Spezielle Aspekte des HCI: Interaction Design with Arduino

Using the Arduino OpenHardware Platform to sketch and develop physical interactions and tangible user interfaces

Timm Wunderlich, Oliver Stickel, Julian Dax
Wirtschaftsinformatik und Neue Medien
Universität Siegen
timm.wunderlich@uni-siegen.de
oliver.stickel@uni-siegen.de
julian.dax@uni-siegen.de

UNIVERSITÄT SIEGEN

Spielregeln

- Vorlesung plus Praktikum

- Vorlesung
  - ± Theoretischer Hintergrund und Handlungsleitende Exkurse

- Praktikum
  - ± Selbstständiges Arbeiten in Teams
  - ± Fachliche Begleitung der Projekte
  - ± Zwischenprüfung
    - Vorgegebenes Projekt bzw. Auswahl
    - Um Weihnachten
  - ± Abschlussarbeit
    - Eigenständige Definition von Klein-Projekten
    - Im Anschluss der Vorlesung (Semesterferien)

Leistungserwartung

- Bis zu 50% : Zwischenprüfung
- 25% Abschlussprojekt Umsetzung
- 25% Abschlussprojekt Dokumentation

5 ETCS

Ressourcen

- Buch:
  Joshua Noble: Programming Interactivity
  Make Magazine: Making Things Talk
  Alasdair Allan: iOS Sensor Apps with Arduino

- www.arduino.cc

Mikrocontroller

- µC, uC, MCU
- kleiner Computer in einer einzelnen Integrierten Schaltung (IC) die aus einem Prozessor (CPU), Arbeits-(SRAM) und Programmspeicher(FLASH) und weiteren Peripheriefunktionen besteht.wie z.B. :
  - Digital/Analog Wandler (D/A Wandler, DAU, DAC)
  - Anschlüsse für Digitale/Analoge Eingabe/Ausgabe (I/O Pins)
  - Anschlüsse für Bus Systeme (z.B. USB, Seriell, CAN, LIN, I²C, SPI...)

- Typisch ist eine im Vergleich zum PC eingeschränkte Leistung und Ausstattung, dafür allerdings auch geringe Herstellungs- und Betriebskosten
  - Takfrequenz < 100MHz
  - nur wenige kB RAM ROM
  - keine Multitasking-Fähigkeit
  - bekommt man schon für < 10€

- Einsatz hauptsächlich in eingebetteten Systemen (embedded systems) wie z.B. Haushaltsgeräten, Chipkarten, Kraftfahrzeugen...

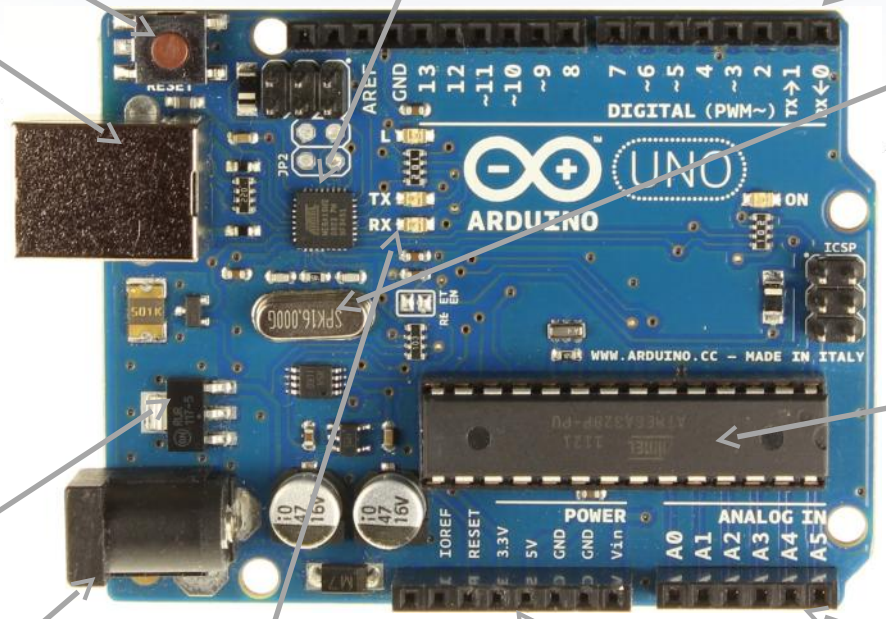
Funktionales Modell (IPO Modell)

- Measure **I**nput
  - ± Analog (Sensors, Sliders)
  - ± Digital (Buttons, Switches, Serial Devices, …)
- **P**rocess (Calculate and Convert)
  - ± Programming Language
- Generate **O**utput
  - ± Digital (High/Low, PWM, Serial Signals,…)

Arduino Plattform

- Aus Hard- und Software bestehende
  Open Source Physical-Computing-Platform
- soll auch technisch weniger Versierten den Zugang zur Programmierung und zu Mikrocontrollern erleichtern
- Die Arduino Boards basieren auf Atmel AVR Mikrocontrollern
  - ATmega168 (16MHz, 16kB Flash, 1kB RAM)
  - ATmega328 (16MHz, 32kB Flash, 2kB RAM)
  - ATmega1280 (16MHz, 128kB Flash, 8kB RAM)
  - ATmega2560 (16MHz, 256kB Flash, 8kB RAM)
  - ATMEL SAM3U (96MHz, 256kB Flash, 50kB RAM)
- Ausser dem eigentlichen Mikrocontroller befindet sich alle für den Betrieb nötige Peripherie auf der Platine
  - Quarz für den Takt
  - Spannungsversorgung
  - Pin-Leisten für die I/O Pins
  - USB Schnittstelle für die Programmierung
  - Reset-Schalter

Arduino Plattform



Reset-Pushbutton

USB2Serial

Digital (Input/Output)

USB-Port (TTL)

Clock (quartz)

MCU

Power-Regulator

Power-Supply

Serial In/Out Control LEDs

5V-Potential / Ground

Analog Input

Arduino Boards

- Arduino Duemilanove
  - ATmega168/328P
  - 14/6 Pins (digital/analog)

- Arduino Mega(2560)
  - ATmega1280/2560
  - 54/16 Pins (digital/analog)

- Arduino Nano
  - ATmega168 or ATmega328
  - 14/8 Pins (digital/analog)

- Arduino Mini Pro
  - ATmega168
  - 14/6 Pins (digital/analog)

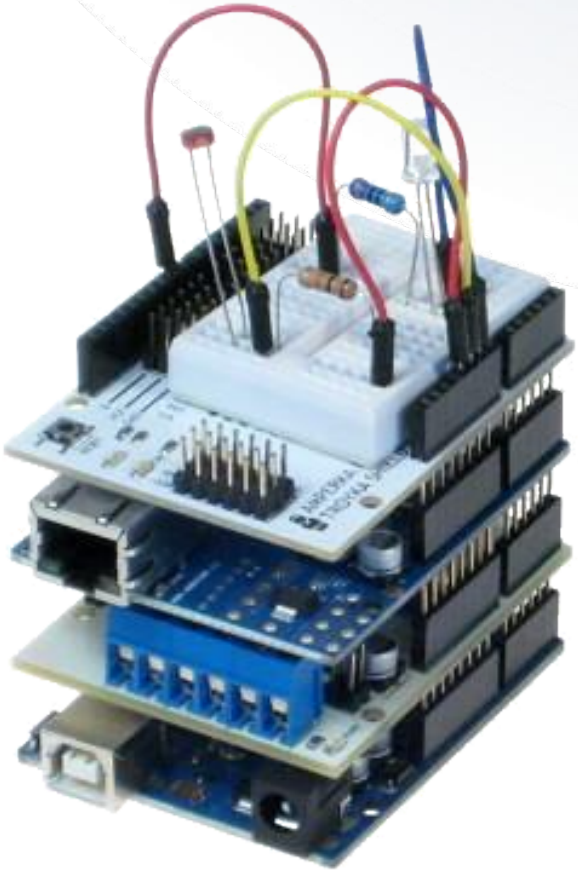
Alternativen

- Microsoft .NET Micro
  - FEZ domino board (Arduino-like)
  - Basiert auf .NET Micro framework
  - Visual Studio .NET benötigt

- PICmicro
  - RISC
  - Programmierung in C oder Assembler
- Phidgets
  - Physical widgets
  - Just I/O boards
  - Controlled externally by PC/Mac or external microcontroller
  - APIs available for several languages

- Raspberry Pi
  - Modell B+ hat 40 GPIO Pins
  - Unterstützt diverse Bus Syteme (UART, SPI, I^2C) um externe Komponenten an zu schliessen
  - Bibliotheken für z.B. Python, JAVA, C, C++

Hardware-Architektur



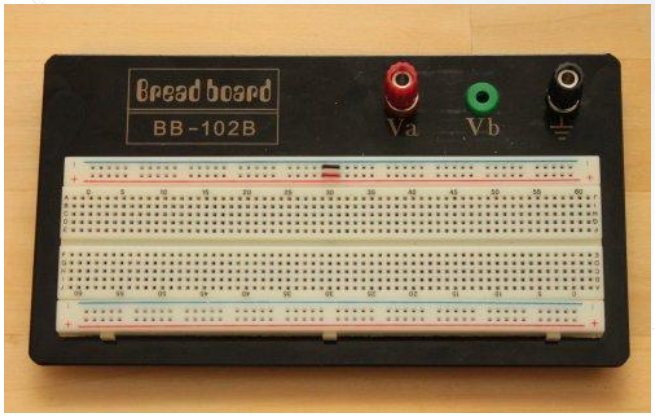
- Die meisten Arduino Boards haben einen einheitlichen Form-Faktor
  - ± Grösse des Boards (PCB)
  - ± Position und Abstand (Rastermaß) der Pin-Leisten
  - ± Spannung und Stromstärke der I/O Signale
- Jede Menge fertige Erweiterungen (Shields) für unterschiedlichste Anwendungsfälle vorhanden
- Generische Shields mit Steckplätzen für Prototypen ohne feste Verlötung
- Shields sind stapelbar

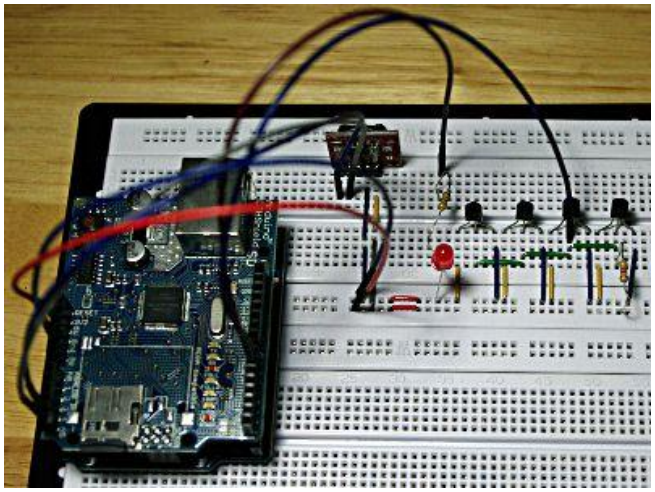
Arduino programmierung

- Programmiersprache
  - ± C++ Dialekt
  - ± Eingeschränkte Anzahl an Datentypen und Funktionen, bestimmt durch den verwendeten Controller
  - ± Jede Menge Bibliotheken für die Ansteuerung von Hardware und Shields
- Entwicklungsumgebungen (IDEs)
  - ± Arduino IDE (basiert auf Processing)
  - ± VisualMicro (Add-On für MS-Visual Studio, basiert auf der Arduino IDE)
  - ± Eclipse-PlugIns (based on C/C++ Toolchain of the Eclipse Project, AVR-Tools, etc.)
  - ± Schreiben (programmieren) von Code und Bootloader in den Mikrocontroller

UNIVERSITÄT SIEGEN

Prototyping Werkzeuge

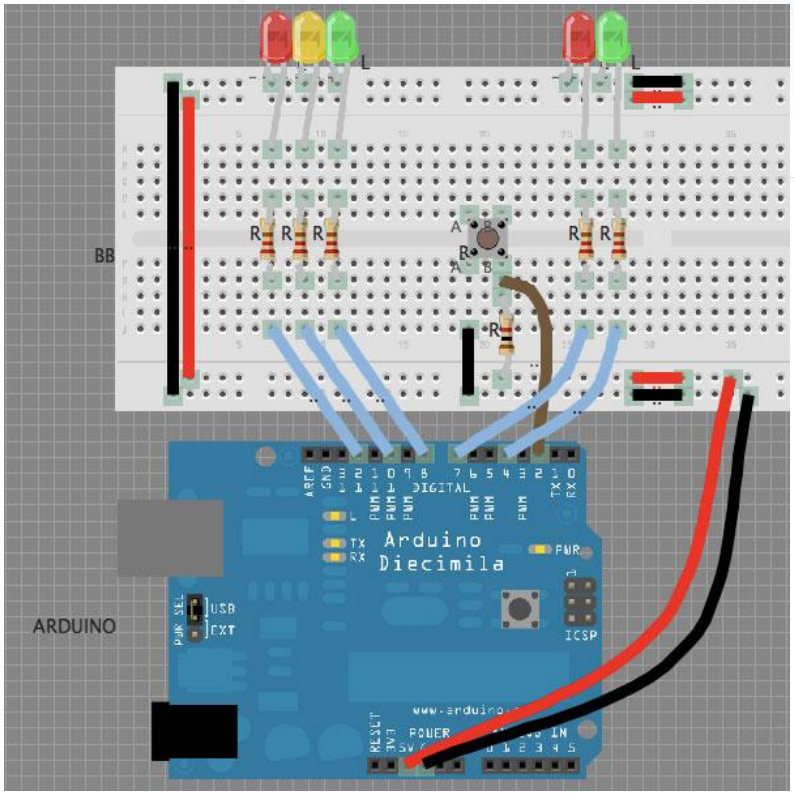


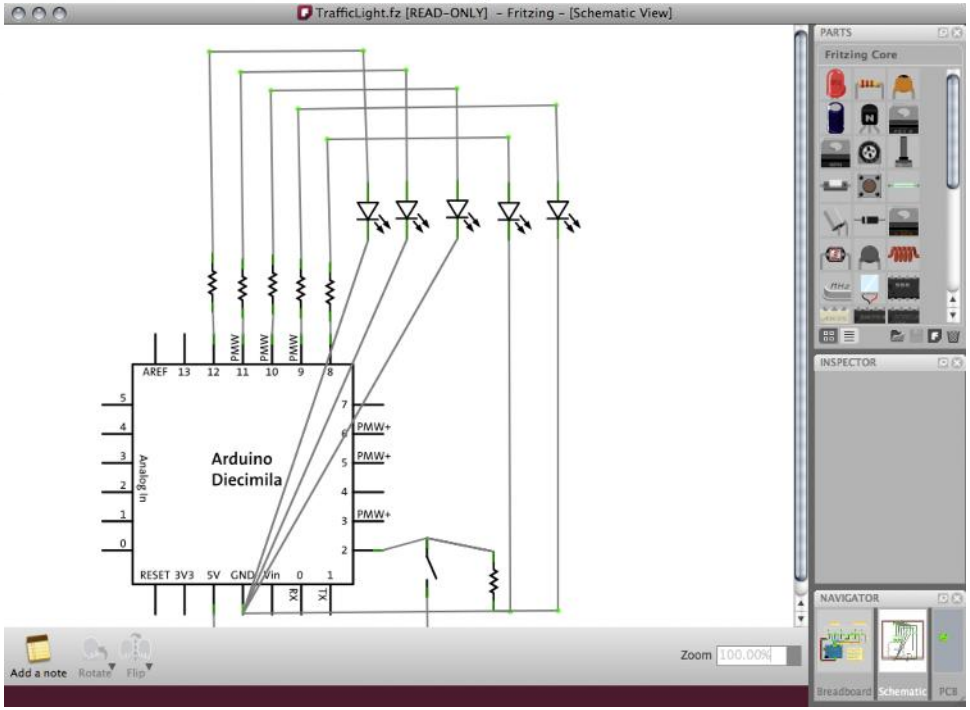


- Protoboard / Breadboard
  - ± Vertikal und Horizontal verbundene Pin-Leisten mit Rastermaß 2,54mm
  - ± Komponenten einfach einstecken und verbinden
  - ± Kein Löten notwendig
  - ± Fehler zerstörungsfrei beheben oder zerstörte Komponenten einfach austauschen ;)

Planung und Dokumentation

Fritzing (www.fritzing.org)





Lieferanten für Komponenten und Shields

sparkfun ELECTRONICS

CONRAD

Farnell

wetterott electronic

reichelt elektronik

UNIVERSITÄT SIEGEN

Beispiele

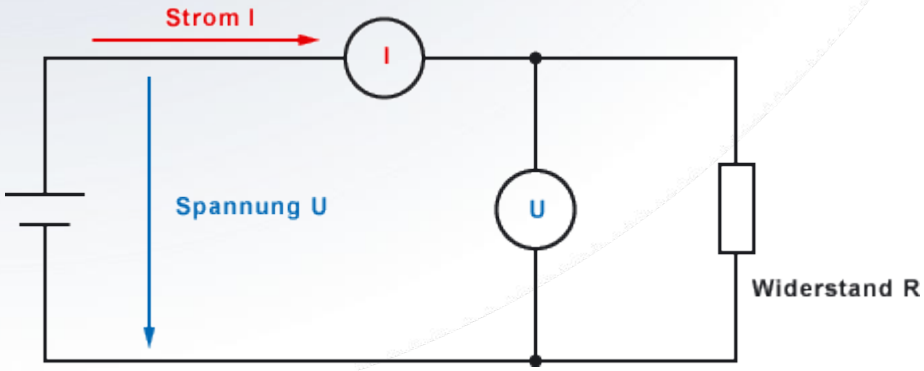
- TableTalk project
  - http://www.youtube.com/watch?v=jEGwsfpHevU
- Selfmade ambilight
  - http://www.youtube.com/watch?v=Am55k0k9eq8
- Light Seeking Arduino Robot
  - http://www.youtube.com/watch?v=BuWAZL51YSM
- The Fun Theory
  - http://www.youtube.com/watch?v=Z0gmtoqvGg8&playnext=1&list=PL82314C4102569410
  - http://www.youtube.com/watch?v=2lXh2n0aPyw
  - http://www.youtube.com/watch?v=cbEKAwCoCKw

Exkurs Elektrotechnik

- Auffrischen von Grundlagen
  - Hier nur das allernötigste
  - kein wissenschaftlicher Anspruch
  - Anschaulich aber falsch :)
  - „Lügen für Erwachsene“

- Das wichtigste vorab
  - Wir arbeiten nur mit Gleichstrom
  - Strom „besteht“ aus negativ geladenen Elektronen, beim Gleichstrom fliessen diese durch einen Leiter von der Spannungsquelle zur Senke
  - Technische Darstellung der Flussrichtung von Strom von + nach -
  - Physikalische Flussrichtung von – nach +

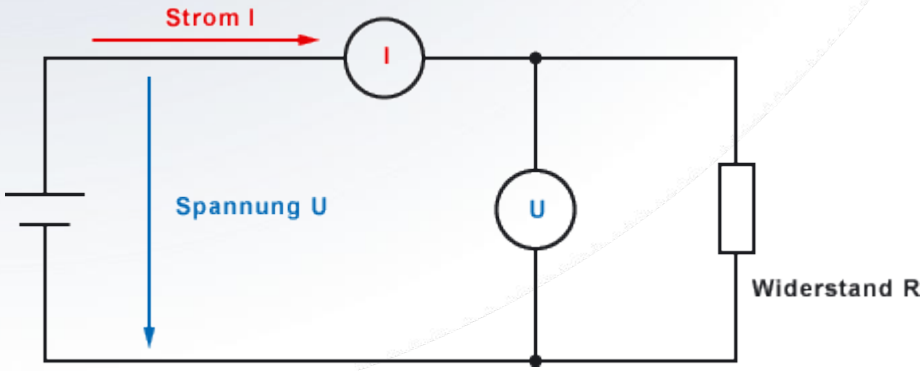
Exkurs Elektrotechnik



- Spannung
  - ± Variable U
  - ± Einheit V (Volt)
  - ± Finger weg von > 30V
  - ± Spannung ist die Energie die nötig ist um Elektronen von der Quelle zur Senke oder umgekehrt zu bewegen

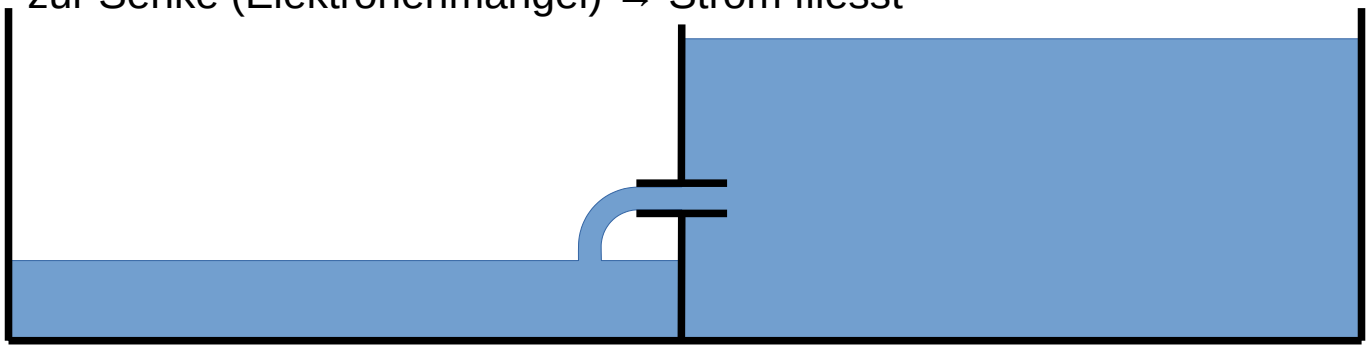
Ladungen im System wollen sich ausgleichen,
es entsteht eine Potentialdifferenz → Spannung

Exkurs Elektrotechnik

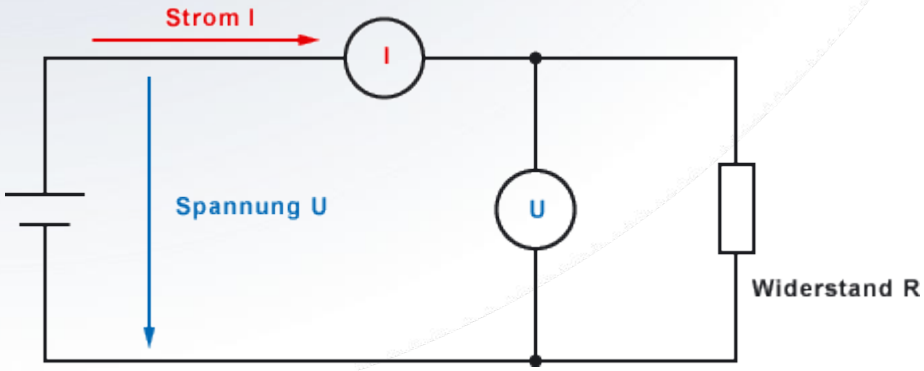


- Strom
  - Formelzeichen I
  - Einheit Ampere (A)
  - Potentialdifferenz wird ausgeglichen
  - Negativ geladene Elektronen sorgen für Ladungsverschiebungen

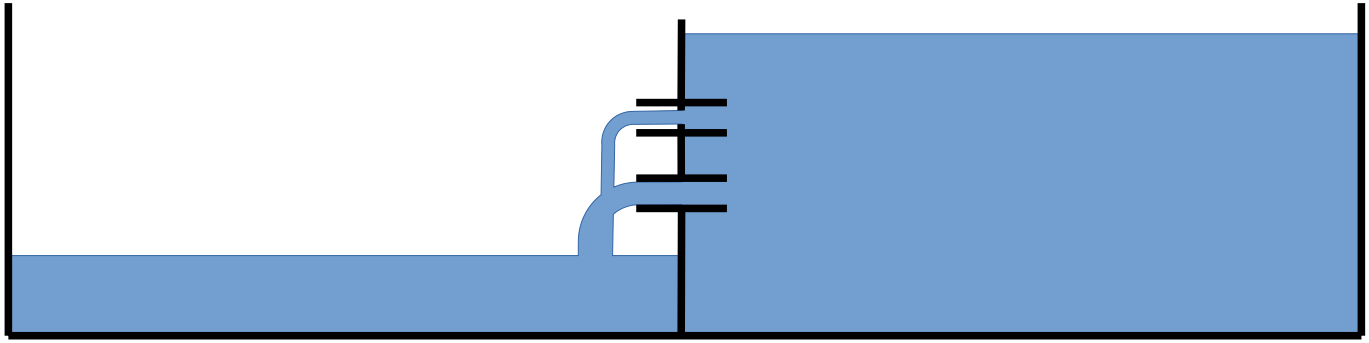
Elektronen fliessen von der Quelle (Elektronenüberschuss) zur Senke (Elektronenmangel) → Strom fliesst



Exkurs Elektrotechnik



- Widerstand
  - ± Variable R
  - ± Einheit Ω (Ohm)
  - ± Je kleiner der Widerstand desto mehr Strom kann bei einer bestimmten Spannung fliessen
  - ± Je kleiner der Widerstand desto niedriger ist die benötigte Spannung um einen bestimmten Stromfluss zu ermöglichen



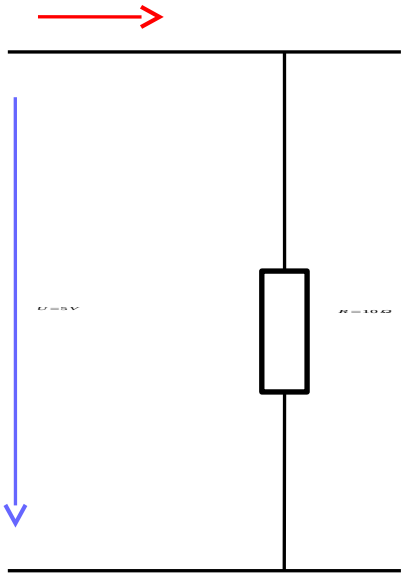
Exkurs Elektrotechnik

- Ohm‘sches Gesetz

$$U = R * I \Rightarrow I = \frac{U}{R} \Rightarrow R = \frac{U}{I}$$

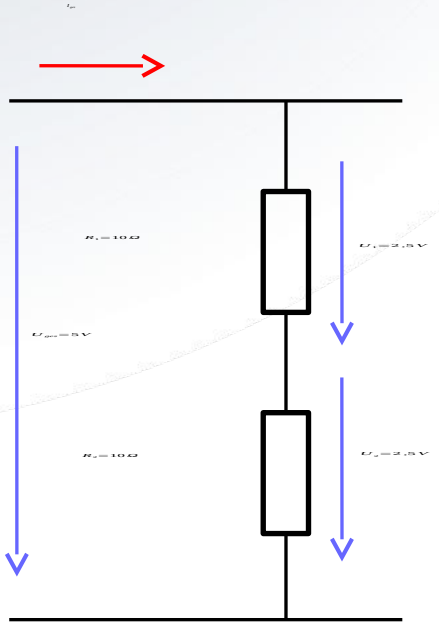
- Beispiel:
  - Der Strom I, der durch eine Schaltung fließt, ergibt sich aus der angelegten Spannung und dem Widerstand der Schaltung

$$I = \frac{U}{R} = \frac{5V}{10\Omega} = 0,5A = 500mA$$



Exkurs Elektrotechnik

- Reihenschaltung
  - Die an der Schaltung angelegte Spannung U_{ges} teilt sich in die Teilspannungen U_1, U_2
    $U_{ges} = U_1 + U_2$
  - Der Gesamtwiderstand der Schaltung R_{ges} ist gleich der Summe aller Teilwiderstände
    $R_{ges} = R_1 + R_2$
  - Der Strom ist an jeder Stelle konstant
    $I_{ges} = I_1 = I_2$
  - Je höher der Widerstand, umso höher die Spannung, die „über dem Widerstand abfällt“
  - Die Gesamtspannung teilt sich im Verhältnis der Widerstände auf.



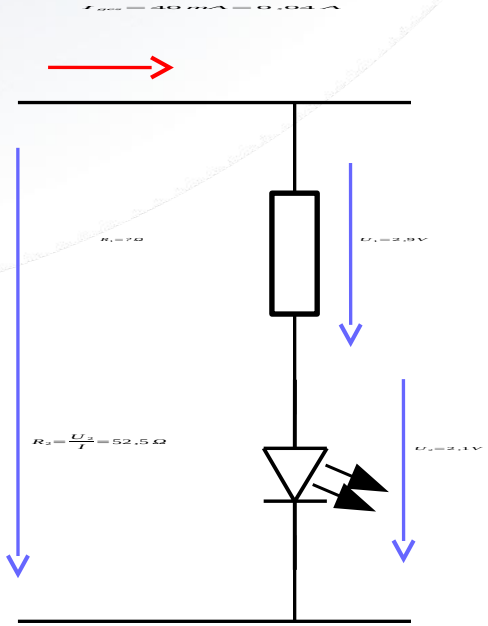
$$I = \frac{U_{ges}}{R_{ges}} = \frac{U_1}{R_1} = \frac{U_2}{R_2}$$

$$I_1 = \frac{U_1}{R_1} = \frac{2{,}5\,V}{10\,\Omega} = 0{,}25\,A = 25$$

$$I_2 = \frac{U_2}{R_2} = \frac{2{,}5\,V}{10\,\Omega} = 0{,}25\,A = 25$$

Exkurs Elektrotechnik

- Beispiel : Vorwiderstand für LED berechnen
  - U_{ges} = 5V
  - U_D laut Datenblatt 2,1V (Grün)
  - I_D laut Datenblatt 40mA
  - U_R soll also $U_{ges} – U_D$ = 2,9V betragen, sprich 2,9V müssen vernichtet werden
  - $R_V = U_R / I = 2,9 / 0,04 = 72,5Ω$
  - 72,5Ω Widerstände gibt es nicht → nächst höheren nehmen → 82Ω
  - Vorwiderstände sollte man nur bei kleinen Strömen einsetzen da die zu vernichtende Spannung in Wärme umgewandelt wird



Exkurs Elektrotechnik

- Leistung
  - Formelzeichen **P**
  - Einheit Watt (W)
  - Produkt aus Spannung und Strom
    P = U * I

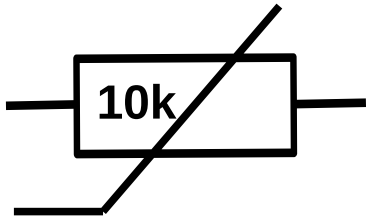
- Maß für
  - Leistungsfähigkeit
  - Stromaufnahme

- Beispiel Vorwiderstand LED
  - U = 2,9 V
  - I = 40 mA
  - P = U * I = 2,9 V * 0,04 A = 0,116 W
  - Wir benötigen also einen 0,25 W Widerstand

Widerstand

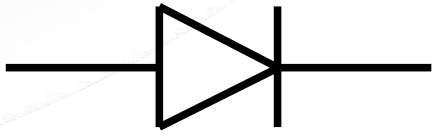
- Jeder Leiter und jedes Bauteil hat einen Widerstand, lediglich bei Supraleitern geht der Widerstand gegen 0
- Widerstände mit fixen Werten gibt es als Bauteil
  - Nennwerte von Widerständen werden nach geometrischen Folgen abgestuft. Allgemeine Praxis : wenn es keinen passenden Widerstand gibt nimmt man den nächst höheren

- Widerstände mit variablen Werten in einem bestimmten Bereich nennt man Potentiometer
- Widerstände haben eine zulässige Maximal-Leistung
- Anwendung
  - Vorwiderstand
  - Spannungsteiler
  - Erzeugung eines definierten Pegels (PullUp oder PullDown, dazu später noch mehr...)

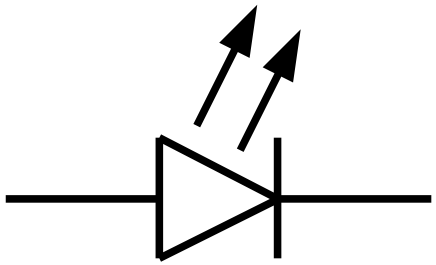


Diode

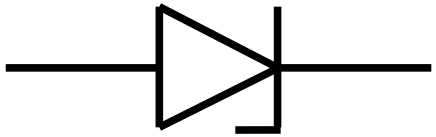
- Dioden können vereinfacht als „Rückschlagventil“ verstanden werden, d.h. Strom wird nur in einer Fließrichtung hindurch geleitet:
  - Durchlassrichtung
  - Sperrichtung



- Viele unterschiedliche Arten von Dioden:
  - Dioden
  - Leuchtdioden (Light-Emitting-Diode => LED)
  - Zener- oder Z-Dioden (haben niedrige Sperrspannung, werden in Sperrrichtung betrieben, leiten dann alles oberhalb der Sperrspannung ab.)

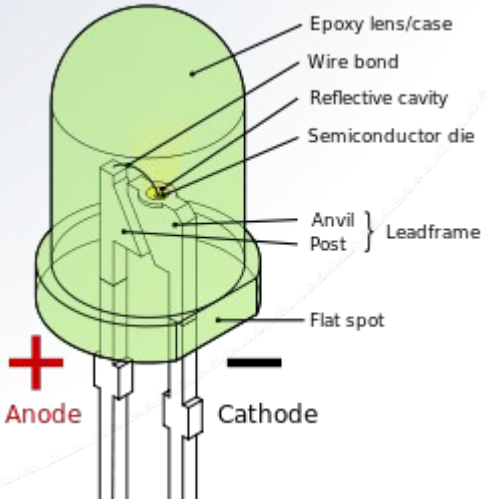


- Unterscheidung zwischen Durchlassspannung und Sperrspannung
  - Ab der Durchlassspannung leitet die Diode in Durchlassrichtung
  - Bis zur Sperrspannung sperrt die Diode in Sperrrichtung



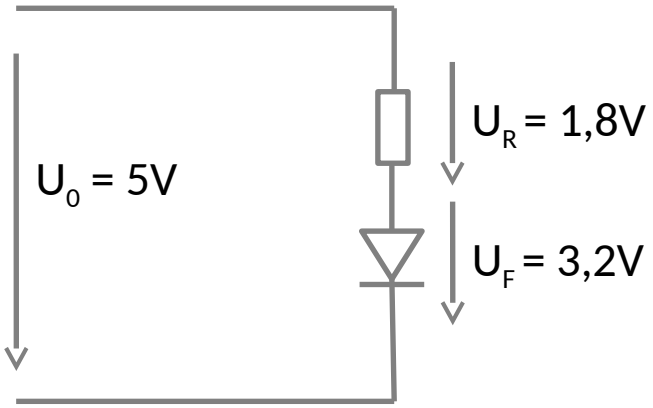
- Anwendung
  - Schutz vor Überspannung
  - LED.....
  - In Sperrrichtung geschaltete Z-Dioden eignen als Spannungsregler / Überspannungsschutz

LEDs im Detail



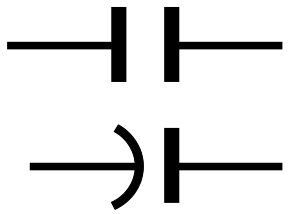
LEDs im Detail

- LED:
  - Licht emittierendes Halbleiter-Bauelement
  - elektrische Eigenschaften einer Diode
  - Fließt durch die Diode elektrischer Strom in Durchlassrichtung, so strahlt sie Licht
  - Wellenlänge von Material (Substrat) und Dotierung abhängig
  - Diode öffnet bei UF vollständig, d.h. der Widerstand sinkt stark ab (theoretischer Kurzschluss)
  - Diode emittiert Licht nahe UF. Wird UF zu stark unterschritten, steigt Widerstand stark an. Es wird kein Licht emittiert.
  - LED über Spannungsänderung nur bedingt dimmbar.



Kondensator

- Kleine „Batterien“
- Laden sich schnell auf
- Entladen sich schnell
- Lade- und Entladeeigenschaften können ausgenutzt werden
  - ± Debouncing
  - ± Schwingkreise



Schaltsymbole

Digitale Ausgabe am Arduino

- Die Digitalen Pins am Arduino können als Ein- oder Ausgabepins konfiguriert werden
- Die Pins sind Nummeriert und über die Nummer kann man einzelne Pins konfigurieren und ansprechen
- Die Konfiguration erledigt das laufende Programm
  - bei der Arduino IDE über den pinMode() Befehl
  - PinMode(13, OUTPUT) setzt Pin Nummer 13 als Ausgabe Pin
- Zustände von Ausgabe Pins
  - Digitale Ausgabe Pins können im Allgemeinen zwei Zustände annehmen, HIGH oder LOW
  - Je nach Atmega Controller bedeutet HIGH dass an dem Pin 3,3V oder 5V anliegen
  - Bei LOW liegen dementsprechend 0V an
  - Je nach Atmega Controller kann ein Ausgabe Pin eine Stromstärke von 20-50 mA liefern
  - Eine zu hohe Last am Pin kann diesen bzw. den Transistor hinter dem Pin zerstören, also ggfs. mit Widerstand schützen
  - Es gibt besondere Pins, mit PWM oder ~ gekennzeichnet, diese betrachten wir später genauer

Hands on! Digitale Ausgabe

- Ziel: Lasst die LED blinken

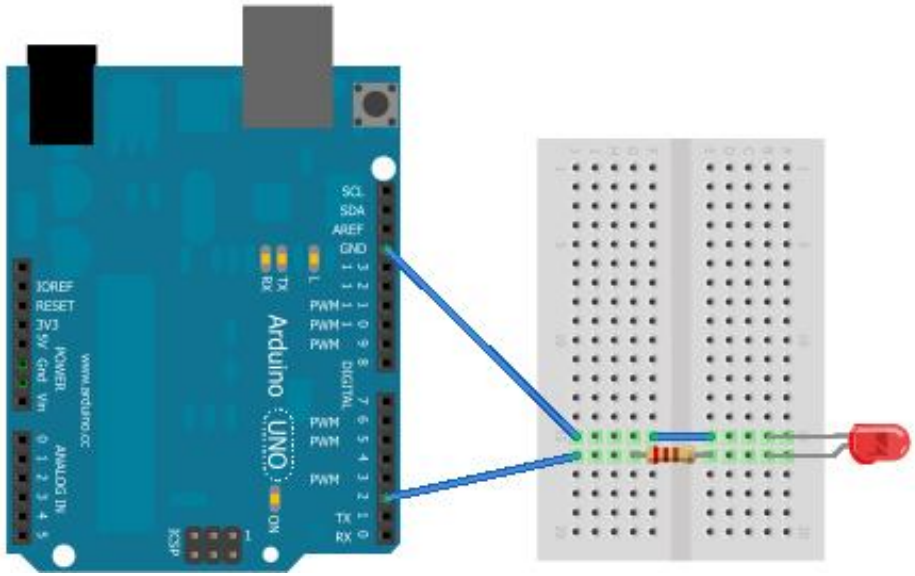
- Schritte:
  - Vorwiderstand errechnen
  - Schaltung aufbauen
  - Schaltung mit Arduino verbinden
  - Code schreiben
  - Mikrocontroller auf dem Arduino programmieren

- Spielt rum:
  - Ändert Parameter
  - Zerstört die LED
  - Lasst euch inspirieren
  - Habt Spass!

Tools

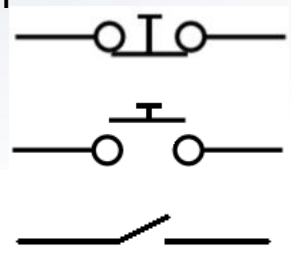
- Fritzing
- Arduino IDE
- Breadboard
- Arduino
- Jump-Wire
- LEDs mit Datenblättern
- Vorwiderstände

Aufbau der Schaltung

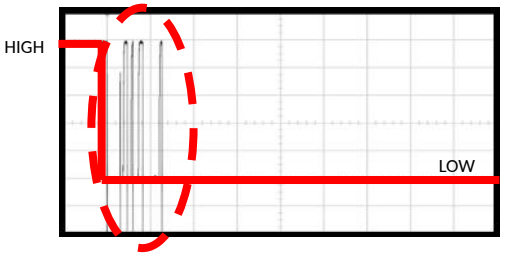


Schalter

- Es gibt viele unterschiedliche Bauformen und Arten von Schaltern. Am gängigsten sind
  - Schalter - schliessen bzw. öffnen dauerhaft
  - Taster - schliessen bzw. öffnen temporär



- Schalter sind nicht besonders spannend aber beim Anschluss an den Mikrocontroller sind gewisse Besonderheiten zu beachten
  - Je nach Schaltermaterial und Bauform ist der Schaltvorgang nicht „sauber“ und sog. „bouncing“ oder flattern tritt auf, d.h. es können bei einem Schaltvorgang am Schalter mehrere Schaltvorgänge vom Controller erkannt werden

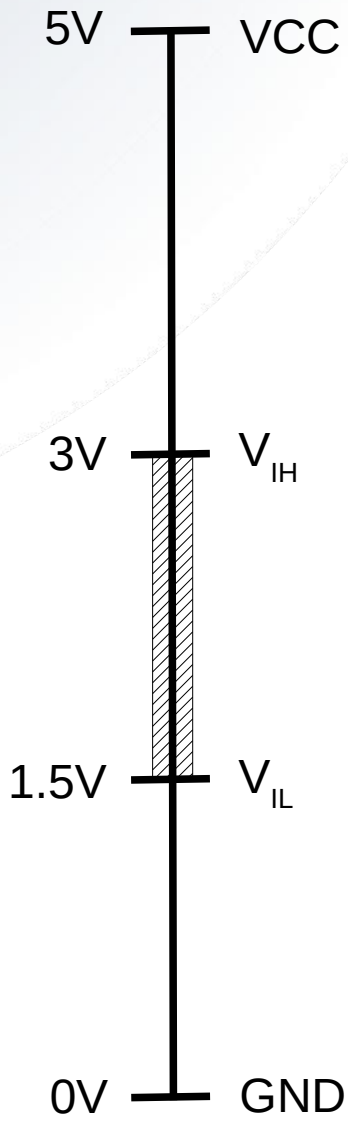


  - Entweder verhindert man das „bouncing“ über zusätzliche Komponenten wie Kondensatoren und Trigger oder man nutzt die Möglichkeiten des Mikrocontrollers, d.h. man überprüft z.b. über einen gewissen Zeitraum ob der Schalter gedrückt ist und kann das „bouncing“ dadurch zwar nicht verhindern aber kompensieren
  - Für Schalter gibt es extra Bibliotheken die den Umgang vereinfachen →Arduino Debounce Library

Digitale Eingabe am Arduino

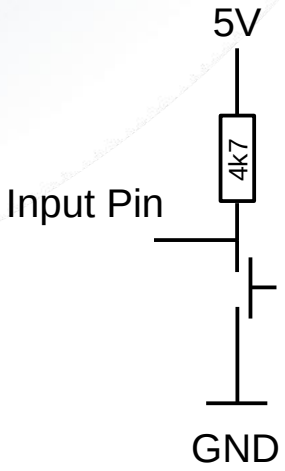
- Konfiguration als digitaler Eingabe Pin
  - bei der Arduino IDE über den pinMode() Befehl
  - PinMode(2, INPUT) setzt Pin Nummer 2 als Eingabe Pin

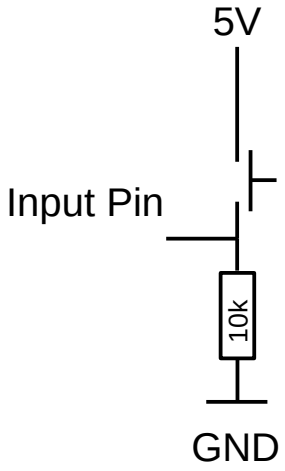
- Zustände von Eingabe Pins
  - Digitale Eingabegabe Pins können zwei Zustände annehmen : HIGH oder LOW
  - Minimum Spannung um HIGH zu erkennen liegt bei 3 V
  - Sobald die Spannung am Pin 1.5 V unterschreitet wird LOW erkannt
  - Der Bereich dazwischen ist undefiniert, d.h. in diesem Bereich gibt es keinen festen Zustand HIGH oder LOW
  - Eingabe Pins haben einen sehr hohen Widerstand, reagieren also bereits auf geringe Ströme und sind deshalb empfindlich gegen EMF



Pull Up/Down Widerstand am Digitalen Eingang

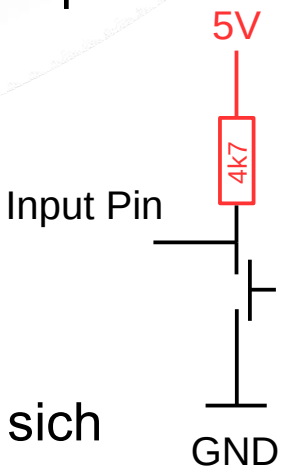
- Ein Pullup- oder Pulldown-Widerstand wird dazu verwendet, einen Eingang auf einen definierten Wert ein zu stellen
  - Normalerweise ist der Eingang im Zustand "schwebend/hochohmig" also zwischen HIGH und LOW
  - Durch EMF und Induktion kann kurzzeitig ein Wert über- oder unterschritten werden und der Eingang plötzlich ein HIGH oder LOW Signal bekommen. Dies führt dann zu unerklärlichen und unregelmäßig auftretenden Fehlern
- Pullup-Widerstand
  - Zwischen VCC und dem Eingang ein hochohmiger Widerstand
  - Während nichts am Eingang passiert, wird dieser fest auf HIGH „eingestellt“
  - Bei geschlossenem Taster wird nun der Strom über den Leiter mit dem geringeren Widerstand nach GND abfließen und LOW gelesen
- Pulldown-Widerstand
  - Der Pulldown-Widerstand funktioniert analog zum Pullup-Widerstand
  - Wird zwischen GND und dem Eingang platziert
  - Störende Impulse können nach GND abfliessen
  - Schließt man den Taster, liegt am Eingang Vcc an, es kann nicht alles über den Widerstand abfliessen, also wird HIGH gelesen
- Dimensionierung
  - Allgemeine Praxis sind 4,7 kOhm für Pullup- bzw. 10 kOhm für Pulldown-Widerstände





Interne PullUp Widerstände

- Interne PullUp Widerstände
  - Der Atmega Mikrocontroller hat eingebaute 20k Pullup Widerstände an den Digitalen Pins
  - Beispiel Code:
    pinMode(pin, INPUT);
    digitalWrite(pin, HIGH); //pullup aktivieren
  - Durch die Nutzung der internen Pullups kann man sich den Rot markierten Teil der Schaltung sparen



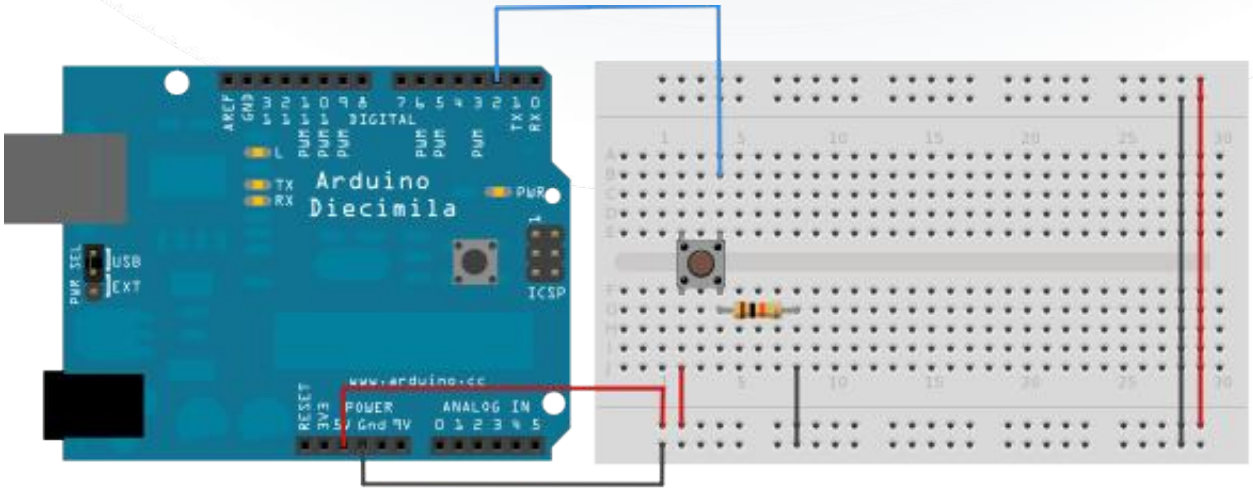
Hands on! Digitale Eingabe

- Ziel: Schaltet die LED

- Schritte:
  - Schaltung aufbauen
  - Schaltung mit Arduino verbinden
  - Die Debounce Library installieren
  - Code schreiben
  - Mikrocontroller auf dem Arduino programmieren
  - LED schalten

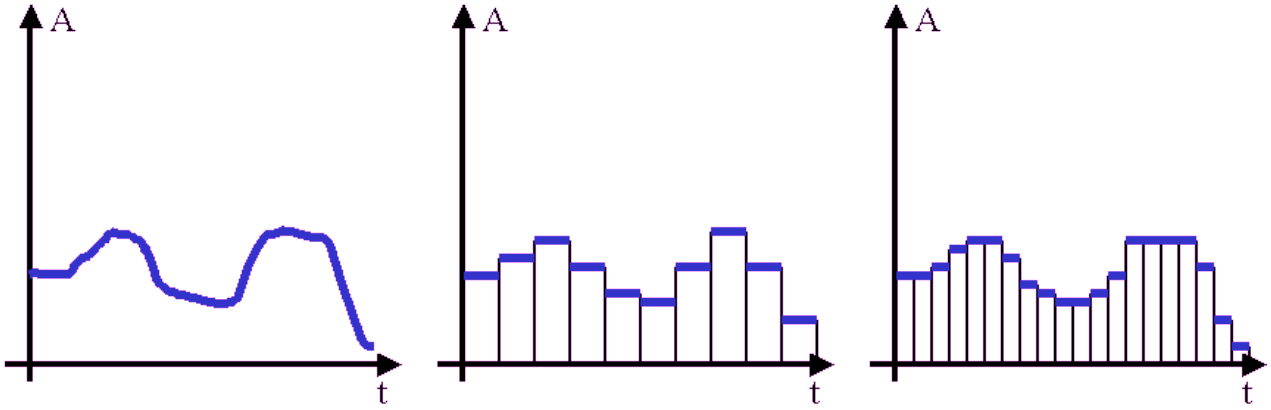
- Spielt rum:
  - Ändert Parameter
  - Lasst den Pull-Down Widerstand weg
  - Lasst euch inspirieren
  - Habt Spass!

Aufbau der Schaltung



Analoge Signale

- Analoge Signale haben keine festen Zustände (HIGH/LOW) sondern setzen sich aus variablen Werten zwischen zwei Pegeln, z.B. 0V und 5V zusammen
- Ein D/A-Wandler (Digital/Analog) setzt diese Eingangsspannung am analogem Pin um in Digitale Werte
- Je höher die Auflösung des D/A Wandlers in Bit desto feiner die Werte
- Ein 10 Bit D/A Wandler kann die gelesene Spannung in Werte zwischen 0 und 1023 auflösen, ein 12 Bit D/A Wandler in Werte zwischen 0 und 4095
- Die Abtastrate in Hz legt fest wie viele Werte pro Sekunde verarbeitet werden (z.B. 48KHz bei Audio)



Analog signal – continuously varying

Digital signal – large time divisions

Digital signal – small time divisions

Analoge Eingabe am Arduino

- Der Atmega Mikrocontroller verfügt über mehrere Analoge Input Pins
  - Markiert mit A0-Ax
  - Die Analogen Pins können angelegte **Spannungen** zwischen 0 und 5V bzw. 3.3V lesen
  - Können auch als Digitale I/O Pins benutzt werden
- Die Atmega8, Atmega168, Atmega328, und Atmega1280 verfügen über einen 10 Bit Analog/Digital Konverter
  - Analoge Eingaben werden in Werte von 0 – 1023 übersetzt
  - Die Werte zwischen 0 und 5V können also in 5 / 1024 = 0,00488 V Abständen gelesen werden
- Über den AREF Pin kann man eine Referenzspannung anlegen
  - Hat man einen Sensor der Werte zwischen 0 und 2,5V liefert kann man 2,5V am AREF Pin anlegen. Der D/A Wandler kann nun in 2,5 / 1024 = 0,00244V Abständen lesen, ist somit genauer

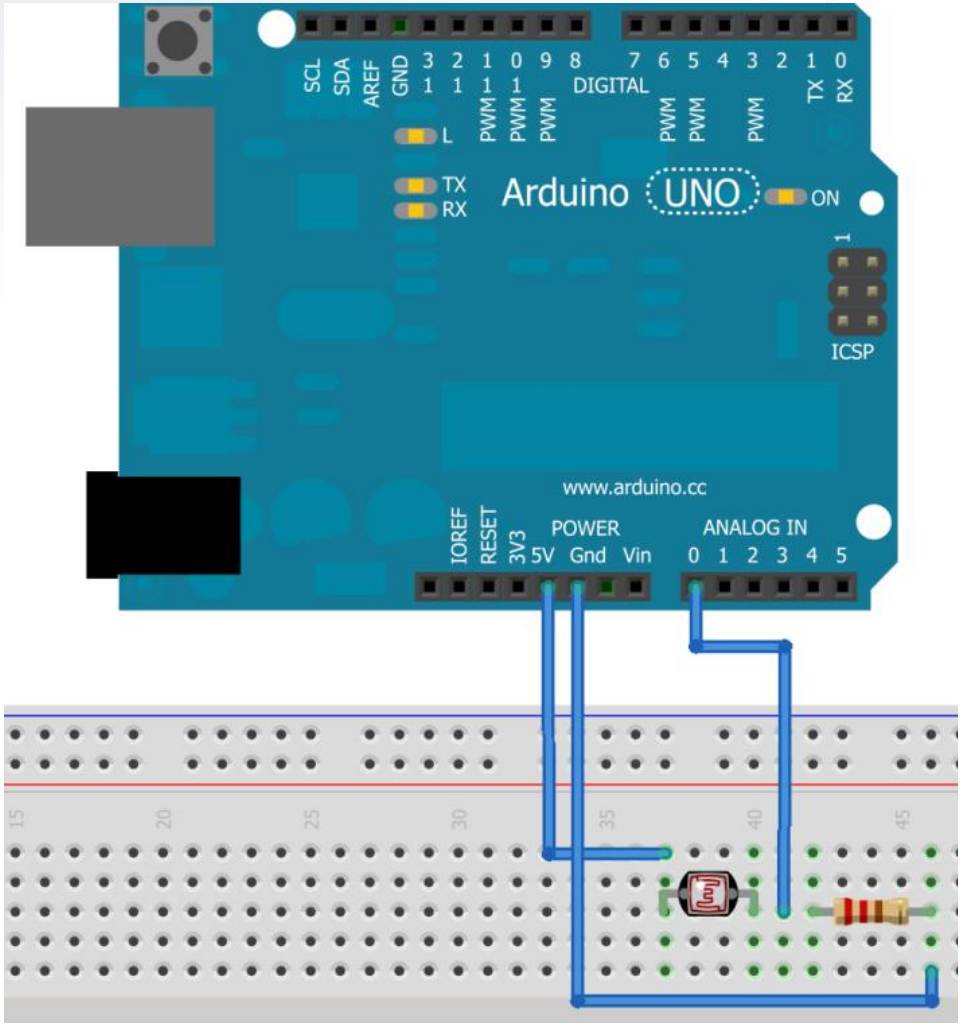
Analoge Eingabe am Arduino

- Die Analogen Input Pins müssen nicht initialisiert werden, es sei denn sie sollen als Digitale Pins genutzt werden
- Analoge Werte werden mit dem „*analogRead*“ Befehl gelesen:

```
int wert = 0;
wert = analogRead(0);
```

weist der Variable „*wert*“ den aktuellen Analog Wert an Pin A0 zu

- An diesem Beispiel sieht man dass Analogen Pins Spannungen messen und keine Ströme. Der Lichtempfindliche Widerstand regelt den Strom, wenn man ihn also direkt an A0 anschliesst misst man kontinuierlich 5V. Man muss sich also einen Spannungsteiler bauen und kann nun den vom Strom abhängigen Spannungsabfall hinter dem LDR messen

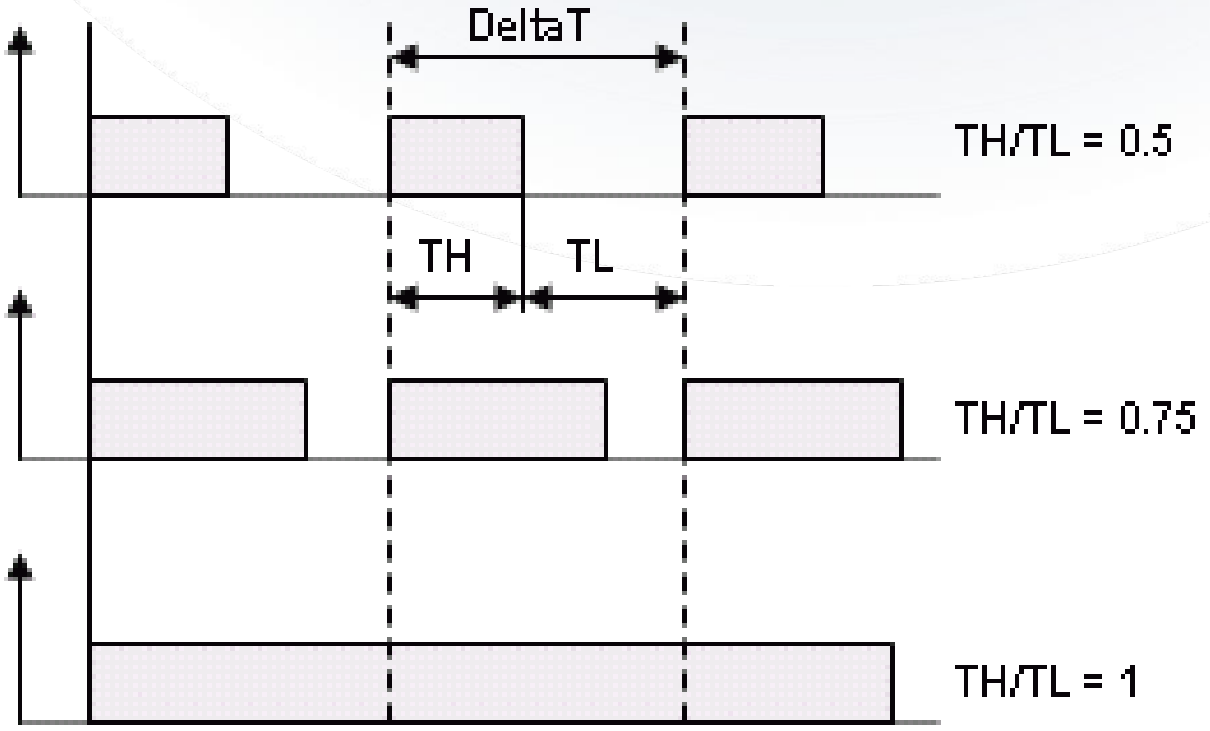


Analoge Ausgabe am Arduino

- An dieser Stelle würde man die „Analoge Ausgabe“ erwarten, es gibt allerdings keine richtige Analoge Ausgabe
  - ± Die Analogen Pins können nur lesen
  - ± Es gibt einige spezielle Digitale Pins die mit „PWM“ oder „~“ gekennzeichnet sind
  - ± Diese Pins beherrschen Pulsweiten Modulation, d.h. diese Pins können das Ausgabe Signal mit einer bestimmten Frequenz zu bestimmten Anteilen HIGH/LOW schalten
    - Frequenz Pins 3, 9, 10, und 11 ist 31250 Hz
    - Frequenz Pins 5 und 6 ist 62500 Hz
  - ± Ist ein digitaler Pin (normal 5V) zu 50% an und zu 50% abgeschaltet erkennt ein träges Analoges Bauteil das als 2,5V
  - ± Der Anteil kann mit einem 8 Bit Wert, also 0 - 255 angegeben werden
    - 127 wären also 50% HIGH, 50% LOW, also 2,5V

Pulse-Width-Modulation

Pulsweiten-Modulation (PWM)



TH/TL ~ n

PWM und Arduino

Pulsweiten-Modulation (PWM)

- Arduino Mega 2560
  - ± 15 digitale PWM Pins
- Arduino Uno
  - ± 6 digitale PWM Pins
- Arduino Leonardo
  - ± 7 digitale PWM Pins
- Hängt vom verbauten mC ab

PWM und Arduino

Pulsweiten-Modulation (PWM)

```
// the setup routine runs once when you press reset:
void setup()  {
  // declare pin 'led' to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop()  {
  // set the brightness of pin 'led':
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;
}
```

PWM und Arduino

Pulsweiten-Modulation (PWM)

- Syntax:
  `analogWrite(pin, value);`

  `pin`: the pin to write to
  `value`: the duty cycle: between 0 (always off) and 255 (always on)

- Auflösung
  - Standardmäßig 8 Bit = 255
  - Arduino DUE hat zwei spezielle Pins, deren Auflösung auf 12 Bit erhöht werden kann (nur mit Arduino DUE möglich)
  - 12 Bit = 4096

- Syntax:
  `analogWriteResolution(bits);`

PWM und Arduino

Pulsweiten-Modulation (PWM)

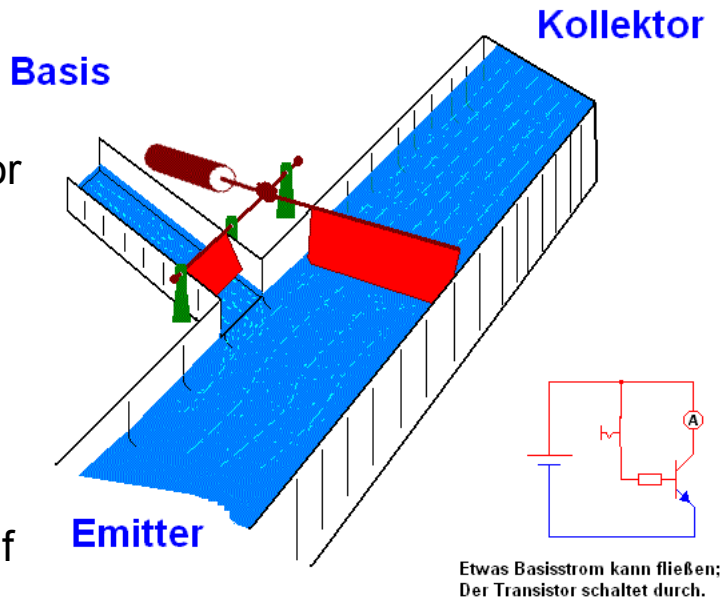
- Beispiel RGB LED:
  Wie viele Farben kann man mit 3 typischen Arduino PWM Pins und einer RGB-LED darstellen?

- 8 Bit Auflösung pro Pin:
  - 256 Rot-Intensitäten
  - 256 Grün-Intensitäten
  - 256 Blau-Intensitäten
  - 256 * 256 * 256 = 16.777.215 = 24Bit Farbtiefe

- http://www.youtube.com/watch?v=2x4CbOOOFvE

Verstärker mit Transistoren

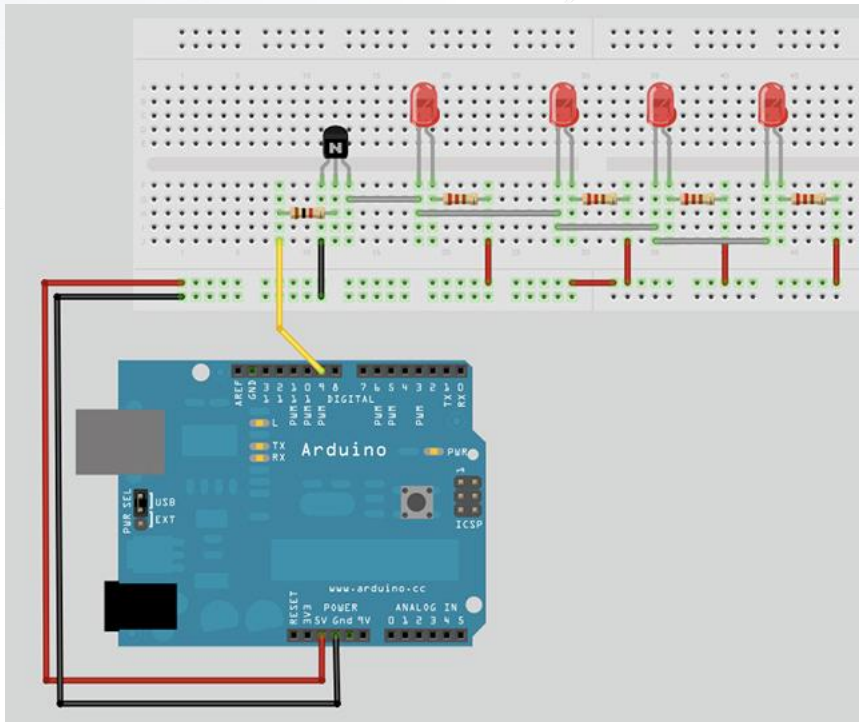
- Ein digitaler Ausgang des Arduino Duemilanove kann einen Strom von 40 mA bereitstellen. Das reicht für kleine Signale aus. Will man aber z.B. mehrere LEDs, Motoren oder Elektromagneten schalten, ist das zu wenig. Hierfür benötigt man eine Verstärkerschaltung
- Es gibt jede Menge Bauformen und Arten von Transistoren, aber im Allgemeinen kann man einen Transistor mit einem regelbaren Ventil vergleichen. Ein Transistor hat drei Anschlüsse, die Basis, den Kollektor und den Emitter. Sobald an der Basis ein kleiner (Steuer)Strom fleisst kann durch den Kollektor ein grosser (Last)Strom fliessen
- Mit einem Transistor kann man somit sehr hohe Spannungen und Ströme mit einem Mikrocontroller schalten
- Bei hohen Spannungen und oder Strömen für Kühlung sorgen!
- Da der Transistor ein Halbleiterbauelement ist, ist auf die Polung zu achten



Etwas Basisstrom kann fließen;
Der Transistor schaltet durch.

Transistoren und Arduino

- Die Basis des Transistors wird über einen Widerstand (1 – 10 kΩ) mit den DigitalOut verbunden. Liegt an der Basis ein HIGH-Signal an, so schaltet sich der Transistor ein und die LEDs können leuchten. Der Hauptstrom fließt nun nicht mehr von DigitalOut zu den LEDs, sondern vom 5V+ über die LEDs, durch den Transistor in den GND. Der Strom, der aus dem DigitalOut kommt wird also nur zum schalten des Transistors benötigt
- Bei Motoren benötigt man mindestens zwei Transistoren oder eine H-Brücke (IC), wenn man den Motor in beide Richtungen laufen lassen möchte
- Bei Motoren, Schrittmotoren oder Servos rate ich zum Arduino Motor Shield

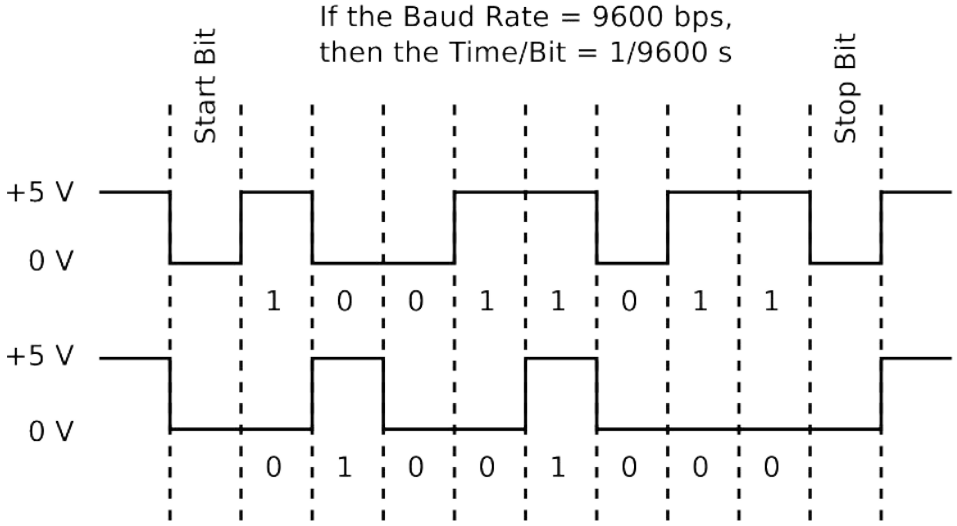


Serielle Kommunikation

Abgrenzung

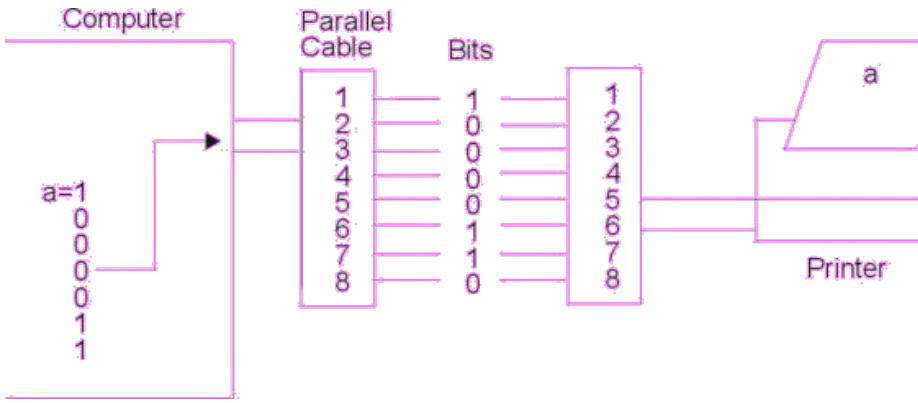
Serielle Kommunikation

- Übertragungsmodus
  - ± Übertragung von Daten über eine Signalleitung *nacheinander*



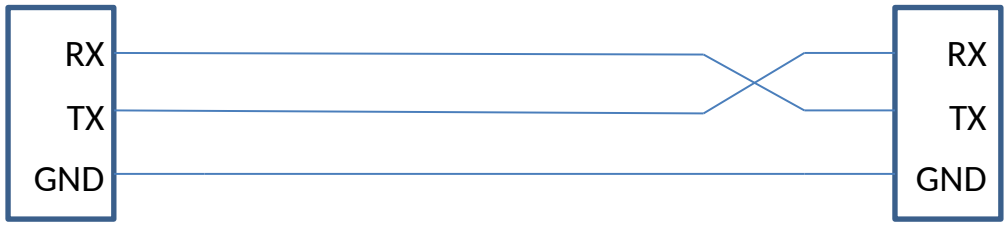
Parallele Kommunikation

- Übertragungsmodus
  - ± Übertragung von Daten über mehrerer Signalleitungen *nebeneinander*



UART

- **U**niversal **A**synchronous **R**eceiver **T**ransmitter
  - *Serielles* Protokoll
  - Exklusive Leitung notwendig (kein Bus-System)
- Schaltung
  - RX (Receive)
  - TX (Transmit)
  - GND (Masse als gemeinsame Refrenz)



UART Arduino Code Snippets

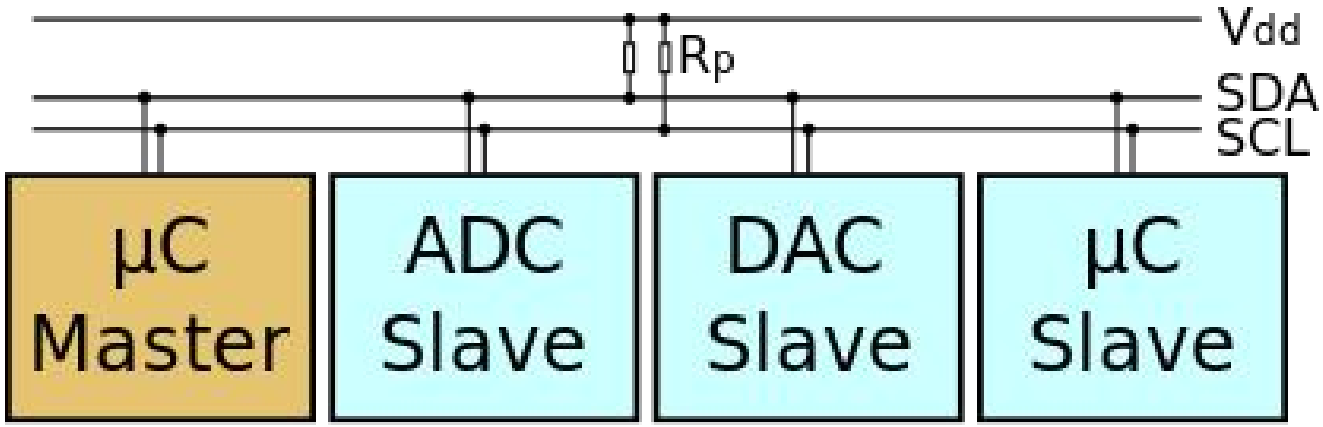
Wesentliches

- Initialisierung:
  - Serial.begin(int baudrate);// Serial initialisieren
  - while (!Serial) ; // Warten bis Init fertig
- Schreiben (senden) und Lesen (empfangen)
  - Serial.println(char[]); // formatiertes Schreiben von Chars
  - Serial.print(char[]); // formatiertes Schreiben von Chars
  - Serial.write(byte[]); // unformatiertes Schreiben von Bytes
  - byte Serial.read(); // byte-weises Lesen am RX-Pin
  - boolean Serial.available();// Prüfen, ob weitere Bytes eingehen
- Schließen
  - Serial.end()

Bus-Systeme | I²C

- Master/Slave Architektur
- Adressierung:
  - I²C-Adresse ist das erste vom Master gesendete Byte
  - die ersten sieben Bit die eigentliche Adresse
  - I²C hat einen Adressraum von 7 Bit
  - 112 Knoten auf einem Bus

- Beginn der Übertragung wird mit *Start*-Signal vom Master angezeigt,
- dann folgt die Adresse,
- Diese wird durch ACK-Bit vom adressierten Slave bestätigt
- Byte-weises Lesen vom oder Schreiben zum Slave
- Eine Übertragung wird durch das *Stop*-Signal beendet

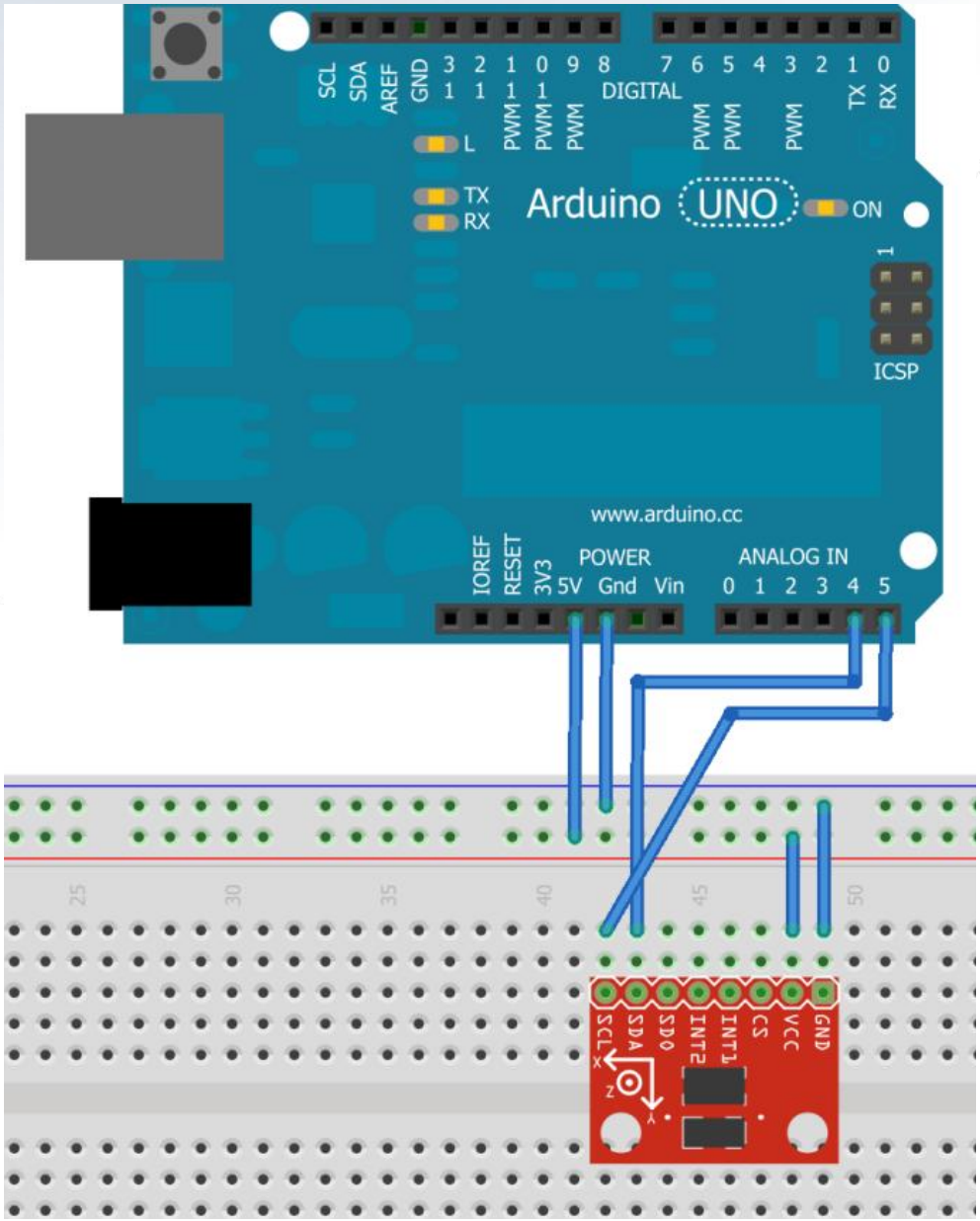


I²C Arduino

- 2 Pins für SDA und SDL

Board	I2C / TWI pins
Uno, Ethernet	A4 (SDA), A5 (SCL)
Mega2560	20 (SDA), 21 (SCL)
Leonardo	2 (SDA), 3 (SCL)
Due	20 (SDA), 21 (SCL), SDA1, SCL1

- abstrahiert alles notwendige



I²C Arduino Code Snippets

Wire Library

- Initialisierung
  - Wire.begin

- Senden
  - Wire.beginTransmission(byte deviceAddress);//start transmission
  - Wire.write(something); // send value to write
    Wire.write(somethingElse); // send value to write
  - Wire.endTransmission(); // end transmission

Spezielle Aspekte des HCI: Interaction Design with Arduino

Using the Arduino OpenHardware Platform to sketch and develop physical interactions and tangible user interfaces

Matthias Betz
Wirtschaftsinformatik und Neue Medien
Universität Siegen
matthias.betz@uni-siegen.de

UNIVERSITÄT SIEGEN

Vorverarbeitung von Sensorrohdaten

- Durch Eigenschaften bestimmten Messverfahren kommt es zu unerwarteten Phänomenen, die man behandeln muss.
  - ± Beispiel:
    Ultraschall-Reflektionen nicht perfekt, z.B. bei Stoff

- Notwendigkeit des Glättens und Filterns
  - ± Glätten: Ausgleichen von nicht plausiblen oder erwünschten Sprüngen
  - ± Filtern: Entfernen von nicht erwünschten Daten

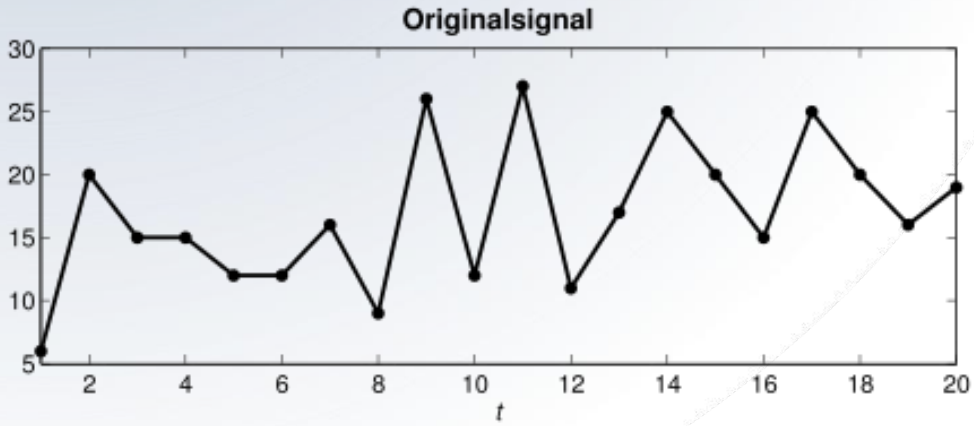
Glätten

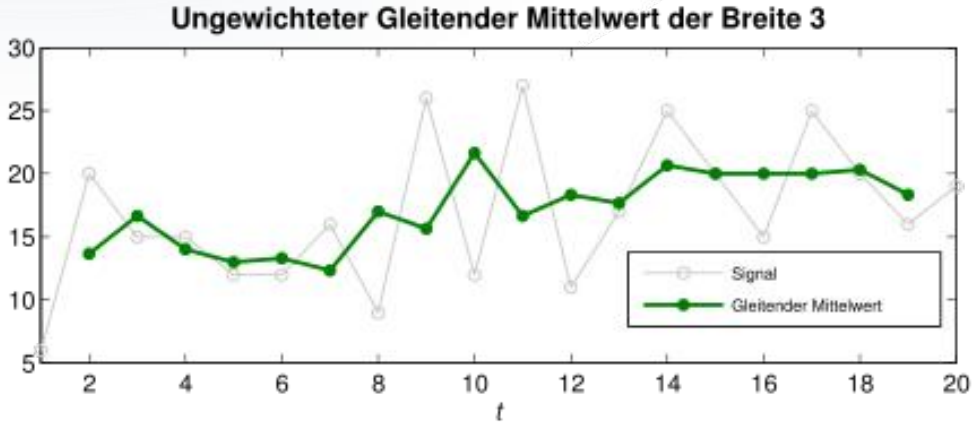
- Einfacher gleitender Mittelwert

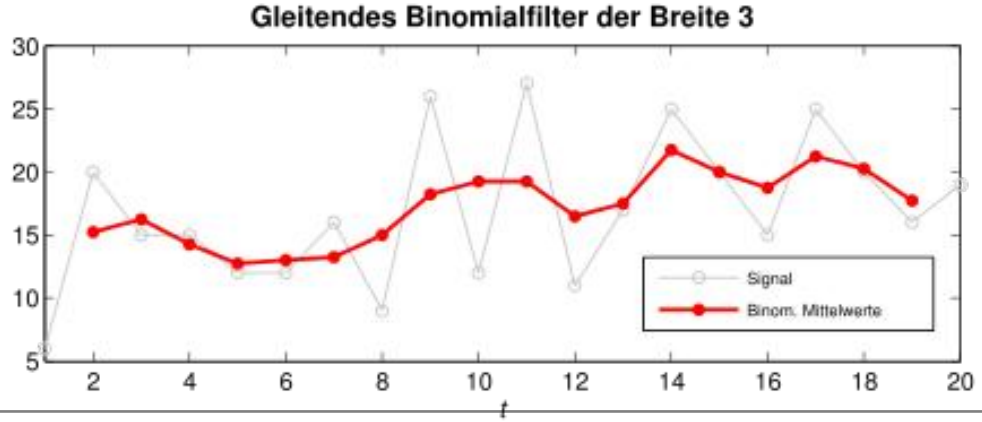
$$m_{MA}^{(n)}(t) = \frac{1}{n} \sum_{i=0}^{n-1} x(t-i)$$

- Gewichteter gleitender Mittelwert

$$m_x^{(n)}(t-\tau) = \sum_{i=0}^{n-1} w_i \, x(t-i)$$







Einfach gleitender Mittelwert

1	2	3	4	5	6	7	8	9	10	11
408	372	480	444	447	492	429	411	486	525	495

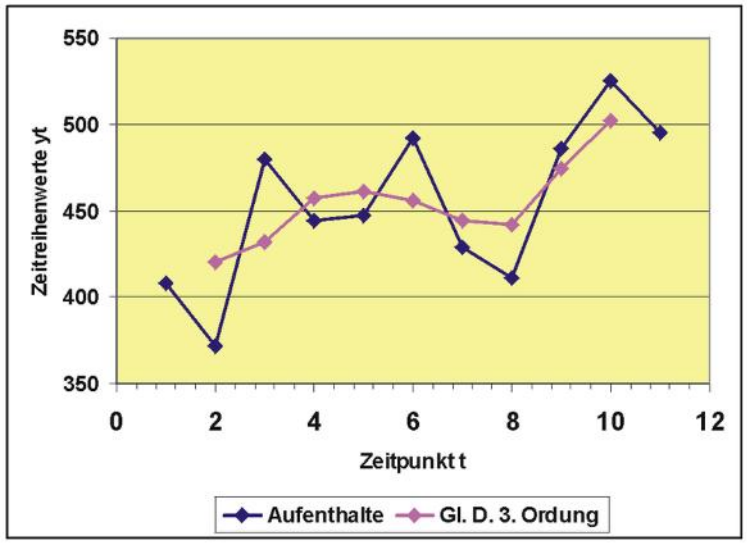
$$\overline{y}_2 = \frac{1}{3} \cdot (408 + 372 + 480) = 420$$

$$\overline{y}_3 = \frac{1}{3} \cdot (372 + 480 + 444) = 432$$

$$\overline{y}_9 = \frac{1}{3} \cdot (411 + 486 + 525) = 474$$

$$\overline{y}_9 = \frac{1}{3} \cdot (411 + 486 + 525) = 474$$

$$\overline{y}_{10} = \frac{1}{3} \cdot (486 + 525 + 495) = 502$$



Gewichteter gleitender Mittelwert

1	2	3	4	5	6	7	8	9	10	11
408	372	480	444	447	492	429	411	486	525	495

- Werte bei der Mittelwert-Generierung werden unterschiedlich gewichtet
- Gewichtungen müssen in Summe 1 sein, um Ergebnis nicht zu dämpfen / zu verstärken

$$m_x^{(n)}(t-\tau) = \sum_{i=0}^{n-1} w_i \; x(t-i)$$

$$w_0 = \tfrac{1}{4}, w_1 = \tfrac{1}{2}, w_2 = \tfrac{1}{4}$$

$$m_{BIN}(t-1) = \tfrac{1}{4}x(t) + \tfrac{1}{2}x(t-1) + \tfrac{1}{4}x(t-2)$$

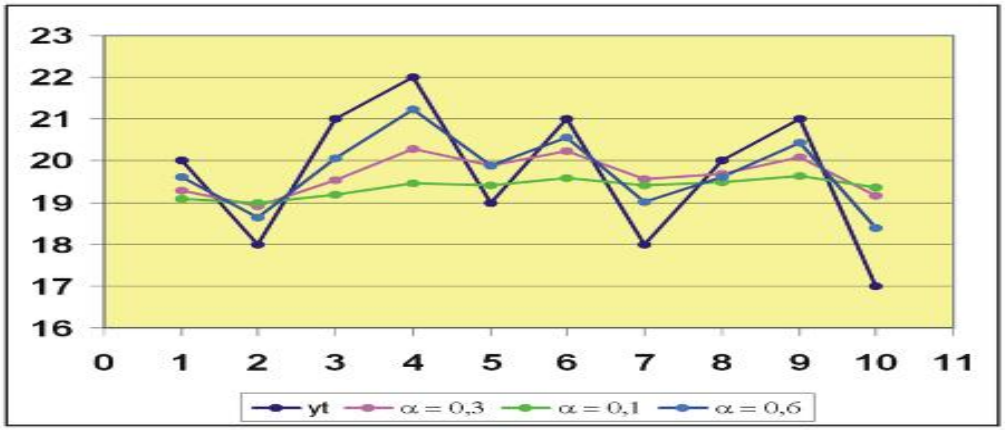
$$y_t^* = \alpha \cdot y_t + (1-\alpha) \cdot y_{t-1}^* \cdot$$

Exponentielle Glättung

1	2	3	4	5	6	7	8	9	10	alpha
20	18	21	22	19	21	18	20	21	17	-
19,3	18,91	19,54	20,28	19,89	20,23	19,56	...	...	...	**0,3**
19,1	18,99	19,19	19,47	19,42	19,58	19,42	...	...	...	**0,1**
19,6	18,64	20,06	21,22	19,89	20,56	19,02	...	...	...	**0,6**

$$y_1^* = \alpha y_1 + (1-\alpha) y_0^* ,$$
$$y_2^* = \alpha y_2 + (1-\alpha) y_1^* ,$$
$$y_3^* = \alpha y_3 + (1-\alpha) y_2^* ,$$

$$y_1^* = 0{,}3 \cdot 20 + 0{,}7 \cdot 19 = 6 + 13{,}3 = 19{,}3$$
$$y_2^* = 0{,}3 \cdot 18 + 0{,}7 \cdot 19{,}3 = 18{,}91$$



Aufgabe 1 : Zoetrope

- Mit einem Zoetrope kann man aus einigen statischen Bildern eine kurze bewegte Sequenz erzeugen, ähnlich einem Daumenkino
- Die Aufgabe besteht darin ein Mikrocontroller gesteuertes Zoetrope zu bauen
  - ± die Steuerung wird mit einem Arduino Mikrocontroller realisiert
  - ± die Motorsteuerung wird mit einer H-Brücke realisiert
    - der Motor soll sich in beide Richtungen drehen können wobei die Drehrichtung durch einen Schalter oder Taster geändert werden kann
    - die Geschwindigkeit soll mit einem Poti geregelt werden
  - ± es soll eine LED für die Beleuchtung eingebaut werden, die Frequenz muss nicht automatisch geregelt werden sondern kann auch durch ein weiteres Poti manuell an die Geschwindigkeit des Motors angepasst werden
- Hoher Bastel- und geringer Programmieraufwand

Aufgabe 2 : Magic Ball

- Ein Magic Ball gibt einfache Antworten auf die grossen Fragen des Lebens. Typischerweise stellt man dem Magic Ball eine Frage, schüttelt den Magic Ball und erhält daraufhin eine Antwort
- Die Aufgabe besteht darin einen elektronischen Magic Ball zu bauen der diese Funktionen realisiert
  - die Steuerung wird mit einem Arduino Mikrocontroller realisiert
  - die Ausgabe erfolgt auf einem LCD Display
  - die Bewegunserkennung erfolgt mit einem Gyroskop
  - das Gerät soll erkennen wenn es in die Hand genommen wird und den Nutzer auffordern eine Frage zu stellen
  - dann soll das Gerät erkennen wenn es geschüttelt wird und eine zufällige Antwort aus einem Katalog von mindestens 8 Antworten geben
- Gleichmässige Verteilung von Bastel- und Programmieraufwand

Aufgabe 3 : Flappy Bird auf LCD

- Flappy Bird ist ein einfaches Spiel bei dem der Spieler einen Vogel durch das Tippen auf den Bildschirm durch eine von rechts nach links scrollende Spielwelt führt, wobei der Vogel die paarweise von oben und unten ins Bild ragenden grünen Röhren nicht berühren darf, sondern zwischen ihnen hindurchfliegen muss. Die Position der Flugschneise variiert dabei
- Die Aufgabe besteht darin Flappy Bird auf einem 4 zeiligen LCD Display zu realisieren
  - die Steuerung wird mit einem Arduino Mikrocontroller realisiert
  - die Ausgabe erfolgt auf einem 4 Zeiligen LCD Display
  - die Eingabe erfolgt über einen Taster
- Geringer Bastel- und hoher Programmieraufwand

Vorgehensweise Aufgaben

- Planen
  - welche Komponenten werden benötigt
  - Datenblätter zusammensuchen
  - Schaltplan erstellen mit Fritzing oder auf Papier
- Bauen und Coden
  - Schaltung auf Breadboard zusammenbauen
  - Evtl. Komponenten bzw. Baugruppen einzeln testen (LCD, Gyro, Motor)
  - Code der einzelnen Komponenten zusammenführen