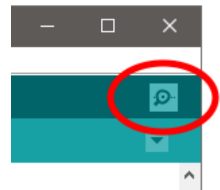
Stopwatch met seriële communicatie

Arduino-experiment 130130-EN	*Trefwoorden* Meer integertypes, serieel, millis(), lokale variabelen, functies	
Versie 2018-07-04 / HS	*Niveau* Basiscursus, module 5	*p.* 1/5

Wat je leert:

- Stuur tekst en getallen naar je PC
- Registreer tijd op de milliseconde nauwkeurig
- Schrijf je eigen functies in het programma
- Selecteer de baud-rate voor je communicatie

1 – Seriële communicatie



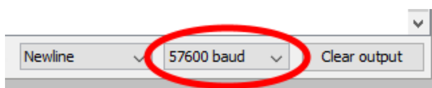
Het monitorvenster

De Arduino IDE (het programma op de PC) kan een venster openen voor communicatie met de Arduino via de USB-kabel. Dit gebeurt door op de knop rechts te klikken:

De communicatie via USB wordt bit voor bit uitgevoerd, wat *seriële* communicatie wordt genoemd.

Onderaan het nieuwe venster (het *monitorvenster*) vind je een menu om de communicatiesnelheid tussen de Arduino en de PC te regelen. In het onderstaande voorbeeld is “57600 baud” geselecteerd. 10 baud komt ongeveer overeen met 1 letter per seconde.

Er is voorlopig geen reden om deze instelling te wijzigen – maar onthoud het nummer, het wordt gebruikt in het programma.



Demonstratie van seriële communicatie

Voer het volgende programma in en draai het:

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

void loop() {
  static int N=0;      // Start from 0
  Serial.println(N);   // Print out
  N++;                 // Increment by 1
  delay(250);          // Pause
}
```

Er zijn nogal wat nieuwe dingen in deze paar regels geperst!

> ***Toolbox:*** Het object Serial
>
> *Voorbeeld:*
>
> ```
> Serial.begin(57600);
> Serial.println(”Hi!”);
> ```
>
> Deze regels openen de communicatie en geven de tekst `Hi!` (plus een nieuwe regel) weer in het monitorvenster.
>
> *Uitleg:*
>
> Bij het programmeren in de Arduino IDE is al veel werk verricht – zo is er bijvoorbeeld een zogenaamd *object* met de naam `Serial` aanwezig.
>
> Objecten kunnen verschillende *methoden* beschikbaar maken, zoals bijvoorbeeld `begin()` en `println()`.
>
> Om deze methoden te gebruiken, worden ze aangeroepen met hun volledige naam, geïnitieerd met de naam van het object, gevolgd door een punt zoals in `Serial.begin`

De communicatie wordt afgehandeld door het object Serial zoals uitgelegd in de toolbox rechtsboven.

In `setup()`, wordt het object verteld welke snelheid te gebruiken in de communicatie via de parameter `begin()`.

Verderop wordt de methode `println()` gebruikt voor het uitvoeren van de waarde van de variabele `N`.

(`Serial` heeft ook de methode `print()` die geen nieuwe regel toevoegt. Het is nuttig voor het uitvoeren van meer dan één ding op dezelfde regel.)

`N` wordt aangemaakt als een *statische* variabele (zie toolbox rechts).

De output in het monitorvenster moet een kolom met getallen van 0 en hoger zijn.

Je had ook `N` kunnen aanmaken als globale variabele (buiten de `setup()`- en `loop()`–functies). Dan wordt de waarde nog steeds gehandhaafd voor elke rondgang in `loop()`. Maar je kunt in grotere programma's snel de globale variabelen uit het oog verliezen.

2 – Hoe laat is het?

De ingebouwde klok

Zoals de naam van deze handleiding al aangeeft, zou je aan het eind een stopwatch gemaakt moeten hebben – laten we eens kijken hoe een Arduino de tijd bijhoudt.

Er is een ingebouwde functie `millis()` die het aantal milliseconden teruggeeft dat sinds de inschakeling is gepasseerd. Het gegevenstype dat per `millis()` wordt teruggegeven is een `unsigned long` (zie toolbox rechts).

Om een tijdsinterval te vinden moet je de begin- en eindtijden kennen en die vervolgens simpelweg van elkaar aftrekken. Het resultaat wordt in milliseconden gegeven.

Toolbox: GROTE gehele getallen

Voorbeeld:

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

Twee integer-variabelen maken die **veel** grotere getallen kunnen bevatten dan `int` en `word`.

Uitleg:

De twee datatypen `long` en `unsigned long` gebruiken allebei 4 bytes. Ze gebruiken de volgende nummerreeksen:

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

Een alternatief voor delay()

Wanneer je een pauze in het programma invoert (bijvoorbeeld door `delay(250);`), wordt alle programma-uitvoering geblokkeerd totdat de vertraging voorbij is. Dit kan soms acceptabel zijn – andere keren kan het nodig zijn om tijdens het wachten taken uit te voeren.

In plaats van vertraging te gebruiken, kun je de starttijd in een variabele opslaan en dan – voor elke uitvoering van `loop()` – controleren of de tijd op is. Zie het stukje code hieronder:

```
const int interval = 75;
// Repeat something at 75 ms intervals

void loop() {
  static unsigned long lastTime=0;
// Variable for storing start time
  if (millis()-lastTime>interval) {
// Are we done now?
    lastTime += interval;
// - then update variable lastTime
    // Code to be executed every 75
milliseconds goes here
  }
  // Lines here are executed regardless
of the time
}
```

Toolbox: Verschillende soorten lokale variabelen

Voorbeeld:

```
void loop() {
  int j=7;
  static int N=0;
  N++;
  // Etc. …
}
```

Uitleg:

Hier maken we twee lokale variabelen aan, `j` en `N`. Dat wil zeggen dat ze alleen kunnen worden gebruikt binnen de accolades { } in de `loop()`-functie.
`j` krijgt de waarde 7, *elke keer dat* `loop()` draait.
`N` krijgt de waarde 0, *de eerste keer dat* `loop()` draait; de waarde wordt van tijd tot tijd opgeslagen. Dit wordt een *statische* lokale variabele genoemd.

Uitdaging 1

Sla het programma op met een nieuwe naam en herschrijf het zodat het de hierboven beschreven techniek gebruikt.

Het moet nog steeds een getal in het monitorvenster schrijven, dat elk kwart van een seconde (250 ms) toeneemt – maar je mag de `delay()`-functie niet gebruiken.

3 – Twee schakelaars op een breadboard aansluiten

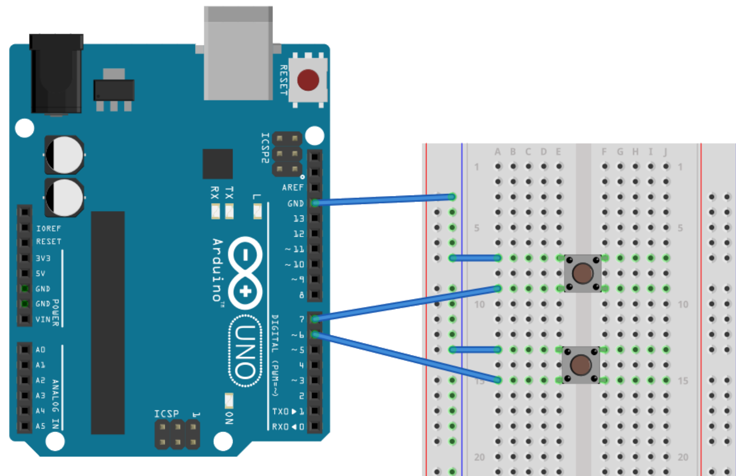
Opstelling
Je hebt twee schakelaars nodig, verbonden tussen GND en pin 6, resp. pin 7.

Programmering
Definieer constanten voor de gebruikte pins:

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

In `setup()`, moet je de twee pins configureren als *inputs* met *pull-up*.

(Zoals in 130115-EN Een schakelaar gebruiken.)



4 – De stopwatch programmeren

Maak je eigen functies
Tot nu toe heb je gebruik gemaakt van functies zoals `digitalWrite()` en `delay()` die in het systeem zijn ingebouwd. En in feite zijn zelfs `setup()` en `loop()` functies.

Je kunt je eigen functies maken die elders in het programma gebruikt *(called* of 'aangeroepen') kunnen worden. Zie de toolbox rechts.

Een functie schrijven doe je voornamelijk om twee redenen:

- Als de functie op meerdere plaatsen in het programma wordt aangeroepen, scheelt dit geheugen.
- Je krijgt meer overzicht door een aantal bewerkingen af te bakenen in een functie met een zinvolle naam.

Vooral die laatste reden is in dit project van toepassing.

Toolbox: Functies

Voorbeeld:

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

Hier definiëren we een functie genaamd `writeSum` die later kan worden aangeroepen. Bijvoorbeeld zo:

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

Deze geeft "12" weer in het monitorvenster op de PC.

Uitleg:

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

A Het type van een mogelijke return-waarde – hier sturen we niets terug, geschreven als `void`

B De naam van de functie – door jou gekozen.

C Een lijst met mogelijke parameters. De typen moeten worden gespecificeerd. Hier hebben we twee integer-parameters, genaamd `a` en `b`. (De lijst kan leeg zijn.)

D Programmaregels die bij het aanroepen van de functie moeten worden uitgevoerd.

Houd de ingedrukte toetsen bij
De stopwatchlogica kan als volgt worden samengevat:

> Als – en alleen als – de timer ***niet*** is gestart, moet de timer worden gestart door op de *Start*-knop te drukken.
>
> Als – en alleen als – de timer ***loopt***, moet de timer worden gestopt door op de *Stop*-knop te drukken.

Gebruik een Booleaanse variabele om bij te houden of de timer loopt of niet.

Start- en stoptijden moeten in variabelen van het type `unsigned long` worden opgeslagen.

We zullen het indrukken van de knoppen en de timing verzamelen in een functie `checkButtons()` die moeten worden aangeroepen voor elke rondgang van de `loop()`. Dit leidt tot het volgende stukje code:

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

void checkButtons() {
  if (running) {
    if (digitalRead(pinStop)==LOW) {   // Are we stopping?
```

```
        stopTime=millis();
        running=false;
      }
    } else {
      if (digitalRead(pinStart)==LOW) { // Are we starting?
        startTime=millis();
        running=true;
      }
    }
  }
```

De `loop()`-functie kan nu worden gereduceerd tot een enkele regel:

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

Uitdaging 2

In de code hierboven wordt geen output gegenereerd! Maak een nieuw dergelijk programma en maak het compleet:

Vergeet niet om nog steeds `Serial.begin()` aan te roepen in `setup()` waar je ook de input pins configureert.

Voeg een paar regels toe aan de `checkButtons()`-functie om het programma de tekst ”Go!” te laten uitvoeren wanneer de *Start*-knop wordt ingedrukt, en voer het tijdsverschil `(stopTime-startTime)` uit wanneer *Stop* wordt ingedrukt.

5 – Leesbare output

Van milliseconden naar uren, minuten, seconden en decimalen

Als je een tijdsinterval van meer dan een paar seconden hebt, is het resultaat al snel niet meer in milliseconden af te lezen.

Als in plaats daarvan het tijdverschil `stopTime-startTime` wordt gebruikt in een aanroep van `outputTimeDifference()` – die we hieronder definiëren – krijg je het grote aantal milliseconden verdeeld in uren, minuten, seconden en honderdsten van seconden. In deze functie wordt de *fraction bar* / gebruikt voor het splitsen van gehele getallen – dit resulteert in een zogenaamde *integersplitsing* die in de toolbox rechts wordt behandeld.

Toolbox: Integersplitsing

Voorbeeld:

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

De variabele `y` is nu gelijk aan 3 en `z` is gelijk aan 7

Uitleg:

Integers hebben geen decimalen – ze worden gewoon afgesneden. Dus `y` wordt geen 3.7 maar slechts 3 .

`z` haalt *de rest van* de splitsing op.

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

  hh = theTime / 3600000;     // whole hours (1 hour = 60*60*1000 milliseconds)
  theTime -= hh*3600000;
  mm = theTime / 60000;       // whole minutes
  theTime -= mm*60000;
  ss = theTime / 1000;        // whole sekonds
  theTime -= ss*1000;
  dd = (theTime / 10);        // hundredths of a second
  printTime(hh, mm, ss, dd);
}
```

De laatste regel van de functie `outputTimeDifference()` roept een andere functie `printTime()` aan. Dit moet je zelf implementeren. Het moet de volgende structuur hebben:

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

In deze functie heb je `Serial.print()` nodig als je het resultaat op één regel wilt uitvoeren. (Uiteindelijk zul je `Serial.println()` gebruiken om een nieuwe regel toe te voegen.)

Uitdaging 3

Herschrijf het programma om "mooie" outputs te maken. 443452 milliseconden moet worden `00:07:23.45`

Sla het programma op – het kan zijn dat je deze functies later nog nodig hebt.

Continue weergave van de resultaten

Op een normale stopwatch wordt de looptijd weergegeven *terwijl* de timing doorloopt. Dat kunnen wij ook (we moeten alleen accepteren dat elk resultaat op een nieuwe regel wordt weergegeven).

Om dit te bereiken moeten we de technieken van *Uitdaging 1* en *Uitdaging 3* combineren.

Er is geen reden om de tijd sneller te updaten dan het oog kan volgen; je kunt er bijvoorbeeld voor kiezen om elke 93 ms te updaten.

Uitdaging 4

Herschrijf het programma om een output met looptijd te maken. Gebruik opnieuw de functie `outputTimeDifference()`.

De tijd van *Start* tot *nu* is te vinden als `millis()-startTime`.

(Als je een "mooie" waarde kiest, zoals 100 ms, lijkt het laatste cijfer achter de komma "vast te zitten".)