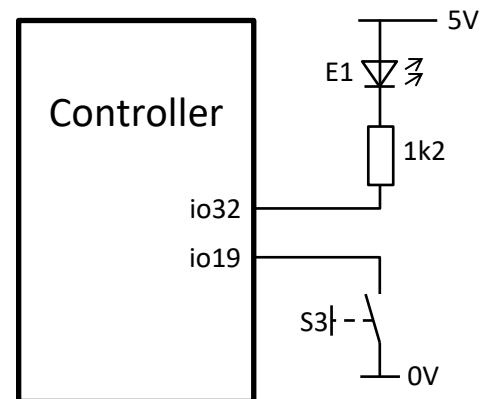
 Friedrich-Ebert-Schule Esslingen FES	IT: Hardwarenahes Programmieren	Name: Rahm Datum: 17.10.2022 1_5_Tastendruck_auswerten.docx
	Programmierung: Schalten mit Taster	1.5.1

Projekt: Schalten einer Lampe mit Taster

Oftmals müssen Aktionen mit einem Taster dauerhaft gesetzt werden (z.B. Lampe ein/aus, Motor ein/aus, ...). Dazu muss der gewünschte Zustand gespeichert werden, da der Taster nach dem Loslassen wieder öffnet.

Im abgebildeten Programmlisting wird der Zustand zunächst noch nicht gespeichert, es soll jedoch als Ausgangspunkt für das Projekt dienen.



```
const int LED = 32;
const int tasterS3 = 19;

bool tasterStatus;          //Statusvariable

void setup()
{
    // Hier Initialisierung ergänzen
}

void loop()
{
    tasterStatus = digitalRead(tasterS3);
    digitalWrite(LED, tasterStatus);
}
```

Arbeitsauftrag

1. Erstellen Sie das abgebildete Programm.
2. Ergänzen Sie die Initialisierung von Taster und Lampe (LED) und testen Sie die Funktion.
3. Überlegen Sie, warum die Lampe (LED) nach dem Loslassen des Tasters ausgeht.
4. Dokumentieren Sie Ihre Ergebnisse im Laborprotokoll.

Die Lampe darf nur beim Herunterdrücken des Tasters (= fallende Flanke) umschalten. Um dies zu erkennen, werden zwei Statusvariablen (`oldStatus` und `newStatus`) benötigt. Das Verhalten der Statusvariablen ist im Zeitdiagramm dargestellt:

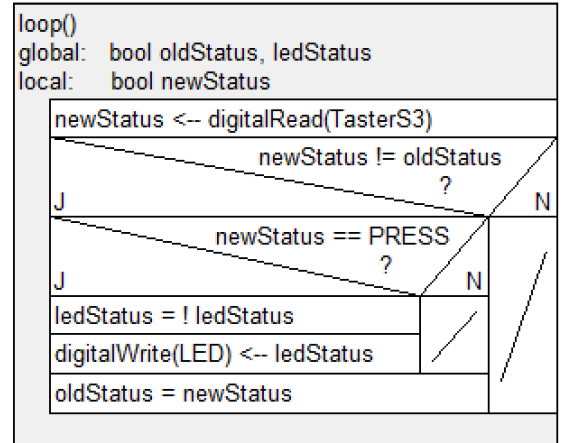
oldStatus	1	1	1	1	1	0	0	1	1	1	1	0	0	1	1
newStatus	1	1	1	1	0	0	1	1	1	1	0	0	1	1	1
Taster	released				pressed		released				pressed		released		
LED	LED OFF				LED ON						LED OFF				

Die Variable **oldStatus** muss global deklariert sein, weil sie den Wert des Tasterzustands auch nach Verlassen von `loop()` behalten muss. Der Datentyp `bool` (`pressed`, `released`) ist hier ausreichend. Die Variable **newStatus** kann auch lokal in `loop()` deklariert werden. Bei jedem Aufruf von `loop()` wird zuerst der Zustand des Tasters abgefragt und in **newStatus** gespeichert.

Der anschließende Vergleich **newStatus \neq oldStatus** ergibt **J** nur bei einer Flanke des Tastersignals (siehe Zeitdiagramm).

Um die fallende von der steigenden Flanke zu unterscheiden, wird nun der **newStatus** des Tasters auf `pressed (= 0)` abgefragt.

Anschließend kann der Zustand der LED invertiert ausgegeben werden. Schließlich muss noch der `oldStatus` nach jeder Änderung des `newStatus` upgedated werden.



```

const int LED = 32;
const int tasterS3 = 19;


#define PRESS LOW // Symbol für gedrückten Taster
// Null-Aktiv

bool oldStatus; // alter Tasterstatus
bool ledStatus;

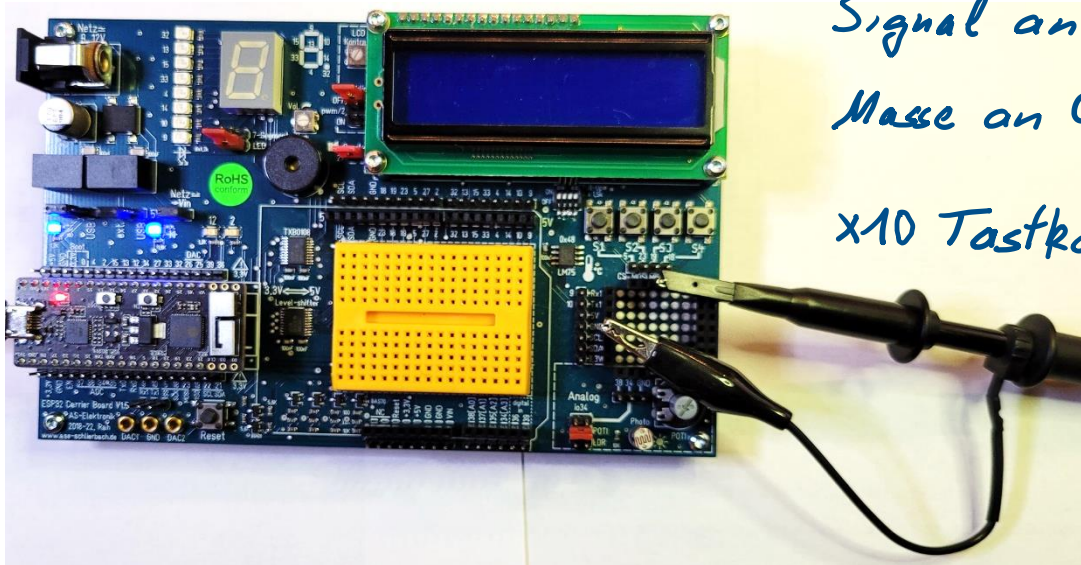
void setup()
{
  pinMode(LED, OUTPUT);
  digitalWrite(LED, HIGH); // LED aus
  ledStatus = true;
  pinMode(tasterS3, INPUT_PULLUP);
}
  
```

Arbeitsauftrag

1. Erstellen Sie die dargestellten Initialisierungen und die Funktion **setup()**.
2. Codieren Sie den im Struktogramm dargestellten Programmalgorithmus in der Funktion **loop()**.
3. Testen Sie das Programm und dokumentieren Sie Ihre Ergebnisse.
4. Sie werden vermutlich feststellen, dass manche Tastendrucke scheinbar nicht erkannt werden, oder manchmal auch beim loslassen geschaltet wird. Dies könnte daran liegen, dass je Tastendruck manchmal mehr Impulse auftreten (Tasterprellen). Untersuchen Sie dieses Verhalten mit dem Oszilloskop.

	IT: Hardwarenahes Programmieren	Name: Rahm Datum: 17.10.2022 1.5_Tastendruck_auswerten.docx
	Programmierung: Schalten mit Taster	1.5.3

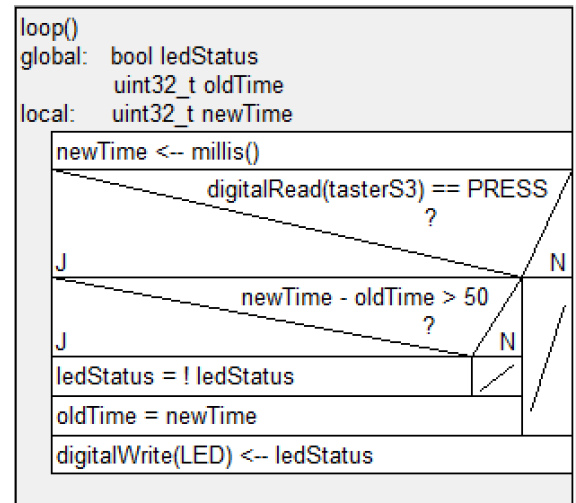
Anschluß des Tastkopfes



Vorgehensweise:

1. x10 Tastkopf anschließen. Eventuell muss ein Abgleich des Tastkopfes erfolgen.
2. X10 Tastkopf in der Eingangskonfiguration des Skopes einstellen. Spannungsablenkung ca. 1V/DIV. Nullline nach unten schieben. Maximal zu erwartender Spannungswert ist 5V.
3. Zeitablenkung auf ca. 5ms/DIV einstellen. (Abhängig vom verwendeten Taster.)
4. Oszilloskop auf Single- oder Repeat-Trigger. Trigger auf ca. 2 V und steigende Flanke.
5. Oszilloskop in den RUN-Mode schalten.
6. Tastendrucke (S3) so oft wiederholen, bis mehrere Impulse klar erkennbar sind. Evtl. Trigger auf **fallende** Flanke ändern.


Der Prellvorgang dauert in der Regel nur wenige Millisekunden, kann aber auch mal einige 10ms dauern. Um das Tastersignal zu entprellen, kann man dafür sorgen, dass nur Impulse die länger als die Prellzeit sind, vom Programm ausgewertet werden. Der abgebildete Algorithmus verwendet die Arduino-Funktion **millis()**¹. Diese liefert die Zeit in ms als 32 Bit Ganzzahl, seit dem letzten Reset des Controllers. Ähnlich dem Zusammenspiel der bekannten Statusvariablen **oldStatus** und **newStatus**, werden nun zwei Zeitvariablen benötigt, **oldTime** und **newTime**.



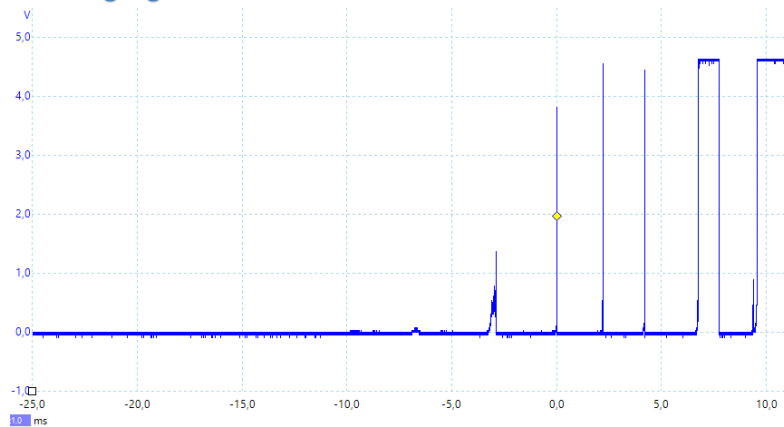
Arbeitsauftrag

1. Erstellen Sie das Programm nach dem abgebildeten Algorithmus und testen Sie es.
2. Machen Sie sich die Arbeitsweise der Tastenentprellung klar.
(Tip: Es werden nur Impulse > 50ms erkannt.)
3. Dokumentieren Sie alle Ergebnisse.

¹ <https://www.arduino.cc/reference/de/language/functions/time/millis/>

 Friedrich-Ebert-Schule Esslingen FES	IT: Hardwarenahes Programmieren	Name: Rahm Datum: 17.10.2022 1_5_Tastendruck_auswerten.docx
	Programmierung: Schalten mit Taster	1.5.4

Prellvorgang beim Loslassen des Tasters



Wenn's mal mit den fast prellfreien Tastern nicht klappt, kann man einfach einen Taster, der im Idealfall „gut“ prellt auf dem Bread-Board aufstecken und mit Patchkabeln oder Drahtbrücken an io19 = S3 anschließen.

Prellen eines nicht prellfreien Digitasters

