# Arduino und Amateurfunk

Workshop USKA Sektion Solothurn Okt 2021

HB9BFD Roland

# Einleitung

- Unterrichtsziel:

  « …. nötige Kompetenz für die Programmierung von Arduinos erwerben. Programme vom Internet anpassen und verwenden können.»

# Wer ist HB9BFD?

- Geb 10.5.1955 (Vater war Papierarbeiter an einem Kalander).
- Aufgewachsen in Gerlafingen/SO. Zur Schule zusammen mit Jürg HB9BIN und Ernst HB9PVI.Wie man sieht, ist aus unserem Jahrgang in Gerlafingen eine richtige Funkergruppe entstanden!
- Kantonsschule in Solothurn mit Matura 1974.
- 31.5.1975 VHF/UHF-Lizenz, als HB9MLV von Gerlafingen aus QRV mit 2m-SemcoMoto von HB9MFM und 70cm Röhrengerät von HB9BCA.
- Frühling 1975 RS in Jassbach, EGM, später EKF Abt46 mit Morsedrill bis zu Tempo 100 (leider alles wieder verlernt)! WK's jeweils als VHF Peiler/Aufklärer.
- Oktober 1975 CW-Prüfung und Wechsel des Rufzeichens zu HB9BFD.
- Kauf einer nigelnagelneuen Drake-Line R4C/T4XC für damals Fr. 4000.- (Sammelbestellung mit Emil HB9BAT).
- Studium Elektrotechnik an der ETH Zürich mit Abschlussdiplom Frühling1980.
- Ab 1980 tätig bei Holcim als Elektroingenieur bis 2016, dann frühzeitig in Rente und neues Hobby als Besucherführer im KKG Gösgen.

# Meine Highlights während den 36 Jahren Holcim


Diverse SPSn ab 1980 (noch mit Kernspeicher)
bis zur
Familie Siemens S7


Allen-Bradley

Modicon PLC

Texas Instruments

ABB

SIEMENS

april

Rockwell Automation

WAGO

BECKHOFF
New Automation Technology

# Wieso Arduinos?

- Günstig (China –Clone für Fr. 2.- erhältlich)
- Grosse Community im Internet
- Viele Programme und Libraries (Bibliotheken)
- Grosse Anzahl von Sensoren und Shields
- Schnelle Realisierung von Ideen
- Leicht erlernbare Programmiersprache
- Viele Projekte bereit zum Anpassen auf meine Bedürfnisse

# Inhalt der Präsentation (4 Kapitel)

- Kapitel 1: Arduino Grundlagen
  - Unterschiede SPS – MikroController
  - Funktion, Hardware
  - Software
  - Grundlagen der Programmierung
  - Übungsbeispiel Blinklampe

- Kapitel 2: Bussysteme mit Arduino
  - Verwendung von Bibliotheken
  - «Bus Systeme» mit Arduino: Seriell, USB, $I^2C$, 1-Wire
  - Übungsbeispiel LCD Anzeige mit Arduino

# Fortsetzung: Inhalt der Präsentation (4 Kapitel)

- Kapitel 3: Sensorik
  - Sensorik mit Arduino
  - Temperaturmessung
  - Übungsbeispiel Dallas Chip oder Ti
- Kapitel 4: Kommunikation
  - Kommunikation mit MQTT
  - Übermitteln von Messwerten an den Broker
- Kapitel 5: Anwendungs-Beispiele im Amateurfunk

# Kapitel 1: Arduino Grundlagen

# Kapitel 1: Arduino Grundlagen

- Inhalt:
  - Unterschiede SPS – MikroController (Arduino HW)
  - Funktion, Hardware
  - Software
  - Grundlagen der Programmierung
  - Übungsbeispiel Blinklampe

# Kapitel 1 : Arduino Grundlagen

- Lernziel:
  - Du bist in der Lage zu entscheiden, welche Lösung für dein Problem am meisten Sinn macht (Kosten, Performance, Sicherheit, Ersatzteile, Dokumentierbarkeit, Lifetime, ….)
  - Umgang mit den Arduino Bespielen vom Internet und der Hardware (Aliexpress, Bangood, DealExtreme,…..).
  - Blinklampe programmiert mit dem Nano-Board

# Unterschiede SPS zu Mikrokontrollern

- SPS

CHF 1’199.95


Allen-Bradley 1774 anno 1980

- Mikrocontroller

CHF 3.01
Geekcreit UNO R3
ATmega328P


Synertek SYM-1
anno 1976

# Unterschiede SPS zu Mikrokontrollern

- SPS
  - Betriebsfertiges Gerät, bereit zur Programmierung
  - Div Eingangs- und Ausgangskarten mit Schraubklemmen o.ä.
  - E/A-Spannung meist abgesichert, 12VDC…230VAC
  - 0/4…20mA E/A-Karten Schnelle Zählerkarten
  - Div E/A-Feldbussysteme direkt unterstützt.
- Mikrocontroller
  - Universell, muss zuerst mit Hardware ergänzt werden.
  - Keine Klemmen, sondern nur Pins im 2.54 Raster
  - E/A(GPIO) Spannung ist meist 5VDC oder 3.3VDC bei Kurzschluss ist Controller defekt!!
  - Meist nur USB oder Ethernet vorhanden.
  - Keine dezentrale Peripherie so einfach möglich.

# Unterschiede SPS zu Mikrokontrollern (Forts.)

- SPS
  - Programmierung «einfach» ohne C-Kenntnisse.
  - KOP, FUP, AML,SCL …..
  - Programm läuft in einem Zyklus ab: Zuerst Eingänge (Abbild), Verarbeitung, dann Resultat auf Ausgänge schreiben und wieder von Vorne beginnen→Loop.
  - Online Prog-Änderungen.
  - Klar strukturierte Progr.
- Mikrocontroller
  - Meist in C-ähnlicher Programmiersprache. Minimale Kenntnisse der Informatik notwendig.
  - Es gibt kein I/O-Image (Abbild). Ein/Ausgänge werden direkt angesprochen.
  - Programm kann nur offline geändert werden

# Unterschiede SPS zu Mikrokontrollern (Forts.)

- SPS
  - Früher relativ langsam. Zykluszeit in der Regel im Bereich von einigen ms bis zu 500ms (Task-abhängig).
  - Eher teuer für kleine Anwendungen
- Mikrocontroller
  - Schnell (bis zu 100 x schneller als eine SPS) im µs- Bereich
  - Sehr preisgünstige Lösungen möglich

# Und was ist der Unterschied zu einem Raspberry?

- Raspberry:
  - Linux Betriebssystem
  - Eigentlich kompletter PC
  - Schnelle Signalverarbeitung
  - Viele Programme gleichzeitig laufend
  - Video Schnittstelle
  - USB Anschlüsse
  - Ethernet
  - 40 pin GPIO Header
- Mikrokontroller Board Bsp ATmega:
  - Bootloader, in C++ programmierbar
  - I/O's mit 5VDC
  - Programm startet automatisch bei poweron
  - Kein Ethernet direkt unterstützt, kein Bildschirm
  - Nur offline programmierbar
  - Bis zu 55 Pins

# Funktion, Hardware (Arduinos)

- Open-Source Soft- und Hardware
- Einfaches E/A Board meist basierend auf Atmel-AVR ATmega2560, ATmega328 oder ARM Cortex-M3
- Spannungsversorgung via USB oder extern 7-12VDC
- Intern entweder 5VDC oder 3.3VDC
- Programmierung über USB (oder ältere Boards via RS232)
- Shields sind sogenannte Erweiterungen, die direkt aufgesteckt werden.

# Funktion, Hardware (Arduinos)

- ARDUINO© ist ein Open-Source-Prototypen-System, das die Herren Massimo Banzi und David Cuartielles am Interaction Design Institute in Ivrea entwickelt und nach einem lokalen Potentaten (vgl. Arduin von Ivrea, König von 1002 bis 1014 n. Chr.) genannt haben.
  (Gerüchten zur Folge könnte Namensgeber auch die gleichnamige Kneipe neben dem Institut gewesen sein !)

# Funktion, Hardware (Arduinos)

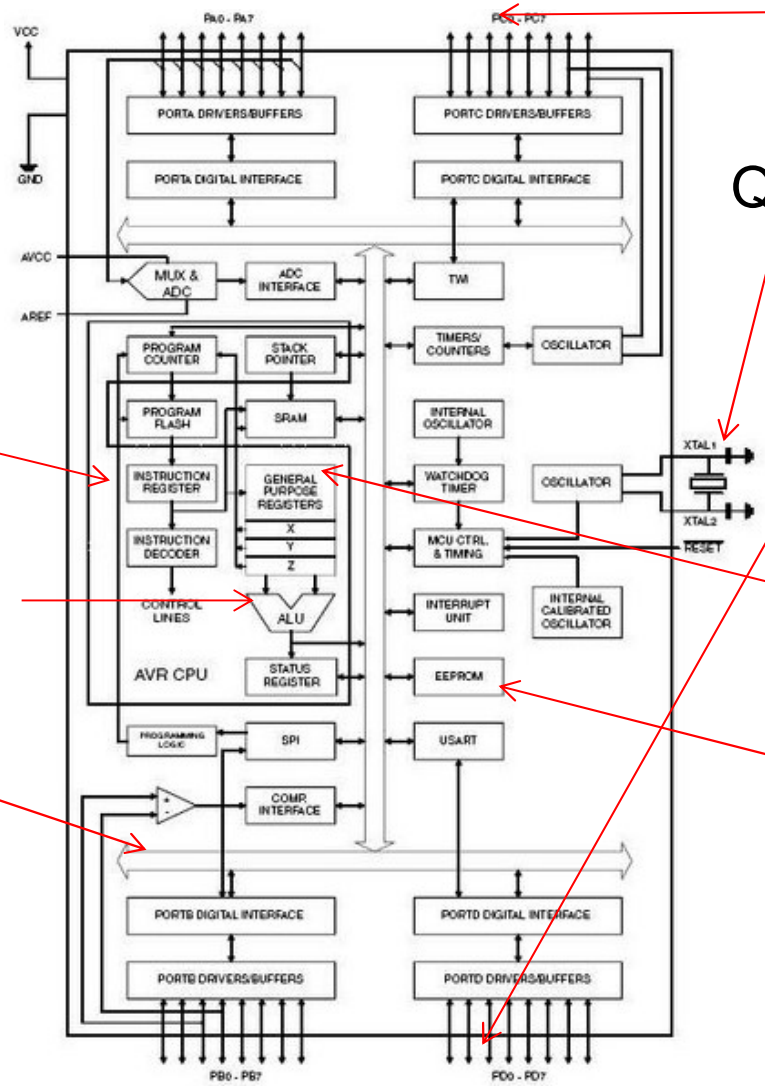
- Ursprünglich sollte es ein einfaches System sein, das Designer und Künstler anregen sollte ihre Objekte intelligent zu animieren (Licht, Farbe, Ton, Bewegung...)
  → Möglichst einfach !
- Weil Studenten mit diesem System arbeiten sollten, musste es preisgünstig sein.
- Es sollten übergreifende Systeme entstehen, die mit Windows, LINUX, Mac zusammenarbeiten können.
  →Entwicklungsumgebung arbeitet unter all diesen Betriebssystemen

# Funktion, Hardware (Arduinos)

- Programmiersprache auf Controller-Ebene ist Processing (C/C++ Abkömmling) mit zusätzlichen vereinfachenden Bibliotheken
  → einfache Anwendung der Sprache C !
- Der ARDUINO© wird über die USB-Schnittstelle angekoppelt (Datenaustausch, Steuerung)

# Funktion, Hardware

Bsp:Atmega328




I/O ports

Quarz

Bsp:Atmega2560

Register

**A**rithmetic **L**ogic **U**nit (ALU)

Speicher (RAM)

Bus

Speicher (EEProm)

# Hardware

- AVR 8-Bit RISC Microcontroller/Processor produziert von Atmel.
- Harvard-Architektur: Programm- und Datenspeicher separat (findet man meist auch bei den SPS).

**ATMEGA 328P-AU** MCU, ATMega AVR RISC, 32 KB, 20 MHz, TQFP-32

**Artikel-Nr.:** ATMEGA 328P-AU

**2,20 €**

inkl. gesetzl. MwSt. zzgl. Versandkosten

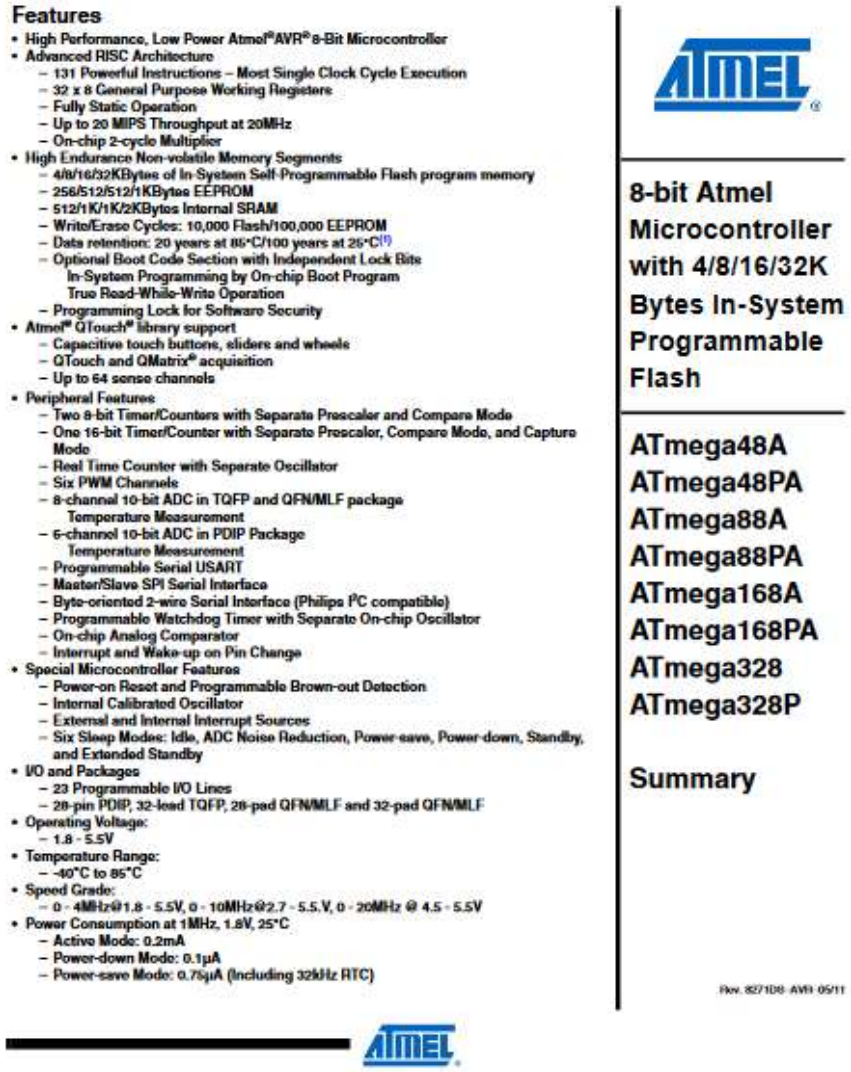
ab Lager, Lieferzeit: 1-2 Werktage

Stück | in den Warenkorb

RISC 8-Bit AVR, 32KB Programmspeicher, 2KB SRAM und 1KB EEPROM, 6 Analog I/O, 14 Digital I/O

3 Timer (8-Bit x 2 & 16Bit x1), SPI und I2C Bus, USI, -40…85°C, 28 Pins, 3.3V ….5VDC

# Hardware; Atmel Data Sheet



**Features**

- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32KBytes of In-System Self-Programmable Flash program memory
  - 256/512/512/1KBytes EEPROM
  - 512/1K/1K/2KBytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix® acquisition
  - Up to 64 sense channels
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I²C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 4MHz@1.8 - 5.5V, 0 - 10MHz@2.7 - 5.5V, 0 - 20MHz @ 4.5 - 5.5V
- Power Consumption at 1MHz, 1.8V, 25°C
  - Active Mode: 0.2mA
  - Power-down Mode: 0.1µA
  - Power-save Mode: 0.75µA (Including 32kHz RTC)

**8-bit Atmel Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash**

**ATmega48A**
**ATmega48PA**
**ATmega88A**
**ATmega88PA**
**ATmega168A**
**ATmega168PA**
**ATmega328**
**ATmega328P**

**Summary**

Rev. 8271D–AVR–05/11

https://cdn-reichelt.de/documents/datenblatt/A200/DS_AT168A_PA_328_P.pdf

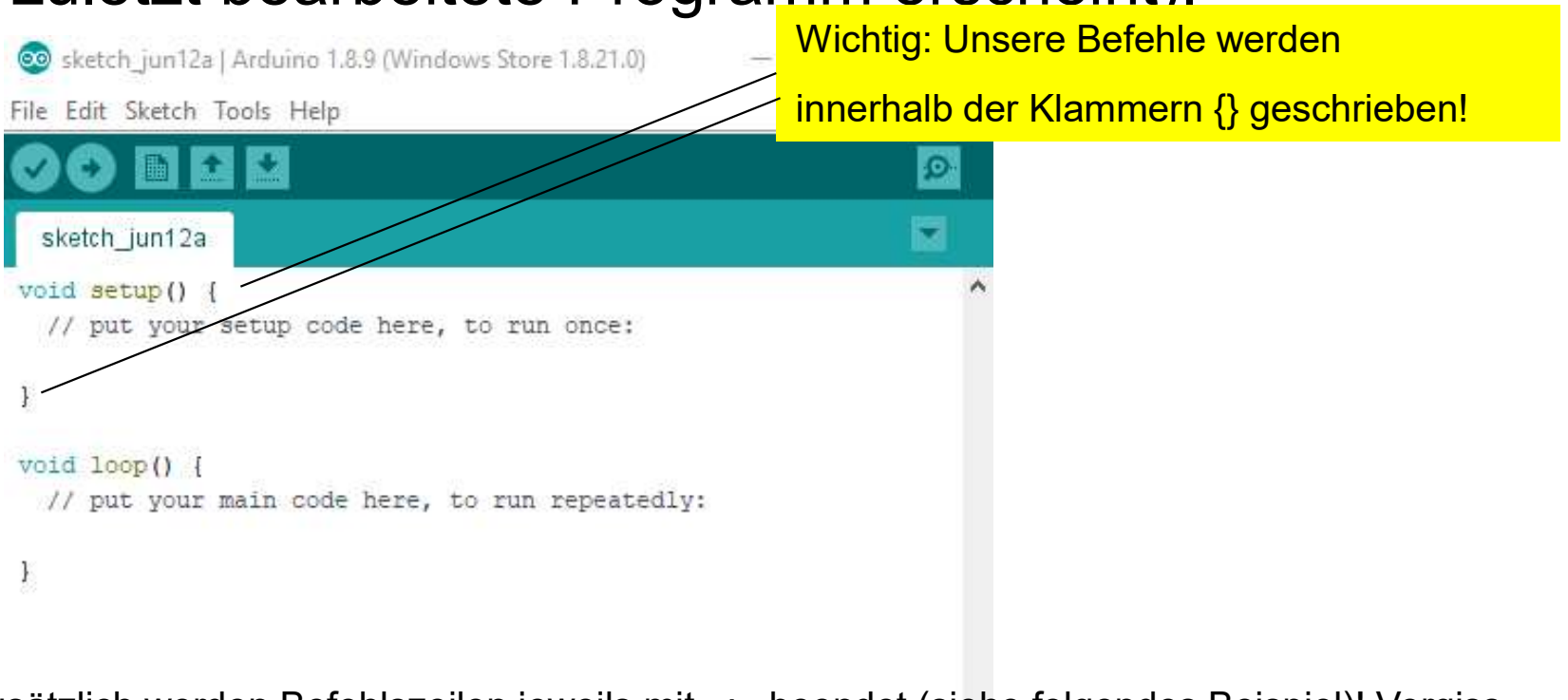
# Software Arduino IDE

- Integrierter Bootloader: Programmierung direkt über serielle Schnittstelle, respektive USB möglich
- Integrierte Entwicklungsumgebung (IDE), kostenlos
- Läuft auf Windows, Linux und macOS
- Arduino-IDE hat Code Editor und Compiler eingebunden
- Zusätzliche Bibliotheken vereinfachen die Programmierung in C und C++ stark

Der **Bootloader** ist ein kleines Programm, welches dem Atmel ATMega328P-pu sagt, dass er ein **Arduino** (genauer gesagt ein **Arduino**-Uno) werden soll. ...
Der **Bootloader** sorgt auch dafür, dass der Atmel ATMega328P-pu nach dem Einschalten prüft ob ein Programm auf ihm gespeichert ist.

# Software Arduino IDE

- Wenn man Arduino IDE aufruft, dann kriegt man zuerst ein Grundgerüst für die Programmierung (oder das zuletzt bearbeitete Programm erscheint).




Zusätzlich werden Befehlszeilen jeweils mit «;» beendet (siehe folgendes Beispiel)! Vergiss ich selbst immer wieder!

# Software Arduino IDE

- Für die Programmierung kennt man grundsätzlich zwei Strukturfunktionen:
  - **setup()** wird beim Start des Programmes einmalig aufgerufen
  - **loop()** wird kontinuierlich durchlaufen, ähnlich wie bei einer SPS (bei Siemens typischerweise OB1).
  - Zusätzlich **definiert** man am Anfang des Programmes die Ein-Ausgänge, Konstante Werte, Instruktionen für den Compiler etc und ruft hier Bibliotheken auf.

# Software Arduino IDE: Syntax

- ;  damit werden einzelne Instruktionen getrennt
- {}  unterscheiden der programmabschnitte
- //  für einzeiligen Kommentar
- /*……*/  für mehrzeiligen Kommentar
- ()  übergeben von Parametern bei Funktionen
- #define  zum Definieren von Anfangswerten
- #include  zum Einbinden von Bibliotheken

# Grundbefehle der Programmierung (Functions)

- **Digital I/O**
  - digitalRead()
    digitalWrite()
    pinMode()
- **Analog I/O**
  - analogRead()
    analogReference()
    analogWrite()
- **Zero, Due & MKR Family**
  - analogReadResolution()
    analogWriteResolution()
- **Time**
  - delay()
    delayMicroseconds()
    micros()
    millis()
- ….
- ….. see Internet
- **Communication**
  - Serial
    Stream

# Grundlagen der Programmierung

- Arduino Datentypen und Konstanten

**Constants**

Floating Point Constants
Integer Constants
HIGH | LOW
INPUT | OUTPUT | INPUT_PULLUP
LED_BUILTIN
true | false

**Conversion**

byte()
char()
float()
int()
long()
word()

**Data Types**

String()
array
bool
boolean
byte
char
double
float
int
long
short
size_t
string
unsigned char
unsigned int
unsigned long
void
word

**Variable Scope & Qualifiers**

const
scope
static
volatile

**Utilities**

PROGMEM
sizeof()

# Grundlagen der Programmierung

- Variablen
  - Eine Variable ist ein kleiner Speicher, in den eine Information einer bestimmten Form passt. Die Form wird durch den sogenannten Variablentyp bestimmt.

| Variablentyp | Bedeutung | Beschreibung |
|---|---|---|
| int | ganze Zahlen | -32'768 bis 32'767 |
| long | ganze Zahlen | -2'147'483'648 bis 2'147'483'647 |
| float | Fließkommazahl | gebrochene Zahlen |
| char | Character | Alphanumerische Zeichen (Buchstaben, Zahlen, Sonderzeichen) |

# Übung: Blinkende LED

Mit diesem Beispiel-Programm wird pin 13 (oder auch ein Anderer!) zum Ausgang deklariert und nachher jeweils mit HIGH und LOW beschaltet!

```
void setup() {
  pinMode(13, OUTPUT);    // sets the digital pin 13 as output
}

void loop() {
  digitalWrite(13, HIGH); // sets the digital pin 13 on
  delay(1000);            // waits for a second
  digitalWrite(13, LOW);  // sets the digital pin 13 off
  delay(1000);            // waits for a second
}
```

Grundsätzlich kann jeder Digital-Pin als Ein- oder Ausgang verwendet werden. Ein paar Pins sind jedoch reserviert für die Programmierung (RX/TX pin 0 und 1).

# Übung: Blinkende LED

- Da ich den Ausgang 13 mehrmals im Programm benütze, kann ich ihn auch als Variable ersetzen:
  - int ledPin=13;
- Nun kann ich im Programm "ledPin" verwenden.
- Die Variablenzuweisung erfolgt vor "void setup" und ist nachher überall gültig.

# Übung: Blinkende LED Was braucht ihr dazu?

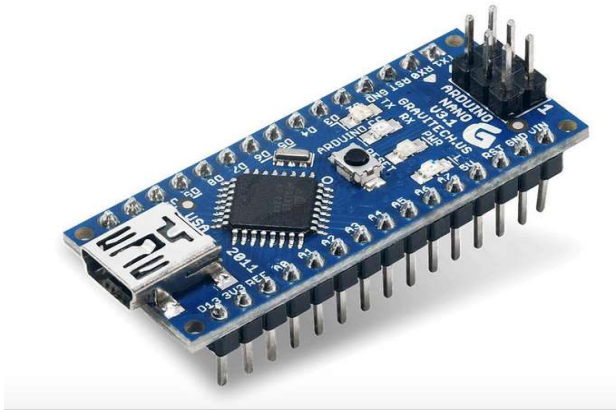
**Arduino Nano Platine**

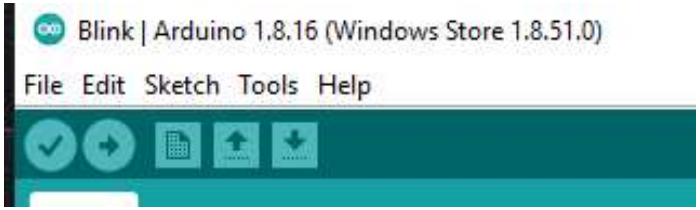
ATmega328 Mikrocontroller, Taktrate: 16 MHz

32 KB Flash Speicher

Betriebsspannung : 5 V, Empfohlene Eingangsspannung: 7-12 V

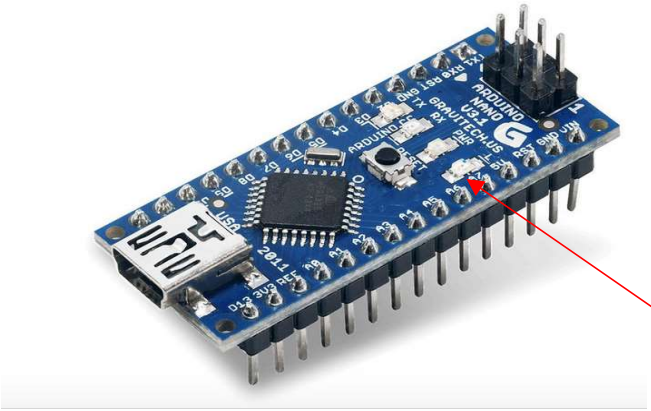
Anschluss über Mini-USB





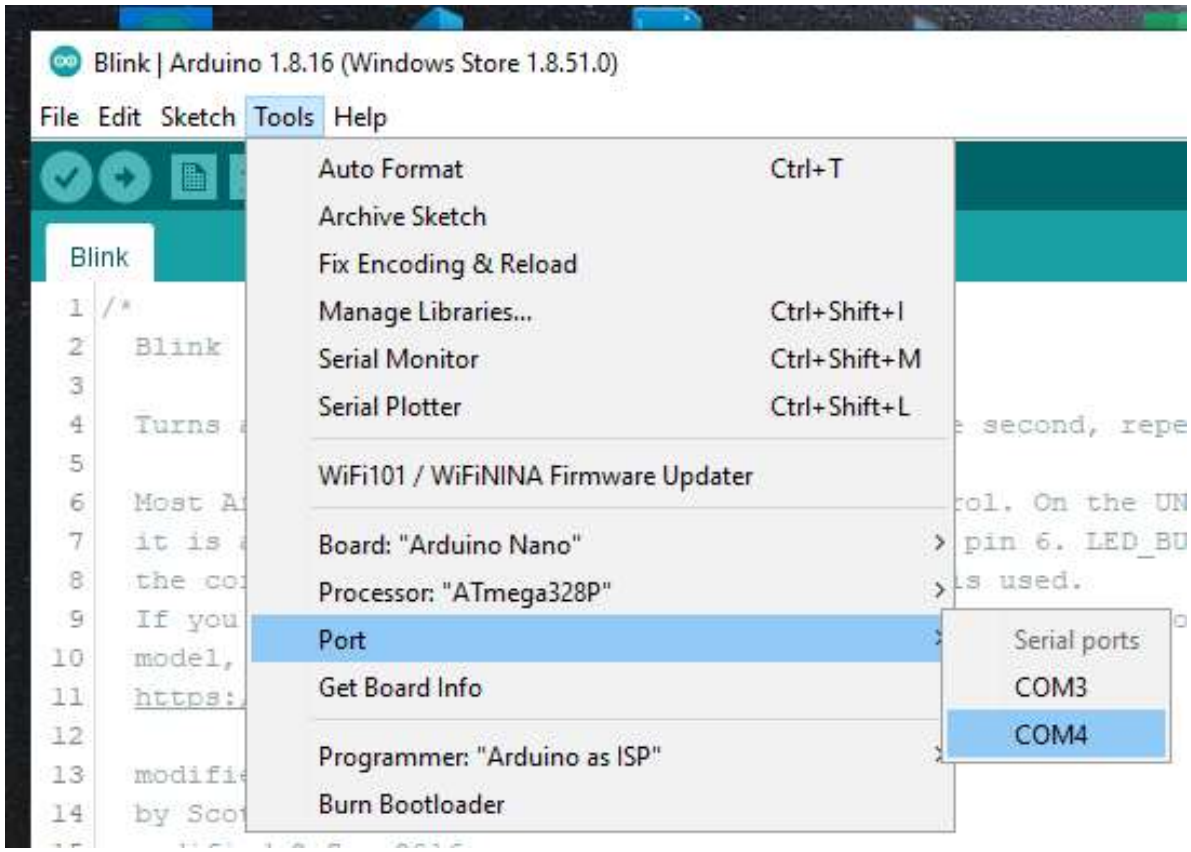

Auf dem PC ist IDE geladen

# Übung: Blinkende LED



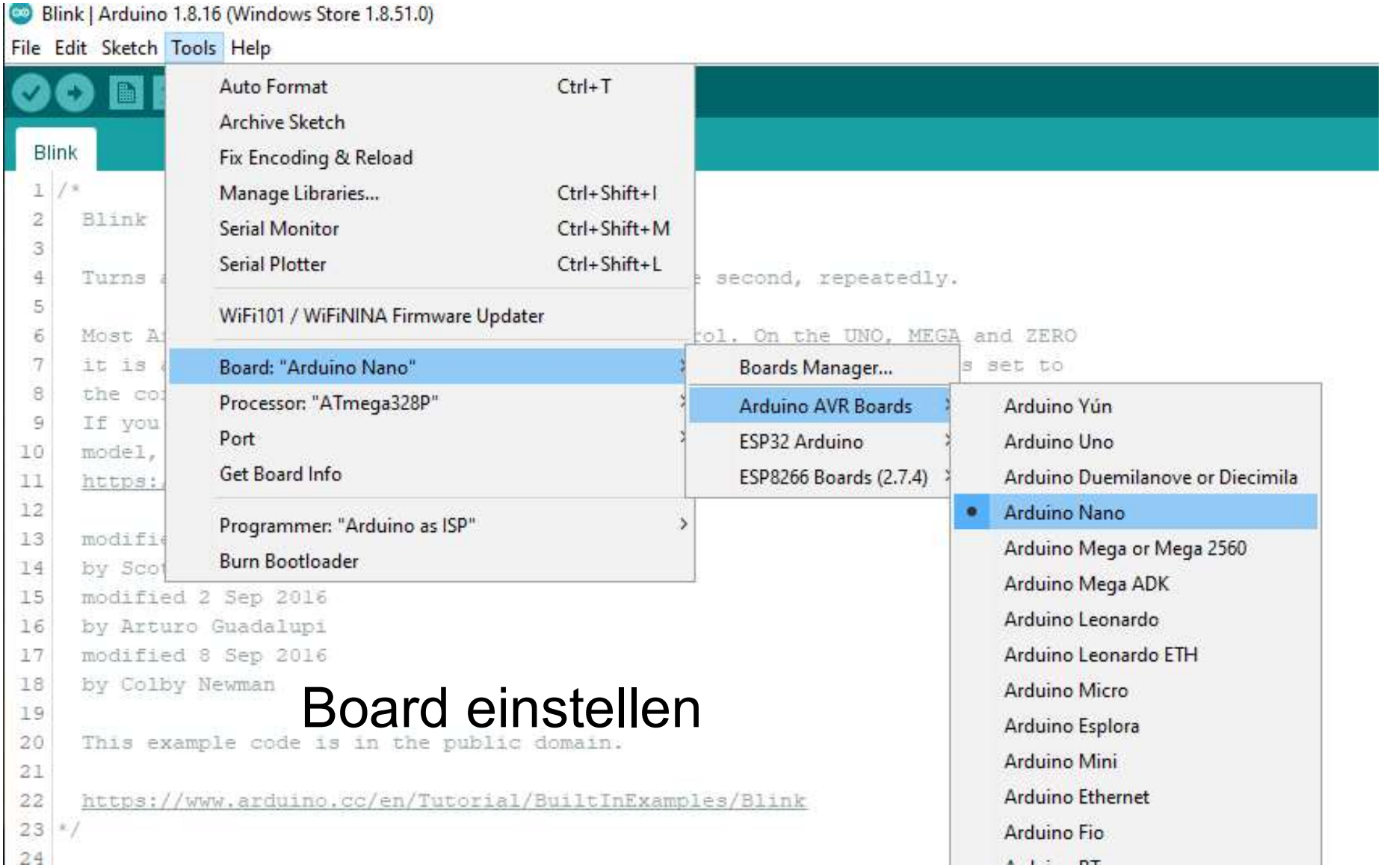
Dieses Board hat, wie die meisten Boards eine eingebaute LED. Diese wird mit LED_BUILTIN angesprochen

# IDE Programmiersoftware



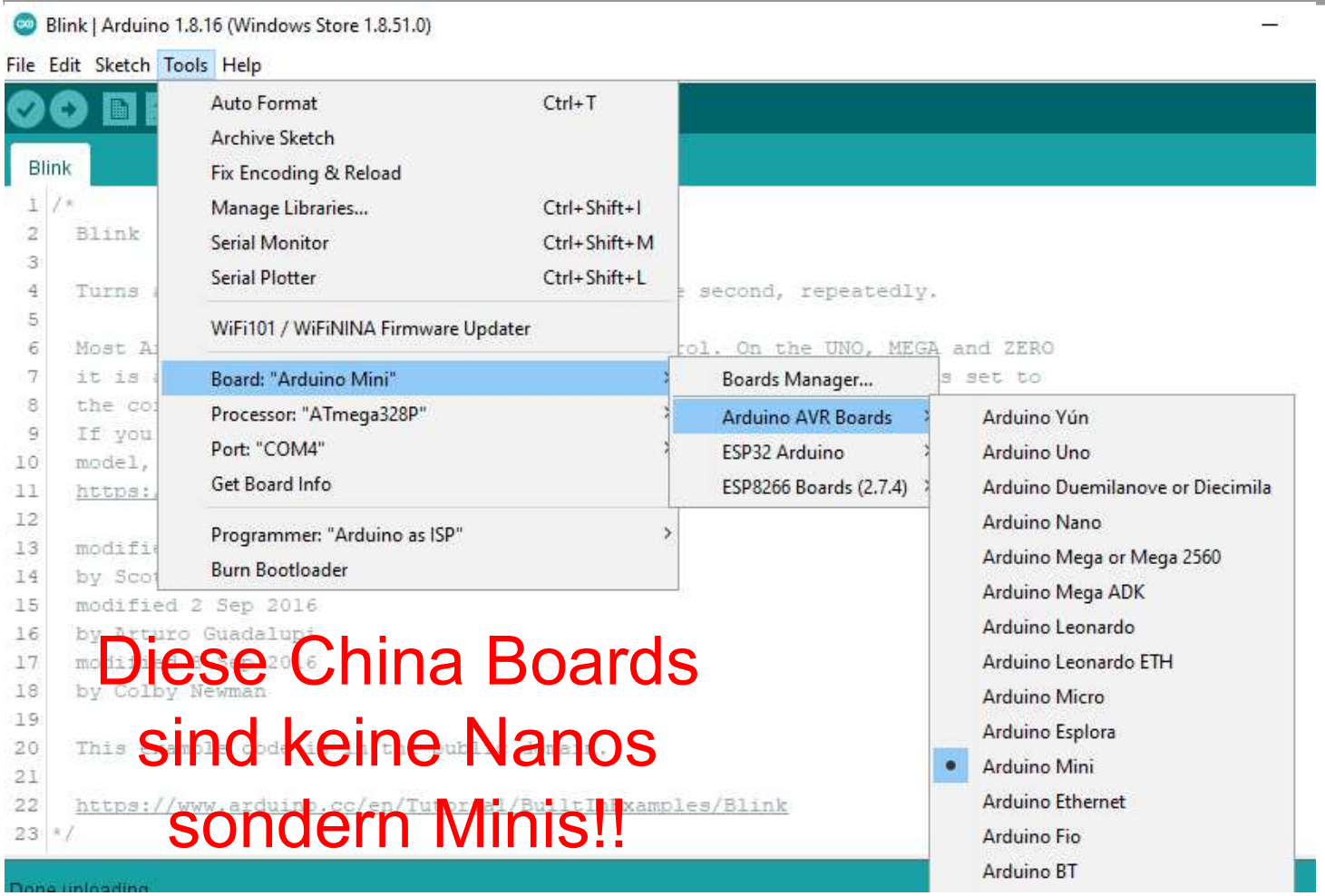

Port einstellen

# IDE Programmiersoftware




Board einstellen

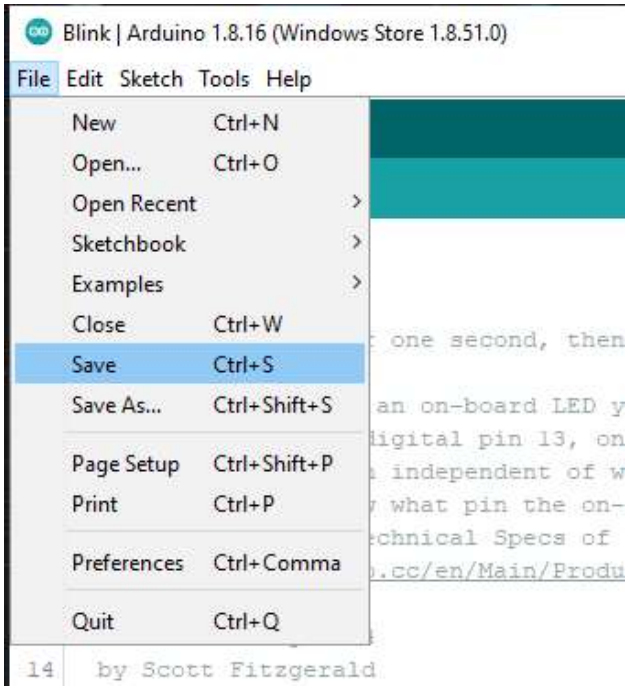
# IDE Programmiersoftware



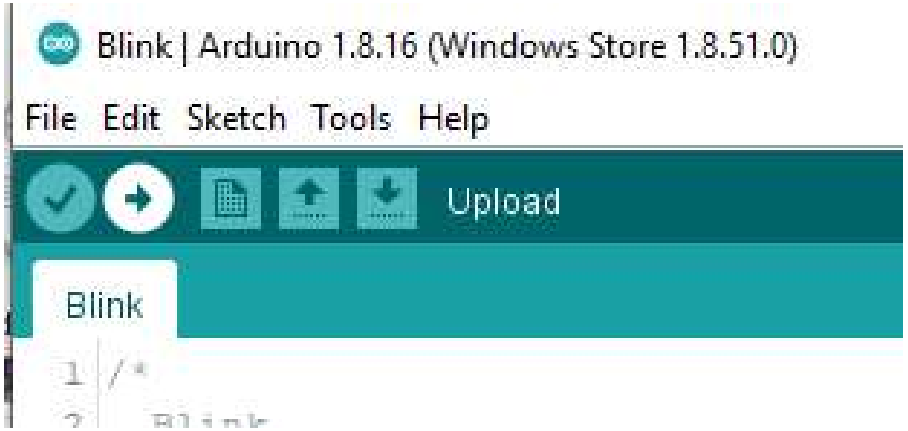

Da hilft dann meist leider nur probieren mit verschiedenen Boards!

# Übung: Blinkende LED

- Programm eintippen, abspeichern (ev.anschliessend Arduino IDE schliessen und Programm "Blinken" wieder laden.



# Übung: Blinkende LED (weitere Versuche)




Wenn man "Upload" anklickt, so wird das Programm übersetzt (compiliert) und wenn alles gut ist, auf den Chip geladen….man sieht dies an den flakernden LED's.

# Übung: Blinkende LED (weitere Versuche)

- Bitte die Ein-/Ausschaltzeiten ändern und Programm wieder übersetzen und auf Arduino laden.

# Übung: Blinkende LED

- Um einen Taster auszulesen, können wir den Befehl digitalRead(); verwenden. Er liefert einen Wert zurück, den man auslesen und vergleichen kann. Zum Vergleichen lässt sich die if-Abfrage nutzen:

```
int ledPin=13;
int buttonPin=2;
void setup() {
pinMode(buttonPin, INPUT);
pinMode(ledPin, OUTPUT);
}
void loop() {
if (digitalRead(buttonPin)==HIGH){
digitalWrite(ledPin, HIGH);
} else {
digitalWrite(ledPin, LOW);
}
}
```

# Übung: Blinkende LED (weitere Versuche)

- Programm so anpassen, wenn eine Taste gedrückt wird, die LED blinkt.

```
// constants won't change. They're used here to set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton stat

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the buttonState is HI
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

- Bussysteme

# Kapitel 2: Bussysteme mit Arduino

- Lernziel:
  - Anwenden von Bibliotheken, Bibliotheken aus dem Internet in Arduino DIE einbinden.
  - Verstehen von einfachen Bus Systemen auf Arduino, einfache Kommunikation
  - Übung: LCD Anzeige via I2C – Bus

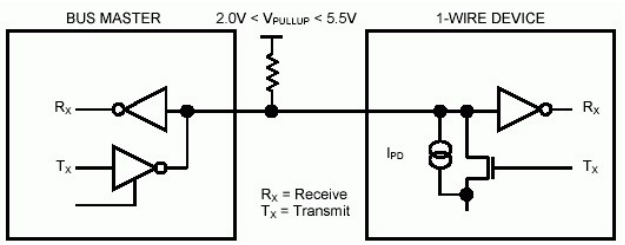
# Kapitel 2: Bussysteme mit Arduino

- Arduino vereinfacht die Kommunikation mittels verschiedenen Bussystemen:
  - $I^2C$
  - OneWire
  - SPI
  - Ethernet
  - CAN
  - RS485  ++……vermutlich noch viele mehr.

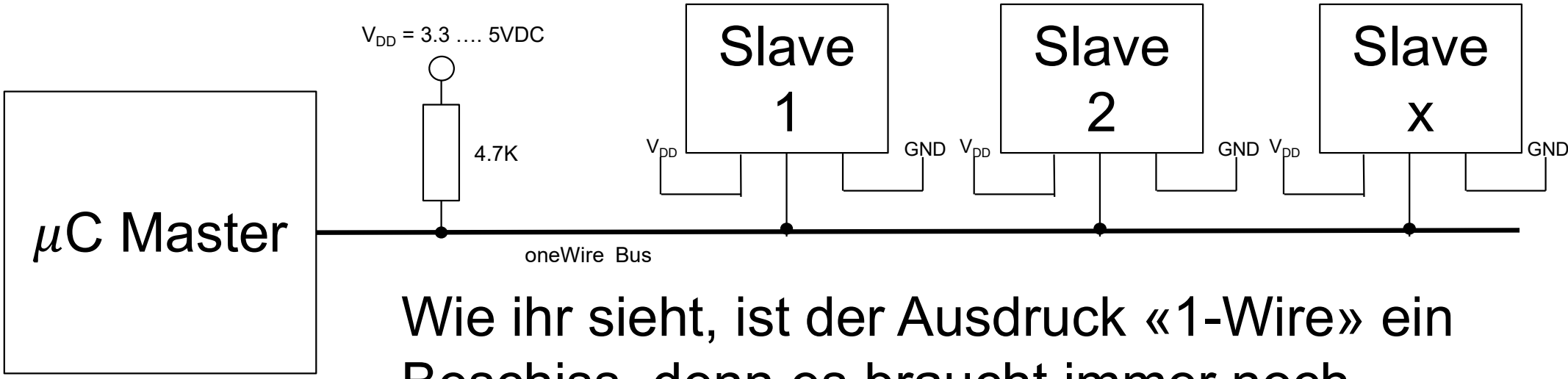
  Kommunikation via USB, Bluetooth oder RX/TX (RS232) sind zwar keine Bussysteme im eigentlichen Sinn; wir listen sie dennoch an dieser Stelle auf.
  Wir betrachten nur den I2C im Detail, Ethernet braucht eine Zusatzhardware (Shield). OneWire wird in Kapitel 3 im Zusammenhang mit Sensoren erklärt.

# Kapitel 2: Bussysteme mit Arduino

- Da das Ausprogrammieren eines Bussystems relativ komplex ist, behelfen wir uns mit dem Verwenden von **fertigen Bibliotheken**. Diese kopieren wir aus dem Internet oder verwenden die bereits vorhandenen im Arduino IDE.

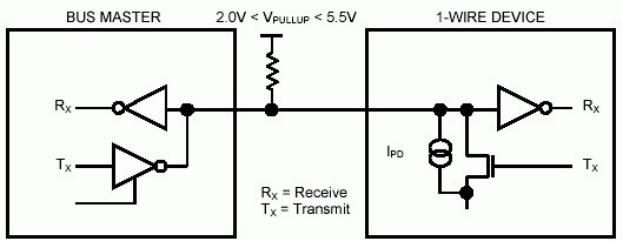



# OneWire Bus



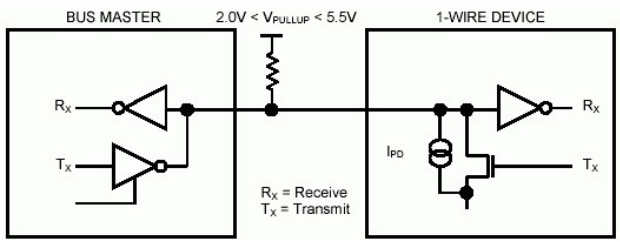

Wie ihr sieht, ist der Ausdruck «1-Wire» ein Beschiss, denn es braucht immer noch mindestens einen zusätzlichen Draht als Gnd. Die Speisung kann jedoch vom Bus erfolgen.

# Kapitel 2: OneWire Bus

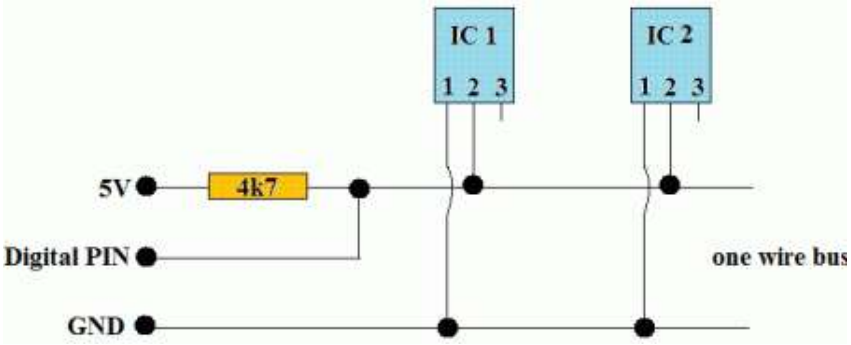



- Serial Signaling Protocol, wobei Daten und Speisung über den gleichen Draht erfolgen.
- Kommunikation erfolgt bidirektional, Halb-Duplex
- Konverter für RS232 zu OneWire erhältlich
- Single Master, mehrere Slaves
- Signal beinhaltet: Clock, Data und Speisung
- Jedes OneWire Device hat eine fixe, einmalige 64Bit Adresse
- Wenn der Bus inaktive ist, dann ist die Leitung «High» und dient in dieser Zeit als Speisung! Energie wird in internem Kondensator gespeichert.

# Kapitel 2: OneWire Bus




- Befehle können an mehre Slaves gleichzeitig gesendet werden.
- Regular Speed: 15.4….125 Kb/s mit bis zu 3m Distanz
- High ESD performance, IEC +/-15kV typ
- Spannung 1.75…5.25 V, bei -40°C…85°C
- Sehr einfach anzuwenden
- Bsp: DS18B20 mit ca Fr. 1.- von Maxim mit Dallas für einfache Temp Messung

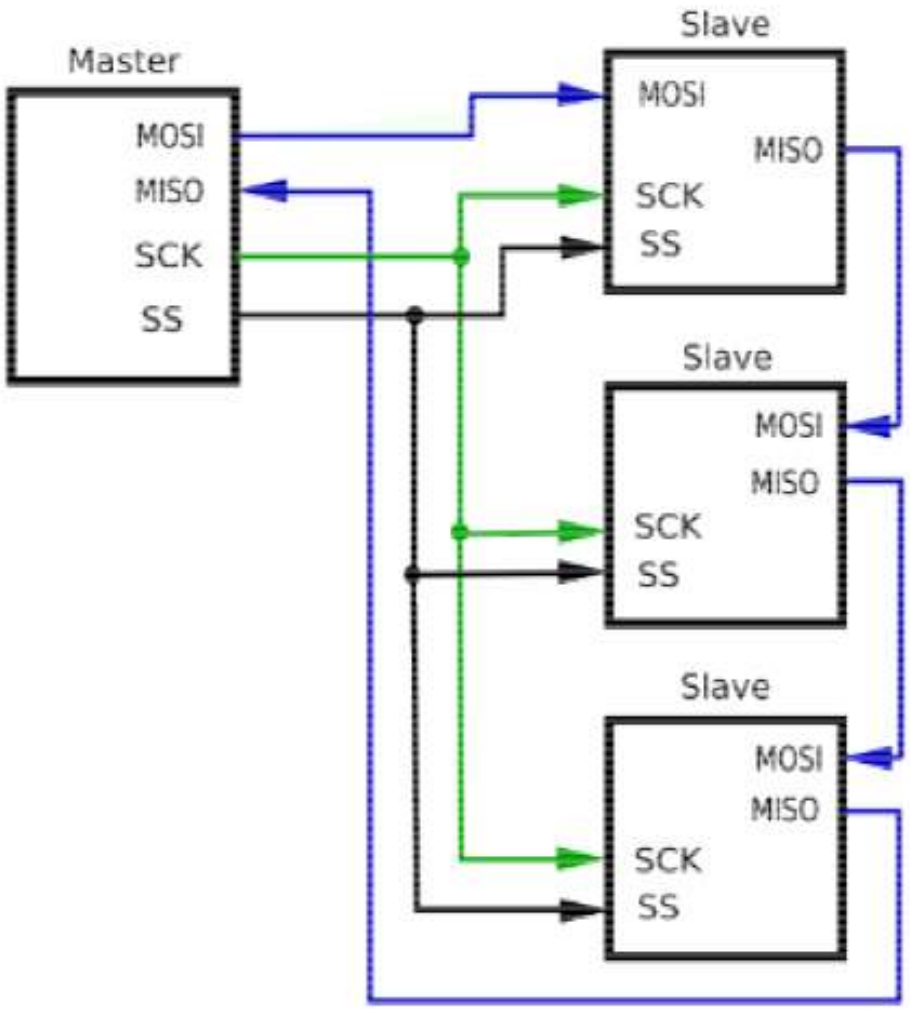


# SPI Serial Peripheral Interface

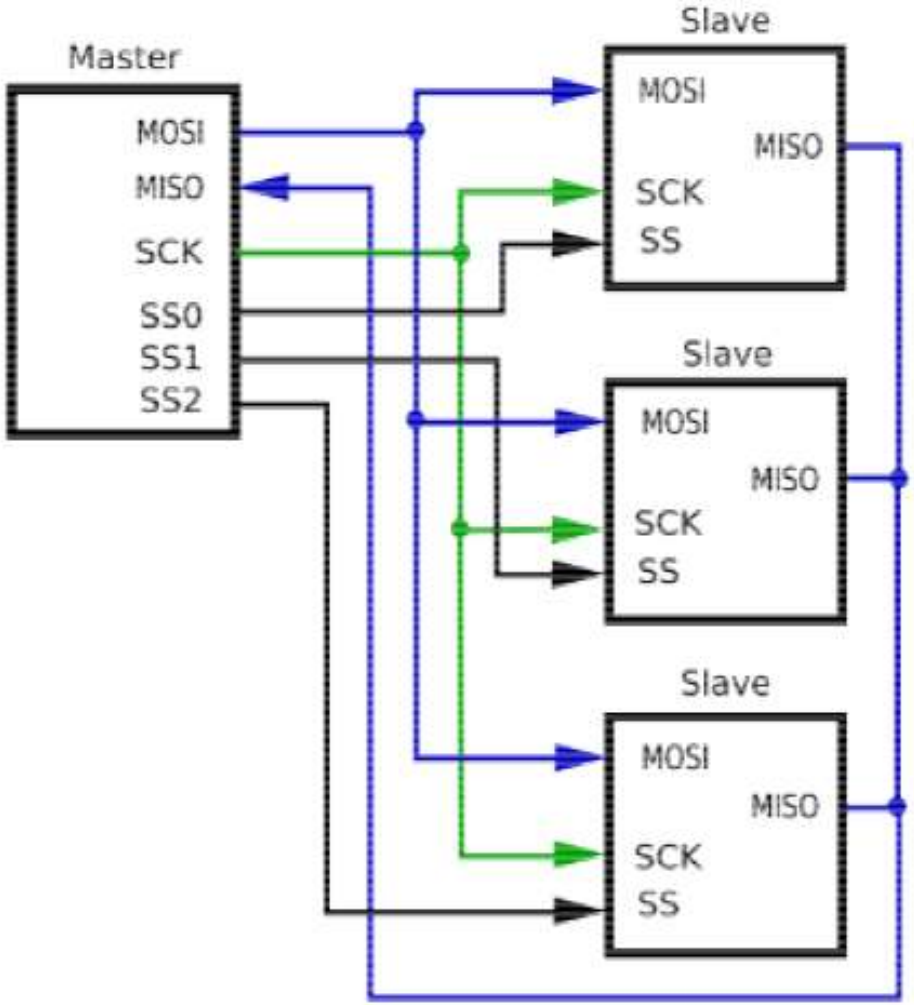
# Kapitel 2: SPI Serial Peripheral Interface

- Bus-System für synchrone serielle Datenübertragung
- Master/Slave Prinzip mit folgenden Signalen:
  - SCLK = Serial Clock, oft auch als SCK bezeichnet und wird vom Master ausgegeben.
  - MOSI = Master Output, Slave Input
  - MISO = Master Input, Slave Output
  - Individueller ChipSelect, meist SS oder mit CE bezeichnet.
- Eigentlich ein 4-Draht-Bus, wird aber auch als 2-Draht gehandelt.
- Vollduplexfähig, hohe Geschwindigkeiten im MHZ Bereich
- Übertragungsdistanz einige Meter, nicht mehr

# Kapitel 2: SPI Serial Peripheral Interface




Struktur: Kaskadierung / Daisy-Chain




Struktur: sternförmig

# Kapitel 2: SPI Serial Peripheral Interface

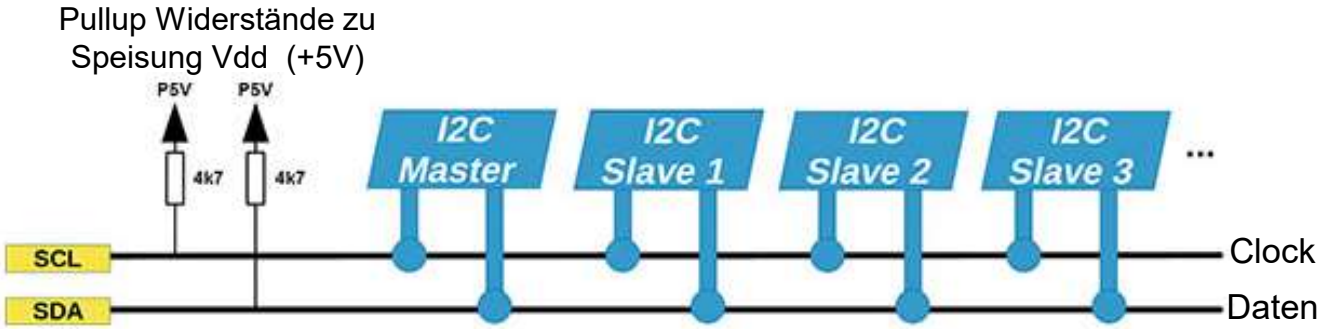
- Eine detaillierte Beschreibung findet ihr im Wiki: https://de.wikipedia.org/wiki/Serial_Peripheral_Interface
- Oder hier: http://gedankenlyrik.bplaced.net/www/studium/seminararbeit_2008.pdf
- https://www.mikrocontroller.net/articles/Serial_Peripheral_Interface
- Zum Programmieren stehen entsprechende Bibliotheken zur Verfügung.
- Wird meist verwendet für SD-Cards, Receiver, etc.

# I²C oder I-Quadrat-C-Bus

# Kapitel 2: $I^2C$ oder I-Quadrat-C-Bus

- Die komplette Beschreibung findet ihr unter:
  - https://www.elv.ch/Arduino-verstehen-und-anwenden-Teil-22-I%C2%B2C-der-Inter-IC-Bus-Grundlagen-und-Anwendungen/x.aspx/cid_726/detail_60435
- Ist ein serieller, bidirektionaler 2-Draht Bus mit:
  - Taktleitung SCL (Serial Clock)
  - Datenleitung SDA (Serial Data)
- 3.4 Mbit/s Übertragungsrate, also etwas langsamer als SPI
- Ursprünglich gedacht um zwischen IC's auf einer Platine zu kommunizieren
- Master/Slave Konzept. Auch Multimaster möglich.

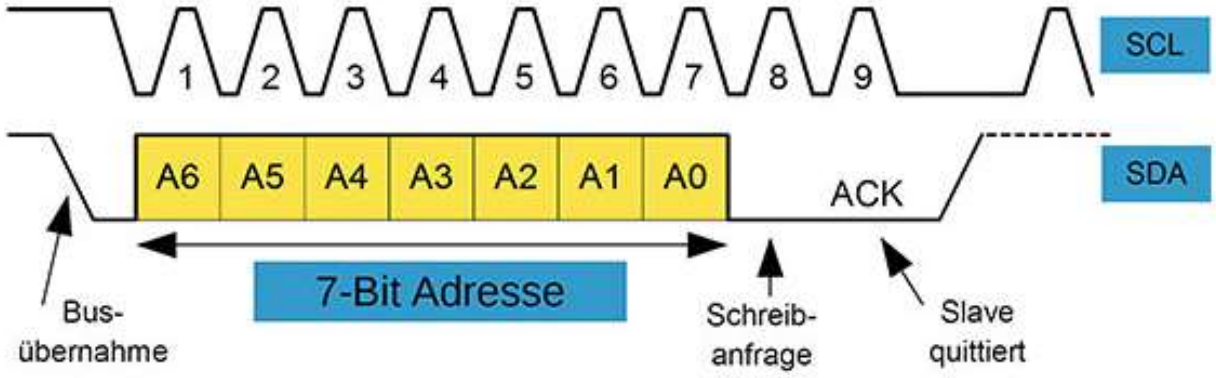
# I²C als Master-Slave-Bus




- Datenaustauch wird immer durch den Master gestartet. Anschliessen reagiert, der über seine Adresse angesprochene Slave auf die Anfrage.
- <0.3xVdd = Low Pegel, >0.7 = High Pegel
- Startsignal = fallende Flanke auf SCL, während SDA high ist.
- Stopsignal = steigende Flanke auf SDA, während SCL high bleibt.
- Dateneinheit sind 8 Datenbits, kann entweder ein Wert oder eine Adresse sein.
- ACK-Bit als Bestätigung vom Slave = Low Pegel auf Datenleitung

# I²C Protokoll und Adressierung

| | Adresse | | | | Geräte Subadresse | | | R/W | | Daten | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Start | 0 | 1 | 0 | 0 | x | x | x | 1 | ACK | x | x | x | x | x | x | x | x | Stopp |




- 7 Bit stehen für die Adresse zur Verfügung, das 8.Bit teilt dem Slave mit ob Daten gelesen oder geschrieben werden. Mit 7 Bit Adressen gibt es $2^7 = 128$ mögliche Teilnehmer, davon 16 sind aber reserviert, so bleiben noch 112 mögliche Teilnehmer.

# I²C Verwendung

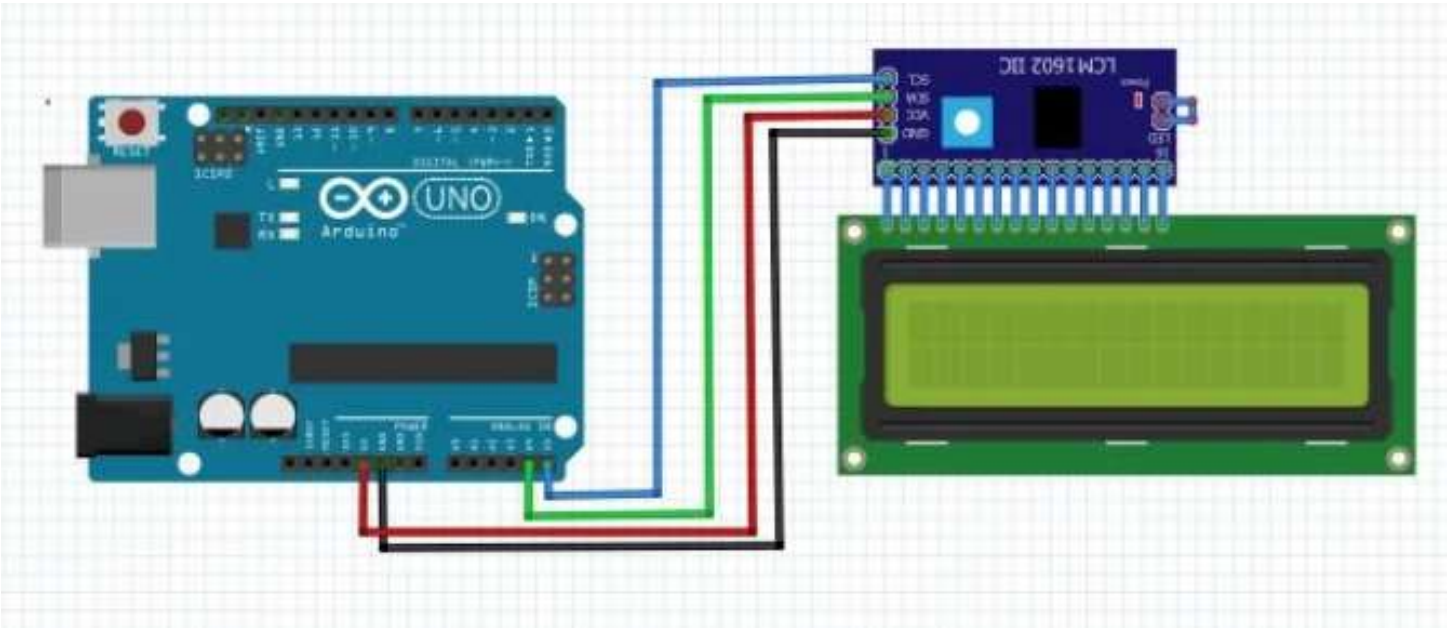
- Beispiele:
  - Temperatursensor LM75
  - Barometrischer Luftdruck BMP180
  - Echtzeituhr DS1307
  - A/D-Wandler MAX127
  - 2x16 Segment LCD Anzeige
  - EEPROM AT24C256
  - Etc……

# Zusammenfassung

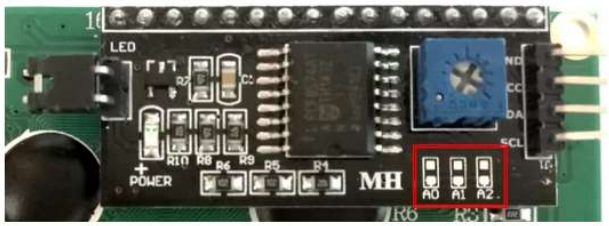
## Addressing

- SPI uses a hardwired chip select to identify the slave
  - Communication continues until hard-wired CS is released
- I2C starts each conversation with a 7-bit or 10-bit address to identify slave
  - Communication continues until stop condition is sent by master
- 1-wire can use dedicated hardware like SPI or addressing like I2C
  - Every device made has a worldwide unique address
  - If a single device on the bus, addressing generally not required
  - If used, addressing is a process of elimination

# Übung: mit I²C LCD-Display anschliessen




- **I2C Board  Arduino**
  - **GND ←→ GND**
  - **VCC ←→ 5V**
  - **SDA ←→ A4**
  - **SCL ←→ A5**




I²C-Modul mit rot markierten Lötstellen.
Standart-HEX-Adresse 0x3F,
veränderbar durch die Lötstellen A1-A3)

# I²C Programmierung

- Zuerst müssen wir die entsprechenden Bibliotheken suchen.
- Wir brauchen « Wire.h », das ist die I²C Bibliothek für die Kommunikation
- Zusätzlich brauchen wir noch die Bibliothek für die entsprechende LCD-Anzeige « LiquidCrystal_I2C.h »
- Wichtig:
  - Beachtet strikte die **Gross- und Kleinschreibung**, sonst wird die Bibliothek nicht gefunden.
  - Bibliotheken werden mit ***#include <Bibliothek.h>*** im ersten Teil **vor** Setup eingebunden.

# Fehlende Bibliotheken in Arduino

- Arduino hat nur einen Standardsatz von Bibliotheken, fehlende können mit «Manage Library» oder direkt vom Internet als ZIP-Datei runtergeladen werden.
- Dazu wird *Sketch -> Include Library-> Add .ZIP Library* aufgerufen. Die ZIP Datei wird dann automatisch sauber ins Verzeichnis eingebunden und steht nun zur Verfügung.

# Das fertige Programm

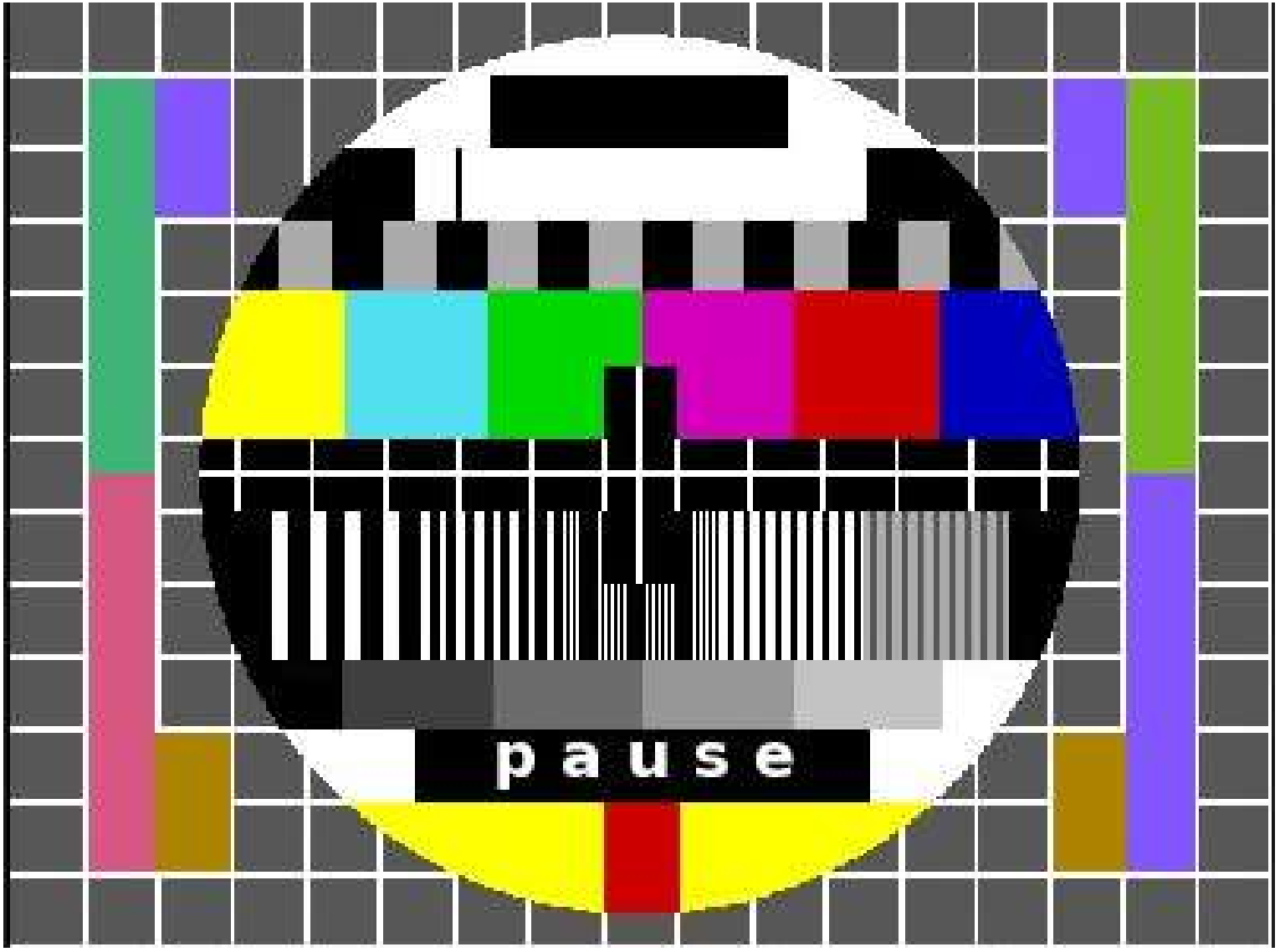
```
#include <Wire.h> // Wire Bibliothek einbinden
#include <LiquidCrystal_I2C.h>                  // Vorher hinzugefügte LiquidCrystal_I2C Bibliothek einbinden
LiquidCrystal_I2C lcd(0x27, 16, 2);                                    ev. Hex Adresse 0x3F verwenden
/*Hier wird festgelegt um was für einen Display es sich handelt.
 In diesem Fall eines mit 16 Zeichen in 2 Zeilen und der HEX-Adresse 0x27. Für ein vierzeiliges I2C-LCD
 verwendet man den Code "LiquidCrystal_I2C lcd(0x27, 20, 4)" */

void setup()
{
lcd.init();             //Im Setup wird der LCD gestartet
lcd.backlight();   //Hintergrundbeleuchtung einschalten (lcd.noBacklight(); schaltet die Beleuchtung aus).
}

void loop()
{
lcd.setCursor(0, 0);                //Hier wird die Position des ersten Zeichens festgelegt.
                                    // In diesem Fall bedeutet (0,0) das erste Zeichen in der ersten Zeile.
lcd.print(“Hallo USKA");
lcd.setCursor(0, 1);                // In diesem Fall bedeutet (0,1) das erste Zeichen in der zweiten Zeile.
lcd.print("Viel Erfolg!");
}
```

- Das müsst ihr aber nicht selbst tippen, denn im Verzeichnis der Bibliotheken findet man immer reichlich Beispiele!
- Erscheint der Text in der Anzeige? Gratulation!

# Ende der Lektion: Pause



# Sensorik

# Kapitel 3: Sensorik

- Lernziel:
  - Du lernst ein paar typische Sensoren und dessen Anschlussmöglichkeiten an Arduino kennen.
  - Du kannst Werte vernünftig auf die Anzeige bringen (skaliert).
  - Du machst dich mit Bussystemen vertraut.

# Kapitel 3: Sensorik

- Temperatur
- Feuchtigkeit
- Druck (Flüssigkeit, Gas)
- Licht (Intensität u. Wärmestrahlung Farbe)
- Entfernung, Füllstand
- Kraft
- Durchfluss
- Chemie (pH-Wert)
- Gaskonzentration ($CO_2$)
- Bewegung, Geschw.keit
- Beschleunigung
- Magnetismus
- Schall (hörbar, unhörbar)
- Helligkeit
- Dehnung
- … um nur einige zu nennen!

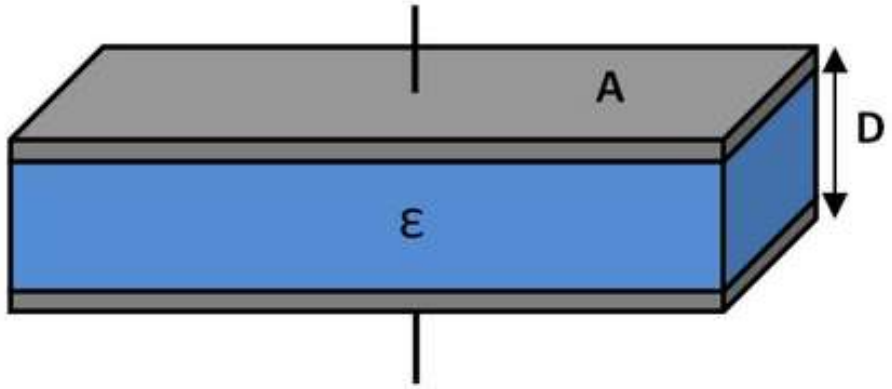
# Kapitel 3: Sensorik

- Wir werden versuchen folgende Sensoren an den Arduino anzuschliessen:
  - Feuchtemessung
  - Temperaturmessung TMP36
  - Druckmessung mit ESP8266 und BMP180

# Kapitel 3: Feuchtemessung

- Wir  nehmen den Feuchtesensor aus unserem Kit:
  - Feuchtemessung (genauer Luftfeuchtemessung)

$$C = eoer * A/D$$

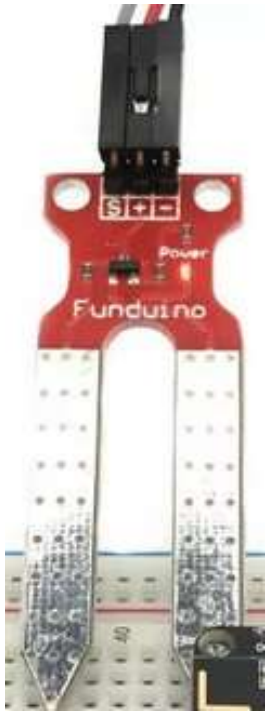



A, D und $e_o$ sind konstant,
$e_r$ ändert sich mit Feuchtigkeit

Also ändert sich die Kapazität entsprechend.

# Kapitel 3: Feuchtemessung

- Messung rein via Widerstand oder Leitwert (ungenau, für Feuchtigkeit in der Erde verwendbar)

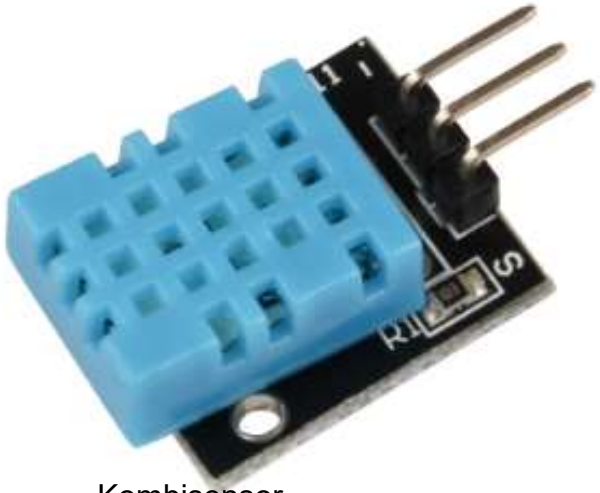



Oder mit Isotopensonde (Verwendung eines Neutronenstrahlers).

Neutronen werden an Wasserstoffatomen abgebremst, je stärker die Abbremsung ist, desto mehr Wasser.

# Kapitel 3: Feuchtemessung

- Verschiedene Feuchtigkeitsmodule für Arduino erhältlich:



Kombisensor
KY-015 mit 1-Wire Bus.
Misst Temperatur und Feuchte,
mit DHT11 oder DHT22 Chip, 5VDC

Auflösung: 0.1%RH, 0.1°C
Genauigkeit: +- 2% RH, +- 0.2°C


# Feuchtemessung mit DHT11

```
// Adafruit_DHT Library wird eingefügt
#include "DHT.h«          // Hier kann der jeweilige EingangsPin deklariert werden
#define DHTPIN 2          // Der Sensor wird initialisiert
#define DHTTYPE DHT11                    // DHT 11
DHT dht(DHTPIN, DHTTYPE);

void setup()
{
Serial.begin(9600);
Serial.println("Test - Temperatur und Luftfeuchtigkeits-Test:"); // Messung wird gestartet
dht.begin();
}
// Hauptprogrammschleife:Das Programm startet die Messung und liest die gemessenen Werte aus
// Zwischen den Messungen wird eine Pause von 2 Sekunden eingelegt, damit beim nächsten Durchlauf eine neue Messung erfasst werden kann.

void loop()
{
delay(2000); // Zwei Sekunden Pause zwischen den Messungen
// Luftfeuchtigkeit wird gemessen
float h = dht.readHumidity();
// Temperatur wird gemessen
float t = dht.readTemperature();
// Hier wird überprüft, ob die Messungen fehlerfrei druchlaufen sind
// Bei Detektion eines Fehlers, wird hier eine Fehlermeldung ausgegeben
if (isnan(h) || isnan(t))
                { Serial.println("Fehler beim Auslesen des Sensors");
                return; }
// Ausgabe in die serielle Konsole
Serial.println("------------------------------------------------------------");
Serial.print("Luftfeuchtigkeit: ");
Serial.print(h);
Serial.print(" %\t");
Serial.print("Temperatur: ");
Serial.print(t);
Serial.print(char(186)); //Ausgabe <°> Symbol
Serial.println("C ");
Serial.println("------------------------------------------------------------");
Serial.println(" ");
}
```

# Feuchtemessung programmieren

```
#include <Wire.h> // Wire Bibliothek einbinden
#include <LiquidCrystal_I2C.h> // Vorher hinzugefügte LiquidCrystal_I2C Bibliothek einbinden
LiquidCrystal_I2C lcd(0x27, 16, 2);

int messwert=0; // neu:Wir definieren den Messwert als Integer-Wert und geben ihm den Anfangswert 0.

void setup()
{
lcd.init(); //Im Setup wird der LCD gestartet
lcd.backlight(); //Hintergrundbeleuchtung einschalten (lcd.noBacklight(); schaltet die Beleuchtung aus).
}

void loop()
{
messwert=analogRead(A0);  // Der Messwert vom Analogen Eingang A0 soll ausgelesen, und unter der Variablen „messwert“ gespeichert werden.
lcd.setCursor(0, 0);      //Hier wird die Position des ersten Zeichens festgelegt. In diesem Fall bedeutet (0,0) das erste Zeichen in der ersten Zeile.
lcd.print("Messwert: ");
lcd.setCursor(0, 1);      // In diesem Fall bedeutet (0,1) das erste Zeichen in der zweiten Zeile.
lcd.print(messwert);
delay(500);
}
```

# Kapitel 3: Temperaturmessung

- Temperaturmessung mit TMP36GT9Z
- Präzisionsmessung, liefert Spannung linear zu Temperatur
- Genauigkeit +/- 1°C bei 25°C und +/-2°C über den ganzen Messbereich von -40…..+125°C
- Speisung mit 2.7 ….5.5 VDC mit weniger als 50$\mu$A, deshalb sehr kleine Selbsterwärmung (0.1°C).
- Ausgangsspannung: 750mV bei 25°C mit 10mV/°C
- Kostet ca Fr. 1.25

# Temperaturmessung mit TMP36

ANALOG DEVICES

## Low Voltage Temperature Sensors

## TMP35/TMP36/TMP37

### FEATURES

Low voltage operation (2.7 V to 5.5 V)
Calibrated directly in °C
10 mV/°C scale factor (20 mV/°C on TMP37)
±2°C accuracy over temperature (typ)
±0.5°C linearity (typ)
Stable with large capacitive loads
Specified −40°C to +125°C, operation to +150°C
Less than 50 µA quiescent current
Shutdown current 0.5 µA max
Low self-heating
Qualified for automotive applications

### APPLICATIONS

Environmental control systems
Thermal protection
Industrial process control
Fire alarms
Power system monitors
CPU thermal management

### GENERAL DESCRIPTION

The TMP35/TMP36/TMP37 are low voltage, precision centigrade temperature sensors. They provide a voltage output that is linearly proportional to the Celsius (centigrade) temperature. The TMP35/ TMP36/TMP37 do not require any external calibration to provide typical accuracies of +1°C at +25°C and ±2°C over the −40°C to +125°C temperature range.

The low output impedance of the TMP35/TMP36/TMP37 and its linear output and precise calibration simplify interfacing to temperature control circuitry and ADCs. All three devices are intended for single-supply operation from 2.7 V to 5.5 V maximum. The supply current runs well below 50 µA, providing very low self-heating—less than 0.1°C in still air. In addition, a shutdown function is provided to cut the supply current to less than 0.5 µA.

The TMP35 is functionally compatible with the LM35/LM45 and provides a 250 mV output at 25°C. The TMP35 reads temperatures from 10°C to 125°C. The TMP36 is specified from

### FUNCTIONAL BLOCK DIAGRAM

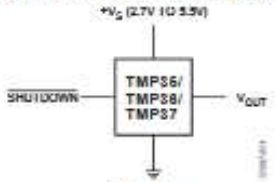



Figure 1.

### PIN CONFIGURATIONS

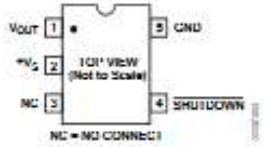



Figure 2. RJ-5 (SOT-23)

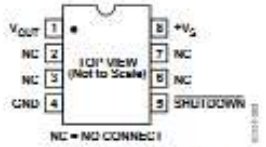



Figure 3. R-8 (SOIC_N)


Figure 4. T-3 (TO-92)

The TMP37 is intended for applications over the range of 5°C to 100°C and provides an output scale factor of 20 mV/°C. The TMP37 provides a 500 mV output at 25°C. Operation extends


$+V_s$ $V_{out}$ Gnd

# Programmierung TMP36

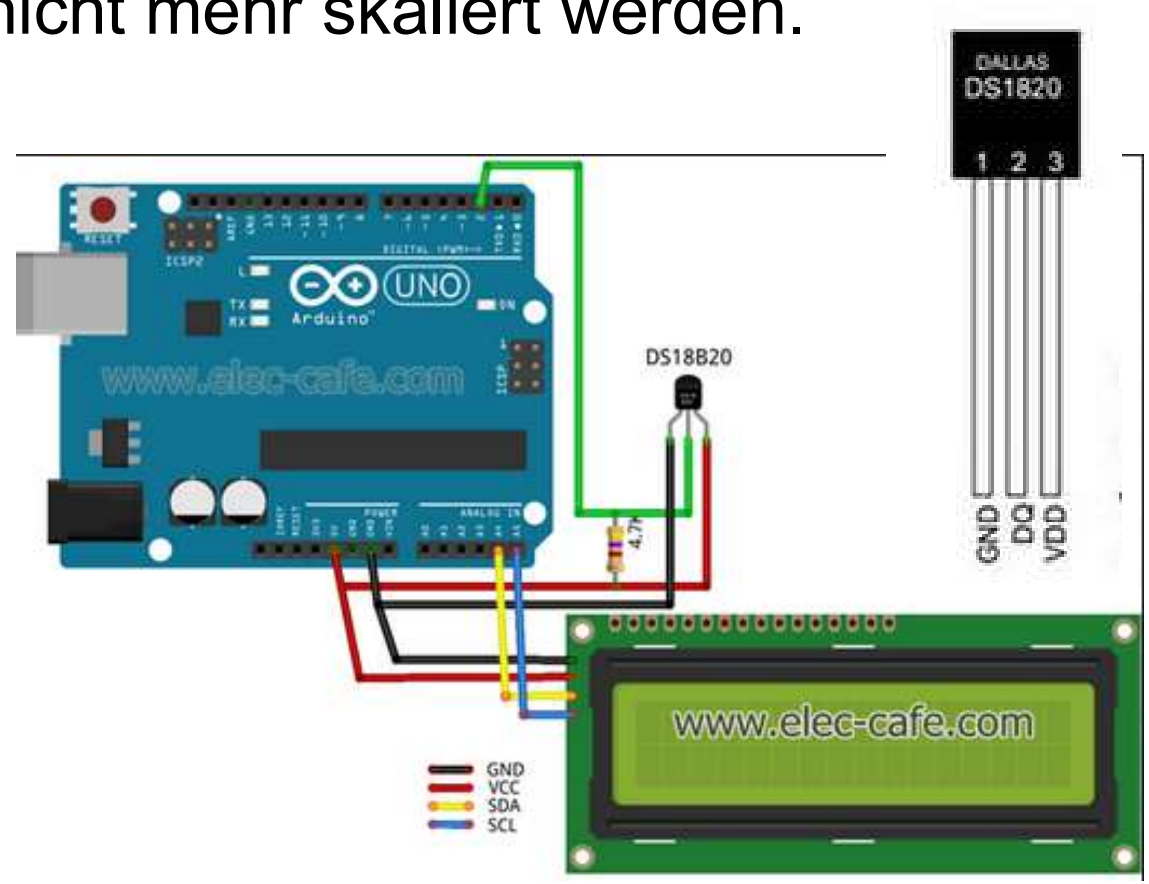
- Es braucht dazu keine spezielle Bibliothek. Wir lesen die Temperatur als  Spannung über A1 ein.
- Bitte den Wert entsprechend skalieren! Dazu wird die Funktion "map()" verwendet.

  map(value, fromLow, fromHigh, toLow, toHigh)

- Anzeige mittels LCD Display in °C

# Programmierung Dallas 18B20

- Es braucht dazu die Bibliothek für den OneWire (wire.h) und die Bibliothek für die Dallas Chips 18B20. Wir lesen die Temperatur direkt als floating point value ein. Der Wert muss also nicht mehr skaliert werden.
- Anzeige mittels LCD Display in °C
- Wichtig: Pullup Widerstand 4.7K auf Data-Leitung



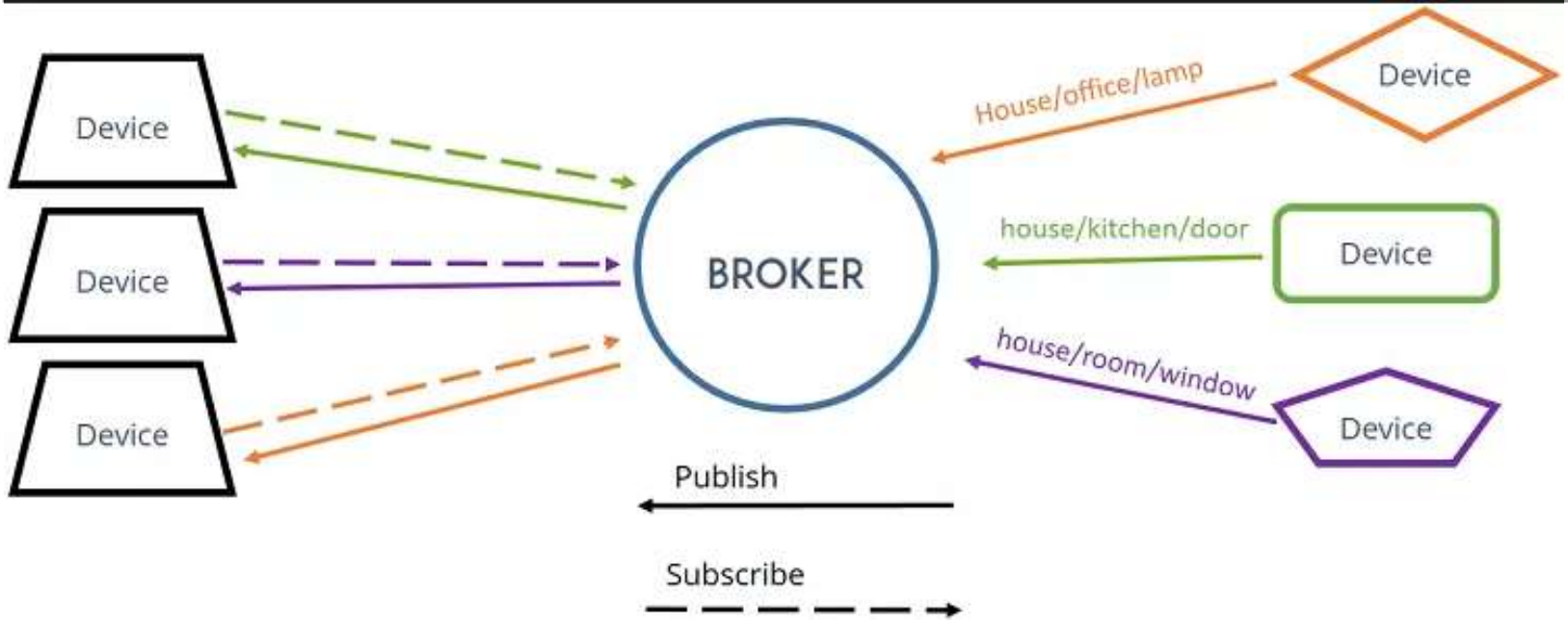
# Kapitel 4: Kommunikation via Mosquitto Broker

- MQTT

# Kapitel 4: Kommunikation via Mosquitto Broker

- Lernziel:
  - Du erkennst die Möglichkeiten von IoT (Internet of Things).
  - Kannst eigene Daten auf dein Handy übertragen.

# Kapitel 4: Kommunikation via Mosquitto Broker



# Kapitel 4: MQTT (kopiert aus Wikipedia)

Message Queuing Telemetry Transport (MQTT) ist ein offenes Nachrichtenprotokoll für Machine-to-Machine-Kommunikation (M2M), das die Übertragung von Telemetriedaten in Form von Nachrichten zwischen Geräten ermöglicht, trotz hoher Verzögerungen oder beschränkter Netzwerke.

Entsprechende Geräte reichen von Sensoren und Aktoren, Mobiltelefonen, Eingebetteten Systemen in Fahrzeugen oder Laptops bis zu voll entwickelten Rechnern.

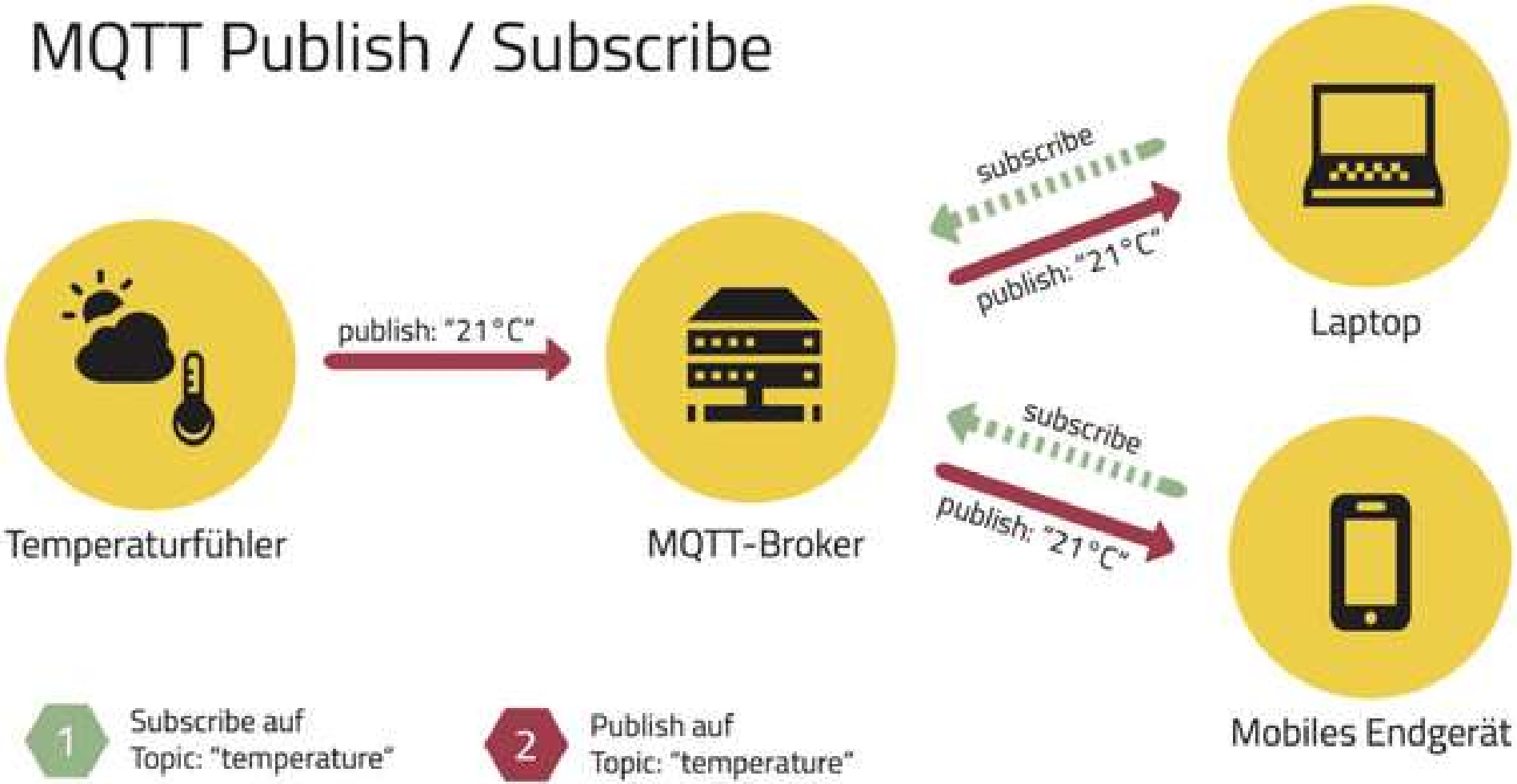
Das MQTT-Protokoll ist auch unter älteren Namen wie „WebSphere MQTT“ (WMQTT), „SCADA-Protokoll“ oder „MQ Integrator SCADA Device Protocol“ (MQIsdp) bekannt.

Die Internet Assigned Numbers Authority (IANA) reserviert für MQTT die Ports 1883 und 8883. MQTT-Nachrichten können mit dem TLS-Protokoll verschlüsselt werden.

# Kapitel 4: MQTT  (kopiert aus Wikipedia)

Interessant ist, dass ein MQTT-Server („Broker“) die gesamte Datenlage seiner Kommunikationspartner hält, und so als Zustands-Datenbank benutzt werden kann.
So ist es möglich, kleine unperformante MQTT-Geräte mit einem MQTT-Broker zu verbinden, wobei die Geräte Daten einsammeln und/oder Befehle entgegennehmen, während ein komplexes Lagebild nur auf dem MQTT-Broker entsteht und hier oder durch einen leistungsfähigen Kommunikationspartner ausgewertet werden kann.
Stelleingriffe können so von einer oder mehreren leistungsfähigen Instanzen an den MQTT-Broker übermittelt und auf die einzelnen Geräte verbreitet werden.
Dadurch eignet sich MQTT sehr gut für Automatisierungslösungen und findet im Bereich IoT durch die einfache Verwendung große Verbreitung.

# Kapitel 4: MQTT Mosquitto Broker Funktion



# Kapitel 5: Anwendungen im Amateurfunk

- Für was ist der Arduino weniger geeignet?
  - Wenn grosse Datenmengen verarbeitet werden müssen.
  - Ansteuern von Graphikdisplays
  - FT8-Signale dekodieren

# Welche Anwendungen kennen wir?

- AD9850 ansteuern (VFO etc.)
- Programmierbarer Oszi Si570  0.5….160Mhz von Axe Schultze, DK4AQ
- ARDUINO©- Mikrocontroller im Amateurfunk von Mathias Dahlke, DJ9MD
- ADF4351-Baustein von SV1AFN
  http://www.kh-gps.de/adf4351.htm
- Stationsuhr von Ingo, DL6IS https://p34.meindarc.de/?p=1182
- Morsekeyer von Ingo, DL6IS https://p34.meindarc.de/?p=1198
- SWR Analyser von Daniel, DL2AB https://wiki.funkfreun.de/projekte/swr-analyser
- CAT- Interface http://www.kh-gps.de/Arduino_8f.pd
- Rotor- Interface, K3NG https://blog.radioartisan.com/yaesu-rotator-computer-serial-interface
- µBitX Transceiver mit Nextion©
- APRS-Terminal von Jürgen, DL8MA  http://www.dl8ma.de/Arduino/APRS-Terminal/

- …..zig viele Beispiele im Internet

# Quellenangaben

- Muss noch komplettiert werden……
- Die meisten Bilder stammen aus dem Internet, eine detaillierte Quellenangabe ist leider meist nicht möglich.
- Da wir **keine Kopierrechte** eingeholt haben, sollten die Bilder möglichst nicht weiter verbreitet werden.