Arduino Programming

Theorie Sessie 2

ARTESIS PLANTIJN
HOGESCHOOL ANTWERPEN

- Loops in Arduino code
- C code vs Arduino code
- Seriële Monitor
- Interrupts
- Code Flags
- Arduino code referentie

Loops in Arduino C

algemeen

Loops zijn vrij consistent doorheen
de volledige familie programmeer talen
gebaseerd op “C”

Arduino code loops =
C, C++, C#, Objective C, ...

Loops in Arduino C

“for” lus

```
for ( initialisatie; conditie; actie ) {

     // code binnen de lus

}
```

Indien niet nodig buiten de loop gebruiken we
bij voorkeur “wegwerp variabelen”
= declaratie in het initialisatie gedeelte

de scoop van deze variablen is de loop zelf.

Loops in Arduino C

“for” lus

In hardware toepassingen zullen for-loops
ook soms gebruikt worden
als alternatief voor de “delay()” functie...

```
for ( word t=0; t<100000; t++ ) {

  //doe niets

}
```

Loops in Arduino C

“for” lus

Met name als we slechts een heel korte delay
nodig hebben komt dit voor

Of wanneer er geen hardware timers beschikbaar zijn
in de hardware om zo’n delay() mee te maken

“Timers” zijn de interne klokjes in de MCU
en daar zijn er slechts enkele van…
In de ATMega328p zijn er maar drie !
Deze worden volgend jaar uitvoerig besproken
in de cursus micro-controllers

Loops in Arduino C

“while” lus

```
while ( conditie ) {

      // code binnen de lus

}
```

In hardware schakelingen zullen condities echter veel vaker dan bvb in C# gewoon een bool zijn.
Ook int’s zondermeer kunnen voorkomen…

Hierbij is geldig dat 0 = false en elk ander cijfer = true

Loops in Arduino C

“while” lus

- ```
  bool sensor = true;
  while ( sensor ) { … }
  ```

- ```
  int spanning = 1024;
  while ( spanning ) {
        spanning = analogRead(A1);
  }
  ```

Loops in Arduino C

“do ..while” lus

```
do {
      // code binnen de lus
} while ( conditie ) ;
```

vergeet hier niet dat de do...while
in tegenstelling tot de gewone while
dient afgesloten te worden met een punt-comma

C code vs Arduino code

C code vs Arduino code

Algemene opbouw

Hoe is nu de verhouding
tussen Arduino code
en de originele C programmeertaal ?

We kijken hiervoor éérst even naar C

C code vs Arduino code

Algemene opbouw

```
#include <stdio.h>                    ← bibliotheek die “printf” bevat

int main()                            ← hoofd programma

{

    printf( "Hello World \n " );

}
```

C code vs Arduino code

Algemene opbouw

```
void mijnfunctie()                    ← aanmaken eigen functies
{
    // code van mijn functie
}
```

C code vs Arduino code

Setup() en Loop()

Arduino Code

```
void setup() {

    // setup code

}

void loop() {

    // loop code

}
```

Achterliggende C/C++-Code

```
void arduino_setup() {
        // setup code
}

void arduino_loop() {
    for ( ; ; ) {
        // loop code
    }
}

int main() {
  arduino_setup();
  arduino_loop();
}
```

C code vs Arduino code

printf

In C hebben we de “*printf*”
functie om tekst op het scherm te zetten

Dit gebruiken we niet enkel als user-output
maar ook vaak tijdens de ontwikkeling van ons programma
om meer inzicht te krijgen in de code en voor debugging

Op een Arduino echter hebben we
geen beeldscherm aansluiting…

Hoe kunnen we dan ooit iets zichtbaar maken?

C code vs Arduino code

printf

Ook hier hebben de ontwerpers
van de Arduino aan gedacht !

Als oplossing hebben ze in hiervoor de
“seriële monitor”
geïntegreerd in de hardware en de IDE

Seriële monitor

Seriële Monitor

Wat ?

De ATmega328p microcontroller
heeft geen beeldscherm aansluiting

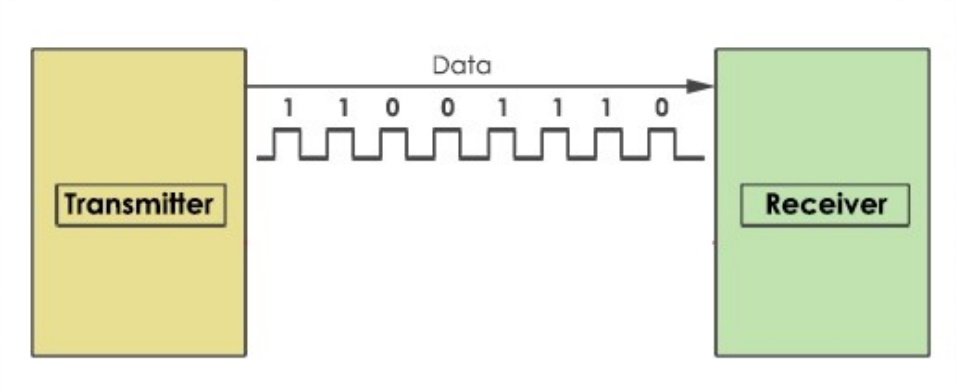
maar heeft wél een seriële communicatie poort

dit zijn 2 draadjes waarover op seriële wijze
bits kunnen gestuurd worden naar de chip
of bits gelezen kunnen worden uit de chip

De Arduino IDE gebruikt deze verbindingen
om ASCII characters
van/naar de microcontroller chip te sturen

Seriële Monitor

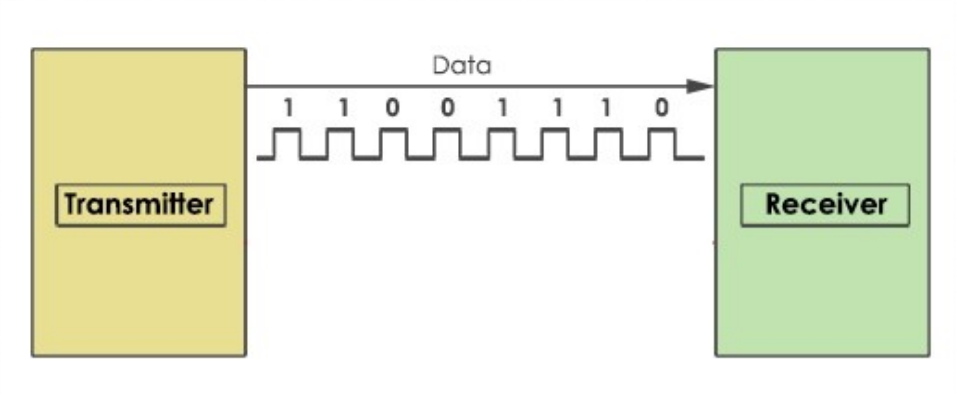
Werkings principe



Het principe is zoals hierboven waar over hetzelfde draadje
de “transmittor” een reeks ééntjes en nulletjes
na elkaar zal verzenden naar de “receiver”

Seriële Monitor

Werkings principe



..en bvb elke 8 bit's die toekomen
zal de receiver omzetten in een "byte"

Daartoe moet de receiver uiteraard weten hoe snel
die ééntjes en nulletjes verzonden worden...

Seriële Monitor

Werkings principe

Stel dat de transmitter

8 x “1” achter elkaar uitzend
hierna 8 x “0”
en hierna terug 8 x “1”

.. maar de receiver luistert maar aan de halve snelheid
dan dat de transmitter deze bits uitstuurt...

Seriële Monitor

Werkings principe

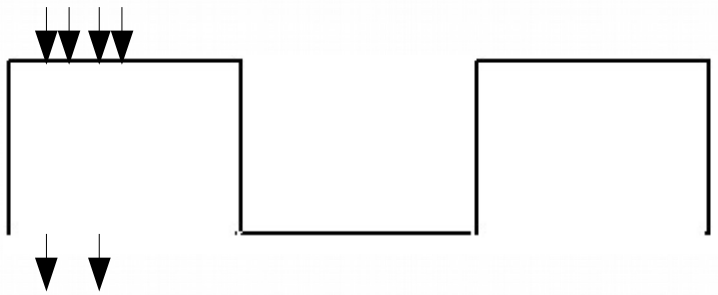
Transmitter → 11111111 00000000 11111111

1 1 1 1 0 0 0 0 1 1 1 1 → Receiver

Seriële Monitor

Werkings principe

Transmitter → 11111111 00000000 11111111



1 1 1 1 0 0 0 0 1 1 1 1 → Receiver

Seriële Monitor

Werkings principe

We zien hier hoe de receiver een totaal ander bit-code ontvangt
dan dat de zendkant verzonden heeft…

Belangrijk bij zo'n seriële communicatie is dus de

transmissie-snelheid

deze wordt uitgedrukt in "BAUD" als eenheid,

men spreekt wanneer men het heeft over de snelheid
dan ook vaak van de "*baud-rate*"

Seriële Monitor

Serial Begin

```
void setup() {

      Serial.begin(9600);

}
```

met het “Serial.begin” commando stellen we in het setup gedeelte van de Arduino code in aan welke snelheid de ATMega328p chip ASCII tekens zal versturen naar onze PC.

Seriële Monitor

Serial println

```
void loop() {

      Serial.println(“Hello World!”);

}
```

met het “Serial.println” commando
versturen we dan uiteindelijk vanuit onze Arduino code
ASCII tekens naar de PC.

Seriële Monitor

Serial print

```
void loop() {

      Serial.print(“Hello ...”);

}
```

Het verschil tussen “print” en “println”
is dat println op het einde van de tekst die we afdrukken
naar de volgende lijn gaat,

daar waar “Serial.print”
de cursor op dezelfde lijn laat staan

Seriële Monitor

Serial print & println

```
void loop() {

      Serial.print(“Hello ”);
      Serial.prinln(“world”);

}
```

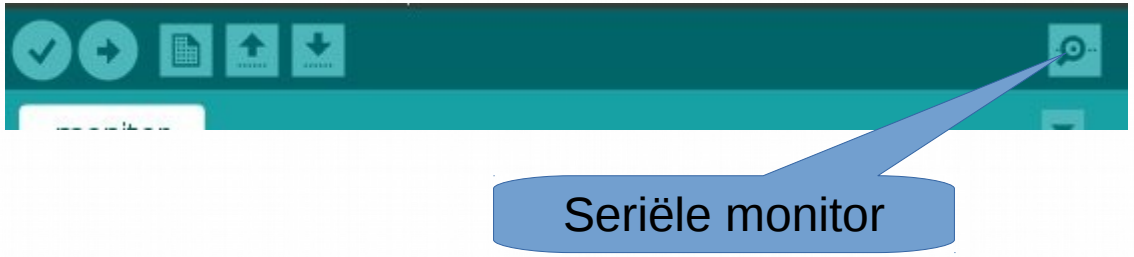
deze twee lijnen samen drukken dus
“Hello world” af op dezelfde lijn

Seriële Monitor

Arduino IDE

Aan de andere uiteinde van het verhaal
hebben we de Seriële monitor van de Arduino IDE

Deze monitor kunnen we openen via het icoontje
rechts zoals hieronder aangegeven:



Seriële Monitor

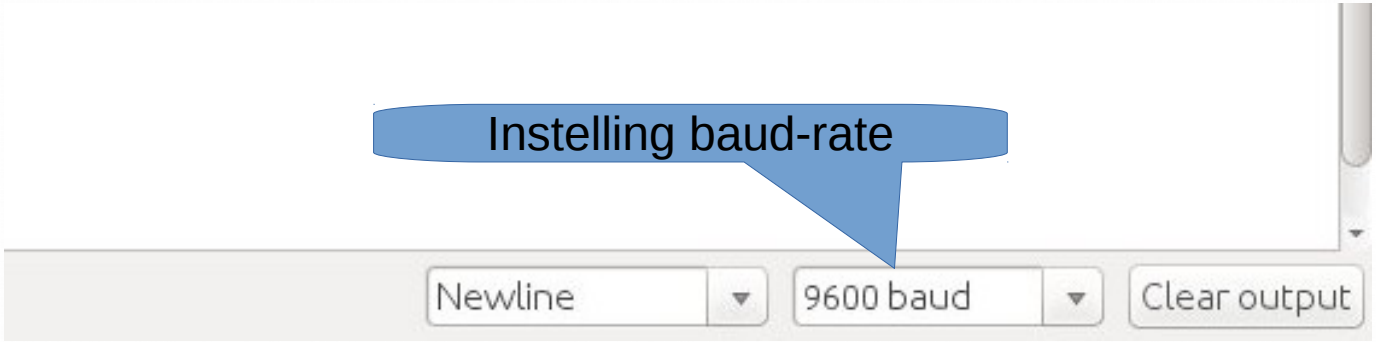
Arduino IDE

Als je daarop klikt krijg je een
nieuw venster zoals hieronder:

Seriële Monitor

Arduino IDE

Onderaan in dit venster kan je de snelheid selecteren waarmee de IDE luistert naar de bits die binnenkomen van de Arduino Hardware



Seriële Monitor

Arduino IDE

Merk op dat je hier niet om het even
welk getal kan intikken maar enkel kan kiezen
uit één van de volgende presets:

300, 1200, 2400,
9600, 19200,
38400, 57600, 74880,
115200, 230400, 250000,
500000, 1000000, 2000000

Seriële Monitor

Arduino IDE

Volgende baud-rates uit deze reeks worden hierbij het vaakst gebruikt:

300, **1200**, 2400,
9600, **19200**,
38400, 57600, 74880,
115200, 230400, 250000,
500000, 1000000, 2000000

Seriële Monitor

Arduino IDE

Als we nu onze “Hello world” uit het println voorbeeld runnen krijgen we de volgende output:

Seriële Monitor

Seriële Monitor

Seriële Monitor

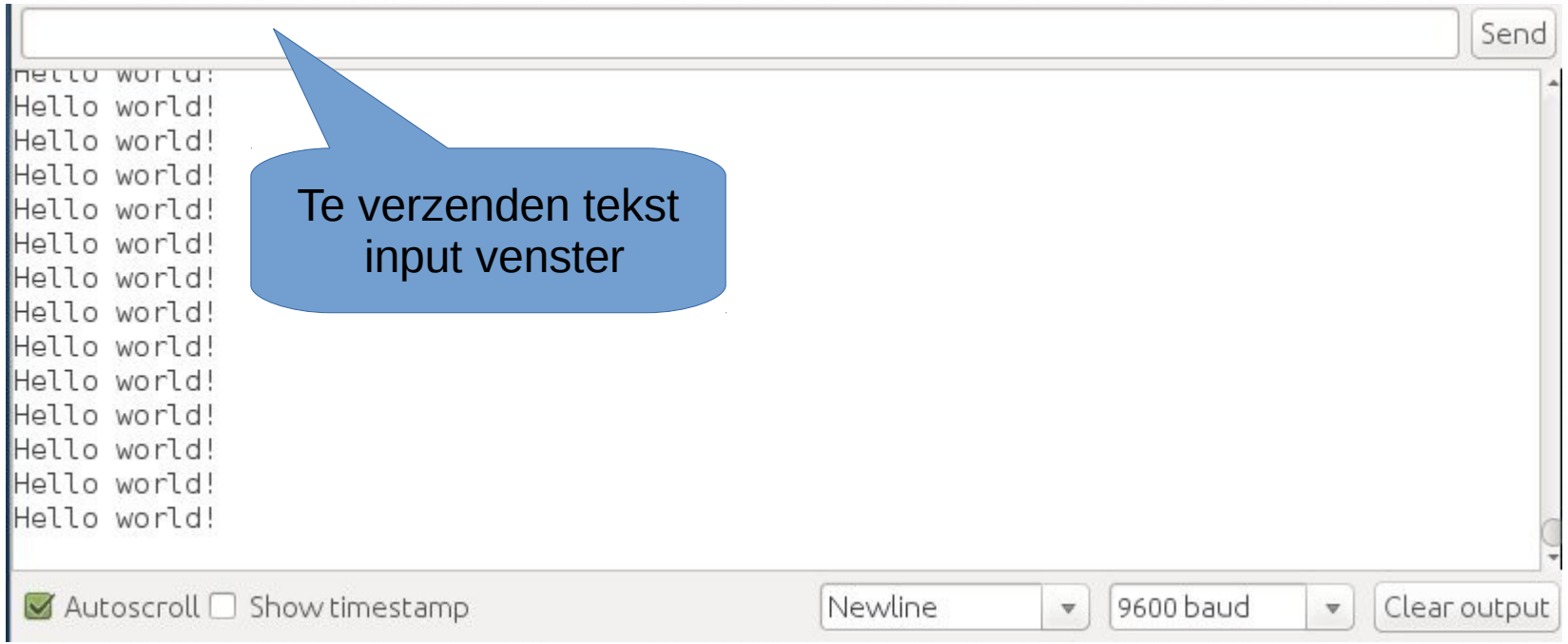
Input sturen naar de Arduino

Het hele proces kan ook gebeuren
in de omgekeerde richting

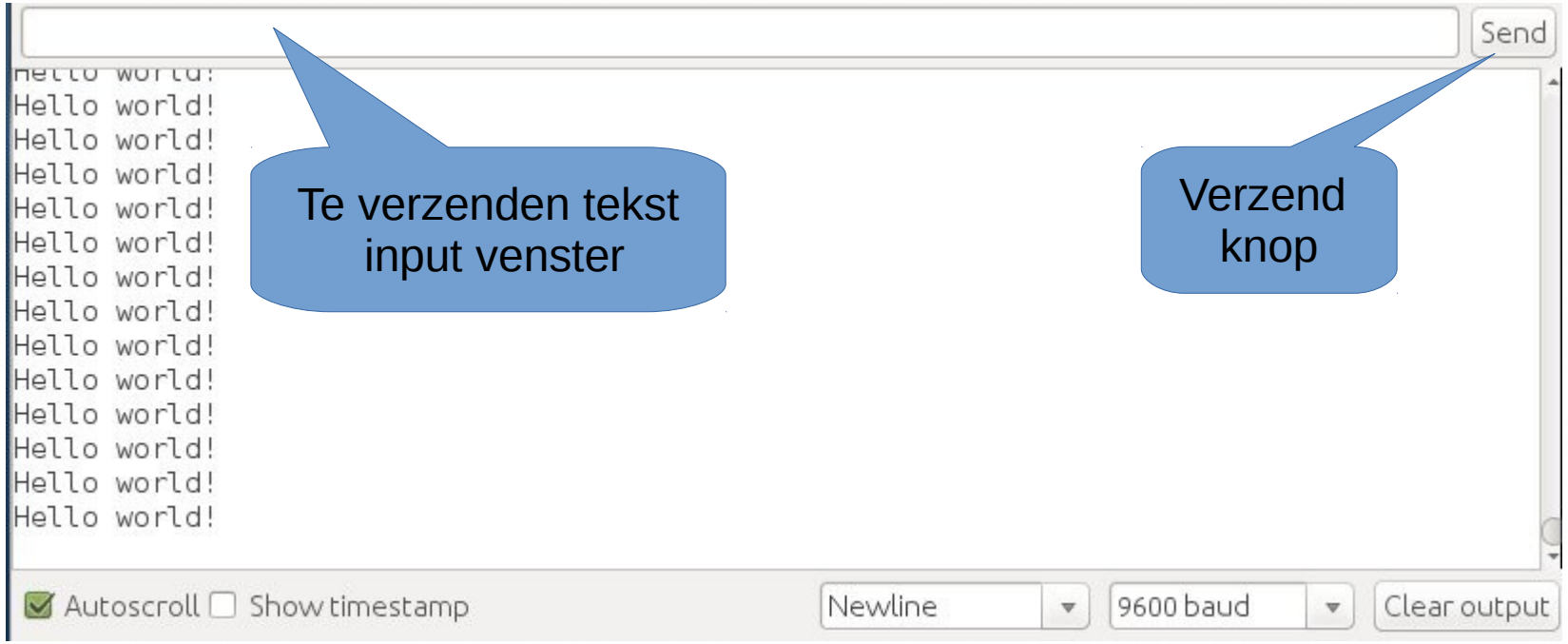
Via de seriële monitor in de IDE
kunnen we ASCII inlezen

en vervolgens de bits
via de seriële communicatie lijnen
verzenden naar de Arduino.

Seriële Monitor



Seriële Monitor



Seriële Monitor

Input sturen naar de Arduino

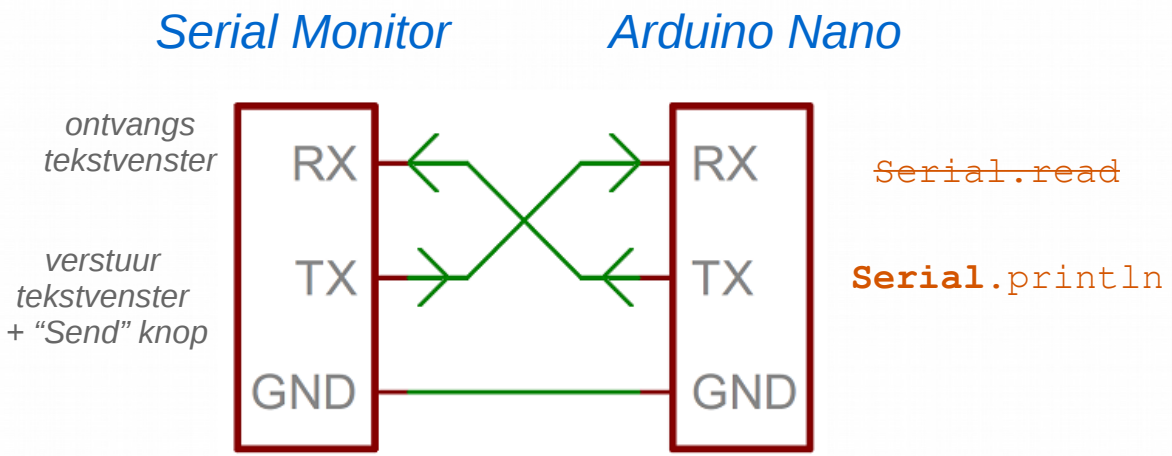
De baud-rate die we instellen in de Arduino IDE
is nu eveneens ook de snelheid
waarmee de ASCII tekens
naar de hardware worden verzonden

Aan de kant van de Arduino code is
de snelheid die we opgaven met Serial.begin()
nu eveneens ook de snelheid waarmee
De ATMega chip de bits *vanuit* de IDE zal lezen

Seriële Monitor

Input sturen naar de Arduino

De volledig communicatie verbinding tussen IDE en Arduino Nano hardware ziet er bijgevolg zo uit :



Seriële Monitor

Input sturen naar de Arduino

Om de Arduino commando's
om deze gegevenstroom te lezen
goed te kunnen begrijpen
moeten we weten wat een"input buffer" is
en weten wat de exacte werking er van is

→ *De bespreking van deze zaken komen in een van de volgende theorie sessies aan bod*

Interrupts

Interrupts

Wat ?

in microcontroller toepassingen is het typisch
dat het verloop van onze code
bepaald wordt door tal van events.

Denk aan drukknop, analoge spanningen ...

interrupts zijn een *speciale versie*
van dergelijke events

Interrupts

Wat ?

Bij een interrupt wordt de code van onze loop()
onderbrokken *ongeacht welke lijn de uitvoering zit*

en springt de code uitvoering onmiddelijk naar een
specifiek stukje code die we “ISR” noemen.

Als alle code die in deze ISR staat uitgevoerd is
dan keert de programma uitvoering terug
naar de plaats in de loop()
waar de code onderbroken was.

Interrupts

ISR

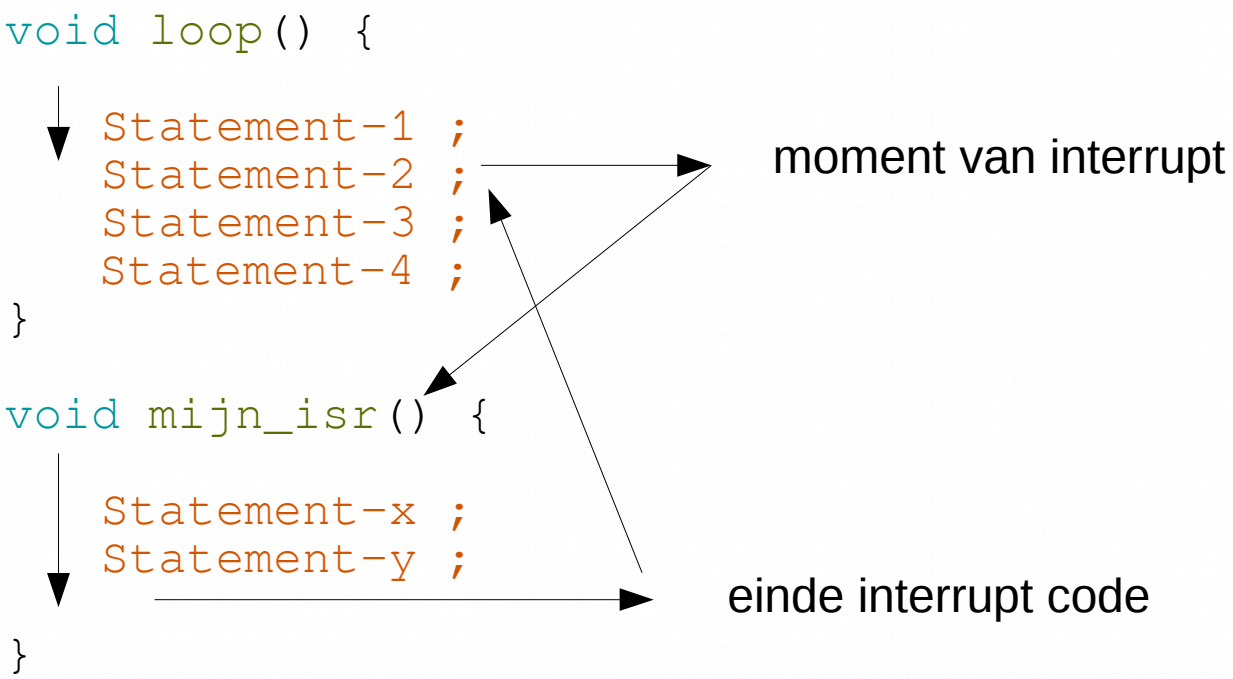
De afkorting ISR staat voor

“ **I**nterrupt **S**ub **R**outine ”

In de praktijk van de Arduino code
is dit een door ons geschreven C-functie

Interrupts

Code Flow



Interrupts

attachInterrupt

Hoe nu kan een extern event
dit soort code onderbreking veroorzaken?

Hiertoe moeten we een verbinding creëren tussen:

- een hardware pin van de arduino,
- de manier van detectie,
- en de ISR-code die moet aangeroepen worden.

Interrupts

attachInterrupt

Dit doen we met behulp van het “attachInterrupt” commando en wel op de volgende manier:

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

pin	=	de hardware pin op de Arduino Nano
ISR	=	de naam van de functie van onze ISR
mode	=	de manier van detectie

Interrupts

digitalPinToInterrupt

er zijn echter wél beperking aan wélke pinnen
we kunnen gebruiken voor zo'n interrupt:

- Op de Arduino Uno en Nano kan dit ***énkel met pin D2 en D3***
- Op de Arduino Micro kan dit met pinnen D0, 1, 2, 3 en 7.
- Bij de Arduino Mega kan dit met pin D2, 3, 18, 19 20 en 21.

Interrupts

mode

Met de manier van detectie bedoelen we
welke actie er nu net voor een interrupt zal zorgen.

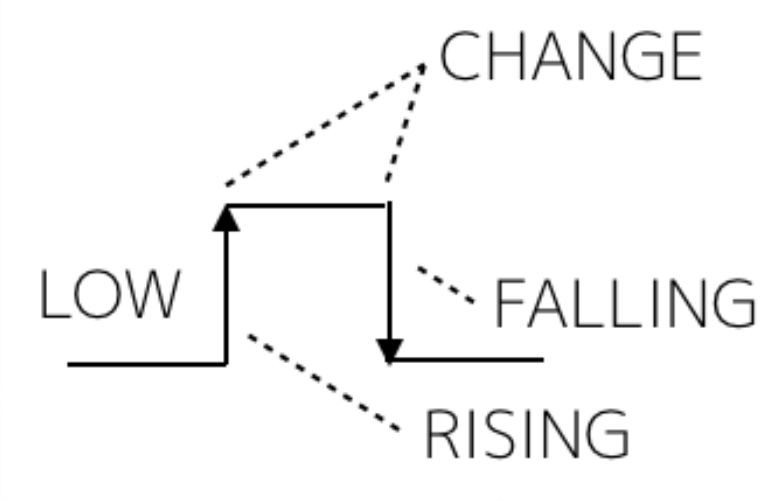
De mogelijkheden zijn :

- LOW → wanneer de pin waarde “0” is
- HIGH → wanneer de pin waarde “1” is
- CHANGE → wanneer de pin van waarde omklapt
- RISING → exact op het moment dat de pin waarde van 0 naar 1 overgaat
- FALLING → exact op het moment dat de pin waarde van 1 naar 0 overgaat

Interrupts

mode

manier van detectie

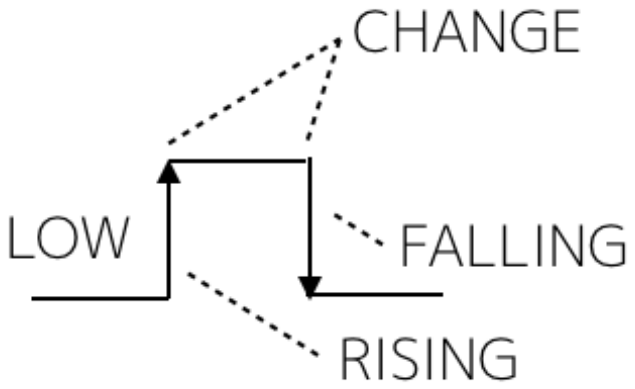


Interrupts

mode

Wat is dan het verschil tussen :

- RISING en HIGH
- FALLING en LOW ?



Interrupts

mode

- RISING → exact op het moment dat de pin waarde van 0 naar 1 overgaat

dat wil zeggen dat indien het signaal hoog of “1” wordt én hoog blijft dit slechts één overgang van 0→1 teweegbrengt.

- HIGH → wanneer de pin waarde “1” is

dat wil zeggen dat indien de code van de ISR is uitgevoerd en terug verder gaat, de interrupt ingang *nog steeds* “1” is en dus de ISR *opnieuw* zal worden aangeroepen en dit zolang het niveau hoog of “1” blijft...

Interrupts

mode

betreft LOW en FALLING :

Dit is uiteraard een identisch verschil
maar dan bij het LOW niveau of “0”

Interrupts

voorbeeld:

```
volatile bool knop = false;


void setup() {
  attachInterrupt(digitalPinToInterrupt(2), knopisr, RISING);
}

void loop() {
    digitalWrite( 7, knop );
}

void knopisr() {
    knop = true;
}
```

Interrupts

“Volatile” keyword

Het opmerkzame oog heeft wellicht gemerkt dat in
het code voorbeeld de boolean declaratie vooraf gegaan werdt
door het onbekende keywoord “volatile”

Doordat de gewone code uitvoering onderbroken wordt
hebben we een speciaal soort variabelen nodig
die dit soort code-jumps kan overleven.

Interrupts

“Volatile” keyword

In het bijzonder is dit belangrijk wanneer
de waarde van dezelfde variable zowel in de loop()
als in de ISR functie kan worden aangepast.

De veilige optie is dus om dus steeds
de variabelen die in de ISR gebruikt worden
bij de declaratie als “volatile” te declareren.

Interrupts

“De ideale ISR”

Het zomaar onderbreken van de code uitvoering op een willekeurig moment bevat nog enkele andere gevaren.

Eens de code uitvoering naar de ISR springt
zijn er namelijk delen van de achterliggende C-code
die niet langer correct zullen blijven functioneren
indien deze ISR te lang duurt...

Een mooie vergelijking is die
van iemand die aan het jongleren is :

Interrupts

“De ideale ISR”

We kunnen de gewone code uitvoering van de **loop()** hier beschouwen als het in de lucht gooien en terug opvangen van de ballen door de jongeleur hiernaast.

Als er een interrupt komt wordt deze gewone uitvoering onderbroken,
wat het equivalent is van de jongeleur die even wegloop naar ergens anders om bvb nog enkele extra ballen op te rapen.

Interrupts

“De ideale ISR”

De jongeleur kan, als hij/zij snel genoeg is, dit doen zonder dat de “gewone code uitvoering” in de war loopt.

Eventueel kan de jongeleur nog even de ballen extra hoog gooien om wat meer tijd te hebben, maar hoe dan ook… als deze te lang wacht dan vallen de ballen uiteindelijk op de grond.

Interrupts

“De ideale ISR”

Het Arduino equivalent is dat de uitvoering van bepaalde achterliggende C-code zoals onder andere de functie millis() niet langer correct zal uitgevoerd worden.

Hoe lang de onderbreking mag duren is arbitrair,
want men weet natuurlijk nooit
wanneer exact de interrupt zal komen..

Denk aan de jongeleur:
als de ballen toevallig nog héél hoog waren mag dit lang zijn,
als ze toevallig nét al aan z’n handen waren
dan is dit slechts héél erg kort..

Interrupts

“De ideale ISR”

Een ISR is met andere woorden best **zo kort mogelijk**

Statements zoals “Serial.print” of loops
in een ISR zijn dus uit den boze !

ideaal is een ISR slechts één statement lang

Interrupts

“De ideale ISR”

We kunnen dit doen door een boolean variable te declareren
en als enige statement in een interrupt sub-routine
deze boolean op true te plaatsen

Zo weet onze loop() dat er een interrupt geweest is.

Eventueel, indien nodig, kan dit om meerdere booleans gaan.

Het zetten of wissen van een boolean gaat immers héél snel.

Interrupts

“De ideale ISR”

Andere typische statements die kunnen voorkomen in ISR’s zijn de incrementors ++ en decrementors --:

Bvb: …

```
void ingangs_isr() {
    n++;
}
```

Wees echter niet te snel om dit te gebruiken met een drukknop als bron voor de interrupt..

Bij contactdender krijgt men hierbij immers meer increments dan verwacht...

Code Flags

Code Flags

Wat ?

code vlaggen of “Flags” zijn typisch
booleaanse variabelen om binnen onze code
zaken te signaleren naar andere code gedeeltes

bvb:

```
bool vlag_a = false, vlag_b = false;

if( situatie==deze ){ vlag_b = true; }

…

if( vlag_b ) {

    // in dat geval komt deze code
}
```

Code Flags

Gebruik van vlaggen

In code van microcontroller schakelingen
maar ook in programma's geschreven in Assembler,
worden vlaggen veelvuldig gebruikt.

Het gebruik van vlaggen
kan een anders vrij complexe codeflow
op een vaak veel eenvoudiger manier bereiken

Het gebruik van vlaggen geniet de voorkeur
boven het creëren van verschillende hulp functies
indien dat enigsinds mogelijk is.

Code Flags

Gebruik van vlaggen

vooral wanneer dit creëren van verschillende functies
als énige reden heeft de code flexibiliteit
naar de status van “hyperflexibel” te verhogen

Ja, dit is tegenstrijdig met wat je leerde in C# of PHP…
… maar daar is een goede reden voor !

Microcontrollers hebben slechts enkele kilobytes aan geheugen
Ze kunnen zich bijgevolg vaak niet de vele extra lijnen code
veroorloven wat wel kan in een PC met vele Gbytes...

Arduino code referentie

Arduino code referentie

Wat ?

Niet alle commando’s van de Arduino taal
zullen in deze theorie sessies besproken worden.

Soms zullen toch enkele van die commando’s
opduiken in de programmeer voorbeelden,

of je zal ze nodig hebben om bepaalde
labo opdrachten te kunnen oplossen...

maar ook daar hebben de Arduino ontwerpers
weer een mooie oplossing voor bedacht !

Arduino code referentie

arduino.cc

Op de centrale website van Arduino
staat er een vrij volledig referentie document

met uitleg en code voorbeelden van
de meest voorkomende arduino commando's

Je kan dit vinden op:

https://www.arduino.cc/reference/en/