Internetradio mit ESP32

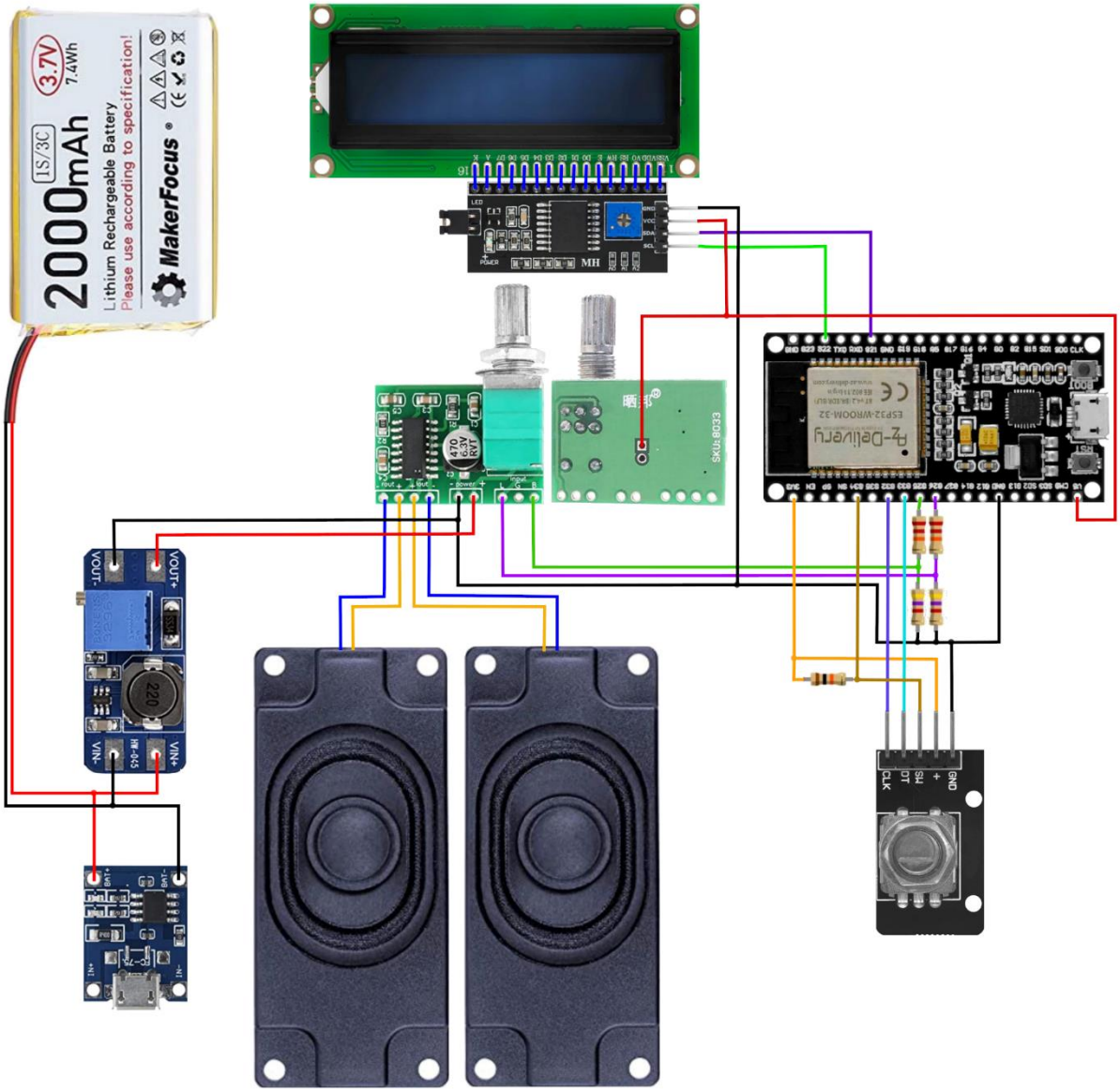
Viele Radiosender können als MP3-Stream über das Internet gehört werden. Da der Mikrocontroller ESP32 einerseits WLAN-Fähigkeiten besitzt und andererseits mit zwei eingebauten Digital/Analog-Wandlern den digitalen Datenstrom in ein Analogsignal umwandeln kann, bietet er sich für dieses Projekt als ideale Lösung an. Zusätzlich wird eine Akku-Stromversorgung, ein Audioverstärker, zwei Lautsprecher, ein Display zu Senderanzeige und ein Eingabegerät zur Sendereinstellung benötigt. Abgerundet wird das Ganze mit einem Gehäuse aus dem 3D-Drucker.

Benötigte Hardware

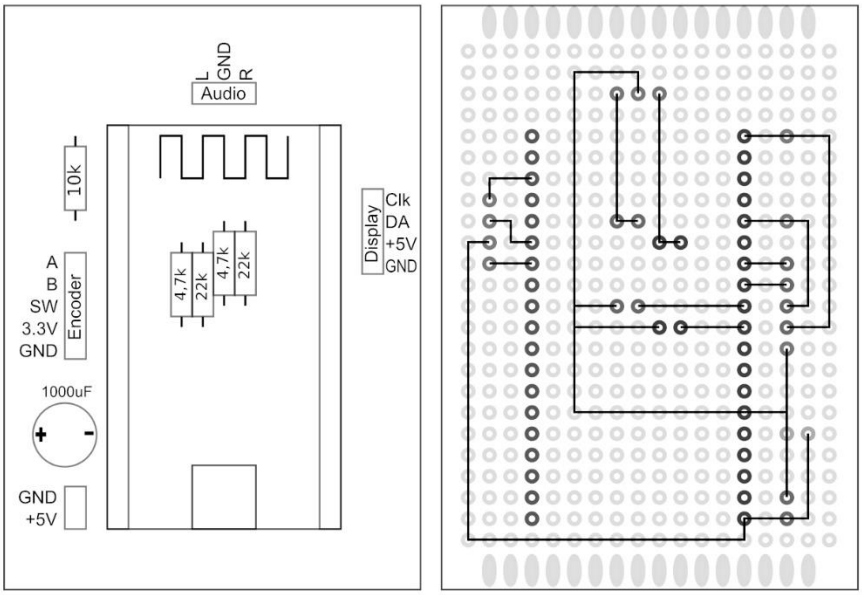
Anzahl	Bauteil	Anmerkung
1	ESP32 Development Board	
1	Audio-Verstärker 3W	
1	Lautsprecher	
2	Widerstände 4.7 kOhm	
2	Widerstände 22 kOhm	
1	Widerstand 10 kOhm	
1	Elko 1000uF / 10V	
1	3.7V Akku 2000mAh	
1	Laderegler	
1	DC-DC Step Up Converter	
1	LCD-Display mit I2C Interface	
1	Drehgeber Encoder	
1	Lochrasterplatte 50x70	
2	Federleisten 19-polig	
1	Stiftleiste 3-polig	
1	Stiftleiste 4-polig	

1	Stiftleiste 5-polig	
2	Drehknöpfe für 6mm Achse	
Mehrere	Jumperkabel weiblich zu weiblich	
1	Gehäuse Frontteil aus dem 3D-Drucker	
1	Gehäuse Rückseite aus dem 3D-Drucker	
1	Deckel zur Akku-Sicherung aus dem 3D-Drucker	

Schaltung



Der ESP32 wird mit den Widerständen und Stiftleisten für die Peripherie auf einer 50x70 mm großen Lochrasterplatte aufgebaut.



Die Abbildung zeigt die Bestückung und die Verdrahtung auf der Unterseite

Verdrahtung

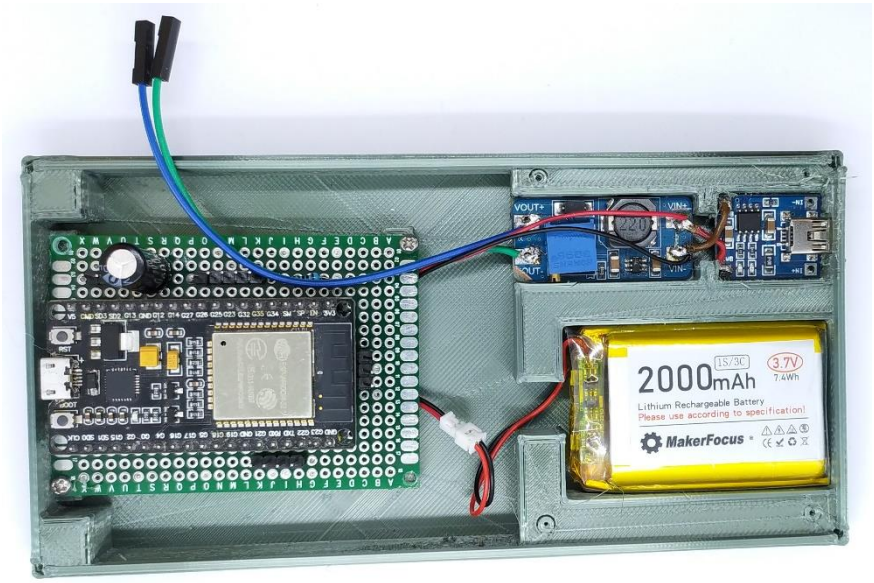
Als erstes wird der Batterieanschluss des Ladereglers mit dem Eingang des DC-DC Step Up Wandlers verbunden. Polung beachten! Der Akku wird über einen geeigneten Steckverbinder ebenfalls mit dem Batterie-Eingang des Ladereglers verlötet. Nun sollte die Ausgangsspannung des Wandlers mit dem blauen Potentiometer auf ca. 5.2 V eingestellt werden. Dazu muss entweder ein Akku angeschlossen oder der USB-Eingang des Ladereglers mit einem USB-Netzgerät verbunden werden.

Wenn die Spannung eingestellt ist, kann der Ausgang des Wandlers mit dem Versorgungseingang des Audioverstärkers verbunden werden. Auf der Rückseite des Verstärkers sind zwei Lötpunkte, wobei der Plus Anschluss über den Schalter des Lautstärkepotentiometer geschaltet wird.

Dieser Anschluss wird zur Versorgung des ESP32 und des Displays verwendet, damit man über das Potentiometer, das Gerät komplett ausschalten kann.
Nun können die Verbindungen zur Steuerplatine auf der Lochrasterplatte hergestellt werden. Am besten werden dazu Jumper-Drähte mit zwei weiblichen Steckern verwendet. Man benötigt eine 3-polige Verbindung vom Audio-Ausgang zum Verstärker, eine 4-polige zum Display und eine 5-polige zum Rotary-Encoder.

Wichtiger Hinweis!
Die Lautsprecher sollten nicht im eingeschalteten Zustand an- oder abgesteckt werden, da induktiver Spannungsspitzen die Verstärkerausgänge zerstören könnten.

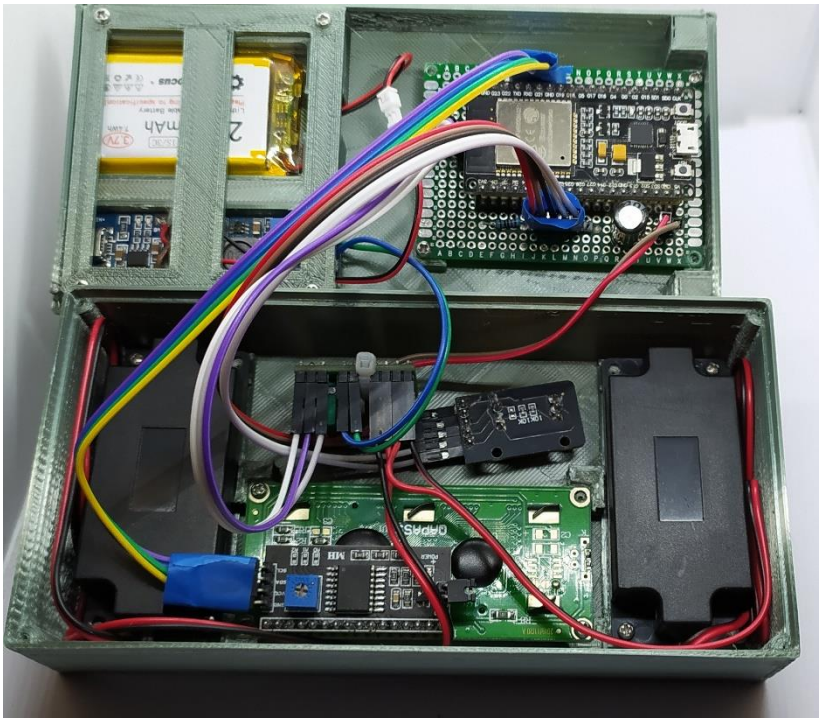
Wird das im Blog-Beitrag vorgestellte Gehäuse verwendet, kommt Akku, Laderegler, DC/DC-Wandler und die Steuerplatine auf die Backplane. Die Lautsprecher, der Verstärker, der Rotary-Encoder und das Display kommen auf die Frontplane. Der Deckel wird verwendet, um den Akku zu sichern.



Software

Damit der Sketch kompiliert werden kann, muss die Arduino IDE entsprechend vorbereitet werden. Die Arduino IDE unterstützt standardmäßig eine große Anzahl von Boards mit unterschiedlichen Mikrocontrollern, nicht aber den ESP32. Damit man Programme für diese Controller erstellen und hochladen kann, muss daher je ein Softwarepaket für die Unterstützung installiert werden.

Zuerst müssen Sie der Arduino-IDE mitteilen, wo sie die zusätzlich benötigten Daten findet. Dazu öffnen Sie im Menü Datei den Punkt Voreinstellungen. Im Voreinstellungs-Fenster gibt es das Eingabefeld mit der Bezeichnung „Zusätzliche Boardverwalter URLs“. Wenn Sie auf das Ikon rechts neben dem Eingabefeld klicken, öffnet sich ein Fenster in dem Sie die URL https://dl.espressif.com/dl/package_esp32_index.json für den ESP32 eingeben können.



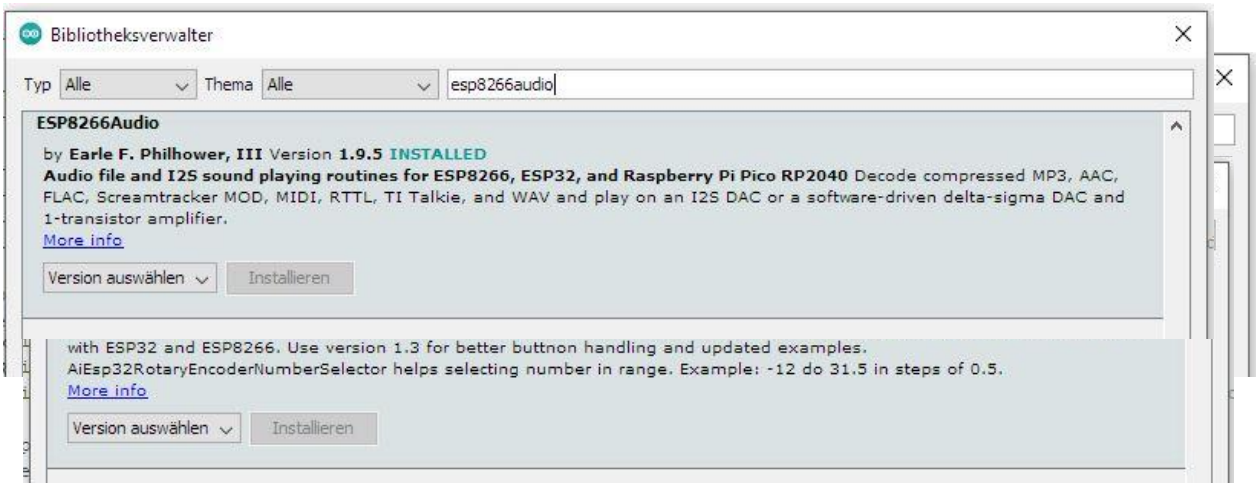
Nun wählen Sie in der Arduino IDE unter Werkzeug → Board die Boardverwaltung.

Es öffnet sich ein Fenster, in dem alle zur Verfügung stehenden Pakete aufgelistet werden. Um die Liste einzugrenzen, gibt man im Suchfeld „esp32“ ein. Dann erhält man nur noch einen Eintrag in der Liste. Installieren Sie das Paket „esp32“.

Für das Display benötigen Sie eine Bibliothek, die über die Arduino Bibliotheksverwaltung installiert werden kann. Das ist die Bibliothek „LiquidCrystal I2C“.

Eine weitere Bibliothek wird für den Rotary-Encoder benötigt. Ihr Name ist „AiEsp32RotaryEncoder“.

Kernstück dieses Projekts ist aber die Bibliothek „ESP8266Audio“.



Diese Bibliothek ermöglicht es verschiedene digitale Eingangsströme zu lesen, zu dekodieren und über verschiedene Ausgangskanäle wiederzugeben. Als Eingang, kann der Programmspeicher, der interne RAM ein Filesystem, eine SD-Karte, ein HTTP-Stream oder ein ICY-Stream genutzt werden. Der ICY-Stream wird typisch von Internet-Radios genutzt.
Dekodiert werden können WAV, MOD, MIDI, FLAC, AAC und MP3 Dateien. Für das Webradio wird MP3 benötigt. Die Ausgabe kann schließlich in Speicher, Files oder I2S erfolgen. Eine Besonderheit gibt es für den ESP32. Der I2S Output kann auf den internen Digital-Analog-Wandler ausgegeben werden. An den Ausgangs-Pins des DAW (Pin 25 und Pin 26) steht dann ein analoges Stereosignal zur Verfügung. Dieses Feature wird im vorliegenden Projekt genutzt.

Wenn alle Bibliotheken installiert sind, kann der Sketch kompiliert und auf die Hardware hochgeladen werden. Achtung! Zum Hochladen muss der Jumper zwischen D0 und RST entfernt werden.

Der Sketch

```
#include <WiFi.h>
//Includes from ESP8266audio
#include "AudioFileSourceICYStream.h" //input stream
#include "AudioFileSourceBuffer.h"    //input buffer
#include "AudioGeneratorMP3.h"        //decoder
#include "AudioOutputI2S.h"           //output stream
//library for LCD display
#include <LiquidCrystal_I2C.h>
```

```
//library for rotary encoder
#include "AiEsp32RotaryEncoder.h"
//esp32 library to save preferences in flash
#include <Preferences.h>

//WLAN access fill with your credentials
#define SSID "************"
#define PSK "**************"

//used pins for rotary encoder
#define ROTARY_ENCODER_A_PIN 33
#define ROTARY_ENCODER_B_PIN 32
#define ROTARY_ENCODER_BUTTON_PIN 34
#define ROTARY_ENCODER_VCC_PIN -1 /* 27 put -1 of Rotary encoder Vcc is connected directly to
3,3V; else you can use declared output pin for powering rotary encoder */

//depending on your encoder - try 1,2 or 4 to get expected behaviour
//#define ROTARY_ENCODER_STEPS 1
//#define ROTARY_ENCODER_STEPS 2
#define ROTARY_ENCODER_STEPS 4

//structure for station list
typedef struct {
  char * url;  //stream url
  char * name; //stations name
} Station;

#define STATIONS 24 //number of stations in tzhe list

//station list can easily be modified to support other stations
Station stationlist[STATIONS] PROGMEM = {
{"http://icecast.ndr.de/ndr/ndr2/niedersachsen/mp3/128/stream.mp3","NDR2 Niedersachsen"},
{"http://icecast.ndr.de/ndr/ndr1niedersachsen/hannover/mp3/128/stream.mp3","NDR1
Hannover"},
{"http://wdr-1live-live.icecast.wdr.de/wdr/1live/live/mp3/128/stream.mp3","WDR1"},
{"http://wdr-cosmo-live.icecast.wdr.de/wdr/cosmo/live/mp3/128/stream.mp3","WDR COSMO"},
{"http://radiohagen.cast.addradio.de/radiohagen/simulcast/high/stream.mp3","Radio Hagen"},
{"http://st01.sslstream.dlf.de/dlf/01/128/mp3/stream.mp3","Deutschlandfunk"},
{"http://dispatcher.rndfnk.com/br/br1/franken/mp3/low","Bayern1"},
{"http://dispatcher.rndfnk.com/br/br3/live/mp3/low","Bayern3"},
{"http://dispatcher.rndfnk.com/hr/hr3/live/mp3/48/stream.mp3","Hessen3"},
{"http://stream.antenne.de/antenne","Antenne Bayern"},
{"http://stream.1a-webradio.de/saw-deutsch/","Radio 1A Deutsche Hits"},
{"http://stream.1a-webradio.de/saw-rock/","Radio 1A Rock"},
{"http://streams.80s80s.de/ndw/mp3-192/streams.80s80s.de/","Neue Deutsche Welle"},
{"http://dispatcher.rndfnk.com/br/brklassik/live/mp3/low","Bayern Klassik"},
```

```
{"http://mdr-284280-1.cast.mdr.de/mdr/284280/1/mp3/low/stream.mp3","MDR"},
{"http://icecast.ndr.de/ndr/njoy/live/mp3/128/stream.mp3","N-JOY"},
{"http://dispatcher.rndfnk.com/rbb/rbb888/live/mp3/mid","RBB"},
{"http://dispatcher.rndfnk.com/rbb/antennebrandenburg/live/mp3/mid","Antenne Brandenburg"},
{"http://wdr-wdr3-live.icecastssl.wdr.de/wdr/wdr3/live/mp3/128/stream.mp3","WDR3"},
{"http://wdr-wdr2-
aachenundregion.icecastssl.wdr.de/wdr/wdr2/aachenundregion/mp3/128/stream.mp3","WDR 2"},
{"http://rnrw.cast.addradio.de/rnrw-0182/deinschlager/low/stream.mp3","NRW Schlagerradio"},
{"http://rnrw.cast.addradio.de/rnrw-0182/deinrock/low/stream.mp3","NRW Rockradio"},
{"http://rnrw.cast.addradio.de/rnrw-0182/dein90er/low/stream.mp3","NRW 90er"},
{"http://mp3.hitradiort1.c.nmdn.net/rt1rockwl/livestream.mp3","RT1 Rock"}};

//buffer size for stream buffering
const int preallocateBufferSize = 80*1024;
const int preallocateCodecSize = 29192;        // MP3 codec max mem needed
//pointer to preallocated memory
void *preallocateBuffer = NULL;
void *preallocateCodec = NULL;

//instance of prefernces
Preferences pref;
//instance for rotary encoder
AiEsp32RotaryEncoder rotaryEncoder = AiEsp32RotaryEncoder(ROTARY_ENCODER_A_PIN,
ROTARY_ENCODER_B_PIN, ROTARY_ENCODER_BUTTON_PIN, ROTARY_ENCODER_VCC_PIN,
ROTARY_ENCODER_STEPS);
//instance for LCD display
LiquidCrystal_I2C lcd(0x27,16,2);  // set the LCD address to 0x27 for a 16 chars and 2 line display
//instances for audio components
AudioGenerator *decoder = NULL;
AudioFileSourceICYStream *file = NULL;
AudioFileSourceBuffer *buff = NULL;
AudioOutputI2S *out;

//Special character to show a speaker icon for current station
uint8_t speaker[8]  = {0x3,0x5,0x19,0x11,0x19,0x5,0x3};
//global variables
uint8_t curStation = 0;   //index for current selected station in stationlist
uint8_t actStation = 0;   //index for current station in station list used for streaming
uint32_t lastchange = 0;  //time of last selection change

//callback function will be called if meta data were found in input stream
void MDCallback(void *cbData, const char *type, bool isUnicode, const char *string)
{
  const char *ptr = reinterpret_cast<const char *>(cbData);
  (void) isUnicode; // Punt this ball for now
  // Note that the type and string may be in PROGMEM, so copy them to RAM for printf
```

```
  char s1[32], s2[64];
  strncpy_P(s1, type, sizeof(s1));
  s1[sizeof(s1)-1]=0;
  strncpy_P(s2, string, sizeof(s2));
  s2[sizeof(s2)-1]=0;
  Serial.printf("METADATA(%s) '%s' = '%s'\n", ptr, s1, s2);
  Serial.flush();
}

//stop playing the input stream release memory, delete instances
void stopPlaying() {
  if (decoder)  {
    decoder->stop();
    delete decoder;
    decoder = NULL;
  }
  if (buff)  {
    buff->close();
    delete buff;
    buff = NULL;
  }
  if (file)  {
    file->close();
    delete file;
    file = NULL;
  }
}

//start playing a stream from current active station
void startUrl() {
  stopPlaying();  //first close existing streams
  //open input file for selected url
  Serial.printf("Active station %s\n",stationlist[actStation].url);
  file = new AudioFileSourceICYStream(stationlist[actStation].url);
  //register callback for meta data
  file->RegisterMetadataCB(MDCallback, NULL);
  //create a new buffer which uses the preallocated memory
  buff = new AudioFileSourceBuffer(file, preallocateBuffer, preallocateBufferSize);
  Serial.printf_P(PSTR("sourcebuffer created - Free mem=%d\n"), ESP.getFreeHeap());
  //create and start a new decoder
  decoder = (AudioGenerator*) new AudioGeneratorMP3(preallocateCodec, preallocateCodecSize);
  Serial.printf_P(PSTR("created decoder\n"));
  Serial.printf_P("Decoder start...\n");
  decoder->begin(buff, out);
}
```

```
//show name of current station on LCD display
//show the speaker symbol in front if current station = active station
void showStation() {
 lcd.clear();
 if (curStation == actStation) {
  lcd.home();
  lcd.print(char(1));
 }
 lcd.setCursor(2,0);
 String name = String(stationlist[curStation].name);
 if (name.length() < 15)
  lcd.print(name);
 else {
  uint8_t p = name.lastIndexOf(" ",15); //if name does not fit, split line on space
  lcd.print(name.substring(0,p));
  lcd.setCursor(0,1);
  lcd.print(name.substring(p+1,p+17));
 }
}

//handle events from rotary encoder
void rotary_loop()
{
 //dont do anything unless value changed
 if (rotaryEncoder.encoderChanged())
 {
  uint16_t v = rotaryEncoder.readEncoder();
  Serial.printf("Station: %i\n",v);
  //set new currtent station and show its name
  if (v < STATIONS) {
   curStation = v;
   showStation();
   lastchange = millis();
  }
 }
 //if no change happened within 10s set active station as current station
 if ((lastchange > 0) && ((millis()-lastchange) > 10000)){
  curStation = actStation;
  lastchange = 0;
  showStation();
 }
 //react on rotary encoder switch
 if (rotaryEncoder.isEncoderButtonClicked())
 {
  //set current station as active station and start streaming
  actStation = curStation;
```

```
    Serial.printf("Active station %s\n",stationlist[actStation].name);
    pref.putUShort("station",curStation);
    startUrl();
    //call show station to display the speaker symbol
    showStation();
  }
}

//interrupt handling for rotary encoder
void IRAM_ATTR readEncoderISR()
{
  rotaryEncoder.readEncoder_ISR();
}

//setup
void setup() {
  Serial.begin(115200);
  delay(1000);
  //reserve buffer für for decoder and stream
  preallocateBuffer = malloc(preallocateBufferSize);         // Stream-file-buffer
  preallocateCodec = malloc(preallocateCodecSize);           // Decoder- buffer
  if (!preallocateBuffer || !preallocateCodec)
  {
    Serial.printf_P(PSTR("FATAL ERROR:  Unable to preallocate %d bytes for app\n"),
preallocateBufferSize+preallocateCodecSize);
    while(1){
      yield(); // Infinite halt
    }
  }
  //start rotary encoder instance
  rotaryEncoder.begin();
  rotaryEncoder.setup(readEncoderISR);
  rotaryEncoder.setBoundaries(0, STATIONS, true); //minValue, maxValue, circleValues true|false
(when max go to min and vice versa)
  rotaryEncoder.disableAcceleration();
  //init WiFi
  Serial.println("Connecting to WiFi");
  WiFi.disconnect();
  WiFi.softAPdisconnect(true);
  WiFi.mode(WIFI_STA);
  WiFi.begin(SSID, PSK);
  // Try forever
  while (WiFi.status() != WL_CONNECTED) {
    Serial.println("...Connecting to WiFi");
    delay(1000);
  }
```

```
  Serial.println("Connected");
  //create I2S output do use with decoder
  //the second parameter 1 means use the internal DAC
  out = new AudioOutputI2S(0,1);
  //init the LCD display
  lcd.init();
  lcd.backlight();
  lcd.createChar(1, speaker);
  //set current station to 0
  curStation = 0;
  //start preferences instance
  pref.begin("radio", false);
  //set current station to saved value if available
  if (pref.isKey("station")) curStation = pref.getUShort("station");
  //set active station to current station
  //show on display and start streaming
  actStation = curStation;
  showStation();
  startUrl();
}

void loop() {
  //check if stream has ended normally not on ICY streams
  if (decoder->isRunning()) {
    if (!decoder->loop()) {
      decoder->stop();
    }
  } else {
    Serial.printf("MP3 done\n");

    // Restart ESP when streaming is done or errored
    delay(10000);

    ESP.restart();
  }
  //read events from rotary encoder
  rotary_loop();

}
```

Sketch zum Herunterladen:

Vor dem Kompilieren muss die SSID und das Passwort für das WLAN gesetzt werden. Am Anfang des Sketchs ist eine Liste mit 24 deutschen Radiostationen. Sie können diese beliebig editieren oder erweitern, um Ihr gewünschtes Programm zu hören. Es können maximal 100 Stationen definiert werden.

Nach dem Hochladen kann das Programm gestartet werden. Mit dem Rotary-Encoder kann durch die Senderliste gescrollt werden. Drückt man den Knopf des Rotary-Encoder, wird der gerade angezeigte Sender als aktiv gesetzt. Diese Auswahl wird im Flash gespeichert, sodass nach einer Stromunterbrechung das Programm wieder mit dem ausgewählten Sender gestartet wird. Die gerade wiedergegebene Station wird im Display durch ein vorangestelltes Lautsprecher-Symbol angezeigt.