativet.

```
void loop() {
  // read the sensor:
  int sensorReading = analogRead(A0);
  // map the sensor range to a range of four options:
  int range = map(sensorReading, sensorMin, sensorMax, 0, 3);

  // do something different depending on the range value:
  switch (range) {
    case 0:    // your hand is on the sensor
      Serial.println("dark");
      break;
    case 1:    // your hand is close to the sensor
      Serial.println("dim");
      break;
    case 2:    // your hand is a few inches from the sensor
      Serial.println("medium");
      break;
    case 3:    // your hand is nowhere near the sensor
      Serial.println("bright");
      break;
  }
  delay(1);        // delay in between reads for stability
}
```

Figuren over til høyre viser et eksempel på bruk av switch/case-kommandoen.

Her gjøres det en avlesning av en lyssensor i linje 8, verdien legges inn i variabelen `sensorReading`. En slik avlesning fra en analog inngang vil f.eks. ha verdier i området 0 – 1023. Dersom vi skulle ha en case for hver verdi, ville det bli et særdeles langt program. I stedet bruker vi en kommando som reskalerer området fra 0 – 1023 ned til et område på fra 0 – 4. Til denne reskaleringen bruker vi funksjonen `MAP()`:

```
range = map(sensorReading, sensorMin, sensorMAX, 0, 3);
```

Hvor `sensorMIN` og `sensorMAX` er henholdvis minste og største avlesning av lyssensoren. Funksjonen `MAP()` vil fordele verdiene mellom `sensorMIN` og `sensorMAX` på heltallsverdier mellom 0 og 3 som tilordnes variabelen `range`. `range` vil dermed få verdier fra 0 til 3 og vil dermed passe godt som variabel i switch/case- funksjonen som vist i figuren over.

Dersom man sløyfer `break;` under hvert tilfelle, vil programmet hoppe til det aktuelle tilfellet i henhold til variabelen, for deretter å gjennomføre alle etterfølgende tilfeller.

2.4.12 Definisjon av egne funksjoner

I figuren under til høyre ser vi `void setup()`-funksjonen og `void loop()`-funksjonen. Disse er ganske tomme i dette eksempelet, men vil normalt være fylt med kode.

Helt nederst har vi laget vår egen funksjon og kalt den `void funksjon1()`. Denne funksjonen er en bit av programmet som gjør en helt spesiell jobb. Det kan f.eks. være å få den grønne lysdioden i trafikklyset til å blinke et visst antall ganger.

Vi legger også merke til at vi finner funksjonsnavnet igjen under `void loop()`-funksjonen:

```
funksjon1();
```

Når programmet gjennomløper `void loop()`-funksjonen og kommer til `funksjon1()`, så hopper programmet ned til selve funksjonen nederst og utfører kommandoene i funksjonskroppen for så å hoppe tilbake til hovedprogrammet i `void loop()`.

Figuren til høyre viser et eksempel på en egendefinert funksjon. Funksjonen har vi kalt `blinkFemGanger()`. Det er viktig å gi funksjoner meningsbærende navn, som gjør at det er lettere å forstå hva programmet gjør når vi leser koden.

```
void setup()
  {
    // Gjøres en gang ved oppstart
  }

void loop()
  {
    // Gjentas hele tiden
   blinkFemGanger(); // kall funksjonen som lager blink
  }

void blinkFemGanger()
 {
  for (int i = 0; i < 5; i++)
    {
      digitalWrite(LEDpin, HIGH); delay(500);
      digitalWrite(LEDpin, LOW);  delay(500);
    }
 }
```

Definisjonen av funksjonen har vi lagt nederst i programmet. Her er den definert med et navn og et innhold som gjør nettopp det vi forventer – at det grønne lyset blinker fem ganger.

Bruk av argumenter

Hittil har parentesene etter funksjonsnavnet vært tomme. Her kan vi overføre parametere som påvirker hvordan programmet i funksjonen oppfører seg. Dersom vi ønsker at funksjonen skal blinke 6 istedet for 5 ganger, så kan vi lage en ny funksjon som nettopp gjør det, blinker 6 ganger.

En mer elegant måte å gjøre en funksjon mer generell på, er å la det være åpent hvor mange blink funksjonen skal utføre. Dette kan vi f.eks. gjøre ved å gi funksjonen et *argument* som overfører det antallet blink vi ønsker at den skal utføre når den kalles.

```
void setup()
  {
    // Gjøres en gang ved oppstart
  }

void loop()
  {
    // Det som skal gjentas hele tiden
   antallBlink = 6;                 // Heltallsvariabel
   blinkNGanger(antallBlink); // kall funksjonen som lager blink
  }

void blinkNGanger(int N)
 {
  for (int i = 0; i < N; i++)
    {
      digitalWrite(LEDpin, HIGH); delay(500);
      digitalWrite(LEDpin, LOW);  delay(500);
    }
 }
```

I eksempelet til høyre har vi gitt funksjonen argumentet `N`, som er et heltall. Legg merke til at typen til variabelen `N` deklareres i argumentet (`int N`). Variabelen `N` kan så brukes til å utføre funksjonens oppdrag, dvs. å blinke grønt `N` ganger.

I `void loop()`-funksjonen setter vi inn det ønskede antall blink som argument når vi kaller funksjonen. Her bruker vi variabelen `antallBlink` for å overføre antallet. Vi legger merke til at navnet på variabelen i kallet (antallBlink) og variabelen i funksjonsdefinisjonen (N) kan være forskjellige, men det er vanlig at variabelens type er den samme, i dette tilfellet et heltall (`int`).

Funksjoner som returnerer en verdi

Noen ganger vil funksjoner utføre beregninger og vi er interessert i resultatet av beregningene. I det neste eksempelet ønsker vi å lage en funksjon som beregner volumet av en kule.

```
...
void loop()
  {
    // Det som skal gjentas hele tiden
    float volum;
    float radiusKule = 5.0;
    volum = volumKule(radiusKule);    // Kall funksjonen
    Serial.println(volum);
  }

float volumKule(float radius)
 {
   float kulevolum;
   kulevolum = (4/3) * 3.14 * pow(radius, 3);
   return kulevolum;
}
```

Vi har kalt funksjonen `volumKule` med argumentet `radius` som er av typen `float` (desimaltall). Når vi kaller funksjonen har vi brukt argumentet `radiusKule`. Det betyr at i kallet så overføres verdien `radiusKule` til funksjonen via variabelen `radius`. Denne brukes så i beregningen. Kulevolumet returneres så med kommandoen `retur kulevolum;` som i sin tur legges i variabelen

`volum` som så skrives ut til monitoren med kommandoen `Serial.println(volum);`. Legg merke til at funksjonens type må være det samme som typen til variabelen vi vil returnere, i vårt eksempel `float`.

Når vi tidligere har brukt typen `void` på funksjoner som f.eks. `void setup()` og `void loop()`. så betyr det at funksjonen ikke returnerer noen verdi. Void betyr "tom".

3 Simulering – TinkerCAD

Trafikklysprosjektet egner seg godt for simulering siden alle komponentene som anvendes finnes i biblioteket til TinkerCAD.

TinkerCAD er et simuleringsprogram som kan brukes både til modellering for 3D-printing, simulering av enkle elektriske og elektroniske kretser i tillegg til programmering av Arduino og Micro:bit. Programmet gir også mulighet til å bygge opp strukturer med LEGO klosser. Programmet er gratis og nettbasert, og har lav brukerterskel.

Den tidligere Google-ingeniøren ***Kai Backman***, startet utviklingen av programmet sammen med ***Mikko Mononen*** i 2010. Målet var å lage et lavterskel programverktøy for design av gjenstander for 3D-printing, samtidig som de ønsket å skape en arena for deling av modeller utviklet av brukerne. I 2011 ble programmet lansert og i 2012 flyttet firmaet fra Europa til San Fransisco.

I 2013 ble TinkerCAD oppkjøpt av Autodesk og i 2017 la Autodisk ned sitt eget modelleringsprogram *123D Sculpt* for å satse på TinkerCAD. Samme år ble deres 123D circuits "Elektronics Lab" integrert i TinkerCAD.

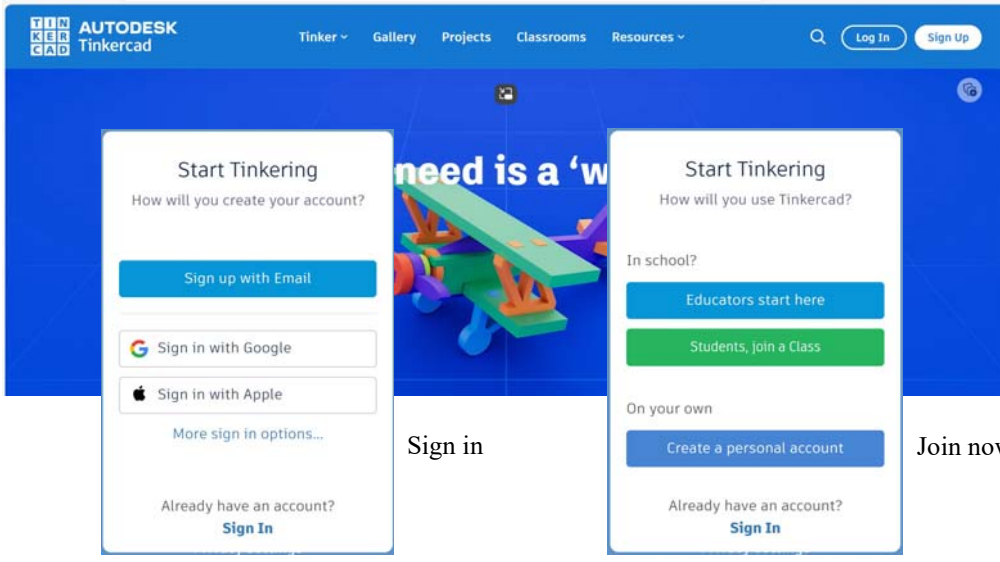
... og det er nettopp denne kretssimulatoren vi ønsker å bruke her i denne oppgaven.

3.1 Komme igang med TinkerCAD – kretssimulering

Programmet åpnes i en nettleser fra siden:

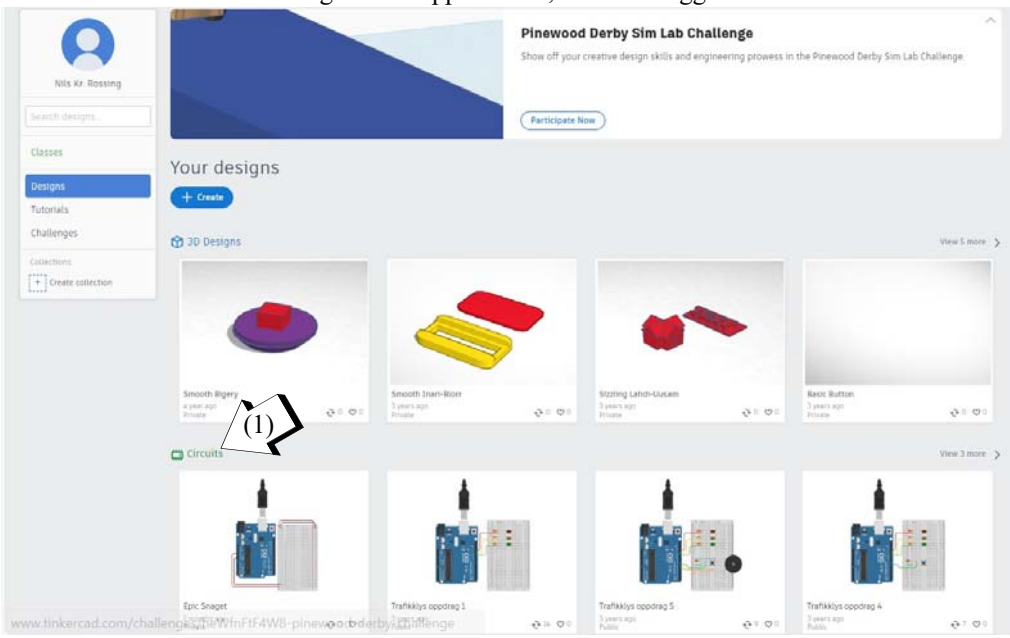
https://www.tinkercad.com/

Ved enten å velge "Log In" eller "Sign Up" får man anledning til å logge inn, opprette en konto som lærer eller delta som student.

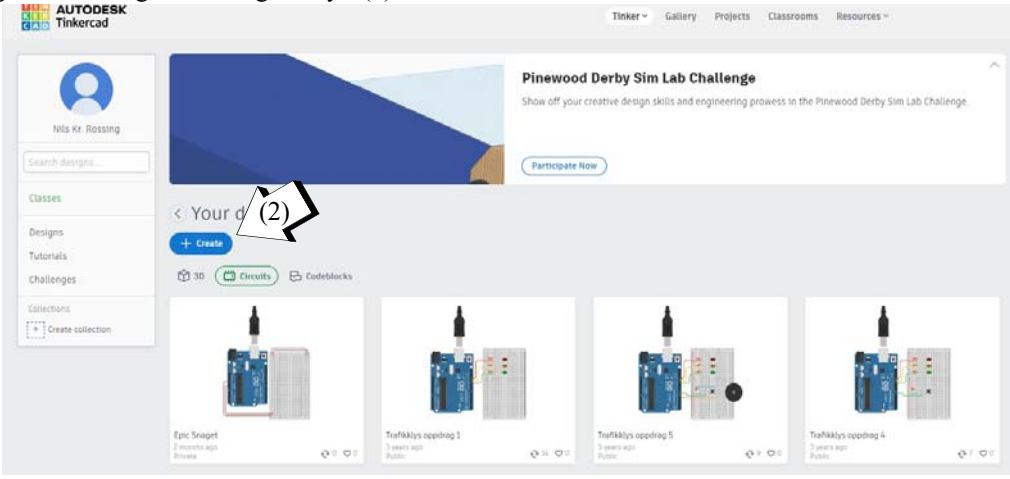


Sign in Join now

Dersom man f.eks. har en Google eller Appel-konto, kan man logge inn via en av dem.



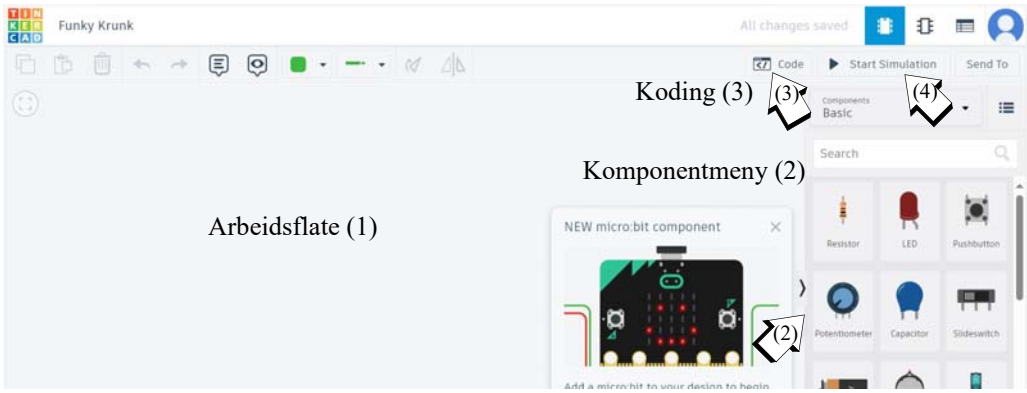
Etter å ha logget inn, kan man velge hva man ønsker å jobbe med: 3D-design, kretser eller kodeblokker. I denne sammenhengen velger vi "circuits" (1) – kretser. Vi kan nå velge å hente opp et gammelt design eller velge et nytt (2).



Vi velger å lage et nytt design og programmets brukergrensesnitt vises.

Alternativt kan man gå direkte til kretser med denne kommandoen:

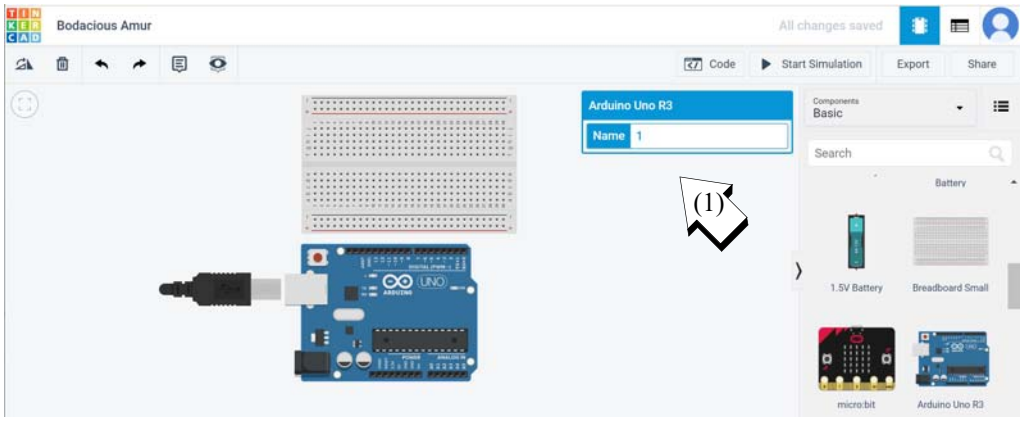
https://www.tinkercad.com/learn/circuits



Ved oppstart kommer det opp en presentasjon av nye komponenter. Denne gangen en micro:bit. Den kan vi fjerne for dette prosjektet.

1. **Arbeidsflata:** På arbeidsflata setter vi sammen og bygger opp kretsene som vi skal simulere.
2. **Komponentmenyen:** Til høyre finner komponentmenyen med en oversikt over komponentene som er tilgjengelige. Denne er ordnet i kategorier. Det er ikke flere komponenter enn at vi kan velge "All" som gir oss oversikt over alle i lageret.
3. **Koding:** Velger vi å arbeide med mikrokontrollerne Arduino UNO eller Micro:bit så vil "Code"-knappen gi oss mulighet til å programmere i tekst (C++) eller blokkode.
4. **Simulering:** Når koden er bygget opp kan vi velge å simulere kretsen, ved å velge "Start simulator".

Kretsen bygges opp ved å velge komponenter fra komponentmenyen og dra dem ut på arbeidsflata. Her har vi valgt et koblingsbrett (half+) og en Arduino UNO. Hver av komponentene kan gis et unikt navn eller nummer om man ønsker det. Dette oppgis i den blå innboksen (1) som framkommer når vi klikker på komponenten.



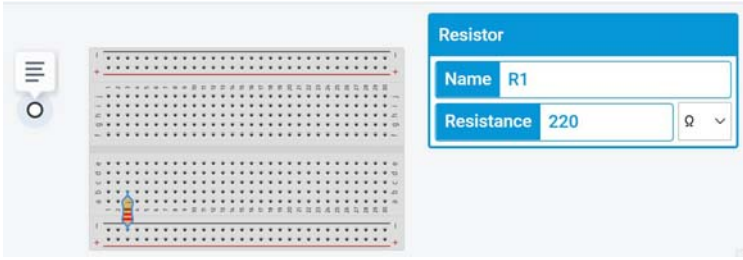
Zoom – Man kan zoome inn eller ut av kretsskjemaet ved hjelp av hjulet på musa.

Flytt – Arbeidsflata kan flyttes ved å holde ned venstre musknapp og dra slik at kretsen blir liggende på ønsket sted i arbeidsflata

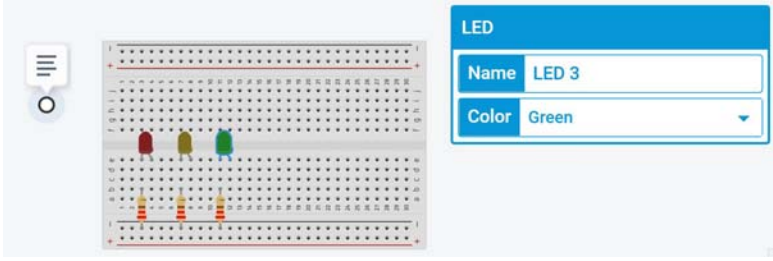
Oppe i venstre hjørne finnes en meny med følgende verktøy:

1. **Kopier** kretsen eller kretsene som er merket.
2. **Ukjent** betydning
3. **Slett** den merkede komponenten
4. **Angre** siste handling
5. **Reetabler** siste angrede handling
6. **Legg inn en kommentar** i kretsskjemaet, eller til en komponent
7. **Skjul merkelapper** som viser kommentarer
8. **Angi farge** på jumperne
9. **Dra nye jumpere**
10. **Drei** den merkede komponenten mot høyre, 90° av gangen.
11. **Speil** den merkede komponenten
12. **Gi kretsen et navn:** "Funky Krunk" erstattes med et passende navn: "Trafikklys 1"

Når man velger motstander så vil man kunne gi hver motstand ett navn og en verdi. Fargekodingen til motstanden endres i henhold til verdien vi har gitt den.

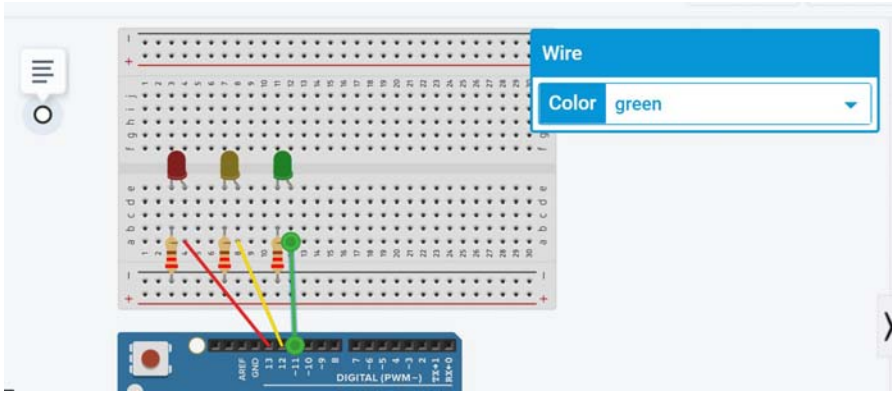


På lignende måte gis andre komponenter navn og evt. verdi, eller som hos lysdiodene, en farge.

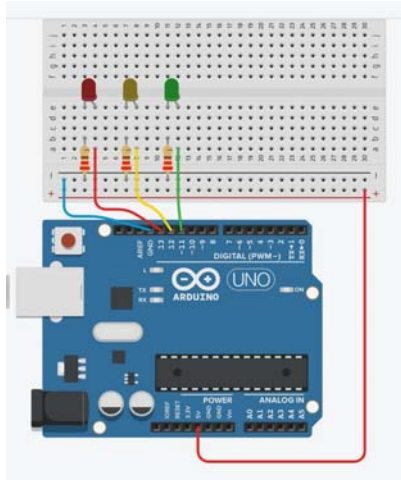


3.2 Oppkobling

Når vi har plassert komponentene der vi ønsker, kan vi begynne å dra ledninger (jumpere).



Forbindelsen opprettes ved å dra en ledning mellom de to kontaktpunktene (hullene) man ønsker å forbinde. Et dobbeltklikk vil feste ledningen til hullet. Det kan være greit at man innledningsvis trekker ledningene i rette linjer mellom hullene for deretter å legge inn bøyer.



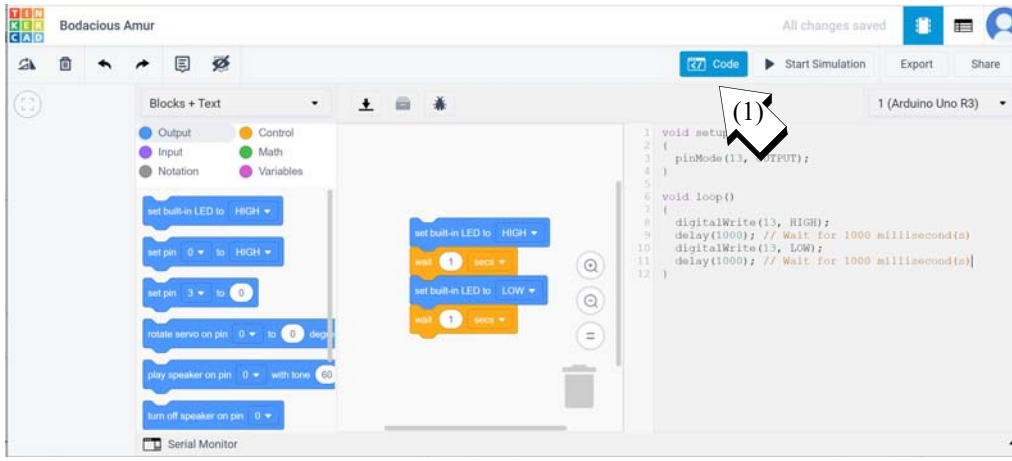
En ledning som er "merket" vil bli noe tykkere og festepunktene vil få kraftige ringer som vist på figuren over. Man kan innføre en bøy på en ledning som er merket, enten ved å slippe opp venstreknappen og så trekke på nytt mens man tegner, eller man kan merke ledningen i ettertid og dobbelklikke på det punktet av ledningen der man ønsker å legge inn en bøy. Ledningen formes ved å dra i punktet.

Når kretsen er koblet opp er man klar til å begynne å programmere.

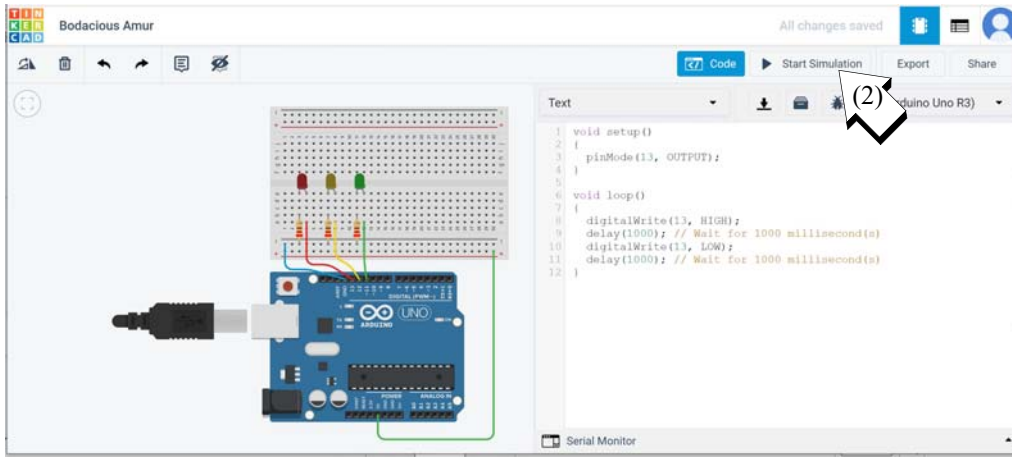
3.3 Koding

Koding av mikrokontrollere kan enten gjøres blokk- eller tekstbasert:

1. **"Code":** Her velges enten blokk- eller tekstbasert koding. Man kan også velge begge deler slik at man kan observere sammenhengen mellom blokk-kode og tekst. Dvs. at når man velger blokker i koden til venstre vil den tekstbaserte koden dukke opp i vinduet til høyre, men ikke omvendt. Dette kan være nyttig når man ønsker å bruke blokk-kode til å lære seg tekstbasert koding.

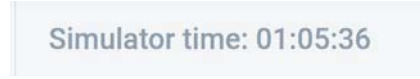


Man kan selvfølgelig velge en av delene, enten blokk-kode eller tekstbasert koding. Vi velger tekstbasert koding og får opp et vindu som vist under:



Etter at koden er skrevet, kan man starte simulatoren.

2. **"Start simulation"**: Velger man å starte simulatoren vil kretsen simuleres og utføre kommandoene i programmet. Oppe midt på skjermen vises en klokke, denne viser tiden fra simulatoren startet.



Det er denne klokka simuleringen forholder seg til. Dersom det f.eks. er lagt inn et delay på 5 sekunder i programmet, er det 5 sekunder på denne klokka som teller. *En bør være klar over at simulatorens klokka kan avvike fra den virkelige klokka*.

3.4 Feilsøking

3.4.1 Bruk av "Break points" og skrittvis kjøring av programmet

Til forskjell fra den ordinære editoren for Arduino (IDE) gir TinkerCAD en enkel mulighet til å legge inn "break points" (stoppunkter) og til å kjøre koden trinn for trinn.

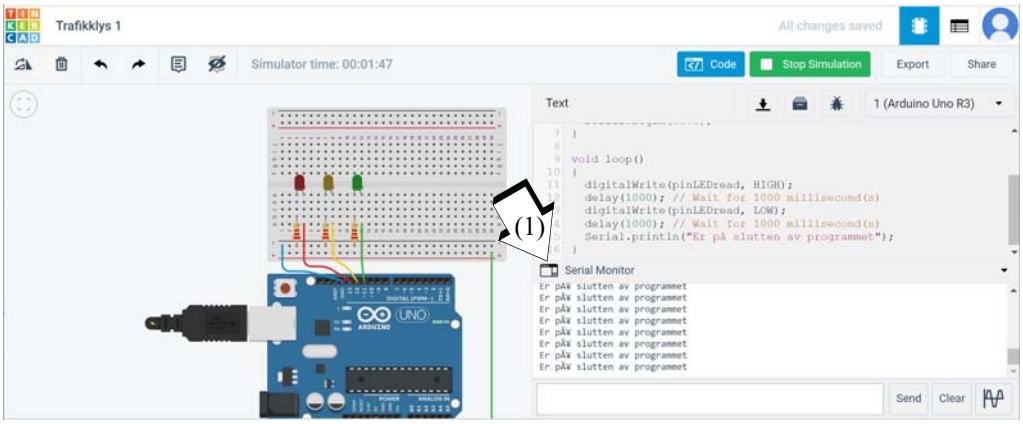
1. Denne tjenesten tilbys simulering:

2. Klikk på den nummeret til den programlinjen der du ønsker at programmet skal stoppe (1). Her er linje 14 valgt (det kan legges inn flere stopp-punkter).
3. Start simuleringen (2).
4. Kjør programmet med Kjør/Pause-knappen (3) og programmet kjører fram til stopppunktet. Ved å bevege muspila over en variabel i programmet, vil man kunne se verdien til variabelen.
5. Ved hjelp av symbolet kan man gå skrittvis gjennom programmet og studere endringer i variabler og hvilke veivalg programmet gjør.

3.4.2 Feilsøking ved hjelp av monitoren

En vanlig måte å feilsøke på er å skrive ut kommentarer til monitoren på ulike steder i koden. Når kommentaren framkommer i monitoren, vet man at denne delen av koden er blitt utført. Likedan

kan man skrive ut variabler på ulike steder i koden, for å studere hvordan variablene endrer seg mens man kjører koden. Også TinkerCAD har en monitor.



1. **Monitor:** Her ser vi monitoren nederst som er et vindu som åpnes ved å velge "Serial Monitor", se figuren til høyre. I eksempelet over har vi valgt å skrive ut teksten "*Er på slutten av programmet*", hver gang programmet kommer til den siste linja i loop()-funksjonen. Vi legger også merke til at monitoren ikke takler norsk tegnsett.

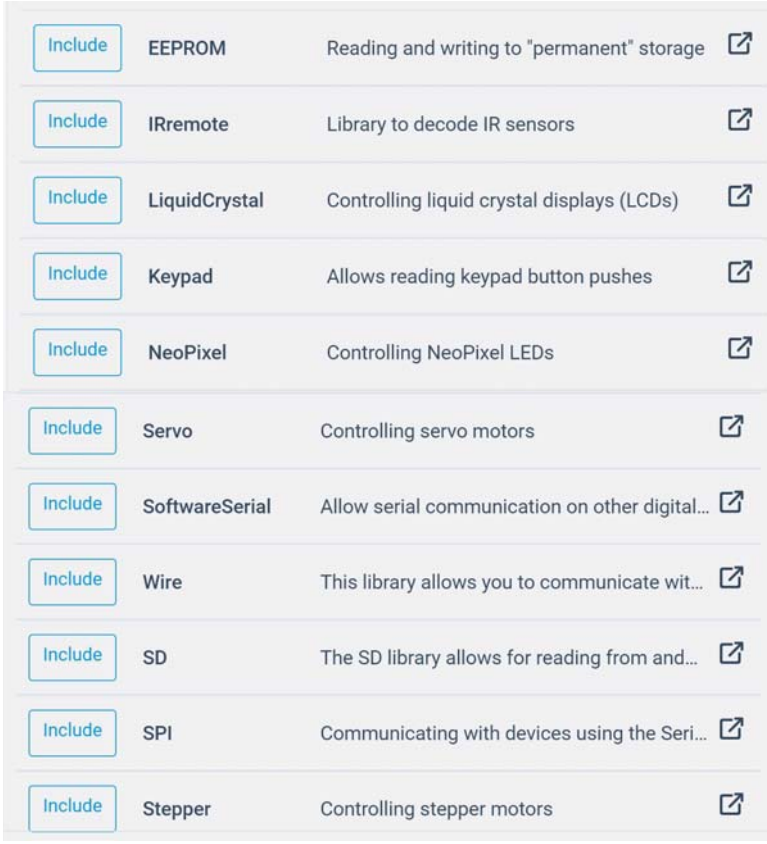
2. **Send kommandoer til mikrokontrolleren.** Under monitorvinduet er det en innboks hvor man kan skrive kommandoer som kan sendes til mikrokontrolleren når man trykker "Send" knappen. Kommandoen overføres da mens programmet kjører. Skal dette ha noe for seg må man skrive kode i programmet som leser kommandoer fra monitoren.

3. **Tøm monitor:** Velger man "Clear" tømmes monitorvinduet for alle tidligere utskrifter.
4. **Grafikk:** Velger man "Graph" vil man kunne lage grafiske kurver av variabler som skrives til monitoren.

3.5 Bruk av biblioteker

Enkelte kretser krever bruk av spesielle biblioteker. Normalt installeres disse i editoren og inkluderes i programmet. Også TinkerCAD tilbyr et utvalg av biblioteker. Ved å velge arkivskuff-symbolet i menylinja vist på figuren over til høyre, får man en oversikt over tilgjengelige biblioteker.

1. **Inkluder biblioteker:** Tabellen under viser utvalget av biblioteker som er inkludert i TinkerCAD. Bibliotekene er knyttet til utvalget av komponenter som ligger i komponentmenyen. Symbolet lengst til høyre for hver linje leder til bibliotekets nettside, med ytterligere beskrivelse. Ved å velge "Include" til venstre vil disse inkluderes i programmet.



2. **Lagre filen:** Programfila kan lastes ned og lagres ved å velge "Last ned"-tegnet til venstre for bibliotekstegnet. Programfila havner i katalogen nedlasting og må evt. flyttes derfra til ønsket katalog. Kretstegningen følger ikke med, men lagres på kontoen din hos TinkerCAD sammen med programfila.

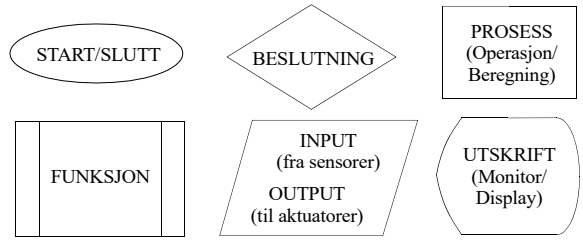
4 Bruk av flytdiagrammer[4]

For å planlegge eller dokumentere programmer kan man blant annet bruke flytdiagrammer for å tydeliggjøre flyten i programmet. Dersom dere ønsker å gjøre litt ekstra ut av flytdiagrammer så kan dere bruke spesiell programvare utviklet til dette formålet.

Et *flytdiagram* eller flytskjema viser strukturen i et dataprogram og er et nyttig verktøy så vel for å planlegge et program, dokumentere programmet og feilsøke i programmet. Flytdiagrammet hjelper oss med å forstå hva som skjer, når og hvordan beslutninger tas og hvilke handlinger som utføres av mikrokontrolleren.

Bruk gjerne litt tid til å tenke gjennom strukturen i programmet, å lage et flytdiagram kan være et nyttig hjelpemiddel for strukturering av programmet. Følgende blokker er ofte nok for å tegne et tilstrekkelig detaljert flytdiagram. Som vi ser har blokkene ulik form avhengig av funksjon:

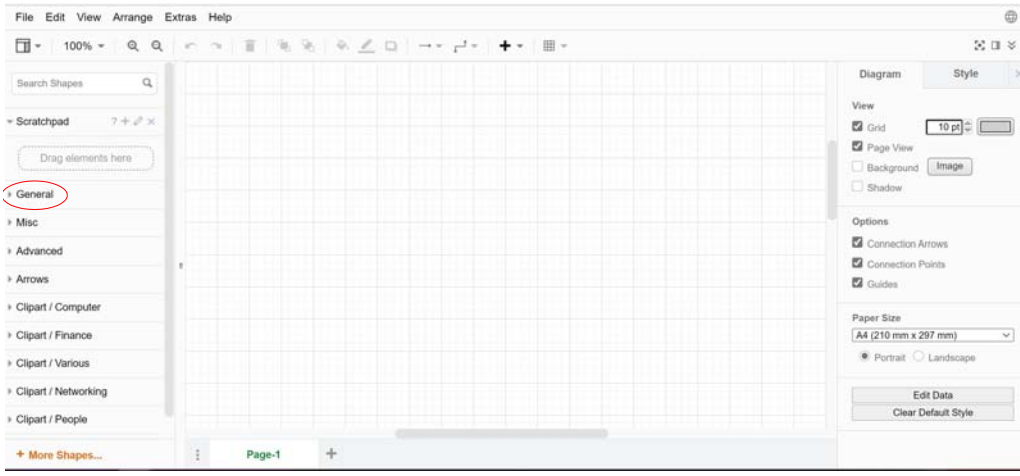
De ulike blokkene forbindes gjerne med linjer og piler for å indikere hvilken vei programmet beveger seg eller *flyter*. Ett slikt flytdiagram er ikke bare et nyttig hjelpemiddel under programmeringen, men også når man skal dokumentere programmet.



Vi skal bruke et gratisprogram som kan være til hjelp for å tegne gode flytdiagrammer:

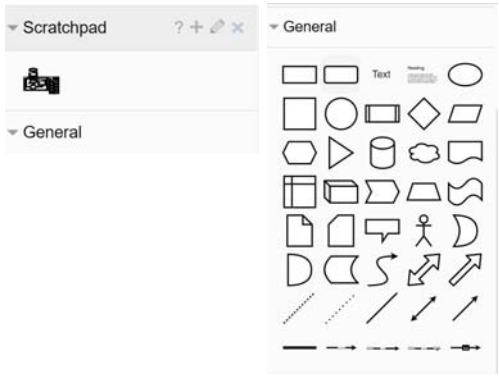
https://app.diagrams.net/

Når man trer inn i programmet blir man møtt med følgende grensesnitt:



4. Dette avsnittet bygger på det arbeidet Freddy Roger Moen utførte januar 2021 ved Levanger videregående skole – 2.0 Flytskjema.pdf

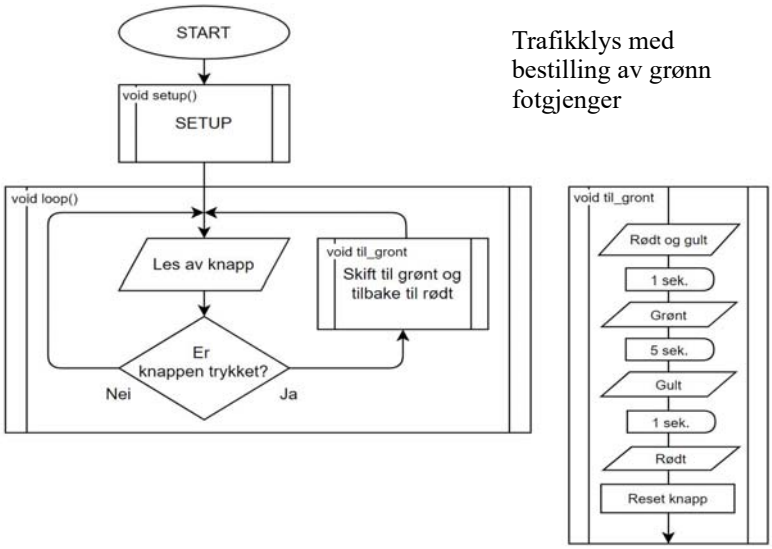
I menyen til venstre finner man flere samlinger av symboler, både for tegning av flytdiagrammer og annet. Velger man *General* så får man opp de vanligste symbolene brukt for tegning av flytdiagram som vist i figuren til høyre.



I menyfanen *Scratchpad* så kan man legge hele eller deler av flytdiagrammer slik at de kan brukes senere. Som det framgår av figur (A) til høyre så inneholder *Scratchpad* et lagret flytdiagram.

La oss se nærmere på et eksempel.

La oss ta utgangspunkt i flytdiagrammet under.



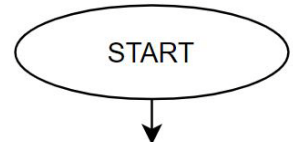
Trafikklys med bestilling av grønn fotgjenger

Vi har en fotgjengerovergang med trafikklys som inkluderer en knapp for bestilling av "grønn fotgjenger". Når ingen har trykket på knappen så er det grønt for bilene og rødt for fotgjengerne. Når knappen trykkes skifter det til grønt i 5 sek. for så å skifte tilbake til rødt igjen. Legg merke til at vi må resette *flagget* (Reset knapp) som leser av knappen, hvis vi har valgt å bruke et slikt flagg.

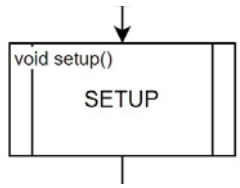
La oss se på de ulike symbolene:

START og STOPP – Ellipsen

Ellipsen eller et rektangel med sirkelbue i begge endrer brukes for indikere at her starter eller slutter programmet. Siden programmer i Arduino gjerne går til strømmen tas, så har vi bare en *START*-ellipse. For programmer som har en slutt, bruker man en tilsvarende ellipse med teksten *STOPP*.



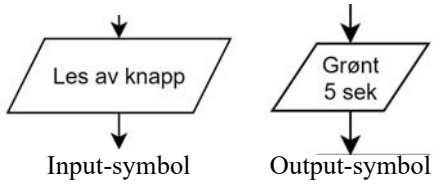
FUNKSJONER – Rektangel med sidekanter



Noen ganger er det nødvendig at et program har en eller flere tilleggsfunksjoner. Da benyttes rektangler med dobbel kant på sidene. Disse funksjonene eller prosessene vil ofte ha egne flytdiagram. Slike funksjoner kalles noen ganger for *sub-rutiner* eller *interrupt rutiner*. I forbindelse med Arduino-programmering kalles de *funksjoner* og avgrenses av to klammeparenteser.

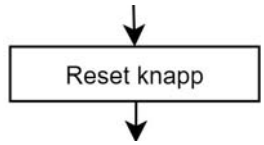
Vi vet at noe av det første mange Arduino-programmer gjør er å starte opp *void setup()*-funksjonen som utføres bare første gang programmet kjøres. Vi har valgt å sette navnet til funksjonen oppe i venstre hjørnet. *Void setup()*-funksjonen inneholder gjerne initiering av andre funksjoner som skal brukes i programmet, bestemme hvilke porter som skal være inn- og utganger og andre ting som skal være uforandret gjennom hele programmet.

INPUT og OUTPUT – Parallellogrammet



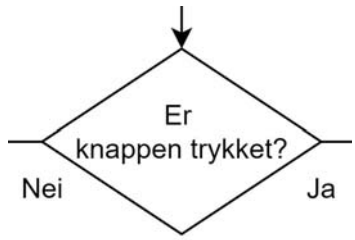
Mikrokontrollere kommuniserer ofte med omverdenen. I flytdiagrammet er dette angitt med et parallellogram. Det er symbolet for I/O (Input/ Output) kommunikasjon. I eksempelet vårt leser mikrokontrolleren av fotgjengerknappen som er en input-kommunikasjon, eller også slår på en eller flere lysdioder, som er output-kommunikasjon.

INTERN PROSESSERING – Rektangel



Rektangler beskriver interne operasjoner som ikke krever kommunikasjon med omverdenen. Dette kan f.eks. være matematiske beregninger eller tilbakestilling (resett) av flagg, som i vårt tilfelle, som må utføres før den går videre i programmet.

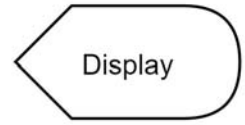
VEIVALG – Rombe



Å velge ulike veier i programmet på grunnlag av ulike *hendinger* (betingelser) er noe som gjør dataprogrammer til kraftige verktøy. Hvilke hendinger som kan inntreffe og hvilke betingelser som gjelder, må være kjent på forhånd. I eksempelet vårt er hendingen at fotgjengerknappen kan ha blitt trykket, og betingelsen er om den er trykket eller ikke. Dersom den ***ikke er trykket*** går den tilbake å tester om knappen er trykket. Dersom den ***er trykket*** så skifter trafikklyset til grønt i 5 sek. for deretter å gå tilbake til rødt igjen. Det er mulig å ha flere enn to hendinger eller betingelser.

UTSKRIFT/DISPLY – Avrundet rektangel med spiss

Ikke sjelden har vi bruk for å skrive ut resultatene av en datainnsamling, en måling eller beregning. Dette kan skje på ulike måter som foreksempel til et display. Da kan vi bruke symbolet vist på figuren til høyre. I programverktøyet vårt finner vi dette symbolet under menyfanen *Advanced*.



FORSINKELSE (DELAY) – Avrundet rektangel

I enkle programmer er det ikke uvanlig å legge inn forsinkelser, eller delay, i programmene. Dette kan være tilfelle i vårt trafikklys-eksempel hvor vi ønsker å beholde lysene på en viss tilstand.

5 Fritzing

5.1 Bruksområde

Fritzing er et tegneprogram for tegning av oppkoblinger på koblingsbrett. Programmet har et fyldig bibliotek med komponenter og kretskort fra bl.a. Arduino, Micro:bit og PIC. Programmet egner seg derfor glimrende for dokumentasjon og til å lage oppgaver som krever anvisning for oppkoblinger.

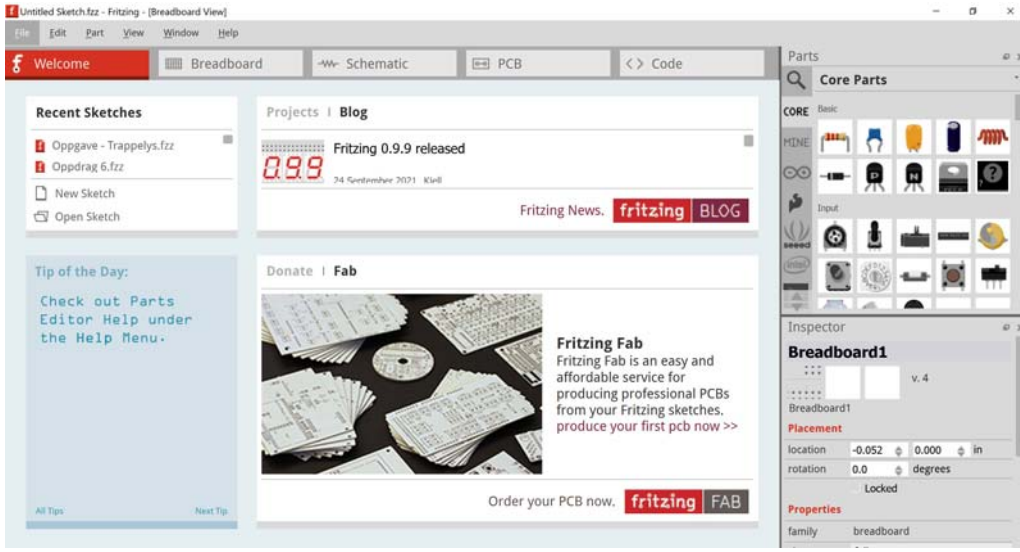
Når man har montert komponentene på koblingsbrettet, kan man konvertere tegningen til et koblingsskjema som viser komponentsymbolene, for deretter å legge ut som et kretskort. De seneste versjonene gir også mulighet for å skrive og utvikle programmer.

5.2 Installasjon

Programmet finnes i skrivende stund i versjon 0.9.9 (24. september 2021) som fritt kan lastes ned fra nettstedet: www.fritzing.org

5.3 Introduksjon til bruk

Når man starter Fritzing kommer følgende introduksjonsvindu opp:

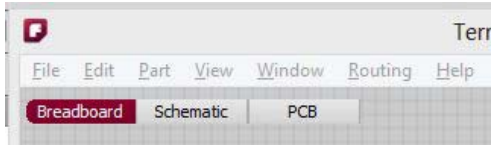


Deretter er det naturlig å velge *BreadBoard* som gir mulighet til å bygge opp en krets på et koblingsbrett som er et naturlig startpunkt for uttesting av kretser som ikke er spesielt kritiske mht. layout. Bruk av koblingsbrett er ikke nødvendigvis en god løsning for uttesting av kretser i RF-området (RF – Radio Frequency).

Velger man BreadBoard-fanen får man opp følgende skjermbilde:

Et standard koblingsbrett ligger klart til bruk i arbeidsfeltet til venstre. Til høyre finner vi *komponentbiblioteket*. Flere *spesialbiblioteker* kan hentes inn ved å trykke på ikonene til venstre for komponentbiblioteket. Ved hjelp av glidebryteren nederst kan en zoome inn og ut i arbeidsfeltet (hjulet på musa kan brukes på samme måte).

På menylinjen oppe til venstre finnes fire faner:

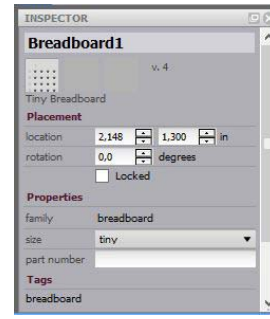


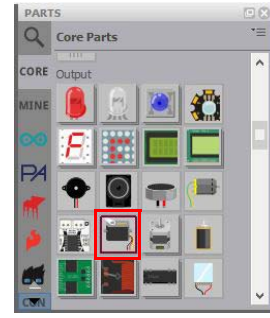
- ***Breadboard*** – Denne fanen gir deg mulighet til å konstruere kretser på koblingsbrett. Dette er et særdeles nyttig hjelpemiddel når du skal dokumentere dine konstruksjoner.
- ***Schematic*** – Ved å velge denne fanen overføres konstruksjon din til kretsskjema, Riktignok er programmet ganske primitivt slik at du selv må plassere og dreie komponentene slik at skjemaet ser greit ut.
- ***PCB*** – Ved å velge denne fanene overføres kretsskjemaet til "layout"-delen av programmet. Her kan du få lagt skjemaet ut som et kretskort og automatisk "rutet" kortet med kobberbaner. Enten på ensidig eller tosidige kretskort. Så kan en generere ulike filer som kan sendes inn for produksjon av kretskortet. Også her må en "hjelpe programmet" med å finne passende komponentplassering.

- ***Code*** – Her kan man skrive og teste ut kode for Arduino og PIC-prosessorer. Tanken er at man både kan skrive og laste ned programmene til mikrokontrollerkortet. Denne funksjonen er foreløpig bare tilgjengelig for PIC-prosessoren, for Arduino må man skrive og teste ut koden, men ikke laste ned fysisk til kretskortet (Rev. 0.9.9).

5.4 Oppbygging på koblingsbrett (Breadboard)[5]

Vi skal her beskrive oppbygging av en liten krets på koblingsbrettet trinn for trinn. Som eksempel velger vi å bygge opp roboten Abot som styres ved hjelp av en Arduino UNO. Sammenkoblingen av servoer og avstandssensoren bygges opp på et lite (tiny) koblingsbrett.



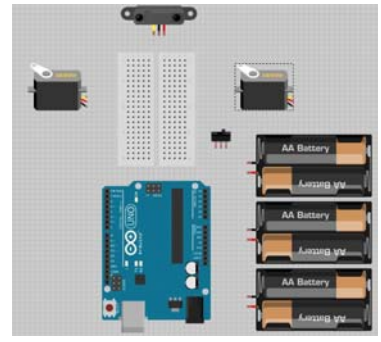


1. **Velg størrelse på koblingsbrettet**
   Klikk på koblingsbrettet som ligger i arbeidsfeltet slik at koblingsbrett menyen kommer opp i komponentfeltet til høyre. Velg *tiny* for størrelse (*size*). Grip brettet med cursoren og legge det mitt på arbeidsfeltet.

2. **Drei koblingsbrettet**
   Komponenter kan dreies 90° ved å høyreklikke på komponenten for så å velg *Rotate* og ønsket dreining og retning. Koblingsbrettet kan bare roteres et helt antall 90°. Man kan også bruke "Dreie" ikonet nede til venstre.

3. **Hent inn komponentene**
   Til vårt formål trengs to 360° servoer. En slik finnes i komponentmenyen, øverst til høyre, under kategorien *output*. Servoer har gjerne tre ledninger (en *gul*, en *rød* og en *sort*). Rød er +5V, sort er GND (0V), og gul er styreinngangen som bestemmer dreiehastigheten. Videre hentes Arduinoen inn og legges i arbeidsfeltet. En avstandssensor (IR proximity sensor) og en bryter (toggel switch) hentes også inn i arbeidsfeltet slik at resultatet blir omtrent som på figuren under til høyre.

4. **Batteri**
   Vi har her valgt å benytte et 9 V batteri. Siden Arduinoen og komponentene normalt benytter 5V, så har vi valgt å benyttet 5V regulatoren på Arduino-kortet til å senke spenningen fra 9V til 5V.



5. **Fest komponentene**
   Dersom en plages med ufrivillig flytting av komponenter. Høyreklikker man på komponenten og velger *Lock Part* i nedtrekksmenyen.

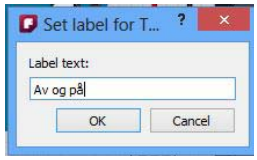
5. Deler av denne fremstillingen er hentet fra en elder utgave av Fritzing.

6. **Trekk forbindelseslinjer**
   Dernest trekkes forbindelser mellom terminalene til komponentene på følgende måte: Finn underkategorien *Breadboard view* i komponentmenyen og velg ikonet  . Forbind så to terminaler med hverandre ved å klikke på den ene og dra musa til den andre, mens venstreknappen holdes nede. Endepunktene skal feste seg til terminalene.

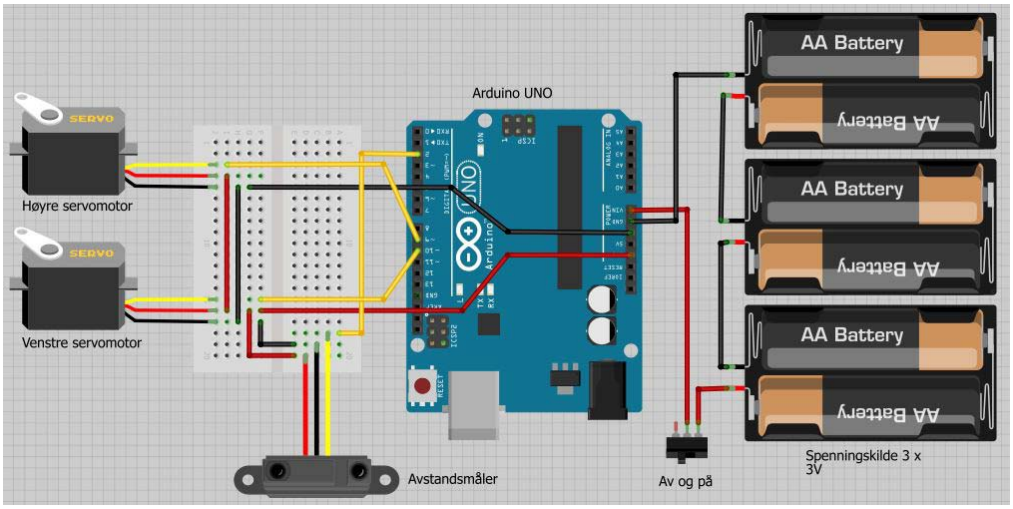
7. **Lag og fjern knekkpunkter på forbindelse**
   Dersom en ønsker en knekk på ledningen, klikker man på ledningen omtrent der en ønsker en knekk. Dra så i knekkpunktet til ønsket posisjon. Et knekkpunkt kan fjernes ved å høyreklikke på punktet, for så å velge *Remove bendpoint* i nedtrekksmenyen.

8. **Endre farge på forbindelseslinjene**
   Det kan være lurt å merke forbindelse med ulike farger. F.eks. rødt for + og sort for –. Dernest kan en velge fritt slik en finner det hensiktsmessig. Farge velges ved å høyreklikke på forbindelsen og så velge *Wire color* i nedtrekksmenyen.

9. **Legge til merkelapper til komponenter**
   Dersom man ønsker å sette en merkelapp på en eller flere av komponentene, høyreklikker man på komponentene og velger *Show part label*, dermed kommer det opp en default merkelapp. Ved å dobbelklikke på denne, kommer det opp en innboks (se figur til høyre) hvor man kan erstatte innholdet med det man måtte ønske.

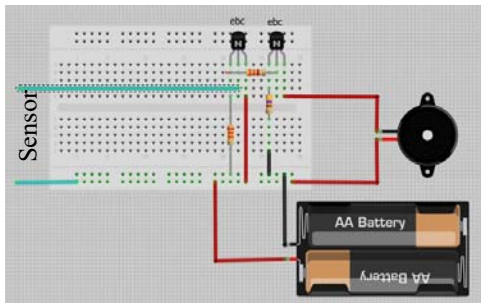


10. **Ferdig oppkobling**
    Figuren under viser den ferdige oppkoblingen som enten kan brukes for dokumentasjon, eller hjelp for elevene til å koble opp etter oppskrift om det er ønskelig.

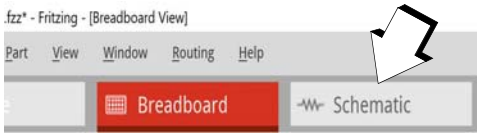


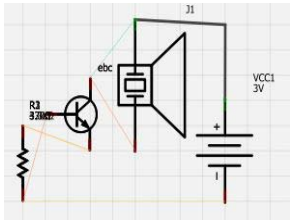
5.4.1 Overføring til koblingsskjema

Etter å koblet opp kretsen på koblingsbrettet og testet at den fungerer som ønsket, kan kretsen overføres til et standard koblingsskjema. Som et eksempel har vi valgt kretsen vist på figuren til høyre.



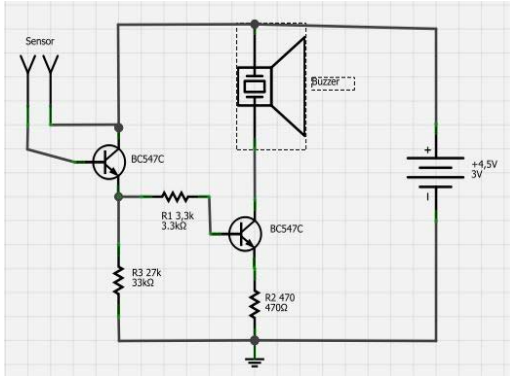
1. **Konvertering til koblingsskjema**
   Ved hjelp av menyen øverst i venstre hjørne kan oppkoblingen på koblingsbrettet konverteres til et koblingsskjema med symboler ved å velge *Schematic*.





2. **Justering av skjema layout**
   Den automatiske konverteringen gir ingen god layout som vist nederst på figuren til venstre. *De to transistorene og de tre motstandene er plassert oppå hverandre.* Derfor er antallet komponenter for lav. Skjemaet må derfor justeres for hånd. Dette gjøres ved å flytte komponentene slik de normalt vil være plassert i et koblingsskjema, med signalflyten fra venstre mot høyre. Flyttingen skjer ved å "gripe" komponenten ved å venstreklikke på komponenten, for så å dra den dit den skal.

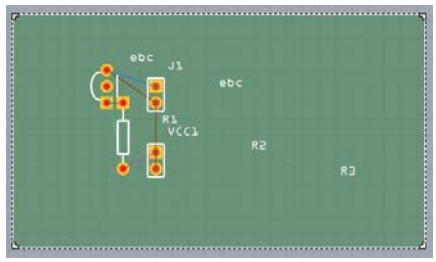
3. **Trekke opp forbindelsene på nytt**
   Etter at komponentene er plassert der de skal være, trekkes forbindelsene opp på nytt. Komponentene er forbundet med tynne streker som indikerer sammenkoblingene. Om noen forbindelser mangler, skyldes det at det er brudd i forbindelsen på koblingsbrettet. Om dette er tilfelle går en tilbake og retter opp forbindelsen. I det en trekker opp forbindelsene på nytt, linjene bli markert med en tykkere strek. Figuren til høyre viser resultatet.



4. **Tekst**
   Teksten kan endres ved å dobbelklikke på tekstrubrikken. Skriv ny tekst inn i innboksen og trykk OK. Tekstfeltet kan så flyttes til ønsket posisjon.

5.5 Trykte kretskort

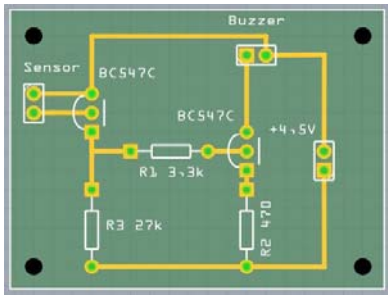
5.5.1 Fra skjema til PCB (Printed Circuit Board)



Når skjemaet er klart, kan dette rutes til et kretskort. Ved å trykke på PCB på menyen (øverst på figuren til høyre) konverteres skjemaet til en layout. Denne er svært uferdig og forutsetter at en manuelt plasserer komponentene slik at PCB-layout-en blir tilfredsstillende (nederst på figuren til høyre).

Vi legger merke til at begge transistorene legges på samme sted, tilsvarende med motstandene.

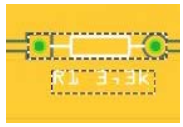
Layout



I utgangspunktet er forbindelseslinjene tynne rette linjer som knytter sammen beina til komponentene. Når komponentene er plassert kan en begynne å trekke de endelige ledningsforbindelsene. Idet man venstreklikker på en av linjeforbindelsene vil denne konverteres til en kobberbane med tilhørende loddeland. Knekkpunkter langs kobberbanen legges inn ved å klikke på banen. Ved å høyreklikke på et knekkpunkt kan en fra nedtrekksmenyen velge å fjerne punktet.

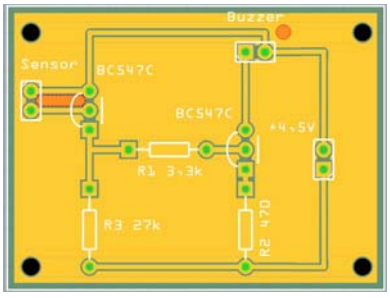
På et tosidig kort ligger de gule linjer på loddesiden av kretskortet, og de oransje linjene ligger på komponentsiden. Dersom en ønsker at alle linjene skal ligge på loddesiden høyreklikker man på kobberbanen og velger *Move to top layer.*

Tekst:



Ved å venstreklikke på komponenten markeres både komponent og tilhørende tekst. En må nå skrive inn ny tekst i tekstrammen samtidig som man kan gripe ramma og flytte den dit man ønsker.

Jordplan:



Det er ikke uvanlig at alle mellomrom mellom kobberbanene fylles med en kobberflate. Ved å forbinde denne til jord, vil man skape et robust jordplan som reduserer spredning av elektrisk støy. Et slikt jordplan legges inn i mellomrommene ved å dra ikonet  fra komponentkategorien *PCB view.*

6 Introduksjonsoppgaver

I dette er skissert oppgaver som egner seg for introduksjon til programmering av Arduino. Inntil videre har vi bare en slik oppgave:

- SOS morsesender

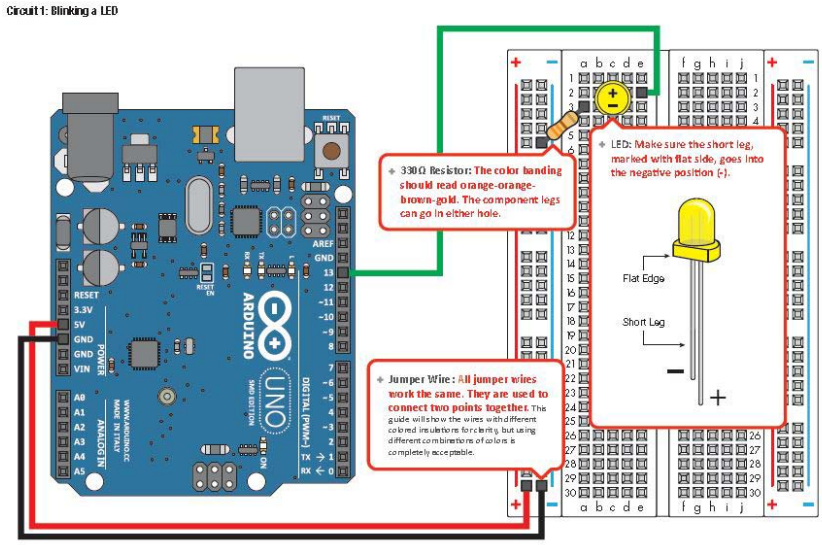
6.1 SOS morsesender

Dette er oppgaver for den som er helt ny med hensyn til programmering og vil egne seg som introduksjon for både lærere og elever.

6.1.1 Deloppdrag 1 – Blinkende lysdiode

Det skal bygges opp og programmeres en blinkende lysdiode.

Bygg følgende krets:



Skriv inn følgende programkode:

```
void setup()
{
    pinMode(13, OUTPUT);}
void loop()
{
   digitalWrite(13, HIGH);// Turn on the LED
   delay(1000); // Vent i et sekund
```

```
    digitalWrite(13, LOW);    // Slå av LED
    delay(1000);              // Vent i et sekund
}
```

Last opp kretsen og test at den fungerer.

Du skal nå utvide funksjonen til oppdrag 1 ved å lage en automatisk morsesender som sender SOS.

6.1.2 Deloppdrag 2 – Send SOS med lys

Vi skal nå lag et program som sender SOS med lyssignaler.

- *SOS som morse er: * * * - - - * * *
  Prikklengde skal være: 200 msek
  Streklengde skal være: 600 msek
  Mellomrom mellom streker og prikker: 200 msek
  Avstanden mellom bokstaver skal være: 600 msek
  Avstanden mellom ord skal være: 1800 msek*

6.1.3 Deloppdrag 3 – SOS med variabel hastighet

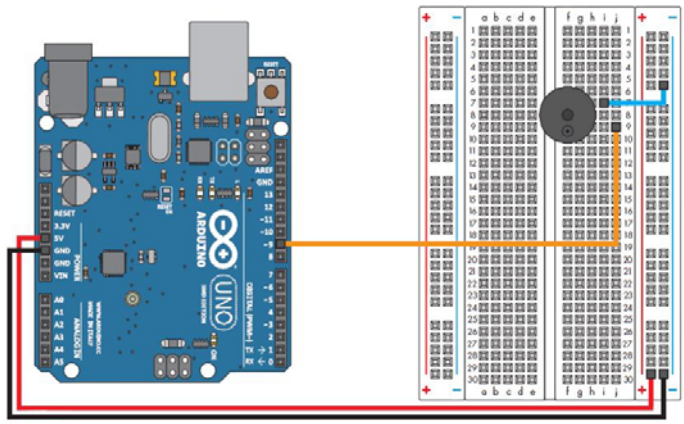
I dette deloppdraget skal vi bruke variabler for lett å endre sendehastigheten.

- *Det skal lages en SOS morsesender hvor det er lett å variere sendehastigheten*
  Den forholdsmessige lengden på prikker, streker og mellomrom skal være som angitt i deloppdrag 2.

6.1.4 Deloppdrag 4 – SOS med lys og lyd

I dette oppdraget skal vi både signalisere med lys og lyd.

- *Det skal lages en SOS morsesender som sender både lys og lyd.*



Koble opp en buzzer i tillegg til lysdioden.

6.1.5 Deloppdrag 5 – Bruk av funksjoner

SOS med lys og lyd og bruk av funksjoner

- *Morsesender som sender både lys og lyd, skal forenkles ved at det lages en funksjon for bokstaven S og bokstaven O. Disse to funksjonene kalles fra loopen.*

Løsningsforslagene finnes i vedlegg A.1, side 145.

Tips:

Bruk av funksjoner

En funksjon er en liten selvstendig programbit som kan kalles fra hovedprogrammet, en eller flere ganger etter som vi trenger den.

Bruk av funksjoner er utbredt og gir mange fordeler. Her vil vi kunne redusere koden ved å gjenbruke funksjonen som sender S istedet for å skrive samme koden to ganger. Den samme fordelen får vi ikke ved å lage en funksjon for O, men vi får en ryddigere og mer oversiktlig kode.

Slik kan vi skrive en enkel funksjon som sender morsetegnet for S:

```
void S()
{
   // Her skrives koden som generer morsetegnet for S
}
```

Vi gir funksjonen navnet S, siden den skal sende morstegnet S. Void forteller at denne funksjonen ikke skal returnere noen verdi. Selve funksjonen er omsluttet av klammeparentesene { }. Husk også de vanlige parentesene () etter funksjonsnavnet.

Når vi ønsker å sende bokstaven S kaller vi funksjonen på denne måten:

```
S();
```

Etter at programmet har utført funksjonen hopper det tilbake til hovedprogrammet og fortsetter der det slapp.

Les mer om funksjoner i avsnitt 2.4.12, side 42.

6.2 To blinkende lysdioder – Test øyets reaksjonsevne

Koble opp to lysdioder en rød og en gul. Koble dem til hver sin utgang på Arduino'en.

6.2.1 Deloppdrag 1 – Blinkende lysdioder

- *Lag et program som er slik at lysdiodene blinker annen hver gang. Når den gule lyser er den røde slukket og omvendt.*

6.2.2 Deloppdrag 2 – Test øyets reaksjonsevne[6]

- *Hvor raske diodeblink kan øyet oppfatte?*
- *For hvilke tidsverdier (grenseverdier) går lyset over til å se ut som sammenhengende lys?*
- *Ser det ut for deg som om ulike lysdioder, av samme farge eller av ulik farge, gir samme grenseverdier?*
- *Sammenlign dine resultater med andres resultater. Er de like? Hva forteller resultatene deg?*

Styrt framgangsmåte:

- *Bruk bare den røde lysdioden. La av- og på-tiden være like lange. Øk blinkhastigheten til det punktet da lysdioden synes å lyse kontinuerlig. Noter grenseverdien.*
- *Gjenta forsøket med den gule lysdioden og undersøk om grenseverdien for å se at den blinker, er forskjellig fra når du brukte rødt lys. Noter grenseverdien.*
- *Hva kan du gjøre for å redusere usikkerheten i målingen?*

Undersøk om grenseverdien varierer med:

- Kjønn
- Alder
- Farge
- Blink
- Forholdet mellom av- og påtid

6. Etter en ide av Ingeborg Ranøyen

7 Guidede oppgaver

7.1 Lag et trafikklys

I denne delen skal vi ta for oss de enkelte delprosjektene og bygge opp et lite system bit for bit til vi har fått det endelige produktet, som i denne sammenhengen er et *trafikklys.*

Du tenker kanskje at trafikklys er ganske enkle og noe vi alle er kjent med. Nettopp derfor kan det være en fin oppgave å begynne med. Dernest vil en oppdage et selv et så enkelt prosjekt kan vise seg å være mer utfordrende og komplekst enn man først tror.

La oss først definere totaloppdraget:

7.1.1 Prosjektets oppdrag

Oppdraget går ut på å lage et trafikklys, altså ikke først og fremst et lyskryss, men et enkelt trafikklys. Lyskrysset kan være en utvidelse på et senere tidspunkt.

> ***Oppdrag:*** *Det skal lages et trafikklys som kan skifte fra grønt til rødt via gult slik trafikklys vanligvis gjør. Dvs. at reglene for skifte mellom lysene skal være riktige. Dernest skal trafikklyset gi mulighet for å bestille grønt, det er jo ganske vanlig når det gjelder fotgjengere. Vi ønsker også å inkludere et varsel om at den grønne perioden nærmer seg slutten, noe som vanligvis skjer ved blinkende grønt lys. Universell utforming er også viktig, derfor vil vi gjerne inkludere lyd slik at synshemmede også kan ha nytte av trafikklyset. Til sist vil vi at trafikklyset skal gå over i blinkende gult når mørket faller på, dermed må vi bruke en lyssensor.*

Det er naturlig at et slikt relativt stort prosjekt deles opp i mindre delprosjekter som testes ut enkeltvis for så å knyttes sammen til det endelige produktet.

Under har vi delt opp spesifikasjonen i en rekke mindre deler:

Beskrivelse av deloppdragene

Deloppdrag 1 – Enkelt trafikklys:

- *Bestem rekkefølgen på lysene i et trafikklys når lyset skifter fra rødt til grønt og fra grønt til rødt.*
- *Programmer trafikklyset slik at: Det er rødt i 5 sek, gult i 1 sek og grønt i 5 sek.*
- *Koble opp trafikklyset med:*
  Rød, gul og grønn lysdiode og skriv programmet.

Deloppdrag 2 – Grønt lys på forespørsel:

- *Endre sekvensen i deloppdrag 1 slik at trafikklyset blir stående på rødt lys, helt til noen trykker på fotgjengerknappen.*
- *Etter 5 sek. skal lyset skifte til grønt på riktig måte, være grønt i 5 sek., før det igjen skifter tilbake til rødt på riktig måte, hvor det blir stående til det kommer et nytt trykk på fotgjengerknappen.*

- ***Nyttige opplysning:***
  - Bruk trykknappen som følger med settet eller som du finner i menyen til TinkerCAD

Deloppdrag 3 – Lag en funksjon av skiftet til grønt og tilbake:

- *Det skal lages en funksjon, gront_lys(), av skiftet til grønt og tilbake igjen. Når det bestilles grønt lys skal programmet hoppe til funksjonen gront_lys() utføre den, og så hoppe tilbake igjen.*

Deloppdrag 4 – Grønt lys på forespørsel med blinkende avslutning:

- *Endre programmet fra deloppdrag 3 slik at det blir grønt lys i 8 sek hvorav de siste 5 skal være blinkende. Hvert blink skal være 500 ms på og 500 ms av. Trafikklyset skal ellers oppføre seg som i deloppdrag 3.*
- ***Nyttige opplysning:***
  Endringen kan i sin helhet gjøres i funksjonen gront_lys().

Deloppdrag 5 – Blinkende grønt lys på forespørsel med lyd:

- *Hver gang det grønne lyset blinker, det vil si slås av eller slås på, skal det etterfølges av et kort pip på 100 ms.*
- *Lyden skal lages av en egen funksjon som genererer lydsignalet med en lengde som bestemmes idet funksjonen kalles.*
- ***Nyttige opplysning:***
  - Bruk Piezo buzzeren som følger med settet eller som du finner i menyen til TinkerCAD

Deloppdrag 6 – Blinkende gult lys etter mørkets frambrudd:

- *Når det blir mørkt skal lyset gå over i blinkende gult. Det skal da være 0,5 sek. på og 0,5 sek. av helt til det blir lyst igjen. Det skal være nok å legge hånda over den lysfølsomme motstanden for at det skal begynne å blinke, evt. endre på lysstyrken (glideren) i simulatoren i TinkerCAD.*
- ***Nyttige opplysninger:***
  - Koble opp fotomotstanden
  - Bruk spenningsdeler

Deloppdrag 7 – To trafikklys som fungerer sammen:

- *A) Tenk dere en fotgjengerovergang. Når en trykker på knappen på trafikklyset på den ene siden av gata så skal begge lysene virke samtidig. Dette skal fungere uansett fra hvilken side bestillingen kommer fra.*
- *B) Tenk dere at dere skal lage et lyskryss hvor det ene lyset er for fotgjengerovergangen og det andre for bilene som krysser forgjengerovergangen.*
- *Kan dere løse oppgavene ved enten å gå sammen to og to grupper, eller ved å inkludere et ekstra sett med lysdioder i den ene Arduino UNO'en dere har (dette er aktuelt når dere bruker TinkerCAD)?*

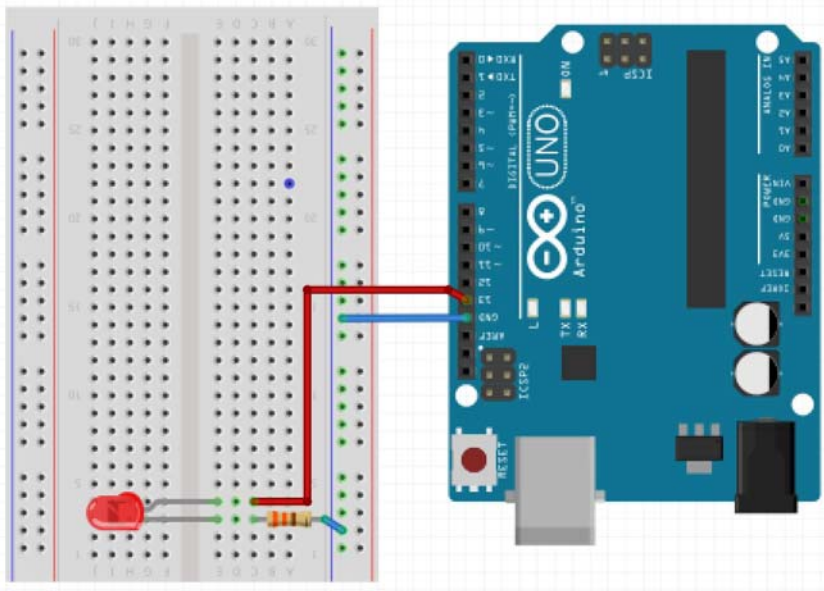
La oss begynne enkelt med en innledende øvelse: *Blinkende LED*.

7.1.2 Deloppdrag 0 – Blinkende LED

Dette er en innledende oppgave for den som er helt ny med hensyn til programmering og vil egne seg som introduksjon for både lærere og elever.

Det skal bygges opp og programmeres en lysdiode som blinker.

Oppbygging:
Figuren under viser oppbygging av blinkende LED**.**



For den som ønsker mer informasjon om LED, se avsnitt 9.1, side 122. For den som ønsker å lære mer om hvordan kretser kan bygges opp i TinkerCAD se avsnitt avsnitt 3 på side 45.

Programmering:

1. Skriv inn følgende programkode i program-editoren:

```
void setup()               // Setup() funksjonen kjøres bare en gang ved oppstart
{
   pinMode(13, OUTPUT);    // Port 13 defineres som utgang
}

void loop()
{
   digitalWrite(13, HIGH); // Slå på LED som er koblet til port 13
   delay(1000);            // Vent ett sekund (1000 ms)
```

```
  digitalWrite(13, LOW);  // Slå av LED som er koblet til port 13
  delay(1000);            // Vent ett sekund
}
```

2. *Last opp programmet og test at det fungerer.*

Se avsnitt 2.2 på side 28 for å lære hvordan du kompilerer og laster opp et program.

3. *Studer strukturen i programmet og betydningen av de ulike kommandoene*

Ser vi nøye etter så består programmet av to funksjoner avgrenset av {}

```
void setup()
{
... // Her skrives all kode som bare skal kjøres ved oppstart av programmet
}

void loop()
{
... // Her skrives kode som skal gjentas om og om igjen
}
```

Her er en kort forklaring til programlinjene i programmet:

Studer kommandoene nøye og vær sikker på at du forstår betydningen. Spør om nødvendig

```
//                      // Alt som står bak // på samme linje er kommentarer
pinMode(13, OUTPUT);    // Port nr. 13 programmeres til å være en utgang

delay(1000);            // Programmet stopper i 1000 millisekunder (1 sek.)
digitalWrite(13, HIGH); // Sett port 13 til 5V, slå på lyset
digitalWrite(13, LOW);  // Sett port 13 til 0V, slå av lyset
```

4. *Øk blinktakten*

Øk blinktakten til ett blink i sekundet.

Undersøk hva som er den høyeste blinkfrekvensen det er mulig å se med det blotte øye. Varierer den fra person til person? (Dette er det vanskelig å demonstrere i TinkerCAD)

5. *Skriv til monitoren*

Monitoren er et vindu som viser informasjon *som sendes fra programmet som kjører i Arduino'en* og tilbake til PC'en. Monitoren åpner du etter at programmet er lastet over i Arduino'en. Når du åpner monitoren restartes programmet.

Kommandoer for skriving til monitoren

For å kunne skrive informasjon tilbake til PC'en, må det legges inn skrivekommandoer i programmet. Legg inn følgende i setup()-funksjonen. Skrives inn mellom klammeparentesene:

```
Serial.begin(9600);     // Setter kommunikasjonshastigheten mellom Arduino og
                           PC til 9600 tegn i sek.
```

Du skal nå skrive "ON" til monitoren når lysdioden slås på og "OFF" når lysdioden slås av. Skrivekommandoen er slik:

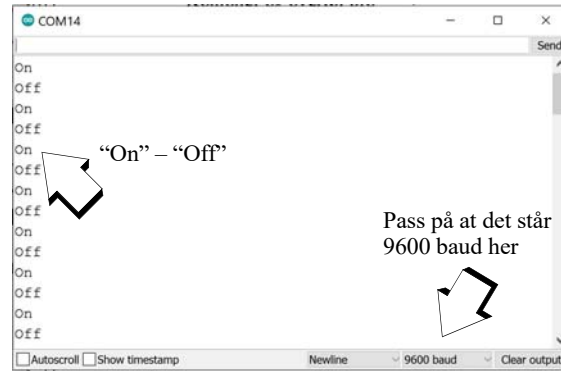
```
Serial.println("On");        // Skriver ON til monitoren
Serial.println("Off");       // Skriver OFF til monitoren
```

Kommandoene må legges inn i `void loop()`-funksjonen idet lyset slås på og når det slås av.

Kompiler og overfør programmet. Åpne monitoren ved å trykke på forstørrelsesglasset øverst i høyre hjørne av IDE-vinduet:

Monitoren skal da vise det dere har skrevet i Serial.print-setningene: "On" og "Off".

Pass på at datahastigheten som ble satt opp med kommandoen: `Serial.begin(9600);` stemmer overens med datahastigheten satt nederst i monitorvinduet. Se figuren til høyre.



START LØSNINGSFORSLAG-0

```
// Oppdrag 0 innledende øvelse - Blinkende lysdiode

void setup()                // Setup() funksjonen kjøres bare en gang ved oppstart
{
  pinMode(13, OUTPUT);    // Port 13 defineres som utgang
  Serial.begin(9600);     // Setter kommunikasjonshastigheten til 9600 tegn i sek.
}

void loop()
{
  digitalWrite(13, HIGH); // Slå på LED
  Serial.println("On");   // Skriv "On" monitoren
  delay(1000);            // Vent ett sekund
  digitalWrite(13, LOW);  // Slå av LED
  Serial.println("Off");  // Skriv "Off" monitoren
}
```

STOPP LØSNINGSFORSLAG

Finn feilen: Det er lagt inn en feil i løsningsforslaget over. Finn og rett opp feilen slik at programmet oppfører som det skal.

7.1.3 Deloppdrag 1 – Enkelt trafikklys

I denne oppgaven skal vi koble opp tre lysdioder, en rød, en gul og en grønn og programmere dem til å lyse slik et trafikklys oppfører seg når det går fra rødt, til grønt og tilbake til rødt igjen.

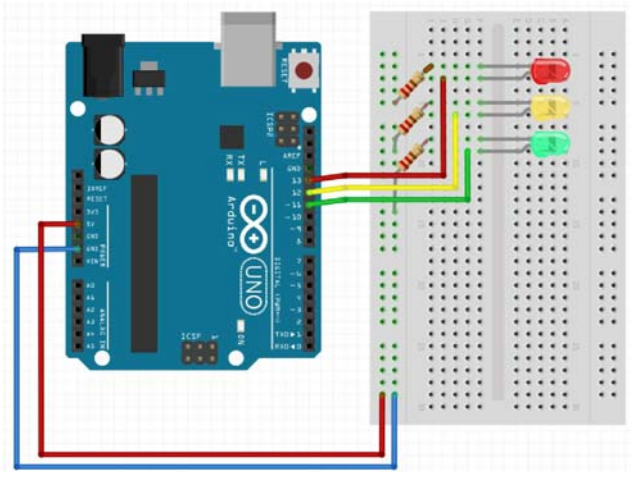
Lag et enkelt trafikklys:

- *Bestem rekkefølgen på lysene i et trafikklys når lyset skifter fra rødt til grønt og fra grønt til rødt.*
- *Koble opp trafikklyset med:*
  Rød, gul og grønn lysdiode og skriv programmet
- *Programmer trafikklyset slik at: Det er rødt i 5 sek, gult i 1 sek og grønt i 5 sek.*

Du kan nå ta bort skriving av "On" og "Off" til monitoren

Oppbygging

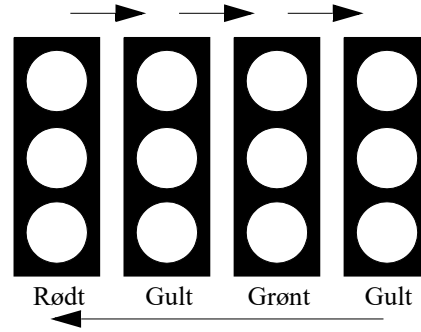
Figuren under viser oppbygging av vårt enkle trafikklys.



Det er lagt inn en seriemotstand på 220 Ohm for hver lysdiode for å begrense strømmen i diodene. Den røde lysdioden styres av port 13, den gule av port 12 og den grønne av port 11.

Programmering

1. **Bestem sekvensen**
   Det første man må gjøre er å bestemme hvor de ulike fargene er plassert og hvordan skiftet fra rødt til grønt og fra grønt til rødt foregår. Skriv eller fargelegg gjerne i figuren til høyre.
2. **Skriv programmet**
   Lag et program som skifter fra rødt til grønt og tilbake igjen på riktig måte



Bruk det dere har lært i Oppdrag 0. Dere kan sløyfe skriving til monitor.

Tips 1: Husk å definer alle tre portene (port 11, 12 og 13) som utganger ved hjelp av:
`pinMode(<pin nr.>, OUTPUT);`
Det skal gjøres i void setup() funksjonen.

Tips 2: Du kan bruke `delay(<forsinkelse i ms>);` til å holde et lys tent eller slukket i en viss tid.

3. Bruk variabler

I stedet for å bruke port nr. 11, 12 og 13 vil vi bruke meningsbærende navn som `ledPinGreen`, `ledPinYellow` og `ledPinRed`.

For å komme videre trenger vi kunnskap om *variabler*. Les om variabler i ramma under:

Bruk av variabler

En variabel kan betraktes som et oppbevaringssted for tall og bokstaver. Når vi skal lage en variabel må vi gi variabelen:

- ... et beskrivende *navn* som er lett å forstå
- ... en *type* f.eks. bare hele tall (int), desimaltall (float) eller bokstaver/tegn (char) m.fl.
- ... en *startverdi* (strengt tatt ikke nødvendig)
- ... *tilhørighet*, dvs. bestemme om den skal være *lokal* eller *global*

Her er noen eksempler med kommentarer:

```
int bestilling = 0;        // variabelen "bestilling" er av typen heltall som settes til
                           // null ved deklarering
float lengde = 1.23;       // Variabelen "lengde" er av typen desimaltall som gis
                           // verdien 1,23 ved deklarering
char minBokstav = 'n';     // Variabelen "minBokstav" er av typen "karakter" og
                           // gis karakteren "n" ved deklarering
```

Dersom man utelater oppstartsverdien så vil den sannsynligvis bli satt lik "0", men ikke sikkert.

Lokal og global variabel.

Lokal: En variabel blir lokal dersom den deklareres inne i en funksjon, da gjelder den kun for denne funksjonen.

Global: En variabel er global dersom den deklareres utenfor funksjonene, f.eks. i starten av programmet. Da vil verdiene til slike variabler være tilgjengelig i alle funksjonene.

Deklarer variablene for portene 13, 12 og 11 som heltallsvariabler (int) på følgende måte:

```
int ledPinGreen = 11;
int ledPinYellow = 12;
int ledPinRed = 13;
```

Vi deklarerer variablene *før* void setup()-funksjonen, dvs. variablene blir globale.

Fordelen med å definere portene som variabler er at om vi ønsker å endre porten som styrer det røde lyset, så trenger vi bare å endre programmet på ett sted og ikke alle steder der portnummeret er brukt.

4. **Test programmet**
   Test programmet om det fungerer som det skal.

Finn feilen i løsningsforslaget: Det er en feil i løsningsforslaget som gjør at trafikklyset ikke oppfører som spesifisert i oppdraget. Finn og rett opp feilen så løsningsforslaget oppfører som det skal.

START LØSNINGSFORSLAG-1

```
/*
Trafikklys Oppgave 1
Lag et trafikklys som lyser slik som et trafikklys skal lyse
når det skifter fra rødt til grønt og tilbake
*/
int ledPinRed = 13;    // Kontakt rød LED til pinne pin 13
int ledPinYellow = 12; // Kontakt rød LED til pinne pin 12
int ledPinGreen = 11;  // Kontakt rød LED til pinne pin 11

// I void sSetup definerer vi hva som er utganger og hva som er innganger
void setup() {
pinMode(ledPinRed, OUTPUT);
pinMode(ledPinYellow, OUTPUT);
pinMode(ledPinGreen, OUTPUT);
}

// loop() kjører om og om igjen i det uendelige
void loop()
{
digitalWrite(ledPinRed, HIGH);    // Slå på rød lysdiode
delay(5000);                      // Rød lysdiode skal være på i 5 sek.

digitalWrite(ledPinRed, LOW);     // Slå av rød lysdiode
digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode
delay(1000);                      // Gul lysdiode skal være på i ett sek.
```

```
digitalWrite(ledPinRed, LOW);     // Slå av rød lysdiode
digitalWrite(ledPinYellow, LOW);  // Slå av gul lysdiode
digitalWrite(ledPinGreen, HIGH);  // Slå på grønn lysdiode
delay(5000);                      // Grønn lysdiode skal være på i fem sek.

digitalWrite(ledPinGreen, LOW);   // Slå av grønn lysdiode
digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode
delay(1000);                      // Gul lysdiode skal være på i ett sek.

digitalWrite(ledPinYellow, LOW);  // Slå på gul lysdiode
}
```

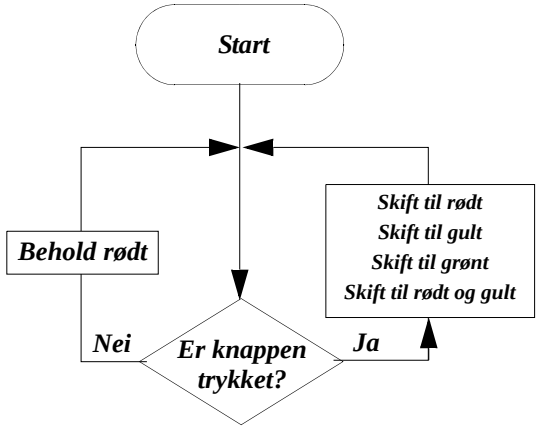
_____STOPP LØSNINGSFORSLAG_____

7.1.4 Deloppdrag 2 – Bestill grønt lys

I dette deloppdraget skal vi ta i bruk en trykkbryter for å få grønt lys. Dette er spesielt aktuelt ved fotgjengeroverganger. Men kan også være aktuelt for biltrafikken. Da vil bryteren skiftes ut med en sensor som detekterer metallet i bilen.

Deloppdrag 2 – Grønt lys på forespørsel:

- *Endre sekvensen i oppgave 1 slik at trafikklyset blir stående på rødt lys, helt til noen gir fotgjengerknappen ett kort trykk.*
- *Etter 5 sek. skal lyset skifte til grønt på riktig måte, være grønt i 5 sek., før det igjen skifter tilbake til rødt på riktig måte, hvor det blir stående til det kommer et nytt trykk på fotgjengerknappen.*

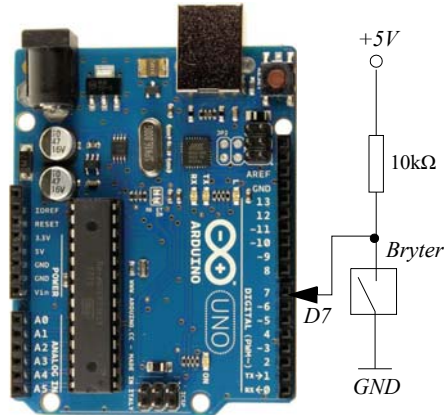


Nyttige opplysninger:

- *Bruk en av trykkbryterne som følger med settet, eller som du finner i menyen til TinkerCAD.*
- *Vi kobler opp bryteren slik at den gir +5V (logisk "1") når vi trykker på den, ellers 0V (logisk "0").*

Er du usikker på hvordan trykkbrytere fungerer og kan brukes i programmet, les om brytere i ramma under.

Bruk og avlesning av brytere



Som vi ser av flytdiagrammet så skal vi bruke trykk-bryteren til å si i fra til mikrokontrolleren at vi ønsker grønt lys.

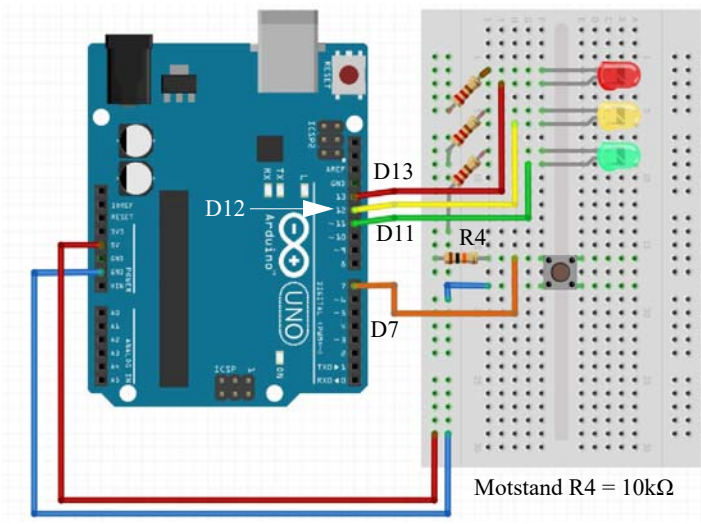
- Vi kobler bryteren opp til en av de digitale inn-gangene på Arduino'en som vist på figuren til høyre. Vi velger digital port nr. D7 (D for digital).
- Når bryteren *ikke* er trykket, så er port D7 koblet til +5V via en motstand. Dvs. at spenningen på D7 = +5V. Dette er også digitalt "1".
- Når bryteren er trykket, så er port D7 koblet til GND (jord). Dvs. at spenning på D7 er 0V. Dette er også digitalt "0".
- Fra programmet har vi mulighet til å lese av tilstanden til port D7 og se om den har digital verdi 0, dvs. bryteren er trykket, eller om den har digital verdi 1, dvs. at bryteren *ikke er* trykket.

  Vi bruker følgende kommando for å lese av porten der bryteren er tilkoblet og legger resultatet i variabelen `bestilling`:

```
bestilling = digitalRead(7); // Leser port 7 og legger resultatet i bestilling
```

Oppbygging

Figuren under viser oppbygging av trafikklyset med bryter for bestilling av grønt lys.



Velg motstanden R4 = 10kΩ (brun, sort, oransje)

Programmering

Den videre gangen i programmeringen kan være slik:

1. ***Definer porten*** som skal fungere som inngang, vi har foreslått port 7

   Tips: Bruk kommandoen:

   ```
   pinMode(7, INPUT);
   ```

   Denne definerer digital port 7 som en *inngang*. Kommandoen skal utføres bare en gang når programmet starter. Vi husker at vi på tilsvarende måte definerte portene 11, 12 og 13 som utganger.

2. ***Deklarer en variabel*** som holder verdien som er avlest fra bryteren

   Tips: Variabelen kan f.eks. defineres som heltall (int) og kan gjøres *global,* dvs. vi må deklarere den utenfor *`void setup()-`* og *`void loop-funksjonen.`* Sett verdien til 0 ved oppstart.

   ```
   int bestilling = 0;              // Definer variabelen "bestilling" som et heltall
                                    // og sett den til null
   ```

3. ***Les av verdien til bryteren*** ("1" trykket – "0" ikke trykket)
   Bruk følgende kommando for å lese av bryteren og legge resultatet i variabelen *bestilling*:

   ```
   bestilling = digitalRead(7);     // Leser av digital port 7 og legger resultatet i
                                    // variabelen bestilling
   ```

   Siden vi stadig må sjekke om bryteren er trykket inn så må vi legge avlesningen av bryteren i void loop()-funksjonen så den blir gjentatt i det uendelige.

Vi skal nå bruke den avleste verdien fra bryteren til å velge mellom å gi grønt lys eller beholde rødt lys. For å kunne utføre et betinget valg mellom ulike alternativer, må vi bli kjent med *if-kommandoen.*

If-kommandoen

På grunnlag av en eller flere betingelse skal det foretas ett av flere valg. Syntaksen for denne kommandoen er som vist under:

```
if (<betingelse 1>)
{
   // Skriv koden som skal utføres under betingelse 1 her
}
else if (<betingelse 2>)
{
  // Skriv koden som skal utføres under betingelse 2 her
}
```

```
else
{
   // Skriv koden som verken passer under betingelse 1 eller 2 her
}
```

Man kan legge inn så mange `else if` som man ønsker og man kan droppe `else` om man ønsker det.

Betingelsen skrives gjerne som en logisk sammenligning. I vårt trafikklysprosjekt kan betingelsen f.eks. være disse:

```
if (bestilling == 1)
{
    // Skriv kode for skifte til grønt og tilbake til rødt her
}
else if (bestilling == 0)
{
    // Skriv kode for å beholde rødt lys her
}
```

Se om du kan forenkle koden.

Det finnes mange ulike logiske operatorer:

==	Har to variabler av samme type like verdier?
!=	Har to variabler ulike verdier?
>	Er den ene større enn den andre?
<	Er den ene mindre enn den andre?
>=	Er den ene større eller lik den andre?
<=	Er den ene mindre eller lik den andre?

For mer informasjon om if()-setninger se avsnitt 2.4.10 på side 40. Eller se: https://www.arduino.cc/reference/en/language/structure/control-structure/else/

---

4. ***Bruk if-setningen.*** Avles verdien fra bryteren og bruk en if-setning til å bestemme om det fortsatt skal være rødt eller om det skal skifte til grønt.

   Tips: Husk og nullstill innholdet i variabelen *bestilling* slik at den er klar til å ta imot en ny bestilling. Hva vil skje om man glemmer og nullstille bryteren?

5. ***Skriv programmet*** og test om det virker.

Finn feilen i løsningsforslaget: Det er en feil i løsningsforslaget som gjør at trafikklyset ikke oppfører som spesifisert i oppdraget. Finn og rett opp feilen så programmet oppfører seg som det skal.

_____START LØSNINGSFORSLAG-2_____

```
/*
Trafikklys Oppgave 2
Lag et trafikklys som lyser slik som et trafikklys skal lyse
```

```
når det skifter fra rødt til grønt og tilbake.
Skifte skal imidlertid kun inntreffe ved trykk på fotgjengerknappen.
*/
int ledPinRed = 13; // Kontakt rød LED til pinne pin 13
int ledPinYellow = 12;      // Kontakt rød LED til pinne pin 12
int ledPinGreen = 11;       // Kontakt rød LED til pinne pin 11

int knappBestilling = 7;    // Kontakt knapp for bestilling av grønn mann
boolean bestilling = false; // Boolsk variabel som holde bestilling

// Setup definerer hva som er utganger og hva som er innganger
void setup() {
pinMode(ledPinRed, OUTPUT);
pinMode(ledPinYellow, OUTPUT);
pinMode(ledPinGreen, OUTPUT);
pinMode(knappBestilling, INPUT);
}

// loop() kjører om og om igjen i det uendelige
void loop()
{
bestilling = digitalRead(knappBestilling); //Les av knappens innstilling

if (bestilling = false)
 {
 digitalWrite(ledPinRed, HIGH);       // Slå på rød lysdiode
 delay(5000);

 digitalWrite(ledPinYellow, HIGH);    // Slå på gul lysdiode
 delay(1000);

 digitalWrite(ledPinRed, LOW);        // Slå av rød lysdiode
 digitalWrite(ledPinYellow, LOW);     // Slå av gul lysdiode
 digitalWrite(ledPinGreen, HIGH);     // Slå av grønn lysdiode
 delay(5000);

 digitalWrite(ledPinGreen, LOW);      // Slå av grønn lysdiode
 digitalWrite(ledPinYellow, HIGH);    // Slå på gul lysdiode
 delay(1000);

 digitalWrite(ledPinYellow, LOW);     // Slå av gul lysdiode
```

```
 bestilling = true; // Sett bestilling til en/true
 }
else
 {
 digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode
 }
}
```

STOPP LØSNINGSFORSLAG

7.1.5 Deloppdrag 3 – Lag en *funksjon* som skifter til grønt og tilbake

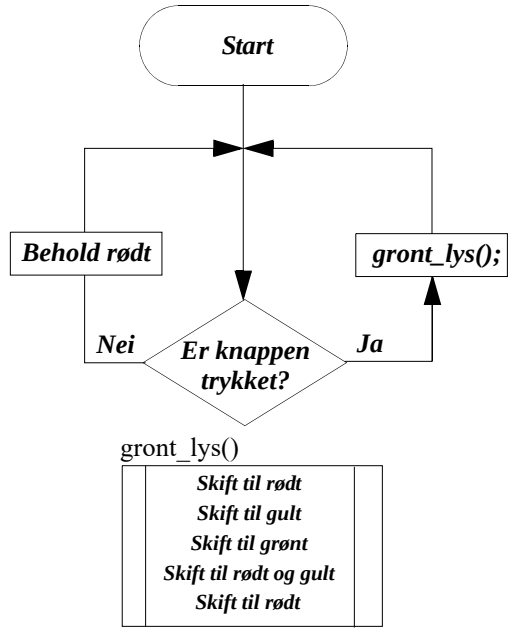
I dette deloppdraget skal vi ta i bruk en egendefinert funksjon. Dvs. at vi tar en del av programmet og lager en "pakke" (funksjon) av denne programdelen som vi kan hente inn i void loop()-funksjonen etter behov.

I dette deloppdraget skal lage en funksjon som skifter til grønt og tilbake:

- *Lag en funksjon som kalles gront_lys() og som skal inneholde skifte fra rødt til grønt og tilbake på riktig måte. Funksjonen kalles etter behov fra hovedprogrammet, void loop().*

Figuren til høyre viser at vi har lagt inn en blokk nederst i figuren med doble streker langs sidene. Dette er symbolet for en funksjon, og som gjør nettopp det vi ønsker.

I flytdiagrammet over ser vi at denne funksjonen kalles når knappen er trykket inn (gront_lys();)



Oppkobling

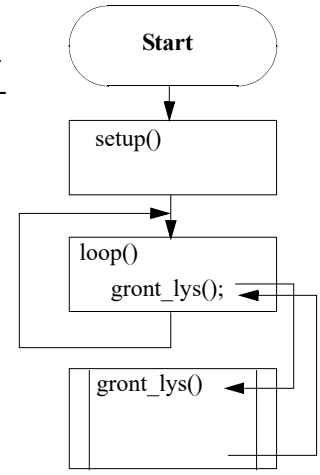
Det trengs ingen endring i oppkoblingen for dette deloppdraget.

Programmering

1. **Lag en funksjon** som kalles opp hver gang programmet skal skifte til grønt og tilbake igjen.

Egendefinerte funksjoner er svært nyttige for å strukturere og gjøre programmet lett å lese. I tillegg vil det kunne gjøre koden mer kompakt. I faktaboksen under lærer vi hvordan vi kan lage vår egen funksjon.

Definisjon av egne funksjoner



En funksjon er et lite program som kan kalles opp fra void setup()- eller void loop()-funksjonen, evt. andre funksjoner. Vi må gi funksjonen *et navn* og *en type*. Egendefinerte funksjoner skrives ofte nederst i programmet, dvs. under void loop(), men kan plasseres hvor som helst *utenfor* `void setup()` - og `void loop()` - funksjonene.

Under har vi vist hvordan vi kan definere funksjonen som vi har kalt gront_lys():

```
void gront_lys()
{
// Skriv her hva funksjonen skal gjøre
}
```

La oss se nærmere på de ulike kommandoene:

`void` betyr "tom" og betyr at funksjonen ikke returnerer noen verdi når den kalles. Dersom den skal returnere et heltall brukes typen `int` osv.

`gront_lys()` er det navnet vi har valgt å gi funksjonen

Hvordan tar vi så i bruk vår nylagde funksjon?

Vi kaller funksjonen fra `void loop()` -funksjonen på følgende måte:

```
void loop()
{
...
gront_lys();
...
}
```

Dersom vi trenger å overføre verdier til variabler som brukes i void loop() i funksjonen vår, så kan vi overføre verdiene i variabler ved å definere variablene som ***globale,*** dermed vil vi ha tilgang til dem i alle funksjonene som programmet består av. Dette er en enkel måte å gjøre det på som fungerer greit i små programmer, men som kan bli rotete og uoversiktlig når programmene øker i størrelse og kompleksitet. Vi skal senere se hvordan vi kan overføre variabler i argumentet til funksjonen.

---

2. **Legg inn sekvensen** av skifte fra rødt til grønt i funksjonen `gront_lys();`

   Flytt koden som endrer lyset fra rødt, til gult, til grønt og tilbake igjen fra void loop() og legg hele denne koden inn i funksjonen gront_lys();

3. **Legg inn kallet til funksjonen** gront_lys på ønsket sted i void loop()-funksjonen.

   Funksjonen gront_lys() kalles på denne måten:
   gront_lys();

4. **Test programmet** og se om det oppfører seg som forventet

Finn feilen i løsningsforslaget: Det er en feil i løsningsforslaget som gjør at trafikklyset ikke oppfører som spesifisert i oppdraget. Finn og rett opp feilen så programmet oppfører seg som det skal.

_____START LØSNINGSFORSLAG-3_____

```
/*
Trafikklys Oppgave 3
Lag et trafikklys som lyser slik som et trafikklys skal lyse
når det skifter fra rødt til grønt og tilbake.
Skifte skal imidlertid kun inntreffe ved trykk på knappen.
Det er laget en funksjon som utfører skiftet fra grønt til rødt og tilbake
*/

int ledPinRed = 13; // Kontakt rød LED til pinne pin 13
int ledPinYellow = 12; // Kontakt rød LED til pinne pin 12
int ledPinGreen = 11; // Kontakt rød LED til pinne pin 11

int knappBestilling = 7; // Kontakt knapp for bestilling av grønn mann
boolean bestilling = false; // Boolsk variabel som holde bestilling

// Setup definerer hva som er utganger og hva som er innganger

void setup() {
pinMode(ledPinRed, OUTPUT);
pinMode(ledPinYellow, OUTPUT);
pinMode(ledPinGreen, OUTPUT);
pinMode(knappBestilling, INPUT);
}

// loop() kjører om og om igjen i det unedelige

void loop()
{
bestilling = digitalRead(knappBestilling); //Les av knappens innstilling

if (bestilling == 0)
 {
   bestilling = 1;     // Sett bestilling til null/falsk
 }
else
```

```
 {
   digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode
 }
}

void gront_lys()                    // Definer funksjonen
{
  digitalWrite(ledPinRed, HIGH);    // Slå på rød lysdiode
  delay(5000);

  digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode
  delay(1000);

  digitalWrite(ledPinRed, LOW);     // Slå av rød lysdiode
  digitalWrite(ledPinYellow, LOW);  // Slå av gul lysdiode
  digitalWrite(ledPinGreen, HIGH);  // Slå av grønn lysdiode
  delay(5000);

  digitalWrite(ledPinGreen, LOW);   // Slå av grønn lysdiode
  digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode
  delay(1000);

  digitalWrite(ledPinYellow, LOW);  // Slå av gul lysdiode
}
```

_____STOPP LØSNINGSFORSLAG_____

7.1.6 Deloppdrag 4 – Blinkende grønt lys

I dette deloppdraget skal vi gi det grønne lyset en blinkende avslutning:

- *Endre programmet fra oppdrag 3 slik at det grønne lyset først har konstant grønt lys i 8 sek., deretter følger det 5 grønne blink. Hvert blink skal være 500 ms på og 500 ms av. Trafikklyset skal ellers oppføre seg som i deloppdrag 3.*
  Ellers skal det gule lyset være på i 1 sek, og det skal gå 5 sek. etter at knappen er trykket til det skifter fra rødt til gult.
- *Endre programmet slik at det er lett å øke lengden på grønt lys uten blink, lengden på hvert blink og antall blink.*
- ***Tips:** Bruk variabler*

Oppkobling

Det trengs *ingen* ekstra oppkobling for denne utvidelsen av programmet.

Programmering

For å løse oppdraget velger vi å bruke en *for-loop* som gjør det mulig å gjenta en del av programmet et visst antall ganger. La oss se nærmere på for-loopen.

---

for-loop'en

En for-loop egner seg spesielt godt til å gjenta deler av programmet et visst antall ganger, som f.eks. å få det grønne lyset til å blinke 5 ganger. Syntaksen for denne kommandoen er som vist under:

```
for (int i = 0; i < 5; i++)
{
   // Skriv koden som skal gjentas her
}
```

La oss se på betydningen av hvert av leddene i betingelsen til for-loop'en

`int i = 0;`	Deklarer en heltalls tellevariabel, vi har valgt å kalle den *i* og sette den til startverdien 0.
`i < 5;`	Sjekk om tellevariablene er nådd opp til ønsket antall. Her satt til 5. Siden vi begynner med $i = 0$ så blir det 5 ganger selv om *i* aldri når opp til 5.
`i++`	Øk tellevariabelen med 1 for hver gang koden i for-loop'en er gjennomløpt. Denne kan også skrives slik $i = i + 1$. Man kan også telle ned med kommandoen $i-- $ eller $i = i - 1$, om ønskelig.

Koden som ønskes gjentatt plasseres mellom klammeparentesene {}. Mer informasjon om for()-loopen finnes i avsnitt 2.4.9 på side 38.

---

1. ***Blinkende grønt.*** Legg inn en for-loop på riktig sted i koden for deloppdrag 2, slik at deloppdrag 4 er oppfylt.

   Tips: Det er ikke noe problem å plassere en for-loop inne i en if-setning.

2. ***Test koden*** og se om det ble som forventet.

Dersom man ønsker å endre lengden på den grønne perioden og evt. antall blink på slutten, er det bare å endre *delay* der det er nødvendig. Dette kan være problematisk dersom programmet er stort og det er mange plasser hvor verdiene må endres. Dette kan løses med å definere variabler for de ulike tidsforsinkelsene og antallene. Alternativt kan vi bruke *konstanter*.

La oss se litt nærmere på forskjellen mellom variabler og konstanter.

Konstanter og variabler

Forskjellen på en konstant og en variabel er: En variabelen kan endres underveis i programmet mens en konstant er uforanderlig. Konstanter kan kun endres av programmereren når programmet skrives. Syntaksen for å definere en konstant er slik:

```
const int ANTALLBLINK = 5;     // Antall blink
const int LENGDEBLINK = 1000; // Lengden på hvert blink inkl av og på
const int RODTID = 5000;       // Lengden av rød tid før skifte
const int GULTID = 1000;       // Lengden av gul tid
const int GRONNTID = 8000;     // Lengden av grønn tid uten blink
```

Også denne kommandoen kan brukes, men er ikke anbefalt:

```
#define ANTALLBLINK 5  // Legg merke til ingen "=" og ingen ";"
```

Det har vært tradisjon for å skrive konstanter med store bokstaver. Merk at norsk tegnsett er ikke å anbefale hverken for konstant- eller variabelnavn.

Fordelen med en konstant framfor en variabel er at den ikke tar plass i den delen av lageret i mikrokontrolleren som er satt av til variabler og data.

---

3. ***Definer konstanter*** evt. variabler og endre koden.
4. ***Test programmet*** og undersøk om det oppfører seg som ønsket

Finn feilen i løsningsforslaget: Det er en feil i løsningsforslaget som gjør at trafikklyset ikke oppfører som spesifisert i oppdraget. Finn og rett opp feilen så programmet oppfører seg som det skal.

START LØSNINGSFORSLAG-4

```
/*
Trafikklys Deloppdrag 4
Lag et trafikklys som lyser slik som et trafikklys skal lyse når det skifter fra
rødt til grønt og tilbake.
Skifte skal imidlertid kun inntreffe ved trykk på knappen. Fra nå av skal
det grønne lyset lyse konstant i 8 sek. for deretter å etterfølges av 5 blink
med et halvt sekund på og et halvt sekund av. De gule lysene skal være på i 1 sek,
og det skal gå 5 sek. etter at det er trykket på knappen før det skifter til gult
*/

int ledPinRed = 13;          // Kontakt rød LED til pinne pin 13
int ledPinYellow = 12;       // Kontakt rød LED til pinne pin 12
int ledPinGreen = 11;        // Kontakt rød LED til pinne pin 11
int knappBestilling = 7;     // Kontakt knapp for bestilling av grønn mann
boolean bestilling = false; // Boolsk variabel som holde bestilling

// Definisjon av konstanter
const int RODTID = 5000;        // 5 sekunder før det blir grønt
const int GULTID = 1000;        // 1 sekund gultid
const int ANTALLBLINK = 5;      // 5 blink, ett blink er på og av.
```

```
const int LENGDEBLINK = 1000;  // 1 sek, inkluderer halvparten av og alvparten på
const int GRONNTID = 8000;     // Lengden det skal være fast grønn før det begynner
å blinke.

// Setup definerer hva som er utganger og hva som er innganger

void setup() {
pinMode(ledPinRed, OUTPUT);
pinMode(ledPinYellow, OUTPUT);
pinMode(ledPinGreen, OUTPUT);
pinMode(knappBestilling, INPUT);
}

// loop() kjører om og om igjen i det undelige

void loop()
{

bestilling = digitalRead (knappBestilling); //Les av knappens innstilling

if (bestilling == 0)
 {
   gront_lys();        // Utfør endring til grønt lys og tilbake
   bestilling = 1;     // Sett bestilling til null/falsk
 }
 else
 {
   digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode
 }
}

void gront_lys()
{
  digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode
  delay(RODTID);

  digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode
  delay(GULTID);

  digitalWrite(ledPinRed, LOW); // Slå av rød lysdiode
  digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
  digitalWrite(ledPinGreen, HIGH); // Slå på grønn lysdiode
```

```
  delay(GRONNTID);

  for (int i = 0; i = ANTALLBLINK; i++)
   {
    digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
    delay(LENGDEBLINK/2);
    digitalWrite(ledPinGreen, HIGH); // Slå på grønn lysdiode
    delay(LENGDEBLINK/2);
  }

   digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
   digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode

   delay(GULTID);

   digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
}
```

_____STOPP LØSNINGSFORSLAG_____

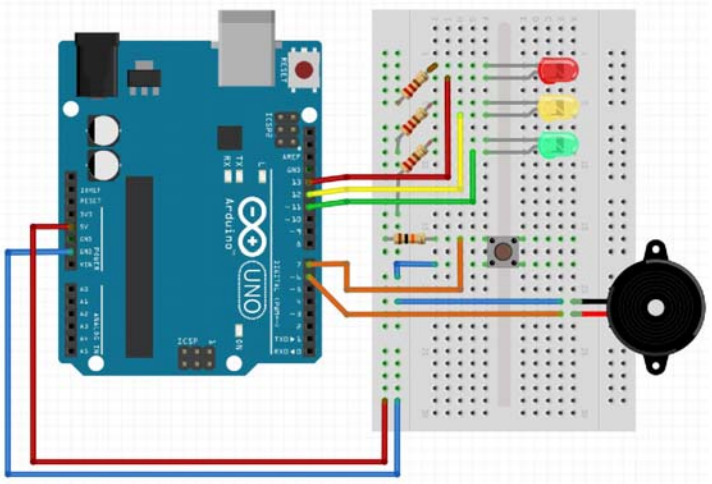
7.1.7 Deloppdrag 5 – Universell utforming

I dette oppdraget skal vi sette lyd til trafikklyset slik at også synshemmede kan høre når det er grønt. Vi nøyer oss i første omgang med å sette lyd til de grønne blinkene.

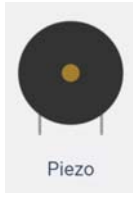
- *Hver gang det grønne lyset blinker, det vil si slås av eller slås på, skal det etterfølges av et kort pip på 100 ms.*
- *Lyden skal lages av en egen funksjon som genererer lydsignalet med en lengde som bestemmes idet funksjonen kalles.*
- *Definer gjerne en konstant som holder lengden på lydpulsen.*

Oppkobling

Figuren under viser oppbygging av trafikklyset med bryter for å bestille grønt lys og bruk av passiv buzzer for å lage lyd. Passive buzzere kan være av ulike typer. På figuren under er vist både den som finnes i Sparkfun Inventors kit og den som TinkerCAD tilbyr. I figuren har vi brukt den varianten som er tilgjengelig i programmet Fritzing.



Sparkfun Inventors kit



TinkerCAD

Vi har valgt **port 6** som utgang for å drive den ***passive buzzeren.***

La oss se litt nærmere på buzzere.

Passive og aktive buzzere

I denne sammenhengen skal vi nøye oss med å se på forskjellen mellom aktive og passive buzzere. Begge kan lage lyd.

En aktiv buzzer er en buzzer som har en innebygget oscillator som gjør at det er tilstrekkelig å koble til en spenning på noen Volt (her 5V) for at den skal avgi lyd. En aktiv buzzer egner seg derfor ikke til å lage melodier siden den kun gir én tone, den tonen den er laget for.

Programmering ved bruk av aktiv buzzer: Denne kan normalt kobles mellom en digital port og jord (GND) på Arduino'en. Lyden slås på ved å sette porten høy, dvs. til 5V.

```
digitalWrite(buzzerPin, HIGH); // Slå på lyd
digitalWrite(buzzerPin, LOW);  // Slå av lyd
```

Sjekk at utgangen hos Arduino'en kan levere nok strøm og spenning til buzzeren.

En passiv buzzer er som en liten høyttaler som omdanner en varierende spenning til en lyd. En slik buzzer vil kun gi et klikk dersom vi påfører en statisk spenning. Skal vi få lyd må derfor mikrokontrolleren lage den oscillerende spenningen som den passive buzzeren trenger. En passiv buzzer egner seg derfor til å lage melodier siden mikrokontrolleren har full kontroll

over tonehøyden (frekvensen).

Programmering: Denne kan normalt kobles til en digital port på Arduino'en som tilbyr PWM (puls breddemodulasjon) merket ~, siden det kreves at det sendes ut en varierende spenning som genereres i mikrokontrolleren. Til dette kan vi bruke følgende funksjoner:

```
tone(lydPin, frekvens);  // Slår på tonen på port lydPin
noTone(lydPin);          // Slår av tonen på port lydPin
```

Funksjonen `tone()` slår på lyden, mens `noTone()` slår av lyden. `lydPin` er porten som piezo-høyttaleren (passiv buzzer) er tilkoblet og `frekvens` er tonehøyden.

Man kan også velge å legge inn varigheten til tonen:

```
tone(lydPin, frekvens, varighet);
```

`varighet` angir lengden på tonen i millisekunder. Dette kan være hensiktsmessig når man ønsker å genere et kort lydstøt som i vårt oppdrag, men sjelden dersom man ønsker å lage musikk.

Programmering

1. **Velg en port for å levere lyd** til den passive buzzeren. Vi har foreslått port 6.
2. **Definer porten som OUTPUT** i `void setup()`-funksjonen.
3. **Lag korte lydsignaler** i takt med at det grønne lyset blinker. Bruk kommandoene som er omtalt over.
4. **Velg frekvens** på lyden og varighet på lydpulsen.
5. **Lag en funksjon** som kalles opp hver gang programmet skal genere et lydsignal.

Som vi har sett er egendefinerte funksjoner et svært nyttig hjelpemiddel for å strukturere og gjøre programmet lett å lese. Her virker det litt unødvendig å lage en egen funksjon for å gi et lydstøt, men illustrerer hvordan vi kan overføre lydpulsens lengde som en del av argumentet når vi kaller funksjonen. La oss se hvordan vi kan overføre variabler i argumentet når vi kaller funksjonen.

Definisjon av egne funksjoner med overføring av variabler i argumentet

Gå tilbake til deloppdrag 4 og repeter hva egendefinerte funksjoner er og hvordan de fungerer.

Her skal vi vise hvordan vi kan overføre en variabel, i dette tilfellet lengden av lydpulsen som argument i funksjonen.

Under har vi vist hvordan vi kan definere funksjonen som vi har kalt Lyd():

```
void Lyd(int varighet)
{
  tone(lydPin, frekvens, varighet);
}
```

Vi husker at de ulike kommandoene betyr følgende:

`void`	forteller oss at funksjonen ikke returnerer noen verdi
`Lyd ( )`	er det navnet vi har valgt å gi funksjonen
`(int varighet)`	definerer en variabel av typen heltall som skal overføre lengden til tonen når vi kaller funksjonen. Den er et heltall gitt i millisekunder
`lydPin`	angi hvilken port som er koblet til den passive buzzeren. Kan gjerne defineres om en global heltallsvariabel.
`frekvens`	angir frekvensen til lydsignalet. Kan gjerne defineres om en global heltallsvariabel

Vi kaller funksjonen inne fra loop()-funksjonen som sist, men må passe på å kalle funksjonen med lengden av lyden i argumentet:

```
loop()
{
int toneLengde = 100; // Tonelengden er 100 msek)
...
Lyd(toneLengde);      // Kaller funksjonen toneLengde()
...
}
```

Vi legger merke til at tonens lengde holdes av heltallsvariabelen `toneLengde` som settes til 100 msek. Så kalles funksjonen vi har laget med navnet `Lyd(toneLengde)` med parameteren `toneLengde` som argument. Innholdet i variabelen `toneLengde` overføres til den lokale variabelen `varighet` og brukes i funksjonen `Lyd()`. Når `Lyd()` er utført hopper programmet tilbake til stedet der den ble kalt og fortsetter derfra.

Legg merke til at det er naturlig å kalle funksjonen `Lyd()` fra innsiden av funksjonen `gront_lys()`.

---

6. **Lag funksjonen** og legg kallet inn i programmet på riktig plass
7. **Test programmet** og sjekk at det fungerer som tiltenkt

Finn feilen i løsningsforslaget: Det viser seg at når programmet kjører så høres ingen lyd til tross for at funksjonen `void Lyd()` er laget og ligger i programmet. Det viser seg at man har glemt å kalle funksjonen der lyden skal inn. Legg inn kallet på riktig(e) sted(er) og få programmet til å fungere som ønsket.

START LØSNINGSFORSLAG-5

```
/*
Trafikklys Deloppdrag 5
Lag et trafikklys som lyser slik som et trafikklys skal lyse når det skifter fra
rødt til grønt og tilbake.
Skiftet skal imidlertid kun inntreffe ved trykk på knappen. Fra nå av skal
```

```
det grønne lyset lyse konstant i 8 sek. for deretter å etterfølges av 5 blink
med et halvt sekund på og et halvt sekund av. De gule lysene skal være på i 1 sek,
og det skal gå 5 sek. etter at det er trykket på knappen før det skifter til grønt.
Det skal gis et kort lydpuls hver gang det blinkende grønne lyset skifter
*/

int ledPinRed = 13;    // Kontakt rød LED til pinne pin 13
int ledPinYellow = 12; // Kontakt rød LED til pinne pin 12
int ledPinGreen = 11;  // Kontakt rød LED til pinne pin 11
int tonePin = 6;       // Pinnen der den passive buzzeren er tilkoblet
int knappBestilling = 7; // Kontakt knapp for bestilling av grønn mann
boolean bestilling = false; // Boolsk variabel som holde bestilling

// Definisjon av konstanter

  const int RODTID = 5000;       // 5 sekunder før det blir grønt
  const int GULTID = 1000;       // 1 sekund gultid
  const int ANTALLBLINK = 5;     // 5 blink, ett blink er på og av.
  const int LENGDEBLINK = 1000;  // 1 sek, inkluderer halvparten av og alvparten på
  const int GRONNTID = 8000;     // Lengden det skal være fast grønn før det
begynner å blinke.
  const int KORTLYDPULS = 100;   // Kort lydpuls er satt til 100 msek.

// Setup definerer hva som er utganger og hva som er innganger

void setup() {
pinMode(ledPinRed, OUTPUT);
pinMode(ledPinYellow, OUTPUT);
pinMode(ledPinGreen, OUTPUT);
pinMode(tonePin, OUTPUT);
pinMode(knappBestilling, INPUT);
}

// loop() kjører om og om igjen i det undelige

void loop()
{

bestilling = digitalRead (knappBestilling); //Les av knappens innstilling

if (bestilling == false)
 {
```

```
   gront_lys();        // Utfør endring til grønt lys og tilbake
   bestilling = true; // Sett bestilling til null/falsk
 }
 else
 {
   digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode
 }
}

void gront_lys()
{
  digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode
  delay(RODTID);

  digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode
  delay(GULTID);

  digitalWrite(ledPinRed, LOW); // Slå av rød lysdiode
  digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
  digitalWrite(ledPinGreen, HIGH); // Slå på grønn lysdiode
  delay(GRONNTID);

  for (int i = 0; i < ANTALLBLINK; i++)
   {
     digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
     delay(LENGDEBLINK/2);
     digitalWrite(ledPinGreen, HIGH);
     delay(LENGDEBLINK/2);
  }

   digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
   digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode
   delay(GULTID);

   digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
}

void Lyd(int varighet)
 {
  tone(tonePin, 1500, varighet);
// Angir pinnenummer for piezoelektrisk lydgiver, frekvens og varighet i msek
```

```
}
```

STOPP LØSNINGSFORSLAG

Forslag til utvidelse

- **Deloppdrag 5.1:** *Bruk den samme lydfunksjonen til å legge inn lange lydstøt mens det grønne lyset ikke blinker, og korte når lyset blinker.*

7.1.8 Deloppdrag 6 – Blinkende gult lys

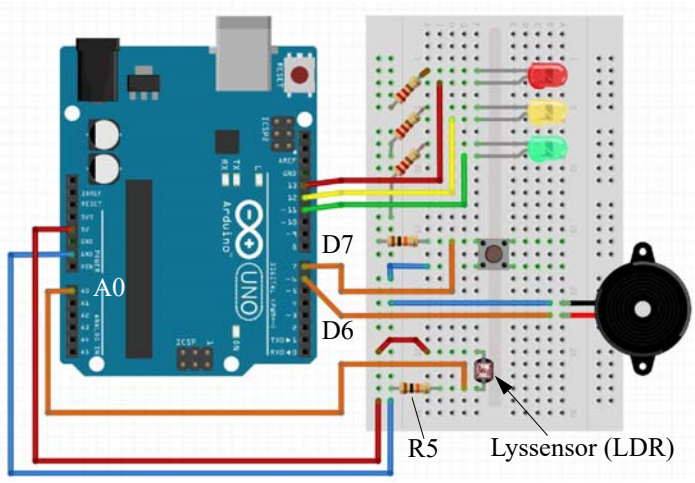
I dette oppdraget skal vi bruke en lysfølsom motstand, en LDR, til å styre funksjonen til lyset slik at det starter å blinke gult etter mørkets frambrudd.

- *Når det blir mørkt skal trafikklyset gå over i blinkende gult. Det skal da være 0,5 sek. på og 0,5 sek. av helt til det blir lyst igjen. Det skal være nok å legge hånda over den lysfølsomme motstanden for at det skal begynne å blinke. Alternativt justere glideren i TinkerCAD til halv lysstyrke.*
- *Lag blinkende gult (`gult_blinkende();`) som en funksjon som kalles fra void loop()-funksjonen*

Bruker man TinkerCAD så justerer man glideren over lyssensoren for å endre belysningen.

Oppkobling

Figuren under viser oppbygging av trafikklyset med bryter for å bestille grønt lys (digital port 7), passiv buzzer for å lage lyd (digital port 6) og lyssensor for å gi blinkende gult lys ved mørkets frambrudd (analog inngang A0). Bruker du TinkerCAD, velg motstand R5 = 1kOhm (brun, sort, rød). Bruker du en LDR fra Arduino Starter kit eller Sparkfun Inventors kit, bør de gå opp i verdi, f.eks. R5 = 10kOhm (brun, sort, oransje).



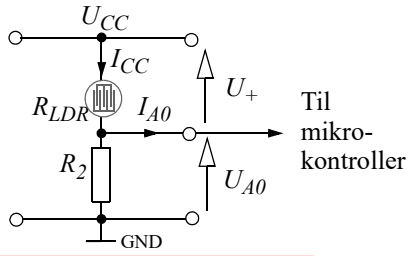
Lysstyrken påvirker resistansen i lyssensoren (LDR). For å omdanne en endring i resistans til en endring i spenning, benyttes en *spenningsdeler*. La oss se hvordan spenningsdeleren fungerer.

Spenningsdeleren – Fra endring i resistans til endring i spenning

En spenningsdeler er nyttig for å konvertere en variasjon i resistive sensorer, f.eks. i en LDR, til en variasjon i spenning som mikrokontrolleren kan registrere på en av sine analoge innganger.

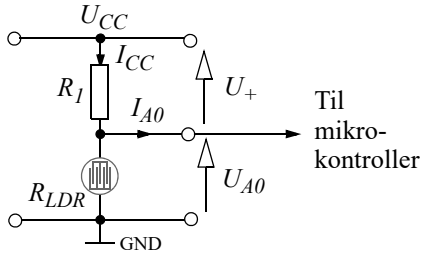
Ofte holder det at vi forstår hvordan spenningsdeleren prinsipielt fungerer siden vi ikke trenger å kjenne de eksakte verdiene for strømmer og spenninger.

A)



Økende lysstyrke –> Økende U_{A0}

B)



Økende lysstyrke –> Redusert U_{A0}

Som det framgår av figuren over så har vi to mulige måter å koble opp den lysfølsomme motstanden på avhengig av om vi vil at spenningen skal øke med økende lysstyrke (A) eller minke med økende lystyrke (B).

Her er det opp til oss å velge hva vi ønsker. Vi har valgt løsning A.

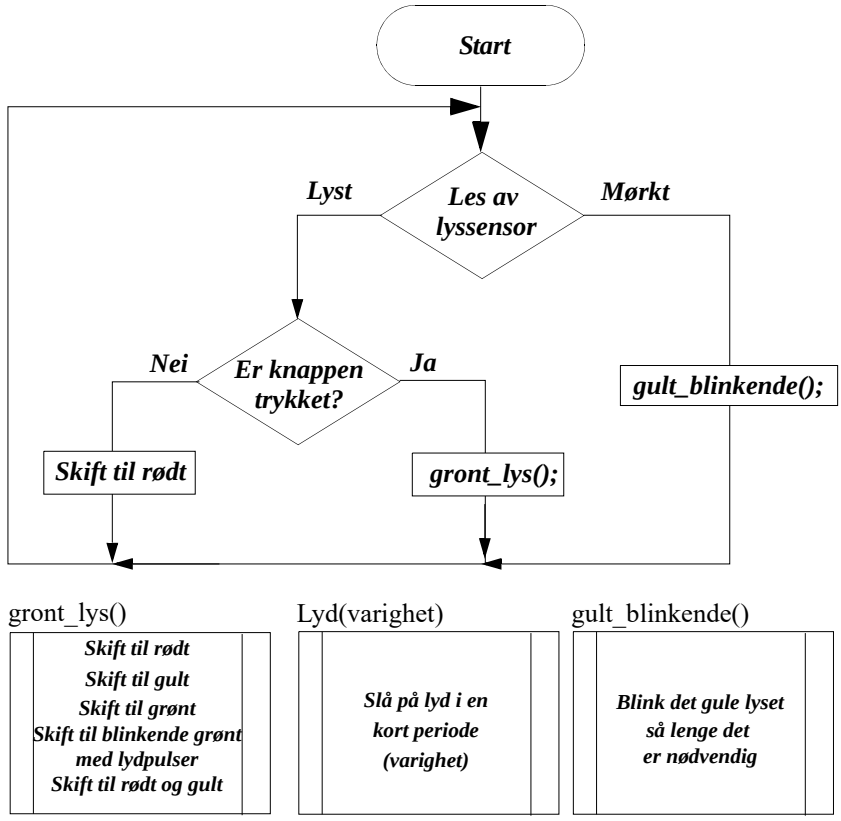
Ser vi på tilfelle A) og tenker oss at mørket faller på, dvs. at belysningen går fra lys til mørke, så vil resistansen hos R_{LDR} bli større. Når resistansen i seriekoblingen (R_{LDR}+ R_2) øker, vil strømmen reduseres. Det vil også si at strømmen i R_2 blir mindre, hvilket betyr at spenningen over R_2 også blir mindre, som igjen vil si at spenningen inn på mikrokontrolleren blir mindre. Dermed vil spenningen øke når det blir mørkere, og minke når det blir lysere.

Vi kan bruke det samme resonnementet på tilfelle B), og vil da oppdage at spenningen inn på mikrokontrolleren i dette tilfelle vil øke når det blir mørkere og reduseres når det blir lysere.

Den som ønsker en mer utdypende forståelse av spenningsdeleren, se avsnitt 9.3. I avsnitt 9.4 er den lysfølsomme motstanden beskrevet.

Programmering

Figuren under viser et flytdiagram hvor det blinkende gule lyset er inkludert.



1. **Les av en analog port**

   Fra koblingsskjemaet finner vi hvilken port lyssensoren er koblet til, vi har valgt port A0. Siden vi har koblet opp etter alternativ A) så vil *økende lys gi økende spenning*. Den analoge porten leses av med følgende kommando:

   ```
   lysStyrke = analogRead(<port nr.>);
   ```

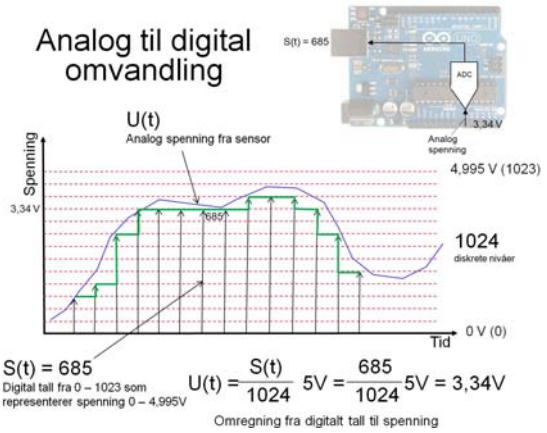
   Vi har valgt å oppbevare verdien til den avleste lysstyrken i variabelen `lysStyrke` som kan være et heltall (int). Når vi leser av spenningen på en analog port, så vil vi ikke få spenningen i volt, men som et tall mellom 0 – 1023. La oss derfor se litt nærmere på hva vi får ut av en analog til digital omformer når vi tilfører den en spenning. Husk at spenningen inn på en port alltid er referert til jord (GND, og er som regel det samme som 0V eller – på batteriet).

Analog til digital omforming

Spenningen inn på en analog port hos Arduino UNO kan være fra 0 – 4,995V. Negative spenninger og spenninger over 5V kan skade mikrokontrolleren. Analoge spenninger kan i prinsippet innta alle mellomliggende verdier. Når vi skal gjøre om en analog spenning til et digitalt tall, må vi bestemme oss for hvor nøyaktig målingen skal være. Jo mer nøyaktig, jo flere bit trenger vi for å representere verdien digitalt.

Analog til digital konverteren hos Arduino UNO har 10 bit, det vil si at den kan representere tall fra $0 – (2^{10} – 1) = 0 – 1023$. Siden den største spenningen vi kan måle er 4,995V, så vil nøyaktigheten pr. bit ikke bli bedre enn

$$4{,}995\text{V}/1024 = 4{,}9\ \text{mV}^{7}$$



Den analoge spenningen vil dessuten ikke kunne representeres kontinuerlig i tid, men punktprøves ("samples") i gitte tidspunkter. Figuren til høyre illustrerer hvordan den analoge spenningen, U(t), blir målt (punktprøvd) i bestemte tidspunkter og gitt en digital verdi, *S(t),* som ligger nærmest den virkelige verdien.

Ønsker vi å få gjengitt spenningen i punktet, *U(t)*, må vi regne om fra det punktprøvde tallet, *S(t)*, til spenning på følgende måte:

$$U(t) = \frac{S(t)}{1024} \cdot 5\text{V} \tag{7.1}$$

i eksempelet på figuren får vi:

$$U(t) = \frac{685}{1024} \cdot 5\text{V} = 3{,}34\text{V} \tag{7.2}$$

Program for å lese av verdi fra sensoren:

```
void setup() {
  Serial.begin(9600);          // Setter hastigheten til kommunikasjon PC
}
```

7. Egentlig er den høyeste spenning Arduino UNO kan måle lik 4,995V. Dvs. at 1023 tilsvarer 4,995V, hvilket betyr at 1024 representerer 5,0V

```
void loop(){
  int lysStyrke = analogRead(A0); // Leser av sensoren
  Serial.println(lysStyrke);      // Skriver resultatet til monitoren
  delay(1000);                    // Måler en gang i sekundet
}
```

2. **Bestem terskelverdi mellom lys og mørke**

   Bruk programmet til å lese av spenningen fra lyssensoren og skriv verdien ut i monitoren. Siden vi kun er interessert i endringer mellom lys og mørke, er det tilstrekkelig å studere *det digitale tallet uten å regne om til spenning*.

   Tallverdi når det er normal belysning: _____

   Tallverdi når det er mørkt: _____

   Det er viktig at forskjellen mellom de to tallverdiene er stor nok slik at vi kan legge inn en grenseverdi mellom.

   Grenseverdien velges til: _____

3. **if-setning med betingelse**

   Bruk en if-setning og sett opp en passende betingelse slik at utfallet blir forskjellig avhengig av om det er lys eller mørke.

   Lyst: Programmet skal følge vanlig rutine med bestilling av grønt.

   Mørkt: Programmet skal vise blinkende gult.

   Bruk gjerne flytdiagrammet over.

4. **Skriv programmet** og test det
5. Undersøk om det oppfører seg som forventet

Finn feilen i løsningsforslaget: Det er satt inn feil terskelverdi for når blinkende gult lys skal starte å blinke, hvilket betyr at uansett hvor mørkt det blir så blinker ikke trafikklyset gult. Juster terskelverdien i henhold til dine målinger slik at trafikklyset fungerer etter hensikten.

_____START LØSNINGSFORSLAG-6_____

```
/*
Trafikklys Deloppdrag 6
Lag et trafikklys som lyser slik som et trafikklys skal lyse
når det skifter fra rødt til grønt og tilbake.
Skifte skal imidlertid kun inntreffe ved trykk på knappen. Fra nå av skal
det grønne lyset lyse konstant i 8 sek. for deretter å etterfølges av 5 blink
med et halvt sekund på og et halvt sekund av. De gule lysene skal være på i 1 sek,
og det skal gå 5 sek. etter at det er trykket på knappen før det skifter til grønt
Det skal gis en kort lydpuls hver gang det blinkende grønne lyset skifter
Når mørket faller på skal det gule lyset begynne å blinke*/
```

```
int ledPinRed = 13;    // Kontakt rød LED til pinne pin 13
int ledPinYellow = 12; // Kontakt rød LED til pinne pin 12
int ledPinGreen = 11;  // Kontakt rød LED til pinne pin 11
int tonePin = 6;       // Pinnen der den passive buzzeren er tilkoblet
int knappBestilling = 7; // Kontakt knapp for bestilling av grønn mann
int lysStyrke; // Angir hvor lyst det er i ved trafikklyset
boolean bestilling = false; // Boolsk variabel som holde bestilling

// Definer konstanter

const int RODTID = 5000;       // 5 sekunder før det blir grønt
const int GULTID = 1000;       // 1 sekund "gultid"
const int ANTALLBLINK = 5;     // 5 blink, ett blink er på og av.
const int LENGDEBLINK = 1000;  // 1 sek, inkluderer halvparten av og halvparten på
const int GRONNTID = 8000;    // Lengden det skal være fast grønn før det begynner
å blinke.
const int KORTLYDPULS = 100;   // Kort lydpuls er satt til 100 msek.

// Setup definerer hva som er utganger og hva som er innganger

void setup() {
pinMode(ledPinRed, OUTPUT);
pinMode(ledPinYellow, OUTPUT);
pinMode(ledPinGreen, OUTPUT);
pinMode(tonePin, OUTPUT);
pinMode(knappBestilling, INPUT);
}

// loop() kjører om og om igjen i det uendelige

void loop()
{
 lysStyrke = analogRead(A0);                 //Leser lysintensiteten på LDR
 bestilling = digitalRead (knappBestilling); //Les av knappens innstilling

if (bestilling == 0)
 {
   gront_lys();        // Utfør endring til grønt lys og tilbake
   bestilling = true;  // Sett bestilling til null/falsk
 }
 else
 {
```

```
 if ((lysStyrke < 0) && (bestilling == 1))
  {
  gult_blinkende();              // Start blinkende gult
  }
 else
  {
  digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode
  }
 }
}

void gront_lys()
{
  digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode

  delay(RODTID);

  digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode

  delay(GULTID);

  digitalWrite(ledPinRed, LOW); // Slå av rød lysdiode
  digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
  digitalWrite(ledPinGreen, HIGH); // Slå på grønn lysdiode

  delay(GRONNTID);

  for (int i = 0; i < ANTALLBLINK; i++)
   {
     digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
     Lyd(KORTLYDPULS); // Gi lydstøt på 100ms
     delay(LENGDEBLINK/2);
     digitalWrite(ledPinGreen, HIGH);
     Lyd(KORTLYDPULS); // Gi lydstøt på 100ms
     delay(LENGDEBLINK/2);
  }

   digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
   digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode

   delay(GULTID);
```

```
   digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
}

void Lyd(int varighet)
{
  tone(tonePin, 1500, varighet); // Angir pinnenummer for piezoelektrisk lydgiver,
                                 // frekvens og varighet i msek
}

void gult_blinkende()
{
  digitalWrite(ledPinRed, LOW); // Slå av rød lysdiode
  digitalWrite(ledPinYellow, HIGH); // Slår på gult
  delay(LENGDEBLINK/2);
  digitalWrite(ledPinYellow, LOW); // Slår av gult
  delay(LENGDEBLINK/2);
}
```

_____STOPP LØSNINGSFORSLAG_____

7.1.9 Deloppdrag 7 – Sammenkobling av to trafikklys

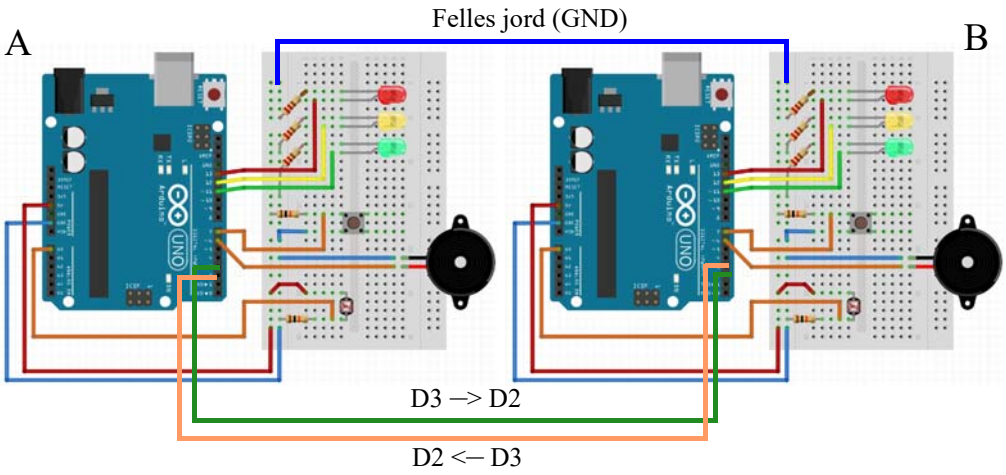
I dette oppdraget skal to trafikklys kobles sammen:

- *A) Tenk dere en fotgjengerovergang. Når noen trykker på knappen på trafikklyset på den ene siden av gata så skal begge lysene virke samtidig. Dette skal fungere uansett fra hvilken side bestillingen kommer.*
- *A) Tenk dere at dere skal lage et lyskryss hvor det ene lyset er for fotgjengerovergangen og det andre for bilene.*
- *Kan dere løse oppgavene ved enten å gå sammen to og to grupper, eller ved å inkludere et ekstra sett med lysdioder i den ene Arduino UNO'en dere har (dette er aktuelt når dere bruker TinkerCAD)?*

I dette oppdraget overlater vi til dere å finne løsninger, men vi tillater oss gi noen tips som dere kan velge å bruke eller ikke:

Tips: Dersom dere bruker to sett

- Bruk to porter på hver av de to Arduino'ene for å overføre informasjon mellom de to kortene som antydet på figuren under:



- Bruk de to signalene til å varsle det andre trafikklyset om at det ønskes et skifte
- Husk også å koble sammen jord på de to kortene

1. ***Skriv de to programmene*** og undersøk om de fungerer sammen.

Tips: Dersom dere bruker TinkerCAD

- Bruker dere TinkerCAD må dere koble opp begge lysdiodesettene på det samme koblingsbrettet.
- Dere trenger to knapper for å bestille grønt lys, men kun en lyssensor for å måle lysintensiteten.

7.2 Lag et kolorimeter

Det skal lages et *kolorimeter* for å måle transmisjon av lys gjennom en oppløsning. Et slikt instrument brukes ofte til å bestemme konsentrasjonen av en oppløsning. I et kolorimeter måles absorbsjonen av lys gjennom oppløsningen. Jo mindre lys som slipper gjennom, jo høyere er konsentrasjon av det oppløste stoffet. Som test kan man benytte vann tilsatt dråper med melk, selv om lyset i dette tilfellet spres mer enn det absorberes.

Man alternativt måle spredning av lyset fra partiklene i blandingen. I så fall monteres lyskilden vinkelrett på lyssensoren, dvs. man måler reflektert lys framfor absorbert lys. Et slikt instrument kalles et *turbidimeter*. Her foreslår vi at dere bruker instrumentet som et kolorimeter.

Som test skal instrumentet kunne bestemme rekkefølgen til 10 plastkrus med 1,5 dl vann og som er "forurenset" med ulike antall dråper melk.

Det er laget et eget hefte som guider dere gjennom følgende deloppgaver:

7.2.1 Deloppdrag 1 – Montering av lysdiode og lyssensor i kyvettehuset

Et kolorimeter består av en lyskilde og en lyssensor, montert i et mørkt kammer som hindrer lys fra utsiden å slippe inn. Kammeret må være så stort at det kan romme et plastkrus med plass til 1,5 – 2 dl vann.

- *Koble opp kammeret som anbefalt med lysdiode og lyssensor og tilhørende elektronikk*

7.2.2 Deloppdrag 2 – Lag et program for å avlese lysstyrken

- *Lag et program som leser av lysstyrken til lyset som slipper gjennom oppløsningen og skriv resultatet til Arduino-monitoren. Start gjerne med å tegn et flytdiagram for programmet.*
- *Bestem følsomheten til instrumentet, dvs. endring i måleverdi som funksjon av antall dråper melk i oppløsningen. Bruk de vedlagte begrene, pipetten og en skvett melk.*

7.2.3 Deloppdrag 3 – Gjør målinger på 10 beger og sett dem i rett rekkefølge

Gjør så målinger på hvert av de ti begrene.

- *Gjør systematiske målinger på de ti begrene med melk og bestem rekkefølgen fra det med færrest dråper til det som har flest dråper.*

7.2.4 Deloppdrag 4 – Koble opp og skriv til et eksternt display

Koble opp et lite display slik at det blir mulig å lese av verdiene uten å bruke monitoren.

- *Monter displayet og installer biblioteket og test ut at det fungerer som ønsket.*

7.2.5 Deloppdrag 5 – Skriv ut antall dråper i hvert beger

Bestem antall dråper melk i hvert enkelt beger.

- *Finn grenseverdiene for skifte mellom de ulike antall dråper. Lag en tabell og skriv ut antall dråper på displayet. Instrumentet er nå kalibrert til dråper melk.*
- *Undersøk hvor følsomt instrumentet er ved å bestemme en gjennomsnittlig avstand i måleverdiene for hver økning av en dråpe. Sjekk om det er en lineær sammenheng mellom absorbansen og antall tilsatte dråper.*

Et par tips:

Vurder å:

- ... midle måleresultatene over mange målinger.
  Hva vil fordelen med en slik metode være?
- ... slå av lyskilden mellom hver måling.
  Hva vil du kunne oppnå med en slik løsning?

For detaljer, se eget hefte: *Måling av partikkeltetthet i væske med Arduino - Lærerveiledning.*

7.3 Lag en høydemåler

Vi skal i denne oppgaven lage en barometrisk høydemåler, dvs. et instrument som måler høyden over havet basert på kunnskapen om hvordan lufttrykket faller med økende høyde. Det er et slikt instrument som også flyene tradisjonelt bruker for å bestemme høyden til flyet.

Det er også til denne oppgaven laget et eget arbeidshefte:

7.3.1 Deloppdrag 1 – Blinkende lysdiode, nedtelling og bruk av display

Også disse oppdragene egner seg som introduksjon til programmering av Arduino. Vi begynner derfor litt forsiktig. Om man synes introduksjon er for elementær kan man gå fort over de innledende oppdragene:

- *Skriv et program som får en lysdiode til å blinke. Varier blinktakten.*
- *Bruk print-kommandoen og tell ned i monitoren. "Stopp" programmet når nedtellingen kommer til null.*
- *Koble opp et eksternt display (OLED SSD1306) og vis nedtellingen på displayet*

7.3.2 Deloppdrag 2 – Lag et voltmeter og et termometer

Vi skal nå bruke Arduino,en til noe mer fornuftig:

- *Lag et enkelt Voltmeter og skriv den målte verdien til displayet med tekst og benevning*
- *Koble opp temperatursensoren TMP36, avles temperaturen og skriv resultatet til displayet med tekst og benevning.*

7.3.3 Deloppdrag 3 – Måling av lufttrykk med BMP280

BMP280 måler både lufttrykk og temperatur og er grunnlaget for den barometriske høydemåleren

- *Koble opp BMP280, installer bibliotekene og les av lufttrykk og temperatur*
- *Skriv temperatur og lufttrykk med i °C og i mBar på displayet med to desimaler, skriv tekst og benevning sammen med avlest verdi.*

7.3.4 Deloppdrag 4 – Omregning fra lufttrykk til høyde

Lufttrykket endrer seg med høyden over havet. I nedre del av atmosfæren er det relativt enkelt å modellere hvordan dette lufttrykket endrer seg med høyden.

- *Bruk en vanlig omregningsformel og beregn høyde som funksjon av lufttrykk og temperatur.*
- *Skriv ut den beregnede høyden som funksjon av lufttrykket med to desimaler, tekst og benevning*
- *Midle det avleste lufttrykket over 100 eller 500 målinger og registrer at verdien blir mer stabil fra måling til måling.*

7.3.5 Deloppdrag 5 – Kalibrering av instrumentet

Foreløpig måler instrumentet kun relativ høyde, for å kunne måle absolutt høyde må instrumentet kalibreres.

- *Gå inn på en nærliggende metrologisk stasjon i nærheten og lese av stasjonens lufttrykk og høyde over havet.*
- *Kalibrer instrumentet med de innhentede måledataene og les av den kalibrerte høyden. Skriv resultatet ut på displayet uten desimaler med tekst og benevning.*
- *Kontrollert at den målte verdien er i nærheten av den virkelige høydene som man finner på et kart.*

Opplegget er beskrevet i detalj i heftet: *Grunnkurs programmering*

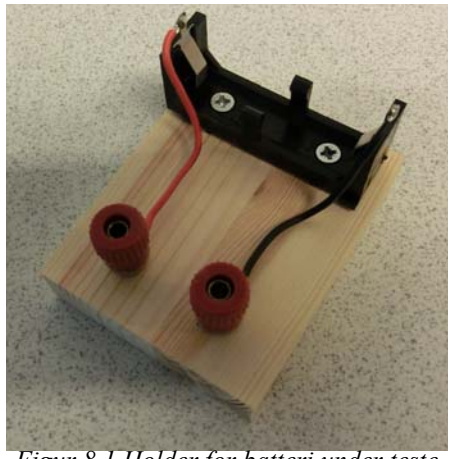
8 Halvåpne oppgaver

I dette kapittelet skal vi gi eksempler på ulike halvåpne oppgave, dvs. oppdraget er gitt som en funksjonspesifikasjon, slik at ordentlig oppdrag gjerne er gitt av en oppdragsgiver.

8.1 Lag en batteritester

Det skal lages en batteritester for 1,5 V eller 2 x 1,5 V batterier.

- *Studer databladet for batteriet og finn ut hvilke kriterier dere vil sette for et ubrukelig, et dårlig og et godt batteri.*



Figur 8.1 Holder for batteri under teste.

8.1.1 Deloppdrag 1 – Les av batterispenning

- *Les av kalibrert spenning, og skriv den til monitoren i Arduino-monitoren*

8.1.2 Deloppdrag 2 – Godt eller dårlig batteri

- *Skriv tekst til monitoren som forteller om batteriet er godt eller dårlig eller ubrukelig*

8.1.3 Deloppdrag 3 – LED som viser godt eller dårlig batteri

- *Monter dioder på koblingsbrettet som viser om batteriet er godt (grønn), dårlig (gult) eller ubrukelig (rødt)*

8.1.4 Deloppdrag 4 – Tegn koblingsskjema

Ev. ekstra oppgave:

- *Tegn koblingsskjema i Fritzing*

8.1.5 Deloppdrag 5 – Uttesting av batteritesteren

- *Hvordan vil dere teste ut batteritesteren?*

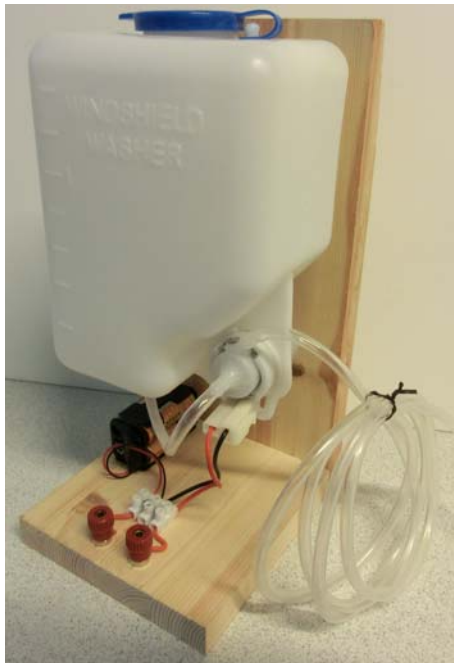
8.1.6 Utstyr:

- Batteriholder med batteri og tilkoblingsklemme
- Multimeter
- Sparkfun's Inventors kit med Arduino UNO

Løsningsforslag finnes i vedlegg C.1.

8.2 Automatisk blomstervanner

Det skal lages en automatisk blomstervanner.



8.2 Vannbeholder med pumpe og tilkoblin for styring av automatisk vanning.

8.2.1 Deloppdrag 1 – Vanning ved tørr jord:

- *Vanning skal skje kun når planten er tørr*
- *Vannmengden skal tilpasses plantens behov*
- *Hvordan vil du **teste** ut om denne fungerer som den skal?*

8.2.2 Deloppdrag 2 – Sikkerhet mot vannsøl:

- *Det skal legges inn ekstra **sikkerhet** mht. til søl. Dersom det kommer vann på bordet skal vanningen stoppe umiddelbart.*
- *Hvordan vil du **teste** ut om denne fungerer som den skal?*

8.2.3 Deloppdrag 3 – Differensiert vanning

- *Noen planter trenger lite vann ofte, andre mye vann sjelden.*
- *Vurdere fuktighetsgraden slik at litt fuktig gir 0,5 dl vann. Svært tørr gir 1 dl vann.*
- *Hvordan vil du **teste** ut om denne fungerer som den skal?*

8.2.4 Deloppdrag 4 – Tegn koblingsskjema

Ev. ekstra oppgave:

- *Tegn koblingsskjema i Fritzing*

8.2.5 Utfordring

- *Hvordan vil dere teste ut vanneren?*

8.2.6 Utstyr:

- Begerglass, vann
- Komplett sett med motstander (E12) (Clas Ohlson)
- Koblingsledning (ELFA)
- Vannbeholder med pumpe, batteri og slanger (Biltema)

- Multimeter (Clas Ohlson/ELFA)
- Sparkfun's Inventors kit med Arduino UNO (Sparkfun)

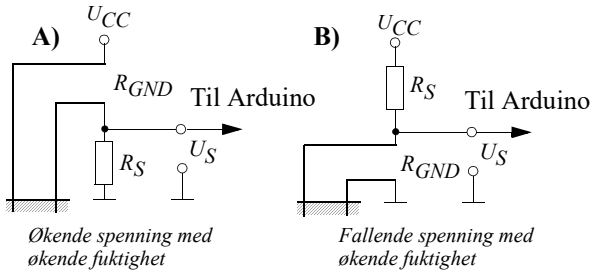
Løsningsforslag finnes i vedlegg C.2.

8.2.7 Tips:

Fuktighetsdetektor

En enkel fuktighetsdetektor kan lages med to ledninger som stikkes ned i jorda. Betingelsen er at fuktig jord har lavere resistans enn tørr jord, hvilket er en god antagelse. Siden en mikrokontroller ikke direkte kan måle endring i resistans, må vi gjøre om endringen av resistans til en endring i spenning som tilføres en av mikrokontrollerens analoge innganger (A0 – 5). Dette gjør vi ved å benytte spenningsdeleren (se avsnitt 9.3, side 128).

Siden vann ikke er noen god leder, selv om fuktig jord er vesentlig bedre, så bruker vi en spenningsdeler med en relativt høy seriemotstand. Her kan man gjerne eksperimentere litt, men en seriemotstand på 470kΩ – 1MΩ kan være et fornuftig sted å starte. Som ellers kan vi selv velge om spenningen skal øke med økende fuktighet eller minke med økende fuktighet, som vist på figuren til høyre.



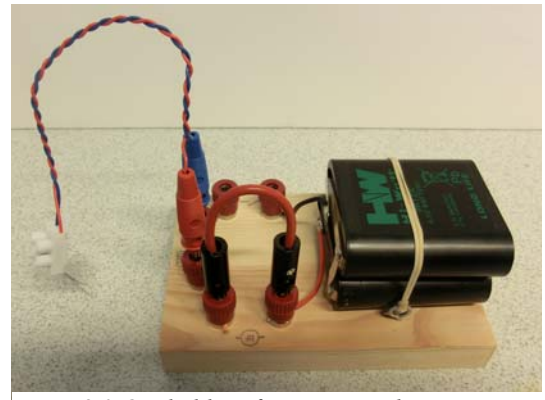
Det finnes også mer "profesjonelle" sensorer for å registrere fuktighet i jord. Et par slike er omtalt i avsnitt 9.6, side 138. Her er det hovedsaklig to typer å velge i. Det er de som baserer seg på endring i jordas ledningsevne som omtalt foran, og de som bruker *endring i kapasitet* som følge av at sensoren omgir seg med tørr eller fuktig jord. Den sistnevnte har en fordel siden den ikke korroderer like lett som den førstnevnte.

Rele med transistordriver og "flyback" diode

For å kunne styre pumpa må vi bruke et rele som drives av en transistor. For nærmere beskrivelse se side 119.

8.3 Automatisk vannvarmer

Vannet i et begerglass skal varmes opp til 30°C ved hjelp av et lite varmeelement.



Figur 8.3 Oppkobling for automatisk vannvarmer. Temperaturmåleren (NTC) er ikke med.

8.3.1 Deloppdrag 1 – NTC-motstand

- *Koble opp NTC-motstanden i spenningsdeleren. Velg en seriemotstand på 47kΩ, som er det nærmeste vi kommer i E12 rekken.*
- *Mål og finne en sammenheng mellom spenning og temperatur i området 15 – 50°C. Tegn en graf.*

8.3.2 Deloppdrag 1 – Termostat

- *Lag en innretning ved hjelp av Arduino UNO som varmer opp og holder temperaturen konstant på 30°C. Vi bruker NTC-motstanden til å måle temperaturen. Vi bruker et rele til å slå av og på strømmen i varmeelementet.*

8.3.3 Deloppdrag 2 – Alarm

Lag en alarm ...

- *... som gir fra seg en høy tone hver gang den går fra 29°C til 30°C.*
- *... som gir fra seg en dyp tone hver gang den går fra 30°C til 29°C.*

Bruk buzzeren.

8.3.4 Deloppdrag 3 – Tegn koblingsskjema

Ev. tilleggsoppgave:

- *Tegn koblingsskjema i Fritzing*

8.3.5 Utstyr:

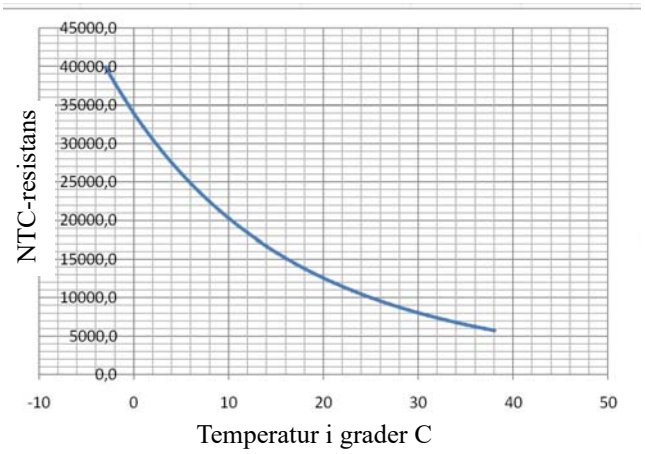
- Begerglass, termometer, vann,
- Varmtvannskran/Vannkoker
- NTC motstand (50 kΩ)
- Varmeelement, batteri m/tilkobling
- Multimeter
- Sparkfun's Inventors kit med Arduino UNO

Løsningsforslag finnes i vedlegg C.3.

8.3.6 Tips:

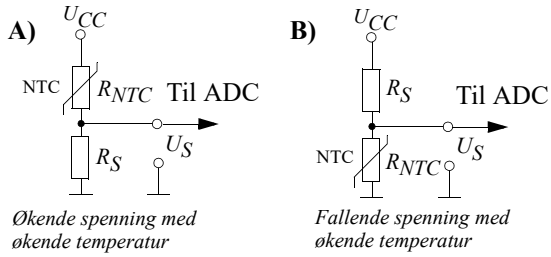
NTC-motstand som temperaturmåler

En NTC-motstand er en motstand som endrer resistans som funksjon av temperaturen. NTC har negativ temperatur koeffisient hvilket betyr at resistansen avtar med økende temperatur. Sammenhengen temperatur er ulineær som vist på figuren til høyre.

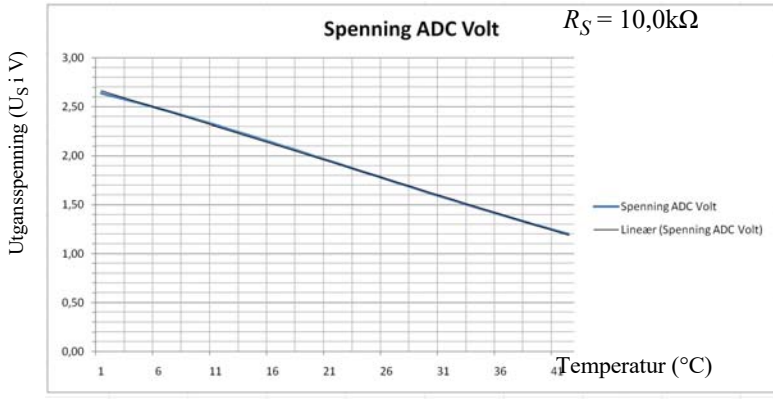


Vi legger merke til at resistansen for nettopp denne NTC-motstanden er 10kΩ ved 25°C. Denne verdien kaller vi NTC-motstandens referanseverdi (R_{25}) og 25°C er i dette tilfellet referansetemperaturen (T_r).

Siden vi ikke direkte kan måle resistans via portene på kontrollerkortet, men kun kan måle spenning, kobler vi NTC-motstanden i serie med en motstand som vist i figuren til høyre. Dersom vi velger verdien til seriemotstanden lik referanseverdien til NTC-motstanden, så viser det seg at sammenhengen mellom temperatur og spenning blir relativt lineær i området rundt referansetemperaturen (T_r), som vist på figuren under. Legg merke til at oppkoblingen på tegning A gir økende spenning, U_S, med økende temperatur, mens



oppkoblingen i tegning B gir fallende spenning, U_S, med økende temperatur.



Vår NTC-motstand har en referansemotstand på 50kΩ ved 25°C.

For ytterligere beskrivelse av NTC-motstanden brukt som temperatur sensor, se avsnitt 9.5, side 133.

Rele med transistordriver og "flyback" diode

For detaljer se side 119.

8.4 Elektronisk terning

Det skal lages en elektronisk terning ved hjelp av 7 lysdioder.

8.4.1 Deloppdrag 1 – Planlegging av "rettferdig" terningen

- *Med ett trykk på en bryter skal terningen gi en verdi fra 1 – 6.*
- *Verdiene skal angis på samme måte som øynene på en vanlig terning*
- *Hva er det minste antall I/O utganger som er nødvendig for å lage terningen?*

8.4.2 Deloppdrag 2 - Oppkobling av terningen

- *Koble opp terningen med 7 LED og bryter, og ellers det som trengs*

8.4.3 Deloppdrag 3 – Skriv programmet

- *Skriv et program som genererer tilfeldige tall fra 1 til og med 6 når du trykker på bryteren, skriv resultatet til monitoren*
- *Styr ut lysdiodene slik at øynene på terningen gjengir verdiene fra 1 til 6.*

8.4.4 Deloppdrag 4 – Uttesting

- *Hvordan vil dere definere kvaliteten til terningen og hvordan vil dere teste at den oppfører seg som forventet?*

8.4.5 Deloppdrag 5 – Lag en "urettferdig" terning

- *Dere skal nå legge inn en bryter til som gir 50% større sjanse for å få en 6'er enn den "rettferdige" terningen.*
- *Skriv programmet til den "urettferdige" terningen*
- *Hvordan vil dere teste at den oppfyller kravet?*

8.4.6 Deloppdrag 6 – Hemmelig bruk av den "urettferdig" terningen

- *Lag mekanismen for å velge mellom de to terningene slik at den som kjenner trikset lett kan velge den "urettferdige" terningen uten at andre merker det.*
- *Undersøk om ett av de andre lagene klarer å avsløre jukset?*

8.4.7 Deloppdrag 7 – Tegn koblingsskjema

Ev. tilleggsoppgave:

- *Tegn koblingsskjema i Fritzing*

Løsningsforslag finnes i vedlegg C.4.

8.4.8 Utstyr:

- Multimeter
- Sparkfun's Inventors kit med Arduino UNO

8.4.9 Tips:

Tilfeldighetsfunksjonen

I denne oppgaven er funksjonen *random (min, maks)* funksjonen sentral. Denne funksjonen returnerer heltallsverdier mellom minimum (inklusive min) og maksimum (eksklusive maks). Så dersom man vil ha verdier mellom 1 og 6 så kan man f.eks. skrive følgende:

```
int tilfeldig;
tilfeldig = random(1, 7);
```

random() funksjonen, eller tilfeldighetsfunksjonen, genererer såkalte *psaudo random*-verdier, dvs. at den genererer en rekke tall som i store antall vil ha en tilnærmet flat fordeling, men hvor man kan erfare at utvalget ikke er så tilfeldig om man er uheldig med prøvetakingen. Ønsker man

en mer tilfeldig generator kan man bruke et såkalt "såkorn" (seed) som utgangspunkt for genereringen. "Såkornet" må i seg selv være tilfeldig, og det er ikke uvanlig å hente såkornet fra en analog inngang som henger åpen og dermed fanger opp støy.

```
int seed = analogRead(A0); // Kan legges i void setup()
randomSeed(seed);          // Kan legges i void setup()
int tilfeldig;
tilfeldig = random(1, 7);
```

Det vil si at tilfeldighetsgeneratoren er blitt startet med et tilfeldig støygenerert tall som såkorn slik psaude random generatoren vil få et forskjellig forløp fra gang til gang. Legges den i setup()-funksjonen vil den få en ny start hver gang setup()-funksjonen kjøres.

Bruk av interrupt

Interrupt brukes når vi ønsker å avbryte programmet uansett hvor det er i programflyten, vi kaller programvare-interrupt. Dette kan både gjøres av programmet, men vanligvis ved at en hendelse utenfor mikrokontrolleren krever oppmerksomhet, vi kaller det *hårdvare interrupt*. I vårt tilfelle ønsker at bryteren skal avbryte programmet.

Det er bare noen få av portene på Arduino'en som kan betjene et hårdvare-interrupt.

Hos UNO og NANO er bare portene 2 og 3 som kan håndtere interrupt. Arduino MEGA har et større utvalg, der portene 2, 3, 18, 19, 20, 21 kan brukes, så fremt 20 og 21 ikke brukes som I^2C-bus.

Bruken skjer i flere trinn. Først må man fortelle hvilken port man ønsker å bruke til interruptet:

```
attachInterrupt(0,visResultat,RISING); // Initier interrupt på port 2
```

Det er slik at interruptene har interne nummer, her kalt 0 og 1. Hos UNO er interrupt 0 knyttet til port 2 og interrupt 1 til port 3. Vi ønsker å bruke interrupt på port 2, dvs. interrupt 0.

Dessuten vil vi at interruptet skjer idet signalet på porten går fra lav til høy, dvs. stigende flanke (RISING), alternativt kan en velge FALLING (fallende flanke) eller CHANGE (endring) eller LOW (lavt nivå).

Dernest må vi fortelle hvilken funksjon programmet skal hoppe til når programmet avbrytes av et interrupt. Her har vi valgt å gå til funksjonen som viser resultatet av "terningkastet", `visResultat;`

```
void visResultat()
{
  // Skriv her det du vil at funksjonen skal gjøre før går tilbake
}
```

Man kan også slå av interruptet ved følgende kommando:

```
detachInterrupt(0); // Slå av interrupt på port 2
```

Man bør være klar over at når man befinner seg i interrupt-funksjonen så vil ikke `delay ()` fungere og `millis()` vil ikke bli oppdatert.

Bruk av switch/Case

Switch/Case kommandoen kan minne om `if()`-setningen, i og med at det er en kommando som styrer retningen til programmet på bakgrunn av verdien til en variabel <`var`>.

```
switch (var) {
    case 1:
      //Gjør noe når variabelen var er lik 1
      break;
    case 2:
      //Gjør noe når variabelen var er lik 2
      break;
    default:
      // Om ingenting stemmer med variabelens verdi, gjør default
      // default er valgfritt
break;
  }
```

For mer informasjon se avsnitt 2.4.11, side 41.

8.5 Stoppeklokke

Det skal lages en stoppeklokke, som kan brukes til å måle lange og korte tidsintervaller. En trykk-bryter skal kunne brukes til å starte og stoppe klokka. Klokka kan også brukes som reaksjonstester. Utvikle prosjektet i flere trinn.

Lag en klokke med følgende spesifikasjoner:

8.5.1 Deloppdrag 1 – Lag en klokke som går

- *Klokke skal starte på 0 sekunder hver gang programmet starter.*
  Den skal vise sekunder og minutter med en nøyaktighet på +/- 0,5 sek
  Ta utgangspunkt i Eksempel øving 15 "LCD display"
- *Lag et flytdiagram for programmet*

8.5.2 Deloppdrag 2 – Klokke med minutter, sekunder og 10-deler

- *Lag klokka slik at den også viser 10-dels sekunder*
  Ellers som i oppdrag 1
- *Juster flytdiagrammet for programmet*

8.5.3 Deloppdrag 3 – Start klokka med en trykkbryter

- *Bruk en trykkbryter og start klokka på 0 sekunder hver gang bryteren trykkes inn og slippes.*
  Ellers som i oppdrag 3

- *Tegn koblingsskjema for oppkoblingen av trykkbryteren*
- *Juster flytdiagrammet for programmet*

8.5.4 Deloppdrag 4 – Stopp klokka ved andre gangs trykk på bryteren

- *Bruk den samme bryteren til å stoppe klokka og holde siste verdi så lenge knappen holdes inne. Ved et nytt trykk skal klokka starte på nytt på null. Ellers som i C.*
- *Juster flytdiagrammet for programmet*

8.5.5 Deloppdrag 5 – Bruk stoppklokka til å måle reaksjonstida

- *Når startknappen trykkes går det en tilfeldig tid på noen sekunder en lysdiode tennes. Ellers som i oppdrag 4.*
- *Idet dioden tennes skal knappen trykkes på nytt og vise reaksjonstiden.*
- *Drøft feilkilder for stoppeklokka*

Løsningsforslag finnes i vedlegg C.5.

8.5.6 Tips

Bruk av funksjonen millis()

Funksjonen millis() returnerer tiden i millisekunder siden Arduinoen ble slått på, eller resatt. Ukalibrert egner den seg godt til å holde orden på tidsintervaller, som f.eks. i en stoppeklokke. Pass på at funksjonen returnerer lange heltallsverdier slik at variabler som skal ta vare på avlesninger av millis(), bør ha type long.

```
long millisekunder;
millisekunder = millis();
```

For mer stoff om hvordan bruke millis() se avsnitt 2.4.5, side 34.

Bryter med pullup motstand

Se side 118 for nærmere beskrivelse av bryter med pullup motstand og hvordan vi skal lese av en bryter.

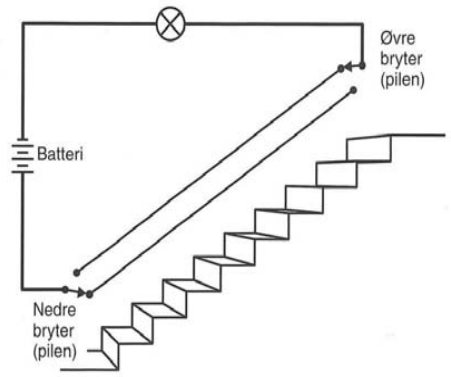
8.6 "Trappelys"

I dette oppdraget skal vi lage et moderne system for lys i en trappeoppgang med tre etasjer. I hver etasje er det en bryter og en lyspære. Uansett i hvilken etasje vi befinner oss skal vi kunne slå av eller på lyset i hele trappeoppgangen. Det betyr at bryteren i etasjen vi befinner oss i skal fungere som av-bryter når lyset er på, og som på-bryter når lyset er av.

Figuren til høyre viser hvordan dette ble løst med to etasjer i "gamle dager" da man ikke brukte så mye elektronikk som man gjør idag.

I eksempelet i figuren er lyset slått på. Uansett hvilke av de to bryterne (venderne) som betjenes vil kretsen sluttes og lyset slås på.

Dette blir ganske komplisert når det blir mange etasjer. Da er det lurt å bruke et rele og trykkbrytere.



Lyset er skrudd av med øvre bryter! Studer hvordan strømmen går!

8.6.1 Deloppdrag 1 – Tegn koblingsskjema

- *Tegn et enkelt koblingsskjema for kretsen med tre etasjer og bruk av rele.*

8.6.2 Deloppdrag 2 – Koble opp kretsen

- *Koble opp kretsen med en Arduino UNO, koblingsbrett, 3 x trykkbrytere, 3 x lysdioder og ett rele.*

8.6.3 Deloppdrag 3 – Skriv programmet

- *Lag et program som er slik at uansett hvilken av de tre bryterne som trykkes inn, så skal alle LED'ene tennes i hele trappeoppgangen. Om de er av skal de tennes, om de er på skal de slukkes.*

8.6.4 Deloppdrag 4 – Automatisk tenning av lyset ved mørkets frambrudd

- *Dersom vi glemmer å slukke lyset i trappegangen når det blir lyst, så skal dette skje automatisk. Lyset skal være av helt til noen trykker på en av bryterne. Dersom noen forsøker å tenne lyset mens det er lyst skal lyset slås på, men slås av etter 5 sekunder. Dersom det er mørkt skal lyset bli stående på til det blir lyst eller aktivt slukkes. En LDR-sensor kan brukes til å registrere lysnivået (se under).*

8.6.5 Deloppdrag 5 – Automatisk tenning av lyset av bevegelse

- *For ytterligere å spare elektrisk energi så ønsker vi å tenne lyset når noen kommer inn gjennom inngangsdøra og det er mørkt. Lyset skal da stå på i 5 sekunder før det slukkes automatisk. Lyset skal bare tennes når det er mørkt. En PIR-sensor kan brukes til å registrere bevegelsen.*

Løsningsforslag finnes i vedlegg C.6.

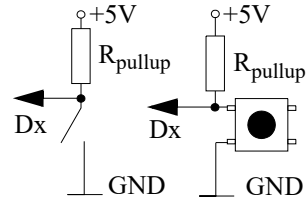
8.6.6 Utstyr

For å løse dette oppdraget trenger vi følgende komponenter:

1 stk Arduino UNO
1 stk koblingsbrett
3 stk trykkbrytere med pulldown motstand
3 stk lysdioder med seriemotstander på 330Ohm
1 stk rele med flyback diode
1 stk lyssensor (ev. LDR m/10kOhm seriemotstand)
1 stk PIR-sensor
Div jumpere

8.6.7 Tips

Bryter med pullup motstand



Figuren til høyre viser hvordan vi kan koble opp en bryter. Dx er en fritt valgt digital port på mikrokontrolleren. Når bryteren *ikke* er trykket inn, vil verdien på inngangen, Dx, være 5V dvs. høy (1). Når den så trykkes inn vil den legge Dx til jord (0). Hensikten med en slik motstand er at inngangen skal ha en definert verdi når den ikke er trykket, her er den høy.

Istedet for å koble på en ekstern motstand, R_{pullup}, så kan vi via programmet legge inn en intern pullup-motstand. Det gjøre vi med følgende kommando i void setup()-funksjonen:

```
pinMode(Dx, INPUT_PULLUP);
```

På denne måten definerer vi porten som en inngang med pullup-motstand.

Vi har imidlertid erfart at dersom flere brytere kobles på den samme inngangen så, kan den innvendige pullup-motstanden bli for svak og det kan være nødvendig med en ekstra utvendig på 10kΩ.

Avlesning av bryter

Bryterne avleses og den avleste verdien legges inn i en variabel med følgende kommandoer:

```
int brytere;
brytere = digitalRead(pinBrytere);
```

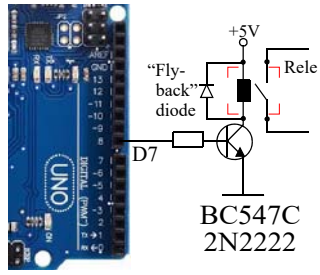
`pinBrytere` er portnummeret der bryterutgangen er koblet til mikrokontrolleren. `brytere` er variabelen som holder den avleste verdien: `LOW` (0) om den er trykket inn, og `HIGH` (1) om den ikke er trykket inn.

Siden mikrokontrollere jobber ekstremt fort, så kan en lett oppleve at den rekker å lese av en inntrykket knapp flere tusen ganger før man rekker å slippe den. I noen tilfeller er dette en ulempe. For å unngå at det skjer kan man la programmet gå i en vente-loop helt til man igjen har sluppet knappen. Dermed vil et trykk kun resultere i en registrert verdi. Til dette formålet kan man benytte en while(<betingelse>)-loop.

```
while(digitalRead(pinBrytere) == LOW);
```

Så lenge betingelsen er oppfylt, dvs. en av bryterne trykket inn, så vil det leses LOW (0) fra port `pinBrytere`. Dermed er betingelsen oppfylt og programmet vil gjenta while()-kommandoen og ikke gå videre i programmet før bryteren er sluppet, dvs. verdien HIGH (1) leses fra porten.

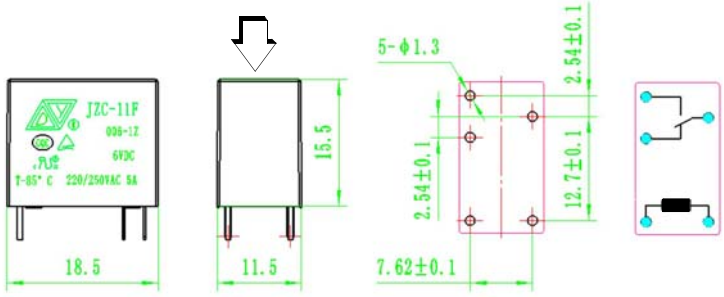
Rele med transistordriver og "flyback" diode



Releer er elektronisk styrte brytere som gjerne slå av og på høyere spenninger og strømmer, enn hva f.eks. en transistor klarer. Imidlertid er det ikke alltid at utgangen på mikrokontrolleren klarer å drive et rele. Derfor er det vanlig at vi bruker en transistor mellom utgangen til mikrokontrolleren og releet.

En seriemotstand settes inn i baseledningen til transistoren for å begrense strømmen. Denne bør ikke være for stor. Blir basemotstanden for stor vil strømmen inn i basen bli for liten slik at transistoren ikke klarer å dra til releet.

I denne oppgaven skal vi bruke releet JZC-11F. Figuren under viser hvordan releet er koblet opp innvendig. Tegningen av tilkoblingsterminalene til høyre på figuren er vist sett fra oversiden, dvs. gjennom huset til releet.



Når vi bryter strømmen i en spole (induktans) vil det oppstå en spenningspuls, motsatt den spenningen vi vanligvis legger over spolen. Tradisjonelle releet styres gjerne av en spole. Slike spenningspulser kan lett ødelegge transistoren. For å hindre at det skjer, kobler vi en diode over relespolen som vist på figuren over. Dioden kobles i sperreretning slik at den normalt ikke vil lede strøm, men kun kortslutte de korte spenningspulsene som oppstår når vi bryter strømmen i spolen.

PIR-sensor

PIR-sensorer (Passive InfraRed sensor) detekterer varmestråling fra bl.a. mennesker som er i bevegelse i forhold til en bakgrunn. Sensoren sender ikke ut noe signal, men lytter kun til infrarød innstråling.

Kretsen trenger arbeidsspenning fra 3 – 5 V og har en rekkevidde på opp til 12 m. Responstiden er under 1 sekund[8].

Kretsen leveres med en kontakt med fire terminaler, VCC, GND og SIG. NC er ikke tilkoblet. VCC tilkoblet +5V og GND koles til jord. Når kretsen "oppdager" et menneske i bevegelse, legger den pinne SIG til høy verdi. SIG kobles til en digital inngang hos mikro-kontrolleren. Etter deteksjon holder den verdien i noen sekunder.

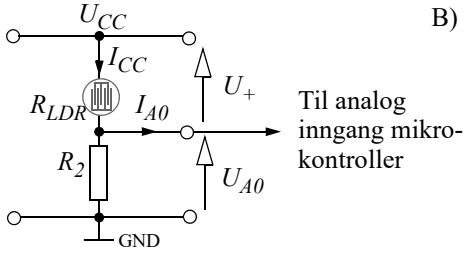
LDR lysdetektor med seriemotstand

Her velger vi å bruk en LDR (Light Dependent Resistor - Lysfølsom motstand) koblet i serie med en motstand mellom 5V og GND, dvs. vi lager oss en *spenningsdeler*.

En spenningsdeler er nyttig for å omdanne *en variasjon i en resistive sensorer*, som f.eks. i en LDR, til *en variasjon i spenning* som mikro-kontrolleren kan registrere på en av sine innganger.

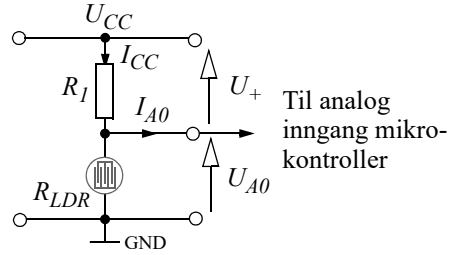
Ofte holder det at vi forstår spenningsdelerens prinsipielle funksjon siden vi ikke trenger å kjenne de eksakte verdiene for strømmer og spenninger.

A)



Økende lysstyrke —> Økende U_{A0}

B)



Økende lysstyrke —> Redusert U_{A0}

Som det framgår av figuren over så har vi to mulige måter å koble opp den lysfølsomme motstanden på avhengig av om vi vil at spenningen skal øke med økende lysstyrke (A) eller minke med økende lystyrke (B).

Her er det opp til oss å velge hva vi ønsker. Vi har valgt løsning A.

8. https://www.elfadistrelec.no/Web/Downloads/_t/ds/101020793_eng_tds.pdf

Den som ønsker en utdypende forståelse av spenningsdeleren, se avsnitt 9.3. I avsnitt 9.4 er den lysfølsomme motstanden beskrevet.

9 Utdypende teori om aktuelle komponenter

I dette kapittelet skal vi omtale noen av komponentene vi bruker litt mer uttømmende enn i oppgavesamlingen.

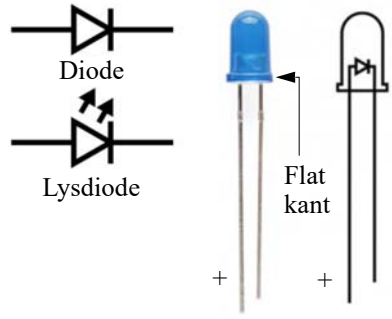
9.1 Lysdioder

Lysdioder finnes i svært mange varianter, både med hensyn til størrelse, utforming, farge og lysstyrke. I tillegg inneholder noen lysdioder flere enkeltdioder med ulike farger som kan styres individuelt.

9.1.1 Ordinære lysdioder[9]

Vi skal se litt nærmere på ordinære lysdioder som det finnes mange varianter av.

En lysdiode omdanner en elektrisk strøm til fotoner med ulik bølgelengde, farge. Fargen bestemmes av materialvalgene i halvlederen. Som navnet sier så er det en diode og en diode leder strøm den ene veien, hvilket også er tilfelle for en lysdiode (LED – Light Emitting Diode). Det betyr at for at lysdioden skal lyse så må den kobles rett vei i forhold til pluss og minus.



Polaritet: Lysdioder med bein har vanligvis et langt og et kort bein, hvor det lengste beinet normalt er den positive polen. Kanna (huset) til dioden har også gjerne en flat kant ved det negative beinet (kort). Dette kan det være nyttig å vite om dersom beina er klippet like lange i forbindelse med bruk.

Effektforbruk og virkningsgrad: Virkningsgraden for en lyskilde kan måles i lm/W (lysstyrke i lumen pr. tilført elektrisk effekt i Watt). Normalt trekker en lysdiode langt mindre effekt enn en glødelampe ved samme lysintensitet. Den er derfor en mer effektiv lyskilde enn glødelampen og lysstoffrøret. Dette er selvfølgelig også grunnen til at LED i stadig større grad overtar for tradisjonelle lyskilder.

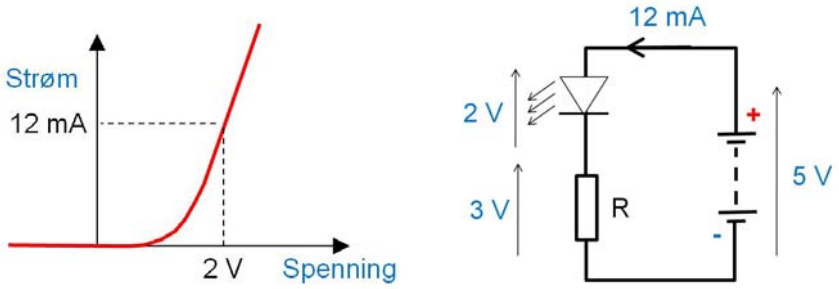
Strømstyrt: Det er en direkte sammenheng mellom strømtrekk og lysintensitet. Det betyr at ved å kontrollere strømstyrken kan vi regulere lysintensiteten. En lyssterk lysdiode vil naturlig nok også trekke mer strøm og tappe batteriet raskere. Det er også viktig at vi begrenser strømmen slik at den ikke brenner opp. For å regulere strømmen kan vi bruk en regulerbar strømkilde, men for laveffekts lysdioder brukes kun en motstand.

Små lysdioder trekker fra 10 – 50 mA, mens sterke lysdioder kan trekke fra 350 mA til flere Ampere. Strømmen i små lysdioder begrenses vanligvis med en motstand på fra 150 – 330 Ω. Mens kraftige lysdioder styres med konstante eller variable strømkilder.

9. https://learn.sparkfun.com/tutorials/light-emitting-diodes-leds?_ga=2.37374368.1867605279.1589209561-856163918.1476131557

Beregning av strøm: La oss se på et enkelt regnestykke for å beregne seriemotstanden til en lysdiode brukt som indikator (ikke lyskilde). For denne typen lysdioder er det stort slingringsmonn mht. valg av seriemotstand.

Ofte kan en få oppgitt *foroverspenningen* over dioden ved en gitt strøm. Disse verdiene stammer fra diodekarakteristikken som viser sammenhengen mellom foroverspenning og strømmen i dioden. I vårt tilfelle er spenningen over dioden 2V når det går en strøm på 12 mA (til venstre på figuren under). Vi har også valgt en spenningskilde lik 5V som er typisk for Arduino.



Vi bruker Kirchhoffs spenningslov og ser at en strøm på 12 mA gir en spenning på 3V over motstanden. Når vi kjenner ønsket strøm og spenning over en motstand, kan vi beregne nødvendig motstandsverdi etter Ohms lov:

$$R = U / I = 3V / 12mA = 250\Omega \quad (9.1)$$

I dette tilfellet kan man sannsynligvis oppnå gode resultater med motstandsverdier fra 150 – 330 Ω.

Datablader: La oss ganske kort se på et eksempel på datablad for en lysdiode omtrent som den vi har omtalt foran.

Tabellen under viser noen interessante parametere, deriblant maksimalverdier.

ITEMS	Symbol	Absolute Maximum Rating	Unit
Forward Current	I_F	20	mA
Peak Forward Current	I_{FP}	30	mA
Suggestion Using Current	I_{su}	16-18	mA
Reverse Voltage (V_R=5V)	I_R	10	uA
Power Dissipation	P_D	105	mW
Operation Temperature	T_{OPR}	-40 ~ 85	℃
Storage Temperature	T_{STG}	-40 ~ 100	℃
Lead Soldering Temperature	T_{SOL}	Max. 260℃ for 3 Sec. Max. (3mm from the base of the expoxy bulb)	

Forward Current angir den maksimale strømstyrken man kan la gå i lysdioden over lengre tid.

Peak Forward Current er absolutt maksimal strøm (her 30mA)som kan tillates i korte perioder (sekunder)

Suggestion Using Current er anbefalt område for strømmen (her 16–18mA).

Power Disipation er maksimalt avgitt effekt fra dioden som beregnes ut fra strømmen i dioden x spenningen over dioden. I vårt eksempel 12 mA x 2V = 24 mW.

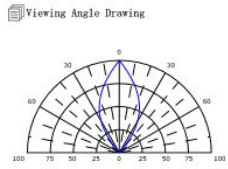
Operation Temperature er anbefalt temperaturområde under vanlig drift. Blir dioden varmere enn 100°C over noe tid, kan den lett gå istykker.

Følgende tabell viser typiske verdier brukt under testing og kan være gode retningslinjer for valg av f.eks. foroverspenning:

ITEMS	Symbol	Test condition	Min.	Typ.	Max.	Unit
Forward Voltage	V_F	I_F=20mA	1.8	---	2.2	V
Wavelenength (nm) or TC(k)	$\Delta\lambda$	I_F=20mA	620	---	625	nm
*Luminous intensity	I_V	I_F=20mA	150	---	200	mcd

Forward Voltage angir typisk spenning over dioden når det går 20 mA. Dette er en viktig parameter når man skal beregne seriemotstanden.

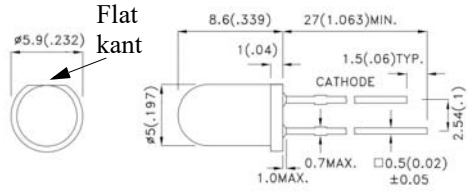
Wavelength angir utstrålt bølgelengde i nanometer ved en strømstyrke lik 20 mA. 620 nm ligger i det røde området.



Luminous intensity angir utstrålt lysintensitet i millicandela ved 20 mA. 200 mcd er ingen kraftig lysdiode, men grei som indikatorlys.

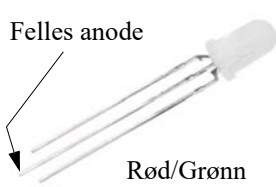
De fleste lysdioder angis også med hvilken åpningsvinkel lyset stråler ut og angis i intensitet i prosent av maksimalverdien, og som funksjon av vinkelen til side for senterlinjen 0°. Vi legger merke til at lysintensiteten er falt til under 50% sett 30° til siden for senterlinjen i vårt eksempel. Noen dioder stråler smalt andre bredt. Diagrammet over til høyre viser en relativt smal utstråling. Utstrålingsvinkelen er gjerne bestemt av hvordan toppen av plasthuset er utformet. En smal utstrålingsvinkel oppnås ved å lage toppen som en linse.

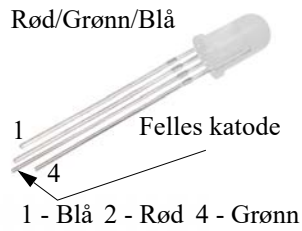
Utforming: Lysdioder leveres i mange forskjellige utforminger, men de vanligste er runde med en tverrsnittsdiameter på huset på 3, 5 eller 10 mm. 5 mm er utvilsomt det vanligste (se figuren til høyre). Legg spesielt merke til den flate kanten av det runde huset som angir den siden katoden (–) er plassert på. Men det finnes også lysdioder med andre former f.eks. rektangulære og overflatemonterte.

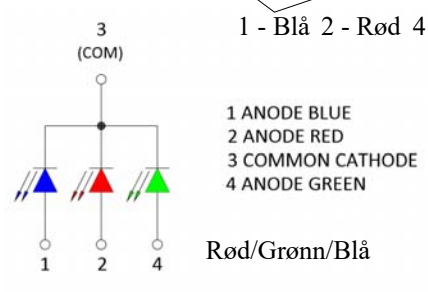


9.1.2 Flerfargede lysdioder – RGB-dioder

Flerfargede lysdioder kan også være utformet på ulikt vis. Men mange er av typen med runde hus med diameter 5mm, hvor det er montert to eller tre lysdioder inne i det samme huset. Disse kan styres individuelt ved hjelp av separate bein, gjerne med en felles katode eller anode. Det siste er verdt å merke seg da man kan bli ganske hjelpeløs dersom man forveksler disse to. Det vanligste er at de har felles katode. Vanligvis finnes de i kombinasjonene Grønn/Gul, Grønn/Rød og Rød/Grønn/Blå (RGB).







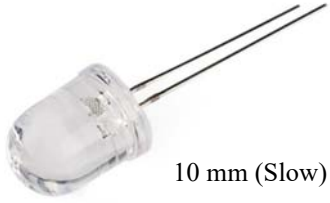
De er ofte utformet med diffust hus siden man ønsker å kunne blande farger. Erfaringer har imidlertid vist at blandingsfarger kan være vanskelig å få fram og blir ikke tydelig med mindre man betrakter dioden fra en spesiell kant. Et diffust hus vil imidlertid hjelpe til at fargene blandes bedre enn for et blankt hus.

9.1.3 Sykliske RGB-dioder

Sykliske RGB LED har som vanlige RGB LED tre lysdioder montert i samme hus. Men istedet for at de styres eksternt ved hjelp av ulike pinner, så har de innebygget en integrert styringskrets som veksler mellom å sette spenning på de ulike diodene, gjene også flere slik at man får fargeblanding (Flashing LED). Figuren til høyre viser hvordan styringskretsen er plassert inne i huset sammen med diodene[10]. De finnes i to typer: FAST og SLOW. I tillegg finnes de med 3, 5 og 10 mm diameter.


3 mm (Slow)


5 mm (Slow)


10 mm (Slow)

10.https://learn.sparkfun.com/tutorials/light-emitting-diodes-leds?_ga=2.37374368.1867605279.1589209561-856163918.1476131557

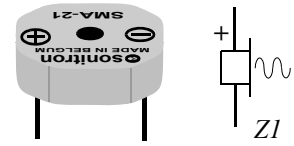
Typisk foroverspenning på disse diodene som selges av Sparkfun[11] er 2V og fargene skifter med noen få sekunders mellomrom.

9.2 Lydgivere – Buzzere og Piezoelektrisk høyttaler

Det er lett å blande sammen det jeg vil kalle buzzere (også kalt aktive buzzere) og piezo-elektriske høyttalere (også kalt passive buzzere). Begge gir fra seg lyd, ligner ofte på hverandre fysisk og begge benytter gjerne en piezo-elektrisk lydgiver.

Piezo-elektriske krystaller har den egenskapen at dersom de påføres en spenning, skjer en mekanisk forandring med krystallet. Dersom spenningen varierer vil krystallet begynne å vibrere i takt med spenningen. Dersom vibrasjonen blir rask nok vil det høres en lyd.

9.2.1 Buzzeren eller piezo-summer (aktiv buzzer)



Figur 9.1 Buzzer.

Buzzeren har to tilkoblingsledninger (bein). Den ene er merket +, den andre –. Tilkoblingsledninger merket med + kobles nærmest den positive polen på batteriet, og tilsvarende for den ledningen som er merket med –. En oscillator i buzzeren skaper en varierende spenning som tilføres det piezo-elektriske materialet og det høres en lyd. Buzzeren gir derfor fra seg en pipetone når den får tilført tilstrekkelig likespenning. Tonefrekvensen til en buzzer er derfor gjerne bestemt under produksjonen. SMA-21LT gir en tone på over 80dB med et strømtrekk på 5mA ved 5V og kan derfor lett drives av en Arduino digital port som kan levere inntil 20mA når utgangen er lagt høyt[12].

Programmering:

Denne kan normalt kobles til en digital port på Arduino'en. Lyden slås på ved å sette porten høy, dvs. til 5V.

```
digitalWrite(buzzerPin, HIGH); // Slå på lyd
digitalWrite(buzzerPin, LOW); // Slå av lyd
```

En bør imidlertid sjekke følgende:

- Er 5V tilstrekkelig spenning til å gi lyd?
- Kan utgangen hos Arduino'en levere tilstrekkelig strøm til å drive den aktive buzzeren.

11. https://www.sparkfun.com/products/11452

12.

9.2.2 Piezo-elektrisk høyttaler eller piezo-element (passive buzzere)

Den viktigste forskjellen mellom en buzzer og en piezo-elektrisk lydgiver er at mens buzzeren genererer den varierende spenningen selv, så har ikke den piezo-elektriske lydgiveren noen egen innebygget oscillator, men må tilføres en varierende spenning utenfra. Den har derfor vanligvis heller ikke noen foretrukket polaritet.

Den kan være utformet som et flatt element som vist til høyre på figuren over eller innebygget i et plasthus. Siden denne ikke har noen innebygget oscillator, så vil den bare gi fra seg et klikk når den tilføres spenning eller spenningen brytes. Skal man få lyd må man generere en ytre varierende spenning med ønsket frekvens.

Fordelen med piezo-elektriske lydgivere er at man kan generere flere toner og dermed gjenskape et mer komplekst lydbilde som inneholder mange frekvenser. Den har likevel en resonansfrekvens som gir høyere lyd enn de andre frekvenser.

Programmering:

Denne kan normalt kobles til en digital port på Arduino'en, men krever at man sender ut en varierende spenning som genereres i mikrokontrolleren til dette er det laget noen funksjoner som setter opp en slik spenning på en av de digitale portene:

```
tone(lydPin, frekvens);
noTone(lydPin);
```

Funksjonen `tone()` slår på lyden, mens `noTone()` slår av lyden. `lydPin` er den pinnen piezo-høyttaleren er tilkoblet og `frekvens` er frekvensen til tonen.

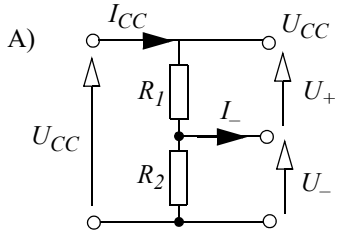
Man kan også velge å legge inn lengden til tonen:

```
tone(lydPin, frekvens, varighet);
```

`varighet` angir lengden på tonen i millisekunder.

9.3 Spenningsdeleren

En spenningsdeler er nyttig for å etablere spenningsnivåer som normalt ikke er tilgjengelig fra strømforsyningen, men også for å konvertere en *variasjon i en motstandsverdi*, f.eks. i en LDR (lysfølsom motstand), til en *spenningsvariasjon* som mikrokontrollere kan registrere på en av sine analoge innganger.



B)

$$U_- = \frac{R_2}{R_1 + R_2} U_{CC}$$

Ukjent R_2

$$R_2 = \frac{U_- \cdot R_1}{U_{CC} - U_-}$$

Ukjent R_1

$$R_1 = \frac{\langle U_{CC} - U_- \rangle \cdot R_2}{U_-}$$

Dersom vi har en spenning U_{CC} (f.eks. batterispenningen), så kan vi ved hjelp av to motstander ta ut en mindre del av denne spenningen. Tegning A i figuren over viser en enkel spenningsdeler. Spenningen U_- vil være en neddeling av spenningen U_{CC}, bestemt av motstandsverdiene til R_1 og R_2.

Vi ønsker å finne sammenhengen mellom U_{CC} og U_- som funksjon av R_1 og R_2.

Vi antar at $I_- = 0$, dvs. spenningsdeleren er ubelastet, hvilket er en god tilnærmelse dersom den kobles til en av inngangene til en mikrokontroller. Vi kan da sette opp en sammenheng mellom strømmer og spenninger i kretsen:

$$U_{CC} = (R_1 + R_2)\, I_{CC} \tag{9.2}$$

$$I_{CC} = \frac{U_{CC}}{(R_1 + R_2)} \tag{9.3}$$

Siden $I_- = 0$, vet vi at hele I_{CC} går gjennom R_2. Vi kan da bruke Ohms-lov på R_2 og finner da et uttrykk for U_- som er spenningen over R_2:

$$U_- = R_2 \cdot I_{CC} = \frac{R_2}{R_1 + R_2} U_{CC} \tag{9.4}$$

Tilsvarende kan vi finne U_+ som er spenningen over R_1:

$$U_+ = R_1 \cdot I_{CC} = \frac{R_1}{R_1 + R_2} U_{CC} \tag{9.5}$$

Vi vet også at:

$$U_{CC} = U_{-} + U_{+} \tag{9.6}$$

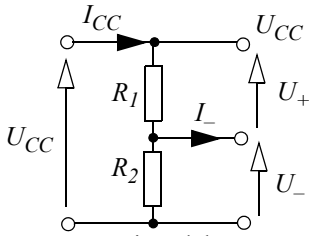
En vanligere situasjon er at vi kjenner spenningen U_{-}, men ønsker å bestemme verdien til en av motstandene. Dersom vi kjenner R_1 og U_{-}, men ønsker å bestemme R_2, kan vi bruke følgende uttrykk:

$$R_2 = \frac{U_{-} \cdot R_1}{U_{CC} - U_{-}} \tag{9.7}$$

Dersom vi kjenner R_2 og U_{-}, men ønsker å bestemme R_1, kan vi bruke følgende uttrykk:

$$R_1 = \frac{\langle U_{CC} - U_{-} \rangle \cdot R_2}{U_{-}} \tag{9.8}$$

På tilsvarende måte kan vi bestemme resistansene når vi kjenner U_{+}.



Spenningsdeleren

La oss se hvordan vi kan få en intuitiv forståelse av hvordan en spenningsdeler fungerer.

Fra Ohms lov vet vi at:

$$U_{+} = R_1 \cdot I_{CC} \tag{9.9}$$

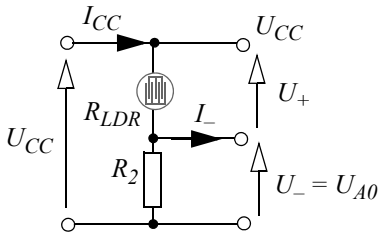
og

$$U_{-} = R_2 \cdot I_{CC} \tag{9.10}$$

Siden strømmen er den samme i begge motstandene, så kan vi dividere de to ligningene med hverandre og skrive:

$$\frac{U_{+}}{U_{-}} = \frac{R_1}{R_2} \tag{9.11}$$

Vi ser altså at forholdet mellom spenningene er forholdet mellom resistansene til de to motstandene.



Spenningsdeleren med LDR

I figuren til venstre har vi byttet ut R_1 med en resistiv sensor, i dette tilfellet en lysfølsom motstand

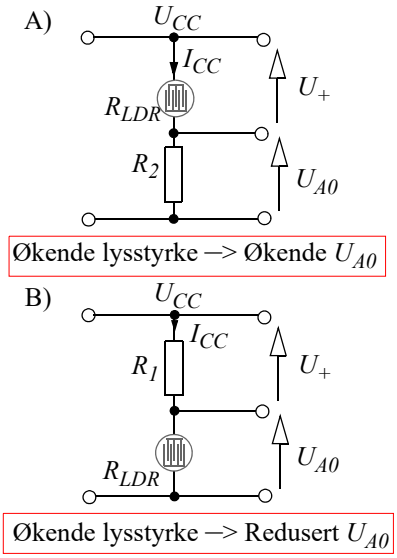
Vi tenker oss at U_{-} kobles til en analog inngang på mikrokontrolleren vi kaller den U_{A0} siden vi kobler den til den analoge inngangen *A0*.

Fra ligning 9.4 vet vi at forholdet mellom spenningen U_{A0} og motstandsverdien til LDR'en, R_{LDR} kan uttrykkes slik:

$$U_{-} = \frac{R_2}{R_{LDR} + R_2} U_{CC} \qquad (9.12)$$

som vi ser så er det ikke nødvendigvis en lineær sammenheng. Men fra ligning (9.8) så vet vi at vi kan regne oss tilbake å finne R_{LDR} med følgende omregning:

$$R_{LDR} = \frac{\langle U_{CC} - U_{A0} \rangle \cdot R_2}{U_{A0}} \qquad (9.13)$$



Spenningsdeleren med LDR

Ofte holder det at vi forstår rent kvalitativt hvordan spenningsdeleren fungerer da vi gjerne bruker LDR i en detektor.

Vi antar at *lysstyrken øker*. Det betyr at motstandsverdien til R_{LDR} *blir mindre.* Når R_{LDR} blir mindre, vil strømmen I_{CC} øke. Siden den samme strømmen går gjennom R_2 forteller Ohms lov oss at spenningen U_{A0} *øker* ($U_{A0} = I_{CC} R_2$).

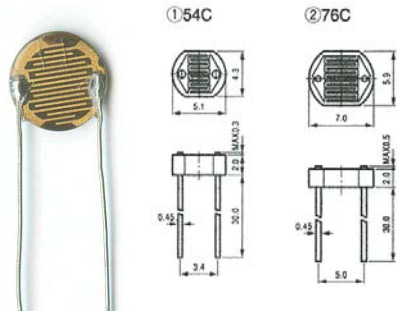
Vi kan imidlertid tenke oss en alternativ plassering av LDR'en ved at den erstatter R_2 istedet for R_1.

Dersom vi nå antar at *lysstyrken øker* og motstandsverdien til R_{LDR} *blir mindre.* Så vil strømmen I_{CC} øke som sist. Siden I_{CC} også går gjennom R_1, forteller Ohms lov oss at spenningen U_+ også vil øke. Siden $U_{A0} = U_{CC} - U_+$ så vil U_{A0} *avta*.

9.4 Lysfølsom sensor – LDR

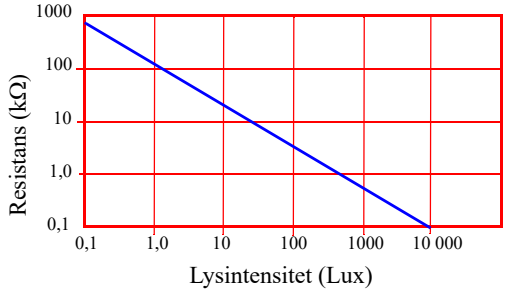
Deteksjon av lys kan gjøres på mange ulike måter. I dette avsnittet skal vi se hvordan vi kan bruke *LDR* (Light Dependent Resistor) som lysfølsomme komponenter.

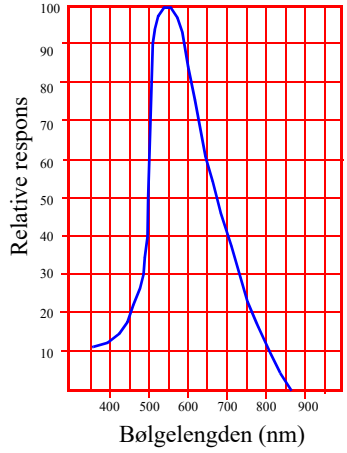
9.4.1 Fotomotstand (LDR - Light Dependent Resistor)



Grunnen til at vi velger å omtale fotomotstander er at den gjennom tidene har vært en gjenganger i mange elektronikkprosjekter og brukes i mange sammenhenger som f.eks. for automatisk tenning av gatebelysning, som enkle lysmålere, som automatisk tenning av frontlyktene på bil, innbruddsalarm ved at en lysstråle brytes o.l.

Fotomotstander har tradisjonelt vært laget av Cadmium-Sulfid (CdS) belagt med fingerelektroder som vist på figuren til høyre. I mørket vil stoffet CdS være omtrent isolerende og kan gi en motstand på over 1 MΩ. Belyses stoffet, kan resistansen i fotomotstanden falle til under 1 kΩ. Årsaken er at fotoner (lys) med tilstrekkelig energi, eksiterer elektroner fra valensbåndet til ledningsbåndet, hvor de kan bevege seg fritt og bidra til ladningstransporten. Effekten er imidlertid ikke like framtredene for alle frekvenser. Til høyre på figuren under ser vi at materialet er spesielt følsomt for lys i det synlige området av spekteret nær 540 nm (nanometer, 10^{-9}m). Vi ser også (til venstre på figuren) at det er en omtrent lineær sammenheng mellom lysstyrken målt i lux og resistansen (begge skalaer er logaritmiske). Økende lysstyrke gir fallende resistans, dvs. økt ledningsevne.





Resistans som funksjon av lysintensitet (venstre), følsomhet som funksjon av bølgelengde.

Lysfølsomme motstander er imidlertid relativt langsomme. En endring i lysstyrken på noen µsek, kan gi en responstid på opp til 100 msek. hos fotomotstanden, men som i mange tilfeller er mer enn godt nok. Bruker man fotomotstander bør man også være klar over følgende [7]:

- Resistansen for fotomotstander kan variere mye fra eksemplar til eksemplar. Dette gjelder spesielt resistansen ved mørke.
- Resistansen hos en LDR vil også være temperaturavhengig
- En LDR kan også brenne opp ved for store strømmer. Dette er noe en bør være oppmerksom på dersom den skal brukes i sterkt sollys. Under slike forhold vil motstanden fall og strømme øke. I slike tilfeller må man sørge for en tilstrekkelig høy seriemotstand slik at strømstyrken og dermed også effekttapet holdes under verdier som kan være skadelige for LDR'en.

- Dersom sensoren går i metning ved sterkt sollys, forsøk å reduser serieresistansen (men pass på effekttapet). Ønsker man større følsomhet i det mørke området øker man serieresistansen

For å konvertere endring i resistans til spenning, kan vi bruke en enkel spenningsdeler (se figuren under). Her trengs normalt ingen målebro eller forsterker for å registrere endring i resistans siden endringen er så stor.

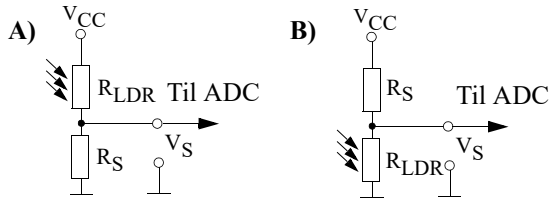
Lysintensitet måles i lux. 1 lux er 1 lumen pr. m^2.

Dette tilsvarer:

- Fullt sollys 11 000 lux (eller ca. 1000 W/m^2)
- Sollyset en tidlig morgen 6 000 lux
- Belysningen i et TV-studio 1 000 lux
- Et godt opplyst kontor 400 lux
- Lyset fra en fullmåne 1 lux

Oppkobling mot ADC

Siden grensnittet til kontrolleren krever en spenning, kobles LDR-motstanden i serie med en motstand som vist i figuren til høyre. Velg verdien på seriemotstanden lik den typiske resistansen til fotomotstanden (LDR) i det aktuelle lysintensitetsområdet der fotomotstanden skal brukes.



$$V_S = \frac{R_S}{R_{LDR} + R_S} V_{CC} \qquad V_S = \frac{R_{LDR}}{R_{LDR} + R_S} V_{CC}$$

Ønsker man maksimal spenningsvariasjon fra dypt mørke til sterkt lys kan seriemotstanden beregnes ut fra følgende ligning [7]:

$$R_S = \sqrt{R_{LDR(light)} \times R_{LDR(dark)}} \tag{9.14}$$

hvor $R_{LDR(light)}$ er resistansen i sterkt lys og $R_{LDR(dark)}$ er resistansen i mørke.

Spenningsnivået V_S beregnes fra formlene som antydet på figuren. Legg merke til at oppkoblingen på tegning A gir økende spenning V_S med økende lysstyrke, mens oppkoblingen i tegning B gir fallende spenning med økende lysstyrke. For en utdypende diskusjon av spenningsdeleren se avsnitt 9.3.

Kalibrering:

Utfordringen blir å finne en omregningsformel fra lysstyrke til spenning:

1. Mål spenning som funksjon av lysstyrke (krever lysmåler)
2. Bruk regresjon for å finne et best tilpasset funksjonsuttrykk

3. Legg omregningsformelen inn i prosessoren

9.5 Temperaturfølsom motstand (NTC- og PTC-motstander)

Metaller vil normalt ha økende resistans med økende temperatur. I et halvledermateriale vil flere ladningsbærere løftes opp i ledningsbåndet slik at ledningsevnen går opp, dvs. at resistansen blir mindre.

De fleste motstandsmaterialer endrer resistans som funksjon av temperaturen. Som regel er dette uønsket, men i noen spesielle tilfeller ønsker man nettopp en slik endring og utformer komponenten og materialet deretter. Slike motstander brukes også i forbindelse med måling eller deteksjon av temperaturendringer, eller til å motvirke uønsket temperaturdrift i elektronisk utstyr.

- NTC – *Negative Temperatur Coefficient*, dvs. at resistansen avtar med økende temperatur.
- PTC – *Positive Temperatur Coefficient*, dvs. at resistansen øker med økende temperatur.

9.5.1 NTC-motstanden

NTC-motstander er laget av et materiale hvis resistivitet varierer sterkt med temperaturen. Som navnet sier (*Negative Temperature Coefficient* – NTC) så avtar resistansen med økende temperatur.

NTC-motstander er derfor vanligvis bygget opp som en polykrystalinsk *halvleder* som kan bestå av en blanding av krom, mangan, jern, kobolt og nikkel, som sintres[13] sammen med et plastisk bindemiddel.

En forenklet sammenheng mellom resistansen (R_{NTC}) og temperaturen (T) kan uttrykkes som:

$$R_{NTC} = Ae^{B/T} \tag{9.15}$$

hvor A og B er *tilnærmet konstante* innen begrensede temperaturområder.

I datablader for NTC-motstander oppgis gjerne resistansen (R_r) for en referansetemperatur (T_r). I et temperaturområde rundt referansetemperaturen antas B-verdien å være tilnærmet konstant ($B_{25/85}$ – B-verdien er tilnærmet konstant innen området 25°C til 85°C).

Vi kan da sette opp følgende to ligninger:

$$R_{NTC} = Ae^{\frac{B_{25/85}}{T}} \tag{9.16}$$

$$R_r = Ae^{\frac{B_{25/85}}{T_r}} \tag{9.17}$$

Hvor den målte motstanden i NTC-motstanden er lik R_{NTC} ved temperaturen T.

13. Sintring betyr at metallpulver knyttes sammen ved hjelp av oppvarming, men uten å smelte.

Ved å eliminere A fra disse uttrykkene, kommer vi fram til følgende sammenheng, løst med hensyn til resistansen R_{NTC} i NTC-resistoren:

$$R_{NTC} = R_r \cdot e^{\left(\frac{B_{25/85}}{T} - \frac{B_{25/85}}{T_r}\right)} \tag{9.18}$$

Dette uttrykket går under betegnelsen *Beta-formelen*.

Når vi skal beregne verdien for en NTC-motstand ved en gitt temperatur, slår vi opp B-verdien, R_r og T_r i databladet, sørger for at de aktuelle temperaturene ligger innenfor området til B-verdien, og beregner R ved å sette inn ønsket temperatur T. Temperaturen angis i grader Kelvin.

Siden det ofte er B for området 25° – 85°C som er oppgitt kan en lett oppleve at man havner på siden av det spesifiserte området, siden vi ofte ønsker å måle i området 0° – 25°C. Siden vi linearisererer og kalibrerer sensoren trenger ikke dette bety så ny for vår anvendelse.

9.5.2 NTCLE201E3103S

Fra databladet[14] for *NTCLE201E3103S* finner vi følgende: R_{25} er referansemotstand (R_r) ved 25 °C (T_r = 298 K):

ELECTRICAL DATA AND ORDERING INFORMATION				
R_{25} (Ω)	R_{25}-TOL. (± %)	$B_{25/85}$ (K)	$B_{25/85}$-TOL. (± %)	SAP MATERIAL AND ORDERING NUMBER
3000	2.18	3977	0.75	NTCLE201E3302SB
5000	2.18	3977	0.75	NTCLE201E3502SB
10 000	2.18	3977	0.75	NTCLE201E3103SB
3000	2.18	3977	0.75	NTCLE300E3302SB
5000	2.18	3977	0.75	NTCLE300E3502SB
10 000	2.18	3977	0.75	NTCLE300E3103SB

Figur 9.2 Datablad for NTC-motstand NTCLE201E3103SB, 3–10 kΩ

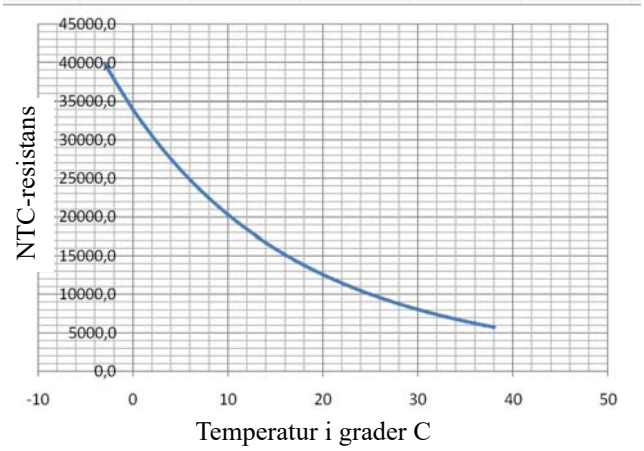
Innsatt disse dataene i formelen vår kan vi skrive:

$$R_{NTC} = 10\text{k} \cdot e^{\left(\frac{3977}{T} - \frac{3977}{298}\right)} \tag{9.19}$$

hvor $B_{25/85}$ = 3977 (NTCLE201E3103SB – 10 kΩ) og referansetemperaturen T_r = 298 K.

14. Databladet er hentet fra: http://www.elfa.se/pdf/60/06027916.pdf

Dersom vi beregner verdier for R_{NTC} i temperaturområdet 25° – 85°C, får vi følgende graf:



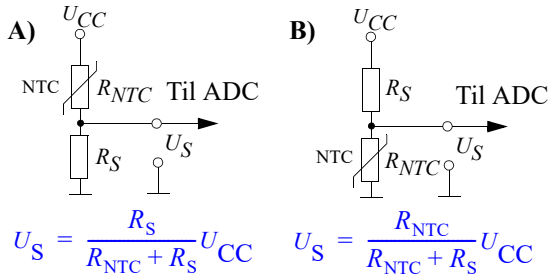
Figur 9.3 Resistansen til NTC motstand som funksjon av temperaturen NTCLE201E3103SB – 10 kΩ

En annen viktig parameter for NTC-motstander, er hvor raskt resistansen endrer seg ved sprang i temperaturen. Denne parameteren betegnes *NTC-motstandens tidskonstant* (τ), og angir den tiden det tar for resistansen og endre seg til 63,2% av den nye resistansen etter at temperaturen har endret seg med et sprang på 1 K (Kelvin) over omgivelsestemperaturen. En antar at temperaturendringen ikke er forårsaket av indre oppvarming på grunn av elektrisk strøm som flyter gjennom motstanden.

Vi var ikke istand til å finne en verdi for tidskonstanten for den aktuelle NTC-motstanden, men det er rimelig å anta at den er under 10 sek. avhengig av omgivelsene (stillestående eller luft i bevegelse).

9.5.3 Oppkobling mot ADC

Siden vi ikke direkte kan måle resistans via portene på kontrollerkortet, men kun kan måle spenning, kobles NTC-motstanden i serie med en motstand som vist i figuren til høyre. Dersom vi velger verdien til seriemotstanden lik referanseverdien til NTC-motstanden (R_{25}), så vil det vise seg at sammenhengen mellom temperatur og spenning blir relativt lineært i området rundt referansetemperaturen (T_r). Spenningsnivået ut av spenningsdeleren, U_S, beregnes fra formlene som antydet på figuren. Legg merke til at oppkoblingen på tegning A gir økende spenning, U_S, med økende temperatur, mens oppkoblingen i tegning B gir fallende spenning, U_S, med økende temperatur.



9.5.4 Optimal seriemotstand

Vi har registrert at motstandsverdien til NTC-motstanden er svært ulineær som funksjon av temperaturen. Nå er det ikke motstanden vi måler, men spenningen på utgangen av spenningsdeleren. Så la oss se hvordan spenningen avhenger av temperaturen og hvordan lineariteten varierer med verdien til seriemotstanden.

På bakgrunn av ligningene foran kan vi sette opp et uttrykk for temperaturen som funksjon av spenningen som ev. kan legges inn i programmet i mikrokontrolleren. Vi har valgt å bruke en NTC-motstand med referanse resistans like 10kΩ.

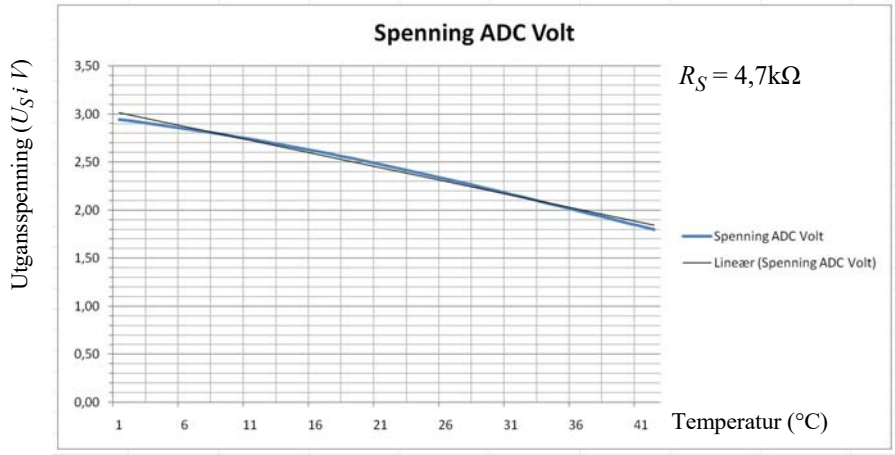
Vi setter:

$$R_{NTC} = 10\text{k} \cdot e^{\left(\frac{3977}{T} - \frac{3977}{298}\right)} \tag{9.20}$$

inn i ligningen:

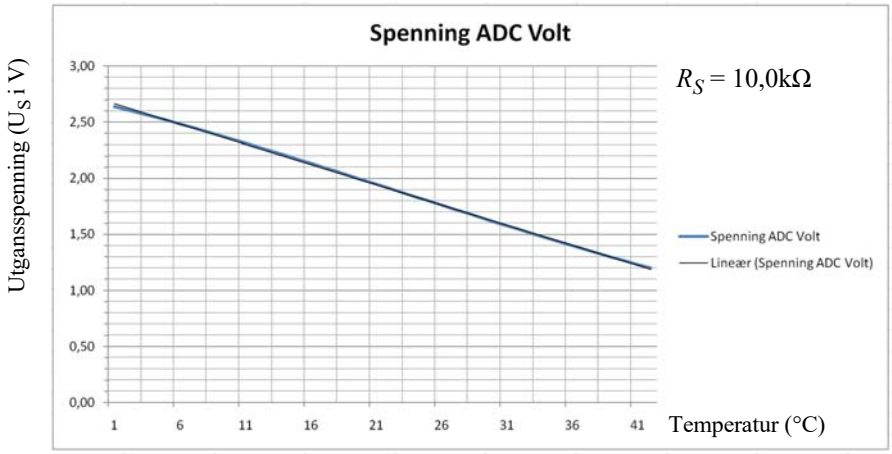
$$U_S = \frac{R_{NTC}}{R_{NTC} + R_S} U_{CC} \tag{9.21}$$

... og vi kan beregne U_S som funksjon av temperaturen for ulike verdier av seriemotstanden, R_S. I figuren under har vi modellert spenningen U_S som funksjon av temperaturen i området 0 – 42°C med en seriemotstand på R_S = 4,7 kΩ, altså ikke den mest optimale for temperaturer nær referansetemperaturen:



Sammen med spenningskurven har vi lagt inn den best tilpassede lineære sammenhengen (tynn rett linje). Vi legger merke til at avvikene er betydelig i endene av området. I temperaturområdet 0 – 42°C er spenningssvinget lik U_{diff} = 0,82 V.

Dersom vi imidlertid velger en seriemotstand R_S = 10kΩ ser kurven langt mer lineær ut.



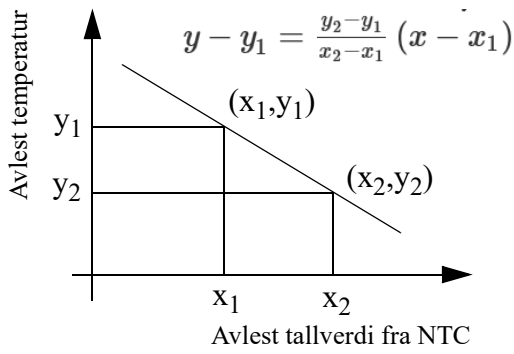
Vi registrerer dermed at avvikene ved endene av området er langt mindre med 10kΩ enn ved 4,7kΩ. Dessuten er spenningssvinget i området 0 – 42°C lik U_{diff} = 1,08 V. Dvs. 10kΩ gjør ikke bare kurven mer lineær, men man utnytter det dynamiske området til ADC'en bedre.

Vi anbefaler derfor å bruke en seriemotstand på 10kΩ for *NTCLE201E3103SB* dersom man ønsker å linearisere funksjonen i området 0 – 40°C. Vi skal senere se hvordan man kan beregne temperaturen direkte med et uttrykk for resistansen i NTC-motstanden som er bedre tilpasset den virkelige sammenhengen.

9.5.5 Kalibrering av temperatursensoren

Denne beregningen antar at det er en omtrent lineær sammenheng mellom måleverdi og temperaturer i det aktuelle temperaturområdet. Dette er ikke langt fra sannheten i et avgrenset område (f.eks. 0 – 40°C) dersom man velger motstandsverdien i seriemotstanden lik referansemotstanden. Under denne betingelsen kan følgende metode benyttes med tilstrekkelig nøyaktighet:

1. Bruk et termometer å mål "virkelig" temperatur f.eks. innendørs (y_1), les av tallverdien (gjerne den digitale verdien) som kommer fra NTC-motstanden (x_1).
2. Ta så mikrokontrolleren med ut samtidig som du logger data (vi antar at det er kaldere ute). Mål "virkelig" temperatur med et termometer (y_2) og les av tallverdien fra NTC-motstanden (x_2).



Figuren til høyre viser hvordan topunktsformelen kan brukes for å finne et uttrykk for sammenhengen mellom målte verdier og temperatur.

Topunktsformelen for lineære ligninger kan også skrives slik:

$$y = ((y_2 - y_1)/(x_2 - x_1)) * (x - x_1) + y_1 \quad (9.22)$$

Hvis $k = (y_2 - y_1)/(x_2 - y_2)$ (stigningskoeffisienten), kan vi skrive ligningen slik:

$$y = k\,(x - x_1) + y_1 \quad (9.23)$$

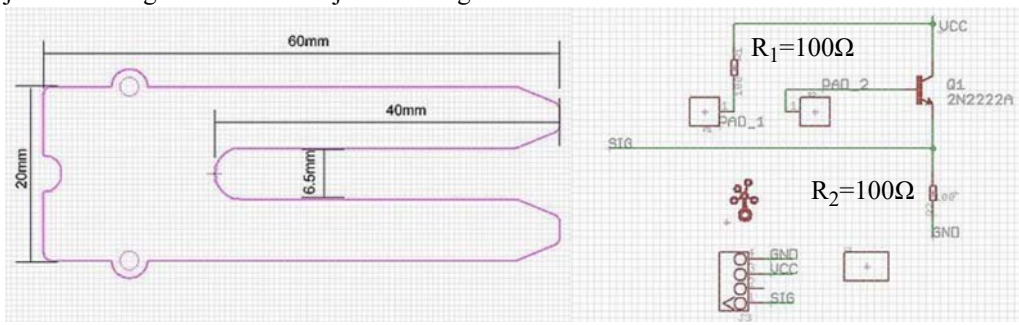
Sett verdiene inn i formelen over og finn et uttrykk for y (temperatur i °C) som funksjon av tallverdien hentet fra AD-konverteren som måler spenningen fra NTC-motstanden (x).

9.6 Fuktighetssensorer for bruk i jord

Det finnes flere varianter for å måle fuktigheten i jord. Her skal vi beskrive et par stykker.

9.6.1 SeeedStudio Grove – Moisture Sensor (RB-See-186)[15]

Sensoren er laget for bruk til å måle fuktighet i jord med tanke på å avdekke planters behov for vanning både innendørs og utendørs i en hage. Sensoren er resistiv og baserer seg på endring av jordas ledningsevne som funksjon av fuktighet.



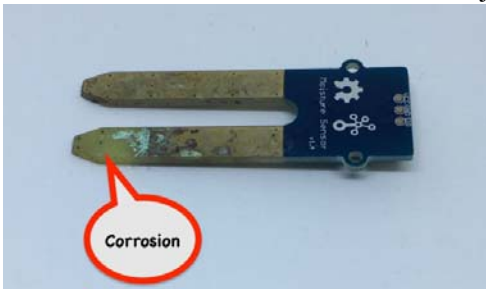
Sensoren består av to metallplater som henholdsvis er koblet til basen på en transistor (2N2222) og til Vcc (supplyspenning) via en resistor på 100Ω. Utgangssignalet er tatt ut over emittermotstanden slik at trinnet fungerer som en emitterfølger. Jo mer fuktighet jorda inneholder jo større strøm går det i transistoren og jo høyere verdi måles på utgangen. Spenningsområdet og maksimal strømstyrke er oppgitt i tabellen under.

Item	Min	Typical	Max	Unit
Voltage	3.3	/	5	V
Current	0	/	35	mA

15.https://www.robotshop.com/media/files/pdf/grove-moisture-sensor-sen92355p.pdf

Sensoren kobles til Vcc og GND samt en analog inngang. Spenningen som måles på utgangen (SIG) vil være et mål for fuktigheten i jorda.

Ulempen med denne typen sensorer er at den korroderer når den står i jord over noe tid[16].



9.6.2 Capacitive Soil Moisture Sensor

Denne sensoren baserer seg ikke den fuktige jordas ledningsevne, men at en kapasitet i sensoren endres med jordas vanninnhold. Det er derfor ikke noe nakent metall som kommer i kontakt med den fuktige jorda. Den er ikke så nøyaktig som den resistive sensoren, men den er mer robust og billig å framstille.

Sensoren skal ha 3,3 V og vil levere en spenning på utgangen avhengig av graden av fuktighet.

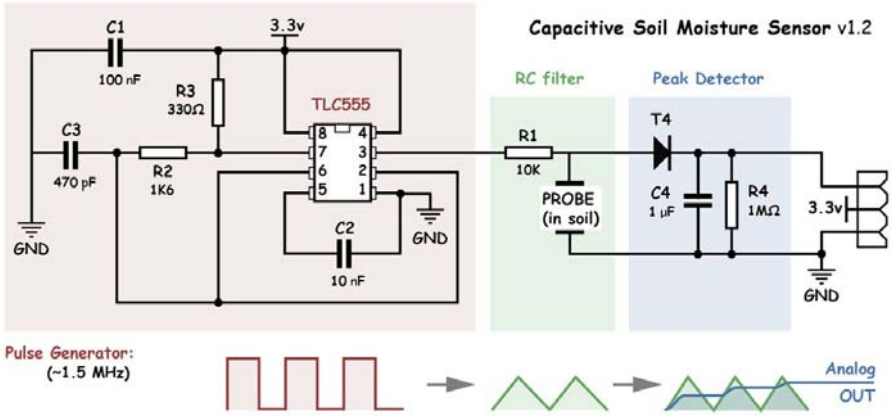
Målinger antyder at den gir:

- 2V i luft (helt tørr)
- 1V i vann (helt fuktig)

Virkemåten kan beskrives slik: Den delen av sensoren som stikkes ned i jorden består av to kobberplater som danner en kondensator. De to platene er dekket med lakk slik at de er isolert fra den fuktige jorda. Verdien til kondensatoren vil endre seg avhengig av omgivelsene, da spesielt fuktigheten i jorda. Sensoren består blant annet av en timerkrets (TLC555) som kan levere et tog av elektriske firkantpulser på ca. 1,5 MHz. Kondensatoren utgjør en del av et elektrisk lavpass-filter som gjør at firkantpulsene blir omdannet til trekantpulser hvor høyden av trekantpulsene bestemmes av filteret. Når verdien til kondensatoren endrer seg avhengig av fuktigheten i jorda, vil filteret endre seg og igjen gjøre

16.https://www.switchdoc.com/2020/06/tutorial-capacitive-moisture-sensor-grove/

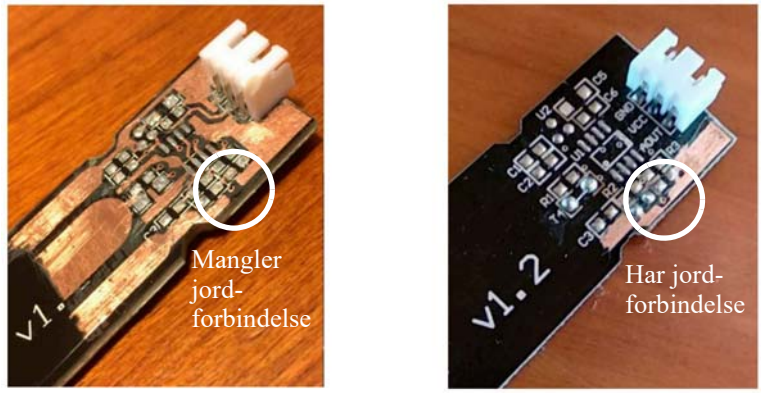
at høyden på trekantene endres. Til slutt er kretsen utstyr med en toppdetektor som måler høyde av trekantene. Det er denne høyden som er den spenningen sensoren leverer og som måles av mikrokontrolleren. Dette er vist på figuren under[17]:



Feil i noen av sensorene

Det er funnet feil i flere sensorer. Følgende er oppdaget:

- På enkelte av sensorene mangler C_1, dvs. den er ikke montert fra produsenten[18]. Feilen kan fikses ved å lodde på en kondensator på 100nF. Om man ikke har en overflate montert komponent kan man ev. bruk en vanlig hullmontert.
- På andre så mangler R_4 jordforbindelse. Den er koblet til et jordplan som har mistet forbindelsen til jord og henger "løst". Se figuren under.

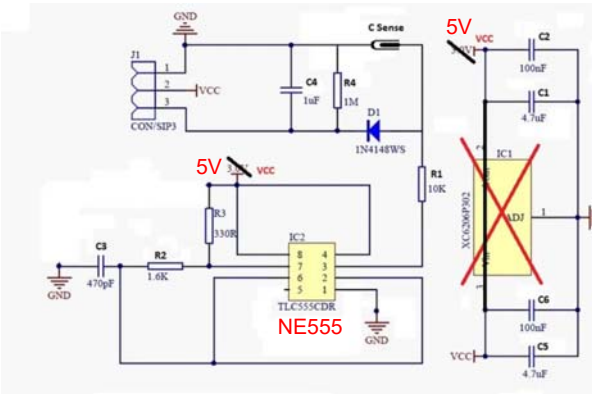


(a) (b)

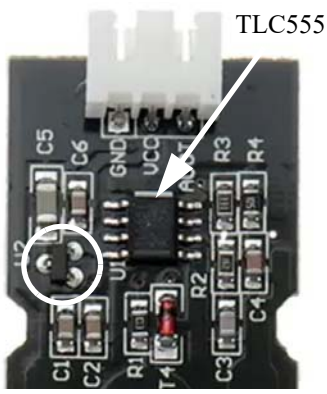
17. https://thecavepearlproject.org/2020/10/27/hacking-a-capacitive-soil-moisture-sensor-for-frequency-output/
18. https://www.youtube.com/watch?v=QGCrtXf8YSs

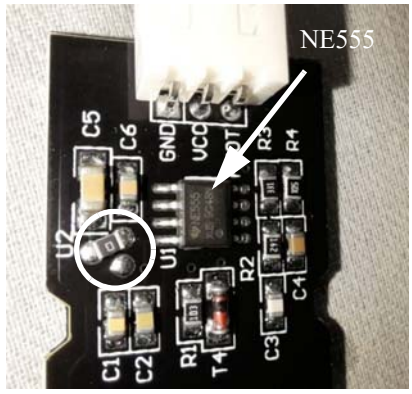
I a) så kan vi skimte en smal kobberfolie forbindelse en forbindelse som er borte på figur b)[19]. Problemet kan løses på flere måter: Enten ved å lodde seg forbi bruddet, eller ved å koble en 1MOhms motstand mellom Aout og jord.

En mer alvorlig feil er at enkelte av fuktighetssensorene leveres med NE555 i stedet for TLC555. Den første krever 4,5 V og leverer en spenning på utgangen på over 3,3V. TLC555 derimot virker ned til 3,3 V og leverer en spenning på utgangen under 3V og passer derfor godt til å arbeide sammen med ESP32. Sammen med TLC555 er det montert en regulator som senker en ev. supplyspenning fra 5V og ned til 3,3 V slik at kretsen både kan brukes for 5V og 3,3V.



Med NE555 så er regulatoren fjernet og erstattet med en 0 Ω motstand som vist i figuren til over til høyre[20]. Figuren under viser hvordan regulatoren er fjernet og erstattet med en 0 Ω motstand.

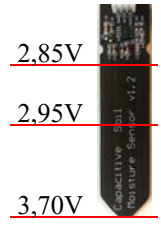




Dersom vi forsyner sensoren med 5V så måler vi følgende:

- Tørr: 3,55 V, og synes å driver noe opp verdi (3,55 – 3,65V)
- Vann: 2,85 (med vannstand ca. 3 mm under elektronikken, se figur til høyre.)

Det ser ikke ut til at den nedre grenseverdien forandrer seg så mye fra vannet står opp under elektronikken til vannstanden passerer under halvveis opp på sensoren, se figuren til høyre.



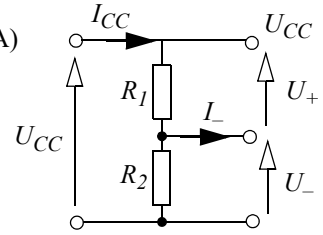
19. file:///C:/Users/Nikro/AppData/Local/Temp/MicrosoftEdgeDownloads/55e3b9dd-1d52-4772-831c-cf552ac04756/sensors-20-03585%20(1).pdf
20. https://imgur.com/gallery/1G9Lqwc

Løsning på problemet.

La oss se hvordan vi kan løse problemet når sensoren er utstyrt med NE555 og spenningsregulatoren er erstattet med en 0 Ω motstand. At den 1 MΩ motstanden på utgangen ikke er jordet og dermed ikke inngår i kretsen er en fordel. Dersom den har jordforbindelse så må motstanden fjernes.

Under disse betingelsen er det enkleste å sette på en spenningsdeler på utgangen. Dette vil gjøre at spenningsområdet vil reduseres noe, men er likevel til å leve med. Siden den leverte sensoren har en 1 MΩ motstand på utgangen som ikke har jordkontakt, så er det lett å erstatte denne med en ekstern spenningsdeler som senker spenningen til under 3V.

A)



B)

$$U_- = \frac{R_2}{R_1 + R_2} U_{CC}$$

Ukjent R_2

$$R_2 = \frac{U_- \cdot R_1}{U_{CC} - U_-}$$

Ukjent R_1

$$R_1 = \frac{\langle U_{CC} - U_- \rangle \cdot R_2}{U_-}$$

Vi setter opp følgende krav:

- $R_1 + R_2 = 1M\Omega$ (summen av de to motstandene erstatter den interne 1 MΩ motstanden)
- $U_{CCmax} = 3{,}70\ V$
- $U_{-max} = 3{,}0\ V$

Vi løser lign. mht. R_2 og får:

$$R_2 = U_-/U_{CC} \cdot 1M\Omega = 3{,}0/3{,}70 \cdot 1M\Omega = 810\ k\Omega \tag{9.24}$$

$$R_1 = 1M\Omega - 810\ k\Omega = 190\ k\Omega \tag{9.25}$$

De nærmeste standardverdiene er $R_2 = 820\ k\Omega$ og $R_1 = 180\ k\Omega$

Dette gir følgende resultater:

- Tørr: 2,55 V
- Vann: 2,10 (3 mm under elektronikken)

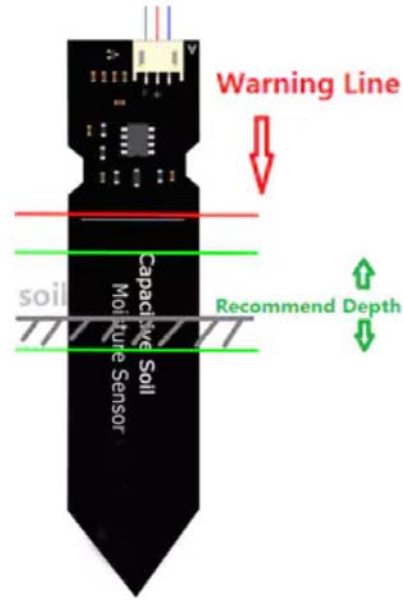
Målinger gjort med tørr sensor og sensor i vann gir følgende verdier ut av AD-konverteren:

- Tørr: 3468 – 3448 i løpet av 5 min (ligger stille på bordet)
- Vann: 2540 – 2560 i løpet av 5 min. (12 mm under elektronikken)

For å få stabile målinger velger vi å midle over 10 000 målinger. Selv med så mange målinger gjøres måleserien unna på ca. 0,5 sek. Det kan derfor se ut til at vi har en betydelig margin, med mindre at den driver flere hundre i løpet av flere døgn.

Beskytt elektronikken

Selv om selve sensoren er beskyttet mot fuktighet så er ikke elektronikken øverst isolert. Den bør derfor ikke komme i kontakt med den fuktige jorda. Figuren til høyre viser anbefalt dybde mellom de grønne linjene ("Recommend Depth")[21]. Elektronikken kan ev. beskyttes med et lag med lakk eller silikon.



Kalibrering

Dersom man ønsker nøyaktige målinger må sensoren kalibreres. En meget enkel kalibrering kan være å måle med tørr sensor i luft (0% fuktighet) og notere målt spenning. Dernest å måle spenningen når sensoren er dyppet i vann (100% fuktighet). Deretter kan man anta en lineær sammenheng mellom disse ytterpunktene. Dette er imildertid en temmelig røff måling.

Ønsker man en mer nøyaktig måling må man lage samler av jord med kjent fuktighet i %. Ved hjelp av slike prøver kan man da finne punkter langs en standard kurve for denne spesielle jorden. Ved å interpolere målinger mellom de kjente punktene kan man bestemme nøyaktige verdier for jordas fuktighet. For en mer detaljert beskrivelse se referanse [8].

21. https://www.circuitschools.com/interface-capacitive-soil-moisture-sensor-v1-2-with-arduino-lcd-and-oled/

10 Referanser

[1] Mer informasjon om Sparkfun Invention's kit:
https://www.sparkfun.com/products/11227

[2] Koblingsskjema for Arduino UNO:
http://arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf.

[3] For mer informasjon om Arduino UNO R3 layout:
http://arduino.cc/en/Main/ArduinoBoardUno/

[4] For nedlasting av programeditoren IDE for Arduino:
http://arduino.cc/hu/Main/Software

[5] For nedlasting av Referansemanualen for Arduino C++
http://arduino.cc/en/Reference/HomePage

[6] Skolelaboratoriets blå hefteserie:
https://www.ntnu.no/skolelab/bla-hefteserie

[7] Elliot Williams, *Make: AVR Programming – Learing to Write Software for Hardware*, Make Community, LLC; 1 edition (February 25, 2014)

[8] Kalibrering av kapasitiv fuktighetsmåling (Joshua Hrisko, 15. juni, 2020):
https://makersportal.com/blog/2020/5/26/capacitive-soil-moisture-calibration-with-arduino

Vedlegg A Løsningsforslag på introduksjonsoppgaver

Oppgavene bygger på oppgave 1 i SIK guiden og gir stadig nye utfordringer til denne innledende oppgaven.

A.1 SOS morsesender

A.1.1 Deloppdrag 1 – Blinkende LED

```
// Denne koden skriver elevene inn for hånd
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{

  digitalWrite(13, HIGH);          // Turn on the LED
  delay(1000);                     // Wait for one second
  digitalWrite(13, LOW);           // Turn off the LED
  delay(1000);                     // Wait for one second
}
```

A.1.2 Deloppdrag 2 – Enkelt trafikklys

```
// Innfører variablene pinDiode og blinkDelay

int pinDiode = 13;
int blinkDelay = 500;

void setup()
{
  pinMode(pinDiode, OUTPUT);
}

void loop()
{

  digitalWrite(pinDiode, HIGH);         // Turn on the LED
  delay(blinkDelay);                    // Wait for one second
  digitalWrite(pinDiode, LOW);          // Turn off the LED
```

```
  delay(blinkDelay);                    // Wait for one second
}
```

A.1.3 Deloppdrag 3 – Kode som blinker SOS

```
// SOS

int pinDiode = 13;

int prikkLengde = 200;              // Definerer lengen av en prikk, 200 msek
int strekLengde = 600;              // Definerer lengen av en strek, 600 msek
int tegn_tegnLengde = 200;          // Definerer tidsrommet mellom to tegn, 200
msek
int bokstav_bokstavLengde = 600;    // Definerer tidsrommet mellom to bokstaver
600 msek
int ord_ordLengde = 1800;           // // Definerer tidsrommet mellom to ord 1800
msek

void setup()
{
  pinMode(pinDiode, OUTPUT);
}

void loop()
{

  // Send S som tre prikker ***
  digitalWrite(pinDiode, HIGH);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);
```

```
  delay(bokstav_bokstavLengde);

  // Sende O som tre streker - - -
  digitalWrite(pinDiode, HIGH);
  delay(strekLengde);
  digitalWrite(pinDiode, LOW);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  delay(strekLengde);
  digitalWrite(pinDiode, LOW);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  delay(strekLengde);
  digitalWrite(pinDiode, LOW);

  delay(bokstav_bokstavLengde);

  // Send S som tre prikker ***
  digitalWrite(pinDiode, HIGH);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);

  // Mellomrom ord - ord
  delay(ord_ordLengde);
}
```

A.1.4 Deloppdrag 4 – SOS med variabel hastighet

```
// SOS med variabel hastighet

int pinDiode = 13;

int hastighet = 50;                // Definerer hastigheten
int prikkLengde = 2*hastighet;     // Definerer prikklengde relativt til
hastigheten
int strekLengde = 6*hastighet;     // Definerer streklengde relativt til
hastigheten
int tegn_tegnLengde = 2*hastighet;
int bokstav_bokstavLengde = 6*hastighet;
int ord_ordLengde = 18*hastighet;

void setup()
{
  pinMode(pinDiode, OUTPUT);
}

void loop()
{

  // S ***
  digitalWrite(pinDiode, HIGH);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);

  delay(bokstav_bokstavLengde);
```

```
  // O - - -
  digitalWrite(pinDiode, HIGH);
  delay(strekLengde);
  digitalWrite(pinDiode, LOW);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  delay(strekLengde);
  digitalWrite(pinDiode, LOW);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  delay(strekLengde);
  digitalWrite(pinDiode, LOW);

  delay(bokstav_bokstavLengde);

  // S ***
  digitalWrite(pinDiode, HIGH);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);

  // Mellomrom ord - ord
  delay(ord_ordLengde);
}
```

A.1.5 Deloppdrag 5 – SOS med lys og lyd

```
// SOS med lys og lyd

int pinDiode = 13;
int pinBuzzer = 9;

int hastighet = 50;
int prikkLengde = 2*hastighet;
int strekLengde = 6*hastighet;
int tegn_tegnLengde = 2*hastighet;
int bokstav_bokstavLengde = 6*hastighet;
int ord_ordLengde = 18*hastighet;

int frekvens = 880; // Hertz

void setup()
{
  pinMode(pinDiode, OUTPUT);
  pinMode (pinBuzzer, OUTPUT);
}

void loop()
{
  // S ***
  digitalWrite(pinDiode, HIGH);
  tone(pinBuzzer, frekvens);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);
  noTone(pinBuzzer);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  tone(pinBuzzer, frekvens);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);
  noTone(pinBuzzer);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
```

```
tone(pinBuzzer, frekvens);
delay(prikkLengde);
digitalWrite(pinDiode, LOW);
noTone(pinBuzzer);

delay(bokstav_bokstavLengde);

// O - - -
digitalWrite(pinDiode, HIGH);
tone(pinBuzzer, frekvens);
delay(strekLengde);
digitalWrite(pinDiode, LOW);
noTone(pinBuzzer);

delay(tegn_tegnLengde);

digitalWrite(pinDiode, HIGH);
tone(pinBuzzer, frekvens);
delay(strekLengde);
digitalWrite(pinDiode, LOW);
noTone(pinBuzzer);

delay(tegn_tegnLengde);

digitalWrite(pinDiode, HIGH);
tone(pinBuzzer, frekvens);
delay(strekLengde);
digitalWrite(pinDiode, LOW);
noTone(pinBuzzer);

delay(bokstav_bokstavLengde);

// S ***
digitalWrite(pinDiode, HIGH);
tone(pinBuzzer, frekvens);
delay(prikkLengde);
digitalWrite(pinDiode, LOW);
noTone(pinBuzzer);

delay(tegn_tegnLengde);
```

```
  digitalWrite(pinDiode, HIGH);
  tone(pinBuzzer, frekvens);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);
  noTone(pinBuzzer);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  tone(pinBuzzer, frekvens);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);
  noTone(pinBuzzer);

  // Mellomrom ord - ord
  delay(ord_ordLengde);
}
```

A.1.6 Deloppdrag 6 – SOS med bruk av funksjoner

```
// SOS med variabel hastighet og bruk av funksjoner

int pinDiode = 13;
int pinBuzzer = 9;

int hastighet = 50;
int prikkLengde = 2*hastighet;
int strekLengde = 6*hastighet;
int tegn_tegnLengde = 2*hastighet;
int bokstav_bokstavLengde = 6*hastighet;
int ord_ordLengde = 18*hastighet;

int frekvens = 440; // Hertz

void setup()
{
  pinMode(pinDiode, OUTPUT);
  pinMode (pinBuzzer, OUTPUT);
}

void loop()
{
```

```
 s();                              // Kall funksjonen S
  delay(bokstav_bokstavLengde);
 o();                              // Kall funksjonen O
  delay(bokstav_bokstavLengde);
  s();                             // Kall funksjonen S
  // Mellomrom ord - ord
  delay(ord_ordLengde);
  while (true);
}

void s ()
{
   // S ***
  digitalWrite(pinDiode, HIGH);
  tone(pinBuzzer, frekvens);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);
  noTone(pinBuzzer);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  tone(pinBuzzer, frekvens);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);
  noTone(pinBuzzer);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  tone(pinBuzzer, frekvens);
  delay(prikkLengde);
  digitalWrite(pinDiode, LOW);
  noTone(pinBuzzer);
}

void o()
{
     // O - - -
  digitalWrite(pinDiode, HIGH);
  tone(pinBuzzer, frekvens);
```

```
  delay(strekLengde);
  digitalWrite(pinDiode, LOW);
  noTone(pinBuzzer);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  tone(pinBuzzer, frekvens);
  delay(strekLengde);
  digitalWrite(pinDiode, LOW);
  noTone(pinBuzzer);

  delay(tegn_tegnLengde);

  digitalWrite(pinDiode, HIGH);
  tone(pinBuzzer, frekvens);
  delay(strekLengde);
  digitalWrite(pinDiode, LOW);
  noTone(pinBuzzer);
}
```

Vedlegg B Løsningsforslag guidede oppgaver

B.1 Trafikklys

B.1.1 Deloppdrag 0 – Blinkende LED

```
// Oppdrag 0 innledende øvelse - Blinkende lysdiode


void setup()                // Setup() funksjonen kjøres bare en gang ved
oppstart
{
  pinMode(13, OUTPUT);    // Port 13 defineres som utgang
  Serial.begin(9600);     // Setter kommunikasjonshastigheten til 9600
tegn i sekundet
}


void loop()
{
  digitalWrite(13, HIGH); // Slå på LED
  Serial.println("On");
  delay(1000);    // Vent ett sekund
  digitalWrite(13, LOW);  // Slå av LED
  Serial.println("Off");
  delay(1000);      // Vent ett sekund
}
```

B.1.2 Deloppdrag 1 – Enkelt trafikklys

```
/*
Trafikklys Oppgave 1
Lag et trafikklys som lyser slik som et trafikklys skal lyse
når det skifter fra rødt til grønt og tilbake:
-
*/
int ledPinRed = 13; // Kontakt rød LED til pinne pin 13
int ledPinYellow = 12; // Kontakt rød LED til pinne pin 12
int ledPinGreen = 11; // Kontakt rød LED til pinne pin 11
```

```
// Setup definerer hva som er utganger og hva som er innganger
void setup() {
pinMode(ledPinRed, OUTPUT);
pinMode(ledPinYellow, OUTPUT);
pinMode(ledPinGreen, OUTPUT);
}

// loop() kjører om og om igjen i det uendelige
void loop()
{
digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode

delay(5000);

digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode

delay(1000);

digitalWrite(ledPinRed, LOW); // Slå av rød lysdiode
digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
digitalWrite(ledPinGreen, HIGH); // Slå på grønn lysdiode

delay(5000);

digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode

delay(1000);

digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
}
```

B.1.3 Deloppdrag 2 – Bestilling av grønt

```
/*
Trafikklys Oppgave 2
Lag et trafikklys som lyser slik som et trafikklys skal lyse
```

```
når det skifter fra rødt til grønt og tilbake.
Skifte skal imidlertid kun inntreffe ved trykk på knappen.
*/
int ledPinRed = 13; // Kontakt rød LED til pinne pin 13
int ledPinYellow = 12; // Kontakt rød LED til pinne pin 12
int ledPinGreen = 11; // Kontakt rød LED til pinne pin 11

int knappBestilling = 7; // Kontakt knapp for bestilling av grønn mann
boolean bestilling = false; // Boolsk variabel som holde bestilling

// Setup definerer hva som er utganger og hva som er innganger
void setup() {
pinMode(ledPinRed, OUTPUT);
pinMode(ledPinYellow, OUTPUT);
pinMode(ledPinGreen, OUTPUT);
pinMode(knappBestilling, INPUT);
}

// loop() kjører om og om igjen i det uendelige
void loop()
{
bestilling = digitalRead (knappBestilling); //Les av knappens innstilling

if (bestilling == false)
 {
 digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode

 delay(5000);

 digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode

 delay(1000);

 digitalWrite(ledPinRed, LOW); // Slå av rød lysdiode
 digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
 digitalWrite(ledPinGreen, HIGH); // Slå av grønn lysdiode
```

```
 delay(5000);

 digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
 digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode

 delay(1000);

 digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode

 bestilling = true; // Sett bestilling til null/falsk
 }
else
 {
 digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode
 }
}
```

B.1.4 Deloppdrag 3 – Lag en funksjon for skiftet til grønt og tilbake

```
/*
Trafikklys Oppgave 3
Lag et trafikklys som lyser slik som et trafikklys skal lyse
når det skifter fra rødt til grønt og tilbake.
Skifte skal imidlertid kun inntreffe ved trykk på knappen.
Det er laget en funksjon som utfører skiftet fra gønrt til rødt og tilbake
*/

int ledPinRed = 13; // Kontakt rød LED til pinne pin 13
int ledPinYellow = 12; // Kontakt rød LED til pinne pin 12
int ledPinGreen = 11; // Kontakt rød LED til pinne pin 11

int knappBestilling = 7; // Kontakt knapp for bestilling av grønn mann
boolean bestilling = false; // Boolsk variabel som holde bestilling

// Setup definerer hva som er utganger og hva som er innganger
```

```
void setup() {
pinMode(ledPinRed, OUTPUT);
pinMode(ledPinYellow, OUTPUT);
pinMode(ledPinGreen, OUTPUT);
pinMode(knappBestilling, INPUT);
}

// loop() kjører om og om igjen i det undelige

void loop()
{
bestilling = digitalRead (knappBestilling); //Les av knappens innstilling

if (bestilling == false)
 {
   gront_lys();        // Kall funksjonen gront_lys()

   bestilling = true; // Sett bestilling til null/falsk
 }
else
 {
   digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode
 }
}

void gront_lys()
{
  digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode

 delay(5000);

 digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode

 delay(1000);

 digitalWrite(ledPinRed, LOW); // Slå av rød lysdiode
```

```
 digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
 digitalWrite(ledPinGreen, HIGH); // Slå av grønn lysdiode

 delay(5000);

 digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
 digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode

 delay(1000);

 digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
}
```

B.1.5 Deloppdrag 4 – Blinkende grønt

```
/*
Trafikklys Deloppdrag 4
Lag et trafikklys som lyser slik som et trafikklys skal lyse
når det skifter fra rødt til grønt og tilbake.
Skifte skal imidlertid kun inntreffe ved trykk på knappen. Fra nå av skal
det grønne lyset lyse konstant  i 8 sek. for deretter å etterfølges av 5
blink
med et halvt sekund på og et halvt sekund av. De gule lysene skal være
på i 1 sek,
og det skal gå 5 sek. etter at det er trykket på knappen før det skifter
til grønt*/

int ledPinRed = 13; // Kontakt rød LED til pinne pin 13
int ledPinYellow = 12; // Kontakt rød LED til pinne pin 12
int ledPinGreen = 11; // Kontakt rød LED til pinne pin 11
int knappBestilling = 7; // Kontakt knapp for bestilling av grønn mann
boolean bestilling = false; // Boolsk variabel som holde bestilling

// Definisjon av konstanter
const int RODTID = 5000;       // 5 sekunder før det blir grønt
const int GULTID = 1000;       // 1 sekund gultid
const int ANTALLBLINK = 5;     // 5 blink, ett blink er på og av.
```

```
const int LENGDEBLINK = 1000;  // 1 sek, inkluderer halvparten av og alv-
parten på
const int GRONNTID = 8000;     // Lengden det skal være fast grønn før
det begynner å blinke.


// Setup definerer hva som er utganger og hva som er innganger

void setup() {
pinMode(ledPinRed, OUTPUT);
pinMode(ledPinYellow, OUTPUT);
pinMode(ledPinGreen, OUTPUT);
pinMode(knappBestilling, INPUT);
}

// loop() kjører om og om igjen i det undelige

void loop()
{

bestilling = digitalRead (knappBestilling); //Les av knappens innstilling

if (bestilling == false)
 {
   gront_lys();        // Utfør endring til grønt lys og tilbake
   bestilling = true; // Sett bestilling til null/falsk
 }
 else
 {
   digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode
 }
}

void gront_lys()
{
  digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode
```

```
  delay(RODTID);

  digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode

  delay(GULTID);

  digitalWrite(ledPinRed, LOW); // Slå av rød lysdiode
  digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
  digitalWrite(ledPinGreen, HIGH); // Slå på grønn lysdiode

  delay(GRONNTID);

  for (int i = 0; i < ANTALLBLINK; i++)
   {
    digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
    delay(LENGDEBLINK/2);
    digitalWrite(ledPinGreen, HIGH); // Slå på grønn lysdiode
    delay(LENGDEBLINK/2);
  }

   digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
   digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode

   delay(GULTID);

   digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
}
```

B.1.6 Deloppdrag 5 – Universal utforming

```
/*
Trafikklys Deloppdrag 5
Lag et trafikklys som lyser slik som et trafikklys skal lyse
når det skifter fra rødt til grønt og tilbake.
Skifte skal imidlertid kun inntreffe ved trykk på knappen. Fra nå av skal
```

```
det grønne lyset lyse konstant  i 8 sek. for deretter å etterfølges av 5 blink
med et halvt sekund på og et halvt sekund av. De gule lysene skal være på i 1 sek,
og det skal gå 5 sek. etter at det er trykket på knappen før det skifter til grønt
Det skal gis et gort lydpuls hver gang det blinkende grønne lyset skifter*/


int ledPinRed = 13;    // Kontakt rød LED til pinne pin 13
int ledPinYellow = 12; // Kontakt rød LED til pinne pin 12
int ledPinGreen = 11;  // Kontakt rød LED til pinne pin 11
int tonePin = 6;       // Pinnen der den passive buzzeren er tilkoblet
int knappBestilling = 7; // Kontakt knapp for bestilling av grønn mann
boolean bestilling = false; // Boolsk variabel som holde bestilling


// Definisjon av konstanter


  const int RODTID = 5000;       // 5 sekunder før det blir grønt
  const int GULTID = 1000;       // 1 sekund gultid
  const int ANTALLBLINK = 5;     // 5 blink, ett blink er på og av.
  const int LENGDEBLINK = 1000;  // 1 sek, inkluderer halvparten av og alvparten på
  const int GRONNTID = 8000;     // Lengden det skal være fast grønn før det begynner å blinke.
  const int KORTLYDPULS = 100;   // Kort lydpuls er satt til 100 msek.


// Setup definerer hva som er utganger og hva som er innganger


void setup() {
pinMode(ledPinRed, OUTPUT);
pinMode(ledPinYellow, OUTPUT);
pinMode(ledPinGreen, OUTPUT);
pinMode(tonePin, OUTPUT);
pinMode(knappBestilling, INPUT);
}
```

```
// loop() kjører om og om igjen i det undelige

void loop()
{

bestilling = digitalRead (knappBestilling); //Les av knappens innstilling

if (bestilling == false)
 {
   gront_lys();        // Utfør endring til grønt lys og tilbake
   bestilling = true; // Sett bestilling til null/falsk
 }
 else
 {
   digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode
 }
}

void gront_lys()
{
  digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode

  delay(RODTID);

  digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode

  delay(GULTID);

  digitalWrite(ledPinRed, LOW); // Slå av rød lysdiode
  digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
  digitalWrite(ledPinGreen, HIGH); // Slå på grønn lysdiode

  delay(GRONNTID);

  for (int i = 0; i < ANTALLBLINK; i++)
   {
```

```
      digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
      Lyd(KORTLYDPULS); // Gi lydstøt på 100ms
      delay(LENGDEBLINK/2);
      digitalWrite(ledPinGreen, HIGH);
      Lyd(KORTLYDPULS); // Gi lydstøt på 100ms
      delay(LENGDEBLINK/2);
  }

   digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
   digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode

   delay(GULTID);

   digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
}

void Lyd(int varighet)
 {
  tone(tonePin, 1500, varighet);
// Angir pinnenummer for piezoelektrisk lydgiver, frekvens og varighet i
msek
}
```

B.1.7 Deloppdrag 6 – Blinkende gult lys

```
/*
Trafikklys Deloppdrag 6
Lag et trafikklys som lyser slik som et trafikklys skal lyse
når det skifter fra rødt til grønt og tilbake.
Skifte skal imidlertid kun inntreffe ved trykk på knappen. Fra nå av skal
det grønne lyset lyse konstant  i 8 sek. for deretter å etterfølges av 5 blink
med et halvt sekund på og et halvt sekund av. De gule lysene skal være på i 1 sek,
og det skal gå 5 sek. etter at det er trykket på knappen før det skifter til grønt
Det skal gis et gort lydpuls hver gang det blinkende grønne lyset skifter
Når mørket faller på skal det gule lyset begynne å blinke*/

int ledPinRed = 13;    // Kontakt rød LED til pinne pin 13
int ledPinYellow = 12; // Kontakt rød LED til pinne pin 12
int ledPinGreen = 11;  // Kontakt rød LED til pinne pin 11
```

```
int tonePin = 6;       // Pinnen der den passive buzzeren er tilkoblet
int knappBestilling = 7; // Kontakt knapp for bestilling av grønn mann
int blinkendeGult = 0; // Kontakt for tilkobling av LDR
int lysStyrke; // Angir hvor lyst det er i ved trafikklyset
boolean bestilling = false; // Boolsk variabel som holde bestilling

// Definer konstanter

const int RODTID = 5000;       // 5 sekunder før det blir grønt
const int GULTID = 1000;       // 1 sekund gultid
const int ANTALLBLINK = 5;     // 5 blink, ett blink er på og av.
const int LENGDEBLINK = 1000;  // 1 sek, inkluderer halvparten av og alvparten på
const int GRONNTID = 8000;    // Lengden det skal være fast grønn før det begynner
å blinke.
const int KORTLYDPULS = 100;   // Kort lydpuls er satt til 100 msek.

// Setup definerer hva som er utganger og hva som er innganger

void setup() {
pinMode(ledPinRed, OUTPUT);
pinMode(ledPinYellow, OUTPUT);
pinMode(ledPinGreen, OUTPUT);
pinMode(tonePin, OUTPUT);
pinMode(knappBestilling, INPUT);
}

// loop() kjører om og om igjen i det uendelige

void loop()
{
 lysStyrke = analogRead(blinkendeGult); //Leser lysintensiteten på LDR
 bestilling = digitalRead (knappBestilling); //Les av knappens innstilling

if (bestilling == false)
 {
   gront_lys();        // Utfør endring til grønt lys og tilbake
   bestilling = true; // Sett bestilling til null/falsk
 }
 else
 {
 if ((lysStyrke < 400) && (bestilling == true))
  {
```

```
  gult_blinkende();              // Start blinkende gult
  }
 else
  {
  digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode
  }
 }
}

void gront_lys()
{
  digitalWrite(ledPinRed, HIGH); // Slå på rød lysdiode

  delay(RODTID);

  digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode

  delay(GULTID);

  digitalWrite(ledPinRed, LOW); // Slå av rød lysdiode
  digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
  digitalWrite(ledPinGreen, HIGH); // Slå på grønn lysdiode

  delay(GRONNTID);

  for (int i = 0; i < ANTALLBLINK; i++)
   {
     digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
     Lyd(KORTLYDPULS); // Gi lydstøt på 100ms
     delay(LENGDEBLINK/2);
     digitalWrite(ledPinGreen, HIGH);
     Lyd(KORTLYDPULS); // Gi lydstøt på 100ms
     delay(LENGDEBLINK/2);
  }

   digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
   digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode

   delay(GULTID);

   digitalWrite(ledPinYellow, LOW); // Slå av gul lysdiode
```

```
}

void Lyd(int varighet)
{
  tone(tonePin, 1500, varighet); // Angir pinnenummer for piezoelektrisk lydgiver,
                                 // frekvens og varighet i msek
}

void gult_blinkende()
{
  digitalWrite(ledPinRed, LOW); // Slå av rød lysdiode
  digitalWrite(ledPinYellow, HIGH); // Slår på gult
  delay(LENGDEBLINK/2);
  digitalWrite(ledPinYellow, LOW); // Slår av gult
  delay(LENGDEBLINK/2);
}
```

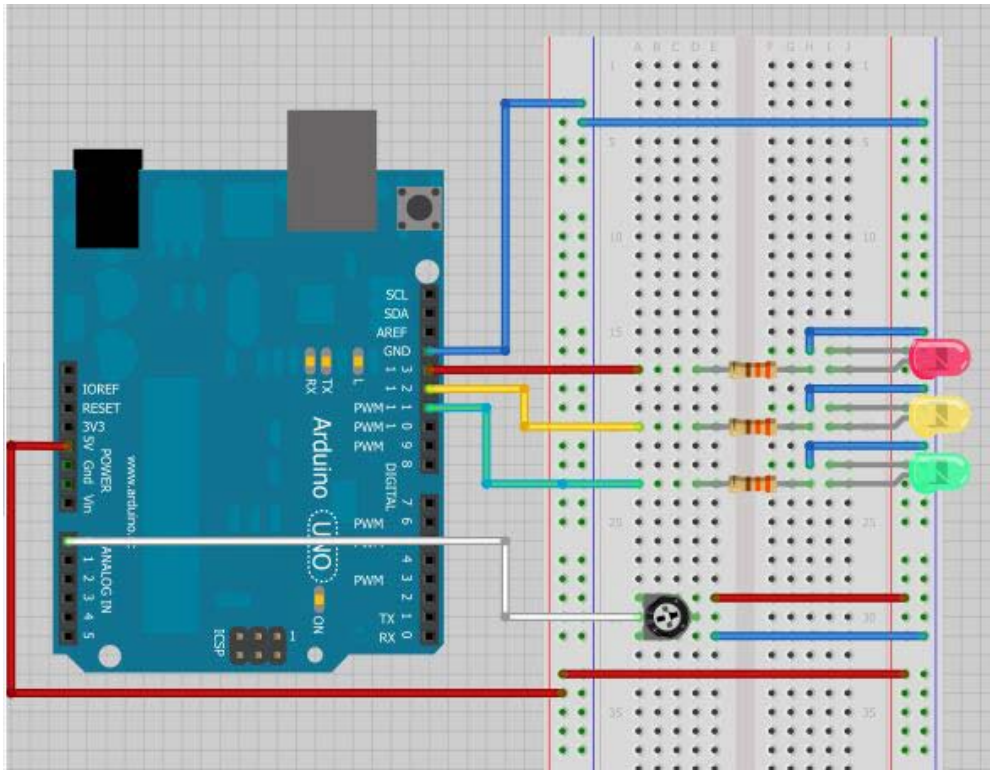
Vedlegg C Løsningsforslag Halvåpne oppgaver

C.1 Batteritester

Oppgaveteksten finnes på side 107.

C.1.1 Oppkobling

Oppkoblingen inkluderer et potensiometer som skal "simulere" et batteri hvor spenningen blir dårligere etter som batteriet svekkes. I stedet for at spenningen faller, så justeres spenningen ut fra potensiometeret ned. Ved tilfredsstillende "batterispenning lyser den grønne dioden. Ved lav, men tilstrekkelig "batterispenning" lyser den gule dioden, og ved for lav "batterispenning" lyser den røde dioden. Under målingen bør batteriet belastes med en passende belastningsmotstand.



Figur C.1 Oppkobling Batteritester.

C.1.2 Løsningsforslag deloppdrag 1 – Les av batterispenningen

```
/*
  Batteritester deloppdrag 1
```

```
 Lag en batteritester som måler spenningen på batteret,
 og skriver resultatet til monitoren
*/

int analogPin =  0;                          // Kontakt for måling av spenning
int verdiDigital = 0;                         // Avlest digital verdi
int kalibrertDigital = 655;                  // Avlest maks digital spenning
float kalibrertSpenning = 3.22;              // Maks verdi er 3,22 V
float verdiSpenning;                         // Målt kalibrert spenningsverdi

// Setup definerer hva som er utganger og hva som er innganger

void setup()
{
 Serial.begin(9600);                         // Sett opp kommunikasjon til monitor
 pinMode(analogPin,INPUT);                   //
}

// loop() kjører om og om igjen i det undelige,

void loop()
{
 verdiDigital = analogRead(analogPin);       // Les av spenningen
 delay(500);
 verdiSpenning = (1.0*verdiDigital/kalibrertDigital)*kalibrertSpenning;
 Serial.print("Digital batterispenning:" );
 Serial.print(verdiDigital);
 Serial.print(", Maalt batterispenning:" );
 Serial.println(verdiSpenning);

}
```

C.1.3 Løsningsforslag deloppdrag 2 – Godt eller dårlig batteri

```
/*
 Batteritester deloppdrag 2
 Lag en batteritester som måler spenningen på batteriet,
 og skriver en tekst om batteriet kan brukes eller ikke
*/
```

```
int analogPin =  0;                          // Kontakt for måling av spenning
int verdiDigital = 0;                        // Avlest digital verdi
int kalibrertDigital = 655;                  // Avlest maks digital spenning
float kalibrertSpenning = 3.22;              // Maks verdi er 3,22 V
float verdiSpenning;                         // Målt kalibrert spnningsverdi
float terskelGodtBatteri = 3.0;      /       / Nedre terskel for godt batteri
float terskelUbrukeligBatteri = 2.7;         // Øvre terskel for ubrukelig batteri

// Setup definerer hva som er utganger og hva som er innganger

void setup()
{
  Serial.begin(9600);                        // Sett opp kommunikasjon til monitor
  pinMode(analogPin,INPUT);                  //
}

// loop() kjører om og om igjen i det undelige,

void loop()
{
 verdiDigital = analogRead(analogPin);       // Les av spenningen
 delay(500);
 verdiSpenning = (1.0*verdiDigital/kalibrertDigital)*kalibrertSpenning;

 if (verdiSpenning > terskelGodtBatteri)
 {
   Serial.print("Batteriet er godt, spenning: ");
   Serial.print(verdiSpenning);
   Serial.println(" Volt");
 }
 else if ((verdiSpenning < terskelGodtBatteri) && (verdiSpenning > terskelUbrukeligBatteri))
 {
   Serial.print("Batteriet er daarlig men kan brukes, spenning: ");
   Serial.print(verdiSpenning);
   Serial.println(" Volt");
 }
 else
```

```
 {
   Serial.print("Batteriet er ubrukelig og boer kastes, spenning: ");
   Serial.print(verdiSpenning);
   Serial.println(" Volt");
 }
}
```

C.1.4 Løsningsforslag deloppdrag 3 – LED som viser godt eller dårlig batteri

```
/*
  Batteritester deloppdrag 3
  Lag en batteritester som måler spenningen på batteriet,
  og skriver en tekst om batteriet kan brukes eller ikke.
  Dessuten skal en grønn, gul eller rød lysdiode lyse for å vise kvaliteten til batteriet
*/

int analogPin =  0;                     // Kontakt for måling av spenning
int verdiDigital = 0;                   // Avlest digital verdi
int kalibrertDigital = 655;             // Avlest maks digital spenning
boolean God = false;
boolean Daarlig = false;
boolean Ubrukelig = false;
int pinGod = 11;
int pinDaarlig = 12;
int pinUbrukelig = 13;
float kalibrertSpenning = 3.22;         // Maks verdi er 3,22 V
float verdiSpenning;                    // Målt kalibrert spnningsverdi
float terskelGodtBatteri = 3.0;         // Nedre terskel for godt batteri
float terskelUbrukeligBatteri = 2.7;    // Øvre terskel for ubrukelig batteri

// Setup definerer hva som er utganger og hva som er innganger

void setup()
{
  Serial.begin(9600);                   // Sett opp kommunikasjon til monitor
  pinMode(analogPin,INPUT);             //
  pinMode(pinGod, OUTPUT);
  pinMode(pinDaarlig, OUTPUT);
  pinMode(pinUbrukelig, OUTPUT);
```

```
}

// loop() kjører om og om igjen i det undelige,

void loop()
{
 verdiDigital = analogRead(analogPin);              // Les av spenningen
 delay(500);
 verdiSpenning = (1.0*verdiDigital/kalibrertDigital)*kalibrertSpenning;

 if (verdiSpenning > terskelGodtBatteri)
 {
   Serial.print("Batteriet er godt, spenning: ");
   Serial.print(verdiSpenning);
   Serial.println(" Volt");
   God = true;
   Daarlig = false;
   Ubrukelig = false;

 }
 else if ((verdiSpenning < terskelGodtBatteri) && (verdiSpenning > terskelUbrukeligBatteri))
 {
   Serial.print("Batteriet er daarlig men kan brukes, spenning: ");
   Serial.print(verdiSpenning);
   Serial.println(" Volt");
   God = false;
   Daarlig = true;
   Ubrukelig = false;
 }
 else
 {
   Serial.print("Batteriet er ubrukelig og boer kastes, spenning: ");
   Serial.print(verdiSpenning);
   Serial.println(" Volt");
   God = false;
   Daarlig = false;
   Ubrukelig = true;
 }
```

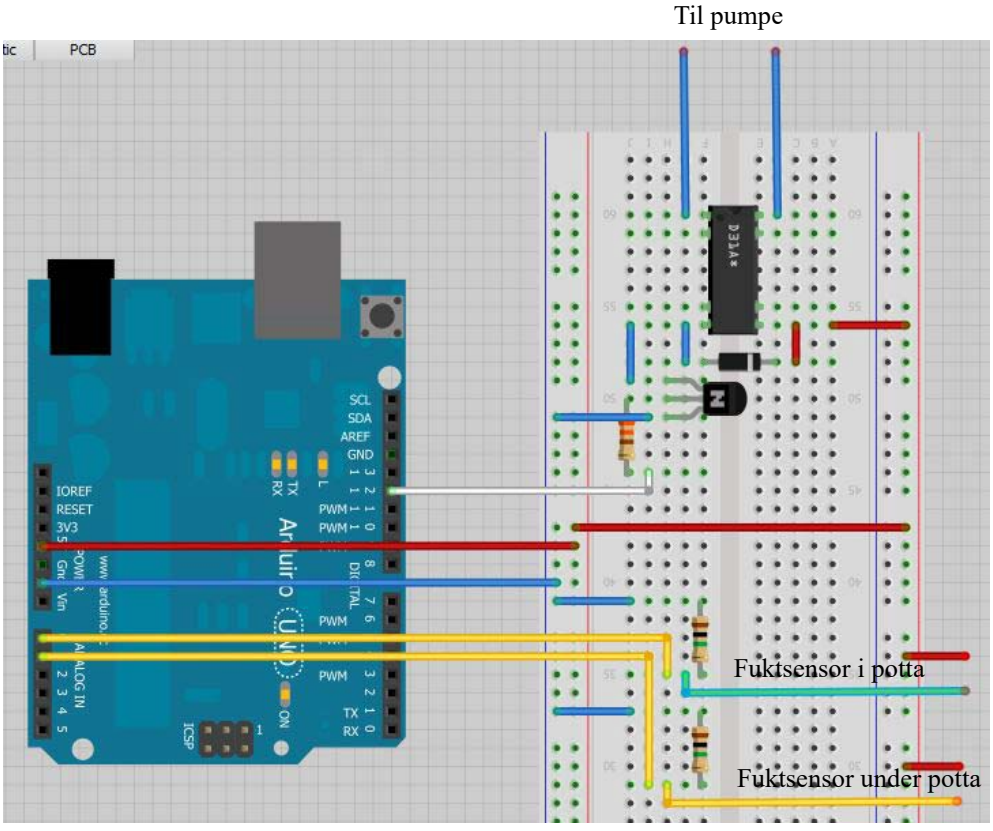
```
  digitalWrite(pinGod, God);
  digitalWrite(pinDaarlig, Daarlig);
  digitalWrite(pinUbrukelig, Ubrukelig);
}
```

C.2 Automatisk blomstervanner

Oppgaveteksten finnes på side 108.

C.2.1 Oppkobling

Kretsen har to sensorer en som måler fuktigheten i jorda i potta (*fuktsensor i potte*), og en som måler fuktighet på skåla eller på gulvet rundt potta (*fuktsensor under potta*). Den siste skal være en sikkerhet som gjør at vanningen avbrytes umiddelbart. I tillegg benyttes et rele som kobler inn pumpa fra ekstern strømkilde. Det gjøres oppmerksom på at det releet som er avbildet på figuren under ehar en annen utforming enn releet som følger med Sparkfun Inventors kit.



Figur C.2 Oppkobling automatisk blomstervanner.

C.2.2 Løsningsforslag deloppdrag 1

```
/*
  Blomstarvanner deloppdrag 1
  Det skal lages en krets som automatisk vanner blomster
  når blomsten er blitt tørr. Mengden vann skal kunne tilpasses
  plantens behov.
*/

int vanningPin = 12;
int torrPin = 0;
int torr = 0;                                   // Avlest grad av fuktighet
int terskel_torr = 300;                         // Terskelverid for når planten er tørr
int mengde_vann = 2000;                         // Angir mengde vann
int min_tid_hver_vanning = 5000;                // Minimumstid mellom hver vanning

// Setup definerer hva som er utganger og hva som er innganger

void setup()
{
  pinMode(vanningPin,OUTPUT);
  pinMode(torrPin,INPUT);
  Serial.begin(9600);
}

// loop() kjører om og om igjen i det undelige,

void loop()
{
  torr = analogRead(torrPin);
  Serial.print("Torrverdi: ");
  Serial.println(torr);
  delay(1000);
  if (torr < terskel_torr)
  {
    digitalWrite(vanningPin,HIGH);
    delay(mengde_vann);
    digitalWrite(vanningPin,LOW);
    delay(min_tid_hver_vanning);
```

```
  }
}
```

C.2.3 Løsningsforslag deloppdrag 2

```
/*
  Blomstervanner deloppdrag 2
  Det skal lages en krets som automatisk vanner blomster
  når blomsten er blitt tørr. Mengden vann skal kunne tilpasses
  plantens behov. I tillegg skal det legges inn sikkerhet mot lekkasje.
*/

int vanningPin = 12;
int torrPin = 0;
int lekkasjePin = 1;
int torr = 0;                                   // Avlest grad av fuktighet
int lekkasje = 0;
int terskel_torr = 300;                         // Terskelverdi for når planten er tørr
int terskel_lekkasje = 300;                     // Terskelverdi for deteksjon av lekkasje
int mengde_vann = 2000;                         // Angir mengde vann
int min_tid_hver_vanning = 5000;                // Minimumstid mellom hver vanning

// Setup definerer hva som er utganger og hva som er innganger

void setup()
{
  pinMode(vanningPin,OUTPUT);
  pinMode(torrPin,INPUT);
  Serial.begin(9600);
}

// loop() kjører om og om igjen i det uendelige,

void loop()
{
  torr = analogRead(torrPin);
  lekkasje = analogRead(lekkasjePin);
  Serial.print("Torrverdi: ");
  Serial.println(torr);
```

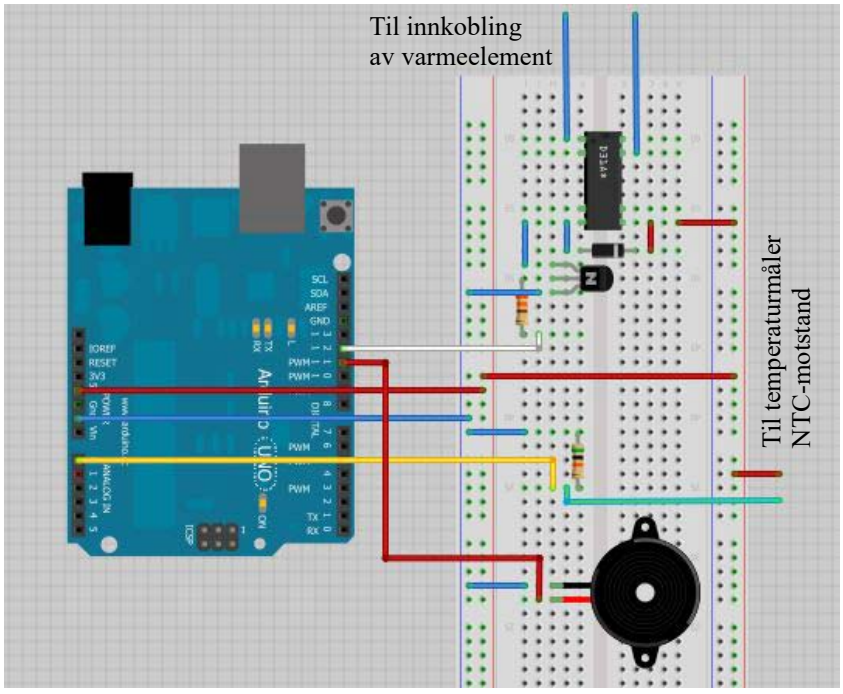
```
  delay(1000);

  if ((torr < terskel_torr) && (lekkasje < terskel_lekkasje))
  {
    digitalWrite(vanningPin,HIGH);
    delay(mengde_vann);
    digitalWrite(vanningPin,LOW);
    delay(min_tid_hver_vanning);
  }
}
```

C.3 Automatisk vannvarmer

Oppgaveteksten finnes på side 110.

C.3.1 Oppkobling

Kretsen styrer et rele som kan slå av og på et lite varmeelement laget med en glødetråd. Ved hjelp av en NTC-motstand måles temperaturen. NTC-motstanden er karakterisert på forhånd, dvs. man kjenner sammenhengen mellom temperatur og motstand. Kretsen regulerer temperaturen i glasset rundt den ønskede temperaturen.Dernest legges det inn en lyd når temperaturen passerer 30C. En høy tone på vei opp og dyp tone på vei ned.



C.3.2 Løsningsforslag deloppdrag 1 – 3

```
/*
  Vannvarmer deloppdrag 1 - 3
  Det skal lages en vannvarmer som varmer opp vann.
  Ved å måle strøm og spenning over varmeelementet skal en beregne
  hvor lang tid det vil ta å varme opp 20 ml vann fra 20C til 25C
  Strøm x Spenning = Effekt (W), 1 Watt x 1 sek = 1 J
  Det skal 4,18 J til for å varme opp 1ml til 1 grad K
  Dernest skal det legges inn lyd når temperaturen passerer 30C.
  En høy tone på vei opp og dyp tone på vei ned.
*/

int oppvarmingPin = 12;
int lydPin = 11;
boolean lydFlag = true;
int temperaturPin = 0;
int kal_digiLav = 459;
int kal_digiHoy = 637;
float kal_anaLav = 21.3;
float kal_anaHoy = 36.6;
float a,b;
int tempDigital;
float tempAnalog;
float terskel_temp_C = 30;          //


// Setup definerer hva som er utganger og hva som er innganger

void setup()
{
  pinMode(oppvarmingPin,OUTPUT);
  pinMode(temperaturPin,INPUT);
  pinMode(lydPin,OUTPUT);
  Serial.begin(9600);
}

// loop() kjører om og om igjen i det uendelige,
```

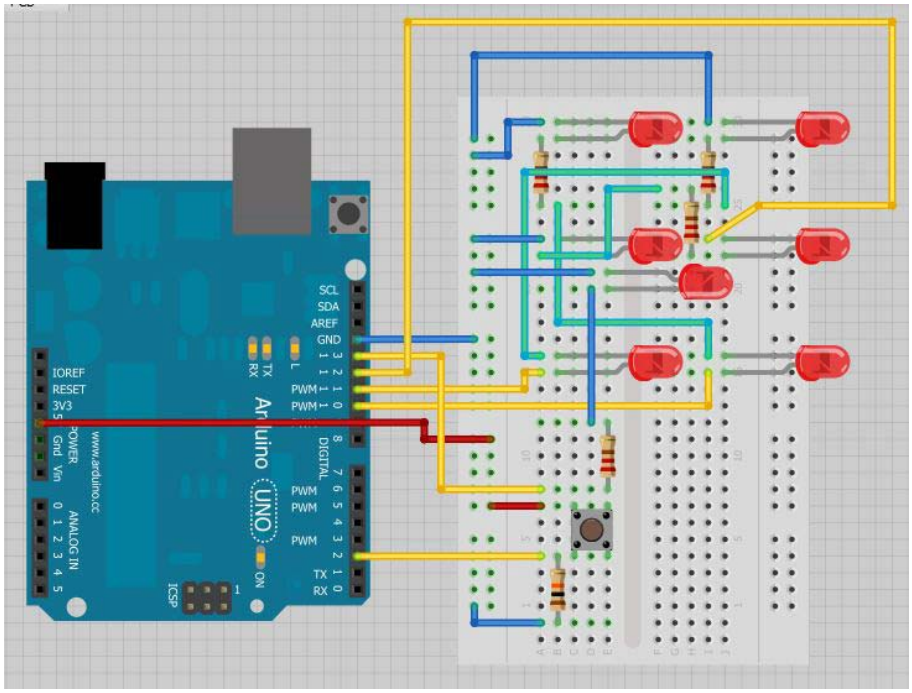
```
void loop()
{
  tempDigital = analogRead(temperaturPin);
  a = (kal_anaHoy - kal_anaLav)/(kal_digiHoy - kal_digiLav);
  b = kal_anaHoy - a*kal_digiHoy;
  tempAnalog = a*tempDigital + b;
  Serial.print("Temperatur digital: ");
  Serial.println(tempAnalog);
  delay(1000);

  if (tempAnalog < terskel_temp_C)
  {
    if (lydFlag == true)
      {
        tone(lydPin,2000,200);
        lydFlag = false;
      }
    digitalWrite(oppvarmingPin,HIGH);
    delay(1000);
  }
  else
  {
    if (lydFlag == false)
      {
        tone(lydPin,300,200);
        lydFlag = true;
      }
    digitalWrite(oppvarmingPin,LOW);
    delay(1000);
  }
}
```

C.4 Elektronisk terning

Oppgaveteksten finnes på side 112.

C.4.1 Oppkobling

Ved å trykke på knappen viser terningen et tilfeldig tall mellom 1 og 6.



C.4.2 Løsningsforslag deloppdrag 1

```
// Elektronisk terning
// 1     2
// 3  4  5
// 6     7

int kast;
int tilfeldig;

const int pinne_1_7 = 10;
const int pinne_2_6 = 11;
const int pinne_3_5 = 12;
const int pinne_4 = 13;
const int pinne_kast = 2;                    // IO-port for interrupt
```

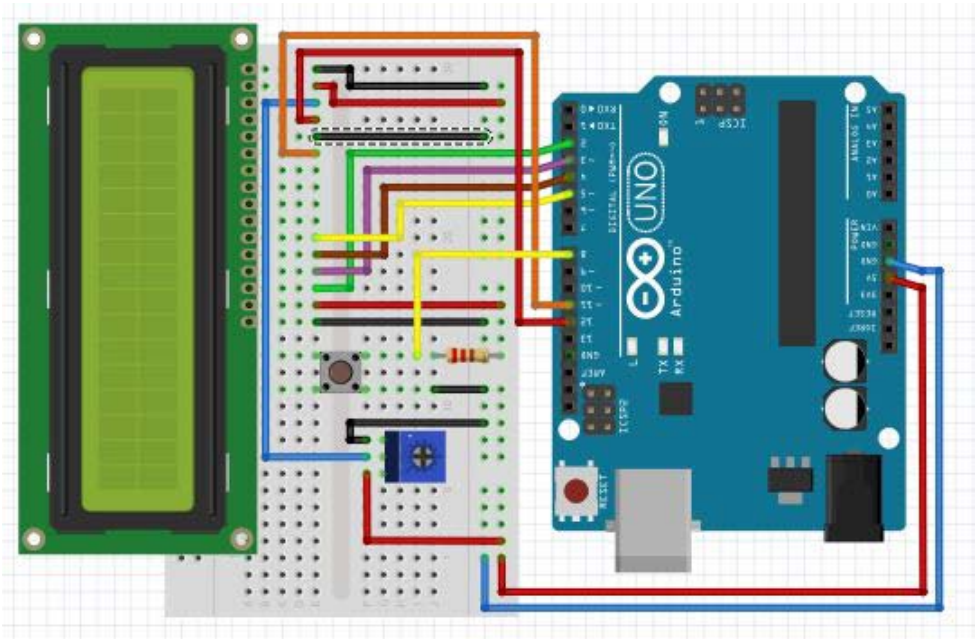
```
void setup ()
{
  attachInterrupt(0,visResultat,RISING);          // Initier interrupt på IO-port 2

  pinMode(pinne_1_7,OUTPUT);
  pinMode(pinne_2_6,OUTPUT);
  pinMode(pinne_3_5,OUTPUT);
  pinMode(pinne_4,OUTPUT);
  pinMode(pinne_kast,INPUT);
}

void loop()
{
  tilfeldig = random(1,7);
  kast = analogRead(pinne_kast);
}

void visResultat()
 {
    // Nullstill alle lysdioder
    digitalWrite(pinne_1_7,LOW);
    digitalWrite(pinne_2_6,LOW);
    digitalWrite(pinne_3_5,LOW);
    digitalWrite(pinne_4,LOW);

    if (tilfeldig == 1)
    {
     digitalWrite(pinne_4,HIGH);
    }

    if (tilfeldig == 2)
    {
     digitalWrite(pinne_1_7,HIGH);
    }
    if (tilfeldig == 3)
    {
     digitalWrite(pinne_1_7,HIGH);
     digitalWrite(pinne_4,HIGH);
```

```
    }
    if (tilfeldig == 4)
    {
     digitalWrite(pinne_1_7,HIGH);
     digitalWrite(pinne_2_6,HIGH);
    }
    if (tilfeldig == 5)
    {
     digitalWrite(pinne_1_7,HIGH);
     digitalWrite(pinne_2_6,HIGH);
     digitalWrite(pinne_4,HIGH);
    }
    if (tilfeldig == 6)
    {
     digitalWrite(pinne_1_7,HIGH);
     digitalWrite(pinne_2_6,HIGH);
     digitalWrite(pinne_3_5,HIGH);
    }
   delay(100);
}
```

C.5 Stoppeklokke

Oppgaveteksten finnes på side 115.

C.5.1 Forslag til oppkobling



C.5.2 Løsningsforslag deloppdrag 4

```
/*
Stoppeklokke Nils Kr. Rossing 12.12.14
*/

#include <LiquidCrystal.h>

LiquidCrystal lcd(12,11,5,4,3,2);

long int subSek = 0;
long int startSubSek = 0;
long int sekunder = 0;
long int startSek = 0;
long int minutter = 0;
long int startMin = 0;
    int startFlag = 0;
    int stopFlag = 1;
```

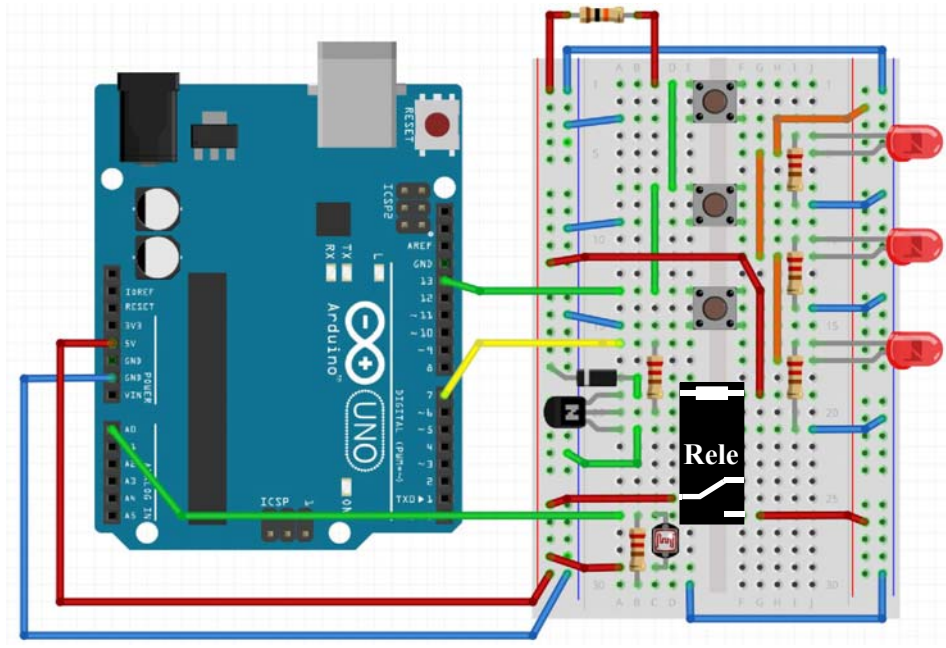
```
    int knapp = 1;

void setup()
{
  lcd.begin(16, 2);
  lcd.clear();
  lcd.print("Klar til start? Trykk knapp");
}

void loop()
{
knapp = digitalRead (8);          // Les av trykknappen
if (startFlag == 0)                 // Går klokka eller står den?
{
  while (knapp == 0)                // Stopp programmet ved ved trykk
  {
    knapp = digitalRead (8);
   startSubSek = millis()/10;     // Les av millisek. ved oppstart etter
trykk
    startFlag = 1;
    subSek = 0;
    sekunder = 0;
    minutter = 0;
    lcd.clear();
  }
}
else if (startFlag == 1)          // Aksjon når programmet går og knappen
trykkes
{
  while (knapp == 0)                // Aksjon når knappen trykkes mens
programmet gåt
  {
    knapp = digitalRead (8);
    startFlag = 0;
  }
}
```

```
if (startFlag == 1)                // Måler tiden når knappen har vært
trykket
{
  subSek = millis()/10 - startSubSek;  // Definer starttiden i forhold
til millis()

  if (subSek >= 100)               // Nullstill hundredeler
  {
    startSubSek = startSubSek + 100;
    subSek = 0;
    sekunder++;
    lcd.clear();
  }

  if (sekunder >= 60)              // Nullstill sekunder
  {
    sekunder = 0;
    minutter++;
    lcd.clear();
  }

  if (minutter >= 60)              // Nullstill minutter
  {
  minutter = 0;
  lcd.clear();
  }

  lcd.setCursor(0,1);              // Skriv minutter til displayet
  lcd.print(minutter);

  lcd.setCursor(3,1);              // Skriv sekunder til displayet
  lcd.print(sekunder);

  lcd.setCursor(6,1);              // Skriv hundredeler til displayet
  lcd.print(subSek);
}
}
```

C.6 Trappelys

Oppgaveteksten finnes på side 116.

C.6.1 Forslag til oppkobling



C.6.2 Løsningsforslag deloppdrag 3

```
// Løsningsforslag Trappelys_3
// Ved å trykke en vilkårlig bryter skal lyset tennes i hele oppgangen.
// Bryterne skal kunne "toggle" mellom på og av, fra trykk til trykk
// Nils Kr. Rossing 07.10.21

int pinRele = 7;       // Tilkobling av rele
int pinBrytere = 13;   // Tilkobling av brytere

int lysStatus = 0;     // Status på lys, på (1), av (0)
int brytere;           // Respons fra brytere - Inngang

void setup()
{
```

```
pinMode(pinBrytere, INPUT_PULLUP);
pinMode(pinRele, OUTPUT);
}

void loop()
{
  brytere = digitalRead(pinBrytere);
  delay(100);

  if(brytere == LOW)
  {
    if (lysStatus == HIGH)
    {
      while(!digitalRead(pinBrytere));
      digitalWrite(pinRele, LOW);
      delay(100);
      lysStatus = LOW;
    }
    else if (lysStatus == LOW)
    {
      while(!digitalRead(pinBrytere));
      digitalWrite(pinRele, HIGH);
      delay(100);
      lysStatus = HIGH;
    }
  }
}
```

C.6.3 Løsningsforslag deloppdrag 4

```
// Løsningsforslag Trappelys_4
// Ved å trykke en vilkårlig bryter skal lyset tennes i hele oppgangen.
// Bryterne skal kunne "toggle" mellom på og av, fra trykk til trykk
// Når en lyssensor registrerer at det er dagslys, slukkes lyset
automatisk
// og tennes kun i 5 sek dersom en bryter trykkes
// Nils Kr. Rossing 07.10.21
```

```
int pinRele = 7;       // Utgangen for styring av rele
int pinBrytere = 13;   // Tilkobling av brytere

int lysStatus = 0;     // Status på lys, på (1), av (0)
int brytere;           // Respons fra brytere - Inngang
int lysStyrke;         // Målt lysstyrke
int lysTerskel=500;    // Terskelverdi for omslag

void setup()
{
pinMode(pinBrytere, INPUT_PULLUP);
pinMode(pinRele, OUTPUT);

Serial.begin(9600);
}

void loop()
{
  brytere = digitalRead(pinBrytere);
  delay(100);

  // Registrering av avlesninger for testing
  /*
  Serial.print("Lys: ");
  Serial.print(lysStyrke);
  Serial.print(", Bryter: ");
  Serial.println(brytere);
  */

  if(brytere == LOW)
  {
    if (lysStatus == HIGH)
    {
      while(!digitalRead(pinBrytere));
```

```
      digitalWrite(pinRele, LOW);
      delay(100);
      lysStatus = LOW;
    }
    else if (lysStatus == LOW)
    {
      while(!digitalRead(pinBrytere));
      digitalWrite(pinRele, HIGH);
      delay(100);
      lysStatus = HIGH;
    }
  }

 lysStyrke = analogRead(0);

 if((lysStyrke > lysTerskel) && (lysStatus == HIGH))
 {
  delay(5000);
  digitalWrite(pinRele, LOW);  // Er det lyst og lyset står på slå av
lyset etter 5 sek.
  lysStatus = LOW;
  brytere == LOW;
 }
 delay(100);
}
```

C.6.4 Løsningsforslag deloppdrag 5

```
// Løsningsforslag Trappelys_5
// Ved å trykke en vilkårlig bryter skal lyset tennes i hele oppgangen.
// Bryterne skal kunne "toggle" mellom på og av, fra trykk til trykk
// Når en lyssensor registrerer at det er dagslys, slukkes lyset
automatisk
// og tennes kun i 5 sek dersom en bryter trykkes
// Etter at det er blitt mørkt vil også lyset tennes ved bevegelse ved
inngangen
// Nils Kr. Rossing 07.10.21
```

```
int pinRele = 7;      // Utgangen for styring av rele
int pinPIR = 6;       // Ingang for avlesning av PIR-sensor
int pinBrytere = 13;  // Tilkobling av brytere

int lysStatus = 0;    // Status på lys, på (1), av (0)
int brytere;          // Respons fra brytere - Inngang
int lysStyrke;        // Målt lysstyrke
int lysTerskel=500;   // Terskelverdi for omslag
int pirSensor;        // PIR-sensor innslag

void setup()
{
pinMode(pinBrytere, INPUT_PULLUP);
pinMode(pinPIR, INPUT);
pinMode(pinRele, OUTPUT);

Serial.begin(9600);
}

void loop()
{
  brytere = digitalRead(pinBrytere);
  lysStyrke = analogRead(0);
  pirSensor = digitalRead(pinPIR);
  delay(100);

  // Registrering av avlesninger for testing
  /*
  Serial.print("Lys: ");
  Serial.print(lysStyrke);
  Serial.print(", PIR: ");
  Serial.print(pirSensor);
  Serial.print(", Bryter: ");
  Serial.println(brytere);
  */
```

```
  if(brytere == LOW)
  {
    if (lysStatus == HIGH)
    {
      while(!digitalRead(pinBrytere));
      digitalWrite(pinRele, LOW);
      delay(100);
      lysStatus = LOW;
    }
    else if (lysStatus == LOW)
    {
      while(!digitalRead(pinBrytere));
      digitalWrite(pinRele, HIGH);
      delay(100);
      lysStatus = HIGH;
    }
  }

 if((lysStyrke > lysTerskel) && (lysStatus == HIGH))
 {
  delay(5000);
  digitalWrite(pinRele, LOW);  // Er det lyst og lyset står på slå av
lyset etter 5 sek.
  lysStatus = LOW;
  brytere == LOW;
 }

 if((pirSensor == HIGH) && (lysStyrke > lysTerskel) && (lysStatus == LOW))
 {
  digitalWrite(pinRele, HIGH);  // Er det lyst og lyset står på slå av
lyset etter 5 sek.
  while(digitalRead(pinPIR));  // Lyset holdes på så lenge sensoren leve-
rer et høyt signal
  digitalWrite(pinRele, LOW);   // Så snart sensoren går lav slukkes lyset
 }
 delay(100);
```

```
}
```

Vedlegg D “Plantede” feil i løsningsforslagene

Følgende plantede feil og mangler er lagt inn i løsningsforslagene i trafikklysoppgaven:

Oppdrag 0

- Mangler
  ```
  delay(1000);
  ```
  på slutten av void loop()
  Se side 72.

Oppdrag 1

- Det røde lyset slås av ide trafikklyset skifter fra rødt til gult
  ```
  digitalWrite(ledPinRed, LOW);     // Slå av rød lysdiode
  digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode
  delay(1000);                  // Gul lysdiode skal være på i ett sek.
  ```
  Det skal være slik:
  ```
  digitalWrite(ledPinYellow, HIGH); // Slå på gul lysdiode
  delay(1000);                  // Gul lysdiode skal være på i ett sek.
  ```
  Se side 75.

Oppdrag 2

- Argumentet inneholder en tilordning og ikke en betingelse:
  ```
  if (bestilling = 0) {}
  ```
  Skal være:
  ```
  if (bestilling == 0) {}
  ```
  Se side 79.

Oppdrag 3

- Mangler funksjonskall i void loop()
  Skal stå:
  ```
  if (bestilling == 0)
  {
    gront_lys();        // Kall funksjonen gront_lys()
    bestilling = 1;     // Sett bestilling til null/falsk
  }
  ```
  Se side 83.

Oppdrag 4

- I betingelsen er det skrevet “=” i stedet for “<“.
  Står:
  ```
  for (int i = 0; i = ANTALLBLINK; i++)
  {
  ```

```
        // Her står det som skal gjenta
}
Skal stå:
for (int i = 0; i < ANTALLBLINK; i++)
{
        // Her står det som skal gjenta
}
```

Oppdrag 5

- Skal programmet fungere som ønsket så må funksjonen Lyd() kalles i for-loopen()

```
for (int i = 0; i < ANTALLBLINK; i++)
{
  digitalWrite(ledPinGreen, LOW); // Slå av grønn lysdiode
  Lyd(KORTLYDPULS);               // Gi lydstøt på 100ms
  delay(LENGDEBLINK/2);           // La lyset være av i halve blinktid
  digitalWrite(ledPinGreen, HIGH);// Slå på grønn lysdiode
  Lyd(KORTLYDPULS);               // Gi lydstøt på 100ms
  delay(LENGDEBLINK/2);           // La lyset være på i halve blinktid
}
```

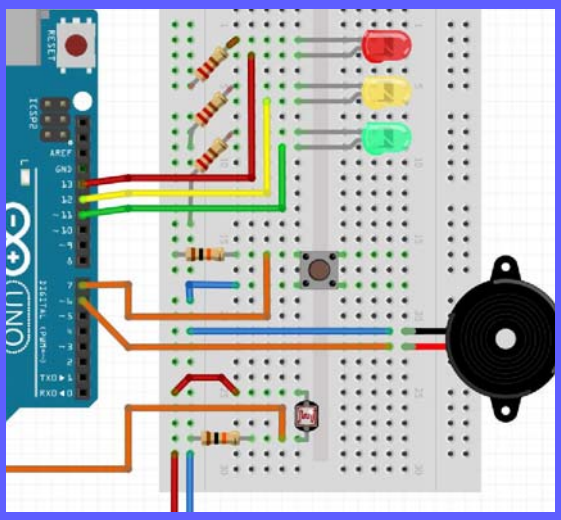
Oppdrag 6

- Terskelen for å slå på blinkende gult er satt alt for lavt (0). Verdien kan aldri bli under 0.
  Det står:

```
if ((lysStyrke < 0) && (bestilling == 1))
{
  gult_blinkende();              // Start blinkende gult
}
```

  En bedre løsning er sannsynligvis:

```
if ((lysStyrke < 400) && (bestilling == 1))
{
  gult_blinkende();              // Start blinkende gult
}
```



Heftet er ment som en enkel introduksjon til Arduino-programmering. Fordelen er at de fleste oppgavene er lite krevende når det gjelder utstyr. Et vanlig *Arduino starter* kit vil derfor egne seg godt. Oppgavene egner seg derfor også godt for simulering. Denne utgaven av heftet inkluderer derfor en beskrivelse av kretssimulatoren i TinkerCAD slik flere av oppgavene skal kunne gjennomføres som ren digital undervisning.

Undervisningsopplegget inkluderer bruk av variabler, egen kodete funksjoner, for-sløyfer og if-kommandoer og bruk av både sensorer (bryter og lysfølsom motstand) og aktuatorer (lysdioder og lydgiver). Et trafikklys er dessuten noe alle er kjent med samtidig som få kjenner den tekniske virkemåten.

Fra og med utgave 3.0 er det relativt tidlig innført bruk av egendefinerte funksjoner som er brukt gjennom resten av opplegget. Flytskjema er også omtalt i innledningen og brukt for å vise programstrukturen tydeligere i oppgavedelen. Et gratis program for tegning av flytskjema er beskrevet i kapittel 4, side 54.

Fra og med utgave 4,0 er det lagt inn løsningsforslag etter hvert oppdrag for trafikklysoppgaven. Løsningsforslaget har en mangel som må rettes opp for å få det til å fungere etter spesifikasjonene. Mangelen, eller feilen er knyttet til et sentralt læringsmål ved det aktuelle oppdraget.

Fra og med utgave 5.0 er oppgavene delt inn i tre deler:

- *Introduksjonsoppgaver*
  Dette er oppgaver som egner seg som første innføring i programmering. Foreløpig er det bare to slike oppgaver.
- *Guidete oppgaver*
  Disse er oppgaver hvor kandidaten ledes trinn for trinn fram til en komplett løsning. Det gis også tips under veis. I to av oppgavene henvises til egne hefter. Det gjelder *Lag en høydemåler* og *Lag et kolorimeter.*
- *Halvåpne oppgaver*
  I de halvåpne oppgavene overlates en større del av oppdraget til kandidaten selv. Flere av disse oppgavene krever også noe ekstrautstyr ut over Arduino starter kit eller Sparkfun invention kit.

Nils Kr. Rossing
Dosent ved Skolelaboratoriet
E-post: nils.rossing@ntnu.no
Prosjektleder ved Vitensenteret
E-post:nkr@vitensenteret.com

Rev 5.3 – 31.01.24 ISBN

Trykk: NTNU Grafisk senter

NTNU
Trondheim

Institutt for fysikk

Skolelaboratoriet
for matematikk, naturfag og teknologi

Tlf. 73 55 11 43
https://www.ntnu.no/skolelab