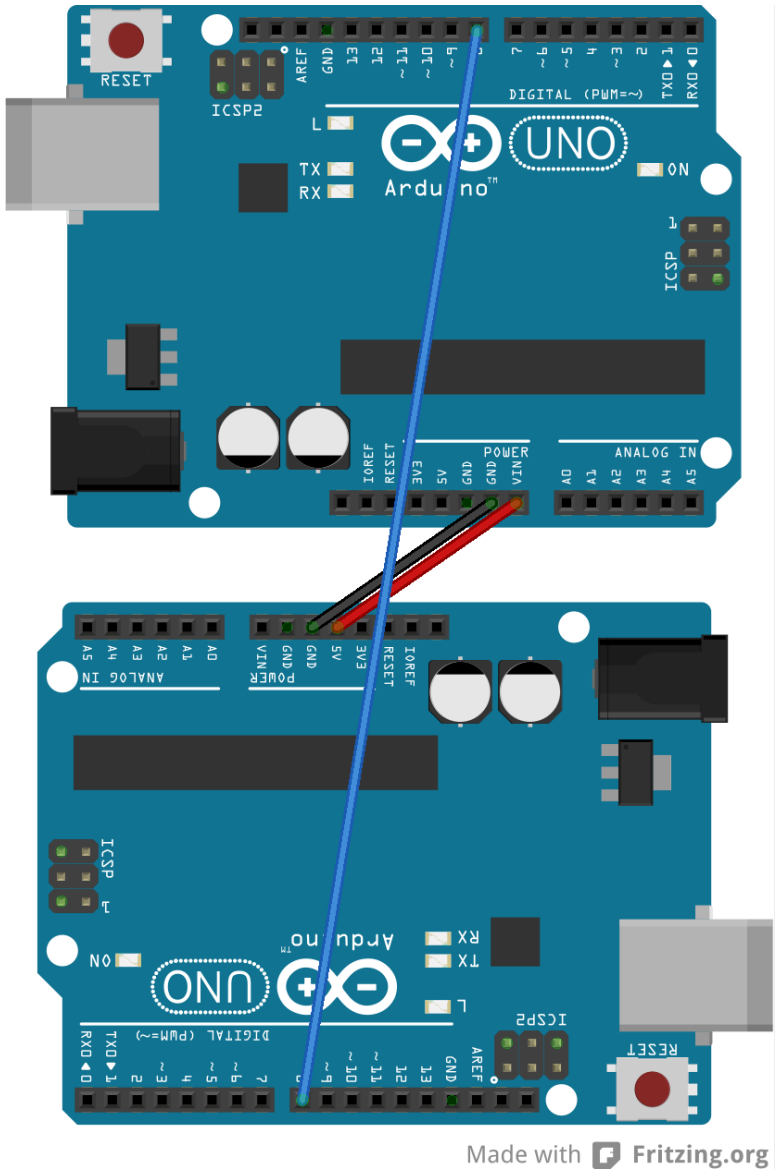
Serielle Kommunikation

Arduinos verbinden

Wenn wir zwei Arduinos wie abgebildet mit Strom versorgen und ihren GND-Pins verbinden, können diese über einen digitalen Pins (hier Nr. 8) miteinander kommunizieren.

Dazu verwendet man etwa die folgenden Sketches:



Die Software zur Kommunikation

```
// Sender

int com = 8;


void setup() {
  pinMode(com, OUTPUT);
  sendBit();
}

void sendBit() {
  digitalWrite(com, HIGH);
  delay(1000);
  digitalWrite(com, LOW);
}


void loop() {
}
```

```
// Receiver

int led = 13;
int com = 8;

void setup() {
  pinMode(led, OUTPUT);
  digitalWrite(led, LOW);
  pinMode(com, INPUT);
}

void blink(){
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
}

void loop() {
  int val=digitalRead(com);
  if(val==HIGH)
    blink();
}
```

Einfache Kommunikation

In diesem einfachen Szenario, sendet ein Arduino immer wenn er eingeschaltet wird, eine Nachricht an den anderen Arduino, der dafür sorgt, dass die LED 13 aufleuchtet.

Komplexere Nachrichten bedürfen eines komplexeren Protokolls, das etwa auch für die Synchronisierung der Arduinos sorgt.

Besser ist es vielleicht, hier ein Standard-Protokoll zu nutzen.

Serielle Kommunikation auf dem Arduino

Host und Targets verwenden **serielle Kommunikation**, um

- einen Sketch auf den Arduino zu laden
- allgemeine Daten auszutauschen

Ein einfacher Use Case

Bevor wir die serielle Kommunikation diskutieren, noch einige praktische Vorgehensweisen:

Wozu sollen Target und Host Daten austauschen?

Einfacher Use Case:

- Eingabe einer Zahl n am Host
- Am Target leuchtet eine LED für n Sekunden

Allgemein kann die serielle Kommunikationen sogar zur **Vernetzung** mehrere Knoten genutzt werden.

Wie geht das?

Die Klasse `Serial` der Arduino API enthält Methoden für die Arbeit mit der seriellen Schnittstelle

1. Beispiel: Die Methode `println`
Einen Text vom Arduino an die serielle Schnittstelle **senden**:

```
void setup() {
  Serial.begin(9600);
  Serial.println("Hello World");
}

void loop() {

}
```

Jedes Mal, wenn die Reset-Taste gedrückt wird, wird ein Text verschickt

Anzeigen serieller Daten

Wie kann man Texte sichtbar machen, die an die serielle Schnittstelle gesendet werden?

1. Verwendung eines Terminals

2. Verwendung eines Terminal-Emulators auf dem Host :

a. Die Arduino IDE hat einen integrierten

Emulator: Werkzeuge: Serieller Monitor

b. Möglich ist auch die Verwendung von PuTTY, screen und anderen Terminal-Emulatoren.

Ein Beispiel-Sketch

2. Beispiel:

Arduino **empfängt** einen Text von der seriellen Schnittstelle:

```
int incomingByte = 0;

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

void loop() {
  if (Serial.available() > 0) {
    incomingByte = Serial.read();
    Serial.print("I received: ");
    Serial.println(incomingByte);
  }
}
```

Die Methode `serialEvent`

Wenn man im Terminal ein Zeichen eingibt, gibt der Sketch den zugehörigen ASCII-Code aus. Das Zeichen selbst erhält man mit der Anweisung

```
char(incomingByte)
```

Die **stets gleiche Abfrage** nach neuen Zeichen kann man vermeiden, wenn man die Funktion `serialEvent` nutzt, die nach jedem Durchlauf von `loop` aufgerufen wird, wenn Daten über die serielle Schnittstelle verfügbar sind.

Wir ändern unseren Sketch wie folgt:

Ein Beispiel-Sketch

```
int incomingByte = 0;
int led=13;

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

void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}

void serialEvent() {
  incomingByte = Serial.read();
  Serial.print("I received: ");
  Serial.println(incomingByte);
}
```

Der Arduino blinkt, empfängt und sendet Daten.

Bibliotheken für serielle Kommunikation

Geht das auch mit eigener Software auf dem Host?

Für alle gängigen Programmiersprachen (C, Java, Python, …) gibt es **Bibliotheken**, mit deren Hilfe man auf die serielle Schnittstelle zugreifen kann.

In C gibt es etwa die **termios**-Bibliothek, die Funktionen enthält, mit deren Hilfe man Parameter wie die Baudrate oder die Stop-Bits konfigurieren kann.

Ein einfaches C-Programm

Hier ein Beispiel in C, um einen Text zu versenden. Beachten Sie, dass das Programm mit den geeigneten Rechten ausgeführt werden muss.

Es werden die Standardeinstellungen (Baudrate usw.) für den Treiber des Gerätes verwendet. Eine spezifische Konfiguration ist so nicht möglich. Es gibt daher keine Garantie dafür, dass das Programm auf allen Plattformen gleich arbeitet

Ein einfaches C-Programm

```
#include <stdio.h>

void send(char* word){
    FILE *file;
    file = fopen("/dev/cu.usbmodemfa141","w");
    sleep(2);
    fprintf(file,"%s",word);
    fclose(file);
}

int main(){
    printf("starting...\n");
    char name [20]="text\n";
    name[5]=0;
    send(name);
    printf("terminating...\n");
   return 0;
}
```

Es muss das passende Unix-Device eingetragen werden

Ein wichtiger Hinweis

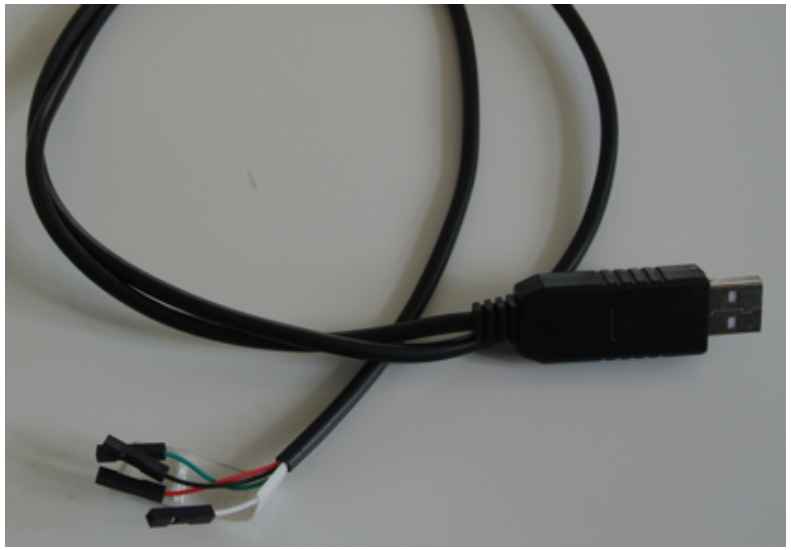
Achtung:

Die serielle Verbindung zwischen Host und Arduino wird immer vom Host angestoßen. Da in der `setup`-Methode eines Sketches oft wichtige Initialisierungen durchgeführt werden, werden einige Arduino Modelle (wie der Uno und der Due) beim Aufbau einer seriellen Verbindung **automatisch** neu gestartet. Dieser Vorgang dauert etwa 1 Sekunde. Daten, die in dieser Zeit vom Host gesendet werden, gehen **verloren**! Daher wurde die Zeile `sleep(2)` eingefügt.

Den Neustart eines Arduinos erkennt man (je nach Modell) am Flackern von LEDs.

Der Raspberry Pi

Wir hatten bereits früher mit serieller Übertragung zu tun. Der Raspberry Pi kommuniziert über das abgebildete Kabel mit dem Host:

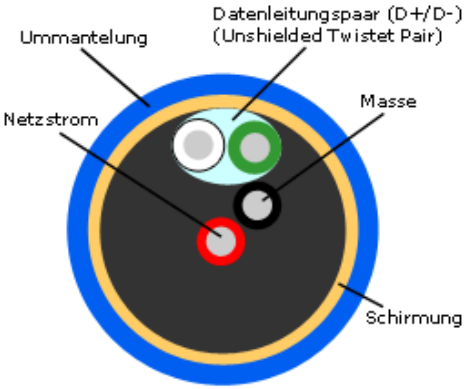


Auf der

- einen Seite ist *ein* USB-Stecker
- anderen Seite sind *vier* Ausgänge

Ein USB-Kabel

Querschnitt durch ein USB-Kabel

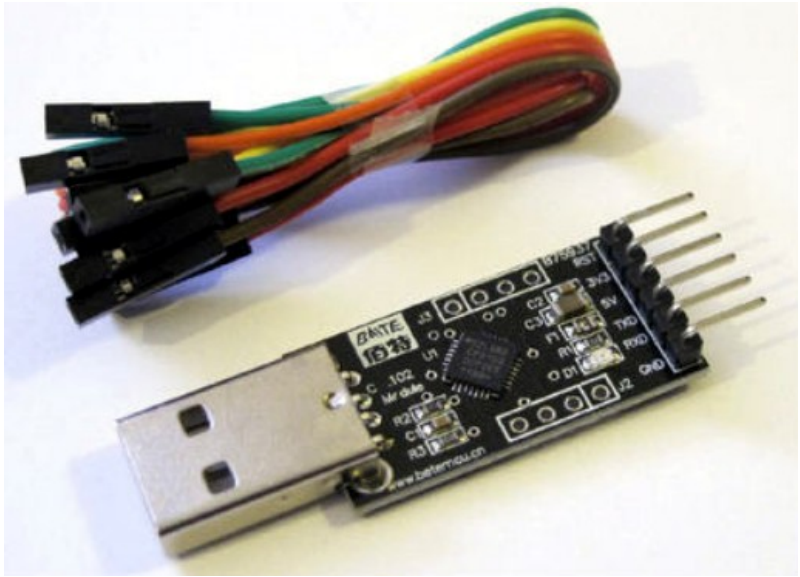


Interessant ist, dass ein USB-Kabel neben der Stromversorgung nur ein einziges Kabelpaar für die Datenübertragung enthält. Für die Übertragung wird nicht die serielle Schnittstelle genutzt.

Die Transformation der Signale für die Ausgänge erfolgt mit Hilfe eines FTDI-Chips, der in das Kabel integriert ist. Der grüne und der weisse Ausgang liefern dann serielle Daten

Der direkte Draht

Ein solcher Chip wird auch bei Adaptern mit eigenen Pins verwendet, die direkt über Jumper-Kabel mit der USB-Schnittstelle des Targets verbunden werden können:



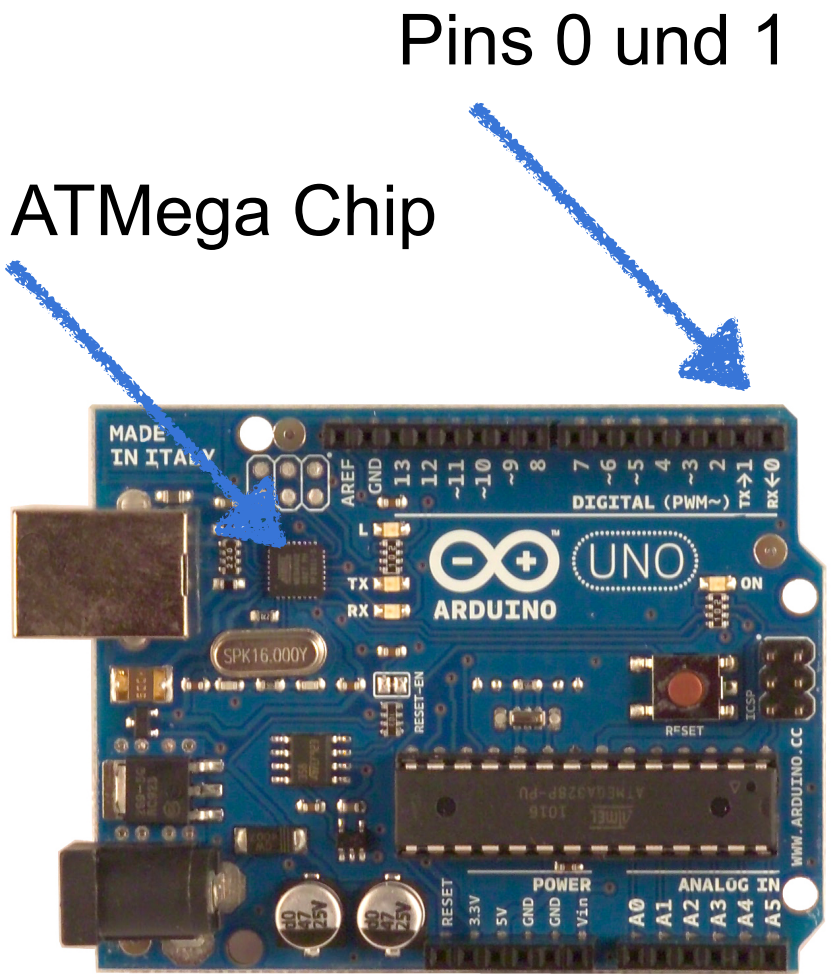
Der direkte Draht

Alternativ kann auch ein konfektioniertes Kabel genutzt werden.

Wie ist das beim Arduino?

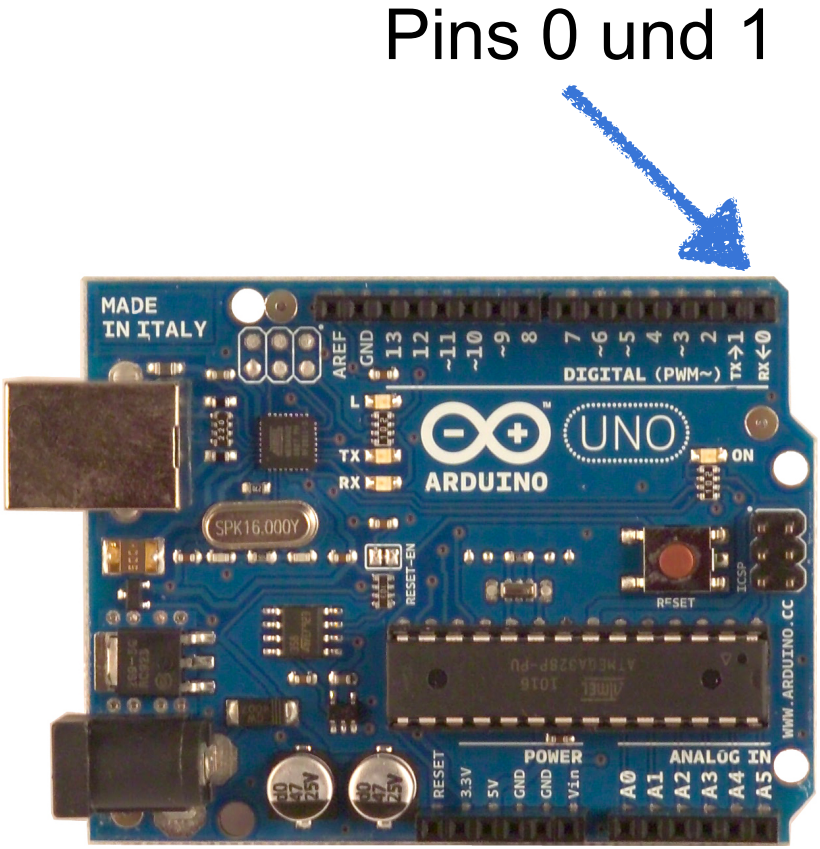
Ähnlich ist dies beim Arduino möglich. Der Uno besitzt ein RX/TX-Paar auf den Pins 0 und 1.

Diese sind mit einem ATmega16U2 USB-to-TTL Chip verbunden, der die Daten für die USB-Schnittstelle transformiert.



Wie ist das beim Arduino?

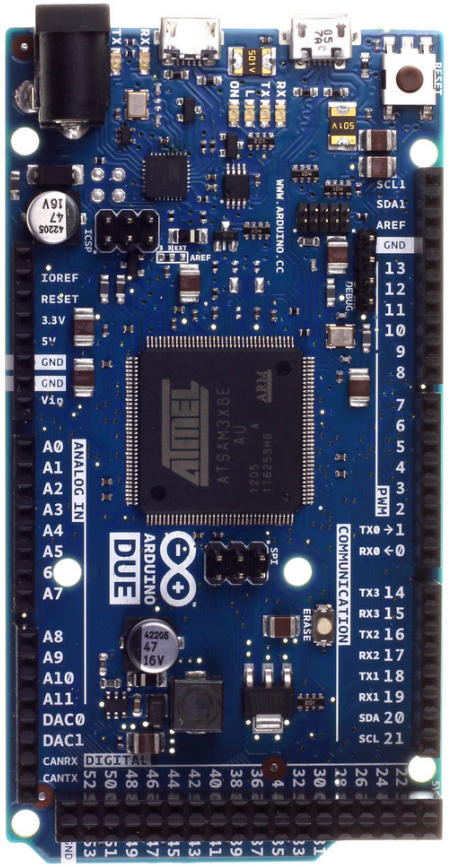
Grundsätzlich kann der Uno also über die Pins 0 und 1 mit anderen Geräten seriell kommunizieren. Dies ist nicht möglich, solange die USB-Verbindung genutzt wird: Diese verwendet ebenfalls die Pins 0 und 1.



Wenn die USB-Verbindung benötigt wird, gibt es zu diesem Problem **zwei Lösungen.**

1. Lösung

Modelle wie der Arduino Due verfügen neben den Pins 0 und 1 über weitere Paare, die mit dem ATMega Chip verbunden sind und zur seriellen Kommunikation genutzt werden können.



Beachten Sie, dass es für diese zusätzlichen Pin-Paare eigene API-Methoden gibt.

2. Lösung

Standardmäßig nutzen die Pins **Hardware** für die Kommunikation. Ein eigner Chip ist etwa für die Verarbeitung der Signale der Pins 0 und 1 zuständig. Dieser Chip ist direkt mit dem zentralen Microcontroller verbunden.

Die API bietet auch die Möglichkeit diesen Vorgang mit Hilfe von **Software** nachzubilden

Beachten Sie, dass es für diese zusätzlichen Pin-Paare eigene API-Methoden gibt.

Virtuelle serielle Pins

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // RX, TX

void setup(){
  Serial.println("Goodnight moon!");
  mySerial.begin(4800);
  mySerial.println("Hello, world?");
}

void loop()
{
  if (mySerial.available())
    Serial.write(mySerial.read());
  if (Serial.available())
    mySerial.write(Serial.read());
}
```