NTNU
Kunnskap for en bedre verden

Institutt for elektroniske systemer

IELET1002 - Datateknikk

Hackathon 1 Arduino

Authors:
Karl Emil Ingebrigtsen
Ellen Iren Johnsen
Knut Ola Nøsen
Petter Wandsvik
Elias Loe Eritsland
Johannes Ravn Munkvold
Trond Forstrøm Christiansen

16. februar 2023

Mål for dette hackathonet:

- Lære konseptet Interrupt
- Forstå grunnleggende hva en motordriver er og hvordan den brukes
- Bli kjent med I2C-protokollen og OLED-skjerm
- Anvende ulike programmeringskonsepter til et større system

Generelle krav:

- Bruk de eksakte navn og spesifikasjoner gitt i oppgaveteksten
- Alle pins skal konfigureres med variabler. Eks: `pinMode(BUTTON_PIN)`, ikke `pinMode(2)`
- Hackathon 1 skal være et individuelt gruppearbeid. Det vil si at gruppene jobber hver for seg. Juks og plagiat kan du lese mer om på ntnu.no.
- Hackathon 2 vil bygge på Hackathon 1, så ta vare på utdelt batteri og kildekode
- Hackathon 1 gjennomføres i to deler: Del 1 består av fysiske oppmøtet og demonstrasjon av løsningen til stud.ass på sal. Del 2 består av refleksjonsnotat.
  - Dersom gruppen ikke blir ferdige eller ønsker å levere løsningen på et senere tidspunkt skal løsningen leveres i .pdf-format med bilder for å demonstrere funksjonalitet samt tilhørende kode. Hvert gruppemedlem leverer samme eksemplar. Frist for etterlevering av Hackathon 1 har samme frist som refleksjonsnotatet. Se under.
- Frist for innlevering av refleksjonsnotat: mandag 20/2, kl. 23.59. Se informasjon rundt refleksjonsnotat på BB.

OLED skjerm

I øvingen tenkes det at vi skal bruke Arduinoen tilkoblet et batteri, eller en annen strømkilde enn PCen. Når Arduinoen ikke er koblet til en PC via USB, har vi heller ikke tilgang til Serial Monitor. Vi bruker derfor en OLED skjerm til å vise informasjon til brukeren. Skjermen dere har i kittet er monofarget og har 128x64 piksler. Skjermen kan tenkes på som et koordinatsystem, og det øvre hjørnet til venstre er posisjon (0,0).
For at vi skal få kommunikasjon mellom Arduinoen og skjermen bruker vi I2C-protokollen, og et bibliotek som hører til den spesifikke Adafruit-skjermen.

For å koble opp skjermen kobler vi ledninger mellom skjermen og Arduinoen på følgende plasser:

- SCL til A5
- SDA til A4
- V_{CC} til 5 V
- GND til GND

For å laste ned Arduino-biblioteket til skjermen gjør følgende:

- Trykk på "Sketch"
- Trykk på "Include library"
- Trykk på "Manage libraries"
- Søk på "Adafruit SSD1306"i søkefeltet
- Trykk på "Install"

Dere kan nå kjøre en av eksempelkodene i biblioteket for å verifisere at dere har koblet skjermen riktig.

- Trykk File
- Examples
- Adafruit SD1306
- velg "ssd1306_128x64_I2C"

Når dere skal programmere selv kan disse kommandoene være nyttige:

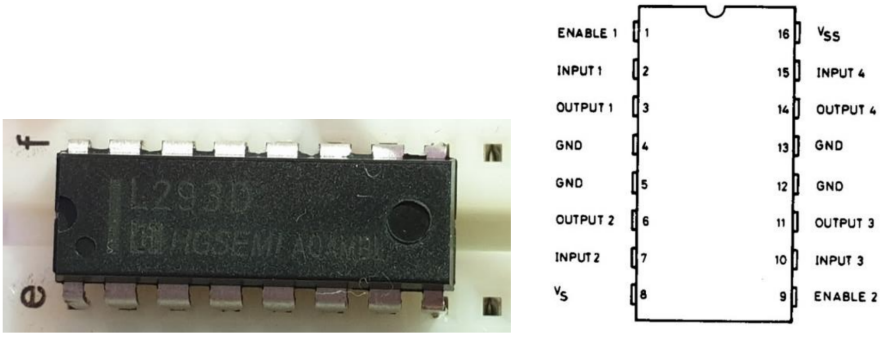
- display.clearDisplay() - Skrur av alle piksler
- display.setCursor(x,y) - Setter startkoordinatene for tekst
- display.setTextSize(n) - Setter fontstørrelsen, skj ermen støtter mellom 1 og 8
- display.print("message") - printer teksten ved x,y
- display.display() - denne metoden gjør at endringer tar effekt

DC-motoren

DC-motorer kommer i forskjellige varianter og størrelser og i denne oppgaven skal en liten børstemotor brukes. Børsten er på innsiden av motoren og sørger for magnetfeltsendringene mens motoren roterer. At motoren er børstedrevet gjør at den kan kjøres uten en motordriver, men da blir det uten kontroll. Motoren kan trekke store strømmer ved oppstart eller last så vi bruker derfor en motordriver til å kontrollere hastigheten, og for å unngå at mikrokontrolleren utsettes for de store strømmene som går gjennom motoren.

Motordriver

En motordriver er en krets laget for å styre DC-motorer. De bruker logiske inngangssignal, f.eks. fra mikrokontrollere, for å styre aktuatorer / motorer. Som regel inkluderer motordriverene fire INPUT-pinner, fire OUTPUT-pinner og to ENABLE-pinner. Motordriveren kan da styre to sett med motorer. I arduinokitet fra Elektra finner man en svært vanlig motordriver: L293D. Dette er en integrert krets (IC) som kan brukes til forskjellige småskala prosjekter, og på figur 4 kan man se pinne-oversikten.
En motordriver har to pinner for spenning, en for motorspenning, og en for logikkspenning. Driveren har to forskjellige spenningspunkter for å holde motoren og mikrokontrolleren separat da mikrokontrolleren ikke tåler strømmen motoren trekker.
V_s er spenningen for motoren. Her kobles strøm og spenningskilden til motoren. Dette er gjerne en større kilde som kan tilføre mer effekt [Watt] enn det mikrokontrolleren kan.



Figur 1: L293DPinout

V_{ss} er spenningen for logikken. Her kobles strøm og spenningskilden til mikrokontrolleren.
Det er fire GND pinner på driveren. Strøm fra både mikrokontrolleren og motoren. Alle pinnene kobles opp for å fordele effekten [Watt] over kablene, og motordriveren. Dette gjør vi for å motvirke varmegang.
ENABLE1 brukes til å skru på motoren, og kontrollere hastighet. Hastighet reguleres med kommandoen *analogWrite*.
INPUT1 og INPUT2 pinnene styrer retning på motoren. Kun en av disse skal settes høy av gangen, dersom begge settes høy skrus motoren av. For å endre retning byttes hvem av disse pinnene som er høy.
OUTPUT1 og OUTPUT2 er tilkoblingspunktene for motoren. Her har det lite å si hvilken vei man kobler, men retningen motoren kjører vil reverseres dersom punktene byttes om.
Det er også enda et sett med ENABLE, INPUT og OUTPUT, disse er for kobling av enda en motor.

I2C-protokollen

I I2C-protokollen blir to ledninger brukt til å sende og motta data mellom flere enheter: SCL og SDA. SCL sin jobb er å synkronisere informasjonsoverføringen mellom enhetene. Det er på SDA-ledningen informasjon blir overført. Når enheter er koblet sammen med I2C har de en av to roller: master eller slave. Det kan kun være en master i et system, men man kan koble på opptil 127 slaver. Masteren starter opp kommunikasjonen og adresserer slave-enhetene, disse formidler bare informasjon når master ber om det. Dette forhindrer kræsj mellom data fra ulike enheter. Alle I2C enheter har en egen adresse, det er denne Arduinoen bruker for kommunikasjon. De fleste I2C enheter har en kontakt du kan slutte (lodde over) eller splitte for å endre adressen til en annen. Dette er for å øke kompatibilitet mellom ulike I2C-enheter, da kun en slave kan ha en adresse av gangen. Disse adressene står som oftest oppført på kretskortet til slaven, eller så skal det være oppført i dokumentasjonen. I Hackaton-oppgaven fungerer Arduino kortet som master og OLED skjermen som slave. Mer om I2C **her**.

Interrupts

En ISR er en spesiell type funksjon som ikke kan ta inn noen parametere eller returnere noe. Disse avbryter koden som kjører, for å kjøre interruptfunksjonen. Disse bør derfor generelt være korte og raske for å bryte minst mulig med kodeflyten. Dersom man inkluderer flere interrupt-funksjoner i et program kan de ikke kjøre på samme tid, men vil utføres etter prioritet. Det er viktig å bemerke at vi unngår å bruke millis og delay i interrupts, for at interrupten skal kjøre raskest mulig. Det er også greit å bemerke at telleren for millis blir stoppet opp midlertidig under en interrupt. Siden ISR funksjoner kjøres av interrupts, bryter de med den normale setup-loop programflyten. Med dette kommer noen fallgruver som dere kan lese om i Arduino dokumentasjonen. Disse kan derimot unngås ved å kun bruke datatypen `volatile bool` eller `volatile byte`. `volatile` gjør at compileren ikke optimaliserer bort logikken bak variabelen, siden den tilsynelatende aldri endres i hovedprogrammet. `bool` og `byte` skal brukes fordi de er datastrukturer som ikke er større enn 1 byte. Hvis for eksempel en ISR aktiveres når hovedprogrammet er midt i å lese fra en `int`, kan man risikere at variabelen oppdateres mens den blir lest, noe som kan føre til feil verdier. For å aktivere en interrupt bruker vi en Pinne på Arduinoen (vi kan bruke pinne 2 og 3 på Arduino UNO). Dette setter vi opp i setup.

```
attachInterrupt(digitalPinToInterrupt(pin), interruptFunction, mode);
```

- pin: Pinnen vi leser av
- interruptFunction - Funksjonen som skal kjøre når vi detekterer endringen. Legg merke til over at interruptFunction legges inn i attachInterrupt uten parentes. Dette er fordi selve funksjonen er argumentet til attachInterrupt, ikke returverdien til funksjonen..
- mode: definerer når en interrupt skal trigges (f.eks LOW: trigger når pinnen er lav)

Moduser vi kan bruke med Arduino Uno:

- LOW: trigger når pinnen er lav
- CHANGE: trigger når pinnen endrer verdi
- RISING: trigger når pinnen går fra lav til høy
- FALLING trigger når pinnen går fra høy til lav

Se **Arduino Reference** for eksempelkode og informasjon.

Oppgaver

Etter mye høylytt oppussing på elektro bygget er temperamentet høyt blant de ansatte. Arne har derfor bestilt 50 små håndholdte vifter for å kjøle ned situasjonen som må være ferdige innen mandag. På grunn av den korte tidsfristen har vi ikke tid til og bestille de inn, det blir derfor deres jobb og lage viftene.Dere skal lage en håndholdt vifte som består av en børstemotor for å drive viftebladet, en OLED skjerm for å vise nødvendig data fra motoren samt knapper for å kontrollere hastighet.

1 Delfunksjonalitet

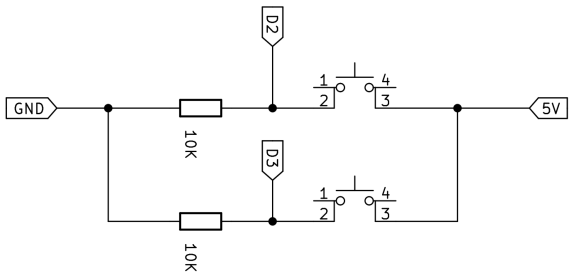
Fordel oppgavene til gruppemedlemene og løs deloppgavene hver for dere. Etter at koden til deloppgavene har blitt skrevet og funksjonaliteten testet skal dere sy sammen de individuelle kodene til ett større program. Det er derfor viktig at kodene blir skrevet på ryddig og oversiktlig måte. Det er også viktig at dere bruker de samme variabelnavnene og funksjonssignaturene som er definert øverst i hver oppgave, slik at alle funksjonene deres fungerer sammen til slutt.

a) Knappestyring

Variabler:

```
volatile byte motorState = 0;

const int SPEED_UP_BUTTON_PIN = 2;
const int SPEED_DOWN_BUTTON_PIN = 3;
```

i) Koble opp to knapper med pulldown motstander til GPIO 2 og 3. Her er det viktig å benytte GPIO 2 og 3 ettersom det er de eneste GPIO pinnene på Arduino Uno som kan brukes til interrupts.



Figur 2: Button diagram

ii) Lag to funksjoner `increaseMotorState()` og `decreaseMotorState()`. Funksjonene skal henholdsvis øke og redusere verdien til `motorState` med trinn på +-1, fra og med 0 til og med 3.

iii) Test funksjonene ved å kalle dem og printe ut `motorState` i `void loop()`, og sjekk at de oppfører seg som forventet.

iv) Lag to funksjoner `onSpeedUpButtonPressed()` og `onSpeedDownButtonPressed()`. Funksjonene skal ikke returnere noe og har heller ingen parameter. Disse funksjonene skal kalle funksjonene fra oppgaven over. Disse er helt vanlige funksjoner, men når de brukes av `attachInterrupt()` blir de til ISR funksjoner.

v) Funksjonene `onSpeedUpButtonPressed()` og `onSpeedDownButtonPressed()` skal settes opp for å kalles fra interupt. Bruk `attachInterrupt()` til å sette funksjenene som ISR. Les "Interrupts"dokumentasjonen over for å se hvordan dette gjøres. Grunnen til at disse funksjonene lages

i tillegg til funksjonene fra oppgaven over er at de andre funksjonene skal brukes til noe annet senere, og at det er mer tydelig å forstå hva koden gjør ut ifra funksjonsnavnene.

vi) Print `motorState` i `void loop()` og trykk på knappene for å sjekke at det fungerer. Dere vil kanskje observere at `motorState` noen ganger øker med mer enn 1 per knappetrykk. Dette er på grunn av "bounce", som skjer fordi knappene er billige og spretter opp og ned når de blir trykt ned. Vi ser symptomet av dette oftere, fordi interrupts er raskere enn metoden dere har brukt for flanketrigging før. Dette er en byrde her, men i tilfellet komunikasjon og encodere er dette nødvendig for å sørge for at men plukker opp alle signalene. Dette kan løses med "button debounce", noe dere skal gjøre i oppgave 3. For øyeblikket kan dere bare ignorere problemet.

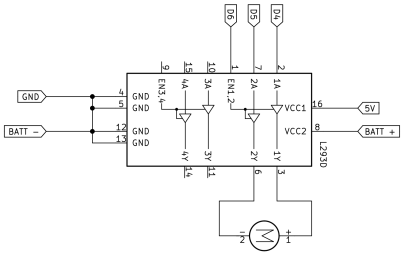
Oppgave 1a LF

```
const int SPEED_UP_BUTTON_PIN = 2;
const int SPEED_DOWN_BUTTON_PIN = 3;

volatile byte motorState = 0;

void increaseMotorState()
{
    if (motorState < 3)
    {
        motorState += 1;
    }
}

void decreaseMotorState()
{
    if (motorState > 0)
    {
        motorState -= 1;
    }
}

void onSpeedUpButtonPressed()
{
    increaseMotorState();
}

void onSpeedDownButtonPressed()
{
    decreaseMotorState();
}

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

    pinMode(SPEED_UP_BUTTON_PIN, INPUT);
    pinMode(SPEED_DOWN_BUTTON_PIN, INPUT);

    attachInterrupt(digitalPinToInterrupt(SPEED_UP_BUTTON_PIN),
    onSpeedUpButtonPressed, FALLING);
    attachInterrupt(digitalPinToInterrupt(SPEED_DOWN_BUTTON_PIN),
    onSpeedDownButtonPressed, FALLING);
}

void loop()
{
    Serial.println(motorState);
    // Serial.println(speedUpButtonState);
    // Serial.println(speedDownButtonState);
}
```

b) Motorstyring

i) Koble opp motoren og motordriveren i henhold til skjematikken og test at den fungerer. Dette kan gjøres med å sette *enable* pinnen høy ved og flytte ledningen fra A0 til 5V.

Obs! Her er det viktig og passe på at man kobler V_{ss} og V_s riktig så man ikke sender 9v til mikrokontrolleren.



Figur 3: Motor diagram

ii) Lag en funksjon `writeMotorSpeed()` som setter rotasjonsretningen til motoren (en av INPUT1 og INPUT 2 på motordriveren HIGH og den andre LOW), og hastigheten til motoren ved og skrive en analog verdi til *enable*. Tallverdien som skal skrives til motoren skal tas inn som et argument.

iii) Test at funksjonen fungerer ved å kjøre `writeMotorSpeed(100);` i `void loop()`, og se at motorfarten endres når argumentet endres.

iv) Det skal nå lages en funksjon med signaturen `updateMotorSpeed()`. Denne funksjonen skal inneholde en switch case som bruker den globale variabelen `motorState` til og sette motorhastigheten til en av fire hastigheter mellom av og helt på. For casene av `motorState` 0, 1, 2, og 3, skal det kalles `writeMotorSpeed();` med argumentet 0, 100, 200, 255 respektivt.

v) Test at `updateMotorSpeed` fungerer ved å kalle den i `void loop()` og hardkode en verdi for `motorState`.

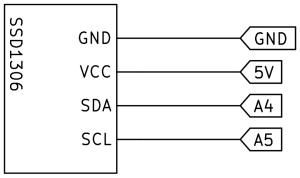
Oppgave 1b LF

```
const int MOTOR_CCW_PIN = 4;
const int MOTOR_CW_PIN = 5;
const int MOTOR_ENABLE_PIN = 6;

volatile byte motorState = 0;

void writeMotorSpeed(int speed)
{
    // Set the direction
    digitalWrite(MOTOR_CW_PIN, LOW);
    digitalWrite(MOTOR_CCW_PIN, HIGH);

    // Set the speed
    analogWrite(MOTOR_ENABLE_PIN, speed);
}

void updateMotorSpeed()
{
    switch (motorState)
    {
    // Stopped
    case 0:
        writeMotorSpeed(0);
        break;
    // Slow
    case 1:
        writeMotorSpeed(100);
        break;
    // Medium
    case 2:
        writeMotorSpeed(200);
        break;
    // Fast
    case 3:
```

```
        writeMotorSpeed(255);
        break;
    }
}

void increaseMotorState()
{
    if (motorState < 3)
    {
        motorState += 1;
    }
}

void decreaseMotorState()
{
    if (motorState > 0)
    {
        motorState -= 1;
    }
}

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

    pinMode(MOTOR_CCW_PIN, OUTPUT);
    pinMode(MOTOR_CW_PIN, OUTPUT);
    pinMode(MOTOR_ENABLE_PIN, OUTPUT);
}

void loop()
{
    // Up
    increaseMotorState();
    updateMotorSpeed();
    delay(1000);

    increaseMotorState();
    updateMotorSpeed();
    delay(1000);

    increaseMotorState();
    updateMotorSpeed();
    delay(1000);

    increaseMotorState();
    updateMotorSpeed();
    delay(1000);

    // Down
    decreaseMotorState();
    updateMotorSpeed();
    delay(1000);

    decreaseMotorState();
    updateMotorSpeed();
    delay(1000);

    decreaseMotorState();
    updateMotorSpeed();
    delay(1000);

    decreaseMotorState();
    updateMotorSpeed();
    delay(1000);
}
```

c) OLED skjerm

i) Koble opp OLED skjermen i henhold til skjematikken og test at den fungerer med å bruke eksempelfilen **ssd1306 128x64_I2** fra **Adafruit SSD1306** biblioteket.



Figur 4: L293DPinout

ii) Det skal skrives to funksjoner for å styre OLED-skjemen.

- `setupOled()` Denne funksjonen skal inneholde all koden som trenges for og initialisere OLED skjermen. Denne koden kan kopieres fra eksempel filen, men vær sikker på at dere vet hva koden dere kopierer gjør.
- `printToDisplay()` Denne funksjonen skal ta inn en `String`, cleare displayet, printe teksten til OLED skjermen, og displaye teksten.

iii) Test funksjonaliteten til koden ved å kalle den i `void loop()`

Oppgave 1c LF

```
#include<avr/wdt.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

//////////////////// OLED Setup from tutorial ////////////////////////
// https://arduinogetstarted.com/tutorials/arduino-oled

#define SCREEN_WIDTH 128 // OLED display width,  in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// declare an SSD1306 display object connected to I2C
Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

void setupOled()
{
    // initialize OLED display with address 0x3C for 128x64
    if (!oled.begin(SSD1306_SWITCHCAPVCC, 0x3C))
    {
        Serial.println(F("SSD1306 allocation failed"));

        // Enable the watchdog timer so the microcontroller resets after 2 seconds
        wdt_enable(WDTO_2S);
        while (true) {
            // Since the conenction failed,
            // wait for the watchdog timer to restart the microcontroller
        }
    }

    delay(2000);         // wait for initializing
    oled.clearDisplay(); // clear display

    oled.setTextSize(1);          // text size
    oled.setTextColor(WHITE);     // text color
    oled.setCursor(0, 10);        // position to display
    oled.println("Starting ..."); // text to display
    oled.display();               // show on OLED
    delay(2000);
}

//////////////////// The actual exercise ////////////////////////
```

```
volatile byte motorState = 0;

void printToDisplay(String message)
{
    oled.clearDisplay();
    oled.println(message);
    oled.display();
}

void printMotorSpeed()
{
    String message = "Motor speed: " + String(motorState);
    printToDisplay(message);
}

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

    setupOled();
}

void loop() {
    // printToDisplay("Test");

    motorState += 1;
    printMotorSpeed();
}
```

2 Sett sammen koden

a) Forklaring

Forklar koden dere skrev i oppgave 1 til de andre på gruppen slik at alle forstår den.

b) Vifte

Sett sammen de ulike delene fra de foregående oppgavene til ett program. Koden skal:

- Forandre verdien til tilstandsvariabel `motorState` med interupts fra knappene.
- Sette hastigheten til motoren med en switch-case som bruker variabelen `motorState`
- Vise hvilken tilstand viften er i ved og printe variabelen `motorState` og forklarende tekst til OLED skjermen.

- Alt koden som ligger i `loop` skal være kall til funksjonene dere allerede har laget.
- Koden skal ikke inneholde noe blokkerende kode. DVS, kode som forsinker prosessen (f.eks delay, for-løkker og while-løkker).

Oppgave 2 LF

```
////////////////////////////////////////////////////////
//////////////////// Exercise 1 ////////////////////////
////////////////////////////////////////////////////////

#include<avr/wdt.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

```
//////////////////// OLED Setup from tutorial ///////////////////////
// https://arduinogetstarted.com/tutorials/arduino-oled

#define SCREEN_WIDTH 128 // OLED display width,  in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// declare an SSD1306 display object connected to I2C
Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

void setupOled()
{
    // initialize OLED display with address 0x3C for 128x64
    if (!oled.begin(SSD1306_SWITCHCAPVCC, 0x3C))
    {
        Serial.println(F("SSD1306 allocation failed"));

        // Enable the watchdog timer so the microcontroller resets after 2 seconds
        wdt_enable(WDTO_2S);
        while (true) {
            // Since the conenction failed,
            // wait for the watchdog timer to restart the microcontroller
        }
    }

    delay(2000);         // wait for initializing
    oled.clearDisplay(); // clear display

    oled.setTextSize(1);          // text size
    oled.setTextColor(WHITE);     // text color
    oled.setCursor(0, 10);        // position to display
    oled.println("Starting ..."); // text to display
    oled.display();               // show on OLED
    delay(2000);
}

//////////////////// The actual exercise ///////////////////////

volatile byte motorState = 0;

void printToDisplay(String message)
{
    oled.clearDisplay();
    oled.println(message);
    oled.display();
}

void printMotorSpeed()
{
    String message = "Motor speed: " + String(motorState);
    printToDisplay(message);
}

////////////////////////////////////////////////////////
//////////////////// Exercise 2b ///////////////////////
////////////////////////////////////////////////////////

const int MOTOR_CCW_PIN = 4;
const int MOTOR_CW_PIN = 5;
const int MOTOR_ENABLE_PIN = 6;

// volatile byte motorState = 0;

void writeMotorSpeed(int speed)
{
    // Set the direction
    digitalWrite(MOTOR_CW_PIN, LOW);
    digitalWrite(MOTOR_CCW_PIN, HIGH);

    // Set the speed
    analogWrite(MOTOR_ENABLE_PIN, speed);
}
```

```
void updateMotorSpeed()
{
    switch (motorState)
    {
    // Stopped
    case 0:
        writeMotorSpeed(0);
        break;
    // Slow
    case 1:
        writeMotorSpeed(100);
        break;
    // Medium
    case 2:
        writeMotorSpeed(200);
        break;
    // Fast
    case 3:
        writeMotorSpeed(255);
        break;
    }
}

void increaseMotorState()
{
    if (motorState < 3)
    {
        motorState += 1;
    }
}

void decreaseMotorState()
{
    if (motorState > 0)
    {
        motorState -= 1;
    }
}

////////////////////////////////////////////////////////
/////////////////////// Exercise 3 ////////////////////////
////////////////////////////////////////////////////////

const int SPEED_UP_BUTTON_PIN = 2;
const int SPEED_DOWN_BUTTON_PIN = 3;

// volatile byte motorState = 0;
//
// void increaseMotorState()
// {
//     if (motorState < 3)
//     {
//         motorState += 1;
//     }
// }
//
// void decreaseMotorState()
// {
//     if (motorState > 0)
//     {
//         motorState -= 1;
//     }
// }

void onSpeedUpButtonPressed()
{
    increaseMotorState();
}

void onSpeedDownButtonPressed()
{
    decreaseMotorState();
}
```

```
////////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////////////////

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

    setupOled();

    pinMode(MOTOR_CCW_PIN, OUTPUT);
    pinMode(MOTOR_CW_PIN, OUTPUT);
    pinMode(MOTOR_ENABLE_PIN, OUTPUT);

    pinMode(SPEED_UP_BUTTON_PIN, INPUT);
    pinMode(SPEED_DOWN_BUTTON_PIN, INPUT);

    attachInterrupt(digitalPinToInterrupt(SPEED_UP_BUTTON_PIN),
    onSpeedUpButtonPressed, FALLING);
    attachInterrupt(digitalPinToInterrupt(SPEED_DOWN_BUTTON_PIN),
    onSpeedDownButtonPressed, FALLING);
}

void loop()
{
    updateMotorSpeed();

    printMotorSpeed();
}
```

3 Ekstra funsjonalitet

Koden til viften skall nå utvides med ekstra funksjonalitet. For å gjøre dette enklere og mer oversiktlig anbefaler vi at dere kopierer inn ferdige biblioteker som dere har fått gjennomgang av i forelesningene. Disse bibliotekene er Timer og Button. Med disse skal dere legge til følgende forbedringer i programmet. Hvis dere ikke deltok i den forelesningen, se forskjellige eksempler på bruk **her** og kontakt en studass om dere trenger mer forklaring.

a) Aktiv tid

Vi vill vite hvor lenge viften har stått på. Kopier Timer structet og lim det inn på toppen av kodefilen. Timer stuctet ligger **her** og eksempel på bruk ligger **her**. Lag en global timer `motorActiveTime` som skal spore antall millisekunder viften har vært aktiv, og en funksjon `printActiveTime()` som printer den aktive tiden til OLED-skjermen. For å vite hvor lenge timeren har gått kan dere bruke Timer funksjonen `getElapsedTime()` som returnerer antall millisekunder siden Timer funksjonen `reset()` ble kjørt. Lag en funksjon `updateActiveTime()` som resetter `motorActiveTime` timeren når motoren er av.

b) Rullerende OLED

Vi ønsker å vise frem både den aktive tiden til viften, og hastigheten til viften om hverande. Lag en global variabel `oledSequenceStep` med startverdien 0, og en funksjon `updateOledScreen()` som benytter en switch-case til å bytte mellom hvilken verdi som vises frem.

Lag en ny global Timer variabel `oledSequenceTimer` og en funksjon `updateOledTimer()` som inkrementerer `oledSequenceStep` hvert sekund. Se eksempelkoden fra forelesningen om structs her for hvordan Timer kan brukes for å oppnå dette.

c) Debounce

Som diskutert i en tidligere oppgave vil ett knappetrykk kunne øke `motorState` flere ganger på grunn av "bounce". Det er ikke mulig å løse "debounce"inni en ISR (interrupt funksjon) og derfor må det lages en funksjon som håndterer debounce-logikken. Siden det nå er mye logikk som skal kjøre på hver knapp, kan det lett bli vanskelig å feilsøke og lese koden. Samtidig er det ikke lett å bruke denne koden i andre prosjekter senere. Derfor er det nå hensiktsmessig å benytte en struct for knappene.

Videre skall dere gjøre noen forandringer på biblioteket dere får utdelt. Dette kan være lit forvirrende, så kontakt en studass ved spørsmål, forvirring eller problemer.

i) Trykk **her** for å kopiere Button biblioteket, og lim det inn øverst i kodefilen under Timer structet. Se **her** for eksempelkode på hvordan knapp structet skal brukes. Denne eksempelkoden er ganske nærme slik deres kode skal se ut, så spør en studass om forklaring hvis dere ikke skjønner den.

ii) Oppdater koden slik at `increaseMotorState()` blir kjørt av en egen funksjon:

`updateMotorSpeedControls()` utenom ISRene, som vist i eksempelkoden for Button structet.

iii) Denne koden støtter fremdeles ikke debounce, men nå er det enklere å legge det til. Gjør følgende endringer i Button structet:

iv) Lag en Timer variabel `debounceTimer` i Button structet.

v) Kall `reset()` funksjonen til `debounceTimer` i `void updateInterrupt()`, slik at `debounceTimer` begynner å telle på nytt når `void updateInterrupt()` kjøres.

vi) Legg til en if-setning øverst i `void update()`, som returnerer fra funksjonen hvis `debounceTimer` ikke er ferdig å telle ned fra 50 millisekunder. Det vil sørge for at resten av innholdet i funksjonen ikke kjøres før vi har ventet i 50 millisekunder, slik at knappen har fått tid til å stabilisere seg.

Hint: Når man returnerer fra en funksjon vil funksjonen avslutte kjøringen og ikke kjøre kodelinjene under retur setningen. For funksjoner med `void` som returtype kan man skrive `return;` uten å oppgi en verdi for å avslutte kjøringen av funksjonen. En if-setning som returnerer fra en funksjon for å ikke kjøre resten av koden i funksjonen kalles for en "Guard Clause", og dere kan lese mer om det **her** eller se en video om det **her**

vii) Test at funksjonaliteten til koden er det samme som før, men at problemet med "bounce"er løst.

d) Holde inn knapp

Vi ønsker også å ha mer detaljert kontroll over viftehastigheten. Øk antallet motor-hastigheter fra fem til ti. For at man ikke skal trenge og trykke på knappen veldig mange ganger for store forandringer i hastighet ønsker vi også og kunne holde inne knappen. Legg til en global variabel (ikke i Button structet) `buttonHoldTimer`, og en funksjon `updateButtonHolds()` som oppdaterer motorState hvert sekund når knappene holdes nede. Forskjellige måter å kombinere knapper og timere på, kan dere finne i eksemplene fra struct forelesningen her.

Oppgave 3 LF

```
#include <avr/wdt.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

struct Timer
{
    unsigned long startTime;
```

```
    void reset()
    {
        startTime = millis();
    }

    unsigned long getElapsedTime()
    {
        return millis() - startTime;
    }

    bool isFinished(unsigned long duration)
    {
        return getElapsedTime() >= duration;
    }
};

struct Button
{
    int pin;
    // true for pulldown, false for pullup
    bool pulldown;

    bool state = false;
    bool prevState = false;
    bool pressed = false;
    bool released = false;

    int debounceTime = 50;
    Timer debounceTimer;

    volatile bool detectedInterrupt = false;

    void updateInterrupt()
    {
        detectedInterrupt = true;
        debounceTimer.reset();
    }

    void update()
    {
        if (!debounceTimer.isFinished(debounceTime))
            return;

        if (!detectedInterrupt)
            return;

        if (pulldown)
            state = digitalRead(pin);
        else
            state = !digitalRead(pin);

        pressed = state && !prevState;
        released = !state && prevState;
        prevState = state;

        detectedInterrupt = false;
    }
};

volatile byte motorState = 0;

const int MOTOR_CCW_PIN = 4;
const int MOTOR_CW_PIN = 5;
const int MOTOR_ENABLE_PIN = 6;

Button speedUpButton;
Button speedDownButton;

//////////////////// OLED Setup from tutorial ////////////////////////
// https://arduinogetstarted.com/tutorials/arduino-oled

#define SCREEN_WIDTH 128 // OLED display width,  in pixels
```

```
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// declare an SSD1306 display object connected to I2C
Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

void setupOled()
{
    // initialize OLED display with address 0x3C for 128x64
    if (!oled.begin(SSD1306_SWITCHCAPVCC, 0x3C))
    {
        Serial.println(F("SSD1306 allocation failed"));

        // Enable the watchdog timer so the microcontroller resets after 2 seconds
        wdt_enable(WDTO_2S);
        while (true)
        {
            // Since the conenction failed,
            // wait for the watchdog timer to restart the microcontroller
        }
    }

    delay(2000);         // wait for initializing
    oled.clearDisplay(); // clear display

    oled.setTextSize(1);          // text size
    oled.setTextColor(WHITE);     // text color
    oled.setCursor(0, 10);        // position to display
    oled.println("Starting ..."); // text to display
    oled.display();               // show on OLED
    delay(2000);
}

void printToDisplay(String message)
{
    oled.clearDisplay();
    oled.println(message);
    oled.display();
}

void printMotorSpeed()
{
    String message = "Motor speed: " + String(motorState);
    printToDisplay(message);
}

void writeMotorSpeed(int speed)
{
    // Set the direction
    digitalWrite(MOTOR_CW_PIN, LOW);
    digitalWrite(MOTOR_CCW_PIN, HIGH);

    // Set the speed
    analogWrite(MOTOR_ENABLE_PIN, speed);
}

void updateMotorSpeed()
{
    switch (motorState)
    {
    // Stopped
    case 0:
        writeMotorSpeed(0);
        break;
    // Slow
    case 1:
        writeMotorSpeed(100);
        break;
    // Medium
    case 2:
        writeMotorSpeed(200);
        break;
    // Fast
    case 3:
```

```
        writeMotorSpeed(255);
        break;
    }
}

void increaseMotorState()
{
    if (motorState < 3)
    {
        motorState += 1;
    }
}

void decreaseMotorState()
{
    if (motorState > 0)
    {
        motorState -= 1;
    }
}

void updateMotorSpeedControls()
{
    if (speedUpButton.pressed)
    {
        increaseMotorState();
    }

    if (speedDownButton.pressed)
    {
        decreaseMotorState();
    }
}

void onSpeedUpButtonPressed()
{
    speedUpButton.updateInterrupt();
}

void onSpeedDownButtonPressed()
{
    speedDownButton.updateInterrupt();
}

////////////////////////////////////////////////////////
//////////////////// Extra Logic ///////////////////////
////////////////////////////////////////////////////////

Timer buttonHoldTimer;

void updateButtonHolds()
{
    if (!speedUpButton.state && !speedDownButton.state)
    {
        buttonHoldTimer.reset();
    }

    if (buttonHoldTimer.isFinished(1000))
    {
        if (speedUpButton.state)
        {
            increaseMotorState();
        }

        if (speedDownButton.state)
        {
            decreaseMotorState();
        }

        buttonHoldTimer.reset();
    }
}
```

```
//////////////////////////////////////////////////////

Timer activeTime;

void printActiveTime()
{
    String message = "Active time: " + String(activeTime.getElapsedTime());
    printToDisplay(message);
}

void updateActiveTime()
{
    if (motorState == 0)
    {
        activeTime.reset();
    }
}

//////////////////////////////////////////////////////

int oledSequenceStep = 0;
Timer oledSequenceTimer;

void updateOledTimer()
{
    if (oledSequenceTimer.isFinished(1000))
    {
        oledSequenceStep += 1;
        oledSequenceTimer.reset();
    }
}

void updateOledScreen()
{

    switch (oledSequenceStep)
    {
    case 0:
        printMotorSpeed();
        break;
    case 1:
        printActiveTime();
        break;
    default:
        oledSequenceStep = 0;
        break;
    }
}

//////////////////////////////////////////////////////
//////////////////////////////////////////////////////
//////////////////////////////////////////////////////

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

    setupOled();

    pinMode(MOTOR_CCW_PIN, OUTPUT);
    pinMode(MOTOR_CW_PIN, OUTPUT);
    pinMode(MOTOR_ENABLE_PIN, OUTPUT);

    speedUpButton.pin = 3;
    speedUpButton.pulldown = true;
    pinMode(speedUpButton.pin, INPUT);

    speedDownButton.pin = 2;
    speedDownButton.pulldown = true;
    pinMode(speedDownButton.pin, INPUT);

    attachInterrupt(digitalPinToInterrupt(speedUpButton.pin),
    onSpeedUpButtonPressed, FALLING);
```

```
    attachInterrupt(digitalPinToInterrupt(speedDownButton.pin),
    onSpeedDownButtonPressed, FALLING);
}

void loop()
{
    speedUpButton.update();
    speedDownButton.update();

    updateMotorSpeed();
    updateButtonHolds();
    updateActiveTime();
    updateOledTimer();
    updateOledScreen();

    printMotorSpeed();
}
```