Opgave 1

Med udgangspunkt i jeres nuværende VsCode projekt skal I lave et nyt projekt, som skal opfylde nogle krav som specificeret herunder. I praksis laver man de krav, man nu kan uden at få det dårligt med, at man ikke når at lave alle punkter.

1) Som jeres nuværende projekt, skal det være muligt at kunne debugge i det nye projekt.

2) Lav det sådan, at I kan simulere den ene side af et blinklys. Det vil sige, at de 3 farver rød, gul og grøn skal i spil. Jeg vil foreslå, at man styrer dette med en state maskine. Jeg forklarer på forhånd, hvad en state maskine er => det er faktisk ikke specielt svært at styre en state maskine på dette niveau.

   Det skal også være muligt at ændre on tiderne for de enkelte lys ved at sende kommandoer til dette fra en PC af ved brug af seriel kommunikation.
   For at modtage karakterer fra en tilkoblet PC, kan koden herunder anvendes:

```
volatile int IncomingByte = 0; // for incoming serial data
volatile char IncomingChar = '0';
```

```
if (USE_SERIAL.available() > 0)
    {
      // read the incoming byte:
      IncomingByte = USE_SERIAL.read();
      IncomingChar = static_cast<char>(IncomingByte);

      TreatUartReceivedCharAccordingToState(IncomingChar);
    }
```

   I koden herover er funktionen: TreatUARTReceivedCharAccordingToState en funktion jeg har i mit program. I jeres program vil funktionen sikkert have et andet navn.
   Koden herover skal placeres i main loop i jeres program. **Så modsat hvad jeg har fortalt jer tidligere om god embedded programmering, står man her og poller efter data i stedet for at anvende en UART Receive Character interrupt rutine. Men det lever vi lige med i første omgang** 👍👍👍

   I skal selv definere en protokol til at styre dette med. Det er meget lettere, end det umiddelbart ser ud. Og jeg viser selvfølgelig et praktisk eksempel på dette.
   Som udgangspunkt skal on tiderne for de enkelte lys være som angivet her:
   Rød on : 3s
   Rød - gul on: 2s
   Grøn on: 5s
   Gul on: 1s

Mens I laver denne funktionalitet, er det nok en fordel, at I sætter tiderne ned for ikke at bruge for lang tid på at sidde og vente på de enkelte skift i lyset.
En rigtig god ide vil være, at I bruger ét af de oscilloskoper vi har til rådighed til at teste tiderne for de enkelte lys.
Det er "forbudt" at bruge delay funktionen til at implementere lyskryds funktionaliteten. I bør i stedet for oprette et passende antal Ticker timere til at styre det med. Har man mod på det, kan man også gå et lag dybere og direkte benytte sig af en eget udviklet Timer interrupt funktionalitet.
Et link til Ticker timere (som også findes i Moodle læringsrummet) er givet her:
https://www.tutorialspoint.com/what-is-arduino-ticker-library

3) Kobl ét eller flere AdaFruit i2C Bus display til jeres opstilling. Hvis man vælger at bruge mere end ét display, skal man bruge en I2CBus multiplexer for at kunne kommunikere med flere displays. I hvert fald hvis man ønsker at kunne skrive separate ting ud på de forskellige displays, skal man anvende multiplexeren. Se linket her som også findes i Moodle læringsrummet: **https://randomnerdtutorials.com/tca9548a-i2c-multiplexer-esp32-esp8266-arduino/**
   Hvis man vælger at bruge mere end ét display, skal man lige lodde stikben på en af de tilgængelige I2CBus multiplexere. Man kan evt. alliere sig her med en "erfaren" loddekolbe svinger, hvis man aldrig har loddet noget før.

4) Kobl en knap til jeres opstilling. Når man trykker på denne knap, skal der genereres et eksternt interrupt. I Interrupt Service rutinen for det eksterne interrupt, skal der sættes et flag og en ekstern interrupt counter skal tælles op med en. I programmets hovedløkke skal der testes for, om dette flag er sat. Når dette flag er sat, skal man skrive antallet af eksterne interrupts ud på Uart og også skrive antallet i (ét) AdaFruit display. Se link her for hvordan man arbejder med en knap/ekstern interrupt:
   https://roboticsbackend.com/arduino-interrupts/

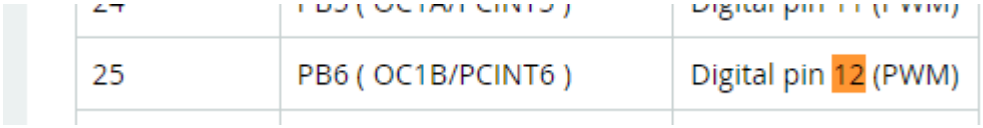
5) Udover at skrive antallet af eksterne interrupts ud på UART i display, skal man også bruge en DTH22, når der genereres et ekstern interrupt. På DHT22'eren skal man skifte mellem at:

   - Læse temperatur
   - Læse fugtighed
   - Læse temperatur index

   Så her vil det også være en god ide at lave en (mini) state maskine til at styre, hvad der skal læses på DTH22'eren. Igen kan det laves meget simpelt.
   Link til opkobling/programmering af en DTH22 er her:
   https://www.makerguides.com/dht11-dht22-arduino-tutorial/

Den aflæste værdi på jeres DTH22 skal I udskrive i display samt sende på UART også. Formater jeres udskrift således, at det er let at se, hvilken af de 3 typer måling, der er foretaget.

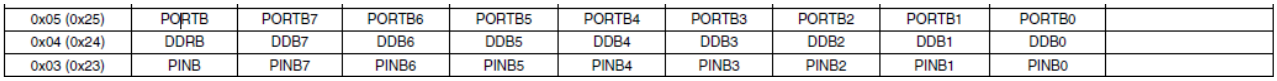
6) Udvid jeres state maskine med det i modtager fra en PC med denne funktionalitet. Udover at kunne modtage nye værdier for on tider for de forskellige tilstande i jeres blinklys, skal der nu også være mulighed for, at man kan sætte en vilkårlig bit i en vilkårlig port (Port A - Port H) høj eller lav. Jeg vil her foreslå, at I kobler en lysdiode på en af bit positionerne i en (eller flere) af de nævnte porte og så ser, om der er mulighed for at tænde/slukke denne lysdiode på baggrund af det, I har sendt til jeres AtMega SW fra jeres terminal program på PC'en.
   En skitsering af hvordan det kan gøres:
   - Start med at sende en adresse til jeres AtMega SW. Hvis man f.eks. har koblet en Lysdiode (og serie modstand !!!) til Digital Pin 12 på sit AtMega board, så skal man ind på linket her: https://docs.arduino.cc/hacking/hardware/PinMapping2560 og se hvilken AtMega port og bit position Digital Pin 12 er koblet til. På linket kan vi se, at det er PortB, bit-position 6 som Digital Pin 12 er koblet til, som det også fremgår af Figur 1 herunder.



25	PB6 (OC1B/PCINT6)	Digital pin 12 (PWM)

Figur 1: Sammenhæng mellem Digital Pin 12 og AtMega Port (Port B, bit position 6)

   - I Databladet for AtMega2560 (sie 389) kan vi nu se, at PortB er placeret på adresserne 0x0023 - 0x0025, som det fremgår af Figur 2 herunder.



0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	

Figur 2: Port adresser for PortB på en AtMega2560

7) I skal have defineret en uint_8t pointer i jeres program. Denne pointer skal I sætte til at pege på den modtagne adresse fra PC'en.

8) I skal nu sende (og modtage) en bit position (bit 0 - 7) fra jeres terminal program på PC'en.

9) Sluttelig skal I modtage en værdi (0 eller 1), der viser, om I vil sætte jeres port høj (tænde Lysdioden på porten), eller om I vil sætte jeres port lav (slukke Lysdioden på porten).

10) I skal sætte jeres state maskine til at stå i forskellige tilstande alt afhængig af, om I modtager adressen, bit position eller bit værdien.

11) Når I som det sidste har modtaget bit værdien, skal I til at sætte den angive port høj eller lav. Dette kan gøres som skitseret herunder for henholdsvis høj og lav "situationen".

- **Sætte porten høj:**
  1) Sæt jeres uint_8t pointer til at pege på den modtagede adresse fra PC'en. Jeg antager her, at i har givet værdien 0x0025, hvis det er PortB, I har koblet en lysdiode til => altså selve PORTB output registret.

  2) Læs værdien på den port som pointeren peger på og lav en **bitvis Or operation** med værdien af den bit position, I arbejder på. Lyder kringlet men er faktisk meget let, netop fordi vi bruger en pointer!!! Se kode eksempel her:

     ***P_AddressPointer = *P_AddressPointer | (1 << bit position);**
     Eller den kompakte notationsform:
     ***P_AddressPointer |= (1 << bit position);**

     <u>Hvor bliver ting dog bare let, når man gør brug af en pointer !!!!!</u>

     I skal også lige huske at den pågældende port skal være sat til Output. Ellers nytter det ikke så meget, at vi sætter porten til høj (eller lav). Men heldigvis er dette også meget let, da vi netop bruger en pointer til at styre det med.

     Da vi eksemplet her antager, at vores P_AddressPointer indeholder værdien 0x0025, så skal vi i henhold til Figur 2 "bare" sætte vores P_AddressPointer til at pege på adresse **0x0024** (adresse 0x0024 er direction registret for PORTB !!!) og så ellers udføre samme operation på denne adresse, som vi gjorde på vores PortB output register. I kode:

     ***(P_AddressPointer - 1) = *(P_AddressPointer - 1) | (1 << bit position);**
     Eller den kompakte notationsform:
     ***(P_AddressPointer - 1) |= (1 << bit position);**

     Heldigvis er alle port adresser i AtMega2560 chippen opbygget på samme måde som PortB, så vi kan altid regne med, at direction registret for en given port <u>ALTID</u> er placeret på en adresse, der er én mindre en portens output register!!!

- **Sætte porten lav:**
  1) Her gør vi det på helt samme måde, som når vi vil sætte porten høj. Den eneste forskel er, at vi skal sætte Port bitten og Port direction bitten til lave i stedet for til høje. Og da vi er rigtig gode til det med både at sætte en (eller flere) bit(s) i en port

høj eller lav, ved vi godt, at vi skal lave en bitvis And operation med det modsatte bitmønster for at opnå den ønskede effekt. Se kode eksempel her:

Output Register:
***P_AddressPointer = *P_AddressPointer & ~(1 << bit position);**
Eller den kompakte notationsform:
***P_AddressPointer &= ~(1 << bit position);**

Direction Register:
***(P_AddressPointer - 1) = *(P_AddressPointer - 1) & ~(1 << bit position);**
Eller den kompakte notationsform:
***(P_AddressPointer - 1) &= ~(1 << bit position);**

12) Happy Coding 👍👍👍

Ting jeg umiddelbart kommer til at tænke på til besvarelse af punkt 1 - 5 (og efterfølgende punkter), når jeg ser en opgave som denne er:

- Enums til at beskrive tilstande i en state maskine
- Løkke til at gennemløbe de enkelte states i en state maskine
- Tabeller man kan slå op i, i stedet for at f.eks. at bruge Case-Switch.
- Funktions pointere
- Brugen af structs

Man bestemmer selvfølgelig selv, hvordan man vil opbygge sit program, men med de her opskrevne ting, vil det være meget lettere at vedligeholde og senere udvide sit program.