EMBEDDED SOFTWARE DEVELOPMENT (ESD)

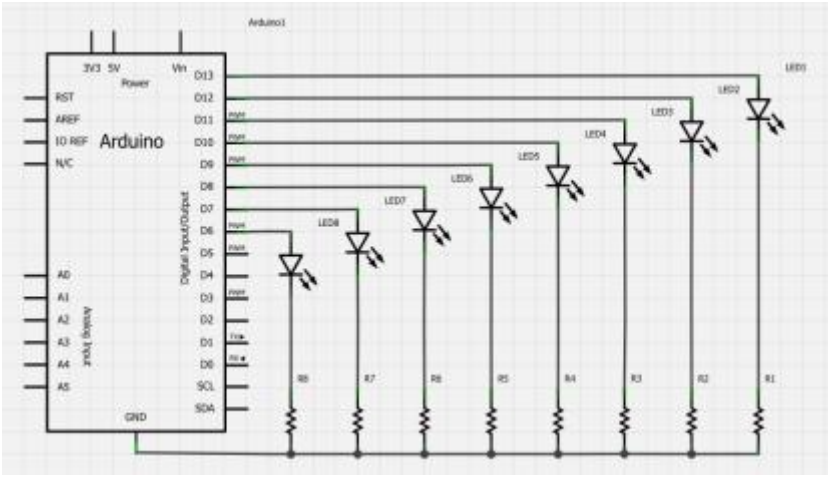
LES 1-2: SERIAL EN LEDPATROON IN CODE (VOORBEREIDING)

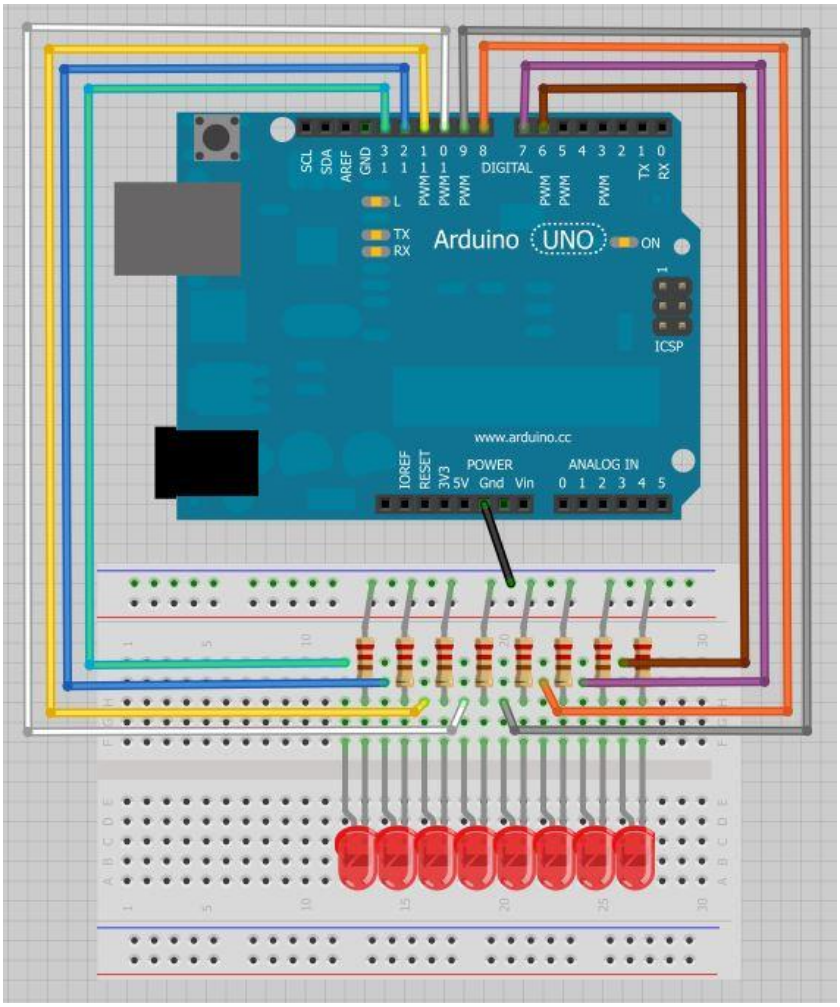
Versie: 7-2-2017 14:11:00

INHOUDSOPGAVE

1 MEERDERE LEDS

1.1 AANSLUITEN MEERDERE LED'S





(de kleur van de ledjes is niet belangrijk)

2 COMMUNICATIE

2.1 INLEIDING

In dit onderdeel gaan we de computer/laptop niet alleen gebruiken om het programma voor de Arduino te ontwikkelen en te versturen naar het board, maar gaan we het ook gebruiken om tijdens de uitvoering van het programma berichten tussen de laptop en de Arduino uit te wisselen. Dit is ontzettend handig bij het vinden van fouten, maar we kunnen het ook gebruiken om een opdracht, een commando, naar de Arduino te sturen.

Bij datacommunicatie zijn er een aantal belangrijke begrippen waaronder simplex, half duplex en full duplex. Ook belangrijk is het verschil tussen parallelle en seriële communicatie.

2.2 SIMPLEX, HALF EN FULL DUPLEX

Simplex

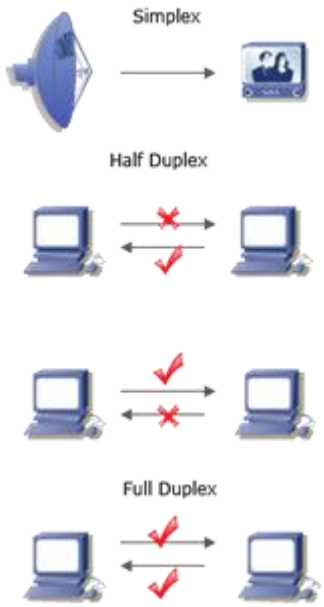
Bij simplex is er sprake van een-weg communicatie. Denk b.v. aan radio en televisie; er is een zender en er zijn een of meer ontvangers, die niets terug kunnen sturen naar de zender.

Full duplex

Bij full duplex kunnen beide partijen gelijktijdig zenden en ontvangen.

Half duplex

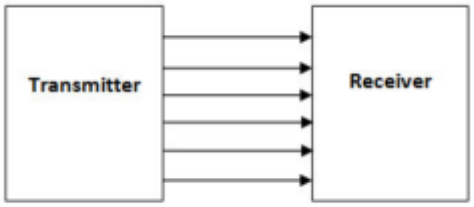
Bij half duplex kunnen de partijen zenden en ontvangen alleen niet gelijktijdig. Ze mogen om beurten zenden. Denk hierbij aan een mobilofoon/walkie talkie. De gebruikers moeten afspreken wanneer de ander mag gaan praten ('over').



Figuur 1 http://ciscofriend.blogspot.nl/2011/12/different-types-of-communication.html

2.3 PARALLEL

Bij de communicatie tussen computeronderdelen of computer onderling willen we het liefst nibbles (4 bits), bytes (8), woorden (16) of dubbele woorden (32) in één keer uitwisselen. Dit is mogelijk door alle bits tegelijk via het medium te versturen b.v. door meerdere draden parallel te gebruiken. **Parallelle communicatie** is snel, omdat alle bits tegelijk gaan maar is ook duur, want het kost zoveel draden.



Parallel transmission

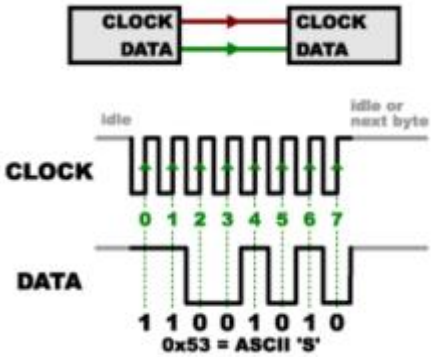
2.4 SERIEEL

Als we vanwege kosten de bits niet tegelijk kunnen versturen, blijft de mogelijkheid over om ze achter elkaar te versturen: **seriële communicatie** dus.

Serial transmission

2.5 SYNCHRONE COMMUNICATIE

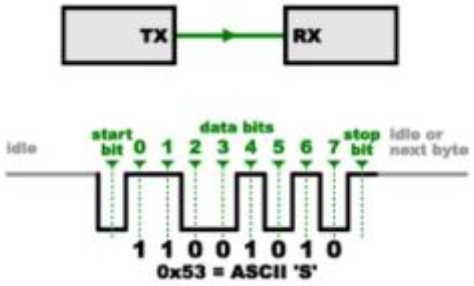
Wanneer bits achter elkaar verstuurd worden, is het nodig dat de ontvanger weet wanneer de volgende bit binnenkomt. Dit kun je doen door de verzender op een extra draad (=duur) een kloksignaal mee te sturen (zie afbeelding hieronder). Telkens als de klok van hoog naar laag gaat of andersom, weet de ontvanger dat er op de data-draad een nieuwe bit uitgelezen kan worden.



2.6 ASYNCHRONE COMMUNICATIE

Ook zonder een kloksignaal kun je bits achter elkaar versturen. Je spreekt dan van tevoren af hoe snel de bits achter elkaar verstuurd worden, en spreekt af hoe de ontvanger weet dat het signaal begint. Het voordeel hiervan is dat je een draad minder nodig hebt, het nadeel is dat de kans groter is dat de verzender en ontvanger 'uit sync' gaan lopen en de data niet juist wordt overgestuurd. Dit zal gebeuren als de klokken van zender en ontvanger niet exact even snel lopen (en in de praktijk zal dat altijd zo zijn). Wanneer je een

lagere overdrachtssnelheid kiest, zal het langer duren voordat de klokken echt uit sync raken, maar dit heeft dan weer als nadeel dat de dataoverdracht langer zal duren.



In het voorbeeld hierboven begint de communicatie met een 1 gevolgd door een 0; dit noemen we het startbit. Daarna volgen de databits: soms 6, 7 of 8 bits, steeds gevolgd door een 1 zijnde het stopbit. De eenvoud van dit protocol verklaart zijn enorme populariteit; er is immers geen extra draadje nodig. Nadeel is dat we minstens 2 bits gebruiken voor start en stop en dat zijn geen databits! Ook blijkt dat het asynchroon gelijklopen van de klok werkt tot ong. 100 kbps; daarboven is het aantal fouten te groot. Dit komt, omdat de ontvanger zijn klok als het ware gelijk zet op het moment van het startbit. Daarna moet de ontvangende klok hopen dat hij synchroon met de zender blijft lopen.

Bronnen:

https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

2.7 PARITEIT

Pariteit is een manier om bij het transport van meerdere bits te controleren of er onderweg fouten zijn opgetreden. Dit wordt gedaan door 1 extra bit toe te voegen: het pariteitsbit. Er moet worden afgesproken of er gebruik wordt gemaakt van **even** of **oneven** pariteit. Bij even pariteit zorgt het pariteitsbit ervoor dat het totaal aantal 1'n van de bitten inclusief het pariteitsbit even is.

Original Data	Even Parity	Odd Parity
00000000	0	1
01011011	1	0
01010101	0	1
11111111	0	1
10000000	1	0
01001001	1	0

De ontvanger controleert op identieke manier de pariteit en als dit niet klopt kan de ontvanger besluiten de data als fout te zien en niet door te geven aan b.v. het scherm. Het probleem van pariteit is dat een enkele bitfout wel wordt gedetecteerd, maar een dubbele fout niet.

Om deze reden wordt pariteit tegenwoordig niet vaak meer in datacommunicatie toegepast; daarvoor zijn betere methoden zoals de Cyclic redundancy check (waarop we later in het vak terugkomen). Pariteit wordt nog wel gebruikt bij de communicatie tussen geheugens en processor.

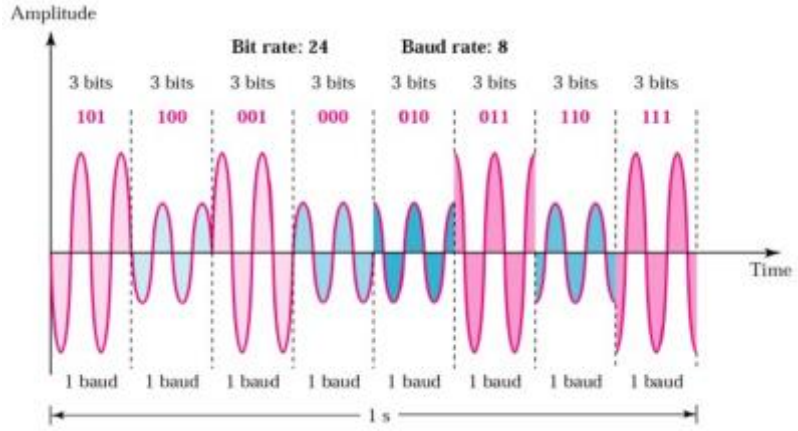
2.8 OVERHEAD

Bij asynchrone communicatie wordt dus een deel van de communicatie gebruikt voor het mogelijk maken van de communicatie. Als we b.v. 8 databits willen versturen wordt 1 startbit en 1 stopbit toegevoegd. Dit betekent dat als we 1000 bits per seconde kunnen versturen via een seriele verbinding er 100 stukken worden verzonden van ieder 10 bits (8 databits, 1 start- en 1 stopbit). Effectief kunnen er dus maar 800 bps aan data worden verzonden en gaan 200 bps 'verloren' aan start- en stopbits. Dit noemen we de overhead.

2.9 BAUD EN BIT/S (WORDT NIET GETOETST)

Vaak zie je de term baud of baudsnelheid gebruikt worden (ook bij de seriële monitor in de Arduino IDE). Helaas wordt ook vaak ten onrechte verteld dat de baudsnelheid gelijk is aan het aantal bits per seconde. Voor eenvoudige seriële datacommunicatie is de baudsnelheid gelijk aan het aantal bits per seconde.

De baudsnelheid is echter gelijk aan het aantal signaalveranderingen per seconde op de communicatielijn tussen zender en ontvanger en technisch kunnen er meer bits in één signaalverandering zitten. Zie een voorbeeld met een baudsnelheid van 8 en een bitsnelheid van 24. Hierbij worden de bitten 'verstopt' in zowel de amplitude als de discontinuïteit van de sinusgolven (de codering wordt 8 QAM genoemd):



Figuur 2 http://www.slideshare.net/soumya604159/ch-05

3 SERIAL

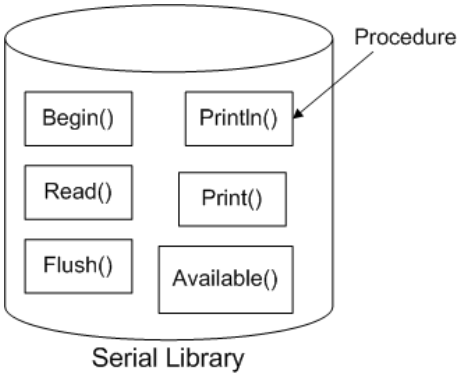
3.1 ARDUINO USB-AANSLUITING

De Arduino kan via een kabel verbonden worden met een USB-aansluiting van de laptop. USB is een seriële asynchrone full duplex verbinding (dat het om asynchrone communicatie gaat, ga je onder andere zien aan het feit dat we een overdrachtssnelheid moeten instellen in zowel het Arduino-programma als op de laptop).

Via deze verbinding kunnen we programma's vanuit de IDE naar de Arduino 'uploaden'. Ook kun je verbinding gebruiken om tijdens het uitvoeren van je programma dingen te laten afdrukken op de laptop en om opdrachten naar het programma te sturen b.v. om een ander knipperpatroon te laten zien.

3.2 INGEBOUWDE SERIAL LIBRARY

In de Arduino zit standaard een seriële interface ingebouwd, die je in code kunt aanspreken.

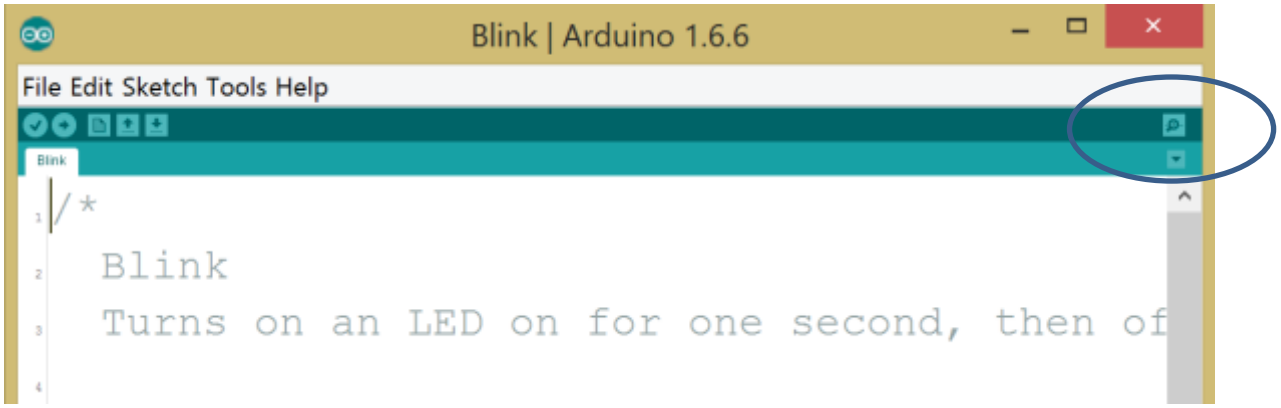


Het gebruiken van de library-functies doe je door voor de functienaam de naam van de library te plaatsen dus Serial.begin

Om in een stuk code aan te geven, dat er gecommuniceerd gaat worden via de seriële verbinding moeten we de snelheid instellen; dat gaat via de parameter van de Serial.begin b.v. Serial.begin(9600).

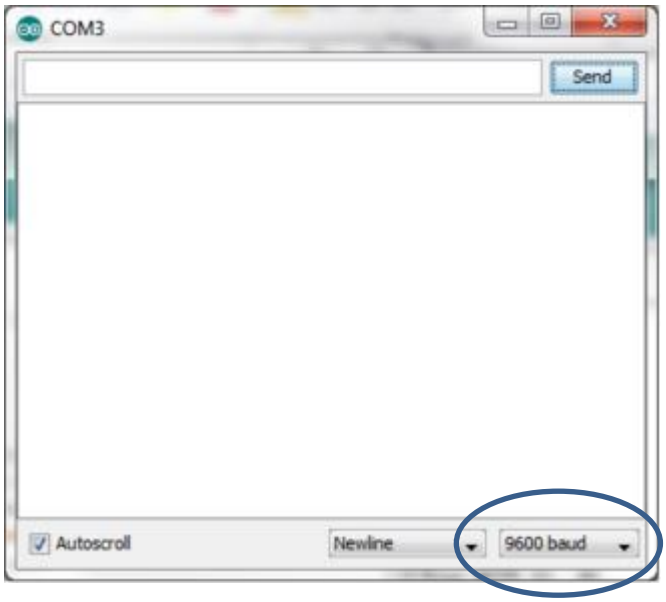
3.3 SERIAL MONITOR

Normaliter staat op het scherm de code van het programma, dat we aan het bouwen zijn. Om via de seriële verbinding met de Arduino te communiceren moeten we de seriële monitor openen. Die zit rechts bovenaan op het scherm van de Arduino-IDE.



Als je erop klikt, verschijnt de seriële monitor:

Pas op: de snelheid rechtsonder moet overeenkomen met de in het programma ingestelde snelheid!



3.4 PRINT() EN PRINTLN()

Met de functies print() en println() uit de library Serial kan informatie vanuit de Arduino via de seriële verbinding naar de laptop worden verzonden. Het verschil tussen print() en println() is dat de laatste na de te printen gegevens tekens meestuurt die ervoor zorgen dat er een nieuwe regel begonnen wordt.

Voorbeeld:

```
void setup() {
  ledControlSetup();
}

// the loop routine runs over and over again forever:
void loop() {
  print("Led aan");
  ledControlSetLedOn();    // turn the LED on
  delay(DELAYTIME);         // wait for a second
```

```
  println(" Led uit");
  ledControlSetLedOff();  // turn the LED off
  delay(DELAYTIME);        // wait for a second
}
```

Zie ook: https://www.arduino.cc/en/Serial/Print

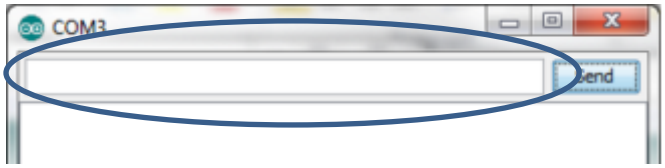
3.5 READ() EN AVAILABLE()

Het lezen van de seriële gegevens is iets complexer. Voordat je gaat lezen wat er ontvangen is, moet je controleren of er al iets ontvangen is. Dit laatste controleer je via de functie Serial.available().

Kijk naar onderstaand voorbeeld (uit: https://www.arduino.cc/en/Serial/Read) waarbij een op de laptop ingetypte letter meteen wordt geretourneerd door de Arduino-code:

```
int incomingByte = 0;   // for incoming serial data

void setup() {
        Serial.begin(9600);     // opens serial port, sets data rate to
9600 bps
}

void loop() {

        // send data only when you receive data:
        if (Serial.available() > 0) {
                // read the incoming byte:
                incomingByte = Serial.read();

                // say what you got:
                Serial.print("I received: ");
                Serial.println(incomingByte, DEC);
        }
}
```

Het versturen van informatie naar de Arduino toe, kun je door in de bovenste balk van de serial monitor iets in te typen gevolgd door een druk op Send:



4 KNIPPEREN MEERDERE LED'S IN CODE

4.1 APARTE FUNCTIES VOOR ÉÉN LED

In de vorige code les hebben we de volgende functies gebouwd:

- ledControlSetup()
- ledControlSetLedOn()
- ledControlSetLedOff()

Of een variant daarvan.

4.2 APARTE FUNCTIES VOOR IEDERE LED

We kunnen natuurlijk het volgende bedenken:

- ledControlSetup1()
- ledControlSetLedOn1()
- ledControlSetLedOff1()
- ledControlSetup2()
- ledControlSetLedOn2()
- ledControlSetLedOff2()

Wat is hiervan het nadeel?

4.3 FUNCTIESPECIFICATIE VOOR BESTUREN MEERDERE LED'S

We gaan de volgende functies bouwen:

- ledControlSetup()
- ledControlSetLedOn(ledNumber)
- ledControlSetLedOff(ledNumber)
- ledControlAllLedsOn()
- ledControlAllLedsOff()

De functie ledControlSetup() initialiseert alle poorten voor alle aangesloten LED's.

De functie ledControlSetLedOn(ledNumber) zet LED met **volgnummer** ledNumber aan net zo als ledControlSetLedOff(ledNumber) deze weer uit zet.

Het is belangrijk dat we werken met een voor ons programmeurs logisch volgnummer en dat de werkelijk pinnummers verborgen zitten in de library: je neemt de pinnummers dus op in het tabblad waarin de daadwerkelijke ledcontrole plaatsvindt, zodat andere tabbladen zonder kennis van die pinnummers toch met de LED's kunnen werken.

De functie ledControlAllLedsOn() zet alle LED's in één keer aan en duidelijk ledControlAllLedsOff() zet ze allemaal uit.

4.4 OPZET LEDCONTROLSETUP()

We geven een stukje van de code weg; door gebruik te maken van een array met echte pinnummers, kunnen we in de software verwijzen naar PINARRY[ledNumber] waarbij ledNumber het logische volgnummer van de LED is. Ook zijn voor de gebruiker van deze library de echte pinnummers niet meet nodig; hij gebruikt altijd logische volgnummers.

```
const int NUMBEROFLEDS = 2;
const int PINARRAY[] = {
  13, 12};

 // setup for initializing LEDs
void ledControlSetup() {
  for (int ledNumber = 0; ledNumber < NUMBEROFLEDS; ledNumber++) {
    // initialize the digital pin as an output.
    pinMode(PINARRAY[ledNumber], OUTPUT);
  }
}
```