ESP32-radio V2.0

This document describes the realization of an Internet radio based on an ESP32 WiFi module.
For the ESP32 an Internet radio was build in 2017 using a VS1053 mp3 decoder. Now the software is updated to support I2S output. The latest version also supports SPDIF (Toslink) output.
A 1.8 TFT color display to give some information about the radio station that's playing. Other types of display are also supported, including NEXTION.

In the simplest configuration, you can build an Internet radio with only 2 components.

1 Features:

- Can connect to thousands of Internet radio stations that broadcast MP3 or AAC audio streams.
- Output to VS1053, SPDIF, I2S[1)].
- MP3 playlists supported.
- Uses a minimal number of components.
- Has a preset list of a maximum of 100 favorite radio stations as preferences in flash.
- Can be controlled by a tablet or other device through the built-in webserver.
- Can be controlled by MQTT commands. Status info is published by using MQTT.
- Can be controlled by commands on the serial input.
- Can be controlled by IR.
- Can be controlled by rotary switch encoder.
- Up to 14 input ports can be configured as control buttons like volume, skip to the next/previous/first/favorite preset station.
- I/O pins for TFT, VS1053, I2S, IR and rotary encoder can be configured in preferences.
- The strongest available WiFi network is automatically selected. Passwords are kept as preferences in flash. Heavily commented source code, easy to add extra functionality.
- Debug information through serial output.
- 2nd processor is used for smooth playback.
- Update of software over WiFi (OTA) through PlatformIO IDE.
- Parameters like WiFi information and presets can be edited in the web interface.
- Preset, volume, bass and treble settings are saved in preferences.
- Displays time of day on TFT.
- Optional displays remaining battery charge.
- Bit-rates up to 320 kbps.
- Control of TFT back-light.
- PCB available through PCBWay.
- 3D printable box available.

1.1 Restrictions for I2S/SPDIF and internal DAC output.

1.1.1 Restrictions for SPDIF output.

An external amplifier with SPDIF (Toslink) input is required. For headphone output only, you may use a simple spdif to analog converter.

1.1.2 Restrictions for I2S output.

There is no tone control for I2S output. You have to use analog controls on the external amplifier.

1.1.3 Restrictions for DAC output.

There is no tone control for internal DAC . You have to use analog controls on the external amplifier.
The internal DAC uses only 8 bits for the digital to analog conversion. This effects the sound quality.

1 See restrictions for I2S output

1.2 Bluetooth output.

If I2S output is enabled and you are using a PCM5102 converter, you can connect a “Bluetooth 5.0 4.2 Receiver Transmitter”. See here. Due to technical limits, it is not possible to use the internal ESP bluetooth.

2 Software:

The software for the radio is supplied as a PlatformIO project using the Arduino platform.
The following libraries are used:

- Wire
- knolleary/PubSubClient@^2.8
- adafruit/Adafruit BusIO@^1.9.3
- adafruit/Adafruit GFX Library@^1.10.4
- adafruit/Adafruit ILI9341@^1.5.9
- adafruit/Adafruit ST7735 and ST7789 Library@^1.7.5
- me-no-dev/AsyncTCP@1.1.1
- me-no-dev/ESP Async WebServer@^1.2.3
- yveaux/AC101@^0.0.1
- djuseeq/Ch376msc @ ^1.4.4knolleary/PubSubClient@^2.8.0

The data directory of the project contains the files for the web interface: index.html, favicon.ico, style.css, search.html, config.html, defaultprefs.txt, (no)mp3play.html and about.html.

The directory structure of the project is developed for PlatformIO. If you want to build the project using the Arduino IDE, you should follow the instructions in chapter 17 of this document.

2.1 Configuration in config.h.

In order tor enable or disable various features the config.h file can be edited before compiling.

You can set a fixed WiFi network to simplify the initial setup, define the MP3 decoder, the display and the type of the rotary encoder.

2.2 Preferences

Preferences for ESP32-Radio are kept in Flash memory (NVS).
You may change the contents of NVS with the “Config”-button in the web interface. An example of the contents is:

```
bat0   = 2178                                          # ADC for 0% battery capacity left
bat100 = 3256                                          # ACD for 100% battery capacity
clk_dst = 1                                            # Offset during daylight saving time (hours)
clk_offset = 1                                         # Offset with respect to UTC in hours
clk_server = pool.ntp.org                              # Time server to be used
#
gpio_00 = uppreset = 1
gpio_12 = upvolume = 2
gpio_13 = downvolume = 2
gpio_14 = stop
gpio_17 = resume
#
ir_40BF = upvolume = 2
ir_C03F = downvolume = 2
#
mqttbroker = none
mqttpasswd = none
mqttport = 1883
mqttprefix = none
mqttuser = none
#
pin_i2s_bck = 27            # I2S (UDA1334, PCM5102)
pin_i2s_din = 25
pin_i2s_lck = 26
pin_i2s_spdif = 15          # SP/DIF output
#
preset = 9
preset_00 = 109.206.96.34:8100                         #  0 - NAXI LOVE RADIO, Belgrade, Serbia
preset_01 = airspectrum.cdnstream1.com:8114/1648_128 #  1 - Easy Hits Florida 128k
preset_02 = us2.internet-radio.com:8050                #  2 - CLASSIC ROCK MIAMI 256k
preset_03 = airspectrum.cdnstream1.com:8000/1261_192 #  3 - Magic Oldies Florida
preset_04 = airspectrum.cdnstream1.com:8008/1604_128 #  4 - Magic 60s Florida 60s Classic Rock
preset_05 = us1.internet-radio.com:8105                #  5 - Classic Rock Florida - SHE Radio
preset_06 = icecast.omroep.nl:80/radio1-bb-mp3         #  6 - Radio 1, NL, 192k
preset_07 = 205.164.62.15:10032                        #  7 - 1.FM - GAIA, 64k
preset_08 = www.doowopradio.com:8000                   #  8 - Doo-Wop Radio, 96k
preset_09 = server-27.stream-server.nl:8192/stream     #  9 - 192 Radio Nederland, 320k
#
toneha = 0
tonehf = 0
tonela = 0
tonelf = 0
#
volume = 42
#
wifi_00 = ADSL-11/DEADC0DE11
wifi_01 = SSID2/PASSWD2
```

Note that the size of the NVS is limited to 20 kB.
Lines starting with “#” are comment lines and are ignored. Comments per line (after “#”) will also be ignored, except for the “preset_” lines. The comments on the “preset_” lines are used to identify the presets in the web interface.
The maximum line length is 150 characters.
Note that the preset numbers ranges from 000 to 199. The range can be extended by the definition of MAXPRESETS. Note that there is also a limit on the NVS space. If the highest numbered station is reached, the next station will be 00 again. URLs with mp3 files or mp3 playlists (.m3u) are allowed.

Preferences in the form "pin_xxxx" specify pin numbers for the various interfaces. For a FEATERBOARD you have to change some settings, especially for the display. The pins for SPI default to SCK=18, MISO=19 and MOSI=23, but they may be configured by “pin_spi_xxx” commands.
The complete list of configurable interface pins is:

Configuration name	Function
pin_ir	Pin for IR sensor (VS1838B)
pin_enc_clk	Rotary encoder CLK pin
pin_enc_dt	Rotary encoder DT pin
pin_enc_up	ZIPPY B5 side switch up
pin_enc_dwn	ZIPPY B5 side switch down
pin_enc_sw	Rotary encoder switch down
pin_tft_cs	Display SPI version CS pin
pin_tft_dc	Display SPI version DC pin
pin_tft_scl	Display I2C version SCL. For OLED, LCD1602, LCD2004
pin_tft_sda	Display I2C version SDA For OLED, LCD1602, LCD2004
pin_tft_bl	Display back-light. The BL is disabled after some time if there is no activity. This helps to reduce power when running on batteries.
pin_tft_blx	Display back-light (inversed logic)
pin_nxt_rx	NEXTION (input from NEXTION)
pin_nxt_tx	NEXTION (output to NEXTION)
pin_sd_cs	SD card select
pin_sd_detect	SD detect inserted card (LOW is inserted)
pin_vs_cs	VS1053 CS pin
pin_vs_dcs	VS1053 DCS pin
pin_vs_dreq	VS1053 DREQ pin
pin_shutdown	Amplifier shutdown pin. Pin will be set if the radio is not playing or the volume is set to zero. This output-pin can be used to shut down the amplifier.
pin_shutdownx	Amplifier shutdown pin (inversed logic)
pin_i2s_bck	I2S BCK signal for I2S devices
pin_i2s_lck	I2S L(R)CK signal for I2S devices
pin_i2s_din	I2S DIN signal for I2S devices
pin_i2s_spdif	SP/DIF signal
pin_spi_sck	SPI bus SCK pin, default pin 18
pin_spi_miso	SPI bus MISO pin, default pin 19
pin_spi_mosi	SPI bus MOSI pin, default pin 16
pin_eth_mdc	For boards with Ethernet, MDC signal, default pin 23
pin_eth_mdio	For boards with Ethernet, MDIO signal, default pin 18
pin_eth_power	For boards with Ethernet, POWER signal, default pin 16

The "gpio_" lines specify input pins and the command that is executed if the input pin goes from HIGH to LOW.

The "touch_" lines specify the GPIO input pins for this signal and the command that is executed if the input pin is activated.

The "ir_XXXX" lines specify IR-codes and the command that is executed if the IR-code is received. The code part is a hexadecimal number in upper-case characters.

The "clk_" lines are used for the display of the current time on the TFT. Please change the values for your timezone. The values in the example are valid for the Netherlands.

The “bat0” and “bat100” lines are for displaying the remaining battery capacity on the display. See the details at the paragraph with optional features.

The preferences can be edited in the web interface. Changes will in some cases be effective after restart of the Esp-radio. If the list of preferences is empty (first start), you may use the default button. The list will show some default values which can be edited.

2.3 Using Winamp to find out the correct preset line for a station.

Press Alt-3 in the main window (left picture). You will see info for the playing station (right picture).
The top line (with “http”) will contain the information for the preset, in this example:
“us1.internet-radio.com:8105”. Remove the “http...”-part. The complete line for this station in the preferences would be:

```
preset_05 = us1.internet-radio.com:8105            #  5 - Classic Rock Florida - SHE Radio
```

Optional features:

2.3.1 Digital control through input pins:

Normally the radio is controlled by the web interface. However, free digital inputs (GPIO) may be connected to buttons to control the radio. Their function can be programmed using the web interface.

You can assign commands to the digital inputs by adding lines in the configuration (Webinterface, "Config" page). Examples:

```
gpio_00 = uppreset = 1
gpio_12 = upvolume = 2
gpio_13 = downvolume = 2
gpio_21 = station = icecast.omroep.nl:80/radio1-bb-mp3
```

In this example the ESP32-Radio will execute the command "uppreset=1" if GPIO0 will go from HIGH to LOW. The commands are equal to the commands that are handled by the serial input or by the MQTT interface.
The same format can be used for the touch-inputs:

```
touch_04 = uppreset = 1
touch_13 = upvolume = 2
```

In this example the ESP32-Radio will execute the command "uppreset=1" if TOUCH0 (GPIO04) will be activated. The commands are equal to the commands that are handled by the serial input or by the MQTT interface.

2.3.2 IR Interface.

The radio can be controlled by an IR remote control like this:

To use this interface, the "out" pin of a VS1838B receiver must be connected to a GPIO pin of the ESP32:

Add the assigned GPIO pin to the preferences through the config page in the web interface. Example:

```
pin_ir = 35                    # GPIO Pin number for IR receiver VS1838B
```

VCC is connected to 3.3 Volt. A 220 µF capacitor should be connected between VCC and GND.
The software will read the raw code of the IR transmitter, making it possible to use virtually any remote control to be used. I tested it with the 21 button remote as well as with an LG TV remote.
To assign functions to the buttons, watch the debug log output while pressing a button. For example, press the +volume button. You will see something like:

```
D: IR code 807F received, but not found in preferences!
```

Now add the command:

```
ir_807F = upvolume = 2
```

to the preferences in the config page of the web interface. Likewise you can assign functions to all buttons, for example:

```
ir_8A31 = uppreset = 1
ir_719A = station = us1.internet-radio.com:8105
ir_1F6B = mute
ir_661A = sleep
```

2.3.3 Display remaining battery capacity:

The remaining battery capacity is computed by measuring the battery voltage on ping GPIO36 (ADC0).
The battery voltage must be supplied to ADC0 through a resistor voltage divider. The maximum voltage fort his input is 1.0 volt. I used 100 kΩ and 22 kΩ fort his purpose.
Furthermore you need to specify ADC value for both the full and the empty voltages. These values must be in the preferences like:

```
bat0   = 2178   # ADC for 0% battery capacity left
bat100 = 2690   # ACD for 100% battery capacity
```

To calibrate the settings you may use the "TEST" command in the serial monitor. This will display the current reading of the ADC0 pin. Do this for a fully charge battery and for an (almost) empty one.
Note that it will take about 6 seconds before the reading is stable. This is a result of a filter in the measurement of the ADC.

The remaining capacity is displayed as a green/red bar near the top of the display.

3 Hardware:

The radio is built with the following widely available hardware:

- An ESP-32 module. This is basically an ESP32 on a small print. I used a DOIT ESP32 Development Board. See figure 1 below. The ESP32 is running on 160 MHz. On Aliexpress: this.
- An optional VS1053 module. See figure 2.
- An optional PCM5102, 1334A or similar I2S to analog module. See figure 14.
- An optional SP/DIF transmitter. See figure 15.
- A 1.8 inch color TFT display 160x128. Optional, see figure 3 and 4. On Aliexpress: this or this.
- A 1.44 inch color TFT display 128x128. Optional, see figure 9. On Aliexpress: this.
- An OLED 128x64 display. Optional, see figure 7. On Aliexpress: this.
- An 1602 LCD display with I2C backpack. Optional, see figure 8.
- An 2004 LCD display with I2C backpack. Optional, see figure 13.
- A NEXTION display. Optional, see figure 10. Tested with a NX3224T024_011.
- Two small speakers.
- A Class D stereo amplifier to drive the speakers. Best quality if powered by a separate power source.
- A rotary encoder switch. Optional, see figure 5.
- A ZIPPY B5 side switch. Optional. This switch may replace the rotary decoder. Define “ZIPPYB5” in config.h and define “pin_enc_dwn” and “pin_enc_sw” in the config page of the webinterface. See figure 12.
- A IR receiver. Optional, see figure 6.
- Two audio transformers. Optional, see figure 11.

Fig. 1

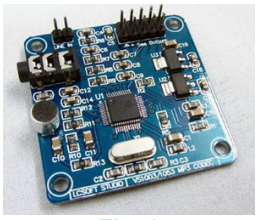


Fig. 2

Fig. 3

Fig. 4

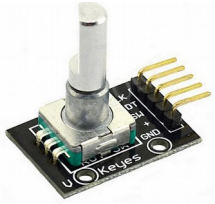


Fig. 5

Fig. 6

Fig. 7

Ffig. 8

Fig. 9

Fig. 10

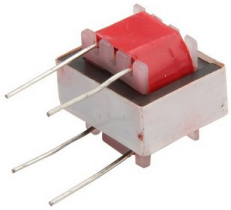


Fig. 11

Fig. 12

Fig. 13

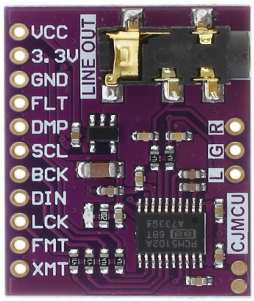
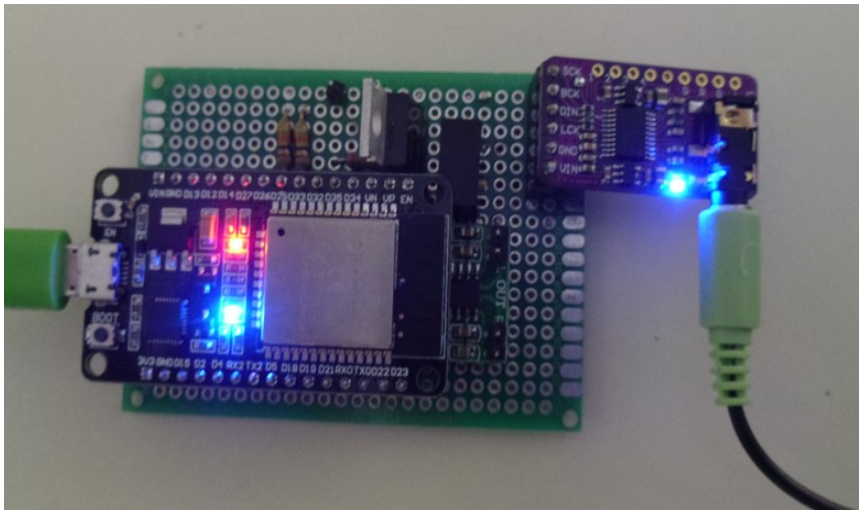


Fig. 14

Fig. 15

Here is a picture of the radio in a test configuration with software MP3 decoding. Only the PCM5102 is connected. There is also a PAM8303 mounted under the ESP32-board to test internal DAC output.
Tip: connect the SCK pin of the PCM5102 to ground. This will protect against heavy noise.



4 PCB.

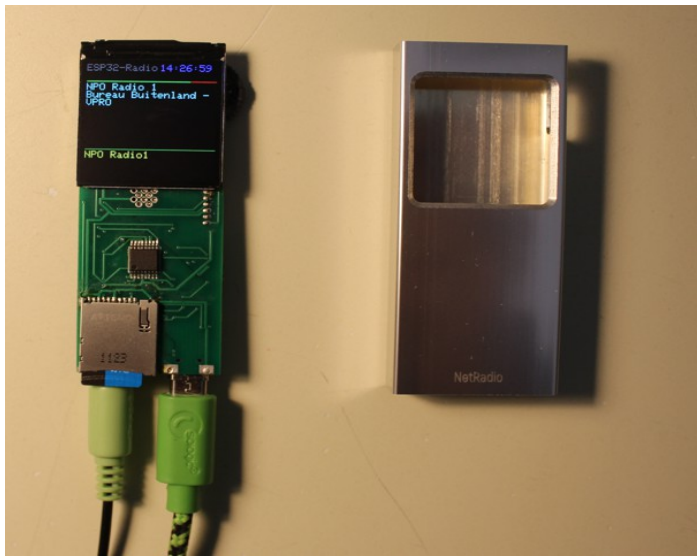
You can order a PCB for this radio here.

Hans Vink designed a 3D printable case for this radio. See https://www.thingiverse.com/thing:3346460

It looks like this:

4.1 Another model.

Print and case by Simeon Kartalov from Bulgaria. It has headphone output and uses a Zippy B5 side switch for the rotary encoder.



5 Wiring:

The logic wiring in the table below. The analog amplifier and the speakers are not included.
Note that the GPIO pins are just an example. The assignment of GPIO pins is defined in the preferences that can be edited in the "config" page of the web interface.

ESP32dev	Signal	Wired to LCD	Wired to VS1053	Wired to the rest
GPIO16		-	pin 1 XDCS	-
GPIO5		-	pin 2 XCS	-
GPIO4		-	pin 4 DREQ	-
GPIO2		pin 3 D/C or A0	-	-
GPIO22		-	-	-
GPIO21	RXD2	-	-	TX of NEXTION (if in use)
GPIO17	TXD2	-	-	RX of NEXTION (if in use)
GPIO18	SCK	pin 5 CLK or SCK	pin 5 SCK	-
GPIO19	MISO	-	pin 7 MISO	-
GPIO23	MOSI	pin 4 DIN or SDA	pin 6 MOSI	-
GPIO15		pin 2 CS	-	-
GPI03	RXD0	-	-	Reserved serial input
GPIO1	TXD0	-	-	Reserved serial output
GPIO34	-	-	-	Optional pull-up resistor
GPIO35	-	-	-	Infrared receiver VS1838B
GPIO25	-	-	-	Rotary encoder CLK
GPIO26	-	-	-	Rotary encoder DT
GPIO27	-	-	-	Rotary encoder SW
GND	-	pin 8 GND	pin 8 GND	Power supply GND
VCC 5 V	-	pin 7 BL	-	Power supply
VCC 5 V	-	pin 6 VCC	pin 9 5V	Power supply
EN	-	pin 1 RST	pin 3 XRST	-

6 Uploading firmware and data.

Find out what COM-poort is used by the USB to Serial convertor on the ESP32 board. That should be visible in the Windows device manager. In PlatformIO, edit the COM port settings. Example for COM9:

```
monitor_port = COM9
upload_port = COM9
```

Then open a terminal in PlatformIO with this button:

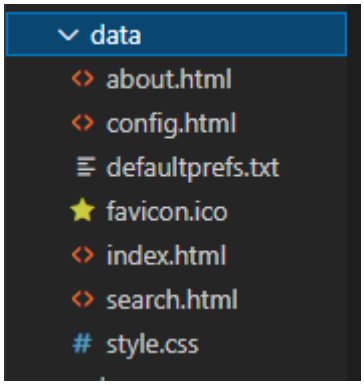
Flash the firmware with PlatformIO. Use this command in the terminal:

```
pio run -t upload
```

Now upload the filesystem for the web interface from the data directory with the command:

```
pio run -t uploadfs
```

Your data directory should look like this:



Start in PlatformIO the serial monitor with the button:

Press reset on the ESP32.

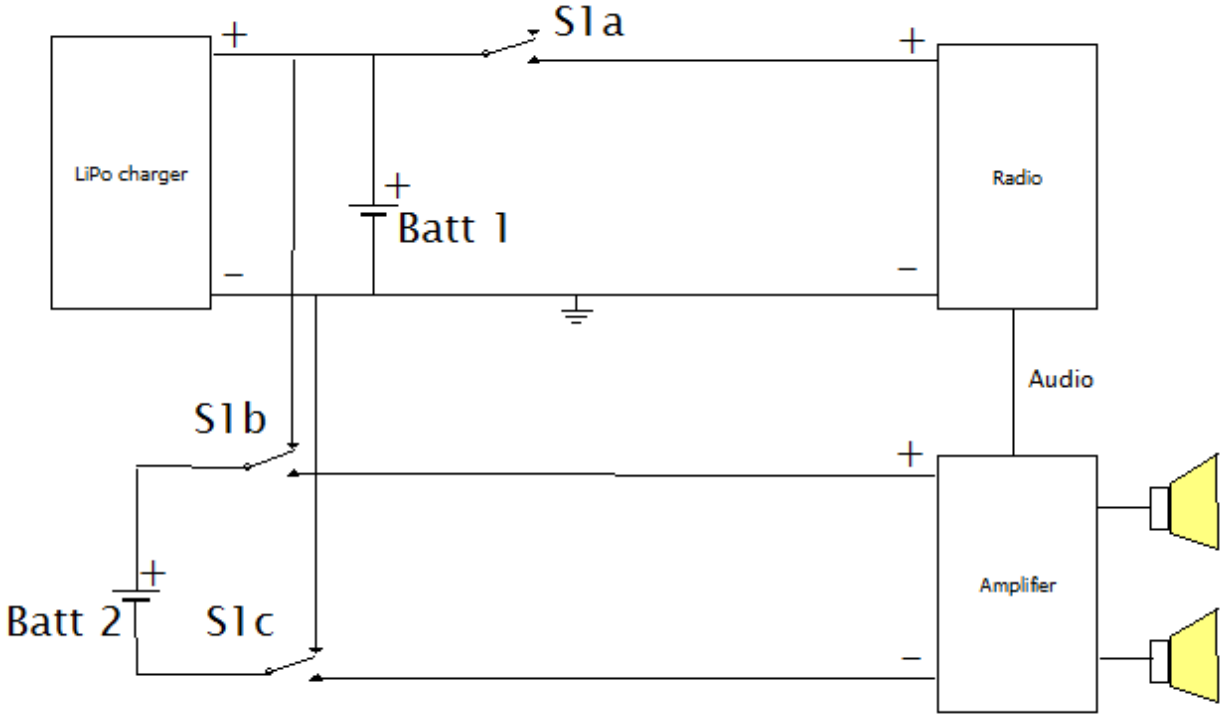
In the serial output you see lines like this:

D: Connect to WiFi
D: WiFi Failed! Trying to setup AP with name ESP32Radio and password ESP32Radio.
D: IP = 192.168.4.1

Connect to the access point “ESP32Radio” on your Windows machine. Password is “ESP32Radio”.
Open in your web browser “http://192.168.4.1”. You should see the home page of the radio.
Go to the config tab and do some configuration.

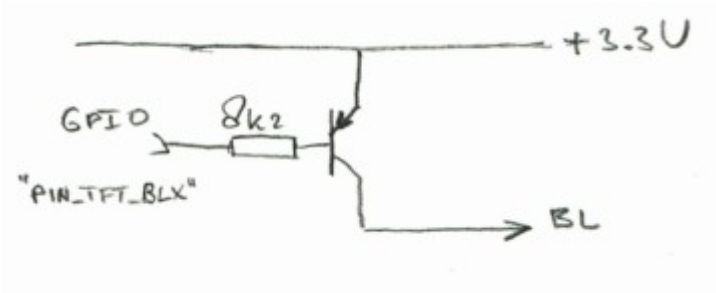
7 Amplifier and power circuit (VS1053 version).

The amplifier is a class D stereo amplifier. If the power is shared with the power supply of the radio, you will hear much noise. So I used a separate LiPo battery (Batt 2) for the amplifier.
During operation only Batt 1 will be charged. If the radio is switched off, both batteries will be recharged by the LiPo charger. S1a, S1b and S1c is a triple On-On switch.
Note that there may be high currents in the “off”-position if one of the batteries fully discharged. Use protected batteries only!



A better approach is to use 2 small audio transformers to isolate the output of the VS1053 from the power circuit. This is used on the PCB mentioned before.

8 Backlight control circuit.



9 SD card.

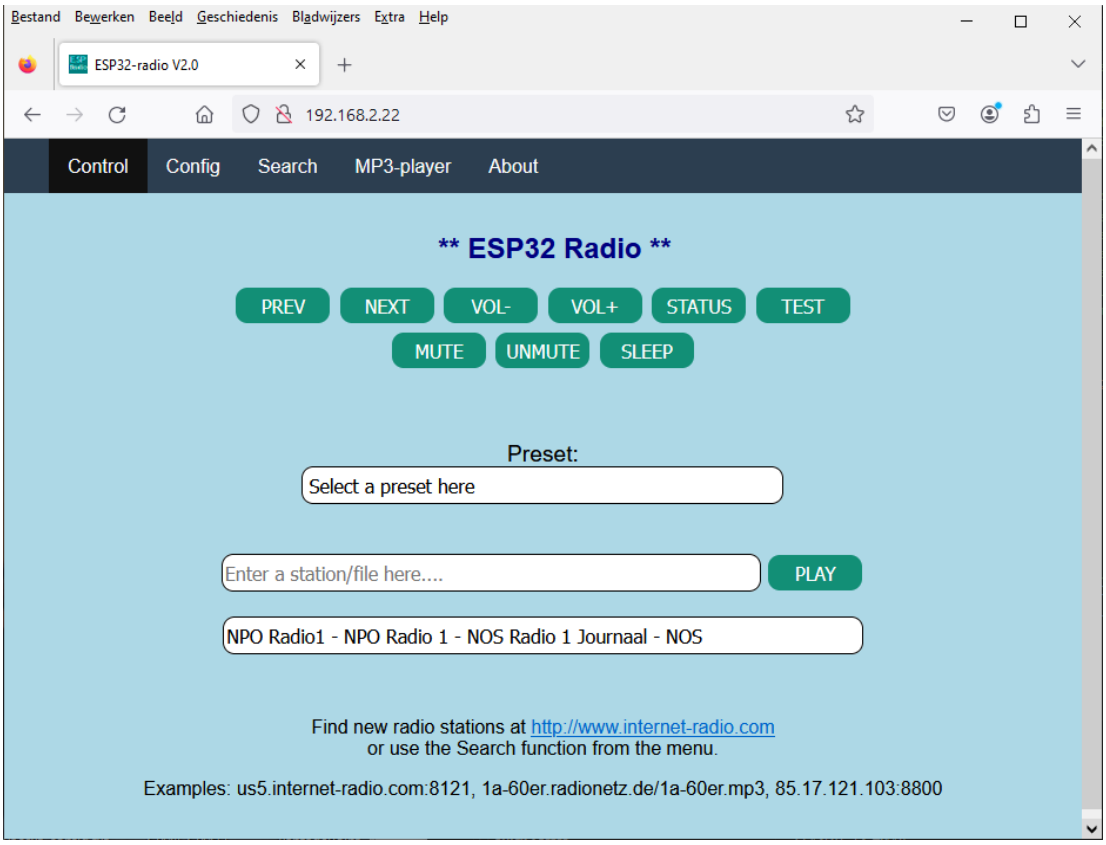
As soon as a new SD is detected in the (optional) SD card slot, the radio starts to index all MP3 tracks on the card. The tracks (compressed filenames) are stored in a file called “tracklist.dat” on the card. If the tracklist is already on the card, no new file will be generated, but the tracks in the file are read in order to count the number of available tracks.
If you add new tracks to the card, you should remove the file “tracklist.dat” in the rot directory of the SD card. The next time you insert the card into the radio, a new index will be generated.
You may choose a tracktitle in the MP3-player page of the webinterface.

10 Web interface.

The web interface is simple and can be adapted to your needs. The basic idea is to have a html page with embedded javascript that displays an interface to the radio. Command to the radio can be sent to the http server on the ESP32. The IP address of the webserver will be displayed on the TFT during startup. The webpages are defined in the data directory.

10.1 Capabilities of the webserver:

Let’s assume that the IP of the ESP32-radio is 192.168.2.12. From your browser you can show a simple root page by entering the following URL: http://192.168.2.12. This will display the index.html page from the ESP32 as well as favicon.ico.
If your computer is configured for mDNS, you can also use http://ESP32Radio.local in your browser.
The following simple web interface will be displayed (VS1053 version):



Clicking on one of the available buttons will control the ESP32-radio. The reply of the webserver will be visible in the status box below the buttons. A click will be translated into a command to the ESP32-radio in the form:
http://192.168.2.12/?<parameter>=<value>
For example: http://192.168.2.12/?upvolume=2

The “STATUS” and “TEST” buttons give some info about the playing stream/file. The “Search” menu option can be used to discover stations. The list of stations is maintained by Community Radio Browser. See https://www.radio-browser.info.

The “SLEEP” button puts the ESP32 into sleep mode. Only the reset button can wake-up the radio; a reset is necessary.

Not all functions are available as buttons in the web interface shown above. Commands may also come from MQTT or serial input. Not all commands are meaningful on MQTT or serial input. Working commands are:

```
preset     = 12                   Select start preset to connect to
preset_00  = <mp3 stream>         Specify station for a preset 00-max *)
volume     = 95                   Percentage between 0 and 100
upvolume   = 2                    Add percentage to current volume
downvolume = 2                    Subtract percentage from current volume
toneha     = <0..15>              Setting treble gain
tonehf     = <0..15>              Setting treble frequency
tonela     = <0..15>              Setting bass gain
tonelf     = <0..15>              Setting treble frequency
station    = <mp3 stream>         Select new station (will not be saved)
station    = <URL>.mp3            Play standalone .mp3 file (not saved)
station    = <URL>.m3u            Select playlist (will not be saved)
(un)mute                          Mute/unmute the music
wifi_00    = mySSID/mypassword    Set WiFi SSID and password *)
mqttbroker = mybroker.com         Set MQTT broker to use *)
mqttprefix = XP93g                Set MQTT broker to use
mqttport   = 1883                 Set MQTT port to use, default 1883 *)
mqttuser   = myuser               Set MQTT user for authentication *)
mqttpasswd = mypassword           Set MQTT password for authentication *)
mqttrefresh                       Request to refresh all MQTT items
clk_server = pool.ntp.org         Time server to be used *)
clk_offset = <-11..+14>           Offset with respect to UTC in hours *)
clk_dst    = <1..2>               Offset during dst in hours *)
sleep                             Put radio into deep sleep mode.  Wake-up by reset only.
settings                          Returns setting like presets and tone
status                            Show current URL to play
test                              For test purposes
debug      = 0 or 1               Switch debugging on or off
reset                             Restart the ESP32
bat0       = 2318                 ADC value for an empty battery
bat100     = 2916                 ADC value for a fully charged battery
```

Commands marked with "*)" are sensible during power-up only.

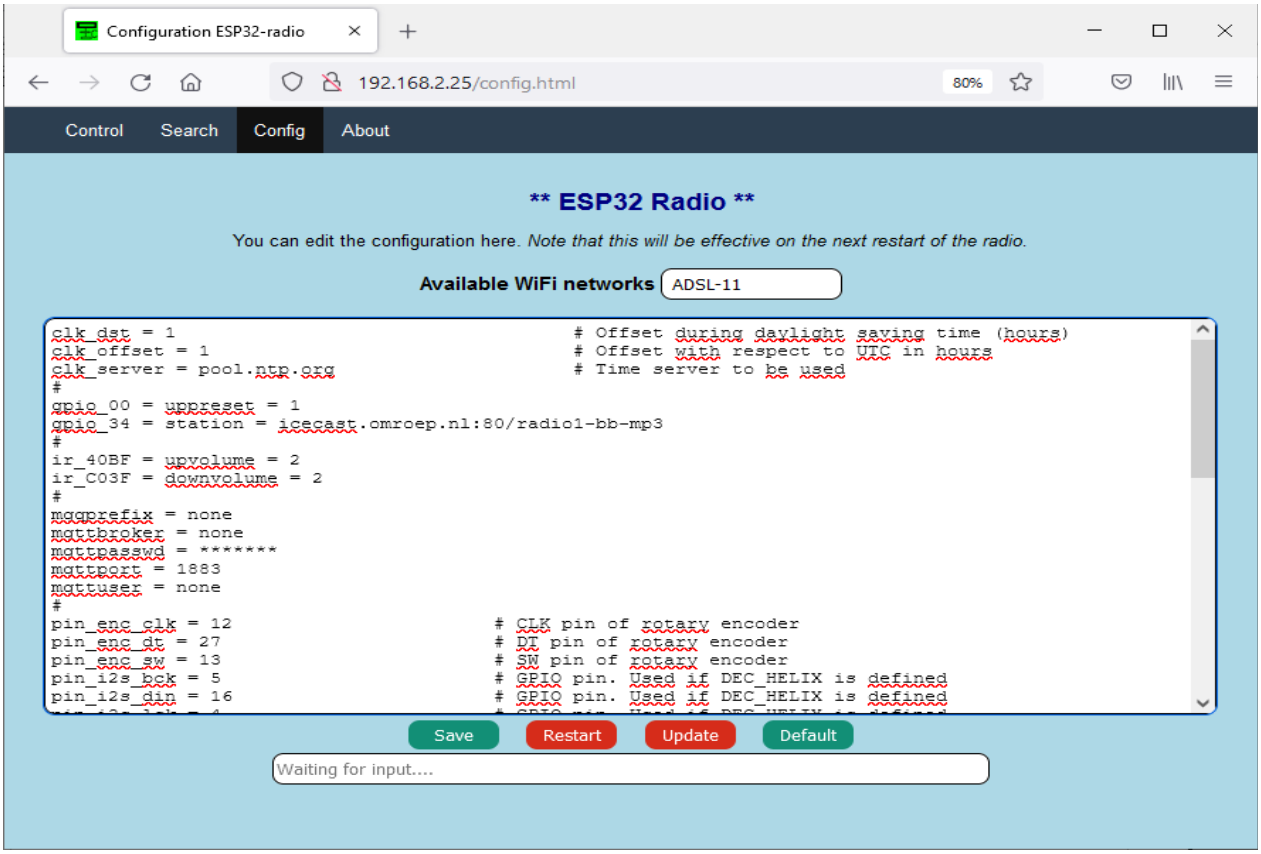
Station may also be of the form “skonto.ls.lv:8002/mp3”. The default port is 80.
Station may also point to an mp3 playlist. Example: “stream.laut.fm/radioparty.m3u”.
Station may be an .mp3-file on a remote server. Example: "samples-files.com/samples/Audio/mp3/sample-file-1.mp3".
It is allowed to have multiple (max 200) “preset_” lines. The number after the “_” will be used as the preset number. The comment part (after the “#”) will be shown in the web interface.

It is also allowed to have multiple “wifi_” lines. The strongest WiFi access point will be used.

11 Configuration

The "Config" button will bring up a second screen. Here you can edit the preferences. The config screen will be shown automatically if the ESP32 cannot connect to one of the WiFi stations specified in the preferences. In that case the ESP32 will act as an access point with the name “ESP32-Radio”. You have to connect to this AP with password “ESP32-Radio”. Then the ESP32-radio can be reached at http://192.168.4.1.
The configuration screen will be displayed and you have the opportunity to edit the preferences. A "default" button is available to fill the edit-box with some example data. For a quick start, just fill in you WiFi network name and password and click "Save". Restart the radio “Restart” button.

After changing the contents of this page, it must be saved to the preferences by clicking the “Save” button. Changes will have effect on the next restart of the ESP-radio, so click the “Restart” button.



12 Rotary encoder interface.

The rotary encoder switch can control some essential functions of the ESP32-radio. The "GND"- pin must be connected to ground. Some encoders have a “VCC” pin. In that case, connect it to the 3.3 Volt pin of the ESP32 DEV module.

12.1 Rotary encoder volume control.

The default function is volume control. Turning the knob will result in lower or higher volume. Pressing the knob will mute/unmute the signal. The text “Mute” or “Unmute” will be shown on the TFT during 4 seconds.

12.2 Rotary encoder preset mode.

A double click selects the preset-mode. Rotation of the switch will select one of the preset stations. The preset will be shown on the TFT. Once a preset is selected, you can activate this preset by a single click.
Without rotation, the next preset is selected.
After an inactivity of 4 seconds the rotary encoder will return to its default function (VOLUME).

12.3 Rotary encoder SD track select.

A triple click will select SD mode. Rotate to scroll through the list of tracks and press the knob to play the track.

12.4 Rotary encoder stop player.

A long click will stop the radio/mp3player.

13 Supported displays.

The preferred display is a 1.8 inch TFT with 160x128 pixels. Other displays may also be used:

- No display. Edit the config.h file (around line 35), so that “DUMMYTFT” is defined and all other display types are commented out.
- The preferred display 160x128 pixels. Define “BLUETFT” and comment out the other types.
- An ILI9341 display, default 320x240 pixels. Define “ILI9341” and comment out the other types.
- A 128x128 1.44 inch display. Edit bluetft.h (around line 58), use “INITR_144GREENTAB” and set dsp_getwidth() to 128.
- A ST7789 240 x 240 display. Define “ST7789 “ and comment out the other types.
- A 128x64 OLED. Define “”OLED” and comment out the other types.
- An LCD 1602 display with I2C backpack. Define “LCD1602I2C” and comment out the other types. Note that this display is limited as it has only 32 characters.
- An LCD 2004 display with I2C backpack. Define “LCD2004I2C” and comment out the other types. Note that this display is limited as it has only 80 characters.
- A NEXTON model NX#@@$T024_11 display connected to GPIO 16 and 17.

The OLED, 2004 and 1602 displays use the serial I2C bus. There must be definitions for this in the preferences, for example:

```
pin_tft_scl = 13                              # GPIO Pin number for SCL
pin_tft_sda = 14                              # GPIO Pin number for SDA
```

Th I2C address of the 2004 and 1602 displays is defined in the corresponding module.

14 Wired (ethernet) version.

A wired version of the radio is also possible. For testing I used a WT32-ETH01 module that has a built-in LAN8720 ethernet controller. The number of free pins is therefore limited. With the VS1053 and the BLUETFT display I used the following pin setting in the configuration page of the web interface:

```
pin_vs_cs = 2         # GPIO VS1053 CS
pin_vs_dcs = 12       # GPIO VS1053 DCS
pin_vs_dreq = 4       # GPIO VS1053 DREQ
#
pin_spi_sck = 5       # GPIO SPI SCK
pin_spi_miso = 33     # GPIO SPI MISO
pin_spi_mosi = 17     # GPIO SPI MOSI
#
pin_tft_cs = 15       # GPIO BLUETFT CS
pin_tft_dc = 14       # GPIO BLUETFT A0
```

This module has also been tested with I2S output. The pins used above for the VS1053 were used for the 3 I2S signals.

15 AI Thinker Audio kit V2.1.

This board has an I2S DAC on-board and there are speaker outputs as well as an output for headphones.
The number of free GPIOs is limited. The internal DAC is a AC101. Note that later versions of this board may have a different DAC. The software works for the AC101 only!

GPIOs that are used internally are 2, 4, 13, 14, 15, 25, 26, 27, 32, 33, 34.

GPIOs that are free and available at the 12 pin header are 21, 22, 19, 23, 18, 5.

If you want to use the KEYs, only 21 and 22 are free.

Since there are only a few GPIO pins are available, the best option for the human interface is probably a NEXTION display. For testing I used “pin_nxt_rx = 18” and “pin_nxt_tx = 5” in de config page of the web interface.

The GPIO configuration in the web interface should contain the following lines:

```
pin_i2s_bck = 27                                # AI GPIO Pin for I2S "BCK"
pin_i2s_din = 25                                # AI GPIO Pin for I2S "DIN"
pin_i2s_lck = 26                                # AI GPIO Pin for I2S "L(R)CK"
pin_nxt_rx = 18                                 # NEXTION serial input
pin_nxt_tx = 5                                  # NEXTION serial output
pin_sd_cs = 13                                  # AI GPIO Pin for Chip select SD card
pin_sd_detect = 34                              # AI GPIO Pin for CD inserted
pin_spi_miso = 2                                # AI GPIO Pin for SPI MISO
pin_spi_mosi = 15                               # AI GPIO Pin for SPI MOSI
pin_spi_sck = 14                                # AI GPIO Pin for SPI SCK
```

Before compilation, change the config.h file like this:

Section “..MP3/AAC decoder”:

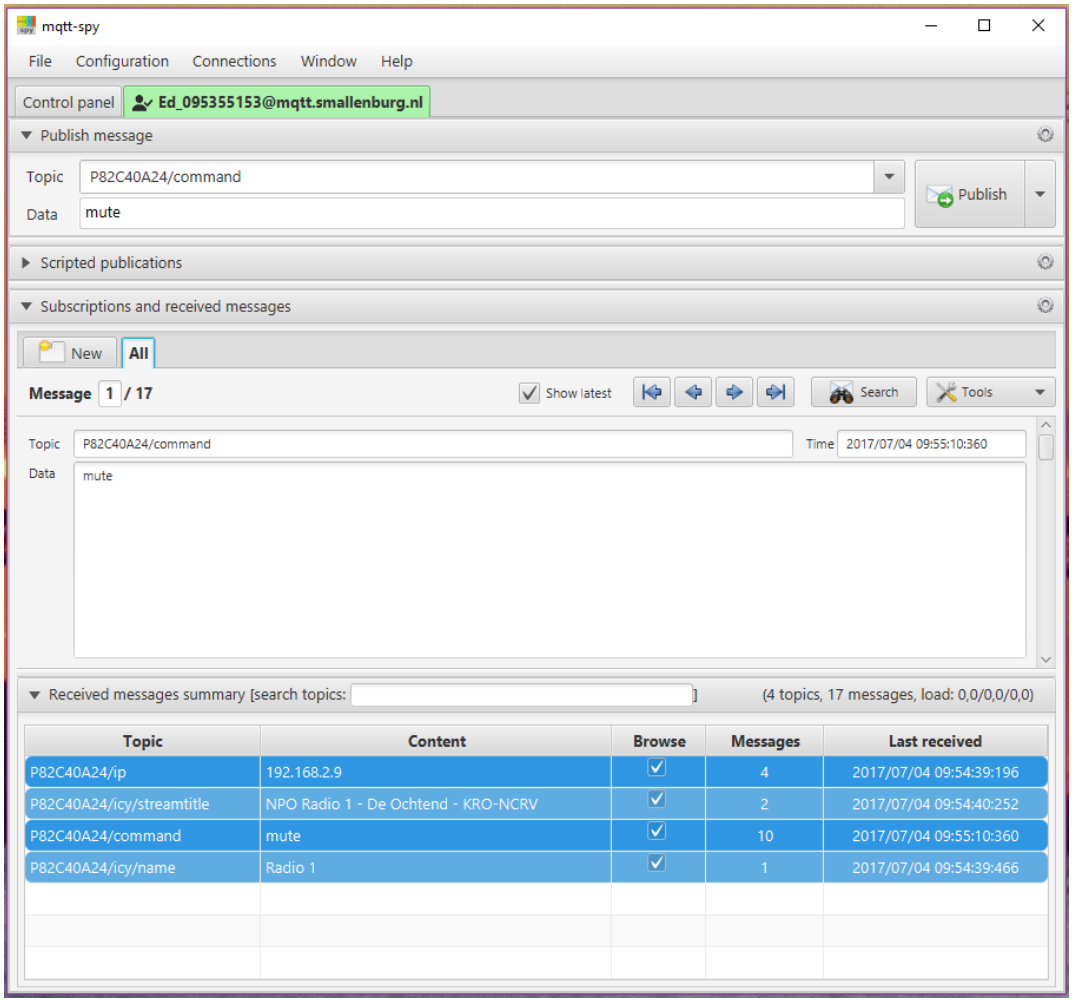
```
#define DEC_HELIX_AI              // Software decoder for AI Audio kip (AC101)
```

Section “..display..”:

```
#define NEXTION                   // Nextion display
```

16 MQTT interface.

The MQTT interface can handle the same commands as the web interface.
As publish command on a Linux system may look like:
`$ mosquitto_pub -h broker.hivemq.com -t espradio -m volume=80`
Note that `broker.hivemq.com` is heavily used, this may cause some delay. If you use your own broker the reaction on commands will be much better.
Remove the lines starting with “mqtt” if no MQTT is required or set mqttbroker to "none".
You can use an MQTT client like https://kamilfb.github.io/mqtt-spy/ to view the MQTT interface.



Subscribe to “P82C40A24/ip” and you will see the IP-address of your radio. You will see the IP address of your radio or the IP address of a different user. So be sure to use unique names for your topics. This can be accomplished by specifying a unique prefix in the preferences, see below.
The parameters in preferences for this example are:

```
# MQTT broker, credentials and topic  to subscribe
mqttbroker = broker.hivemq.com         # Broker to connect with
mqttprefix = P82C40A24                 # Prefix for pub/sub.  Default is part of MAC-address.
mqttport = 1883                        # Portnumber (1883 is default)
mqttuser = none                        # (No) username for broker
mqttpasswd = none                      # (No) password for broker
#
```

In this example I published the command “mute” to the radio. The radio published the IP-address 192.168.2.8 to the broker (once every 10 minutes).

16.1.1 MQTT PUB topics.

In this version 7 topics will be published to MQTT and can be subscribed to by an MQTT client:

prefix/ip	The IP-address of the ESP32-Radio
prefix/icy/name	The name of the station
prefix/icy/streamtitle	The name of the stream

prefix/nowplaying	Track information
prefix/preset	Preset currently plaing
prefix/volume	Current volume setting
prefix/playing	1 if playing, 0 if stopped

Command to control the ESP32-Radio can be published to "prefix/command".

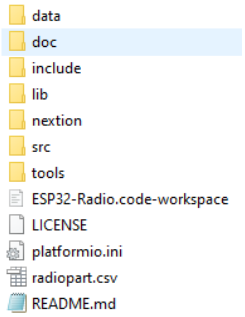
16.1.2 Alternative way of configuration.

An extra sketch "Esp32_radio_init.ino" is supplied as an alternative to initialize the preferences (in Non-Voltile Storage of the ESP32). Just change lines 39 and 40 (the specs for WiFi networks) to match your network(s).
Upload and run the sketch once and then load the ESP32-Radio.

17 Compiling with Arduino IDE.

The preferred way to compile this project is to using PlatformIO. For the ArduinoIDE (version 2.3.2), some extra steps must be taken:

- Download the project from github (https://github.com/Edzelf/ESP32Radio-V2/archive/refs/heads/main.zip). Store the zip file in any directory.
- Go to this directory and unzip the zip file. This will produce a directory with the name “ESP32Radio-V2-main”.
- Rename “ESP32Radio-V2-main” to “ESP32Radio”. That will be your work directory. Got to this directory. It will look like this:



- Remove ”doc”, ”tools”, ”ESP32-Radio.code-workspace”, ”LICENCE”, ”platformio.ini” and ”README.md”
- Move the 3 files in “src” to your work directory. And remove the (now empty) “src” directory.
- Move all files from “include” to your work directory and remove the “include” directory.
- Edit the file config.h to meet your configuration (display, codec, ..). Edit your WiFi credentials in “FIXEDWIFI” and uncomment this line.
- Move the files in “lib/codecs/src” to your work directory. If you use VS1053, you may skip the aac* and mp3* files.
- Depending on your type of display, move the files in “lib/ILI9341/src” to your work directory,
  or move the files in “lib/LCD1602/src” to your work directory,
  or move the files in “lib/LCD2004/src” to your work directory,
  or move the files in “lib/NEXTION/src” to your work directory,
  or move the files in “lib/bluetft/src” to your work directory,
  or move the files in “lib/dummytft/src” to your work directory,
  or move the files in “lib/OLED/src” to your work directory,
  or move the files in “lib/ST7789/src” to your work directory.
- Remove the “lib” directory if you want to save some diskspace.
- Rename the file “main.cpp” to “ESP32Radio.ino”.
- Your work directory will now contain “data”, “nextion”, the source files and maybe “lib”.
- Start the Arduino IDE by double clicking “ESP32Radio.ino”.
- Connect the USB serial port of your ESP32 to your computer.
- You may need to install “esp32 by Espressif” in the “BOARDS MANAGER”.
- Select the right board (ESP32 Dev Module) in Tools->Board and set the right Port.
- Enable debug output in Tools->Core Debug Level. Set it to “Info”.
- You can now verify the sketch. You probably have to install some libraries.

Some of the libraries used (depends on configuration):

```
WiFi at version 2.0.0
PubSubClient at version 2.8
ESPmDNS at version 2.0.0
SPI at version 2.0.0
ArduinoOTA at version 2.0.0
Update at version 2.0.0
ESPAsyncWebServer-master at version 1.2.0
FS at version 2.0.0
AsyncTCP-master at version 1.0.3
SPIFFS at version 2.0.0
Adafruit_ST7735_and_ST7789_Library at version 1.7.4
```

```
Adafruit_GFX_Library at version 1.10.12
Adafruit_BusIO at version 1.9.3
Wire at version 2.0.0
```

18 Compiler/upload output.

18.1 Software upload.

```
Processing esp32 (platform: espressif32; board: esp32doit-devkit-v1; framework: arduino)
------------------------------------------------------------------------------------------------------------
-Verbose mode can be enabled via `-v, --verbose` option
CONFIGURATION: https://docs.platformio.org/page/boards/espressif32/esp32doit-devkit-v1.html
PLATFORM: Espressif 32 (3.5.0) > DOIT ESP32 DEVKIT V1
HARDWARE: ESP32 240MHz, 320KB RAM, 4MB Flash
DEBUG: Current (esp-prog) External (esp-prog, iot-bus-jtag, jlink, minimodule, olimex-arm-usb-ocd,
olimex-arm-usb-ocd-h, olimex-arm-usb-tiny-h, olimex-jtag-tiny, tumpa)
PACKAGES:
 - framework-arduinoespressif32 3.10006.210326 (1.0.6)
 - tool-esptoolpy 1.30100.210531 (3.1.0)
 - tool-mkspiffs 2.230.0 (2.30)
 - toolchain-xtensa32 2.50200.97 (5.2.0)
LDF: Library Dependency Finder -> https://bit.ly/configure-pio-ldf
LDF Modes: Finder ~ chain, Compatibility ~ soft
Found 46 compatible libraries
Scanning dependencies...
Dependency Graph
|-- <Wire> 1.0.1
|-- <SD(esp32)> 1.0.5
|   |-- <FS> 1.0
|   |-- <SPI> 1.0
|-- <PubSubClient> 2.8.0
|-- <Adafruit BusIO> 1.11.3
|   |-- <Wire> 1.0.1
|   |-- <SPI> 1.0
|-- <Adafruit GFX Library> 1.10.14
|   |-- <SPI> 1.0
|   |-- <Adafruit BusIO> 1.11.3
|   |   |-- <Wire> 1.0.1
|   |   |-- <SPI> 1.0
|   |-- <Wire> 1.0.1
|-- <Adafruit ILI9341> 1.5.10
|   |-- <Adafruit GFX Library> 1.10.14
|   |   |-- <SPI> 1.0
|   |   |-- <Adafruit BusIO> 1.11.3
|   |   |   |-- <Wire> 1.0.1
|   |   |   |-- <SPI> 1.0
|   |   |-- <Wire> 1.0.1
|   |-- <SPI> 1.0
|-- <Adafruit ST7735 and ST7789 Library> 1.9.3
|   |-- <Adafruit GFX Library> 1.10.14
|   |   |-- <SPI> 1.0
|   |   |-- <Adafruit BusIO> 1.11.3
|   |   |   |-- <Wire> 1.0.1
|   |   |   |-- <SPI> 1.0
|   |   |-- <Wire> 1.0.1
|   |-- <SPI> 1.0
|-- <ESP Async WebServer> 1.2.3
|   |-- <AsyncTCP> 1.1.1
|   |-- <FS> 1.0
|   |-- <WiFi> 1.0
|-- <AsyncTCP> 1.1.1
|-- <codecs>
|   |-- <SPI> 1.0
|-- <bluetft>
|   |-- <Adafruit ST7735 and ST7789 Library> 1.9.3
|   |   |-- <Adafruit GFX Library> 1.10.14
|   |   |   |-- <SPI> 1.0
|   |   |   |-- <Adafruit BusIO> 1.11.3
|   |   |   |   |-- <Wire> 1.0.1
|   |   |   |   |-- <SPI> 1.0
|   |   |   |-- <Wire> 1.0.1
|   |   |-- <SPI> 1.0
|-- <dummytft>
|-- <ILI9341>
|   |-- <Adafruit ILI9341> 1.5.10
|   |   |-- <Adafruit GFX Library> 1.10.14
|   |   |   |-- <SPI> 1.0
|   |   |   |-- <Adafruit BusIO> 1.11.3
```

```
|   |   |   |   |-- <Wire> 1.0.1
|   |   |   |   |-- <SPI> 1.0
|   |   |   |   |-- <Wire> 1.0.1
|   |   |   |-- <SPI> 1.0
|-- <LCD1602>
|-- <LCD2004>
|-- <NEXTION>
|-- <OLED>
|-- <ST7789>
|   |-- <Adafruit ST7735 and ST7789 Library> 1.9.3
|   |   |-- <Adafruit GFX Library> 1.10.14
|   |   |   |-- <SPI> 1.0
|   |   |   |-- <Adafruit BusIO> 1.11.3
|   |   |   |   |-- <Wire> 1.0.1
|   |   |   |   |-- <SPI> 1.0
|   |   |   |-- <Wire> 1.0.1
|   |   |-- <SPI> 1.0
|-- <ArduinoOTA> 1.0
|   |-- <Update> 1.0
|   |-- <WiFi> 1.0
|   |-- <ESPmDNS> 1.0
|   |   |-- <WiFi> 1.0
|-- <ESPmDNS> 1.0
|   |-- <WiFi> 1.0
|-- <SPI> 1.0
|-- <SPIFFS> 1.0
|   |-- <FS> 1.0
|-- <WiFi> 1.0
|-- <FS> 1.0
Building in release mode
Compiling .pio\build\esp32\src\main.cpp.o
Linking .pio\build\esp32\firmware.elf
Retrieving maximum program size .pio\build\esp32\firmware.elf
Checking size .pio\build\esp32\firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM:   [==        ]  18.0% (used 58932 bytes from 327680 bytes)
Flash: [========= ]  85.5% (used 1120346 bytes from 1310720 bytes)
Building .pio\build\esp32\firmware.bin
esptool.py v3.1
Merged 1 ELF section
Configuring upload protocol...
AVAILABLE: esp-prog, espota, esptool, iot-bus-jtag, jlink, minimodule, olimex-arm-usb-ocd, olimex-
arm-usb-ocd-h, olimex-arm-usb-tiny-h, olimex-jtag-tiny, tumpa
CURRENT: upload_protocol = esptool
Looking for upload port...
Use manually specified: COM3
Uploading .pio\build\esp32\firmware.bin
esptool.py v3.1
Serial port COM3
Connecting........_____.....______....
Chip is ESP32-D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: cc:50:e3:95:88:90
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Flash will be erased from 0x00001000 to 0x00005fff...
Flash will be erased from 0x00008000 to 0x00008fff...
Flash will be erased from 0x0000e000 to 0x0000ffff...
Flash will be erased from 0x00010000 to 0x00121fff...
Compressed 17104 bytes to 11191...
Writing at 0x00001000... (100 %)
Wrote 17104 bytes (11191 compressed) at 0x00001000 in 0.6 seconds (effective 234.7 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 128...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (128 compressed) at 0x00008000 in 0.1 seconds (effective 394.0 kbit/s)...
Hash of data verified.
Compressed 8192 bytes to 47...
Writing at 0x0000e000... (100 %)
Wrote 8192 bytes (47 compressed) at 0x0000e000 in 0.1 seconds (effective 478.3 kbit/s)...
```

```
Hash of data verified.
Compressed 1120448 bytes to 644211...
Writing at 0x00010000... (2 %)
Writing at 0x0001a3c4... (5 %)
Writing at 0x00027b54... (7 %)
Writing at 0x00032214... (10 %)
..
..
Writing at 0x00113ca2... (95 %)
Writing at 0x00119941... (97 %)
Writing at 0x0011f4c0... (100 %)
Wrote 1120448 bytes (644211 compressed) at 0x00010000 in 15.5 seconds (effective 576.9 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

18.2 Filesystem upload.

```
PS D:\docs\PlatformIO\Projects\ESP32-Radio> pio run -t uploadfs
Processing esp32doit-devkit-v1 (platform: espressif32; board: esp32doit-devkit-v1; framework: arduino)
-------------------------------------------------------------------------------------------------------------------------------
-------------Verbose mode can be enabled via `-v, --verbose` option
CONFIGURATION: https://docs.platformio.org/page/boards/espressif32/esp32doit-devkit-v1.html
PLATFORM: Espressif 32 (3.3.2) > DOIT ESP32 DEVKIT V1
HARDWARE: ESP32 240MHz, 320KB RAM, 4MB Flash
DEBUG: Current (esp-prog) External (esp-prog, iot-bus-jtag, jlink, minimodule, olimex-arm-usb-ocd, olimex-arm-
usb-ocd-h, olimex-arm-usb-tiny-h, olimex-jtag-tiny, tumpa)
PACKAGES:
 - framework-arduinoespressif32 3.10006.210326 (1.0.6)
 - tool-esptoolpy 1.30100.210531 (3.1.0)
 - tool-mkspiffs 2.230.0 (2.30)
 - toolchain-xtensa32 2.50200.97 (5.2.0)
LDF: Library Dependency Finder -> http://bit.ly/configure-pio-ldf
LDF Modes: Finder ~ chain, Compatibility ~ soft
Found 47 compatible libraries
Scanning dependencies...
Dependency Graph
|-- <Wire> 1.0.1
|-- <SD(esp32)> 1.0.5
|   |-- <FS> 1.0
|   |-- <SPI> 1.0
|-- <PubSubClient> 2.8.0
|-- <Adafruit ST7735 and ST7789 Library> 1.7.4
|   |-- <Adafruit GFX Library> 1.10.12
|   |   |-- <SPI> 1.0
|   |   |-- <Adafruit BusIO> 1.9.3
|   |   |   |-- <Wire> 1.0.1
|   |   |   |-- <SPI> 1.0
|   |   |-- <Wire> 1.0.1
|   |-- <SPI> 1.0
|-- <Adafruit BusIO> 1.9.3
|   |-- <Wire> 1.0.1
|   |-- <SPI> 1.0
|-- <Adafruit GFX Library> 1.10.12
|   |-- <SPI> 1.0
|   |-- <Adafruit BusIO> 1.9.3
|   |   |-- <Wire> 1.0.1
|   |   |-- <SPI> 1.0
|   |-- <Wire> 1.0.1
|-- <Adafruit ILI9341> 1.5.9
|   |-- <Adafruit GFX Library> 1.10.12
|   |   |-- <SPI> 1.0
|   |   |-- <Adafruit BusIO> 1.9.3
|   |   |   |-- <Wire> 1.0.1
|   |   |   |-- <SPI> 1.0
|   |   |-- <Wire> 1.0.1
|   |-- <SPI> 1.0
|-- <ESP Async WebServer> 1.2.3
|   |-- <AsyncTCP> 1.1.1
|   |-- <FS> 1.0
|   |-- <WiFi> 1.0
|-- <AsyncTCP> 1.1.1
|-- <codecs>
|   |-- <SPI> 1.0
|-- <bluetft>
|   |-- <Adafruit ST7735 and ST7789 Library> 1.7.4
|   |   |-- <Adafruit GFX Library> 1.10.12
|   |   |   |-- <SPI> 1.0
|   |   |   |-- <Adafruit BusIO> 1.9.3
|   |   |   |   |-- <Wire> 1.0.1
|   |   |   |   |-- <SPI> 1.0
|   |   |   |-- <Wire> 1.0.1
```

```
|   |   |-- <SPI> 1.0
|-- <dummytft>
|-- <ILI9341>
|   |-- <Adafruit ILI9341> 1.5.9
|   |   |-- <Adafruit GFX Library> 1.10.12
|   |   |   |-- <SPI> 1.0
|   |   |   |-- <Adafruit BusIO> 1.9.3
|   |   |   |   |-- <Wire> 1.0.1
|   |   |   |   |-- <SPI> 1.0
|   |   |   |-- <Wire> 1.0.1
|   |   |-- <SPI> 1.0
|-- <LCD1602>
|-- <LCD2004>
|-- <NEXTION>
|-- <ArduinoOTA> 1.0
|   |-- <Update> 1.0
|   |-- <WiFi> 1.0
|   |-- <ESPmDNS> 1.0
|   |   |-- <WiFi> 1.0
|-- <ESPmDNS> 1.0
|   |-- <WiFi> 1.0
|-- <SPI> 1.0
|-- <SPIFFS> 1.0
|   |-- <FS> 1.0
|-- <WiFi> 1.0
Building in release mode
Building SPIFFS image from 'data' directory to .pio\build\esp32doit-devkit-v1\spiffs.bin
/about.html
/config.html
/defaultprefs.txt
/favicon.ico
/index.html
/search.html
/style.css
Looking for upload port...
Use manually specified: COM4
Uploading .pio\build\esp32doit-devkit-v1\spiffs.bin
esptool.py v3.1
Serial port COM4
Connecting....
Chip is ESP32-D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: 24:6f:28:b0:7b:e0
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Flash will be erased from 0x00290000 to 0x003fffff...
Compressed 1507328 bytes to 12150...
Writing at 0x00290000... (100 %)
Wrote 1507328 bytes (12150 compressed) at 0x00290000 in 8.0 seconds (effective 1505.9 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

19 Debug (serial 115200 Baud) output.

This is an example of the debug output. Note that the debug level (CORE_DEBUG_LEVEL in platformio.ini) is set to 5.

```
rst:0xc (SW_CPU_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:1184
load:0x40078000,len:13232
load:0x40080400,len:3028
entry 0x400805e4
[    34][D][esp32-hal-cpu.c:244] setCpuFrequencyMhz(): PLL: 480 / 2 = 240 Mhz, APB: 80000000 Hz

Starting ESP32-radio running on CPU 1 at 240 MHz.
[  3118][I][main.cpp:2447] setup(): [main] Version Mon, 19 Feb 2024 16:30:00 GMT.  Free memory 110580
[  3121][I][main.cpp:2448] setup(): [main] Display type is DUMMY
[  3211][I][main.cpp:2458] setup(): [main] SPIFFS is okay, space 956561, used 32630
[  3215][I][main.cpp:2479] setup(): [main] Found partition 'nvs     ' at offset 0x009000 with size    20480
[  3217][I][main.cpp:2479] setup(): [main] Found partition 'otadata ' at offset 0x00E000 with size     8192
[  3226][I][main.cpp:2479] setup(): [main] Found partition 'spiffs  ' at offset 0x2F0000 with size  1048576
[  3236][I][main.cpp:2479] setup(): [main] Found partition 'coredump' at offset 0x3F0000 with size    65536
[  3253][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::clk_dst type=33
[  3253][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::clk_offset type=33
[  3259][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::clk_server type=33
[  3267][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::gpio_00 type=33
[  3273][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::gpio_12 type=33
[  3280][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::gpio_13 type=33
[  3287][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::gpio_14 type=33
[  3294][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::gpio_17 type=33
[  3301][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::ir_40BF type=33
[  3308][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::ir_C03F type=33
[  3315][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::mqttbroker type=33
[  3326][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::mqttpasswd type=33
[  3329][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::mqttport type=33
[  3336][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::mqttprefix type=33
[  3343][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::mqttuser type=33
[  3350][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::pin_i2s_bck type=33
[  3357][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::pin_i2s_din type=33
[  3364][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::pin_i2s_lck type=33
[  3371][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::pin_i2s_spdif type=33
[  3379][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::preset_00 type=33
[  3386][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::preset_01 type=33
[  3393][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::preset_02 type=33
[  3400][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::preset_03 type=33
[  3407][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::preset_04 type=33
[  3414][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::preset_05 type=33
[  3421][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::preset_06 type=33
[  3428][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::preset_07 type=33
[  3435][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::preset_08 type=33
[  3442][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::preset_09 type=33
[  3449][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::toneha type=33
[  3456][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::tonehf type=33
[  3466][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::tonela type=33
[  3469][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::tonelf type=33
[  3476][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::wifi_00 type=33
[  3483][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::wifi_01 type=33
[  3490][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::volume type=33
[  3497][I][main.cpp:2337] fillkeylist(): [main] ESP32-Radio::preset type=33
[  3503][I][main.cpp:2350] fillkeylist(): [main] Read 37 keys from NVS
[  3514][I][main.cpp:1851] readIOprefs(): [main] 'pin_i2s_spdif' set to 15
[  3516][I][main.cpp:1851] readIOprefs(): [main] 'pin_spi_sck  ' set to 18
[  3523][I][main.cpp:1851] readIOprefs(): [main] 'pin_spi_miso ' set to 19
[  3529][I][main.cpp:1851] readIOprefs(): [main] 'pin_spi_mosi ' set to 23
[  3546][I][main.cpp:2519] setup(): [main] GPIO0 is HIGH
[  3556][I][main.cpp:2519] setup(): [main] GPIO2 is LOW, probably no PULL-UP
[  3566][I][main.cpp:2519] setup(): [main] GPIO4 is HIGH
[  3576][I][main.cpp:2519] setup(): [main] GPIO5 is HIGH
[  3586][I][main.cpp:2519] setup(): [main] GPIO12 is HIGH
[  3596][I][main.cpp:2519] setup(): [main] GPIO13 is HIGH
[  3606][I][main.cpp:2519] setup(): [main] GPIO14 is HIGH
[  3616][I][main.cpp:2519] setup(): [main] GPIO15 is HIGH
[  3626][I][main.cpp:2519] setup(): [main] GPIO16 is HIGH
[  3636][I][main.cpp:2519] setup(): [main] GPIO17 is HIGH
[  3646][I][main.cpp:2519] setup(): [main] GPIO18 is HIGH
[  3656][I][main.cpp:2519] setup(): [main] GPIO19 is HIGH
[  3666][I][main.cpp:2519] setup(): [main] GPIO21 is HIGH
[  3676][I][main.cpp:2519] setup(): [main] GPIO22 is HIGH
[  3686][I][main.cpp:2519] setup(): [main] GPIO23 is HIGH
[  3696][I][main.cpp:2519] setup(): [main] GPIO25 is HIGH
[  3706][I][main.cpp:2519] setup(): [main] GPIO26 is HIGH
[  3716][I][main.cpp:2519] setup(): [main] GPIO27 is HIGH
[  3726][I][main.cpp:2519] setup(): [main] GPIO32 is HIGH
[  3736][I][main.cpp:2519] setup(): [main] GPIO33 is HIGH
[  3746][I][main.cpp:2519] setup(): [main] GPIO34 is LOW, probably no PULL-UP
[  3756][I][main.cpp:2519] setup(): [main] GPIO35 is HIGH
[  3766][I][main.cpp:2519] setup(): [main] GPIO39 is LOW, probably no PULL-UP
[  3767][I][main.cpp:1687] readprogbuttons(): [main] gpio_00 will execute uppreset = 1
[  3771][I][main.cpp:1687] readprogbuttons(): [main] gpio_12 will execute upvolume = 2
[  3778][I][main.cpp:1687] readprogbuttons(): [main] gpio_13 will execute downvolume = 2
[  3786][I][main.cpp:1687] readprogbuttons(): [main] gpio_14 will execute stop
[  3793][I][main.cpp:1687] readprogbuttons(): [main] gpio_17 will execute resume
[  3801][I][main.cpp:2536] setup(): [main] Start DUMMY display
[  3805][E][main.cpp:2553] setup(): [main] Display not activated
[  3811][I][main.cpp:2219] mk_lsan(): [main] Create list with acceptable WiFi networks
[  3819][I][WiFiMulti.cpp:84] addAP(): [WIFI][APlistAdd] add SSID: ADSL-11
[  3826][I][WiFiMulti.cpp:84] addAP(): [WIFI][APlistAdd] add SSID: Nokia-basis-11
[  4365][D][WiFiGeneric.cpp:1035] _eventCallback(): Arduino Event: 0 - WIFI_READY
[  4461][V][WiFiGeneric.cpp:340] _arduino_event_cb(): STA Started
[  4462][D][WiFiGeneric.cpp:1035] _eventCallback(): Arduino Event: 2 - STA_START
[  4979][I][main.cpp:894] nextPreset(): [main] nextPreset is 0
[  4994][I][main.cpp:2580] setup(): [main] Connect to network
[  4996][V][WiFiGeneric.cpp:343] _arduino_event_cb(): STA Stopped
[  4997][D][WiFiGeneric.cpp:1035] _eventCallback(): Arduino Event: 3 - STA_STOP
[  6023][D][WiFiGeneric.cpp:1035] _eventCallback(): Arduino Event: 0 - WIFI_READY
[  6027][V][WiFiGeneric.cpp:340] _arduino_event_cb(): STA Started
[  6027][D][WiFiGeneric.cpp:1035] _eventCallback(): Arduino Event: 2 - STA_START
[ 13441][V][WiFiGeneric.cpp:383] _arduino_event_cb(): SCAN Done: ID: 128, Status: 0, Results: 8
[ 13442][D][WiFiGeneric.cpp:1035] _eventCallback(): Arduino Event: 1 - SCAN_DONE
[ 13446][I][WiFiMulti.cpp:114] run(): [WIFI] scan done
```

```
[ 13451][I][WiFiMulti.cpp:119] run(): [WIFI] 8 networks found
[ 13456][D][WiFiMulti.cpp:151] run():        0: [10][28:C6:8E:12:C4:1E] NETGEAR-11 (-58) *
[ 13464][D][WiFiMulti.cpp:149] run():  --->   1: [3][C0:06:C3:4E:29:56] ADSL-11 (-59) *
[ 13472][D][WiFiMulti.cpp:149] run():  --->   2: [3][C0:06:C3:4C:95:A2] ADSL-11 (-59) *
[ 13480][D][WiFiMulti.cpp:151] run():        3: [3][CA:06:C3:4E:29:56] ADSL-11-IOT (-59) *
[ 13488][D][WiFiMulti.cpp:151] run():        4: [3][CE:06:C3:4C:95:A2] ADSL-11-IOT (-59) *
[ 13496][D][WiFiMulti.cpp:149] run():  --->   5: [6][E4:8E:10:0A:4F:4D] Nokia-basis-11 (-76) *
[ 13504][D][WiFiMulti.cpp:151] run():        6: [1][2A:35:D1:17:63:99] Ziggo (-88) *
[ 13511][D][WiFiMulti.cpp:151] run():        7: [1][18:35:D1:17:63:99] Ziggo8571367 (-91) *
[ 13520][I][WiFiMulti.cpp:160] run(): [WIFI] Connecting BSSID: C0:06:C3:4E:29:56 SSID: ADSL-11 Channel: 3 (-59)
[ 13530][V][WiFiGeneric.cpp:97] set_esp_interface_ip(): Configuring Station static IP: 0.0.0.0, MASK: 0.0.0.0, GW: 0.0.0.0
[ 14169][V][WiFiGeneric.cpp:355] _arduino_event_cb(): STA Connected: SSID: ADSL-11, BSSID: c0:06:c3:4e:29:56, Channel: 3, Auth: WPA2_PSK
[ 14171][D][WiFiGeneric.cpp:1035] _eventCallback(): Arduino Event: 4 - STA_CONNECTED
[ 14939][V][WiFiGeneric.cpp:369] _arduino_event_cb(): STA Got New IP:192.168.1.87
[ 14940][D][WiFiGeneric.cpp:1035] _eventCallback(): Arduino Event: 7 - STA_GOT_IP
[ 14943][D][WiFiGeneric.cpp:1098] _eventCallback(): STA IP: 192.168.1.87, MASK: 255.255.255.0, GW: 192.168.1.254
[ 14953][I][WiFiMulti.cpp:174] run(): [WIFI] Connecting done.
[ 14958][D][WiFiMulti.cpp:175] run(): [WIFI] SSID: ADSL-11
[ 14964][D][WiFiMulti.cpp:176] run(): [WIFI] IP: 192.168.1.87
[ 14969][D][WiFiMulti.cpp:177] run(): [WIFI] MAC: C0:06:C3:4E:29:56
[ 14975][D][WiFiMulti.cpp:178] run(): [WIFI] Channel: 3
[ 14980][I][main.cpp:1568] connectwifi(): [main] SSID = ADSL-11
[ 14986][I][main.cpp:1572] connectwifi(): [main] IP = 192.168.1.87
[ 14992][I][main.cpp:2590] setup(): [main] Start web server
[ 14998][I][main.cpp:2601] setup(): [main] Network found. Starting clients
[ 15009][I][main.cpp:2635] setup(): [main] MDNS responder started
[ 16012][I][main.cpp:2680] setup(): [main] Rotary encoder is disabled (-1/-1/-1)
[ 16013][I][main.cpp:4249] gettime(): [main] Sync TOD
[ 16014][I][main.cpp:4262] gettime(): [main] TOD synced
[ 16019][I][main.cpp:4404] playtask(): [main] Starting I2S playtask..
[ 16027][I][main.cpp:4437] playtask(): [main] Output to SPDIF, pin 15
[ 16122][I][main.cpp:894] nextPreset(): [main] nextPreset is 1
[ 16123][I][main.cpp:3283] radiofuncs(): [main] Radiofuncs cmd is 1
[ 16123][I][main.cpp:4480] playtask(): [main] Playtask stop song
[ 16125][I][main.cpp:1329] connecttohost(): [main] Connect to host airspectrum.cdnstream1.com:8114/1648_128
[ 16369][I][main.cpp:2405] onConnect(): [main] Connected to host at 144.217.252.222 on port 8114
[ 16440][I][main.cpp:1390] connecttohost(): [main] send GET command
[ 16445][I][helixfuncs.h:85] player_setVolume(): [helixfuncs] Volume set to 38
[ 16576][I][main.cpp:3532] handlebyte_ch(): [main] Switch to HEADER
[ 16576][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: Content-Type: audio/mpeg
[ 16579][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: icy-br:128
[ 16585][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: ice-audio-info: ice-samplerate=44100;ice-bitrate=128;ice-channels=2
[ 16596][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: icy-br:128
[ 16603][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: icy-description:Today's Easy Hits from the 70s, 80s, 90s through Today!
[ 16615][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: icy-genre:Easy Listening
[ 16623][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: icy-name:Easy Hits Florida
[ 16631][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: icy-private:0
[ 16637][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: icy-pub:1
[ 16644][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: icy-url:http://www.easyhitsflorida.com
[ 16653][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: Server: Icecast 2.4.0-kh12
[ 16660][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: Cache-Control: no-cache, no-store
[ 16669][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: Access-Control-Allow-Origin: *
[ 16677][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: Access-Control-Allow-Headers: Origin, Accept, X-Requested-With, Content-Type
[ 16690][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: Access-Control-Allow-Methods: GET, OPTIONS, HEAD
[ 16699][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: Connection: Close
[ 16706][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: Expires: Mon, 26 Jul 1997 05:00:00 GMT
[ 16715][I][main.cpp:3553] handlebyte_ch(): [main] Headerline: icy-metaint:16000
[ 16722][I][main.cpp:3623] handlebyte_ch(): [main] Switch to DATA, bitrate is 128 kbps, metaint is 16000
[ 16731][I][main.cpp:4472] playtask(): [main] Playtask start song
[ 16737][I][helixfuncs.h:114] helixInit(): [helixfuncs] helixInit called for audio/mpeg
[ 16745][I][helixfuncs.h:233] playChunk(): [helixfuncs] Sync found at 0x0000
[ 16759][I][helixfuncs.h:286] playChunk(): [helixfuncs] Bitrate     is 128000
[ 16759][I][helixfuncs.h:287] playChunk(): [helixfuncs] Samprate    is 44100
[ 16765][I][helixfuncs.h:288] playChunk(): [helixfuncs] Channels    is 2
[ 16772][I][helixfuncs.h:289] playChunk(): [helixfuncs] Bitpersamp  is 16
[ 16778][I][helixfuncs.h:290] playChunk(): [helixfuncs] Outputsamps is 2304
[ 17097][I][main.cpp:1223] showstreamtitle(): [main] Streamtitle found, 34 bytes
[ 17097][I][main.cpp:1224] showstreamtitle(): [main] StreamTitle='Santana -  Foo foo ';
```

20 Quickstart.

Assuming that PlatformIO is installed on your PC, perform the following steps:

1. Download the zipfile from github in any directory and unpack it.
2. Double-click on “ESP-Radio.code-workspace”. PlatformIO will open.
3. Open the “config.h” file in “include” and edit the values for “FIXEDWIFI”, the type of the decoder and the display. FIXEDWIFI should contain the SSID and the PASSWORD of your WiFi network in the form “SSID/PASSWD”.
4. Open “platformio.ini” and edit the values of “upload_port” and “monitor_port”, so it will contain the right serial port.
5. Compile and upload the program and the contents of the data-directory to the ESP32. In the platformIO pane  you may choose for “Upload Filesystem Image” and after that “Upload and monitor”.
6. Observe the output to the serial monitor and remember the line with the IP address of the radio (“...IP = xxxxx”).
7. Use your browser to show the webinterface of the radio using this IP address.
8. In the web interface, go to the “Config” page.
9. Click on the “Default” button.
10. Edit the preferences, depending on your configuration.
11. Click on the “Save” button and then on the “Restart” button.