

RadioESP32S3

Introduction

First of all, it is worth mentioning that this project would probably never have been created without the popular project "ESP32Radio-V2" by Edzelf. I am very grateful to the author for it and thank him for his work. Originally, of course, I took over the original project in its unchanged form, but very soon it stopped being convenient for me. I especially disliked the fact that the display did not display characters with diacritics.

The second biggest problem for me was the functioning of the file player from the SD card. One of my requirements for the device was that it should allow listening to audiobooks. In this case, however, the files need to be played in a defined (i.e. alphabetical) order. But that did not work with the original SW. After copying the files from the HDD to the SD card, the files were always arranged in a kind of random order and listening to the individual parts in the correct order was very complicated (the correct part had to be selected manually).

In addition to the two problems mentioned above, there were a number of other small things that I imagined to work differently. For example, I can mention a function that ensured that the last station listened to was always played after switching on (and with the last volume level used). But that did not suit me at all, I prefer that a default station (with the default volume) be played after switching on the device. I also planned to add some new functions that the original design did not have. In particular, I wanted the radio to be able to be turned on and off by pressing one (same) button and for the radio to have a sleep function (i.e. automatic shutdown after a set time).

Despite the above facts, I took over several things from the original "ESP32Radio-V2" project almost unchanged and my work was thus made easier.

The website design, the way the ESP32-S3 communicates with the computer (via a web browser), and a few other solutions (like how to upgrade or work with the configuration file) are taken from the **esp-rfid** project. I have used it in several projects and it exactly suits my needs and ideas.

Development with VS Code and PlatformIO

I use VS Code/PlatformIO as a development environment. Using the `platformio.ini` file, it is possible to create several software variants by specifying or not specifying the so-called "build-flag". The DATAWEB, SDCARD, BATTERY, AUTOSHUTDOWN and OTA build-flags are particularly important.

- Using the DATAWEB build-flag, the SW is compiled in a version that has the web server files stored in the "LittleFS" file system area, while without the DATAWEB parameter, the SW is compiled in a version in which the web server files are stored in the program memory (using the PROGMEM attribute) in the form of variables. I call the first version „data“ for short, the second „webh“. The **data** variant is intended primarily for the development phase (but nothing prevents it from being used as a production one), while the **webh** variant is intended exclusively as a production one and it is more or less impossible to develop with it.
- Using the SDCARD build-flag, support for the SD card file player function is enabled. Without using the aforementioned build-flag, the device is only able to play radio stations.
- Using the BATTERY build-flag, support for the supply voltage measurement function is enabled. This is of course suitable in the case of battery power. The display then shows the charge level.
- Using the AUTOSHUTDOWN build-flag, the support of the function that allows the device to be turned on and off with one button and also the sleep function (adjustable automatic shutdown time) is enabled. Of course, the condition is that the HW is equipped with the necessary circuit.
- Using the OTA build-flag, the function that allows a comfortable upgrade from the web interface is enabled. See the Upgrade chapter.

4 types of "envs" are also prepared in the `platformio.ini` file. With them, you can select both the SW variant (development, i.e. **data** – without OTA and production, i.e. **webh** – including OTA) and the variant with or without display. So, 4 combinations in total (noota, withota, nootanodisp and withotanodisp). The relevant "env" version is selected by commenting/uncommenting in the `[platformio]` section.

Using the DATAWEB build-flag, I proceed as follows when developing the SW:

- 1) Development takes place exclusively in the software **data** variant (the DATAWEB build-flag is used in the `platformio.ini` file - „env“ is either `noota` or `nootanodisp`). After editing the source files, compilation is performed in the usual way and the device is upgraded via the USB (serial) port.
- 2) If it is necessary to edit the *.html or *.js files of the web server to fix or add a function, you need to open the page <http://<IP address of the device>/edit>. For this to work, the PC must be connected to the Internet. The online library "ace" (Ajax.org Cloud9 Editor) is used for editing. For more information, see <https://ace.c9.io/>.
- 3) Once the required functionality has been achieved using steps 1 and 2, the **webh** variant of the software can be created. This is done by running the user script **Download FS & Create WEBH**, which is available via the **PROJECT TASKS/<selected env version>/Custom** menu. This will transform the web server files (from the LittleFS file system) into **webh** files. The utility "pio-esp32-esp8266-filesystem-downloader" by **maxgerhardt** is used to download the file system image and extract it.
- 4) Switch to **webh** variant (in the `platformio.ini` file, the build-flag DATAWEB needs to be removed [commented out] - „env“ is either `withota` or `withotanodisp`) and compile. The resulting image "*firmware_WEBH_X.X.X.bin*" (created during the compilation process) can be found in the `/bin/<selected env version>` directory. The version designation "*X.X.X*" is also set as a "build-flag" in the `platformio.ini` file before compilation.

Upgrade

A device running the **webh** firmware variant can be upgraded very easily via the web interface.
Procedure:

- a) Select the **Update** item in the menu (on the left side of the screen).
- b) A window will appear, showing the current version approximately in the middle.
- c) Use the "Select file" button to search for the firmware file on your computer (e.g. *firmware_WEBH_1.0.0.bin*) and start the process with the **Update** button.

If everything goes well, the device will already contain the new software version after the restart.

First start

When the device is started for the first time, it runs in default settings and in AP mode. On the PC, connect to the "RadioESP32S3" network and in the browser to the address 192.168.4.1. A web page should appear through which the device settings need to be made.

Main menu

There is always the main menu on the left side of the screen. Description of the main menu items:

Radio Player

After clicking on this item, a page appears with which you can comfortably control the radio.

SD Card Player

This item is only displayed if the SW is compiled with the SDCARD option.

After clicking on this item, a page appears with which you can comfortably control the functions of the SD Card Player.

Status

Clicking on this item will bring up a page with basic device information.

Settings

If you need to adjust the device settings (for example, when you first start it), select Settings. Clicking on it will expand the menu and you will see eight submenus:

Radio Settings

This page is used to create a radio station menu. You can also set the default station (it is always automatically selected when the device is turned on), the default volume, and the values of a simple equalizer.

SD Card Settings

This item is only displayed if the SW is compiled with the SDCARD option. Only one parameter can be set on this page, and that is "Seek step". This is the length of the jump (forward and backward) in seconds. The default volume and equalizer settings on the **Radio Settings** page also apply to the SD card file player.

Network Settings

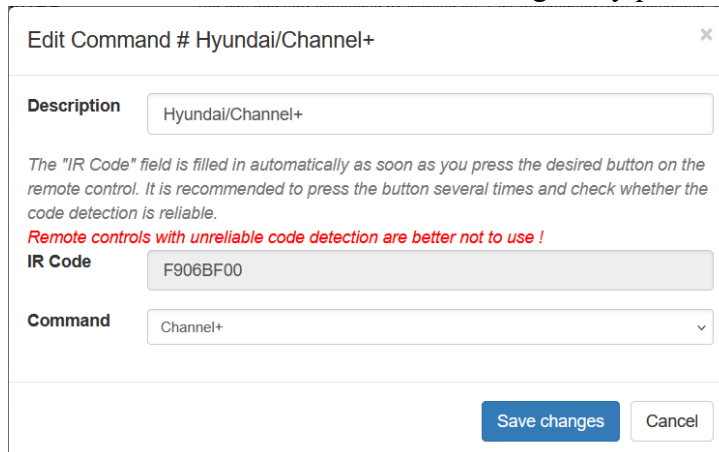
On this page, you need to set the network (or multiple networks) to which the device is to connect. Click [Edit Networks](#) and then [New Network](#). In the dialog box that appears, set the required values. You can use the Scan button to search for available networks. After setting, click [Save Changes](#). Under the network table, you can also change the SSID, IP address and mask (all in AP mode). You need to confirm the changes by clicking the [Save](#) button and then the yellow button with the text **You have uncommitted changes, please click here to commit and reboot ...**. However, this yellow button should only be used after going through all other settings, because after using it, the device will restart.

Hardware Settings

On this page it is possible to set some used GPIO pins (some are set before compilation in `platformio.ini`) and several other parameters. The set values must be confirmed again by pressing the [Save](#) button.

IR Remote Settings

This page is where you configure the remote control commands. Adding a new command is very simple and intuitive. The set commands must be confirmed again by pressing the [Save](#) button.



Dialog box titled "Edit Command # Hyundai/Channel+" with a close button (X).

Description: Hyundai/Channel+

The "IR Code" field is filled in automatically as soon as you press the desired button on the remote control. It is recommended to press the button several times and check whether the code detection is reliable.

Remote controls with unreliable code detection are better not to use !

IR Code: F906BF00

Command: Channel+

Buttons: Save changes, Cancel

Display Settings

On this page, it is possible to set, in particular, the display backlight levels and the clock display settings in idle mode. The user can also name the days of the week in their own language (the names are English by default). As always, after all changes, the [Save](#) button must be pressed.

General Settings

On this page you can set the "Host Name" and also calibrate the battery level display. Confirm by pressing the [Save](#) button again.

NTP (Time) Settings

On this page you can set the NTP server, synchronization interval (in minutes) and time zone. Changes are confirmed again by pressing the [Save](#) button.

Sometimes it may happen that at the time of switching on (or restarting) the device none of the known WiFi networks are available. Then of course it is not possible to synchronize the time with the NTP server. In such an extreme situation the [Sync Browser Time to Device](#) button will help.

Backup & Restore

In the main menu we will also find the item “Backup & Restore”. The meaning is obvious from the name. It is possible to save a file with a backup of the settings to the PC and, if necessary, easily return to the same settings. Under the mentioned item there is another function – reboot the device.

Factory reset

This main menu item hides a function of the same name. After activating it, the configuration file is deleted and after rebooting the device is in the same state as when it was first started.

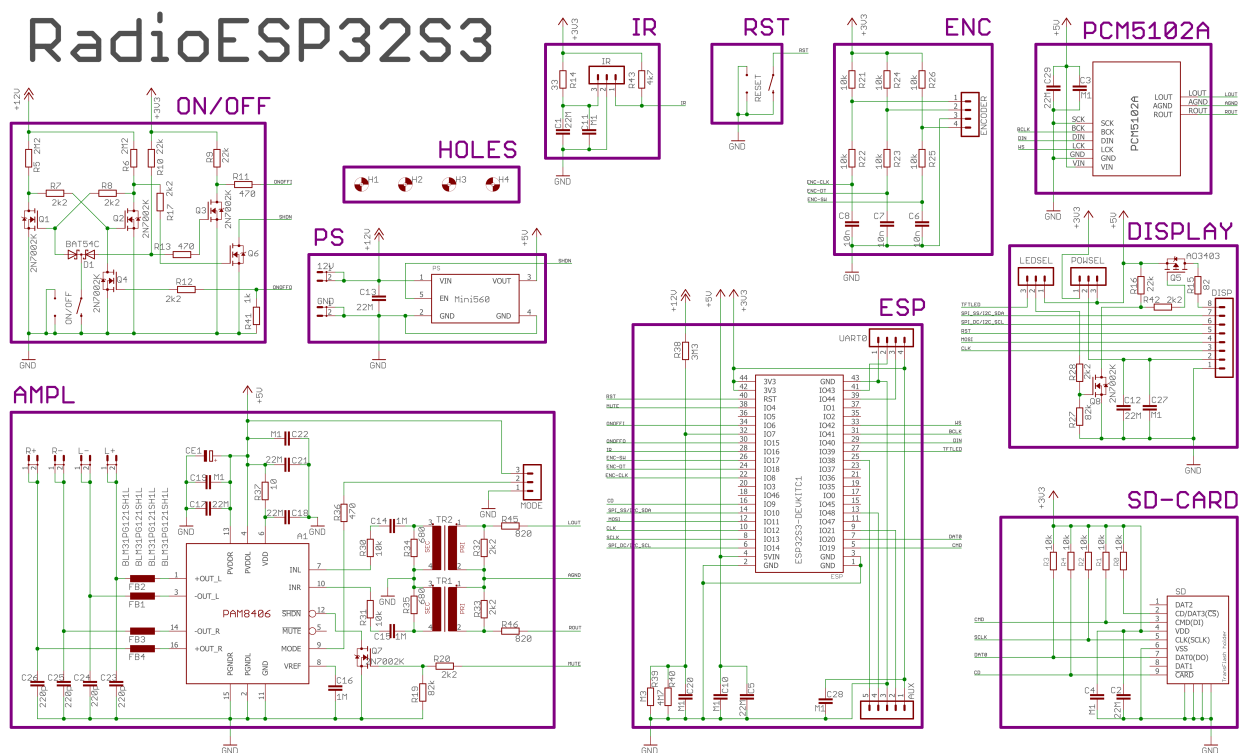
Update

This item is displayed only if the SW was compiled with the OTA build-flag, i.e. in the case of "envs" withota and withotanodisp. The procedure is described above in the Upgrade paragraph.

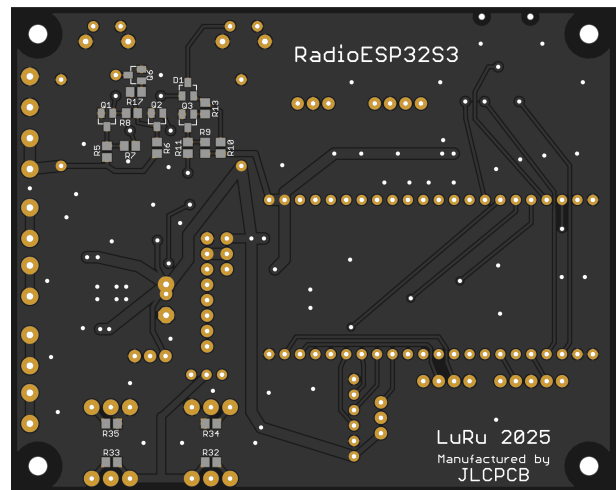
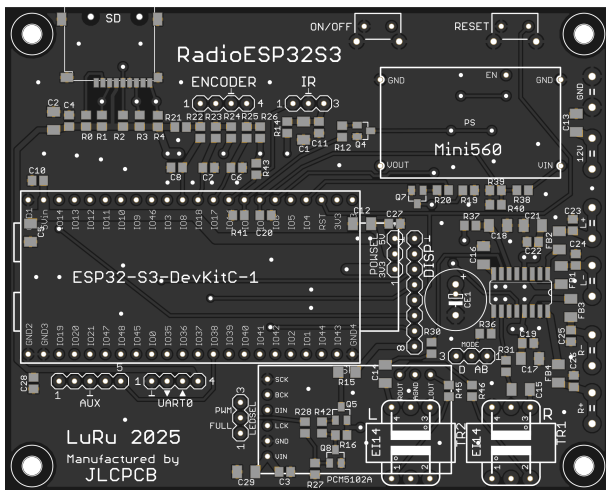
Device design

The circuit diagram and printed circuit board were created using the Eagle design system.

Circuit diagram

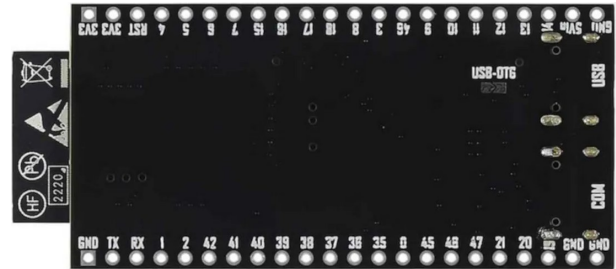
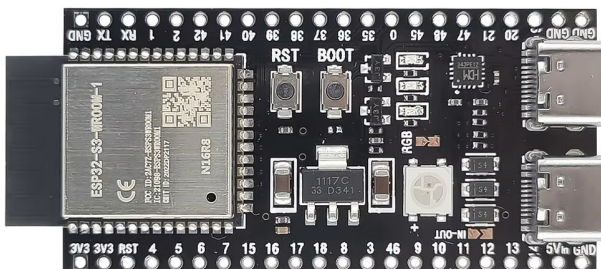


Printed circuit board

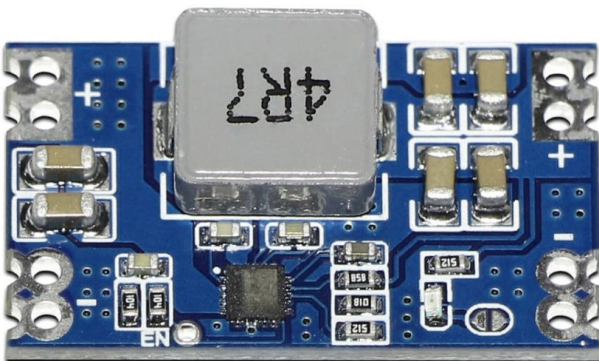


Component selection

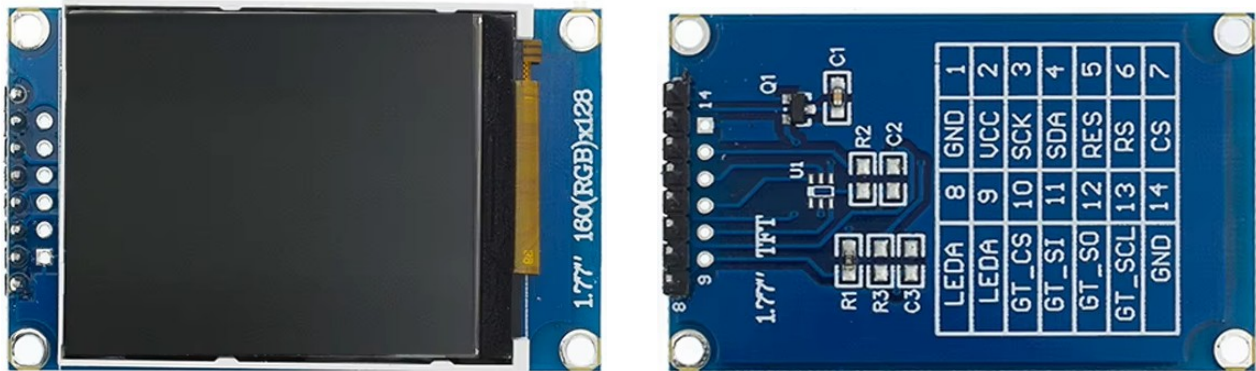
In the first phase of development, I used the VS1053 module (as a DAC) following Edzelf's example. It worked well, but I found this solution unnecessarily expensive and quite problematic (for example, I once accidentally bought a VS1003 instead and it didn't work well). That's why I completely abandoned this path and switched exclusively to a solution with I2S. As the basis for the SW, I chose the excellent "ESP32-audioI2S" library from the author **shreibfaul1**. This led to the need to use a type of ESP32-S3 development kit that is equipped with PSRAM memory (without this memory, no more complex project can be created with the "ESP32-audioI2S" library, you will soon run out of RAM) and flash memory of at least 8MB (to be able to upgrade via OTA). For this reason, I chose a development kit of the **ESP32-S3-DevKitC-1** type.



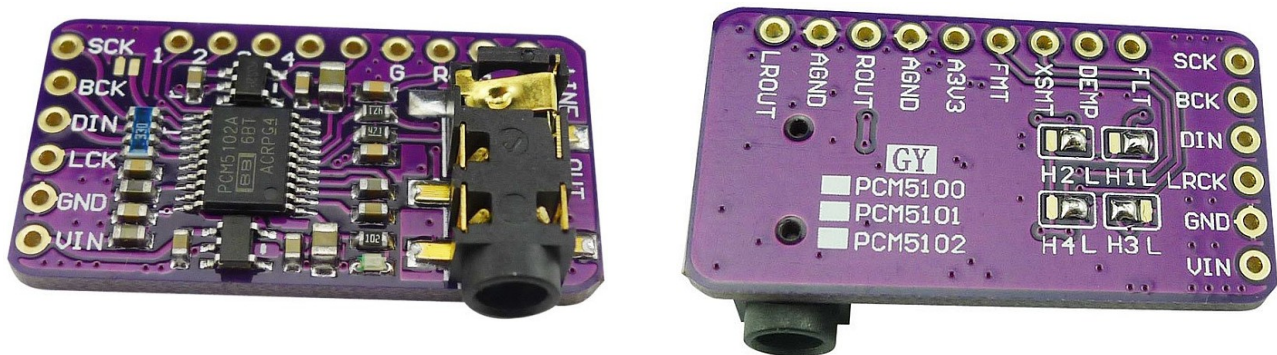
I decided that the basic supply voltage of the device will be 12V, because it can be very well implemented with three 18650 Li-ion cells. However, the development kit requires a voltage of 5V (as well as some other components) and therefore it is necessary to use a suitable step-down module. I chose the Mini560 module, which is sufficiently dimensioned (both voltage and current) and is equipped with an EN input (necessary for my needs).



Originally (like Edzelf) I kept the option to choose between OLED and TFT display when compiling the software. But I finally gave up and OLED displays are no longer supported at all. So there are only two options - either a TFT display or none. I used a display with a diagonal of 1.77 inches and a resolution of 160x128. However, it should not be a problem to use any other one, supported by the TFT_eSPI library.



The essential part of the device is the digital-to-analog converter. I chose a module with the PCM5102A chip, which suited me due to its dimensions and also because the audio output is not only connected to a jack connector, but also to a pin header. This greatly simplifies the connection of the converter to the motherboard.



As an amplifier, I chose the PAM8406 type, which is powerful enough, works with a 5V supply voltage, and can be easily soldered by hand. This amplifier has one more advantage - it can operate in either class D or AB. I have kept the option of choosing on the printed circuit board as well - the mode is selected by placing the appropriate jumper.



It is necessary to include isolation transformers between the DAC and the amplifier to break the ground loop and prevent unpleasant interference. I chose transformers that are only 9.1mm high and fit well between the converter board and the motherboard.



An important part of the device is also a rotary encoder. I used a completely common and cheap type EC11.

An obvious requirement for the device's function was also the possibility of control via infrared remote control. I tried the VS1838B type as a receiver and it worked well.

Finally, it was necessary to solve the method of connecting the supply voltage and speakers to the motherboard. I wanted to avoid terminal blocks with screws and therefore I chose faston connectors with a width of 4.8mm.

