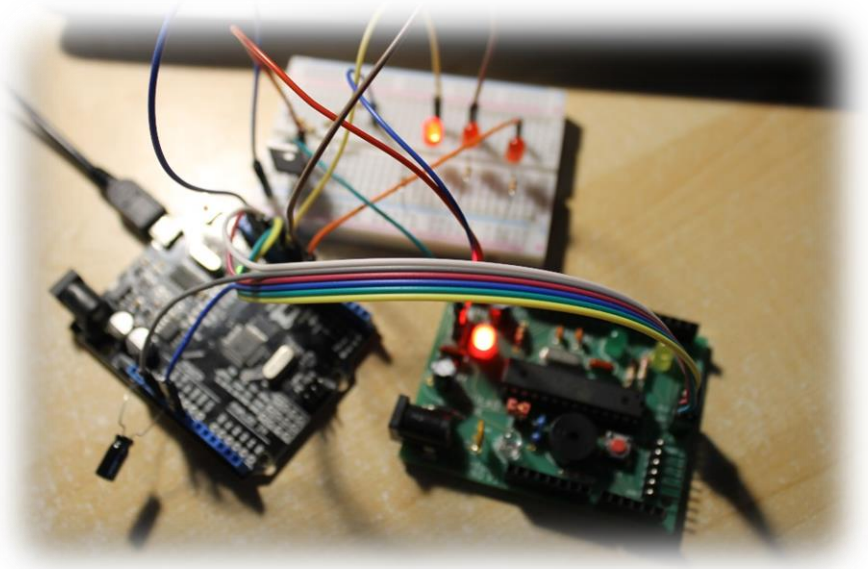
Arduino

voor gevorderden

Deel 2



Versie: 4.0

Datum: 24/08/2021

Auteur: Frank Marchal

Doelgroep: 15 tot 88 jaar (na basiskennis en gevorderden 1 Arduino)

1 Inleiding

Dit document heeft tot doel de cursist zover te krijgen dat hij de **Arduino UNO** omgeving verder kan inzetten voor projecten met allerhande communicatie met andere chips.

In de basiscursus werd er met **basiselektronica (LED, switch, buzzer, motor zonder/met PWM)** gewerkt en gingen we zo de programmeer omgeving leren ontdekken en uitzoeken. Daarna zochten we uit hoe we met een Arduino UNO bordje allerlei sensoren zoals een **afstandssensor** konden aansturen. Ook leerden we de robot met **bluetooth** en **lijnvolgers** besturen.

In de **gevorderden cursus deel 1** gingen we nu nog meer elektronica aansturen. Zo hebben we het o.a. over de **analoge inputs, LCD schermen, I2C, stappenmotoren, servo motoren, IR signalen gehad.**

In principe is het geen must om eerst gevorderden 1 gevolgd te hebben voor je naar gevorderden 2 kan gaan. Natuurlijk ga je jezelf comfortabeler voelen als dit wel zo is.

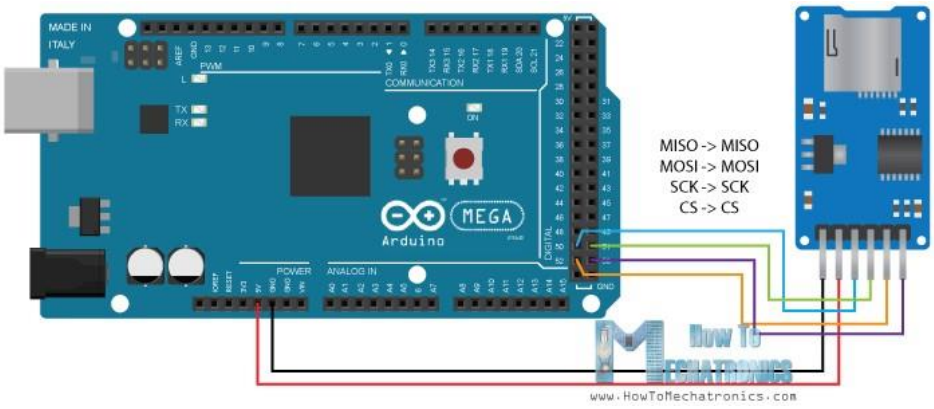
In **gevorderden deel 2** leggen we nu meer de nadruk op **communicatie** met andere chips: **RS232, I2C met geheugens, USB, SPI via SD card lezen, RF signalen, RFID, WIFI** zullen de revue passeren.

We gaan ook een **eigen arduino UNO** solderen en van bootloader voorzien (zie hiervoor het apart toegevoegde document op jouw USB stick kopie)(Zie 4EE voor studenten).

Info hierover kan je ook vinden op www.arduino.cc

Veel succes !

Frank Marchal (www.edulab.be)



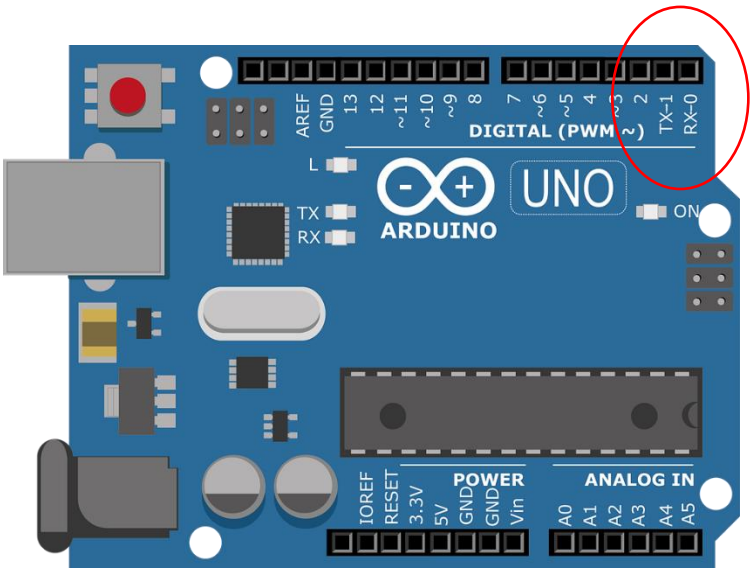
1. RS232 communicatie onderzoeken

Om deze cursus te starten gaan we eerst de RS232 communicatie eens grondig onderzoeken.

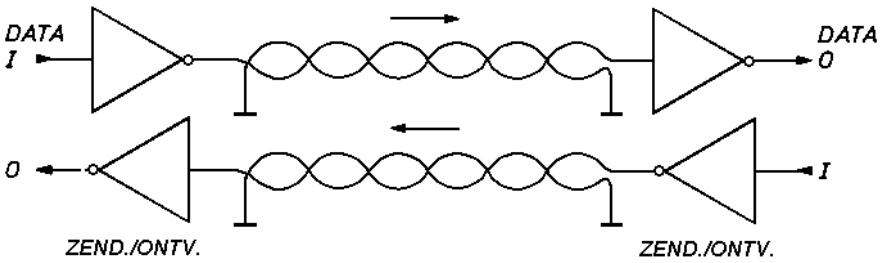
We gaan op verschillende manieren RS232 gebruiken om met andere apparaten te kunnen communiceren. Deze eenvoudige manier van data versturen kan helpen bij het debuggen van een code of het aansturen van een apparaat.

Elke Arduino UNO heeft een RX en TX pin (pin 0 en pin 1). Deze worden gebruikt om te communiceren met de buitenwereld, maar kunnen ook ingezet worden om een programma op te slaan in het geheugen van de Arduino (dit leren we tijdens de lessen “bouw je eigen Arduino).

RX staat voor “receiver” (ontvanger) terwijl TX staat voor “transmitter” (verzender).



Een RS232 signaal hoort bij de familie van de **“full duplex”** verbindingen. Dit wil zeggen dat je in beide richtingen **“tegelijkertijd”** kan communiceren. De data zal bit per bit (dus serieel) doorgestuurd worden.

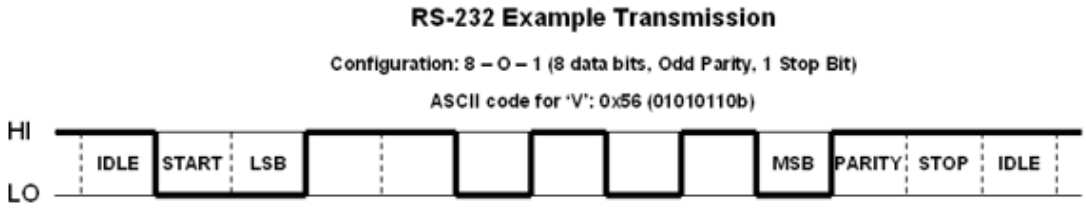


Typische full duplex toepassingen zijn GSM’s en RS232

Hoe is een RS232 signaal opgebouwd?

Een RS232 signaal hoort tot de **"asynchrone"** seriële verbindingen. Dit wil zeggen dat de data op elk moment kan verstuurd worden, **zonder** rekening te houden met een **aparte klok** lijn.

De klok zit als het ware verstopt in het datasignaal. Daar hebben ze de start en stop bit voor bedacht.



Bovenstaande tekening geeft aan hoe de microcontroller het RS232 signaal verstuurd.

- We starten met een **startbit van 1 bittijd** om het begin van het datawoord aan te geven.
- We stoppen met een **stopbit van 1 of 2 bittijden** (in te stellen) om het einde van het datawoord in te stellen.
- Merk op dat de data start met de **LSB** (Least Significant Bit = minst beduidende bit) en eindigt met de **MSB** (Most Significant Bit = meest beduidende bit). Je kan 5, 7 of 8 bits instellen.
- Ter controle, of de bits goed zijn verzonden, wordt er een **Pariteitsbit** mee verstuurd (instelbaar). **ODD** (oneven) bits wil zeggen dat de data bits + de pariteitbit samen een oneven aantal "1"-en vormt. **EVEN** (even) zorgt ervoor dat we een even aantal "1"-en hebben die we gaan versturen.
- **IDLE** is de tijd dat er niets wordt verstuurd. Dan blijft de TX of RX datalijn hoog.

Hoe ziet een RS232 signaal eruit op de **logic analyzer**?

We kunnen het RS232 signaal makkelijk meten met de 5V logic analyzer.

Hiervoor sluiten we bijvoorbeeld op de UNO, op de TX pin (pin 1) de CH1 van de analyzer aan t.o.v. GND. Op CH2 sluiten we RX aan.

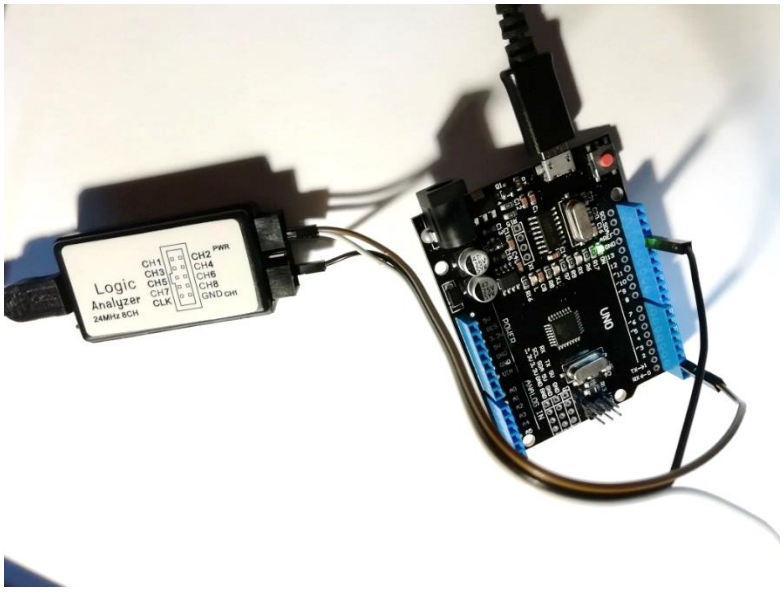
We downloaden de volgende code in onze UNO.

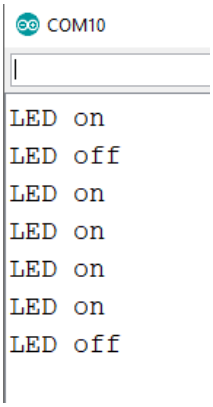
serial_com_TX_meten

```
//geen virtuele poort versie
//RX = 0 (UNO)  (uittrekken draadje tijdens download code!)
//TX = 1 (UNO)

#define LED 13//ingebouwde LED
char a; //opslag inkomend karakter

void setup()
{
  pinMode(LED, OUTPUT);
  Serial.begin(9600);
}
```

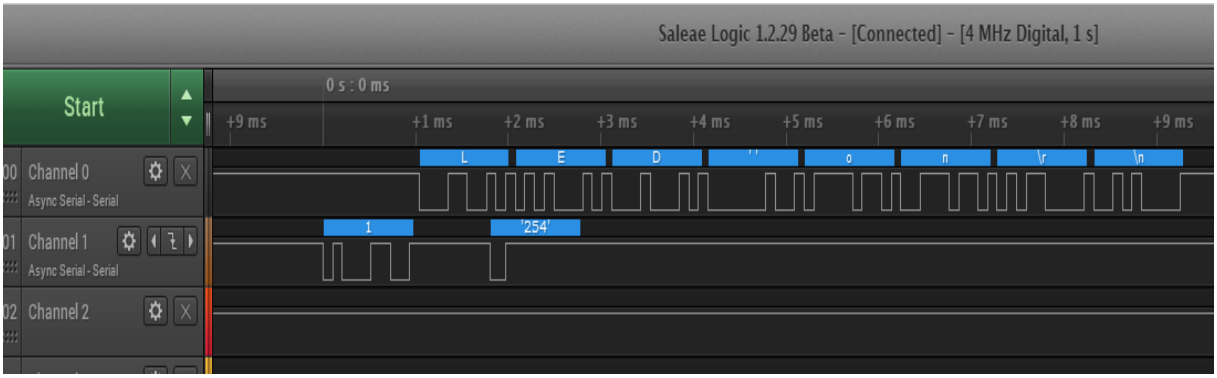
```
void loop()
{
  if (Serial.available())
  // if text arrived in from    serial...
  {
    a=(Serial.read());
    if (a=='1')
    {
      digitalWrite(LED, HIGH);
      Serial.println("LED on");
    }
    if (a=='2')
    {
      digitalWrite(LED, LOW);
      Serial.println("LED off");
    }
  }
}
```

De code zorgt ervoor dat we, telkens we een “1” of “2” verzenden via de serial monitor, een LED op de Arduino gaan aan/uit doen op pin 13 (BUILD IN LED). Ondertussen wordt er een tekst gestuurd naar de serial monitor via TX. Deze tekst willen we graag meten, samen met de “1” of de “2” op de RX lijn.





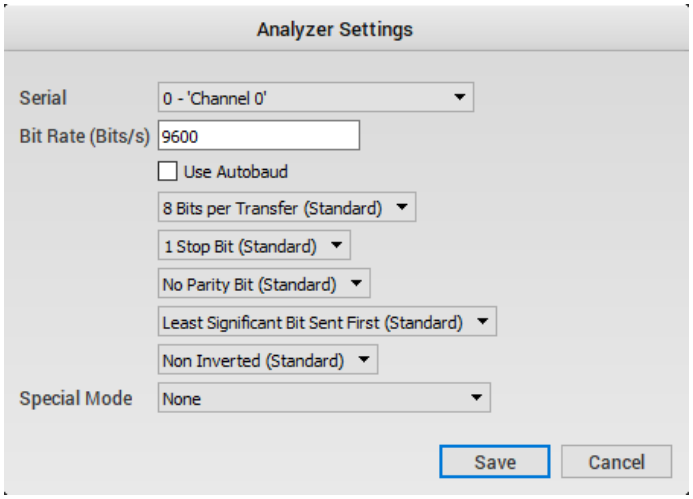
CH1 aansluiten op TX en CH2 op RX



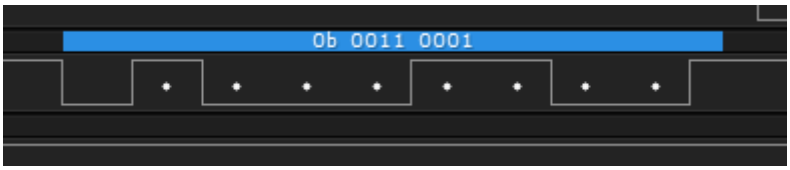
Er is getriggerd op de neergaande flank van CH0 (op de analyzer), terwijl het CH1 in de software is (1 kanaal verschoven)!

Snelheid logic analyzer: 4MHz sample rate, 1 sec

Instellingen serial analyser:



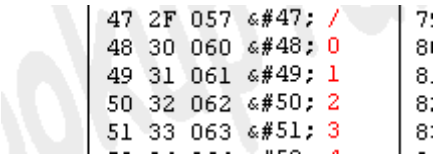
Als we de “1” beter bekijken die we verstuurd hebben via TX:



Er staat op de scope start + 1000 1100 + stop

Als we LSB en MSB omkeren krijgen we “0011 0001” = 0x31

Als we in de ASCII tabel kijken dan krijgen we de “1” als verzonden karakter.



➔ **Taak 1:** test de code voor de communicatie tussen UNO en PC en meet enkele characters met de logic analyzer. Leg het verband tussen de theorie van RS232 en de gemeten signalen.

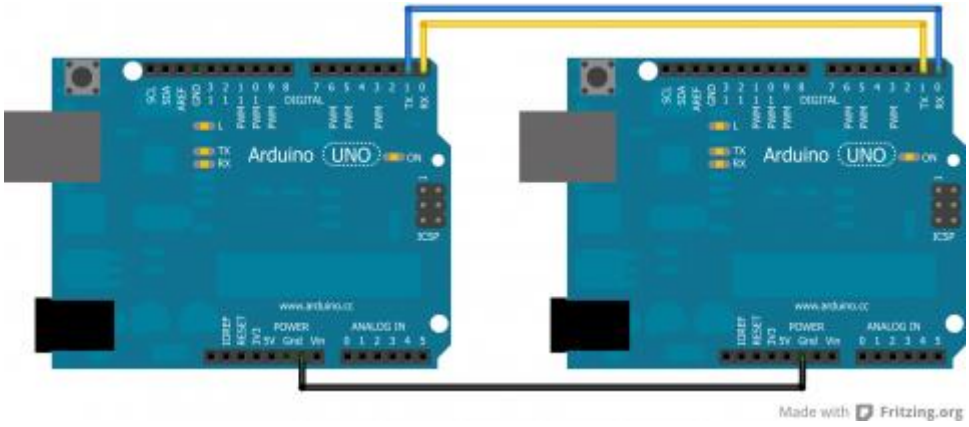
1.1 RS232 communicatie tussen 2 arduino's

Nu we kunnen communiceren tussen de COM poort van de PC en de Arduino, gaan we een andere communicatie bus opzetten. Dit keer praten de 2 UNO's met elkaar.

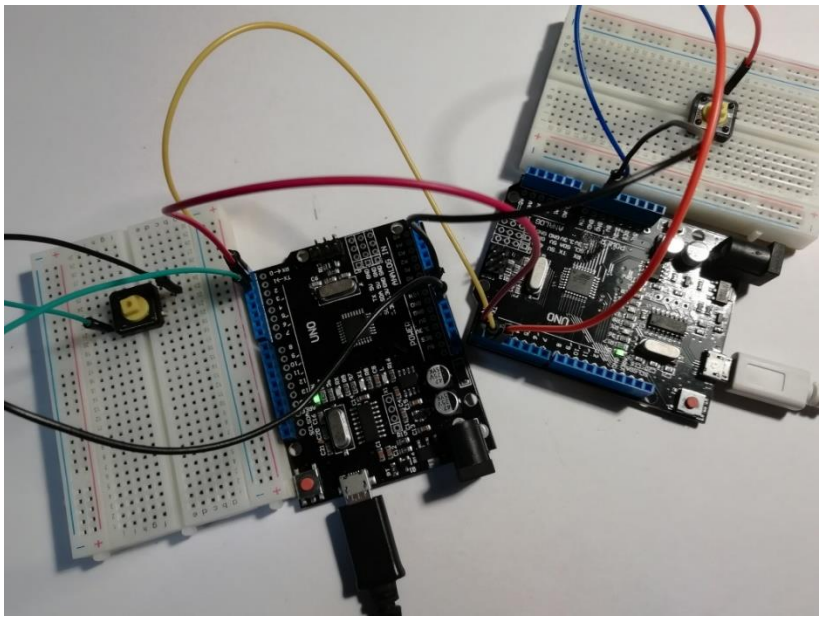
Ideaal is dat je de beide UNO's op een andere PC aansluit. Zo kan je dan via de serial monitor van beide PC's controlleren of er communicatie is. Je kan ook 1 UNO op de PC aansluiten en de andere via een GSM adapter. Merk op dat je geen data van de 2 UNO's tegelijk naar dezelfde serial monitor kan sturen.

We kunnen ook code schrijven waarbij een knop op de ene UNO een LED op de andere aanstuurt en omgekeerd. De linkse UNO hangt bijvoorbeeld aan de PC voeding/COM poort en de rechtste UNO is aangesloten op een GSM adapter.

Bouw de volgende opstelling:



Voeg aan beide UNO's op pin 2 nog een laag actieve knop toe.



Laad nu de volgende code in beide UNO's: TX en RX draad even verwijderen!

serial_com_tussen_2_uno_s

```
//geen virtuele poort versie
//RX = 0 (UNO)  (uittrekken draadje tijdens download code!)
//TX = 1 (UNO)

#define LED 13//ingebouwde LED
#define KNOP 2
bool vlag = 0;
char a; //opslag inkomend karakter

void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(KNOP,INPUT_PULLUP);
  Serial.begin(9600);
}


    void loop()
    {
      if (digitalRead(KNOP) == 0){ //laag actief
        delay(200);//antidender
        vlag = !vlag;
        Serial.print("vlag = ");
        Serial.println(vlag);

      }

      if (Serial.available())
      // if text arrived in from BT serial...
      {
        a=(Serial.read());
        if (a=='0')
        {
          digitalWrite(LED, HIGH);
          Serial.println("LED on");
        }
        if (a=='1')
        {
          digitalWrite(LED, LOW);
          Serial.println("LED off");
        }
      }
    }
```

Laad deze code in beide UNO's. Test dan de knoppen uit.

Wat gebeurd er met de LED 13?

Check de serial monitor.

Meet ook hier met de Logic Analyser wat er op de scope passeert.

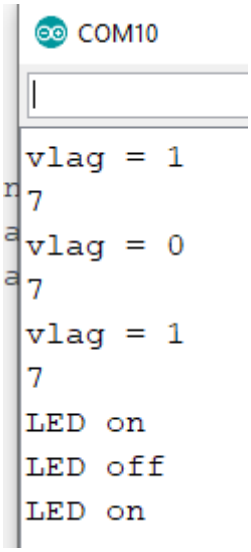
- ➔ **Taak 1:** test de code voor de communicatie tussen 2 UNO's en meet enkele characters met de logic analyzer. Leg het verband tussen de theorie van RS232 en de gemeten signalen.
- ➔ **Opdracht:** maak nu zelf de oefening van Taak 1, maar met de **softwareSerial** functie. Kijk op www.arduino.cc hoe je dan de code en bedrading moet aanpassen.

Test nu de volgende code op de 2 UNO's uit. De code is voorzien van een **string**.

Pas daarom een deel van de code aan:

serial_com_tussen_2_uno_s_string

```
#define LED 13//ingebouwde LED
#define KNOP 2
bool vlag = 0;
char a; //opslag inkomend karakter
byte lengte = 0;//opslag lengte string
char tekst[20] = "vlag = ";//array van 20 karakters met een tekst
                           //array mag groter zijn dan tekst
void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(KNOP,INPUT_PULLUP);
  Serial.begin(9600);
}

void loop()
{
  if (digitalRead(KNOP) == 0){ //laag actief
    delay(200);//antidender
    vlag = !vlag;
    Serial.print(tekst);
    Serial.println(vlag);
    lengte = strlen(tekst);//geef lengte van string terug
    Serial.println(lengte);//print lengte van string
   }


  if (Serial.available())

  {
    a=(Serial.read());
    if (a=='0')
    {
      digitalWrite(LED, HIGH);
      Serial.println("LED on");
    }
    if (a=='1')
    {
      digitalWrite(LED, LOW);
      Serial.println("LED off");
    }
  }
}
```



Hoe werkt de **string**?

De string is een verzameling van karakters. Je geeft op het begin in de code aan hoeveel plaatsen er in de string zijn. Hier moet je binnen blijven. Je mag altijd meer plaatsen dan inhoud voorzien.

Een string kan je vullen met aparte karakters:

Char test[7] = {'t','e','s','t',\0'};

\0 betekent dat dit het einde van de string is.

Om deze terug te tonen doe je het volgende: Serial.println(test);

Het intypen van al deze aparte karakters vraagt veel tijd en je moet nog eens de '\0' toevoegen.

Daarom is er in Arduino een gemakkelijker c-code manier voorzien:

Char test[] = "test";

Nu moeten we geen ' ' meer ingeven, geen \0 en ook geen lengte van de string. Het komt allemaal goed door de tekst gewoon tussen " …. " te plaatsen.

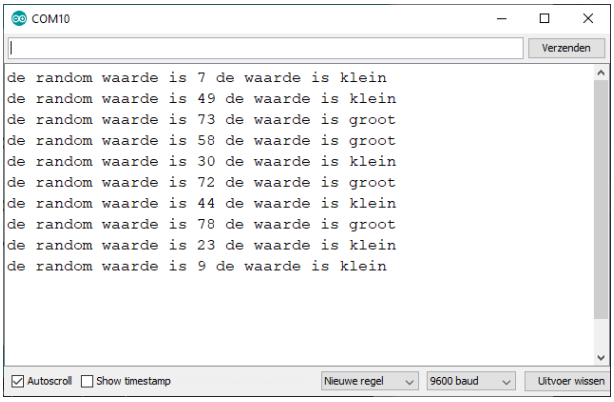
Wat kan ik nog met een string doen?

strcpy(s1,s2)	Kopieer inhoud s2 in s1
strcmp (s1,s2)	Vergelijk inhoud van s1 en s2: alles ze gelijk zijn is het resultaat 0, als s1 > s2 negatief, als s1 < s2 positief
strlen(s1)	Geef de lengte van de string terug
strstr(s1,s2)	Geef de locatie terug van s2 in s1

➔ **Test** volgende code met string manipulaties eens uit: hier wordt in de zelfde variabele telkens een andere string inhoud meegestuurd.

serial_string_manipulatie

```
char msg[50];
long rand1;//gebruik NIET "rand", is gereserveerd!

void setup() {
  Serial.begin(9600);
}
void loop() {
  randomSeed(analogRead(0));
  for (int i = 0; i < 10; i++) {

    rand1 = random(100);
    if (rand1 < 50) {
      strcpy(msg, " de waarde is klein");
    }
    else {
      strcpy(msg, " de waarde is groot");
    }
    Serial.print("de random waarde is ");
    Serial.print(rand1);
    Serial.println(msg);
  }
}
```



Hoe stuur je variabelen door i.p.v. characters via de serial monitor?

Hiervoor gaan we eerst de variabele waarde in de zender omzetten van integer naar een karakter array. Dit kan met het speciale commando **itoa(value,str,10);**

Itoa() staat voor "integer to ASCII" convertie.

Daarna wordt de variabele verstuurd in afzonderlijke karakters achter elkaar.

Zender code:

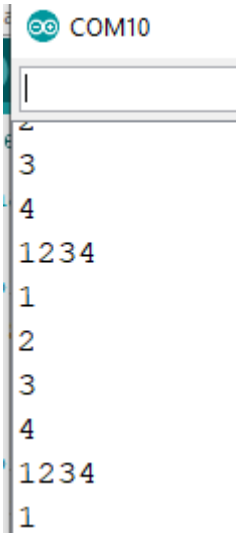
serial_com_zender §

```
char str[4];

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

void loop() {
  int value = 1234;//vb van sensor waarde

  itoa(value,str,10);//zet nummer om in een karakter (char) array
  Serial.write(str,4);//stuur 4 karakters

  delay(1000);
}
```

Ontvanger code:

serial_com_ontvanger §

```
//Receiver Code
char str[5];
int i = 0;
void setup() {
  Serial.begin(9600);
}
void loop() {
  if (Serial.available()) {
    delay(100); //zo sta je toe dat eerst de 4 karakters
    //tegelijk ontvangen worden
    while (Serial.available() && i < 4) { //haal de 4 karakters binnen
      str[i] = Serial.read();
      Serial.println(str[i]);
      str[i++];
    }
    str[i] = '\0'; //voeg het einde van de string toe
  }
  if (i == 4) {
    Serial.println(str);//print inhoud str
    Serial.println(str.toInt());//print int waarde van str inhoud
    i = 0;
  }
}
```



We lezen eerst alle karakters in en verzamelen de juiste hoeveelheid in 1 string. We voegen er ook het "\0" (einde van de string) aan toe. Daarna printen we gans het getal weer af in 1 keer.

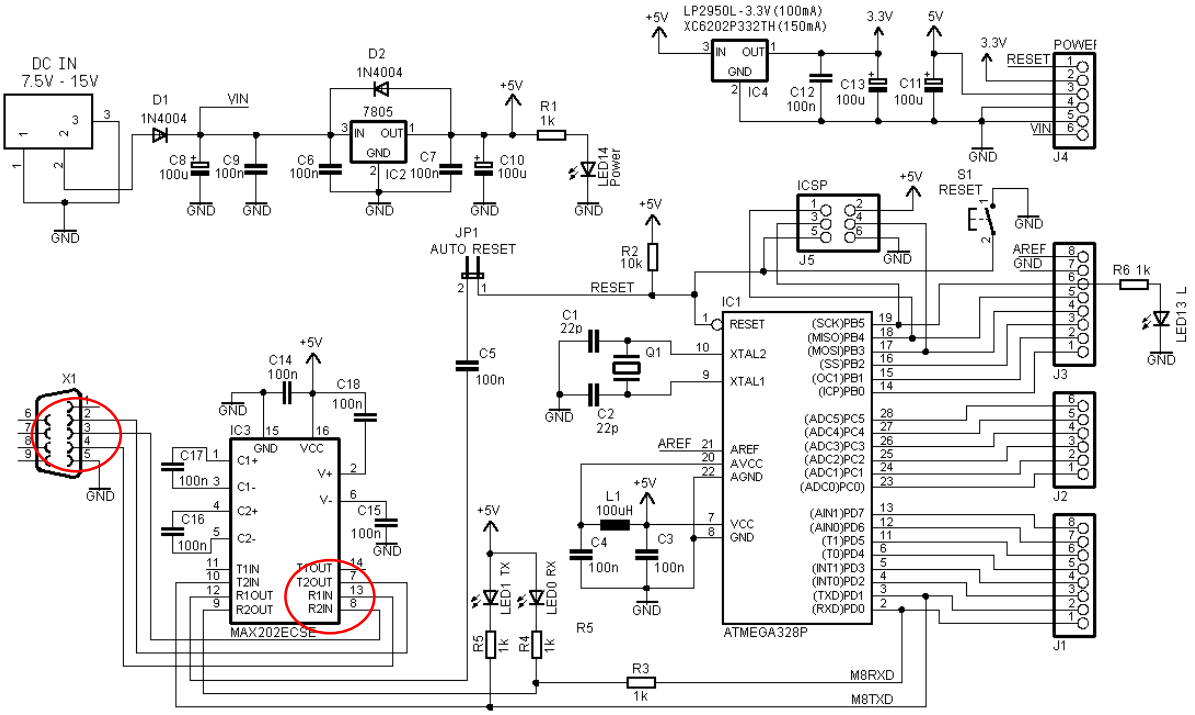
➔ **Taak 2:** test de vorige 2 oefeningen van stringmanipulatie uit. Steeds de serial monitor output ook in jouw verslag vermelden!

1.2 RS232 communicatie via een MAX202 chip

We sturen van de Arduino UNO seriële boodschappen uit naar de PC via de speciale FTDI chip die op de UNO is voorzien. Stel nu dat je geen USB poort hebt en ook geen FTDI chip dewelke de omzetting moet doen van RS232 <-> USB. Wat doe je dan?

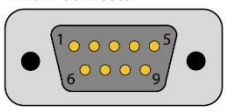
Dan maak je gewoon je **eigen omzetter met een MAX202 chip**.

Onderstaand schema geeft hiervan een mooi overzicht.



Merk op hoe de aansluitingen zijn voorzien op de RS232 connector.

Hier worden de TX en RX omgekeerd aangesloten t.o.v. de MAX202 chip. Dit is natuurlijk nodig om de **TX -> RX en RX->TX twist** te krijgen. Zorg dan wel dat **de RS232 kabel** die jouw opstelling verbind met de PC een **straight** (rechtdoor) verbinding heeft voor TX en RX.



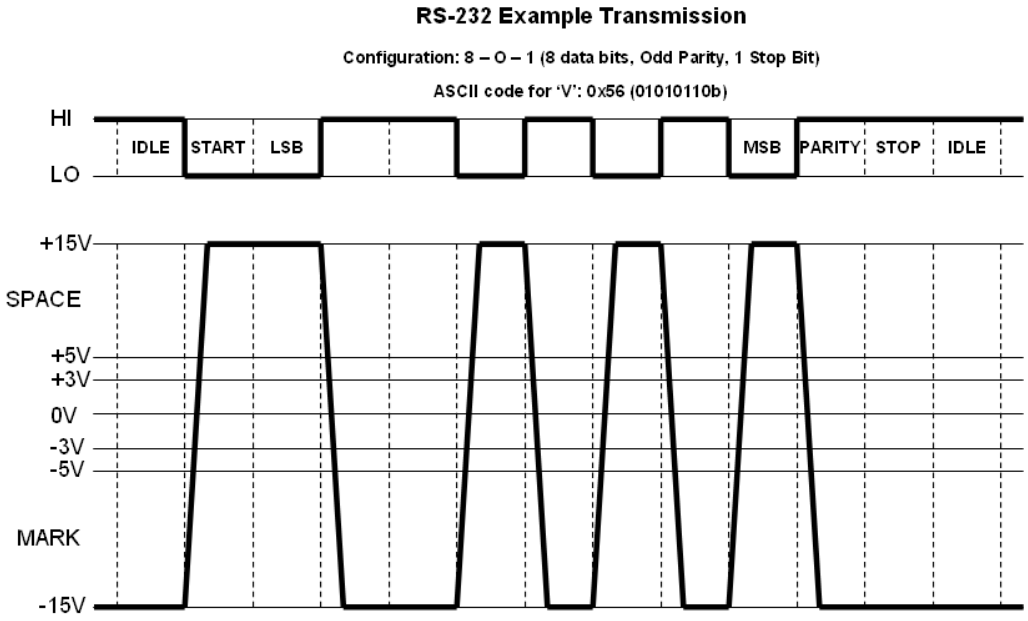
RS232 Pin Out

Pin #	Signal
1	DCD
2	RX
3	TX
4	DTR
5	GND
6	DSR
7	RTS
8	CTS
9	RI

Merk op dat je 1uF of 100nF condensators moet gebruiken. Als je electrolytische condensators gebruikt moet je de **polariteit** goed in het oog houden in het schema!

De MAX202 chip zet de 5V spanning van de UNO om in een +/- 10V spanning aan PC zijde.

Deze hoge spanning is voorzien om **minder storingsgevoelige** verbindingen te maken en om **langere bedrading** mogelijk te maken (zie spanningsval bij lange draden). Enkel met een scope op meten!

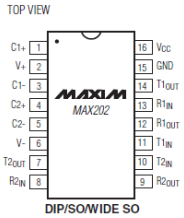


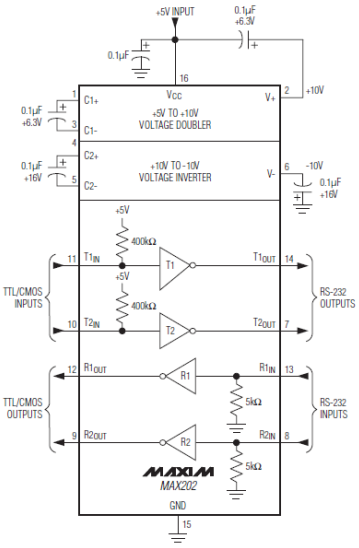
Merk op dat de signalen tussen de TTL uitgang van de UNO en de RS232 geïnverteerd zijn.

De hoge signalen van de RS232 noemen ze ook de **SPACE** en de lage signalen de **MARK.**

In praktijk gaan we, om de processor te beschermen tegen te grote stromen, **220 ohm in serie zetten** met de TX en RX, tussen de MAX202 chip en de UNO.

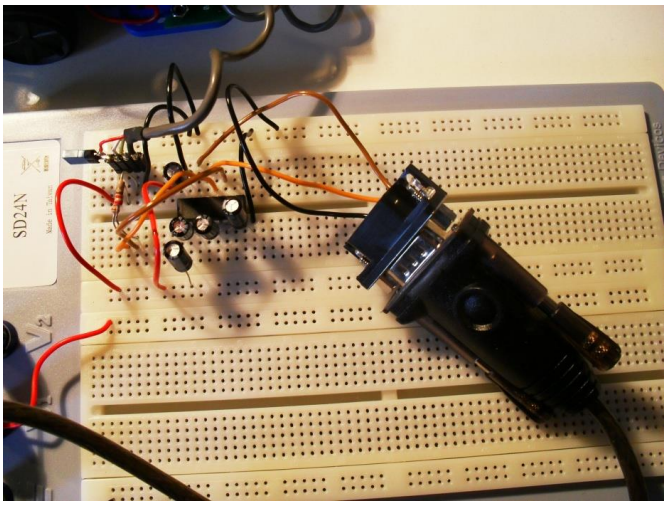
Teken hier de aansluitingen t.o.v. de UNO:





Mogelijk heb je geen RS232 poort meer op jouw PC (deze worden stilaan niet meer voorzien). Wil je dit experiment toch doen, dan moet je een kabel voorzien die USB omzet naar RS232.

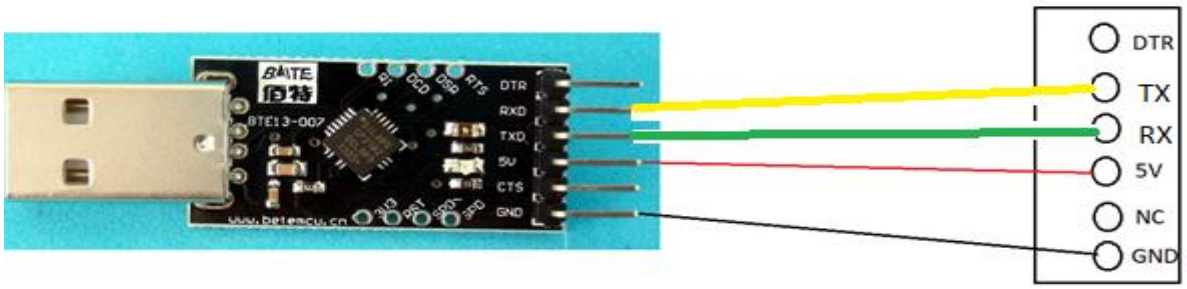
Via een RS232 vrouwelijke connector kan je contact maken met jouw MAX202.



➔ Merk op dat we tegenwoordig deze **MAX202 chip + USB omzetter terugvinden in 1 FTDI of Silicon Lab chip**.

Deze gaan we dan gebruiken om onze eigen Arduino te kunnen programmeren (zie 4EE).

Dan hebben we niet meer te maken met +/- 10V signalen, maar direct 5V TTL signalen.



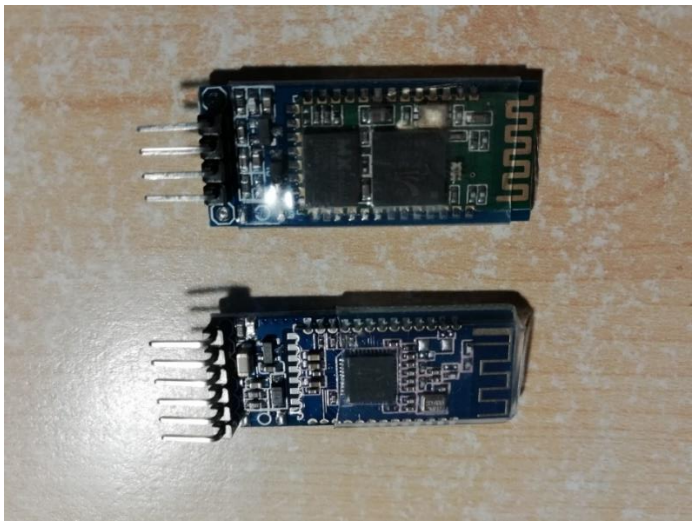
➔ **Taak 3:** doe een test op een breadboard met de communicatie tussen de MAX202 opstelling en de PC. Meet weer met een logic analyzer enkele characters die je verstuurd (let op dat je meet tussen UNO en MAX202, 5V level!)

1.3 RS232 via bluetooth communicatie

(uitbreiding op oefening in 5EE, geen oef maken)

Een serial monitor kan je ook draadloos maken. Hiervoor is de reeds ontdekte bluetooth module HC05, HC06 of HM10 (Apple versie) geschikt. Deze standaard werkt op 2.4GHZ (UHF).

We gaan eerst wat dieper in op de technologie, dan sturen we via TX en RX, op pin0 en pin 1 van de Arduino, de module aan.



Modules kan je gebruiken voor bluetooth communicatie met de Arduino: de bovenste is de HC06 en de onderste is de HM10.

Waar zit het verschil in de HC06 en HM10?

csr

Device Features

- Fully Qualified Bluetooth v2.0+EDR
- Enhanced Data Rate (EDR) compliant with v2.0.E.2 of specification for both 2Mbps and 3Mbps modulation modes
- Full Speed Bluetooth Operation with Full Piconet Support
- Scatternet Support
- 1.8V core, 1.8 to 3.6V I/O
- Low Power 1.8V operation
- 8 x 8mm 96-ball TFBGA and 6 x 6mm 96-ball VFBGA Package options
- Minimum External Components
- Integrated 1.8V Regulator
- USB and Dual UART Ports
- Support for 802.11 Co-Existence
- Support for 8Mbit External Flash
- RoHS Compliant

BlueCore™4-External

Single Chip Bluetooth®
v2.0+EDR System

Production Information Data Sheet For

BC417143B-IQN-E4

BC417143B-IRN-E4

July 2005

De CSR-BC417 chip zit op de HC06. Deze heeft een bluetooth 2.0 core

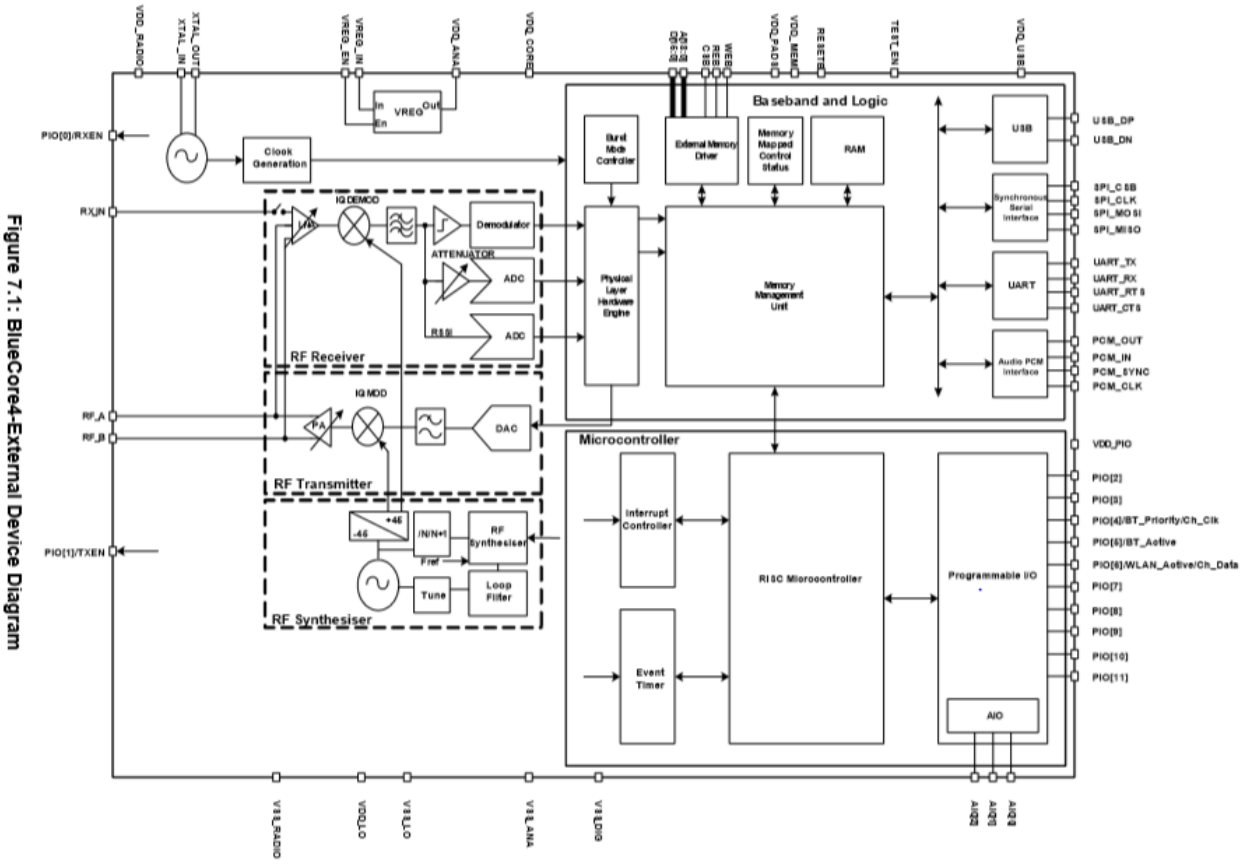


Figure 7.1: BlueCore4-External Device Diagram

De HC06 chip bevat meerdere manieren om de bluetooth signalen om te zetten naar een andere handige communicatie poort. Wij gebruiken de RS232 = UART (Universal Asynchronious Receiver Transmitter).

TEXAS INSTRUMENTS

CC2540F128, CC2540F256

www.ti.com

SWRS084F – OCTOBER 2010 – REVISED JUNE 2013

2.4-GHz *Bluetooth*® low energy System-on-Chip

Check for Samples: CC2540F128, CC2540F256

FEATURES

- True Single-Chip BLE Solution: CC2540 Can Run Both Application and BLE Protocol Stack, Includes Peripherals to Interface With Wide Range of Sensors, Etc.
- 6-mm × 6-mm Package
- RF
  - *Bluetooth* low energy technology Compatible
  - Excellent Link Budget (up to 97 dB), Enabling Long-Range Applications Without External Front End
  - Accurate Digital Received Signal-Strength Indicator (RSSI)
  - Suitable for Systems Targeting Compliance With Worldwide Radio Frequency Regulations: ETSI EN 300 328 and EN 300 440 Class 2 (Europe), FCC CFR47 Part 15 (US), and ARIB STD-T66 (Japan)
- Layout
- ~~TPS62730~~ Compatible
- Low Power in Active Mode
  - RX Down to 15.8 mA (3 V Supply)
  - TX (-6 dBm): 18.6 mA (3 V Supply)
- Microcontroller
  - High-Performance and Low-Power 8051 Microcontroller Core
  - In-System-Programmable Flash, 128 KB or 256 KB
  - 8-KB SRAM
- Peripherals
  - 12-Bit ADC with Eight Channels and Configurable Resolution
  - Integrated High-Performance Op-Amp and Ultralow-Power Comparator
  - General-Purpose Timers (One 16-Bit, Two 8-Bit)

De CC2540 module werkt reeds met BLE (Bluetooth Low Energie)

Bluetoothapparatuur is verdeeld in 3 verschillende klassen:

- Class 1: Ontworpen voor lange afstandsverbindingen (tot ~100m) (100 milliwatt)
- Class 2: Voor normaal gebruik (tot ~10m) (2,5 milliwatt)
- Class 3: Voor korte afstanden (10 cm - 1 m) (1 milliwatt)

Er bestaan ook verschillende bluetoothversies:

- Versie 1: de datasnelheid bedraagt bruto 1 Mbit/s.
- Versie 1.2: deze vernieuwde versie maakt datasnelheid tot 2 Mbit/s mogelijk. Daarnaast verbeterde het spraakkwaliteit en audio-overdracht.
- Versie 2: eind 2004 is een nieuwe verbeterde versie van bluetoothstandaard ontwikkeld en goedgekeurd. De belangrijkste kenmerken zijn:
  - 3 keer zo hoge datasnelheid
  - lager stroomverbruik (wat de levensduur van de batterij verlengt)
  - verbeterde foutcorrectie
  - verbeterde mogelijkheid verbindingen met meerdere apparaten.
- Versie 3: op 21 april 2009 werd een nieuwe versie van bluetooth gepresenteerd. De nieuwe bluetoothversie is weer een stuk sneller en betrouwbaarder en gebruikt wifi (802.11n).
- Versie 4: op 7 juli 2010 werden de specificaties van deze standaard vastgelegd, waarbij er vooral aan de energiezuinigheid werd gewerkt. Vanaf deze versie zijn bluetoothaccessoires die werken op een knoopcel mogelijk.
- Versie 5: op 16 juni 2016 werd versie 5 van de specificatie door de Bluetooth SIG voorgesteld. IoT-technologie speelde een belangrijke rol en de snelheid van BLE (Bluetooth Low Energy) werd naar 2 Mbit/s verdubbeld.

In deze wikipedia tabel kan je mooi de evolutie zien van de verschillende bluetooth standaarden.

Ook deze technologie is full duplex.

Eerst wordt er een **point to multipoint** opgezet. Enkel de bron kan m.a.w. meerdere ontvangers bedienen. Op het moment als er gekoppeld wordt tussen 2 bluetooth apparaten ontstaat er een **piconet.**

Op de volgende website vond ik nog meer info over de 2 soorten radio versies van bluetooth:

https://www.bluetooth.com/specifications/bluetooth-core-specification/

Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR)

The Bluetooth BR/EDR radio is designed for low power operation and also leverages a robust Adaptive Frequency Hopping approach, transmitting data over 79 channels. The Bluetooth BR/EDR radio includes multiple PHY options that support data rates from 1 Mb/s to 3 Mb/s, and supports multiple power levels, from 1mW to 100 mW, as well as multiple security options. It supports a point-to-point network topology that is optimized for audio streaming.

Bluetooth Low Energy (LE)

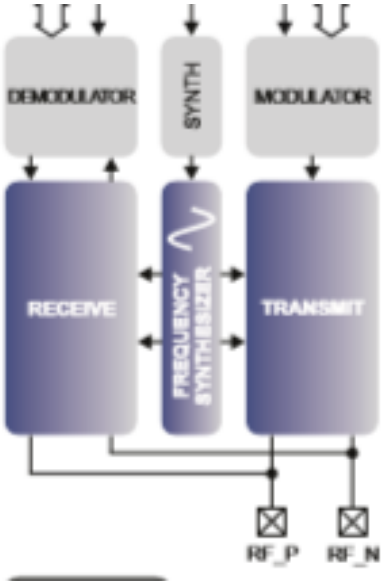
The Bluetooth Low Energy (LE) radio is designed for very low power operation. To enable reliable operation in the 2.4 GHz frequency band, it leverages a robust frequency-hopping spread spectrum approach that transmits data over 40 channels. The Bluetooth LE radio provides developers a tremendous amount of flexibility, including multiple PHY options that support data rates from 125 Kb/s to 2 Mb/s, multiple power levels, from 1mW to 100 mW, as well as multiple security options up to government grade.

Bluetooth LE also supports multiple network topologies, including a point-to-point option used for data transfer, a broadcast option used for location services and a mesh option used for creating large-scale device networks.

De HC06 werkt volgens de EDR en de HM10 werkt volgens de BLE standaard.

Hoe kan je tegen zo een grote snelheid data versturen?

Dit kan door modulatie technieken toe te passen.



In elke Bluetooth module zit een modulator/demodulator

Deze zet de originele datastream om in een verzendbaar signaal, op hogere snelheden

Modulation	GFSK	GFSK, π/4 DQPSK, 8DPSK

We spreken hier van baud en baudrate. Een **baud is 1 element.**

Het **aantal elementen per seconde is baudrate**.

Bij RS232 gebruiken we meestal een baudrate van 9600bps. Hier is **1 element gelijk aan 1 bit**. Dus we hebben geen modem nodig om de snelheid om te zetten.

Bij bluetooth willen we veel hogere snelheden halen. Hier gebruiken we bijvoorbeeld **8DPSK.**

Dit staat voor **8 combinaties** die we kunnen maken in het **veranderen van de fase van een element t.o.v. het vorige element** (DPSK = differential Phase Shift Keying). Elk element is een combinatie van 3 bits (2^3 = 8). Had je enkel PSK dan was de faseverschuiving steeds t.o.v. de draaggolf verandert.

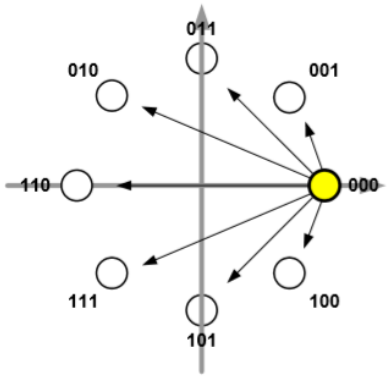
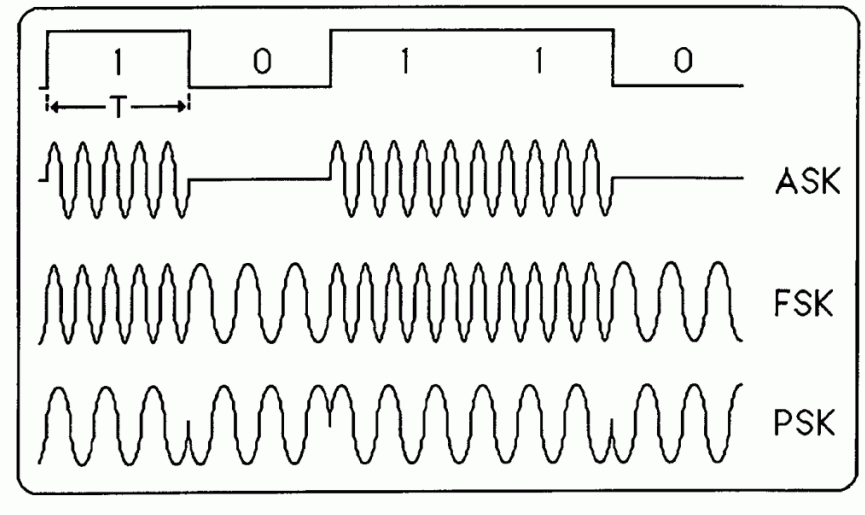


Figure 10.3: 8DPSK Constellation Pattern

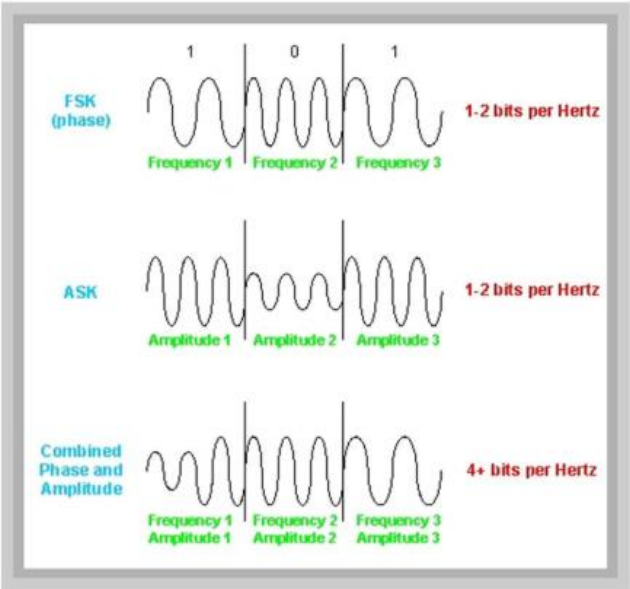
Bit Pattern	Phase Shift
000	0
001	π/4
011	π/2
010	3π/4
110	π
111	-3π/4
101	-π/2
100	-π/4



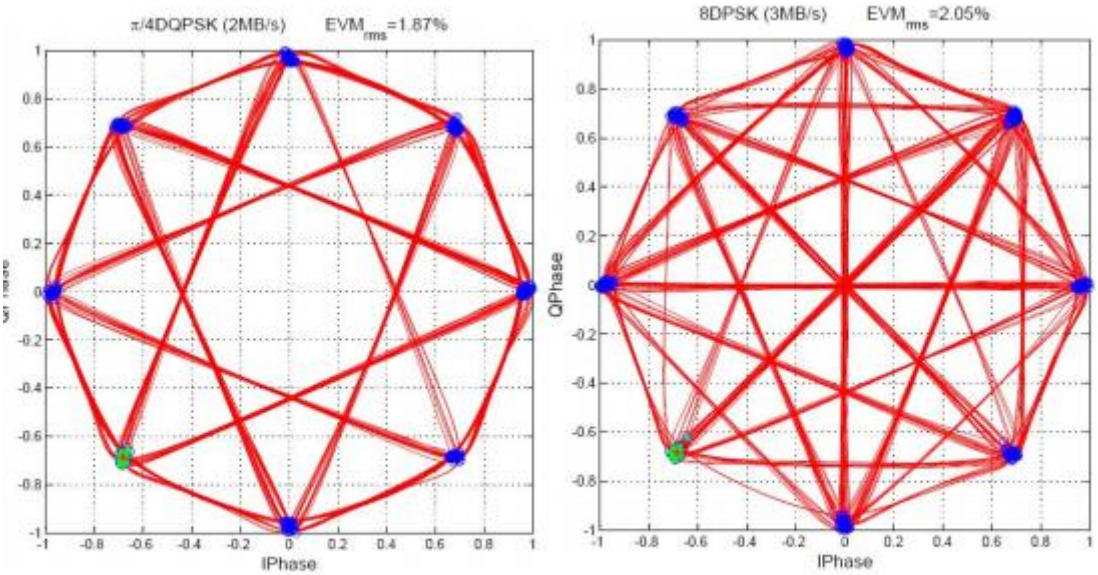
Soorten shift keying (amplitude, frequentie en fase)

Bluetooth Modulation Types

- **GFSK**
  - original Bluetooth
- **pi/4 QPSK**
- **D8PSK**



Nog eens een samenvatting van verschillende bluetooth modulatie technieken.



a.) π/4DQPSK (2MBPS) b.) 8DPSK (3MBPS)

Fig. 10 Measured constellation diagrams

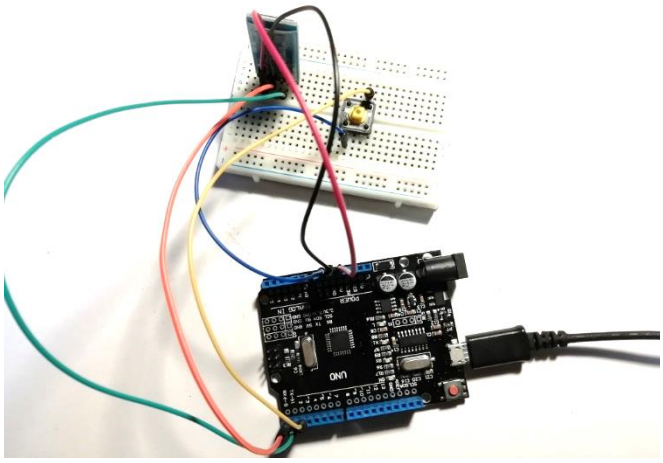
Dit constellatie diagram geeft aan welke punten er allemaal gebruikt worden tijdens de modulatie.

Zo kan men controlleren of het signaal netjes binnen de specificaties zit i.v.m. amplitude en fase.

Praktische oefening met bluetooth:

bluetooth_demo_serial

```
//geen virtuele poort versie
//RX = 0 (UNO) -> TX HC06 (uittrekken draadje tijdens download code!)
//TX = 1 (UNO) -> RX HC06

int LedGL = 4;

void setup()
{
  // set digital pin to control as an output
  pinMode(LedGL, OUTPUT);
  // set the data rate for the SoftwareSerial port
  Serial.begin(9600);
  // Send test message to other device (onze GSM)
  Serial.println("Hello from Arduino");
}
char a; // stores incoming character from other device
void loop()
{
  if (Serial.available())
  // if text arrived in from BT serial...
  {
    a=(Serial.read());
    if (a=='1')
    {
      digitalWrite(LedGL, HIGH);
      Serial.println("LED on");
    }
    if (a=='2')
    {
      digitalWrite(LedGL, LOW);
      Serial.println("LED off");
    }
    if (a=='?')
    {
      Serial.println("Send '1' to turn LED on");
      Serial.println("Send '2' to turn LED off");
    }
    // you can add more "if" statements with other characters to add more commands
  }
}
```



Test deze code uit. Merk op dat we hier geen SoftwareSerial library gebruiken (zie Arduino basis cursus) en **rechtstreeks via de TX RX poort** gaan van de UNO. Als je de code download moet je even de RX draad van de bluetooth loskoppelen.

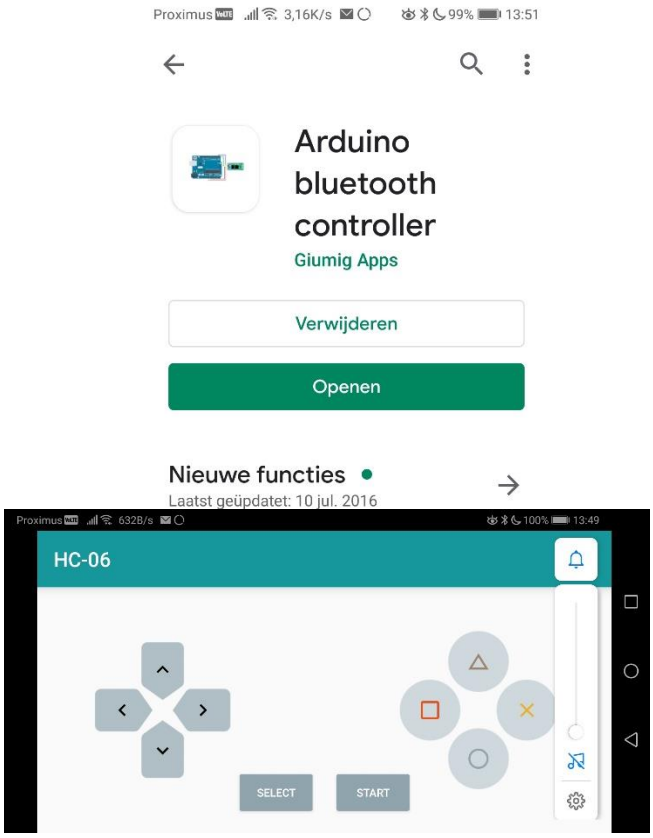
Indien je nu met de Logic Analyzer zou meten op RX en TX kan je weer dezelfde signalen meten als we reeds zagen in het begin van dit hoofdstuk.

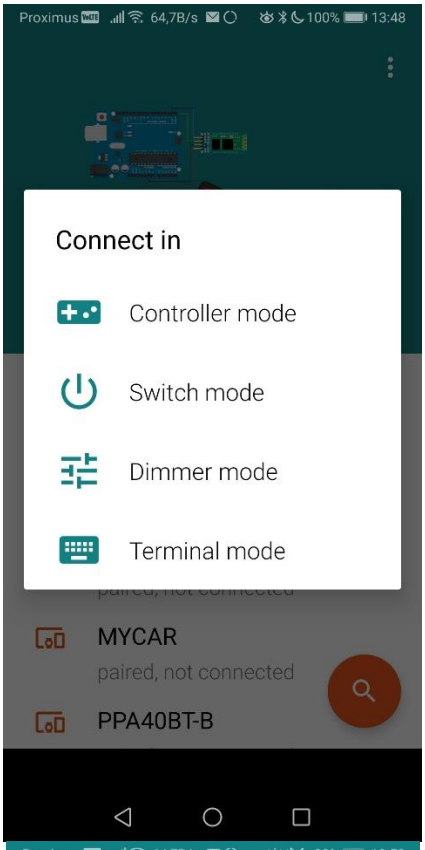
Op pin 4 hebben we een test LED voorzien.

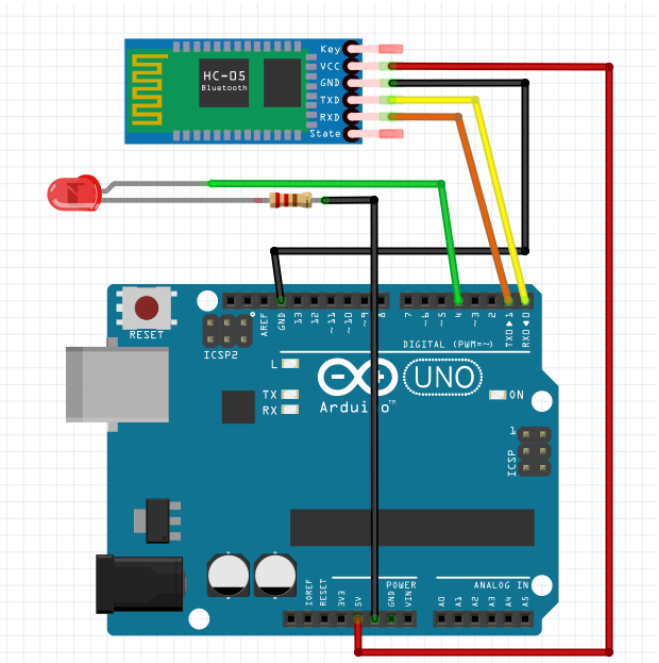
Vergeet niet van eerst de module te koppelen met “1234”.

Je kan de bluetooth module aansturen via bijvoorbeeld de app “**Arduino bluetooth controller**”.

Hier wijs je aan elke knop een karakter toe, overeenkomende met jouw programma.

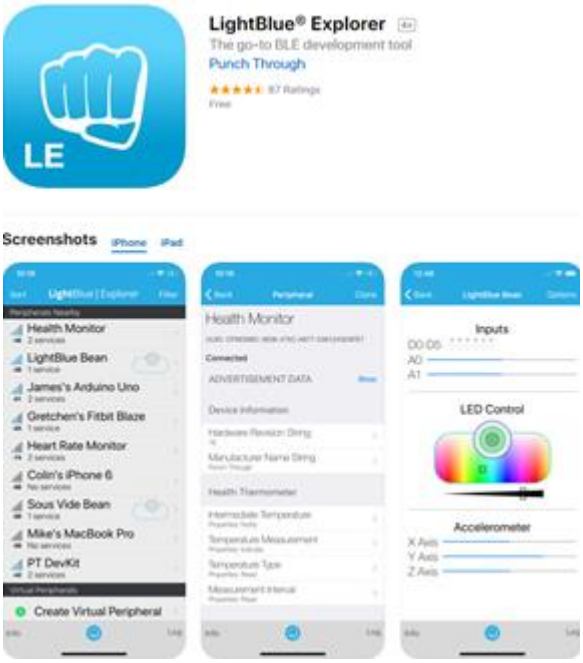






Indien je een Apple GSM hebt moet je eerst de app **LightBlue Explorer** installeren alvorens je de HM10 module kan koppelen. Anders ziet de Apple de LBE module niet.

Met **de iPhone volg je best de volgende stappen**: **HM-10** toont zich wel eens als **JDY-09-V4.3 of HC05** op het scherm.

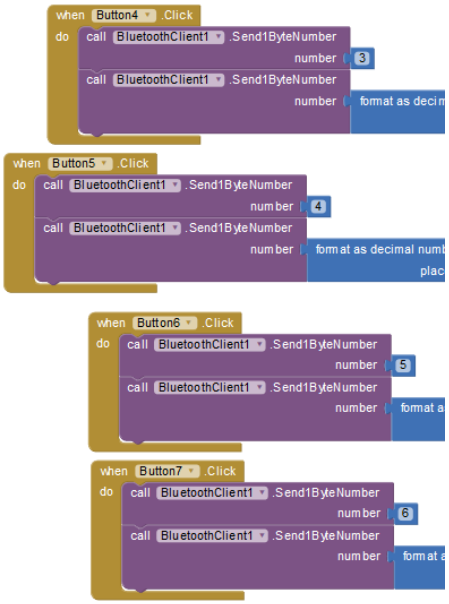


1) Download de app
2) Maak verbinding met de bluetooth module
3) Kies de laatste optie op dit scherm (onderste)
4) Je komt nu op een nieuw scherm druk rechts vanboven op hex
5) Druk nu op de "min" en het aantal veranderd naar ∞
6) Druk op null
7) Ga terug naar het vorige scherm
8) Druk op "write value" hier kan je de gewenste Commando's ingeven.

Extra uitdaging:

Je kan proberen een eigen app te bouwen met de MIT online software APPINVENTOR.

http://appinventor.mit.edu/



De software is in scratch stijl en is heel makkelijk aan te leren.

2 Timers leren gebruiken

(herhaling 5EE)

Onze Arduino heeft een oscillator met een **16MHz kristal** aan boord. Het X-tal levert een nauwkeurigheid op van 1:1000 000. Om het anders uit te drukken: de klok zal hoogstens een paar seconden per maand achter gaan lopen.

De klok die we hier willen maken heeft **een timer** die nauwkeurig genoeg moet zijn. Deze klok kan dan ingezet worden bij huishoudelijke klussen.

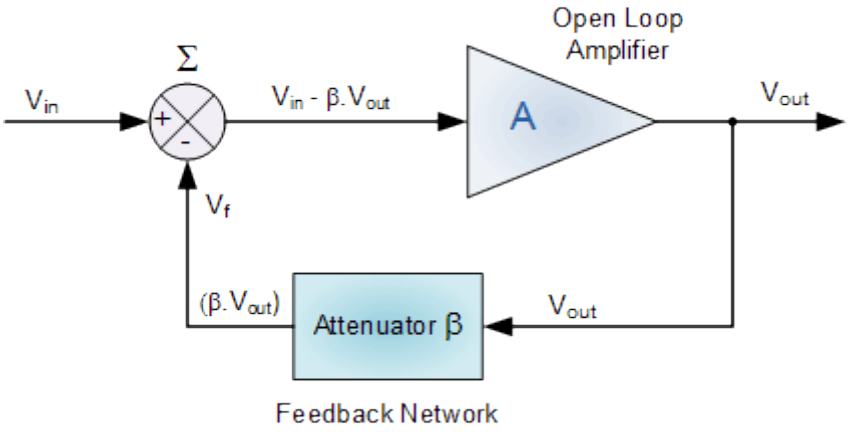
Merk op dat een oscillator beïnvloed kan worden door temperatuur, variatie in belasting, verandering in gelijkstroomspanning enz…

Een **kwarts-kristaloscillator** kan deze **oscillatorstabiliteit het beste** verkrijgen.

2.1 Wat zijn de voorwaarden om een oscillator te maken?

Wanneer kan een schakeling gaan oscilleren? Daarvoor moet er aan enkele voorwaardes voldaan worden:

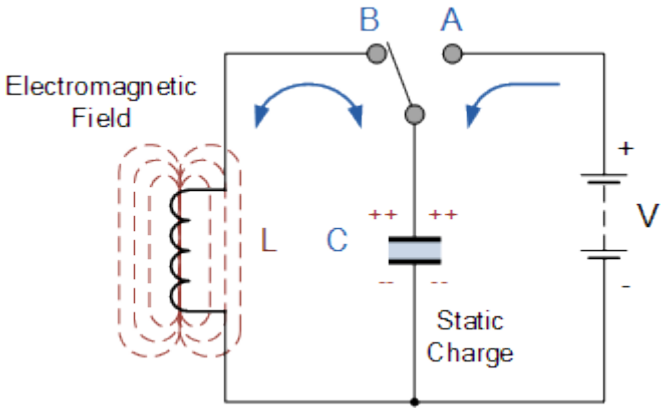
1. De totale versterking A . B van gans de schakeling moet 1 zijn.
2. De rondgaande faseverschuiving moet 0 zijn.



A = de versterking van de schakeling

B = de verzwakking door de frequentie selectieve LC tank

Hoe werkt een LC tank?



De schakeling bevat een spoel en een condensator.

De condensator slaat energie op in de vorm van een elektrostatisch veld. De spoel slaat energie op in de vorm van elektromagnetisch veld.

Als we de knop in stand A zetten wordt de condensator geladen tot de spanning V. Als de condensator volledig is opgeladen zetten we de knop in stand B.

De condensator ontlaadt nu in de spoel. De spanning van de condensator zakt en de stroom van de spoel stijgt. De volledig ontladen spanning van de condensator wordt nu als een elektromagnetisch veld opgeslagen in de spoel.

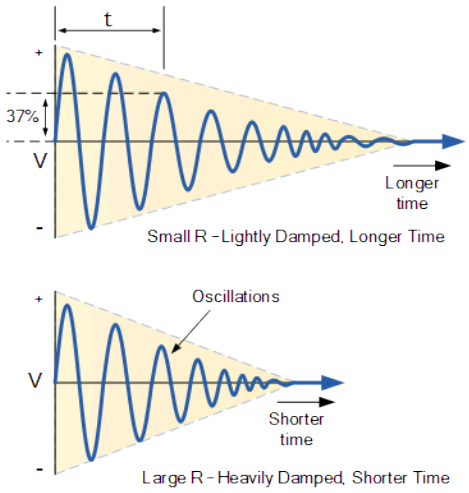
Als er nu geen externe spanning zit in het circuit, dan gaat het magnetisch veld van de spoel terug afnemen. Een tegen emk wordt geïntroduceerd en houdt de stroom in de oorspronkelijke richting.

De stroom laadt nu de condensator weer op totdat het magnetisch veld van de spoel helemaal verdwenen is. De energie is dus nu teruggekeerd naar de condensator. De polariteit van de condensator is echter omgekeerd.

Nu begint de condensator terug met ontladen in de spoel en het hele proces herhaald zich opnieuw. De polariteit van de spanning verandert constant en de energie wordt in een sinusvormige spanning geproduceerd.

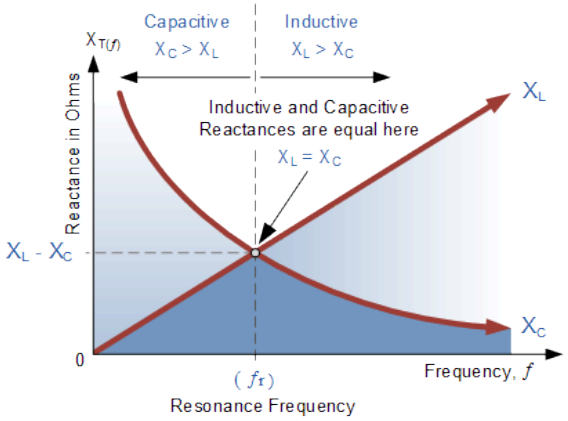
Natuurlijk zijn er energieverliezen in de componenten en zal de energie na verloop van tijd nul zijn.

De LC tank zorgt dus voor **demping** van het signaal.



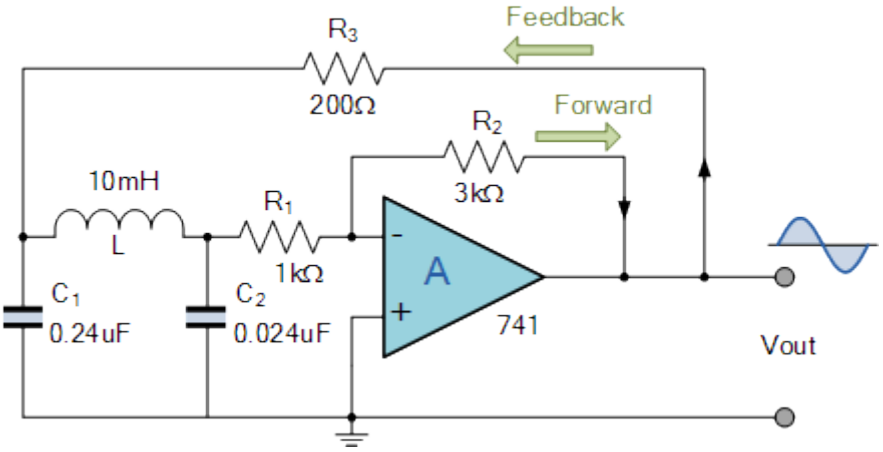
De frequentie hangt af van de **inductantie XL en de capacitive reactantie XC** van in de LC tank.

Er is 1 punt waar XC en XL gelijk zijn, en dat is de resonantiefrequentie fr.



$$f_r = \frac{1}{2\pi\sqrt{LC}}$$

Via onze versterker, gekoppeld aan de LC tank gaan we **telkens weer wat energie toevoegen**, zodat de LC tank kan blijven uitwisselen.

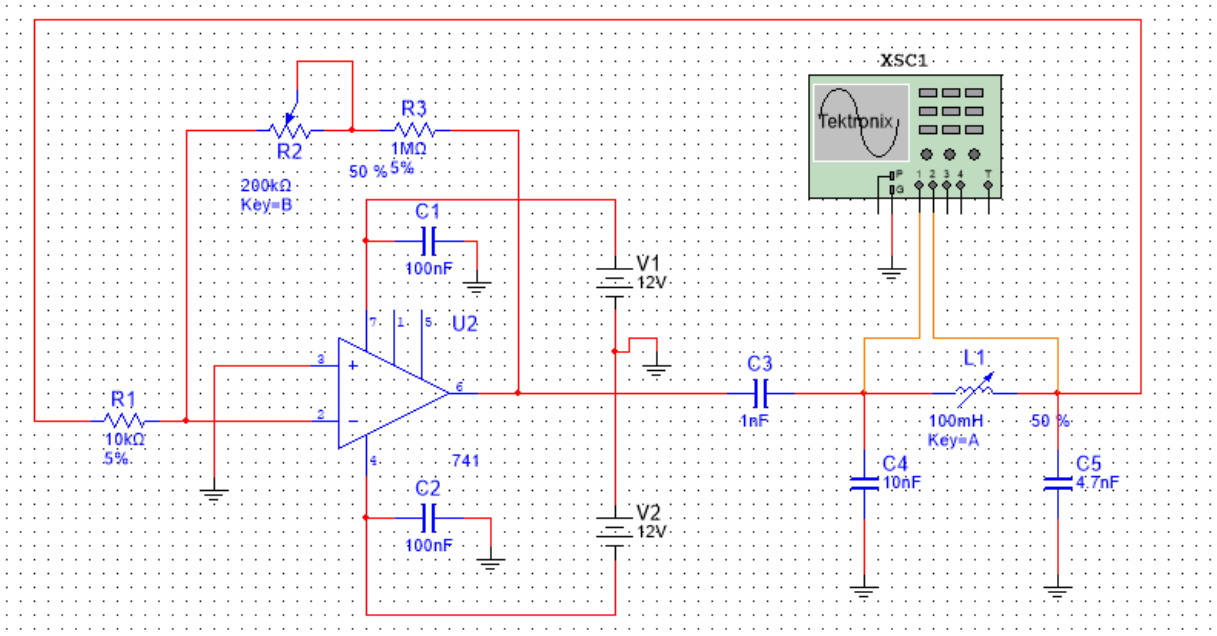


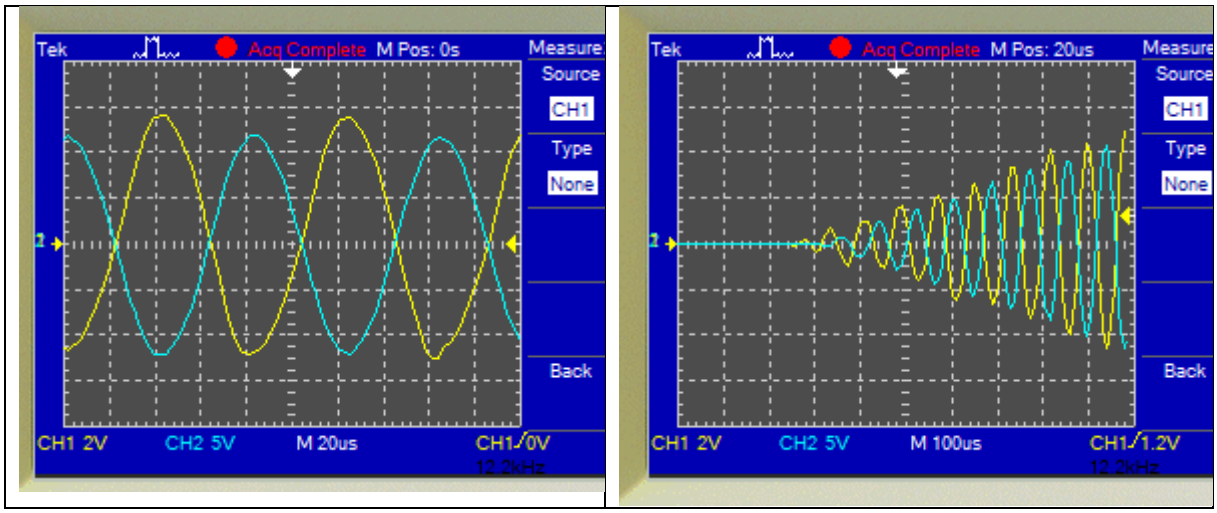
De **opamp versterker zorgt voor een -180 graden versterking**. De LC tank zorgt ook voor een **selectieve filtering van de frequentie** en een **verschuiving van 180 graden**. Dus de totale faseverschuiving is 0 graden. Als je R2 regelbaar maakt kan je A . B = 1 afregelen en gaat de oscillator constant dezelfde frequentie produceren.

Hoe start dan de oscillator?

Eerst is A . B > 1 en zal uit **de ruis van de voeding** de frequentie oppikken die is ingesteld. Stillaan wordt het signaal groter en gaat A . B steeds meer naar 1 toewerken.

Op een bepaald moment is dan **A . B = 1** en blijft de frequentie constant.



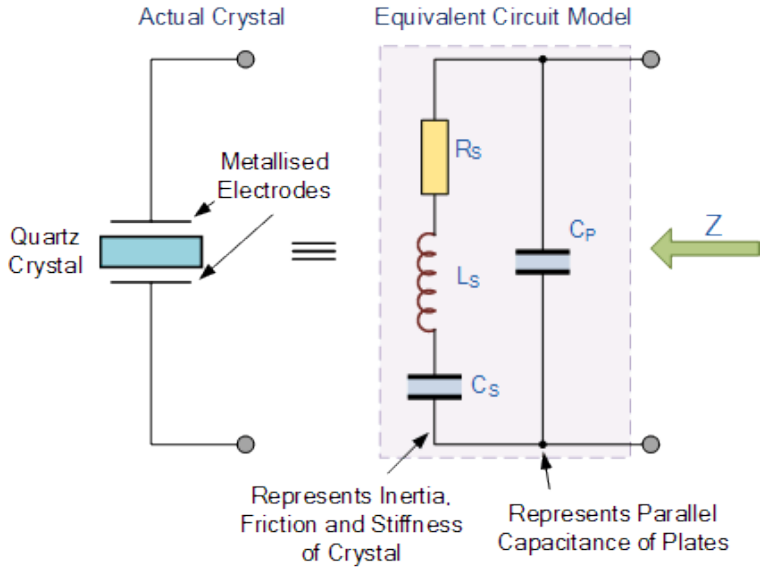


➔ **Taak 1:** teken de LC oscillator in Multisim en simuleer de bovenstaande signalen. Onderzoek de invloed van L1 (regelbare spoel als kristal) en de R2 (terugkoppelweerstand die de inverterende versterking regelt).

2.2 Hoe werkt een X-tal oscillator?

Als je het kwartskristal als een zeer dun schijfje van een spanningsbron voorziet, dan treed **het piëzo-elektrisch effect** op. Door een elektrische lading wordt een mechanische kracht geproduceerd op het kristal en verandert het van vorm. Andersom levert een mechanische kracht ook een elektrische lading op.

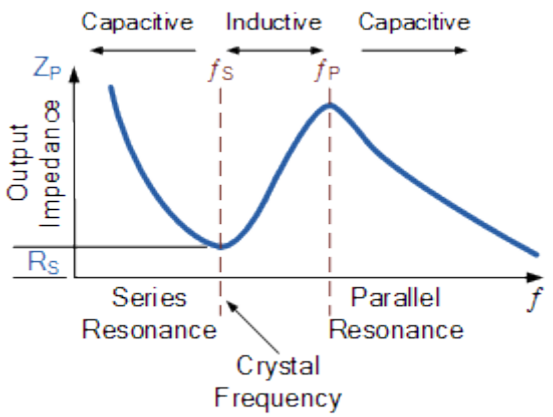
De grote en vorm, hoe het kristal is gesneden, bepaald de oscillatiefrequentie.



Een quartz kristal kan je ook tekenen als een elektrische kring van een condensator in parallel met een serieketen van weerstand, spoel en condensator.

De RLC keten stelt de mechanische trilling van het kristal voor, terwijl de Cp de elektrische aansluitingen zijn van het kristal met de buitenwereld.

Een kristal heeft 2 resonantiefrequenties: **serie en parallelresonantie.**



$$R = R \quad \text{and} \quad X_{LS} = 2\pi f L_S$$

$$X_{CS} = \frac{1}{2\pi f C_S} \quad \text{and} \quad X_{CP} = \frac{1}{2\pi f C_P}$$

$$Z_S = \sqrt{R_S^2 + \left(X_{LS} - X_{CS}\right)^2}$$

$$\therefore Z_P = \frac{Z_S \times X_{CP}}{Z_S + X_{CP}}$$

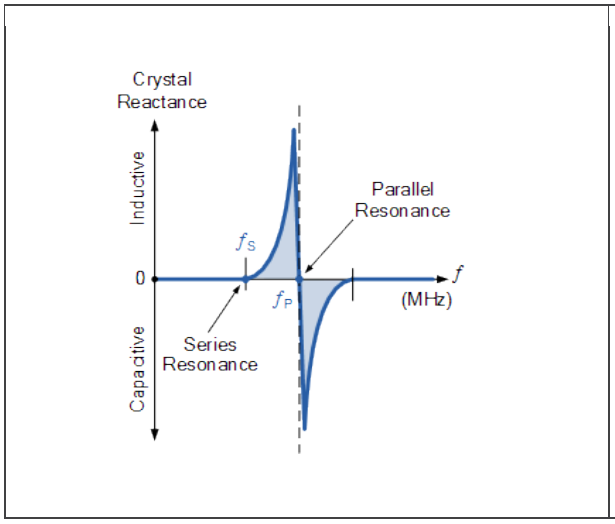
Bovenstaande curve geeft mooi aan hoe het kristal zich gedraagt. Bij een frequentie lager dan fs gedraagt het kristal zich als een **capaciteit,** net zoals wanneer de frequentie groter is als fp.

Tussen beide resonantie frequenties lijkt het kristal spoel gedrag te vertonen.

Op het moment van fp bereikt het kristal zijn maximum impedantie waarde Zp. Nu wordt er een **afgestemde LC tank** gecreëerd. Deze kunnen we verder in een LC oscillator inschakelen.

We moeten alleszins ons kristal afstemmen op 1 van de 2 frequenties, want op allebei tegelijk kan niet. **Afhankelijk van de eigenschappen van de schakeling rondom het het kristal** gedraagt deze zich als een spoel, condensator, serieresonantiekring of parallelresonantiekring. Dit kan voorgesteld worden door volgende curve.

Kristal reactantie t.o.v. de frequentie:



$$X_S = R^2 + \left(X_{LS} - X_{CS}\right)^2$$

$$X_{CP} = \frac{-1}{2\pi f C_P}$$

$$X_P = \frac{X_S \times X_{CP}}{X_S + X_{CP}}$$

$$f_S = \frac{1}{2\pi\sqrt{L_S C_S}}$$

Serie resonantie frequentie

$$f_P = \frac{1}{2\pi\sqrt{L_S\left(\frac{C_P C_S}{C_P + C_S}\right)}}$$

Parallelle resonantie frequentie

Oefening op berekenen van fs en fp:

Een kwartskristal heeft de volgende waarden: Rs = 6,4Ω , Cs = 0,09972pF en Ls = 2,546mH . Als de capaciteit over zijn terminal, Cp wordt gemeten bij 28,68 pF.

Bereken dan de fundamentele oscillatiefrequentie van het kristal en zijn secundaire resonantiefrequentie.

De serie resonantiefrequentie van de kristallenreeks, f_S

$$f_S = \frac{1}{2\pi\sqrt{L_S C_S}} = \frac{1}{2\pi\sqrt{2.546\text{mH} \times 0.09972\text{pF}}}$$

$$f_S = \frac{1}{2\pi\sqrt{0.002546 \times 99.72 \times 10^{-15}}} = 9.987\text{MHz}$$

De parallelle resonantiefrequentie van het kristal, f_P

$$f_P = \frac{1}{2\pi\sqrt{L_S\left(\frac{C_P C_S}{C_P + C_S}\right)}}$$

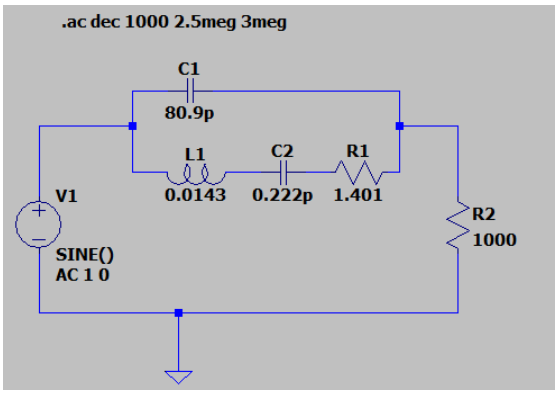
$$f_P = \frac{1}{2\pi\sqrt{2.546\text{mH}\left(\frac{28.68\text{pF} \times 0.09972\text{pF}}{28.68\text{pF} + 0.09972\text{pF}}\right)}}$$

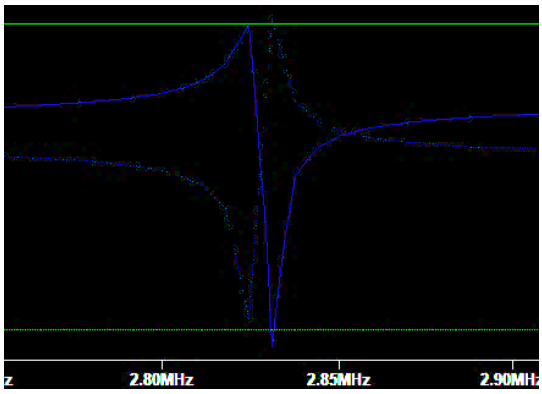
$$f_P = 10{,}004{,}996\text{Hz or } 10.005\text{MHz}$$

We gaan dit **kristal** nu simuleren in **een LTspice Analog Device omgeving** (of in Multisim/Tina TI).

Je kan deze simulator gratis downloaden via volgende link:

https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html#





De **kwaliteitsfactor van het kristal** wordt gegeven door het berekenen van **de Q-factor**. Bij de serieresonantie toegepast op de vorige oefening is deze :

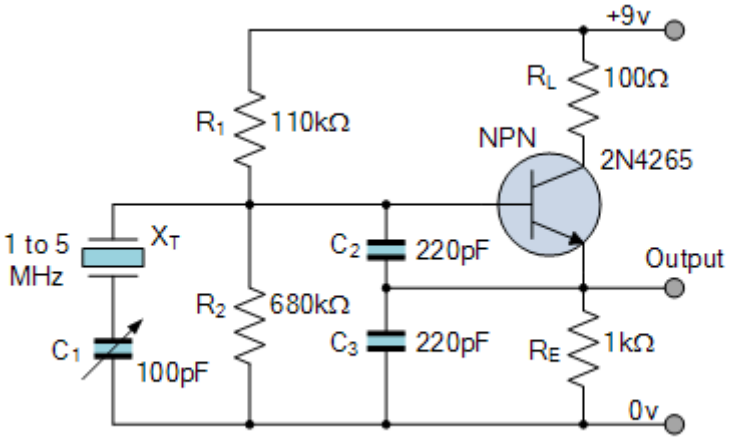
$$Q = \frac{X_L}{R} = \frac{2\pi f L}{R} = \frac{2\pi \times 9.987 \times 10^{6} \times 0.002546}{6.4}$$

$$Q = 24966 \text{ or } 25{,}000$$

Dit is **behoorlijk hoog** t.o.v. een gewone LC oscillator waar we nog niet aan 1000 komen.

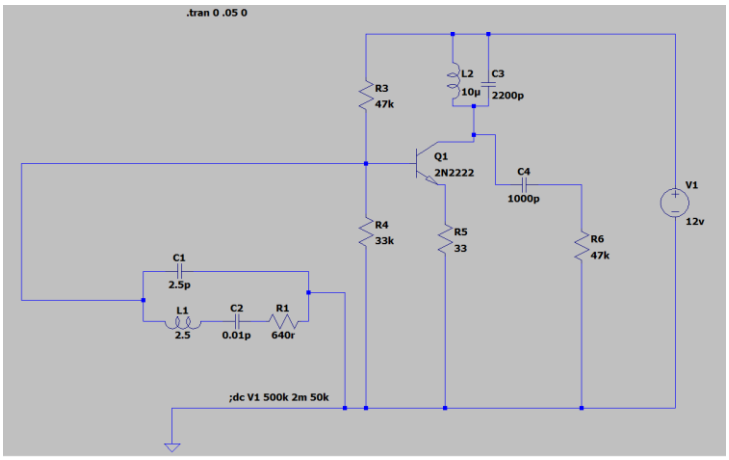
In een kwarts kristal oscillator wil de kristal altijd **oscilleren op de parallel frequentie** wanneer je deze aansluit op een spanningsbron.

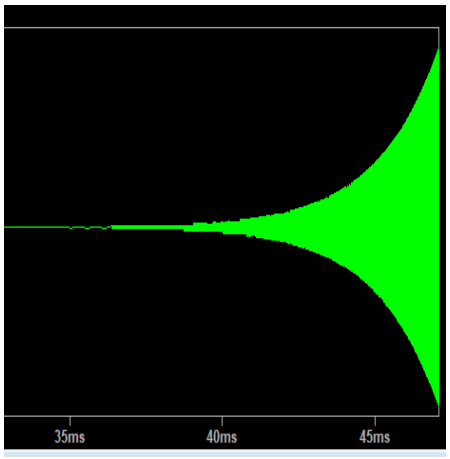
Onderstaand schema is een voorbeeld van een **Colpitts kristal oscillator.**



De oscillator is ontworpen rond **een gemeenschappelijke collector versterker**.

R1 en R2 regelen de DC waarde aan de basis, terwijl RE de uitgangsspanning instelt.

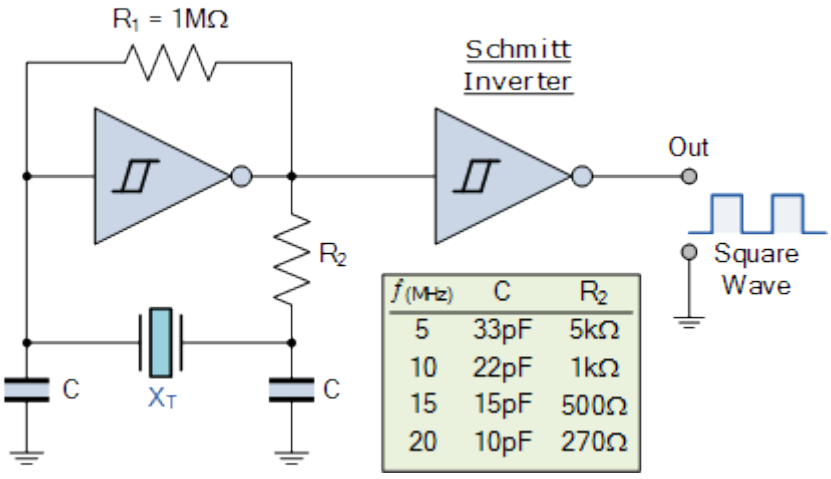




Spice voorbeeld van de X-tal in een open collector schakeling.

Zie hoe het X-tal **uit de ruis stilletjes op gang komt**.

Dikwijls wordt in een microcontroller de volgende opstelling gebruikt van een **CMOS kristal oscillator**:



f (MHz)	C	R2
5	33pF	5kΩ
10	22pF	1kΩ
15	15pF	500Ω
20	10pF	270Ω

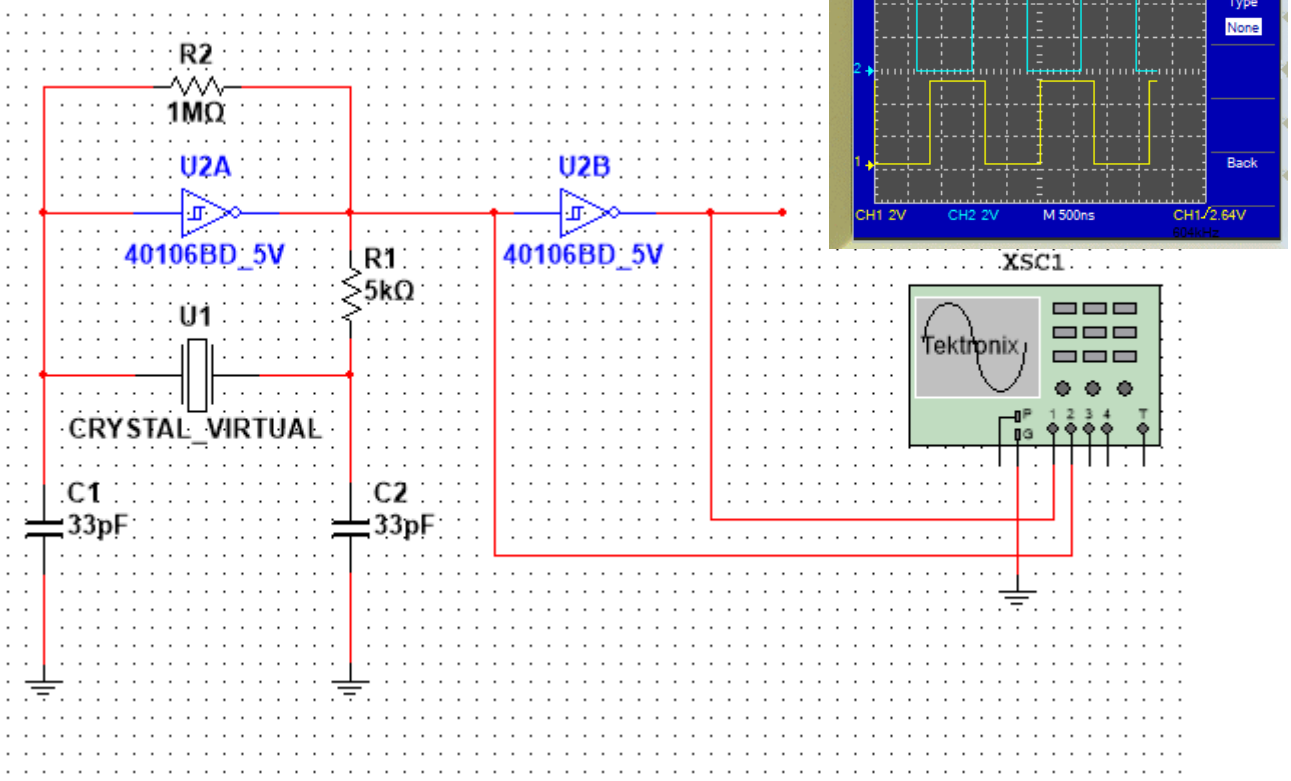
Deze oscillator is zodanig ingesteld dat ij werkt bij de serie resonantie frequentie fs.

De 1M ohm weerstand zorgt ervoor dat de oscillator ingesteld is in een gebied waar de versterking hoog is.

De rechtse Schmitt-trigger zet het sinus signaal mooi **gebufferd** voor een belasting om in een bloksignaal.

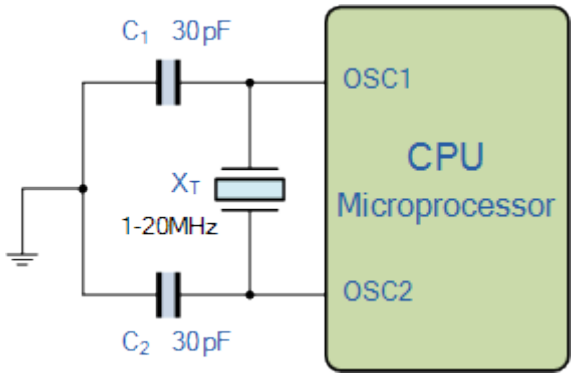
De linkse inverter levert een **180 graden faseverschuiving**. Het kristal circuit levert ook 180 graden verschuiving. Zodoende is de **rondgaande faseverschuiving 360 graden**, wat nodig is om een oscillator te bekomen (zie voorwaardes).

➔ **Taak 2:** simuleer deze oscillator in Multisim.



De microcontroller oscillator:

Uiteindelijk gaan we een **kristal samen met 2 ceramische condensatoren** aansluiten op de oscillatorpinnen van de CPU van de microprocessor. Tijdens het solderen van je eigen UNO gaan we dit er zeker over hebben.



In de datasheet geeft de fabrikant aan wat de verschillende waardes moeten zijn om een bepaalde frequentie te bekomen van onze kwarts kristal oscillator. De rest van de schakeling zit in de chip.

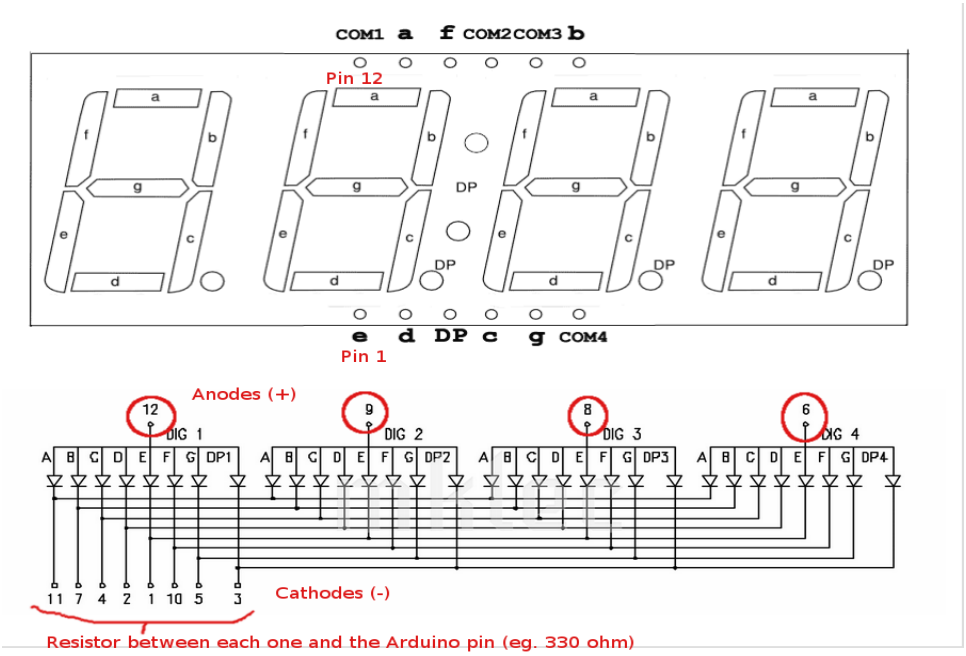
Hier kan je opnieuw de voorgaande fs en fp formules op loslaten.

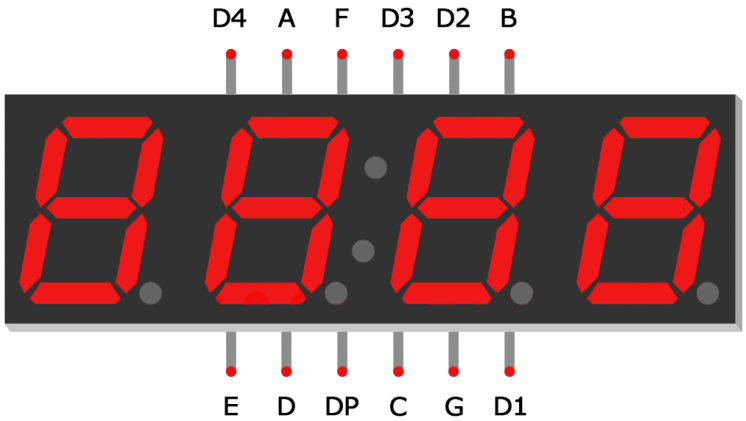
2.3 Gebruik van de timer

Nu we weten waar de **nauwkeurige klok** in onze microcontroller vandaan komt, gaan we deze inzetten in onze Arduino projecten.

We gaan **een 4x7 segmenten display gebruiken om cijfers op te tonen**. Elk **meetinstrument** met 7 segment displays maakt van deze techniek gebruik. Hier kan je dan o.a. een **klok** mee bouwen.

Hiervoor gaan we beroep moeten doen op een **timer in de microcontroller**. Deze timer is met onze klok verbonden en kan **heel nauwkeurig** ingesteld worden. De techniek die volgt is dus **nauwkeuriger dan wanneer we enkel met delay()** commando's zouden werken.





Ons display **CL5642BY30** is **common cathode (CA).** Dit wil zeggen dat we eerst de **CA** moeten aanzetten **per digit (D1 – D4)** alvorens we de LED DISPLAY onderdelen **A – G** gaan laag maken.

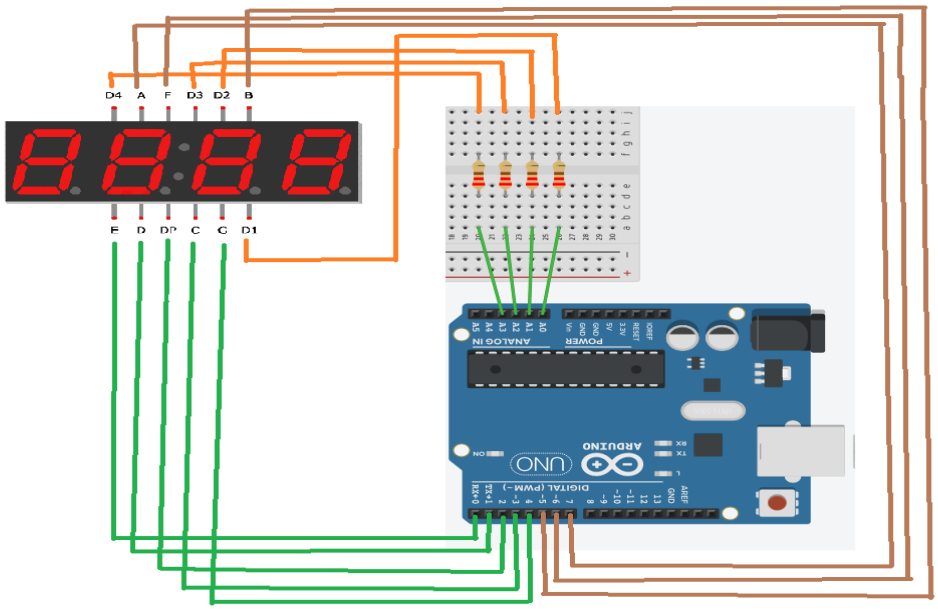
Opdracht:

- Plaats een 4x7 segment display op een breadboard.
- Stel de regelbare voeding in op slechts **2V** (spanning van 1 LED zonder voorschakelweerstand).
- Plaats de +2V aan 1 van de CA pinnen (D1 – D4).
- Plaats nu de massa bij 1 digit (7-segment) telkens aan een ander segment (A-G en DP).
- Test zo de verschillende segmenten.

Nu gaan we het 4x7-segment display aansluiten op de Arduino.

Merk op dat we nu **220 ohm gaan plaatsen in de CA van elke digit**. Dit is om de stroom te beperken wanneer we op +5V werken. I = U / R = 5V – 2V / 220 ohm = 13.6mA per LED. Merk op dat we altijd maar 1 segment van 1 van de digits maar zullen aanzetten. Zo moeten we geen extra weerstanden en transistoren gebruiken om het stroomverbruik aan te kunnen. Ze noemen dit de **"dubbele" multiplexmethode**.

Maak het volgende schema:

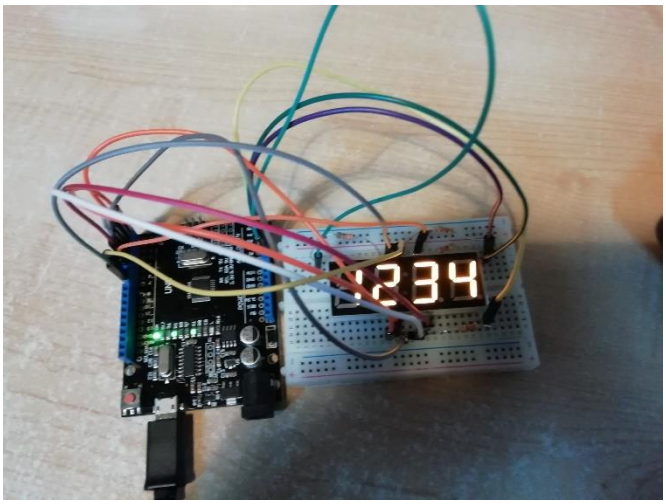


Toewijzing Arduino pennen aan het display

Arduino-pen	0	1	2	3	4	5	6	7	A0	A1	A2	A3
Display-pen	1	2	3	4	5	7	10	11	12	9	8	6
Segment	e	d	dp	c	g	b	f	a	D1	D2	D3	D4

Pen 1 van het display zit aan de segment E. Dan tel je , net als bij een IC van links naar rechts, en dan de hoek om (van D1 naar B) enz ….

Opmerking: de aansluitingen volgorde A0 – A3 zou ook A3 – A0 kunnen zijn. Nog verder testen !

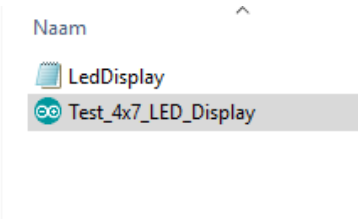


Onze code bestaat in dit geval uit 2 bestanden:

Het hoofdprogramma **“4x7_LED_DISPLAY.ino”** is heel kort gehouden.

Het tweede deel **“LedDisplay.h”** is een header file en bevat de driver van het display.

Beide files plaats je in **dezelfde directory**.



Wanneer je het hoofdprogramma opent gaat **automatisch de header file mee open** gaan in de Arduino IDE.

Als 2de bibliotheek van de bovenstaande timer oefening includen we onze eigen LedDisplay.h file.

In de setup() gaan we het **display initialiseren**. Deze functie is in de LedDisplay.h file uitgelegd.

Tenslotte sturen we **“1234” naar het display** in de loop(). Ook deze is in de LedDisplay.h file uitgelegd.

Test_4x7_LED_Display | LedDisplay.h

```
// Listing 6-2
// Test 4x7 LED Display
#include <TimerOne.h>
#include "LedDisplay.h"  // include our display driver

void setup() {
  initLedDisplay();
}

void loop() {
  numberOutput(1234);
}
```

In het hoofdprogramma wordt de "**TimerOne.h**" bibliotheek toegevoegd. We gaan dus, zoals eerder gezegd, gebruik maken van timer mogelijkheden. Zie library op Smartschool!

Timer 1 wordt nog al eens gebruikt om **PWM periodes en frequenties** nauwkeurig in te stellen. Ook andere features zoals een **timer overflow interrupt actie** kunnen hiermee gemaakt worden (hierover verder meer).

De Arduino **ATMEGA328P heeft 3 timers** waar we kunnen gebruik van maken.

Timer 0 en Timer 2 zijn beide 8-bit timers. Zij tellen van 0 tot 255. **Timer 1** is een 16 bits timer (telt van 0 tot 65535).

Timer 0 wordt nog al eens makkelijk ingezet bij het commando millis().

Als je de TimerOne.h file open doet in een kladblok dan zie je het volgende staan:

```
#define RESOLUTION 65536    // Timer1 is 16 bit

class TimerOne
{
  public:

    // properties
    unsigned int pwmPeriod;
    unsigned char clockSelectBits;
        char oldSREG;                                   // To hold S

    // methods
    void initialize(long microseconds=1000000);
    void start();
    void stop();
    void restart();
        void resume();
        unsigned long read();
    void pwm(char pin, int duty, long microseconds=-1);
    void disablePwm(char pin);
    void attachInterrupt(void (*isr)(), long microseconds=-1);
    void detachInterrupt();
    void setPeriod(long microseconds);
    void setPwmDuty(char pin, int duty);
    void (*isrCallback)();
};
```

In deze file kan je **alle functies** terugvinden dewelke men reeds heeft geschreven voor **Timer 1**.

Als je wil weten, op c-code niveau, wat er nu zit achter die functies, moet je de **TimerOne.cpp** eens openen in een kladblok (zit ook in dezelfde map van de library TimerOne):

Zo krijg je dan voor de functie TimerOne::**start()** de volgende c-code:

```
void TimerOne::start()  // AR addition, renamed by Lex to reflect it's actual role
{
  unsigned int tcnt1;

  TIMSK1 &= ~_BV(TOIE1);        // AR added
  GTCCR |= _BV(PSRSYNC);               // AR added - reset prescaler (NB: shared with all 16 bit timers);

  oldSREG = SREG;                              // AR - save status register
  cli();                                              // AR - Disable interrupts
  TCNT1 = 0;
  SREG = oldSREG;                      // AR - Restore status register
        resume();
  do {  // Nothing -- wait until timer moved on from zero - otherwise get a phantom interrupt
        oldSREG = SREG;
        cli();
        tcnt1 = TCNT1;
        SREG = oldSREG;
  } while (tcnt1==0);

//  TIFR1 = 0xff;                       // AR - Clear interrupt flags
//  TIMSK1 = _BV(TOIE1);              // sets the timer overflow interrupt enable bit
}
```

2.4 Wat is nu een timer?

Voor meer uitgebreide info kan je altijd terecht op:

https://maxembedded.wordpress.com/2011/06/22/introduction-to-avr-timers/

In de elektor (mei 2008 pg 72) kan je hier ook meer info over vinden.

Een timer zit **in zowat elk apparaat** met een microcontroller. Er zonder zou de wereld nu stoppen.

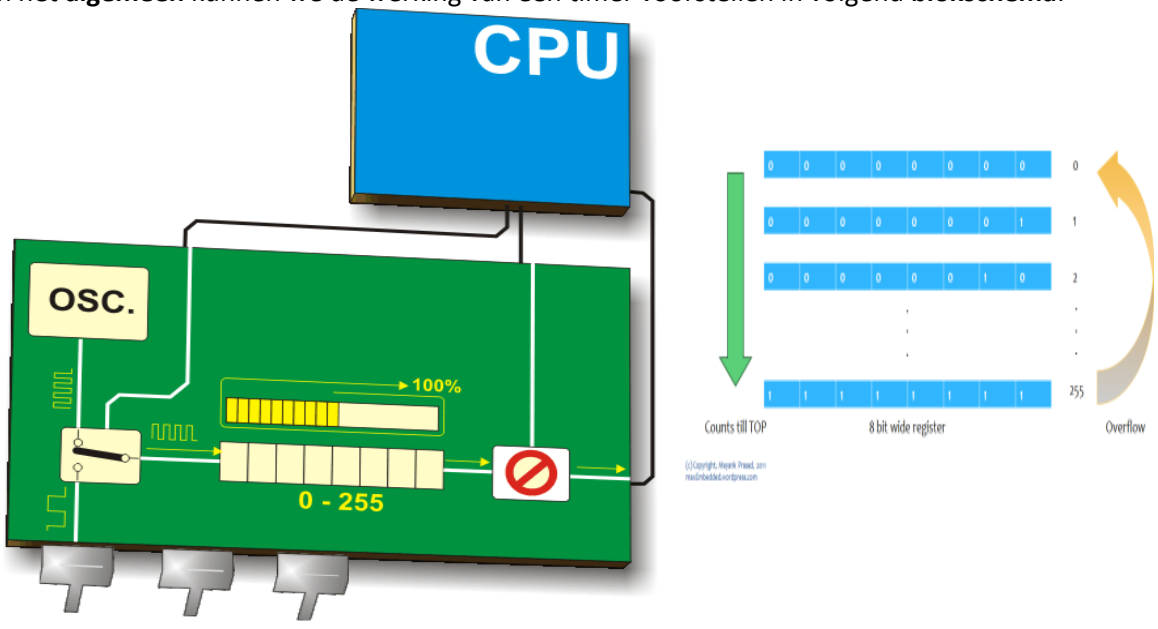
Deze kan handig zijn wanneer we bijvoorbeeld een chronometer of klok willen bouwen, of een exacte tijd willen meten.

We zitten hier in de range van **microseconden**.

Het probleem is dat de delay() functie geen nauwkeurige tijdsmeting uitvoert, daarom gaan we hier onderzoeken hoe we dit wel kunnen instellen en programmeren in de µC.

Onze timer 1 gaan we zo instellen dat deze een **intern interrupt** gegenereerd in de microcontroller.

In het **algemeen** kunnen we de werking van een timer voorstellen in volgend **blokschema**:

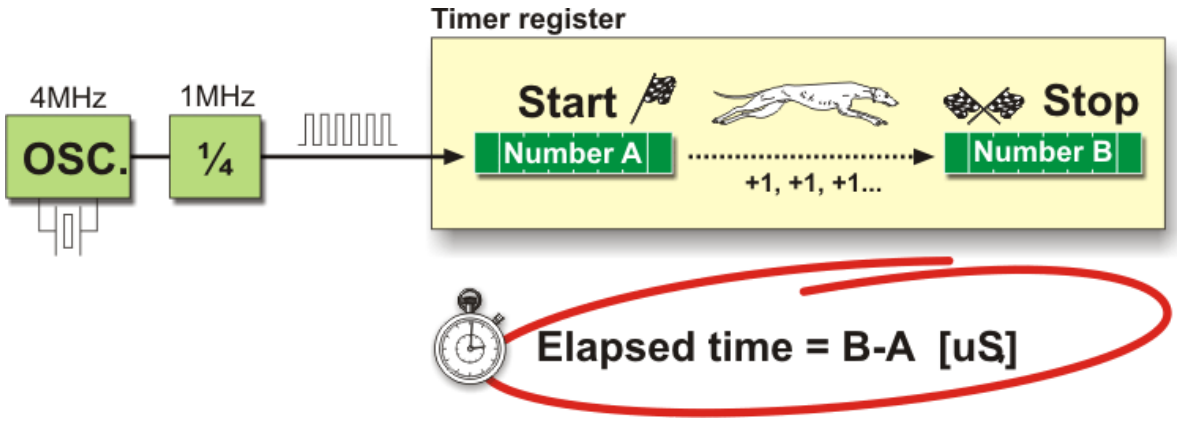


De **externe of interne klok** wordt softwarematig ingesteld via **de configuratiebits** van de processor. Dan wordt deze klok aangeboden aan een **teller (counter).** Deze 8 bits teller telt van 0 tot 255 en na een overflow (van 255 naar 0) telt hij opnieuw op. Indien het , zoals bij Timer1 een 16 bits teller is zal deze van 0 tot 65 535 tellen en daarna weer opnieuw.
Wanneer de **overflow** gebeurt (overgang max naar min waarde van teller) wordt kort **een puls** doorgegeven aan de CPU. Deze puls is de **timer interrupt**.

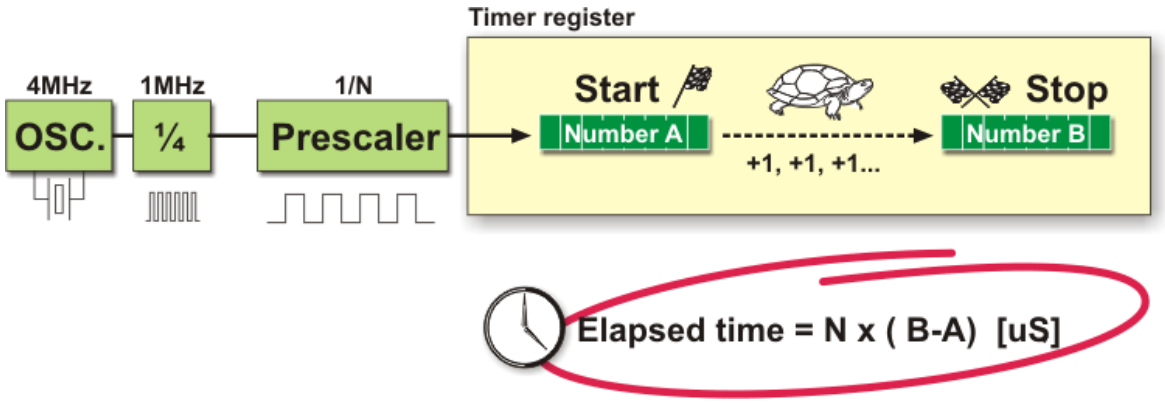
Dit **interrupt** gaat in onze c-code een **interrupt service routine** aanroepen en zodoende een **speciale routine** uitvoeren. Daarna keert de processor terug naar de code waar hij, vóór het interrupt, mee bezig was.

In onderstaande voorbeeld tekening kan je zien hoe het oscillator signaal van 4MHz intern gedeeld wordt door 4. Dit is omdat **een instructie gemiddeld 4 klokpulsen nodig heeft om 1 instructie af te werken**.

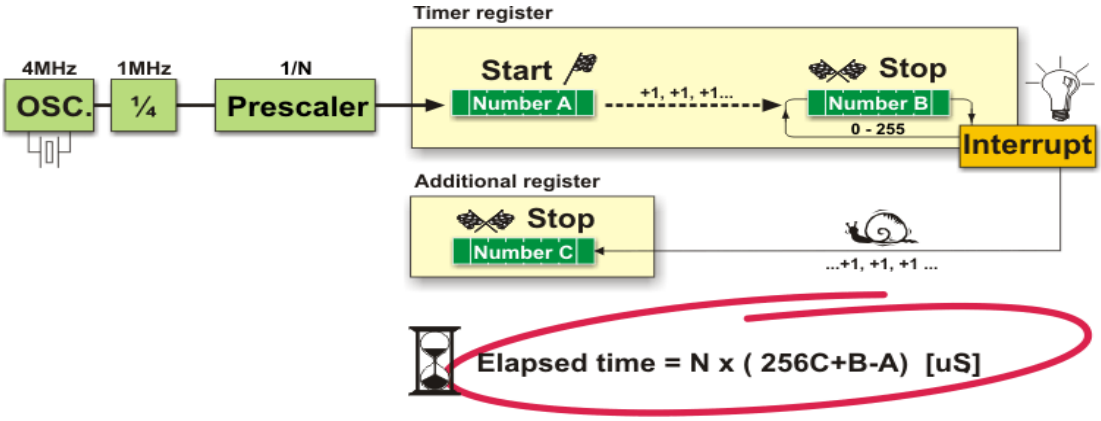
Het timer register start in principe bij 0 en **stopt bij het ingesteld getal of het max getal**. Zo kan je een bepaalde delay maken.



Omdat de snelheid van de oscillator echter zo groot is, zodat er veel te snel tot 255 (bij een 8 bits teller) zou geteld worden, moeten we de klok eerst via een **prescaler** herschalen naar een **grotere periode** of lagere frequentie. **We delen** met andere woorden de klok nogmaals door een **instelbare waarde** (vb 256) alvorens deze aan het **timer register** wordt aangeboden.



De volgende figuur maakt duidelijk hoe uiteindelijk, **na een overflow**, een **interrupt** gegenereerd wordt **door het timer register** en zo een ander register of **variabele kan verhoogd** worden in de c-code. Op een bepaald moment zijn dan **x aantal pulsen bereikt** en komt dit overeen met een **nauwkeurig** ingestelde tijd.
De snelle klok aan het begin kan dus omgezet worden in een tijd zoals wij hem graag willen instellen.



Wanneer we hetzelfde effect zouden willen bekomen met een delay() functie zou dit nooit zo'n nauwkeurige tijdsmeting opleveren.

Wat is dan een counter?

- ➔ Een counter is in feite een **teller** die aangestuurd wordt door **externe pulsen**, afkomstig van een sensor. **Er is geen constante tijd tussen de pulsen vereist.** Hier kan je uiteraard ook interrupts voor gebruiken (zie int0 en int1 bij de Arduino UNO). Zo kan je bijvoorbeeld flesjes tellen die aan een band passeren. Nadat er een bepaald aantal flessen voorbij zijn gekomen geeft de counter (ook een teller) dan een **intern interrupt en kan je een actie ondernemen.**
- ➔ Merk op dat een timer wel een **constante tijd tussen de pulsen** vereist waarbij de pulsen worden gestuurd door bijvoorbeeld een oscillator. Via een teller kan je dan een **bepaalde tijd afmeten.**

Rekenvoorbeeld:

Stel dat je een LED elke 10ms wil aan en uit laten knipperen. Hoe stel je dan de timer in?

De frequentie van de LED is dus f = 1 / T = 1/10 ms = 100Hz.

Stel dat je een extern kristal hebt van 4MHz. Dus we gebruiken ook een CPU frequentie van 4MHz. Het timer 1 register in de Arduino UNO (MEGA328P) telt van 0 tot 65 535.

T_CPU = 1 / f = 1 / 4M = 0.00025ms.

Indien je nu 10ms wil hebben moet je de volgende berekening maken:

Timer teller = (gewenste delay / T_CPU) – 1 = (10ms / 0.00025ms) – 1 = 39999

De klok heeft dus 39999 keren getikt voordat de delay van 10ms voorbij is.

De 8-bit timer kunnen we niet gebruiken. Deze gaat maar van 0 – 255. De 16 bit teller kan wel dienen. Zijn maximum is 2^16 – 1 = 65535.

Met de f_{CPU} = 4MHz en een 16 bits timer kunnen we dus maximum aan 16, 384 ms geraken. (0.00025ms x 65535 = 16.384ms)

Wat als we nu 20ms willen hebben?

Hiervoor gebruiken we dan de **prescaler.** We gaan dus niet de CPU frequentie verkleinen, maar we leiden een frequentie af van de 4MHZ klok. Door enkele bits in te stellen in het timer register kunnen we dit klaar krijgen (zie later).

Stel dus dat we de prescaler instellen zodanig dat we de **klok delen door 8**. 4MHz wordt dan 0.5MHz.

1 / 0.5MHz = 0.002ms. Willen we 20ms dan hebben we nu een timer teller nodig van

Timer teller = (20ms/0.002ms) – 1 = 9999

Deze waarde **past opnieuw** in de 16 bits timer.

Merk wel op dat de **nauwkeurigheid een beetje slechter is geworden.**

Bij de eerste timer kregen we ronde getallen: bijvoorbeeld 0.1125/0.00025 = 450

Bij de timer met prescaler moeten we afronden: bijvoorbeeld 0.1125/0.002 = 56.25

Deze timer kan dus ofwel 0.112 us of 0.114 us meten, maar niet wat er tussen staat. Dus de **nauwkeurigheid is lager geworden**.

Hoe een prescaler kiezen?

Stel dat we 184 ms delay wensen. We kunnen 4 soorten prescalers instellen (zie tabel).

Welke prescaler kiezen we dan?

Required Delay = 184 ms
F_CPU = 4 MHz

Prescaler	Clock Frequency	Timer Count
8	500 kHz	91999
64	62.5 kHz	11499
256	15.625 kHz	2874
1024	3906.25 Hz	717.75

Prescaler 8 is niet goed omdat de timer teller > 65535 gaat. Bij 1024 als prescaler krijgen we een komma, dus de nauwkeurigheid is slechter. Blijven 64 en 256 over. We kiezen dan de 64 omdat deze de grootste resolutie heeft. 256 kan je kiezen als de timer langer moet duren.

Wanneer de timer is afgelopen (= **overflow)** wordt er, indien je dat wenst, een **interrupt gegenereerd in de microcontroller.** Hierdoor wordt een **ISR (interrupt service routine)** aangeroepen en dus de **originele code even onderbroken**. Je kan het zien als een speciale subroutine. Nadat deze is afgelopen spring je automatisch terug naar waar je was gebleven in de code.

Hoe zet je deze ISR aan? Hiervoor moet je een **bit instellen** in een register in de AVR (zie verder).

➔ Rekenvoorbeeld van 20ms timer met schematische voorstelling:

2.5 TimerOne bibliotheek downloaden:

Indien je deze bibliotheek niet hebt in jouw IDE omgeving kan je deze altijd downloaden via

https://playground.arduino.cc/Code/Timer1/

Klik op de volgende link:

License: Creative Commons

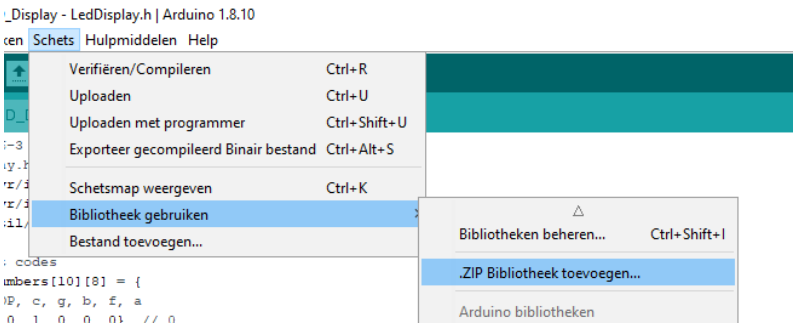
Download -> TimerOne Google Code download

To install, simply unzip and put the files in Arduino/hardware/libraries/Timer1/

Selecteer nu de laatste versie en download de zip file.

Er zijn nu 2 manieren om deze zipfile in de Arduino IDE te voorzien:

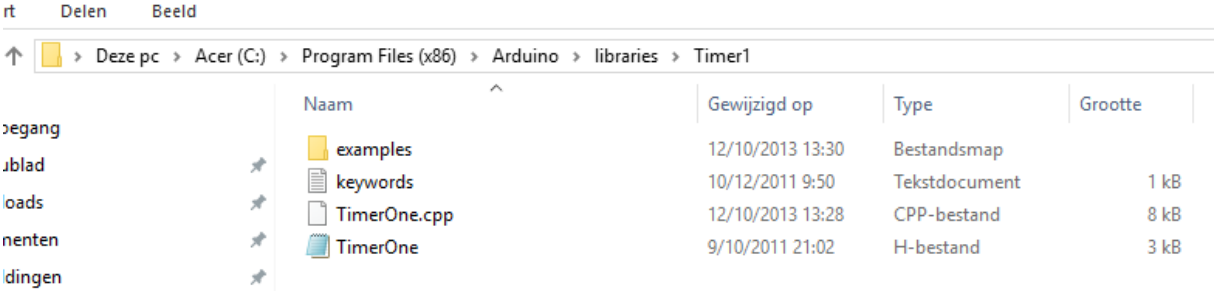
De makkelijkste manier is door via Arduino IDE -> schets -> bibliotheek gebruiken -> ZIP bib toevoegen. Verwijs naar de ZIP file en OPEN deze.



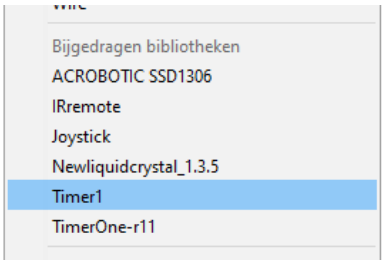
Als alles goed is gegaan moet nu de bibliotheek geïmporteerd zijn in Arduino.

Controleer dit door de TimerOne terug te zien staan in schets -> bibliotheek gebruiken -> TimerOne-rxx (xx staat voor de versie nummer).

Je kan ook handmatig de TimerOne file gaan plaatsen in de juiste map onder Arduino:



Hierna moet je de Arduino IDE omgeving sluiten en opnieuw opstarten alvorens de files zijn geïmporteerd.



Hier zie je de beide bibliotheken staan. Uiteraard is 1 van de 2 voldoende!

2.6 Wat gebeurt er in de LedDisplay.h driver file?

Test_4x7_LED_Display | LedDisplay.h

```
// Listing 6-3
// LedDisplay.h
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

// 7-segment codes
const int numbers[10][8] = {
  // e, d, DP, c, g, b, f, a
  {0, 0, 1, 0, 1, 0, 0, 0}, // 0
  {1, 1, 1, 0, 1, 0, 1, 1}, // 1
  {0, 0, 1, 1, 0, 0, 1, 0}, // 2
  {1, 0, 1, 0, 0, 0, 1, 0}, // 3
  {1, 1, 1, 0, 0, 0, 0, 1}, // 4
  {1, 0, 1, 0, 0, 1, 0, 0}, // 5
  {0, 0, 1, 0, 0, 1, 0, 0}, // 6
  {1, 1, 1, 0, 1, 0, 1, 0}, // 7
  {0, 0, 1, 0, 0, 0, 0, 0}, // 8
  {1, 0, 1, 0, 0, 0, 0, 0}  // 9
};

// store single digits
volatile int D1, D2, D3, D4;

volatile uint8_t activeDigit = 0;

#define sbi(PORT, bit) (PORT |= (1 << bit)) // set bit in PORT
#define cbi(PORT, bit) (PORT &= ~(1 << bit)) // clear bit in PORT
```

```
// Write number at indicated position to display
void digit(int value, int pin) {
  sbi(PORTC, pin); // pin high => current digit on
  PORTD = 0b11111111;

   // activate one segment after another
  for (int i = 0; i <= 7; i++) {
    if (0 == numbers[value][i]) {
      cbi(PORTD, (i)); // set segment
      _delay_ms(1);
      sbi(PORTD, (i)); // clear segment
    }
  }
}

// distribute number to the digits
void numberOutput (uint16_t number) {
  D1 = number/1000; number %= 1000;
  D2 = number/100;  number %= 100;
  D3 = number/10;   number %= 10;
  D4 = number;
}


void updateDisplay() {
  PORTC = 0b00000000; // all digits off

  if (0 == activeDigit) {
    digit (D1, PORTC0);
  }

  if (1 == activeDigit) {
    digit (D2, PORTC1);
  }

  if (2 == activeDigit) {
    digit (D3, PORTC2);
  }

  if (3 == activeDigit) {
    digit (D4, PORTC3);
  }

  activeDigit++;
  if (4 == activeDigit) {
    activeDigit = 0;
  }
}


void initLedDisplay() {
  DDRD = 0b11111111;
  DDRC = 0b00001111;
  // Interrupt every 1000 us = 1 ms
  Timer1.initialize(1000);
  Timer1.attachInterrupt(updateDisplay);
}
```

In onze LedDisplay.h file bepalen we hoe een cijfer op het display moet weergegeven worden. Hiervoor gebruiken de 2 functies **initLedDisplay() en numberOutput().**

Merk op dat er verschillende functies in elkaar genest zitten. Gevorderden werk, heel leerrijk dus 😊

In **initLedDisplay()** stelt men eerst, via poort registers, op een snelle manier de poortrichting in als INPUT of en OUTPUT. I.p.v. dus met pinMode te werken gaan we **direct de richting van de poorten instellen**. Veel minder type werk, dus efficiënter.

```
DDRD = 0b11111111;
```

We zetten alle 8 de bits op **PORTD** op **1 = OUTPUT.** Hierlangs sturen we straks via **D0 tot D7** de segmenten aan.

```
DDRC = 0b00001111;
```

We zetten de 4 hoogste bits op **0 = INPUT.** De rest als output. Dit zijn de analoge/digitale uitgangen die nu digitaal zijn ingesteld op **PORTC** (**C0 tot C3**). Daarmee zetten we een segment via de CA aan of niet.

Meer info vind je op https://www.arduino.cc/en/Reference/PortManipulation

Dan wordt de **timer1 ingesteld**. Hiervoor vallen we terug op de functies dewelke met geschreven heeft in de **Timerone.h** bibliotheek. Eerst wordt de **snelheid van de timer ingesteld op 1000 us**.

Gelukkig moeten we zelf niet op register niveau deze timer instellen. Meer info vind je op:

https://maxembedded.wordpress.com/2011/06/28/avr-timers-timer1/

```
Timer1.initialize(1000);
```

Na elke ms wordt de ISR aangeroepen en wordt de functie **updateDisplay** aangeroepen en uitgevoerd omdat de ingestelde tijd is bereikt in het timer1 register.

```
Timer1.attachInterrupt(updateDisplay);
```

Wat gebeurd er in de ISR functie updateDisplay() ?

```
void updateDisplay() {
  PORTC = 0b00000000; // all digits off

  if (0 == activeDigit) {
    digit (D1, PORTC0);
  }

  if (1 == activeDigit) {
    digit (D2, PORTC1);
  }

  if (2 == activeDigit) {
    digit (D3, PORTC2);
  }

  if (3 == activeDigit) {
    digit (D4, PORTC3);
  }

  activeDigit++;
  if (4 == activeDigit) {
    activeDigit = 0;
  }
}
```

Eerst worden de CA van de displays uitgezet via de snelle PORTC manipulatie.

PORTC = 0b0000000;

Dan wordt de variabele “activeDigit” afgevraagd en wordt **slechts 1 digit tegelijk aangezet.**

Bovenaan in de code is deze variabele op 0 gezet. **Elke digit zal 1 keer aangezet** worden indien dit van toepassing is (telkens wanneer de timer de routine oproept wisselt de digit). Dit is dus de **eerste multiplex techniek.**

Het getal dat op de digits komt te staan wordt bepaald via de numberOutput() functie. Oorspronkelijk zit er nog niets in de variabelen. Na 1 keer de code te doorlopen worden ze ingevuld.

Hoe worden de getallen per digit uitgepuzzeld?

```
// distribute number to the digits
void numberOutput (uint16_t number) {
  D1 = number/1000; number %= 1000;
  D2 = number/100;  number %= 100;
  D3 = number/10;   number %= 10;
  D4 = number;
}
```

Stel we hebben “1234” ingevuld als number in de hoofdloop, wat krijgen we dan als waardes per digit?

D1 = number / 1000;

D1 = 1234 / 1000; //nu komt het **getal voor de komma** in D1 (1234/1000 = **1**.234 -> D1 = 1)

number %= 1000;

1234 %= 1000; // nu komt **de rest waarde** van de deling in number te zitten (1234/1000 = 1.**234** -> number = 234)

D2 = 234 / 100 = 2

Number = 234 % 100 = 34

D3 = 34/ 10 = 3

Number = 34 % 10 = 4

D4 = 4

Deze code haalt dus de verschillende getallen per digit uit elkaar.

Via de functie **digit (Dx , PORTCx);** wordt nu het getal naar de juiste digit gestuurd (7-segment display).

Vb **digit (D0,PORTC0);** waarbij D0 = 1 , dan wordt de “1” op het segment, aangesloten met zijn CA op C0, aangezet. Dit is de meest linkse digit.

Hoe doet deze dat?

```
#define sbi(PORT, bit) (PORT |= (1 << bit)) // set bit in PORT
#define cbi(PORT, bit) (PORT &= ~(1 << bit)) // clear bit in PORT

// Write number at indicated position to display
void digit(int value, int pin) {
  sbi(PORTC, pin); // pin high => current digit on
  PORTD = 0b11111111;

   // activate one segment after another
  for (int i = 0; i <= 7; i++) {
    if (0 == numbers[value][i]) {
      cbi(PORTD, (i)); // set segment
      _delay_ms(1);
      sbi(PORTD, (i)); // clear segment
    }
  }
}
```

Even de volgende defines uitzoeken:

#define sbi(PORT, bit) (PORT |= (1 << bit))

Als je de naam van de poort en de bit van die poort invult ergens in de code in **sbi(PORT,bit);** dan gaat de aangegeven bit in die poort hoog gemaakt worden (set bit = sbi).

Vb: sbi(PORTD, 1); // zet D1 = 1

Stel PORTD = **0x0000 0000**

PORTD = 1 << 1 // schuif een 1 één keer naar links in PORTD

Dit geeft 0x0000 0010

PORTD = PORTD | 0x0000 0010 = **0x0000 0000** OR 0x0000 0010 = 0x0000 0010

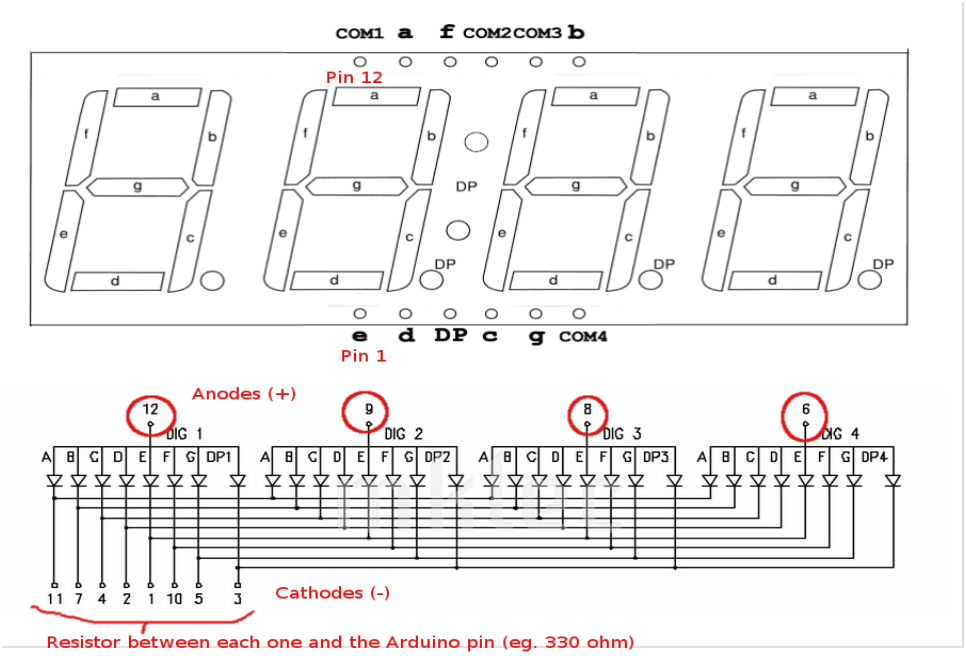
#define cbi(PORT,bit) (PORT &= (1<<bit))

Deze define zal de gewenste bit op 0 zetten in PORTx en van de rest afblijven.

Dus de bovenstaande code zal er voor zorgen dat telkens **1 segment (gestuurd via de bit) van de digit** (dewelke is geactiveerd via een PORTC pin) eerst '0' (clear) en na 1ms terug '1' (set) wordt. De 7 segmenten van de digit worden zo tegen hoge snelheid even om de beurt aangezet.

Dit is de **2de multiplexer. Door het kort om de beurt aanzetten van de segmenten moet er geen extra weerstanden voorzien worden om de stroom op te vangen.**

Merk op dat deze **segmenten laag actief** zijn (eerst clear, daarna set).



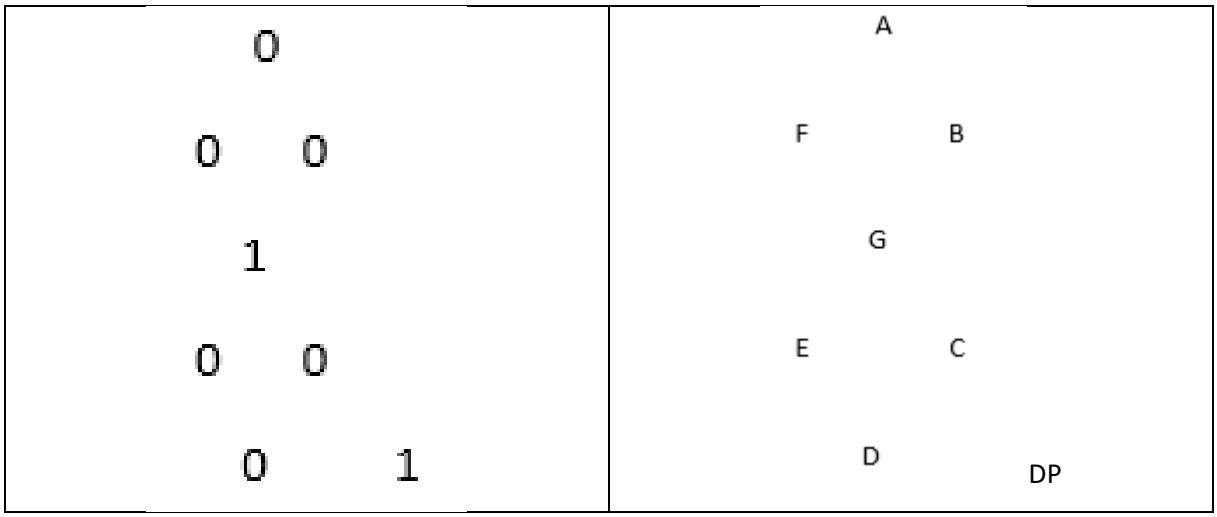
Hoe kunnen we nu **flexibel de cijfers vormen**?

We stoppen ze in een **array**. Dit is een verzameling van variabelen die in principe allemaal de zelfde naam hebben, maar de index er achter geeft de positie in de array weer.

In dit geval hebben we een 2D array vooraf opgesteld. Zowel de **x (kolom) als y (rij) coördinaat** moet je ingeven om de juiste waarde van het betreffende segment te bekomen.

Merk op dat we met **een "0"** de betreffende **segment aanzetten.**

Vb Een '0' op een laag actief 7-segment is : const int number[10] [8] = { …….}



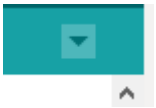

```
// 7-segment codes
const int numbers[10][8] = {
  // e, d, DP, c, g, b, f, a
  {0, 0, 1, 0, 1, 0, 0, 0}, // 0
  {1, 1, 1, 0, 1, 0, 1, 1}, // 1
  {0, 0, 1, 1, 0, 0, 1, 0}, // 2
  {1, 0, 1, 0, 0, 0, 1, 0}, // 3
  {1, 1, 1, 0, 0, 0, 0, 1}, // 4
  {1, 0, 1, 0, 0, 1, 0, 0}, // 5
  {0, 0, 1, 0, 0, 1, 0, 0}, // 6
  {1, 1, 1, 0, 1, 0, 1, 0}, // 7
  {0, 0, 1, 0, 0, 0, 0, 0}, // 8
  {1, 0, 1, 0, 0, 0, 0, 0}  // 9
};
```

➔ **Taak 1 :** maak de originele test opstelling en test de code. Zie dat je de code begrijpt.

2.3 Extra uitdagingen:

- Onderzoek nu ook de andere oefening waarbij je een **klok** maakt met behulp van de kennis van bovenstaande code en nieuwe code. De **displaydriver blijft dezelfde**. Het verschil zit hem nu in het aansturen van de klok. Mogelijk moet je eerst een **extra headerfile aanmaken** waarin je dan de **LedDisplayV3.h code** plakt.

  Een extra file in arduino toevoegen kan je via het volgende knopje rechts boven:



- In de code vind je de ISR functie **update_time()** terug. 1 keer per seconde zal de variable tc++ met 1 verhoogd worden. Dit zal gebeuren tot de waarde tc = 60 (zie rest code).

```
void update_time() {
  tc++;
}
```

- Deze ISR functie wordt elke seconde door de **Timer1 aangeroepen**.

```
Timer1.initialize(1000000); // Interrupt every 1000000 us = 1.000000 s
Timer1.attachInterrupt(update_time);
```

- Telkens wanneer de volgende voorwaarde klopt wordt het scherm geupdated:

```
while (sec == tc) {
  refresh();
}
```

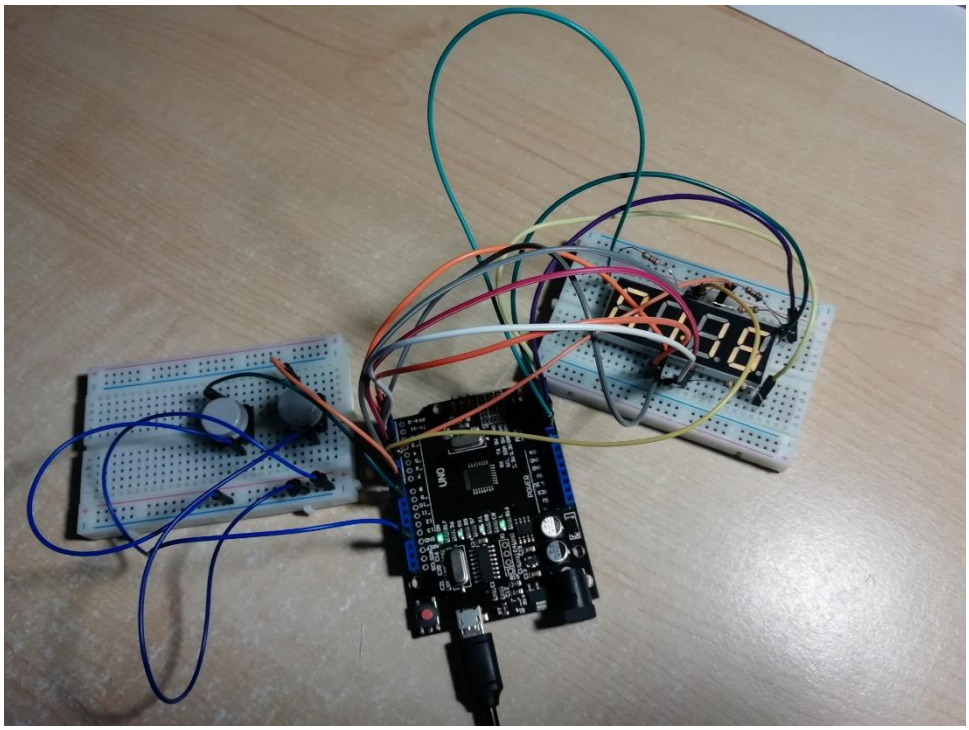
- De **functie refresh()** verwijst naar de functie in de LedDisplayV3.h file. Daar wordt zoals eerder al uitgelegd het display aangestuurd en de waardes op de digits aangepast.
- De variabele tc zet het hele klok mechanisme aan het werk **en past per 60 seconden de minuten** aan. Daarna wordt het uur na 60 minuten aangepast. Tot slot wordt alles weer gereset.
- Het gene we op het display gaan zien wordt weer bepaald door de functie numberOutput().
- Deze keer wordt het getal gevormd door de uren x 100 te vermenigvuldigen (2 plaatsen naar links op te schuiven) en dan er de minuten bij te tellen.

```
numberOutput(hr * 100 + mn);
```

➔ **Taak 2:** test de code van de klok uit en onderzoek hoe deze geschreven is.

De dubbelpunt DP wordt aangezet tussen de uren en minuten. Is laag actief bij elk getal.

```
const int numbers[10][8] = {
  {0, 0, 0, 0, 1, 0, 0, 0}, // 0
  {1, 1, 0, 0, 1, 0, 1, 1}, // 1
  {0, 0, 0, 1, 0, 0, 1, 0}, // 2
  {1, 0, 0, 0, 0, 0, 1, 0}, // 3
  {1, 1, 0, 0, 0, 0, 0, 1}, // 4
  {1, 0, 0, 0, 0, 1, 0, 0}, // 5
  {0, 0, 0, 0, 0, 1, 0, 0}, // 6
  {1, 1, 0, 0, 1, 0, 1, 0}, // 7
  {0, 0, 0, 0, 0, 0, 0, 0}, // 8
  {1, 0, 0, 0, 0, 0, 0, 0}  // 9
};
```



➔ **Taak 3:**

- Voorzie in de ISR functie een **vrije uitgangspin** die even omhoog en dan weer omlaag gaat.
- Neem de **logic analyser** en meet op deze pin het ritme van de timer1. Kijk in hoeverre dit overeenkomt met de ingestelde waarde.
- Maak een **PWM signaal** dat je uitstuurt naar een LED. Gebruik eerst een **delay van 20ms**. Daarna maak je ook een vertraging met een **timer van 20ms**. Meet beide PWM signalen op de digitale scope / logic analyzer. Welke is de nauwkeurigste?

3 Spelen met geheugens

In deze workshop gaan we het interne en externe geheugen leren aanspreken via een UNO.

We willen graag metingen doen van temperatuur, fijnstof, licht of dergelijke. Maar hoeveel metingen kunnen we hiervan opslaan, en vooral hoelang kunnen we dit doen?

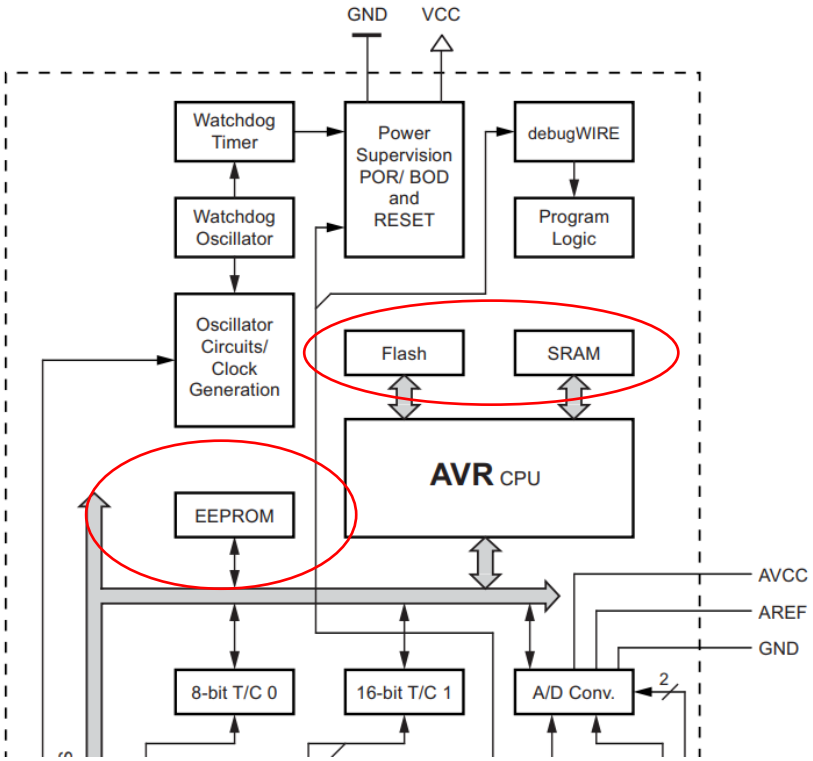
Stel dat je in een klaslokaal 48 keer per dag (om het half uur) de temperatuur wil meten en je wil dit uitzetten op een grafiek na een week. Hopelijk valt in de tussentijd de stroom niet uit of is de batterij voldoende opgeladen voor een ganse week te meten.

Stel dat je de gemeten waardes opslaat in het werkgeheugen (RAM type, vluchtig) dan zou je alles kwijt geraken als de spanning van de UNO zou wegvallen.

Gelukkig hebben ze daarom EEPROM geheugen uitgevonden. Dit is niet-vluchtig geheugen. Het zit wel in mindere mate in de UNO, maar we kunnen altijd uitbreiden via I2C.

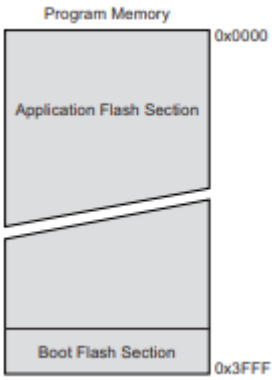
- 32K bytes of in-system self-programmable flash program memory
- 1Kbytes EEPROM
- 2Kbytes internal SRAM
- Write/erase cycles: 10,000 flash/100,000 EEPROM

De MEGA328P vermeld deze waardes in de datasheet



In het blokschema van de MEGA328P vinden we de 3 soorten geheugens terug.

Program Memory Map ATmega328P

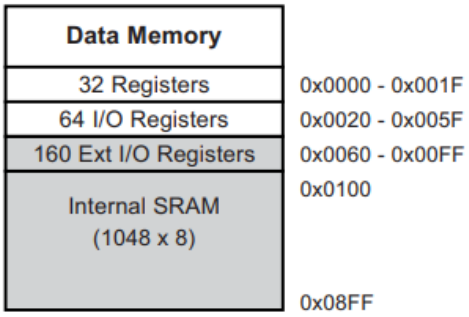


Het program memory (flash) bevat 0x3FFF = 16K adressen

Elk adres bevat 16 bits. Omdat elke instructie in de AVR architecture van de chip 16 bits of 32 bits breed is, hebben ze elke geheugenplek van het flash geheugen dus ook 16 bits breed gemaakt.

Merk op dat er voor de bootloader ook enkele geheugenplaatsen zijn voorzien (zie hiervoor het experiment waarbij we de bootloader via de ISP bus hebben gedownload in jouw UNO kit).

Data Memory Map



Het interne SRAM geheugen bevat 2048 bytes (0x08FF)

EEPM1	EEPM0	Programming Time	Operation
0	0	3.4ms	Erase and write in one operation (atomic operation)
0	1	1.8ms	Erase only
1	0	1.8ms	Write only
1	1	–	Reserved for future use

De snelheid van de 1K bytes aan EEPROM geheugen is hier aangegeven.

3.1 Intern geheugen van de UNO leren gebruiken

In eerste instantie gaan we gebruik maken van de 1024 bytes intern EEPROM voor deze UNO oefening. Deze toepassing van geheugen zou je bijvoorbeeld kunnen gebruiken om **calibraties op te slaan bij een drone.**

Als we willen gaan werken moeten we de library #include <EEPROM.h> toevoegen aan onze code.

Welke commando's kunnen we inzetten voor het interne geheugen?

- EEPROM Clear: Clear the bytes in the EEPROM.
- EEPROM Read: Read the EEPROM and send its values to the computer.
- EEPROM Write: Stores values from an analog input to the EEPROM.
- EEPROM Crc: Calculates the CRC of EEPROM contents as if it was an array.
- EEPROM Get: Get values from EEPROM and prints as float on serial.
- EEPROM Iteration: Understand how to go through the EEPROM memory locations.
- EEPROM Put: Put values in EEPROM using variable semantics.
- EEPROM Update: Stores values read from A0 into EEPROM, writing the value only if different, to increase EEPROM life.

Uitgebreide info over deze commando's vind je op https://www.arduino.cc/en/Reference/EEPROM

➔ **Test:** Probeer de volgende EEPROM schrijf/lees oefening eens uit:

eeprom_write_read

```
#include <EEPROM.h>
#define led 13

int addr = 0;
int value = 0;

void setup() {
    pinMode(led,OUTPUT);
  Serial.begin(9600);
}

void loop() {

  int val = analogRead(0) / 4;

  digitalWrite(led,HIGH);
  Serial.print("write: ");
  Serial.println(val);
  EEPROM.write(addr, val);


  value = EEPROM.read(addr);

  Serial.print("read: ");
  Serial.print(addr);
  Serial.print("\t");//tab
  Serial.print(value, DEC);
  Serial.println();

  // Arduino Uno:        1kb EEPROM storage.

  addr = addr + 1;

  if (addr == EEPROM.length()) //lees grote van eeprom en vergelijk
  {
    addr = 0;
  }

  delay(100);
  digitalWrite(led,LOW);
  delay(100);
}
```

Begrijp jij wat er gebeurd?

```
read: 499       81
write: 86
read: 500       86
write: 92
read: 501       92
write: 86
read: 502       86
write: 82
read: 503       82
write: 81
read: 504       81
write: 86
read: 505       86
```

Schrijf waarde van A0 in eeprom geheugen.

Lees daarna waarde uit op zelfde plaats uit geheugen.

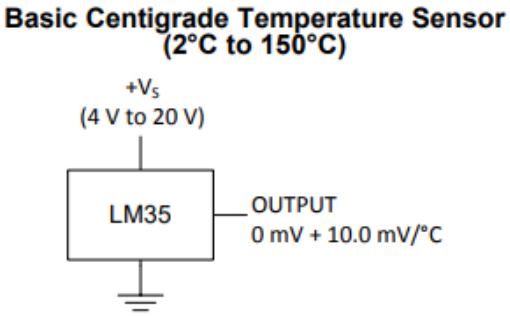
Toon ook de adresplaats in het geheugen.

3.1.1 LM35 temperatuur waardes opslaan in de interne EEPROM

We gaan het interne geheugen inzetten om LM35 waarde in op te slaan.

Merk op dat er via ALI express ook FAKE EEPROM's verkocht worden! Deze geven willekeurige waardes terug. Dit zijn dus geen goede LM35's.

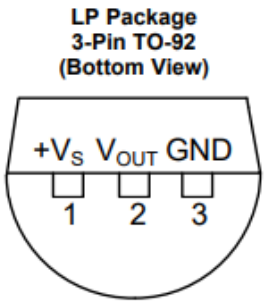
Wat is een LM35?

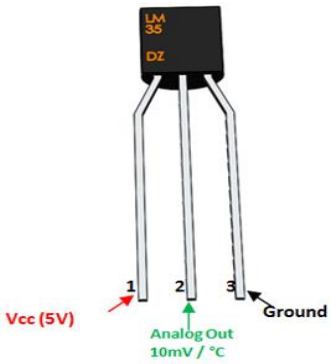


Dit is een **analoge sensor** waarbij de temperatuur mag variëren van 2 tot 150 graden.

De uitgang gaat met **10mV stijgen bij elke graad** dat **erbij** komt.

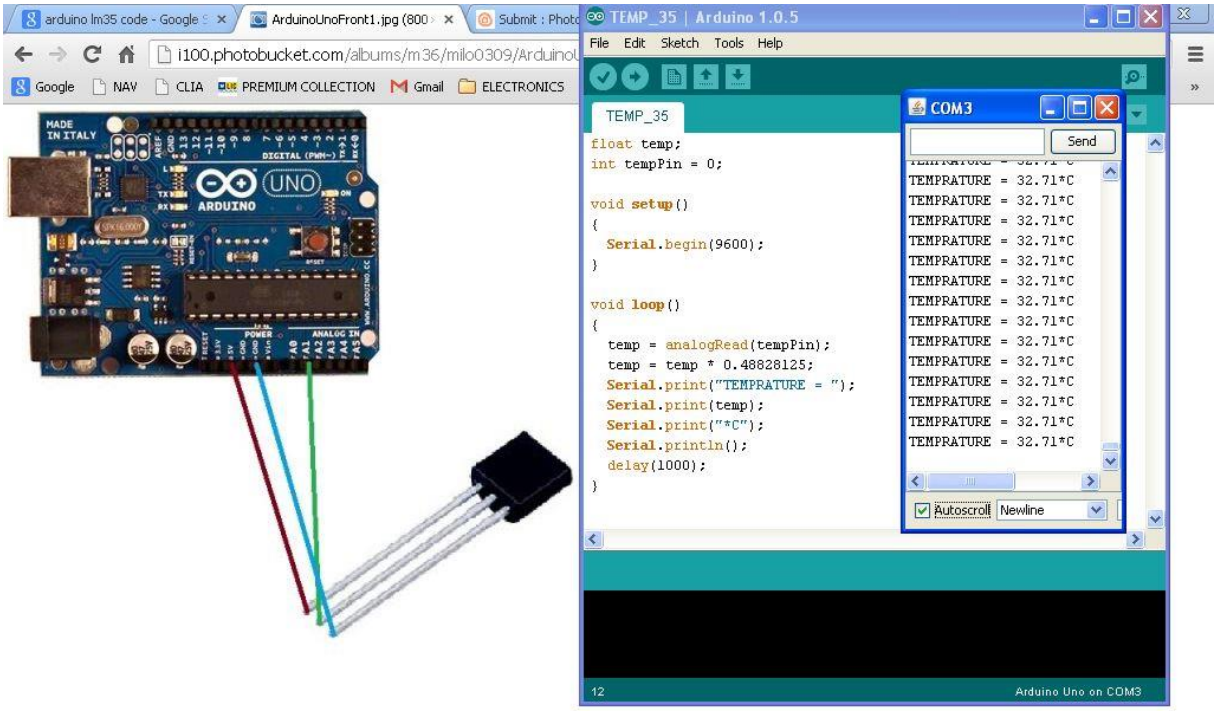
Het aansluiten is heel eenvoudig. 5V op de VS (voeding) is al voldoende om deze te laten werken.





BOTTOM view geeft de volgende aansluitingen van de LM35 weer.

Bouw de volgende UNO schakeling:



+5V gaat naar pin 1

De analoge uitgang A1 gaat naar pin 2

De massa gaat naar pin 3

Geef nu de volgende code in om de LM35 uit te lezen en op de serial monitor te tonen:

LM35_op_serial_monitor

```
int val;
int tempPin = 1;//A1

void setup()
{
Serial.begin(9600);
}
void loop()
{
val = analogRead(tempPin);
Serial.print("analoge waarde A1 = ");
Serial.println(val);
float mv = ( val/1024.0)*5000;
float cel = mv/10;
float farh = (cel*9)/5 + 32;

Serial.print("TEMPERATUUR = ");
Serial.print(cel);
Serial.print("*C");
Serial.println();
delay(1000);


/* uncomment this to get temperature in farenhite
Serial.print("TEMPERATUUR = ");
Serial.print(farh);
Serial.print("*F");
Serial.println();


*/
}
```

De **resolutie** (kleinste veranderlijke te meten waarde) van de analoge meting is 5V / 1024 = 4.88mV

1024 bits (2^10 mogelijkheden) in de ADC (analoge digitale converter)

5 V = 5000mV (max te meten waarde)

$$\text{Gemeten spanning (in mV)} = \frac{gemeten\ spanningswaarde\ van\ A0}{\max combinaties} \text{ x max te meten waarde}$$

Stel dat je 1023 meet:

$$\text{Gemeten spanning (in mV)} = \frac{1023}{1023} \text{ x } 5000 = 5000\text{mV} = 5\text{V}$$

Omzettingen mV spanning van LM35 naar celsius :

Er komt 10 mV per graad celsius bij, dus willen we de graden weten moeten we de gemeten waarde delen door 10.

Vb: 240 mV / 10 = 24 graden celsius

Andere berekening LM35:

We weten dat de ADC 1024 stappen heeft van 5mV.

Bij 20 graden gemeten is de meetspanning op A0 20 graden x 10mV/graden celsius = 200mV.

De ADC geeft dus als resultaat 200mV / 5mV = 40.

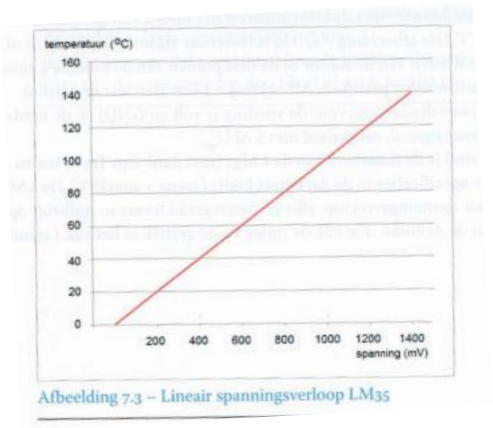
Als je nu 40 / 2 = 20 graden doet heb je de waarde in graden celsius.

Dan ziet de code er zo uit:

LM35_op_serial_monitor_alternatief §

```
int val;
int tempPin = 1;//A1

void setup()
{
Serial.begin(9600);
}
void loop()
{
val = analogRead(tempPin);
Serial.print("analoge waarde A1 = ");
Serial.println(val);
Serial.print("TEMPERATUUR = ");
Serial.print(val / 2);
Serial.print("*C");
Serial.println();
delay(1000);

}
```

```
analoge waarde A0 = 42
TEMPERATUUR = 20.51*C
analoge waarde A0 = 43
TEMPERATUUR = 21.00*C
analoge waarde A0 = 43
TEMPERATUUR = 21.00*C
analoge waarde A0 = 43
```



Afbeelding 7.3 – Lineair spanningsverloop LM35

Lineair verloop LM35: spanning = 0.01 x temp

Nu gaan we de gemeten LM35 waarde opslaan in het interne geheugen van de UNO.

Test volgende code:

eeprom_intern_met_LM35

```
#include <EEPROM.h>
#define led 13
#define LM35 1 //A1
int addr = 0;
int value = 0;
int val = 0;

void setup() {
    pinMode(led,OUTPUT);
    Serial.begin(9600);
}

void loop() {

  for(int teller = 0;teller < 10; teller++){


    val = analogRead(LM35);

    val = val / 2; //zet om naar celsius waarde
```

```
      digitalWrite(led,HIGH);
      Serial.print("write temp: ");
      Serial.println(val);
      EEPROM.write(addr, val);

      value = EEPROM.read(addr);

      Serial.print("read temp: ");
      Serial.print(addr);
      Serial.print("\t");//tab
      Serial.print(value, DEC);
      Serial.println();

      addr = addr + 1;

      delay(100);
      digitalWrite(led,LOW);
      delay(100);
    }

   addr = 0;//reset adres
}
```

➔ **Verslag maken:** sla 20 temperatuurwaardes op, trek daarna de stekker uit van de UNO. Lees nu de 20 waardes terug uit en toon ze op de serial monitor. Zijn dit nog steeds de zelfde waardes als je had opgeslagen voordat je de stekker had uitgetrokken van de spanning. Bewijs dit!

3.2 Extern geheugen via I2C leren gebruiken

Nu we weten hoe we de interne EEPROM kunnen gebruiken om niet vluchtige data op te slaan, gaan we nog een stapje verder. De **interne EEPROM van de UNO kan slecht 1Kbyte** aan data op slaan.

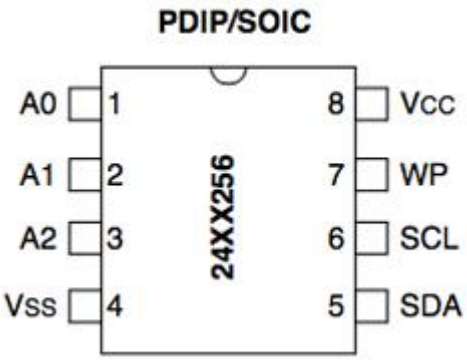
Stel dat je meer data moet bewaren?

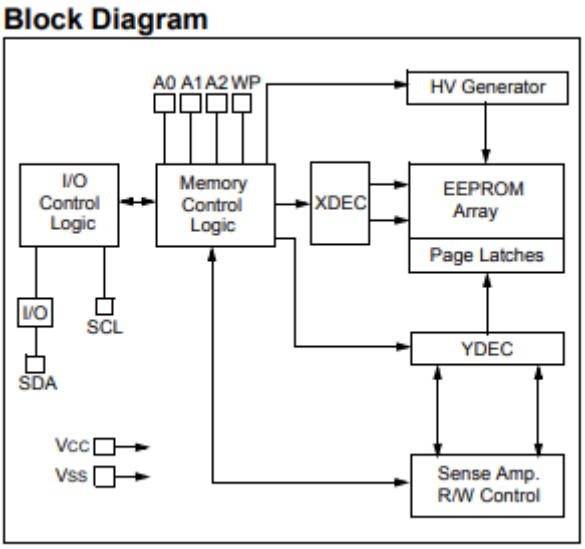
Dan kunnen we bijvoorbeeld gebruik maken van een **externe EEPROM, de 24LC256**. In dit geval gaan we zo weinig mogelijk draden gebruiken. Dus we kiezen voor de **I2C methode**. Deze is reeds aanbod gekomen in de gevorderden deel 1 cursus.

Hier toch even wat korte uitleg over de EEPROM en de I2C aansluitingen.

De EEPROM 24LC256 bevat **32kByte** geheugen ruimte. De “256” komt van 262144 bits = 256kbit.

262144 / 8 bits = 32 kbyte.





We gaan aan de slag met een 8 polige DIL versie.

Welke aansluitingen zitten er op de EEPROM?

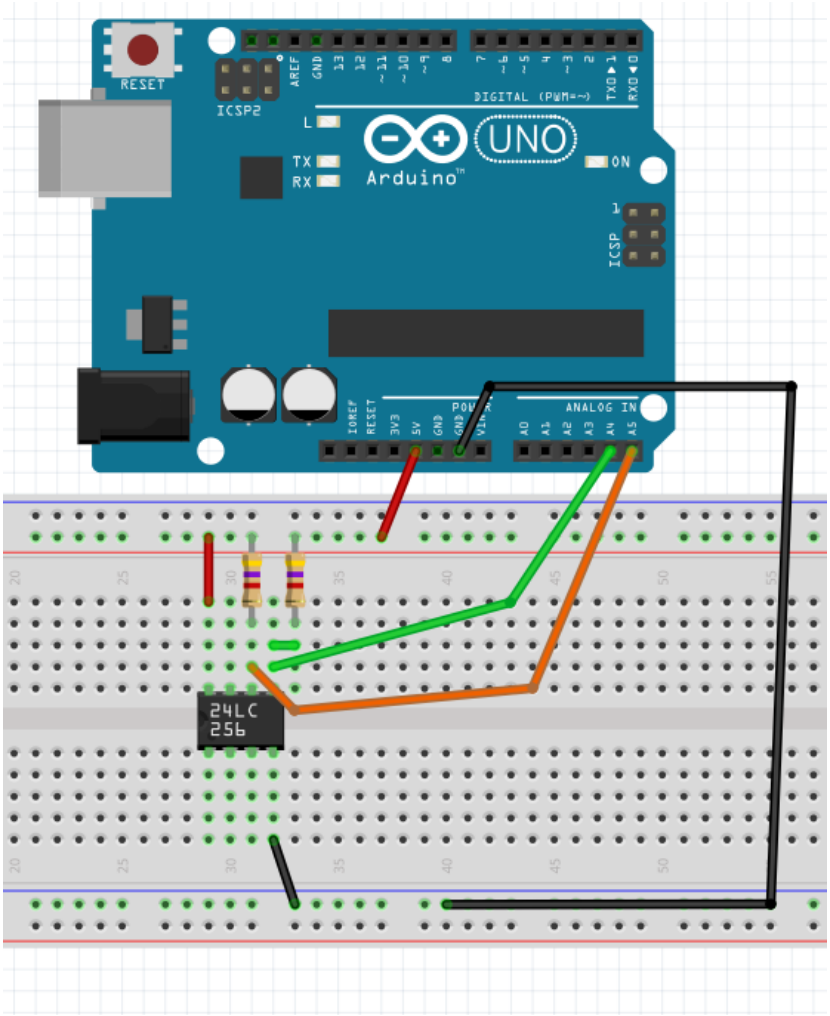
A0 (I2C adres)	1
A1	2
A2	3
VSS (massa = GND)	4
SDA (data lijn I2C)	5
SCL (klok lijn I2C)	6
WP (write protect)	7
VCC (+5V)	8

WP = 1 wil zeggen dat je de inhoud van de EEPROM niet kan overschrijven.

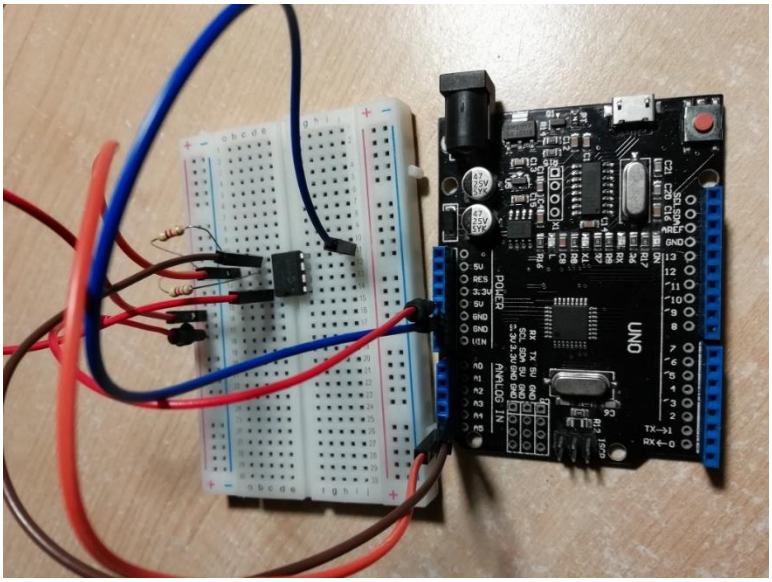
Merk op dat de adreslijnen A0A1A2 en de WP een interne pull-down weerstand hebben. Ze zijn dus standaard al aan massa aangesloten. Zo hangt er adres 000 op de chip en is hij altijd overschrijfbaar, wat nodig is in ons geval.

Op de SDA en SCL lijn moeten we een 4K7 pull-up weerstand voorzien die aan de +5V hangt. Dit hoort bij het I2C protocol. SDA is pin A4 en SCL is pin A5 op de UNO.

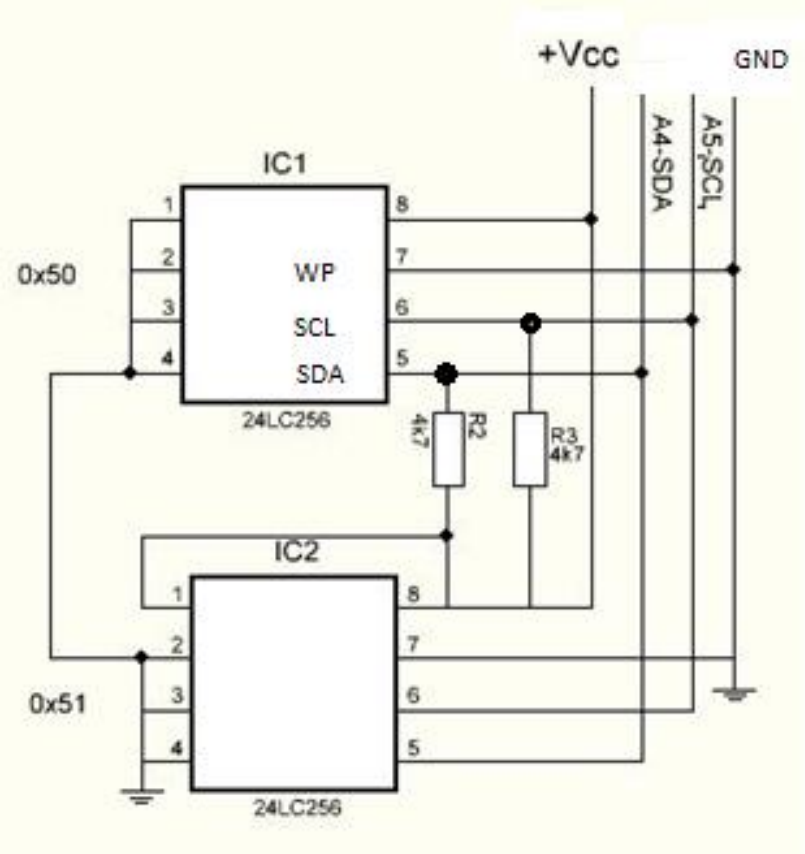
Sluit de 24LC256 aan op de UNO:



Vergeet niet van de 4k7 aan te sluiten tussen de SDA en SCL naar de +5V.



Indien je meerdere EEPROMS wil aansluiten kan dit als volgt:



Merk op dat de WP en adres lijnen die ‘0’ moeten zijn in feite niet aangesloten moeten zijn op de massa. Dat gebeurd al door de interne pull-down weerstand.

Hoe werkt het I2C protocol alweer?

We moeten eerste een start conditie voorzien (SDA al omlaag terwijl SCL nog hoog is).

Daarna wordt er groepen van 8 bits aan data verstuurd.

Na elke 8 bits volgt een ACK (Acknowledge). Hierbij gaat de slave de SDA lijn even omlaag trekken tijdens klokpuls 9.

Tot slot stoppen we met een stop conditie (SDA pas omhoog als SCL al omhoog is gegaan)

We starten altijd met de controle byte door te sturen.

FIGURE 4-1: DATA TRANSFER SEQUENCE ON THE SERIAL BUS

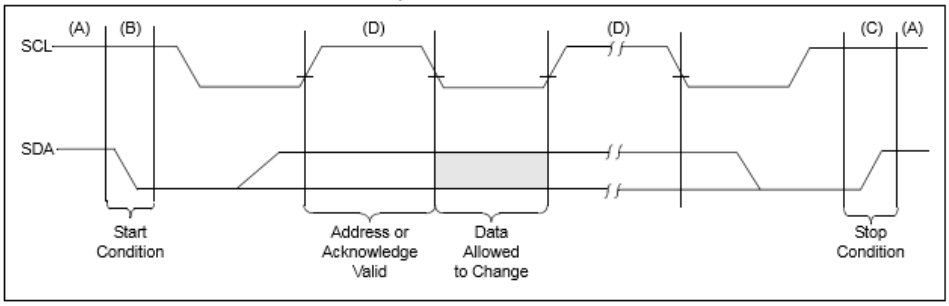
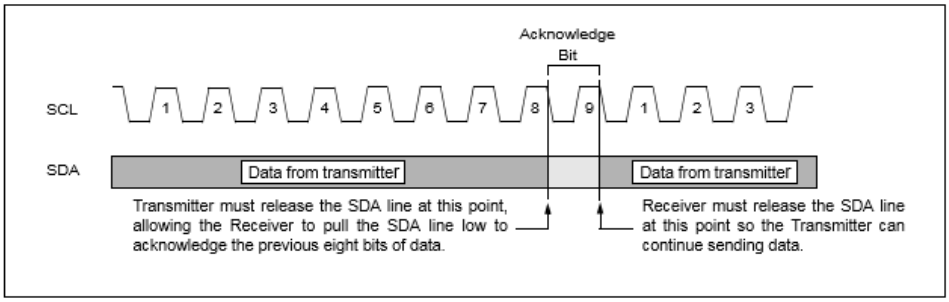


FIGURE 4-2: ACKNOWLEDGE TIMING



Merk op dat er bij het schrijven van data via de I2C lijnen dan het volgende ontstaat:

FIGURE 6-1: BYTE WRITE

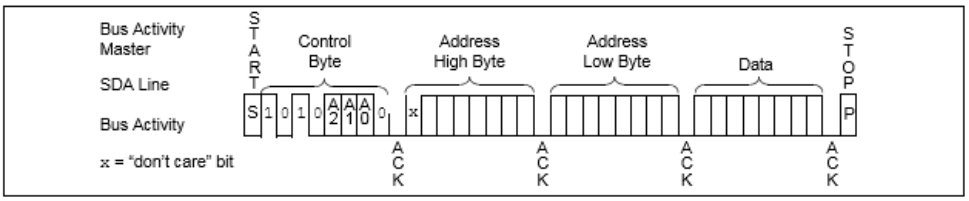
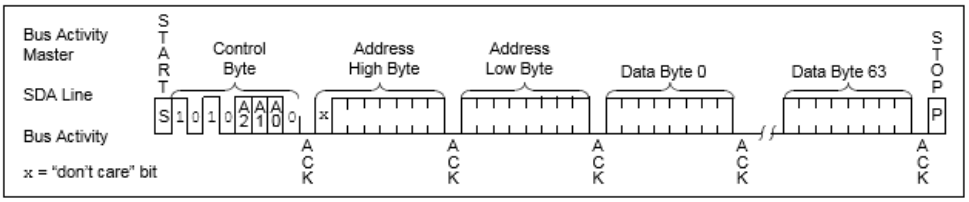


FIGURE 6-2: PAGE WRITE



De R/W bit staat op "0" om te kunnen schrijven.

Je moet dus maar 1 keer het adres door te geven. Daarna worden de adresplaatsen automatisch met 1 verhoogd, telkens er een nieuwe byte wordt doorgestuurd. Steeds wordt er een ACK verstuurd na de byte.

Merk op dat bij het lezen van data eerst een write actie is van de controle byte, en daarna pas een read actie van de data byte. Na de laatste ingelezen byte wordt een NO ACK verstuurd van de master (de microcontroller). Hierbij blijft de lijn 1 kloktijd hoog.

FIGURE 8-1: CURRENT ADDRESS READ

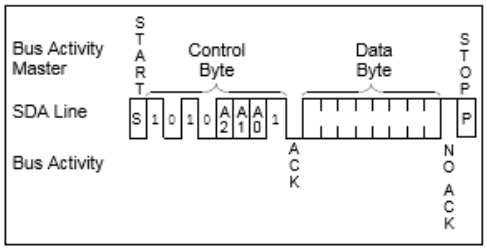
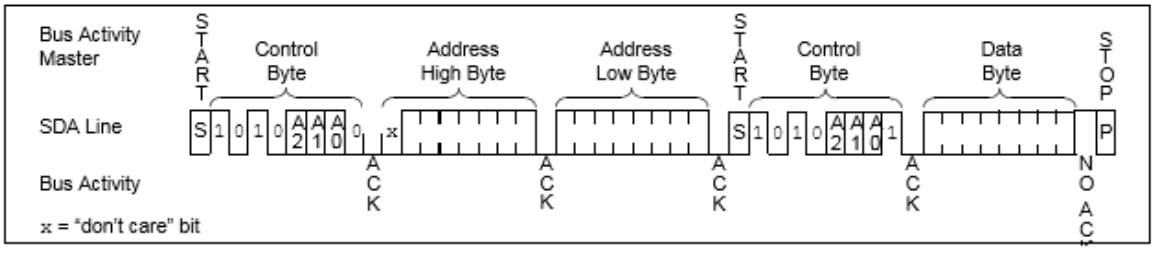
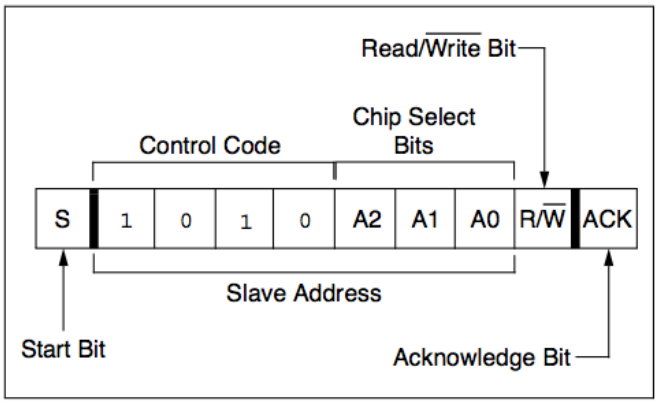


FIGURE 8-2: RANDOM READ



De R/W bit staat eerst op "0" om de controle byte en adresstart te sturen (schrijven), daarna op "1" om de data in te lezen.

Hoe zit de controle byte in elkaar?



De eerste 4 bits zijn de unieke ID van de EEPROM. In dit geval is dit “1010”. Deze nummer is gegeven door de fabrikant aan deze specifieke chip. Zo kan je ook andere I2C chips aansturen op dezelfde draden. Elke I2C chip luistert naar zijn ID.

Daarna volgt het adres van 3 bits. Hiermee zijn we in staat om 2^3 = 8 chips van hetzelfde type aan te sluiten op een I2C bus. Zo kan je dus 8 EEPROMS aansluiten op jouw UNO van dit type.

Tenslotte wordt er aangegeven of we gaan lezen (READ = 1) of schrijven (WRITE = 0) van de master naar de slave. Deze bit kan je mooi terugvinden in de data voorbeelden op de vorige figuren.

Na de controle byte volgt een ACK.

➔ **Test nu de volgende code uit:**

i2c_eeprom_oef

```
#include <Wire.h>
#define testpin 2

#define chip 0x50    //ID van 24LC256 eeprom chip
         //101 0000 = chip adres 101  = 5 + 4de bit ID + A2A1A0 = 0 000 (R/W niet meegestuurd)

void setup(void)
{
  Serial.begin(9600);
  Wire.begin();
  pinMode(testpin,OUTPUT);

  unsigned int address = 256;// MSB + LSB = 0000 0000 0000 0000 = 0
                             //             0000 0001 0000 0000 = 256
  digitalWrite(testpin,HIGH);
  writeEEPROM(chip, address, 123);//stuur 123 (max 255 = 1 byte) naar de I2C chip op plaats 256
  digitalWrite(testpin,LOW);
  Serial.print(readEEPROM(chip, address), DEC);

}
```

De **Wire.h bibliotheek** is normaal al in Arduino voorzien voor de I2C commando’s te kunnen uitvoeren. Zoniet kan je deze installeren als zip file (zie op de USB stick).

De testpin dient om met de logic analyzer het juiste moment van triggeren uit te kiezen. Deze pin 2 wordt even omhoog en omlaag gedaan op het moment dat de EEPROM data wordt geschreven via de I2C bus naar de 24LC256.

➔ Hoe het I2C adres ontdekken als je het niet weet?

Indien deze niet werken kan je nog de **I2C adres scanner** gebruiken:

http://playground.arduino.cc/Main/I2cScanner

Download deze sketch in jouw Arduino, sluit de I2C pinnen aan en kijk via de **serial monitor** op **welk adres** het device is aangesloten.

```
I2C Scanner
Scanning...
I2C device found at address 0x50  !
done
```

Dit is het resultaat van de ID en adresaansluitingen op de EEPROM.

0x50 = 101 0000

Merk op dat de bits anders gesorteerd zitten. We moeten hier niet rekening houden met de R/W bit (die wordt bepaald door de read/writeEEPROM functie zelf). Dus schuiven de bits 1 plek op en is de 7de bit altijd '0'.

Zo is 101 = 5 en 0 + 000 (laagste ID bit + adresbits) = 0 omdat A2A1A0 = 000.

Als adres kiezen we voor de locatie '256' in de EEPROM. Hier schrijven we en lezen we in deze oefening onze data = 123.

```
void loop(){} //hier gebeurt niets, enkel 1 keer setup uitvoeren

void writeEEPROM(int deviceaddress, unsigned int eeaddress, byte data )
{
  Wire.beginTransmission(deviceaddress);
  Wire.write((int)(eeaddress >> 8));   // MSB   0000 0001 0000 0000
                                //-> schuif 8 keer rechts   0000 0001
  Wire.write((int)(eeaddress & 0xFF)); // LSB   0000 0000 1111 1111
                                //             &  0000 0001 0000 0000
                                //                0000 0000 0000 0000
  Wire.write(data);
  Wire.endTransmission();

  delay(5);
}
```

Bij het schrijven van het adres moeten we eerst de MSB (hoogste deel van het adres) en daarna de LSB doorsturen. Hiervoor doen we de nodige verschuivingen (>>8) en bitmaskering 0xFF.

Je kan in de code mooi de verschillen I2C stappen terugvinden. Indien je meer data wil schrijven kan dit makkelijk, want het adres wordt automatisch verhoogd in de EEPROM met 1 na het schrijven.

```
byte readEEPROM(int deviceaddress, unsigned int eeaddress )
{
  byte rdata = 0xFF;

  Wire.beginTransmission(deviceaddress);
  Wire.write((int)(eeaddress >> 8));   // MSB
  Wire.write((int)(eeaddress & 0xFF)); // LSB
  Wire.endTransmission();

  Wire.requestFrom(deviceaddress,1);//schakel I2C bus om van schrijven naar lezen
                                    //herstart de bus

  if (Wire.available())
  {
      rdata = Wire.read();
  }
  return rdata;
}
```

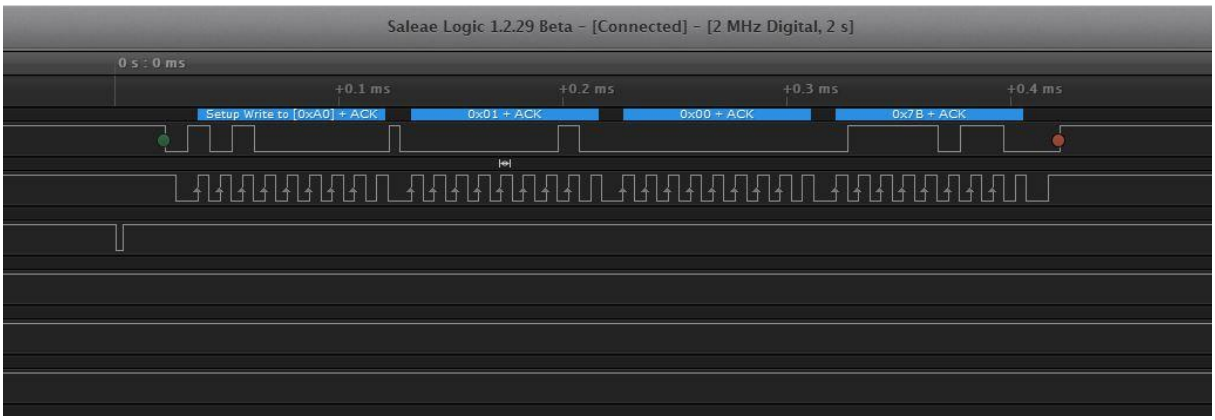
Bij het lezen moeten we eerst de controle byte schrijven en het adres in de EEPROM klaar zetten. Daarna wisselen we de data richting om en lezen we de data binnen.

De ‘1’ bij de functie Wire.requestFrom(deviceaddress,1) staat voor het aantal bytes dat je wil uitlezen uit de EEPROM.

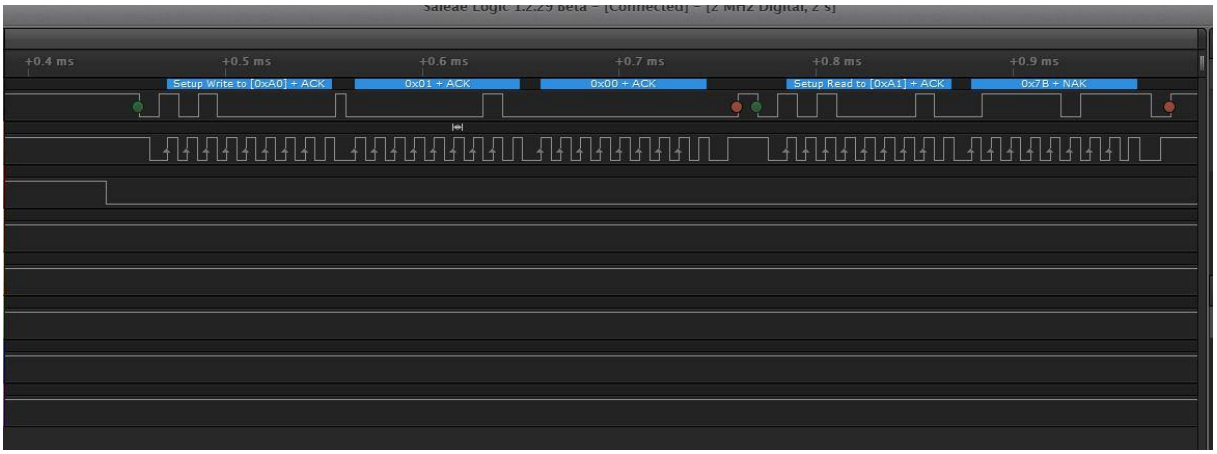
Via de “return rdata” krijg je de ingelezen data terug om daarna te printen op het serial monitor.

➔ **Verslag :**

- Probeer nu 10 bytes te schrijven en te lezen uit de externe EEPROM. Zorg hierbij dat je de code zo kort mogelijk houd (herhalingslussen gebruiken)!
- Probeer nu de voorgaande oefening te maken met arrays.
- Meet met een LA de data die verstuurd wordt naar de I2C EEPROM en kijk of je alle stappen van het protocol kan terugvinden.



Schrijf data 0x7B= 123 naar EEPROM (controle byte = 0xA0, op adres 256 = 0x0100)



Lees data uit EEPROM. Merk op dat er een herstart gebeurd van de I2C bus en schrijven verandert in lezen.

3.3 De DS18B20 waardes van Dallas in intern/extern geheugen stoppen (bij veel tijd)

Er bestaan verschillende temperatuur sensors: een gewone NTC, een LM35, een DHT11, een PT100 (in combinatie met een verschilversterker).

De meeste produceren een analoge uitgang tussen 0 en 5V dewelke we dan aan de ADC van de microcontroller moeten aansluiten. Enkel de DHT11 was een buitenbeentje. Dit was een 1-wire sensor.

Om de lijst compleet te maken voegen we er nog de DS18B20 van DALLAS aan toe.

Meer info hierover kan je o.a. vinden op volgende plek:

http://domoticx.com/arduino-temperatuur-sensor-ds18b20/

Ook deze sensor is een **1-wire bus** component.

Dit wil zeggen dat er slechts 1 draad gebruikt wordt om de data langs te verzenden van de DALLAS chip naar de UNO en omgekeerd.

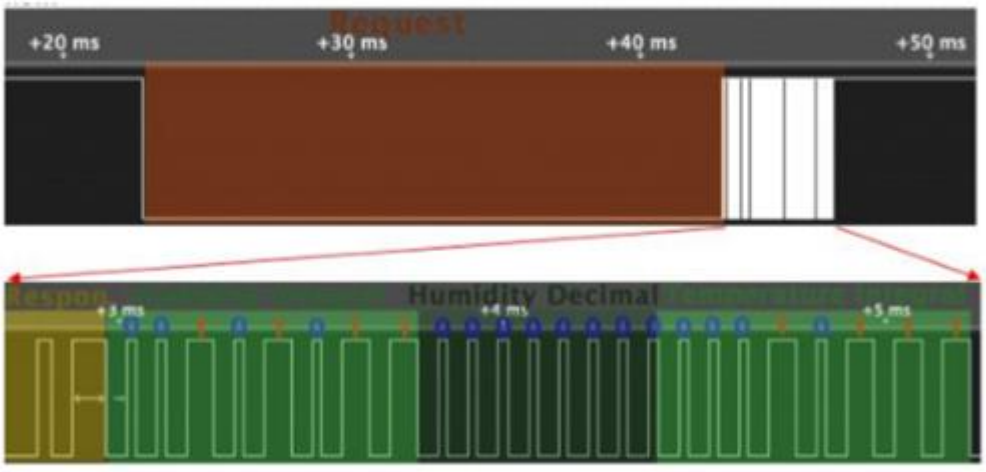
In de Arduino gevorderden deel 1 zijn we reeds uitgebreid ingegaan op dit protocol:

De sensor heeft 3 draden.

Buiten de +5V en de GND is er nog 1 data draad. Hier langs kan er in beide richtingen gecommuniceerd worden volgens het **1-wire** principe.

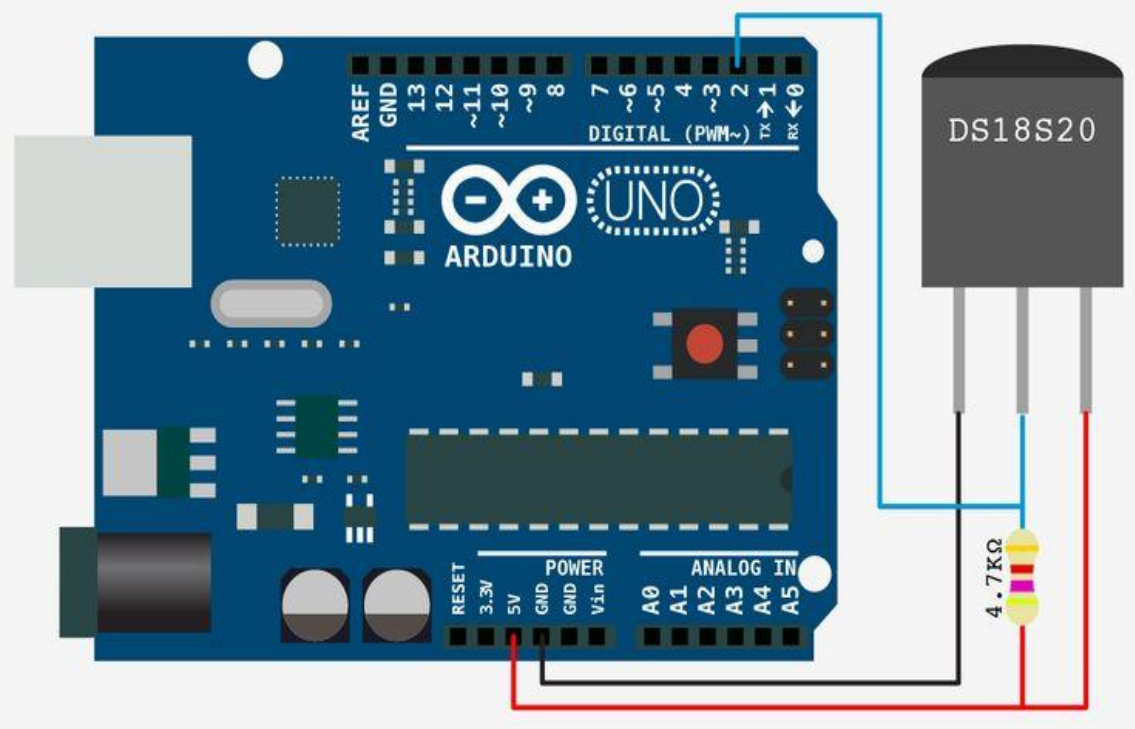
We moeten een **pullup weerstand** voorzien op de signaal uitgang naar +5V van **4K7** tot 10k. Dit is omdat we weer werken met een **open collector** uitgang. Wanneer er niets gestuurd wordt is de bus normaal gezien hoog. Als de zender of ontvanger wil praten trekt hij de lijn omlaag (lijkt op I2C, maar in dit geval is er geen aparte klok voorzien).

Men gaat altijd in **3 stappen** te werk om data te versturen (dit is gelukkig al door een Arduino library voorzien): request, response, data reading.



Je kan hier een logic analyzer meting vinden van de databus. In stapjes worden alle bits over de draad gestuurd.

Sluit de DS18B20 aan als volgt op de UNO:



De 4K7 vormt een pullup weerstand voor het data signaal op D2.

DS18B20 pin:	Arduino pin:
1 (GND)	GND
2 (Digitale uitgang)	D2
3 (Vdd (+3.3v))	+3.3v

Je mag +3V tot +5V aansluiten op de VDD pin.

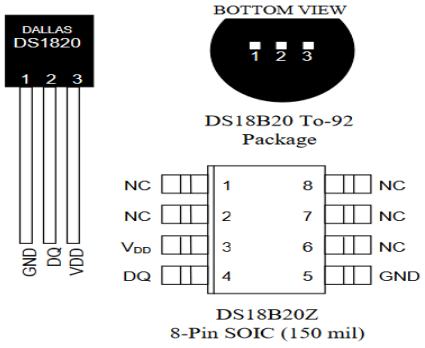
DALLAS SEMICONDUCTOR

DS18B20
Programmable Resolution 1-Wire® Digital Thermometer

www.dalsemi.com

FEATURES
- Unique 1-Wire interface requires only one port pin for communication
- Multidrop capability simplifies distributed temperature sensing applications
- Requires no external components
- Can be powered from data line. Power supply range is 3.0V to 5.5V
- Zero standby power required
- Measures temperatures from -55°C to +125°C. Fahrenheit equivalent is -67°F to +257°F
- ±0.5°C accuracy from -10°C to +85°C
- Thermometer resolution is programmable from 9 to 12 bits
- Converts 12-bit temperature to digital word in 750 ms (max.)
- User-definable, nonvolatile temperature alarm settings
- Alarm search command identifies and addresses devices whose temperature is outside of programmed limits (temperature alarm condition)
- Applications include thermostatic controls, industrial systems, consumer products, thermometers, or any thermally sensitive

PIN ASSIGNMENT



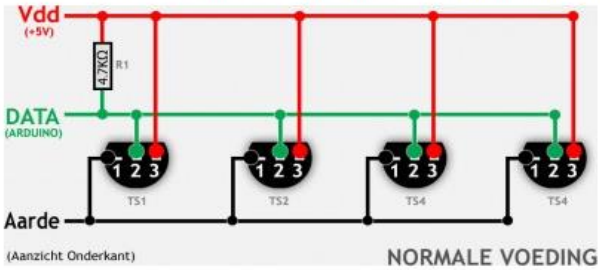
PIN DESCRIPTION

GND - Ground
DQ - Data In/Out
V_{DD} - Power Supply Voltage
NC - No Connect

Meerdere sensoren aansluiten

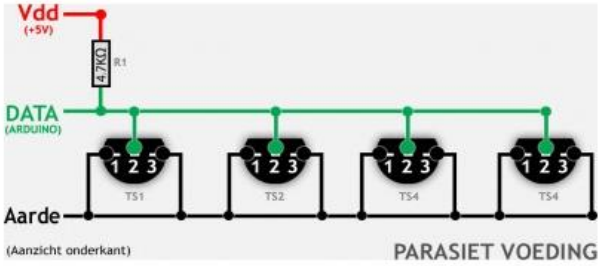
Normale Voeding

Als je de normale voeding gebruikt, dan is de onderstaande schakeling wat je nodig hebt om meerdere sensoren aan te sluiten.



Parasiet Voeding

In deze methode sluiten we in principe pin 1 en 3 van de sensor kort en trekt de sensor stroom via de data pin.



Meetresolutie instellen

Het is mogelijk om de meetresolutie van de sensor te cconfigureren dat kan met het inladen van de dallas bibliotheek:

Voor 12-Bit: `sensors.setResolution(temp1, 12);`

Maar des te preciezer, hoe langer het duurt voordat data opgevraagd wordt:

Mode	Resol	Conversion time
9 bits	0.5°C	93.75 ms
10 bits	0.25°C	187.5 ms
11 bits	0.125°C	375 ms
12 bits	0.0625°C	750 ms

Om de data uit de sensor te halen hebben we dus verschillende modes: hoe nauwkeuriger we de meting willen laten zijn, hoe meer bits we moeten uitlezen. Dit kost uiteraard meer tijd.

Verder volgt er Arduino code waarmee we de resolutie kunnen instellen. Je gebruikt hier de speciale **DALLAS bibliotheek** voor.

Meer info hierover vind je op de site http://domoticx.com/arduino-temperatuur-sensor-ds18b20/

Bouw het arduino circuit van de DS18B20 (datapin op D2) en doe eerst een **"1-wire scanner" test** om het adres van de chip te bekomen. Deze vind je terug bij de oefeningen op de USB stick.

Voordat we de code kunnen compileren moet eerst de **OneWire.h bibliotheek** geïnstalleerd zijn. Deze kan je terugvinden op de USB stick.

Wanneer je deze scanner laat lopen krijg je het volgende resultaat:

```
Looking for 1-Wire devices...

Found '1-Wire' device with address:
0x28, 0xD4, 0x62, 0x16, 0xA8, 0x01, 0x3C, 0xAF

That's it.
```

➔ **Test nu** de volgende code om de temperatuur uit te lezen uit de DS18B20 op pin 2 (D2):

DS18B20_uitlezen

```
#include <OneWire.h>
// DS18B20 op pin 2.
OneWire ds(2);

void setup() {
  // Start seriele poort.
  Serial.begin(9600);
}

void loop() {
  byte data[2];
  ds.reset();
  ds.write(0xCC);
  ds.write(0x44);
  delay(750);
  ds.reset();
  ds.write(0xCC);
  ds.write(0xBE);
  data[0] = ds.read();
  data[1] = ds.read();
  int Temp = (data[1]<<8)+data[0];
  Temp = Temp>>4;
```

In Temp stoppen we de MSB (data[0]) en LSB (data[1]). Daarna wordt de inhoud nog eens 4 bits naar rechts geschoven.

```
  // schrijf temperatuur naar console.
  Serial.print(" T = ");
  Serial.print(Temp);
  Serial.println("'C");
}
```

Het resultaat wordt geprint op de serial monitor:

```
T = 21'C
T = 21'C
T = 21'C
T = 21'C
T = 21'C
T = 22'C
T = 22'C
```

➔ **Uitdagingen:**

1. Meet de temperatuur met de DS18B20 en sla 20 waardes op in de interne / externe EEPROM. (zie info in de vorige hoofdstukken).
2. Meet met een logic analyzer de MSB en LSB bij een bepaalde temperatuur. Onderzoek of deze waarde overeenkomt met de onderstaande tabel:

2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	LSB
MSb			(unit = °C)				LSb	
S	S	S	S	S	2^6	2^5	2^4	MSB

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	0000 0111 1101 0000	07D0h
+85°C	0000 0101 0101 0000	0550h*
+25.0625°C	0000 0001 1001 0001	0191h
+10.125°C	0000 0000 1010 0010	00A2h
+0.5°C	0000 0000 0000 1000	0008h
0°C	0000 0000 0000 0000	0000h
-0.5°C	1111 1111 1111 1000	FFF8h
-10.125°C	1111 1111 0101 1110	FF5Eh
-25.0625°C	1111 1110 0110 1111	FF6Fh
-55°C	1111 1100 1001 0000	FC90h

*The power on reset register value is +85°C.

Meer info over de OneWire commando's kan je terugvinden in de datasheet van de DS18B20:

Convert T [44h]

This command begins a temperature conversion. No further data is required. The temperature conversion will be performed and then the DS18B20 will remain idle. If the bus master issues read time slots following this command, the DS18B20 will output 0 on the bus as long as it is busy making a temperature conversion; it will return a 1 when the temperature conversion is complete. If parasite-powered, the bus master has to enable a strong pullup for a period greater than t_{conv} immediately after issuing this command.

Read Scratchpad [BEh]

This command reads the contents of the scratchpad. Reading will commence at byte 0 and will continue through the scratchpad until the ninth (byte 8, CRC) byte is read. If not all locations are to be read, the master may issue a reset to terminate reading at any time.

Skip ROM [CCh]

This command can save time in a single drop bus system by allowing the bus master to access the memory functions without providing the 64-bit ROM code. If more than one slave is present on the bus and a Read command is issued following the Skip ROM command, data collision will occur on the bus as multiple slaves transmit simultaneously (open drain pulldowns will produce a wired AND result).

DS18B20 COMMAND SET Table 4

INSTRUCTION	DESCRIPTION	PROTOCOL	1-WIRE BUS AFTER ISSUING PROTOCOL	NOTES
TEMPERATURE CONVERSION COMMANDS				
Convert T	Initiates temperature conversion.	44h	<read temperature busy status>	1
MEMORY COMMANDS				
Read Scratchpad	Reads bytes from scratchpad and reads CRC byte.	BEh	<read data up to 9 bytes>	
Write Scratchpad	Writes bytes into scratchpad at addresses 2 through 4 (TH and TL temperature triggers and config).	4Eh	<write data into 3 bytes at addr. 2 through. 4>	3
Copy Scratchpad	Copies scratchpad into nonvolatile memory (addresses 2 through 4 only).	48h	<read copy status>	2
Recall E^2	Recalls values stored in nonvolatile memory into scratchpad (temperature triggers).	B8h	<read temperature busy status>	
Read Power Supply	Signals the mode of DS18B20 power supply to the master.	B4h	<read supply status>	

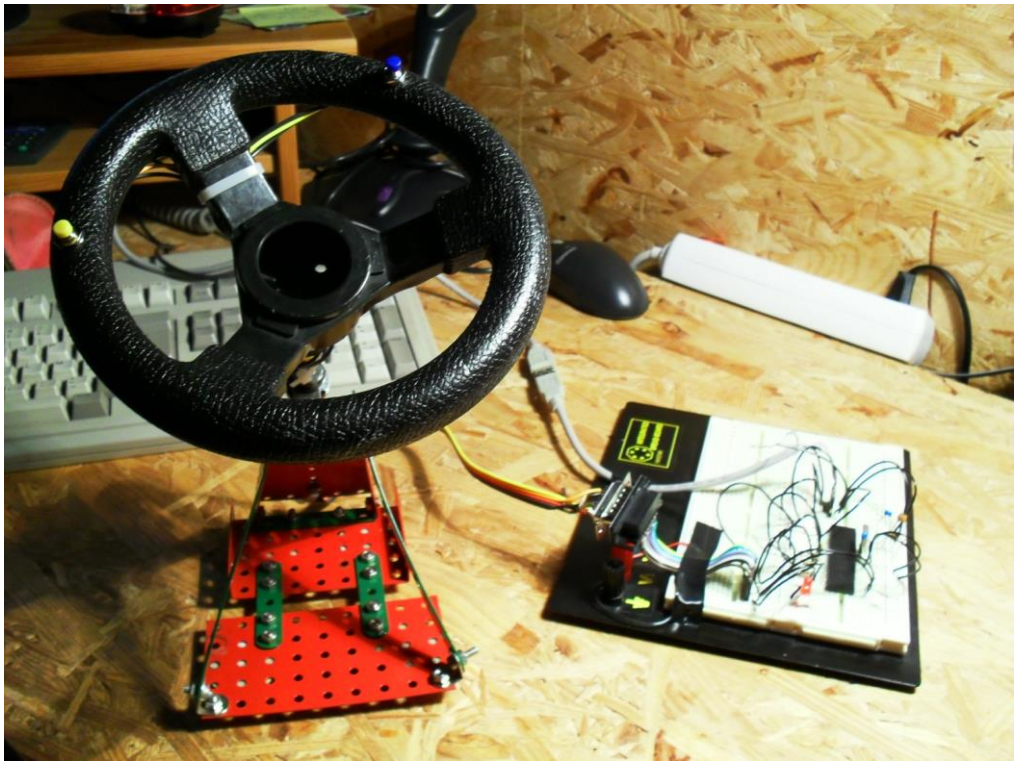
4 Maak een USB game controller

Onze UNO hangt met een USB kabel aan de PC. Dat wil zeggen dat er een communicatie protocol is opgezet om via USB met de UNO / Leonardo te communiceren.

In dit hoofdstuk gaan we uitzoeken hoe USB werkt en hoe we dit kunnen inzetten in onze projecten.

We gaan ook uitzoeken hoe we een HID device (human interface device) kunnen bouwen, zoals een keyboard of een joystick.

4.1 Hoe werkt het USB protocol?



Op deze figuur zie je een eerste poging bij het maken van een USB joystick om te gamen met het spel CARS of de flight simulator.

4.1.1 De USB quiz

Waar informatie vinden?

www.usb.org

http://www.usb.org/developers/onthego

http://nl.wikipedia.org/wiki/Universal_Serial_Bus

zie je eigen cursus

zie elektor artikel i.v.m. USB3.0 op smartschool

Doel is om per groep opzoekingen te doen van de volgende USB items en deze te verwerken in 1 grote klas PPP. Knip en plak alle tekeningen en info netjes in deze PPP zodat je achteraf dit vlot vooraan kan komen vertellen in groep.

Elke groep/lln zal uiteindelijk enkele items vooraan moeten komen uitleggen van hun PPP.

Vragen:

1. Welke USB logo's ken je (plak ze allemaal in je verslag) en waar staan ze voor.
2. Waarom kiest men voor USB? Geef 3 voordelen. Wat zijn nadelen van USB?
3. Welke USB snelheden bestaan er op dit moment + benaming. Kan je het verband vinden tussen de verschillende snelheden? Geef bij elke snelheid 2 toepassingen + foto.
4. Kan je een USB apparaat zomaar inpluggen of heeft de PC hier nog iets voor nodig? Leg dit uit aan de hand van een flowchart. Waar kan je in windows weten of een USB apparaat goed geïnstalleerd is?
5. Hoeveel draden zitten er in een USB connector? Waarvoor dienen deze en welke kleur hebben ze? Wat is de maximum spanning op de USB datalijnen. Wat is de maximum stroom dat een USB apparaat mag verbruiken/kan krijgen van een PC? Kijk ook een USB 3.0 connector na. Wat is het nut hier van de draden?
6. Welk type seriële verbinding hebben we hier? Maak een meting met de digitale scope/logic analyzer en een BOB van een USB signaal terwijl je een USB apparaat (muis, printer, PLC….) aanstuurt via een PC. Duid de D+/D- en idle tijd aan. Tot welke snelheidsklasse hoort jouw gemeten apparaat?
7. Welke soorten USB connectoren bestaan er? Toon een foto van elk type. Hoeveel verschillende soorten USB kabels bestaan er? Maakt de snelheid hier iets uit?
8. USB classes, wat weet je daarover? Bespreek er minstens 4. Bespreek zeker de HUB. Wat is zijn nut?

9. Hoe zit een USB topologie in elkaar. Wat is het maximum in te pluggen devices op 1 PC? Wat is downstream en upstream? Hoe diep kan je gaan in een topology? Wat is het maximum aantal meters USB kabel dat je kan gebruiken om iets langs te sturen?
10. Wat is een frame? Hoelang duurt dit? Maak een schets van een USB datastream. Wordt er een klok gebruikt bij de USB signalen of hoe lost men dit op?
11. Wat is een enumeratie? Hoe kan een PC de snelheid herkennen van een USB apparaat?
12. Kan ik een lampje veilig aansluiten op een USB connector van een laptop?
13. Wat zijn endpoints? Welke soorten hebben we? Wat is hun nut?
14. Zoek de volgende termen uit: single ended, differentieel, twisted pair, serieel, asynchroon, klokwinning via de sync, bitstuffing.
15. Zoek de website op van volgende USB apparaten (via google -> afbeeldingen kom je er wel ☺ Waar dienen ze voor en welke snelheid hebben ze:

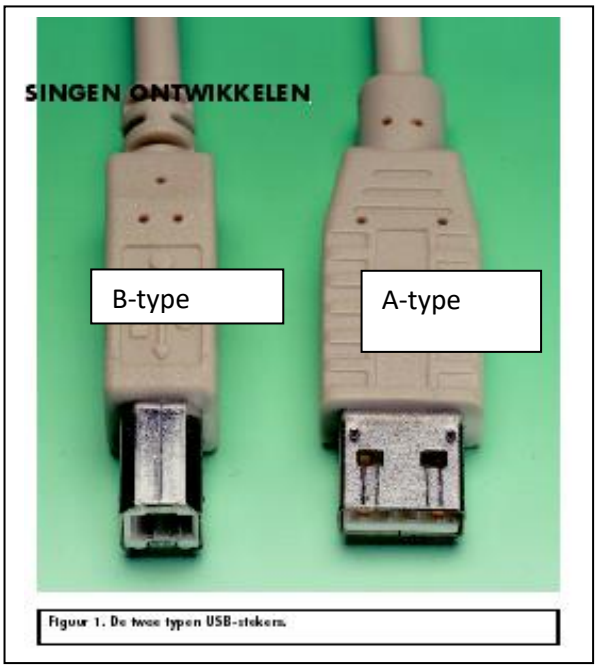
4.1.2 De USB informatie

USB is een serieel bussysteem (Universal Serial Bus) waarop meerdere apparaten aangesloten en geadresseerd kunnen worden.

USB is aanmerkelijk complexer dan RS232 en met maximaal 1,5 Mbit per seconde (lowspeedapparatuur) of 12 Mbit per seconde (fullspeed-apparatuur) ook veel sneller.

Ondertussen is er ook reeds de USB High speed (480Mb/s) en de USB supperspeed (4.8Gb/s). Ook USB OTG en Wireless USB zijn reeds ontwikkeld.

Bovendien hebben de ontwerpers die aan de wieg stonden van USB grote nadruk gelegd op eenvoud en betrouwbaarheid in het gebruik.



Figuur 1. De twee typen USB-stekers

Eigenlijk heeft ‘Plug&Play’ met de komst van USB pas werkelijk betekenis gekregen. Normaal gesproken wordt een USB-apparaat namelijk eenvoudigweg bij al dan niet ingeschakelde PC aangesloten en initialiseert zich vervolgens automatisch. Het gevecht tegen dubbel gebruikte interrupts, foutieve adressen en ontbrekende drivers is eindelijk voorbij (meestal toch ☺.

Deze vereenvoudiging voor de gebruiker betekent een aanzienlijk grotere inspanning voor de fabrikanten. Wie zelf een USB-apparaat wil ontwikkelen, zal zich dan ook in de complexe materie van USB moeten inwerken, uiteindelijk een USB- microprocessor moeten programmeren en een bijbehorende driver moeten schrijven. verderop zijn hiervoor een aantal informatiebronnen opgesomd. Bij flowcode kan je een eigen driver aanmaken.

4.1.3 USB: kabels en voeding

De USB kent verschillende typen stekers, de meest bekende is het type A en type B. Het systeem is zo ontworpen dat verwisseling onmogelijk is. In tegenstelling tot RS232 is er ook geen onderscheid tussen zogenaamde gekruiste en rechte verbindingen.

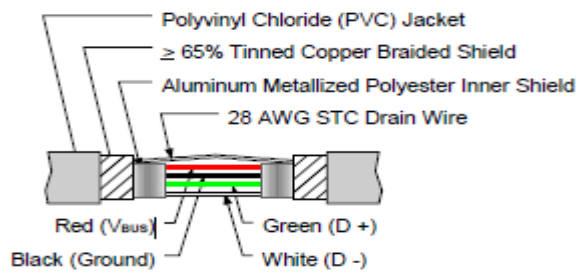
Kabels zijn altijd 1-op-1 bedraad en de penbezetting is altijd hetzelfde:

1 +5V (rood)

2 Data – (wit)

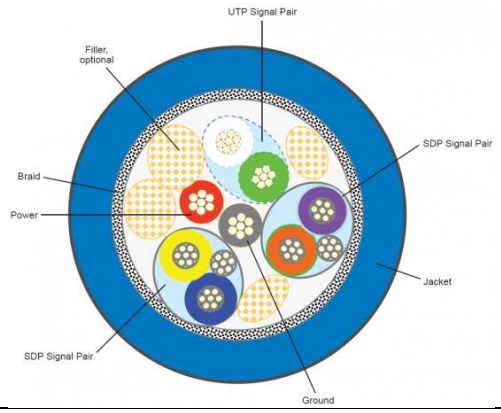
3 Data+ (groen)

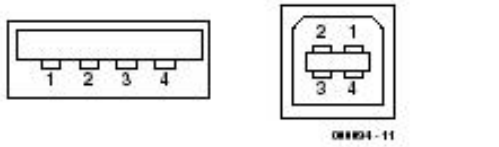
4 Massa (zwart)



Voor LS, FS en HS devices

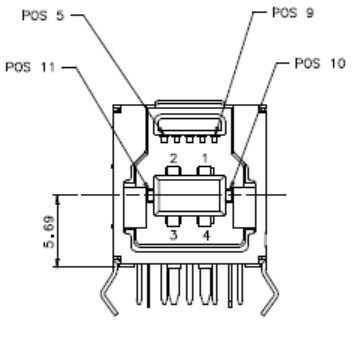
Voor SS (SupperSpeed) is de connector anders opgebouwd

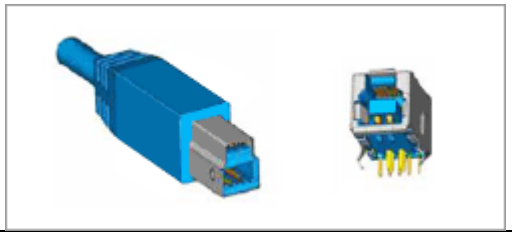




Figuur 2. Aansluitgegevens van stekers A en B.

Figuur 3. USB-chassisdeel type B.





Zie ook de mini USB A en B type in de USB spec. Deze vind je bijvoorbeeld terug op jouw MP3 speler. Ondertussen zijn er nog meerdere combinaties mogelijk sinds de USB 3.0 spec.

Aan de achterkant van een moderne PC zitten gewoonlijk minimum twee connectoren van het type A. Hier kunnen twee apparaten direct op aangesloten worden.

Kleinere lowspeed-apparaten, zoals bijvoorbeeld een muis, gebruiken een dunne, vast aangesloten kabel met aan het uiteinde een connector van het type A. In alle andere gevallen beschikt het apparaat zelf over een type B chassisdeel. De verbinding wordt dan gemaakt met een type A-B kabel. Deze kabels zijn alleen kant-en-klaar verkrijgbaar, met aangegoten connectoren.

Losse USB-stekkers zijn niet verkrijgbaar (wel in de elektronica zaak ☺. De lengte, de kabeldoorsnede, de afscherming enzovoort zijn allemaal nauwkeurig gespecificeerd. Ook het onderscheid tussen fullspeed en lowspeed speelt hierbij een rol. Het systeem met de gespecificeerde bekabeling verhindert op betrouwbare wijze dat een lowspeed

kabel voor een fullspeed verbinding ingezet zou worden. Alle losse verbindingskabels zijn namelijk fullspeed. Lowspeed kabels zitten altijd vast aan apparatuur gemonteerd. Ook is er nog een type A-A verlengkabel beschikbaar, mocht deze nodig zijn (zie onze Quartus bordjes).

Ondertussen is er ook de USB High Speed kabel. Hiervoor vind je een speciaal logo of sticker op de kabel. Deze kabel is speciaal gemaakt voor hoge snelheden omdat de opbouw van de kabel anders is. Zie spec 3.0 Mechanical part. Ook de USB Supperspeed kabel vraagt zijn eigen speciale kabel. Haal dus de snelheden en kabels niet door elkaar. Dit kan problemen geven.

Hoewel losse USB-connectoren voor kabelmontage dus niet verkrijgbaar zijn, zijn er wel USB-chassisdelen voor printmontage te koop. Zelf experimenten uitvoeren is dus zeker mogelijk (zie onze PIC robot, enz …).

De USB-aansluiting stelt een voeding van +5 V ter beschikking, die zonder verdere maatregelen tot 100 mA belast kan worden.

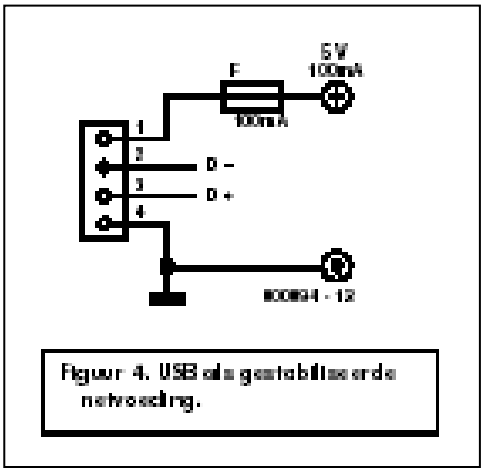
Dat is voldoende voor een klein bureaulampje, maar natuurlijk ook voor zinvollere toepassingen. Veel digitale schakelingen en microcontrollers werken op 5 V. Als men zulke schakelingen in combinatie met de PC gebruikt, is dus ook meteen de voeding verzorgd. Er moet echter beslist op worden gelet dat een goede kortsluitbeveiliging toegepast wordt. Hiervoor lenen zich een normale smeltveiligheid of een zogenaamde polyswitch-zekering.

De twee data-aansluitingen D+ en D zijn alleen te gebruiken met speciale USB-componenten, dedicated microcontrollers dus. Deze kunnen dan direct via de USB-aansluiting van voeding worden voorzien. Na een speciaal commando richting PC kan een USB-apparaat zelfs tot 500 mA opnemen. De voedingsspanning op de bus mag maximaal

5,25 V bedragen en bij grotere belastingen afnemen tot 4,2 V. Een spanningsregelaar zal zelfs in dat laatste geval nog een stabiele 3,3 V kunnen leveren.

Het totale systeem van kabels en apparatuur is zo ontworpen dat ook bij de maximale belasting de voedingsspanning nooit onder 4,2 V zal zakken. Apparaten die meer dan de genoemde 100 mA nodig hebben, moeten hun vermogensbehoefte aan het systeem doorgeven. Zulke apparatuur wordt alleen toegelaten als er nog genoeg vermogensreserve beschikbaar is.

Er wordt onderscheid gemaakt tussen apparatuur met een ingebouwde netvoeding (**self powered)** en apparaten die uit de USB gevoed worden (**bus powered**). In de meeste gevallen kan tussen beide mogelijkheden gekozen worden. Een apparaat is dan bijvoorbeeld uitgerust met een aansluiting voor een externe voeding waarop, als dat nodig is, een netadapter op aangesloten kan worden. Volgens de USB-specificaties is de stroomopname uit de bus automatisch begrensd. Als er meer dan de toegelaten stroom opgenomen wordt, zal de voeding uitgeschakeld worden. Zie hiervoor de windowsmelding op het scherm.



Figuur 4. USB als gestabiliseerde netvoeding.

4.1.4 USB categoriën

Meestal wordt bij de aanschaf van een muis een diskette met bijbehorende driver geleverd. Als echter tegenwoordig een USB-muis wordt gekocht, zal tot veler verrassing de diskette ontbreken. Nadat men van de schrik bekomen is en de muis ondanks alle twijfel toch is aangesloten, volgt een nog grotere verrassing. Het besturingssysteem Windows 7, XP en 98 vindt de driver namelijk automatisch!

De **HID-driver** (Human Interface Device) die voor dit soort toepassingen al meegeleverd was, wordt vanzelf geladen. De muis hoort namelijk tot een gespecificeerde categorie van apparaten waarvoor kant-en-klare drivers beschikbaar zijn.

Tot de **HID-apparaten** behoren muizen, keyboards, aanwijsmiddelen en joysticks. Behalve voor HID's bestaan er ook nog USB-categorieën voor geluidskaarten, printers enzovoort.

Alle 'normale' apparaten zijn dus in bepaalde categorieën ingedeeld, waarvoor standaard drivers in het systeem beschikbaar zijn. Daarmee is tegelijkertijd een zekere standaardisering opgelegd, omdat de fabrikanten van USB-apparatuur zich nauwgezet aan het raamwerk van de specificaties moeten houden. Tegenwoordig zijn apparaten die met USB zijn uitgerust, niet zo duur meer.

4.1.5 Bus topologie

De USB is een stervormige bus met één master. Ze noemen het ook een "getrapte ster" (**TIERED STAR of soms ook TREE**)(zie basiscommunicatie cursus voor meer info).

Om meerdere USB-randapparaten aan te kunnen sluiten, is een hub nodig.

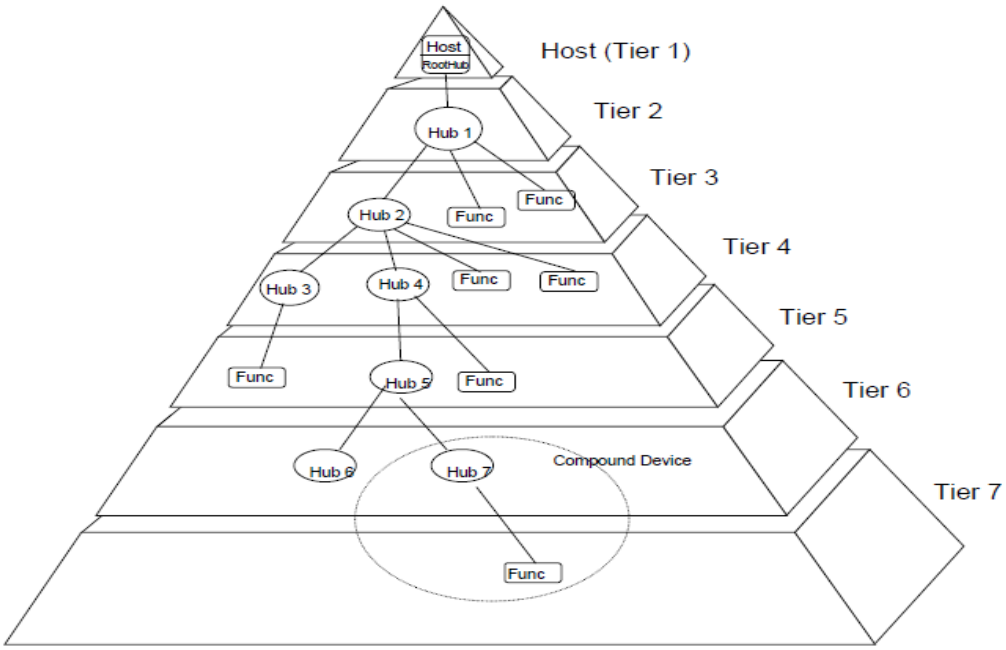


Figure 4-1. Bus Topology

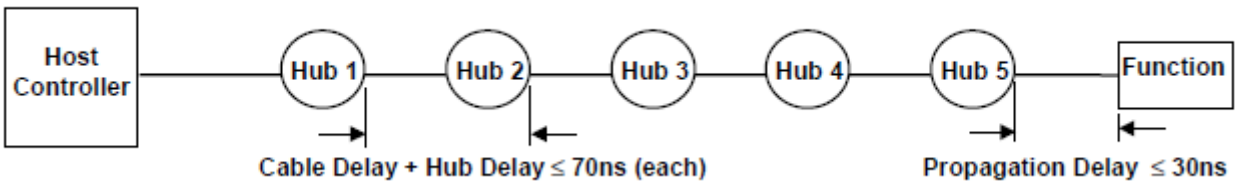
Een hub is een busverdeler met meerdere poorten. De naam is afkomstig van het engelse woord voor wielnaaf. Hub is een aanduiding voor de stervormige verbindingen, net zoals een naaf de spaken van een wiel verbindt.

Het meest gebruikelijke is een externe hub, met één upstream-poort en vier downstream-poorten. De PC zelf heeft ook al een hub ingebouwd, om de twee of meerdere

aansluitingen mogelijk te maken. Deze zogenaamde root-hub is geïntegreerd

op het moederbord.

Op de downstream-poort van een hub kan weer een volgende hub worden aangesloten. In totaal kunnen zo **tot vijf hubs achter elkaar** geplaatst worden. Hieruit is ook het **maximale aantal apparaten** af te leiden dat aan te sluiten is, **namelijk 127**. Dit is een theoretische waarde die volgt uit het feit dat de beschikbare bandbreedte op de bus verdeeld moet worden onder alle aangesloten apparaten.



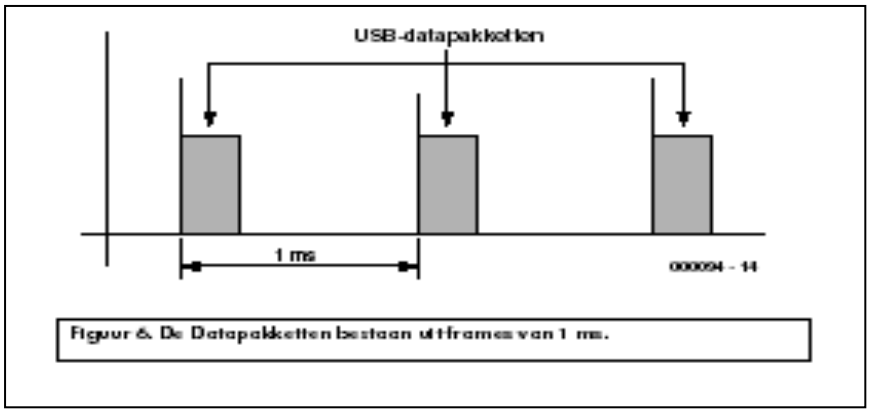
Het **adres 0 is het aansluitadres**. Elk device gaat luisteren op dit adres bij het inpluggen. Daarna geeft windows een volgend nummer = adres in de rij.

4.1.6 USB signalen

De signalen op de leidingen D+ en D-werken met spanningsniveaus van 0 en

3,3 V. De voedingsspanning van de aangesloten microcontroller zal in veel gevallen

3,3 V zijn. De USB is een bus met maar één master. Alle activiteiten worden door de PC gecoördineerd. Hij broadcast de data (net als bij een TV of radio uitzending). De data wordt in pakketten van 8 tot 256 bytes verzonden en ontvangen.



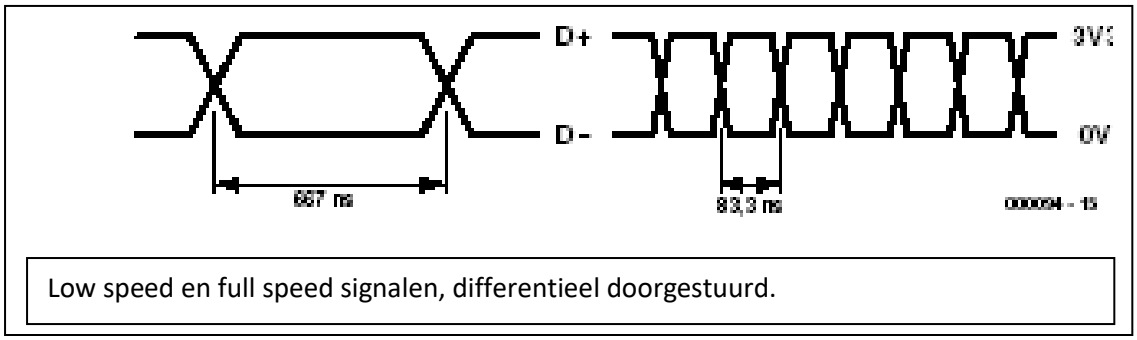
Figuur 6. De Datapakketten bestaan uit frames van 1 ms.

Een HS device heeft 1000 micro frames in 1 frame van 1ms

.

De PC kan data van een apparaat opvragen. Omgekeerd echter kan een apparaat niet uit zichzelf data versturen. Een uitzondering hiervoor is echter de **remote wakeup** functie die instelbaar is in de device manager voor een bepaald USB apparaat dat deze functie ondersteunt. Er moeten alleszins knoppen op het device aanwezig zijn zoals bij een muis of gamepad.

Het dataverkeer is opgedeeld in tijdslots (**frames**) van precies één milliseconde. In een tijdslot kunnen na elkaar pakketten voor meerdere apparaten afgehandeld worden. Hierbij kunnen lowspeed en fullspeed-pakketten tegelijkertijd voorkomen.



Low speed en full speed signalen, differentieel doorgestuurd.

Noteer hier wat het nut is van differentieel doorsturen (zie ook in de cursus basiscommunicatie hfdst basisbegrippen):

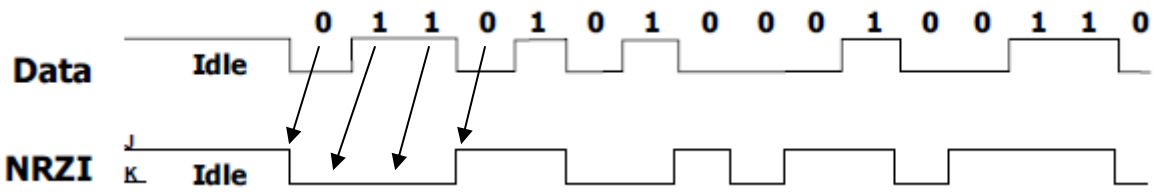
Een USB host communiceert met de devices via **endpoints.** Er bestaan control, isochronous, interrupt en bulk endpoints.

- **Control endpoints:** om de algemene stuurdata en info te verzenden tussen host en slave (vb geef een descriptor, …). Elk device heeft een control endpoint.
- **Interrupt endpoint**: een soort register dat bijvoorbeeld bij een muis om de 100ms wordt afgevraagd. Is de muis bewogen, dan zal de veranderde data ingelezen worden.
- **Bulk endpoint**: om heel veel data gecontroleerd (via CRC check) door te sturen. Vb bij een USB stick mag de data van de files niet verloren gaan, zelfs niet 1 bitje. Merk op dat een CRC toevoegen als nadeel heeft dat alles trager wordt. Dit omdat er meer data verstuurd moet worden.
- **Isochronous endpoint:** om veel data zonder controle door te sturen. Dit gaat sneller dan bij de bulk omdat de controle wegvalt (minder overhead). Een camera of printer hoeft bijvoorbeeld niet exact elk puntje juist te versturen. Dit zie je toch niet met het blote oog.

Als meerdere apparaten aangesproken worden, zorgt de hub voor de verdeling. Lowspeed-apparaten werken met een datarate van 1,5 Mb/s.

Een bit duurt dus precies 666,7 ns. Bij fullspeed-verbindingen op 12 Mb/s is dat nog maar 83,33 ns. De snelheid wordt alleen door de master bepaald. De ‘slaven’ zullen moeten synchroniseren met de datastroom.

Omdat geen apart kloksignaal wordt doorgegeven, moet de klok uit de data teruggewonnen worden. Daarom wordt een NRZI-codering (Non Return to Zero Inverted) gebruikt (zie voor meer info in de USB spec). **Nullen** in de data veroorzaken hierbij een **niveauverandering**, terwijl **enen** het niveau **onveranderd** laten.



Een USB-apparaat geschikt in het algemeen over meerdere FIFO's waarin data-overdracht kan laatsvinden. Na het apparaat-adres volgt nog een 'endpoint'-adres dat aangeeft waar de data naar toe gaan of waar ze vandaan gehaald moet worden. Zo heeft een USB muis bijvoorbeeld altijd een endpoint 0 en een endpoint 1. Endpoint 0 wordt hier gebruikt tijdens de initialisatie. De eigenlijke muisdata worden door de microcontroller in de interrupt endpoint-1-FIFO geschreven en van daaruit naar de PC gestuurd.

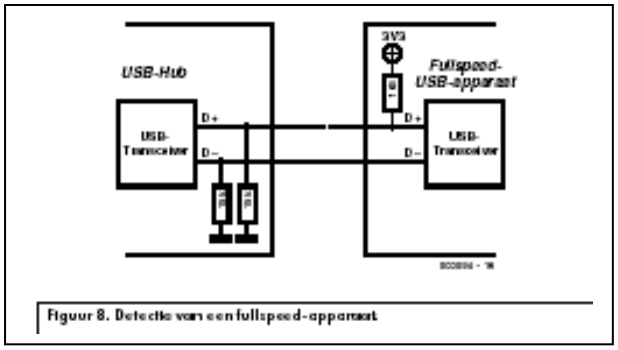
De USB-software vormt zogenaamde pipes naar de afzonderlijke **endpoints**. Een pipe is een logisch kanaal naar een endpoint in een apparaat. Een pipe is voor te stellen als een datakanaal dat bestaat uit een enkele draadverbinding. In werkelijkheid worden de data in een pipe echter als een pakket in een **frame van een milliseconde** doorgegeven. Door de hardware worden de data uiteindelijk aan de hand van het endpoint- adres op de juiste fysieke geheugenadressen geplaatst. Een apparaat kan meerdere pipes tegelijkertijd inzetten, waardoor de beschikbare datarate dan evenredig toeneemt.

Merk op dat een HS device om de µs een frame heeft. Vandaar de naam **µframe**.

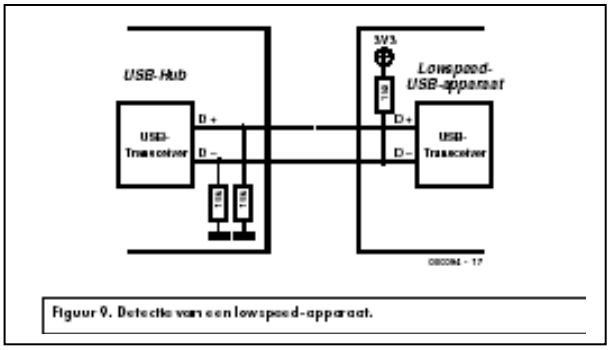
4.1.7 Enumeratie

Een niet gebruikte USB-aansluiting is niet actief en de hub stuurt er geen pakketten naar toe. Beide signaalleidingen zijn laag en hebben een uitgangsweerstand van 15 kohm.

Ieder USB-randapparaat heeft intern een weerstand van 1,5 kohm die een van beide signaalleidingen met +3,3 V verbindt. Bij een fullspeed-apparaat wordt D+ hoog gemaakt, bij een lowspeed-apparaat D-.



Figuur 8. Detectie van een fullspeed-apparaat.



Figuur 9. Detectie van een lowspeed-apparaat.

De hub herkent hieraan het type apparaat en zal de dataverbinding met de juiste overdrachtssnelheid opbouwen.

Bij het aansluiten en de daarop volgende toekenning van een ID aan het apparaat, de enumeratie, wordt automatisch een passende driver geladen. Omgekeerd zal het besturingssysteem ook herkennen dat een apparaat van de bus verwijderd wordt. In dit geval zal tevens de driver uit het geheugen van de computer verwijderd worden.

Vanwege dit principe kan men dus betrekkelijk eenvoudig één USB-apparaat op meerdere PC's gebruiken. Het volstaat de kabel uit PC nummer 1 te verwijderen en bij PC nummer 2 weer aan te sluiten. Vervolgens zal er een nieuwe enumeratie plaatsvinden. Zo is bijvoorbeeld een USB-printer gemakkelijk op twee PC's te gebruiken.

Enumeratie volgorde:

1. Snelheid via pull-up weerstand te weten komen + chirping (bij HS)
2. Device descriptor via adres 0 inlezen (PC moet te weten komen welke driver class, stroom, self/bus powered het device nodig heeft).
3. Driver wordt geladen
4. Adres wordt aangepast op het laatste nieuwe adres
5. Rest van desciptors wordt opgehaald
6. Toestel is klaar voor gebruik

URL's voor meer informatie

www.usb.org : adres van de makers van USB. Ook zijn hier de vertegenwoordigers te vinden van de grote bedrijven die zich verenigd hebben om de USB-standaard te definiëren. De belangrijkste informatie is de USB-specificatie, die hier kan worden gedownload in PDF-formaat. Eigenlijk staat daar alles in. Maar gelukkig is dit niet de enige informatiebron.

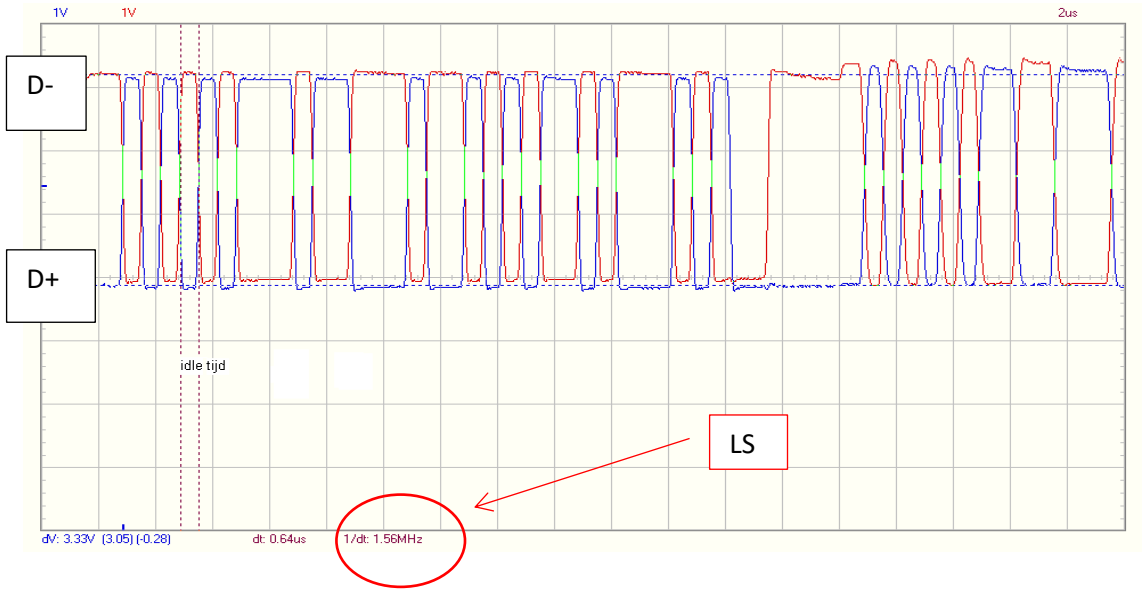
Craig Peacock (www.beyondlogic.org) heeft zelf een driver voor de USB-thermometer geschreven. Hij heeft een uitgesproken kijk op USB en hij heeft de opbouw van een USB-driver hier gedocumenteerd. Behalve dit biedt deze site nog meer unieke dingen die met USB te maken hebben.

Op de homepage van de auteur (home.t-online.de/home/b.kainka) is nog meer informatie beschikbaar en zijn verwijzingen naar andere literatuur te vinden.

Meer info over USB 3.0 vind je op smartschool in het project USB.

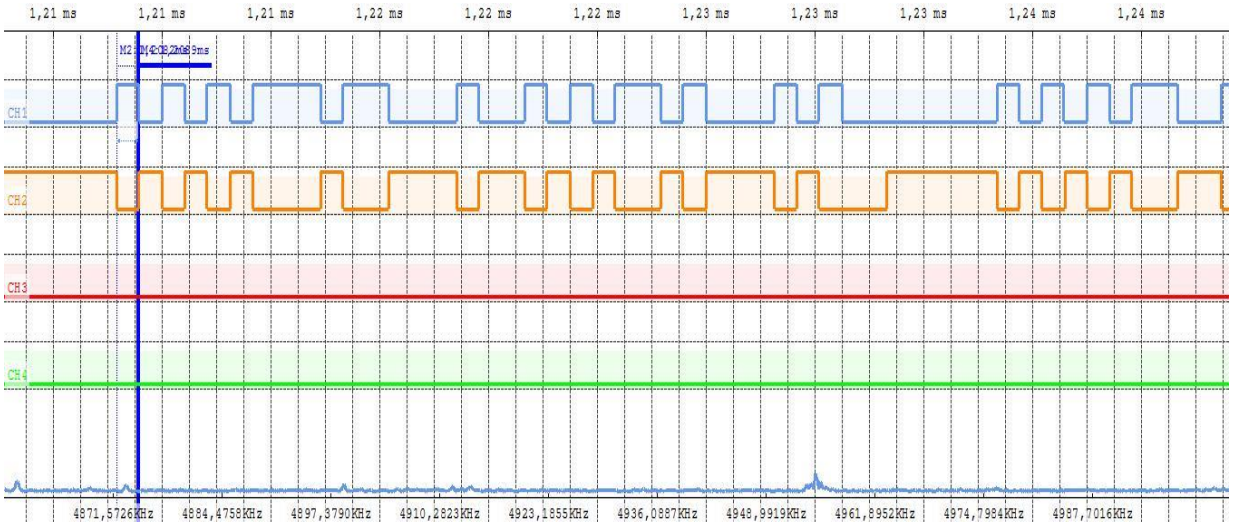
Hier vind je een elektor artikel van 2009 terug over USB 3.0

4.1.8 Metingen van een USB signaal met de logic analyzer en digitale scope:



Digitale scope meting van USB memory stick signaal. Het signal is 3.3V groot en de kleinste bit waarde is 0.64us. Deze waarde krijgen we omdat het een lowspeed device is (1/1.5MHz = 667ns = 0.66us)
Je kan een IDLE signaal gevolgd door een SYNC en data signaal mooi waarnemen. Ook een EOP op het einde van het data signaal kan je zien. Meer info vind je hierover terug in de USB spec.

Logic Analyzer meting van USB signaal:



Deze scan toont een meting van een low speed muis. De tijd van 1 bit is +/- 600ns bij de analyzer (1 / 1.5MHz = 667ns in werkelijkheid).
Kan jij de SYNC terug vinden. En de EOP.
Je kan ook mooi waarnemen dat dit een differentieel signaal is (D+ en D- zijn tegengestelden om de fouten, die gebeuren tijdens het sturen van de signalen, te minimaliseren in de ontvanger.

Je kan zien dat dit een LS apparaat is omdat de D- op CH2 hoog is in idle state terwijl D+ op CH1 laag is in idle state. D- wordt dus in idle state omhoog getrokken door de 1K5 pull-up weerstand.

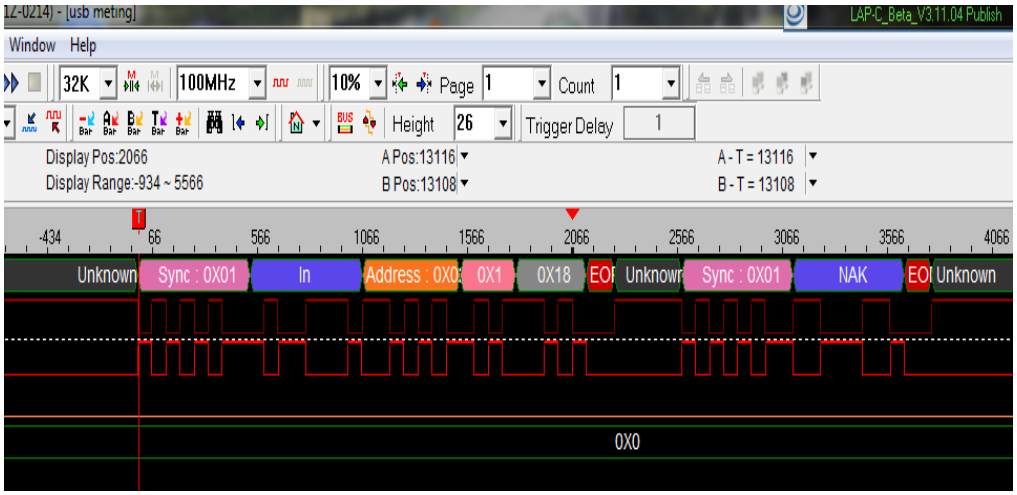
Merk op dat hogere snelheden niet kunnen gemeten worden met nog de scope, nog de analyzer. Daarvoor is de sample rate te klein. Om bloksignalen mooi te meten moet je minstens enkele harmonischen kunnen meten van het zelfde signaal. Bij 1.5MHz is dit bijvoorbeeld 3, 4.5, 6MHz. Bij een analyzer met 20MHz sample rate is dit geen probleem.

Als je echter 12MHz als start wil gebruiken (Full speed), dan kan je zelfs de eerste harmonische van 24MHz niet meer meten en krijg je dus een fout in je meting.

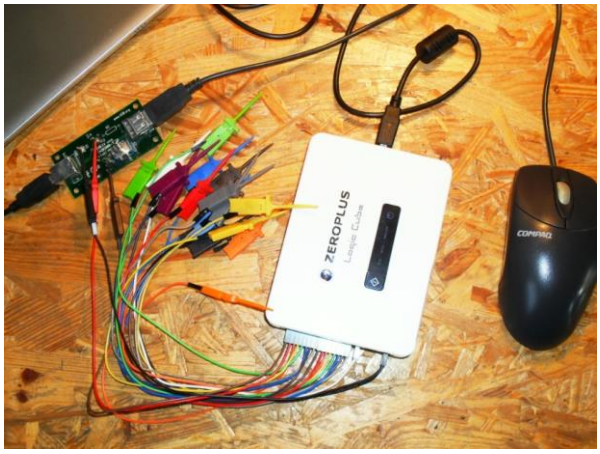
De digitale scope zou met zijn 60MHz in principe wel Full speed moeten kunnen meten.
Ken jij zo een apparaat? (vb de PIC robot).

Logic cube meting:

De volgende metingen zijn verricht met de LOGIC CUBE. A0 is verbonden met D- en A1 is verbonden met D+. Er is ook een massa aangesloten op het BOB (break out board). Wanneer je de A0 en A1 als bus instelt kan je ook de USB1.1 Decoder los laten op jouw meting.



Dit is de meting van een USB1.1 Muis (low speed, zie idle).

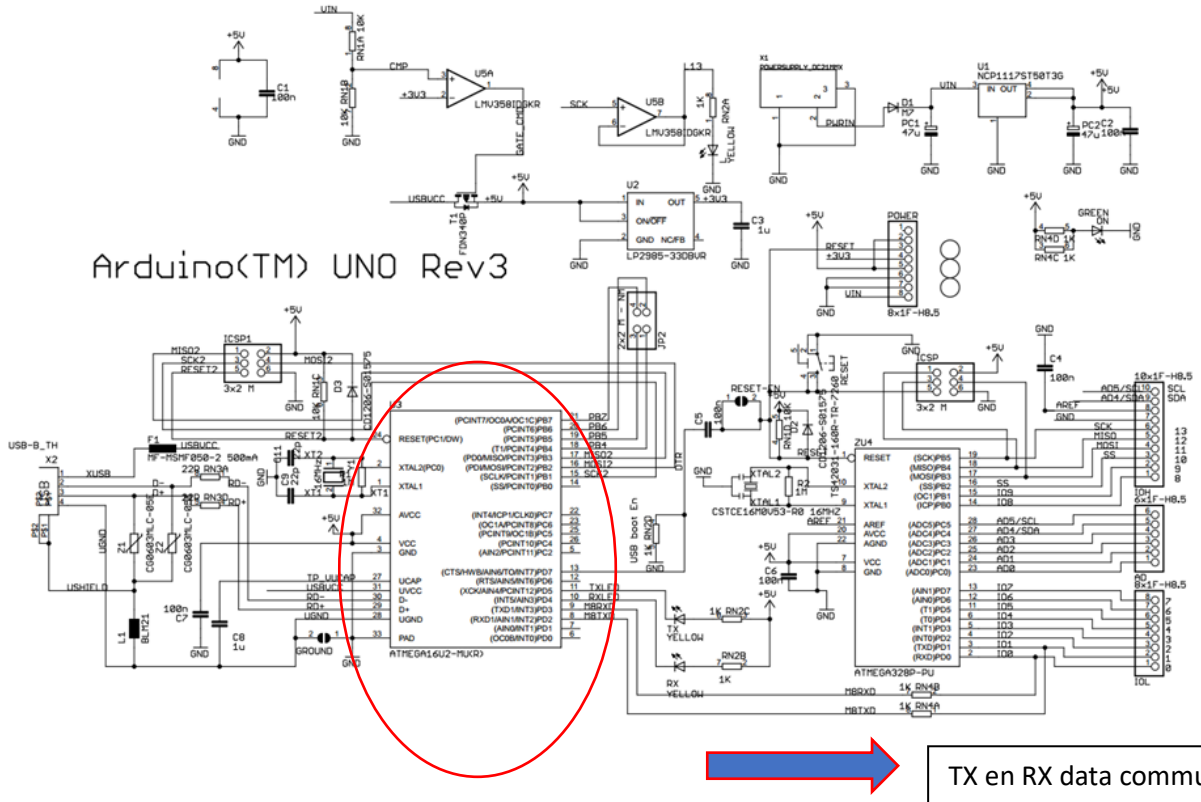


De muis wordt via het BOB verbonden met de PC. De logic cube is aangesloten op het BOB.

4.2 De Leonardo i.p.v. UNO gebruiken

Wanneer we naar de opbouw van de UNO gaan kijken, dan merken we dat de ATMEGA328P **geen eigen USB module aan boord** heeft. De USB communicatie wordt voorzien door een aparte chip, die direct na de USB-B connector is voorzien.

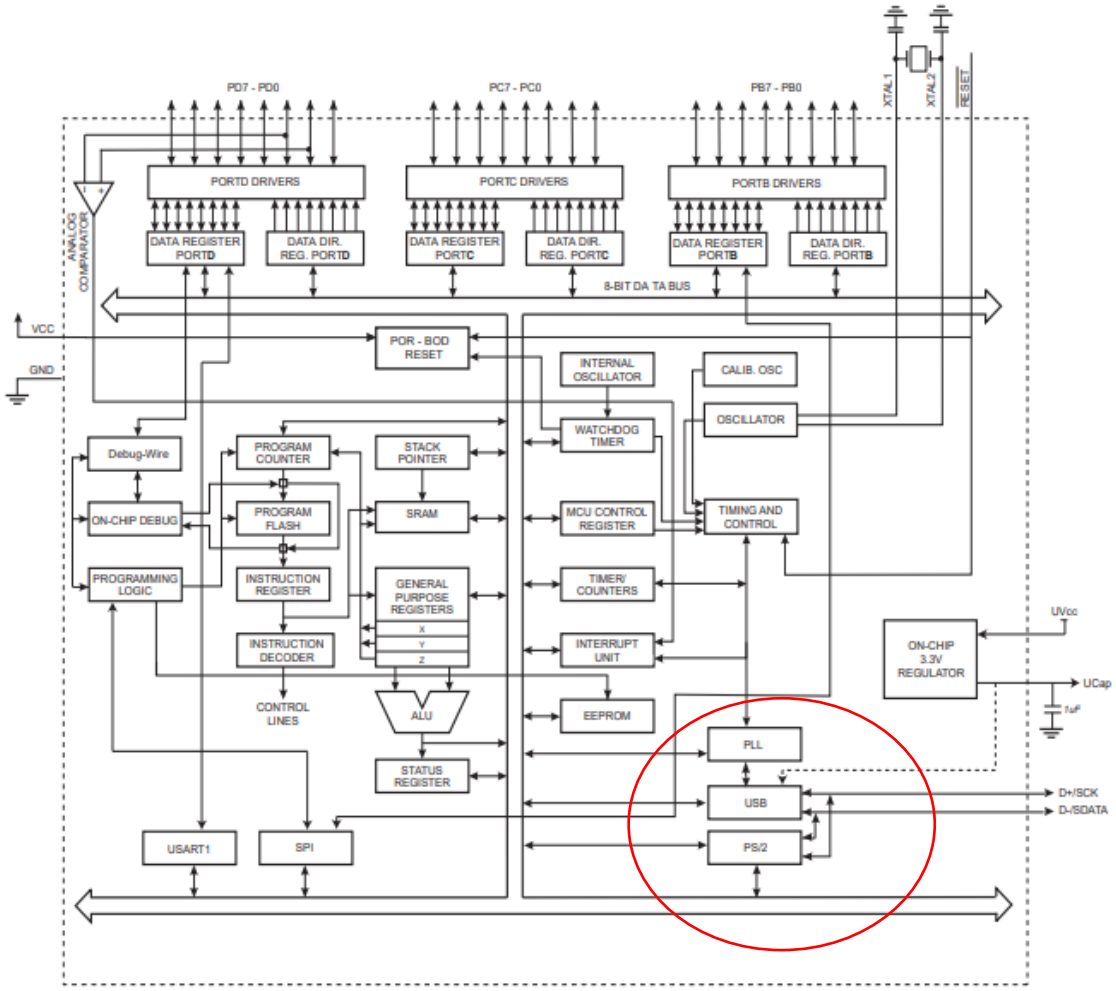
Bij de originele UNO is dit de ATMEGA16U2. Deze vormt een **omzetter tussen de USB en RS232** protocollen. Ze kiezen voor de ATMEGA328P omdat deze **makkelijk verwijderbaar** is (DIL versie) en een **beperkte complexiteit** bevat voor de beginner.



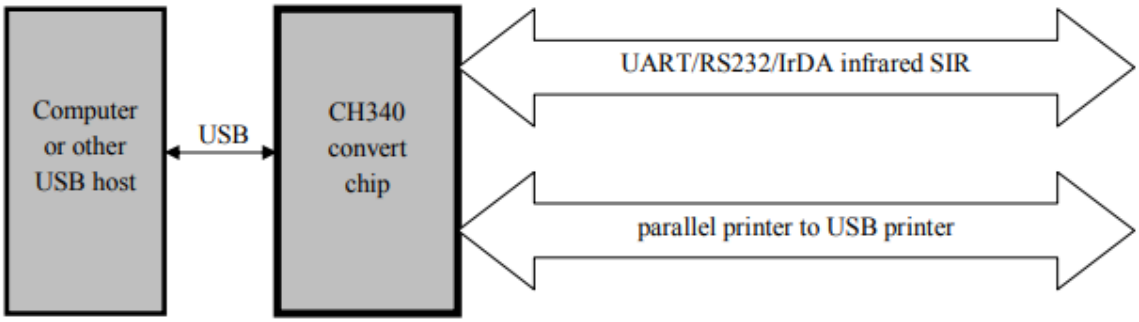
TX en RX data communicatie tussen ATMEGA16U2 en de ATMEGA328P

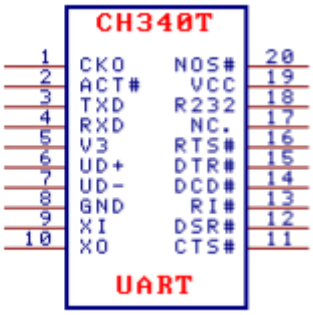
De ATMEGA16U2 is een chip met een USB module.

- Programming Lock for Software Security
- **USB 2.0 Full-speed Device Module with Interrupt on Transfer Completion**
  - **Complies fully with Universal Serial Bus Specification REV 2.0**
  - **48 MHz PLL for Full-speed Bus Operation : data transfer rates at 12 Mbit/s**
  - **Fully independant 176 bytes USB DPRAM for endpoint memory allocation**
  - **Endpoint 0 for Control Transfers: from 8 up to 64-bytes**
  - **4 Programmable Endpoints:**
    - **IN or Out Directions**
    - **Bulk, Interrupt and IsochronousTransfers**
    - **Programmable maximum packet size from 8 to 64 bytes**
    - **Programmable single or double buffer**
  - **Suspend/Resume Interrupts**
  - **Microcontroller reset on USB Bus Reset without detach**
  - **USB Bus Disconnection on Microcontroller Request**
- **Peripheral Features**



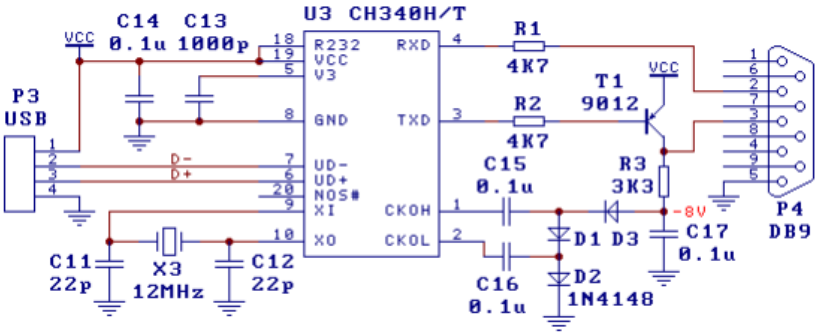
De chinese Arduino versie heeft een **CH340 chip** aan boord met USB naar RS232 module:



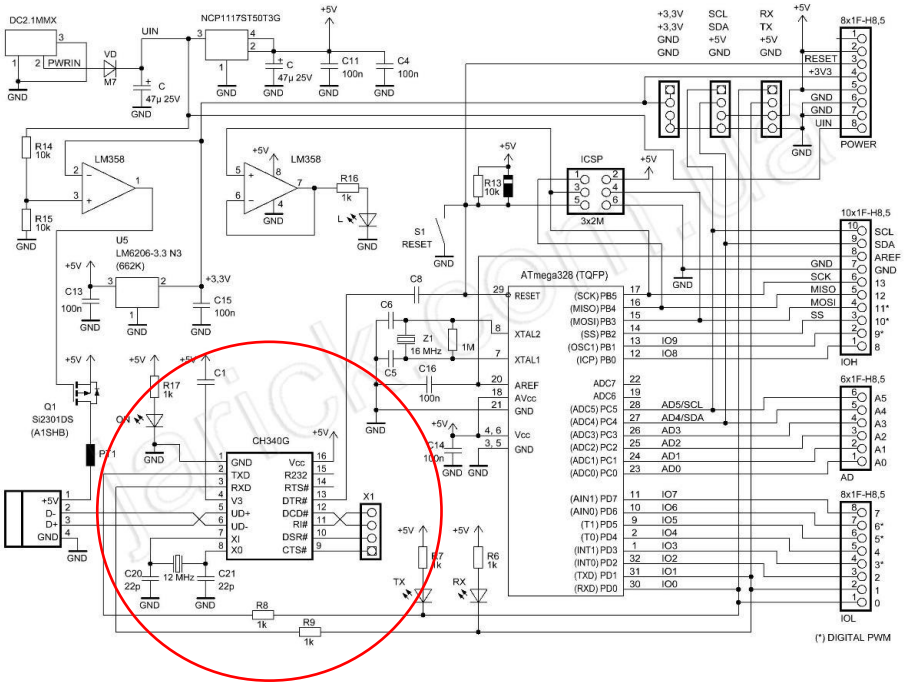


7.3. USB convert RS232 serial interface, handing edition (the following image)

The following image also is USB convert three RS232 serial interface, the function of this circuit is the same with 7.2. except the range of output RS232 is low. R232 in CH340 is high-level starts assistant RS232 function, only add diode、audion、resistor and capacitance can replace the special level convert circuit U5 in 7.2.. So the hardware cost is lower.



Info in de datasheet van de CH340 geeft aan hoe je deze chip moet aansluiten opdat het USB protocol wordt omgezet naar een RS232 protocol. In dit geval is er geen sprake van een complexe microcontroller die enkel een USB apparaat moet zijn.



Merk op dat de er geen beveiliging van de USB poort is bij de Chinese UNO!

Voordeel Chinese UNO:

Hij is goedkoper.

Nadeel Chinese UNO:

De Chinese UNO heeft een niet verwijderbare ATMEGA328P.

Je hebt een extra CH340 driver voor Windows nodig.

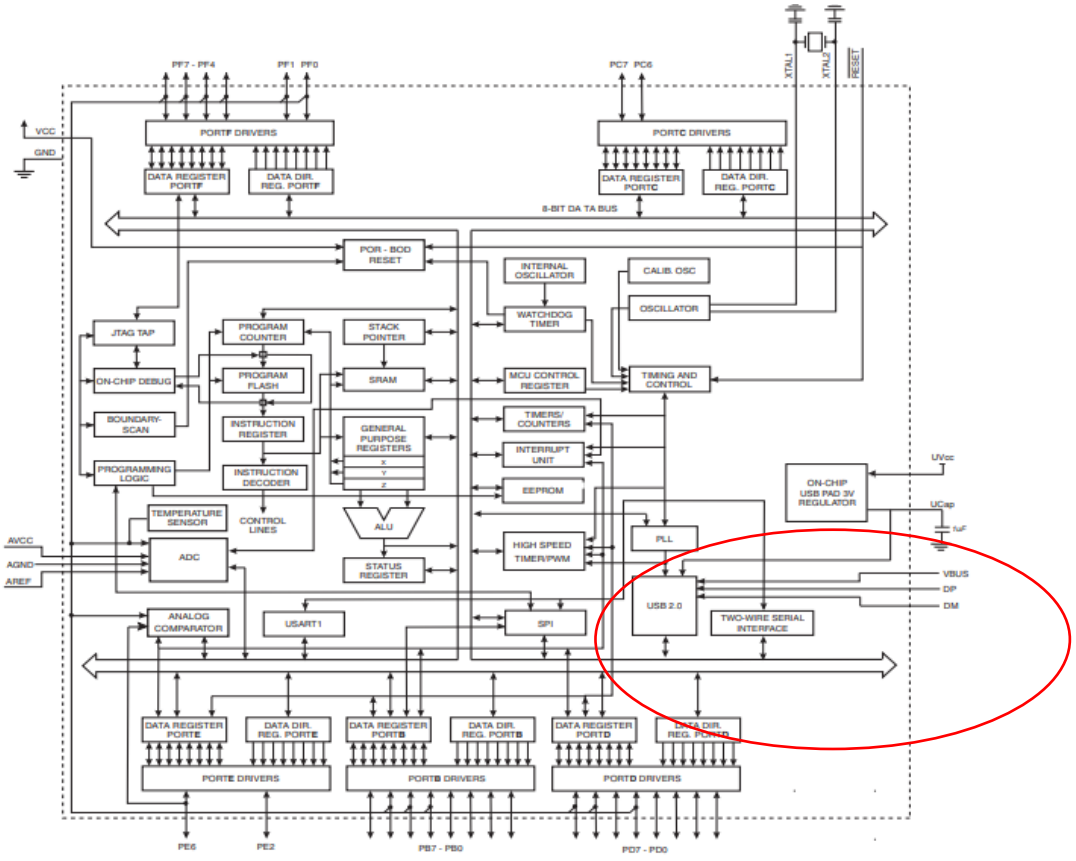
De USB poort is niet beveiligt.

De +5V power van de USB poort bevat niet altijd een 500mA multifuse.

Probleem voor beide UNO bordjes is dat we dus de **USB interface niet kunnen manipuleren via software.** Alle USB hardware ligt buiten de ATMEGA328P.

De oplossing hiervoor ligt bij het gebruiken van een ATMEGA waarbij wel de USB module in de chip zit die we met Arduino willen programmeren.

Dit is bijvoorbeeld bij de **ATMEGA32U4.** Deze chip zit o.a. op de **Arduino Leonardo en de Arduino micro.**

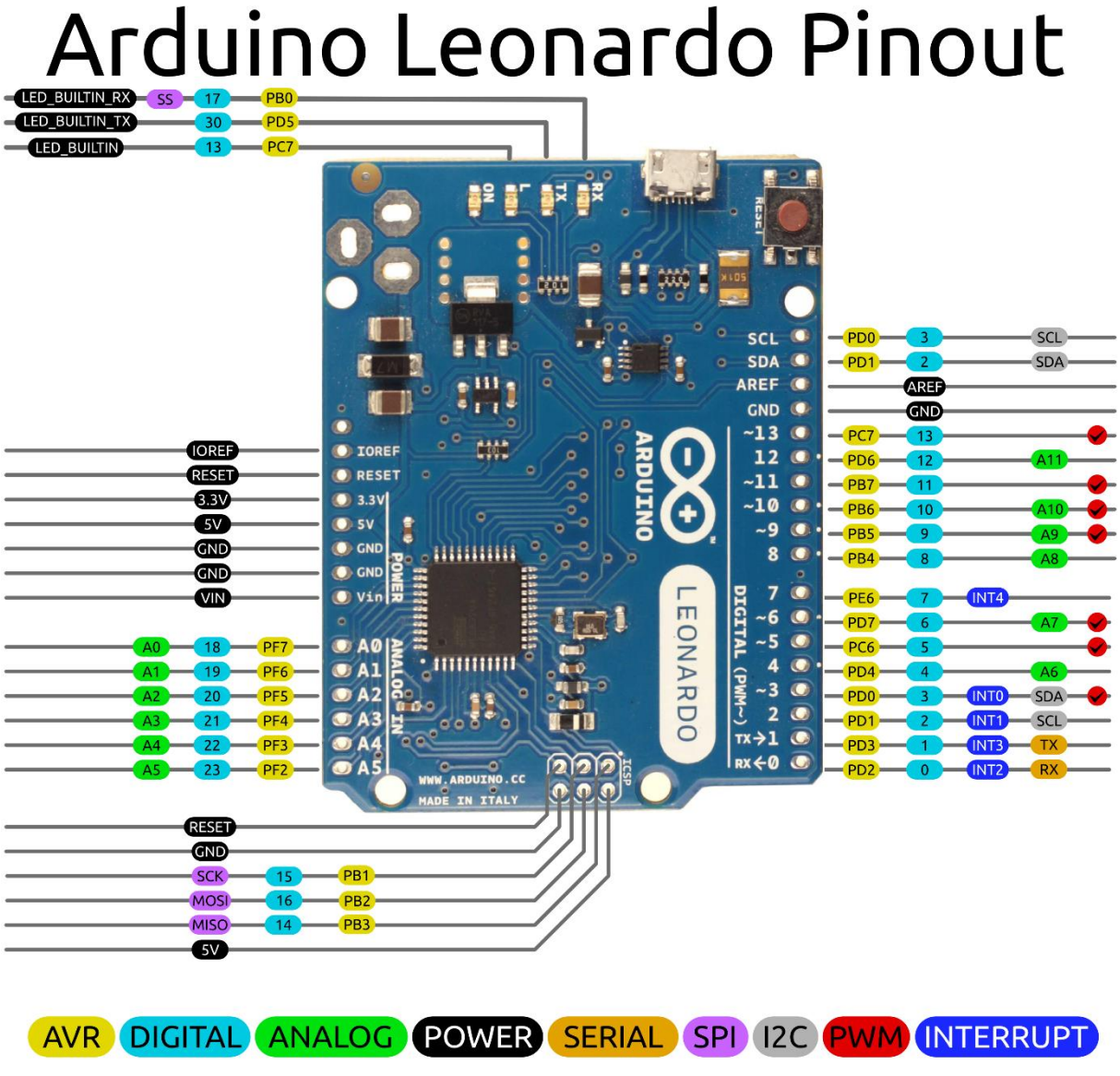


De **ATMEGA32U4 bevat een zelfprogrammeerbare USB module**

Wat kan de Arduino Leonardo allemaal meer t.o.v. een gewone UNO?

- USB programmatie (virtuele COM poort mogelijk, USB 2.0)
- Meerdere interrupts : INT0 tot INT4 -> UNO heeft enkel INT 0 en INT 1
- 7 PWM uitgangen t.o.v. 6 bij de UNO
- 12 analoge uitgangen t.o.v. 6 bij de UNO
- De Leonardo kan als een serieel USB device maar ook als muis en keyboard geprogrammeerd worden.

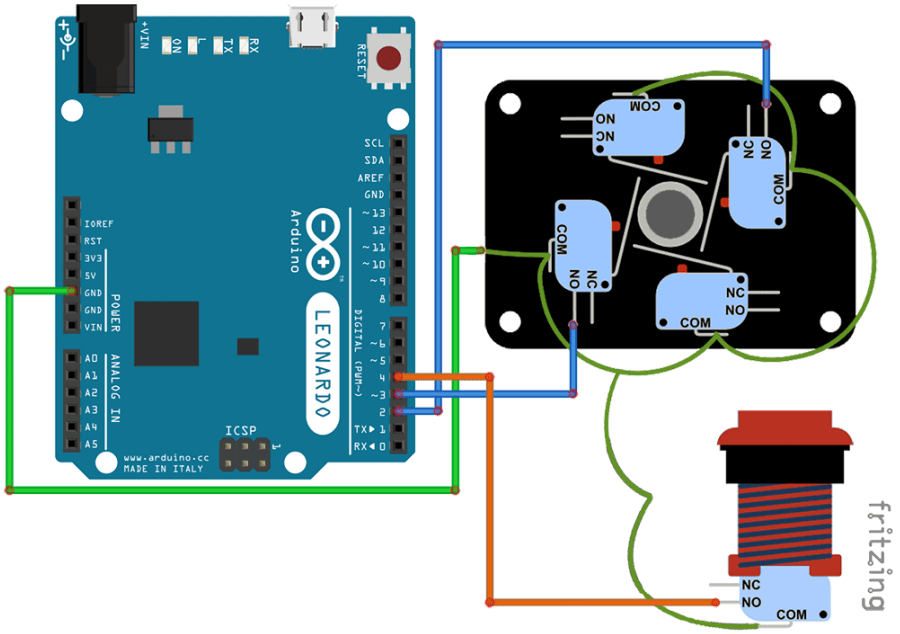
https://www.arduino.cc/en/Main/Arduino_BoardLeonardo



4.3 De USB gamecontroller met digitale knoppen testen

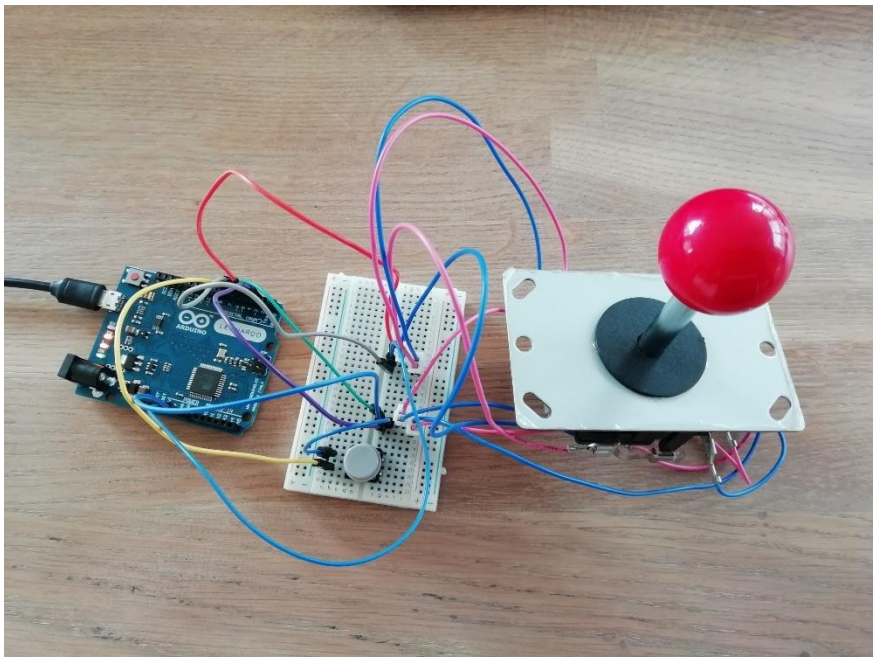
We willen in eerste instantie een USB gamecontroller maken van onze Leonardo waarbij we de standaard digitale knoppen gaan gebruiken. In dit geval zelfs laag actieve knoppen.

Op de volgende link vond ik een leuke arcade game controller met gewone knoppen:
https://www.brainy-bits.com/leonardo-arcade-controller/



Bouw deze schakeling maar gebruik nu de 4 knoppen van de joystick + 1 extra knop om te selecteren.

Je kan deze schakeling ook bouwen met 5 losse knoppen. Ze zijn allemaal laag actief aangesloten.



➔ **Schrijf nu de volgende code voor de joystick en test uit op een opstelling:**

Vergeet niet van in jouw Arduino IDE programma de "Leonardo" te selecteren i.p.v. de UNO en ook een Leonardo te gebruiken!

```
#include <Keyboard.h>

#define LEFT 3

#define RIGHT 2

#define FORWARD 5

#define BACKWARDS 4

#define FIRE 6


void setup() {


        Keyboard.begin();

        Serial.begin(9600);


        //Joystick and button connections

        pinMode(LEFT, INPUT_PULLUP); //Joystick Left Switch

        pinMode(RIGHT, INPUT_PULLUP); //Joystick Right Switch

        pinMode(FORWARD, INPUT_PULLUP);   //forward

        pinMode(BACKWARDS, INPUT_PULLUP);   //back

        pinMode(FIRE, INPUT_PULLUP); //Fire Button

}
```

In de defines kan je zien dat we de **keyboard.h** library gaan oproepen. Dit geeft ons de mogelijkheid om straks via de USB communicatie onze Leonardo te laten communiceren met de PC, alsof we een keyboard aan het aansturen zijn.

Definieer dan de verschillende knoppen en geef hen een toepassing.

Ik zelf gebruik deze opstelling ook om een **Raspberry Pi arcade game machine** aan te sturen.

Na een **keyboard.begin** is de USB communicatie opgestart.

We gebruiken laag actieve knoppen, dus INPUT_**PULLUP** gebruiken!

In de loop gaan we tenslotte de knoppen afvragen en wanneer er gedrukt wordt dan sturen we de nummer van de toets het keyboard naar de PC.

```
void loop() {
 // Joystick Left = Arrow Left Key
if (digitalRead(LEFT) == LOW) {
        while(digitalRead(LEFT) == LOW){
          delay(10);
        }
        Keyboard.press(216);
         Serial.println("LEFT = 216");
}
else {
         Keyboard.release(216);
    }
}
```

Deze code kan je nu kopiëren en zo kan je zoveel knoppen als je wil toevoegen.

De “216” staat in dit geval voor de pijl die naar links wijst op jouw keyboard.

Eerst doen we een **keyboard.press(216).** Als we de knop hebben losgelaten moeten we dit ook doorgeven. Dit kan door **keyboard.release(216)** te gebruiken.

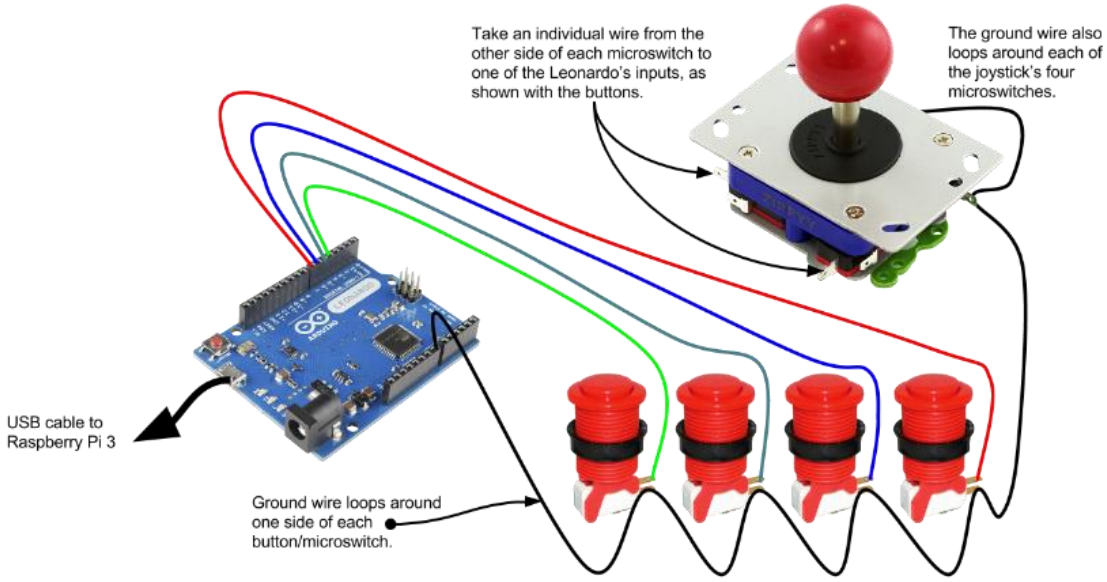
➔ **Uitdaging:**

- Voeg nu zelf in jouw Arduino code de 5 verschillende knoppen toe zodat je de **joystick en ENTER toets kan laten werken.**

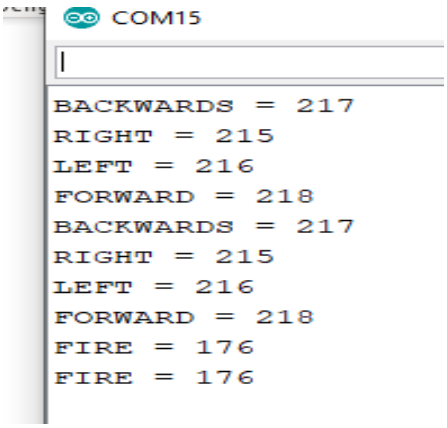
  Test de code uit in een spel. Dit vind je bijvoorbeeld op de scratch website.

  https://scratch.mit.edu/search/projects?q=pacman

- Extra: Maak nu een **complete gamecontroller.** Print een case via de 3D printer en geeft het voldoende functionaliteit om jouw zelf geschreven scratch programma aan te sturen (vb pong, pacman, arcade game via raspberry pi …..)



https://www.element14.com/community/docs/DOC-80946/l/pik3a-the-raspberry-pi-3-ikea-retro-gaming-table



De serial monitor geeft de knop waarde aan.

Deze lijst bevat de meeste keyboard nummers die we nodig hebben:

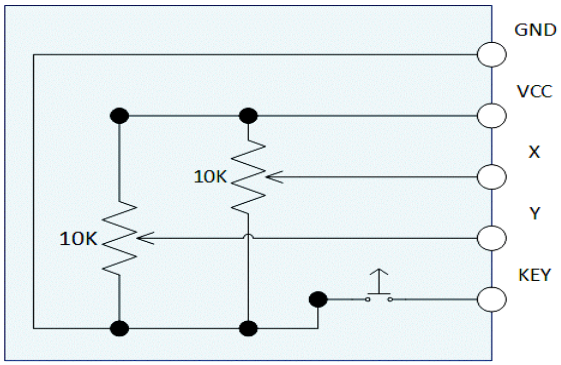
Key	Hexadecimal value	Decimal value
KEY_LEFT_CTRL	0x80	128
KEY_LEFT_SHIFT	0x81	129
KEY_LEFT_ALT	0x82	130
KEY_LEFT_GUI	0x83	131
KEY_RIGHT_CTRL	0x84	132
KEY_RIGHT_SHIFT	0x85	133
KEY_RIGHT_ALT	0x86	134
KEY_RIGHT_GUI	0x87	135
KEY_UP_ARROW	0xDA	218
KEY_DOWN_ARROW	0xD9	217
KEY_LEFT_ARROW	0xD8	216
KEY_RIGHT_ARROW	0xD7	215
KEY_BACKSPACE	0xB2	178
KEY_TAB	0xB3	179
KEY_RETURN	0xB0	176
KEY_ESC	0xB1	177
KEY_INSERT	0xD1	209
KEY_DELETE	0xD4	212
KEY_PAGE_UP	0xD3	211
KEY_PAGE_DOWN	0xD6	214
KEY_HOME	0xD2	210
KEY_END	0xD5	213
KEY_CAPS_LOCK	0xC1	193
KEY_F1	0xC2	194
KEY_F2	0xC3	195
KEY_F3	0xC4	196
KEY_F4	0xC5	197
KEY_F5	0xC6	198
KEY_F6	0xC7	199
KEY_F7	0xC8	200
KEY_F8	0xC9	201
KEY_F9	0xCA	202
KEY_F10	0xCB	203
KEY_F11	0xCC	204
KEY_F12	0xCD	205

4.4 De USB gamecontroller met analoge potentiometers testen

Ook nu gaan we aan de slag met een Leonardo Arduino.

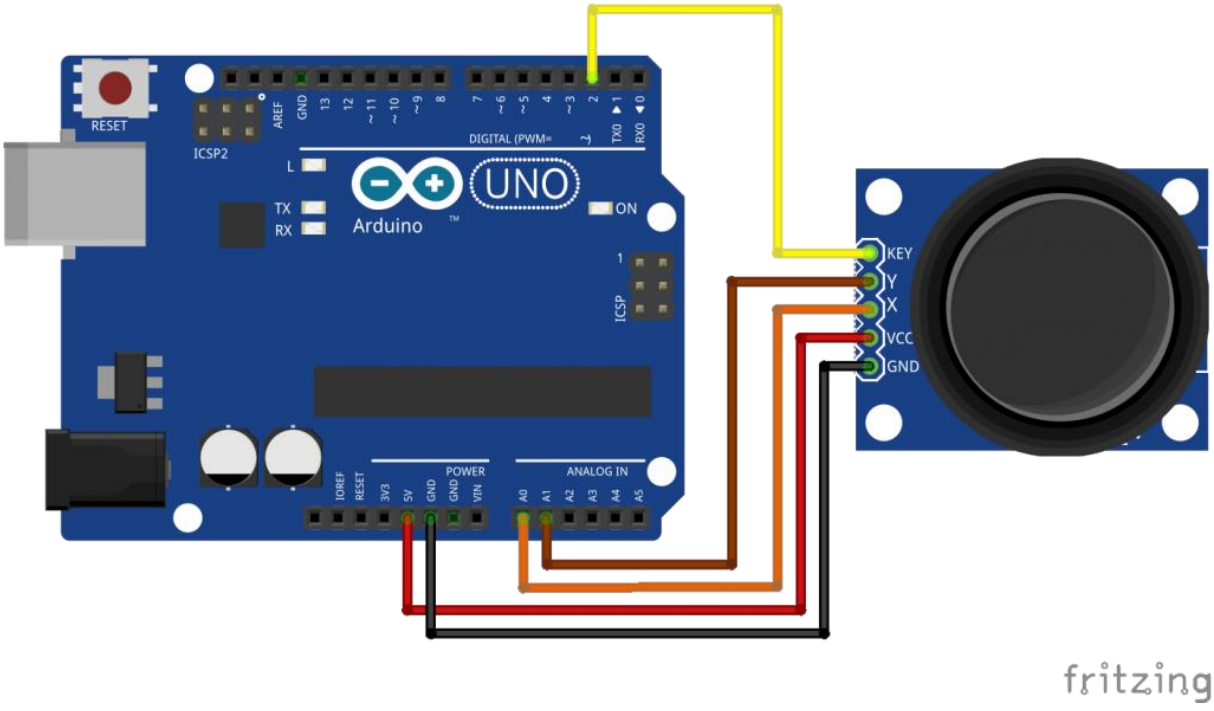
In dit geval vervangen we de digitale knoppen door analoge potentiometers.

Merk op dat een potentiometer een spanningsdeler is (zie gevorderden deel 1) en we dus een spanning gaan terugkrijgen tussen 0V en 5V.



De knop in de joystick is een laag actieve knop!

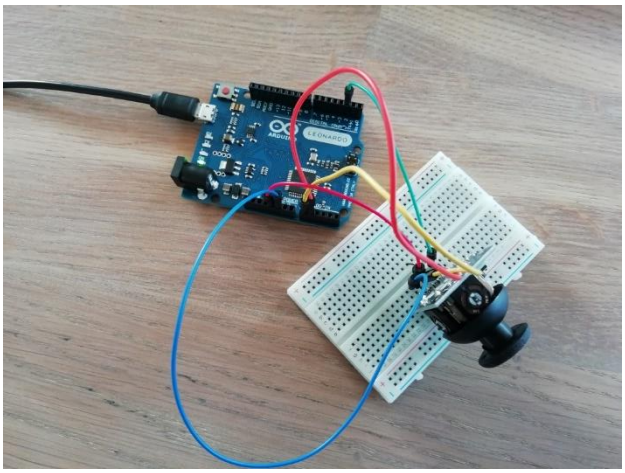
Bouw daarom de volgende schakeling:



We gaan eerst eens testen **welke waardes de potentiometer** teruggeeft en wat de min/max waardes zijn. Schrijf daarom de volgende code en test uit op de serial monitor:

analoge_joystick

```
//Stel de Arduino in als Leonardo

// Arduino pin numbers
const int SW_pin = 2; // digital pin connected to switch output
const int X_pin = 0; // analog pin connected to X output
const int Y_pin = 1; // analog pin connected to Y output

void setup() {
  pinMode(SW_pin, INPUT_PULLUP);//LAAG ACTIEVE KNOP
  Serial.begin(9600);
}

void loop() {
  Serial.print("Switch:  ");
  Serial.print(digitalRead(SW_pin));
  Serial.print("\n");
  Serial.print("X-axis: ");
  Serial.print(analogRead(X_pin));
  Serial.print("\n");
  Serial.print("Y-axis: ");
  Serial.println(analogRead(Y_pin));
  Serial.print("\n\n");
  delay(500);
}
```



We lezen de knop en de 2 analoge waardes binnen. Daarna sturen we deze naar de serial monitor.

Merk op dat “\n” hetzelfde betekent dan “ln” schrijven achter “print”. Dan springen we 1 regel naar beneden en beginnen weer op begin van de regel (net alsof je op ENTER hebt gedrukt).

```
Switch:  1
X-axis: 516
Y-axis: 514
```

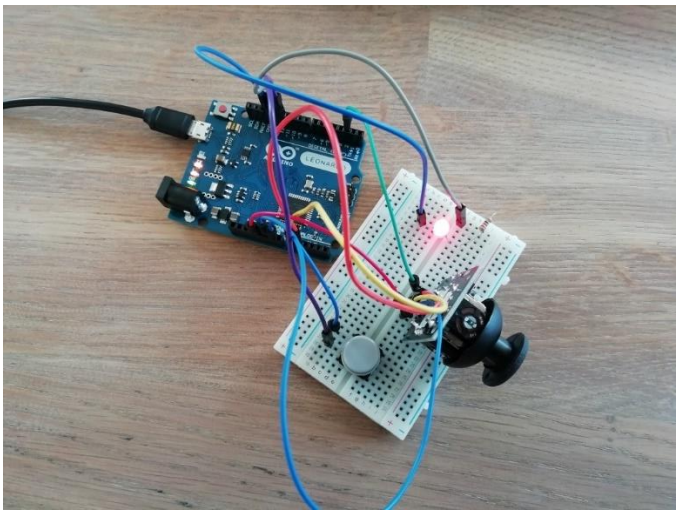
Wanneer je niet op de joystick drukt krijg je een “1” (laag actief) en de potentiometers geven de waarde van 1024 / 2 = 512 ongeveer aan. We gebruiken 1024 omdat dit 2^10 combinaties is van de ADC.

Nu gaan we van de analoge joystick een muis maken:

Pas de bovenste schakeling zo aan dat je een extra laag actieve knop voorziet op pin 12.

Deze knop doet dienst als aan/uit knop voor de muisfunctionaliteit.

Daarbuiten zorg je ook voor een hoog actieve led op pin 13. Deze geeft aan of de muis wel of niet actief is.



usb_analoge_joystick_mouse_v1

```
#include "Mouse.h"

// set pin numbers for switch, joystick axes, and LED:
const int switchPin = 12;      // switch to turn on and off mouse control
const int mouseButton = 2;     // input pin for the mouse pushButton
const int xAxis = A0;          // joystick X axis
const int yAxis = A1;          // joystick Y axis
const int ledPin = 13;          // Mouse control LED

// parameters for reading the joystick:
int range = 12;                // output range of X or Y movement
int responseDelay = 5;         // response delay of the mouse, in ms
int threshold = range / 4;     // resting threshold
int center = range / 2;        // resting position value

bool mouseIsActive = false;    // whether or not to control the mouse
int lastSwitchState = HIGH;        // previous switch state

void setup() {
  pinMode(switchPin, INPUT_PULLUP);  // de laag actieve switch pin
  pinMode(mouseButton,INPUT_PULLUP);//laag actief op joystick
  pinMode(ledPin, OUTPUT);          // the LED pin 13
  // take control of the mouse:
  Mouse.begin();
  Serial.begin(9600);
}
```

Veel nieuw is er niet te vinden in deze code, enkel dat we de **Mouse.h** library gaan gebruiken en **Mouse.begin()** aan het werk zetten in de setup. Nu werkt de Leonardo als een USB muis.

```
void loop() {
  // read the switch: zet de muis aan/uit
  int switchState = digitalRead(switchPin);
  // if it's changed and it's high, toggle the mouse state:
  if (switchState != lastSwitchState) {
    if (switchState == HIGH) {
      mouseIsActive = !mouseIsActive;
      // turn on LED to indicate mouse state:
      digitalWrite(ledPin, mouseIsActive);
      Serial.print("mouseIsActive = ");
      Serial.println(mouseIsActive);
    }
  }
  // save switch state for next comparison:
  lastSwitchState = switchState;

  // read and scale the two axes:
  int xReading = readAxis(A0);
  int yReading = readAxis(A1);
  Serial.print("x-as muis = ");
  Serial.println(xReading);
  Serial.print("y-as muis = ");
  Serial.println(yReading);
```

In de loop vragen we eerst de knop af die de muis actief of niet maakt. Via de flipflop techniek kunnen we dit klaar krijgen. Kan jij dit deel terug vinden in de code?

Daarna lezen we zowel A0 als A1 (de stand van de 2 potmeters van de joystick) in.

We springen hiervoor naar de zelfgeschreven **readAxis(Ax) functie.**

```
  // if the mouse control state is active, move the mouse:
  if (mouseIsActive) {
    Mouse.move(xReading, yReading, 0);
  }

  // read the mouse button (knop van joystick) and click or not click:
  // if the left mouse button is pressed:
  if (digitalRead(mouseButton) == LOW) { //LAAG ACTIEF
    // if the mouse is not pressed, press it:
    if (!Mouse.isPressed(MOUSE_LEFT)) {
      Mouse.press(MOUSE_LEFT);
      Serial.println("linkse muisknop ingedrukt");
    }
  }
  // else the left mouse button is not pressed:
  else {
    // if the mouse is pressed, release it:
    if (Mouse.isPressed(MOUSE_LEFT)) {
      Mouse.release(MOUSE_LEFT);
      Serial.println("linkse muisknop niet ingedrukt");
    }
  }

  delay(responseDelay);
}
```

Als de muis is geactiveerd via de extra hoog actieve knop mag de muis bewegen van de **Mouse.move(x,y,0) functie** van de Mouse.h library.

De laatste waarde zetten we op 0 omdat we het scroll wheel niet willen beïnvloeden.

Ook de muisknop (bij ons nu de laag actieve knop op de joystick) wordt ingelezen. Afhankelijk van de status geeft deze, net als bij de keyboard functies, door of de muisknop is ingedrukt of niet. Zie hiervoor de **functies Mouse.isPressed, Mouse.press en Mouse.release.**

Syntax

```
Mouse.move(xVal, yVal, wheel)
```

Parameters

`xVal`: amount to move along the x-axis. Allowed data types: `signed char`.
`yVal`: amount to move along the y-axis. Allowed data types: `signed char`.
`wheel`: amount to move scroll wheel. Allowed data types: `signed char`.

https://www.arduino.cc/reference/en/language/functions/usb/mouse/mousemove/

Op het einde van de code wordt nog een ResponseDelay variabele gebruikt om de reactiesnelheid van de muis in te stellen.

```
/*
  reads an axis (0 or 1 for x or y) and scales the analog input range to a range
  from 0 to <range>
*/

int readAxis(int thisAxis) {
  // read the analog input:
  int reading = analogRead(thisAxis);

  // map the reading from the analog input range to the output range:
  reading = map(reading, 0, 1023, 0, range);

  // if the output reading is outside from the rest position threshold, use it:
  int distance = reading - center;

  if (abs(distance) < threshold) {
    distance = 0;
  }

  // return the distance for this axis:
  return distance;
}
```

Tot slot wordt de functie van de readAxis() beschreven. Via “thisAxis” komt er een pinwaarde A0 of A1 binnen in de functie.

Na het inlezen van de analoge waarde, die ligt tussen 0 en 1024, wordt deze met de map() functie omgezet naar de range dewelke de muis gebruikt. De “range” variabele was op het begin van het programma al ingesteld op 12. Dus de analoge 0 komt overeen met de nul van de muis. De analoge 1023 komt nu overeen met de 12.

Nu moeten we alleen nog te weten komen of de muis naar links of rechts bewoog.

Als center hebben we 12 / 2 = 6 gekozen.

Stel dat de muis = joystick **helemaal naar links** bewoog:

Reading = 318 (“3” na mapping)

distance = reading – center = 3 – 6 = -3

absolute waarde van -3 = 3

3 < range / 4 => 3 < 3 => neen

Dan is het antwoord **“-3”** dat teruggestuurd wordt naar de muis (absolute waarde is dan niet toegepast).

Stel dat de muis = joystick **niet** bewoog:

Reading = 512 -> “6” na mapping (in het midden)

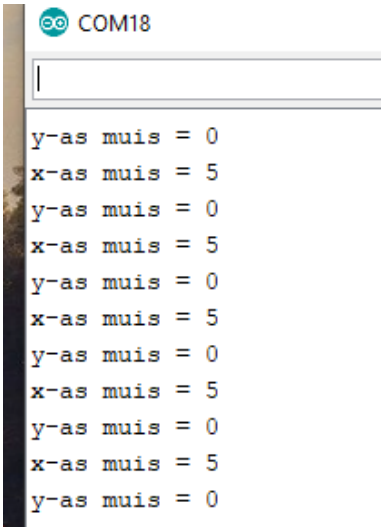
distance = reading – center = 6 – 6 = 0

absolute waarde van 0 = 0

0 < range / 4 => 0 < 3 => juist

Dan is het antwoord **“0”** dat teruggestuurd wordt naar de muis

(absolute waarde niet toegepast)



Stel dat de muis = joystick **helemaal naar rechts** bewoog:

Reading = 1020 -> “11” na mapping

distance = reading – center = 11 – 6 = 5

absolute waarde van 5 = 5

5 < range / 4 => 5 < 3 => fout

Dan is het antwoord **“5”** dat teruggestuurd wordt naar de muis.

(absolute waarde niet toegepast)

```
reading = 515
reading na mapping = 6
distance = 0
distance na threshold test = 0
x-as muis = -6
y-as muis = 0
```

Dus afhankelijk van de beweging krijgen we een negatieve waarde (naar links), een nul (in het midden) of een positieve waarde (naar rechts). Dat is voldoende om aan de PC door te geven hoe we bewegen.

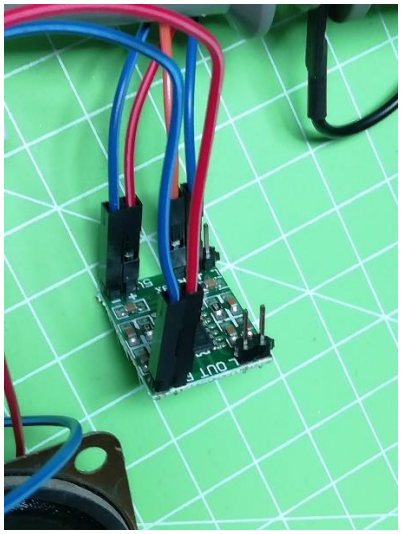
Test zelf de berekening uit door met de serial monitor uit te zoeken wat de code nu allemaal berekent.

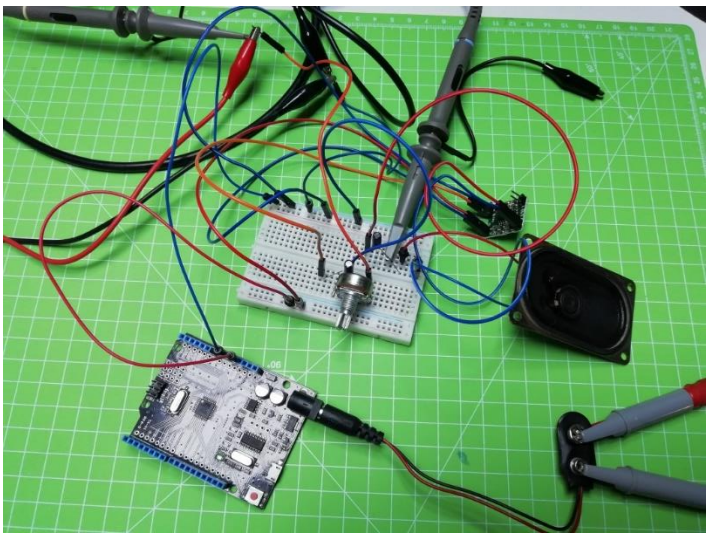
➔ **Uitdaging:**

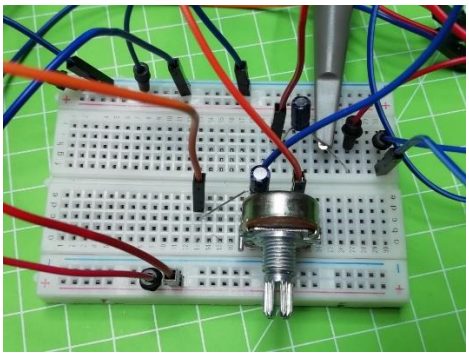
Maak nu **een 2de joystick** aan en zorg dat ook hier de bewegingen zichtbaar kunnen worden op het scherm. Zorg dat je beide joysticks met een knop kan activeren/deactiveren. Want er is maar 1 muispijltje 😊

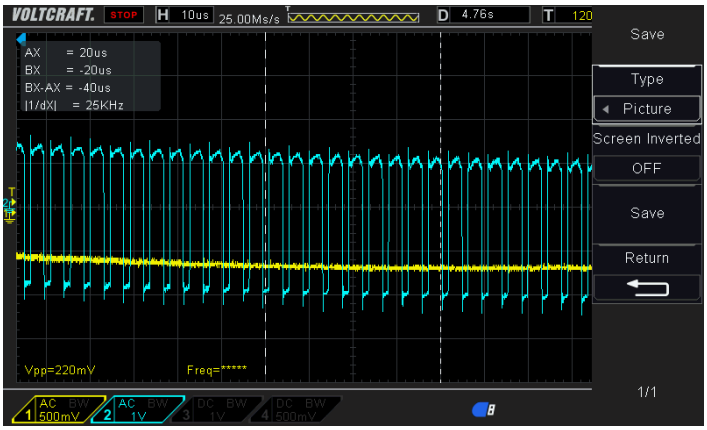
5 SD card leren inlezen en een wav player maken (bij extra tijd)

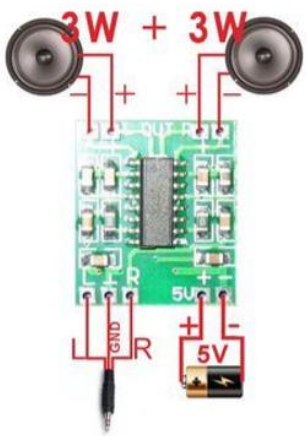
5.1 De klasse-D versterker laten werken

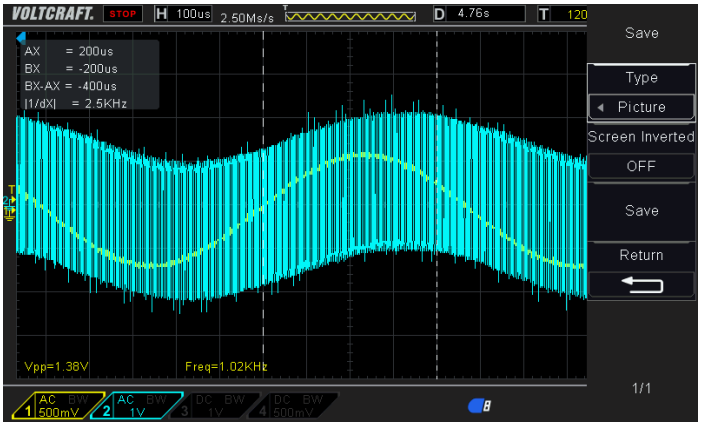












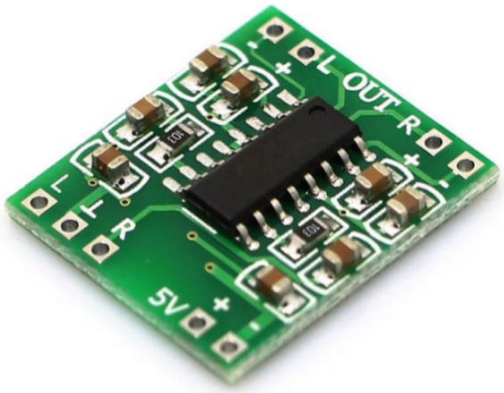
Allereerst moeten we de versterker bouwen om het **audiogeluid te versterken** van de Arduino.

Hoe doen we dit.

We kunnen met een handvol componenten rond een opamp of transistor een versterker bouwen, maar we weten ook dat versterkers bouwen op een breadboard veel ruis en fouten met zich mee kan brengen.

Dus kiezen we voor een **klasse-D versterker** die reeds op een PCB van 2 op 2cm is gesoldeerd.

De aansluitingen zijn heel eenvoudig van de **PAM8403** CLASS D DC stereo versterker.



https://www.instructables.com/id/PAM8403-6W-STEREO-AMPLIFIER-TUTORIAL/

Soldeer pinnen op het versterkerbordje. Zorg dat de headerpinnen omhoog steken aan de chip zijde.

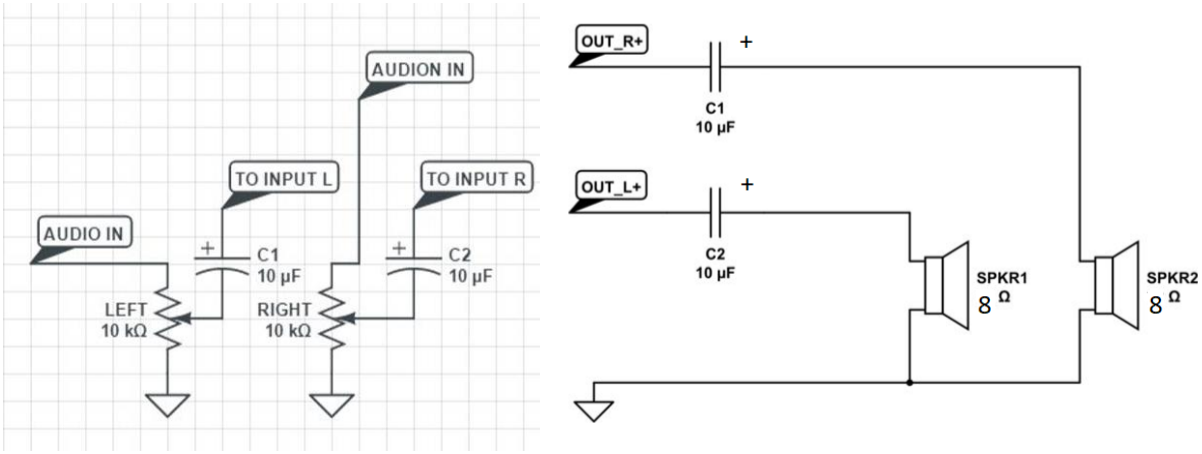
Voorzie dan draden op de volgende aansluitingen.

+5V	+5V van de arduino
GND	GND van de arduino
L	Audio input links (pin 9 van de arduino)
GND	Audio massa van de arduino
L OUT	8 ohm 3W speaker uitgang versterker (+)
GND OUT	Speaker uitgang (massa)

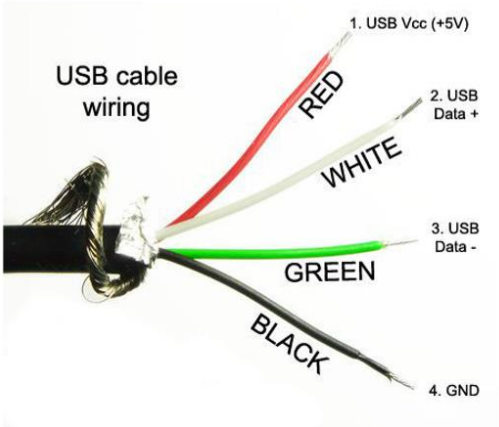
Test de versterker door de +5V voorlopig nog aan te sluiten via een externe +5V voeding met stroombeperking (100mA is in principe voldoende).

Voordat je de ingang aanlegt aan een **1KHz sinus signaal van 1Vpp** moet je een condensator van **10uF** in serie voorzien tussen de functiegenerator en de ingang van de versterker. Er is ook een **potmeter van 10K** voorzien om het volume te kunnen regelen.

Tussen de uitgang van de versterker en de box gaan we eveneens **10uF** aansluiten. Let op de polariteit (zie schema)! Dit schema is voor de stereo versie, maar we gebruiken hem hier enkel **mono.**



Als je ruis wil voorkomen of invloeden van buiten af dan moet je best met **coax kabels** werken of **afgeschermde draden**. Een USB kabel zou je ook kunnen stuk maken en enkel gebruik maken van de voedingsdraden. De afscherming zou je als massa kunnen gebruiken en de rode als signaal.



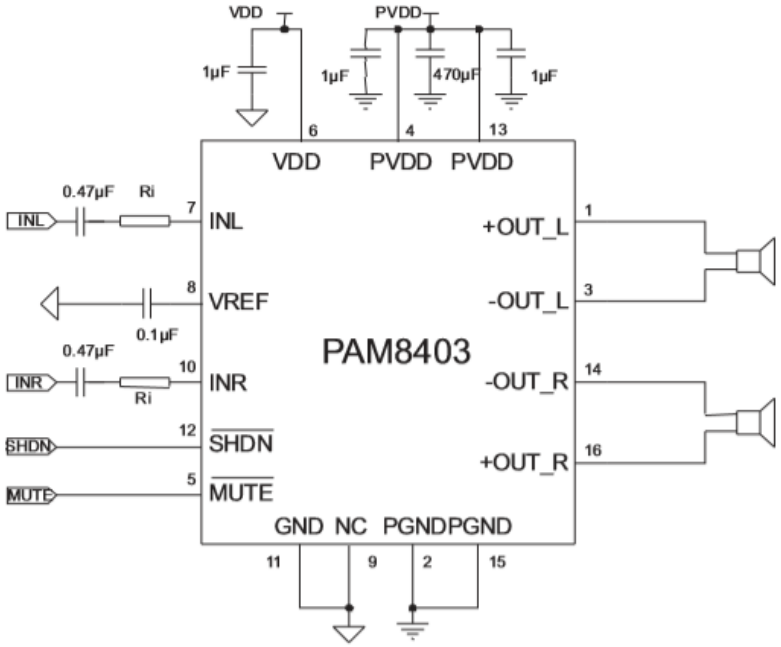
Uitdaging:

Meet zelf met een oscilloscope de ingang en de uitgang van de klasse-D versterker na.

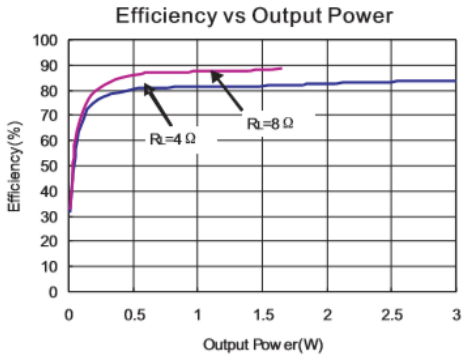
Stuur er een 1KHz frequentie van 1Vpp aan in en met de uitgang. Regel met de potmeter het signaal aan de ingang zo af dat je aan de uitgang nog een mooi signaal bekomt.

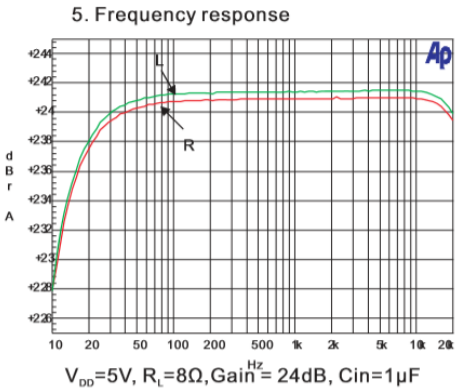
Je kan ook met een sweep generator de versterking van het audio gebied (20Hz tot 20KHz) testen van de versterker. Kijk op de uitgang wat er gebeurt met de amplitude.

Nog wat meer info over de PAM8403 versterker chip:



Merk op dat de randcomponenten reeds op het versterkerprintje zijn voorzien.





Merk op dat de versterker met 4 en 8 ohm luidsprekers kan werken. Bij de 4 ohm speakers kan je wel tot aan de 3W uitgang geraken. Bij de 8 ohm speaker raak je maar tot de helft.

Je kan in de rechts tabel zien dat dit een versterker is voor audio. We mogen een versterking van 24dB verwachten.

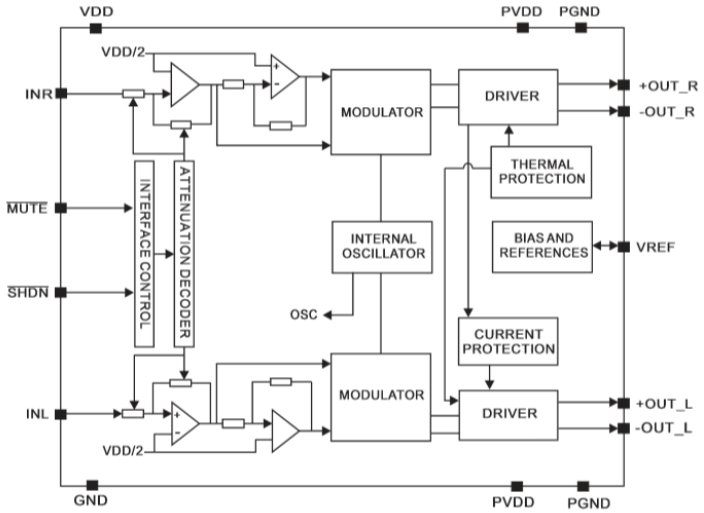
Wat is dan de spanningsversterking van deze versterker?

24dB = 20 log Uuit / Uin

$$10^{24/20} = 10^{\log x}$$

$$x = 10^{1.2} = 15.84$$

Wat zit er in een klasse-D versterker en hoe werkt het?



- *Klasse D eindversterker*
  In deze eindversterker wordt aan het ingangssignaal een heel hoogfrequent modulatiesignaal van rond de 100 kHz toegevoegd (Fig. 4).

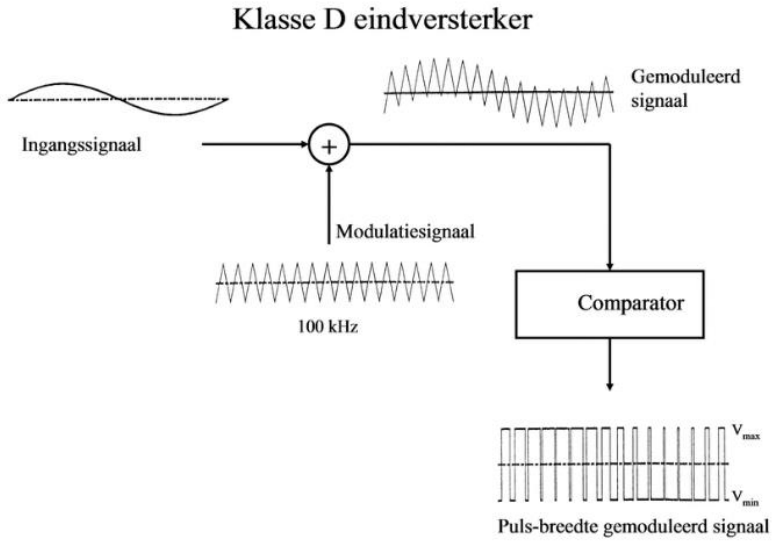


Fig.4. Voorbeeld van de werking van een klasse D versterker.

https://audiologieboek.nl/content/9-2-42-analoge-signaalbewerking-in-luchtgeleidingshoorstellen/

Dit betekent dat de nuldoorgangen van het hoogfrequente gemoduleerde signaal verschoven komen te liggen, afhankelijk van de grootte van het ingangssignaal. Door vervolgens het gemoduleerde signaalsignaal te clippen gaat dat bestaan uit blokpulsjes die breder zijn op plaatsen waar het ingangssignaal groter was (pulsbreedte modulatie). De energie die de telefoon aangeboden krijgt is het gemiddelde van de blokpulsjes in een verschuivend tijdvenster.

Als *voordelen* kunnen genoemd worden het nóg lagere stroomverbruik dan in de klasse B versterker, de lage vervorming bij normale ingangsniveaus en de toepasbaarheid in kleine hoortoestellen (CIC). *Nadelen* zijn de relatief hogere kosten, de hoge vervorming bij hogere ingangsniveau's en daarmee dus ook het niet toepasbaar zijn in hoortoestellen die een groot uitgangsvermogen moeten leveren.

Deze klasse D eindversterker bevat geen transistorschakeling maar CMOS-schakelaars, die worden bestuurd door het pulsbreedte gemoduleerde signaal. Daardoor wordt de batterijspanning met een frequentie van rond de 100 kHz afwisselend in positieve of negatieve richting aan de telefoonspoel aangeboden. Het analoge uitgangssignaal wordt in de vorm van elektrische pulsen naar de telefoon gestuurd. Door de zelfinductie van de telefoonspoel en de massatraagheid van het telefoonmembraan wordt de hoge frequentie niet weergegeven.

Eerst in mootjes hakken zorgt voor koelte

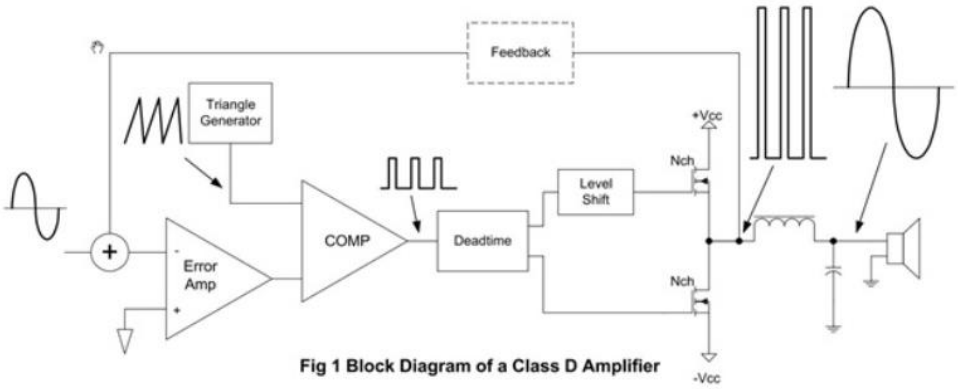
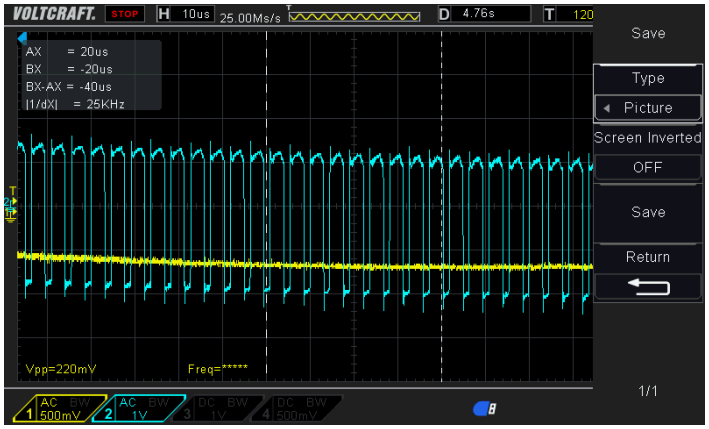


Fig 1 Block Diagram of a Class D Amplifier

Waarom wordt het signaal in mootjes gehakt? De gedachte hier achter is dat bij een 'normale' versterkerschakeling de uitgangstransistoren continu wisselen in geleiding, afhankelijk van de aansturing. Als een vermogenstransistor half in geleiding staat wordt de overige helft in warmte omgezet. Vandaar de dikke koelprofielen die je op versterkers ziet. Wist je dat als je een solid state versterker statisch op eenderde van zijn maximale uitgangsvermogen instelt, je binnen de kortste tijd de handen niet meer aan de koelprofielen kunt lijden. En al die verstookte hitte komt niet als muziek bij je luidsprekers. Nu is muziek niet statisch maar dynamisch, dus valt het bij een klasse AB ingestelde eindtrap qua warmteontwikkeling wel mee, maar je begrijpt dat het rendement van de conventionele versterkertechniek niet optimaal is.

https://audio-creative.nl/recensie-2/klasse-d-versterkers/2/

Hoe ziet ons klasse-D versterker signaal er aan de uitgang uit?



Je kan mooi de modulatie frequentie terugvinden in het blauwe signaal.

Het gele signaal is de 1KHz die we aan de ingang hebben aangesloten.

5.2 De WAV player aan het werk zetten met een Arduino

Nu we de klasse-D versterker hebben uitgetest kunnen we aan de WAV player beginnen bouwen.

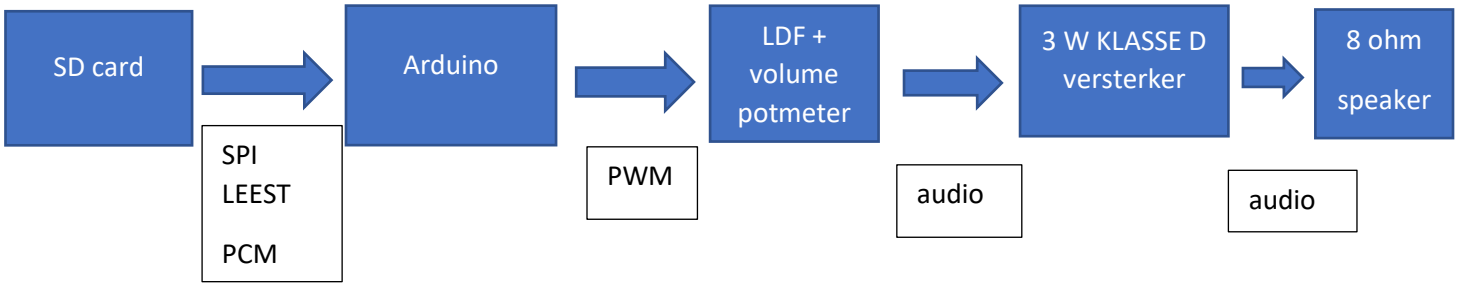
De schakeling is opgebouwd uit de volgende delen:

We nemen een micro SD-card module en gaan deze aansluiten met 6 SPI draden op de Arduino.

De Arduino leest de PCM code uit de SD-card en zet het digitaal signaal om in een PWM uitgang.

Daarna gaan we de PWM uitgang via een LDF (laag doorlaat filter) aansluiten op onze versterker.

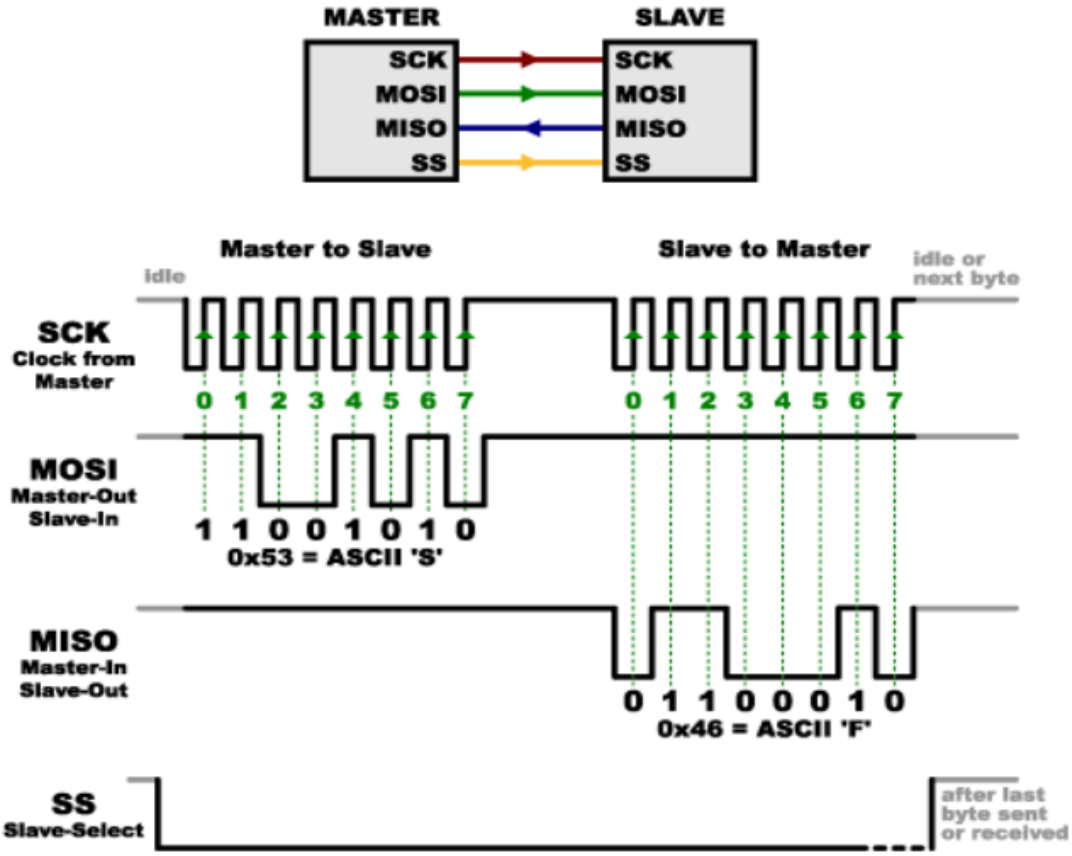
Het audio signaal wordt extra en efficiënt versterkt en weergegeven in de speaker.



Je kan zien dat in deze schakeling verschillende protocollen zijn gebruikt. Deze worden verder even toegelicht.

De **SPI of Serial Peripheral Interface** is het communicatie protocol tussen de SD-card en de Arduino.

We zijn deze bus al eerder tegen gekomen bij het programmeren van de bootloader in de zelf gesoldeerde UNO.



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

De bovenstaande tekening geeft mooi weer hoe de synchrone SPI bus werkt.

De **SCK of clock** houdt het ritme van de bits tussen master (microcontroller) en slave (onze SD card) in de gaten. Er wordt op de opgaande klok ingelezen.

Als je data uit de SD card wil halen moet je **MISO lijn aansturen (Master In Slave Out).**

Als je data op de SD-card wil schrijven moet je de **MOSI lijn gebruiken (Master Out Slave In).**

Je kan enkel met het SPI device werken als je de **SS laag maakt (Slave Select).**

Handig is dat we bij dit protocol geen start en stopbits moeten gebruiken. Alles wordt via de synchrone klok geregeld.

De snelheid van de data is dus afhankelijk van de snelheid van de apparaten die er aanhangen.

De hardware achter de SPI bus is een gewone **schuifregister. Veel eenvoudiger dan de UART hardware met zijn extra seriële protocol regels**

Het SPI protocol wordt in de Arduino netjes afgehandeld via de **SPI.h library.**

De **PCM** staat voor: https://nl.wikipedia.org/wiki/Pulscodemodulatie

Pulscodemodulatie (**PCM**) is een digitale voorstelling van een analoog signaal, waarbij de signaalwaarde op regelmatige tijdstippen bemonsterd wordt, en gekwantiseerd tot een serie waarden in een digitale (om precies te zijn binaire) code. PCM wordt gebruikt in digitale telefoonsystemen en is ook de standaardopslagvorm van digitaal geluid in computers, in verschillende bestandsformaten en op cd.

Voor spraak is de kwantisering gedefinieerd in de ITU-T-aanbeveling G.711, waarin de logaritmische kwantisatie van het spraaksignaal is vastgelegd; hiervoor bestaan twee varianten: 'A-law' en 'μ-law' (=mu-law).

Een veel gebruikte, afgeleide vorm is de differentiële pulscodemodulatie (DPCM). Bij deze techniek worden niet de monsters zelf verzonden, maar de verschillen tussen het huidige en het vorige monster. Het idee hierachter is dat het verschil tussen twee opeenvolgende monsters veel kleiner is dan de grootte van het monster zelf. Doordat er minder data te verzenden zijn, is ook de benodigde bandbreedte kleiner.

Inleiding [bewerken | brontekst bewerken]

Pulscodemodulatie is het technologische hart van de communicatie in de huidige digitale wereld. Het is een proces waarin analoge signalen geconverteerd worden naar een digitale vorm. Het analoge signaal (zoals spraak van een telefoongesprek) wordt eerst geconverteerd naar een pulsamplitudemodulatiesignaal (PAM) door het signaal te bemonsteren. Vervolgens kan men dit PAM-signaal kwantiseren en coderen, zodat een digitaal PCM-signaal verkregen wordt.

Filteren [bewerken | brontekst bewerken]

De eerste stap om een analoog signaal om te zetten naar een digitaal is het wegfilteren van hogere frequenties uit het signaal. Dit maakt het achteraf makkelijker om het signaal om te zetten. De belangrijkste frequenties bij spraak liggen ergens tussen de 200 of 300 Hz en 2700 of 2800 Hz. Zo verkrijgt men een bandbreedte van ongeveer 3000 Hz voor standaardspraakcommunicatie. Hiervoor zijn geen precieze (vaak ook erg dure) filters nodig. Een bandbreedte van 4000 Hz wordt hardwarematig verkregen. Deze limiterende bandfilter wordt gebruikt om aliasing te voorkomen (anti-aliasing). Aliasing vindt plaats wanneer het inkomende analoge spraaksignaal wordt onderbemonsterd, dus indien niet wordt voldaan aan het criterium van Nyquist dat luidt: Fs ≥ 2(BW). Als de bemonsteringsfrequentie kleiner is dan twee maal de hoogste frequentie in het inkomende analoge signaal, ontstaat een overlapping tussen het frequentiespectrum van het inkomende analoge signaal en een gespiegelde versie daarvan. De laagdoorlaatuitgangsfilter die wordt gebruikt om het inkomende signaal terug te reconstrueren, is niet in staat om deze overlapping ongedaan te maken. Hierdoor wordt een nieuw signaal opgebouwd dat afwijkt van het originele ingangssignaal. Het creëren van een vals signaal door overlapping wordt aliasing genoemd.

Bemonsteren [bewerken | brontekst bewerken]

De tweede stap om een analoog signaal om te zetten naar een digitaal is het bemonsteren van het gefilterde ingangssignaal. Bemonsteren houdt in dat de momentele waarde van een analoog signaal wordt gemeten op tijdstippen die een constant tijdsinterval van elkaar verschillen. Dit wordt ook wel pulsamplitudemodulatie (PAM) genoemd. Deze stap gebruikt het originele analoge signaal om de amplitude van een pulstrein te moduleren, die zelf een constante amplitude en frequentie heeft.

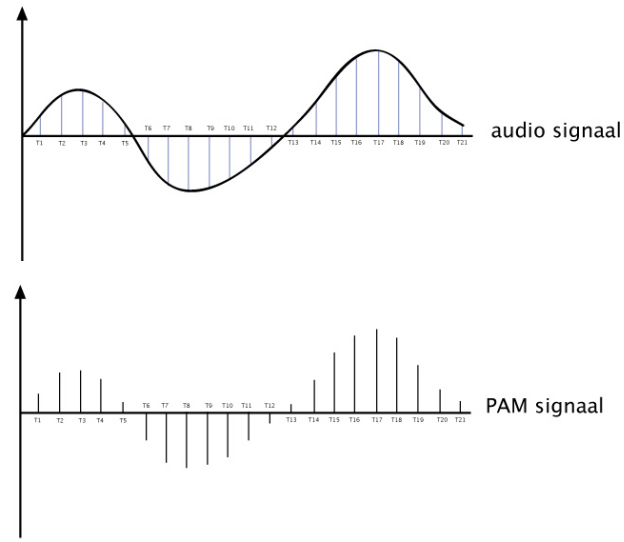
Bemonsteren houdt in dat de momentele waarde van een analoog signaal met een periodiek tijdsinterval wordt gemeten. Een bemonsterd signaal bevat slechts dan alle in het signaal aanwezige informatie, als de bemonsterfrequentie ten minste tweemaal zo hoog is als de hoogste frequentie die voorkomt in het signaal. Dit volgt uit het bemonsteringstheorema van Shannon en Nyquist. Bij toepassen van een dergelijke bemonsterfrequentie zal het mogelijk zijn het oorspronkelijke signaal te reconstrueren op de plaats van bestemming. Het Nyquistcriterium zegt het volgende:

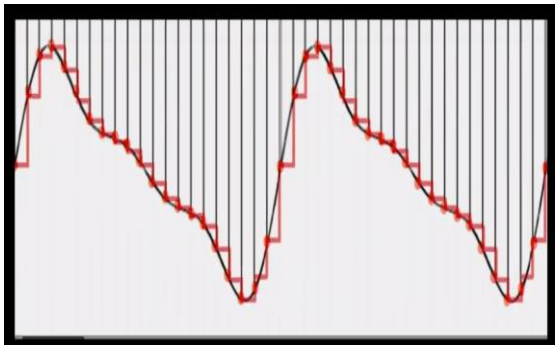
$F_s \geq 2 \times BW$

F_s = bemonsterfrequentie

BW = bandbreedte van het originele analoge spraaksignaal

Door filtering wordt de bandbreedte bij telefonie aan de zendzijde beperkt tot frequenties tussen 300 en 3400 Hz. Hieruit volgt dat een bemonsteringsfrequentie van 8000 Hz (om de 125 μs) geschikt is.

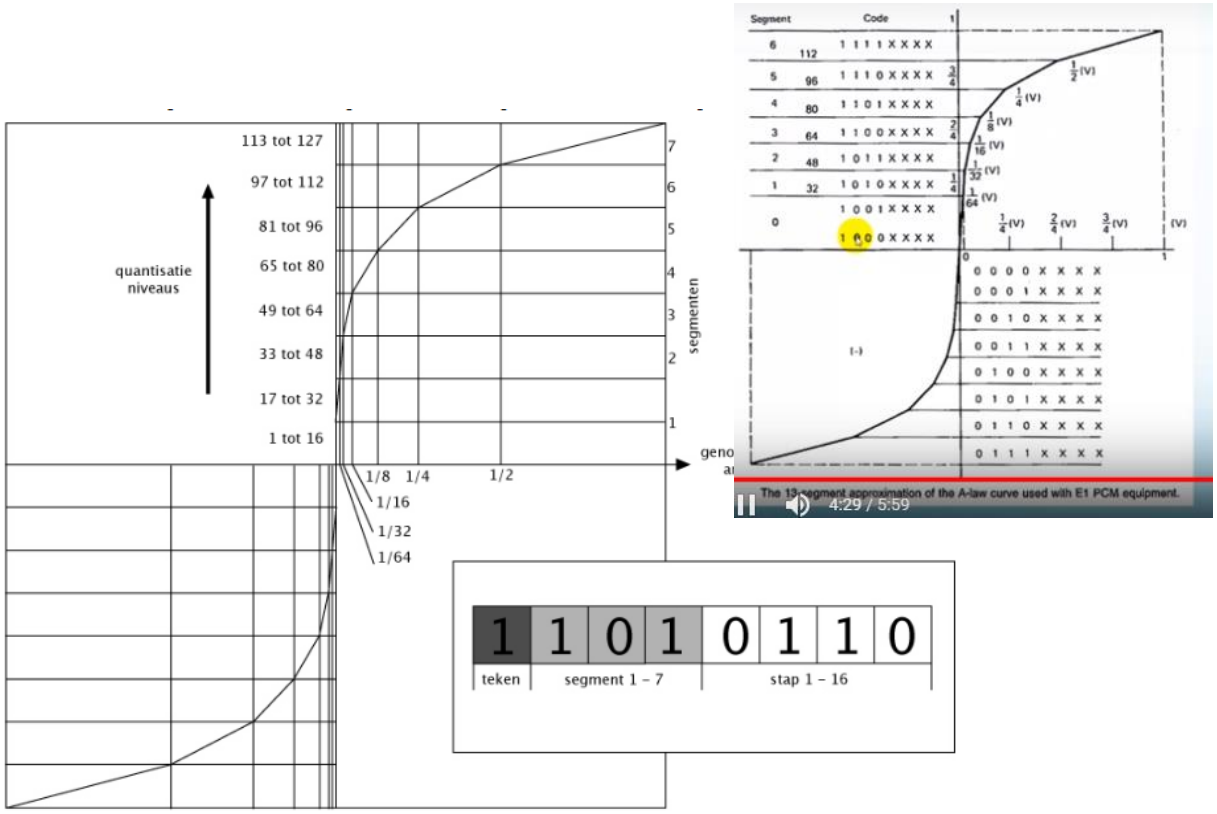




Bemonstering van het analoge signaal

Merk op dat wij reeds een digitale versie hebben van ons analoog signaal. De PCM code, geconverteerd uit het MP3 signaal bevat dus binaire codes. Elke code staat voor een monster van het originele analoge signaal. We hebben 16000 samples. De bandbreedte is max 8000Hz (Fs >= 2 x BW).

Kwantisatie van het analoge naar digitale signaal bij een 8 bits binair getal:



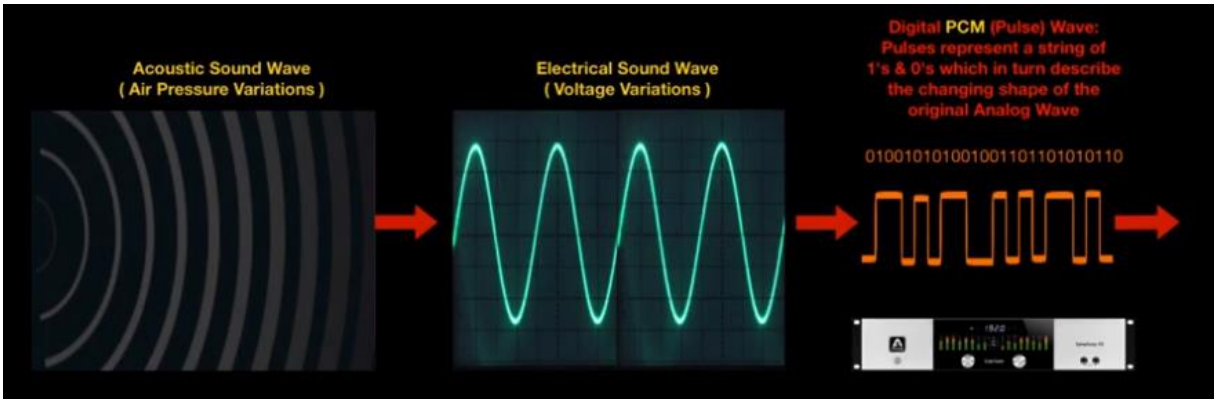
Hoe is het analoog signaal omgezet naar een digitaal 8 bits getal?

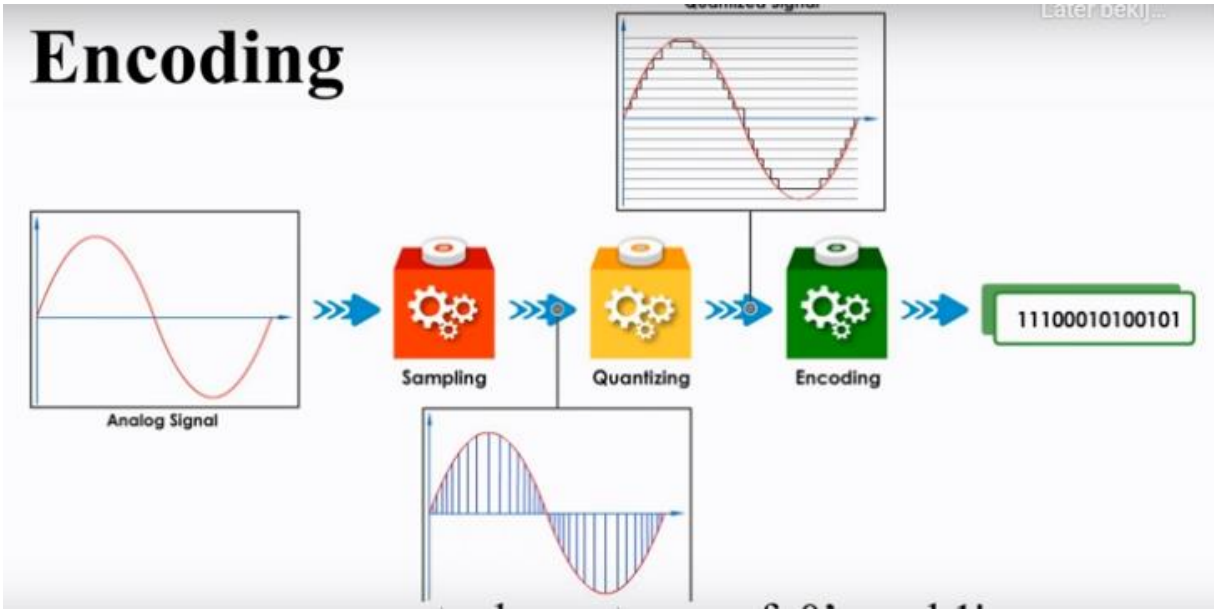
De max waarde is 128. Dit kan + of – zijn, dus we krijgen 256 niveaus.

In bovenstaande tekening zie je hoe het 8-bits getal is opgebouwd:

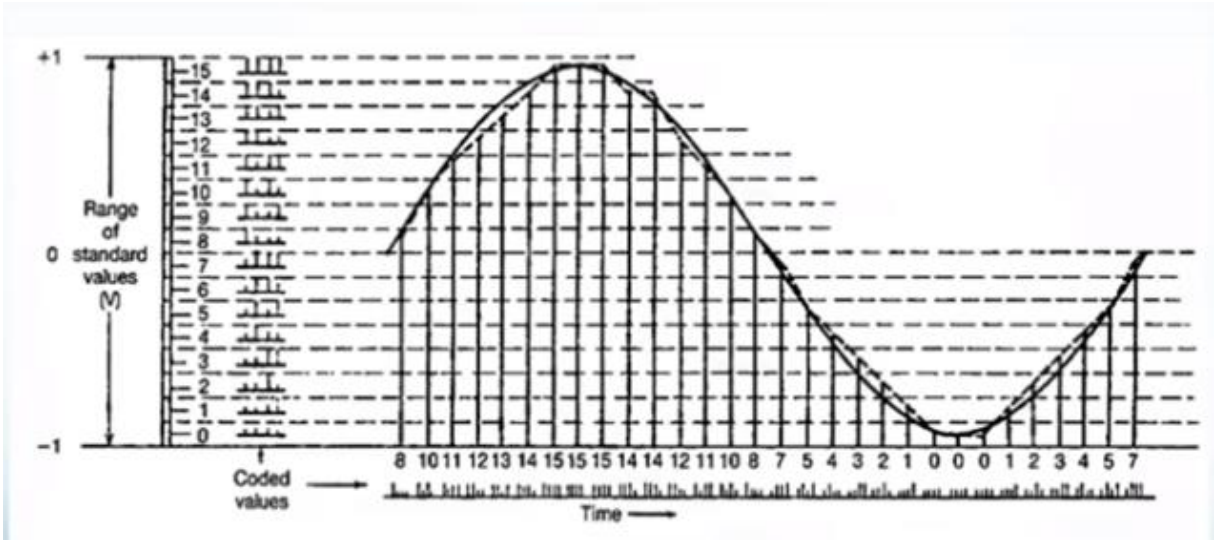
Eerst de tekenbit, dan 3 bits voor de aanduiding van het segment en tenslotte 4 bits om 1 van de 16

Stappen binnen een segment aan te duiden.



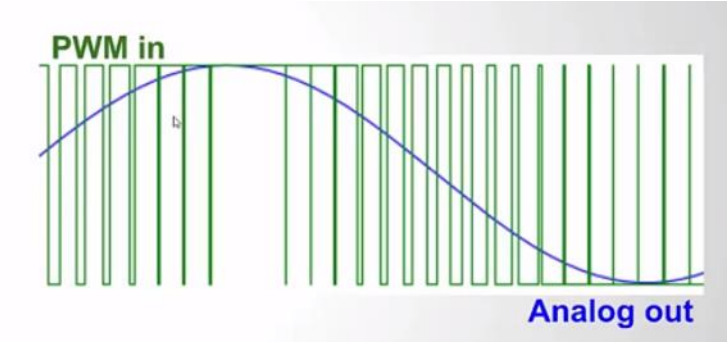


Mooi overzicht van hoe analoge signalen omgezet worden in een digitale versie



Nog meer info over de kwantisatie

Uiteindelijk leest de Arduino het **binaire PCM getal** uit de SD card voor een bepaald monster en zal dit terug omzetten in een puls met een bepaalde lengte. We sturen m.a.w. **PWM** uit. Vandaar dat we ook op pin 9 zitten (met een PWM uitgang!)



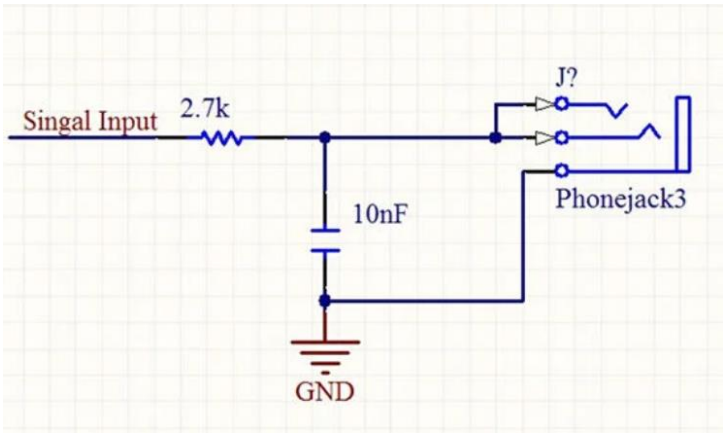
Hoe breder de puls, hoe hoger de amplitude !

Je kan het digitale signaal omzetten naar een analoog audio signaal (DAC) op verschillende manieren:

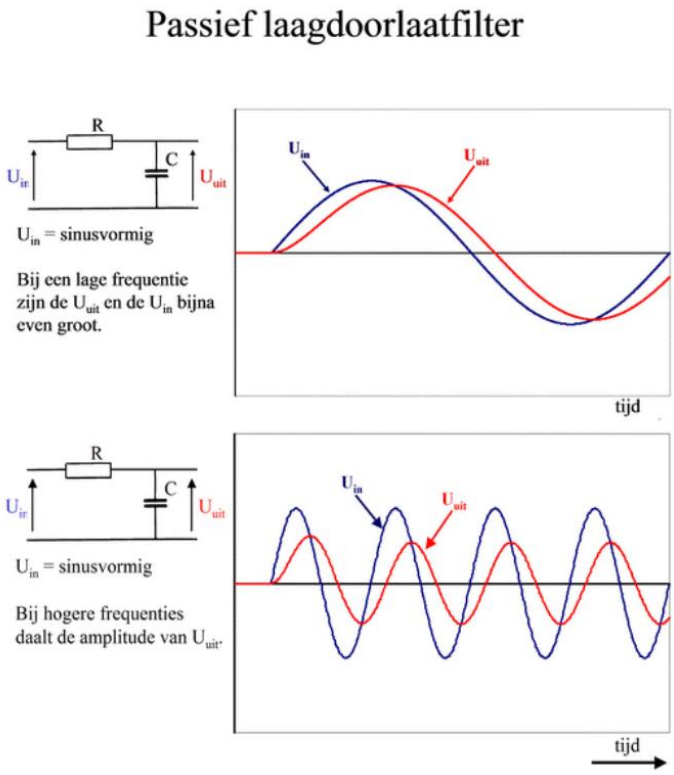
- Een **LDF filter** gebruiken met een RC of LC keten waar je PWM naar toe stuurt (zo gaan wij het doen)
- Een **R2R** weerstanden netwerk als DAC
  (zie https://www.youtube.com/watch?v=Y2OPnrgb0pY)
- Een speciale DAC chip aan de uitgang van de Arduino voorzien
  (zie https://learn.adafruit.com/mcp4725-12-bit-dac-tutorial/overview)

De LDF op de uitgang van de Arduino.

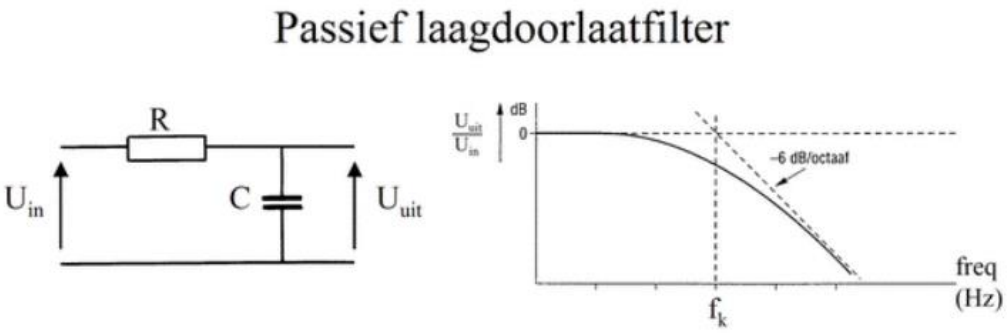
Voeg tussen de uitgang van de Arduino en de ingang van de klasse D versterker een **LDF** (laag doorlaat filter) toe.



Wat is het effect van de passieve laag doorlaatfilter (LDF) op de uitgang van de Arduino?



Het is duidelijk dat de lage frequenties worden doorgelaten terwijl bij hoge frequenties de amplitude kleiner wordt en deze dus uit het signaal gefilterd worden.



Fk = 1 / 2 pi R C

Fk = 1 / 2 x 3.14 x 2k7 x 10n = 5.9 KHz

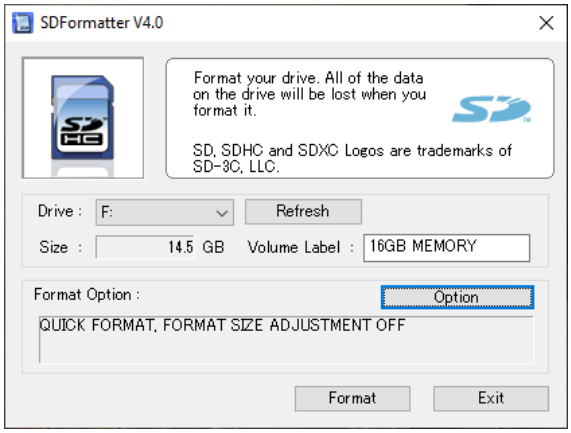
Merk op dat we tegen 16KHz de PWM signalen uit de Arduino sturen.

Dus deze worden weg gefilterd aan de uitgang. Enkel de audio blijft over.

Als we dit niet wegfilteren horen we deze gemoduleerde pulsen ook in de audio, en dat is niet de bedoeling.

Hoe een MP3 bestand omzetten **op een SD card in PCM formaat**?

1. Eerst en vooral **het SD kaartje** even leegmaken en er het juiste bestandstype van maken. Dit kan je doen met het programma "**SDFormatter**". Zorg dat de SD card in een **FAT32** bestandsstype is geformatteerd. Wees voorzichtig met het gebruiken van deze software, kijk eerst na of je de data die nog op de SD card stond nog kan gebruiken!



Selecteer Format nadat je de juiste drive hebt gekozen.

2. Zoek een **MP3 bestand** naar keuze
3. Open nu online de MP3 naar **WAV converter** software

   https://audio.online-convert.com/convert/mp3-to-wav

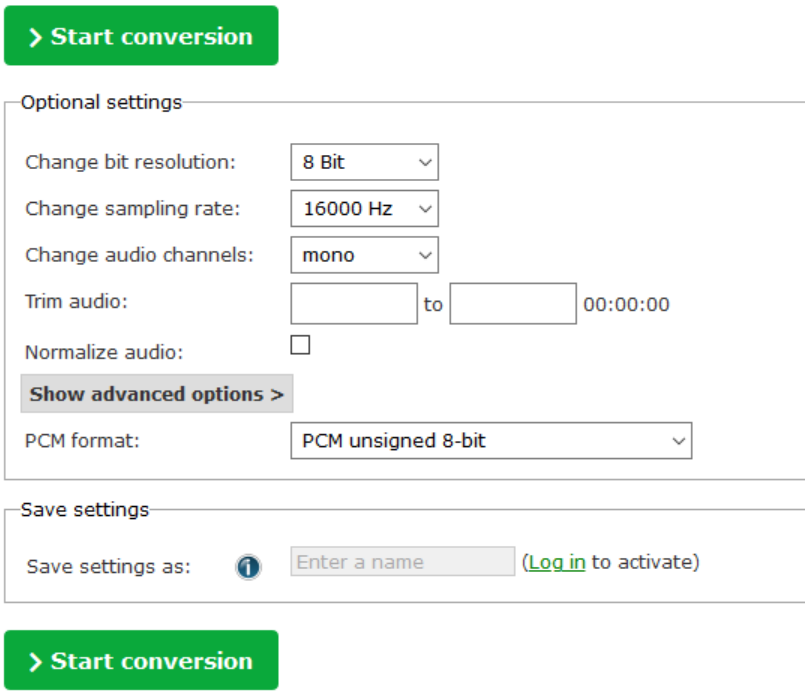
   Stel de volgende settings in en converteer de file naar een WAV file.

   bit resolution: 8 bit

   sampling rate: 16000Hz

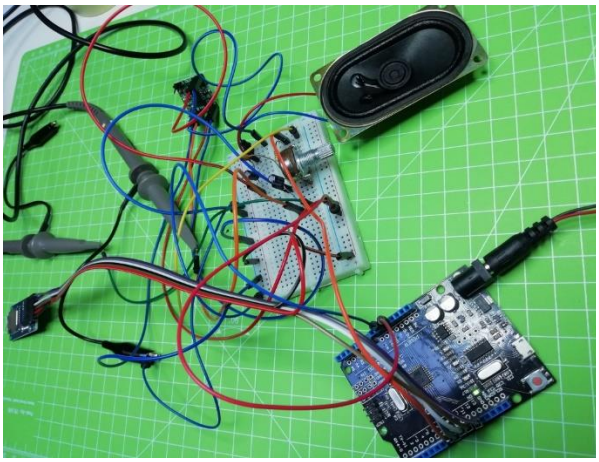
   audio : mono

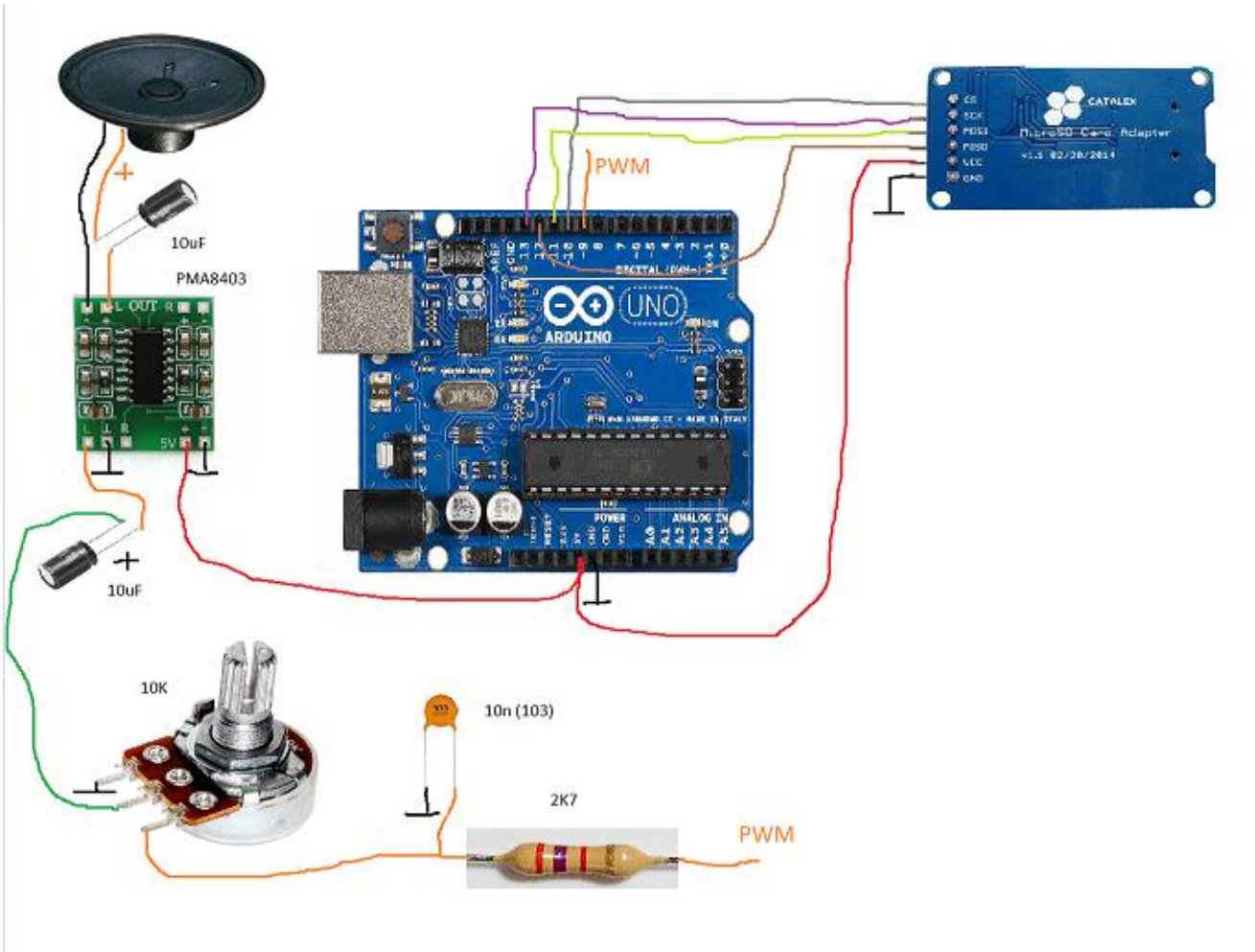
   PCM format: PCM unsigned 8-bit
   Sleep de MP3 file in de groene kader en druk op “start conversion”.



4. Kopieer nu deze file op het SD kaartje en geef het **de naam “test”** en de extensie blijft “*.wav” . Aan de extensie moet je dus niets meer doen.
5. Verwijder de SD kaart veilig en plaats deze in het **SPI card lezer printje.**

Bouw nu de volgende Arduino wav player schakeling:





De ganse wav player schakeling

SD card connecties

SD card	**Arduino**
GND	GND
VCC	+5V
MISO	12
MOSI	11
SCK	13
SC of SS	10

Nu de schakeling klaar is kunnen de we de code gaan compileren en downloaden in de Arduino UNO.

Eerst moet er wel nog de **TMRpcm.zip** library toegevoegd worden in Arduino (zie USB stick).

Ga naar sketch -> include library -> add zip library en verwijs naar de zip file op de USB stick.

SD_card_wave_player

```
#include "SD.h"
#define SD_ChipSelectPin 10
#include "TMRpcm.h"
#include "SPI.h"

TMRpcm tmrpcm;

void setup()
{

tmrpcm.speakerPin=9;
Serial.begin(9600);

Serial.println("start player");

if(!SD.begin(SD_ChipSelectPin))
{
  Serial.println("SD fail");
  return;
}
tmrpcm.setVolume(3);
tmrpcm.play("test.wav");
Serial.print("start playing wav file");

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Laad nu de code in van de WAV player.

De SS pin zit op pin 10. We kunnen deze aanpassen afhankelijk van hoeveel SPI toestellen je tegelijk wil sturen. Als deze laag wordt kan er gecommuniceerd worden met de SD card module.

De TMRpcm library zorgt ervoor dat we heel eenvoudig de SD card kunnen aansturen.

SD.begin() start uiteraard de SD communicatie op. Als de SD kaart goed is geopend krijgen we een “1” terug. Dan wordt er geen “SD fail” afgedrukt op de monitor.

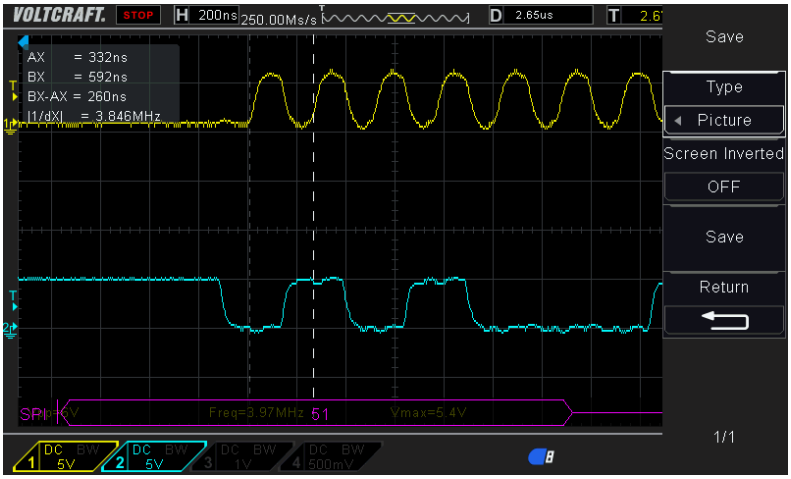
De functies tmrpcm.speakerpin, setvolume en play zorgen ervoor dat we kunnen starten met het spelen van de muziek op de juiste pin, met het juiste volume.

Merk op dat de uitgang van de PCM een PWM pin moet zijn (vb hier pin 9).

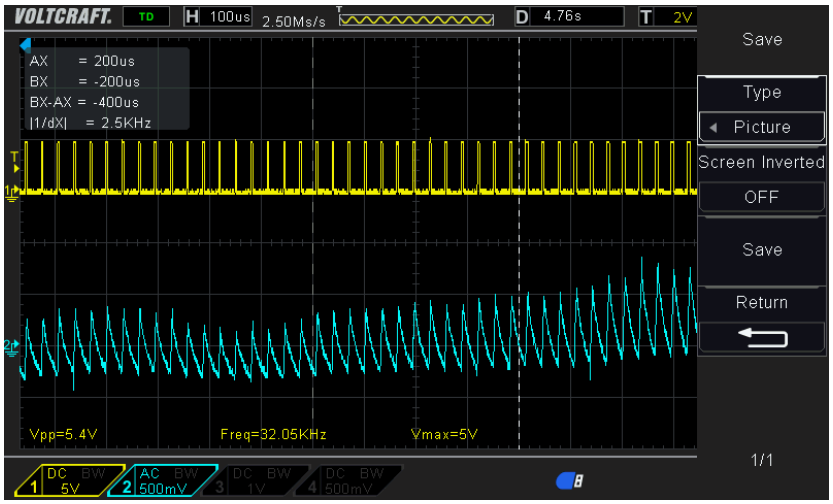
Uitdaging:

1. Meet op de **SPI bus** met jouw logic analyzer op de 4 lijnen en probeer zo het SPI protocol te analyzeren.
2. Meet op de **PWM uitgang** van de arduino met een logic analyzer / digitale scope en analyzer het signaal (voor en na de LDF).
3. Ga eens op zoek naar een **MP3 sinus signaal (misschien zelf opnemen)** en probeer dit te af te spelen en te meten op de PWM uitgang. Kan jij het verband terugvinden tussen PCM en de breedte van de pulsen.
4. Maak nu van de **WAV player een echte speler**: ik wil de knoppen start, stop, select hebben op mijn speler. Voeg er nog een digitale volume regeling bij. Alle info en de naam van de wav file is te vinden op een LCD of OLED. Zorg ook dat je meerdere liedjes kan afspelen. Enkele LEDs geven de status aan van de SD kaart (groen = SD kaart werkt goed, rood = SD kaart failed).

Voorbeeld van SPI meting met digitale scope:



Je moet steeds bij de **opgaande flanken** van ch1 (clk) de bits op ch2 (MISO) aflezen.
SPI signaal is dan 0b0101 0001 = 0x51



Hoe breder de puls op ch1 hoe hoger de amplitude op ch2. De elco gaat iets langer opladen tijdens de puls en dan weer ontladen tussen 2 pulsen in.

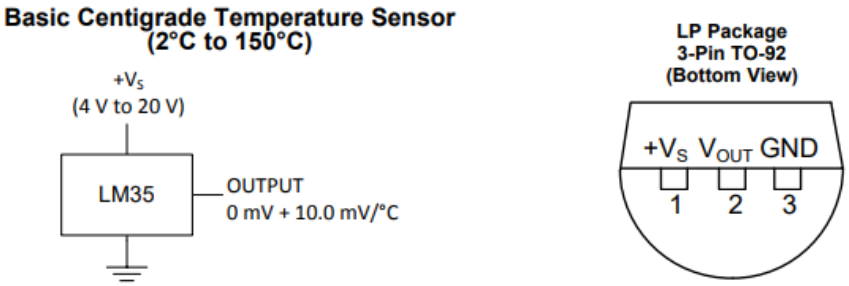
5.3 SD card gebruiken om waardes op te slaan en uit te lezen.

We kunnen niet alleen geluidsfiles lezen van een SD card. We kunnen ook gewone data opslaan en terug uitlezen van zo'n kaartje.

Dit kan bijvoorbeeld handig zijn als je een fijnstof sensor hebt die dagen lang data moet opslaan zonder aan een PC te hangen via een serial monitor.

Daarna kunnen we de SD kaart gewoon uitlezen en de waardes in excel plaatsen.

De opstelling van de arduino schakeling is in dit geval eenvoudig. We nemen als analoge meetsensor opnieuw de LM35 temperatuur sensor. Zie hfdst 3.1.1 van deze cursus voor meer info over de LM35.



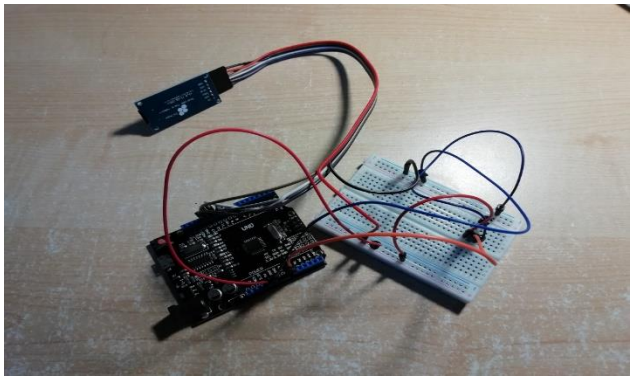
De LM35 geeft een analoge waarde terug met een precisie van 10mV / graden celsius.

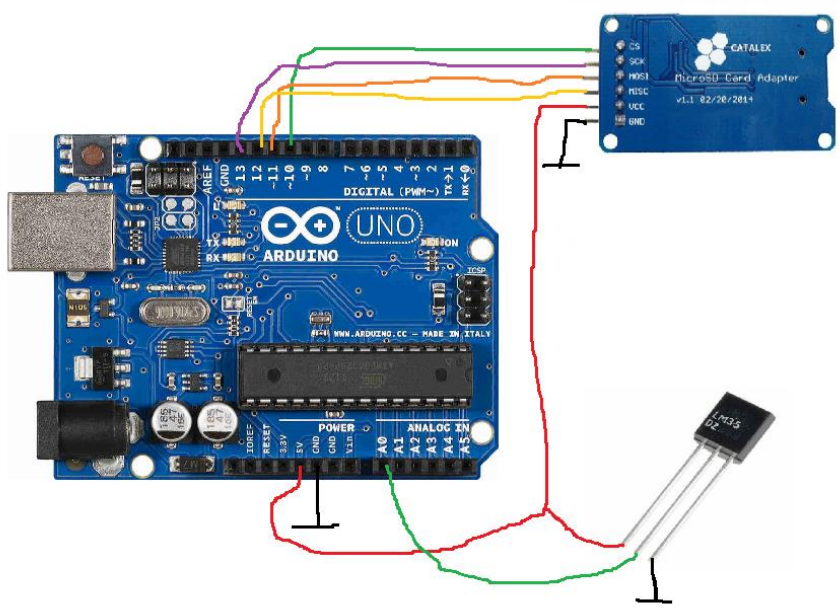
We sluiten de **Vout van de LM35 aan op pin A0** van de Arduino. Voor de rest sluit je massa en VCC aan op de voeding van de Arduino.

De SD card sluiten aan, net zoals bij de wav player.

SD card connecties

SD card	**Arduino**
GND	GND
VCC	+5V
MISO	12
MOSI	11
SCK	13
SC of SS	10





Schema van de LM35 temperatuursensor naar SD card schrijver.

Stop nu de volgende code in de Arduino:

LM35_met_SD_kaart

```
#include <SPI.h>
#include <SD.h>

#define LM35 A0      //LM35 zit op A0 of pin 14

const int chipSelect = 10;

unsigned long sampletime_ms = 3000; //3 sec
unsigned long tijd = 3;
unsigned long starttime;
int val;
```

Eerst stellen we, na het inladen van de SD en SPI library, de analoge pin van de LM35 in.

Ook de chipSelect van de SD card lezer kunnen we zelf bepalen, hier op pin 10 ingesteld.

Om de 3 seconden (3000 ms) wordt er een meting gedaan.

Merk op dat we voor de tijdmetingen een “unsigned long” variabele type hebben gebruikt.

Zo kunnen we heel lang, via de SD card metingen doen.

Het type long is 2^32 bits en geeft dus een heel groot getal. Reken maar eens uit.

```
void setup(){
  Serial.begin(9600);//start communication with computer
  //(not necessary just for error checking or code modification)

  pinMode(LM35,INPUT);  //LM35 zit op analoge ingang A0
  starttime = millis();//start de tijd voor de metingen

  Serial.print("Initializing SD card... ");
  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    return;
  }
  Serial.println("Card initialized.");
  Serial.print("LM35 meting elke seconde");
  Serial.println("\n");

}
```

In de setup() zetten we de tijdmeting aan en starten we de SD kaart op. Indien de SPI communicatie niet in orde is (foute draden gestoken, slechte SD kaart) krijgen we hier een foutmelding.

```
void loop (){

 //meet LM35 waarde in graden celsius elke sampletime en schrijf dan naar de SD kaart

  if ((millis()-starttime) >= sampletime_ms){//following code executed if the sampletime

    val = analogRead(LM35); //A0

    // print to the serial port too:
    Serial.print("Temperatuur: ");
    Serial.print(tijd);
    Serial.println(" seconden:");

    float mv = (val / 1024.0)*5000; // bereken mV
    float graden = mv / 10; //zet mV waarde om naar celsius waarde , graden is per 10mV
    Serial.print(graden);
    Serial.println(" graden celsius");
    Serial.print("\n");

     // open the file. note that only one file can be open at a time,
     // so you have to close this one before opening another.
     File dataFile = SD.open("meting.txt", FILE_WRITE);
```

In de loop gaan we elke 3 seconden aan de slag.

Dit wordt netjes gemeten door de voorwaarde in de if()

Daarna wordt de analoge waarde ingelezen van de LM35 en wordt deze omgerekend naar graden celsius.

Tenslotte wordt een data file op de SD kaart aangemaakt en geopend. De file naam is “meting.txt”.

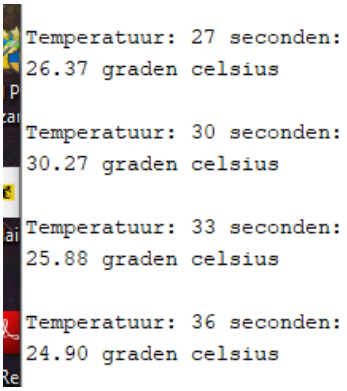
```
    // if the file is available, write to it:

     if (dataFile) {
       dataFile.print(tijd);
       dataFile.print(";");
       dataFile.println(graden);
       dataFile.close();
    }
    // if the file isn't open, pop up an error:
    else {
      Serial.println("error opening meting.txt");
    }

    tijd = tijd + 3;
    starttime  = millis();//reset the start time

  }
}
```

Indien de datafile aangemaakt of open gedaan is schrijven we de waarde van de variabele tijd en graden in de file, gescheiden door een “;”.

Daarna wordt de file weer netjes afgesloten voor een volgende keer.

De tijd wordt met 3 seconde verhoogt en de timer terug gereset.

Als je de serial monitor zou opendoen krijg je volgende gelijkaardige temperatuur metingen te zien:



In de file krijg je de volgende waardes te zien als je deze open doet:

Je kan deze files **importeren in excel** en zo kan je dan een curve maken van de temperatuur in functie van de tijd. Volg het volgende stappen plan om dit klaar te krijgen:

Haal de SD kaart uit de Arduino en stop deze in een kaartlezer of laptop

Sla de file op op jouw PC.

Start excel op en ga naar “gegevens”.

Klik op “uit tekst/csv”. Selecteer op jouw PC de txt file met de waardes.

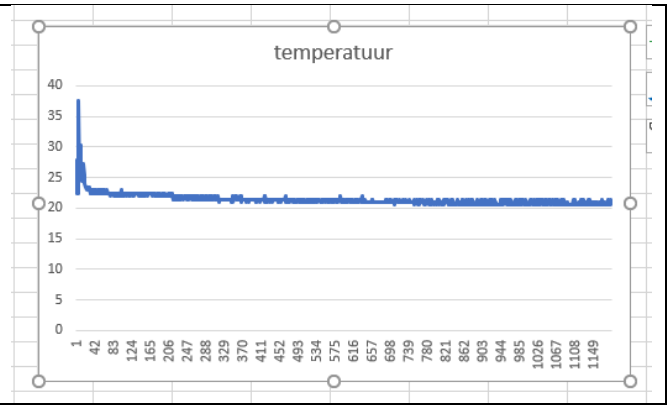
Controleer of de tabelinhoud klopt en bevestig door te klikken op “laden”

Zorg dat de temperatuur kolom / 100 is gedaan (de “.” in de tekstfile wordt niet als “,” gezien). Maak hiervoor een nieuwe kolom C en deel daarin de kolom B / 100

Geef de kolom C de naam “temperatuur”

Selecteer nu de temperatuur kolom en klik op “invoegen” -> lijn vlak diagram toevoegen -> 2D lijn

	A	B	C
1	tijd	0	temperatuur
2	3	2783	27,83
3	6	2246	22,46
4	9	2246	22,46
5	12	2246	22,46
6	15	2295	22,95
7	18	3760	37,6
8	21	2490	24,9
9	24	2539	25,39
10	27	2637	26,37
11	30	3027	30,27



6 Data versturen via RF signalen (extra)

In deze cursus hebben we al heel wat communicatie protocollen doorlopen. RS232, USB, I2C, SPI, PWM en PCM. Tot nog toe waren dit altijd **vaste bedradingen** (met uitzondering van de bluetooth signalen).

Nu is het tijd geworden dat we de draadloze communicatie eens onder de loop nemen in enkele experimenten. Eerst starten we met een RF verbinding op te zetten, in de volgende 2 hoofstukken zullen we de RFID en WIFI aan te tand voelen.

Een RF (radio frequentie) verbinding wordt typisch gebruikt bij drones, radio gestuurde wagentjes enz…

Wij gaan in dit experiment gebruik maken van 433MHz RF modules.

https://www.instructables.com/id/RF-315433-MHz-Transmitter-receiver-Module-and-Ardu/

Vooralleer we aan de slag kunnen gaan met het testen moeten we eerst het protocol weer onder de loop nemen.

We gaan onze 433MHz radio frequentie signalen versturen langs een **antenne.** In dit geval is dit een eenvoudige draad van een bepaalde lengte. Weet dat de lengte van de golf in eerste instantie even lang of kleiner moet zijn als de lengte van de draad om het signaal te kunnen ontvangen. Dan is de draad pas een **transmissielijn**. Merk op dat de antenne in feite het spoeltje is dat op de PCB zit, maar dit is te weinig om goed te kunnen ontvangen, vandaar **de externe te solderen antenne!**

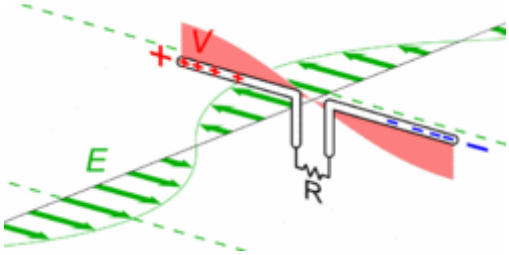
Golflengte = lichtsnelheid in lucht / frequentie

Golflengte = λ = **c / f** = 300 000 000 / 433 000 000 = 0.692 m = 69 cm

We mogen deze antenne lengte **delen door 4:**

Antenne lengte = λ / 4 = 0.692 / 4 = 0.173m = **17 cm** van de te solderen draad.

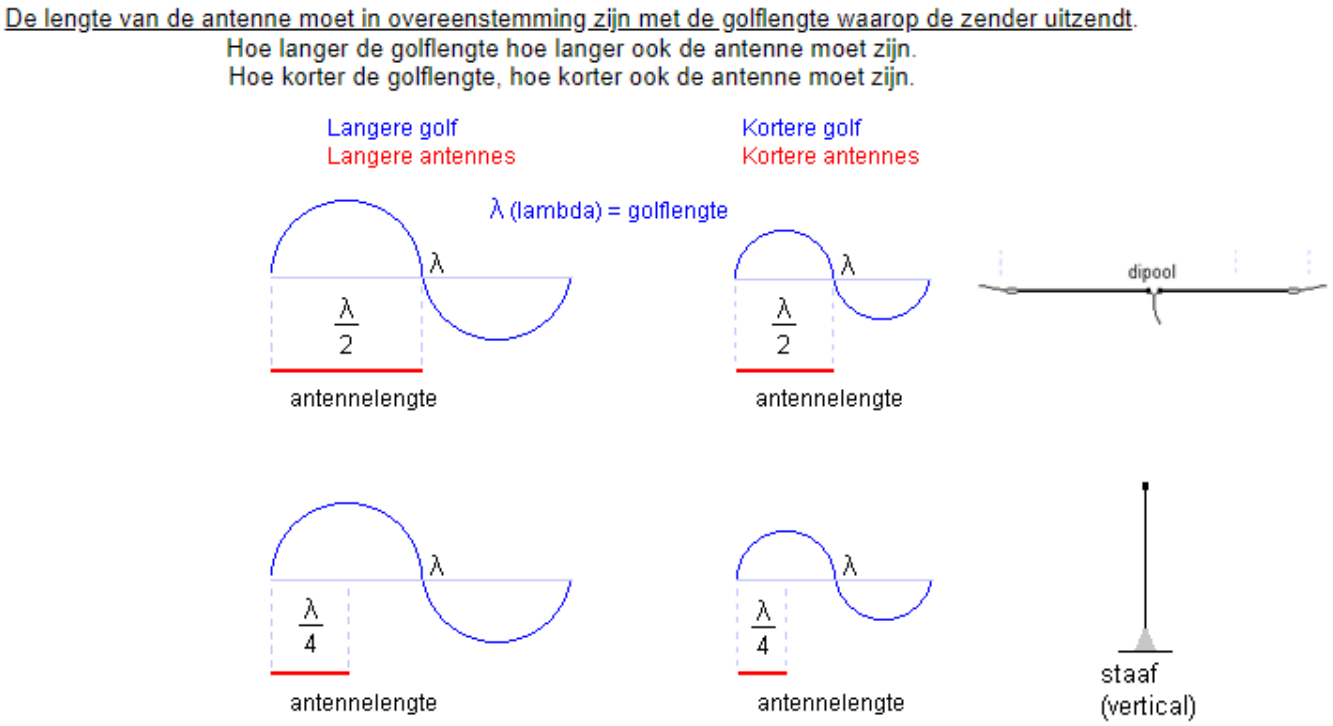
Hoe werkt de antenne en waarom delen door 4?



Een dipool (2 polen) antenne vangt op deze manier de golven op.

Er ontstaat over de R een polariteitsverschil. Dat gaan we insturen aan onze Arduino.

Merk op dat de elektrische energie (V) en de magnetische energie (E) loodrecht op elkaar staan.



We gaan gebruikmaken van een verticale staaf antenne (of gewoon een losse draad). Deze heeft dus genoeg aan ¼ golflengte te ontvangen van het ganse signaal om toch genoeg elektrische energie af te leveren aan de Arduino.

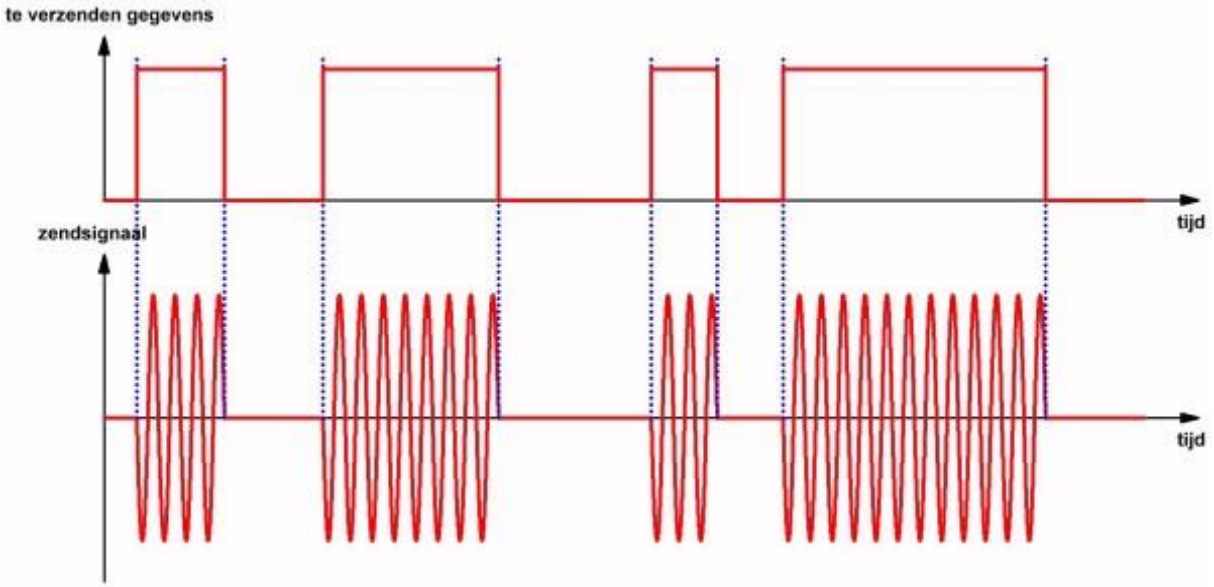
In ons project sluiten we de massa van de antenne aan op de Arduino en de antenne draad wordt gestuurd in de 433MHz module.

Merk op dat deze frequentie een vrijgegeven frequentie is voor alle draadloze communicatie over kleine afstanden.

6.1 De 433 MHz frequentieband

Waarvoor mag u 433 MHz gebruiken?
De 433 MHz frequentieband is voor iedereen beschikbaar voor draadloze digitale laagvermogen communicatie tussen apparaten zoals sensors, weerstations, garagepoort openers, draadloze deurbellen en domotica. De bekende goedkope zenders en ontvangers van onder andere **KlikAanKlikUit,** waarmee u apparaten op afstand draadloos kunt bedienen en verlichting kunt dimmen, werken in deze 433 MHz band.

433 MHz communicatie werkt met Amplitude Shift Keying
Deze apparaten werken met 100 % amplitude-modulatie. De draaggolf met een frequentie van 433 MHz is ofwel aanwezig, ofwel afwezig. U kunt bijvoorbeeld afspreken dat de aanwezigheid van de draaggolf overeen komt met het uitzenden van een digitale 'H' en het afwezig zijn deze draaggolf betekent dat een digitale 'L' wordt uitgezonden. Dit systeem noemt men **'Amplitude Shift Keying',** afgekort tot ASK. Dank zij het ontbreken van bidirectionele communicatie en encryptie zijn de communicatie-protocollen eenvoudig van aard en bijvoorbeeld met uw Arduino gemakkelijk te maken.



Het principe van Amplitude Shift Keying of ASK. (© 2018 Jos Verstraten)

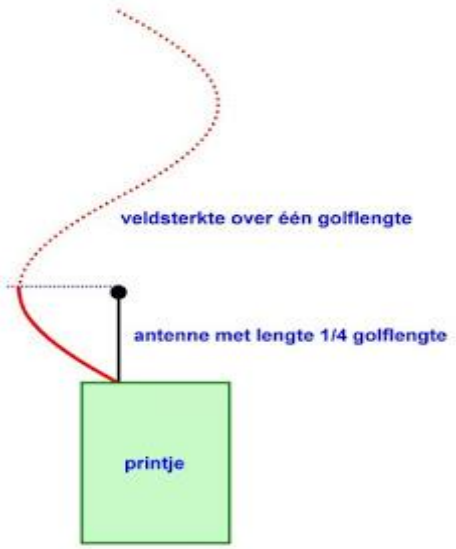
433 MHz communicatie werkt met korte telegrammen
Er zijn, misschien zonder dat u het weet, ongetwijfeld een heleboel apparaten in uw buurt aanwezig die zenden en ontvangen op 433 MHz. Denk aan de draadloze deurbel van uw buren, uw eigen wanddimmers en de garagepoort opener van uw andere buren. Omdat deze elkaar uiteraard kunnen storen is het verboden dat die apparaten constant signalen uitzenden. Stel dat u een weerstation hebt dat draadloos communiceert met een buitenthermometer. Die thermometer zal bijvoorbeeld iedere tien minuten even snel de momentane temperatuur verzenden, in de hoop dat zijn basisstation deze uitzending opvangt en er de temperatuur uit weet te filteren. Zo'n **pulstrein van**

433 MHz signalen noemt men een 'telegram'. Een telegram bestaat steeds uit twee woorden. Het eerste woord bevat een **unieke ID- of adres-code,** waardoor de ontvanger weet dat 'zijn' zender aan het zenden is. Het tweede woord bevat de **gegevens** die de zender moet versturen, bijvoorbeeld de temperatuur. Het is toegestaan dat de zender zijn telegram een paar maal achter elkaar verstuurt, zodat de kans vrijwel 100 % wordt dat de ontvanger het bericht ontvangt.
Dank zij deze communicatie via korte telegrammen met ID's kunt u in uw huis zowel zenders en ontvangers van KlikAanKlikUit toepassen, een draadloos weerstation installeren en een elektronisch te openen garagepoort gebruiken zonder dat deze drie systemen elkaar in de weg zitten.

Wat u goed moet beseffen
ASK-communicatie op 433 MHz is een **uiterst eenvoudig systeem**. Het is ideaal om zélf mee te experimenteren omdat u er weinig kennis en eenvoudige hardware voor nodig hebt. Het systeem is echter **niet 100 % betrouwbaar**. Als **twee zenders toevallig precies op hetzelfde moment** een telegram versturen is de kans groot dat beide signalen met elkaar **interfereren** en dat de ontvangers de telegrammen niet als 'eigen' beschouwen en er niets mee doen.

Het gebruik van antennes
Als u 433 MHz modules gebruikt zonder extra antenne's zult u maar een **heel beperkte reikwijdte** krijgen. Vandaar dat wordt aanbevolen zowel de zender- als de ontvanger-modules aan te sluiten op een **externe antenne (zelf te solderen).** De meeste printjes hebben hiervoor een aansluiting.

Blijft de vraag hoe die antenne er uit moet zien en hoe groot hij moet zijn.
In de meeste artikelen over dit onderwerp wordt aangeraden een **staafantenne met een lengte van een kwart golflengte toe te passen, een zogenaamde *'quarter wave'* antenne.**
Zoals uit onderstaande figuur blijkt is de **veldsterkte op de top** van een dergelijke antenne **maximaal**. Bij het **signaalinvoerpunt** is de **veldsterkte echter nul en de stroom maximaal**. Op deze manier wordt er een **maximaal vermogen in de antenne gepompt** en zal de reikwijdte het grootst zijn.
Zoals u misschien weet is de golflengte van een signaal gelijk aan de lichtsnelheid gedeeld door de frequentie van het signaal. Als u die formule toepast op een signaal van 433 MHz blijkt dat de golflengte van een dergelijk signaal gelijk is aan 69,28 cm. Een quarter wave voor deze frequentie moet dus een lengte hebben van 17,32 cm.

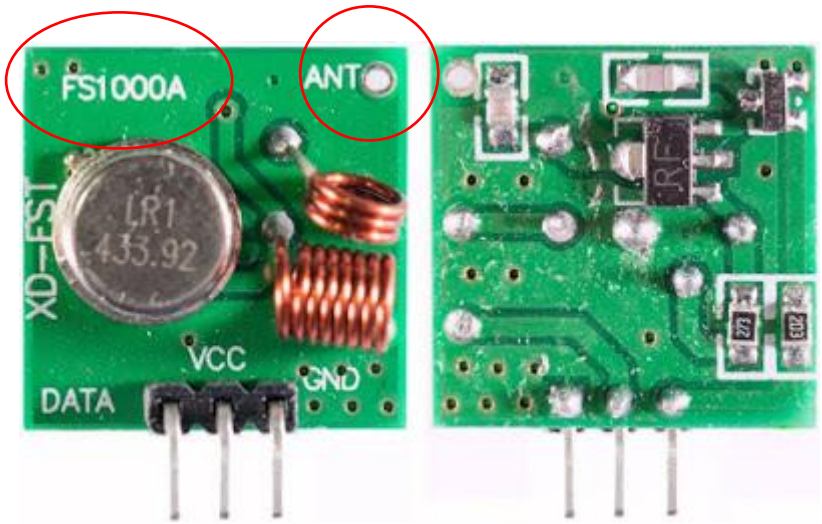


Het principe van een quarter wave antenne. (© 2018 Jos Verstraten)

Keuze in overvloed
Er bestaan tientallen modules die een signaal van 433 MHz genereren en die een data-aansluiting hebben om er ASK-modulatie op toe te passen. Ook voor het ontvangen en demoduleren van zo'n signaal kunt u kiezen uit tientallen printjes. Na uitgebreid speuren op internet vonden wij een eenvoudige en uiterst goedkope combinatie die uitstekend bruikbaar is voor het uitvoeren van uw allereerste experimenten. De bodemprijs die wij voor deze set vonden bedraagt slechts € 1,42. Let er bij het bestellen op dat u de frequentie moet specificeren. De set wordt namelijk zowel met een frequentie van **433 MHz** als 315 MHz geleverd. **315 MHz is in Japan** toegestaan voor vrije communicatie, maar niet in **Europa.**

6.2 De zendermodule FS1000A of XD-FST

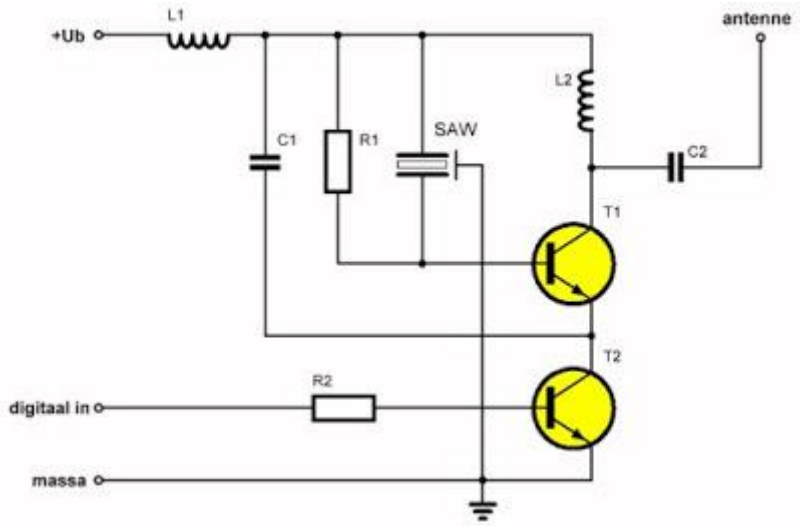
De presentatie
De in onderstaande figuur voorgestelde module wordt geleverd met twee benamingen, namelijk FS1000A of XD-FST. Op dit slechts 19 mm bij 19 mm grote printje zie u aan de ene zijde twee spoeltjes en een rond, metalen onderdeel met de code LR1 433.92. Dat blijkt een zeer exotisch onderdeel te zijn, namelijk een **SAW-resonator** die resoneert op een frequentie van **433,92 MHz.** SAW is het letterwoord van **S**urface **A**coustic **W**ave. Op de andere zijde treft u twee SMD-transistoren en een paar weerstandjes en condensatoren aan. De drie aansluitpennetjes laten geen twijfel bestaan over hun functie. Rechtsboven ziet u het printgaatje voor het aansluiten van een antenne.



De zendermodule FS1000A of XD-FST. (© 2018 Jos Verstraten)

Het schema van de zendermodule
Het schema van deze module is getekend in onderstaande figuur. De transistor T1 is de oscillator. De **SAW-resonator** is opgenomen tussen de collector en de basis en zorgt dus voor het vervullen van de **oscillatie-voorwaarde** van de schakeling. Alleen **ruis met een frequentie van 433,92 MHz wordt rondgekoppeld en zal dus steeds meer worden versterkt tot er een mooi zendsignaal** met deze frequentie ontstaat. Dit signaal wordt via een LC-kring gekoppeld aan de antenne. Het zendsignaal vloeit uiteraard ook door dit **spoeltje L2**. Rond dit spoeltje ontstaat dus een **elektromagnetisch veld** en het is dit veld dat er verantwoordelijk voor is dat de module ook zonder antenne een bepaald **klein zendbereik** heeft.
De ASK-modulatie wordt op een wel heel primitieve manier verzorgd door T2. Als u de basis van deze transistor stuurt met een **'L' zal de transistor sperren**. Transistor T2 wordt stroomloos en houdt

uiteraard op met oscilleren. Stuurt u **een 'H' in de basis, dan gaat T2 geleiden** en wordt de oscillator rond T1 aan het werk gezet.



Het schema van de zendermodule FS1000A of XD-FST. (© 2018 Jos Verstraten)

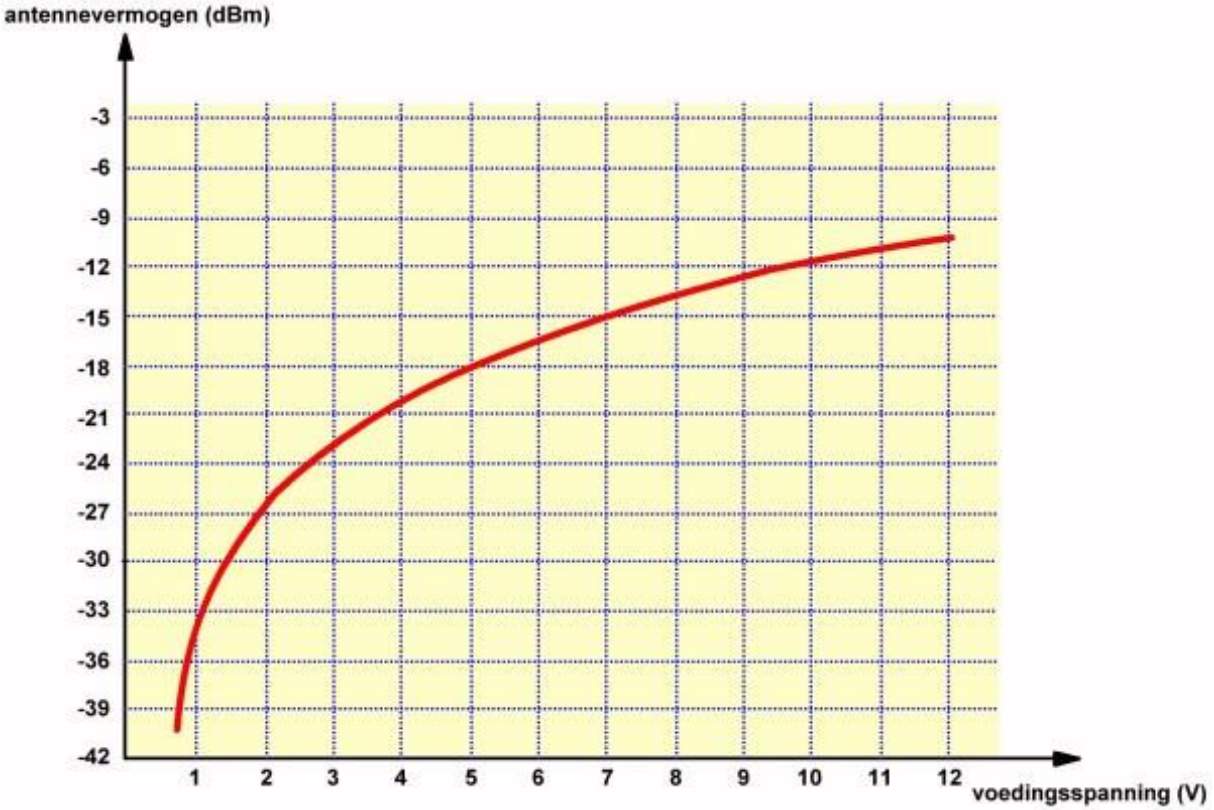
De technische specificaties

De technische specificaties van deze module zijn, kort samengevat:

- **Productcode:** XD-FST of FS1000A
- **Fabrikant:** onbekend
- **Frequentie:** 433,92 MHz of 315,00 MHz
- **Modulatie:** ASK
- **Zendvermogen:** 10 mW
- **Datasnelheid:** 10 kB/s
- **Bereik:** ongeveer 20 m (afhankelijk van voeding en antenne)
- **Voedingsspanning:** 3,0 Vdc ~ 12,0 Vdc
- **Voedingsstroom:** 28 mA max. actief
- **Afmetingen:** 19 mm x 19 mm
- **Gewicht:** 3 g

Het zendvermogen

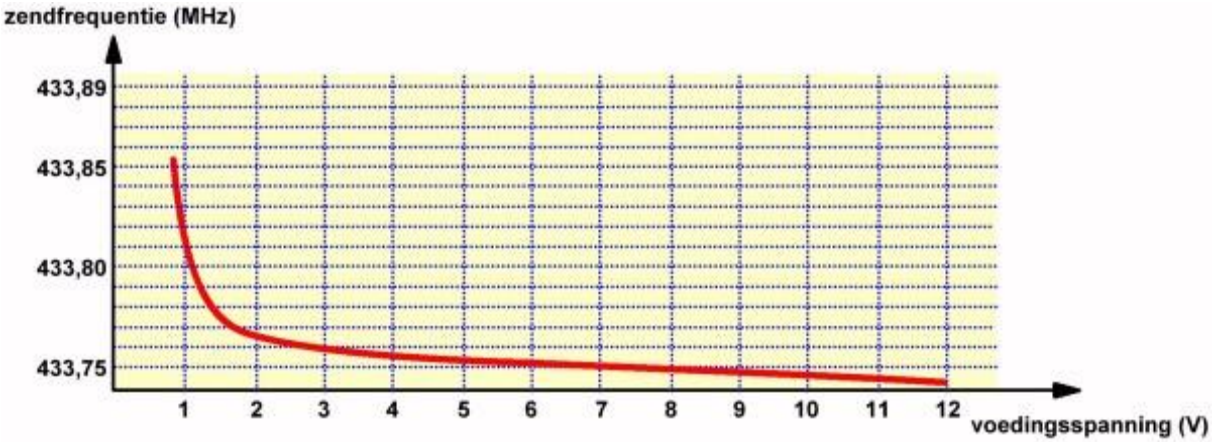
Het zendvermogen van de module wordt gespecificeerd als **10 mW.** In de praktijk blijkt echter dat het zendvermogen nogal afhankelijk is van de voedingsspanning. Onze collega's van Gough's Tech Zone hebben het reële zendvermogen in functie van de voedingsspanning gemeten, metingen die wij in een mooie grafiek hebben verwerkt. Hieruit blijkt dat het zendvermogen met 13 dB stijgt als u de voedingsspanning opvoert van 3,0 V tot 12,0 V.



Het zendvermogen in functie van de voedingsspanning.
(© 2018 Jos Verstraten naar gegevens van Gough's Tech Zone)

Stabiliteit van de frequentie

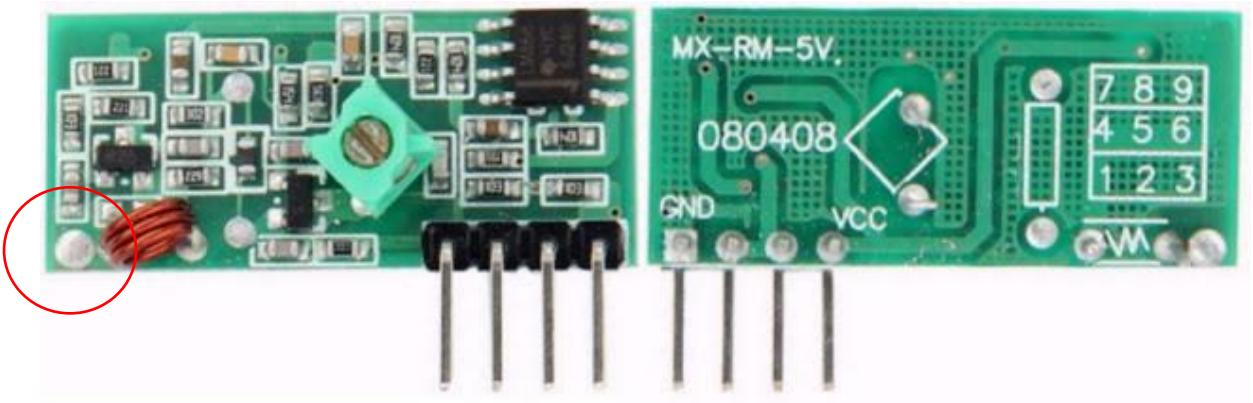
Door de collega's van Gough's Tech Zone werd ook de zendfrequentie in functie van de voedingsspanning opgemeten. Uit onderstaande grafiek blijkt dat deze ongeveer 100 kHz varieert tussen een voedingsspanning van 1,0 V en 12,0 V. Dat is vreemd want u denkt waarschijnlijk, net zoals wij, dat de SAW een zeer stabiele resonantiefrequentie heeft die zich niets aantrekt van de voedingsspanning van de schakeling waarin hij wordt toegepast.



De zendfrequentie in functie van de voedingsspanning.
(© 2018 Jos Verstraten naar gegevens van Gough's Tech Zone)

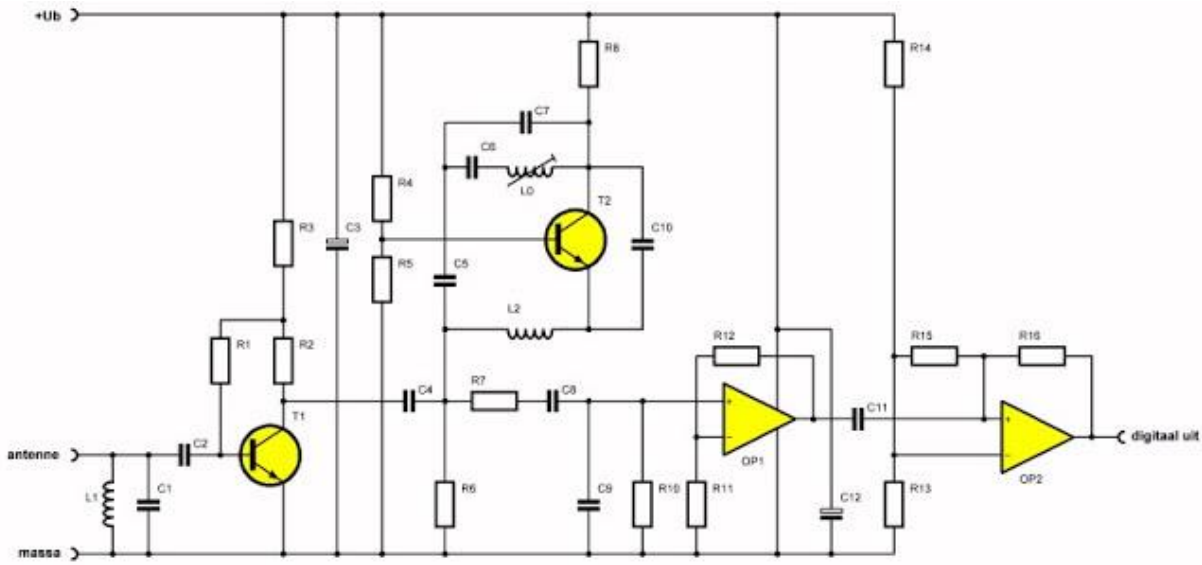
6.3 De ontvangermodule XD-RF-5V

De presentatie
Zoals onderstaande foto bewijst is de ontvanger heel wat complexer dan de zender. Op de 30 mm x 14 mm grote print ziet u een heleboel componenten. Dat is logisch, want de toch al lage vermogensflux bij de zendantenne neemt heel snel af als u de afstand tussen zender en ontvanger vergroot. De ontvanger zal waarschijnlijk maar **een paar µW vermogen oppikken** en moet dat **kleine signaal flink versterken.** Bovendien krijgt de ontvanger te maken met ruis en het ligt voor de hand dat er een **AVR**, een **A**utomatische **V**ersterking **R**egeling, aanwezig moet zijn om de versterking van de 433 MHz versterker te regelen. U ziet twee transistoren en een dubbele operationele versterker. De twee middelste pennen van de connector zijn met elkaar verbonden en vormen de uitgang van de schakeling. Helemaal **linksonder, naast het spoeltje, is het soldeereilandje te zien waarop u eventueel een antenne kunt aansluiten.**



De voor- en achterzijde van de ontvangerprint XD-RF-5V. (© 2018 Jos Verstraten)

Het schema van de ontvangstmodule
Er zijn diverse versies van de module in omloop met kleine variaties in het schema en in de print. Toch komt de elektronica in grote lijnen overeen met wat wij in onderstaand schema hebben getekend.
Het antennesignaal wordt aangeboden aan een **resonantiekring L1/C1** die is afgestemd op de zenderfrequentie. Nadien volgt een **transistorversterker** rond T1. Het versterkte signaal wordt uitgevoerd via de scheidingscondensator C4. Rond transistor T2 is een eenvoudige **spanningsgestuurde verzwakker** samengesteld. Deze sluit een kleiner of groter deel van het signaal over R6 kort naar de voedingsspanning. Op deze manier zal de schakeling maximaal versterken als er geen signaal wordt ontvangen en zal er **minimaal versterkt worden als een mooi 433 MHz signaal** wordt **ontvangen**. Via het afstembaar spoeltje L0 kunt u deze kring precies afregelen op de zenderfrequentie.
Het signaal wordt nadien weer versterkt door de schakeling rond OP1. De **kleine bandbreedte** van deze versterker zorgt ervoor dat **alleen de omhullende van de 433 MHz bursts wordt doorgekoppeld. Opamp OP2 is als comparator met hysteris** geschakeld, die verantwoordelijk is voor het omzetten van het **ASK-gemoduleerde signaal in een mooie digitale puls.**



Het schema van de ontvangerprint XD-RF-5V. (© 2018 Jos Verstraten)

De technische specificaties

De technische specificaties van de ontvangermodule zijn:

- **Productcode:** XD-RF-5V
- **Fabrikant:** onbekend
- **Frequentie:** 433,92 MHz
- **Demodulatie:** ASK
- **Voedingsspanning:** 5 Vdc
- **Voedingsstroom:** 4 mA
- **Gevoeligheid:** -105 dB
- **Afmetingen:** 30 mm x 14 mm x 7 mm

Opmerkingen

De schakeling is extreem gevoelig voor variaties in de voedingsspanning, u moet dus absoluut een **gestabiliseerde 5 V voeding toepassen**. De ontvanger is bovendien **heel erg gevoelig voor stoorsignalen.** Dat komt doordat de versterking maximaal wordt opgeschroefd als geen 433 MHz signaal wordt ontvangen. De ruis die dan wordt versterkt kan tot gevolg hebben dat de comparator aan de uitgang toch omslaat en een puls genereert.

U doet er verstandig aan vóór het verzenden van het echte telegram **eerst een burst 433 MHz sinussen te verzenden**, zodat de ontvanger deze ontvangt en zijn **versterking kan aanpassen** aan de grootte van het ontvangen signaal.

In de 'VirtualWire library for Arduino' worden ook softwarematige oplossingen voorgesteld om het probleem van stoorpulsen aan te pakken.

6.4 Het toepassen van beide modules

Zonder intelligente hardware gaat het niet
Als u nu denkt dat u draadloos een LED'je kunt aansturen door met een schakelaar het zendertje in en uit te schakelen en de uitgang van de ontvanger via een transistor met de LED te verbinden hebt u het goed mis. De ontvanger pikt **immers ieder 433 MHz signaal** op en reageert bovendien ook op flinke ruissignalen. De LED zou voortdurend aan- en uitfloepen zonder dat u er iets voor moet doen. Het systeem werkt alleen foutloos als u de zender zó uitbreidt dat deze de reeds besproken **telegrammen** verzendt. Ook de ontvanger moet **intelligente elektronica** aan de uitgang krijgen die de telegrammen herkent en alleen als zo'n telegram wordt ontvangen de gewenste actie uitvoert.

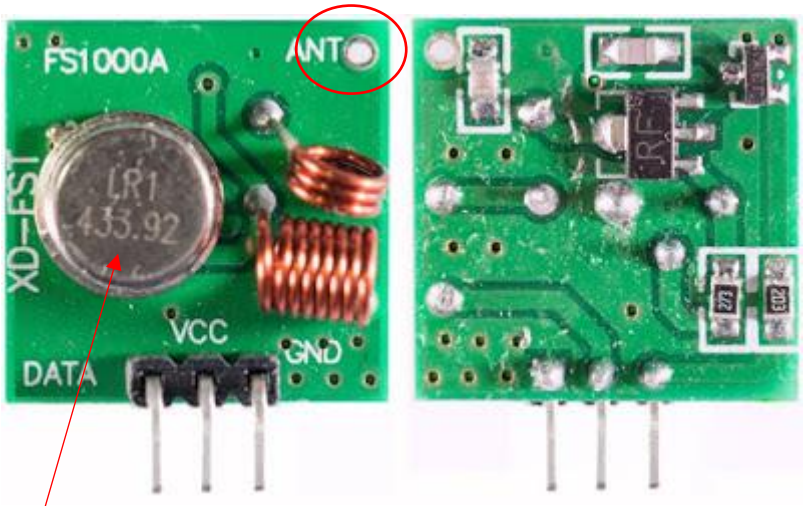
Hier kunnen de Arduino's of Raspberry pi's dienst doen!

De antennes (die op de PCB gesoldeerd zijn)
De twee meegeleverde antennes zijn gewikkeld van solide koperdraad met een diameter van 0,8 mm, bevatten 23,5 windingen, zijn 32 mm lang en hebben een diameter van 5,5 mm. De impedantie bij de zendfrequentie bedraagt 50 Ω en de versterking 2,1 dB. De staande golf verhouding (Voltage Standing Wave Ratio VSWR) is, als het u wat zegt, kleiner dan 1,5.

6.5 Aan de slag met Arduino

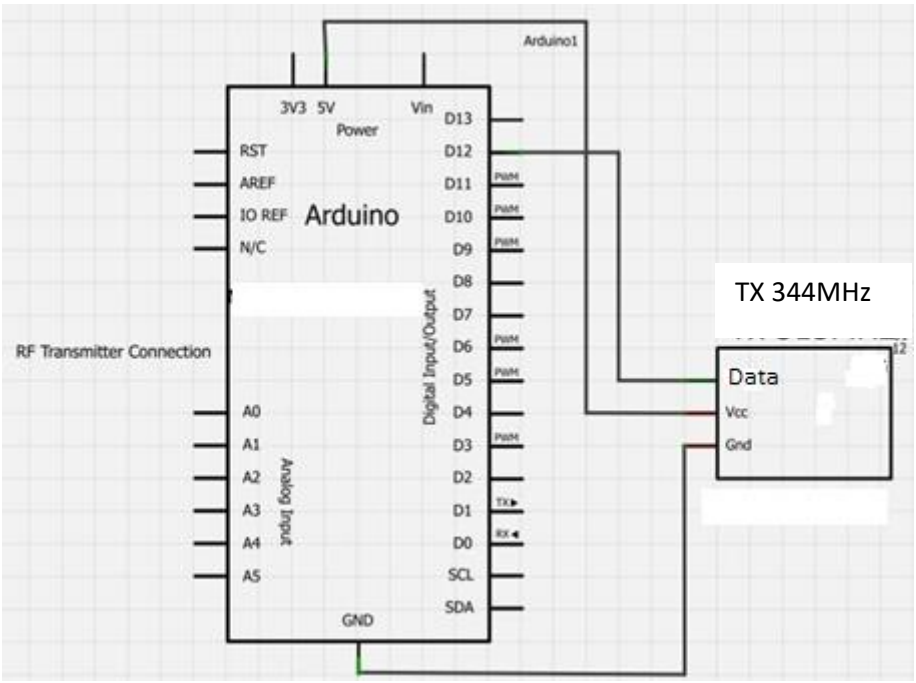
We moeten eerst en vooral gebruik maken van 2 uno's. De ene gaan we instellen als zender, de andere als ontvanger.

6.5.1 De 433MHz zender



SAW met frequentie aanduiding

Sluit het volgende schema aan op de eerste Arduino UNO.



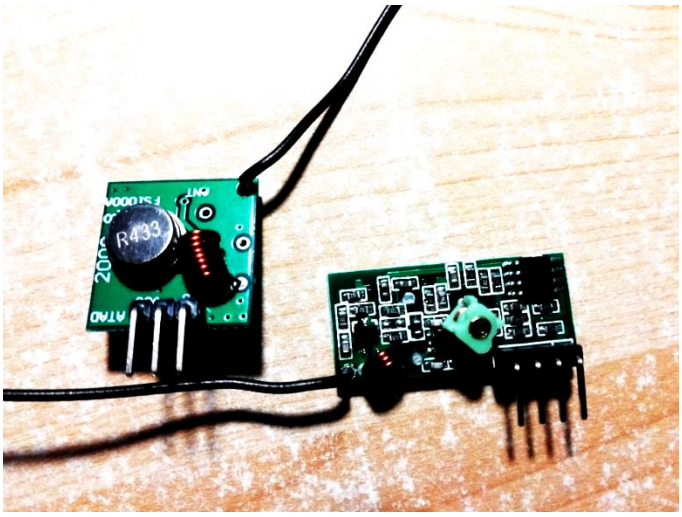
Transmitter (FS1000A) 433MHz	**Arduino**
Data	Pin 12
VCC	+5V
GND	GND

Merk op dat de zender 2 frequenties kan hebben: de Japanse 315MHz en de Europese 433MHz.

Kijk na wat er op de SAW resonator staat (zie metalen ronde schijfje op de PCB).

De zender en ontvanger moeten van **hetzelfde frequentie type** zijn!

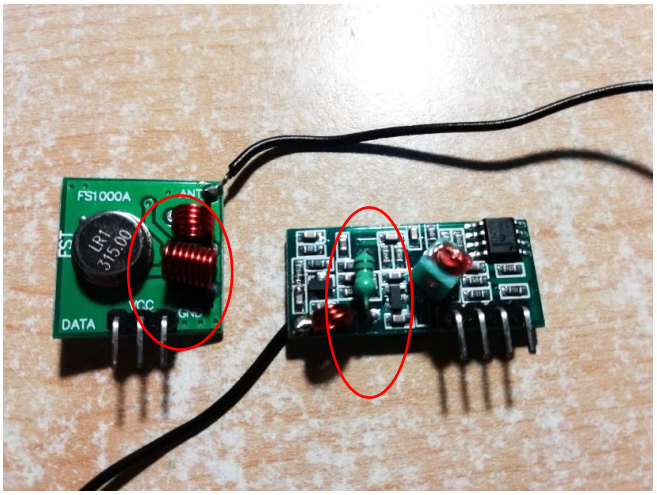
De 433MHz TX en RX zien er zo uit: **433MHz SAW en tekst "MX-RM-5V" op de ontvanger.**



433MHz op de SWA resonator, geen dikke weerstand op de ontvanger!

Antenne lengte ongeveer 17 cm.

De 315MHz TX en RX zien er zo uit**: 315MHz SAW en tekst "RF-5V" op ontvanger.**

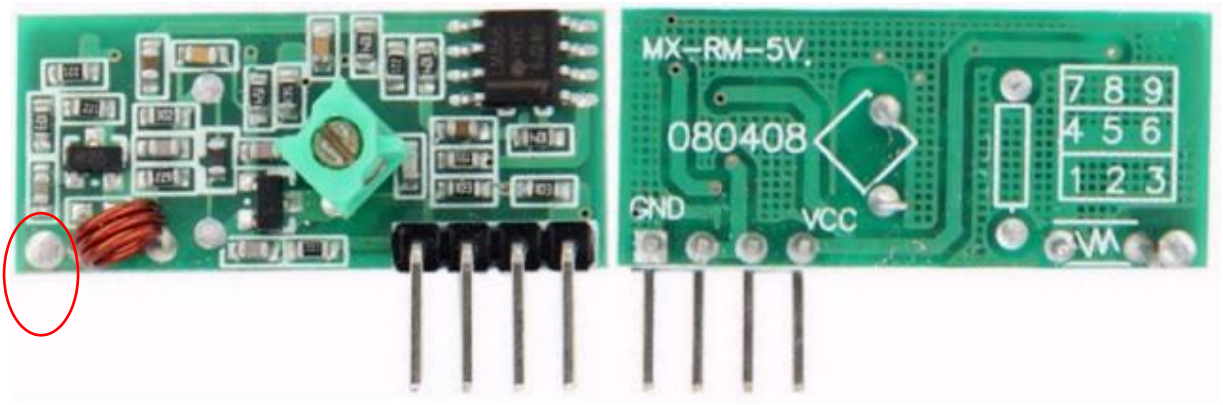


315MHz op de resonator op de zender en wel een **dikke weerstand** extra op de ontvanger. Merk op dat in dit geval de antenne korter is!

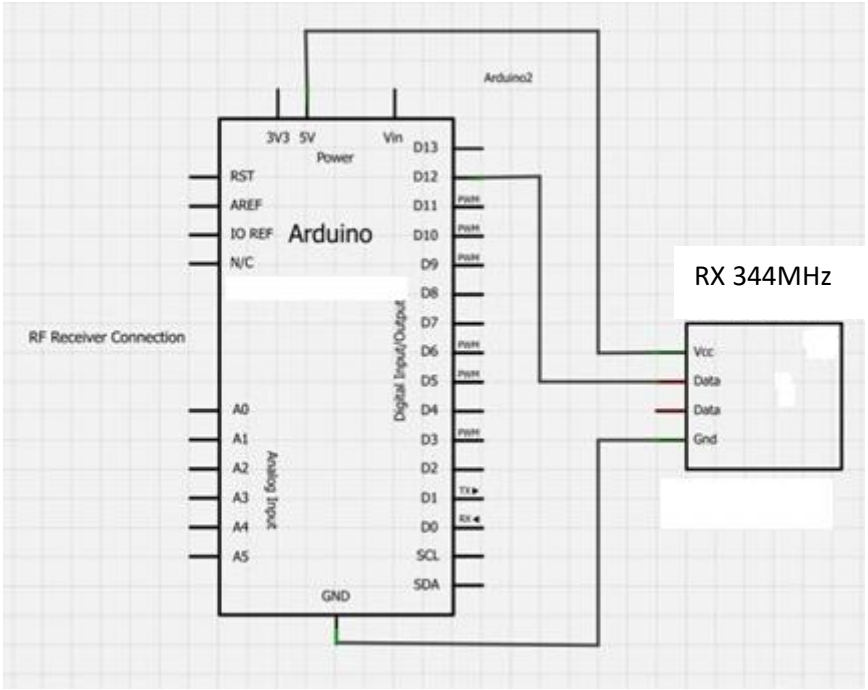
Golflengte = 300 000 000 / 315MHz = 21 cm

Soldeer de antenne draad van 17,32 of 21 cm aan de aangeduide verbinding op de zender PCB.

6.5.2 De 433MHz ontvanger



Hierop staat de **tekst "MX-RM-5V".**

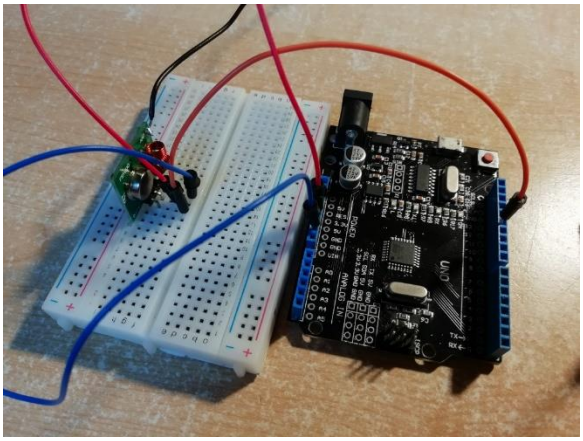


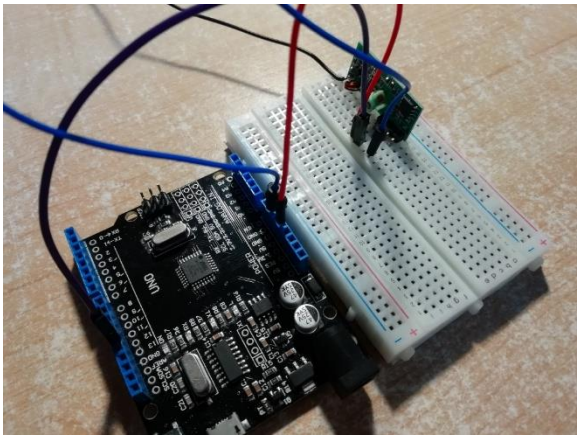
Sluit ook nu de ontvangst module aan op de 2de Arduino UNO.

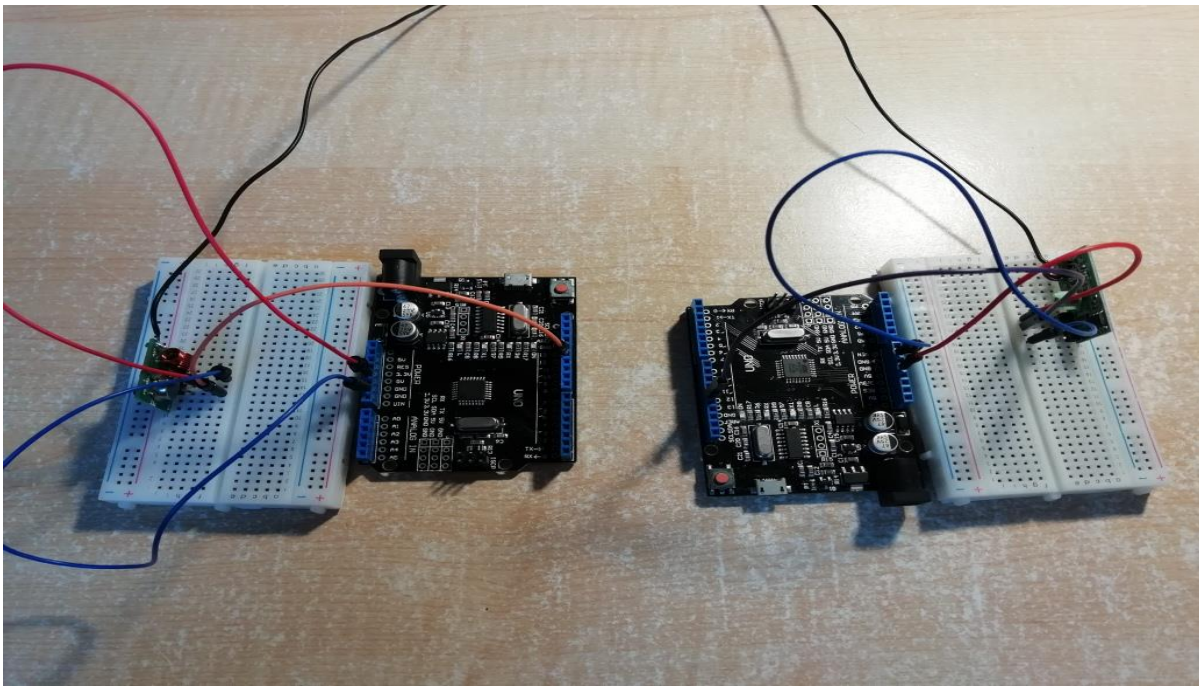
Receiver (XD-RF-5V) 433MHz	**Arduino**
VCC	+5V
Data (2 middenste pinnen)	Pin 12
GND	GND

Soldeer eveneens hier een draadje van **17.32 cm op de antenne** ingang om een betere ontvangst te hebben. Tenzij je de Chinese versie hebt van 315MH. Dan is het 21 cm.

6.5.3 De code voor de zender en ontvanger







Voorzie voor beide modules een aparte voeding (vb via 9V batterijen). Indien je via de serial monitor wil gaan werken moet elke module op een andere PC aangesloten worden. Een LCD aansturen is hier dan makkelijker.

Nu de modules zijn opgebouwd kunnen we deze voorzien van software.

We gaan gebruik maken van de library **“VirtualWire.h”**.

Haal de zip file (zie USB stick) binnen in Arduino via de gekende weg.

Nu gaan we eerste de zender van software voorzien. Open dus de juiste arduino oefening!

Test de code via de serial monitor en houd de LED13 in het oog.

RF_zender_433MHz_test_LED

```
//simple Tx on pin D12
//Written By : Mohannad Rawashdeh
// 3:00pm , 13/6/2013
//http://www.genotronex.com/
//.................................
#include <VirtualWire.h>
char *controller;

void setup() {
    pinMode(13,OUTPUT);
    vw_set_ptt_inverted(true); //
    vw_set_tx_pin(12);
    vw_setup(4000);// speed of data transfer Kbps
    Serial.begin(9600);
}
```

We stellen de data snelheid in van de zender (4000 Kbps) en geven aan dat we langs pin 12 gaan uitsturen.

```
void loop(){

  controller="1" ;
  vw_send((uint8_t *)controller, strlen(controller));
  vw_wait_tx(); // Wait until the whole message is gone
  digitalWrite(13,1);
  Serial.println("led13 AAN");
  delay(2000);

  controller="0" ;
  vw_send((uint8_t *)controller, strlen(controller));
  vw_wait_tx(); // Wait until the whole message is gone
  digitalWrite(13,0);
  Serial.println("led13 UIT");

  delay(2000);

}
```

Als we een “1” zenden dan gaat ook LED13 aan op de zend UNO.

Als we een “0” zenden gaan de LED13 uit doen.

Download nu in de ontvanger de volgende code. Ook hier testen met de serial monitor en LED 13 in het oog houden.

RF_ontvanger_433MHz_test_LED

```
//simple Tx on pin D12
//Written By : Mohannad Rawashdeh
// 3:00pm , 13/6/2013
//http://www.genotronex.com/
//.................................
#include <VirtualWire.h>

void setup()
{
    vw_set_ptt_inverted(true); // Required for DR3100
    vw_set_rx_pin(12);
    vw_setup(4000); // Bits per sec
    pinMode(13, OUTPUT);

    vw_rx_start(); // Start the receiver PLL running
    Serial.begin(9600);
    Serial.println("start ontvanger");
}
```

Nu stellen we de RX (ontvanger) in op pin 12 met dezelfde snelheid van 4000 bps.

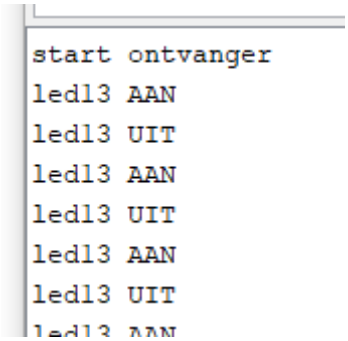
Dan starten we de ontvanger. Deze wacht nu op data.

```
void loop()
{
    uint8_t buf[VW_MAX_MESSAGE_LEN];
    uint8_t buflen = VW_MAX_MESSAGE_LEN;

        if (vw_get_message(buf, &buflen)) // Non-blocking
        {
          if(buf[0]=='1'){
            digitalWrite(13,1);
            Serial.println("led13 AAN");

          }
          if(buf[0]=='0'){
              digitalWrite(13,0);
              Serial.println("led13 UIT");
          }

        }
}
```

Als we een “1” ontvangen gaat de LED13 aan, anders uit.

Tenslotte sluit je de zender aan op een 9V batterij. Nu begint de zender met de uitzending van een “0” of een “1” (LED13 uit of aan).

De ontvanger kan je laten hangen op de serial monitor. Hier verschijnt nu het volgende:

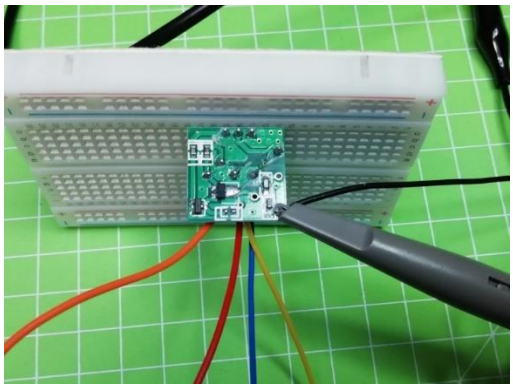


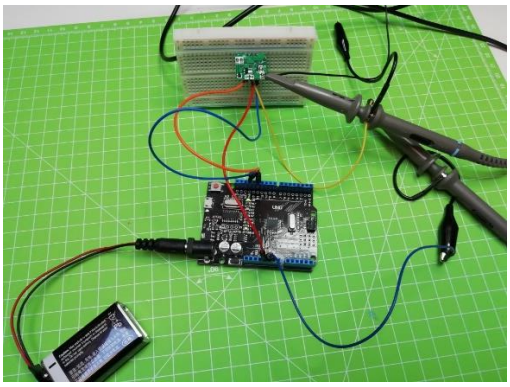
De LED13 gaat op het zelfde ritme aan/uit op de ontvanger UNO.

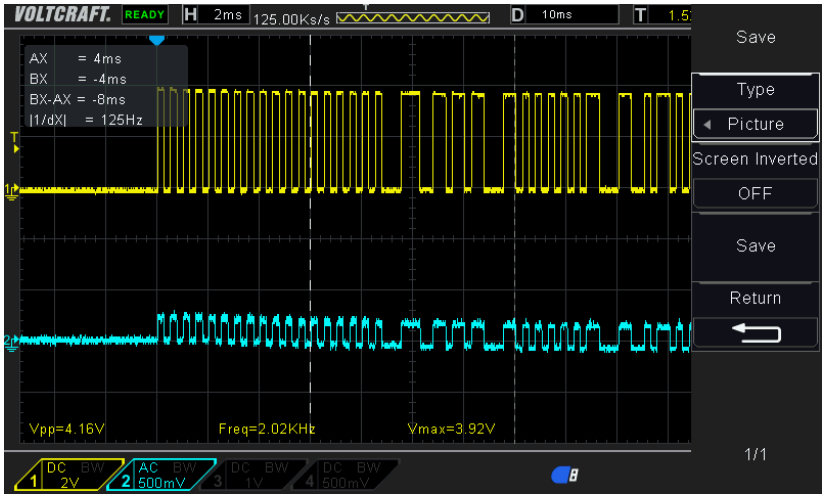
Uitdagingen:

1. Meet de ingang (pin 12) van de zender UNO samen met de uitgang van de zender (tussen L2 en C2) en kijk of je de data kan ontleden (hoe ziet de ASK er uit?)
2. Probeer nu “meerdere letters” door te sturen en op een serial monitor en LCD scherm te tonen op de ontvanger.
   https://randomnerdtutorials.com/rf-433mhz-transmitter-receiver-module-with-arduino/
3. Maak een duplex systeem (beide zijden van de RF hebben een zender en ontvanger). Zo kunnen we in 2 richtingen teksten sturen en tonen op een LCD scherm.
   Heb je hiervoor 2 ID’s nodig?
4. Stuur via RF de servo pan en tilt aan. Dit zijn 2 servo’s waar een laser op gemonteerd is (zie gevorderden 1 cursus).
   Zorg dat je aan de TX zijde een analoge / digitale joystick voorziet om de bewegingen en het ON/OFF schakelen van de laser in te stellen.
5. Doe de oefening 4 opnieuw, maar nu met het aansturen van een Arduino robot via RF. Zorg dat de robot draadloos in de 4 richtingen kan rijden en dat de snelheid instelbaar is.
   Extra: de robot geeft feedback terug via een afstandsensor aan de zender. Te kort tegen een hindernis zorgt ervoor dat een LED gaat branden op de zender.
6. Maak een morse code machine. Je hebt een knop op de TX dewelke de lengte van het indrukken van de knop meet. Kort drukken is een “0”, langer drukken is een “1”. Zend deze info door naar de ontvanger. De ontvanger zet de 0 en 1 terug om in een korte en lange piep op een buzzer.

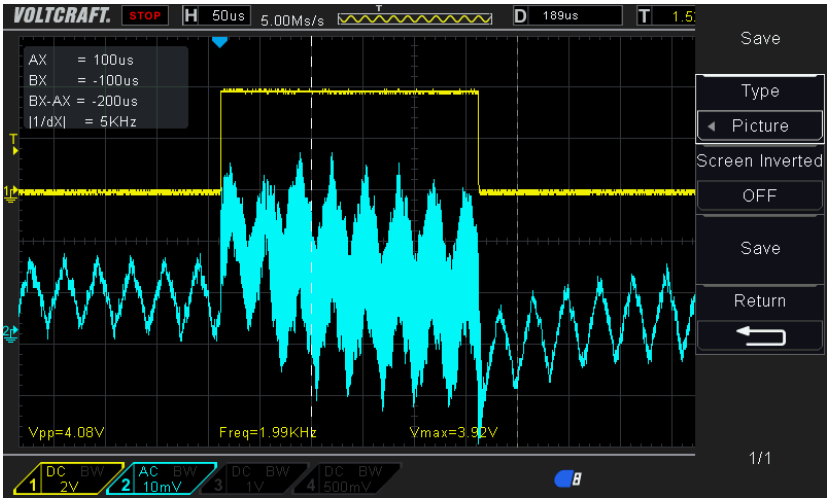
Enkele metingen van de 433Mhz zender:







Meting op pin 12 van de Arduino en tussen L2 en C2 van de zender.



Duidelijk zichtbaar hoe ASK werkt (hoog is grotere amplitude dan laag)
Je kan ook zien dat het digitale signaal op een hogere frequentie is gemoduleerd.

7 RFID ingezet om een relays aan te sturen

We hebben via bluetooth en RF over grotere afstanden een draadloos signaal gebruikt. Nu gaan we werken met RFID (radio frequentie identificatie) . Je kent de poortjes wel in de winkel of bib. Wanneer we er doorlopen zonder betaald te hebben of de spullen hebben laten uitscannen, gaat er een alarm af. Je kan ook RFID gebruiken om een poortje te openen. Hoe werkt dit nu?

Voor het RFID systeem hebben vooral een **tag** nodig

en een RFID twee weg radio transmitter – receiver, m.a.w. **de reader** die het signaal stuurt naar de tag en het antwoord leest.

Hoe werkt de RFID techniek?

We gaan weer werken met **radio frequenties**, of anders gezegd, we sturen elektromagnetische golven door de lucht. We proberen achteraf dan deze golven terug op te vangen. De frequentie range kan heel uitgebreid zijn. Deze RF522 module werkt op **13.56 MHz** (de helft als er staat op het X-tal op de module). **ID staat voor identification.** Iets proberen te identificeren.

Je kan de MFRC522 chip via SPI, UART of I2C laten werken. Wij kiezen voor **de SPI bus** in ons voorbeeld. Dit is een 10Mbps bus. De chip werkt op 3V3. Dus geen 5V aansluiten !!!

3. Features and benefits

- Highly integrated analog circuitry to demodulate and decode responses
- Buffered output drivers for connecting an antenna with the minimum number of external components
- Supports ISO/IEC 14443 A/MIFARE and NTAG
- Typical operating distance in Read/Write mode up to 50 mm depending on the antenna size and tuning
- Supports MF1xxS20, MF1xxS70 and MF1xxS50 encryption in Read/Write mode
- Supports ISO/IEC 14443 A higher transfer speed communication up to 848 kBd
- Supports MFIN/MFOUT
- Additional internal power supply to the smart card IC connected via MFIN/MFOUT
- Supported host interfaces
  - SPI up to 10 Mbit/s
  - I^2C-bus interface up to 400 kBd in Fast mode, up to 3400 kBd in High-speed mode
  - RS232 Serial UART up to 1228.8 kBd, with voltage levels dependant on pin voltage supply
- FIFO buffer handles 64 byte send and receive
- Flexible interrupt modes
- Hard reset with low power function
- Power-down by software mode
- Programmable timer
- Internal oscillator for connection to 27.12 MHz quartz crystal
- 2.5 V to 3.3 V power supply
- CRC coprocessor
- Programmable I/O pins
- Internal self-test

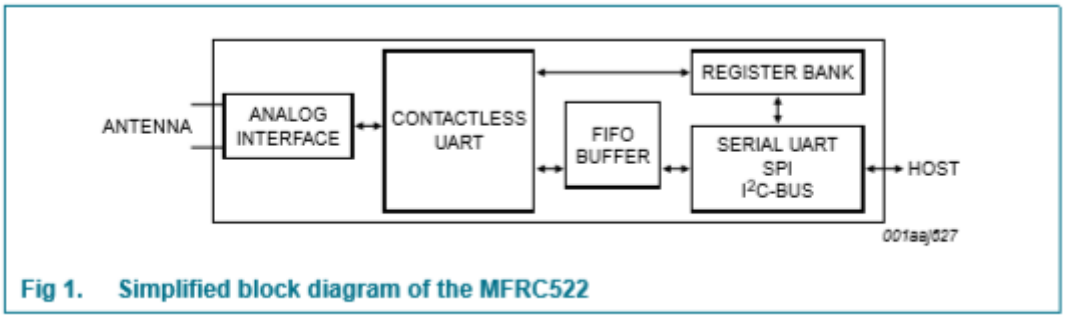


Fig 1. Simplified block diagram of the MFRC522

Het blokschema van de FRC522 toont aan hoe de antenne de data gaat uitlezen van de kaart en al deze data dan via de juiste interface zal doorgeven aan de Arduino / uc.

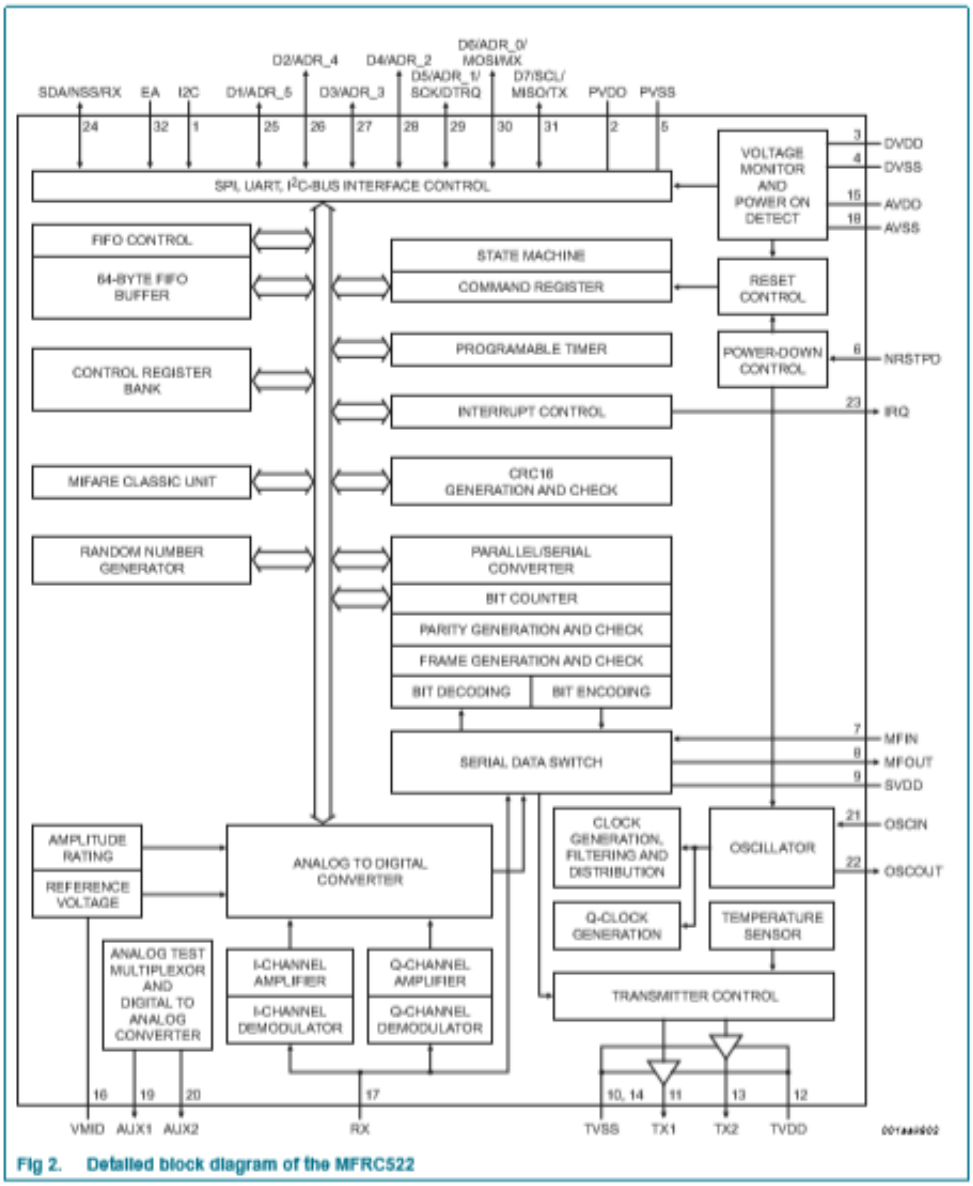


Fig 2. Detailed block diagram of the MFRC522

Het interne blokschema geeft aan dat dit best wel een complexe chip is.

The MFRC522 transmission module supports the Read/Write mode for ISO/IEC 14443 A/MIFARE using various transfer speeds and modulation protocols.

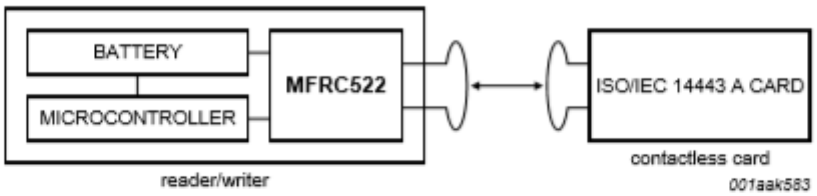


Fig 4. MFRC522 Read/Write mode

The physical level communication is shown in Figure 5.

(1) Reader to card 100 % ASK, Miller encoded, transfer speed 106 kBd to 848 kBd.

(2) Card to reader subcarrier load modulation, Manchester encoded or BPSK, transfer speed 106 kBd to 848 kBd.

Fig 5. ISO/IEC 14443 A/MIFARE Read/Write mode communication diagram

Bij RFID spreekt men van een ISO norm 14443. Dit wil zeggen dat de RFID communicatie tussen de FRC522 en de tags zal gebeuren volgends deze normen.

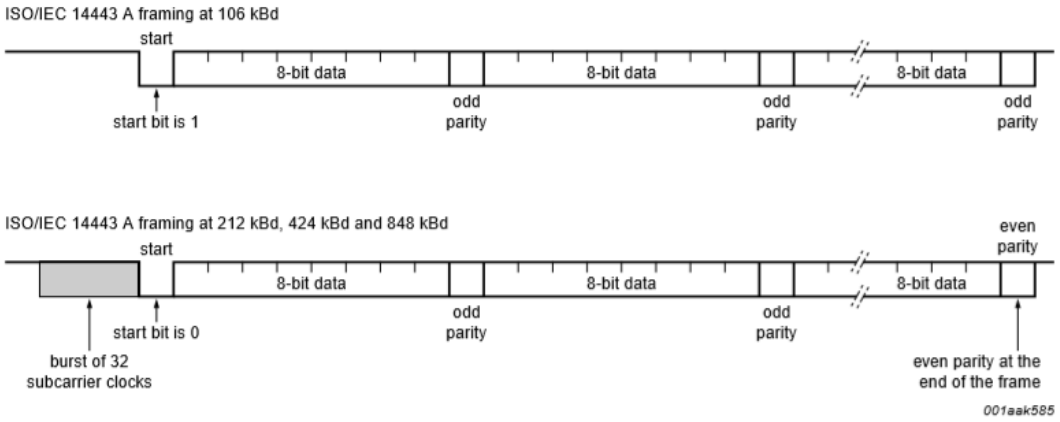


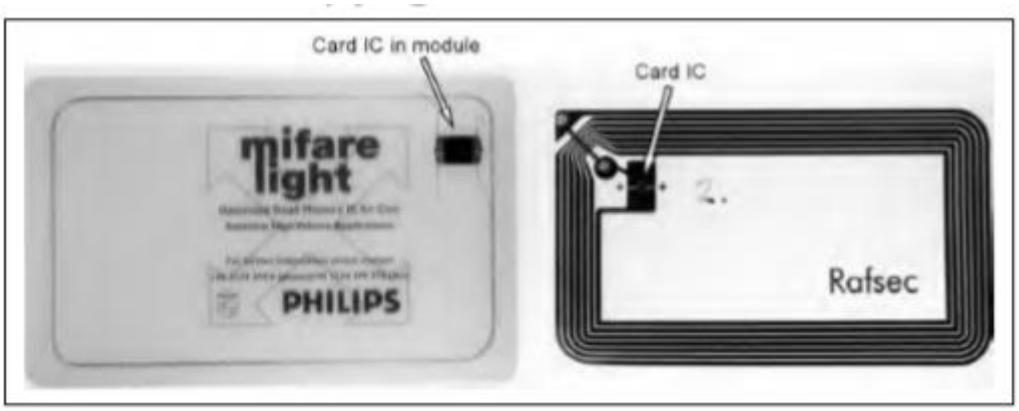
Fig 6. Data coding and framing according to ISO/IEC 14443 A

Deze figuur geeft aan wat er gaat passeren door de lucht als we de data proberen uit te lezen. Ook hier is een startbit aanwezig.

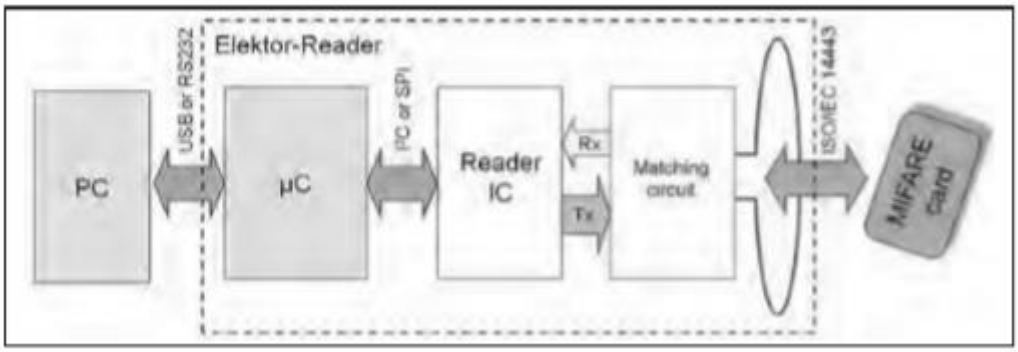
Table 1.1. *RFID frequencies*

Frequency	Transmission principle	Technology	Geographic area
125 kHz (LF)	magnetic coupling	passive	worldwide
13.56 MHz (HF)	magnetic coupling	passive	worldwide
433 MHz (UHF)	electromagnetic wave	active	worldwide
868 MHz (UHF)	electromagnetic wave	passive	Europe
915 MHz (UHF)	electromagnetic wave	passive	USA
2.45 GHz (SHF)	electromagnetic wave	passive / active	worldwide

Er zijn verschillende frequenties in omloop.

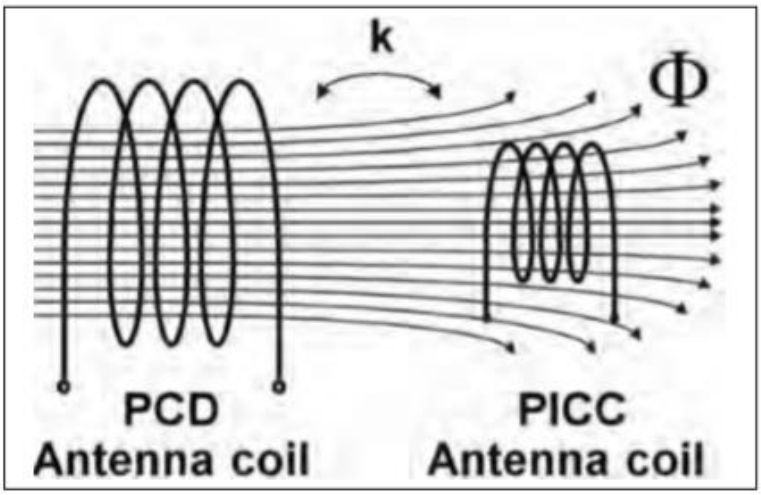


Op de tag zit een chip, antenne en een body waar alle inzit.



Dit is een eenvoudig blokschema van de grote delen die er zitten tussen de tag en de Arduino/PC

Figure 1.8.
Principle of energy transfer via antenna.



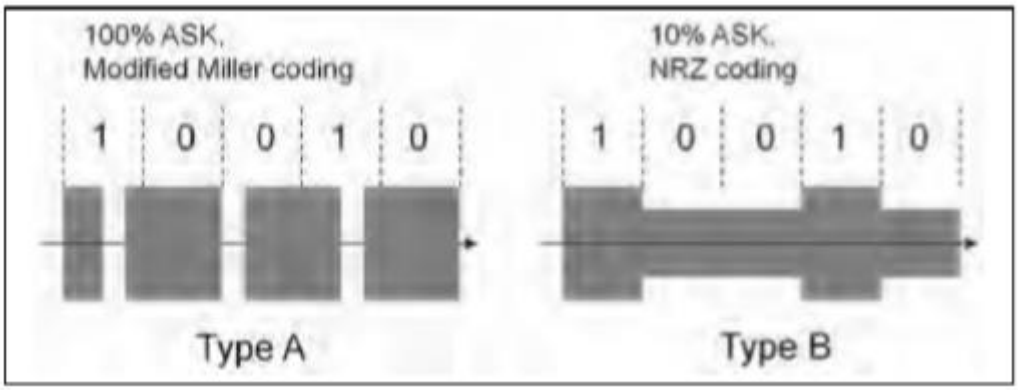
Deze foto geeft aan dat je een primaire (zenderchip) en secundaire winding (op de tag) hebt. Het magnetisch veld ontstaat tussen beide windingen en breng zo de magnetische energie over van de ene spoel naar de andere.

Omdat de k waarde (coupling magnitude) hier niet erg groot is moeten we de kaarten al kort tegen elkaar leggen om energie over te brengen (de efficiency is 10% tegen over een transfo met zijn 90%).

De max energie van het magnetisch veld moet binnen bepaalde grenzen blijven, om zo te voorkomen dat je andere toestellen stoort.

Merk op dat de tag energie moet ontvangen om de chip erop te kunnen laten werken.

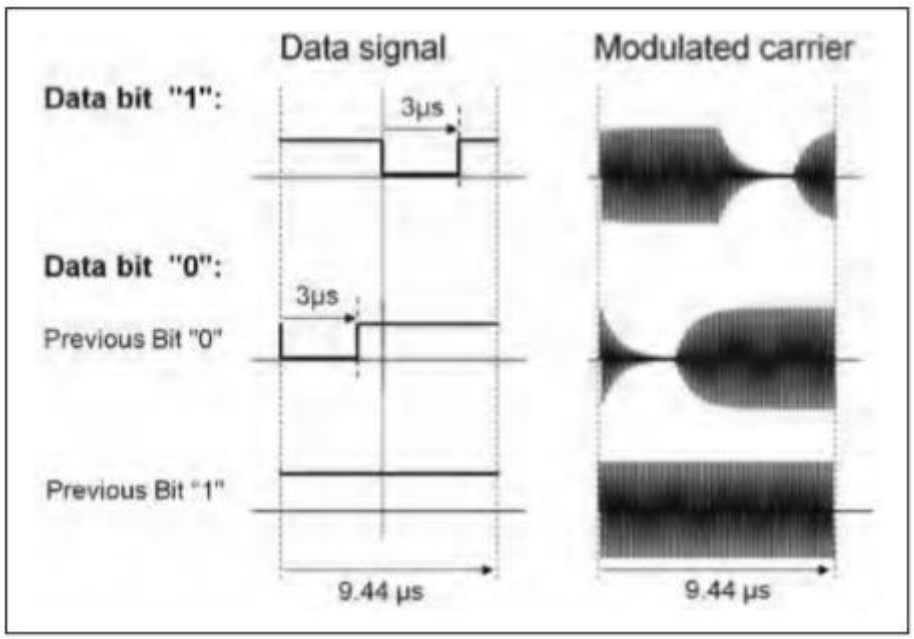
Hoe wordt nu de data overgedragen?



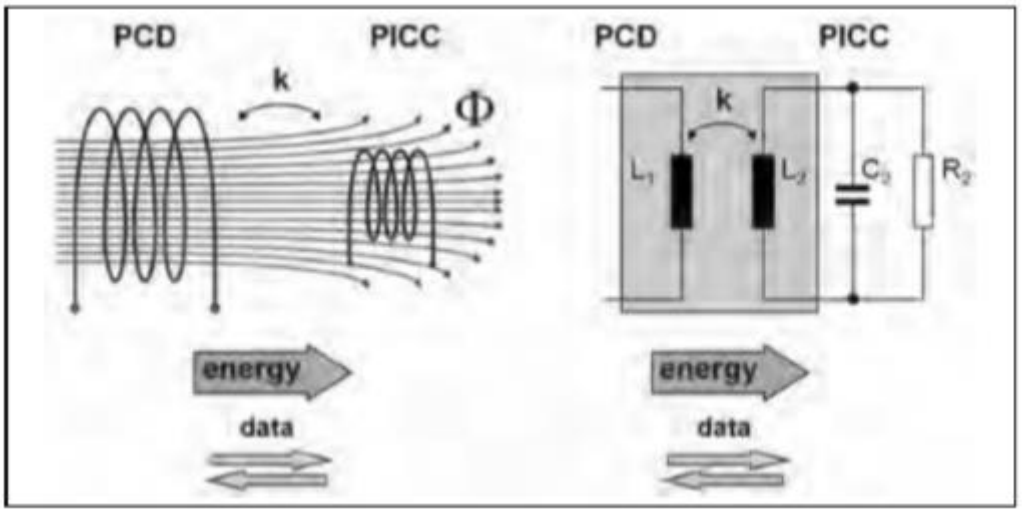
We gaan gebruik maken van **Amplitude modulatie. Bij ASK** is “0” een lage amplitude en “1” een hoge amplitude. Dit is de gemakkelijkste manier van werken. We zetten m.a.w. de energie wel of niet aan.

Voordeel is dat het eenvoudig en robust is. Nadeel is dat als de transmissie onderbroken is, de kaart (tag) geen energie meer krijgt.

Bij de modified Miller code is een “1” een hoog naar laag signaal en bij een “0” zijn er 2 mogelijkheden. Was de vorige bit een “0”, dan moet er even omlaag gegaan worden. Was de vorige bit een “1” dan blijft de data doorlopen. De tijd van omlaag gaan is zeer kort (3 us) want we hebben energie voor de kaart nodig om de chip te laten werken!



Modified Miller Codering



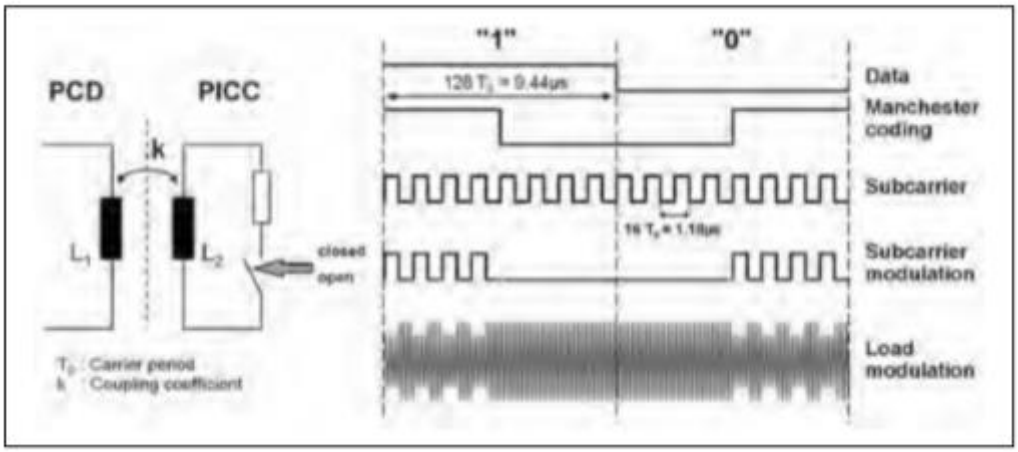
Deze tekening geeft links nog eens het transformator effect weer terwijl rechts te elektrische componenten zijn getoond.

L2 is de antenne van de tag terwijl C2 en R2 de ingangscondensator en weerstand van de antenne zijn.

In de bovenstaande tekening wordt de data van de Arduino zijde naar de tag zijde overgebracht.

Hoe stuur je dan data van de tag terug naar de Arduino zijde?

We gaan in dit geval gebruik maken van **load modulation.**



Door op de kaart een schakelaar open en toe te doen kunnen we een modulatie signaal voorzien op onze belasting. Bovenstaande figuur is weer met ASK gedaan. Het corresponderende signaal wordt hierdoor gestuurd van de tag naar de Arduino zijde en daarna gedecodeerd.

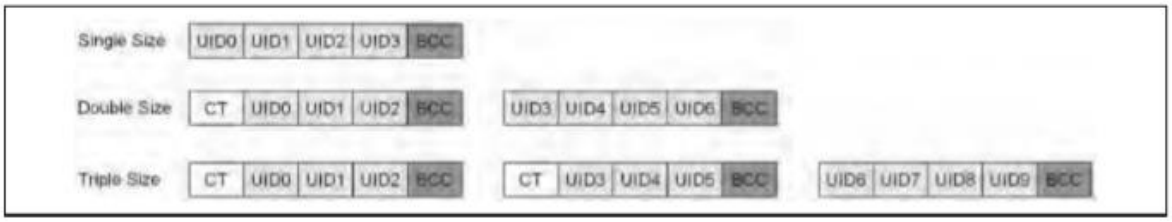


Fig. 2.3. *UID lengths for Type A: single size UID, double size UID and triple size UID.*

De data die verstuurd wordt bevat steeds als eerste een **UID (unique identifier).** Deze kan uit 1 of meerdere bytes bestaan. Deze UID zorgt ervoor dat elke kaart een unique adres heeft. In Arduino gaan we opzoek naar deze nummers om zo te beslissen of de kaart toegang geeft of niet.

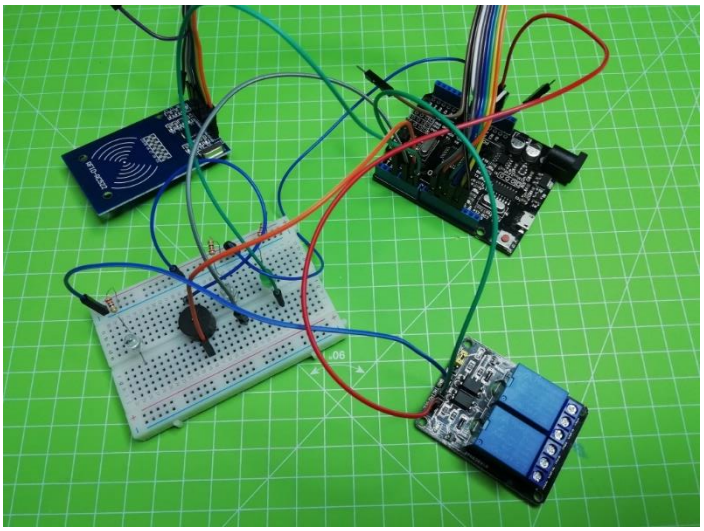
We gaan aan de slag 😊

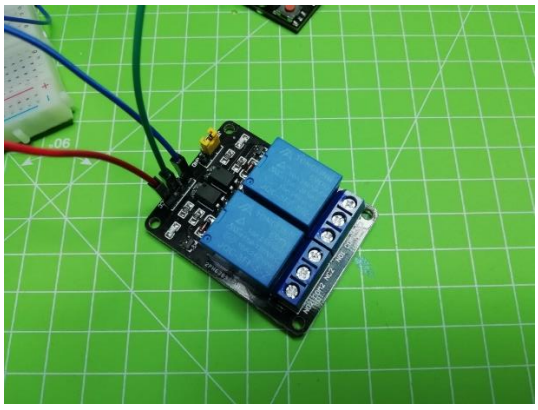
Library toevoegen!

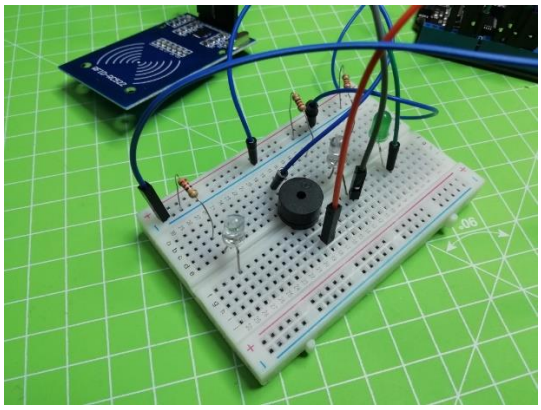
Zorg dat je de library **RFID-master.zip** (zie usbstick) toevoegt aan jouw Arduino omgeving via de gebruikelijke weg.

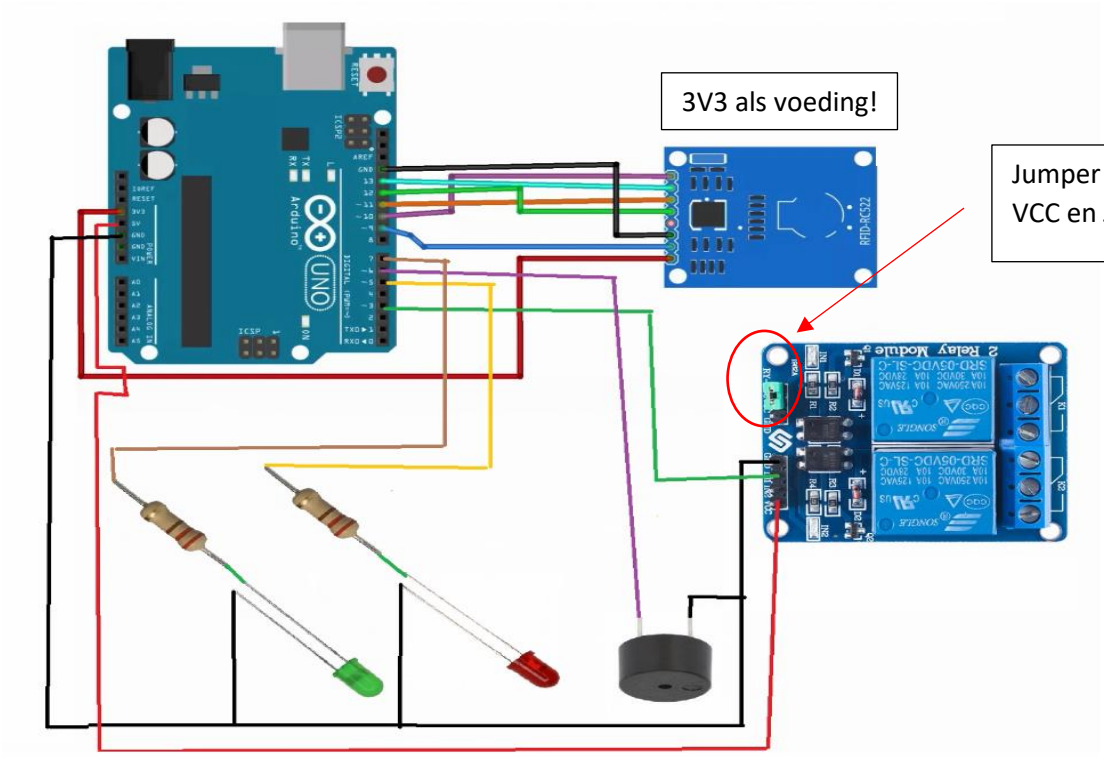
Merk op dat je geen enkel andere RFID library mag geïnstalleerd hebben. Anders krijg je foutmeldingen tijdens de compilatie. Check daarom zeker vooraf even of je in **C:\gebruiker\naam\documenten\arduino\library** geen andere RFID files meer staan!

Bouw nu de volgende schakeling:









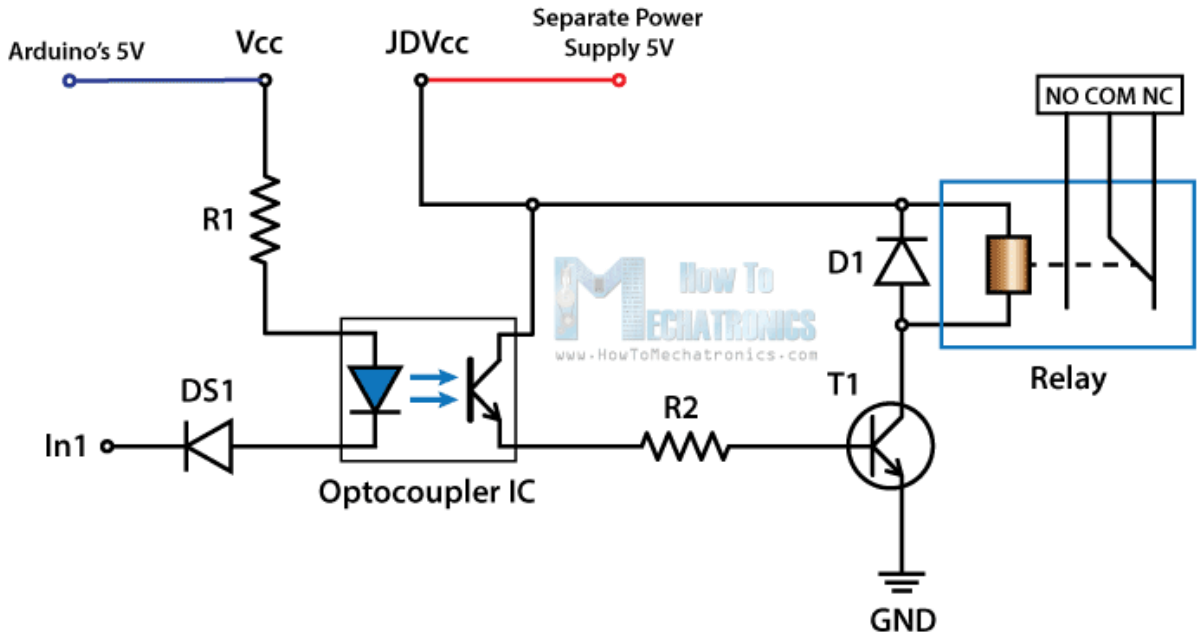
Aansluitingen schakeling:

RC522	**Arduino**
SDA = chip select	10
SCK	13
MOSI	11
MISO	12
IRQ	Unconnected
GND	GND
RST	9
3.3V	3.3V (zeker niet op de 5V aansluiten !!!)

Relais	Arduino
VCC	+5V
GND	GND
IN1	3
IN2	Unconnected
Jumper op relais: tussen VCC en JD-VCC	

Rode LED	5
Groene LED	7
Buzzer	6

De werking van de relais is hier uitgelegd:



Merk op dat de ingang van de relais aangestuurd wordt met een laag signaal. Dan pas gaat de LED branden in de optokoppelaar. Als deze brandt gaat de basis van T1 aangestuurd worden. Hierdoor komt er stroom Ic in de collector en staat de volle 5V over de spoel van de relais. Nu kan deze sluiten.

Merk op dat er over de spoel van de relais een diode moet staan in sperzin (D1) om als vrijloopdiode te dienen en de tegen-EMK op te vangen als de spoel niet bekrachtigd wordt via de transistor.

DS1 dient om te voorkomen dat de optokoppelaar fout wordt aangesloten kwa polariteit.

Hoe ziet de code eruit?

RFID_flipflop

```
#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN 10
#define RST_PIN 9
#define RELAY 3
#define LEDR 5
#define LEDG 7
#define BUZZER 6

MFRC522 mfrc522(SS_PIN, RST_PIN);   // Create MFRC522 instance.

void setup()
{
  pinMode(RELAY,OUTPUT);
  pinMode(LEDR,OUTPUT);
  pinMode(LEDG,OUTPUT);
  pinMode(BUZZER,OUTPUT);
  Serial.begin(9600);   // Initiate a serial communication
  SPI.begin();      // Initiate  SPI bus
  mfrc522.PCD_Init();   // Initiate MFRC522
  Serial.println("Approximate your card to the reader...");
  Serial.println();

  digitalWrite(RELAY,HIGH);//RELAY uitzetten in het begin
      //LAAG ACTIEF
     // zet jumper over VCC EN JDVCC om te kunnen schakelen met Arduino

}
```

Eerst de nodige libraries inladen. Niet alleen de RFID maar ook de SPI lib doet mee (dit kon je al raden aan de MOSI en MISO aansluitingen).

Meest nieuwe in deze code is dat we een object mfrc522 aanmaken en hier onze MFRC522 lib header file aan koppelen.

Daarna moeten we de MFRC522 communicatie opzetten, net als de SPI en serial monitor communicatie.

```
void loop()
{
  // Look for new cards
  if ( ! mfrc522.PICC_IsNewCardPresent())
  {
    return;
  }
  // Select one of the cards
  if ( ! mfrc522.PICC_ReadCardSerial())
  {
    return;
  }
  //Show UID on serial monitor
  Serial.print("UID tag :");
  String content= "";
  byte letter;
  for (byte i = 0; i < mfrc522.uid.size; i++)
  {
     Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
     Serial.print(mfrc522.uid.uidByte[i], HEX);
     content.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
     content.concat(String(mfrc522.uid.uidByte[i], HEX));
  }
  Serial.println();
  Serial.print("Message : ");
  content.toUpperCase();
```

We starten met het nakijken of er een nieuwe kaart is gelezen door de mfrc522 module.

Er wordt heel wat info uit de kaart gehaald. Wat we vooral willen weten is de **UID van de tag.**

Aan de hand van deze UID gaan we nu beslissen of het de juiste of foute kaart is die wordt gescand.

```
if (content.substring(1) == "C0 FE 8D 25") //change here the UID of the card/cards
{
  Serial.println("Authorized access");
  digitalWrite(RELAY,LOW);//LAAG actief

  digitalWrite(LEDG,HIGH);

  tone(BUZZER,1000);
  Serial.println();
  delay(3000);
  digitalWrite(RELAY,HIGH);

  digitalWrite(LEDG,LOW);
  noTone(BUZZER);
}

else   {
  Serial.println(" Access denied");
  digitalWrite(LEDR,HIGH);
   for (int i=0;i < 1000;i+=20){
      tone(BUZZER,i);
      delay(50);
  }

  digitalWrite(LEDR,LOW);

  noTone(BUZZER);

 }
}
```

```
Approximate your card to the reader...

UID tag : C0 FE 8D 25
Message : Authorized access

UID tag : F2 9D 5A D3
Message :  Access denied
UID tag : C0 FE 8D 25
Message : Authorized access
```

Mogelijk moet je de UID aanpassen zodat jouw kaart overeenkomt met de juiste code.

Daarna kan je de leds, buzzer en relais aan het werk zetten.

Uitdaging:

- Maak een **telsysteem** voor aanwezigen. Telkens er een keer gescand wordt moet op een LCD scherm of 7-segment een getal verhogen. Voorzie ook een knop om de waarde te resetten.

8 Via WIFI een ESP8266 ESP-01 aansturen (IOT project, extra)

Welke technologie en protocol er zeker in deze cursus niet mocht ontbreken is het sturen van data over een **WIFI verbinding.** Dingen zoals LEDs en relais kunnen aansturen via een website van op afstand (of via de GSM) dat zou toch fantastisch zijn. We noemen **dit IOT apparaten (internet of things).** Steeds meer toestellen worden via het net met elkaar verbonden en gegevens worden verzameld en uitgewisseld..

De ESP8266 is zo'n chip die we makkelijk kunnen inzetten om aan IOT te doen.

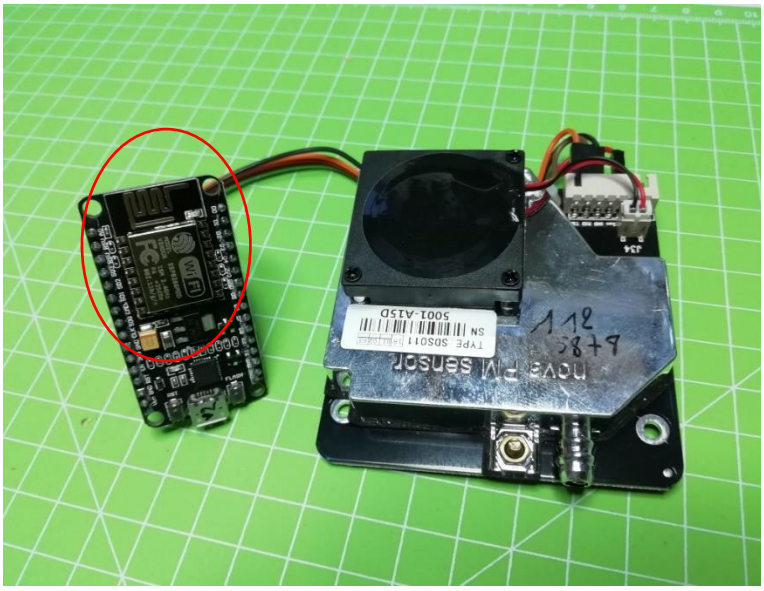
ESP8266 ESP-01

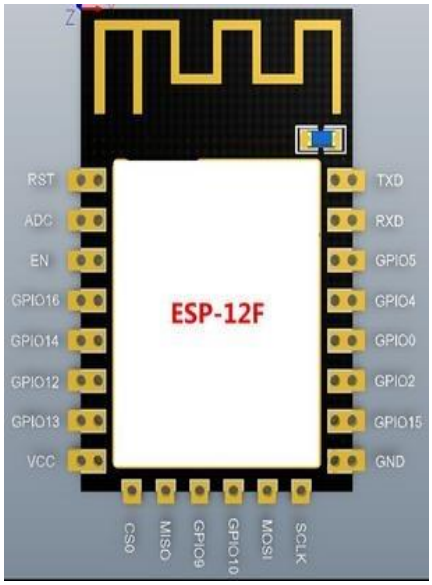
De inspiratie is gevonden op https://www.instructables.com/id/ESP8266-WiFi-Module-for-Dummies/

Merk op dat er ondertussen een hele hoop ESP bordjes bestaan. De ESP-01 is de meest eenvoudige met de minst aantal GPIO pinnen. We kunnen in feite **maar 2 pinnen gebruiken om I/O** aan te sturen. Net genoeg voor een LED, knop of relais. Maar ook daar kan je al wat mee aanvangen.

https://www.esp8266.com/wiki/doku.php?id=esp8266-module-family

Tegenwoordig is het populair om het fijnstof te meten en dit dan door te geven naar een website. Dit IOT device maakt gebruik van een **ESP-12-E/Q versie**.





Meer info over de fijnstofsensormetingen vind je op https://luftdaten.info/nl/startpagina/

Wat is de ESP voor iets?

De ESP is zoals reeds gezegd een SOC van Espressif Systems. Deze chip bevat een eigen processor, bevat eigen geheugen en modules voor bluetooth, wifi, UART en dergelijke.

Table 1.1 shows comparison of the basic features of ESP32 and ESP8266 processors.

Specifications	ESP32	ESP8266
CPU	32-bit Xtensa L106 double-core	32-bit Xtensa LX6 dual-core
Operating frequency	160 MHz	80 MHz
Bluetooth	Bluetooth 4.2	None
Wi-Fi	Yes (HT40)	Yes (HT20)
SRAM	512 KB	160 KB
GPIOs	36	17
Hardware PWM	1	None
Software PWM	16	8
SPI/I2C/I2S/UART	4/2/2/2	2/1/2/2
CAN	1	None
ADC	12-bit	10-bit
Touch sensor	10	None
Temperature sensor	1	None
Ethernet MAC interface	1	None

Table 1. Comparison of ESP32 and ESP8266

De ESP66 heeft nog een groter broer, de ESP32.

The functional diagram of ESP8266EX is shown as in Figure 3-1.

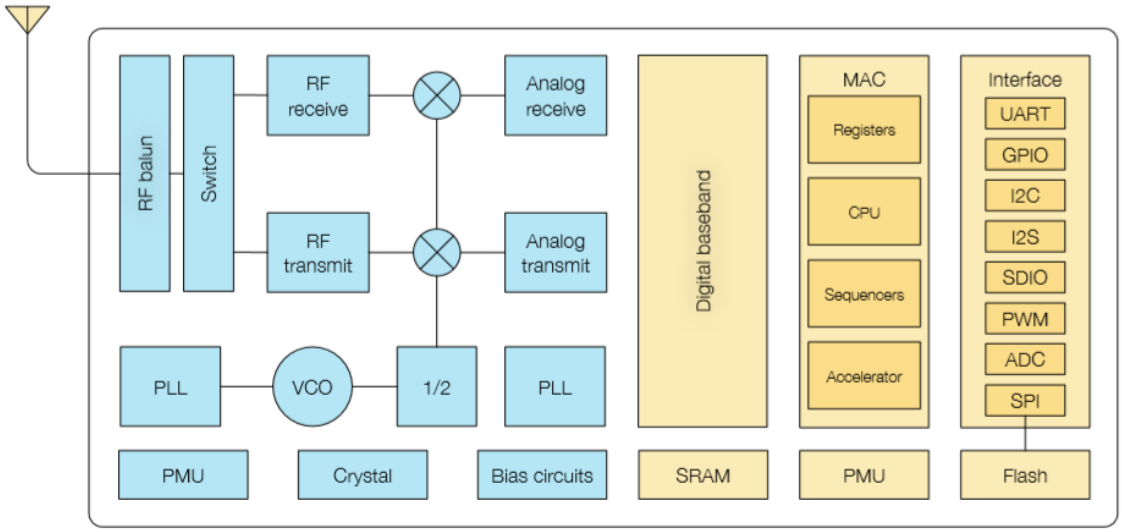
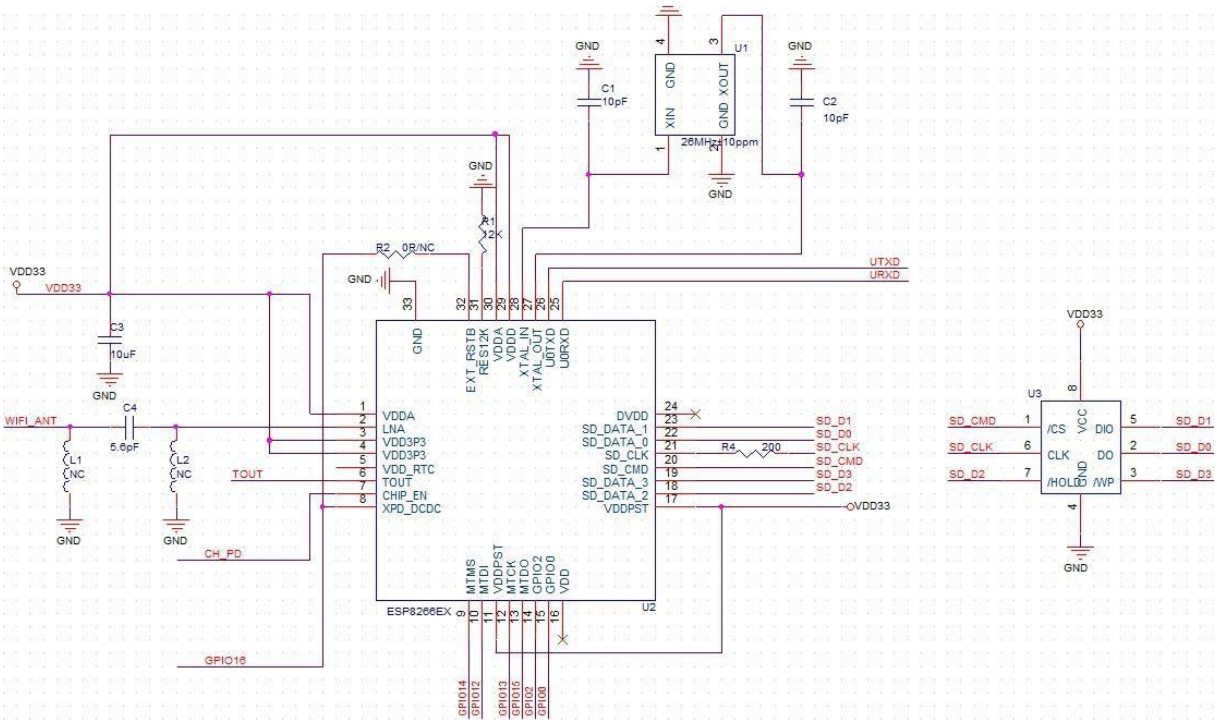
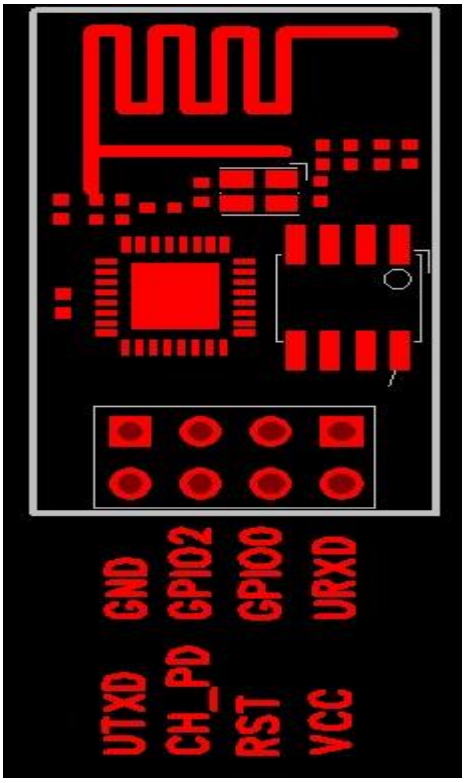
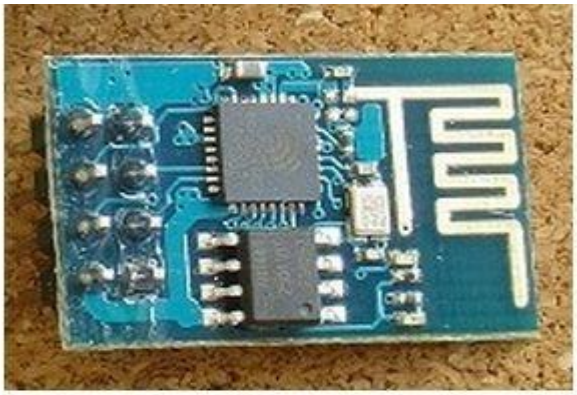


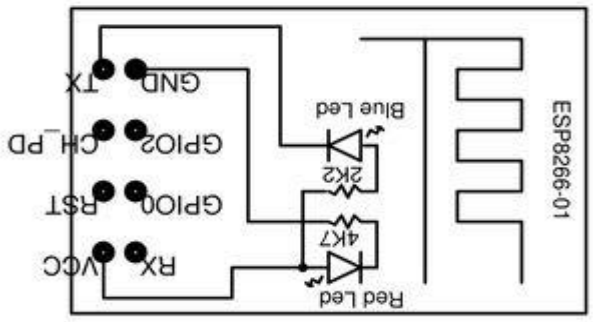
Figure 3-1. Functional Block Diagram



Dit is het schema van het ESP01 bordje.



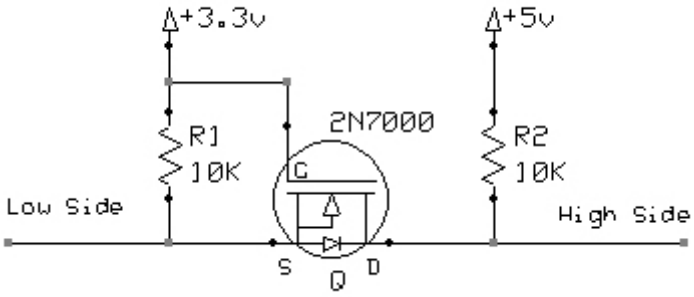


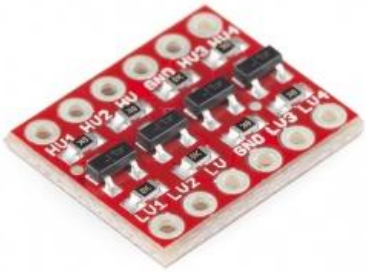


Dit is de PCB met aansluitingen van de ESP01

In dit hoofdstuk gaan we de Arduino IDE omgeving enkel gebruiken om de ESP8266 SOC (System on chip) te programmeren. We kunnen namelijk de Arduino niet zelf inzetten omdat deze op 5V werkt terwijl het ESP bordje maar op 3V3 werkt.

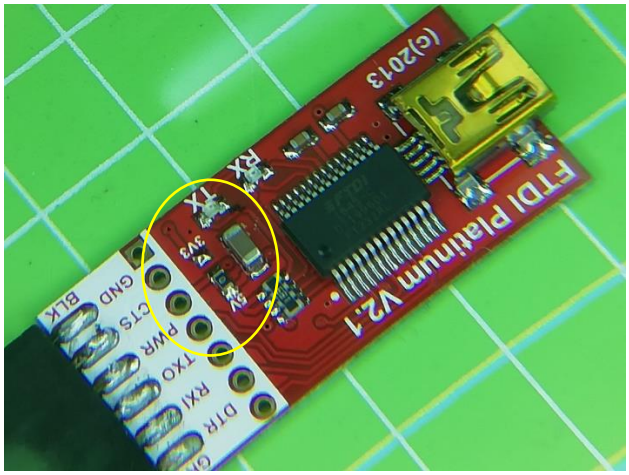
Uiteraard kan je gebruik maken van een **level shifter** van 5V naar 3V3. Dan kan je wel met de Arduino UNO aan de slag.





http://www.hobbytronics.co.uk/mosfet-voltage-level-converter

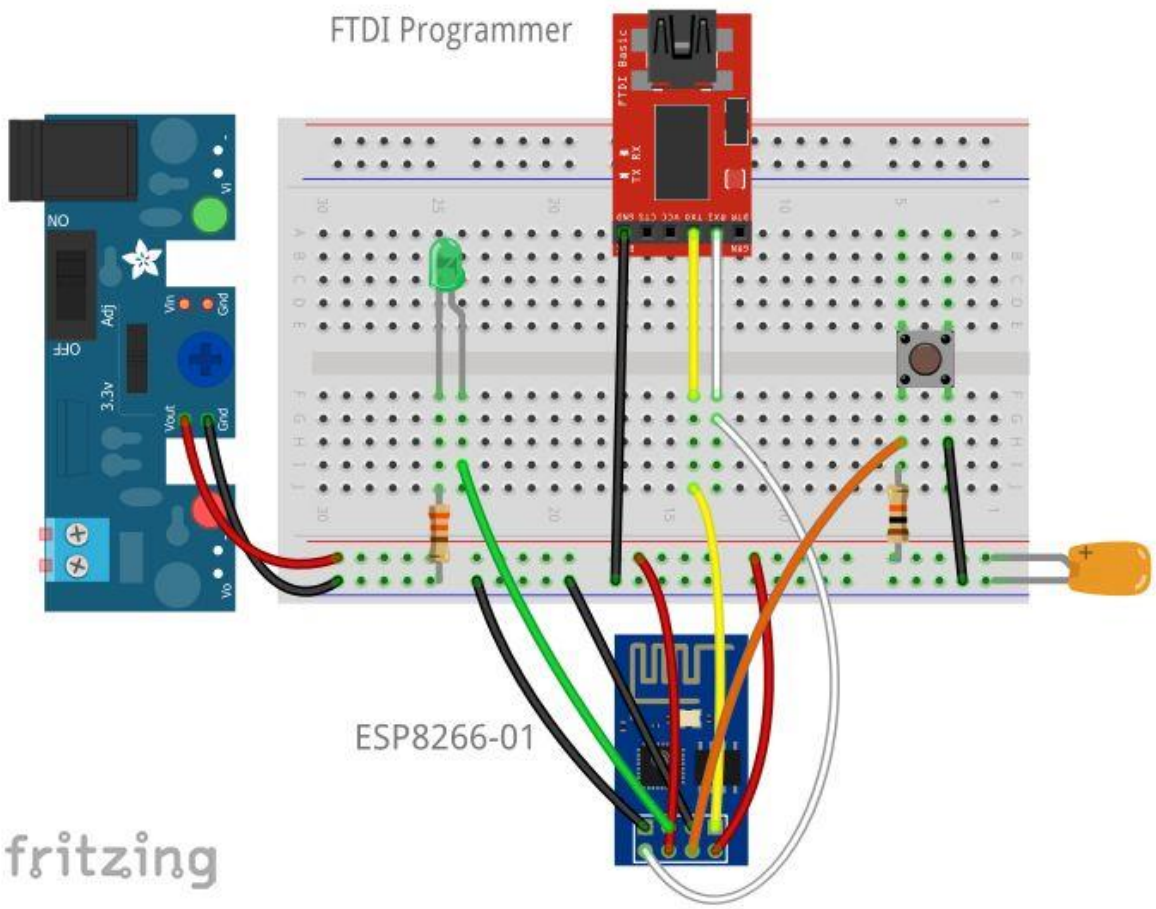
Voor het programmeren van de ESP8266 gaan we echter hier gebruik maken van een **USB naar UART** omzetter die ineens ingesteld staat op 3V3. FTDI heeft zo'n omzetter waar je kan **kiezen** tussen 5V of **3V3.**



Vergeet niet van de **0 ohm weerstand te solderen op de 3V3 verbinding!**

Mogelijk is er een jumper voorzien waarbij je de spanning kan kiezen.

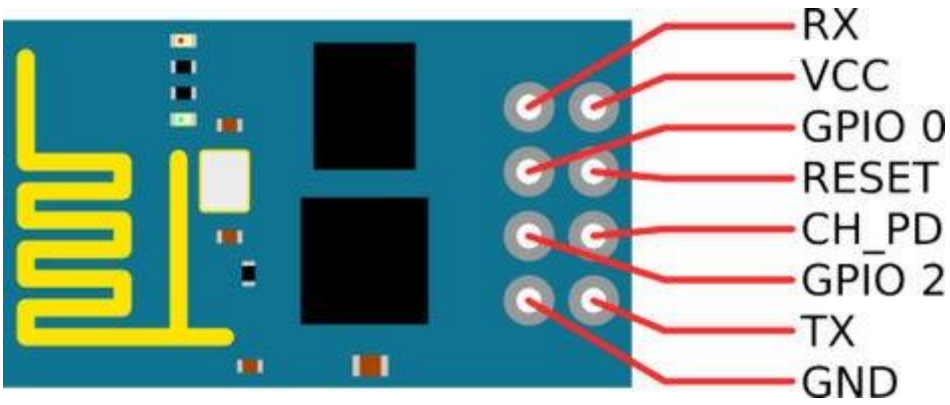
Bouw nu de volgende schakeling:



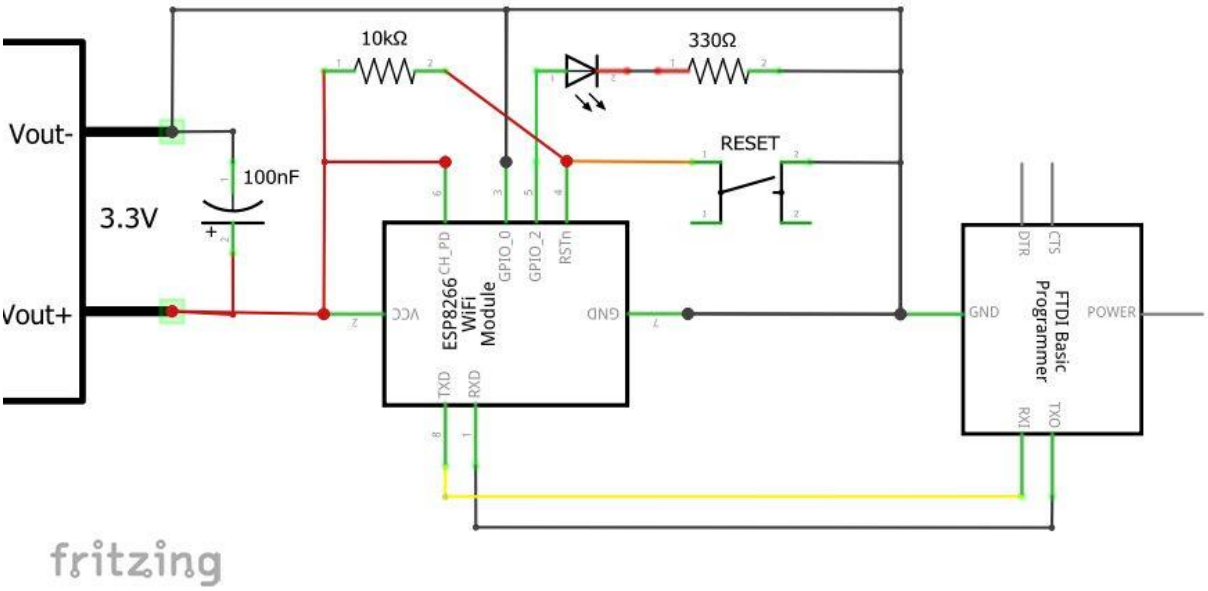
ESP8266	
GND	GND
TX	RX van USB -> UART converter (3V3)
RX	TX van USB -> UART converter (3V3)
GPIO0	Aan GND tijdens programmatie, anders loslaten of als GPIO gebruiken
GPIO2	Via 220 weerstand aan rode LED (kan ook een relais aansturen)
CH_PD	3V3
RESET	Via 10K weerstand en een knop aan de 3V3
VCC	3V3 (dus niet de 5V van de Arduino) Zorg dat dit een externe voeding is! Het bordje is gevoelig voorstoringen tijdens programmatie!

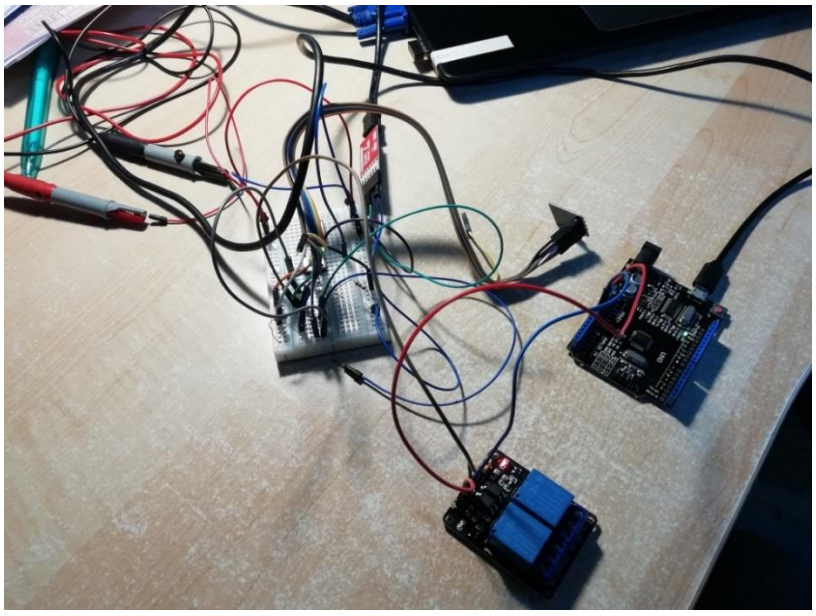
Plaats 100n over de GND en 3V3 om HF storingen af te leiden naar de GND.

Als je een relais op 5V gebruikt kan je de +5v voeding aftappen van een Arduino, maar de GND hangt aan de andere GND's. De GPIO0 stuurpin van de ESP hangt echter nog altijd aan de relais ingang.



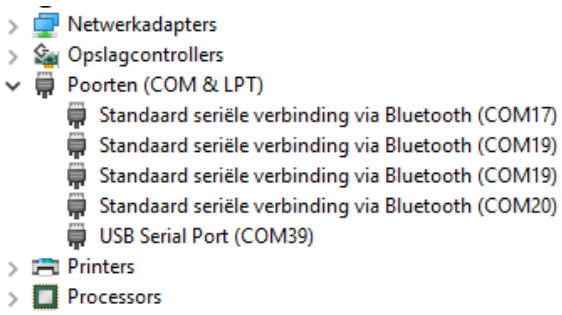
Overzicht ESP8266 aansluitingen.





Nadat de schakeling is gebouw sluiten we ESP aan op de PC. Nu brandt de rode LED op de ESP constant.

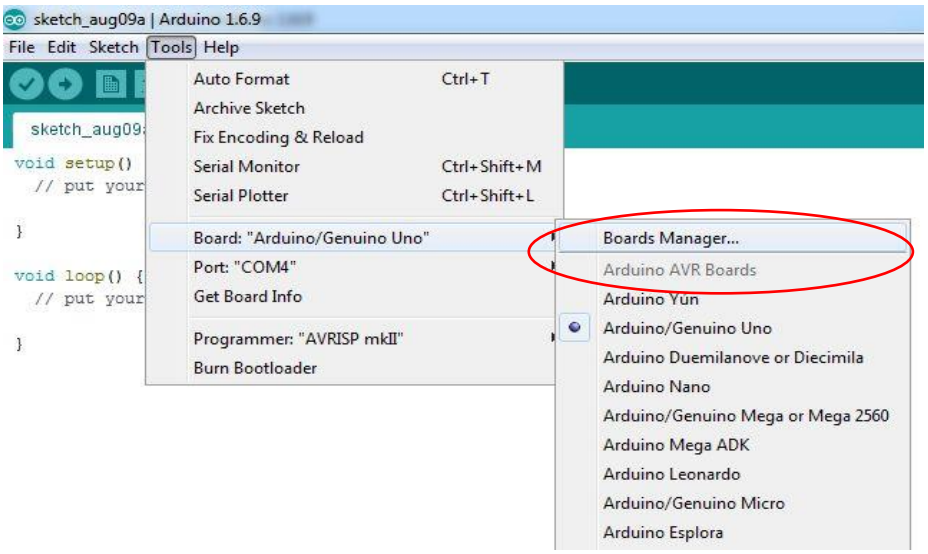
Onderzoek via de device manager op welke COM poort we gaan werken.



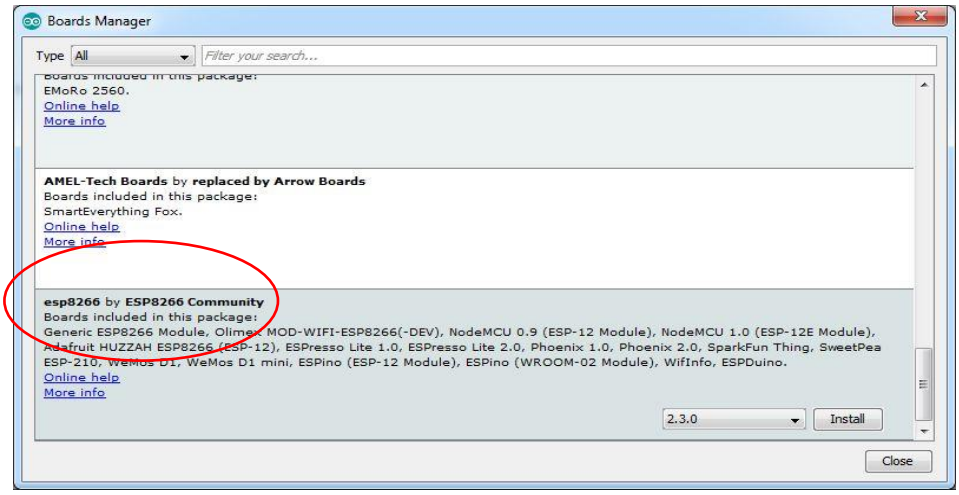
In dit geval op poort COM39

Start nu de Arduino IDE omgeving.

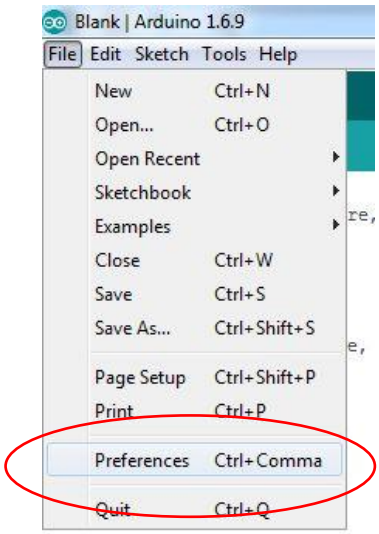
Ga naar tools -> board -> board manager



Zoek naar de ESP8266 en installeer de laatste versie van de ESP8266 community.

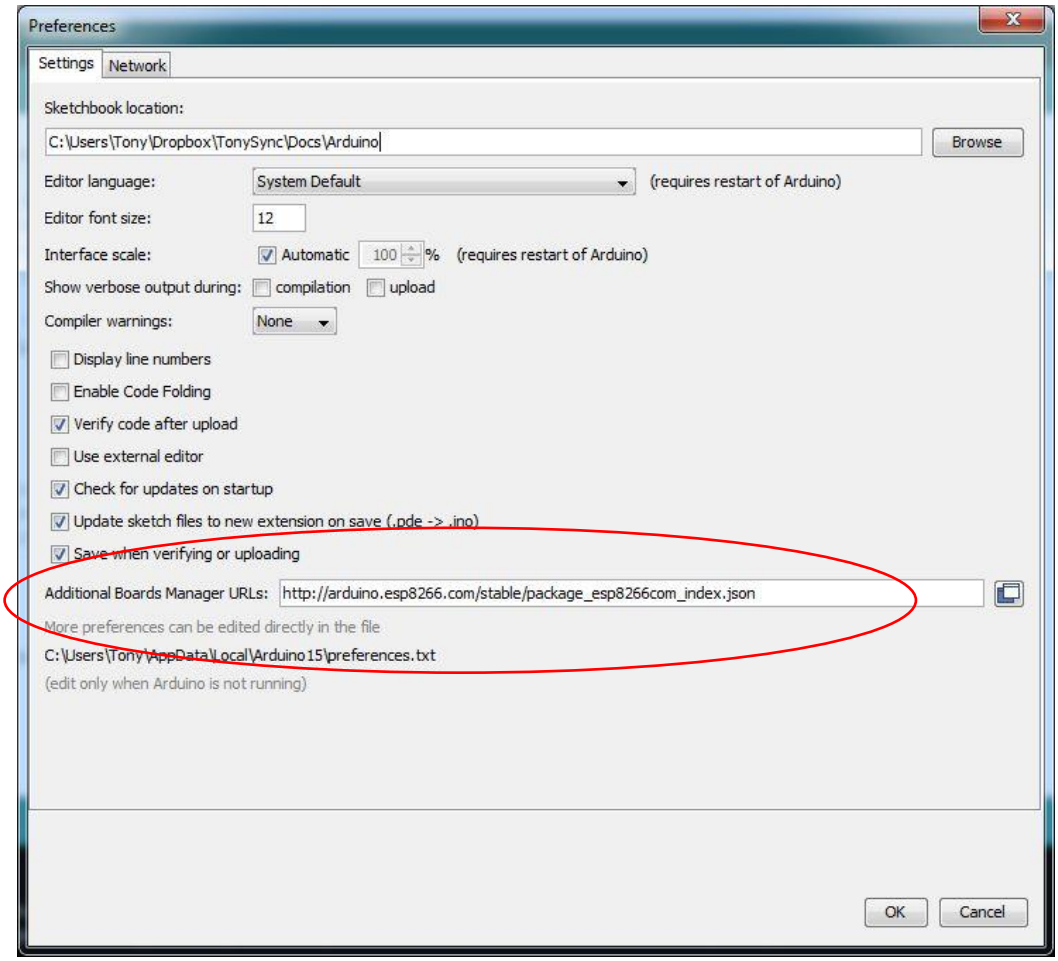


Ga nu naar file -> “Preference” in de Arduino IDE

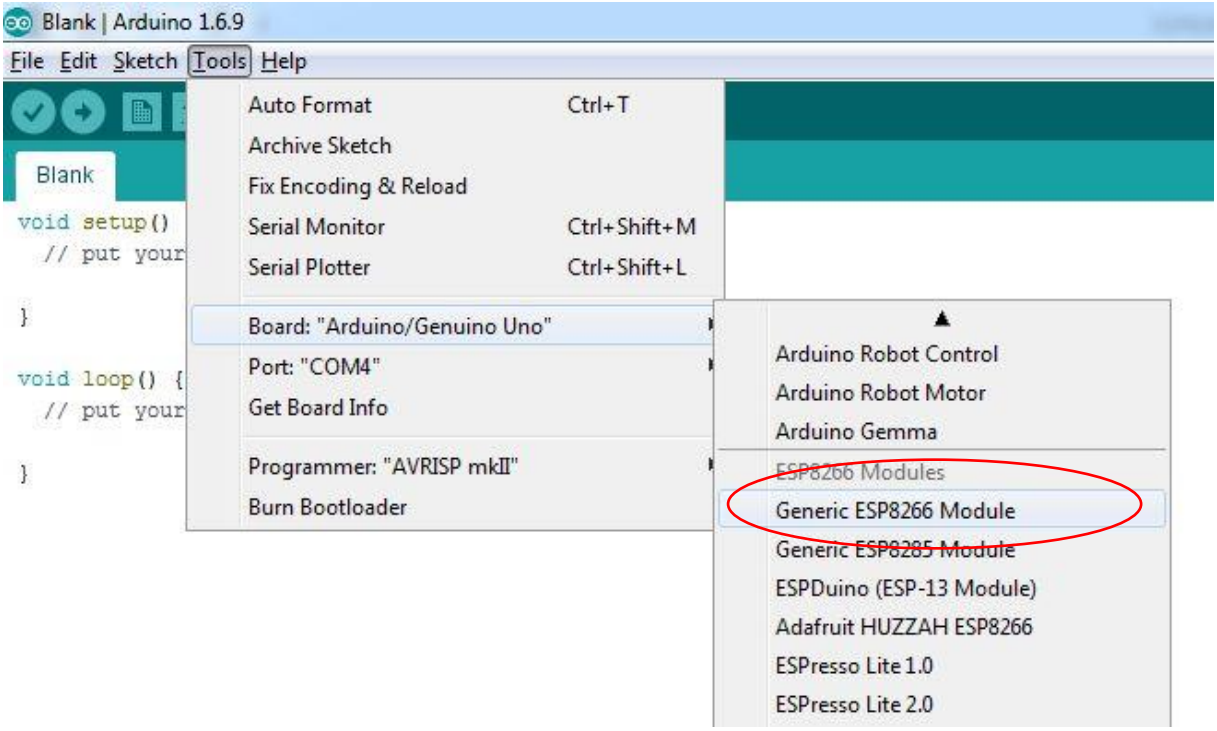


Voeg in de Additional Boards Manager URLs de volgende URL toe:

http://arduino.esp8266.com/stable/package_esp8266com_index.json

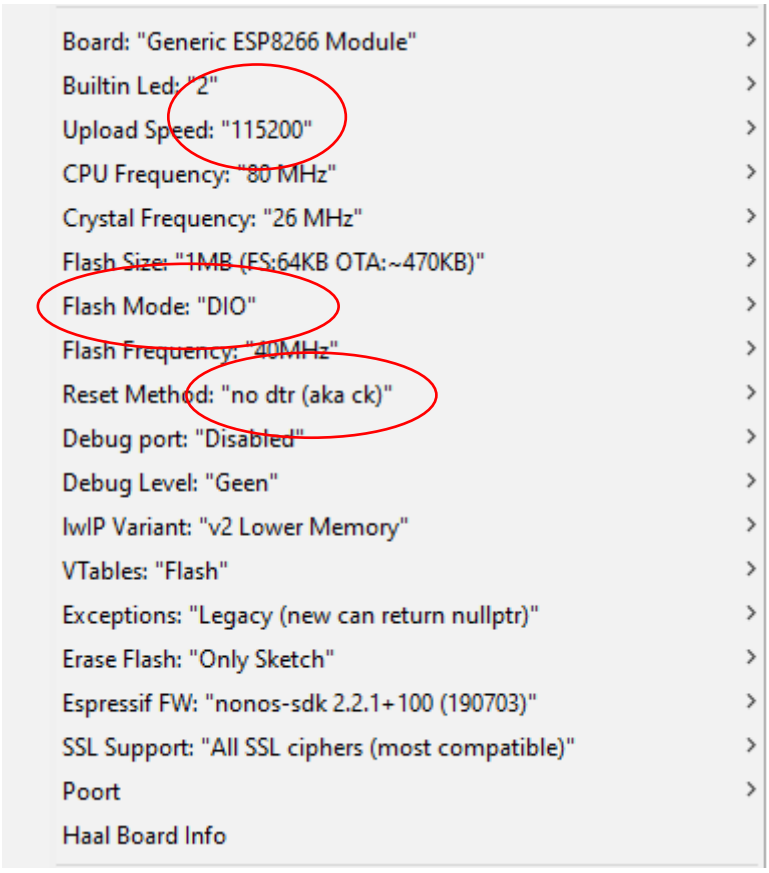


Kies nu het juiste Board via tools -> board -> **Generic ESP8266 Module**



Stel nu de instellingen in van de ESP8266 module zoals aangegeven in onderstaande figuur.

Vergeet ook niet van de COM poort op de juiste nummer te zetten (hier nr 39)

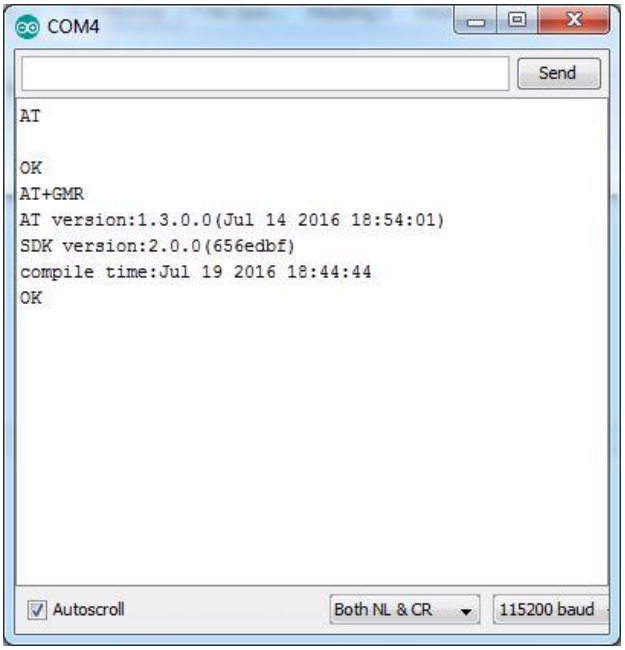


Voordat je de eerste keer de ESP8266 wil gaan programmeren met je eigen code, zit er nog originele firmware in van de firma. Eens je een programma hebt geprogrammeerd in de ESP is dit dus weg.

Merk op dat je eerst de GPIO0 moet los doen van de GND en eens op reset drukken om de ESP in de gewone werk mode krijgen.

Wat kan je met de originele versie doen?

Je kan met AT commando's info halen uit de ESP.



Ga naar de serial monitor en zet deze op 115200 baud.

Stel ook de "both NL en CR" in om alle data netjes op het scherm te krijgen.

Type "AT" en je krijgt normaliter een "OK" terug. Dan weet je dat alles goed is aangesloten.

Type "AT+GMR". Dan krijg je de versie nummer van de firmware en dergelijke.

Extra:

Wil je toch nog de ESP terug van de originele firmware voorzien, dan kan je de volgende link en info volgen:

https://www.instructables.com/id/The-Simple-Guide-to-Flashing-Your-ESP8266-Firmware/

Meer info over AT commando's kan je hier vinden:

https://www.itead.cc/wiki/ESP8266_Serial_WIFI_Module#AT_Commands

Haal nu de volgende ESP8266_LED_Control code binnen in de Arduino IDE omgeving.

ESP8266_LED_Controlv1

```
/*
Dit is code om een ESP8266 zonder Arduino aan te sturen
Je kan via de aangegeven IP adres in de serial monitor de GPIO2 aansturen
Je kan bij de ESP 1 geen analoge ingangen inlezen (ondanks dat de pin op de chip zit
gpio0 kan ook gebruikt worden nadat hij voor de download mode is gebruikt
*/
//eerst code downloaden
//dan GPIO0 draad losmaken van GND dan download (want anders blijft de esp8266 in program mode)
//druk op reset om de SOC opnieuw te starten
//zorg dat de voeding zeker op 500mA staat


// Include the ESP8266 Library. This library is automatically provided by the ESP8266 Board Manager
#include <ESP8266WiFi.h>

String codeVersion = "FRANK EDULAB v0.1";

// WiFi Router Login - change these to your router settings
const char* SSID = "type hier jouw SSID";//case sensitive naam netwerk!
const char* password = "type hier jouw paswoord"; // code van thuisnetwerk

// Setup GPIO2
int pinGPIO2 = 2; //To control LED
int ledStatus = 0; //0=off,1=on,2=dimmed

// Create the ESP Web Server on port 80
WiFiServer WebServer(80);
// Web Client
WiFiClient client;
```

In dit deel van de code roep je de ESP8266Wifi library aan en zet je jouw SSID en password klaar om contact te maken met het Wifi netwerk.

Je stelt ook de weinige GPIO pins in, vb GPIO2.

Je start hier met enkele eenvoudige commando’s jouw webserver op op port 80.

```
void setup() {
  Serial.begin(115200);
  delay(10);
  Serial.println();
  Serial.println();
  Serial.println(codeVersion);

  // Setup the GPIO2 LED Pin - this demo also uses PWM to dim the LED.
  pinMode(pinGPIO2, OUTPUT);
  digitalWrite(pinGPIO2, LOW);
  ledStatus = 0;

  // Connect to WiFi network
  Serial.println();
  WiFi.disconnect();
  WiFi.mode(WIFI_STA);
  Serial.print("Connecting to ");
  Serial.println(SSID);
  WiFi.begin(SSID, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("Connected to WiFi");

  // Start the Web Server
  WebServer.begin();
  Serial.println("Web Server started");



  // Print the IP address
  Serial.print("You can connect to the ESP8266 at this URL: ");
  Serial.print("http://");
  Serial.print(WiFi.localIP());
  Serial.println("/");
}
```

In de setup wordt de serial monitor opgestart en de GPIO pin verder ingesteld.

Daarna probeert de ESP een connectie te maken met het Wifinetwerk. Hij gebruikt hiervoor jouw SSID en password.

Tenslotte wordt de URL nummer uitgeprint. Dit is de IP locatie in de browser waarlangs we onze kleine ESP server gaan aansturen.

```
void loop() {
  // Check if a user has connected
  client = WebServer.available();
  if (!client) {//restart loop
    return;
  }

  // Wait until the user sends some data
  Serial.println("New User");
  while (!client.available()) {
    delay(1);
  }

  // Read the first line of the request
  String request = client.readStringUntil('\r\n');
  Serial.println(request);
  client.flush();
```

Er wordt eerst getest of de gebruiker via het IP adres contact heeft gemaakt met de ESP server en of er aan vraag is gesteld van de user (vb we hebben gedrukt op een link).

De vraag wordt afgeprint op het scherm (vb “/LED=ON”)

```
  // Process the request:
  if (request.indexOf("/LED=ON") != -1) {
    analogWrite(pinGPIO2, 1023);
    ledStatus = 1;
  }
  if (request.indexOf("/LED=OFF") != -1) {
    analogWrite(pinGPIO2, 0);
    ledStatus = 0;
  }
  if (request.indexOf("/LED=DIM") != -1) {
    analogWrite(pinGPIO2, 512);
    ledStatus = 2;
  }
```

Afhankelijk van de inhoud van request wordt nu de GPIO2 pin aangestuurd (aan, uit of PWM).

```
  // Return the response
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/html; charset=UTF-8");
  client.println("");
  client.println("<!DOCTYPE HTML>");
  client.println("<html>");
  client.println("<head>");
  client.println("<title>ESP8266 Demo</title>");
  client.println("</head>");
  client.println("<body>");
  client.println("<a href=\"/\">Refresh Status</a>");
  client.println("</br></br>");


  //check the LED status
  if (ledStatus == 0) {
    client.print("LED is Off</br>");
    client.println("Turn the LED <a href=\"/LED=ON\">ON</a></br>");
    client.println("Set LED to <a href=\"/LED=DIM\">DIM</a></br>");
  } else if (ledStatus == 1) {
    client.print("LED is On</br>");
    client.println("Turn the LED <a href=\"/LED=OFF\">OFF</a></br>");
    client.println("Set LED to <a href=\"/LED=DIM\">DIM</a></br>");
  } else if (ledStatus == 2) {
    client.print("LED is Dimmed</br>");
    client.println("Turn the LED <a href=\"/LED=OFF\">OFF</a></br>");
    client.println("Turn the LED <a href=\"/LED=ON\">ON</a></br>");
  }


  client.println("</br>");
  client.println("<a href=\"http://www.instructables.com/id/ESP8266-WiFi-Module-for-Dummies/\" target=\"_blank\"
  client.println("<a href=\"http://www.instructables.com/id/The-Simple-Guide-to-Flashing-Your-ESP8266-Firmware/\

  client.println("</br>");
  client.println("</body>");
  client.println("</html>");

  delay(1);
  Serial.println("User disconnected");
  Serial.println("");

}
```

Deze info is nodig om de website in html code op te bouwen en de status van de links aan te passen.

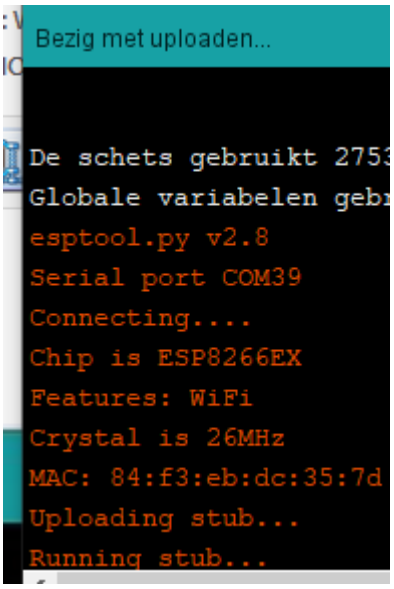
Daarna wordt de connectie onderbroken.

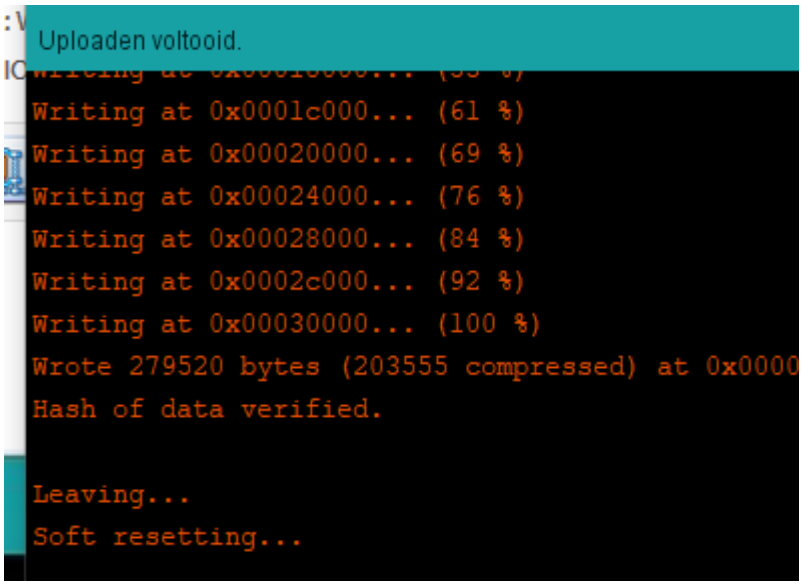
Kijk na of de 3V3 voeding aanstaat en de GPIO0 terug aan de GND zit om de ESP in programmeermode te krijgen. Staat de voeding op 500mA ?

Druk ook nog eens op de rest knop opdat de ESP ook in de programmeermode start.

Upload nu pas de bovenstaande code (SSID en paswoord aangepast op jouw netwerk?)

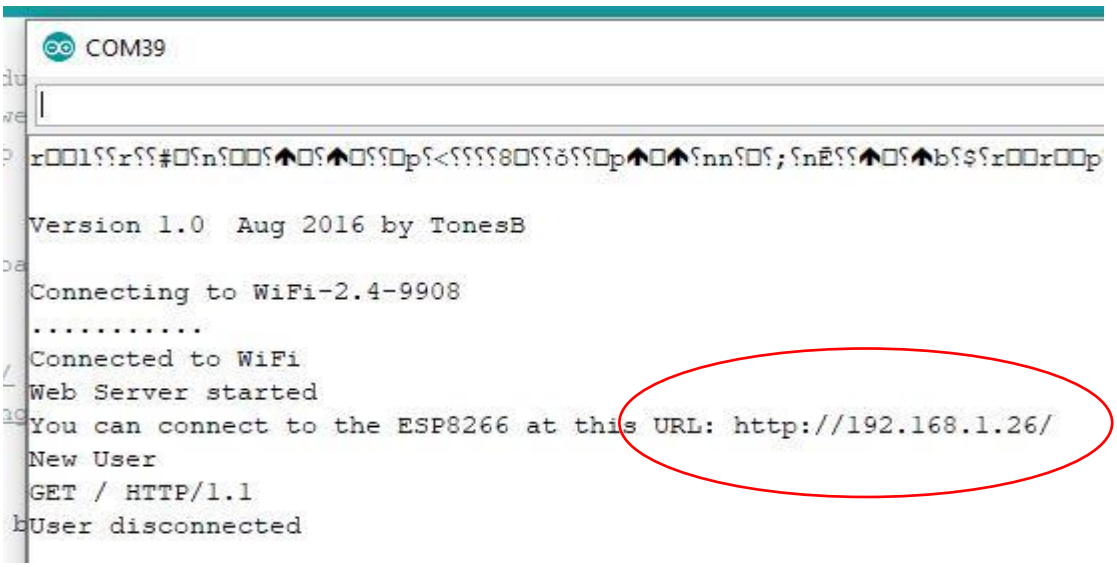
De blauwe LED op de ESP zal af en toe kort oplichten.





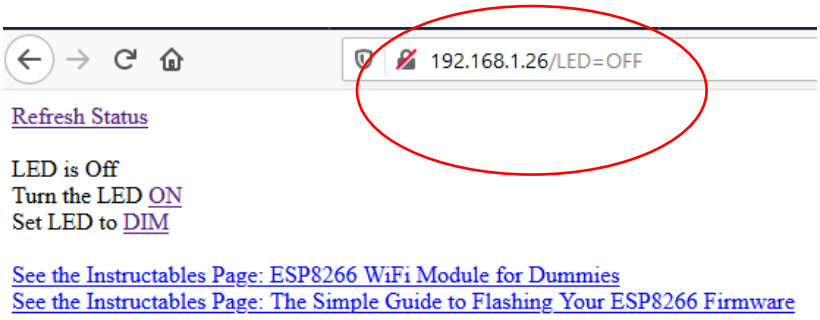
Tijdens de programmatie zie je bovenstaande info verschijnen.

Doe nu de serial monitor open, **verwijder de GND van de GPIO0 pin en druk op reset.**



Nu start de ESP op als een kleine server en wordt jouw html pagina getoond op de URL die gegeven is op de monitor.

Ga naar deze pagina in een browser als firefox of explorer. Type hiervoor de URL in de bovenste balk.



Nu verschijnt de volgende tekst. Klik op “ON” of “DIM” om de LED op jouw bordje via wifi aan te sturen. In de serial monitor kan je nog steeds de status van de ESP volgen.

Nadat je de programmer hebt verwijdert van de ESP kan je deze laatste zelfstandig ergens plaatsen in de ruimte en van een 3V3 voeding voorzien. Zo kan je nu verschillende IOT apparaten koppelen aan jouw omgeving. Vergeet niet van even de GPIO0 draad naar massa te verwijderen (en eventueel op een andere I/O aan te sluiten. Daarna op reset drukken om de ESP weer in de werkmode te krijgen.

Uitdagingen:

- Zorg er nu voor dat je een knop kan inlezen en dat je op jouw website een tekst ziet veranderen (vb aan/uit).
- Zorg dat op GPIO0 een LED zit en GPIO2 een schakelaar. Toon de status van de schakelaar op de website. Zorg dat de LED aangaat nadat je op de website op een link hebt geklikt.
- Maak nu de website pagina nog knapper en probeer van hieruit 2 LEDs aan te sturen.
- Probeer nu een relais aan te sluiten op de ESP en deze te schakelen via de website. http://www.forward.com.au/pfod/ESP8266/GPIOpins/index.html
- Jammer genoeg heeft de ESP 01 geen analoge ingangen ter beschikking. De ESP v7 en v12 wel. https://www.instructables.com/id/ESP8266-ADC-Analog-Sensors/