Audio-Kontrolle für ZOE

Rüdiger.Brause@web.de

20.10.2019

1. Spezifikation

Das System sollte es ermöglichen, beliebige Audiodateien abzuspielen. Dies soll durch Taster (Sensortasten) geregelt werden mit folgenden Bedeutungen:

Taste A
1x kurz tasten = AN; 1x kurz tasten = AUS.
Nicht implementiert, da in der Praxis nicht benutzt:
1x lang tasten = lauter.
2x kurz tasten = auf StandardLautstärke zurücksetzen

Taste B
1x kurz tasten = nächste Tondatei.
1x lang tasten = vorherige Tondatei.

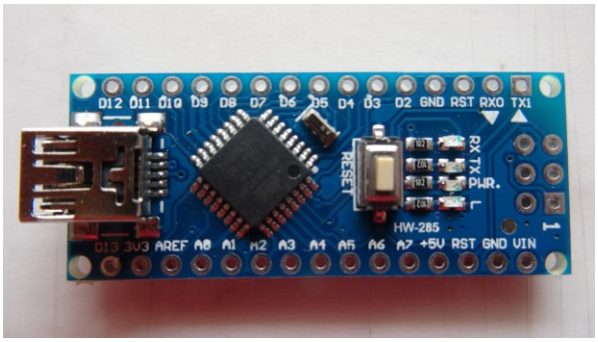
Eingabeleitung C
HIGH = Rückfahrwarnton an
LOW = Rückfahrwarnton aus

Taste A und Taste B gleichzeitig gedrückt (2 Finger)
AudioDatei Nr. 0 abspielen (Sonderton)

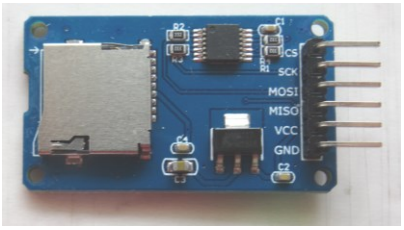
2. Installation mit dem Arduino-Entwicklungssystem

1.Schritt: Hardware kaufen

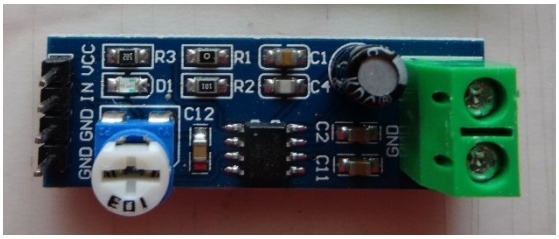
Das Arduino-System besteht aus einer Entwicklungssystem, das auf einem Rechner (PC, Notebook,…) installiert wird und per USB-Kabel mit einem Microprozessor verbunden ist. Im Unterschied zu dem Raspberry-Pi-System ist der Mikroprozessor, meist ein ATME 328, nur mit wenig Speicher ausgestattet und damit sehr preiswert. Für unseren Zweck benötigen wir nur einen Arduino nano V3.0 (3,43€) von TECNOIOT



sowie in SD-Karten-Modul (1,60€) von KeeYees SPI Reader Micro Speicher SD TF



und ein Verstärkermodul, hier mit dem LM 386 (1,16€).



Alles zusammen für 6,19€. Dazu kommen natürlich noch zwei Touch-Sensoren (TTP223B Digitaler Kapazitiver Touch Sensor, je 1€) und diverses Kleinmaterial (Kabel, Stecker, etc.). Also alles unter 10 Euro.

2.Schritt: Entwicklungssystem installieren

Zum Einrichten des Entwicklungssystems IDE reicht es, die neueste Arduino-Software-Version von https://www.arduino.cc herunterzuladen und zu installieren.

Für die Audiofunktionen benutze ich die TMRpcm()-Bibliothek. Sie wird im Entwicklungssystem durch die Menüpunkte *Werkzeuge/Bibliotheken verwalten…* installiert. Hier gibt man den Namen in der Suchzeile oben ein und klickt dann auf „installieren“. Das war‘s.
Für den Arduino Nano wird noch der neue USB-Treiber für den CH340-Chip auf dem Arduino benötigt. Dazu schließt man den Mikroprozessor mit Hilfe eines USB-Ministecker-Kabels auf Prozessorseite und des USB-Steckers auf PC-Seite an. Unter MS-Windows wird nun ein unbekanntes Gerät im Gerätemanager angezeigt. Eine online-Suche installiert prompt den korrekten Treiber.
Für alle weiteren Schritte sollte der Mikroprozessor per USB Mini-Kabel angeschlossen bleiben. Als Grundeinstellung sollte man nun das Board mittels Klick auf den Menüpunkt *Werkzeuge/Board/Aduino Nano,* auf den Punkt *Werkzeuge/Prozessor/ATmega 328 (Old Bootloader)* sowie auf *Werkzeuge/Port/COM9* wählen.

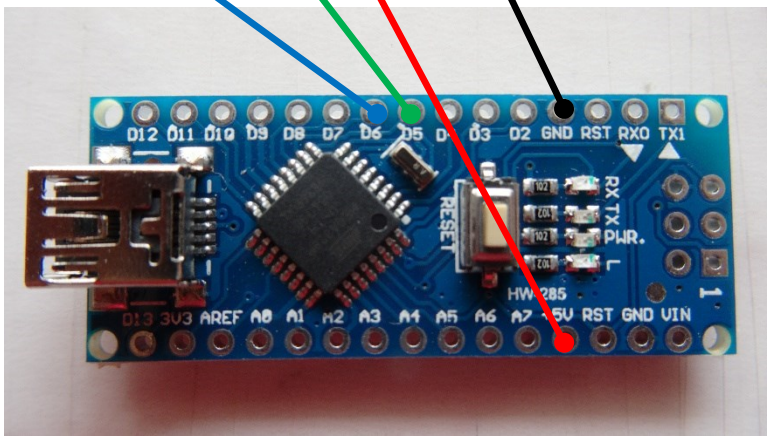
3.Schritt: System testen

Nun kann man testen, ob das IDE richtig funktioniert. Dazu laden wir uns ein simples Programm hoch, etwa mit *Datei/Beispiele/Basics/Blink*, und compilieren und Laden das Programm, indem wir auf das „Pfeil rechts“Symbol klicken. Darauf sollte die onboard-LED auf dem Arduino anfangen, im 1 sek Takt zu blinken.
Falls stattdessen Fehlermeldungen unten im Programmfenster auftreten, lese man diese

sorgfältig durch. Es gibt eine FAQ, die man auch über *Hilfe/Fehlersuche* erreichen kann.

4. Schritt: Module verdrahten und Audioprogramm einrichten

Als Eingänge wählen wir uns die Pins für die Tasten A und B aus. Der Kontakt wird zwischen einem Pin und +5V hergestellt (HIGH), die Touchsensoren halten bei Inaktivität den Pegel auf 0V (LOW). Wir verbinden dazu Pin 8 (D5), 5V und GND mit den drei Kontakten des ersten TouchSensors A sowie Pin 9 (D6), 5V und GND mit den drei Kontakten des zweiten TouchSensors B.



Um den Zustand eines Pins abzufragen, wird im Programm eine spezielle Abfrage benutzt. Sie lautet hier nach der Initialisierung

```
const int button = 5; // D5 für ersten button
pinMode(button, INPUT); //Button einrichten: input mode
```

als direkte Abfrage

```
Ergebnis = digitalRead(button)// Zustand des Button abfragen
```

Dies müssen wir dauernd in einer Schleife abfragen. Nun geht das Abfragen sehr schnell, so dass wir nach einer Reaktion sicherstellen müssen, dass zuerst der Tastenzustand wieder normal, d.h. LOW ist, bevor wir wieder abfragen für eine nächste Reaktion.
Der Code lautet dazu

```
boolean enable = true;       // ermögliche Reaktion

Void loop {           // Wiederhole den folgenden Code ewig
  if (digitalRead(button)==HIGH  && enable == true){
              // reagiere auf Knopfdruck
          enable=false;     // und verhindere erneute Reaktion
                            //bei nächstem Schleifendurchgang
  }
  if (digitalRead(button)==LOW)      //Taste im Grundzustand?
               enable=true;          // dann ja

}// -------------- loop ende ---------------------------
```

Dies ist der grundsätzliche Code um beim Drücken der Taste (Übergang LOW ->HIGH), um einen Ton abzuspielen.

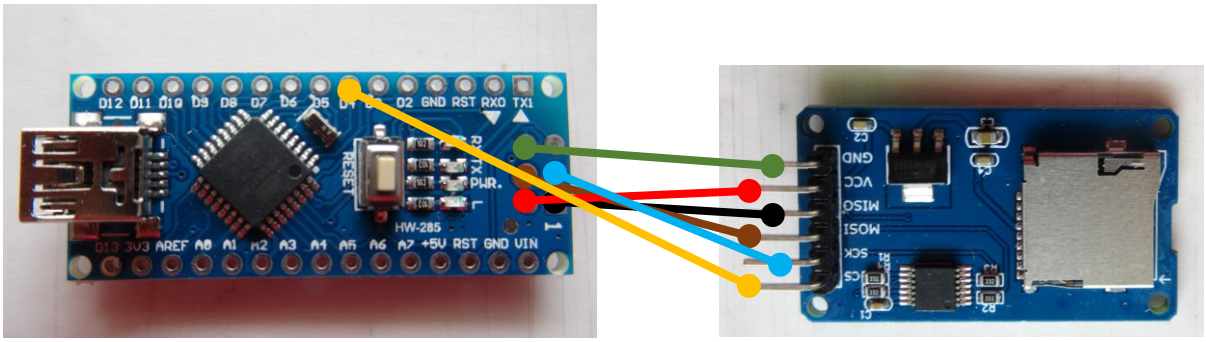
Nun müssen wir noch die Attribute „lang“ und „kurz“ definieren. Dazu müssen wir Zeitmessungen einführen. Hier ein Testprogramm.

```
t1 = millis();    // registriere den Zeitpunkt
while (digitalRead(button)==HIGH) ; // do nothing
t2 = millis();    // registriere den zweiten zeitpunkt
if ((t2-t1) < shortR){ // wenn die Tastendrucklänge klein ist
  // mach was
} else {
  // sonst mach was anderes
}
```

Die Differenz t2-t1 lässt sich mit vorgegebene Werten für „kurz“ bzw. „lang“ vergleichen. Das Ergebnis führt dann zu verschiedenen Reaktionen.

Die Audiodateien auf der Mikro SD Karte werden mit Hilfe der SD-Bibliothek eingelesen. Üblicherweise werden alle Kontakte der seriellen Verbindung SPI vom Nano auf dem 6Pin-Feld mit dem SD-Modul verbunden. Dabei sind laut Schaltplan

MISO - pin 1 on ICSP
VCC - pin 2 on ICSP
CLK - pin 3 on ICSP
MOSI - pin 4 on ICSP
GND - pin 6 on ICSP



Traditionellerweise wird Pin D4 für das *chip select* Signal benutzt.

```
#include "SD.h"      //Lib to read SD card
#include "SPI.h"     //SPI lib for SD card
#define SD_ChipSelectPin 4
if (!SD.begin(SD_ChipSelectPin)) { // SD Karte ok?
      Serial.println("SD fail");
      return;
}
```

Dabei müssen die Audiodateien ein bestimmtes Format haben, um digital vom Hauptprozessor abgespielt zu werden: mono, 8 Bit, 16.000 Hz sample rate, WAV-Format.
Für das Abspielen der Audiodateien nutzen wir die TMRpcm Bibliothek.

```
#include "TMRpcm.h" //Lib to play audio
```

Dabei wird ein Pin für die Ausgabe definiert; hier D9.

```
audio.speakerPin = 9;    //Audio out pin
```

Und ein Objekt für die Funktionen.

```
TMRpcm audio; //create new object, named "audio"
```

Danach kann man die Funktionen der Bibliothek nutzen.

```
audio.play("audio0.wav");
```

Das ganze Programm lautet dann wie folgt.

3. Das Programm

```
/*
A program to generate sound for electric cars in order
to warn pedestrians on the road.

 Version 1.0
 created  20/09/2019
 by Rüdiger Brause, Bad Soden, Germany

As warning sound, several audio files in 16kHz 8Bit mono wav format
is used which has to be placed in the roor directory of a SD card. They
have to be named "audioX.wav" where X is 0..9.
To control the sound play, two sensor buttons are used
on the front panel in the car.

A third input is given by the backward light,
producing a warning sound when driving backwards.

Additional to the audio ouput, the standby line of an external audio ampli-
fier
is put HIGH (play) whenever a sound is played.
Since it is the unique process, we can deal everything in a big loop with-
out any interrupts.

button 1
1x shortly pressed = ON; 1x shortly pressed = OFF.

button 2
1x shortly pressed = next audio file.
1x longly pressed  = previous audio file.

button3: input line of back lights
line connected to GND    = buzzer off
line connected to 12V    = buzzer on

button A and button B pressed at the same time
 = play audio file #0, a special sound

The audio files are SD card, played by pressing push buttons
```

```
 The circuit:
 * Push Buttons on pins D5,D6 and D7,
 * amplifier standby control on pin D8
 * Audio Out on pin D9
 *
 * SD card attached to SPI bus as follows (ICSP pins: of Arduino nano):
 ** CS   - pin D4
 ** MISO - pin 1 on ICSP
 ** VCC  - pin 2 on ICSP
 ** CLK  - pin 3 on ICSP
 ** MOSI - pin 4 on ICSP
 ** GND  - pin 6 on ICSP
 */

#include "SD.h"     //Lib to read SD card
#include "SPI.h"    //SPI lib for SD card
#include "TMRpcm.h" //Lib to play audio

// global constants
#define SD_ChipSelectPin 4 //Chip select is pin number 4
const int button1 = 5;    // touch sensor button for PLAY/STOP
const int button1_on = HIGH; // normally LOW, active on HIGH
const int button2 = 6;    // touch sensor button for change of audio sound
file
const int button2_on = HIGH; // normally LOW, active on HIGH
const int button3 = 7;    // button pressed by back driving
const int button3_on = HIGH; // normally LOW, active on HIGH
const int amplifier = 8;  // output pin for amplifier standby control
const int amplifier_on = HIGH; // high level switches the amplifier on
const int amplifier_off = LOW; // high level switches the amplifier off
const int Audio_pin = 9;  // get audio output on pin 9
const int file_num = 10;  // number of audio files to consider
const unsigned long shortR = 400;  // a short touch duration in millisec

// global var
TMRpcm audio; //create new object, named "audio"
int audio_number=1;
boolean enable1 = true; // This is necessary for avoiding
boolean enable2 = true; // the effect of bouncing contacts
boolean enable3 = true;
char filename[] = "audio1.wav";
boolean repeat = false;
boolean special_play = false;


// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void setup(){

pinMode(button1, INPUT); //Button 1 to play/pause
pinMode(button2, INPUT); //Button 2 to change track
pinMode(button3, INPUT); //Button 3 by backlight line
pinMode(amplifier, OUTPUT);     //Pin for amplifier standby control
digitalWrite(amplifier, amplifier_off); // initially disable amplifier
audio.speakerPin = Audio_pin;   //Audio out pin
Serial.begin(9600);     //Serial Com for debugging
if (!SD.begin(SD_ChipSelectPin)) {
  Serial.println("SD fail");
  return;
}
audio.setVolume(5);          //  Set volume level between 0 to 5.
```

```
audio.quality(1);            //  Set 1 for 2x oversampling, set 0 for nor-
mal
audio.loop(true);         // always repeat the sound
digitalWrite(Audio_pin,HIGH); // no initial noise on audio pin, please

}//
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void loop()
{
  // Button 1 was depressed
  if (digitalRead(button1)==button1_on  && enable1 == true) //Button 1
Pressed
  { enable1 = false;
    Serial.println("LEFT KEY PRESSED");
    audio.stopPlayback();     // stop if something is still playing
    while (digitalRead(button1)==button1_on &&
           !digitalRead(button2)==button2_on) ; // wait for action
    if (digitalRead(button2)==button2_on){ // Button 2 although pressed ?
        enable2 = false;
        digitalWrite(amplifier, amplifier_on); // enable amplifier
        audio.play("audio0.wav");   // play special file
        Serial.println("Start special playing");
        delay(500); // wait a small time before next action
        repeat = true;          // stop on next left touch
    }
    else if (!repeat) { // no repeat, first left press
        digitalWrite(amplifier, amplifier_on); // enable amplifier
        audio.play(filename);   // play current file again
        Serial.println("Start playing");
        repeat = true;          // stop on next left touch
    } else {  // stop on 2nd touch
        Serial.println("Stop playing");
        digitalWrite(amplifier, amplifier_off); // disable amplifier
        repeat = false;   }
  }

  // Button 2 was depressed
  if (digitalRead(button2)==button2_on  && enable2 == true) //Button 2
Pressed
  { unsigned long t1; unsigned long t2;
    enable2 = false;
    Serial.println("RIGHT KEY PRESSED");
    t1 = millis();
    while (digitalRead(button2)==button2_on) ; // do nothing
    t2 = millis();
    if ((t2-t1) < shortR){
      audio_number++;
      if (audio_number==file_num) audio_number=1;
    } else {
      audio_number--;
      if (audio_number<1)  audio_number = file_num-1;
    }
    filename[5] = '0'+audio_number; // compute file number
    Serial.print("audio_number="); Serial.print(audio_number);
    Serial.print(" file="); Serial.println(filename);
    digitalWrite(amplifier, amplifier_on); // enable amplifier
    audio.play(filename);   //Play song from start asynchronously
    repeat = true;
  }

  // Button3 is triggered by the rear lights on
```

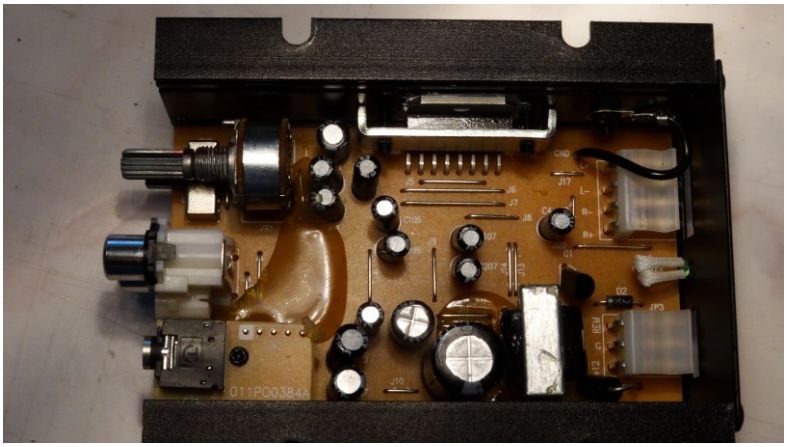
```
  if (digitalRead(button3)==button3_on  && enable3 == true) //play signal
tone if car moves backwards
  { boolean was_playing;
     enable3=false;
     was_playing = audio.isPlaying();
    if (was_playing) {
        audio.stopPlayback();
        Serial.println("stop playing");
    }
    else
        digitalWrite(amplifier, amplifier_on);
    audio.play("back.wav");  // play beep tone for backward driving
    while (digitalRead(button3) == HIGH){} // busy wait
    audio.stopPlayback();
    if (was_playing)
      audio.play(filename);
    else
      digitalWrite(amplifier, amplifier_off);     // put amplifier in
standby mode
  }

  // enable button actions after setting back
  if (digitalRead(button1)==!button1_on)  //Button reset?
    enable1=true;
  if (digitalRead(button2)==!button2_on)  //Button reset?
    enable2=true;
  if (digitalRead(button3)==!button3_on)  //Button reset?
    enable3=true;
}
```

4. Die Implementierung

Der Verstärker
Die Audio-Ausgabe des Arduino Nano ist sehr begrenzt und muss deshalb verstärkt werden, etwa mit dem LM328 aus, wie in Abschnitt 1 beschrieben. Zur Ansteuerung eines Leistungsverstärkers für die Lautsprecher reicht dies dann aus.

Eine preiswerte Möglichkeit eines Leistungsverstärkers ist ein billiger Allround-Verstärker mit 12V Betriebsspannung (30€) , etwa der Excalibur XM-2.250. Allerdings reicht die interne Verstärkerleistung bei diesem nicht aus. Dazu schrauben wir ein Seitenteil auf. Da der LeistungsIC zur Kühlung an der Seite (am oberen Rand zu sehen) angeschraubt ist, müssen wir auch die seitlichen Schrauben lösen. Dann lässt sich die gesamte Platine seitlich aus dem Alu-Profil herausziehen.



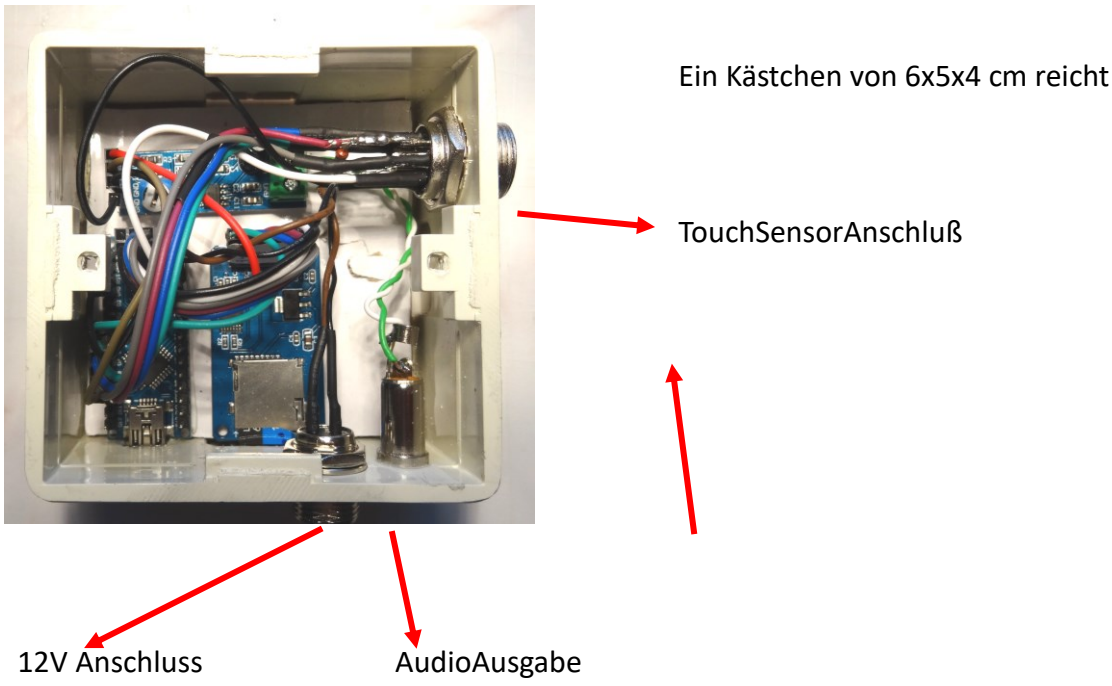
Auf der Unterseite der Platine ist ein schwarzer Plastik-IC zu sehen: der Vorverstärker 4558. Hier müssen wir die Gegenkopplungswiderstände R102 und R202 von 15kOhm (SMD-Technik) auslöten und stattdessen Widerstände mit größerem Wert (also größerer Verstärkung), etwa 50kOhm, einlöten.

Lautsprecher
Für die Lautsprecher habe ich mich für wasserfeste Lautsprecher entschieden, hier für die CT-50 von ChiliTec (Paar für 14€) mit max. 15W RMS / 80W Musikbelastung.

Signale
Für die Eingabetaster verwende ich Tastsensoren, hier den touch sensor TTP223B, z.B. von der Fa. AZ-Delivery.
Das Rückfahrsignal von 12V wird mit einem Widerstandsteiler von 50kOhm zu 15 kOhm auf 5V abgesenkt.

Spannungsversorgung
Als Spannungsversorgung für den Arduino Nano reicht eine 12V Zuführung vom Sicherungs-kasten im Innenraum aus. Die Versorgung geht hierbei zu einem speziellen Pin 30, der mit „UIN“ betitelt die äußere Spannung auf interne 5V herabregelt. Dies kann dann von den anderen Moduln (SD-Kartenleser, LM328 Verstärker) genutzt werden.

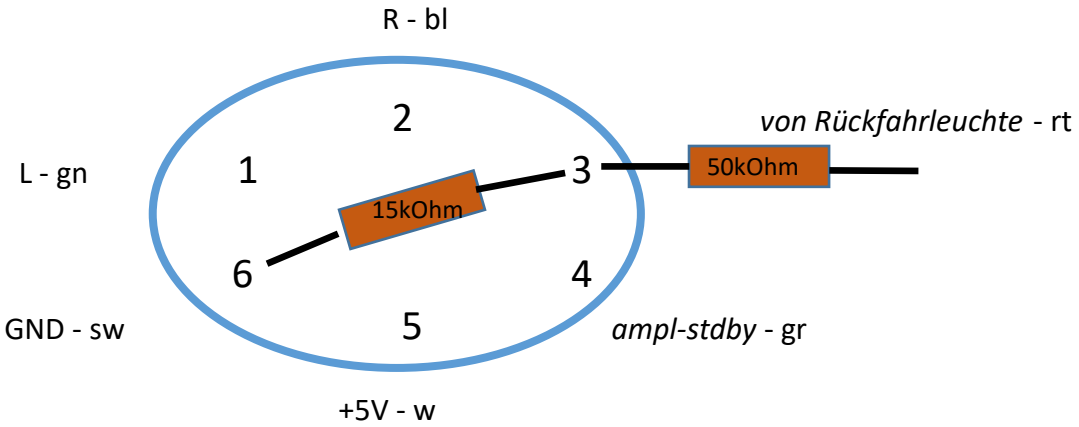
Alle Module lassen sich so in einem Kästchen bündeln und mit robusten Buchsen und Steckern versehen, um so die Elektronik zuerst auf dem Werktisch testen zu können.



Schema TouchsensorAnschluß

1	D5	gn – Linker Sensor Eingang, +5V bei Touch, sonst 0V
2	D6	bl – Rechter Sensor Eingang +5V bei Touch, sonst 0V
3	D7	rt – Rückfahrleuchte *back*-Signal, 12V auf 5V abgesenkt mit 50kOhm
4	D8	gr – *amp-stdby* Ausgabe zum Verstärker. +5V bei *amp_on*, sonst 0V
5	Ucc	w – +5V Ausgang für die Sensortaster
6	GND	sw - GND

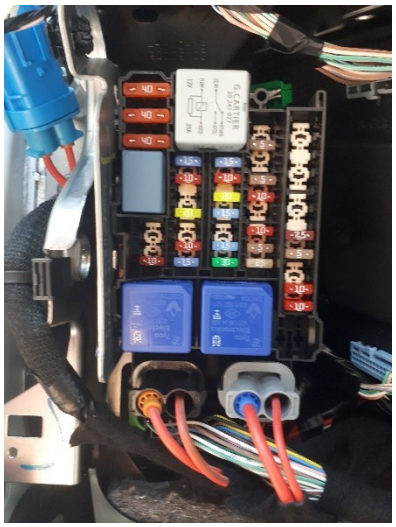
Buchse von hinten gesehen: Kontakte 7,8,9 nicht belegt. Kontakt3 mit 15kOhm zu GND Kontakt6 verbunden, so dass an Kontakt3 max 5V anliegen.



Die Anschlussbuchsen sollten sehr solide ausgeführt werden, da hier leicht Wackelkontakte entstehen.

5. Der Einbau in den Zoe.

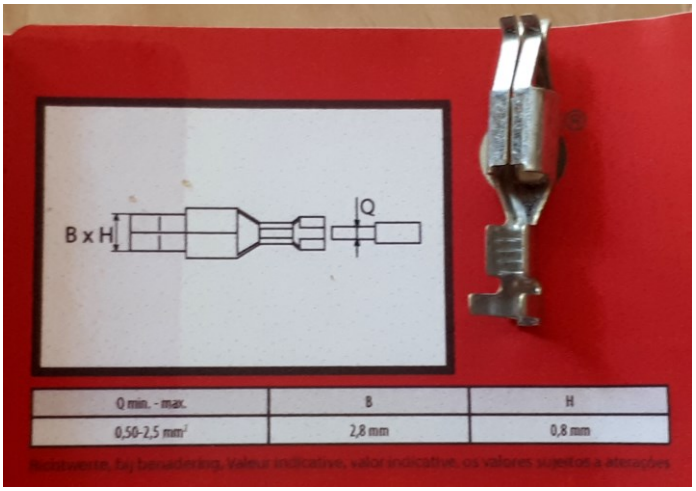
Zuerst ist dazu die innere Blenden am Armaturenbrett abzumontieren, siehe Anleitung hier.
Es ergibt sich folgendes Bild:



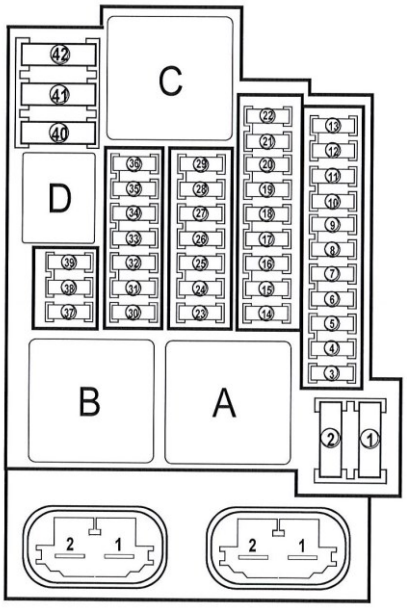
Für den Arduino ist ein 12V-Anschluss vorzusehen sowie ein 12-Anschluss für die Verstärker-Endstufe. Beide können über eine 5A-Mini-Sicherung abgesichert werden. Für den Einbau dieses 12V-Anschlusses gibt es zwei Möglichkeiten:

1. Man verwende Sicherungsplatz F16, die dritte Sicherung von unten der zweiten Spalte von rechts. Dies ist die Sicherung für den bereits eingebauten Fahrgeräuschsimulator, den sog. Warntongeber. Dazu besorge man sich ein sog. Stromdieb, also einen Flachsicherungsverteiler, nehme die Mini-Sicherung heraus, stecke den Verteiler ein und in den Verteiler sowohl die alte 10A-Sicherung als auch die neue 5A-Sicherung.

2. Man verwende einen freien Sicherungsplatz, etwa Sicherungsplatz F22, der oberste Platz der zweiten Spalte von rechts. Dazu benötigt man einen Crimp-Federsteckereinsatz, der von hinten den leeren Platz füllt.



Das Anschlusskabel wird an den Stecker angecrimpt und dann der Stecker von hinten eingesteckt, bis er fest sitzt. Dann von vorn die Mini-Sicherung einstecken.



Zum Anschluss und Kontrolle des Arduino verwende ich einen Wipp-Schalter, den ich an der Innenverkleidung rechts neben dem werkseitig eingebauten Fahrgeräusch-Simulator des ZOE eingebaut habe. So kann man das ganze System abschalten, falls irgendwas schief läuft.

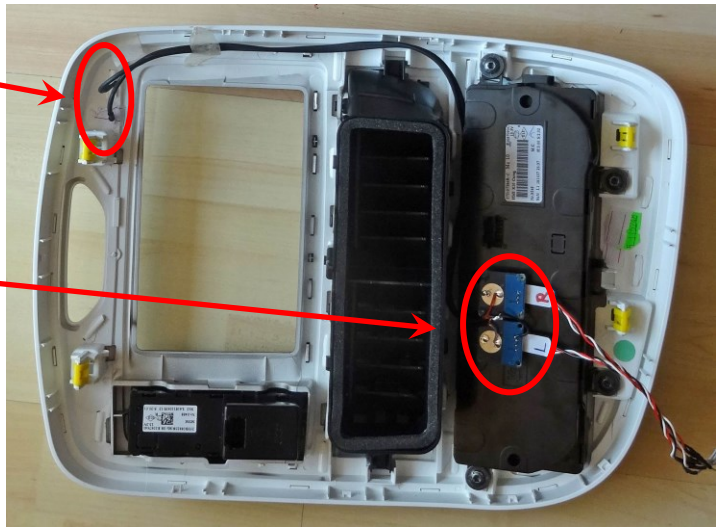
Das Audiokabel und die 12V-Versorgungskabel (Plus, Masse) sowie die Anzapfung des Rückfahrsignals und das *amp-std-by*-Kabel müssen durch die Trennwand zum Motorraum geführt werden. Dazu sollte ein ausreichend großes Loch gebohrt werden, am besten von innen links von dem Kabelhauptstrang, der hinter dem Armaturenbrett durchgeführt ist. Sinnvollerweise wird dazu eine flexible Plastikröhre verwendet, in die die Kabel eingezogen werden.

Zur Montage der Sensortasten wird die Plastikabdeckung der Armaturenbrett-Mittelkonsole abgenommen, siehe Anleitung.

Die Touchsensoren werden auf der Rückseite des Bedienfelds in der Mitte angeschlossen. Dazu werden zwei Löcher hineingebohrt, in die mit kurzen dünnen Kabelstücken verlängerte vernickelte Polsternägel mit Sekundenkleber eingeklebt werden. Die kapazitive Sensorseite kann direkt an dieses Kabelstück angelötet werden.

Polsternägel von hinten

Touch-Sensoren,
mit doppelseitigem Klebeband und
mit Steckern am Kabelstrang befestigt

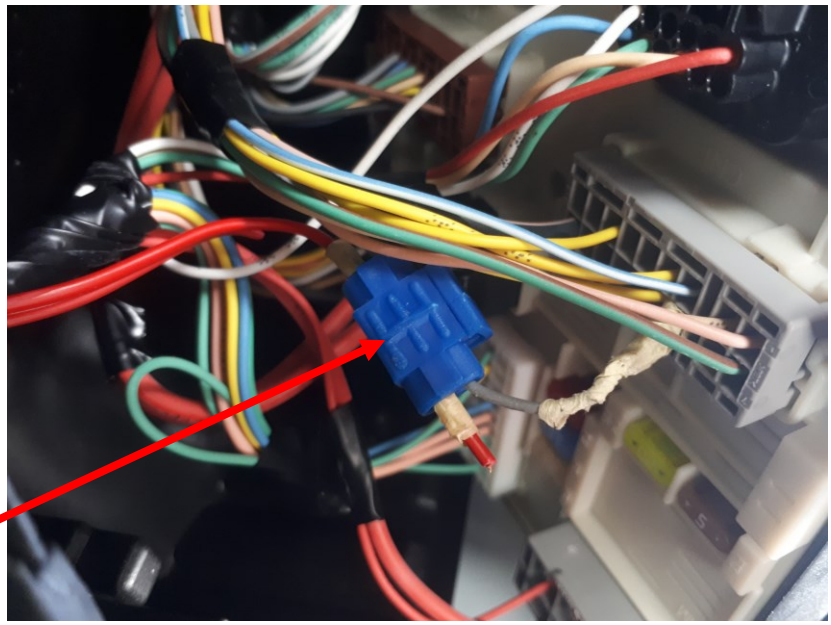


Von vorn sieht dann das Ganze so aus:

2 Touch-Sensoren

Auch die Kontrolle der Rückfahrscheinwerfer muss angezapft werden. Dazu verwenden wir das Signal, das am Schalthebel an dem Schalter im Motorraum anfällt. Es ist bei dem Sicherungskasten im Motorraum vorhanden. Nach dem Plan ist dies Pin 5 beim grauen

Stecker CN1. Hier ist das Rückfahrsignal am grauen Kabel vorhanden, wo man eine Vampirklemme (blau) anwenden kann.



Die Absenkung der +12V Spannung des Rückfahrlichts auf das vom Arduino erwartete 5V-Niveau wird erst durch Widerstände im AnschlussStecker des ArduinoModuls erreicht, siehe Abschnitt 4, Seite 10.

Für die Montage der beiden Lautsprecher muss die Vorderabdeckung (Frontschürze) des Autos abgenommen werden, siehe Anleitung dazu. Es ergibt sich folgendes Bild, nachdem die Plastikschürze vorn ausgeklinkt wurde:

Dann werden die Lautsprecher vorn links in die Plastikschürze eingeschraubt; auf der rechten Seite ist leider kein Platz frei. Dazu müssen vorher mit der Stichsäge die 12cm Durchmesser

ausgesägt werden. Danach sieht das Ganze von vorn so aus:

Und von rechts sieht man den Endverstärker mit den Zuleitungen (Audiokabel und 12 V Stromversorgung des Verstärkers).