



Spielereien mit Mikrocontrollern

HeLa Projektwoche 2019
J. Geisler und K. Gojdka

Ablauf



Tag 1

- Grundlagen und Beispiele

Tag 2

- Projektidee, Vertiefung und Ausarbeitung

Tag 3

- Ausarbeitung

Tag 4

- Projektvorstellung

Tag 1



- Theorie: Mikrocontroller
- Praxis: „Let it Blink“
- Theorie: C Programmierung
- Praxis: ASCII Tabelle
- Theorie: Breadboard und Pin-Funktionen
- Praxis: Buttons und LEDs
- Theorie: Mehr Befehle und Programmiertechnik

Wer nicht fragt
bleibt dumm...



Theorie: Mikrocontroller

A yellow pencil and a pink eraser are positioned in the top right corner of the slide, appearing to be on a piece of paper.

- Kleine Rechner, in vielen Geräten
 - Prozessor (CPU)
 - Arbeitsspeicher (RAM)
 - Dauerspeicher (EEPROM oder Flash)
 - Peripherie (eigenständige Spezialfunktionen)
 - Beinchen / Pins zur Ein- und Ausgabe
-
- Kein Betriebssystem = Volle Kontrolle und sehr schnell
 - Aber: alles muss selber gemacht werden

Theorie: Der Prozessor



- Die Recheneinheit
 - Kann nur Basisrechenarten
 - Nur Eins nach dem Anderen
 - Nur 16bit-Zahlen
(Werte von 0 bis 65536 bzw. -32768 bis 32767)
 - Zwischenergebnisse können im RAM gespeichert werden
- Die Steuereinheit
 - Sagt der Recheneinheit, was sie tun soll
 - Arbeitet ein „Kochrezept“ ab (das Programm)
 - Streng nach Reihenfolge
 - Das Programm liegt im Flash

Theorie: Die Peripherie



- Timer
 - Zähler mit präzisiertem Takt
- Analog-Digital-Konverter
 - Spannung in 10bit-Zahl umwandeln
- Serielle Kommunikation
 - UART: die gute alte serielle Schnittstelle
 - SPI, I2C: Kommunikation mit anderen Chips
 - IrDA: Code von Infrarot Fernbedienungen
- Capacitive-Touch
 - Berührungslose Schalter oder Näherungssensoren

Theorie: Die Beinchen



- Die meisten Beinchen können verschiedene Funktionen haben
 - Digitaler Eingang
 - Analoger Eingang
 - Digitaler Ausgang
 - (Quasi)-Analoger Ausgang
 - Kommunikationsschnittstelle
- Einige Beinchen sind schon auf dem LaunchPad beschaltet
- **Achtung: durch falsche Beschaltung können Beinchen zerstört werden!**
 - **Kurzschluss oder externe Spannung: nie direkt mit VCC oder GND oder anderen Pins verbinden!**

Theorie: Die Beinchen



Energia

LaunchPad with MSP430G2553

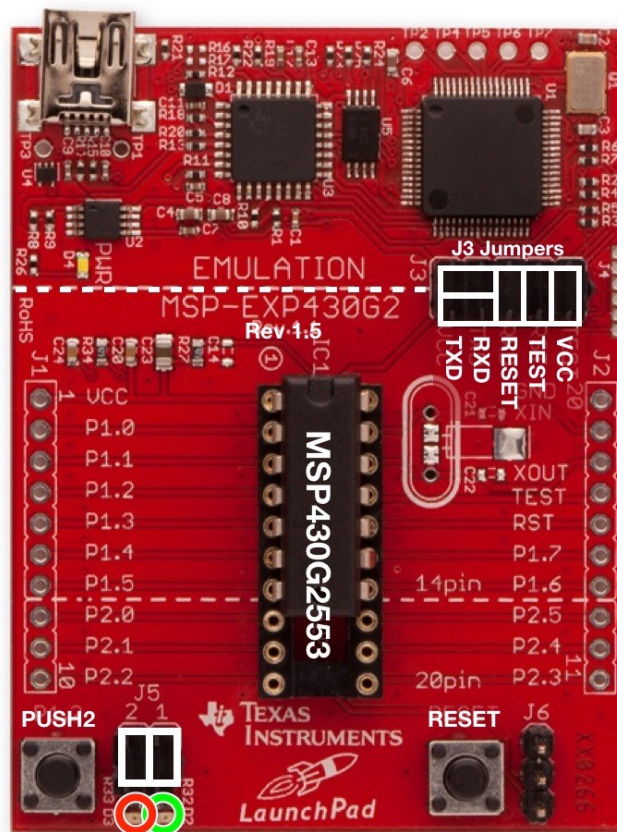
Revision 1.5

Flash 16 KB
RAM 512 B

Serial	Hardware
ADC	10 bits
Use pins numbers only!	
Default I ² C = (1)	
Software I ² C (1) master only	
PWM 4 or 14 or 19	
PWM 9 or 10	
PWM 12 or 13	

+3.3V				1
RED_LED		A0	P1_0	2
	RXD	A1	P1_1	3
	TXD	A2	P1_2	4
PUSH2		A3	P1_3	5
		A4	P1_4	6
	SCK (B0)	A5	P1_5	7
	CS (B0)		P2_0	8
	SCL (1)		P2_1	9
	SDA (1)		P2_2	10

temperature A10



Hardware
Pin number

I²C
Serial UART
SPI

analogRead()
digitalRead() and digitalWrite()
digitalRead(), digitalWrite()
and analogWrite()

20				GROUND
19	P2_6			XIN
18	P2_7			XOUT
17				TEST
16				RESET
15	P1_7	A7	SDA (0)	MOSI (B0)
14	P1_6	A6	SCL (0)	MISO (B0)
13	P2_5			GREEN_LED
12	P2_4			
11	P2_3			

GND
GND
+3.3V

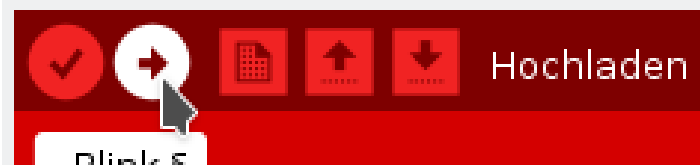


Rei Vilo, 2012-2018

embeddedcomputing.weebly.com

version 2.1 2015-09-13

Praxis: „Let it Blink“

- Energia starten
- LaunchPad per USB mit Rechner verbinden
- Menü Werkzeuge: Board
MSP-EXP430G2 w/ MSP430G2553 wählen
- Menü Werkzeuge: Port > COM6 wählen
- Menü Datei: Beispiele > 01.Basics > Blink
wählen
- Button Hochladen 
- Spielt mit den Werten von `delay(1000);`

Achtung Achtung



Theorie: C Programmierung



- C ist eine Programmiersprache
- Sie ist sehr streng
(Zeichensetzung, Groß-/Kleinschreibung)
- Aber gut verständlich
- Sie zeigt noch deutlich, was im Prozessor passiert
- Aber ermöglicht sehr komplexe Algorithmen
- Sie muss durch einen Compiler in Maschinencode umgewandelt werden
- Der Compiler gibt Fehlermeldungen, wenn das Programm nicht den Regeln entspricht
- Aber ein Programm das den Regeln entspricht kann trotzdem falsch sein bzw. Falsches tun!

Theorie: C Programmierung



- Ordnung muss sein
 - Jeder Befehl wird mit einem ; abgeschlossen
 - Gruppen von Befehlen werden in { ... } eingeschlossen und heißen Blöcke
 - Blöcke können in Blöcken geschachtelt sein, wie eine Matroschka Puppe
 - Kommentare helfen, den Code zu verstehen
 - `//` gilt bis zum Ende der Zeile
 - `/*` ein Block, der erst hier endet `*/`
- Berechnungen werden in Variablen gespeichert
 - Variablen haben einen Datentyp und müssen deklariert werden
 - `int i;` ist eine ganze Zahl
 - `float f;` ist eine Komma-Zahl,
sie braucht sehr viel mehr Rechenzeit
 - Variablen können (fast) beliebige Namen haben
 - Variablen gelten in dem Block, in dem sie deklariert sind und in allen verschachtelten Blöcken

Theorie: C Programmierung



- Variablen werden Werte mit = zugewiesen
 - Das können Zahlen sein, z.B. `i = 13;`
 - Oder Berechnungen, z.B. `i = 3 * 6;`
 - Es gibt die Grundrechenarten + - * / und Klammern
 - Oder das Ergebnis von Funktionen, z.B. `i = analogRead(A0);`
 - Oder eine Mischung aus beidem:
`i = 13 * max(0, k);`
 - ... und natürlich können Variablen auch Teil der Berechnung sein

Theorie: C Programmierung



- Funktionen helfen, komplizierte Berechnungen zusammenzufassen (zu Kapseln)
 - Funktionen haben immer einen Namen und ()
 - In den Klammern können Eingabewerte stehen
 - Mehrere Werte werden mit Komma getrennt
 - Wie viele Eingabewerte es gibt und welchen Datentyp sie haben hängt von der Funktion ab
 - Es gibt viele fertige Funktionen
 - Funktionen, die Eingaben in Ausgaben umwandeln
 - Funktionen, die was mit den Pins machen
 - Manche Funktionen haben keinen Rückgabewert

Theorie: C Programmierung



- Kontrollstrukturen helfen, das Programm flexibel und lebendig zu machen
 - Tue etwas, aber nur wenn ...
 - `if` (Bedingung)
 { tue_was }
 - `else`
 { sonst_tue_dies }
 - Bedingung ist ein logischer Ausdruck
 - Z.B. der Vergleich von Werten:
 `i<0`, `i>0`, `i<=0`, `i>=0`, `i!=0`
 - Oder die Verknüpfung von Bedingungen
 - `(i>0) && (i<2)`: und-Verknüpfung
 - `(i>0) || (i<2)`: oder-Verknüpfung
 - `!(i<2)`: Negierung („ist nicht“)

Theorie: C Programmierung



- Schleifen machen Wiederholung einfach
 - Tue etwas x-mal
 - `for (int i= 0; i<n; i++) { tue_dies }`
 - `int i= 0;` initialisiert die Laufvariable, es wäre auch z.B. `int i= 2*n;` denkbar
 - `i<n;` ist die Abbruchbedingung. Sie wird auch gleich zu Beginn überprüft. Bei `i!=i;` würde die Schleife nie ausgeführt
 - `i++` ist der Schleifenbefehl. Er wird am Ende jedes Durchgangs ausgeführt. Möglich wäre z.B. auch `i--` oder `i= i*2`
 - Tue etwas bis ...
 - `while(Bedingung) { tue_dies }`
 - Bedingung ist wie bei `if()`
 - `while(true) { }` hört nie auf und tut nichts. Es hält das Programm also effektiv an.

Funktionen in jedem Programm




- `void setup() { dein_code }`
 - Wird einmal ganz am Anfang ausgeführt
 - Streng genommen ist das einfach die Definition einer Funktion und `void` ist der Rückgabedatentyp, aber in Energia hat es eine besondere Bedeutung
- `void loop() { dein_code }`
 - Wird nach `setup` immer wieder ausgeführt

Funktionen zur Kommunikation



- **Serial.begin(9600);**
 - Einrichten der seriellen Schnittstelle mit 9600 bit/s (baud)
 - Muss einmal in **setup** gemacht werden.
 - Die Serielle Schnittstelle wird vom LaunchPad über USB and den PC weitergeleitet
- **Serial.println("Dein text hier");**
 - Sendet einen Text mit Zeilenumbruch über die Schnittstelle
- **Serial.println(i, basis);**
 - Wandelt die Zahl *i* in Text um und sendet diesen über die Schnittstelle
 - Bei der Umwandlung wird das Zahlensystem *basis* angewendet, es kann **BIN**, **OCT** oder **DEC** sein
- **Serial.available()**
 - Abfrage, ob Zeichen empfangen wurden, z.B. in **if(Serial.available()) { c= Serial.read(); }**
- **char zeichen= Serial.read();**
 - Lesen eines Zeichens von der Schnittstelle. Char ist der Datentyp für Zeichen.
- **int i= Serial.parseInt();**
 - Lesen von Zeichen von der Schnittstelle und Umwandeln der Zeichen in eine ganze Zahl
- **delay(milli_seconds);**
 - Anhalten der Programmausführung für eine gegebene Anzahl Millisekunden
- Weitere Funktionen und deren Beschreibung findet ihr hier:
 - <https://www.arduino.cc/en/pmwiki.php?n=Reference/HomePage>
 - <https://fkainka.de/befehlsliste-arduino/>

Praxis: ASCII Tabelle

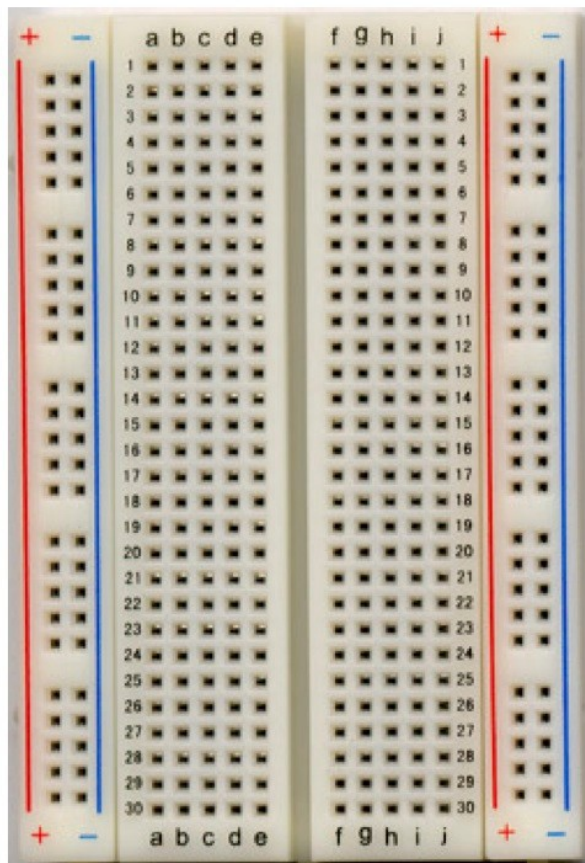
- Menü Datei: Beispiele > 04.Communication > ASCIITable wählen
- Button Hochladen 
- Menü Werkzeuge: Serieller Monitor
- Reset-Taste am LaunchPad drücken
- Versteht das Programm
- Macht ein paar lustige Änderungen

Theorie: Breadboard

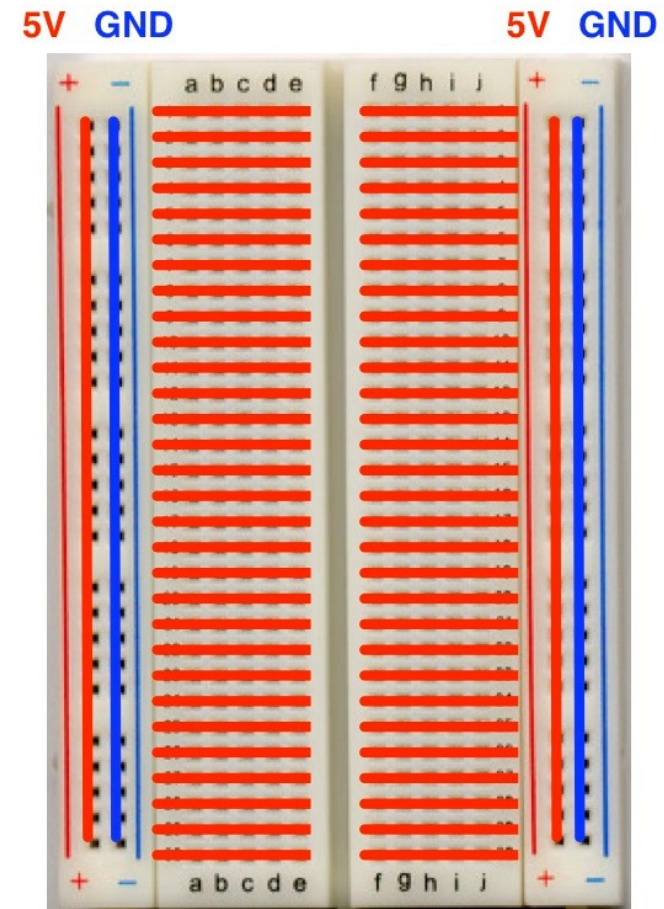
„+“, VCC, 3,3V, da kommt der Strom her

„-“, GND, Ground, 0V, da fließt der Strom hin

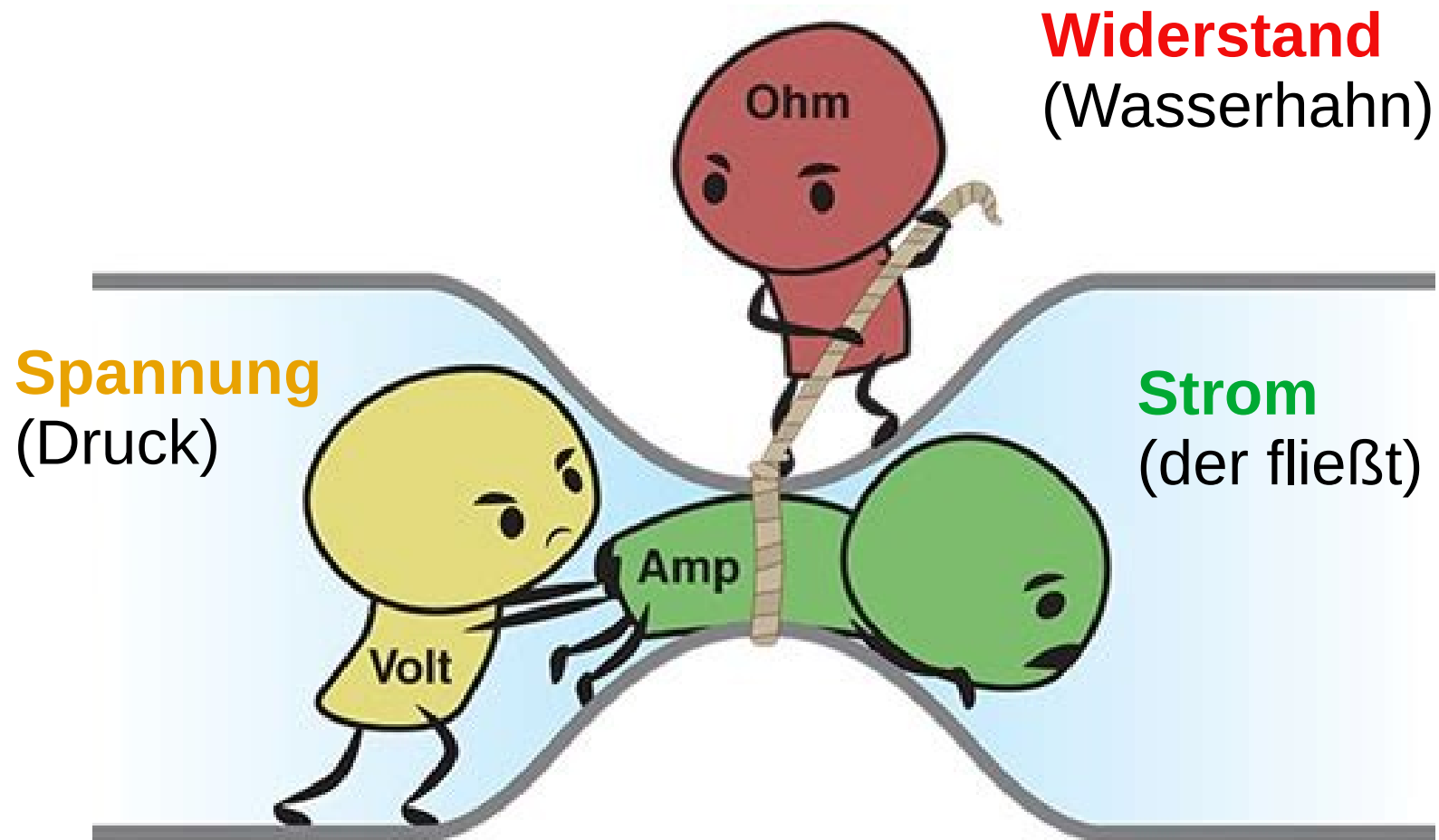
Breadboard (photo)



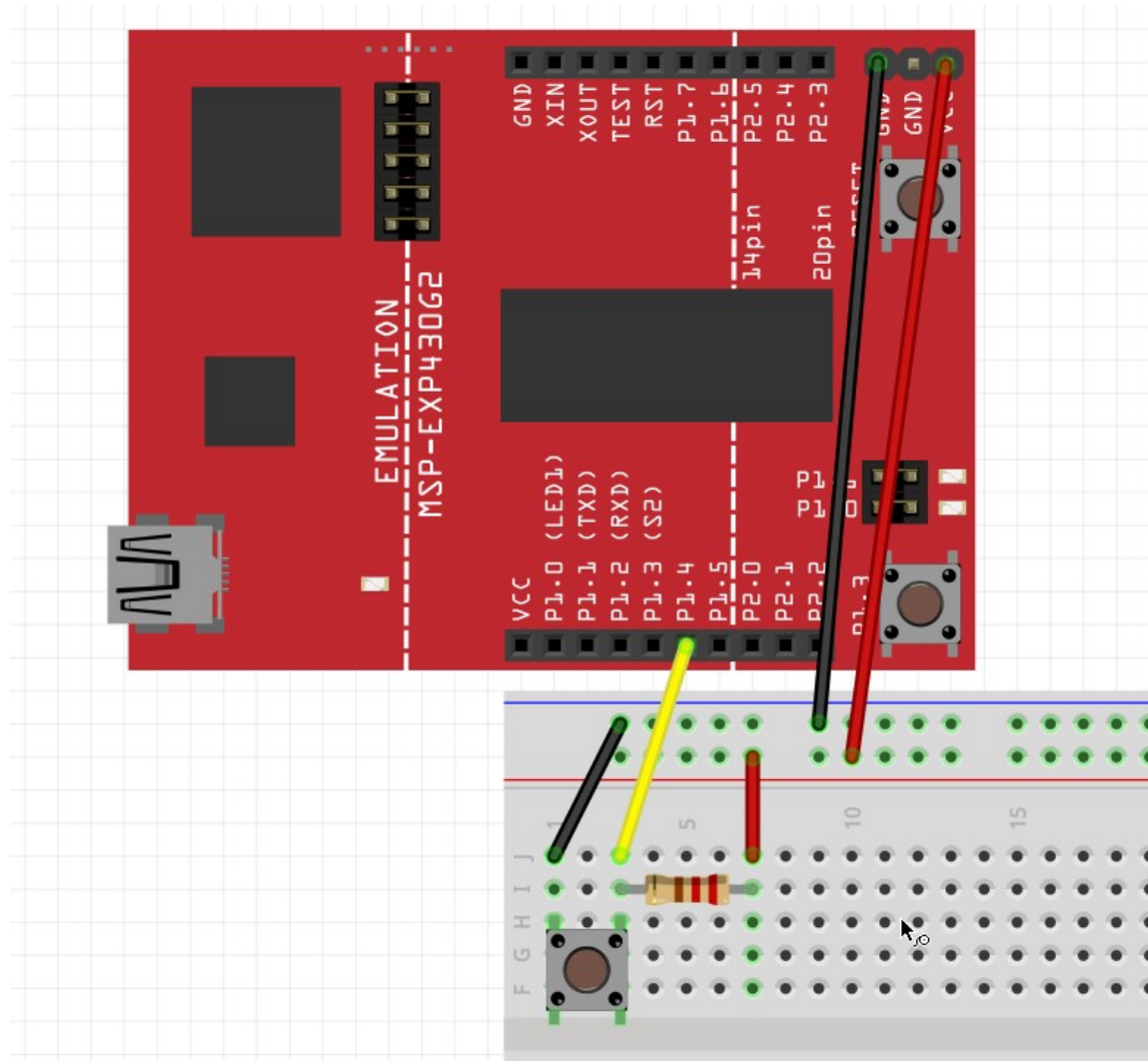
Breadboard (schematic)



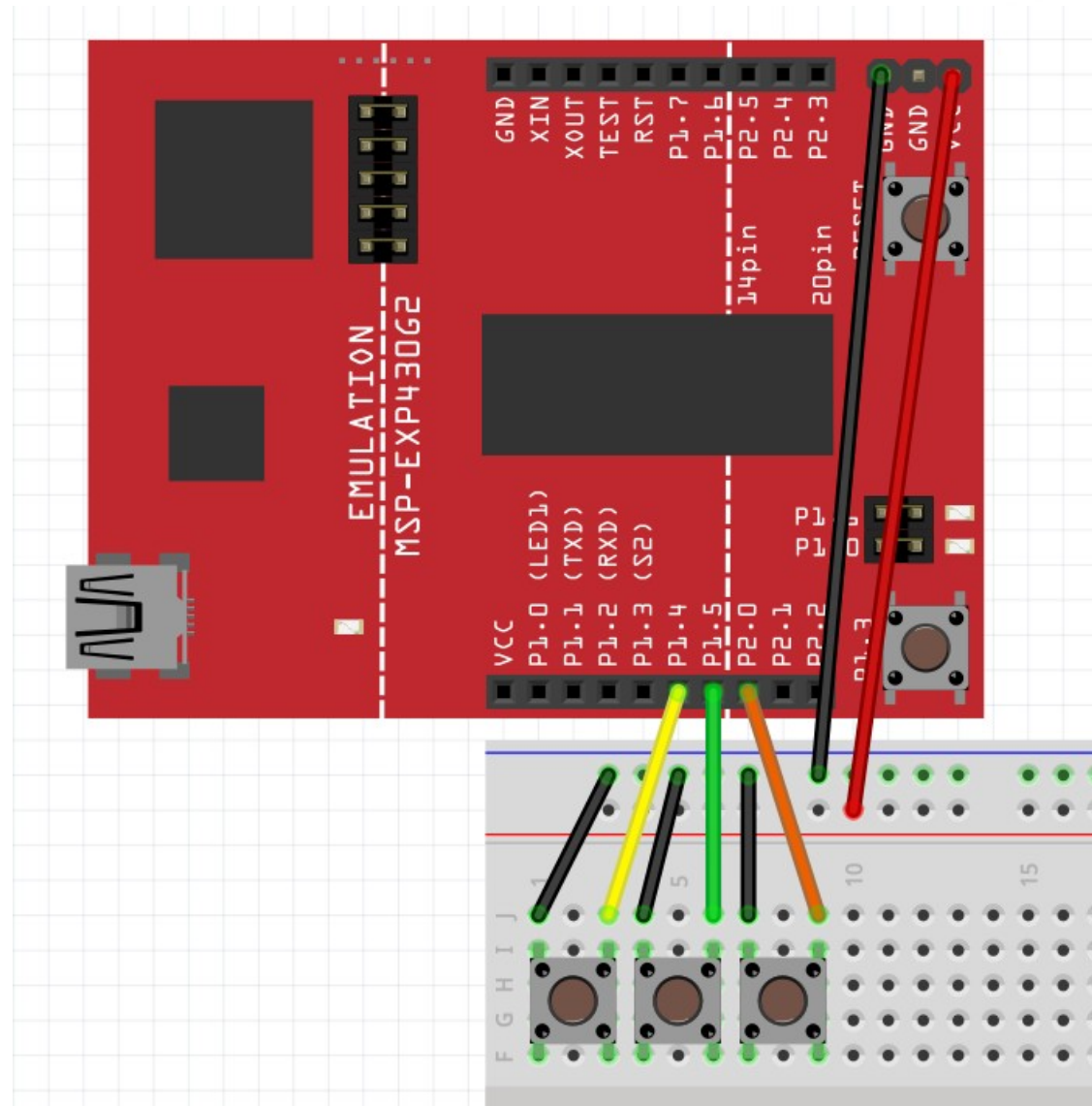
Theorie: Breadboard



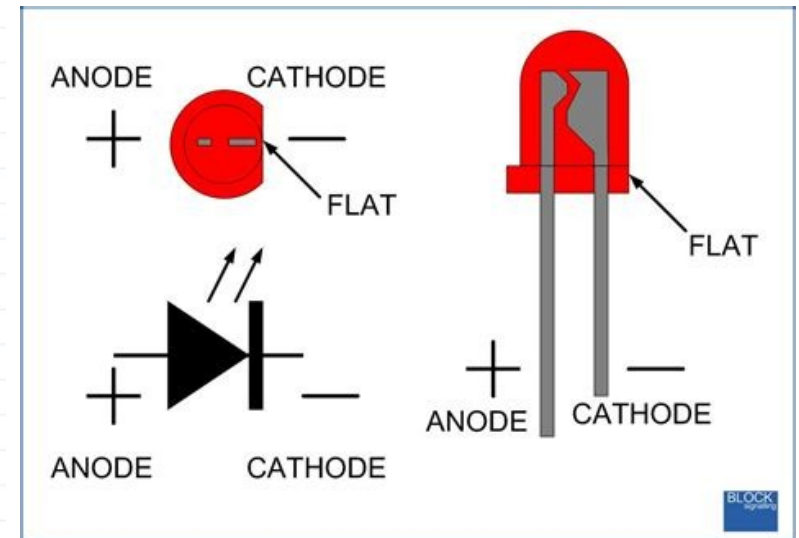
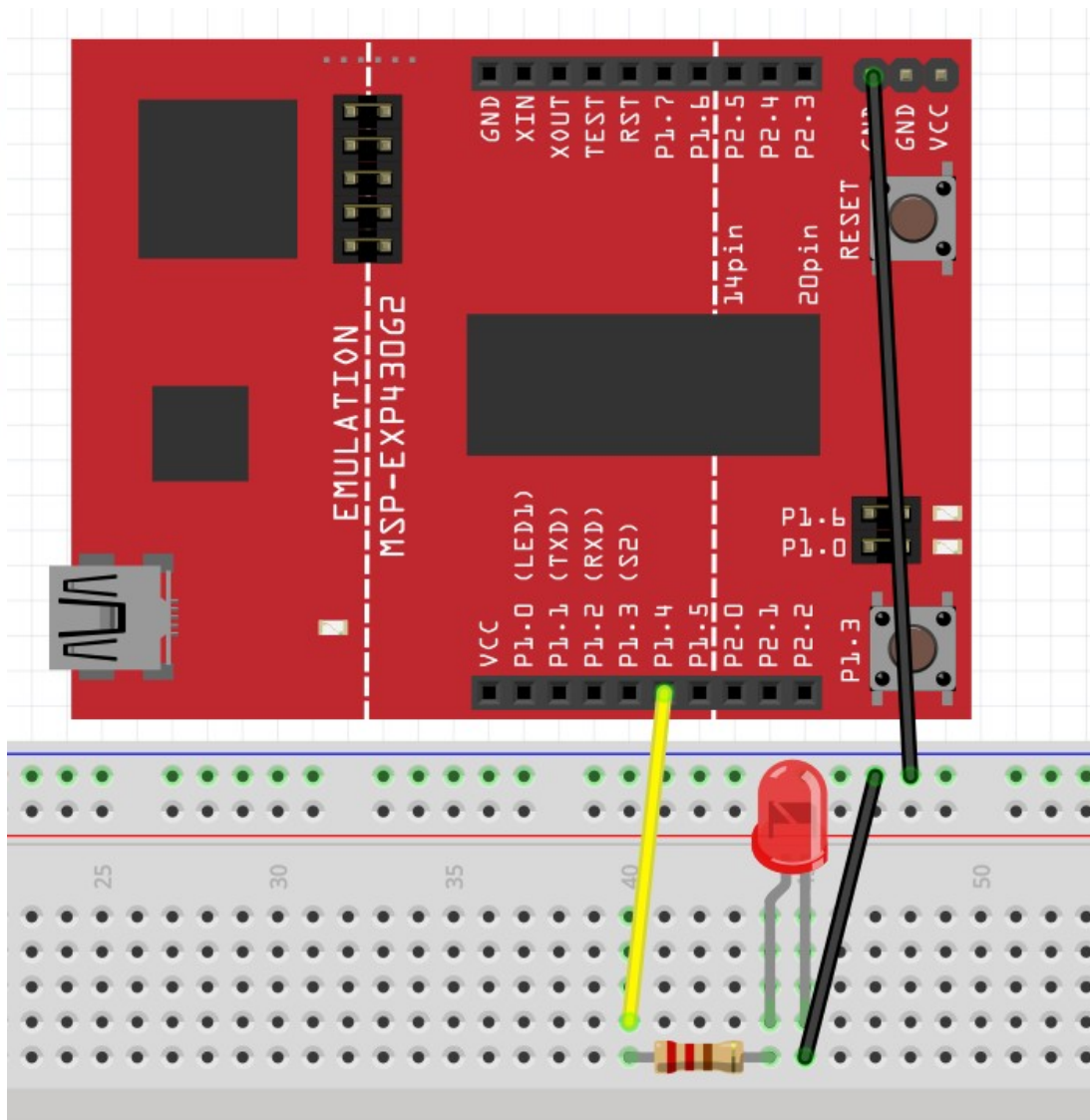
Theorie: Breadboard, Taster



Theorie: Breadboard, Taster



Theorie: Breadboard, LED



Achtung!
LEDs brauchen einen
Vorwiderstand (ca.
220Ω)

<https://www.elektronik-kompendium.de/sites/bau/1109111.htm>

Digitale Pin-Funktionen



- **pinMode**(pin, mode);
 - pin: z.B. **P1_0** (rote LED), **P1_6** (grüne LED), **P1_3** (Taster), siehe Übersichtsbild
 - mode: **INPUT**, **OUTPUT**, or **INPUT_PULLUP**
- **i = digitalRead**(pin);
 - Rückgabewert: **HIGH**, **LOW**
- **digitalWrite**(pin, value);
 - value: **HIGH**, **LOW**

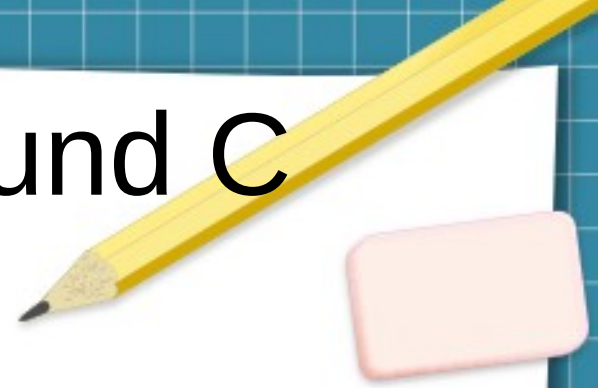
Praxis: Buttons und LEDs



- Verbindet zwei Taster und zwei LEDs mit dem LaunchPad
- Schreibt ein Programm, dass die LEDs bei gedrücktem Taster einschaltet
 - Ein Taster für die eine LED, der andere für die zweite
- Schreibt ein Programm, dass die LED beim ersten Drücken ein- und erst beim zweiten Drücken wieder ausschaltet

Theorie: Mehr Befehle und C

Analoge Pin-Funktionen



- `i = analogRead(pin);`
 - Wandelt eine Spannung zwischen 0 und 3,3V in eine Zahl zwischen 0 und 1023 um
- `analogWrite(pin, value);`
 - Schaltet `pin` schnell an und aus, wobei die Dauer der An-Zeit und der Aus-Zeit von `value` abhängt
 - Bei `value 0` ist der Pin fast nur aus
 - Bei `value 255` ist der Pin fast nur an
 - Bei `value 127` die halbe Zeit an und die halbe Zeit aus
 - Dadurch entsteht quasi eine variable Spannung zwischen 0 und 3,3V
 - Es lassen sich z.B. LEDs dimmen
 - Das nennt man Puls-Weiten-Modulation (PWM)

Theorie: Mehr Befehle und C

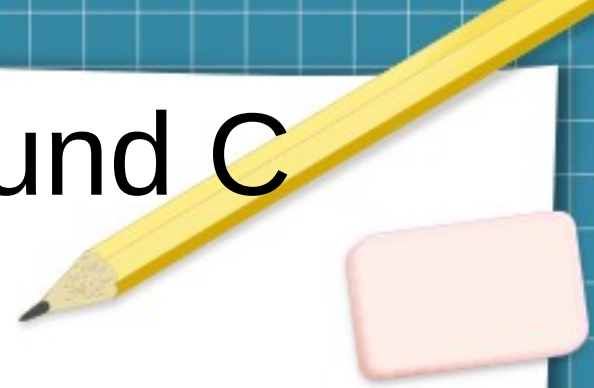
Zufallszahlen und Zeitmessung



- `int i= random(max)`
 - Gibt eine zufällige Zahl zwischen 0 und max-1 zurück
 - Es ist auch `random(min, max)` möglich
 - Achtung: ohne zufälligen Anfangswert sind die Zahlen immer gleich
- `randomSeed(seed)`
 - Stellt den Anfangswert für die Zufallszahlen ein
 - Auch der Anfangswert sollte zufällig sein, z.B. so `randomSeed(analogRead(0))`;
 - Oder über `millis()` beim Drücken eines Tasters
- `unsigned long m= millis()`;
 - Gibt die Zeit in Millisekunden seit Start des Controllers an

Theorie: Mehr Befehle und C

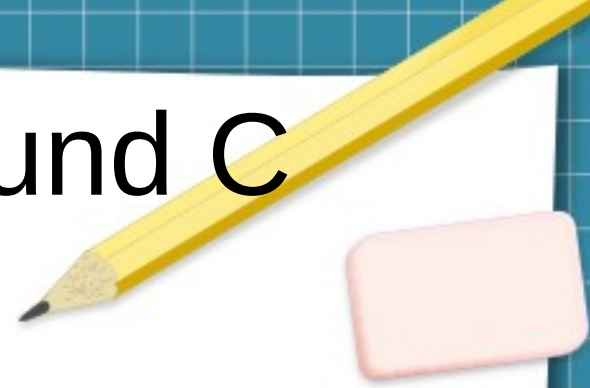
Töne (Piepser)



- **tone**(pin, frequency)
 - Schaltet pin mit der Frequenz frequency an und aus
 - Wenn ein Lautsprecher oder Kopfhörer an den Pin angeschlossen wird, kann man einen Ton hören
- **noTone**(pin)
 - Schaltet den Ton wieder aus

Theorie: Mehr Befehle und C

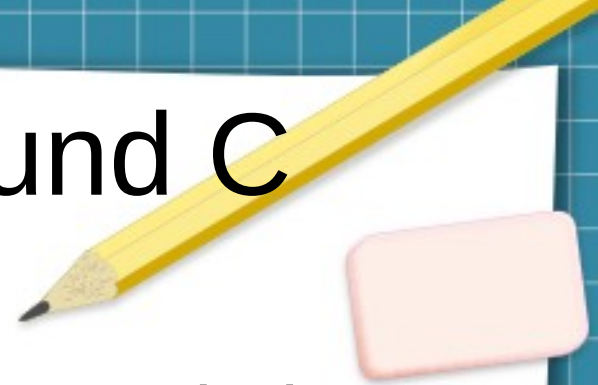
Eigene Funktionen



- Funktionen haben einen
 - Rückgabewert
 - Wird nur mit seinem Datentyp angegeben
 - Bei Funktionen ohne Rückgabewert schreibt man `void`
 - Zwischen Rückgabewert und Namen steht ein Leerzeichen
 - Was zurückgegeben wird, steht im Funktionskörper mit `return`
 - Einen Namen
 - Darf nicht mit einer Zahl beginnen und keine Leerzeichen enthalten
 - Übergabewerte
 - Stehen in Klammern
 - Werden mit Semikolon getrennt
 - Sehen aus wie Variablendeklarationen, also Datentyp Leerzeichen Name
 - Funktionen ohne Übergabewerte haben einfach nur leere Klammern: `()`
 - Einen Funktionskörper
 - Steht in geschweiften Klammern
- Beispiel:
 - ```
int meine_funktion(int i; int j)
{
 tu_was; i= i*j; return i;
}
```

# Theorie: Mehr Befehle und C

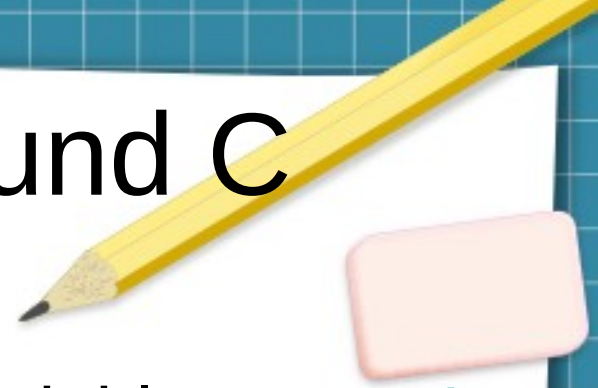
break, continue, return



- In Schleifen gibt es die Möglichkeit, die Bearbeitung des Schleifen-Blocks frühzeitig zu beenden
  - **break** beendet die Schleife sofort
  - **continue** springt ans Ende des Schleifen-Blocks und fährt mit der Bearbeitung der Schleife fort
  - Diese beiden Befehle werden typischer Weise bedingt in **if ( )** Anweisungen eingesetzt
- In Funktionen beendet **return** den Funktionsblock sofort

# Theorie: Mehr Befehle und C

## Konstanten



- Schreibt man vor die Deklaration einer Variablen `const` so kann der Wert nicht geändert werden:
  - `const int i= 13;`
- Zahlen in Hexadezimal (0 bis F)
  - `0xDEADBEEF`
- Zahlen in Binär (0, 1)
  - `B10101010`
- Oktal (0 bis 7)
  - `012`



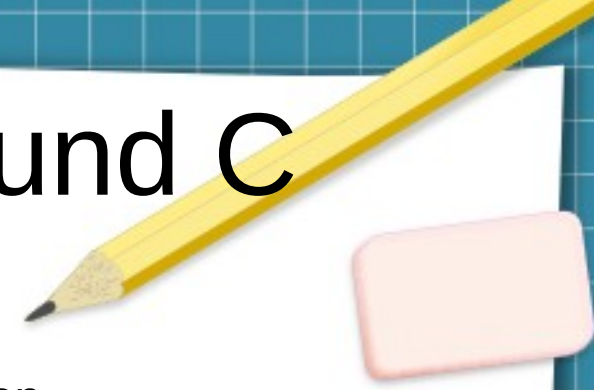
# Theorie: Mehr Befehle und C

Arrays: Ganz viele Zahlen in einer Variablen

- Will man viele Zahlen speichern, bieten sich sogenannte Arrays an:
  - `int mein_array[12];`
  - Deklariert ein Array mit 12 Plätzen
- Zugriff auf die Werte
  - `mein_array[0] = 23;`
  - `i = mein_array[11];`
  - `i = 3; i = mein_array[i];`
- Bei der Deklaration können auch schon Werte zugewiesen werden:
  - `int mein_array[] = {1, 2, 3, -1};`
  - Ist eine Array mit 4 Plätzen

# Theorie: Mehr Befehle und C

## 8x8 LED Matrix



- Zur Nutzung der LED Matrix muss eine Bibliothek installiert werden
  - Download („Clone or download“ Button, Download ZIP)  
<https://github.com/elpaso/ledcontrol-energia>
  - Enpacken und in „Dokumente\Energia\libraries\LedControl“ verschieben
  - Energia neu starten
- Nutzung
  - Global **LedControl** lc=**LedControl**(dataPin, clkPin, csPin);
    - dataPin: Pin, an dem „DIN“ hängt
    - clkPin: Pin, an dem „CLK“ hängt
    - csPin: Pin, an dem „CS“ hängt
  - In setup: lc.**init**(); lc.**shutdown**(0, **false**); lc.**setIntensity**(0,8);  
lc.**clearDisplay**(0);
  - In loop: z.B.
    - **setRow**(0, row, value); value ist eine 8-bit Wert, jedes Bit entspricht einer LED in der Zeile row
    - **setColumn**(0, col, value); value ist eine 8-bit Wert, jedes Bit entspricht einer LED in der Spalte col
    - **setLed**(0, row, col, state); state true schaltet LED in Zeile row/Spalte col an; false schaltet sie aus

# Theorie: Mehr Befehle und C

## CapTouch Berührungslose Sensoren

- Zur Nutzung der LED Matrix muss eine Bibliothek installiert werden
  - Download („Clone or download“ Button, Download ZIP)  
<https://github.com/elpaso/ledcontrol-energia>
  - Entpacken und in „Dokumente\Energia\libraries\LedControl“ verschieben
  - Energia neu starten
- Nutzung
  - Global **LedControl** lc=**LedControl**(dataPin, clkPin, csPin);
    - dataPin: Pin, an dem „DIN“ hängt
    - clkPin: Pin, an dem „CLK“ hängt
    - csPin: Pin, an dem „CS“ hängt
  - In setup: lc.**init**(); lc.**shutdown**(0, false); lc.**setIntensity**(0,8);  
lc.**clearDisplay**(0);
  - In loop: z.B.
    - **setRow**(0, row, value); value ist eine 8-bit Wert, jedes Bit entspricht einer LED in der Zeile row
    - **setColumn**(0, col, value); value ist eine 8-bit Wert, jedes Bit entspricht einer LED in der Spalte col
    - **setLed**(0, row, col, state); state true schaltet LED in Zeile row/Spalte col an; false schaltet sie aus

# Theorie: Mehr Befehle und C

## CapTouch Berührungslose Sensoren



- Zur Nutzung der CapTouch Funktion muss eine Bibliothek erstellt werden
  - Download („Download ZIP“) <https://gist.github.com/robertinant/2941071>
  - Entpacken und in „Dokumente\Energia\libraries\CapTouch“ verschieben
  - Energia neu starten
- Nutzung
  - Global `#include "CapTouch.h"`
  - In setup: `Touch.add(pin);`
    - `pin` ist der Pin, an dem ein langer Draht oder ein Draht mit einem Stück Alu-Folie hängt
  - In loop: z.B.
    - `Touch.measure(pin)` wobei `pin` der Pin ist, den man in setup ge-addet hat