

ESP32 Spritpreis Webradio



Walter Nordsiek
13. März 2022

Inhalt

1 Übersicht	2
1.1 Die Sinnfrage	2
1.2 Hardware- Was brauchen wir alles?	2
1.2.1 Komponentenliste	2
1.2.2 Verkabelung	3
1.3 Bedienung - Was kann es denn?	6
1.3.1 Grundfunktionen	6
1.3.2 Weitere Funktionen	7
1.3.3 Encoder	8
1.3.4 Touchfunktionen	9
2 Installation	10
2.1 ArduinoIDE und ESP32	10
2.2 Bibliotheken und Tools	11
2.2.1 VS1053	11
2.2.2 LITTLEFS	12
2.2.3 File system uploader	13
2.2.4 mklittlefs	13
2.2.5 ArduinoJson	14
3 Konfiguration	14
3.1 Tankerkönig-API	14
3.2 Netzwerk	17
3.3 Radiostationen	18
3.4 Benzinpreise	20
3.5 Gong	21
4 Inbetriebnahme	22
4.1 Einstellungen	22
4.2 Bauen und Laden	24

1 Übersicht

1.1 Die Sinnfrage

Nun – eigentlich ist die Sinnfrage »Warum das Ganze?« schnell beantwortet: Das braucht kein Mensch.

Internet- oder Webradios kann man kommerziell kaufen und Benzinpreise leicht über eine App auf dem Mobilcomputer (umgangssprachlich »Handy«) in Erfahrung bringen.

Trotzdem ein paar Worte zum Anlaß des Projekts. Es soll als Ersatz für ein wegen schlechter Empfangsverhältnisse nur unzureichend funktionierendes DAB-Radio dienen. Ein »paar Minuten« Recherche führt zu dem Webradio-Projekt¹ von Ed Smallenburg als Basis des Ganzen. Das funktioniert und ist offensichtlich universell mit allen möglichen Displays einsetzbar, aber so ganz warm geworden bin ich mit dem Code nicht. Während der Arbeit an den Änderungen tauchte im Arduino-Forum ein Beitrag über Benzinpreisanzeige² mit der API von Tankerkönig³ auf und ich habe die hervorragende VS1053-Library⁴ von Wolle gefunden. Das waren dann die Trigger, das Projekt komplett neu aufzusetzen – hier ist das Ergebnis.

1.2 Hardware- Was brauchen wir alles?

1.2.1 Komponentenliste

Die Komponentenliste ist überschaubar:

- ESP32 Modul
- Display Nextion NX4832T035
- MP3-Decoder VS1053 Modul
- ein Verstärker
- Drehencoder
- Stromversorgung
- zwei kleine Audioübertrager

¹<https://github.com/Edzelf/ESP32-Radio>

²<https://forum.arduino.cc/t/projektvorstellung-spritpreis-von-deiner-heimtankstelle-mit-esp32-auf-display-anzeigen/924802>

³<https://creativecommons.tankerkoenig.de/>

⁴https://github.com/schreibfaul1/ESP32-vs1053_ext

- zwei Lautsprecher und Abdeckungen
- Kabel, Stecker und Knöpfe

Es gibt für alles jeweils verschiedene Ausführungen; in diesem Projekt eingesetzt sind die folgenden Teile - bei den Amazon-Links sind z.Zt. (13.3.2022) Display und VS1053 nicht verfügbar:

- <https://www.az-delivery.de/products/esp32-dev-kit-c-unverlotet>
- <https://www.amazon.de/dp/B07PLNRPMH>
- <https://www.amazon.de/dp/B07XCTJZVK>
- <https://www.amazon.de/dp/B07QF93WWL>
- <https://www.az-delivery.de/products/drehimpulsgeber-modul>
- <https://www.amazon.de/dp/B07R64MNGZ>
- <https://www.amazon.de/dp/B075R26XWC>
- <https://www.amazon.de/dp/B0017KT3S2>
- <https://www.amazon.de/dp/B003A6C3CY>
- <https://www.amazon.de/dp/B081VMBQQS>
- <https://www.amazon.de/dp/B07H9RQM5W>
- <https://www.amazon.de/dp/B0009Y80AK>

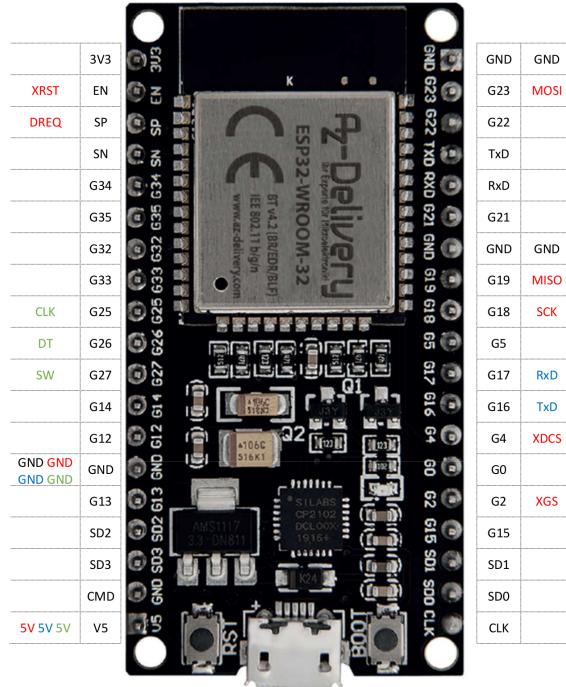
1.2.2 Verkabelung

Bei der Verkabelung habe ich mich an den Vorgaben von Edzelf und Wolle orientiert.

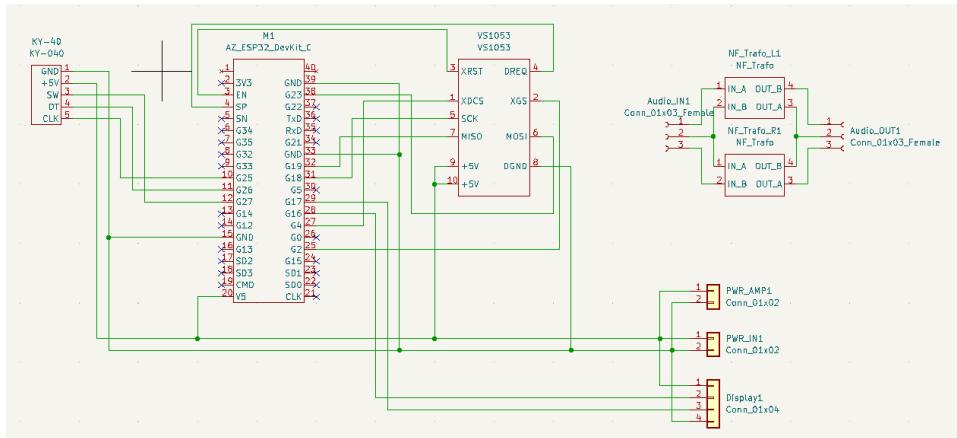
Es gibt unterschiedliche Auffassungen über die 5V-Fähigkeit der Ein- und Ausgänge von ESP32-Modulen. Ich habe auf Widerstände oder LevelShifter für die serielle Schnittstelle verzichtet und bisher (das Projekt ist jetzt über ein halbes Jahr »in der Mache«) gehabt.

Die Audiosignale laufen vom Kopfhörerausgang des VS1053-Moduls über jeweils einen Trenntrafo für rechten und linken Kanal zum Verstärkereingang. Damit habe ich das sonst laut und deutlich vernehmbare Brummen einigermaßen eliminieren können, wenn der Ausgangspegel am VS1053 genügend hoch eingestellt wird.

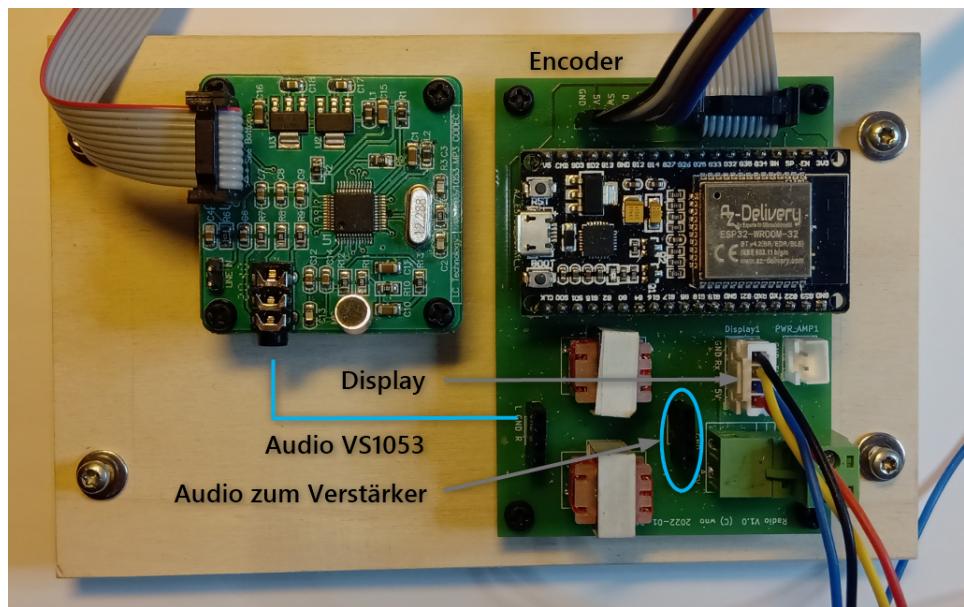
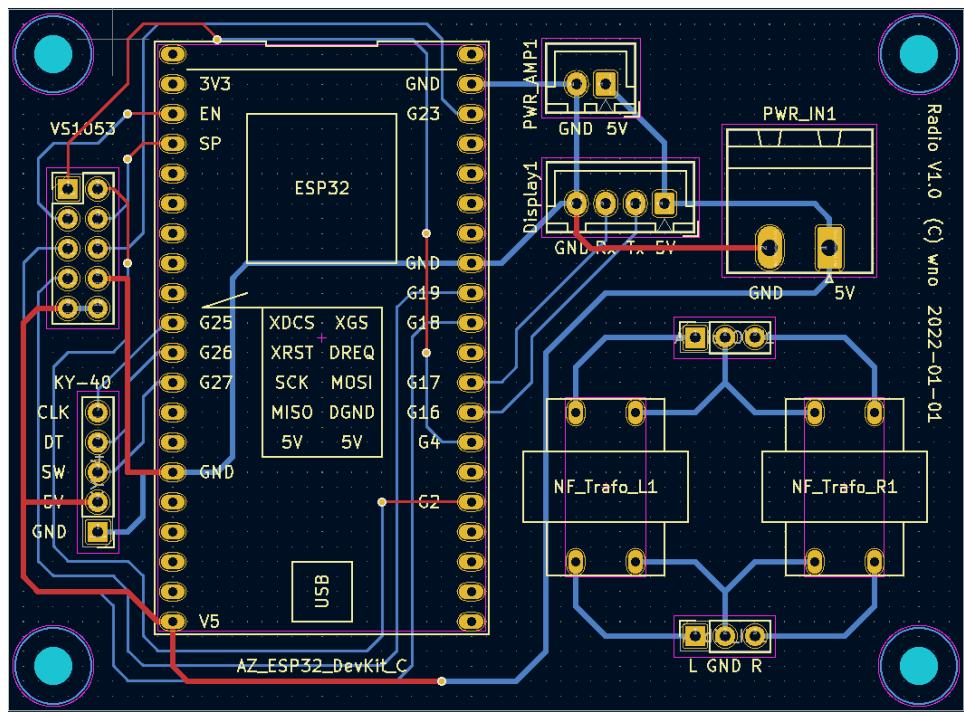
VS1053 Board Nextion Encoder



Da das im Laboraufbau (Breadboard) recht wild aussah, habe ich eine kleine Platine⁵ fertigen lassen. Damit ist es erträglich geworden.



⁵<https://aisler.net/p/IJ0XXPAY>



1.3 Bedienung - Was kann es denn?

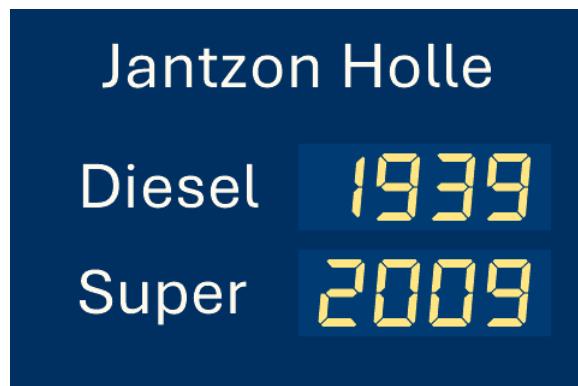
1.3.1 Grundfunktionen

Das ist schnell zusammengefasst:

- Webradio-Sender spielen



- Benzinpreise (Diesel und Super E5) anzeigen und überwachen



- Zeit anzeigen

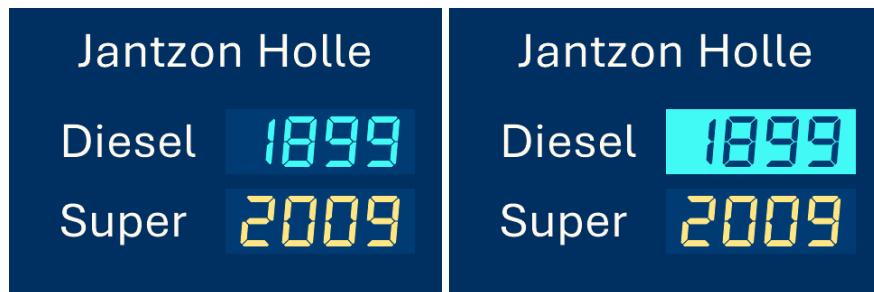


1.3.2 Weitere Funktionen

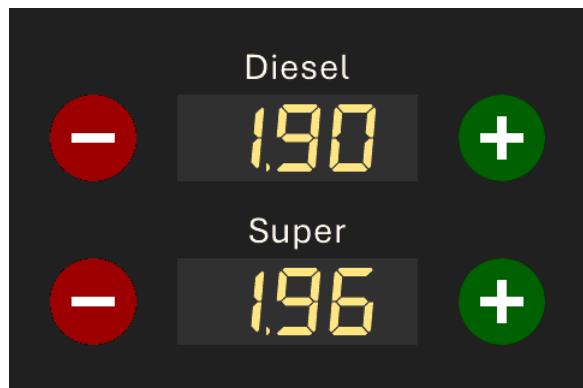
Der Radio-Bildschirm hat unten fünf Stationstasten zur Direktanwahl des gewünschten Senders. Diese werden in der Initialisierung aus der Senderliste vorbelegt.



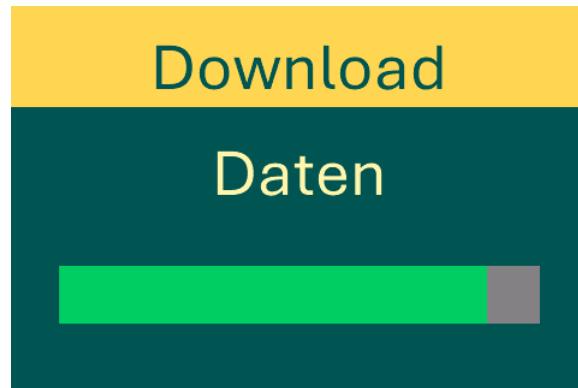
Die Benzinpreisanzeige enthält auch eine Überwachung auf Unterschreiten von Schwellwerten. Falls der Fall eintritt, wird automatisch auf die Benzinpreisanzeige gewechselt und der Preis blinkt. Die Ausgabe des Radiosenders wird unterbrochen, ein Warnton wird ausgegeben und die Tankstelle sowie der neue Preis werden vorgelesen. Danach wird Radioaudio wieder aktiviert.



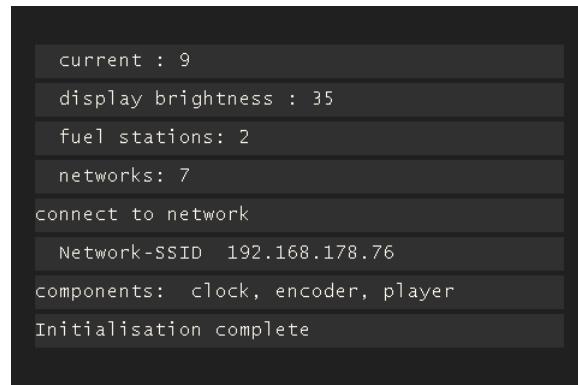
Die Benzinpreis-Schwellwerte können bei Bedarf angepaßt werden:



Zwei weitere Spielereien sind noch eingebaut: Wenn über WLAN ein neues Programm hochgeladen wird (OTA) oder das Filesystem des ESP mit Daten versorgt wird gibt es eine Downloadanzeige:



Während der Entwicklungszeit hat sich eine Art »Debug«-Bildschirm als hilfreich erwiesen, der wird auch beim normalen Hochfahren des Radios zum Anzeigen von Informationen verwendet:



1.3.3 Encoder

Der Dreh-Encoder hat kontext-abhängig unterschiedliche Funktion:

Radio Wechselt zum nächsten bzw. vorherigen Sender aus der Senderliste⁶. Damit können auch die Sender ohne Stationstaste erreicht werden.

Benzinpreise Wenn mehrere Tankstellen in der Überwachung sind, kann man sich vor- und rückwärts durch die Liste bewegen.

alle Ein-/Aus-Funktion: Druck auf den Encoder schaltet die Audioausgabe ab, wechselt auf die Uhr und dimmt den Bildschirm herunter. Erneuter Druck schaltet wieder ein.

⁶Diese wird unten im Abschnitt 3.3 beschrieben

1.3.4 Touchfunktionen

Auf allen Bildschirmen können Aktionen durch Touch ausgelöst werden. Der Bereich der Radiosender-Stationstasten hat nur im Radiobildschirm eine Funktion.

oben links Helligkeitseinstellung Display: dunkler
Abbruch (Einstellungen Benzinpreis-Schwellwerte)

oben Mitte Update Nextion-Display (soweit Datei vorhanden)
Wechsel in den Einstellungsbildschirm (nur Benzinpreis)

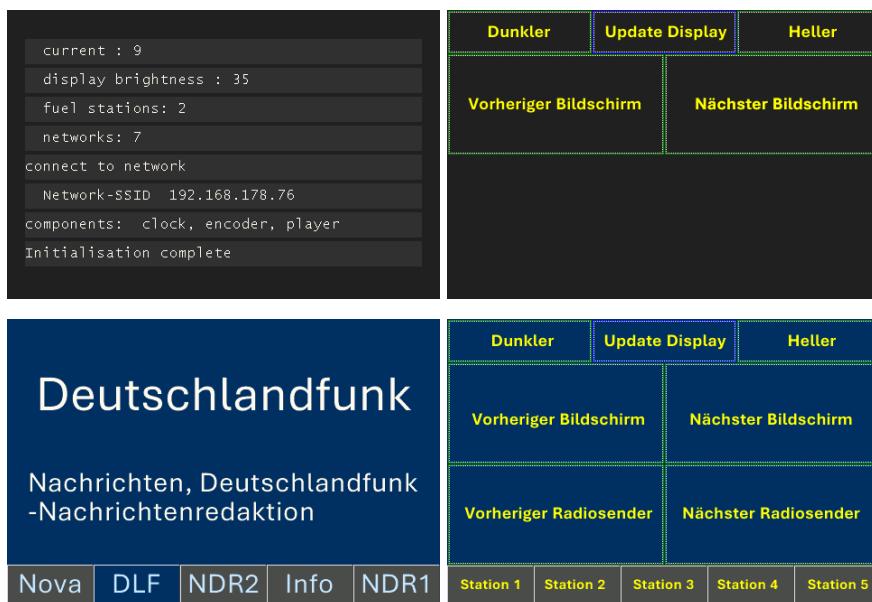
oben rechts Helligkeitseinstellung Display: heller
Speichern (Einstellungen Benzinpreis-Schwellwerte)

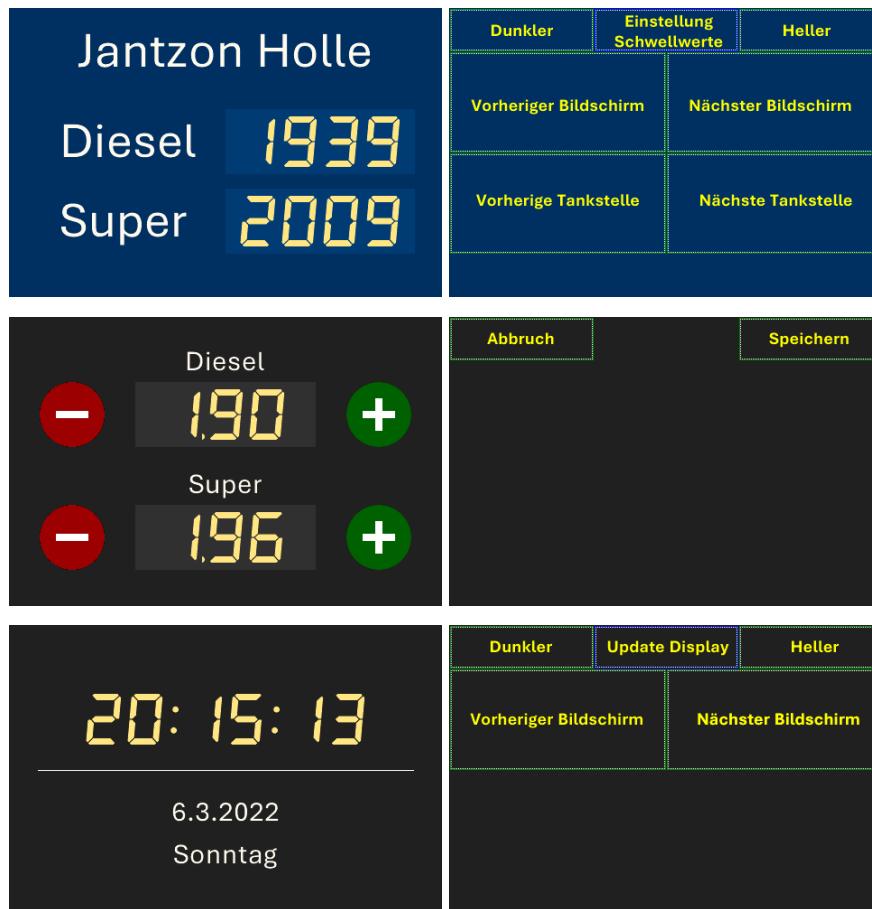
Mitte links Wechsel zum vorherigen Bildschirm
Radio → Debug → Uhr → Benzinpreis

Mitte rechts Wechsel zum nächsten Bildschirm
Radio → Benzinpreis → Uhr → Debug

Unten links Wechsel zum vorherigen Listeneintrag (nur Radio und Benzinpreis); entspricht Encoder-Drehung links

Unten rechts Wechsel zum nächsten Listeneintrag (nur Radio und Benzinpreis); entspricht Encoder-Drehung rechts





2 Installation

2.1 ArduinolDE und ESP32

Die Installation⁷ der Arduino-IDE und der Erweiterungen für die Espressif CPUs ESP8266 und (in diesem Fall) ESP32 ist im Netz an vielen Stellen ausführlich dokumentiert. Deshalb an dieser Stelle nur die Verweise auf die verwendeten Quellen:

- Arduino-IDE Download: <https://www.arduino.cc/en/software>
- Arduino-IDE Installation: <http://docs.arduino.cc/software/ide-v1/tutorials/PortableIDE>⁸

⁷Hier wird alles nur für Windows 10 beschrieben.

⁸Die portable Installation erlaubt die Trennung zwischen »echten« Arduino- und den ESP-Projekten.

- ESP32: <https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/>

Eine kleine Eigenheit ist noch zu erwähnen: Ich mag die öffnenden Klammern am Ende der Zeile nicht. »Allman-Style« lässt sich erreichen, wenn `formatter.conf` von `arduino-1.8.19\lib\` nach `arduino-1.8.19\portable\` kopiert wird und in der Kopie eine Zeile eingefügt wird:

```
9 style=allman
```

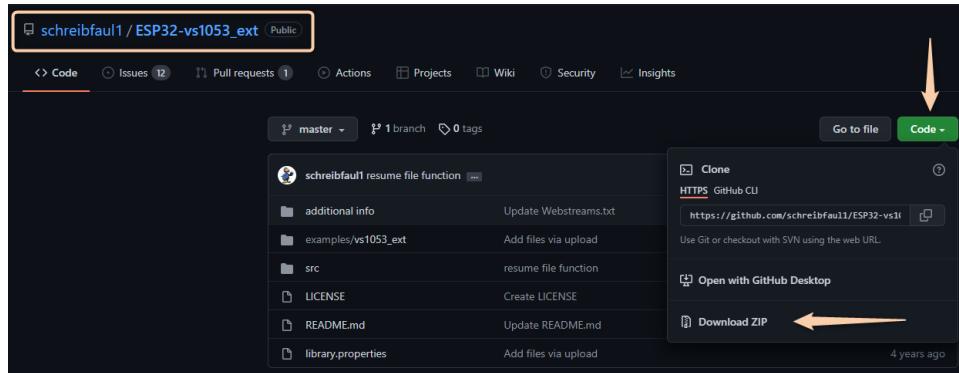
2.2 Bibliotheken und Tools

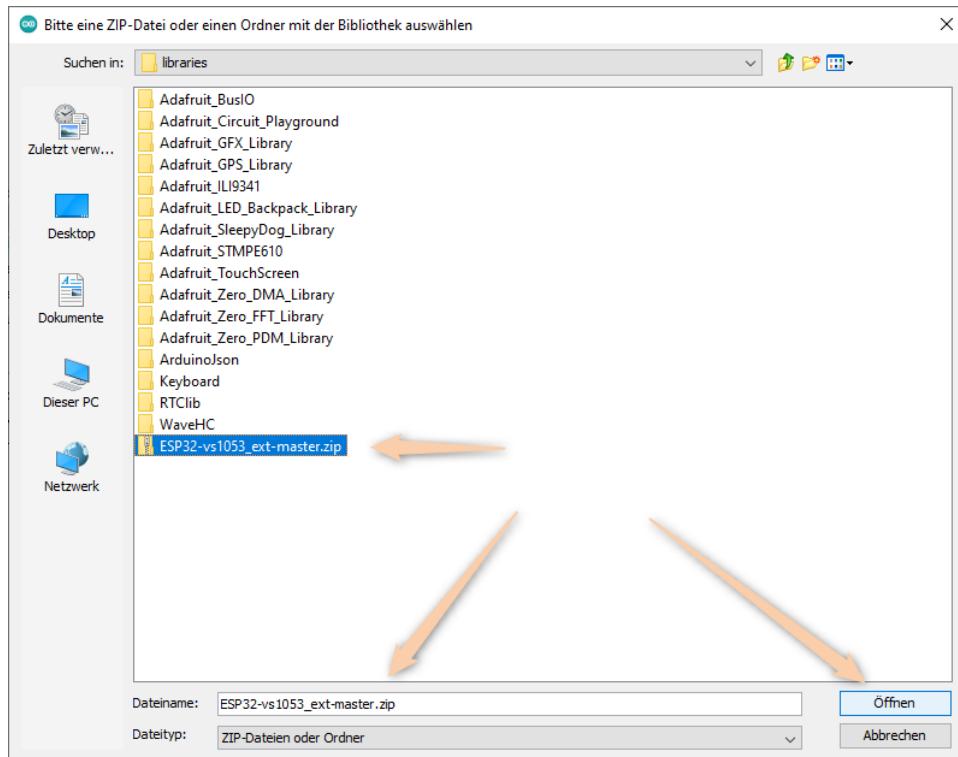
Ein solches Projekt lässt sich m.E. ohne Verwendung externer Bibliotheken nicht verwirklichen. Manche Sachen können andere einfach viel besser, das Rad ist auch schon erfunden und ich konnte mich auf die gewünschte Funktionalität konzentrieren.

2.2.1 VS1053

Die wichtigste Bibliothek ist die ESP32-VS1053 von Wolle »schreibfaul1«: https://github.com/schreibfaul1/ESP32-vs1053_ext.

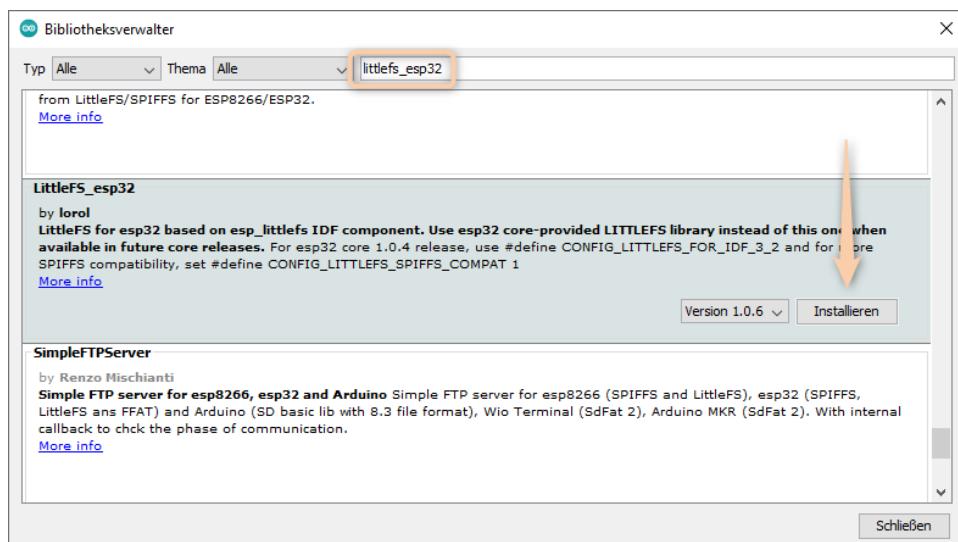
Von dort das Zip-File herunterladen und dann über Sketch → Bibliothek einbinden → .ZIP-Bibliothek hinzufügen... einfügen.





2.2.2 LITTLEFS

Das Dateisystem wird über die Bibliotheksverwaltung (`littlefs_esp32`) installiert. Ab ESP Core Version 2.0 soll es schon im Standardumfang enthalten sein⁹



⁹Quelle: <https://github.com/lorol/LITTLEFS>

2.2.3 File system uploader

Um Dateien in das Filesystem über den COM-Port oder »Over The Air« in das ESP32-Modul zu laden, braucht es eine Erweiterung der Arduino-IDE.

Datei: esp32fs.zip

Quelle: <https://github.com/lorol/arduino-esp32fs-plugin/releases>

Das darin enthaltene File esp32fs.jar kommt dann – bei der portablen IDE – in das Verzeichnis

arduino-1.8.19\portable\packages\esp32\hardware\esp32\1.0.6\tools

arduino-1.8.19 > portable > sketchbook > tools > ESP32FS > tool				
	Name	Änderungsdatum	Typ	Größe
	esp32fs.jar	02.02.2021 17:40	Executable Jar File	10 KB

Die Installation für die normale (nicht portable) IDE ist im Abschnitt »Installation« im Plugin-Verzeichnis <https://github.com/lorol/arduino-esp32fs-plugin/> beschrieben.

2.2.4 mklittlefs

Damit der File System Uploader seine Aufgabe erfüllen kann, wird noch das Tool mklittlefs benötigt.

Datei: x86_64-w64-mingw32-mklittlefs-295fe9b.zip

Quelle: <https://github.com/earlephilhower/mklittlefs/releases>

Das Archiv enthält das Programm mklittlefs.exe zur Ablage in

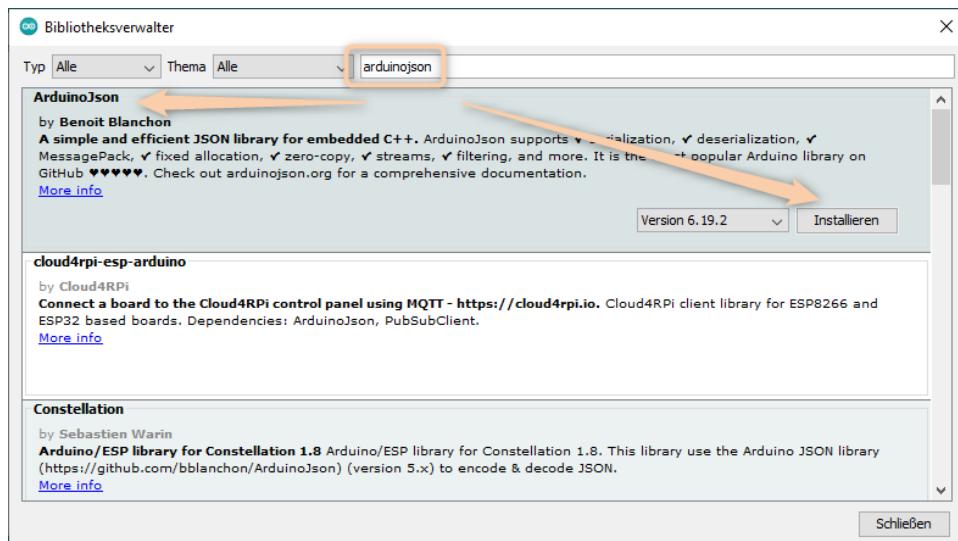
arduino-1.8.19\portable\sketchbook\tools\ESP32FS\tool

arduino-1.8.19 > portable > packages > esp32 > hardware > esp32 > 1.0.6 > tools >				
	Name	Änderungsdatum	Typ	Größe
	partitions	26.03.2021 12:26	Dateiordner	
	sdk	26.03.2021 12:26	Dateiordner	
	espota.exe	26.03.2021 12:26	Anwendung	3.936 KB
	espota.py	26.03.2021 12:26	Python File	10 KB
	esptool.py	26.03.2021 12:26	Python File	155 KB
	gen_esp32part.exe	26.03.2021 12:26	Anwendung	3.262 KB
	gen_esp32part.py	26.03.2021 12:26	Python File	20 KB
	mklittlefs.exe	16.08.2020 21:50	Anwendung	960 KB
	platformio-build.py	26.03.2021 12:26	Python File	11 KB

Wie üblich sollte nach der Installation des Plugins und des Tools die Arduino IDE einmal neu gestartet werden.

2.2.5 ArduinoJson

Die Benzinpreise, die über die Tankerkönig-API bezogen werden, kommen als JSON-Dokument. Als Decoder wird die Bibliothek von Benoît Blanchon verwendet. Installation erfolgt über die Bibliotheksverwaltung.



3 Konfiguration

Das Programm muß wegen der Tankerkönig-Integration sowieso einen JSON-Decoder enthalten, deshalb werden auch die Grundfunktionen durch JSON-Konfigurationsdateien gesteuert. Das ist ein wesentlicher Unterschied im Vergleich zu Edzelfs ESP32-Radio. Es hat außerdem den Vorteil, dass man diese bequem am PC vorbereiten und mit dem Filesystem-Tool dann gesammelt hochladen kann.

3.1 Tankerkönig-API

Für die Benzinpreisfunktion steht unter

<https://creativecommons.tankerkoenig.de/>

die Möglichkeit zur Verfügung, für ausgewählte Tankstellen Preisinformationen abzufragen. Von den drei zur Verfügung stehenden Methoden (Umkreissuche, Preisabfrage, Detailabfrage) verwendet dieses Projekt die Preisabfrage. Damit können Preise für Diesel, Super E5 und Super

E10 von bis zu 10 Tankstellen abgefragt werden. Andere Sorten und Gas stehen nicht zur Verfügung, weil diese nicht an die Markttransparenzstelle¹⁰ gemeldet werden müssen.

Zur Abfrage benötigt man einen API-Key. Der wird vollkommen problemlos nach Eingabe einer Mailadresse und des Verwendungszwecks (z.B. »Magic Mirror«) zugewieilt.

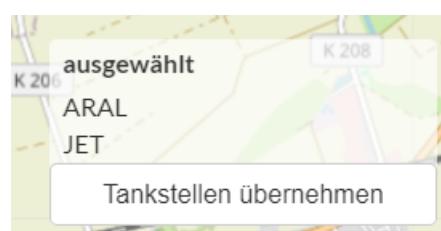
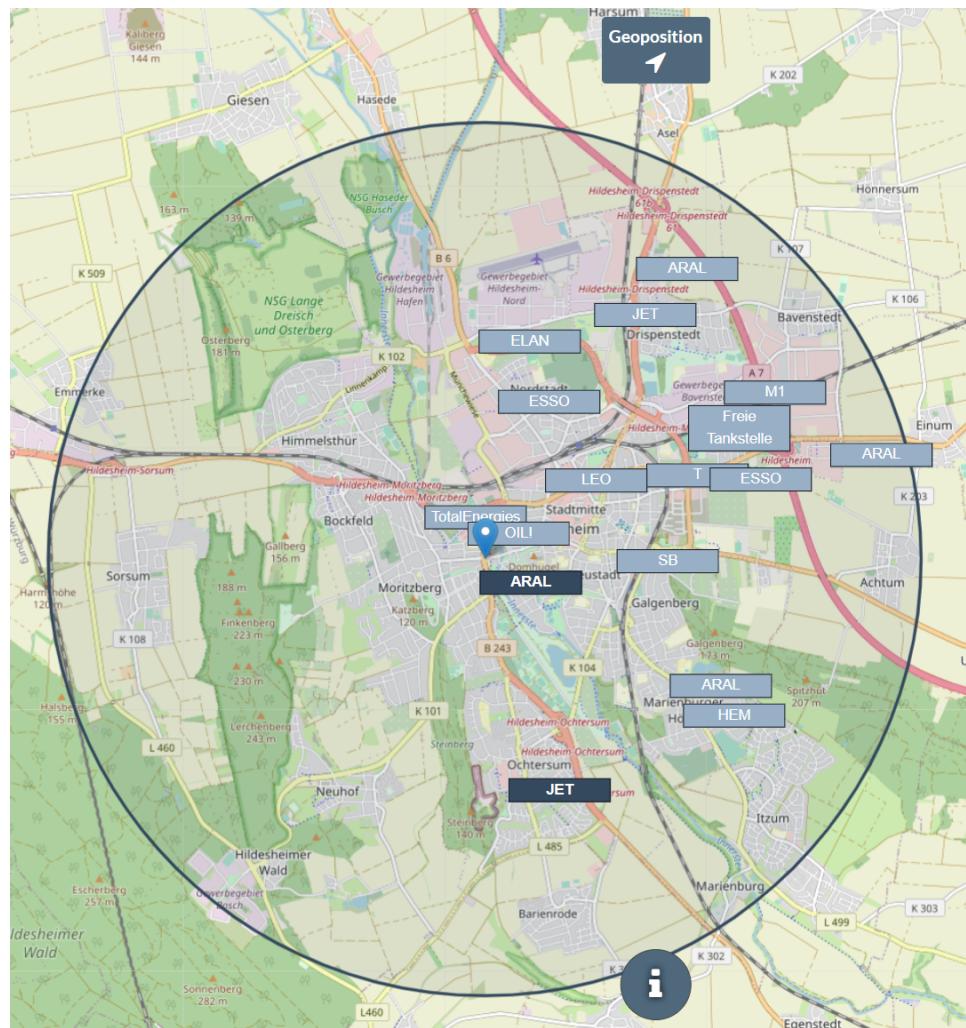
The screenshot shows a dark-themed web interface for Tankerkönig. At the top, there's a navigation bar with links: HOME, INFO, TERMS, API-KEY (which is highlighted in blue), HISTORY, PRO, REFERENZEN, and TOOLS. Below the navigation, a message says "API-Key vergessen?". Underneath it, a link says "Hier neu zuschicken lassen". The main content area has a heading "API-Key beantragen". It contains fields for "E-Mail:" (with "meine@mailadresse.de" entered) and "Verwendung:" (with "MagicMirror" selected). A note next to the "Verwendung:" field states: "SmartMirror-artiges Gerät auf Mikrocontrollerbasis (ESP32); Nicht öffentlich. Falls die App / die Website / der Dienst öffentlich erreichbar ist / wird, lässt es uns wissen und schickt uns einen Link zur Website, App-Store oder was auch immer." There's also a checkbox for "Nutzungsbedingungen akzeptiert" (with the note "Bitte bestätigen, dass die Nutzungsbedingungen und die Erklärung zum Datenschutz gelesen und akzeptiert wurden.") and a "Senden" button at the bottom.

Was nun noch fehlt, ist die Identifizierung der abzufragenden Tankstellen. Dazu gibt es bei Tankerkönig ein schönes, wunderbar einfach zu bedienendes Werkzeug, zu finden unter

<https://creativecommons.tankerkoenig.de/TankstellenFinder/index.html>

Mit Doppelklick in die Karte wird der Mittelpunkt des Suchkreises festgelegt (hier Hildesheim Dammtor), aus den Tankstellen kann man die gewünschten auswählen und mit »übernehmen« bekommt man die nötigen Informationen als JSON angezeigt. Die "id" ist das, was in die Konfigurationsdatei (s. 3.4) als "key" eingetragen werden soll.

¹⁰https://www.bundeskartellamt.de/DE/Wirtschaftsbereiche/Mineral%C3%B6l/MTS-Kraftstoffe/mtskraftstoffe_node.html



```

1 [
2 {
3   "id": "1alec4ba-cc2a-4663-8330-81efc48b9256",
4   "name": "Aral Tankstelle",
5   "brand": "ARAL",
6   "street": "Alfelder Straße",

```

```

7     "house_number": "99",
8     "post_code": 31139,
9     "place": "Hildesheim",
10    "lat": 52.1472321,
11    "lng": 9.938522,
12    "isOpen": true
13  },
14  {
15    "id": "51d4b5e1-a095-1aa0-e100-80009459e03a",
16    "name": "JET HILDESHEIM REX-BRAUNS-STR. 2",
17    "brand": "JET",
18    "street": "REX-BRAUNS-STR.",
19    "house_number": "2",
20    "post_code": 31139,
21    "place": "HILDESHEIM",
22    "lat": 52.1258,
23    "lng": 9.94333,
24    "isOpen": true
25  }
26 ]

```

3.2 Netzwerk

Die Konfigurationsdatei für die Netzwerke-Informationen heißt `network.json`¹¹. Sie hat folgendes Format:

```

1  {
2   "NetworkList": [
3     {
4       "ssid": "<YOUR NETWORK SSID>",
5       "password": "<YOUR NETWORK PASSWORD>"
6     }
7   ]
8 }

```

In Zeile 4 die SSID des Netzwerks, in Zeile 5 das entsprechende Zugangspasswort einfügen.

Wenn wie in meinem Anwendungsfall mehrere Netzwerke zur Auswahl

¹¹Der Name ist einstellbar; siehe unten in 4.1

stehen oder das Gerät an verschiedenen Orten (zu Hause, Wochenendhaus, Schrebergarten) betrieben werden soll, kann die Liste auch erweitert werden:

```
1  {
2      "NetworkList": [
3          {
4              "ssid": "<YOUR NETWORK SSID>",
5              "password": "<YOUR NETWORK PASSWORD>"
6          },
7          {
8              "ssid": "<other network>",
9              "password": "<other password>"
10         }
11     ]
12 }
```

Das Programm sucht alle verfügbaren Netzwerke und verbindet sich dann mit demjenigen aus der Konfigurationsliste, das die höchste Empfangsstärke hat.

3.3 Radiostationen

Hier heißt die Konfigurationsdatei `stations.json`; sie soll die Liste der insgesamt auswählbaren Radiostationen enthalten. Zusätzlich wird darin auch die Zuweisung auf die Stationstasten festgelegt.

```
1  {
2      "name": "Deutschlandfunk",
3      "key": "DLF",
4      "mem": 2,
5      "url": "st01.dlf.de/dlf/01/128/mp3/stream.mp3"
6  }
```

Da die Webradios manchmal recht merkwürdige und lange Sendernamen mitteilen (der NDR ist mir da unangenehm aufgefallen), ist der Stationsname, der im Radiobildschirm angezeigt wird, hier auch einstellbar.

name Stationsname

key Kurzname für die Stationstaste (maximal vier Buchstaben)

mem Nummer der Stationstaste (1...5 bzw. 0 wenn nicht benutzt)

url Stream-URL des Senders

Sender-URLs findet man auf verschiedenen Übersichts-Webseiten (z.B. <https://www.radio-browser.info/>) bzw. mehr oder weniger gut versteckt auf den Homepages der Senderketten:

- <https://www.deutschlandradio.de/streamingdienste.3236.de.html>
- <https://www.ndr.de/service/Die-Radio-Livestream-Links/livestreams101.html>
- <https://www.br.de/service/m3u-livestreams-100.html> (im PDF)

Eine Senderliste ist dann ein Array aus Stationen - etliche sind der Übersichtlichkeit halber ausgelassen:

```
1  {
2      "DefaultStationIndex": 3,
3      "StationList": [
4          {
5              "name": "NDR 1 NDS",
6              "key": "NDRI",
7              "mem": 5,
8              "url": "www.ndr.de/resources/metadaten/audio/m3u/ndr1niedersachsen.m3u"
9          },
10         ...
11         {
12             "name": "NDR Blue",
13             "key": "Blue",
14             "mem": 0,
15             "url": "www.ndr.de/resources/metadaten/audio/m3u/ndrblue.m3u"
16         },
17         {
18             "name": "Deutschlandfunk",
19             "key": "DLF",
20             "mem": 2,
21             "url": "st01.dlf.de/dlf/01/128/mp3/stream.mp3"
22         },
23         ...
24         {
25             "name": "Jazz Gospel",
26             "key": "J GO",
```

```

27      "mem": 0,
28      "url": "jazz-wr07.ice.infomaniak.ch/jazz-wr07-64.aac"
29    }
30  ]
31 }
```

Der Eintrag DefaultStationIndex ganz am Anfang ist nur für die allererste Inbetriebnahme nötig, wenn noch kein Wert für den zuletzt gespielten Sender im Speicher abgelegt ist. Der für die JSON-Dekodierung im Programm reservierte Speicherplatz sollte für eine Liste mit bis zu 20 Einträgen ausreichen.

3.4 Benzinpreise

tanken.json ist die dritte erforderliche Konfigurationsdatei. Sie enthält den API-Key, um überhaupt von Tankerkönig Daten zu erhalten sowie die Liste der Tankstellen, deren Preise interessieren.

Eine einzelne Tankstelle benötigt diese Einträge:

```

1  {
2    "uname": "ARAL Hildesheim",
3    "spname": "Aral",
4    "spcity": "Hildesheim, Alfelder Straße",
5    "key": "1alec4ba-cc2a-4663-8330-81efc48b9256"
6 }
```

Dabei bedeuten:

uname analog zum Radio: Stationsname der angezeigt wird

spname Name für die Sprachausgabe (»speech name«)

spcity Stadt für die Sprachausgabe (»speech city«)

key Tankerkönig-Stations-Identifikation

Anmerkung zur Sprachausgabe: Das Programm lässt bei Unterschreiten einer Benzinpreisschwelle durch die VS1053-Bibliothek (bei Google) einen Satz von Text in Sprache umwandeln: »<Diesel|Super> bei <spname> in <spcity> jetzt <preis> Euro.«

```

1  {
```

```

2   "APIkey": "<YOUR TANKEKOENIG API KEY>",
3   "StationList": [
4     {
5       "uname": "Jantzon Holle",
6       "spname": "Jantzon",
7       "spcity": "Holle",
8       "key": "3c034790-3d2a-4093-8417-032a48cb9f25"
9     },
10    {
11      "uname": "HEM Groß Düngen",
12      "spname": "Hem",
13      "spcity": "Groß Düngen",
14      "key": "e1a15081-24c2-9107-e040-0b0a3dfe563c"
15    },
16    {
17      "uname": "Jet Hi Ochtersum",
18      "spname": "Jet",
19      "spcity": "Hildesheim Ochtersum",
20      "key": "51d4b5e1-a095-1aa0-e100-80009459e03a"
21    },
22    {
23      "uname": "ARAL Hildesheim",
24      "spname": "Aral",
25      "spcity": "Hildesheim, Alfelder Straße",
26      "key": "1a1ec4ba-cc2a-4663-8330-81efc48b9256"
27    }
28  ]
29 }
```

3.5 Gong

`gong.mp3` ist natürlich keine Konfigurationsdatei. Das ist der Soundschnipsel, der am Anfang einer Bezinpreismeldung abgespielt wird. Erzeugt wurde es mit SonicPi (<https://sonic-pi.net/>) und folgendem kurzen »Code«:

```

1  2.times do
2   use_synth :beep
3   play :C5, sustain: 0.7, release: 0.3
4   sleep 0.5
```

```
5   play :A4, sustain: 0.7, release: 0.3
6   sleep 2
7 end
```

4 Inbetriebnahme

4.1 Einstellungen

Im Programmcode gibt es die Datei config.h. Darin sind alle Einstellungen versammelt, an denen wegen unterschiedlicher Hardware oder anderen Präferenzen, was das Verhalten des Systems angeht, geschraubt werden kann.

Namen der Konfigurations-Dateien aus dem LITTLEFS.

Die Datei nextionTftFile muss nicht zwingend mit in das Dateisystem aufgenommen werden. Das ist die HMI-Information für das Nextion-Display. Das kann bei laufendem System aus dem Filesystem heraus aktualisiert werden; danach wird die Datei entfernt um unbeabsichtigtes Flashen des Displays zu vermeiden.

```
7 // files
8 constexpr char radioStationsFile[] = "/stations.json";
9 constexpr char networkCredentialsFile[] = "/network.json";
10 constexpr char fuelDataFile[] = "/tanken.json";
11 constexpr char nextionTftFile[] = "/radio.tft";
12 constexpr char gongFile[] = "/gong.mp3";
```

Im nächsten Abschnitt wird die Größe der Puffer für die JSON-Dekodierung festgelegt. Hier lohnt es sich, vom Umfang stark abweichende Stations- oder Tankstellenlisten vorher in den ArduinoJson-Assistenten¹² zu kippen und in der Konfiguration dann mindestens die dort empfohlene Größe einzustellen. Ich war allerdings schon großzügig; an Speicher mangel leidet der ESP32 ja nicht so sehr.

```
21 // json decoding
22 constexpr uint32_t jsonRadioStationListDocSize = 2048;
23 constexpr uint32_t jsonNetworkListDocSize = 256;
24 constexpr uint32_t jsonFuelStationListDocSize = 1024;
```

¹²<https://arduinojson.org/v6/assistant/>

Die Zeit kommt von der hauseigenen Fritz!Box und wird so etwa alle zwei Minuten aktualisiert. Die meisten Home-Router stellen einen ntp-Dienst bereit; falls die Verwendung eines öffentlichen Zeitservers erforderlich ist, sollte das Intervall deutlich verlängert werden.

```
30 // ntp server
31 constexpr char ntpServer[] = "fritz.box";
32 constexpr unsigned long ntpUpdateInterval = (123UL); // in seconds (~2min)
```

Die Hardware-Einstellungen finden sich gegen Ende der Datei:

```
52 // hardware
53 // MP3 board
54 constexpr uint8_t spiMOSI    = 23;
55 constexpr uint8_t spiMISO    = 19;
56 constexpr uint8_t spiSCK     = 18;
57 constexpr uint8_t vs1053CS   = 2;
58 constexpr uint8_t vs1053DCS  = 4;
59 constexpr uint8_t vs1053DREQ = 36;
60
61 // Nextion display
62 constexpr uint8_t nextionRXD = 16;
63 constexpr uint8_t nextionTXD = 17;
64
65 // Encoder
66 constexpr uint8_t encCLK    = 25; // Rotary encoder CLK
67 constexpr uint8_t encDT     = 26; // Rotary encoder DT
68 constexpr uint8_t encSW     = 27; // Rotary encoder SW
```

Ganz am Ende kommen noch die Einstellungen für die Benzinpreisüberwachung. Das Update-Intervall sollte nicht unter zehn Minuten liegen. Wann am Tage die Prüfung laufen soll ist einstellbar; ob auch im ausgeschalteten Zustand abgefragt werden soll und die Sprachausgabe sind abschaltbar. Die Variablennamen sind hoffentlich selbsterklärend.

Während der Entwicklungszeit sollte die Emulation der Abfrage (»Fake«) eingeschaltet werden um den Tankerkönig-Server zu entlasten.

```
72 // Tankerkoenig
73 constexpr unsigned long fuelUpdateInterval = 678UL; // in seconds (~11.3min)
74 constexpr size_t alarmTextLength = 256;
75 constexpr int32_t initialLimitDiesel   = 180;
76 constexpr int32_t initialLimitSuper   = 190;
```

```

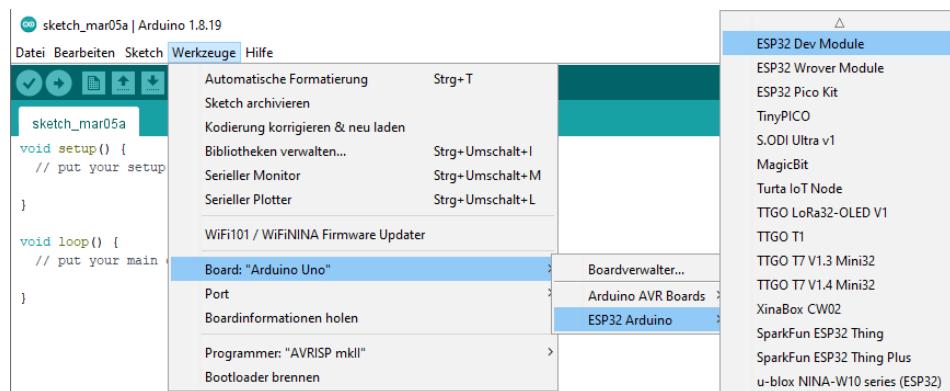
77  constexpr int32_t initialLimitSuperE10 = 187;
78  constexpr uint8_t fuelScanStartHour = 6; // starts at ">="
79  constexpr uint8_t fuelScanEndHour = 22; // ends before "<"
80  constexpr bool enableFuelPriceScanWhileOff = true;
81  constexpr bool enableSpeechOutput = true;
82  constexpr bool UseTankerkoenigFakeValues = false;

```

4.2 Bauen und Laden

Nun denn - wenn die Hardware bereit ist, kann es ans Bauen und Laden gehen.

Im Verzeichnis `radio` findet sich das Arduino-Projekt, wobei `radio.ino` das Hauptfile ist. Als Board habe ich das ESP32 DevModule eingestellt:



Wenn alles passt, endet Kompilieren und Linken – das dauert einen Moment – in etwa mit diesen Angaben:

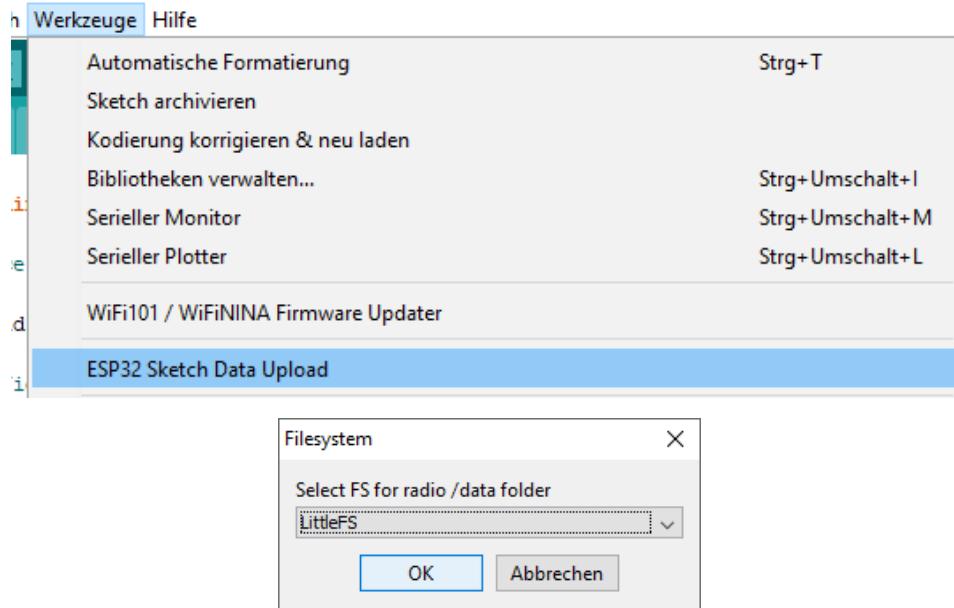
-
- 1 Der Sketch verwendet 1034046 Bytes (78%) des Programmspeicherplatzes.
 - 2 Das Maximum sind 1310720 Bytes.
 - 3 Globale Variablen verwenden 46368 Bytes (14%) des dynamischen Speichers,
 - 4 281312 Bytes für lokale Variablen verbleiben. Das Maximum sind 327680 Bytes.
-

Initial sollte das Nextion-Display einmalig über SD-Karte oder aus dem Nextion-Editor über USB (serielle Schnittstelle unter Verwendung eines FTDI-Adapters¹³) mit der Datei `radio.tft` versorgt werden.

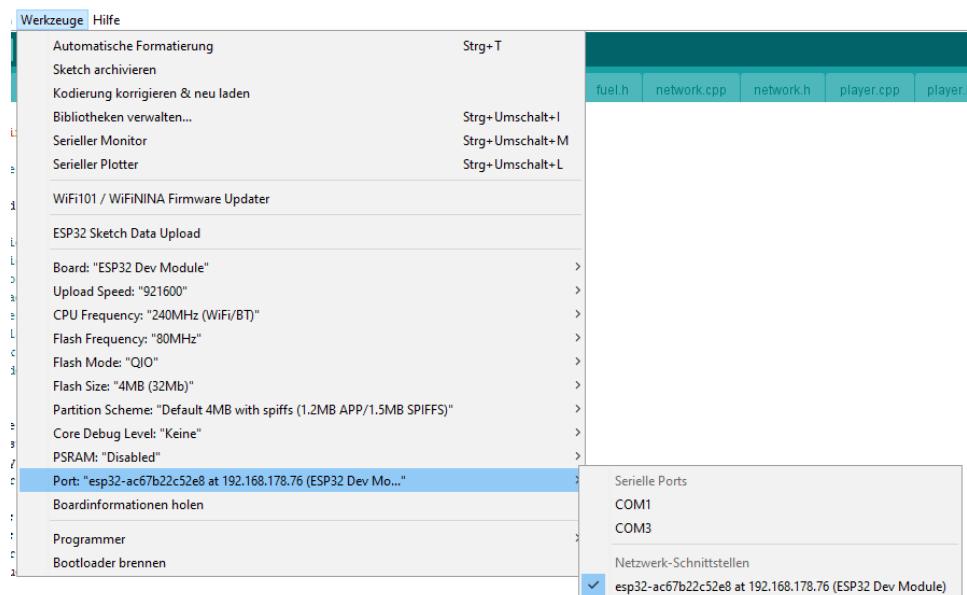
Die Inbetriebnahme sollte am Rechner über den USB-Port erfolgen. Das Dateisystem ist ja noch nicht mit den Konfigurationsdateien gefüllt, deshalb wird das Programm auch keine Verbindung zum Netzwerk aufbauen können. Wenn also irgendein Programm auf dem ESP läuft, sollte zunächst das Dateisystem geladen werden. Die hochzuladenden Dateien

¹³<https://www.az-delivery.de/products/ftdi-adapter-ft232rl>

müssen wegen des Uploaders im Unterverzeichnis `data` stehen.



Danach kann dann der Sketch hochgeladen werden und ab dann sollte in der Portliste das ESP-Modul auch über Funk erreichbar sein.



Wenn sich das Modul mit dem Netzwerk verbunden hat, können von nun an alle Uploads über WLAN durchgeführt werden. Solange das Gerät auch noch am USB hängt, kann mit dem seriellen Monitor oder über ein Terminalprogramm wie PuTTY genauer nachgesehen werden, was

gerade so abläuft (115200bps). Dazu muss in `trace.h` die zweite Zeile aktiv sein:

```
2 #define DEBUGGING
```

Für diesen Fall ist im Sketch auch vorgesehen, UI-Aktionen über einzelne Buchstaben zu triggern. Da kann man sich zum Ausprobieren beliebig austoben.

```
546 // TEST - simulate input devices
547 if (Serial.available())
548 {
549     byte value = Serial.read();
550     switch (value)
551     {
552         case '1':
553             ...
554 }
```

Viel Erfolg beim Nachbauen!