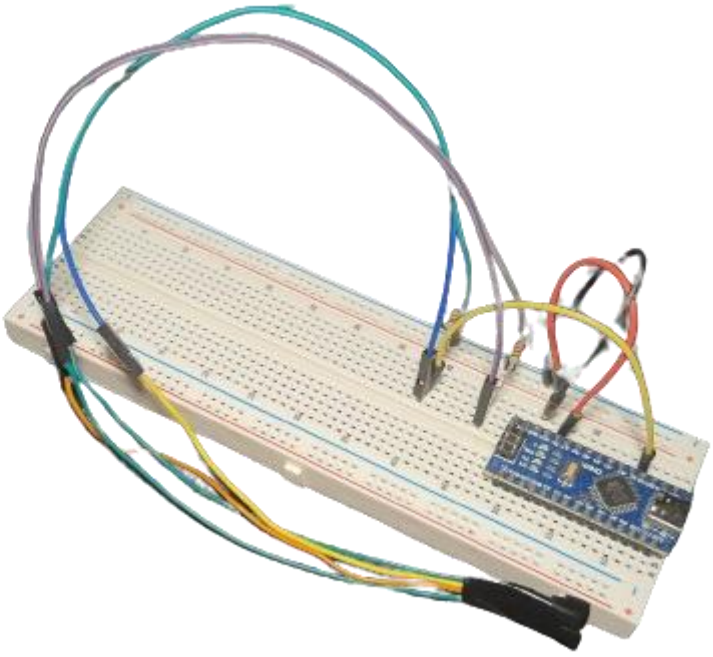
Waibl Leonhard

Projektdokumentation: Barcodescanner

Das Projektziel ist das Einlesen eines **EAN-8** Barcodes mit Hilfe eines selbstgebauten Sensors. Der Sensor besteht aus Bauteilen für wenige Euro: einem **Phototransitor** und einer **LED**. Die Datenverarbeitung erfolgt durch einen **Mikrokontroller** (Arduino).



Fertiges Projekt

Waibl Leonhard

Benötigte Hardware & Software:

Hardware:

Produkt	**Preis in €**
Phototransistor (**PT204-6C**)	1,07
LED (**Rot**)	0,02
Widerstand (**220Ω & 1kΩ**)	0,01/ 12€ kit
Jumper Kabel x2	0,17
Breadboard Kabel mit inbegriffen	4
Arduino Nano	6-21
Gesamt	**Ca. 15**

Alle Angaben von Amazon.it
Produkte können auch günstiger gefunden werden.

Software:

Es wird nur ein **IDE** benötigt, welches die Programmierung von Arduinos unterstützt. Ich habe **Visual Studio Code** mit **PlatformIO** benutzt. Eine andere Möglichkeit wäre das **Arduino IDE**.

Arbeitszeit:

Arbeitsschritt	**Zeit in Stunden**
Sensordesign	6
Code Design	3
Programmieren	35
Projektbeschreibung	6
Gesamt	**Ca. 50**

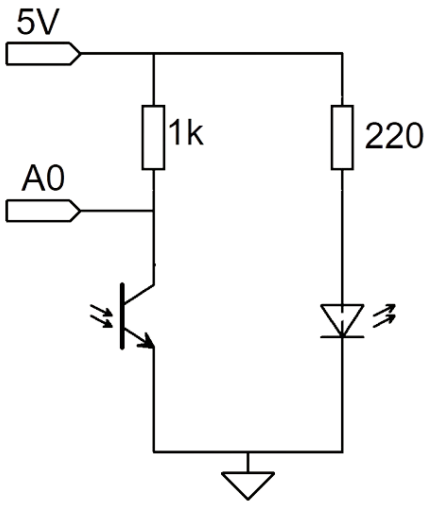
Waibl Leonhard

Arbeitsablauf:

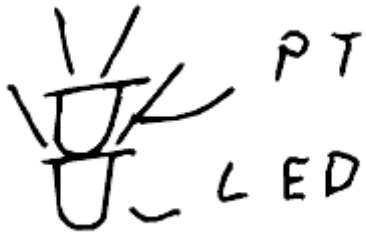
Das Projekt wurde in mehreren Schritten abgearbeitet. Als erstes wurde der Sensor entwickelt, danach ging es zur Software.

Hardware (Sensor):

Ich habe das Projekt zur Entwicklung des Sensors begonnen. Die **Grundschaltung** sowie eine **Idee** haben wir bereits im Unterricht vorgegeben bekommen. Zunächst habe ich mich an dieser Idee orientiert, **aber dann mein eigenes Konzept entwickelt**. Da ich den Sensor zuerst nicht gelötet hatte, kam es zu **Glitches am Eingang**, weshalb ich ihn nochmals umbauen musste. Die Designs sind in den folgenden Darstellungen zu sehen.



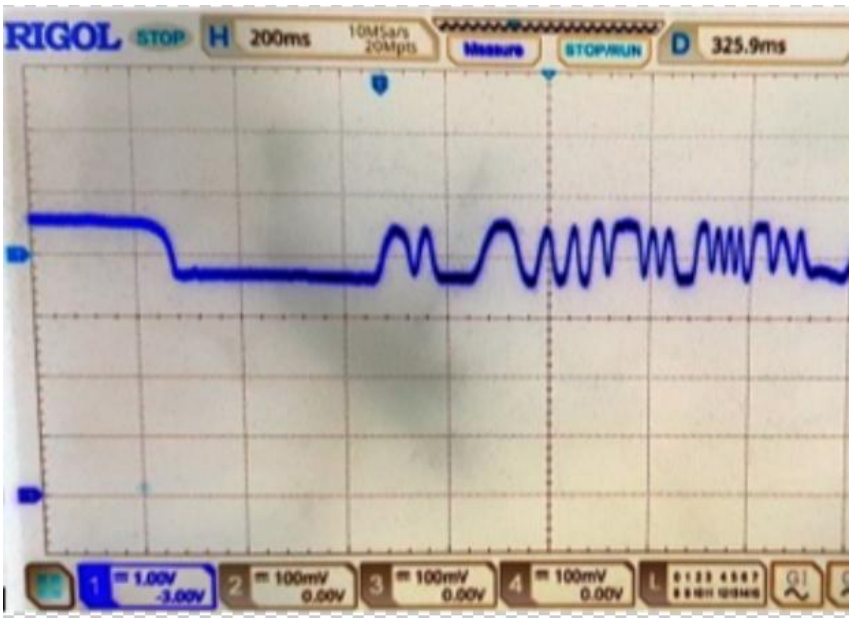
Grundschaltung



Sensor Skizze

Waibl Leonhard

Sensor Bild



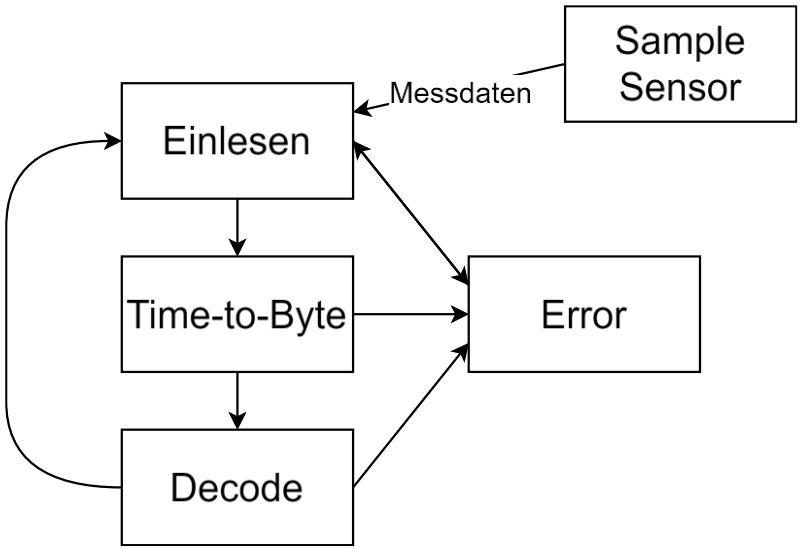
Sensor Signal

Waibl Leonhard

Software:

Ich habe mich für eine **objektorientierte Lösung (OOP)** entschieden damit das Programm einfacher in Kombination mit anderer Software genutzt werden kann. Für eine reibungslose Integration mit anderen Anwendungen müssen jedoch noch einige Teile überarbeitet werden. Beispielsweise könnte eine **Flag** eingeführt werden, die anzeigt, wenn die Software gerade Daten einliest und nicht gestört werden darf. Außerdem sollte der Zugriff auf die ausgelesenen Daten vereinfacht werden.

Es wurden zuerst die einzelnen **Unterelemente** ausprogrammiert. Der gesamte Code wurde dann in einer **Status-Maschine zusammengeführt**. Für das einfachere Testen habe ich auch einen **Barcodegenerator** geschrieben, der die eingelesenen Zeiten simuliert. Auf Anfrage kann ich diesen auch freigeben.



Status-Maschine

Waibl Leonhard

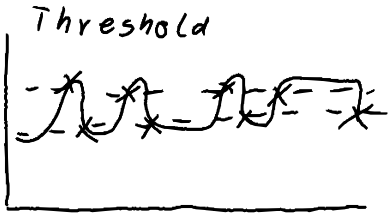
Unterprogramme in der Übersicht:

1) **Sample Sensor**

Dieses Programm liest jede **Millisekunde** einen Sensorwert aus. Dieser wird dann beim Einlesen verarbeitet. Im **Advanced-Mode** werden mehrere Werte gespeichert, um damit den **Gradienten** der Eingangsfunktion zu berechnen.

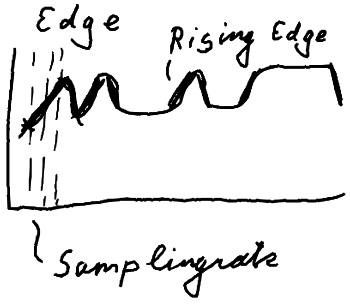
2) **Einlesen**

Ich habe mich lieber in die Entwicklung der Software gestürzt. Zuerst habe ich über ein **Auslese-System für den Sensor** nachgedacht. Die einfachste Lösung, die mir gleich in den Sinn kam, war **die Schwellenerkennung**. Dabei wurde das Signal als **High** erkannt, wenn die Spannung über eine Schwelle stieg, und als **Low**, wenn sie unter eine andere fiel.



Schwellenerkennung

Eine andere Idee beschäftigt sich mit **Flanken**. Wenn ein Signal innerhalb von **5 ms** beispielsweise um **100** ansteigt, wird dies als **steigender Pegel** erkannt. Wenn es in der gleichen Zeit wieder abfällt, handelt es sich um eine **fallende Flanke**. Für diese Idee muss eine **Samplingrate** festgelegt werden, da eine gewisse Zeit während der Messungen verstreichen muss. Eine weitere Verbesserungsmöglichkeit wäre, **obere und untere Schwellen beizubehalten und diese dynamisch nach einer Flanke zu verändern**. Dadurch darf eine Spitze nur eine gewisse Abweichung zur vorherigen haben.



Flankenerkennung

Waibl Leonhard

3) Time to Byte

Um ein **Byte zu berechnen**, kann mithilfe der **Referenzstreifen** eine durchschnittliche Zeit pro Bit ermittelt werden. Nachdem diese berechnet wurde, teilt man einfach jede Flanke durch diese Zeit und rundet auf die nächste Ganzzahl. Diese Methode hat jedoch ihre Tücken. Es kann vorkommen, dass man während der Fahrt schneller wird und die Durchschnittszeit nicht mehr stimmt. Im **Advanced-Modus** wurde das durch Segmentierung behoben.

4) Decoding

Wenn die **Bytes bekannt sind**, muss man eigentlich nur noch **suchen und ersetzen**.
Die **Bytewerte für den EAN-8 Barcode** können hier gefunden werden. Zusätzlich kann man das Programm noch damit ergänzen, dass ein Barcode auch **rückwärts gelesen werden kann**. Dafür müssen dann einfach die Werte umgekehrt werden.

5) Error Handling

Es werden keine genauen **Fehler** aufgenommen. Es gibt Fehler beim **Einlesen**, bspw. wenn beim Einlesen zu lange **keine Flanke** erkannt wird. Der andere Fehler geschieht, wenn, die gemessenen Daten nicht im **EAN-8 lookup-table** gefunden werden können. Bei einem Fehler wird das Programm **resettet und startet von neuem**.