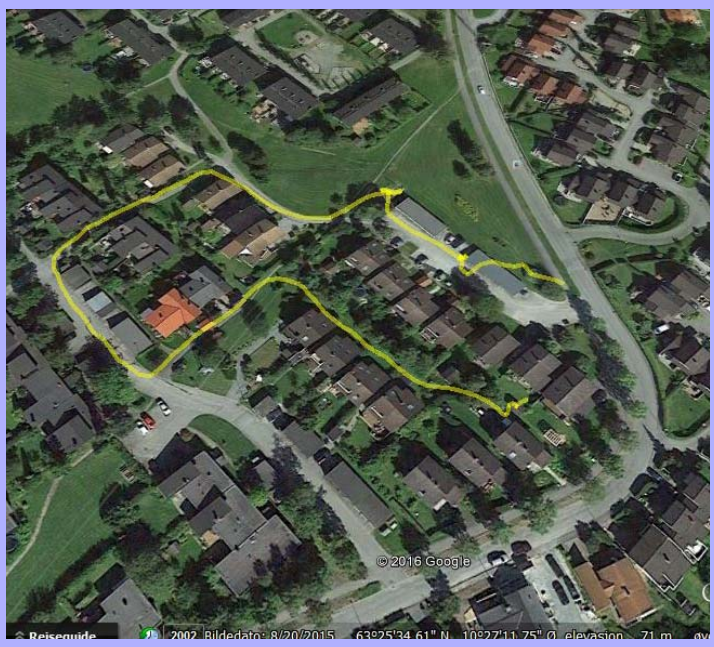
Rev 1.2

Nils Kr. Rossing

ESP32 – Sensornode med GPS og datalogging (YFL)



NTNU
Trondheim

Institutt for fysikk

Skolelaboratoriet
for matematikk, naturfag og teknologi

Juni 2021

ESP32 – Sensornode med GPS og datalogging (YFL)

Nils Kr. Rossing, Skolelaboratoriet NTNU

ESP32 – Sensornode med GPS og datalogging (YFL)

Trondheim 2021

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

Trykk: NTNU Grafisk senter

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

Kursholdere: 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 1.2 – 01.06.21

Forord

Heftet tar utgangspunktet i et opplegg laget som et forkurs i programmering til CanSat-kurset som har fokus på bruk av trykksensoren BMP180/280 som grunnlag for måling av lufttrykk. Ved å regne om fra endring i lufttrykk til endring i høyde, så inkluderes en praktisk anvendelse av et fysisk fenomen, nemlig sammenhengen mellom fallende lufttrykk med økende høyde over havet. Barometriske høydemålere brukes fortsatt i forbindelse med flytrafikk da den har en langt bedre nøyaktighet enn GPS så fremt instrumentet kalibreres ved hver flyvning. Vi har derfor inkludert kalibrering av instrumentet opp mot nærliggende meteorologiske stasjoner for å kunne måle absolutt høyde over havet. Et enkelt OLED display brukes for å vise resultatet av målingene. I tillegg har vi inkludert en GPS-mottaker og en datalogger (OpenLog) slik at vi kan ta vare på måledata på et micro SD-kort og merke dem med stedskoordinater. I tillegg bruker vi en Real Time Clock (RTC) som har batteri backup for å tidsstemple målingene våre.

ESP32 er valgt som mikrokontroller. ESP32 gir oss mulighet til å koble oss opp mot Internett med de fordeler det gir.

Det må innrømmes at den faglige tilknytningen til Yrkesfagene for denne delen av kurset kan virke litt svak. Dette skyldes hovedsakelig at kursmodulen ble utvikles under veis av videreutdanningskurset våren 2021, da det i løpet av kurset viste seg at kursdeltakerne måtte deles i to grupper for å tilpasse kurset til deltagernes faglige nivå. På grunn av begrensede ressurser har vi derfor vært tvunget til å gjenbruke undervisningsopplegg som er utviklet for andre formål. Det må likevel presiseres at det meste av det som tas opp i dette heftet er tema som ble foreslått av deltagerne under kursets 2. samling og som er særdeles relevant med tanke på at elevene skal utvikle egne produkter.

En spesiell takk til ***Harald Sæther,*** *student ved NTNU,* som på sparket laget et tillegg til biblioteket hos Circus of Things (CoT) slik at det lot seg gjøre å overføre lengde- og breddegrader og høyde sammen med måledata. Dette gjorde det mulig å plotte innsamlede måledata i kartet ved hjelp av CoT. En takk til ***Jaume Miralles*** som har utviklet Circus of Things og som tilbyr tjenesten vederlagsfritt.

Skolelaboratoriet ved NTNU
Juni 2021
Nils Kr. Rossing

Innhold

1 Innledning

Kurset er en påbygningningsmodul til kursene: *ESP32 – Grunnkurs programmering (YFL)* og Wi-Fi programmering med ESP32 – (YFL) og har som målsetning å gi kursdeltagerne en bredere programmeringserfaring.

Kursmodulen fokuserer blant annet på bruk av trykksensoren BMP280 som grunnlag for måling av temperatur og lufttrykk. Ved å regne om fra endring i lufttrykk til endring i høyde, inkluderes en praktisk anvendelse av et fysisk fenomen, nemlig hvordan en barometrisk høydemåler fungerer. Barometriske høydemålere brukes fortsatt i forbindelse med flytrafikk da den har en langt bedre nøyaktighet enn GPS så fremt instrumentet kalibreres ved hver flyvning. Vi har derfor inkludert kalibrering av instrumentet for å kunne måle absolutt høyde over havet. Et enkelt OLED display brukes for å vise resultatene av målingene. Både trykkmåleren og displayet benytter I^2C-buss.

Som en videre utvidelse av prosjektet har vi inkludert tre nye komponenter, GPS, datalagring på micro SD-kort ved hjelp av OpenLog og en klokkekrets som holder rede på tiden. Dette gir oss blant annet mulighet til å sammenligne to måter å måle høyden over havet, samt studere relativ nøyaktighet hos en billig GPS mottaker. I tillegg bruker vi dataloggeren for å samle inn måledata og legge dem på SD-kortet. Dette gjør det også mulig å behandle dataene i ettertid og ev. vise resultatet i Google Earth. OpenLog kan håndtering SD-kort fra 64MByte – 32 GByte.

Siden kurset har fokus på bruk av ESP32 så har vi valgt denne som mikrokontroller. Et viktig aspekt med bruk av ESP32 er oppkobling til Internett via Wi-Fi i tillegg til lavt strømforbruk. Det siste oppnås ved å legge ESP32 i dvale mellom målingene. Vi har som nevnt valgt å inkludere en ekstern RTC-klokke av typen DS1307, som gjør en ytterligere nedstengning mulig samtidig som den gir en nøyaktig tidsangivelse for målinger.

Ved å anvende mobiltelefonen som ruter for oppkobling til Internett, kan man ta med seg oppkoblingen samtidig som man samler inn og overfører dat til Circus of Things (CoT) via nettet. CoT gir dessuten mulighet til å knytte de innsamlede måledataene til geografiske lokasjoner. og vise dem på et kart.

1.1 Oppbyggingen av undervisningsopplegget

Undervisningsopplegget er bygget opp av en rekke *oppdrag* som skal løses. Hvert oppdrag tilfører ny kunnskap samtidig som det tilfører et nytt element i det endelige systemet. Vi kaller det *oppdrag* og ikke oppgaver. Vi mener at et oppdrag har større grad av autentisitet: "Noen har bedt om hjelp til å finne en løsning på en problemstilling". En oppgave kan lett oppfattes som konstruert for et undervisningsformål, hvilket også denne er. Likevel skal det være lett å se at oppdragene har relevans.

Det overordnete oppdraget er som følger:

> **Oppdraget:**
>
> *Det skal lages en barometrisk høydemåler som måler lufttrykk og temperatur og regner om til relativ høyde som vises på et display. Instrumentet skal kalibreres slik at den også kan vise absolutt høyde over havet. Høydemålerens nøyaktighet bestemmes ved å konsultere kartdata*

og gjøre fysiske målinger på stedet. Måledataene lagres på et micro SD-kort for ev. senere etterbehandling. Måleinstrumentet utstyres med en GPS-mottaker for å måle lengde- og breddegrader til målingene. Dessuten ønsker vi å sammenligne vår barometriske høydemåler med høydemåleren til GPS-mottakeren. En RTC-klokke tidsstempler målingene våre. Til sist vil vi koble instrumentet opp til CoT hvor vi ønsker å plotte målingene inn på kartet.

La oss ganske kort beskrive de ulike oppdragene:

Oppdrag 1: *Koble opp ESP280 og ESP32, installer biblioteket til kretsen og vis lufttrykk og temperatur i monitoren.*

Oppdrag 2: *Koble opp OLED displayet og vis lufttrykk og temperatur på displayet*

Oppdrag 3: *Bruk lufttrykket og bestemt relativ høyde med hjelp av omregningsformelen.*

Oppdrag 4: *Undersøk variasjonen i trykk og høydemålingen. Noter ytterpunktene. Utfør en midling av 100 målinger og studer hvordan det endrer variasjonsbredden. Noter resultatet.*

Oppdrag 5: *Utfør måleoppdrag og mål relativ høydeforskjell på to steder i nærområdet, f.eks. mellom to etasjer i skolebygget.*

Oppdrag 6: *Finn den nærmeste lokale metrologiske målestasjon og finn en oppdatert verdi av lufttrykket sammen med målestasjonens høyde over havet. Legg verdiene inn i programmet slik at du får en kalibrert høyde over havet på stedet du befinner deg. Sjekk med kartdata hvor godt måleverdien stemmer med virkeligheten.*

Oppdrag 7: *Koble opp GPS mottakeren, les ut stedskoordinatene og høyden og skriv ut i monitoren. Skriv også dataene ut på OLED-displayet. Sammenlign GPS høydedata og høyden beregnet av den barometriske trykkmåleren. Diskuter ev. avvik.*
Tips: *Temperaturmålingen må sannsynligvis tas ut av OLED-displayet siden det ikke er plass til mer enn 5 linjer.*

Installer Google Earth og kopier GPS-data inn i kommandolinjen og sjekk at avleste data stemmer med virkeligheten.

Oppdrag 8: *Monter og koble opp SD-kortleseren OpenLog som gjør det mulig å lagre måledata på et SD-kort. Skriv nummer på målingen, lengdegrad, breddegrad, høyde, lufttrykk, temperatur på en linje i filen.*

Oppdrag 9: *Vi legger mer merke til at eksakt samme verdier av lengde og breddegrad kommer flere gang og mistenker at samme verdi leses ut flere ganger etter hverandre. Lag en algoritme som ekskluderer like verdier før de skrives til dataloggeren.*

Oppdrag 10: *Gå en runde og samle inn lengde- og breddegrad og høyde fra en tur på ca. 10 minutter i nærområdet der du holder til. Last dataene over i en editor og inkluder dataene i en KML-fil slik at du kan vise turen i Google Earth.*

Oppdrag 11: *Koble opp ESP32 mot Circus of Things og lag konsoller som viser lufttrykk, temperatur, barometrisk høyde. GPS data lengde- og breddegrader og høyde skal knyttes til de enkelte målingene og ikke overføres som separate verdier.*

Oppdrag 12: *Sørg for å koble ESP32 opp mot Internett med mobiltelefonen slik at du kan ta med deg utstyret og gå en tur. Gå en runde og samle inn stedskoordinater, lufttrykk barometrisk høyde og temperatur.*

Gå inn på panelet på CoT og vis måledata av lufttrykk som funksjon av ruta du har gått.

Oppdrag 13: *Monter RTC-klokka og koble den opp på I^2C-bussen, og lag et lite program som stiller klokka og leser av nøyaktig tidsangivelse med dato og tid som skrives ut i monitoren. Bruk programmet til å stille klokka.*

Oppdrag 14: *Legg inn nøyaktig tidsangivelse i loggefila på OpenLog og i et konsoll hos CoT.*

1.2 Fagfornyelsen 2020

Vi har i denne sammenhengen i første rekke sett på kompetansemålene for yrkesfag Elektro.

Kursets innhold bygger på følgende punkter i LK20:

Skoleåret 2020/21 er det Vg1 som har læreplaner etter Fagfornyelsen (LK20).

Læreplanen for Vg1 Elektrofag fremhever programmering både i fagets kjerneelementer og i flere kompetansemål. Fra læreplanens kjerneelementer kan vi lese at: *"Kjerneelementet komponenter, kretser og utstyr handler om å regne på og utføre målinger på elektriske og elektroniske kretser og å kunne anvende utstyr og komponenter i helhetlige systemer. Det handler også om å kunne programmere utstyr og komponenter."*

Eksempler på kompetansemål fra læreplan på Vg1 i emnet Elektroniske kretser og nettverk:

Elektroniske kretser og nettverk

Elevene skal kunne ...

- *bygge og programmere et selvvalgt produkt som består av mikrokontroller, analoge kretser, relevante sensorer og aktuatorer for å oppnå ønsket virkemåte*
- *koble sammen ulike datateknologiske enheter til et system, konfigurere aktuelle komponenter ved hjelp av programvare og opprette kommunikasjon mellom enhetene for å oppnå ønsket virkemåte*
- *montere og konfigurere et mindre datanettverk med internettilkobling, utføre relevante målinger og gjøre rede for enkle tiltak for å sikre nettverket*
- *velge og bruke egnede instrumenter og programvare for å utføre målinger og feilsøking, og vurdere måleresultatet opp mot forventede verdier*

I denne modulen har vi lagt spesielt vekt på "*montere og konfigurere et mindre datanettverk med internettilkobling, utføre relevante målinger og gjøre rede for enkle tiltak for å sikre nettverket"*. Undervisningsopplegget dekker samtidig flere av de andre punktene.

2 ESP 32 - WROOM - 32

2.1 ESP 32 - WROOM - 32 med 38 pinner

ESP32 er et kontrollerkort bygget opp omkring en kombinert Wi-Fi og Bluetooth enhet som benytter 2,4 GHz båndet som kommunikasjonskanal.

Kortet er bygget opp omkring to prosessorer, en ESP-WROOM-32 og en CP2102N, sistnevnte tar seg av kommunikasjon med PC'en via USB-tilkoblingen. Lagerkapasiteten er som følgende: 448 kB ROM (Read Only Memory), 520 kB SRAM (Static Random Access Memory), i tillegg til 16 kB SRAM i en RTC (Real Time Clock).

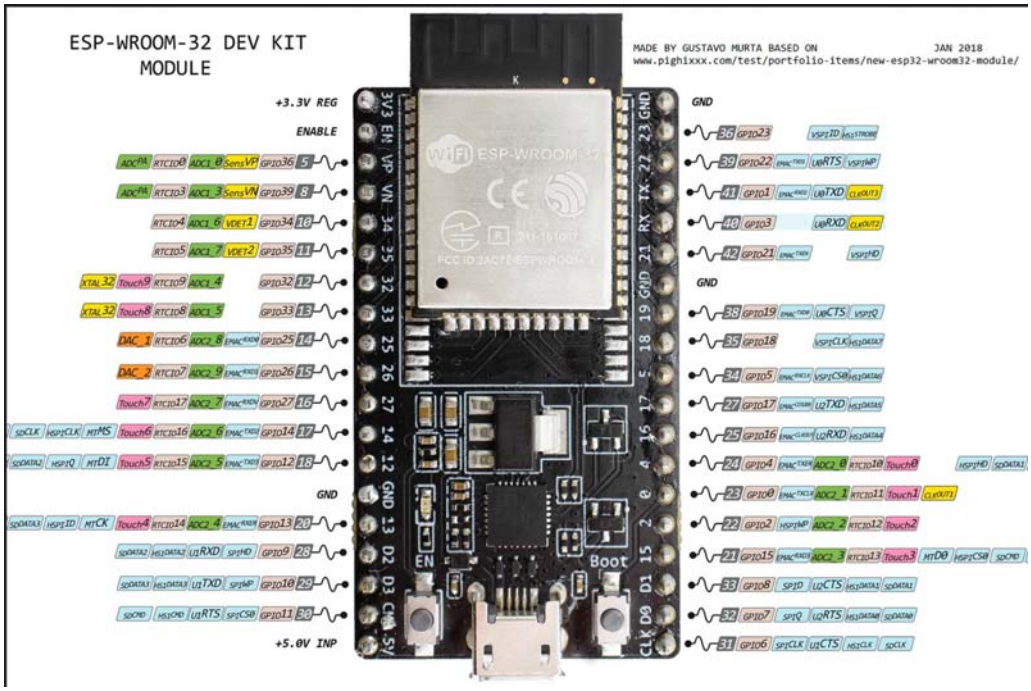
Senderen har en maksimal effekt på +12 dBm (8 mW) og mottakeren en følsomhet på –94 dBm Det er målt rekkevidder på over 200 meter mellom ESP32-enheter i åpent terreng[1].

Kortet har følgende porter:

- **Analoge og Digitale I/O-porter**
  Kortet har 34 digitale[2] inn/utporter (I/O-porter) som kan programmeres til enten å være inn- eller utganger. 12 bit ADC (Analog til Digital Converter) med opp til 18 kanaler. I tillegg har den 2 x 8 bit DAC (Digital til Analog Converter). 10 av inngangene kan fungere som touch sensor. I tillegg har den mulighet til å styre motorer med PWM og opp til 16 PWM for styring av LED.
- **Serie kommunikasjon:** Kortet har 4xSPI kanaler. Dessuten har det 2 x I^2C, 2 x I^2S [3] og 3 x UART[4]. Den har også mulighet for å kommunisere med IR (InfraRed) (Tx/Rx).
- **USB-kontakt** for direkte tilkobling av PC, for programmering av kortet og strømtilførsel. 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. USB C er designet for høyere strømtrekk, men det er PC'en som til syvende og sist bestemmer hvor mye den kan levere.
- **Strømtilførselen**[5] skjer via USB-kontakten som leverer 5V og reguleres ned til 3,3 V. Prosessorene har arbeidsspenning fra 2,2 – 3,3 V. Typisk strømtrekk er 80mA, maksimalt 500 mA. Kretsen har imidlertid flere nivåer av dvale tilstand fra "lett sovende" (0,8 mA) til "vinterdvale" (5 µA) hvor bare "real time" klokka (RTC) er aktiv.

1. https://randomnerdtutorials.com/esp-now-esp32-arduino-ide/
2. Dette kan variere noe fra versjon til versjon
3. I^2S er en enkel buss for overføring av digitale audio mellom kretser, med en klokkefrekvens på 44.1 kHz × 16 × 2 = 1.4112 Mbit/s som egner seg for overføring av 2 x 16 bits kanaler. https://en.wikipedia.org/wiki/I²S
4. UART – Universal Asynchronous Receiver/Transmitter, linje for toveis overføring av data.
5. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

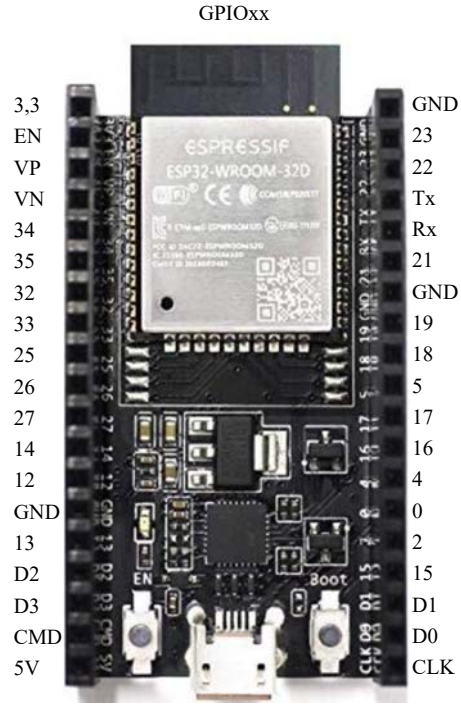
Figuren under viser pinningen for ESP-Wroom-32 DEV KIT med 38 pinner. Vi legger merke til at hver pinne har flere funksjoner.



- *Enable-knappen*
  Denne knappen resetter og starter programmet på nytt. Noen ganger kan det være nødvendig å starte programmet med Enable-knappen etter opplasting av programmet.
- *Boot-knappen*
  NB! ***Noen varianter av ESP32 krever at man bruker denne ved opplasting av programmet. Dette gjelder den varianten vi bruker i dette kurset.*** Andre varianter har innebygget elektronikk som laster opp programmet uten bruk av Boot. Ved å trykke Boot-knappen sammen med EN-knappen går kretsen inn i en tilstand der det er mulig å installere ny firmware via serieporten.

Innbygde sensorer og ADC og DAC[6]

Under brukes betegnelsen GPIOxx, tallet (xx) angir nummeret som er påført kretsen.



- *Hall sensor*
  En hall magnetfelt sensor er integrert på kortet og kan tilsluttes en av de analoge inngangene.
- *Temperatursensor*
  Kretskortet inneholder en innebygget temperatursensor.
- *Kapasitive berøringssensorer*
  10 av inngangene kan fungere som kapasitive berøringssensorer. Dette er Touch 0 –9: GPIO4, 0, 2, 15, 13, 12, 14, 27, 33 og 32.
- *ADC (Analog til digital omvandler)*
  Følgende innganger kan knyttes til de 18 12 bits ADC-inngangene, som er ordnet i to grupper ADC1 0, 3–7: GPIO 36, 39, 32, 33, 34 og 35. ADC2 0 – 9: GPIO 4, 0, 2, 15, 13, 12, 14, 27, 25 og 26. I tillegg har vi to som går under betegnelsen ADC^{PA}: GPIO36 og 39.
- *DAC (Digital til analog omvandler)*
  ESP32 har to 8 bits DA-omvandlere som betegnes DAC_1 og DAC_2, og er tilkoblet GPIO35 og 26.
- *IR-kontroll*
  I alt 8 kanaler kan betjene ulike IR-protokoller.

6. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

Hvilke pinner kan vi bruke til ulike ting?

Vi trenger et skjema som gir oss en enkel oversikt over hvordan vi kan bruke de ulike pinnene. De alle fleste pinnene har mange forskjellige funksjoner. Vi velger her og ta ut de vi har mest bruk for og heller henvise til andre referanser når det gjelder pinner med mer spesielle funksjoner. Vi ønsker å forholde til pinnenes GPIO[7] nummer.:

Innerst slik kretsen er merket, ytterst GPIO numrene

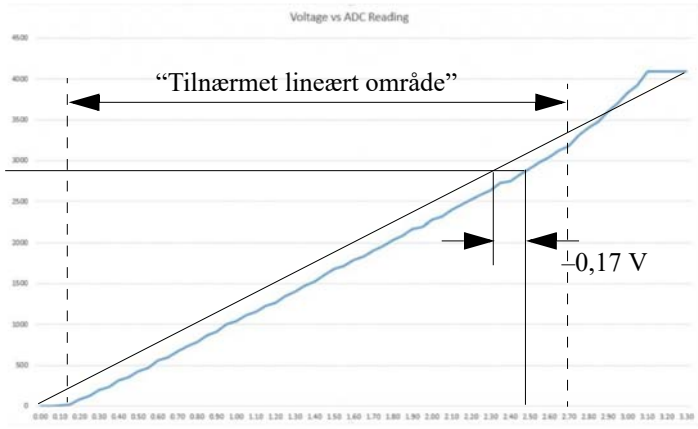
Tabellen under viser anbefalt bruk (Br. 1 – Digital, Br. 2 – Analog og Br. 3/4 – I^2C og Touch).

GPIO	Merk	Br. 1	Br. 2	Br. 3	GPIO	Merk	Br. 1	Br. 2	Br. 4	GPIO	Merk	Br. 1	Br. 2	Br. 3	GPIO	Merk	Br. 1	Br. 2	Br. 3
0	0	I/O	A2-1	T4	10	D3	x	x	x	20	-	-	-	x	30	-	-	-	
1	Tx	Tx			11	CMD	x	x	x	21	21	I/O	SDA (I^2C)		31	-	-	-	
2	2	I/O	A2-2	T2	12	12	O	A2-5	T5	22	22	I/O	SCL (I^2C)		32	32	I/O	A1-4	T9
3	Rx	Rx			13	13	I/O	A2-4	T4	23	23	I/O			33	33	I/O	A1-5	T8
4	4	I/O	A2-0	T0	14	14	I/O	A2-6	T6	24	-	-	-		34	34	I	A1-6	
5	5	I/O			15	15	I/O	A2-3	T3	25	25	I/O	A2-8	DAC1	35	35	I	A1-7	
6	CLK	x	x		16	16	I/O			26	26	I/O	A2-9	DAC2	36	VP	I	A1-0	
7	D0	x	x		17	17	I/O			27	27	I/O	A2-7	T7	37	-	-	-	
8	D1	x	x		18	18	I/O			28	-	-	-		38	-	-	-	
9	D2	x	x		19	19	I/O			29	-	-	-		39	VN	I	A1-3	

7. GPIO General Purpose Input Output port

Merk:

1. Alle GPIO som er skravert i tabellen finnes ikke tilgjengelig på pinner
2. Ikke alle analoge innganger A2 0–9, kan brukes sammen med Wi-Fi
3. GPIO-0 har intern pullup når den brukes som digital inngang
4. Alle GPIO kan konfigureres for interrupt
5. Alle GPIO kan konfigureres som PWM (Pulsbredde modulering)
6. ADC (Analog til Digital Converter) er ikke lineær i endene som vist i figuren under. Dette medfører at ved omregning fra digital verdi til spenning (med bruk av 3,3V/4096) så vil en kunne ha et typisk avvik på –0,17V i det "tilnærmet lineære området" av kurven. Dvs. man må legge til +0,17V. Verdien kan være forskjellig fra komponent til komponent.



Hvordan lese og skrive til de ulike portene

Nå har vi funnet ut hvilke porter og GPIO nummer som kan brukes til digitale, analoge og touch formål. Så er spørsmålet hvordan vi når disse portene fra programmet? Det er `pinMode()`-funksjonen eller funksjonen vi bruker for å avlese eller skrive til GPIO-porten som definerer dens funksjon:

Digital (inngang)

Vi kan lese verdien på den digitale inngangen slik:

```
int verdi                // Kan holde heltallsverdier med fortegn
pinMode(16, INPUT);      // Definerer GPIO 16 som en digital inngang
verdi = digitalRead(16); // Leser GPIO 16 som en digital inngang
```

Deklarasjonene står gjerne i starten av programmet, `pinMode()` står vanligvis i void setup()-funksjonen og definerer GPIO som en digital inngang, og avlesning av porter er gjerne plassert i void loop()-funksjonen.

Digital (utgang)

Tilsvarende kan vi definere en GPIO-port som en utgang:

```
pinMode(17, OUTPUT); // Definerer GPIO 17 som en digital utgang
digitalWrite(17, HIGH); // Gir GPIO-port 17 verdien høy
```

Som vi ser så forholder vi oss til GPIO-portnumrene.

Lese en analog inngang

En analog port trengs ikke defineres, heller ikke trenger vi å spesifisere om det gjelder ADC1 eller ADC2 vi ønsker å bruke. Det er kun GPIO-nummeret som gjelder. Vi må imidlertid bruk en GPIO som kan konfigureres som en analog inngang:

```
int verdi; // Vil ha verdier 0 - 4096, 12 bit
verdi = analogRead(34); // Hvor GPIO-34 angir ADC1_CH6
```

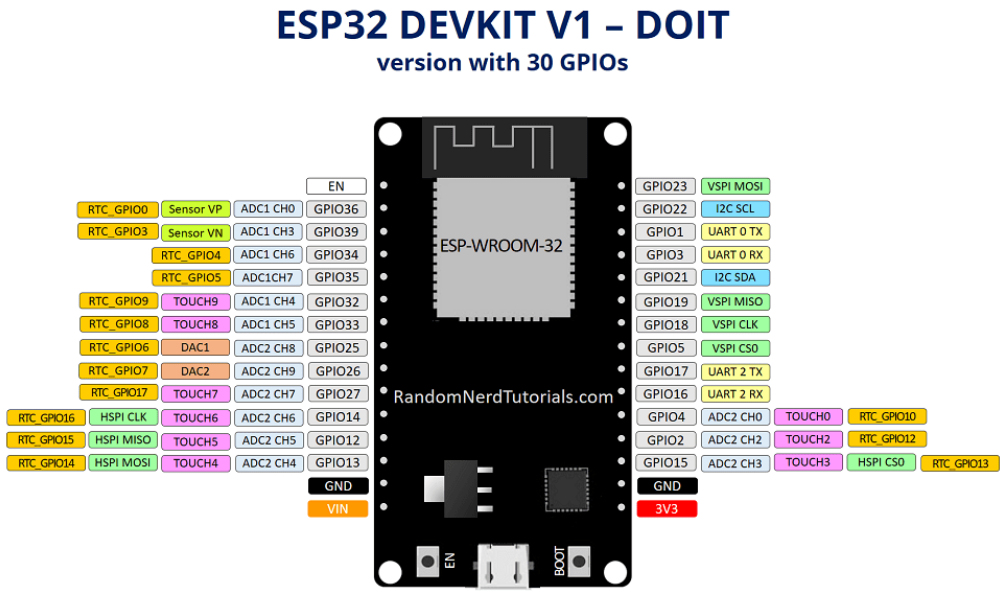
Touch – funksjon (inngang)

Verdien fra en Touch pinne kan nås i programmet ved følgende kommando:

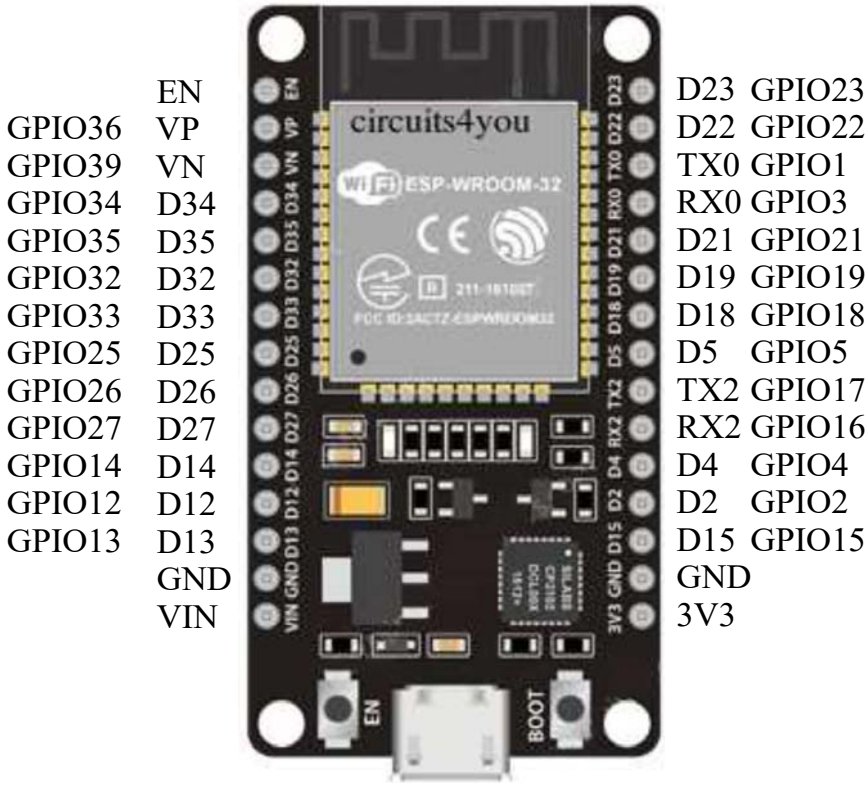
```
int verdi; // Vil ha verdier typisk mellom 10 og 100
verdi = touchRead(4); // Leser tilstanden til GPIO 4, Touch 0
```

2.2 ESP - WROOM - 32 med 30 pinner

Den samme kretsen finnes i flere utgaver vi har av ulike årsaker bl.a. på grunnlag av pris og tilgjengelighet valgt å bruke en utgave med 30 pinner: ESP32 DEVKIT V1 - DOIT.



Vi kan ganske kort sammenligne de to kretsene og se hva som er forskjellen.



Forskjellen på ESP32 (38 pin) og ESP32 (30 pin) er følgende:

GPIO	Merk	Br. 1	Br. 2	Br. 3	GPIO	Merk	Br. 1	Br. 2	Br. 4	GPIO	Merk	Br. 1	Br. 2	Br. 3	GPIO	Merk	Br. 1	Br. 2	Br. 3
0	0	I/O	A2-1	T4	10	D3	x	x	x	20	-	-	-	x	30	-	-	-	
1	Tx	Tx			11	CMD	x	x	x	21	21	I/O	SDA (I^2C)		31	-	-	-	
2	2	I/O	A2-2	T2	12	12	O	A2-5	T5	22	22	I/O	SCL (I^2C)		32	32	I/O	A1-4	T9
3	Rx	Rx			13	13	I/O	A2-4	T4	23	23	I/O		MOSI	33	33	I/O	A1-5	T8
4	4	I/O	A2-0	T0	14	14	I/O	A2-6	T6	24	-	-	-		34	34	I	A1-6	
5	5	I/O		CSO	15	15	I/O	A2-3	T3	25	25	I/O	A2-8	DAC1	35	35	I	A1-7	
6	CLK	x	x		16	16	I/O			26	26	I/O	A2-9	DAC2	36	VP	I	A1-0	
7	D0	x	x		17	17	I/O			27	27	I/O	A2-7	T7	37	-	-	-	
8	D1	x	x		18	18	I/O		CLK	28	-	-	-		38	-	-	-	
9	D2	x	x		19	19	I/O		MISO	29	-	-	-		39	VN	I	A1-3	

Forskjellen på ESP32 (38 pin) og ESP32 (30 pin) er følgende: Alle GPIO som er skravert i tabellen over finnes ikke tilgjengelig på pinner hos ESP32 (30 pin). Dvs. at ESP32 (30 pin) mangler følgende i forhold til ESP32 (38 pin): En GND, pin D0 – D3, CLK, CMD i tillegg til GPIO 0. Dvs. alt-i-alt så er det meste med hos ESP32 (30 pin) og vi velger å betrakte de to variantene som likeverdige til våre formål.

2.3 Energisparing

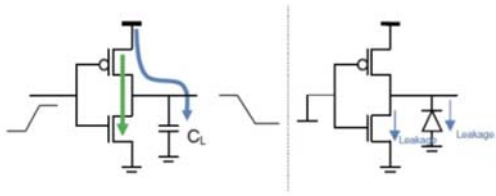
En av de store fordelene ved ESP32 er at kretsen kan legges helt eller delvis i dvale. Dette er en viktig egenskap til IoT sensornoder som er batteridrevet og kun trenger å gjøre målinger en gang i blant. Etter hver måling kan kontrolleren legges i dvale for å spare energi.

La oss først se litt å hva som påvirker energiforbruket hos digitale elektroniske kretser. Problematikken er ganske kompleks så vi nøyer oss med å betrakte en forenklet modell.

2.3.1 Faktorer som påvirker energiforbruket i digitale elektroniske kretser

Det er flere måter å redusere energiforbruket på hos digitale logiske kretser.

Det snakkes ofte om *dynamisk* og *statisk* energiomsetning. Digitale logiske kretser er bygget opp av en mengde elektronisk styrte transistorbrytere som slås av og på.



Omslagshastigheten kan til sine tider komme opp i GHz. For å redusere forbruket benytter man komplementær elektronikk som er forsøkt forklart i figuren til høyre[8]. Som vi ser på tegningen til venstre så består bryteren av to FET-transistorer[9]. Når den ene er åpen er den andre lukket og omvendt. Når inngangen er lav, så er den øverste åpen og den nederst stengt. Dermed vil utspenning ligge høyt, vi har en inverter, "1" inn gir "0" ut og omvendt. I prinsippet skulle en slik bryter ikke trekke strøm, men slik er det dessverre ikke. Akkurat ved omslaget må man flytte ladning, hvilket vil resultere i en kort strømpuls. Jo oftere transistorene slår om, jo oftere går det en strømpuls og jo mer energi må tilføres den digitale bryteren. Dette er *dynamisk energiomsetning* og er avhengig av omslagshastigheten, eller klokkefrekvensen til kretsen. Den dynamiske energiomsetningen kan derfor reduseres ved å redusere klokkefrekvensen til kretsen.

Redusere lekkasjestrømmer: I tegningen over til høyre ser vi et par blå piler. Dette er lekkasjestrømmer fordi en bryter ikke er helt "tett", noe strøm lekker gjennom. Denne lekkasjen er ikke avhengig av hvor ofte kretsene slår om, men hvor "tett" vi klarer å gjøre transistorene. Dette bidrar til den *statiske energiomsetning*.

8. Figuren er hentet fra https://semiengineering.com/knowledge_centers/low-power/low-power-design/power-consumption/
9. Denne typen konfigurasjon kalles CMOS (Complementary metal-oxside-semiconductor), NMOS og PMOS i kombinasjon.

$$P_{Leakage} = f\,(U_{dd},\ V_{th},\ W/L) \qquad (2.1)$$

Lekkasjeeffekten ($P_{Leakage}$) er en funksjon av forsyningspenningen (U_{dd}), terskelspenningen (V_{th}) for omslag fra høyt til lavt og omvendt, og dimensjonene på transistorene (Bredde/Lengde – W/L).

Redusert kapasitans: I tegningen over til venstre ser vi også en kondensator (C_L) . Kondensatorer som utsettes for varierende spenninger vil medføre opp- og utladinger, dvs. det vil skje en ladningsforflytning som en konsekvens av at vi slår bryterne av og på. Slike kondensatorer er ofte uønsket og en konsekvens av at transistorer og andre komponenter har en fysisk utstrekning. Forflytning av ladning er strøm og vil fungere som energiomvandlere. Jo større kondensatorene er, jo lengre tid tar det å flytte på ladning. Skal vi få ting til å gå fort så ønsker vi minst mulig kapasitans slik at lite ladning trengs å flyttes. Det kan vi oppnå med å lage ting smått. Små transistorer er derfor raskere, men komponenter som er små og ligger tett inntil hverandre vil gjerne gi økt kapasitans på denne måten. Så her får vi en avveining. Kondensatorer vil derfor bidra til den dynamiske energiomvandlingen. Små dimensjoner vil dessuten medføre begrensninger i strømmen.

Redusert spenning: Vi vet at effekt som omsettes i motstand er avhengig av kvadratet av spenningen over motstanden:

$$\text{Effekt} = U^2/R_L \qquad (2.2)$$

Vi ser at klarer vi å redusere spenningen på kretsen så er det mye å vinne på redusert effektomvandling. Tradisjonelt har det vært vanlig å bruke 5V som forsyningsspenning for digitale kretser. Etter hvert går man mer og mer over til å bruke 3,3V eller lavere. På den måten sparer man energi.

Reduserte aktivitet: En annen viktig metode for å spare energi er å redusere aktiviteten i kretsen. Dette gjør man ganske enkelt ved å slå av strømmen på deler av kretsen når disse ikke trengs. En typisk situasjon er når en krets skal gjøre målinger en gang i blant. I mellomtiden kan store deler av kretsen være avslått. En må bare passe på å slå dem på når målingene skal utføres. Dette skal vi se nærmere på i neste avsnitt.

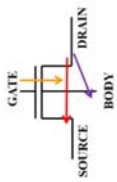
Følgende formel uttrykker det vi har snakket om på en kompakt måte:

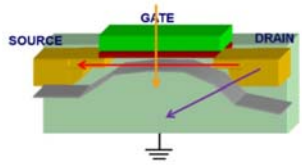
$$P_d = \alpha \cdot f \cdot C \cdot U^2 \qquad (2.3)$$

Hvor:

α – Aktivitetsnivået.
f – Frekvensen, hyppighet for omslag.
C – Kapasitans.
U – Supplyspenning.

Men også lekkasjestrømmer bidrar til effekt-omvandlingen. Figuren til høyre viser en FET-transistor, hvor det er antydet hvilke veier lek-kasjestrømmene kan ta. Det er summen av disse som utgjør lekkasjen i enkelt transistorer. Den totale lekkasjen er summen av alle transistorer som bidrar til lekkasjestrømmen hver på sin måte.





Jo mindre transistorene er jo mer lekkasje, samtidig vil små transistorer gi små kapasitanser og dermed redusere de kapasitive strømmene. Små transistorer vil normalt gjøre at hastigheten går opp.

I tillegg finnes det flere andre metoder for å få ned energiforbruket som ikke vil bli omtalt her.

Som vi skal se så trenger en aktiv ESP32 en god del strøm, spesielt ved sending av data. Det er derfor aktuelt å legge den i dvale for å spare energi, når det er deler av kretsen som ikke trengs.

Illustrasjonene under er hentet fra nettsiden til Last Minute Engineers[10] som på en meget god måte illustrerer de ulike dvaletilstandene.

2.3.2 Ulike dvaletilstander

En ESP32 består av en rekke deler. Figuren under viser mikrokontrollerens ulike deler:

- **Wi-Fi –** Dette er enheten som setter opp protokollene og pakker data for å kunne kommuniserer med rutere og videre ut på Internett.
- **Bluetooth –** Dette er enheten som pakker data-ene som gjerne brukes når kretsen skal kommunisere med annen elektronikk som PC-er eller Smarttelefoner.
- **Radio –** Dette er enheten som gjør de digitale signalene om til radiosignaler som sendes til antenna og videre som elektromagnetiske bøl-ger. Sendeeffekten er med på å bestemme rekkevidden, men også energiomvandlingen.
- **ESP32 Core** – Dette er hovedprosessoren som kjørerer programmet og utfører beregninger og flytter rundt på data.
- **ULP-processor** – Dette er en såkalt co-prosessor som har som hovedoppgave å gjøre målin-ger, gjøre om analoge signaler til digitale, kommunisere med sensorer via I^2C-bussen o.l.. Prosessoren har også tilgang til et ganske langsomt lager knyttet til RTC-enheten (Real Time

10.https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/

Clock), og slik kan lagre unna og hente fram viktig informasjon, mens resten av kontrolleren er i dvale. ULP står for Ultra Low Power.

- **Peripherals** – Denne enheten tar hånd om kommunikasjonen med omverdenen gjennom analoge og digitale porter.
- **RTC and RTC peripherals** – "Real Time Clock" holder blant annet rede på tiden og har et lager som kan ta vare på viktige data mens resten av kretsen er i dvale. Den har også en viktig funksjon i forbindelse med vekking av kretsen etter dvale.

Oversikt over ulike dvaletilstander

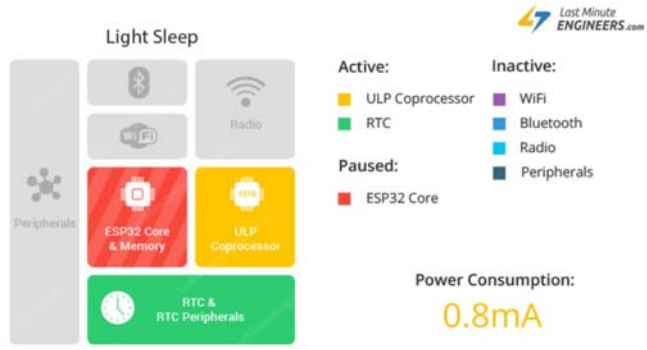
Aktiv mode: I denne tilstanden er alle enhetene operative og i full aktivitet. Avhengig av blant annet sendereffekten så kan forsyningsstrømmen bli ganske høy. I enkelte tilfeller kan strømmen nærme seg 800mA i korte øyeblikk dersom begge kommunikasjonsenhetene (Wi-Fi og Bluetooth) er aktive.

Sendereffekt fra Wi-Fi kan være fra 13 – 21 dBm (ca. 20 – 130 mW).

ESP32 modem sleep: I denne tilstanden er all kommunikasjon med omverdenen avslått og strømtrekket er dramatisk redusert.

Vi legger imidlertid merke til at begge prosessorene er i full aktivitet, dvs. at programmet går. Klokkehastigheten for prosessoren kan kjøres i lav (3 mA) eller høy hastighet (20 mA) noe som påvirker strømforbruket sterkt. Prosessorene kan vekke opp Wi-Fi, Bluetooth og radioenheten etter behov. I denne modus kan ***ESP32 også vekkes opp av et kallesignal fra ruteren*** som gjerne kommer med jevne mellomrom på fra 100 – 1000 ms.

ESP32 light sleep: I denne tilstanden reduseres strømtrekket ytterligere. Dette gjøres ved å slutte å klokke deler av kretsen slik at enkelte vipper og logiske kretser ikke skifter tilstand (0–1/1–0) fordi det nettopp er omslagene som trekker strøm. Dette kalles "Clock gating". Før kretsen går inn i "light sleep" må kretsens indre tilstand lagres slik at den kan kalles tilbake ved vekking.



ESP32 deep sleep: I denne tilstanden er det meste koblet ned slik at bare ULP coprosessoren og Real Time Clock med sitt tilhørende "langsomme" lager er aktive. ULP coprosessoren utfører fortsatt målinger og vekker hovedprosessoren på bakgrunn av aktivitet på portene. Det kan f.eks. være knapper som blir trykket. Sammen med hovedprosessoren er også datalagringsenheten slått av. Nødvendig data for å kunne vekke opp kretsen er å finne i RTC-laget slik at det skal være mulig å få kretsen på nett når den vekkes.

ESP32 Hibernation mode: I denne tilstanden går så og si hele kretsen i dvale, såkalt "vinterdvale". Kun RTC-klokka går og noen GPIO-funksjoner holdes "i live" slik at det skal være mulig å vekke kretsen igjen. Til og med RTC-lageret er koblet ned. Når kretsen så våkner vil programmet starte fra begynnelsen, uten hukommelse av hva som skjedde før nedstengningen.

2.3.3 Programtekniske grep for å legge ESP32 i dyp dvale.

I dette avsnittet skal vi se nærmere på hvordan vi går fram for å legge kretsen i dyp dvale (deep sleep). Tilsvarende kommandoer finnes for de andre dvaletilstandene.

Følgende må på plass:

1. **Etablering av betingelsene for dvalen**
   - Her legger man bl.a. forutsetningen for hvordan vekking skal skje.
2. **Igangsetting av dvalen**
   - Her kalles funksjonen som legger kretsen i dvale.
3. **Vekking av dvalen**
   - Her vekkes mikrokontrolleren fra dvalen, ut fra premissene som ble lagt under punkt 1.

Etablering av betingelsene for dvalen

Betingelsene for vekking fra dvalen kan være forskjellige og settes gjerne opp i setup()-funksjonen. Følgende muligheter finnes:

- **Timer**
  Mikrokontrolleren går ut av dvale etter en viss tid. Dette er gunstig dersom det skal samles inn data med jevne mellomrom. F.eks. hvert minutt, hver time eller en gang i døgnet, tiden skal spesifiseres i µs. Det er en spesiell timer i Real Time Clock (RTC) som sørger for dette.

  ```
  esp_sleep_enable_timer_wakeup(<spesifiseres i µsekunder>);
  ```

  ... spesifiserer hvor lenge kretsen skal ligge i dvale. Gjøres gjerne i void setup()-funksjonen

  ```
  esp_deep_sleep_start();
  ```

  ... legger kontrolleren i dvale for den angitte tiden.

- **Touch pad**
  Følgende porter for ESP32 har evnen til fungere som Touch pad, dvs. at portene reagerer på berøring: 0, 2, 4, 12–15, 27, 32 og 33

  ```
  touchAttachInterrupt(<pinne nr.>, callback, <terskelnivå>;
  ```

  callback er funksjonen som definerer hva som er det første som skal skje etter at kretsen er vekket. Denne funksjonen kan ev. også legge kontrolleren tilbake i dvale.

  ```
  void callback()
    {
    // Definer hva som skal gjøres ved oppstart
    }
  ```

  terskelnivå bestemmer hvor følsom Touch pad'en skal være. 40 kan være et grei verdi å begynne. Økes verdien øker følsomheten.

  ```
  esp_sleep_enable_touchpad_wakeup();
  ```

  Denne definerer touch pad som kilde til vekking og legges i setup()-funksjonen.

  ```
  esp_deep_sleep_start();
  ```

  Legger kontrolleren i dyp søvn. Funksjonen plasseres i programmet der man ønsker at "deep sleep" skal inntreffe.

  Se eksempel: https://lastminuteengineers.com/esp32-deep-sleep-wakeup-source/

- **Ekstern vekking**
  Denne fungerer slik at kretsen vekkes ved at en eller flere porter blir lagt lav eller høy, etter hvordan det er spesifisert. Det finnes to typer for ekstern vekking:
  - *ext0* – Vekking skjer ved hjelp av at en spesifikk port 0, 2, 4, 12–15, 25–27, 32 og 33, enten får høyt eller lavt nivå avhengig av hva som er spesifisert. Denne metoden krever at RTC-peripherals har spenning.
  - *ext1* – Vekking skjer ved at et sett av porter legges enten logisk høyt eller lavt etter som det er spesifisert. I tillegg må det spesifiseres hvilke porter vi ønsker skal være aktive. Dette gjøres ved å definere en maske, se under.

  ext0 – Følgende funksjon sørger for å legge kontrolleren i dyp dvale av typen ext0:

  ```
  esp_sleep_enable_ext0_wakeup(GPIO_PIN, LOGIC_LEVEL);
  ```

  `GPIO_PIN` spesifiserer hvilken port som skal sørge for vekking.
  `LOGIC_LEVEL` er enten `LOW` eller `HIGH` og spesifiserer hvilket nivå som gir vekking.
  Funksjonen legges gjerne i void setup()-funksjonen.

  ```
  esp_deep_sleep_start():
  ```

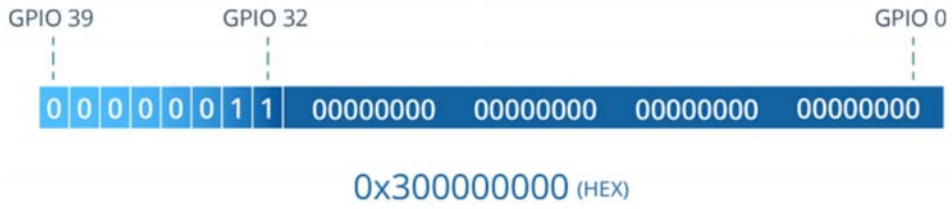
  Sørger for å legge kontrolleren i dvale.

  ext1 – Følgende funksjon sørger for å legge kontrolleren i dyp dvale av typen ext1:

  ```
  esp_sleep_enable_ext1_wakeup(BUTTON_PIN_MASK, LOGIC_LEVEL);
  ```

  `BUTTON_PIN_MASK` er en maske som bestemmer hvilke porter som er aktive for vekking.
  `LOGIC_LEVEL` er enten `LOW` eller `HIGH` og spesifiserer hvilket nivå som sørger for vekking.

  Masken settes opp på følgende måte:



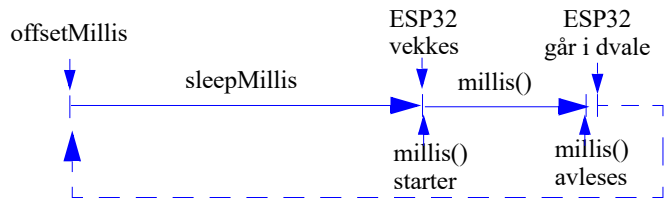

  ```
  #define BUTTON_PIN_MASK 0x300000000
  ```

  Denne setter port 32 og 33 aktive (1) og resten inaktive. Legg merke til at to siffer i Hex styrer 8 binære siffer.

2.3.4 Å gi målinger et tidsstempel ved bruk av "deep sleep"

Normalt vil vi kunne bruke millis()-funksjonen for å ta tiden fra vi slår på strømmen til kretsen, da denne funksjonen måler tiden i millisekunder fra programmet startet eller ble resatt.

Det viser seg imidlertid at "deep sleep" "dreper" denne tidtakeren og starter den på nytt for hver dvaleperiode. La oss se hvordan vi kan komme rundt dette problemet. I figuren til høyre har vi forsøkt å indikere hvordan vi resonerer:



offsetMillis = offsetMillis + sleepMillis + millis()

Vi definerer en variabel og legger den et sted i lageret som ikke slettes under dvaleperioden. Vi vet at det finnes et lite lagerområde i Real Time Clock (RTC) området som har denne egenskapen. For å kunne definere en lagerplass her så bruker vi følgende kommando:

```
RTC_DATA_ATTR unsigned long millisOffset = 0;
```

`millisOffset` er navnet vi har gitt variabelen som holder den akkumulerte verdien av klokka ved det tidspunktet vi legger ESP32 kretsen i dvale. Neste gang den legges i dvale er det gått tiden den har ligget i dvale (`sleepMillis`) som vi har spesifisert, pluss tiden den har brukt til å kjøre kommandoene i programmet før den legges i dvale på nytt. Denne tiden kan vi lese av fra funksjonen `millis()`.

```
offsetMillis = offsetMillis + sleepMillis + millis();          (2.4)
```

Vi legger også merke til at det er et lite tidsintervall fra vi leser av millis() og til kretsen faktisk går i dvale som vi ikke tar med i regnestykket vårt. Denne feilen vil akkumuleres over tid og kunne gi betydelig feil. Det er om å gjøre å redusere denne mest mulig når vi programmerer kretsen. Dersom dette feilintervallet er relativt konstant er det mulig å korrigere noe for det. Vi må i så fall måle avviket over tid og finne avviket som tilføres den akkumulerte tiden hver gang kretsen går i dvale, for så legge dette avviket inn i ligningen 2.4.

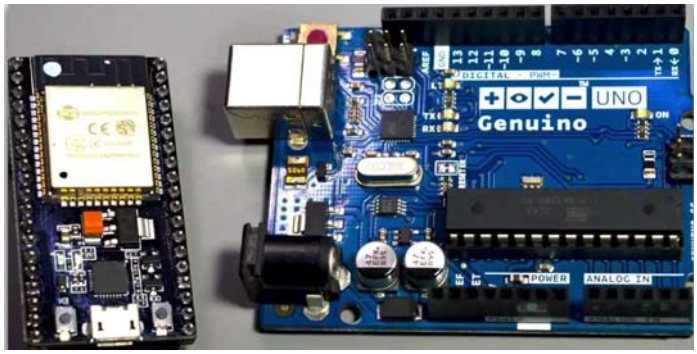
For nærmere beskrivelse av denne problematikken se: https://www.robmiles.com/journal/2020/1/22/esp32-retaining-timing-over-deep-sleep

3 Programmering

Kapittelet gir noen grunnleggende tips til programmering av Arduino og ESP32. De av dere som har erfaring med denne type programmering kan nøye dere med lese avsnitt 3.1, og 3.3, side 32, som omtaler installasjon av ekstrapakken som gjør det mulig å bruke Arduino-grensesnittet for ESP32.

3.1 Forskjeller og likheter hos Arduino og ESP32

La oss ganske kort sammenligne ESP32 og en "vanlig" Arduino, f.eks. UNO[11].



Av utseende er de to kortene svært forskjellige, dette gjelder på de fleste områder som f.eks. innebygde funksjoner, lager- og prosesseringskapasitet, antall GPIO-porter og kommunikasjonsmuligheter, ikke minst det siste siden ESP32 både har Bluetooth og Wi-Fi (selv om noen Arduino varianter også har slikt). At prosesseringshastigheten er så høy skyldes dels den høye klokkefrekvensen på 160 eller 240 MHz (16 MHz på de fleste Arduino) og at ESP32 har to prosessorkjerner.

Hva som likevel gjør ESP32 så attraktiv for Arduino-brukere, er programvaren som er utviklet av Espressif, som har gjort det mulig å bruke mesteparten av de samme kommandoene og det samme programmeringsmiljøet (IDE) som for Arduino. Likeså kan over 90% av bibliotekene som er utviklet for Arduino også brukes direkte for ESP32. Her må man imidlertid være litt oppmerksom og sjekke om det finnes spesialtilpassede biblioteker utviklet spesielt for ESP32. For å kunne bruke Arduino IDE til å programmere ESP32, trengs det imidlertid noe tilleggsprogramvare som vi snart skal komme tilbake til (se avsnitt 3.3, side 32).

Naturlig nok finnes det også noen tillegg i språket som er knyttet til de tilleggsfunksjonene som ESP32 tilbyr, ellers er det omtrent total overlapp. Programmer som f.eks. er skrevet for Arduino UNO kan godt lastes inn i editoren og kompileres og overføres til ESP32. Det man imidlertid må passe på er ***at IO-portene er plassert annerledes***. Dette er jo et svært viktig poeng, dog er det som oftest håndterbart.

11. Stoffet til denne sammenligningen er hentet fra https://techexplorations.com/guides/esp32/begin/esp32ard/

Normalt vil prisen for ESP32 ligge under Arduino. Man kan derfor se det slik at man får økt lagerkapasitet, hastighet og Wi-Fi/Bluetooth, gratis.

ESP32 er likevel ganske kompleks dersom man ønsker å utnytte mulighetene. Det anbefales derfor ikke å begynne opplæringen med denne, men gjerne starte med en Arduino UNO. Vi velger likevel å gjøre det for at skrittet over til Internett of Things (IoT) skal være lettere.

3.2 Installasjon av programvare

Om du har installert Arduino IDE tidligere kan du hoppe over dette avsnittet og gå dirkete til avsnitt 3.3, side 32.

La oss først se hvordan vi kan installere: Arduino programeditor, IDE.

3.2.1 Arduino programeditor, IDE

Nedlasting av programvare

Arduino programeditor og kompilator hentes fra:

http://arduino.cc/hu/Main/Software

Versjonen som er brukt i denne sammenheng er 1.8.13. Filen som har navnet *arduino-1.8.13-windows,* er pakket som en zip-fil. En tilsvarende fil er tilgjengelig for Mac fra samme nettsted.

Det er også helt greit å bruke Windows installer versjonen som finnes på samme side. Det er imidlertid erfart noen uregelmessigheter ved bruk av Windows app-versjonen så unngå gjerne den.

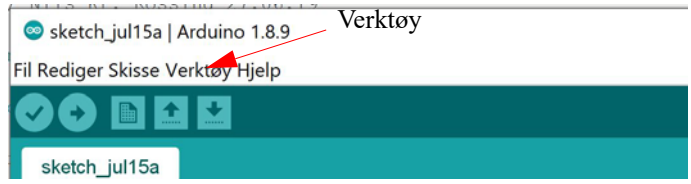
Installasjon av programvaren:

1. Klikk på *Windows installer* og følg anvisningen under installasjonen. For MAC-brukere klikk på *MAC OS X*

2. Programmet startes ved å klikke på programikonet:  .

3. Koble USB-kabelen til ønsket USB-port på PC-en.

4. Klikk på *Verktøy (Tools)* på menylinjen og velg *Kort*. Her velges hvilken variant i Arduino-familien du ønsker å jobbe med.

   Etter at vi har installert tilleggsprogramvare for DOIT ESP-32 DEVKIT V1 så velger vi denne (se avsnitt 3.3).

5. Klikk på *Verktøy* på menylinjen og velg *Port*. Sjekk at riktig port (Com?) er valgt. Normalt står det hvilken Arduino-variant porten er koblet til eller velg det høyeste nummeret.

Programmet er nå klart til bruk og du kan skrive inn programlinjene. Når programmet er ferdig skrevet, skal det *kompileres*, dvs. overføres til en binærkode som mikrokontrolleren forstår. Dersom programmet inneholder ulovlige kommandoer eller skrivefeil, vil kompilatoren varsle om det og vise på hvilken linje feilen er avslørt. Det er ikke nødvendigvis alltid der feilen befinner seg.

Dernest skal programmet lastes ned til mikrokontrollerens minne (Arduino-kortet). Dette gjøres ved å trykke på knappen  .

Kort oversikt over Arduino-editoren

Man finner følgende kommandoikoner på den grafiske menylinjen:

 Kompiler og verifiser at koden er riktig

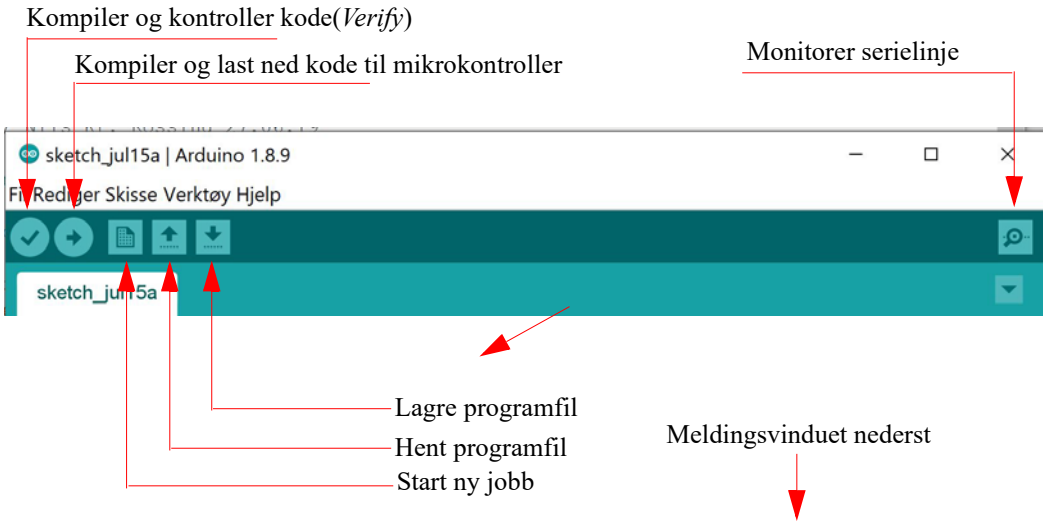
 Kompiler og last ned programmet til mikrokontrolleren

 Hent nytt "arbeidsark", start ny jobb

 Hent en eksisterende programfil

 Lagre programfil

 Monitorer data sendt tilbake fra mikrokontrollerkortet på serielinjen



Manglende kontakt med kortet

Det hender at en ikke oppnår umiddelbar kontakt med Arduino-kortet når en forsøker å laste ned et program. Feilmeldingen: *avrdude: stk500_getsync(): not in sync: resp=0x00* i meldingsvinduet betyr at det ikke oppnås kontakt med kortet. Dette kan skyldes flere ting:

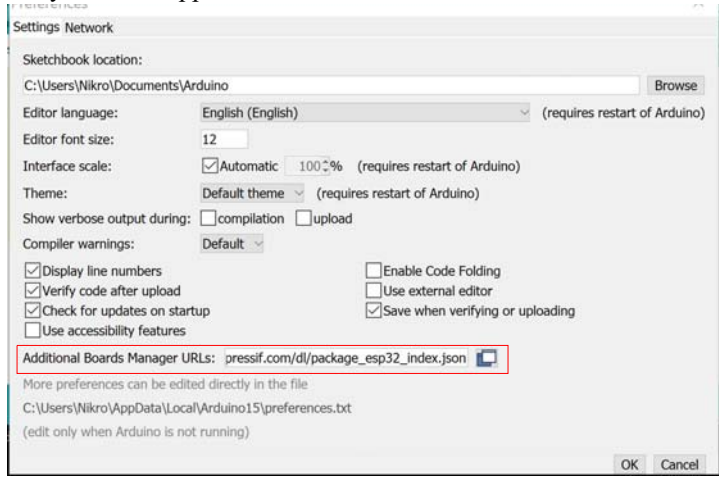
- Kabelen er ikke tilkoblet, eller ødelagt
- Feil port er valgt av programeditoren, som kan endres ved å velge: *Verktøy* og *Port* fra meny-linjen i editoren
- Feil type mikrokontroller-kort er valgt
  Endres ved å velge: *Verktøy* og *Kort* fra menylinjen, for så å velge rett kort, i vårt tilfelle *Arduino/Genuino UNO*.
- Manglende drivere, i så fall må driverne installeres manuelt
- Eller rett og slett at programmet har hengt seg opp. I så fall kan man løse problemet med å lukke og starte programmet på nytt.
- Enkelte andre programmer kan ta kontroll over USB-porten, og slik blokkere for overføring til Arduino-kortet. Ett slikt program er CURA (Ultimaker). I så fall lukkes programmet og man prøver å overføre programmet på nytt.

3.3 Installasjon av ekstra pakke for programmering av ESP32

I dette avsnittet skal vi se hvordan vi kan installere tillegget som gjør det mulig å bruke IDE til å programmere ESP32[12].

For dere som bruker MAC så må ekstra drivere lastes ned og installeres:
https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers
Dernest kan dere gå videre med følgende:

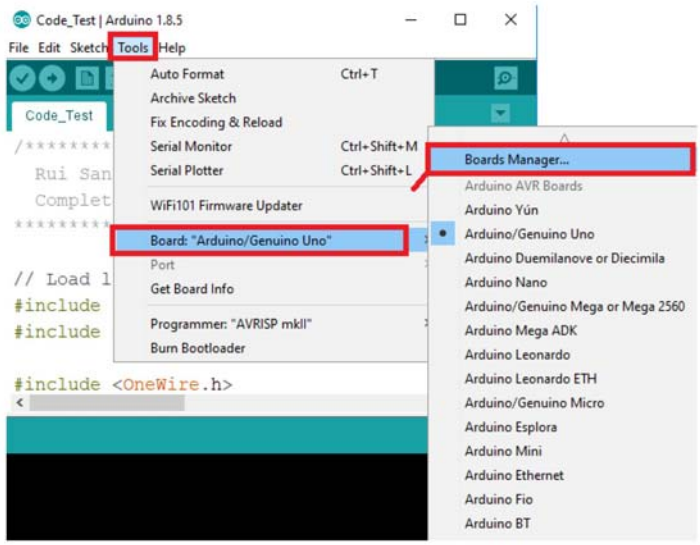
1. Gå til menylinjen på Arduino IDE og velg: *File —> Preferences.*
   Følgende meny kommer opp:



12. Beskrivelsen er delvis hentet fra Santos, Learn ESP32 with Arduino IDE (https://randomnerdtutorials.com/learn-esp32-with-arduino-ide/).

2. Legg inn følgende nettadresse https://dl.espressif.com/dl/package_esp32_index.json i feltet merket "*Additional Board Manager URLs*". Dersom det alt ligger noe der, gå til slutten av linjen skriv komma og legg inn den nye adressen bak de eksisterende.
   Klikk deretter på "OK" knappen.

3. Gå så til *Tools —> Boards —> Bords Manager* som vist på figuren under:



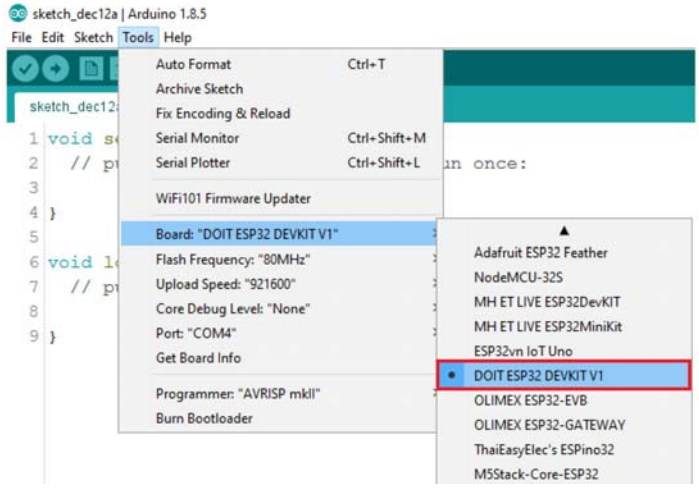
4. Når du åpner *Boards Manager* får du opp et nytt vindu, hvor du skriver *esp32* i søkevinduet, etter kort tid vil du få opp følgende:

   Dersom du ikke får opp et forslag til installasjon, så skyldes det sannsynligvis at du skrev inn feil adresse under *Preferences*, da *det* er adressen der *Boards Manager* leter etter tillegget.

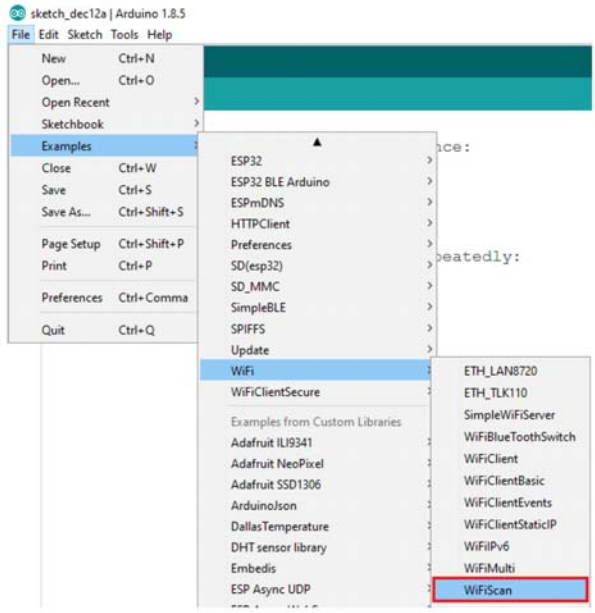
Vi skal nå teste at tillegget fungerer. Gjør følgende:

5. Åpne Arduino IDE (om du ikke alt har den åpen).

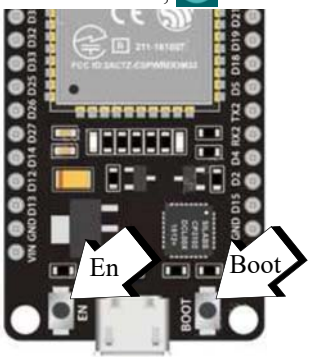
6. Gå til *Tools* på menylinjen og velg: *Tools —> Boards —> ESP32 Arduino —> DOIT ESP32 DEVKIT V1* fra lista til høyre som vist på figuren under (legg merke til at det i nyere versjoner finnes et nivå til: ESP32 Arduino). Du har nå gitt IDE'en beskjed om hvilket kort du har tenkt å bruke.



7. Gå til *File* på menylinja og velg: *File —> Examples —> Wi-Fi (under DOIT ESP32 DEVKIT V1) —> Wi-FiScan.* Dette er et program som scanner hvilke nettverk som finnes i nærheten.



8. Last opp programmet og velg *kompiler og overfør* til mikrokontrolleren,  :



9. Kompileringen tar ganske lang tid og når opplastingen skjer er det normalt at det dukker opp rød tekst i meldingsvinduet nederst.

   Programmet skal nå kjøre i mikrokontrolleren ESP32 og returnere informasjon om nettverk som finnes i nærheten. Disse skrives ut i monitoren.

   NB! Dersom opplastingen venter og ikke kommer videre ev. gir en feilmelding: TRYKK BOOT-knappen (til høyre på figuren over) og hold den nede til programmet begynner å lastes, dette gjelder vår 30-pins variant. Dette må i så fall gjøres ved hver opplasting. Noen ganger kan det også være nødvendig å trykke ENABLE-knappen for å starte programmet.

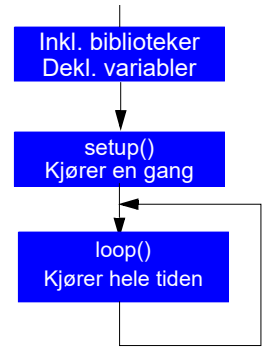
10. Åpne monitoren ved å trykke på symbolet  . Pass på at overføringshastigheten (Baud rate) for monitoren er satt til **115 200 baud**, som vist på figuren under.

11. Du skal nå se tilgjengelig trådløse nettverk. I vårt tilfelle hele 11 stykker.

Du er nå klar til å programmere i ESP32.

3.4 Programstruktur og bruk av funksjoner

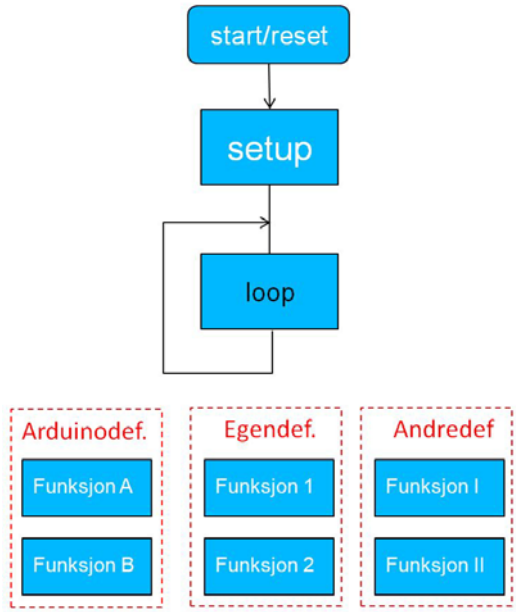
3.4.1 Programstruktur



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 deklarasjon** 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. Lokale variabler kan også uten problemer gjenbrukes i andre funksjoner.
- Det er også vanlig å skrive ***egne funksjoner*** som gjerne legges på slutten av programmet, utenfor og etter `void loop()`-funksjonen(), men kan i prinsippet legges hvor som helst utenfor `void setup()`- og `void loop()`-funksjonene.

```
#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
 }
```



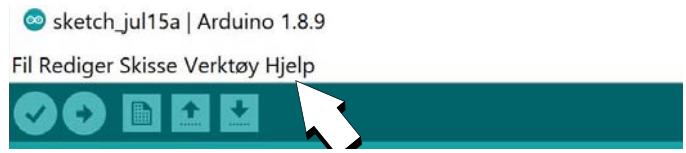
De to hovedfunksjonene i Arduino-programmene omsluttet av *klammeparenteser*. I `void setup()`-funksjonen initieres mikrokontrolleren og ev. sensorer som er tilkoblet, mens selve programmet legges under `void loop()`-funksjonen, som vist på figuren over.

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

I tillegg til disse to faste funksjonene så bruker vi funksjoner fra Arduino's eget bibliotek (Arduinodef. på figuren over), 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 3.5.14, side 49.

3.5 Viktige kommandoer

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



Referansemanualen kan også nås via *Hjelp* på menylinjen i programeditoren (figuren over).

3.5.1 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 setup-funksjonen med kommandoen: Serial.begin(9600); her er datahastigheten satt til 9 600 baud[13] (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 vanlig å bruke. Pass på at mottakerhastigheten for monitoren er satt til samme hastighet. Dette gjøres nederst til høyre i monitoren som vist på figuren under.

13. 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 følsomheten for støy.

Kommandoer for å skrive tilbake til PC – til monitoren i programeditoren:

Følgende kommandoer skriver en variabel eller en tekst tilbake til terminalvinduet (monitoren) i programeditoren.

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

Det er også mulig å kombinere tekst og variabler i samme printkommando:

```
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
```

3.5.2 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 ev. andre som skal lese og forstå programmet senere.

Ønsker man å kommentere bort flere linjer etter hverandre 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.

3.5.3 Bruk av variabler

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

> *Vi kan betrakte variabler som oppbevaringssteder ("skuffer") for tekst eller verdier 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.*

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 slutttidspunktet. Dette føres oss til den tredje viktige egenskapen:

Vi kan bruke variabler som lagringsplass for verdier over tid.

Deklarasjon av lokale og globale variabler

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 i variablene uavhengig av hvor de brukes.

Deklarering 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()-funksjonen:

```
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 bytes heltallsvariabel f (32 bit)
                   // uten fortegn
   boolean g;      // deklarasjon av en boolsk variabel g
                   // som kan ha verdiene 1 eller 0, "sant" eller "usant"
   // Her følger resten av programkoden i funksjonen loop()-
}
```

Dersom vi deklarerer variabler i begynnelsen av void loop()-funksjonen så vil de bli redeklarert for hver runde i loopen, hvilket betyr at de vil miste verdien for hver gang funksjonen utføres.

3.5.4 Strukturer

Mens et *array* er en variabel som inneholder et sett av *elementer av samme type*[14] (f.eks. *enten* int *eller* float osv.), så er en *structure* en variabel som kan inneholde et *sett av elementer av ulike typer* (f.eks. *både* int *og* float med flere). La oss like godt se på et eksempel på hvordan vi kan definere en struktur[15].

14.Et array kalles i noen sammenhenger også en *vektor.*
15.https://www.tutorialspoint.com/cprogramming/c_structures.htm

Definisjon av strukturer

I dette eksempelet skal vi definere en struktur som holder måledata fra sensorer *sensor_measure*, vi kaller denne typen struktur for *measurements*. I den forbindelse ønsker vi å lagre følgende informasjon om målingene[16]:

```
typedef struct measurements {
  char message[32];
  String source;
  int instance;
  float temperature;
  float humidity;
};
```

Hvor:

`typedef struct`	– Viser at dette er definisjon av en struktur
`measurements`	– Navnet vi har gitt til denne *typen* struktur
`char message[32];`	– En tekststreng med informasjon om målingen
`int instance;`	– Målenummer
`String source;`	– Kilden for målingen (f.eks. MAC-adresse til ESP32)
`float temperature;`	– Temperaturen målt i grader Celcius
`float humidity;`	– Relativ luftfuktighet målt i prosent

Nå har vi definert en type struktur som passer for vårt bruk. Nå skal vi bruke denne definisjonen til å *deklarere* våre spesielle målinger.

Deklarasjon av strukturer

Nå når vi har definert en type struktur som vi har kalt *Measurements*, så kan vi lage oss flere strukturer, dvs. samlinger av variabler, med ulike navn av typen Measurements:

```
Measurements packet1;
Measurements packet2;
```

Vi har valgt å lage variablene `packet1` og `packet2` av typen `Measurements`.

Tilordning av innhold til medlemmene av strukturen

Vi ønsker nå å legge inn informasjon i de ulike *medlemmene* av strukturen. Dette kan vi gjøre slik for et array av karakterer:

```
packet1.message[32] = "Maaling med BME280";
```

eller for en *int* eller *float* gjør vi slik:

```
packet1.temperature = 23.4;
```

16. Det finnes flere måter å definerer strukturer på som er like riktig, vi har valgt denne fordi den tydelig indikerer hva dette handler om: Definisjon av en type struktur (typedef struct { ... })

eller slik for *strengvariabler*:

```
packet1.source = String("Dette er en streng");
```

String-kommandoen kan også brukes til å konvertere et tall til en tekststreng. I dette tilfellet omformes et heltall (45) til et heksadesimalt tall (HEX) uttrykt som en "tekststreng", dvs. en streng med bokstaver og tall:

```
packet1.source = String(45, HEX);
```

Tilgang til innhold hos medlemmer av strukturer

Vi skal jo også kunne ta ut innholdet til ett eller flere medlemmer i en struktur og bruke det i ulike sammenhenger som f.eks. ved en utskrift. Dette kan vi gjøre slik:

```
Serial.print(packet.message);      // For et tekstarray
Serial.print(packet.instance);     // For et heltall
Serial.print(packet.source);       // For en streng
Serial.print(packet.temperature); // For en float
```

Vi kan også legge innholdet over i en enkel variabel slik:

```
float T = packet.temperature;
```

Ekstra slagkraftig blir bruken av strukturer når vi skal overføre et sett av variabler "pakket" i en struktur i argumentet i en funksjon.

3.5.5 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.

3.5.6 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()` holder tiden i millisekunder fra vi slo på strømmen eller resatte programmet.

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:

```
unsigned 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.

Vi definerer en variabel `unsigned long naaTidspunkt`; Grunnen til at vi definerer variabelen som type unsigned long er for å unngå at variabelen skal tildeles verdier som er utenfor maksimal størrelse til variabelen. Bruker vi en `int` vil denne nå grensen for område sitt 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()`.

3.5.7 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
```

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

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

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

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

I tillegg til de aritmetiske operasjonene har vi også tre logiske funksjoner

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

3.5.8 Adresser og pekere

Adresser:

I språket C og C++ har vi direkte tilgang til *adressen* i lageret hvor en variabel ligger. I noen tilfeller kan det være praktisk å kjenne adressen til en variabel. Dersom vi definerer en variabel, f.eks.:

```
float f;
```

... så kan vi finne adressen til variabelen som:

```
long adr_f = &f;
```

Ved å sette tegnet "`&`" foran variabelnavnet så får vi adressen der variabelen er lagret.

Pekere:

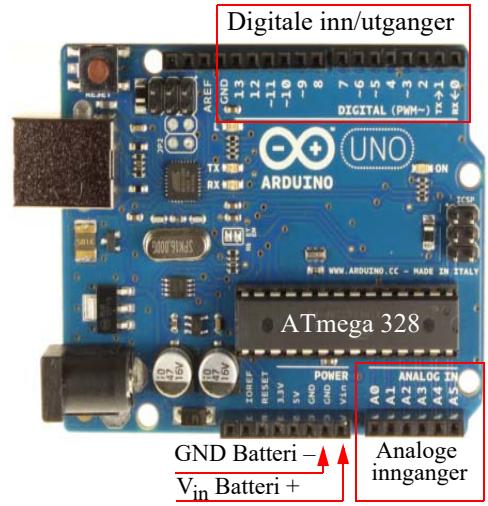
Vi kan også gjøre det motsatte. Vi kan definere en peker. En *peker* er definert som en adresse og angis med en stjerne "*". La oss deklarere følgende:

```
int resultat;      // Deklarerer en heltallsvariabel med navn resultat
int i = 6;         // i er et heltall som gis verdien 6
long *pek_i;       // *pek_i er en peker som kan peke på en adresse

*pek_i = &i;       // *pek_i tilordnes adressen til heltallet "i"
resultat = pek_i;  // pek_i angir innholdet av adressen *pek_i
                   // resultat inneholder verdien 6
```

Vi legger merke til at fjerner vi "*" foran pekeren så angir navnet verdien pekeren peker på.

3.5.9 Digitale porter

Definer digitale porter som inn- eller utganger:



En port er et 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). De digitale portene kan være innganger eller utganger. Hver port må derfor defineres som en inn- eller utgang. Dette gjøres gjerne i `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 gjennom 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 3,3 V 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 brukes for 0 og 1
void loop()
{
digitalWrite(8, HIGH);   // Setter port 8 høy (5V ev. 3,3 V 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).

Tilsvarende gjelder for ESP32, se avsnitt 2.1 og 2.2.

3.5.10 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); //Digitale verdien fra analog inngang 0
                        //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 *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 AD-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 – 5 V til en digital verdi fra 0 – 1023 (0000000000 – 1111111111). Dvs. AD-konverteren har 10 bits nøyaktighet. Tilsvarende kan ESP32 håndtere analoge spenninger fra 0 – 3,3 V og har AD-konvertere med en oppløsning på 12 bit.

For å regne om fra digital verdi (0 – 1023) til analog spenningsverdi (0 – 5V) benytter man følgende formel:

```
anatemp = 100 * digitemp * 5.0/1024;
```

`anatemp` er en float (desimaltall), mens `digitemp` er den digitale avleste verdien. Vi multiplisererer med 100 siden en vanlig brukt temperatursensor TMP36 da vil gi resultatet direkte i grader C.

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.

3.5.11 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
}
```

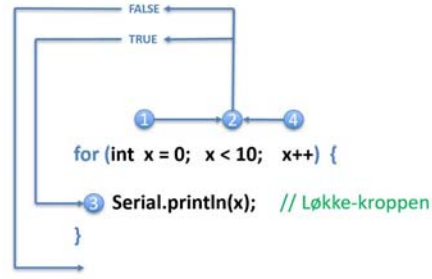
x er en heltallsvariabel (`int x`) som brukes som teller. Denne starter på verdien 0 (`int x = 0;`) økes med 1 (`x++`) for hver runde i loopen og stopper ikke før den når opp til 100 (`x < 100;`). De ulike utrykkene skilles med semikolon. Det som skal gjentas står innenfor klammeparentesene.

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[17].

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 { }.

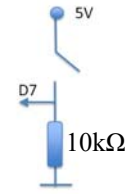
En while()-loop egner seg også for å hindre at en avlest bryter skal medføre et skred av avlesninger av bryteren.

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 å 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. Siden vi ønsker at handlingen kun skal utføres en gang, må vi huske på å nullstille variabelen `trykkBryterVerdi` før vi forlater `if()`-setningen.

3.5.12 `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:

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

```
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.

3.5.13 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.

```
6  void loop() {
7    // read the sensor:
8    int sensorReading = analogRead(A0);
9    // map the sensor range to a range of four options:
10   int range = map(sensorReading, sensorMin, sensorMax, 0, 3);
11
12   // do something different depending on the range value:
13   switch (range) {
14     case 0:    // your hand is on the sensor
15       Serial.println("dark");
16       break;
17     case 1:    // your hand is close to the sensor
18       Serial.println("dim");
19       break;
20     case 2:    // your hand is a few inches from the sensor
21       Serial.println("medium");
22       break;
23     case 3:    // your hand is nowhere near the sensor
24       Serial.println("bright");
25       break;
26   }
27   delay(1);        // delay in between reads for stability
28 }
```

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.

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 henholdsvis 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.

3.5.14 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 det grønne lyset i et trafikklys 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.

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.

```
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);
    }
 }
```

Her bruker vi variabelen `antallBlink` for å overføre antallet. Vi legger merke til at navnet på variabelen i kallet og variabelen i funksjonsdefinisjonen 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.

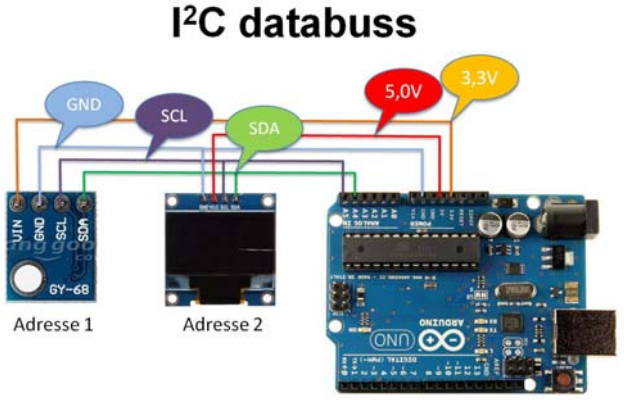
```
...
void loop()
  {
   // Det som skal gjentas hele tiden
   float volum;
   float radiusKule = 5.0;
   volum = volumKule(radiusKule);    // Kall funksjonen
   Serial.println(volum);
  }

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.

3.6 Bruk av I²C buss og biblioteker

I dette avsnittet skal se på hvordan vi kan bruke I²C-[18] buss til å overføre data fra sensorer utstyrt med denne typen kommunikasjonsgrensesnitt, som f.eks. trykkmåleren BMP180/BMP280 eller BME280 og displayet SSD1306. Til dette bruker vi som oftest spesiallagede biblioteker som må installeres dersom de ikke alt finnes blant bibliotekene som følger med Arduino editoren (IDE).



Den såkalte I²C-bussen består at to kommunikasjonslinjer, SDA og SCL, som gjør det mulig å sende data etter hverandre (på serieform) mellom ulike kretser. Siden hver krets som er koblet til bussen har sin unike adresse, kan man adressere meldingene til den komponenten man er interessert i å kommunisere med.

18. I²C står for Inter Integrated Circuit, også forkortelsen I2C og IIC brukes.

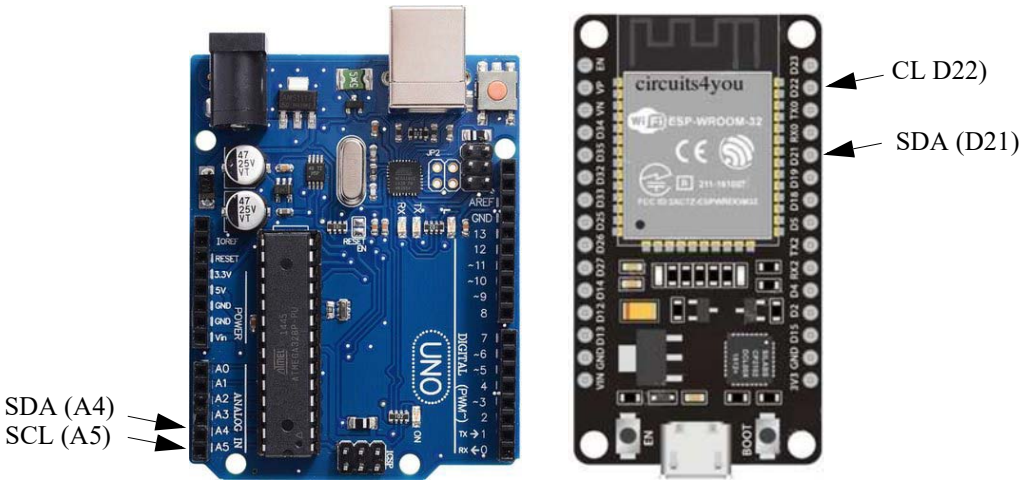
Standardbiblioteket for I^2C følger med Arduino programvaren, men krever at man inkluderer "header"-filen: "wire.h" øverst i koden ved å skrive:

```
#include <wire.h>
```

Dette medfører at programmet får tilgang til en del funksjoner knyttet til kommunikasjon via I^2C-bussen.

For mange spesiallagde kretser finnes det spesielle biblioteker med funksjoner som gjør dem lettere å bruke. Slike biblioteker må gjerne hentes ned fra leverandøren eller andre og installeres i programeditoren.

For Arduino UNO brukes A4 (SDA) og A5 (SCL) for å kommunisere med I^2C-bussen. For ESP32 brukes gjerne D21 (SDA) og D22 (SCL).



Arduino UNO **ESP32**

I neste avsnitt skal vi se hvordan vi kan laste ned og installere biblioteker generelt og deretter biblioteker for I^2C spesielt.

3.6.1 Installasjon av pakkede biblioteker generelt

Som eksempel bruker vi biblioteket som trengs for å lese data fra trykksensoren BMP180 som også inneholder en temperatursensor.

1. Last ned biblioteket i pakket form (*.zip). Denne pakken inneholder header-filer (*.h), biblioteksfunksjoner (*.cpp) og gjerne eksempler på bruk av sensoren. Er man heldig inneholder den også en beskrivelse av biblioteksfunksjonene. Man finner gjerne biblioteket på nett ved søke etter den aktuelle kretsen etterfulgt av ordet *library* om adressen ikke er kjent.

   Fra og med Arduino IDE versjon 1.5 kan man fra menylinjen velge:
   Sketch/Include library/Manage Libraries
   og skrive inn den aktuelle sensoren i søkefeltet. Man får da beskjed om biblioteket er installert eller forslag til hvor man kan hente det fra.

2. **Last ned biblioteket fra ekstern kilde:**
   Dersom biblioteksfilen for komponenten ikke finnes, kan man hente biblioteksfilene til BMP180 fra ulike steder, f.eks. fra SparkFun's hjemmeside:

   https://learn.sparkfun.com/tutorials/bmp180-barometric-pressure-sensor-hookup-/installing-the-arduino-library

   For BMP280 gjelder følgende adresse:

   https://github.com/adafruit/Adafruit_BMP280_Library

   og for BME280 gjelder følgende adresse:

   https://github.com/adafruit/Adafruit_BME280_Library

   Vi har også lagt dem ut under den blå hefteserien:

   https://www.ntnu.no/skolelab/bla-hefteserie

3. **Installer biblioteket:**
   Dernest installeres biblioteket i Arduino-programvaren ved å velge:
   *Sketch/Include Library/Add *.ZIP Library* for så å velge den nedlastede zip-pakkede filen.
   Biblioteksfilene blir dermed lagt under *Sketch/include library* og eksempler under: *File/Eksempler*

4. **Inkluder headerfilen (*.h) i programmet**
   Det aktuelle biblioteket inkluderes i programmet ved å velge:
   Sketch/Include Library/<aktuelle biblioteket>
   Dermed er header-filen på plass i programmet.
   Det vanligste er at man skriver den inn manuelt hvilket forutsetter at man kjenner navnet på h-filen.

3.6.2 Bibliotek for bruk av I^2C

Også I^2C-bussens bibliotek må lenkes inn i koden når den skal brukes. Bussen fungerer ved at en "master" styrer kommunikasjonen. Vanligvis er dette Arduino mikrokontrollerkortet som automatisk blir definert som master ved kommandoen `Wire.begin();` kommandoen i setup-funksjonen.

Biblioteket – Wire.h:

Biblioteket inneholder noen funksjoner som vi skal oppsummere her:

"Header file":

```
#include <Wire.h>
```

"Header" kommandoen knytter I^2C-biblioteket til programmet slik at kompilatoren vet hvilket bibliotek funksjonene skal hentes fra.

```
Wire.begin();
```

Denne funksjonen definerer Arduino mikrokontrolleren som master for kommunikasjonen, dvs. at all kommunikasjon med sensorer og andre periferikretser som er koblet til I^{2}C-bussen styres av Arduino'en. Siden Arduino'en er masteren, er det ikke nødvendig å definere en adresse (`Wire.begin(<adresse>);`) for denne. Ev. kan adressen være et tall mellom 0 – 127 som identifiserer masteren på bussen.

```
Wire.beginTransmission(<adresse>);
```

Med funksjonen ***`Wire.beginTransmission(<adresse>);`*** kan man starte en dataoverføring til en enhet på bussen med den spesifiserte adressen.

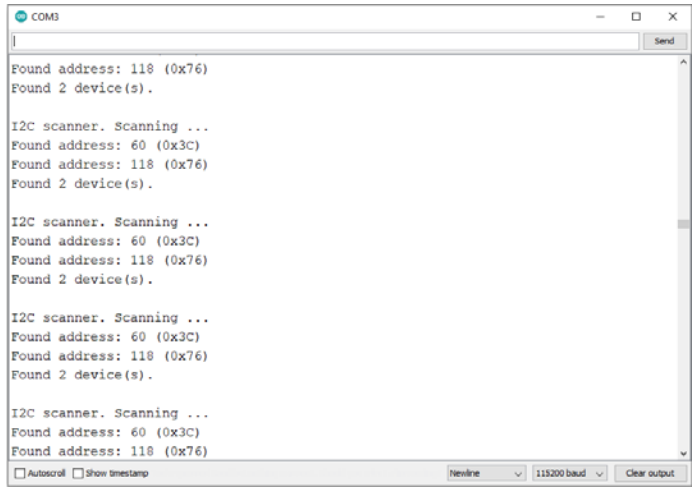
Vi skal senere se hvordan vi anvender dette i praksis.

For tilkobling til I^{2}C hos ESP32 ta gjerne en titt på følgende nettside: https://randomnerdtutorials.com/esp32-i2c-communication-arduino-ide/#1

Program for scanning av adresser på I^{2}C-buss

Dersom man er usikker på hva enhetens adresse er kan man koble enheten til bussen og benytte et programhjelpemiddel for scanning av adressene til de tilkoblete enhetene. Man kobler opp enheten og installerer scanningprogrammet: *i2C_scanning* i Arduino'en. Programmet kan kopieres fra denne siden: https://playground.arduino.cc/Main/I2cScanner/ Programmet vil lete etter adressene til de enhetene som er koblet på bussen og vise resultatet i monitoren som vist på figuren under.

Som vi ser av figuren kan flere enheter med ulike adresser være koblet til bussen, og programmet viser alle som det finner.



4 Oppkobling Internett via ESP32s Wi-Fi-enhet

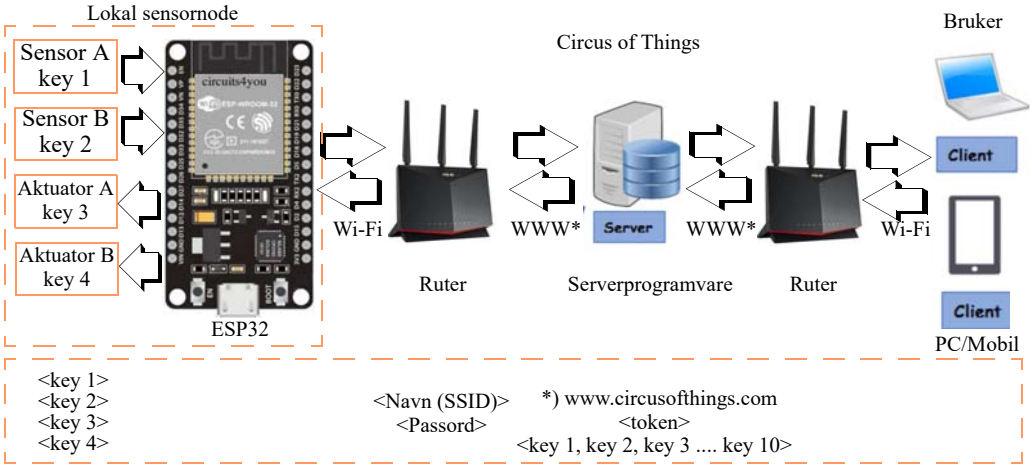
I dette kapittelet skal vi se på to måter vi kan bruke Wi-Fi-enheten til ESP32 på:

- Oppkobling mellom ESP32 og Internett via en lokal ruter og CoT (Circus of Things)
- Oppkobling direkte mellom flere ESP32-er – ESP-NOW

4.1 Circus of Things – ESP32 brukt som sensornode[19]

I dette avsnittet skal vi se på det som gjør ESP32 spesiell, nemlig Wi-Fi enheten. Kommunikasjonen med nettet er i vårt tilfelle så tett koblet til Circus of Things at dette konseptet vil bli en vesentlig del av denne beskrivelsen.

Circus of Things er en gratis tjeneste som er etablert og drives av ***Jaume Miralles***, som holder til på Mallorca i Spania. Tjenesten gjør det mulig å bygge opp software konsoller som viser innsamlede data fra sensorer og konsoller for fjernstyring av aktuatorer. Tjenesten tilbys både for PC/MAC og smarttelefon. Dessuten kan signalene offentliggjøres slik at andre kan få se målingene. Eneste begrensning er inntil videre at maksimalt antall signaler til hver bruker er satt til 10. Dette kan ev. økes ved å henvende seg til Jaume. Tjenesten er heller ikke særlig rask. Den egner seg derfor ikke til "real time" styring



Figuren over viser hvordan sensorer og aktuatorer hos den lokale sensornoden, overfører data via en sentral server og videre til brukeren (klienten). Hos serveren gis signalene knyttet til sensorene og aktuatorene hvert sitt *nøkkelnummer* ("key 1 – N") og hvert sitt *konsoll* (bryter, glidebryter, display). Konsollene organiseres i paneler som kan inneholde flere konsoller som ønskes relatert til hverandre (f.eks. temperatur og luftfuktighet i det aktuelle rommet vårt, sammen med brytere for å slå på lys og ventilasjon). Brukeren får når vedkommende blir registrert, en *brukeridentitet* ("token") som også knyttes til signalene som overføres.

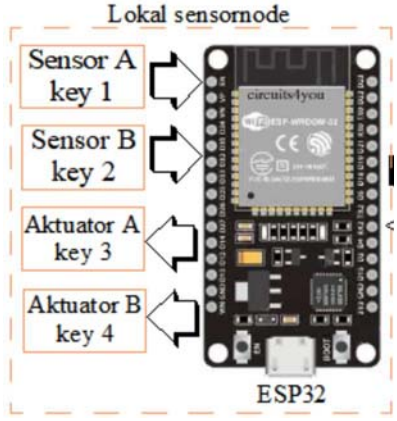
19. Dette kapittelet bygger på en presentasjon utarbeidet av Kåre-Benjamin H. Rørvik

Prosedyre for å etablere kontrollpaneler:

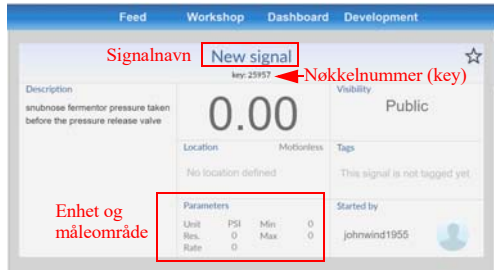
Vi skal nå gi en oversikt over hvordan vi kan lage paneler og konsoller på nettstedet Circus og Things (CoT): www.circusofthings.com. Senere skal vi se i detalj hvordan vi gjør dette.

- **Opprett bruker** med passord hos www.circusofthings.com.
  Man får da en signatur ("token"), en lang bokstav- og tallkode, som identifiserer brukeren.

- **Oppkobling av sensornode**: Koble opp ESP32 med sensorer og aktuatorer.



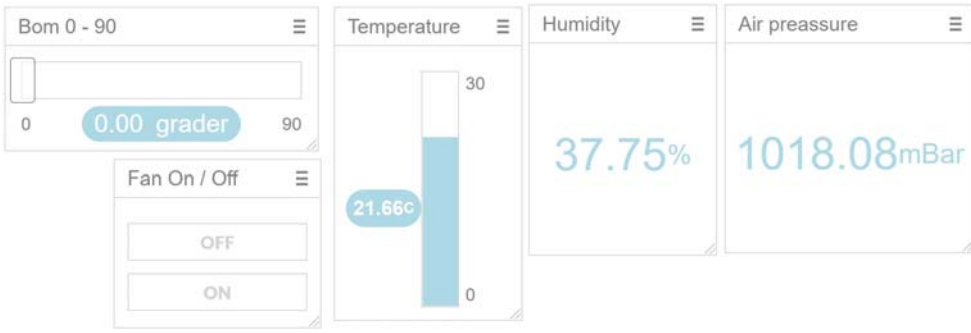
- **Gi navn til signaler:** I *"Workshop"* hos CoT spesifiseres signalnavn, enheter og måleområde som harmonerer med sensorene og aktuatorene som er koblet opp til den lokale ESP32.



Samtidig får man tildelt et nøkkelnummer ("key") knyttet til signalet. Se figur over.

- **Bygg opp panelet:** Panelet er sammenstillingen av brytere og displayer for å styre aktuatorene og monitorere sensordata fra sensornoden. Dette gjøres i fanen "Dashboard" i Circuit of Things. Bildet under viser monitorering av tre sensorverdier for temperatur, luftfuktighet og lufttrykk. Til venstre er to konsoller for styring av rele (ON/OFF) og bommen (Glidebryter 0 – 90°).

  Panelet:



- **Programmer ESP:** Så må man skrive programmet som leser av sensorene og sender data over til serveren slik at de kan vises på panelet. Likeså må programmet kunne motta styredata fra serveren slik at den kan styre servoer, lys og releer, o.l. fra konsollene.
  - ***Først henter man biblioteket*** som trengs for å kommunisere med Circus of Things (CoT). Dette finner vi under oversikt over biblioteker på nettsiden til CoT[20]. Disse er gjerne knyttet til ulik type hårdvare som brukes. Vi bruke ESP32 og henter inn følgende:

  #include <CircusESP32Lib.h>

  Biblioteket legges inn i starten av programfila.

  - ***Så defineres variablene*** som holder informasjonen som trengs for å opprette kommunikasjonen med hvert av signalene:

```
char ssid[] = "<Ruterens navn (SSID) legges inn her>";
char password[] =  "<Ruterens passord legges inn her>";
char server[] = "www.circusofthings.com"; // Nettsiden hvor CoT-serveren er
char token[] = "<token settes inn her>";

char order_key_relay[] = "27094"; // Nøkkel for å kommunisere med releet
char order_key_gate[] = "19499";  // Nøkkel for å kommunisere med bommen
char temperature_key[] = "21251"; // Nøkkel for å kom. med temperatursensoren
char humidity_key[] = "20852"; // Nøkkel for å kom. med luftfukt.sensoren
char pressure_key[] = "13531"; // Nøkkel for å kom. med lufttrykksensoren
```

  I dette tilfellet har vi definert 5 signaler som har fått hvert sitt nøkkelnummer. Nøkkel-numrene gjengitt her vil være individuelle og må skiftes ut med det som framkommer når hver enkelte bruker etablerer konsollene.

20. https://circusofthings.com/dev.jsp under Libraries under Targeted thing: ESP32 - CircusESP32Lib-1.0.0

- Dernest opprettes et objekt som vi velger å kalle `circusESP32()` og som er av typen `CircusESP32Lib`

```
CircusESP32Lib circusESP32(server,ssid,password);
```

Legg merke til at her inngår server-adressen, ruternavnet (SSID) og passordet til ruteren.

- ***Initialisering:*** I void setup()-funksjonen initialiseres funksjonene som står for kommunikasjonen med serveren ved CoT:

```
circusESP32.begin();        // Initialiser Circus of Things
```

- ***Les data mottatt fra panelet hos CoT:***

Så leser vi data som sendes over fra panelet og de ulike konsollene:

```
double dashboard_order_gate = circusESP32.read(order_key_gate,token);
```

Det er styredata for å heve og senke bommen et visst antall grader som er vist i eksempelet over. Verdien som overføres fra bryterkonsollet med nøkkelnummer: `order_key_gate` legges i variabelen: `dashboard_order_gate`. Vi legger også merke til at sammen med nøkkelnummeret til den spesifikke konsollen så sendes også brukeridentiteten ("token").

Tilsvarende gjentas for alle signaler som mottas av sensornoden.

- ***Skriv data til panelet hos CoT:***

Dernest sendes data over til panelet til de ulike konsollene (displayene):

```
circusESP32.write(temperature_key,temperature,token);
circusESP32.write(humidity_key,humidity,token);
circusESP32.write(pressure_key,pressure,token);
```

I eksempelet over oversendes måledata fra temperatur-, luftfuktighet- og lufttrykksensorene hos BME280 til panelet ved CoT. Sensordataene er lest inn i variablene `temperature, humidity` og `pressure`. Disse knyttes til nøkkelnumrene til de tre displayene `temperature_key, humidity_key` og `pressure_key.` Vi legger også merke til at brukeridentiteten ("token") knyttes til hver av signalene.

I tillegg må signalene behandles på vanlig måte, både ved avlesning av sensorene og pådrag til aktuatorene.

5 Oppgavesamling

Gjennom en rekke oppdrag skal vi gradvis bygge opp et lite system en litt avansert form for datalogger. Vi antar at dere har installert Arduino IDE og at dere har grunnleggende kjennskap til Arduino-programmering og bruk av ESP32.

5.1 Oppdrag 1 – Oppkobling og programmering av BMP280

Dette burde være kjent stoff.

> **Oppdrag 1:** *Koble opp ESP280 og ESP32, installer biblioteket til kretsen og vis lufttrykk og temperatur i monitoren.*

5.1.1 Kort omtale av BMP280

BMP280 er en kombinert temperatur- og lufttrykksensor. Sensoren er spesielt beregnet for mobile anvendelser med sitt lave strømforbruk. Det finnes to storebrødre BME280 og BME680 som i tillegg inneholder henholdsvis en sensor for måling av luftfuktighet og en sensor for måling av gass deriblant CO_2.

Her er noen nøkkeldata og BMP280[21]:

- *Spenning og energiforbruk:*
  Supplyspenning: 1,7 – 3,6 V
  Strømforbruk standby: 0,1 µA
  Strømforbruk med måling 1 x pr. sek.: 3,6 µA (H, P, T)
- *Grensesnitt:*
  I^2C eller SPI
- *Lufttrykksensor:*
  Måleområde trykksensor: 300 – 1100 hPa
  Relativ nøyaktighet: ± 0,12hPa
  Absolutt nøyaktighet: ± 1hPa (0 – 65°C)
- *Temperatursensor:*
  Måleområde: -40 – +65°C
  Nøyaktighet: ± 1°C (0 – 65°C)

For mer informasjon se avsnitt Vedlegg A, side 132.

21. Informasjonen er hentet fra: https://www.bosch-sensortec.com/products/environmental-sensors/pressure-sensors/bmp280/

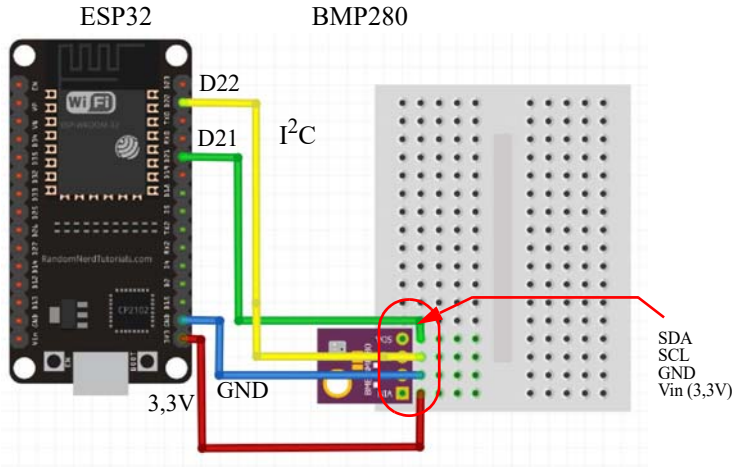
5.1.2 Oppkobling

Figuren under viser pinningen til to varianter av BMP280. BME280 som inkluderer måling av luftfuktighet, har også samme pinning:

Topp Bunn Topp Bunn

Legg merke til at det kan se ut som om rekkefølgen av pinnene er speilet i forhold til hverandre. Det er de derimot ikke dersom vi holder fast på at trykksensoren er på oversiden (Topp) av kretsen. Vi legger imidlertid merke til at teksten er plassert henholdsvis på undersiden og oversiden.

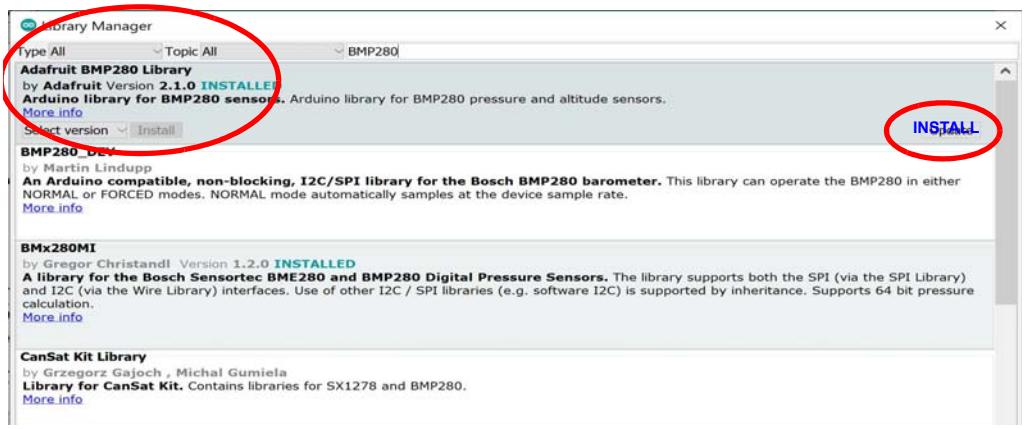
Figuren under viser oppkoblingen av BMP280.



5.1.3 Programmet

For å løse dette oppdraget må vi installere biblioteket om det ikke alt er installert. Biblioteket gjør det enkelt å kommunisere med BMP280. Vi velger å hente ned biblioteket fra Adafruit:

1. **Installer biblioteket**
   For å installere biblioteket til BMP280 går vi til menyen *Sketch/Include Library/Manage Library* og skriver BMP280. Ganske snart dukker det opp flere alternativer. Velg det øverste, Adafruit BMP280 som vist på figuren under:



   Velg **INSTALL** og biblioteket installeres. Lukk vinduet. Vi har nå de funksjonene vi trenger for å kommunisere med BMP280.

2. **Inkluder biblioteker** ved hjelp av følgende:

```
#include <SPI.h>              // Bibliotek for kommunikasjon på serie linjer
#include <Wire.h>             // Bibliotek for kommunikasjon via I2C
#include <Adafruit_BMP280.h>// Bibliotek for kommunikasjon til BMP280
```

   I tillegg må vi ha med et par andre biblioteker for seriekommunikasjon. Kommandoen plasseres i toppen av programmet.

3. **Definer et objekt:**
   Vi definerer objektet `bmp280` av typen `Adafruit_BMP280` slik:

```
Adafruit_BMP280 bmp280;
```

   Dette kan godt stå under inkluderingen av bibliotekene og før setup()-funksjonen.

4. **Initialiser biblioteket:**
   I initialiseringen av biblioteket til BMP280 inkludere adressen til I^{2}C-bussen:

```
bme280.begin(0x76);
```

   Initialiseringen skal plasseres i setup()-funksjonen. Vi legger også merke til at I^{2}C-buss adressen er 0x76 (hex). Ikke glem å initialiser seriekommunikasjonen til monitoren. Det er greit å bruke 115200 baud for denne kommunikasjonen.

   Om I^{2}C-adressen er ukjent kan man laste opp et program som scanner alle adresser som er koblet til I^{2}C til ESP32. Dette er beskrevet i avsnitt 3.6.2, side 53.

5. **Deklarer globale variabler:**
   Vi synes det er greit å deklarere variablene:

```
float temperature;
float pressure;
```

6. **Avlesing av dataene**
   Følgende kommandoer leser av sensorene og lagrer dem i en variabel av typen float:
   ```
   temperature = bme280.readTemperature();
   pressure = bme280.readPressure()/100.0F;
   ```
   F'en plassert etter 100.0 sørger for at divisjonen ender opp med et desimaltall (float).

7. **Skriv til monitoren**
   Vi vil at verdiene skal skrives til monitoren og lager programmet slik at utskriften blir på formen:
   ```
   -----------------------
   Temperatur: 20.42 *C
   Trykk:      992.07 hPa
   -----------------------
   ```
   Tips: Bruk `Serial.print( );` og `Serial.println();`

8. Egendefinerte utskriftsfunksjon

Selv om det kanskje virker overdrevent vil vi at dere skal lage to egendefinert funksjoner med navnene:

```
void readSensorValuesBMP280()
 {
   // Her skrives innholdet i funksjonen
 }
```

og

```
void sensorValuesToMonitor()
 {
   // Her skrives innholdet i funksjonen
 }
```

Den ene leser av de to sensorverdiene og den andre skriver dem ut til monitoren som beskrevet foran.

Når dere ønsker å skrive ut verdiene kaller dere bare opp funksjonene i void loop() som henholdsvis leser og skriver ut verdiene. Kallene av funksjonene ser slik ut:

```
readSensorValuesBMP280();
sensorValuesToMonitor();
```

Kallene skal stå i veid loop()-funksjonen, mens funksjonene defineres utenfor loop()-funksjonen. Gjerne nedenfor.

9. **Test ut at programmet**
   ... fungerer som ønsket.

10. *Lagre programmet*

Husk å lagre programmet før du går over til neste oppdrag. Husk og bruk norsk tegnsett – f.eks. *Oppdrag-1*.

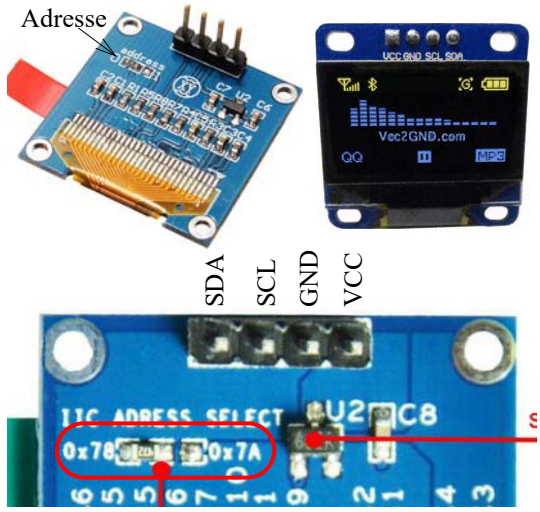
5.2 Oppdrag 2 – Oppkobling og bruk av OLED display

Vi skal nå ta i bruk en ny komponent, et lite display.

Oppdrag 2: *Koble opp OLED displayet og vis lufttrykk og temperatur på displayet.*

5.2.1 Kort om det grafiske OLED displayet SSD1306

Displayet er relativt lite og billig med 128 x 64 piksler og kan typisk vise tre – fem linjer med kort tekst, avhengig av skriftstørrelsen. Ved å gå ned på fontstørrelsen kan det vises opp til 5 linjer. Det egner seg godt for å skrive ut måleverdier og kan arbeide med fra 3 – 5 V og programmeres via I^2C-buss og et bibliotek som hentes fra nett. Det leveres vanligvis med en stiftlist med fire kontakter Vcc (5V), GND, SDA (seriell data), SCL (seriell klokke). Og egner seg godt for å kobles til I^2C-bussen til f.eks. Arduino UNO eller ESP32. Default adresse er 0x3C (hex), men kan omprogrammeres til 0x3D ved å flytte strapen mot høyre (se bilde til høyre). En legger merke til at merkingen av kretsen synes å antyde at de to adressene er 0x78 eller 0x7A. For mer informasjon se avsnitt 6.3, side 112.

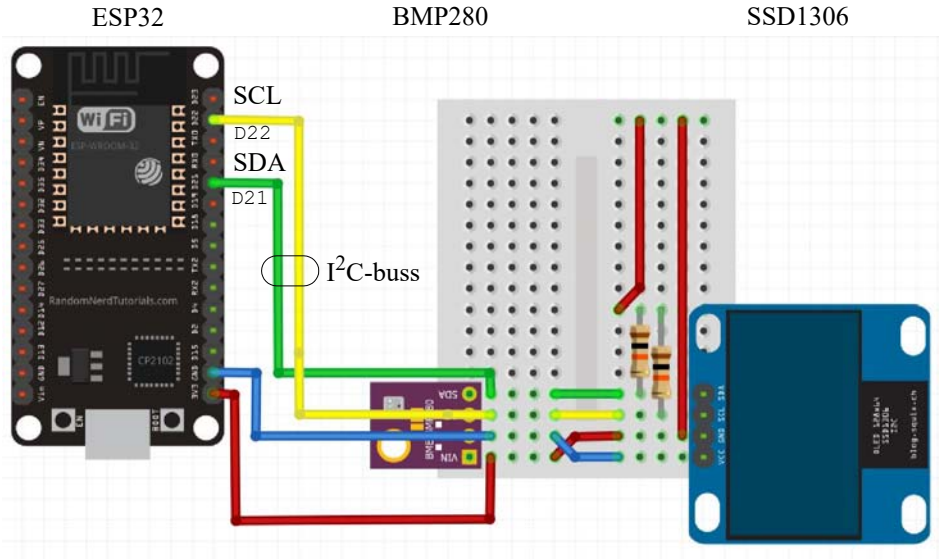


5.2.2 Oppkobling

1. Koble opp OLED displayet

Så langt har vi brukt monitoren på PC'en, nå ønsker vi å ta i bruk et lite OLED display av typen SSD1306. Koble ESP32 fra PC'en før du kobler opp displayet.

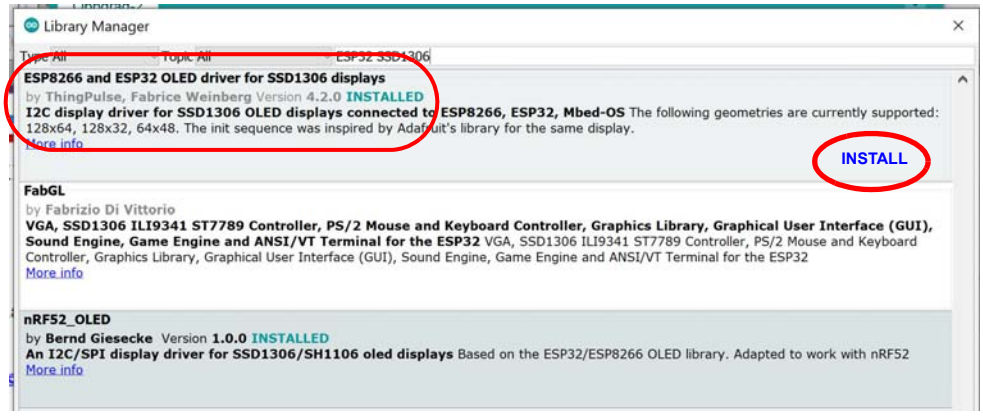
Montere displayet som vist på figuren under. Kommunikasjonen mellom micro-kontrolleren og displayet skjer via en I^{2}C-buss. Vi legger merke til at de to komponentene bruker samme ledninger for å kommunisere med micro-kontrolleren. De er begge koblet til I^{2}C-bussen som består av linjene *Serial clock* (SCL) og *Serial data* (SDA).



5.2.3 Programmet

1. Installer biblioteket

For å kunne kommunisere med displayet (SSD1306), må vi installere ett bibliotek. Vi bruker som vanlig *Library Manager* (*Sketch/Include Library/Manager Libraris*) og skriver *Esp32 SSD1306* i søkefeltet og får opp følgende:



Vi installerer det øverste.

2. ***Inkluder biblioteker i programmet***
Øverst i programmet inkluderer man biblioteket for SSD1306. Dermed får programmet adgang til alle funksjonene som trengs for å skrive til displayet.

```
// Inkludering av biblioteker
#include "SSD1306Wire.h"          // Inkluder bibliotek for å skrive til display
```

Disse legges inn helt i starten av programmet før deklarasjon av variabler og setup().

3. Opprett et objekt knyttet til displayet
Dernest må vi opprette et objekt av typen SSD1306Wire som vi plasserer før setup().

```
SSD1306Wire display(0x3c, SDA, SCL);  // Deklarasjon av enheten "display" av typen
                                      SSD1306Wire
```

4. **Initialiser displayet**
Displayet initialiseres i slik at det skal være klart til å motta tekst:

```
display.init(); // Initialiser displayet
```

Initialiseringskommandoen plasseres i setup()-funksjonen.

Det kan også være lurt å plassere et par andre kommandoer i setup()-funksjonen. Med denne kommandoen kan en velge fra hvilken kant en ønsker å se displayet:

```
display.flipScreenVertically(); // Snu displayet opp-ned, tilpass plassering
```

Med denne velger vi å skrive teksten fra venstre mot høyre:

```
display.setTextAlignment(TEXT_ALIGN_LEFT); // Skriv fra venstre side
```

5. **Posisjoner markør, tøm displayet og skriv ut tekst og variabler.**
Før vi begynner å skrive må vi tømme displayet og bestemme størrelsen på teksten. Disse kommandoene kan skrives inn i programmet der vi ønsker at utskriften skal gjøres, eller i setup()-funksjonen.

Slett informasjon på display:

```
display.clear();
```

Standard font og størrelse 10. Ønsker vi større tekst kan vi alternativt skrive 16 og 24 med økende størrelse på teksten:

```
display.setFont(ArialMT_Plain_10);
```

Posisjonerer markøren og skriver ut tekst og variabel i samme linje temperatur

```
display.drawString(0, 0, "tekst" + String(variabel) + "tekst");
display.drawString(0, 12, "tekst" + String(variabel) + "tekst");
```

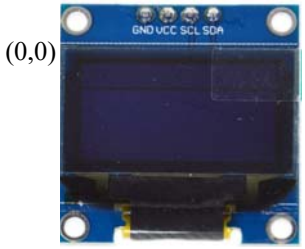
Bytt ut tekst og variabler slik at du får skrevet det du ønsker. Legg spesielt merke til forflytningen til neste linje ved å skrive 0, 12 i starten av argumentet.

6. **Send teksten til displayet**
Når alt er klart, sendes informasjonen til displayet med kommandoen:

```
display.display();
```

7. Om å orientere seg på skjermen

På figuren under ser vi hvordan vi kan posisjonere markøren rundt om på skjermen:



Koordinater for plassering av markør

Legg merke til at med skriftstørrelse 10 så er det greit å flytte markøren 12 bildepunkter ned for hvert linjeskift. Merk at koordinaten til markøren tar utgangspunkt øverste venstre hjørnet i karakterruta (origo).

8. Skriv programmet og test ut

Skriv ut temperatur og trykk til displayet og sjekk at det blir som forventet.

9. Lagre programmet

Husk å lager programmet før du går over til neste oppgave. Husk og bruk norsk tegnsett – f.eks. *Oppdrag-2*.

5.3 Oppdrag 3 – Omregning fra lufttrykk til høyde

I dette oppdraget skal vi regne om endring i lufttrykk til endring i høyde.

Oppdrag 3: *Bruk lufttrykket og bestemt relativ høyde med hjelp av omregningsformelen.*

La oss først se litt på sammenhengen mellom lufttrykk og høyde over havet.

5.3.1 Omregning fra trykk til høyde

Omregning fra trykk til høyde

Lufttrykket vil kunne endre seg med høyden over havet. Trykkmåleren er så følsom at vi kan se endringer selv om vi bare løfter den opp 1 meter.

Sammenhengen mellom trykk, temperatur og høyde kan uttrykkes med formelen[22]:

$$h = \frac{T_1}{a}\left(\left(\frac{P}{p_1}\right)^{-\frac{aR}{g_0}} - 1\right) + h_1 \tag{5.1}$$

22. Se lign. (6.2), side 103 for nærmere omtale.

Hvor:

H	Beregnet høyde i meter
P_1	Trykk i mBar ved referansehøyden H_1
H_1	Referansehøyden i meter
T_1	Temperatur ved referansehøyden H_1 i K(elvin)
P	Målt trykk i mBar
a	Temperaturgradient, foreslått verdi -0,0065 K/m
g_0	Tyngdeakselerasjonen 9,81 m/s^2
R	Den spesifikke gasskonstant 287,06 J/kg K

Denne omregningen går under betegnelsen Den *hypsoemtriske* omregningsformelen[23].

5.3.2 Oppkobling

Det trengs ingen ekstra oppkobling for å løse dette oppdraget.

5.3.3 Programmet

1. Lag en funksjon for omregning fra lufttrykk til høyde

Vi foreslår at du lager en funksjon som regner om fra lufttrykk i henhold til den formelen foran. Lufttrykket overføres via argumentet og funksjonen returnerer den beregnede høyden i meter. Finn et navn som er betegnende for hva funksjonen gjør, f.eks.:

```
float airPressureToHight(float P)
  {
    // Skriv inn koden som gjør omregningen
    return H;
  }
```

2. Skriv inn variablene som trengs i formelen

Før vi legger inn selve formelen kan det være greit å deklarere alle nødvendige variabler som vi trenger. Ikke alle er variabler, men vi velger her å definere dem som det.

// Deklarerer variabler for høydeberegning

```
float H;            // Beregnet høyde over havet
float H1 = 127;     // Referansehøyde i meter
float P1 = 992.80;  // Referansetrykk i mbar ved referansehøyde
float T1 = 17.2;    // Temperatur i grader C, husk at formelen krever grader Kelvin
float a  = 0.0065;  // Temperaturgradient i K/m
float R  = 287.06;  // Den spesifikke gasskonstanten i J/kg K
float g0 = 9.81;    // Tyngdeakselerasjonen i m/s2
```

23.https://keisan.casio.com/has10/SpecExec.cgi?path=06000000.Science%252F02100100.Earth%2520science%252F12000300.Altitude%2520from%2520atmospheric%2520pressure%252Fdefault.xml&charset=utf-8

Vi legger spesielt merke til *referansetrykk (P1)* og *referansehøyde (H1)*. Disse skal vi senere bruke til å kalibrere målingene slik at vi kan finne absoluttverdien for høyden. Verdiene som ligger der er verdier fra da programmet ble laget. Selv om vi ikke har kalibrert målingene så kan vi fortsatt gjøre relative målinger.

Det er også lagt inn en temperatur i grader. Dette er *ute-temperaturen.* Det er derfor ikke alltid meningsfullt å bruke den temperaturen vi måler med instrument vårt dersom vi sitter innendørs.

Vi kommer tilbake til dette under kalibreringen.

3. Utfør omregningen og skriv til display

Sett opp ligning (5.1), side 66 med de definerte variablene og verdiene.

```
// Beregning av høyden
  H = ........ ;
```

Det er læring i å skrive det matematiske uttrykket på en form som passer programspråket. Under har vi vist hvordan de matematiske operasjonene uttrykkes i Arduino C. Parenteser brukes på vanlig måte. "." brukes som desimalpunkt.

Aritmetiske funksjoner:

+	Addisjon
-	Subtraksjon
*	Multiplikasjon
/	Divisjon
pow(g,e);	Opphøye et grunntall (g) i en eksponent (e)
log(x);	Naturlig logaritme, definer argument og resultat som *double* (brukes ikke her)

4. Skriv programbiten som regner om og skriver resultatet til displayet

Skriv ut høyden i "meter" til displayet på linje to.

5. Test programmet og se at det fungerer som ønsket.

Du kan nå løfte kretsen opp så langt kabelen rekker og se at høyden endrer seg. Du vil også se at desimalene etter komma "flagrer" litt. Naturlig nok vil støy i trykkmålingen også gi seg utslag i variasjoner hos beregnet høyde.

6. Lagre programmet

Husk å lagre programmet før du går over til neste oppgave. Bruk norsk tegnsett – f.eks. *Oppdrag-3*.

5.4 Oppdrag 4 – Midling av målingen av lufttrykket

Støy er ofte et problem når gjør målinger. En måte å fjerne *tilfeldig* støy hos målingene er å utføre mange målinger for deretter å finne middelverdien til måleserien.

Oppdrag 4: *Undersøk variasjonen i trykk og høydemålingen. Noter ytterpunktene.*

Utfør en midling av 100 målinger og studer hvordan det endrer variasjonsbredden. Noter resultatet.

Vi foreslå at det gjøres en sammenligning før og etter midlingen for å se om det er oppnådd noen forbedring.

5.4.1 Oppkobling

Det trengs ingen ekstra oppkobling for å gjennomføre dette oppdraget.

5.4.2 Programmet

Det skal gjennomføres en midling av målingene gjort av lufttrykket.

1. **Beregn gjennomsnittet for 100 målinger**

   Vi foreslår at midlingen gjøres i funksjonen der avlesningen av lufttrykket foretas.

   Lag en for-loop og utfor en midling av 100 målinger. ESP32 er så rask at du ikke vil merke forskjellen i tiden det tar.

   Syntaksen for en for-loop er som følger:

   ```
   for(int i=0; i<100; i++)
    {
      // Gjør ny måling og akkumuler.
    }
   ```

   Husk å del resultatet på antallet målinger etter avsluttet for-loop.

2. **Skriv resultatet ut på displayet**

5.4.3 Måleoppdrag

Utfør følgende måleoppdrag:

1. **Sett antall målinger til 1,** dvs. ingen midling.
   Les av verdier i lufttrykket over en tidsperiode og noter ytterpunktene for målingene

   Avlest laveste verdi:_____

   Avlest høyeste verd:_____

   Forskjell mellom høyeste og laveste verdi:_____

2. **Sett antall målinger til 100**, dvs midling over 100 målinger.
   Les av verdier i lufttrykket over en tidsperiode og noter ytterpunktene for målingene

   Avlest laveste verdi:_____

   Avlest høyeste verd:_____

   Forskjell mellom høyeste og laveste verdi:_____

3. **Kommenter forskjellen med og uten midling:**

5.5 Oppdrag 5 – Gjør relative målinger av høyde

Vi skal nå teste ut høydemåleren.

> **Oppdrag 5:** *Utfør måleoppdrag og mål relativ høydeforskjell på to steder i nærområdet, f.eks. mellom to eller flere etasjer i skolebygget.*

Dersom høydeforskjellen kan fastslås med fysisk måling hadde det vært fint.

Å drasse med seg hele PC'en for å foreta målinger kan være utfordrende. Dersom dere har en Power bank så kan den brukes som en erstatning.

Vurder resultatet: _____

5.6 Oppdrag 6 – Finn absolutt høyde over havet

Måling av relativ høyde kan være vel og bra, men hva om vi ønsker å finne ut hvor høyt vi faktisk befinner oss over havet. For å fastslå det må vi kalibrerer høydemåleren vår.

> **Oppdrag 6:** *Finn den nærmeste lokale metrologiske målestasjon og les av en oppdatert verdi av lufttrykket sammen med målestasjonens høyde over havet. Legg verdiene inn i programmet slik at du får en kalibrert høyde over havet på stedet du befinner deg. Sjekk med kartdata hvor godt måleverdien stemmer med virkeligheten.*

La oss se hvordan vi kan bruke en offisiell målestasjon for kalibrering.

5.6.1 Kalibrering av høydemåleren

En høydemåler basert på måling av lufttrykk kan kalibreres på flere ulike måter:

1. Ved at man kjenner lufttrykket og høyden over havet til stedet der man starter målingen. I så fall kan man benytte lign. (5.1), side 66 og skrive inn den kjente lufttrykket (*P1*) og høyden (*H1*). Metoden er aktuell når man skal sende opp en sonde eller gå en tur i terrenget, og gir absolutt høyde over havet.
2. Ved at man nullstiller høydemåleren på bakkenivå, eller fra det stedet som man ønsker å måle høyden relativt til. Denne måten vil gi en relativ høydemåling i forhold til starthøyden for måleserien.
3. Ved at man finner en offisiell målestasjon i nærheten, for eksempel en flyplass eller en havn som viser en oppdatert verdi av lufttrykket. Slike målestasjoner oppgir også hvor høyt de ligger over havet, ev. at de referer målingene sine til havnivået[24]. Ulempen med flere slike

24. Undersøkelser har vist at de normalt refererer til den høyde stasjonen ligger på.

målestasjoner er at de ikke viser kontinuerlige målinger, men oppdaterer målingene for eksempel hver time. Dessuten kan de være et stykke unna stedet der man befinner seg. En slik kalibrering burde imidlertid gi en absolutt høyde over havet med rimelig god nøyaktighet.

Uansett hvilken metode som benyttes så må man foreta relativt hyppige kalibreringer siden luft-trykket kan endre seg ganske raskt.

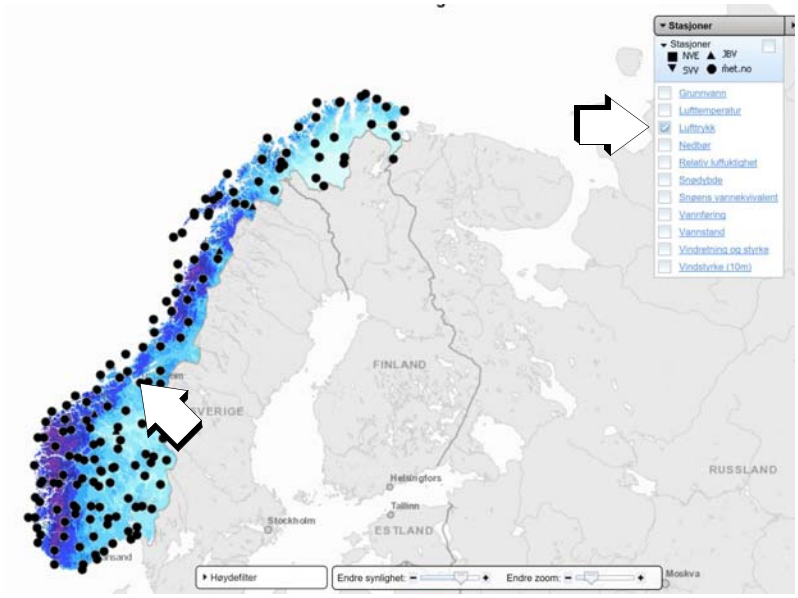
Her velger vi å bruke metode 3.

5.6.2 Valg av målestasjon og bruken av den

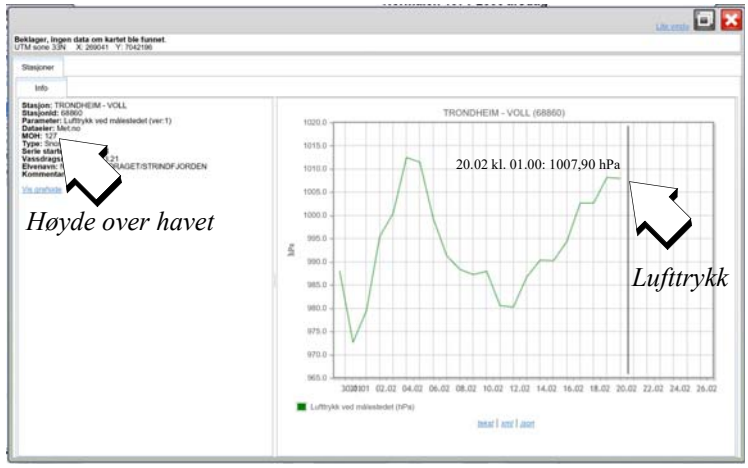
Gå til følgende nettadresse: http://www.xgeo.no/index.html?p=klima

Dette er en åpen portal på Internett, som viser daglig oppdaterte kart over snø-, vær- og vann-forhold og klima i Norge – og mye mer.

Velg “Stasjoner” og kryss av for “Lufttrykk”. Velg en målestasjon nær der du befinner deg og som registrerer lufttrykk. Vi velger *Voll* forsøksgård i Trondheim siden den er nærmest oss.



Etter å ha valgt målestasjon får man opp en graf som viser lufttrykket de siste 30 døgn. Målingene oppdateres en gang i timen så avvikene burde være til å leve med.



1. **Les av siste målte lufttrykk og høyde over havet hos referansestasjonen** og legg inn i programmet (xxx og yyyy.yy):

```
float H;            // Beregnet høyde over havet
float H1 = xxx;     // Referansehøyde i meter
float P1 = yyyy.yy; // Referansetrykk i mbar ved referansehøyde
```

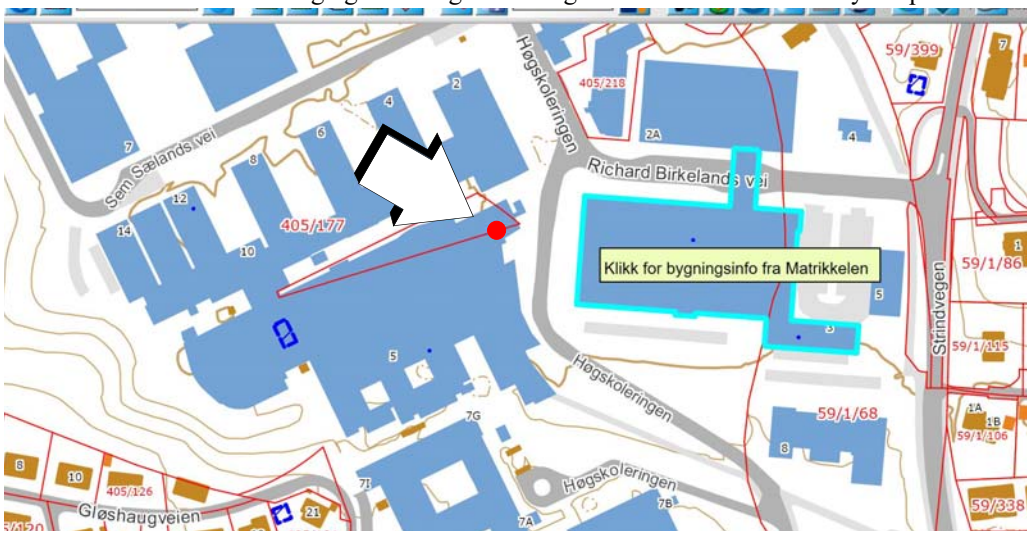
```
float T1 = 17.2;     // Utetemperatur i grader C
float a  = 0.0065;   // Temperaturgradient i K/m
float R  = 287.06;   // Den spesifikke gasskonstanten i J/kg K
float g0 = 9.81;     // Tyngdeakselerasjonen i m/s2
```

2. **Kompiler og last opp programmet**
   Sjekk om det fungerer som forventet

3. **Undersøk om høyden stemmer med virkeligheten**

   Gjør målinger av høyden over havet der du sitter. Ev. sammenlign målinger utført av andre som befinner seg på samme sted og se på avvik. Hvordan kan vi ev. forklare avvik i absolutte målinger?

   Kartet under er et utsnitt fra Gløshaugen ved NTNU (Trondheim). Finn tilsvarende kart for stedet der du befinner deg og se hvor godt målingene dine stemmer med høyden på kartet.



   Kartet er hentet fra:
   https://kart5.nois.no/trondheim/Content/Main.asp?layout=trondheim&time=1550437879&vwr=asv

   Hvilke avvik registrerer du i forhold til høyden fra kartdata og den du måler etter kalibrering? Hva tror du ev. avvikene skyldes?

5.7 Oppdrag 7 – Koble opp og test ut GPS-mottakeren

Oppdrag 7: *Koble opp GPS mottakeren, les ut stedskoordinatene og høyden og skriv ut i monitoren. Skriv også dataene ut på OLED-displayet. Sammenlign GPS høydedata og høyden beregnet av den barometriske trykkmåleren og GPS. Diskuter ev. avvik.*
Tips: *Temperaturmålingen må sannsynligvis tas ut siden det ikke er plass til mer enn 5 linjer på OLED-displayet.*

Installer Google Earth og kopier GPS-data inn i kommandolinjen og sjekk at avleste koordinater er korrekte.

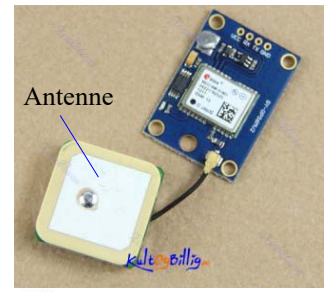
5.7.1 Bakgrunn - GPS

GPS eller *Global Positioning System* består av 24 satellitter som kretser omkring jorda med en omløpstid på 11 t 58 min. i en høyde av ca. 20200 km over jordoverflata. Normalt vil dette antallet være tilstrekkelig for, til enhver tid, å ha fri sikt til 8 – 10 satellitter i åpent terreng. Hver satellitt sender ut et kodet tidssignal som mottas av mottakerne. I tillegg til å inneholde informasjon om nøyaktig tid, inneholder signalet tidspunkt for utsendelse og en lang kode som mottakerne bruker for nøyaktig å bestemme tidspunktet for mottatt signal. På denne måten kan mottakersystemet bestemme tiden hvert av signalene bruker fra hver av satellittene og til mottakeren. De målte tidsforsinkelsene brukes så til å beregne posisjonen til mottakeren [1].

En ***GPS-modul*** er et sett av GPS mottakere som gjør det mulig å følge flere GPS-satellitter samtidig. Seks er ikke uvanlig. 4 er et minimum for å kunne beregne koordinater pluss høyde. Normalt vil GPS være mere nøyaktig på lengde- og breddegrad enn i høyden. For en mer nøyaktig høydemåling vil en kalibrert barometer være bedre.

5.7.2 Teknisk - spesifikasjon GY-NEO-6MV2 Flight (GPS-modul)

Dette er en vanlig brukt GPS-modul, senere er det imidlertid kommet nye varianter. Den kan bl.a. kjøpes fra BangGood for ca. kr. 75,-[25]. Den består av to separate enheter forbundet med en koaksialkabel. Det ene er antennemodulen, den andre analyse- og datamodulen.



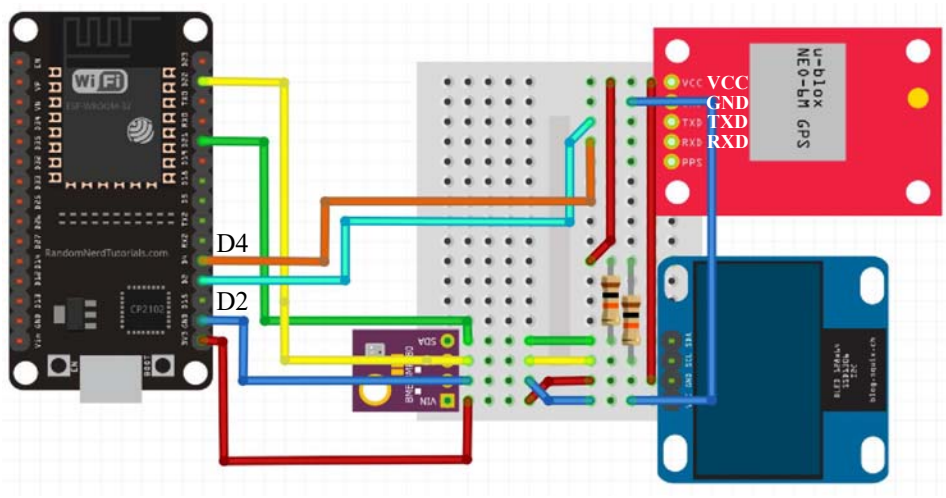
- Enheten er bygget opp rundt en ARM-prosessor
- Mottakerfrekvens: 1575,42 MHz
- Antall kanaler: 50 kanaler
- Følsomhet: -161dBm ("Tracking & Navigation)
- Nøyaktighet: - Horisontal: 2,5 m
  - Hastighetsmåling: 0,1 m/s
  - Kurs retning: 0,5°
  - Tid: 30 ns (tidspulssignalets nøyaktighet)
- Dato: Leverer data og tid ned til sekunder
- Låsetid:
  - Varm (Hot) start: 1 sek. i gjennomsnitt
  - Kald start: 27 sek. i gjennomsnitt
- Maks verdier:
  - Maks. målehøyde: 50 000 meter
  - Maks. hastighet 500 m/s
  - Maks. akselerasjon 4 g
  - Maks. oppdatering 5 Hz
- Forsyning:
  - Spenning: 2,7 – 3,6 V DC
  - Strømforbruk: Max. 47 mA

25.https://www.banggood.com/no/GY-GPS-Module-Board-9600-Baud-Rate-With-Antenna-p-1196661.html

- Dig. utgang: Spenningsnivå: Digitale signaler, nivåer (0 – 2,85 V)
  UART, USB (12 Mbit/sek), SPI (100 kbit/sek)
  Baud rate: 9 600 baud (symboler/sek.)
  Format: NMEA GSV, RMC, GSA, GGA, GLL, VTG, TXT
- Størrelse (med "stor" antenne): 25 x 25 x 8 mm (antennemodul) +
  25 x 35 x 5 mm (kontrollmodul)
- Temperaturområde: –40°C til 85°C
- Backup batteri: For å holde data ved "Power down"

5.7.3 Oppkobling

Koble opp GPS-mottakeren som vist på figuren under.



Også denne kretsen skal ha 3,3 V. Kretsen kommuniserer på serielinje (ikke I^2C) som vi kobler til D2 og D4.

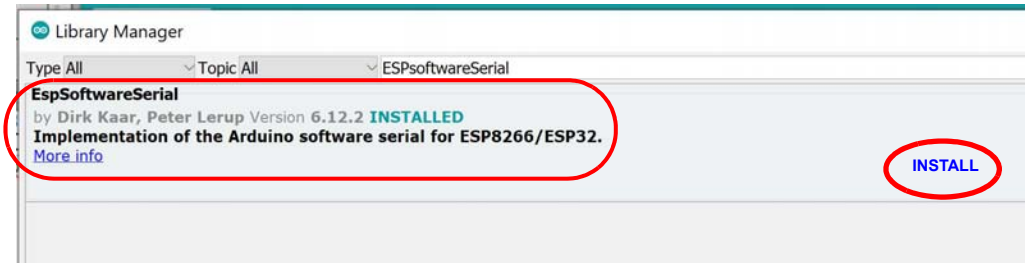
ESP32 – D2 (Tx) —> GPS – RXD
ESP32 – D4 (Rx) —> GPS – TXD
ESP32 – GND —> GPS – GND
ESP32 – 3V3 —> GPS – VCC

5.7.4 Programmet

Også denne trenger to biblioteker som må installeres.

1. **Installasjon av biblioteker:**
   SoftwareSerial.h - Gir oss mulighet til å bruke alle digitale porter for seriekommunikasjon
   TinyGPS++.h - Gir os mulighet til å kommunisere med GPS-mottakeren

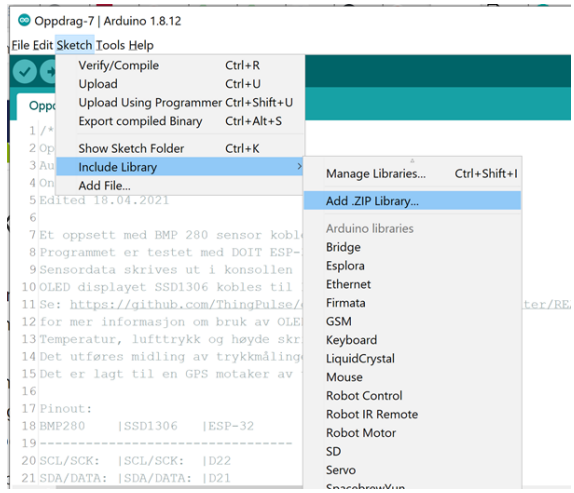
ESP softwareSerial: Vi bruker som vanlig Sketch/Include Library/Manage Libraries og skriver ESP softwareSerial i søkefeltet.



TinyGPSplus: Dernest skal vi installere biblioteket som kommuniserer med GPS-mottakeren. Det ser ut til at dette må lastes ned og installeres "manuelt"

Gå til: https://github.com/mikalhart/TinyGPSPlus/tree/v1.0.2a
og velg ***Code*** (grønn knapp) og last ned *Download.zip*
Legg fila på et sted der du kan finne den igjen.

Installer biblioteket manuelt ved å gå til *Sketch/Include Library/Add zip. library* og velg det biblioteket du lastet ned: *TinyGPSPlus-1.0.2a*



2. **Inkluder bibliotekene i programmet**
Øverst i programmet inkluderer man de to bibliotekene. Dermed får programmet adgang til alle funksjonene som trengs for å kommunisere med GPS-mottakeren via en digital port.

```
// Inkludering av biblioteker
  #include <SoftwareSerial.h>  // Inkluder bibliotek for å skrive til serielinje (GPS)
  #include <TinyGPS++.h>       // Inkluder bibliotek for å les data fra GPS NEO 6M
```

Disse legges inn helt i starten av programmet før deklarasjon av variabler og setup().

3. **Definer objekter knyttet til GPS-mottakeren og seriekommunikasjonen**

Definerer to konstanter `RXPin` og `TXPin` som angir portene vi skal bruke for å kommunisere med vår GPS-mottaker:

```
  static const int RXPin = 2, TXPin = 4;
```

Deretter oppretter vi to objekter.

```
  TinyGPSPlus gps;                    // Oppretter objektet gps av typen TinyGPSPlus
  SoftwareSerial ss(RXPin, TXPin);// Oppretter objektet ss av typen SoftwareSerial
```

Disse kommandolinjene plasseres før setup()-funksjonene.

4. Initialiser og sett datahastigheten til GPS-mottakeren

I setup()-funksjonen initialiserer vi serielinjen med følgende kommando. Siden GPS-mottakeren har som eneste kommunikasjonhastighet 9600 baud så setter vi det inn i argumentet:

```
ss.begin(9600);        // Setter kommunikasjonshastigheten til GPS'en til 9600 baud
```

5. Les ut lengde- og breddegrad og høyde fra GPS-kretsen

Følgende rutine leser lengde- og breddegrad og høyde fra GPS-kretsen, etter først å ha sjekket om det er mottatt gyldige data. Om det *ikke* er tilfelle, skriv "UGYLDIG" i monitoren. Erfaringer viser at en må gjerne gå noen runder i rask rekkefølge før man får "napp". Har man gått 25 runder uten å få gyldige data, venter en til neste runde Det har derfor vist seg lurt å legge inn en for-loop.

```
for(int i = 0; i<25; i++) // Den må ha noen forsøk før data kan leses
{
  while (ss.available() > 0) // Sjekk om data er tilgjengelig på ss
  }
  if (gps.encode(ss.read())) // Les data
    {
    if (gps.location.isValid()) // Sjekk om dataene er gyldige
      {
        breddegrad = gps.location.lat();    // Les lengdegrader
        lengdegrad = gps.location.lng();    // Les breddegrader
        hoyde      = gps.altitude.meters(); // Les høyde i meter
        delay(2000);
        break;
      }
    else
      {
      Serial.println(F("UGYLDIG")); // Varsle om data er ugyldige
      }
    }
  if (millis() > 5000 && gps.charsProcessed() < 10)
    {
     Serial.println(F("GPS ikke detektert: Sjekk oppkobling"));
    }
  }
delay(20); // La det gå litt tid mellom hver runde
}
```

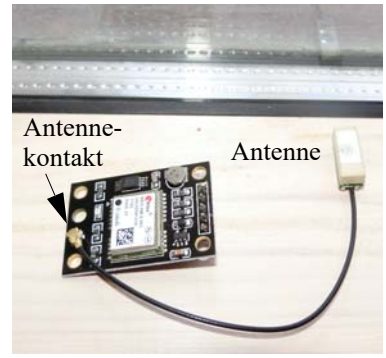
Det anbefales å legge lesningen av GPS-data inn i en egen funksjon med et hensiktsmessig navn. Vi valgte også å definere *breddegrader, lengdegrader* og *hoyde* som globale variabler.

6. **Skriv data til monitoren**
   Dernest kan man skrive ut bredde- og lengdegrad, og høyde til monitoren

```
Serial.print(breddegrad,6);
Serial.print(", ");
Serial.print(lengdegrad,6);
Serial.print(", ");
Serial.prinln(hoyde,1);
```

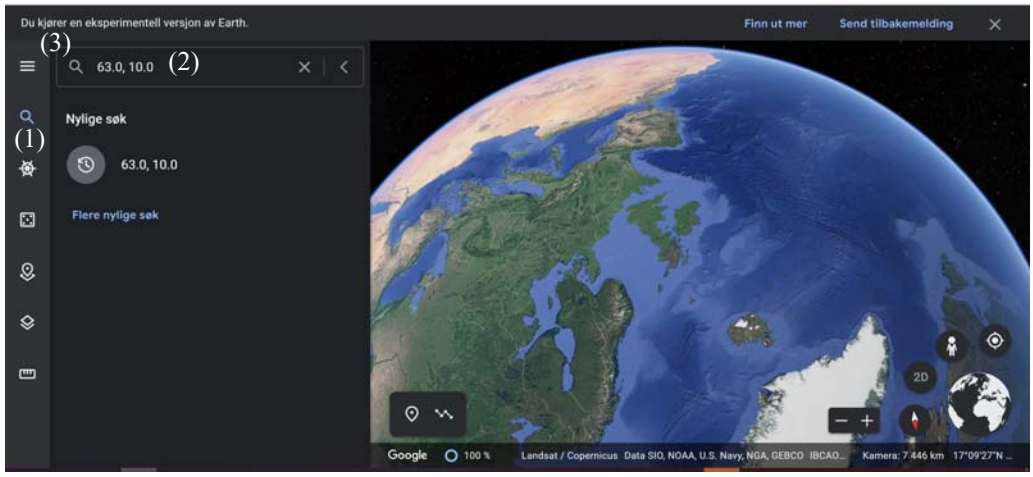
   Dersom man skriver verdiene med komma i mellom, kan man ta dem rett i Google Earth og få plottet lokasjonen. Vi har valgt å ha med 6 desimaler for lengre- og breddegrader og 1 desimal for høyden (den ene desimalen er neppe signifikant).

7. **Test programmet**
   Pass på at antennekontakten er plassert i sokkelen på kortet. For å få testet programmet så vil det være nødvendig å legge antenna i vinduskarmen, helst slik at den har fri utsikt til en større del av himmelen. Dette er spesielt viktig siden kretsen har en særdeles liten satellittantenne.



   I hus hvor man befinner seg nært et tretak kan det gå greit også innendørs.

8. **Plott posisjon i Google Earth**
   Gå til https://earth.google.com/
   Velg *Søk* i venstre meny (1) og skriv inn koordinatene på formen f.eks. 63.00000, 10.00000 (Breddegrad, Lengdegrad) (2) og trykk deretter forstørrelsesglasset til venstre for koordinatene (3). Det går ikke an å skrive inn høyden.



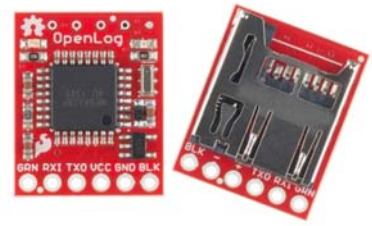
9. **Skriv bredde-, lengde og høyde ut på OLED-displayet**
   Fjern temperaturen og flytt lufttrykk og beregnet barometriske høyde opp til øverst på displayet, og legg inn GPS-bredde-, lengde og høyde nederst på displayet.
10. **Sammenligning den barometriske og GPS-høyden**
    Gjør en sammenligning mellom barometrisk høyde og GPS-høyde og studer avvikene. Hva kan det komme av at de ev. avviker? Har du nylig kalibrert Barometrisk høyde?

5.8 Oppdrag 8 – Logging av data på SD-kort

Vi skal nå installere en liten SD-kort logger og skrive data til file over tid. Til det bruker vi OpenLog.

> **Oppdrag 8:** *Monter og koble opp SD-kort leseren OpenLog som gjør det mulig å lagre måledata på et SD-kort. Skriv nummer på målingen, lengdegrad, breddegrad, høyde, lufttrykk, temperatur på en linje i filen.*

5.8.1 OpenLog – Lagring av data på SD-kort



OpenLog datalogger er en enhet som på enkel måte gjør det mulig å lagre store datamengder på et micro SD-kort. Enheten finnes i flere utførelser og prisklasser. Ønsker man billige utgaver finnes de for under 56 kr. hos BangGood[26].

Figuren til høyre viser for- og baksiden av komponenten. På baksiden ser vi holderen for SD-kortet.

Enheten kan tilføres supply-spenninger fra 3,3 – 12 V, spenningene inn på RXI-inngangen bør være i området 2,0 – 3,3 V, mens spenningene ut av TXO er 3,3 V. Strømtrekket er fra 2 – 5 mA uten SD-kort, og fra 5 – 6 mA med kortet installert. Under skriving til kortet vil strømtrekket komme opp i 20 – 23 mA.

OpenLog er bygget opp rundt en ATmega328, med en klokkefrekvens på 16MHz. ATmega328 benytter Optiboot bootloader, som gjør OpenLog kompatibel med Arduino Uno innstillingene i Arduino IDE. Normalt vil man kommunisere med kortet via dets UART (RXI og TXO). To lysdioder finnes på OpenLog kortet. STAT1 – (blå) blinker når det overføres et tegn. STAT2 – (grønn) blinker når SPI-grensesnittet er aktivt.

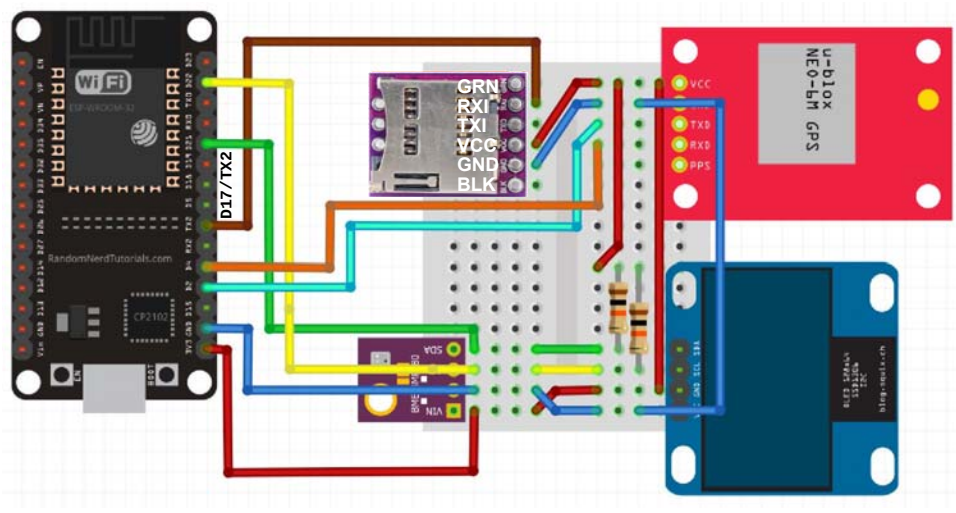
OpenLog kan håndtere kortstørrelser på opp til 32 GByte.

For mer informasjon se avsnitt 6.5 på side 124. Her finnes detaljer om mer avansert programmering av kretsen.

26. https://www.banggood.com/CJMCU-717-OpenLog-Data-Recorder-Flash-Recorder-Sensor-Module-Support-64GB-Micro-SD-Card-p-1461328.html

5.8.2 Oppkobling

Figuren under viser oppkoblingen av kretsen på koblingsbrettet vårt.



Siden vi bare skal skrive *til* SD-kortet så kobler vi bare til Vcc, GND og RXI linjen. RXI kobler vi til D17 – TX2 på ESP32.

5.8.3 Programmet

Strengt tatt trenger vi ikke å installere noe spesielt bibliotek for å skrive til OpenLog. Vi kan bruke vanlige print kommandoer for skriv og formatere tekst og variabler til OpenLog.

1. **Inkludering av biblioteker**
   Vi passer på å inkludere følgende bibliotek:

   ```
   #include <HardwireSerial.h>
   ```

   Tidligere har vi inkludert:

   ```
   #include <SPI.h>
   ```

   Disse legges inn i starten av programmet om de ikke alt ligger der.

2. **Lag et object**
   Vi lager et object av typen `HardwireSerial` med navnet `ol` (for OpenLog):

   ```
   HardwareSerial ol(1);
   ```

   Som plasseres før setup()-funksjonen

3. **Bestemmer porter for kommunikasjon med OpenLog**
   Vi velger å koble serielinjen til portene D16 og D17.

   ```
   static const int RXOL = 16, TXOL = 17; // Porter for kommunikasjon til OpenLog
   ```

Plasseres før setup()-funksjonen

4. **Initialiserer serieportene for OpenLog**
   Her formidler vi også hvilke porter vi planlegger å bruke for denne kommunikasjonen, RXOL (D16) og TXOL (D17):

   ```
   ol.begin(115200, SERIAL_8N1, RXOL, TXOL);
   ```

   Kommandoen plasseres i setup()-funksjonen.

5. **Skriv til OpenLog**
   Opprett gjerne en egen funksjon for skriving til OpenLog. Vi har valgt å kalle funksjonen:
   `void writeDataToOpenLog()`

   Selve skriving gjøres på vanlig måte med print-kommandoer, men vi bruker objektet `ol` som vi har opprettet. Dersom vi ønsker å vise resultatene i Google Earth, bruker vi formatet: *lengdegrad, breddegrad, høyde*

   Dette vil kunne ta seg ut slik:

   ```
   ol.print(lengdegrad,6);
   ol.print(",");
   ol.print(breddegrad,6);
   ol.print(",");
   ol.println(hoyde,1);
   ```

   Filen skal brukes til å visualisere traseen i Google Earth, så skal det ikke være mellomrom hverken foran eller etter komma.

   MERK: Dersom vi ønsker å plotte data i Google Earth vil det være lurt inntil videre kun å skrive GPS-data skilt med komma.

   Disse kommandoene legges inn i funksjonen som skriver til SD-kortet. Funksjonen kalles fra void loop()-funksjonen som vanlig.

6. **Skriv programmet**
   ... og test ut og bekreft at det fungerer som det skal.

5.9 Oppdrag 9 – Fjern duplikater

Vi ønsker å fjerne dubletter av måledata før de skrives inn i loggfila.

> **Oppdrag 9:** *Vi legger mer merke til at eksakt samme verdier av lengde- og breddegrad kommer flere gang og mistenker at samme verdi leses ut flere ganger etter hverandre. Lag en algoritme som ekskluderer like verdier før de skrives på fil.*

5.9.1 Oppkobling

Ingen ekstra oppkobling for dette oppdraget.

5.9.2 Programmet

1. **Identifiser dubletter**
   Vi foreslår at det gjøres en sammenligning mellom siste og neste siste måling. Dersom disse er like skrives den siste målingen ikke til OpenLog, dersom de ikke er like skrives den til loggfila. Det kan synes fornuftig at denne sammenligningen gjøres i funksjonen som skriver dataene til OpenLog: `void writeDataToOpenLog()`.

   Tips: bruk en if-setning med sammenligning i argumentet.

2. **Test programmet**
   ... og finn ut om det fungerer som det skal.

5.10 Oppdrag 10 – Gå en tur og vis turen i Google Earth

Vi ønsker å samle stedsdata fra en tur i nærområde for så å forsøke å vise dem i Google Earth.

> **Oppdrag 10:** *Gå en tur og samle inn lengde- og breddegrad og høyde fra en tur på ca. 10 minutter i nærområdet der du holder til. Last dataene over i en editor og inkluder dataene i en KML-fil slik at du kan vise turen i Google Earth.*

5.10.1 Oppkobling

For å kunne utføre dette oppdraget vil det være nødvendig å ta med seg utstyret og gå en tur i nærområdet, mens utstyret samler inn GPS-data og legger på fil. For å kunne tilføre utstyret strøm, må man enten ha med PC'en eller ta med et batteri. En power pack er fin til dette formålet siden USB-kabelen fra ESP32 kan kobles direkte til batteriet som vil levere strøm til ESP32 så lenge power pack'en er ladet.

5.10.2 Programmet

Denne gangen skal det ikke skrives så mange programmer, men vi skal hente ut fila fra OpenLog og pakke den inn i en KML-kode som gjør den forståelig for Google Earth.

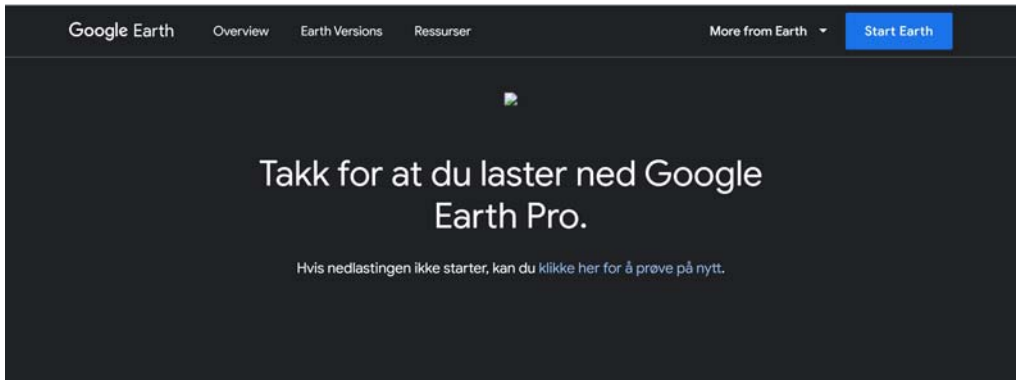
Det er viktig at posisjondataene skrives på riktig måte:

```
10.454113,63.425967,72.4
```

Lengdegrad,Breddegrad,Høyde med komma, men uten mellomrom.

1. Installasjon av Google Earth

Google Earth kan lastes ned og installeres fra følgende adresse: https://www.google.com/earth/versions/download-thank-you/



2. Plotting av en trase i Google Earth

Dersom man ønsker å plotte en trase i Google Earth kan man benyttet et programmerings-språk kalt "Keyhole Markup Language" (KML) som er et språk utviklet for visualisering av to- og tredimensjonale strukturer knyttet til kartdata.

Siden språket er temmelig omfattende og vi kun trenger å bruke en beskjeden del av det, så benytter vi en ferdige programkode og klipper inn våre data for lengde-, breddegrad og høyde. Disse legges inn som en liste med data i kml-koden (se under), før kml-fila lagres med et ønsket navn.

Koordinater og høyde data legges inn som vist under:

-112.2550785337791,36.07954952145647,2357
Lengdegrader [°],Breddegrader [°],Høyde [m]

Programkode skrevet i kml (legg merke til at et sett med eksempelkoordinater og høyde er klippet inn - rød kode). Her er det lagt inn 11 posisjoner. Du må gjerne legge inn flere eller færre.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <name>Paths</name>
    <description>Examples of paths. Note that the tessellate tag is by default
      set to 0. If you want to create tessellated lines, they must be authored
      (or edited) directly in KML.</description>
    <Style id="yellowLineGreenPoly">
      <LineStyle>
        <color>7f00ffff</color>
        <width>4</width>
```

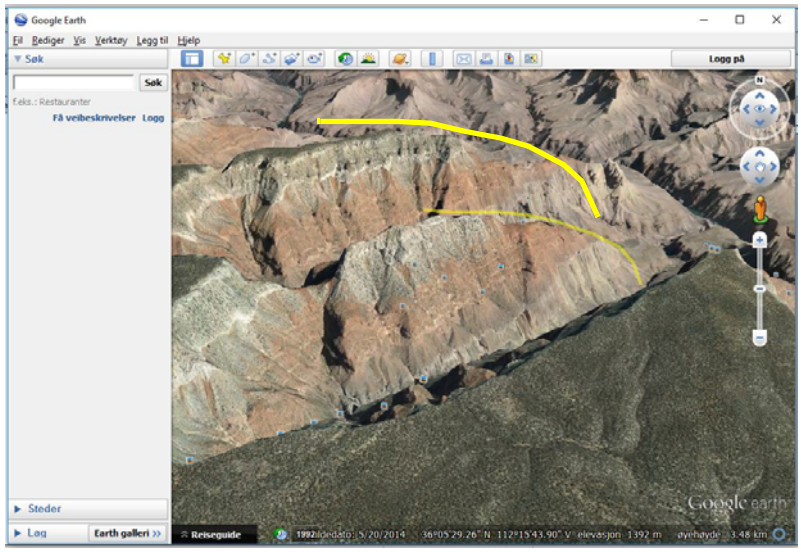
```
    </LineStyle>
    <PolyStyle>
      <color>7f00ff00</color>
    </PolyStyle>
  </Style>
  <Placemark>
    <name>Absolute Extruded</name>
    <description>Transparent green wall with yellow outlines</description>
    <styleUrl>#yellowLineGreenPoly</styleUrl>
    <LineString>
      <extrude>0</extrude>
      <tessellate>0</tessellate>
      <altitudeMode>absolute</altitudeMode>
      <coordinates>
        -112.2550785337791,36.07954952145647,2357
        -112.2549277039738,36.08117083492122,2357
        -112.2552505069063,36.08260761307279,2357
        -112.2564540158376,36.08395660588506,2357
        -112.2580238976449,36.08511401044813,2357
        -112.2595218489022,36.08584355239394,2357
        -112.2608216347552,36.08612634548589,2357
        -112.262073428656,36.08626019085147,2357
        -112.2633204928495,36.08621519860091,2357
        -112.2644963846444,36.08627897945274,2357
        -112.2656969554589,36.08649599090644,2357
      </coordinates>
    </LineString>
  </Placemark>
 </Document>
</kml>
```

De koordinatene som pr. i dag ligger i eksempelet angir en helikopterflyvning over Grand Canyon i Colorado, USA.

Eksempeldataene byttes ut med de aktuelle dataene ***og kodefila lagres under et ønsket filnavn som ender med .kml.***

Filen hentes opp i Google Earth ved å dobbelklikke på filnavnet. Siden fila ender på .kml, så skal den automatisk bli gjenkjent av Google Earth og bli lastet inn i programmet.

En vil da få tegnet inn traseen som angitt av lista med koordinater og vist på riktig sted. Eksempelet under viser traseen til helikopteret over Grand Canyon (gul linje).



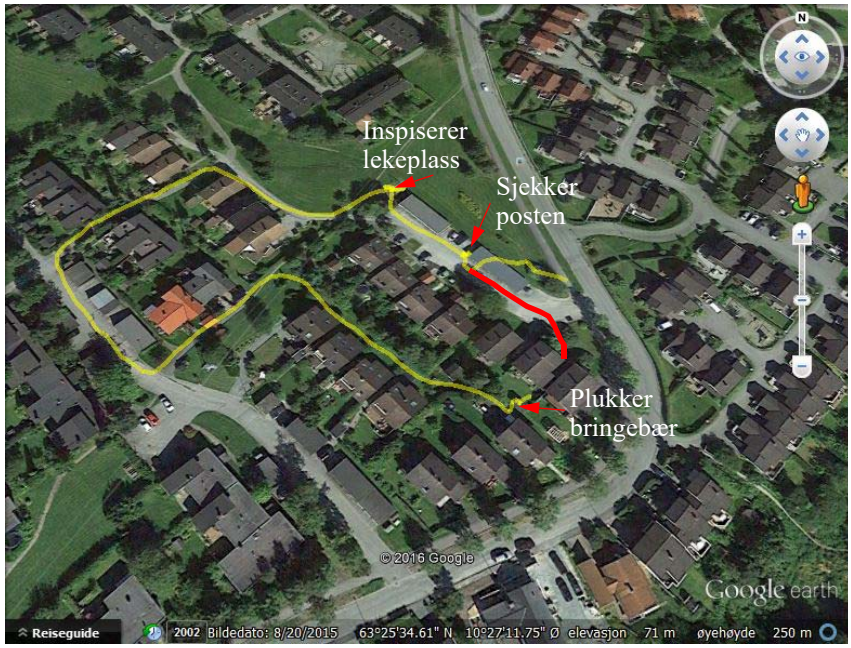
Dersom man ønsker å vise hvor man har gått en tur så kan det være upraktisk å måtte vise absolutt høyde. Høydemålinger kan ha store avvik slik at en kan oppleve at traseen går mange meter over eller forsvinner under bakken til tross for at man hadde begge beina på jorda. I slike situasjoner kan det være greit å bytte ut kommandoen:

```
<altitudeMode>absolute</altitudeMode>
```

med kommandoen:

```
<altitudeMode>clampToGround</altitudeMode>
```

Dermed vil kurven følge bakken som vist på traseen på bildet under.



Vi legger imidlertid merke til at mottakeren har problemer i starten. I dette området er avviket stort før den tar seg inn. Den røde kurven angir den riktige ruta. Deretter er den svært nøyaktig. Posisjonsdata samples hvert 3. sekund.

3. Editering av kml-fila

Hvilket program skal man så bruke for å legge inn koordinater og høyde. Et nyttig editeringsverktøy til dette formålet er Notepad++. Dette programmet er en litt avansert teksteditor som ikke gjør noen endringer med filformatet så fremt man ikke ønsker det. Notepad++ kan lastes ned fra:

https://notepad-plus-plus.org/downloads/

For å forenkle prosessen med å klippe inn data i kml-koden, er det praktisk å skrive koordinat- og høydedata i det formatet som programmet ønsker: Lengdegrad,breddegrad,høyde. Husk å bruke punktum som desimalpunkt og komma mellom hver verdi. ***NB! Det skal ikke være mellomrom etter kommaene.***

5.11 Oppdrag 11 – Plotting av lokasjon i Circus of Things

Vi skal nå koble programmet opp mot Circuit of Things (CoT) og etter hvert plotte traseen omtrent i real time, mens vi går. Dette forutsetter at vi kan bære med oss ruteren, hvilket betyr at vi må bruke mobiltelefonen til dette formålet.

Oppdrag 11: *Koble opp ESP32 mot Circus of Things og lag konsoller som viser lufttrykk, temperatur, barometrisk høyde. GPS data lengde- og breddegrader og høyde skal knyttes til de enkelte målingene og ikke overføres som separate verdier.*

Alternativt er det aktuelt å logge måleverdier og posisjon der du sitter. Forutsetningen er at du er plassert slik at GPS-mottakeren har tilgang til et passende antall satellitter.

5.11.1 Oppkobling

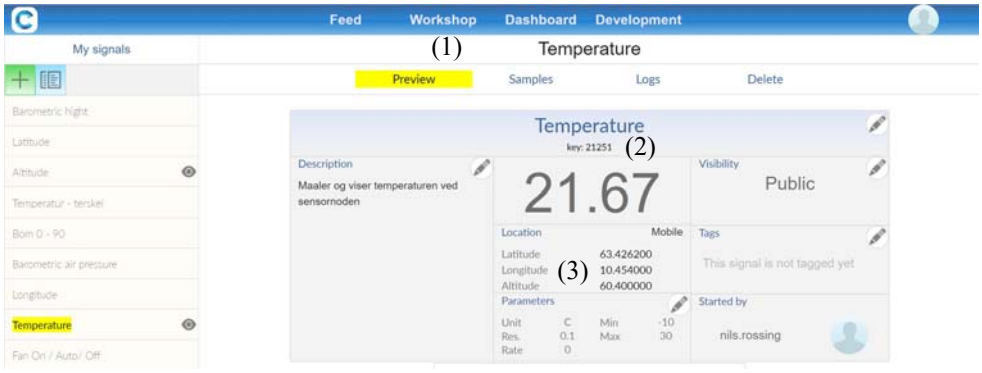
Det er ikke noen ekstra oppkobling for dette oppdraget.

5.11.2 Konsollet

I dette oppdraget må vi opprette konsoller ved Circus of Things[27] (CoT), dvs. konsoller for temperaturmåling, lufttrykk og den beregnede verdien for barometrisk høyde.

Vi antar at dere er kjente med CoT slik at vi her kun tar en kort introduksjon (ev. se 4.1, side 55).

1. **Log inn på Circus of Things (CoT)**
   Log inn eller opprett konto på www.circusofthings.com.
2. **Etabler tre signaler under fanen Worshop (1)**
   - Temperature (fra BMP280)
   - Barometric air pressure (fra BMP280)
   - Barometric hight (beregnet fra lufttrykk)

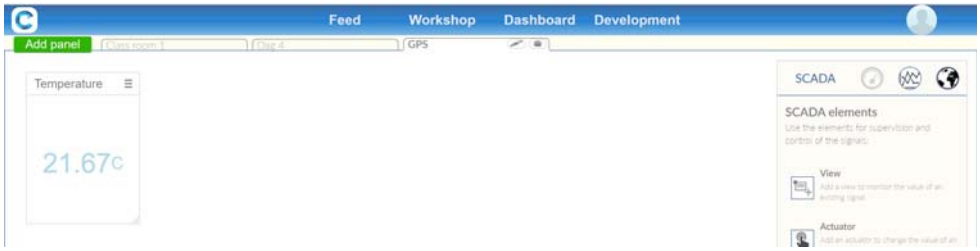


Registrer nøkkelkoder for hvert av signalene for senere bruk.

Vi legger merke til at eksempelet inkluderer informasjon om lokalitet (Location (3)). Denne informasjon vil bli lagt til i programmet og er før vi har laget programmet, ikke tilgjengelig.

27. Om ikke så henvises til et tidligere kurshefte: Wi-Fi programmering med ESP32 – (YFL) (Rev. 2.0), kap. 5.1. Se: https://www.ntnu.no/skolelab/bla-hefteserie Under fanen Grunnkurs programmering 32 – Dag 3

3. **Etabler et panel**
   Et panel er som vi kanskje husker, en samling konsoller. I vårt tilfelle skal vi foreløpig lage et panel (velg *Add panel* (grønn fane)) som kun inkludere en parameter, ett konsoll. Vi velger temperatur for enkelhets skyld.



   Dette gjør vi ved å velge *View* til høyre for så å velge signalet Temperatur.

   Resten skal vi komme tilbake til når vi har bygget opp programmet.

5.11.3 Programmet

La oss først bygge opp programmet.

Det er i forbindelse med dette opplegget laget et nytt bibliotek ved CoT som gjør det mulig å inkludere posisjonsdata sammen med signaler når man bruke ESP32. Tidligere var det bare mulig når man bruke Rapberry Pi.

1. **Installer biblioteket**
   Siden det er noe usikkert om biblioteket er lagt ut ved CoT så velger vi å hende det fra utviklerens eget GitHub:
   https://github.com/Hasjald/COT_Modified_ESP_32_Lib
   Velger *Code* og *Download zip*

   Legg den nedlastede fila på et sted der du kan finne den igjen og installer den i Arduino IDE ved å velge: Sketch/Include Library/Add zip Library

2. **Inkluder biblioteket i programmet**
   Dette gjørs på vanlig måte i starten av programmet:

```
#include <CircusESP32Lib.h>
```

   Dette gjøres helt i starten sammen med inkludering av øvrige biblioteker

3. **Etablering av informasjon for oppkobling**
   For at vi skal klare å koble oss opp til CoT så må vi første koble oss igjennom vår egen ruter og videre til hjemmesiden til CoT.

```
// ------------------------------------------------------------------------
// CircusESP32Lib relaterte deklarasjoner for oppkobling til WiFi og CoT
// ------------------------------------------------------------------------
char ssid[] = "Sett inn ditt ruternavn>";// Skriv inn navnet til din ruter
char password[] =  "<Sett inn ditt ruterpassord>";// Skriv passordet til din ruter
char token[] = "<Sett inn din token>";    // Plasser din brukeridentitet her (token)
char server[] = "www.circusofthings.com";// Her ligger serveren
```

```
char BMP280_temperature_key[] = "21251"; // Nøkkel-kode for å kommunisere temp.
char BMP280_pressure_key[] = "13531";    // Nøkkel for å kommunisere lufttrykk
char BMP280_hight_key[] = "28856";       // Nøkkel for å kommunisere høyde
```

Disse plasseres før setup()-funksjonen

4. **Opprett objekt**
   Deretter skal vi opprette et objekt som vi bruker når vi henvender oss til bibliotekets funksjoner:

```
CircusESP32Lib circusESP32(server,ssid,password);
```

   Det er også i denne kommandoen vi overfører informasjon om ruteren (navn, passord og token). Denne plasseres før setup()-funksjonen

5. **Initialisering av CoT-kommunikasjon**
   Vi setter da opp kommunikasjon til CoT med følgende kommando:

```
circusESP32.begin(); // Initialiser Circus of Things
```

   Denne plasseres i setup()-funksjonen

6. **Opprett en funksjon**
   Vi velger å opprette en funksjon som vi kaller:

```
sendDataToCoT()
 {
    // Her skrives kroppen til funksjonen
 }
```

   Man står selvfølgelig fritt til å velge navn på funksjonen, men det kan være lurt å velge et navn som gir mening.

7. **Send data til CoT**
   Vi legger merke til at kommandoene som overfører informasjonen også inneholder lengde- og breddegrader, og høyde. Dette er informasjonen den trenger for å kunne vise målingene på kartprojeksjonen og som er nytt ved dette biblioteket:

```
circusESP32.write
 (BMP280_temperature_key,temperature,breddegrad,lengdegrad,hoyde,token);
circusESP32.write
 (BMP280_pressure_key,pressure,breddegrad,lengdegrad,hoyde,token);
circusESP32.write
 (BMP280_hight_key,hight,breddegrad,lengdegrad,hoyde,token);
```

   Vi har merket med rødt det som er nytt for dette biblioteket. Disse kommandoene plasseres i funksjonen `sendDataToCoT();`

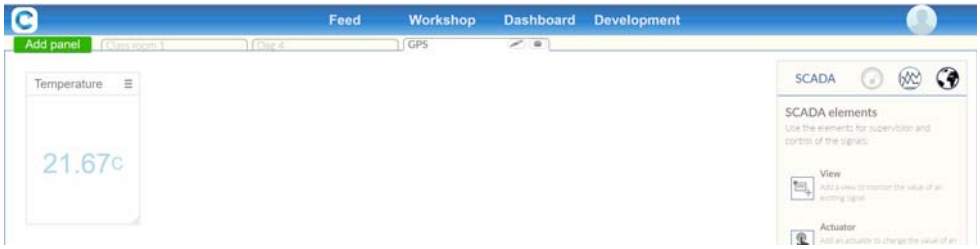
8. **Kall av funksjon for overføring av data**
   Vi må heller ikke glemme å kalle funksjonen i void loop() der vi ønsker å overføre data:

```
sendDataToCoT();
```

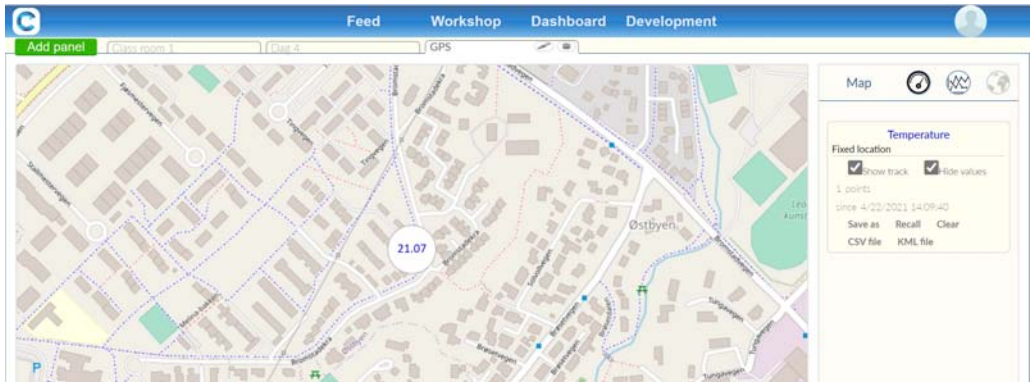
5.11.4 Konsollet – Visning av innsamlede data

Log inn hos Circus of Things og gå til fanen workshop. Der vil du se at signalene er supplert med posisjonsdata fra GPS-målingene som vist på figuren under:

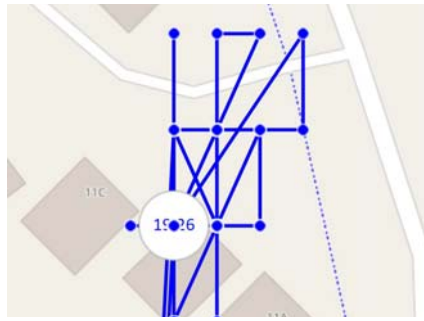
Vi skal nå gå tilbake til vårt panel som viser temperaturen:



Ved å velge *jordkloden* lengst til høyre, får vi opp et kart over området der målingene er gjort.



I dette tilfellet har sensornoden som utfører målingene, vært i ro. Vi legger merke til at siste måling er avmerket med tallverdi. Lar vi målingene pågå en stund vil flere målinger akkumuleres og vi vil få en samling av målepunkter som kan avvike noe fra hverandre. Figuren til høyre viser en serie målinger gjort natt til 22. april 2021. Dette er med en litt eldre utgave av biblioteket som bare overførte 4 desimaler etter komma. Vi ser at støy gjør at posisjonen hopper mellom flere verdier. I senere utgaver av biblioteket er antall desimaler som overføres endret til 6.



Det er ikke sikkert at bedre oppløsning vil gi mindre spredning, men målepunktene vil hoppe mellom et mer finmasket nettverk.

Vi legger også merke til at CoT trekker streker mellom målepunktene.

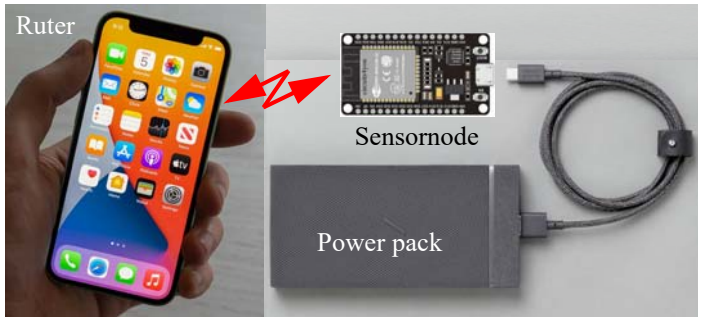
5.12 Oppdrag 12 – Gå en runde å samle inn data

> **Oppdrag 12:** *Sørg for å koble ESP32 opp mot Internett med mobiltelefonen slik at du kan ta med deg utstyret og gå en tur. Gå en runde og samle inn stedskoordinater, lufttrykk, barometrisk høyde og temperatur.*
>
> *Gå inn på panelet på CoT og vis måledata av lufttrykk som funksjon av ruta du har gått.*

Vi har registrert at GPS-koordinatene oppdateres ganske langsomt når mottakeren ligger i vinduet

5.12.1 Oppkobling

Dersom du planlegger å bære med deg kretsen så kan det være praktisk å koble den til et eksternt batteri. Det enkleste er om du har en power bank med USB-plugg slik figuren til høyre antyder (en må selvfølgelig ta med seg hele oppkoblingen, men kan la PC'en bli igjen hjemme).



5.12.2 Konsollet

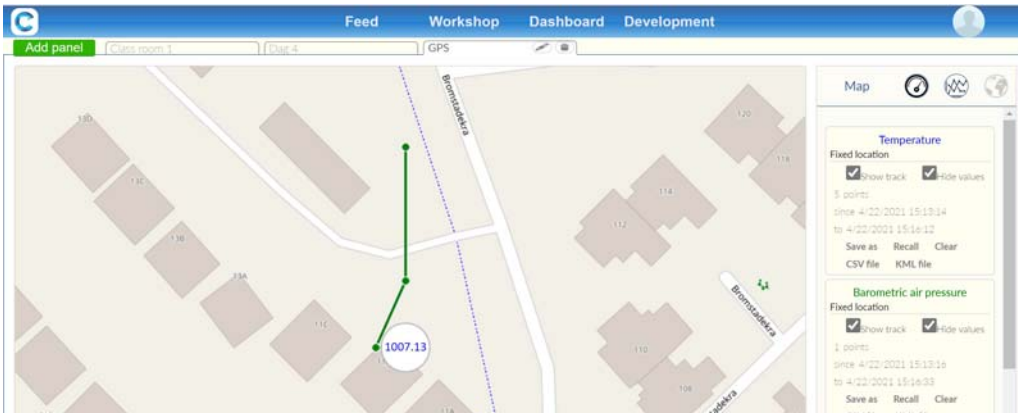
Sørg for at du er logget inn på CoT. Dermed skulle dataene kunne tikke rett inn i presentasjonen hos CoT.

5.12.3 Programmet

Det er viktig å huske på at programmet er utstyrt med nødvendig navn og passord for å bruke mobiltelefonen som ruter.

5.12.4 Måleresultater

Måleresultatene vil nå framstå som punkter med mellomliggende streker som vist på kartet under:



Ved hjelp av panelet til høyre kan man velge å vise temperatur eller lufttrykk. Dessuten kan vi velge å se bort fra verdiene eller traseen ved å hake av i de respektive boksene.

5.13 Oppdrag 13 – Sett opp RTC-klokka og les av tidspunkt

En RTC (Real Time Clock) er en krets som kan holde rede på tiden når strømmen går eller alle andre kretser legges i dvale. Den må derfor være svært nøktern mht til strømtrekk samtidig som den må gå riktig over lang tid.

> **Oppdrag 13:** *Monter RTC-klokka og koble den opp på I^2C-bussen, og lag et lite program som stiller klokka og leser av nøyaktig tidsangivelse med dato og tid som skrives ut i monitoren.*

5.13.1 Bakgrunn "Real Time Clock"[28]

DS1307 er en klokkekrets som er istand til å holde rede på tiden selv om hovedstrømmen skulle falle fra. Dette er mulig da den er utstyrt med et 3V batteri (CR1220) som har en holdbarhet på minst 5 år. Dersom spenningen faller under 1,25 x U_{bat} = 3,75 Volt med U_{bat} = 3,0 V, så kuttes kommunikasjonen med omverdenen, men den interne klokka fortsetter å gå. Når supply spenningen faller under batterispenning, tar batteriet over. Strømtrekket i batterimodus er på under 500 nA. Normalt vil kretsen kreve typisk 5,0 V supplyspenning.

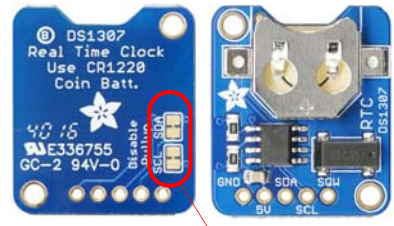
28. Datablad: https://datasheets.maximintegrated.com/en/ds/DS1307.pdf

Etter at klokka er riktig stilt så holder den rede på sekunder, minutter, timer, dager, dato og år, inkludert skuddår. I tillegg til at kretsen inneholder klokke og kalender så har den et lite minne på 56 byte. Den kan også gi ut en neddelt klokkefrekvens på 1 Hz, 4,0 kHz, 8,0 kHz, 32,0 kHz på pinnen SQW/OUT.

Klokkas nøyaktighet er avhengig av hvor nøyaktig krystallet er og hvor godt man klarer å stille den. Krystallet har en svingefrekvens på 32,768 kHz.

Kretsen er utstyrt med en I^2C bus som gjør det mulig å nå alle kretsens interne funksjoner.

Den integrerte kretsen er levert fra Maxim, mens breack out kortet som vi bruker her er levert av Adafruit[29]. Kretsen skal utstyres med en 3V knappecelle av typen CR1220. Kretsen har 5 terminaler:



Disse brytes, i stedet anvendes 10K pullup på koblingsbrettet

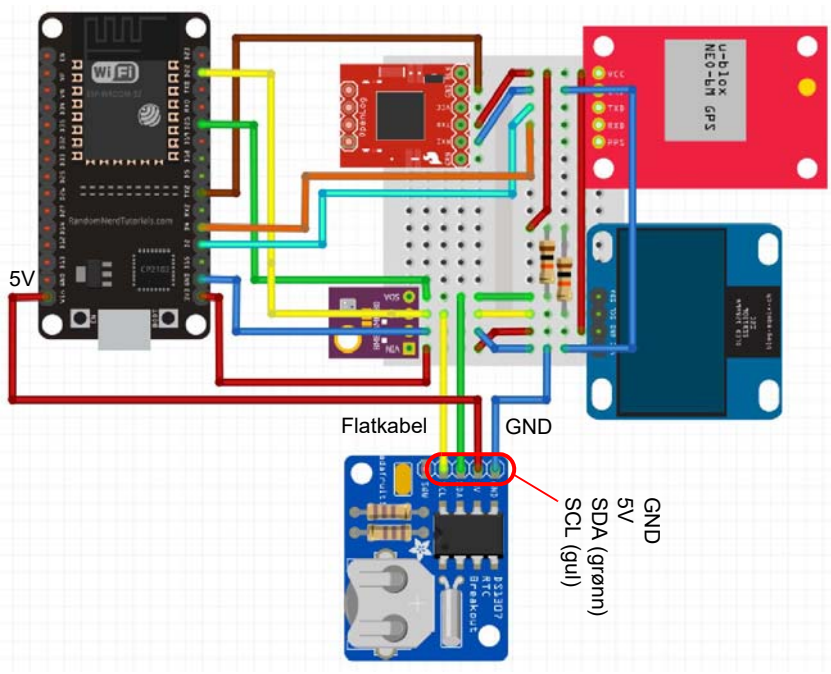
- GND Jord eller batteri –
- 5V Supplyspenning: + 4,5 til 5,5 V
- SDA Seriell datalinje for I^2C-bussen
- SCL Seriell klokkelinje for I^2C-bussen
- SQW En port som leverer en firkantkurve med 1Hz, 4kHz, 8kHz, 32kHz og 0V og 5V.

Siden kretsen har en supplyspenning på 5V og resten av komponentene bruker 3,3 V, så er det nødvendig å redusere spenningen på I^2C-bussen hos DS1307. Dette gjøres ved å bryte de to strap'ene på kortet som vist på figuren til høyre og legge inn to pullup motstander til 3,3 V på koblingsbrettet, se avsnitt 5.13.2, side 94.

29. file:///D:/Arduino/Kurs/YFL-kurs/Etterutdanning%202020/Ressurser/DS1307/Adafruit_DS1307_RTC_eng_tds.pdf

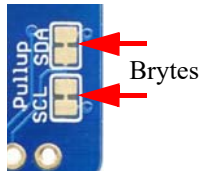
5.13.2 Oppkobling

Figuren under viser hvordan RTC-klokka kan kobles til resten av kretsen.



Siden det ikke er plass til RTC-kretsen på koblingsbrettet, så foreslås å bruke flatkabelen med fire ledninger[30]. Den femte terminalen (SQW) er ikke i bruk.

Om det ikke alt er gjort så skal strap'ene på kortet brytes slik at de eksterne på koblingsbrettet brukes istedet for de interne (2 x 10kΩ) som er koblet til +5V. Se figuren til høyre.



Monter knappebatteriet CR1220 slik at når RTC-klokka er stilt så holder den de neste årene.

5.13.3 Programmet

Det første vi må gjøre er å installere et bibliotek som er tilpasset bruk av DS1307 sammen med ESP32.

30. Kabelen fulgte med Adgangskontrollen

1. **Installer bibliotek**
   Gå til nettsiden: https://github.com/Erriez/ErriezDS1307
   Velg den grønne knappen *Code* og *Download zip*. Legg filen et sted der du finner den igjen.

   Installer filen ved å velg følgende fra menyen: *Sketch/Include Library/Add ZIP Library* og velg den nedlastede zip.filen.

2. **Inkluder biblioteket**
   Inkluder biblioteket i programmet ved å skrive:

   ```
   #include <ErriezDS1307.h>
   ```

   inn i starten av programmet.

3. **Opprett objekt**
   Opprett et objekt med et selvvalgt navn:

   ```
   ErriezDS1307 rtc;
   ```

   Vi valgt navnet `rtc`. Kommandoen legges inn i programmet før setup()-funksjonen.

4. **Initialiser RTC-klokka**
   Vi initialiserer RTC-klokka ved å bruke kommandoene:

   ```
   wire.begin(); // Initialiser I2C-bussen

   // Oppsette av RTC-klokka
   if (! rtc.begin())
     {
       Serial.println("Kunne ikke finne RTC-klokka");
       abort();
     }
   rtc.clockEnable(true);                  // Koble inn RTC-klokka
   ```

   Denne initialiserer klokka (`rtc.begin()`) og gjør den tilgjengelig (`rtc.clockEnable(true);`). Siden kretsen bruke I^2C-bussen er det viktig at `wire.begin();` er inkludert, hvilket skal være gjort tidligere.

5. **Still klokka**
   Denne kommandoen stiller klokka til den angitte datoen og tidspunktet:

   ```
   if (!rtc.setDateTime(12, 22, 56, 26, 04, 2021, 0))
     // Timer, minutter, sekunder, dag, måned, år, ukedag
     {
       Serial.println(F("Sett dato/tid mislyktes"));
     }
   ```

   Denne legges inn i setup()-funksjonen.

6. **Avlesning av dato og tid**
   Avlesningen kan gjøres i en egen funksjon. Vi har valgt å gjøre det så enkelt som mulig.

   ```
   uint8_t hour, minu, sec, mday, mon, wday;
   uint16_t year;
   ```

```
// Read date/time
if (!rtc.getDateTime(&hour, &minu, &sec, &mday, &mon, &year, &wday))
  {
   Serial.println(F("Lesing av dato/tid mislyktes"));
   return;
  }
```

Det opprettes variabler for hver av tids- og datoangivelsene. Disse er av typen byte uten fortegn (`uint8_t`) eller for år av typen *word* (16 bit) uten fortegn (`uint16_t`).

7. **Utskrift av tid og dato**
   Utskriften til f.eks. monitor gjøres på vanlig måte. Følgende kan være et passende format:

   ```
   31.12.2020 13:30:00
   ```

   Bruk Serial.print kommandoer til å skrive ut tid og dato som vist over.
   Det er ønskelig at tomme tall erstattes med 0 eksempel vis når minutter og sekunder er mindre enn 10 så skrives:

   ```
   31.12.2020 12:04:08
   ```

   Dette kan ev. gjøres slik:

   ```
   if(minu < 10) {Serial.print(F("0"));}
   ```

   Er tallene under 10 legges det til en 0 foran tallet slik at antallet sipper blir det samme uansett.

8. **Test programmet**
   ... og se at resultatet blir som forventet

5.14 Oppdrag 14 – Legg inn tid og dato i OpenLog-fila og et konsoll hos CoT

Til slutt skal vi benytte RTC-klokka til å legge inn rett dato og tid i loggefila og hos CoT.

> **Oppdrag 14:** *Legg inn nøyaktig tidsangivelse i loggefila på OpenLog og i konsollene hos CoT.*

5.14.1 Oppkobling

Det trengs ingen ekstra oppkobling for dette oppdraget.

5.14.2 Programmet

Ta utgangspunkt i oppdrag 12 og 13 og overfør programkoden fra oppdrag 13 til oppdrag 12 som er nødvendig for å kunne:

- Skrive ut dato og tid foran hver måling som blir logget på OpenLog.
- Overføre tidspunkt og dato til et konsoll hos CoT.

Tips til organisering

Det som skal skje i dette oppdraget er å skrive inn tidspunkt og dato for hver måling som legges på SD-kortet. Vi kan tenke oss at det vil se ut noe slikt:

```
31.12.2020 13:30:00 63.425967,10.454113,72.4
```

Dato . måned . år time : minutt : sekund lengdegrad, breddegrad , høyde over havet

Det er flere måter å gjøre dette på. Her er et forslag:

- Variablene som holder tids- og datoverdiene deklareres globalt
- Tids- og datoverdiene hentes fra RTC-klokka i en egen funksjon
- Tids- og datovariablene skrives inn på SD-kortet gjøres i funksjonen som er laget til dette formålet.

1. **Inkluder biblioteket for RTC-klokka**
   Inkluder biblioteket, opprett et objekt, initialiser og gjør biblioteksfunksjonen tilhørende RTC-klokka tilgjengelig. Bruk oppdrag 13 som mal.
2. **Deklarasjon av tid- og datovariabler**
   Vi velger å deklarere variablene som holder de enkelte delene av tid og data som globale. Bruk oppdrag 13 som mal.
3. **Les av klokka**
   Opprett en funksjon som leser av RTC-klokka og legger de ulike verdiene inn i de globale variablene. Bruk oppdrag 13 som mal.
4. **Skriv tid og dato i loggefila**
   Skriv tidspunkt og dato inn i loggefila for hver måling. Dette kan f.eks. gjøres i funksjonen som skriver data til OpenLog.
5. **Test programmet**
   Test programmet og sjekk i loggefila at tid og dato kommer inn på rett plass.

Det siste som skal gjøres er å overfør tid og dato til et konsoll hos CoT.

6. **Overføring av tid og dato til CoT**
   For å få til dette kan det være nødvendig å sette sammen de ulike delene for tid og dato til en streng. Dette lar seg lett gjøre på følgende måte. Her har vi laget oss en funksjon som bygger opp

```
String makeDateAndTimeString()
{
  String timeAndDate;

  timeAndDate += mday;
  timeAndDate += ".";
  timeAndDate += mon;
  timeAndDate += ".";
  timeAndDate += year;
  timeAndDate += " ";
  timeAndDate += hour;
  timeAndDate += ":";
  timeAndDate += minu;
  timeAndDate += ":";
```

```
  timeAndDate += sec;

  return timeAndDate;
}
```

7. **Overføring til CoT**
   Det kan imidlertid se ut som om det ikke lar seg gjøre å overføre strenger via write-kommandoen til CoT

```
circusESP32.write(BMP280_hight_key,<String>,breddegrad,lengdegrad,
hoyde,token);
```

Så dermed kommer vi foreløpig ikke lengre.

6 Utdypende om viktige komponenter

6.1 Måling av lufttrykk med BMP180 og BMP280

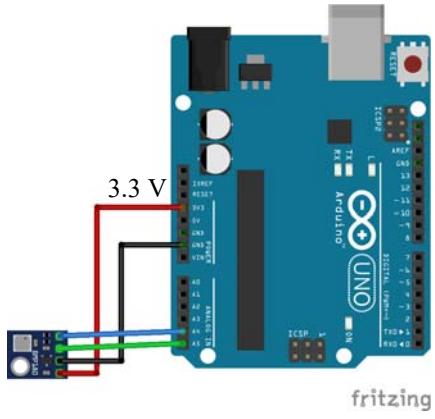
6.1.1 Måling av lufttrykk ved endring i resistans (piezo-resistivitet)

Den piezo-resistive effekten er forskjellig fra den piezo-elektriske effekten. Den piezo-resistive-effekten ble oppdaget av ***Lord Kelvin*** i 1856. Først i 1954 oppdaget C.G. Smith at germanium- og silisiumkrystaller hadde spesielt store variasjoner i ledningsevnen når de ble utsatt for mekanisk stress. Ledningsevnen til materialer er avhengig av mengden ladningsbærere i ledningsbåndet og hvor lett elektroner kan frigjøres fra valensbåndet. Dette er igjen avhengig av størrelsen på *båndgapet* mellom lednings- og valensbåndet i materialet. Når de nevnte materialene utsettes for stress, vil båndgapet endre seg og dermed også ledningsevnen.

6.1.2 Barometeret BMP180 (Bosch)

BMP180 er en trykksensor som går under betegnelsen *barometer*. Kretsen erstatter BMP085. Senere er også BMP280 kommet på markedet. Selve sensoren er basert på teknologi knyttet til piezo-resistivitet og levert av firmaet Robert Bosch. Kretsen leverer trykkdata via en digital I^2C buss. I tillegg til trykksensoren inneholder kretsen en temperatursensor. Her er noen nøkkeldata:

- Måleområde: 300 – 1100 hPa (millibar) – som typisk tilsvarer 9 000 til –500 m
- Oppløsning, trykk: 0,01 hPa (0,01 mbar)
- Oppløsning, høyde: 0,25 meter
- Temp. nøyaktighet (abs.) ± 1 °C (0 – 65 °C), ± 0,5 °C @ 25 °C
- Oppløsning i temp.: 0,1 °C
- Konverteringstid trykk: 7,5 ms (standard)
- Supplyspenning: 1,8 – 3,6 V
- Lavt effektforbruk: 5µA ved 1 måling pr. sek.
- Responstid: 7,5 ms (maks)
- Standby strøm: 5 µA
- Langtidsstabilitet: ± 1 hPa pr. 12 måneder.
- Absolutt nøyaktighet – 4,0 – + 2,0 hPa (mbar)



Kretsen leveres fra bl.a. Sparkfun og er montert på et kretskort for lettere å kunne kobles til f.eks. en Arduino (*"breakout board"*). Kretsen leveres også fra firmaet www.KultOgBillig.no og andre.

For å lese dataene via I^2C bussen brukes biblioteket: #include <wire.h> i tillegg til biblioteket som er laget til kretsen: `#include <SFE_BMP180.h>`

For mer informasjon se: https://www.sparkfun.com/tutorials/253. Her finner du også databladet og programvare for Arduino.

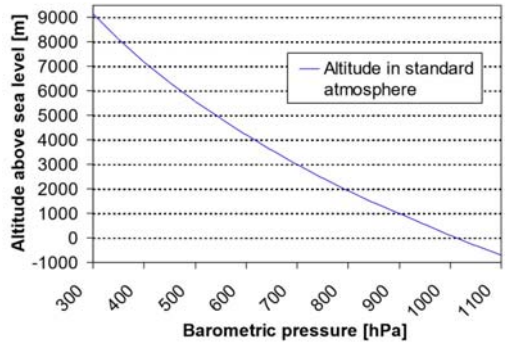
Biblioteket som følger med BMP180 beregner høyden på bakgrunn av trykkmålinger. Dvs. man ser bort fra temperaturgradienten som funksjon av høyden. Den beregnede høyden vil alltid være i forhold til høyden til stedet der referansetrykket er målt:

$$h = 44330 \cdot \left(1 - \left(\frac{p}{p_0} \right)^{\frac{1}{5{,}255}} \right) \tag{6.1}$$

Hvor:

h	Beregnet relativ høyde i forhold til referansehøyde ved p_0 (gjerne ved havnivå)
p	Målt trykk
p_0	Målt referansetrykk

Figuren under viser en typisk sammenheng mellom trykk og høyde.



For en mer generell behandling, se avsnitt 6.1.3, side 101.

Biblioteket til BMP180:

Biblioteket inneholder en rekke biblioteksfunksjoner som vil bli beskrevet her:

Inkludering av bibliotek og deklarasjon av trykksensor:

```
#include <SFE_BMP180.h>
SFE_BMP180 pressure;
```

Definer et *objekt* med et passende navn. Navnet velges av brukeren. Dersom man har flere like trykkmålere, velger man objekter med forskjellig navn for de to trykkmålerne. Her velger vi som eksempel: *pressure* som objektnavn.

```
pressure.begin ();
```

Initier kommunikasjon med objektet. Om kommunikasjonen med objektet lyktes, returneres 1 ellers 0 som gir mulighet for å generere en feilmelding.

status = pressure.startTemperature ();

Funksjonen initierer temperaturmåling. Dersom kommunikasjonen med objektet lyktes, returneres (til *status*) antall millisekunder enheten bør vente før temperaturen avleses. Om den mislykkes returneres verdien 0.

status = pressure.getTemperature (T);

Om status er forskjellig fra 0 så legges temperaturen i °C inn i variabelen T (argumentet til funksjonen).

status = pressure.startPressure(oversampling);

Argumentet *oversampling* angis med en verdi fra 0 til 3, og uttrykker graden av oversampling. En verdi på 3 vil gi større nøyaktighet, men tar lengre tid for å vente på svar. Variabelen *status* angir hvor lang tid en bør vente før en henter verdien for trykket. Returneres verdien 0 betyr det at det ikke er oppnådd kontakt med objektet.

status = pressure.getPressure(P,T);

Funksjonen *pressure.getPressure(P,T)* avleser både absolutt trykk (*P*) og temperatur (*T*). Trykket angis i millibar (mBar) og temperaturen i °C. Funksjonen returnerer status som angir om kontakt med sensoren er vellykket (returnerer delay før avlesning i ms) eller om kommunikasjonen mislykkes (returneres verdien 0).

p0 = pressure.sealevel(P,ALTITUDE);

Funksjonen *p0 = pressure.sealevel(P,ALTITUDE);* returnerer trykket ved havnivå (*p0*) på bakgrunn av kjent høyde over havet på målestedet (ALTITUDE). I forbindelse med værmelding er det vanlig å normalisere trykket til havnivå.

a = pressure.altitude(P,p0);

Biblioteket inneholder også en funksjon som beregner høyden over havet i meter (*a*) på bakgrunn av trykket målt på stedet i mbar (*P*) og trykket ved havnivået også i mBar.

Beskrivelsen av funksjonene er hentet fra eksempelet: SFE_BMP180_example av Mike Grusin, SparkFun Electronics 10/24/2013 V1.1.2 Updates for Arduino 1.6.4 5/2015 og ligger vedlagt biblioteket.

6.1.3 Måling av høyde basert på trykkmålinger

Det finnes flere ulike modeller for omregning fra trykk til høyde.

Vanlig brukt modell for omregning fra trykk til høyde i CanSat

Trykk måles normalt i Pascal hvor 1 Pa = 1 N/m^2.

Tidligere ble trykk målt i atmosfærer (atm), mmHg eller Bar.

En normalverdi for lufttrykket er:

1 atm = 760 mmHg = 1.01325 Bar = 1013.25 mBar = 101325 Pa = 1013.25 hPa

Vi legger merke til at h(ekto)Pa er det samme som m(illi)Bar og at 1 Bar er lik 100 000 Pa.

Lufttrykket er bestemt av tyngden til det "havet" av luft som vi befinner oss på bunnen av. Lufttrykket er derfor avhengig av mengden luft som til en hver tid befinner seg over hodet på oss. Vekta av luftmengden er avhengig av tyngdekraften, tykkelsen og tettheten til luftlaget, som igjen er avhengig av hvordan lufta forflytter seg og av temperaturen, dvs. værforholdene. Som vi ser er det mange faktorer å ta hensyn til. Likevel finnes det matematiske modeller som gjør at en kan gjøre rimelig nøyaktige høydemålinger på bakgrunn av trykkmålinger. Imidlertid er kalibrering særdeles viktig i denne sammenhengen

En regner normalt at trykket faller med 1 millibar pr. 8 meter, eller ca 12.5 millibar pr. 100 meter. Dette stemmer ikke så verst for de første 2000 meter, deretter minker trykket mindre for hver 1000 meter.

Målinger som refereres til på http://www.xgeo.no/index.html?p=klima angir lufttrykk på stedet, dvs. ikke korrigert mht. høyde over havet.

Tabellen under viser typiske verdier for sammenhengen mellom trykk, lufttetthet, temperatur og høyde over havet.

HoH	**Temperatur**	**Lufttrykk**	**Tetthet**	
(m)	*(C)*	*(hPa)*	*(kg/m^3)*	
0000	15.0	1013	1.2	
1000	8.5	900	1.1	
2000	2.0	800	1.0	(Galdhøpiggen)
3000	-4.5	700	0.91	
4000	-11.0	620	0.82	
5000	-17.5	540	0.74	
6000	-24.0	470	0.66	
7000	-30.5	410	0.59	
8000	-37.0	360	0.53	
9000	-43.5	310	0.47	(Mount Everest)
10000	-50.0	260	0.41	(Marsjhøyde rutefly)
11000	-56.5	230	0.36	
12000	-56.5	190	0.31	
13000	-56.5	170	0.27	
14000	-56.5	140	0.23	
15000	-56.5	120	0.19	
16000	-56.5	100	0.17	
17000	-56.5	90	0.14	
18000	-56.5	75	0.12	
19000	-56.5	65	0.10	
20000	-56.5	55	0.088	
21000	-55.5	47	0.075	
22000	-54.5	40	0.064	
23000	-53.5	34	0.054	
24000	-52.5	29	0.046	
25000	-51.5	25	0.039	
26000	-50.5	22	0.034	
27000	-49.5	18	0.029	
28000	-48.5	16	0.025	
29000	-47.5	14	0.021	
30000	-46.5	12	0.018	
31000	-45.5	10	0.015	

32000	-44.5	8.7	0.013
33000	-41.7	7.5	0.011
34000	-38.9	6.5	0.0096
35000	-36.1	5.6	0.0082

Omregningen fra trykk til høyde bør også ta hensyn til temperaturen, og temperaturen vil normalt forandre seg med høyden.

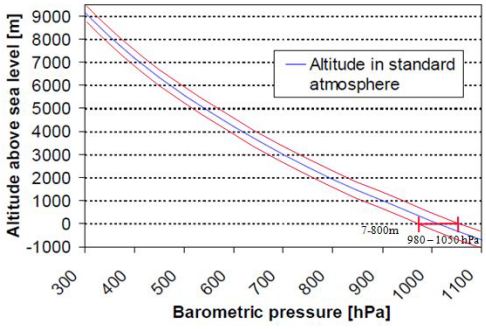
I programmer er det normalt lettere å forholde seg til en omregningsformel enn en tabell. Ulempen med en formel er at de mange parametrene kan gi stor usikkerhet i beregningene. Følgende formel er vanlig å bruke:

$$h = \frac{T_1}{a}\left(\left(\frac{p}{p_1}\right)^{-\frac{aR}{g_0}} - 1\right) + h_1 \qquad (6.2)$$

Hvor:

h	Beregnet høyde i meter
h_1	Starthøyde i meter
T	Temperatur i Kelvin
T_1	Starttemperatur i høyden h_1
a	Temperaturgradient, foreslått verdi -0,0065 K/m
p	Målt trykk i Pa
p_1	Trykk i Pa ved starthøyden
g_0	Tyngdeakselerasjonen 9,81 m/s^2
R	Den spesifikke gasskonstant 287,06 J/kg K

Denne formelen kan enten legges inn i datainnsamlingsenheten i f.eks. CanSat, men bedre i programvaren som behandler data. Har man rådataene fra sonden, har en større mulighet til etterbehandling enn om man bare har de omregnede dataene.



Diagrammet til høyre viser sammenhengen mellom trykk og høyde med økende høyde over havnivået. Normale variasjoner i luft-trykket ved havnivået kan være fra 980 hPa til 1050 hPa (millibar). Denne naturlige variasjonen kan derfor gi en absolutt endring i høydeberegningen på 7 – 800 meter dersom man ikke kalibrerer målingene.

Vi skjønner derfor at det er svært viktig med hyppig kalibrering dersom man skal kunne stole på høydemålerens absolutte høydeverdier.

Alternative beregningsmodeller

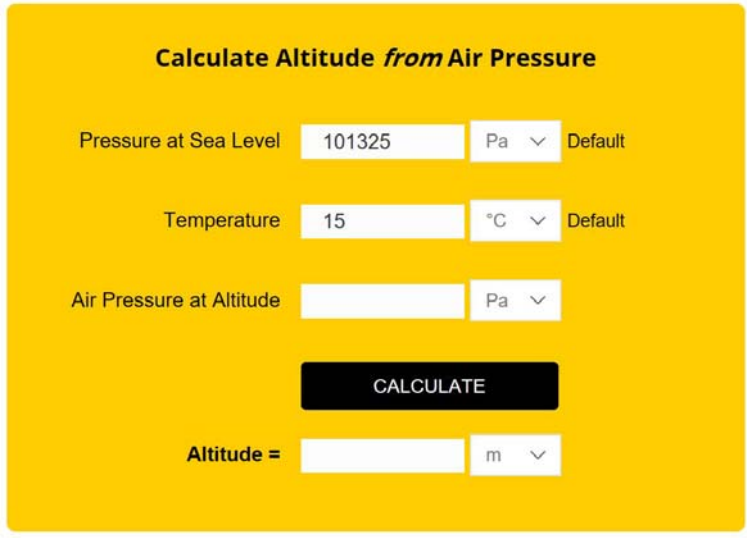
Det finnes imidlertid flere ulike modeller for sammenhengen mellom trykk og høyde. Her er en alternativ[31] modell hentet fra en teknisk note publisert av firmaet Vaisala som utvikler instrumenter for værobservasjoner og værstasjoner.

$$h = \left(\frac{R}{g}\right) T_m \ln\left(\frac{p_0}{p}\right) \tag{6.3}$$

Hvor:

h	Beregnet høyde over havet
R	Den spesifikke gasskonstant 287,06 J/kg K
g	Tyngdeakselerasjonen 9,81 m/s^2
T_m	Gjennomsnittlig lufttemperatur i det aktuelle måleområdet
p_0	Lufttrykk ved havnivået
p	Målt lufttrykk på det aktuelle stedet

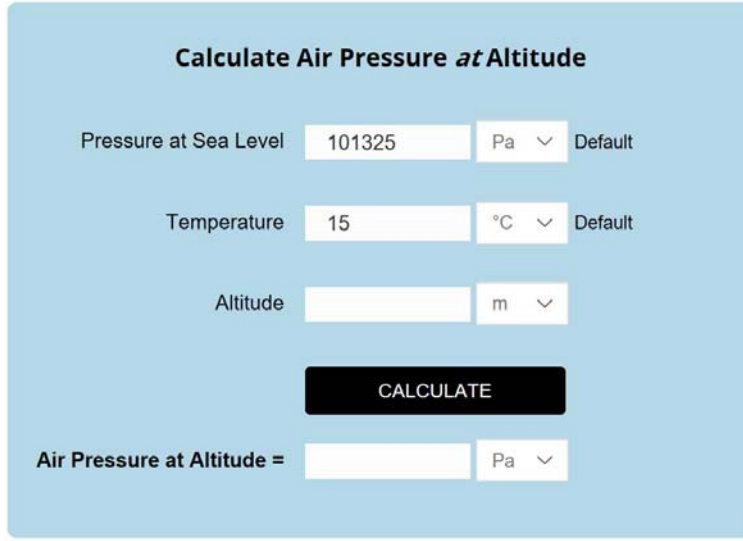
Det finnes også omregningskalkulatorer på nett. Firmaet MIDE Engeneering Solutions har en på nettsiden: https://www.mide.com/pages/air-pressure-at-altitude-calculator.



Her skriver man inn trykk og temperatur ved havnivå (ferskvare) i tillegg til målt trykk på den lokasjonen der man befinner seg, og man får et estimat av høyden på det stedet man er. Det er viktig at man bruker en referansehøyde ved havnivå som er i nærheten av stedet der man befinner seg.

31.http://www.vaisala.fi/Vaisala%20Documents/Measurement%20Theory/height_calculation.pdf

Tilsvarende kan man finne et estimat av trykket der man befinner seg dersom man kjenner trykket og temperaturen ved havnivået og hvor høyt man befinner seg.



6.1.4 Målinger utført med BMP180

Her skal vi vise to enkle forsøk utført med Arduino og BMP180.

Registrering av små endringer i lufttrykk med BMP180

For å illustrere hvor følsom BMP180 er så har vi gjort følgende enkle forsøk:

Vi har latt Arduino'en med BMP180 måle trykket i rommet og latt programmet skrive trykket til monitoren med følgende enkle programkode. Det er viktig at målingene gjøres så raskt som mulig. Her satt til ca. hvert 10 ms. Vi husker dessuten at responstiden til BMP180 er 7,5 ms maks:

```
Serial.print(i);
  Serial.print(", ");
  Serial.println(P, 2);
  i = i+1;
  delay(10);
```

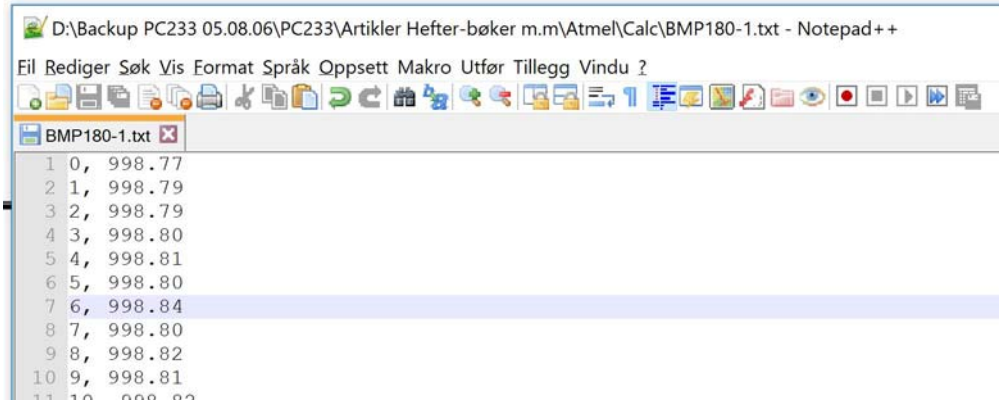
På denne måten vil vi få en lang rekke trykkmålinger skrevet ut i monitoren omtrent som utsnittet under:

```
0, 998.77
1, 998.79
2, 998.79
3, 998.80
```

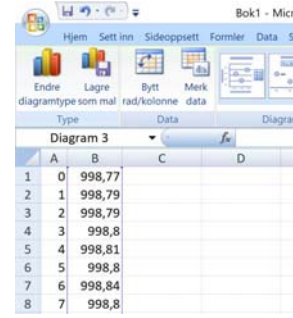
```
4, 998.81
5, 998.80
6, 998.84
...
```

I alt tok vi nær 500 målinger av trykket i løpet av ca. 30 sek. Til venstre for trykket er en telle-variabel som forteller oss nummeret til målingen. Vi må også passe på å legge inn et skilletegn mellom de to tallene på samme linje. Her har vi valgt komma, men et semikolon kan være bedre.

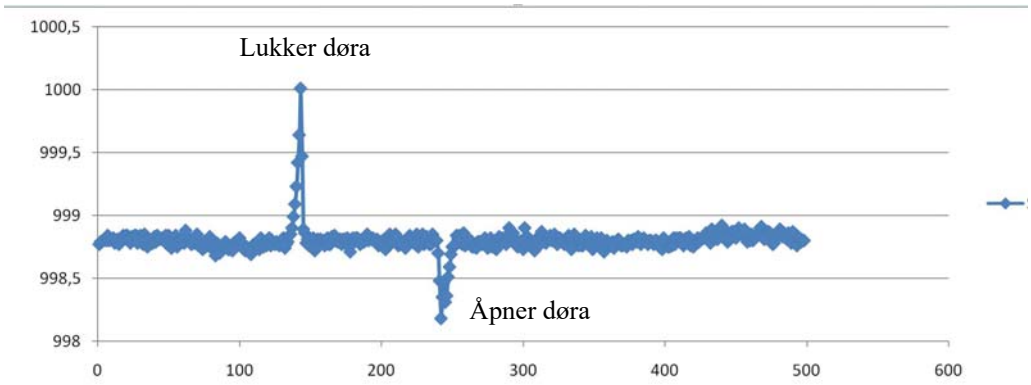
Vi kopierte tallrekken med programmet NotePad som vist på figuren under. Det fine med NotePad er at den kopierer data uten å trekke noe fra eller legge noe til.



Dernest lagret vi dataene i en tekst-fil som vi f.eks. kan importere til Excel. Dataene blir liggende som to kolonner som vist under. Her har vi sørget for at desimaltegnet er et komma, som er viktig for at Excel skal oppfatte verdiene som tall og ikke som tekst.



Dermed kan vi plotte dataene som funksjon av tiden (eller sample nummeret). På figuren under ser vi hvordan trykket endrer seg med tiden. Vi legger også merke til to topper, en som rager opp over gjennomsnittet og en som stikker ned under gjennomsnittet. Disse to avvikene skyldes at vi lukket døra til rommet for så å åpne den igjen. Ser vi nøye på grafen vil vi oppdage at utsvingene er relativt beskjedene men rager godt opp over støyen.



Det må innrømmes at forsøket ble gjort i et relativt lite rom og med kraftige bevegelser av døra.

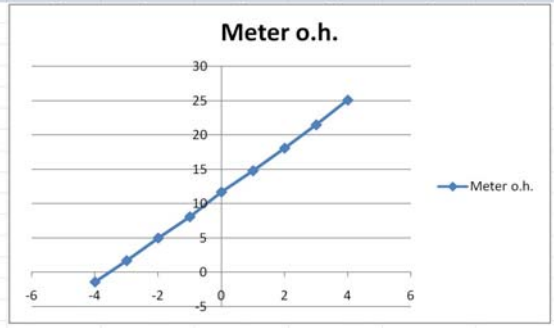
Innendørs målinger av trykk i ulike etasjer

Disse målingene er gjort innendørs i Realfagbygget ved NTNU. Dette bygget har 9 etasjer hvorav fire er under bakken. Det er foretatt ni målinger ved samme høyde i hver av de ni etasjene. Disse er lagt inn i et regneark.

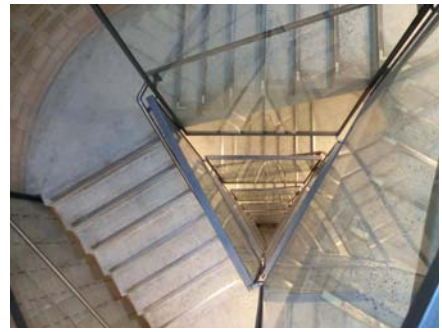
Figuren under viser resultatene av måleserien, dvs. sammenhengen mellom etasje og relativ høyde. Vi ser at den relative nøyaktigheten er rimelig god, men den absolutte er feil, da det kan se ut til at underetasje U4 ligger under havets overflate. Dette skyldes hovedsakelig at utrustningen ikke er godt nok kalibrert. For å få en mest mulig stabil måling har vi midlet lufttrykket over 100 enkeltmålinger. Det vil si at hver måling tar ca. 1 – 2 sekunder. Likevel ser vi at målingene

varierer med 50 – 60 cm fra måling til måling ved samme høyde. Legg også merke til at vi har latt første etasje være etasje 0 slik at "x-aksen" blir riktig, dette problemet slipper man f.eks. i England.

Etasje markør	Meter o.h.	Etasje
-4	-1,4	U4
-3	1,7	U3
-2	5	U2
-1	8,1	U1
0	11,7	1
1	14,8	2
2	18,1	3
3	21,5	4
4	25,1	5



Bildet til høyre viser trappesjakta der målingene ble gjort. Det ble også foretatt målinger med laser. Denne målingen gikk fra målehøyden i øverste etasje og ned til gulvet i U4, og viste en total høyde på 31.2 meter. Dersom vi legger til høyden fra målepunktet og ned til gulvet i U4 i våre målinger basert på trykk, får vi en total høyde på 26,3 m som er et betydelig avvik fra lasermålingen. Her er det rom for diskusjon og videre utforskning for å forklare avviket. Skyldes det luftrykksmålingen eller lasermålingen? Kanskje kontrollmålingen burde ha vært gjort med målebånd.



6.2 Barometer, temperatur- og fuktighetsmåler– BME280 (Bosch)

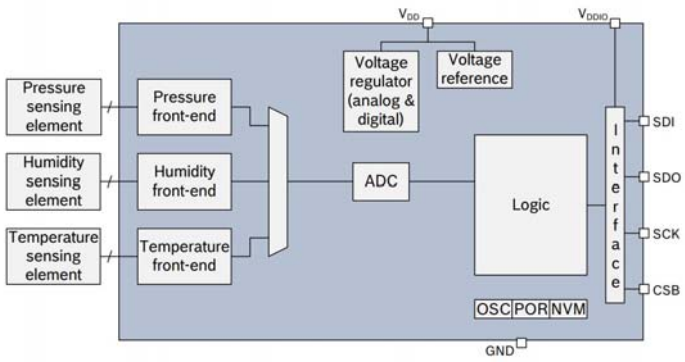
Dette er en utvidelse av BMP280 hvor man også har inkludert fuktighetsmåling. Dette kvalifiserer tydeligvis for å bytte ut P (Pressure) med E (Environment). Hvilket gir løfter om en sensor som i større grad en BMP-serien gir er mer fullstendig "bilde" av miljøet. Sensoren er spesielt beregnet for mobile anvendelser med sitt lave strømforbruk. Her er noen nøkkeldata[32]:

- *Spenning og energiforbruk:*
  Supplyspenning: 1,7 – 3,6 V
  Strømforbruk standby: 0,1 µA
  Strømforbruk med måling 1 x pr. sek.: 3,6 µA (H, P, T)
- *Grensesnitt:*
  I^2C eller SPI[33]

32. Informasjonen er hentet fra: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf

- *Luftfuktighetssensor:*
  Responstid: 1 sek.
  Nøyaktighet: ± 3% mellom 20 – 80% relativ fuktighet.
- *Lufttrykksensor:*
  Måleområde trykksensor: 300 – 1100 hPa
  Relativ nøyaktighet: ± 0,12hPa
  Absolutt nøyaktighet: ± 1hPa (0 – 65°C)
- *Temperatursensor:*
  Måleområde: -40 – +65°C
  Nøyaktighet: ± 1°C (0 – 65°C)

Figuren viser et blokkdiagram over sensoren:



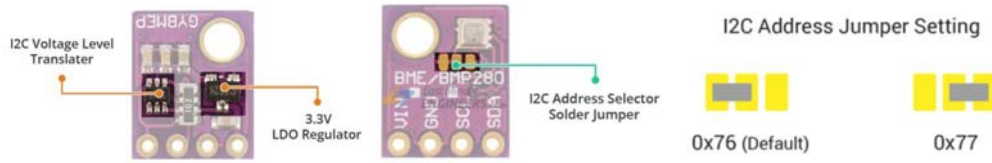
Vi legger merke til at det er en felles ADC for alle tre sensorene og at power supply for sensordelen og logikken er separert, hvilket er gunstig mht å redusere støy.

Oppkobling

Figuren under viser pinningen til to utgaver til BME280:

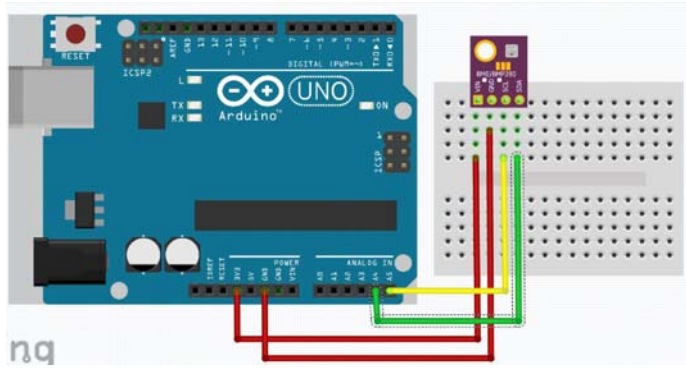
33. Billige versjoner kommer ofte med bare en av bussene, dvs. at fabrikanten av kortet har bestemt hvilken kommunikasjonsbuss som skal tilbys brukeren. Mens andre leverandører gir brukeren mulighet til å velge hvilke grensesnitt hen ønsker å bruke.

En viktig forskjell er at hos varianten til høyre kan man sette to ulike adresser i "adressefeltet". Default adresse er 0x76 HEX, men den kan endres til 0x76 HEX om man forbinder to pad'er som vist helt til høyre på figuren under. Dessuten har denne varianten en innebygget regulator slik at den kan arbeide på spenninger fra 3,3 – 5,0 V.



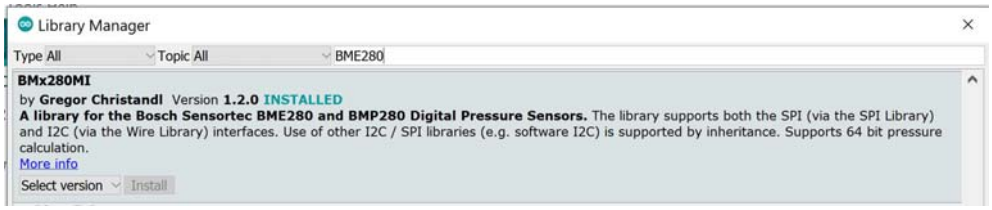
Spenningsregulator Adresse-setting

Figuren under viser oppkoblingen.



Programmering av BME280:

- **Hent biblioteket**: Biblioteket for BME280 finner man lett ved å bruke *Manage Libraries* under *Sketch/Include Library*. Skriv BME280 i søkefeltet og man får opp flere alternative biblioteker. Scroll ned til du finner:



  Velg å installer denne. De fleste andre vil også funne fungere godt[34]. Denne har den egenskapen at det kan fungere for både BMP280 og BME280, derfor kalles den BMx280. En kan også bestemme hvilken av de to som er tilkoblet.

- **Legg inn bibliotekene** i programmet. Disse legges inn helt først:

34.I forbindelse med oppdrag 1 så har vi valgt et annet bibliotek fra Adafruit. Dette skyldes tilfeldigheter.

```
#include <Wire.h>      // Inkluder bibliotek for kommunikasjon via I2C
#include <BMx280I2C.h> // Inkluder biblioteket for måling av lufttrykk,
                       // temperatur og luftfuktighet
```

- **Deklarasjon** av objektet `bmx280` av typen (klassen) `BMx280I2C`:

```
BMx280I2C bmx280(0x76);// Den gis navnet "bmx280" og er av typen
                       // BMx280I2C. Den må også inkludere adressen: 0x76
```

Denne deklarasjonen legges inn i starten før setup()-funksjonen

- **Initialiser** måleren `bmx280`

```
bmx280.begin();    // Initialiser BME280-sensoren
Wire.begin();      // Også wire må initialiseres for å betjene I2C-bussen
```

Man kan også resette sensoren til default settinger:

```
bmx280.resetToDefaults();
```

Dernest settes en oversamlingsrate for lufttrykks-, temperatur- og luftfuktighetsmålinger. Her velges en oversampling på 16 ganger.

```
bmx280.writeOversamplingPressure(BMx280MI::OSRS_P_x16);
bmx280.writeOversamplingTemperature(BMx280MI::OSRS_T_x16);
bmx280.writeOversamplingHumidity(BMx280MI::OSRS_H_x16);
```

Dersom man ønsker at programmet selv skal kunne skille mellom BMP280 og BMx280 så legger mann inn følgende test for å bestemme om man skal sette oversamplingsraten til luftfuktighet eller ikke:

```
if (bmx280.isBME280())
bmx280.writeOversamplingHumidity(BMx280MI::OSRS_H_x16);
```

Man undersøker hvilken krets det er snakk om ved å legge inn en if-setning med `bmx280.isBME280()`. Om det er en BME så settes oversamlingsraten også for luftfuktigheten.

Alle disse legges inn i setup()-funksjonen.

- **Start måling**

Man starter en måling med kommandoen:

```
bmx280.measure();
```

Denne returnerer en verdi lik "0" dersom den ikke klarer å starte måling. Dersom man ønsker å få et varsel om at målingen mislyktes kan man istedet bruke følgende kommando:

```
if(!bmx280.measure())
{
 Serial.println("could not start measurement, is a measurement already
running?");
 return;
}
```

- **Innhenting av måleverdier**

Verdiene leses fra de ulike sensorene med følgende kommando:

```
bmx280.hasValue();
```

Siden det kan ta litt tid kan det være lurt å legge denne inn i en venteløkke til verdiene er klare:

```
do {
   delay(100);
   }
while (!bmx280.hasValue());
```

Denne går i ventemodus så lenge `bmx280.hasValue()` returnerer "0".

Dernest kan verdiene leses ut for de ulike parameterne. Verdiene for lufttrykk, temperatur og lufttrykk leses slik.

```
float lufttrykk = bmx280.getPressure();
float temperatur = bmx280.getTemperature();
float luftfuktighet = bmx280.getHumidity();
```

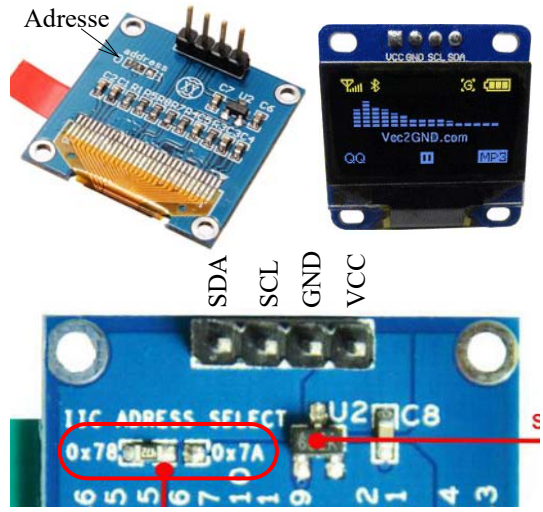
Avlesningen finner vi som regel i loop()-funksjonen.

- **Andre nyttige funksjoner**

```
bmx280.isBME280();
```

Er en funksjon som gjenkjenner hvilken krets som er tilkoblet. `bmx280.isBME280();` returnerer "1" når vi har tilkoblet en BME280 og "0" når det er en BMP280 som er koblet til.

6.3 Kort om det grafiske OLED displayet SSD1306

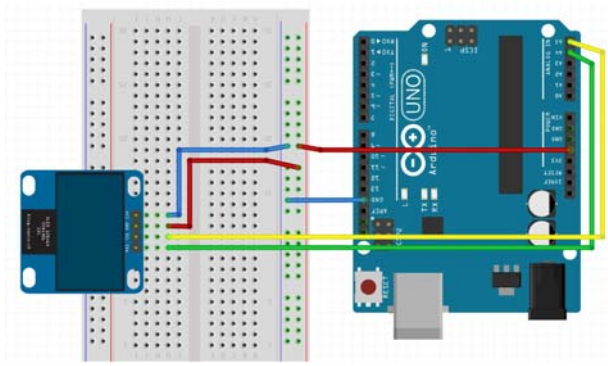


Displayet er relativt lite og billig med 128 x 64 piksler og kan typisk vise tre – fem linjer med kort tekst, avhengig av skriftstørrelsen. Ved å gå ned på fontstørrelsen kan det vises opp til 5 linjer. Det egner seg godt for å skrive ut måleverdier og kan arbeide med fra 3 – 5 V og programmeres via I^2C-buss og et bibliotek som hentes fra nett. Det leveres vanligvis med en stiftlist med fire kontakter Vcc (5V), GND, SDA (seriell data), SCL (seriell klokke). Og egner seg godt for å kobles til I^2C-bussen til f.eks. Arduino UNO (se figur under) eller ESP32. Default adresse er 0x3C (hex), men kan omprogrammeres til 0x3D ved å flytte strapen mot høyre (se bilde til høyre). En legger merke til at merkingen av kretsen synes å antyde at de to adressene er 0x78 eller 0x7A. Dersom man er i tvil hva som er riktig adresse, kan man bruke en I^2C scanner programvare som kjøres i mikrokontrolleren, mens displayet er tilkoblet I^2C-bussen. På denne måten leser man av adressen til alle enheter som er tilkoblet bussen og viser adressen i monitoren.

Displayet kan leveres som en-farget eller to-farget. Sistnevnte betyr at øvre og nedre del av displayet vises i forskjellige farger (se figuren til høyre over), dvs. at fargene er gitt av hårdvaren.

Figuren til høyre viser hvordan vi kan koble displayet opp til en Arduino UNO-krets med kun to linjer (SDA og SCL) i tillegg til spenningsforsyning (5V) og jord (GND). Tilsvarende kan vi koble oss mot ESP32.

Vi bruker litt forskjellig biblioteker når vi bruker displayet mot Arduino eller ESP32.



6.4 Bruk av GPS - NEO-6M

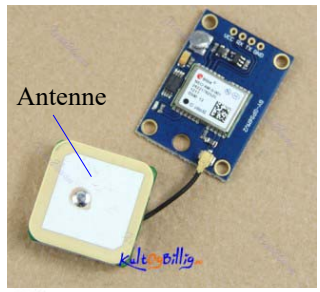
6.4.1 Bakgrunn

GPS eller *Global Positioning System* består av 24 satellitter som kretser omkring jorda med en omløpstid på 11 t 58 min. i en høyde av ca. 20200 km over jordoverflata. Normalt vil dette antallet være tilstrekkelig for, til enhver tid, å ha fri sikt til 8 – 10 satellitter i åpent terreng. Hver satellitt sender ut et kodet tidssignal som mottas av mottakerne. I tillegg til å inneholde informasjon om nøyaktig tid, inneholder signalet tidspunkt for utsendelse og en lang kode som mottakerne bruker for nøyaktig å bestemme tidspunktet for mottatt signal. På denne måten kan mottakersystemet bestemme tiden hvert av signalene bruker fra hver av satellittene og til mottakeren. De målte tidsforsinkelsene brukes så til å beregne posisjonen til mottakeren.

En ***GPS-modul*** er et sett av GPS mottakere som gjør det mulig å følge flere GPS-satellitter samtidig. Seks er ikke uvanlig. 4 er et minimum for å kunne beregne koordinater pluss høyde. Normalt vil GPS være mere nøyaktig på lengde- og breddegrad enn i høyden. For en mer nøyaktig høydemåling vil en kalibrert barometer være bedre.

6.4.2 GY-NEO-6MV2[35] Flight (GPS-modul)

Teknisk - spesifikasjon



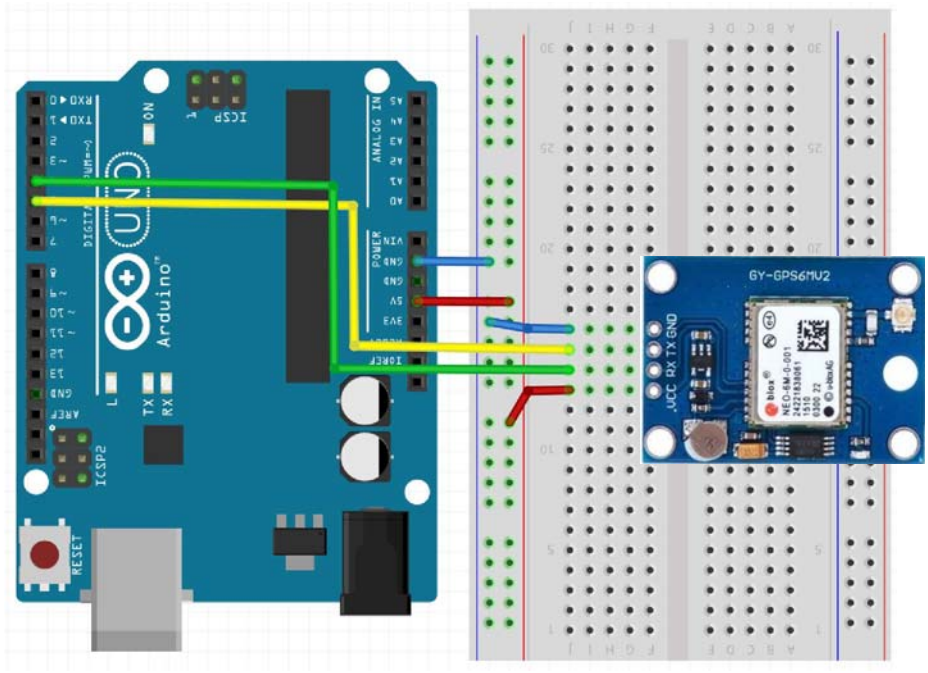
Dette er en vanlig brukt GPS-modul. Den kan bl.a. kjøpes fra www.kultogbillig.no for kr. 119,- inkl. MVA, eller fra AliExpress for ca. kr. 35,00 (U$ 4,0)[36]. Den består av to separate enheter forbundet med en koaksialkabel. Det ene er antennemodulen, den andre analyse- og datamodulen.

- Enheten er bygget opp rundt chip-settet: ARM
- Mottakerfrekvens: 1575,42 MHz
- Antall kanaler: 50 kanaler
- Følsomhet: -161dBm ("Tracking & Navigation)
- Nøyaktighet: - Horisontal: 2,5 m
  - Hastighet: 0,1 m/s
  - Kurs retning: 0,5°
  - Tid: 30 ns (tidspulssignalets nøyaktighet)
- Dato: Leverer data og tid ned til sekunder
- Låsetid: Varm (Hot) start: 1 sek. i gjennomsnitt
  Kald start: 27 sek. i gjennomsnitt
- Maks verdier: Maks. målehøyde: 50 000 meter
  Maks. hastighet 500 m/s
  Maks. akselerasjon 4 g
  Maks. oppdatering 5 Hz
- Forsyning: Spenning: 2,7 – 3,6 V DC
  Strømforbruk: Max. 47 mA
- Dig. utgang: Spenningsnivå: TTL nivå (0 – 2,85 V)
  UART, USB (12 Mbit/sek), SPI (100 kbit/sek)
  Baud rate: 9 600 baud (symboler/sek.)
  Format: NMEA GSV, RMC, GSA, GGA, GLL, VTG, TXT
- Størrelse: 25 x 25 x 8 mm (antennemodul) +
  25 x 35 x 5 mm (kontrollmodul)
- Temperaturområde: –40°C til 85°C
- Backup batteri: For å holde data ved "Power down"

35. Datablad: https://www.openimpulse.com/blog/wp-content/uploads/wpsc/downloadables/GY-NEO6MV2-GPS-Module-Datasheet.pdf
36. https://www.aliexpress.com/

6.4.3 Oppkobling

Figuren under viser oppkoblingen som er ganske enkel.



6.4.4 Programmering:

- **Installasjon av biblioteker:** Vi tenger to biblioteker for å bruke GPS-mottakeren NEO-6MV2 slik vi monterer den. Siden vi ønsker å bruke to ordinære digitale porter for seriekommunikasjon (UART) så trenger vi biblioteket: #include <SoftwareSerial.h> til dette formålet. Dette er standard i Arduino og trenger ikke installeres, men må inkluderes i programmet. I tillegg trenger vi det spesielle biblioteket for NEO6MV2 som kan lestes ned fra https://github.com/mikalhart/TinyGPS/releases/tag/v13. Man kan også bruke Library manager hos Arduino og søke etter NEO6M.
- **Inkluder bibliotekene:** Følgende biblioteker inkluderes i programfilen:
  ```
  #include <SoftwareSerial.h>
  #include <TinyGPS++.h>
  ```
  Disse plasseres helt i starten av programfila før setup()-funksjonen
- **Deklarer** en instans `gpsCom` (et navn vi velger) av typene (klassen) `SoftwareSerial` og en instans `gps` av typen `TinyGPSPlus`:
  ```
  #define GPS_RX 5 //RX på GPS-modulen kobles til port 5
  #define GPS_TX 4 //XX på GPS-modulen kobles til port 4
  SoftwareSerial gpsCom(GPS_RX, GPS_TX); // Vi gir den navnet gpsCom
  ```

```
TinyGPSPlus gps; // Vi gir den navnet gps
```
Vi ser at `SoftwareSerial` trenger å vite hvilke porter som skal brukes til kommunikasjonen, `GPS_RX` og `GPS_TX` som settes inn i argumentet til funksjonen.

- **Initalisering:** Dernest må vi initalisere gpsCom og fortelle hvilken datarate vi ønsker å bruke:
  ```
  gpsCom.begin(9600);
  ```
  Denne kommandoen skal stå i setup()-funksjonen.

- **Lesing av data:** Det er ikke helt rett fram å lese data fra GPS-modulen. De følgende kommandoene skal plasseres i loop()-funksjonen. Man starter med:
  ```
  gpsCom.listen(); // Åpne for lesing fra denne kanalen
  ```
  Denne kommandoen iverksetter lytting på akkurat denne serieporten. Man kan ha flere slike UART-kanaler, men man kan bare lytte til en av gangen, Vi har bare en.
  Dernest må vi se etter om det er noen data tilgjengelig:
  ```
  gpsCom.available(); // Sjekk om det er tilgjengelige data
  ```
  Dernest leses data fra GPS-enheten
  ```
  gps.encode(gpsCom.read());
  ```
  Så undersøker man om dataene er gyldige og om de er oppdaterte. De kan godt være gyldige, men ennå ikke oppdatert i forhold til forrige gang de ble lest:
  ```
  bool gpsValid     = gps.location.isValid(); // Sjekk gyldighet
  bool gpsUpdated   = gps.location.isUpdated(); // Sjekk om de er nye
  ```
  Når dataene er gyldige og oppdaterte kan de skrives ut:
  ```
  Serial.print(long(millis()/1000));   // Skriv ut sekunder
  Serial.print(",");
  Serial.print(gps.location.lat(), 6); // Breddegrader i desimalgrader
  Serial.print(",");
  Serial.print(gps.location.lng(), 6); // Lengdegrader i desimalgrader
  Serial.print(",");
  Serial.print(gps.altitude.meters(), 1); // Høyde i meter, 1 desimaler
  Serial.println();
  ```
  Legg merke til at vi velger å skrive ut sekunder i første posisjon, deretter bredde- og lengdegrad og til slutt høyden over havet. Dersom man vil sjekke om dataene ser fornuftige ut kan man bare kopiere bredde og lengdegrad og legge dem inn i Google Earth så skal man forflytte seg til det aktuelle stedet.

  Dersom man ønsker å kopiere dataene inn i en fil for å skrive ut traseen så bør man velge følgende format på utskriften
  ```
  Serial.print(gps.location.lng(), 6); // Lengdegrader i desimalgrader
  Serial.print(",");
  Serial.print(gps.location.lat(), 6); // Breddegrader i desimalgrader
  Serial.print(",");
  Serial.print(gps.altitude.meters(), 1); // Høyde i meter, 1 desimaler
  Serial.println();
  ```
  Vi skal i neste avsnitt se hvordan vi kan bruke denne for å plotte traseen.

I forbindelse med bruk av OpenLog datalogger (SD-kort skriver) så vil det være aktuelt å skrive til file med dette formatet.

Etter innsamling av data kan disse lastes opp i f.eks. Notepad++ eller i Excel. Dersom man ønsker å visualisere traseen i Google Earth så klippes dataene inn i kml-koden vist i avsnitt 6.4.5, side 117.

Kode med feilsjekk og flere data

Den enkle programkoden omtalt inneholder ingen sjekk varsling mht. om dataoverføringen mellom GPS-enheten og Arduino er korrekt.

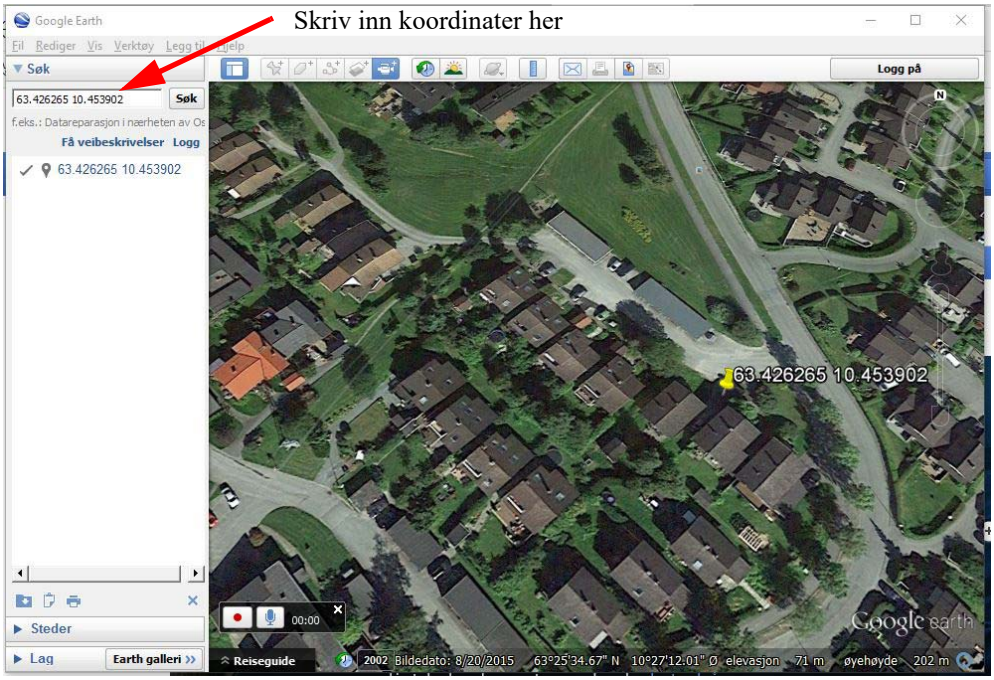
6.4.5 Visualisering av GPS-data i Google Earth

Data fra GPS-moduler krever gjerne en litt annen behandling enn ved hjelp av regneark. Slike data kan f.eks. plottes i Google Earth. I dette avsnittet skal vi se hvordan vi kan hente inn de utskrevne eller lagrede dataene og plotte dem i Google Earth.

Installasjon av Google Earth

Google Earth kan lastes ned og installeres fra følgende adresse: https://www.google.com/earth/

Skjermbildet under viser brukergrensesnittet for Google Earth.



I øverste venstre hjørne finnes et søkefelt. Her kan man enten skrive en adresse, eller man kan skrive inn koordinater (med desimaler). I dette tilfellet har jeg skrevet inn:

Breddegrad: 63.426265
Lengdegrad: 10.453902

Da vil Google Earth zoome inn akkurat til disse koordinatene som i dette tilfellet var rett utenfor veggen der jeg sitter. Dvs. det kan se ut som den bommer med et par meter (ringen angir plasseringen av mottakeren).

Høydeangivelsen for det angitte stedet varier mellom 60 – 80 meter over havet (Google Earth angir en høyde som varierer mellom 65 – 75 meter).

Plotting av en trase i Google Earth

Dersom man ønsker å plotte en trase i Google Earth kan man benyttet et programmeringsspråk kalt "Keyhole Markup Language" (KML). Dette er et "markup language" utviklet for visualisering av to- og tredimensjonale strukturer knyttet til kartdata. Språket ble utviklet i forbindelse med etableringen av Google Earth som ble lansert i 2004. I 2008 ble KML godkjent som en internasjonal standard for denne type geografisk visualisering.

Siden språket er temmelig omfattende og vi kun trenger å bruke en beskjeden del av det, så benytter vi en ferdige programkode og klipper inn våre data for lengde-, breddegrad og høyde. Disse legges inn som en liste med data i kml-koden (se under), før kml-fila lagres med et ønsket navn.

Koordinater og høyde data legges inn som vist under:

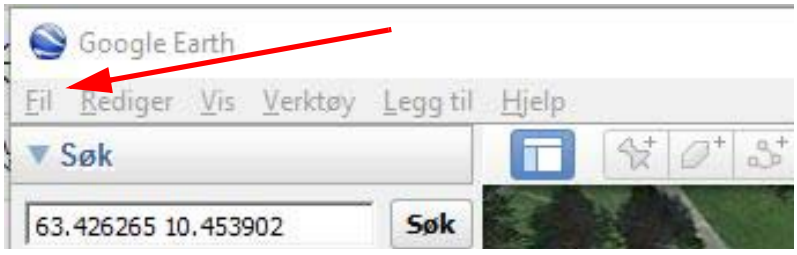
-112.2550785337791,36.07954952145647,2357

Lengdegrader [°],Breddegrader [°],Høyde [m]

Programkode skrevet i kml (legg merke til at et sett med eksempelkoordinater og høyde er klippet inn).

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <name>Paths</name>
    <description>Examples of paths. Note that the tessellate tag is by default
      set to 0. If you want to create tessellated lines, they must be authored
      (or edited) directly in KML.</description>
    <Style id="yellowLineGreenPoly">
      <LineStyle>
        <color>7f00ffff</color>
        <width>4</width>
      </LineStyle>
      <PolyStyle>
        <color>7f00ff00</color>
      </PolyStyle>
    </Style>
    <Placemark>
      <name>Absolute Extruded</name>
```

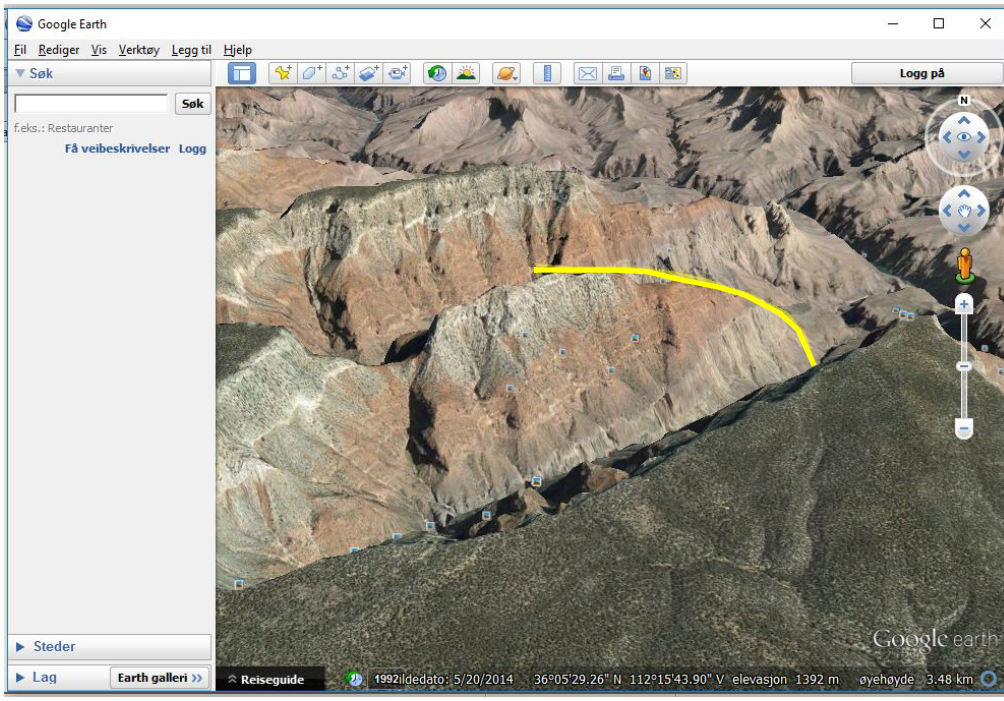
```
    <description>Transparent green wall with yellow outlines</description>
    <styleUrl>#yellowLineGreenPoly</styleUrl>
    <LineString>
      <extrude>0</extrude>
      <tessellate>0</tessellate>
      <altitudeMode>absolute</altitudeMode>
      <coordinates>
        -112.2550785337791,36.07954952145647,2357
        -112.2549277039738,36.08117083492122,2357
        -112.2552505069063,36.08260761307279,2357
        -112.2564540158376,36.08395660588506,2357
        -112.2580238976449,36.08511401044813,2357
        -112.2595218489022,36.08584355239394,2357
        -112.2608216347552,36.08612634548589,2357
        -112.262073428656,36.08626019085147,2357
        -112.2633204928495,36.08621519860091,2357
        -112.2644963846444,36.08627897945274,2357
        -112.2656969554589,36.08649599090644,2357
      </coordinates>
    </LineString>
  </Placemark>
 </Document>
</kml>
```



De koordinatene som pr. i dag ligger i eksempelet angir en helikopterflyvning over Grand Canyon i Colorado, USA. Eksempeldataene byttes ut med de aktuelle dataene og kodefila lagres under et ønsket filnavn som ender med .kml.

Filen hentes opp i Google Earth ved å velge *Fil* og *Åpne* for å laste opp den filen hvor koden og dataene ligger.

En vil da få tegnet inn traseen som angitt av lista med koordinater og vist på riktig sted. Eksempelet under viser traseen til helikopteret over Grand Canyon (gul linje).



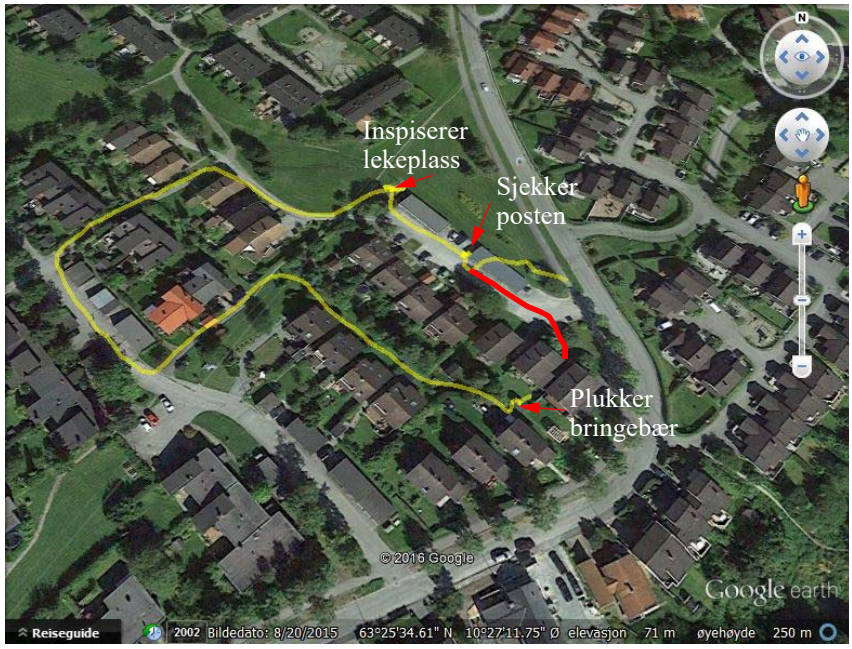
Dersom man ønsker å vise hvor man har gått en tur så kan det være upraktisk å måtte vise absolutt høyde. Høydemålinger kan ha store avvik slik at en kan oppleve at traseen går mange meter over eller forsvinner under bakken til tross for at man hadde begge beina på jorda. I slike situasjoner kan det være greit å bytte ut kommandoen:

<altitudeMode>absolute</altitudeMode>

med kommandoen:

<altitudeMode>clampToGround</altitudeMode>

Dermed vil kurven følge bakken som vist på traseen på bildet under.



Vi legger imidlertid merke til at mottakeren har problemer i starten. I dette området er avviket stort før den tar seg inn. Den røde kurven angir den riktige ruta. Deretter er den svært nøyaktig. Posisjonsdata samples hvert 3. sekund.

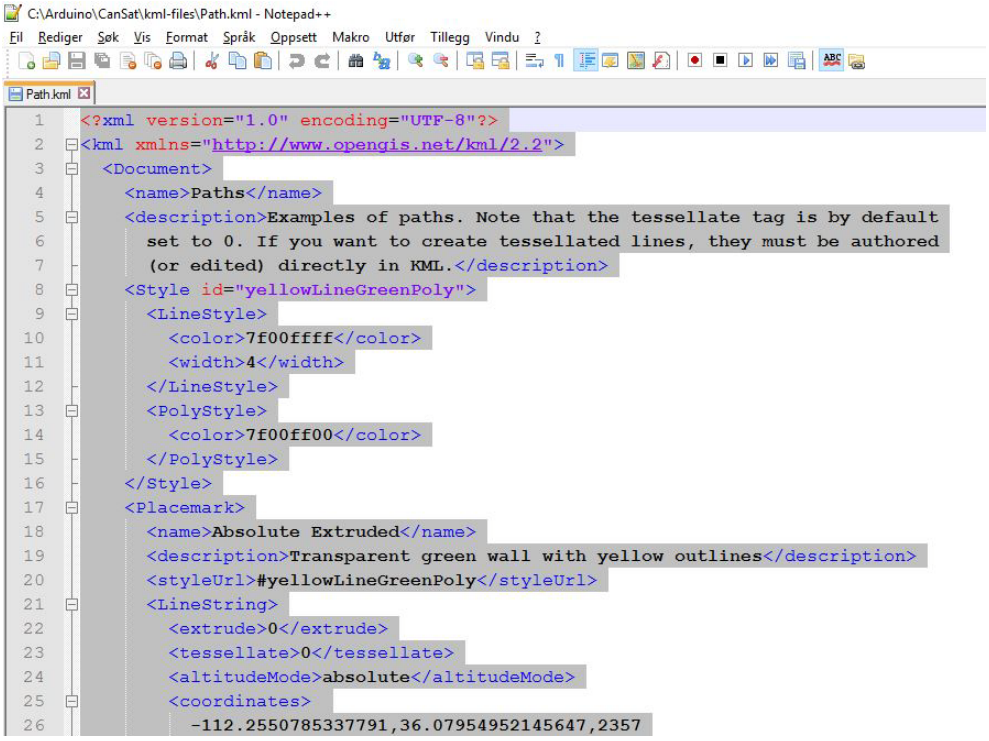
Ved bruk av GPS-data ved f.eks. oppsending av CanSat kan man selvfølgelig ikke neglisjere høydeinformasjonen. Man bør imidlertid være klar over risikoen for relativt store avvik. En bør vurdere å legge inn høydedata fra den barometriske høydemåleren framfor GPS-høydedata.

Editering av kml-fila

Hvilket program skal man så bruke for å legge inn koordinater og høyde. Et nyttig editeringsverktøy til dette formålet er Notepad++. Dette programmet er en litt avansert teksteditor som ikke gjør noen endringer med filformatet så fremt man ikke ønsker det. Notepad++ kan lastes ned fra:

https://notepad-plus-plus.org/downloads/

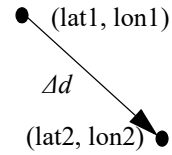
Figuren under viser et utsnitt av brukergrensesnittet til Notepad++.



For å forenkle prosessen med å klippe inn data i kml-koden, er det praktisk å skrive koordinat- og høydedata i det formatet som programmet ønsker: Lengdegrad, breddegrad, høyde. Husk å bruke punktum som desimalpunkt og komma mellom hver verdi. ***NB! Det skal ikke være mellomrom etter kommaene***.

6.4.6 Kode for beregning av akkumulert distanse

Det finnes i dag en rekke App'er for pader og smarttelefoner som beregner gangavstand etter som man beveger seg i gater eller i terrenget, disse er basert på smarttelefonens eller padens innebygde GPS mottaker. Et eksempel på en slik er Runkeeper[37].



Har vi først koblet en GPS-mottaker til vår Arduino, så burde det også være mulig å beregne tilbakelagt distanse. Siden høydemålingene er relativt usikre så nøyer vi oss med å bruke lengde- og breddegradene.

Som vist på figuren over så har vi to sett med koordinater (lat_1, lon_1 og lat_2, lon_2) og ønsker å beregne avstanden, Δd , mellom disse to settene av koordinater. Siden jorda er krummet som en kule så er ikke dette en triviell beregning. Under er gjengitt formelapparatet for beregningen[38]:

37.https://runkeeper.com/

$$\varphi_1 = \frac{\pi \cdot lat_1}{180} \tag{6.4}$$

$$\varphi_2 = \frac{\pi \cdot lat_2}{180} \tag{6.5}$$

$$\Delta\varphi = \varphi_2 - \varphi_1 \tag{6.6}$$

$$\Delta\lambda = \pi \cdot \frac{(lon_2 - lon_1)}{180} \tag{6.7}$$

$$a = \sin\left(\frac{\Delta\varphi}{2}\right)^2 + \cos(\varphi_1) \cdot \cos(\varphi_2) \cdot \sin\left(\frac{\Delta\lambda}{2}\right)^2 \tag{6.8}$$

$$c = 2 \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)}) \tag{6.9}$$

$$\Delta d = R \cdot c \tag{6.10}$$

Hvor:

φ_1 = Breddegrad posisjon 1 (lat_1) målt i radianer
φ_2 = Breddegrad posisjon 2 (lat_2) målt i radianer
atan2 = Arkus tangens med to argumenter for å beholde informasjon om kvadrant
Δd= Distansen i meter
R = Jordradien i m ($6{,}371 \cdot 10^6$ m)

Dette er en relativt nøyaktig beregning, men kan være litt plundrete da standardbiblioteket til Arduino f.eks. ikke tilbyr atan2(). Imidlertid finnes det en forenklet versjon som kan være nyttig når man ikke trenger stor nøyaktighet eller når avstandene mellom målepunktene er liten.

Ekvirektangulær tilnærming

I vårt tilfelle er avstandene noen meter slik at det skulle være helt uproblematisk å bruke den forenklede beregningsmetoden:

$$\varphi_1 = \frac{\pi \cdot lat_1}{180} \tag{6.11}$$

$$\varphi_2 = \frac{\pi \cdot lat_2}{180} \tag{6.12}$$

$$\Delta\lambda = \pi \cdot \frac{(lon_2 - lon_1)}{180} \tag{6.13}$$

38.http://www.movable-type.co.uk/scripts/latlong.html

$$x = \Delta\lambda \cdot \cos\left(\frac{(\varphi_1 + \varphi_2)}{2}\right) \tag{6.14}$$

$$y = \varphi_2 - \varphi_1 \tag{6.15}$$

$$\Delta d = R \cdot \sqrt{x^2 + y^2} \tag{6.16}$$

Hvor:

φ_1 = Breddegrad posisjon 1 (lat_1) målt i radianer
φ_2 = Breddegrad posisjon 2 (lat_2) målt i radianer
Δd = Distansen mellom punktene i meter
R = Jordradien i m ($6{,}371 \cdot 10^6$ m)

Akkumulert distanse

Den akkumulerte distansen finner vi da enkelt ved å summer opp alle de små distansene mellom hvert målepunkt. En må passe på at målepunktene ikke blir for sjeldene slik at en mister de fine detaljene i registreringen.

$$d = \sum_{1}^{n} \Delta d \tag{6.17}$$

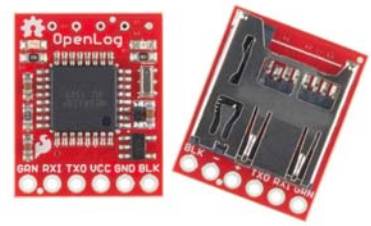
Hvor

d= Lengden av traseen, dvs. summen av de mange små avstandene

n= Antallet målepunkter

6.5 OpenLog – Lagring av data på SD-kort

OpenLog datalogger er en enhet som på enkel måte gjør det mulig å lagre store datamengder på et micro SD-kort. Enheten finnes i flere utførelser og prisklasser. Ønsker man billige utgaver finnes de for under 40 kr. hos AliExpress.



Figuren til høyre viser for og baksiden av komponenten. På baksiden ser vi holderen for SD-kortet.

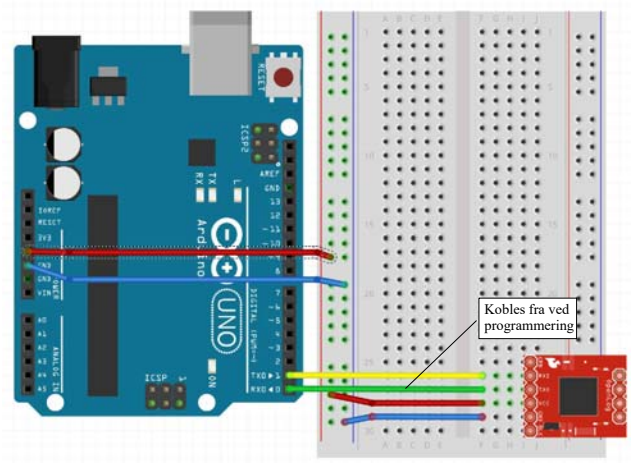
Enheten kan tilføres supply spenninger fra 3,3 – 12 V, spenningene inn på RXI bør være i området 2,0 – 3,3 V, mens spenningene ut av TXO er 3,3 V. Strømtrekket er fra 2 – 5 mA uten SD-kort, og fra 5 – 6 mA med kortet installert. Under skriving til kortet vil strømtrekket komme opp i 20 – 23 mA.

OpenLog er bygget opp rundt en ATmega328, med en klokkefrekvens på 16MHz. ATmega328 benytter Optiboot bootloader, som gjør OpenLog kompatibel med Arduino Uno innstillingene i Arduino IDE. Normalt vil man kommunisere med kortet via dets UART (RXI og TXO). To lysdioder finnes på OpenLog kortet. STAT1 – (blå) blinker når det overføres et tegn. STAT2 – (grønn) blinker når SPI-grensesnittet er aktivt.

OpenLog kan håndtere kortstørrelser på opp til 32 GByte.

6.5.1 Oppkobling

Figuren under viser oppkobling av OpenLog. I denne oppkoblinger vi valgt å koble OpenLog opp parallelt med USB-kanalen. Det har en ulempe og det er at TXO-porten på OpenLog må frakobles mens Arduino'en programmeres.



6.5.2 Forenklet datalagring

Dersom *Tx* på Arduino UNO er tilknyttet *RXI* på Openlog, og *Rx* på Arduino UNO er tilkoblet *TXO* på Openlog, kan en skrive til loggeren ved hjelp av kommandoene *print()* og *println()* som vist i eksempelet under. Pass på at dataene skrives på ASCII-form og ikke binært. Dermed kan dataene på kortet lett leses med en vanlig teksteditor.

```
void setup
     {
     Serial.begin(9600);      // Setter dataraten til 9600 bps (er også default)
     }
void loop
     {
     Serial.println("123");
     delay(500);
     }
```

Dette enkle programmet skriver tallene *123* inn i en fil på kortet. *Ny fil opprettes automatisk hver gang Openlog slås på*. Filene får navn ala *LOG0000n.txt*, hvor n er et løpende nummer som går fra 0000.

Ved større datamengder og stor hastighet kan det være nødvendig å legge inn et lite delay etter Serial.print kommandoen:

```
void setup
     {
     Serial.begin(115200);          //Sett dataraten til115 200 bps
     }
   void loop
   {
   for(int i = 1 ; i < 10 ; i++) {
    Serial.print(i, DEC);
    Serial.println(":abcdefghijklmnopqrstuvwxyz-!#"); // Skriv ASCII strenger
                                                          til Open log
    delay(15);
   }
```

Følgende fil skriver verdier for variablene spenning, trykk og temperatur til serieporten. Dersom Openlog og/eller radiolinken er tilkoblet, skrives det samme til disse.

```
   char buff[50];
   float Voltage = 3.5134;
   float Trykk = 100.650;
   int Temp = 22;

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

   void loop()
   {
     Serial.print("Spenning = "); // Skriver tekst
     Serial.print(Voltage,2);     // Skriver spenningen med 2 desimaler
     Serial.print(" V, ");        // Skriver benevningen "V"
     Serial.print("Trykk = ");    // Skriver tekst
     Serial.print(Trykk,3);       // Skriver trykket med 3 desimaler
     Serial.print(" kPa, ");      // Skriver benevningen "kPa"
     sprintf(buff,"Temperatur = %01d", Temp);
                                  //Skriver streng inn i buffer på alternativ måte
     Serial.println(buff);       //Skriver strengen buff til serieport/Openlog
   }
```

6.5.3 Avansert datalagring

Noen ganger ønsker vi å utføre mer avanserte funksjoner som f.eks. å opprette filer med spesifiserte navn, ev. slette andre filer i datalagringsenheten (OpenLog). For å utføre slike kommandoer må man sette OpenLog i kommandomodus. Men før vi ser på hvordan vi gjør det, la oss se på noen kommandoer som det kan være nyttig å kjenne til:

new *filnavn*

Oppretter en ny fil med navnet *filnavn*. Filnavnet skal være av formen 8.3 (f.eks. **new** *filename.txt*). Etterfølgende data skrives inn i filen.

append *filnavn*

Legger inn nye data på slutten av en tidligere opprettet fil, gjerne etter at programmet har vært stoppet og startet på nytt (f.eks. **append** *filename.txt*). Dersom en ønsker å avslutte denne operasjonen kan en sende kommandoen ctrl+z (ASCII 26).

write *filnavn offset*

Denne kommandoen gir mulighet til å skrive inn i filen *filename* fra en gitt posisjon, *offset*. F.eks. vil kommandoen **write** *filename.txt 516*, starte skrivingen ved byte nr. 516 ved neste *serial.print* kommando.

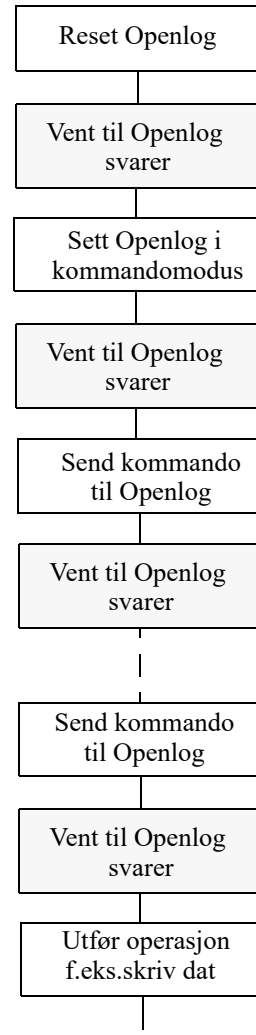
rm *filnavn*

Denne kommandoen fjerner filen med navnet *filnavn* fra OpenLog.

I tillegg finnes en rekke andre kontrollkommandoer. Disse er omtalt bl.a. på følgende nettside:

https://github.com/sparkfun/OpenLog/wiki/Command-Set

For å anvende slike kommandoer må vi sette Openlog i kommandomodus. Flytdiagrammet til høyre viser hvordan dette gjøres:



Reset Openlog:

Openlog resettes ved å legge pinne GRN lav i 100 ms. GRN er koblet til D2 (digital port 2) på Arduino-kortet. *Reset* gjøres med følgende kommando:

```
digitalWrite(2, LOW);
delay(100);
digitalWrite(2, HIGH);
```

Vent til OpenLog svarer:

Ved *reset* eller overføring av en hvilken som helst annen kommando, må en vente på at OpenLog svarer med "<". Denne ventefunksjonen kan utføres på følgende måte:

```
while(1)
{
if(Serial.available())
if(Serial.read() == '<') break;
}
```

while(1) rutinen vil forsette i det uendelige. Den eneste måten å komme ut av *while*-rutinen, er ved å nå fram til et *break*. Dette oppnås i dette tilfellet dersom begge *if*-betingelsene er oppfylt. *Serial.available()* angir hvor mange data som ligger klart i utbufferet hos OpenLog, dersom dette er forskjellig fra 0 leses innholdet i bufferet med kommandoen *Serial.read()*. Om den leste karakteren er *"<"* er betingelsen gyldig og programmet bryter ut av while-rutinen.

Sett OpenLog i kommandomodus:

For å sette Openlog i kommandomodus sendes tre ganger *ctrl-z* etter hverandre. *Ctrl-z* har ASCII-verdien 26.

```
Serial.write(26);
Serial.write(26);
Serial.write(26);
```

Deretter må en vente på at Openlog svarer med "<".

Send kommandoer til Openlog:

Dernest sendes den ønskede kommandoen. I sin enkleste form kan den se slik ut dersom vi ønsker å opprette en ny fil med navnet CanSat.txt":

```
Serial.println("new CanSat.txt");
```

Deretter må en som vanlig vente på at OpenLog svarer med "<".

Normalt vil en ikke bare åpne en ny fil, en vil også skrive til den, ikke bare en gang, men fortløpende legge til data etter som disse kommer inn. Da kan en f.eks. benytte append kommandoen:

```
Serial.println("append CanSat.txt");
```

for så å vente på at Openlog svarer med "<".

Skriv data til Openlog:

Skriving av data gjøres på vanlig måte med følgende kommando:

```
Serial.println("...data");
```

Dataene skrives på ASCII-form, dvs. i hermetegn.

En komplett rutine for skriving på nettstedet:

https://github.com/sparkfun/OpenLog/blob/master/OpenLog_CommandTest/OpenLog_CommandTest.ino

og kode for skriving til OpenLog finnes på:

https://github.com/sparkfun/OpenLog/blob/master/OpenLog_Test_Sketch/OpenLog_Test_Sketch.ino

og for å skrive og lese fra fil i OpenLog:

https://github.com/sparkfun/OpenLog/blob/master/OpenLog_ReadExample/OpenLog_ReadExample.ino

Men som sagt innledningsvis. Det enkleste er å koble den til Arduino'ens Rx og Tx porter (D0 og D1) og bruke Serial.print-kommandoene.

7 Referanser

[1] Gunnar Stette, *Romtekologi - Del av faget Teknologi og Forskningslære (CanSat)*, NTNU juli 2011.

[2] Mer informasjon om Sparkfun Invention's kit:
https://www.sparkfun.com/products/11227

[3] Mer informasjon om I^2C-bussen:
http://www.i2c-bus.org/

[4] Mer informasjon om Serikommunikasjon på SPI-bussen:
http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

[5] Koblingsskjema for Arduino UNO:
http://arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf.

[6] For mer informasjon om Arduino UNO R3 layout:
http://arduino.cc/en/Main/ArduinoBoardUno/

[7] For nedlasting av programeditoren IDE for Arduino:
http://arduino.cc/hu/Main/Software

[8] For nedlasting av Referansemanualen for Arduino C++
http://arduino.cc/en/Reference/HomePage

[9] For mer informasjon om BMP180 og bruk av biblioteket
https://learn.sparkfun.com/tutorials/bmp180-barometric-pressure-sensor-hookup-/installing-the-arduino-library

[10] For kjøp av BMP180
https://learn.sparkfun.com/tutorials/bmp180-barometric-pressure-sensor-hookup-/installing-the-arduino-library

[11] Skolelaboratoriets blå hefteserie:
https://www.ntnu.no/skolelab/bla-hefteserie

[12] Portal for norske målestasjoner
http://www.xgeo.no/index.html?p=klima

[13] Kommunekart, her Trondheim
https://kart5.nois.no/trondheim/Content/Main.asp?layout=trondheim&time=1550437879&vwr=asv

[14] Norske værstasjoner:
http://www.vaisala.fi/Vaisala%20Documents/Measurement%20Theory/height_calculation.pdf

[15] Omregningstabell for trykk og høyde
https://www.mide.com/pages/air-pressure-at-altitude-calculator

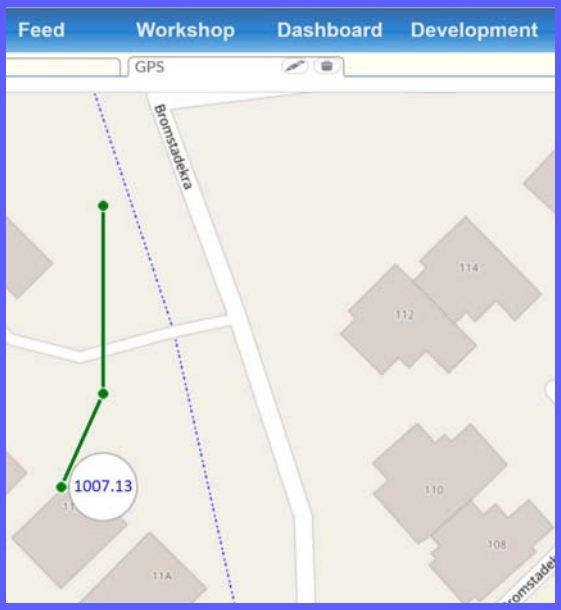
[16] Privat oversikt over værstasjoner:
http://www.bjonnes.net/weatherstations/

[17] Oversikt over vind i norske havner
https://portwind.no

Vedlegg A Komponentliste

Tabellen under inneholder noen av komponentene som er brukt og hvor de kan skaffes fra:

Typebetegnelse	Type komponent	Leverandør
ESP32 WROOM 32	30 pin (kr. 54,74)	https://www.banggood.com/3pcs-Geek-creit-ESP32-WiFi+Bluetooth-Development-Board-Ultra-Low-Power-Consumption-Dual-Cores-Unsoldered-p-1652457.html
Adafruit SSD1306	0,96 Inch Display128 x 64 OLED LCD (kr. 55,82)	https://www.banggood.com/Geekcreit-0_96-Inch-4Pin-White-IIC-I2C-OLED-Display-Module-12864-LED-Geekcreit-for-Arduino-products-that-work-with-official-Arduino-boards-p-958196.html
GPS NEO 6M	En enkel GPS mottaker med satellitt antenne (kr. 74,47)	https://www.banggood.com/no/GY-GPS-Module-Board-9600-Baud-Rate-With-Antenna-p-1196661.html
BMP280	BOSCH Barometrisk luft-trykksmåler (kr. 13,73)	https://www.banggood.com/10Pcs-GY-BMP280-3_3-High-Precision-Atmospheric-Pressure-Sensor-Module-p-1113784.html
OpenLog	Open source datalogger utviklet av SparkFun, men det finnes mange kloner (kr. 55,83)	https://www.banggood.com/CJMCU-717-OpenLog-Data-Recorder-Flash-Recorder-Sensor-Module-Support-64GB-Micro-SD-Card-p-1461328.html
16GB SD-kort:	Micro SD-kort (kr. 33,22)	https://www.banggood.com/no/SOMNAM-bulisT-Mini-16GB-32GB-64GB-128GB-Memory-TF-SD-Card-Flash-Card-Smart-Card-for-Mobile-Phone-Laptop-p-1727893.html
SD-kort adapter	Micro SD-kort adapter: (kr. 13,89)	https://www.banggood.com/no/32G-Memory-Card-CLASS-10-High-speed-Micro-SD-Card-USB-Card-Reader-for-TF-Card-p-1735582.html
DS1307 RTC:	Real Time Clock med batteri CR1220 (kr. 70,60)	https://www.elfa.se/en/ds1307-real-time-clock-adafruit-3296/p/30129214



Undervisningsopplegget tar utgangspunktet i et opplegg laget som et forkurs i programmering til CanSat-kurset som har fokus på bruk av trykksensoren BMP180/280 som grunnlag for måling av lufttrykk. Sammenhengen mellom lufttrykk og høyde over bakken anvendes så til å bestemme relativ høydeforandring. Barometriske høydemålere brukes fortsatt i forbindelse med flytrafikk da den har en langt bedre nøyaktighet enn GPS så fremt instrumentet kalibreres ved hver flyvning. Vi har derfor inkludert kalibrering av instrumentet opp mot nærliggende meteorologiske stasjoner for å kunne måle absolutt høyde over havet. Et enkelt OLED display brukes for å vise resultatet av målingene. I tillegg har vi inkludert en GPS-mottaker og en datalogger (OpenLog) slik at vi kan ta vare på måtedata på et micro SD-kort og merke dem med stedskoordinater. I tillegg bruker vi en Real Time Clock (RTC) som har batteri backup for å tidsstemple målingene våre. ESP32 er valgt som mikrokontroller. Også dette opplegget anvender ESP32 som gir oss mulighet til å koble oss opp mot Internett. Vi anvender denne muligheten ved å tegne inn måleresultatene i en kartdatabase og vise resultatet på Circus og Things via Internett.

Nils Kr. Rossing
Dosent ved Skolelaboratoriet
E-post: nils.rossing@ntnu.no

ISBN
ISSN

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