Sketching with Hardware

06: Arduino 02

Einige Tipps...

Nicht blockierender Code

- Der Arduino kann nur einen Thread ausführen
- Parallelisierung von Tasks schwierig
- Die `delay()`-Funktion blockiert das gesamte Programm
- `delay()` kann vermieden werden, indem ein Timer benutzt wird

Einfacher Timer

```ino
int buttonPin = 5;
int ledPin = 13;
int ledState = LOW;

void loop() {
  // blink the LED
  if(ledState == LOW) {
    ledState = HIGH;
  }
  else {
    ledState = LOW;
  }

  // turn off LED if button is not pressed
  if(digitalRead(buttonPin) == LOW) {
    ledState = LOW;
  }

  digitalWrite(ledPin, ledState);
  delay(1000); // wait for a second
}
```

blockierend

```ino
// ...
long lastMillis = 0;

void loop() {
  // check time since last update
  if(millis() - lastMillis >= 1000) {
    lastMillis = millis();

    if(ledState == LOW) {
      ledState = HIGH;
    }
    else {
      ledState = LOW;
    }
  }

  // turn off LED if button is not pressed
  if(digitalRead(buttonPin) == LOW) {
    ledState = LOW;
  }

  digitalWrite(ledPin, ledState);
}
```

nicht blockierend

State Machine: Keksautomat

```
#define WAIT 1
#define ORDER 2
#define PAYMENT 3
#define DISPENSE 4

int state = WAIT;
int order;

// cookie dispenser functions
int getInput() { … }
int getOrder() { … }
void handlePayment(int product) { … }
void dispenseProduct(int product) { … }
```

```
void loop() {
  switch(state) {
    case WAIT:
      if(getInput() != 0) state = ORDER;
      else delay(1000);
      break;
    case ORDER:
      order = getOrder();
      state = PAYMENT;
      break;
    case PAYMENT:
      handlePayment(order);
      state = DISPENSE;
      break;
    case DISPENSE:
      dispenseProduct(order);
      state = WAIT;
      break;
}
```

Debug Levels

- Die Arduino IDE hat keinen eingebauten Debugger
- Serielle Ausgabe wird zum loggen des Programm-zustands verwendet
- Mithilfe von Debug Levels kann man schnell den Detailgrad der Ausgaben anpassen

```
#define NONE 0
#define ERROR 1
#define WARN 2
#define DEBUG 3
#define ALL 4
#define DEBUG_LEVEL WARN

void loop() {
  if(DEBUG_LEVEL >= WARN) {
    Serial.println("Warning!");
  }
}
```

Arduino-Libraries

- Erweitern die Funktionen des Arduino
- Arduino ist eine offene Plattform
- Kompatible Produkte von vielen Herstellern
- Libraries zur einfacheren Anbindung dieser Komponenten

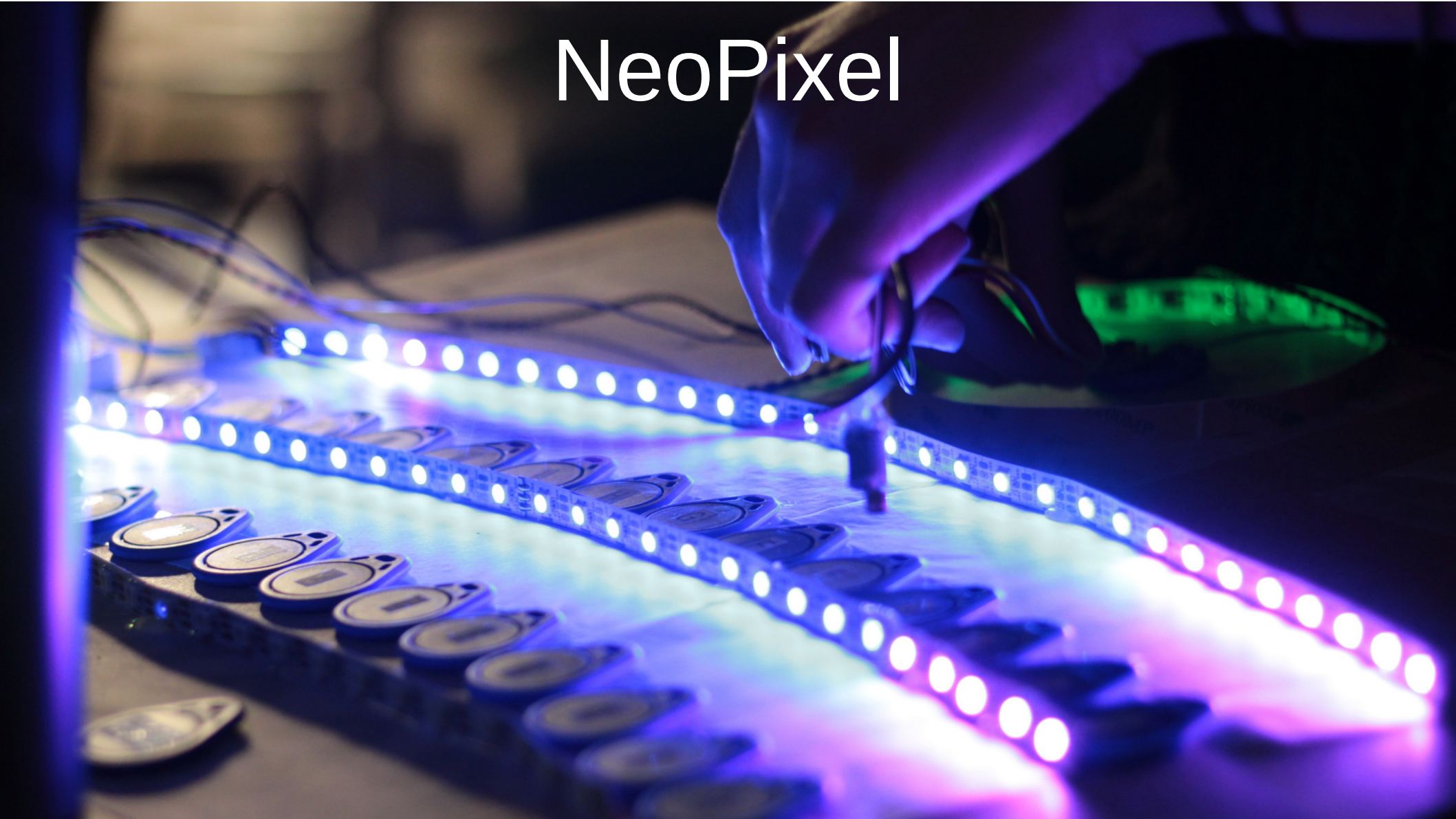
Beispiele für Libraries

- **Speicher** (EEPROM, SD)
- **Networking** (Ethernet, GSM, WiFi)
- **Protokolle** (SPI, SoftwareSerial)
- **Ausgabe** (LCD, TFT, Keyboard)
- **Signalverarbeitung** (debouncing, FFT)

Libraries installieren

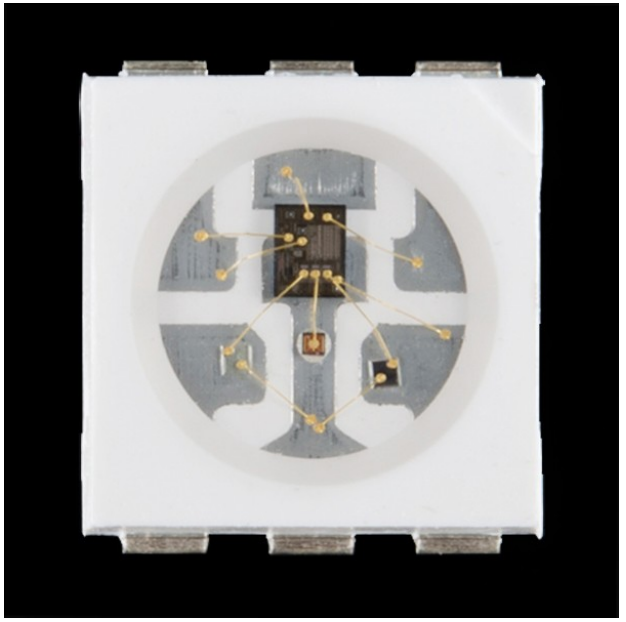
```
pio lib search “neopixel”

pio lib search “header:Adafruit_NeoPixel.h”

pio lib install 28
```

NeoPixel

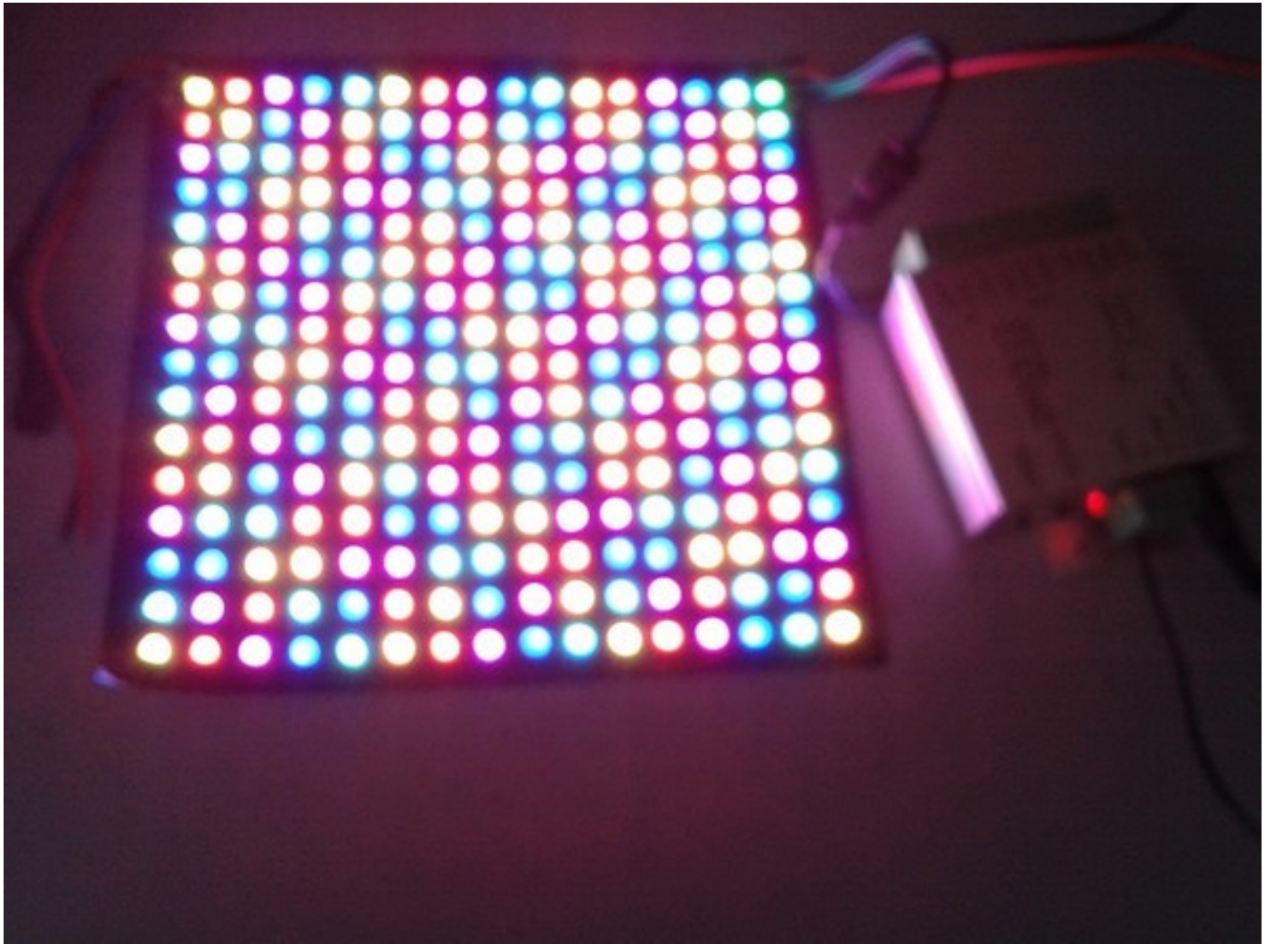


WS2812-LEDs

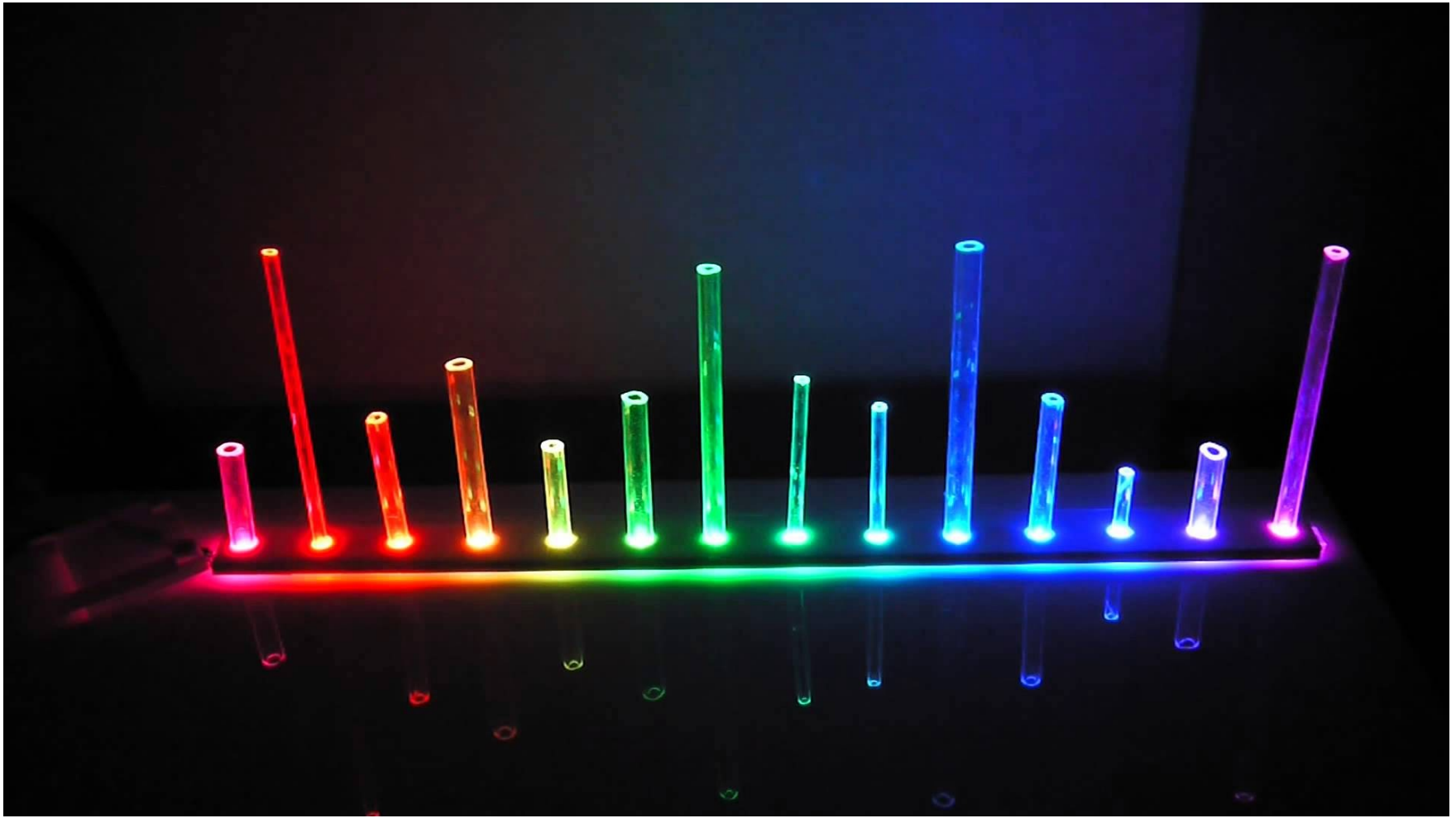
- Chip mit RBG-LEDs
- Können über serielles Protokoll angesteuert werden
- Einfach benutzbare Library für Arduino



https://cdn.sparkfun.com//assets/parts/8/0/3/7/11821-02.jpg



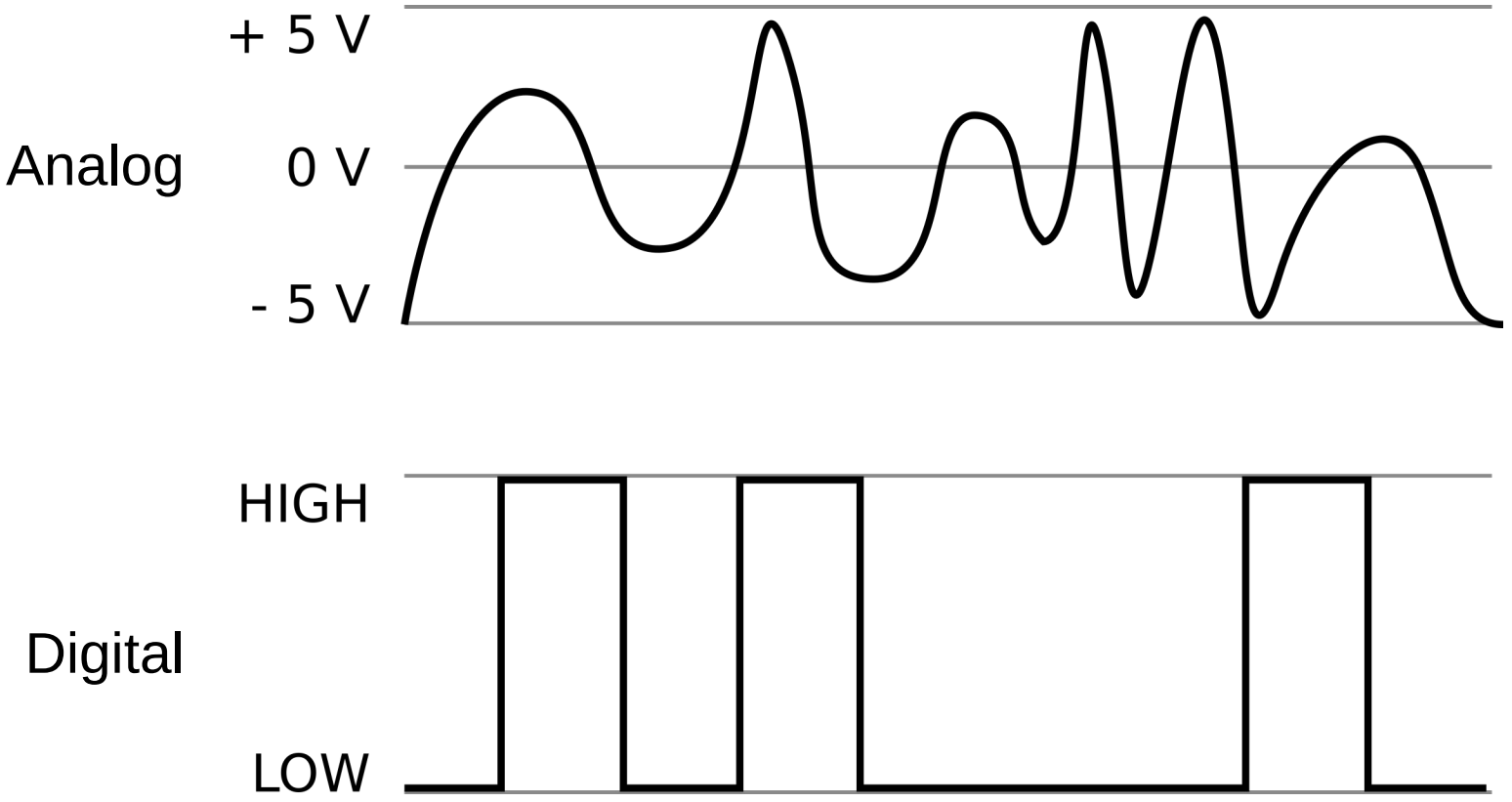
http://www.ledlite007.com/upfile/20131016104757492.jpg



https://i.ytimg.com/vi/aTfXeM4NJJs/maxresdefault.jpg

Digitalelektronik

Analog vs Digital



Vor- und Nachteile digitaler Signale

- Klar definiert, entweder es liegt Spannung an oder nicht
- Weniger anfällig für Störungen und Rauschen
- Kann einfach gespeichert und reproduziert werden

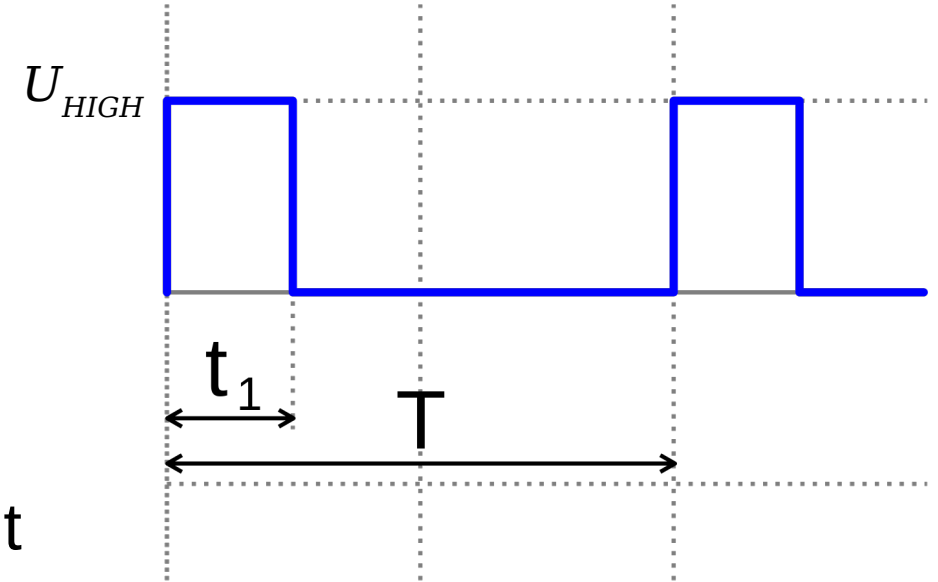
Aber:

- Weniger Information als analoges “Original”
- Digitale Komponenten sind komplexer

Pulsbreitenmodulation (PWM)

- Digitale “Simulation” eines analogen Signals

$$U = U_{HIGH} * \frac{t_1}{T}$$



- Signal ist für einen bestimmten Anteil (t_1) eines Zeitintervalls (T) *HIGH,* den Rest davon *LOW*
- Nettospannung über *T* entspricht der Spannung von *HIGH* mal dem zeitlichen Anteil des *HIGH*-Signals

Image: https://upload.wikimedia.org/wikipedia/commons/thumb/9/9b/Pulse_wide_wave.svg/1280px-Pulse_wide_wave.svg.png

Serielle Datenübertragung

- Der Arduino Micro hat zwei serielle Ports (**UART**):
  - **Serial** kommuniziert über USB mit dem Computer
  - **Serial1** verwendet die Pins *RX* (receive) und *TX* (transceive), um mit verbundenen Geräten zu kommunizieren
- Aktivität der seriellen Ports wird durch zwei LEDs (*RX* and *TX*) angezeit
- Die *baudrate* muss bei verbundenen Geräten übereinstimmen und vorher eingestellt werden
  - **Arduino:** `Serial.begin(`*`baudrate`*`);`
  - **platformio.ini:** `monitor_speed =` *`baudrate`*

Inter-Integrated Circuit (I²C)

- Bidirektionaler, serieller Datenbus nach dem *Master/Slave*-Prinzip
- Mehrere S*laves* können an einen *Master* angeschlossen werden
- Benötigt zwei Leitungen:
  - Serial Clock (SCL): Alterniert zwischen *HIGH* and *LOW* und bestimmt so die Übertragungsrate
  - Serial Data (SDA): Überträgt daten data (ein Bit pro Zyklus)
- Das erste übertragene Byte einer Nachricht bestimmt Richtung und Empfänger

Serial Peripheral Interface (SPI)

- Synchroner, serieller Datenbus nach dem *Master/Slave*-Prinzip
- Drei gemeinsame Leitungen:
  - SCLK (*Serial Clock*)
  - MISO (*Master in, Slave out*)
  - MOSI (*Master out, Slave in*)
- Eine *SS* (*Slave Select*) Leitung vom *Master* zu jedem *Slave*
- Sobald ein *Slave* vom *Master* über seinen *SS* angesprochen wird, “lauscht” er an *MOSI* und antwortet über *MISO*

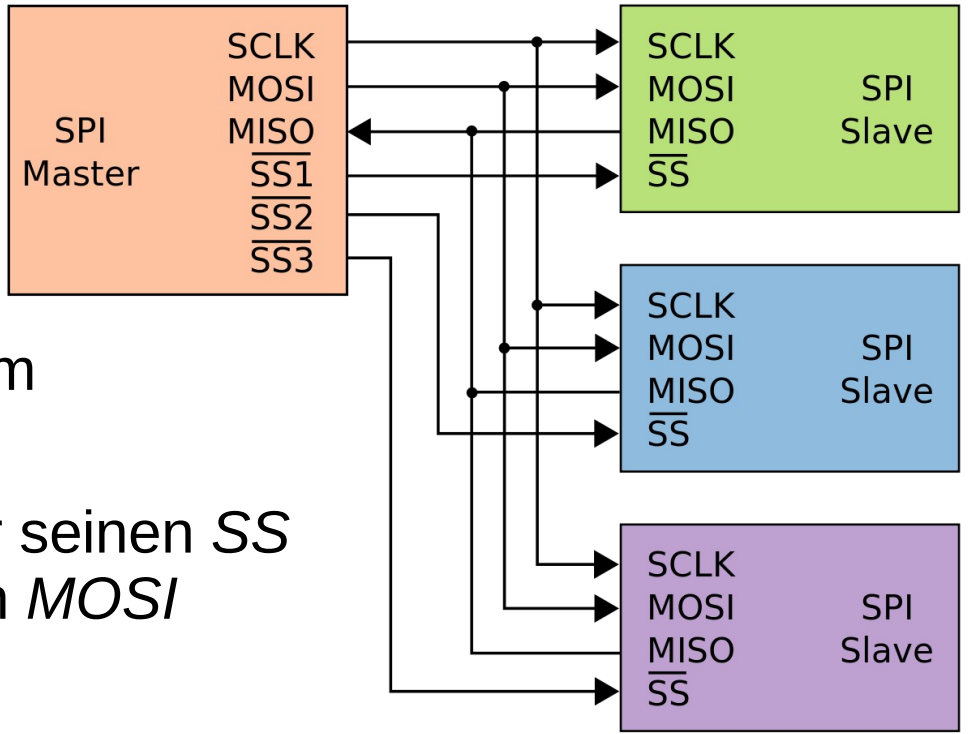


Image: https://de.wikipedia.org/wiki/Datei:SPI_three_slaves.svg

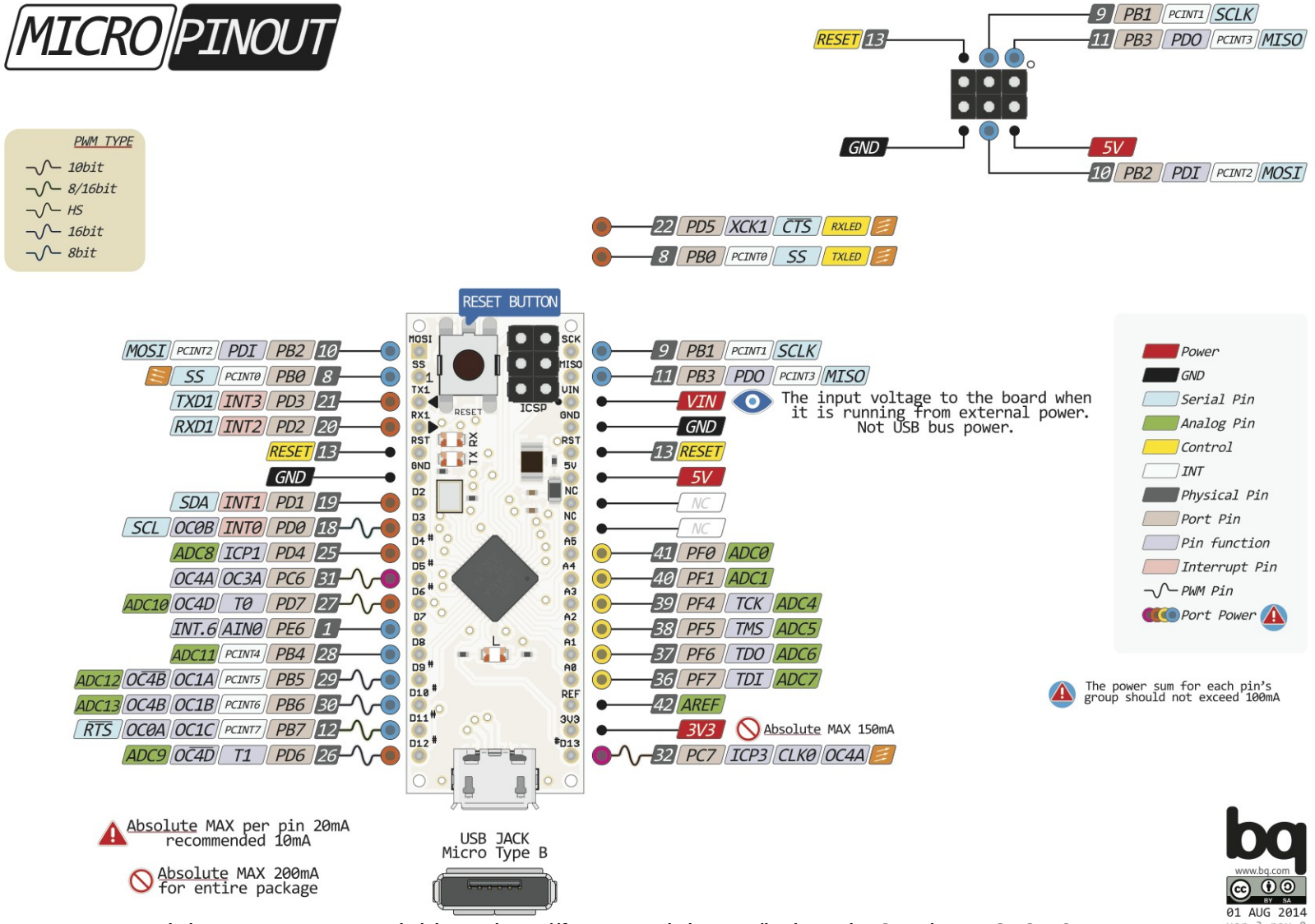


Image: Arduino Forum, User pighixxx, http://forum.arduino.cc/index.php?topic=148734.0