Rev 3.0

Kåre-Benjamin H. Rørvik og
Nils Kr. Rossing

Wi-Fi programmering med ESP32 – (YFL) – UL

NTNU
Trondheim
Institutt for fysikk
Skolelaboratoriet
for matematikk, naturfag og teknologi
Institutt for elektroniske systemer

Mars 2021

Wi-Fi programmering med ESP32 – (YFL)

Kåre-Benjamin H. Rørvik, Institutt for elektroniske systemer og
Nils Kr. Rossing, Skolelaboratoriet

Wi-Fi programmering med ESP32 – (YFL) – Uten løsningsforslag

Trondheim 2021

Layout og redigering:	Nils Kr. Rossing, Skolelaboratoriet ved NTNU
Trykk:	NTNU Grafisk senter
Tekst og bilder:	Nils Kr. Rossing, Skolelaboratoriet, NTNU, Kåre-Benjamin H. Rørvik, Institutt for elektroniske systemer, NTNU
Forsidebilde:	Internett: www.circusofthings.com
Programmering:	Kåre-Benjamin H. Rørvik, Nils Kr. Rossing, NTNU
Kursholdere:	Arne Midjo, Kåre-Benjamin H. Rørvik, Even Johan Christiansen Institutt for elektroniske systemer

Faglige spørsmål rettes til:
Skolelaboratoriet for matematikk, naturfag og teknologi
Institutt for fysikk
v/ Nils Kr. Rossing, nils.rossing@ntnu.no
v/ Kåre-Benjamin H. Rørvik , kbrorvik@stud.ntnu.no

Skolelaboratoriet ved NTNU
Realfagbygget,
Høgskoleringen 5,
7491 Trondheim

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

Rev 3.0 – 19.03.21

Undervisningsopplegget er utviklet i samarbeid med Institutt for elektroniske systemer ved Arne Midjo.

Forord

Kurshefte beskriver et undervisningsopplegg som er en videreføring av kursmodulen: *ESP – Grunnkurs programmering (YFL)*. Opplegget er utarbeidet spesielt med tanke på yrkesfagelever på *elektro*, men burde også passe for *bygg og anleg*g. Opplegget har spesielt fokus på bruk av Wi-Fi-modulen til ESP32, som gir den mulighet til å fungere som en trådløs sensornode som kan kobles opp mot Internett. Kretsen er derfor spesielt egnet til bruk i forbindelse med Internet of Things (IoT).

Siden den bruker samme programmeringsverktøy som Arduino er den spesielt attraktiv for tidligere Arduino-brukere.

Det omtalte undervisningsopplegget legger størst vekt på å ta i bruk ESP32s Wi-Fi egenskaper og viser hvordan ESP32 gir oss mulighet til å overføre signaler mellom en lokal sensornode og en nett-tilkoblet PC eller Smarttelefon. Utstyret egner seg derfor godt for trådløs styring og overvåking innen rekkevidden av en lokal ruter.

Fra revisjon 3.0 er det lagt inn løsningsforslag i oppdragsteksten. Dette er ufulstendige løsningsforslag som krever at deltakerne finner en feil eller supplerer med kode.

Utviklingen er delvis finansiert av Udir og Trøndelag fylkeskommune, og er en "spinn off" av et mer omfattende videreutdanningskurs rettet mot yrkesfaglærere både innen Elektro, Bygg og anlegg og Teknisk Industriell Produksjon (TIP) som vil bli gjennomført første gang våren 2021 ved NTNU i Trondheim.

En spesiell takk til Institutt for elektroniske systemer og Arne Midjo som har fulgt prosjektet og kommet med verdifulle innspill. Videre at instituttet har stilt midler til rådighet slik at Kåre-Benjamin H. Rørvik fikk lagt grunnlaget for valg av Circus of Things som nettressurs, og i samarbeid med Skolelaboratoriet har være en viktig partner for å utvikle og teste ut denne delen av undervisningsopplegget. Instituttet har også bidratt med studentassistenter som veiledere på selve kursdagen.

Skolelaboratoriet ved NTNU
Mars 2021
Kåre-Benjamin H. Rørvik
Nils Kr. Rossing

Innhold

1 Innledning

Kurset er en påbygningningsmodul til kurset: *ESP32 – Grunnkurs programmering (YFL)* og har som målsetning å gi kursdeltagerne en bredere programmeringserfaring. Fokuset for denne kursmodulen er ESP brukt som sensornode koblet opp mot Internett. Videre viser også denne delen av kurset hvordan et system kan brytes ned til enklere enheter som så kan settes sammen til en funksjonell helhet. Tanken er at deltagerne skal arbeide selvstendig med arbeidsheftet under veiledning og nå delmål etter delmål samtidig som de underveis kombinerer enkeltdeler til et større system. Dermed blir det ikke så viktig at de når alle delmålene i undervisningsopplegget, men at de heller kan få ro til fordypning innen hvert delmål. Hovedmålet eller den endelige prosjektideen kan likevel gi arbeidet med delmålene en retning og være en viktig motivasjon. Dersom det er mulig anbefaler vi sterkt at flere lærere samarbeider under kurset, som spesielt gjelder siste del av undervisningsopplegget der man vil trenge minst to ESP32 for å opprette direkte Wi-Fi-kontakt mellom to eller flere ESP32.

Selv om denne modulen står på egne ben, er det en fordel om man på forhånd har gått gjennom *ESP – Grunnkurs programmering (YFL)* da denne modulen bygger på deler av det kurset. Selv om vi ikke tar i bruk hele oppkoblingen fra grunnkurset, så bruker vi noen av elementene slik at deltakerne kan se at de to systemene som utvikles kan knyttes sammen.

I tillegg til å koble opp kontrolleren til Internett vil vi ha fokus på å måle luftkvalitet i et rom. Deltakerne vil måle temperatur, luftfuktighet og lufttrykk og overføre målingene til en nettside som kan nås med PC eller mobil. I tillegg demonstreres fjernstyring av noen utvalgte aktuatorer knyttet til "ventilasjon" og adgangskontroll av rommet.

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 går rykter om at konferansesal A, som nylig har fått installert elektronisk adgangskontroll, har dårlig luftkvalitet. Vi ønsker å overvåke rommet over tid ved å måle temperatur og luftfuktighet. Vi ønsker at måleresultatene skal være tilgjengelig over nett slik at vi kan kontrollere tilstanden med jevne mellomrom fra en kontroll- og overvåkningssentral. Vi ønsker også å studere virkningen av at ventilasjonsanlegget slås av og på. Ved i tillegg å måle lufttrykket ved start og stopp av anlegget, vil vi kunne si noe om luftgjennomstrømningen i rommet. Vi ønsker også å ha mulighet til å studere hvordan temperaturen og luftfuktigheten i rommet utvikler seg over tid, og anledning til automatisk å slå på ventilasjonsanlegget når temperaturen overskrider en terskelverdi styrt fra sentralen.*

Vi ønsker også å ha mulighet til å utvide systemet til å omfatte flere rom, ved at vi plasserer batteridrevne sensornoder med lavt strømforbruk i naborom. Disse skal kunne kommunisere med en sentral node som via det lokale trådløse Wi-Fi nettet skal overføre informasjonen til sentralen.

Det er en selvfølge at avlåsing og åpning av rommet skal kunne styres fra sentralen.

La oss ganske kort beskrive de ulike oppdragene:

Oppdrag 1: *Koble opp viftereleet (som styrer viften) og skriv et lite program som slår releet av og på.*

Dette innledende oppdraget er en enkel intro.

Oppdrag 2: *Slå viftereleet av og på via Internett ved hjelp av nettressursen Circus of Things.*

Dette er et ganske sammensatt oppdrag hvor vi først må skaffe oss en bruker hos Circus of Things (CoT), konstruere et enkelt brukergrensesnitt (panel) og opprette forbindelsen til vår lokale ESP32.

Oppdrag 3: *Koble opp sensoren BME280 og skriv ut måleverdiene for luftfuktighet, temperatur og lufttrykk til monitoren i IDE (Arduino editoren).*

Her introduseres sensoren BME280 som måler parametere knyttet til luftkvaliteten. BME680 er kanskje enda mer relevant siden den i tillegg måler gasser i lufta såkalt VOC (Volatile Organic Compounds). BME680 vil ikke bli videre omtalt i dette heftet.

Oppdrag 4: *Les av sensorene og presentere resultatene på panelet hos Circus of Things.*

I tillegg til å lese av sensorene, sendes måleverdiene til CoT som viser måleverdiene fra BME280 på ulike konsoller.

Oppdrag 5: *Styr viftereleet fra panelet og les av sensorene og presenter dem på panelet hos Circus of Things.*

I dette oppdraget kombinerer vi oppdrag 2 og 4 om det ikke alt er gjort. Med det viser vi hvordan vi kan integrere to oppdrag i samme programvare.

Oppdrag 6: *Styr bommen fra panelet slik at den kan åpnes i en vilkårlig vinkel mellom 0 og 90°. Integrer styring av bommen i resten av programmet – Oppdrag 5.*

Vi kobler til bommen (dørlåsen) slik at også den kan fjernstyres fra sentralen. Det er ganske vanlig i dag at låsene rundt om i et bygg kan fjernstyres fra en sentral enhet. Ved hjelp av en glidebryter i styringspanelet kan vi til og med sette bommen i en ønsket vinkel mellom 0 – 90°. Dette er mest for å vise at det er mulig.

Oppdrag 7: *Ta utgangspunkt i oppdrag 6 og lag fem funksjoner som utfører følgende oppgaver:*
– Leser temperatur fra BME280 og sender til CoT.
– Leser luftfuktighet fra BME280 og sender til CoT.
– Leser lufttrykk fra BME280 og sender til CoT.
– Mottar styringsinfo. for releet fra CoT og setter det i ønsket stilling.
– Mottar styringsinfo. for servo fra CoT og setter den i ønsket stilling.

Vi er kommet til et punkt i utviklingen av systemet at det er naturlig å definere egne funksjoner for hver av oppgavene. Dette er en hjelp til å strukturere programmet og få oversikt. Bruk av beskrivende navn på funksjonene er viktig.

Oppdrag 8: *Ta utgangspunkt i oppdrag 7 og legg til en funksjon som skriver temperaturen til 7-segment displayet.*

Oppdrager viser hvordan vi i større grad kan inkludere deler av systemet brukt under adgangskontrollen. Vi har valgt å ikke inkludere alle delene av adgangskontrollen, men heller vise at dette er mulig. Det er derfor opp til elevene å inkludere det de synes gir mening. Vi har valgt å vise temperaturen på displayet.

Oppdrag 9: *Lag et nytt panel hos CoT som viser hvordan temperaturen målt av BME280 har endret seg over tid. Legg til luftfuktighet i samme diagram.*

Hensikten med dette oppdraget er å vise hvordan man kan vise trender i f.eks. hvordan temperatur og luftfuktighet endrer seg. Noe som kan være aktuelt for å overvåke hva som skjer med luftkvaliteten over tid.

Oppdrag 10: *Lag en ny konsoll og endre den eksisterende slik at vifta kan settes i tre stillinger:*
- On – Går hele tiden
- Auto – Går når temperaturen er over en terskelverdi
- Off – Går ikke
Terskelverdien skal kunne settes kontinuerlig fra 10 – 40°C.
Endre programmet slik at det oppfyller betingelsene.

Dette oppdraget utfordrer elevene til å koble målinger med styring. Dvs. at på bakgrunn av en måling utføres en automatisk respons hvor terskelnivåene kan stilles inn av en operatør. Vifta starter når temperaturen overskrider en terskelverdi. Dette blir enda mer meningsfylt dersom en kan måle f.eks. CO_2 nivå og starte vifta når dette er for høyt.

Oppdrag 11: *Lag et tillegg til programmet som legger ESP32 i dvale mellom hver måling.*

I forbindelse med Internet of Things er det spesielt viktig å redusere strømforbruket hos en krets. I dette oppdraget undersøker vi konsekvensene av å legge ESP32 i dyp søvn mellom hver måling. Under dyp søvn kan strømforbruket reduseres ned til 10µA.

Oppdrag 12: *Opprett kontakt mellom to ESP32, node-1 og node 2. Node-1 samler inn måleresultater og sender informasjonen til node-2 som viser temperaturen på displayet.*

Hensikten med oppdraget er å vise at det er mulig å oppnå trådløs kommunikasjon mellom to eller flere sensornoder, noe som kan være aktuelt dersom man ønsker at en sensornode koordinerer data fra en samling av sensornoder.

Oppdrag 13: *Opprett forbindelse mellom node 2 og CoT og vis data fra begge nodene på hvert sitt panel.*

Dette oppdraget skal vise at det er mulig å koordinere data fra flere noder og vise det samlede resultatet i en konsoll med tilgang på nett. Dette kan være en aktuell problemstilling dersom man ønsker å sammenligne utvendig og innvendig temperatur for et rom.

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 Internet of Things

La oss ganske kort se på hva som ligger i begrepet “Internet of Things” (IoT).

2.1 Definisjon og nytteverdi[1]

> *Internet of Things er et system av sammenkoblete mekaniske og digitale maskiner, objekter, dyr eller mennesker som er utstyret med unike identifikatorer og har evnen til å overføre data over et nettverk. Interaksjonen mellom de ulike objektene er uavhengig av menneske-til-menneske eller menneske-til-datamaskin interaksjon.*

Umiddelbart kan dette virke ganske skremmende siden en kan få inntrykk av at utveksling av data skjer uten vår viten og vilje. Vi legger merke til at sammenkoblingen ikke bare gjelder gjenstander, men at også mennesker og dyr betraktes som “gjenstander” i denne sammenhengen. Et eksempel på det siste er:

> *Innen helsevesenet kan IoT tilby mange fordelaktige tjenester som f.eks. overvåking og analysering av pasientdata. Dette kan være tilfelle på et sykehus eller en intensivavdeling for tidlig å bli klar over kritiske situasjoner. Eller for å holde oversikt over lagerbeholdninger når det gjelder medisiner og medisinsk utstyr.*

Et annet eksempel knyttet til jordbruk kan være:

> *Overvåking av dyrka mark ved å måle og overføre data om lysforhold, temperatur, luftfuktighet og fuktigheten i jorda. IoT gir dermed mulighet til automatisk oppstart av vanningsanlegg, ev. regulering av lys og temperatur i drivhus.*

I slike sammenhenger ser en klart en gevinst mht. optimalisering av jordbruket.

I bynære områder kan en se for seg andre nyttige anvendelsesområder:

> *I forbindelse med “Smart city” kan man se for seg en rekke anvendelser som f.eks. smart gatebelysning og smart trafikkovervåking som gjøre trafikken mer smidig, innføre tiltak som sparer energi og miljø, og forbedre sanitære forhold.*

På et mer personlig plan kan en se for seg:

> *Smarte boliger (“Smart house”) hvor sensorer styrer lys og temperatur i rom på bakgrunn av bruksmønster. Eller styrer luftkondisjoneringsanlegget på bakgrunn av hvor mange som befinner seg i et rom og således trenger utskifting av lufta og styring av oppvarming eller kjøling. På denne måten kan man redusere energibruken på en intelligent måte.*
>
> *Noen av oss disponerer kombinerte kopi- og utskriftsmaskiner som automatisk varsler leverandøren om at det begynner å bli tomt for toner eller papir og sender ut*

1. https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT

nye forsyninger etter behov. Lignende maskiner er dessuten koblet sammen i nettverket slik at uansett hvilken maskin jeg oppsøker så gjenkjennes jeg slik at jeg kan få skrevet ut min tekst på den nærmeste maskinen.

Med fornuftig bruk kan IoT bidra til en bærekraftig utvikling, samtidig som man må være seg bevisst at overdreven bruk like lett kan medføre det motsatte.

En litt mer spesiell anvendelse kan man oppnå ved å montere sensorer på kroppen og i klærne slik at de kan påvirke livsmønsteret vårt:

Bærebare enheter med sensorer og programvare kan samle og analysere og overføre data, til installasjoner i omgivelsene som gjør livet lettere og mer komfortabelt. Slike bærbare enheter kan også redusere responstiden ved behov for akutt hjelp, ved f.eks. at den trengendes lokalisering overføres automatisk til redningsmannskapet, for å gi dem den optimale kjøreveien til den hjelpetrengende. Det samme gjelder for brannvesenet under utrykning.

Vi legger merke til at flere av de nevnte tjenestene finnes alt i dag.

2.2 Litt historie

Som så mange andre ting så oppsto tanken om automatisk fjernstyring og overvåking tidlig. Allerede på 1970-tallet snakket man om "*embeded internet and pervasive computing*", hvilket kanskje kan oversettes som *Integret nettilkoblet databehandling*.

Den første internettapplikasjonen var f.eks. en Cola-automat ved Carnegie Mellon University tidlig på 80-tallet, hvor programmere ved universitetet kunne sjekke om det var kalde drikker tilgjengelig i automaten, slik at det var verdt turen for å hente seg en kald drikke.


Kevin Ashton (1968 –) *Neil Gershfeld (1959 –)*

Det var imidlertid ***Kevin Ashton (1968 –)***, en av grunnleggeren av Auto-ID senteret ved Massachusetts Institute of Technology (MIT), som først brukte begrepet Internet of Things. Han brukte begrepet i en presentasjon han gjorde for Procter & Gamble i 1999. Presentasjonen hadde til hensikt å rette søkelyset mot RFID (Radio Frequency IDentification), en teknologi vi i dag er godt kjent med gjennom f.eks. adgangs- og betalingskort. Den gang kalte Ashton forelesningen sin Internet of Things". På omtrent samme tid skrev professor ***Neil Gershfeld (1959 –)*** ved MIT boka: *When Things start to Think*. Selv om han ikke brukte begrepet IoT så uttrykket han en klare visjon om at vi er på vei mot det vi i dag omtaler som IoT.

Siden den gang har trådløse teknologi, mikroelektromekaniske systemer og Internett nærmet seg hverandre og etter hvert blitt en fungerende enhet.

2.3 Fordeler og ulemper med IoT

Generelt kan man si at IoT gir større muligheter til å overvåke prosessene i et foretak og på den måten øke produktiviteten hos de ansatte, gjøre kundeservicen bedre og mer effektiv og på den måten spare tid og penger. Fordelene avhenger imidlertid av i hvilken grad man er i stand til å dra nytte av den innhentete informasjon i foretakets driftmodeller slik at man kan ta bedre beslutninger og på den måten skape større overskudd. De mest åpenbare bedriftsmessige fordelene ser en kanskje innen produksjon, transport og lagerhold, men også innen effektivisering av jordbruk (jfr. tidligere eksempel), hushold og hjemmeautomatisering (f.eks. energiøkonomisering) og i forbindelse med "Smart cities" ser en fordeler mht. redusert forsøpling og energisparing.

Andre fordeler er i større grad knyttet til bedriftspesifikke forhold og kan relateres til følgende forhold:

- Gir mulighet til å innhente informasjon om gjenstander, når som helst og fra hvor som helst slik at en f.eks. tidlig kan påvise feil og slitasje og dermed redusere større skader med påfølgende driftsstans.
- Forbedrer informasjonsflyt mellom ulike gjenstander som er tilkoblet nettverket.
- Gir betydelige innsparinger som et resultat av at informasjonen kan overføres automatisk via nettverket.
- Automatisk håndtering av informasjon uten menneskelig inngripen øker kvaliteten på prosesser både innen næringslivet og samfunnet forøvrig.

Men det finnes også ulemper:

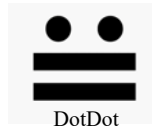
- Ettersom antallet tilkoblede enheter øker og stadig mer informasjon deles over nettverket, blir også faren for at utenforstående får urettmessig adgang til gradert informasjon større.
- Utfordringen med å håndtere store mengder data på en fornuftig måte vil øke.
- Feil som oppstår i et slikt komplekst system vil lett ødelegge data fra samtlige noder siden alt henger sammen.
- Siden det foreløpig ikke finnes noen enhetlig internasjonal standard på dette feltet så oppstår det lett problemer når ulike systemer skal utveksle data.

2.4 Internasjonal standard

Det finnes en rekke kandidater til å bli en internasjonal standard på markedet, men foreløpig har ingen tatt skrittet fullt ut. Her er kanskje de vanligste og mest aktuelle:

- **iPv6** – "Low-Power Wireless Personal Area Networks" (6LoWPAN) er en åpen standard introdusert av *Internet Engineering Task Force* (IETF). Denne standarden tillater en hver laveffekts radio å kommunisere via Internet. iPv6 inkluderer og bygger på protokollen definert av 804.15.4. I denne familien finner vi også *Bluetooth Low Energy* (BLE) og *Z-Wave* som brukes innen hjemmeautomatisering.

- **ZigBee** er et laveffekts trådløst nettverk med lav datarate oftest brukt i industrielle nettverk og er basert på *Institute of Electrical and Electronics Engineers* (IEEE) 802.15.4 standarden. ZigBee-sammenslutningen har utviklet et felles programspråk DotDot[2] for IoT som gjør det mulig for "smarte objekter" å kommunisere trykt og forståelig med hverandre på alle nettverk.


DotDot

- **LiteOS** er et Linux-basert operativsystem for trådløse sensornettverk. LiteOS støtter Smarte telefoner og klokker, wearables[3], men brukes også i forbindelse med Smarte hjem og kjøretøyer koblet opp mot Internet.
- **OneM2M** er en kommunikasjonsstandard beregnet til å kommunisere mellom ulike maskiner f.eks. i en produksjonskjede.
- **DDS** – *Data distribution Service* ble utviklet av *the Object Management Group* (OMG) og er en standard for kommunikasjon mellom maskiner i en produksjonslinje. Standarden er skreddersydd for kommunikasjon med høy kvalitet i "real time", og lar seg lett skalere opp fra små til store systemer.
- **LoRaWAN** – *Long Range Wide Area Network* er en protokoll for Wide Area Network – WAN, utviklet for å støtte f.eks. *Smart cities*, med millioner av enheter som sender på lave effekter.

2. https://zigbeealliance.org/solution/dotdot/
3. Wearables er elektronikk som festes nær kroppen, gjerne på huden, og er elektronikk som samler inn data, analyserer og sender videre informasjon om f.eks. vitale data om kroppens tilstand eller om omgivelsene.

3 ESP 32 - WROOM - 32

3.1 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).

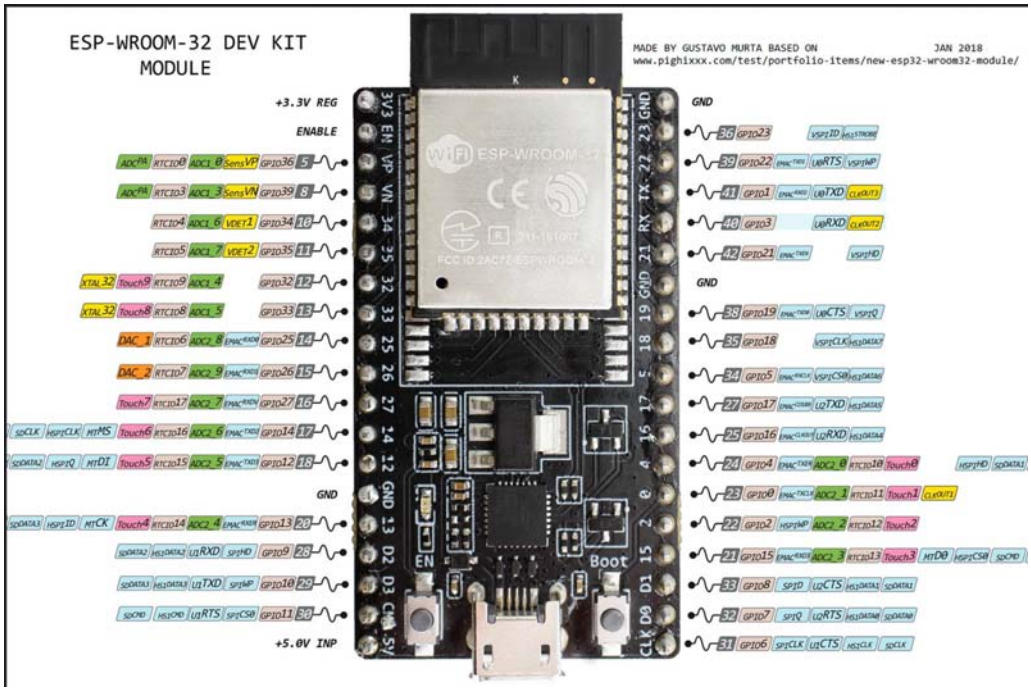
Senderen har en maksimal effekt på +12 dBm (8 mW) og mottakeren en følsomhet på –94 dBm Det er målt rekkevidder på over 200 meter mellom ESP32-enheter i åpent terreng[4].

Kortet har følgende porter:

- **Analoge og Digitale I/O-porter**
  Kortet har 34 digitale[5] 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 [6] og 3 x UART[7]. 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[8]** 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.

4. https://randomnerdtutorials.com/esp-now-esp32-arduino-ide/
5. Dette kan variere noe fra versjon til versjon
6. 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
7. UART – Universal Asynchronous Receiver/Transmitter, linje for toveis overføring av data.
8. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

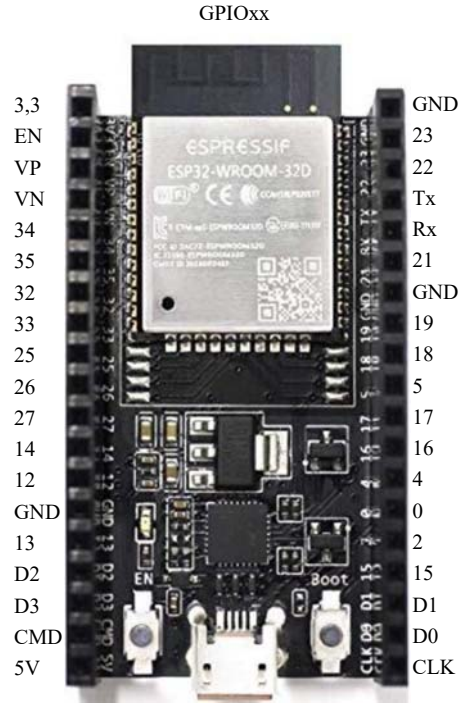
Figuren under viser pinningen for ESP-Wroom-32 DEV KIT med 38 pinner. Vi legger merke til at hver pinne har flere funksjoner.



- *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[9]

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

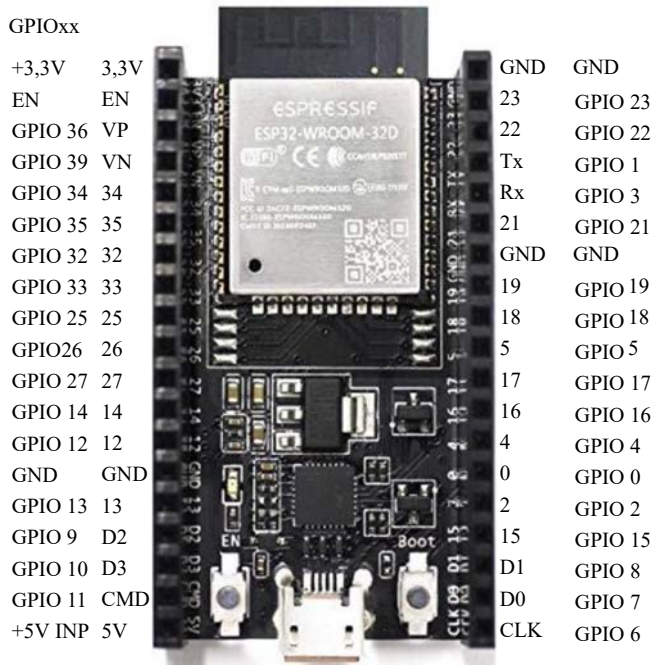


- *Hall sensor*
  En hall magnetfelt sensor er integrert på kortet og kan tilsluttes en av de analoge inngangene.
- *Temperatursensor*
  Kretskortet inneholder en innebygget temperatursensor.
- *Kapasitive berøringssensorer*
  10 av inngangene kan fungere som kapasitive berøringssensorer. Dette er Touch 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.

9. 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[10] nummer.:



Innerst slik kretsen er merket, ytterst GPIO numrene

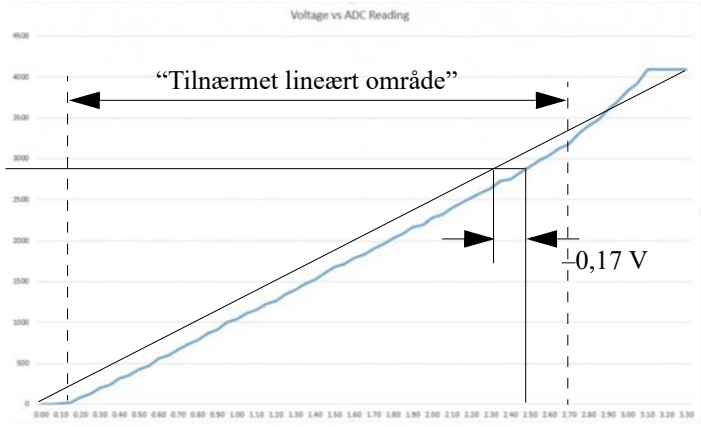
Tabellen under viser anbefalt bruk (Br. 1 – Digital, Br. 2 – Analog og Br. 3/4 – I^2C og Touch).

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	

10.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
```

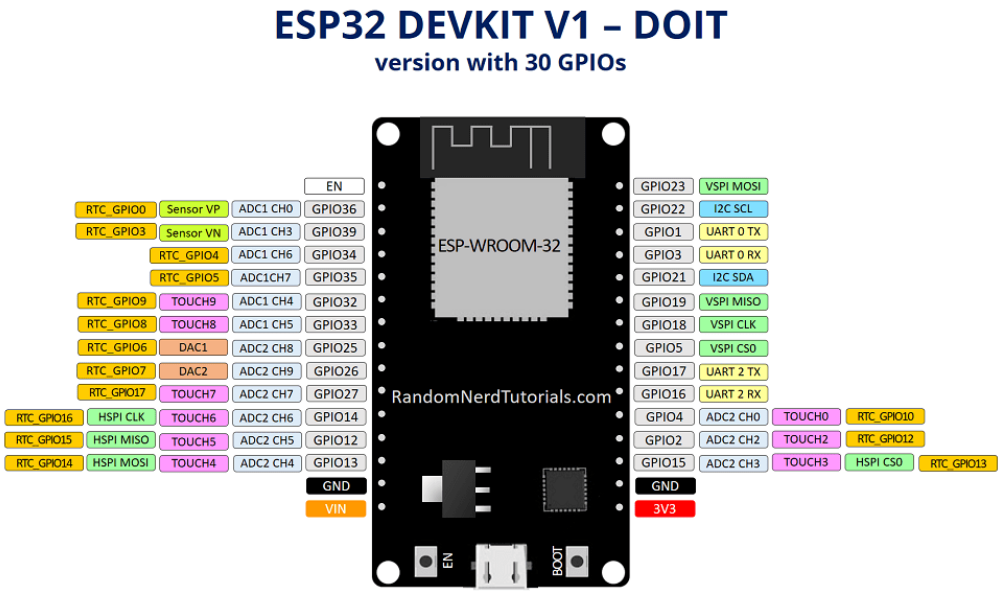
Touch – funksjon (inngang)

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

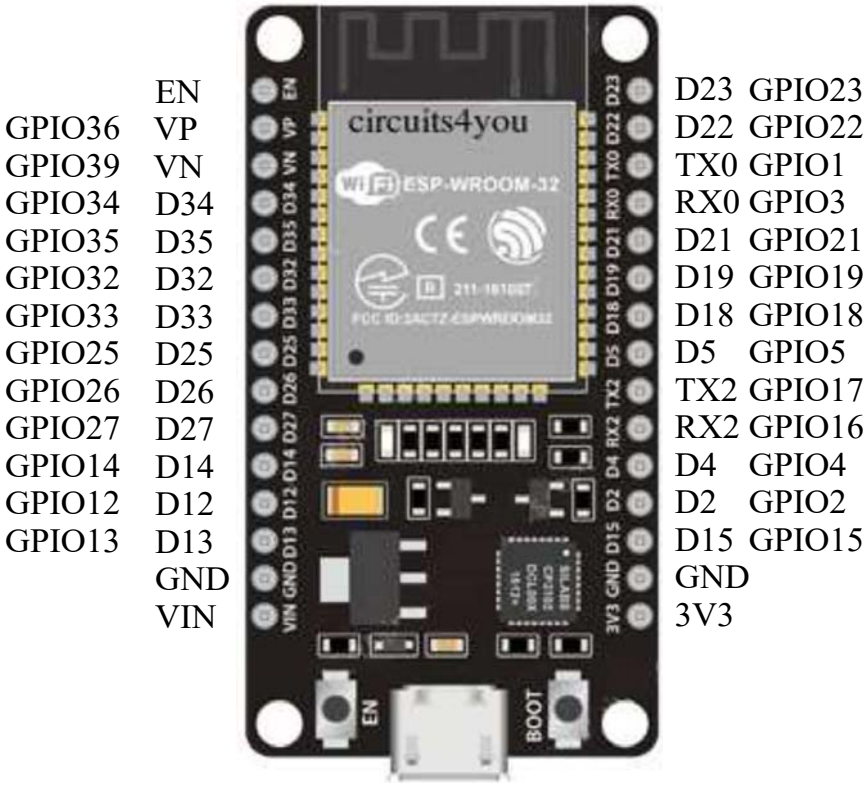
```
int verdi; // Vil ha verdier typisk mellom 10 og 100
verdi = touchRead(4); // Leser tilstanden til GPIO 4, Touch 0
```

3.2 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.

3.3 Energisparing

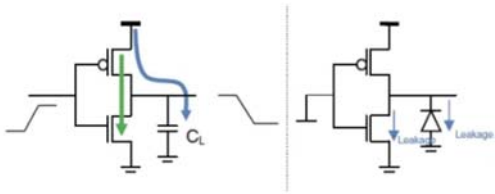
En av de store fordelene ved ESP32 er at kretsen kan legges helt eller delvis i dvale. Dette er en viktig egenskap til IoT sensornoder som er batteridrevet og kun trenger å gjøre målinger en gang i blant. Etter hver måling kan kontrolleren legges i dvale for å spare energi.

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.

3.3.1 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[11]. Som vi ser på tegningen til venstre så består bryteren av to FET-transistorer[12]. 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*.

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

$$P_{Leakage} = f\,(U_{dd},\ V_{th},\ W/L) \qquad (3.1)$$

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:

$$\text{Effekt} = U^2/R_L \qquad (3.2)$$

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.

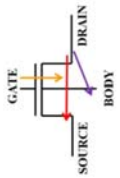
Følgende formel uttrykker det vi har snakket om på en kompakt måte:

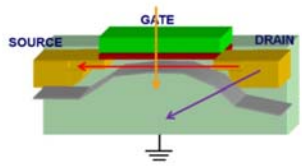
$$P_d = \alpha \cdot f \cdot C \cdot U^2 \qquad (3.3)$$

Hvor:

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

Men også lekkasjestrømmer bidrar til effektomvandlingen. 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[13] som på en meget god måte illustrerer de ulike dvaletilstandene.

3.3.2 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 dataene 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 bølger. 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 målinger, 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 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.

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

- **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

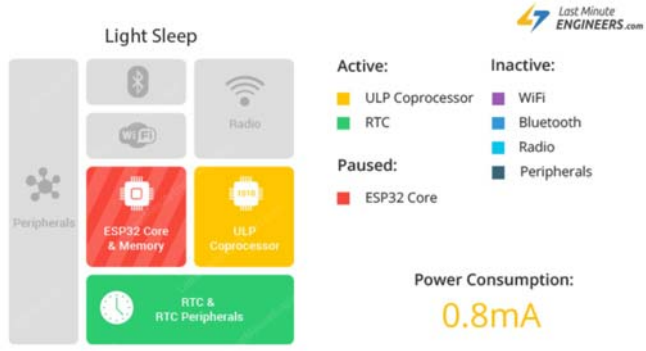
Aktiv mode: I denne tilstanden er alle enhetene operative og i full aktivitet. Avhengig av blant annet sendereffekten så kan forsyningsstrømmen bli ganske høy. I enkelte tilfeller kan strømmen nærme seg 800mA i korte øyeblikk dersom begge kommunikasjonsenhetene (Wi-Fi og Bluetooth) er aktive.

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.

3.3.3 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():
```

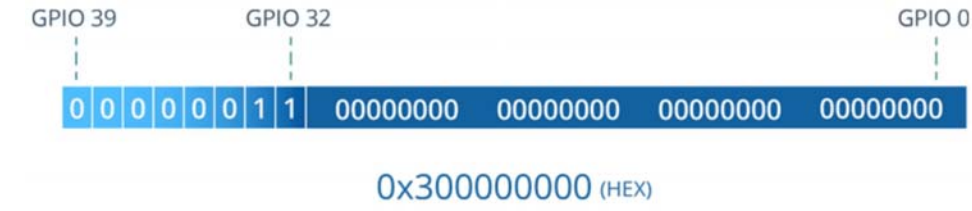
Sørger for å legge kontrolleren i dvale.

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
```

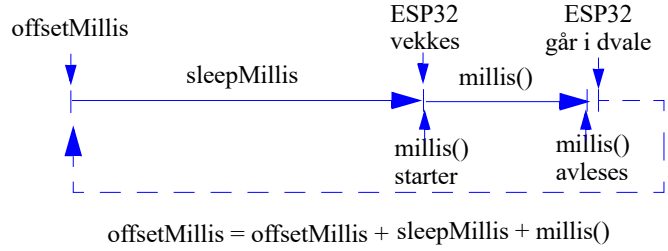
Denne setter port 32 og 33 aktive (1) og resten inaktive. Legg merke til at to siffer i Hex styrer 8 binære siffer.

Et eksempel på denne finnes i vedlegg B.3, side 146. I dette eksempelet vekkes kretsen ved at to knapper trykkes samtidig.

3.3.4 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:



Vi definerer en variabel og legger den et sted i lageret som ikke slettes under dvaleperioden. Vi vet at det finnes et lite lageområde 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();          (3.4)
```

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 3.4.

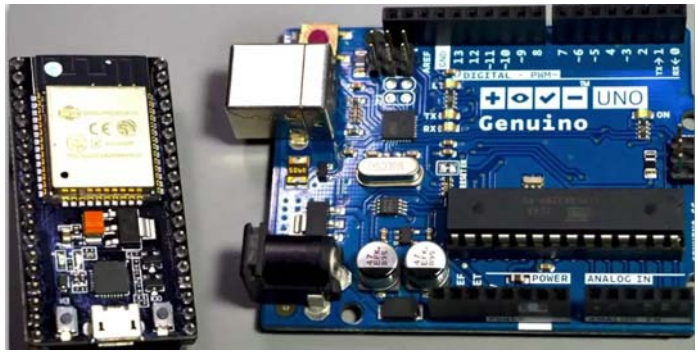
For nærmere beskrivelse av denne problematikken se: https://www.robmiles.com/journal/2020/1/22/esp32-retaining-timing-over-deep-sleep

4 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 4.1, og 4.3, side 38, som omtaler installasjon av ekstrapakken som gjør det mulig å bruke Arduino-grensesnittet for ESP32.

4.1 Forskjeller og likheter hos Arduino og ESP32

La oss ganske kort sammenligne ESP32 og en "vanlig" Arduino, f.eks. UNO[14].



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 4.3, side 38).

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.

14. 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.

4.2 Installasjon av programvare

Om du har installert Arduino IDE tidligere kan du hoppe over dette avsnittet og gå dirkete til avsnitt 4.3, side 38.

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

4.2.1 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:

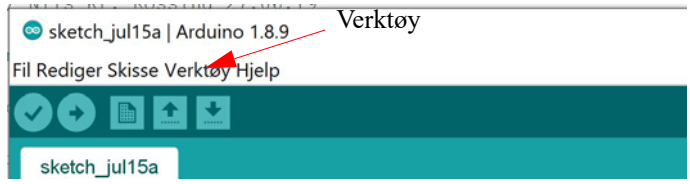
1. Klikk på *Windows installer* og følg anvisningen under installasjonen. For MAC-brukere klikk på *MAC OS X*

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 4.3).

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

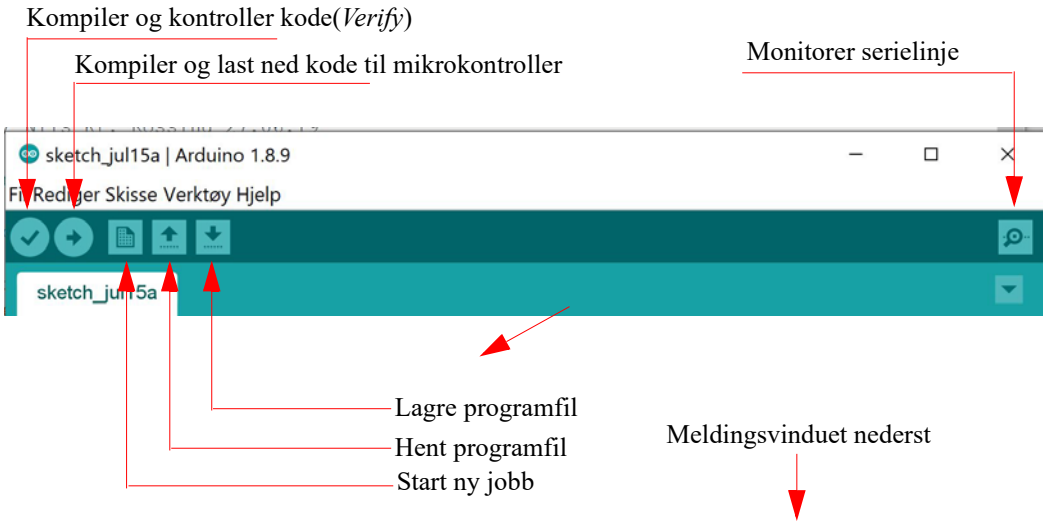
 Kompiler og last ned programmet til mikrokontrolleren

 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:

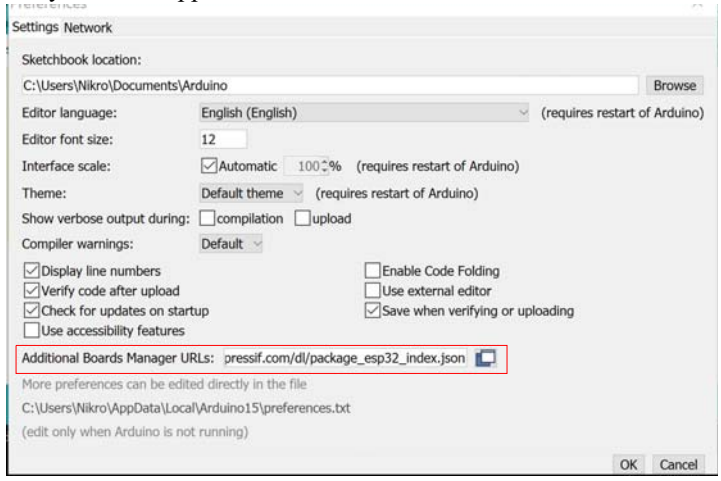
- Kabelen er ikke tilkoblet, eller ødelagt
- Feil port er valgt av programeditoren, som kan endres ved å velge: *Verktøy* og *Port* fra menylinjen i editoren
- Feil type mikrokontroller-kort er valgt
  Endres ved å velge: *Verktøy* og *Kort* fra menylinjen, for så å velge rett kort, i vårt tilfelle *Arduino/Genuino 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.

4.3 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[15].

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:

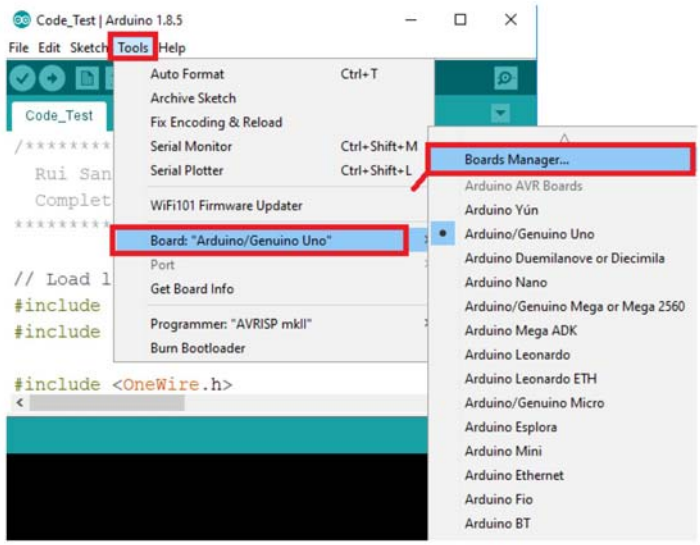
1. Gå til menylinjen på Arduino IDE og velg: *File —> Preferences.*
   Følgende meny kommer opp:



15. 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:



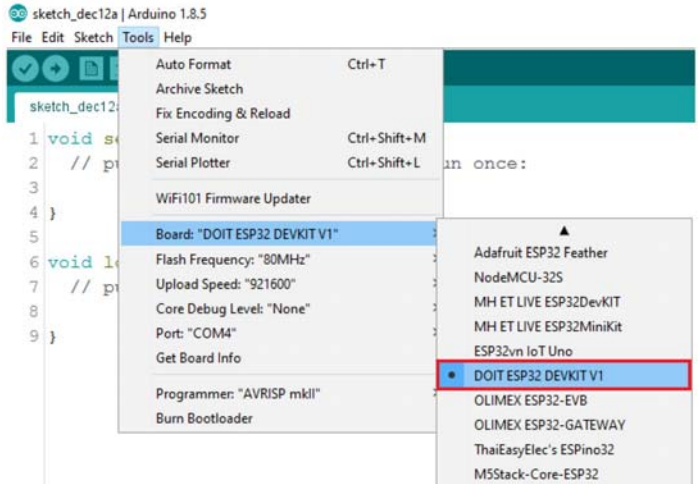
4. Når du åpner *Boards Manager* får du opp et nytt vindu, hvor du skriver *esp32* i søkevinduet, etter kort tid vil du få opp følgende:

   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.

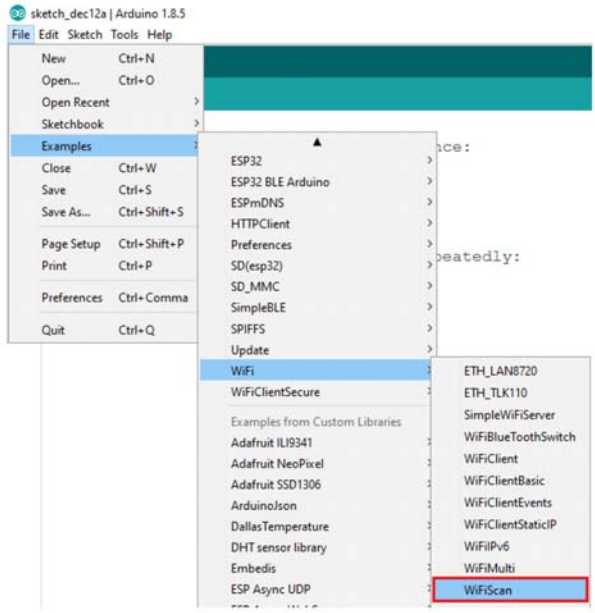
Vi skal nå teste at tillegget fungerer. Gjør følgende:

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

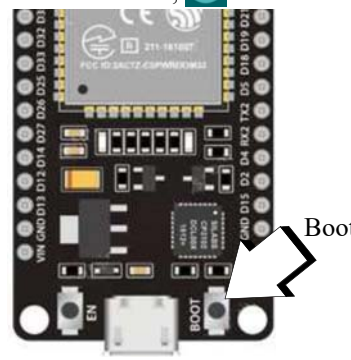
6. Gå til *Tools* på menylinjen og velg: *Tools —> Boards —> ESP32 Arduino —> DOIT ESP32 DEVKIT V1* fra lista til høyre som vist på figuren under (legg merke til at det i nyere versjoner finnes et nivå til: ESP32 Arduino). Du har nå gitt IDE'en beskjed om hvilket kort du har tenkt å bruke.



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,  :

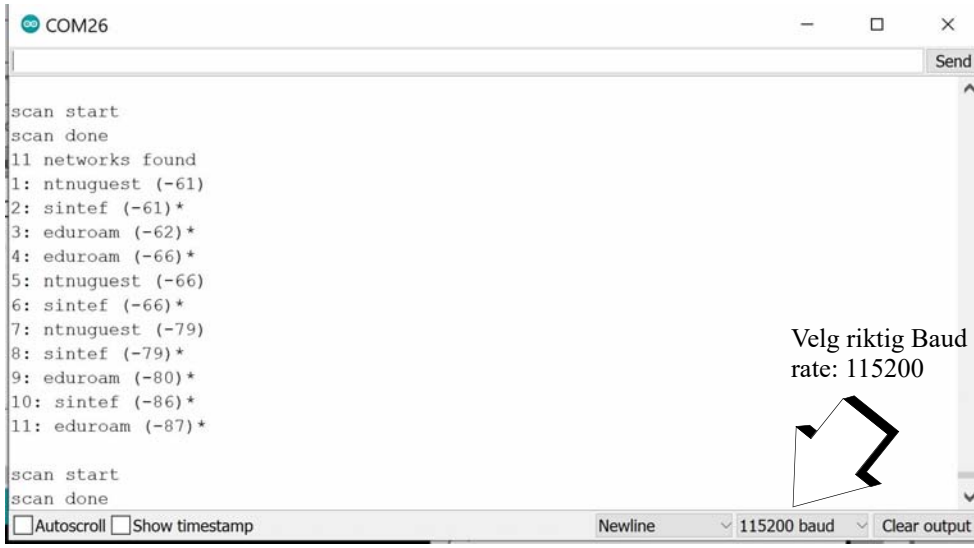


9. Kompileringen tar ganske lang tid og når opplastingen skjer er det normalt at det dukker opp rød tekst i meldingsvinduet nederst.

   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.

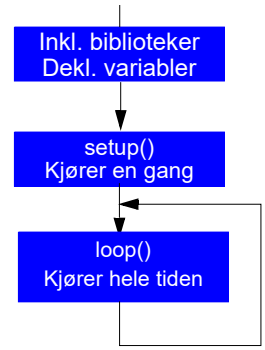


11. Du skal nå se tilgjengelig trådløse nettverk. I vårt tilfelle hele 11 stykker.

Du er nå klar til å programmere i ESP32.

4.4 Programstruktur og bruk av funksjoner

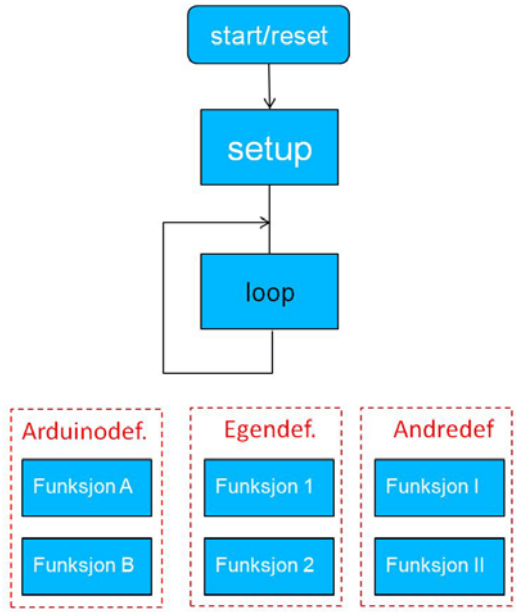
4.4.1 Programstruktur



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
 }
```



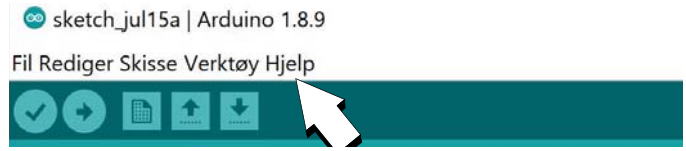
De to hovedfunksjonene i Arduino-programmene omsluttet av *klammeparenteser*. I `void setup()`-funksjonen initieres mikrokontrolleren og ev. sensorer som er tilkoblet, mens selve programmet legges under `void loop()`-funksjonen, som vist på figuren over.

`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 4.5.14, side 55.

4.5 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).

4.5.1 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[16] (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.

16. 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
```

4.5.2 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.

4.5.3 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.

4.5.4 Strukturer

Mens et *array* er en variabel som inneholder et sett av *elementer av samme type*[17] (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[18].

17.Et array kalles i noen sammenhenger også en *vektor.*
18.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[19]:

```
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 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;
```

19. 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.

4.5.5 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.

4.5.6 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()`.

4.5.7 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
```

4.5.8 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:

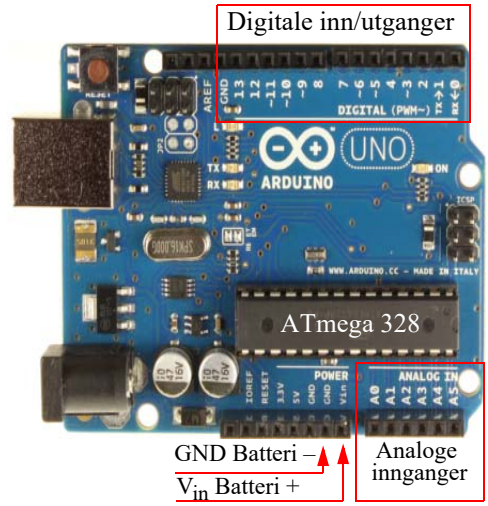
Vi kan også gjøre det motsatte. Vi kan definere en peker. En *peker* er definert som en adresse og angis med en stjerne "*". La oss deklarere følgende:

```
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.

4.5.9 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 3.1 og 3.2.

4.5.10 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.

4.5.11 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
}
```

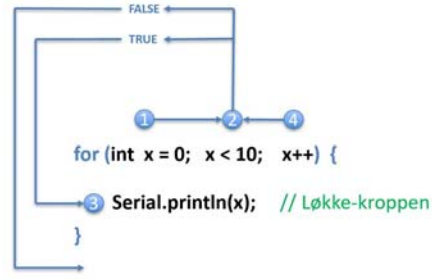
x er en heltallsvariabel (`int x`) som brukes som teller. Denne starter på verdien 0 (`int x = 0;`) økes med 1 (`x++`) for hver runde i loopen og stopper ikke før den når opp til 100 (`x < 100;`). De ulike utrykkene skilles med semikolon. Det som skal gjentas står innenfor klammeparentesene.

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[20].

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 { }.

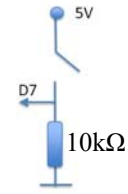
En while()-loop egner seg også for å hindre at en avlest bryter skal medføre et skred av avlesninger av bryteren.

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.

4.5.12 `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:

20. 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.

4.5.13 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.

4.5.14 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. 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`).

```
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);
    }
 }
```

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.

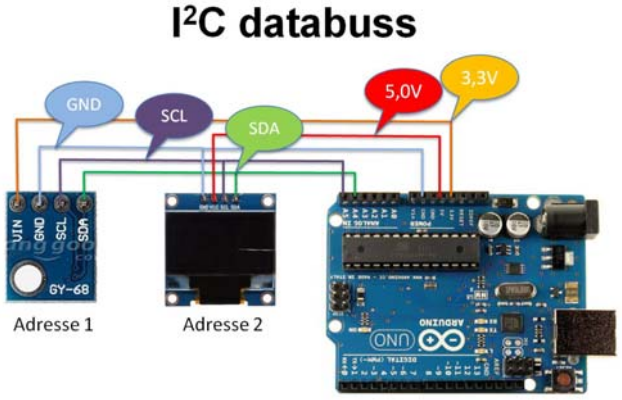
```
...
void loop()
  {
   // Det som skal gjentas hele tiden
   float volum;
   float radiusKule = 5.0;
   volum = volumKule(radiusKule);    // Kall funksjonen
   Serial.println(volum);
  }

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.

4.6 Bruk av I²C buss og biblioteker

I dette avsnittet skal se på hvordan vi kan bruke I²C-[21] 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.

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

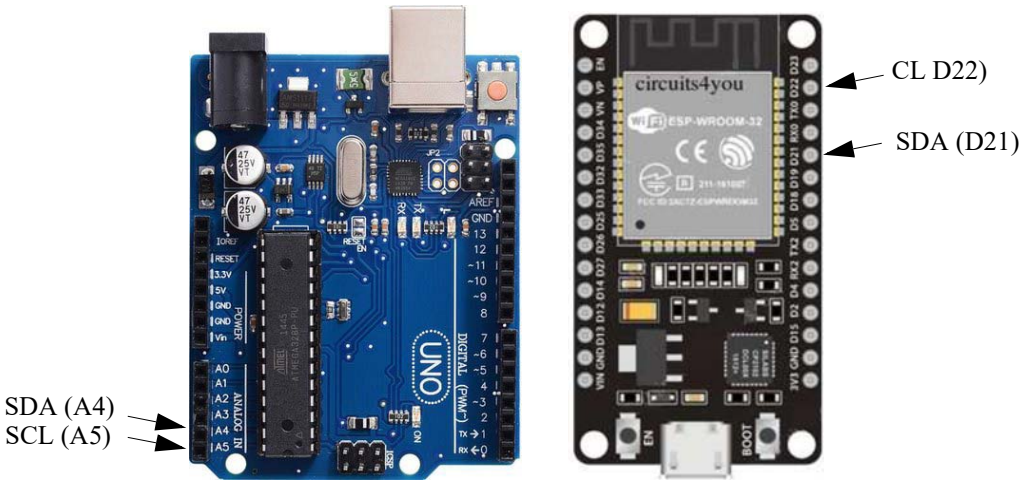
Standardbiblioteket for I^{2}C følger med Arduino programvaren, men krever at man inkluderer "header"-filen: "wire.h" øverst i koden ved å skrive:

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

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

4.6.1 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.

4.6.2 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.

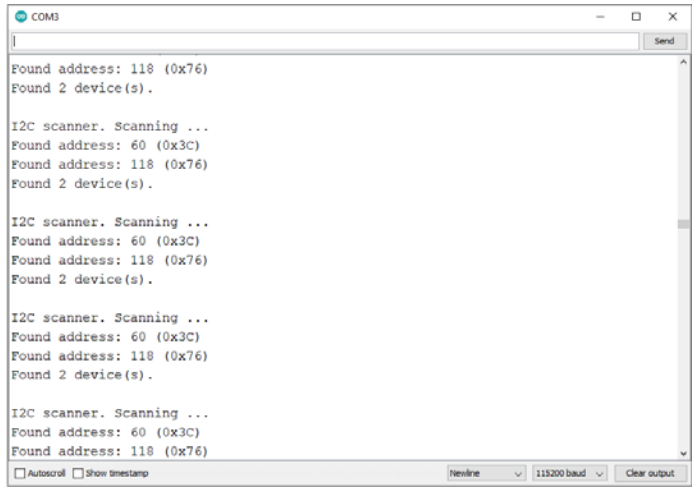
Vi skal senere se hvordan vi anvender dette i praksis.

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.



5 Oppkobling via ESP32s Wi-Fi-enhet

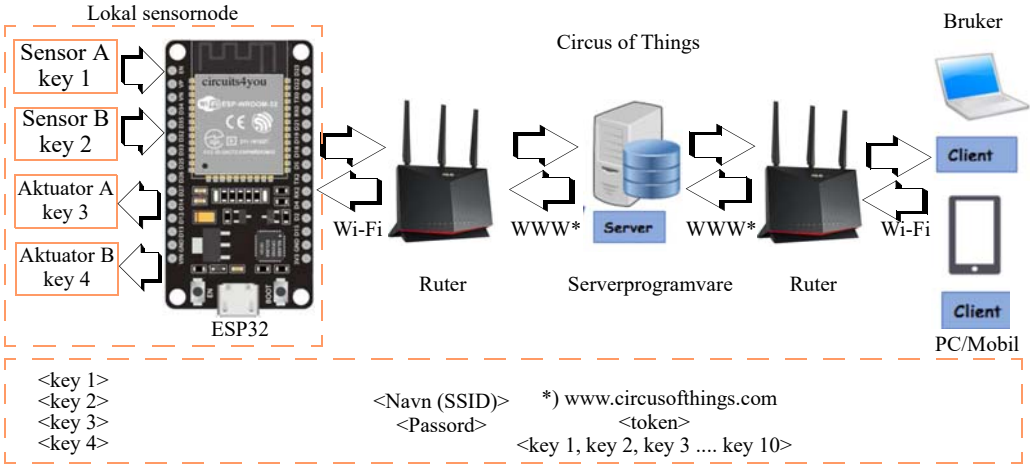
I dette kapittelet skal vi se på to måter vi kan bruke Wi-Fi-enheten til ESP32 på:

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

5.1 Circus of Things – ESP32 brukt som sensornode[22]

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.

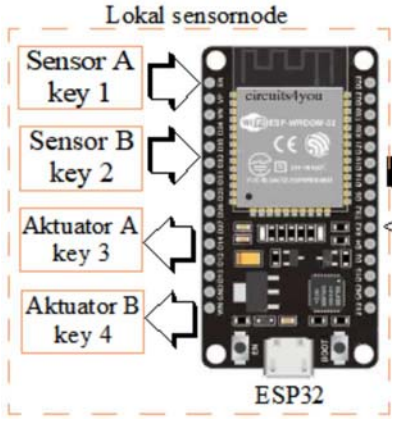
22. Dette kapittelet bygger på en presentasjon utarbeidet av Kåre-Benjamin H. Rørvik

Prosedyre for å etablere kontrollpaneler:

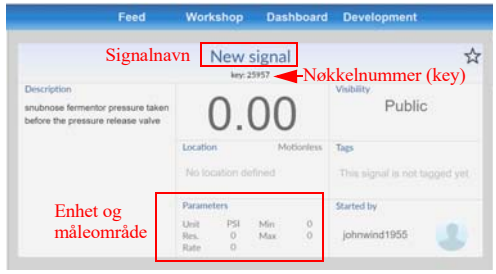
Vi skal nå gi en oversikt over hvordan vi kan lage paneler og konsoller på nettstedet Circus og Things (CoT): www.circusofthings.com. Senere skal vi se i detalj hvordan vi gjør dette.

- **Opprett bruker** med passord hos www.circusofthings.com.
  Man får da en signatur ("token"), en lang bokstav- og tallkode, som identifiserer brukeren.
  For detaljer om oppretting av konto og signatur se avsnitt 6.3.2, side 72.

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



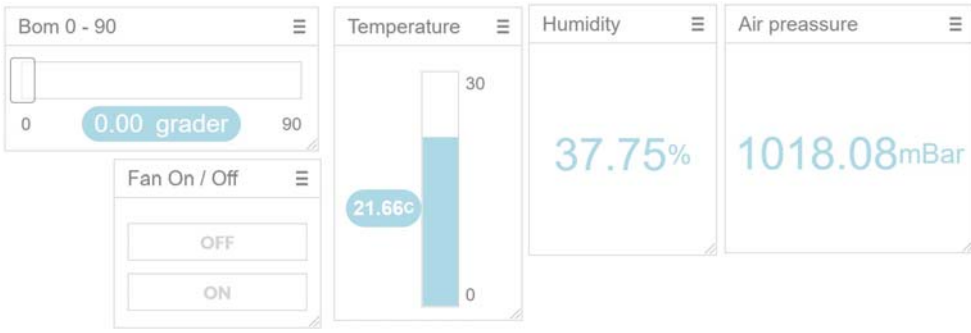
- **Gi navn til signaler:** I *"Workshop"* hos CoT spesifiseres signalnavn, enheter og måleområde som harmonerer med sensorene og aktuatorene som er koblet opp til den lokale ESP32.



Samtidig får man tildelt et nøkkelnummer ("key") knyttet til signalet. Se figur over.
Dette er omtalt i detalj i avsnitt 6.3.3, side 74.

- **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[23]. 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. For detaljer se avsnitt 6.3.4, side 77.

  - ***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.

23. 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.

5.2 ESP-NOW – Kommunikasjon direkte mellom flere ESP32[24]

ESP-NOW er et kommunikasjonskonsept hvor flere ESP32 kobler seg direkte opp mot hverandre uten først å være oppkoblet mot Internett. Formatet på dataene som overføres (protokollen) er definert av Espressif[25]. Når man skal overføre data mellom ulike ESP32 så er man avhengig av å adressere dataene slik at de kommer fram til riktig ESP32. Til dette bruker man enhetens MAC-adresse som er unik for hver enhet som kan kobles på nett.

24.https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html
25.https://www.espressif.com/en/products/software/esp-now/overview

5.2.1 MAC-adressen

Alt datautstyr som kan koble seg til nettet, det være seg PC-er, printere, nettverkskort, rutere eller ESP32 og lignende har en adresse som identifiserer dem på nettet og som går under betegnelsen MAC-adressen (Media Access Control address). MAC-adressen kalles også Ethernet-adresse, maskinvareadresse, adapter-adresse eller fysisk adresse. Vi leser på Wikipedia:

> *MAC-adressen skrives vanligvis i heksadesimalt format som seks grupper av heksadesimale tall og med to sifre i hver gruppe. Gruppene er gjerne atskilt med kolon eller bindestrek, som for eksempel 00:20:18:61:f1:8a eller 00-20-18-61-f1-8a. I eksempelet er produsentkoden 00-20-18, noe som er koden til CIS Technology Inc. Denne delen av MAC-adressen omtales også som OUI (Organizationally Unique Identifier) og administreres av IEEE (Institute of Electrical and Electronics Engineers, Incorporated).*
>
> *På noe utstyr kan man selv tildele en MAC-adresse. En slik adresse skal starte med 40 (hex).*
>
> *MAC-adresser brukes til å identifisere nettverksutstyr, og dette kan utnyttes slik at en spesiell MAC-adresse alltid skal tildeles en gitt IP-adresse. MAC-adressen kan også benyttes som identifikator for adgangskontroll til nettverk, slik at kun utstyr med gitte MAC-adresser får tilgang til et nettverk eller en tjeneste*[26].

Det er også MAC-adressen til de enkelte ESP32 som brukes for å identifisere de ulike enhetene i det lokale nettverket. Denne kan også brukes i ESP-NOW til å sende meldinger til spesifikke enheter i nettverket. For å kunne gjøre det må man kjenne de ulike enhetenes MAC-adresser.Ved hjelp av et lite program kan man lett finne enhetenes MAC-adresser. Programmet er gjengitt i vedlegg B.2, side 145.

5.2.2 Data-formatet for ESP-NOW

Informasjonen sendes digitalt på serieform som pakker eller rammer (“frames”), dvs. en lang rekker 1’er og 0’er som er ordnet i byte som igjen er ordnet i grupper hvor hver gruppe inneholder informasjon av en spesiell type. Størrelsen av en slik pakke kan være fra 57 – 562 byte avheng av hvor mye data en ønsker å overføre.

MAC Header	Category Code	Organization Identifier	Random Values	Vendor Specific Content	FCS
24 bytes	1 byte	3 bytes	4 bytes	7~255 bytes	4 bytes

Element ID	Length	Organization Identifier	Type	Version	Body
1 byte	1 byte	3 bytes	1 byte	1 byte	0~250 bytes

Normalt regner en 250 byte som det maksimale en slik pakke kan overføre av “nyttedata” (omtales som “Vendor Specific Content” i figuren over).

En ESP32 (node) i en ESP-NOW konfigurasjon vil kunne fungere på mange mulige måter:

26.https://no.wikipedia.org/wiki/MAC-adresse

- *Fra en sender til en mottaker.*
- *Fra en sender til flere mottakere. F.eks. for å sende styredata til flere mottakere.*
- *Fra flere sendere til en mottaker. F.eks. for å samle inn måledata fra ulike stasjoner.*
- *Alle i nettverket kan utføre toveis-kommunikasjon med alle de andre.*

Som vi skjønner er vi helt avhengige av å kjenne MAC-adressene til nodene i nettverket.

5.2.3 Overføring av data fra en sender til en mottaker

For å overføre et sett med data legger vi dataene inn i en struktur (se avsnitt 4.5.4 på side 45) som vist under. Vi ønsker f.eks. å overføre temperatur, luftfuktighet, lufttrykk, MAC-adressen til sensornoden og en teller som angir målenummeret. Typen struktur er kalt `struct_message` og vår spesielle struktur har vi kalt `packet`:

```
typedef struct struct_message {
  int instance;      // Nummer i rekken av målinger
  float temperature; // Måleverdi temperatur
  float humidity;    // Måleverdi luftfuktighet
  float pressure;    // Måleverdi lufttrykk
  String source;     // MAC-adresse til sensornoden
                     // som har utført målinger
} packet;
```

Den samme strukturen må defineres både på sender- og mottakersiden.

Programstruktur – sender

La oss først se hvordan vi kan bygge opp programmet på sendersiden:

1. ***Deklarasjoner og biblioteker***
   - Inkluder følgende biblioteker:
     ```
     #include <esp_now.h>
     #include <WiFi.h>
     ```
   - Definer strukturen som inneholder all informasjon skal sendes. F.eks. følgende:
     ```
     typedef struct struct_message {
       int instance;      // Nummer i rekken av målinger
       float temperature; // Måleverdi temperatur
       float humidity;    // Måleverdi luftfuktighet
       float pressure;    // Måleverdi lufttrykk
       String source; // MAC-adresse til sensornoden som har utført målinger
     } packet;
     ```
2. ***Sett opp MAC-adressen***
   Vi vet at vi må overføre MAC-adressen fra senderen slik at mottakeren kan se hvem senderen er. Denne finner vi ved å laste opp scanne-programmet for MAC-adressen og kjøre det i ESP32 på sendersiden. Programmer finnes i vedlegg B.2, side 145. Adressen legges inn i en variabel som vi deklarerer før void setup()-funksjonen:

- Deklarer og sett MAC-adresse sendersiden:
  `uint8_t broadcastAddress[] = {0xFF, 0xFF,0xFF,0xFF,0xFF,0xFF};`

3. *void setup()-funksjonen*

- *Sett Wi-Fi i stasjonsmodus:*
  `WiFi.mode(WIFI_STA);`
- *Initier ESP-NOW:*
  `esp_now_init();`
- *Angi Callback-funksjonen:*
  Initier og gi funksjonen som skal kalles når data er sendt via ESP-NOW et navn:
  `esp_now_register_send_cb(activateOnCallback);`
  Denne typen funksjoner går under betegnelsen *Callback-funksjoner* og har i vårt eksempel fått navnet: `activateOnCallback`.
- *Angi peer-informasjon:*
  Dernest defineres en instans `peerInfo` av typen `esp_now_peer_info_t` som skal sørge for å koble de to ESP32 opp mot hverandre
  `esp_now_peer_info_t peerInfo;`
  Dernest kopierer vi senderens adresse inn i peerInfo som består av 6 heksadesimale tall:
  `memcpy(peerInfo.peer_addr, broadcastAddress, 6);`
  Likeså velges en kanal for overføringen
  `peerInfo.channel = 0;`
  Til sist velges om det skal foretas kryptering:
  `peerInfo.encrypt = false;`
  Vi velger ikke å kryptere
- *Koble sammen de to i par (peering):*
  Send over informasjon for peering av de to ESP32
  `esp_now_add_peer(&peerInfo);`
  Legg merke til argumentet, `&peerInfo`, der peer-informasjonen ligger, bl.a. adressen.

4. *void loop()-funksjonen*

På sendersiden er void loop()-funksjonen aktiv, siden det er her målingene skal fylles inn i strukturen, packet.

- *Fyll inn måleverdiene*
  ... i strukturen, `packet`, som skal sendes over til mottakeren:

```
packet.instance = <antall>;          // Nummer i rekken av målinger
packet.temperature = <temperatur>;   // Måleverdi temperatur
packet.humidity = <luftfuktighet>;   // Måleverdi luftfuktighet
packet.pressure = <lufttrykk>;       // Måleverdi lufttrykk
packet.source = <MAC-adresse>;       // MAC-adresse
```

- *Send over måleverdier*
  ... pakket i strukturen packet:
  `esp_now_send(broadcastAddress, (uint8_t *) &packet, sizeof(packet));`
  Pakken sendes over ved å peke på adressen til `&packet`.

La oss så gå over til å se på mottakeren.

Programstruktur – mottaker

La oss se hvordan vi bygger opp ESP-NOW kommunikasjon på mottakersiden:

1. ***Deklarasjoner og biblioteker***
   - *Inkluder biblioteker:*
     ```
     #include <esp_now.h>
     #include <WiFi.h>
     ```
   - *Definer strukturen*
     ... som inneholder all informasjon skal overføres. F.eks. følgende:
     ```
     typedef struct struct_message {
       int instance;      // Nummer i rekken av målinger
       float temperature; // Måleverdi temperatur
       float humidity;    // Måleverdi luftfuktighet
       float pressure;    // Måleverdi lufttrykk
       String source; // MAC-adresse til sensornoden som har utført målinger
     } packet;
     ```
     Denne må være lik den som er definert på sendersiden.
2. ***void setup()-funksjonen***
   - *Sett Wi-Fi i stasjonsmodus:*
     ```
     WiFi.mode(WIFI_STA);
     ```
   - *Initier ESP-NOW:*
     ```
     esp_now_init();
     ```
   - *Angi Callback-funksjonen:*
     Initier og gi et navn til funksjonen som skal kalles når det mottas data via ESP-NOW:
     ```
     esp_now_register_recv_cb(activateOnCallback);
     ```
     Som på sendersiden har vi gitt Callback-funksjonen navnet: `activateOnCallback`.
3. ***void loop()-funksjonen***
   I denne sammenhengen kan denne være tom, da Callback-funksjonen er den funksjonen som håndterer data og som kalles fra interne funksjoner når det er mottatt data via Wi-Fi.
4. ***Callback-funksjonen:***
   Callback-funksjonen er den funksjonen som behandler data når det detekteres at det er mottatt data. Legg merke til at denne funksjonen ikke kalles fra noen av de andre funksjonene vi ser i filen, heller ikke i void loop()-funksjonen. Callback-funksjonen kalles av interne, skjulte funksjoner som ligger i biblioteket og som aktiveres så snart det kommer data inn i mottakbufferet. Callback-funksjonen kan inneholde flere ting, men må bl.a. ta hånd om dataene:
   - *Funksjonens argumenter:*
     Vi er vant til at når vi kaller en funksjon så returneres resultatet gjennom selve funksjonkallet, ved at funksjonen har en type, det være seg `int, float` eller lignende. Callback-funksjonen returnerer mottatte data gjennom *argumentet* til funksjonen:

```
void activateOnCallback(const uint8_t * mac, const uint8_t *incomingData,
int len)
{
  // Funksjoneskroppen
}
```

Følgende argumenter returnerer ulike data:

```
const uint8_t *mac,            // Pekeren til senderens mac-adresse
const uint8_t *inncommingData, // Pekeren til oversendt struktur med data
int len                        // Antall overførte byte
```

`inncommingData` er ingen struktur, men kun måledata som ligger fortløpende etter hverandre i lageret hos mottakeren. Vi legger også merke til at argumentet inneholder bare pekerne til starten av MAC-adressen og måledataene ligger i lageret hos mottakeren. Det er kun lager-adressen til første byte vi finner i argumentet. Heltallet `len` inneholder hvor mange byte meldingen består av.

- *Ompakking av overførte data:*
  For lettere å komme til dataene våre kopieres de over i vår egen definerte struktur som vi her har kalt: `packet`. Dette gjør vi med funksjonen `memcpy();`

  ```
  memcpy(&packet, incomingData, sizeof(packet));
  ```

  Denne funksjonen kopierer data fra startadressen `incommingData` til de mottatte dataene, til adressen der vår pakkestruktur `packet` er lagt. Adressen der denne starter er angitt som: `&packet`. Dersom alt har gått som planlagt så skal alle de mottatte dataene ligge pent og pyntelig ordnet i strukturen: `packet`.

- *Behandling av mottatte data*
  Vi kan f.eks. velge å skrive ut mottatte data til monitoren. Da kan programkoden komme til å se slik ut:

  ```
  Serial.print("Data packet number: ");
  Serial.print(packet.instance);
  Serial.print(" from: ");
  Serial.println(packet.source);
  Serial.print("Temperature is: ");
  Serial.println(packet.temperature);
  Serial.print("Humidity is: ");
  Serial.println(packet.humidity);
  Serial.print("Pressure is: ");
  Serial.println(packet.pressure);
  ```

6 Oppgavesamling

I denne delen skal vi ta for oss de enkelte delprosjektene og bygge opp systemet bit for bit.

6.1 Prosjektoppgaven – Oppdraget

Som nevnt tar vi sikte på å bygge opp kompetansen gradvis gjennom å arbeide med små maskin- og programvaremoduler. Det endelige produktet (oppdraget) kan beskrives slik:

> **Oppdraget:** *Det går et rykte om at konferansesal A, som nylig har fått installert elektronisk adgangskontroll, har dårlig luftkvalitet. Vi ønsker å overvåke rommet over tid ved å måle temperatur og luftfuktighet. Vi ønsker at måleresultatene skal være tilgjengelig over nett slik at vi kan kontrollere tilstanden med jevne mellomrom fra en kontroll- og overvåknings-sentral. Vi ønsker også å studere virkningen av at ventilasjonsanlegget slås av og på. Ved å måle lufttrykket ved start og stopp av anlegget, vil vi også kunne si noe om luftgjennomstrøm-ningen i rommet. Vi ønsker også å ha mulighet til å studere hvordan temperaturen og luftfuktigheten i rommet utvikler seg over tid. Dessuten vil vi ha anledning til automatisk å slå på ventilasjonsanlegget når temperaturen overskrider en terskelverdi satt ved sentralen.*
>
> *Vi kan tenke oss å ha mulighet til å utvide systemet til å omfatte flere rom, ved at vi plasserer batteridrevne sensornoder med lavt strømforbruk i naborom. Disse skal kunne kommunisere med en sentral node som via et lokalt trådløse Wi-Fi nett (ESP-NOW) skal overføre informa-sjon til sentralen.*
>
> *Det er en selvfølge at avlåsing og åpning av rommet skal kunne styres fra sentralen.*

NB! Det må imidlertid presiseres at selv om man kun gjennomfører 4 - 5 oppdrag vil man få med deg seg det viktigste fra dette undervisningsopplegget.

6.2 Oppdrag 1 – Styring av rele hos sensornoden

I dette første oppdraget skal vi koble opp og teste ut et rele for styring av en vifte (ventilasjon).

> **Oppdrag 1:** *Koble opp viftereleet og skriv et lite program som slår releet av og på.*

6.2.1 Kort omtale av releet

Releet vi har valgt å bruke i denne oppgaven er av typen JQC-3FF-S-Z som er et enpolet rele som kan styre en strøm på inntil 10A ved en spenning på opp til 250VAC.

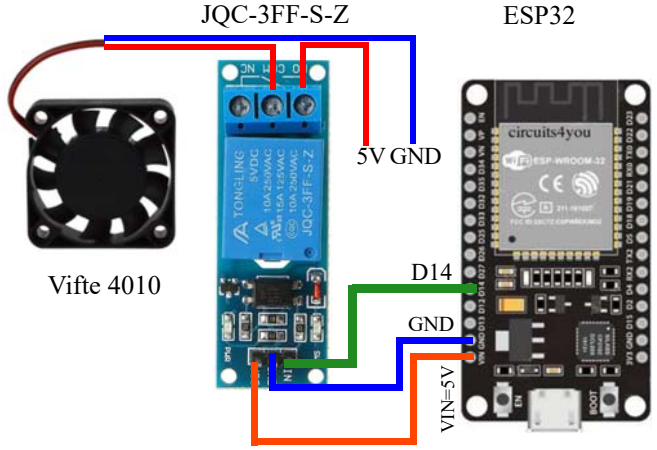
For ikke å belaste ESP32s 3,3V spenningsregulator for mye, har vi valgte et rele med drivspenning 5V (V_{CC}). Tester har vist at det likevel lar seg styre (V_{IN}) med et digitalt høyt nivå på 3,3V fra en av ESP32s digitale utganger. Releet slutter med aktiv høy, det betyr at et høyt nivå (3,3V) på V_{IN} kobler til rele-kontakten.

6.2.2 Oppkobling

Figuren under viser pinningen til releet JQC-3FF-S-Z:

Figuren under viser oppkoblingen av releet. Vi bruker tre korte hann – hunn jumpere.

I tillegg vil vi gjerne koble opp vifta som trenger 5V og jord. For å få til det må vi ta i bruk koblingsbrettet. Siden vi skal bruke koblingsbrettet til å koble opp sensoren BME280 (ev. BMP280) og servoen, så må vi være forberedt på å flytte litt rundt på jumperne etter som vi bygger opp systemet.

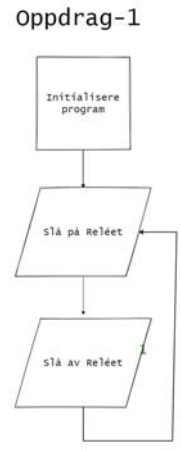


6.2.3 Programmet

Programmeringen av kretsen kan være særdeles enkel siden det ikke trengs å installeres noe bibliotek. Vi må bare huske følgende:

- ... å definere D14 som utgang
- ... at releet slår inn når VIN er høy

Det er tilstrekkelig at testprogrammet slår releet inn og ut med 1 sekund mellomrom. Figuren til høyre viser et flytdiagram for programmet vårt.



Den som vil lese mer om releer og styring av større strømmer, kan se avsnitt 7.2, side 132.

6.2.4 Løsningsforslag oppdrag 1

I løsningsforslaget er det lagt inn en utfordring eller feil som må finnes.

START LØSNINGSFORSLAG-1

```
/*
Oppdrag-1
*/

int RelayPIN = 14;

void setup() {
  pinMode(RelayPIN, OUTPUT); // Define our pin for the relay as an output
}

void loop() {
  digitalWrite(RelayPIN, HIGH);
  delay(1000);
  digitalWrite(RelayPIN, LOW);
}
```

STOPP LØSNINGSFORSLAG-1

6.3 Oppdrag 2 – Styring av rele via Internett og "Circus of Things"

I dette oppdraget skal vi koble ESP32 opp mot Internett og slå av og på rele via nettet ved å ta ibruk nettresursen Circus of Things.

Oppdrag 2: *Slå releet av og på via Internett ved hjelp av nettressursen Circus of Things*

Dette er et ganske sammensatt oppdrag hvor vi først må skaffe oss en bruker hos Circus of Things, konstruere et enkelt brukergrensesnitt (panel) og opprette forbindelsen. Vi tar det derfor trinnvis.

Det forutsettes at man har tilgang til et trådløst nettverk med kjent navn og passord.

NB! ***Vær klar over at tradisjonelle skolenettverk kan sette begrensninger på den type oppkobling vi ønsker. Derfor anbefales å koble seg opp med egen mobiltelefon eller bruke hjemmenettverket for utprøving. For bruk i undervisningen anbefales å ta kontakt med IT-avdelingen slik at det kan opprettes et eget laboratorienettverk med færre restriksjoner.***

6.3.1 Oppkobling

Om man har gjort den foregående oppgaven så skal kretsen nå være ferdig oppkoblet med releet for styring av vifta.

6.3.2 Opprett bruker på Circus of Things

Circus of Things (CoT) er en open-source IoT felles plattform for datalogging, overføring og utveksling av data. CoT kan fritt brukes for utviklings-, utdannings- og næringslivsformål. Plattformen gir også rike muligheter for å utveksle data med andre i fellesskapet.

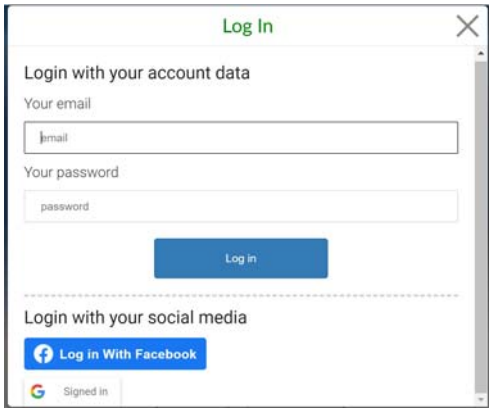
Plattformen gjør det mulig å lage enkle brukergrensesnitt, etablere trådløs kontakt mellom en sensornode med f.eks. ESP32 og andre, se figuren til høyre. Kontakten kan gå begge veier fra noden til en PC, Smart phone eller nettbrett og vise versa. Innsamlede data kan vises i "real time" som tall eller grafer, og aktuatorer kan fjernstyres ved hjelp av software-brytere og glidere for å justere fart eller lysstyrke.

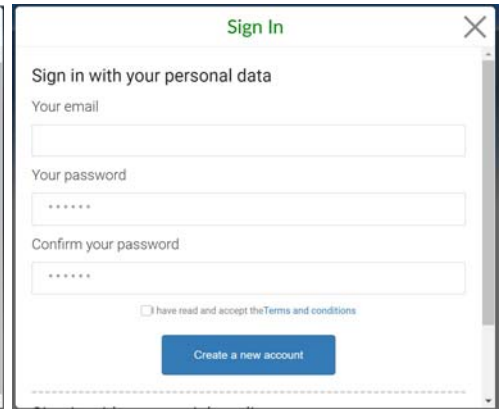
Plattformen gjør det også mulig å dele design og data. Hvem som helst kan gå ut å observere åpne data som er lagt ut av medlemmer av fellesskapet, eller man kan la dataene være private eller dele dem med et utvalg partnere.

CoT ble etablert og drives av ***Jaume Miralles*** som holder til på Mallorca, Spania og kan kontaktes via info@circusofthings.com[27].

Pålogging / Signing in – Opprett en personlig konto

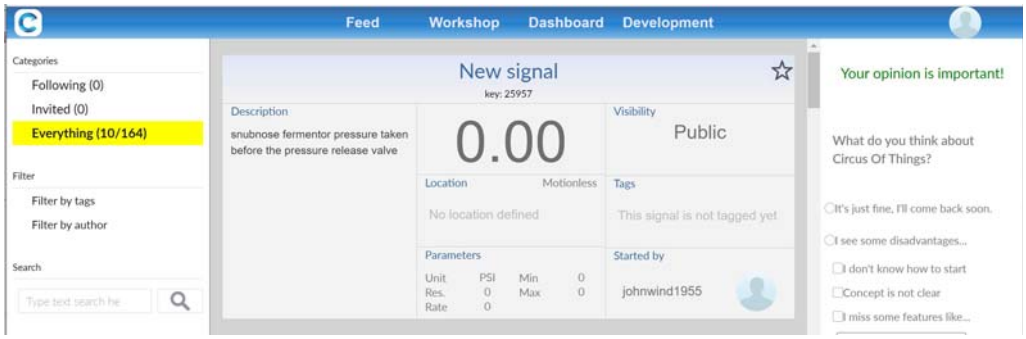
Gå til https://circusofthings.com/welcome og velg "*Sign in*" dersom du ønsker å opprette en egen konto.Velg "*Log in*" dersom du har en konto eller ønsker å logge inn via din Google- eller Facebook-konto.





27.https://circusofthings.com/welcome#about

Velg f.eks. *Sign in* og skriv inn e-post og et unikt passord. Du får da en e-post med en nettadresse for å bekrefte at du ønsker å opprette en bruker. Deretter blir du logget på og blir møtt med følgende vindu:

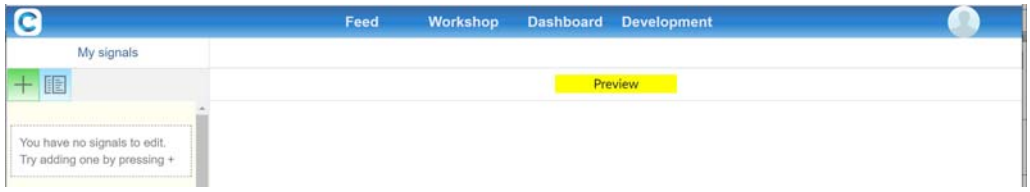


Øverst (blå linje) finner vi fire valg: *Feed*, *Workshop*, *Dashboard* og *Development*.

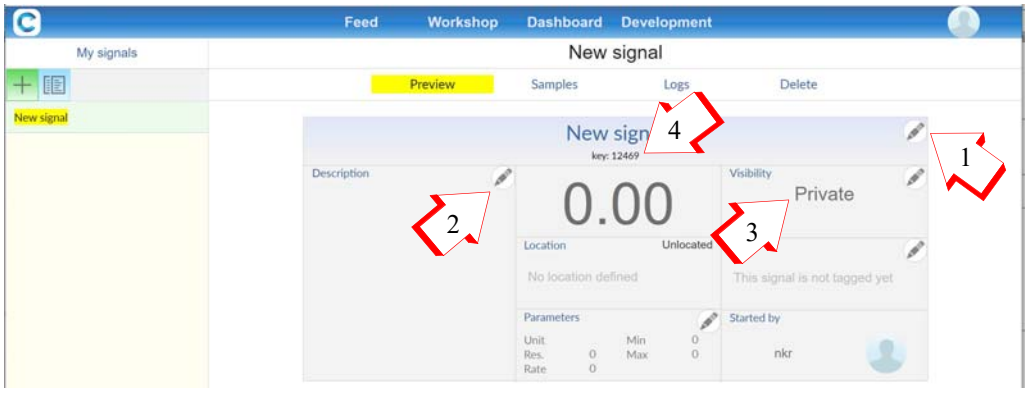
6.3.3 Kontrollpanelet

Vi ønsker å lage et konsoll for å kunne slå av og på releet for å starte ventilasjonsanlegget. Vi går da fram på følgende måte:

1. Velg *Workshop*. Dette er stedet hvor vi kan bygge opp konsollene våre som i første omgang vil være en på/av-knapp for å styre releet. Vi får opp følgende vindu.:



2. Vi skal nå legge til det signalet vi ønsker å overføre. Vi velger + lengst oppe til venstre og får opp følgende vindu hvor vi kan fylle inn informasjon om signalet vårt.

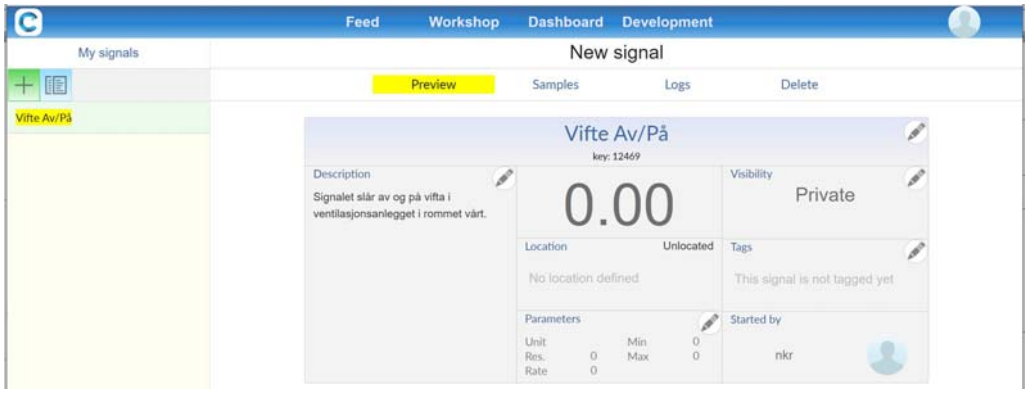


Her fyller vi ut følgende:

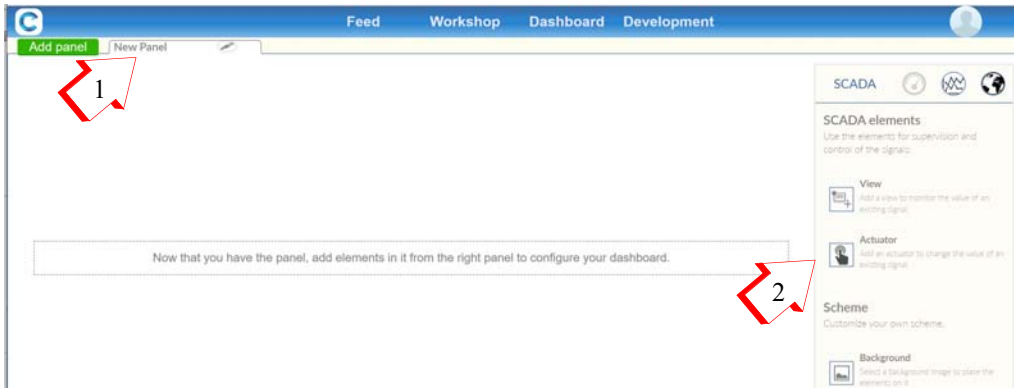
- **Gi signalet et navn (1)** ved å klikke på blyanten til høyre for *New Signal*. Vi får da opp følgende vindu:

  Vi velger å kalle signalet vårt *Fan On / Off* og trykker så ***done*** nederst i vinduet.

- **Beskriv signalet (2).** Her kan man lage en beskrivelse av hvilken type signal dette er og hva det formidler av informasjon. I vårt tilfelle skal det slå av releet som styrer vifta til ventilasjonsanlegget.
- **Privat eller offentlig (3).** Her kan vi velge å gjøre signalet tilgjengelig for offentligheten om vi ønsker det. Vi velger foreløpig å la det være privat.
- **Kopier nøkkel (4):** Når vi oppretter et nytt signal vil dette få en nøkkelkode (key), dvs. et nummer som vi kan bruk til å opprette forbindelse mellom signalet i programmet hos vår ESP32 og konsollet ved serveren hos CoT. Vi bruker det også når vi skal velge signaler til bryterkonsollene på *Dashbordet* vårt. Vi kopierer derfor nøkkelkoden (12469) som vil være unik for hver bruker. I det vi forlater Workshop'en ser bildet slik ut:

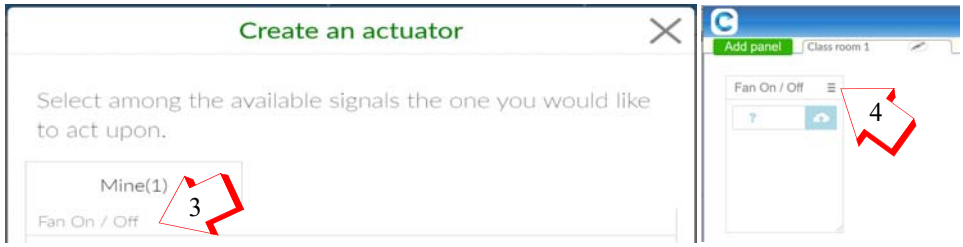


3. **Gå til Dashbordet** og få opp følgende vindu:

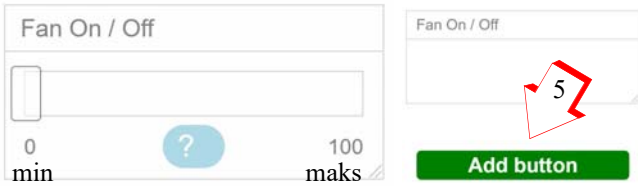


Vi skal nå lage et panel og bestykke det med konsoller, brytere for styring av aktuatorer og etter hvert displayer for å vise sensordata.

- **Legg til et panel (1):** Her kan vi legge til ett nytt panel ved å velge *Add panel*. Siden det alt ligger et ubrukt panel med Navnet *New Panel* så velger vi å gjenbruke dette og gi det et passende navn. Det gjør vi ved å velge blyanten til høyre for *New Panel*. Vi velger å kalle det *Class room 1*.
- **Legg til aktuator (2):** Vi ønsker nå å lage et konsoll som vi plasserer på panelet, for å styre releet som slår av og på vifta til ventilasjonsanlegget. Vi går da til der det står *Aktuator* til høyre for panelet (2). Når vi klikker på *Aktuator* kan vi velge vårt nylagde konsoll *Fan On / Off* under *Mine* (3).

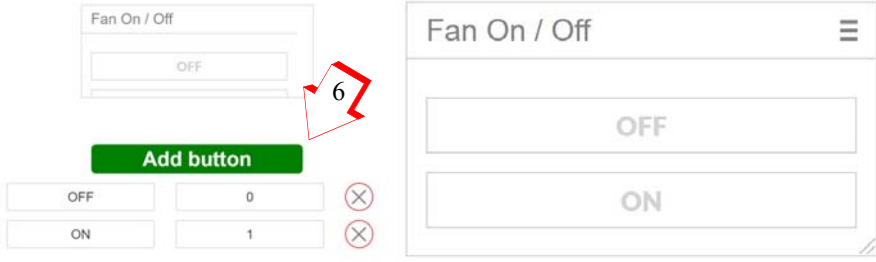


- **Definer konsollets knapper:** Når vi velger Fan On/Off får vi opp konsollet vårt, men foreløpig uten knapper. Opp til høyre på konsollet er det et meny-ikon (4). Her kan vi velge mellom *Add Button* (knapper) (On/Off) eller glider (f.eks. 0 – 100) som kan være aktuelt dersom vi f.eks. ønsker å regulere farten kontinuerlig til ei vifta eller lystyrken til en lyskilde. Vi velger knapper (5).



Sett også tallverdier for min. og maks.

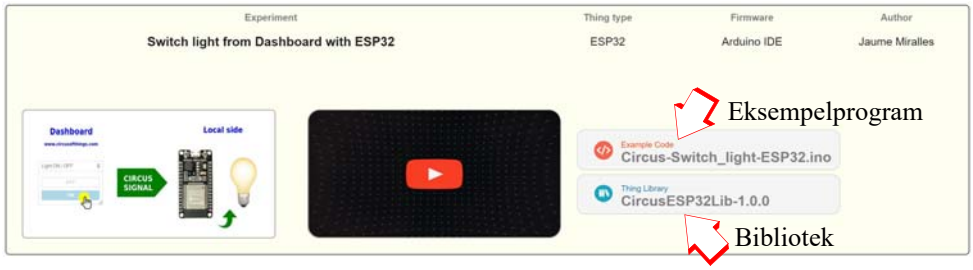
- **Legg til knapper (On og Off) (6):** Vi velger å lage et konsoll med knappene "On" og "Off". Det gjør vi ved å bruke knappen *Add button* to ganger. Vi må skrive inn navnene på knappene "On" og "Off" samtidig som vi må definere hvilken logisk tilstand de representerer. Vi velger "On" lik "1" og "Off" lik "0" som vist til venstre på figuren under. Når vi trykker *Save* nederst til høyre i vinduet, kommer konsollet opp i panelet vårt. Ved å strekke ut rammene får vi se begge bryterne som vist til høyre på figuren under. Vi kan også ombestemme oss ved å trykke *Remove actuator* nederst til venstre i vinduet.



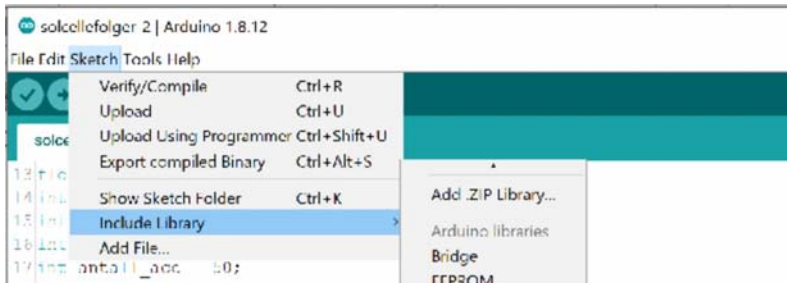
6.3.4 Programmet

Vi skal nå lage programmet som skal lastes opp på ESP32 som er plassert lokalt der releet vårt er. Både *eksempelprogram* og nødvendige *biblioteker* ligger på hjemmesiden til www.circusof-things.com. Vi finner begge deler under fanen *Development* lengst til høyre. Da får vi opp en meny med fem valg.

Velger vi *Experiment* vil vi få en oversikt over en rekke eksempler hvor vi finner programkoder, nødvendig biblioteker og instruksjonvideoer som også viser nødvendig utstyr og oppkoblinger. Vi velger en video som anvender ESP32 for å fjernstyre tenning og slukking av en lampe, denne passer godt til det vi skal gjøre. Figuren under viser fanen til det valgte eksempelet.

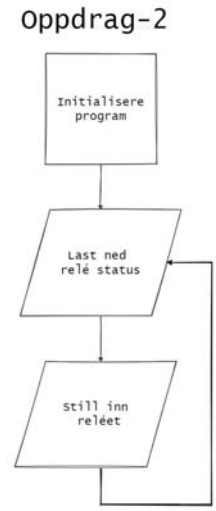


1. **Last ned biblioteket** som zip-fil og legg det et sted hvor det er lett å finne det igjen
2. **Last ned eksempelfilen** og legg den et sted der den er lett å finne igjen
3. **Installer biblioteket.** Biblioteket installeres i IDE på vanlig måte. Gå til Sketch/Include/ Library/Add .ZIP Library ...



Velg og installer det nedlastede biblioteket.

Eksempelprogrammet



Vi skal nå skrive programmet for ESP32 som sender data over til CoT. Figuren til høyre viser et enkelt flytdiagram for programmet. Vi tar utgangspunkt i programmet vi fra oppdrag 1.

1. **Inkluder biblioteket.** Det første vi må gjøre er å inkludere det aktuelle biblioteket i programmet. Det gjør vi med følgende *#include* kommando øverst i programmet:

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

2. **Etabler forbindelsen mellom ESP32.** Vi skal nå etablere forbindelsen mellom vår ESP32, via den lokale ruteren, nettverket og til vårt konsoll *Fan On/Off* ved serveren hos Circus of Things. Vi begynner med å identifisere vår lokale ruter, f.eks. *Wireless* med passord *mikro*.

   - Ruterens navn SSID:

```
  char ssid[] = "Wireless";
// Her setter du inn navnet på din ruter, husk at " " skal være med
```

   - Ruterens passord:

```
 char password[] =  "mikro";
// Her legger du inn passordet til din ruter
```

   - Serveren til Circus of Things

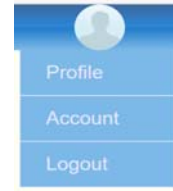
```
  char server[] = "www.circusofthings.com";
// Her legges nettadressen til Circuit of Things inn som vist
```

   - Nøkkelkoden til konsollet vi har laget (du må sette inn *din* nøkkelkode):

```
  char order_key[] = "27094"; // Fan On / Off
```

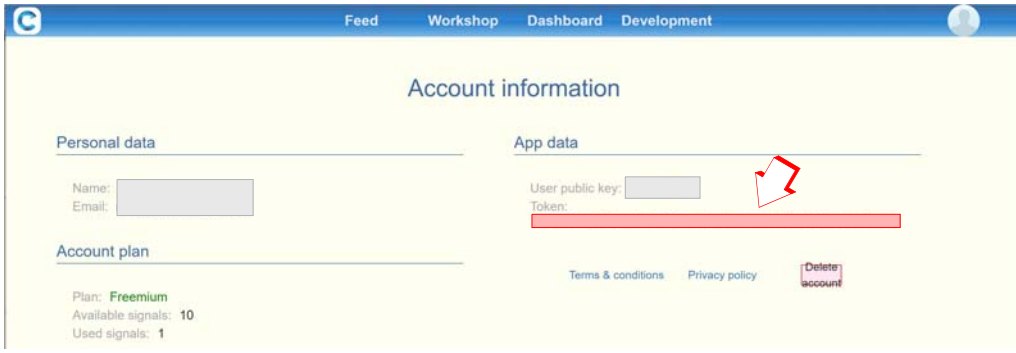
   - Sett så inn din identitet hos Circus of Things:

```
  char token[] = "plasser_din_identitet_hos_CiT_her";
```



Denne finner vi under vår personlige *Account*. Velg "bilde" øverst i høyre hjørne og velg *Account* (se figur til høyre). Koden er svært lang.

Åpner vi denne får vi opp følgende vindu:



Vi finner den aktuelle identiteten under betegnelsen *token*. Denne limes inn istedet for `plasser_din_identitet_hos_CiT_her` som vist over. Husk at den skal være omsluttet av hermetegn.

- Lag et objekt av typen: CircusESP32Lib vi har valgt å kalle det: circusESP32( ). Argumentet til dette objektet skal inneholde server-navn (`server`), SSID (`ssid`) og passord (`password`) som vi tidligere har fylt med innhold. Vi skriver da:

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

Alt dette skal plasseres foran `setup()`-funksjonen

3. **Definer globale variabler**
   - Gi et navn til porten som styrer releet. Vi valgte port `D14` da vi koblet opp kretsen.
4. **Initialisering i void setup()-funksjonen**
   - I void setup()-funksjonen husker vi å initialisere objektet som knytter ESP32 opp mot serveren. Det gjør vi slik:

```
circusESP32.begin();
```

   Husk ev. andre ting som vanligvis ligger i setup()-funksjonen.
5. **Etabler void loop()-funksjonen** og utfør kommando for styring av rele

Det siste vi nå skal gjøre er å lese av verdien som skal styre releet og som er overført via nettet. Dette gjør vi med følgende kommando:

```
double dashboard_order_relay =
circusESP32.read(order_key_relay,token);
```

Her leser vi fra det som oversendes fra serveren hos Circus of Things og velger riktig kode for konsollet `order_key_relay` (12469), og med riktig personlig identitet (`token`).

Vi må teste på den oversendte verdien `dashboard_order_relay` og sjekke om den er "1", i så fall skal releet kobles inn. Er den derimot "0" skal releet kobles ut.

6. **Ferdigstill programmet** og test det.

   Sjekk at programmet oppfører seg som forventet.

6.3.5 Løsningsforslag oppdrag 2

I løsningsforslaget er det lagt inn en utfordring eller feil som må finnes.

_____START LØSNINGSFORSLAG-2_____

```
/*
Oppdrag-2
*/

//Inkluderer biblioteket for å kommunisere med Circus of Things
#include <CircusESP32Lib.h>  // Biblioteket til Circus of Things.

// ---------------------------------------------------------------------
// CircusESP32Lib relaterte deklarasjoner for oppkobling til WiFi og CoT
// ---------------------------------------------------------------------

char ssid[] = "ssid"; // Skriv inn navnet til ruteren din her
char password[] = "password"; // Skriv inn passordet til ruteren din her
char token[] = "token"; // Plasser din brukeridentitet her (token)
char server[] = "www.circusofthings.com"; // Her ligger serveren
char order_key_relay[] = "1234"; // Nøkkel-informasjon om konsollet for styring
av relet hos CoT
CircusESP32Lib circusESP32(server,ssid,password);// Her leses nettadressen til
CoT, ruternavn og passord inn.

// -----------------------------------------------
// Deklarerer en konstant som holder portnummeret for oppkobling til rele
// -----------------------------------------------

#define RELAYPIN 14 // Digitalt utgang for ESP32 for styring av vifterele.

// -----------------------------------------------

void setup() {
    pinMode(RELAYPIN, INPUT);
    Serial.begin(115200); // Setter datahastigheten mellom ESP32 og PC.
    circusESP32.begin(); // Initialiserer oppkobling mot CoT
    Serial.println("Oppdrag-2");
    // Skriv ut hvilket program som ligger i mikrokontrolleren
}
```

```
void loop() {
    delay(1000);

    // Mottar styreinformasjon til reléet fra Circus of Things.
    int dashboard_order_relay = circusESP32.read(order_key_relay,token);

    // Reléet bruker verdien som hentes fra CoT, og styrer reléet direkte med den.
    digitalWrite(RELAYPIN, dashboard_order_relay);
}
```

6.4 Oppdrag 3 – Avlesning av sensoren BME280 (ev. BMP280) hos sensornoden

Så skal vi koble opp og teste BME280 eller BMP280 (avhengig av hva som er tilgjengelig).

> **Oppdrag 3:** *Koble opp BME280 (ev. BMP280) og skriv ut måleverdiene for luftfuktighet (bare for BME280), temperatur og lufttrykk til monitoren i IDE (Arduino editoren).*

6.4.1 Kort omtale av BME280

Dette er en utvidelse av BMP280 hvor man også har inkludert måling av fuktighet. Dette kvalifiserer tydeligvis for å bytte ut P (Pressure) med E (*Environment*). Hvilket gir løfte om en sensor som i større grad en BMP-serien gir et mer fullstendig "bilde" av miljøet. Sensoren er spesielt beregnet for mobile anvendelser med sitt lave strømforbruk. Til vårt formål så kan vi like godt bruke BMP280 da vi ikke er avhengig av å måle luftfuktigheten for å bli kjent med CoT. Pinningen for de to kretsene er også lik.

Her er noen nøkkeldata[28]:

- *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)

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

- *Temperatursensor:*
  Måleområde: -40 – +65°C
  Nøyaktighet: ± 1°C (0 – 65°C)
- *Luftfuktighetssensor (bare for BME280):*
  Responstid: 1 sek.
  Nøyaktighet: ± 3% mellom 20 – 80% relativ fuktighet.

For mer informasjon se avsnitt 7.1, side 128.

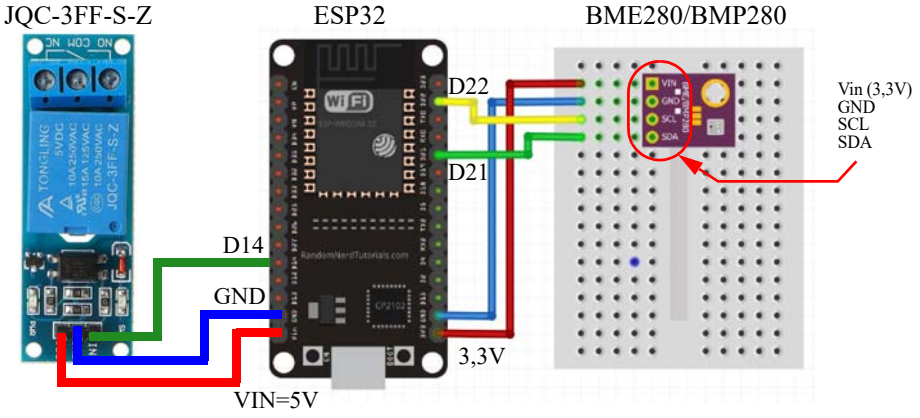
6.4.2 Oppkobling

Figuren under viser pinningen til to utgaver til BME280/BMP280:

Top Bottum Top Bottum

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 (Top) av kretsen. Vi legger imidlertid merke til at teksten henholdsvis er plassert på undersiden og oversiden.

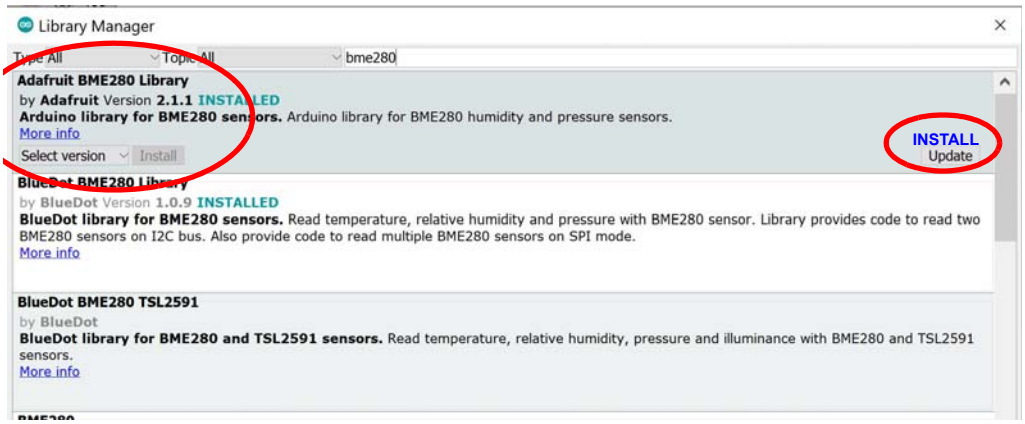
Figuren under viser oppkoblingen av BME280/BMP280 til høyre.



6.4.3 Programmet

For å løse dette oppdraget må vi installere et bibliotek som gjør det enkelt å kommunisere med BME280/BMP280. Vi velger å hente ned biblioteket fra Adafruit:

For å installere biblioteket til BME280/BMP280 går vi til menyen *Sketch/Include Library/ Manage Library* og skriver BME280 i søkefeltet (ev. BMP280). Ganske snart dukker det opp flere alternativer. Velg det øverste, Adafruit BME280 som vist på figuren under:

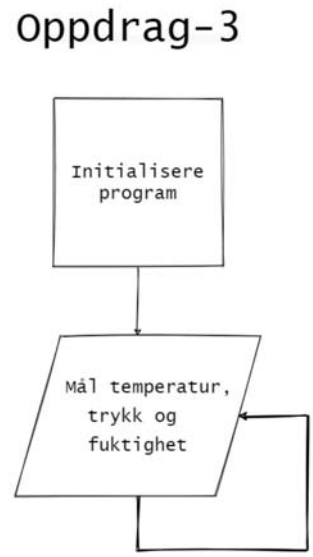


Alternativt vil vi kunne få opp biblioteket til Adafruit BMP280.

Velg **INSTALL** og biblioteket installeres. Lukk vinduet.

Vi har nå de funksjonene vi trenger for å kommunisere med BME280, alternativt BMP280.

Figuren til høyre viser et enkelt flytdiagram for programmet.



1. **Inkluder bibliotekene** ved hjelp av følgende:

```
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h> // eller ...
//#include <Adafruit_BMP280.h>
```

   Vi legger merke til at vi inkluderer to biblioteker. Disse to kommandoene plasseres helt i toppen av programmet.

2. **Definer et objekt:**
   Vi definerer objektet `bme280` av typen `Adafruit_BME280` slik:

```
Adafruit_BME280 bme280; // eller ...
// Adafruit_BMP280 bmp280;
```

   Dette kan godt stå under inkluderingen av bibliotekene.

3. **Initialiser biblioteket:**
   I initialiseringen av biblioteket til BME280/BMP280 må vi inkludere adressen til I^2C-bussen som vi bruker til å kommunisere med BME280/BMP280:

```
bme280.begin(0x76);
//bmp280.begin(0x76);
```

   Initialiseringen skal plasseres i setup()-funksjonen. Vi legger også merke til I^2C-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 4.6.2, side 59.

4. **Avlesing av dataene**
   Følgende kommandoer leser av sensorene og lagrer dem i en variabel av typen float:

```
float temperature = bme280.readTemperature();
float preassure = bme280.readPressure()/100.0F;
float humidity = bme280.readHumidity();
```

   Alternativt for BMP280:

```
float temperature = bmp280.readTemperature();
float preassure = bmb280.readPressure()/100.0F;
```

   Denne avlesningen gjøres i loop()-funksjonen. F plassert etter 100.0 sørger for at vi ender opp med at divisjonen ender opp med et desimaltall (float).

5. **Skriv til monitoren**
   Vi vil at dere skrive ut de tre verdiene til monitoren og skriver programmet slik at utskriften blir på formen:

```
---------------------
Temperatur: 20.42 *C
Trykk: 992.07 hPa
Fuktighet: 40.16 % // Bare for BME280
---------------------
```

6. **Egendefinerte utskriftsfunksjon**

   Selv om det kanskje virker overdrevent vil vi nå at dere skal lage en egendefinert funksjon med navnet:

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

   Denne skal lese av de tre (to) sensorverdiene og skrive dem ut til monitoren som beskrevet foran.

   Når dere ønsker å skrive ut verdiene kaller dere bare opp funksjonen som leser og skriver ut verdiene. Kallet av funksjonen ser slik ut:

```
sensorValuesToMonitor();
```

   Kallet skal stå i loop()-funksjonen, mens selve funksjonen defineres utenfor loop()-funksjonen. Gjerne nedenfor.

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

6.4.4 Løsningsforslag oppdrag 3

I løsningsforslaget er det lagt inn en utfordring eller feil som må finnes.

NB! Vær oppmerksom på at bme280 må byttes ut med bmp280 dersom denne brukes i stedet.

START LØSNINGSFORSLAG-3

```
/*
Oppdrag-3.ino
*/

#include <Adafruit_BME280.h>  // Inkluder biblioteket for BME280.

// Lager et Adafruit_BME280 objekt kalt bme280.
// Slik får vi tilgang på funksjonene tilhørende objektet.
Adafruit_BME280 bme280;

#define RELAYPIN 14 // Digitalt utgang for ESP32 for styring av vifterele.

void setup() {

  Serial.begin(115200);         // Setter datahastigheten mellom ESP32 og PC.
  bme280.begin(0x76);           // Initialiserer BME280 sitt bibliotek og legger
                                // inn I2C bussadressen (0x76).
  pinMode(RELAYPIN, OUTPUT);
  Serial.println("Oppdrag-3"); // Skriv ut hvilket program som ligger i
mikrokontrolleren.
}

void loop() {
  // Leser av sensoren og skriver til monitor, se funksjonsbeskrivelsen lengre ned.
  sensorValuesToMonitor();
}

void sensorValuesToMonitor() {
  // bme280 objektet måler temperatur, trykk og fuktighet.
  float temperature = bme280.readTemperature();
  float pressure = bme280.readPressure()/100.0;
  // Deler på 100.0 og får da hPa og mBar, vanlig benevning for atmosfærisk trykk.
  float humidity = bme280.readHumidity();
  // Denne må ev. fjernes ved bruk av BMP280 i stedet for BME280

  Serial.println("---------------------");
  Serial.print("Temperatur: ");
  Serial.print(temperature);
  Serial.println(" *C");
```

```
  Serial.print("Trykk: ");
  Serial.print(temperature);
  Serial.println(" hPa");

  Serial.print("Fuktighet: ");
  Serial.print(humidity);
  Serial.println(" %");

  Serial.println("---------------------");
  Serial.println();
}
```

STOPP LØSNINGSFORSLAG-3

6.5 Oppdrag 4 – Avlesing av sensoren BME/P280 hos sensornoden via Internett

I oppdrag 2 viste vi hvordan vi kan fjernstyre et rele ved bruk av Circus of Things via Internett. Her skal vi se hvordan vi kan overføre sensordata avlest ved en *sensornode* (ESP32) ved bruk av CoT via Internett.

Oppdrag 4: *Les av sensorene og presenter dem i panelet hos Circus of Things.*

Oppdraget krever også at en bygger opp tre (to) nye konsoller som viser de avleste verdiene.

6.5.1 Oppkobling

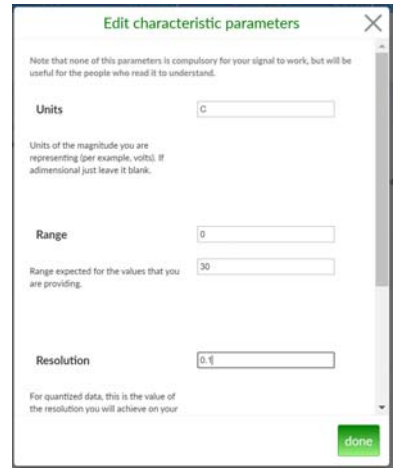
Om man har løst de foregående oppdragene så skal kretsen nå være oppkoblet med sensoren BME/P280 og releet for styring av vifta.

6.5.2 Kontrollpanelet

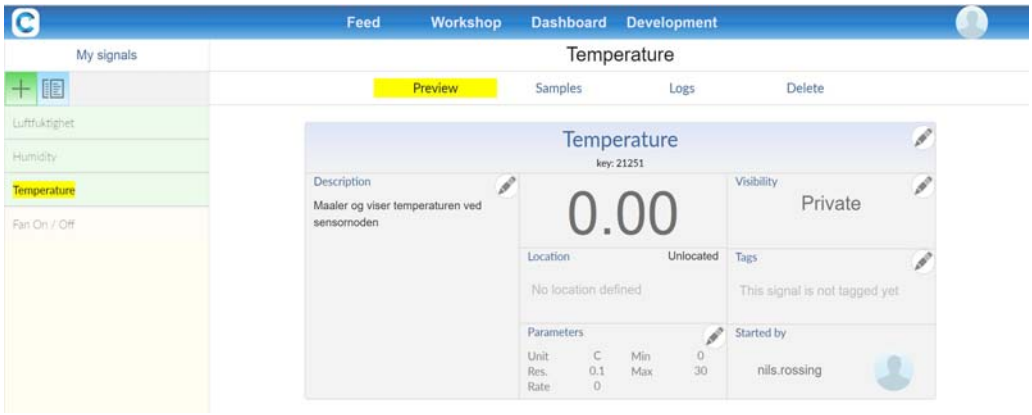
Som sist må vi "lage" konsollet som viser de avleste sensordataene.

1. **Definer signaler:** Gå til *workshop* og legge til et *nytt signal (+)* for visning av temperatur. Legg også inne en beskrivelse av hva signalet er: *Måler og viser temperaturen ved sensornoden.*

   Vi velger også å legge inn verdier i *Parameters* menyen (nederst midt på). Her legger vi inn enheter (*Units*) lik C, område (*Range*) 0 – 30°C og oppløsning (*Resolution*) lik 0.1 som vist på figuren over til høyre.

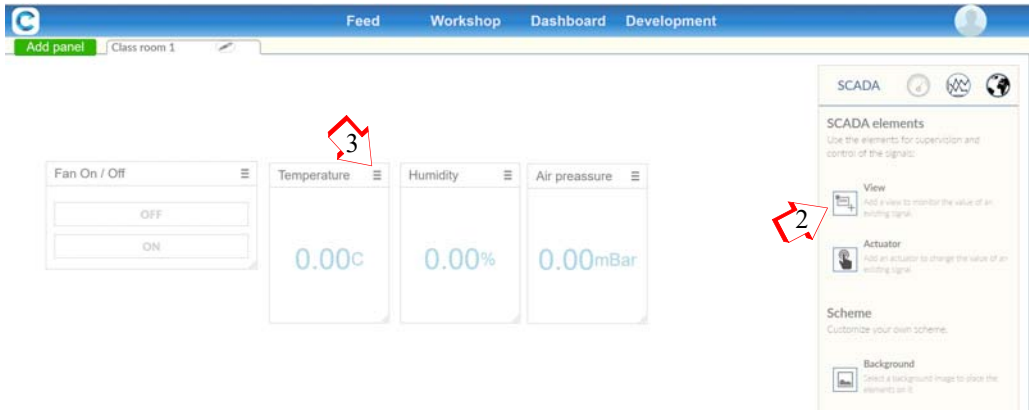


Vi får da et skjermbilde som vist på figuren under:



Vi lager tilsvarende nye signaler for *Lufttrykk* (Område: *800 – 1100*, Enhet *mBar*, Oppløsning *0.1*) og ev. *Luftfuktighet* (Område: *0 – 100*, Enhet *%*, Oppløsning *0.1*) dersom vi bruker BME280.

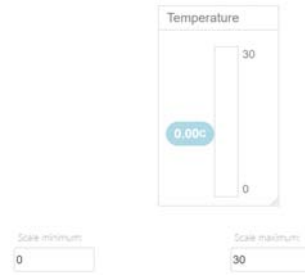
2. **Presentasjon av målinger:** Gå til *Dashboard* og velg hvordan de ulike signalene skal presenteres i panelet. Vi velger å legge også disse parametrene i panel *Class room 1*. Siden dette ikke handler om styring, men om presentasjon så velger vi ikonet *View* (2) til venstre for panelet. Figuren under viser at vi har valgt de tre konsollene *Temperature*, *Air pressure* og ev. *Humidity.*



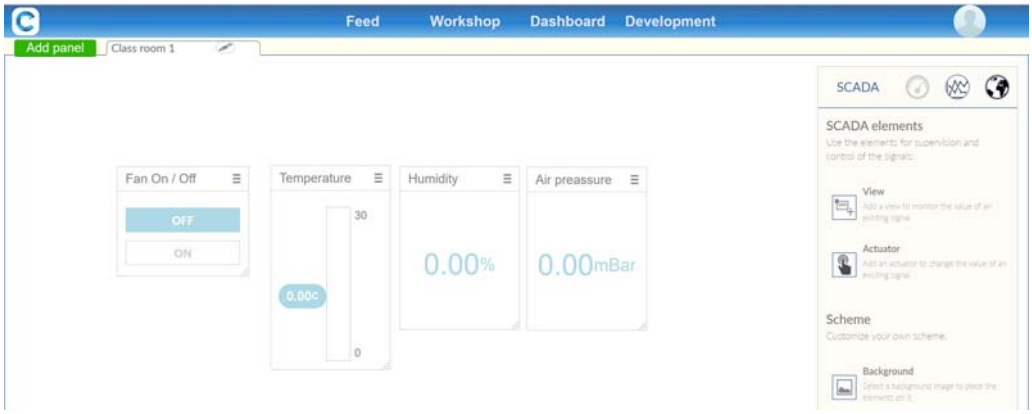
3. **Presentasjonsform:** Vi har foreløpig bare valgt å presentere tallverdiene, men vi kan også velge søylepresentasjon for temperaturen. Vi velger da menyikonet øverst i høyre hjørne (3) og får da opp to valg som vist på figuren til høyre:

   Figuren under viser det endelige panelet, både for styring av releet og for avlesning av sensorverdier.

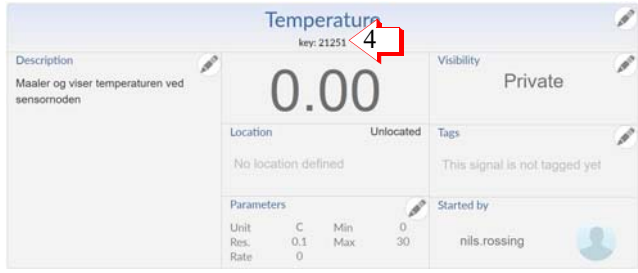
Tall-representasjon



Søyle-representasjon



4. **Lagre nøklene:** For å kunne knytte programmet hos sensornoden (ESP32) opp mot de ulike konsollene, må vi ta vare på nøkkelkodene til hver av dem. Disse leser vi av på hver av signalbeskrivelsene som vist under (4):



I vårt eksempel finner vi at:

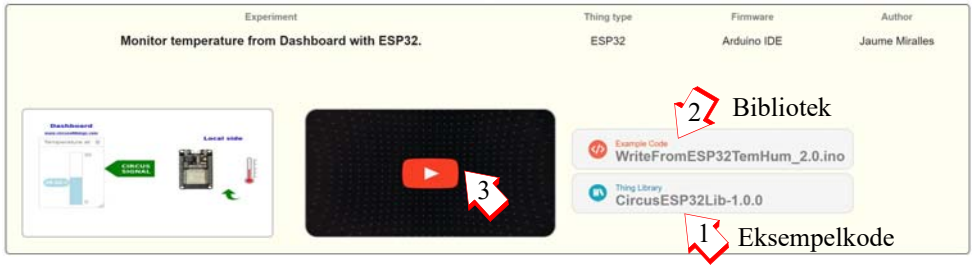
Temperature: *Key: 21251*
Air Pressure: *Key: 13531*
Humidity: *Key: 20852*

Her må hver enkelt sette inn personlige nøkkelkoder (*key*).

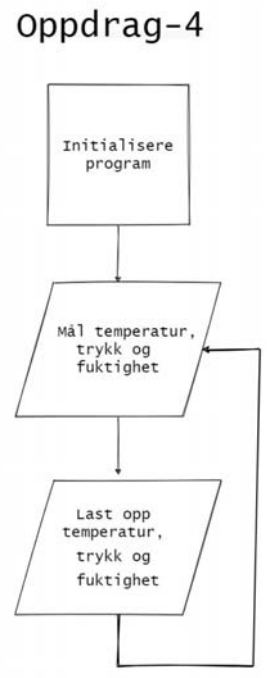
6.5.3 Programmet

Vi skal nå bygge opp programmet og tar utgangspunkt i et eksempel. Både *eksempelprogram* og nødvendige *biblioteker* ligger på hjemmesiden til www.circusofthings.com. Som i oppdrag 2 finner vi begge deler under fanen *Development* lengst til høyre (1).

Vi velger *Experiment* (2 over) og får oversikten over eksemplene hvor vi også finner eksempelkodene (1 under), nødvendig biblioteker (2 under) og instruksjonvideoer (3 under) som sist. Vi velger en video som anvender ESP32 og som måler temperatur, denne passer godt til det vi skal gjøre nå. Figuren under viser fanen til det valgte eksempelet.



Figuren til høyre viser et enkelt flytdiagram for programmet.



1. **Biblioteket** er det samme som sist så det har vi alt installert <CircusESP32Lib.h>. I tillegg trenger vi biblioteket til BME280 <Adafruit_BME280.h> eller BMP280 <Adafruit_-BMP280.h>og <Wire.h> for bruk av I^2C-bussen.
2. **Last ned eksempelkoden (1)** og legg den et sted der den er lett å finne igjen.
3. **Installer biblioteker.** Om nødvendig installer biblioteker. Se avsnitt 6.4.3, side 83 om du er usikker på hvordan du installerer et bibliotek.

Eksempelkoden

Vi skal nå bygge opp programmet for oppdrag 4 trinn for trinn. Dere vil da måtte ta i bruk det dere har lært så langt.

1. **Inkluder nødvendige biblioteker i koden**
   for å kommunisere med:
   Circus of Things <CircusESP32Lib.h>, I^2C-bussen <Wire.h> og sensoren BME280 <Adafruit_BME280.h> eller for BMP280 <Adafruit_BMP280.h>.

2. **Legg inn alle nødvendige parametere:**
   Disse skal du dels hentes fra CoT og din lokale ruter som du ønsker å bruke for å komme deg på nett:

```
char ssid[] = "<Navnet på ruteren din>";
// Place your wifi SSID here

char password[] =  "<Passord for ruteren din>";
// Place your wifi password here

char token[] = "<Token fra din personlige konto ved CoT>";
// Place your token, find it in 'account' at Circus. It will identify you.

char server[] = "www.circusofthings.com";
// Server name at Circus of Things

char temperature_key[] = "<Nøkkel til signalet fra temperaturmåler>"
// Place the Key of the signal you created at CoT for the Temperature

char humidity_key[] = "<Nøkkel til signalet fra luftfuktighets>";
// Place the Key of the signal you created at CoT for the Humidity
// Denne fjernes ved bruk av BMP280.

char pressure_key[] = "<Nøkkel til signalet fra lufttrykksmåler>";
// The key for the signal for air pressure

CircusESP32Lib circusESP32(server,ssid,password);
// The object representing an ESP32 to whom you can order to Write or Read
```

3. **Definer et objekt for sensoren** `bme280` av typen `Adafruit_BME280`:

```
Adafruit_BME280 bme280;
```

4. **Initiering:** I setup()-funksjonen skal du initiere sensoren bme280 og CoT. Legg merke til at vi legger adressen til sensoren bme280 inn i argumentet til initieringsfunksjonen.

```
bme280.begin(0x76);  // Start the sensor BME280
circusESP32.begin(); // Initier kommunikasjon til CoT
```

5. **Les data fra sensoren.** I `void loop()`-funksjonen ønsker vi å deklarere variabler for de tre parametrene som vi samtidig henter fra sensoren BME280, ev. BMP280:

```
float temperature = bme280.readTemperature();  // Les temperaturen
float pressure = bme280.readPressure()/ 100.0F;// Les lufttrykk
float humidity = bme280.readHumidity();        // Les luftfuktighet
```

   Legg merke til at vi deler luftrykket med 100.0F. Årsaken til dette er at vi ønsker at resultatet skal vises i mBar og at vi ønsker å tvinge resultatet til et desimaltall (float), derfor F.

6. **Overfør resultatet til CoT.** Til sist skal målingene sendes over til Circus of Things med nøkkelkoden til det aktuelle konsollet (signal) slik at resultatene kommer ut på rett sted, i riktig form og benevning:

```
circusESP32.write(temperature_key,temperature,token);
// Rapporter temperatur til Circus of Things.

circusESP32.write(humidity_key,humidity,token);
// Rapporter luftfuktighet til Circus of Things.
```

```
    circusESP32.write(pressure_key,pressure,token);
    // Rapporter lufttrykk til Circus of Things.
```

7. **Kjør kode:** Da er det bare å laste opp koden til ESP32 og kjøre koden, og systemet skal fungere.
8. **Test programmet:** Undersøk om programmet oppfører seg som forventet ved å studere verdiene som kommer opp i konsollene på CoT.

6.5.4 Løsningsforslag oppdrag 4

I løsningsforslaget er det lagt inn en utfordring eller feil som må finnes.

NB! Vær oppmerksom på at bme280 må byttes ut med bmp280 dersom denne brukes i stedet.

_____START LØSNINGSFORSLAG-4_____

```
/*
Oppdrag-4
*/

#include <CircusESP32Lib.h>   // Inkluder biblioteket som kommuniserer med CoT.
#include <Adafruit_BME280.h>  // Inkluder biblioteket for BME280/BMP280.

// ---------------------------------------------------------------------
// CircusESP32Lib relaterte deklarasjoner for oppkobling til WiFi og CoT
// ---------------------------------------------------------------------

char ssid[] = "ssid"; // Skriv inn navnet til ruteren din her
char password[] = "password"; // Skriv inn passordet til ruteren din her
char token[] = "token"; // Plasser din brukeridentitet her (token)
char server[] = "www.circusofthings.com"; // Her ligger serveren
char temperature_key[] = "1234";
// Legg inn nøkkelkode om konsollet for visning av temperatur hos CoT.
char humidity_key[] = "1234";
// Legg inn nøkkelkode om konsollet for visning av luftfuktighet hos CoT.
char pressure_key[] = "1234";
// Legg inn nøkkelkode om konsollet for visning av lufttrykk hos CoT.
CircusESP32Lib circusESP32(server,ssid,password);// Her leses nettadressen til
CoT, ruternavn og passord inn.

// Deklarer et objekt bme280 av typen Adafruit_BME280 ev. tilsvarende for BMP280.
Adafruit_BME280 bme280;

void setup() {
  Serial.begin(115200);        // Setter datahastigheten mellom ESP32 og PC.
  bme280.begin(0x76);          // Initialiser sensoren BME280 ev. BMP280.
```

```
  circusESP32.begin();         // Initialiser Circus of Things.
  Serial.println("Oppdrag-4");
// Skriv ut hvilket program som ligger i mikrokontrolleren.
}

void loop() {
  delay(1000);

  // Les av sensoren BME280, temperatur, lufttrykk og luftfuktighet.
  float temperature = bme280.readTemperature();
  float pressure    = bme280.readPressure()/ 100.0;
// Deler på 100.0 og får da hPa og mBar, vanlig benevning for atmosfærisk trykk.
  float humidity    = bme280.readHumidity();

  // Vis verdiene i monitoren
  Serial.println("-------------------------");
  Serial.print("Temperature: "); Serial.println(temperature);
  Serial.print("Humidity: "); Serial.println(humidity);
  Serial.print("Air pressure: "); Serial.println(pressure);
  Serial.println("-------------------------");
  Serial.println(" ");

  // Send verdiene til Circus of Things
  circusESP32.write(temperature_key,temperature,token);
  // Rapportert målt temperatur til CoT.
  circusESP32.write(humidity_key,humidity,token);
  // Rapportert målt luftfuktighet til CoT.
  circusESP32.write(pressure_key,pressure,token);
  // Rapportert målt lufttrykk til CoT.
}
```

STOPP LØSNINGSFORSLAG-4

6.6 Oppdrag 5 – Avlesing av sensoren BME/P280 og styring av vifterele

I de forrige oppdragene 2 og 4 viste vi hvordan vi kan fjernstyre et rele ved bruk av Circus of Things via Internett og hvordan vi kan overføre og visualisere sensordata fra BME/P280. Her skal vi se hvordan vi kan kombinere disse to funksjonene i ett og samme program.

Dvs. at vi vil at den lokale noden både skal kunne motta kommandoer fra CoT-panelet og se avleste sensorverdier på konsollene for hver av sensorene.

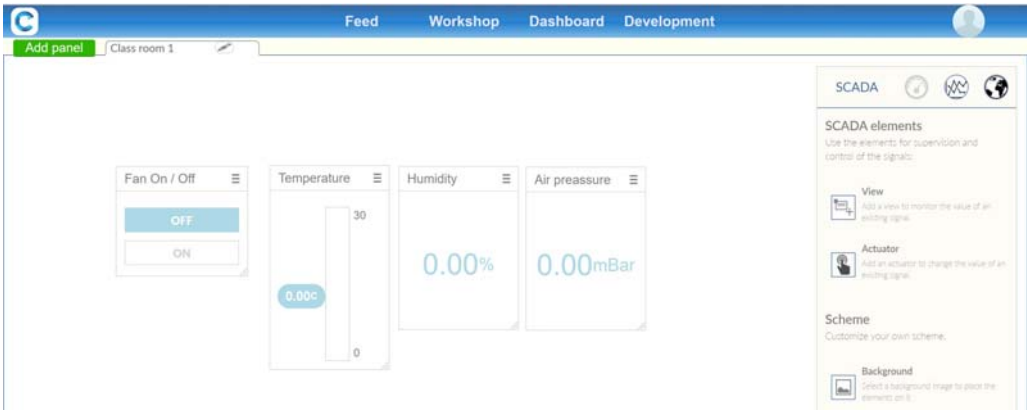
> **Oppdrag 5:** *Styr viftereleet fra panelet og les av sensorene og presenter dem på panelet hos Circus of Things.*

6.6.1 Oppkobling

Om man har gjort de foregående oppgavene, skal kretsen være ferdig oppkoblet med sensoren BME280 eller BMP280 og releet for styring av viften.

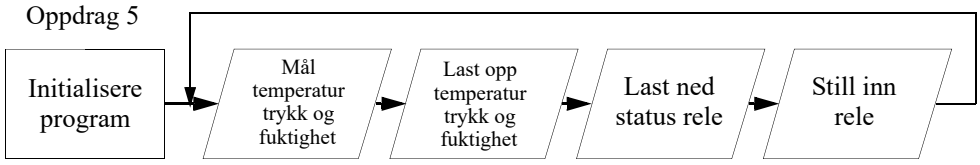
6.6.2 Kombiner programmene i oppdrag 2 og 4

1. Gå gjennom programmene fra oppdragene 2 og 4 og kombiner dem ved å ta vare på det som er felles for begge og suppler oppdrag 4 med tillegget for mottak av kommandoer fra oppdrag 2.
2. En måte å sette sammen panelet hos CoT er vist i figuren under:



3. Kompiler og overfør programmet og sjekk at det oppfører seg som forventet

Figuren under viser flytdiagrammet for programmet:



6.6.3 Løsningsforslag oppdrag 5

Løsningen finnes ved å kombinere løsningsforslagene for oppdrag 2 (avsnitt 6.3.5, side 81) og 4 (avsnitt 6.5.4, side 92).

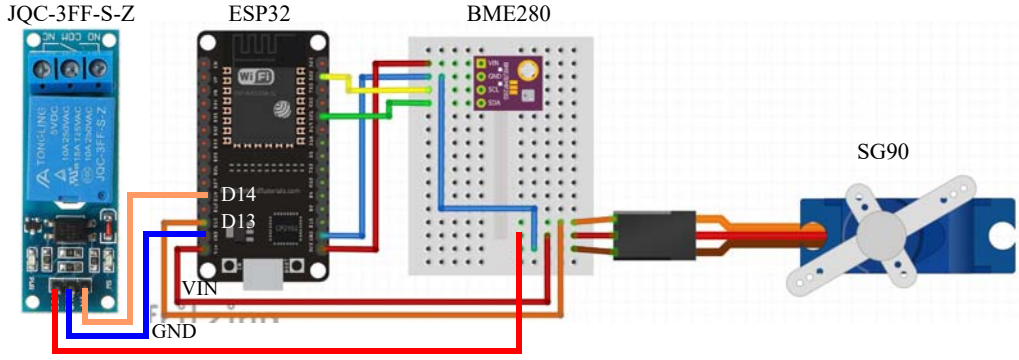
6.7 Oppdrag 6 – Kontinuerlig styring av bommen

I oppdrag 2 og 5 så vi blant annet på hvordan vi kan fjernstyre et rele for å kunne slå av og på en vifte. Den gang brukte vi av og på knapper. I dette oppdraget ønsker vi å styre bommen (servoen i adgangskontrollen) ved hjelp av en *glidebryter*. Dette skal gjøre det mulig å sette den i en hvilken som helst posisjon mellom 0 og 90°.

Oppdrag 6: *Styr bommen fra panelet slik at den kan åpnes i en vilkårlig vinkel mellom 0 og 90°. Integrer styring av bommen i programmet fra oppdrag 5.*

6.7.1 Oppkobling

Vi kobler opp kretsen som vist på figuren under. Ellers bruker vi samme oppkoblingen som vi benyttet i forbindelse med Adgangskontrollen, dvs. servoen (bommen) styres av D13.



Som tidligere velger vi å benytte 5V som spenningskilde for å unngå å belaste den regulerte 3,3V spenningen som forsyner ESP32 og sensoren BME/P280.

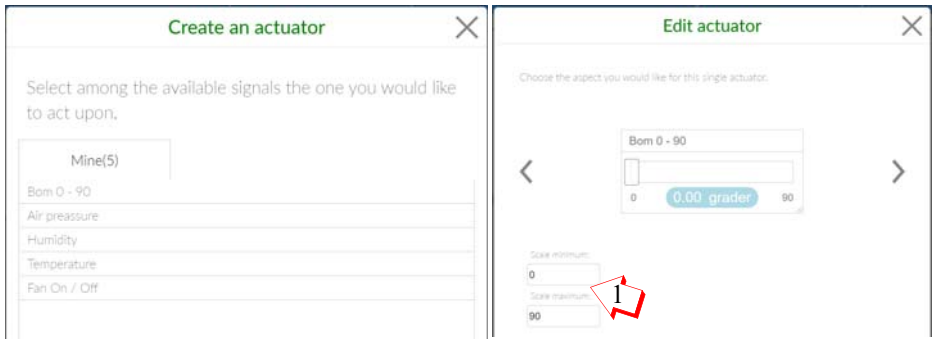
6.7.2 Kontrollpanelet

For å kunne styre bommen kontinuerlig fra 0 – 90° så ønsker vi å benytte en glidebryter.

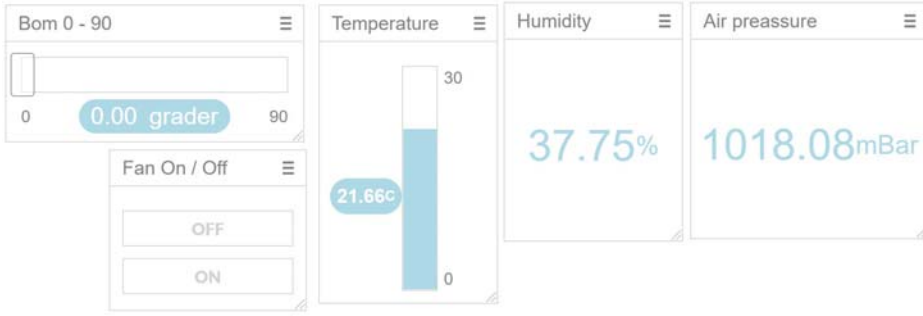
1. Vi henter opp siden www.cir-cusofthings.com og logger inn med e-post og passord. Vi går til *workshop* og velger (+) *New signal*.

   Vi velger å kalle signalet *Bom 0 – 90* og gir det min. og maks. verdier på henholdsvis 0 og 90°. Vi merker oss også

nøkkelkoden til signalet (*key*): I vårt eksempel 19499. Dernest går vi til *Dashboard* og velger ny *Actuator* fra menyen til høyre. Her velger vi glider og skriver inn min. og maks. verdiene (1) for glideren som vist på figuren under.



Vi skulle da ende opp med et panel som ligner på dette:

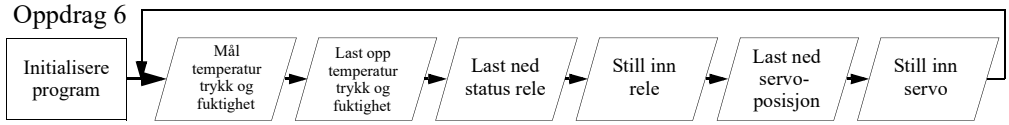


Da er vi klare til å skrive programmet.

6.7.3 Programmet

Vi skal nå overføre en verdi mellom 0 og 90° fra konsollet til vår lokale node. Vi tar utgangspunkt i oppdrag 5 og videreutvikler dette programmet til oppdrag 6.

Figuren under viser flytdiagrammet for oppdrag 6:



1. **Inkluder biblioteket** for å styre servoen

   ```
   #include <Servo.h>
   ```

2. **Inkluder nøkkelkoden** (*key*) i variabel definisjonene. Vi har valgt å kalle variabelen som holder nøkkelkoden: `order_key_gate[]`

Nøkkelkoden finner vi i CoT workshop – i konsollet som definerer signalet for styring av bommen.

3. **Deklarer et objekt av typen Servo**, vi har valgt å kalle objektet *myservo*:

   ```
   Servo myservo;
   ```

   Plasseres før `void setup()`

4. Deklarer en konstant som holde portnummeret for styring av bommen: Pin D13 – plasseres før `void setup()`. Vi velger å kalle den: `GATEPIN`.

   ```
   #define GATEPIN 13
   ```

5. Definer port D13 som en utgang i `void setup()`-funksjonen:

   ```
   pinMode(GATEPIN, OUTPUT);
   ```

6. **Initialiser servoen** i `void setup()`-funksjonen og sett inn portnummeret til porten som styrer servoen:

   ```
   myservo.attach(GATEPIN);
   ```

7. Legg inn kommandoen som leser den overførte verdien fra kontrollpanelet i `void loop()`-funksjonen:

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

8. Bruk variabelen `dashboard_order_gate` til å styre ut vinkelen til bommen:

   ```
   myservo.write(int(dashboard_order_gate));
   ```

   Legg merke til at vi bruker funksjonen int() for å konvertere innholdet i `dashboard_order_gate` til et heltall.

9. **Kompiler og last ned programmet** og undersøk om det fungerer.

6.7.4 Løsningsforslag oppdrag 6

I dette løsningsforslaget har vi valgt å vise kun de nye kommandoene som skal legges inn på riktig plass i oppdrag 5. Hent opp oppdrag 5 og lagre den under navnet oppdrag 6 og legg inn kommandoene på riktig plass.

_____START LØSNINGSFORSLAG-6_____

```
/*
Oppdrag 6:

Legg inn følgende programlinjer på riktig sted i programmet for oppdrag 5
*/
  #include <Servo.h>               // Inkluder servo-biblioteket.
  char order_key_gate[] = "1234";  // Nøkkelkode for å kommunisere med bommen.
  Servo myservo;                   // Definerer objektet myservo av typen Servo.
  #define GATEPIN 13               // Digital utgang for ESP32 for styring av bom.
```

```
  pinMode(GATEPIN, OUTPUT);     // Definer GATEPIN som en utgang.
  myservo.attach(GATEPIN);      // Port nummer som servoen er tilsluttet.

// Motta posisjonen til bommen i grader fra CoT.
int dashboard_order_gate = circusESP32.read(order_key_gate,token);
// Still inn bommen til ønsket vinkel dashboard_order_gate.
myservo.write(int(dashboard_order_gate));
```

_____STOPP LØSNINGSFORSLAG-6_____

6.8 Oppdrag 7 – Bygg opp programmet med funksjoner

Programmet fra oppdrag 6 utfører tre oppgaver:

1. Leser av temperatur, luftfuktighet og lufttrykk fra sensorkortet BME280 eller BMP280 og sender informasjonen til CoT
2. Mottar kommandoer for styring av viftereleet fra CoT og setter releet i ønsket stilling
3. Mottar kommandoer for styring av bommen fra CoT og setter bommen i ønsket posisjon

Etter som programmer blir mer sammensatte og komplekse, blir det viktigere og viktigere å organisere programmene på en god og oversiktlig måte. En måte å gjøre dette på er å legge oppgaver som programmet skal utføre i egne funksjoner som vi f.eks. kaller opp i void loop()-funksjonen vår. Dersom vi gir funksjonene beskrivende navn blir det også lett å lese programmene vi lager, for oss selv og for andre.

> **Oppdrag 7:** *Ta utgangspunkt i oppdrag 6 og lag fem funksjoner som utfører følgende oppgaver. Vi har tillatt oss å foreslå navn på funksjonene, men her står dere fritt:*
> - *readTempBME280AndSendToCoT() – Leser temperatur fra BME280 og sender til CoT*
> - *readHumBME280AndSendToCoT() – Leser luftfuktighet fra BME280 og sender til CoT*
> - *readPresBME280AndSendToCoT() – Leser lufttrykk fra BME280 og sender til CoT*
> - *readCoTAndSetRelay() – Mottar styringsinfo. for releet fra CoT og setter det i ønsket stilling*
> - *readCoTAndSetServo() – Mottar styringsinfo. for servo fra CoT og setter den i ønsket stilling*

Legg merke til at funksjonsnavnene avspeiler hva funksjonen gjør. Av årsaker vi kommer tilbake til så foreslår vi å skille de tre avlesningene fra BME280 i tre funksjoner, alternativt i to funksjoner for BMP280.

6.8.1 Oppkobling

Vi bruker samme oppkoblingen som tidligere så vi trenger ikke å gjøre noe med den.

6.8.2 Kontrollpanelet

Vi trenger heller ikke gjøre noe med kontrollpanelet hos CoT.

6.8.3 Programmet

De tre funksjonene som leser BME280 ønsker vi skal returnere sensorverdiene som leses. Verdiene for henholdsvis temperatur, luftfuktighet og lufttrykk er desimaltall, funksjonene må derfor deklareres som type `float`, og vil ha formen:

```
float <Funksjonsnavn>(){
// Funksjonskropp
return <parameter>;
}
```

Ved å avslutte funksjonskroppen med `return` og navnet på parameteren vi ønsker å returnere, så vil funksjonen kunne levere denne parameteren til en variabel når programmet forlater funksjonen. Funksjonskallet vil derfor kunne se slik ut:

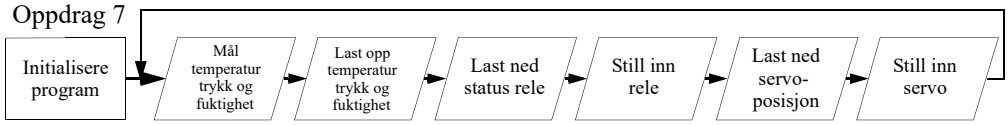
```
<variabelnavn> = <funksjonsnavn>();
```

Hvor returverdien fra funksjonen settes inn i variabelen.

De to siste funksjonene trenger ikke å overføre noe argument og ikke returnere noen verdi. De kan derfor ha formen:

```
void <Funksjonsnavn>(){
// Funksjonskropp
}
```

Figuren under viser flytdiagrammet for oppdrag 7 som er likt med oppdrag 6:



1. **Bygg opp funksjonene** og se om programmet oppfører seg som forventet.

Dersom du er usikker på hvordan du lager funksjoner se avsnitt 4.5.14, side 55.

6.8.4 Løsningsforslag oppdrag 7

Løsningsforslaget mangler funksjonskallene. Så før koden kan kjøre riktig, så må funksjonskallene til funksjonene legges inn på riktig måte i void loop():

_____START LØSNINGSFORSLAG-7_____

```
/*
Oppdrag-7
*/

#include <CircusESP32Lib.h>   // Inkluder biblioteket som kommuniserer med CoT
#include <Servo.h>            // Inkluder servo-biblioteket
#include <Adafruit_BME280.h>  // Inkluder biblioteket for BME280
```

```
char ssid[] = "ssid";                          // Skriv inn navnet til ruteren din her
char password[] = "password"; // Skriv inn passordet til ruteren din her
char token[] = "token"; // Plasser din brukeridentitet her (token)
char server[] = "www.circusofthings.com"; // Her ligger serveren
char order_key_gate[] = "1234"; // Nøkkel for å kommunisere med bommen.
char order_key_relay[] = "1234";
// Nøkkel-informasjon om konsollet for styring av relet hos CoT
char temperature_key[] = "1234";
// Nøkkel-informasjon om konsollet for visning av temperatur hos CoT.
char humidity_key[] = "1234";
// Nøkkel-informasjon om konsollet for visning av luftfuktighet hos CoT.
char pressure_key[] = "1234";
// Nøkkel-informasjon om konsollet for visning av lufttrykk hos CoT.

CircusESP32Lib circusESP32(server,ssid,password);
// Definer objektet circusESP32 av typen CircusESP32Lib.
Adafruit_BME280 bme280;
// Definer objektet bme280 av typen Adafruit_BME280.
Servo myservo;
// Definerer objektet myservo av typen Servo.

// Deklarasjon av konstanter som definerer oppkobling til ESP32.
#define RELAYPIN 14       // Digitalt utgang for ESP32 for styring av vifterele.
#define GATEPIN 13        // Digitalt utgang for ESP32 for styring av bom.

void setup() {
  Serial.begin(115200);       // Sett opp kommunikasjonshastigheten til monitoren.
   pinMode(RELAYPIN, OUTPUT);   // Definer RELAYPIN som en utgang.
   pinMode(GATEPIN, OUTPUT);    // Definer GATEPIN som en utgang.
   circusESP32.begin();         // Initialiser Circus of Things.
   bme280.begin(0x76);          // Initialisering av sensor BNE280.
  myservo.attach(GATEPIN); // Gi informasjon om hvilken port servoen er tilsluttet.
   Serial.println("Oppdrag-7"); // Skriv ut hvilket program som ligger i
mikrokontrolleren.
}

void loop() {
  // Les av temperatur og send til CoT.
  // Legg inn riktig funksjonskall her

  // Les av luftfuktighet og send til CoT, bare ved bruk av BME280.
  // Legg inn riktig funksjonskall her
```

```
  // Les av lufttrykk og send til CoT.
  // Legg inn riktig funksjonskall her

  // Motta styreinforasjon for releet og sett det i ønsket stilling.
  // Legg inn riktig funksjonskall her

  // Motta styreinforasjon for bommen og sett den i ønsket stilling.
  // Legg inn riktig funksjonskall her

  delay(500);
}

//----------------Funksjon som leser og sender temperatur ----------------
float readTempBME280AndSendToCoT(){

  // Leser av temperatur fra sensorene BME280
  float temperature = bme280.readTemperature();

  // Rapporterer temperatur fra sensorene til CoT.
  circusESP32.write(temperature_key,temperature,token);

  return temperature;
}

//----------------Funksjon som leser og sender luftfuktighet ----------------
float readHumBME280AndSendToCoT(){

  // Leser av luftfuktighet fra sensorene BME280.
  float humidity    = bme280.readHumidity();

  // Rapporterer luftfuktighet fra sensorene til CoT.
  circusESP32.write(humidity_key,humidity,token);      // Rapporter luftfuktighet
til Circus of Things.

  return humidity;
}

//----------------Funksjon som leser og sender lufttrykk ----------------
float readPresBME280AndSendToCoT(){

  // Leser av lufttrykk fra sensorene BME280.
  float pressure    = bme280.readPressure()/ 100.0F;
```

```
  // Rapporterer lufttrykk fra sensorene til CoT.
  circusESP32.write(pressure_key,pressure,token);      // Rapporter lufttrykk til
Circus of Things.

  return pressure;
}

//---------------Funksjon som mottar styresignaler til releet----------------
void readCoTAndSetRelay(){

  // Motta styreinformasjon til reléet fra Circus of thing.
  int dashboard_order_relay = circusESP32.read(order_key_relay,token);

  // Reléet bruker verdien som hentest fra CoT, og styrer direkte Reléet med den.
  digitalWrite(RELAYPIN, dashboard_order_relay);
}

//------------Funksjon som mottar informasjon og styrer bommen ----------------
void readCoTAndSetServo(){

    // Leser posisjonen til bommen i grader fra CoT.
    double dashboard_order_gate = circusESP32.read(order_key_gate,token);

    // Still inn bommen til ønsket vinkel dashboard_order_gate.
    myservo.write(int(dashboard_order_gate));
}
```

_____STOPP LØSNINGSFORSLAG-7_____

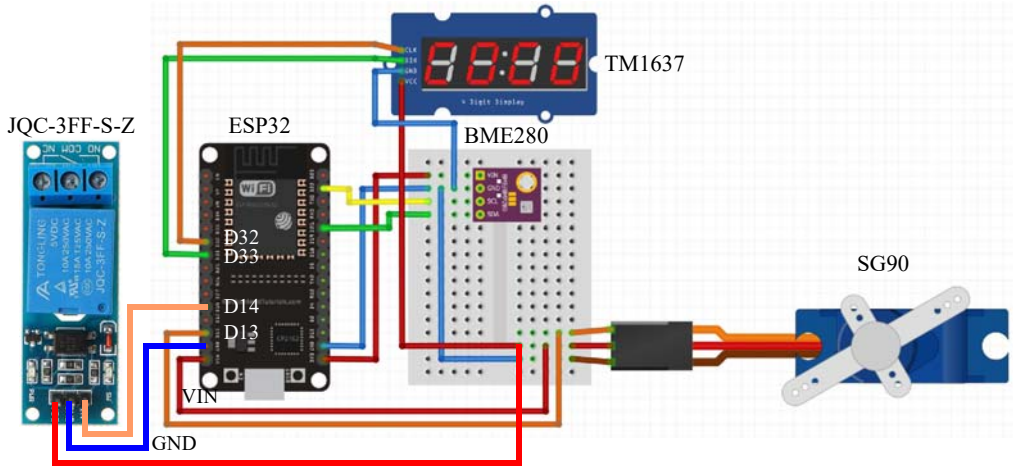
6.9 Oppdrag 8 – Skriv temperaturen til 7-segment displayet

I dette oppdraget skal vi ta den avleste temperaturen fra BME280 (ev. BMP280) og skrive ut til 7-segmentdisplayet. Vi ønsker fortsatt å bruke funksjoner og velger derfor at skriving til displayet i sin helhet legges i en funksjonen.

> **Oppdrag 8:** *Ta utgangspunkt i oppdrag 7 og legg til en funksjon som skriver temperaturen til 7-segment displayet.*

6.9.1 Oppkobling

Til dette oppdraget må vi montere og koble opp 7-segment displayet som vist på figuren under. Til dette er det laget en egen flatkabel med fire ledninger. Fargene vil ikke stemme med tegningen.



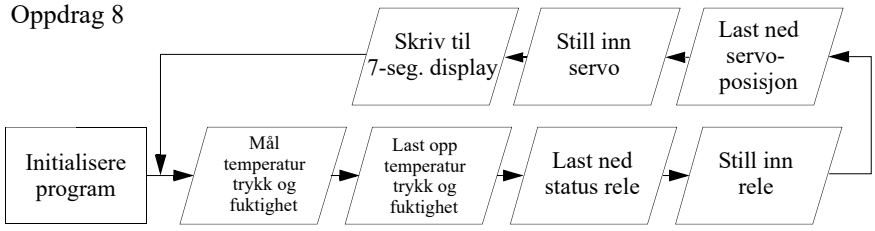
Oppkoblingen av displayet er den samme som for Adgangskontrollen.

6.9.2 Kontrollpanelet

Vi trenger ikke gjøre noe med kontrollpanelet hos CoT for dette oppdraget.

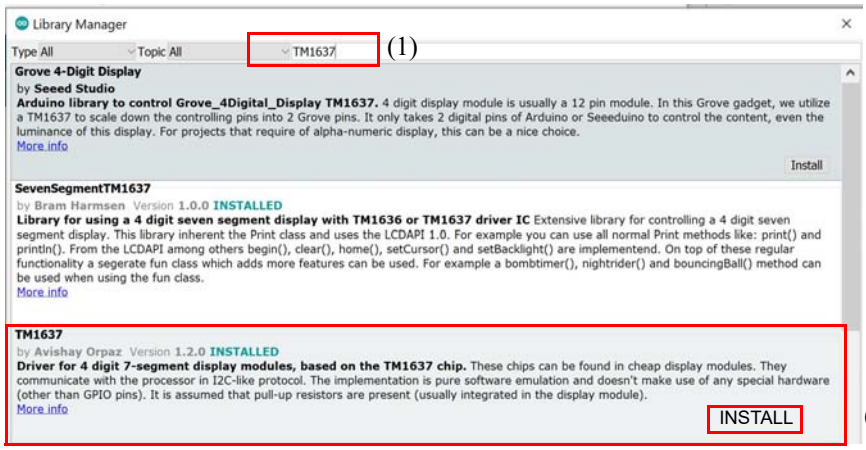
6.9.3 Programmet

Figuren under viser flytdiagrammet for oppdrag 8:



For å bruke displayet TM1637 så trenger vi å installere biblioteket som hører til dette displayet. Om det ikke alt er gjort, så gjøres det på vanlig måte.

1. **Installer biblioteket for displayet:** Velg *Sketch* menylinjen. Velg så *Include library –> Library manage*. Skriv inn: TM1637 i innboksen (1) og vent. Velg det som kalles TM1637 som vist på figuren under (2) og trykk INSTALL (3).



2. Vi **inkluderer biblioteket** i programmet ved å skrive:

```
#include <TM1637Display.h>     // Inkluder bibliotek for TM1637
```

   Dette plassers øverst i programmet sammen med de andre inkluderte bibliotekene

3. Vi **definerer** de to inngangene CLK og DIO som to konstanter PIN_CLK og PIN_DIO og gir dem pinnene 32 og 33 på ESP32:

```
#define PIN_CLK 32
#define PIN_DIO 33
```

4. Dernest **deklarerer vi et objekt** "`display`" av typen "`TM1637Display`" samtidig som vi opplyser om hvilke porter som skal brukes til å overføre CLK og DIO signalene.

```
TM1637Display display(PIN_CLK, PIN_DIO);
```

5. I setup()-funksjonen setter vi **lysstyrken til displayet** med følgende kommando:

```
display.setBrightness(7);
```

6. Vi ønsker å **lage en funksjon** for å skrive ut den målte temperaturen til displayet, og vi vil overfør *den målte temperaturen gjennom argumentet* til funksjonen. Siden temperaturen er et desimaltall velger vi å definere argumentet som en `float`.

```
void <Funksjonsnavn>(float <argumentnavn>){
// Funksjonskropp
}
```

   Sett et beskrivende navn på funksjonen og på argumentet.

   Inne i funksjonskroppen velger vi å bruke følgende funksjoner for å skrive til displayet vårt:

```
display.clear();
display.showNumberDecEx(int(<argumentnavn>), 0b01000000, false, 4, 0);
```

Først tømmer vi displayet før vi skriver inn nye verdier (`display.clear();`).

Kommandoen `display.showNumberDecEx();` skriver ut tallet til displayet.

La oss se hva hvert av argumentene betyr:

- `int(<argumentnavn>)` – Angir tallet vi ønsker å vise på displayet og som vi gjør om til et heltall
- `0b01000000` – Setter komma etter 2. siffer
- `false` – Angir at vi ikke ønsker å sette inn 0 foran om tallet er mindre enn 10
- `4` – Angir at displayet har 4 siffer
- `0` – Angir at det mest signifikante sifferet skal være plassert til venstre på displayet

Det er en utfordring til, nemlig at vi overfører en temperatur som normalt vil ligge mellom 0 – 30 grader C. Siden displayet vårt har fire siffer og kun viser heltall, så velger vi å multiplisere tallet med 100 og så ta heltallet av dette (`int(<argumentnavn>)`). Siden vi har valgt å legge inn komma etter 2. siffer så vil kommaet alltid komme på rett plass. Følgende eksempel illustrerer dette. 23,50 er temperaturen i Celcius vi ønsker å vise:

23.50 * 100 = 2350.00 => int(2350.00) = 2350 => Skrevet ut til displayet blir dette 23.50 Komma er fast plassert etter 2. siffer.

7. **Kall av funksjonen**

   Vi ønsker altså å skrive ut temperaturen etter at den er avlest fra sensoren BME/P280. La oss anta at temperaturen er lagret i variabelen `temperatur`. Funksjonskallet blir da slik:

   ```
   <Funksjonsnavn>(temperatur);
   ```

   Variabelen `temperatur` kan vi få returnert fra funksjonen som leser ut temperaturen fra BME/P280.

8. **Skriv programmet** og undersøk om det fungerer som ønsket.

Dersom du ønsker å vite mer om displayet TM1637 og hvordan du kan programmere dette displayet se avsnitt 7.4, side 138.

6.9.4 Løsningsforslag oppdrag 8

Løsningsforslaget viser oppbyggingen av funksjonen som skriver temperaturen til displayet. Ta utgangspunkt i løsningsforslaget til oppdrag 7. Skriv funksjonen inn i programmet. Legg inn funksjonskallet med tilhørende variabel som holder temperaturen inne i void loop()-funksjonen:

_____START LØSNINGSFORSLAG-8_____

```
// Funksjon som skriver til displayet

void showAtDisplay (float showNumber){

  // Vis showNumber på display. showNumber er et desimaltall.
```

```
  // Vårt display kan bare vise heltall, men vi kan sette komma fast.
  // Vi har valgt å sette komma fast etter andre siffer (f.eks. 23.00)
  // Ved å multiplisere showNumber med 100,
  // konvertere tallet til et heltall får vi vist den med to desimaler

  showNumber = showNumber*100;
  display.clear();                      // Tøm displayet for data
 display.showNumberDecEx(int(showNumber), 0b01000000, false, 4, 0);   // Vis tall
  delay(500);
}
```

_____START LØSNINGSFORSLAG-8_____

6.10 Oppdrag 9 – Vis hvordan temperaturen har endret seg over tid

CoT gir muligheter for å vise hvordan sensordataene har endret seg over tid.

> **Oppdrag 9:** *Lag et nytt panel som viser hvordan temperaturen målt av BME/P280 har endret seg over tid. Legg til luftfuktighet eller lufttrykk i samme diagram.*

For å få til dette må vi ha noen data å vise fram. Det oppnår vi ved å sende data til CoT over noe tid. Du kan la programmet du laget i oppdrag 8 stå å gå i noen timer, gjerne et sted temperaturen endrer seg. CoT vil på denne måten logge data som den mottar.

6.10.1 Oppkobling

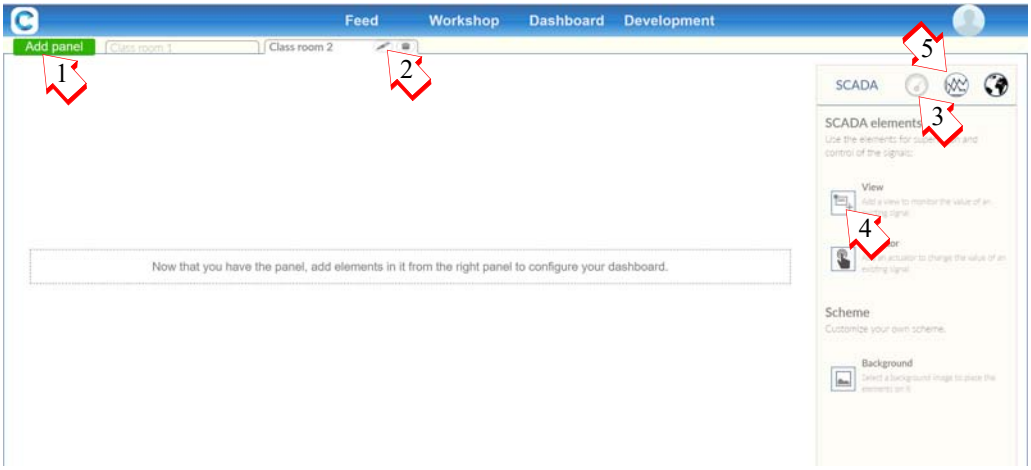
Til dette oppdraget trenger du ikke gjøre noe med oppkoblingen.

6.10.2 Kontrollpanelet

Alt vi skal gjøre i dette oppdraget gjør vi i Circus of Things.

1. **Logg på:** Logg på Circus of Things på vanlig måte.

2. **Opprett nytt panel:** Gå til *Dashboard* og velg *Add panel (1)* øverst til venstre. Velg blyanten til høyre for *New Panel (2)* og gi det et passende navn. Pass på at du befinner deg i *SCADA mode* (3). Velg så *View* (4) og hent opp konsollet ditt som viser temperatur.



3. **Vis tidslinje:** Nå er det på tide og velge å vise *Time line* (5, over) og så bestemme seg for tidsrommet du ønsker å vise i diagrammet.

   Tidsperioden velges til høyre for panelet (6, under) som viser tidslinjen for, i dette tilfellet, temperaturen. Vi velger også å krysse av for *Y axis – Fit to values* (7, under), alternativt kan man sette et spesifikt temperaturområde for y-aksen.

Figuren over viser to måleserier tatt i en hytte og spenner tilsammen over to døgn. Til venstre fyres det opp i ovnen, mens til høyre stiger temperaturen jevnt ut over formiddagen. I perioden mellom er sensornoden avslått. “Spikeren” til høyre skyldes oppvarming med fingeren på sensoren.

4. **Foreta egne målinger:** Gjør egne målinger over noe tid og forsøk å forstå hva endringene skyldes.
5. **Bruk flere sensorer:** Gå tilbake til *SCADA mode* og se hva som skjer dersom du også henter fram konsollet for luftfuktighet (bare ved bruk av BME280).

6. **Les av verdier langs kurven:** Ved å flytte musa langs kurvene, kan du lese av verdiene ved hvert tidspunkt.

6.10.3 Program

Det trengs ikke gjøres noe med programmet i mikrokontrolleren (sensornoden) i dette oppdraget.

6.11 Oppdrag 10 – Legg inn terskelverdi i panelet og styr vifta med temperaturen

I dette oppdraget skal vi legge inn en glidebryter i panelet slik at vi kan sette en terskelverdi der vi vil at vifta skal slå seg på. Samtidig ønsker vi å kunne overstyre terskelverdien med bryteren vi la inn i oppdrag 2.

> **Oppdrag 10:** *Lag et nytt konsoll for å sette terskelverdien med en glidebryter, og endre det eksisterende slik at vifta kan settes i tre stillinger:*
> *- On – Går hele tiden*
> *- Auto – Går når temperaturen er over en terskelverdi*
> *- Off – Går ikke*
> *Terskelverdien skal kunne settes kontinuerlig fra 10 – 40°C.*
> *Endre programmet slik at det oppfyller betingelsene.*

6.11.1 Oppkobling

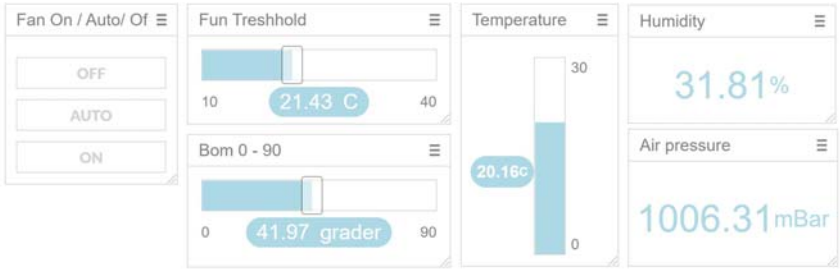
Det trengs ikke gjøres noen endringer i oppkoblingen for dette oppdraget.

6.11.2 Kontrollpanelet

Det må gjøres noen endringer i panelet:

1. **Endre styrekonsoll for vifte:** Legg inn en ekstra knapp med "Auto"-funksjon i konsollet som styrer vifta i tillegg til "On" og "Off". Er du i tvil om hvordan dette skal gjøres se oppdrag 2. Dersom "On" har verdien 1 og "Off" verdien 0, så kan du gi "Auto" verdien 2.
2. **Terskelverdi:** Legg inn et konsoll for å sette terskelverdien mellom 10 og 40°C. Her er det naturlig å bruke en glidebryter. Er du i tvil om hvordan dette skal gjøres, se oppdrag 6.

Figuren under viser et eksempel på hvordan panelet kan ta seg ut:



6.11.3 Programmet

Her står du ganske fritt, men her er noen trips:

1. Legg inn nøkkelkode for det nye displayet

   ```
   char order_key_relay[] = "1234"; // Legg inn din nøkkelkode
   ```

   Denne legges inn før setup()-funksjonen sammen med de andre

2. Endre koden i funksjonen som styrer releet

   For å kunne styre releet i AUTO-funksjon trenger vi terskelverdien som vi må hente fra CoT og temperaturen som vi leser av fra sensoren vår. Vi kan velge å overføre den målte temperaturen via argumentet til funksjonen som styrer releet eller lese av temperaturen på nytt inne i rele-funksjonen. Det samme gjelder i og for seg terskelverdien. Vi har valgt å overføre temperaturen gjennom argumentet og lese av terskelverdien inne i funksjonen.

   Vi har tidligere laget en funksjon som leser av og returnerer temperaturen:

   ```
   float temperatur = readTempBME280AndSendToCoT();
   ```

3. Hent in informasjon fra CoT om hvilken tilstand Fun On/Auto/Off bryteren står i.
   Dette gjør vi ved å endre på funksjonen som styrer releet. Denne har vi tidligere kalt:

   ```
   void readCoTAndSetRelay(float temperatur)
   {
     // Kroppen til funksjonen
   }
   ```

   Vi ser at temperaturen overføres i argumentet til funksjonen.

4. I tillegg til temperaturen som vi har lest av sensoren og som overføres i argumentet, så trenger vi å vite om releet skal slås *På* ("On" = 1), slås *Av* ("Off" = 0) eller settes i *Auto* ("Auto" = 2). Dersom den er satt i Auto må vi vite hvilken terskelverdi for temperaturen vi vil at releet skal slås inn. *Terskelverdien* er satt i konsollet hos CoT og sendes til sensornoden (ESP32) og må derfor leses ned. Denne informasjonen henter vi med følgende kommandoer:

   ```
   double dashboard_order_relay     = circusESP32.read(order_key_relay,token);
   double dashboard_order_treshhold = circusESP32.read(order_key_treshhold,token);
   ```

   Resultatene velger vi å legge i to variabler `dashboard_order_relay` og `dashboard_order_treshhold` som vi nå skal bruke.

5. Lag en if-setning som tar hensyn til bryterens stilling og utfører oppdraget, slår *Av* eller *På* releet og vifta, ev. setter den i *Auto*-funksjon.

   ```
   if(dashboard_order_relay == 1)       // ON
   {
     // Plasser kommando som slår På releet her
   }
   if(dashboard_order_relay == 0)       // OFF
   {
     // Plasser kommando som slår Av releet her
   }
   if(dashboard_order_relay == 2)       // AUTO
   {
     // Plasser kode som sjekker temperatur opp mot terskelverdi
   }
   ```

6. Lag en ny if-setning for Auto som sjekker om temperaturen er over eller under terskelverdien.

   ```
   if(temperatur > dashboard_order_treshhold) // Sjekk om temp. er over terskel
   {
     // Kode som slår På releet og vifta
   }
   else
   {
     // Kode som slår Av releet og vifta
   }
   ```

7. Skriv programmet og undersøk om det fungerer som forventet.

6.11.4 Løsningsforslag oppdrag 10

Løsningsforslaget viser oppbyggingen *av funksjonen som slår av eller på vifta*. Funksjonen må kalles fra void loop()-funksjonen samtidig som den målte temperaturen fra temperatursensoren må overføres i argumentet til funksjonen:

_____START LØSNINGSFORSLAG-10_____

```
void readCoTAndSetRelay(float temperatur){
```

```
    // Leser kommando til releet
    double dashboard_order_relay     = circusESP32.read(order_key_relay,token);
    double dashboard_order_treshhold =
circusESP32.read(order_key_treshhold,token);

    // Slå på vifta når dashboard_order_realy er lik 1
    if(dashboard_order_relay == 1)      // ON
    {
      digitalWrite(RELAYPIN, HIGH); // Koble til relekontakt
    }
    else if(dashboard_order_relay == 2) // AUTO
    {
      if(temperatur > dashboard_order_treshhold)
      {
        digitalWrite(RELAYPIN, HIGH);
        // Koble til relekontakt når tempeperaturen er høyere en terskelverdien
      }
      else
      {
        digitalWrite(RELAYPIN, LOW);
        // Koble fra relekontakt når tempeperaturen er høyere en terskelverdien
      }
    }
    else if (dashboard_order_relay == 0) // OFF
    {
      digitalWrite(RELAYPIN, LOW);  // Koble fra relekontakt
    }
}
```

_____STOPP LØSNINGSFORSLAG-10_____

6.12 Oppdrag 11 – Sett sensornoden i "Sleep mode" mellom hver måling

Siden vi jobber med Internet of Things så er effektforbruket ved hver sensornode kritisk siden nodene gjerne er batteridrevet. ESP32 er utstyrt med funksjoner som gjør det mulig å koble ned mikrokontrolleren i større eller mindre grad mellom hver måling. I dette oppdraget ønsker vi å gjøre en måling hvert 10 sekund for så å legge kontrolleren i dvale mellom hver måling.

Dessuten vil vi gjerne registrere tidspunktet for hver måling og sende dette tilbake til CoT.

> **Oppdrag 11** – *Ta utgangspunkt i oppdrag 10 og lag et tillegg i programmet slik at det gjøres målinger ca. hvert 10 sek. Mellom hver måling skal kretsen legges i dyp dvale ("deep sleep"). Lag et konsoll i panelet ved CoT som viser tiden i sekunder fra måleserien startet.*

Det anbefales å lese avsnitt 3.3, side 26 om energisparing og dvaletilstander før du går videre med dette oppdraget.

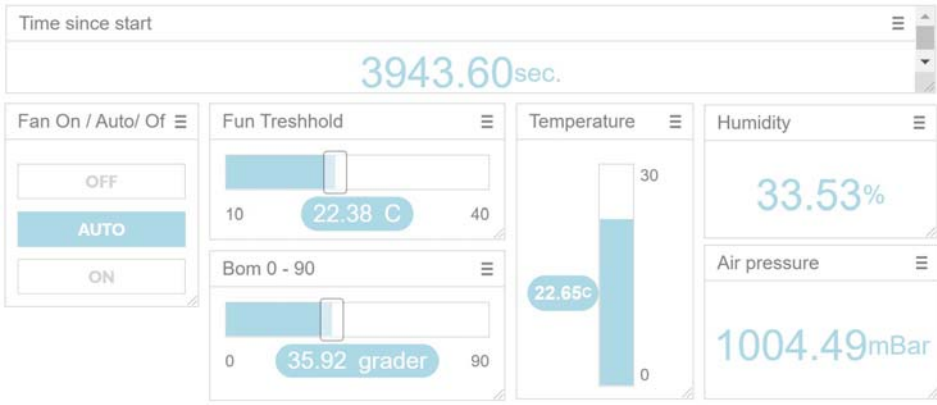
6.12.1 Oppkobling

Det trengs ikke gjøres noen endringer i oppkoblingen for dette oppdraget.

6.12.2 Kontrollpanelet

Det må gjøres noen endringer i panelet:

1. **Vis tiden fra start:** Lag et konsoll som viser tiden i sekunder siden vi startet måleserien. Bruk *workshop* hos CoT til å definere konsollet og legg det inn i panelet sammen med alle de andre målingene. Det kan f.eks. se slik ut:



   Husk å ta vare på nøkkelkoden til displayet slik at du kan legge den inn i programmet.

6.12.3 Programmet

Følgende må gjøres i programmet, under skal vi vise trinn for trinn hvordan vi gjør det:

- Sett opp betingelsene for dvaletilstanden. Vi ønsker at vekking skal gjøres av en *timer*. (Se avsnitt 3.3.3, side 30)
- Legg kontrolleren i dvale når den har utført gjøremålene sine

I tillegg må vi sørge for å legge tidsinformasjonen på en trygg plass i lageret mens kontrolleren "sover" og oppdatere denne for hver oppvåkning for så å sende den til CoT og vise den på konsollet: "Time since start".

- Deklarere en `unsigned long` variabel og legg den i RTC-lageret[29] som ikke blir slettet under dvaletilstanden.

29. RTC lageret: Lageret knyttet "Real Time Clock" mister ikke informasjonen under "Deep sleep".

- Beregn og akkumuler tiden som er gått siden oppstart rett før hver dvaletilstand (se avsnitt 3.3.4, side 32)
- Send akkumulert tid til RTC-lageret
- Gå i dvale

La oss ta det trinn for trinn. Bakgrunnsinformasjon til denne utvidelsen av programmet finnes i avsnitt 3.3.3, side 30 og 3.3.4, side 32.

1. Sett opp betingelsene for dvaletilstanden

Vi ønsker å sette opp en dvaletiµstand som styres av RTC-klokka ("Timer") og vekker kretsen hvert 10. sekund. Til dette bruker vi kommandoen:

```
esp_sleep_enable_timer_wakeup(MAALEINTERVALL * MEGA);
```

Hvor vi spesifiserer dvaleintervallet i mikrosekunder (µs) som legges i `void setup()`-funksjonen. Vi har valgt å definere `MAALEINTERVALL` i sekunder og multiplisere det med `MEGA = 1000000`, slik at vi får verdien i mikrosekunder (µs).

2. Start dvalen

Vi starter dvalen med følgende kommando:

```
esp_deep_sleep_start();
```

Det er naturlig å legge denne kommandoen på det stedet i programmet hvor alle oppgavene er utført, gjerne i `void loop()`-funksjonen.

Det neste vi ønsker er å overføre den akkumulerte tiden til CoT, fra vi startet og til hver måling ble utført.

3. Definer en variabel i RTC-lageret

For å kunne ta vare på tiden mellom hver gang vi legger kretsen i dvale, må vi sørge for at den legges på et "trykt" sted, nemlig i RTC-lageret. Dette gjør vi med denne variabelen og null-stille den ved oppstart.

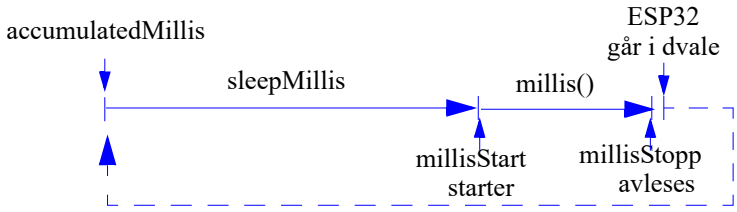
```
RTC_DATA_ATTR unsigned long millisOffset = 0;
```

`RTC_DATA_ATTR` sørger for å legge variabelen i RTC-lageret. Vi har kalt vår variabel `millisOffset`

4. Beregn og akkumuler tiden

Siden vi ikke kan måle tiden mens ESP32 er i dvale, er det beste estimatet vi har at tiden i dvale er nær den vi oppgir (kalt sleepMillis i figuren under), i vårt tilfelle 10 sek. Dernest kommer den tiden fra vi vekker ESP32 til det tidspunktet der vi igjen legger den i dvale. Siden millis() startes på nytt ved hver vekking, så burde det være tilstrekkelig å lese av millis() idet vi stenger ned kretsen (millisStopp). Om vi velger å kalle den akkumulerte tiden for accumulatedMillis så kan vi sette opp følgende sammenheng:

accumulatedMillis = accumulatedMillis + sleepMillis + millisStopp



accumulatedMillis = accumulatedMillis + sleepMillis + millisStopp

Vi har imidlertid også en mistanke om at vi mister litt tid, spesielt rett før kretsen legges i dvale. Der vil det gå litt tid fra vi måler tiden millis() og til den faktisk går i dvale.

Hvordan kan vi ev. måle et slikt avvik?

5. **Legg inn formelen for akkumulert tid**
   Vi har valgt å legge hele peregningen i en egen funksjon som kalles fra void loop() rette før den går i dvale:

```
float showTimeToCoT(){
  // Leser av og beregner tid siden start:
     millisOffset = millis() + millisOffset + MAALEINTERVALL*1000;
  // Regn om til sekunder før overføring:
     float timeSec = float(millisOffset / 1000.0);
  // Rapporterer tid siden start i sekunder til CoT:
     circusESP32.write(time_key, timeSec, token);
}
```

6. **Overfør akkumulert tid til CoT**
   Bruk følgende kommando for å overføre akkumulert tid til CoT og til riktig konsoll

   `circusESP32.write(time_key,`<navn på akkumulert verdi>`,token);`

   Hvor `time_key` er en tekststreng som inneholder nøkkelkoden til konsollet som viser akkumulert tid.

   `time_key[] = "`<nøkkelkoden>`";`

7. **Skriv programmet** og test om det oppfører seg som planlagt.

8. **Måling av avvik**
   Hvordan vil du gå fram for å måle et ev. avvik fra virkelig akkumulert tid og hvordan vil du korrigere for et slikt avvik i koden? Ev. er det mulig?

6.13 Oppdrag 12 – ESP-NOW – Trådløs overføring av data mellom to ESP32

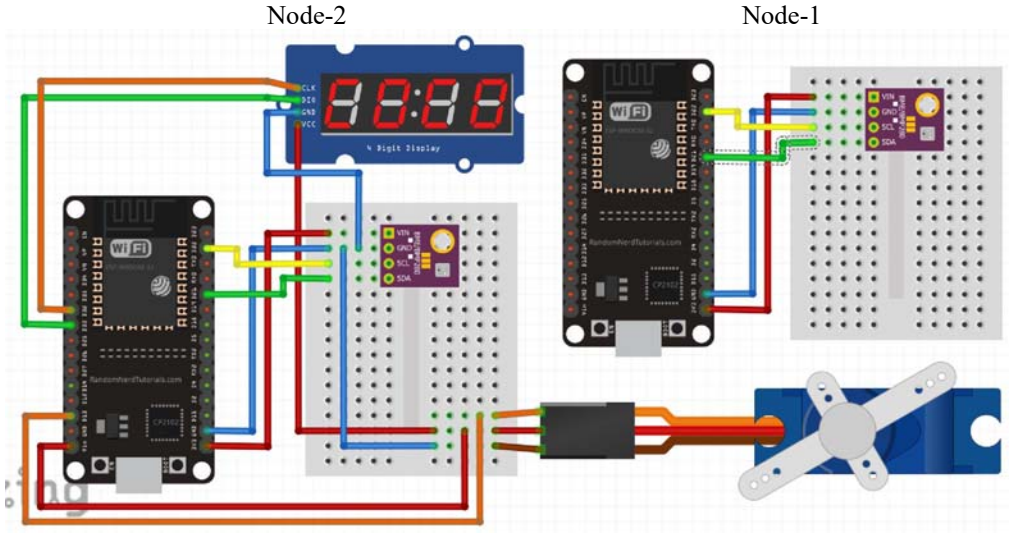
I dette oppdraget skal vi teste ut ESP-NOW som gir mulighet til å overføre kommandoer og måledata mellom to frittstående ESP32-kretser uten å gå via ruter eller Internett, men direkte via Wi-Fi kommunikasjonen til de to kretsene. For mer informasjon om ESP-NOW se avsnitt 5.2 på side 64.

Oppdrag 12 – *Koble opp to ESP32. Den ene, Node-1, skal fungere som sensornode og måle temperatur, luftfuktighet og lufttrykk. Måledata skal overføres trådløst til den andre ESP32, Node-2, som skal ta imot data og vise temperaturen på 7-segment displayet ved noden.*

For å løse oppdraget må man ha to ESP32 som fungerer som Node-1 og -2. Samarbeid gjerne med en kollega.

6.13.1 Oppkobling

Som nevnt trengs to ESP32 og gjerne to BME/P280 som vist i figuren under



Vi har valgt å beholde Node-2 uforandret fra tidligere oppdrag. Node-1 kan i prinsippet også være utstyrt som Node-2, bare at vi kun bruker ESP32 og BME/P280 ved noden.

6.13.2 Kontrollpanelet

Kontrollpanelet skal ikke brukes i dette oppdraget, men vi skal bruke det i senere oppdrag.

6.13.3 Uttesting av ferdige programmer

Denne gangen skal vi gjøre tingene litt annerledes. Vi har laget to eksempelprogrammer, ett for *sensornoden* (Node-1, senderdelen) og ett for *uploadnoden* (Node-2, mottakerdelen). Mottakerdelen av programmet er også forberedt for videresending av data til Circus of Things, derfor kaller vi den *uploadnoden*. Disse er gjengitt i henholdsvis vedlegg B.4.1, side 149 og B.4.2, side 154.

For å teste ut disse to programmene, må vi gjøre følgende:

1. **Finn MAC-adressen ESP ved uploadnoden**

   Denne finner vi ved å laste opp og kjøre scanning-programmet for MAC-adresser som er listet i vedlegg B.2, side 145.

Når vi kjører programmet og trykker på Enable-knappen (Reset) på ESP32 mikrokontrollerkortet og åpner monitoren, så kan vi lese av MAC-adressen nederst i utskriften. Denne består av 6 tosifrete heksadesimale tall, med kolon eller bindestrek mellom.

2. **Åpne programmet ved sensornoden i editoren: ESP32SensorNode.ino**

   Gå til linje 35 og 36 og erstatt adressen som er skrevet i programmet med den vi fant under forrige punkt:

```
// Adressen til mottakeren ved ESP-NOW kommunikasjon,
   dette er MAC adressen skrevet på hex format.
uint8_t broadcastAddress[] = {0x24, 0x6F, 0x28, 0x9E, 0x9F, 0x70};
```

   Erstatt de seks heksadesimale tallene i adressen på linje 36 i programmet (se over).

3. **Kompiler og last opp programmet hos sensornoden**

   Kompiler og last opp ESP32SensorNode.ino

4. **Åpne programmet ved uploadnoden (mottakerdelen) i editoren: ESP32Upload.ino** og kommenter ut opplasting til CoT:

   Gå til linje 93 – 95:

```
void loop() {
  uploadTheData(20000);
}
```

   Dette er den eneste kommandoen i void loop()-funksjonen og skal sørge for å sende de mottatte dataene fra sensornoden til Circus of Things. Inntil videre sløyfer vi denne for kun å sjekke at forbindelsen mellom de to ESP32 fungerer (ESP-NOW). Dette gjør vi ved å kommentere ut linje 94:

```
void loop() {
  //uploadTheData(20000);
}
```

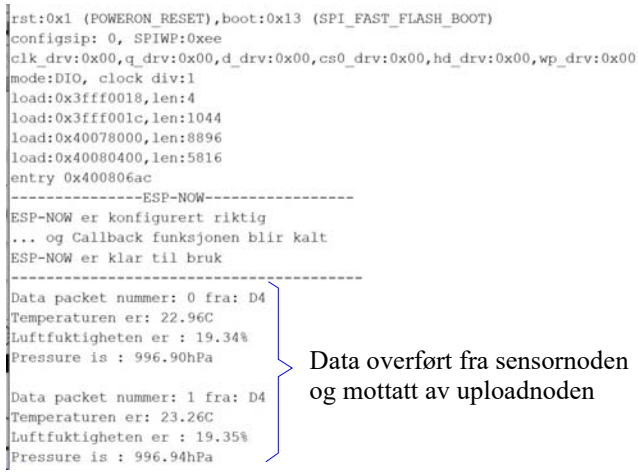
5. **Kompiler og last opp programmet på uploadnoden**

   Kompiler og last opp ESP32SensorNode.ino

6. **Test ESP-NOW forbindelsen**

   Gi begge ESP32-utviklingskortene spenning og sørg for at det er mulig å vise data sendt fra uploadnoden til monitoren på PC'en, åpne monitorvinduet.

Dersom alt fungerer som det skal, skal du se følgende i monitoren:



I tillegg skal temperatur, luftfuktighet, lufttrykk og antallet sendte målinger vekselvis vises på displayet på uploadnoden.

6.13.4 Programmet

Vi skal nå forsøke å bygge opp et særdeles strippet utgave av programmet som overfører data mellom to ESP32-utviklingskort som demonstrert over. En minimumsutgave vil gi oss bedre oversikt over hvordan programmet fungerer, men vil også ha en del ulemper:

- Siden ingen eller få funksjoner for å varsle feilmeldinger er tatt med så vil vi, dersom programmet ikke oppfører seg som forventet, ikke få noen tilbakemelding om hva som er galt.
- Siden vi har valgt å redusere antall funksjoner, så vil programmet være mindre modulært og vanskeligere å bygge videre på, dessuten vil det gi mindre oversikt når kompleksiteten til programmet øker.

Vi anbefaler derfor sterkt å sammenligne den strippede versjonen med den utbygde funksjonen som ble testet innledningsvis.

Oppbygging av strippet kode av sendernoden: oppdrag-12-ESP32sensornode

1. Inkluder biblioteker

```
#include <esp_now.h>
#include <WiFi.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
```

WiFi.h er knyttet til den trådløse Wi-Fi overføringen til Internett via en ruter, mens esp_now.h er nødvendig for å kunne kommunisere direkte mellom ESP32-utviklingskort. Adafruit_Sensor.h og Adafruit_BME280.h er knyttet til avlesning av målinger fra BME280. Alle biblioteker inkluderes normalt i starten av programmer.

2. **MAC-adresse mottaker**

```
uint8_t broadcastAddress[] = {0x24, 0x6F, 0x28, 0x9E, 0x9F, 0x70};
```

Her defineres variabelen broadcastAddress som angir MAC-adressen til Uploadnoden, dit måledataene skal sendes.

3. **Definer en struktur som holder måledata fra BME280**

```
Adafruit_BME280 bme280;
```

Her opprettes en struktur bme280 av typen Adafruit_BME280 som skal holde måleresultatene. Denne plasseres foran void setup()-funksjonen. Tilsvarende kan gjøres for BMP280 om det er den vi bruker.

4. **Antall målinger**

```
int consecutive_messages = 0;
```

Denne variabelen teller fortløpende opp antall målinger gjennom forløpet, fra programmet ble startet. Variabelen er global og plasseres foran void setup()-funksjonen.

5. **Definer en type struktur som holder data for overføring** mellom ESP-NOW enhetene

```
typedef struct {
  int instance;
  float temperature;
  float humidity;
  float pressure;
  String source;
} ESPNOWData;
```

Denne strukturen som er av typen ESPNOWData inneholder alle de tre målingene av Temperatur, luftfuktighet og lufttrykk. I tillegg inneholder den antall målinger og siste byte i adressen til sensornoden som har levert dataene. Definisjonen av strukturen plasseres foran void setup()-funksjonen.

6. **Deklarer en struktur "packet" av typen ESPNOWData**

```
ESPNOWData packet;
```

"Packet" er den strukturen som holder data som oversendes fra sensornoden til uploadnoden.

7. **Void setup()-funksjonen**
Her definerer vi innholdet i void setup()-funksjonen

- *Sett opp kommunikasjon til monitoren*

```
Serial.begin(115200);
```

Vi velger 115200 baud (her bit pr. sekund) for datahastigheten.

- *Initiering av ESP-NOW*

```
initESPNOW();
```

Dette er en egendefinert funksjon som inneholder en rekke funksjoner for å få ESP-NOW operativt, og som vi skal komme tilbake til om litt

- *Initiering av BME/P280*

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

Her initierer vi sensoren BME280 (ev. BMP280) og angir adressen til sensoren på I^2C-bussen

8. **Void loop()-funksjonen**
I dette programmet er void loop()-funksjonen relativt kort

- *Øk antallet akkumulerte målinger*

```
consecutive_messages++;
```

Her bruker vi den forenklede formen for å øke innholdet i en variabel med 1

- *Les av måledatene og legg dem inn i strukturen "packet"*

```
packet.pressure = bme280.readPressure();
packet.instance = consecutive_messages;
packet.temperature = bme280.readTemperature();
packet.source = "5C";
packet.humidity = bme280.readHumidity();
```

Vi legger merke til packet.source er angitt som siste siffer i adressen. Vi merker oss at dette er siste siffer i MAC-adressen til ESP32 knyttet til sensornoden og har selvfølgelig en annen verdi enn den adressen vi bruker for å adressere uploadnoden.

- *Send data til uploadnode*

```
esp_now_send(broadcastAddress, (uint8_t *) &packet, sizeof(packet));
delay(2000);
```

Sendfunksjonen inneholder adressen til mottakeren (uploadnoden) (`broadcastAddress`), en peker som peker på pakken med måledate (`&packet`) og antallet byte som denne pakken inneholder (`sizeof(packet)`). Til sist går det 2 sekunder mellom hver måling.

9. **Initialisering av ESP-NOW** (`initESPNOW()`)
Denne egendefinerte funksjonen inneholder en rekke funksjoner. La oss se på hver av dem:

- *Konfigurer ESP-NOW i stasjonsmodus*

```
WiFi.mode(WIFI_STA);
```

Når vi ønsker at ESP32-utviklingskort skal kommunisere med hverandre så må vi konfigurere nettverket i stasjonsmodus (`WIFI_STN`).

- *Initier ESP-NOW*

```
esp_now_init();
```

Denne funksjonen returnerer en statusrapport som kan brukes til å melde om feil som inntreffer. Se vedlegg C.1.1, side 162 for detaljer.

- *Bygg opp struktur for paring av ESP32*
Det trengs opplysninger for å opprette informasjon mellom sensornoden og uploadnoden, denne informasjonen samles i denne strukturen som defineres og fylles opp i følgende linjer:

```
esp_now_peer_info_t peerInfo;
```

Først definerer vi strukturen `peerInfo` av typen `esp_now_peer_info_t`.

Dernest fyller vi strukturen med informasjon:

```
memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.channel = 0;
peerInfo.encrypt = false;
```

Først kopieres mottakeradressen på 6 byte (`broadcastAddress, 6`) over strukturen ved hjelp av funksjonen `memcpy()`. Dernest legger vi inn `peerInfo.channel` som vi setter til 0, og bestemmer at vi ikke skal kryptere dataene (`false`).

- *Opprett peer-kommunikasjonen*

```
esp_now_add_peer(&peerInfo);
```

Her opprettes selve kommunikasjonen på basis av dataene som ble lagt inn over.

10. Opprett Callback-funksjonen

Dette er en funksjon som kalles når data er sendt. I vårt tilfelle så er den tom.

```
void activateOnCallback(const uint8_t *mac_addr, esp_now_send_status_t sta-
tus) { }
```

Dersom vi ser på den mer omfattende utgaven av programmet som vi brukte innledningsvis, vil vi se at det her er lagt inn melding når overføringen av data er blitt utført, ev. var vellykket eller mislykket.

11. Skriv programmet og test det

Ta gjerne en till på det tidligere nevnte testprogrammet og søk støtte der om dere er usikre på detaljer. Studer også hvordan statusrapporter og feilmeldinger er brukt. Lagg gjerne deler av programmet i funksjoner.

Oppbygging av strippet kode av uploadnoden: oppdrag-12-ESP32Uploadnode

I dette programmet bruker vi displayet for å vise at overføringen av data fungerer.

1. Inkluder biblioteker

```
#include <esp_now.h>
#include <WiFi.h>
#include <Wire.h>
#include <TM1637Display.h>
```

esp_now.h og WiFi.h er knyttet til den trådløse Wi-Fi overføringen mellom ESP32-utviklingskortene. `#include <TM1637Display.h>` er knyttet til 7-segment displayet. Disse legges i starten av programmet.

2. Definer en struktur som initialiserer displayet TM1637

```
const byte PIN_CLK = 32;
const byte PIN_DIO = 33;
TM1637Display display(PIN_CLK, PIN_DIO);
```

Her opprettes en struktur `display()` av typen `TM1637Display` som blant annet krever klokkelinje (`PIN_CLK`) og datalinje `(PIN_DIO)`. Portene på ESP32 som brukes til disse er definert som konstanter.

3. **Definer en type struktur som holder data for overføring** mellom ESP-NOW enhetene

```
typedef struct {
  int instance;
  float temperature;
  float humidity;
  float pressure;
  String source;
} ESPNOWData;
```

Denne strukturen som er av typen ESPNOWData inneholder alle de tre målingene av temperatur, luftfuktighet og lufttrykk. I tillegg inneholder den antall målinger og siste byte i adressen til sensornoden som har levert dataene. Definisjonen av strukturen plasseres foran void setup()-funksjonen. Mottatte data fra sensornoden pakker data i en slik struktur.

4. **Deklarer en struktur "packet" av typen ESPNOWData**

```
ESPNOWData packet;
```

"Packet" er den strukturen som holder data oversendt fra sensornoden til uploadnoden.

5. **Void setup()-funksjonen**
Her definerer vi innholdet i void setup()-funksjonen

- *Sett opp kommunikasjon til monitoren*

```
Serial.begin(115200);
```

Vi velger 115200 baud (her bit pr. sekund) for datahastigheten

- *Initiering av ESP-NOW*

```
initESPNOW();
```

Dette er en egendefinert funksjon som inneholder en rekke funksjoner for å få ESP-NOW operativt på lignende måte som for programmet for sensornoden, og som vi skal komme tilbake til om litt

- *Sett lysstyrken til displayet TM1637*

```
display.setBrightness(7);
```

Her setter vi lysstyrken til 7-segment displayet til maksimal styrke

6. **Void loop()-funksjonen**
I dette programmet er denne relativt kort

void loop() {}

... da den foreløpig er helt tom. Etter hvert skal vi legge inn kommando for sending av data ut på Internett i neste oppdrag.

7. **Initialisering av ESP-NOW** (`initESPNOW()`)
Denne egendefinerte funksjonen inneholder en rekke funksjoner. La oss se på hver av dem:

- *Konfigurer ESP-NOW i stasjonsmodus*

  ```
  WiFi.mode(WIFI_STA);
  ```

  Når vi ønsker at ESP32-utviklingskort skal kommunisere med hverandre så må vi konfigurere nettverket i stasjonsmodus (`WIFI_STN`).

- *Initier ESP-NOW*

  ```
  esp_now_init();
  ```

  Denne funksjonen returnerer en statusrapport som kan brukes til å melde om feil som inntreffer. Se vedlegg C.1.1, side 162 for detaljer.

- *Angi navn på callback-funksjon*

  ```
  esp_now_register_recv_cb(activateOnCallback);
  ```

  Argumentet til `esp_now_register_recv_cb()` angir navn på funksjonen som skal kalles når det registreres innkomne data fra sensornoden. Innkomne data registreres med et internt software-interrupt som sender programmet til callback-funksjonen hvor de innkomne data blir tatt hånd om. Vi kommer ganske snart tilbake til dette.

8. **Definisjon av callback-funksjonen**
   Dette er funksjonen som tar hånd om de innkomne dataene og i vårt tilfelle skriver dem ut til monitoren og til 7-segment-displayet.

   - *Callback-funksjones navn og argumenter*
     Navnet til funksjonen er gitt som under initialiseringen som omtalt over.

     ```
     void activateOnCallback(const uint8_t * mac, const uint8_t *incomingData,
     int len) {
     // Funksjonskroppen...
     }
     ```

     `*mac` – er pekeren som peker på MAC-adressen til sensornoden som har sendt dataene
     `*incommingData` – peker startpunktet for hvor de mottatte dataene ligger i lageret
     `len` – angir antall mottatte byte og som omfattes av incommingData.

   - *Kopiering av data fra `incommingData` til strukturen `packet`*
     Funksjonen `memcpy` kopiere innholdet i `incommingData` til strukturen `packet`:

     ```
     memcpy(&packet, incomingData, sizeof(packet));
     ```

     `sizeof(packet)` angir størrelsen på strukturen `packet` i byte.

   - *Utskrift til monitor*
     Vi har valgt å skrive ut mottatte data til monitoren på PC

     ```
     Serial.print("Data packet nummer: ");
     Serial.print(packet.instance);
     Serial.print(" fra: ");
     Serial.println(packet.source);
     Serial.print("Temperaturen er: ");
     Serial.print(packet.temperature);
     Serial.println("C");
     Serial.print("Luftfuktigheten er : ");
     ```

```
  Serial.print(packet.humidity);
  Serial.println("%");
  Serial.print("Pressure is : ");
  Serial.print(packet.pressure/100);
  Serial.println("hPa");
  Serial.println();
```

Som vi ser så hentes dataene direkte ut av strukturen "packet"

- *Utskrift til 7-segment display*
  I denne omgang har vi valgt kun å skrive ut temperaturen med to desimaler og med fast komma etter 2. siffer:

  ```
  showAtDisplay(packet.temperature);
  ```

  Vi har lagt utskriften i en egen funksjon kalt `showAtDisplay()` hvor argumentet er måledataene som skal skrives ut. Funksjonen omtales under.

  Det oppmuntres til å studere displayfunksjonene i testprogrammet da disse er langt mer avanserte og gir flere muligheter for mer tilpasset utskrift.

9. Utskriftsfunksjon til 7-segment display TM1637

Som vi ser så gjør vi tallet om til et firesifret heltall med fast komma etter andre siffer. Dette gjøres enkelt ved å multiplisere temperaturen med 100. Dette går bra så lenge temperaturen er mellom 0 – 99°C.

```
void showAtDisplay (float showNumber){
  showNumber = showNumber*100;
  display.clear();
  display.showNumberDecEx(int(showNumber), 0b01000000, false, 4, 0);
  delay(500);
}
```

Displayet må først tømmes før tallet skrives ut. Deretter sendes tallet til displayet med funksjonen hvor argumentene betyr følgende:

`int(<argumentnavn>)` – Angir tallet vi ønsker å vise på displayet omgjort til et heltall
`0b01000000` – Setter komma etter 2. siffer
`false` – Angir at vi ikke ønsker å sette inn 0 foran om tallet er mindre enn 10
`4` – Angir at displayet har 4 siffer
`0` – Angir at det mest signifikante sifferet skal være plassert til venstre på displayet

10. Skriv programmet

Skriv programmet og undersøk om det fungerer som forventet.
Ta gjerne utgangspunkt i løsningsforslaget som finnes i vedlegg C.1.

6.14 Oppdrag 13 – ESP-NOW – Overføring data fra sensornoden til CoT

I dette oppdraget skal vi kombinere overføring av data mellom en sensornode og en uploadnode, som i sin tur videresender informasjonen til Circus of Things som visere de overførte dataene på konsoller i et panel. For mer informasjon om ESP-NOW se avsnitt 5.2 på side 64.

Oppdrag 13 – *To ESP32-utviklingskort er forbundet trådløst ved hjelp av ESP-NOW. Sensornoden, Node-1, måler temperatur, luftfuktighet og lufttrykk som sendes over til Uploadnoden, Node-2. Målet temperatur vises på 7-segment displayet ved noden. I tillegg skal Uploadnoden sende data til Circus of Things som så skal vise målingene av temperatur, luftfuktighet og lufttrykk på konsoller.*

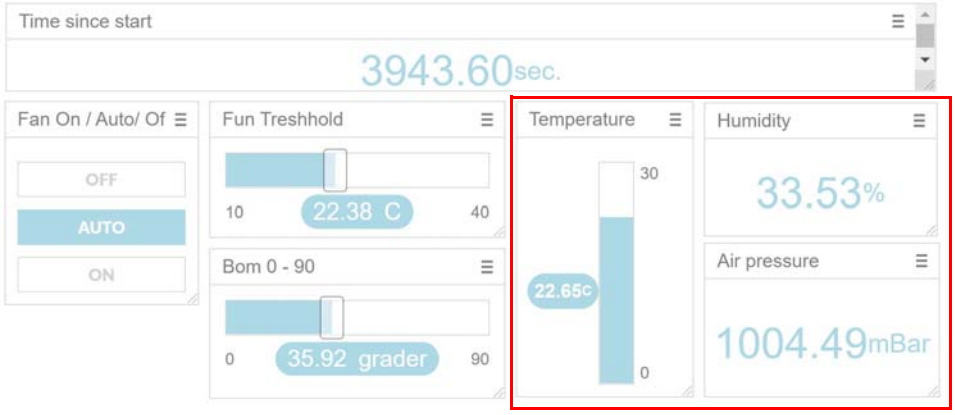
For å løse oppdraget bygger vi videre på oppdrag 12. Samarbeid gjerne med en kollega.

6.14.1 Oppkobling

Det trengs ingen ytterligere oppkobling for å løse dette oppdraget.

6.14.2 Kontrollpanelet

I dette oppdraget skal vi bruke kontrollpanelet ved Circus of Things, men vi skal bruke de konsollene som er utviklet i tidligere oppdrag, nemlig konsollene for visning av temperatur, luftfuktighet og lufttrykk som er rammet inn på figuren under.



Det er selvfølgelig fritt fram og inkludere de øvrige delene av programmene slik at hele panelet kan tas i bruk slik vi har omtalt det i oppdragene 1 til 10.

6.14.3 Uttesting av ferdige programmer

Vi skal nå teste om vi får Uploadnoden til å videreformidle sensordataene til Circus of Things og vise resultatene på konsollene for visning av temperatur, luftfuktighet og lufttrykk. Dette forutsetter at konsollene er intakte og vi fortsatt har konto ved CoT.

Igjen tar vi utgangspunkt i de to ferdige programeksemplene i vedlegg B.4.1, side 149 og vedlegg B.4.2, side 154.

1. **Test overføring av data fra uploadnoden til Circus of Things**
   Vi skal nå åpne for overføring av data til CoT:

   - *Åpne uploadnode-programmet i editoren*: ESP32Upload.ino
     Gå til linje 57 – 64 i programmet og legg inn personlige nøkkeldata:

```
char ssid[] = "<nettverksnavn>"; // Legg in navnet på nettverket ditt
char password[] =  "<passord>";// Legg in nettverkspassordet ditt
char server[] = "www.circusofthings.com";// Adressen til CoT
char temperature_key[] = "<nr.>";// Legg inn nøkkelkoden til temperaturkonsoll
char humidity_key[] = "<nr.>";// Legg inn nøkkelkoden til luftfuktighetkonsoll
char pressure_key[] = "<nr.>";// Legg inn nøkkelkoden til lufttryk-konsoll
char order_key[] = "<nr.>";// Legg inn nøkkelkoden til styring av rele
char token[] = "<personlig token>";// Legg inn din login token
```

   - *Legg kommando for opplasting* av data til CoT
     Gå til linje 93 – 95:

```
void loop() {
  uploadTheData(20000);
}
```

     ... og fjern // slik at opplasting til CoT igjen kan skje.

   - *Last opp programmet*
     Kompiler og last opp programmet i uploadnoden.

2. **Test forbindelse med CoT**
   Sjekk at konsollene i CoT mottar data fra sensornoden via uploadnoden.

6.14.4 Programmet

Vi tar utgangspunkt i programmet som vi utviklet i Oppdrag 12 og utvider vår minimumsløsning og legger inn nøkkeldata for å overføre målinger til CoT, i tillegg må vi legge over måledatene fra strukturen (packet) og inn i funksjonene som overfører målingene til CoT.

Oppbygging av strippet kode av sendernoden: oppdrag-13-ESP32sensornode

1. **Inkluder biblioteker**
   Vi må nå supplere biblitekene med biblioteket fra Circus of Things

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

   Dette legges sammen med de andre bibliotekene i starten av programmet.

2. **Sett opp strategiske data for Upload til Circuit of Things**

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

Her må man legge inn de verdiene som passer til hver enkelts lokale ruter, med navn og passord. I tillegg til nøkkelkoder for konsollene i CoT. token er også en personlig kode som er gitt av CoT under "acount".

3. **Lag en struktur circusESP32 av typen CircusESP32Lib**
   med argumentene for servernavn ved CoT, lokal nettverksnavn (ssid) med tilhørende passord for ruteren.

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

4. **void setup()-funksjon**

   Denne er uforandret fra oppdrag-12- ESP32UploadNode

5. **void loop()-funksjon**

   Her legger vi inn kallet til funksjonen som sender data til Circus of Things

```
void loop() {
  uploadTheData(10000);
}
```

   Funksjonen `uploadTheData()` er en egendefinert funksjon som sender data til CoT, denne skal vi straks komme tilbake til. Argumentet angir tiden mellom hver overføring av data i millisekunder.

6. **Funksjon for overføring av data til CoT** void uploadTheData(int delayLength)
   Under er vist hele funksjonen. Vi vil deretter gå gjennom funksjonen ledd for ledd.

```
void uploadTheData(int delayLength){
  digitalWrite(LED_BUILTIN, HIGH);

  circusESP32.begin();

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

  digitalWrite(LED_BUILTIN, LOW);
  delay(10);

  WiFi.disconnect(true);
  WiFi.mode(WIFI_STA);

  delay(delayLength);
}
```

   - *Slå på intern lysdiode*
     For å indikere at data overføres slås den interne lysdioden på kortet på i det overføringen starter og slås av når den er ferdig:

```
digitalWrite(LED_BUILTIN, HIGH);
...
```

```
digitalWrite(LED_BUILTIN, LOW);
```

- *Initier overføringen*
  Initier overføring av data til CoT

```
circusESP32.begin();
```

- *Overfør data*
  Så kan hver parameter overføres med riktig nøkkelkode og token for hver parameter

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

- *Sett opp Wi-Fi*
  Mens overføringen gjøres stenges Wi-Fi samtidig som Stasjonsmodus settes opp på ny

```
WiFi.disconnect(true);
WiFi.mode(WIFI_STA);
```

- *Vent en tid før neste overføring skal skje*

```
delay(delayLength);
```

  Ventetiden overføres til funksjonen via argumentet.

7. Legg inn koden og test ut programmet

Sørg for at du er logget på Circus of Things og sjekk at dataene blir overført til de riktige konsollene. Studer gjerne løsningsforslaget som finnes i vedlegg C.2, side 167.

7 Relevante sensorer og aktuatorer for oppdragene

I dette kapittelet skal vi se nærmere på noen av sensorene som er benyttet i oppdragene som omtales i heftet. La oss først se på BME280.

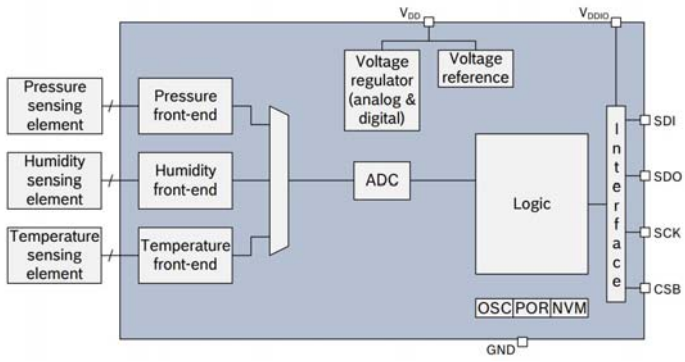
7.1 Barometer, temperatur- og fuktighetsmåler– BME280 (Bosch)

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[30]:

- *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[31]
- *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)

30. Informasjonen er hentet fra: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf
31. 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.

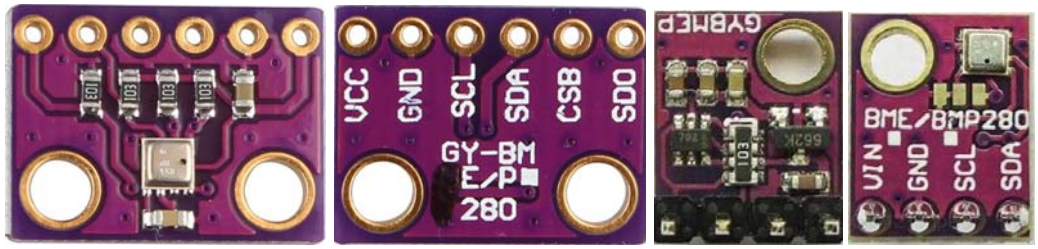
Figuren viser et blokkdiagram over sensoren:



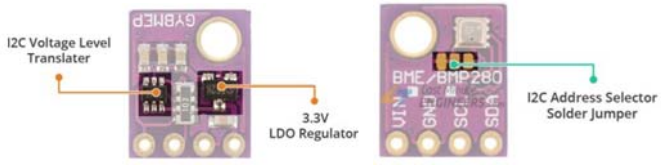
Vi legger merke til at det er en felles ADC for alle tre sensorene og at power supply for sensordelen og logikken er separert, hvilket er gunstig mht å redusere støy.

Oppkobling

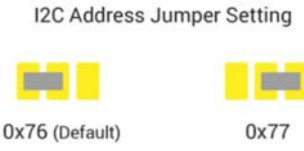
Figuren under viser pinningen til to utgaver til BME280:



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.

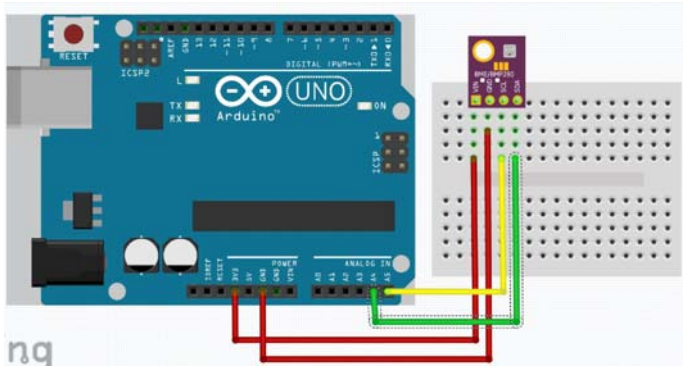


Spenningsregulator



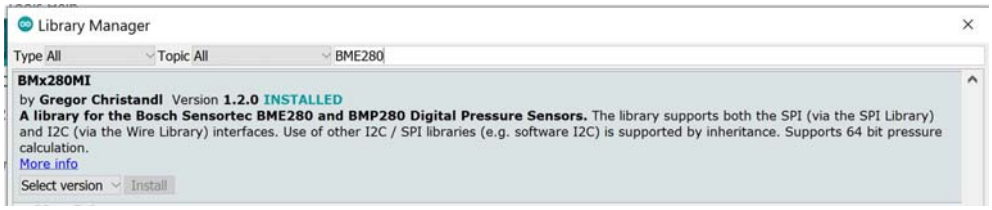
Adresse-setting

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[32]. 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:

```
#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
```

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

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();
```

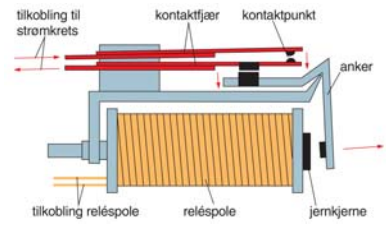
Er en funksjon som gjenkjenner hvilken krets som er tilkoblet. `bmx280.isBME280();` returnerer "1" når vi har tilkoblet en BME280 og "0" når det er en BMP280 som er koblet til.

7.2 Releer

Det sier seg selv at en mikrokontroller ikke uten videre kan styre store strømmer eller spenninger siden de opererer med spenninger fra typisk 3,0 – 5,0 V og strømmer på maksimalt 20 – 40 mA. Vi trenger derfor en komponent som lar seg styre med lave spenninger og strømmer, men som selv kan styre store strømmer og spenninger. Vi skal se nærmere på releer som nettopp har denne egenskapen.

7.2.1 Releets funksjon

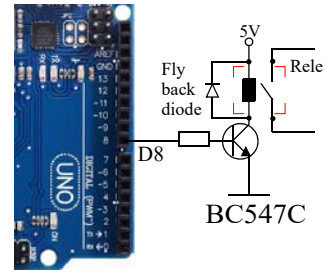
Figuren høyre[33] viser hvordan et tradisjonelt rele fungerer. Spolen til en elektromagnet er koblet til lavspenningsdelen til f.eks. en mikrokontroller. Når spolen får strøm fra kontrolleren vil *ankeret* trekkes til og en bryter legges over og slutter en høyspenningskrets. Som vi ser er det en rent elektromagnetisk-mekanisk bryter.



Det som ikke er så uvanlig når det gjelder tilkobling av reler til f.eks. Arduino er at selv strømmen som kreves for å dra ankeret i posisjon er større enn det en Arduino normalt klarer. En Arduino kan normalt lever fra 20 - 40 mA, mens et rele kanskje trenger en noe større strøm.

7.2.2 Transistordriveren

En transistor er en komponent som gjør det mulig å styre en større strøm med en liten strøm eller liten spenning. Derfor kan det være ideelt å bruke en transistor mellom en Arduino-port og et rele som vist på figuren til høyre.



"Fly back" dioden

Vi legger merke til dioden som er koblet over spolen. Den er montert slik ar det normalt ikke går strøm i den. Dette er en viktig komponent som skal beskytte transistoren når strømmen i

33.Store norske leksikon

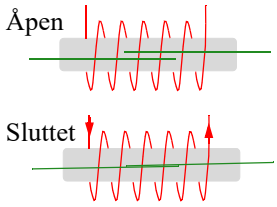
spolen brytes. Når en strøm brytes brått i en spole, så vil dannes en motspenning som hindrer strømmen i å forsvinne brått. Denne motspenningen kan lett bli svært høy og ta knekken på transistoren. Flyback-dioden derimot vil kortslutte og dempe spenningstoppen. Dioden er derfor svært viktig. Uten den vil transistoren ganske snart ødelegges.

7.2.3 Reed releet

Det finnes mange typer releer. Et av de mest populære i sammenheng med små kretser med relativt små strømmer er reed-releer. Som navnet sier så består de av to tynne metallfjærer montert i et rør av glass og som normalt ikke berører hverandre (se bildet under). Disse metallfjærene er av et metall som lar seg magnetisere. Dersom vi nærmer oss disse metallfjærene med en magnet vil releet slå inn og slutte kretsen. De to fjærene blir magnetiske slik at de tiltrekker hverandre.

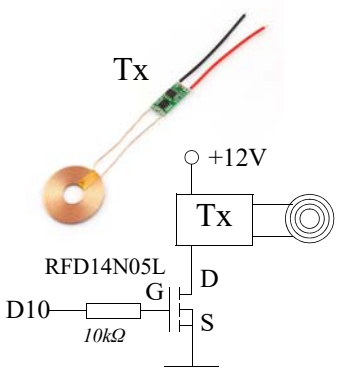
Reed reler egner seg derfor godt i en sammenheng hvor en elektrisk krets skal sluttes når en mekanisk arm nærmer seg et punkt.

En kan også vikle en spole rundt glassrøret og på den måten oppnå at metallfjærene slutter kretsen når det går en strøm i spolen. Dette siste er gjerne tilfellet for reed releer i salg. Glassrør med fjærer og spole er montert i et plasthus som det stikker ledninger ut av som vist på figuren til høyre. Også disse må gjerne ha en drivertransistor koblet mellom kontrollerkortet og spolen.



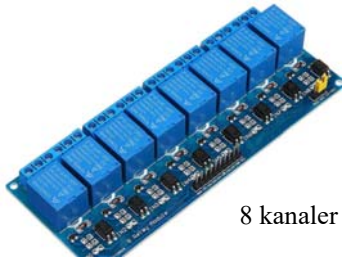
7.2.4 FET-transistorer for styring av store likestrømmer.

Dersom vi har behov for en relativt stor strøm, men begrenset spenning så kan vi f.eks. benytte FET-transistorer. Disse kan gjerne drives direkte fra en av de digitale portene til en Ardioino siden de er spenningstyrte og ikke krever noen strøm. Det er også enkelt å styre en slik FET-transistor selv komponenten den styrer trenger en høyere spenning en 5V. Figuren til høyre viser hvordan en FET-transistor slå av på en strøm på mange amper med spenninger på noen titals volt. I vårt eksempel slåes det på en ladespole (Tx) for overføring av trådløs energi til en mobiltelefon.



7.2.5 Releer med innebygget driver

Det leveres en rekke ulike relekort med fra ett til 8 releer som kan styres direkte fra en Arduino siden kortet inneholder drivere. På figuren under er det vist et par eksempler. Disse er utstyrt med opto-kobler på inngangen slik at lav- og høyspenningsdelen og er galvanisk skilt fra hverandre. Relene tåler 10A, 230V AC, og 10A, 30V DC.



8 kanaler

8 kanaler

2 kanaler

Det finnes en mengde slike varianter om man leter.

7.3 Servomotorer

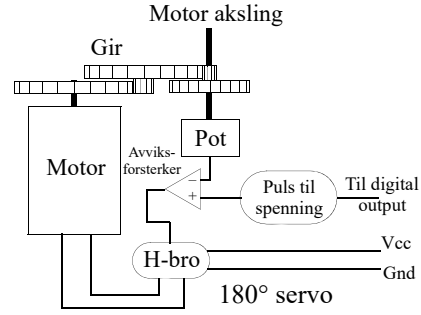
Det finnes flere typer servomotorer. Her skal vi ta for oss 180° og 360° motorer. 180° graders motorer kan dreie en arm til en gitt vinkel fra 0 – 180° (f.eks. FS90). En 360° motor oppfører seg som en vanlig motor (FS90R). Spesielt med motorene er at dreievinkelen eller hastigheten kan styres ganske nøyaktig ved hjelp av lengden av en puls. Her vil vi behandle begge typende under ett.

FS90

FS90R

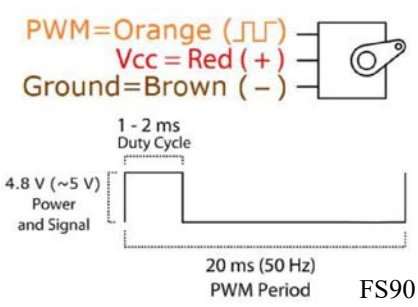
7.3.1 Virkemåte

En 180° servo er en motor hvor en har full kontroll på vinkelen akslingen skal dreie. Normalt fra 0° til 180°. Dette skjer ved en intern tilbakekobling som vist i figuren til høyre. Servoen styres av en puls på styreinngangen. FS90 er en slik 180° servo.



En *kontinuerlig roterende servo* kan dreie 360° og vil fungere som en vanlig motor. En slik vil ikke ha tilbakekobling som vist på figuren til høyre.

Motorene tilføres normalt tre ledninger. I tillegg til spenning, Vcc, typisk 5V, og jord (GND) så tilføres et styresignal som hentes fra en av de digitale portene hos mikrokontrolleren. Det må være en port som kan tilby PWM – Pulsbreddemodulasjon.



FS90

FS90: 180° motorene styres ved hjelp av et pulstog[34]. Figuren til venstre viser en typisk situasjon for FS90. Den forlanger en puls på mellom 1 – 2 ms, som gjentar seg med en periode på 20ms (50Hz).

En pulslengde på 1,5 ms vil posisjonere servoen i posisjon 0°. En puls på 1,0 ms vil dreie den –90° mot høyre, mens en puls på 2,0 ms vil dreie den 90° mot venstre. Ved å endre på pulslengden så endres posisjonen. Fargene på figuren over angir fargene på de tre ledningene til servoen.

FS90R: En **360° motor** styres på tilsvarende måte av en lignende puls. Dette gjelder f.eks. FS90R der en pulslengde på 1,5 ms ±0,05 ms vil sørge for at servoen står i ro. En puls på 0,9 ms vil sørge for at motoren får full fart mot høyre, mens en puls på 2,1 ms vil gi full fart mot venstre. Ved å endre på pulslengden så endres farten.

En kan imidlertid erfare at motoren ikke står stille ved en pulslengde på 1,5 ms. I så fall finnes det en trimmeskrue på undersiden av servoen som kan justeres slik at den står stille ved den angitt pulslengden på 1,5 ms. Denne motoren har normalt en vindu hvor den skal være i ro på ±0,045 ms[35].

7.3.2 Viktige parametere for servoer

Av viktig parametere kan vi nevne følgende. Vi bruker fortsatt FS90 som eksempel på en 180° servomotor og FS90R som eksempel på en 360° servomotor:

- **Driftsspenning og strøm:** Denne angir typisk spenningsområde for servoene. For både FS90 og FS90R er dette 4,8 – 6V, de egner seg derfor godt for bruk med Arduino. Servoene har selvfølgelig et strømtrekk som varierer fra når de er i ro (typ. 5 – 6mA) til de er i aktiv bevegelse (typ. 100–120mA). Dersom motorene blokkeres vil strømmen øke betydelig typisk 700 – 800mA for FS90 og 550 – 650 for FS90R, hvilken lett kan være til skade om det får stå på en stund.
- **Dimensjoner:** Størrelse og vekt er viktige dersom servoen skal brukes i fly. Dette kan være tilfelle dersom servoen skal dreie side- eller haleror. Både FS90 og FS90R har en vekt 9g og dimensjoner (23,2 x 12,0 x 22,0 mm).

34.http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf
35.https://www.pololu.com/product/2820

- **Styrke:** Styrken oppgis som dreiemomentet (kraft x arm) når motoren hindres i å rotere, dette kalles "stall torque". For både FS90 og FS90R er det oppgitt til å ha en toppverdi på 1,3 – 1,5 kg cm når motoren hindres i å rotere ("staller"). Dreiemomentet er avhengig av hvilken spenning motoren tilføres. Det opplyses også om hvilket materiale giret er laget av da dette har betydning for hvilken påkjenning det tåler. Det må antas at nylon, som i dette tilfellet, er svakere enn metall.
- **Hastighet:** For en 180° servo vil hastigheten angis ved hvor lang tid det tar å dreie den 60°. For FS90 er det ca. 0.1 – 0,12 sek. avhengig av spenningen. Jo høyere spenning, jo raskere. For en 360° servo angis hastigheten som antall omdreininger pr minutt (rpm). For FS90R er denne på 110 – 130 rpm avhengig av spenningen.
- **Tilleggsutstyr:** Normalt følger det med ulike armer og festeskiver med servoen for ulik bruk. Disse tres ned på akslingen som gjerne er riflet. Settet som er vist på figuren over følger gjerne alle servomotorer av denne typen.

7.3.3 Programmering av servoer

Arduino har egne bibliotek som gir oss et sett av funksjoner som gjør det lett å programmere servomotorer. Det er viktig å merke seg at det kan være forskjellige biblioteker avhengig av om vi bruker ESP32 eller f.eks. Arduino UNO.

- **Hent bibliotek:** Biblioteket for styring av servoer er standard for Arduino og trenger normalt ikke å installeres. Dersom vi bruker ESP32 må vi installere et eget bibliotek som kan hentes her: https://github.com/RoboticsBrno/ServoESP32
- **Inkludering av biblioteket:**
  Biblioteket inkluderes ved å skrive
  ```
  #include <Servo.h>
  ```
  øverst i fila.
- **Deklarasjon av servoene:**
  Dernest må vi deklarere servoen eller servoene med navn og en type. Dersom vi ønsker å bruke flere servoer gir vi dem forskjellige navn. Det kan f.eks. være tilfelle dersom vi skal laga en robot og trenger en servo for hvert av de to hjulene. I alt kan man definere 8 servoer for en en Arduino UNO:
  ```
  Servo myleftservo;  // Deklarerer objektet myleftservo av typen servo
  Servo myrightservo; // Deklarerer objektet myrightservo av typen servo
  Servo myarmservo;   // Deklarerer objektet myarmservo av typen servo
  ```
  Deklarasjonene gjøres før setup()-funksjonen

- **Tilkobling og frakobling**
  Dernest må vi fortelle Arduino'en hvilken digital utgang <pin> som skal kobles til styreinngangen på servoen.

```
myleftservo.attach(9);  // Knytt venstre servo til pinne 9
myrightservo.attach(10);// Knytt høyre servo til pinne 10
myarmservo.attach(11);  // Knytt servoen styrer armen til pinne 11
```

  Disse tilordningene gjøres i setup-funksjonen
  Tilsvarende kan en frakoble servoene om det er behov for det med følgende kommandoer:

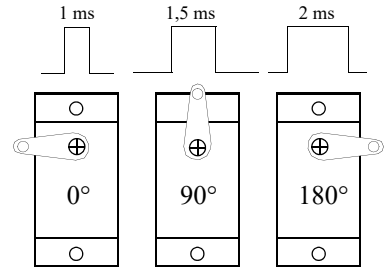
```
myleftservo.detach();  // Koble venstre servo fra pinne 9
myrightservo.detach(); // Koble høyre servo fra pinne 10
myarmservo.detach();   // Koble høyre servo fra pinne 11
```

  Det kreves ikke noe pinnenummer ved frakobling.

- **Styring av servo med "write()":**
  Servoen styres med kommandoen – "write"

```
myarmservo.write(60);
```

  Kommandoen skrives normalt i loop()-funksjonen.
  Hos en *180° servo* vil vinkel angi dreiningsvinkelen til servoen i grader (0° – 180°).
  Hos en *kontinuerlig roterende servo* vil *vinkelen* angi motorens hastighet og retning. Her vil 0° angi full hastighet i den ene retningen, 90° være i ro, mens 180° angir full hastighet i motsatt retning.
  Kommandoene plasseres i loop()-funksjonen etter behov



Egentlig styres servoen av pulser som antydet på figuren over.

- **Styring av servo med "writeMicroseconds()":**
  Denne kommandoen fungerer omtrent som "write", men angir direkte lengden av pulsen i mikrosekunder. Følgende er vanlige verdier for *en 180° servo*:
  1000µs dreies til posisjon 0°
  1500µs dreies til posisjon 90°
  2000µs dreies til posisjon 180°
  Følgende er vanlige verdier for *en kontinuerlig roterende servo*:
  1000µs full fart *mot* urviseren
  1500µs i ro
  2000µs full fart *med* urviseren
  Det er viktig å unngå at servoen står å stanger mot et ytterpunkt, da dette trekker mye strøm, samtidig som det kan skade komponenten.

- **Monitoreringskommandoen – "read"**
  Kommandoen "read" returnerer siste "write"-verdi sendt til servoen.

```
vinkel = myarmservo.read();
```

  Kommandoen plasseres i loop()-funksjonen etter behov.

7.4 Numeriske 7 segment display – TM1637

7.4.1 Kort omtale:

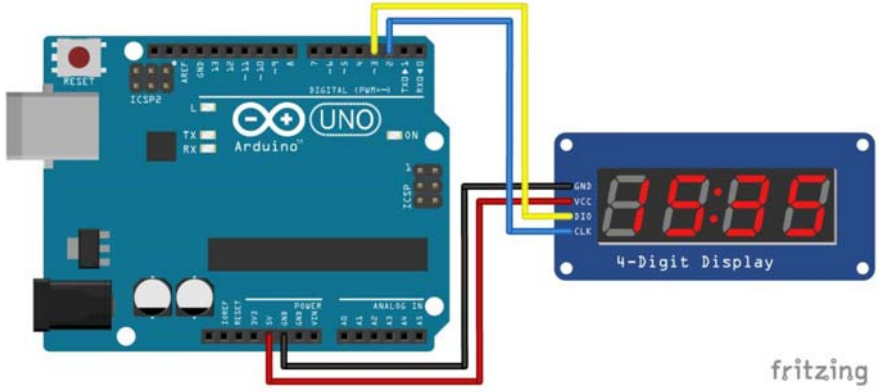
Dette er et numeriske display med fire 7-segment siffer som egner seg godt til små prosjekter da det er billig og lett å programmere via en serie-buss. Displayet er montert på et kretskort med kretsen TM1637 som gjør at det kun trenger 2 tilkoblinger til mikrokontrolleren i tillegg til spenning og jord:

5V – Displayet trenger en spenning på 3,3 – 5V
GND – Jordforbindelse (–)
DIO – Serielle data inn
CLK – Klokkesignal

Displayet kan leveres i ulike farger, hvit, blå, grønn, rød og gul. Det har et strømtrekk på typisk 80 mA ved 5V og lysstyrken kan settes til 8 nivåer.

Displayet leveres både med desimalpunkt for hvert siffer og med kolon mellom andre og tredje siffer for bruk som klokkedisplay.

Figuren under viser hvordan displayet kan kobles opp mot Arduino UNO[36].

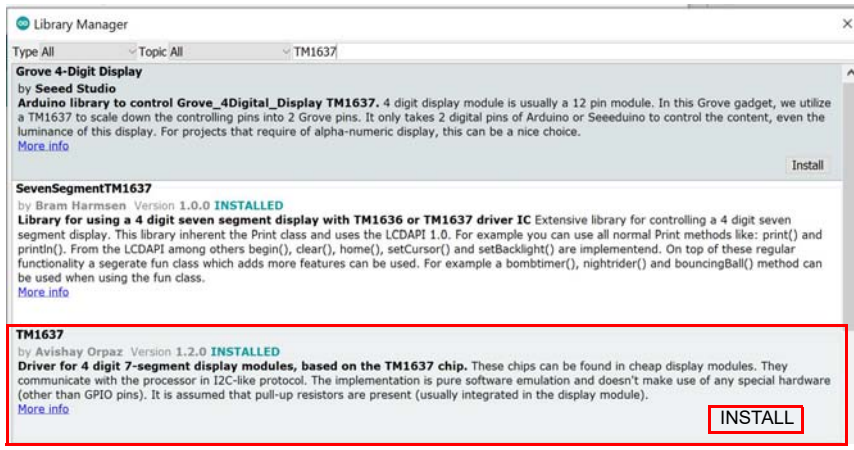


7.4.2 Programmering

1. For å bruke displayet TM1637 så må vi installere biblioteket som hører til dette displayet. Velg *Sketch* menylinjen. Velg så *Include library –> Library manage*. Skriv inn: TM1637 i innboksen.

36.Figuren er hentet fra https://www.makerguides.com/tm1637-arduino-tutorial/

2. Etter hvert vil det dukke opp flere alternative biblioteker for TM1637. Flere kan sikkert velges, men den innrammede varianten (Avishay Orpaz) synes å fungere godt også med ESP32[37].



Etter kort tid er biblioteket installert og klart til å brukes.

3. **Inkluder biblioteket**
Biblioteket inkluderes med følgende kommando i starten av programmet:

```
#include <TM1637Display.h>
```

Denne kommandoen gir tilgang til bibliotekets mange funksjoner

4. **Deklarer instanser**
Deklarasjonen gir vårt display navnet *display* av typen *TM1637Display*, samtidig så forteller deklarasjonen hvilke pinner klokke (CLK) og datainngangen (DIO) skal kobles til (2, 3):

```
#define CLK 2
#define DIO 3
TM1637Display display = TM1637Display(CLK, DIO);
```

Deklarasjonen gjøres tidlig i programmet før setup()-funksjonen.

Dersom man ønsker å koble opp flere displayer så kan disse tilknyttes andre porter på mikrokontrolleren. Eksempelet under viser hvordan vi kan bruke tre displayer med ulike navn og forskjellig tilkoblingspunkter for klokke- og data-inngang:

```
TM1637Display display_1  = TM1637Display(2, 3);
TM1637Display display_2  = TM1637Display(4, 5);
TM1637Display display_3  = TM1637Display(6, 7);
```

5. **Tøm displayet**
Dersom vi ønsker å slå av samtlige segmenter slik at displayet går i "svart" kan vi bruke kommandoen:

37. Beskriver kommandoen i biblioteket grundig: https://github.com/avishorp/TM1637/blob/master/TM1637Display.h

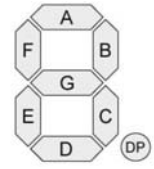
```
display.clear();
```
Hvor *display* er navnet vi har gitt displayet. Denne kan brukes når som helst, men det kan jo være greit og "tømme" displayet i setup()-funksjonen før bruk.

6. **Sett lysstyrke**
   Biblioteket gir mulighet til å sette lysstyrken til displayene i 8 trinn fra 0 (slukket) til 7 (maksimal lysstyrke). I eksempelet under er lysstyrken satt til maks.:
   ```
   display.setBrightness(7);
   ```
   Kommandoen kan brukes hvor som helst, men det er ikke unormalt å sette lysstyrken i settup()-funksjonen. Den fullstendige funksjonen kan skrives slik:
   ```
   void display.setBrightness(uint8_t brightness, bool on = true);
   ```
   Det vil si at parameter nr. to kan slå på (`true`) eller av displayet (`false`).

7. **Skriv tall**
   Vi kan skrive heltall, f.eks. innholdet i variabelen *i*, til displayet med følgende kommando:
   ```
   display.showNumberDec(i);
   ```
   Denne vil kunne skrive tall, positive tall opp til 9999, og negative tall opp til –999. Kommandoen vil vise fortegnet for negative tall på displayet.

   Kommandoen har flere argumenter og kan fult utskrevet uttrykkes som:
   ```
   display.showNumberDec(i, true, 4, 0);
   ```
   Hvor *i* er tallet som ønskes vist, *true* angir at ledende 0'er skal vises, *false* at de ikke skal vises. 4 tallet angir antall siffer i displayet, i dette tilfellet 4, og tilslutt angir *0* at mest signifikante siffer er lengst til venstre, *3* betyr at mest signifikante siffer er lengst til høyre.

   Det finnes også en utvidet kommando som gir mulighet til å sette desimalpunktet der man ønsker:
   ```
   display.showNumberDecEx(i, 0b00100000, true, 4, 0);
   ```
   Hvor *i* er tallet som ønskes vist, *0b0010000* angir desimalpunkt etter 3. siffer fra venstre, *true* angir ledende *0* skal vises, *false* at de ikke skal vises og *0* at mest signifikante siffer er lengst til venstre, *3* betyr at mest signifikante siffer er lengst til høyre,

8. **Skriv til segmenter**
   

   Man kan også skrive til de enkelte segmentene til hvert av sifrene hos displayet. En måte å gjøre dette på er å definere et array av fire 8-bits elementer (byte), hvor hvert element henviser til hvert av de fire sifrene. I eksempelet under er arrayet kalt *data*:
   ```
   const uint8_t data[] = {0xFF, 0xFF, 0xFF, 0xFF};
   ```
   Arrayet er i denne sammenhengen definert som en konstant bestående av fire byte uten fortegn `uint8_t`. Hver bit i hver byte representerer ett av de 8 segmentene inkludert desimalpunktet i hvert av de fire sifrene. Konstanten `data[]` vil slå på alle segmentene (0xff). Tilsvarende vil følgende eksempel:
   ```
   const uint8_t data[] = {0x00, 0x00, 0x00, 0x00};
   ```
   slå av segmentene og tilsvare kommandoen `display.clear();`
   For å sende dette arrayet til displayet brukes kommandoen:
   ```
   display.setSegments(data);
   ```

For lettere å vite hvilket segment man skriver til inneholder biblioteket mulighet til å slå på de ulike segmentene ved å bruke konstantene: SEG_A, SEG_B, SEG_C, SEG_D, SEG_E, SEG_F, SEG_G. Dette egner seg godt dersom man f.eks. vil skrive en tekst på displayet:

```
const uint8_t done[] = {
SEG_B | SEG_C | SEG_D | SEG_E | SEG_G,          // d
SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F,  // O
SEG_C | SEG_E | SEG_G,                          // n
SEG_A | SEG_D | SEG_E | SEG_F | SEG_G           // E
};
```

Her defineres ordet `dOnE`.
Dette skrives da til displayet med følgende kommando:

```
display.setSegments(done);
```

8 Referanser

[1] Mer informasjon om Sparkfun Invention's kit:
https://www.sparkfun.com/products/11227

[2] Mer informasjon om I^2C-bussen:
http://www.i2c-bus.org/

[3] Mer informasjon om Serikommunikasjon på SPI-bussen:
http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

[4] Koblingsskjema for Arduino UNO:
http://arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf.

[5] For mer informasjon om Arduino UNO R3 layout:
http://arduino.cc/en/Main/ArduinoBoardUno/

[6] For nedlasting av programeditoren IDE for Arduino:
http://arduino.cc/hu/Main/Software

[7] For nedlasting av Referansemanualen for Arduino C++
http://arduino.cc/en/Reference/HomePage

[8] Skolelaboratoriets blå hefteserie:
https://www.ntnu.no/skolelab/bla-hefteserie

[9] Elliot Williams, *Make: AVR Programming – Learing to Write Software for Hardware*, Make Community, LLC; 1 edition (February 25, 2014)

Vedlegg A Komponentliste

Tabellen under inneholder noen av komponentene som er brukt og hvor de kan skaffes fra:

Typebetegnelse	Type komponent	Leverandør	Nettadresse
Mini	Koblingsbrett	Banggood	https://www.banggood.com/Mini-Solderless-Prototype-Breadboard-170-Points-For-Arduino-Shield-p-74814.html
TM1637	Display 4x7 seg.	Banggood	https://www.banggood.com/0_36-Inch-4-Digit-LED-Display-Tube-7-segments-TM1637-30x14mm-Yellow-Decimal-Point-Module-p-1654852.html
Jumpere	Diverse	Banggood	https://www.banggood.com/40pcs-20cm-Male-To-Female-Jumper-Cable-Dupont-Wire-For-Arduino-p-973822.html?rmmds=search&cur_warehouse=CN
ESP WROOM 32 30 pin m/hylselist	Mikrokontroller	Kult og Billig	http://www.kultogbillig.no/ESP32-Development-Board-WiFi-Bluetooth-Ultra-Low-Power-Consumption-Dual-Cores-ESP-32-ESP-32S-Board?keyword=ESP32
ESP WROOM 32 30 pin m/hylselist	Mikrokontroller	Banggood	https://www.banggood.com/Geekcreit-ESP32-WiFi-bluetooth-Development-Board-Ultra-Low-Power-Consumption-Dual-Cores-Unsoldered-p-1214159.html
USB-kabel	USB A —> micro	Banggood	https://www.banggood.com/BlitzWolf-BW-MC12-Micro-USB-Charging-Data-Cable-1ft0_3m-For-Samsung-S7-S6-Xiaomi-Redmi-Note-5-p-1339804.html
BME280	Lufttrykk, luftfuktighet og temperatursensor	Banggood	https://www.banggood.com/CJMCU-280E-BME280-High-Precision-Atmospheric-Pressure-Sensor-Module-Board-p-1103115.html

Vedlegg B Ressursprogrammer

Dette vedlegget inneholder noen nyttige programmer som vi kan ha bruk for når vi setter opp prosjektet vårt.

B.1 Scan I²C-bussen etter adresser til påkoblede elementer

Programmet scanner I²C-bussen og lister opp adressene til alle som er koblet på bussen. Programmet er nyttig for å finne I²C-adressen til en sensor eller aktuator.

```
// ESP32 I2C Scanner
// Based on code of Nick Gammon  http://www.gammon.com.au/forum/?id=10896
// ESP32 DevKit - Arduino IDE 1.8.5
// Device tested PCF8574 - Use pullup resistors 3K3 ohms !
// PCF8574 Default Freq 100 KHz

#include <Wire.h>

void setup()
{
  Serial.begin (115200);
  Wire.begin (21, 22);   // sda= GPIO_21 /scl= GPIO_22
}

void Scanner ()
{
  Serial.println ();
  Serial.println ("I2C scanner. Scanning ...");
  byte count = 0;

  Wire.begin();
  for (byte i = 8; i < 120; i++)
  {
   Wire.beginTransmission (i);         // Begin I2C transmission Address
(i)
    if (Wire.endTransmission () == 0)  // Receive 0 = success (ACK
response)
    {
      Serial.print ("Found address: ");
```

```
      Serial.print (i, DEC);
      Serial.print (" (0x");
      Serial.print (i, HEX);     // PCF8574 7 bit address
      Serial.println (")");
      count++;
    }
  }
  Serial.print ("Found ");
  Serial.print (count, DEC);        // numbers of devices
  Serial.println (" device(s).");
}

void loop()
{
  Scanner ();
  delay (100);
}
```

B.2 Finn kontrollerens MAC-adresse

Programmet lastes opp på en ESP og vil returnere komponentens MAC-adresse.

```
// Complete Instructions to Get and Change ESP MAC Address: https://
RandomNerdTutorials.com/get-change-esp32-esp8266-mac-address-arduino/

#include "WiFi.h"
#include <esp_now.h>

void setup(){
  Serial.begin(115200);
  WiFi.mode(WIFI_MODE_STA);
  Serial.println(WiFi.macAddress());
}

void loop(){

}
```

Figuren under viser resultatet av kjøring av programmet på ESP32.

```
COM18

ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1044
load:0x40078000,len:8896
load:0x40080400,len:5816
entry 0x400806ac
24:6F:28:9E:9F:70
```

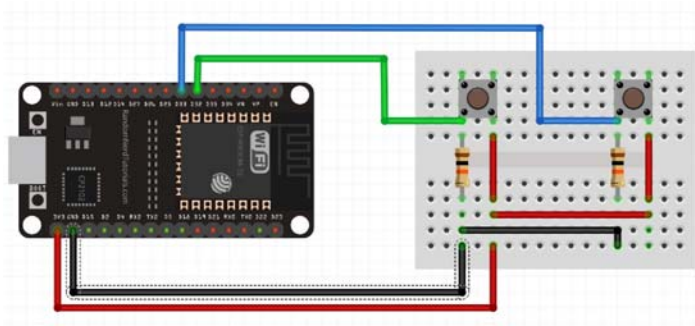
MAC-adressen til denne komponenten er: 24:6F:28:9E:9F:70 Adressen er gitt i heksdesimale siffer.

B.3 Eksempelkode: Deep sleep med ekstern oppvekking (ext1)

Denne koden er skrevet for å demonstrere at man kan vekke en ESP32-kontroller ved trykke på to knapper.

Oppkobling

Oppkoblingen kan være som vist i figuren under.



Program

```
/*
Author: Kåre-Benjamin Hammervold Rørvik 9.12.2020
On behald of NTNU.
Edited 10.12.2020

Et oppsett med to knapper koblet til PIN 33 og PIN 32.
Knappene gir enten en logisk LAV (GND) eller HØY (3.3V).
```

```
Knappene kan brukes til å vekke ESP fra dvale (deep sleep).
ESP32 vil ligge i dvale frem til en knapp blir trykket.
Når ESP er våken spiller den av et lysshow før den går til dvale igjen.
I dvalet bruker den vesentlig mindre energi (low-power operation).
Sketchen er testet med variantene: ESP32 DEV MODULE og DOIT ESP-32 DEVKIT V1.
For varianten ESP32 DEV MODULE kan ikke LED_BUILTIN brukes (den har ikke innebygd
programerbart LED).
Dersom ESP32 DEV MODULE skal brukes, kan den benytte et eksternt led som kobles
til en PIN av eget valg (husk en motstand i serie).

Pinout:

Knapp1          |     ESP-32
---------------------------------
Høyre side      |     PIN 32
Høyre side      |     GND (via en resistor)
Venstre side    |     3V3 (3.3V)

Knapp2          |     ESP-32
---------------------------------
Høyre side      |     PIN 33
Høyre side      |     GND (via en resistor)
Venstre side    |     3V3 (3.3V)
*/

//Bitmaske for å ta i bruk PIN 32 og PIN 33.
#define BUTTON_PIN_BITMASK 0x300000000

//Variabler som styrer egenskapene til aktiviteten på indikatorlampen
(LED_BUILTIN).
int loopLength = 15;
int baseDuration = 30;

void setup(){
  Serial.begin(115200);
  //Bruker bitmasken for PIN 32 og PIN 33 til å indikere "wake up condition"
  //Altså av hvem skal den våkne fra (BUTTON_PIN_BITMASK)
  //Definerer at ESP32 skal våkne av en ekstern stimuli (kilder)
  //Altså den skal våkne kun av logisk høye inngangssignaler
(ESP_EXT1_WAKEUP_ANY_HIGH).
  esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK,ESP_EXT1_WAKEUP_ANY_HIGH);
  pinMode(LED_BUILTIN, OUTPUT);
```

```
}

void loop(){
  Serial.print("Jeg er våken! \n");
  Serial.print("Blinker lys \n");

  //ESP blinker med led for å indikere at den ikke ligger i dvale og er klar til
bruk.
  //Lyset blinker med variabele mellomrom (frekvens).
  //Mellomrommet mellom blinkene øker med tiden, altså frekvensen synker.
  for (int i = 0; i < loopLength; i++){
    digitalWrite(LED_BUILTIN, HIGH);
    delay(i*baseDuration);
    digitalWrite(LED_BUILTIN, LOW);
    delay(i*baseDuration);
  }

  //Lyset blinker med variabele mellomrom (frekvens).
  //Mellomrommet mellom blinkene synker med tiden, altså frekvensen øker.
  for (int i = loopLength; i > 0; i--){
    digitalWrite(LED_BUILTIN, HIGH);
    delay(i*baseDuration);
    digitalWrite(LED_BUILTIN, LOW);
    delay(i*baseDuration);
  }

  //Alt av aktivit er over og ESP32 går til dvale igjen.
  Serial.print("Jeg er ferdig og går til dvale! \n");
  esp_deep_sleep_start();
}
```

B.4 Eksempelprogram for ESP-NOW

Disse programmene forutsetter følgende:

- At man på senderdelen (sensornoden) har koblet opp en ESP32 og BME280
- At man på mottakerdelen har koblet opp en ESP32 og displayet

Dessuten:

- At man har funnet MAC-adressen til mottakerdelen
- At man har all informasjon om tilkobling til Circus of Things og konsoller for visning av temperatur, luftfuktighet og lufttrykk.
- Ev. at man intil videre sløyfer overføring av data til CoT

B.4.1 Sensornode (Node-1, Senderdelen)

Under er listet programmet for sensornoden:

```
/*
Author: Kåre-Benjamin Hammervold Rørvik og Nils Kristian Rossing 31.12.2020
On behald of NTNU.
Edited 29.12.2020
Et oppsett med BME 280 sensor koblet via I2C til ESP32.

En ESP32 som står i Station Mode og fungerer som sensor node.
Sender datapakker over ESP-NOW.
Sender og mottar data over Wi-Fi opp mot CoT.
Bytter mellom Wi-Fi og ESP-NOW.
Antall meldinger sendt vises på displayet, temperatur fuktighet og trykk i
rotering.

Sketchen er testet med variantene: ESP32 DEV MODULE og DOIT ESP-32 DEVKIT V1
Sensordata skrives ut i konsollen (serial monitor).

BME280       |     ESP-32
----------------------------------
SCL/SCK:     |     D22
SDA/DATA:    |     D21
VDD:         |     3V3 (3.3V)
GND:         |     GND

Det er anbefalt å koble opp en kondensator (avkoppling) mellom GND og VDD på sen-
sorer. F.eks.: 0.1uF keramisk kondensator.

// Sources:
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/
network/esp_now.html
https://randomnerdtutorials.com/esp-now-esp8266-nodemcu-arduino-ide/
?fbclid=IwAR0pH-ilF45ceHlyqjB1Im5oDkOJUtxJ7g1TDABbaSOkiBep5xYlsH1y5_M
*/

#include <esp_now.h>
#include <WiFi.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

// Adressen til mottakeren ved ESP-NOW kommunikasjon, dette er MAC addressen skre-
vet på hex format.
```

```
uint8_t broadcastAddress[] = {0x24, 0x6F, 0x28, 0x9E, 0x9F, 0x70};

Adafruit_BME280 bme280;
unsigned long delayTime;

// Teller antall meldinger som er sendt, starter på 0.
int consecutive_messages = 0;

// Teller antall målinger som er gjort.
int consecutive_meassures = 0;

// Transmitted data
typedef struct {
    int instance;
    float temperature;
    float humidity;
    float pressure;
    String source;
} ESPNOWData;


// Oppretter et packet objekt for strukturen ESPNOWData.
ESPNOWData packet;

void setup() {
  Serial.begin(115200);
  initESPNOW();
  checkBME280();
}

void loop() {
  // Høster inn data til packet.
  preparePacket();
  // Skriver ut temperatur, fuktighet og trykk fra packet til monitor og sjekker
om verdiene er gyldige
  consecutive_meassures = sensorValuesToMonitor(consecutive_meassures);
  // Sender data i packet.
  transmitData();
  delay(2000);
}

void initESPNOW(){
```

```
  // Init ESP-NOW med en promise structure.
  // ESP-32 må være konfigurert i Station Mode for at ESP-NOW skal kunne benyttes.
  WiFi.mode(WIFI_STA);

  // esp_now_init() vil returnere enten ESP_OK eller ESP_ERR_ESPNOW_INTERNAL
  // Disse antyder om ESP-NOW er konfigurert korrrekt eller ikke.
  // ESP_OK betyr at alt er riktig konfigurert, ESP_ERR_ESPNOW_INTERNAL er en
feilmelding.
  // Den kan sees på som aktiv lav: ESP_OK = 0 = LOW, ESP_ERR_ESPNOW_INTERNAL = 1
= HIGH.
  int initStatus = esp_now_init();

  // Funksjoner som aktiveres og returnerer en status:
  Serial.println("---------------ESP-NOW-----------------");
  if (initStatus == ESP_OK){

    // Setter opp en callback for å hente ut informasjon om sendingen.
    esp_now_register_send_cb(activateOnCallback);

    // Registrerer peer
    esp_now_peer_info_t peerInfo;
    memcpy(peerInfo.peer_addr, broadcastAddress, 6);
    peerInfo.channel = 0;
    peerInfo.encrypt = false;

    Serial.println("ESP-NOW er konfigurert riktig");
    if (esp_now_add_peer(&peerInfo) == ESP_OK){
      Serial.println("... og Peer er lagt til");
    }
    else {
     Serial.println("... Noe gikk galt etter konfigureringen, Peer kunne ikke bli
lagt til?");
      Serial.println("Programmet restartes etter ti sekunder");
      delay(10000);
      ESP.restart();
    }
    Serial.println("ESP-NOW er klar til bruk");
  }
  else {
   Serial.println("ESP-NOW er ikke konfigurert riktig, programmet restartes etter
ti sekunder");
    delay(10000);
```

```
    ESP.restart();
  }
  Serial.println("----------------------------------------");
}

void checkBME280() {
  // En TRUE/FALSE variabel (bool) som sjekker om sensoren er detektert og oppfører
seg som forventet.
  // bme280.begin tar inn addressen på I2C buss (0x76) og sjekker på denne
addressen.
  bool readyCheck = bme280.begin(0x76);

  //Sjekker om sensoren er klar, hvis ikke henger systemet i while og sjekker om
sensoren blir detektert.
  if (!readyCheck) {
    Serial.println("BME 280 er ikke funnet, se over skjematikken og koden");
    Serial.println("Sjekk at: SCL/SCK er koblet til D22 og SDA/DATA på D21");
    Serial.println("Programmet resetartes om sensoren blir riktig detektert");
    while (1){
      // Står i loop...
      // Men om sensoren detekteres restartes programmet.
      readyCheck = bme280.begin(0x76);
     // Et delay kan ofte være greit for å unnga timing problemer, og unødvendig
prossesering.
      // Ulike størrelser kan testes.
      delay(1000);
      if(readyCheck){
        ESP.restart();
        }
    }
  }
  Serial.println("BME 280 er klar og fungerer som forventet");
}

void transmitData(){
  esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &packet,
sizeof(packet));

  if (result == ESP_OK) {
    Serial.println("Alt er klart, og data sendes...");
    consecutive_messages += 1;
  }
```

```
  else {
    Serial.println("Data kan ikke sendes fordi noe er feilkonfigurert...");
  }
}

void preparePacket(){
  packet.pressure = bme280.readPressure();
  packet.instance = consecutive_messages;
  packet.temperature = bme280.readTemperature();
  packet.source = "D4";
  packet.humidity = bme280.readHumidity();
}

// callback when data is sent
void activateOnCallback(const uint8_t *mac_addr, esp_now_send_status_t status) {
  if (status == ESP_NOW_SEND_SUCCESS){
    Serial.println("Data har blitt sendt og er mottatt av mottakeren!");
  }
  else {
    Serial.println("Noe gikk galt, sendingen var ikke vellykket");
  }
}

int sensorValuesToMonitor(int successfullMeassures) {
  //Skriver ut temperatur, fuktighet og trykk verdiene som ligger i packet objektet
  //Sjekker om tall er i gydlighetsområdet, hvis ikke kjøres en restart.

  successfullMeassures ++;

  Serial.println("----------------------");
  Serial.print("Antall målinger på rad: ");
  Serial.println(successfullMeassures);


  Serial.print("Temperatur: ");
  Serial.print(packet.temperature);
  Serial.println(" *C");

  Serial.print("Trykk: ");
  Serial.print(packet.pressure/100);
  Serial.println(" hPa");
```

```
  Serial.print("Fuktighet: ");
  Serial.print(packet.humidity);
  Serial.println(" %");

  Serial.println("---------------------");
  Serial.println();

  if(((packet.temperature < -30 || packet.humidity < -30) || packet.temperature
> 100 || packet.humidity > 100)){
    Serial.print("Verdier er utenfor gydlighetsområdet, det antas en error, en
restart påfølger om 5 sekunder");
      delay(5000);
      ESP.restart();
  }
  return successfullMeassures;
}
```

B.4.2 Uploadnoden (Node-2, mottakerdelen)

Under er listet programmet for uploadnoden. Dette er noden som tar imot data oversendt fra sensornoden og sender informasjonen videre til Circus of Things.

```
/*
Author: Kåre-Benjamin Hammervold Rørvik og Nils Kristian Rossing 23.9.2020
On behald of NTNU.
Edited 27.12.2020

En ESP32 som står i Station Mode og fungerer som uplink og gateway.
Mottar datapakker over ESP-NOW.
Sender og mottar data over Wi-Fi opp mot CoT.
Bytter mellom Wi-Fi og ESP-NOW.
Antall meldinger sendt vises på displayet, temperatur og fuktighet i rotering.

Pinout:

7-Segment    |     ESP-32
---------------------------------
CLK          |     D32
DIO          |     D33
VDD:         |     3V3 (3.3V)
GND:         |     GND
```

```
// Sources:
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/
network/esp_now.html
https://randomnerdtutorials.com/esp-now-esp8266-nodemcu-arduino-ide/
?fbclid=IwAR0pH-ilF45ceHlyqjB1Im5oDkOJUtxJ7g1TDABbaSOkiBep5xYlsH1y5_M
*/



#include <esp_now.h>
#include <WiFi.h>
#include <CircusESP32Lib.h>
#include <Wire.h>
#include <TM1637Display.h>

#define displayDividerBitMask 0b01000000

// 7-segment display
const byte PIN_CLK = 32;
const byte PIN_DIO = 33;
TM1637Display display(PIN_CLK, PIN_DIO);

// Lager et array av typen byte (byte = 8bit).
// Dette brukes ofte for å styre 7-segment display.
// Merk: Displayene har enten 7 eller 8 LED segment, som akkurat går opp i en byte.

const byte UP[] = {
  SEG_B | SEG_C | SEG_D | SEG_E | SEG_F, // U
  SEG_A | SEG_B | SEG_E | SEG_F | SEG_G, // P
  0,
  0,
};

/*
Disse deklarasjonene inneholder informasjonen som er nødvendig for å koble opp mot
skyen og ta i bruk de ønskede funksjonalitetene.
Når en bruker char på denne måten opprettes en vektor, som senere kan leses ut (i
motsettning til string som er litt enklere å ta i bruk).
Det er påkrevd å bruke dette formatet for å kunne kommunisere med skytjenesten til
CoT.
*/

char ssid[] = "<Navn på ruter>"; // Legg in navnet på nettverket ditt her. Denne
verdien er Case sensitivt (den skiller mellom små og store bokstaver).
```

```
char password[] =  "<ruter passord>"; // Legg in nettverkspassordet ditt her (denne er også Case sensitiv).
char server[] = "www.circusofthings.com"; // Addressen til tjenesten vi skal kommunisere med [Trenger ikke endres]
char temperature_key[] = "16906";  // Place the Key of the signal you created at Circus Of Things for the Temperature
char humidity_key[] = "15160";  // Place the Key of the signal you created at Circus Of Things for the Humidity
char pressure_key[] = "10333";  // Place the Key of the signal you created at Circus Of Things for the Preassure
char order_key[] = "5957"; // Type the Key of the Circus Signal you want the ESP32 listen to.
char token[] = "<token>"; // Dette er din login token, den finner du på kontoen din (Account).

// Her lages et objekt og vi får tilgang til dets funksjoner.
// Grunnen til at vi tar inn informasjonen om nettet her, er at vi ikke trenger å legge det inn flere ganger andre steder.
// Disse har nå vi gitt objektet tilgang på allerde (og slik er det definert av CoT på forhånd).
CircusESP32Lib circusESP32(server,ssid,password);

// Transmitted data
typedef struct {
    int instance;
    float temperature;
    float humidity;
    float pressure;
    String source;
} ESPNOWData;

// Oppretter et packet objekt for strukturen ESPNOWData.
ESPNOWData packet;

// Denne variabelen brukes i switch/case for å angi hvilken state som blir den neste.
// Vi starter på 0, altså default case.
int nextState = 0;

void setup() {
  Serial.begin(115200);
  initDisplays();
  initESPNOW();
}
```

```
void loop() {
  //uploadTheData(20000);
}

void initDisplays(){
  // Initialises the displays.
  // 7-segment directly from I/O.
  display.setBrightness(7);
  pinMode(LED_BUILTIN, OUTPUT);
}

void initESPNOW() {
  // Init ESP-NOW med en promise structure.
  // ESP-32 må være konfigurert i Station Mode for at ESP-NOW skal kunne benyttes.
  WiFi.mode(WIFI_STA);

  // Funksjoner som aktiveres og returnerer en status:
  // Disses statusene er de overnevnte.

  // esp_now_init() vil returnere enten ESP_OK eller ESP_ERR_ESPNOW_INTERNAL
  // Disse antyder om ESP-NOW er konfigurert korrrekt eller ikke.
  // ESP_OK betyr at alt er riktig konfigurert, ESP_ERR_ESPNOW_INTERNAL er en
feilmelding.
  // Den kan sees på som aktiv lav: ESP_OK = 0 = LOW, ESP_ERR_ESPNOW_INTERNAL = 1
= HIGH.
  int initStatus = esp_now_init();

  // esp_now_register_recv_cb aktiverer callback funksjonen "activateOnCallback"
når ESP-32 får inn en datapakke.
  // Funksjonen returnere også status slik som esp_now_init gjør.
  int callbackStatus = esp_now_register_recv_cb(activateOnCallback);

  // En serie med logikk som brukes for å angi om ESP-NOW er riktig konfigurert
og er klar til å brukes.
  // Dersom noe går galt vil ESP-32 bli restartet.
  Serial.println("---------------ESP-NOW-----------------");
  if (initStatus == ESP_OK){
    Serial.println("ESP-NOW er konfigurert riktig");
    if (callbackStatus == ESP_OK){
      Serial.println("... og Callback funksjonen blir kalt");
    }
```

```
    else {
      Serial.println("... Noe gikk galt etter konfigureringen, kanskje callback
deklareringen?");
      Serial.println("Programmet restartes etter ti sekunder");
      delay(10000);
      ESP.restart();
    }
    Serial.println("ESP-NOW er klar til bruk");
  }
  else {
   Serial.println("ESP-NOW er ikke konfigurert riktig, programmet restartes etter
ti sekunder");
    delay(10000);
    ESP.restart();
  }
  Serial.println("----------------------------------------");
}

void writeToDisplays(float dataToPrint, bool decimalNumber = false){
  // Skriver data fra dataToPrint variabelen til 7-segment display
  // Det alternative argumentet decimalNumber styrer hvordan displayet viser data.
  // Ved false så tiplasses displayet 2 siffer før og etter kommaet.(ved hjelp av
calcualteIntcustomForm funksjonen)
  // Ellers skriver displayet som vanlig.
  if(decimalNumber){

display.showNumberDecEx(calculateIntCustomForm(dataToPrint,2),displayDividerBit-
Mask);
    // DisplayDividerBitMask brukes for å angi at displayet skal vise en ":" på
midten av displayet.
    // Dette brukes vi for å skille mellom heltall og desimal-delen av tallet.
  }
  else {
    display.showNumberDec(dataToPrint);
    // Skriver ut tallet på vanlig vis (uten skillet på midten).
  }
}

int calculateIntCustomForm(float floatingInput, int numberOfDecimals) {
// Denne funksjonen tar inn en float (flyttall) med n (numberOfDecimals) siffer
presisjon etter kommaet.
// Funksjonen returnerer heltallet og desimaldelen som et sammenslått intNumber.
// Ved flere desimaler kan avrundingsfeil bli større.
```

```
// Dersom H er et heltall og D er desimaldelen vil det ta formen: HHDD ved 2 siffer før og etter kommaet.
// F.eks.: vil 22,54 se ut som 2254 altså uten komma.
// Funksjonen ville da være egnet for input (floatingInput) i området -9,99 til 99,99.

  int offset = pow(10,numberOfDecimals);

  // Dersom verdien er større enn 0 må den justeres for avrundingsfeil (konverteringer rundes opp)
  if (floatingInput == 0){
    int intNumber = 0;
    return intNumber;
  }
  // Dersom verdien ikke er null justerer vi den direkte.
  else {
    int intNumber = floatingInput * offset;
    return intNumber;
  }
}

void uploadTheData(int delayLength){
  // Laster opp data til CoT.
  // Tar i mot en delayLength som bestemmer hvor ofte data skal lastes opp til skyen.
  digitalWrite(LED_BUILTIN, HIGH);
  display.setSegments(UP);
  circusESP32.begin(); // Let the Circus object set up itself for an SSL/Secure connection
  circusESP32.write(temperature_key,packet.temperature,token); // Report the temperature measured to Circus.
  circusESP32.write(humidity_key,packet.humidity,token); // Report the humidity measured to Circus.
  circusESP32.write(pressure_key,packet.pressure,token); // Report the pressure measured to Circus.
  digitalWrite(LED_BUILTIN, LOW);
  delay(10);
  WiFi.disconnect(true);
  WiFi.mode(WIFI_STA);
  delay(delayLength);
}
```

```
void activateOnCallback(const uint8_t * mac, const uint8_t *incomingData, int len)
{
  // Tar i mot incomingData og lagrer en kopi til packet objektet.
  // Receives the incoming data and writes to terminal.
  // len og mac argumentene brukes ikke direkte i vår funksjon, men må være med i
argument listen da denne formen angitt av Espressif (ESP-NOW).


  // Memcpy koppierer verdiene i incomingData til minneblokken som vi lagrer packet
i.
  // Data kopieres fra incomingData objektet til packet objektet.
  // Sizeof operasjonen angir antall bytes som utgjør packet,
  // ... og denne informasjonen brukes i memcpy for å bestimme antallet bytes som
skal kopieres.
  memcpy(&packet, incomingData, sizeof(packet));
  Serial.print("Data packet nummer: ");
  Serial.print(packet.instance);
  Serial.print(" fra: ");
  Serial.println(packet.source);
  Serial.print("Temperaturen er: ");
  Serial.print(packet.temperature);
  Serial.println("C");
  Serial.print("Luftfuktigheten er : ");
  Serial.print(packet.humidity);
  Serial.println("%");
  Serial.print("Pressure is : ");
  Serial.print(packet.pressure/100);
  Serial.println("hPa");
  Serial.println();
  switch (nextState) {
  case 1:
    writeToDisplays(packet.humidity, true);
    nextState++;
    break;
  case 2:
    writeToDisplays(packet.temperature, true);
    nextState++;
    break;
  case 3:
    writeToDisplays(packet.pressure/100);
    // deler på 100 og får dermed enheten hPa (hecto Pascal)
    // Displayet viser bedre verdier som ikke blir for store.
    nextState = 0;
```

```
    break;
  default: //printAmounOfTransfers
    writeToDisplays(packet.instance);
    nextState++;
    break;
  }
}
```

Vedlegg C Programmer – Løsningsforslag

Kun for oppdrag 12 og 13 i denne utgaven.

C.1 Oppdrag 12 – ESP-NOW – Trådløs overføring av data mellom to ESP32

Løsningsforslag for kommunikasjon mellom to ESP32 ved hjelp av ESP-NOW:

C.1.1 Program for sensornoden

Listingen under viser en strippet versjon av programmet for sensornoden

```
/*
Author: Kåre-Benjamin Hammervold Rørvik og Nils Kristian Rossing 01.01.2021
On behald of NTNU.
Edited: 01.01.2021
Et oppsett med BME 280 sensor koblet via I2C til ESP32.

En ESP32 som står i Station Mode og fungerer som sensor node.
Sender datapakker over ESP-NOW.
Sender og mottar data over Wi-Fi opp mot CoT.
Bytter mellom Wi-Fi og ESP-NOW.
Antall meldinger sendt vises på displayet, temperatur fuktighet og trykk i
rotering.

Sketchen er testet med variantene: ESP32 DEV MODULE og DOIT ESP-32 DEVKIT V1
Sensordata skrives ut i konsollen (serial monitor).

BME280       |     ESP-32
-----------------------------------
SCL/SCK:     |     D22
SDA/DATA:    |     D21
VDD:         |     3V3 (3.3V)
GND:         |     GND

Det er anbefalt å koble opp en kondensator (avkoppling) mellom GND og VDD på sen-
sorer. F.eks.: 0.1uF keramisk kondensator.

// Sources:
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/
network/esp_now.html
https://randomnerdtutorials.com/esp-now-esp8266-nodemcu-arduino-ide/
?fbclid=IwAR0pH-ilF45ceHlyqjB1Im5oDkOJUtxJ7g1TDABbaSOkiBep5xYlsH1y5_M
*/
```

```
#include <esp_now.h>
#include <WiFi.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

// Adressen til mottakeren ved ESP-NOW kommunikasjon, dette er MAC addressen skre-
vet på hex format.
uint8_t broadcastAddress[] = {0x24, 0x6F, 0x28, 0x9E, 0x9F, 0x70};

Adafruit_BME280 bme280;
unsigned long delayTime;

// Teller antall meldinger som er sendt, starter på 0.
int consecutive_messages = 0;

// Transmitted data
typedef struct {
    int instance;
    float temperature;
    float humidity;
    float pressure;
    String source;
} ESPNOWData;

// Oppretter et packet objekt for strukturen ESPNOWData.
ESPNOWData packet;

void setup() {
  Serial.begin(115200);
  initESPNOW();
  bme280.begin(0x76); // Initier BME28.0 og legg inn I2C-adressen
}

void loop() {

  // Tell opp antall målinger
  consecutive_messages++;

  // Samler inn data fra sensorene
  packet.pressure = bme280.readPressure();
  packet.instance = consecutive_messages;
```

```
  packet.temperature = bme280.readTemperature();
  packet.source = "5C";
  packet.humidity = bme280.readHumidity();

  // Sender data i packet.
  esp_now_send(broadcastAddress, (uint8_t *) &packet, sizeof(packet));
  delay(2000);
}

void initESPNOW(){

  // ESP-32 må være konfigurert i Station Mode for at ESP-NOW skal kunne benyttes.
  WiFi.mode(WIFI_STA);

  // Initier ESP-NOW
  esp_now_init();

  // Lag en struktur med navnet peerInfo av typen esp_now_peer_info_t som inne-
holder informasjon for å opprette kommunikasjon
  esp_now_peer_info_t peerInfo;

  // Legg inn data i strukturen peerInfo
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Opprett kommunikasjonen mellom ESP-NOW enhetene
  esp_now_add_peer(&peerInfo);
}

// callback when data is sent
void activateOnCallback(const uint8_t *mac_addr, esp_now_send_status_t status) {
}
```

C.1.2 Program for uploadnoden

Listingen under viser en strippet versjon av programmet for uploadnoden

```
/*
Author: Kåre-Benjamin Hammervold Rørvik og Nils Kristian Rossing 23.9.2020
On behald of NTNU.
Edited 27.12.2020
```

```
En ESP32 som står i Station Mode og fungerer som uplink og gateway.
Mottar datapakker over ESP-NOW.
Sender og mottar data over Wi-Fi opp mot CoT.
Bytter mellom Wi-Fi og ESP-NOW.
Antall meldinger sendt vises på displayet, temperatur og fuktighet i rotering.

Pinout:

7-Segment    |     ESP-32
-----------------------------------
CLK          |     D32
DIO          |     D33
VDD:         |     3V3 (3.3V)
GND:         |     GND

// Sources:
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/
network/esp_now.html
https://randomnerdtutorials.com/esp-now-esp8266-nodemcu-arduino-ide/
?fbclid=IwAR0pH-ilF45ceHlyqjB1Im5oDkOJUtxJ7g1TDABbaSOkiBep5xYlsH1y5_M
*/

#include <esp_now.h>
#include <WiFi.h>
#include <Wire.h>
#include <TM1637Display.h>

// 7-segment display
const byte PIN_CLK = 32;
const byte PIN_DIO = 33;
TM1637Display display(PIN_CLK, PIN_DIO);

// Transmitted data
typedef struct {
    int instance;
    float temperature;
    float humidity;
    float pressure;
    String source;
} ESPNOWData;

// Oppretter et packet objekt for strukturen ESPNOWData.
```

```
ESPNOWData packet;

void setup() {
  Serial.begin(115200);
  // Initier ESP-NOW
  initESPNOW();
  // Sett lysstyrke på display
  display.setBrightness(7);

  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {} // void loop er tom

void initESPNOW() {

  // ESP-32 må være konfigurert i Station Mode for at ESP-NOW skal kunne benyttes.
  WiFi.mode(WIFI_STA);

  // Initier ESP-NOW
  esp_now_init();

  // esp_now_register_recv_cb aktiverer callback funksjonen "activateOnCallback"
når ESP-32 får inn en datapakke.
  esp_now_register_recv_cb(activateOnCallback);
}

void showAtDisplay (float showNumber){

  // Vis showNumber på display. showNumber er et desimaltall.
  // Vårt display kan bare vise heltall, men vi kan sette komma fast.
  // Vi har valgt å sette komma fast etter andre siffer (f.eks. 23.00)
  // Ved å multiplisere showNumber med 100,
  // konvertere tallet til et heltall får vi vist den med to desimaler

  showNumber = showNumber*100;
  display.clear();                      // Tøm displayet for data
 display.showNumberDecEx(int(showNumber), 0b01000000, false, 4, 0);   // Vis tall
  delay(500);                           // Vent i 500 millisek. (0,5 sek.)
}
```

```
void activateOnCallback(const uint8_t * mac, const uint8_t *incomingData, int len)
{
  // Memcpy kopierer verdiene i incomingData til packet objektet.
  // Sizeof operasjonen angir antall bytes som utgjør packet,
  // og denne informasjonen brukes i memcpy for å bestemme antallet bytes som skal
kopieres.
  memcpy(&packet, incomingData, sizeof(packet));

  // Dataene skrives ut til monitoren
  Serial.print("Data packet nummer: ");
  Serial.print(packet.instance);
  Serial.print(" fra: ");
  Serial.println(packet.source);
  Serial.print("Temperaturen er: ");
  Serial.print(packet.temperature);
  Serial.println("C");
  Serial.print("Luftfuktigheten er : ");
  Serial.print(packet.humidity);
  Serial.println("%");
  Serial.print("Pressure is : ");
  Serial.print(packet.pressure/100);
  Serial.println("hPa");
  Serial.println();

  // Vis temperatur på 7-segment displayet
  showAtDisplay(packet.temperature);
}
```

C.2 Oppdrag 13 – ESP-NOW for videre overføring til CoT

Løsningsforslag for kommunikasjon mellom to ESP32 ved hjelp av ESP-NOW i tillegg til overføring av måledata til CoT:

C.2.1 Program for sensornoden

Programmet er uforandret fra Oppdrag 12, se vedlegg C.1.1, side 162.

C.2.2 Program for uploadnoden

Først nå kan vi egentlig kalle dette programmet for upload, siden det er først nå at vi laster opp dataene til Circus of Things.

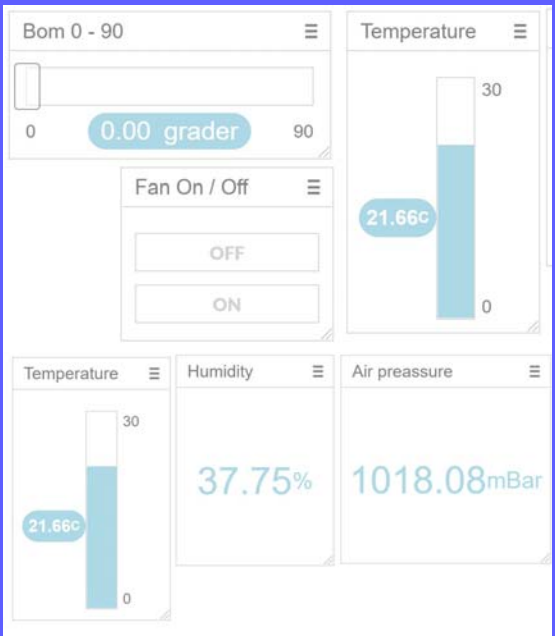
```
*
Author: Kåre-Benjamin Hammervold Rørvik og Nils Kristian Rossing 23.9.2020
On behald of NTNU.
```

```
Edited 27.12.2020

En ESP32 som står i Station Mode og fungerer som uplink og gateway.
Mottar datapakker over ESP-NOW.
Sender og mottar data over Wi-Fi opp mot CoT.
Bytter mellom Wi-Fi og ESP-NOW.
Antall meldinger sendt vises på displayet, temperatur og fuktighet i rotering.

Pinout:

7-Segment    |     ESP-32
-----------------------------------
CLK          |     D32
DIO          |     D33
VDD:         |     3V3 (3.3V)
GND:         |     GND

// Sources:
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/
network/esp_now.html
https://randomnerdtutorials.com/esp-now-esp8266-nodemcu-arduino-ide/
?fbclid=IwAR0pH-ilF45ceHlyqjB1Im5oDkOJUtxJ7g1TDABbaSOkiBep5xYlsH1y5_M
*/

#include <esp_now.h>
#include <WiFi.h>
#include <Wire.h>
#include <TM1637Display.h>
#include <CircusESP32Lib.h>

// 7-segment display
const byte PIN_CLK = 32;
const byte PIN_DIO = 33;
TM1637Display display(PIN_CLK, PIN_DIO);

// Disse deklarasjonene inneholder informasjonen som er nødvendig for å koble opp
mot Circus of Things

char ssid[] = "<Lokalt nettverksnavn>";       // Skriv inn navnet til ruteren din
her
char password[] =  "<Nettverkspassord>";      // Skriv inn passordet til ruteren
din her
```

```
char token[] = "<token>"; // Plasser din brukeridentitet her (token)
char server[] = "www.circusofthings.com";         // Nettsiden hvor CoT-serveren
ligger
char temperature_key[] = "21251";                 // Nøkkel for å kommunisere med
temperatursensoren hos BME280
char humidity_key[] = "20852";                    // Nøkkel for å kommunisere med
luftfuktighetssensoren hos BME280
char pressure_key[] = "13531";                    // Nøkkel for å kommunisere med
lufttrykksensoren hos BME280

// Legger inn et objekt og vi får tilgang til ruter og server ved CoT.
CircusESP32Lib circusESP32(server,ssid,password);

// Transmitted data
typedef struct {
    int instance;
    float temperature;
    float humidity;
    float pressure;
    String source;
} ESPNOWData;

// Oppretter et packet objekt for strukturen ESPNOWData.
ESPNOWData packet;

void setup() {
  Serial.begin(115200);
  // Initier ESP-NOW
  initESPNOW();
  // Sett lysstyrke på display
  display.setBrightness(7);

  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  uploadTheData(10000);
}

void initESPNOW() {

  // ESP-32 må være konfigurert i Station Mode for at ESP-NOW skal kunne benyttes.
```

```
  WiFi.mode(WIFI_STA);

  // Initier ESP-NOW
  esp_now_init();

  // esp_now_register_recv_cb aktiverer callback funksjonen "activateOnCallback"
når ESP-32 får inn en datapakke.
  esp_now_register_recv_cb(activateOnCallback);
}

void showAtDisplay (float showNumber){

  // Vis showNumber på display. showNumber er et desimaltall.
  // Vårt display kan bare vise heltall, men vi kan sette komma fast.
  // Vi har valgt å sette komma fast etter andre siffer (f.eks. 23.00)
  // Ved å multiplisere showNumber med 100,
  // konvertere tallet til et heltall får vi vist den med to desimaler

  showNumber = showNumber*100;
  display.clear();                        // Tøm displayet for data
 display.showNumberDecEx(int(showNumber), 0b01000000, false, 4, 0);   // Vis tall
  delay(500);                             // Vent i 500 millisek. (0,5 sek.)
}

void uploadTheData(int delayLength){
  // Laster opp data til CoT.
  // Tar i mot en delayLength som bestemmer hvor ofte data skal lastes opp til
skyen.
  digitalWrite(LED_BUILTIN, HIGH);
  circusESP32.begin(); // Let the Circus object set up itself for an SSL/Secure
connection
  circusESP32.write(temperature_key,packet.temperature,token); // Report the tem-
perature measured to Circus.
  circusESP32.write(humidity_key,packet.humidity,token); // Report the humidity
measured to Circus.
  circusESP32.write(pressure_key,packet.pressure,token); // Report the pressure
measured to Circus.
  digitalWrite(LED_BUILTIN, LOW);
  delay(10);
  WiFi.disconnect(true);
  WiFi.mode(WIFI_STA);
  delay(delayLength);
}
```

```
void activateOnCallback(const uint8_t * mac, const uint8_t *incomingData, int len)
{
  // Memcpy kopierer verdiene i incomingData til packet objektet.
  // Sizeof operasjonen angir antall bytes som utgjør packet,
  // og denne informasjonen brukes i memcpy for å bestemme antallet bytes som skal
kopieres.
  memcpy(&packet, incomingData, sizeof(packet));

  // Dataene skrives ut til monitoren
  Serial.print("Data packet nummer: ");
  Serial.print(packet.instance);
  Serial.print(" fra: ");
  Serial.println(packet.source);
  Serial.print("Temperaturen er: ");
  Serial.print(packet.temperature);
  Serial.println("C");
  Serial.print("Luftfuktigheten er : ");
  Serial.print(packet.humidity);
  Serial.println("%");
  Serial.print("Pressure is : ");
  Serial.print(packet.pressure/100);
  Serial.println("hPa");
  Serial.println();

  // Vis temperatur på 7-segment displayet
  showAtDisplay(packet.temperature);
}
```



Kurshefte beskriver et undervisningsopplegg som er en videreføring av kursmodulen: ESP – Grunnkurs programmering (YFL), og er utarbeidet spesielt med tanke på yrkesfagelever innen Elektro, men burde også passe for Bygg og anlegg. Opplegget har spesielt fokus på bruk av Wi-Fi-modulen til ESP32, som gir den mulighet til å fungere som en trådløs sensornode som kan kobles opp mot Internett. Kretsen er derfor spesielt egnet til bruk i forbindelse med Internett of Things (IoT).

Siden den bruker samme programmeringsverktøy som Arduino er den spesielt attraktiv for tidligere Arduino-brukere.

Det omtalte undervisningsopplegget legger størst vekt på å ta i bruk ESP32s Wi-Fi muligheter og viser hvordan ESP32 gir oss mulighet til å overføre signaler mellom en lokal sensornode og en nett-tilkoblet PC eller Smarttelefon. Utstyret egner seg derfor godt for trådløs styring og overvåking innen rekkevidden av en lokal ruter.

Kåre-Benjamin Rørvik
Bachelor Inst. for elektroniske systemer, NTNU
E-post: kbrorvik@stud.nnu.no

Nils Kr. Rossing
Dosent ved Skolelaboratoriet
E-post: nils.rossing@ntnu.no

ISBN
ISSN

Trykk: NTNU Grafisk senter

NTNU
Trondheim

Skolelaboratoriet
for matematikk, naturfag og teknologi

Tlf. 73 55 11 43
https://www.ntnu.no/skolelab

Institutt for fysikk

Institutt for elektroniske systemer