Stopur med PC-kommunikation

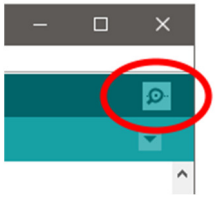
Arduino-eksperiment	130130	*Stikord*	Flere heltalstyper, Serial, millis(), lokale variabler, funktioner	
Version	2018-05-18 / HS	*Niveau*	Grundkursus, modul 5	*p.* 1/4

Det lærer du:

- Send tekst og tal til din PC
- Tag tid med ét millisekunds nøjagtighed
- Lav dine egne funktioner i programmet

- Vælg baud-rate til din kommunikation

1 – Seriel kommunikation



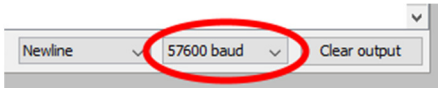
Monitor-vinduet

Arduinos IDE (programmet på PC'en) kan åbne et vindue til kommunikation med Arduinoen gennem USB-kablet. Det åbnes med knappen oppe til højre, som ses på billedet her:

Kommunikationen via USB sker én bit ad gangen, hvilket kaldes *seriel kommunikation*.

Nederst i det nye vindue (*monitor*-vinduet) findes en menu, som styrer hastigheden for kommunikationen mellem Arduino og PC. I eksemplet nedenfor er den sat til "57600 baud". Baud udtales "bo" og 10 baud svarer groft sagt til 1 bogstav pr. sekund.

Der er ikke nogen grund til at ændre denne indstilling lige nu – men husk talværdien, den skal bruges i programmet.



Demonstration af seriel kommunikation

Indtast programmet herunder og kør det:

```
void setup() {
  Serial.begin(57600);
}

void loop() {
  static int N=0;      // Start fra 0
  Serial.println(N);   // Skriv ud
  N++;                 // Tæl op med 1
  delay(250);          // Vent lidt
}
```

Der er proppet en del nye ting ind på disse få linjer!

Kommunikationen varetages af objektet `Serial`, som behandles i værktøjs-boksen ovenfor til højre.

I `setup()` fortælles objektet, hvilken hastighed, det skal kommunikere med ved hjælp af metoden `begin()`. Længere nede anvendes metoden `println()` til at udskrive værdien af variablen `N`.

(`Serial` har også en metode `print()`, som ikke laver linjeskift. Den kan bruges til at skrive flere ting på samme linje.)

`N` er oprettet som en *statisk* variabel (se boks til højre).

Udskriften i monitorvinduet skulle gerne være en stribe tal fra 0 og opefter.

Du kunne også have oprettet `N` som en global variabel (udenfor `setup()` og `loop()`-funktionerne). Så ville værdien også være bevaret for hvert gennemløb af `loop()`. Men man taber hurtigt overblikket i lidt større programmer ved brug af globale variable.

Værktøj: Objektet Serial

Eksempel:

```
Serial.begin(57600);
Serial.println("Hej!");
```

Disse linjer åbner kommunikationen og udskriver teksten `Hej!` (samt et linjeskift) i monitorvinduet.

Forklaring:

Når du programmerer i Arduino IDE'et, er der allerede lavet en del arbejde – der er f.eks. oprettet et såkaldt *objekt* med navnet `Serial`.

Objekter kan stille forskellige *metoder* til rådighed, som f.eks. `begin()` og `println()`.

For at anvende disse metoder, skal de kaldes med deres fulde navn, som indledes med objektets navn med et punktum bagefter, f.eks. `Serial.begin`

Værktøj: Forskellige slags lokale variabler

Eksempel:

```
void loop() {
  int j=7;
  static int N=0;
  N++;
  // Og så videre…
}
```

Forklaring:

Her oprettes to *lokale* variabler `j` og `N`. Det vil sige, at de kun kan bruges indenfor de to krøllede parenteser { } i `loop()`-funktionen.
`j` tildeles værdien 7, *hver gang* `loop()` udføres.
`N` tildeles værdien 0, *første gang* `loop()` udføres; dens værdi bevares fra gang til gang. Det kaldes en *statisk* lokal variabel.

2 – Hvad er klokken?

Det indbyggede ur

Som navnet på denne vejledning antyder, skulle du gerne til sidst have lavet et stopur – så vi må hellere se på, hvordan en Arduino holder styr på tiden.

Der findes en indbygget funktion `millis()`, som angiver, hvor mange millisekunder, der er gået, siden Arduinoen blev startet. Datatypen, som `millis()` returnerer, er en `unsigned long` (se værktøjs-boks til højre).

For at finde en tidsforskel, skal du kende start- og slut-tidspunkterne, som simpelthen kan trækkes fra hinanden. Resultatet bliver i millisekunder.

Værktøj: STORE hele tal

Eksempel:

```
long bigNumber = -1000000000;
unsigned long gross = 1234567890;
```

Oprettelse af to heltalsvariabler, som kan rumme **meget** større tal end `int` og `word` kan.

Forklaring:

De to datatyper `long` og `unsigned long` fylder begge 4 bytes. De rummer disse talintervaller:

`long`:	-2147483648	til	2147483647
`unsigned long`:	0	til	4294967295

Et alternativ til delay()

Når du indsætter en pause i programmet (f.eks. ved at skrive `delay(250);`), blokeres al programafvikling, indtil tiden er gået. Det kan somme tider være acceptabelt – andre gange vil det være nødvendigt at kunne lave noget andet, mens vi venter.

I stedet for at bruge `delay()` kan du gemme starttidspunktet i en variabel, og derefter – for hvert gennemløb af `loop()` – undersøge, om tiden er gået. Se programskitse herunder:

```
const int interval = 75;              // Vi vil gentage noget med 75 ms intervaller

void loop() {
  static unsigned long lastTime=0;    // Opret variabel til at gemme tiden i

  if (millis()-lastTime>interval) {   // Er tiden gået?
    lastTime += interval;             // - så opdaterer vi variablen lastTime
    // Her skriver vi det, som skal ske for hver 75 millisekunder
  }
  // Linjer her udføres, uanset om tiden er gået eller ej
}
```

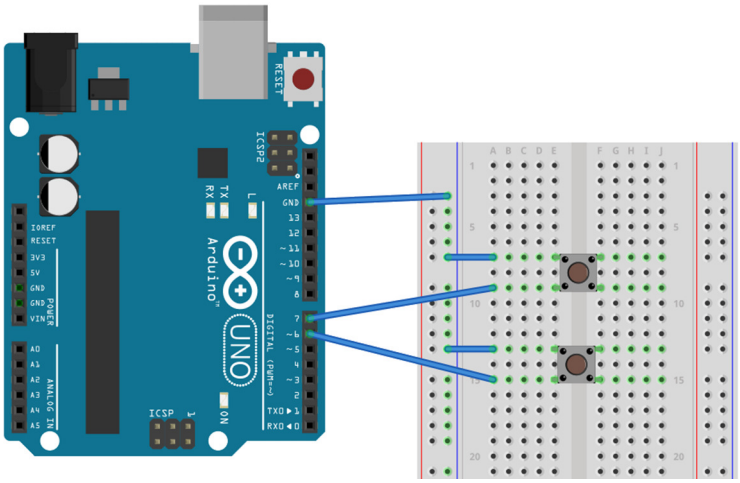
Udfordring 1

Gem programmet under et nyt navn og skriv det om til at benytte ovenstående teknikker. Der skal stadigvæk skrives en række tal i monitorvinduet, som tælles op med én for hvert kvarte sekund (250 ms) – men du må ikke bruge funktionen `delay()`.

3 – Tilslut to kontakter på et breadboard

Opstilling

Du skal bruge to kontakter, som forbindes mellem GND og hhv. ben 6 og ben 7.



Programmering

Definér konstanter for de benyttede ben:

```
const byte pinStart = 6;
const byte pinStop = 7;
```

I `setup()` skal du konfigurere de to ben til at være *indgange* med *pull-up*.

(Ligesom i 130115 Tænd med en kontakt.)

4 – Programmering af stopuret

Lav dine egne funktioner

Du har indtil nu brugt nogle funktioner som `digitalWrite()` og `delay()`, som er indbygget i systemet. Ja, faktisk er `setup()` og `loop()` også funktioner.

Du kan også lave dine egne funktioner, som sidenhen kan bruges (*kaldes*) andre steder i programmet. Se værktøjsboksen til højre.

Der er primært to grunde til at skrive en funktion:

- Hvis funktionen kaldes flere steder i programmet, spares der plads i hukommelsen.
- Det giver overblik at samle en række operationer i en funktion med et fornuftigt navn.

Det er den sidste grund, der spiller ind i dette projekt.

Værktøj: Funktioner

Eksempel:

```
void writeSum(int a, int b) {
  Serial.println(a+b);
}
```

Her defineres en funktion ved navn `writeSum`, som sidenhen kan kaldes. F.eks. således:

```
writeSum(5,7);
```

Dette vil udskrive "12" i monitorvinduet på PC'en.

Forklaring:

```
A B ( C ) {
  D
}
```

A Typen på en evt. returneret værdi – her returneres *ingenting*, det noteres `void`

B Navnet på funktionen – det bestemmer du.

C En liste med evt. parametre. Deres type skal angives. Her oprettes to heltals-parametre med navnene `a` hhv. `b`. (Listen kan være tom.)

D De programlinjer, som skal udføres, når funktionen kaldes.

Hold styr på tastetrykkene

Logikken i stopuret kan opskrives således:

> Hvis tidtagningen ***ikke*** er startet, skal et tryk på *Start*-knappen starte tidtagningen – ellers ikke.
>
> Hvis tidtagningen ***er*** startet, skal et tryk på *Stop*-knappen standse tidtagningen – ellers ikke.

Brug en logisk variabel til at holde styr på, om tidtagningen er i gang eller ej.

Start- og stop-tiderne skal gemmes i to variabler af typen `unsigned long`.

Bogholderiet omkring knapper og tidtagning vil vi samle i en funktion `checkButtons()`, som skal kaldes for hvert gennemløb af `loop()`. Disse tanker leder til følgende programstump:

```
unsigned long startTime;
unsigned long stopTime;
bool running=false;

void checkButtons() {
  if (running) {
    if (digitalRead(pinStop)==LOW) {  // Skal vi stoppe?
      stopTime=millis();
      running=false;
    }
  } else {
    if (digitalRead(pinStart)==LOW) { // Skal vi starte?
      startTime=millis();
      running=true;
    }
  }
}
```

Selve `loop()`-funktionen kan nu reduceres til en enkelt linje:

```
void loop() {
  checkButtons();
}
```

Udfordring 2

I de ovenstående skitser har vi ikke fået noget resultat udskrevet! Start på et nyt program og gør det færdigt:

Husk igen at kalde `Serial.begin()` i `setup()`, hvor du også konfigurerer indgangene.

Tilføj et par linjer til funktionen `checkButtons()`, så programmet udskriver teksten "Go!" ved tryk på *Start*-knappen, og tidsforskellen `(stopTime-startTime)` ved tryk på *Stop*-knappen.

5 – Læselige udskrifter

Fra millisekunder til timer, minutter, sekunder og decimaler

Hvis tidtagningen strækker sig over mere end nogle få sekunder, bliver det ikke særlig overskueligt at få oplyst resultatet i millisekunder.

Hvis i stedet tidsforskellen `stopTime-startTime` bruges i et kald af nedenstående funktion, kaldet `outputTimeDifference()`, får du splittet det store antal millisekunder op i hele timer, hele minutter, hele sekunder og hundrededele af sekunder. Undervejs benyttes brøkstregen `/` til division af hele tal – det resulterer i en såkaldt *heltals-division*, som gennemgås i værktøjs-boksen til højre.

Værktøj: Heltals-division

Eksempel:

```
int x = 37;
int y = x / 10;
int z = x - y*10;
```

Variablen `y` har nu værdien 3 og `z` har værdien 7

Forklaring:

Heltal kan ikke have decimaler – de skæres bare af. Så `y` bliver ikke til 3,7 men bare 3 .

`z` opsamler *resten* efter divisionen.

```
void outputTimeDifference(unsigned long theTime) {
  word hh, mm, ss, dd;

  hh = theTime / 3600000;     // hele timer (1 time = 60*60*1000 millisekunder)
  theTime -= hh*3600000;
  mm = theTime / 60000;       // hele minutter
  theTime -= mm*60000;
  ss = theTime / 1000;        // hele sekunder
  theTime -= ss*1000;
  dd = (theTime / 10);        // hundrededele sekund
  printTime(hh, mm, ss, dd);
}
```

Funktionen `outputTimeDifference()` kalder nederst en ny funktion `printTime()`. Den skal du selv skrive. Den skal have følgende struktur:

```
void printTime(byte hh, byte mm, byte ss, byte dd) {
  // osv.
}
```

I denne funktion får du brug for `Serial.print()`, hvis du vil have resultatet udskrevet på én linje. (Til sidst kan du så bruge `Serial.println()` for at få et linjeskift.)

Udfordring 3

Skriv programmet om, så udskriften bliver "pæn". 443452 millisekunder skal blive til `00:07:23.45`

Gem programmet – du kan få brug for disse funktioner senere.

Løbende visning af resultatet

På et normalt stopur vises tiden, *mens* tidtagningen foregår. Det kan vi også gøre (vi skal blot acceptere, at det giver en ny linje pr. udlæsning).

For at opnå dette, kan du kombinere teknikken fra *Udfordring 1* og *Udfordring 3*.

Det giver ingen mening at opdatere tiden hurtigere, end øjet kan opfatte; vælg f.eks. et opdateringsinterval på 93 ms. (Hvis du vælger et rundt tal, kan resultaterne se ud, som om den sidste decimal er "frosset fast".)

Udfordring 4

Skriv programmet om, så du får udlæst tiden løbende. Brug igen funktionen `outputTimeDifference()`.

Tiden fra *Start* til *nu* findes som `millis()-startTime`.