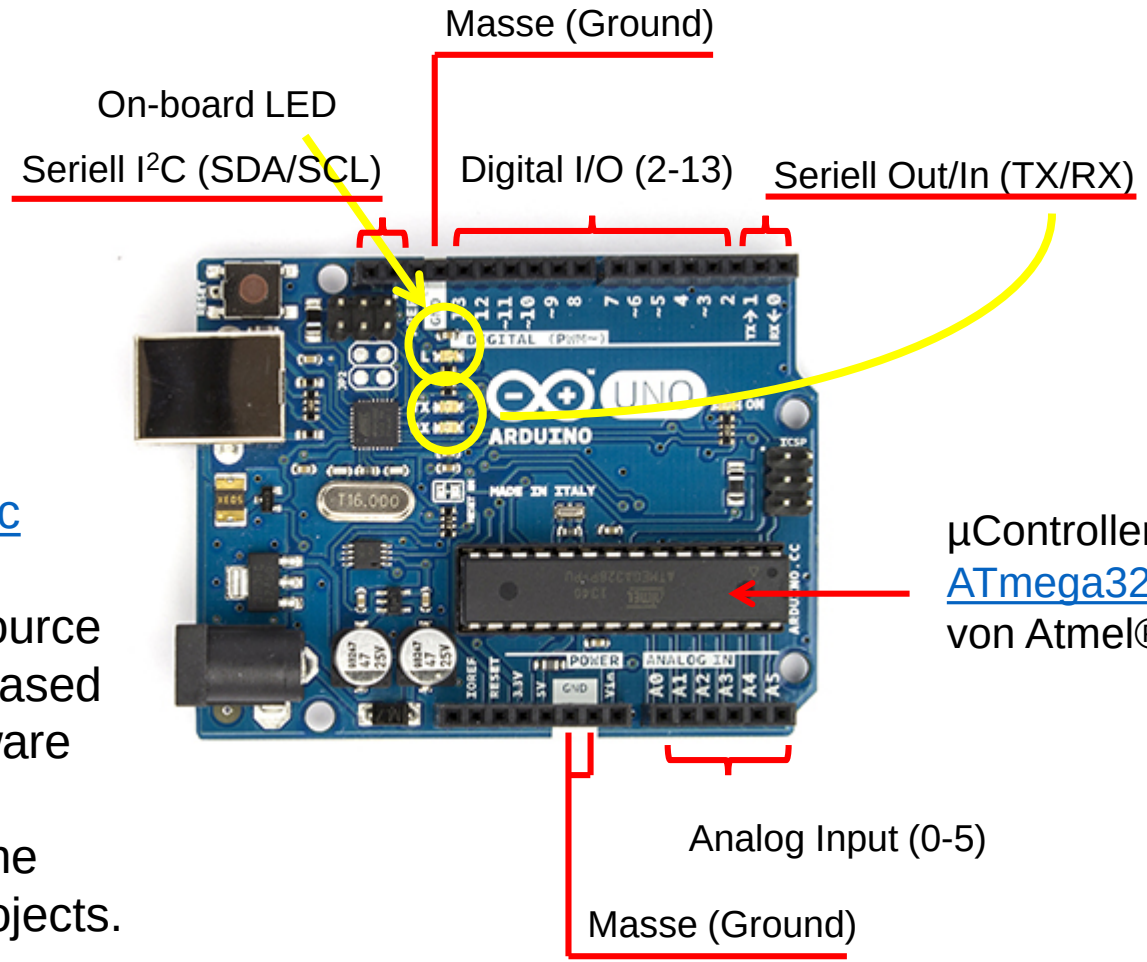
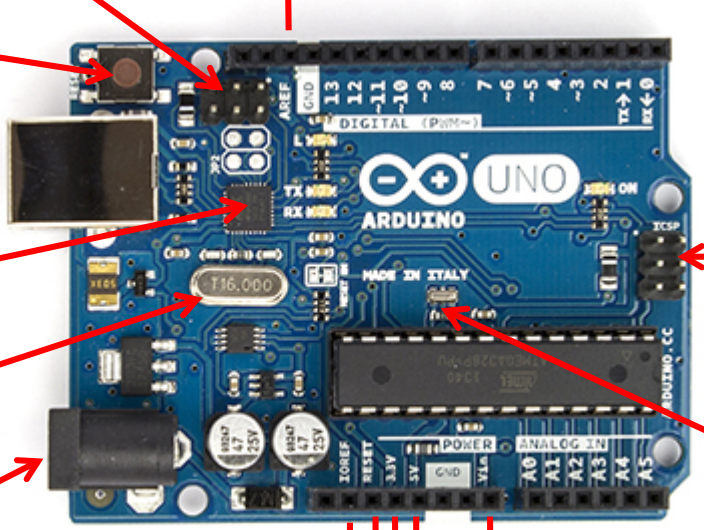
Arduino



https://www.arduino.cc

Arduino is an open-source electronics platform based on easy-to-use hardware and software.
It's intended for anyone making interactive projects.

Arduino


ICSP
ATmega16U2
AREF: analog reference (default: 5 V)
Reset-Knopf
USB
µController
ATmega16U2
steuert USB
16 MHz Quarz
Spannungsversorgung
(alternativ über USB)
ICSP
In-circuit serial
programming
ATmega328
16 MHz Quarz
Voltage In
3.3 V (Out)
5 V (Out)
Reset
IOREF
(indicates 3.3/5 V)

Arduino – Programmierung

8 Bit µController
16 MHz Takt
32 kByte Flash Speicher – für Programm & Bootloader (5 kByte)
2 kByte statischer Speicher (SRAM) – für Variablen
1 kByte EEPROM – Langzeitspeicher

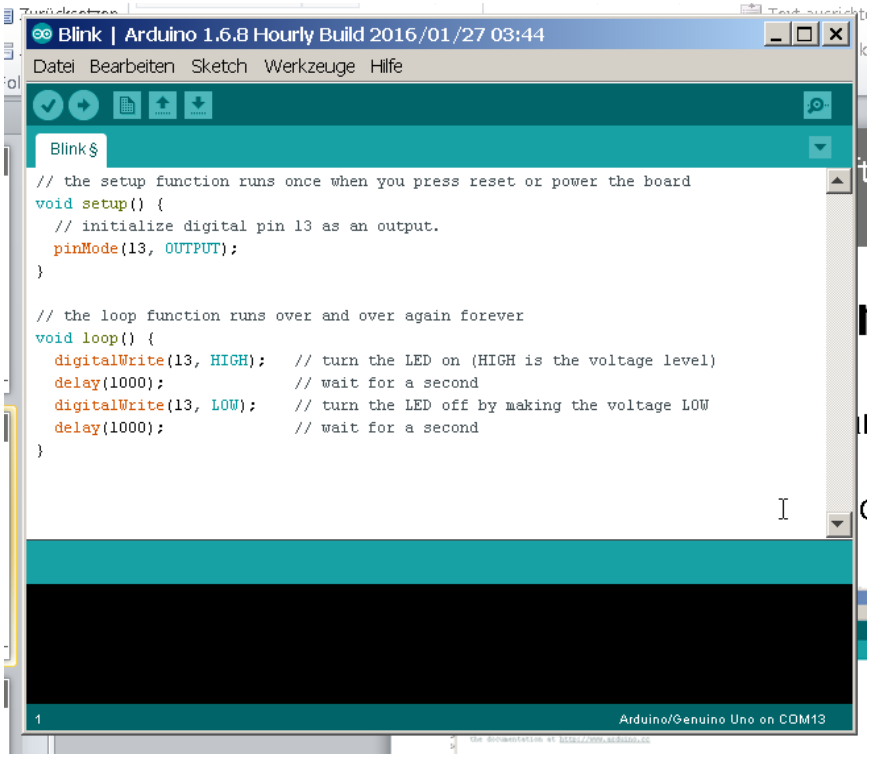
Flash und EEPROM bleiben auch ohne Spannungsversorgung erhalten (non-volatile)

… das ist wenig im Vergleich mit einem PC (64 Bit Prozessor, 4 Kerne, 3 GHz, 32 GByte Speicher,…)

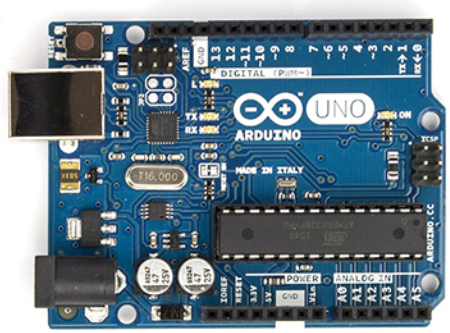
Arduino – Programmierung

Bootloader „zieht“ Programm vom PC auf das Arduino Board (ist installiert)

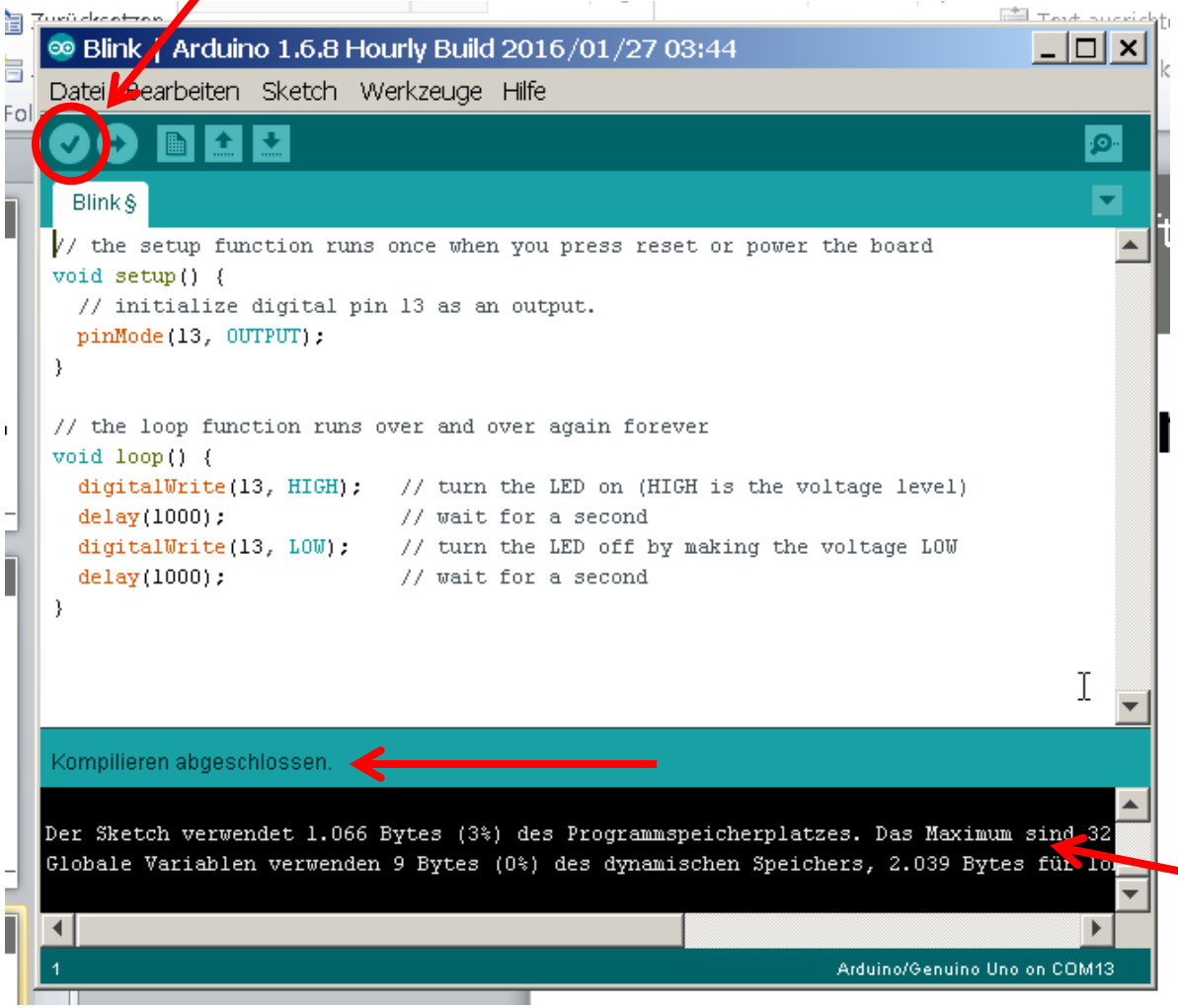
Arduino Integrated Development Environment (IDE) auf einem PC



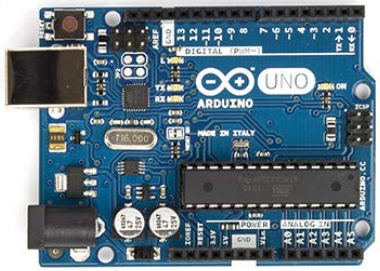
USB Kabel
+ USB Hub



Compilieren (auf PC für ATmega übersetzen)

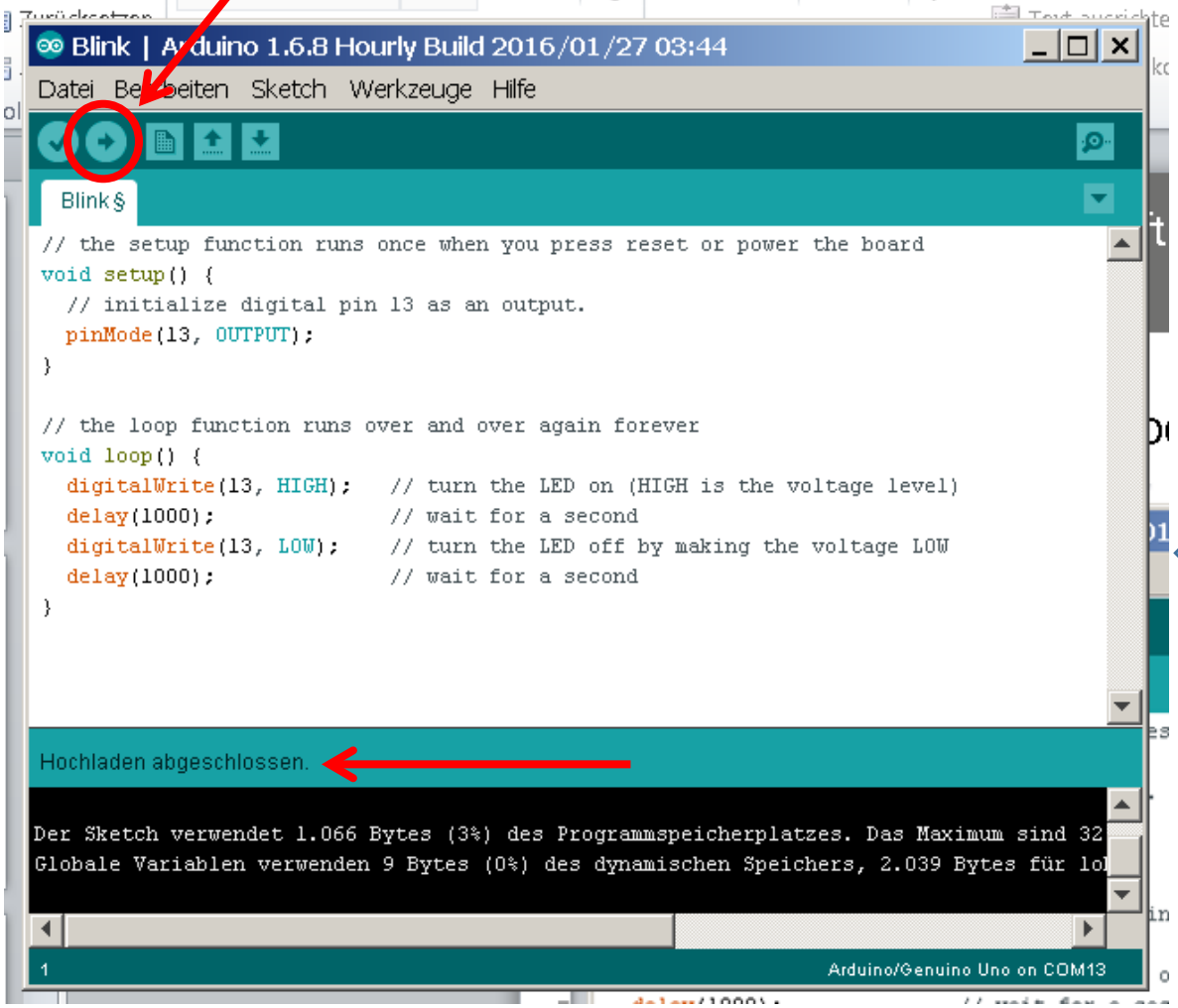


USB Kabel
+ USB Hub



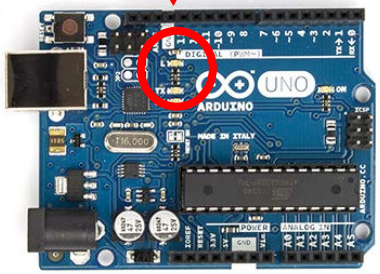
(Fehler-) Meldungen vom Compiler

Auf Arduino hochladen (via USB)



USB Kabel
+ USB Hub

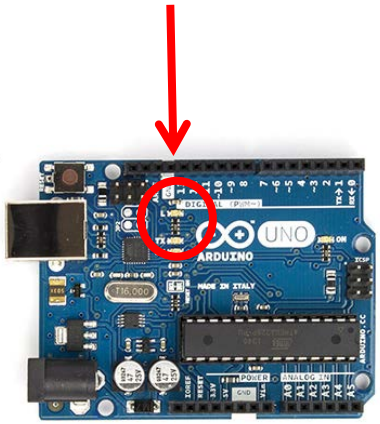
LED blinkt
(Programm läuft)



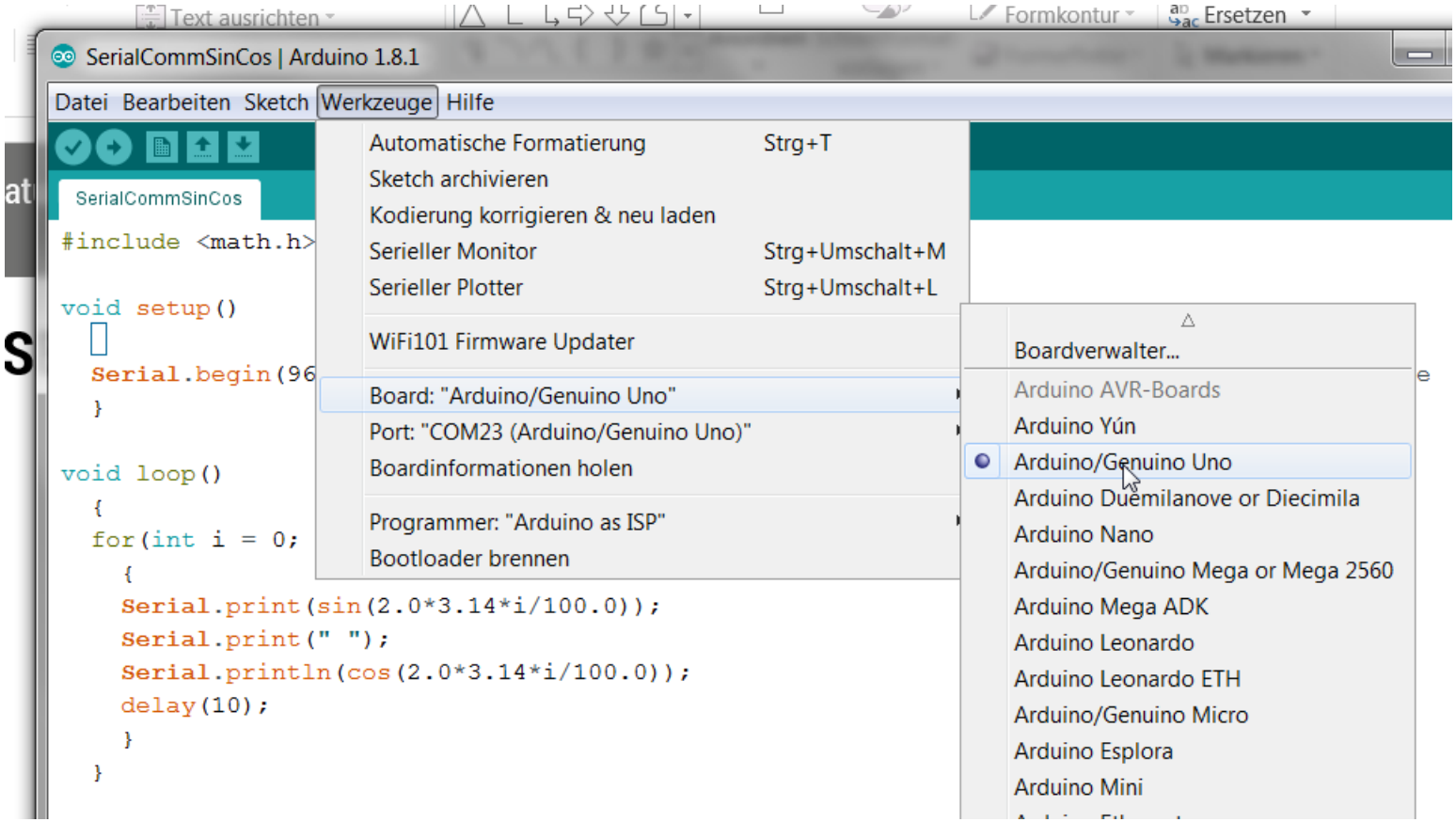
Wenn der Arduino abgesteckt wird…
…und anschließend wieder angesteckt oder anderweitig mit Strom versorgt wird,
… blinkt die LED weiter!

Das Programm ist im Flash Speicher „non-volatile“ gespeichert.

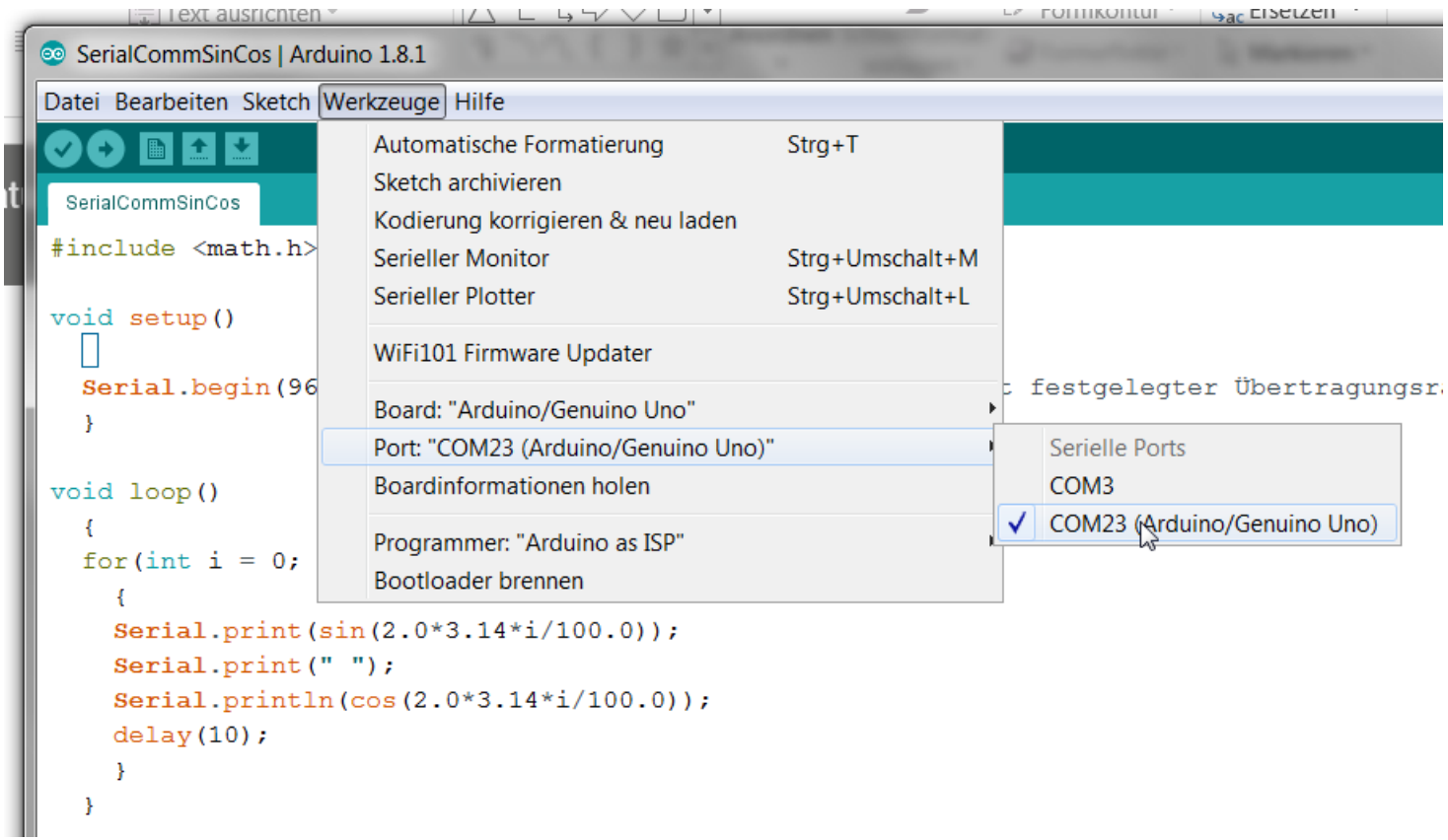
LED blinkt
(Programm läuft)



Arduino USB Bord einstellen



Arduino USB Port einstellen



Arduino Programming Language

https://www.arduino.cc/en/Reference/HomePage

Structure

- setup()
- loop()

Control Structures

- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return
- goto

Further Syntax

- ; (semicolon)
- {} (curly braces)
- // (single line comment)

Variables

Constants

- HIGH | LOW
- INPUT | OUTPUT | INPUT_PULLUP
- LED_BUILTIN
- true | false
- integer constants
- floating point constants

Data Types

- void
- boolean
- char
- unsigned char
- byte
- int
- unsigned int
- word
- long

Functions

Digital I/O

- pinMode()
- digitalWrite()
- digitalRead()

Analog I/O

- analogReference()
- analogRead()
- analogWrite() - *PWM*

Due & Zero only

- analogReadResolution()
- analogWriteResolution()

Advanced I/O

- tone()
- noTone()
- shiftOut()

Erster Sketch „Blinken“

Struktur eines Sketches:

```
void setup() {
// Code in "setup" wird einmal am Anfang ausgeführt
}

void loop() {
// Code in "loop" wird in einer Endlosschleife ausgeführt
}
```

Es gibt kein main() und kein Programmende…

```
// kennzeichnen folgenden Text als Kommentar
```

Farbschema:
Strukturen
Variablen
Funktionen

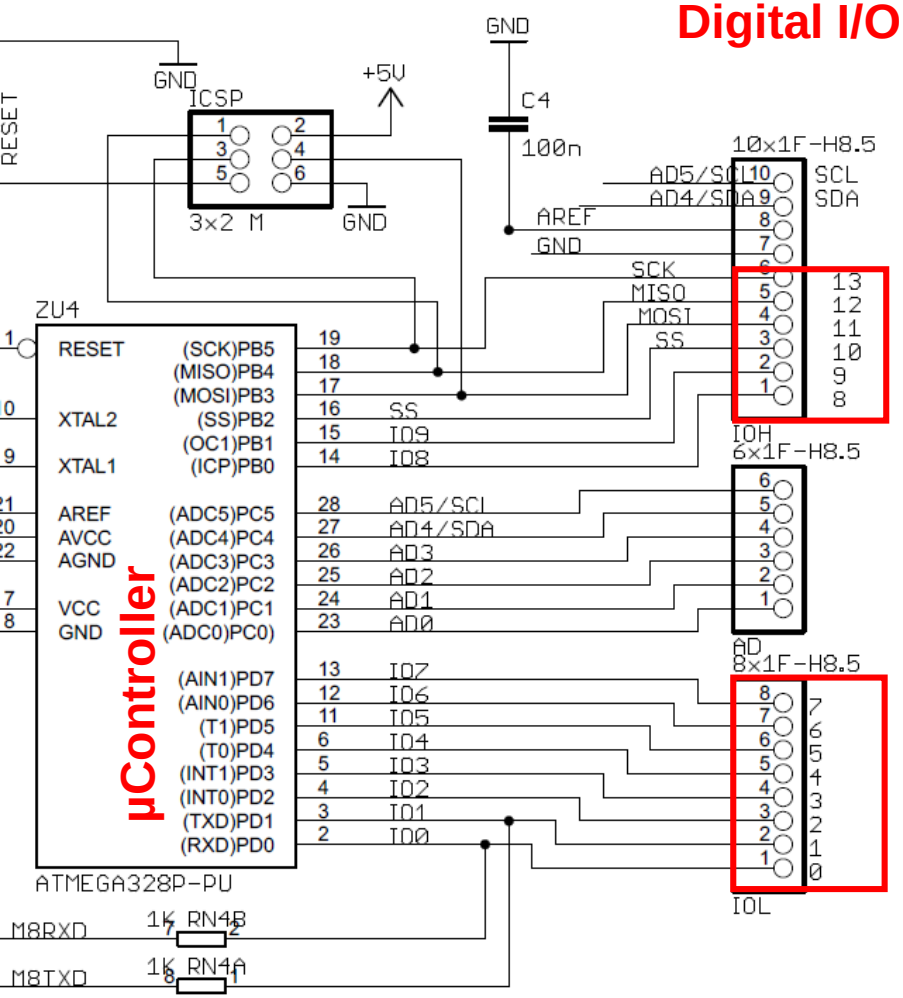
Erster Sketch „Blinken“

Struktur eines Sketches:

```
void setup() {
// initialisiert den digitalen Pin 13 als Ausgang (output).
  pinMode(13, OUTPUT);
}


void loop() {
// Code in "loop" wird in einer Endlosschleife ausgeführt
}
```

Funktion `pinMode(pin, mode)` mit den beiden Argumenten „pin“ und „mode“ legt Funktion eines digitalen Pins fest.
Hier für pin 13, an den die on-board LED angeschlossen ist.
`OUTPUT` ist eine vordefinierte Konstante.
Für das Argument „mode“ ist `INPUT`, `OUTPUT`, `INPUT_PULLUP` möglich.

Arduino - Schaltplan



`pinMode(pin, INPUT)`

- Eingang ist hochohmig (100 MΩ, sehr wenig Strom)
- LOW: < 0.8 V
- HIGH: > 2.2 V
- Offen: unbestimmtes Ergebnis

`pinMode(pin, INPUT_PULLUP)`

- Eingang mit ca. 20 k Ω auf 5V gezogen
- LOW: < 0.8 V
- HIGH: > 2.2 V
- Offen: HIGH

`pinMode(pin, OUTPUT)`

- Eingang ist niederohmig (Stromquelle bis 40 mA)
- LOW: 0 V
- HIGH: 5 V

Erster Sketch „Blinken“

```
void setup() {
// initialisiert den digitalen Pin 13 als Ausgang (output).
  pinMode(13, OUTPUT);
}


void loop() {
  digitalWrite(13, HIGH);   // LED einschalten
  delay(1000);              // eine Sekunde warten
  digitalWrite(13, LOW);    // LED ausschalten
  delay(1000);              // wieder 1000 ms warten
}
```

Funktion `digitalWrite(pin, level)` mit den beiden Argumenten „pin“ und „level“ setzt digitalen Pins auf `HIGH` (5 V) oder `LOW` (0 V).

Funktion `delay(milliseconds)` wartet „milliseconds“ Millisekunden.

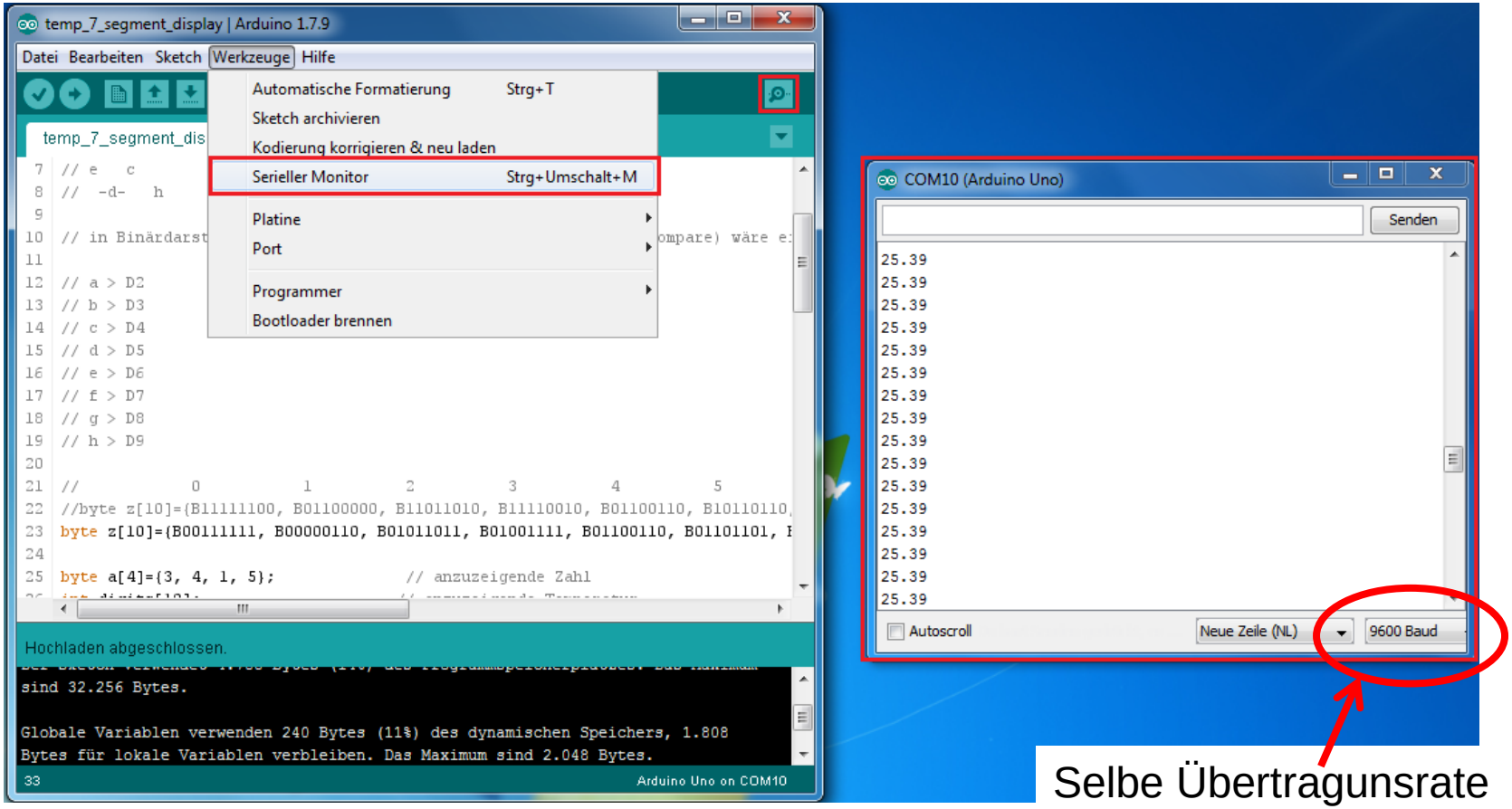
Serielle Kommunikation mit dem PC

- Darstellen und Überprüfen von Sensordaten
- Weiterverarbeitung in externen Programm (Mathematica, Python, R,…)
- Fehlersuche („debuggen“)

Am Anfang war das „Hello World“…
Senden von Daten vom Mikrocontroller an den PC:

```
void setup()
        {
        Serial.begin(9600);    // startet serielle Kommunikation mit
        }                      // festgelegter Übertragungsrate

void loop()
        {
        ...
        Serial.print(“Hello World”);  // Gibt “Hello World” auf dem
                                      // seriellen Monitor aus

        Serial.println(“...”);        // dito, aber mit Zeilenumbruch
        }
```

Serieller Monitor

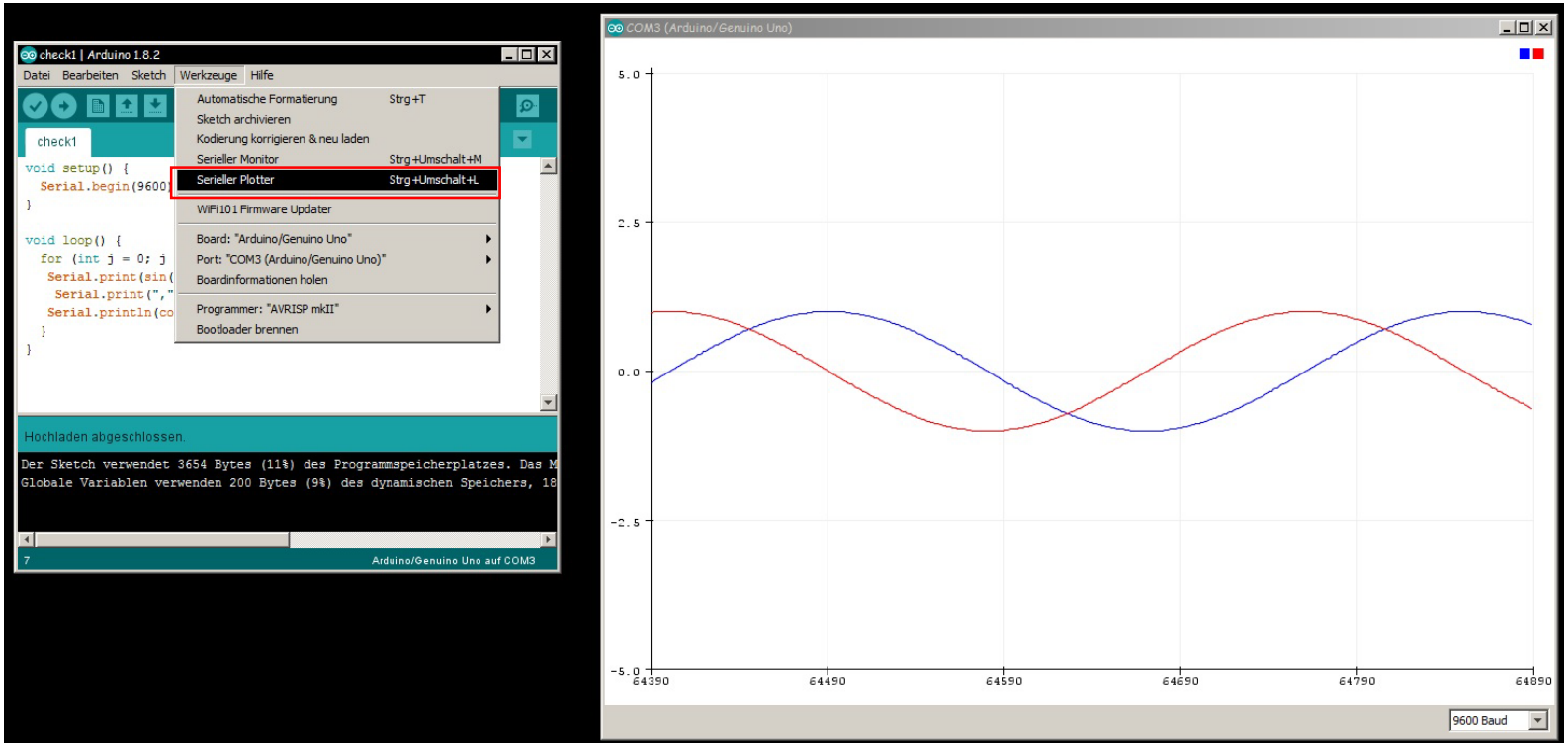


Selbe Übertragunsrate (Baud Rate), wie in `Serial.begin(9600);`

Serielle Kommunikation mit dem PC

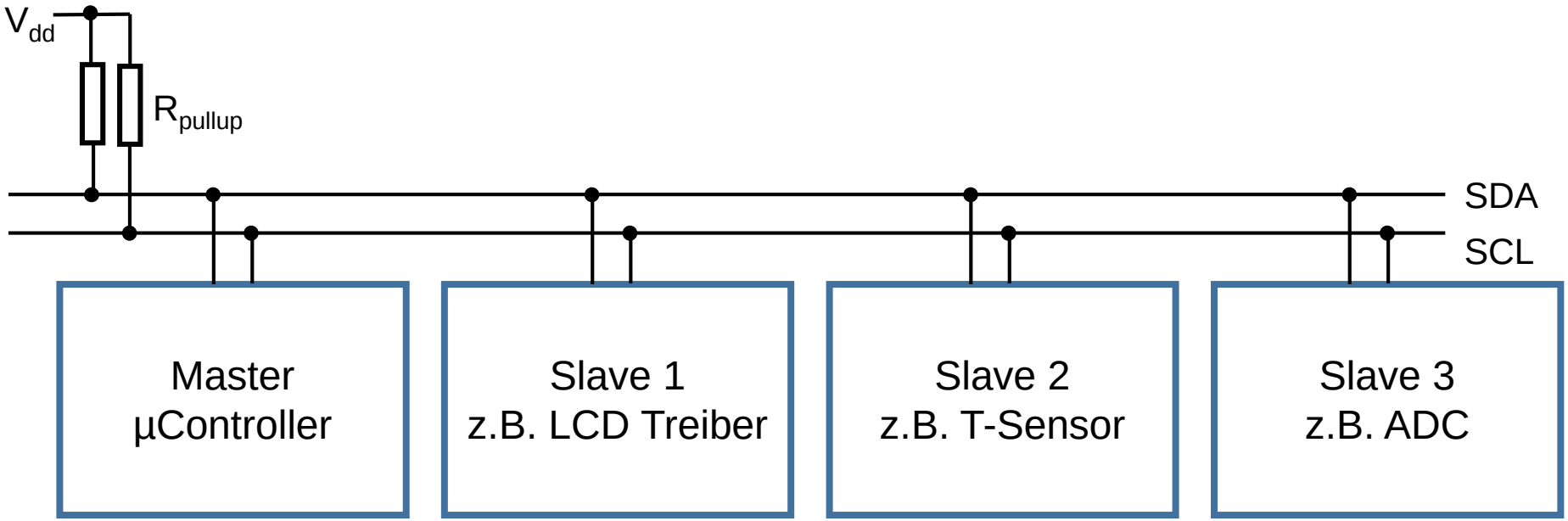
```
void setup()
        {
        Serial.begin(9600);    // startet serielle Kommunikation mit
        }                      // festgelegter Übertragungsrate

void loop()
        {
        for(int i = 0; i < 100; i++)
                {
                Serial.print(sin(2.0*3.14*i/100.0));
                Serial.print(“ “);
                Serial.println(cos(2.0*3.14*i/100.0));
                delay(10);
                }
        }
```

Serieller Plotter



I²C Bus – Inter-Integrated Circuit

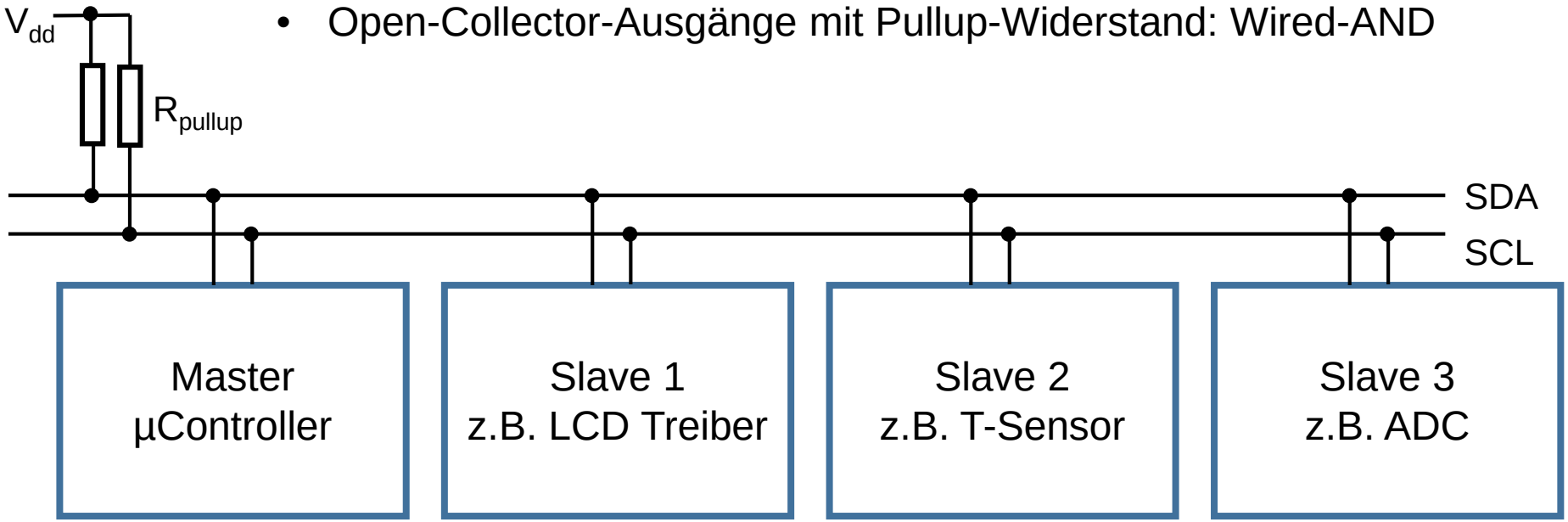
- Serieller Bus
- Master-Slave-Bus
- zwei Signalleitungen: Takt- (SCL) und Datenleitung (SDA)
- Open-Collector-Ausgänge mit Pullup-Widerstand: Wired-AND



I²C Bus – Inter-Integrated Circuit

Kommunikation zwischen Mikrocontroller und Peripherie (z.B. Sensor-ICs).

- Serieller Bus
- Master-Slave-Bus
- zwei Signalleitungen: Takt- (SCL) und Datenleitung (SDA)
- Open-Collector-Ausgänge mit Pullup-Widerstand: Wired-AND

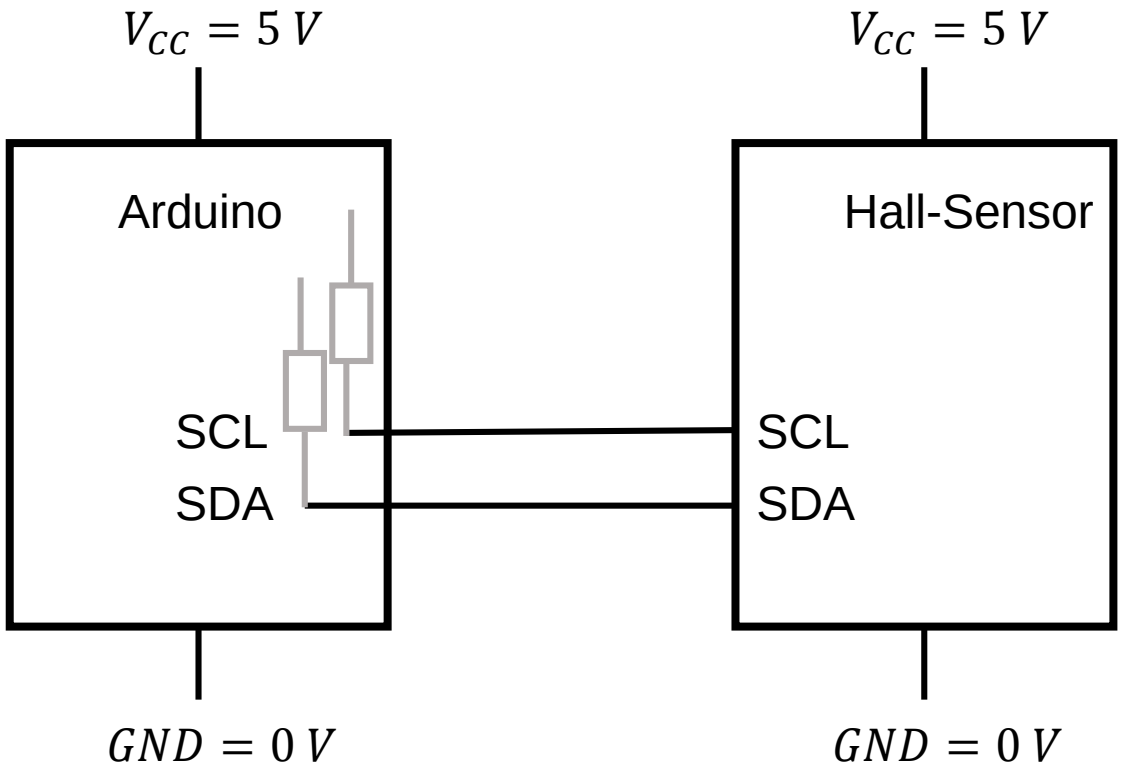


I²C Bus – Einfachstes Beispiel

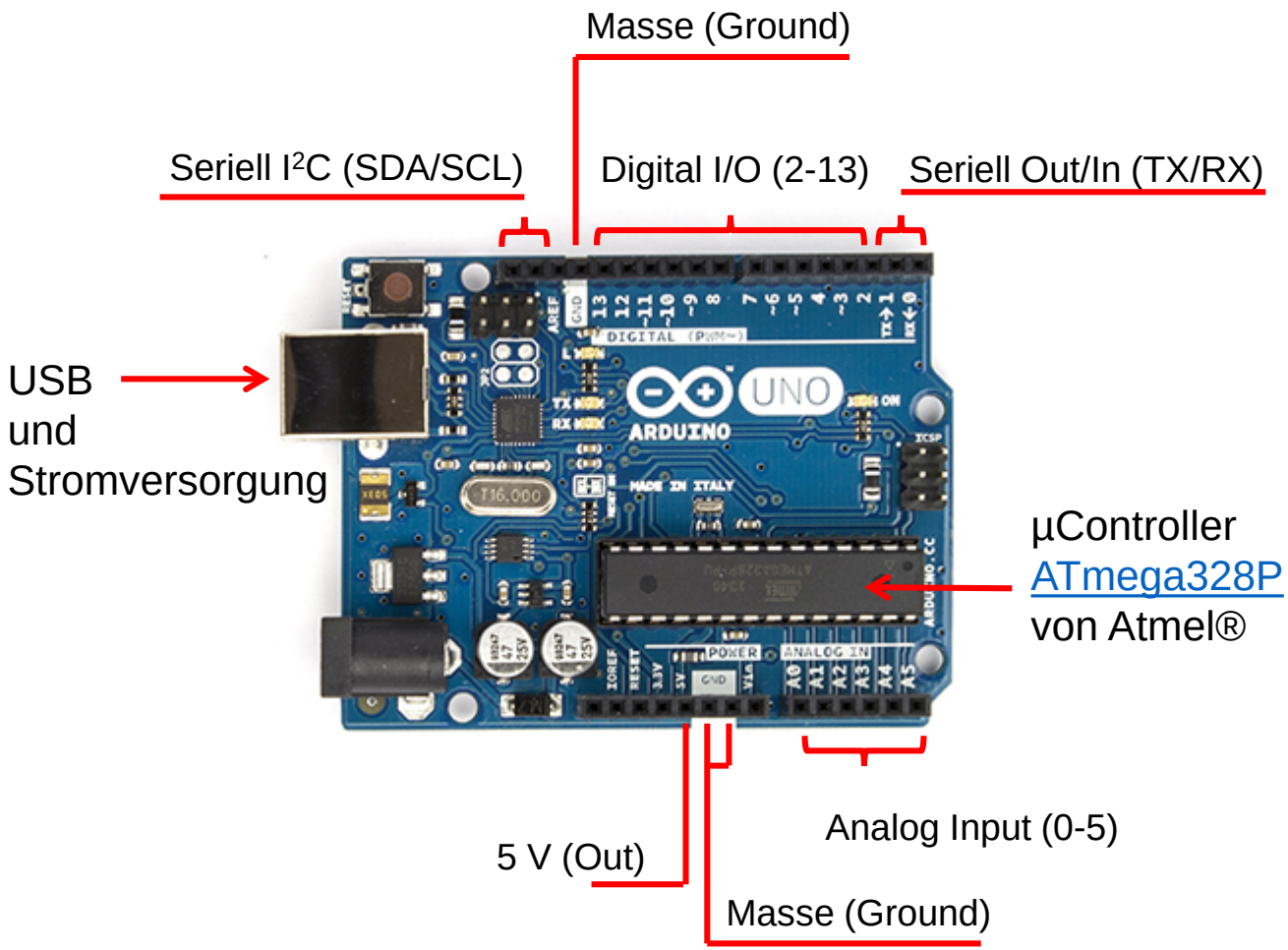
3d Magnetfeldsensor an Arduino: Kommunikation in beide Richtungen

SCL: serial clock
SDA: serial data

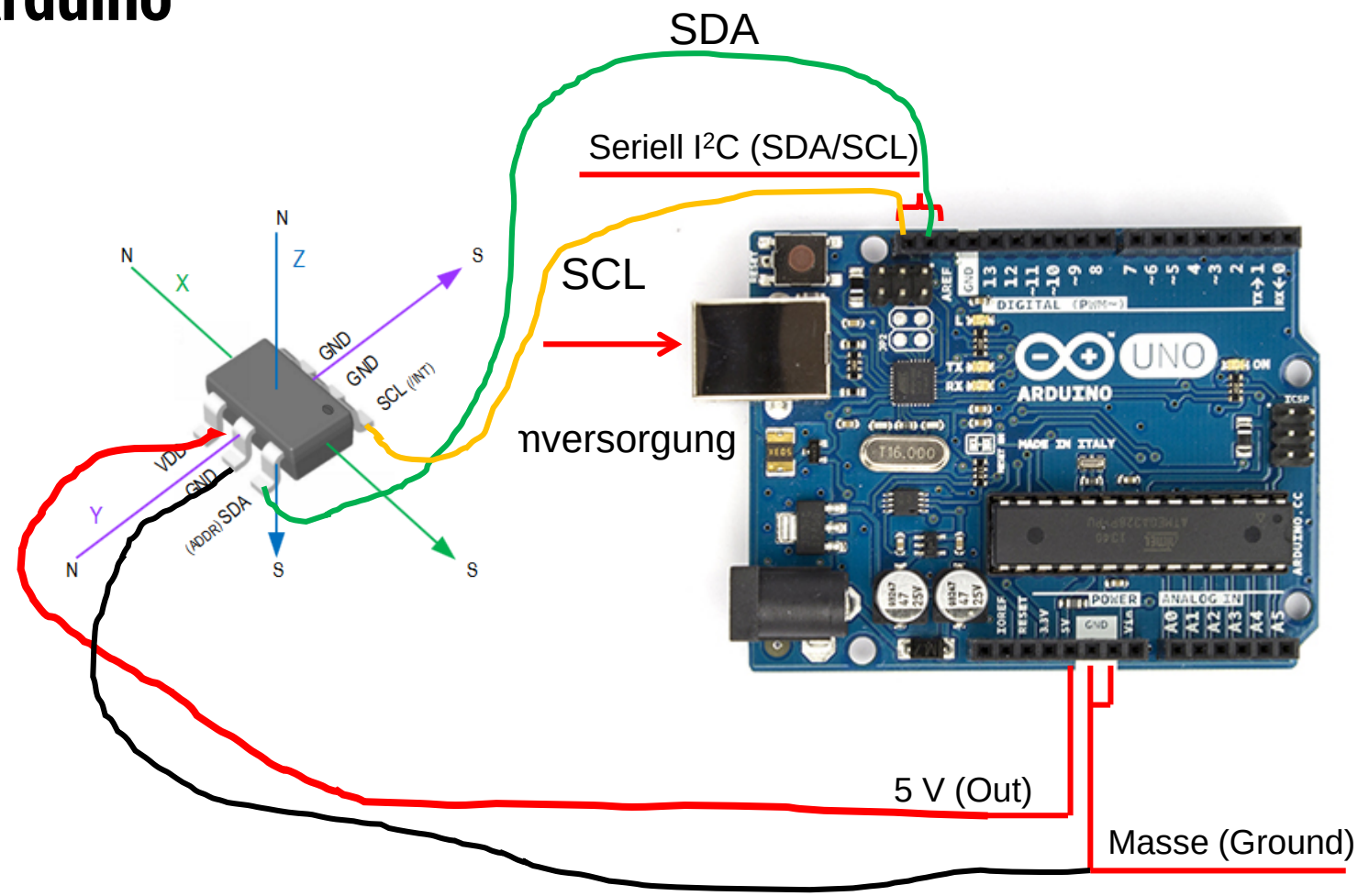
Interne Pull-Up Widerstände



Arduino


Masse (Ground)
Seriell I²C (SDA/SCL)
Digital I/O (2-13)
Seriell Out/In (TX/RX)
USB
und
Stromversorgung
µController
ATmega328P
von Atmel®
Analog Input (0-5)
5 V (Out)
Masse (Ground)

Arduino


SDA
Seriell I²C (SDA/SCL)
SCL
nversorgung
N
Z
X
S
GND
GND
SCL (/INT)
VDD
GND
Y
(ADDR) SDA
N
S
S
5 V (Out)
Masse (Ground)

I²C Bus – Inter-Integrated Circuit

Adressierung:

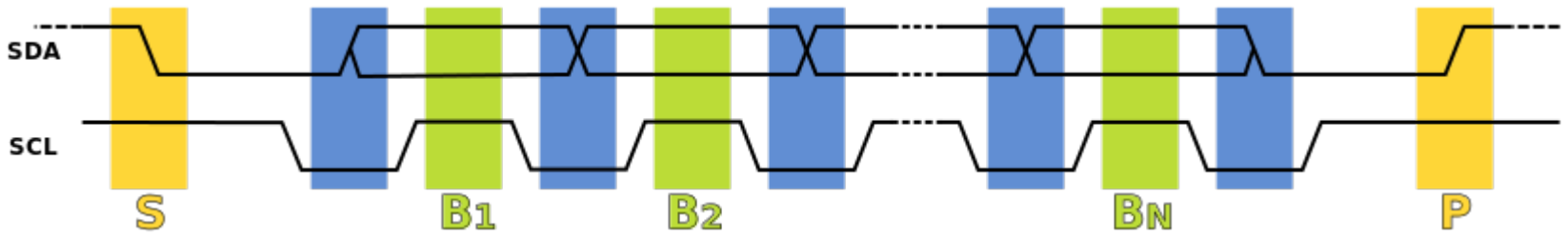
- Erstes vom Master gesendete Byte ist die Adresse des angesprochenen Chips
- Ersten 7 Bit sind die Adresse (112 Knoten, 16 Sonderadressen)
- In aktueller Norm auf 10 bit / 1136 Knoten erweitert
- Achtes Bit (R/W Bit) zeigt an, ober Master schreibt (low) oder liest (high)

- Weitere Bytes zur Datenübertragung (schreibend oder lesend)
- Zweites Byte ist oft eine Adresse für ein Register innerhalb des adressierten ICs

- Die korrekte Implementierung des Protokolls für die Kommunikation zwischen Mikrocontroller und Peripherie liegt in der Verantwortung des Programmierers
- Für viele ICs gibt es Bibliotheken (libraries)

I²C Bus – Inter-Integrated Circuit

- Bustakt (SCL) wird vom Master ausgegeben
- Start der Übertragung: fallende Flanke auf SDA während SCL = High
- Stop der Übertragung: ansteigende Flanke auf SDA während SCL = High
- 8 Bit: Daten
- Datenleitung SDA wechselt **nur** während SCL = Low
- Ausnahme: Start (S) und Ende (P): SDA wechselt während SCL = High
- Daten gültig während SCL = High



Von Marcin Floryan - Eigenes Werk, Gemeinfrei, https://commons.wikimedia.org/w/index.php?curid=1647146

Beispiel: vier Bytes über I²C Bus an Sensor schicken

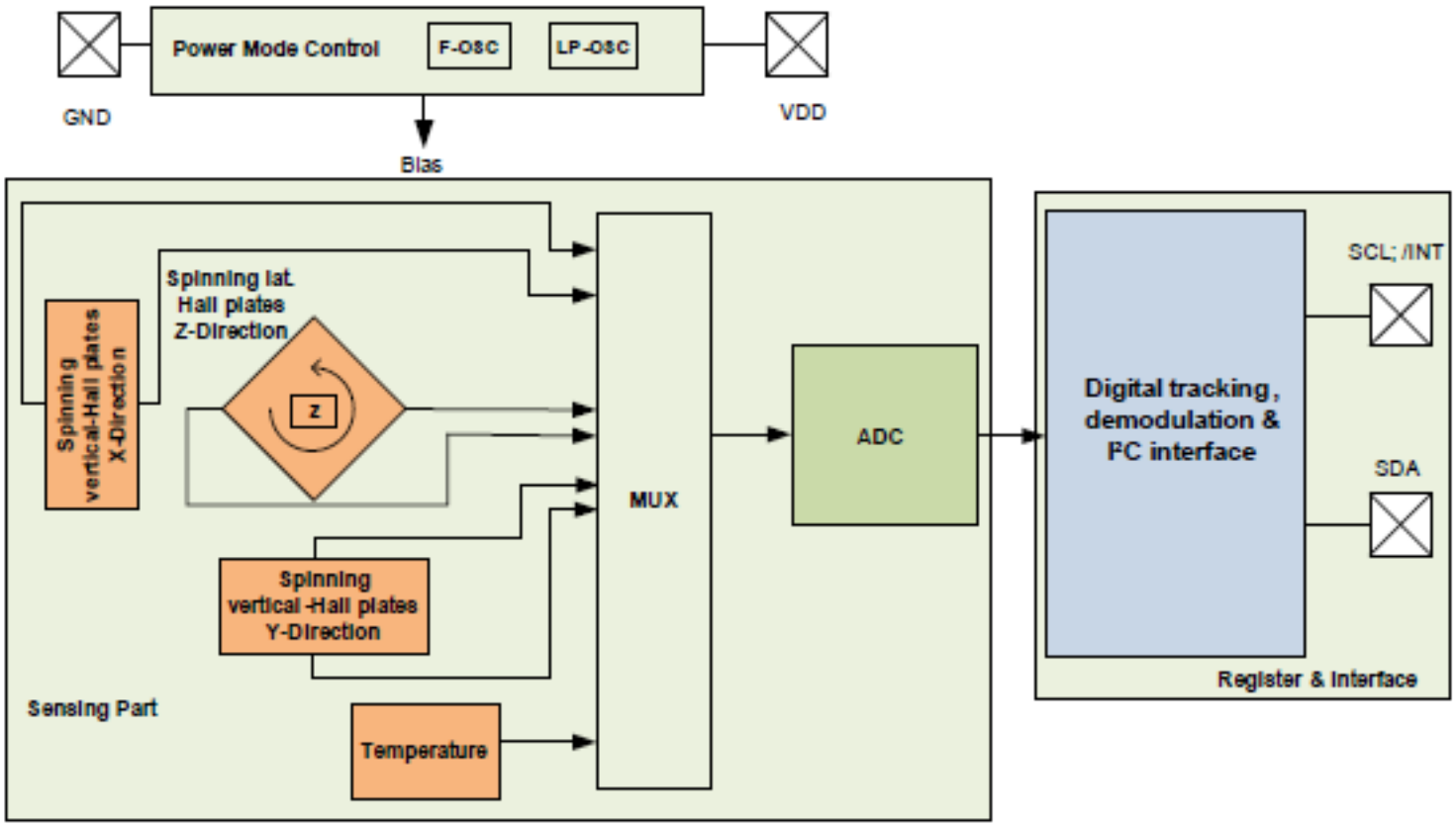
```
#include <Wire.h>                  // I2C Bibliothek

#define HallSensor 0x5E  // 7 bit I2C Bus-Adresse des Hall-Sensors

void setup()
{
     Wire.begin();                 // I2C Bus initialisieren
     Wire.setClock(100000);        // Taktfrequenz des I2C Bus festlegen
}

void loop()
{
    Wire.beginTransmission(HallSensor);  // Übertragung an HallSensor vorbereiten
    Wire.write(0b00000000);              // dieses vier Bytes sollen
    Wire.write(0b00000111);              // an den 3d Magnetfeldsensor
    Wire.write(0b00000000);              // übertragen werden
    Wire.write(0b00000000);
    Wire.endTransmission();              // Übertragung über I2C starten
}
```

```
#include <Wire.h>                  // I2C Bibliothek

#define HallSensor 0x5E  // 7 bit I2C Bus-Adresse des Hall-Sensors

void setup()
{
     Wire.begin();                 // I2C Bus initialisieren
     Wire.setClock(100000);        // Taktfrequenz des I2C Bus festlegen

     Serial.begin(9600);
}

void loop()
{
    Wire.beginTransmission(HallSensor);  // Übertragung an HallSensor vorbereiten
    Wire.write(0b00000000);              // dieses vier Bytes sollen
    Wire.write(0b00000111);              // an den 3d Magnetfeldsensor
    Wire.write(0b00000000);              // übertragen werden
    Wire.write(0b00000000);
    int error = Wire.endTransmission();  // Übertragung starten

    Serial.print("I2C error: ");         // zurückgegebenen Fehlercode ausgeben
    Serial.println(error);
    delay(50);
}
```

Beispiel: 10 Bytes von Sensor lesen

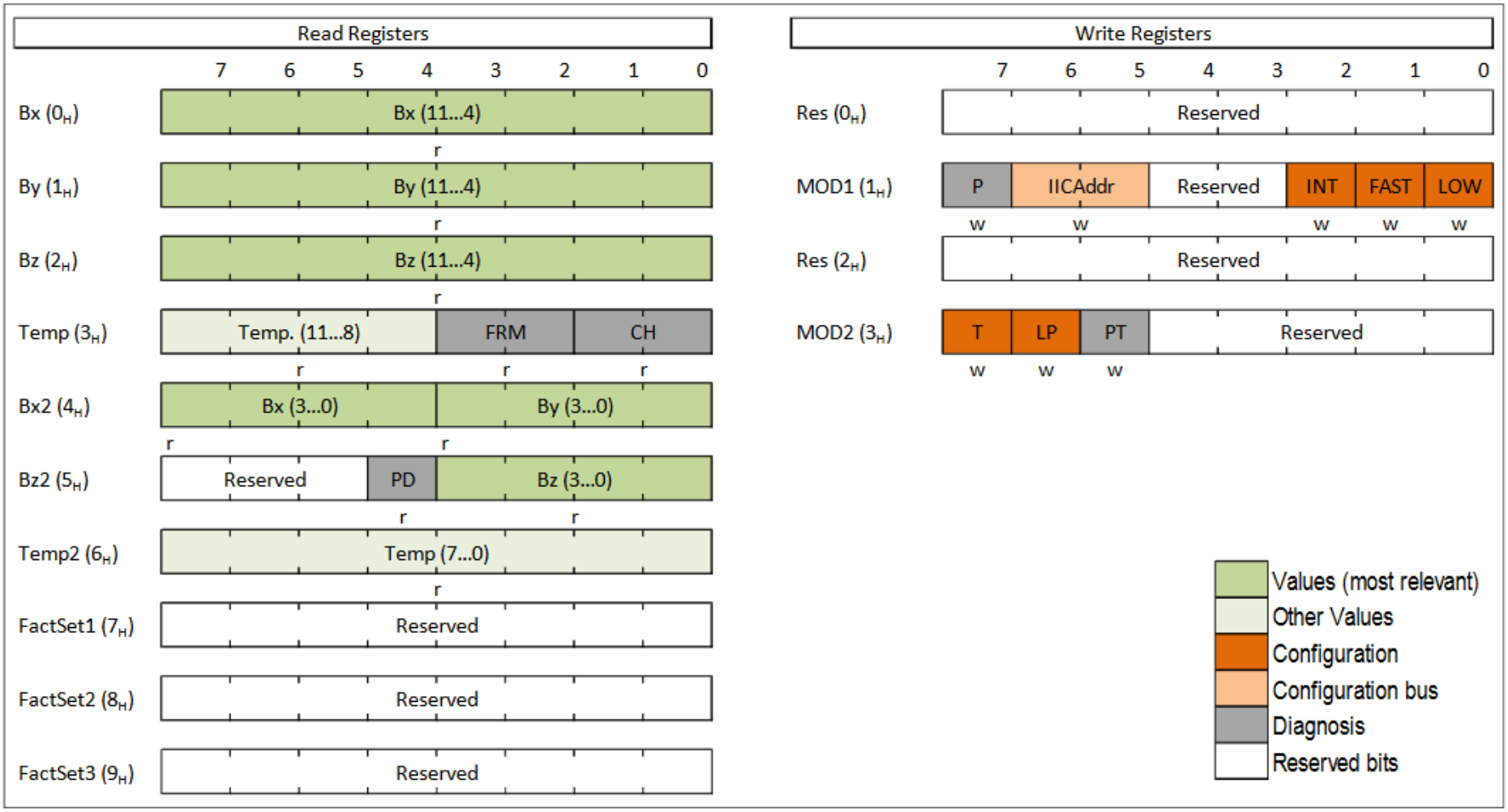
```
byte values[10];                    // Array mit 10 Byte
                                    // für die eingelesenen Werte (Register)


Wire.requestFrom(HallSensor, 10, true); // 10 bytes von Sensor lesen
                                        // „true“ bedeutet, dass die
                                        // Übertragung abgeschlossen
                                        // wird

for(int count = 0; count <=10; count++)
        {
        values[count] = Wire.read();  // Werte aus Buffer
        }                             // in Array übertragen
```

Magnetfeldsensor

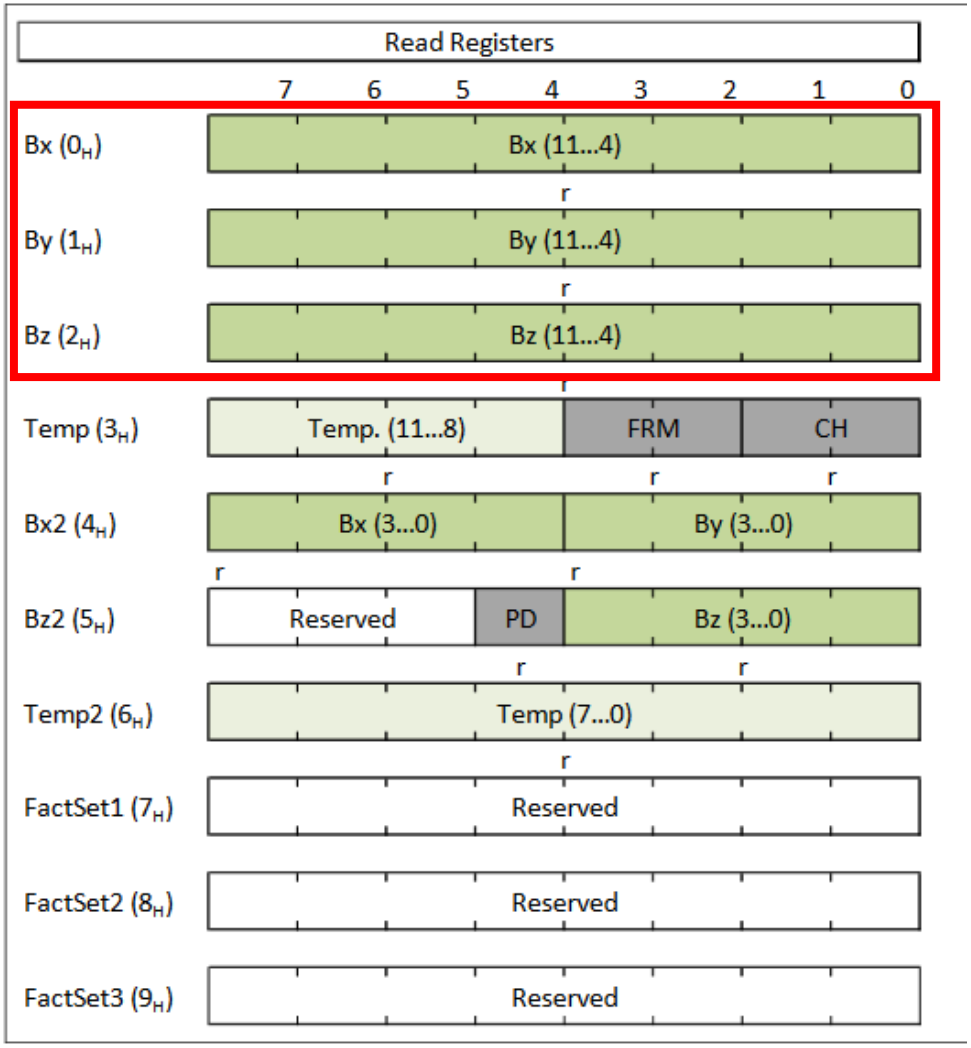


Block Diagram

Magnetfeldsensor: interne Register



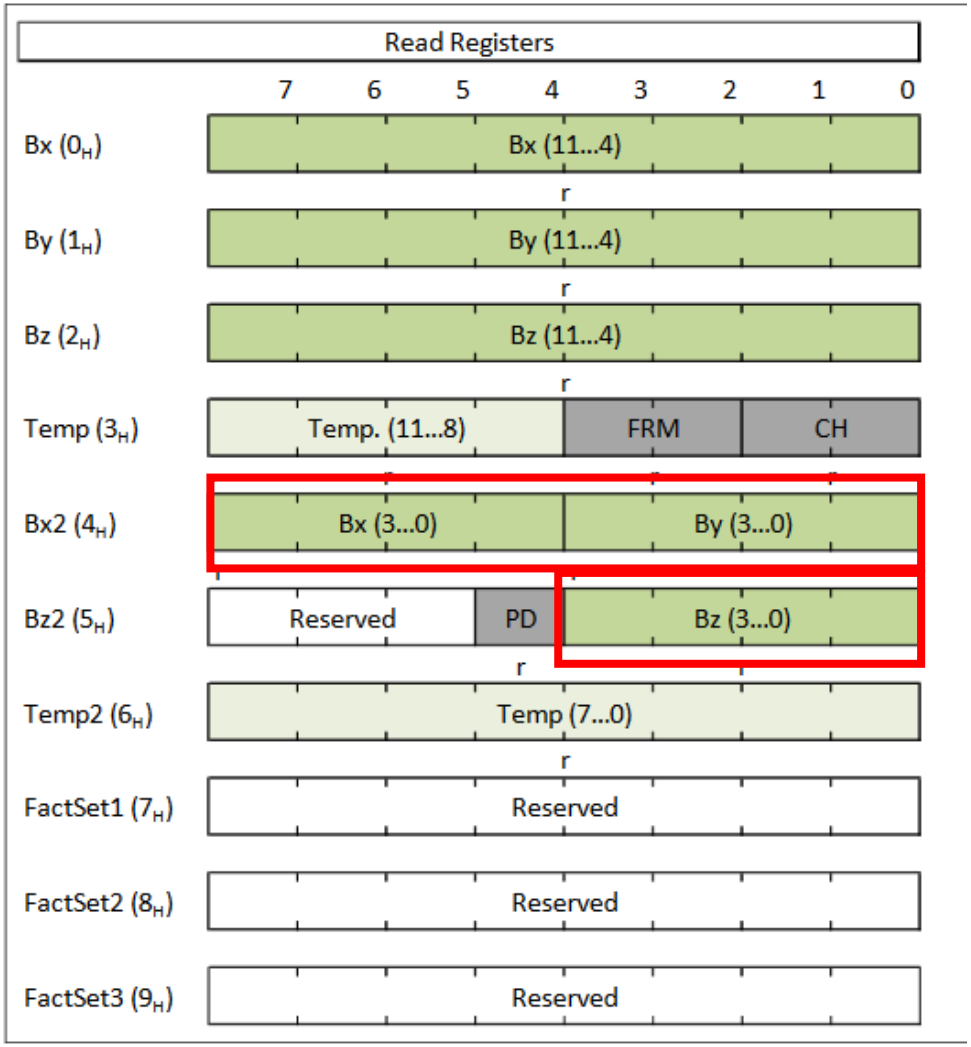
Magnetfeldsensor über I²C ansteuern



Enthalten die 8 MSB's des magnetischen Flusses in x, y und z-Richtung

- 12 bit Auflösung: 98 µT / LSB
- 8 bit Auflösung: 1.56 mT/LSB

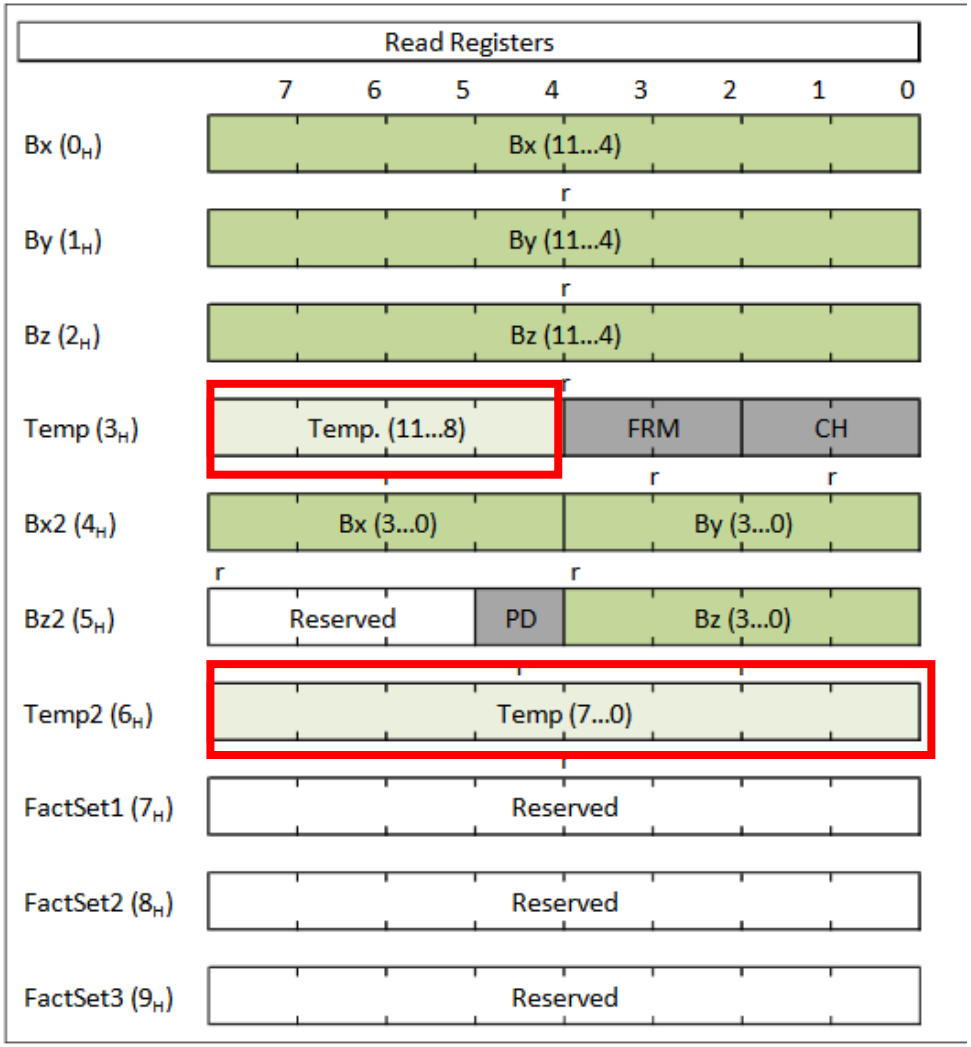
Magnetfeldsensor über I²C ansteuern



- 12 bit Auflösung: 98 µT / LSB
- 8 bit Auflösung: 1.56 mT/LSB

Enthalten die 4 LSB's des magnetischen Flusses in x, y und z-Richtung

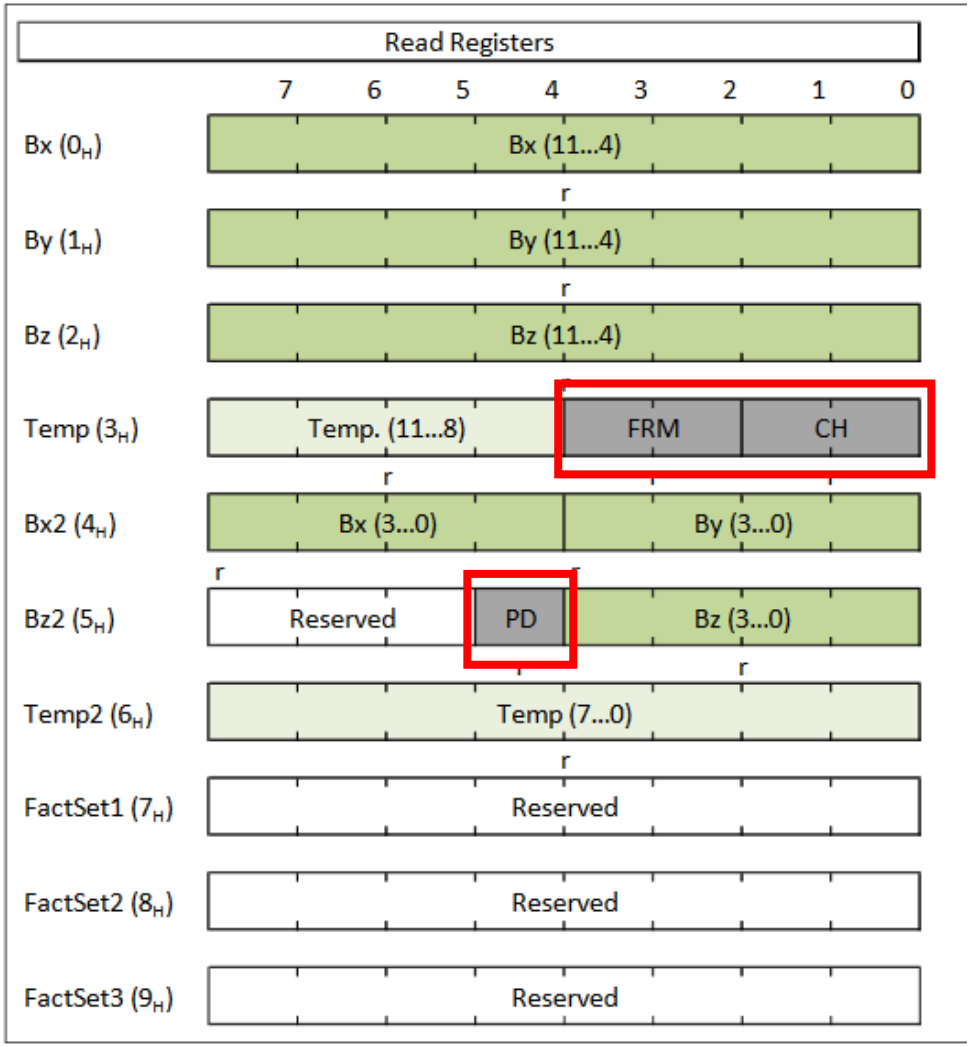
Magnetfeldsensor über I²C ansteuern



Enthält die 4 MSB's der Temperatur

Enthält die 8 LSB's der Temperatur

Magnetfeldsensor über I²C ansteuern

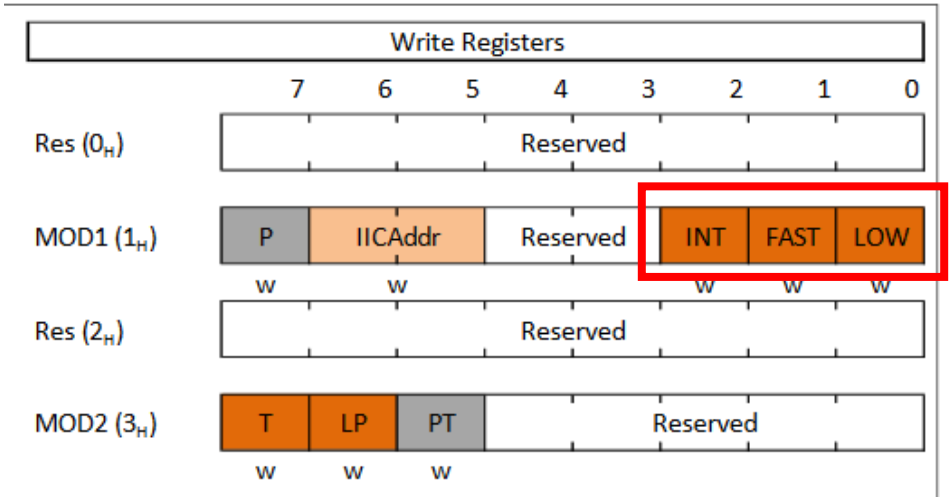


FRM: „Frame-Counter“, erhöht sich um eins, wenn Messzyklus vollendet

CH: „Channel“, zeigt an, welche Messung gerade durchgeführt wird (00 wenn Zyklus abgeschlossen)

PD: „Power-Down-Flag“, zeigt an, ob Berechung läuft (1) oder abgeschlossen ist (0)

Magnetfeldsensor über I²C ansteuern



INT: „Interrupt enable“, wenn aktiv (1), wird ein Interrupt-Signal über die Clock-Leitung geschickt, sobald ein Messzyklus komplett ist

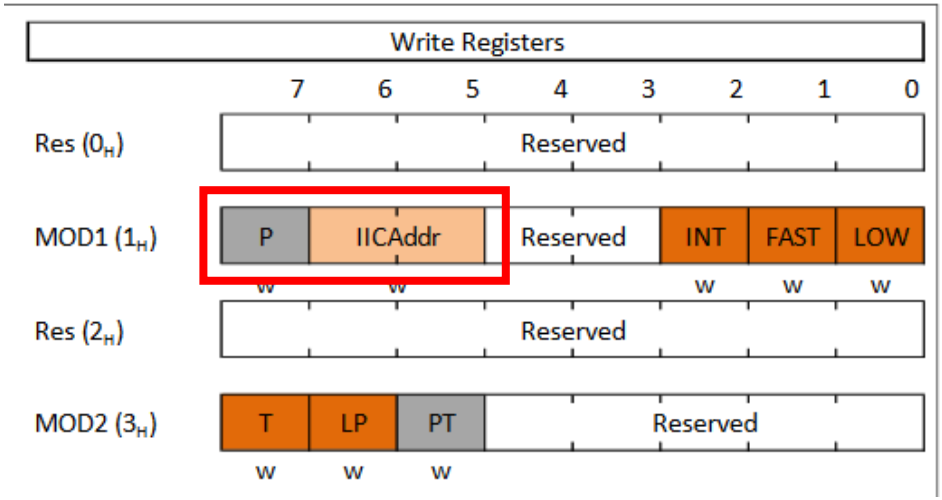
FAST: „Fast-Mode“, aktivieren, um im Fast-Mode zu messen *

LOW: „Low-Power-Mode“, aktivieren, um im Low-Power-Mode zu messen **

* Lesen, während der nächste Messzyklus schon läuft. Liefert nur erste 3 Bytes des Read-Registers.

** Misst mit 100 Hz, begibt sich zwischen Messungen in Ruhezustand (Power Down)

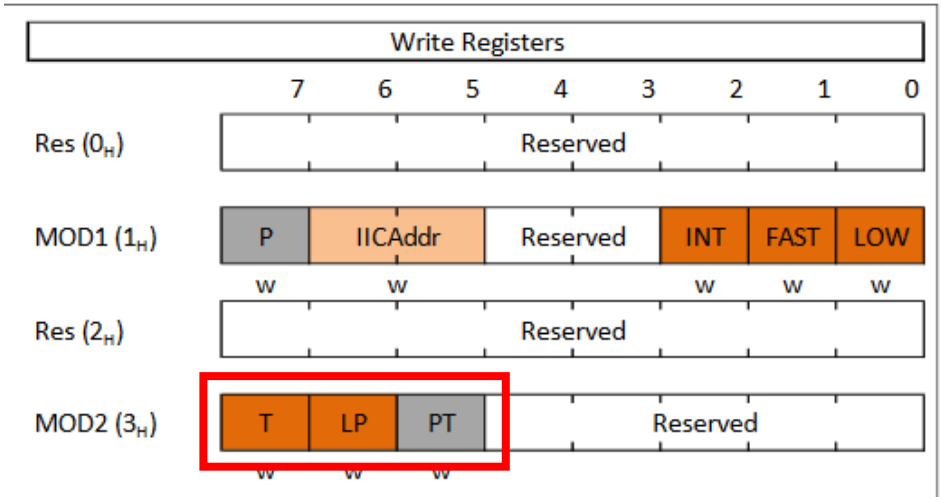
Magnetfeldsensor über I²C ansteuern



P: „Parity bit“, Vergleichswert, der vom Master bei jedem „Write“ gesendet wird

IICAddr: je nach Konfiguration erhält der Sensor eine andere Adresse am Bus

Magnetfeldsensor über I²C ansteuern



T: „Temperature Measurement enabled“, Temperaturmessung findet statt wenn 0

LP: wenn 0: „Low Power Period“,
wenn 1 „ultra low power“

PT: „Parity Test Enabled“, Abgleich des Parity-Bits findet statt wenn PT = 1

Magnetfeldsensor über I²C ansteuern

```
#include <Wire.h>                 // I2C Bibliothek
#define mag_sensor (0x5E)         // I2C Adresse des Sensors

byte values[10];                  // Array mit 10 Byte
                                  // zum Lesen der Register

void setup() {
        Wire.begin();
        }

void loop() {
  Wire.beginTransmission(mag_sensor); // Übertragung vorbereiten
                                      // und Adresse angeben
  Wire.write(0x0);                    // Nacheinander 4 Write-Bytes
  Wire.write(0x7);                    // konfigurieren
  Wire.write(0x0);
  Wire.write(0x0);
  Wire.endTransmission();             // Übertragung starten
```

Magnetfeldsensor über I²C ansteuern

```
Wire.requestFrom(mag_sensor, 10, true); // 10 bytes von mag_sensor

for(int count = 0; count <=10; count++)
        {
        values[count] = Wire.read();  // Werte vom I2C Bus lessen
        }                             // und in Array schreiben



// die ausgelesenen Werte müssen dann weiterverarbeitet werden…
// z.B. in diesem Fall
int x = 16*((int)((char)values[0]))+(int)((values[4]&0xF0) >> 4);
```

Magnetfeldsensor über I²C ansteuern

Alternative mit `while(Wire.available()`

```
Wire.requestFrom(mag_sensor, 10, true); // 10 bytes von mag_sensor

count = 0;
while(Wire.available())        // solange Daten empfangen wurden
        {
        values[count] = Wire.read();  // Werte vom I2C Bus lessen
        count = count + 1;            // und in Array schreiben
        }


// die ausgelesenen Werte müssen dann weiterverarbeitet werden…
// z.B. in diesem Fall
int x = 16*((int)((char)values[0]))+(int)((values[4]&0xF0) >> 4);
```

I²C Bus – Inter-Integrated Circuit

Informationen zum I²C Bus: siehe Vorlesung

Erster Schritt:
4 beliebige Bytes an beliebige Adresse (z.B. 0x5E) schicken, ohne dass ein Sensor mit dem Bus verbunden ist, um die elektrischen Signale auf den Leitungen SCL und SDA anzusehen; das Programm dazu steht auf der nächste Folie.

```
#include <Wire.h>                  // I2C Bibliothek

#define HallSensor 0x5E  // 7 bit I2C Bus-Adresse des Hall-Sensors

void setup()
{
     Wire.begin();                 // I2C Bus initialisieren
     Wire.setClock(100000);        // Taktfrequenz des I2C Bus festlegen

     Serial.begin(9600);
}

void loop()
{
    Wire.beginTransmission(HallSensor);  // Übertragung an HallSensor vorbereiten
    Wire.write(0b00000000);              // dieses vier Bytes sollen
    Wire.write(0b00000000);              // an den 3d Magnetfeldsensor
    Wire.write(0b00000000);              // übertragen werden
    Wire.write(0b00000000);
    int error = Wire.endTransmission();  // Übertragung starten

    Serial.print("I2C error: ");         // zurückgegebenen Fehlercode ausgeben
    Serial.println(error);
    delay(10);
}
```

I²C Bus – Inter-Integrated Circuit

Die Funktion sollte den Fehler „2“ ausgeben, weil kein „Slave“ am I²C Bus ist, der den Empfang „quittiert“ (2: received NACK on transmit of address).

Fehler- oder Status-Codes siehe
file:///C:/Program%20Files%20(x86)/Arduino/reference/www.arduino.cc/en/Reference/Wire.html

Wire.endTransmission() gibt folgende Fehler- oder Status-Codes
byte, which indicates the status of the transmission:
0: success
1: data too long to fit in transmit buffer
2: received NACK on transmit of address
3: received NACK on transmit of data
4: other error

„NACK“ steht für „not acknowledged“ – nicht bestätigt

I²C Bus – Inter-Integrated Circuit

Beobachtungen:

- Nur die Adresse wird ausgegeben
- Wegen Fehler (NACK) bei Adresse werden die folgenden 4 Bytes nicht ausgegeben
- Die Signale sind stark verrundet; die Ursache dafür sind die zu hohen internen Pull-Up Widerstände des Arduino
- Verbinden Sie die SCL und SDA Leitungen mit je einem $2{,}2\ \mathrm{k\Omega}$ Widerstand mit 5 V; die Signale auf SCL und SDA sollten dann in nahezu senkrechte Flanken haben.

3d Hall-Sensor anschließen

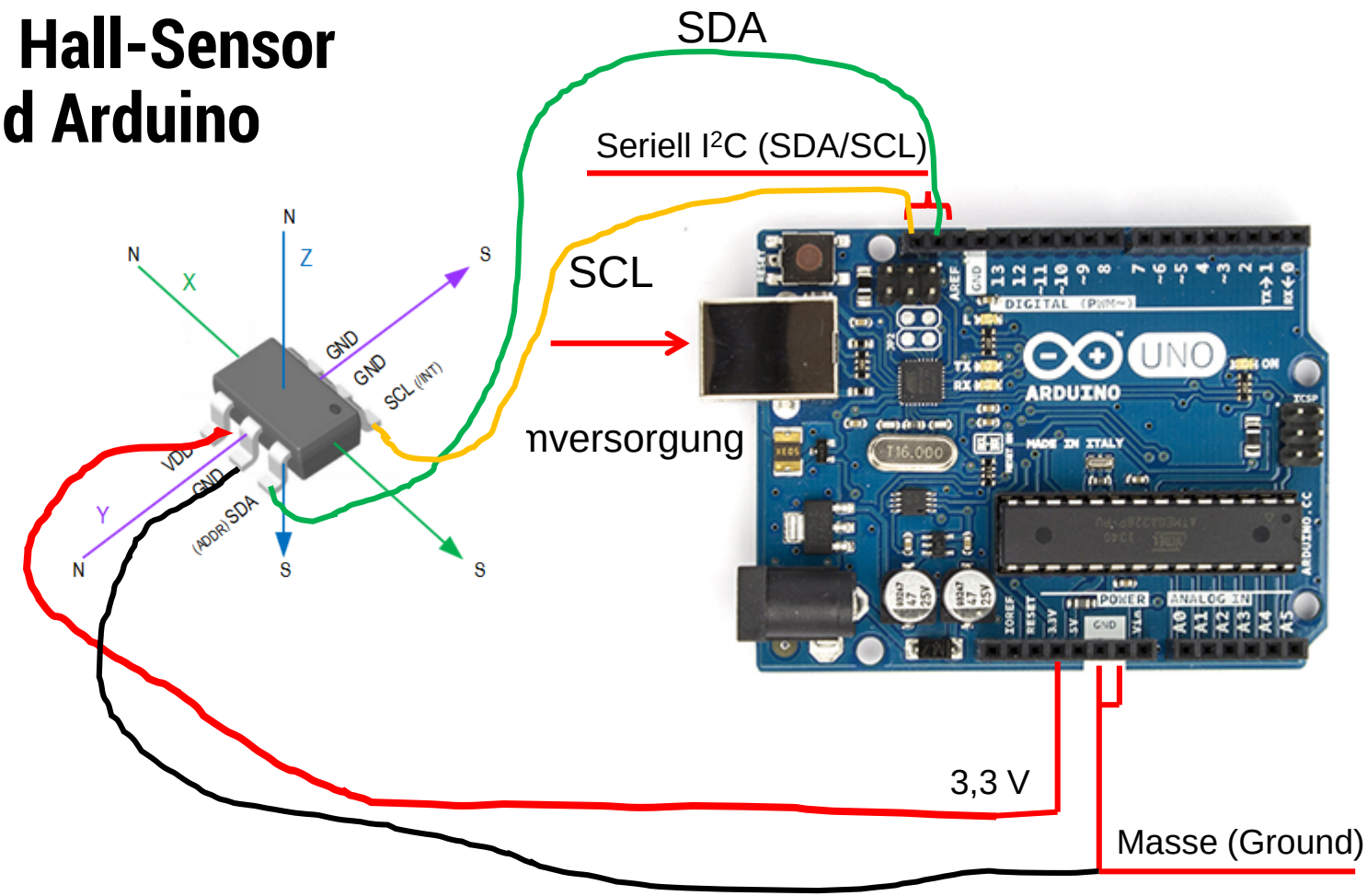
Vorsicht:

Der Hall-Sensor TLV493D von Infineon hat

3,3 V Spannungsversorgung

5 V Versorgungsspannung zerstören den Sensor

3d Hall-Sensor und Arduino



Der 3,3 V Pin liegt neben dem 5 V Pin. Aufpassen!

Nebenbemerkung: SCA und SCL vertragen 5 V

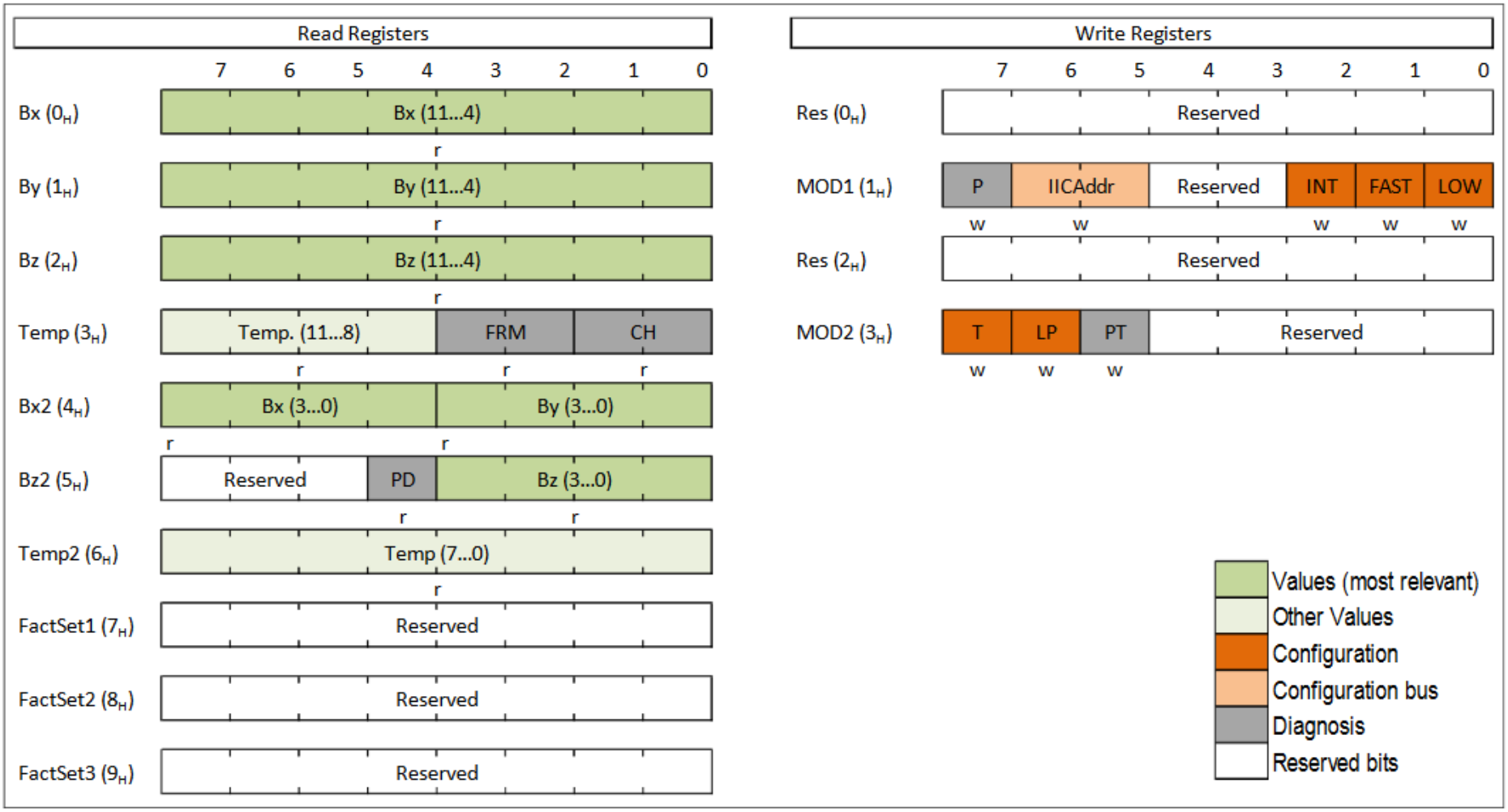
Ein kleiner Sketch, der alle Adressen 17 … 127 nacheinander abscannt, ob ein Baustein (Sensor) antwortet:

```
#include und setup() wie bei obigem Sketch

void loop()
{
    Serial.println("Alle Adressen nacheinander abfragen...");

      for(byte address = 17; address < 127; address++ )
      {
        Wire.beginTransmission(address); // Adressen 17 bis 127 nacheinander
        int error = Wire.endTransmission();  // "ansprechen"

        if (error == 0)                  // kein Fehler --> Baustein vorhanden
        {
          Serial.print("I2C Baustein bei Adresse 0x");
          Serial.println(address,HEX);
        }
      }
      delay(3000);           // nächster Scan nach 3 Sekunden
}
```

Den Hall-Sensor TLV493D initialisieren, in dem in die vier Schreib-Register Werte geschrieben werden.
Im einfachsten Fall ist das viermal die 0x00 (siehe Datenblatt).

```
#include und setup() wie bei obigem Sketch

void loop()
{
    Serial.println(„Hall-Sensor TLV493D initialisieren...");

    Wire.beginTransmission(address); // Übertragung an Hall-Sensor vorbereiten
    Wire.write(0x00);                // diese vier Bytes sollen
    Wire.write(0x00);                // übertragen werden
    Wire.write(0x00);
    Wire.write(0x00);
    int error = Wire.endTransmission();  // Übertragung starten

    Serial.print("I2C Status oder Fehler: ");
    Serial.println(error);

    delay(3000);                         // wiederholen nach 3 Sekunden
}
```

Register des TVL493D



Nächste Aufgabe:

Nur 8-Bit Werte von Bx, By, und Bz lesen.

Hall-Sensor TLV493D im Fast Modus initialisieren (in setup()).
Immer nur die ersten drei Register lesen (in loop()).
Bx, By, Bz auf seriellen Plotter ausgeben.

```
#include <Wire.h>                  // I2C Bibliothek

#define HallSensor 0x5E  // 7 bit I2C Bus-Adresse des Hall-Sensors

void setup()
{
     Wire.begin();                 // I2C Bus initialisieren
     Wire.setClock(100000);        // Taktfrequenz des I2C Bus festlegen

     Serial.begin(9600);

    Wire.beginTransmission(HallSensor);
    Wire.write(0x00);              // Initialisierung
    Wire.write(0x02);              // im Fast Modus
    Wire.write(0x00);
    Wire.write(0x00);
    Wire.endTransmission();
}
```

```
void loop()
{
    char Bx, By, Bz;      // char ist eine vorzeichenbehaftete 8Bit-Zahl

    Wire.requestFrom(HallSensor, 3, true); // 3 Bytes vom Hall-Sensor lesen

    Bx = Wire.read();      // die Werte für Bx, By und Bz
    By = Wire.read();      // aus dem Buffer in Variablen
    Bz = Wire.read();      // übertragen

    Serial.print((int)Bx); // "cast" auf (int), damit "char" nicht
    Serial.print(" ");     // als ASCII Zeichen ausgegeben wird
    Serial.print((int)By);
    Serial.print(" ");
    Serial.println((int)Bz);

    delay(20);
}
```

Nächste Aufgabe:

12-Bit Werte von Bx, By, und Bz lesen.

Hall-Sensor TLV493D im Low (Power) Modus initialisieren (in setup()).
Alle zehn Register lesen (in loop()).
Bx, By, Bz auf seriellen Plotter ausgeben.

```
#include <Wire.h>                  // I2C Bibliothek

#define HallSensor 0x5E  // 7 bit I2C Bus-Adresse des Hall-Sensors

void setup()
{
     Wire.begin();                 // I2C Bus initialisieren
     Wire.setClock(100000);        // Taktfrequenz des I2C Bus festlegen

     Serial.begin(9600);

    Wire.beginTransmission(HallSensor);
    Wire.write(0x00);              // Initialisierung
    Wire.write(0x01);              // im Low (Power) Modus: 100 Hz Rate
    Wire.write(0x00);
    Wire.write(0x00);
    Wire.endTransmission();
}
```

```
void loop()
{
    int xValue, yValue, zValue; // 12-Bit Integer-Werte
    float Bx, By, Bz;       // B-Felder als Fließkommazahlen
    byte values[10];        // Array als Buffer zum Zwischenspeichern

    Wire.requestFrom(HallSensor, 10, true); // 10 Bytes vom Hall-Sensor lesen

    for(int count = 0; count <=10; count++) {
        values[count] = Wire.read(); }  // Werte in Array übertragen


    … Berechnung von xValue, yValue, zValue, Bx, By, und Bz …


    Serial.print(Bx);
    Serial.print(" ");
    Serial.print(By);
    Serial.print(" ");
    Serial.println(Bz);

    delay(20);
}
```

8 Bit und 4 Bit zu 12 Bit zusammensetzen … hier für den x-Wert:

```
xValue = 16*((int)((char)values[0]))
               + (int)((values[4] & 0b11110000) >> 4);
```

- 0xF0 entspricht 11110000, & ist ein bitweiser Operator, also steht (values[4]&0xF0) für die ersten 4 Bits (MSB) von values[4], diese müssen noch nach rechts geschoben werden

- ((int)((char)values[0])) wandelt die byte - Daten aus values[0] entsprechend dem two's complement mit Vorzeichen in einen integer um

- die Multiplikation eines Integers mit 16 entspricht einer bitweisen Verschiebung nach links um vier Stellen in Binärdarstellung

B-Felder (in Milli-Tesla) berechnen:

```
Bx = xValue*0.098;   // Umrechnung laut Datenblatt mit 0.098 mT/LSB
```

Analog für die beiden anderen Richtungen.

Aufgaben / Anwendungen:

Joystick
Winkelsensor
Drehratensensor

LED mit Magnet dimmen

Servo mit Magnet steuern

u.v.m.