Workshop
Arduino-Programmierung #4

while, UART, Taster, Entprellen, long press, random

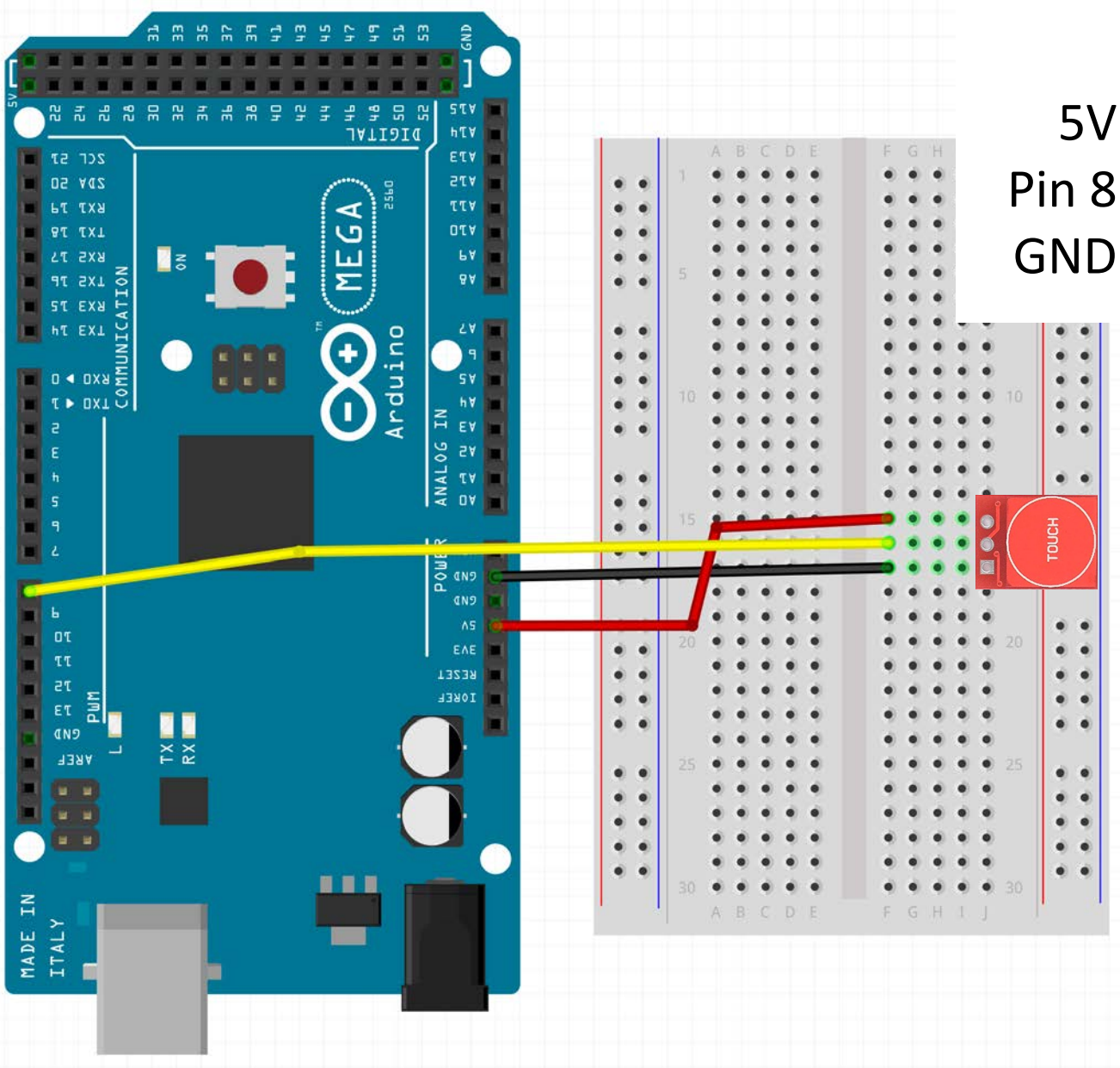
Joachim Baur

E-Mail: post@joachimbaur.de

ZTL-Alias: @joachimbaur

Download für diesen Workshop: www.joachimbaur.de/WS4.zip

Touchmodul auslesen



5V	VCC
Pin 8	I/O
GND	GND

10_Touch_Modul.ino

```
// Touch-Modul TTP223 auslesen

// Verbindungen:
// VCC -> Mega 5V
// I/O -> Mega Pin 8
// GND -> Mega GND

const int TOUCH_IO_PIN = 8;

void setup() {
    Serial.begin(115200);

    pinMode( TOUCH_IO_PIN, INPUT );
}

void loop() {
    int touchSignal = digitalRead( TOUCH_IO_PIN );
    Serial.println( touchSignal );
    delay(100); // Wert nur 10x pro Sekunde ausgeben...
}
```

While-Schleife

- zählt nicht die Schleifendurchgänge wie eine „for“-Schleife, sondern testet bei jedem Durchgang eine Bedingung (wie ein "if")

```
// entspricht „for (int i = 0; i < 10; i++) { … }“ Schleife:

int i = 0;

while ( i < 10 ) {
    // auszuführender Code-Block
    i++;
}
```

Serielle Eingabe empfangen

- `Serial.begin( baud )` wie beim Senden auch nötig
- `Serial.available()` Rückgabewert int: wieviele Zeichen sind im Empfangsspeicher?
- `Serial.read()` ein Byte aus dem Empfangsspeicher abrufen (Wert = 0…255)
- `char()` ein Byte in den Typ char („character“, = Buchstabe) umwandeln, char(65) = 'A'
- `String` Variablen-Typ für eine Zeichenkette (Text), zB `String anrede = "Hallo";`

11_Serial_read.ino

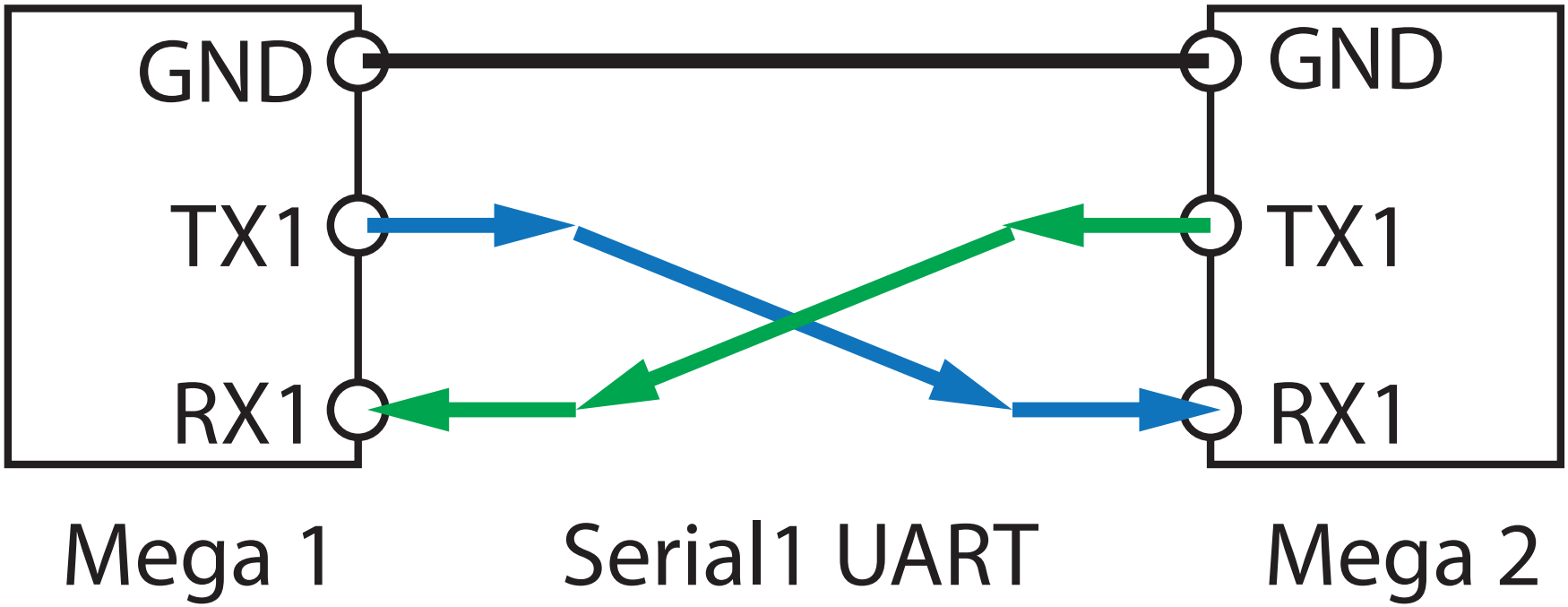
```
String eingabe = "";

void setup() {
    Serial.begin( 115200 );
}

void loop() {
    eingabe = ""; // leere Zeichenkette zu Beginn des Loops

    while (Serial.available() > 0) {
        // solange Bytes im Empfangsspeicher vorhanden sind
        eingabe += char( Serial.read() );
        // Byte auslesen, in Buchstaben umwandeln und an
        // "Eingabe"-Text anhängen
        delay(1); // kurz warten, ob noch Zeichen folgen
    }

    if (eingabe != "") {
        // wir haben etwas empfangen!
        Serial.print("ECHO: ");
        Serial.print(eingabe);
        // wieder in serielle Konsole ausgeben, println nicht nötig
        // weil CRLF schon am Ende von eingabe mit eingelesen
    }
}
```

UART-Verkabelung

- UART = "Universal Asynchronous Receiver Transmitter"
- Asynchronous = ohne zusätzliche Takt-Leitung, die Übertragung kann jederzeit beginnen
- Verkabelung: Mega 1 **RX1** <---> **TX1** Mega 2 und umgekehrt

  RX = Lese-Leitung (Receive)

  TX = Sende-Leitung (Transmit)



Serial1 bis Serial3

- Weitere Hardware-Serial-Ports (UART) beim Mega
- Gleiche Befehle, nur Serial1 statt Serial: **Serial1**.begin()

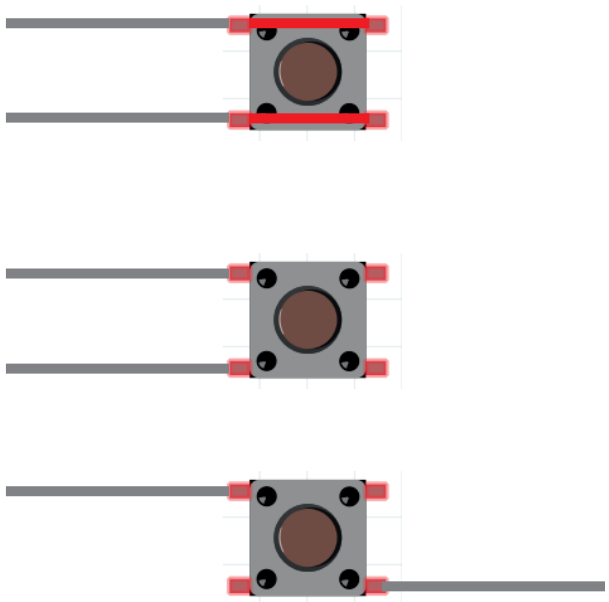
Aufgabe

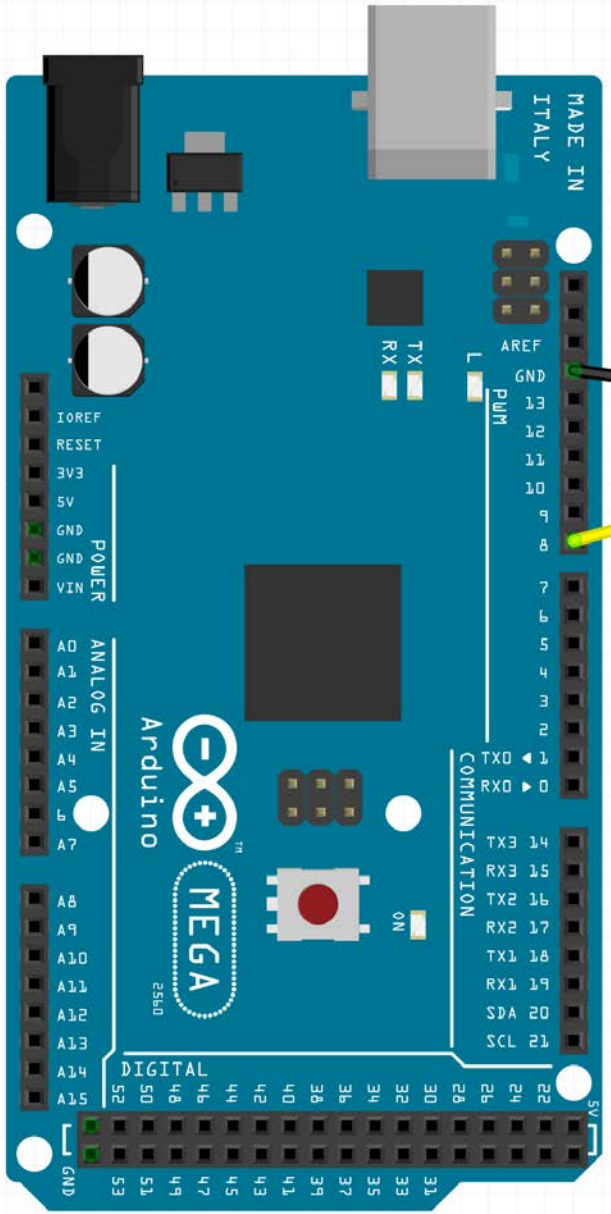
- 2 Megas verbinden jeweils über **Serial1** RX1/TX1
- RX1 mit TX1 verbinden (über Kreuz, direkte Kabel)
- **Wichtig: GND auch verbinden,** aber nicht 5V!
- Messages hin- und hersenden per Serial und Serial1 (Eingabe in serielle Konsole Gerät 1 -> Senden per Serial1 -> Anzeige in serieller Konsole auf Gerät 2)

12_Serial_MegaZuMega.ino

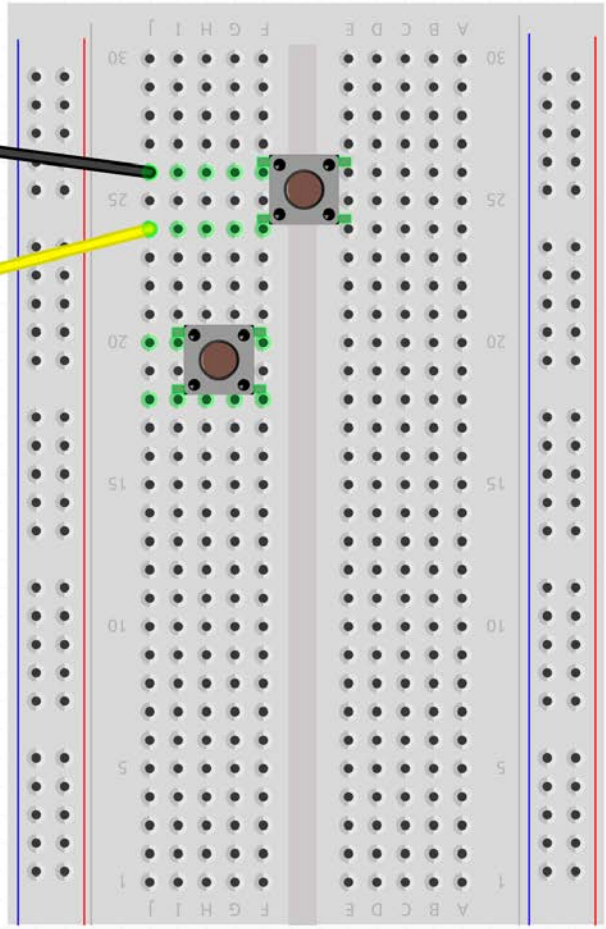
Taster auslesen

4 Pins – oben und unten zusammengeschaltet





Taster an PIN 8 und GND anschließen



Signal mit digitalRead() lesen

13_Taster_v1.ino

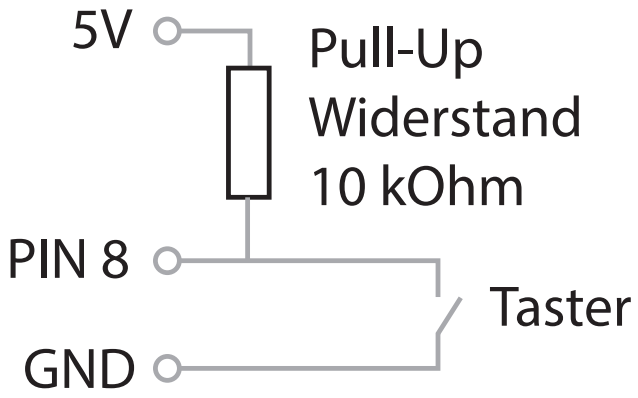
```
const int TASTER_PIN = 8;

void setup() {
    Serial.begin(115200);

    pinMode( TASTER_PIN, INPUT );
}

void loop() {
    int tasterSignal = digitalRead( TASTER_PIN );
    Serial.println( tasterSignal );
}
```

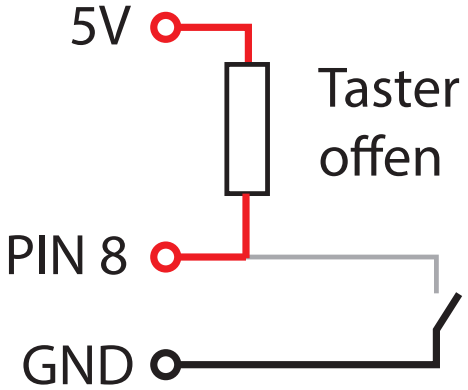
Problem (sichtbar in der seriellen Konsole):
Selbst wenn der Taster gar nicht betätigt wird, wird zufällig “0“ und „1“ von digitalRead() ausgegeben?

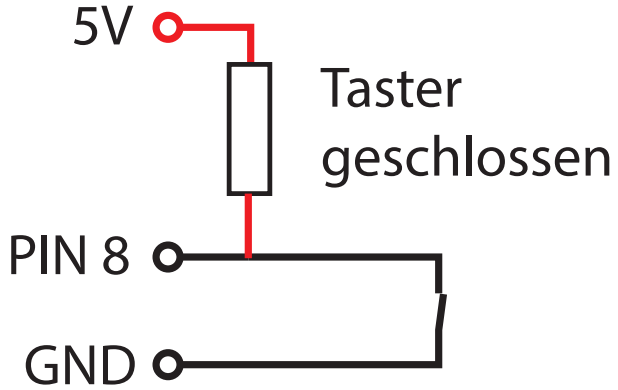
Ursache: „Offener“ Pin, wenn der Taster nicht gedrückt wird -> „float“ (induzierte Störungen)

Pull-Up und Pull-Down Widerstände

- Um einen offenen („floating“) Pin jederzeit in einem definierten Zustand zu halten, wird ein Pull-Up- oder Pull-Down-Widerstand verwendet
- Pull-Up: hält den offenen Pin auf +5V







INPUT_PULLUP

- Der Arduino-Chip hat für die GPIOs eingebaute Pull-Up-Widerstände, die per Code aktiviert werden können:

```
pinMode( TASTER_PIN, INPUT_PULLUP );
```

14_Taster_v2.ino

- Zudem wollen wir im Code nur dann reagieren, wenn sich der Taster-Status gerade geändert hat, also nur einmal, wenn der Taster gerade gedrückt oder losgelassen wurde.

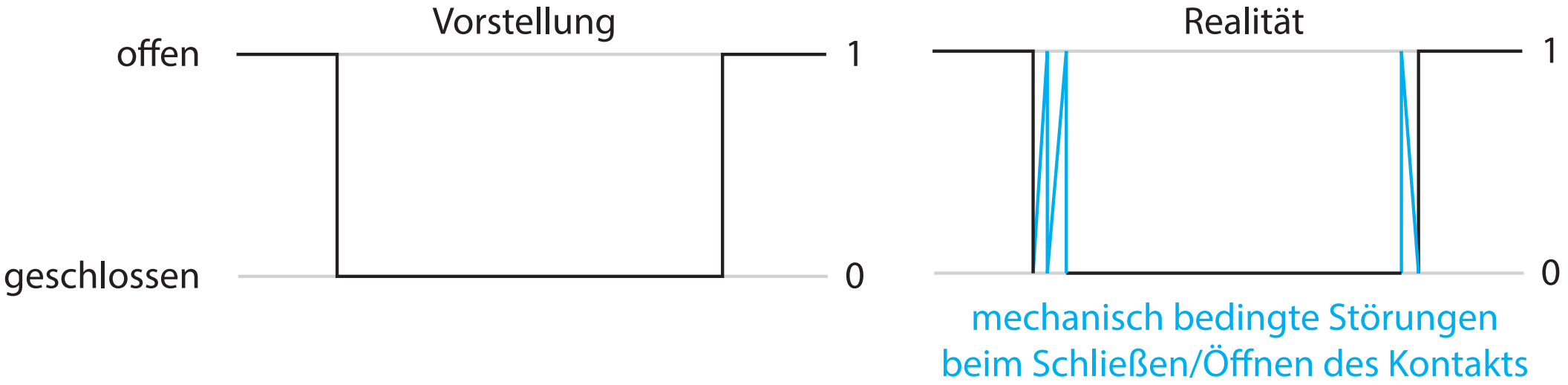
Tasterdruck erhöht zaehler-Variable

15_Taster_v3.ino

```
const int TASTER_PIN = 8;

int tasterStatus = HIGH; // wegen PULLUP
int zaehler = 0;

void setup() {
    Serial.begin(115200);

    pinMode( TASTER_PIN, INPUT_PULLUP );
}

void loop() {
    int tasterSignal = digitalRead( TASTER_PIN );
    if (tasterSignal != tasterStatus) {
        tasterStatus = tasterSignal; // merken
        // jetzt hat sich der Zustand geändert, also Aktion ausführen
        if (tasterStatus == LOW) {
            // Taster wurde gedrückt
            zaehler++;
            Serial.println(zaehler);
        }
    }
}
```

Tasterdruck entprellen

- Problem: manchmal erhöht sich die zaehler-Variable um mehr als 1 nach einem Tasterdruck?
- Grund: das elektrische Signal am Eingabepin ist während des Drückens nicht „sauber“, es „prellt“



- Lösung: Entprellen im Code: nach der 1. festgestellten Änderung das Signal für kurze Zeit (80-100 ms) ignorieren

Tasterdruck entprellen

16_Taster_v4.ino

```
if (tasterSignal != tasterStatus) {
   tasterStatus = tasterSignal;
   . . .
   delay(100); // entprellen
}
```

Statt "delay()" kann wieder eine "state machine" mit einer Statusvariablen "entprellenAktiv" verwendet werden

17_Taster_v5.ino

```ino
const int TASTER_PIN = 8;

int tasterStatus = HIGH; // wegen PULLUP
unsigned long entprellenEndeMillis = 0;
bool entprellenAktiv = false;
const int ENTPRELL_DAUER = 100; // Millisekunden

int zaehler = 0;

void setup() {
     Serial.begin(115200);

     pinMode( TASTER_PIN, INPUT_PULLUP );
}

void loop() {
     if (entprellenAktiv) {
          // entprellenAktiv == true: warten, Taster NICHT abfragen
          if (millis() > entprellenEndeMillis) {
               entprellenAktiv = false; // Wartezeit ist abgelaufen
          }
     } else {
          int tasterSignal = digitalRead( TASTER_PIN );
          if (tasterSignal != tasterStatus) {
               tasterStatus = tasterSignal; // merken
               // jetzt hat sich der Zustand geändert, also Aktion ausführen
               if (tasterStatus == LOW) {
                    // Taster wurde gedrückt
                    zaehler++;
                    Serial.println(zaehler);
               }
               // Entprell-Warteperiode aktivieren, sowohl für gedrückt als auch für losgelassen
               entprellenAktiv = true;
               entprellenEndeMillis = millis() + ENTPRELL_DAUER;
          }
     }
}
```

Taster long press

- Nachdem der Taster eine bestimmte Zeit lang gedrückt bleibt, soll eine weitere Aktion (aber nur 1x) ausgeführt werden
- Dazu muss zusätzlich zu der eigentlichen Taster-Zustandsabfrage in `loop()` ein Test stattfinden, ob
    - A) der Taster im Moment gedrückt ist
    - B) der Taster schon länger als x Millisekunden gedrückt ist

18_Taster_v6.ino

```
…
// nach 2 Sekunden soll die Meldung "Long Press!" ausgegeben werden
unsigned long gedruecktStartZeit = 0;
bool warteAufLongPress = true;
const int LONG_PRESS_DAUER = 2000; // Millisekunden

void loop() {
    if (entprellenAktiv) {
        …
    } else {
        int tasterSignal = digitalRead( TASTER_PIN );
        if (tasterSignal != tasterStatus) {
            tasterStatus = tasterSignal; // merken
            if (tasterStatus == LOW) {
                …
                // Long Press Zeitmessung vorbereiten
                warteAufLongPress = true; // es wurde noch kein LP
                registriert
                gedruecktStartZeit = millis();
            }
        }
    }

    if (tasterStatus == LOW) {
        // Taster ist gerade gedrückt
        if (warteAufLongPress) {
            if (millis() > (gedruecktStartZeit + LONG_PRESS_DAUER)) {
                Serial.println("LONG PRESS!");
                warteAufLongPress = false;
                // sonst wird die Meldung DAUERND ab 2 Sekunden ausgegeben
            }
        }
    }
}
```

Zufallszahlen

- `long random( int max )`

  `long random( int min, int max )`

  - min: inklusiv
  - max: exklusiv
  - Int zahl = random(1, 10); // -> Zufallszahlen von 1 bis 9
- `randomSeed( analogRead(A0) );   // in setup()`

- Anwendung: Würfeln – Drücken des Tasters startet das Würfeln (Ausgabe "Würfel rollt!" per Serial), bei Long Press wird eine Zufallszahl zwischen 1 und 6 ausgegeben ("Gewürfelt: x" per Serial)

19_Taster_Wuerfel.ino

```
void setup() {
    . . .
    randomSeed( analogRead(A0) );
}

void loop() {

    if (tasterStatus == LOW) {
        // Taster wurde gedrückt
        Serial.println("Würfel rollt!");
        . . .
    }

    if (millis() > (gedruecktStartZeit + LONG_PRESS_DAUER)) {
        //Serial.println("LONG PRESS!");
        int augenzahl = random(1, 7); // Zufallszahl von 1 bis 6
        Serial.println("Gewürfelt: " + String(augenzahl));
        . . .
    }

}
```